keyestudio WiKi

keyestudio WiKi

Dec 06, 2023

KEYESTUDIO DOCS

1 1. Description 3

2 2. Kit list 5

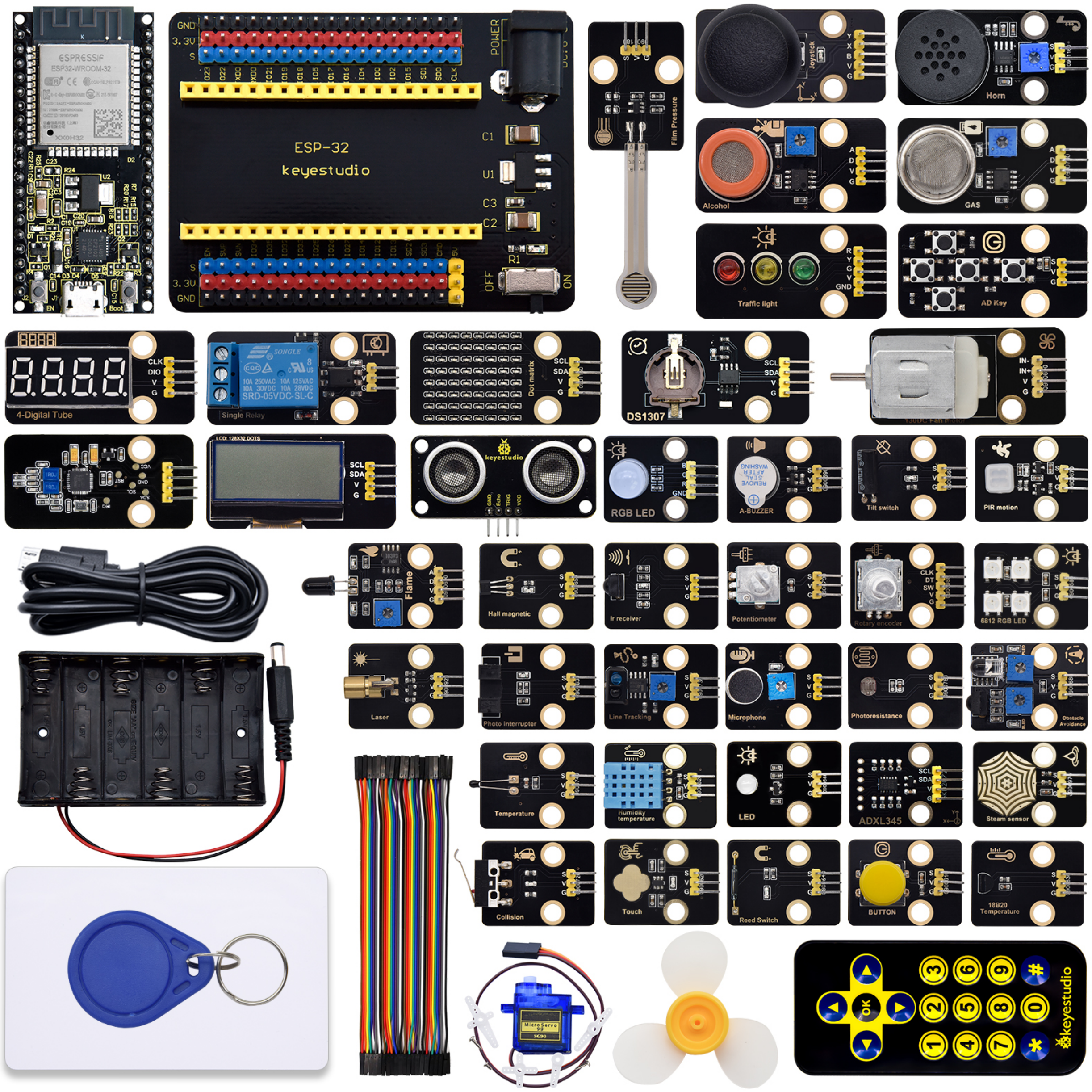
3 3. Tutorials 13

4 ESP32 Mainboard and ESP32 shield 15
- 4.1 1. Keyestudio ESP32 Mainboard . . . 15
- 4.2 2. Keyestudio ESP32-IO shield . . . 19

5 Python tutorial 21
- 5.1 1. Preparation for Python(Windows): . . . 21
- 5.2 2. Single Sensor/Experiment Projects . . . 57
- 5.3 3. Comprehensive Experiments: . . . 205

6 Arduino tutorial 283
- 6.1 1. Get started with Arduino C: . . . 283
- 6.2 2. Basic Projects . . . 324
- 6.3 3. Comprehensive Experiments: . . . 481

7 Arduino(Raspberry-Pi) tutorial 571
- 7.1 1. Install Raspberry Pi OS System . . . 571
- 7.2 2. Preparations for C language: . . . 612
- 7.3 3. Linux SystemRaspberry Pi: . . . 617
- 7.4 4. How to Add Libraries? : . . . 634
- 7.5 5. Basic Projects . . . 641
- 7.6 6. Comprehensive Projects: . . . 795


ESP-32
keyestudio
Traffic light
AD Key
4-Digital Tube
DS1307
RGB LED
ADXL345
LED
BUTTON

CHAPTER

ONE

1. DESCRIPTION

The Keyestudio ESP32 42 in 1 sensor kit mainly contains 42 commonly usedsensors/modules, a ESP32 board, a ESP32 expansion board and Dupontwires.

The 42 sensors and modules are fully compatible with the ESP32 ExpansionBoard. You only need to stack the ESP32 mainboard onto the ESP32 Expansion Board, and hook up them with Dupont wires, which is simple and convenient.

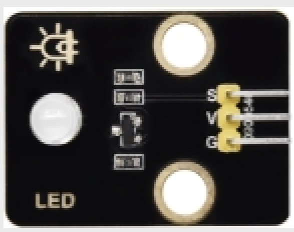
To make you master the electronic knowledge, detailed tutorials (Micropython), schematic diagrams, wiring methods and test code are included. Through these projects, you will have a better understanding about programming, logic and electronics.

CHAPTER

TWO

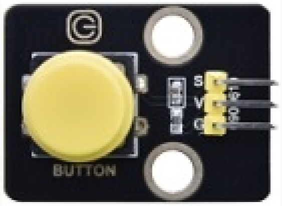
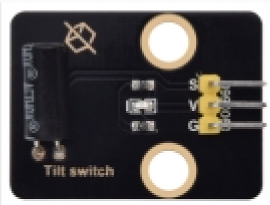
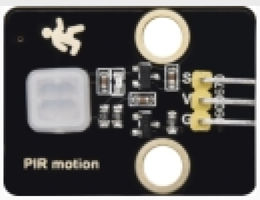
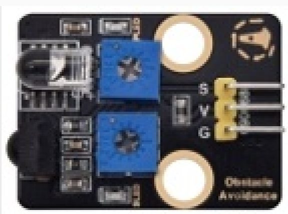
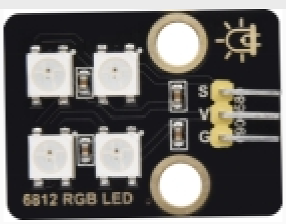
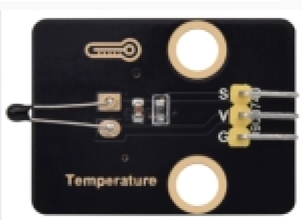
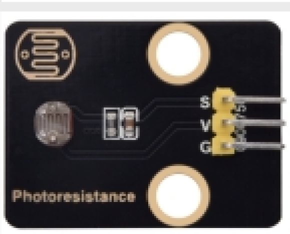
2. KIT LIST

(Note: KS5003 kit include ESP32 mainboardKS5004 kit does‘t include ESP32 mainboard.)

#	Picture	Name	QTY
1		keyestudio LED Module	1
2		Keyestudio Common Cathode RGB Module	1
3		Keyestudio Traffic Lights Module	1
4		Keyestudio Active Buzzer	1
5		Keyestudio 8002b Audio Power Amplifier	1

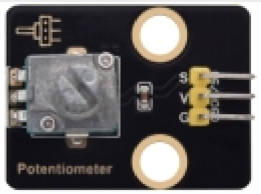
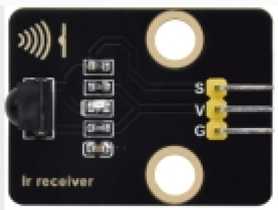
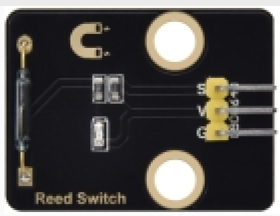
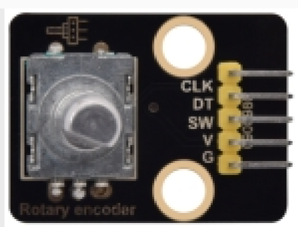
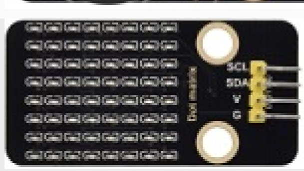
continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
6		Keyestudio Button Module	1
7		Keyestudio Tilt Sensor	1
8		Keyestudio PIR Motion Sensor	1
9		Keyestudio Obstacle Avoidance Sensor	1
10		Keyestudio 6812 RGB Module	1
11		Keyestudio NTC-MF52AT Thermistor	1
12		Keyestudio Photoresistor	1

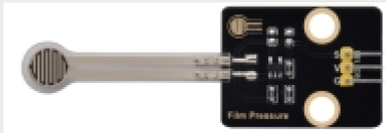
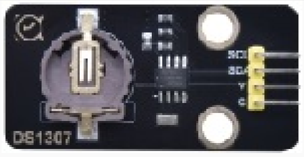
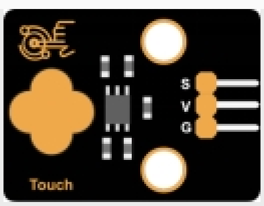
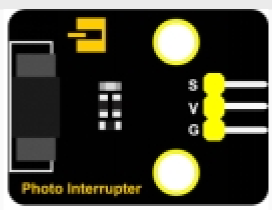
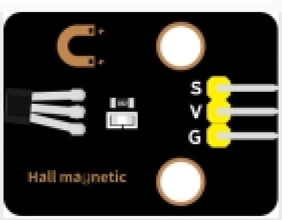
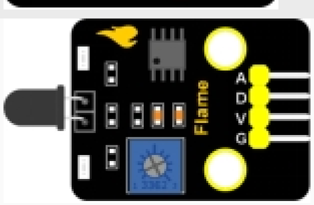
continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
13		Keyestudio Sound Sensor	1
14		KeyestudioRotary Potentiometer	1
15		Keyestudio IR Receiver	1
16		Keyestudio Reed Switch Sensor	1
17		Keyestudio Rotary Encoder Module	1
18		Keyestudio Joystick Module	1
19		Keyestudio HT16K33 8X8 Dot Matrix Module	1
20		Keyestudio TM1650 4-Digit Tube Display	1

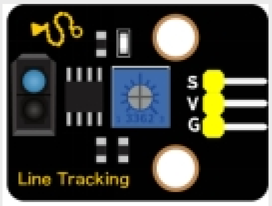
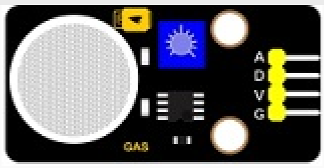
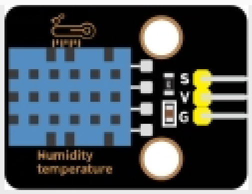
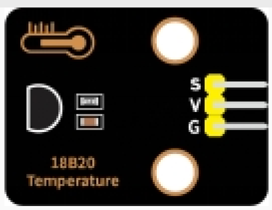
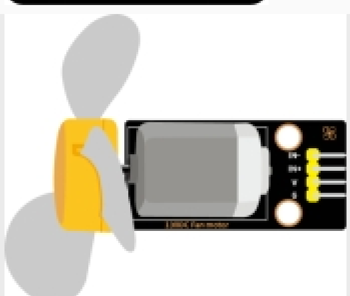
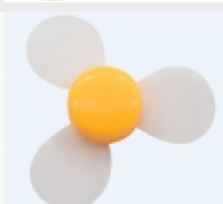
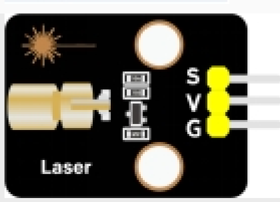
continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
21		Keyestudio Thin-film Pressure Sensor	1
22		Keyestudio DS1307 Clock Sensor	1
23		Keyestudio SR01 Ultrasonic Sensor	1
24		Servo	1
25		Keyestudio Capacitive Sensor	1
26		Keyestudio Photo Interrupter	1
27		Keyestudio Hall Sensor	1
28		Keyestudio Flame Sensor	1

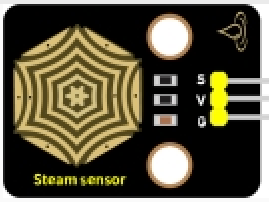
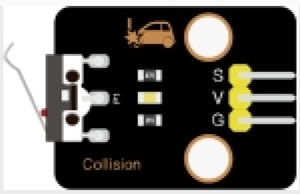
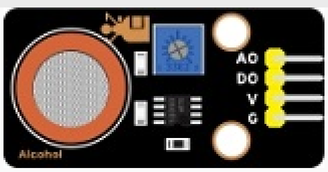
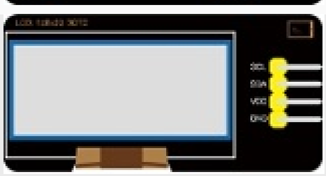
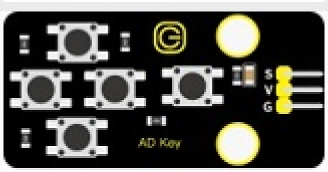
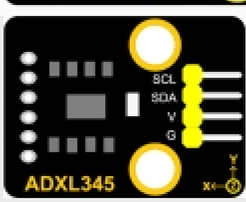
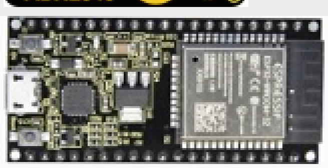
continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
29		Keyestudio Line Tracking Sensor	1
30		Keyestudio Analog Gas Sensor	1
31		Temperature and Humidity Sensor	1
32		Keyestudio 18B20 Temperature Sensor	1
33		keyestudio 130 Motor	1
34		Fan	1
35		Keyestudio Laser Module	1

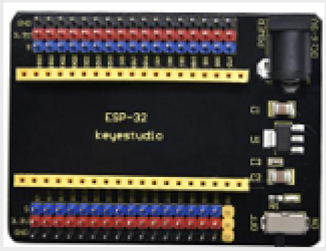
continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
36		Keyestudio Steam Sensor	1
37		Keyestudio Relay Module	1
38		Keyestudio RFID Module	1
39		Keyestudio Collision Sensor	1
40		Keyestudio Alcohol Sensor	1
41		Kyestudio LCD_128X32_DOT Module	1
42		5-Channel AD Button Module	1
43		DXL345 Acceleration Module	1
44		Keyestudio ESP32 Development Board	1

continues on next page

Table 1 – continued from previous page

#	Picture	Name	QTY
45		Keyestudio ESP32-IO Expansion Board	1
46		Keyestudio IR Remote Control	1
47		USB Cable	1
48		F-F Dupont Wire	1
49		White Card	1
50		ABS RFID Key	1
51		Battery Holder	1

CHAPTER

THREE

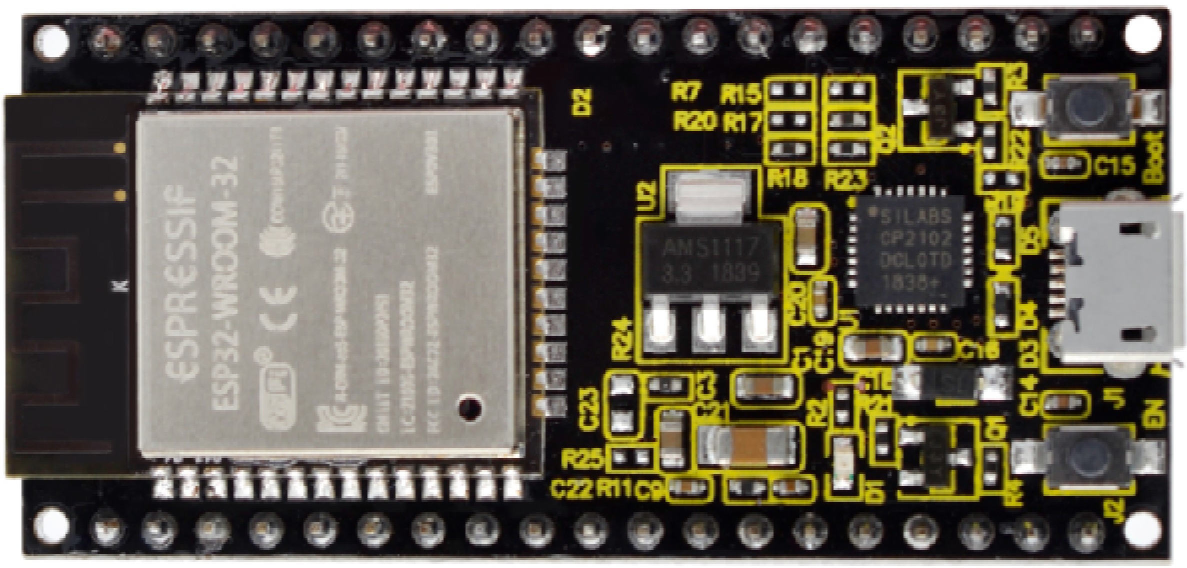
3. TUTORIALS

- *1.ESP32_Mainboard_and_ESP32_Expansion_Board*
- *2.Python_Tutorial(Windows)*
- *3.Arduino_C_Tutorial(Windows)*
- *4.Arduino_C_Tutorial(Raspberry-Pi)*
- 5.Libraries_Driver_Firmware_and_APP
- 6.Codes

CHAPTER FOUR

ESP32 MAINBOARD AND ESP32 SHIELD

4.1 1. Keyestudio ESP32 Mainboard



4.1.1 1.1. Introduction:

Keyestudio ESP32 Core board is a Mini development board based on the ESP-WROOM-32 module. The board has brought out most I/O ports to pin headers of 2.54mm pitch. These provide an easy way of connecting peripherals according to your own needs.

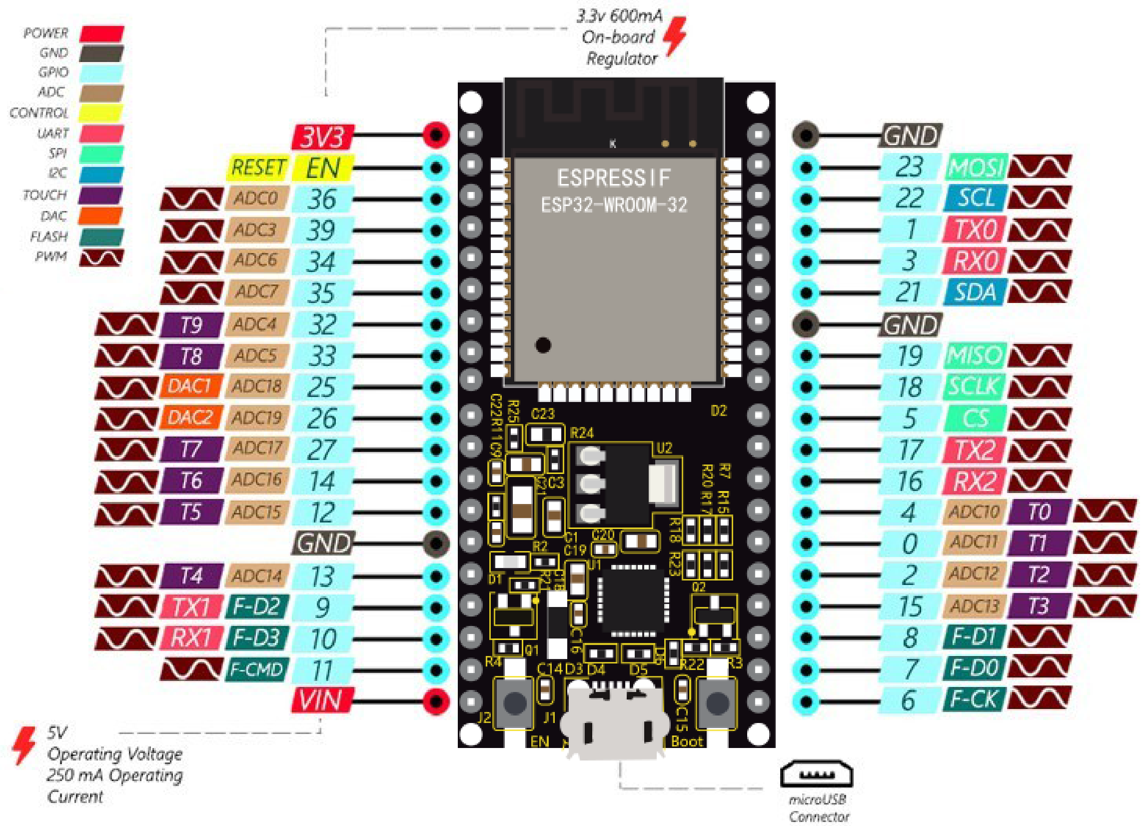
When it comes to developing and debugging with the development board, the both side standard pin headers can make your operation more simple and handy.

The ESP-WROOM-32 module is the industry's leading integrated WiFi + Bluetooth solution with less than 10 external components. It integrates antenna switches, RF balun, power amplifiers, low noise amplifiers, filters as well as power management modules. At the same time, it also integrates TSMC's low-power 40nm technology, power performance and rf performance, making it safe, reliable and easy to expand to a variety of applications.

4.1.2 1.2. Specifications:

- Microcontroller: ESP-WROOM-32Module
- USB to Serial Port Chip: CP2102-GMR
- Working Voltage: DC 5V
- Working Current80mAAverage
- Current Supply500mAMinimum
- Working Temperature Range: -40°C ~ +85°C
- WiFi ModeStation/SoftAP/SoftAP+Station/P2P
- WiFi Protocol802.11 b/g/n/e/i802.11nSpeed up to 150 Mbps
- WiFi Frequency Range2.4 GHz ~ 2.5 GHz
- Bluetooth Protocolconform to Bluetooth v4.2 BR/EDR and BLE Standard
- Dimensions55×26×13mm
- Weight9.3g

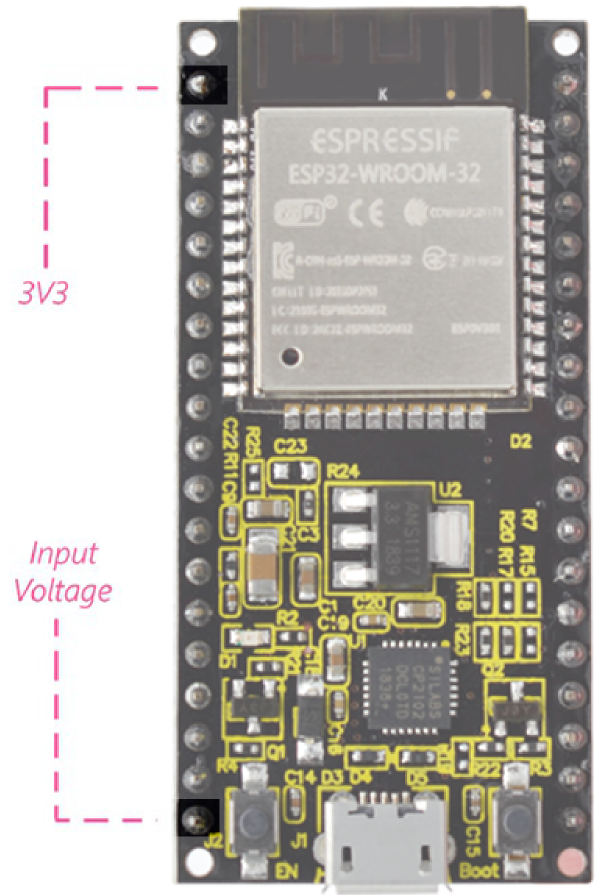
4.1.3 1.3. Pin out:



ESP32 has fewer pins than commonly used processors, but it doesn't have any problems reusing multiple functions on pins.

Warning: The pin voltage level of the ESP32 is 3.3V. If you want to connect the ESP32 to another device with an operating voltage of 5V, you should use a level converter to convert the voltage level.

Power Pins: The module has two power pins +5V and 3.3V. You can use these two pins to power other devices and modules.



GND Pins The module has three grounded pins.

Enable pin (EN): This pin is used to enable and disable modules. The pin enables module at high level and disables module at low level.

Input/Output pins (GPIO): You can use 32 GPIO pins to communicate with LEDs, switches and other input/output devices. You can also pull these pins up or down internally.

Note: GPIO6 to GPIO11 pins (SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD pins) are used for SPI communication for the internal module, which are not recommended.

ADC: You can use the 16 ADC pins on this module to convert analog voltages (the output of some sensors) into digital voltages. Some of these converters are connected to internal amplifiers and are capable of measuring small voltages with high accuracy.

DAC: ESP32 module has two A/D converters with 8-bit precision.

Touch pad: The ESP32 module has 10 pins that are sensitive to capacitance changes. You can attach these pins to certain pads (pads on a PCB) and use them as touch switches.

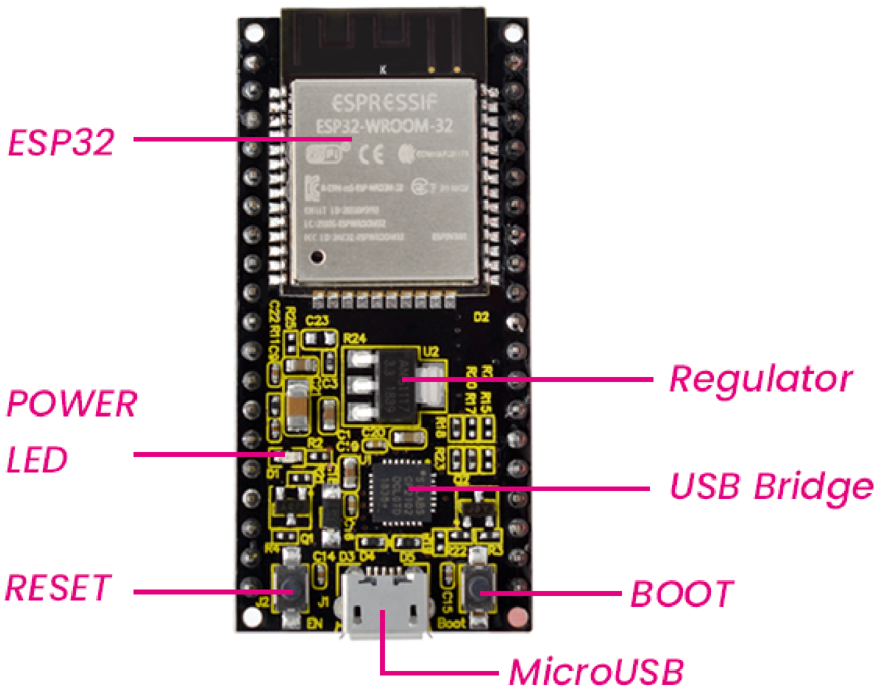
SPI: There are two SPI interfaces on the module, which can be used to connect the display screen, SD/microSD memory card module as well as external flash memory, etc.

I2C: SDA and SCL pins are used for I2C communication.

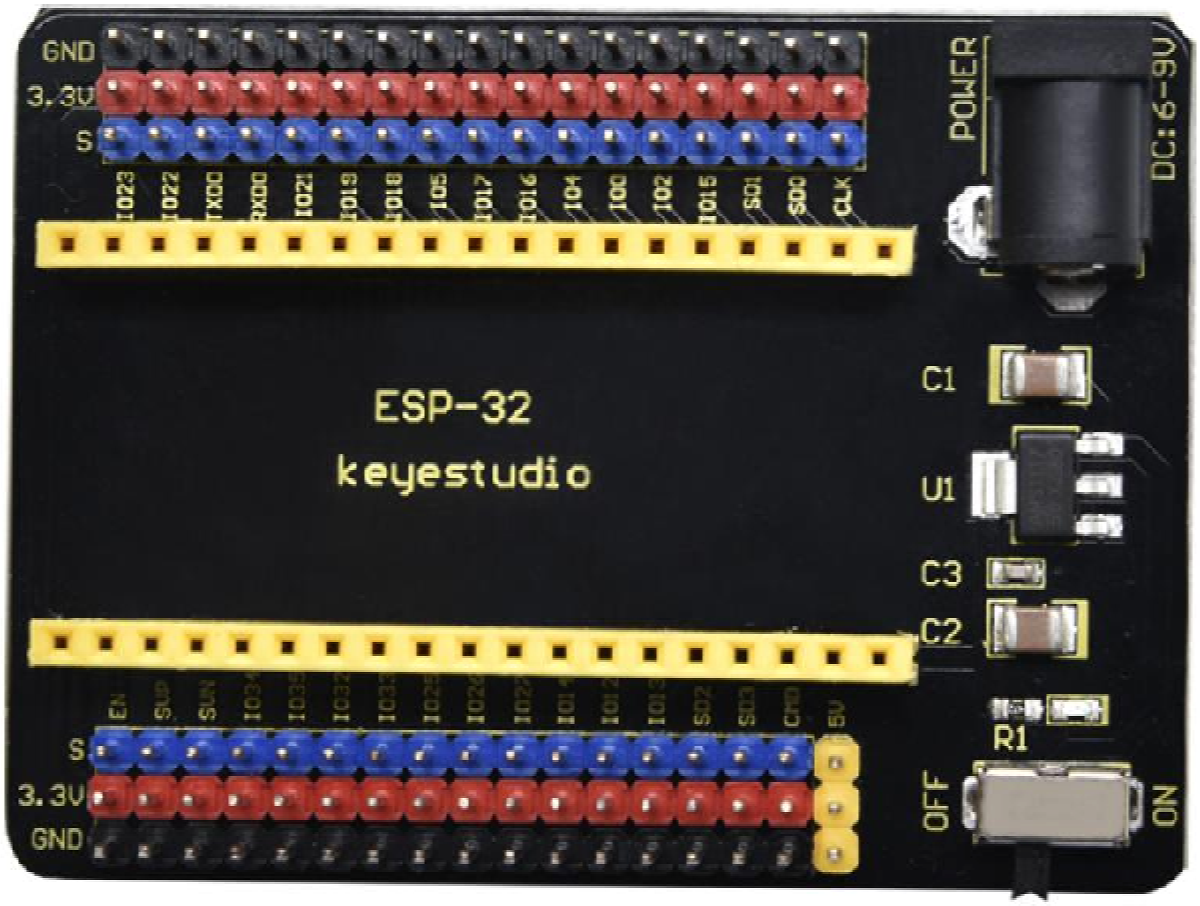
Serial Communication (UART): There are two UART serial interfaces on this module, which can be used to transfer up to 5Mbps of information between two devices. The UART0 also has CTS and RTS control functions.

PWM: Almost all ESP32 input/output pins can be used for PWM (pulse-width modulation). Using these pins can control the motor, LED lights and colors, etc.

4.1.4 1.4. Components:



4.2 2. Keyestudio ESP32-IO shield



4.2.1 2.1. Overview:

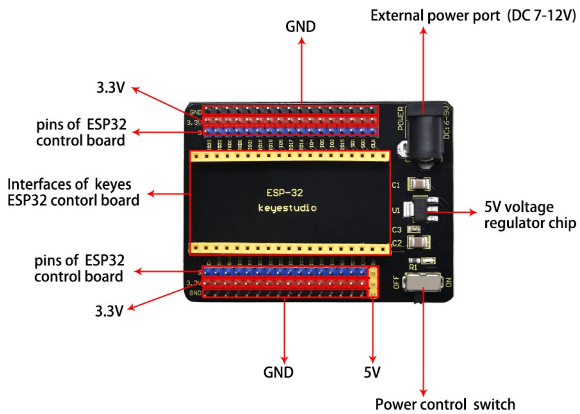
Keyestudio ESP32-IO Expansion Board is designed to be compatible with the Keyestudio ESP32 Core Board (KS0413), which leads all pin connections of the ESP32 Core Board using a row of pins spaced 2.54mm apart. To facilitate the connection of other sensors, it also has two rows of pins with a spacing of 2.54mm rows, which are used to supply 3.3V DC power for external sensors/modules.

A power supply circuit is designed on the control board as it seeks to power the Keyestudio ESP32 Core Board easily. You only need to input DC 6-9V voltage on the black DC head to power it. In addition, it also has a DIP switch to control the power switch.

4.2.2 2.2. Specifications:

- Voltage SupplyDC 6-9V
- Operating Current60mA
- Maximum Power0.3W
- Working Temperature-25℃ to +65℃
- Dimensions30mm*20mm
- Environmental Protection AttributesROHS

4.2.3 2.3. Pins and Components:



CHAPTER

FIVE

PYTHON TUTORIAL

5.1 1. Preparation for Python(Windows):

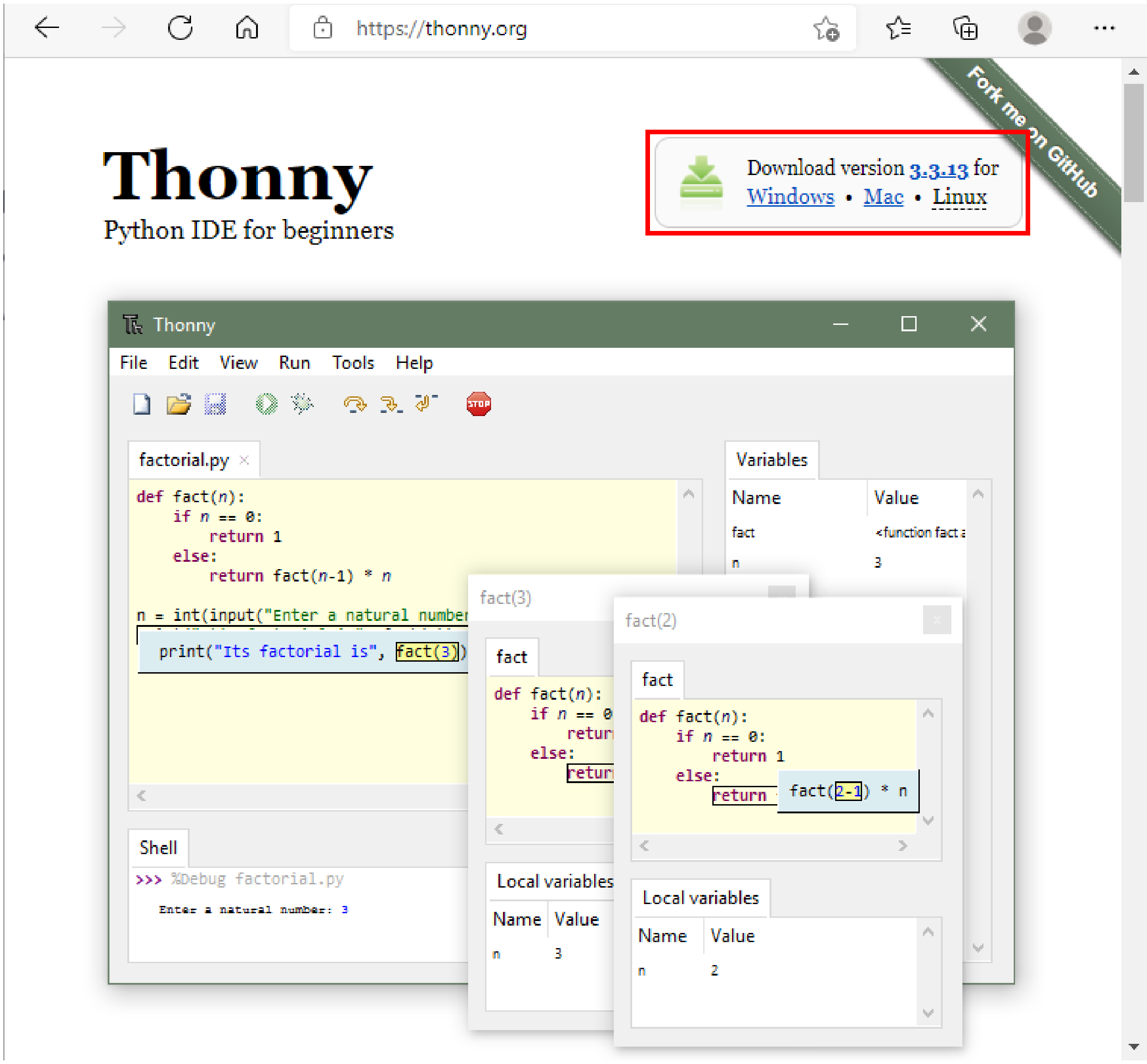
5.1.1 1. Download and Install Thonny

Thonny is a free and open source software platform with small size, simple interface, simple operation and rich functions. It is a Python IDE suitable for beginners. In this tutorial, we use this IDE to develop a ESP32. Thonny supports multiple operating systems including Windows, Mac OS, Linux.

1.1. Download Thonny

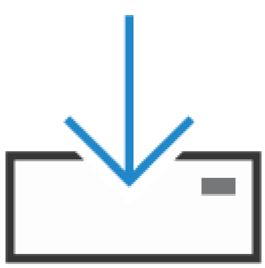
1). Enter the websitehttps://thonny.org to download the latest version of Thonny.

2). Thonny open-source code libraryhttps://github.com/thonny/thonny.

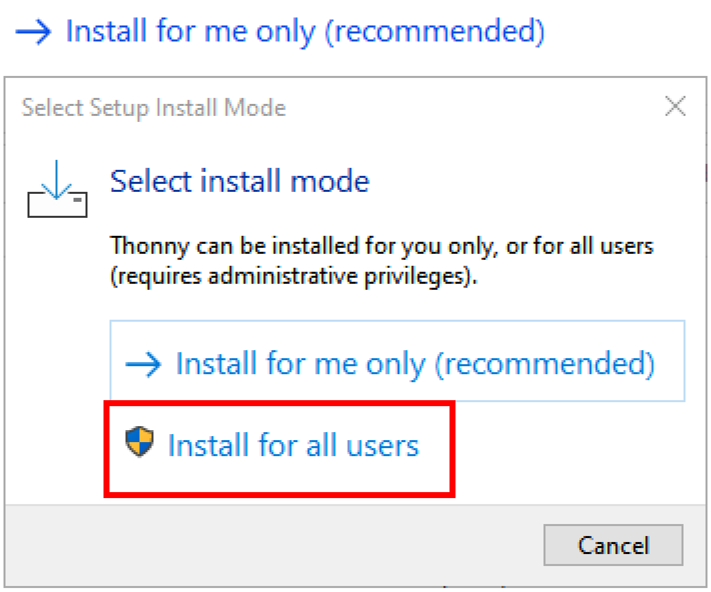


1.2. Install Thonny (Windows System):

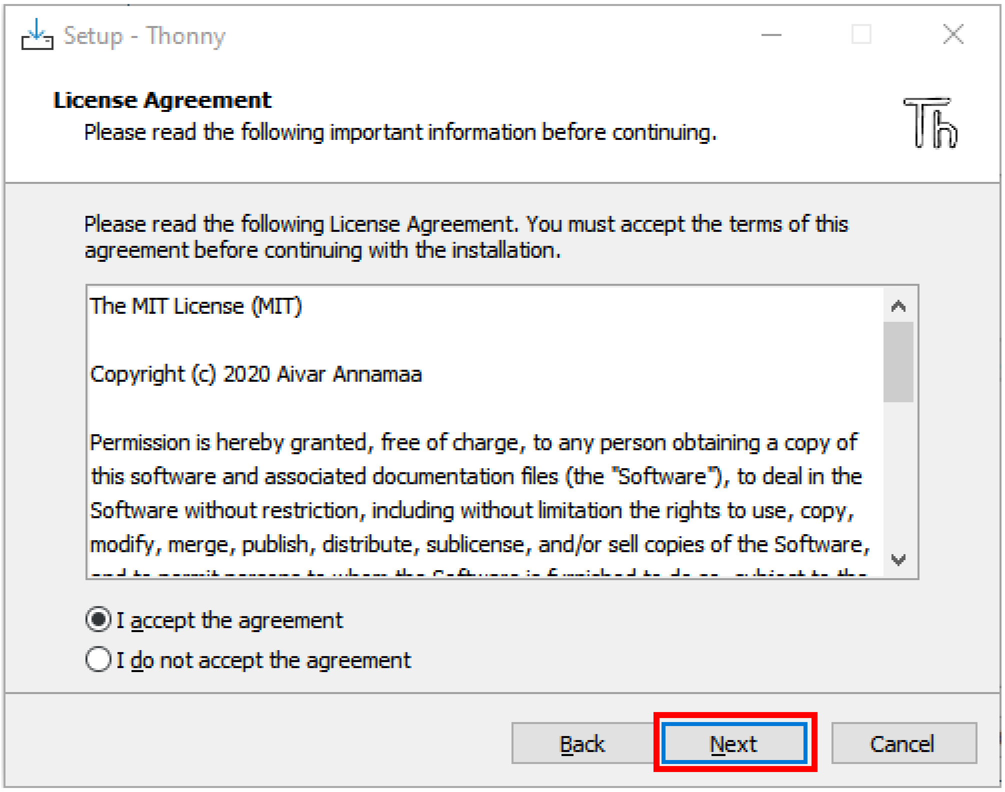
1). The downloaded Thonny icon is as follows:



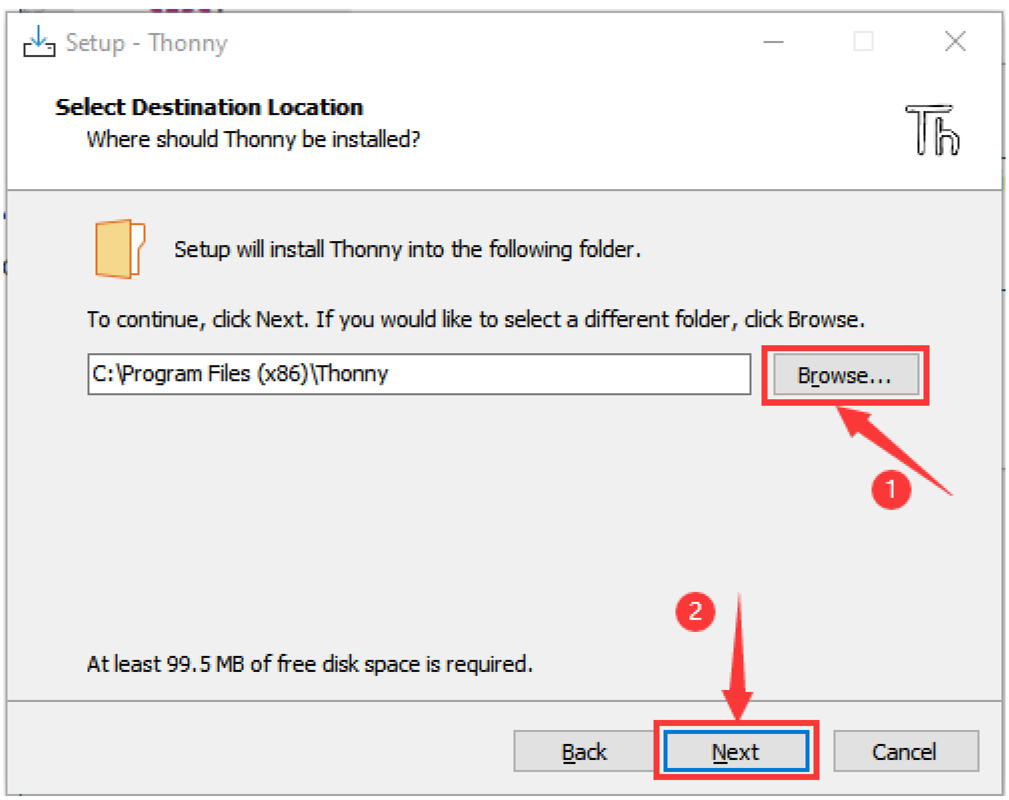
2). Double-click“thonny-3.3.13.exe”and select install mode. You can choose

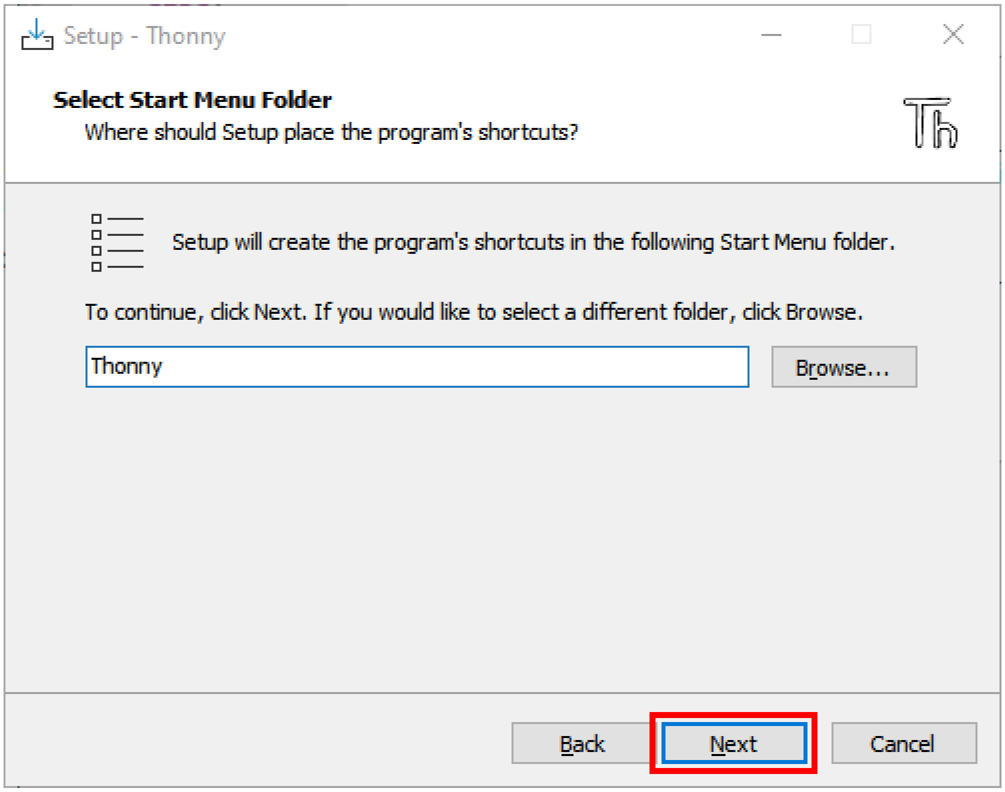


3). You can also keep selecting **Next** to finish install.

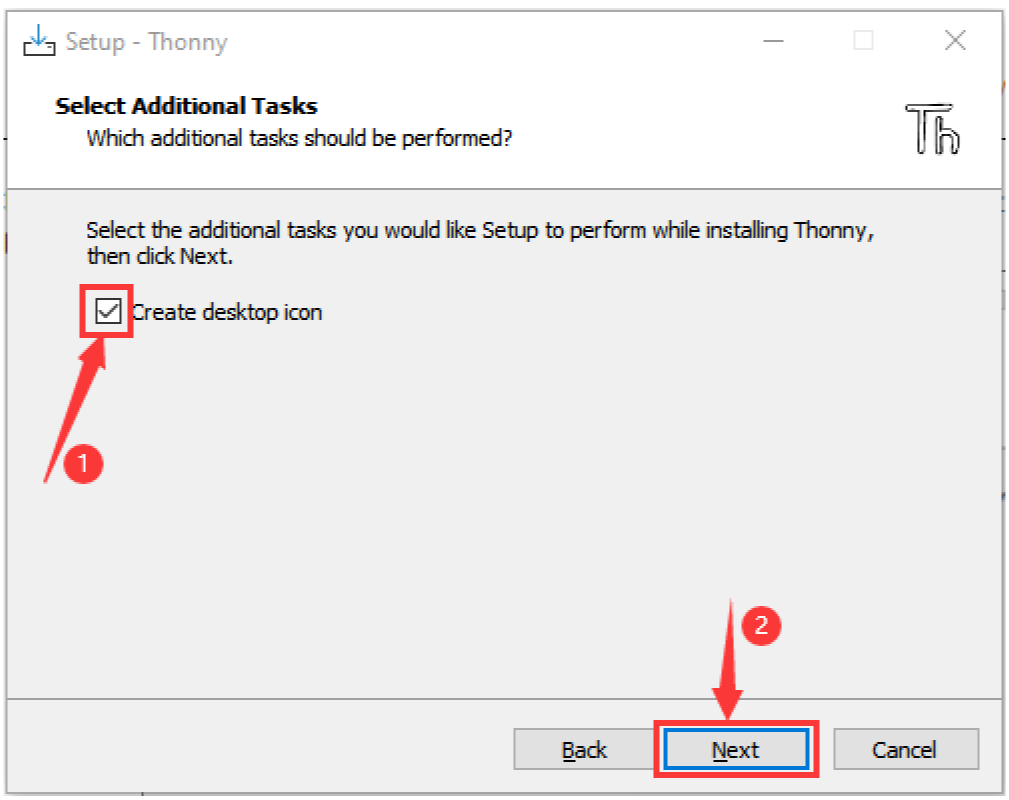


4). If you want to change the route of installing Thonnyjust click"**Browse…**"to select a new route and click **OK**.

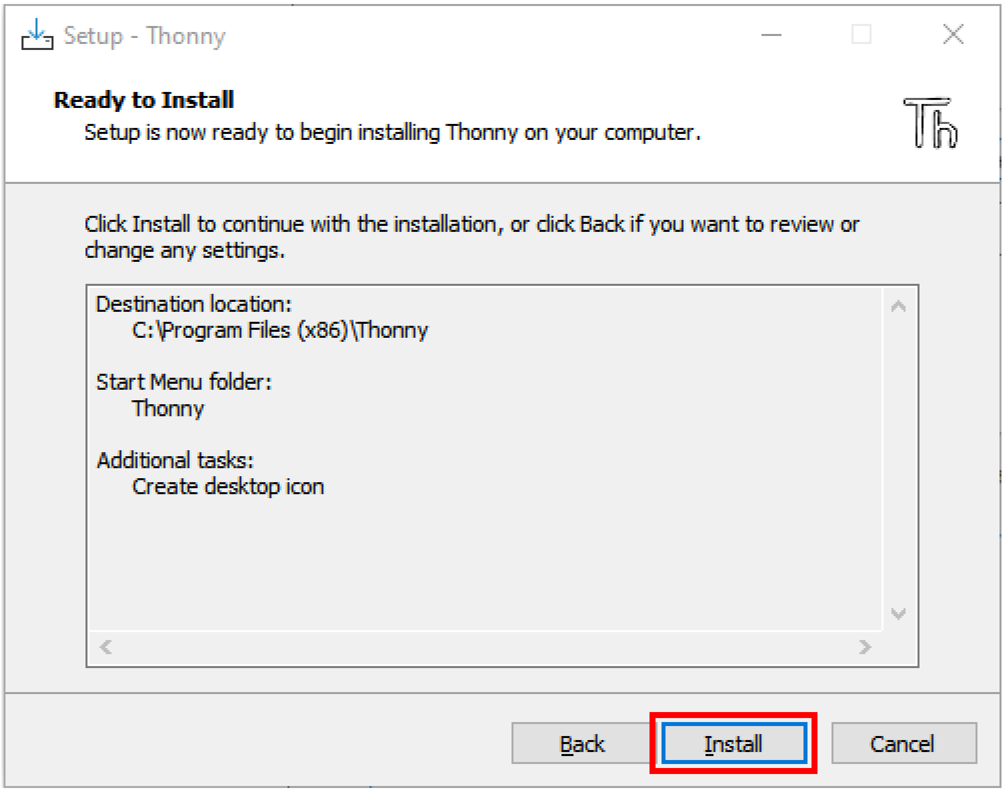




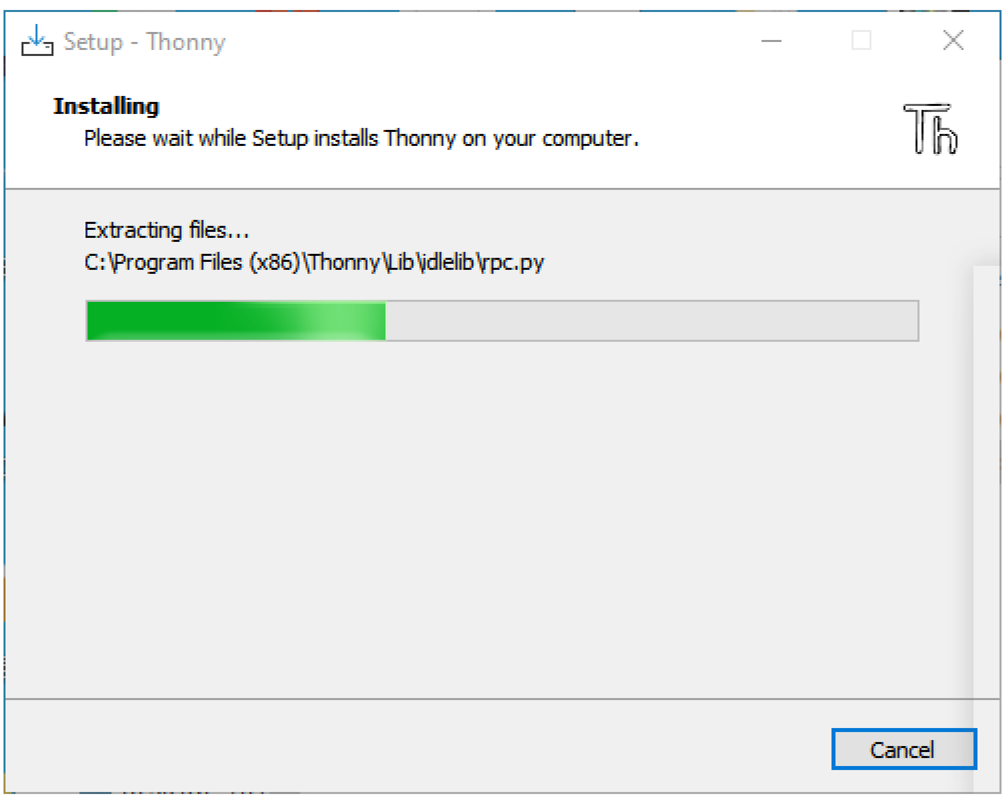
5). Click **Create desktop icon,** you will view Thonny on your desktop.



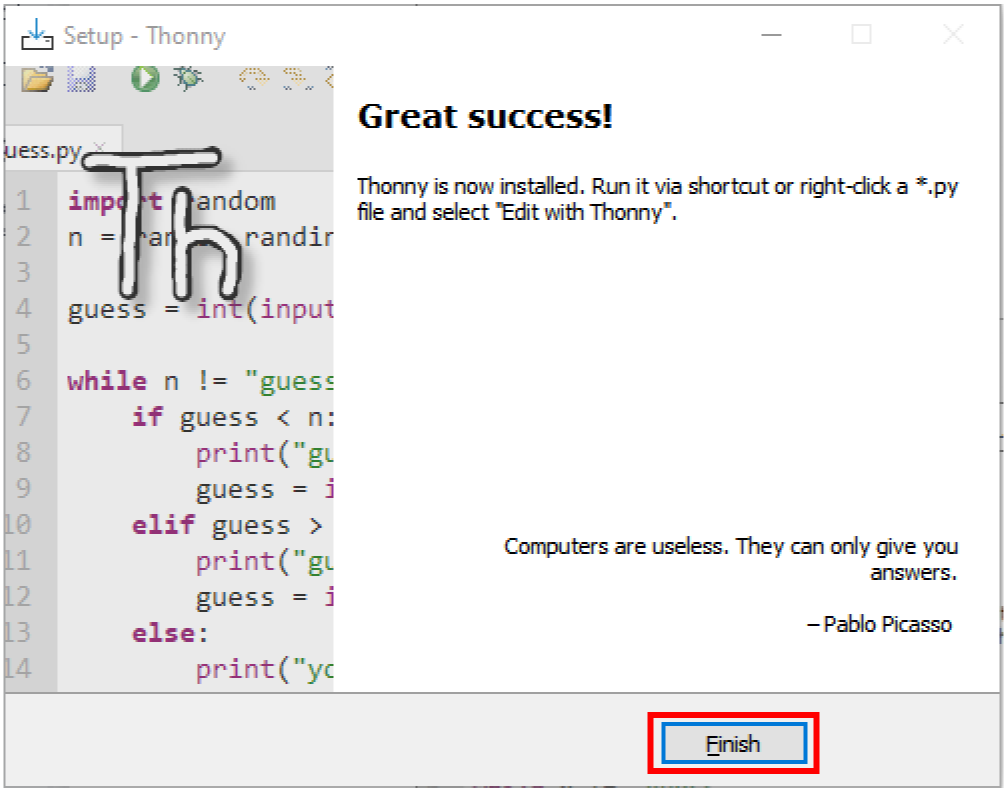
6). Click“**Install**”



7). Wait for a while but don't click **Cancel**

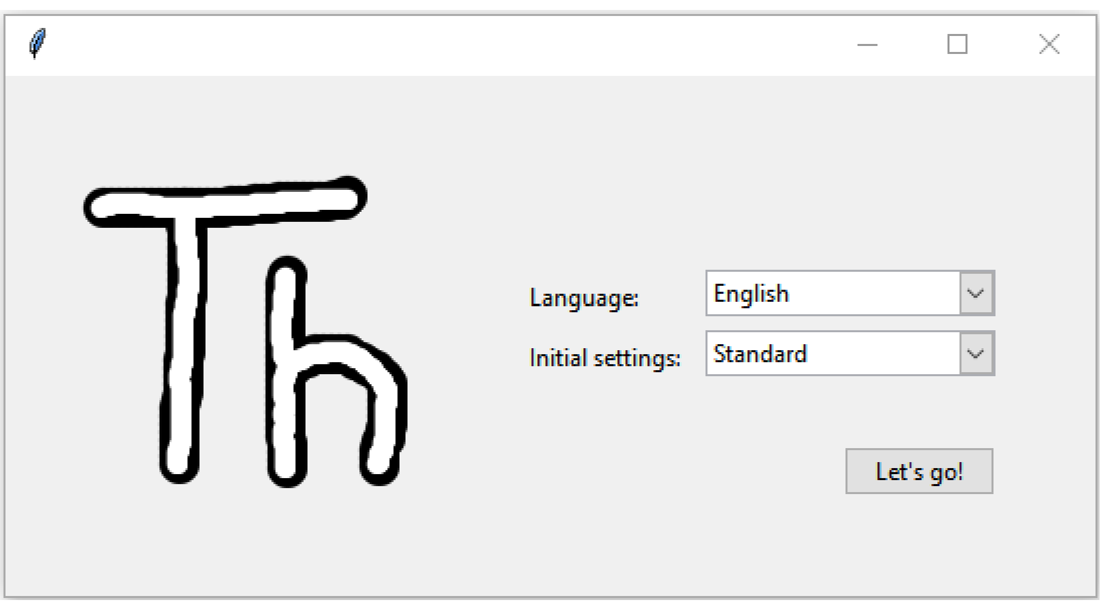


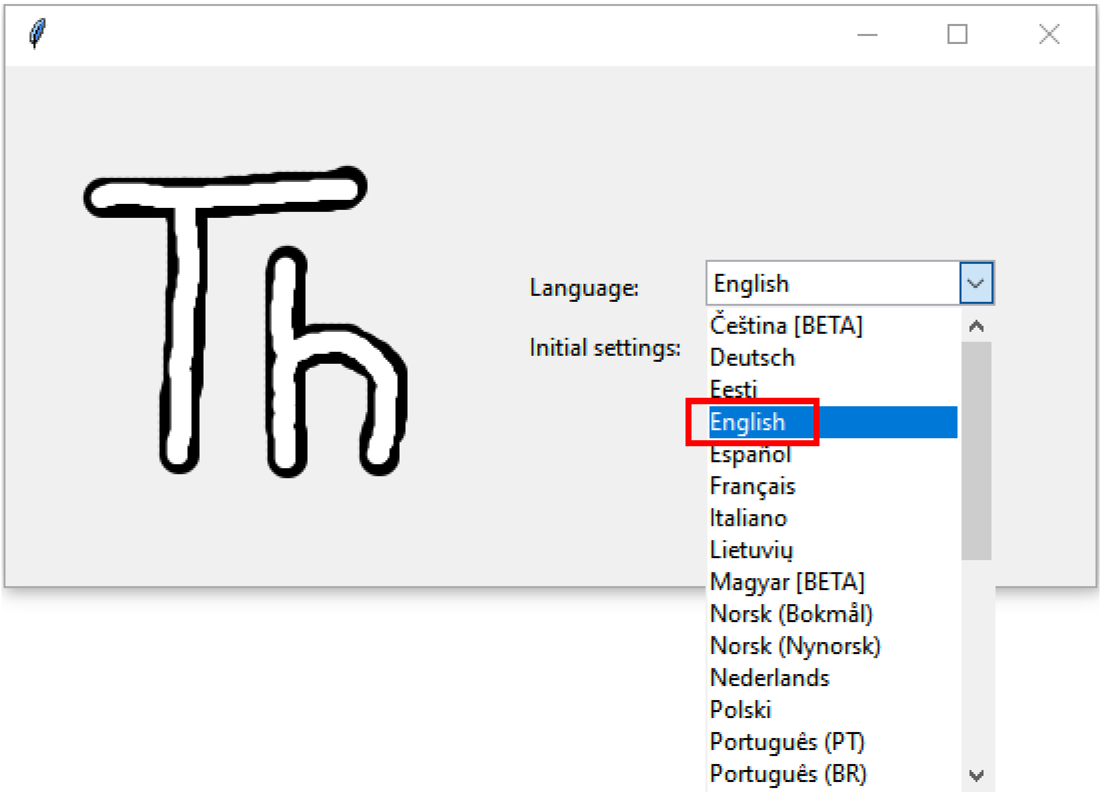
8). Click **"Finish"**

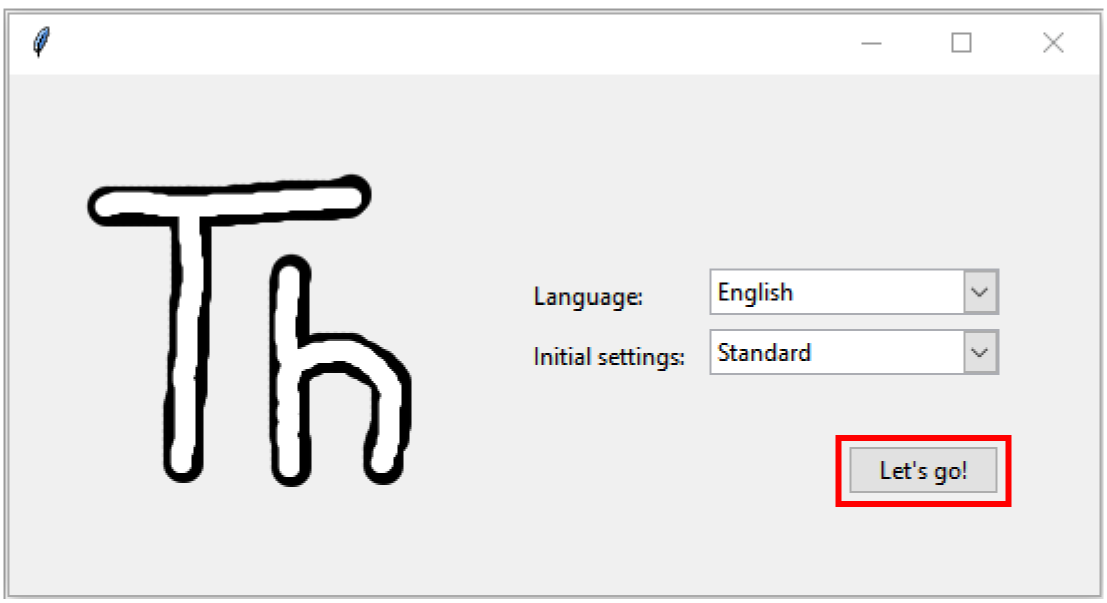


1.3. Basic Setting

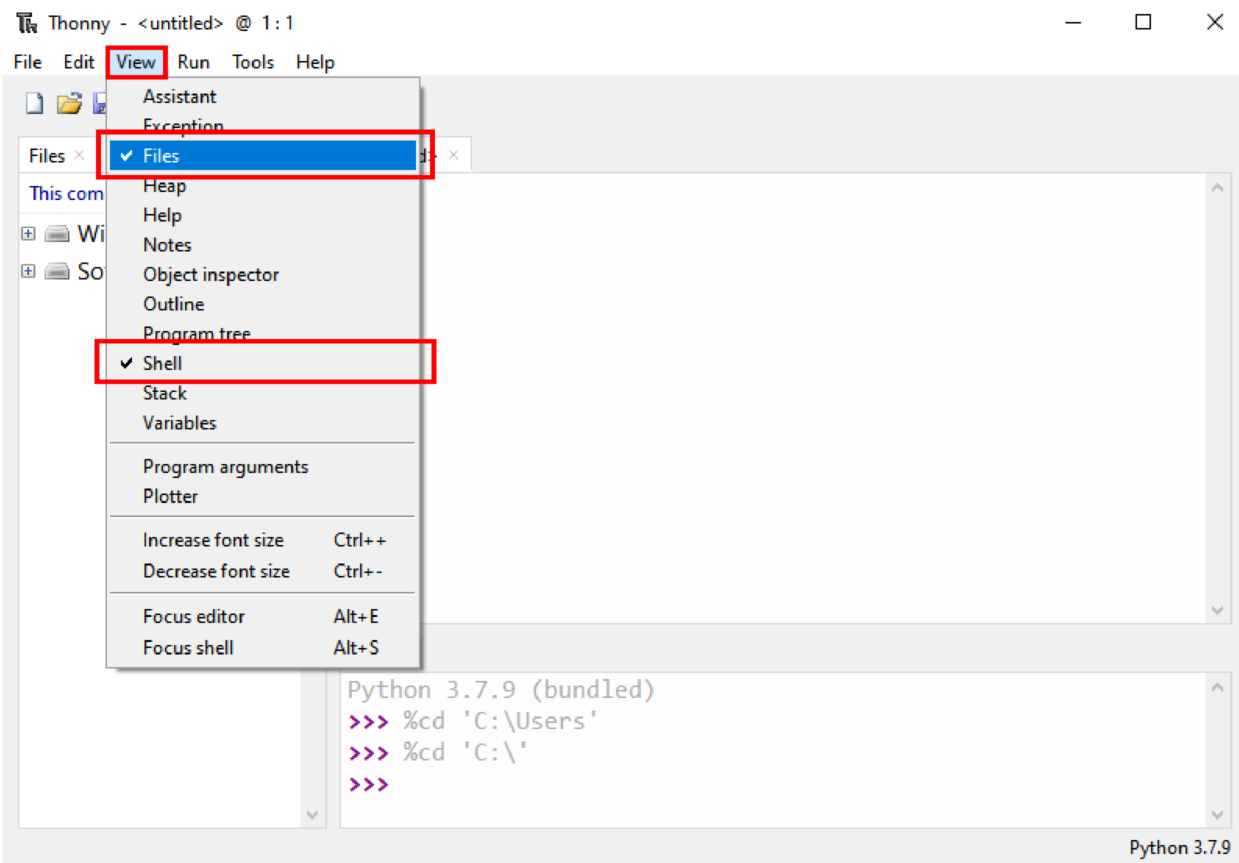
Double-click Thonny, choose lanuage and initial settings and click **Let's go**

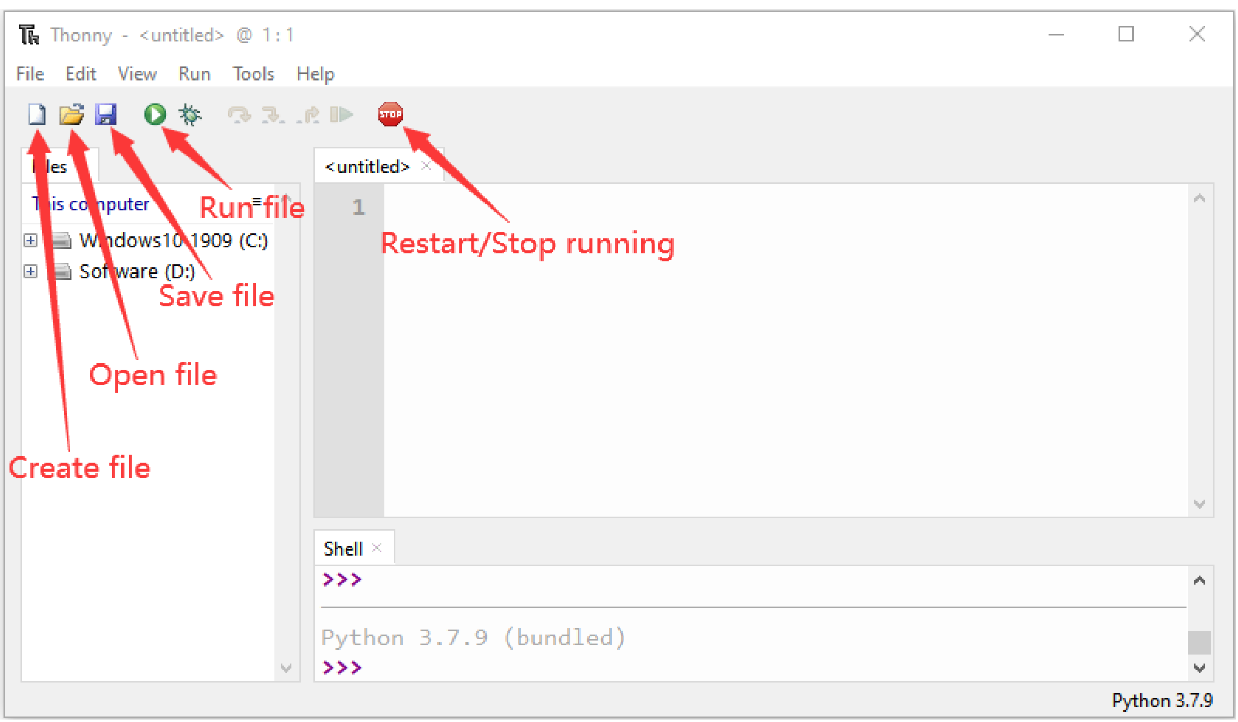


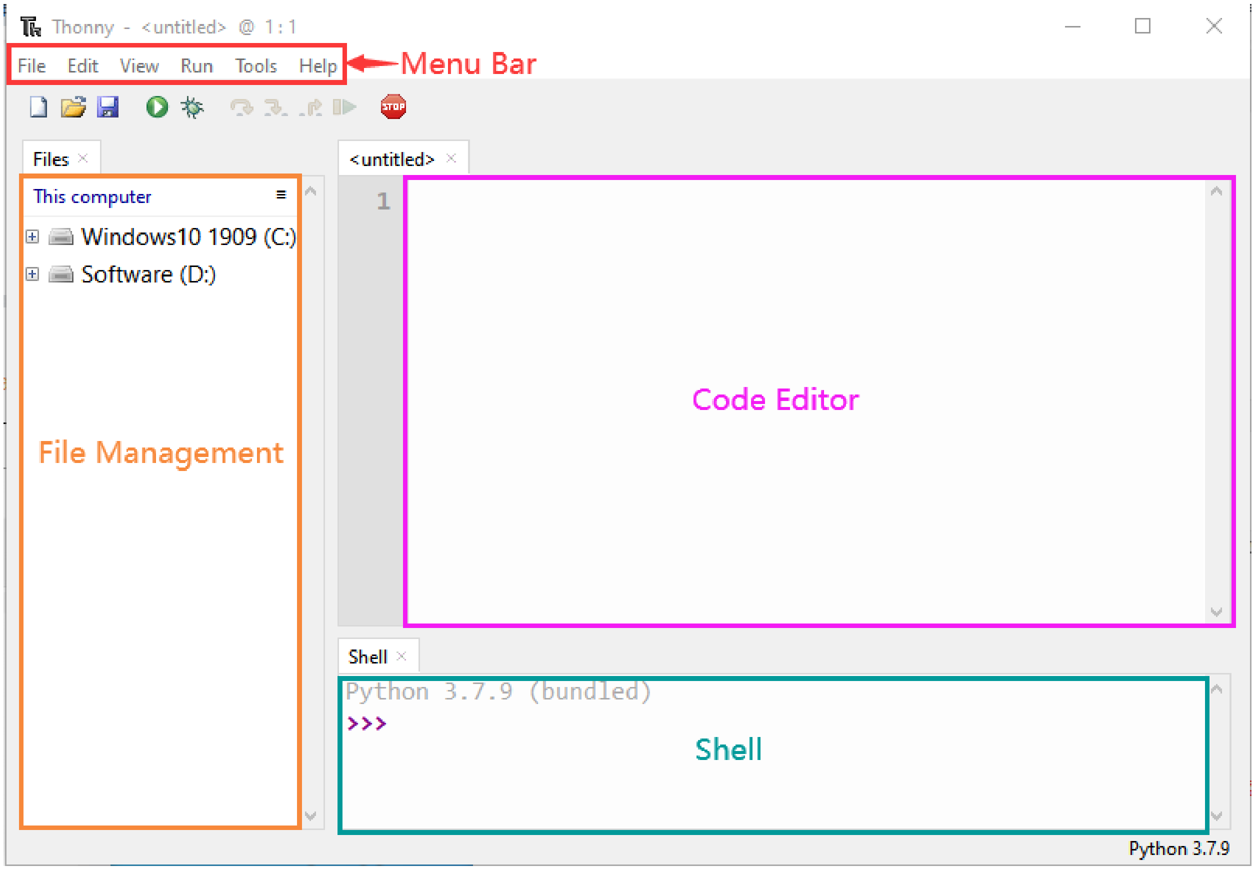




Click"**View**"→"**File**"and"**Shell**"


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
Assistant
Exception
Files
Heap
Help
Notes
Object inspector
Outline
Program tree
Shell
Stack
Variables
Program arguments
Plotter
Increase font size Ctrl++
Decrease font size Ctrl+-
Focus editor Alt+E
Focus shell Alt+S
Python 3.7.9 (bundled)
>>> %cd 'C:\Users'
>>> %cd 'C:\'
>>>
Python 3.7.9


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
Run file
Restart/Stop running
Save file
Open file
Create file
<untitled>
Shell
>>>
Python 3.7.9 (bundled)
>>>
Python 3.7.9



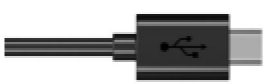
5.1.2 2. Install the CP2102 driver

Before using the Thonny, we need to install the CP2102 driver in the computer.

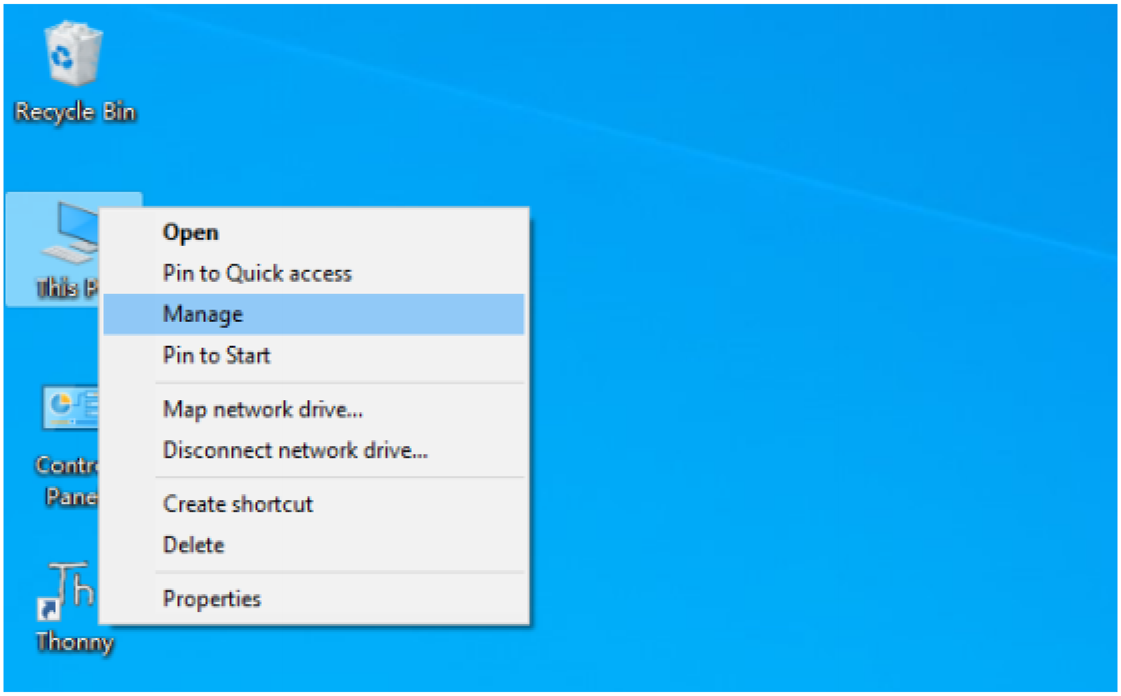
Windows system

Check if the CP2102 driver has been installed.

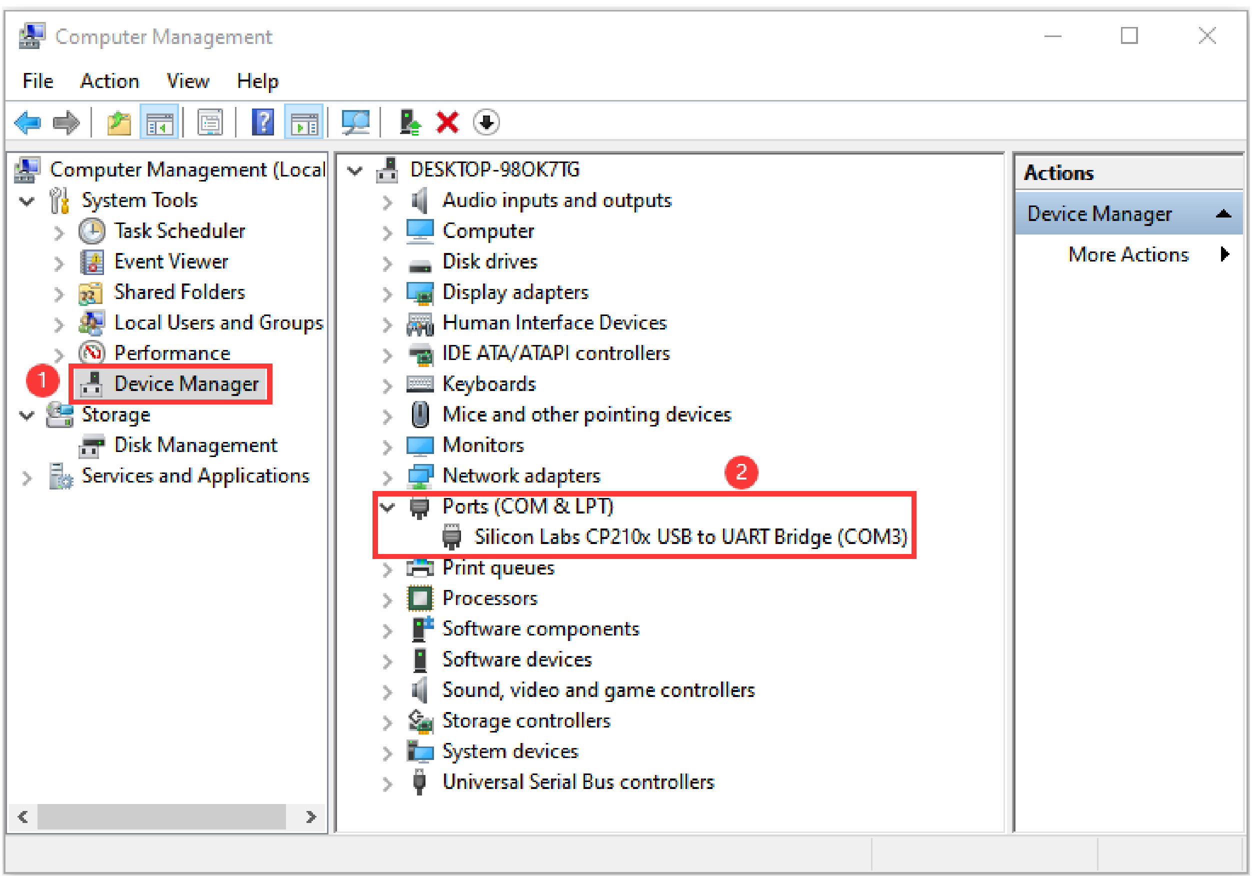
1). Interface the ESP32 with your PC with a USB cable.



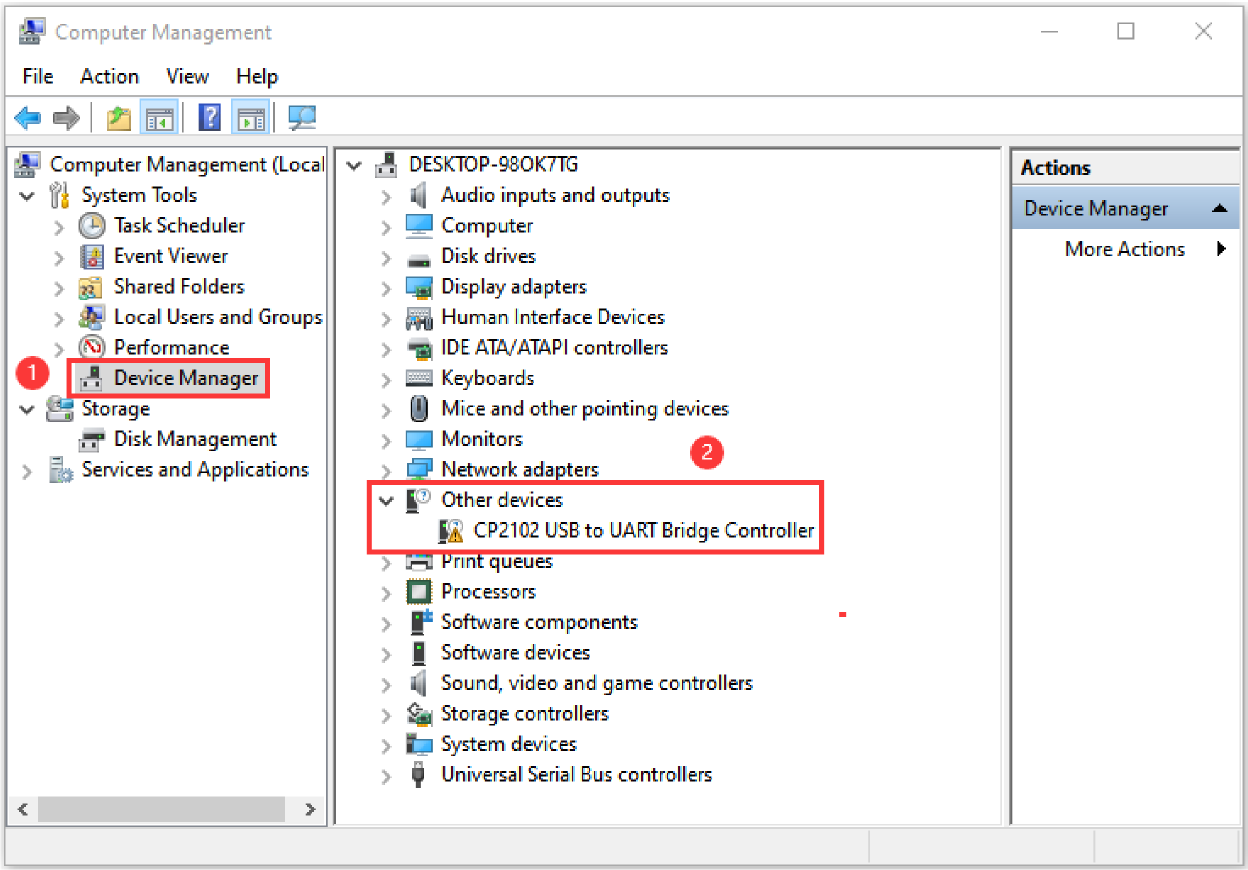
2). Click "This PC" and right-click "Manage".



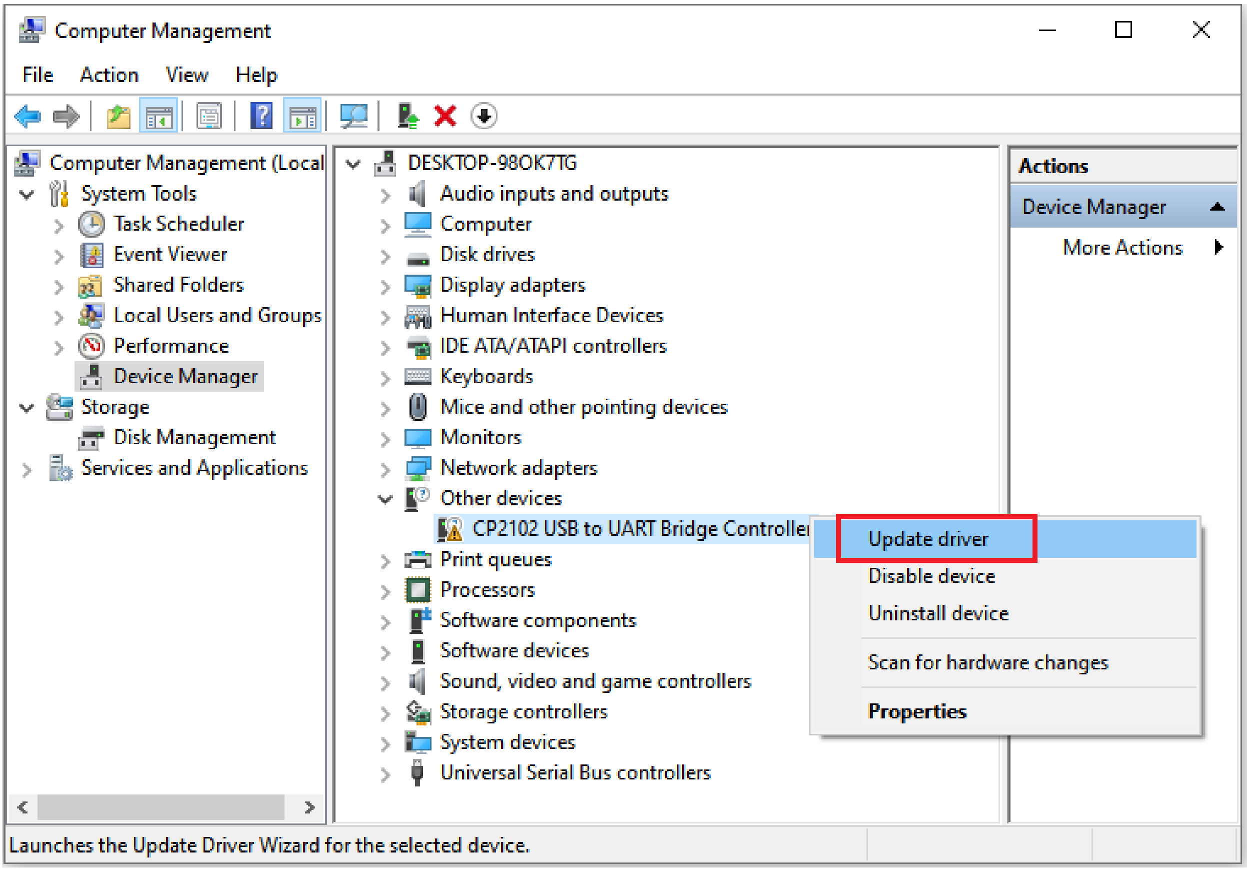
3). Click "Device Manager" , if the CP2102driver has been installed"Silicon Labs CP210x USB to UART Bridge(COMx)" will be shown.



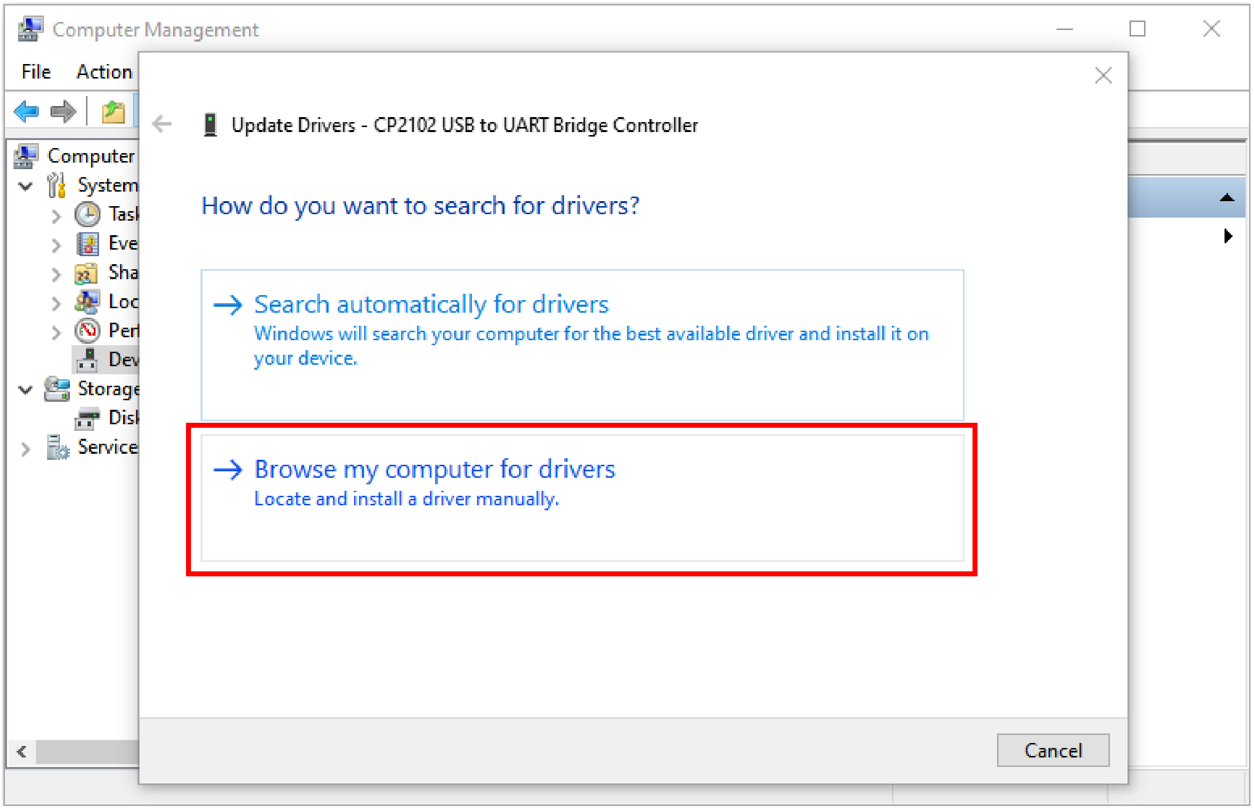
If the CP2102 has not been installed.



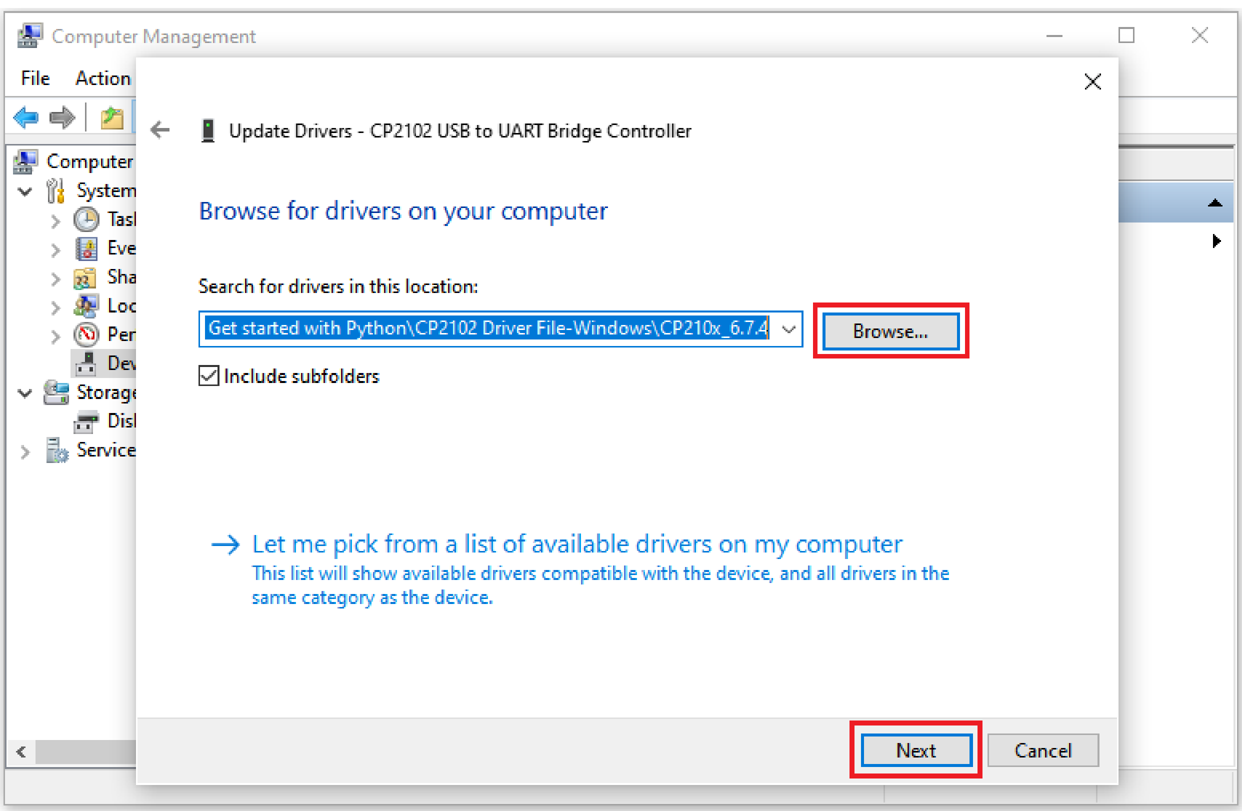
Click "CP2102USB to UART Bridge Controller" and "Update driver".



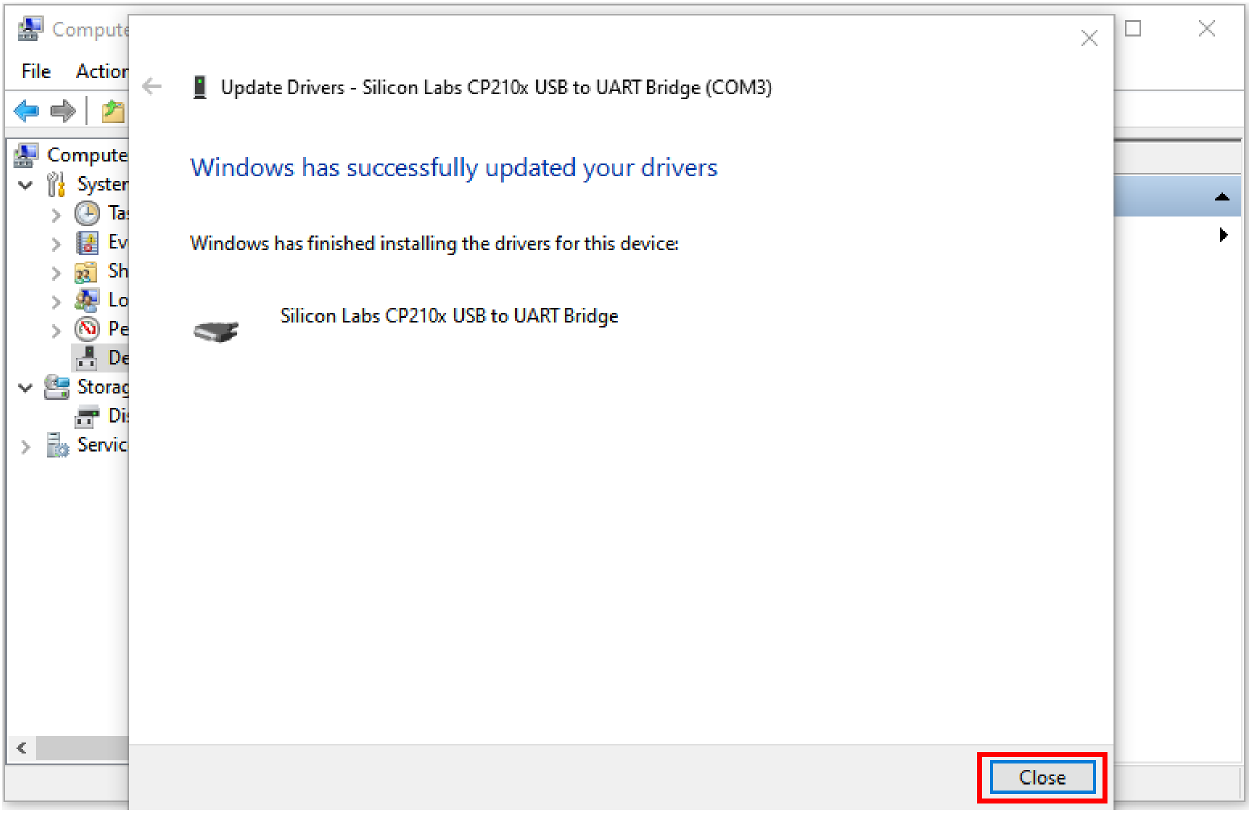
Click “Browse my computer for drivers”.

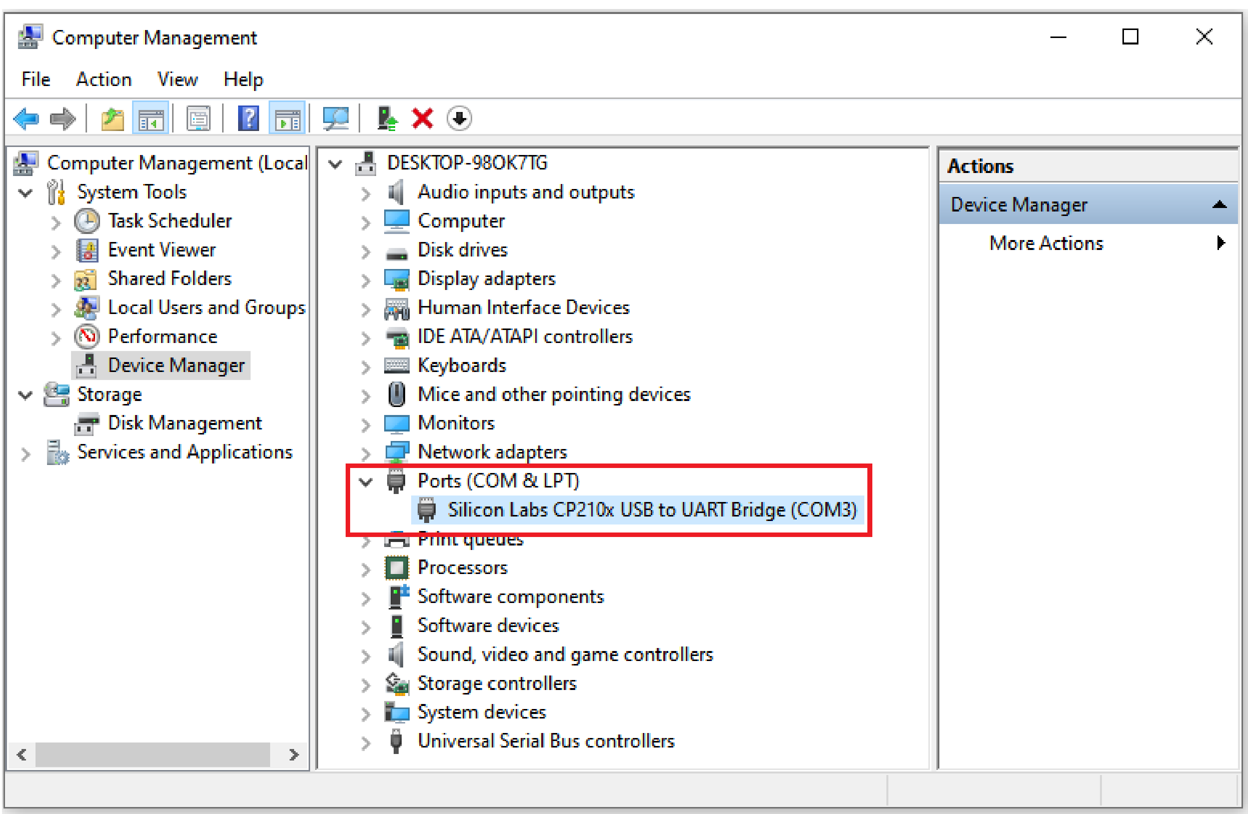


Click "Browse… to" choose "CP2102 Driver File-Windows" and click "Next".



The CP2102 driver will be installed.


Update Drivers - Silicon Labs CP210x USB to UART Bridge (COM3)
Windows has successfully updated your drivers
Windows has finished installing the drivers for this device:
Silicon Labs CP210x USB to UART Bridge
Close

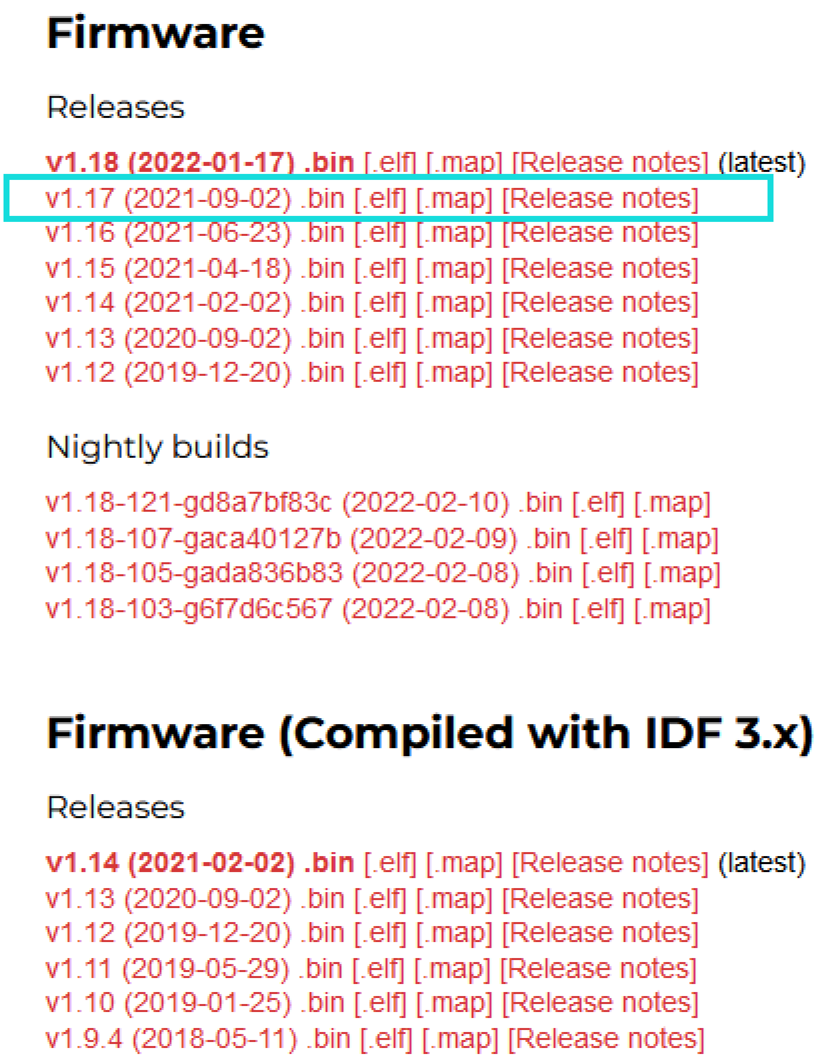

Computer Management
File
Action
View
Help
Computer Management (Local
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Performance
Device Manager
Storage
Disk Management
Services and Applications
DESKTOP-98OK7TG
Audio inputs and outputs
Computer
Disk drives
Display adapters
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Silicon Labs CP210x USB to UART Bridge (COM3)
Processors
Software components
Software devices
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
Actions
Device Manager
More Actions

5.1.3 3. Burn Micropython firmware:

To run a Python program on the ESP32 board, we need to burn the firmware to the ESP32 board first.

Download Micropython firmware microPython websitehttp://micropython.org/

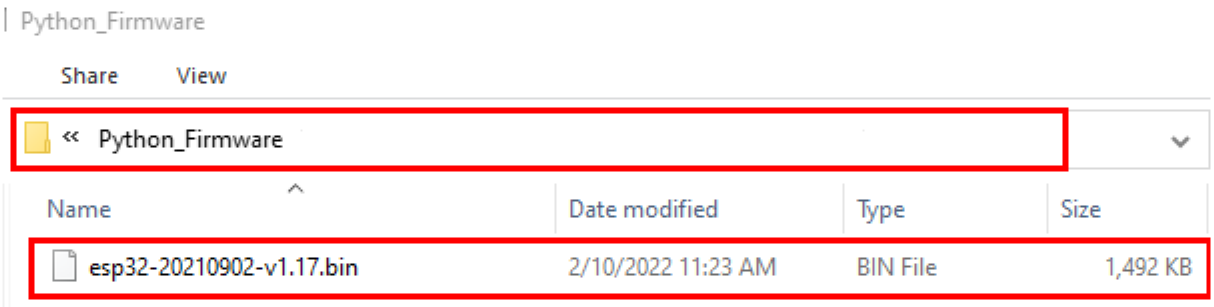
ESP32 firmwarehttps://micropython.org/download/esp32/



The firmware we use**esp32-20210902-v1.17.bin**

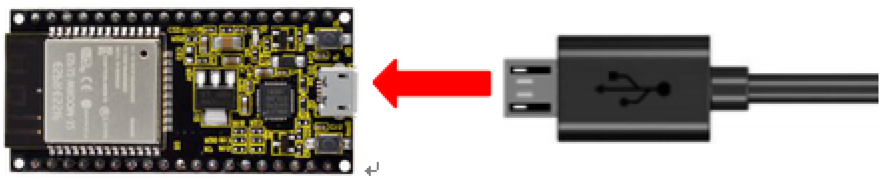
Download firmwarehttps://micropython.org/resources/firmware/esp32-20210902-v1.17.bin

Wo also offer the Firmware"…\Python_Firmware".

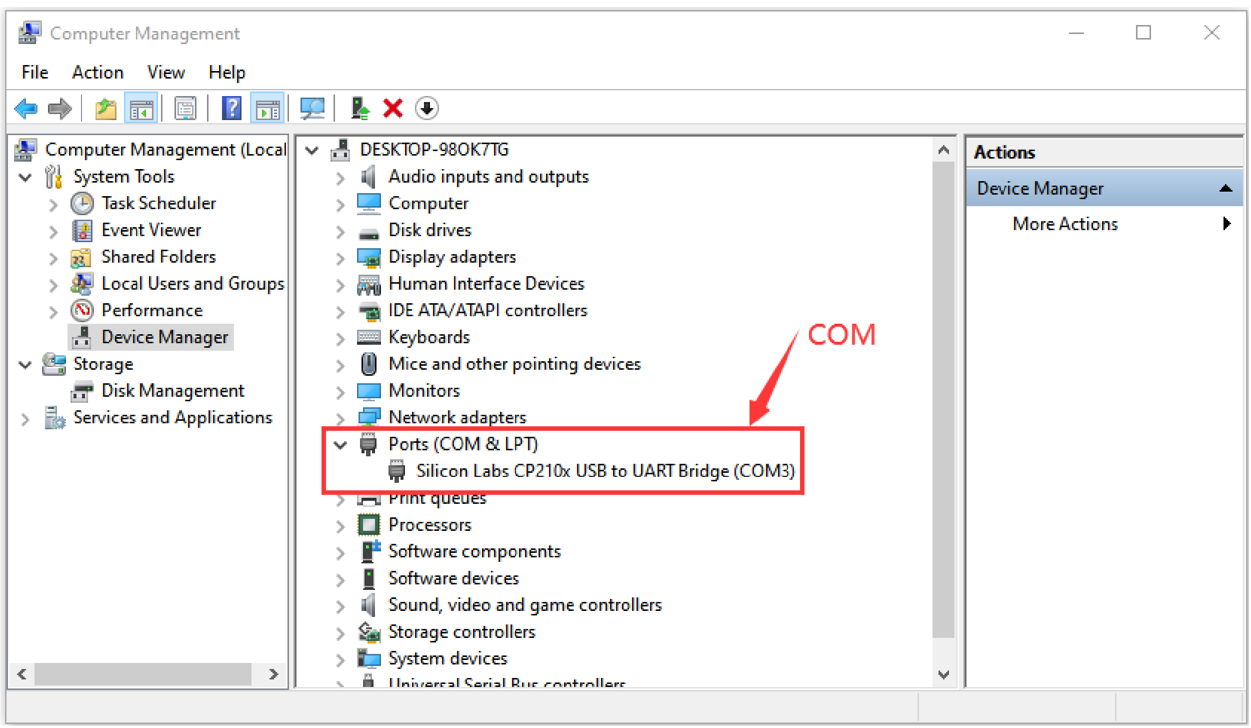


Burn the Micropython firmware

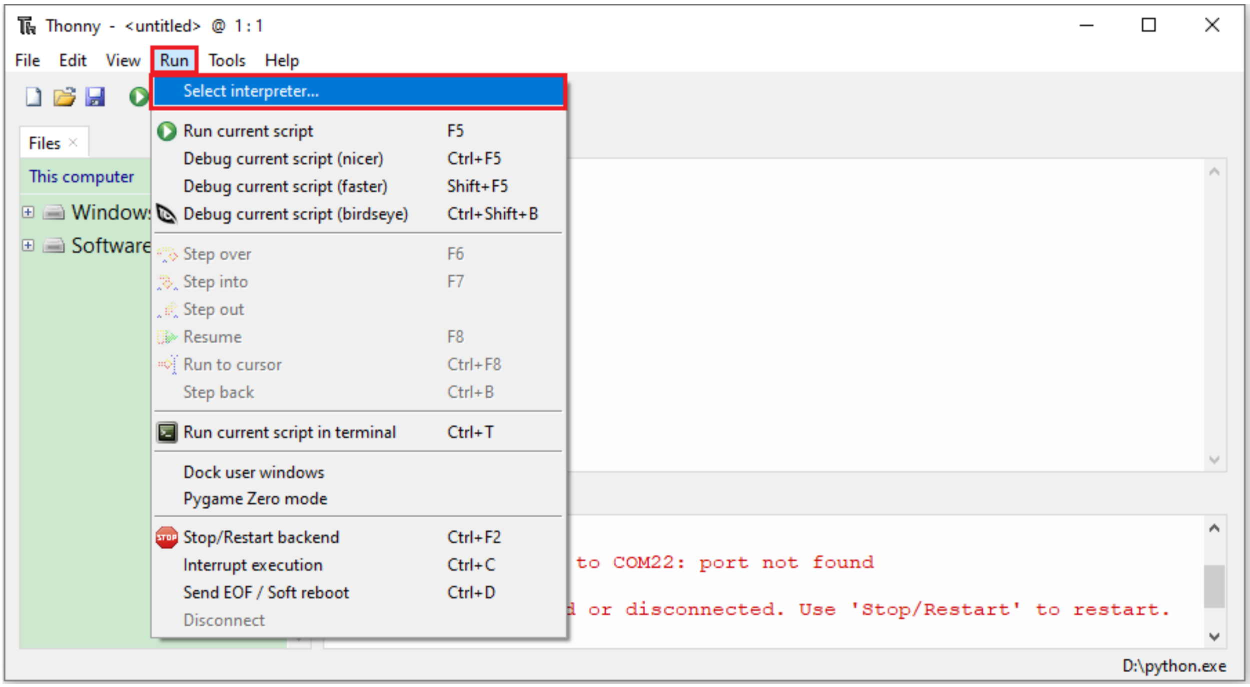
Connect the ESP32 to your PC with a USB cable



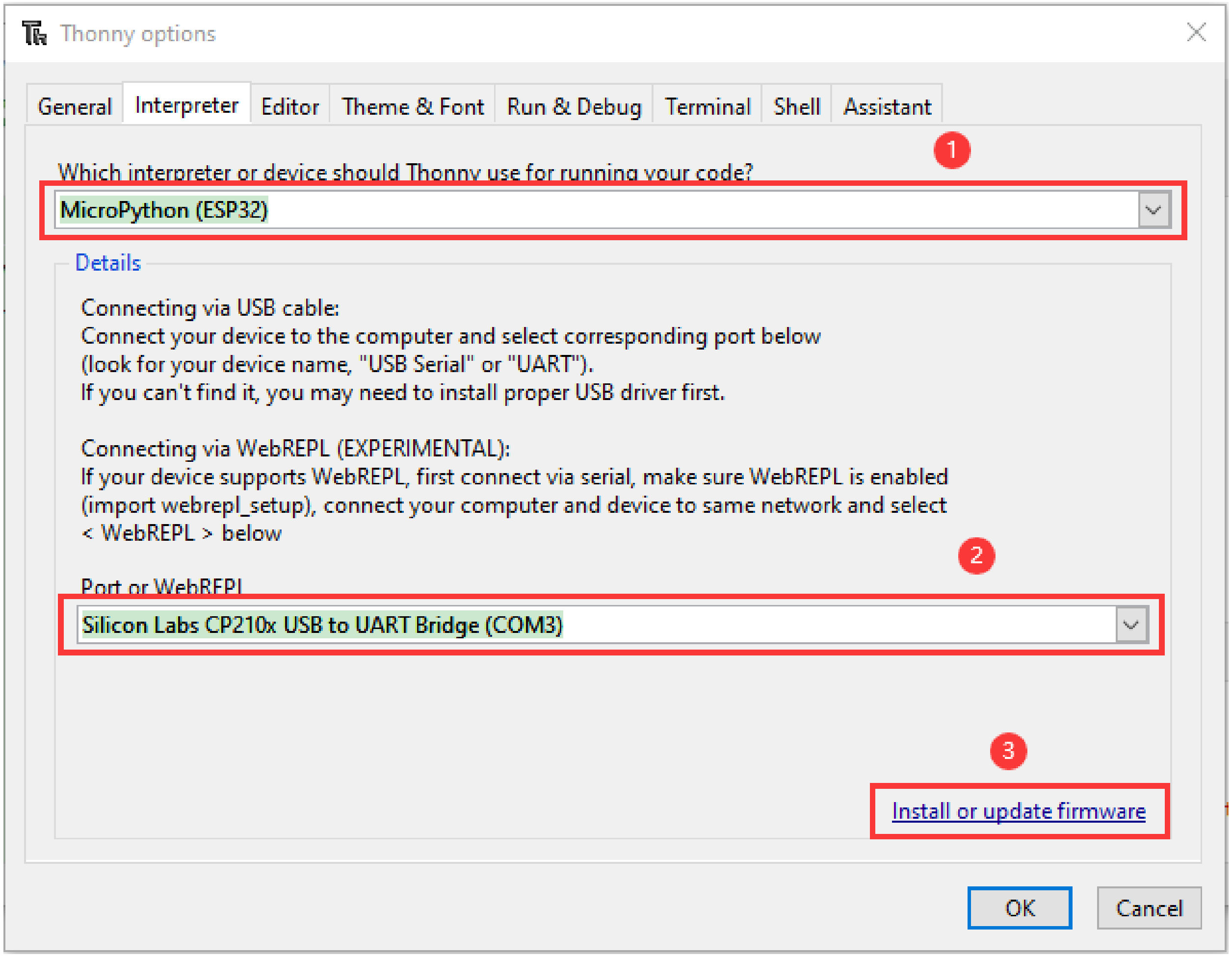
Make sure the driver has been installed successfully and the COM port can be identified correctly. Open Device Manager and expand "Ports".



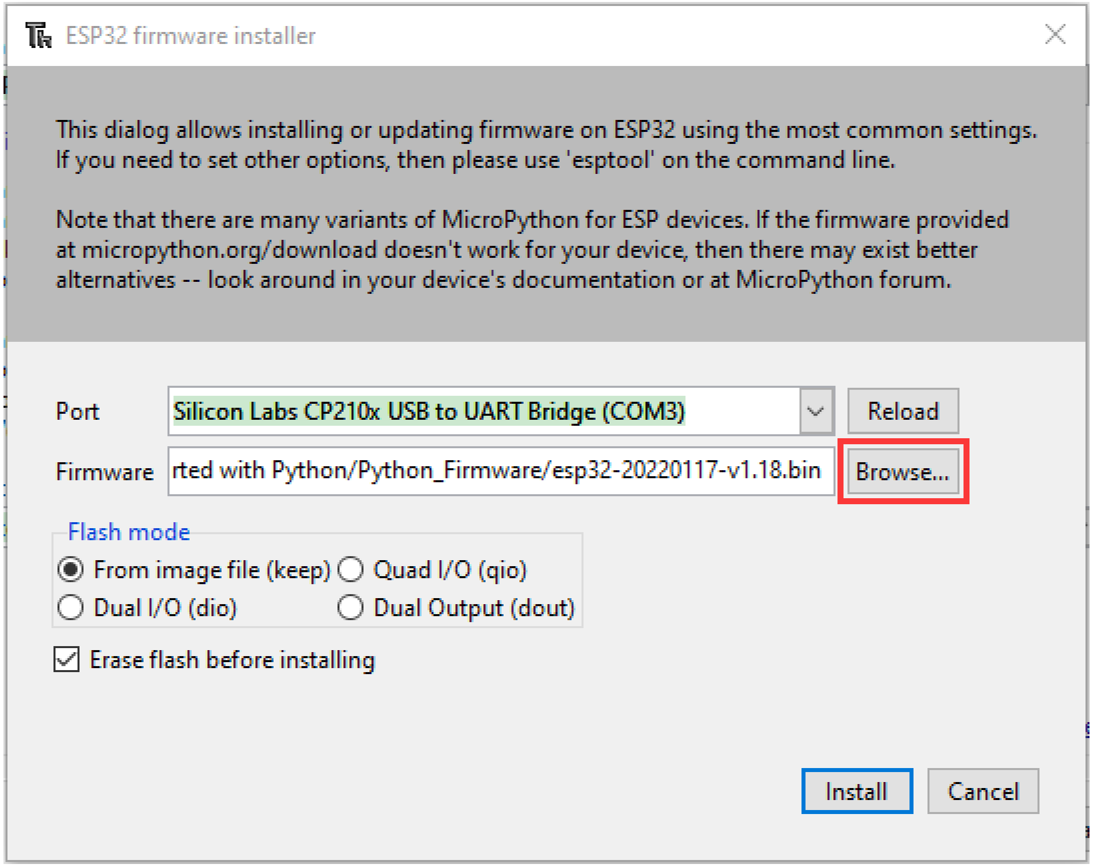
Open Thonnyclick"run"and"Select interpreter…"

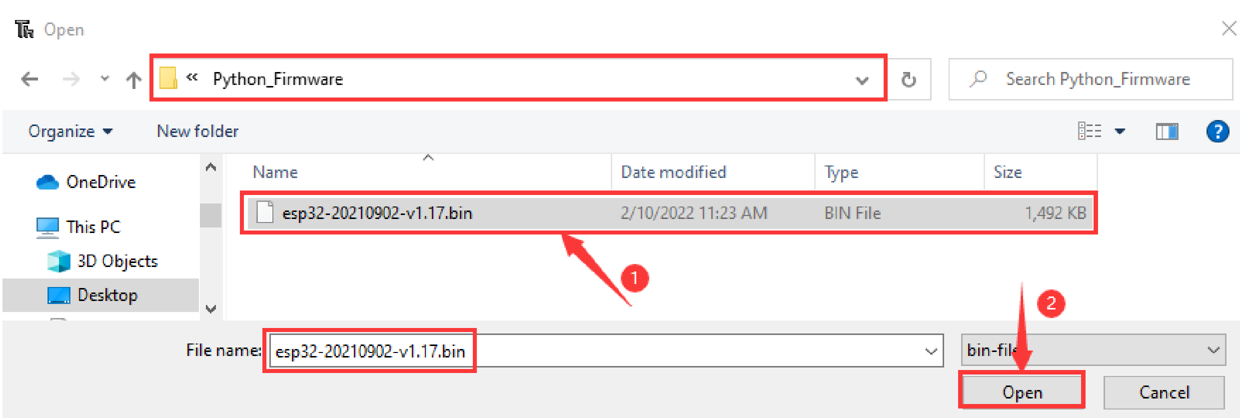


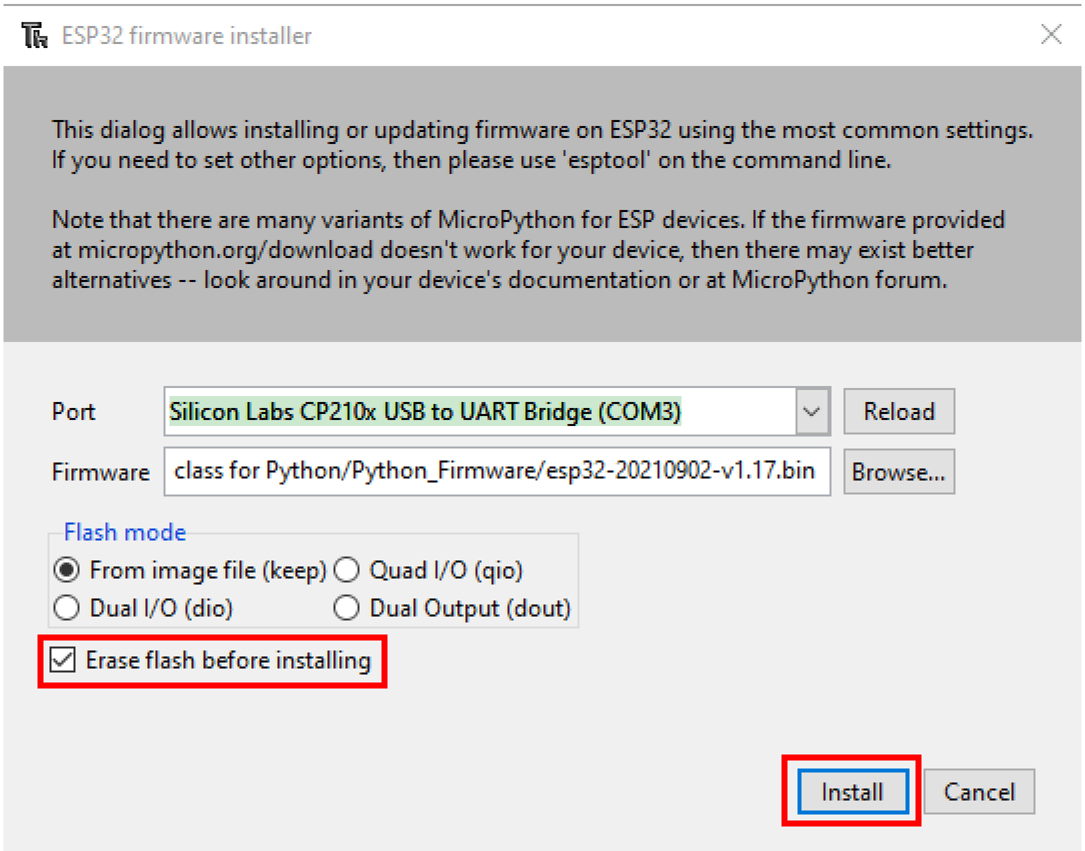
Select "Micropython (ESP32)" and "Silicon Labs CP210x USB to UART Bridge(COM3)" and click "Install or update firmware".



Select"Silicon Labs CP210x USB to UART Bridge(COM3)"click " Browse…"and choose the firmware **esp32-20210902-v1.17.bin.** Check"Erase flash before installing"and"Flash mode"then click"Install".Noteif you fail to install the firmwarepress the Boot button on the ESP32 board and click"Install"

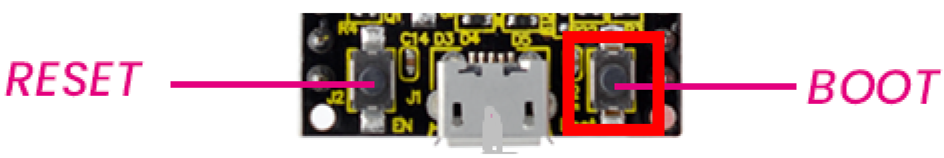

ESP32 firmware installer
This dialog allows installing or updating firmware on ESP32 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.
Port
Silicon Labs CP210x USB to UART Bridge (COM3)
Reload
Firmware
rted with Python/Python_Firmware/esp32-20220117-v1.18.bin
Browse...
Flash mode
From image file (keep)
Quad I/O (qio)
Dual I/O (dio)
Dual Output (dout)
Erase flash before installing
Install
Cancel

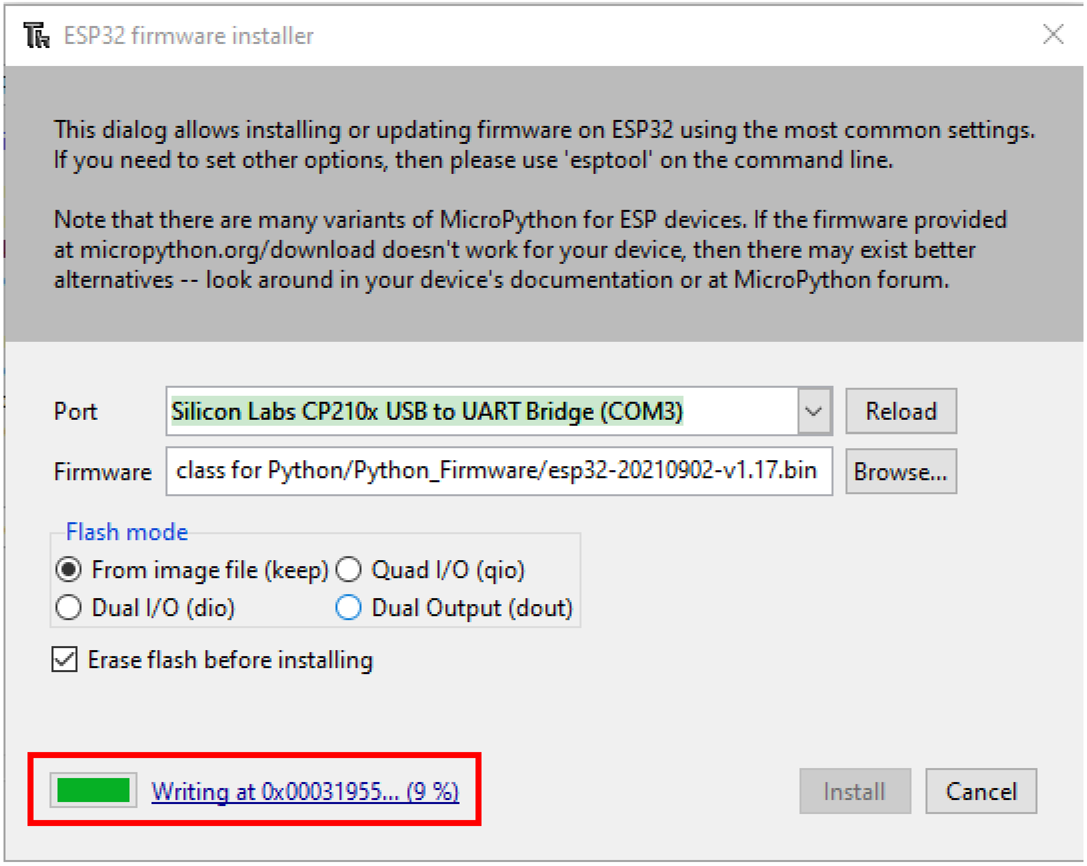

Open
Python_Firmware
Search Python_Firmware
Organize
New folder
OneDrive
This PC
3D Objects
Desktop
Name
Date modified
Type
Size
esp32-20210902-v1.17.bin
2/10/2022 11:23 AM
BIN File
1,492 KB
1
2
File name:
esp32-20210902-v1.17.bin
bin-file
Open
Cancel

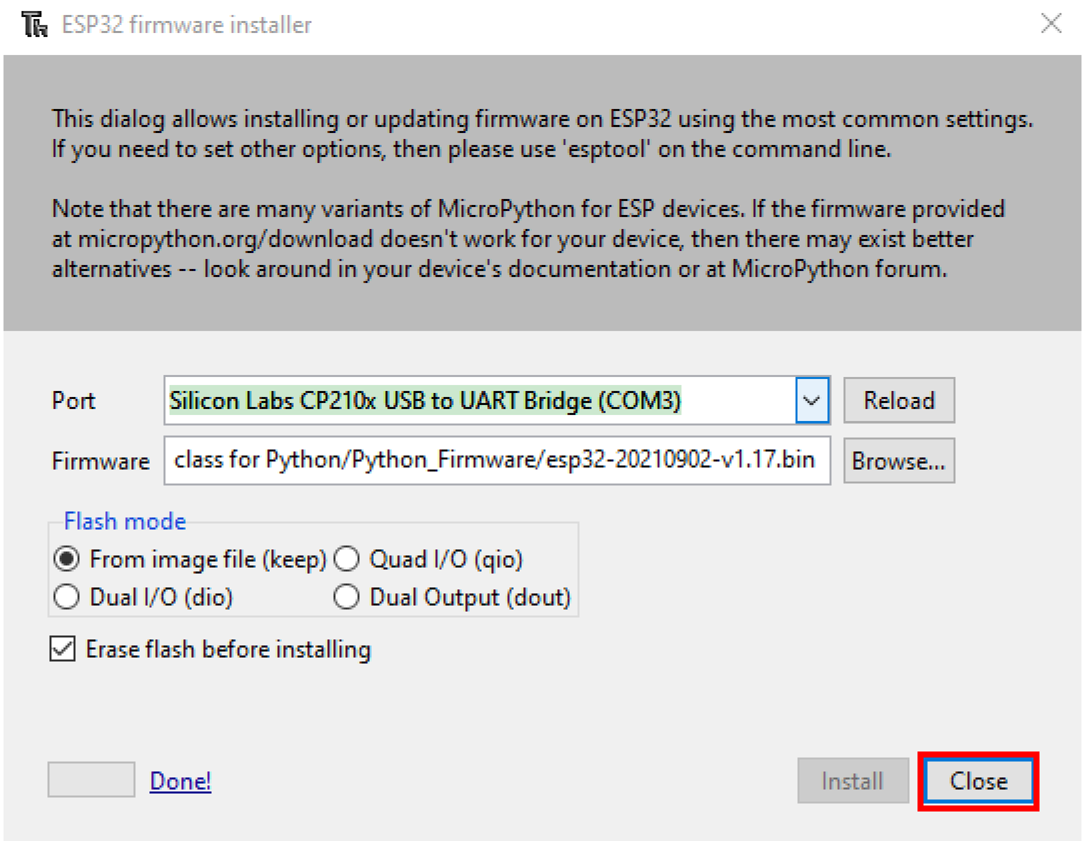


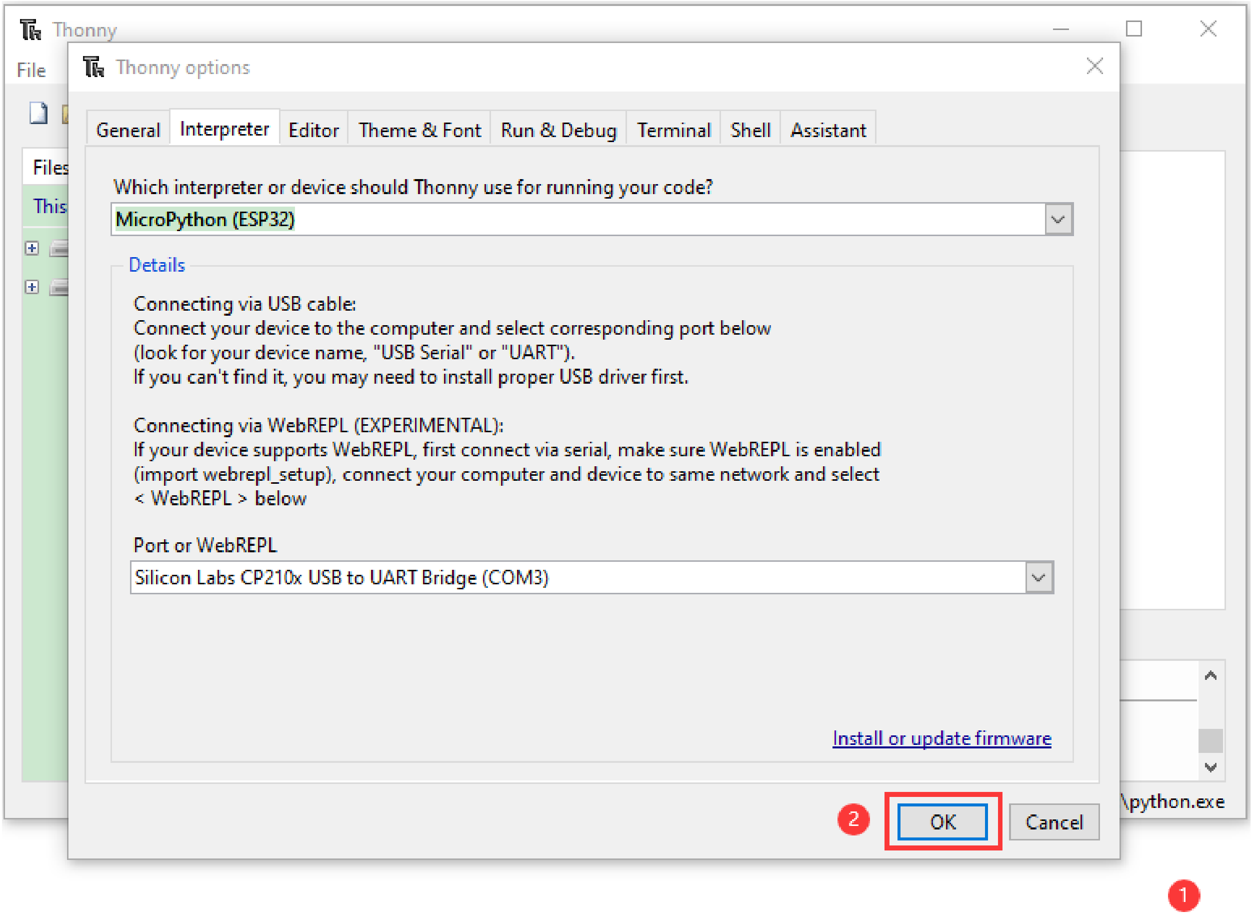
Then click "Close" and "OK".

NoteDuring installation, you can press and hold the Boot button on the ESP32 mainboard. When the upload progress percentage appears, release the button for a while to complete the installation.

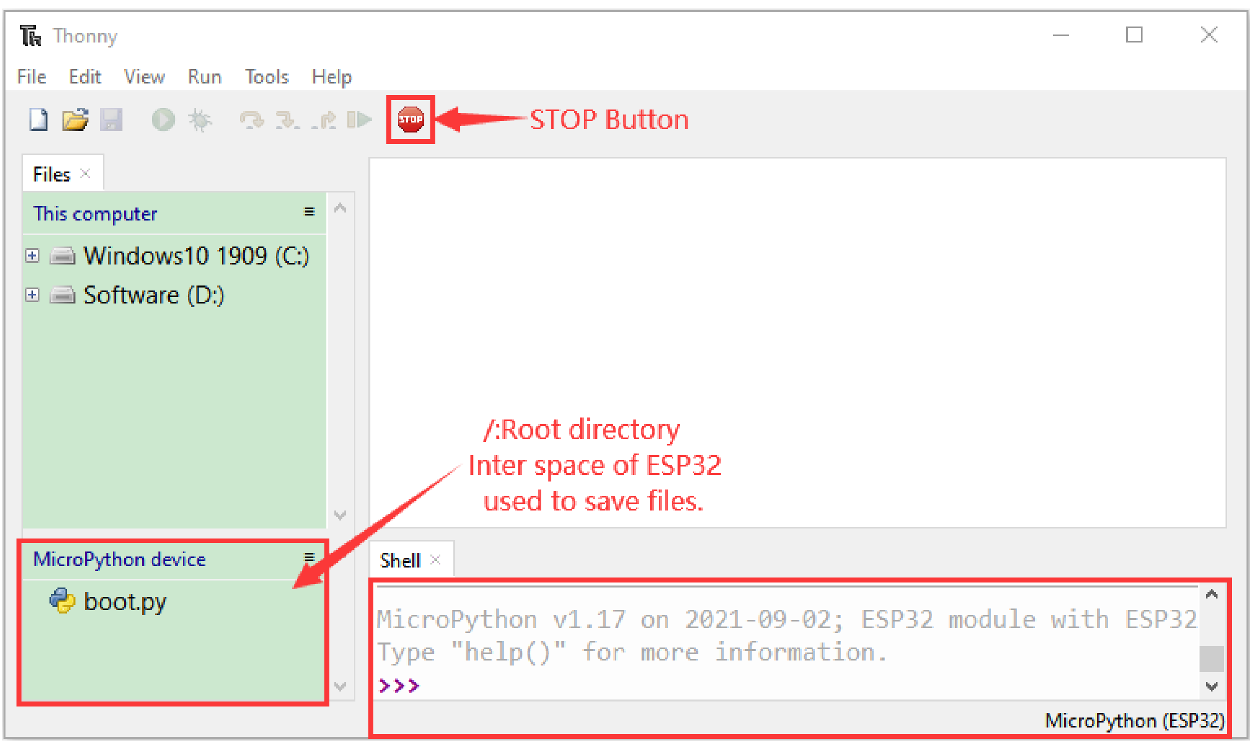



ESP32 firmware installer
This dialog allows installing or updating firmware on ESP32 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.
Port
Silicon Labs CP210x USB to UART Bridge (COM3)
Reload
Firmware
class for Python/Python_Firmware/esp32-20210902-v1.17.bin
Browse...
Flash mode
From image file (keep)
Quad I/O (qio)
Dual I/O (dio)
Dual Output (dout)
Erase flash before installing
Writing at 0x00031955... (9 %)
Install
Cancel


ESP32 firmware installer
This dialog allows installing or updating firmware on ESP32 using the most common settings. If you need to set other options, then please use 'esptool' on the command line.
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.
Port
Silicon Labs CP210x USB to UART Bridge (COM3)
Reload
Firmware
class for Python/Python_Firmware/esp32-20210902-v1.17.bin
Browse...
Flash mode
From image file (keep)
Quad I/O (qio)
Dual I/O (dio)
Dual Output (dout)
Erase flash before installing
Done!
Install
Close



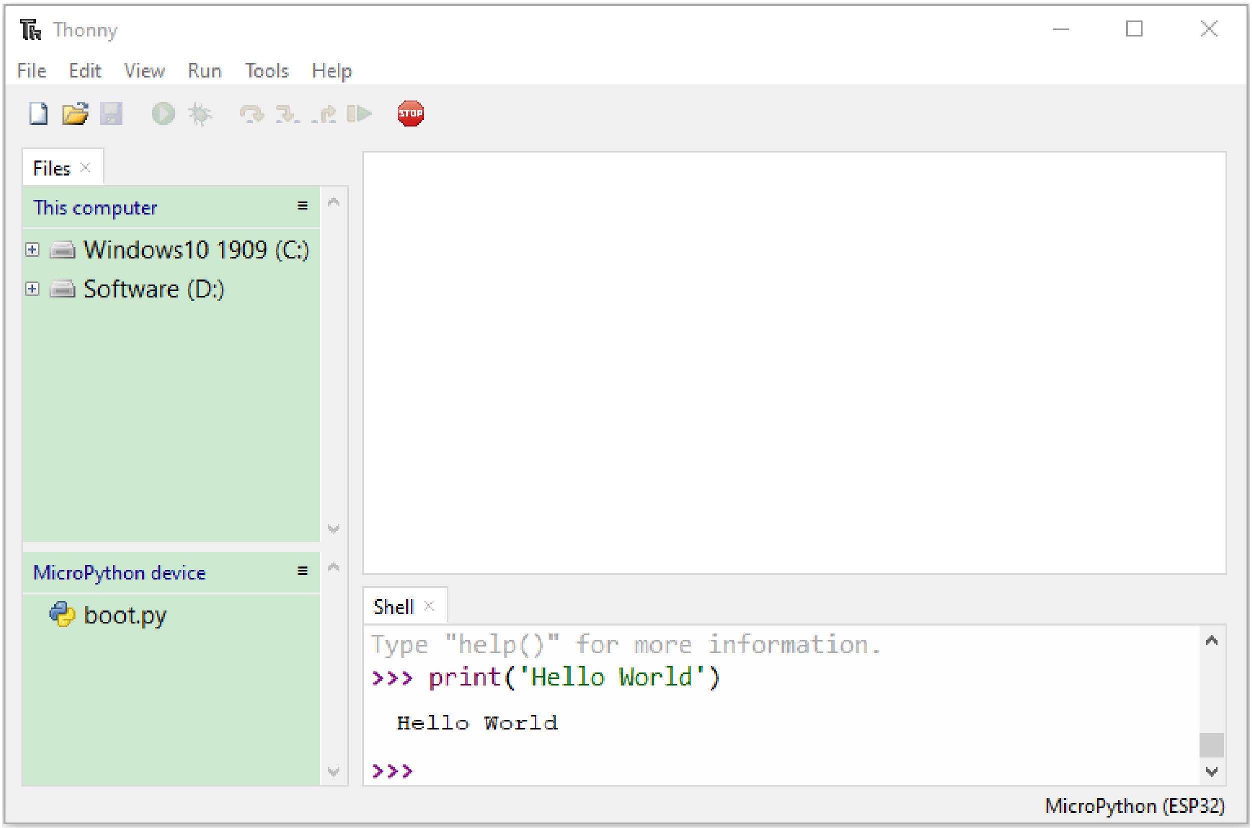
Turn off all windows and turn to the main page and click "STOP".



5.1.4 4. Test Code:

Test the Shell commander

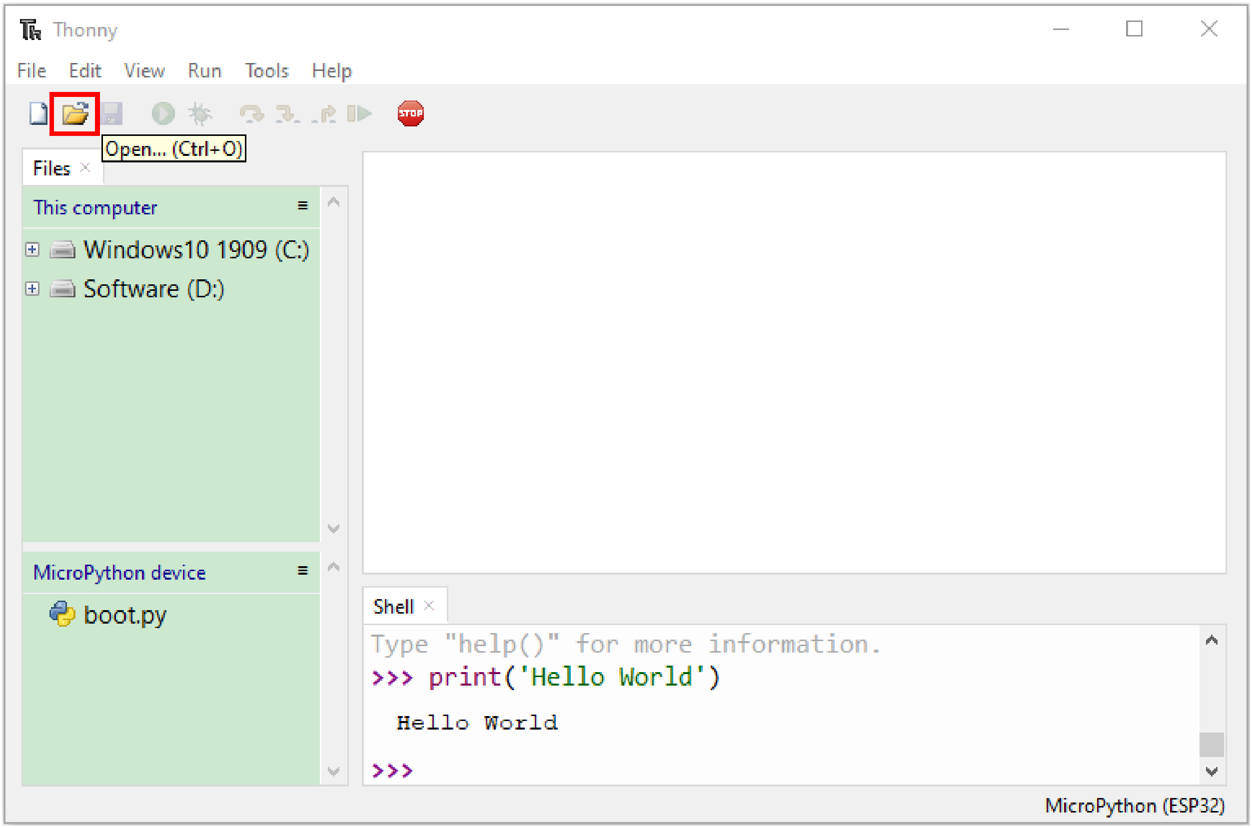
Input print('hello world') in the"Shell"and press "**Enter**".



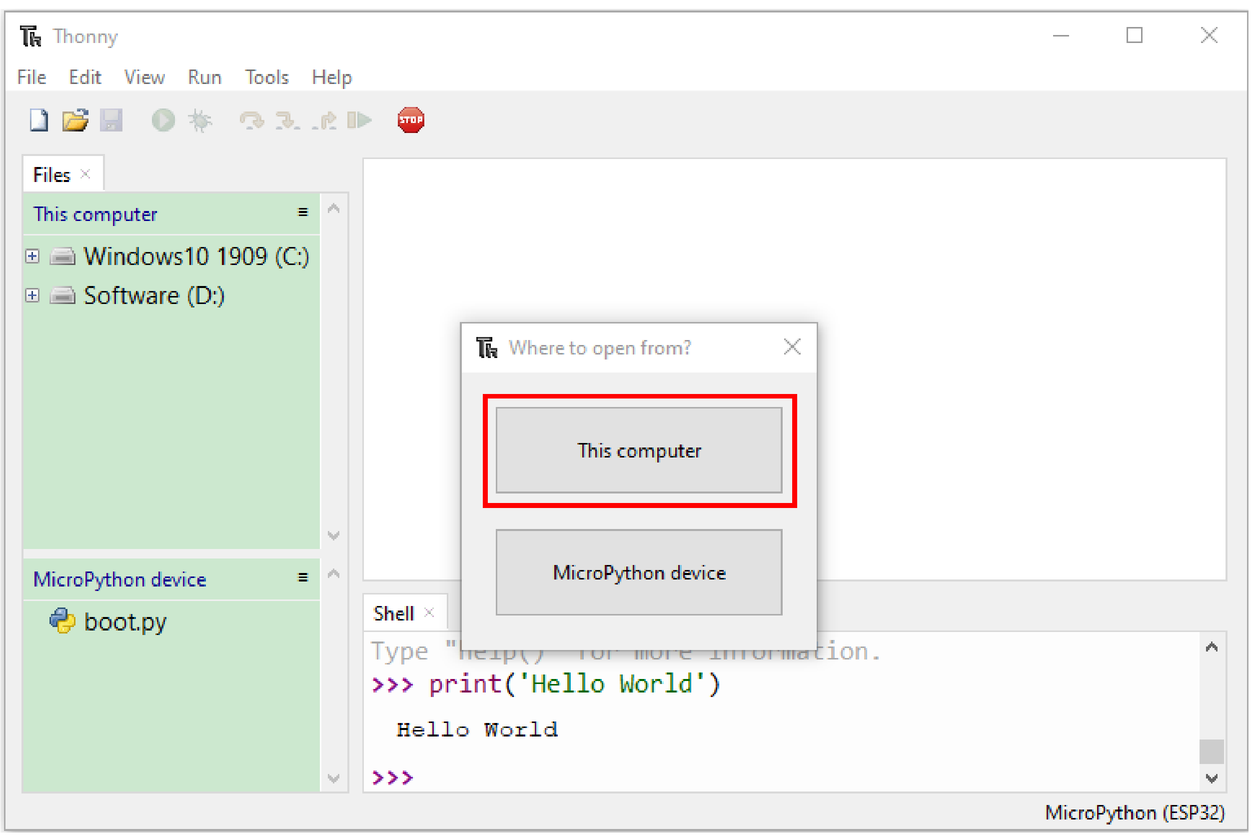
Run the test code(online):

Connect the ESP32 to your PC. Users can program and debug programs with Thonny.

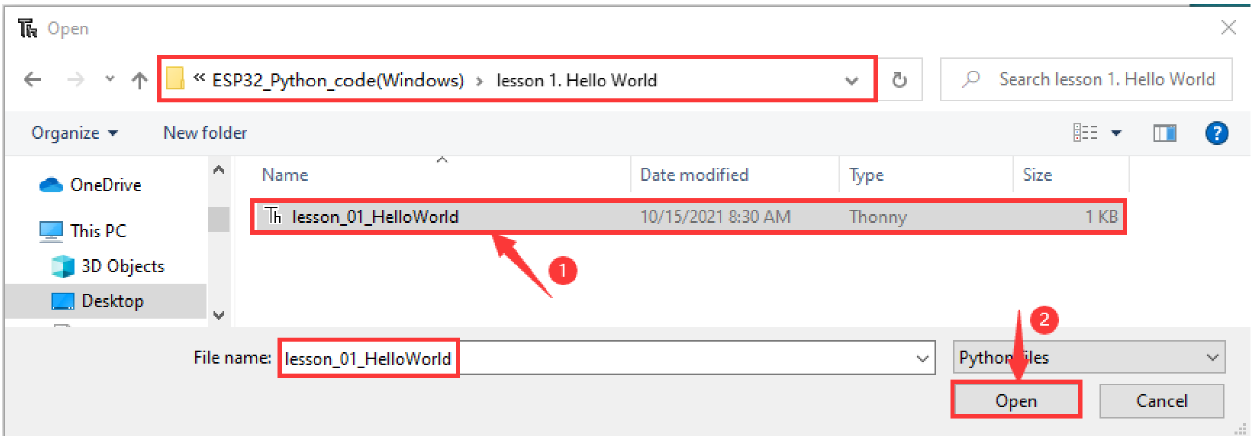
Open Thonny and click "**Open**".



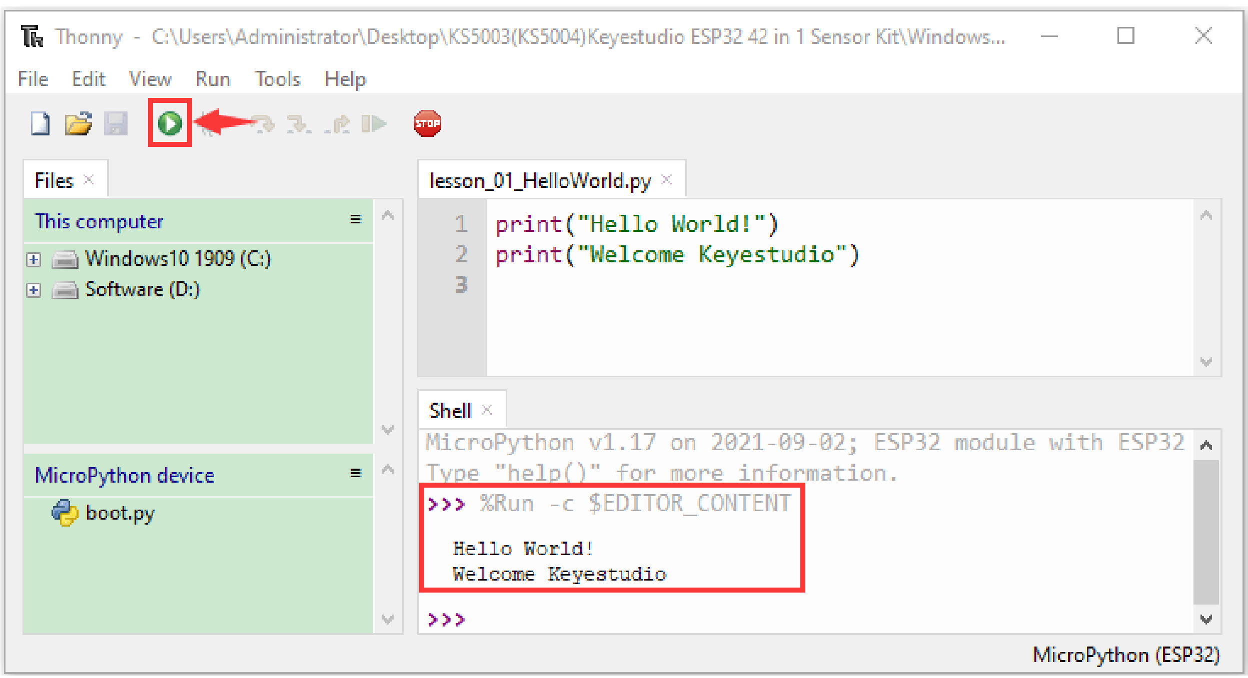
When a new window pops up, click "**This computer**".



Select the file "**lesson_01_HelloWorld.py**".



Click ,"**Hello World**"will be printed in the "**Shell**" monitor.



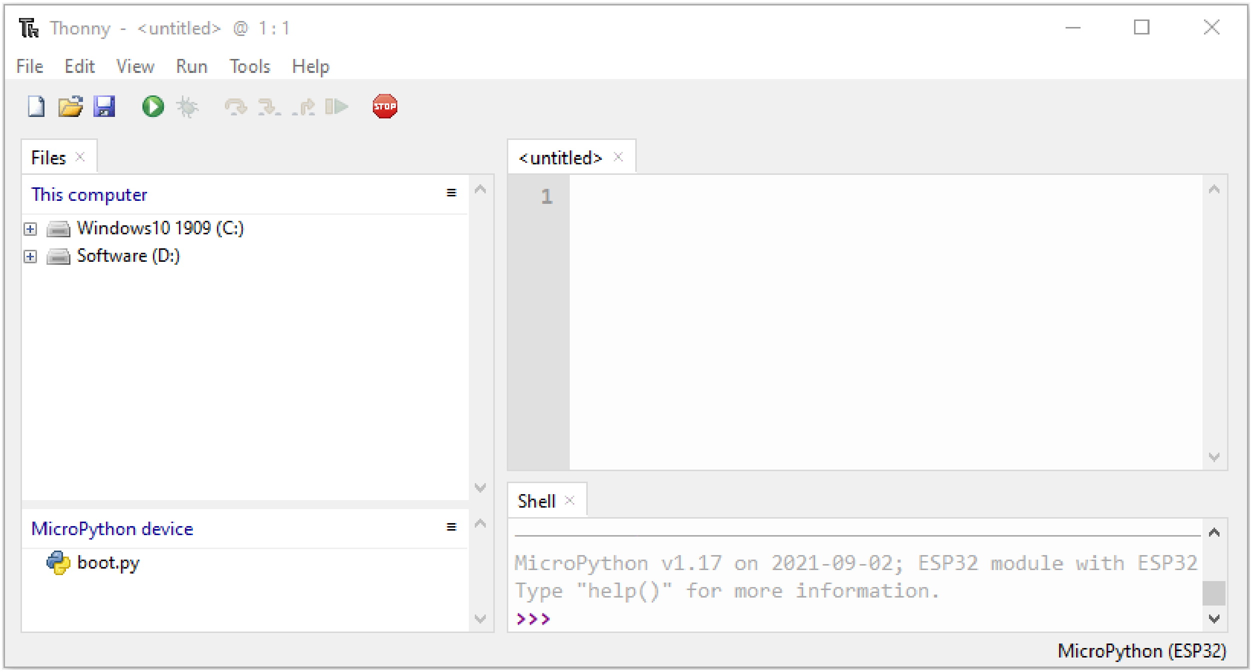
Note: Press the reset button to reboot

Run the test code(offline):

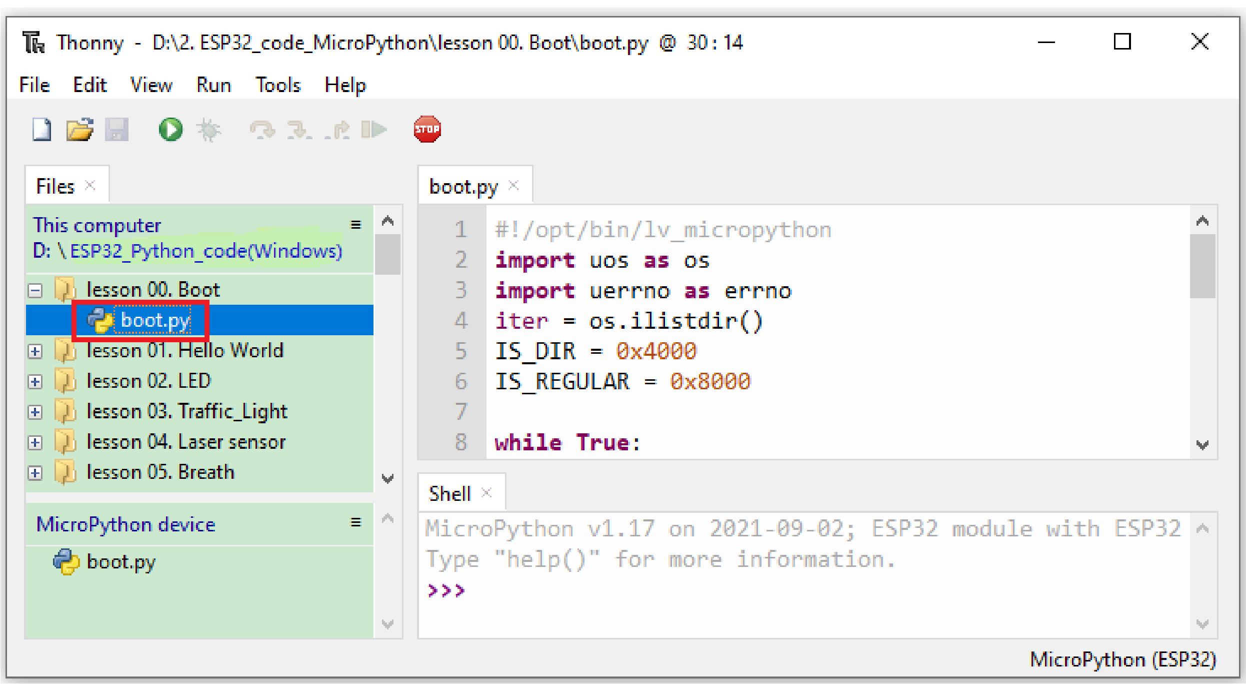
After rebooting the ESP32, run the boot.py file under the root directory first then run your code file.

So, we need to add a guide program to run the code of users.

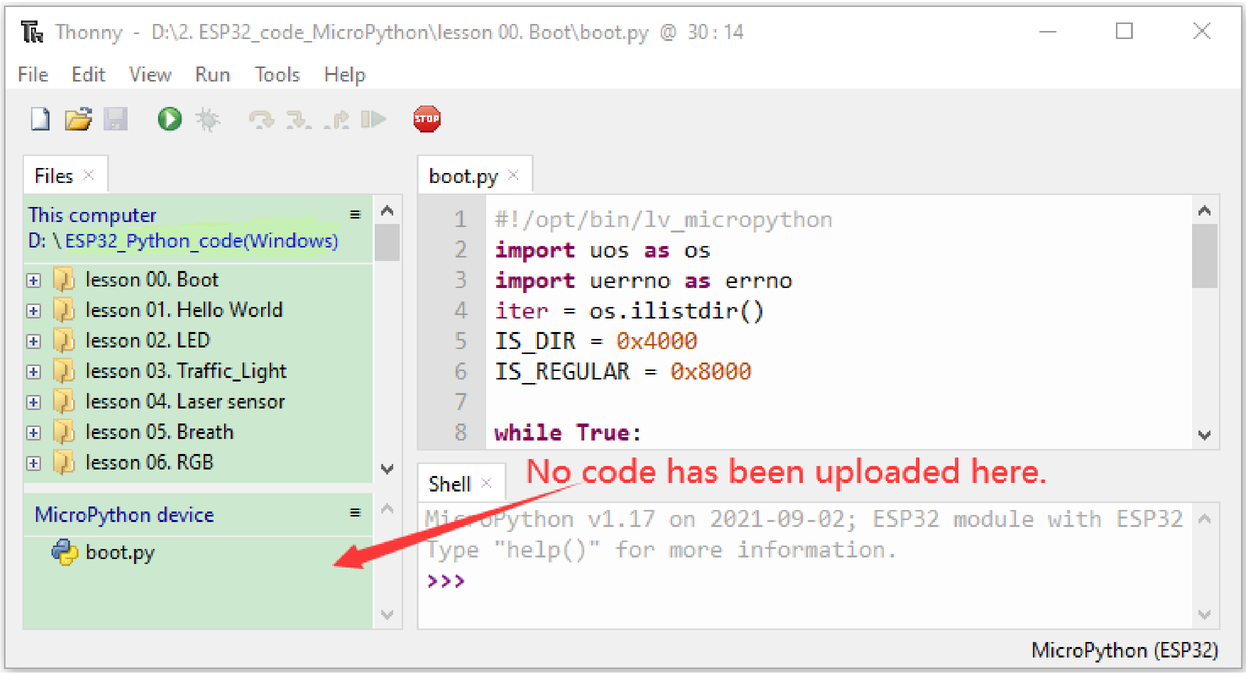
Move the file "**ESP32_code_MicroPython(Windows)**" to the disk(D)the route is "D:/ESP32_code_MicroPython(Windows)".

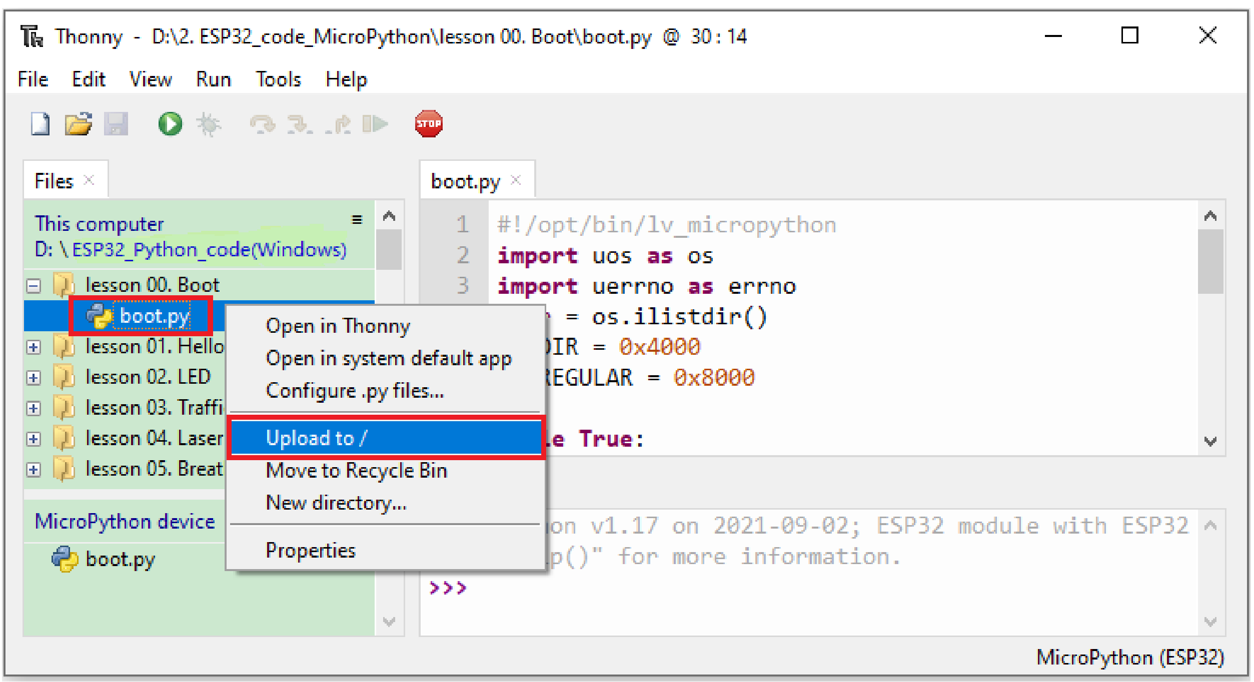


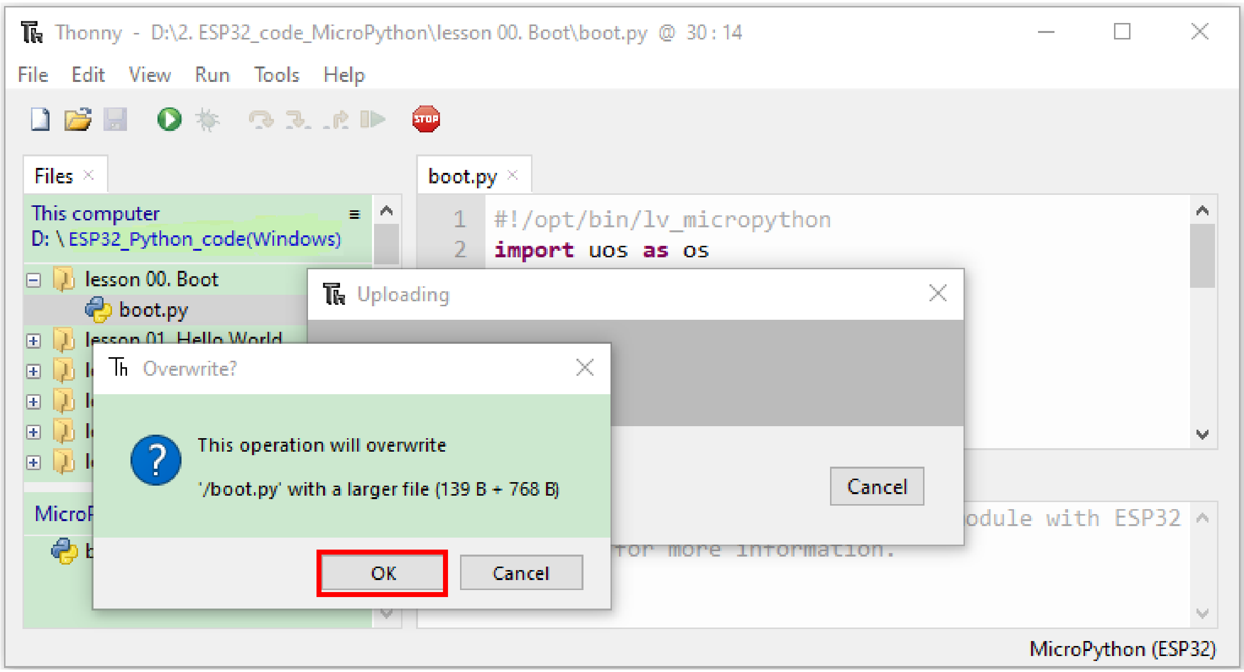
Click lesson 00. Boot and double-click boot.py, then the code under MicroPython device can run offline.



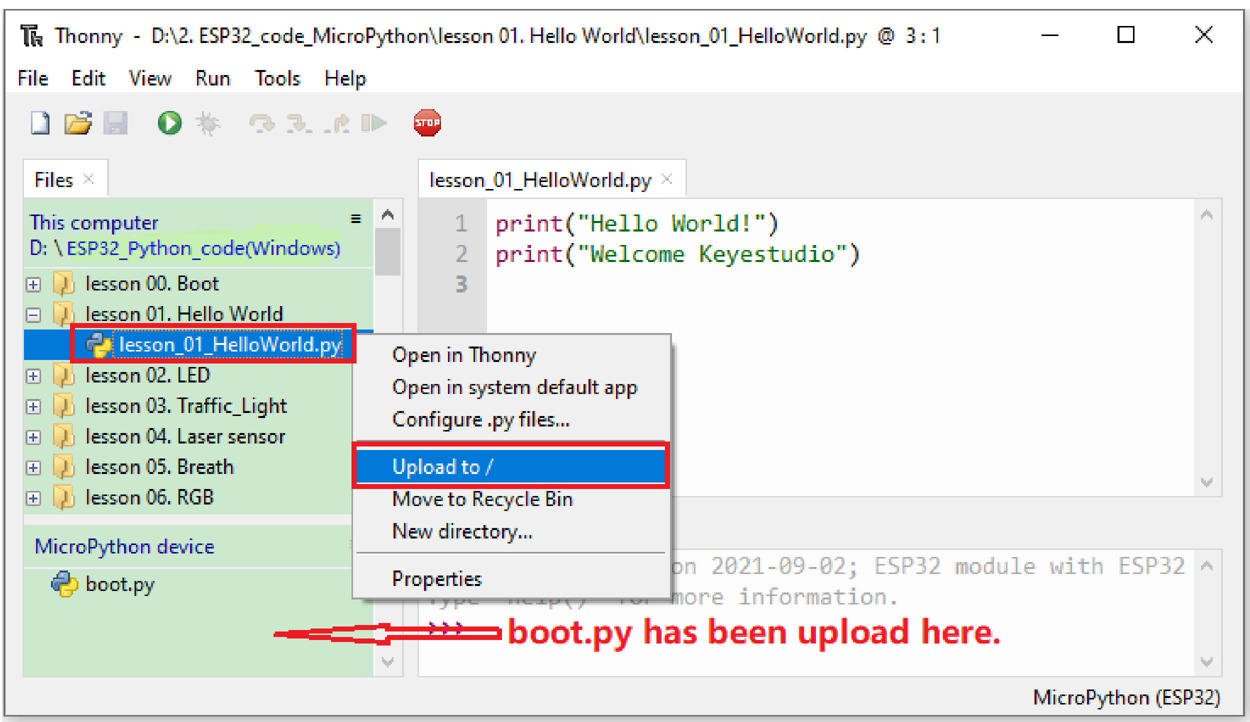
If you want to run the code offline, you nee to upload boot.py and program code to MicroPython device, then press the ESP32's reset button. We will take the lesson 00 and lesson 01 as an example. Select "**boot.py**" and right-click "**Upload to/**".


Thonny - D:\2. ESP32_code_MicroPython\lesson 00. Boot\boot.py @ 30 : 14
File Edit View Run Tools Help
Files
This computer
D: \ ESP32_Python_code(Windows)
lesson 00. Boot
lesson 01. Hello World
lesson 02. LED
lesson 03. Traffic_Light
lesson 04. Laser sensor
lesson 05. Breath
lesson 06. RGB
MicroPython device
boot.py
boot.py
#!/opt/bin/lv_micropython
import uos as os
import uerrno as errno
iter = os.ilistdir()
IS_DIR = 0x4000
IS_REGULAR = 0x8000
while True:
No code has been uploaded here.
Shell
MicroPython v1.17 on 2021-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>>
MicroPython (ESP32)

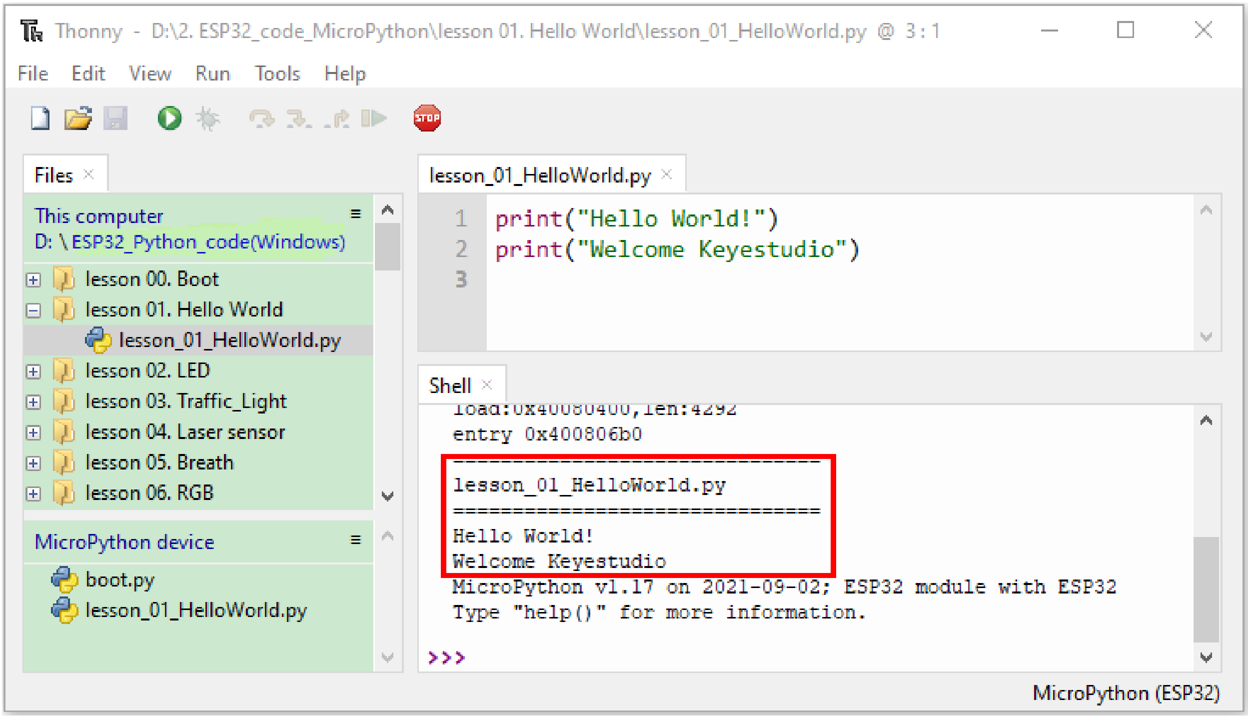

Thonny - D:\2. ESP32_code_MicroPython\lesson 00. Boot\boot.py @ 30 : 14
File Edit View Run Tools Help
Files
This computer
D: \ ESP32_Python_code(Windows)
lesson 00. Boot
boot.py
Open in Thonny
Open in system default app
Configure .py files...
Upload to /
Move to Recycle Bin
New directory...
Properties
MicroPython device
boot.py
MicroPython (ESP32)



Similarly, upload the lesson 01.HelloWorld.py file to the "**MicroPython device**".



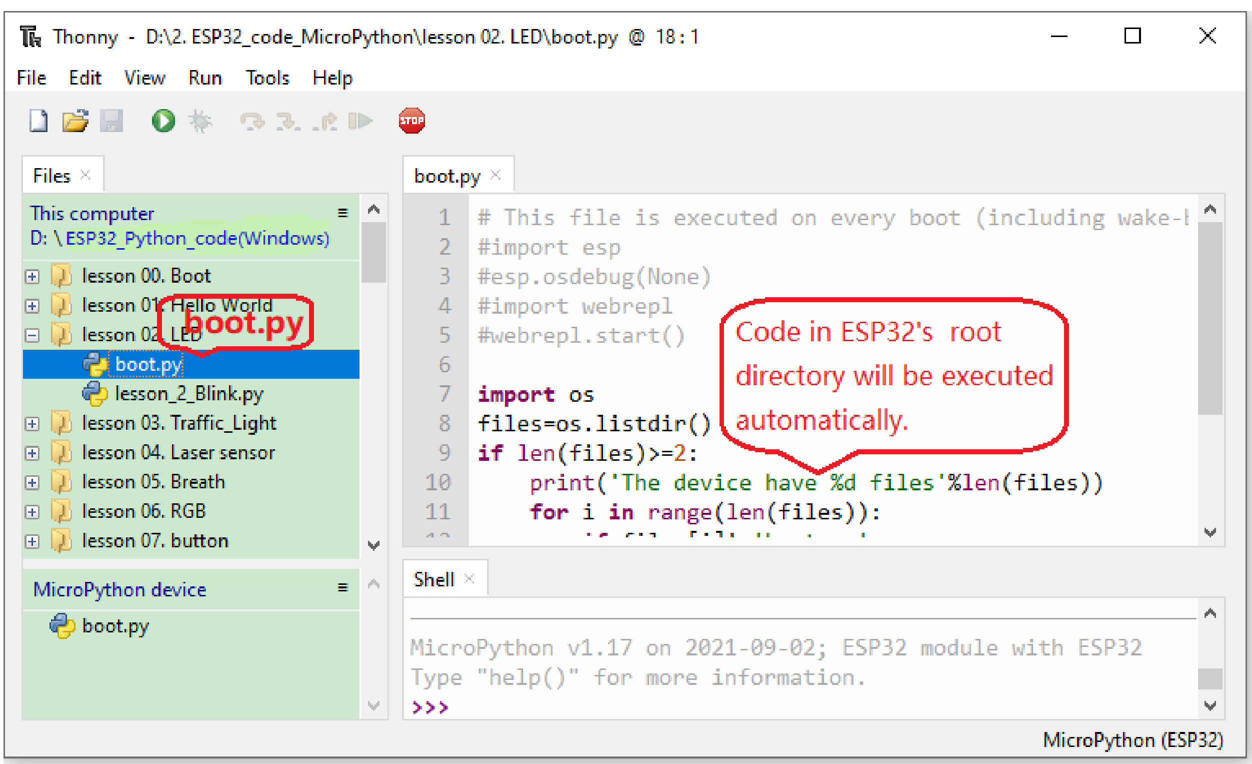
Press the Reset button, you will view code running in the "**Shell**" monitor.



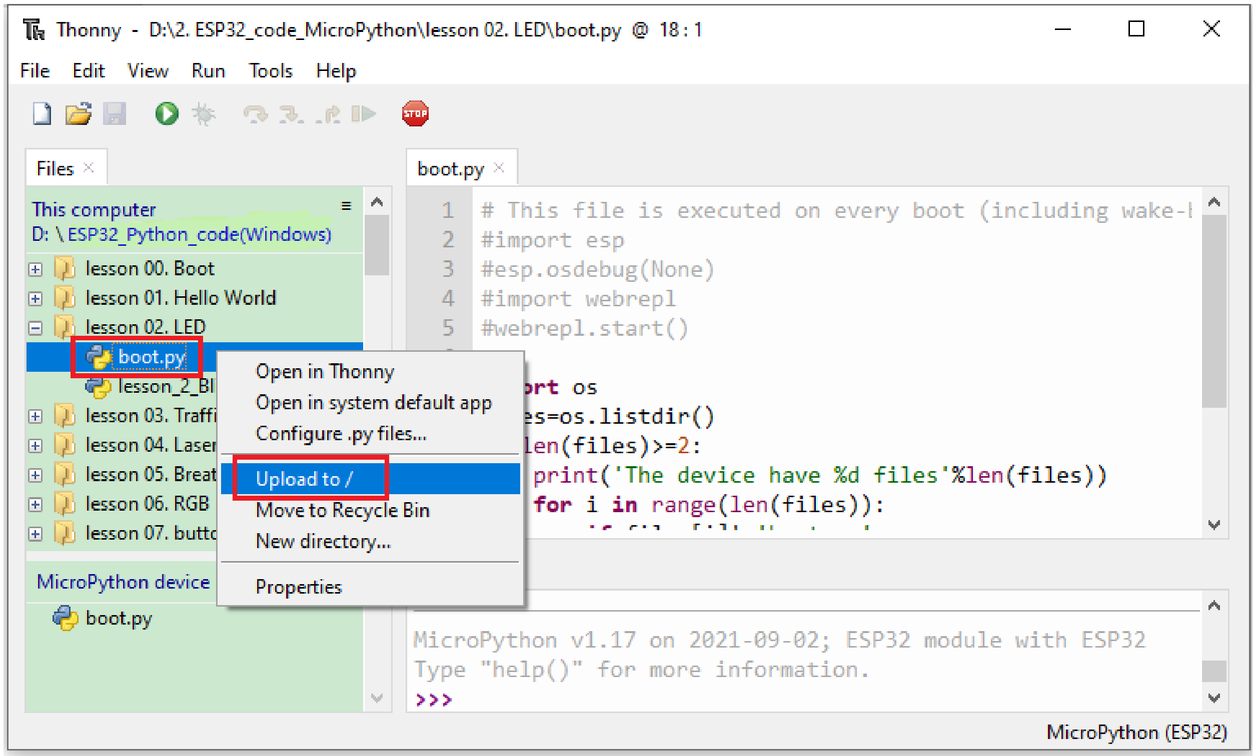
5.1.5 5. Thonny Common Operation:

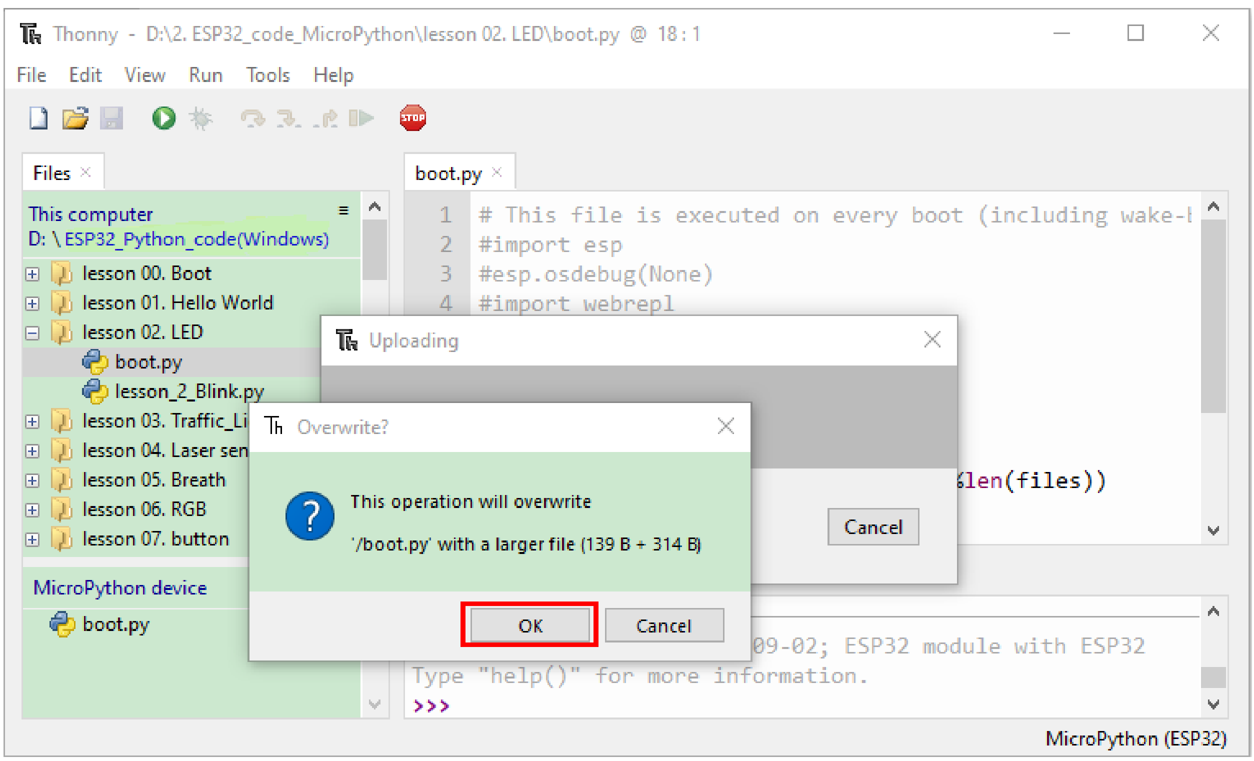
Upload the code to the ESP32

We take the boot.py as an example. If we add a "**boot.py**" in each code directory, reboot the ESP32, the boot.py will run first.



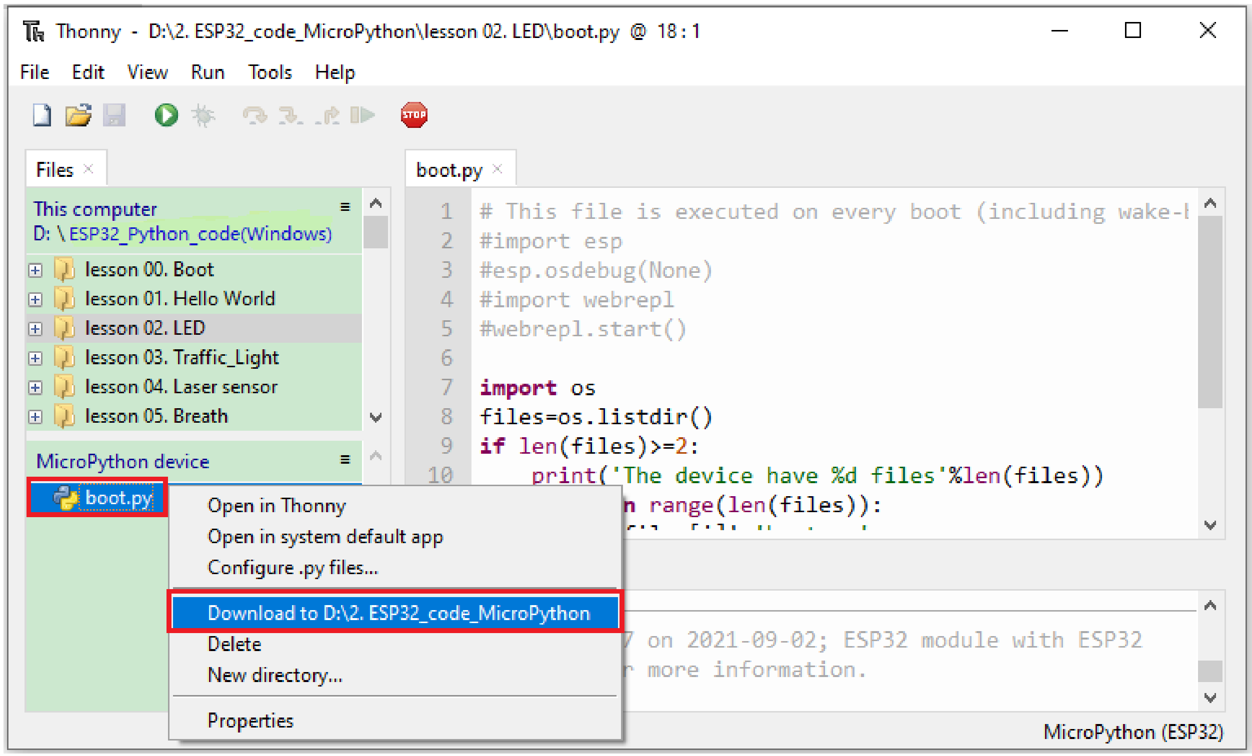
Select "**boot.py**"in the file lesson 02. LED, right-click to select"**Upload to /**". Then the code will be uploaded to the root directory of the ESP32 and click "**OK**".





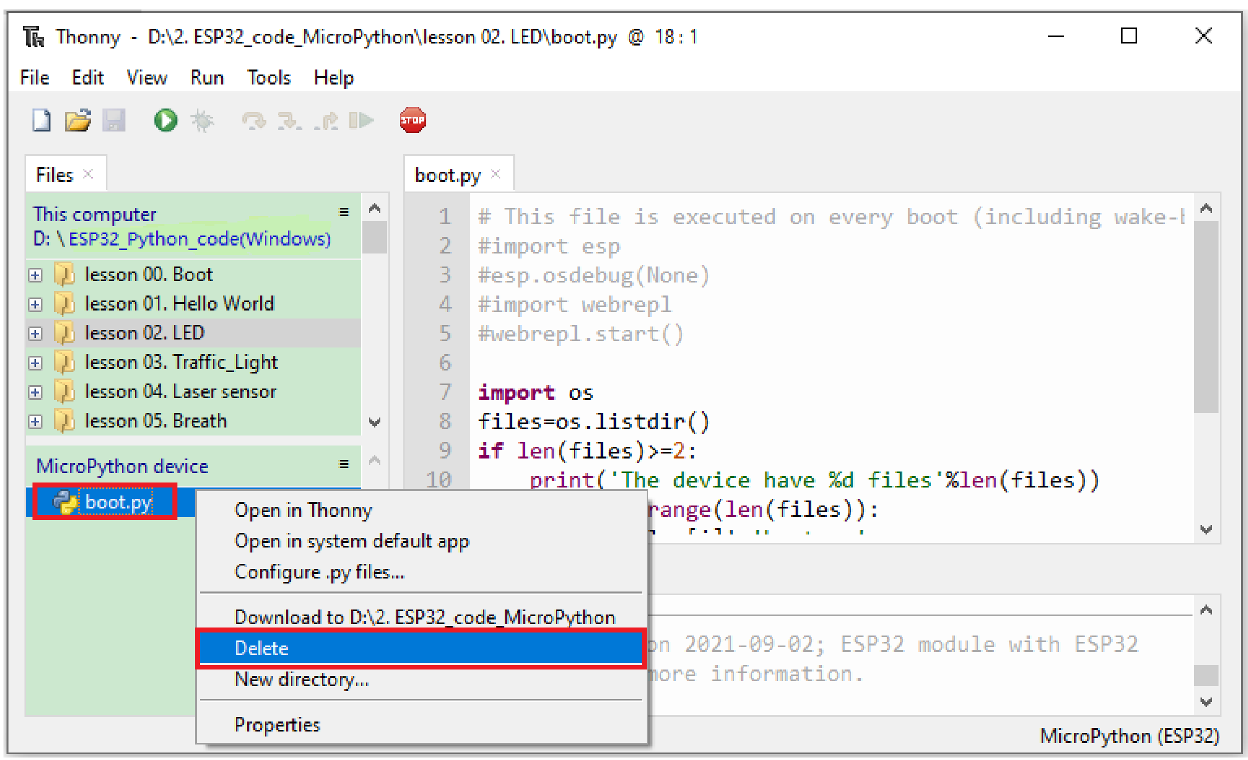
Download the code to your PC

MicroPython device "**boot.py**", then right-click "**Download to...**".

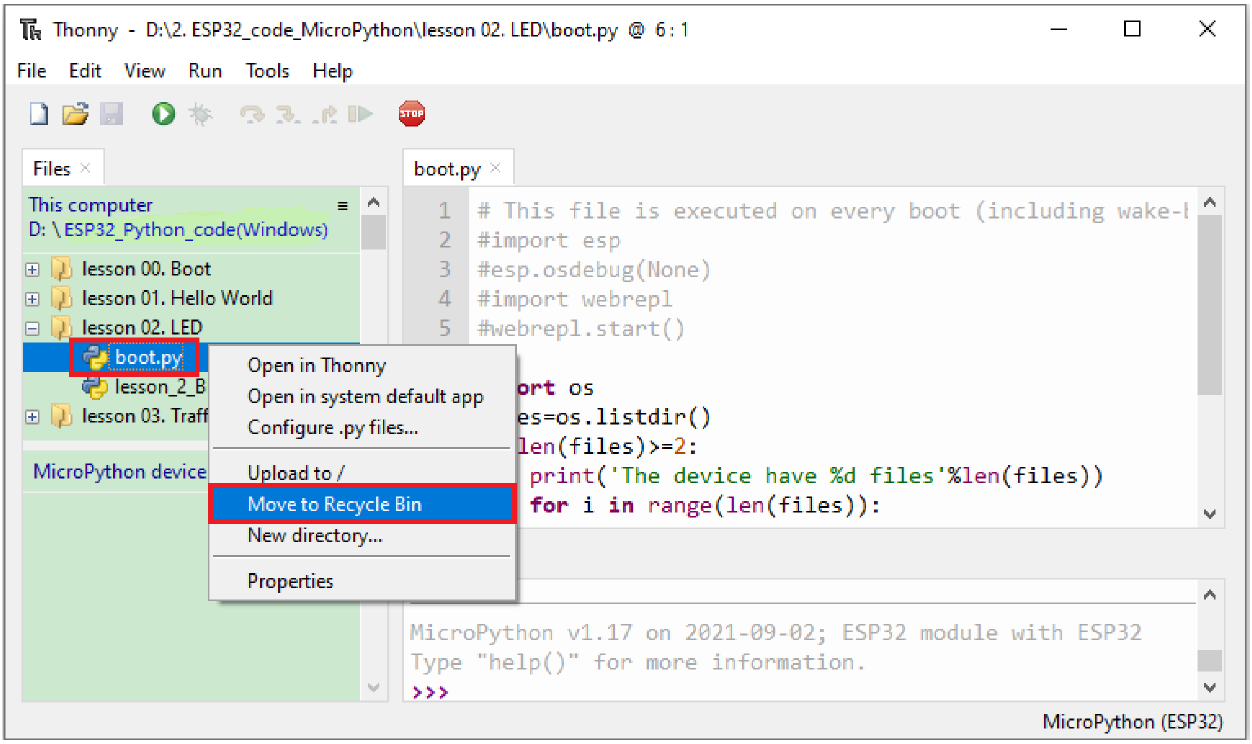


Delete files of the ESP32

For example, click "**boot.py**" in the MicroPython device and right-click "**Delete**".

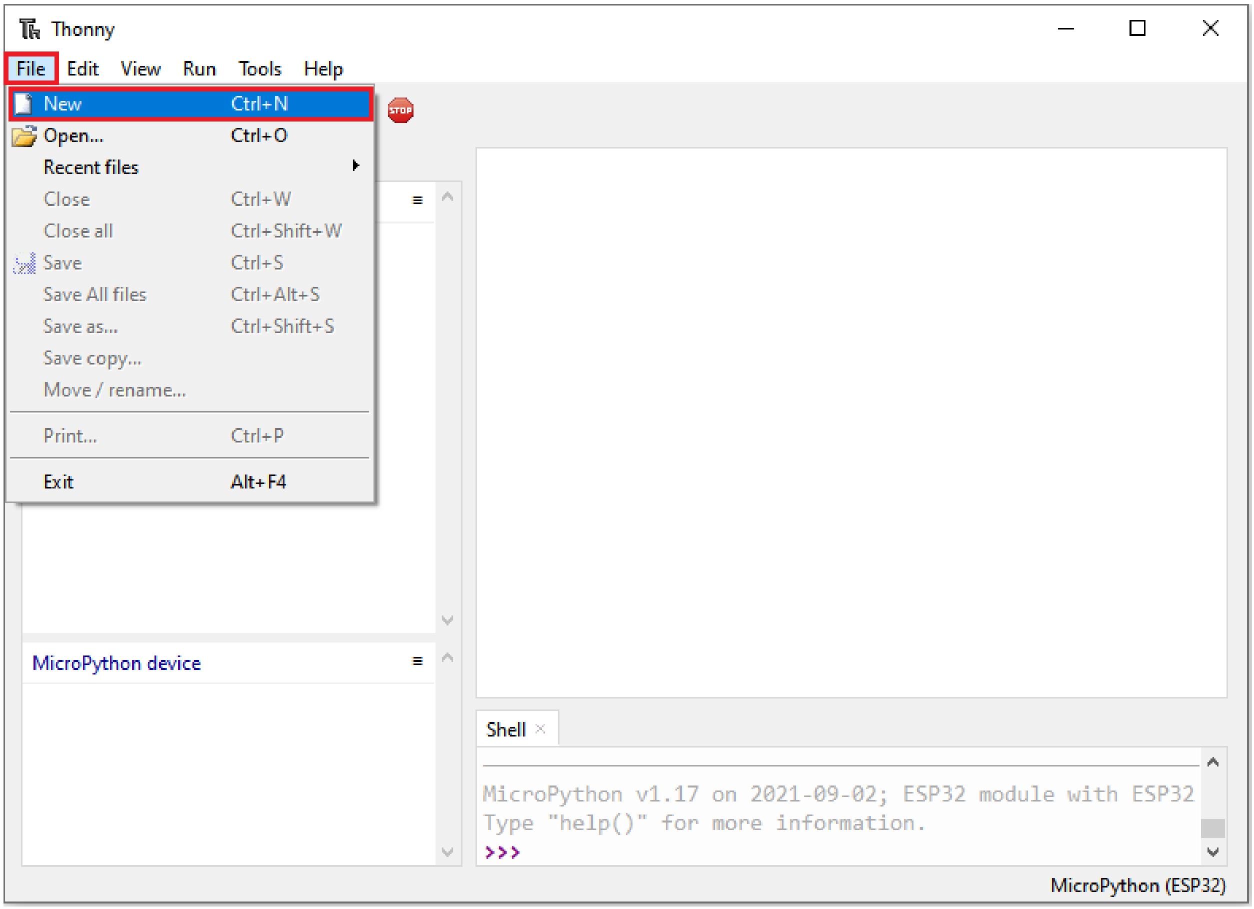


Select "**boot.py**" in the lesson 02. LED folder, right-click "**Move to Recycle Bin**" to delete it.

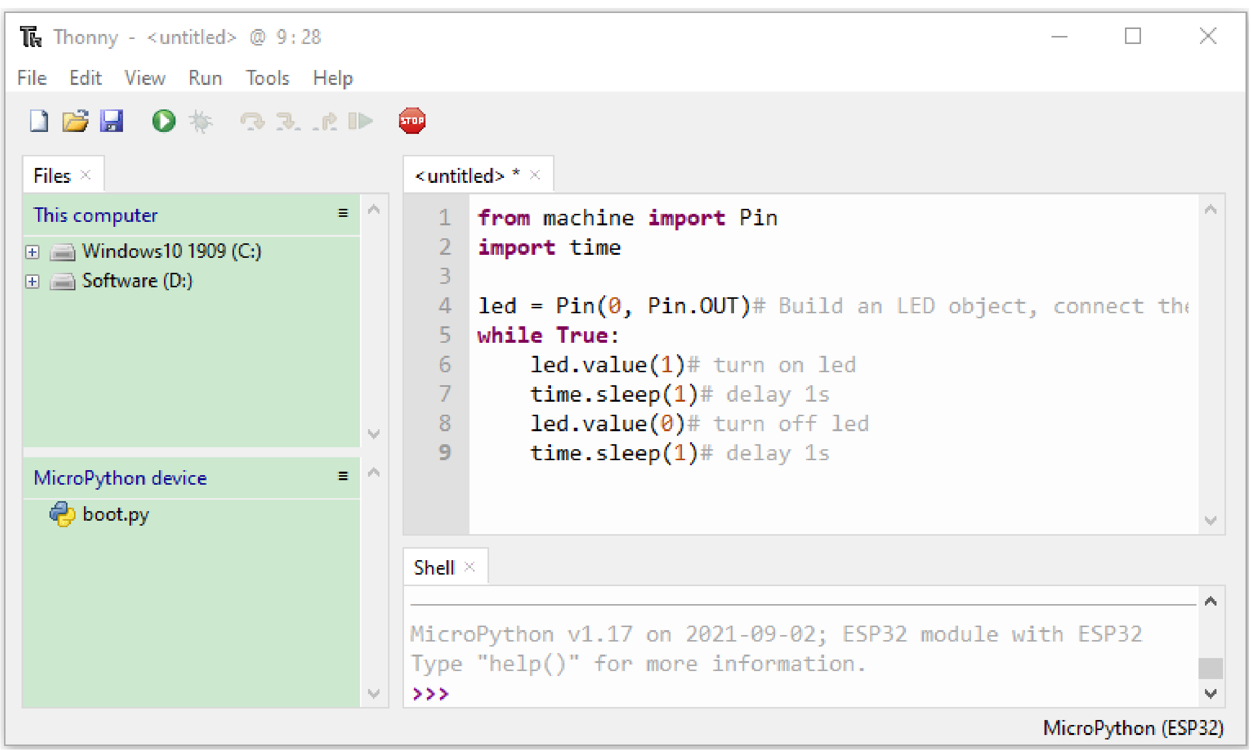


Create and save code

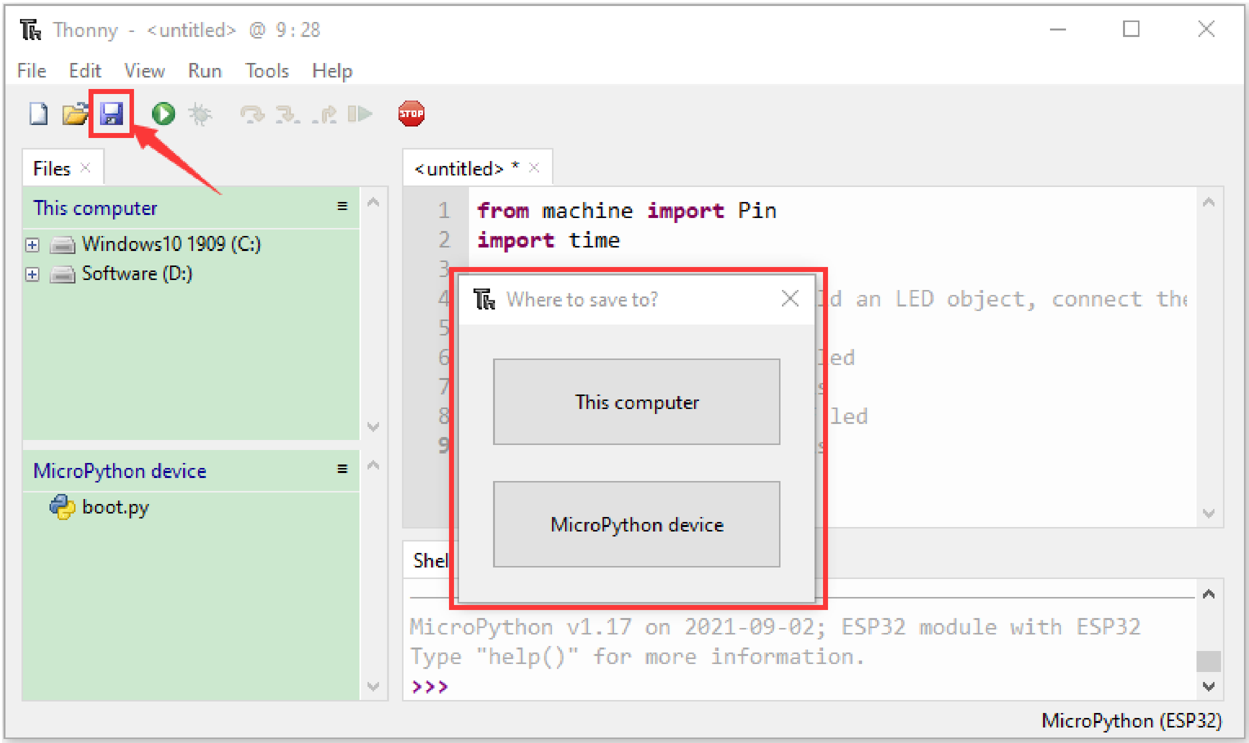
Click “**File**” → “**New**” to create and edit code.



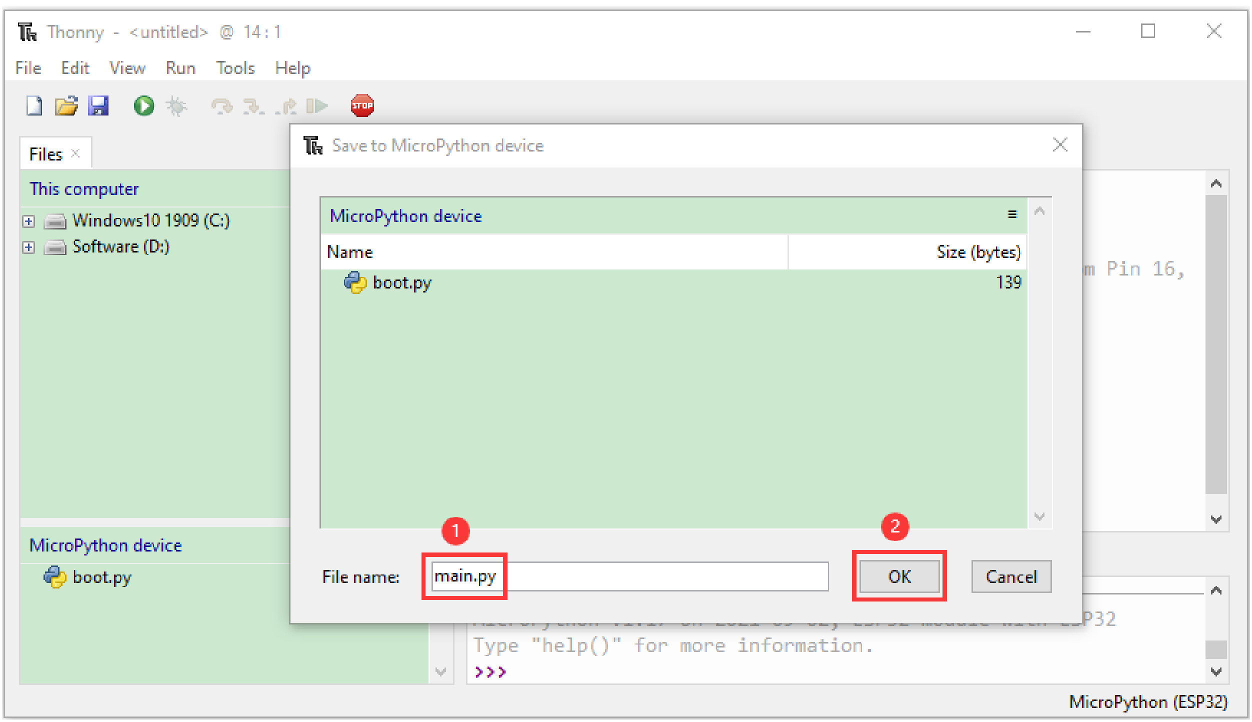
Enter the code in the new file. We take the lesson 02. LED.py as an example.



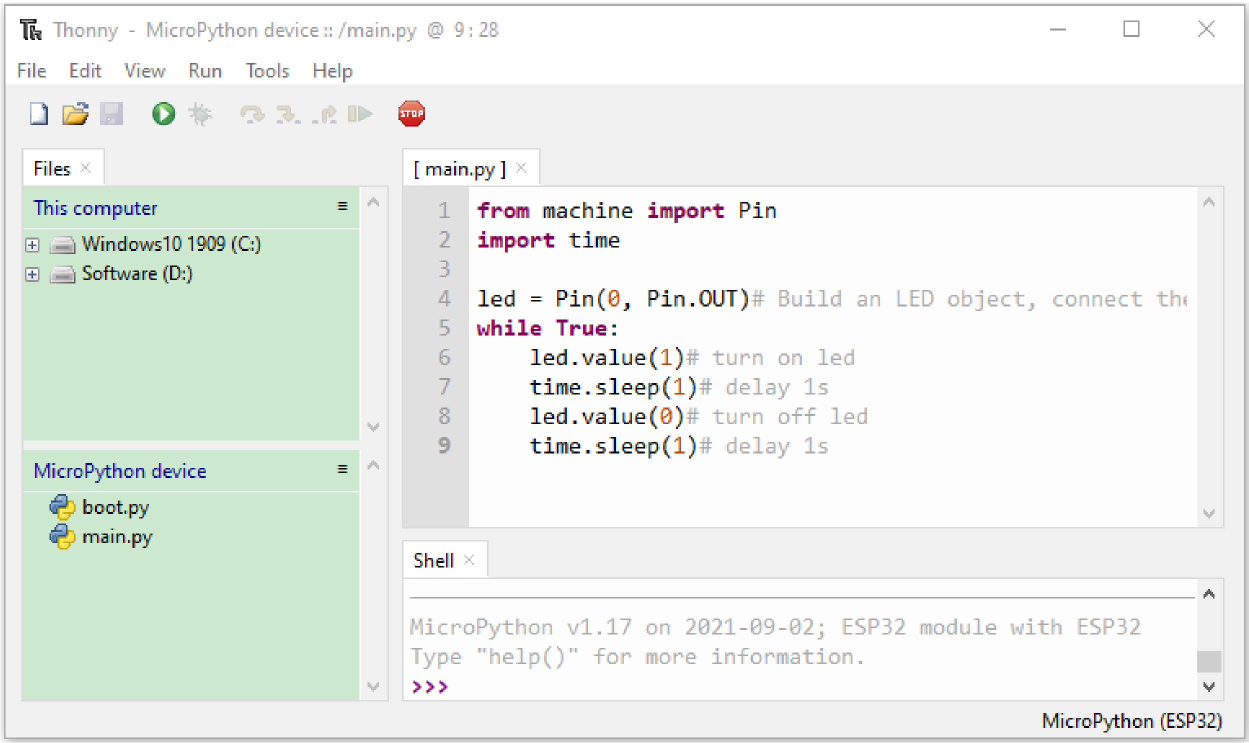
Click  to save the code to your PC or the ESP32.



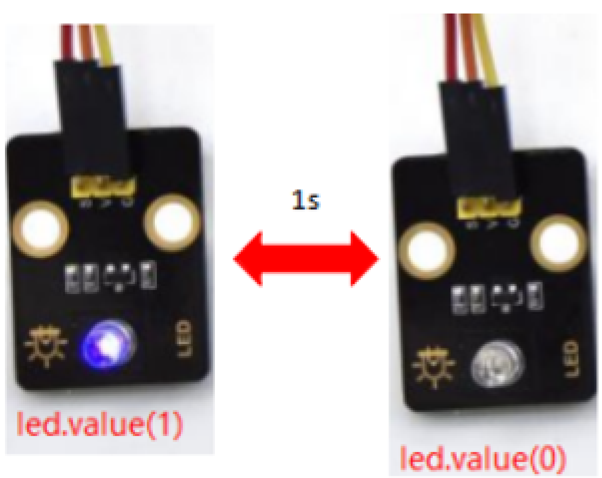
Select MicroPython device and enter main.py in the new page and click "OK".



Then the code will be uploaded to the ESP32.



Disconnect the USB cable and connect it, you can see the effect of the LED flashing continuously in the circuit on a cycle.



5.2 2. Single Sensor/Experiment Projects

When we get the kit, we can see that there are 42 sensors/modules in the kit, which contain the corresponding ESP32 mainboard, ESP32 Expansion Board and wirings. Here, we will connect the 42 sensors individually to the ESP32 mainboard and the ESP32 Expansion Board using a wiring. Then run the corresponding test code to test the function of each sensor separately. Our next lesson is to study the principles of individual modules/sensors from simple to complex as well as some extended applications of sensors to consolidate and deepen our understanding of the kits.

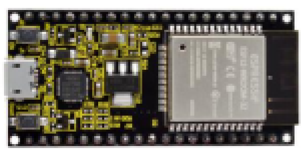
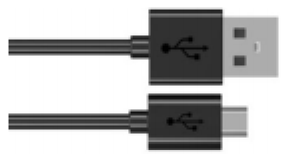
Note : When connecting the module/sensor wirings in the experiment, the wiring method and position must be followed in the document. What's more, do not misconnect the power supply and signal pin, otherwise there may be no experimental results or damage to the modules/sensors.

5.2.1 Project 1: Hello World

Overview

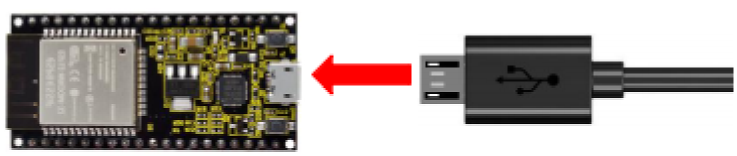
For ESP32 beginners, we will start with some simple things. In this project, you only need a ESP32 mainboard, a USB cable and a computer to complete the "Hello World!" project, which is a test of communication between the ESP32 mainboard and the computer as well as a primary project.

Components

	
ESP32*1	USB Cable*1

Wiring Diagram

In this project, we will use a USB cable to connect the ESP32 to a computer.



Running code online

To run the ESP32 online, you need to connect the ESP32 to the computer, which allows you to compile or debug programs using Thonny software.

Advantages:

1). You can use the Thonny software to compile or debug programs.

2).Through the "Shell" window, you can view error messages and output results generated during the running of the program as well as query related function information online to help improve the program.

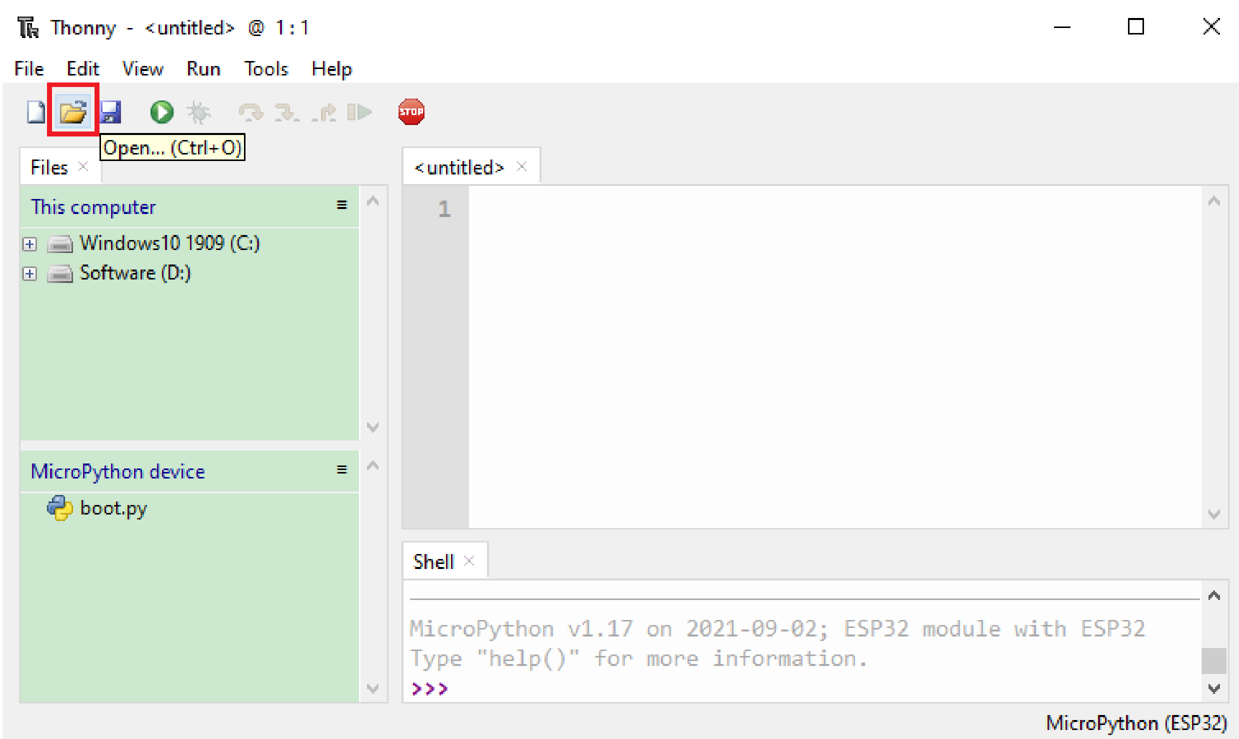
Disadvantages:

1). To run the ESP32 online, you must connect the ESP32 to a computer and run it with the Thonny software.

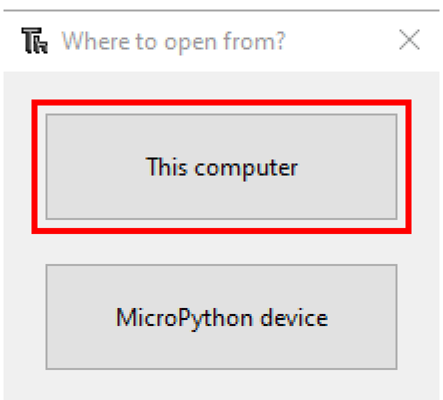
2). If the ESP32 is disconnected from the computer , when they reconnect, the program won't run again.

Basic Operation:

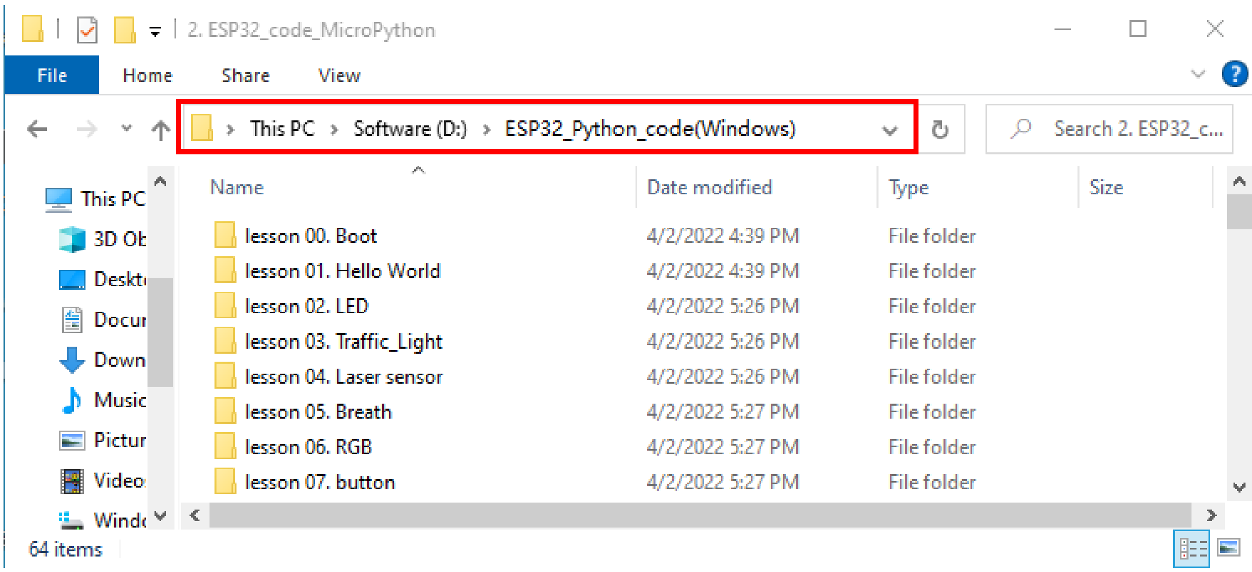
1). Open Thonny and click"Open…".

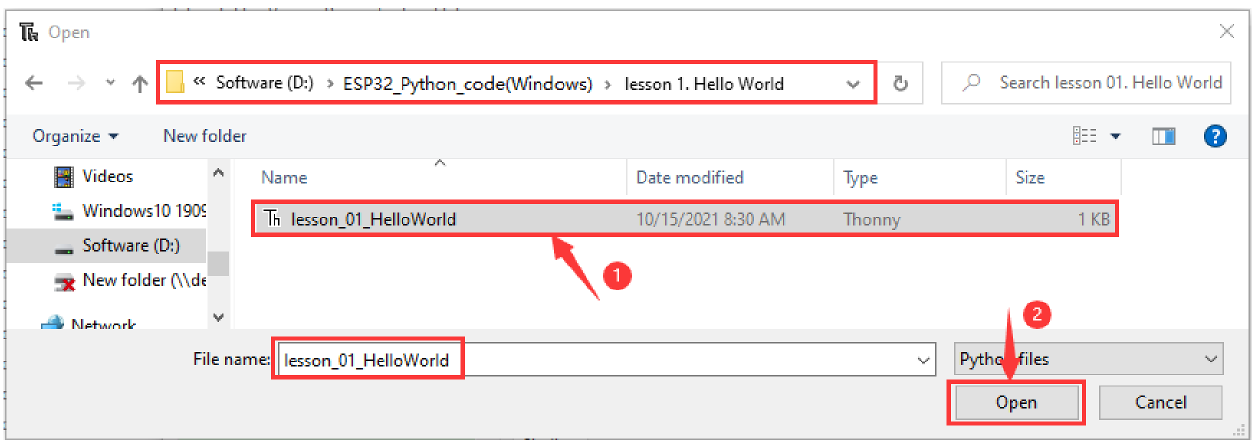


2). Click“This computer”in the new pop-up window.

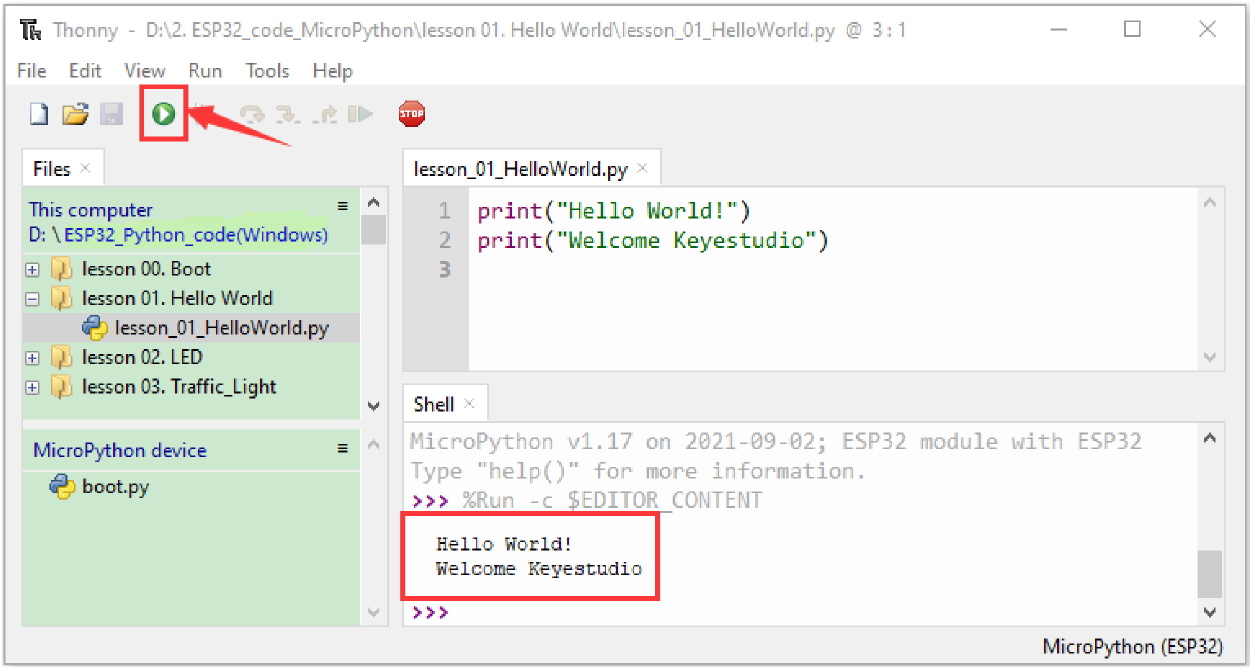


3). In the new dialog boxselect“Project_01_HelloWorld.py”,click“Open”. The code used in this tutorial is saved in the file “**…\6.Codes\ESP32_Python_code(Windows)**”. You can move the code to anywhere. for example, we can save the file “ESP32_Python_code(Windows)” in the Disk(D), the route is **D:\ESP32_Python_code(Windows)**.



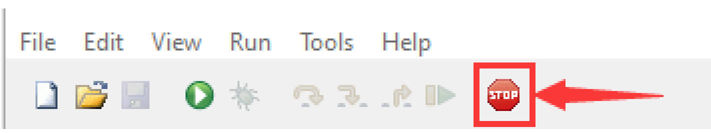


3). Click “Run current script”to execute the program“Hello World!”, “Welcome Keyestudio” , which will be printed in the“Shell”window.



Exit running online

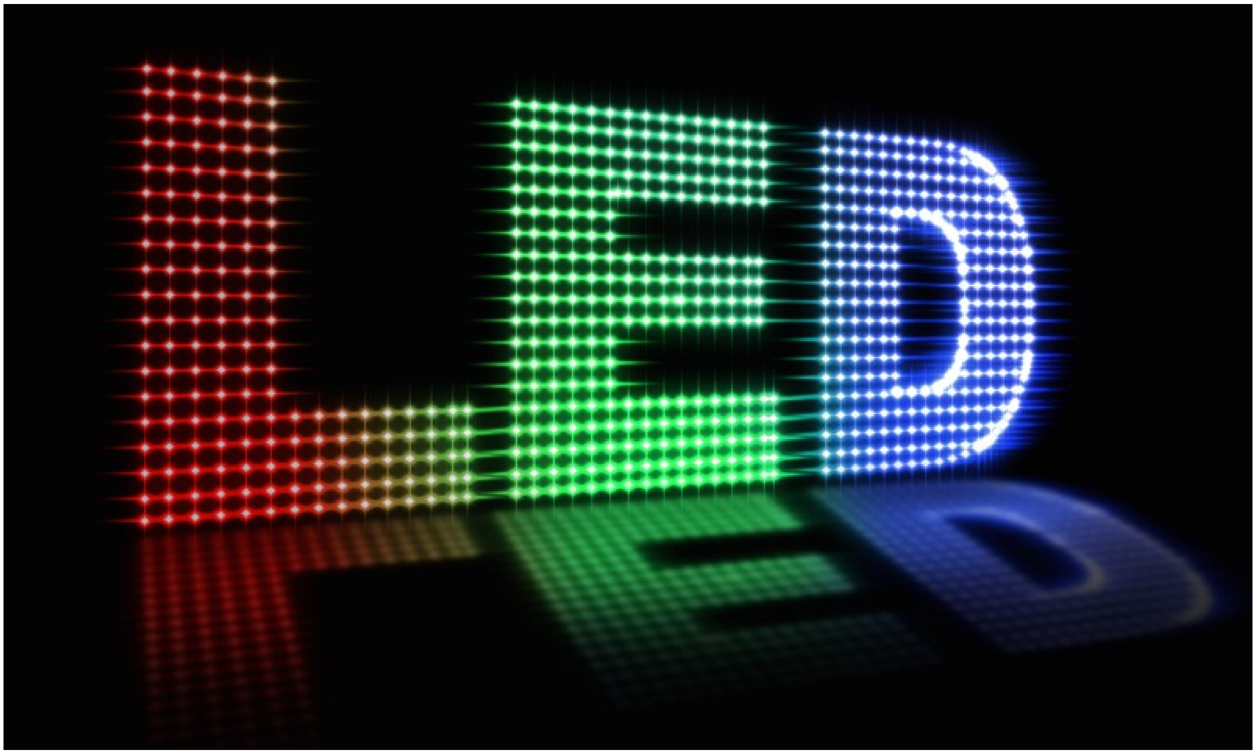
When running online, click  "Stop /Restart Backend"or press "Ctrl+C"on the Thonny to exit the program.



Test Code

```
print("Hello World!")
print("Welcome Keyestudio")
```

5.2.2 Project 2: Lighting up LED



Overview

In this kit, we have a Keyestudio Purple Module, which is very simple to control. If you want to light up the LED, you just need to make a certain voltage across it.

In the project, we will control the high and low level of the signal end S through programming, so as to control the LED on and off.

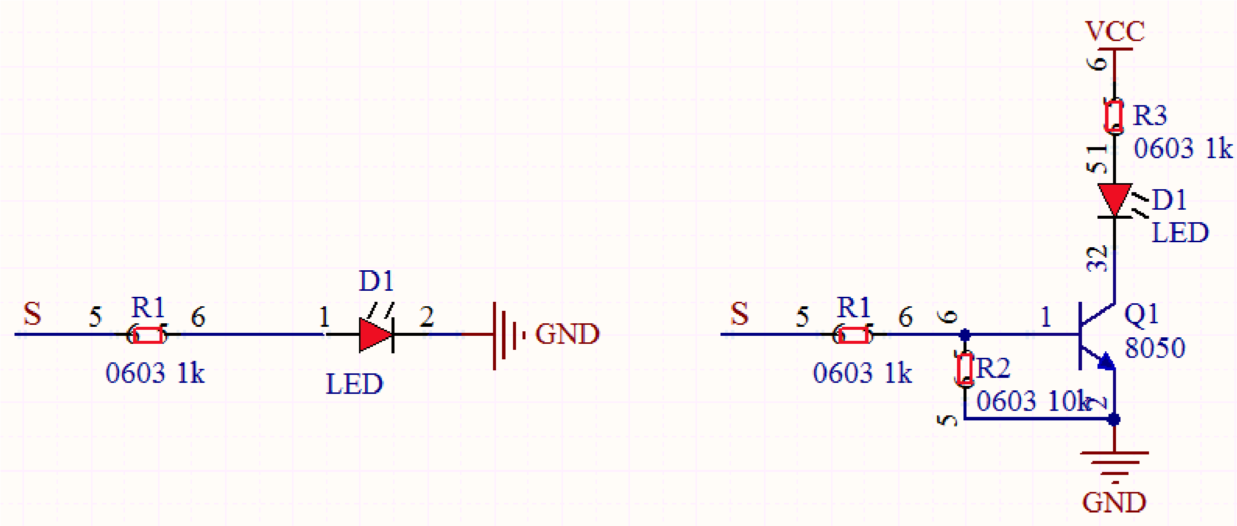
Working Principle

The two circuit diagrams are given.

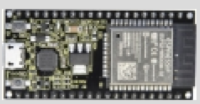
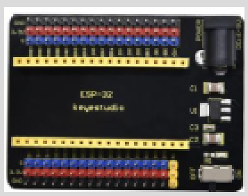
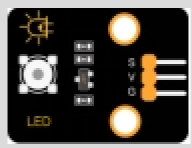
The left one is wrong wiring-up diagram. Why? Theoretically, when the S terminal outputs high levels, the LED will receive the voltage and light up.

Due to limitation of IO ports of ESP32 board, weak current can't make LED brighten.

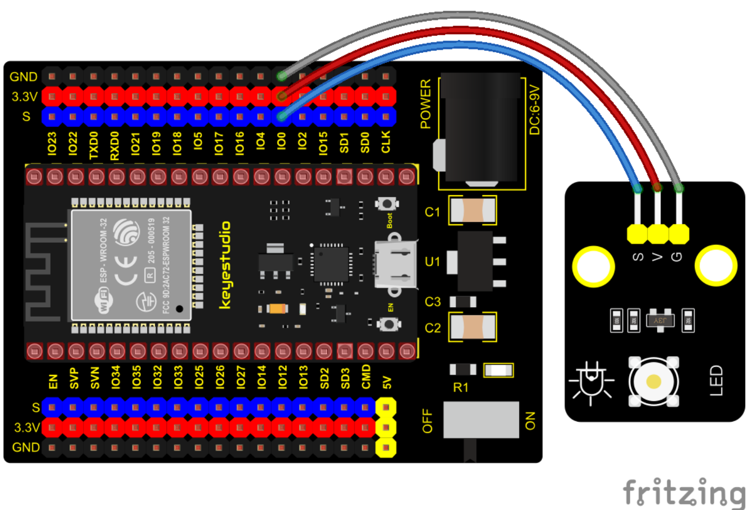
The right one is correct wiring-up diagram. GND and VCC are powered up. When the S terminal is a high level, the triode Q1 will be connected and LED will light up(note: current passes through LED and R3 to reach GND by VCC not IO ports). Conversely, when the S terminal is a low level, the triode Q1 will be disconnected and LED will go off.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1	3P Dupont Wire*1	MicroUSB Cable*1

Wiring Diagram



Test Code

```
from machine import Pin
import time
```

(continues on next page)

(continued from previous page)

```
led = Pin(0, Pin.OUT)# Build an LED object, connect the external LED light to pin 0, and␣
↪set pin 0 to output mode
while True:
    led.value(1)# turn on led
    time.sleep(1)# delay 1s
    led.value(0)# turn off led
    time.sleep(1)# delay 1s
```

Code Explanation

Machine module is indispensable, we use **import machine** or **from machine import…** to program ESP32 with microPython.

time.sleep() function is used to set delayed time, as **time.sleep(0.01),** which means, the delayed time is 10ms.

led = Pin(0, Pin.OUT)created a pin example and we name **led.**

0 is indicative of connected pin GP0**Pin.OUT represents output mode** can use **.value() to output high levels** (3.3V) **led.value(1) or low levels** (0V) **led.value(0)**.

while True is loop function

It means that sentences under this function will loop unless **True** changes into **False.** For the function **whileled.value(1)**, outputs high levels to the pin 0; then LED lights up. Then the delayed function **time.sleep(1)** will wait for 1s. When **led.value(0)** output low levels to the pin 0, the LED will go offand the function **time.sleep(1)** will wait for 1s, cyclically, and LED will flash.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, we will see that the LED in the circuit will flash alternately. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.3 Project 3: Traffic Lights Module

Overview

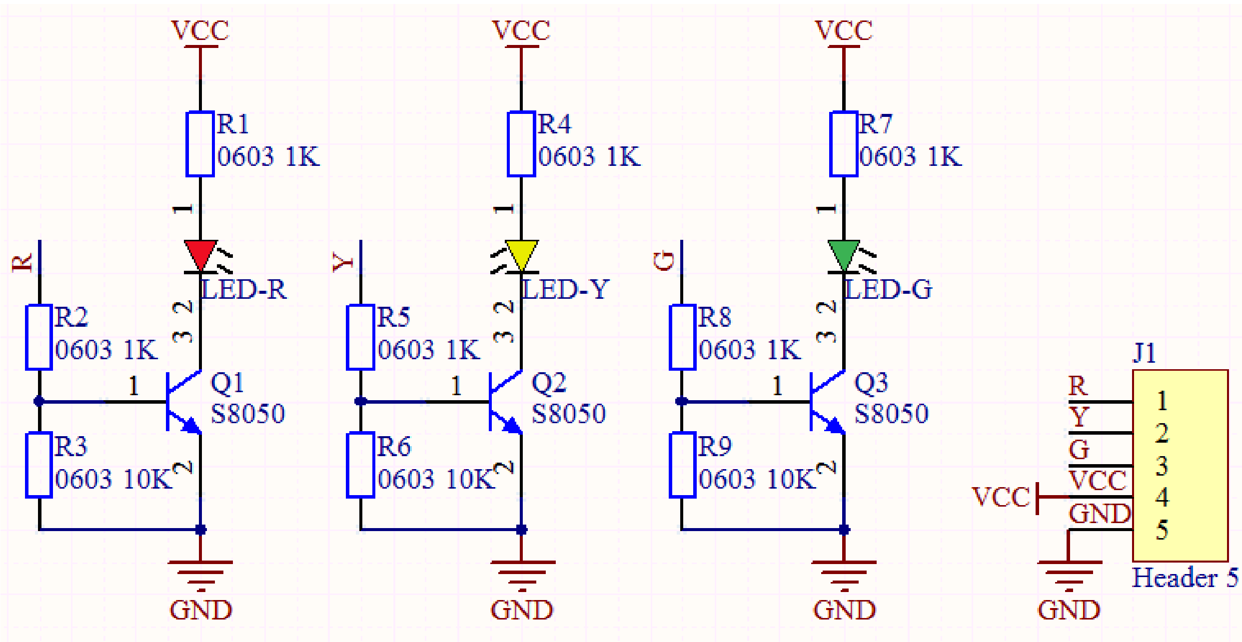
In this lesson, we will learn how to control multiple LED lights and simulate the operation of traffic lights.

Traffic lights are signal devices positioned at road intersections, pedestrian crossings, and other locations to control flows of traffic.

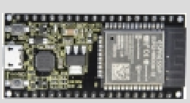
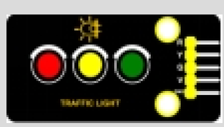
In this kit, we will use the traffic light module to simulate the traffic light.

Working Principle

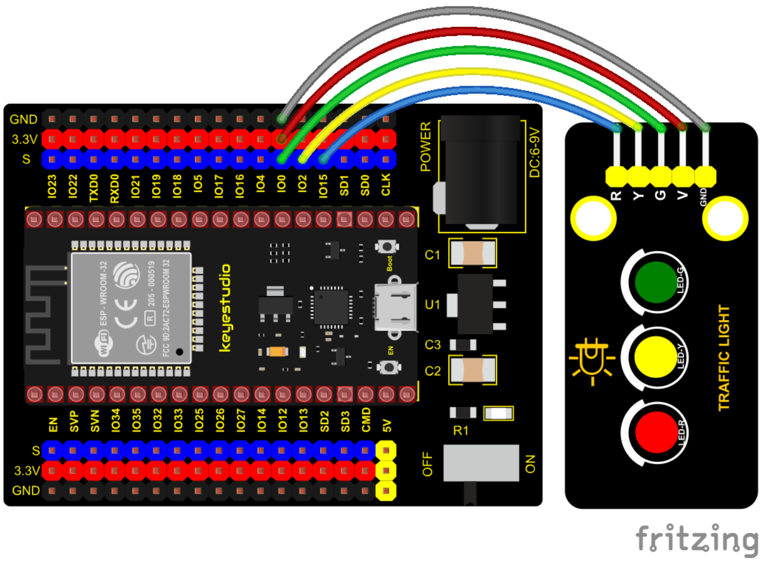
In previous lesson, we already know how to control an LED. In this part, we only need to control three separated LEDs. Input high levels to the signal R(3.3V), then the red LED will be on.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Traffic Lights Module*1	5P Dupont Wire*1	Micro USB Cable*1

Wiring Diagram



Test Code

```
import machine
import time

led_red = machine.Pin(15, machine.Pin.OUT)
led_yellow = machine.Pin(2, machine.Pin.OUT)
led_green = machine.Pin(0, machine.Pin.OUT)

while True:
    led_green.value(1) # green light turn on
    time.sleep(5) # delay 5s
    led_green.value(0) # green light turn off
    for i in range(3): # yellow light blinks 3 times
        led_yellow.value(1)
        time.sleep(0.5)
        led_yellow.value(0)
        time.sleep(0.5)
    led_red.value(1) # red light turn on
    time.sleep(5) # delay 5s
    led_red.value(0) #red light turn off
```

Code Explanation

Create pins, set pins mode and delayed functions.

We use the **for** loop.

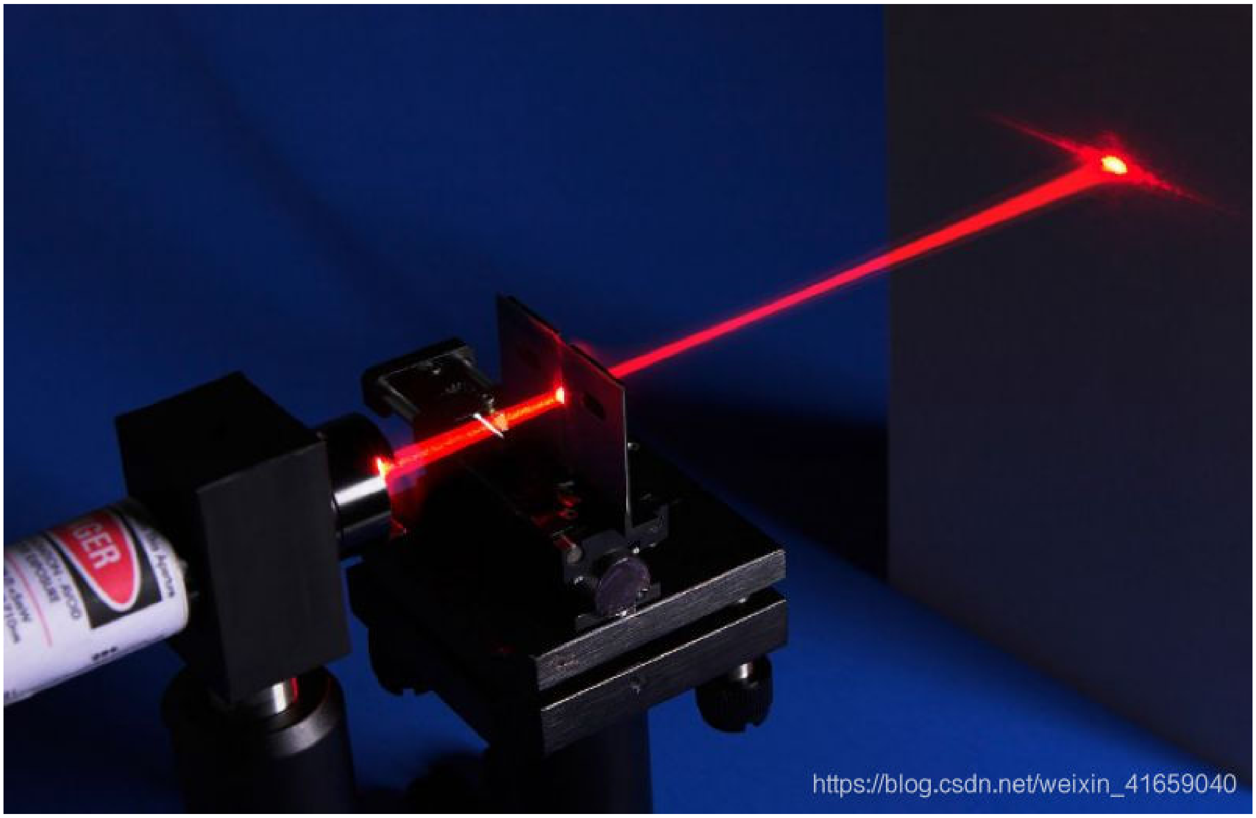
The simplest form is **for i in range()**.

In the code, we used range(3), which means the variable i starts from 0, increase 1 for each time, to 2.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, we will see that the green LED will be on for 5s then off, the yellow LED will flash for 3s then go off and the red one will be on for 5s then off. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.4 Project 4: Laser Sensor



Description

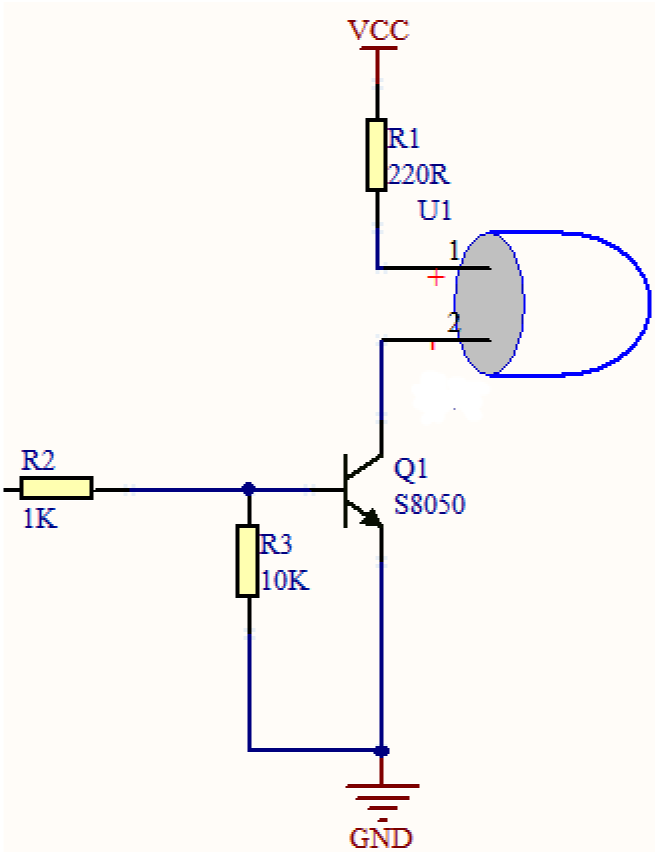
Lasers are widely used to cut, weld, surface treat, and more on specific materials. The energy of the laser is very high. The toy laser pointer may cause glare to the human eye, and it may cause retinal damage for a long time. my country also prohibits the use of laser to illuminate the aircraft.

Working Principle

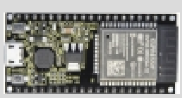
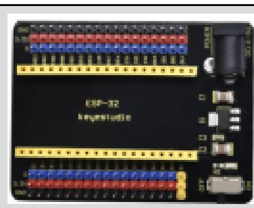
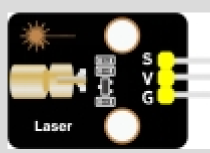
The laser head sensor module is mainly composed of a laser head with a light-emitting die, a condenser lens, and a copper adjustable sleeve.

We can see the circuit schematic diagram of this module which is very similar to the LED we have learned. They are all driven by triodes. A high-level digital signal is directly input at the signal end, then the sensor will start to work; if inputting low levels, the sensor won't work.

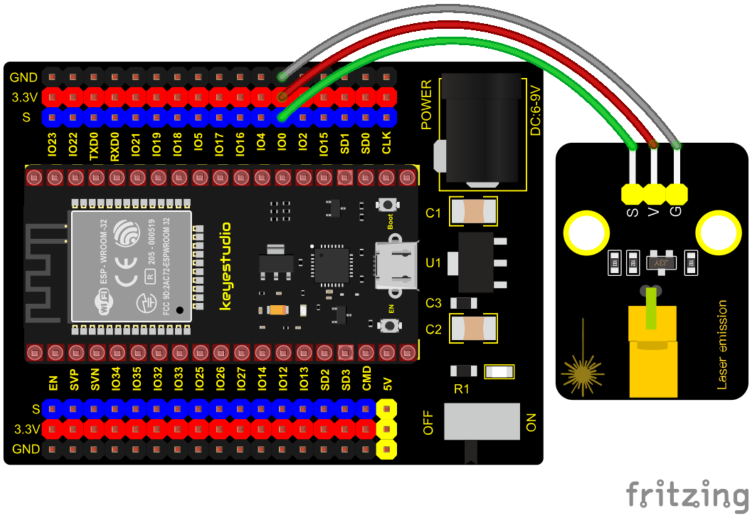
Note: don't point an laser emitter at eyes of people.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Laser Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

laser = Pin(0, Pin.OUT)# Build a laser object, connect the laser to pin 0, and set pin 0 to output mode
while True:
    laser.value(1) # Turn on the laser
    time.sleep(2) # dalay 2s
    laser.value(0) # Turn off the laser
    time.sleep(2) # delay 2s
```

Code Explanation

Please refer to project 2 above for the code setting instructions.

Test Result

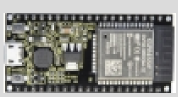
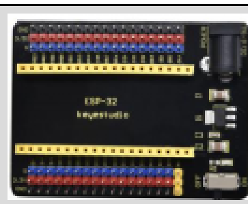
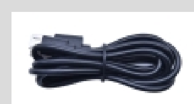
Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, we will see that the laser tube on the module emits a red laser signal for 2 seconds, and stops emitting a red laser signal for 2 seconds. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.5 Project 5: Breathing LED

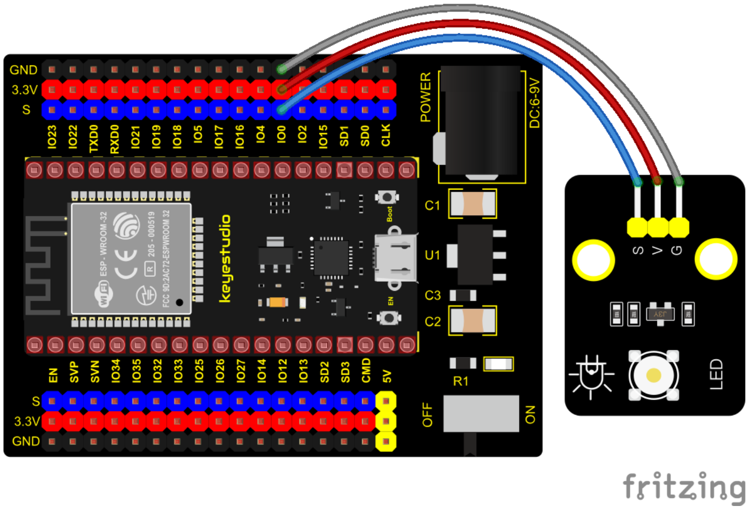
Overview

A“breathing LED”is a phenomenon where an LED’s brightness smoothly changes from dark to bright and back to dark, continuing to do so and giving the illusion of an LED“breathing. This phenomenon is similar to a lung breathing in and out. So how to control LED’s brightness? We need to take advantage of PWM. Please refer to Project 6.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1	3P Dupont Wire*1	MicroUSB Cable*1

Connection Diagram



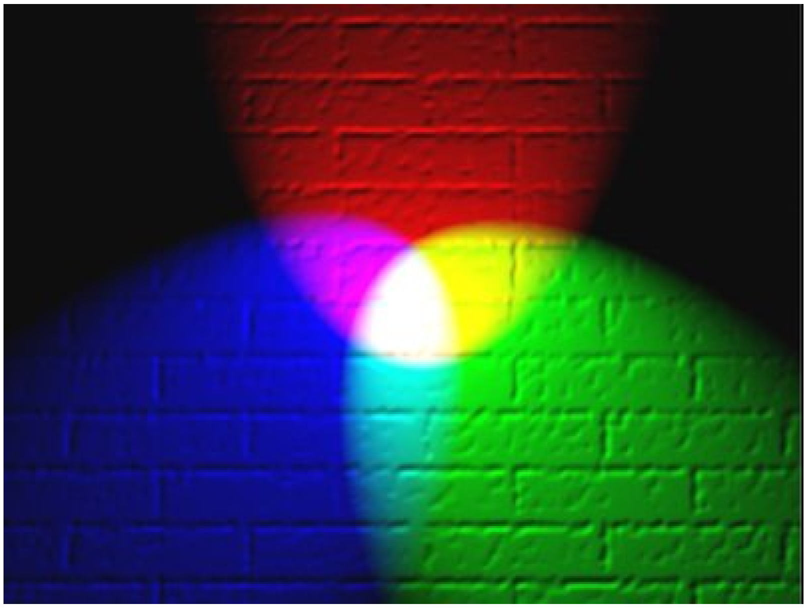
Test Code

```
import time
from machine import Pin,PWM

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

try:
    while True:
#The range of duty cycle is 0-1023, so we use the first for loop to control PWM to change the duty cycle value,making PWM output 0% -100%; Use the second for loop to make PWM output 100%-0%.
        for i in range(0,1023):
            pwm.duty(i)
            time.sleep_ms(1)

        for i in range(0,1023):
            pwm.duty(1023-i)
            time.sleep_ms(1)
except:
#Each time PWM is used, the hardware Timer will be turned ON to cooperate it. Therefore, after each use of PWM, deinit() needs to be called to turned OFF the timer. Otherwise, the PWM may fail to work next time.
    pwm.deinit()
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, we will see that the LED on the module gradually gets dimmer then brighter, cyclically, like human breathe. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.6 Project 6: RGB Module



Overview

Among these modules is a RGB module. It adopts a F10-full color RGB foggy common cathode LED. We connect the RGB module to the PWM port of MCU and the other pin to GND(for common anode RGB, the rest pin will be connected to VCC). So what is PWM?

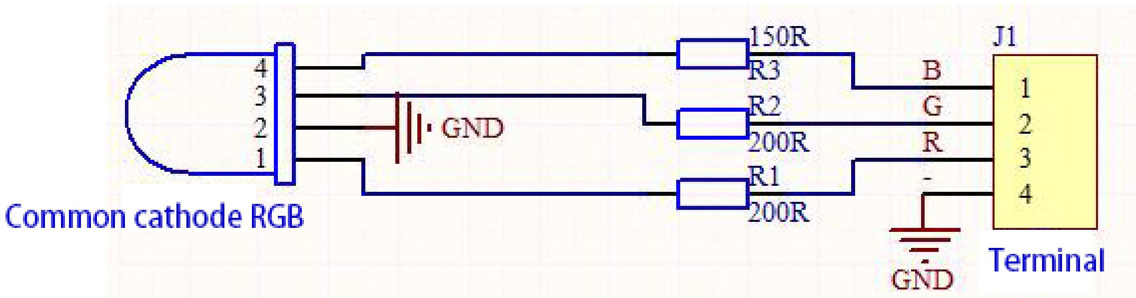
PWM is a means of controlling the analog output via digital means. Digital control is used to generate square waves with different duty cycles (a signal that constantly switches between high and low levels) to control the analog output. In general, the input voltages of ports are 0V and 5V. What if the 3V is required? Or a switch among 1V, 3V and 3.5V? We cannot change resistors constantly. For this reason, we resort to PWM.

For Arduino digital port voltage outputs, there are only LOW and HIGH levels, which correspond to the voltage outputs of 0V and 5V respectively. You can define LOW as"0"and HIGH as"1', and let the Arduino output five hundred'0'or"1"within 1 second. If output five hundred'1', that is 5V; if all of which is'0',that is 0V; if output 250 01 pattern, that is 2.5V.

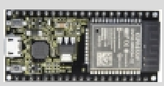
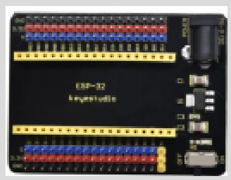
This process can be likened to showing a movie. The movie we watch are not completely continuous. Actually, it generates 25 pictures per second, which cannot be told by human eyes. Therefore, we mistake it as a continuous process. PWM works in the same way. To output different voltages, we need to control the ratio of 0 and 1. The more'0'or'1' output per unit time, the more accurate the control.

Working Principle

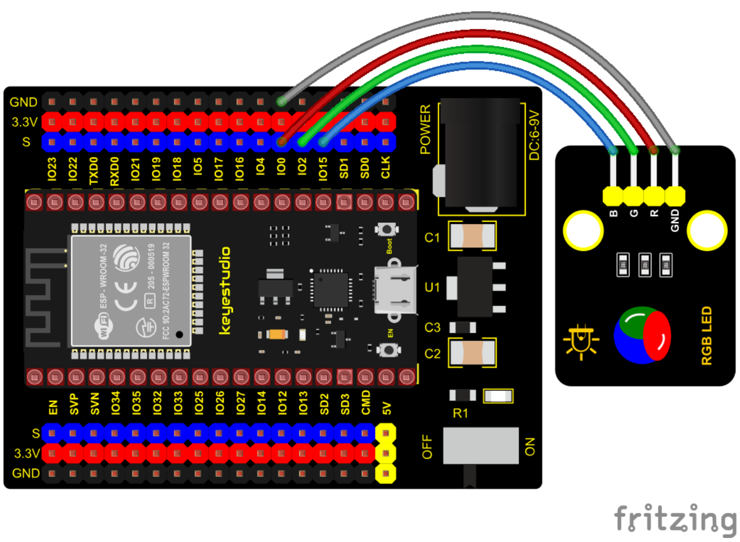
For our experiment, we will control the RGB module to display different colors through three PWM values.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Common Cathode RGB Module *1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# import Pin, PWM and Random function modules.
from machine import Pin, PWM
from random import randint
import time

#Configure ouput mode of GPIO0, GPIO2 and GPIO15 as PWM output and PWM frequency as 10000Hz.
pins = [0, 2, 15]
```

(continues on next page)

(continued from previous page)

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

#define a function to set the color of RGBLED.
def setColor(r, g, b):
    pwm0.duty(1023-r)
    pwm1.duty(1023-g)
    pwm2.duty(1023-b)

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

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, we will see that the RGB LED on the module starts to display random colors. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.7 Project 7: Button Sensor

Overview

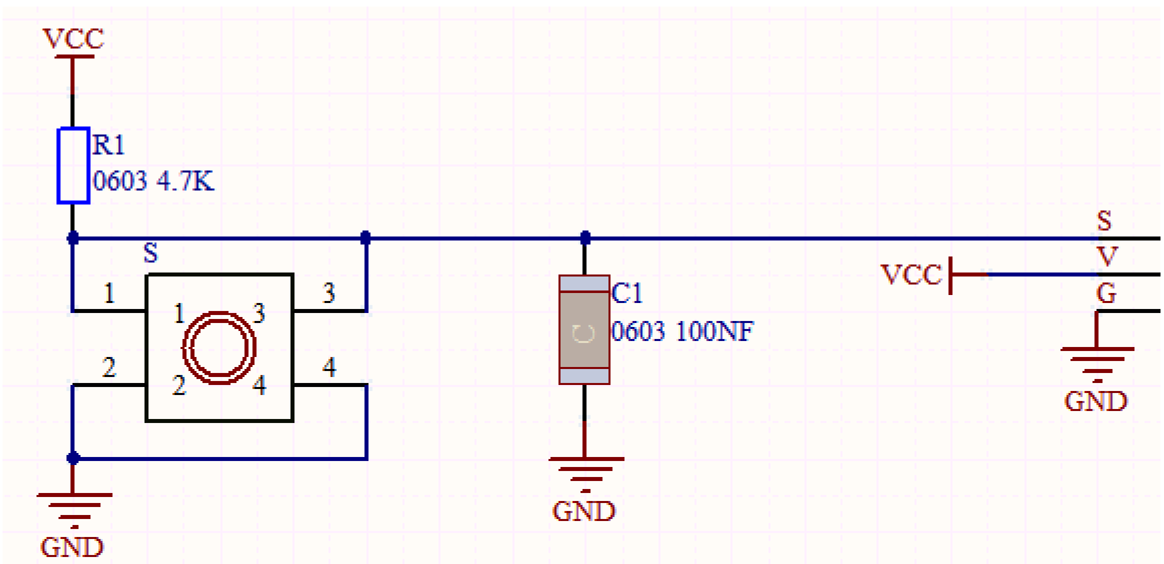
In this kit, there is a Keyestudio single-channel button module, which mainly uses a tact switch and comes with a yellow button cap.

In previous lessons, we learned how to make the pins of our single-chip microcomputer output a high level or low level. In this experiment, we will read the high level (3.3V) and low level (0V).

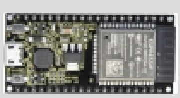
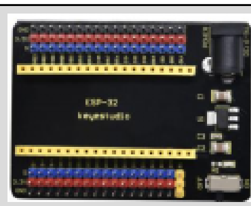
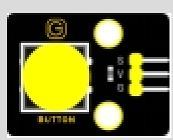
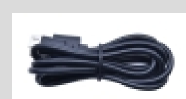
We can determine whether the button on the sensor is pressed by reading the high and low level of the S terminal on the sensor.

Working Principle

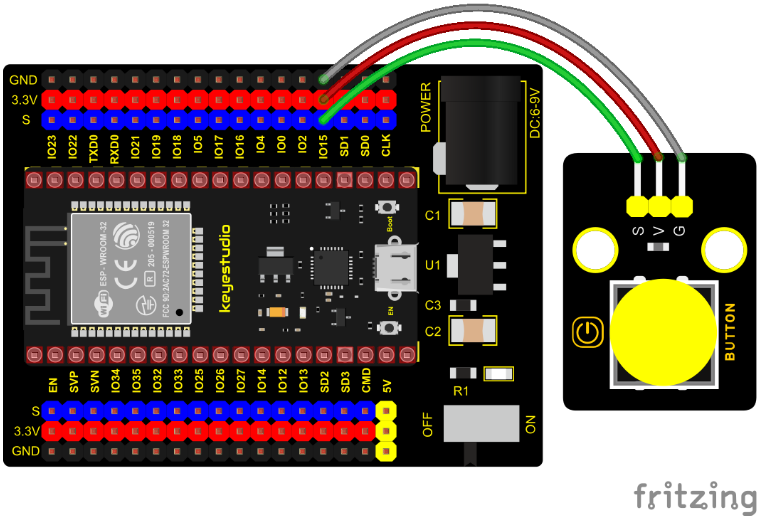
The button module has four pins. The pin 1 is connected to the pin 3 and the pin 2 is linked with the pin 4. When the button is not pressed, they are disconnected. Yet, when the button is pressed, they are connected. If the button is released, the signal end is high level.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Button Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

button = Pin(15, Pin.IN, Pin.PULL_UP)

while True:
    if button.value() == 0:
        print("You pressed the button!")   #Press to print the corresponding information.
    else:
        print("You loosen the button!")
    time.sleep(0.1) #delay 0.1s
```

Code Explanation

button = Pin(15, Pin.IN, Pin.PULL_UP), we define the pin of the button as GP15 and set to PULL-UP mode

We can use **button = Pin(15, Pin.IN) to set INPUT mode,** at this time, the pins are in high resistance state.

1). **button.value(),** read levels of buttons. Function returns High or Low

2). **if…else… sentence,** when the logic judge is TRUE, the code under the if will be activated; otherwise, the code udder the else will be activated.

3). When ESP32 detects the button pressed, the signal end is low level (GP 15 is low level). **button.value() is 0.** If the ESP32 detects the button unpressed, **button.value()** is 1 and else sentence will be activated.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the string will be displayed on the ”Shell“ window.

When the button is pressed, the ”Shell“ window will show“You pressed the button!”when the button is releasedthe ”Shell“ window will show “Loosen the button”; as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

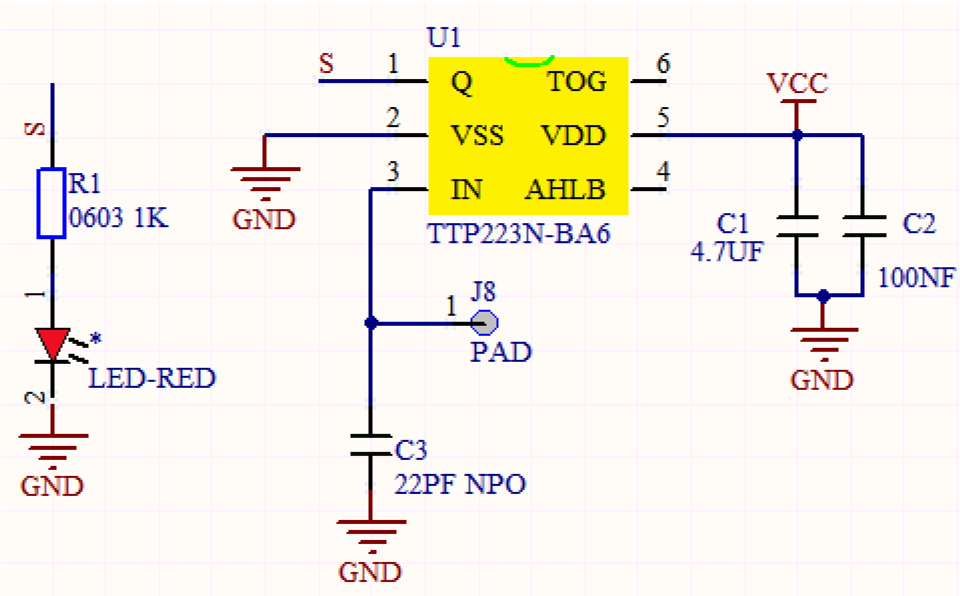
```
Shell
You loosen the button!
You loosen the button!
You loosen the button!
You loosen the button!
You loosen the button!
You loosen the button!
You loosen the button!
You loosen the button!
You pressed the button!
You pressed the button!
You pressed the button!
```

5.2.8 Project 8: Capacitive Sensor

Description

In this kit, there is a capacitive touch module which mainly uses a TTP223-BA6 chip. It is a touch detection chip, which provides a touch button, and its function is to replace the traditional button with a variable area button. When we power on, the sensor needs about 0.5 seconds to stabilize.

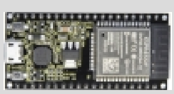
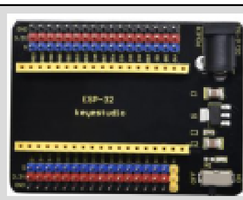
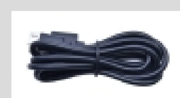
Do not touch the keys during this time period. At this time, all functions are disabled, and self-calibration is always performed. The calibration period is about 4 seconds. We display the test results in the shell.



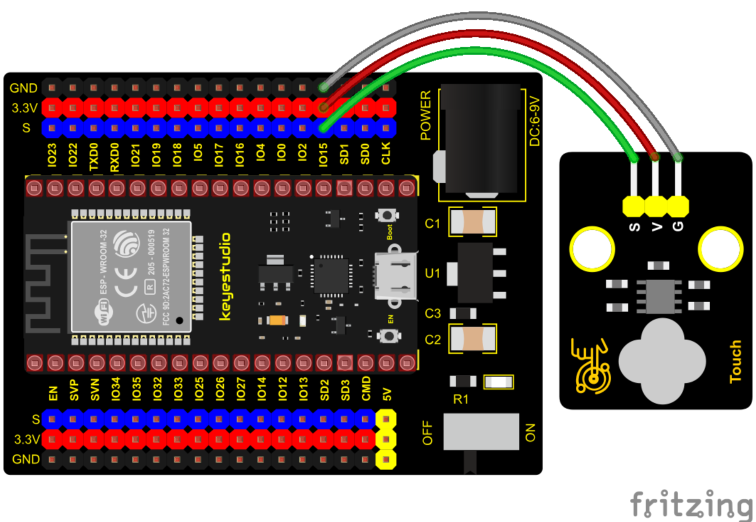
Working Principle

When our fingers touch the module, the signal S outputs high levels, the red LED on the module flashes. We can determine if the button is pressed or not by reading high and low levels on the sensor.

Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Capacitive Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

touch = Pin(15, Pin.IN, Pin.PULL_UP)

while True:
    if touch.value() == 1:
        print("You pressed the button!")   #Press to print the corresponding information.
    else:
        print("You loosen the button!")
    time.sleep(0.1) #delay0.1s
```

Code Explanation

When we touch the sensor, the Shell monitor will show“You pressed the button!”, if not,“You loosen the button!”will be shown on the monitor.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the string will be displayed in the ”Shell“ window. when the button is pressed, the red LED lights up and val is 1. Then the shell shows“You pressed the button!”; if the button is released, the red LED is off and val is 0;“You loosen the button!”will be displayed, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.9 Project 9: Obstacle Avoidance Sensor

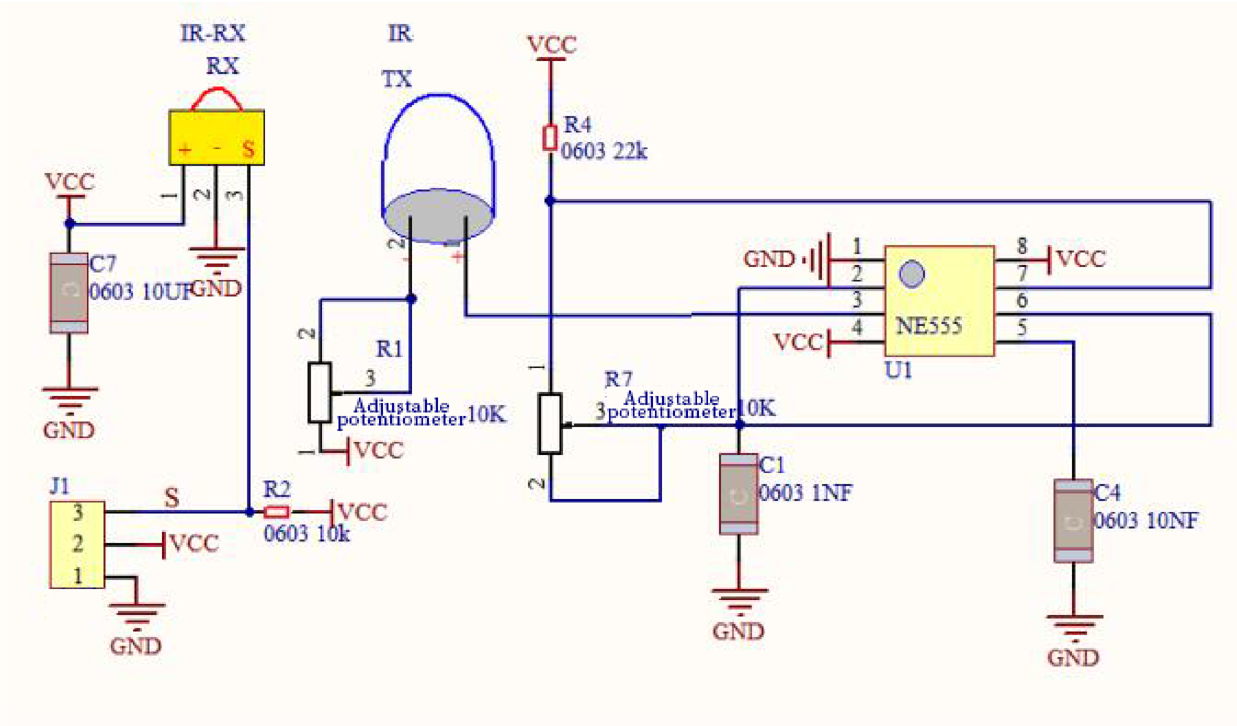
Overview

In this kit, there is a Keyestudio obstacle avoidance sensor, which mainly uses an infrared emitting and a receiving tube. In the experiment, we will determine whether there is an obstacle by reading the high and low level of the S terminal on the sensor.

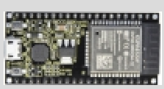
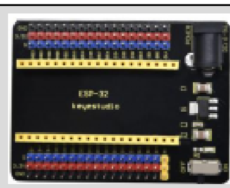
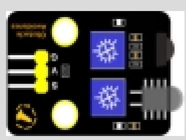
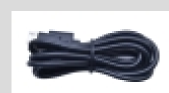
Working Principle

NE555 circuit provides IR signals with frequency to the emitter TX, then the IR signals will fade with the increase of transmission distance. If encountering the obstacle, it will be reflected back.

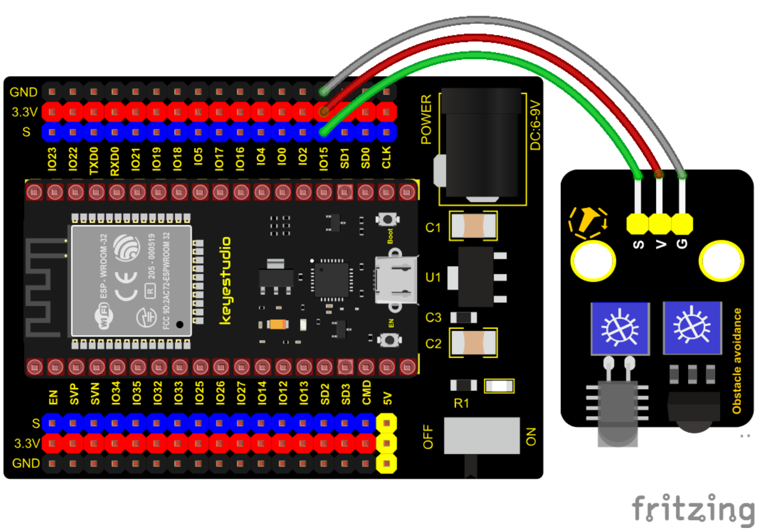
When the receiver RX meets the weak signals reflected back, the receiving pin will output high levels, which indicates the obstacle is far away. On the contrary, it the reflected signals are stronger, low levels will be output, which represents the obstacle is close. There are two potentiometers on the module, and by adjusting the two potentiometers, we can adjust its effective distance.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Obstacle Avoidance Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```python
from machine import Pin
import time

sensor = Pin(15, Pin.IN)
while True:
    if sensor.value() == 0:
        print("There are obstacles")
    else:
        print("All going well")
    time.sleep(0.1)
```

Code Explanation

Note:

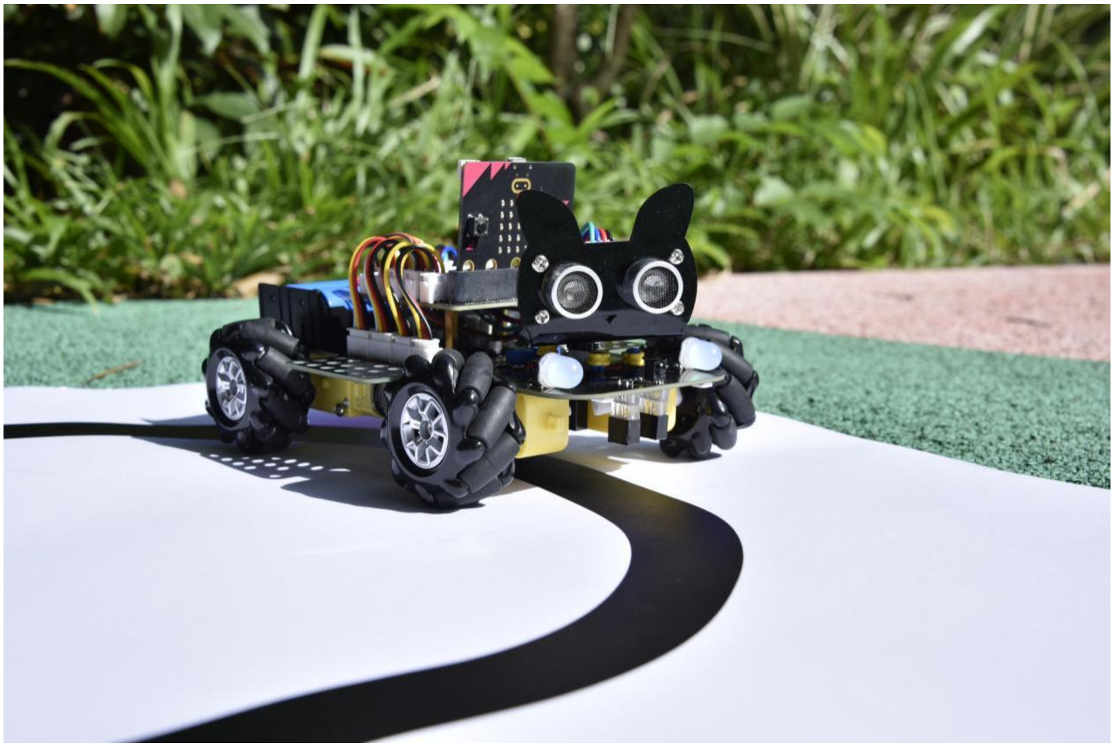
Connect the wires according to the connection diagram. After powering on, we start to adjust the two potentiometers to sense distance.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the string will be displayed in the ”Shell“ window. When the sensor detects the obstacle, sensor.value() is 0, the shell will show “There are obstacles”, if the obstacle is not detected, sensor.value () is 1,“All going well”will be shown, as shown below.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.10 Project 10: Line Tracking Sensor

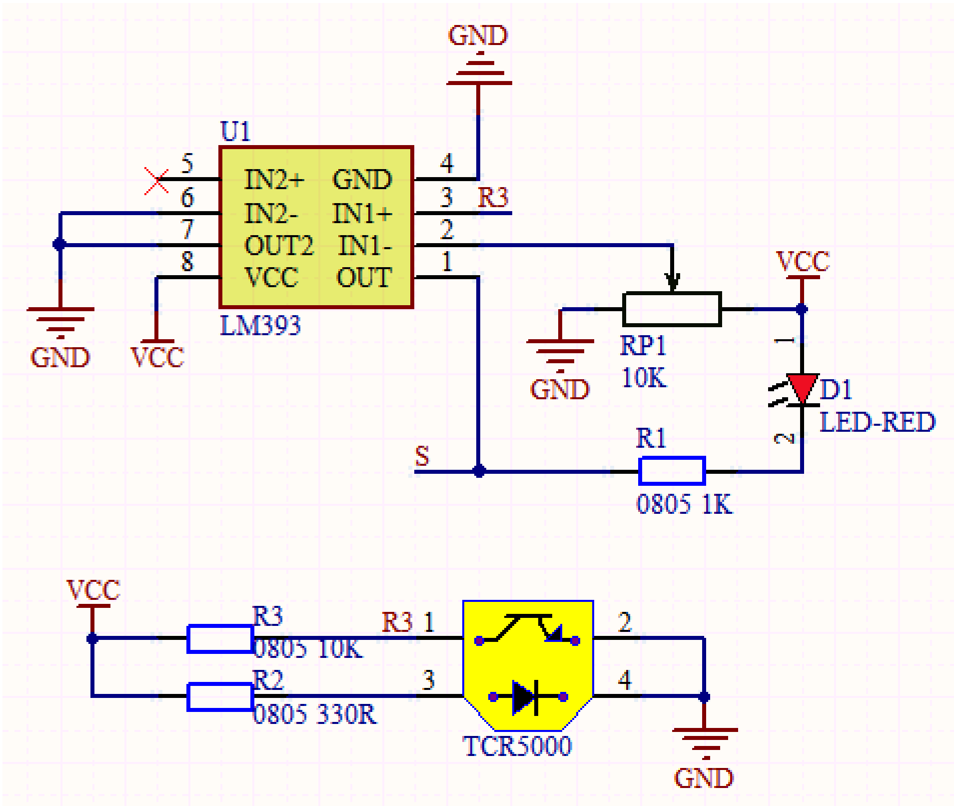


Description

In this kit, there is a DIY electronic building block single-channel line tracking sensor which mainly uses a TCRT5000 reflective black and white line recognition sensor element.

In the experiment, we judge the color (black and white) of the object detected by the sensor by reading the high and low levels of the S terminal on the module; and display the test results on the shell.

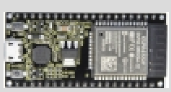
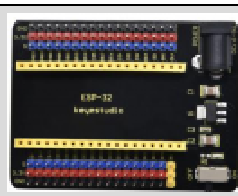
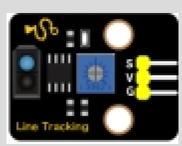
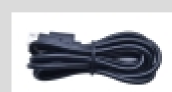
Working Principle



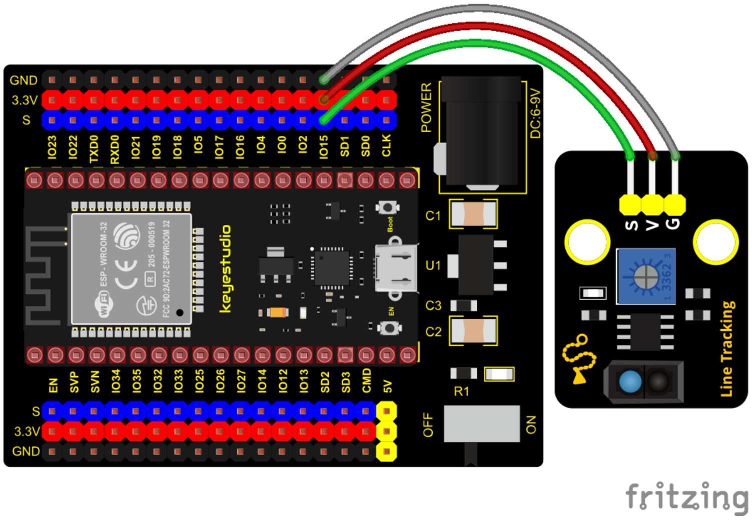
When a black or no object is detected, the signal terminal will output high levels; when white object is detected, the signal terminal is low level; its detection height is 0-3cm.

We can adjust the sensitivity by rotating the potentiometer on the sensor. When the potentiometer is rotated, the sensitivity is best when the red LED on the sensor is at the critical point between off and on.

Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Line Tracking Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

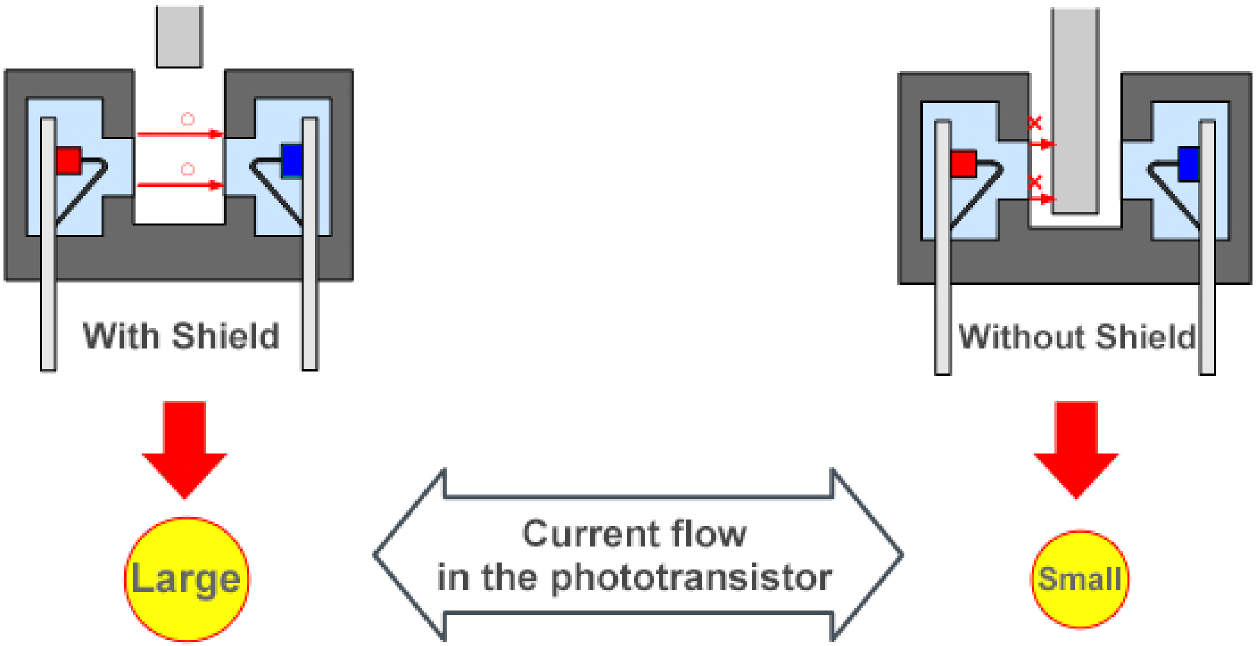
```
from machine import Pin
import time

sensor = Pin(15, Pin.IN, Pin.PULL_UP)

while True:
    if sensor.value() == 0:
        print("0    White")    #Press to print the corresponding information.
    else:
        print("1    Black")
    time.sleep(0.1) #delay 0.1s
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the string and data will be displayed in the "Shell" window.

When the sensor doesn't detect an object or detects a black object, the val is 1, and the shell will display "Black" ; when a white object (can reflect light) is detected, the val is 0, and the shell displays "White", as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.11 Project 11: Photo Interrupter

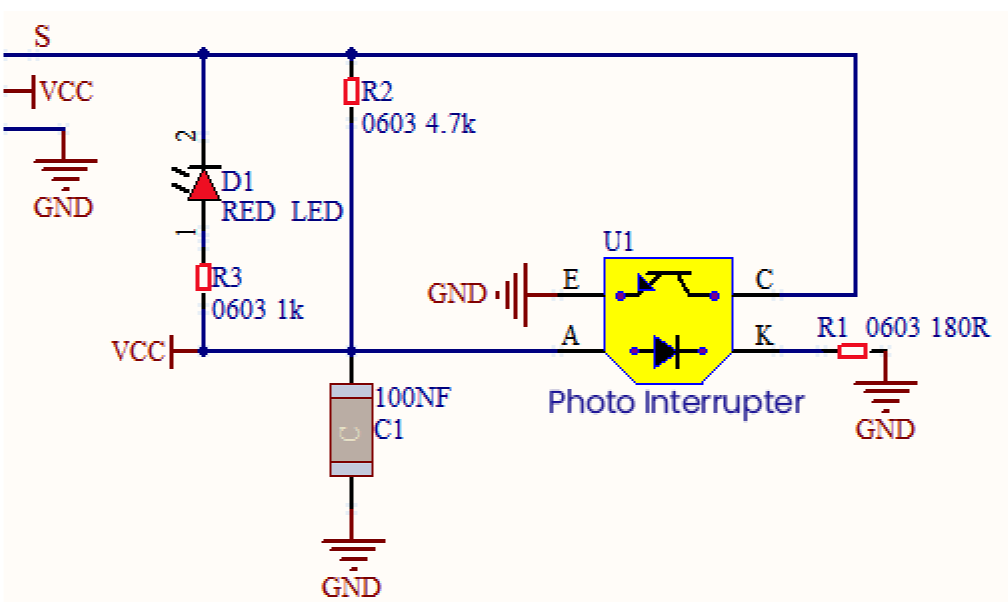


Description

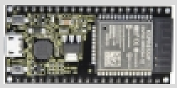
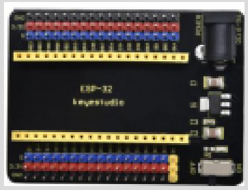
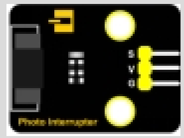
This kit contains a photo interrupter which mainly uses 1 ITR-9608 photoelectric switch. It is a photoelectric switch optical switch sensor.

Working Principle

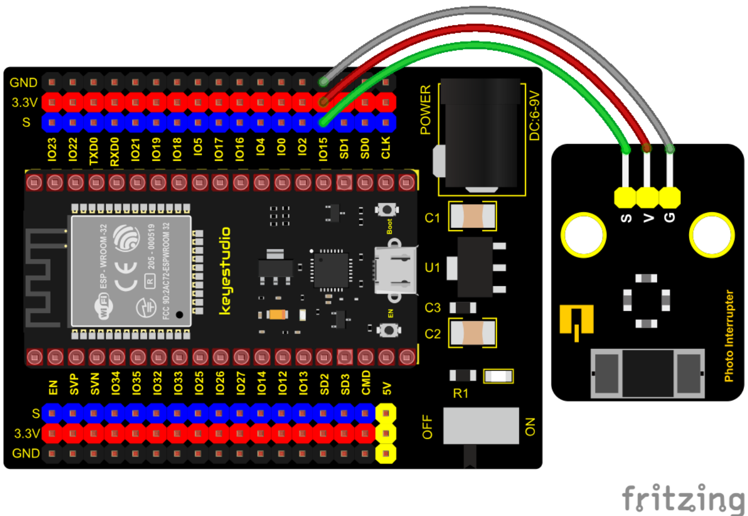
When the paper is put in the slot, C is connected with VCC and the signal end S of the sensor are high levels; then the red LED will be off. Otherwise, the red LED will be on.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Photo Interrupter*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

sensor = Pin(15, Pin.IN, Pin.PULL_UP)
lastState = 0
PushCounter = 0

while True:
    State = sensor.value()
    if  State != lastState:
        if State == 1:
            PushCounter += 1
            print(PushCounter)    #Press to print the corresponding information.
    lastState = State
```

Code Explanation

Logic setting :

Initial Setting	Set PushCounter to 0
	Set State to 0 (value of the sensor)
	Set lastState to 0

Condition	Value	Result
When an object enters the slot	lastState is 0State turns into 1; lastState turns into 1	Set PushCounter to PushCounter+1print the value of PushCounter
When the object leaves the slot	lastState is 1State becomes 0two data are not equallastState turns into 0.	PushCounterdoesn't change;Don't print the value of PushCounter
When the object goes through this slot again	lastState is 0, State becomes 1two data are not equallastState turns into 1.	Set PushCounter to PushCounter+1And print the value of PushCounter
When the object leaves this slot again	lastState is 1State turns into 0two data are not equal lastState turns into 0	PushCounter doesn't change;Don't print the PushCounter value

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, data will be displayed in the ”Shell“ window. Every time when the object passes through the slot of the sensor, the PushCounter data will increase by 1 continuously, as shown below. Press“Ctrl+C”or click “Stop/Restart backend”to exit the program.

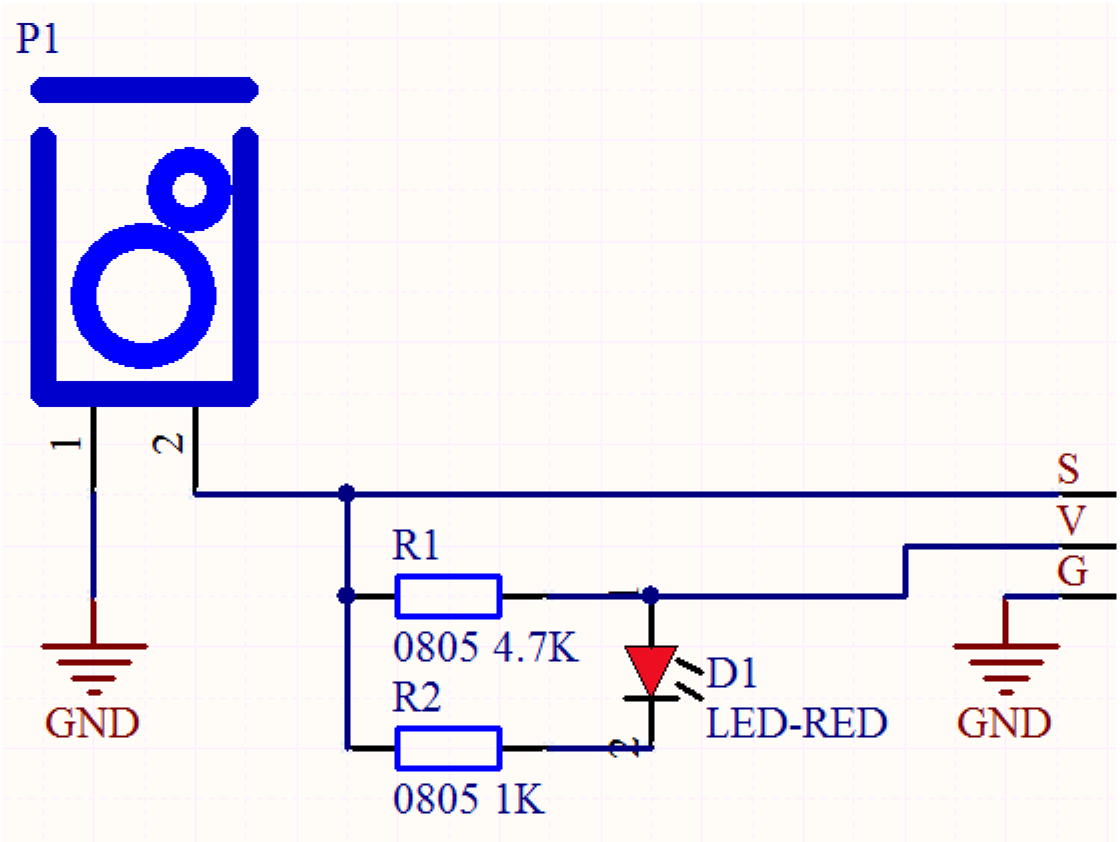
5.2.12 Project 12: Tilt Module

Overview

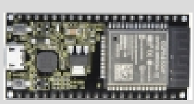
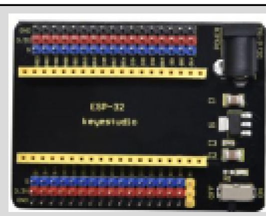
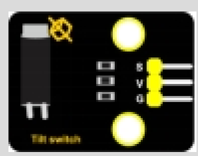
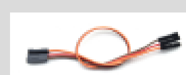
In this kit, there is a Keyestudio tilt sensor. The tilt switch can output signals of different levels according to whether the module is tilted. There is a ball inside. When the switch is higher than the horizontal level, the switch is turned on, and when it is lower than the horizontal level, the switch is turned off. This tilt module can be used for tilt detection, alarm or other detection.

Working Principle

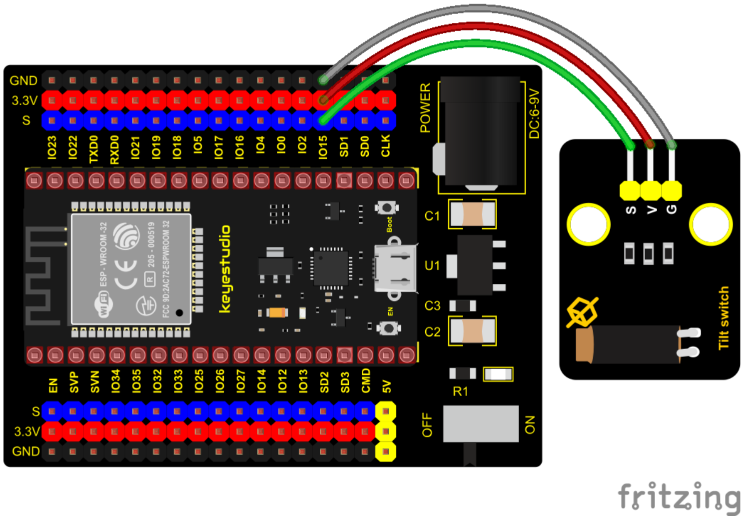
The working principle is pretty simple. When pin 1 and 2 of the ball switch P1 are connected, the signal S is low level and the red LED will light up; when they are disconnected, the pin will be pulled up by the 4.7K R1 and make S a high level, then LED will be off.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Tilt Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

TiltSensor = Pin(15, Pin.IN)

while True:
    value = TiltSensor.value()
    print(value, end = " ")
    if  value== 0:
        print("The switch is turned on")
    else:
        print("The switch is turned off")
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the string and the data will be displayed in the "Shell" window. When the tilt module is inclined to one side, the red LED on the module will be off and the Shell" window will display"1. the switch is turned off.

In contrast, if you make it incline the other side, the red LED will light up and the monitor will display"0, the switch is turned on", as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

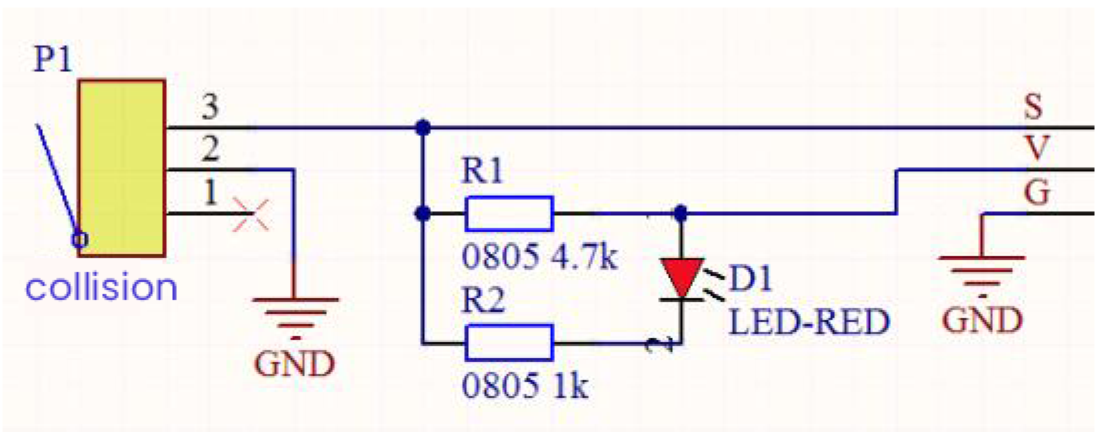
5.2.13 Project 13: Collision Sensor

Description

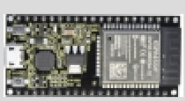
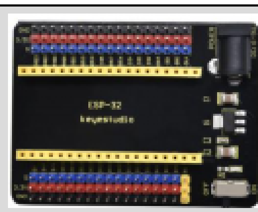
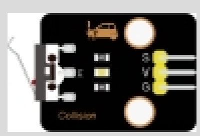
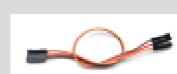
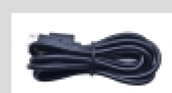
The collision sensor uses a tact switch. This sensor is often used as a limit switch in 3D printers. In the experiment, we judge whether the sensor shrapnel is pressed down by reading the high and low levels of the S terminal on the module; and, we display the test results in the shell.

Working Principle

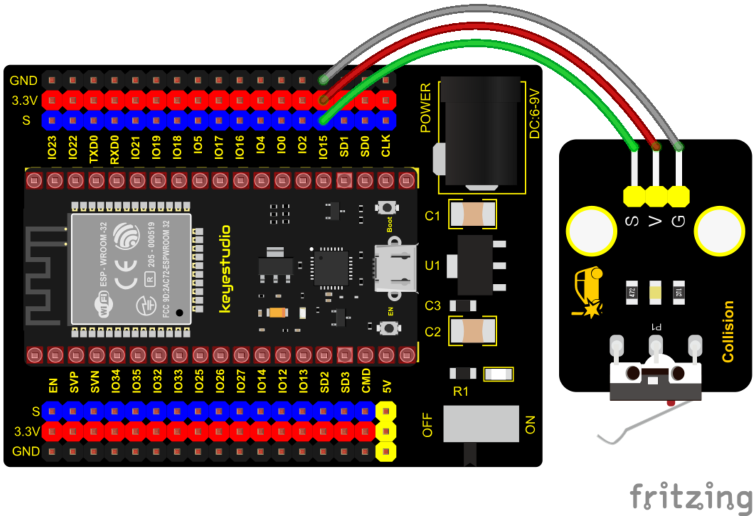
It mainly uses a tact switch. When the shrapnel of the tact switch is pressed, 2 and 3 are connected, the signal terminal S is low level, and the red LED on the module lights up; when the touch switch is not pressed, 2 and 3 are not connected, and 3 is pulled up to a high level by the 4.7K resistor R1, that is, the sensor signal terminal S is a high level, and the built-in red LED will be off at this time.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Collision Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

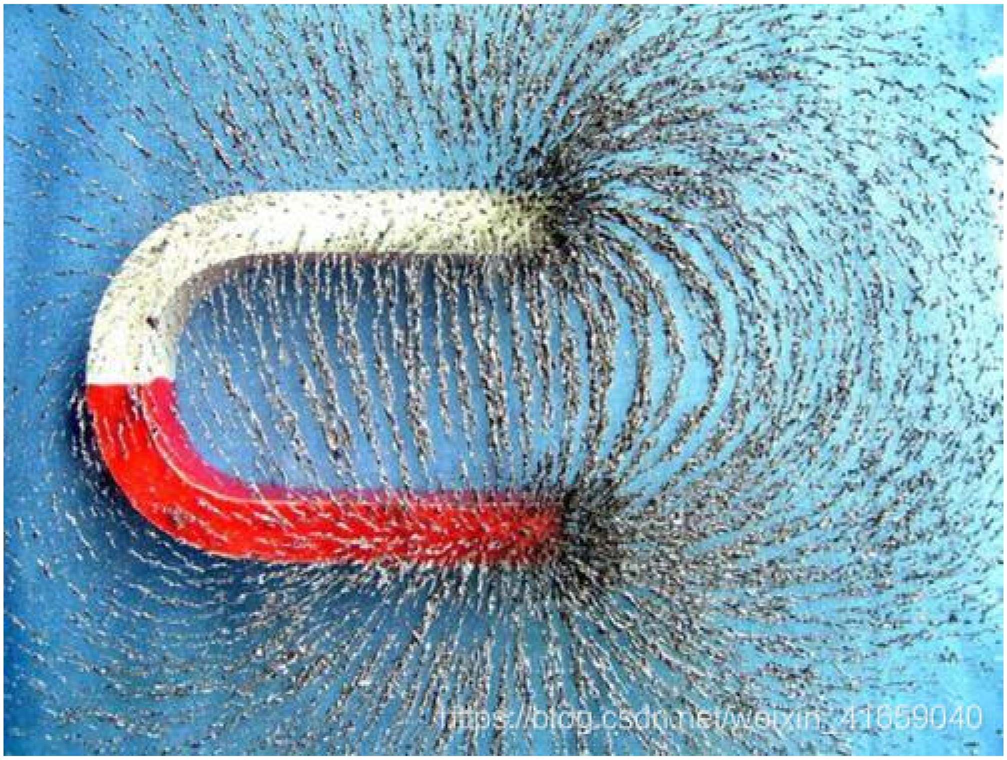
```
from machine import Pin
import time

CollisionSensor = Pin(15, Pin.IN)

while True:
    value = CollisionSensor.value()
    print(value, end = " ")
    if  value== 0:
        print("The end of this!")
    else:
        print("All going well")
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the string and the data will be displayed in the "Shell" window. When the shrapnel on the sensor is pressed down, val is 0, the red LED of the module is on, and "The end of his!" is printed.

When the shrapnel is released, the val is 1, the red LED of the module is off, and "All going well" is printed. !" character, as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.14 Project 14: Hall Sensor



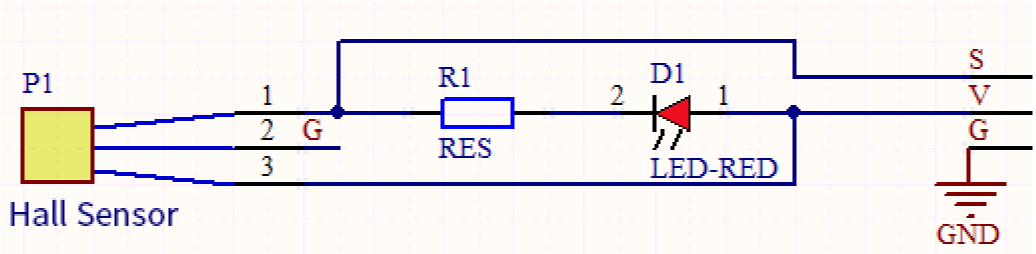
Description

In this kit, there is a Hall sensor which mainly adopts a A3144 linear Hall element. The element P1 are composed of a voltage regulator, a Hall voltage generator, a differential amplifier, a Schmitt trigger, a temperature compensation circuit and an open-collector output stage. In the experiment, we will use the Hall sensor to detect the magnetic field and display the test results on the shell.

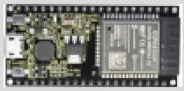
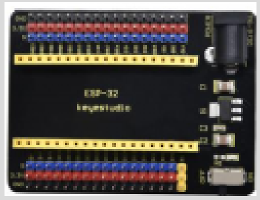
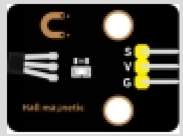
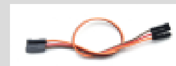
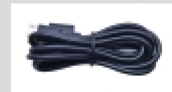
Working Principle

When the sensor detects no magnetic field or a north pole magnetic field, the signal terminal will be high level; when it senses a south pole magnetic field, the signal terminal will be low levels.

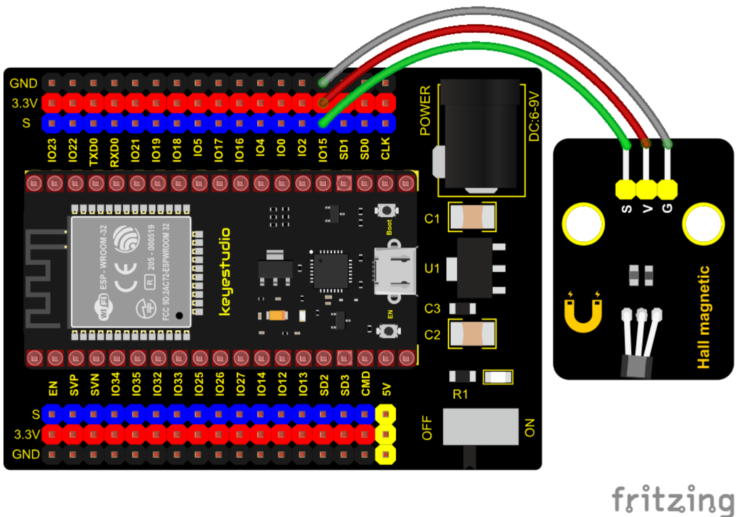
The stronger the magnetic field strength is, induction distance is longer.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Hall Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



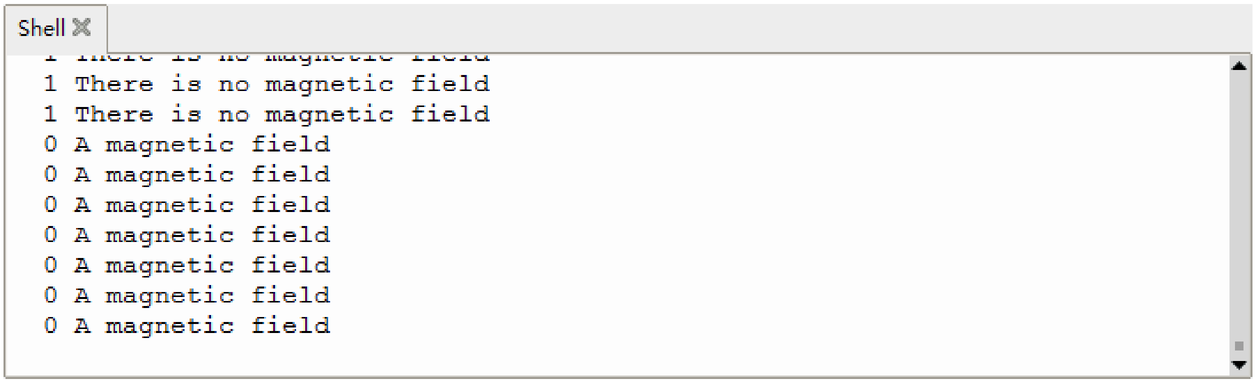
Test Code

```
from machine import Pin
import time

hall = Pin(15, Pin.IN)
while True:
    value = hall.value()
    print(value, end = " ")
    if value == 0:
        print("A magnetic field")
    else:
        print("There is no magnetic field")
    time.sleep(0.1)
```

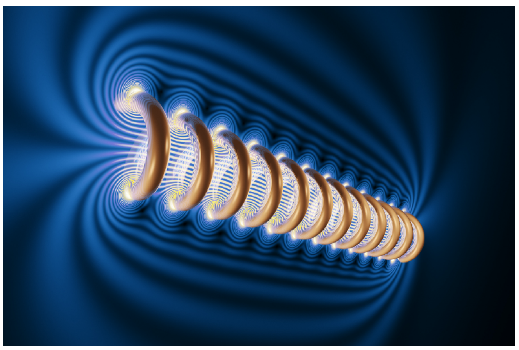
Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the string and the data will be displayed in the "Shell" window. When the sensor detects no magnetic fields or the north pole magnetic field, Shell will show"1 There is no magnetic field"and the LED on the sensor will be off.

When it detects the south pole magnetic field, the Shell will show"0 A magnetic field"and the LED on the sensor will be on, as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.



5.2.15 Project 15: Reed Switch Module



Overview

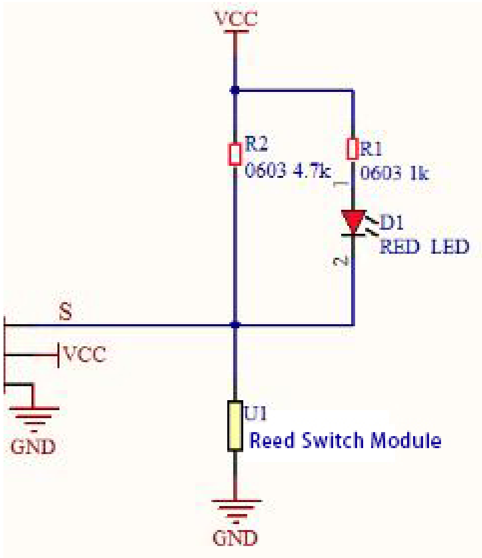
In this kit, there is a Keyestudio reed switch module, which mainly uses a MKA10110 green reed component.

The reed switch is the abbreviation of the dry reed switch. It is a passive electronic switch element with contacts.

It has the advantages of simple structure, small size and easy control. Its shell is a sealed glass tube with two iron elastic reed electric plates.

In the experiment, we will determine whether there is a magnetic field near the module by reading the high and low level of the S terminal on the module; and, we display the test result in the shell.

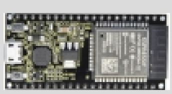
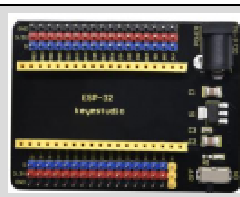
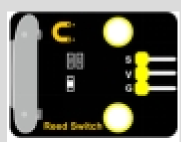
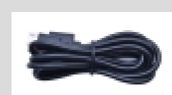
Working Principle



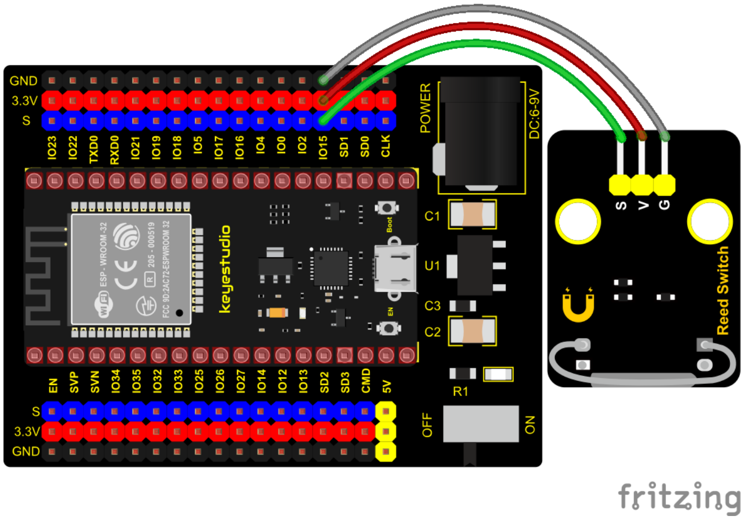
In normal conditions, the glass tube in the two reeds made of special materials are separated. When a magnetic substance close to the glass tube, in the role of the magnetic field lines, the pipe within the two reeds are magnetized to attract each other in contact, the reed will suck together, so that the junction point of the connected circuit communication.

After the disappearance of the outer magnetic reed because of their flexibility and separate, the line is disconnected. The sensor uses this characteristic to build a circuit to convert magnetic field signal into high and low level signal.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Reed Switch Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

ReedSensor = Pin(15, Pin.IN)
while True:
    value = ReedSensor.value()
    print(value, end = " ")
    if value == 0:
        print("A magnetic field")
    else:
        print("There is no magnetic field")
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the string and the data will be displayed in the ”Shell“ window.

When the sensor detects a magnetic field, val is 0 and the red LED of the module lights up, “0 A magnetic field” will be displayed. When no magnetic field is detected, val is 1, and the LED on the module goes out, “1 There is no magnetic field” will be shown, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.16 Project 16: PIR Motion Sensor

Overview

In this kit, there is a Keyestudio PIR motion sensor, which mainly uses an RE200B-P sensor elements. It is a human body pyroelectric motion sensor based on pyroelectric effect, which can detect infrared rays emitted by humans or animals, and the Fresnel lens can make the sensor's detection range farther and wider.

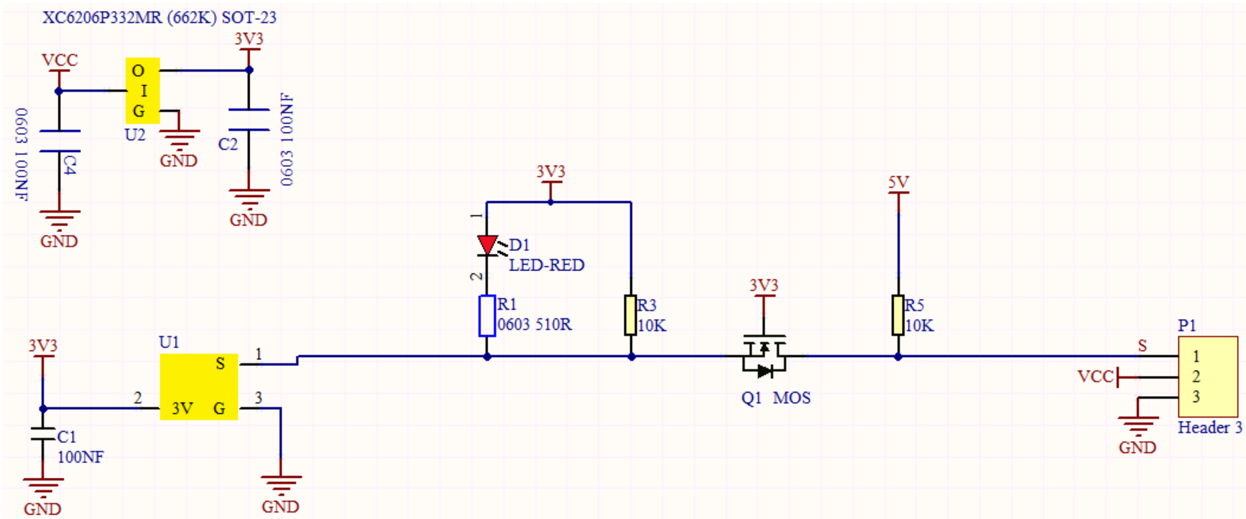
In the experiment, we determine if there is someone moving nearby by reading the high and low levels of the S terminal on the module. The detected results will be displayed on the Shell.

Working Principle

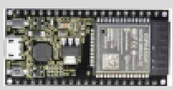
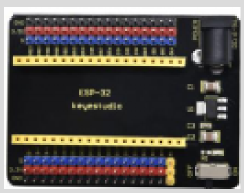
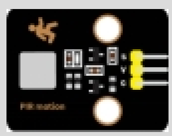
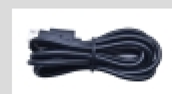
The upper left part is voltage conversion(VCC to 3.3V). The working voltage of sensors we use is 3.3V, therefore we can't use 5V directly. The voltage conversion circuit is needed.

When no person is detected or no infrared signal is received, and pin 1 of the sensor outputs low level. At this time, the LED on the module will light up and the MOS tube Q1 will be connected and the signal terminal S will detect Low levels.

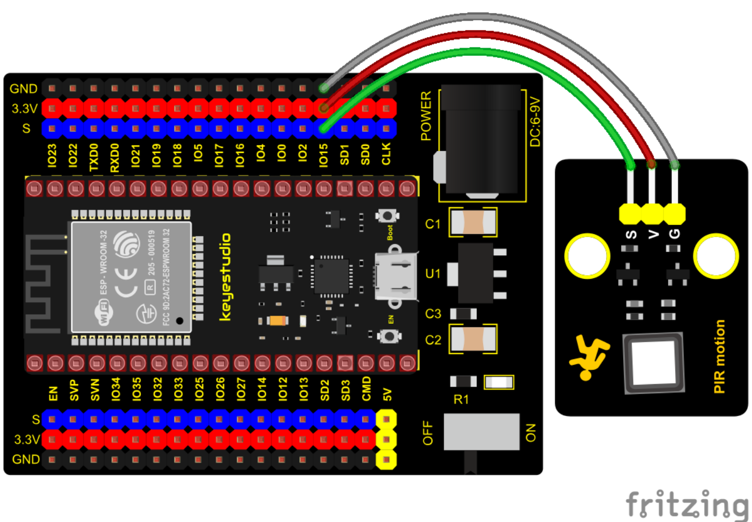
When one is detected or an infrared signal is received, and pin 1 of the sensor outputs a high level. Then LED on the module will go off, the MOS tube Q1 is disconnected and the signal terminal S will detect high levels.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY PIR Motion Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

PIR = Pin(15, Pin.IN)
```

(continues on next page)

(continued from previous page)

```
while True:
    value = PIR.value()
    print(value, end = " ")
    if value == 1:
        print("Some body is in this area!")
    else:
        print("No one!")
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the string and the data will be displayed in the "Shell" window. When the sensor detects someone nearby, value is 1, the LED will go off and the "Shell" window will show "1 Somebody is in this area!".

On the contrary, the value is 0, the LED will go up and "0 No one!" will be shown, as shown below. Press "Ctrl+C" or click "Stop/Restart backend" to exit the program.

5.2.17 Project 17: Active Buzzer

Overview

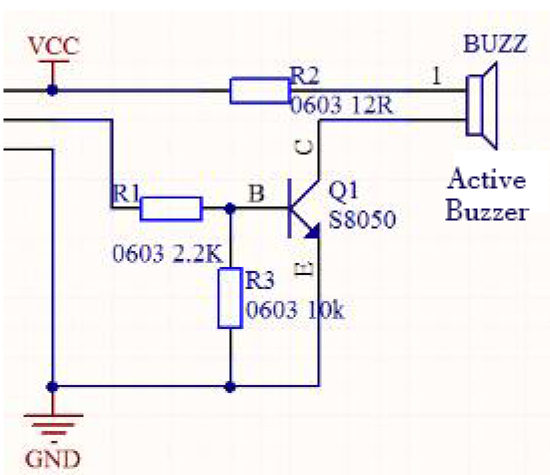
In this kit, it contains an active buzzer module and a power amplifier module (the principle is equivalent to a passive buzzer). In this experiment, we control the active buzzer to emit sounds. Since it has its own oscillating circuit, the buzzer will automatically sound if given large voltage.

Working Principle

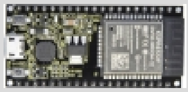
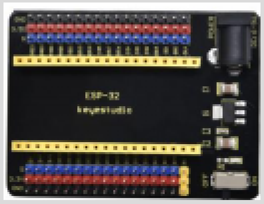
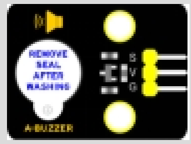
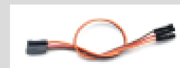
From the schematic diagram, the pin of buzzer is connected to a resistor R2 and another port is linked with a NPN triode Q1. So, if this triode Q1 is powered, the buzzer will sound.

If the base electrode of the triode connected to the R1 resistor is a high level, the triode Q1 will be connected.If the base electrode is pulled down by the resistor R3, the triode is disconnected.

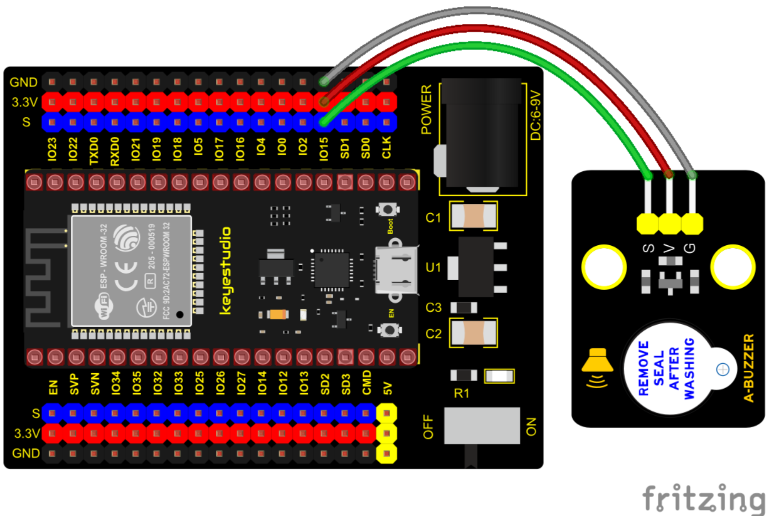
When we output a high level from the IO port to the triode, the buzzer will emit sounds; if outputting low levels, the buzzer won't emit sounds.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

buzzer = Pin(15, Pin.OUT)
while True:
    buzzer.value(1)
    time.sleep(1)
    buzzer.value(0)
    time.sleep(1)
```

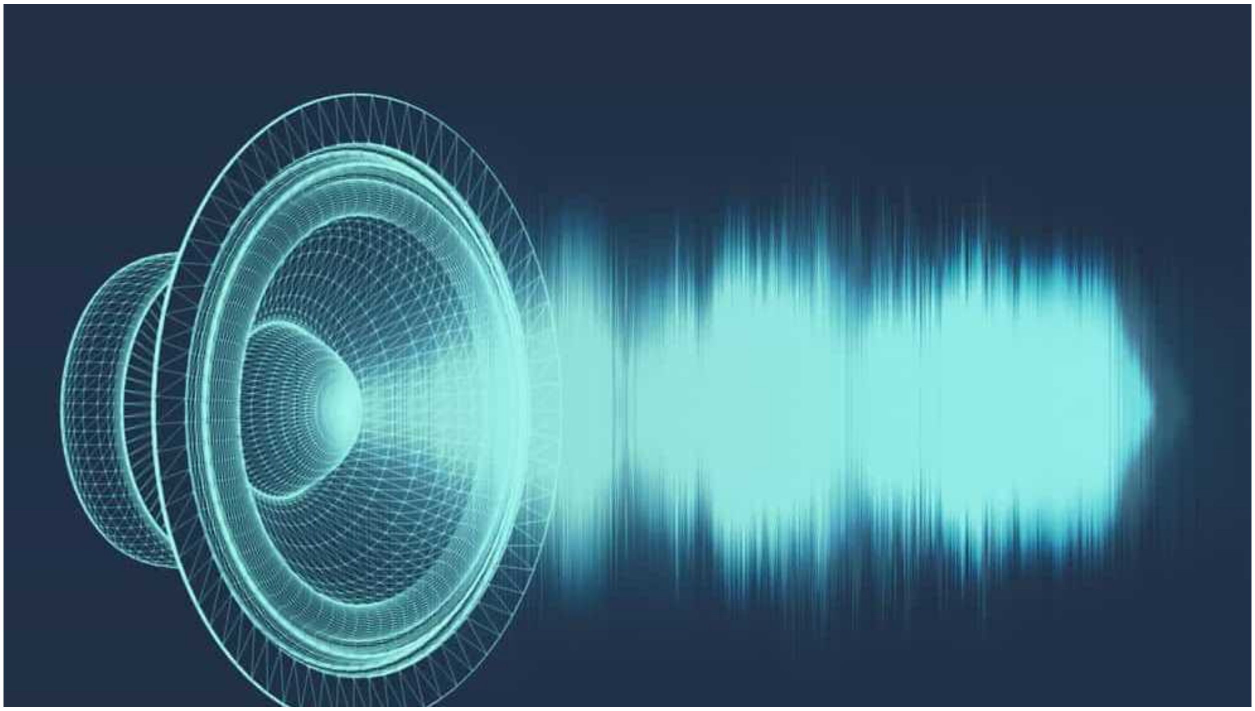
Code Explanation

In the experiment, we set the pin to GPIO15. When setting to high, the active buzzer will beep. When setting to low, the active buzzer will stop emitting sounds.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The active buzzer will emit sound for 1 second, and stop for 1 second. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.18 Project 18: 8002b Audio Power Amplifier



Overview

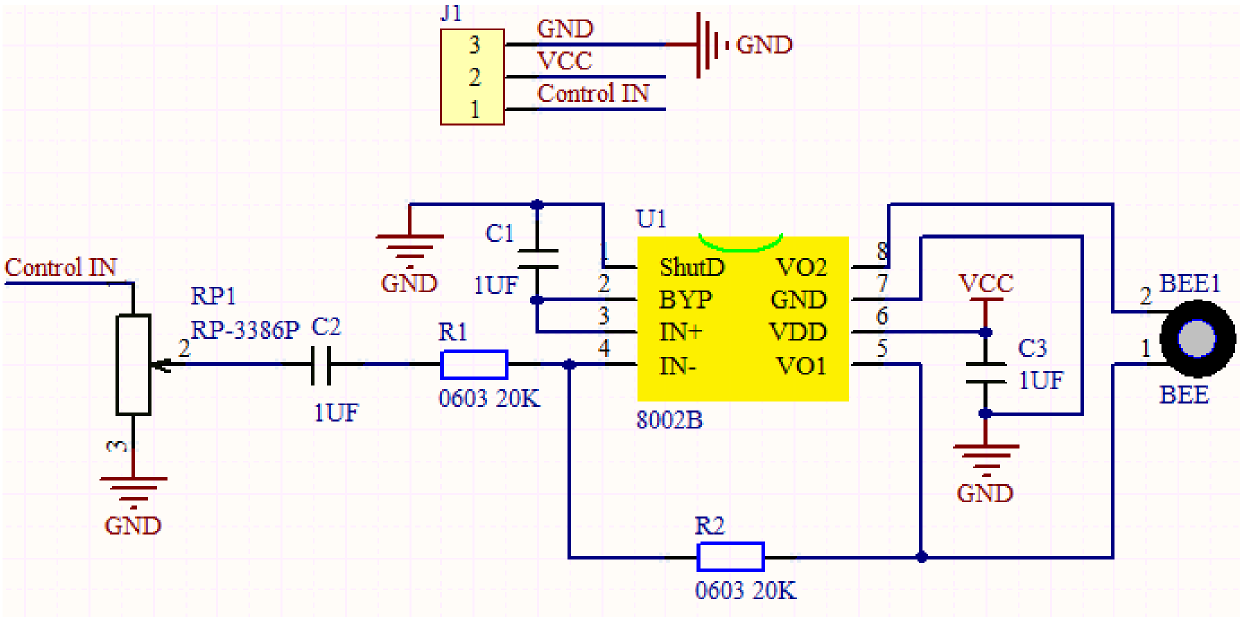
In this kit, there is a Keyestudio 8002b audio power amplifier. The main components of this module are an adjustable potentiometer, a speaker, and an audio amplifier chip;

The main function of this module is: it can amplify the output audio signal, with a magnification of 8.5 times, and play sound or music through the built-in low-power speaker, as an external amplifying device for some music playing equipment.

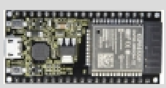
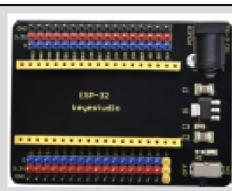
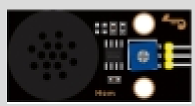
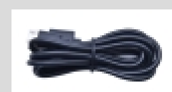
In the experiment, we used the 8002b power amplifier speaker module to emit sounds of various frequencies.

Working Principle

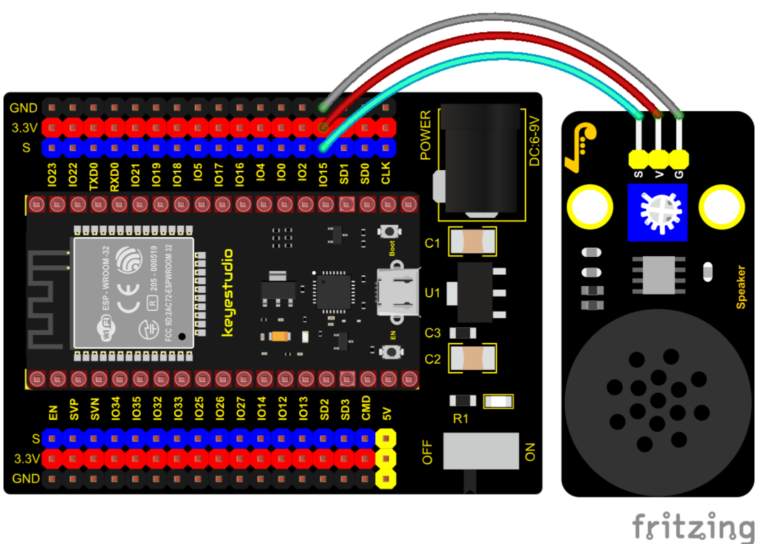
In fact, it is similar to a passive buzzer. The active buzzer has its own oscillation source. Yet, the passive buzzer does not have internal oscillation. When controlling the circuit, we need to input square waves of different frequencies to the positive pole of the component and ground the negative pole to control the buzzer to chime sounds of different frequencies.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 8002b Audio Power Amplifier*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin, PWM
from time import sleep
buzzer = PWM(Pin(15))

buzzer.duty(1000)

buzzer.freq(523)#DO
sleep(0.5)
buzzer.freq(586)#RE
sleep(0.5)
buzzer.freq(658)#MI
sleep(0.5)
buzzer.freq(697)#FA
sleep(0.5)
buzzer.freq(783)#SO
sleep(0.5)
buzzer.freq(879)#LA
sleep(0.5)
buzzer.freq(987)#SI
sleep(0.5)
buzzer.duty(0)
```

Code Explanation

In this experiment, we use the PWM class of the machine module, buzzer = PWM(Pin(15)) to create an instance of the PWM class, and the buzzer pin is connected to GPIO15.

The buzzer.duty(1000): set the duty cycle, and the duty cycle is 1000/4095. The larger the value, the louder the buzzer. When set to 0, the buzzer does not emit sound. **buzzer.freq()** is the frequency setting method.

In the experiment, we use the PWM on the machine module. **buzzer = PWM(Pin(15))**

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The power amplifier module will emit the sound of the corresponding frequency corresponding to the beat :DO for 0.5s, Re for 0.5s, Mi for 0.5s, Fa for 0.5s, So for 0.5s, La 0.5s and Si for 0.5s.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.19 Project 19: 130 Motor

Description

The 130 motor driver module is compatible with servo motors, which has high efficiency and good quality fans.

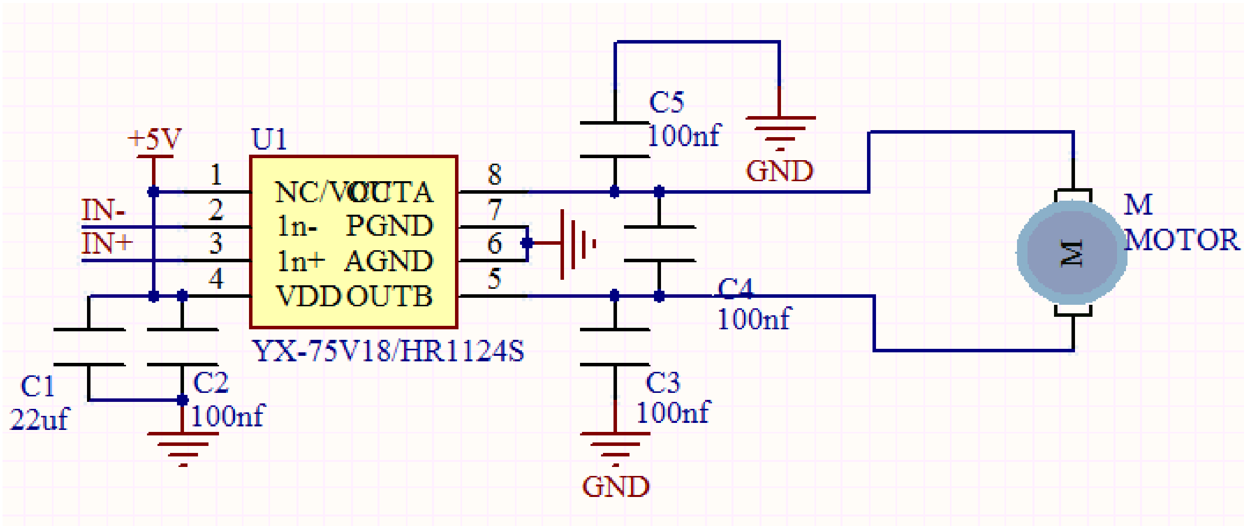
It adopts a HR1124S motor control chip. HR1124S is a single-channel H-bridge driver chip for DC motor solutions. In addition, this chip has low standby current and low quiescent current.

The module is compatible with various single-chip control boards. In the experiment, we can control the rotation direction of the motor by outputting the voltage directions of the two signal terminals IN+ and IN- to make the motor rotate.

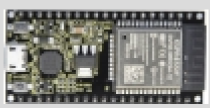
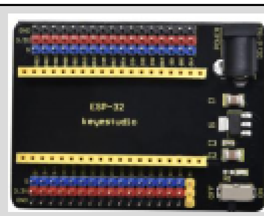
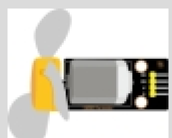
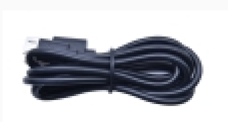
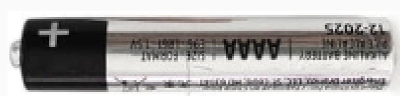
Working Principle

The chip is used to help drive the motor. We can't drive it with a triode or an IO port due to its a large current of need. It is very simple to make the motor rotate. Just apply voltage to both ends of the motor. The direction of the motor is different in different voltage directions. Within the rated voltage, the higher the voltage, the faster the motor rotates; on the contrary, the lower the voltage, the slower the motor rotates, or even unable to rotate.

So we can use the PWM port to control the speed of the motor. We haven't learned PWM here, so we use the high and low levels to control the motor first.



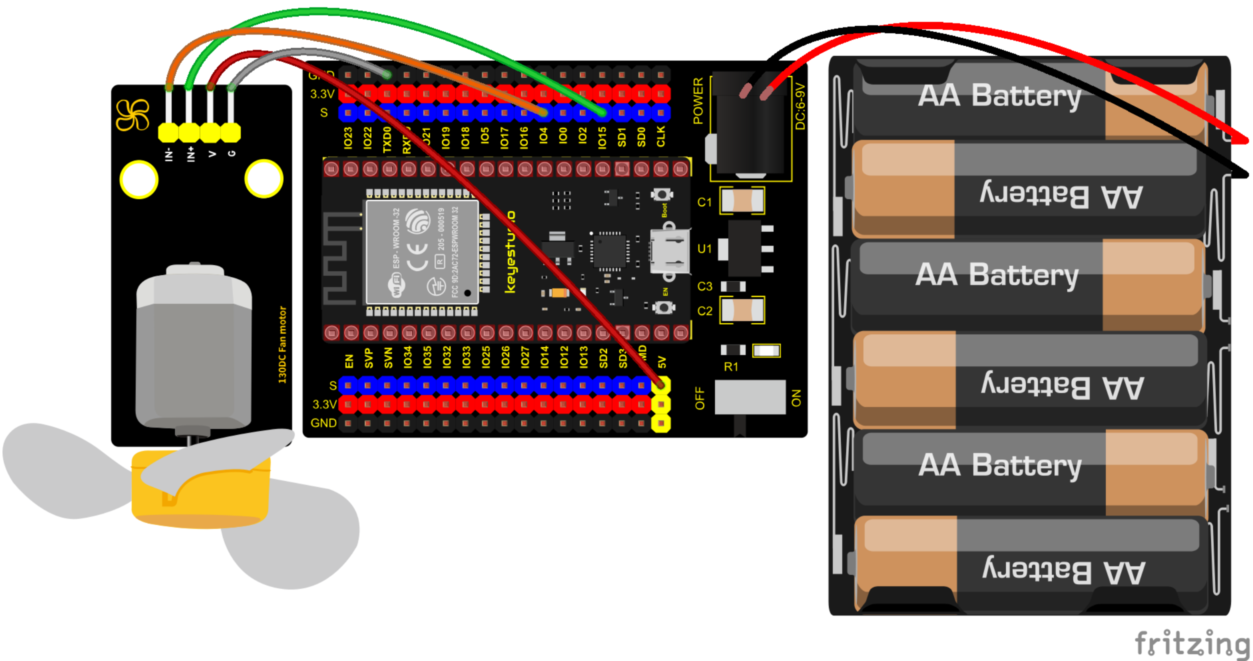
Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY 130 Motor*1	4P Dupont Wire*1
			
Micro USB Cable*1	Battery Holder*1	Battery (not included)*6	

Note: the motor is separated with its fan, you need to assemble it first.

Connection Diagram

130 Motor	ESP32 Expansion Board
G	G
V	5V
IN+	IO15
IN-	IO4



Test Code

```
from machine import Pin
import time

#Two pins of the motor
INA = Pin(15, Pin.OUT) #INA corresponds to IN+
INB = Pin(4, Pin.OUT)#INB corresponds to IN-

while True:
    #Counterclockwise 2s
    INA.value(1)
    INB.value(0)
    time.sleep(2)
    #stop 1s
    INA.value(0)
    INB.value(0)
    time.sleep(1)
    #Turn clockwise for 2s
    INA.value(0)
    INB.value(1)
    time.sleep(2)
    #stop 1s
    INA.value(0)
    INB.value(0)
    time.sleep(1)
```

Code Explanation

Set pins to GPIO4, GPIO15, when the pin GPIO4 outputs low levels and the pin GPIO15 outputs high levels, the motor will rotate counterclockwise; when both pins are set to low, the motor stops rotating.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch ON the ESP32 expansion board to the ON end, after powering on, click "Run current script", the code starts executing, then the fan will rotate counterclockwise for 2 s, stop for 1 s; and rotate clockwise for 2 s and stop for 1 s, cycle alternately. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

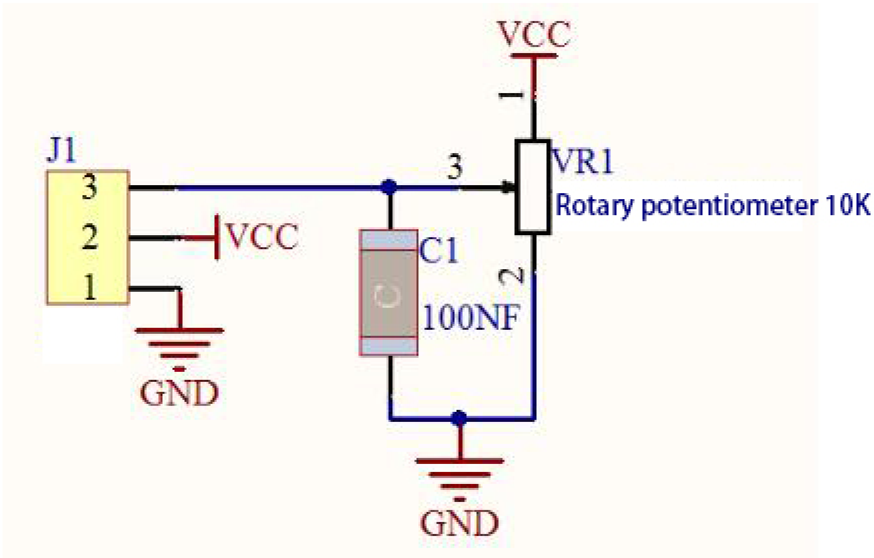
5.2.20 Project 20: Potentiometer

Overview

The following we will introduce is the Keyestudio rotary potentiometer which is an analog sensor.

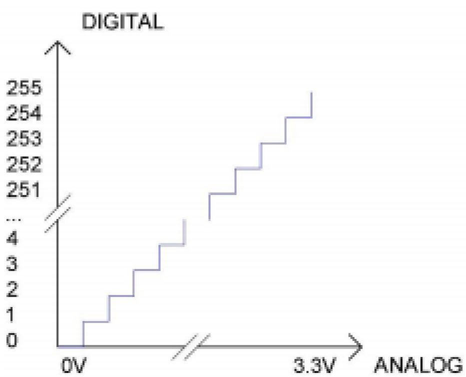
The digital IO ports can read the voltage value between 0 and 3.3V and the module only outputs high levels. However, the analog sensor can read the voltage value through 16 ADC analog ports on the ESP32 board. In the experiment, we will display the test results on the Shell.

Working Principle



It uses a 10K adjustable resistor. We can change the resistance by rotating the potentiometer. The signal S can detect the voltage changes(0-3.3V) which are analog quantity.

ADC The more bits an ADC has, the denser the partitioning of the simulation, the higher the accuracy of the final conversion.



Subsection 1: The analog value within 0V—3.3/4095 V corresponds to the number 0; Subsection 2: The analog value within 3.3/4095V—2*3.3/4095V corresponds to the number 1; ……

The conversion formula is as follows:

DAC The higher the precision of DAC, the higher the precision of the output voltage value.

The conversion formula is as follows:

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

ADC on ESP32

The ESP32 has 16 pins that can be used to measure analog signals. GPIO pin serial numbers and analog pin definitions are shown below:

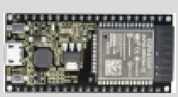
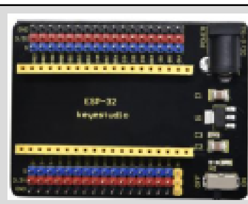
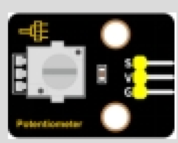
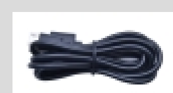
ADC number in ESP32	ESP32 GPIO number
ADC0	GPIO 36
ADC3	GPIO 39
ADC4	GPIO 32
ADC5	GPIO33
ADC6	GPIO34
ADC7	GPIO 35
ADC10	GPIO 4
ADC11	GPIO0
ADC12	GPIO2
ADC13	GPIO15
ADC14	GPIO13
ADC15	GPIO 12
ADC16	GPIO 14
ADC17	GPIO27
ADC18	GPIO25
ADC19	GPIO26

DAC on ESP32

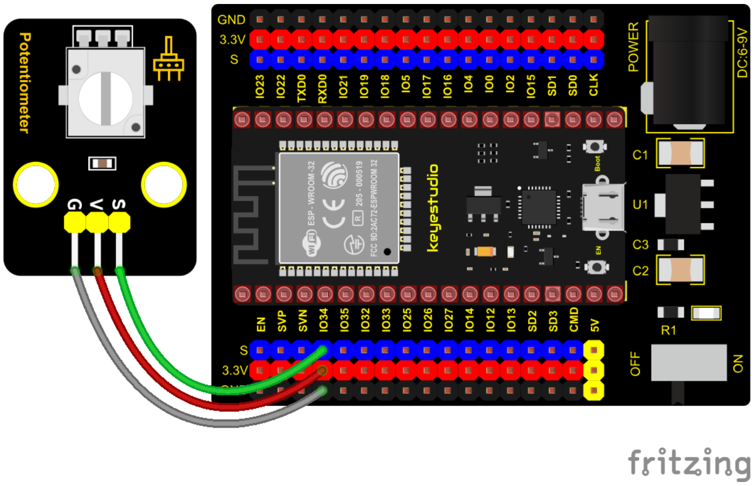
The ESP32 has two 8-bit digital-to-analog converters connected to GPIO25 and GPIO26 pins, which are immutable, as shown below :

Simulate pin number	GPIO number
DAC1	GPIO25
DAC2	GPIO26

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Potentiometer*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read ADC value once every 0.1seconds, convert ADC value to DAC value and output it,
# and print these data to “Shell”.
try:
    while True:
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

Code Explanation

1). In the experiment, add “From Machine import ADC” to the top of your Python file every time you use the ACD module, the same goes for DAC modules.

2). **machine.ADC(pin):** Create an ADC object associated with the given pin. 3). **pin:** The available pins are Pin(36)Pin(39)Pin(34Pin(35)Pin(32)Pin(33). DAC(pin). Create an DAC object associated with the given pin.

4). **machine.ADC(pin):** The available pins are pin (25) pin (26).

5). ADC. Read():Read ADC value and return ADC value.

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

dbattenuation ratio

ADC.ATTIN_0DB —full range of 1.2V

ADC.ATTN_2_5_DB —full range of 1.5V

ADC.ATTN_6DB —full range of 2.0 V

ADC.ATTN_11DB —full range of 3.3V

ADC.width(bit): Set data width.

bitdata bit

ADC.WIDTH_9BIT —9 data width

ADC.WIDTH_10BIT — 10 data width

ADC.WIDTH_11BIT — 11 data width

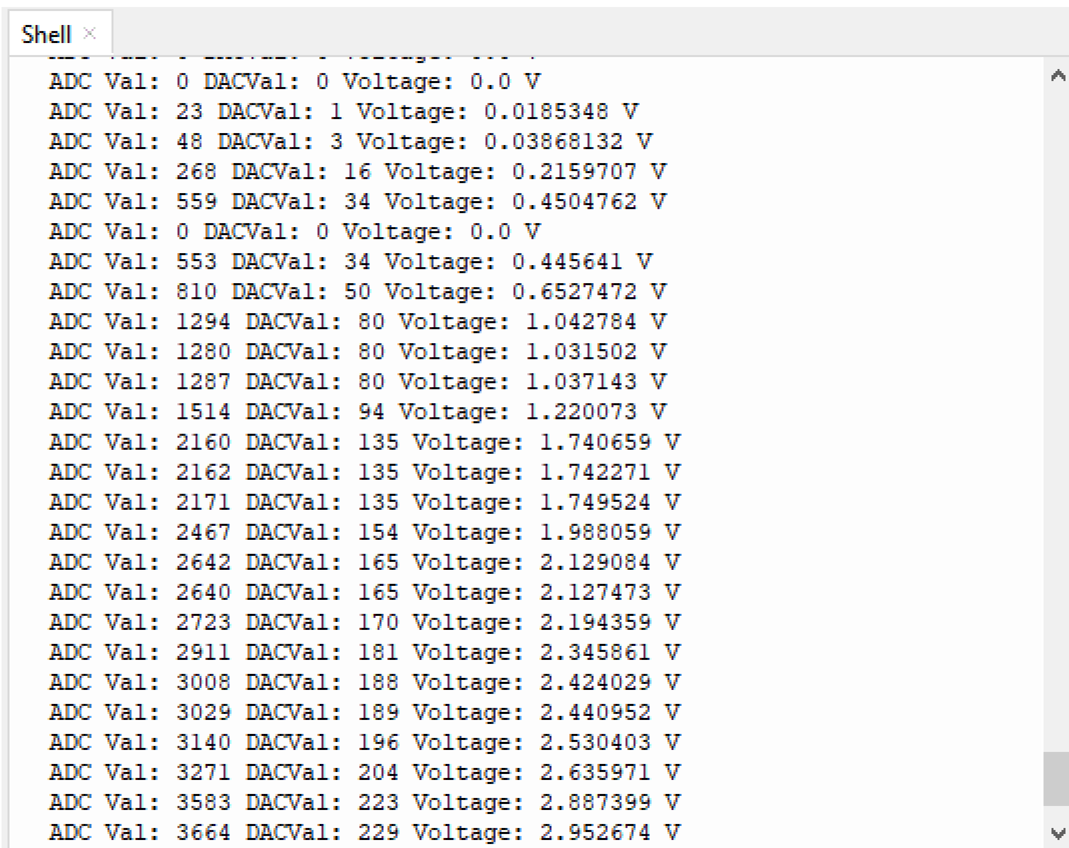
ADC.WIDTH_12BIT — 12 data width

7). **The read()method reads the** ADCvaluerang is 0~4095the **adc.read()** reads the ADC value input by the ADC(Pin(34)) Pin and assigns it to a variable named adcVal.

8). DAC.write(value):Output the voltage value, the data rang : 0-255the corresponding output voltage is 0-3.3V.

Test Result

Connect the wires according to the experimental wiring diagram and poweron. Click “Run current script”, the code starts executing. The “Shell” window prints and displays the potentiometer ADC value, DAC value and voltage value. Rotating the potentiometer handle, the ADC value, DAC value and voltage value will change. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.



```
Shell
ADC Val: 0 DACVal: 0 Voltage: 0.0 V
ADC Val: 23 DACVal: 1 Voltage: 0.0185348 V
ADC Val: 48 DACVal: 3 Voltage: 0.03868132 V
ADC Val: 268 DACVal: 16 Voltage: 0.2159707 V
ADC Val: 559 DACVal: 34 Voltage: 0.4504762 V
ADC Val: 0 DACVal: 0 Voltage: 0.0 V
ADC Val: 553 DACVal: 34 Voltage: 0.445641 V
ADC Val: 810 DACVal: 50 Voltage: 0.6527472 V
ADC Val: 1294 DACVal: 80 Voltage: 1.042784 V
ADC Val: 1280 DACVal: 80 Voltage: 1.031502 V
ADC Val: 1287 DACVal: 80 Voltage: 1.037143 V
ADC Val: 1514 DACVal: 94 Voltage: 1.220073 V
ADC Val: 2160 DACVal: 135 Voltage: 1.740659 V
ADC Val: 2162 DACVal: 135 Voltage: 1.742271 V
ADC Val: 2171 DACVal: 135 Voltage: 1.749524 V
ADC Val: 2467 DACVal: 154 Voltage: 1.988059 V
ADC Val: 2642 DACVal: 165 Voltage: 2.129084 V
ADC Val: 2640 DACVal: 165 Voltage: 2.127473 V
ADC Val: 2723 DACVal: 170 Voltage: 2.194359 V
ADC Val: 2911 DACVal: 181 Voltage: 2.345861 V
ADC Val: 3008 DACVal: 188 Voltage: 2.424029 V
ADC Val: 3029 DACVal: 189 Voltage: 2.440952 V
ADC Val: 3140 DACVal: 196 Voltage: 2.530403 V
ADC Val: 3271 DACVal: 204 Voltage: 2.635971 V
ADC Val: 3583 DACVal: 223 Voltage: 2.887399 V
ADC Val: 3664 DACVal: 229 Voltage: 2.952674 V
```

5.2.21 Project 21: Steam Sensor

Description

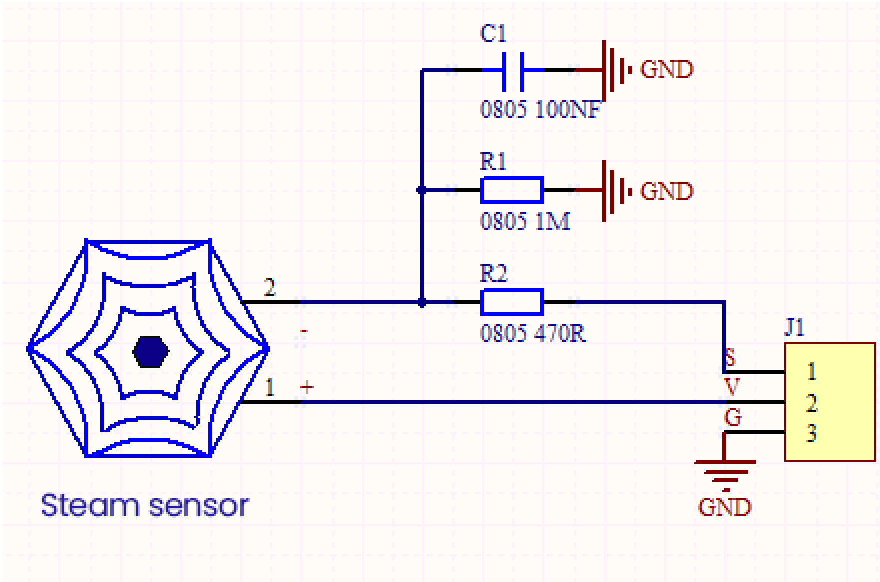
This is a DIY electronic building block water drop sensor. It is an analog (digital) input module, also called rain, rain sensor. It can be used to monitor various weather conditions, detect whether it is raining and the amount of rain, convert it into digital signal (DO) and analog signal (AO) output, and is widely used in Arduino robot kits, raindrops, rain sensors, and can be used for various circumstances. It can monitor various weather conditions, and convert it into digital signal and AO output, and can also be used for automobile automatic wiper system, intelligent lighting system and intelligent sunroof system.

In the experiment, we input the sensor signal terminal (S terminal) to the analog port of the ESP32 development board, sense the change of the analog value, and display the corresponding analog value on the shell.

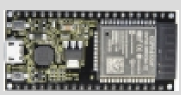
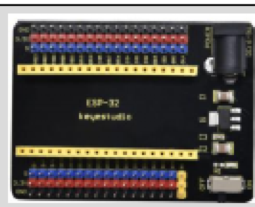
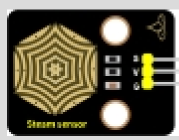
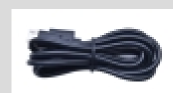
Working Principle

Its principle is to detect the amount of water through the exposed printed parallel lines on the circuit board. The more water there is, the more wires will be connected, and the conductive contact area increases. The voltage output by pin 2 will gradually increase. The larger the analog value detected by the signal terminal S is.

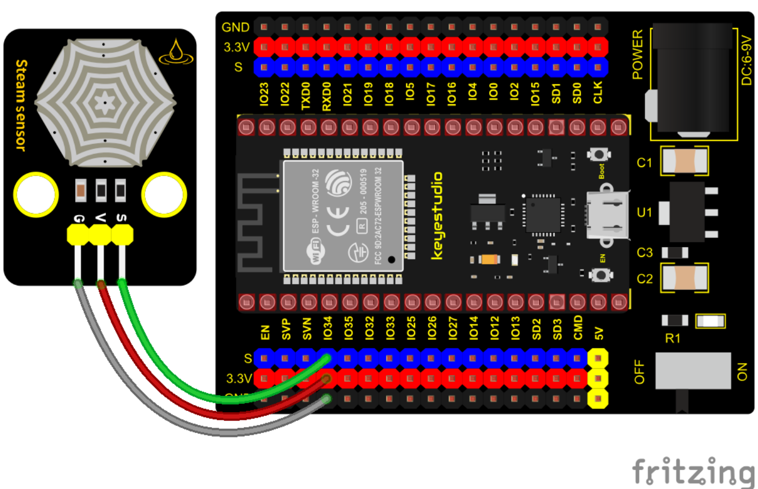
It can also detect steam in the air. Two position holes are used to install on the other devices.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Steam Sensor *1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
```

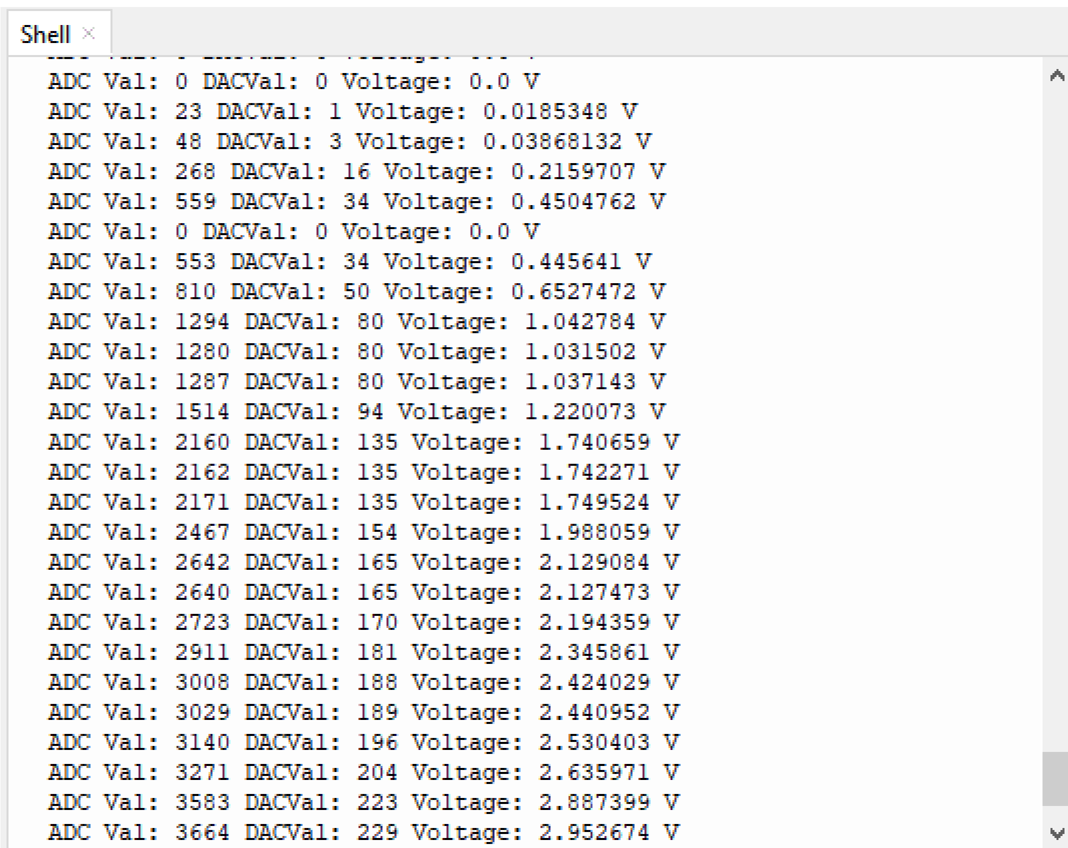
(continues on next page)

(continued from previous page)

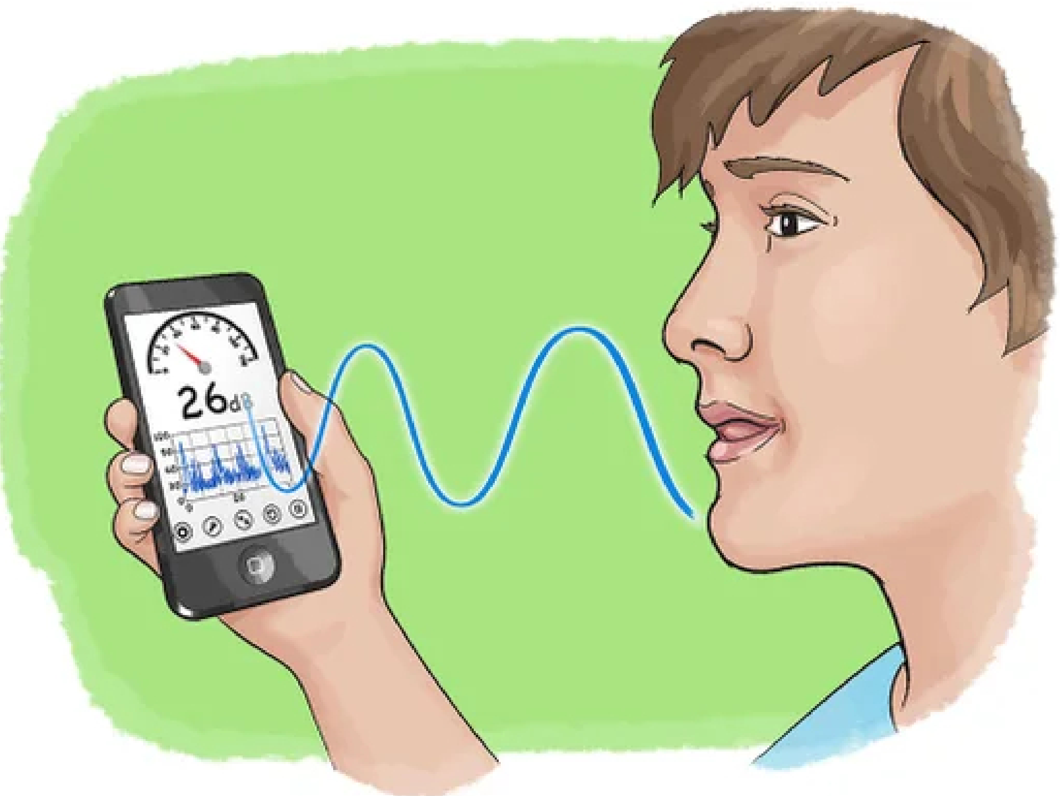
```
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read ADC value once every 0.1seconds, convert ADC value to DAC value and output it,
# and print these data to "Shell".
try:
    while True:
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

Test Result

Wiring up and powering on, then click "Run current script", the code starts executing. The Shell will display ADC value, DAC value and voltage value of the sensor. When a few drops of water are placed in the sensor sensing area, the ADC value, DAC value and voltage value will change. The more water volume, the greater the output voltage value , ADC value and the DAC value. Press"Ctrl+C"or click "Stop/Restart backend"to exit the program.



5.2.22 Project 22: Sound Sensor

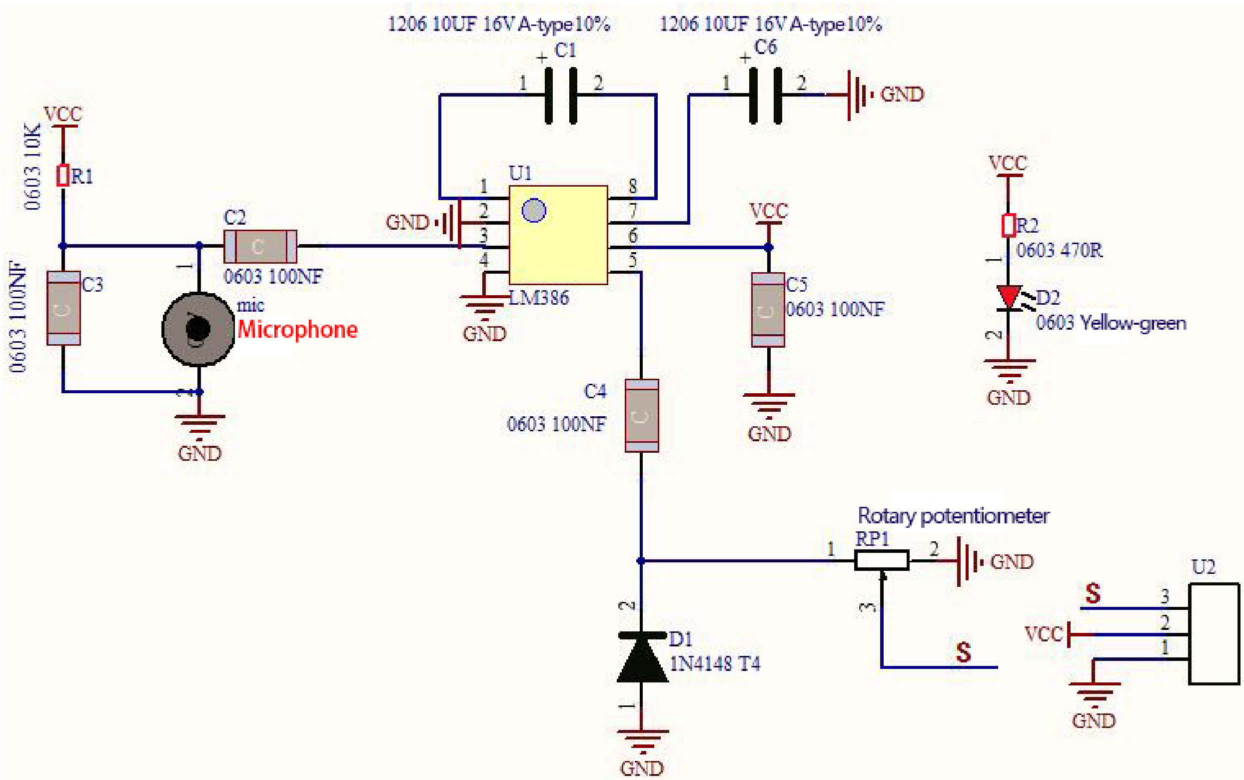


Overview

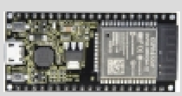
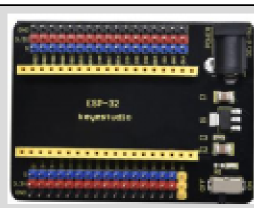
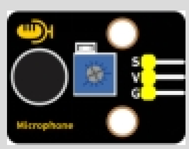
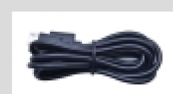
In this kit, there is a Keyestudio DIY electronic block and a sound sensor. In the experiment, we test the analog value corresponding to the sound level in the current environment with it. The louder the sound, the larger the ADC, DAC and the voltage value, and the "shell" window will display the test results.

Working Principle

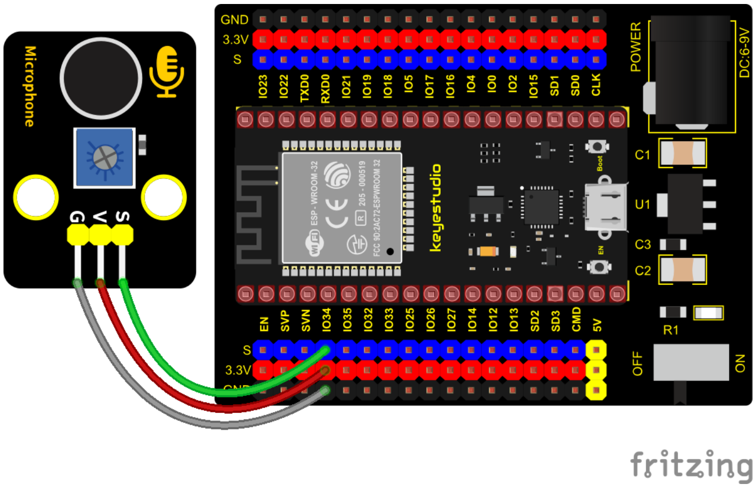
It uses a high-sensitive microphone component and an LM386 chip. We build the circuit with the LM386 chip and amplify the sound through the high-sensitive microphone. In addition, we can adjust the sound volume by the potentiometer. Rotate it clockwise, the sound will get louder.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Sound Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```python
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read ADC value once every 0.1seconds, convert ADC value to DAC value and output it,
# and print these data to “Shell”.
try:
    while True:
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

Test Result

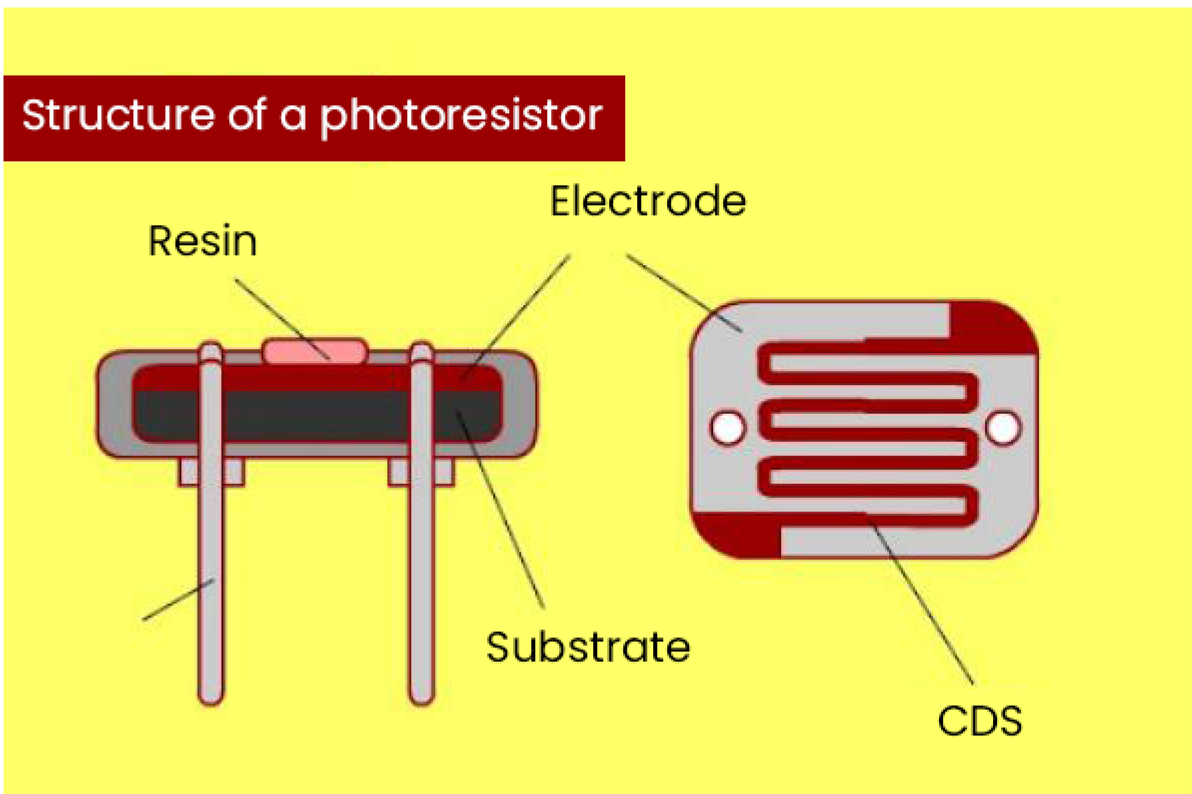
Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The “Shell” window will display the sound sensor ADC value, DAC value and voltage value.

Rotate the potentiometer clockwise and speak at the MIC. Then you can see the analog value get larger, as shown below.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

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

  ADC Val: 0 DACVal: 0 Voltage: 0.0 V
  ADC Val: 23 DACVal: 1 Voltage: 0.0185348 V
  ADC Val: 1520 DACVal: 95 Voltage: 1.224908 V
  ADC Val: 551 DACVal: 34 Voltage: 0.4440293 V
  ADC Val: 2285 DACVal: 142 Voltage: 1.841392 V
  ADC Val: 1395 DACVal: 87 Voltage: 1.124176 V
  ADC Val: 0 DACVal: 0 Voltage: 0.0 V
  ADC Val: 1902 DACVal: 118 Voltage: 1.532747 V
  ADC Val: 0 DACVal: 0 Voltage: 0.0 V
  ADC Val: 0 DACVal: 0 Voltage: 0.0 V
  ADC Val: 633 DACVal: 39 Voltage: 0.5101099 V
```

5.2.23 Project 23: Photoresistor

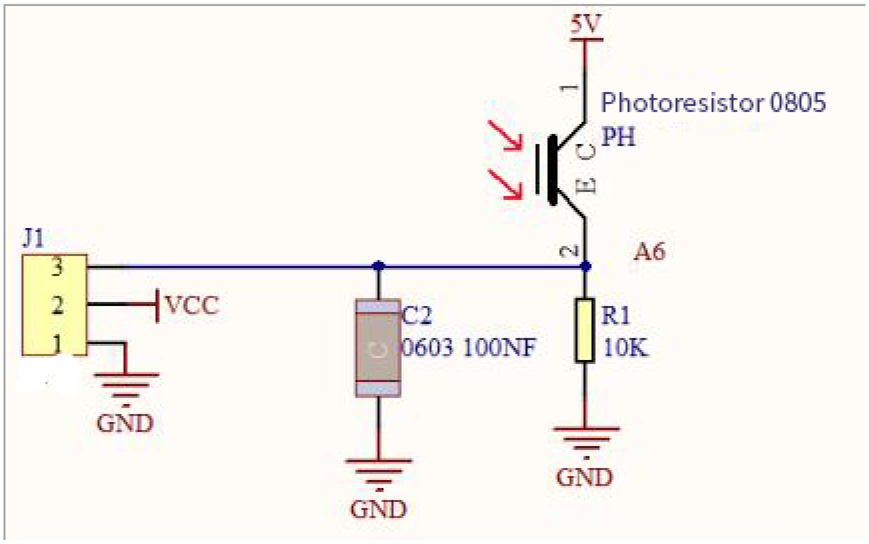


Description

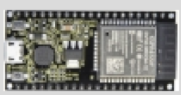
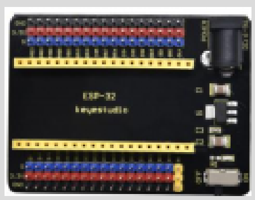
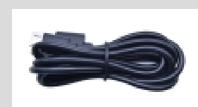
In this kit, there is a photoresistor which consists of photosensitive resistance elements. Its resistance changes with the light intensity. Also, it converts the resistance change into a voltage change through the characteristic of the photosensitive resistive element. When wiring it up, we interface its signal terminal (S terminal) with the analog port of ESP32 , so as to sense the change of the analog value, and display the corresponding analog value in the shell.

Working Principle

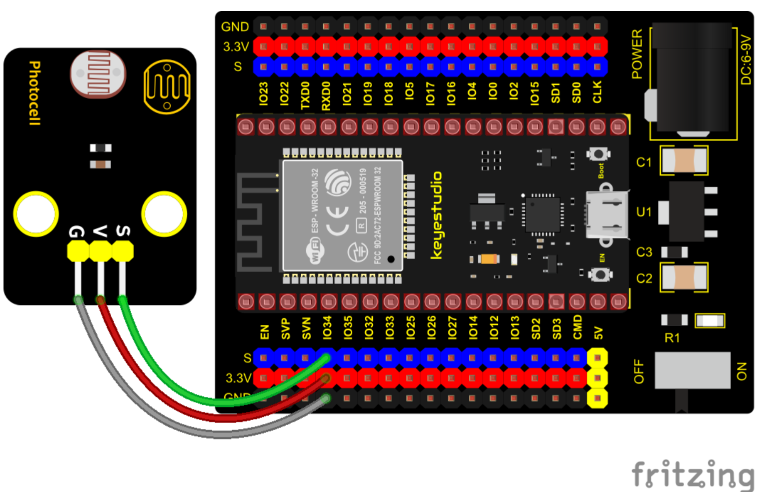
If there is no light, the resistance is 0.2M and the detected voltage at the terminal 2 is close to 0. When the light intensity increases, the resistance of photoresistor and detected voltage will diminish, and the detected voltage is increasing.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Photoresistor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time
```

(continues on next page)

(continued from previous page)

```
# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read ADC value once every 0.1seconds, convert ADC value to DAC value and output it,
# and print these data to “Shell”.
try:
    while True:
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

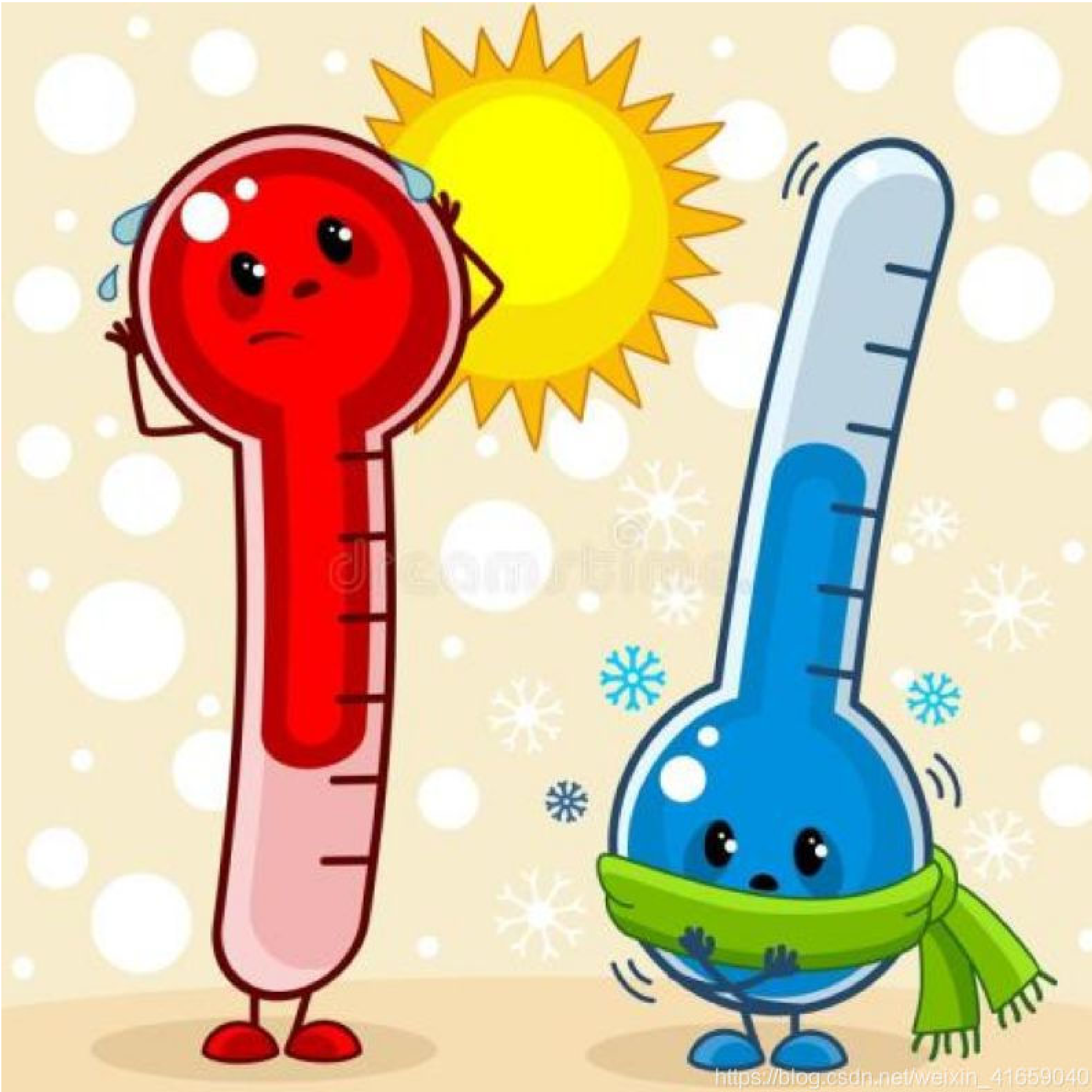
Test Result

Connect the wires according to the experimental wiring diagram and poweron. Click “Run current script”, the code starts executing. The “Shell” window will display the photoresistor ADC value, DAC value and voltage value. The brighter the light, the greater the analog value, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

Shell

```
ADC Val: 1472 DACVal: 92 Voltage: 1.186227 V
ADC Val: 1645 DACVal: 102 Voltage: 1.325641 V
ADC Val: 1847 DACVal: 115 Voltage: 1.488425 V
ADC Val: 2043 DACVal: 127 Voltage: 1.646374 V
ADC Val: 2254 DACVal: 140 Voltage: 1.81641 V
ADC Val: 2442 DACVal: 152 Voltage: 1.967912 V
ADC Val: 2625 DACVal: 164 Voltage: 2.115385 V
ADC Val: 2752 DACVal: 172 Voltage: 2.217729 V
ADC Val: 2832 DACVal: 177 Voltage: 2.282198 V
ADC Val: 2880 DACVal: 180 Voltage: 2.320879 V
ADC Val: 2887 DACVal: 180 Voltage: 2.32652 V
ADC Val: 2873 DACVal: 179 Voltage: 2.315238 V
ADC Val: 2922 DACVal: 182 Voltage: 2.354725 V
ADC Val: 2991 DACVal: 186 Voltage: 2.41033 V
ADC Val: 3051 DACVal: 190 Voltage: 2.458681 V
ADC Val: 3103 DACVal: 193 Voltage: 2.500586 V
ADC Val: 3145 DACVal: 196 Voltage: 2.534432 V
ADC Val: 3163 DACVal: 197 Voltage: 2.548938 V
ADC Val: 3181 DACVal: 198 Voltage: 2.563443 V
ADC Val: 3187 DACVal: 199 Voltage: 2.568278 V
ADC Val: 3214 DACVal: 200 Voltage: 2.590037 V
ADC Val: 3314 DACVal: 207 Voltage: 2.670623 V
```

5.2.24 Project 24: NTC-MF52AT Thermistor

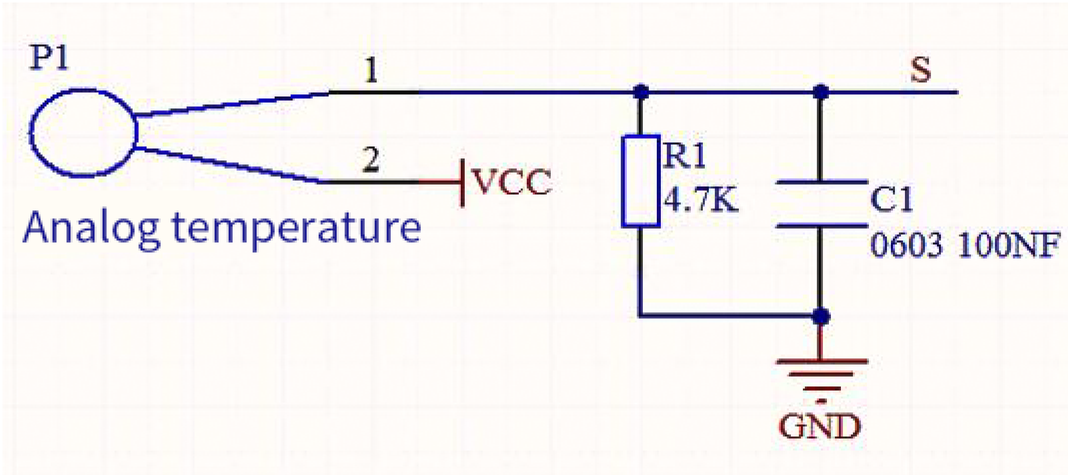


Overview

In the experiment, there is a NTC-MF52AT analog thermistor. We connect its signal terminal to the analog port of the ESP32 mainboard and read the corresponding ADC value, voltage value and thermistor value.

We can use analog values to calculate the temperature of the current environment through specific formulas. Since the temperature calculation formula is more complicated, we only read the corresponding analog value.

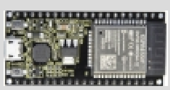
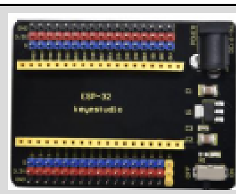
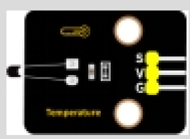
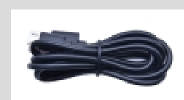
Working Principle



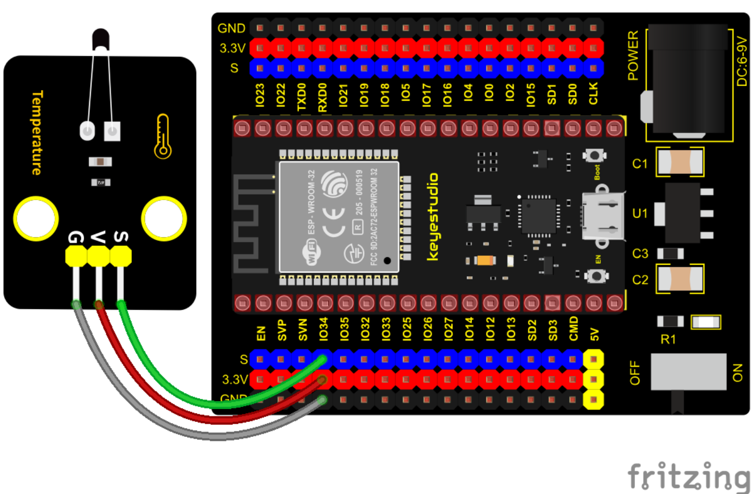
This module mainly uses NTC-MF52AT thermistor element, which can can sense the changes of the surrounding environment temperature. Resistance changes with the temperature, causing the voltage of the signal terminal S to change.

This sensor uses the characteristics of NTC-MF52AT thermistor element to convert resistance changes into voltage changes.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio NTC-MF52AT Thermistor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram

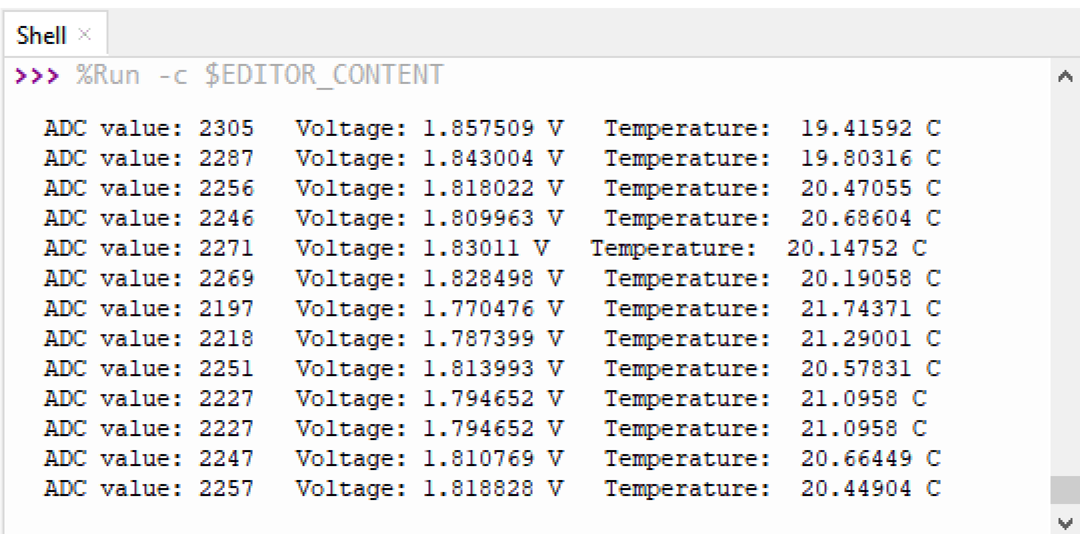


Test Code

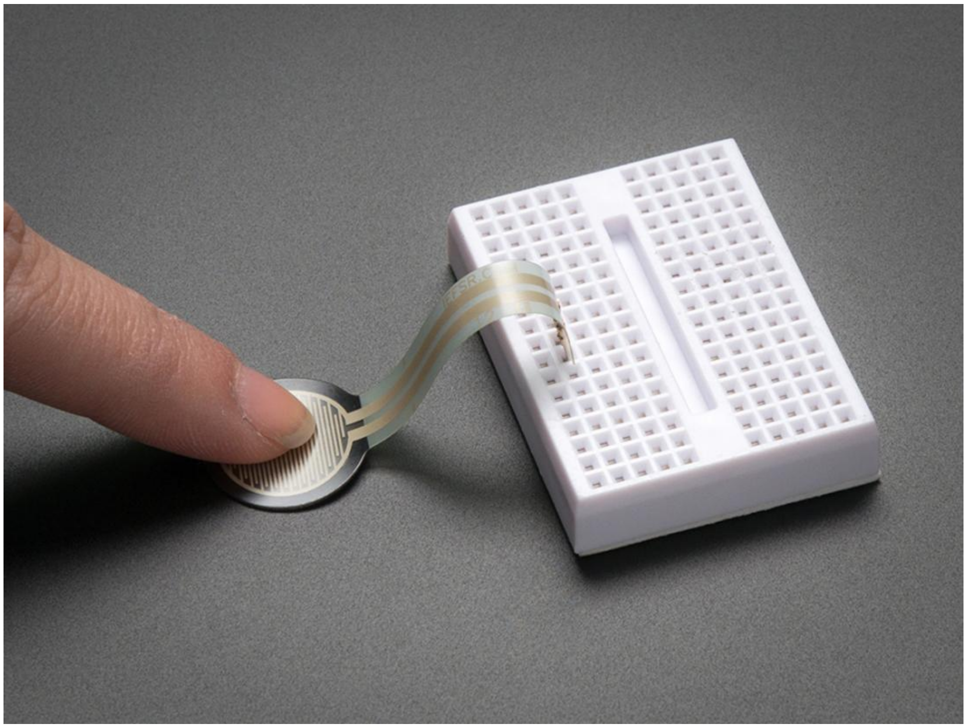
```python
from machine import Pin, ADC
import time
import math

#Set ADC
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

try:
    while True:
        adcValue = adc.read()
        voltage = adcValue / 4095 * 3.3
        Rt = (3.3 - voltage) / voltage * 4.7;
        tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
        tempC = (tempK - 273.15)
        print("ADC value:",adcValue,"  Voltage:",voltage,"V","  Temperature: ",tempC,"C");
        time.sleep(1)
except:
    pass
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The "Shell" window will display the thermistor ADC value, voltage value and temperature value, as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.



5.2.25 Project 25: Thin-film Pressure Sensor



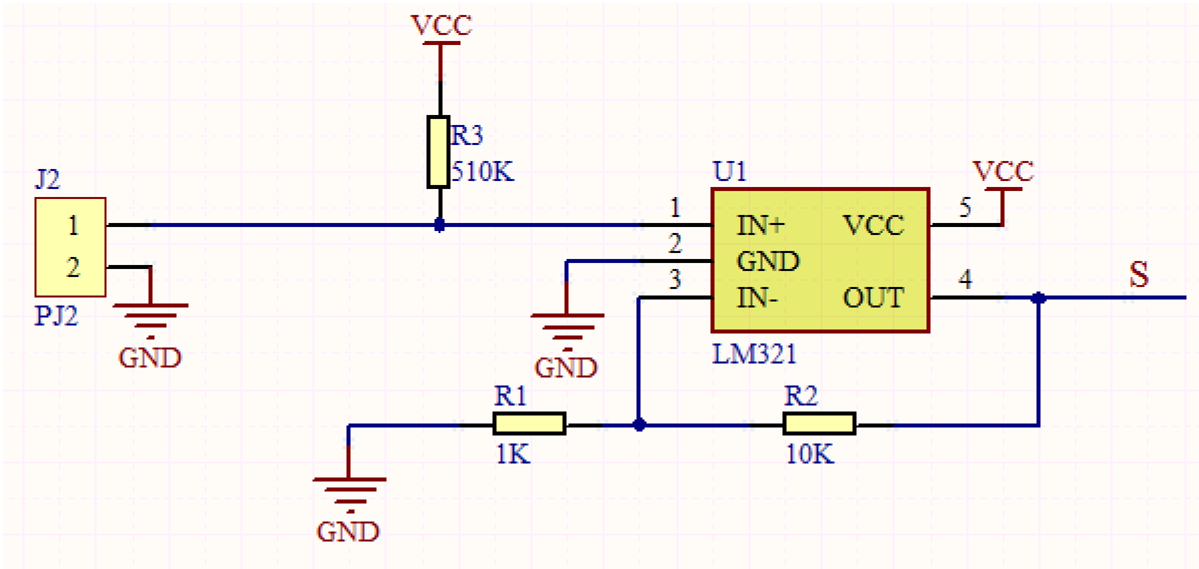
Overview

In this kit, there is a Keyestudio thin-film pressure sensor. The thin-film pressure sensor composed of a new type of nano pressure-sensitive material and a comfortable ultra-thin film substrate, has waterproof and pressure-sensitive functions.

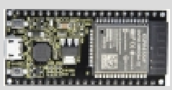
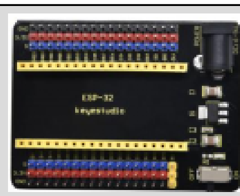
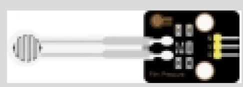
In the experiment, we determine the pressure by collecting the analog signal on the S end of the module. The smaller the ADC value, DAC value and voltage value, the greater the pressure; and the displayed results will shown on the Shell.

Working Principle

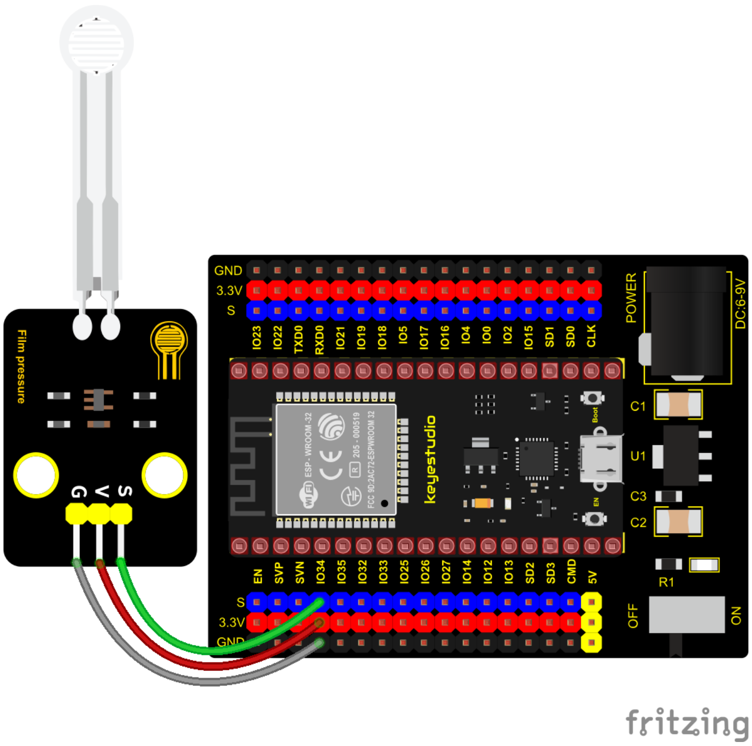
When the sensor is pressed by external forces, the resistance value of sensor will vary. We convert the pressure signals detected by the sensor into the electric signals through a circuit. Then we can obtain the pressure changes by detecting voltage signal changes.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	KeyestudioThin-film Pressure Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read ADC value once every 0.1seconds, convert ADC value to DAC value and output it,
# and print these data to “Shell”.
try:
    while True:
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The “Shell” window will display the thin-film pressure sensor ADC value, voltage value and DAC value. When the thin-film is pressed by fingers, the analog value will decrease, as shown below. Press “Ctrl+C”or click “Stop/Restart backend” to exit the program.

```
Shell
ADC Val: 2278 DACVal: 142 Voltage: 1.835751 V
ADC Val: 2278 DACVal: 142 Voltage: 1.835751 V
ADC Val: 2275 DACVal: 142 Voltage: 1.833333 V
ADC Val: 2277 DACVal: 142 Voltage: 1.834945 V
ADC Val: 2278 DACVal: 142 Voltage: 1.835751 V
ADC Val: 1755 DACVal: 109 Voltage: 1.414286 V
ADC Val: 1153 DACVal: 72 Voltage: 0.9291575 V
ADC Val: 808 DACVal: 50 Voltage: 0.6511355 V
ADC Val: 851 DACVal: 53 Voltage: 0.6857876 V
ADC Val: 593 DACVal: 37 Voltage: 0.4778755 V
ADC Val: 339 DACVal: 21 Voltage: 0.2731868 V
ADC Val: 400 DACVal: 25 Voltage: 0.3223443 V
ADC Val: 349 DACVal: 21 Voltage: 0.2812454 V
ADC Val: 483 DACVal: 30 Voltage: 0.3892307 V
ADC Val: 318 DACVal: 19 Voltage: 0.2562637 V
ADC Val: 325 DACVal: 20 Voltage: 0.2619048 V
ADC Val: 368 DACVal: 23 Voltage: 0.2965568 V
ADC Val: 424 DACVal: 26 Voltage: 0.341685 V
```

5.2.26 Project 26: Flame Sensor

Description

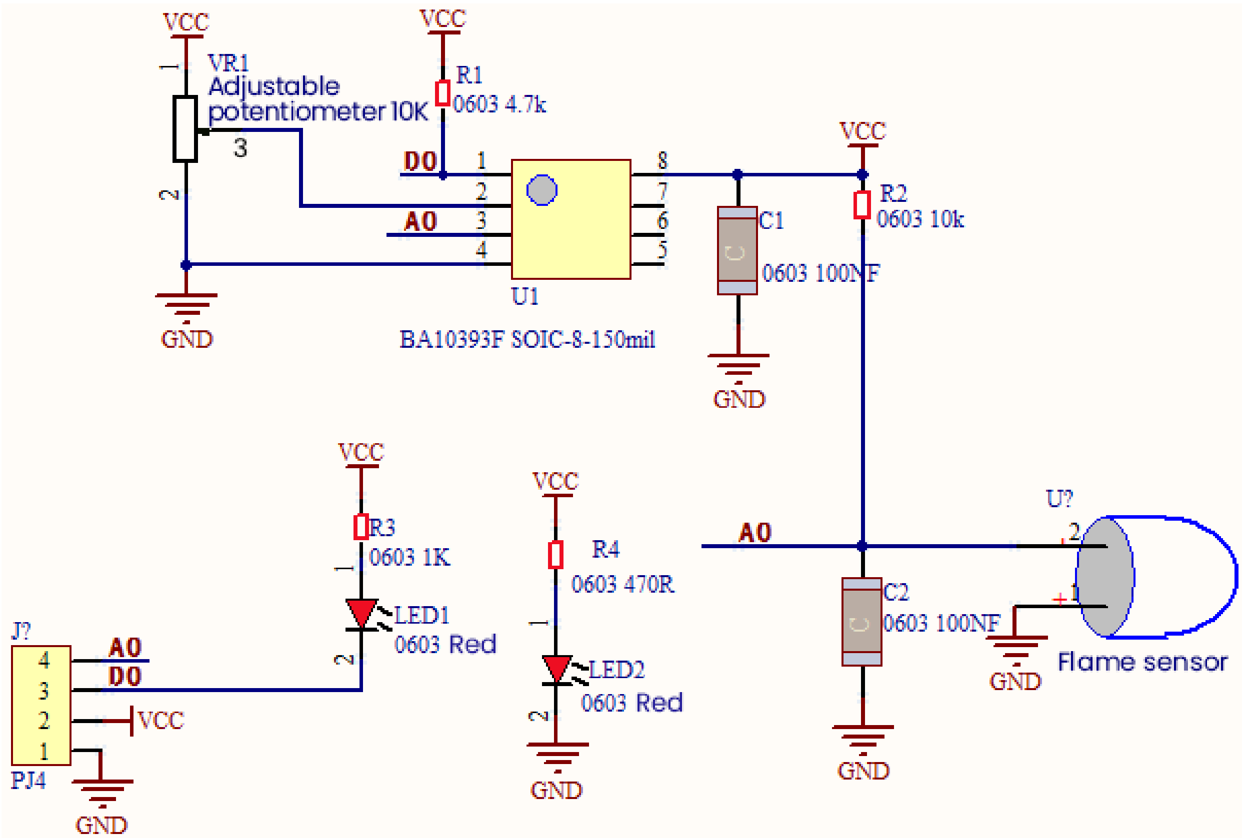
In daily life, it is often seen that a fire broke out without any precaution. It will cause great economic and human loss. So how can we avoid this situation? Right, install a flame sensor and a speaker in those places that easily break out a fire. When the flame sensor detects a fire, the speaker will alarm people quickly to put out the fire.

So in this project, you will learn how to use a flame sensor and an active buzzer module to simulate the fire alarm system.

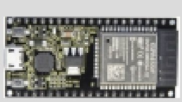
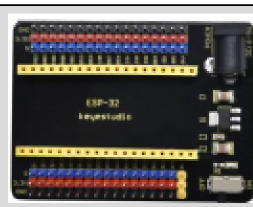
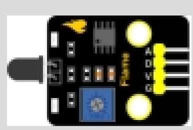
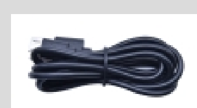
Working Principle

This flame sensor can be used to detect fire or other light sources with wavelength stands at 700nm ~ 1000nm. Its detection angle is about 60°. You can rotate the potentiometer on the sensor to control its sensitivity. Adjust the potentiometer to make the LED at the critical point between on and off state. The sensitivity is the best.

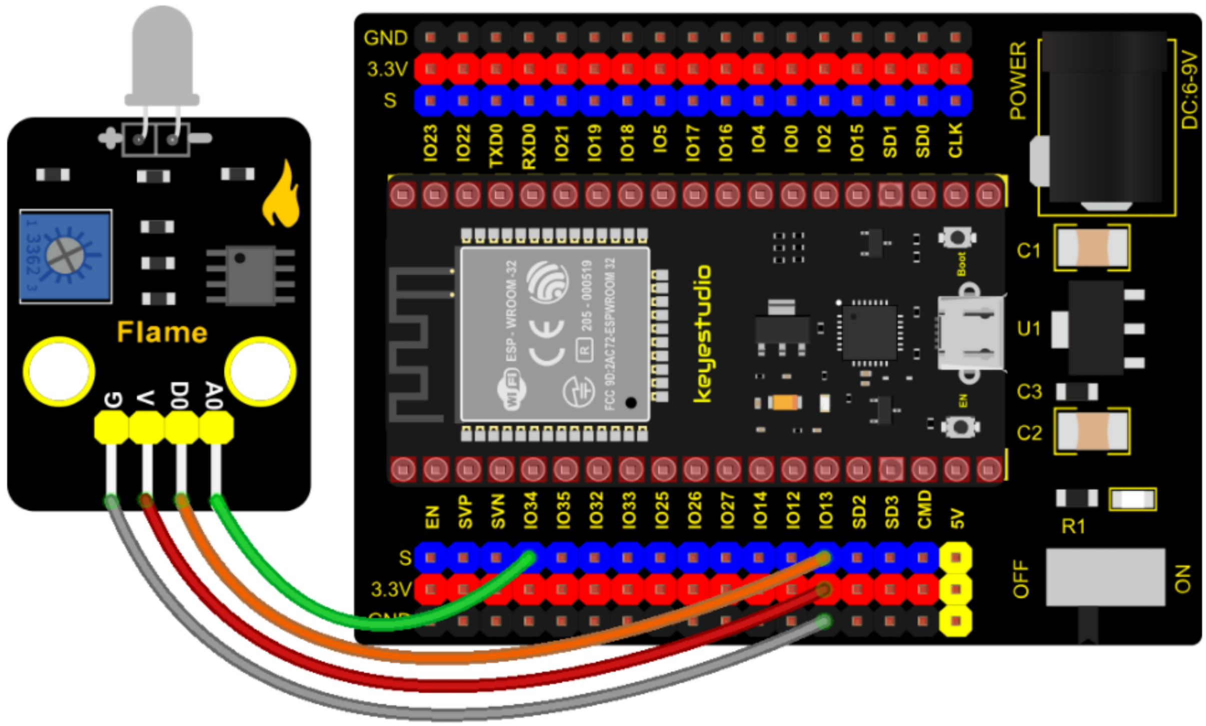
From the below figure, power up. When detecting fire, the digital pin outputs low levels, the red LED2 will light up first, the digital signal terminal D0 outputs a low level, and the red LED1 will light up. The stronger the external infrared light, the smaller the value; the weaker the infrared light, the larger the value.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Flame Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

flame_D = Pin(13, Pin.IN)
# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read digital value and ADC value once every 0.1seconds, convert ADC value to DAC value and Voltage value and output it,
# and print these data to “Shell”.
try:
    while True:
        digitalVal = flame_D.value()
        adcVal=adc.read()
        dacVal=adcVal//16
        voltage = adcVal / 4095.0 * 3.3
        print("digitalVal:",digitalVal,"ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
        time.sleep(0.1)
except:
    pass
```

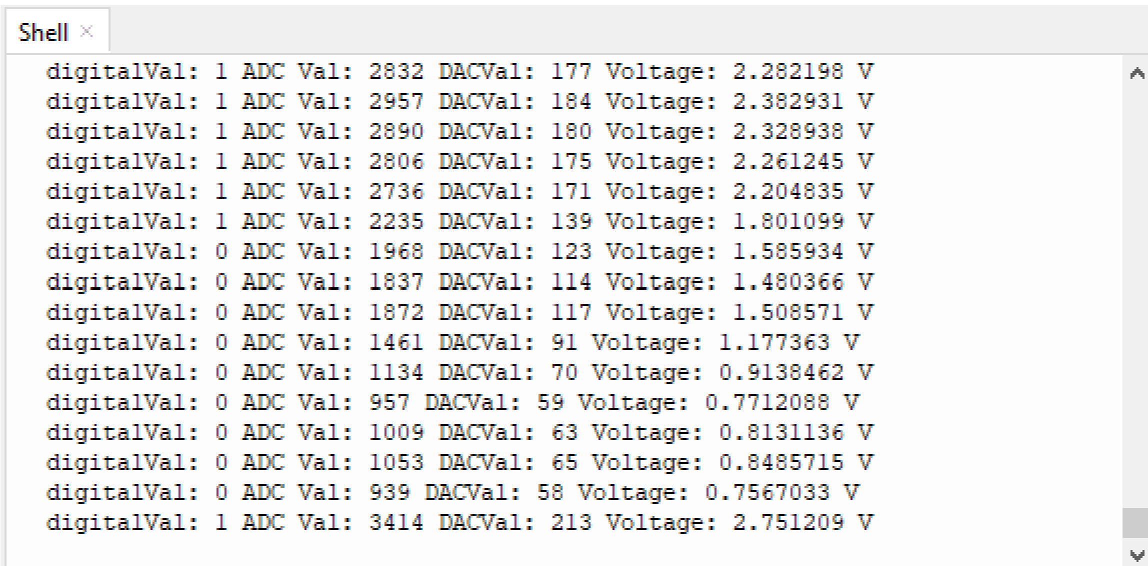
Code Explanation

Two pins we use are defined as GPIO13 and GPIO34 according to the wiring-up diagram, and print digital signals and analog signals respectively.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. After powering on, rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point. The red LED2 on the sensor module is lit, while the red LED1 is not.

The “Shell” window will print and display the digital value, ADC value, DAC value and voltage value of the flame sensor. When fire is detected, the LED1 will be on. the digital value will change from 1 to 0, and the analog value will become smaller, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.



```
Shell
digitalVal: 1 ADC Val: 2832 DACVal: 177 Voltage: 2.282198 V
digitalVal: 1 ADC Val: 2957 DACVal: 184 Voltage: 2.382931 V
digitalVal: 1 ADC Val: 2890 DACVal: 180 Voltage: 2.328938 V
digitalVal: 1 ADC Val: 2806 DACVal: 175 Voltage: 2.261245 V
digitalVal: 1 ADC Val: 2736 DACVal: 171 Voltage: 2.204835 V
digitalVal: 1 ADC Val: 2235 DACVal: 139 Voltage: 1.801099 V
digitalVal: 0 ADC Val: 1968 DACVal: 123 Voltage: 1.585934 V
digitalVal: 0 ADC Val: 1837 DACVal: 114 Voltage: 1.480366 V
digitalVal: 0 ADC Val: 1872 DACVal: 117 Voltage: 1.508571 V
digitalVal: 0 ADC Val: 1461 DACVal: 91 Voltage: 1.177363 V
digitalVal: 0 ADC Val: 1134 DACVal: 70 Voltage: 0.9138462 V
digitalVal: 0 ADC Val: 957 DACVal: 59 Voltage: 0.7712088 V
digitalVal: 0 ADC Val: 1009 DACVal: 63 Voltage: 0.8131136 V
digitalVal: 0 ADC Val: 1053 DACVal: 65 Voltage: 0.8485715 V
digitalVal: 0 ADC Val: 939 DACVal: 58 Voltage: 0.7567033 V
digitalVal: 1 ADC Val: 3414 DACVal: 213 Voltage: 2.751209 V
```

5.2.27 Project 27: MQ-2 Gas Sensor

Description

This analog gas sensor - MQ2 is used in gas leakage detecting equipment in consumer electronics and industrial markets.

This sensor is suitable for detecting LPG, I-butane, propane, methane, alcohol, Hydrogen and smoke. It has high sensitivity and quick response.

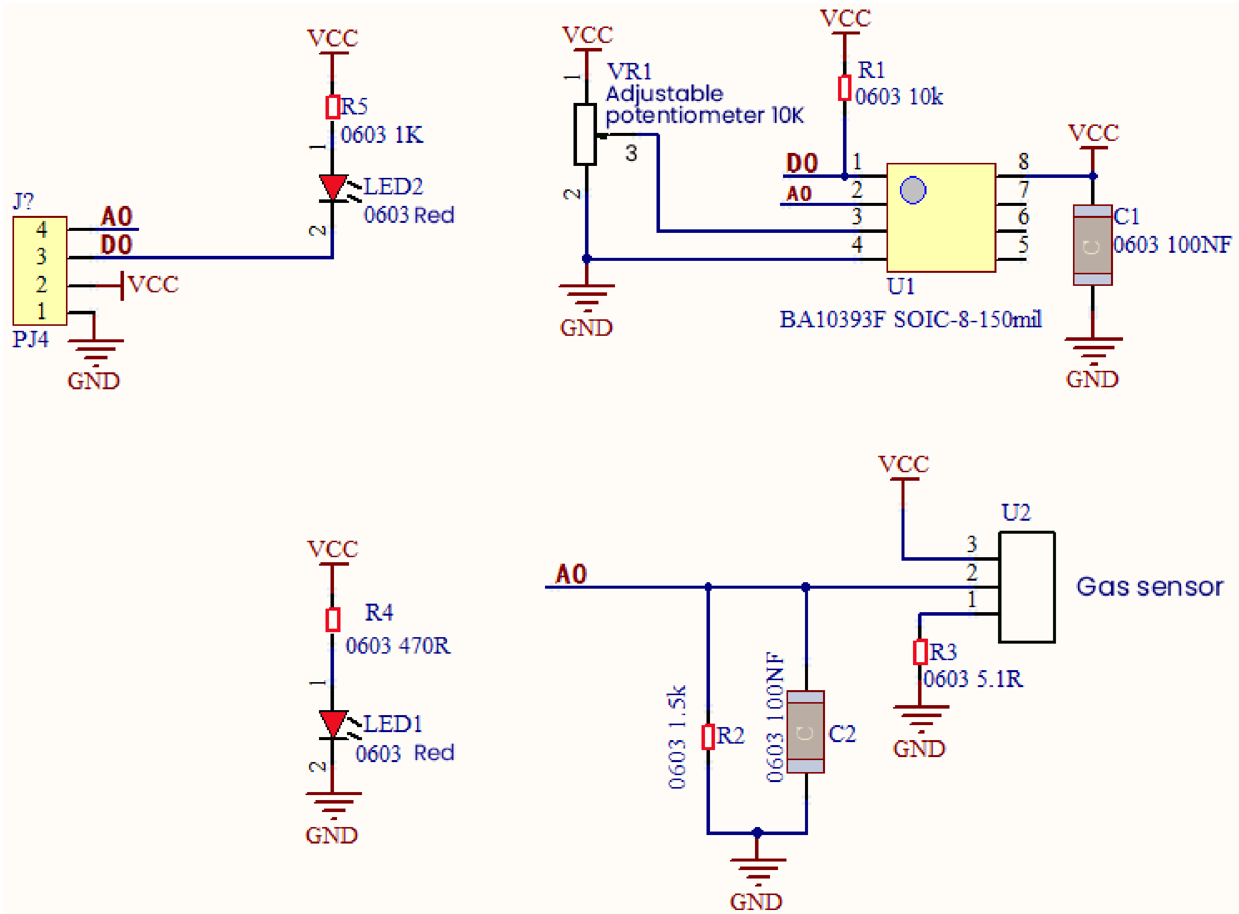
In addition, the sensitivity can be adjusted by rotating the potentiometer.

In the experiment, we read the analog value at the A0 port and the D0 port to determine the content of gas.

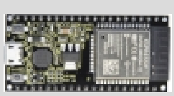
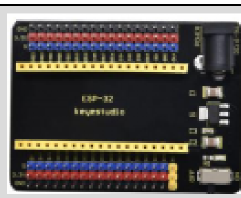
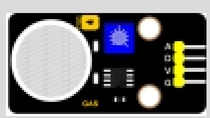
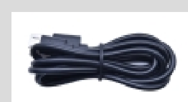
Working Principle

The greater the concentration of smoke, the greater the conductivity, the lower the output resistance, the greater the output analog signal.

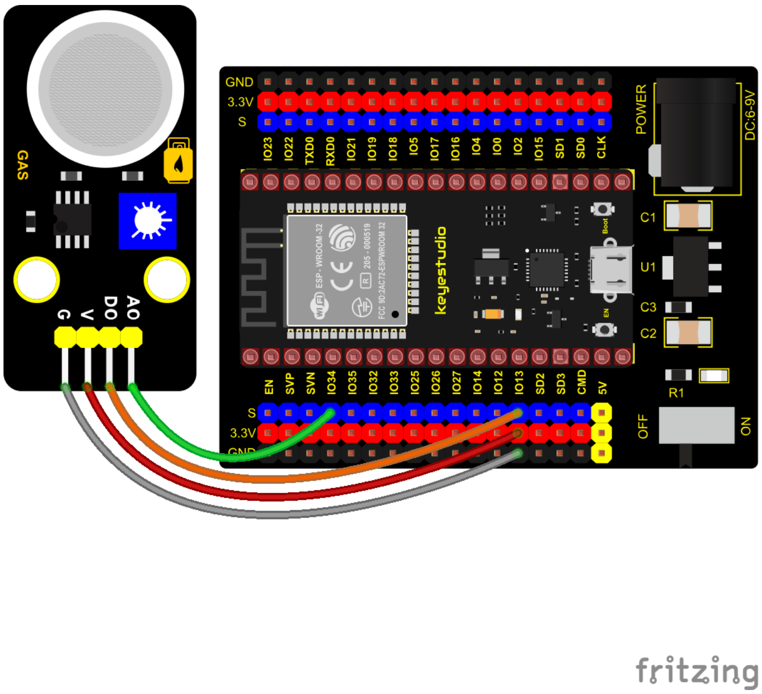
When in use, the A0 terminal reads the analog value of the corresponding gas; the D0 terminal is connected to an LM393 chip (voltage comparator), we can adjust the alarm threshold of the measured gas through the potentiometer, and output the digital value at D0. When the measured gas content exceeds the critical point, the D0 terminal outputs a low level. When the measured gas content does not exceed the critical point, the D0 terminal outputs a high level.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Analog Gas Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



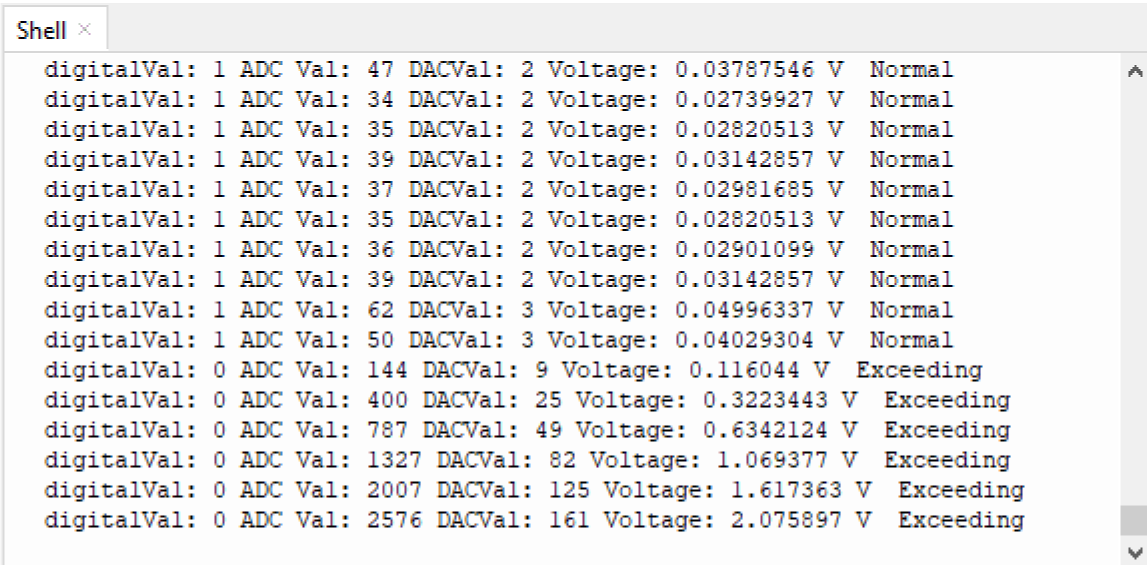
Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

# Turn on and configure the ADC with the range of 0-3.3V
mq2_D = Pin(13, Pin.IN)
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read digital value and ADC value once every 0.1seconds, convert ADC value to DAC value and Voltage value and output it,
# and print these data to “Shell”.

while True:
    digitalVal = mq2_D.value()
    adcVal=adc.read()
    dacVal=adcVal//16
    voltage = adcVal / 4095.0 * 3.3
    print("digitalVal:",digitalVal,"ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V", end = "   ")
    if digitalVal == 0:
        print("Exceeding")
    else:
        print("Normal")
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The "shell" window will display the corresponding data and string. After powering on, by rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point.

When the sensor detects the smoke or combustible gas, the red LED lights up and the digital value in the "Shell" window changes from 1 to 0, the ADC value, DAC value and voltage value increase, as shown below. Press "Ctrl+C"or click  "Stop/Restart backend"to exit the program.



5.2.28 Project 28: MQ-3 Alcohol Sensor

Description

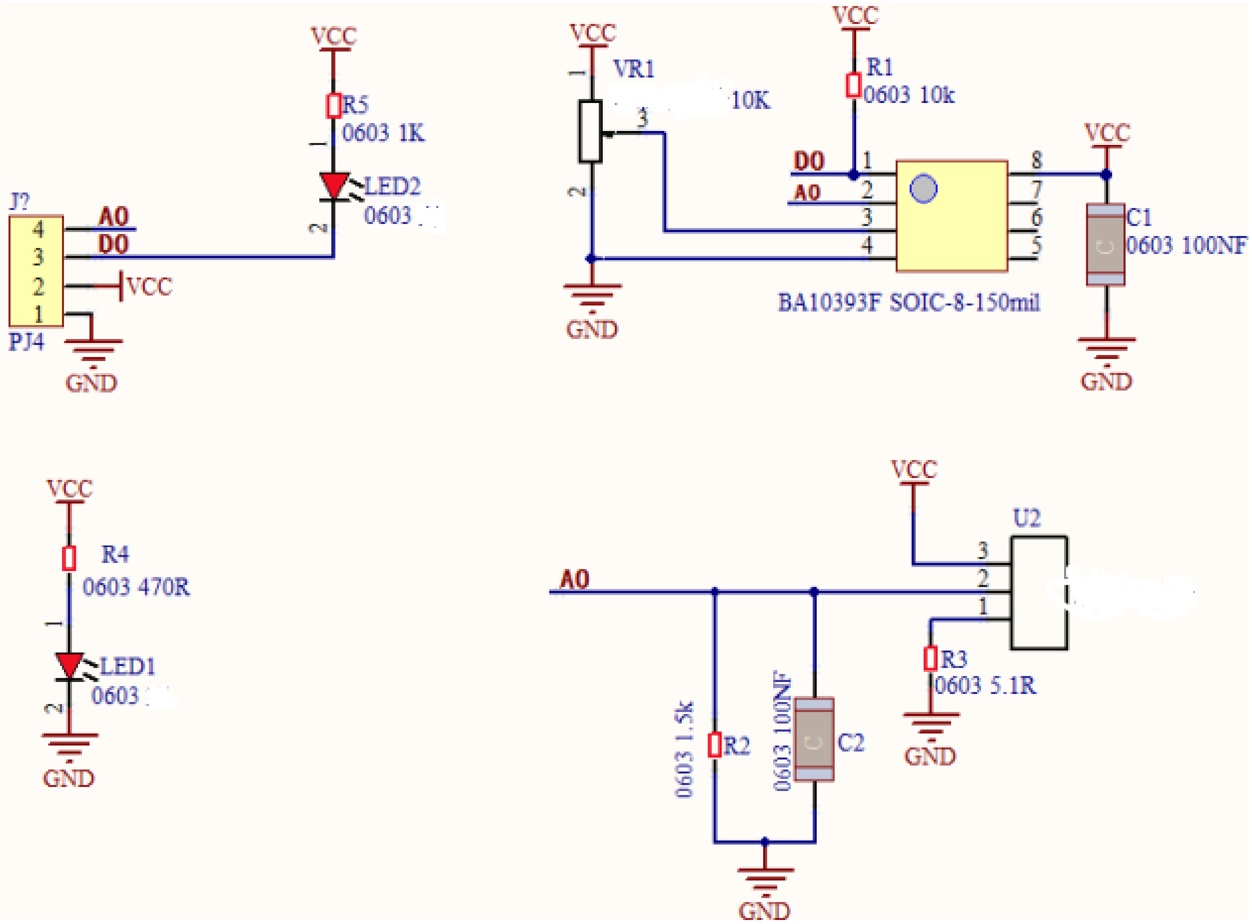
In this kit, there is a MQ-3 alcohol sensor, which uses the gas-sensing material is tin dioxide (SnO2) which has a low conductivity in clean air. When there is alcohol vapor in the environment where the sensor is located, the conductivity of the sensor increases with the increase of the alcohol gas concentration in the air. The change in conductivity can be converted into an output signal corresponding to the gas concentration using a simple circuit.

In the experiment, we read the analog value at the A0 end of the sensor and the digital value at the D0 end to judge the content of alcohol vapor in the air and whether they exceed the standard.

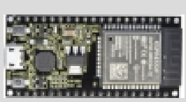
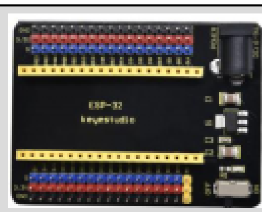
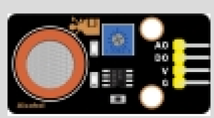
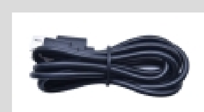
Working Principle

At a certain temperature, the conductivity changes with the composition of the ambient gas. When in use, A0 terminal reads the analog value corresponding to alcohol vapor; D0 terminal is connected to an LM393 chip (comparator), we can adjust and measure the alcohol vapor alarm threshold through the potentiometer, and output the digital value at D0.

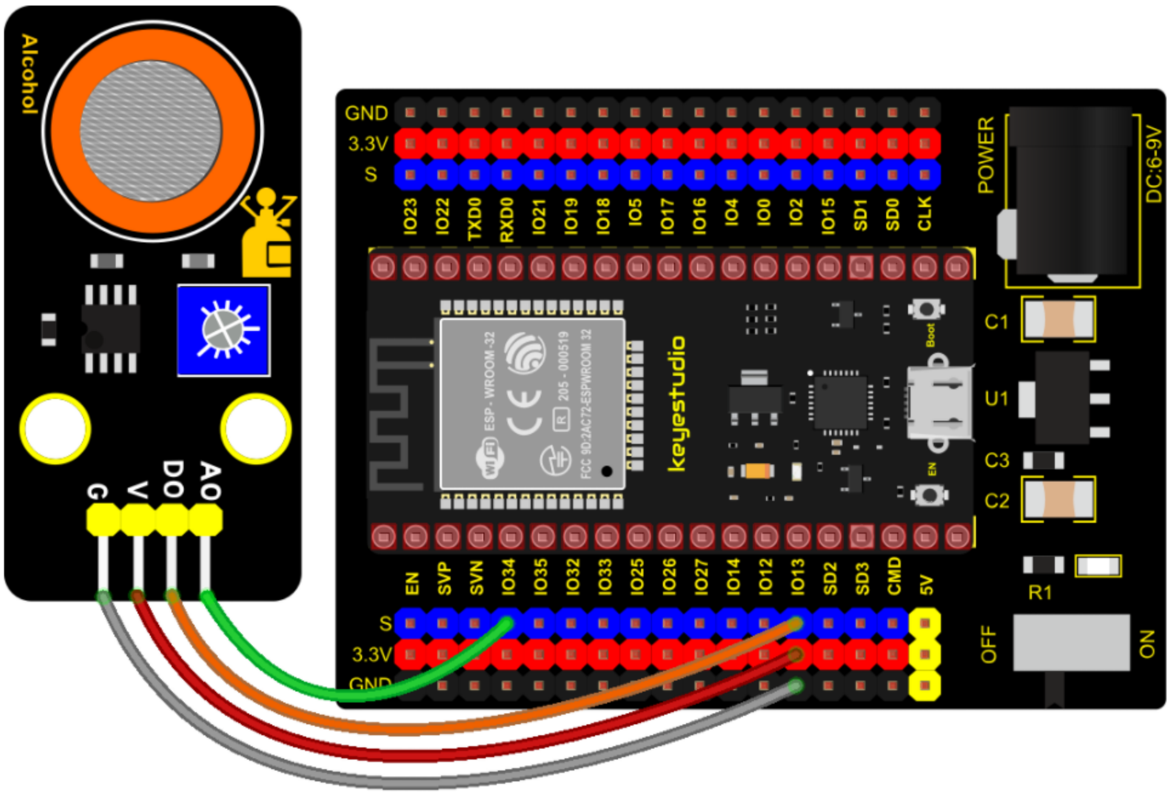
When the measured alcohol vapor content exceeds the critical point, the D0 terminal outputs a low level; when the measured alcohol vapor content does not exceed the critical point, the D0 terminal outputs a high level.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio Alcohol Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin, ADC and DAC modules.
from machine import ADC,Pin,DAC
import time

# Turn on and configure the ADC with the range of 0-3.3V
mq3_D = Pin(13, Pin.IN)
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# Read digital value and ADC value once every 0.1seconds, convert ADC value to DAC value and Voltage value and output it,
# and print these data to “Shell”.

while True:
    digitalVal = mq3_D.value()
    adcVal=adc.read()
    dacVal=adcVal//16
    voltage = adcVal / 4095.0 * 3.3
    print("digitalVal:",digitalVal,"ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V", end = "  ")
    if digitalVal == 0:
        print("Exceeding")
    else:
        print("Normal")
```

(continues on next page)

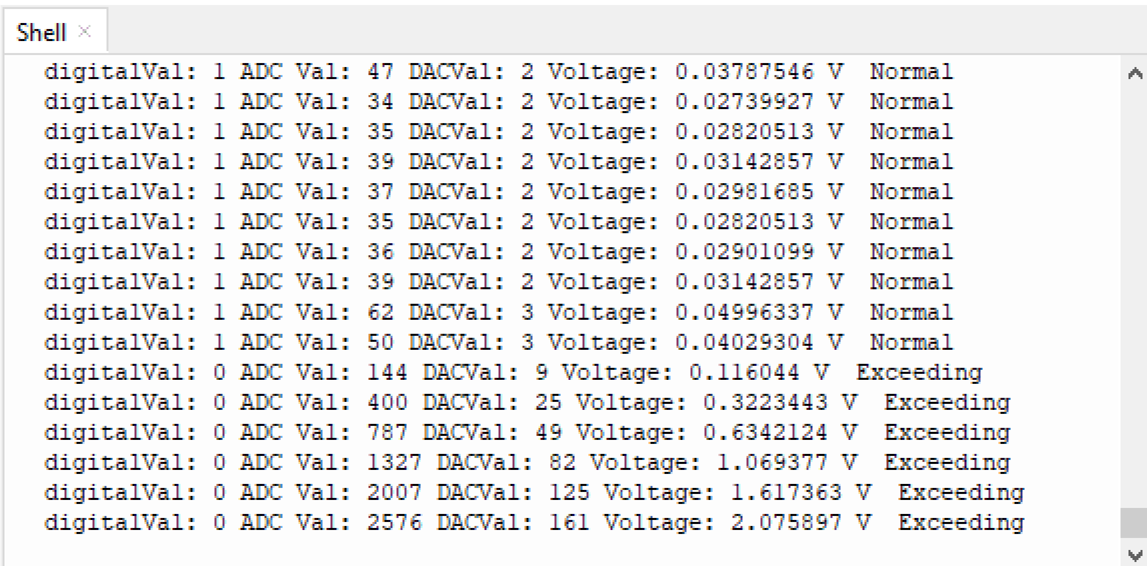
(continued from previous page)

```
    time.sleep(0.1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The "shell" window will display the corresponding data and string. After powering on, by rotating the potentiometer on the sensor, we can adjust the yellow and green LED bright and not bright critical point.

When the sensor detects the alcohol gas, the yellow and green LED lights up and the digital value in the "Shell" window changes from 1 to 0, the ADC value, DAC value and voltage value increase, as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.



5.2.29 Project 29: Five-key AD Button Module

Description

When we talked about analog and digital sensors earlier, we talked about the single-channel key module. When we press the key, it outputs a low level, and when we release the key, it outputs a high level. We can only read these two digital signals. In fact, the key module ADC acquisition can also be performed. In this kit, a DIY electronic building block five-way AD button module is included.

We can judge which key is pressed through the analog value. In the experiment, we print out the key press information in the shell.

Working Principle

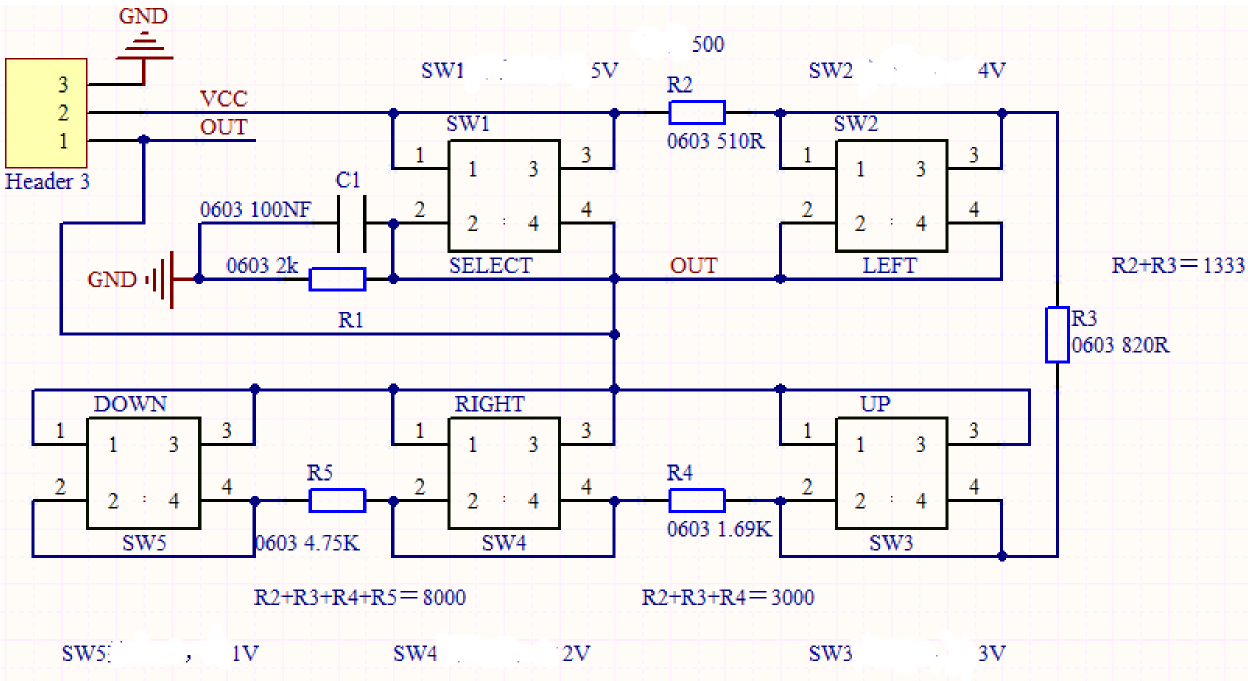
Let's look at the schematic diagram, when we do not press the key, the OUT of S output to the signal end is pulled down by R1. At this time, we read the low level 0V. When we press the key SW1, the OUT of the output to the signal end S is directly connected to the VCC. At this time, we read the high level 3.3V(the figure is marked as a 12-bit ADC(0~4095) and VCC is 5V. The principle is the same. Here we have VCC of 3.3V and ADC mapped to 12 bits), which is an analog value of 4095.

Next,when we press the key SW2, the OUT terminal voltage of the signal we read is the voltage between R2 and R1, namely VCC*R1/(R2+R1), which is about 2.64V, and the analog value is about 3276.

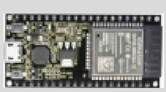
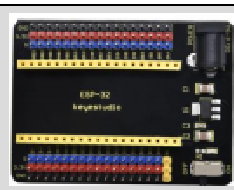
When we press the key SW3, the OUT terminal voltage of the signal we read is the voltage between R2+R3 and R1, namely VCC*R1/(R3+R2+R1), which is about 1.99V, and the analog value is about 2469.

When we press the key SW4, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4 and R1, namely VCC*R1/(R4+R3+R2+R1), about 1.31V, and the analog value is about 1626.

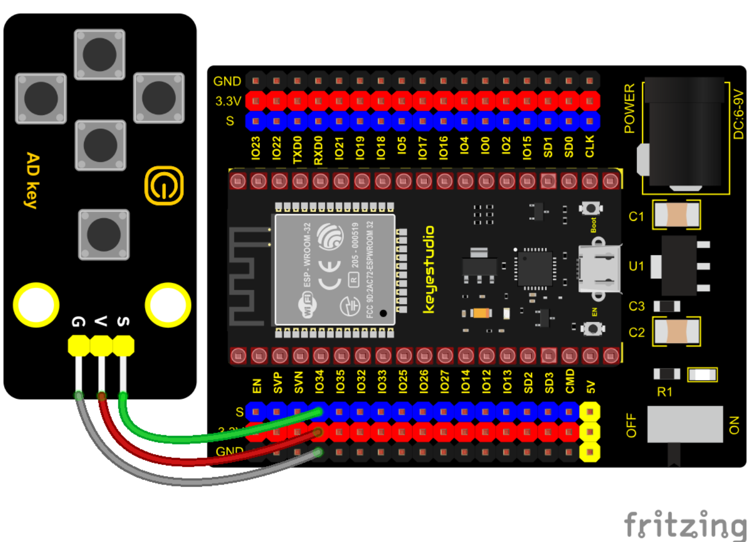
Similarly, when we press the key SW5, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4+R5 and R1, namely VCC*R1/(R5+R4+R3+R2+R1), which is about 0.68V, and the analog value is about 844.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 5-Channel AD Button Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin and ADC modules.
from machine import ADC,Pin
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

while True:
    adcvalue = adc.read()
    print(adcvalue, end = '')
    if adcvalue <= 500:
        print("  no key  is pressed")
    elif adcvalue <= 1000:
        print("  SW5 is pressed")
    elif adcvalue <= 2000:
        print("  SW4 is pressed")
    elif adcvalue <= 3000:
        print("  SW3 is pressed")
    elif adcvalue <= 4000:
        print("  SW2 is pressed")
    else:
        print("  SW1 is pressed")
    time.sleep(0.5)
```

Code Explanation

We assign the read analog value to the variable val, and the shell displays the value of val, (our default setting is 9600, which can be changed). We judge the read analog value. When the analog value is lower than 6000, we judge that the button is not pressed. When the analog value is between 6000 and 20000, we judge that the button SW5 is pressed. Between 20000 and 32000, we judge that the button SW4 is pressed.

when the analog value is between 32000 and 45000, we judge that the button SW3 is pressed. When the analog value is between 45000 and 59000, we judge that the button SW2 is pressed. Press. Otherwise, when the analog value is above 59000, we judge that the button SW1 is pressed. If we only use a fixed value, there will inevitably be errors, so we use the interval to judge.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. when the button is pressed, the shell prints out the corresponding information, as shown in the figure below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

```
Shell
1492  SW4 is pressed
0  no key  is pressed
0  no key  is pressed
0  no key  is pressed
0  no key  is pressed
2321  SW3 is pressed
0  no key  is pressed
0  no key  is pressed
0  no key  is pressed
3170  SW2 is pressed
3166  SW2 is pressed
0  no key  is pressed
0  no key  is pressed
4095  SW1 is pressed
4095  SW1 is pressed
0  no key  is pressed
```

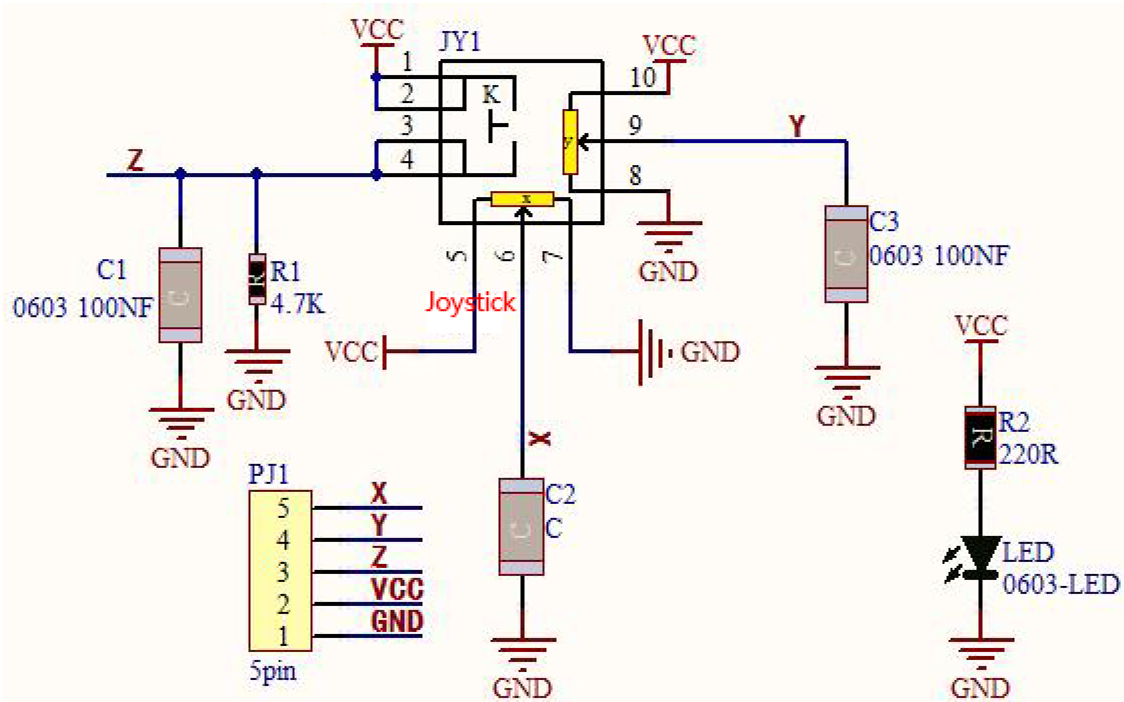
5.2.30 Project 30: Joystick Module

Overview

Game handle controllers are ubiquitous. It mainly uses PS2 joysticks. When controlling it, we need to connect the X and Y ports of the module to the analog port of the single-chip microcomputer, port B to the digital port of the single-chip microcomputer, VCC to the power output port(3.3-5V), and GND to the GND of the MCU. We can read the high and low levels of two analog values and one digital port) to determine the working status of the joystick on the module.

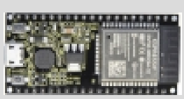
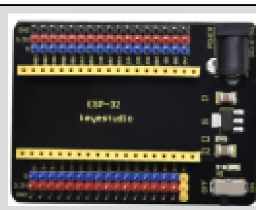
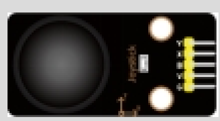
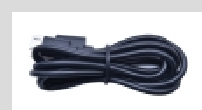
In the experiment, two analog values(x axis and y axis) will be shown on Shell.

Working Principle

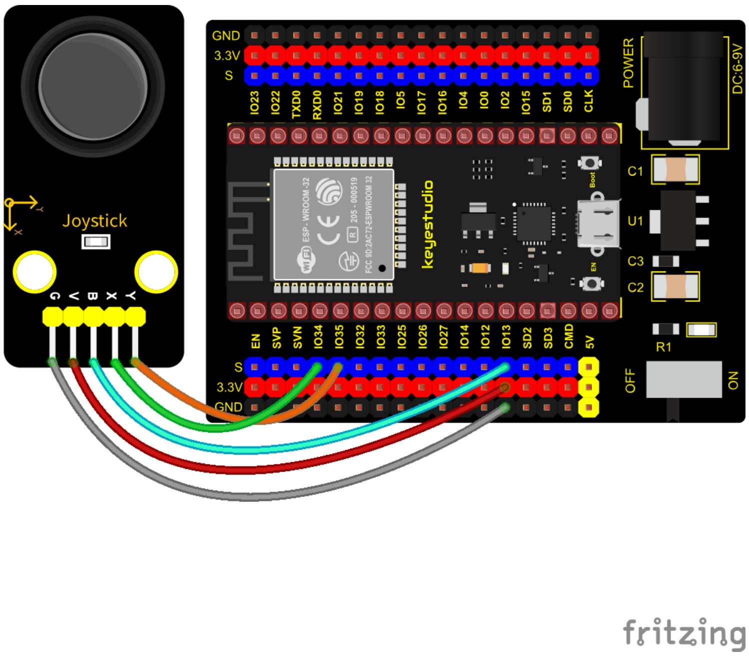


In fact, its working principle is very simple. Its inside structure is equivalent to two adjustable potentiometers and a button. When this button is not pressed and the module is pulled down by R1, low levels will be output ; on the contrary, when the button is pressed, VCC will be connected (high levels). When we move the joystick, the internal potentiometer will adjust to output different voltages, and we can read the analog value.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Joystick Module*1	5P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

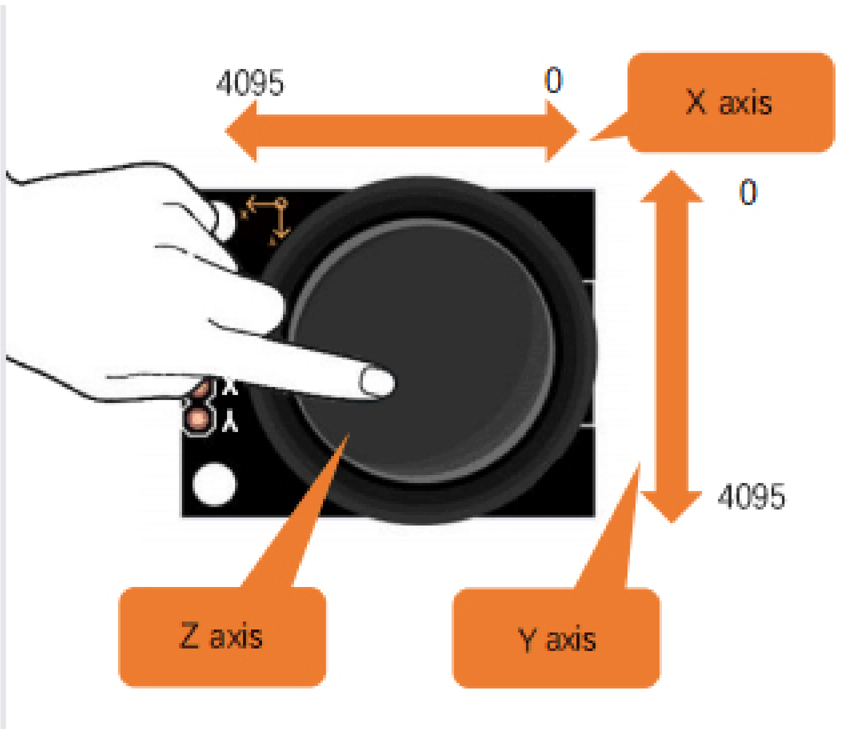
```
from machine import Pin, ADC
import time
# Initialize the joystick module (ADC function)
rocker_x=ADC(Pin(34))
rocker_y=ADC(Pin(35))
button_z=Pin(13,Pin.IN,Pin.PULL_UP)

# Set the acquisition range of voltage of the two ADC channels to 0-3.3V,
# and the acquisition width of data to 0-4095.
rocker_x.atten(ADC.ATTN_11DB)
rocker_y.atten(ADC.ATTN_11DB)
rocker_x.width(ADC.WIDTH_12BIT)
rocker_y.width(ADC.WIDTH_12BIT)

# In the code, configure Z_Pin to pull-up input mode.
# In loop(), use Read () to read the value of axes X and Y
# and use value() to read the value of axis Z, and then display them.
while True:
    print("X,Y,Z:",rocker_x.read(),",",rocker_y.read(),",",button_z.value())
    time.sleep(0.5)
```

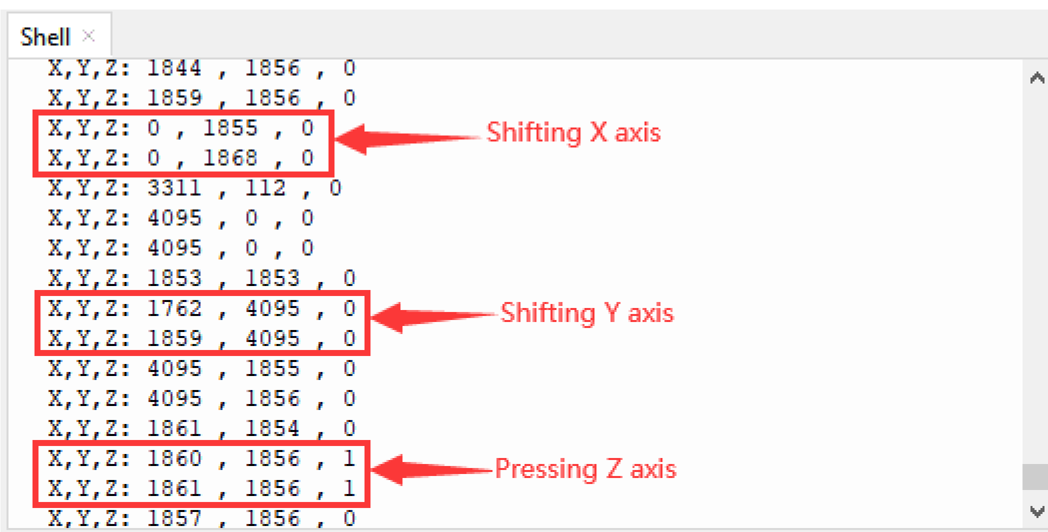
Code Explanation

In the experiment, according to the wiring diagram, the x pin is set to GPIO34, the y pin is set to GPIO35 and the pin of the joystick is set to GPIO13.

Test Result

Wire up, power on and click “Run current script”, the code starts executing. The “Shell” window will print the analog and digital values of the current joystick. Moving the joystick or pressing it will change the analog and digital values in “Shell”. Press“Ctrl+C”or click “Stop/Restart backend”to exit the program.





5.2.31 Project 31: Relay Module

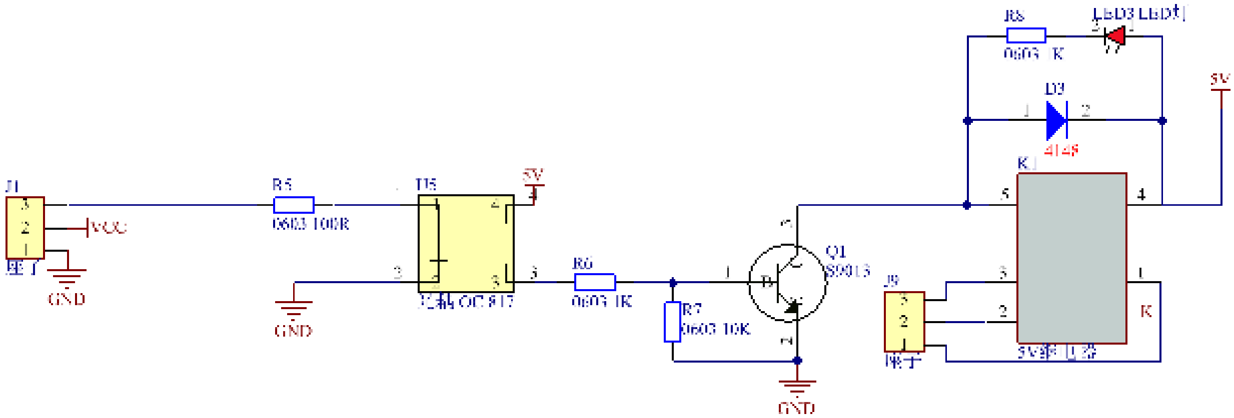
Overview

In our daily life, we usually use communication to drive electrical equipments, and sometimes we use switches to control electrical equipments. If the switch is connected directly to the ac circuit, leakage occurs and people are in danger. Therefore, from the perspective of safety, we specially designed this relay module with NO(normally open) end and NC(normally closed) end.

Working Principle

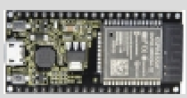
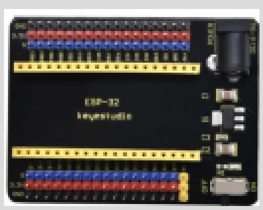
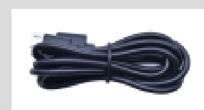
Relay is compatible with a variety of micro-controller control board, such as Arduino series micro-controller, which is a small current to control the operation of large current "automatic switch".

Input Voltage3.3V-5V

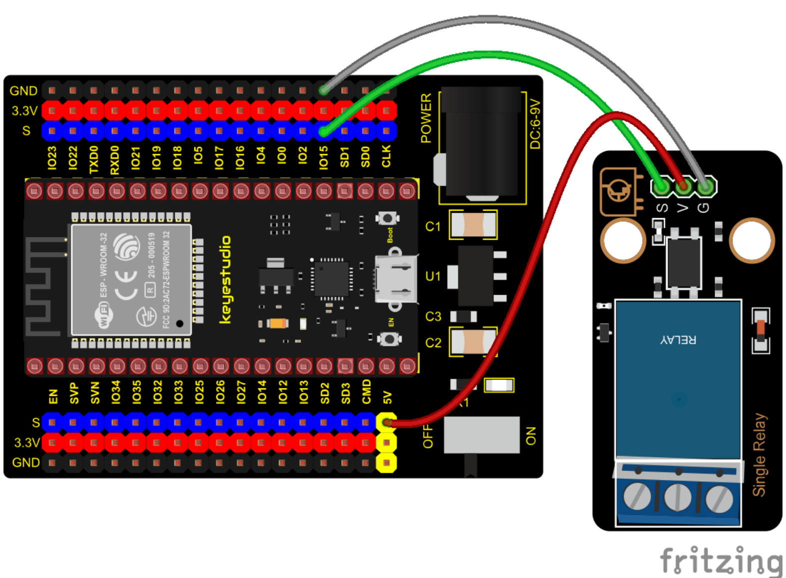


It can let the MCU control board drive 3A load, such as an LED lamp belt, a DC motor, a micro water pump and a solenoid valve plugable interface design, which is easy to use.

Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Relay Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

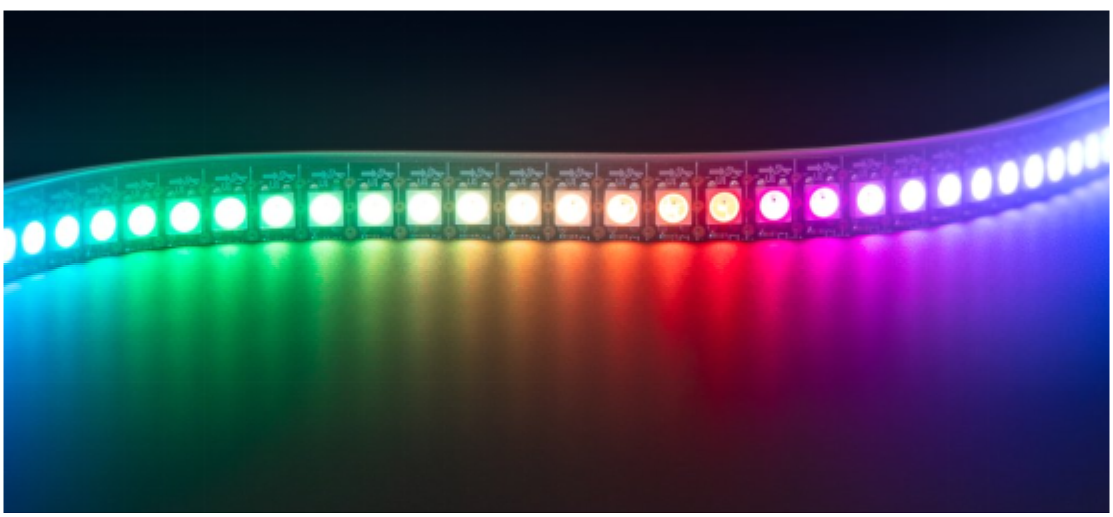
```python
from machine import Pin
import time

# create relay from Pin 15, Set Pin 15 to output
relay = Pin(15, Pin.OUT)

# The relay is opened, COM and NO are connected on the relay, and COM and NC are disconnected.
def relay_on():
    relay(1)

# The relay is closed, the COM and NO on the relay are disconnected, and the COM and NC are connected.
def relay_off():
    relay(0)

# Loop, the relay is on for one second and off for one second
while True:
    relay_on()
    time.sleep(1)
    relay_off()
    time.sleep(1)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The relay will cycle on and off, on for 1 second, off for 1 second. At the same time, you can hear the sound of the relay on and off as well as see the change of the indicator light on the relay.

Press"Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.32 Project 32: SK6812 RGB Module



Overview

In previous lessons, we learned about the plug-in RGB module and used PWM signals to color the three pins of the module.

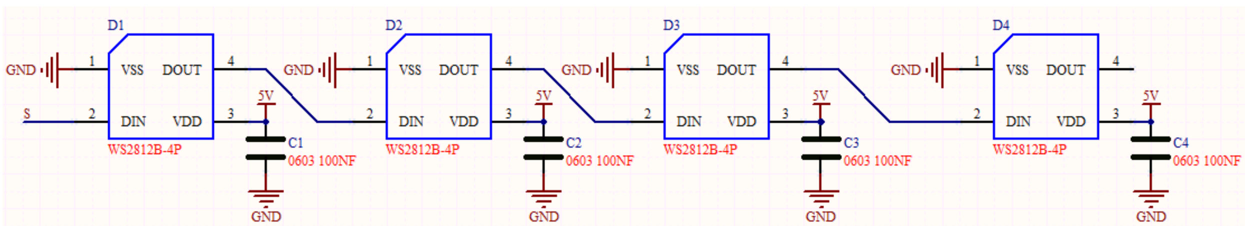
There is a Keyestudio 6812 RGB module whose the driving principle is different from the plug-in RGB module. It can only control with one pin. This is a set. It is an intelligent externally controlled LED light source with the control circuit and the light-emitting circuit. Each LED element is the same as a 5050 LED lamp bead, and each component is a pixel. There are four lamp beads on the module, which indicates four pixels.

In the experiment, we make different lights show different colors.

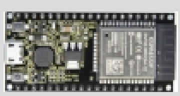
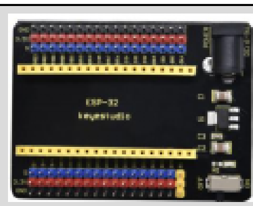
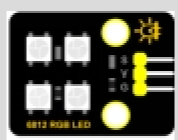
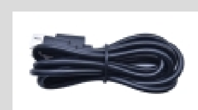
Working Principle

From the schematic diagram, we can see that these four pixel lighting beads are all connected in series. In fact, no matter how many they are, we can use a pin to control a light and let it display any color. The pixel point contains a data latch signal shaping amplifier drive circuit, a high-precision internal oscillator and a 12V high-voltage programmable constant current control part, which effectively ensures the color of the pixel point light is highly consistent.

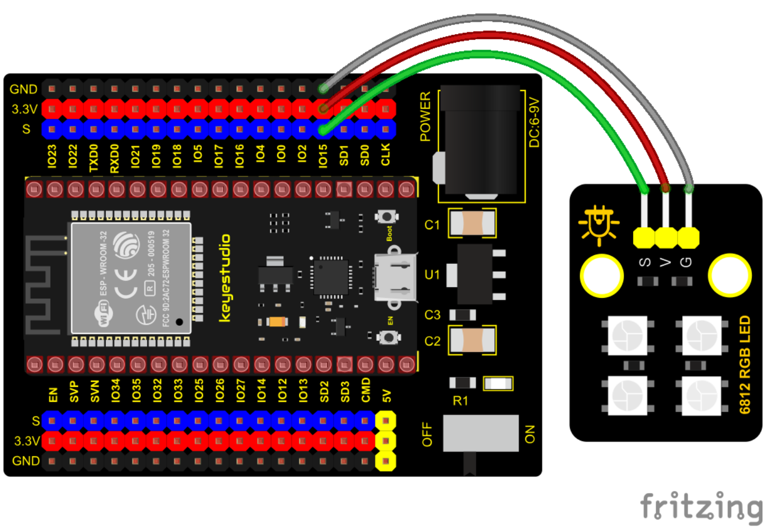
The data protocol adopts a single-wire zero-code communication method. After the pixel is powered up and reset, the S terminal receives the data transmitted from the controller. The first 24bit data sent is extracted by the first pixel and sent to the data latch of the pixel.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 6812 RGB Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```python
#Import Pin, neopiexl and time modules.
from machine import Pin
import neopixel
import time

#Define the number of pin and LEDs connected to neopixel.
pin = Pin(15, Pin.OUT)
np = neopixel.NeoPixel(pin, 4)

#brightness :0-255
brightness=100
colors=[[brightness,0,0],                    #red
        [0,brightness,0],                    #green
        [0,0,brightness],                    #blue
        [brightness,brightness,brightness],  #white
        [0,0,0]]                             #close

#Nest two for loops to make the module repeatedly display five states of red, green, blue, white and OFF.
while True:
    for i in range(0,5):
        for j in range(0,4):
            np[j]=colors[i]
            np.write()
            time.sleep_ms(50)
        time.sleep_ms(500)
    time.sleep_ms(500)
```

Code Explanation

A few function ports and functions:

np = neopixel.NeoPixel(pin, 4) , there are four LED beads, so we set to 4.

pin = Pin(15, Pin.OUT) , this is the pin number, we connect to GP15.

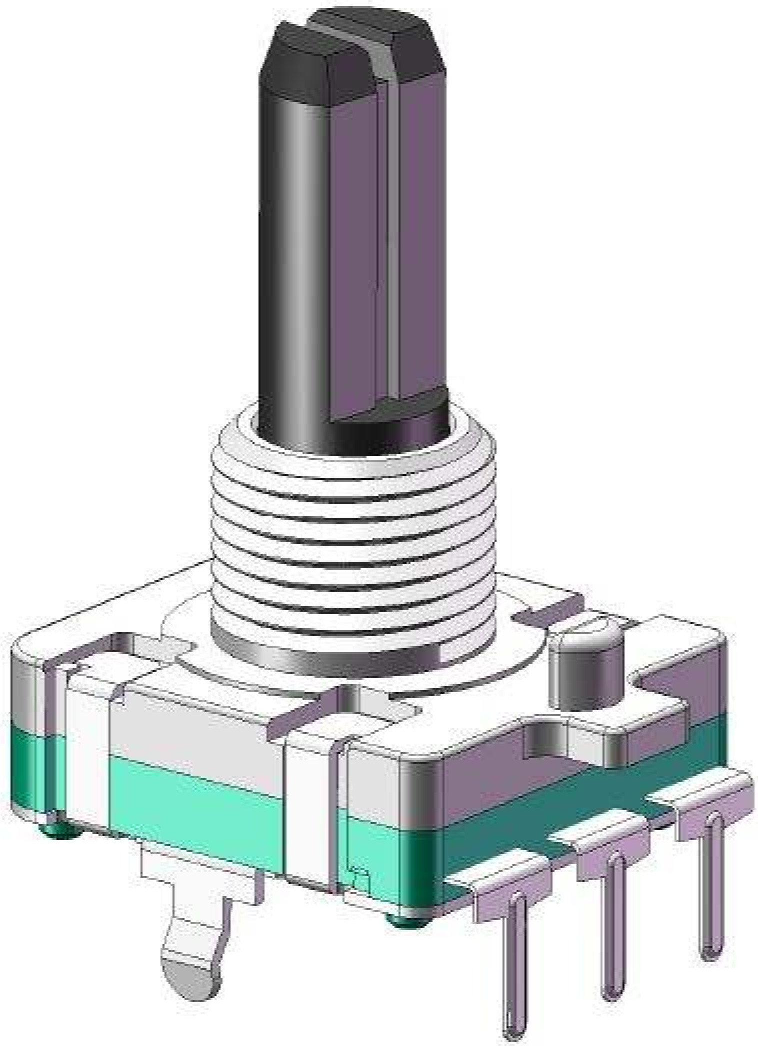
brightness = 100, brightness setting 255 implies brightest.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. Then we can see the four RGB LEDs show various colors.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.33 Project 33: Rotary Encoder



Overview

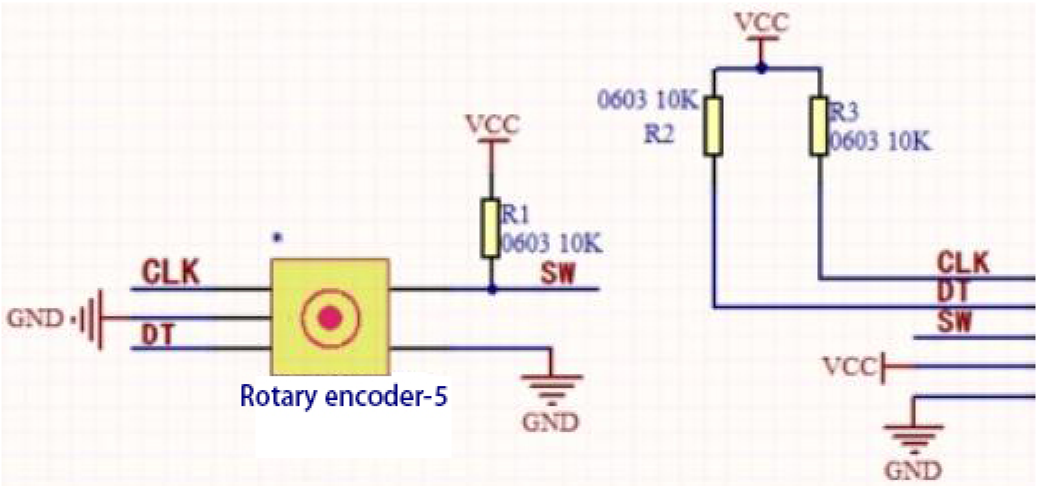
In this kit, there is a Keyestudio rotary encoder, dubbed as switch encoder. It is applied to automotive electronics, multimedia audio, instrumentation, household appliances, smart home, medical equipment and so on.

In the experiment, it it used for counting. When we rotate the rotary encoder clockwise, the set data falls by 1. If you rotate it anticlockwise, the set data is up 1, and when the middle button is pressed, the value will be show on Shell.

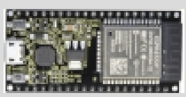
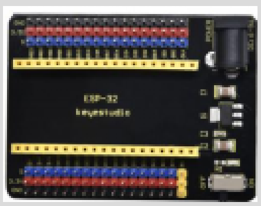
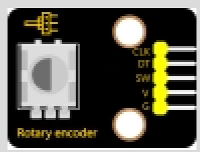
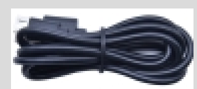
Working Principle

The incremental encoder converts the displacement into a periodic electric signal, and then converts this signal into a counting pulse, and the number of pulses indicates the size of the displacement.

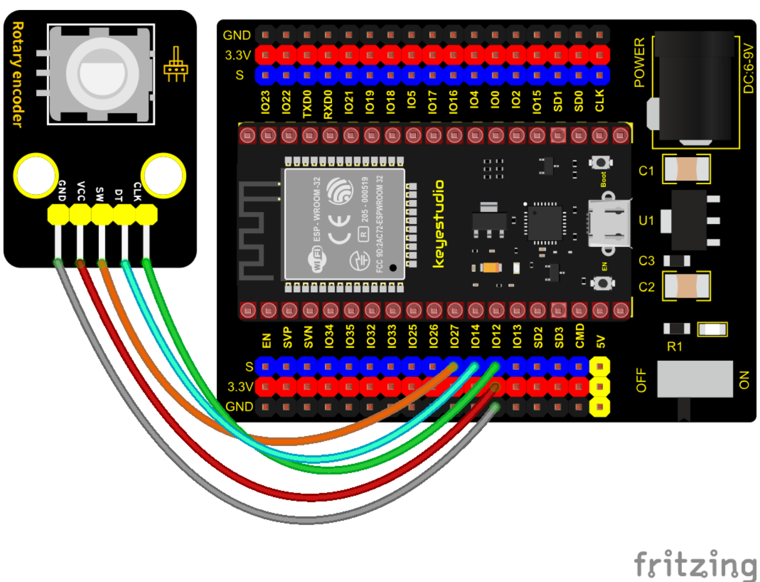
This module mainly uses 20pulse rotary encoder components. It can calculate the number of pulses output during clockwise and reverse rotation. There is no limit to count rotation. It resets to the initial state, that is, starts counting from 0.



Components

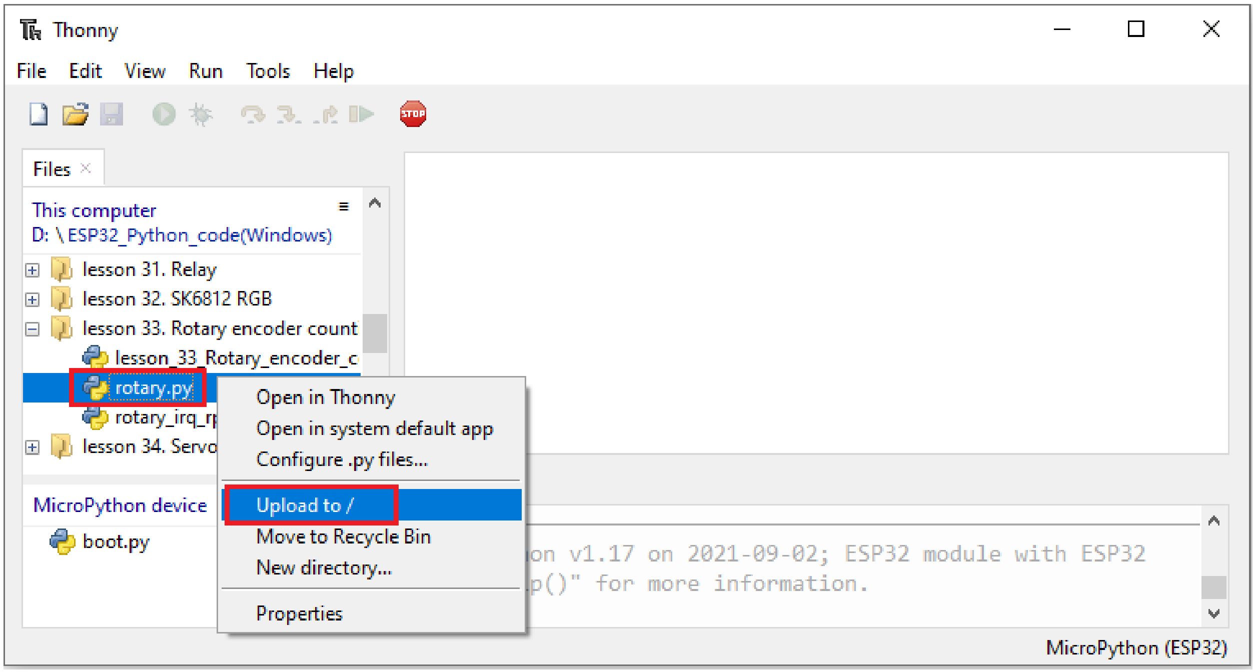
				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Encoder*1	5P Dupont Wire*1	Micro USB Cable*1

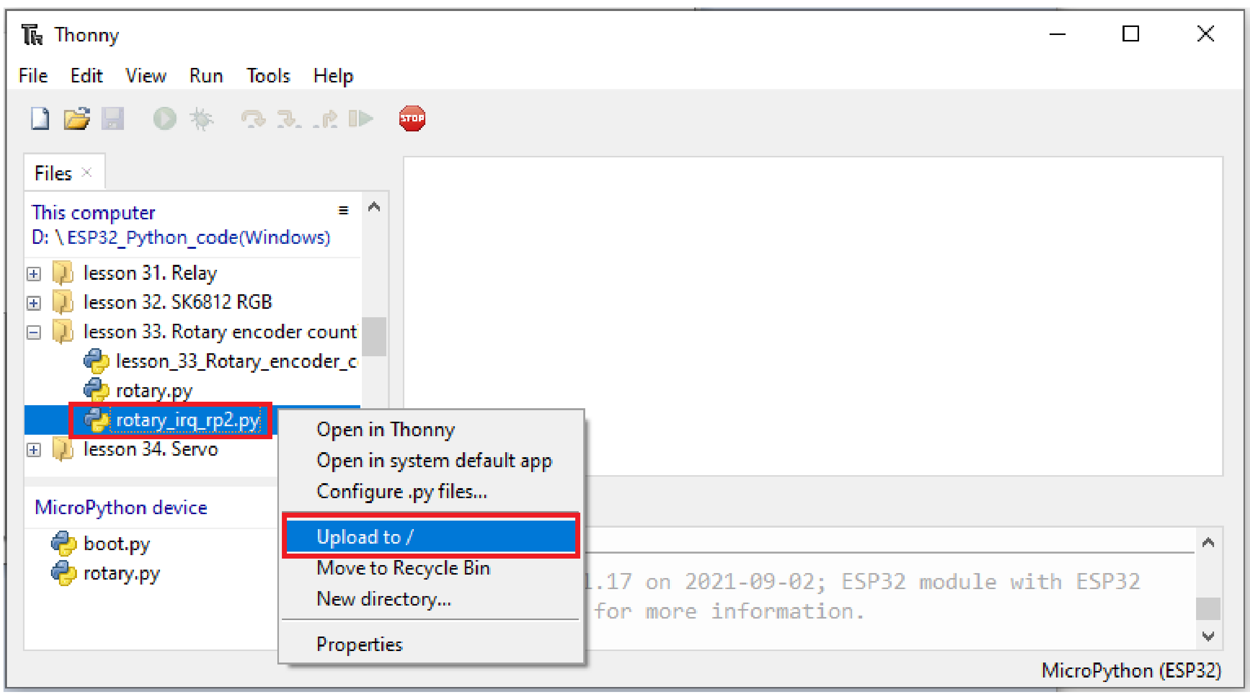
Connection Diagram



Add Library

Open“Thonny”click“This computer”→“D:”→“2. ESP32_code_MicroPython”→“lesson 30. Rotary encoder counting”. Select“rotary.py”and“rotary_irq_rp2.py”right-click“Upload to /”





Test Code

```
import time
from rotary_irq_rp2 import RotaryIRQ
from machine import Pin

SW=Pin(27,Pin.IN,Pin.PULL_UP)
r = RotaryIRQ(pin_num_clk=12,
              pin_num_dt=14,
              min_val=0,
```

(continues on next page)

(continued from previous page)

```
              reverse=False,
              range_mode=RotaryIRQ.RANGE_UNBOUNDED)
val_old = r.value()
while True:
    try:
        val_new = r.value()
        if SW.value()==0 and n==0:
            print("Button Pressed")
            print("Selected Number is : ",val_new)
            n=1
            while SW.value()==0:
                continue
        n=0
        if val_old != val_new:
            val_old = val_new
            print('result =', val_new)
        time.sleep_ms(50)
    except KeyboardInterrupt:
        break
```

Code Explanation

1). We will see the file rotary.py and rotary_irq_rp2.py. This meansthat we save them in the ESP32 successfully. Then we can use **from rotary_irq_rp2 import RotaryIRQ.**

2). **SW=Pin(20,Pin.\IN,Pin.PULL_UP)** indicates that the SW pin is connected to GPIO27, **pin_num_clk=12** indicates that the pin CLK is connected to GPIO12, and **pin_num_dt=14** means that the DT pin is connected to GPIO14. We can change these pin numbers.

3). **try/except** is the python language exception capture processing statement, **try** executes the code, **except** executes the code when an exception occurs, and when we press Ctrl+C, the program exits.

4). **r.value()** returns the value of the encoder

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. Rotate the encoder clockwise, the displayed data decrease, rotate the encoder counterclockwise, the displayed data increase.

Press the middle button of the encoder, the displayed data is the value of the encoder, as shown in the figure below.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

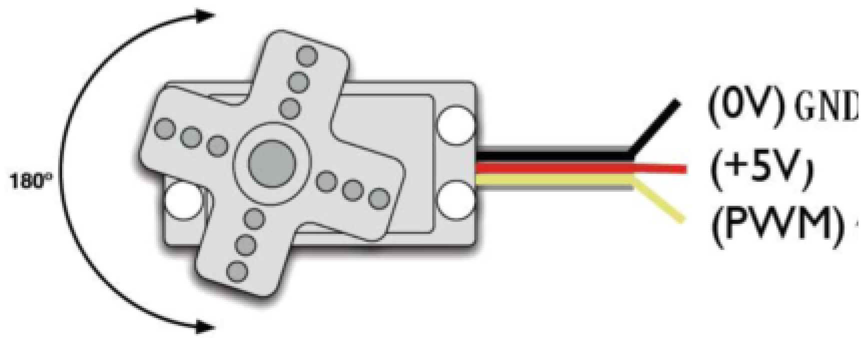
```
Shell

result = 1
result = 2
result = 3
result = 4
result = 3
result = 2
result = 1
result = 0
result = -1
result = -2
result = -3
Button Pressed
Selected Number is :  -3
Button Pressed
Selected Number is :  -3
Button Pressed
Selected Number is :  -3
```

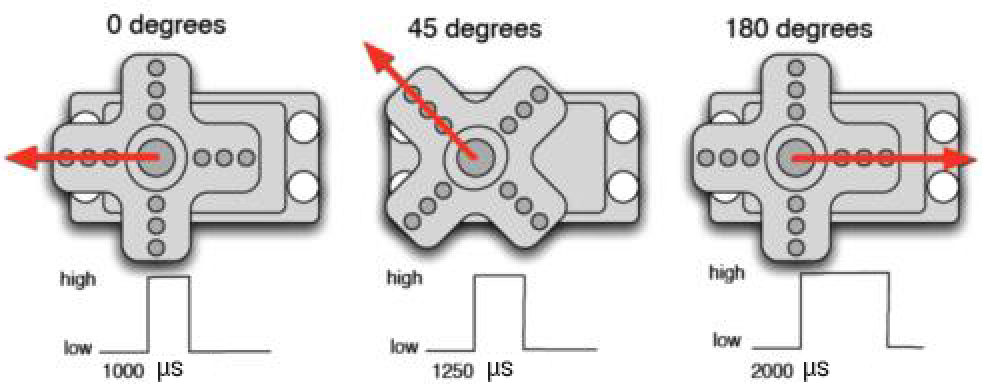
5.2.34 Project 34: Servo Control

Overview

Servo is a position control rotary actuator. It mainly consists of a housing, a circuit board, a core-less motor, a gear and a position sensor.

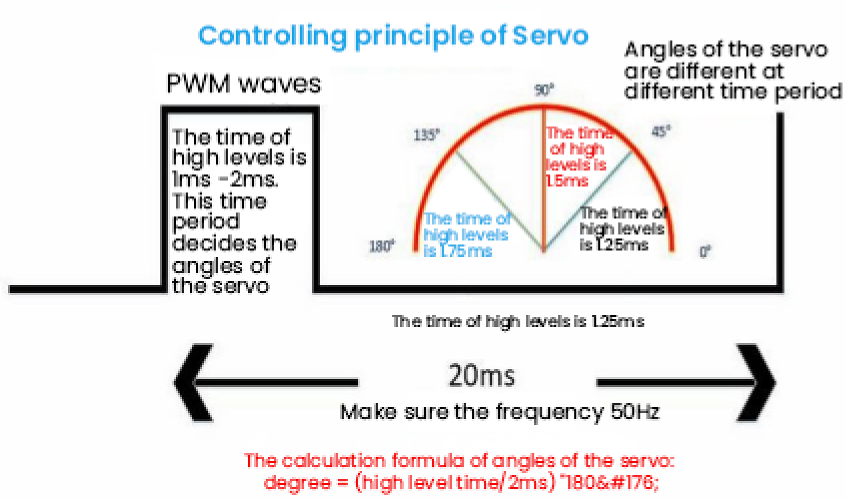
In general, servo has three lines in brown, red and orange. The brown wire is grounded, the red one is a positive pole line and the orange one is a signal line.



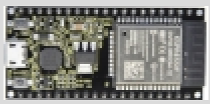
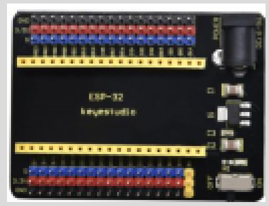
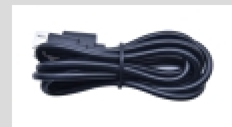


Working Principle

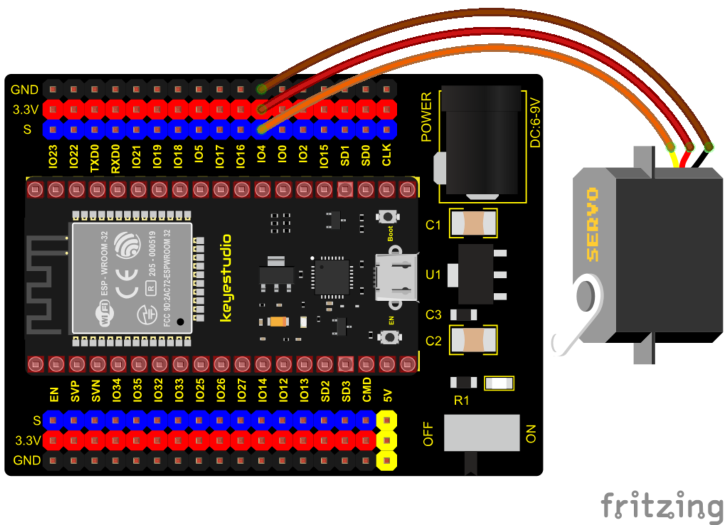
The rotation angle of servo motor is controlled by regulating the duty cycle of PWM (Pulse-Width Modulation) signal. The standard cycle of PWM signal is 20ms (50Hz). Theoretically, the width is distributed between 1ms-2ms, but in fact, it's between 0.5ms-2.5ms. The width corresponds the rotation angle from 0° to 180°. But note that for different brand motors, the same signal may have different rotation angles.



Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Servo*1	Micro USB Cable*1

Connection Diagram



Test Code 1

```
from machine import Pin, PWM
import time
pwm = PWM(Pin(4))
pwm.freq(50)

'''
Duty cycle corresponding to the Angle
0°----2.5%----25
45°----5%----51.2
90°----7.5%----77
135°----10%----102.4
180°----12.5%----128
'''
angle_0 = 25
angle_90 = 77
angle_180 = 128

while True:
    pwm.duty(angle_0)
    time.sleep(1)
    pwm.duty(angle_90)
```

(continues on next page)

(continued from previous page)

```
    time.sleep(1)
    pwm.duty(angle_180)
    time.sleep(1)
```

Code Explanation 1

According to the angle of the signal pulse width, it is converted into a duty cycle. The formula is: 2.5+angle/180*10. The PWM pin resolution of ESP32 is 2^10 = 1024. When converted to 0 degree, its duty cycle is 1024 * 2.5% = 25.6 , when the angle is 180 degrees, its duty cycle value is 1024* 12.5% = 128, these two values will be related to the program, considering the error and rotation angle, I set the duty cycle at between 10 and 150, the servo can rotate smoothly 0~180 degrees

Test Result 1

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the servo will rotate 0°90° and 180° cyclically. Press "Ctrl+C"or click "Stop/Restart backend" to exit the program.

Test Code 2

```
from utime import sleep
from machine import Pin
from machine import PWM

pwm = PWM(Pin(4))#Steering gear pin is connected to GP4.
pwm.freq(50)#20ms period, so the frequency is 50Hz
'''
Duty cycle corresponding to the Angle
0°----2.5%----25
45°----5%----51.2
90°----7.5%----77
135°----10%----102.4
180°----12.5%----128
'''
# Set the servo motor rotation Angle
def setServoCycle (position):
    pwm.duty(position)
    sleep(0.01)

# Convert the rotation Angle to duty cycle
def convert(x, i_m, i_M, o_m, o_M):
    return max(min(o_M, (x - i_m) * (o_M - o_m) // (i_M - i_m) + o_m), o_m)

while True:
    for degree in range(0, 180, 1):#servo goes from 0 to 180
        pos = convert(degree, 0, 180, 20, 150)
        setServoCycle(pos)

    for degree in range(180, 0, -1):#servo goes from 180 to 0
        pos = convert(degree, 0, 180, 20, 150)
        setServoCycle(pos)
```

Code Explanation 2

convert(x, i_m, i_M, o_m, o_M): x is the value we want to map; **i_m, i_M** are the lower and upper limits of the current value; o_m, o_M are the lower and upper limits of the target range we want to map to.

Test Result 2

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The servo rotates from 0° to 180° by moving 1° for each 15ms.

Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.35 Project 35: Ultrasonic Sensor

Bats and some marine animals are able to use high frequencies of sound for echolocation or communication. They can emit ultrasonic waves from the larynx through the mouth or nose and use the sound waves that bounce back to orient and determine the position, size and whether nearby objects are moving.

Ultrasonic is a frequency higher than 20000 Hz sound wave, which has a good direction, a strong penetration ability, and is easy to obtain more concentrated sound energy as well as spread far in the water. It can be used for ranging, speed measurement, cleaning, welding, gravel, sterilization and disinfection. What's more, it has many applications in medicine, military, industry and agriculture.

Overview

In this kit, there is a keyes HC-SR04 ultrasonic sensor, which can detect obstacles in front and the detailed distance between the sensor and the obstacle. Its principle is the same as that of bat flying. It can emit the ultrasonic signals that cannot be heard by humans. When these signals hit an obstacle and come back immediately. The distance between the sensor and the obstacle can be calculated by the time gap of emitting signals and receiving signals.

In the experiment, we use the sensor to detect the distance between the sensor and the obstacle, and print the test result.

Working Principle

The most common ultrasonic ranging method is the echo detection. As shown below; when the ultrasonic emitter emits the ultrasonic waves towards certain direction, the counter will count. The ultrasonic waves travel and reflect back once encountering the obstacle. Then the counter will stop counting when the receiver receives the ultrasonic waves coming back.

The ultrasonic wave is also sound wave, and its speed of sound V is related to temperature. Generally, it travels 340m/s in the air. According to time t, we can calculate the distance s from the emitting spot to the obstacle.

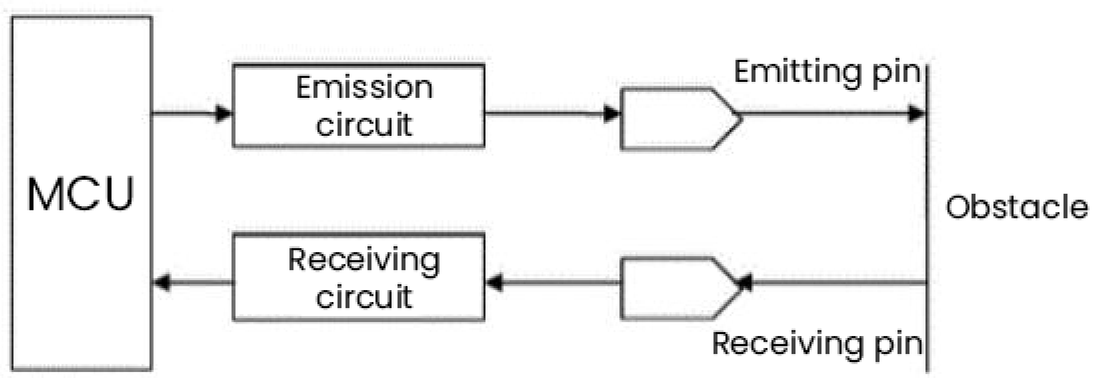
$$s = 340t/2$$

The HC-SR04 ultrasonic ranging module can provide a non-contact distance sensing function of 2cm-400cm, and the ranging accuracy can reach as high as 3mm; the module includes an ultrasonic transmitter, receiver and control circuit. Basic working principle:

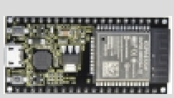
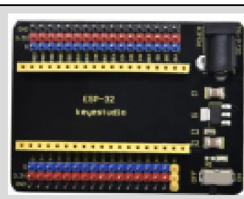
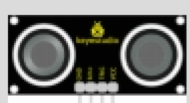
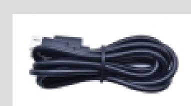
1). First pull down the TRIG, and then trigger it with at least 10us high level signal;

2). After triggering, the module will automatically transmit eight 40KHZ square waves, and automatically detect whether there is a signal to return.

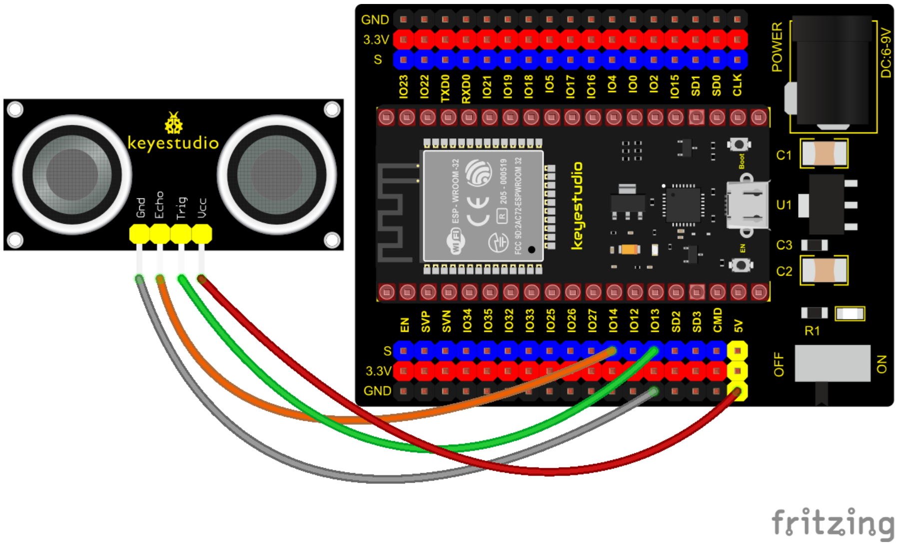
3). If there is a signal returned back, through the ECHO to output a high level, the duration time of high level is actually the time from emission to reception of ultrasonic.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio SR01 Ultrasonic Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

# Define the control pins of the ultrasonic ranging module.
Trig = Pin(13, Pin.OUT, 0)
Echo = Pin(14, Pin.IN, 0)

distance = 0 # Define the initial distance to be 0.
soundVelocity = 340 #Set the speed of sound.

# The getDistance() function is used to drive the ultrasonic module to measure distance,
# the Trig pin keeps at high level for 10us to start the ultrasonic module.
# Echo.value() is used to read the status of ultrasonic module's Echo pin,
# and then use timestamp function of the time module to calculate the duration of Echo
# pin's high level,calculate the measured distance based on time and return the value.
def getDistance():
    Trig.value(1)
    time.sleep_us(10)
    Trig.value(0)
    while not Echo.value():
        pass
    pingStart = time.ticks_us()
    while Echo.value():
        pass
    pingStop = time.ticks_us()
    pingTime = time.ticks_diff(pingStop, pingStart) // 2
    distance = int(soundVelocity * pingTime // 10000)
    return distance

# Delay for 2 seconds and wait for the ultrasonic module to stabilize,
# Print data obtained from ultrasonic module every 500 milliseconds.
time.sleep(2)
while True:
    time.sleep_ms(500)
```

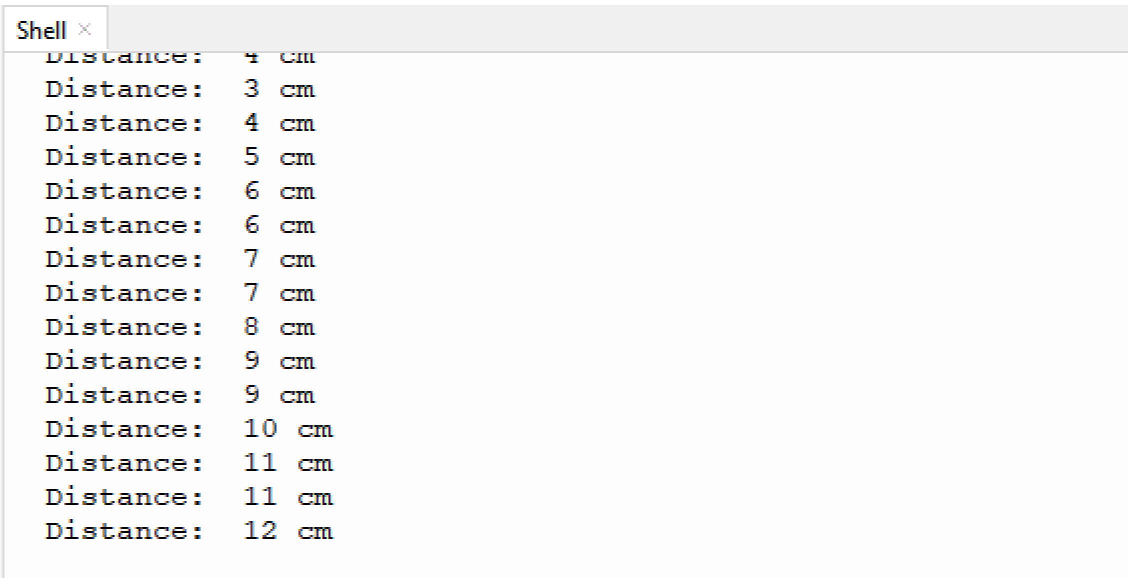
(continues on next page)

(continued from previous page)

```
    distance = getDistance()
    print("Distance: ", distance, "cm")
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The “Shell” window will print the distance between the ultrasonic sensor and the object. Press“Ctrl+C”or click “Stop/Restart backend”to exit the program.



5.2.36 Project 36: IR Receiver Module

Overview

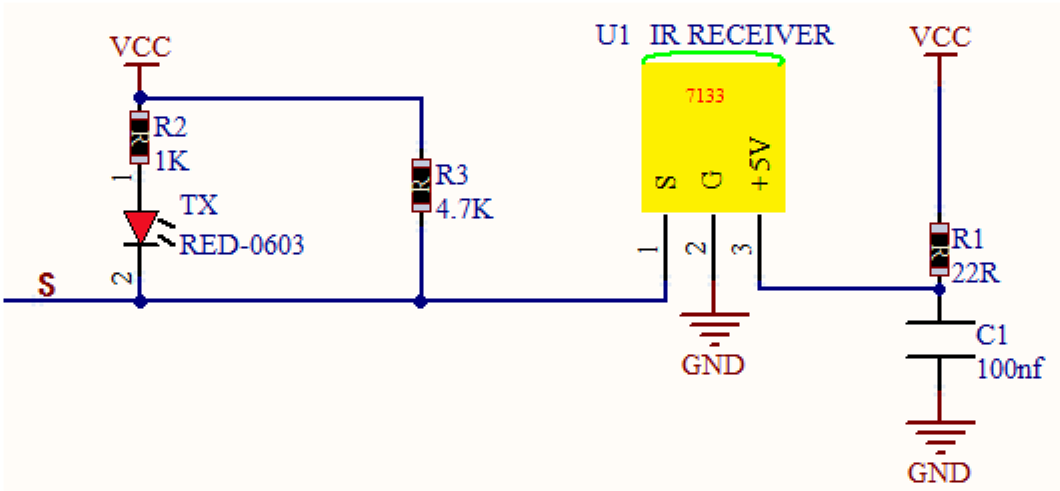
Infrared remote control is currently the most widely used means of communication and remote control, which has the characteristics of small volume, low power consumption, strong function and low cost. Therefore, recorder, audio equipment, air conditioning machine and toys and other small electrical devices have also used the infrared remote control.

Its transmitting circuit is the use of infrared light emitting diode to emit modulated infrared light wave. The circuit is composed of infrared receiving diode, triode or silicon photocell. They convert infrared light emitted by infrared emitter into corresponding electrical signal, and then send back amplifier.

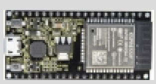
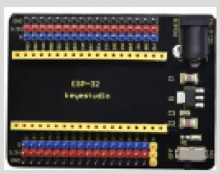
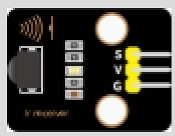
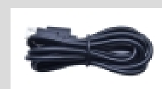
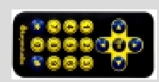
In this experiment, we need to know how to use the infrared receiving sensor, which mainly uses the VS1838B infrared receiving sensor element. It integrates receiving, amplifying, and demodulating. The internal IC has already completed the demodulation, and the output is a digital signal. It can receive 38KHz modulated remote control signal. In the experiment, we use the IR receiver to receive the infrared signal emitted by the external infrared transmitting device, and display the received signal in the shell.

Working Principle

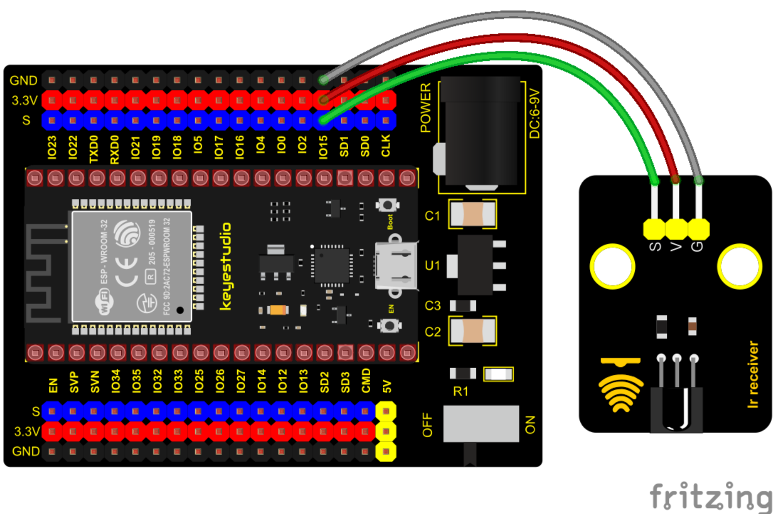
The main part of the IR remote control system is modulation, transmission and reception. The modulated carrier frequency is generally between 30khz and 60khz, and most of them use a square wave of 38kHz and a duty ratio of 1/3. A 4.7K pull-up resistor R3 is added to the signal end of the infrared receiver.



Components

					
ESP32 Board*11	ESP32 Expansion Board*1	Keyestudio DIYIR Receiver*1	3P Dupont Wire*1	Micro USB Cable*1	Remote Control*1

Connection Diagram



Test Code

```
import utime
from machine import Pin

ird = Pin(15,Pin.IN)
```

(continues on next page)

(continued from previous page)

```
act = {"1": "LLLLLLLLHHHHHHHHLHHLHLLLHLLHLHHH","2": "LLLLLLLLHHHHHHHHHLLHHLLLLHHLLHHH","3
↪": "LLLLLLLLHHHHHHHHHLHHLLLLLHLLHHHH",
       "4": "LLLLLLLLHHHHHHHHLLHHLLLLHHLLHHHH","5": "LLLLLLLLHHHHHHHHLLLHHLLLHHHLLHHH","6
↪": "LLLLLLLLHHHHHHHHLHHHHLHLHLLLLHLH",
       "7": "LLLLLLLLHHHHHHHHLLLHLLLLHHHLHHHH","8": "LLLLLLLLHHHHHHHHLLHHHLLLHHLLLHHH","9
↪": "LLLLLLLLHHHHHHHHLHLHHLHLHLHLLHLH",
       "0": "LLLLLLLLHHHHHHHHLHLLHLHLHLHHLHLH","Up": "LLLLLLLLHHHHHHHHLHHLLLHLHLLHHHLH",
↪"Down": "LLLLLLLLHHHHHHHHHLHLHLLLLHLHLHHH",
       "Left": "LLLLLLLLHHHHHHHHLLHLLLHLHHLHHHLH","Right":
↪"LLLLLLLLHHHHHHHHHHLLLLHLLLHHHHLH","Ok": "LLLLLLLLHHHHHHHHLLLLLLHLHHHHHHLH",
       "*": "LLLLLLLLHHHHHHHHLHLLLLHLHLHHHHLH","#": "LLLLLLLLHHHHHHHHLHLHLLHLHLHLHHLH"}

def read_ircode(ird):
    wait = 1
    complete = 0
    seq0 = []
    seq1 = []

    while wait == 1:
        if ird.value() == 0:
            wait = 0
    while wait == 0 and complete == 0:
        start = utime.ticks_us()
        while ird.value() == 0:
            ms1 = utime.ticks_us()
        diff = utime.ticks_diff(ms1,start)
        seq0.append(diff)
        while ird.value() == 1 and complete == 0:
            ms2 = utime.ticks_us()
            diff = utime.ticks_diff(ms2,ms1)
            if diff > 10000:
                complete = 1
        seq1.append(diff)

    code = ""
    for val in seq1:
        if val < 2000:
            if val < 700:
                code += "L"
            else:
                code += "H"
    # print(code)
    command = ""
    for k,v in act.items():
        if code == v:
            command = k
    if command == "":
        command = code
    return command

while True:
    command = read_ircode(ird)
```

(continues on next page)

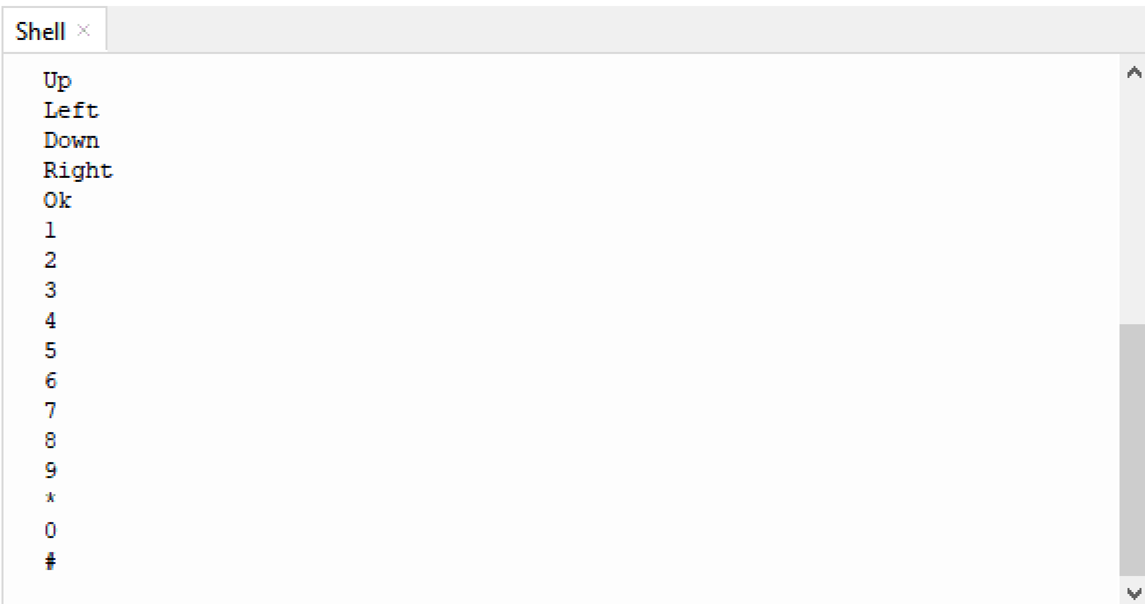
(continued from previous page)

```
    print(command)
    utime.sleep(0.5)
```

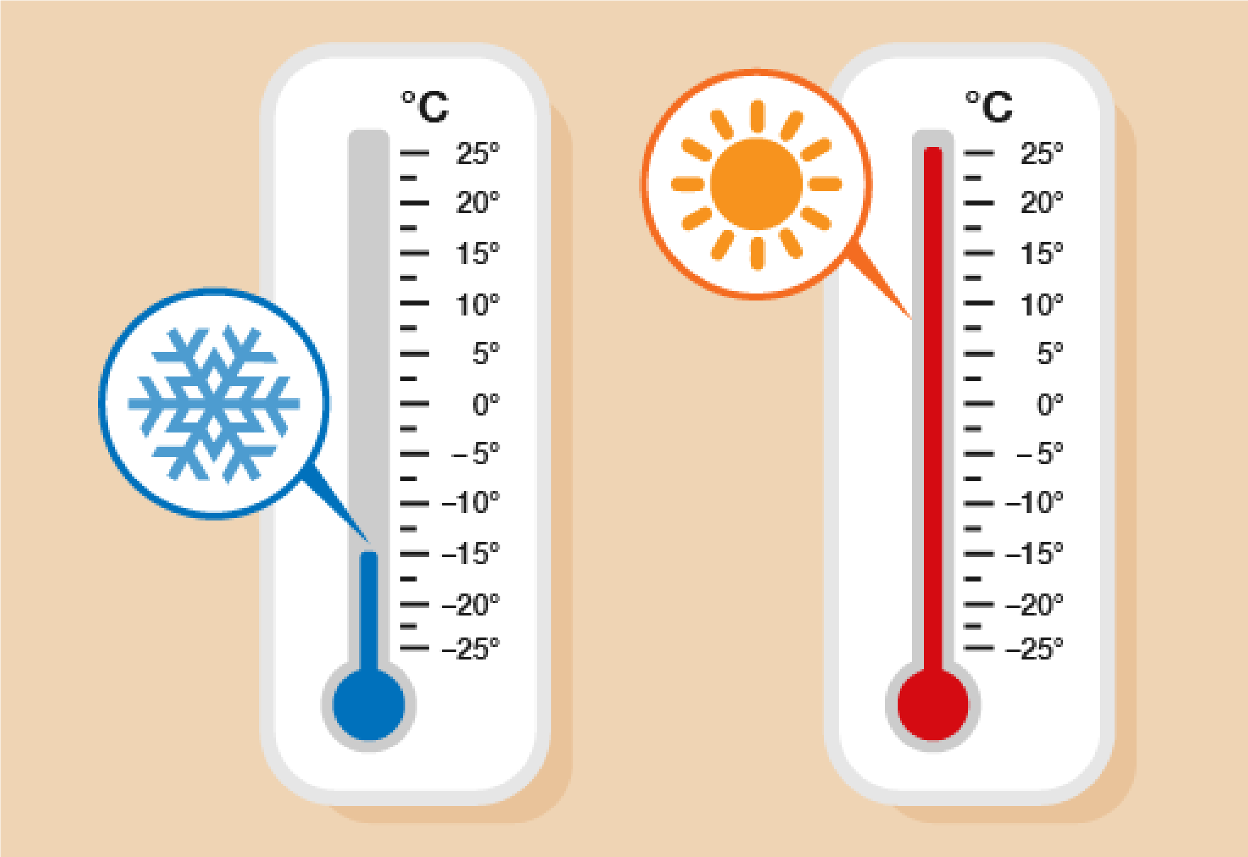
Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. Find the infrared remote control, pull out the insulating sheet, and press the button at the receiving head of the infrared receiving sensor. After receiving the signal, the LED on the infrared receiving sensor also starts to flash, as shown in the figure below.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.



5.2.37 Project 37: DS18B20 Temperature Sensor



Description

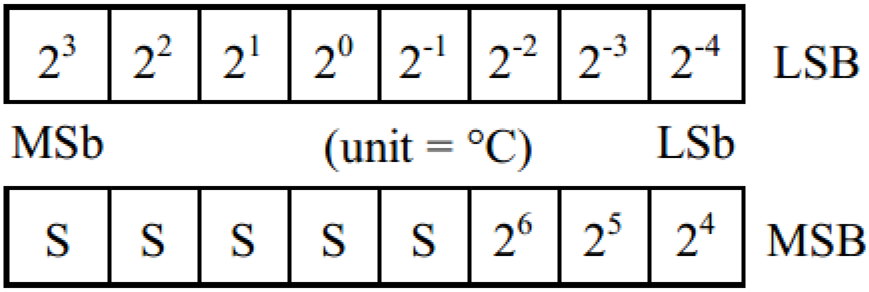
In this kit, there is a DS18B20 temperature sensor, which is from maxim. The MCU can communicate with the DS18B20 through 1-Wire protocol, and finally read the temperature. In this experiment, we will use this temperature sensor to measure the temperature in the current environment. The test result is °**C**, ranging from -55°**C** to +125°**C**. We will display the test result on shell.

Working Principle

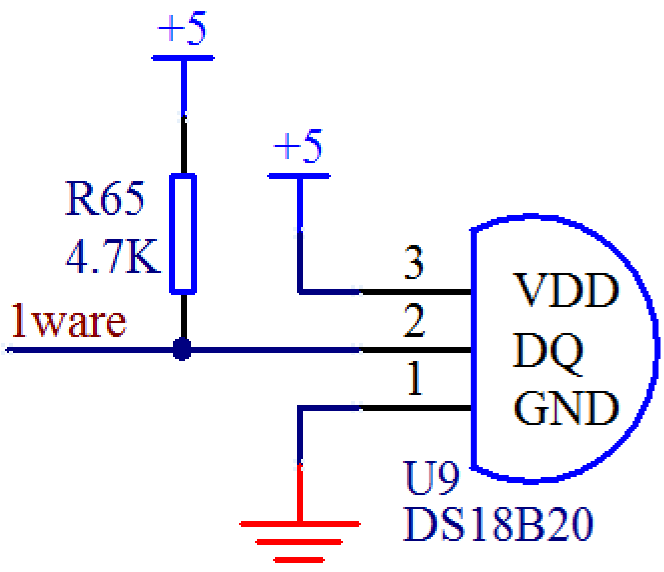
The hardware interface of the 1-Wire bus is very simple, just connect the data pin of the DS18B20 to an IO port of the microcontroller. The timing of the 1-Wire bus is relatively complex. Many students can't understand the timing diagram independently here. We have encapsulated the complex timing operations in the library, and you can use the library functions directly.

Schematic Diagram of DS18B20

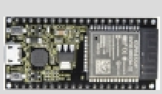
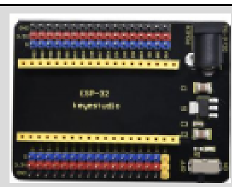
This can save up to 12-bit temperature vale. In the register, save in code complement. As shown below;



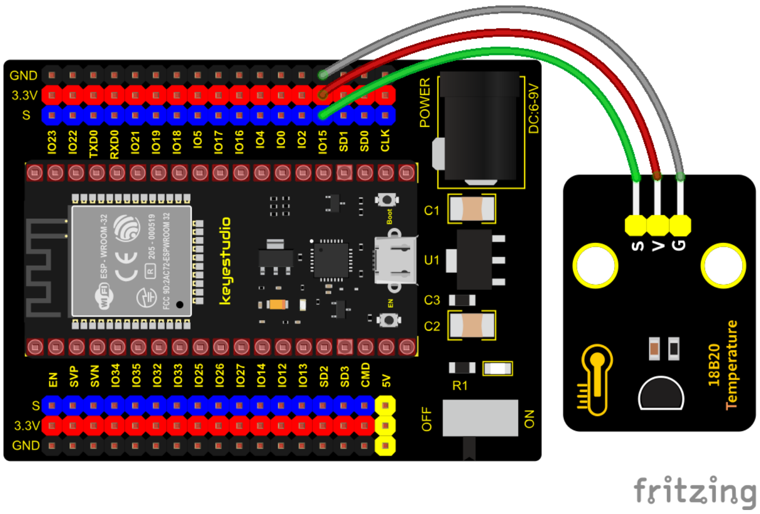
A total of 2 bytes, LSB is the low byte, MSB is the high byte, where MSb is the high byte of the byte, LSb is the low byte of the byte. As you can see, the binary number, the meaning of the temperature represented by each bit, is expressed. Among them, S represents the sign bit, and the lower 11 bits are all powers of 2, which are used to represent the final temperature. The temperature measurement range of DS18B20 is from -55 degrees to +125 degrees, and the expression form of temperature data, S represents positive and negative temperature, and the resolution is 2, which is 0.0625.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY 18B20 Temperature Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

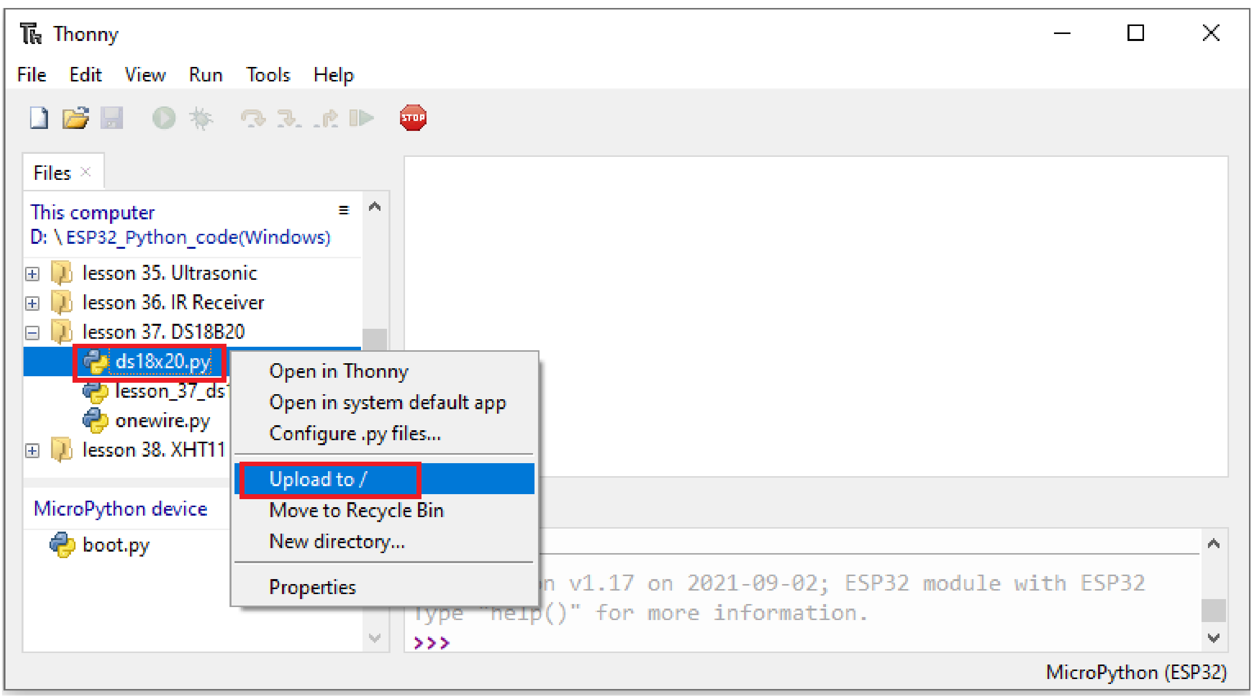
Required Components

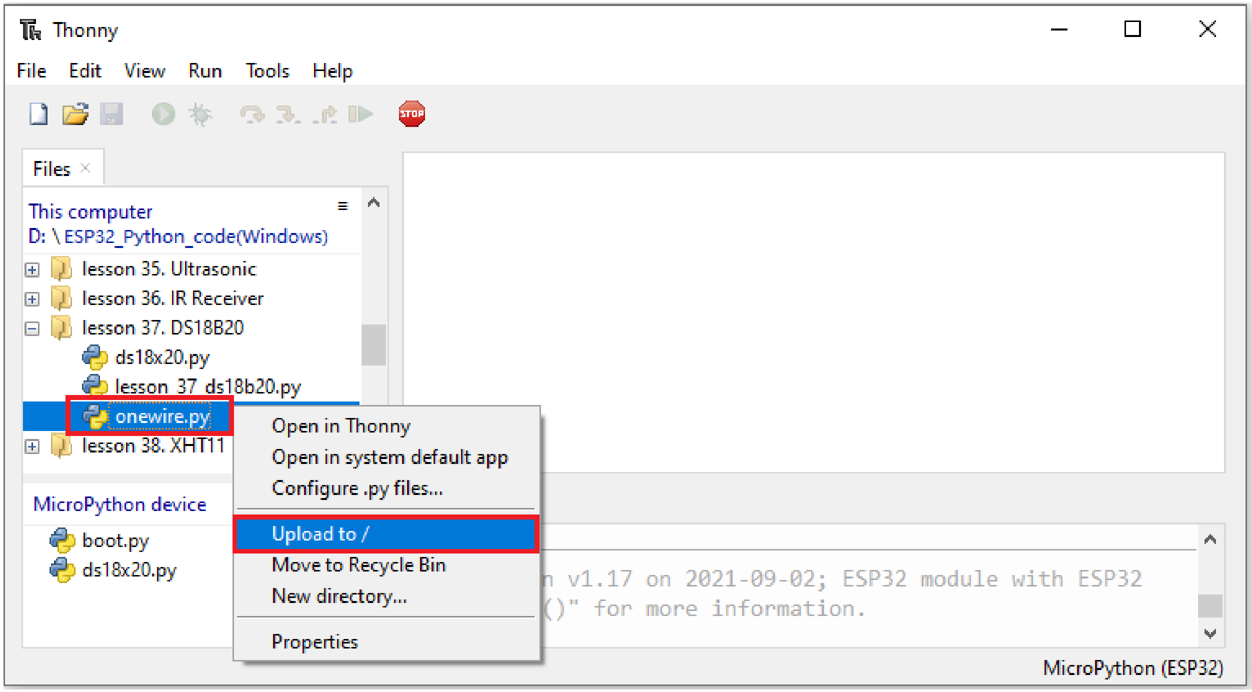


Add Library

Open"Thonny", click"This computer"→"D:"→"2. ESP32_code_MicroPython"→"lesson 37. DS18B20".

Select"ds18x20.py"and"onewire.py"right-click and select"Upload to"waiting for the "ds18x20.py" and "onewire.py"to be uploaded to the ESP32.





Test Code

```
import machine, onewire, ds18x20, time

ds_pin = machine.Pin(15)

ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))

roms = ds_sensor.scan()

print('Found DS devices: ', roms)

while True:

  ds_sensor.convert_temp()

  time.sleep_ms(750)

  for rom in roms:

    #print(rom)

    print(ds_sensor.read_temp(rom))

  time.sleep(1)
```

Code Explanation

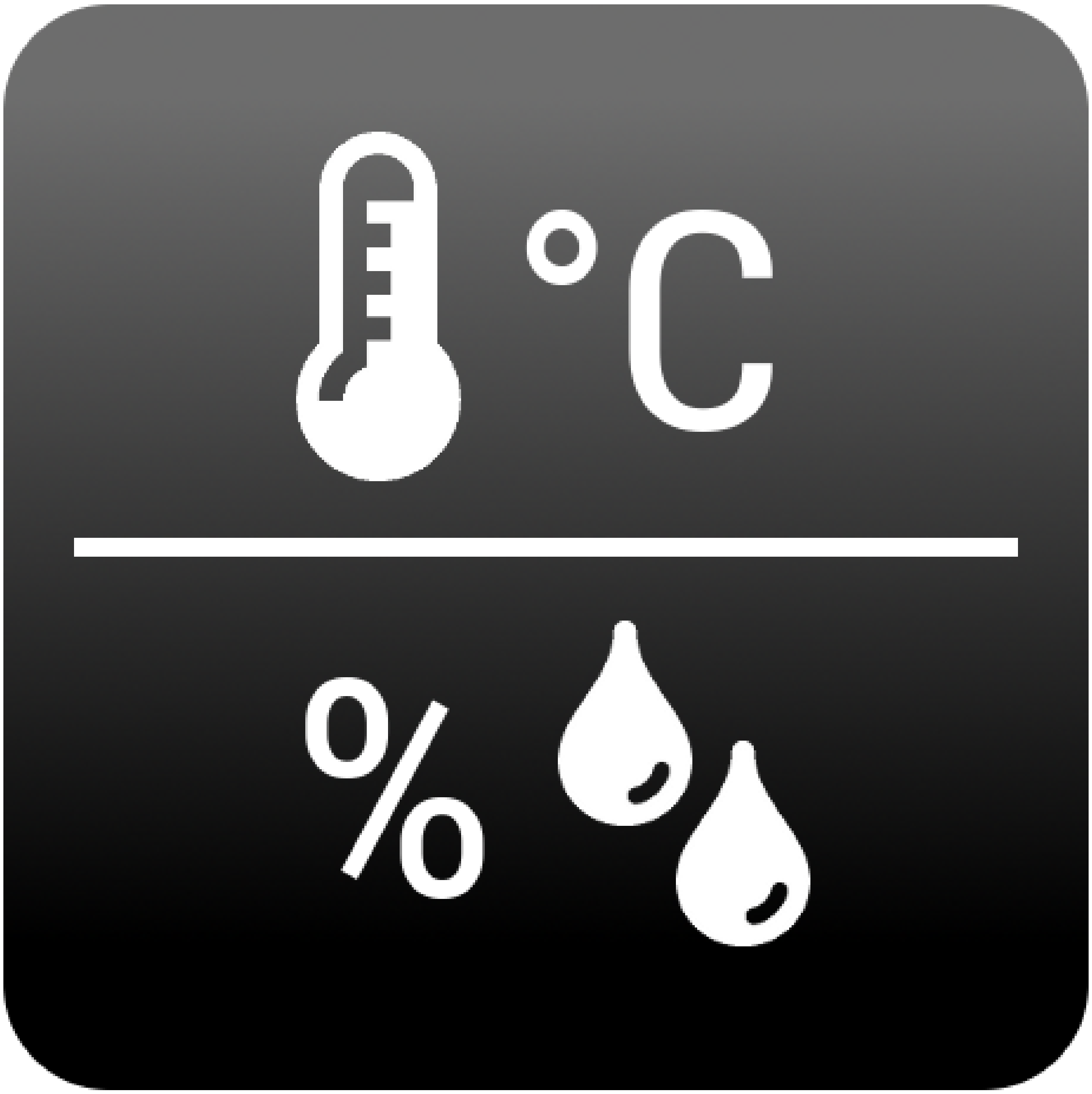
1). We set the pin to GPIO15 and obtain the temperature in the unit of ℃.

2). The Shell window displays the temperature value. Ds_sensor. Read_temp (ROM) indicates the temperature value.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the shell displays the temperature of the current environment, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

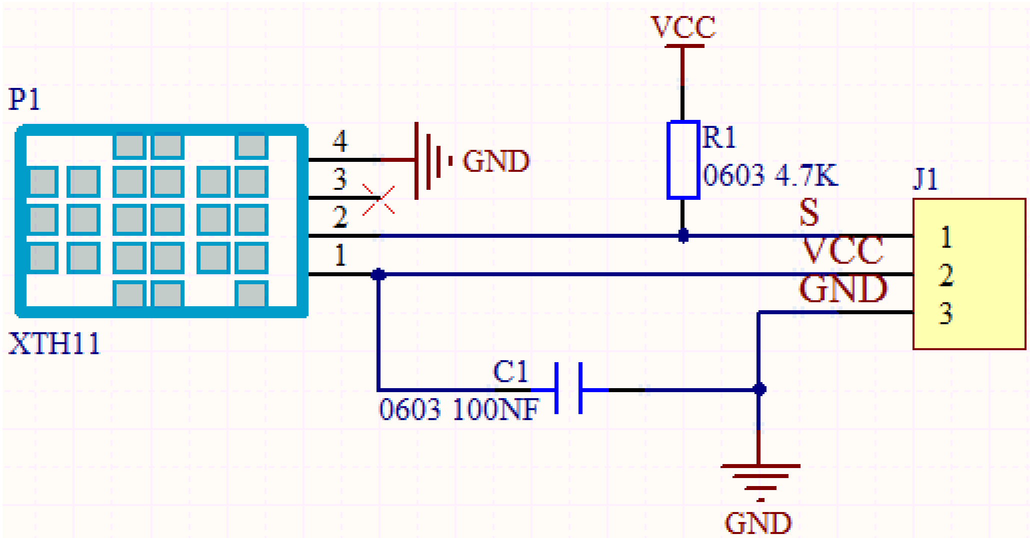
5.2.38 Project 38: XHT11 Temperature and Humidity Sensor



Description

This DHT11 temperature and humidity sensor is a composite sensor which contains a calibrated digital signal output of the temperature and humidity.

DHT11 temperature and humidity sensor uses the acquisition technology of the digital module and temperature and humidity sensing technology, ensuring high reliability and excellent long-term stability. It includes a resistive element and a NTC temperature measuring device.



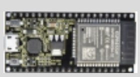
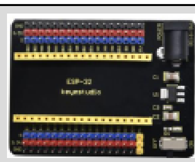
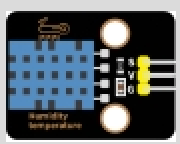
Working Principle

The communication and synchronization between the single-chip microcomputer and XHT11 adopts the single bus data format. The communication time is about 4ms. The data is divided into fractional part and integer part.

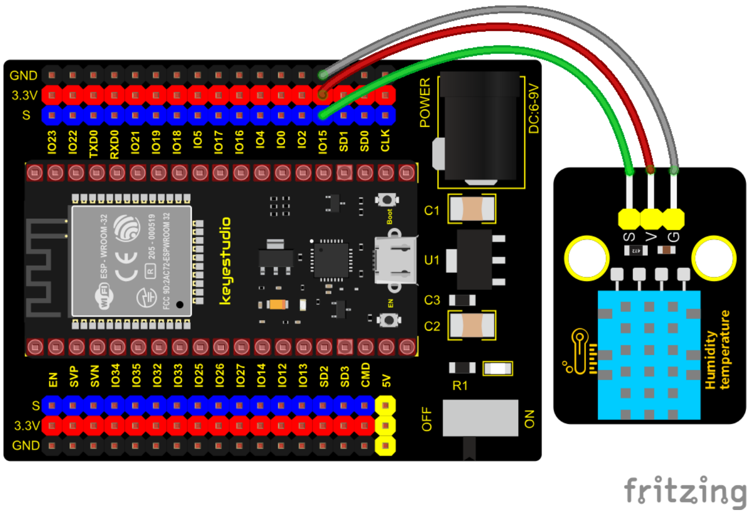
Operation process: A complete data transmission is 40bit, high bit first out. Data format: 8bit humidity integer data + 8bit humidity decimal data + 8bit temperature integer data + 8bit temperature decimal data + 8bit checksum

8-bit checksum: 8-bit humidity integer data + 8-bit humidity decimal data + 8-bit temperature integer data + 8-bit temperature decimal data "Add the last 8 bits of the result.

Required Components

				
ESP32Board*	ESP32 Expansion Board*1	XHT11 Temperature and Humidity Sensor*1compatible with DHT11)	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import machine, time and dht modules.
import machine
import time
import dht

#Associate DHT11 with Pin(15).
DHT = dht.DHT11(machine.Pin(15))

# Obtain temperature and humidity data once per second and print them out.
while True:
    DHT.measure() # Start DHT11 to measure data once.
   # Call the built-in function of DHT to obtain temperature
   # and humidity data and print them in "Shell".
    print('temperature:',DHT.temperature(),'℃','humidity:',DHT.humidity(),'%')
    time.sleep_ms(1000)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the shell displays the temperature and humidity data of the current environment, as shown below.

Press"Ctrl+C"or click "Stop/Restart backend"to exit the program.

Shell

```
temperature: 27 ℃ humidity: 65 %
temperature: 27 ℃ humidity: 66 %
temperature: 27 ℃ humidity: 65 %
temperature: 27 ℃ humidity: 65 %
temperature: 28 ℃ humidity: 69 %
temperature: 28 ℃ humidity: 68 %
temperature: 28 ℃ humidity: 76 %
temperature: 28 ℃ humidity: 74 %
temperature: 28 ℃ humidity: 78 %
temperature: 28 ℃ humidity: 84 %
temperature: 29 ℃ humidity: 85 %
temperature: 29 ℃ humidity: 88 %
temperature: 29 ℃ humidity: 88 %
temperature: 29 ℃ humidity: 88 %
temperature: 30 ℃ humidity: 91 %
```

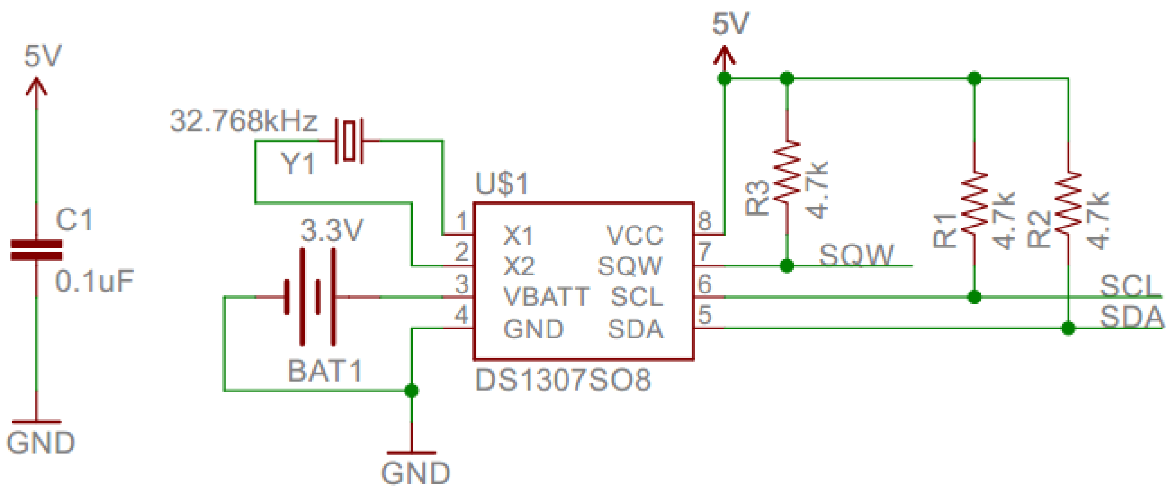
5.2.39 Project 39: DS1307 Clock Module

Overview

This module mainly uses the real-time clock chip DS1307, which is the I2C bus interface chip that has second, minute, hour, day, month, year and other functions as well as leap year automatic adjustment function introduced by DALLAS. It can work independently of CPU, and won't' affected by the CPU main crystal oscillator and capacitance as well as keep accurate time. What's more, monthly cumulative error is generally less than 10 seconds.

The chip also has a clock protection circuit in case of main power failure and runs on a back-up battery that denies the CPU read and write access. At the same time, it contains automatic switching control circuit of standby power supply, so it can guarantee the accuracy of system clock in case of power failure of main power supply and other bad environment.

Going forward, the DS1307 chip internal integration has a certain capacity, with power failure protection characteristics of static RAM, which can be used to save some key data.



In the experiment, we use the DS1307 clock module to obtain the system time and print the test results.

Working Principle

Serial real-time clock records year, month, day, hour, minute, second and week; AM and PM indicate morning and afternoon respectively; 56 bytes of NVRAM store data; 2-wire serial port; programmable square wave output; power failure detection and automatic switching circuit; battery current is less than 500nA.

Pins description

X1, X232.768kHz crystal terminal ;

VBAT: +3V input;

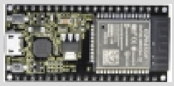
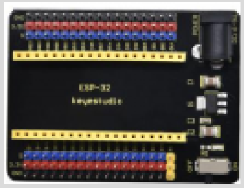
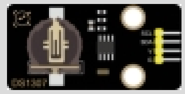
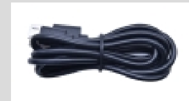
SDAserial data;

SCLserial clock;

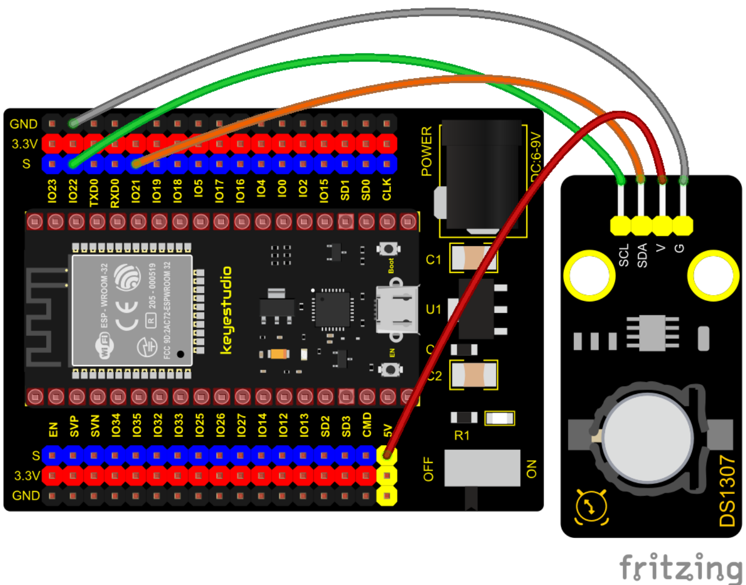
SQW/OUTsquare waves/output drivers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM
		24	PM/ AM							00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

Components

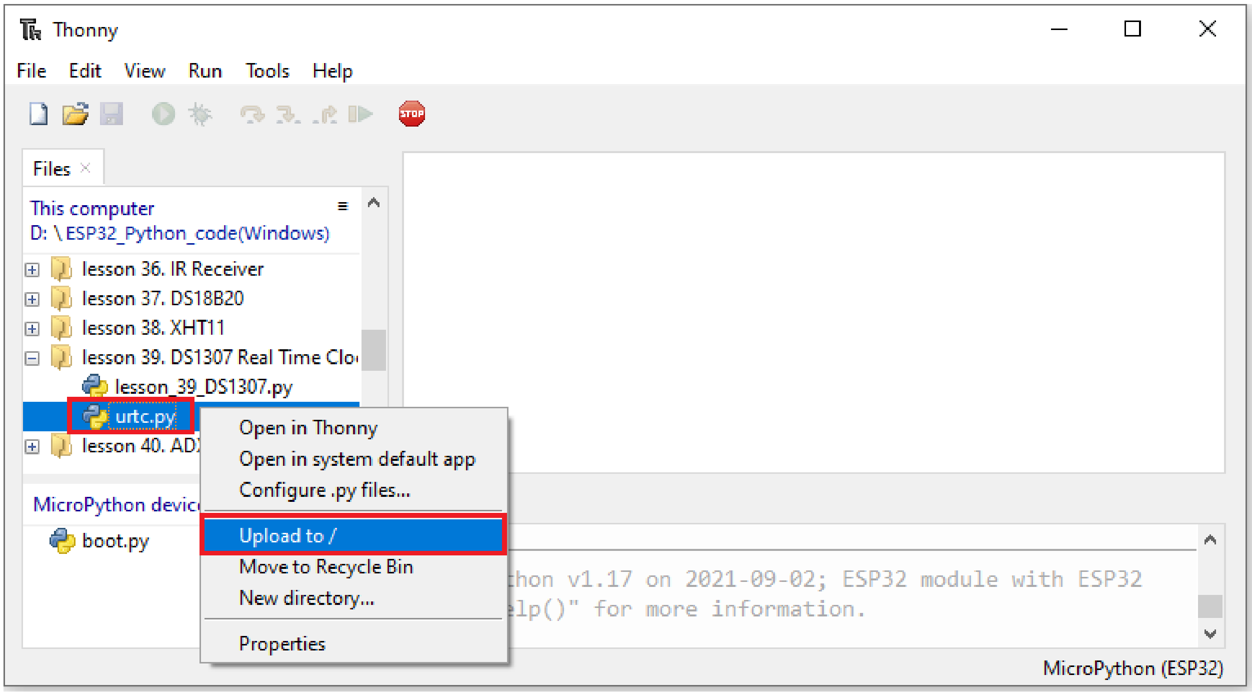
				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DS1307 Clock Module*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Add Library

Open "Thonny", click "This computer"→"D:"→"2. ESP32_code_MicroPython"→"lesson 39. DS1307 Real Time Clock". Select"urtc.py"right-click and select"Upload to /"waiting for the"urtc.py"to be uploaded to the ESP32.



Test Code

```
from machine import I2C, Pin
from urtc import DS1307
import utime

i2c = I2C(1,scl = Pin(22),sda = Pin(21),freq = 400000)
rtc = DS1307(i2c)

year = int(input("Year : "))
month = int(input("month (Jan --> 1 , Dec --> 12): "))
date = int(input("date : "))
day = int(input("day (1 --> monday , 2 --> Tuesday ... 0 --> Sunday): "))
hour = int(input("hour (24 Hour format): "))
minute = int(input("minute : "))
second = int(input("second : "))

now = (year,month,date,day,hour,minute,second,0)
rtc.datetime(now)

#(year,month,date,day,hour,minute,second,p1) = rtc.datetime()
while True:
    DateTimeTuple = rtc.datetime()
    print(DateTimeTuple[0], end = '-')
    print(DateTimeTuple[1], end = '-')
    print(DateTimeTuple[2], end = '  ')
    print(DateTimeTuple[4], end = ':')
    print(DateTimeTuple[5], end = ':')
    print(DateTimeTuple[6], end = '  week:')
    print(DateTimeTuple[3])
    utime.sleep(1)
```

Code Explanation

rtc.datetime()Return a tuple of time. When the program is running, we set the "please input" program, run the code, it will prompt us to input the time and date, after the input is completed, the data will be printed every second.

DateTimeTuple[0]: save years

DateTimeTuple[1]: save months

DateTimeTuple[2]: save days

DateTimeTuple[3]: save weeks

Rtc.GetDateTime().Month(): return months

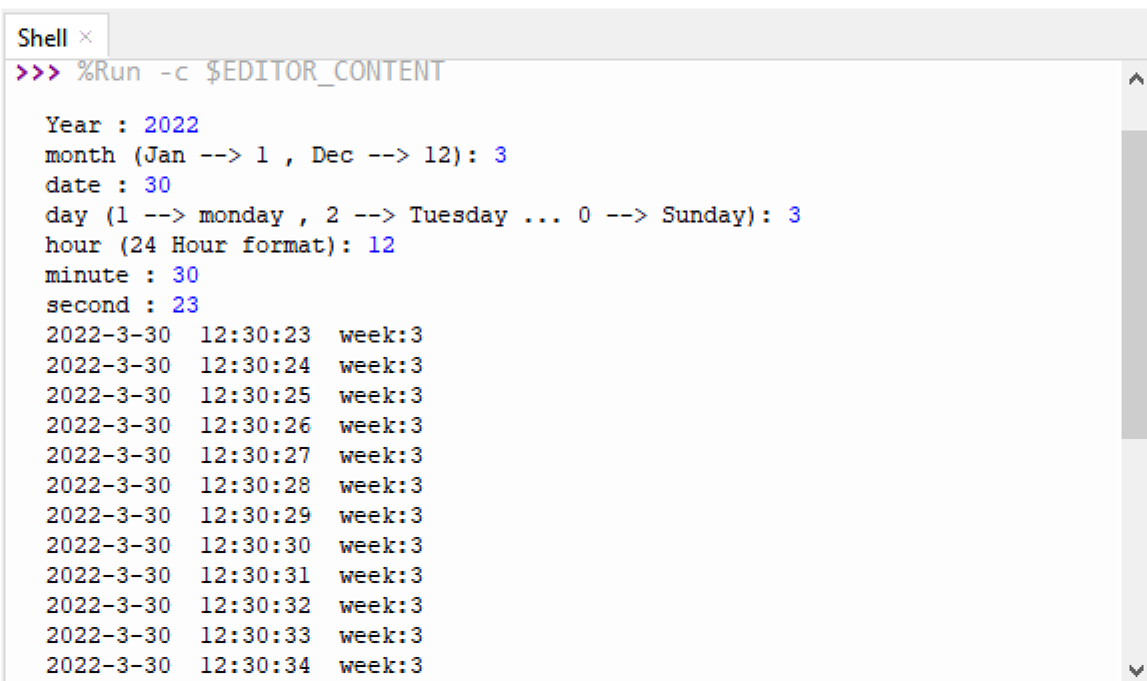
DateTimeTuple[4]: save hours

DateTimeTuple[5]: save minutes

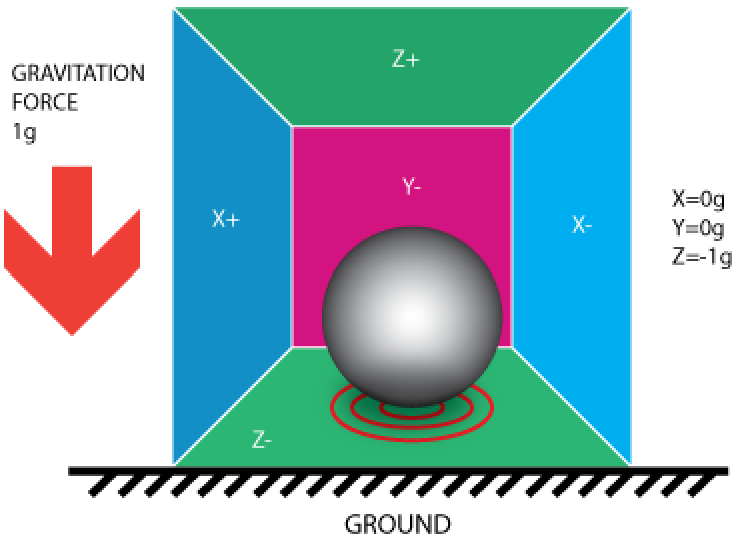
DateTimeTuple[6]: save seconds

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing, the shell will display "Year". Then we enter year, month, day, hour, minute and second, once complete, printed the data every second, as shown below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.



5.2.40 Project 40: ADXL345 Acceleration Sensor

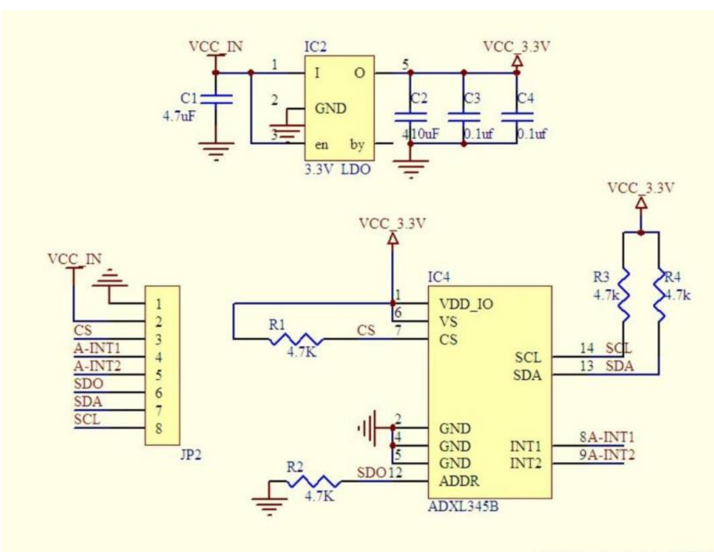


Overview

In this kit, there is a DIY electronic building block ADXL345 acceleration sensor module, which uses the ADXL345BCCZ chip. The chip is a small, thin, low-power 3-axis accelerometer with a high resolution (13 bits) and a measurement range of ±16g that can measure both dynamic acceleration due to motion or impact as well as stationary acceleration such as gravitational acceleration, making the device usable as a tilt sensor.

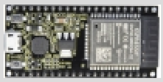
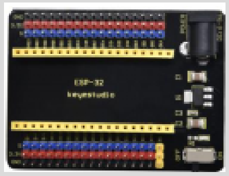
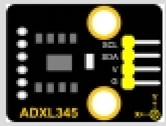
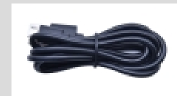
Working Principle

The ADXL345 is a complete 3-axis acceleration measurement system with a selection of measurement ranges of ±2 g, ±4 g, ±8 g or ±16 g. Its digital output data is in 16-bit binary complement format and can be accessed through an SPI (3-wire or 4-wire) or I2C digital interface.

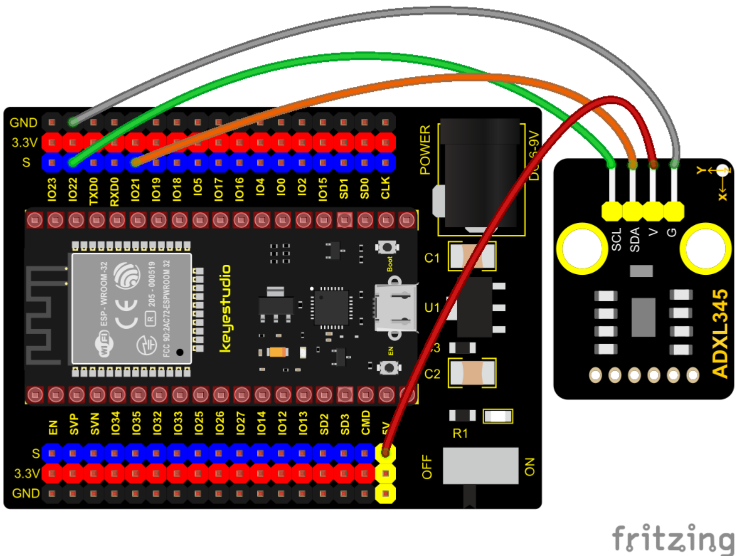


The sensor can measure static acceleration due to gravity in tilt detection applications, as well as dynamic acceleration due to motion or impact. Its high resolution (3.9mg/LSB) enables measurement of tilt Angle changes of less than 1.0°.

Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio ADXL345 Acceleration Module*1	4P Dupont Wire*1	Micro USB Cable*1

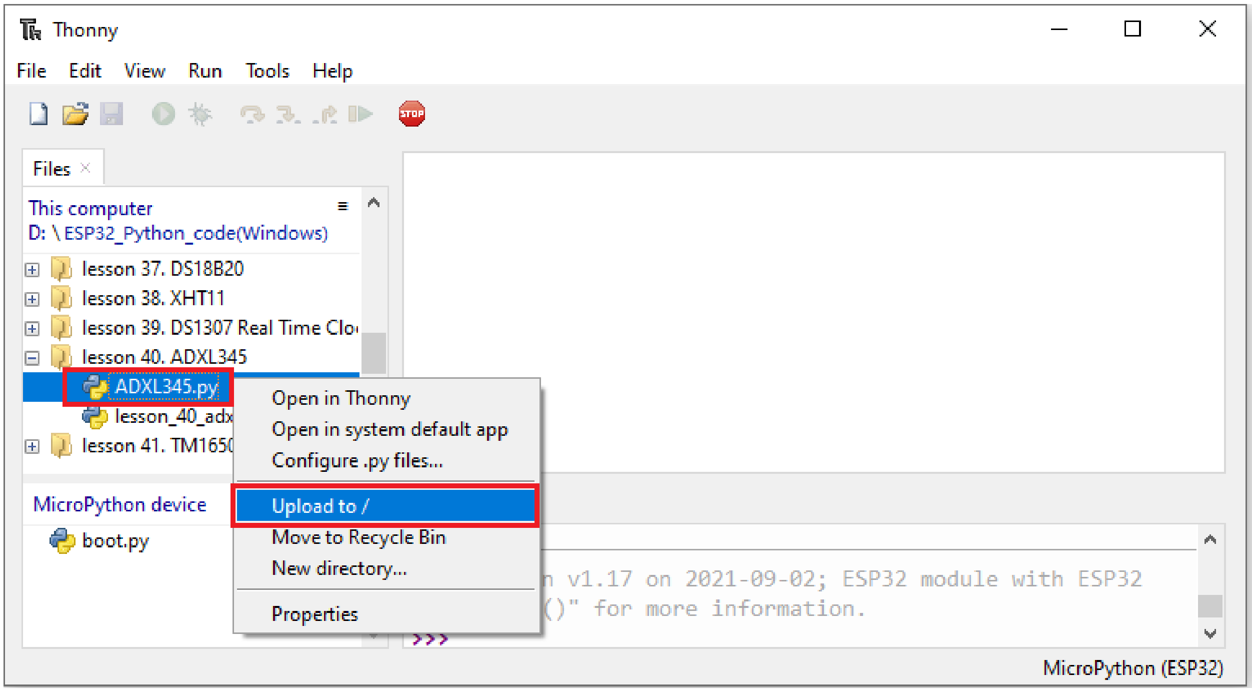
Connection Diagram



Add Library

Open"Thonny", click"This computer"→"D:"→"2. ESP32_code_MicroPython"→"lesson 40. ADXL345".

Select"ADXL345.py"right-click and select"Upload to /"waiting for the "ADXL345.py" to be uploaded to the ESP32.



Test Code

```
from machine import Pin
import time
from ADXL345 import adxl345

scl = Pin(22)
sda = Pin(21)
bus = 0
snsr = adxl345(bus, scl, sda)
while True:
    x,y,z = snsr.readXYZ()
    print('x:',x,'y:',y,'z:',z,'uint:mg')
    time.sleep(0.1)
```

Code Explanation

Set IIC pins, select IIC0sda–>21, scl–>22then assign the value to x, y and z. The shell shows the value of x,y and zunit is mg.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing, the shell will display the corresponding value of the three-axis acceleration in mg, as shown in the following figure, as shown below. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

Shell

```
83
adxl345 found
x: -347.1 y: 491.4 z: 920.4 uint:mg
x: -971.1 y: 315.9 z: 249.6 uint:mg
x: -756.6 y: -947.7 z: -612.3 uint:mg
x: -374.4 y: -491.4 z: -265.2 uint:mg
x: -362.7 y: -292.5 z: 237.9 uint:mg
x: -234.0 y: 405.6 z: 89.7 uint:mg
x: 42.9 y: 600.6 z: 1778.4 uint:mg
x: -296.4 y: -803.4 z: 93.60001 uint:mg
x: -257.4 y: -1060.8 z: 491.4 uint:mg
x: -522.6 y: 951.6 z: 666.9 uint:mg
x: -378.3 y: 819.0 z: 464.1 uint:mg
x: -869.7 y: -764.4 z: 967.2 uint:mg
x: 982.8 y: -1692.6 z: -249.6 uint:mg
x: 721.5 y: -854.1 z: 276.9 uint:mg
x: 323.7 y: -655.2 z: 604.5 uint:mg
x: -561.6 y: 31.2 z: 284.7 uint:mg
x: -889.2 y: 612.3 z: 432.9 uint:mg
x: -569.4 y: 19.5 z: 1567.8 uint:mg
x: -413.4 y: 11.7 z: -288.6 uint:mg
x: -978.9 y: 1930.5 z: 440.7 uint:mg
x: -175.5 y: -542.1 z: 304.2 uint:mg
x: -245.7 y: -448.5 z: 877.5 uint:mg
x: -499.2 y: 850.2 z: 873.6 uint:mg
x: 7.8 y: 163.8 z: 939.9 uint:mg
x: 11.7 y: 159.9 z: 947.7 uint:mg
x: 11.7 y: 156.0 z: 947.7 uint:mg
x: 15.6 y: 159.9 z: 947.7 uint:mg
```

5.2.41 Project 41: TM1650 4-Digit Tube Display

Overview

This module is mainly composed of a 0.36 inch red common cathode 4-digit digital tube, and its driver chip is TM1650. When using it, we only need two signal lines to make the single-chip microcomputer control a 4-bitdigit tube, which greatly saves the IO port resources of the control board.

TM1650 is a special circuit for LED (light emitting diode display) drive control. It integrates MCU input and output control digital interface, data latch, LED drivers, keyboard scanning, brightness adjustment and other circuits.

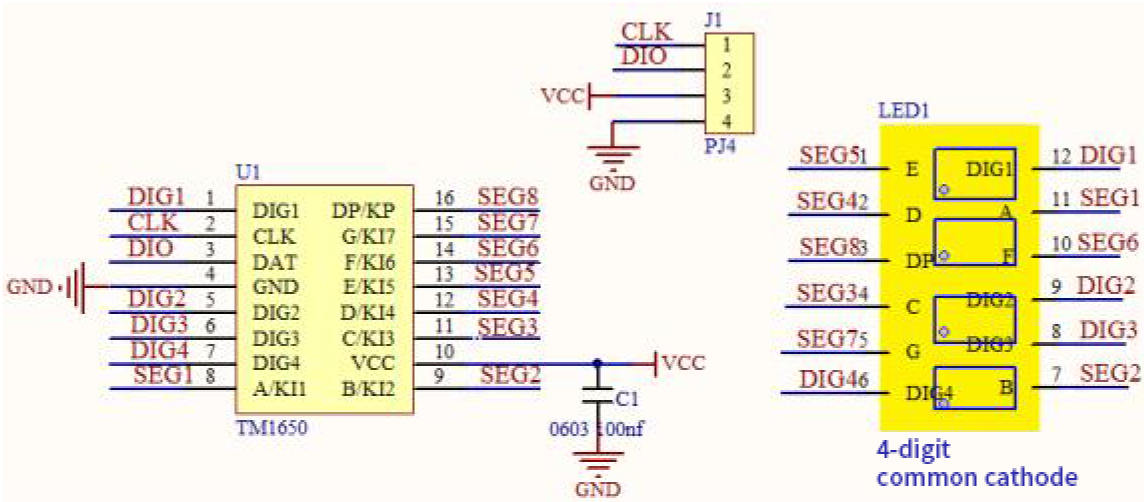
TM1650 has stable performance, reliable quality and strong anti-interference ability.

It can be applied to the application of long-term continuous working for 24 hours.

TM1650 uses 2-wire serial transmission protocol for communication (note that this data transmission protocol is not a standard I2C protocol). The chip can drive the digital tube and save MCU pin resources through two pins and MCU communication.

Working Principle

TM1650 adopts IIC treaty, which uses DIO and CLK buses.



Data command setting: 0x48 means that we light up the digital tube, instead of enable the function of key scanning

B7	B6	B5	B4	B3	B2	B1	B0	Function	Description
×	0	0	0		×	×		Brightness setting	Eight-level brightness
×	0	0	1		×	×			One-level brightness
×	0	1	0		×	×			Two-level brightness
×	0	1	1		×	×			Three-level brightness
×	1	0	0		×	×			Four-level brightness
×	1	0	1		×	×			Five-level brightness
×	1	1	0		×	×			Six-level brightness
×	1	1	1		×	×			Seven-level brightness
×				0	×	×		7/8 segment display control bit	8-segment display way
×				1	×	×			7-segment display way
×					×	×	0	ON/OFF display bit	Off display
×					×	×	1		On display

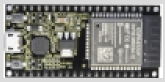
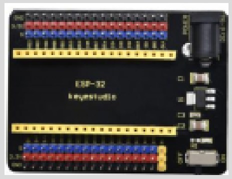
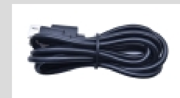
Command display setting:

bit[6:4]set the brightness of tube display, and 000 is brightest

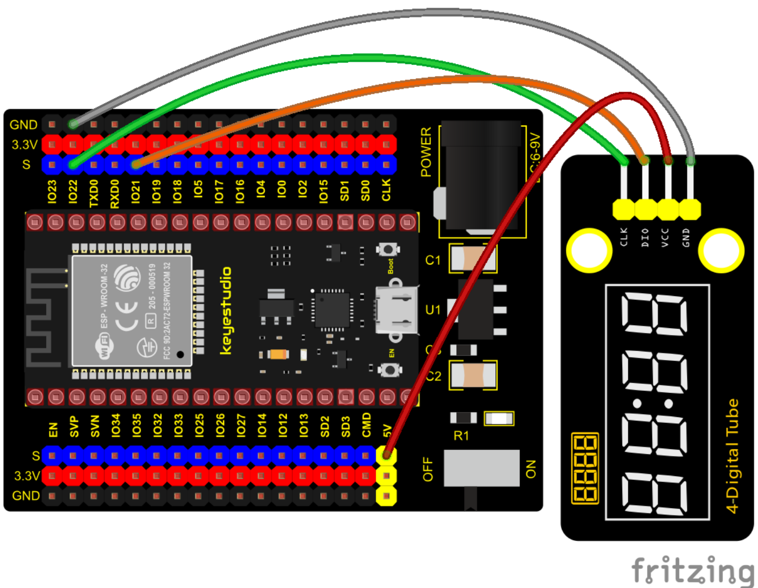
bit[3]set to show decimal points

bit[0]start the display of the tube display

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio TM1650 4-Digit Tube Display*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

# definitions for TM1650
ADDR_DIS = 0x48  #mode command
ADDR_KEY = 0x49  #read key value command

# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTEST      = 7

on  = 1
off = 0

# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]
```

(continues on next page)

(continued from previous page)

```
clkPin = 22
dioPin = 21
clk = Pin(clkPin, Pin.OUT)
dio = Pin(dioPin, Pin.OUT)

DisplayCommand = 0

def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        time.sleep(0.0001)
        clk.value(1)
        time.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return

def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(0)
    return

def ack():
    global clk,dio
    dy = 0
    clk.value(0)
    time.sleep(0.0001)
    dio = Pin(dioPin, Pin.IN)
    while(dio.value() == 1):
        time.sleep(0.0001)
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    time.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, Pin.OUT)
    return

def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
```

(continues on next page)

(continued from previous page)

```
    time.sleep(0.0001)
    dio.value(1)
    return

def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return

def clearBit(bit):
    if(bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    writeByte(0x00)
    ack()
    stop()
    return

    def setBrightness(b = BRIGHT_TYPICAL):
        global DisplayCommand,brightness
        DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
        return
    def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return

def displayOnOFF(OnOff = 1):
```

(continues on next page)

(continued from previous page)

```
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return

def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return

def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return

def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)

InitDigitalTube()

while True:
    #displayDot(1,on)     # on or off, DigitalTube.Display(bit,number); bit=1---4 ␣
↪number=0---9
    for i in range(0,9999):
        ShowNum(i)
        time.sleep(0.01)
```

Code Explanation

clkPin = 22dioPin = 21is pin numberCLK is connected to GPIO22DIO is connected to GPIO21. We can set any two pins at random.

displayBit(bit, num): show numbers at bit(1~4) bit num(0~9)

clearBit(bit): clear up bit(1~4)

setBrightness(): brightness setting

displayOnOFF() 0 means OFF, 1 means ON

displayDot(bit, OnOff) shows dots0 means OFF, 1 means ON

ShowNum(num): show integer numin the range of 0~9999

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. The 4-digit tube display will show integer from 0 to 99999, an increase of 1 for each 10ms, then start from 0 once reaching 99999.

Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.2.42 Project 42: HT16K33_8X8 Dot Matrix Module

Overview

What is the dot matrix display?

If we apply the previous circuit, there will be must one IO port to control only one LED. When more LED need to be controlled, we may adopt a dot matrix.

The 8X8 dot matrix is composed of 64 light-emitting diodes, and each light-emitting diode is placed at the intersection of the row line and the column line. Refer to the experimental schematic diagram below , when the corresponding column is set to a high level and a certain row to low, the corresponding diode will light up. For instance, set pin 13 to a high level and pin 9 to low, and then the first LED will light up.

In the experiment, we display icons via this dot matrix.

Working Principle

As the schematic diagram shown, to light up the LED at the first row and column, we only need to set C1 to high level and R1 to low level. To turn on LEDs at the first row, we set R1 to low level and C1-C8 to high level.

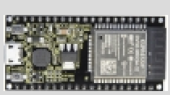
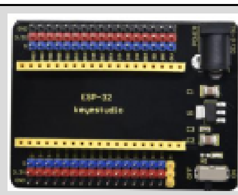
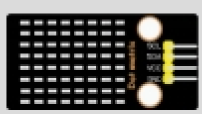
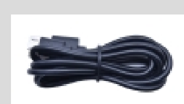
16 IO ports are needed, which will highly waste the MCU resources.

Therefore, we designed this module, using the HT16K33 chip to drive an 8*8 dot matrix, which greatly saves the resources of the single-chip microcomputer.

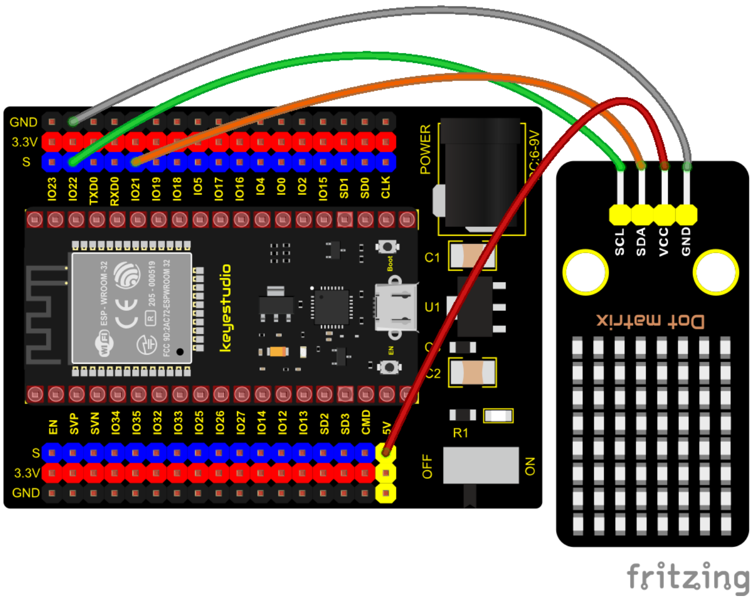
There are three DIP switches on the module, all of which are set to I2C communication address. The setting method is shown below. A0A1 and A2 are grounded, that is, the address is 0x70.

A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）
0X70			0X71			0X72		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
1（ON）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	1（ON）	0（OFF）	1（ON）
0X73			0X74			0X75		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）			
0（OFF）	1（ON）	1（ON）	1（ON）	1（ON）	1（ON）			
0X76			0X77					

Components

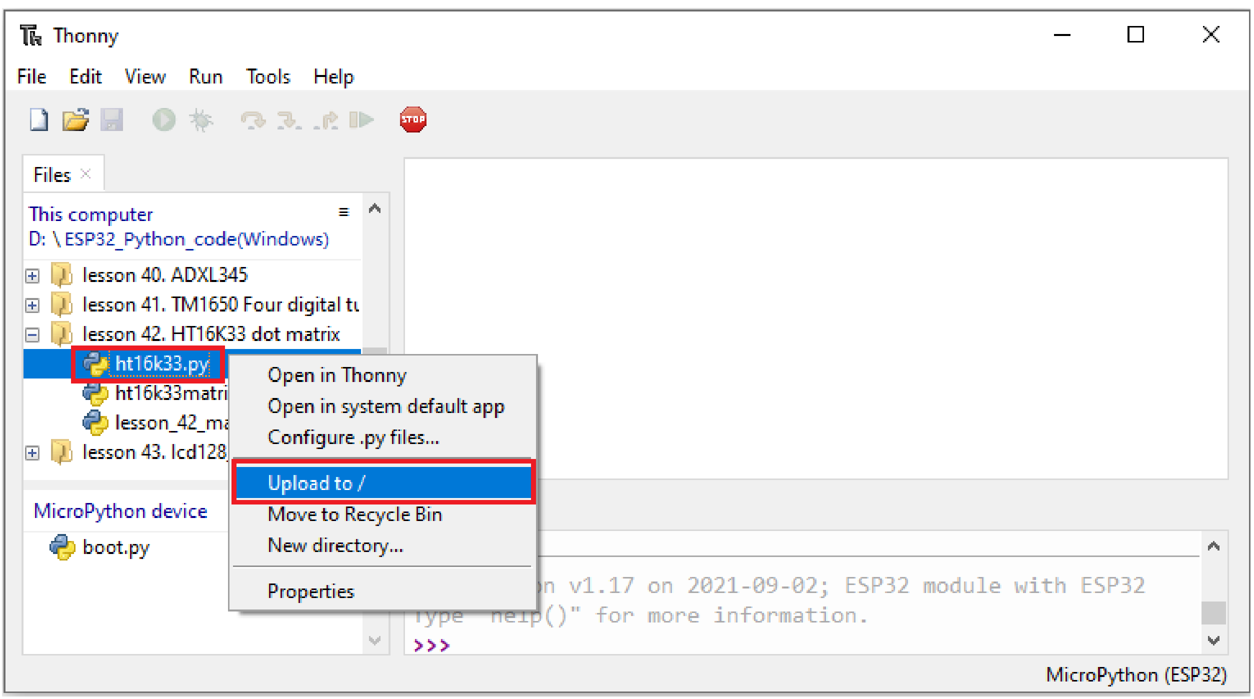
				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HT16K338X8 Dot Matrix*1	4P Dupont Wire*1	Micro USB Cable*1

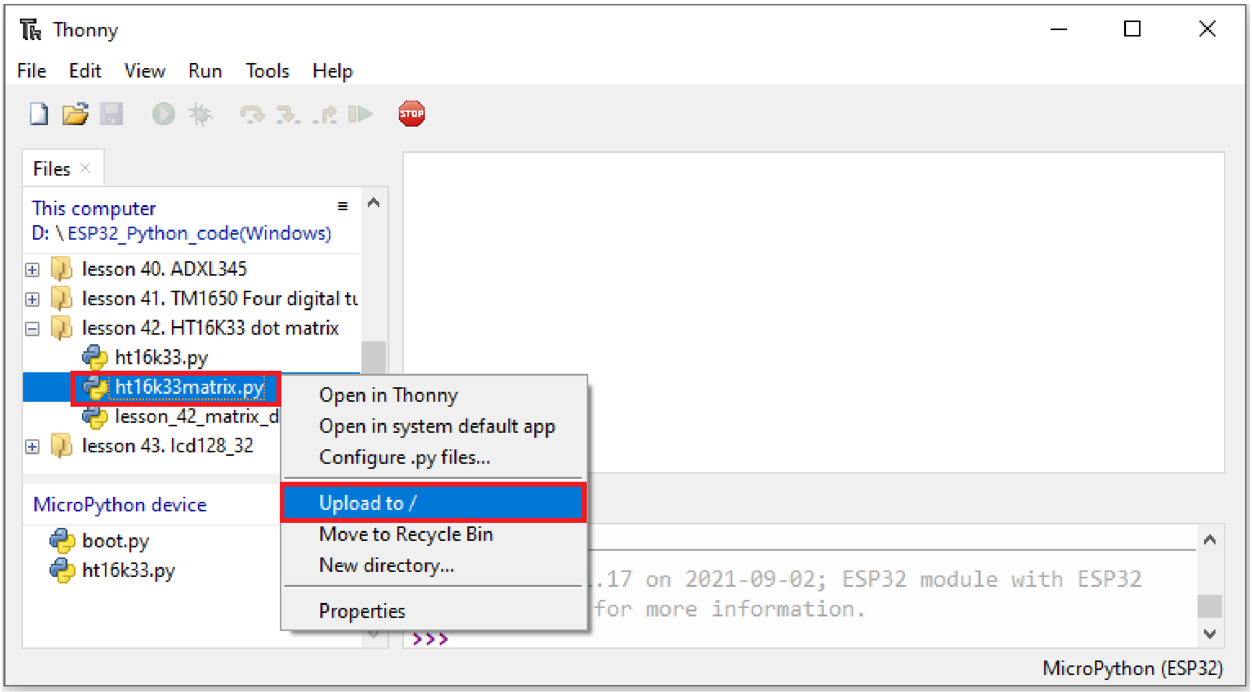
Connection Diagram



Add Library

Open "Thonny", click "This computer" → "D:" → "2. ESP32_code_MicroPython" → "lesson 42. HT16K33 dot matrix". Select "ht16k33.py" and "ht16k33matrix.py", right-click and select "Upload to /", waiting for the "ht16k33.py" and "ht16k33matrix.py" to be uploaded to the ESP32.





Test Code

```python
# IMPORTS
import utime as time
from machine import I2C, Pin, RTC
from ht16k33matrix import HT16K33Matrix

# CONSTANTS
DELAY = 0.01
PAUSE = 3

# START
if __name__ == '__main__':
    i2c = I2C(scl=Pin(22), sda=Pin(21))
    display = HT16K33Matrix(i2c)
    display.set_brightness(2)

    # Draw a custom icon on the LED
    icon = b"\x00\x66\x00\x00\x18\x42\x3c\x00"
    display.set_icon(icon).draw()
    # Rotate the icon
    display.set_angle(0).draw()
    time.sleep(PAUSE)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The dot matrix displays a“ smile ”pattern. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

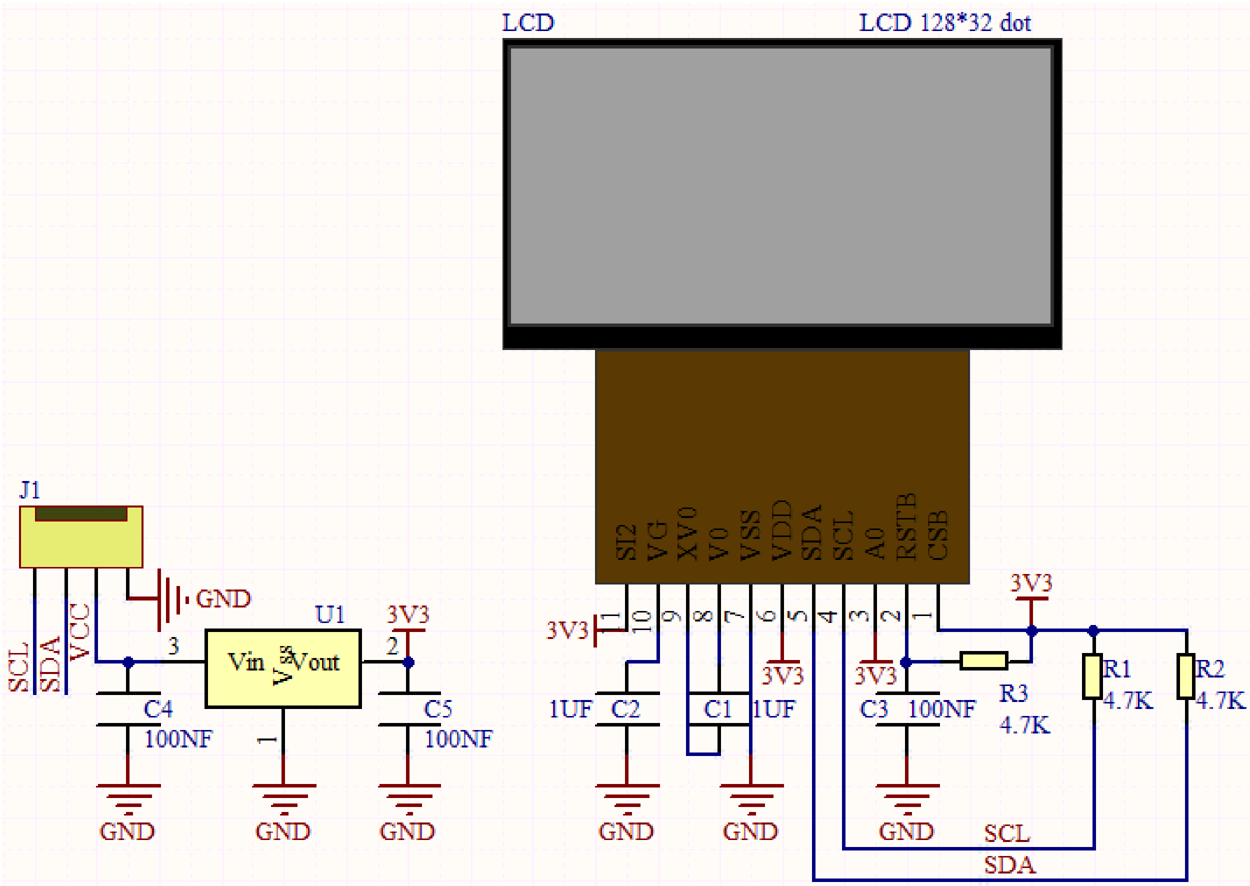
5.2.43 Project 43: LCD_128X32_DOT Module

Description

This is a 128*32 pixel LCD module, which uses IIC communication mode and ST7567A driver chip. At the same time, the code contains all the English letters and common symbols of the library that can be directly called. When used, we can also set English letters and symbols to display different text sizes in our code. To make it easy to set up the pattern display, we also provide a mold capture software that can convert a specific pattern into control code and then copy it directly into the test code for use.

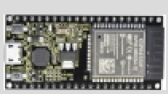
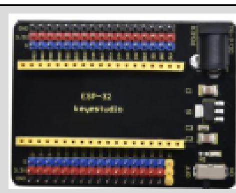
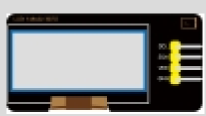
In the experiment, we will set up the display screen to display various English words, common symbols and numbers.

Working Principle

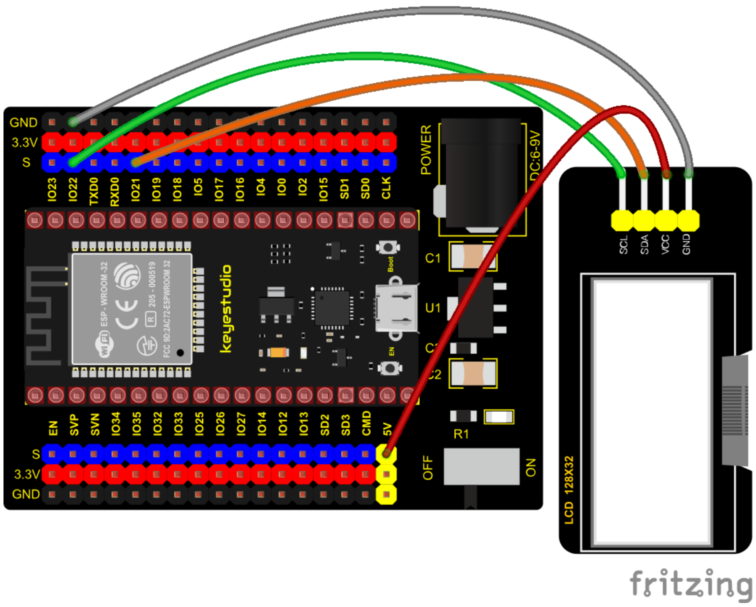


The module uses the IIC communication principle, the underlying functions have been encapsulated in the library surface, we can directly call the library function, if interested, you can also go to understand the underlying driver of the module.

Components

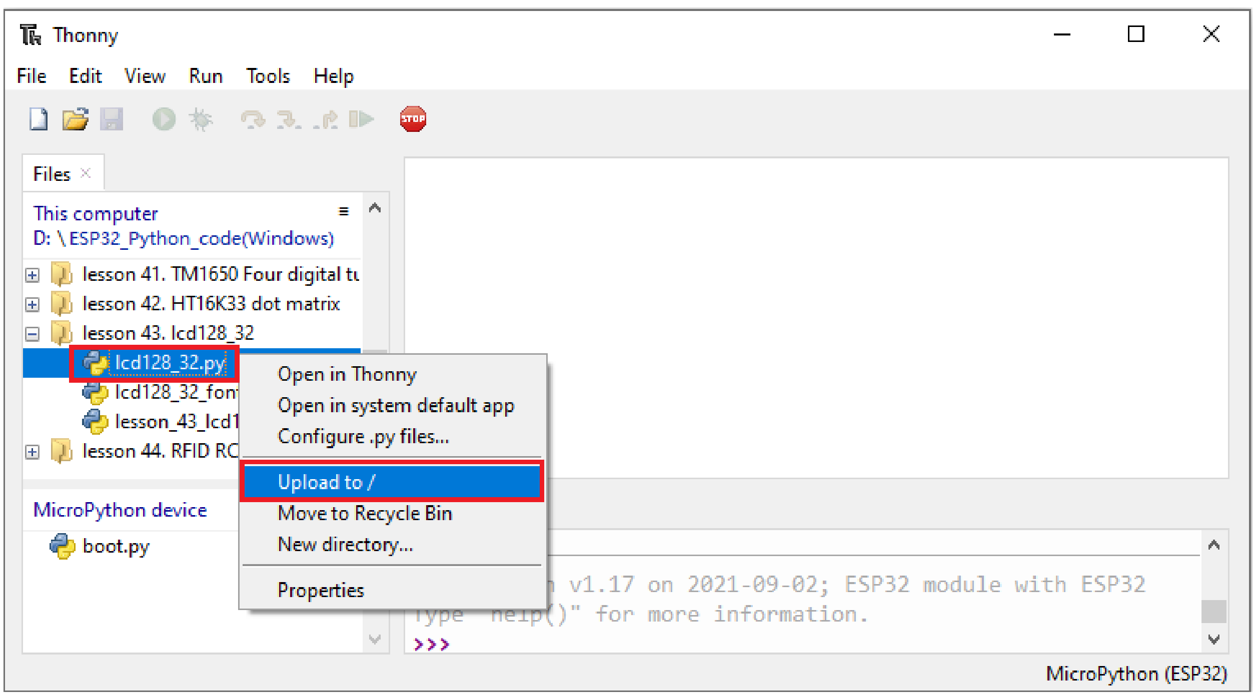
				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio LCD 128X32 DOT Module*1	4P Dupont Wire*1	Micro USB Cable*1

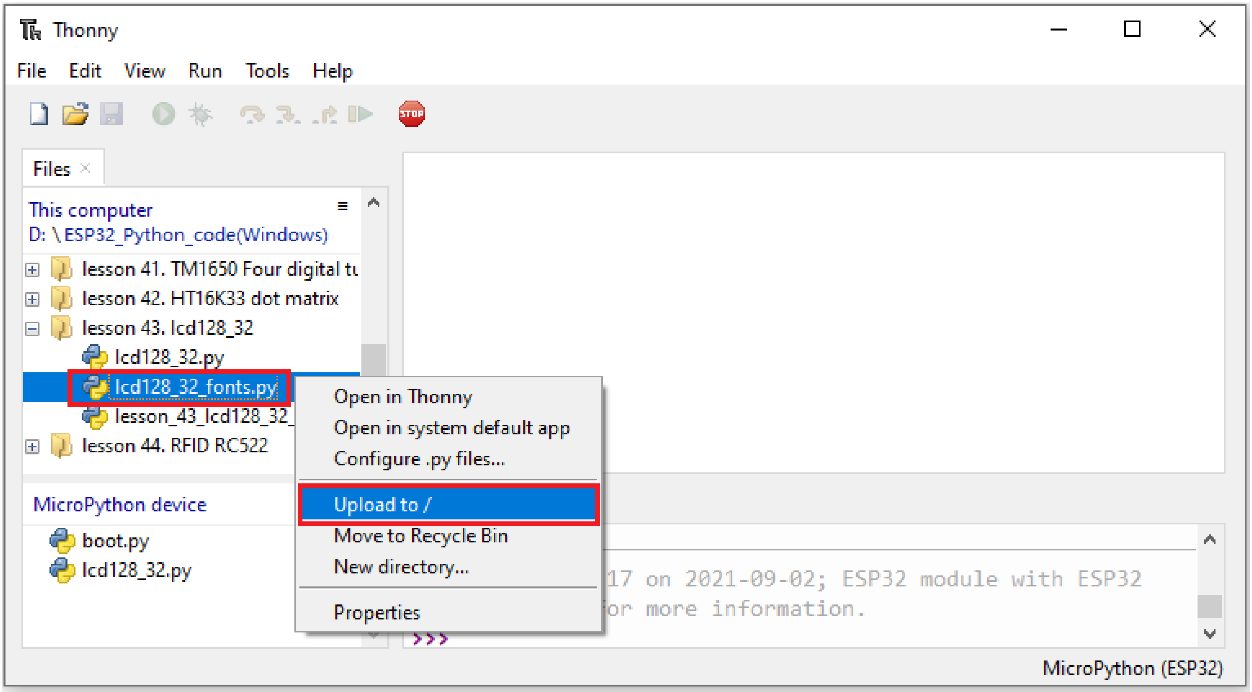
Connection Diagram



Add Library

Open "Thonny", click "This computer" → "D:" → "2. ESP32_code_MicroPython" → "lesson 43. lcd128_32". Select "lcd128_32.py" and "lcd128_32_fonts.py", right-click and select "Upload to /", waiting for the "lcd128_32.py" and "lcd128_32_fonts.py" to be uploaded to the ESP32.





Test Code

```
import machine
import time
import lcd128_32_fonts
from lcd128_32 import lcd128_32

#i2c config
clock_pin = 22
data_pin = 21
bus = 0
i2c_addr = 0x3f
use_i2c = True

def scan_for_devices():
    i2c = machine.I2C(bus,sda=machine.Pin(data_pin),scl=machine.Pin(clock_pin))
    devices = i2c.scan()
    if devices:
        for d in devices:
            print(hex(d))
    else:
        print('no i2c devices')

if use_i2c:
    scan_for_devices()
    lcd = lcd128_32(data_pin, clock_pin, bus, i2c_addr)

    lcd.Clear()
    lcd.Cursor(0, 7)
lcd.Display("KEYES")
lcd.Cursor(1, 0)
```

(continues on next page)

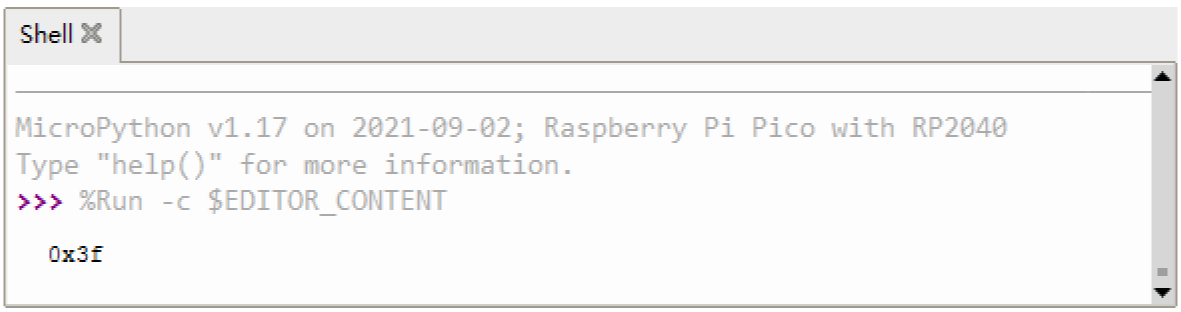
(continued from previous page)

```
lcd.Display("ABCDEFGHIJKLMNOPQR")
lcd.Cursor(2, 0)
lcd.Display("123456789+-*/<>=$@")
lcd.Cursor(3, 0)
lcd.Display("%^&(){}:;'|?,.~\\[]")

while True:
    #scan_for_devices()
    time.sleep(0.5)
```

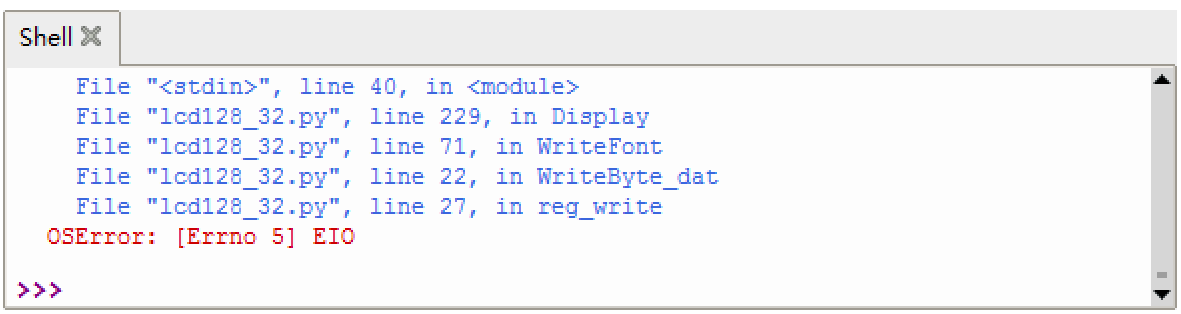
Code Explanation

Scan_for_devices()

This function is an IIC addressing function; if an IIC device is identified, the IIC address of the device is printed, as shown in the figure:



If the device is not recognized, print no i2c devices, and then report an error, as shown in the figure:



lcd.Cursor(0, 7)

In order to set the cursor function, that is, to set the position where the character is displayed on the lcd, the first parameter is the parameter of the row, the second is the parameter of the column, then it is expressed as, the first row, the seventh column starts to display the characters.

lcd.Display("KEYES")

In order to set the character content to be displayed, "KEYES" is displayed here

Test Result

Connect the wires according to the experimental wiring diagram and power on.

Click "Run current script", the code starts executing, the first line of the 128X32LCD module displays "KEYES", the second line displays "ABCDEFGHIJKLMNOPQR", the third line displays "123456789±*/<>=$@", the fourth line displays "%^&(){}:;'|?,.~\[]", as shown in the following image.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.2.44 Project 44: RFID Module

Description

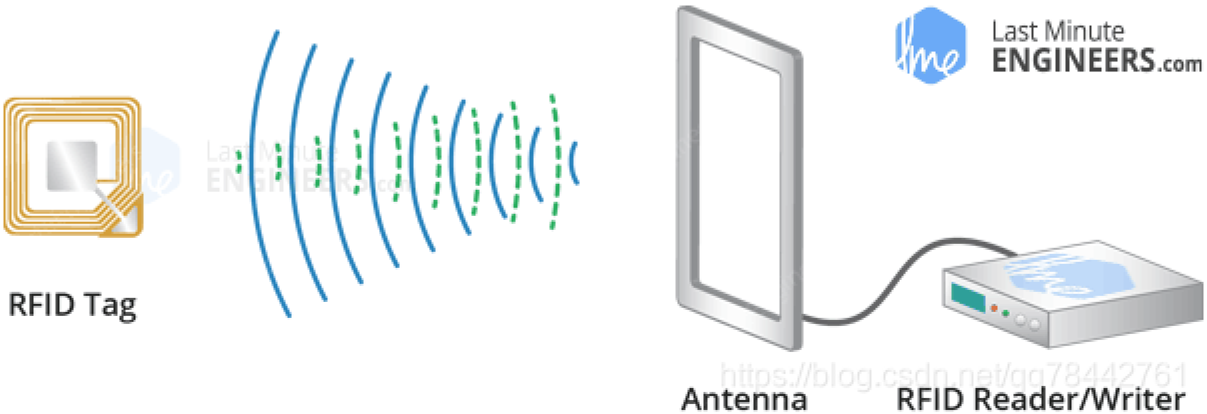
RFIDRFID-RC522 radio frequency module adopts a Philips MFRC522 original chip to design card reading circuit, easy to use and low cost, suitable for equipment development and card reader development and so on.

RFID or Radio Frequency Identification system consists of two main components, a transponder/tag attached to an object to be identified, and a transceiver also known as interrogator/Reader.

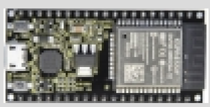
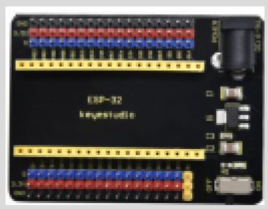
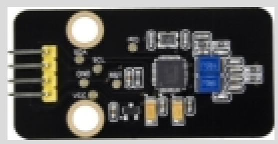
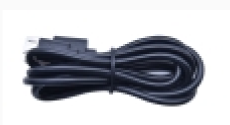
In the experiment, the data read by the card swipe module is 4 hexadecimal numbers, and we print these four hexadecimal numbers as strings. For example, we read the data of the IC card below: 237, 247,148,90 and the data read from the keychain is: 76, 9, 107, 110. Different IC cards and different key chains have diverse data.

Working Principle

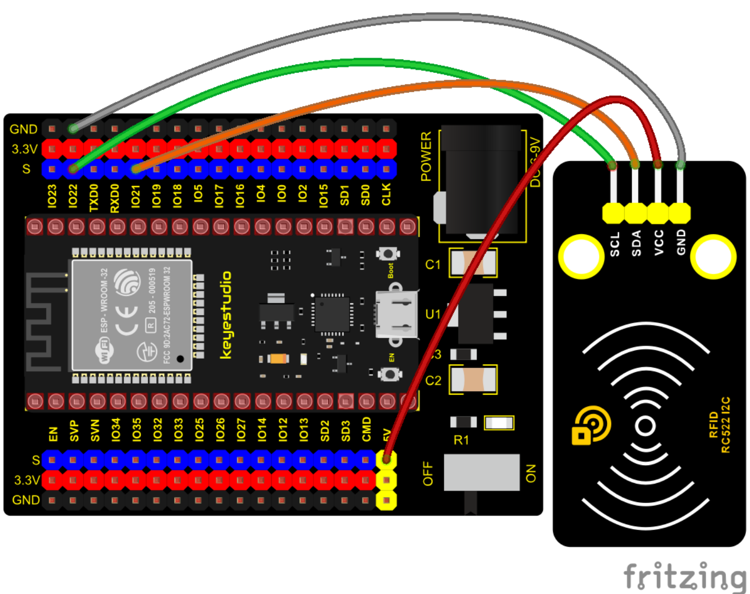
Radio frequency identification, the card reader is composed of a radio frequency module and a high-level magnetic field. The Tag transponder is a sensing device, and this device does not contain a battery. It only contains tiny integrated circuit chips and media for storing data and antennas for receiving and transmitting signals. To read the data in the tag, first put it into the reading range of the card reader. The reader will generate a magnetic field, and because the magnetic energy generates electricity according to Lenz’s law, the RFID tag will supply power, thereby activating the device.



Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY RFID Module*1	4P Dupont Wire*1
			
Micro USB Cable*1	Key*1	IC Card*1	

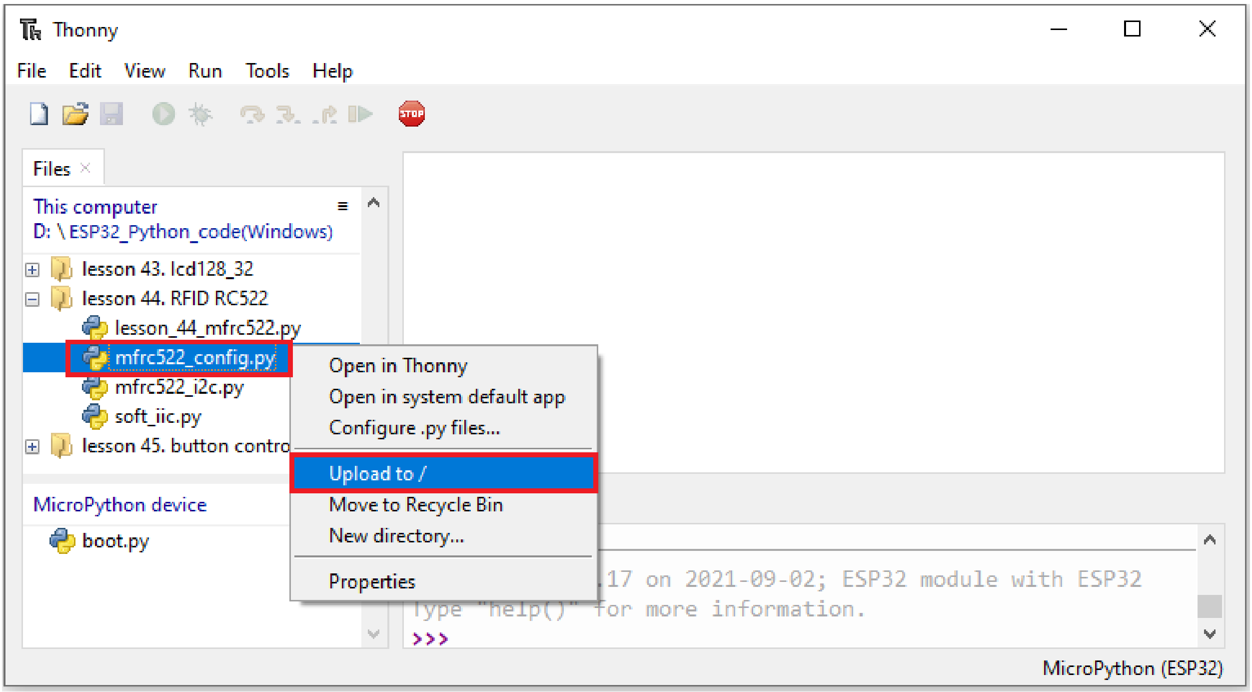
Connection Diagram

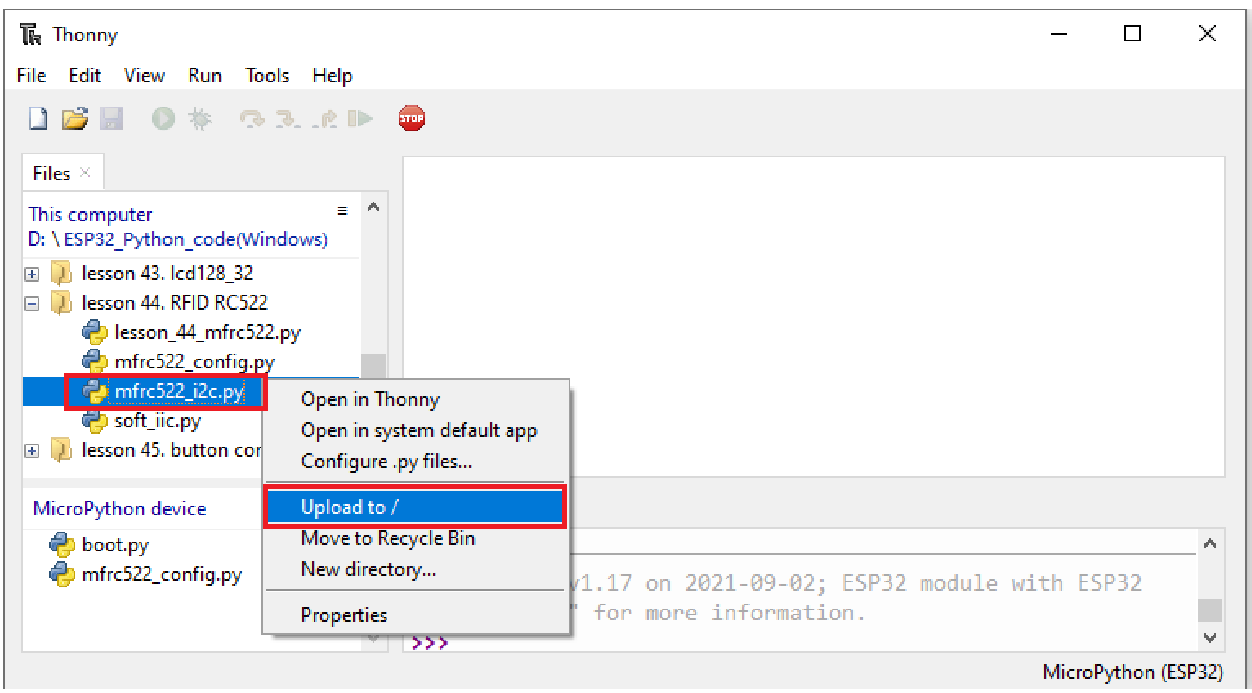


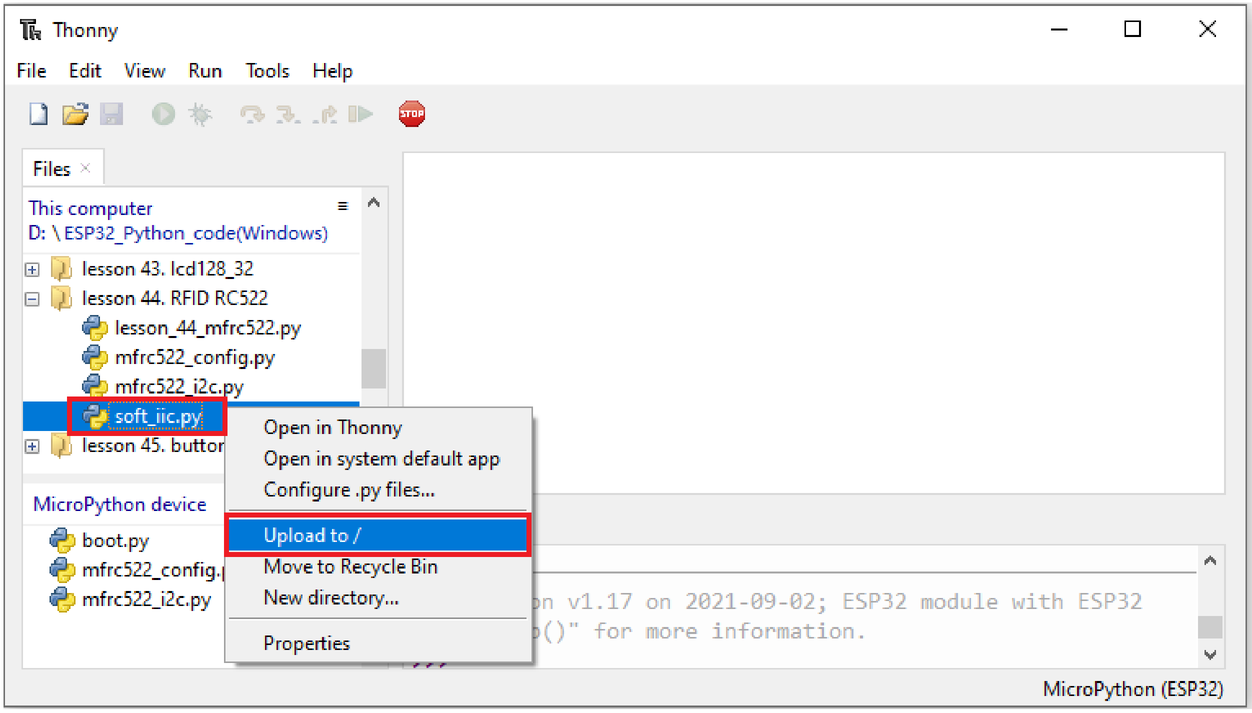
Add Library

Open “Thonny”, click “This computer” → “D:” → “2. ESP32_code_MicroPython” → “lesson 44. RFID RC522”.

Select “mfrc522_config.py”, “mfrc522_i2c.py” and “soft_iic.py”, right-click and select “Upload to /”, waiting for the “mfrc522_config.py”, “mfrc522_i2c.py” and “soft_iic.py” to be uploaded to the ESP32.







Test Code

```
import machine
import time
from mfrc522_i2c import mfrc522

#i2c config
addr = 0x28
scl = 22
sda = 21

rc522 = mfrc522(scl, sda, addr)
rc522.PCD_Init()
rc522.ShowReaderDetails()            # Show details of PCD - MFRC522 Card Reader details

while True:
    if rc522.PICC_IsNewCardPresent():
        #print("Is new card present!")
        if rc522.PICC_ReadCardSerial() == True:
            print("Card UID:")
            print(rc522.uid.uidByte[0 : rc522.uid.size])
    #time.sleep(1)
```

Code Explanation

mfrc522_config.py; This is a configuration file that defines some parameters and commands

mfrc522_i2c.py; Initialization and read and write functions

Soft_iic.py; It is the bottom-level read and write function of software I2C. We use the io port to simulate I2C here.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. When we make the IC card and key chain close to the RFID module, the information will be printed out, as shown in the figure below. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

```
Shell
MFRC522 Software Version:146 = v2.0
Card UID:
[237, 247, 148, 90]
Card UID:
[237, 247, 148, 90]
Card UID:
[237, 247, 148, 90]
Card UID:
[76, 9, 107, 110]
Card UID:
[76, 9, 107, 110]
Card UID:
[76, 9, 107, 110]
```

Note: Different **RFID-RC522** door cards and key chains have diverse values.

5.3 3. Comprehensive Experiments:

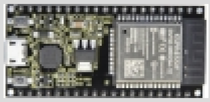
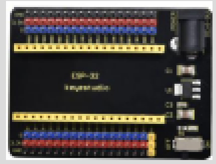
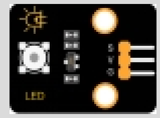
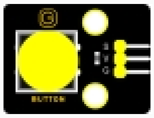
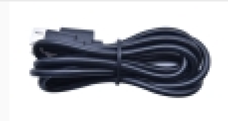
The previous projects are related to single sensor or module. In the following part, we will combine various sensors and modules to create some comprehensive experiments to perform special functions.

5.3.1 Project 45: Button-controlled LED

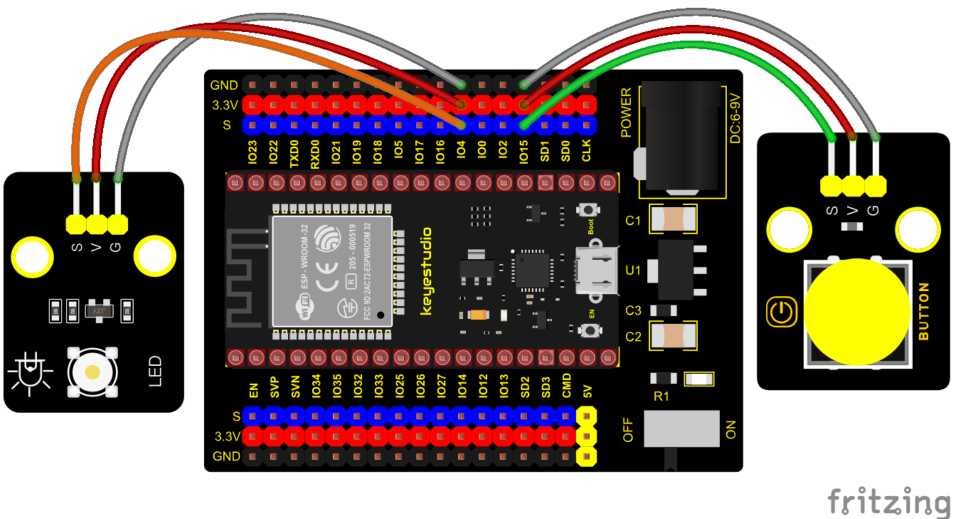
Overview

In this lesson, we will make an extension experiment with a button and an LED. When the button is pressed and low levels are output, the LED will light up; when the button is released, the LED will go off. Then we can control a module with another module.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1
		
Keyestudio DIY Button Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

led = Pin(4, Pin.OUT) # create LED object from Pin 4,Set Pin 4 to output
button = Pin(15, Pin.IN, Pin.PULL_UP) #Create button object from Pin15,Set GP15 to input

#Customize a function and name it reverseGPIO(),which reverses the output level of the LED
def reverseGPIO():
    if led.value():
        led.value(0)     #Set led turn off
    else:
        led.value(1)     #Set led turn on

try:
    while True:
        if not button.value():
            time.sleep_ms(20)
```

(continues on next page)

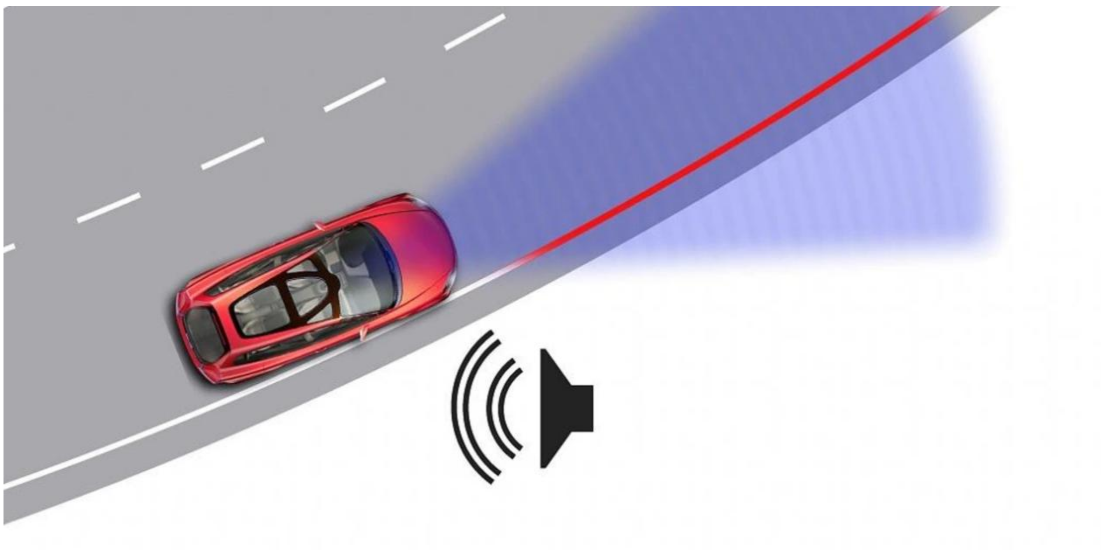
(continued from previous page)

```
            if not button.value():
                reverseGPIO()
                while not button.value():
                    time.sleep_ms(20)
except:
    pass
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing.When the button is pressed, the LED will light up, when pressed again, the LED will go off, cycle this operation. Press “Ctrl+C”or “Stop/Restart backend”to exit the program.

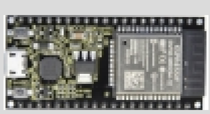
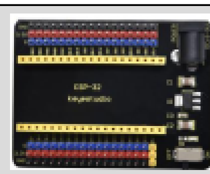
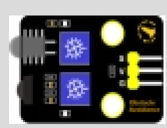
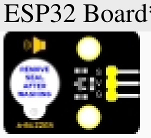
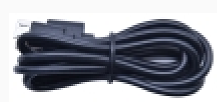
5.3.2 Project 46: Alarm Experiment



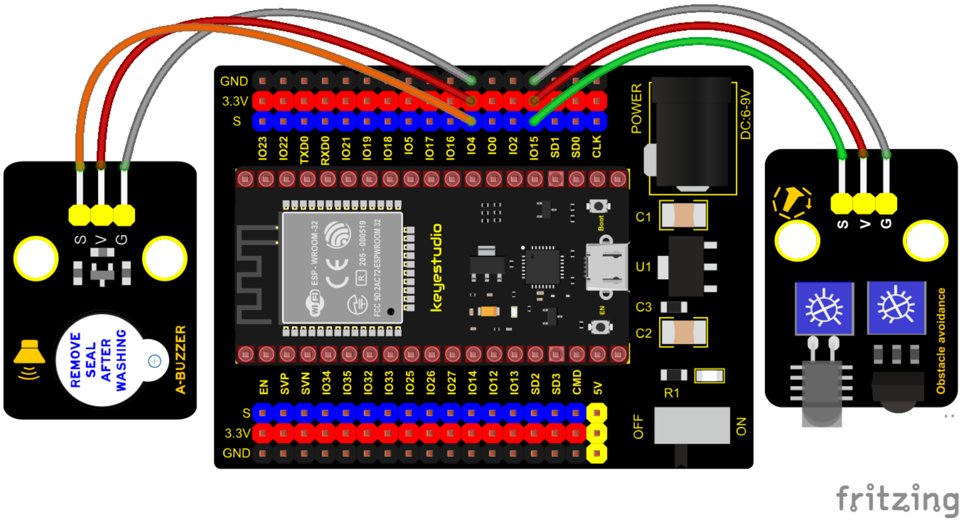
Overview

In the previous experiment, we control an output module though an input module. In this lesson, we will make an experiment that the active buzzer will emit sounds once an obstacle appears.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Obstacle Avoidance Sensor*1
		
Keyestudio Active Buzzer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import Pin
import time

buzzer = Pin(4, Pin.OUT)
sensor = Pin(15, Pin.IN)
while True:
    buzzer.value(not(sensor.value()))
    time.sleep(0.01)
```

Code Explanation

When an obstacle is detected, **sensor.value()** will return a low level signal. So when an obstacle is detected, the GPIO4 connected to the buzzer pin will output a high level signal, the buzzer will emit sounds.

Test Result

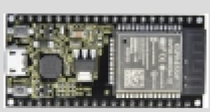
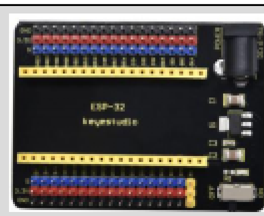
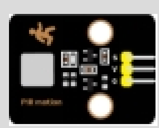
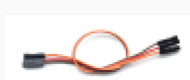
Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The active buzzer will emit sound if detecting obstacles; otherwise, it won’t emit sound. Press “Ctrl+C” or click “Stop/Restart backend” to exit the program.

5.3.3 Project 47: Intrusion Detection

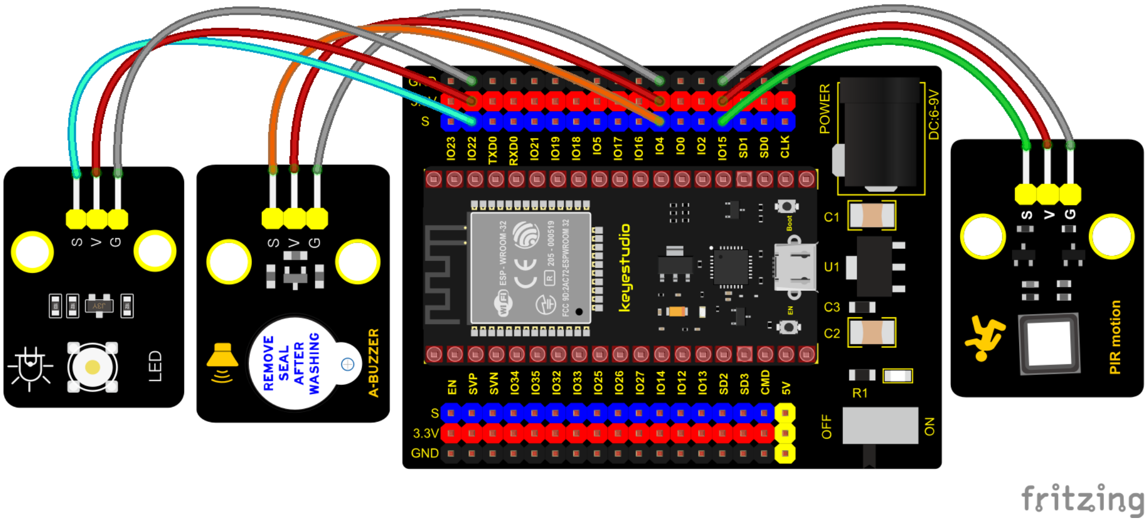
Description

In this experiment, we use a PIR motion sensor to control an active buzzer to emit sounds and the onboard LED to flash rapidly.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	DIY PIR Motion Sensor*1	DIY Active Buzzer*1
			
Purple LED Module*1	3P Dupont Wire*2	Micro USB Cable*1	

Connection Diagram



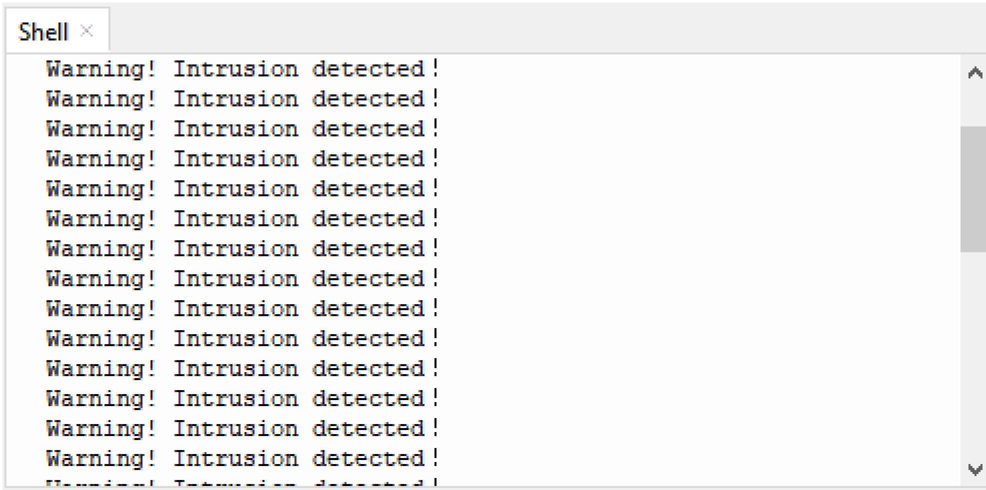
Test Code

```
# Import Pin and time modules.
from machine import Pin
import time

# Define the pins of the Human infrared sensor,led and Active buzzer.
sensor_pir = Pin(15, Pin.IN)
led = Pin(22, Pin.OUT)
buzzer = Pin(4, Pin.OUT)

while True:
      if sensor_pir.value():
          print("Warning! Intrusion detected")
          buzzer.value(1)
          led.value(1)
          time.sleep(0.2)
          buzzer.value(0)
          led.value(0)
          time.sleep(0.2)
      else:
          buzzer.value(0)
          led.value(0)
```

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. If the PIR Motion sensor detects someone moving nearby, the buzzer will emit an alarm , and the LED will flash continuously. At the same time, the “shell” will display“Warning! Intrusion detected”.

Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

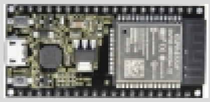
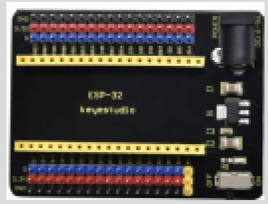
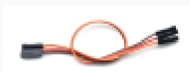
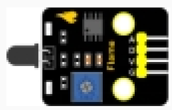


5.3.4 Project 48: Extinguishing Robot

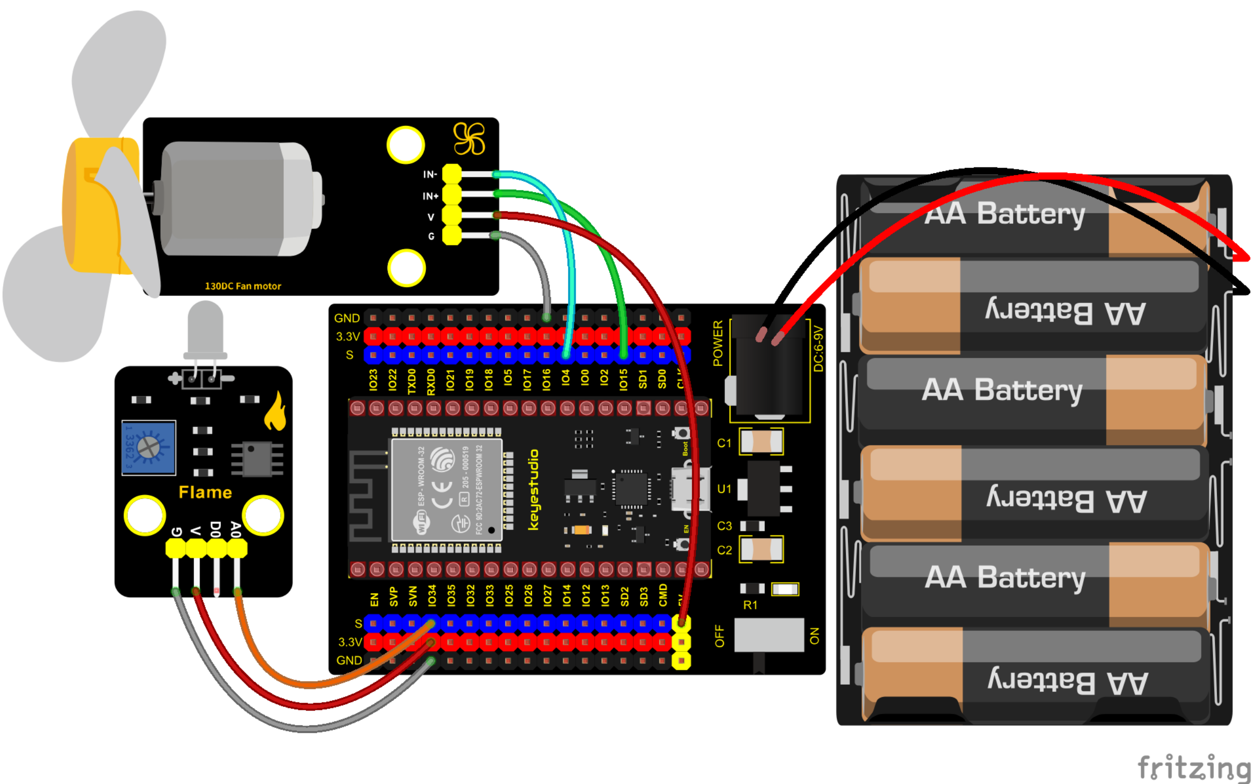
Description

Today we will use Arduino simulation to build an extinguishing robot that will automatically sense the fire and start the fan. In this project we will learn how to build a very simple robot using ESP32, (detecting flames with a flame sensor, blowing out candles with a fan) can teach us basic concepts about robotics. Once you understand the basics below, you can build more complex robots.

Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	130 Motor*1
		
3P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1
		
Battery (provided by yourself)*6	Flame Sensor*1	Battery Holder*1

Connection Diagram



Test Code

```
# Import Pin and ADCmodules.
from machine import ADC,Pin
```

(continues on next page)

(continued from previous page)

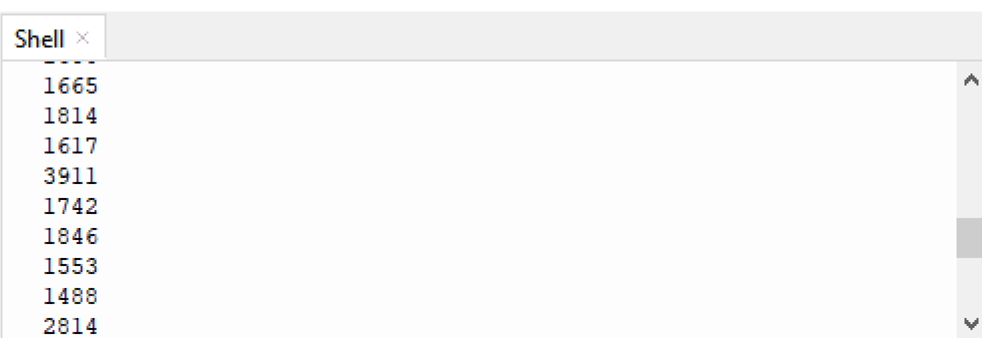
```
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

#Two pins of the moto
INA = Pin(15, Pin.OUT) #INA corresponds to IN+
INB = Pin(4, Pin.OUT) #INB corresponds to IN-

while True:
    adcVal=adc.read()
    print(adcVal)
    if adcVal < 3000:
        #open
        INA.value(0)
        INB.value(1)
    else:
        #stop
        INA.value(0)
        INB.value(0)
    time.sleep(0.1)
```

Code Explanation

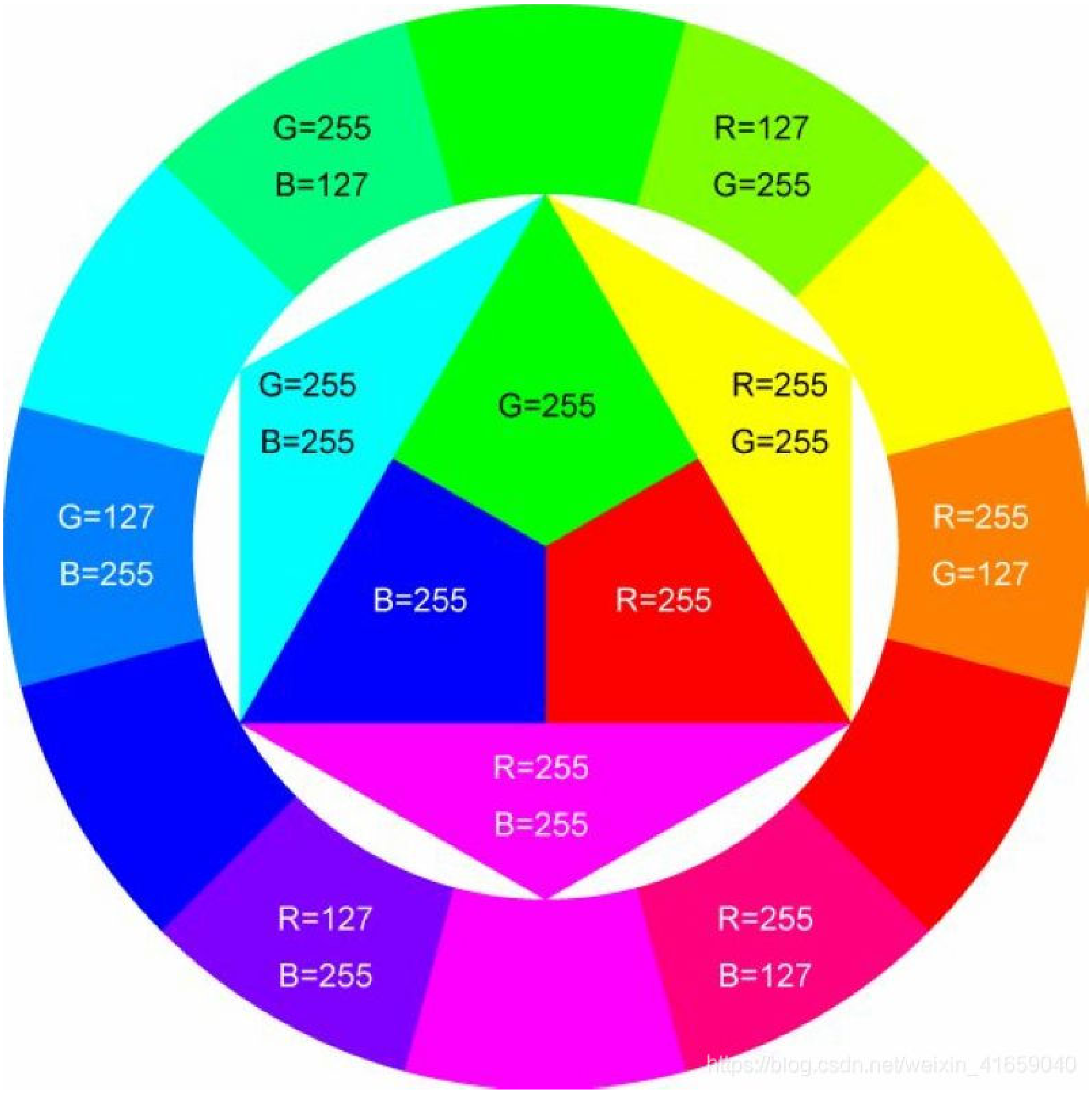
In the code, we set the threshold value to 3000. When the ADC value detected by the flame sensor is lower than the threshold value, the fan will be automatically turned on; otherwise, it will be turned off. For the driving method of the fan, please refer to the 130 Motor.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch on the ESP32 expansion board to the ON end, click “Run current script”, the code starts executing. The shell prints the flame value. When this value is less than 3000, the fan will work to blow out the fire. Basically, the flame value can be set by yourself. Press “Ctrl+C” or click “Stop/Restart backend” to exit the program.



5.3.5 Project 49: Rotary Encoder control RGB

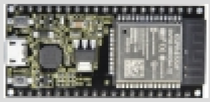
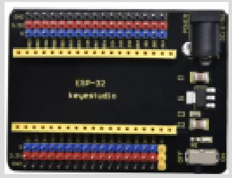
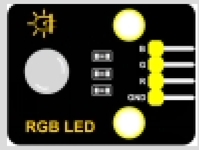
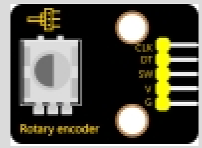
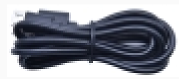


Introduction

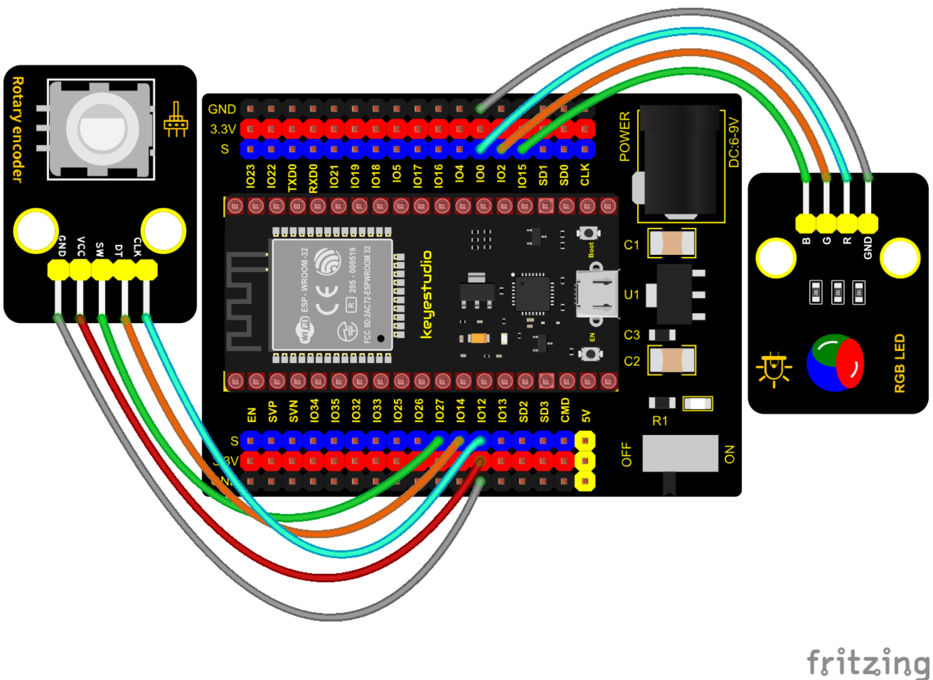
In this lesson, we will control the LED on the RGB module to show different colors through a rotary encoder.

When designing the code, we need to divide the obtained values by 3 to get the remainders. The remainder is 0 and the LED will become red. The remainder is 1, the LED will become green. The remainder is 2, the LED will turn blue.

Components

			
ESP32Board*1	ESP32 Expansion Board*1	KeyestudioCommon Cathode RGB Module*1	KeyestudioRotary Encoder Module*1
			
5P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1	

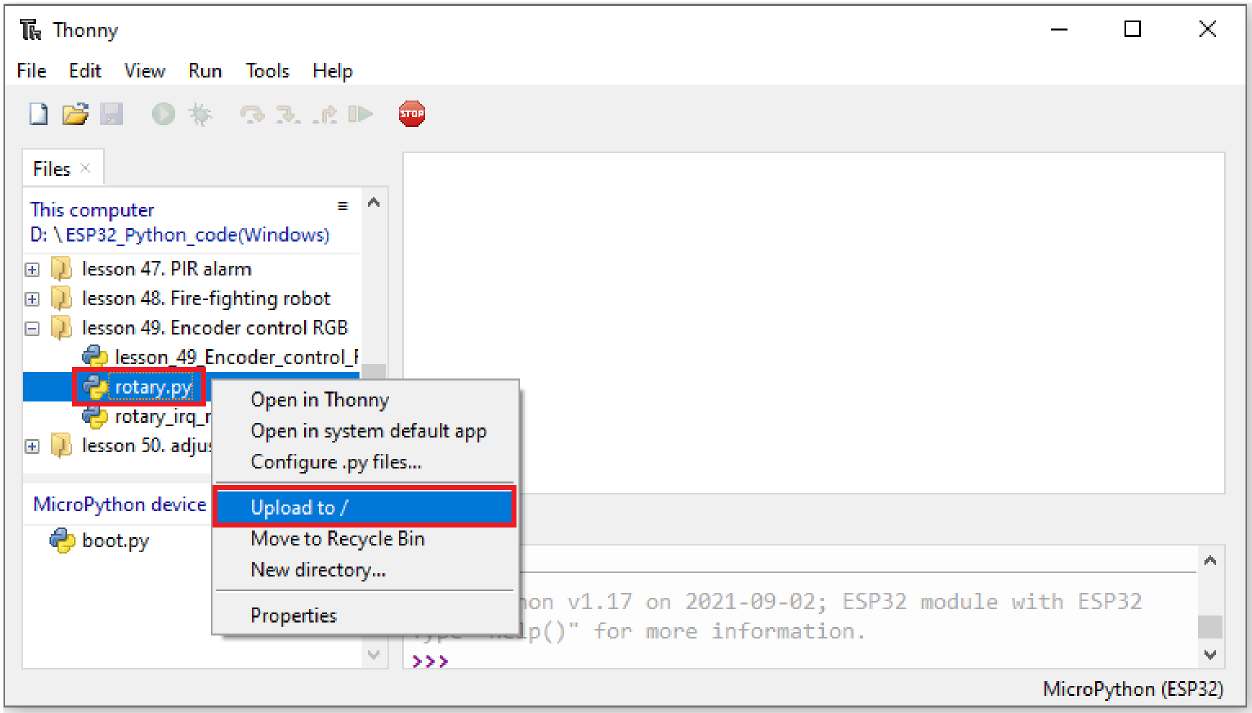
Connection Diagram

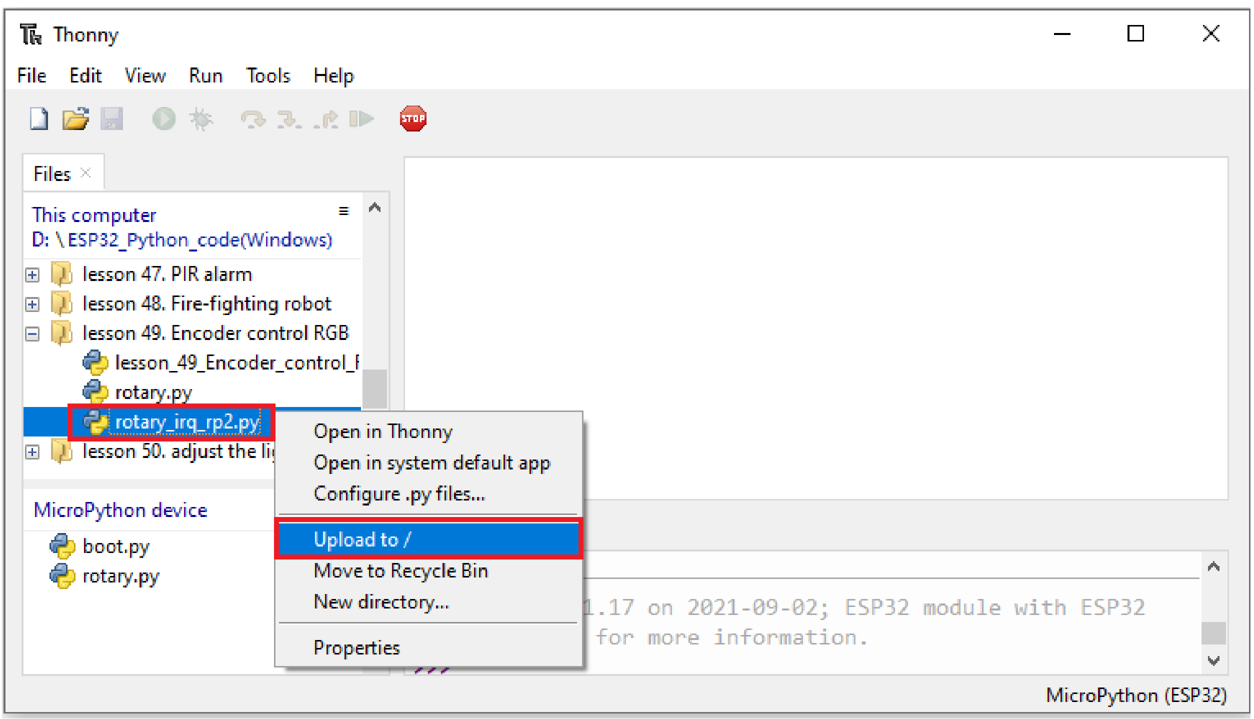


Add Library

Open "Thonny", click "This computer" → "D:" → "2. ESP32_code_MicroPython" → "lesson 49. Encoder control RGB".

Select "rotary.py" and "rotary_irq_rp2.py", right-click and select "Upload to /", waiting for the "rotary.py" and "rotary_irq_rp2.py" to be uploaded to the ESP32.





Test Code

```
import time
from rotary_irq_rp2 import RotaryIRQ
from machine import Pin, PWM

pwm_r = PWM(Pin(0))
```

(continues on next page)

(continued from previous page)

```
pwm_g = PWM(Pin(2))
pwm_b = PWM(Pin(15))

pwm_r.freq(1000)
pwm_g.freq(1000)
pwm_b.freq(1000)

def light(red, green, blue):
    pwm_r.duty(red)
    pwm_g.duty(green)
    pwm_b.duty(blue)

SW=Pin(27,Pin.IN,Pin.PULL_UP)
r = RotaryIRQ(pin_num_clk=12,
              pin_num_dt=14,
              min_val=0,
              reverse=False,
              range_mode=RotaryIRQ.RANGE_UNBOUNDED)

while True:
    val = r.value()
    print(val%3)
    if val%3 == 0:
        light(4950, 0, 0)
    elif val%3 == 1:
        light(0, 4950, 0)
    elif val%3 == 2:
        light(0, 0, 4950)
    time.sleep(0.1)
```

Code Explanation

In the experiment, we set the val to the remainder of Encoder_Count divided by 3. Encoder_Count is the value of the encoder. Then we can set pin GPIO0 (red), GPIO2 (green) and GPIO15 (blue) according to remainders.

Colors of the LEDs can be controlled by remainders.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. Rotate the knob of the rotary encoder to display the reminders, which can control colors of LED(red green blue). Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

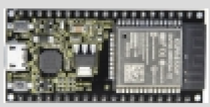
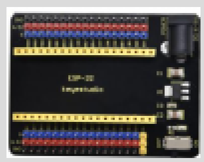
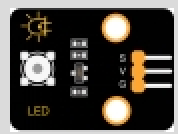
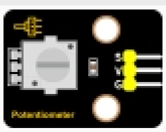
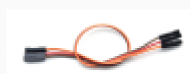
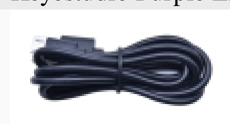
5.3.6 Project 50: Rotary Potentiometer

Introduction

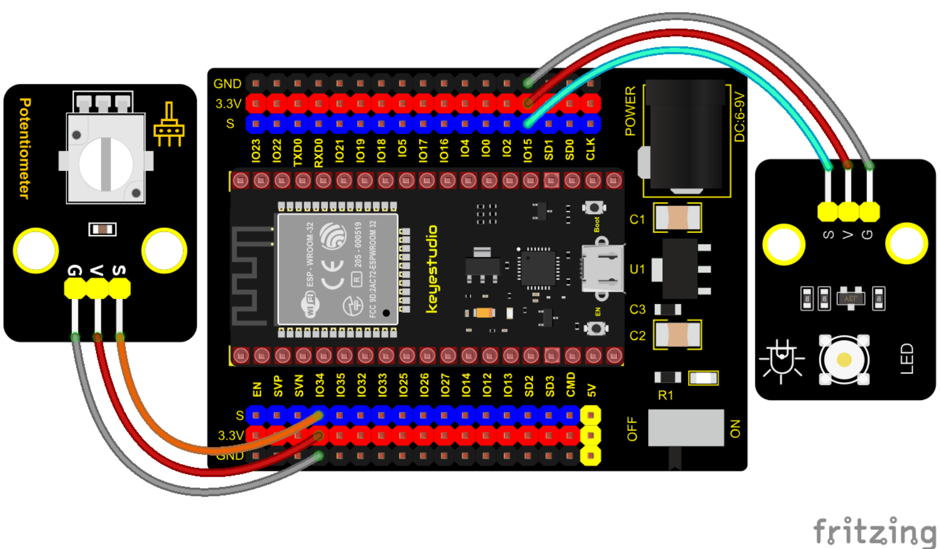
In the previous courses, we did experiments of breathing light and controlling LED with button. In this course, we do these two experiments by controlling the brightness of LED through an adjustable potentiometer. The brightness of LED is controlled by PWM values, and the range of analog values is 0 to 4095 and the PWM value range is 0-255.

After the code is set successfully, we can control the brightness of the LED on the module by rotating the potentiometer.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED*1
		
Keyestudio Rotary Potentiometer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

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

pwm =PWM(Pin(15,Pin.OUT),1000)
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_10BIT)

try:
    while True:
        adcValue=adc.read()
        pwm.duty(adcValue)
        print(adc.read())
        time.sleep_ms(100)
except:
    pwm.deinit()
```

Code Explanation

It is easy to control the brightness of the LED light by a potentiometer. Here we can find that MicroPython unifies the value range of the ADC between 0 and 1023, and assigns values directly, which is simple and convenient.

Test Result

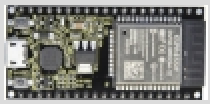
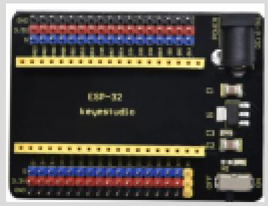
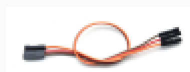
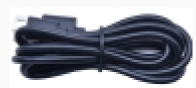
Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. Rotating the potentiometer on the module can adjust the brightness of the LED on the LED module. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.3.7 Project 51: Smart Windows

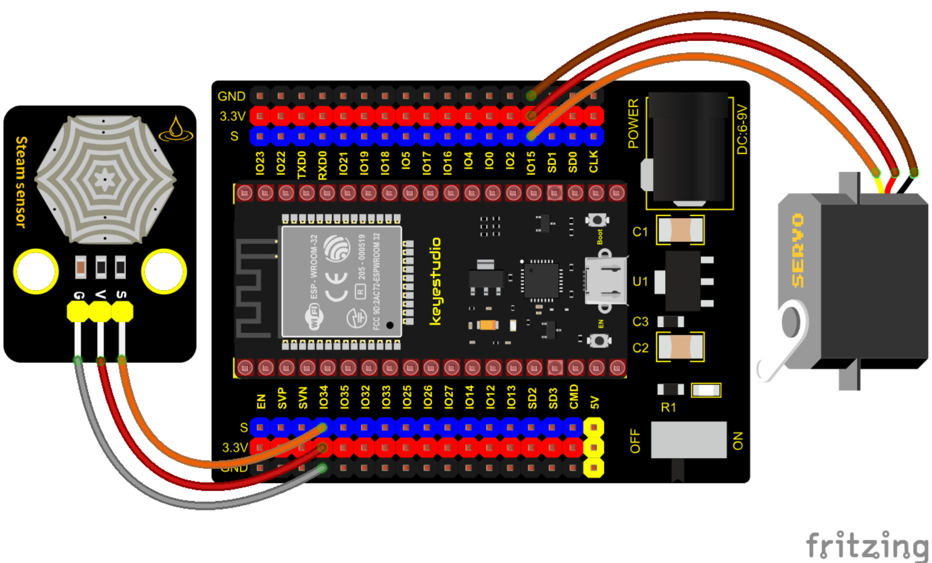
Description

In life, we can see all kinds of smart products, such as smart home. Smart homes include smart curtains, smart windows, smart TVs, smart lights, and more. In this experiment, we use a steam sensor to detect rainwater, and then achieve the effect of closing and opening the window by a servo.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Steam Sensor*1
		
Servo*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin and ADC modules.
from machine import ADC,Pin,PWM
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)
```

(continues on next page)

(continued from previous page)

```
pwm = PWM(Pin(15))#Steering pin connected to GP15
pwm.freq(50)#20ms period, so the frequency is 50Hz
'''
Duty cycle corresponding to the Angle
0°----2.5%----25
45°----5%----51.2
90°----7.5%----77
135°----10%----102.4
180°----12.5%----128
In consideration of the error, the duty cycle is set at 1000~9000, which can smoothly rotate 0~180 degrees
'''
angle_0 = 25
angle_90 = 77
angle_180 = 128

while True:
    adcVal=adc.read()
    print(adcVal)
    if adcVal > 2000:
        pwm.duty(angle_0)
        time.sleep(0.5)
    else:
        pwm.duty(angle_180)
        time.sleep(0.5)
```

Code Explanation

We can control a servo to rotate by a threshold.

Test Result

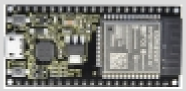
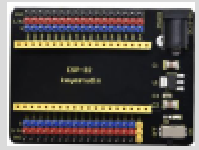
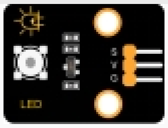
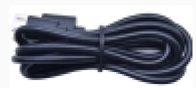
Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. When the sensor detects a certain amount of water, the servo rotates to achieve the effect of closing or opening windows. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.3.8 Project 52: Sound Activated Light

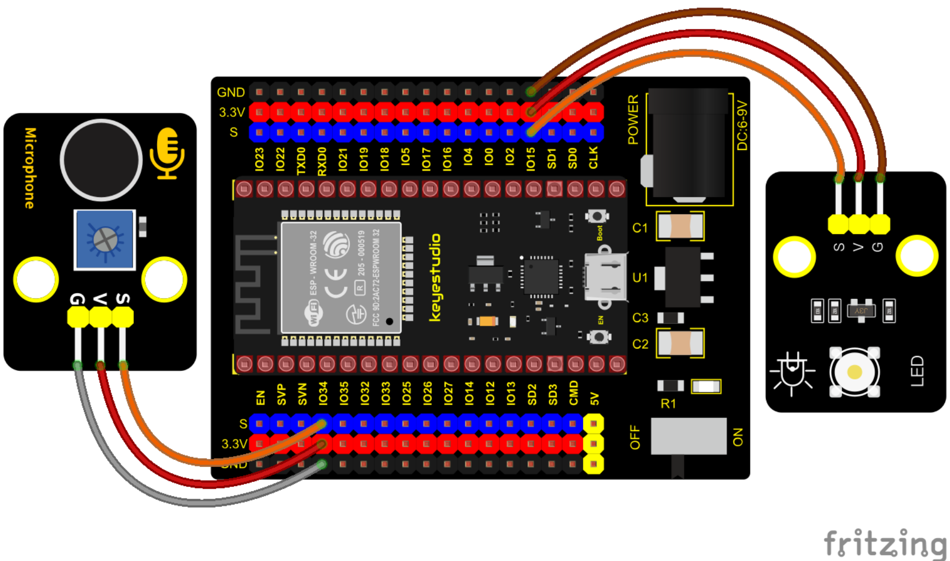
Introduction

In this lesson, we will make a smart sound activated light using a sound sensor and an LED module. When we make a sound, the light will automatically turn on; when there is no sound, the lights will automatically turn off. How it works? Because the sound-controlled light is equipped with a sound sensor, and this sensor converts the intensity of external sound into a corresponding value. Then set a threshold, when the threshold is exceeded, the light will go on, and when it is not exceeded, the light will turn off.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Sound Sensor*1
		
Keyestudio Purple LED Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
from machine import ADC, Pin
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

led = Pin(15,Pin.OUT)

while True:
    adcVal=adc.read()
    print(adcVal)
    if adcVal > 600:
        led.value(1)
        time.sleep(3)
    else:
```

(continues on next page)

(continued from previous page)

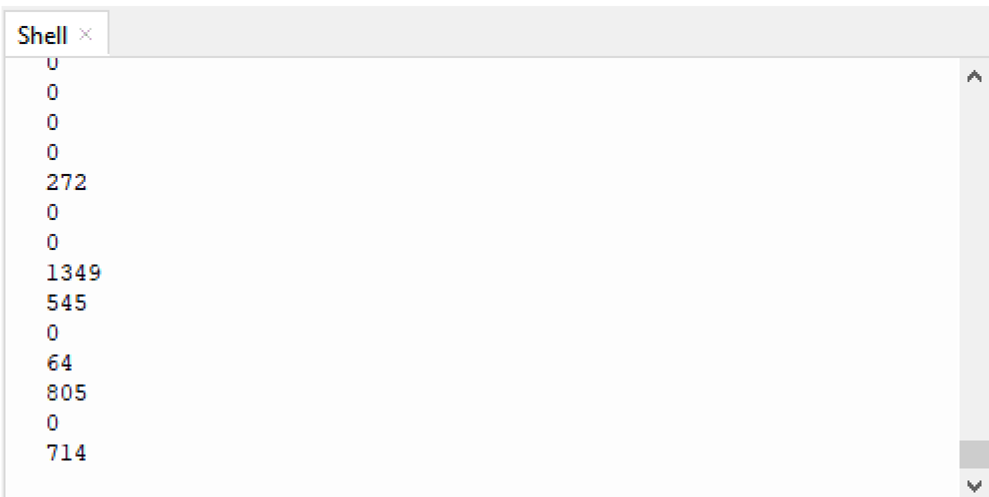
```
        led.value(0)
    time.sleep(0.1)
```

Code Explanation

We set the ADC threshold value to 600. If more than 600, LED will be on 3s; on the contrary, it will be off.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. The shell monitor displays the corresponding volume ADC value. When the analog value of sound is greater than 600, the LED on the LED module will light up, otherwise it will go off. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

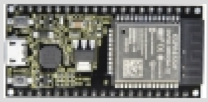
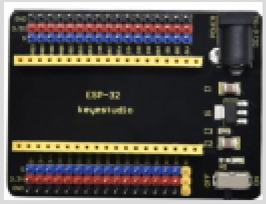
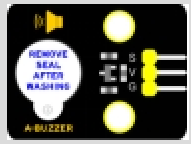
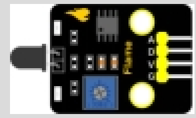
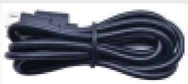


5.3.9 Project 53: Fire Alarm

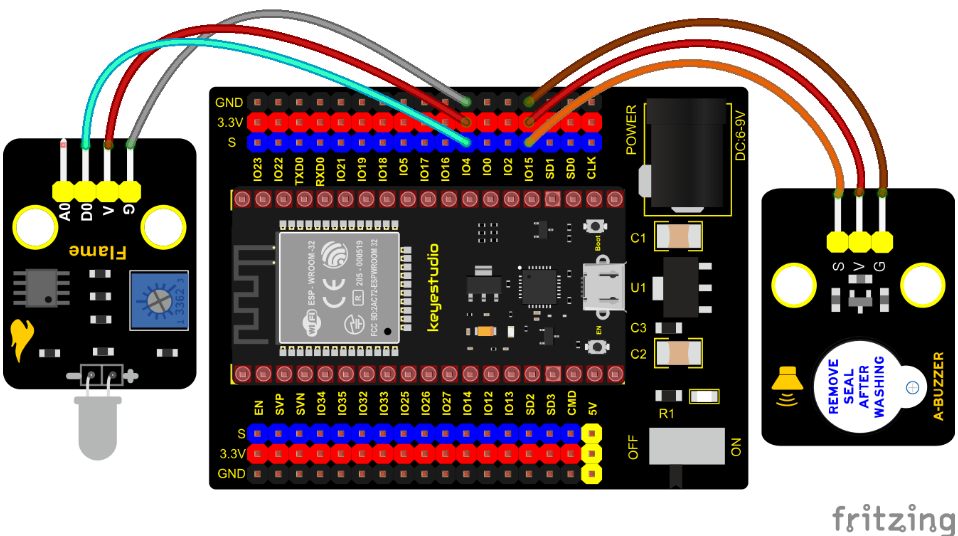
Description

In this experiment, we will make a fire alarm system. Just use a flame sensor to control an active buzzer to emit sounds.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Active Buzzer*1	keyestudio DIY Flame Sensor*1
			
Micro USB Cable*1	3P Dupont Wire*1	4P Dupont Wire*1	

Connection Diagram



Test Code

```
from machine import Pin
import time

buzzer = Pin(15, Pin.OUT)
sensor = Pin(4, Pin.IN)

while True:
    Val = sensor.value()
    print(Val)
    if Val == 0:
        buzzer.value(1)
    else:
        buzzer.value(0)
    time.sleep(0.5)
```

Code Explanation

This flame sensor uses an analog pin and a digital pin. When a flame is detected, the digital pin outputs a low level. In this experiment we will use the digital port.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. When the sensor detects the flame, the external active buzzer will emit sounds, otherwise the active buzzer will not emit sounds.

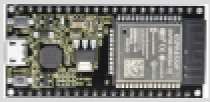
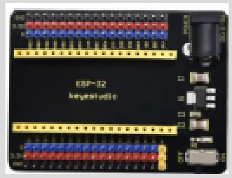
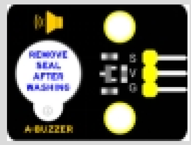
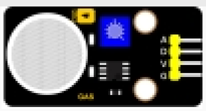
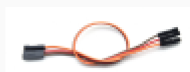
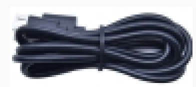
Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.3.10 Project 54: Smoke Alarm

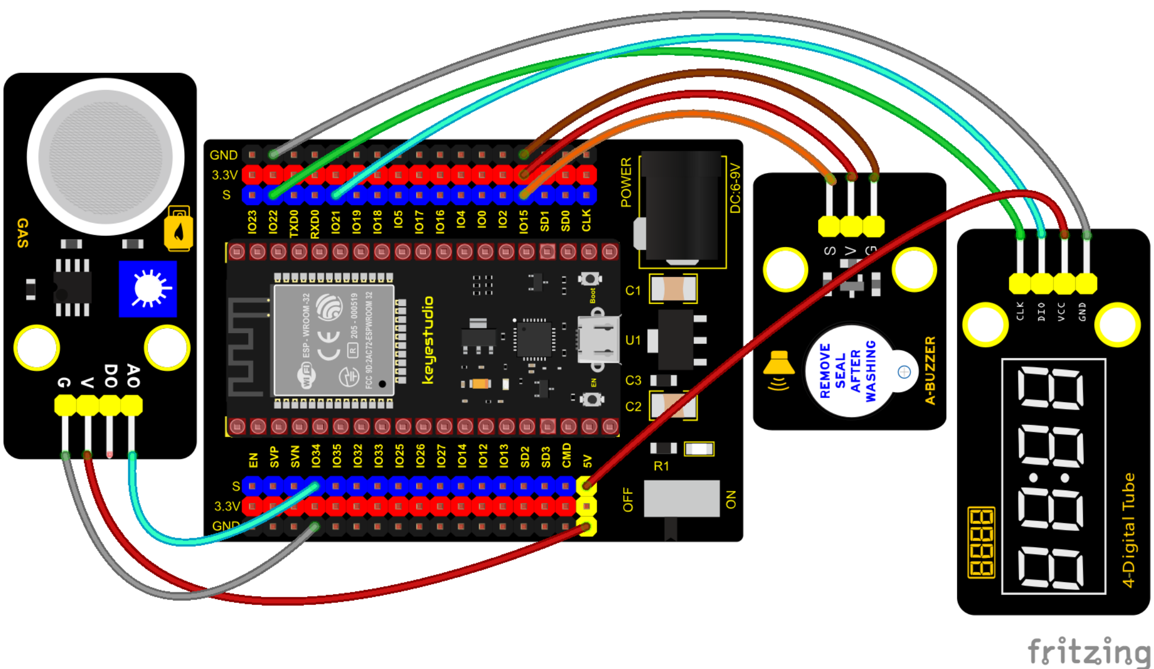
Description

In this experiment, we will make a smoke alarm by a TM16504-Digit segment module, a gas sensor and an active buzzer.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1	Keyestudio TM16504-Digit Segment Module*1
			
keyestudio Analog Gas Senso*1	3P Dupont Wire*1	4P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin and ADC modules.
from machine import ADC,Pin
import time

# Turn on and configure the ADC with the range of 0-3.3V
adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
```

(continues on next page)

(continued from previous page)

```
adc.width(ADC.WIDTH_12BIT)

buzzer = Pin(15, Pin.OUT)
# definitions for TM1650
ADDR_DIS = 0x48  #mode command
ADDR_KEY = 0x49  #read key value command

# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTEST      = 7

on  = 1
off = 0

# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]

clkPin = 22
dioPin = 21
clk = Pin(clkPin, Pin.OUT)
dio = Pin(dioPin, Pin.OUT)

DisplayCommand = 0

def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        time.sleep(0.0001)
        clk.value(1)
        time.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return

def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(0)
    return

def ack():
```

(continues on next page)

(continued from previous page)

```
    global clk,dio
    dy = 0
    clk.value(0)
    time.sleep(0.0001)
    dio = Pin(dioPin, Pin.IN)
    while(dio.value() == 1):
        time.sleep(0.0001)
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    time.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, Pin.OUT)
    return

def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(1)
    return

def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return

def clearBit(bit):
    if(bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
```

(continues on next page)

(continued from previous page)

```
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    writeByte(0x00)
    ack()
    stop()
    return

    def setBrightness(b = BRIGHT_TYPICAL):
        global DisplayCommand,brightness
        DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
        return
    def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return

def displayOnOFF(OnOff = 1):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return

def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return

def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return

def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
```

(continues on next page)

(continued from previous page)

```
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)

InitDigitalTube()

while True:
    adcVal=adc.read()
    print(adcVal)
    ShowNum(adcVal)
    if adcVal > 1000:
        buzzer.value(1)
    else:
        buzzer.value(0)
    time.sleep(0.1)
```

Code Explanation

Define an integer variable val to store the ADC value of the smoke sensor, and then we display the analog value in the four-digit digital tube, and then set a threshold, and when the threshold is reached, the buzzer will sound.

Test Result

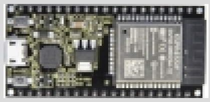
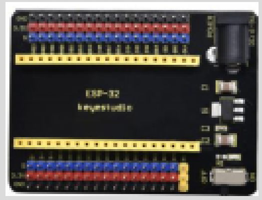
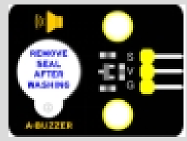
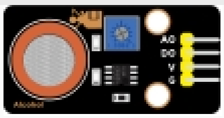
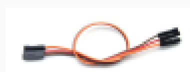
Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. When the concentration of combustible gas exceeds the standard, the active buzzer module will give an alarm, and the four-digit digital tube will display the concentration value. Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.3.11 Project 55: Alcohol Sensor

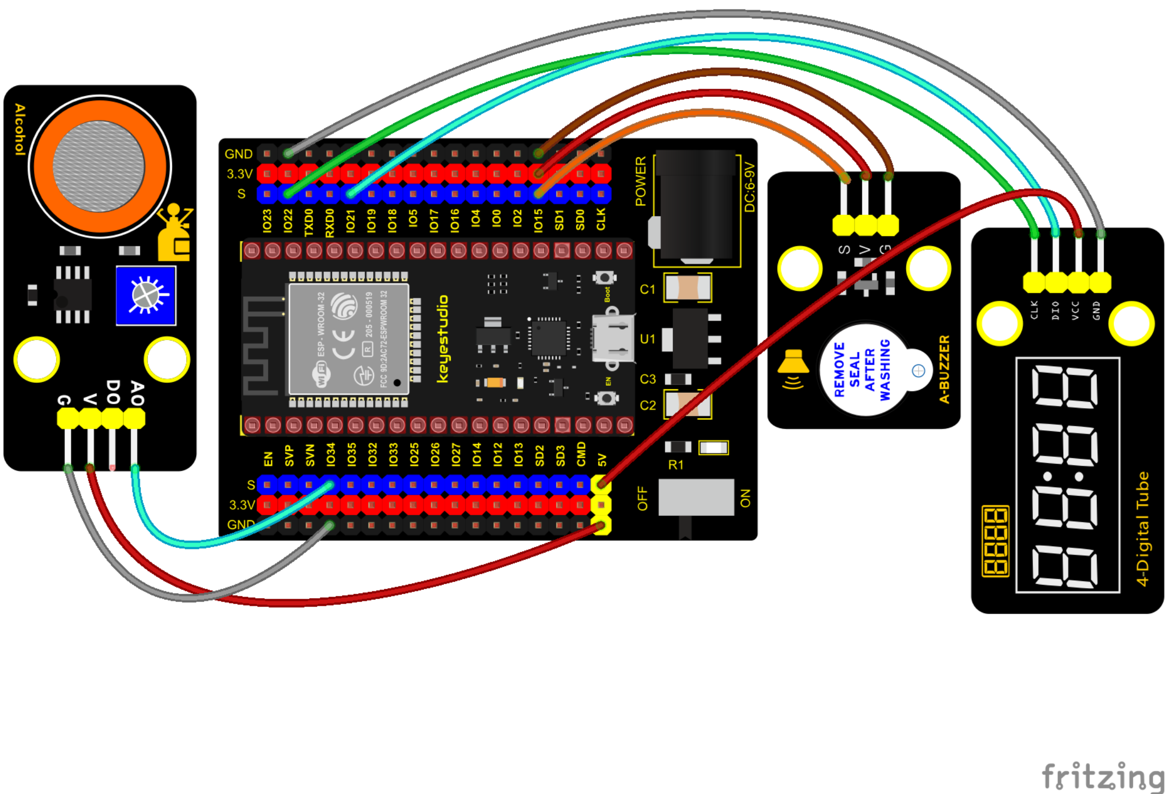
Description

In the last experiment, we made a smoke alarm. In this experiment, we combine the active buzzer, the MQ-3 alcohol sensor, and a four-digit digital tube to test the alcohol concentration through the alcohol sensor. Then, the concentration to control the active buzzer alarm and the four-digit digital tube to display the concentration. So as to achieve the simulation effect of alcohol detector.

Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Active Buzzer*1	Keyestudio DIY TM1650 4-Digit Display*1
			
keyestudio Alcohol Sensor*1	3P Dupont Wire*1	4P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
# Import Pin and ADC modules.
from machine import ADC,Pin
import time

adc=ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
```

(continues on next page)

(continued from previous page)

```
adc.width(ADC.WIDTH_12BIT)

buzzer = Pin(15, Pin.OUT)
# definitions for TM1650
ADDR_DIS = 0x48  #mode command
ADDR_KEY = 0x49  #read key value command

# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTEST      = 7

on  = 1
off = 0

# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]

clkPin = 22
dioPin = 21
clk = Pin(clkPin, Pin.OUT)
dio = Pin(dioPin, Pin.OUT)

DisplayCommand = 0

def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        time.sleep(0.0001)
        clk.value(1)
        time.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return

def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(0)
    return

def ack():
```

(continues on next page)

(continued from previous page)

```
    global clk,dio
    dy = 0
    clk.value(0)
    time.sleep(0.0001)
    dio = Pin(dioPin, Pin.IN)
    while(dio.value() == 1):
        time.sleep(0.0001)
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    time.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, Pin.OUT)
    return

def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(1)
    return

def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return

def clearBit(bit):
    if(bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
```

(continues on next page)

(continued from previous page)

```
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    writeByte(0x00)
    ack()
    stop()
    return

    def setBrightness(b = BRIGHT_TYPICAL):
        global DisplayCommand,brightness
        DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
        return
    def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return

def displayOnOFF(OnOff = 1):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return

def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return

def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return

def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
```

(continues on next page)

(continued from previous page)

```
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)

InitDigitalTube()

while True:
    adcVal=adc.read()
    print(adcVal)
    ShowNum(adcVal)
    if adcVal > 1000:
        buzzer.value(1)
    else:
        buzzer.value(0)
    time.sleep(0.1)
```

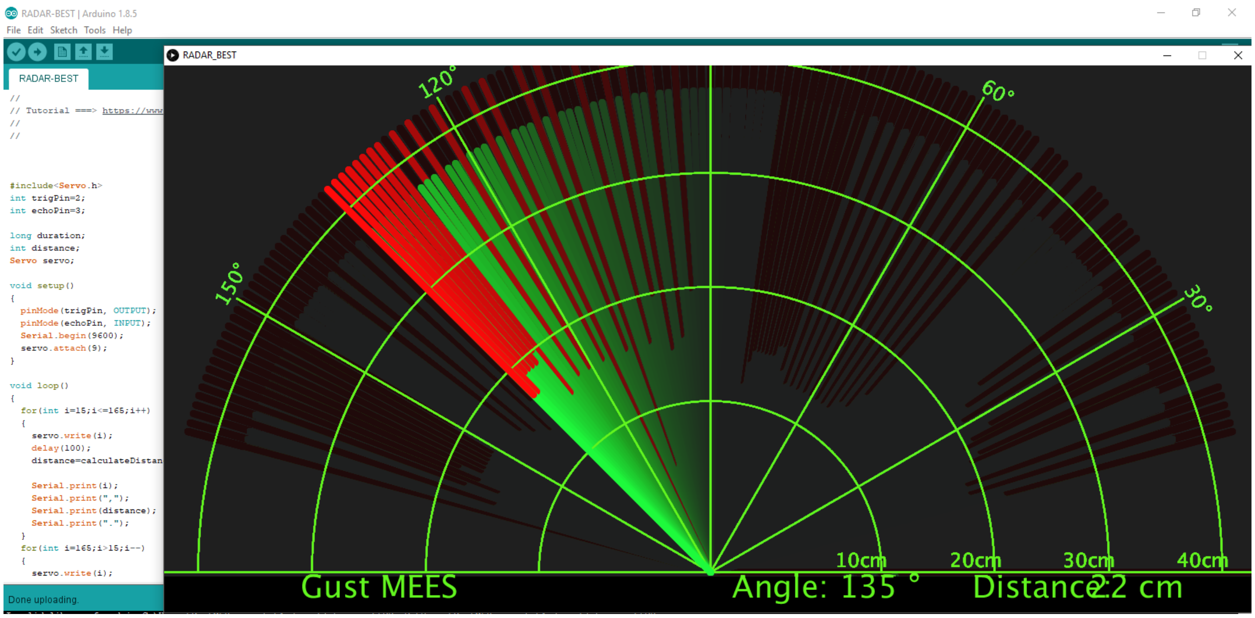
Code Explanation

Define an integer variable val to store the ADC value of the alcohol sensor, then we display the analog value in the four-digit display module and set a threshold.

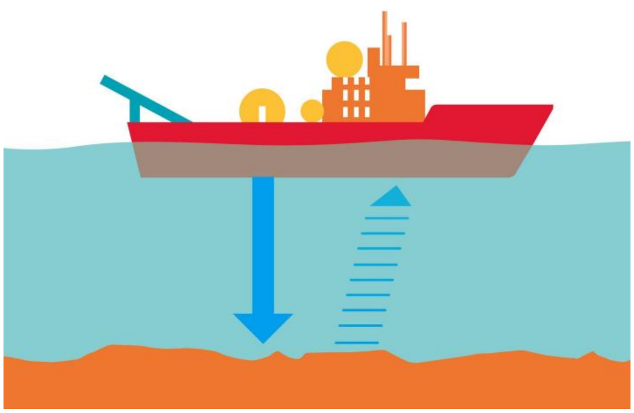
Test Result

Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. When different alcohol concentrations are detected, the active buzzer module will alarm, and the four-digit digital display will show the concentration value. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.3.12 Project 56: Ultrasonic Radar



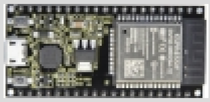
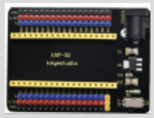
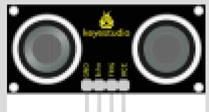
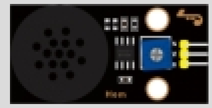
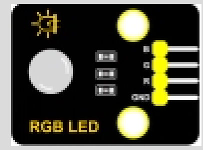
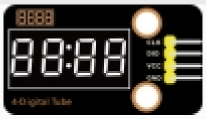
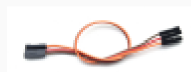
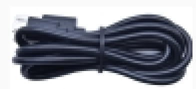
Description



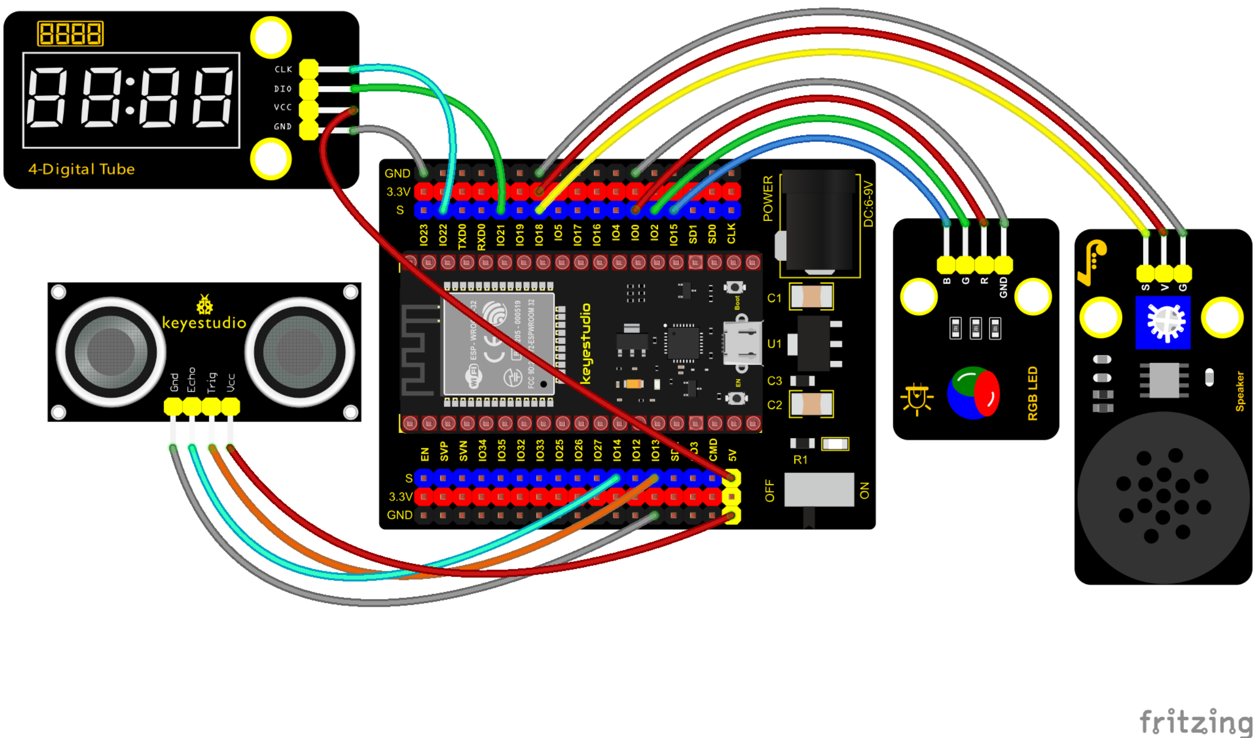
We know that bats use echoes to determine the direction and the location of their preys. In real life, sonar is used to detect sounds in the water. Since the attenuation rate of electromagnetic waves in water is very high, it cannot be used to detect signals, however, the attenuation rate of sound waves in the water is much smaller, so sound waves are most commonly used underwater for observation and measurement.

In this experiment, we will use a speaker module, an RGB module and a 4-digit tube display to make a device for detection through ultrasonic.

Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HC-SR04 Ultrasonic Sensor*1	Keyestudio 8002b Power Amplifier*1	Keyestudio DIY Common Cathode RGB Module *1
				
Keyestudio DIY TM1650 4-Digit Display*1	4P Dupont Wire*3	3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
from machine import Pin, PWM
import utime

# definitions for TM1650
ADDR_DIS = 0x48  #mode command
ADDR_KEY = 0x49  #read key value command
```

(continues on next page)

(continued from previous page)

```
# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTEST      = 7

on  = 1
off = 0

# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]

clkPin = 22
dioPin = 21
clk = Pin(clkPin, Pin.OUT)
dio = Pin(dioPin, Pin.OUT)

DisplayCommand = 0

def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        utime.sleep(0.0001)
        clk.value(1)
        utime.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return

def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    utime.sleep(0.0001)
    dio.value(0)
    return

def ack():
    global clk,dio
    dy = 0
    clk.value(0)
    utime.sleep(0.0001)
    dio = Pin(dioPin, Pin.IN)
    while(dio.value() == 1):
        utime.sleep(0.0001)
```

(continues on next page)

(continued from previous page)

```
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    utime.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, Pin.OUT)
    return

def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
    utime.sleep(0.0001)
    dio.value(1)
    return

def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return

def clearBit(bit):
    if(bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    writeByte(0x00)
    ack()
```

(continues on next page)

(continued from previous page)

```
    stop()
    return

    def setBrightness(b = BRIGHT_TYPICAL):
        global DisplayCommand,brightness
        DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
        return
    def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return

def displayOnOFF(OnOff = 1):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return

def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return

def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return

def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)
```

(continues on next page)

(continued from previous page)

```
pwm_r = PWM(Pin(0))
pwm_g = PWM(Pin(2))
pwm_b = PWM(Pin(15))

pwm_r.freq(1000)
pwm_g.freq(1000)
pwm_b.freq(1000)

def light(red, green, blue):
    pwm_r.duty(red)
    pwm_g.duty(green)
    pwm_b.duty(blue)

# Ultrasonic ranging, unit: cm
def getDistance(trigger, echo):
    # Generates a 10us square wave
    trigger.value(0)   #A short low level is given beforehand to ensure a clean high␣
↪pulse:
    utime.sleep_us(2)
    trigger.value(1)
    utime.sleep_us(10)#After pulling high, wait 10 microseconds and immediately set it␣
↪to low
    trigger.value(0)

    while echo.value() == 0: #Establish a while loop to detect whether the echo pin␣
↪value is 0 and record the time at that time
        start = utime.ticks_us()
    while echo.value() == 1: #Establish a while loop to check whether the echo pin value␣
↪is 1 and record the time at that time
        end = utime.ticks_us()
    d = (end - start) * 0.0343 / 2 #The travel time of the sound wave x the speed of␣
↪sound (343.2 m/s, 0.0343 cm/microsecond), and the distance back and forth divided by 2.
    return d

# set the pin
trigger = Pin(13, Pin.OUT)
echo = Pin(14, Pin.IN)

buzzer = PWM(Pin(18))
def playtone(frequency):
    buzzer.duty(1000)
    buzzer.freq(frequency)

def bequiet():
    buzzer.duty(0)

# main program
InitDigitalTube()
while True:
    distance = int(getDistance(trigger, echo))
    ShowNum(distance)
```

(continues on next page)

(continued from previous page)

```
    if distance <= 10:
        playtone(880)
        utime.sleep(0.1)
        bequiet()
        light(1023, 0, 0)
    elif distance <= 20:
        playtone(532)
        utime.sleep(0.2)
        bequiet()
        light(0, 0, 1023)
    else:
        light(0, 1023, 0)
```

Code Explanation

We set sound frequency and light color by adjusting different distance range.

We can adjust the distance range in the code.

Test Result

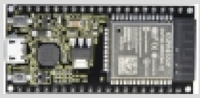
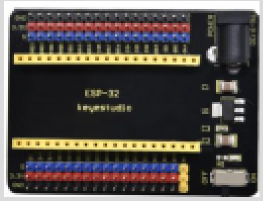
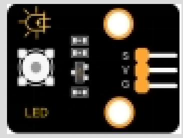
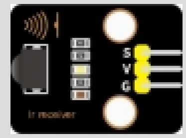
Connect the wires according to the experimental wiring diagram and power on. Click “Run current script”, the code starts executing. When the ultrasonic sensor detects different distances, the buzzer will produce different frequencies of sound(within 20 cm), the RGB will show different colors, and the measured distances are displayed on the 4-digit tube display. Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.3.13 Project 57: IR Remote Control

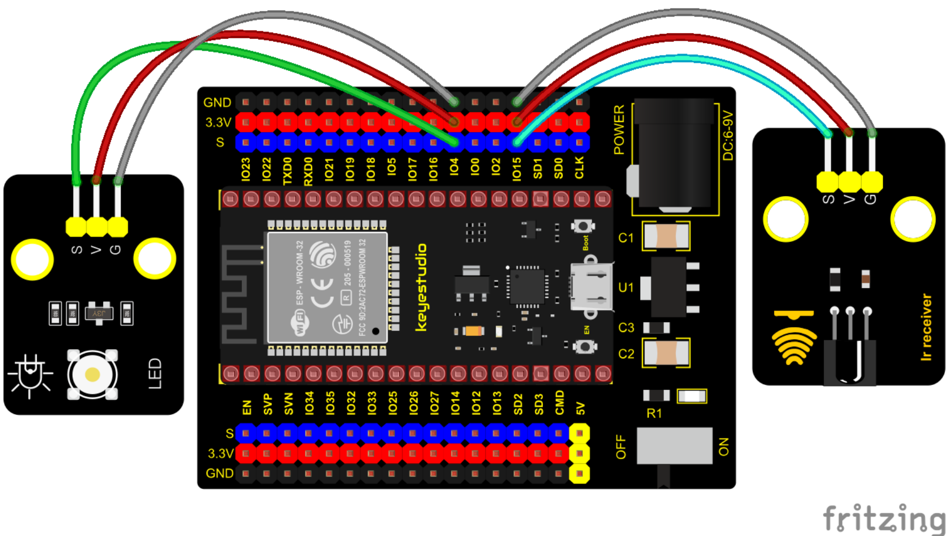
Introduction

In the previous experiments, we learned how to turn on/off the LED and adjust its brightness via PWM and print the button value of the IR remote control in the Shell window. Herein, we use an infrared remote control to turn on/off an LED.

Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Purple LED Module*1	Keyestudio DIY IR Receiver*1
			
Micro USB Cable*1	IR Remote Control*1	3P Dupont Wire*2	

Connection Diagram



Test Code

```
import time
from machine import Pin

led = Pin(4, Pin.OUT)
ird = Pin(15,Pin.IN)

act = {"1": "LLLLLLLLHHHHHHHHLHHLHLLLHLLHLHHH","2": "LLLLLLLLHHHHHHHHHLLHHLLLLHHLLHHH","3
↪": "LLLLLLLLHHHHHHHHHLHHLLLLLHLLHHHH",
       "4": "LLLLLLLLHHHHHHHHLLHHLLLLHHLLHHHH","5": "LLLLLLLLHHHHHHHHLLLHHLLLHHHLLHHH","6
↪": "LLLLLLLLHHHHHHHHLHHHHLHLHLLLLHLH",
       "7": "LLLLLLLLHHHHHHHHLLLHLLLLHHHLHHHH","8": "LLLLLLLLHHHHHHHHLLHHHLLLHHLLLHHH","9
↪": "LLLLLLLLHHHHHHHHLHLHHLHLHLHLLHLH",
       "0": "LLLLLLLLHHHHHHHHLHLLHLHLHLHHLHLH","Up": "LLLLLLLLHHHHHHHHLHHLLLHLHLLHHHLH",
↪"Down": "LLLLLLLLHHHHHHHHHLHLHLLLLHLHLHHH",
```

(continues on next page)

(continued from previous page)

```
        "Left": "LLLLLLLLHHHHHHHHLLHLLLHLHHLHHHLH","Right":
↪"LLLLLLLLHHHHHHHHHHLLLLHLLLHHHHLH","Ok": "LLLLLLLLHHHHHHHHLLLLLLHLHHHHHHLH",
        "*": "LLLLLLLLHHHHHHHHLHLLLLHLHLHHHHLH","#": "LLLLLLLLHHHHHHHHLHLHLLHLHLHLHHLH"}

def read_ircode(ird):
    wait = 1
    complete = 0
    seq0 = []
    seq1 = []

    while wait == 1:
        if ird.value() == 0:
            wait = 0
    while wait == 0 and complete == 0:
        start = time.ticks_us()
        while ird.value() == 0:
            ms1 = time.ticks_us()
        diff = time.ticks_diff(ms1,start)
        seq0.append(diff)
        while ird.value() == 1 and complete == 0:
            ms2 = time.ticks_us()
            diff = time.ticks_diff(ms2,ms1)
            if diff > 10000:
                complete = 1
        seq1.append(diff)

    code = ""
    for val in seq1:
        if val < 2000:
            if val < 700:
                code += "L"
            else:
                code += "H"
    # print(code)
    command = ""
    for k,v in act.items():
        if code == v:
            command = k
    if command == "":
        command = code
    return command

flag = False
while True:
#     global flag
    command = read_ircode(ird)
    print(command, end = "   ")
    print(flag, end = "   ")
    if command == "Ok":
        if flag == True:
            led.value(1)
            flag = False
```

(continues on next page)

(continued from previous page)

```
            print("led on")
        else:
            led.value(0)
            flag = True
            print("led off")
    time.sleep(0.1)
```

Code Explanation

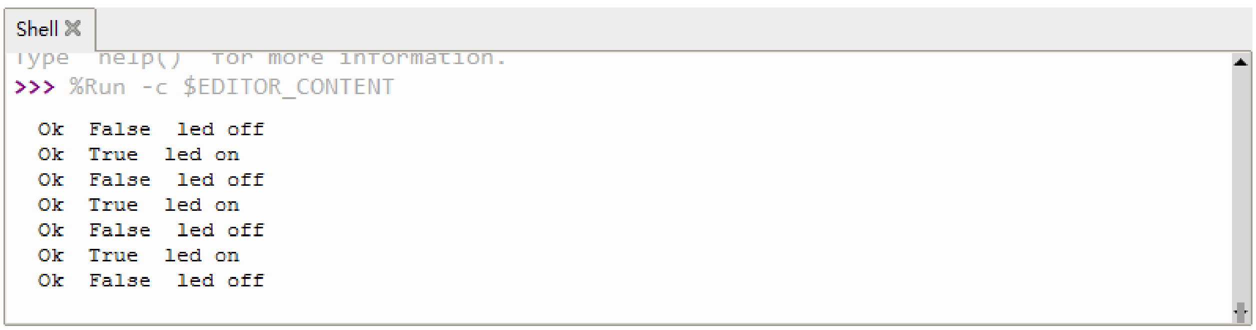
We define a boolean variable. There are two boolean variables. true (true) or false (false).

When we press the OK button, the value of infrared reception is OK. At this time, we need to set a boolean variable flag. When the flag is true (true), the LED is turned on, and when it is false (false), the LED is turned off and turned on. After the LED is on and set it to false. We press the OK key, the LED will be off.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. Press the OK button of the remote, the LED will be on, press it again, the LED will be off.

Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

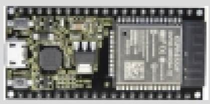
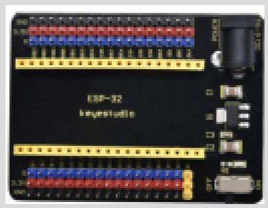
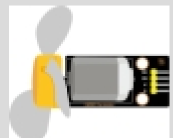
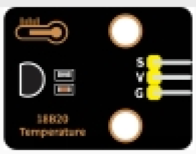
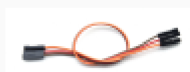
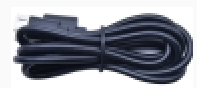


5.[illegible].14 Project 58: Heat Dissipation Device

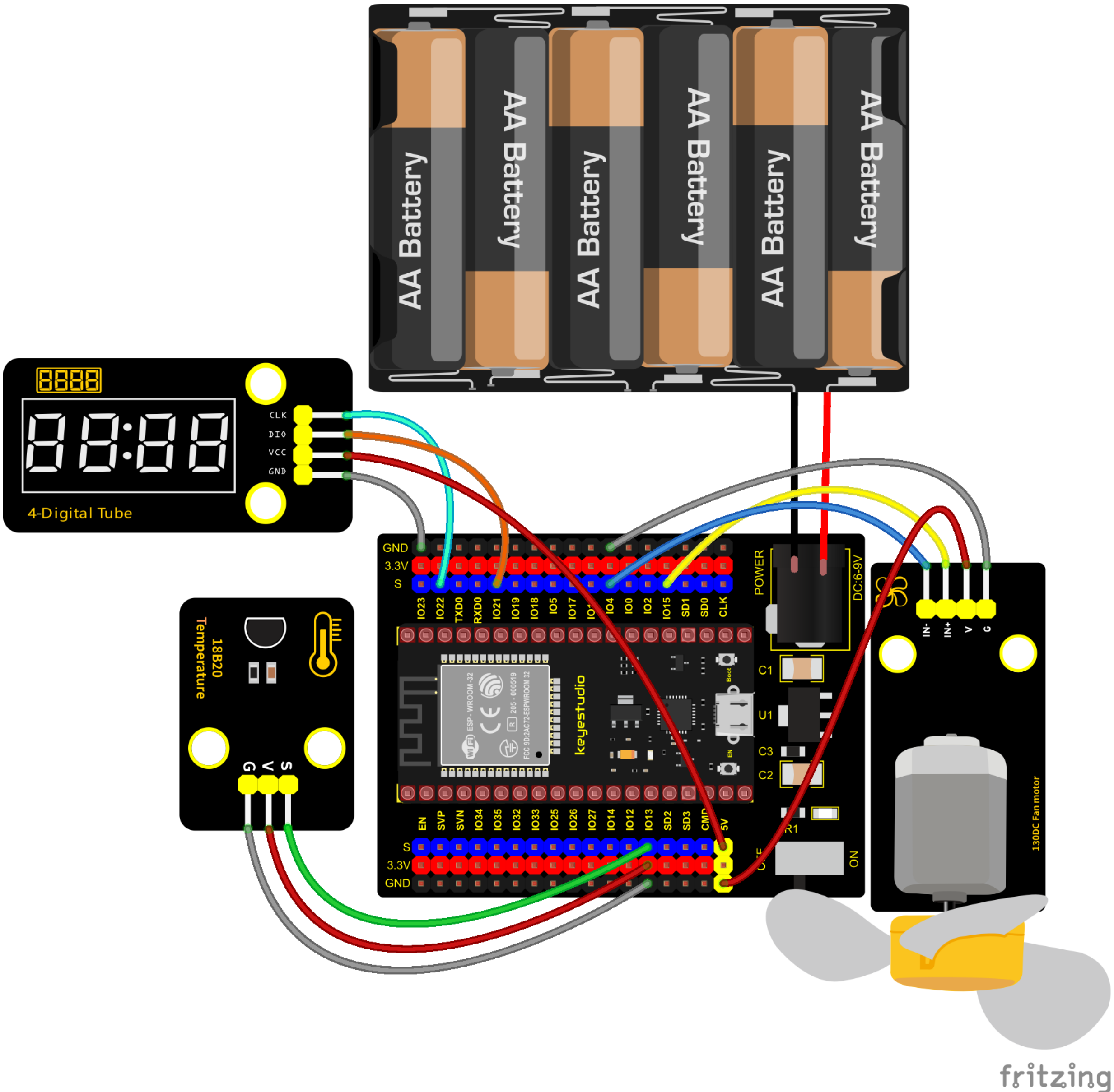
Description

We will use a temperature sensor and some modules to make a smart cooling device in this experiment. When the ambient temperature is higher than a certain value, the motor is turned on, thereby reducing the ambient temperature and achieving the heat dissipation effect. Then display the temperature value in the four-digit segment display.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 130 Motor*1	TM1650 4-Digit Segment Display*1
			
Keyestudio 18B20 Temperature Sensor*1	3P Dupont Wire*1	4P Dupont Wire*2	Micro USB Cable*1
			
Battery Holder*1	Battery(provide for yourself)*6		

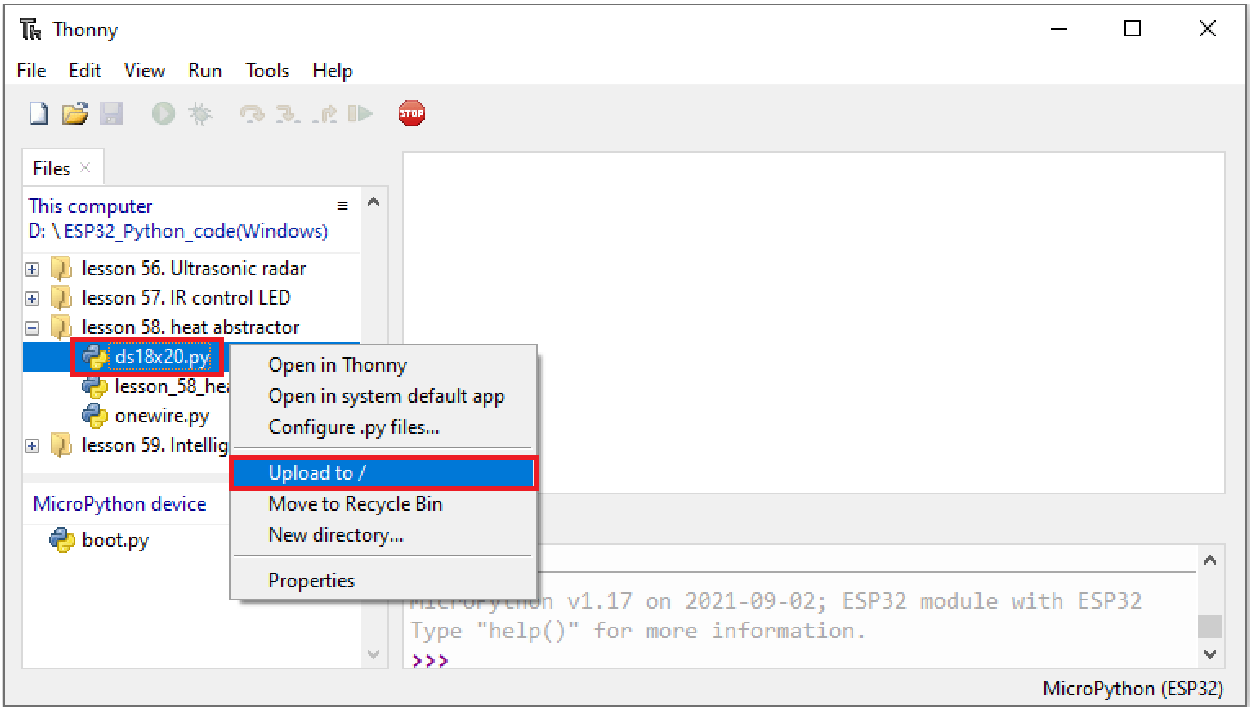
Connection Diagram

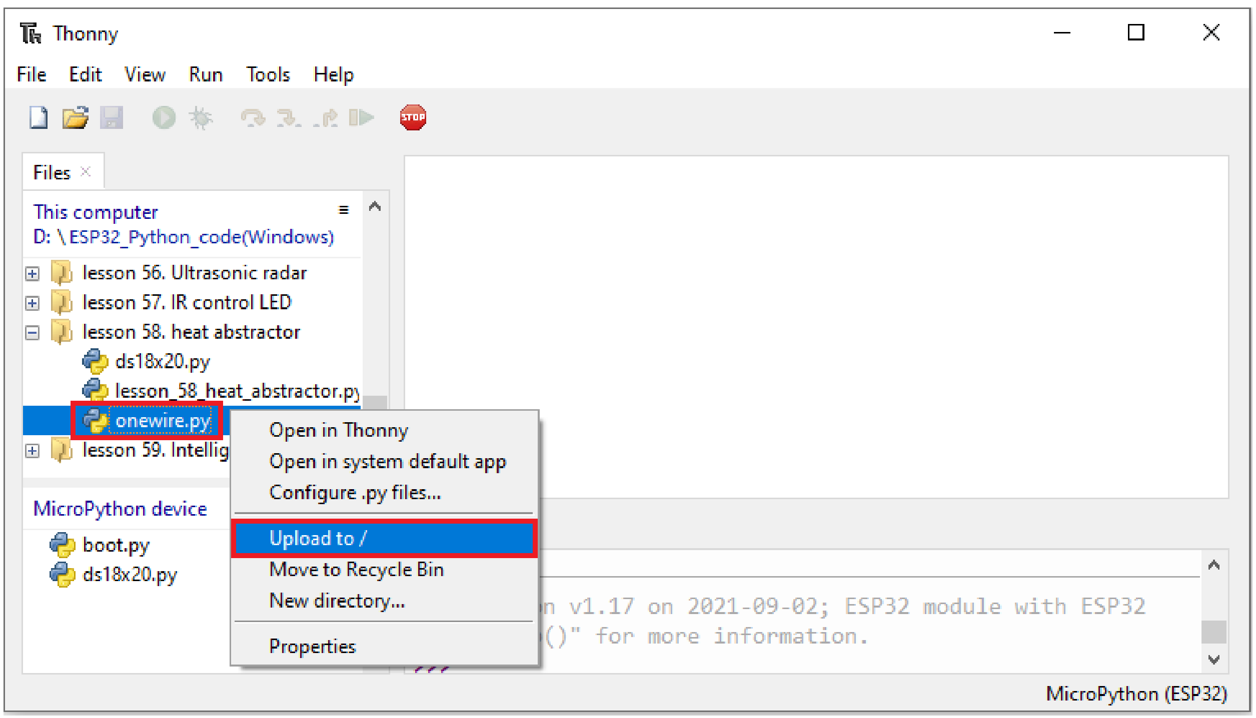


Add Library

Open"Thonny", click"This computer"→"D:"→"2. ESP32_code_MicroPython"→"lesson 58. heat abstractor".

Select"ds18x20.py"and"ds18x20.py"right-click and select "Upload to/"waiting for the "ds18x20.py"and"ds18x20.py"to be uploaded to the ESP32.





Test Code

```
from machine import Pin
import machine, onewire, ds18x20, time

ds_pin = machine.Pin(13)

ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
```

(continues on next page)

(continued from previous page)

```
roms = ds_sensor.scan()

#Two pins of the motor
INA = Pin(15, Pin.OUT) #INA corresponds to IN+
INB = Pin(4, Pin.OUT)#INB corresponds to IN-
# definitions for TM1650
ADDR_DIS = 0x48  #mode command
ADDR_KEY = 0x49  #read key value command

# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTEST      = 7

on  = 1
off = 0

# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]

clkPin = 22
dioPin = 21
clk = Pin(clkPin, Pin.OUT)
dio = Pin(dioPin, Pin.OUT)

DisplayCommand = 0

def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        time.sleep(0.0001)
        clk.value(1)
        time.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return

def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(0)
```

(continues on next page)

(continued from previous page)

```
    return

def ack():
    global clk,dio
    dy = 0
    clk.value(0)
    time.sleep(0.0001)
    dio = Pin(dioPin, Pin.IN)
    while(dio.value() == 1):
        time.sleep(0.0001)
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    time.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, Pin.OUT)
    return

def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
    time.sleep(0.0001)
    dio.value(1)
    return

def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return

def clearBit(bit):
    if(bit > 4):
        return
    start()
```

(continues on next page)

(continued from previous page)

```
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    writeByte(0x00)
    ack()
    stop()
    return

    def setBrightness(b = BRIGHT_TYPICAL):
        global DisplayCommand,brightness
        DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
        return
    def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return

def displayOnOFF(OnOff = 1):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return

def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return

def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return

def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
```

(continues on next page)

(continued from previous page)

```
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)

InitDigitalTube()
print('Found DS devices: ', roms)

while True:
    ds_sensor.convert_temp()
    time.sleep_ms(750)
    for rom in roms:
        value = ds_sensor.read_temp(rom)
        print(value)
        ShowNum(int(value))
        if value > 28:
            INA.value(0)
            INB.value(1)
        else:
            INA.value(0)
            INB.value(0)
```

Code Explanation

The setting of variables and the storage of detection values are the same as what we learned earlier. We also set a temperature threshold and control the rotation of the motor when the threshold is exceeded, and then we use the digital tube to display the temperature value.

Test Result

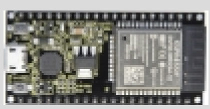
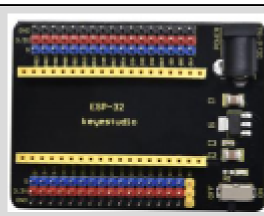
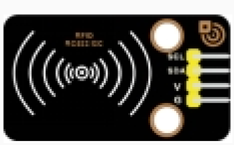
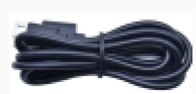
Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch on the ESP32 expansion board to the ON end. Click "Run current script", the code starts executing. We can see the temperature of the current environment (unit is Celsius) on the four-digit segment display, as shown in the figure below. If this value exceeds the value we set, the fan will rotate to dissipate heat. Press"Ctrl+C"or click "Stop/Restart backend"to exit the program.

5.3.15 Project 59: Intelligent Entrance Guard System

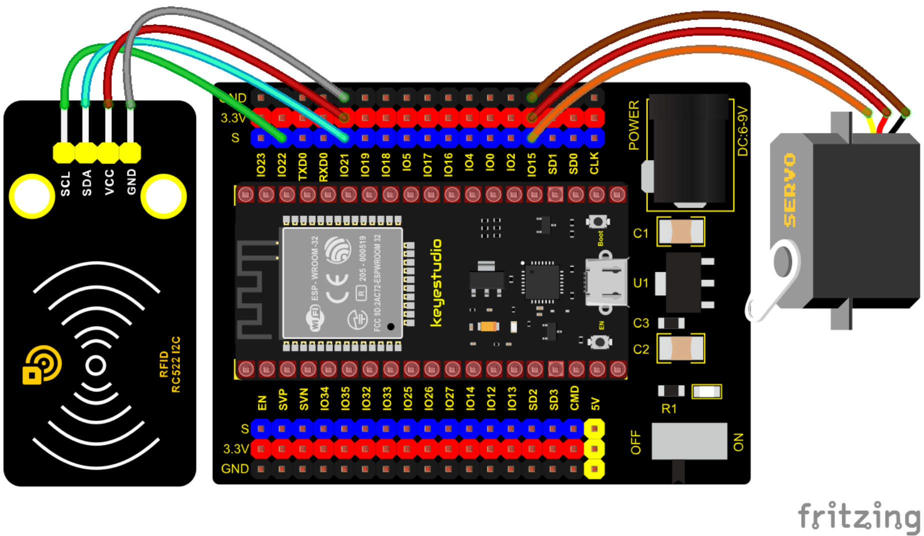
Description

In this project, we use the RFID522 card swiping module and the servo to set up an intelligent access control system. The principle is very simple.We use RFID522 swipe card module, an IC card or key card to unlock.

Required Components

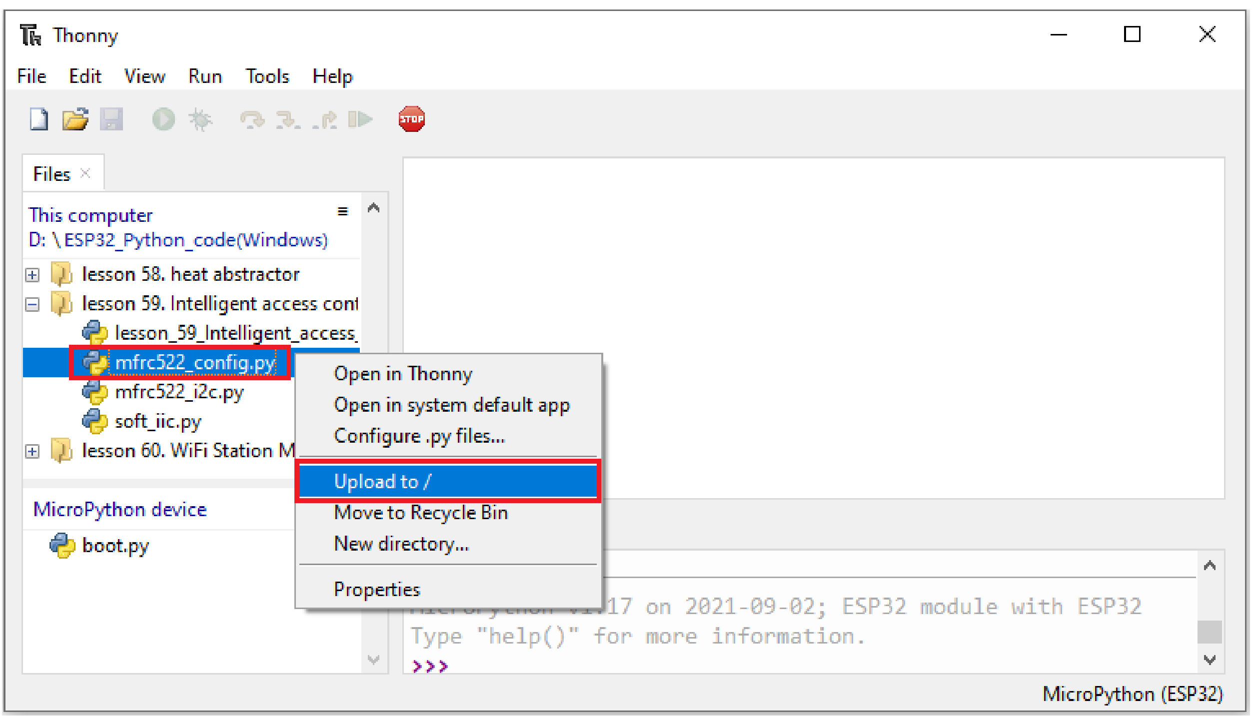
			
ESP32 Board*1	ESP32 Expansion Board*1	Key*1	IC Card*1
			
Keyestudio RFID Module*1	Servo*1	4P Dupont Wire*1	Micro USB Cable*1

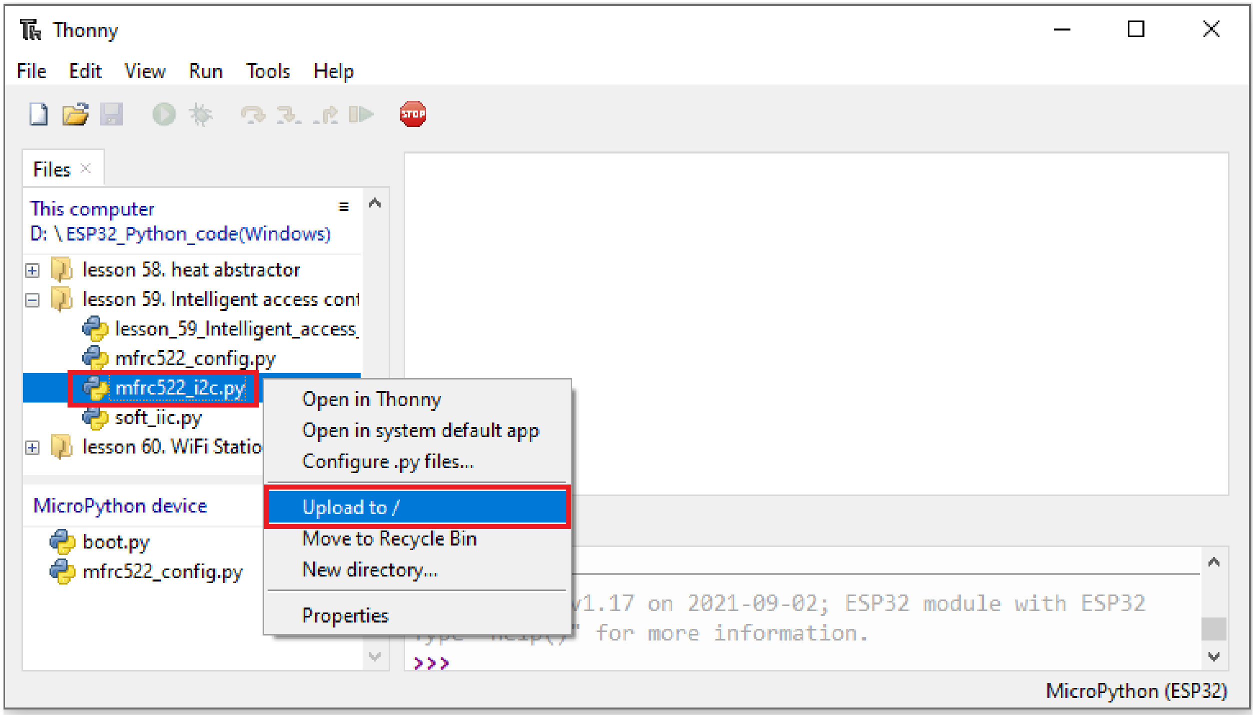
Connection Diagram

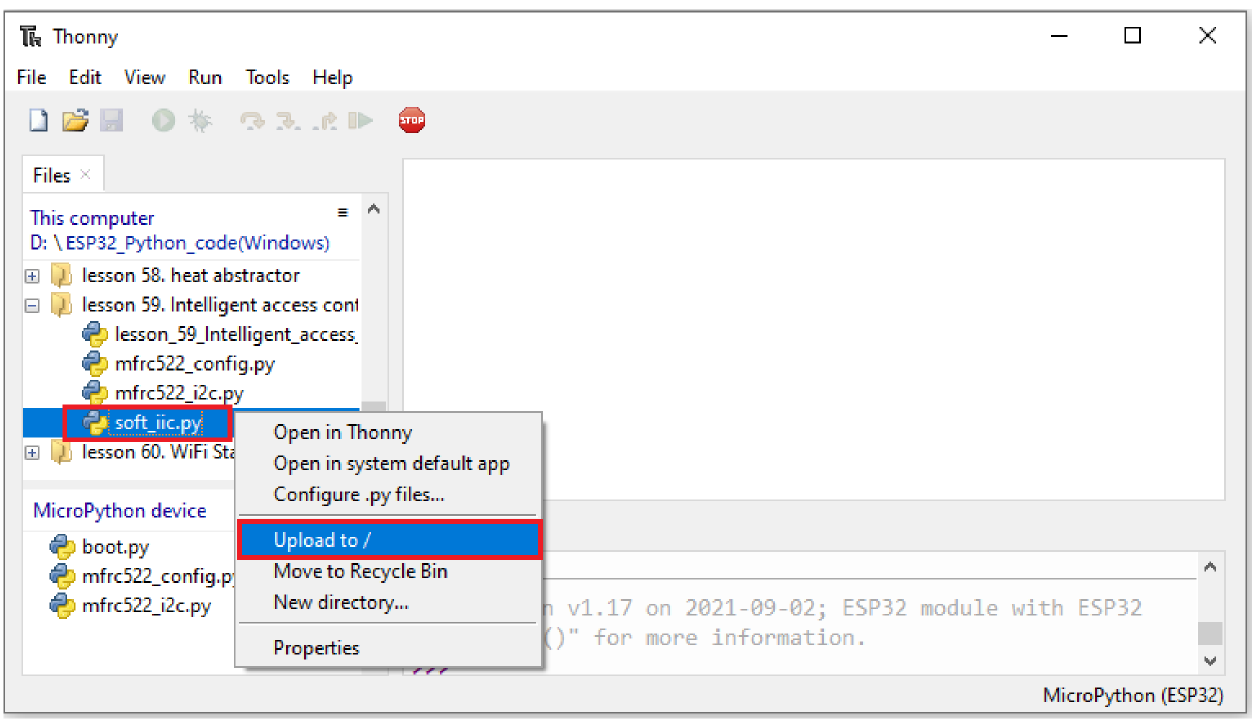


Add Library

Open "Thonny", click "This computer" → "D:" → "2. ESP32_code_MicroPython" → "lesson 59. Intelligent access control". Select "mfrc522_config.py", "mfrc522_i2c.py" and "soft_iic.py", right-click and select "Upload to /", waiting for the "mfrc522_config.py", "mfrc522_i2c.py" and "soft_iic.py" to be uploaded to the ESP32.







Test Code

Note: Different RFID-RC522 modules, ID cards and keys may different uid1 values and uid2 values.

The uID1 and UID2 values of the white card and key chain read by your RRFID RC522 module can be replaced by the corresponding values in the program code. If not, click **"Run current script"** to run the code may cause your own white card and key chain to fail to control the servo.

For example: You can replace the UID1 and UID2 values In the program code

uid1 = [237, 247, 148, 90]
uid2 = [76, 9, 107, 110] with your own white card and key chain values.

```
from machine import Pin, PWM
import time
from mfrc522_i2c import mfrc522

pwm = PWM(Pin(15))
pwm.freq(50)

'''
Duty cycle corresponding to the Angle
Duty cycle corresponding to the Angle
0°----2.5%----25
45°----5%----51.2
90°----7.5%----77
135°----10%----102.4
180°----12.5%----128
'''
angle_0 = 25
angle_90 = 77
angle_180 = 128

#i2c config
addr = 0x28
scl = 22
sda = 21

rc522 = mfrc522(scl, sda, addr)
rc522.PCD_Init()
rc522.ShowReaderDetails()            # Show details of PCD - MFRC522 Card Reader details

uid1 = [237, 247, 148, 90]
uid2 = [76, 9, 107, 110]

pwm.duty(angle_180)
time.sleep(1)

while True:
    if rc522.PICC_IsNewCardPresent():
        print("Is new card present!")
        if rc522.PICC_ReadCardSerial() == True:
            print("Card UID:", end=' ')
            print(rc522.uid.uidByte[0 : rc522.uid.size])
            if rc522.uid.uidByte[0 : rc522.uid.size] == uid1 or rc522.uid.uidByte[0 :
↪rc522.uid.size] == uid2:
                pwm.duty(angle_0)
            else :
                pwm.duty(angle_180)
            time.sleep(500)from machine import Pin, PWM
import time
from mfrc522_i2c import mfrc522
```

(continues on next page)

(continued from previous page)

```
pwm = PWM(Pin(15))
pwm.freq(50)

'''
Duty cycle corresponding to the Angle
Duty cycle corresponding to the Angle
0°----2.5%----25
45°----5%----51.2
90°----7.5%----77
135°----10%----102.4
180°----12.5%----128
'''
angle_0 = 25
angle_90 = 77
angle_180 = 128

#i2c config
addr = 0x28
scl = 22
sda = 21

rc522 = mfrc522(scl, sda, addr)
rc522.PCD_Init()
rc522.ShowReaderDetails()           # Show details of PCD - MFRC522 Card Reader details

uid1 = [237, 247, 148, 90]
uid2 = [76, 9, 107, 110]

pwm.duty(angle_180)
time.sleep(1)

while True:
    if rc522.PICC_IsNewCardPresent():
        print("Is new card present!")
        if rc522.PICC_ReadCardSerial() == True:
            print("Card UID:", end=' ')
            print(rc522.uid.uidByte[0 : rc522.uid.size])
            if rc522.uid.uidByte[0 : rc522.uid.size] == uid1 or rc522.uid.uidByte[0 : rc522.uid.size] == uid2:
                pwm.duty(angle_0)
            else :
                pwm.duty(angle_180)
            time.sleep(500)
```

Code Explanation

In the previous experiment, our card swipe module has tested the information of IC card and key. Then we use this corresponding information to control the door.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Click "Run current script", the code starts executing. When we use the IC card or blue key to swipe the card, the shell displays the card and the key information , at the same time, the servo rotates to the corresponding angle to simulate opening the door.

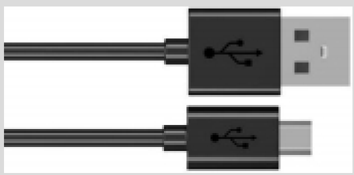
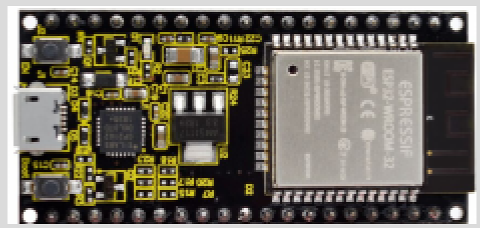
Press “Ctrl+C”or click “Stop/Restart backend”to exit the program.

5.3.16 Project 60WIFI Station Mode

Description

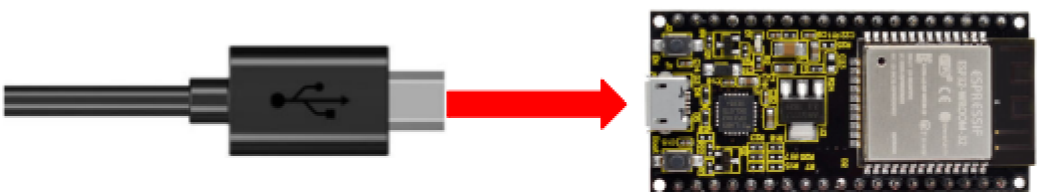
ESP32 has three different WiFi modes: Station mode, AP mode and AP+Station mode. All WiFi programming projects must be configured with WiFi running mode before using, otherwise the WiFi cannot be used. In this project, we are going to learn the WiFi Station mode of the ESP32.

Components

	
USB Cable x1	ESP32*1

Wiring Diagram

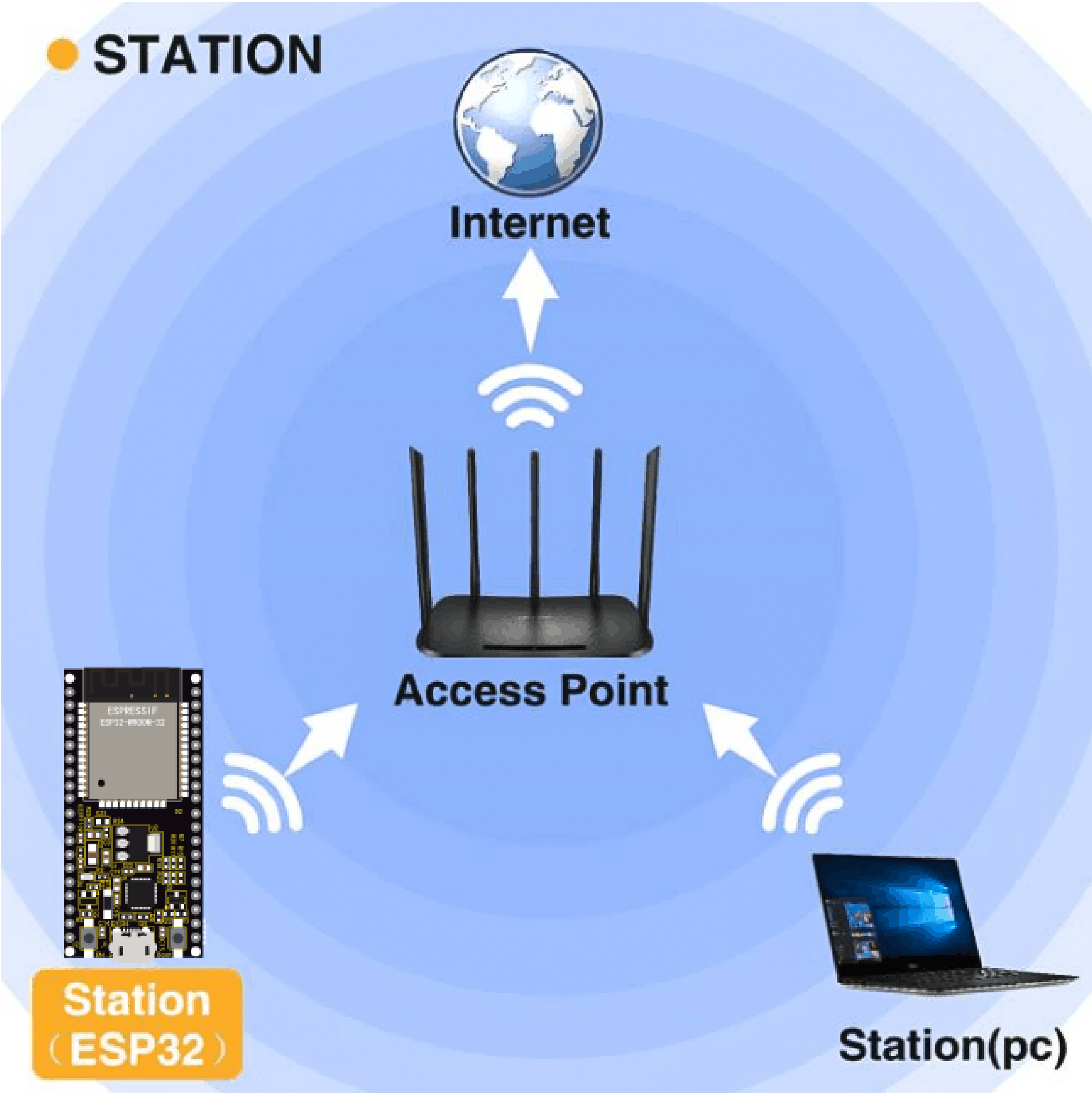
Plug the ESP32 to the USB port of your PC



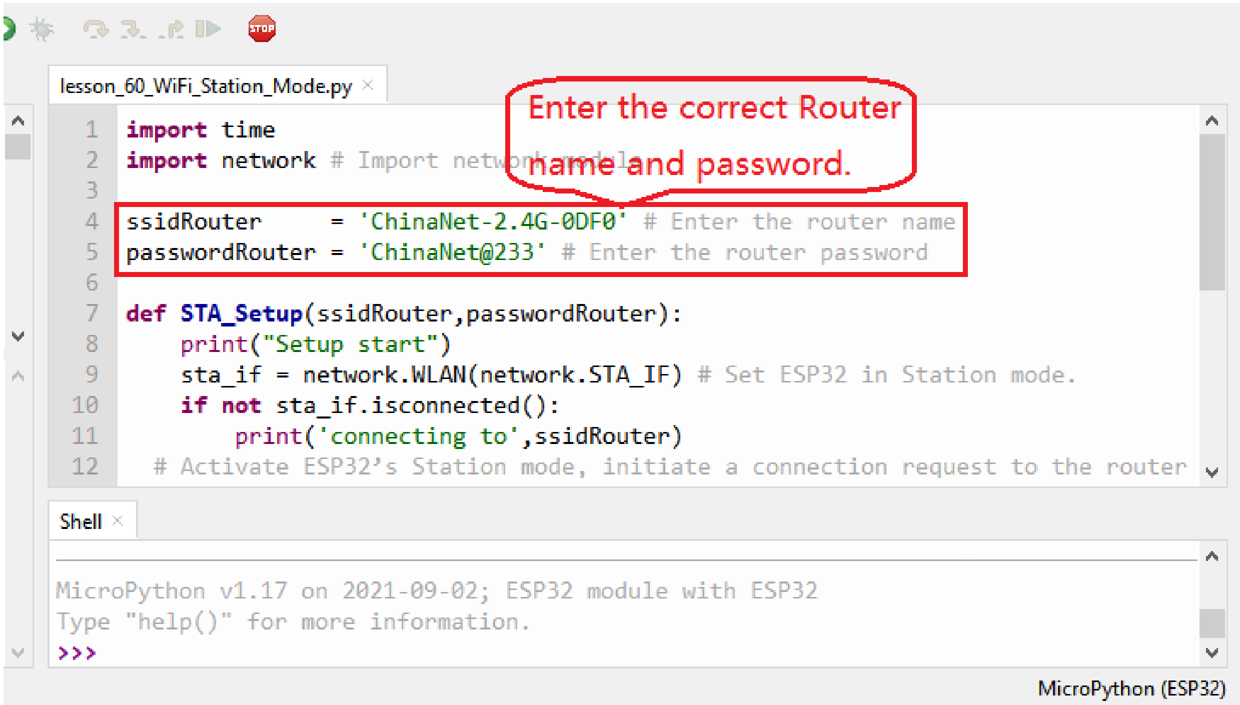
Component Knowledge

Station mode

When setting Station mode, the ESP32 is taken as a WiFi client. It can connect to the router network and communicate with other devices on the router via a WiFi connection. As shown in the figure below, the PC and the router have been connected. If the ESP32 wants to communicate with the PC, the PC and the router need to be connected.



Test Code



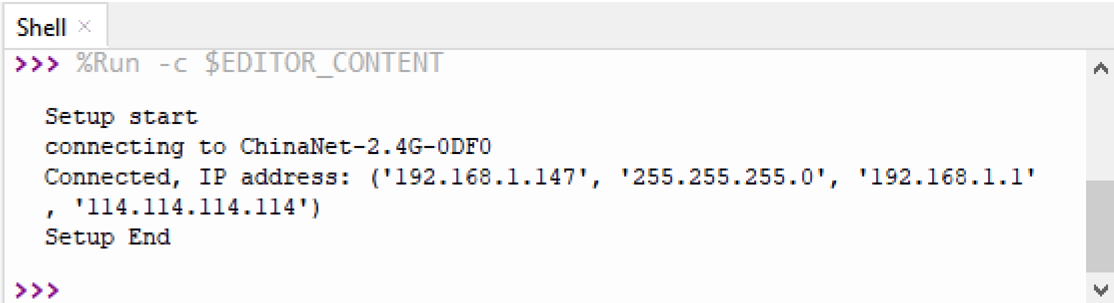

```python
import time
import network # Import network module.

ssidRouter     = 'ChinaNet-2.4G-0DF0' # Enter the router name
passwordRouter = 'ChinaNet@233' # Enter the router password

def STA_Setup(ssidRouter,passwordRouter):
    print("Setup start")
    sta_if = network.WLAN(network.STA_IF) # Set ESP32 in Station mode.
    if not sta_if.isconnected():
        print('connecting to',ssidRouter)
 # Activate ESP32's Station mode, initiate a connection request to the router and enter the password to connect.
        sta_if.active(True)
        sta_if.connect(ssidRouter,passwordRouter)
  #Wait for ESP32 to connect to router until they connect to each other successfully.
        while not sta_if.isconnected():
            pass
  # Print the IP address assigned to ESP32-WROVER in “Shell”.
    print('Connected, IP address:', sta_if.ifconfig())
    print("Setup End")

try:
    STA_Setup(ssidRouter,passwordRouter)
except:
    sta_if.disconnect()
```

Test Result

Since the router name and password are different in various places, so before running the code, the user needs to enter the correct router name and password in the red box shown above.

After entering the correct router name and password, click “Run current script”, the code will start executing.

The Shell monitor will print the IP address of the ESP32 when connecting the ESP32 to your router.

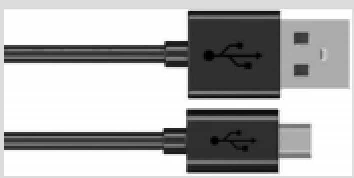
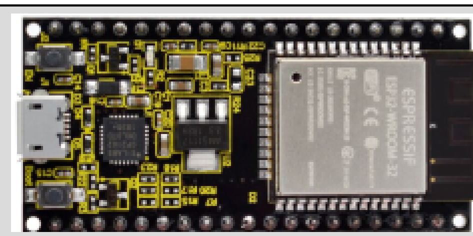


5.3.17 Project 61WIFI AP Mode

Description

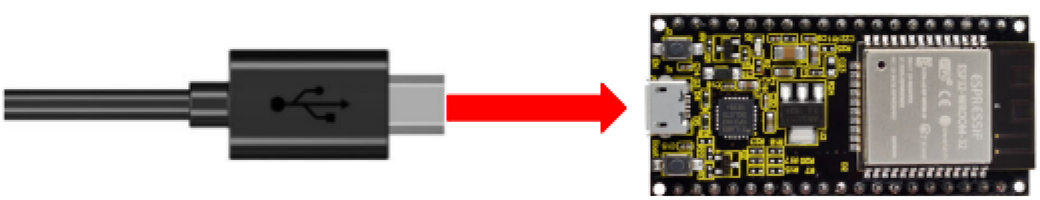
In this project, we are going to learn the WiFi AP mode of the ESP32.

Components

	
USB Cable x1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your PC



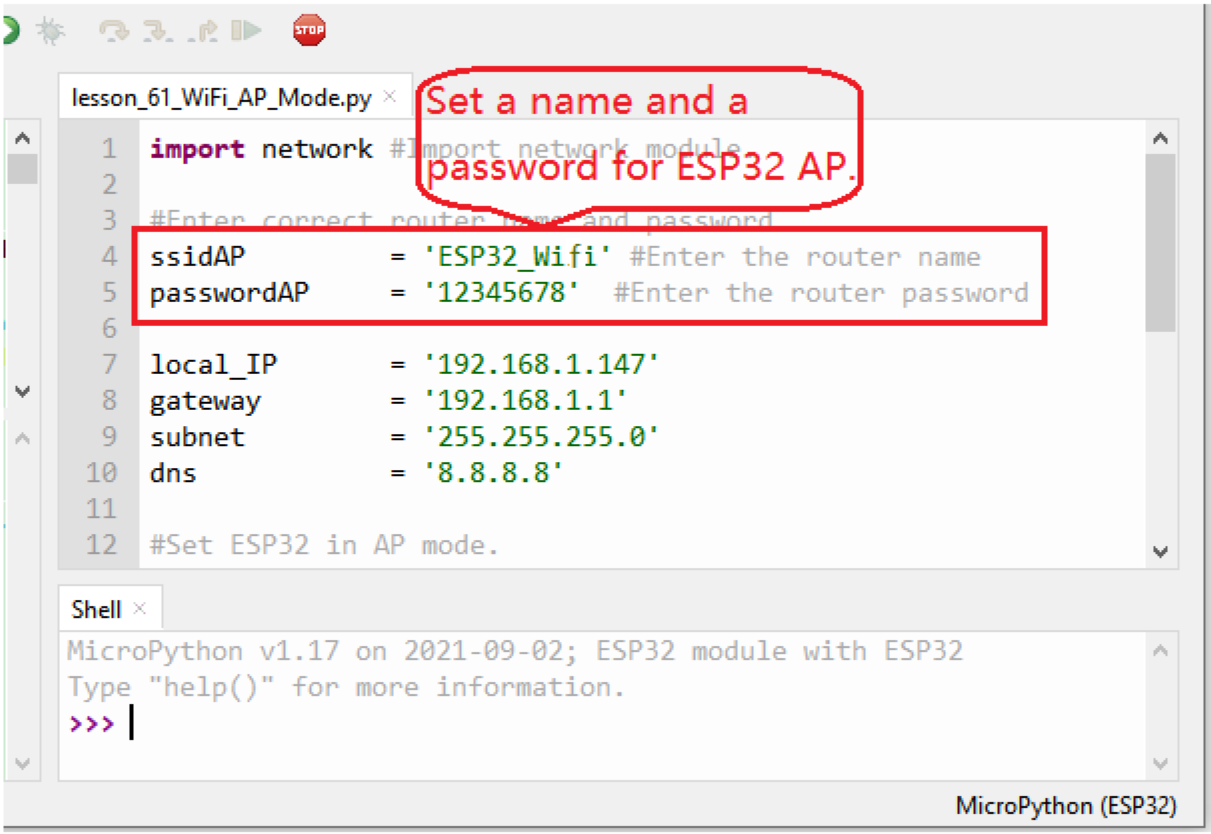
Component Knowledge

AP Mode:

When setting AP mode, a hotspot network will be created, waiting for other WiFi devices to connect. As shown below;

Take the ESP32 as the hotspot, if a phone or PC needs to communicate with the ESP32, it must be connected to the ESP32’s hotspot. Communication is only possible after a connection is established via the ESP32.

Test Code




```python
import network #Import network module.

#Enter correct router name and password.
ssidAP          = 'ESP32_Wifi' #Enter the router name
passwordAP      = '12345678'  #Enter the router password

local_IP        = '192.168.1.147'
gateway         = '192.168.1.1'
subnet          = '255.255.255.0'
dns             = '8.8.8.8'

#Set ESP32 in AP mode.
ap_if = network.WLAN(network.AP_IF)

def AP_Setup(ssidAP,passwordAP):
    ap_if.ifconfig([local_IP,gateway,subnet,dns])
    print("Setting soft-AP  ... ")
    ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
    ap_if.active(True)
    print('Success, IP address:', ap_if.ifconfig())
    print("Setup End\n")

try:
    AP_Setup(ssidAP,passwordAP)
except:
    print("Failed, please disconnect the power and restart the operation.")
```

(continues on next page)

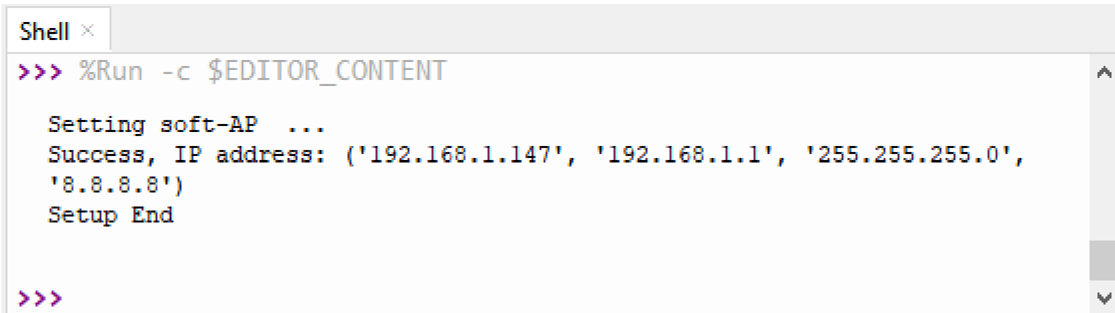
(continued from previous page)

```
    ap_if.disconnect()
```

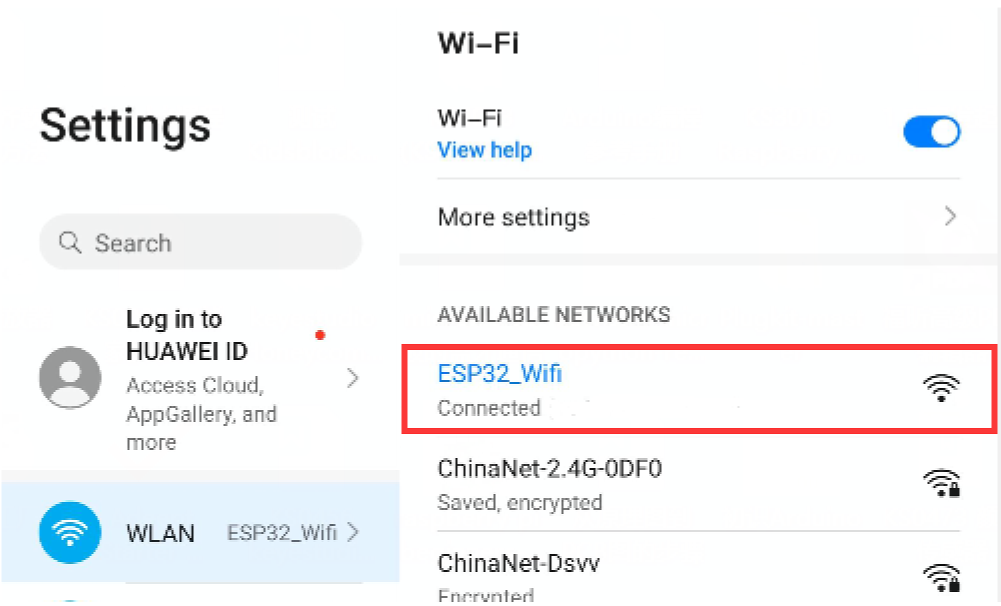
Test Result

You can modify the AP name and password or keep them unchanged.

Click "Run current script", the code will start executing. Open the AP function of the ESP32, the Shell monitor will print the information



Turn on your phone's WiFi search function, then you can see the ssid_AP which is called "ESP32_Wifi" in this code. You can enter the password "12345678" to connect it, or you can modify its AP name and password by code.

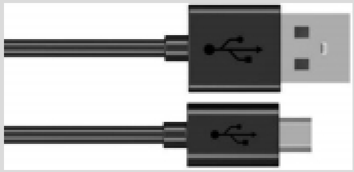
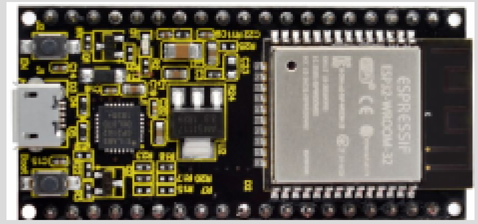


5.3.18 Project 62WIFI AP+Station Mode

Description

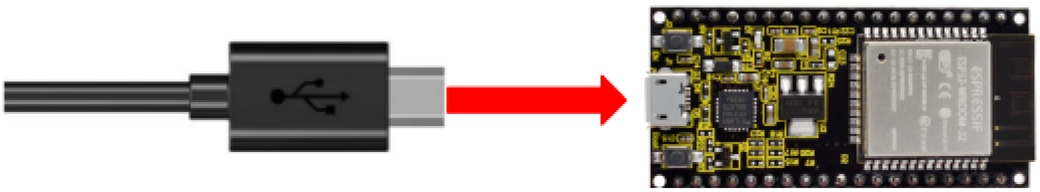
In this project, we are going to learn the AP+Station mode of the ESP32.

Components

	
USB Cable x1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your PC

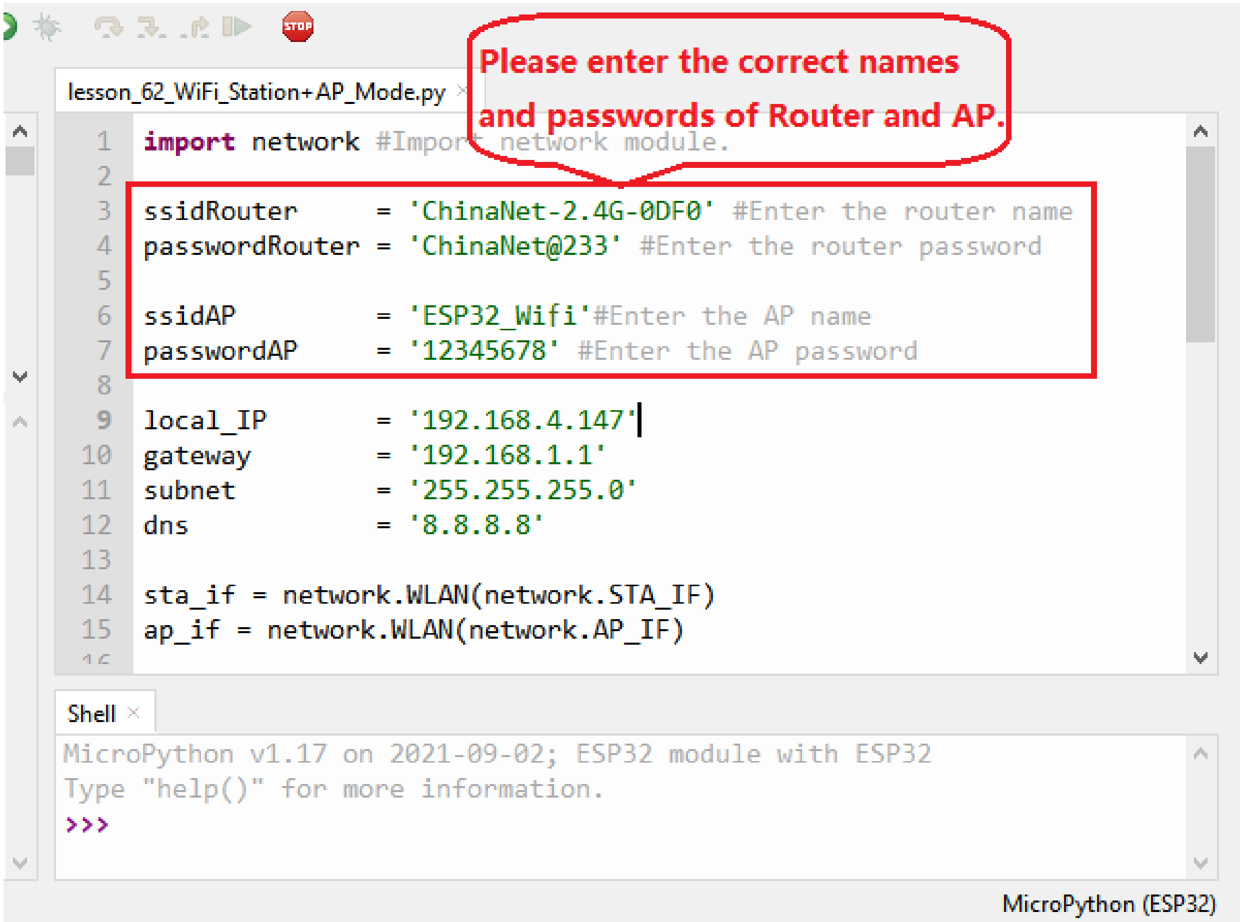


Component Knowledge

AP+Station mode

In addition to the AP mode and the Station mode, **AP+Station mode** can be used at the same time. Turn on the Station mode of the ESP32, connect it to the router network, and it can communicate with the Internet through the router. Then turn on the AP mode to create a hotspot network. Other WiFi devices can be connected to the router network or the hotspot network to communicate with the ESP32.

Test Code




```python
import network #Import network module.

ssidRouter     = 'ChinaNet-2.4G-0DF0' #Enter the router name
passwordRouter = 'ChinaNet@233' #Enter the router password

ssidAP         = 'ESP32_Wifi'#Enter the AP name
passwordAP     = '12345678' #Enter the AP password

local_IP       = '192.168.4.147'
gateway        = '192.168.1.1'
subnet         = '255.255.255.0'
dns            = '8.8.8.8'

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

def STA_Setup(ssidRouter,passwordRouter):
    print("Setting soft-STA  ... ")
    if not sta_if.isconnected():
        print('connecting to',ssidRouter)
        sta_if.active(True)
        sta_if.connect(ssidRouter,passwordRouter)
        while not sta_if.isconnected():
```

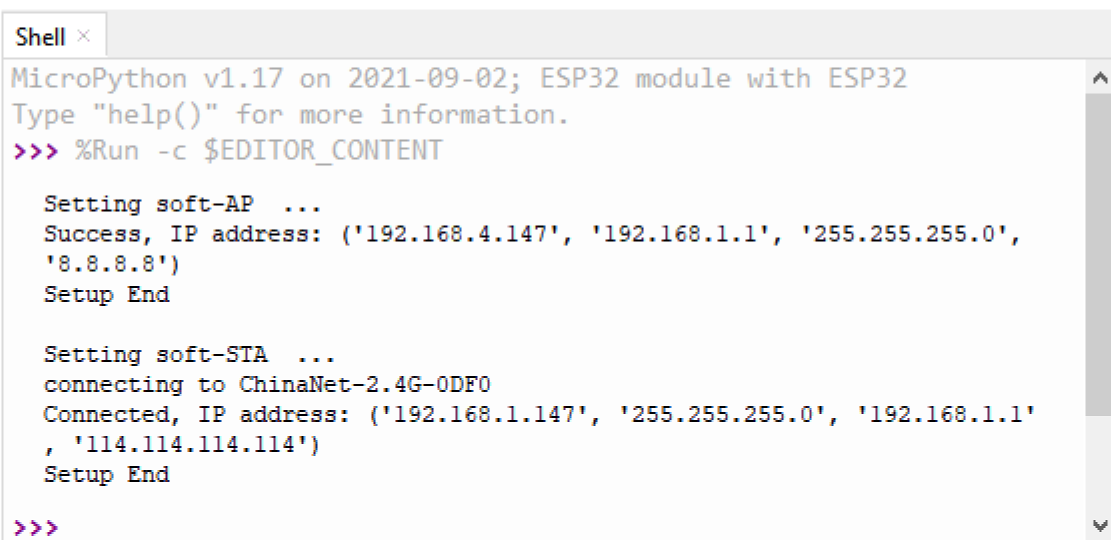
(continues on next page)

(continued from previous page)

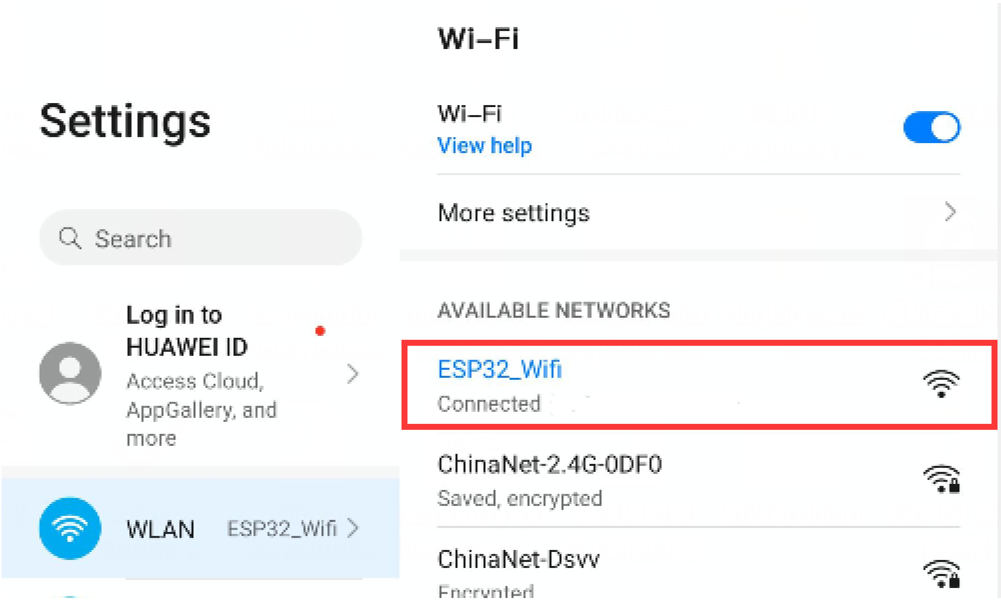
```
            pass
    print('Connected, IP address:', sta_if.ifconfig())
    print("Setup End")

def AP_Setup(ssidAP,passwordAP):
    ap_if.ifconfig([local_IP,gateway,subnet,dns])
    print("Setting soft-AP  ... ")
    ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
    ap_if.active(True)
    print('Success, IP address:', ap_if.ifconfig())
    print("Setup End\n")

try:
    AP_Setup(ssidAP,passwordAP)
    STA_Setup(ssidRouter,passwordRouter)
except:
    sta_if.disconnect()
    ap_if.idsconnect()
```

Test Result

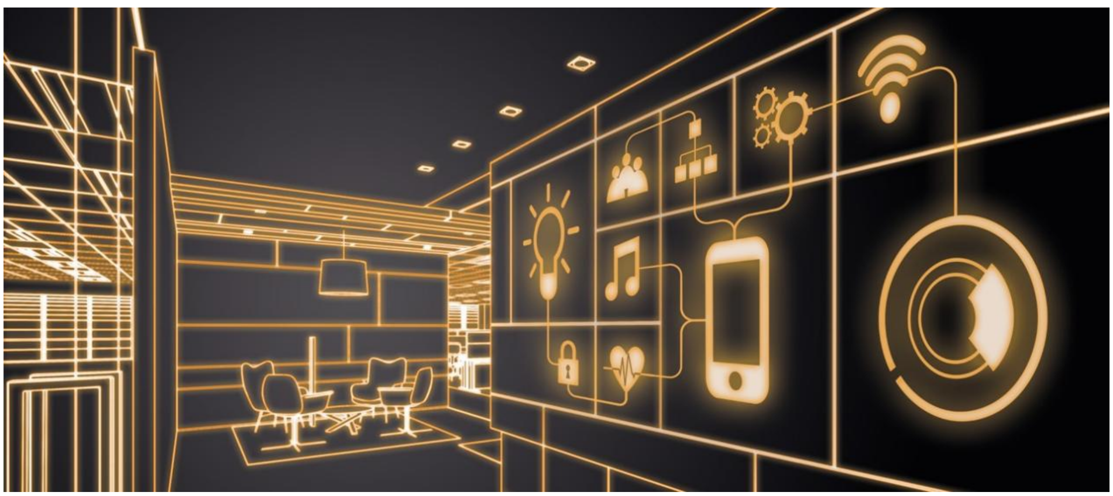
Before running the code, you need to modify ssidRouter, passwordRouter, ssidAP, and passwordAP. After making sure that the code is modified correctly, click "Run current script" and the "Shell" window will display the following:



Then you can see the ssid_A on the ESP32



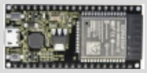
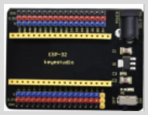
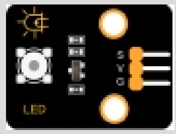
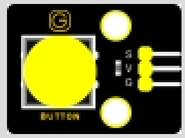
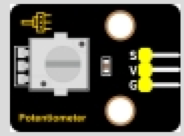
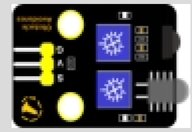
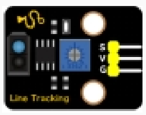
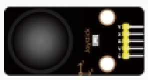
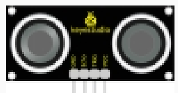
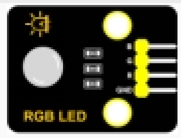
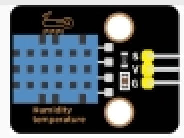
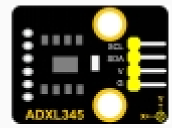
5.3.19 Project 63: Comprehensive Experiment



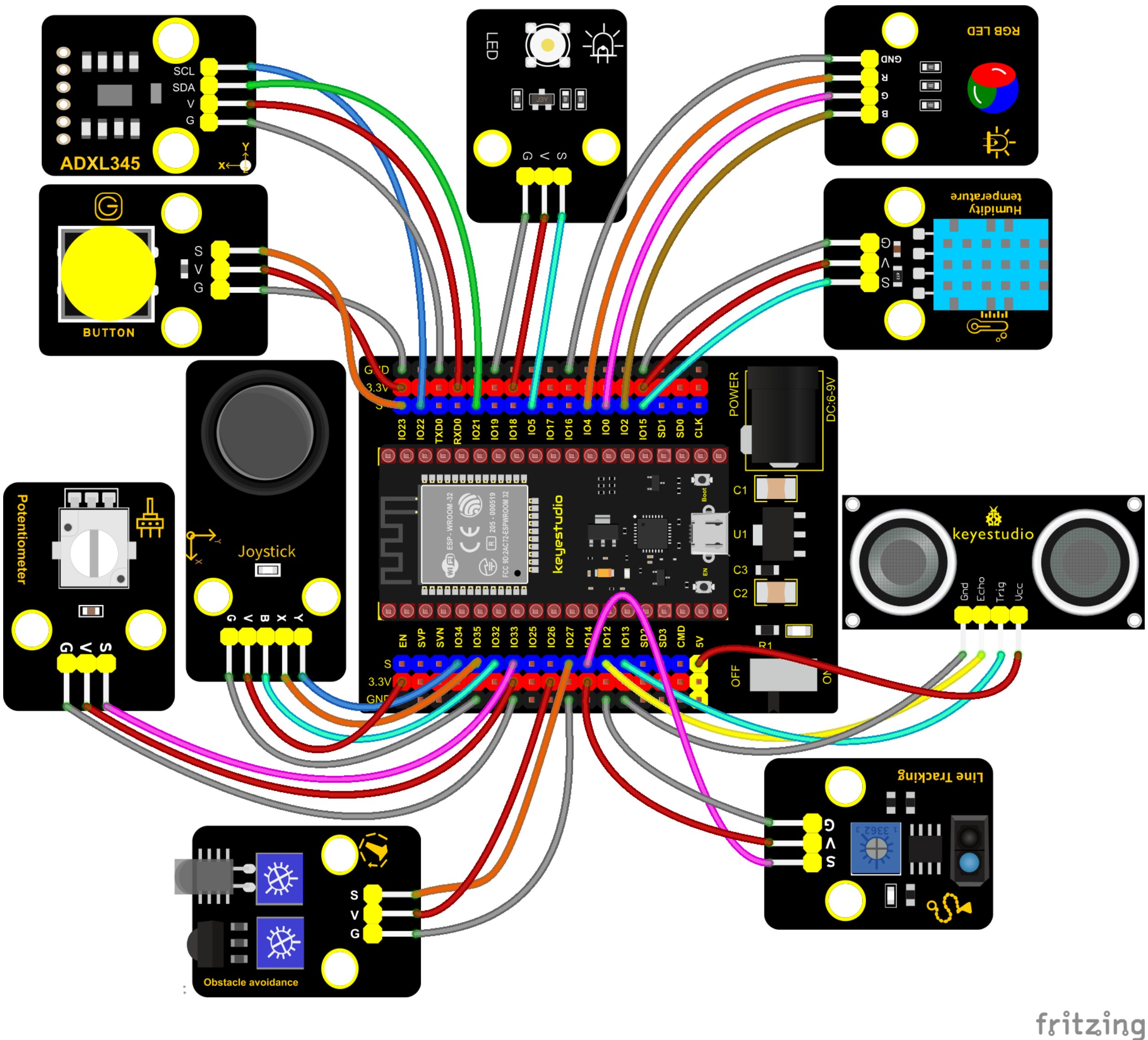
Introduction

We did a lot of experiments, and for each one we needed to re-upload the code, so can we achieve different functions through an experiment? In this experiment, we will use an external button module to achieve different functions.

Components Required

					
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Purple LED Module*1	Keyestudio Button Module*1	Keyestudio Rotary Potentiometer*1	Keyestudio Obstacle Avoidance Sensor*1
					
Keyestudio Line Tracking Sensor*1	Keyestudio DIY Joystick Module*1	Keyestudio HC-SR04 Ultrasonic sensor *1	Keyestudio DIY Common Cathode RGB Module *1	Keyestudio XHT11 Temperature and Humidity Sensor *1	Keyestudio ADXL345 Acceleration Sensor*1
					
Micro USB Cable*1	3P Dupont Wire*6	4P Dupont Wire*3	5P Dupont Wire*1		

Wiring Diagram



Test Code

```
from machine import ADC, Pin, PWM
import time
import machine
import random
import dht
from ADXL345 import adxl345

scl = Pin(22)
sda = Pin(21)
bus = 0
snsr = adxl345(bus, scl, sda)

pwm_r = PWM(Pin(4))
pwm_g = PWM(Pin(0))
pwm_b = PWM(Pin(2))
```

(continues on next page)

(continued from previous page)

```
pwm_r.freq(1000)
pwm_g.freq(1000)
pwm_b.freq(1000)

DHT = dht.DHT11(machine.Pin(15))

potentiometer_adc=ADC(Pin(33))
potentiometer_adc.atten(ADC.ATTN_11DB)
potentiometer_adc.width(ADC.WIDTH_12BIT)

button = Pin(23, Pin.IN)
led = PWM(Pin(5))
led.freq(1000)

tracking = Pin(14, Pin.IN, Pin.PULL_UP)

button_z=Pin(32,Pin.IN,Pin.PULL_UP)
rocker_x=ADC(Pin(35))
rocker_y=ADC(Pin(34))
rocker_x.atten(ADC.ATTN_11DB)
rocker_y.atten(ADC.ATTN_11DB)
rocker_x.width(ADC.WIDTH_12BIT)
rocker_y.width(ADC.WIDTH_12BIT)

avoid = Pin(27, Pin.IN)
# Set ultrasonic pins
trigger = Pin(13, Pin.OUT)
echo = Pin(12, Pin.IN)

def light(red, green, blue):
    pwm_r.duty(red)
    pwm_g.duty(green)
    pwm_b.duty(blue)

# Ultrasonic ranging, unit: cm
def getDistance(trigger, echo):
    # Generates a 10us square wave
    trigger.value(0)   #A short low level is given beforehand to ensure a clean high␣
↪pulse:
    time.sleep_us(2)
    trigger.value(1)
    time.sleep_us(10)#After pulling high, wait 10 microseconds and immediately set it to␣
↪low
    trigger.value(0)

    while echo.value() == 0: #Establish a while loop to detect whether the echo pin␣
↪value is 0 and record the time at that time
        start = time.ticks_us()
    while echo.value() == 1: #Establish a while loop to check whether the echo pin value␣
↪is 1 and record the time at that time
        end = time.ticks_us()
```

(continues on next page)

(continued from previous page)

```
    d = (end - start) * 0.0343 / 2 #The travel time of the sound wave x the speed of␣
↪sound (343.2 m/s, 0.0343 cm/microsecond), and the distance back and forth divided by 2
    return d

    keys = 0
    nums = 0
    print(keys % 8)
    def toggle_handle(pin):
        global keys
        keys += 1
        print(keys % 7)
    button.irq(trigger = Pin.IRQ_FALLING, handler = toggle_handle)

def showRGB():
    R = random.randint(0,1023)
    G = random.randint(0,1023)
    B = random.randint(0,1023)
    light(R, G, B)
    time.sleep(0.3)

def showxht11():
    DHT.measure()
    print('temperature:',DHT.temperature(),'℃','humidity:',DHT.humidity(),'%')
    time.sleep(1)

def showtracking():
    if tracking.value() == 0:
        print("0   White")   #Press to print the corresponding information.
    else:
        print("1   Black")
    time.sleep(0.1) #delay 0.1s

def showJoystick():
    B_value = button_z.value()
    X_value = rocker_x.read()
    Y_value = rocker_y.read()
    print("button:", end = " ")
    print(B_value, end = " ")
    print("X:", end = " ")
    print(X_value, end = " ")
    print("Y:", end = " ")
    print(Y_value)
    time.sleep(0.1)

def adjustLight():
    pot_value = potentiometer_adc.read()
    print(pot_value)
    led.duty(pot_value)
    time.sleep(0.1)

def showAvoid():
    if avoid.value() == 0:
```

(continues on next page)

(continued from previous page)

```
        print("There are obstacles")
    else:
        print("All going well")
    time.sleep(0.1)

def showDistance():
    distance = getDistance(trigger, echo)
    print("The distance is {:.2f} cm".format(distance))
    time.sleep(0.1)

def showADXL345():
    x,y,z = snsr.readXYZ()
    print('x:',x,'y:',y,'z:',z,'uint:mg')
    time.sleep(0.1)

while True:
    nums = keys % 8  #number of keystrokes mod 7 to get 0, 1, 2, 3, 4, 5, 6
    if nums == 0:  #According to RGB
        showRGB()
    elif nums == 1:  #Displays the high and low level of the tracking sensor
        showtracking()
    elif nums == 2:  #Display temperature and humidity
        showxht11()
    elif nums == 3:  #Displays the rocker value
        showJoystick()
    elif nums == 4:  #The potentiometer adjusts the LED
        adjustLight()
    elif nums == 5:  #Display obstacle information
        showAvoid()
    elif nums == 6:  #Display ultrasonic ranging value
        showDistance()
    elif nums == 7:  #Display ADXL345_x/y/z value
        showADXL345()
```

Code Explanation

Calculate how many times the button is pressed, divide it by 8, and get the remainder which is 0, 1 2, 3, 4, 5 , 6 and 7. According to different remainders, construct five unique functions to control the experiment and realize different functions.

We add adxl345 library files in this project.

Following the instructions, we can add or remove sensors/modules in the wiring, and then change the experimental function in the code.

Test Result

Connect the wires according to the wiring diagram, use the USB to power on, and then click  run the test code. At the beginning, the number of keys is 0, the remainder is 0, and the four lamp beads on the RGB module flash with random colors.

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

  [83]
  b'\xe5'
  adxl345 found
  83
  adxl345 found
  0
```

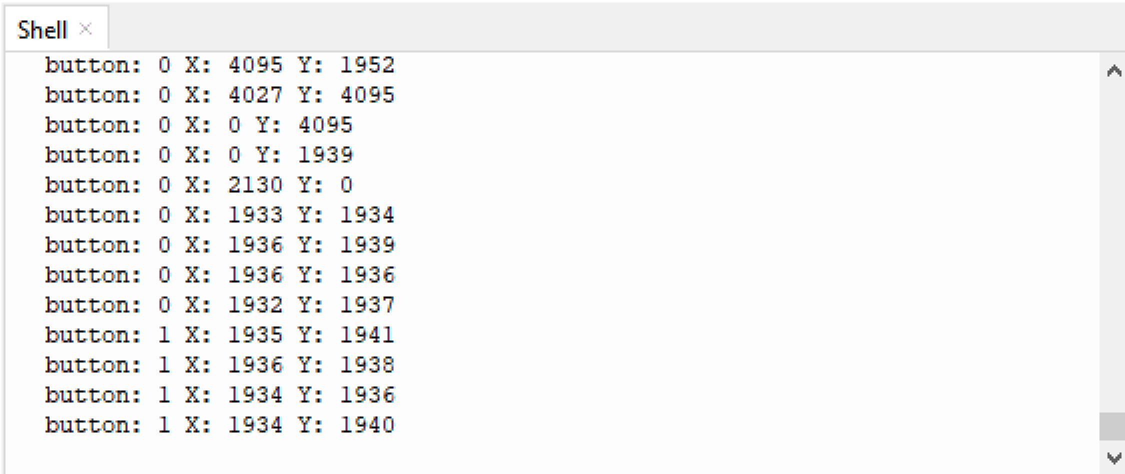
Press the button, the RGB stops flashing, press once, the remainder is the function of the experiment is to track the sensor according to black and white objects read high and low levels, the following information is displayed.

```
Shell
  1   Black
  1   Black
  1   Black
  0   White
  0   White
  0   White
  0   White
  0   White
  0   White
  1   Black
  1   Black
```

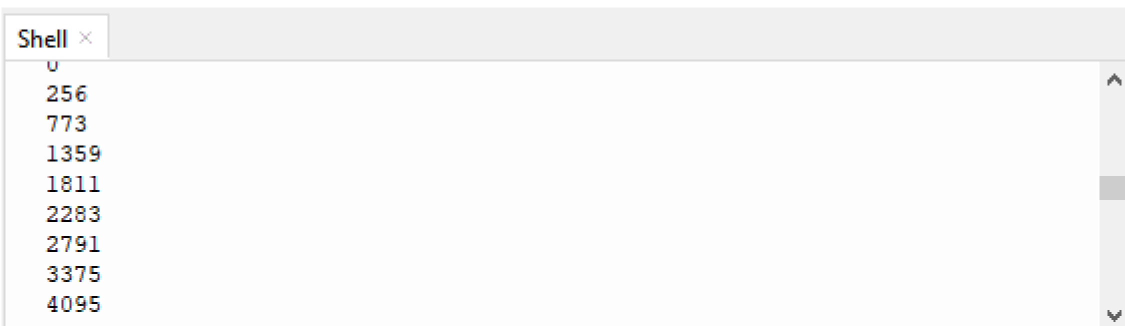
Press the key twice, the time of pressing buttons is 2 and the remainder is 2. Read temperature and humidity values. As shown below;

```
Shell
  temperature: 27 ℃ humidity: 47 %
  temperature: 27 ℃ humidity: 47 %
  temperature: 27 ℃ humidity: 55 %
  temperature: 27 ℃ humidity: 76 %
  temperature: 28 ℃ humidity: 82 %
  temperature: 28 ℃ humidity: 86 %
  temperature: 28 ℃ humidity: 90 %
  temperature: 29 ℃ humidity: 91 %
  temperature: 29 ℃ humidity: 92 %
  temperature: 29 ℃ humidity: 92 %
  temperature: 29 ℃ humidity: 93 %
  temperature: 29 ℃ humidity: 93 %
```

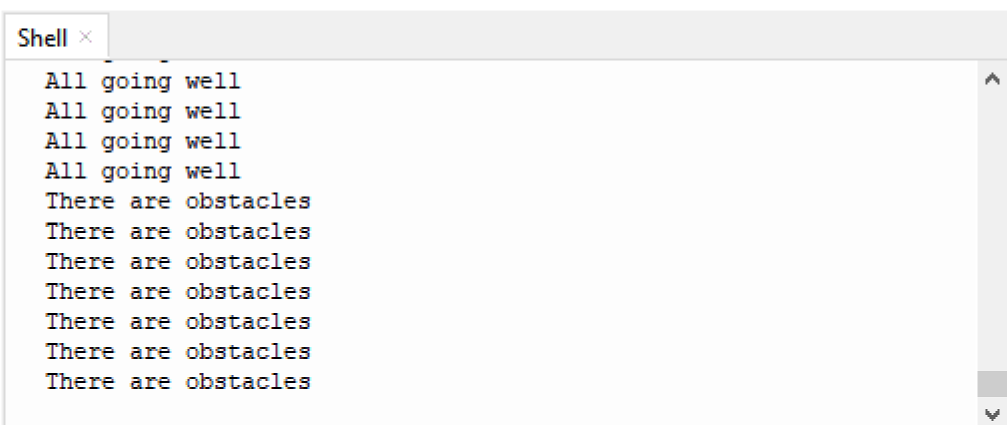
Press the key again, the time of pressing buttons is 3 and the remainder is 3. Read digital values at x, y and z axis of the joystick module. As shown below;



Press the key for the fourth time, the remainder is 4. Then the potentiometer can adjust the PWM value at the GPI05 port to control LED brightness of the purple LED;



Press the key for the fifth time, the remainder is 5. Then the obstacle avoidance sensor can detect obstacles, as shown below;



Press the key for the sixth time, the remainder is 6. Then the ultrasonic sensor can detect distance away from obstacles, as shown below;

```
Shell ×
The distance is :6.98 cm
The distance is :6.12 cm
The distance is :6.98 cm
The distance is :6.98 cm
The distance is :8.09 cm
The distance is :8.56 cm
The distance is :8.97 cm
The distance is :8.16 cm
The distance is :8.51 cm
The distance is :8.97 cm
The distance is :9.36 cm
```

Press the key for seventh time and the remainder is 7. The shell will print out the acceleration value;

```
Shell ×
x: -639.6 y: 830.7 z: 230.1 uint:mg
x: -686.4 y: 538.2 z: 358.8 uint:mg
x: -542.1 y: 666.9 z: -323.7 uint:mg
x: -276.9 y: 783.9 z: -795.6 uint:mg
x: -163.8 y: 444.6 z: -744.9 uint:mg
x: -62.4 y: 284.7 z: -1099.8 uint:mg
x: 93.60001 y: -93.60001 z: -31.2 uint:mg
x: 214.5 y: -308.1 z: -819.0 uint:mg
x: 140.4 y: -378.3 z: -858.0 uint:mg
x: 226.2 y: -370.5 z: -920.4 uint:mg
x: 226.2 y: -315.9 z: -861.9 uint:mg
```

Press the key for eighth time and the remainder is 0. Then the RGB will flash. If you press keys incessantly, remainders will change in a loop way. So does functions.

Press "Ctrl+C"or click "Stop/Restart backend"to exit the program.

CHAPTER

SIX

ARDUINO TUTORIAL

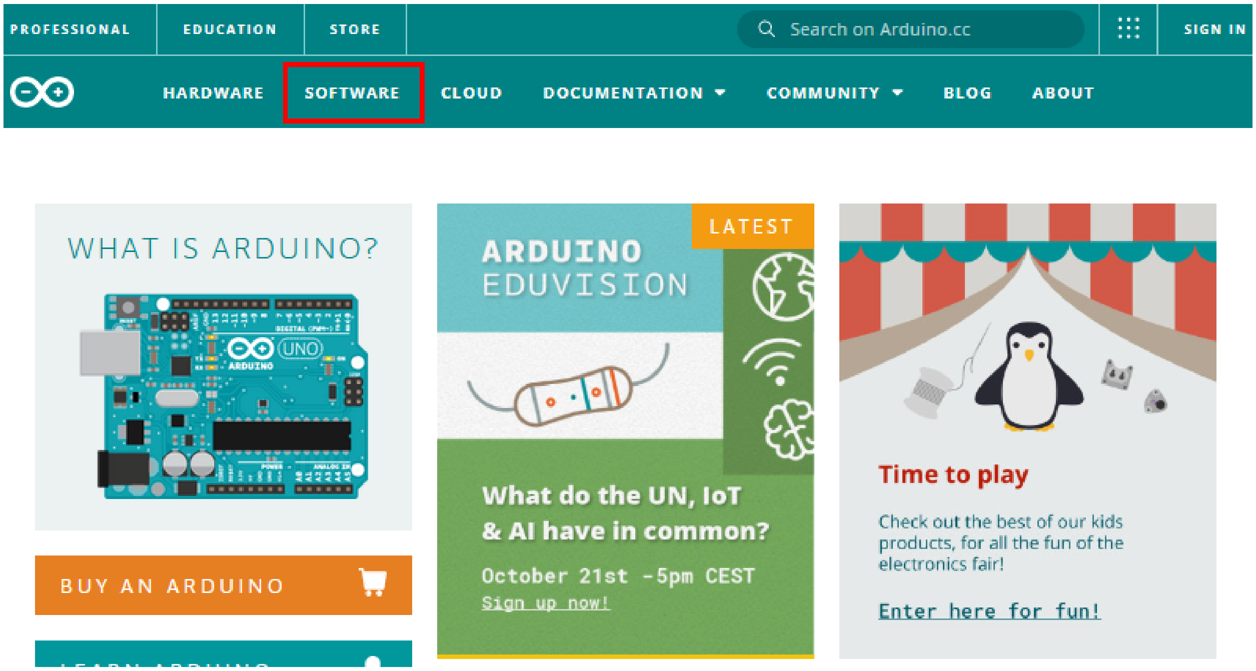
6.1 1. Get started with Arduino C:

6.1.1 1. Windows System

1.1 Installing Arduino IDE:

When you get control board, you need to download Arduino IDE and driver firstly.

You could download Arduino IDE from the official website: https://www.arduino.cc/, click the "**SOFTWARE**"on the browse bar, click "**DOWNLOADS**" to enter download page, as shown below:

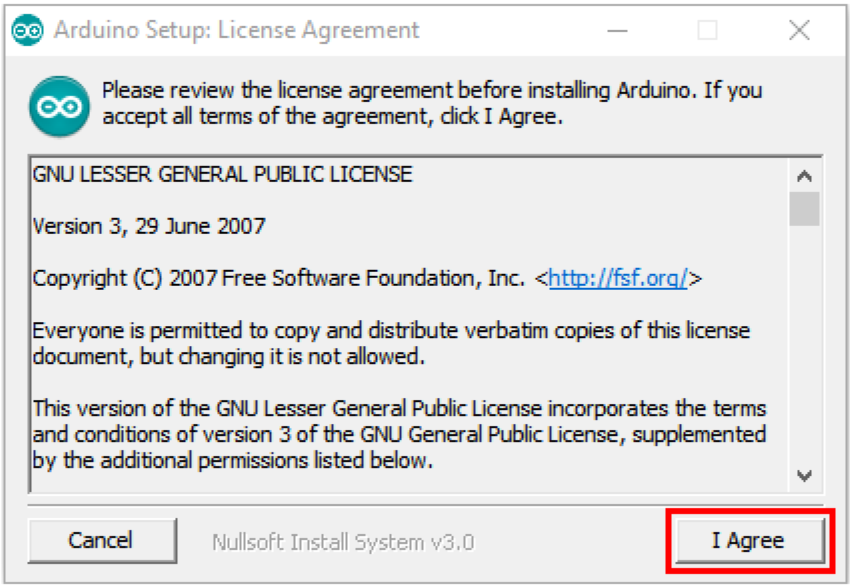


There are various versions of IDE for Arduino. Just download a version compatible with your system. Here we will show you how to download and install the windows version of Arduino IDE.

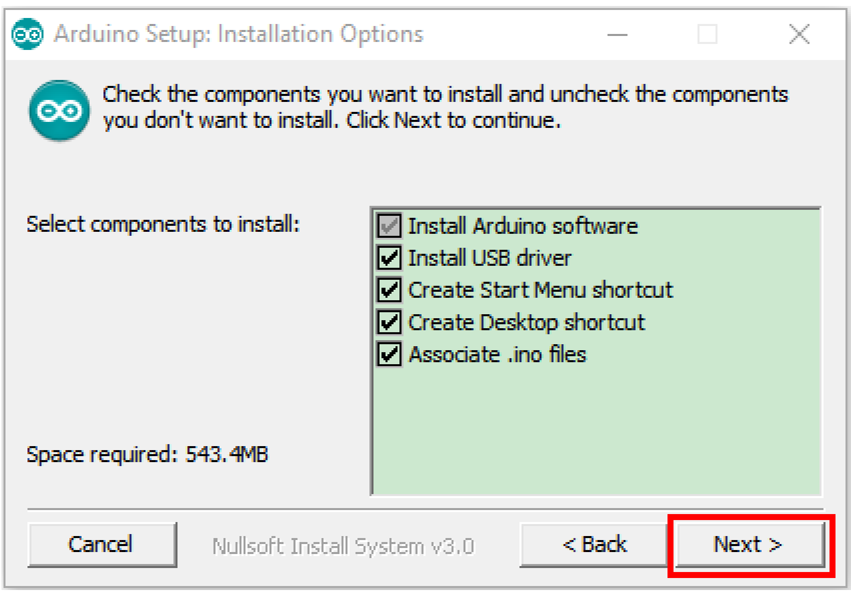
There are two versions of IDE for Windows system: Windows Win7 and newer and Windows ZIP file. The former needs to install manually, while the latter can be directly downloaded, without the need of installing it manually.

You just need to click "**JUST DOWNLOAD**".

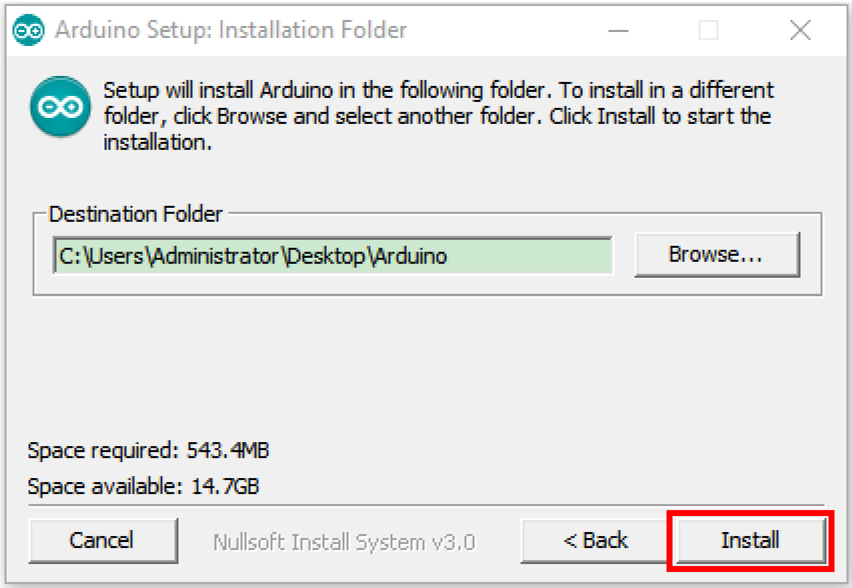
After the Arduino is downloaded, click "**I Agree**" to continue installing.



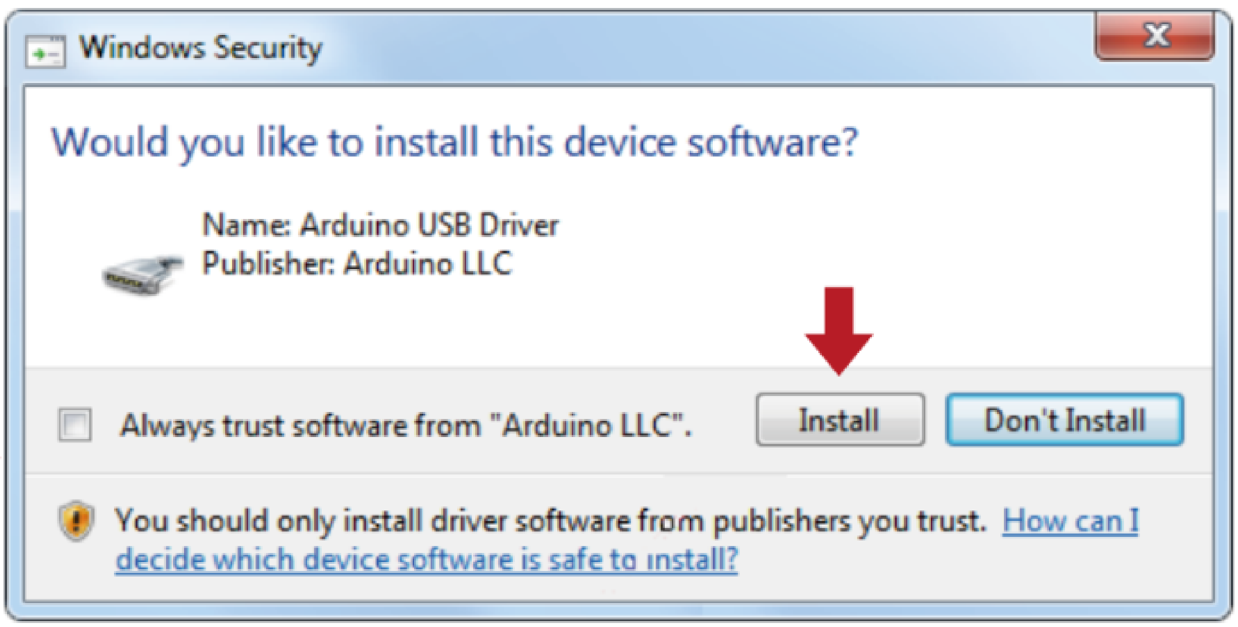
Click “**Next**”.

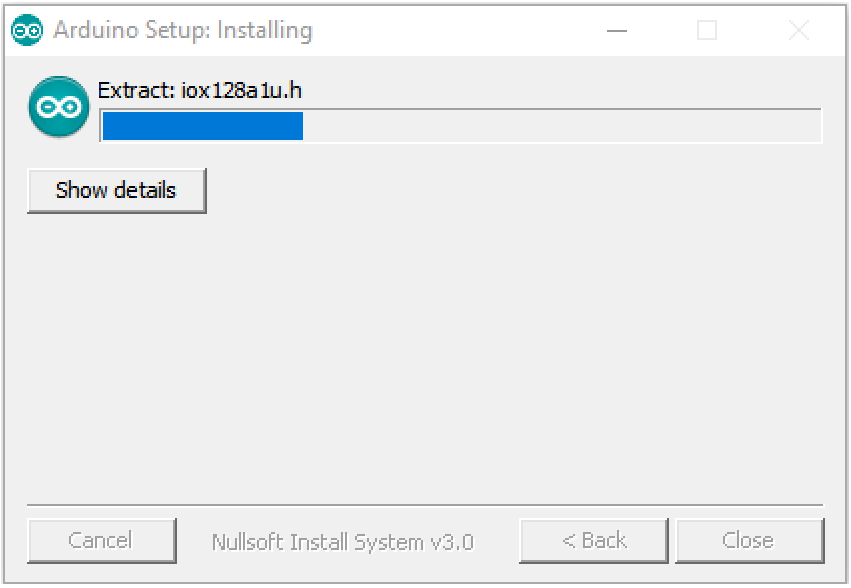


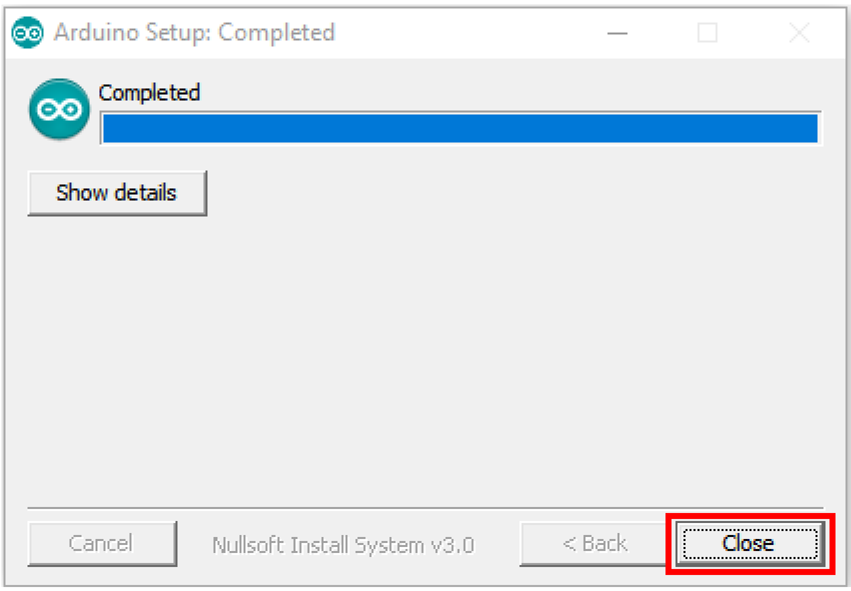
Then click “**Install**”.



If the following page appears, click "**Install**".







1.2 Install a driver on Windows

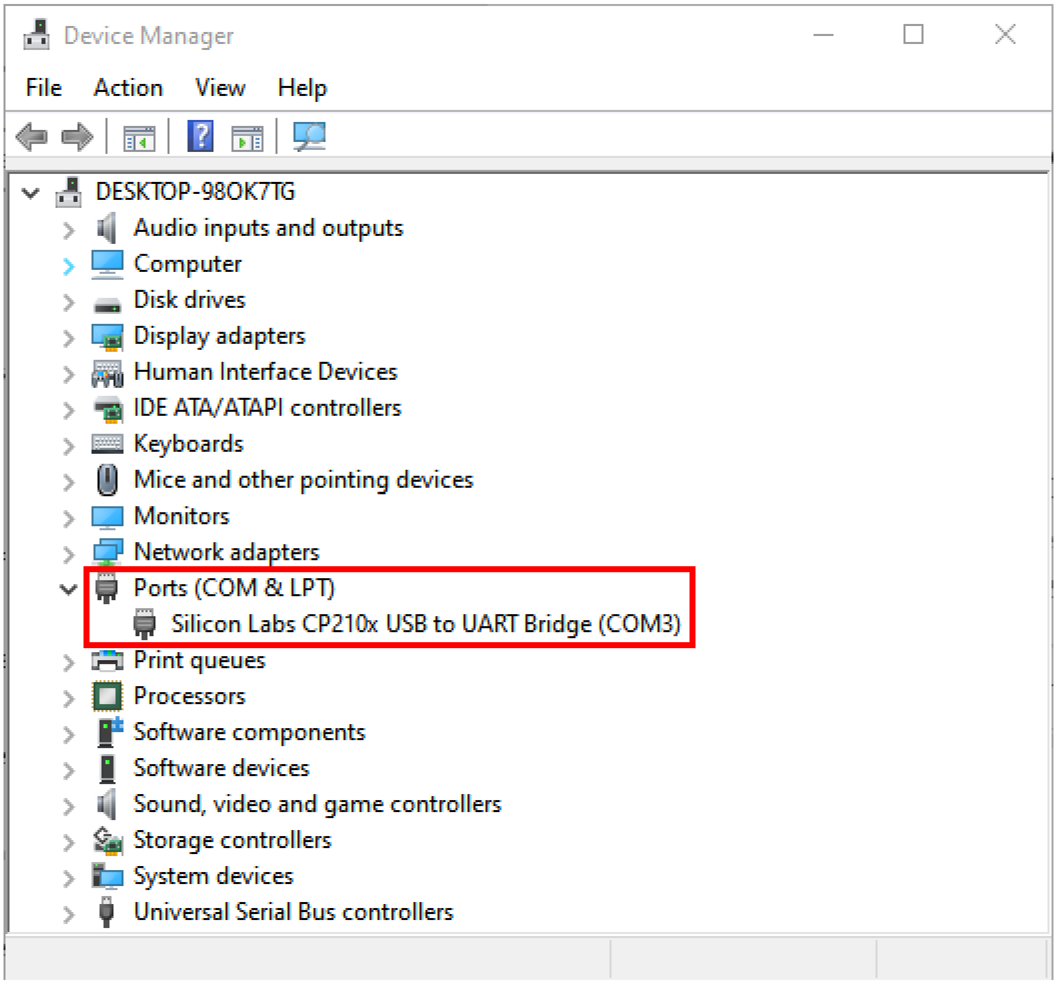
If you have installed the driver, just skip it.

Before using the ESP32 board, you must install a driver, otherwise it will not communicate with computer.

Unlike the USB series chip (ATMEGA8U2) of the Arduino UNO R3, the ESP32 board is used the CP2102 chip USB series chip and USB type C interface.

The driver of the CP2102 chip is included in 1.8.0 version and newer version of Arduino IDE. Usually, you connect the board to the computer and wait for Windows to begin its driver installation process. After a few moments, the process will succeed.

Right click "**Computer**"—– Click "**Properties**"—–Click "**Device Manager**". Look under Ports (COM & LPT) or other devices, The driver of CP2102 is installed successfully. As shown below:



If the driver installation process fail, you need to install the driver manually.

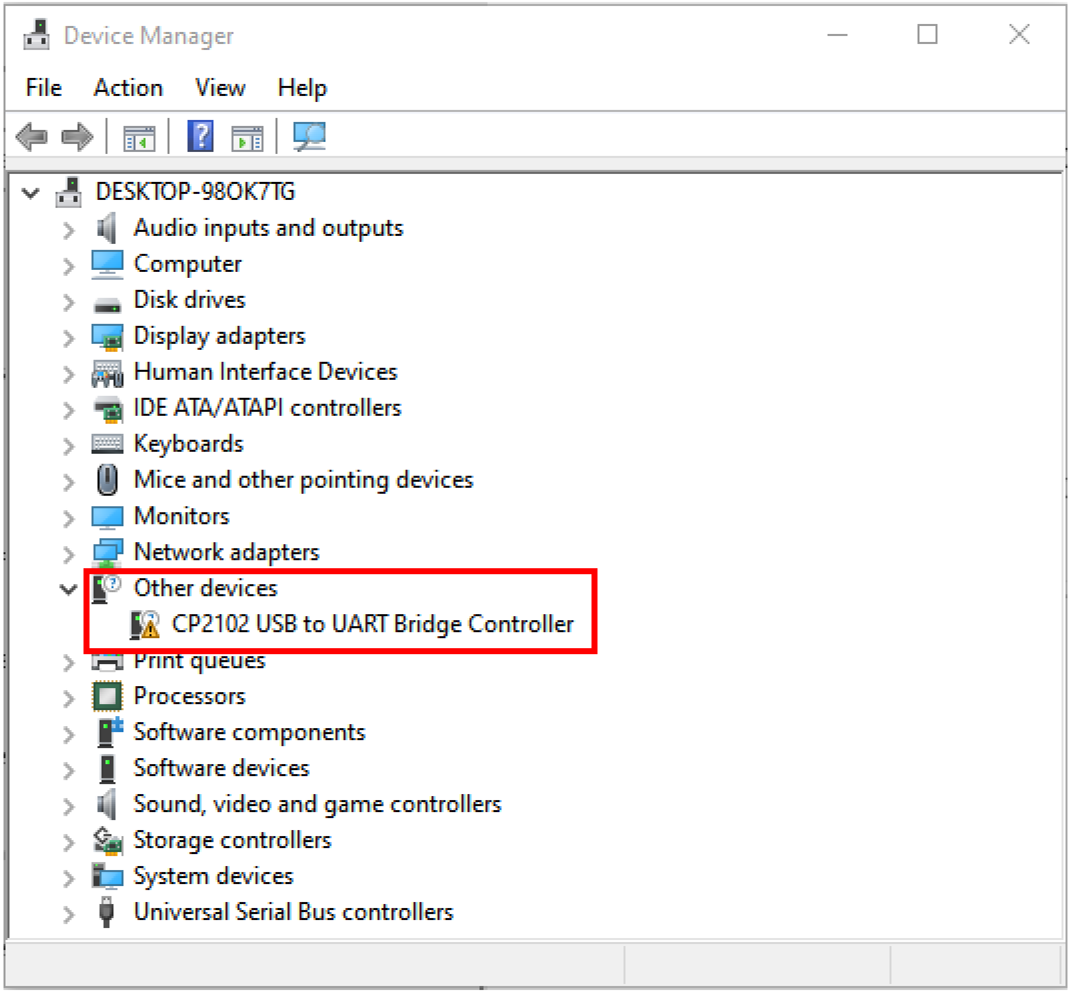
Note:

1). Please make sure that your IDE is updated to 1.8.0 or newer version.

2). If the version of Arduino IDE you download is below 1.8, you should download the driver of CP2102 and install it manually.

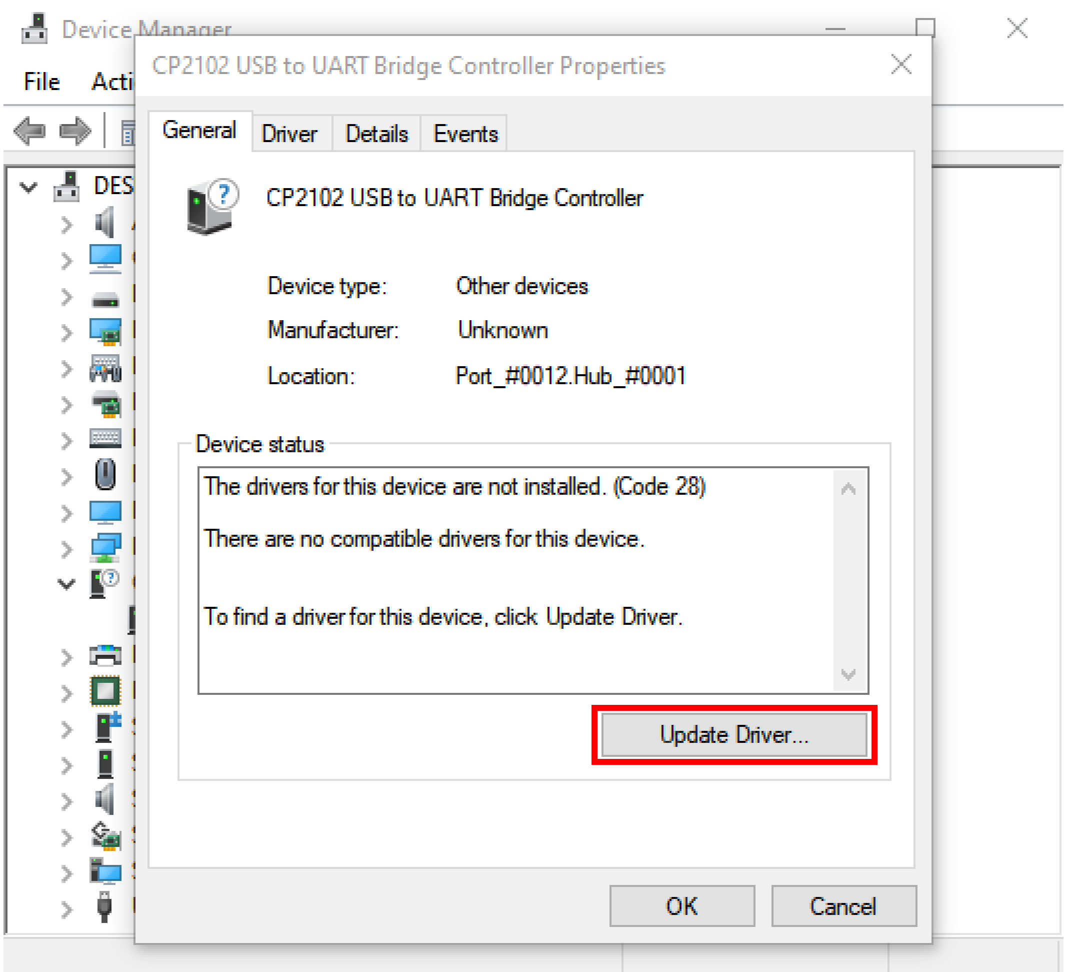
Link to download the driver of CP2102 : https://fs.keyestudio.com/CP2102-WIN

To install the drive manually, open the device manager of computer. A yellow exclamation mark means that the CP2102 driver installation failed.

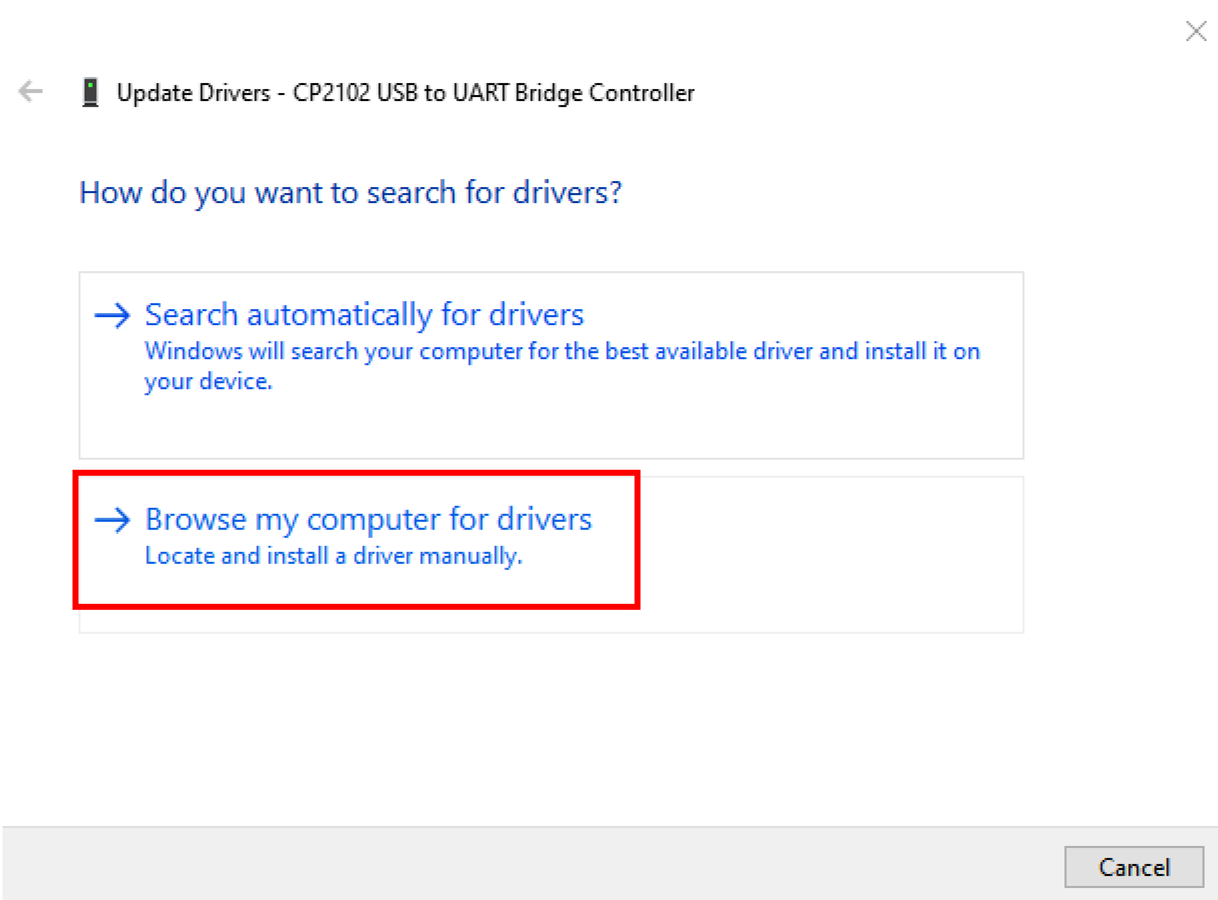


Double-click 

 and click " **Update drive...**"



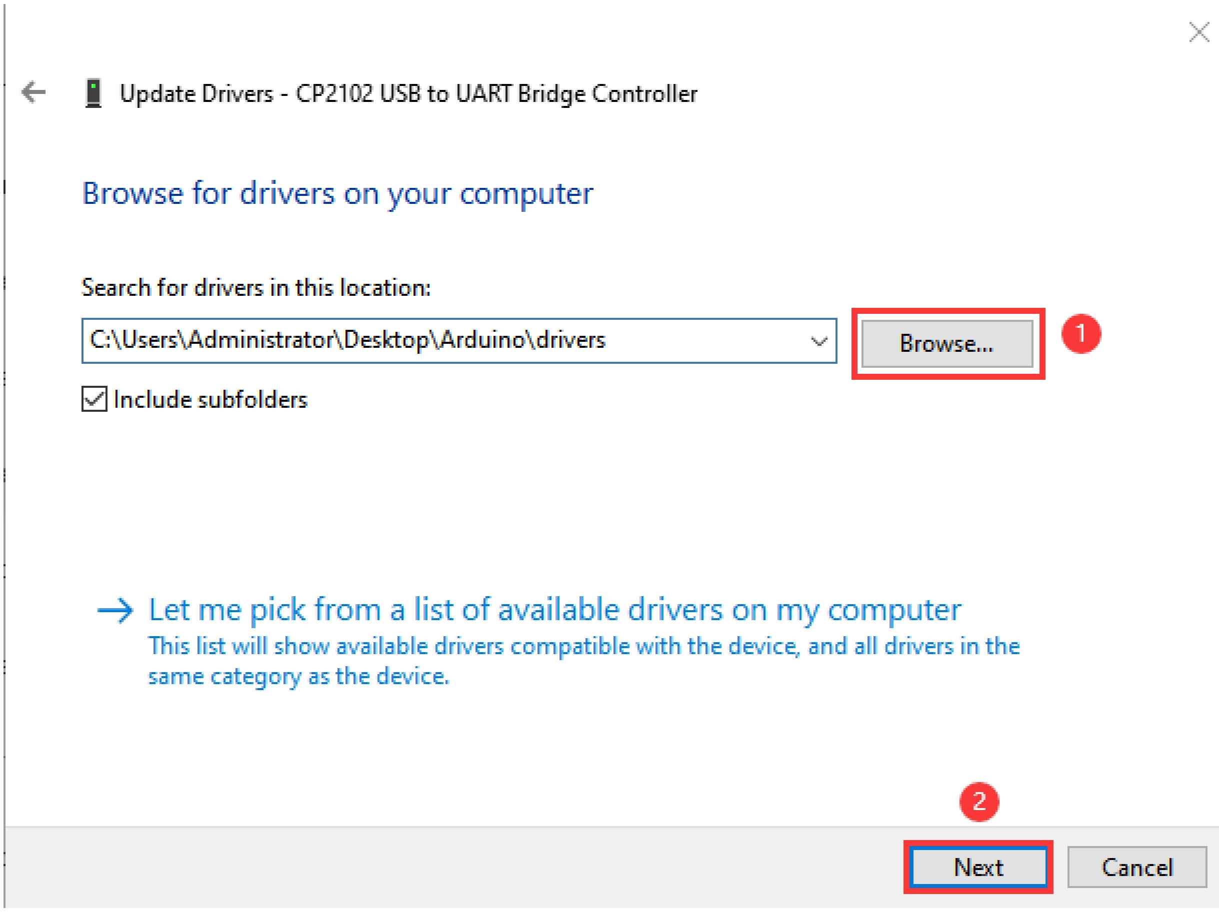
Click “**Browse my computer for drivers**” for updated driver software.

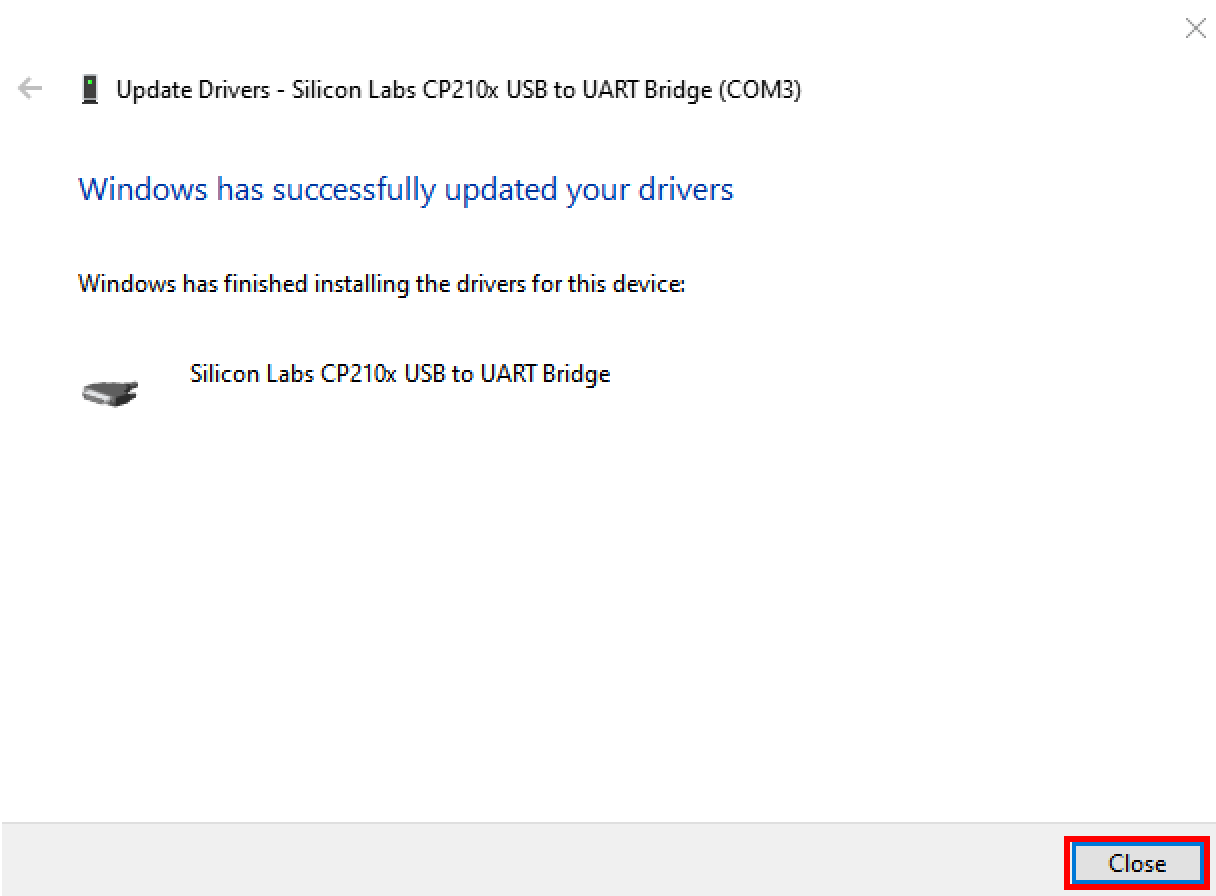


There is a DRIVERS folder in Arduino software installed package 

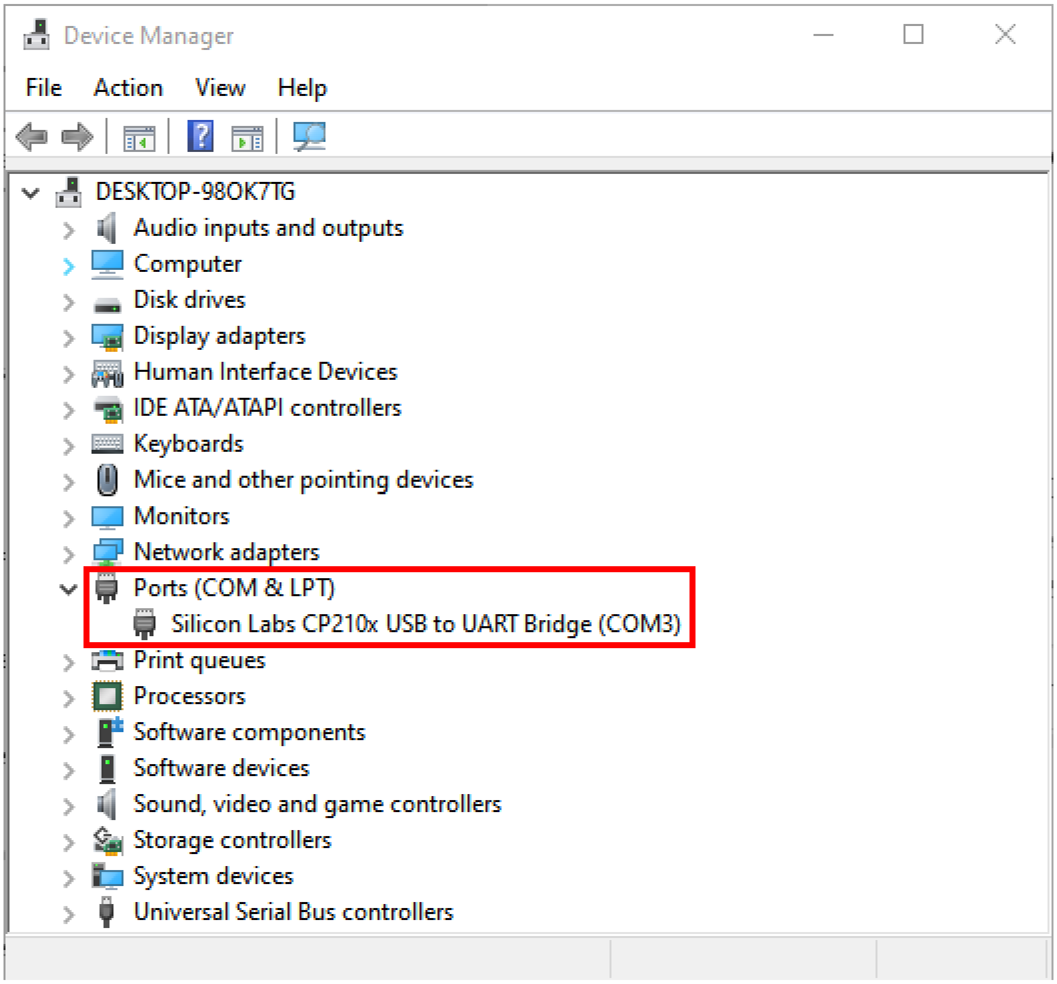
), open driver folder and you can see the driver of CP210X series chips.

Click "**Browse**", then find the driver folder, or you could enter "**driver**"to search in rectangular box, then click "**Next**",


Update Drivers - CP2102 USB to UART Bridge Controller
Browse for drivers on your computer
Search for drivers in this location:
C:\Users\Administrator\Desktop\Arduino\drivers
Browse...
1
Include subfolders
Let me pick from a list of available drivers on my computer
This list will show available drivers compatible with the device, and all drivers in the same category as the device.
2
Next
Cancel



Open **device manager**, you will find the yellow exclamation mark disappear. The driver of CP2102 is installed successfully.

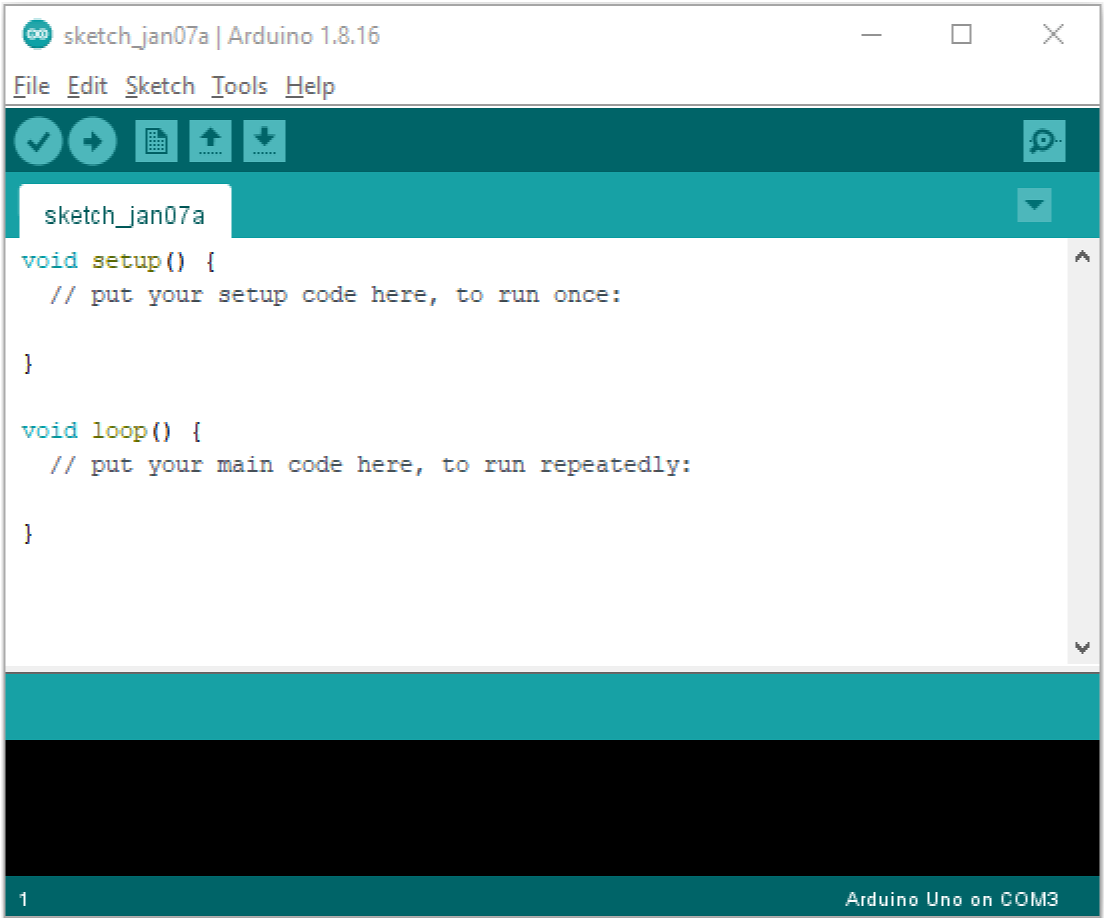


1.3. Install the ESP32 on Arduino IDE

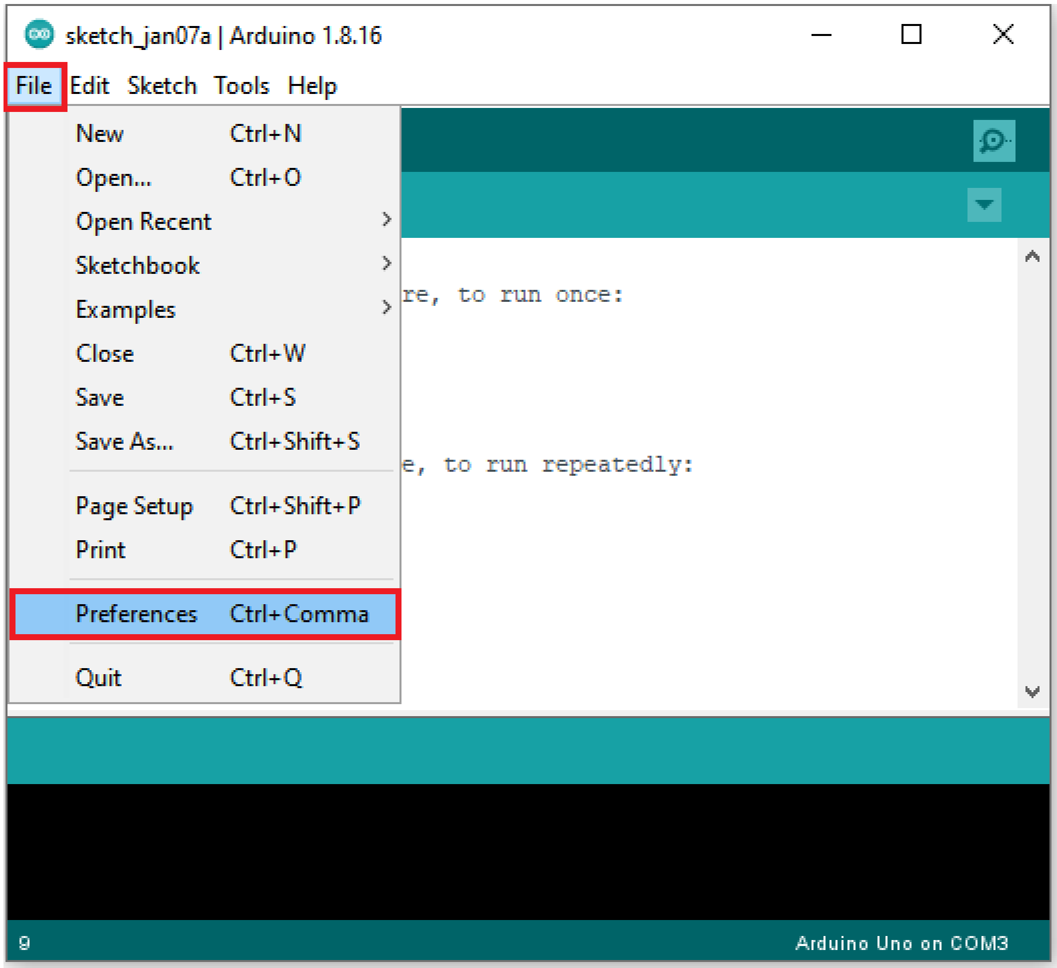
Note: you need to download Arduino IDE 1.8.5 or advanced version to install the ESP32.

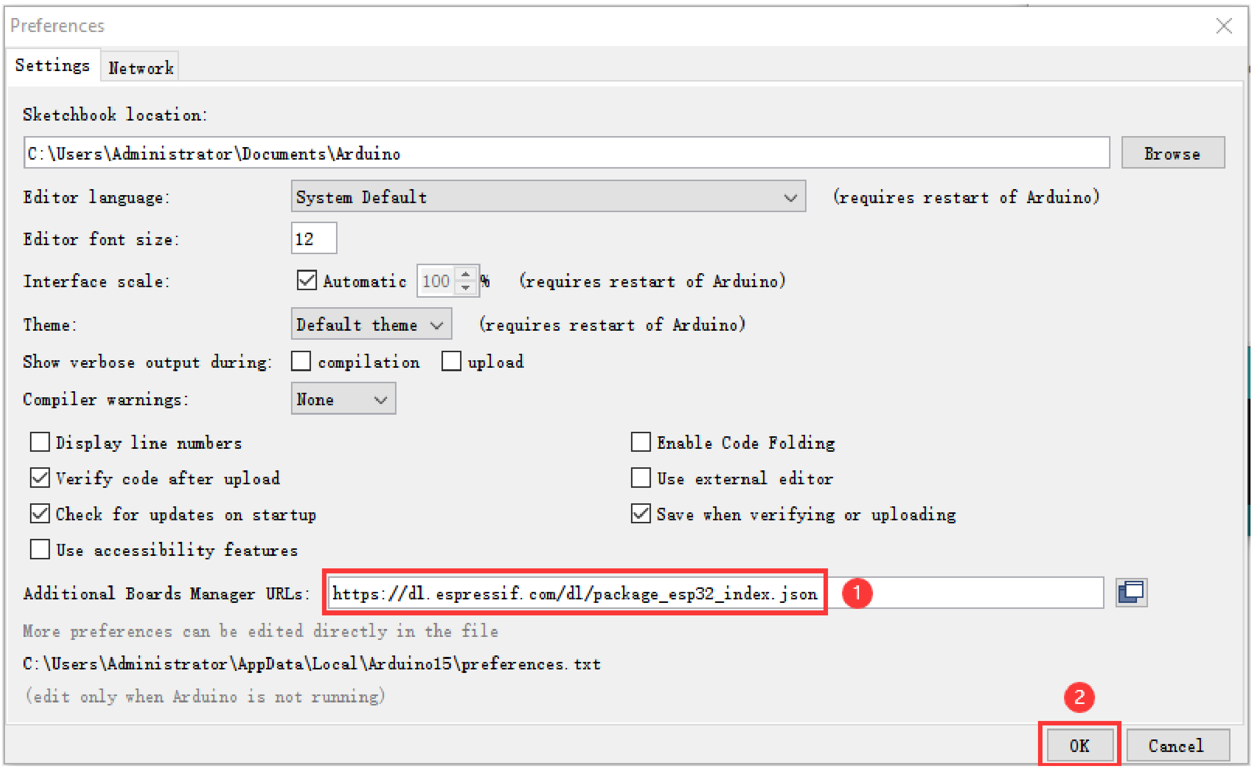
1). Click 

to open Arduino IDE

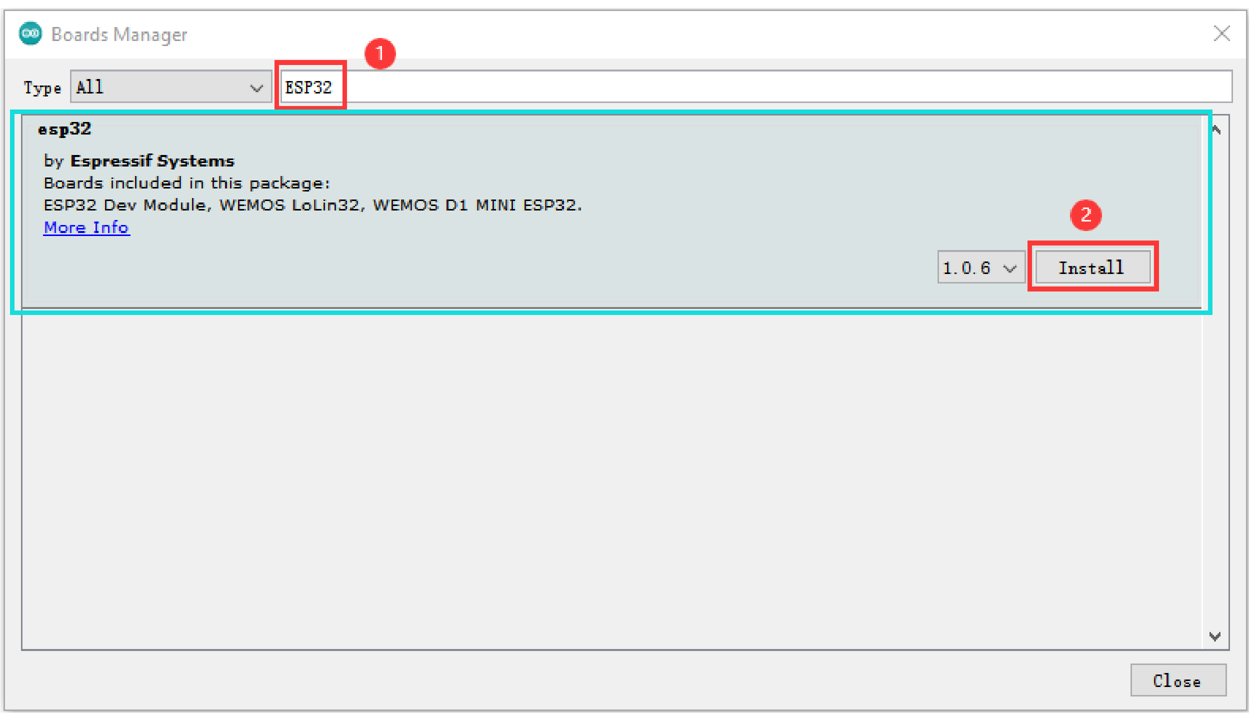


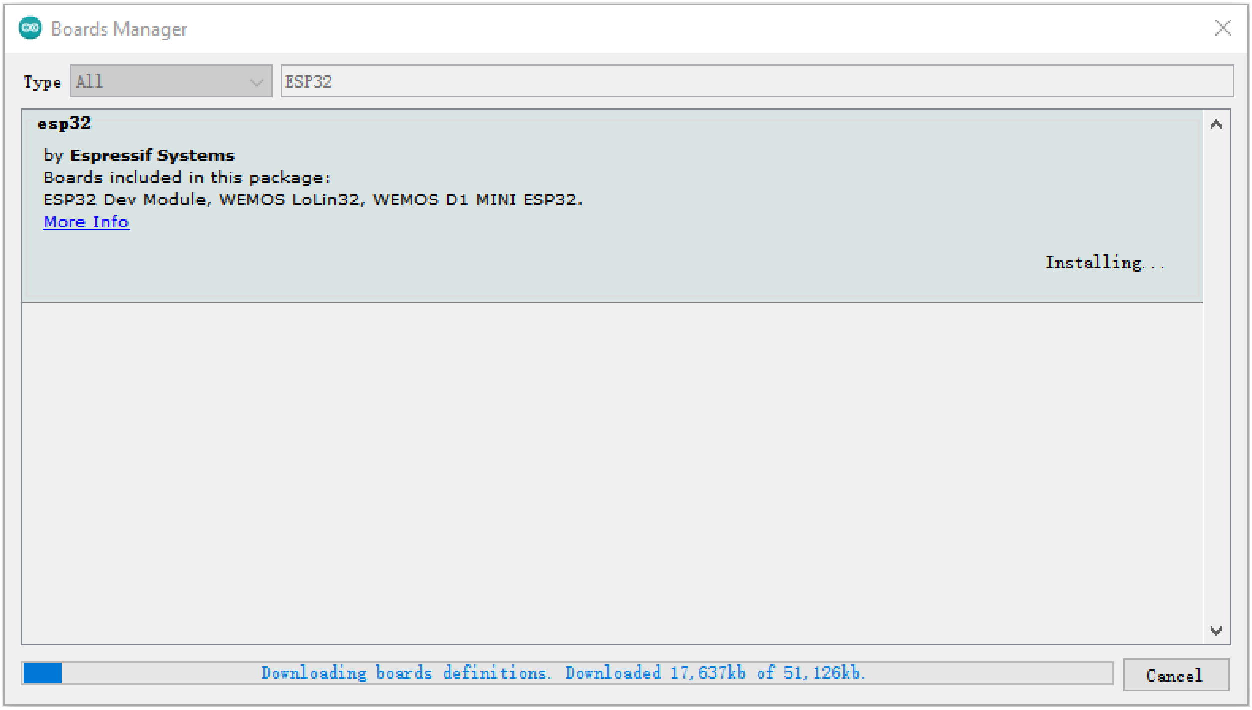
2). Click "**File**" → "**Preferences**"copy the website address https://dl.espressif.com/dl/package_esp32_index.json in the "**Additional Boards Manager URLs:**" and click "**OK**"


sketch_jan07a | Arduino 1.8.16
File
Edit
Sketch
Tools
Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
re, to run once:
e, to run repeatedly:
9
Arduino Uno on COM3



3). Click "**Tools**" → "**Board:**" then click "**Boards Manager…**" to enter "**Boards Manager**". Enter "**ESP32**" as follows, then click "**Install**".



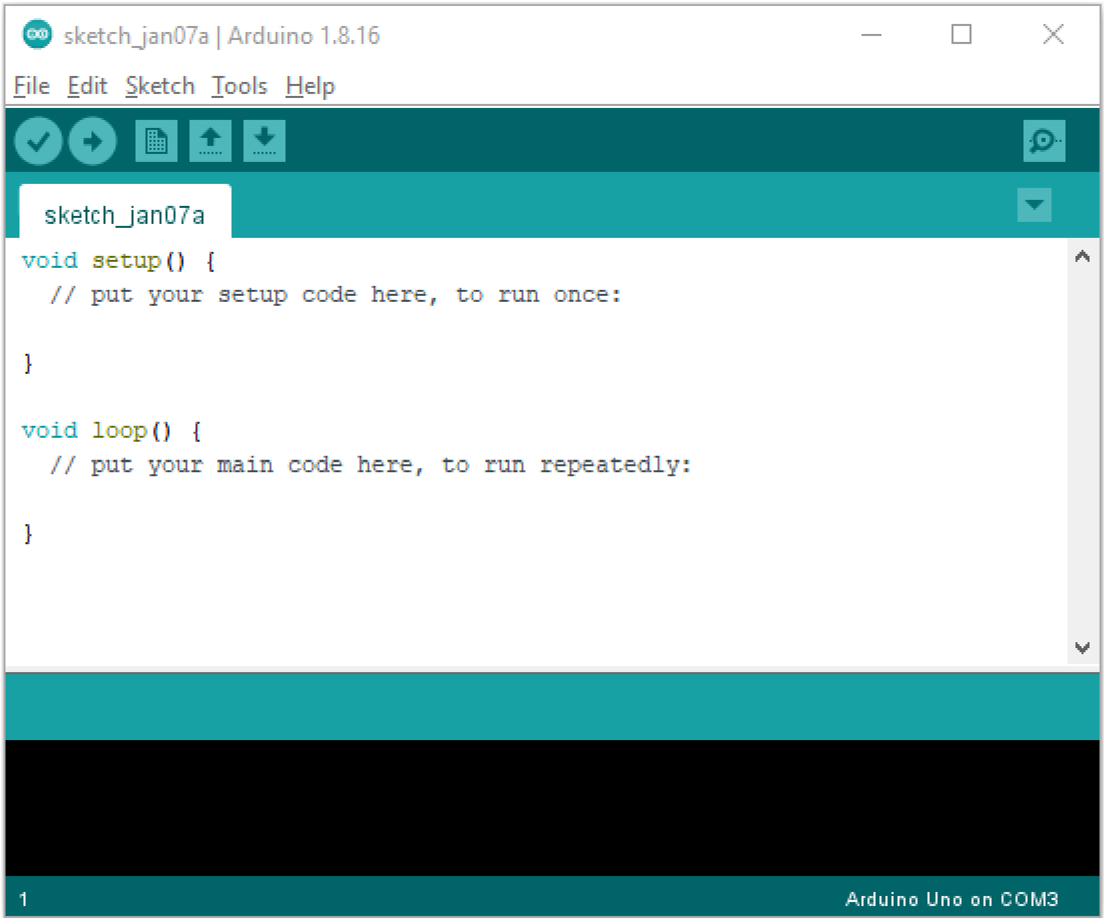


4). After installing, click "**Close**".

1.4. Arduino IDE Setting:

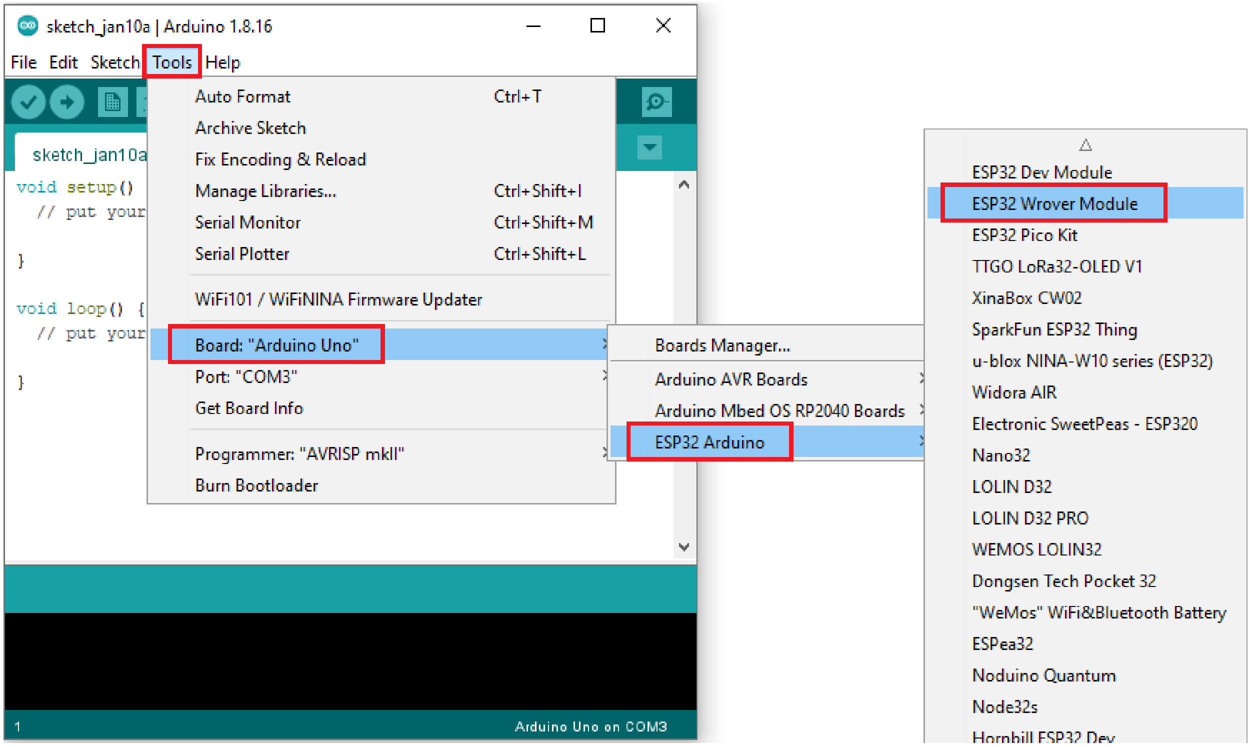
1). Click 

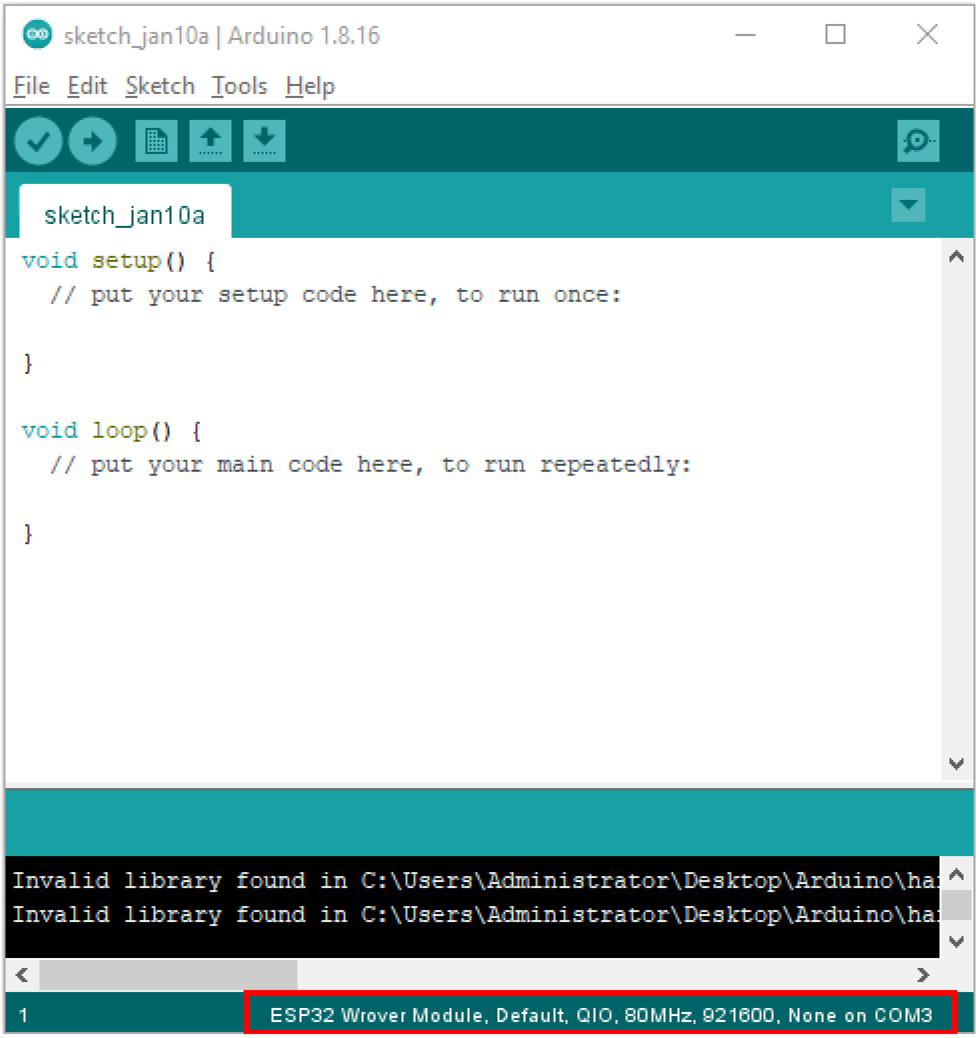
icon to pen Arduino IDE.



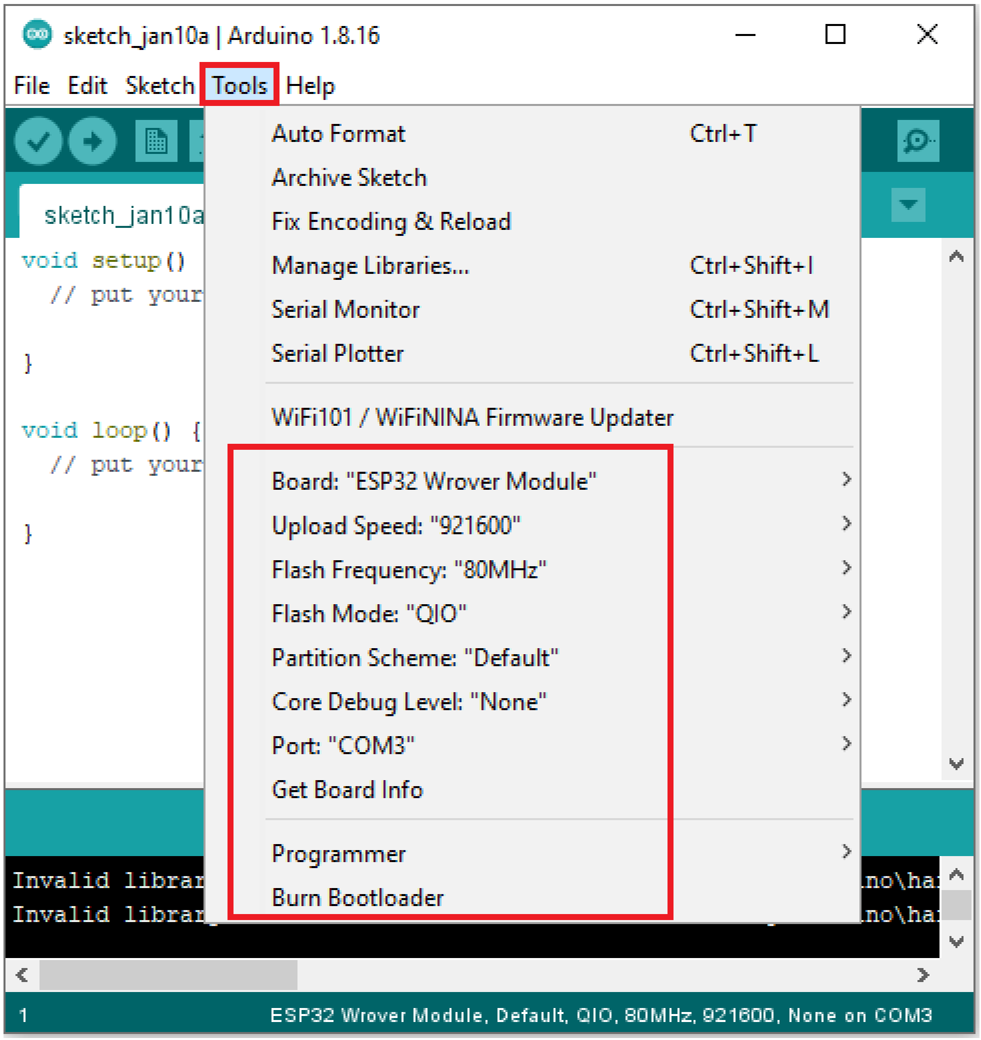
2). When downloading the sketch to the board, you must select the correct name of Arduino board that matches the board connected to your computer. As shown below;

(Note: we use the ESP32 board in this tutorial; therefore, we select ESP32**)**

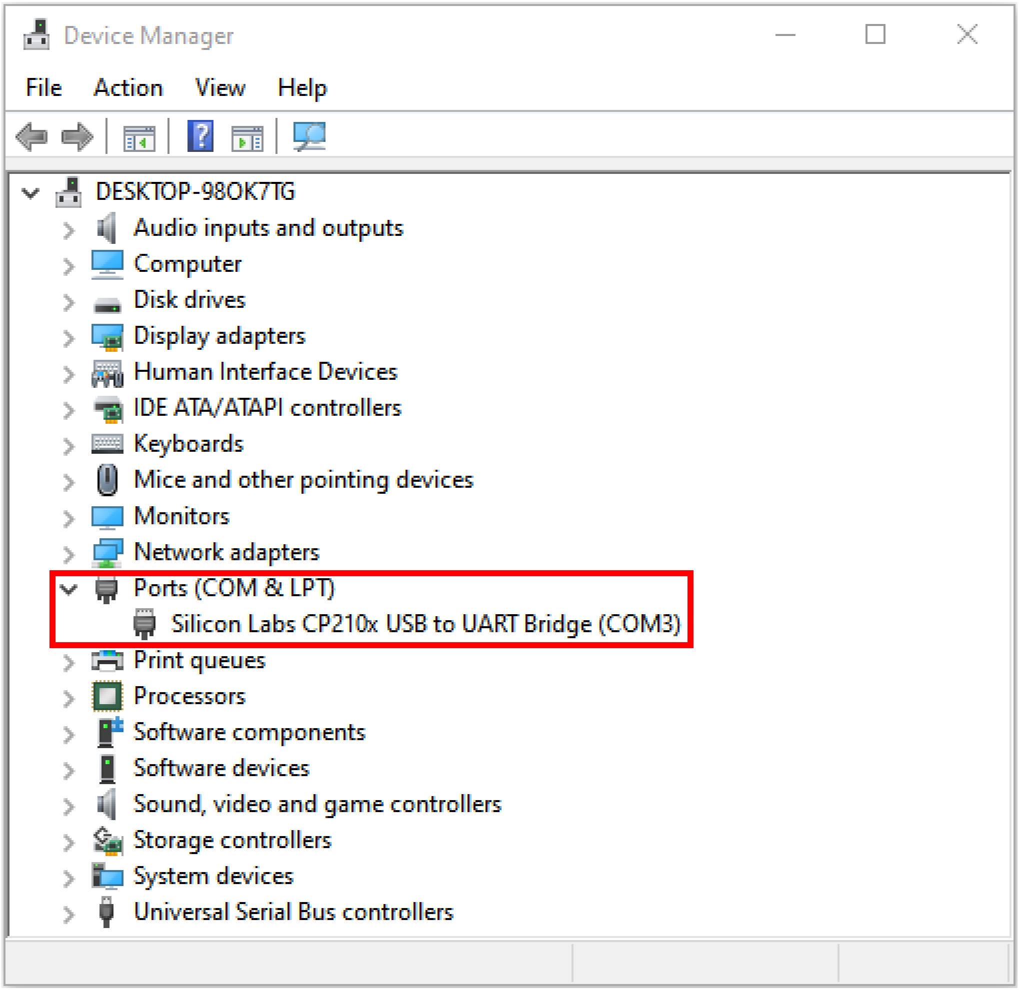

sketch_jan10a | Arduino 1.8.16
File Edit Sketch Tools Help
sketch_jan10a
void setup()
// put your
}
void loop() {
// put your
}
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port: "COM3"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Mbed OS RP2040 Boards
ESP32 Arduino
ESP32 Dev Module
ESP32 Wrover Module
ESP32 Pico Kit
TTGO LoRa32-OLED V1
XinaBox CW02
SparkFun ESP32 Thing
u-blox NINA-W10 series (ESP32)
Widora AIR
Electronic SweetPeas - ESP320
Nano32
LOLIN D32
LOLIN D32 PRO
WEMOS LOLIN32
Dongsen Tech Pocket 32
"WeMos" WiFi&Bluetooth Battery
ESPea32
Noduino Quantum
Node32s
1
Arduino Uno on COM3

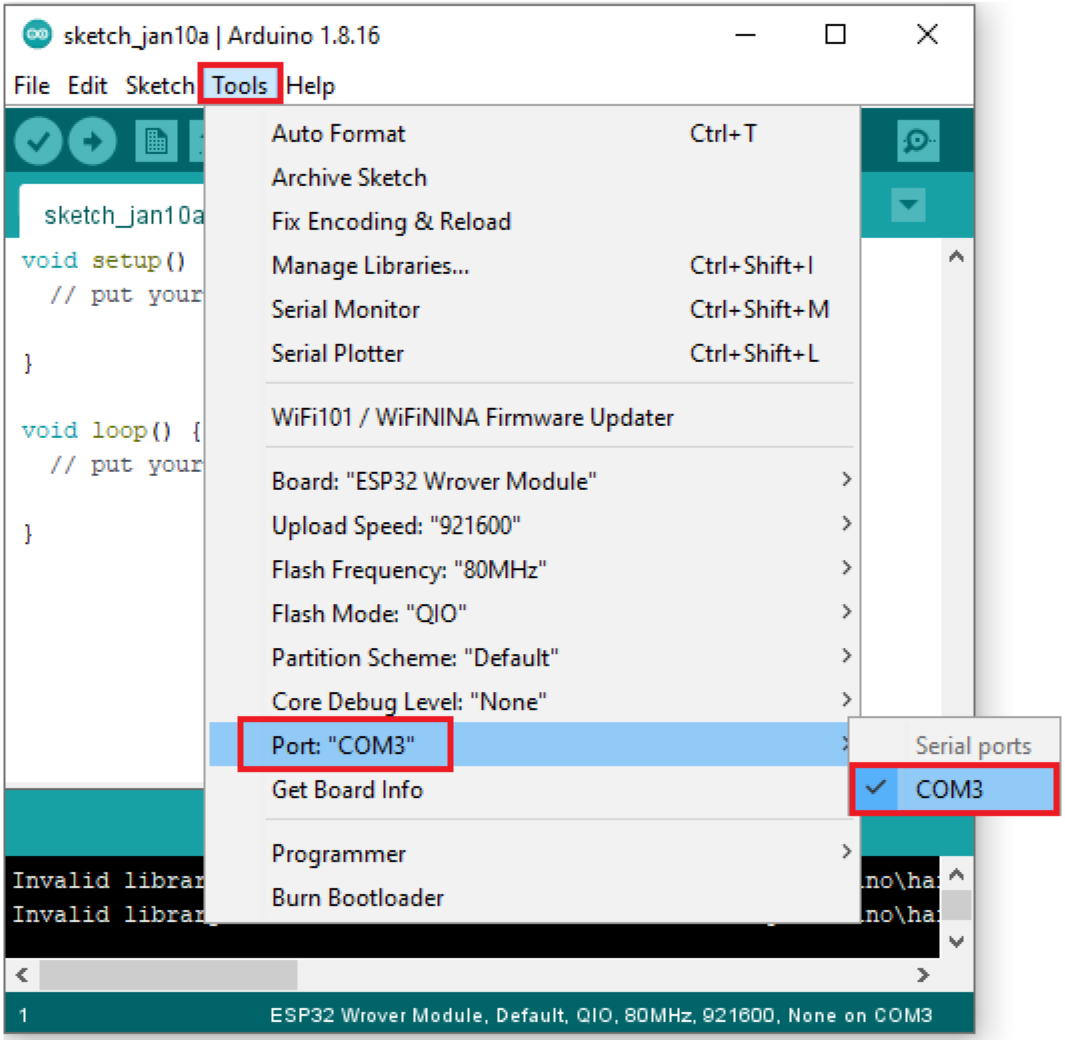


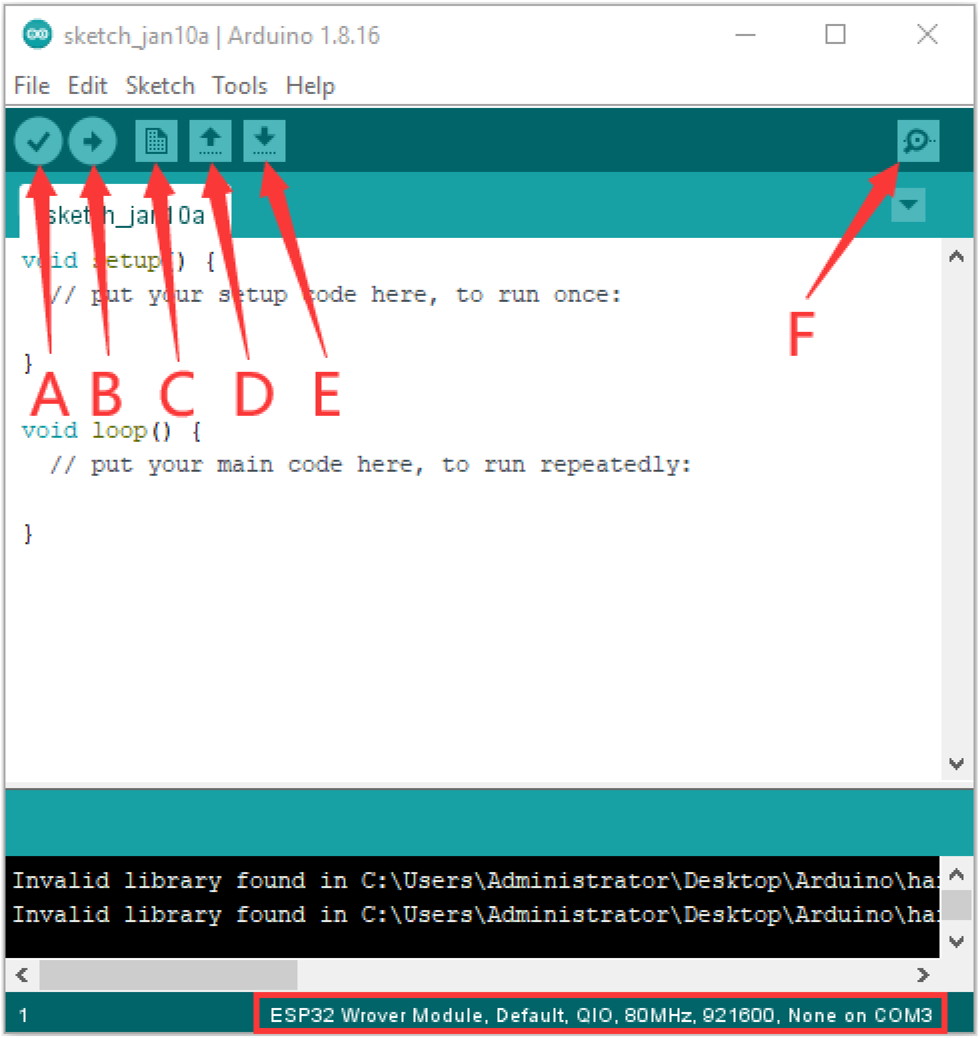
3). Set the board type as follows;



4). Then select the correct COM port (you can see the corresponding COM port after the driver is successfully installed).


Device Manager
File Action View Help
DESKTOP-98OK7TG
Audio inputs and outputs
Computer
Disk drives
Display adapters
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Silicon Labs CP210x USB to UART Bridge (COM3)
Print queues
Processors
Software components
Software devices
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers


sketch_jan10a | Arduino 1.8.16
File Edit Sketch Tools Help
sketch_jan10a
void setup()
// put your
}
void loop() {
// put your
}
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Wrover Module"
Upload Speed: "921600"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Partition Scheme: "Default"
Core Debug Level: "None"
Port: "COM3"
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM3
Invalid librar
Invalid librar
1
ESP32 Wrover Module, Default, QIO, 80MHz, 921600, None on COM3



A- Used to verify whether there is any compiling mistakes or not.

B- Used to upload the sketch to your Arduino board.

C- Used to create shortcut window of a new sketch.

D- Used to directly open an example sketch.

E- Used to save the sketch.

F- Used to send the serial data received from board to the serial monitor.

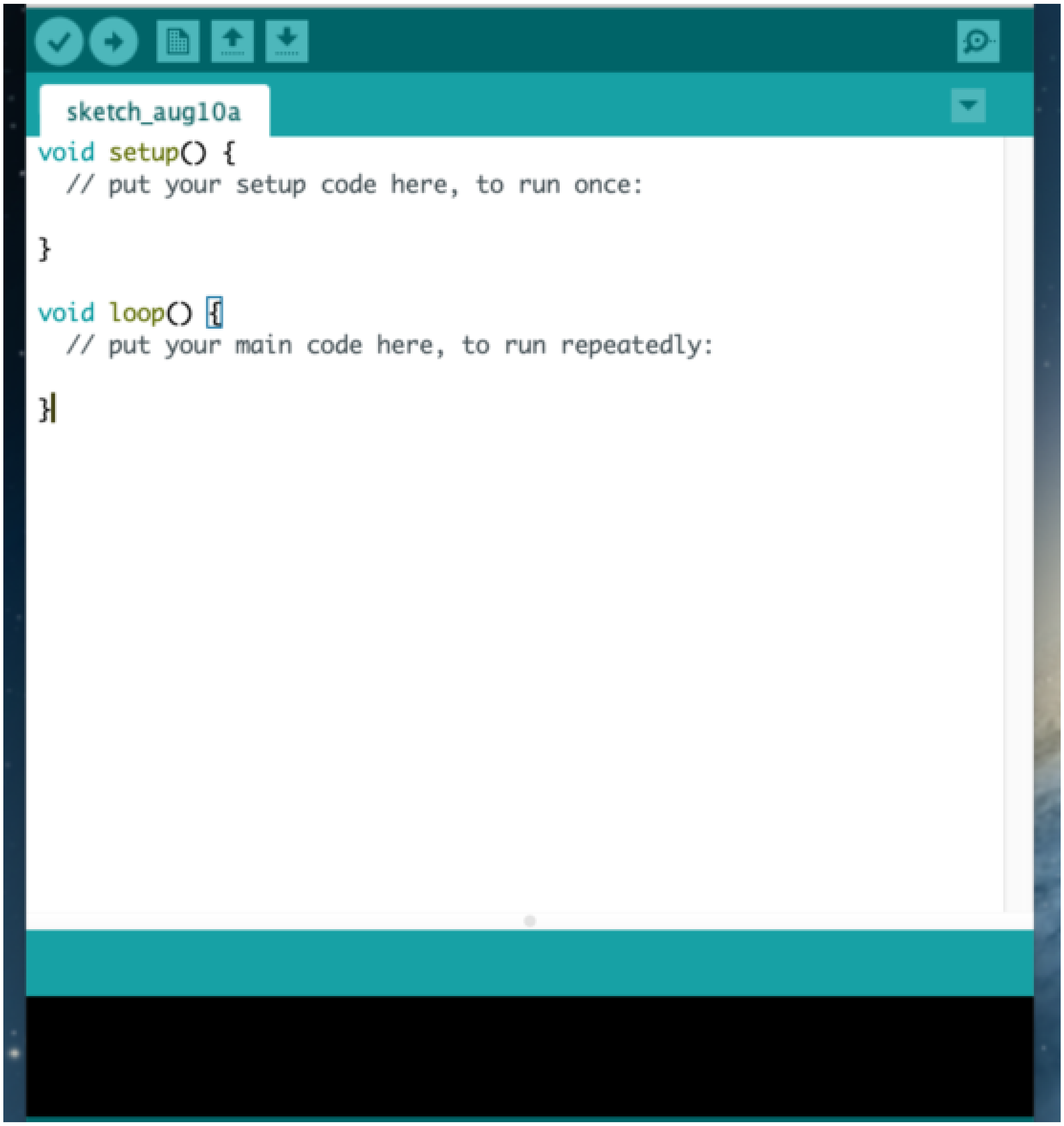
6.1.2 2. Mac System:

2.1. Download Arduino IDE:

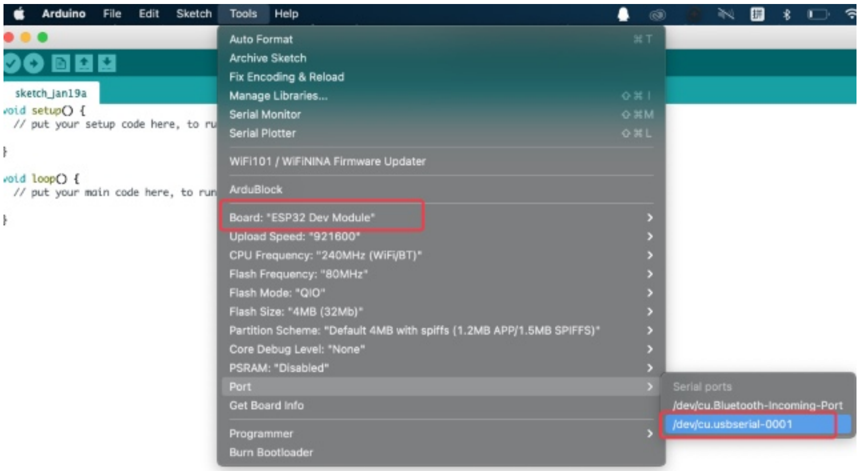
2.2. How to install the CP2102 driver

If you have installed the driver, just skip it.

1). Connect the ESP32 board to your computer, and open Arduino IDE.



2). Click "**Tools→Board:ESP32 Dev Module**" and "**/dev/cu.usbserial-0001**".



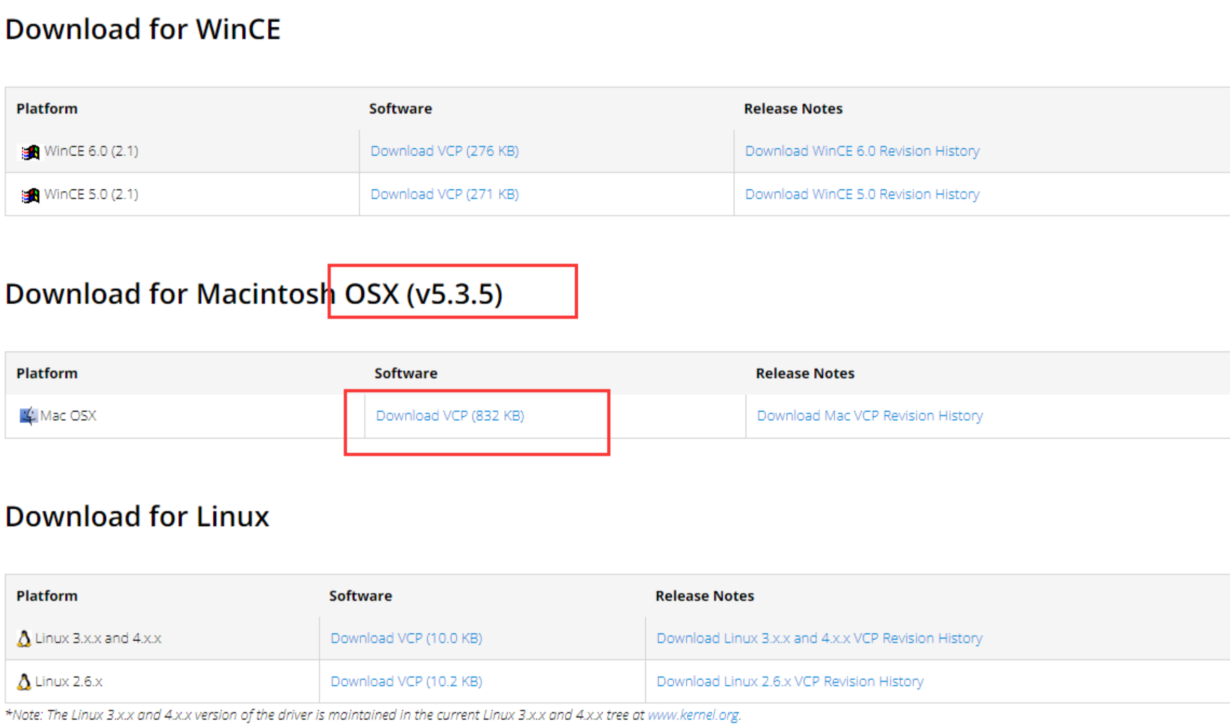
3). Click  to upload code.

Note: If code is uploaded unsuccessfully, you need to install driver of CP2102, please continue to follow the instructions as below:

Download the driver of CP2102:

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Select Mac OSX edition, as shown below;



Download for WinCE

Platform	Software	Release Notes
WinCE 6.0 (2.1)	Download VCP (276 KB)	Download WinCE 6.0 Revision History
WinCE 5.0 (2.1)	Download VCP (271 KB)	Download WinCE 5.0 Revision History

Download for Macintosh OSX (v5.3.5)

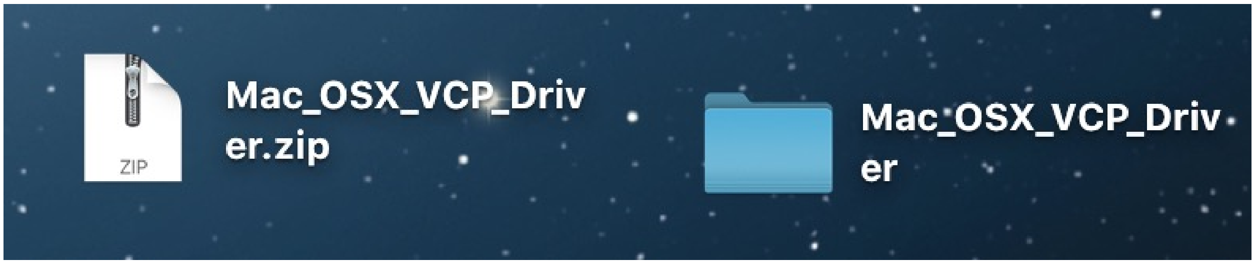
Platform	Software	Release Notes
Mac OSX	Download VCP (832 KB)	Download Mac VCP Revision History

Download for Linux

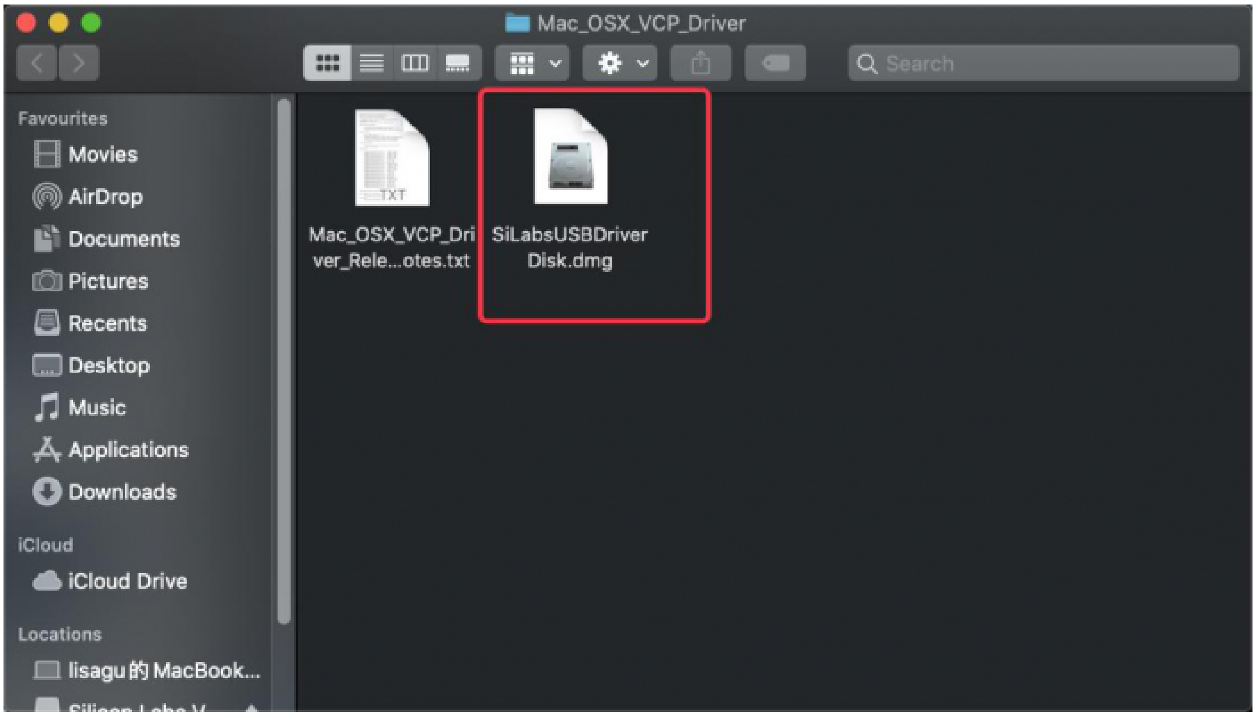
Platform	Software	Release Notes
Linux 3.x.x and 4.x.x	Download VCP (10.0 KB)	Download Linux 3.x.x and 4.x.x VCP Revision History
Linux 2.6.x	Download VCP (10.2 KB)	Download Linux 2.6.x VCP Revision History

*Note: The Linux 3.x.x and 4.x.x version of the driver is maintained in the current Linux 3.x.x and 4.x.x tree at www.kernel.org.

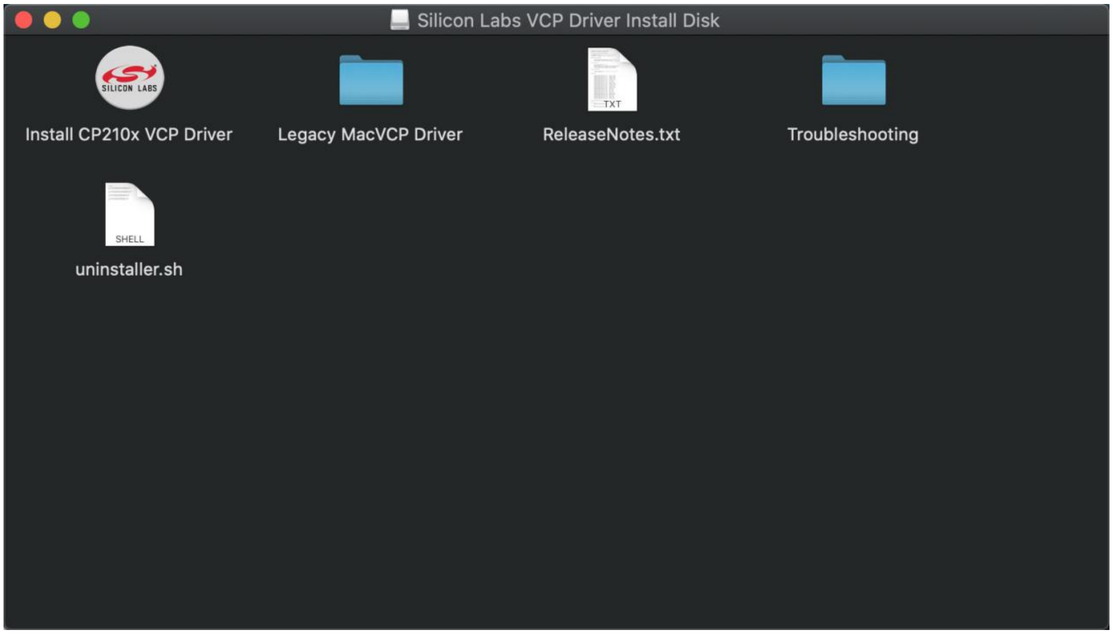
Unzip the downloaded package.



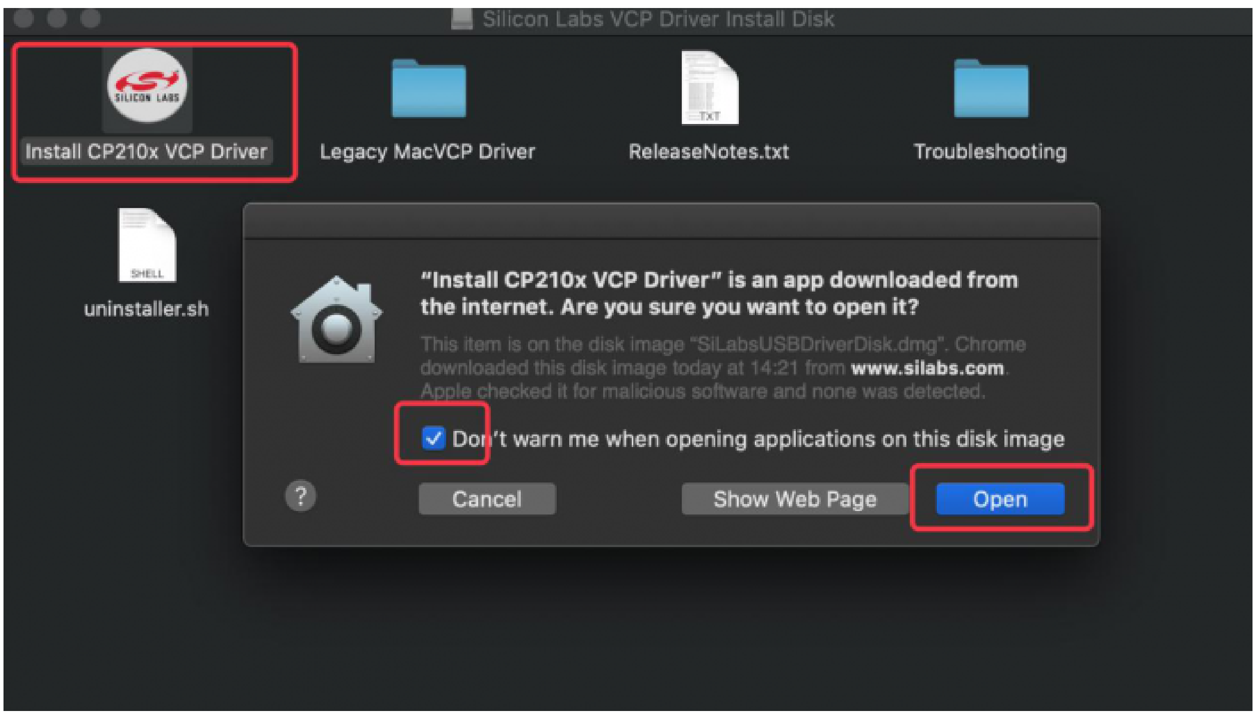
Open folder and double-click "**SiLabsUSBDriverDisk.dmg**" file.



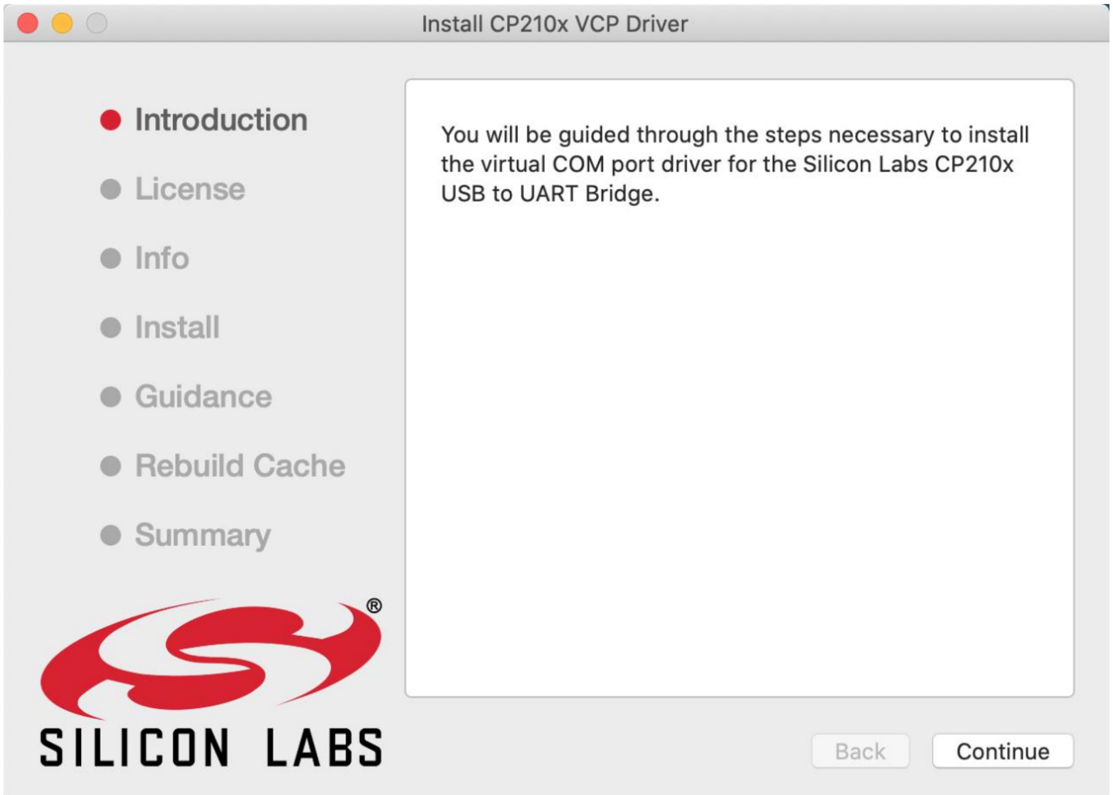
You will view the following files as follows:



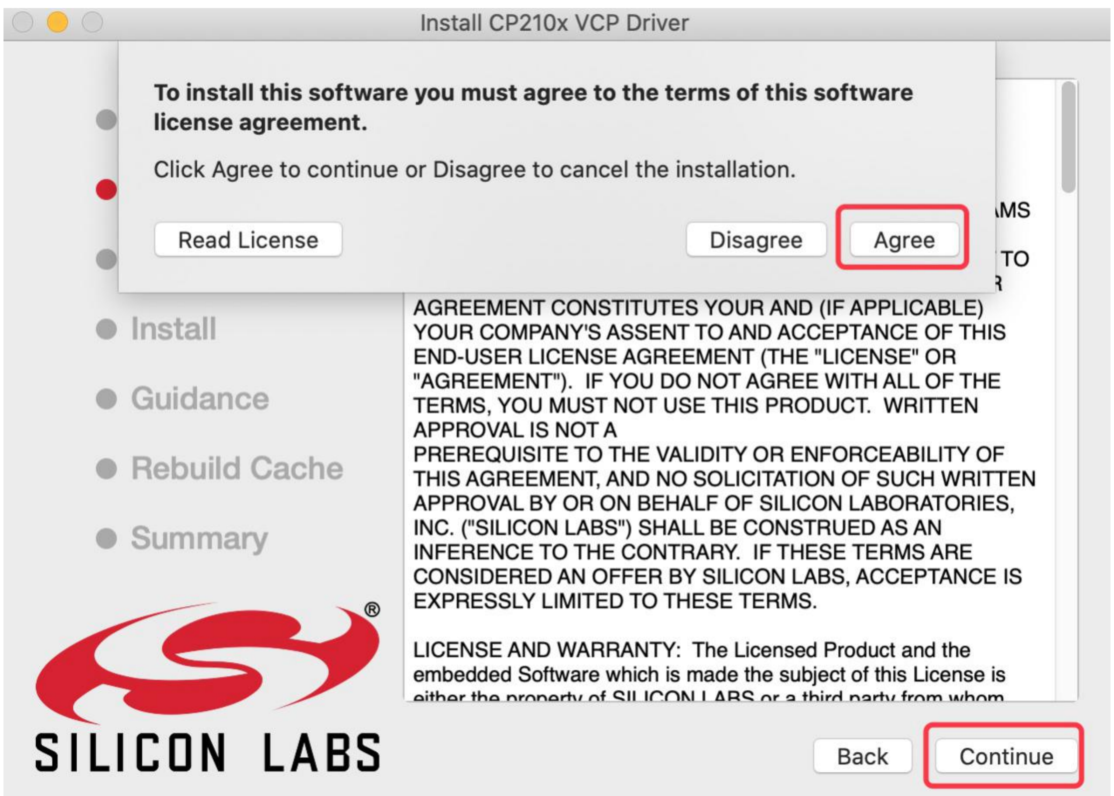
Double-click "**Install CP210x VCP Driver**", tick "**Don't warn me when opening application on this disk image**" and tap "**Open**".



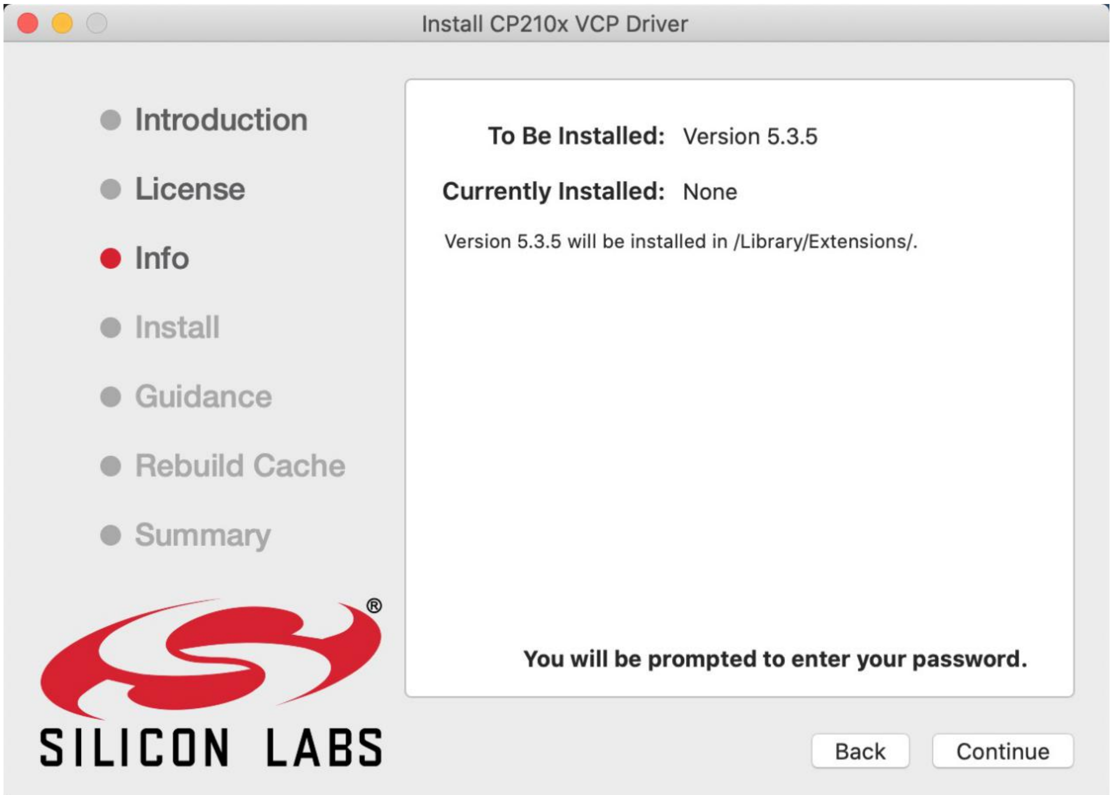
Click "**Continue**".

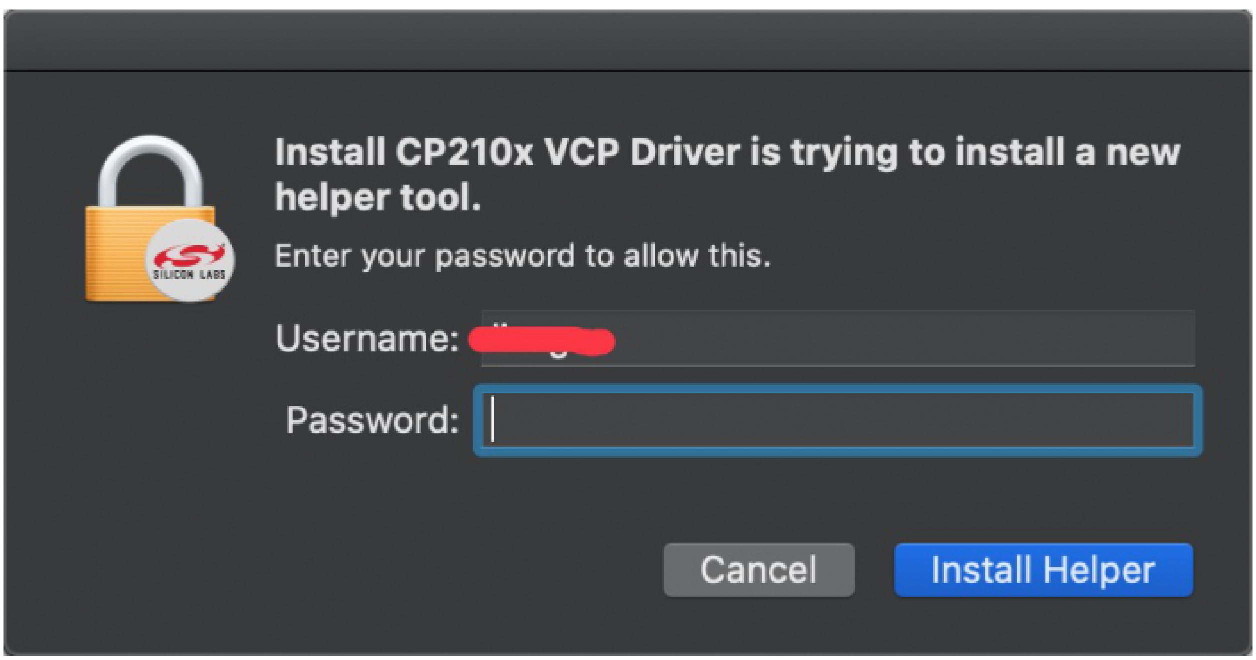


Tap "**Agree**" and "**Continue**".

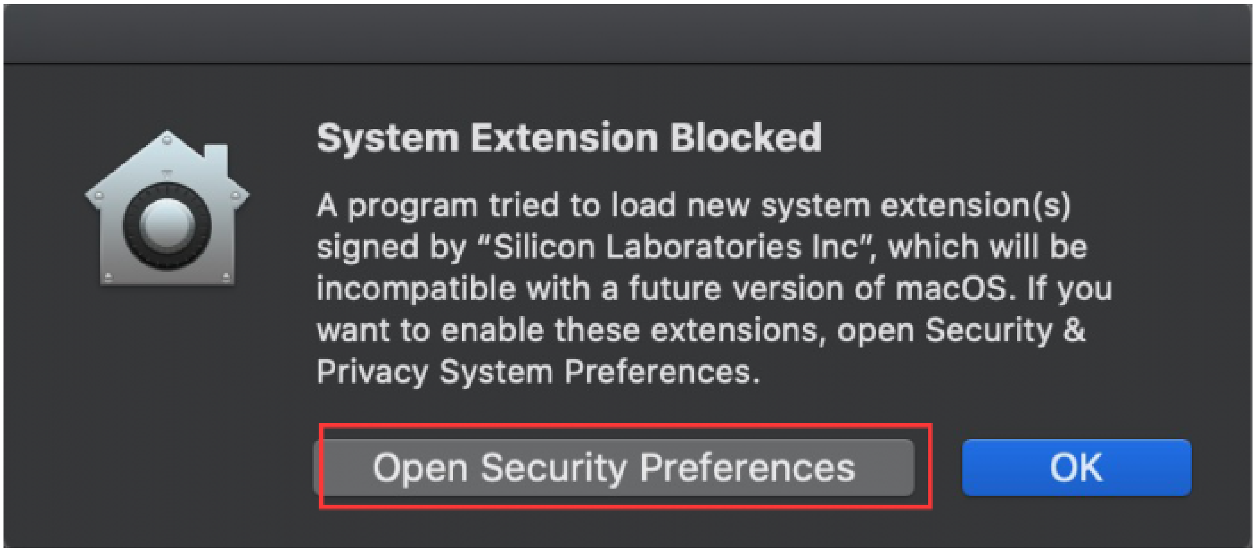


Click "**Continue**" and input your password.

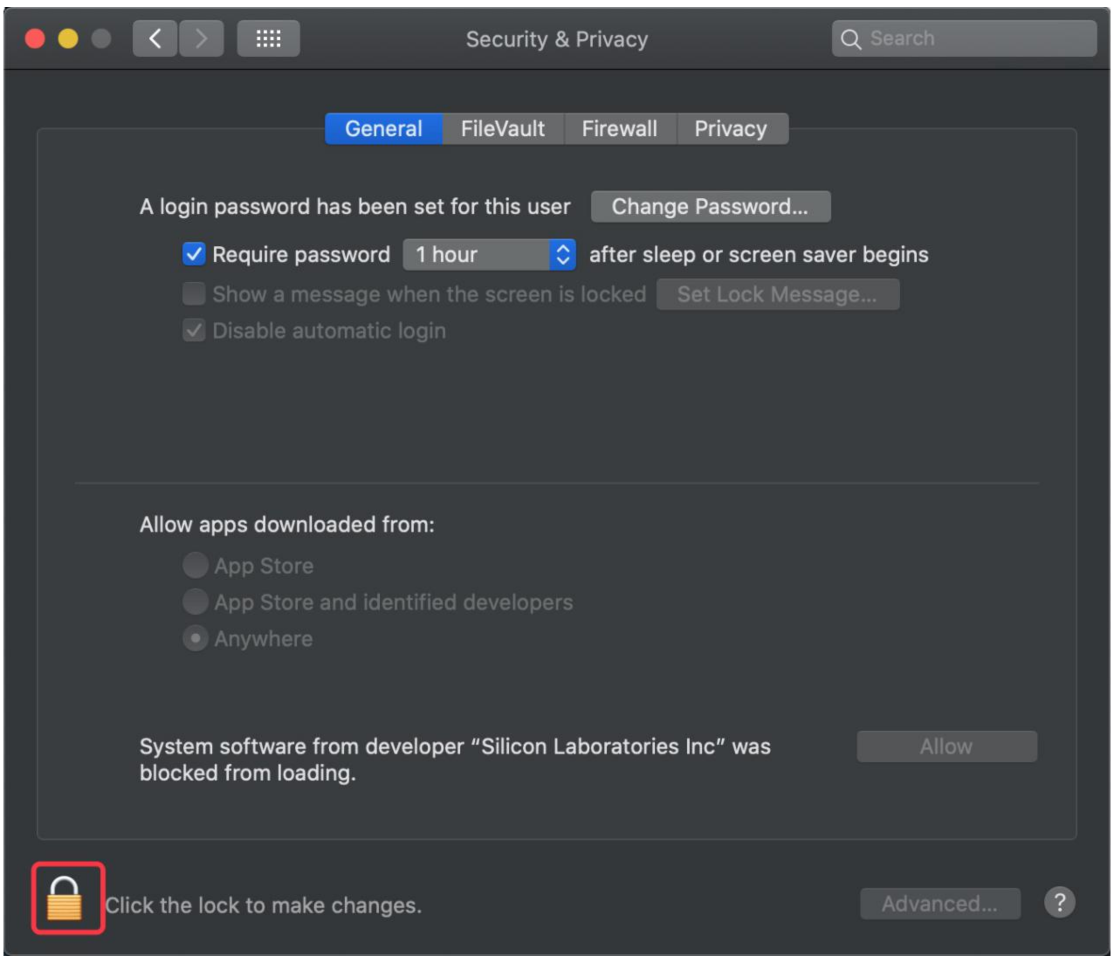




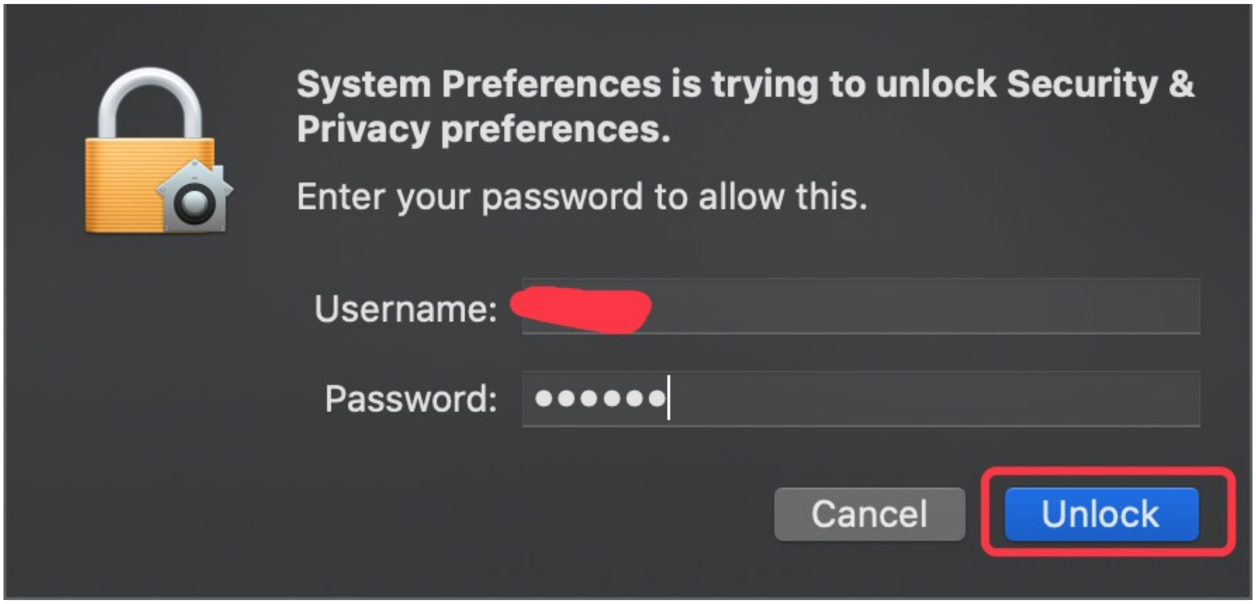
Select "**Open Security Preferences**".



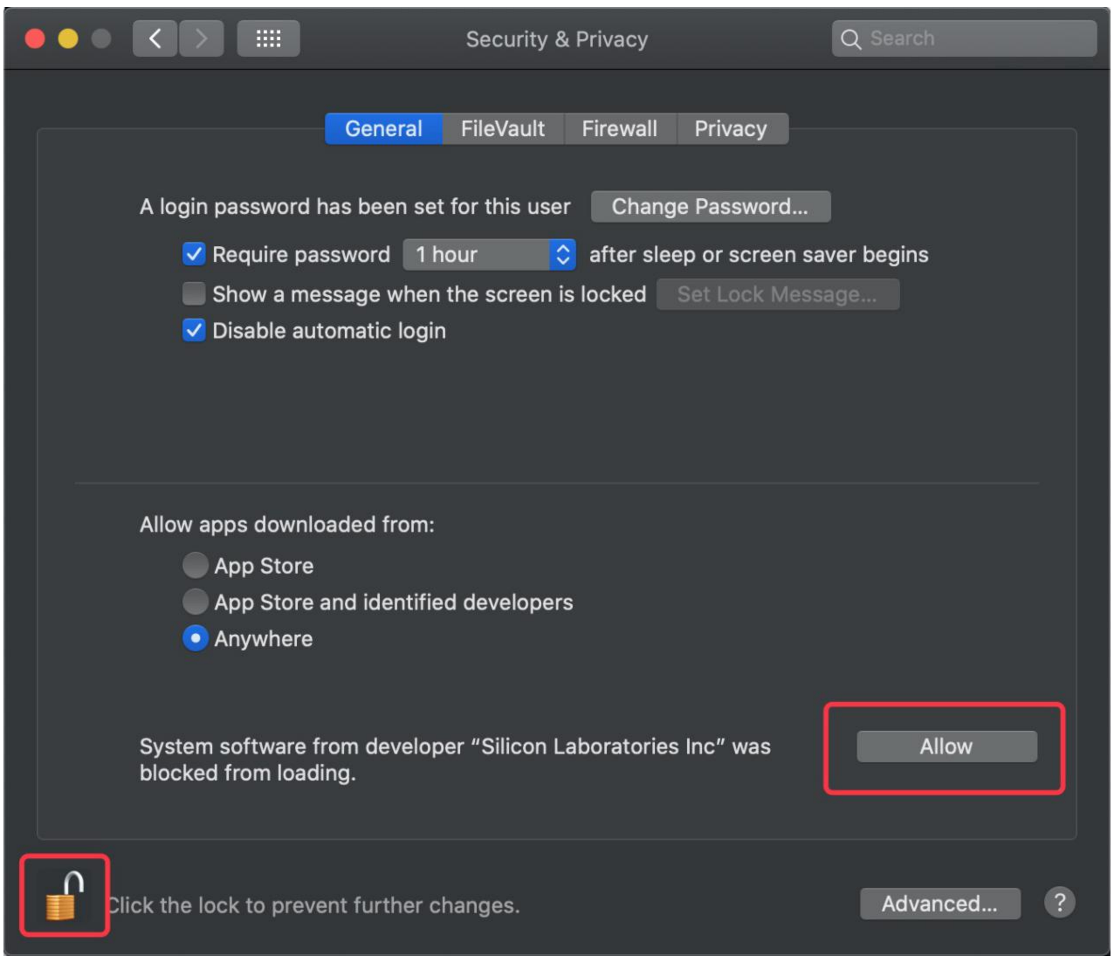
Click the **lock** to unlock "**security & privacy preference**".



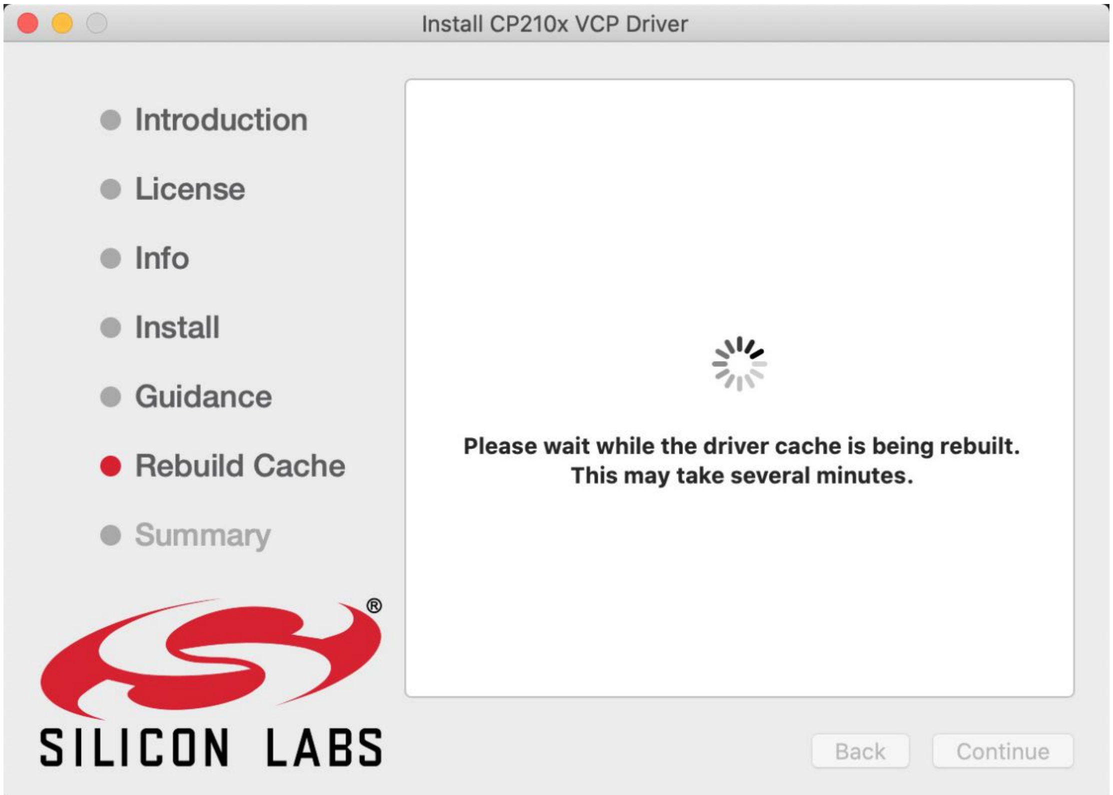
Tap **Unlock and** enter **your Username and password**



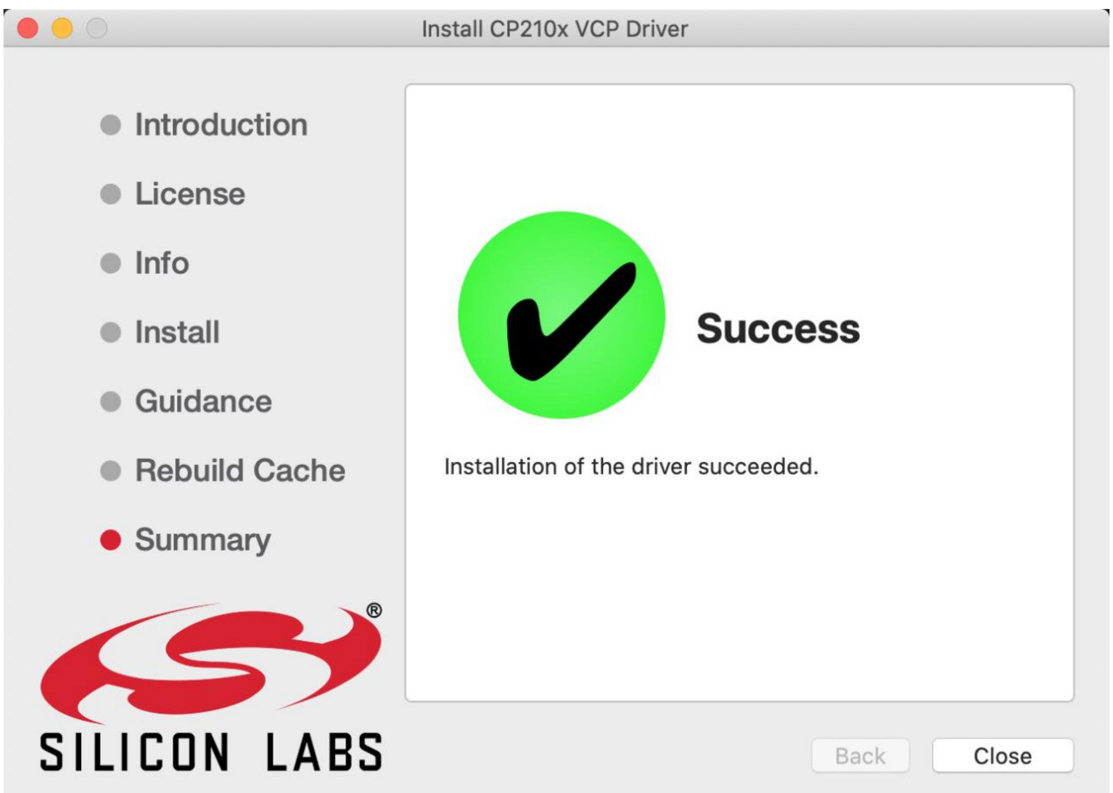
Then click "**Allow**".



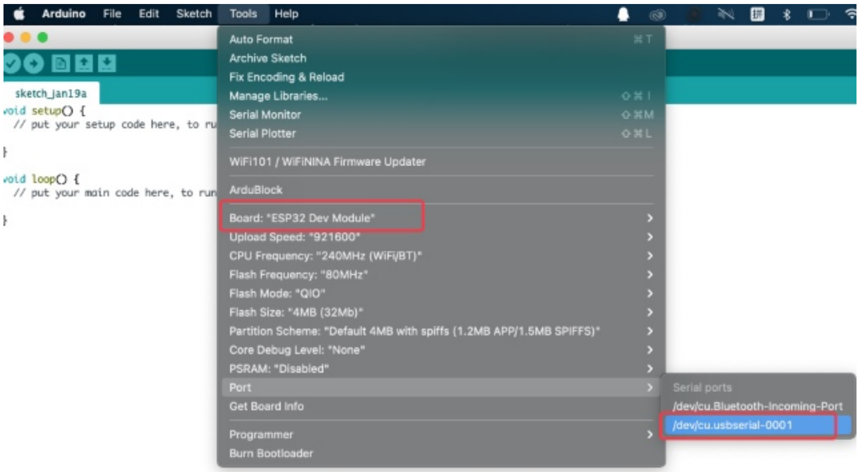
Back to installation page, and wait to install.



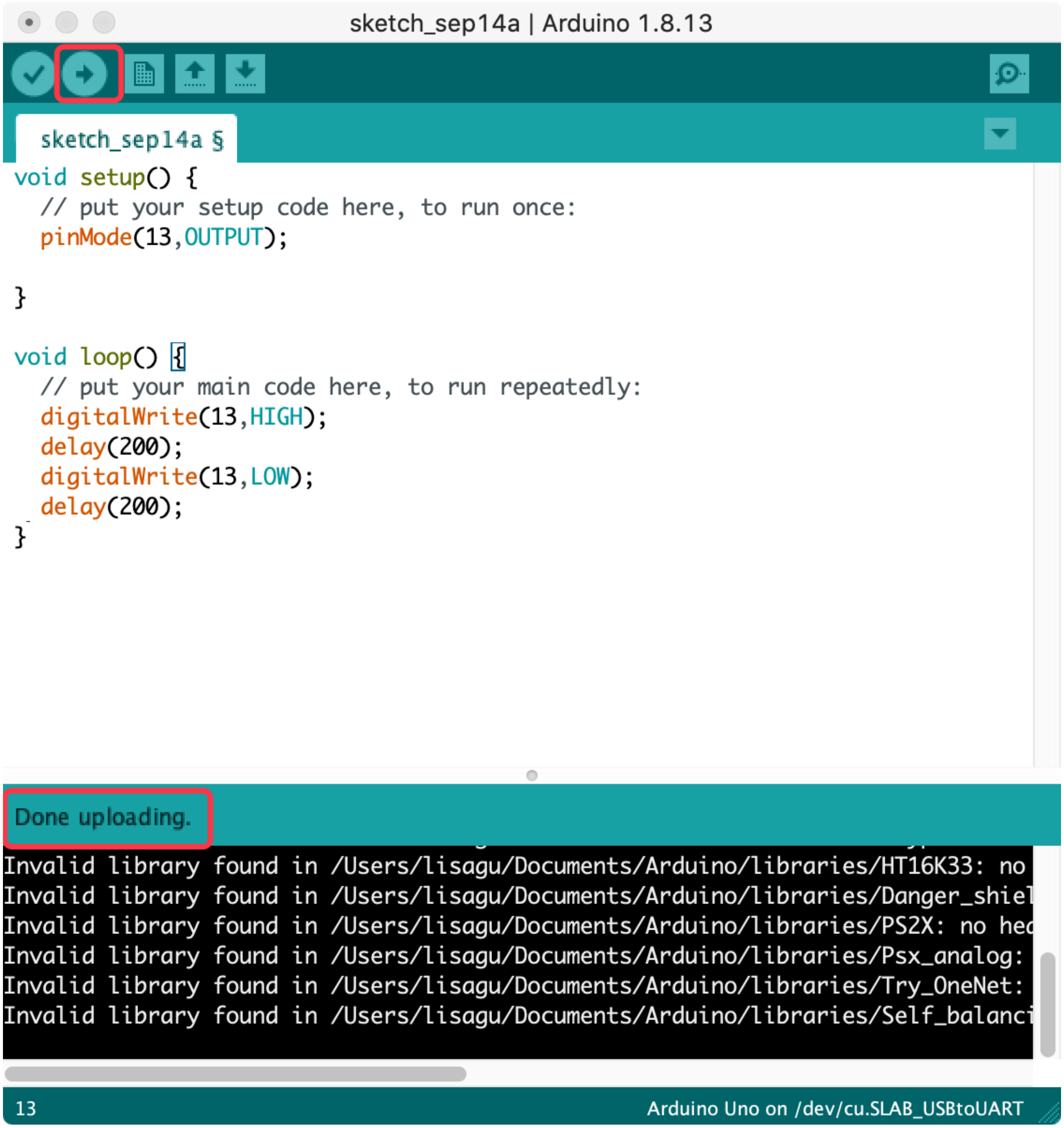
Successfully installed.



Then enter ArduinoIDE, click **Tools** and select Board **ESP32 Dev Module** and the serial port is“**/dev/cu.SLAB_USBtoUAPT.**



Click to upload code and show “**Done uploading**”.



6.1.3 3. How to Add Libraries? :

3.1. What are Libraries ? :

Libraries are a collection of code that make it easy for you to connect sensors,displays, modules, etc.

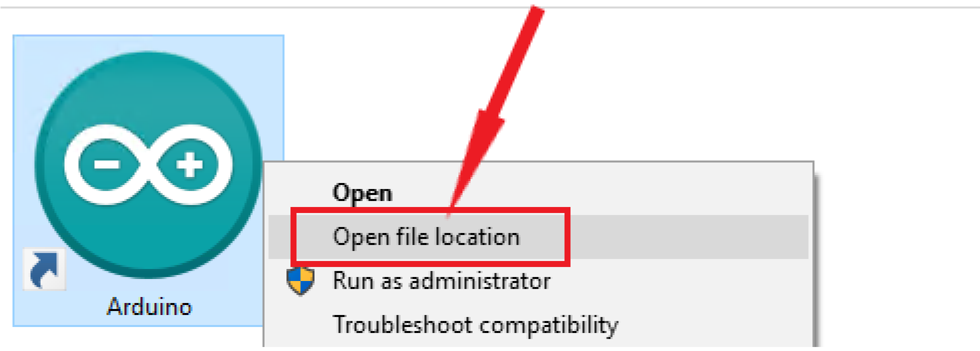
For example, the built-in LiquidCrystal library helps talk to LCD displays. There are hundreds of additional libraries available on the Internet for download.

The built-in libraries and some of these additional libraries are listed in the reference. https://www.arduino.cc/en/Reference/Libraries

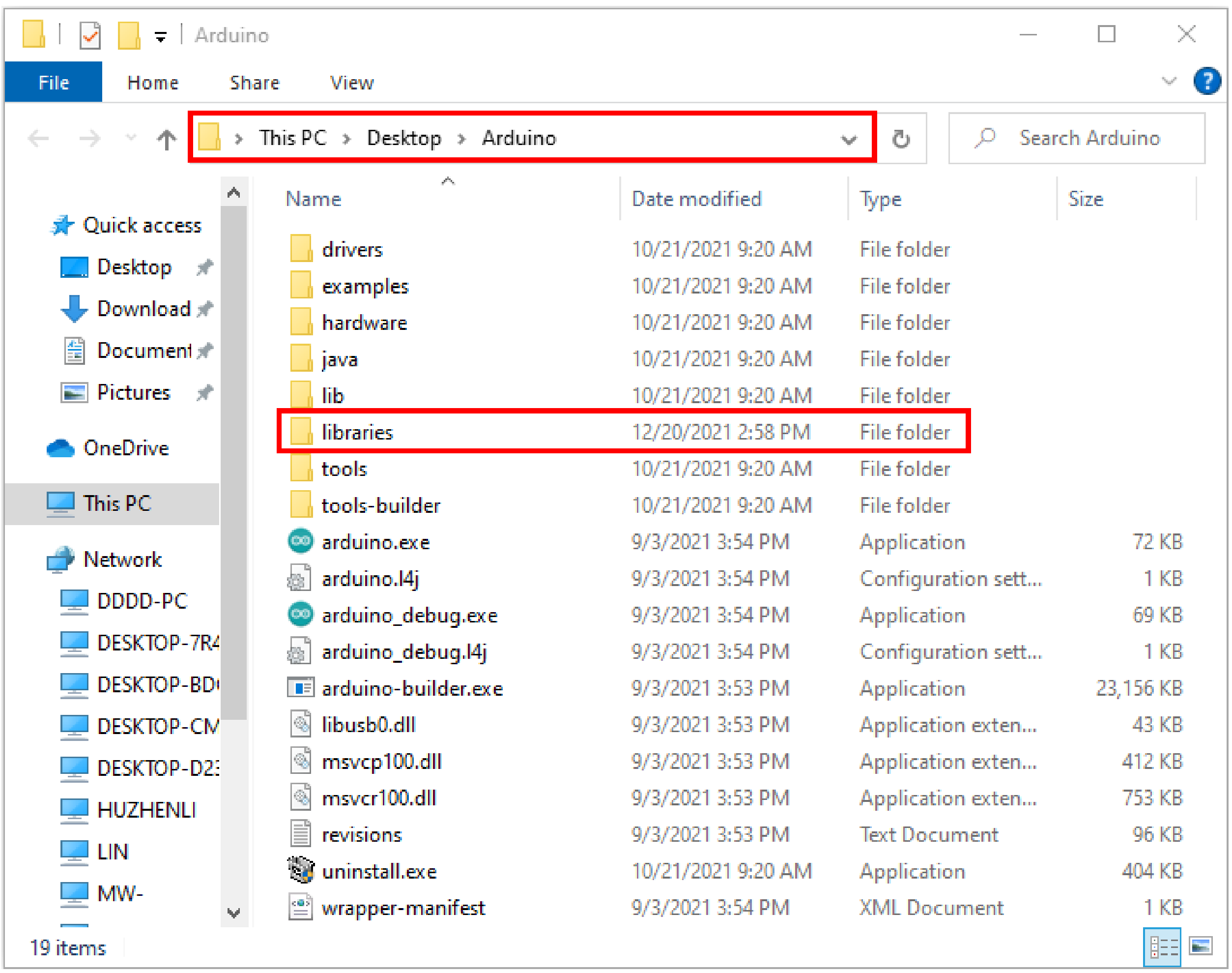
3.2. How to Install a Library ? :

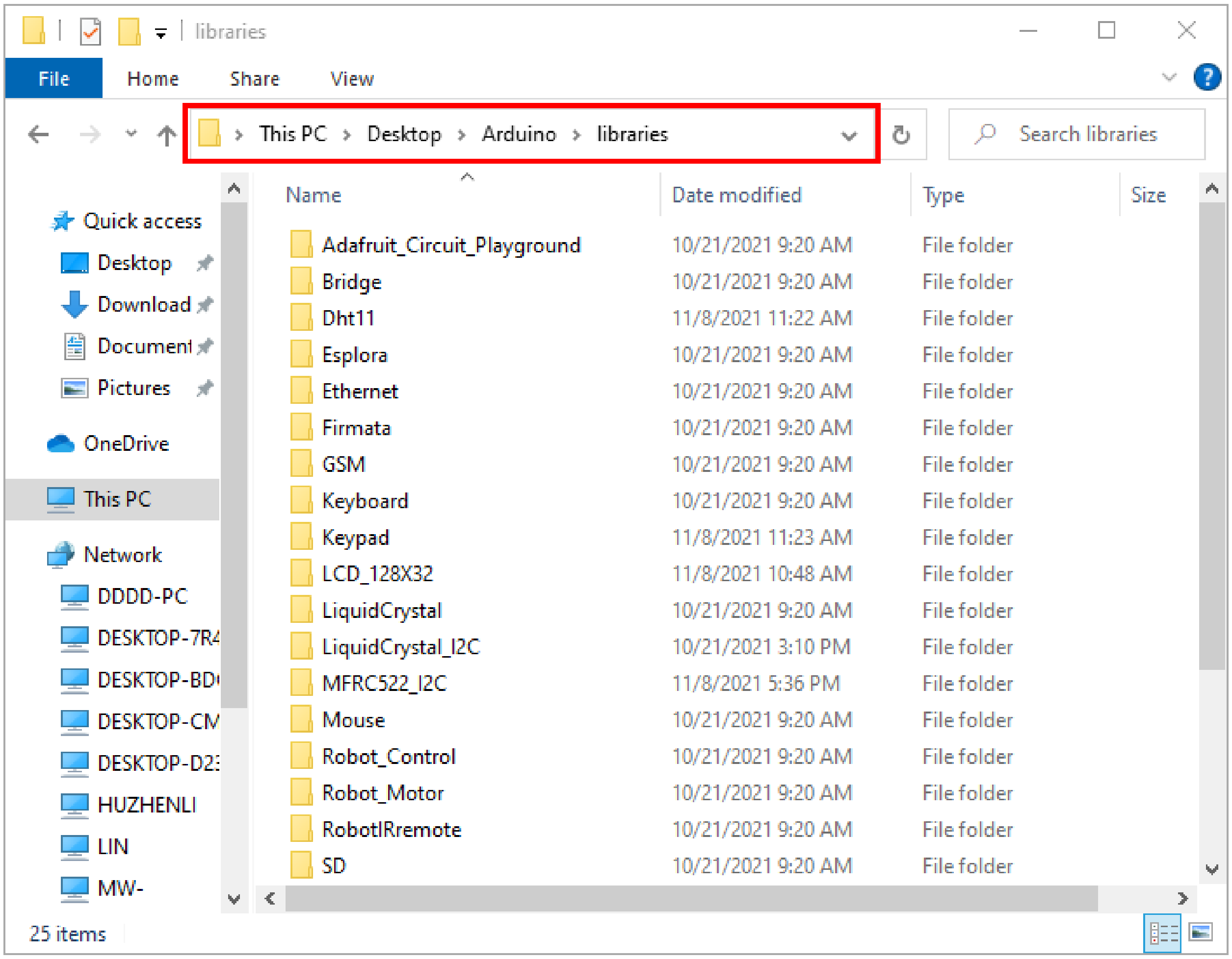
Here we will introduce the most simple way to add libraries .

Step 1: After downloading well the Arduino IDE, you can right-click the icon of Arduino IDE. Find the option "**Open file location**".

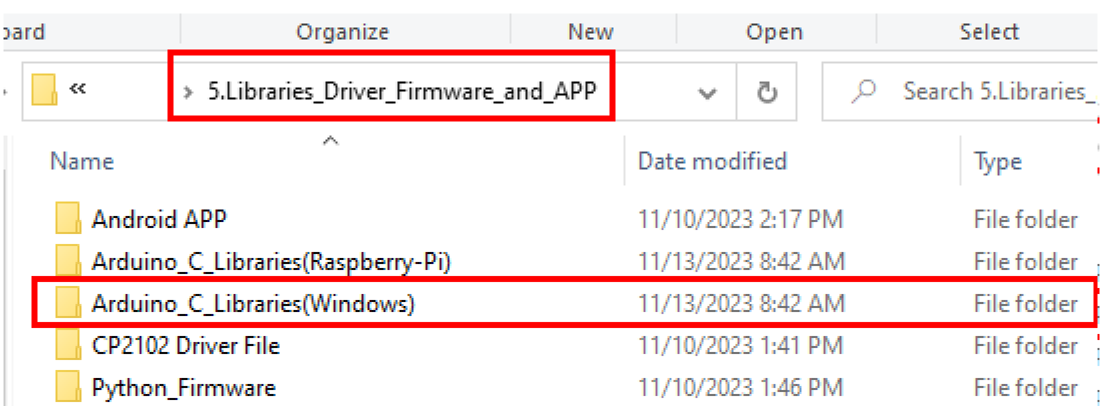


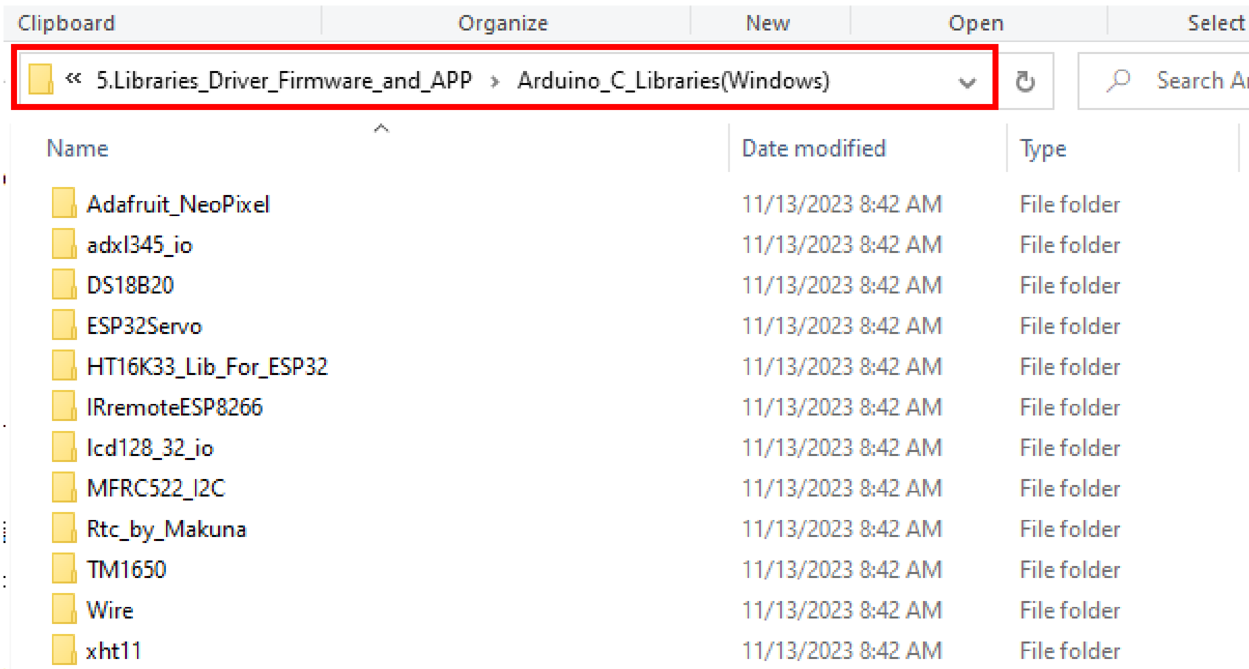
Step 2: Click Open file location >**libraries**.

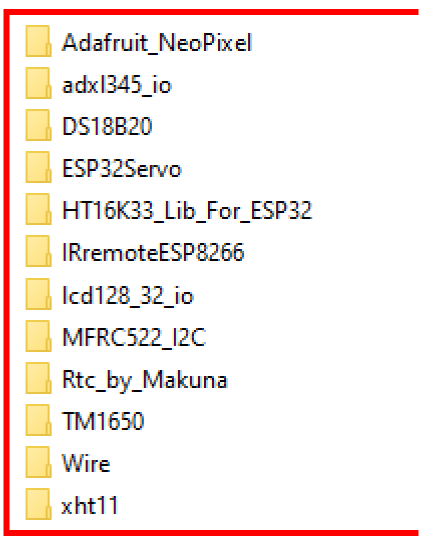




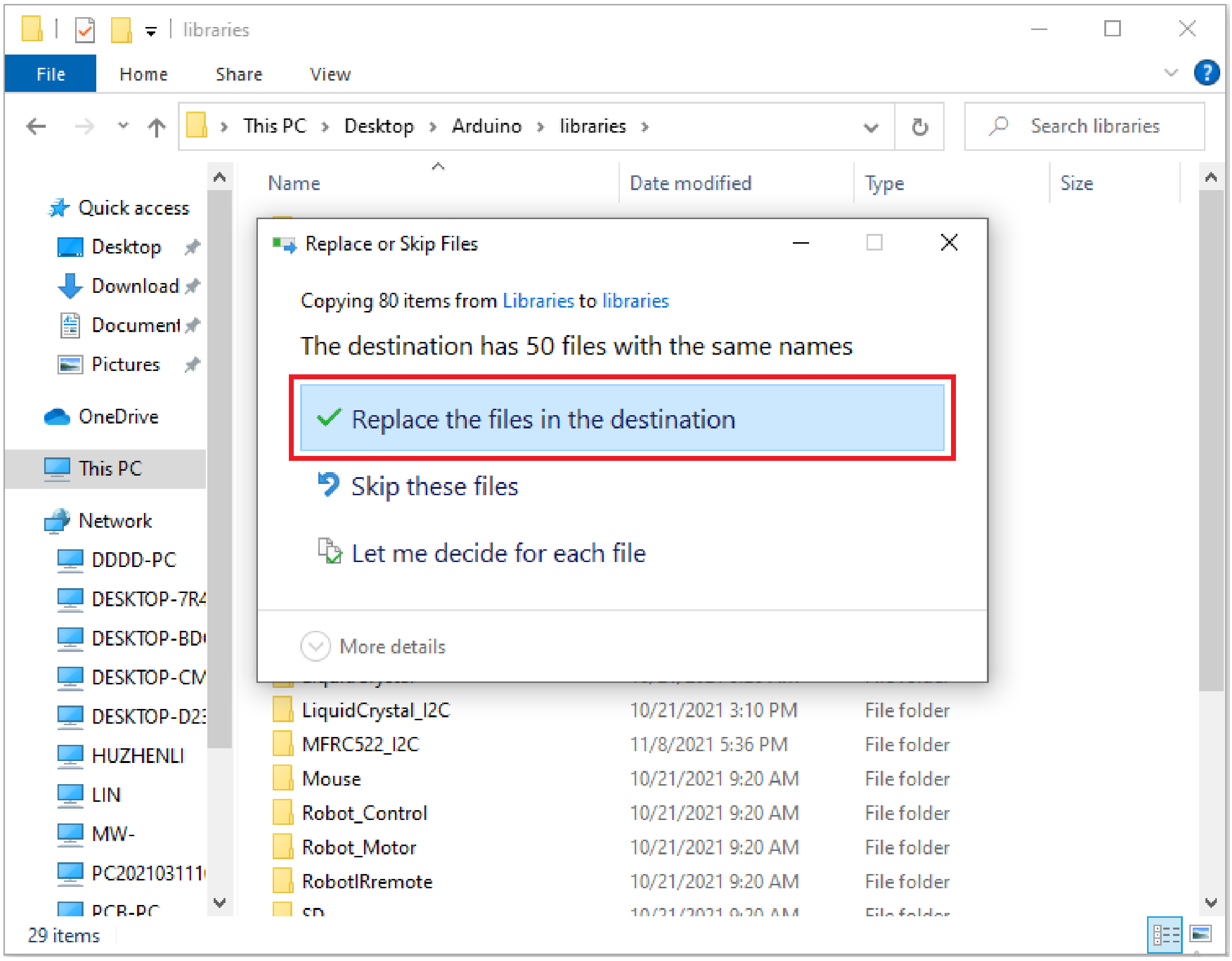
Step 3: Next, find out the "**libraries**" folder.

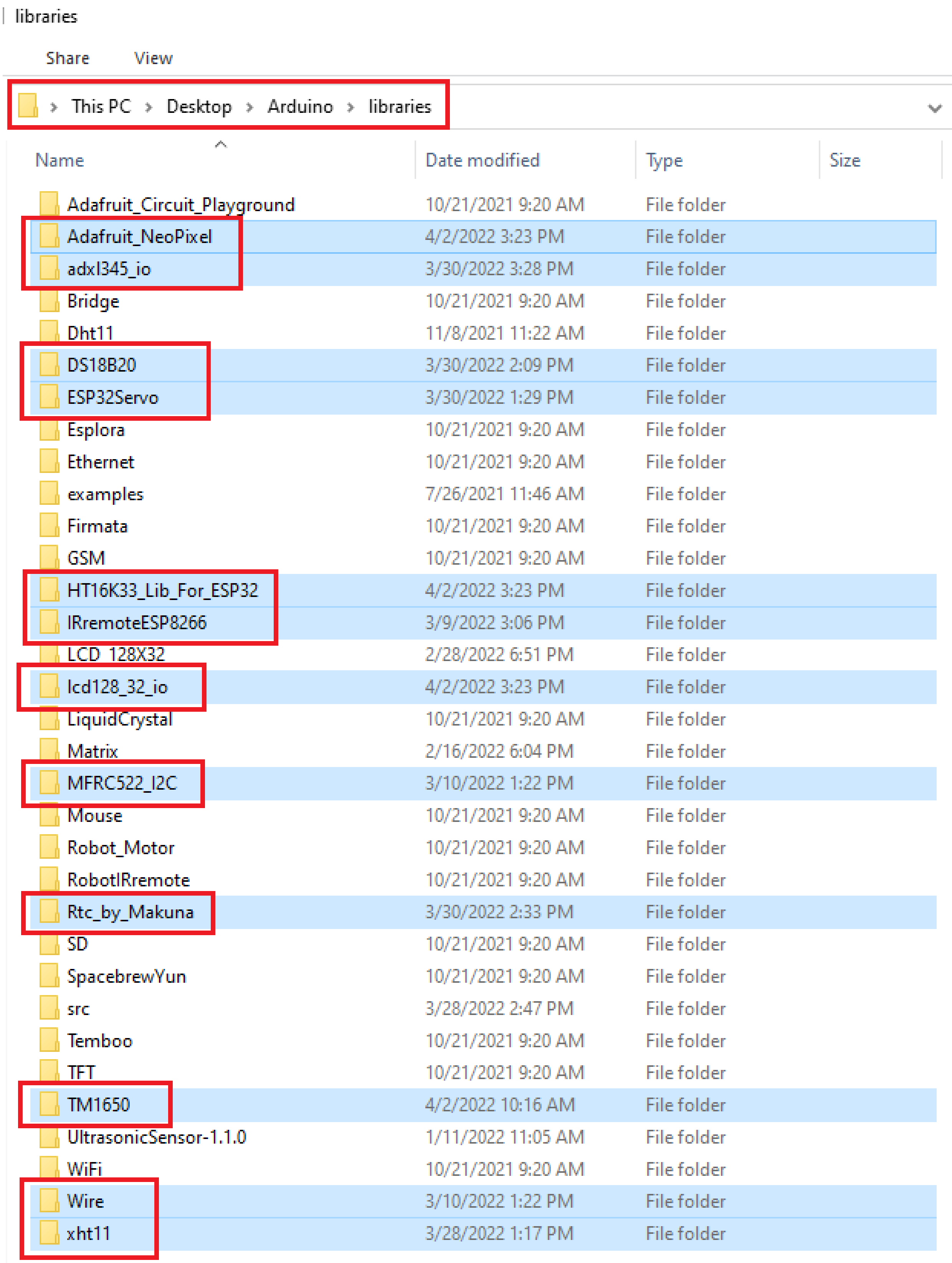




Copy 

in the libraries folder of Arduino. Then click "**Replace the files in the destination**".


libraries
File
Home
Share
View
This PC > Desktop > Arduino > libraries >
Search libraries
Name
Date modified
Type
Size
Quick access
Desktop
Download
Document
Pictures
OneDrive
This PC
Network
DDDD-PC
HUZHENLI
LIN
MW-
Replace or Skip Files
Copying 80 items from Libraries to libraries
The destination has 50 files with the same names
Replace the files in the destination
Skip these files
Let me decide for each file
More details
LiquidCrystal_I2C 10/21/2021 3:10 PM File folder
MFRC522_I2C 11/8/2021 5:36 PM File folder
Mouse 10/21/2021 9:20 AM File folder
Robot_Control 10/21/2021 9:20 AM File folder
Robot_Motor 10/21/2021 9:20 AM File folder
RobotIRremote 10/21/2021 9:20 AM File folder
29 items


libraries
Share
View
This PC > Desktop > Arduino > libraries
Name
Date modified
Type
Size
Adafruit_Circuit_Playground 10/21/2021 9:20 AM File folder
Adafruit_NeoPixel 4/2/2022 3:23 PM File folder
adxl345_io 3/30/2022 3:28 PM File folder
Bridge 10/21/2021 9:20 AM File folder
Dht11 11/8/2021 11:22 AM File folder
DS18B20 3/30/2022 2:09 PM File folder
ESP32Servo 3/30/2022 1:29 PM File folder
Esplora 10/21/2021 9:20 AM File folder
Ethernet 10/21/2021 9:20 AM File folder
examples 7/26/2021 11:46 AM File folder
Firmata 10/21/2021 9:20 AM File folder
GSM 10/21/2021 9:20 AM File folder
HT16K33_Lib_For_ESP32 4/2/2022 3:23 PM File folder
IRremoteESP8266 3/9/2022 3:06 PM File folder
LCD_128X32 2/28/2022 6:51 PM File folder
lcd128_32_io 4/2/2022 3:23 PM File folder
LiquidCrystal 10/21/2021 9:20 AM File folder
Matrix 2/16/2022 6:04 PM File folder
MFRC522_I2C 3/10/2022 1:22 PM File folder
Mouse 10/21/2021 9:20 AM File folder
Robot_Motor 10/21/2021 9:20 AM File folder
RobotIRremote 10/21/2021 9:20 AM File folder
Rtc_by_Makuna 3/30/2022 2:33 PM File folder
SD 10/21/2021 9:20 AM File folder
SpacebrewYun 10/21/2021 9:20 AM File folder
src 3/28/2022 2:47 PM File folder
Temboo 10/21/2021 9:20 AM File folder
TFT 10/21/2021 9:20 AM File folder
TM1650 4/2/2022 10:16 AM File folder
UltrasonicSensor-1.1.0 1/11/2022 11:05 AM File folder
WiFi 10/21/2021 9:20 AM File folder
Wire 3/10/2022 1:22 PM File folder
xht11 3/28/2022 1:17 PM File folder

6.2 2. Basic Projects

When we get the kit, we can see that there are 42 sensors/modules in the kit, which contain the corresponding ESP32 mainboard, ESP32 Expansion Board and wirings. Here, we will connect the 42 sensors individually to the ESP32 mainboard and the ESP32 Expansion Board using wirings. Then run the corresponding test code to test the function of each sensor separately. Our next lesson is to study the principles of individual modules/sensors from simple to complex as well as some extended applications of sensors to consolidate and deepen our understanding of the kits.

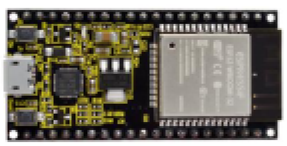
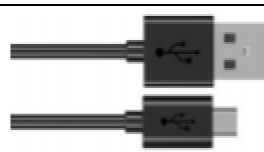
Note : When connecting the module/sensor wirings in the projects, the wiring method and position must be followed in the document. What's more, do not misconnect the power supply and signal pin, otherwise there may be no experimental results or damage to the modules/sensors.

6.2.1 Project 1: Hello World

Overview

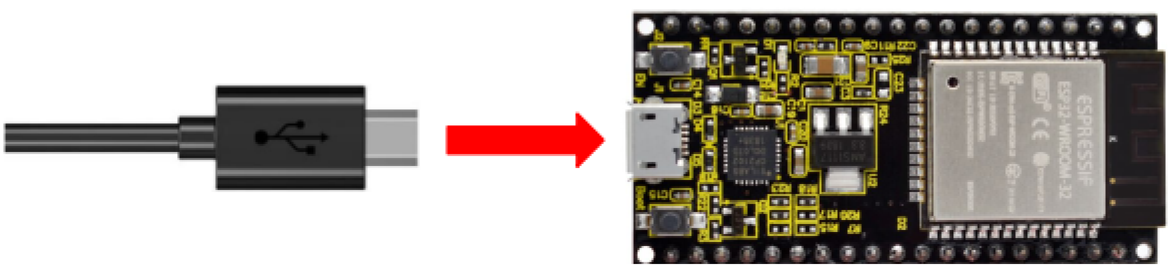
For ESP32 beginners, we will start with some simple things. In this project, you only need a ESP32 mainboard, a USB cable and a computer to complete the "Hello World!" project, which is a test of communication between the ESP32 mainboard and the computer as well as a primary project.

Components

	
ESP32*1	USB Cable*1

Wiring Diagram

In this project, we will use a USB cable to connect the ESP32 to a computer.



Test Code

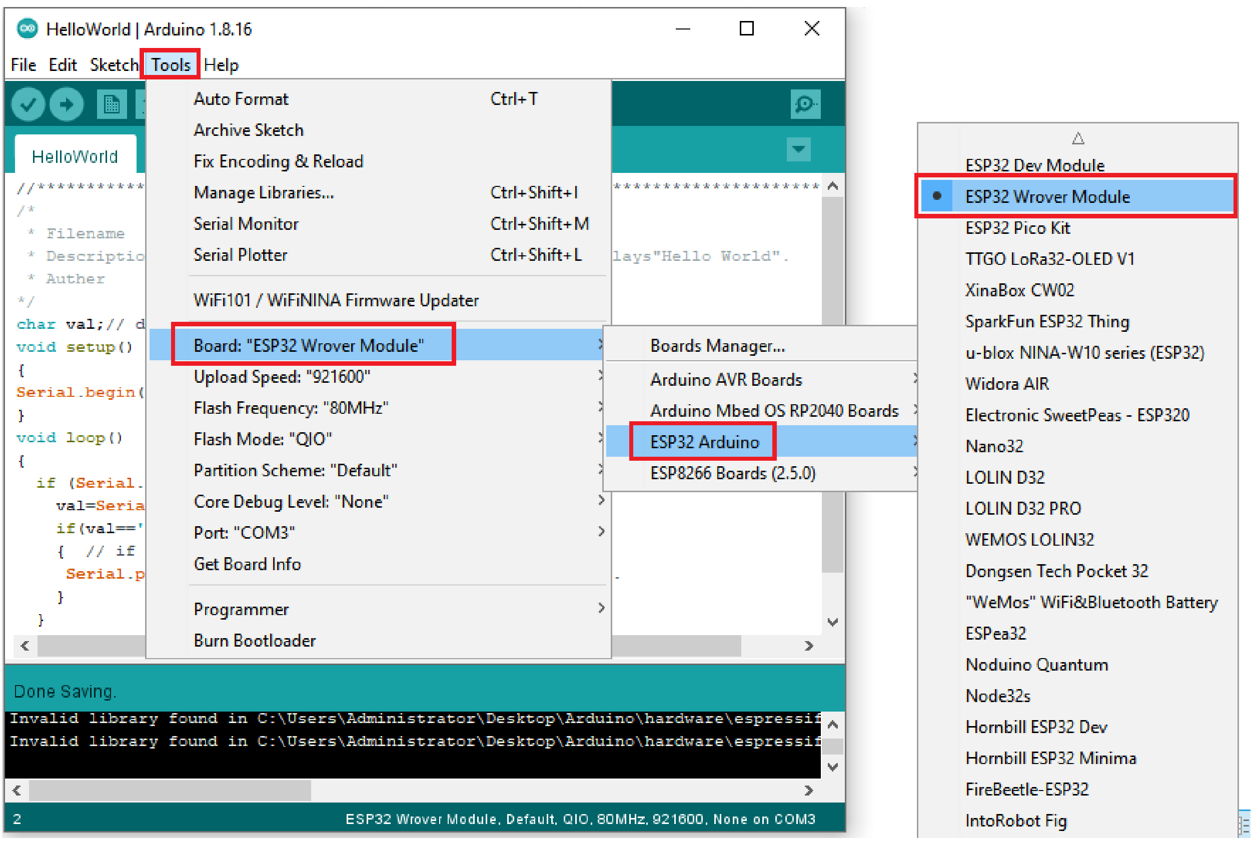
```
//**********************************************************************************
/*
 * Filename    : Hello World
 * Description : Enter the letter R,and the serial port displays"Hello World".
 * Auther      :http//www.keyestudio.com
*/
char val;// defines variable "val"
void setup()
{
```

(continues on next page)

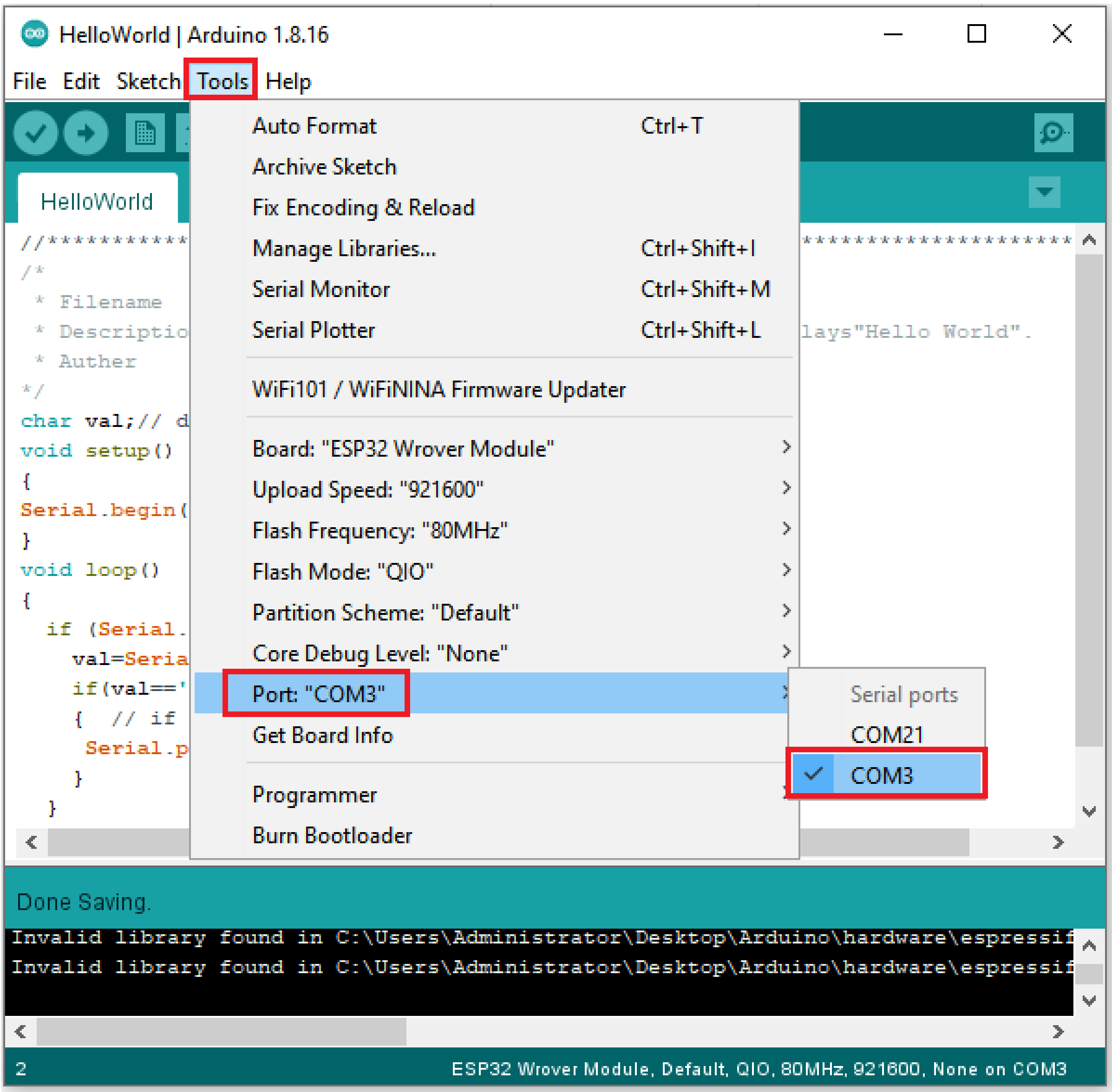
(continued from previous page)

```
Serial.begin(9600);// sets baudrate to 9600
}
void loop()
{
  if (Serial.available() > 0) {
    val=Serial.read();// reads symbols assigns to "val"
    if(val=='R')// checks input for the letter "R"
    {  // if so,
     Serial.println("Hello World!");// shows “Hello World !”.
    }
  }
}
//*************************************************************************************
```

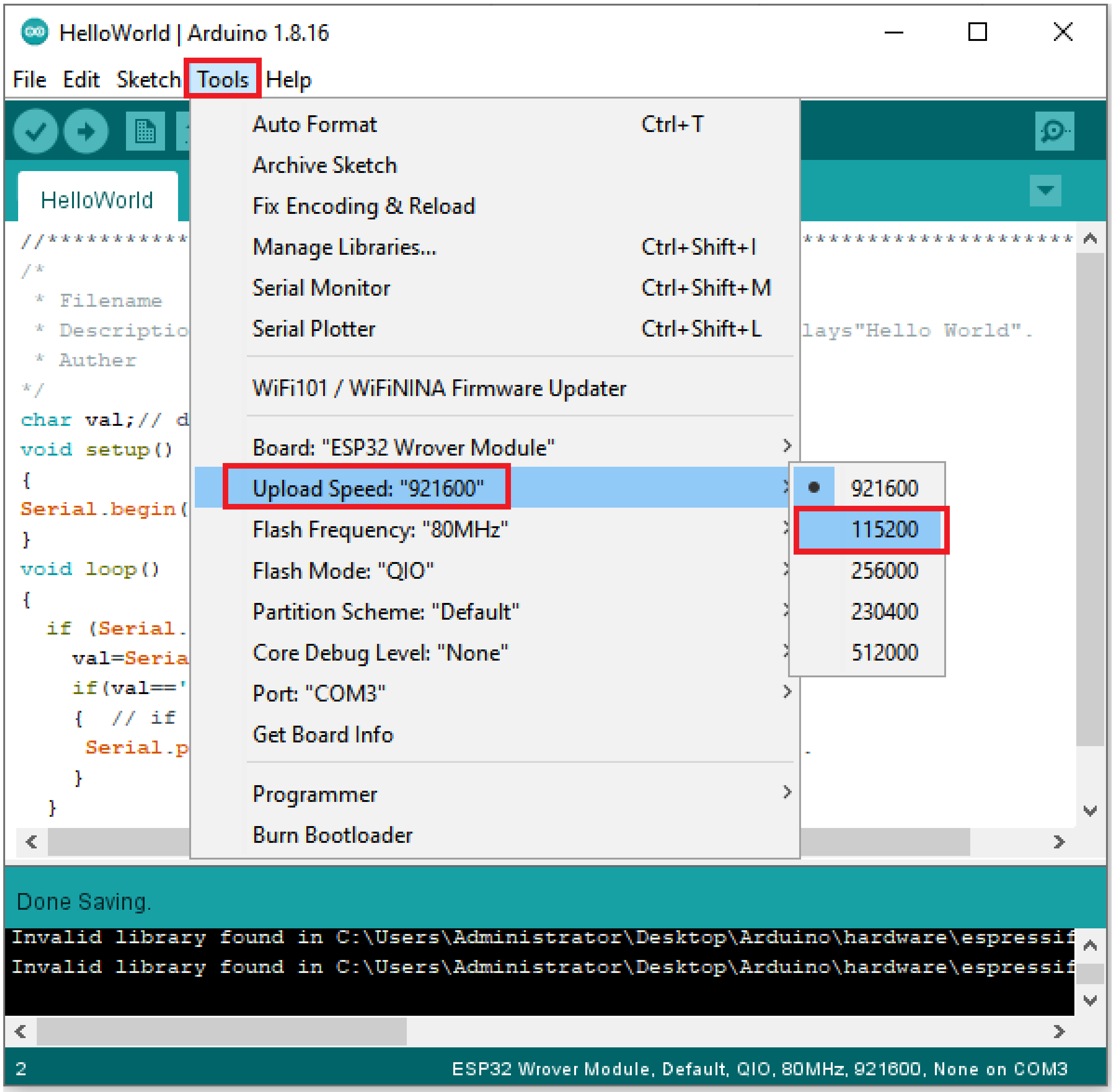
Before uploading the test code to the ESP32click “**Tools**” → “**Board**”select “**ESP32 Wrover Module**”.



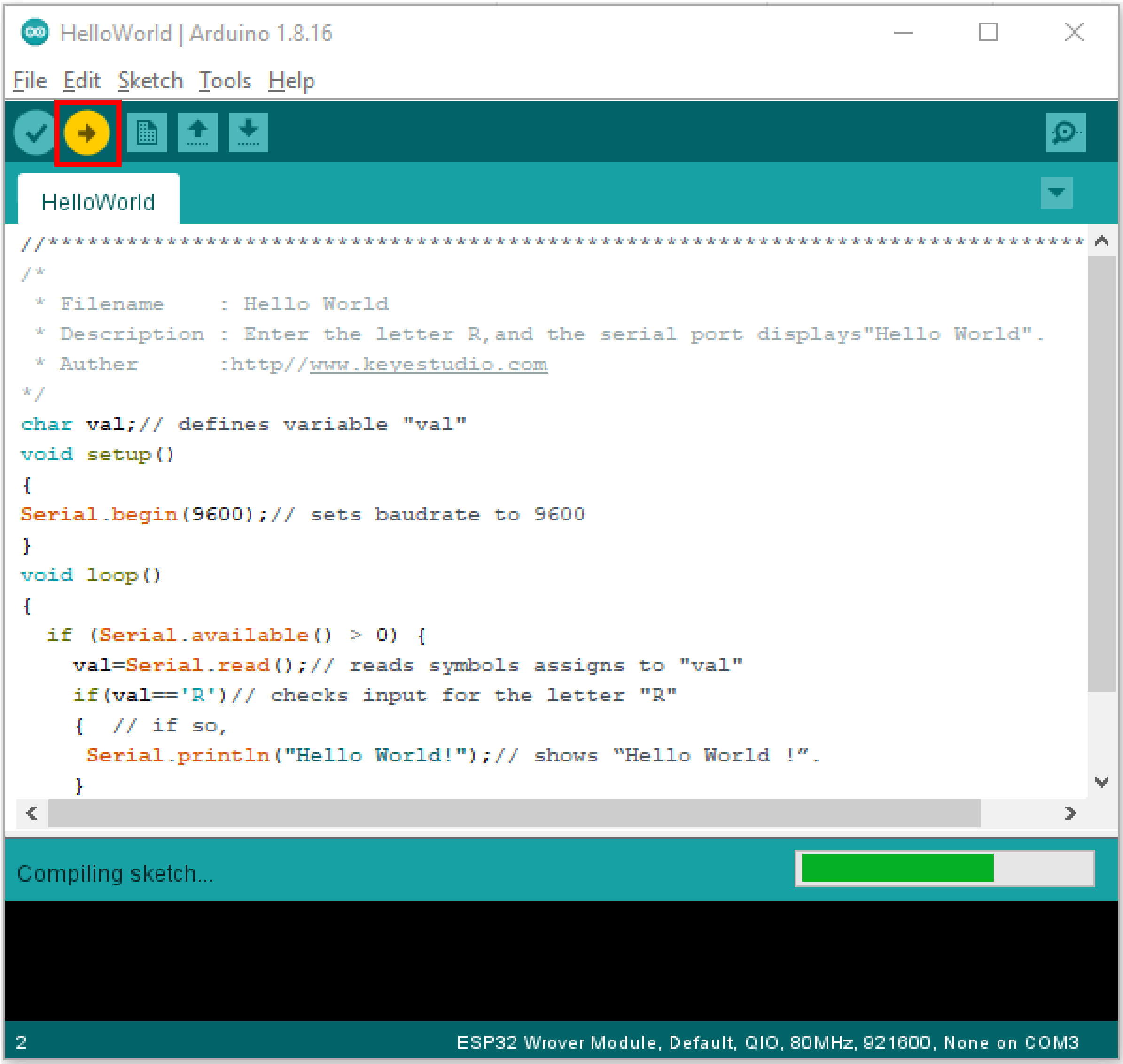
Select the correct serial port.



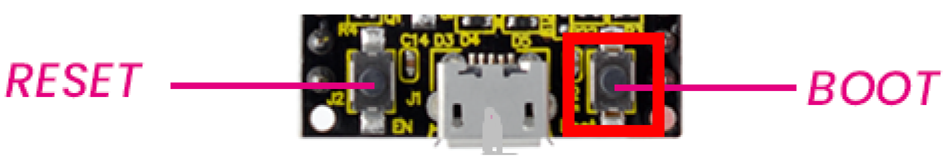
Note: For macOS users, if the upload fails, set the baud rate to 115200 before clicking .

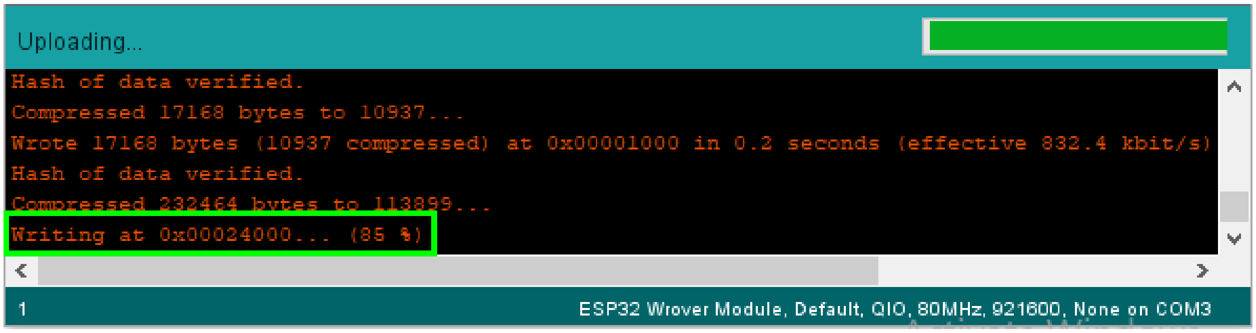


Click  to upload the test code to the ESP32.

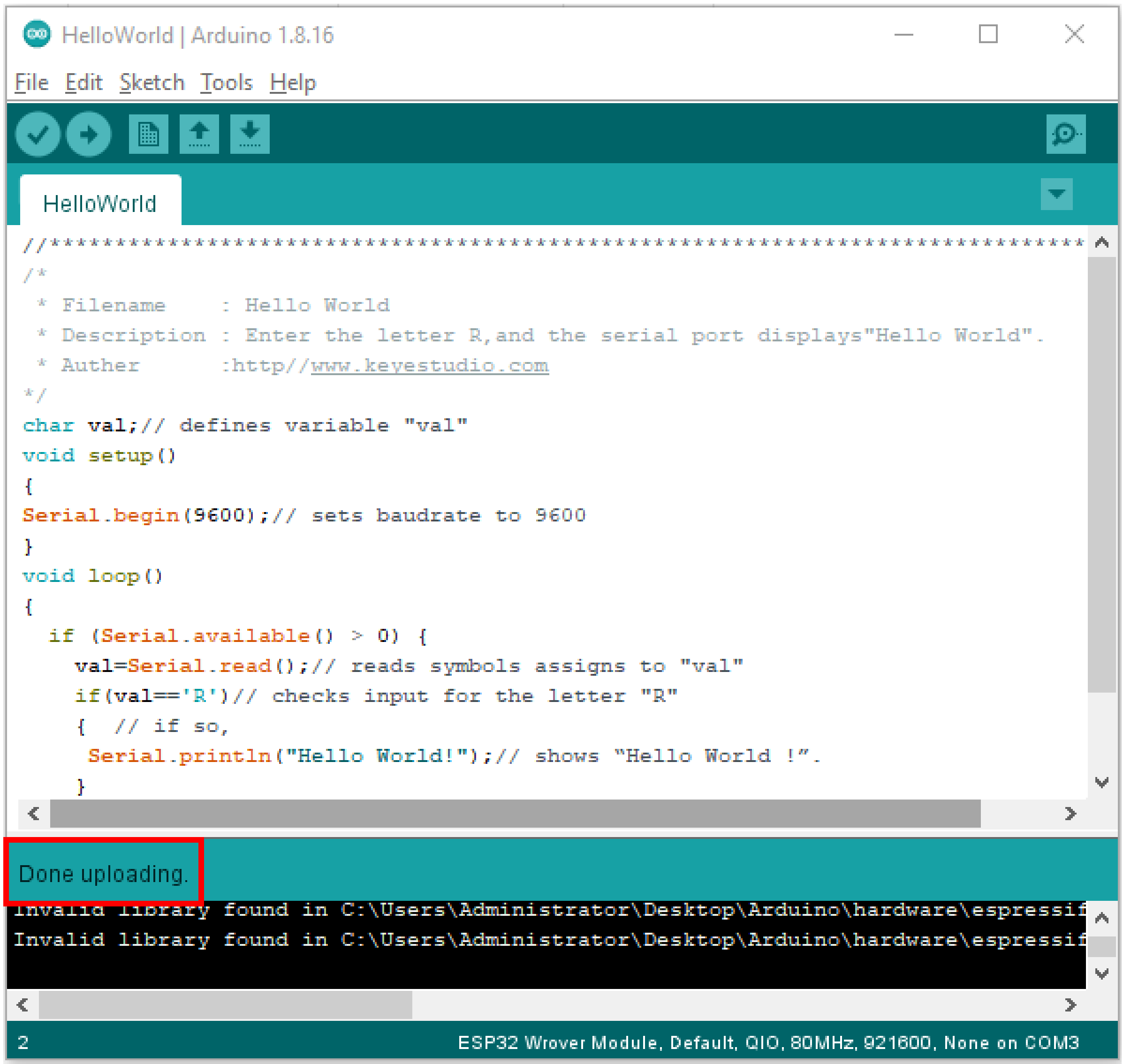


Note: If the uploading code fails, you can press and hold the Boot button on the ESP32 after clicking  and release the Boot button after the percentage of uploading progress appears., as shown below:



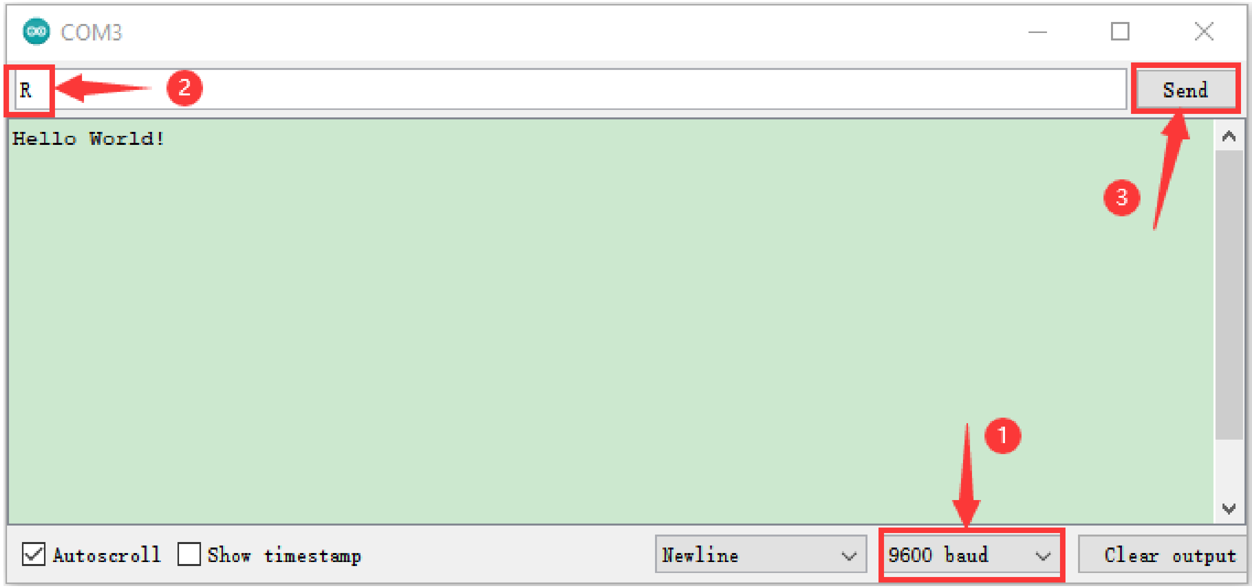


The code is uploaded successfully.

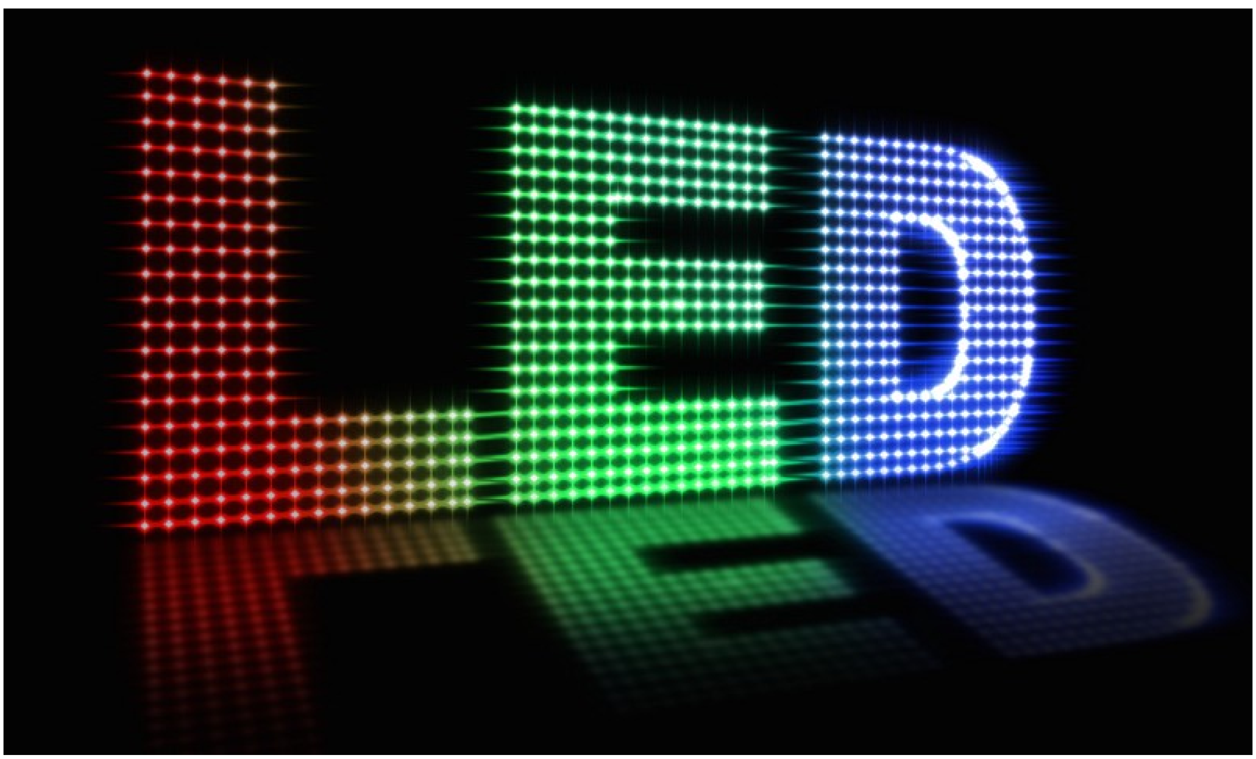


Test Result

After uploading successfullywe will use a USB cable to power on, click,set the baud rate to 9600enter the letter "**R**"click "**Send**"then the serial monitor prints "**Hello World!**".



6.2.2 Project 2: Lighting up LED



Overview

In this kit, we have a Keyestudio Purple Module, which is very simple to control. If you want to light up the LED, you just need to make a certain voltage across it.

In the project, we will control the high and low level of the signal end S through programming, so as to control the LED on and off.

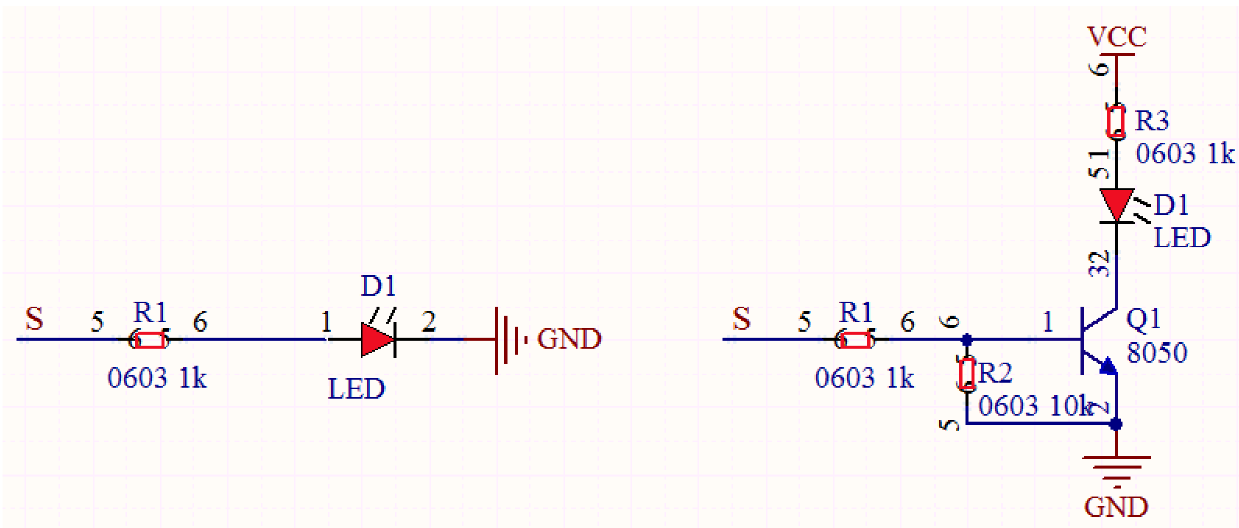
Working Principle

The two circuit diagrams are given.

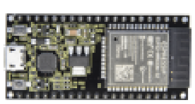
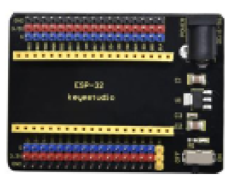
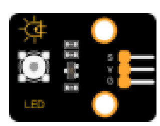
The left one is wrong wiring-up diagram. Why? Theoretically, when the S terminal outputs high levels, the LED will receive the voltage and light up.

Due to limitation of IO ports of ESP32 board, weak current can't make LED brighten.

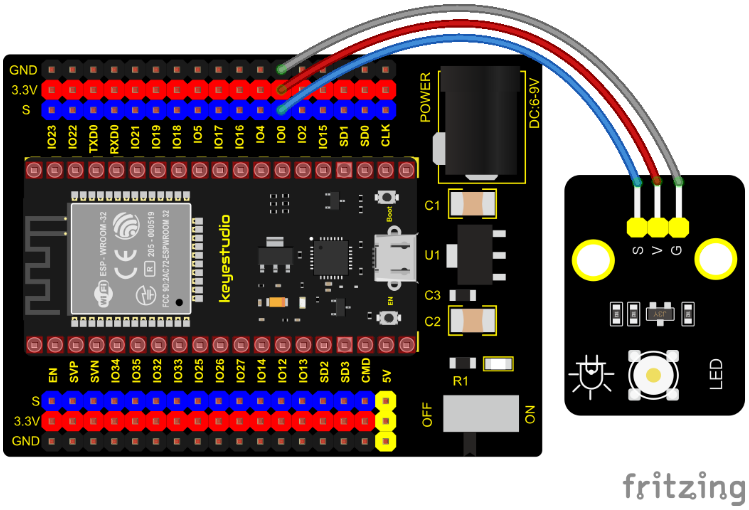
The right one is correct wiring-up diagram. GND and VCC are powered up. When the S terminal is a high level, the triode Q1 will be connected and LED will light up(note: current passes through LED and R3 to reach GND by VCC not IO ports). Conversely, when the S terminal is a low level, the triode Q1 will be disconnected and LED will go off.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1	3P Dupont Wire*1	Micro USB Cable*1

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Blink
 * Description : led Flashing 1 s
 * Auther      : http://www.keyestudio.com
*/
int ledPin = 0; //Define LED pin connection to GPIO0
void setup() {
  pinMode(ledPin, OUTPUT);//Set mode to output
}

void loop() {
  digitalWrite(ledPin, HIGH); //Output high level, turn on led
  delay(1000);//Delay 1000 ms
  digitalWrite(ledPin, LOW); //Output low level,turn off led
  delay(1000);//Delay 1000 ms
}
//**********************************************************************************
```

Code Explanation

1). **PinMode(pin,mode)**: Pin is the ESP32 GPIO pin number used to set the mode, here we set pin 0 as output mode.

2). **DigitalWrite(pin, value)**: Pin is the GPIO pin, which is defined GP0 here. Valueis the digital level that we will outputHIGH/LOW. If the pin is configured to OUTPUT using pinMode(), its voltage is set to the corresponding value: 3.3V is HIGH,low level is 0V (ground). When connect the LEDs to the pins, using the digitalWriteHIGH, the LEDs will get dim.

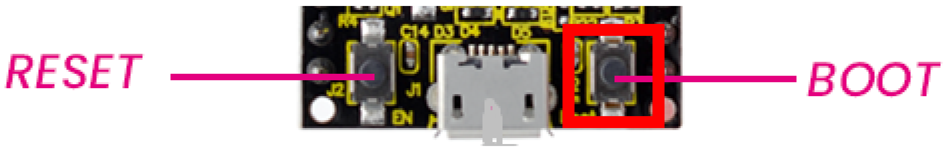
3). **Setup()** executes once, while loop() executes all the time. Delay (ms) is delay function, ms is the number of milliseconds to pause. Data type: unsigned longrange 0~ 4,294,967,295 (2^32 - 1).

4). Firstly, we connect the module signal to ledPIN, namely GP0, and set it to a high level to light the LEDs on the module. Then delay 1000 ms, controlling the LEDs on the module light up for 1s and off for 1s to achieve the flashing effect.

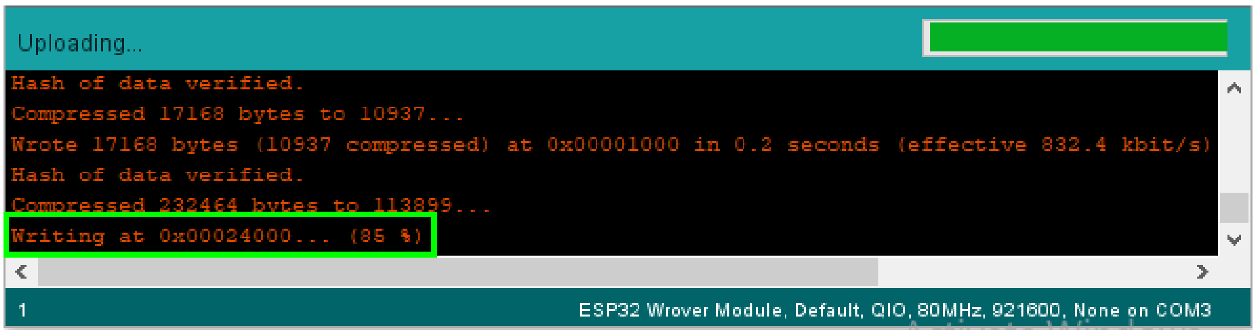
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the LED in the circuit will flash alternately.

Note: If the uploading code fails, you can press and hold the Boot button on the ESP32 after clicking  and release the Boot button after the percentage of uploading progress appears.



as shown below:



6.2.3 Project 3: Traffic Lights Module

Overview

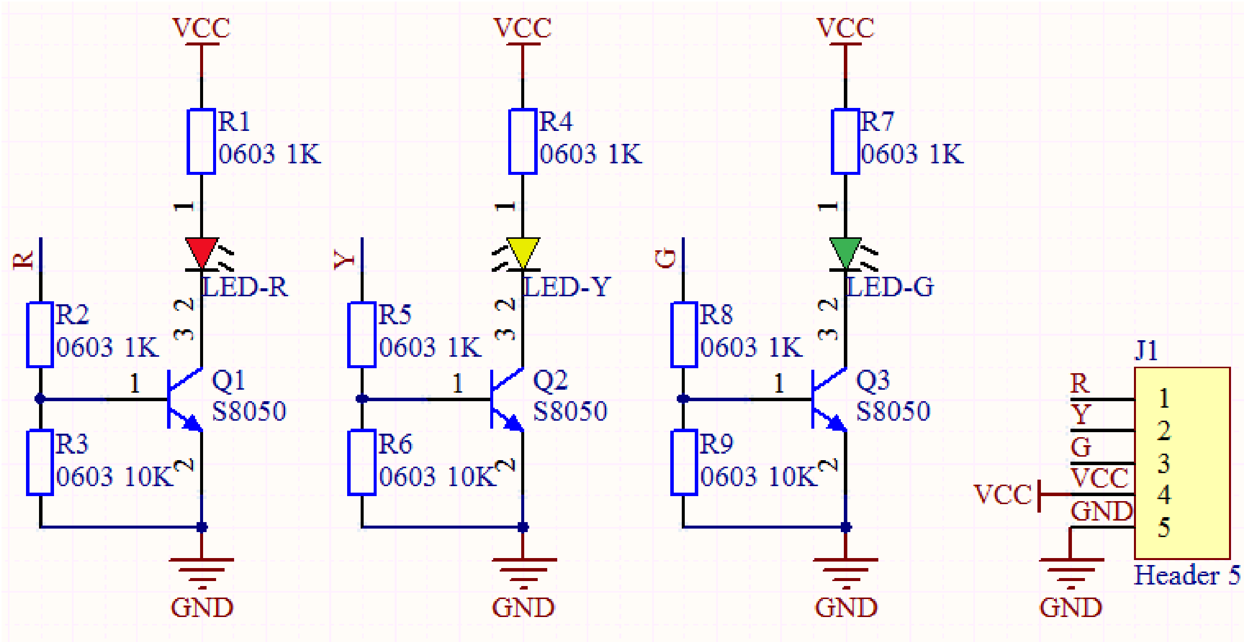
In this lesson, we will learn how to control multiple LED lights and simulate the operation of traffic lights.

Traffic lights are signal devices positioned at road intersections, pedestrian crossings, and other locations to control flows of traffic.

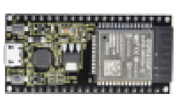
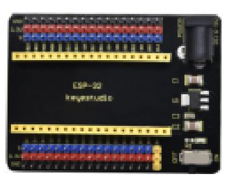
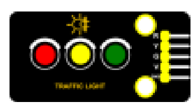
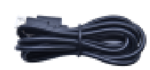
In this kit, we will use the traffic light module to simulate the traffic light.

Working Principle

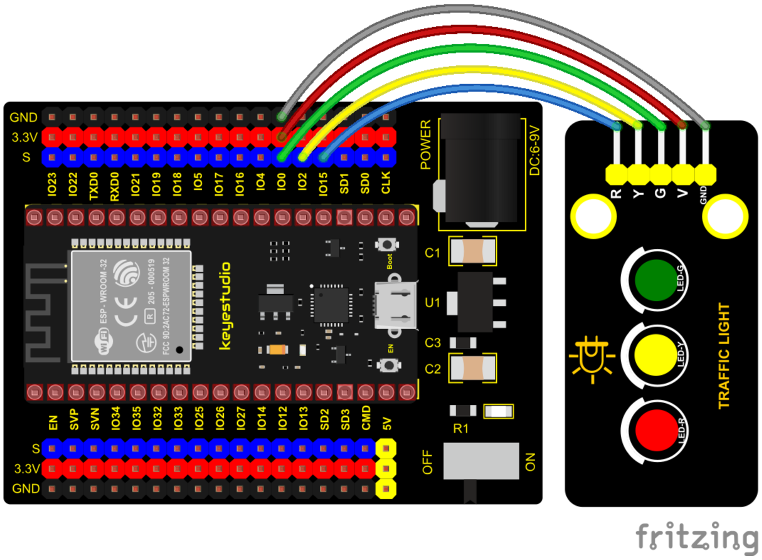
In previous lesson, we already know how to control an LED. In this part, we only need to control three separated LEDs. Input high levels to the signal R(3.3V), then the red LED will be on.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Traffic Lights Module*1	5P Dupont Wire*1	Micro USB Cable*1

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Traffic_Light
 * Description : Simulated traffic lights
 * Auther      : http://www.keyestudio.com
*/
int redPin = 15;   //Red LED connected to GPIO15
int yellowPin = 2; //Yellow LED connected to GPIO2
int greenPin = 0;   //Green LED connected to GPIO0

void setup() {
  //LED interfaces are set to output mode
  pinMode(greenPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(redPin, OUTPUT);
}

void loop() {
  digitalWrite(greenPin, HIGH); //Lighting green LED
  delay(5000);  //Delay for 5 seconds
  digitalWrite(greenPin, LOW); //Turn off green LEDS
  for (int i = 1; i <= 3; i = i + 1) {  //run three times
    digitalWrite(yellowPin, HIGH); //Lighting yellow LED
    delay(500); //Delay for 0.5 seconds
    digitalWrite(yellowPin, LOW); //Turn off yellow LED
    delay(500); //Delay for 0.5 seconds
  }
  digitalWrite(redPin, HIGH); //Lighting red LED
  delay(5000);  //Delay5s
  digitalWrite(redPin, LOW); //Turn off red LED
}
//**********************************************************************************
```

Code Explanation

Create pins, set pins mode and delayed functions.

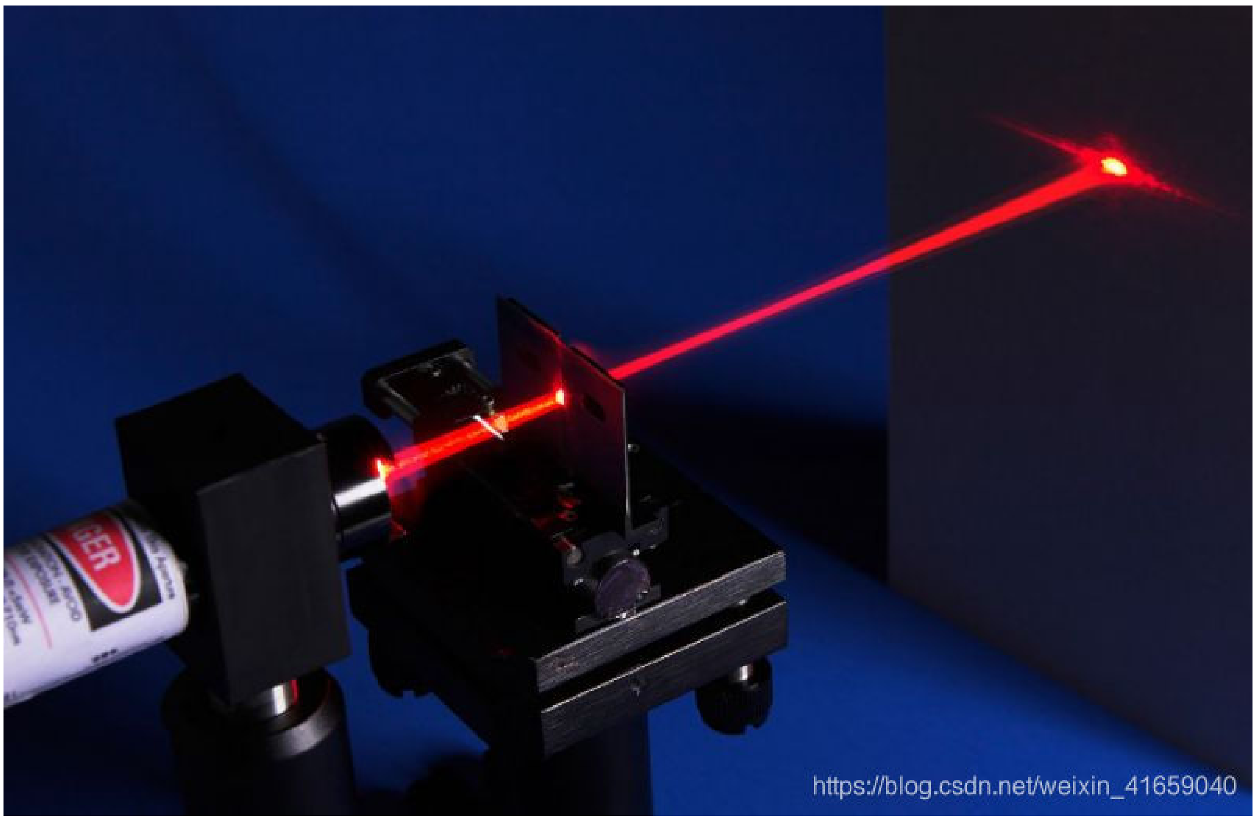
We use the function for(). for (int i = 1; i <= 3; i = i + 1) represents the variable i adds 1 fir each time from 1 to 3.

The function for (int i = 255; i >= 0; i = i - 1) indicates that i reduces by 1 each time. When i<0, exit the for() loop and execute 256 times

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the green LED will be on for 5s then off, the yellow LED will flash for 3s then go off and the red one will be on for 5s then off, the three LED modules will simulate the circulation of traffic lights automatically.

6.2.4 Project 4: Laser Sensor

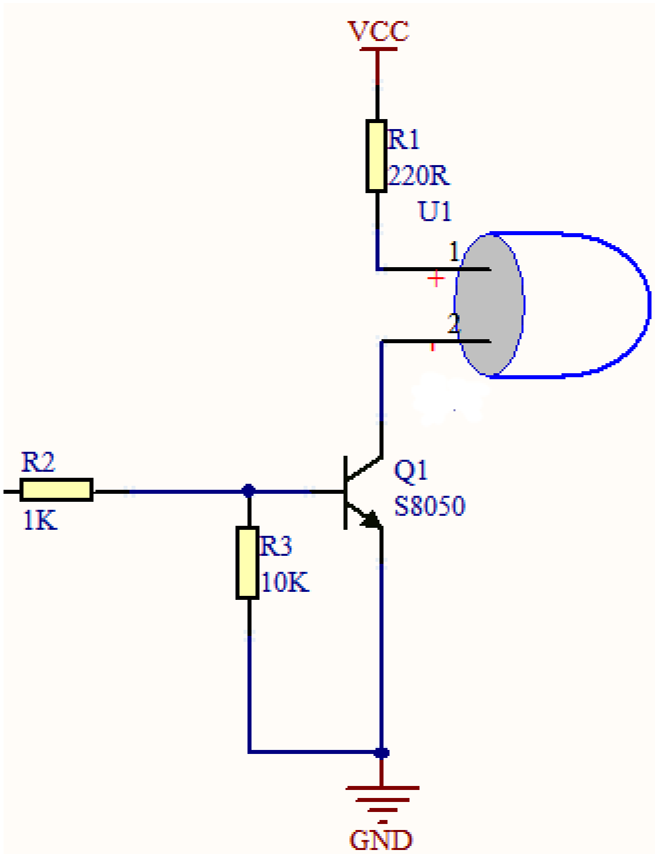


Description

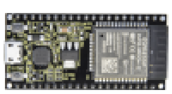
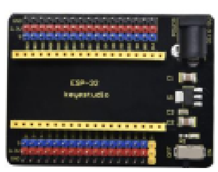
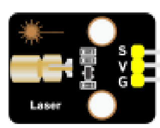
Lasers are widely used to cut, weld, surface treat, and more on specific materials. The energy of the laser is very high. The toy laser pointer may cause glare to the human eye, and it may cause retinal damage for a long time. my country also prohibits the use of laser to illuminate the aircraft.

Working Principle

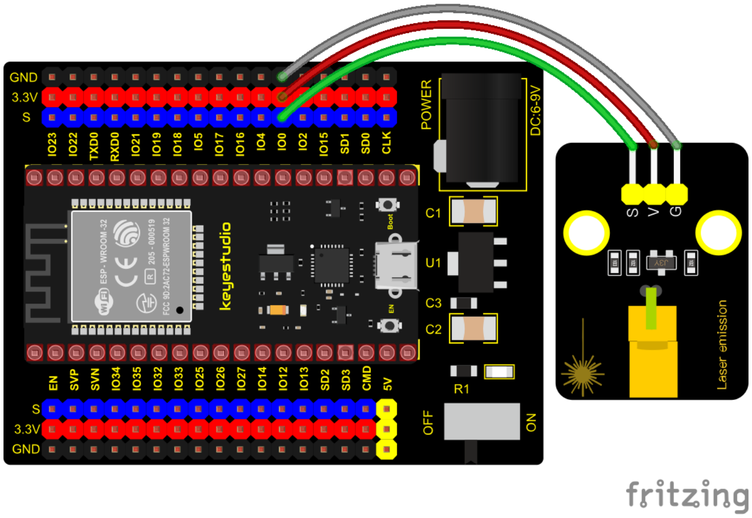
The laser head sensor module is mainly composed of a laser head with a light-emitting die, a condenser lens, and a copper adjustable sleeve. We can see the circuit schematic diagram of this module which is very similar to the LED we have learned. They are all driven by triodes. A high-level digital signal is directly input at the signal end, then the sensor will start to work; if inputting low levels, the sensor won't work.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Laser Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
/*
 * Filename    : Laser sensor
 * Description : Laser light flashing
 * Auther      : http://www.keyestudio.com
*/
int laserPin = 0; //Define the laser pin as GPIO 0
void setup() {
  pinMode(laserPin, OUTPUT);//Define laser pin as output mode
}

void loop() {
  digitalWrite(laserPin, HIGH); //Open the laser
  delay(2000);  //Delay 2 seconds
  digitalWrite(laserPin, LOW); //Shut down the laser
  delay(2000);  //Delay 2 seconds
}
```

Test Result

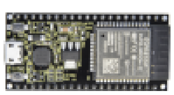
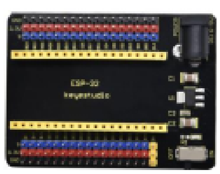
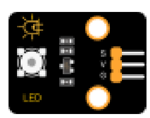
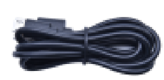
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the laser module will emit red laser signals for 2 seconds and stop emitting signals for 2 seconds on a cycle.

6.2.5 Project 5: Breathing LED

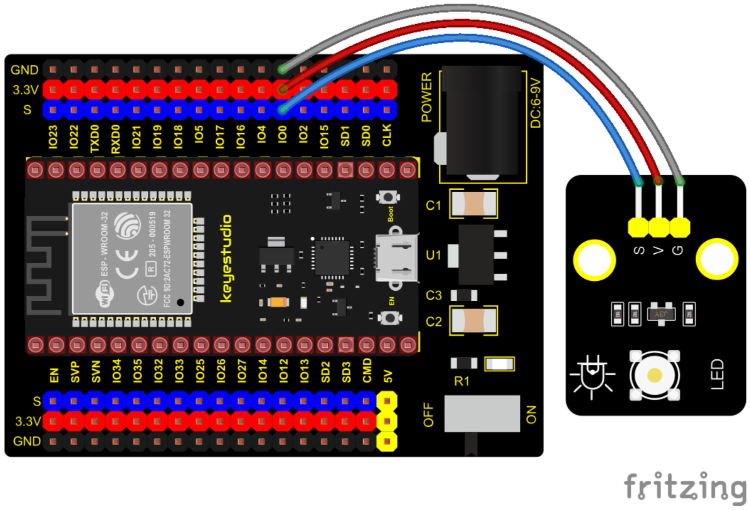
Overview

A"breathing LED"is a phenomenon where an LED's brightness smoothly changes from dark to bright and back to dark, continuing to do so and giving the illusion of an LED"breathing. This phenomenon is similar to a lung breathing in and out. So how to control LED's brightness? We need to take advantage of PWM. You may refer to Project 6.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Purple LED Module*1	3P Dupont Wire*1	MicroUSB Cable*1

Connection Diagram



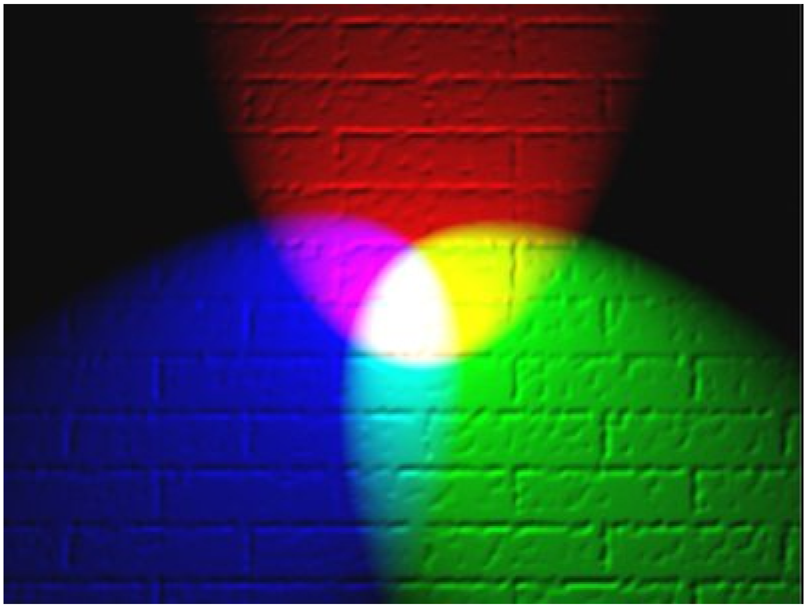
Test Code

```
//**********************************************************************
/*
 * Filename    : Breathing Led
 * Description : Make led light fade in and out, just like breathing.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_LED   0   //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() {
  ledcSetup(CHN, FRQ, PWM_BIT); //setup pwm channel
  ledcAttachPin(PIN_LED, CHN);  //attach the led pin to pwm channel
}

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

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the LED on the module gradually gets dimmer then brighter, cyclically, like human breathe.

6.2.6 Project 6: RGB Module



Overview

Among these modules is a RGB module. It adopts a F10-full color RGB foggy common cathode LED. We connect the RGB module to the PWM port of MCU and the other pin to GND(for common anode RGB, the rest pin will be connected to VCC). So what is PWM?

PWM is a means of controlling the analog output via digital means. Digital control is used to generate square waves with different duty cycles (a signal that constantly switches between high and low levels) to control the analog output.

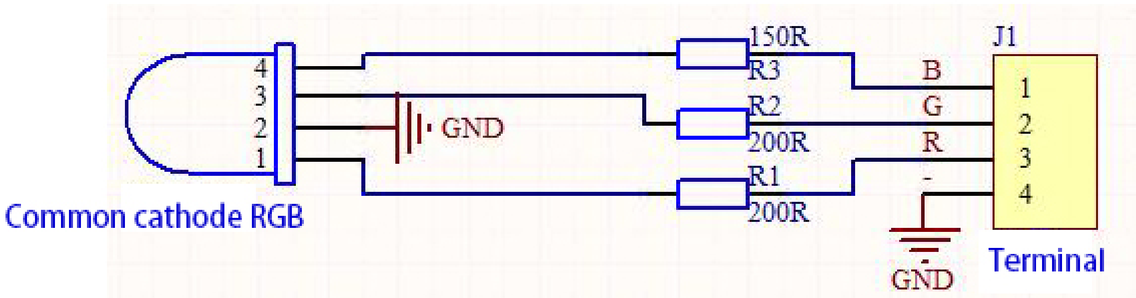
In general, the input voltages of ports are 0V and 5V. What if the 3V is required? Or a switch among 1V, 3V and 3.5V? We cannot change resistors constantly. For this reason, we resort to PWM.

For Arduino digital port voltage outputs, there are only LOW and HIGH levels, which correspond to the voltage outputs of 0V and 5V respectively. You can define LOW as“0”and HIGH as“1’, and let the Arduino output five hundred‘0’or“1”within 1 second. If output five hundred‘1’, that is 5V; if all of which is‘0’,that is 0V; if output 250 01 pattern, that is 2.5V.

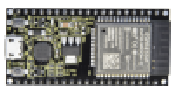
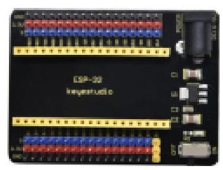
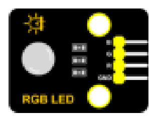
This process can be likened to showing a movie. The movie we watch are not completely continuous. Actually, it generates 25 pictures per second, which cannot be told by human eyes. Therefore, we mistake it as a continuous process. PWM works in the same way. To output different voltages, we need to control the ratio of 0 and 1. The more‘0’or‘1’ output per unit time, the more accurate the control.

Working Principle

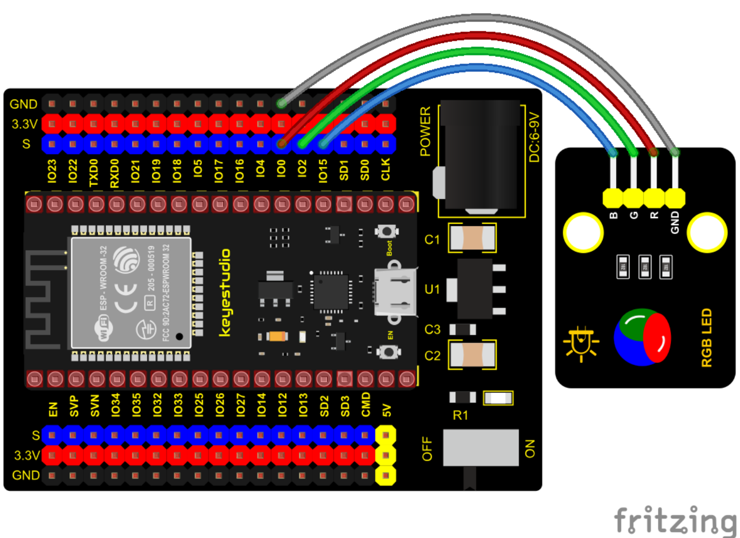
For our experiment, we will control the RGB module to display different colors through three PWM values.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Common Cathode RGB Module *1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : RGB LED
 * Description : Use RGBLED to show random color.
```

(continues on next page)

(continued from previous page)

```
 * Auther      : http//www.keyestudio.com
*/
int ledPins[] = {0, 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
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], 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);
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power onwe will see that the RGB LED on the module starts to display random colors.

6.2.7 Project 7: Button Sensor

Overview

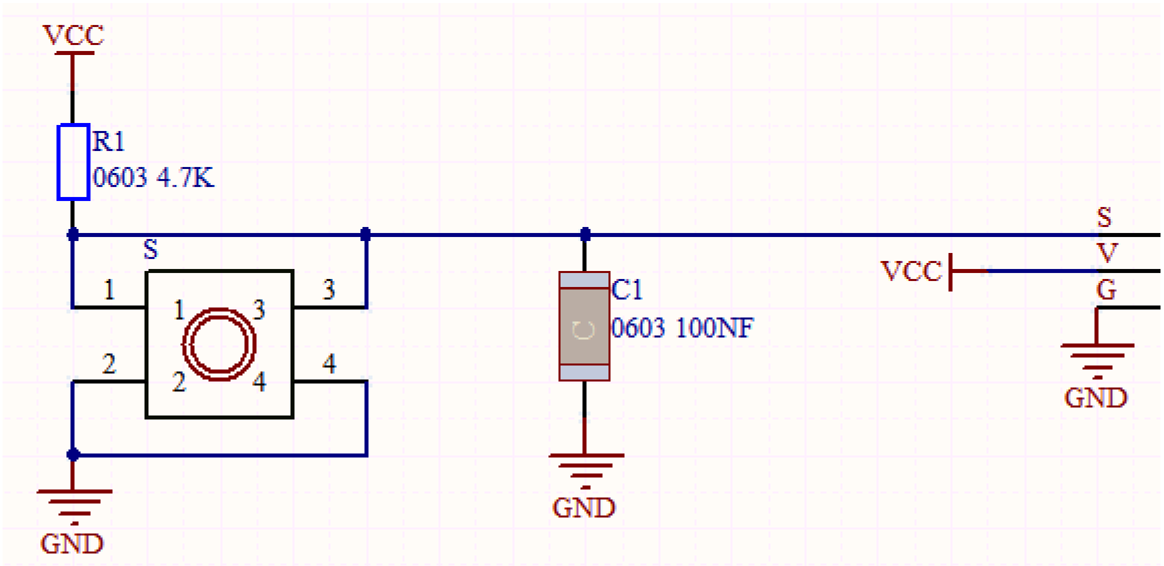
In this kit, there is a Keyestudio single-channel button module, which mainly uses a tact switch and comes with a yellow button cap.

In previous lessons, we learned how to make the pins of our single-chip microcomputer output a high level or low level. In this experiment, we will read the high level (3.3V) and low level (0V).

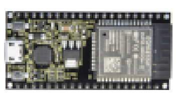
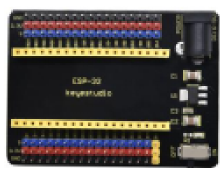
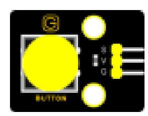
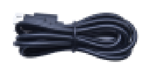
We can determine whether the button on the sensor is pressed by reading the high and low level of the S terminal on the sensor.

Working Principle

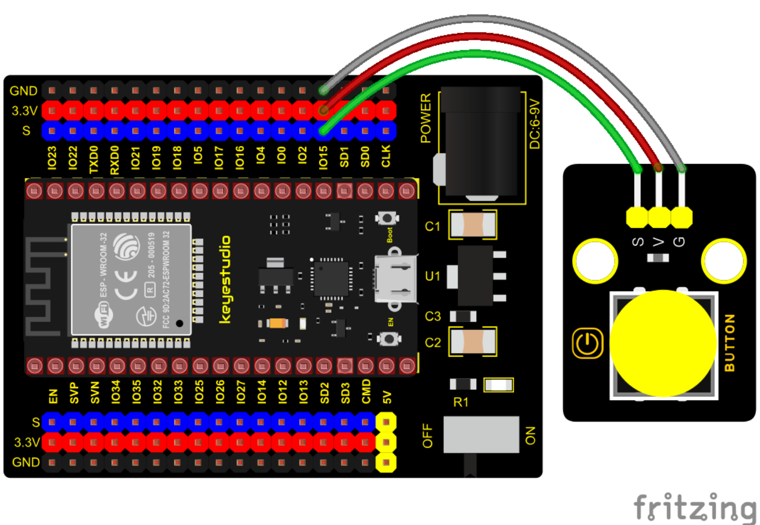
The button module has four pins. The pin 1 is connected to the pin 3 and the pin 2 is linked with the pin 4. When the button is not pressed, they are disconnected. Yet, when the button is pressed, they are connected. If the button is released, the signal end is high level.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Button Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : button
 * Description : Read key value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;  //Useto store key values
int button = 15; //The pin of the button is connected to GP15
void setup() {
  Serial.begin(9600); //Start the serial port monitor and set baud rate to 9600
  pinMode(button, INPUT); //Set key pin to input mode
}

void loop() {
  val = digitalRead(button);  //Read the value of the key and assign it to the variable␣
↪val
  Serial.print(val);  //Print it on the serial port
  if (val == 0) { //Press the key to read the low level and print the press related␣
↪information
    Serial.print("        ");
    Serial.println("Press the botton");
    delay(100);
  }

  else {  //Print information about key release
    Serial.print("        ");
    Serial.println("Loosen the botton");
    delay(100);
  }
}
//**********************************************************************************
```

Code Explanation

1). **pinMode(button, INPUT)**; set the pin of the button module to GP15 and INPUT.

Configure INPUT through pinMode(). INPUT must use the pull-up or pull-down resistor(ours module has the pull-up resistor RI).

2). **Serial.begin(9600)**: Initialize serial communication and set the baud rate to 9600.

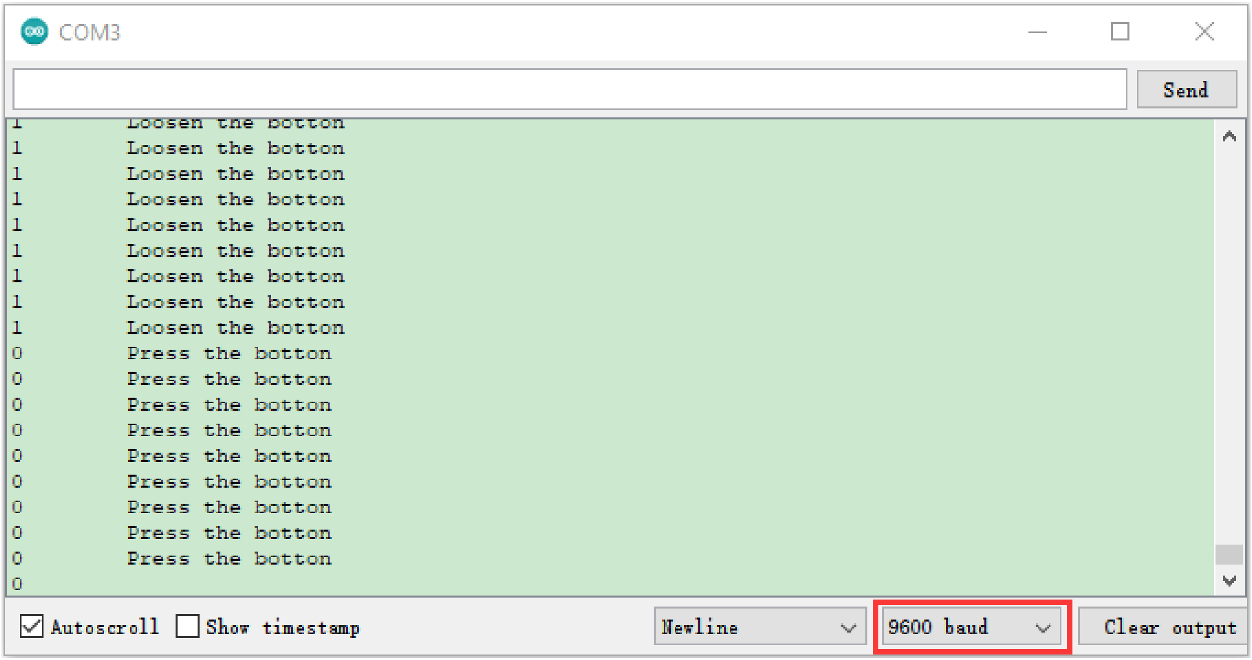
3). **digitalRead(button)**: read the digital level of the button(HIGH or LOW). If this pin is not connected to pins, the digitalRead() will return HIGH or LOW.

4). **if…else…**if the logic behind () is true, execute the code of (); otherwise execute the code of **else**.

5). If the button is pressed, the signal end is low level, GP15 is low level and Val is 0. Then the monitor will show the corresponding value and characters; otherwise, the sensor is released, val is 1 and monitor will show 1 and other characters

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. The serial monitor will display the corresponding data and characters. When the button is pressed, val is 0, the monitor will show"Press the button"when the button is released, val is 1the monitor will show"Loosen the button"; as shown below:



6.2.8 Project 8: Capacitive Sensor

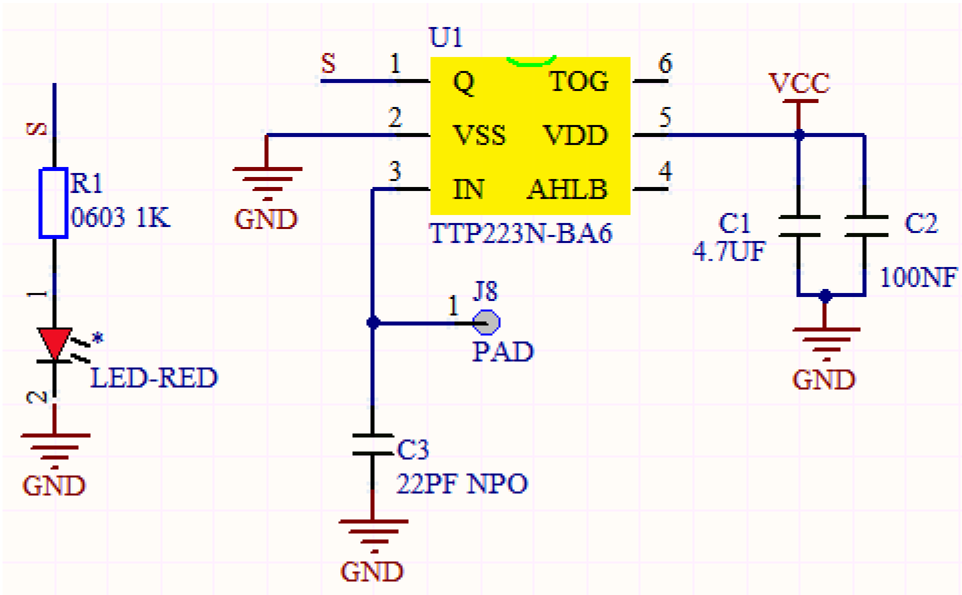
Description

In this kit, there is a capacitive touch module which mainly uses a TTP223-BA6 chip. It is a touch detection chip, which provides a touch button, and its function is to replace the traditional button with a variable area button. When we power on, the sensor needs about 0.5 seconds to stabilize.

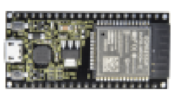
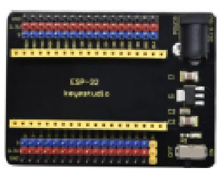
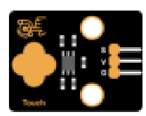
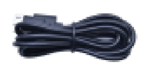
Do not touch the keys during this time period. At this time, all functions are disabled, and self-calibration is always performed. The calibration period is about 4 seconds. We display the test results in the shell.

Working Principle

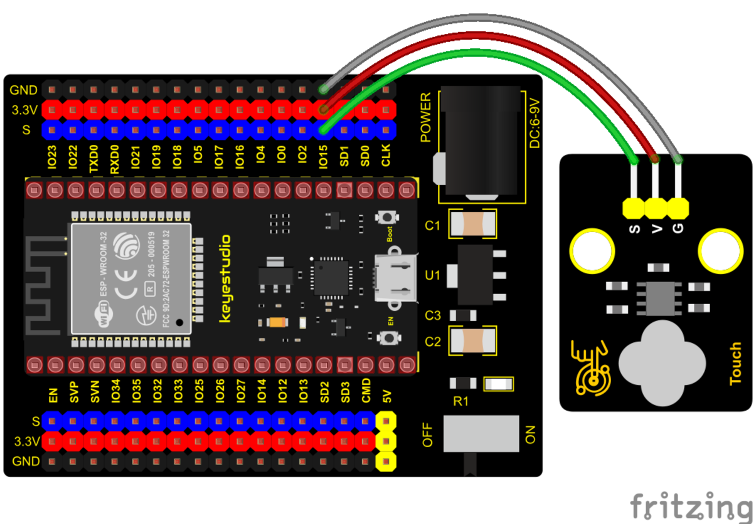
When our fingers touch the module, the signal S outputs high levels, the red LED on the module flashes. We can determine if the button is pressed or not by reading high and low levels on the sensor.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Capacitive Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Touch sensor
 * Description : Reading touch value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int touch = 15; //The key of PIN
void setup() {
  Serial.begin(9600);//Baud rate is 9600
  pinMode(touch, INPUT);//Setting input mode
}

void loop() {
  val = digitalRead(touch);//Read the value of the key
  Serial.print(val);//Print out key values
  if (val == 1) {//Press for high level
    Serial.print("        ");
    Serial.println("Press the button");
    delay(100);
  }
  else {//Release to low level
    Serial.print("        ");
    Serial.println("Loosen the button");
    delay(100);
  }
}
//**********************************************************************************
```

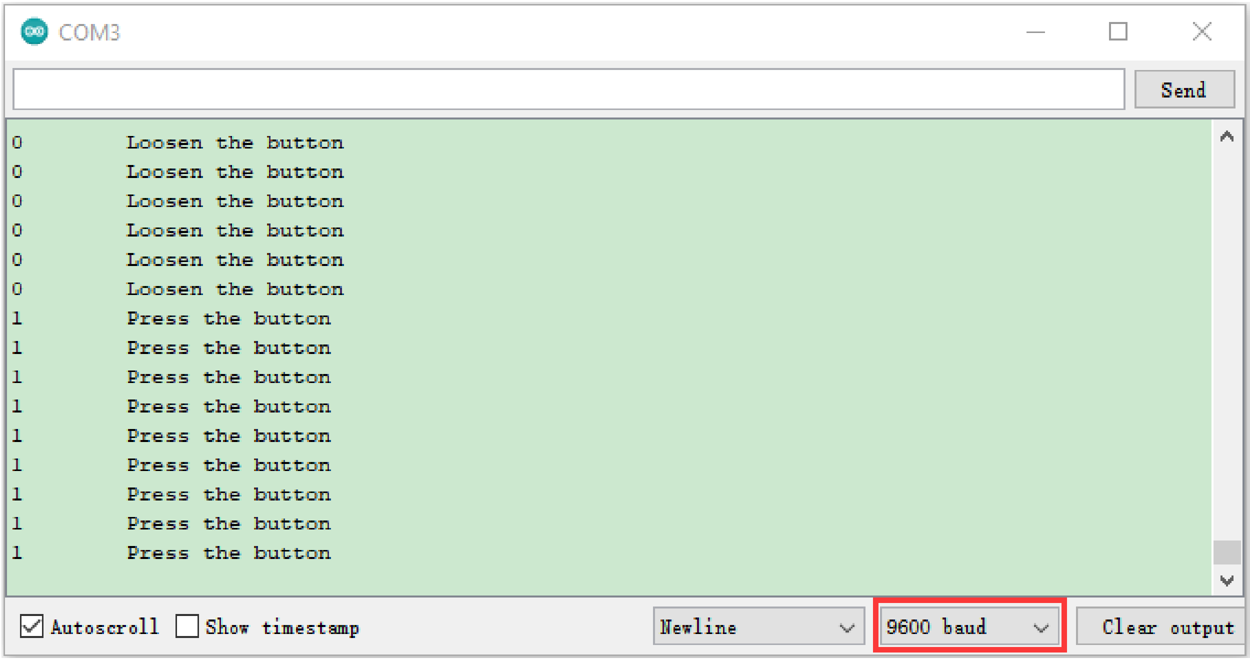
Code Explanation

When we touch the sensor, the Shell monitor will show"Pressed the button!", if not," Loosen the button!"will be shown on the monitor.

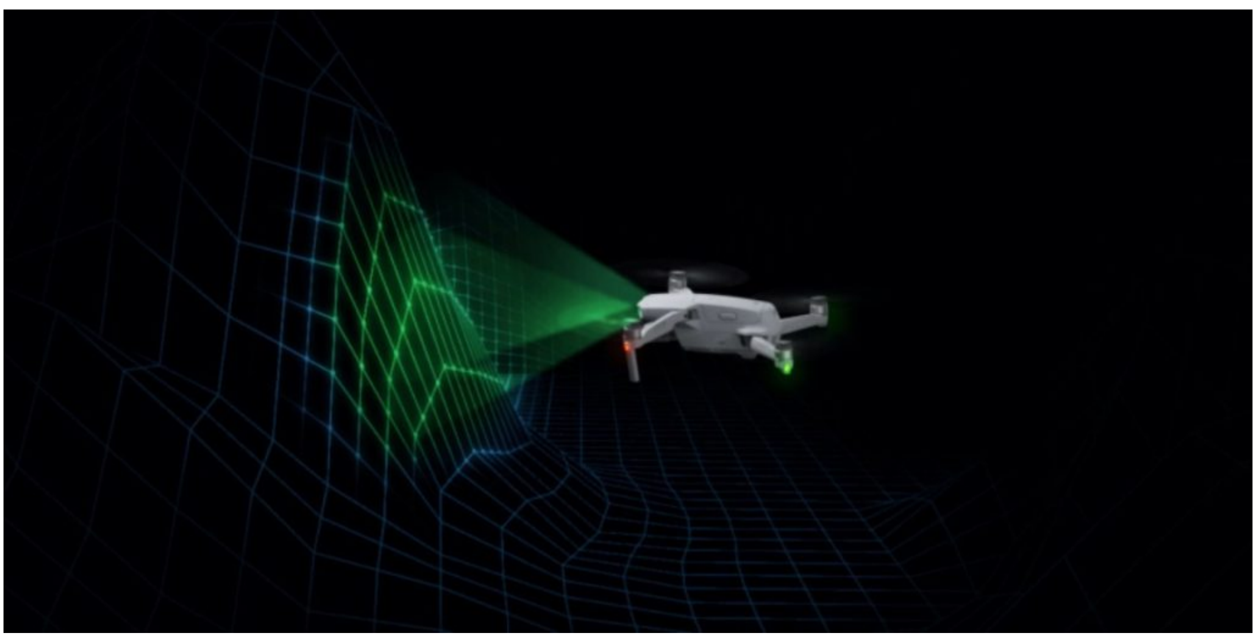
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600.

The serial monitor will display the corresponding data and characters. when the button is pressed, the red LED lights up and val is 1. Then the shell shows "Pressed the button!"; if the button is released, the red LED is off and val is 0,"Loosen the button!"will be displayed.



6.2.9 Project 9: Obstacle Avoidance Sensor



Overview

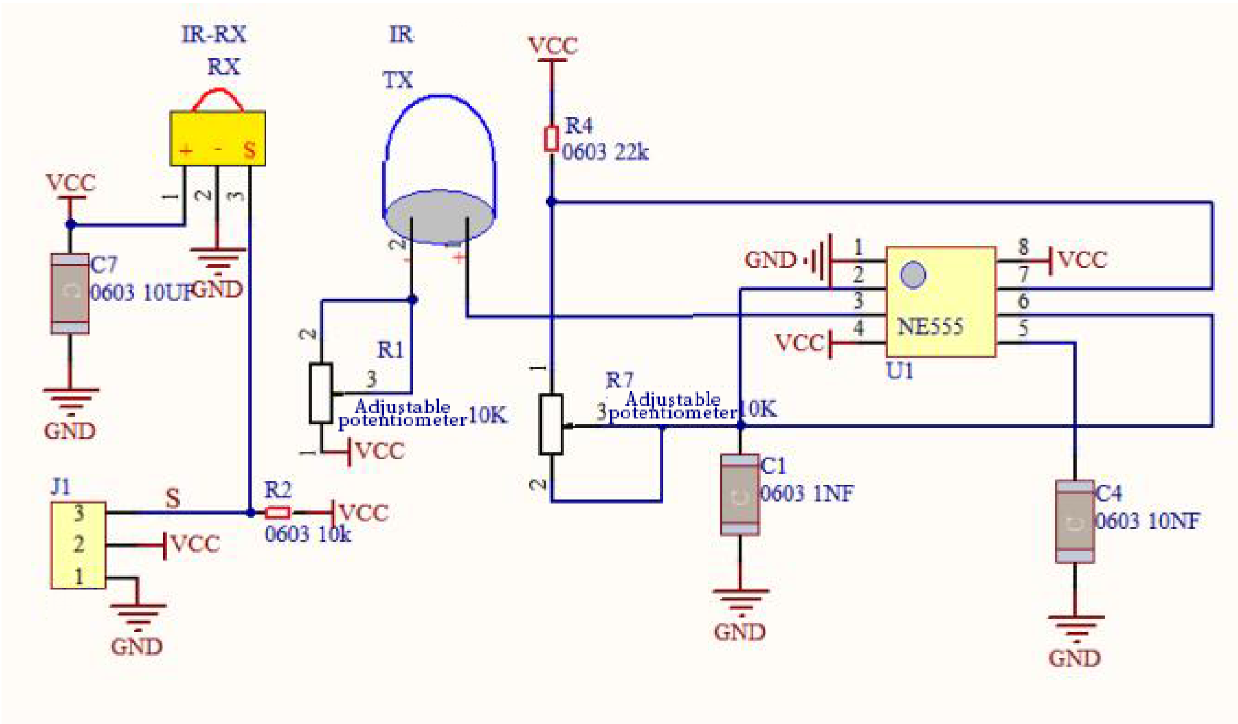
In this kit, there is a Keyestudio obstacle avoidance sensor, which mainly uses an infrared emitting and a receiving tube.

In the experiment, we will determine whether there is an obstacle by reading the high and low level of the S terminal on the sensor.

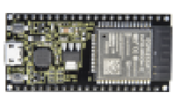
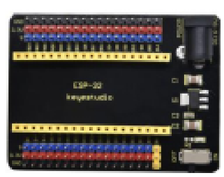
Working Principle

NE555 circuit provides IR signals with frequency to the emitter TX, then the IR signals will fade with the increase of transmission distance. If encountering the obstacle, it will be reflected back.

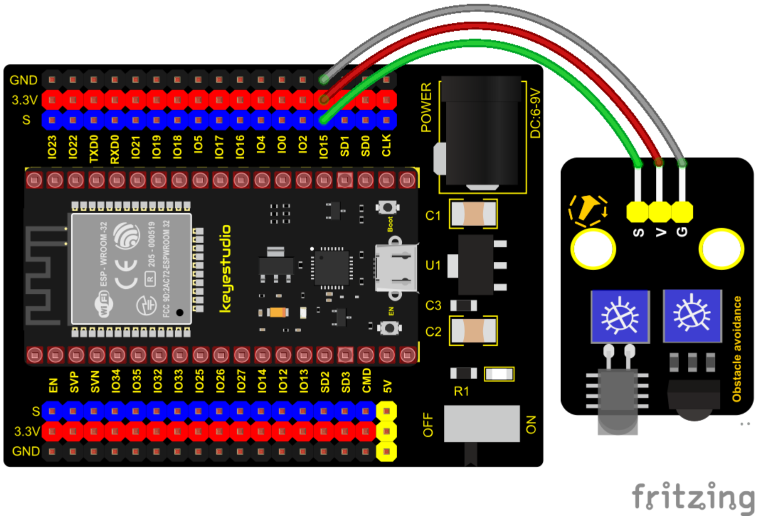
When the receiver RX meets the weak signals reflected back, the receiving pin will output high levels, which indicates the obstacle is far away. On the contrary, it the reflected signals are stronger, low levels will be output, which represents the obstacle is close. There are two potentiometers on the module, and by adjusting the two potentiometers, we can adjust its effective distance.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Obstacle Avoidance Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : obstacle avoidance sensor
 * Description : Reading the obstacle avoidance value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(15, INPUT);//Set pin GP15 to input mode
}

void loop() {
  val = digitalRead(15);//Read digital level
  Serial.print(val);//Print the level signal read
  if (val == 0) {//Obstruction detected
    Serial.print("        ");
    Serial.println("There are obstacles");
    delay(100);
  }
  else {//No obstructions detected
    Serial.print("        ");
    Serial.println("All going well");
    delay(100);
  }
}
//**********************************************************************************
```

Code Explanation

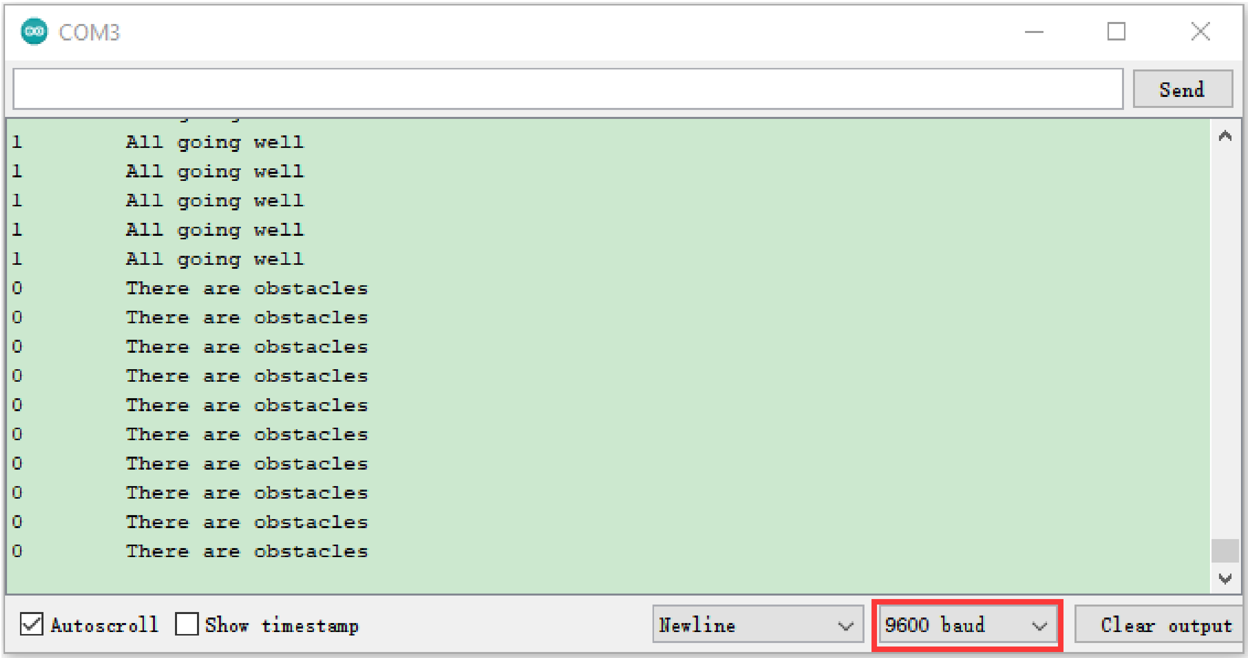
Note:

Upload the test code and wire up according to the connection diagram. After powering on, we start to adjust the two potentiometers to sense distance.

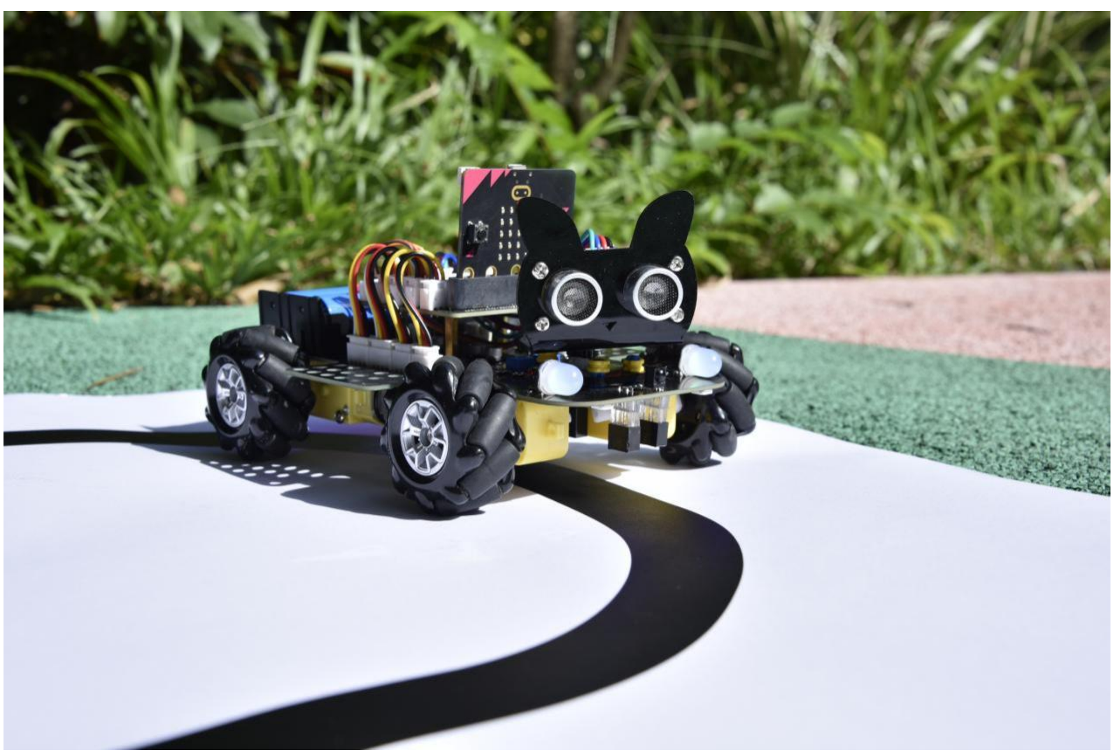
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600.

The serial monitor will display the corresponding data and characters. When the sensor detects the obstacle, the val is 0, the monitor will show“There are obstacles”; if the obstacle is not detected, the val is 1,“All going well” will be shown.



6.2.10 Project 10: Line Tracking Sensor



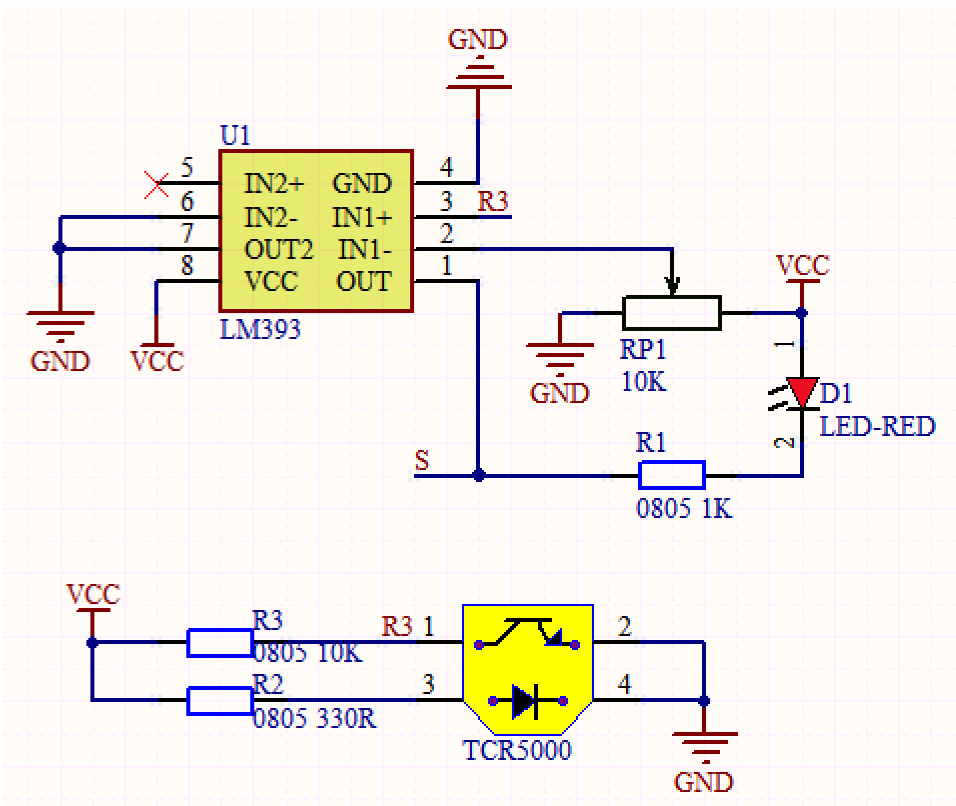
Description

In this kit, there is a DIY electronic building block single-channel line tracking sensor which mainly uses a TCRT5000 reflective black and white line recognition sensor element.

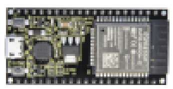
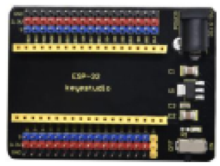
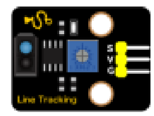
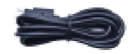
In the experiment, we judge the color (black and white) of the object detected by the sensor by reading the high and low levels of the S terminal on the module; and display the test results on the shell.

Working Principle

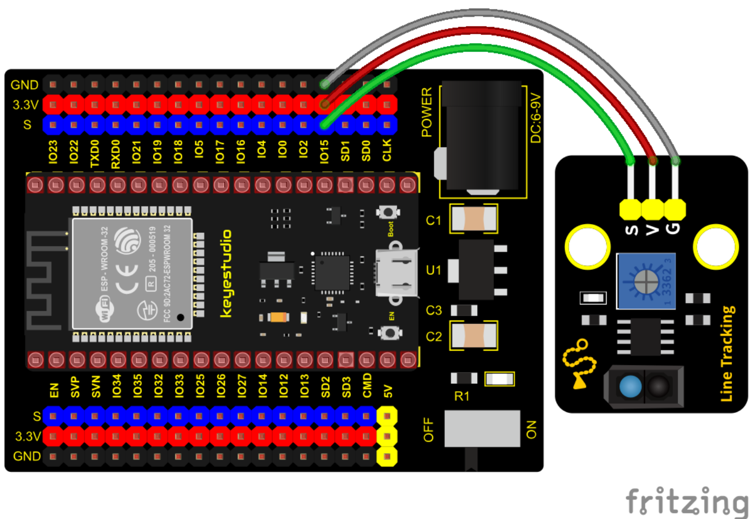
When a black or no object is detected, the signal terminal will output high levels; when white object is detected, the signal terminal is low level; its detection height is 0-3cm. We can adjust the sensitivity by rotating the potentiometer on the sensor. When the potentiometer is rotated, the sensitivity is best when the red LED on the sensor is at the critical point between off and on.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	DIY Line Tracking Sensor*1	3P Dupont Wire*1	MicroUSB Cable*1

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : line tracking
 * Description : Reading the tracking sensor value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(15, INPUT);//Sets sensor pin to input mode
}

void loop() {
  val = digitalRead(15);//Read the digital level output by the patrol sensor
  Serial.print(val);//Serial port print value
  if (val == 0) {//White val is 0 detected
    Serial.print("        ");
```

(continues on next page)

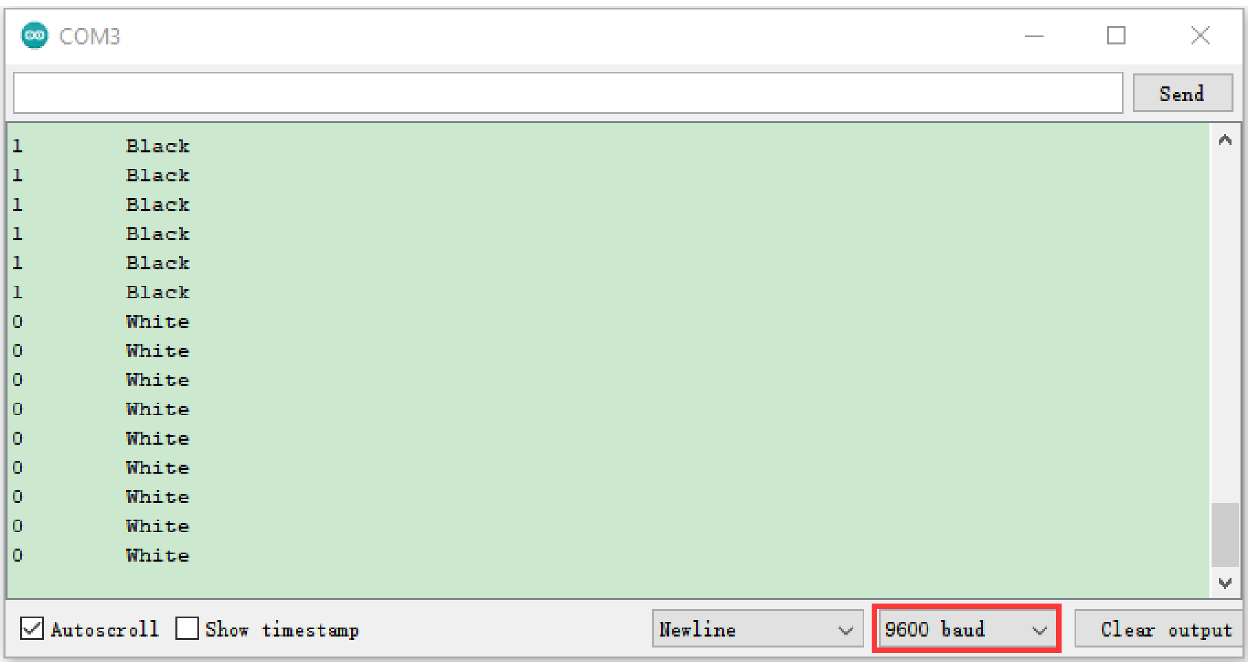
(continued from previous page)

```
    Serial.println("White");
    delay(100);
  }
  else {//Black val is 1 detected
    Serial.print("        ");
    Serial.println("Black");
    delay(100);
  }
}
//*************************************************************************************
```

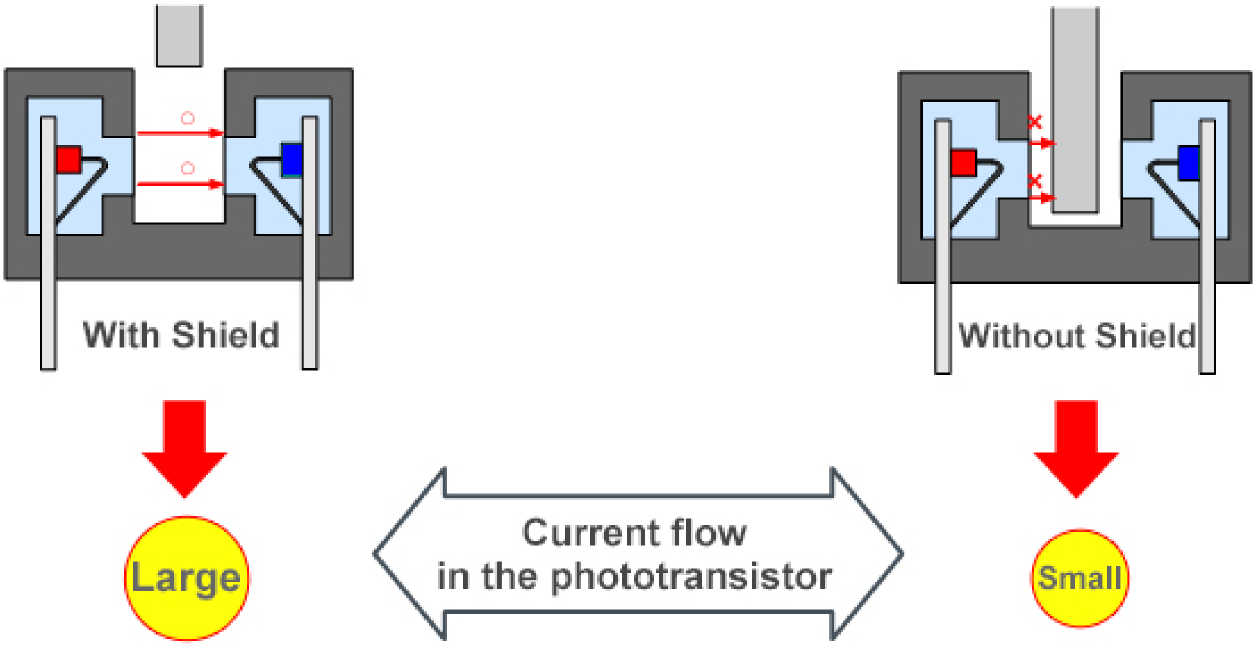
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600.

The serial monitor will display the corresponding data and characters. when the sensor doesn't detect an object or detects a black object, the val is 1, and the monitor will display "1 Black" ; when a white object (can reflect light) is detected, the val is 0, and the monitor will display "0 White".



6.2.11 Project 11: Photo Interrupter

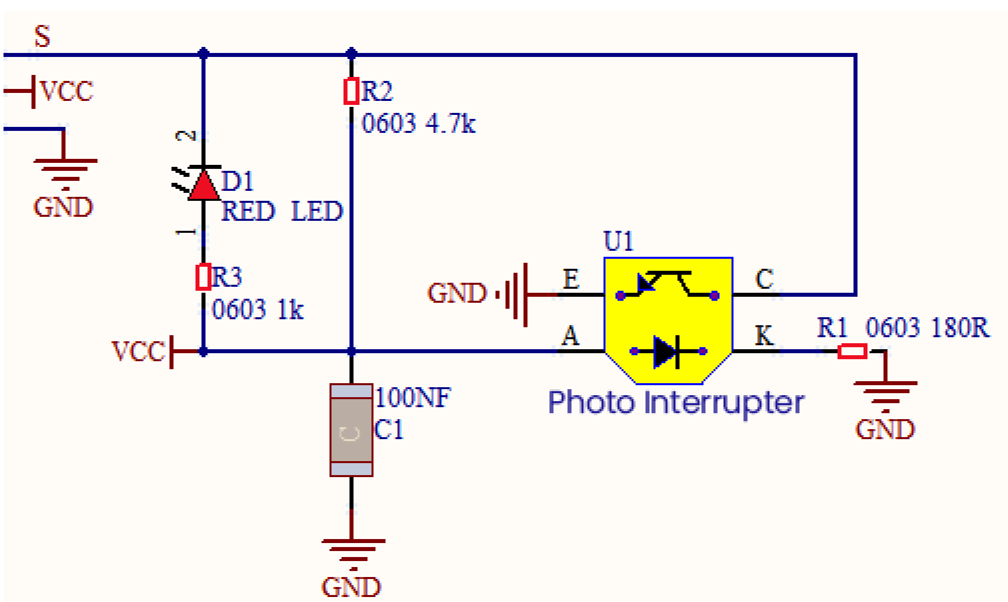


Description

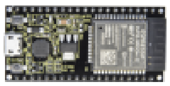
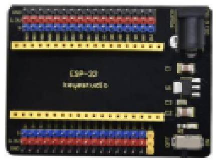
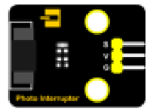
This kit contains a photo interrupter which mainly uses 1 ITR-9608 photoelectric switch. It is a photoelectric switch optical switch sensor.

Working Principle

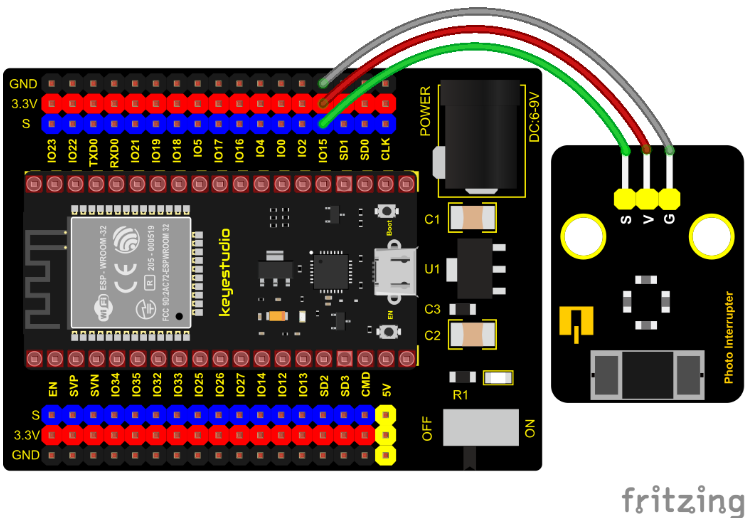
When the paper is put in the slot, C is connected with VCC and the signal end S of the sensor are high levels; then the red LED will be off. Otherwise, the red LED will be on.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Photo Interrupter*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Photo_Interrupt
 * Description : Light snap sensor counting
 * Auther      : http://www.keyestudio.com
*/
int PushCounter = 0; //The count variable is assigned an initial value of 0
int State = 0; //Store the current state of the sensor output
int lastState = 0; //Stores the state of the last sensor output
void setup() {
  Serial.begin(9600);//Set the baud rate to 9600
  pinMode(15, INPUT);//Set the light snap sensor pin to input mode
}

void loop() {
  State = digitalRead(15);//Read current state
  if (State != lastState) {//If the state is different from the last read
    if (State == 1) {//block the light
      PushCounter = PushCounter + 1;//Count + 1
```

(continues on next page)

(continued from previous page)

```
      Serial.println(PushCounter);//Print count
    }
  }
  lastState = State;//Update state
}
//*************************************************************************************
```

Code Explanation

Logic setting:

<table>
<tr><td rowspan="3">Initial Setting</td><td colspan="2">Set PushCounter to 0</td></tr>
<tr><td colspan="2">Set State to 0 (value of the sensor)</td></tr>
<tr><td colspan="2">Set lastState to 0</td></tr>
<tr><td>when an object enters the slot</td><td>lastState is 0， State turns into 1; lastState turns into 1</td><td>Set PushCounter to PushCounter+1

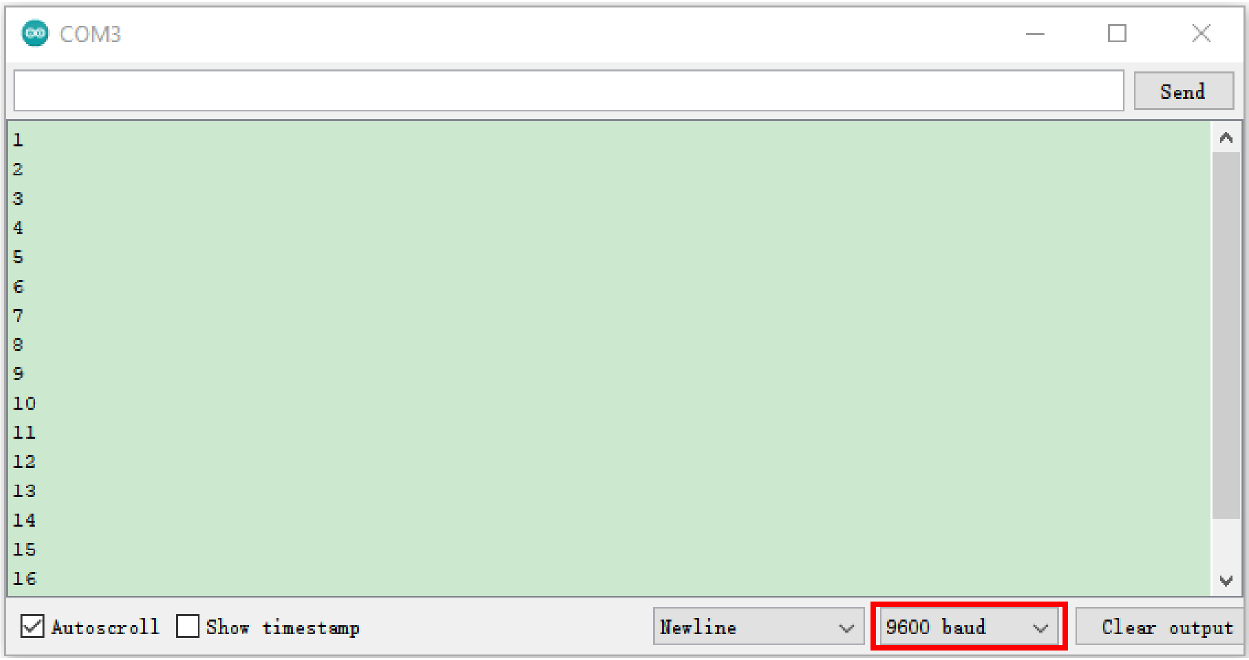
print the value of PushCounter</td></tr>
<tr><td>when the object leaves the slot</td><td>lastState is 1 ， State becomes 0， two data are not equal， lastState turns into 0.</td><td>PushCounterdoesn’t change;

Don’t print the value of PushCounter</td></tr>
<tr><td>When the object goes through this slot again</td><td>lastState is 0, State becomes 1， two data are not equal， lastState turns into 1.</td><td>SetPushCounter to PushCounter+1. And print the value of PushCounter</td></tr>
<tr><td>When the object leaves this slot again</td><td>lastState is 1， State turns into 0， two data are not equal lastState turns into 0</td><td>PushCounter doesn’t change;Don’t print the PushCounter value</td></tr>
</table>

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600.

The serial monitor will display the PushCounter data. Every time when the object passes through the slot of the sensor, the PushCounter data will increase by 1 continuously, as shown below;



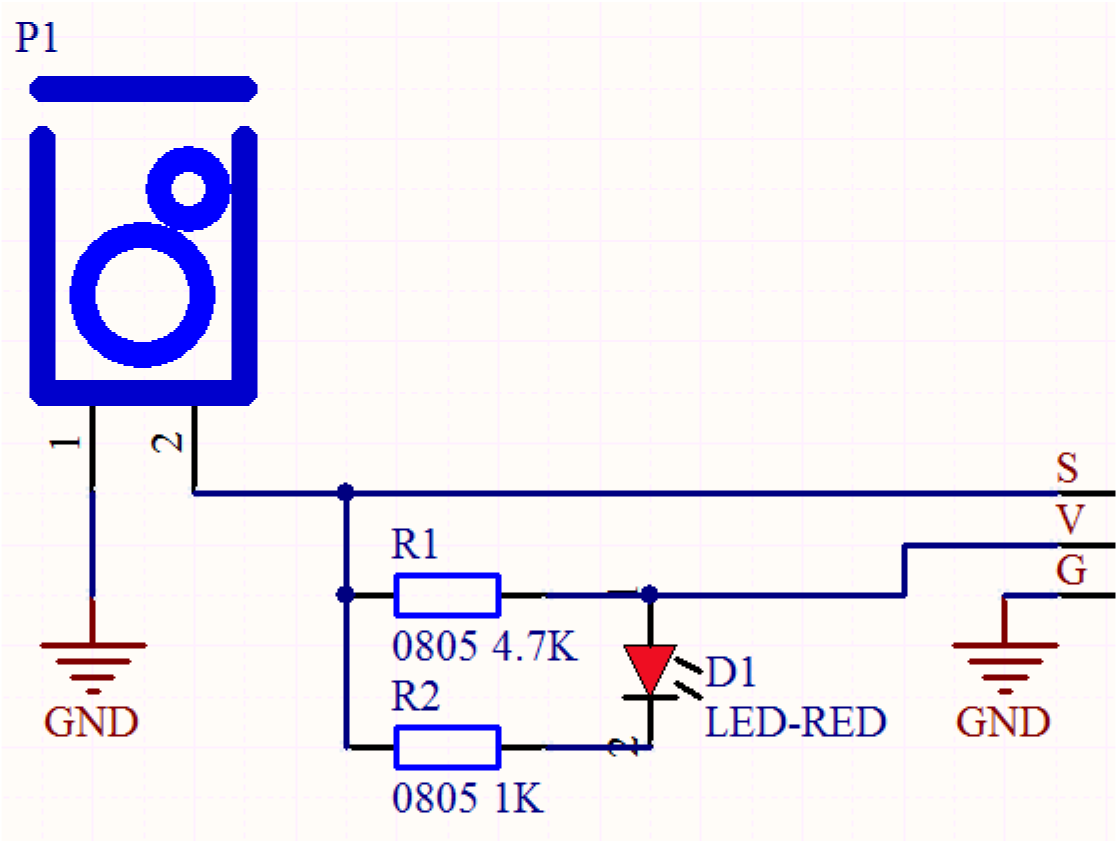
6.2.12 Project 12: Tilt Module

Overview

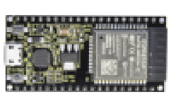
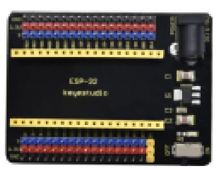
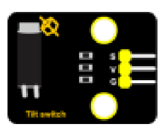
In this kit, there is a Keyestudio tilt sensor. The tilt switch can output signals of different levels according to whether the module is tilted. There is a ball inside. When the switch is higher than the horizontal level, the switch is turned on, and when it is lower than the horizontal level, the switch is turned off. This tilt module can be used for tilt detection, alarm or other detection.

Working Principle

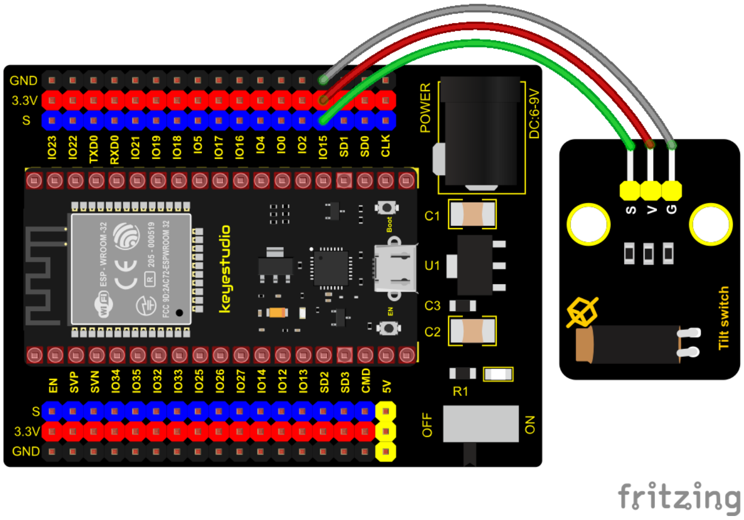
The working principle is pretty simple. When pin 1 and 2 of the ball switch P1 are connected, the signal S is low level and the red LED will light up; when they are disconnected, the pin will be pulled up by the 4.7K R1 and make S a high level, then LED will be off.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Tilt Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram

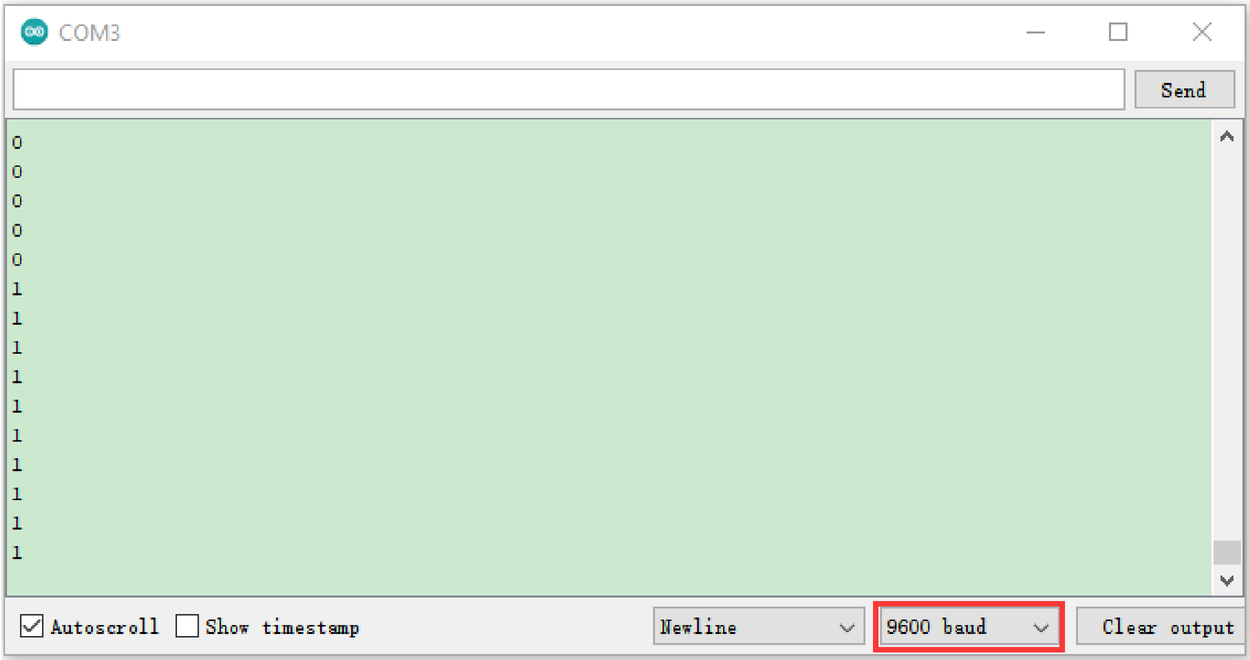


Test Code

```
//**********************************************************************************
/*
 * Filename    : Tilt switch
 * Description : Reading the tilt sensor value
 * Auther      :http://www.keyestudio.com
*/
int val; //Store the level value output by the tilt sensor

void setup() {
  Serial.begin(9600);
  pinMode(15, INPUT);  //Connect the pin of the tilt sensor to GP15 and set GP15 to the input mode
}

void loop() {
  val = digitalRead(15); //Read module level signal
  Serial.println(val);  //Newline print
  delay(100); //Delay for 100 ms
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. Make the tilt module incline to one side, the red LED on the module will be off and the monitor will display“1”. In contrast, if you make it incline the other side, the red LED will light up and the monitor will display “0”.



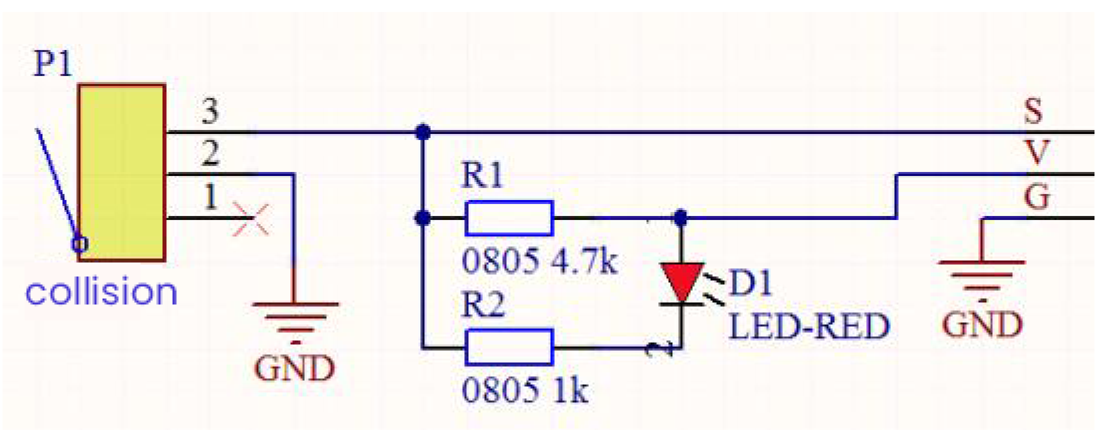
6.2.13 Project 13: Collision Sensor

Description

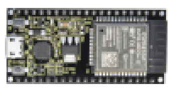
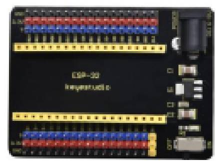
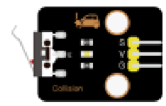
The collision sensor uses a tact switch. This sensor is often used as a limit switch in 3D printers. In the experiment, we judge whether the sensor shrapnel is pressed down by reading the high and low levels of the S terminal on the module; and, we display the test results in the shell.

Working Principle

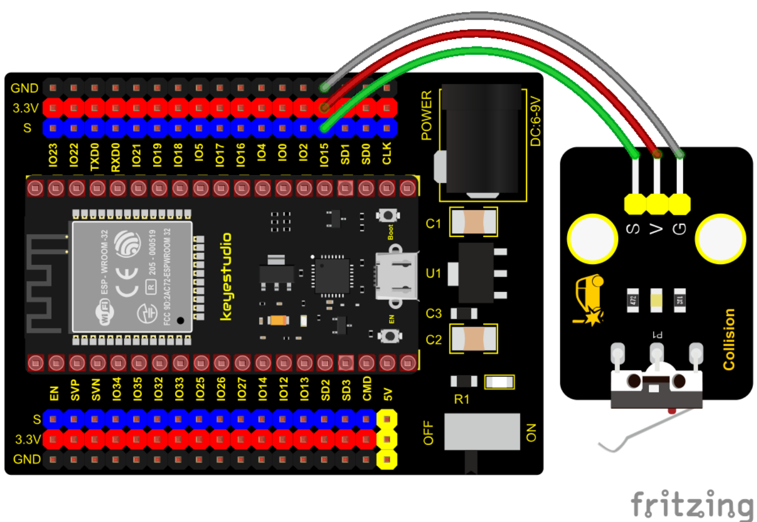
It mainly uses a tact switch. When the shrapnel of the tact switch is pressed, 2 and 3 are connected, the signal terminal S is low level, and the red LED on the module lights up; when the touch switch is not pressed, 2 and 3 are not connected, and 3 is pulled up to a high level by the 4.7K resistor R1, that is, the sensor signal terminal S is a high level, and the built-in red LED will be off at this time.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Collision Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : collision sensor
 * Description : Reading the value of the collision sensor
 * Auther      :  http://www.keyestudio.com
*/
int val = 0;
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(15, INPUT);//Set collision sensor pin 15 to input mode
}

void loop() {
  val = digitalRead(15);//Read the value of the collision sensor
  Serial.print(val);//Newline print
```

(continues on next page)

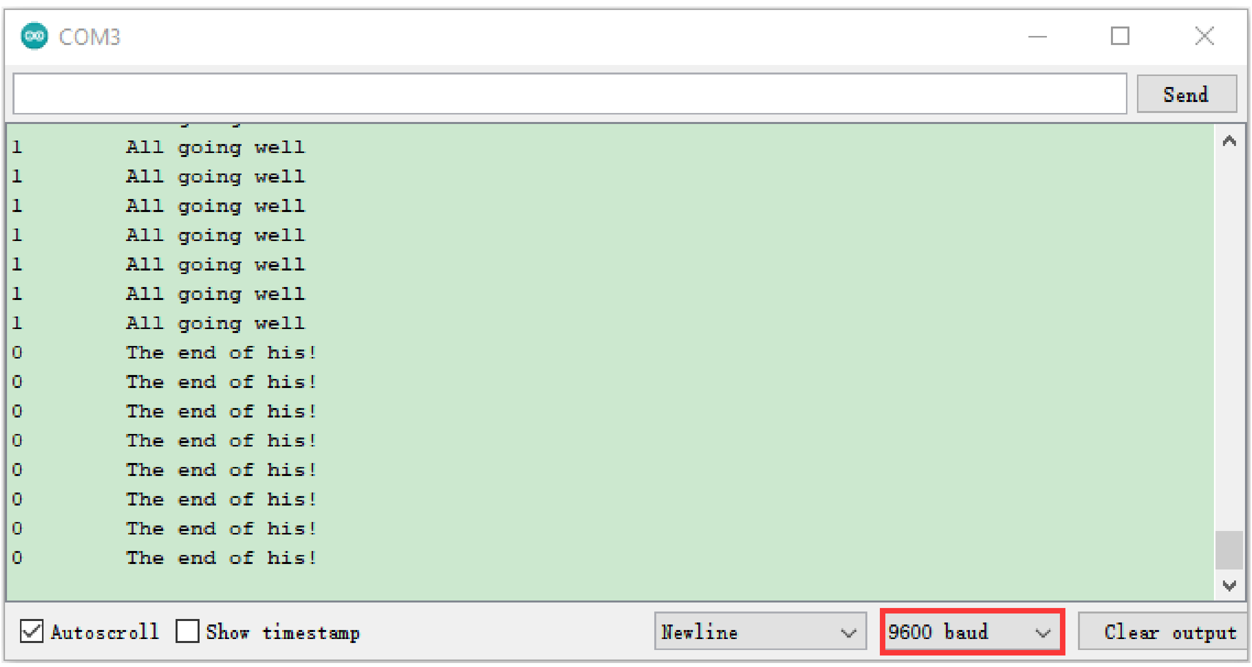
(continued from previous page)

```
  if (val == 0) {//Collision val is 0
    Serial.print("        ");
    Serial.println("The end of his!");
    delay(100);
  }
  else {// No collision val is 1
    Serial.print("        ");
    Serial.println("All going well");
    delay(100);
  }
}
//*************************************************************************************
```

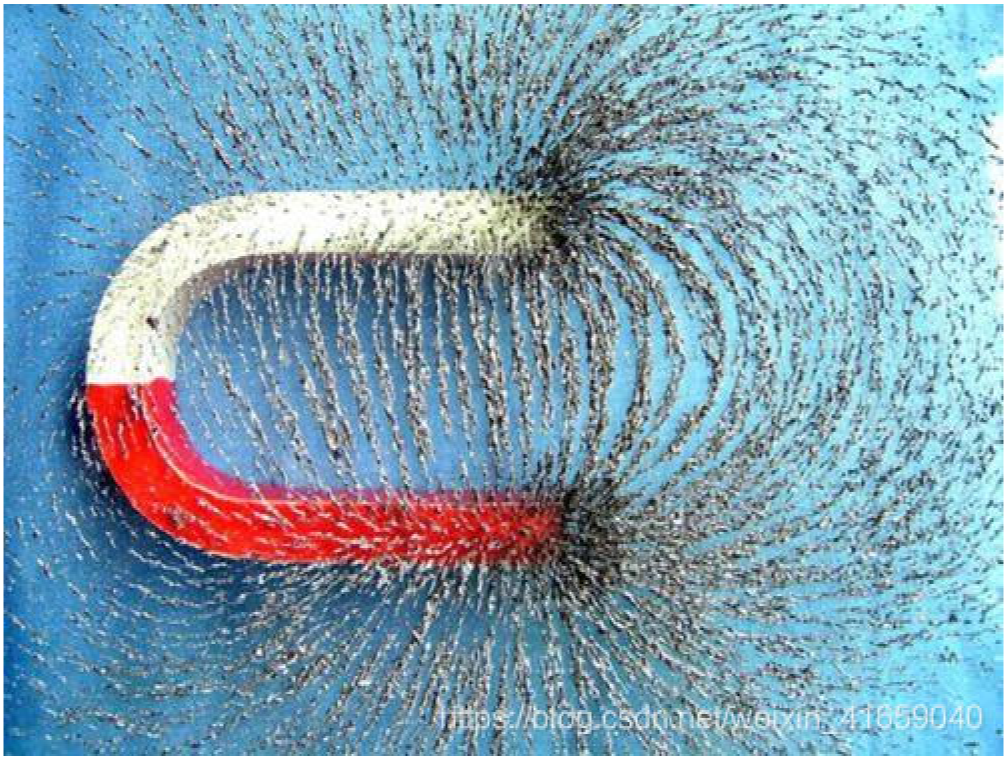
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. The serial monitor will display the corresponding data and characters.

In the experiment, when the shrapnel on the sensor is pressed down, val is 0, the red LED of the module is on, and "0 The end of his!" is printed; when the shrapnel is released, the val is 1, the red LED of the module is off, and "1 All going well" is printed. !" character, as shown below.



6.2.14 Project 14: Hall Sensor

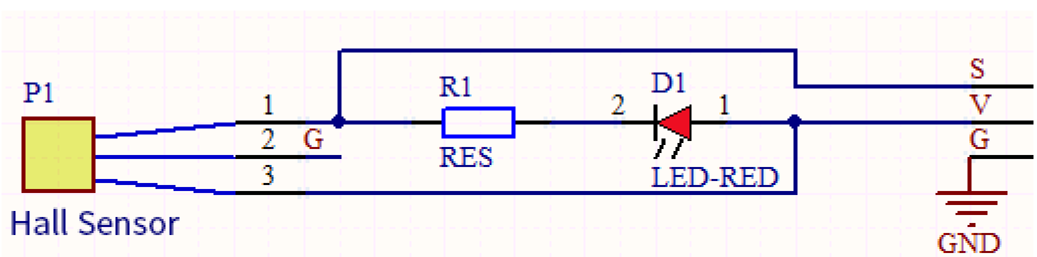


Description

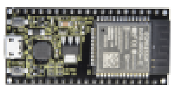
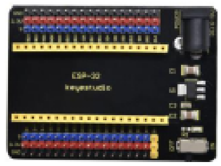
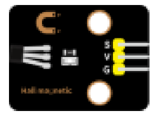
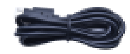
In this kit, there is a Hall sensor which mainly adopts a A3144 linear Hall element. The element P1 is composed of a voltage regulator, a Hall voltage generator, a differential amplifier, a Schmitt trigger, a temperature compensation circuit and an open-collector output stage. In the experiment, we use the Hall sensor to detect the magnetic field and display the test results on the shell.

Working Principle

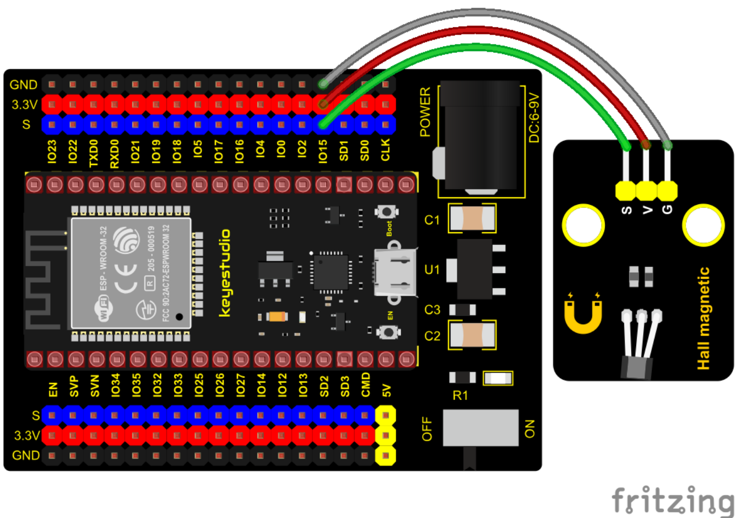
When the sensor detects no magnetic field or a north pole magnetic field, the signal terminal will be high level; when it senses a south pole magnetic field, the signal terminal will be low levels. The stronger the magnetic field strength is, induction distance is longer.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Hall Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Hall magnetic
 * Description : Reading the value of hall magnetic sensor
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int hallPin = 15;  //Hall sensor pin is connected to GPIO15
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(hallPin, INPUT);//Set pin to input mode
}

void loop() {
  val = digitalRead(hallPin);//Read the level value of hall sensor
  Serial.print(val);//Print val
  if (val == 0) {//There is a South Pole magnetic field
    Serial.println("      The magnetic field at the South Pole!");
  }
```

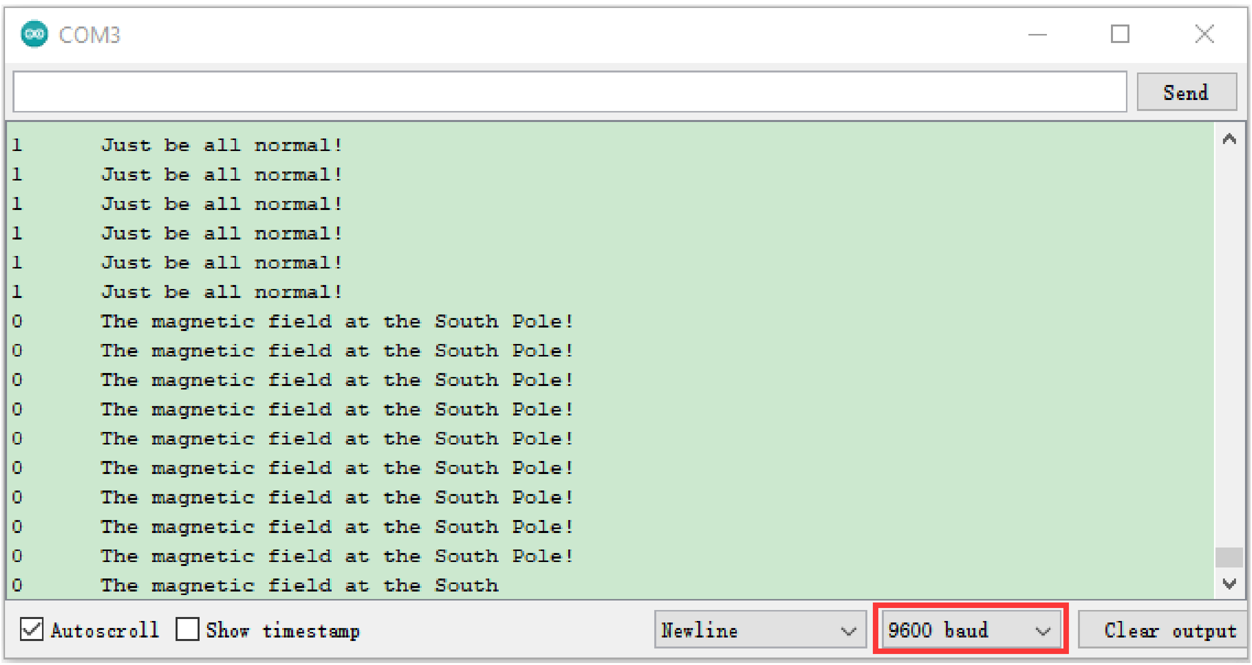
(continues on next page)

(continued from previous page)

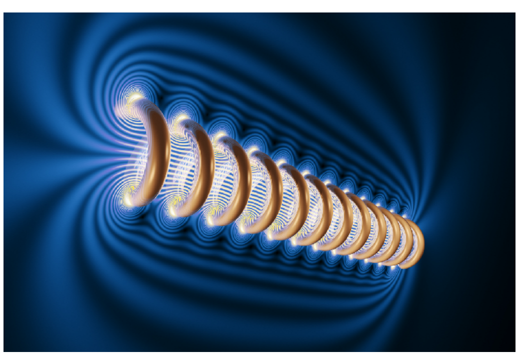
```
  else {//If not
    Serial.println("      Just be all normal!");
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. when the sensor detects no magnetic fields or the north pole magnetic field, the monitor l will show "1 Just be all normal!" and the LED on the sensor will be off; When it detects the south pole magnetic field,"0 The magnetic field at the South Pole!"and the LED on the sensor will be on.



6.2.15 Project 15: Reed Switch Module



Overview

In this kit, there is a Keyestudio reed switch module, which mainly uses a MKA10110 green reed component.

The reed switch is the abbreviation of the dry reed switch. It is a passive electronic switch element with contacts.

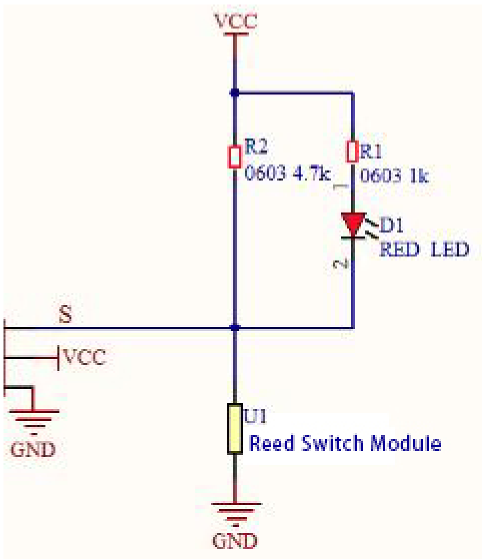
It has the advantages of simple structure, small size and easy control.

Its shell is a sealed glass tube with two iron elastic reed electric plates.

In the experiment, we will determine whether there is a magnetic field near the module by reading the high and low level of the S terminal on the module; and, we display the test result in the shell.

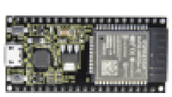
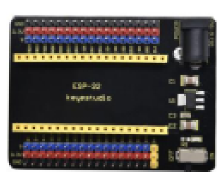
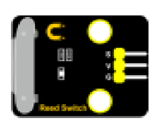
Working Principle

In normal conditions, the glass tube in the two reeds made of special materials are separated. When a magnetic substance close to the glass tube, in the role of the magnetic field lines, the pipe within the two reeds are magnetized to attract each other in contact, the reed will suck together, so that the junction point of the connected circuit communication.

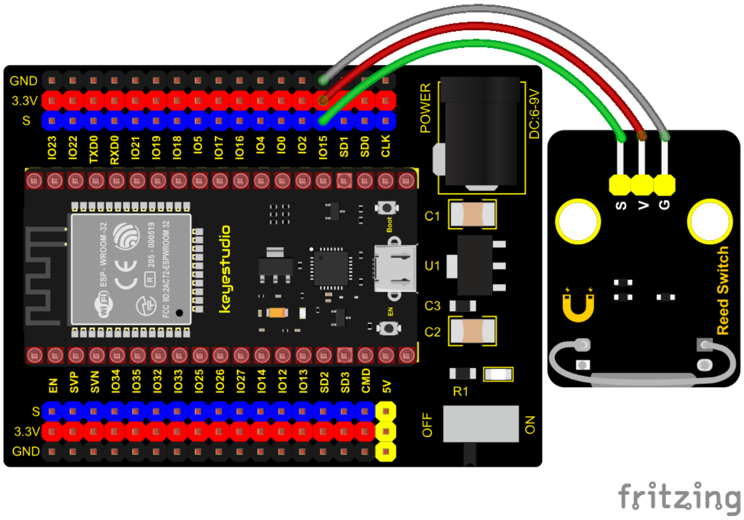


After the disappearance of the outer magnetic reed because of their flexibility and separate, the line is disconnected. The sensor uses this characteristic to build a circuit to convert magnetic field signal into high and low level signal.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Reed Switch Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



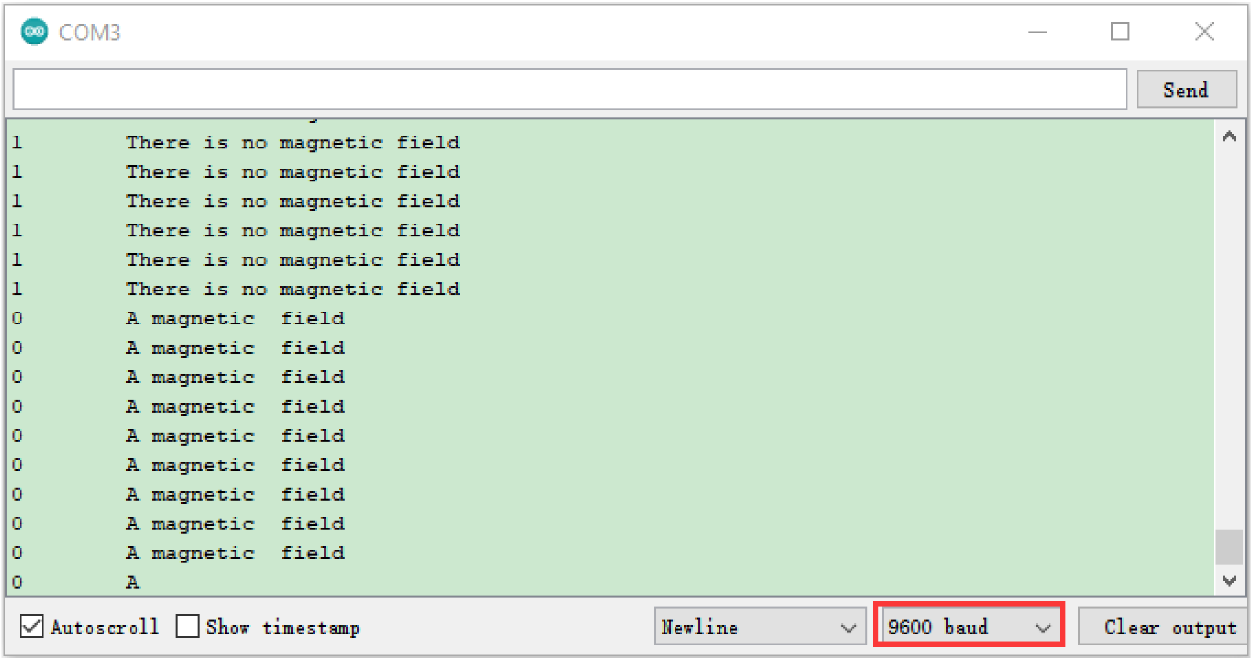
Test Code

```
//**********************************************************************************
/*
 * Filename    : Reed Switch
 * Description : Read the value of the reed sensor
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int reedPin = 15; //Define dry reed module signal pin connected to GPIO15
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(reedPin, INPUT);//Set mode to input
}

void loop() {
  val = digitalRead(reedPin);//Read digital level
  Serial.print(val);//Serial port shows up

  if (val == 0) {//There's a magnetic field nearby
    Serial.print("        ");
    Serial.println("A magnetic  field");
    delay(100);
  }
  else {//There is no magnetic field
    Serial.print("        ");
    Serial.println("There is no magnetic field");
    delay(100);
  }
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. The serial monitor will display the corresponding data and characters.

When the sensor detects a magnetic field, val is 0 and the red LED of the module lights up, "0 A magnetic field" will be displayed; when no magnetic field is detected, val is 1, and the LED on the module goes out, "1 There is no magnetic field" will be shown, as shown below.



6.2.16 Project 16: PIR Motion Sensor

Overview

In this kit, there is a Keyestudio PIR motion sensor, which mainly uses an RE200B-P sensor elements. It is a human body pyroelectric motion sensor based on pyroelectric effect, which can detect infrared rays emitted by humans or animals, and the Fresnel lens can make the sensor's detection range farther and wider.

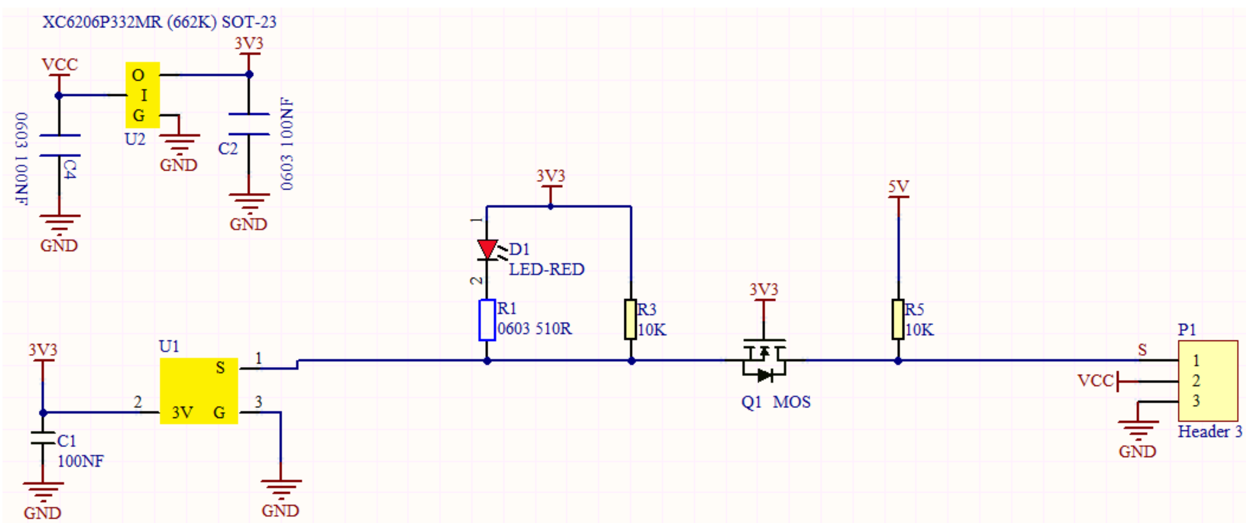
In the experiment, we determine if there is someone moving nearby by reading the high and low levels of the S terminal on the module. The detected results will be displayed on the Shell.

Working Principle

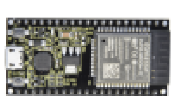
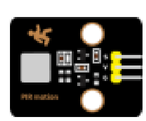
The upper left part is voltage conversion(VCC to 3.3V). The working voltage of sensors we use is 3.3V, therefore we can't use 5V directly. The voltage conversion circuit is needed.

When no person is detected or no infrared signal is received, and pin 1 of the sensor outputs low level. At this time, the LED on the module will light up and the MOS tube Q1 will be connected and the signal terminal S will detect Low levels.

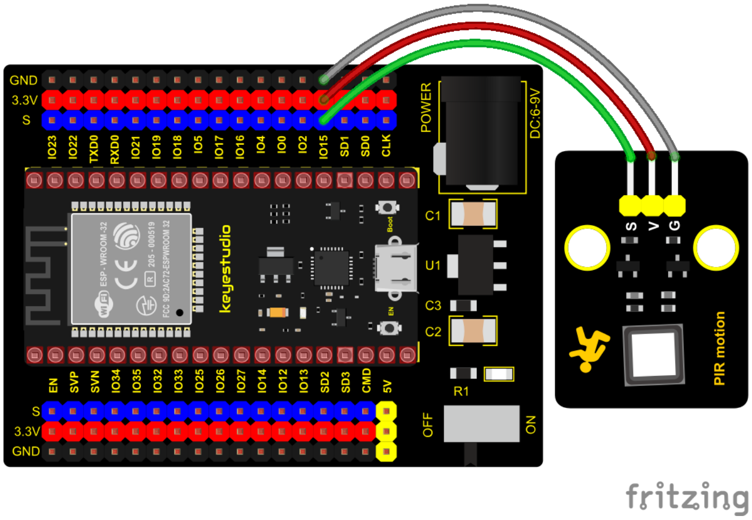
When one is detected or an infrared signal is received, and pin 1 of the sensor outputs a high level. Then LED on the module will go off, the MOS tube Q1 is disconnected and the signal terminal S will detect high levels.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY PIR Motion Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



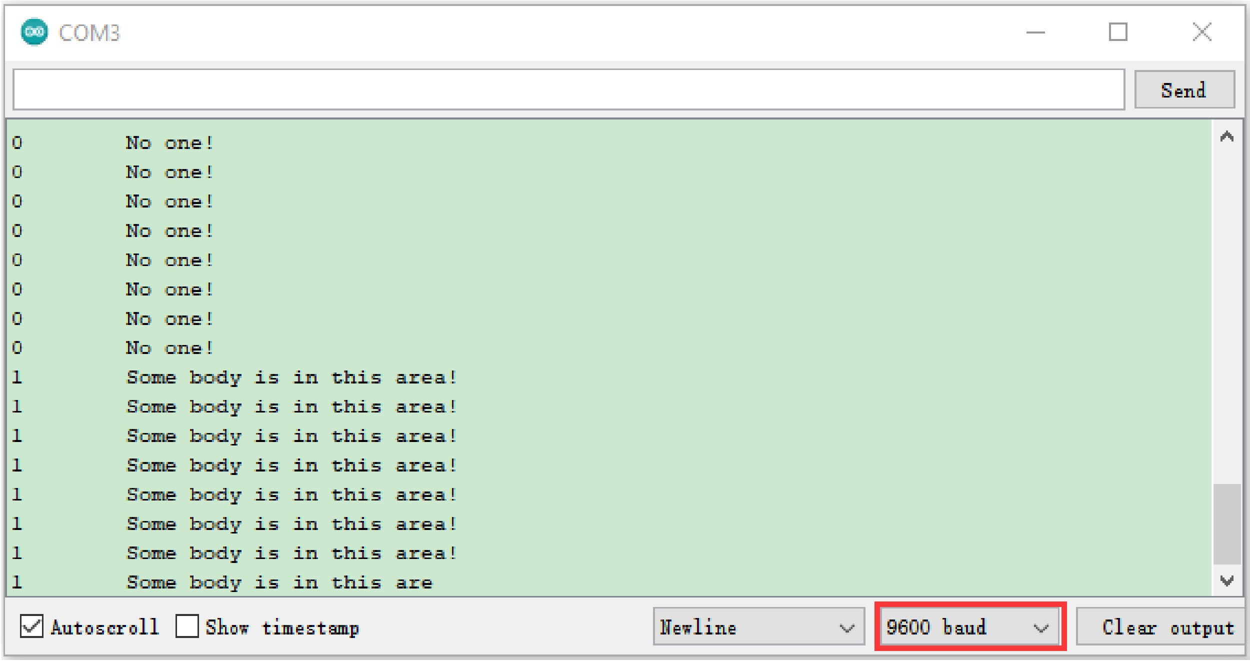
Test Code

```
//**********************************************************************************
/*
 * Filename    : PIR motion
 * Description : Reading the value of the human body infrared sensor
 * Auther      :  http://www.keyestudio.com
*/
int val = 0;
int pirPin = 15; //The pin of PIR motion sensor is defined as GPIO15
void setup() {
  Serial.begin(9600); //Set baud rate to 9600
  pinMode(pirPin, INPUT);  //Set the sensor to input mode
}

void loop() {
  val = digitalRead(pirPin);  //Read the sensor value
  Serial.print(val);//Print val value
  if (val == 1) {//There is movement nearby, output high level
    Serial.print("        ");
    Serial.println("Some body is in this area!");
    delay(100);
  }
  else {//If no  movement nearby, output low level
    Serial.print("        ");
    Serial.println("No one!");
    delay(100);
  }
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. The serial monitor will display the corresponding data and characters.

When the sensor detects someone nearby, value is 1, the LED will go off and the monitor will show"1 Somebody is in this area!". In contrast, the value is 0, the LED will go up and"0 No one!"will be shown.



6.2.17 Project 17: Active Buzzer

Overview

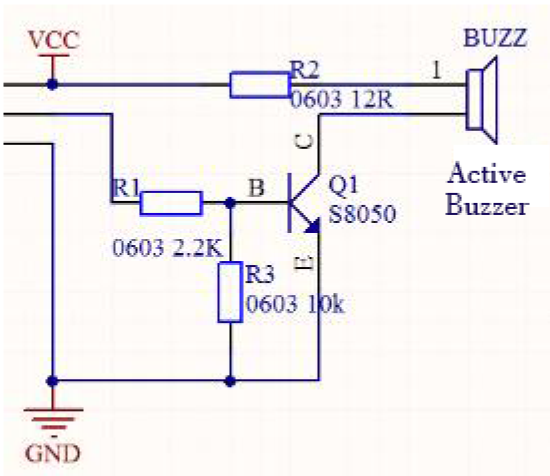
In this kit, it contains an active buzzer module and a power amplifier module (the principle is equivalent to a passive buzzer). In this experiment, we control the active buzzer to emit sounds. Since it has its own oscillating circuit, the buzzer will automatically sound if given large voltage.

Working Principle

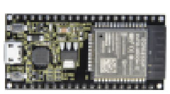
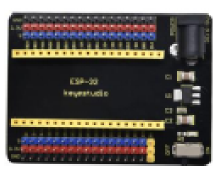
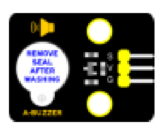
From the schematic diagram, the pin of buzzer is connected to a resistor R2 and another port is linked with a NPN triode Q1. So, if this triode Q1 is powered, the buzzer will sound.

If the base electrode of the triode connected to the R1 resistor is a high level, the triode Q1 will be connected.If the base electrode is pulled down by the resistor R3, the triode is disconnected.

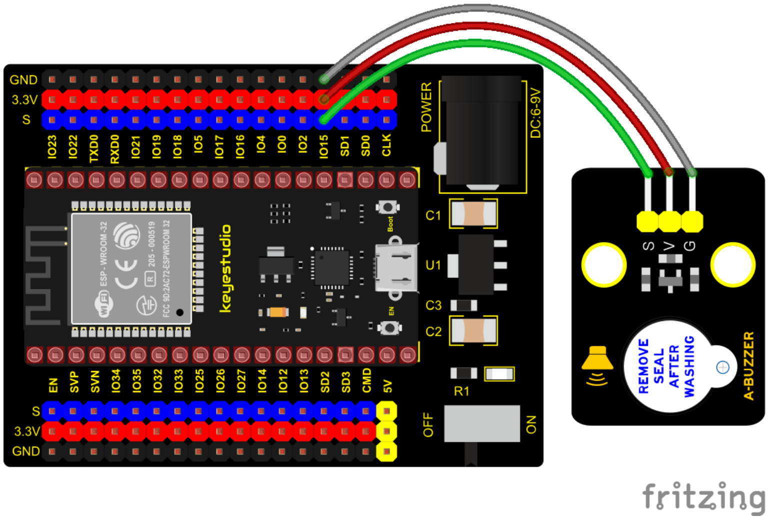
When we output a high level from the IO port to the triode, the buzzer will emit sounds; if outputting low levels, the buzzer won't emit sounds.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Active buzzer
 * Description : An active buzzer produces sound
 * Auther      : http://www.keyestudio.com
*/
int buzzer = 15; //Define buzzer receiver pin GPIO15
void setup() {
  pinMode(buzzer, OUTPUT);//Set the output mode
}

void loop() {
  digitalWrite(buzzer, HIGH); //sound production
  delay(1000);
  digitalWrite(buzzer, LOW); //Stop the sound
  delay(1000);
}
//**********************************************************************************
```

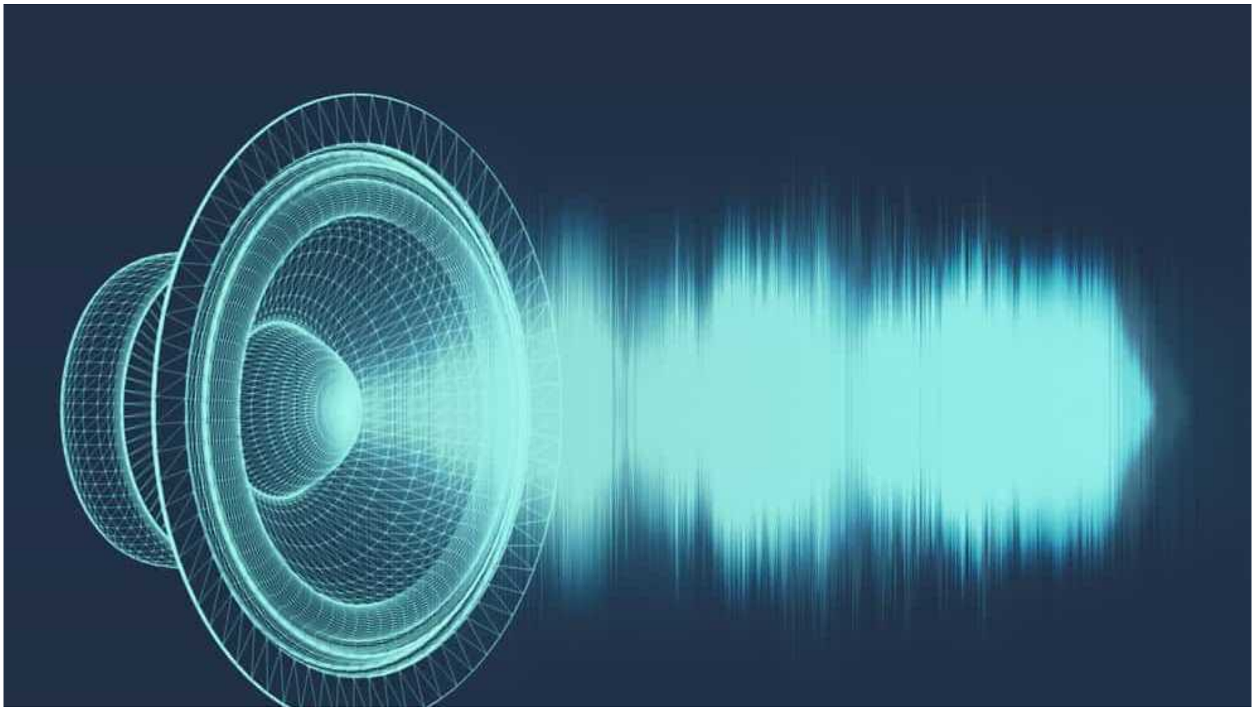
Code Explanation

In the experiment, we set the pin to GPIO15. When setting to high, the active buzzer will beep; when setting to low, the active buzzer will stop emitting sounds.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The active buzzer will emit sound for 1 second, and stop for 1 second.

6.2.18 Project 18: 8002b Audio Power Amplifier



Overview

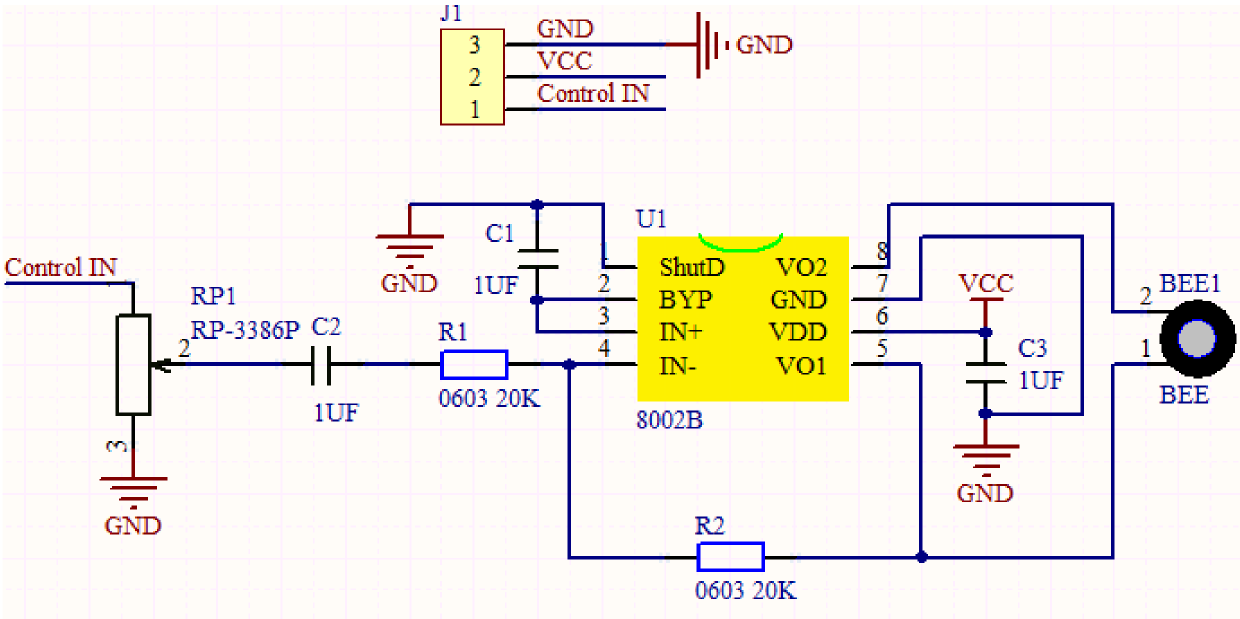
In this kit, there is a Keyestudio 8002b audio power amplifier. The main components of this module are an adjustable potentiometer, a speaker, and an audio amplifier chip.

The main function of this module is: it can amplify the output audio signal, with a magnification of 8.5 times, and play sound or music through the built-in low-power speaker, as an external amplifying device for some music playing equipment.

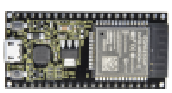
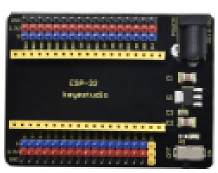
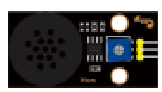
In the experiment, we used the 8002b power amplifier speaker module to emit sounds of various frequencies.

Working Principle

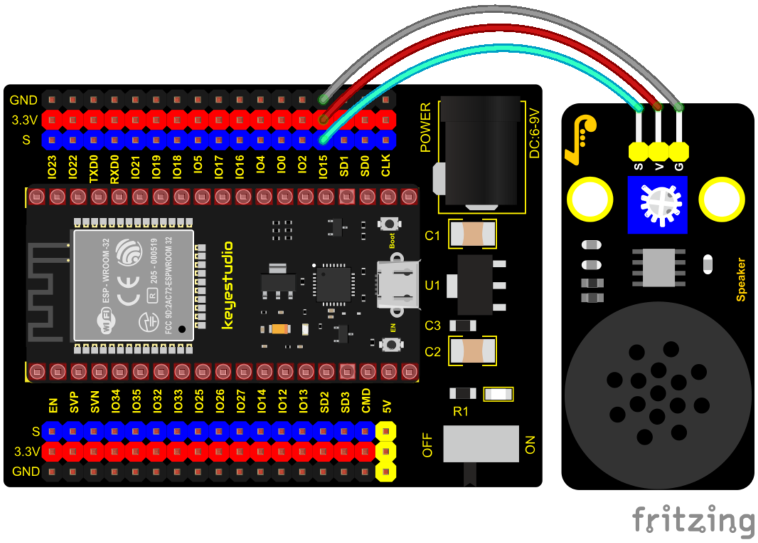
In fact, it is similar to a passive buzzer. The active buzzer has its own oscillation source. Yet, the passive buzzer does not have internal oscillation. When controlling the circuit, we need to input square waves of different frequencies to the positive pole of the component and ground the negative pole to control the buzzer to chime sounds of different frequencies.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 8002b Audio Power Amplifier*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : Passive Buzzer
 * Description : Passive Buzzer sounds the alarm.
 * Auther      : http//www.keyestudio.com
*/
#define LEDC_CHANNEL_0 0

// LEDC timer uses 13 bit accuracy

#define LEDC_TIMER_13_BIT  13

// Define tool I/O ports

#define BUZZER_PIN  15

//Create a musical melody list, Super Mario

int melody[] = {330, 330, 330, 262, 330, 392, 196, 262, 196, 165, 220, 247, 233, 220,␣
↪196, 330, 392, 440, 349, 392, 330, 262, 294, 247, 262, 196, 165, 220, 247, 233, 220,␣
↪196, 330, 392,440, 349, 392, 330, 262, 294, 247, 392, 370, 330, 311, 330, 208, 220,␣
↪262, 220, 262,

294, 392, 370, 330, 311, 330, 523, 523, 523, 392, 370, 330, 311, 330, 208, 220, 262,220,␣
↪262, 294, 311, 294, 262, 262, 262, 262, 262, 294, 330, 262, 220, 196, 262, 262,262,␣
↪262, 294, 330, 262, 262, 262, 262, 294, 330, 262, 220, 196};

//Create a list of tone durations

int noteDurations[] = {8,4,4,8,4,2,2,3,3,3,4,4,8,4,8,8,8,4,8,4,3,8,8,3,3,3,3,4,4,8,4,8,8,
↪8,4,8,4,3,8,8,2,8,8,8,4,4,8,8,4,8,8,3,8,8,8,4,4,4,8,2,8,8,8,4,4,8,8,4,8,8,3,3,3,1,8,4,
↪4,8,4,8,4,8,2,8,4,4,8,4,1,8,4,4,8,4,8,4,8,2};
void setup() {
```

(continues on next page)

(continued from previous page)

```
pinMode(BUZZER_PIN, OUTPUT); // Set the buzzer to output mode
}

void loop() {

  int noteDuration; //Create a variable of noteDuration

  for (int i = 0; i < sizeof(noteDurations); ++i)

  {
      noteDuration = 800/noteDurations[i];

      ledcSetup(LEDC_CHANNEL_0, melody[i]*2, LEDC_TIMER_13_BIT);

      ledcAttachPin(BUZZER_PIN, LEDC_CHANNEL_0);

      ledcWrite(LEDC_CHANNEL_0, 50);

      delay(noteDuration * 1.30); //delay
  }
}
//*************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onthen the power amplifier module will emit the sound on a loop.

6.2.19 Project 19: 130 Motor

Description

The 130 motor driver module is compatible with servo motors, which has high efficiency and good quality fans.

It adopts a HR1124S motor control chip. HR1124S is a single-channel H-bridge driver chip for DC motor solutions. In addition, this chip has low standby current and low quiescent current.

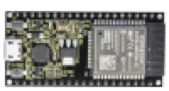
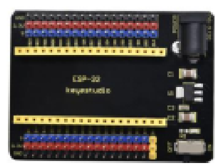
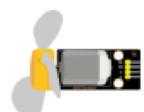
The module is compatible with various single-chip control boards. In the experiment, we can control the rotation direction of the motor by outputting the voltage directions of the two signal terminals IN+ and IN- to make the motor rotate.

Working Principle

The chip is used to help drive the motor. We can't drive it with a triode or an IO port due to its a large current of need. It is very simple to make the motor rotate. Just apply voltage to both ends of the motor. The direction of the motor is different in different voltage directions. Within the rated voltage, the higher the voltage, the faster the motor rotates; on the contrary, the lower the voltage, the slower the motor rotates, or even unable to rotate.

So we can use the PWM port to control the speed of the motor. We haven't learned PWM here, so we use the high and low levels to control the motor first.

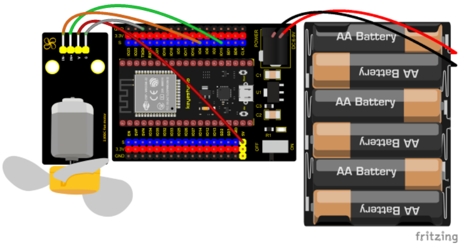
Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY 130 Motor*1	4P Dupont Wire*1
			
Micro USB Cable*1	Battety Holder*1	Battety(not included in kit)*6	

Note: the motor is separated with its fan, you need to assemble it first.

Connection Diagram

130 Motor	ESP32 Expansion Board
G	G
V	5V
IN+	IO15
IN-	IO4



Test Code

```
//*************************************************************************************
/*
 * Filename    : 130DC Fan motor
 * Description : Motor positive and negative rotation
 * Auther      : http://www.keyestudio.com
*/
//Define two pins interfaces of the motor, respectively 15 and 4
int INA = 15; //INA corresponds to IN+
int INB = 4;  //INB corresponds to IN-
void setup() {
  //Set the motor pins as output
  pinMode(INA, OUTPUT);
  pinMode(INB, OUTPUT);
}

void loop() {
  //Turn counterclockwise
  digitalWrite(INA, HIGH);
  digitalWrite(INB, LOW);
  delay(2000);
  //stop
  digitalWrite(INA, LOW);
  digitalWrite(INB, LOW);
  delay(1000);
  //clockwise rotation
  digitalWrite(INA, LOW);
  digitalWrite(INB, HIGH);
  delay(2000);
  //stop
  digitalWrite(INA, LOW);
  digitalWrite(INB, LOW);
  delay(1000);
}
//*************************************************************************************
```

Code Explanation

Set pins to GPIO4GPIO15, when the pin GPIO4 outputs low levels and the pin GPIO15 outputs high levels, the motor will rotate counterclockwise; when both pins are set to low, the motor stops rotating.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch ON the ESP32 expansion board to the ON end, after powering on, compile and upload the code to the ESP32. After uploading successfullythe fan will rotate counterclockwise for 2 seconds, stop for 1 second and clockwise for 2 seconds and stop for 1 second; cycle alternately.

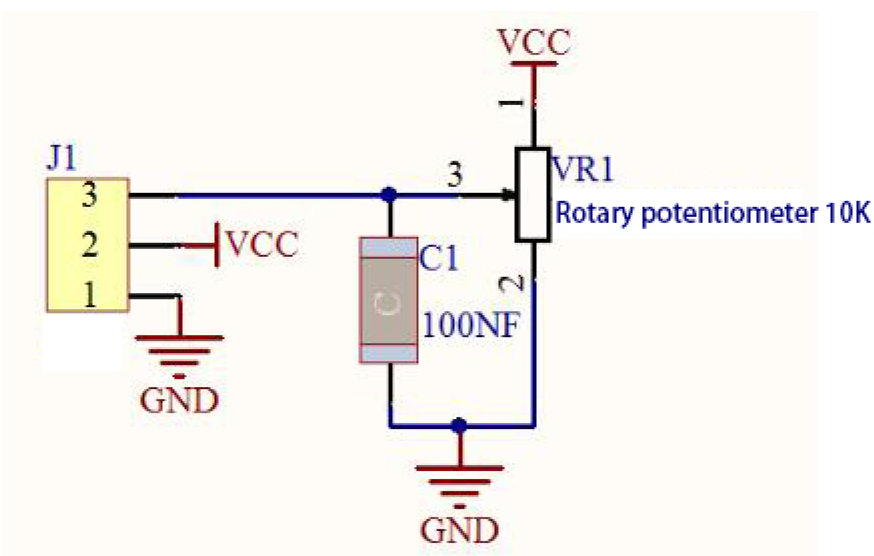
6.2.20 Project 20: Potentiometer

Overview

The following we will introduce is the Keyestudio rotary potentiometer which is an analog sensor.

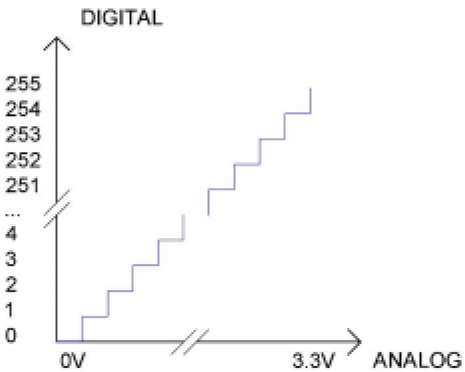
The digital IO ports can read the voltage value between 0 and 3.3V and the module only outputs high levels. However, the analog sensor can read the voltage value through 16 ADC analog ports on the ESP32 board. In the experiment, we will display the test results on the Shell.

Working Principle



It uses a 10K adjustable resistor. We can change the resistance by rotating the potentiometer. The signal S can detect the voltage changes(0-3.3V) which are analog quantity.

ADC: The more bits an ADC has, the denser the partitioning of the simulation, the higher the accuracy of the final conversion.



Subsection 1: The analog value within 0V—3.3/4095 V corresponds to the number 0; Subsection 2: The analog value within 3.3/4095V—2*3.3/4095V corresponds to the number 1; ……

The conversion formula is as follows:

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

DAC: The higher the precision of DAC, the higher the precision of the output voltage value.

The conversion formula is as follows:

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

ADC on ESP32

The ESP32 has 16 pins that can be used to measure analog signals. GPIO pin serial numbers and analog pin definitions are shown below:

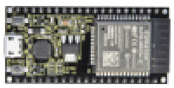
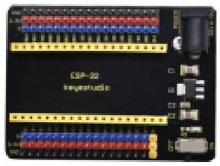
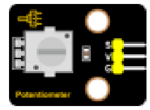
ADC number in ESP32	ESP32 GPIO number
ADC0	GPIO 36
ADC3	GPIO 39
ADC4	GPIO 32
ADC5	GPIO33
ADC6	GPIO34
ADC7	GPIO 35
ADC10	GPIO 4
ADC11	GPIO0
ADC12	GPIO2
ADC13	GPIO15
ADC14	GPIO13
ADC15	GPIO 12
ADC16	GPIO 14
ADC17	GPIO27
ADC18	GPIO25
ADC19	GPIO26

DAC on ESP32

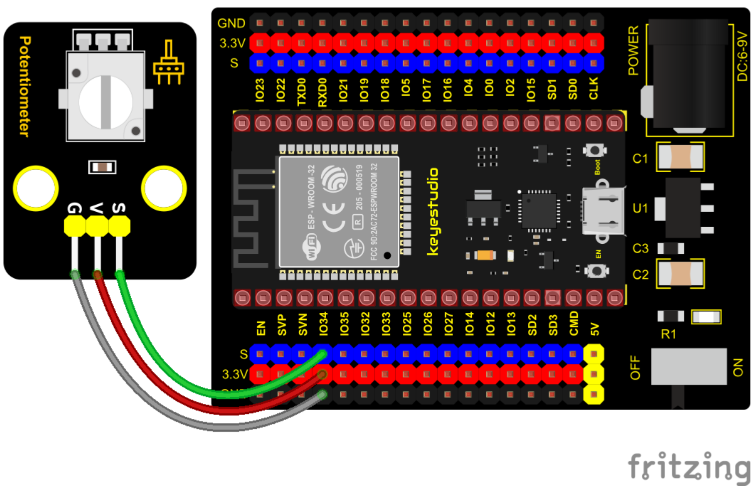
The ESP32 has two 8-bit digital-to-analog converters connected to GPIO25 and GPIO26 pins, which are immutable, as shown below :

Simulate pin number	GPIO number
DAC1	GPIO25
DAC2	GPIO26

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Potentiometer*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Rotary_potentiometer
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN   34  //the pin of the Potentiometer

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
```

(continues on next page)

(continued from previous page)

```
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//*************************************************************************************
```

Code Explanation

1). analogVal means analog value. The rotary potentiometer outputs analog values(0~4095), therefore, we set pins to analog ports. For example, we connect to GPIO34.

2). analogRead(pin): read the value of the specified analog pin. The ESP32 contains a multi-channel, 12-bit converter. This means that it will map the input voltage between 0 and the working voltage (5V or 3.3V) to an integer value between 0 and 4095. For example, this will produce a resolution among readings: 3.3V/4096 stands for 0.0008V per unit.

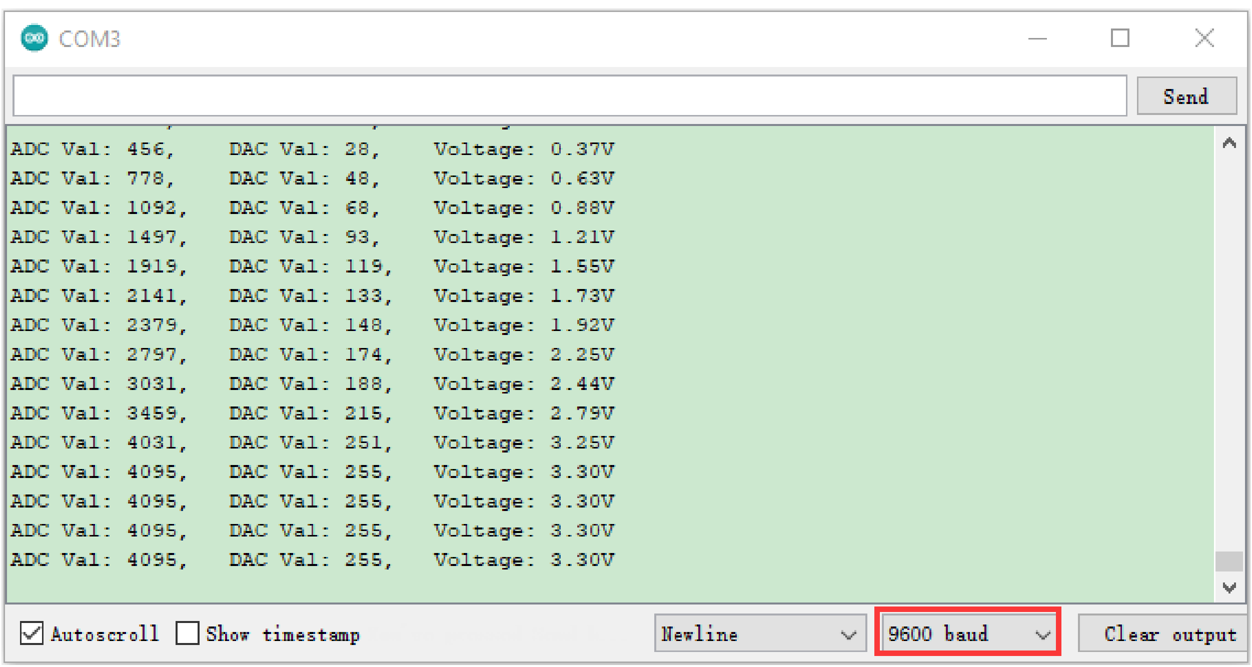
3). The map() function converts this 12-bit DAC value to an 8-bit DAC value.

4). Pin: the name of analog input pin.

5). The serial monitor displays the values of adcVal, dacVal, voltage, the baud rate must be set before display (we default to 9600,which can be changed).

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the potentiometer's ADC value, DAC value and voltage value. Rotate the potentiometer handle, the analog values will change.



6.2.21 Project 21: Steam Sensor

Description

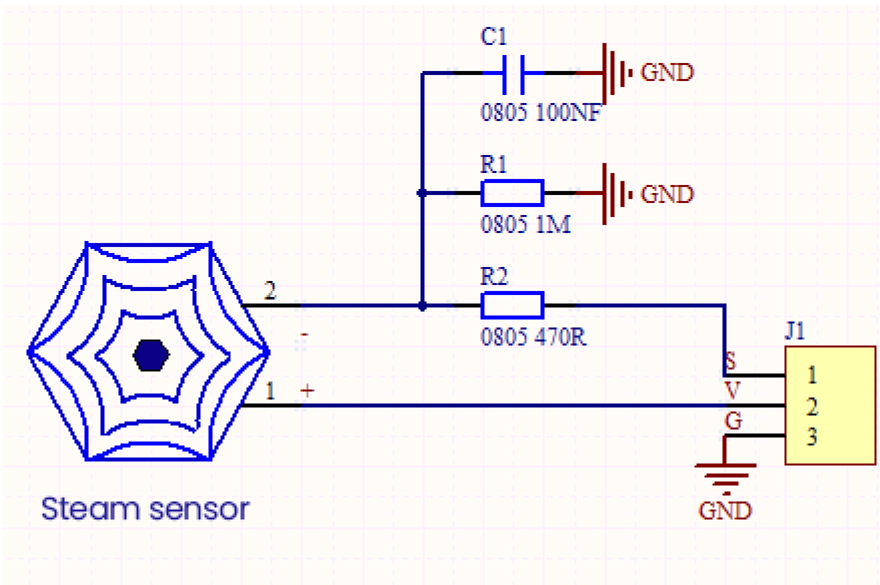
This is a DIY electronic building block water drop sensor. It is an analog (digital) input module, also called rain, rain sensor. It can be used to monitor various weather conditions, detect whether it is raining and the amount of rain, convert it into digital signal (DO) and analog signal (AO) output, and is widely used in Arduino robot kits, raindrops, rain sensors, and can be used for various It can monitor various weather conditions, and convert it into digital signal and AO output, and can also be used for automobile automatic wiper system, intelligent lighting system and intelligent sunroof system.

In the experiment, we input the sensor signal terminal (S terminal) to the analog port of the ESP32 development board, sense the change of the analog value, and display the corresponding analog value on the shell.

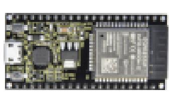
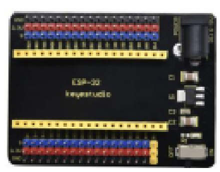
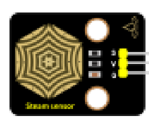
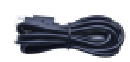
Working Principle

Its principle is to detect the amount of water through the exposed printed parallel lines on the circuit board. The more water there is, the more wires will be connected, and the conductive contact area increases. The voltage output by pin 2 will gradually increase. The larger the analog value detected by the signal terminal S is.

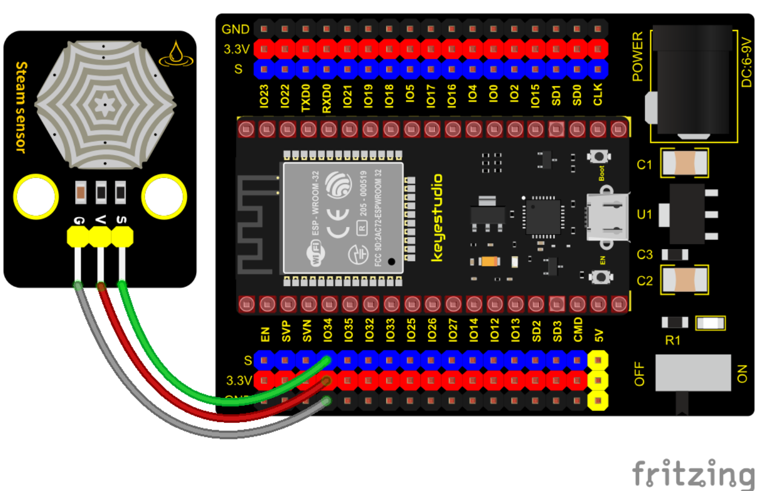
It can also detect steam in the air. Two position holes are used to install on the other devices.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Steam Sensor *1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



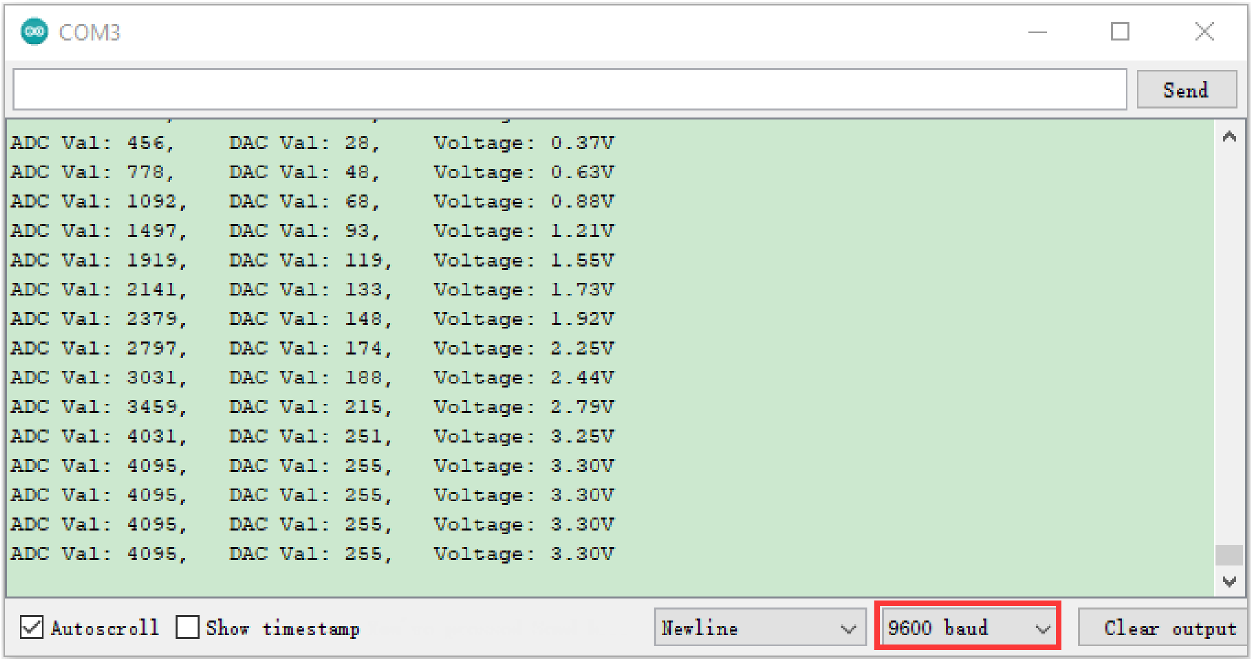
Test Code

```
//**********************************************************************************
/*
 * Filename    : Steam sensor
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Steam sensor

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal,␣
↪voltage);
  delay(200);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32.

After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the steam sensor's ADC value, DAC value and voltage value. When a few drops of water are placed in the sensor sensing area, the values will change. The more water volume, the greater the output voltage value , ADC value and the DAC value .



6.2.22 Project 22: Sound Sensor

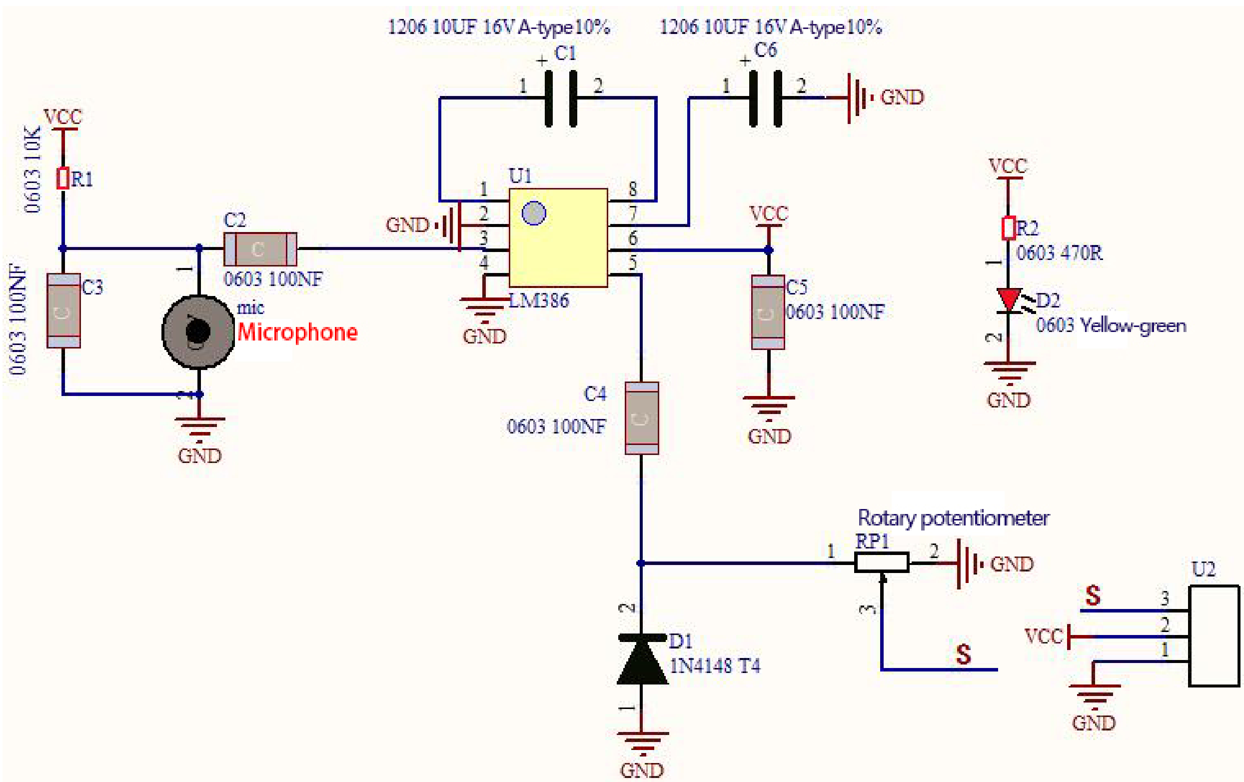
Overview

In this kit, there is a Keyestudio DIY electronic block and a sound sensor.

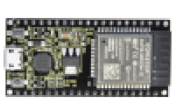
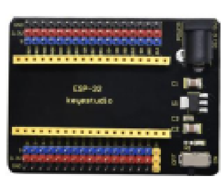
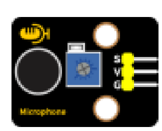
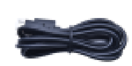
In the experiment, we test the analog value corresponding to the sound level in the current environment with it. The louder the sound, the larger the ADC, DAC and the voltage value, and the serial monitor window will display the test results.

Working Principle

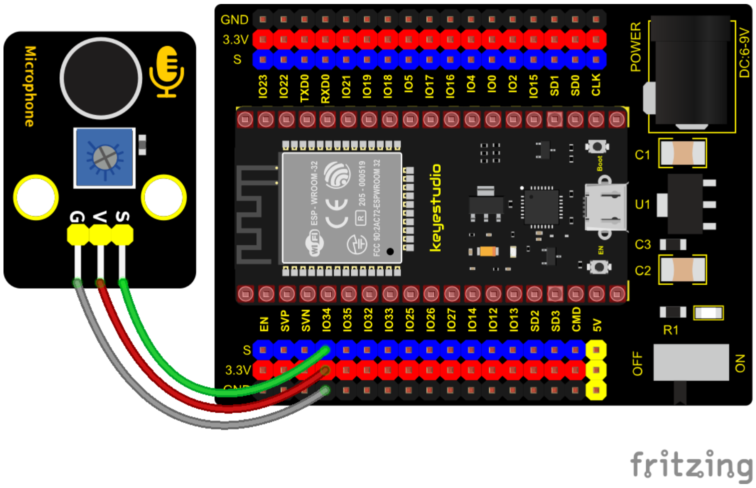
It uses a high-sensitive microphone component and an LM386 chip. We build the circuit with the LM386 chip and amplify the sound through the high-sensitive microphone. In addition, we can adjust the sound volume by the potentiometer. Rotate it clockwise, the sound will get louder.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Sound Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram

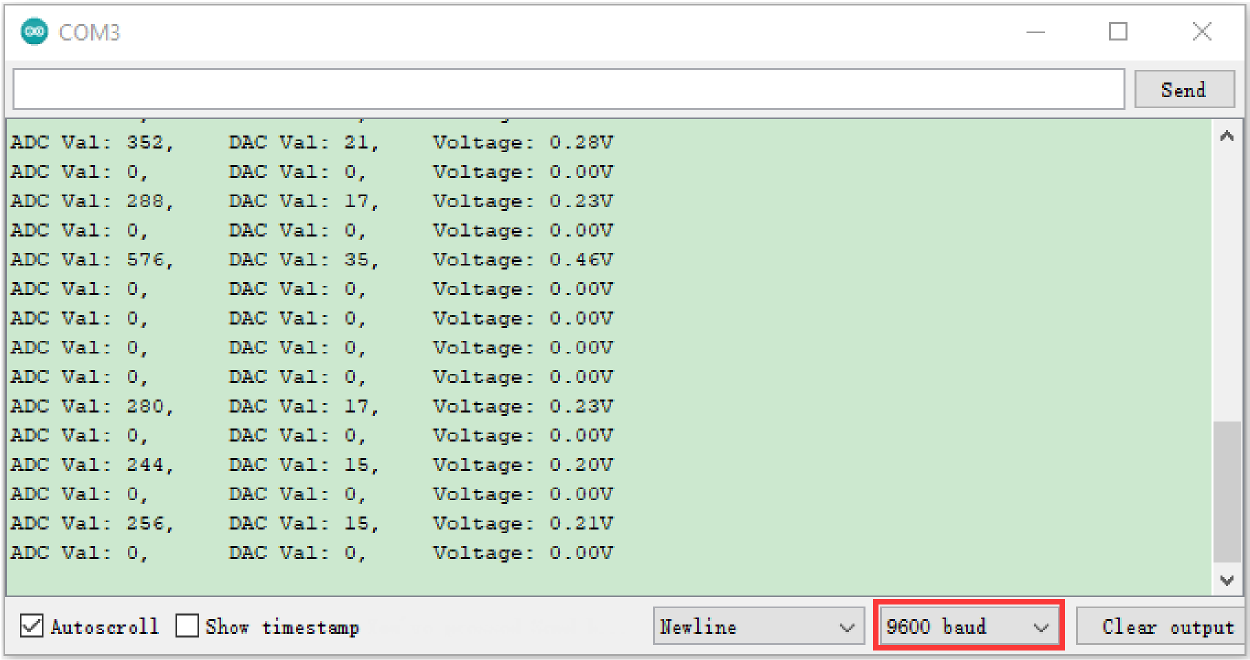


Test Code

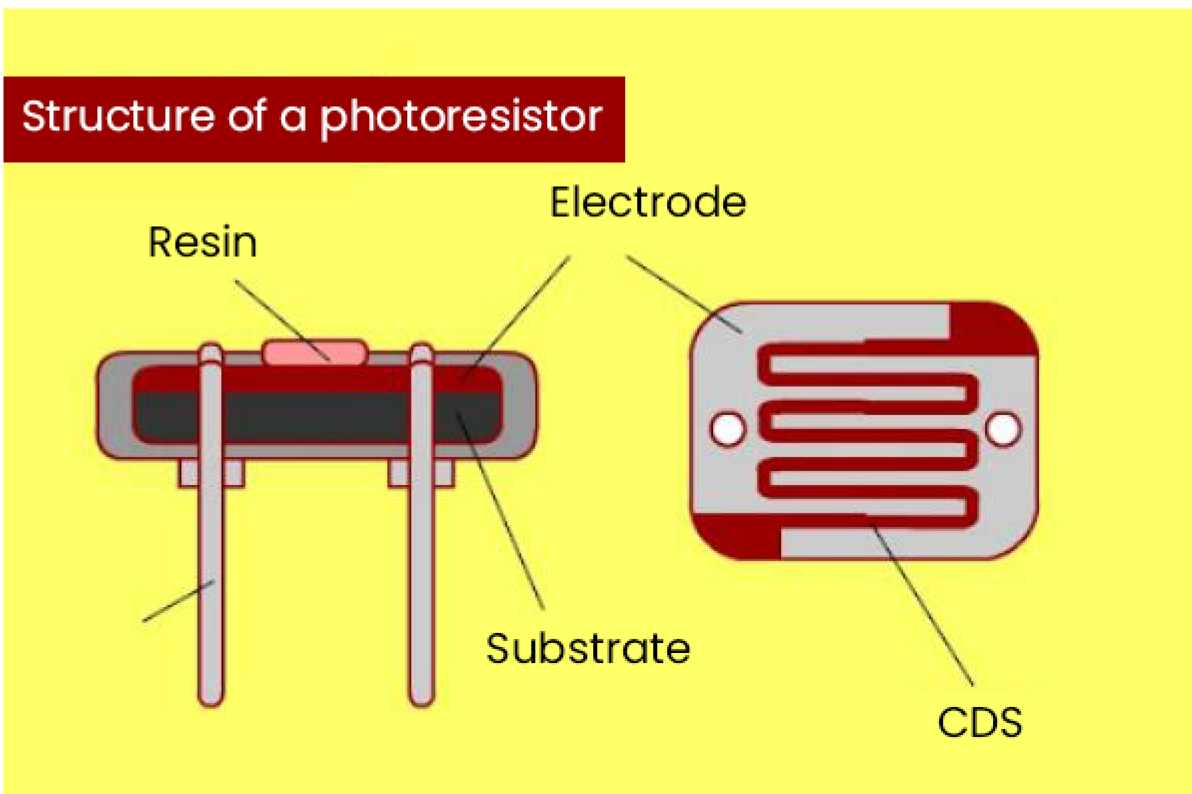
```
//**********************************************************************************
/*
 * Filename    : MicroPhone
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Sound Sensor

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the sound sensor's ADC value, DAC value and voltage value. Rotate the potentiometer clockwise and speak at the MIC. Then you can see the analog value get larger, as shown below:



6.2.23 Project 23: Photoresistor

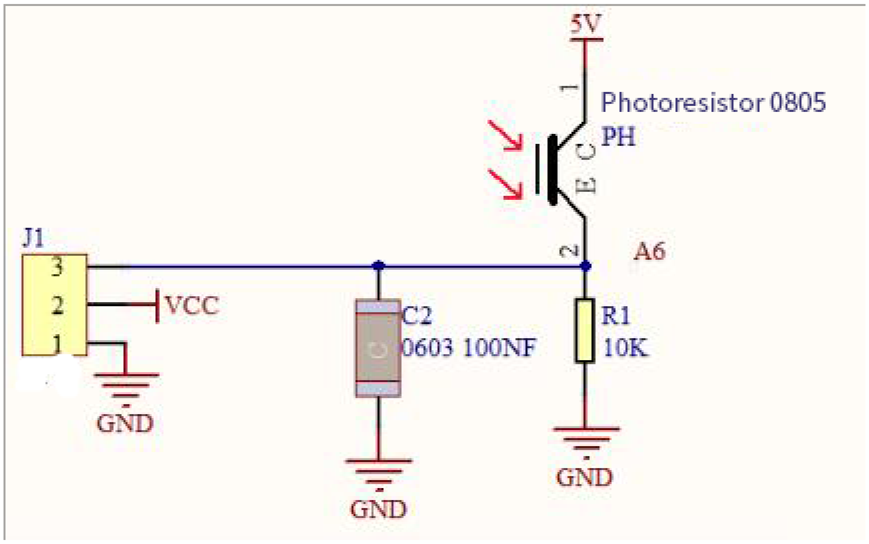


Description

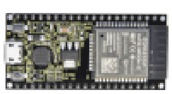
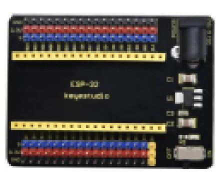
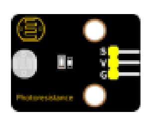
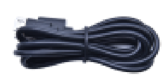
In this kit, there is a photoresistor consists of photosensitive resistance elements. Its resistance changes with the light intensity. Also, it converts the resistance change into a voltage change through the characteristic of the photosensitive resistive element. When wiring it up, we interface its signal terminal (S terminal) with the analog port of ESP32 , so as to sense the change of the analog value, and display the corresponding analog value in the shell.

Working Principle

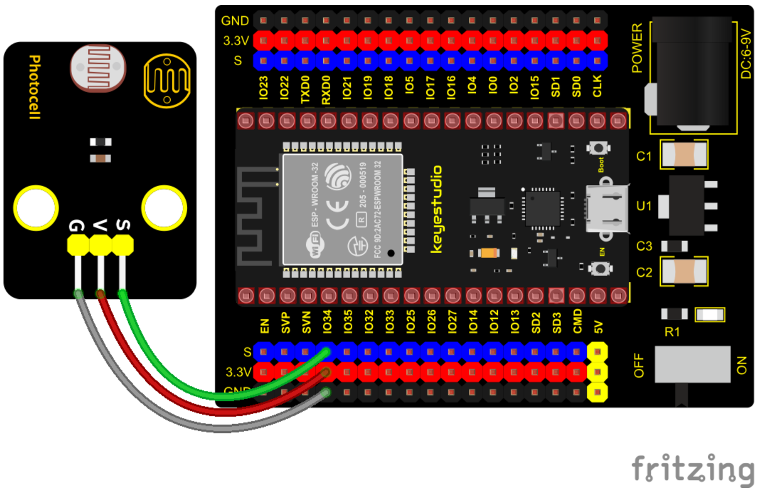
If there is no light, the resistance is 0.2M and the detected voltage at the terminal is close to 0. When the light intensity increases, the resistance of photoresistor and detected voltage will diminish, and the detected voltage is increasing.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Photoresistor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



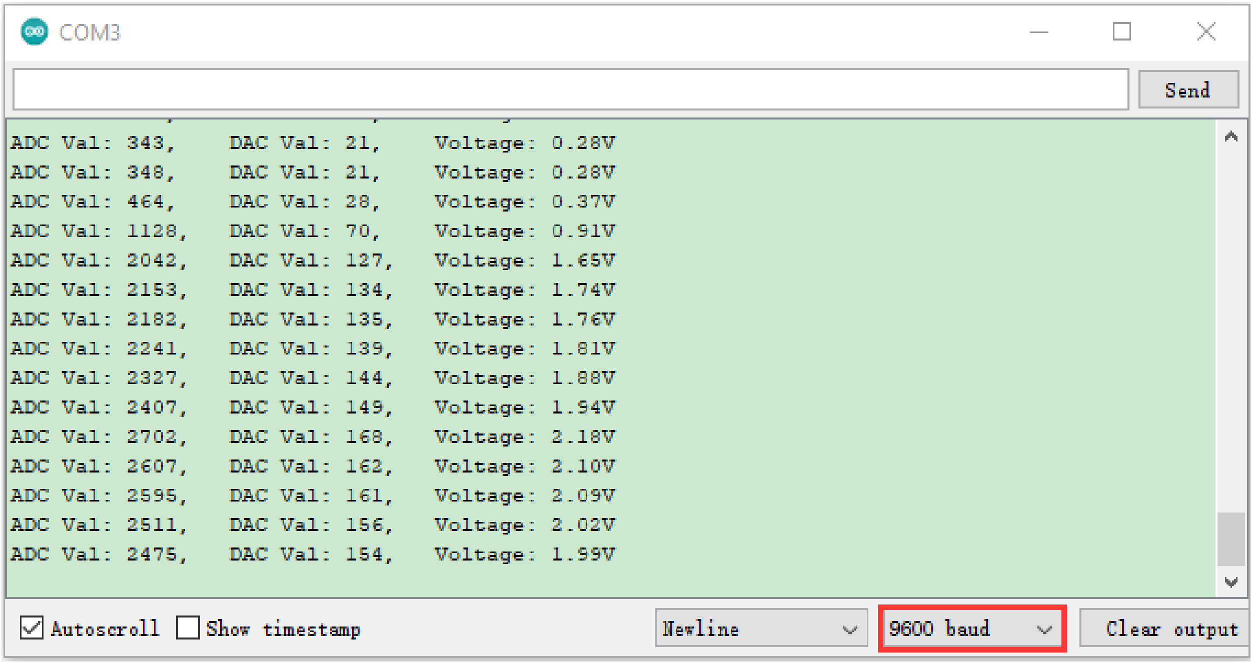
Test Code

```
//**********************************************************************************
/*
 * Filename    : Photoresistance
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN   34  //the pin of the Photoresistance

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//**********************************************************************************
```

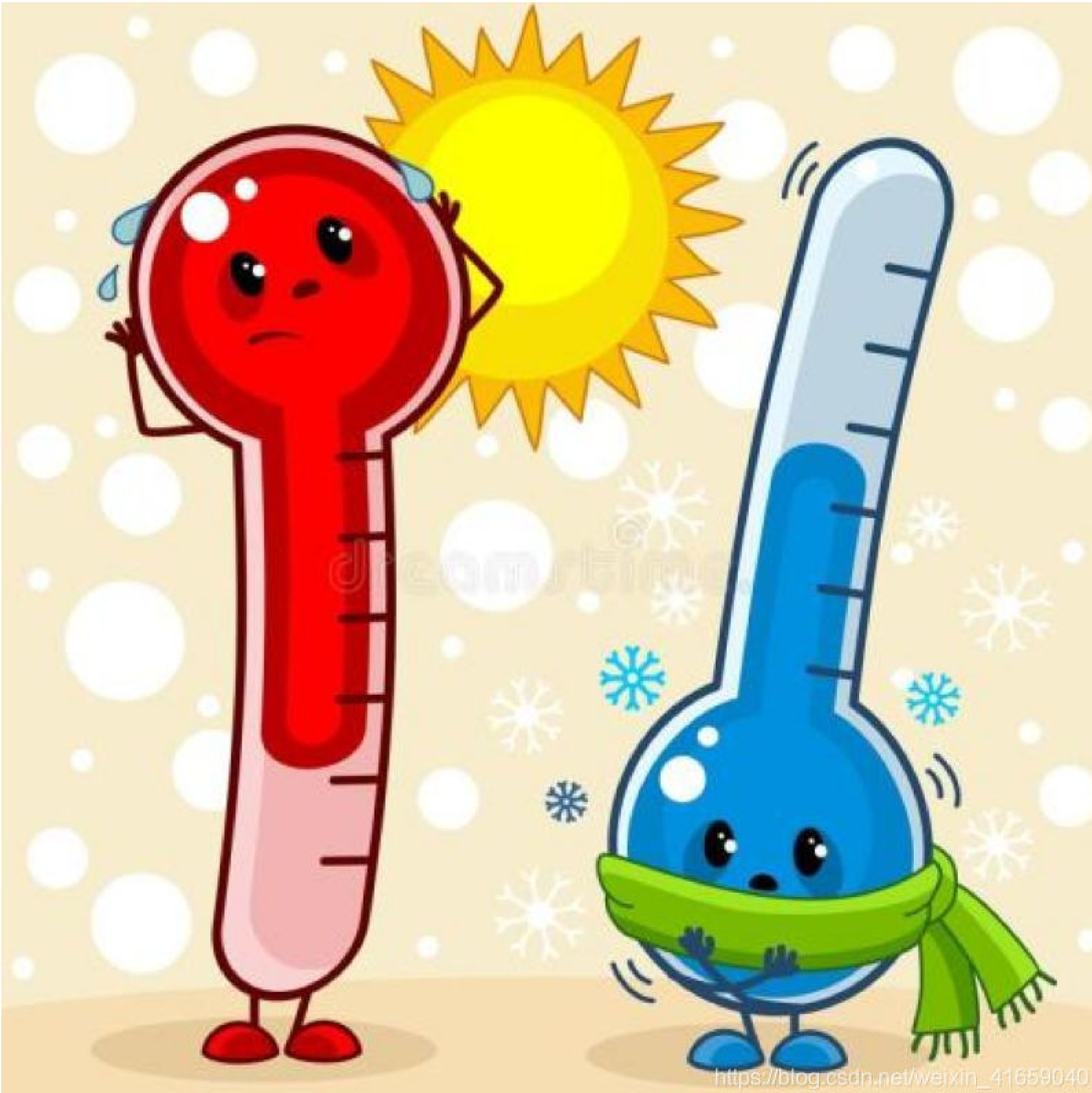
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32.

After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the photoresistor's ADC value, DAC value and voltage value. When the light intensity gets stronger, the analog values will get larger, as shown below:


COM3
Send
ADC Val: 343, DAC Val: 21, Voltage: 0.28V
ADC Val: 348, DAC Val: 21, Voltage: 0.28V
ADC Val: 464, DAC Val: 28, Voltage: 0.37V
ADC Val: 1128, DAC Val: 70, Voltage: 0.91V
ADC Val: 2042, DAC Val: 127, Voltage: 1.65V
ADC Val: 2153, DAC Val: 134, Voltage: 1.74V
ADC Val: 2182, DAC Val: 135, Voltage: 1.76V
ADC Val: 2241, DAC Val: 139, Voltage: 1.81V
ADC Val: 2327, DAC Val: 144, Voltage: 1.88V
ADC Val: 2407, DAC Val: 149, Voltage: 1.94V
ADC Val: 2702, DAC Val: 168, Voltage: 2.18V
ADC Val: 2607, DAC Val: 162, Voltage: 2.10V
ADC Val: 2595, DAC Val: 161, Voltage: 2.09V
ADC Val: 2511, DAC Val: 156, Voltage: 2.02V
ADC Val: 2475, DAC Val: 154, Voltage: 1.99V
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

6.2.24 Project 24: NTC-MF52AT Thermistor

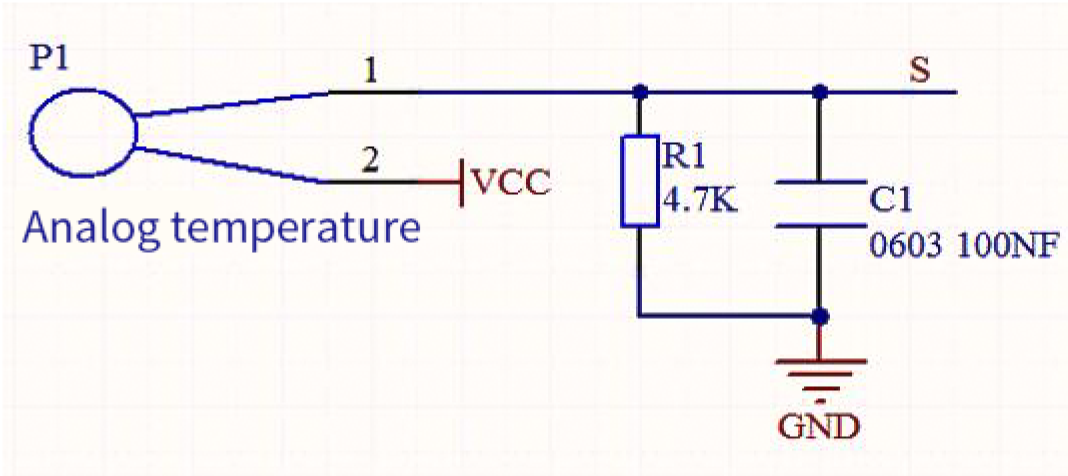


Overview

In the experiment, there is a NTC-MF52AT analog thermistor. We connect its signal terminal to the analog port of the ESP32 mainboard and read the corresponding ADC value, voltage value and thermistor value.

We can use analog values to calculate the temperature of the current environment through specific formulas. Since the temperature calculation formula is more complicated, we only read the corresponding analog value.

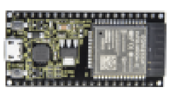
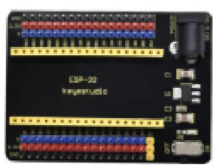
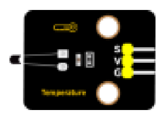
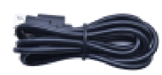
Working Principle



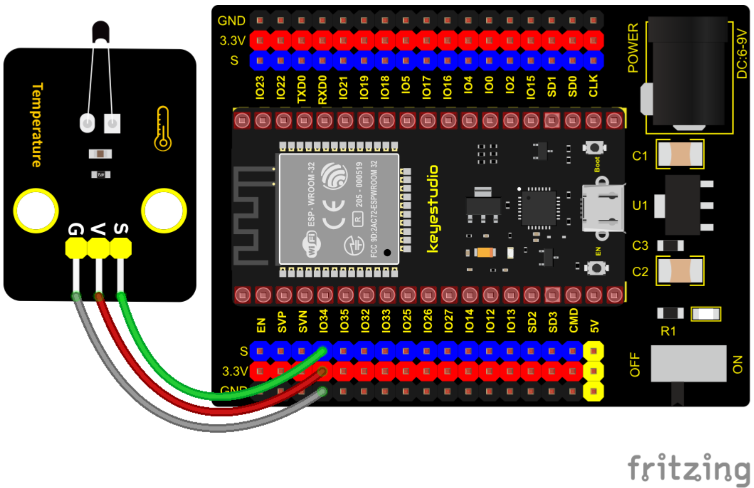
This module mainly uses NTC-MF52AT thermistor element, which can sense the changes of the surrounding environment temperature. Resistance changes with the temperature, causing the voltage of the signal terminal S to change.

This sensor uses the characteristics of NTC-MF52AT thermistor element to convert resistance changes into voltage changes.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio NTC-MF52AT Thermistor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram

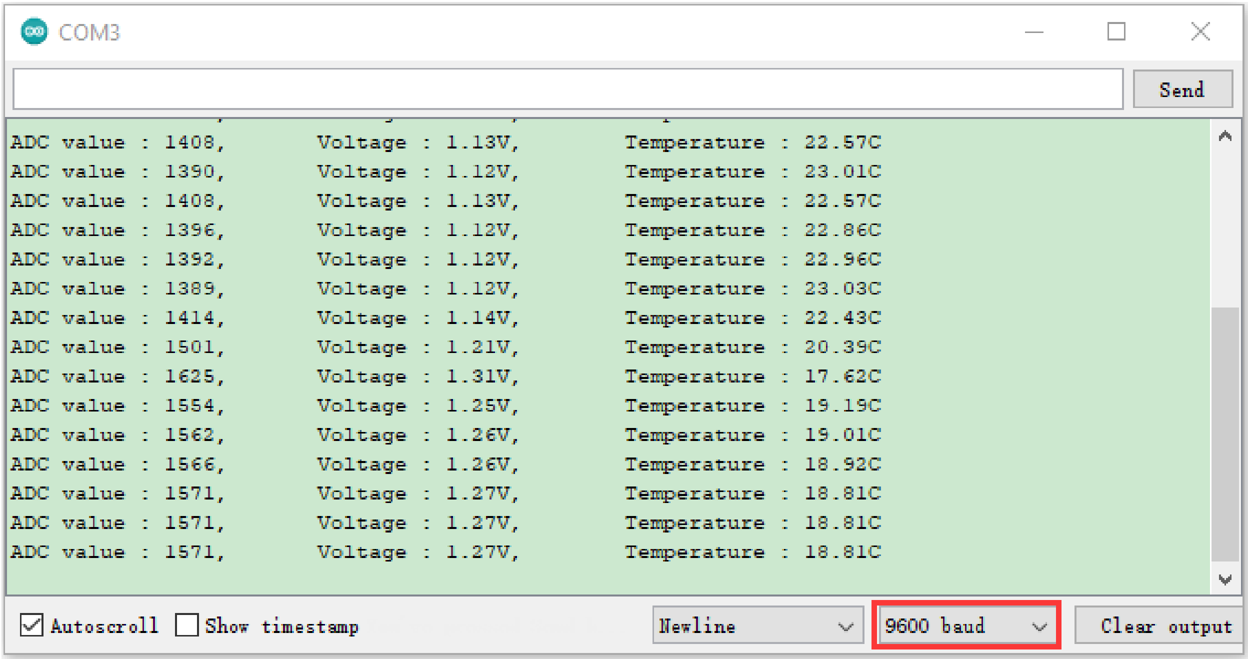


Test Code

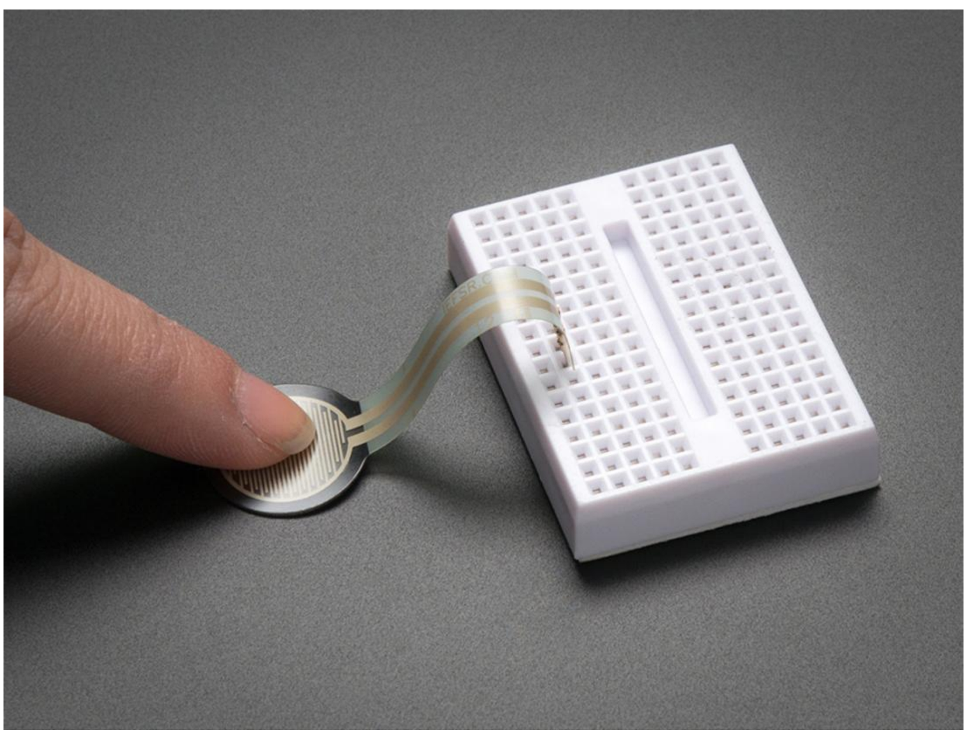
```
//**********************************************************************************
/*
 * Filename    : Temperature sensor
 * Description : Making a thermometer by thermistor.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN   34
void setup() {
  Serial.begin(9600);
}

void loop() {
  int adcValue = analogRead(PIN_ANALOG_IN);                       //read ADC pin
  double voltage = (float)adcValue / 4095.0 * 3.3;                // calculate voltage
  double Rt = (3.3 - voltage) / voltage * 4.7;                     //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);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the thermistor's ADC value, DAC value and voltage value, as shown below:



6.2.25 Project 25: Thin-film Pressure Sensor

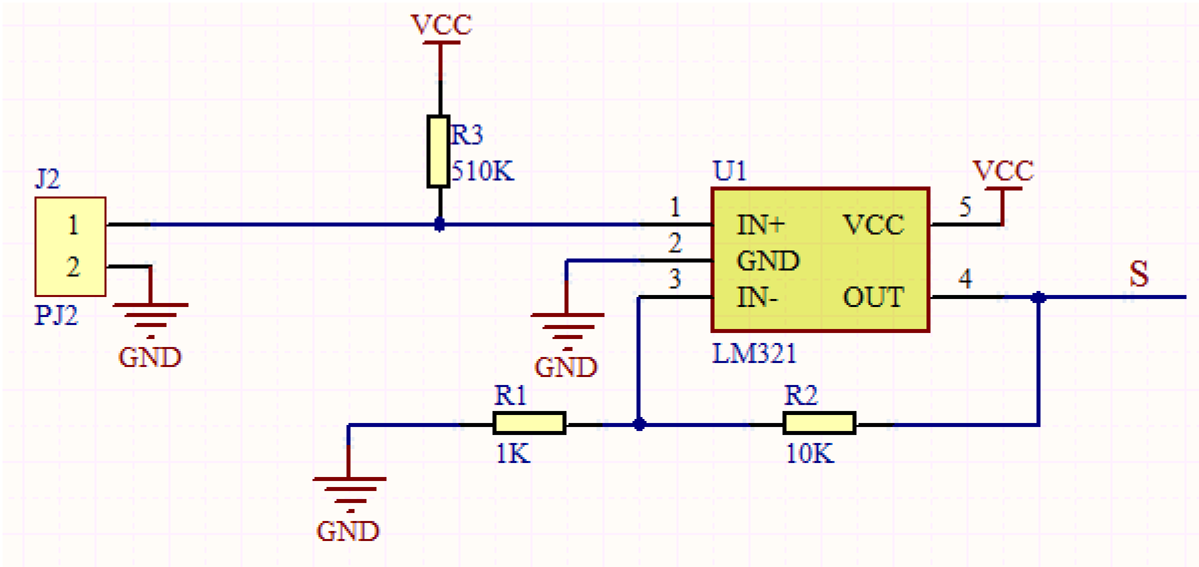


Overview

In this kit, there is a Keyestudio thin-film pressure sensor, which is composed of a new type of nano pressure-sensitive material and a comfortable ultra-thin film substrate, has waterproof and pressure-sensitive functions.

In the experiment, we determine the pressure by collecting the analog signal on the S end of the module. The smaller the ADC value, DAC value and voltage value, the greater the pressure; and the displayed results will shown on the

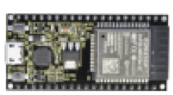
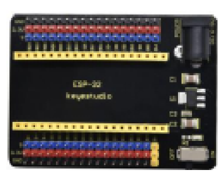
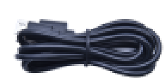
Shell.



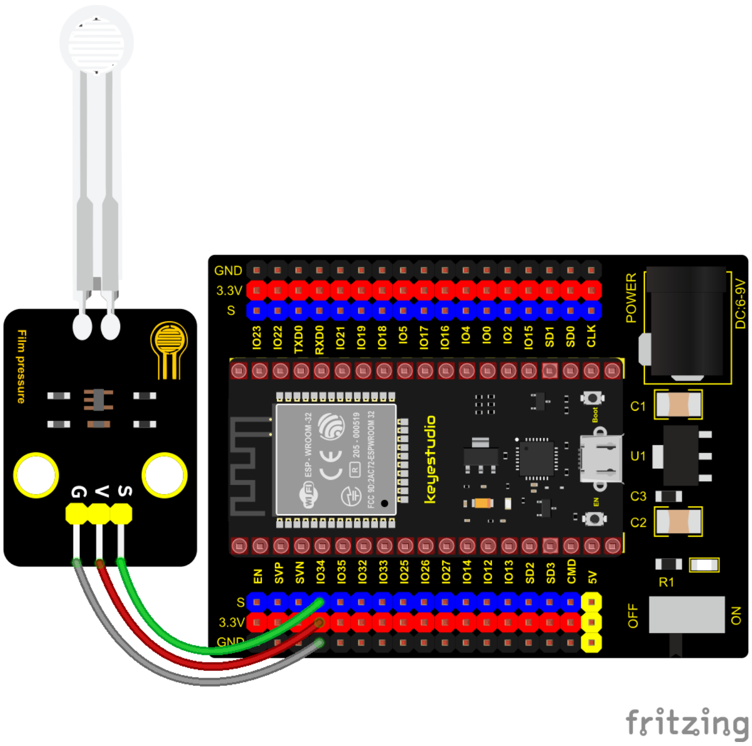
Working Principle

When the sensor is pressed by external forces, the resistance value of sensor will vary. We convert the pressure signals detected by the sensor into the electric signals through a circuit. Then we can obtain the pressure changes by detecting voltage signal changes.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Thin-film Pressure Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



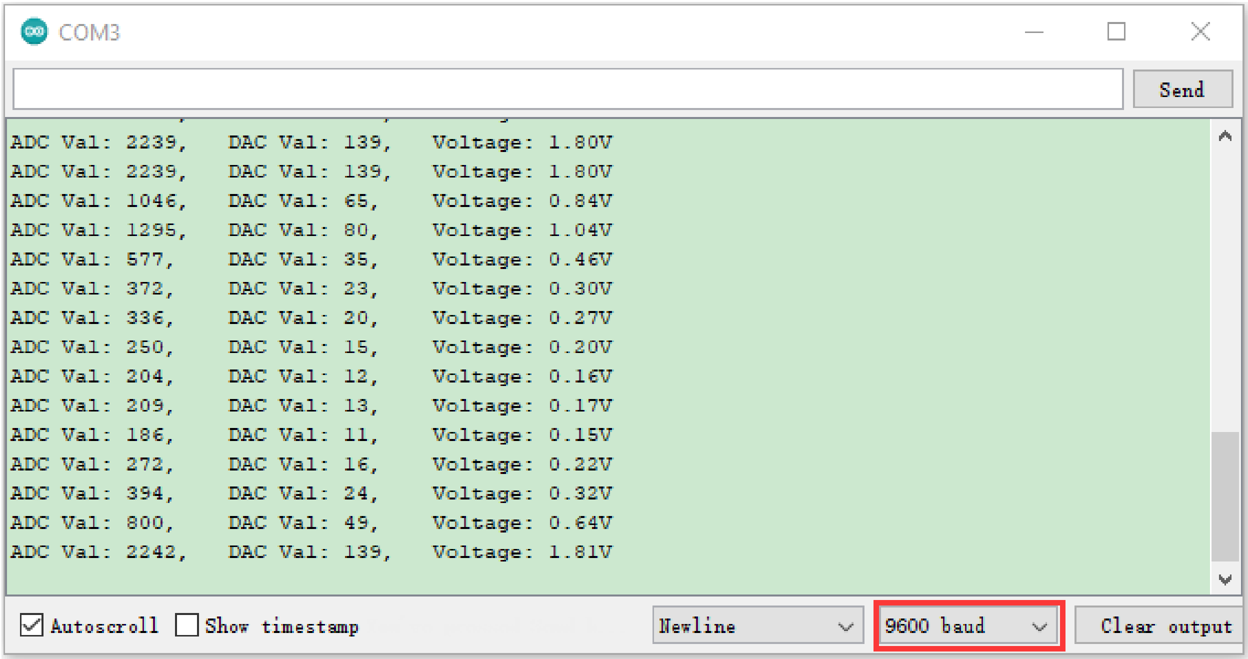
Test Code

```
//**********************************************************************************
/*
 * Filename    : Film pressure sensor
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Film pressure sensor
void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After

uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. The serial monitor will display the thin-film's ADC value, DAC value and voltage value, when the thin-film is pressed by fingers, the analog value will decrease, as shown below;



6.2.26 Project 26: Flame Sensor

Description

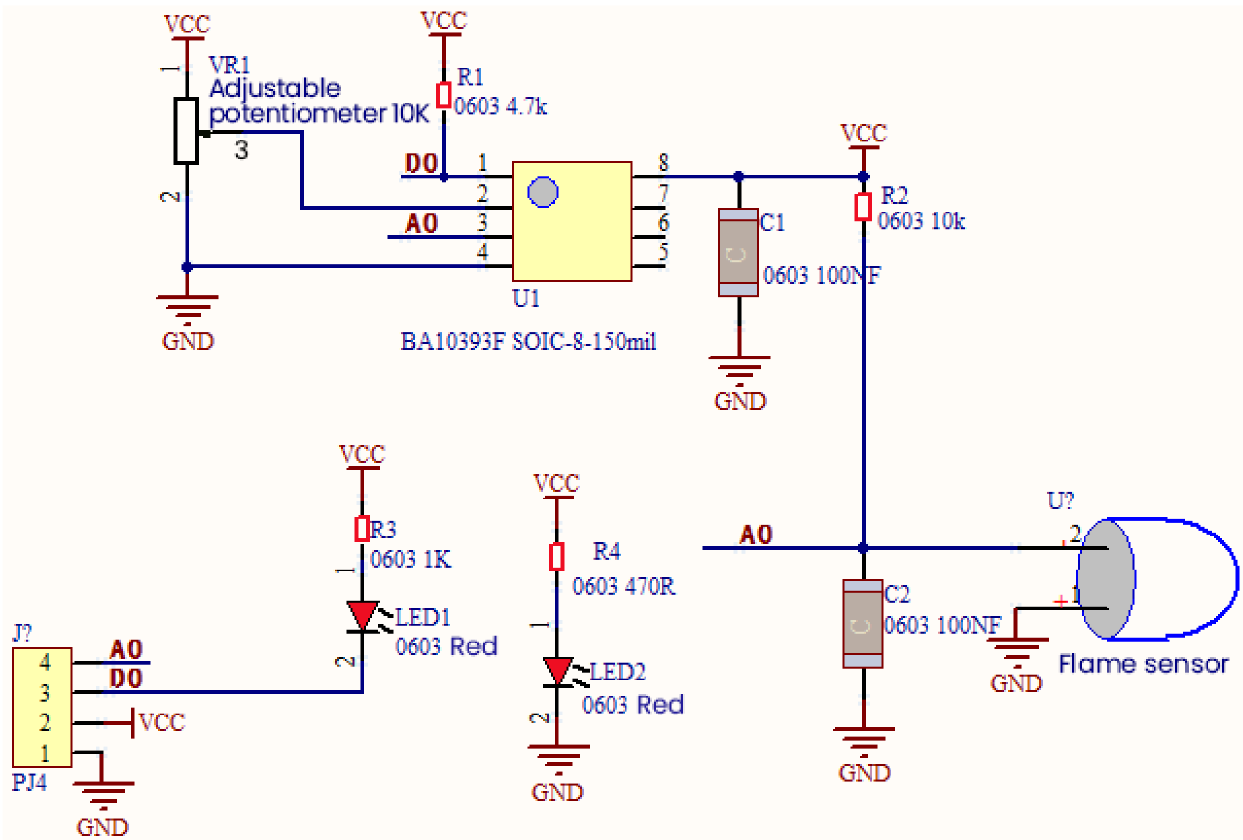
In daily life, it is often seen that a fire broke out without any precaution. It will cause great economic and human loss. So how can we avoid this situation? Right, install a flame sensor and a speaker in those places that easily break out a fire. When the flame sensor detects a fire, the speaker will alarm people quickly to put out the fire.

So in this project, you will learn how to use a flame sensor and an active buzzer module to simulate the fire alarm system.

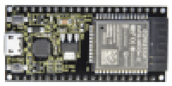
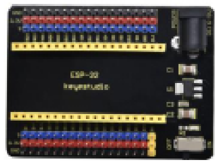
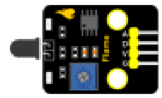
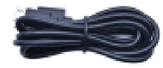
Working Principle

This flame sensor can be used to detect fire or other light sources with wavelength stands at 700nm ~ 1000nm. Its detection angle is about 60°. You can rotate the potentiometer on the sensor to control its sensitivity. Adjust the potentiometer to make the LED at the critical point between on and off state. The sensitivity is the best.

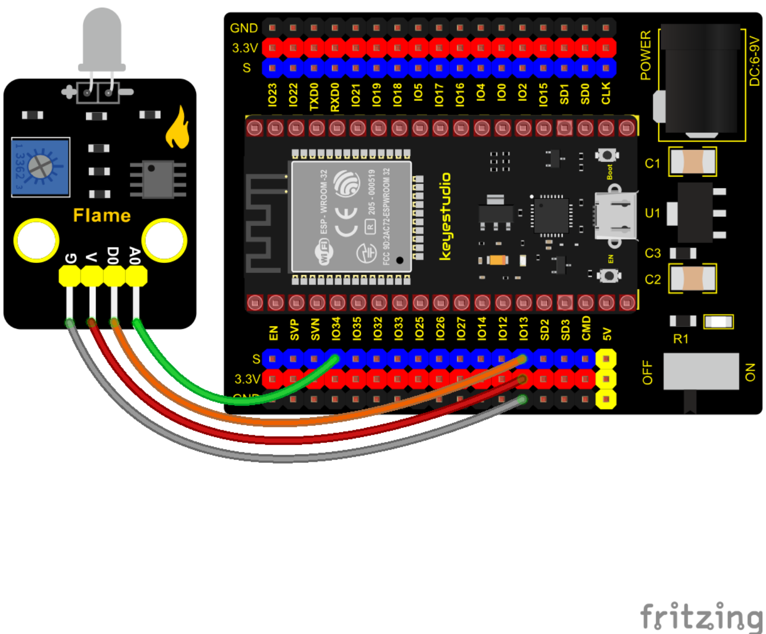
From the below figure, power up. When detecting fire, the digital pin outputs low levels, the red LED2 will light up first, the digital signal terminal D0 outputs a low level, and the red LED1 will light up. The stronger the external infrared light, the smaller the value; the weaker the infrared light, the larger the value.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Flame Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Flame sensor
 * Description : Read the basic usage of DigitalADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
//Flame sensor two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin = 13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;
```

(continues on next page)

(continued from previous page)

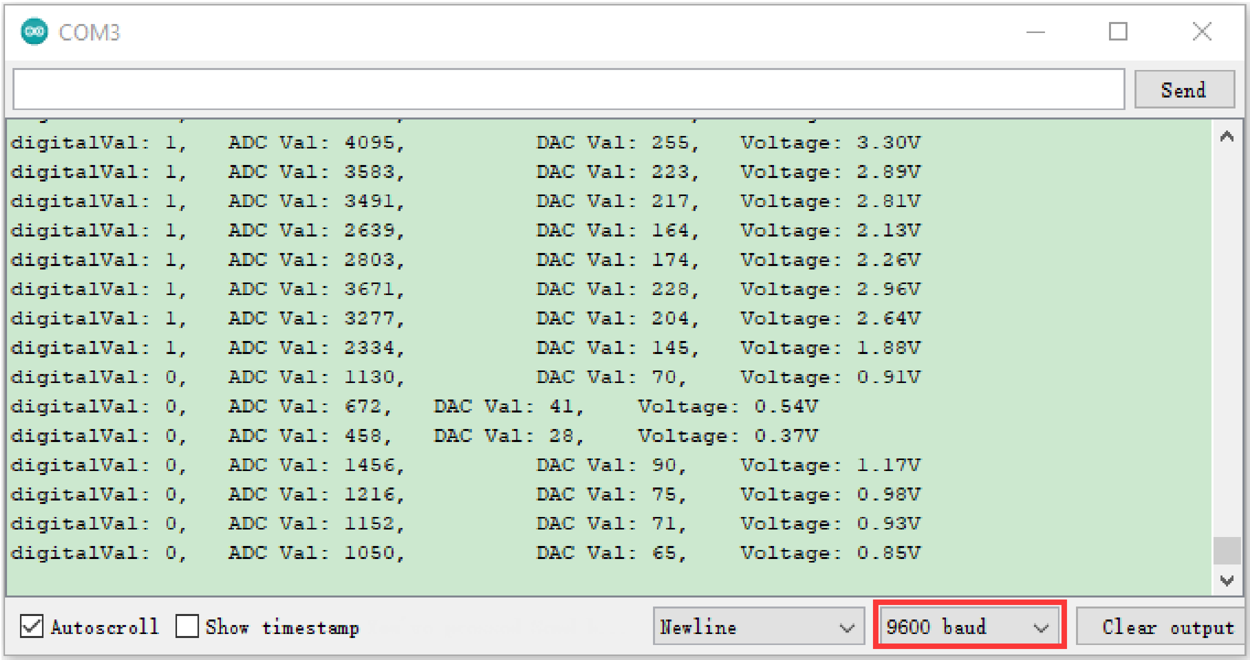
```
void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value,
//the analogRead() function is used to obtain the ADC value.
//the map() function is used to convert the value into an 8-bit precision DAC value.
//The input and output voltage are calculated according to the previous formula, and the
↪information is finally printed out.
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n",
↪digitalVal, adcVal, dacVal, voltage);
  delay(200);
}
//*************************************************************************************
```

Code Explanation

Two pins we use are defined as GPIO13 and GPIO34 according to the wiring-up diagram, and print digital signals and analog signals respectively.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point. The red LED2 on the sensor module is lit, while the red LED1 is not. Open the monitor and set baud rate to 9600. The "Shell" window will display the digital value, ADC value, DAC value and voltage value of the flame sensor. When fire is detected, the LED1 will be on. the digital value will change from 1 to 0, and the analog value will become smaller, as shown below.



6.2.27 Project 27: MQ-2 Gas Sensor

Description

This analog gas sensor - MQ2 is used in gas leakage detecting equipment in consumer electronics and industrial markets.

This sensor is suitable for detecting LPG, I-butane, propane, methane, alcohol, Hydrogen and smoke. It has high sensitivity and quick response.

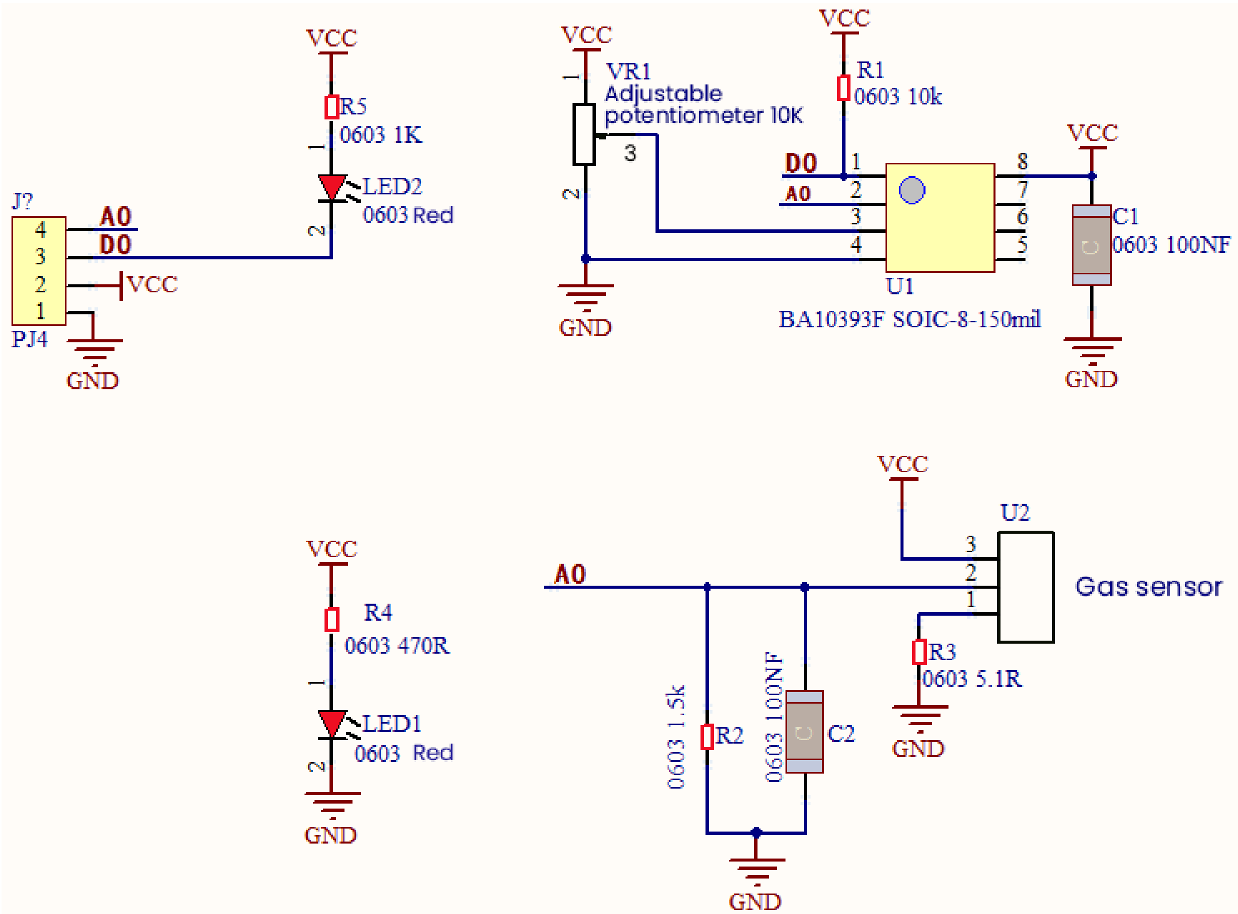
In addition, the sensitivity can be adjusted by rotating the potentiometer.

In the experiment, we read the analog value at the A0 port and the D0 port to determine the content of gas.

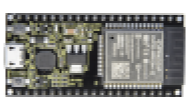
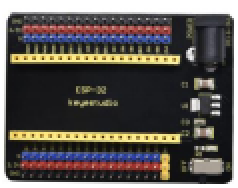
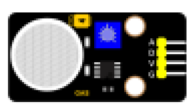
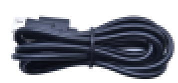
Working Principle

The greater the concentration of smoke, the greater the conductivity, the lower the output resistance, the greater the output analog signal.

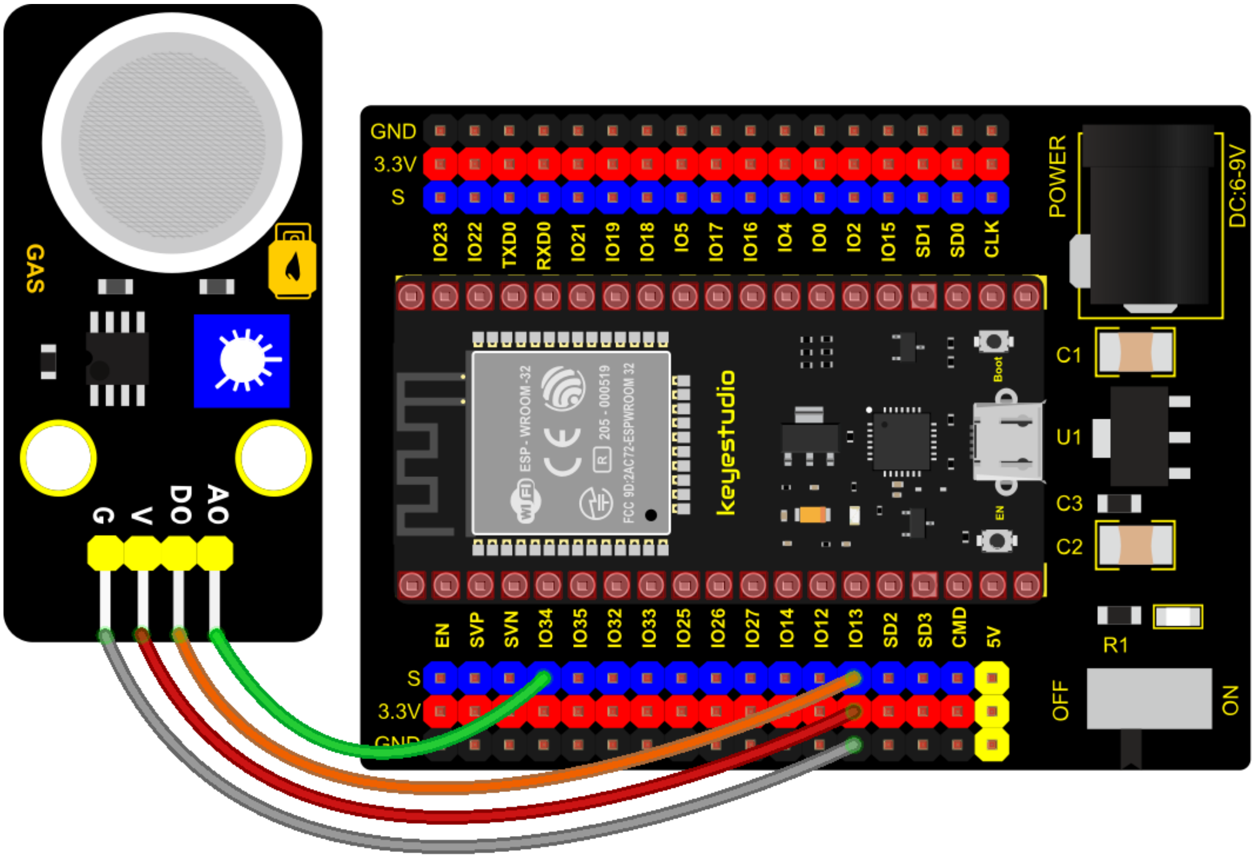
When in use, the A0 terminal reads the analog value of the corresponding gas; the D0 terminal is connected to an LM393 chip (voltage comparator), we can adjust the alarm threshold of the measured gas through the potentiometer, and output the digital value at D0. When the measured gas content exceeds the critical point, the D0 terminal outputs a low level; when the measured gas content does not exceed the critical point, the D0 terminal outputs a high level.



Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Analog Gas Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : MQ2
 * Description : Read the basic usage of Digital, ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
//MQ_2 two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin =  13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value,
//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.
```

(continues on next page)

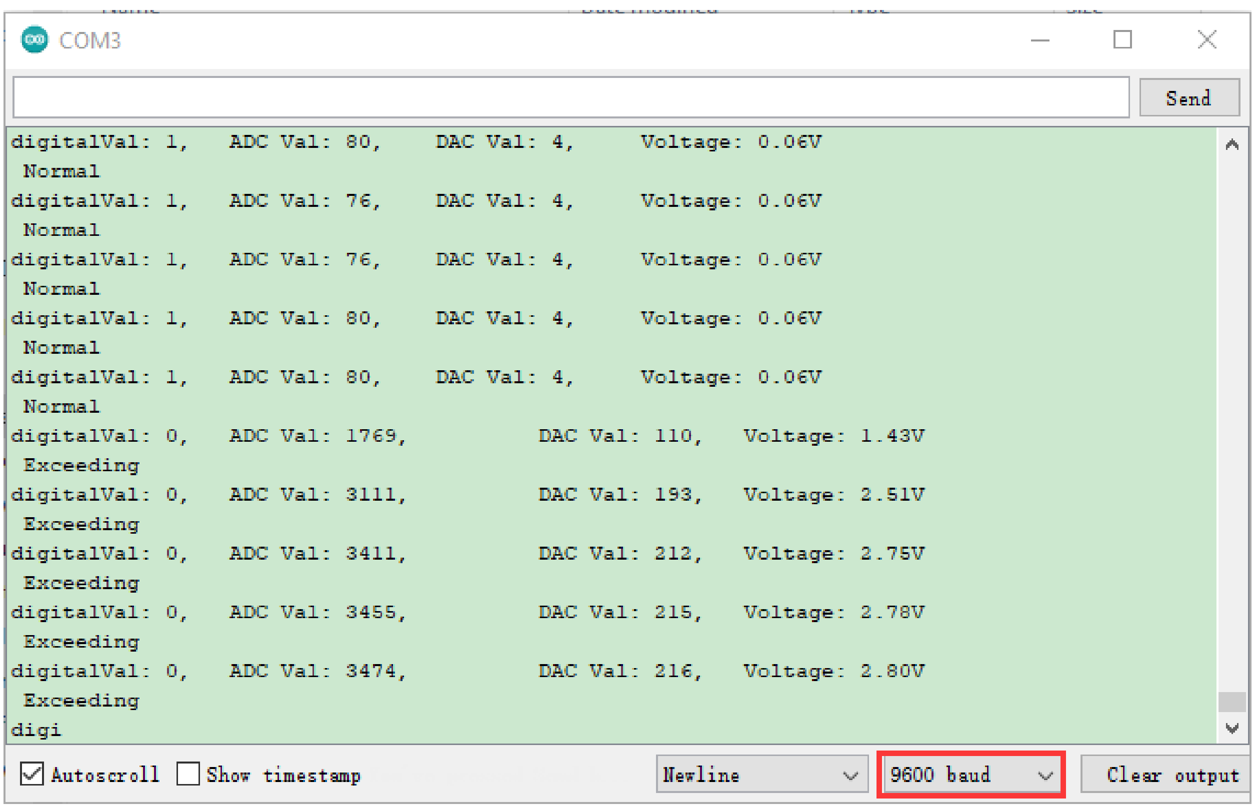
(continued from previous page)

```
//The input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", digitalVal, adcVal, dacVal, voltage);
  if (digitalVal == 1) {
    Serial.println(" Normal");
  }
  else {
    Serial.println(" Exceeding");
  }
  delay(100); //Delay time 100 ms
}
//*********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32.

After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point. Open the monitor , set baud rate to 9600 and display the corresponding data and characters. When the sensor detects the smoke or combustible gas, the red LED lights up and the digital value changes from 1 to 0, the ADC value, DAC value and voltage value increase, as shown below.



6.2.28 Project 28: MQ-3 Alcohol Sensor

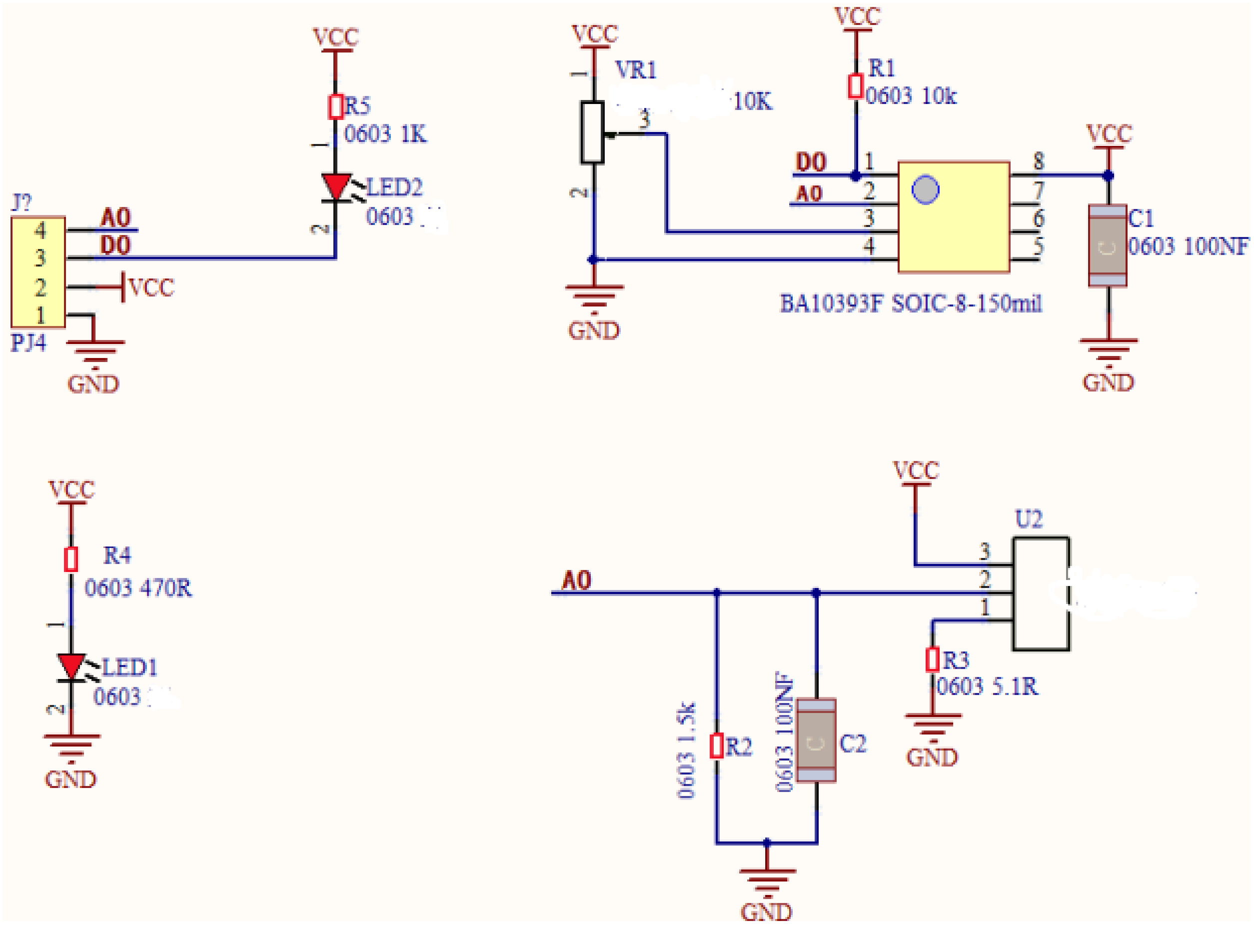
Description

In this kit, there is a MQ-3 alcohol sensor, which uses the gas-sensing material is tin dioxide (SnO2) which has a low conductivity in clean air. When there is alcohol vapor in the environment where the sensor is located, the conductivity of the sensor increases with the increase of the alcohol gas concentration in the air. The change in conductivity can be converted into an output signal corresponding to the gas concentration using a simple circuit.

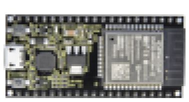
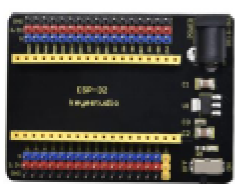
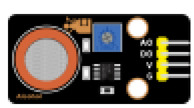
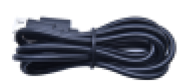
In the experiment, we read the analog value at the A0 end of the sensor and the digital value at the D0 end to judge the content of alcohol vapor in the air and whether they exceed the standard.

Working Principle

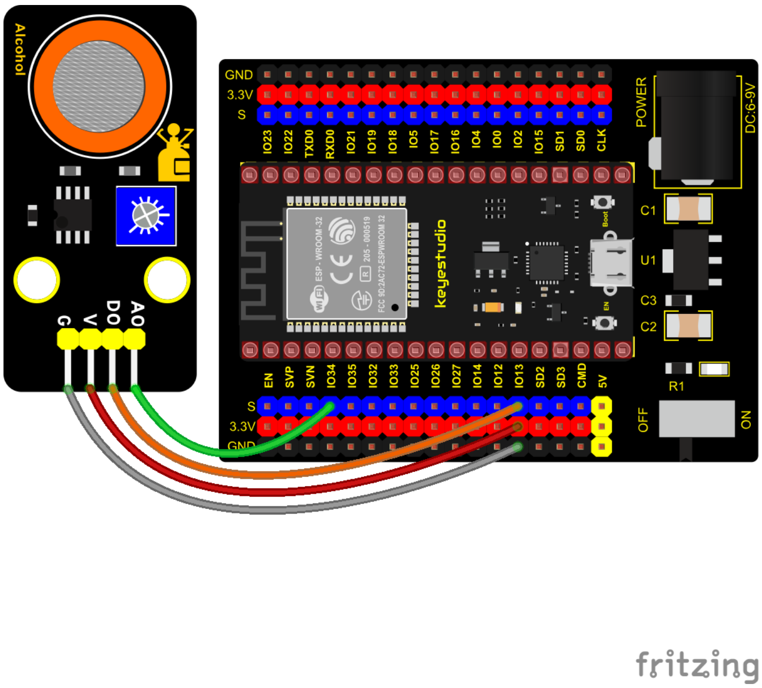
At a certain temperature, the conductivity changes with the composition of the ambient gas. When in use, A0 terminal reads the analog value corresponding to alcohol vapor; D0 terminal is connected to an LM393 chip (comparator), we can adjust and measure the alcohol vapor alarm threshold through the potentiometer, and output the digital value at D0. When the measured alcohol vapor content exceeds the critical point, the D0 terminal outputs a low level; when the measured alcohol vapor content does not exceed the critical point, the D0 terminal outputs a high level.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio Alcohol Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : MQ3
 * Description : Read the basic usage of Digital, ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
//MQ_3 two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin =  13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value,
//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
```

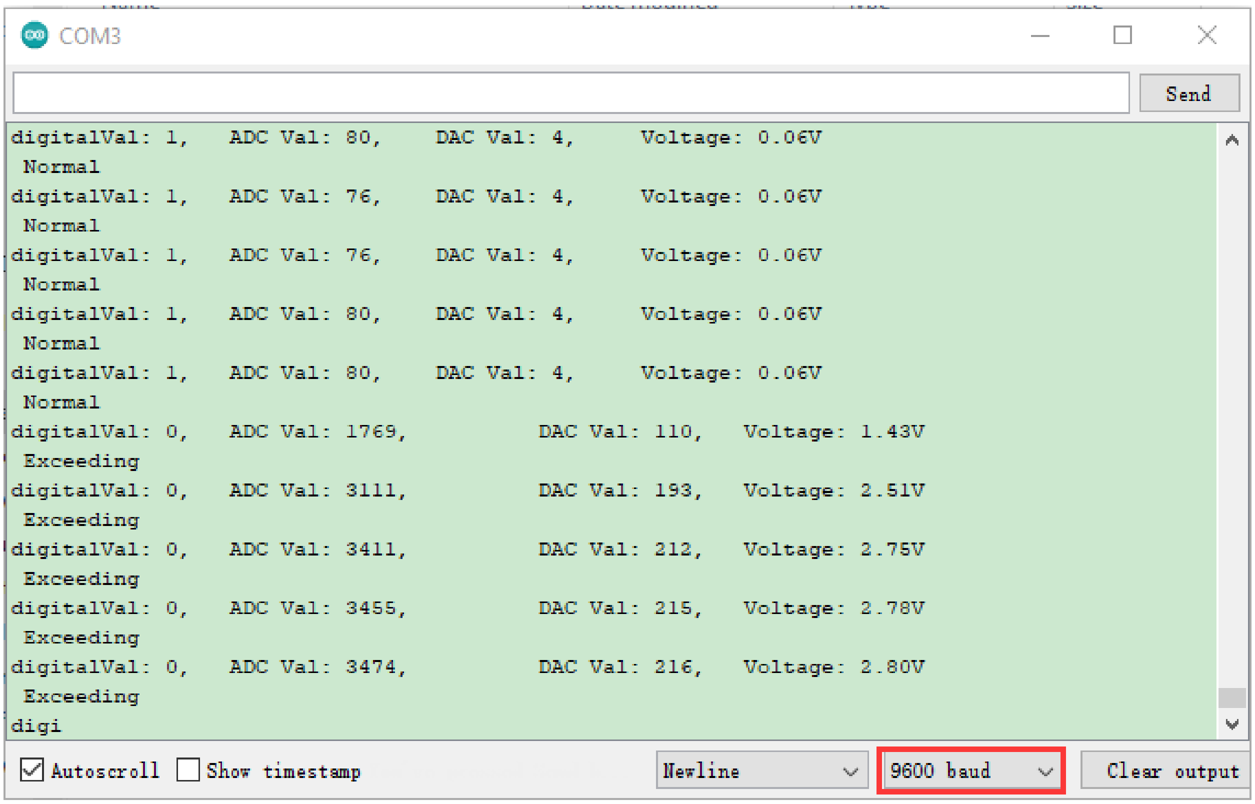
(continues on next page)

(continued from previous page)

```
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", digitalVal, adcVal, dacVal, voltage);
  if (digitalVal == 1) {
    Serial.println("  Normal");
  }
  else {
    Serial.println("  Exceeding");
  }
  delay(100); //Delay time 100 ms
}
//*********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the yellow and green LED bright and not bright critical point. Open the monitor, set baud rate to 9600 and display the corresponding data and characters. When the sensor detects the alcohol gas, the yellow and green LED lights up and the digital value changes from 1 to 0, the ADC value, DAC value and voltage value decrease, as shown below.



6.2.29 Project 29: Five-key AD Button Module

Description

When we talked about analog and digital sensors earlier, we talked about the single-channel key module. When we press the key, it outputs a low level, and when we release the key, it outputs a high level. We can only read these two digital signals. In fact, the key module ADC acquisition can also be performed. In this kit, a DIY electronic building block five-way AD button module is included.

We can judge which key is pressed through the analog value. In the experiment, we print out the key press information in the shell.

Working Principle

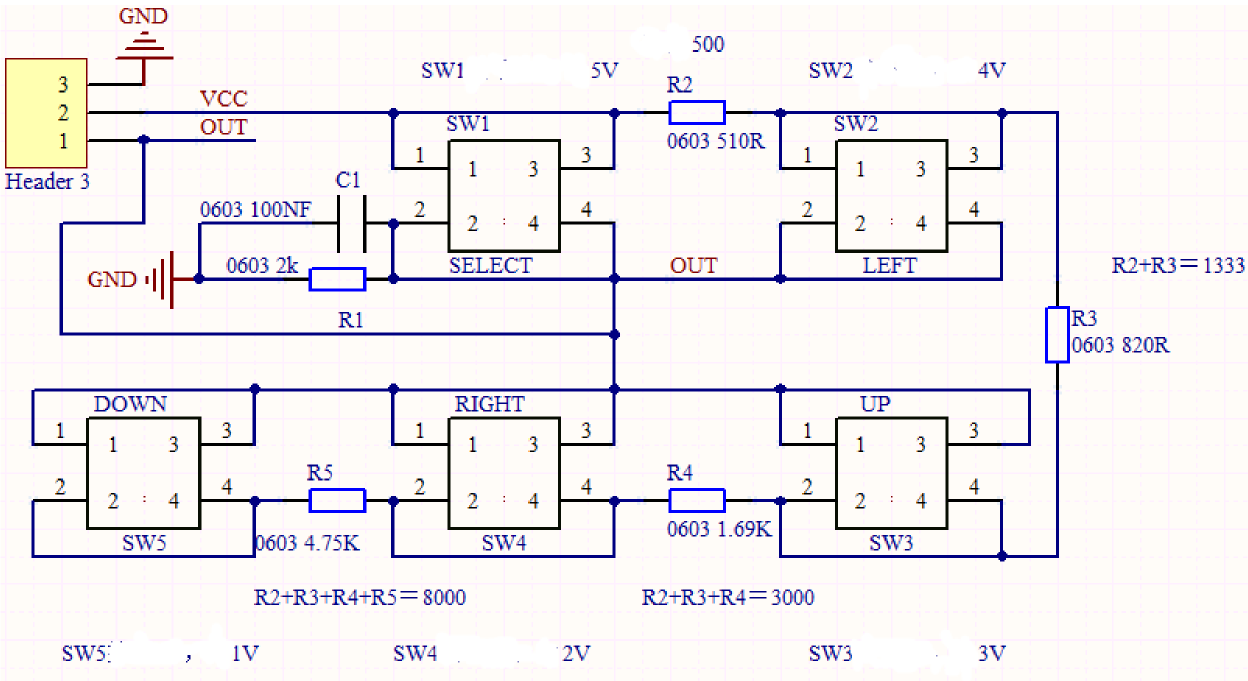
Let's look at the schematic diagram, when we do not press the key, the OUT of S output to the signal end is pulled down by R1. At this time, we read the low level 0V. When we press the key SW1, the OUT of the output to the signal end S is directly connected to the VCC. At this time, we read the high level 3.3V(the figure is marked as a 12-bit ADC(0~4095) and VCC is 5V. The principle is the same. Here we have VCC of 3.3V and ADC mapped to 12 bits), which is an analog value of 4095.

Next,when we press the key SW2, the OUT terminal voltage of the signal we read is the voltage between R2 and R1, namely VCC*R1/(R2+R1), which is about 2.64V, and the analog value is about 3276.

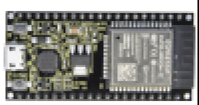
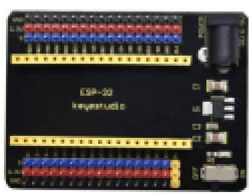
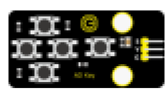
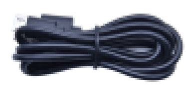
When we press the key SW3, the OUT terminal voltage of the signal we read is the voltage between R2+R3 and R1, namely VCC*R1/(R3+R2+R1), which is about 1.99V, and the analog value is about 2469.

When we press the key SW4, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4 and R1, namely VCC*R1/(R4+R3+R2+R1), about 1.31V, and the analog value is about 1626.

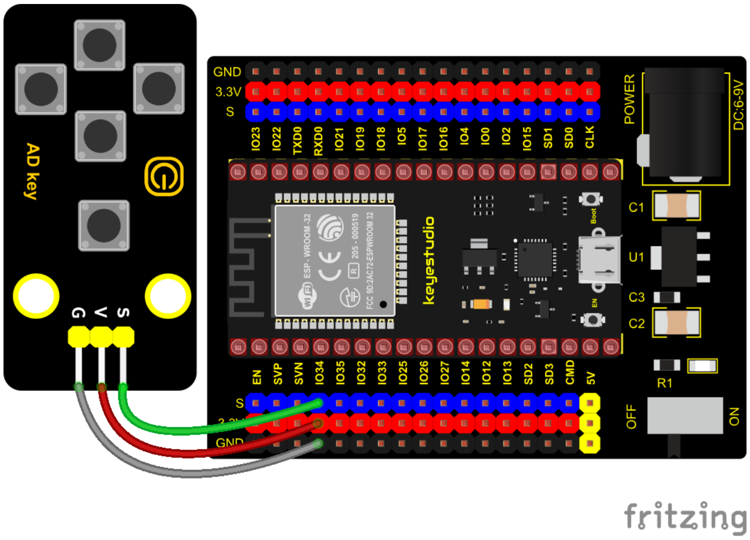
Similarly, when we press the key SW5, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4+R5 and R1, namely VCC*R1/(R5+R4+R3+R2+R1), which is about 0.68V, and the analog value is about 844. R4+R3+R2+R1), which is about 0.68V, and the analog value is about 844.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 5-Channel AD Button Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Five AD Keys
 * Description : Read the value of Five AD Keys
 * Auther      : http//www.keyestudio.com
*/
int val = 0;
int ADkey = 34; //Define five AD keys connected to GPIO36
void setup() {
  Serial.begin(9600); //Set baud rate to 9600
}

void loop() {
  val = analogRead(ADkey);  //Read the simulated value of the AD key and assign it to the variable val
  Serial.print(val);  //A newline prints the variable val
  if (val <= 500) { //Val is less than or equal to 500 when no button is pressed
    Serial.println("   no key  is pressed");
  } else if (val <= 1000) { //When key 5 is pressed,val is between 500 and 1000
    Serial.println("   SW5 is pressed");
  } else if (val <= 2000) { //When pressed,val is between 1000 and 2000
    Serial.println("   SW4 is pressed");
  } else if (val <= 3000) { //When pressed,val is between 2000 and 3000
    Serial.println("   SW3 is pressed");
  } else if (val <= 4000) { //When key 2 is pressed,val is between 3000 and 4000
    Serial.println("   SW2 is pressed");
  } else {  //When key 1 is pressed,val is greater than 4000
    Serial.println("   SW1 is pressed");
  }
}
//**********************************************************************************
```

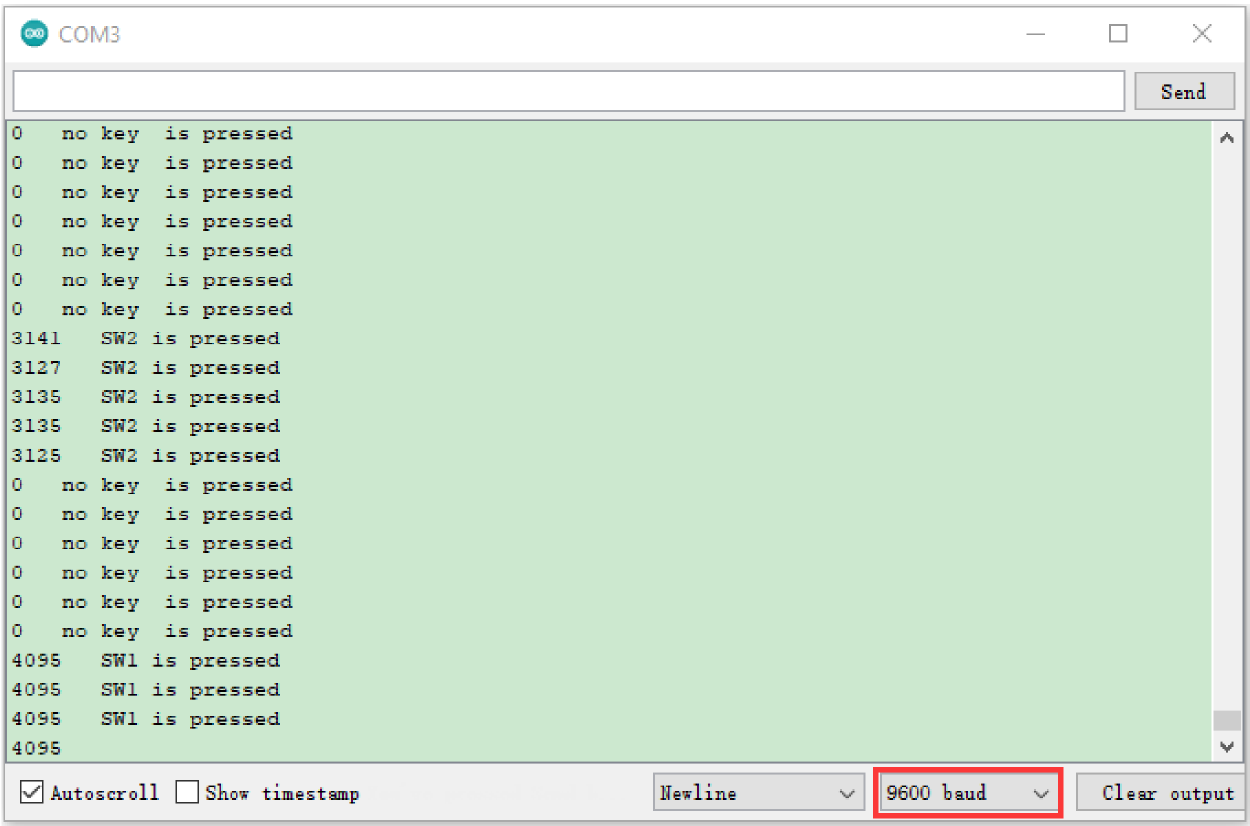
Code Explanation

We assign the read analog value to the variable val, and the serial monitor displays the value of val, (we set to 9600).

When the analog value is in the range of 500 and 1000, the button SW5 is pressed; when the analog value is in the 1000 and 2000, the button SW4 is pressed; when the analog value is between 2000 and 3000, the button SW3 is pressed; when the analog value is between 3000 and 4000, the button SW2 is pressed. When the analog value is above 4000, we judge that the button SW1 is pressed.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600; when the button is pressed, the serial monitor prints out the corresponding information, as shown in the figure below.



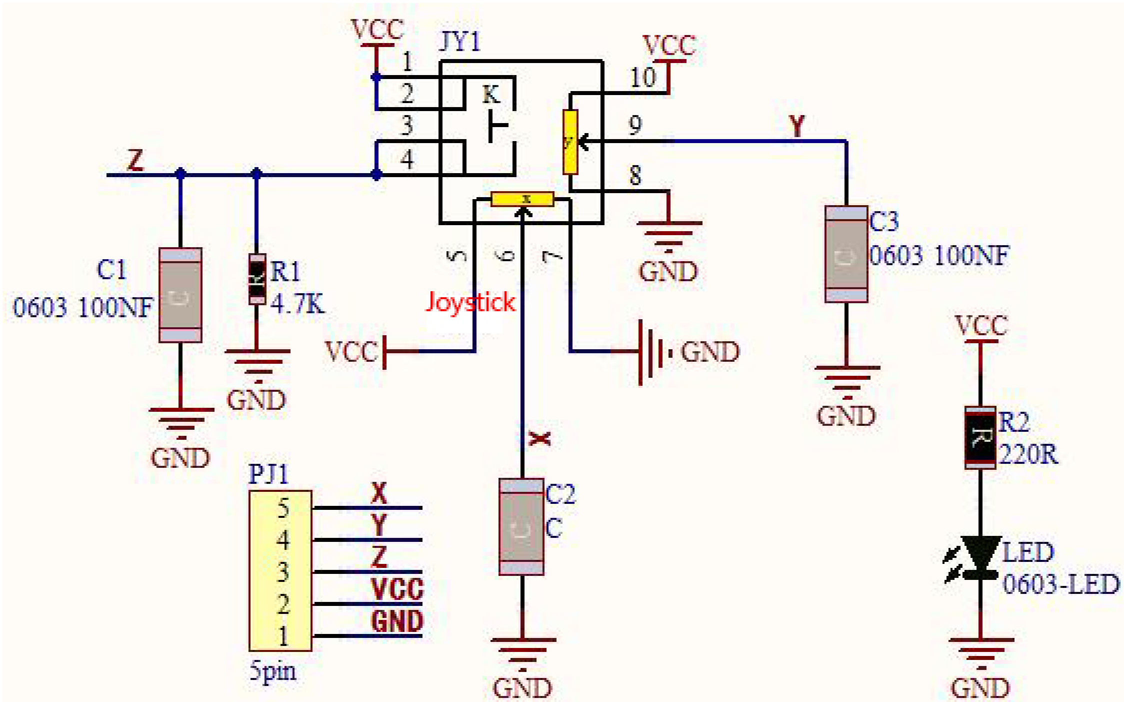
6.2.30 Project 30: Joystick Module

Overview

Game handle controllers are ubiquitous. There is a joystick module in this kit, which mainly uses PS2 joysticks. When controlling it, we need to connect the X and Y ports of the module to the analog port of the single-chip microcomputer, port B to the digital port of the single-chip microcomputer, VCC to the power output port(3.3-5V), and GND to the GND of the MCU. We can read the high and low levels of two analog values and one digital port) to determine the working status of the joystick on the module.

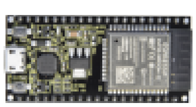
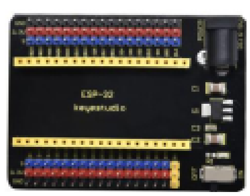
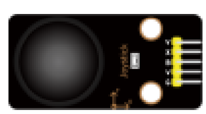
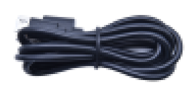
In the experiment, two analog values(x axis and y axis) will be shown on the Shell.

Working Principle

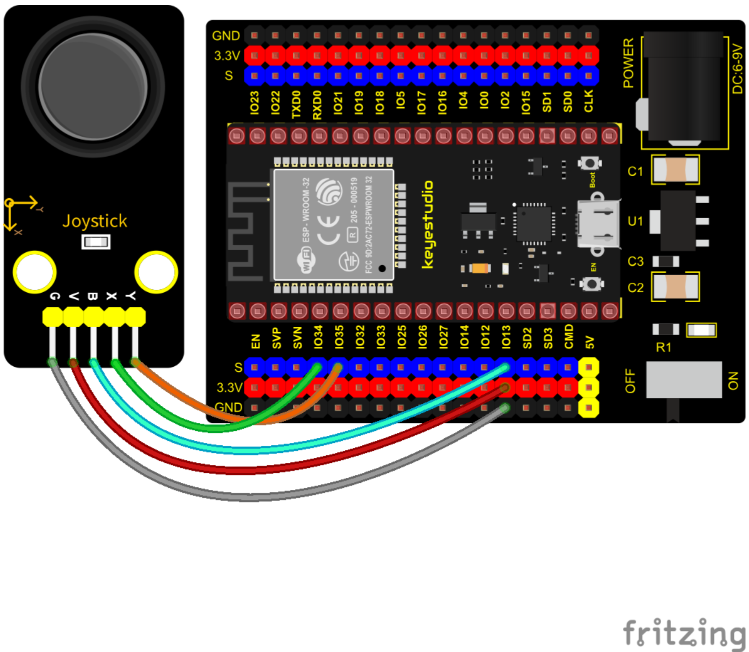


In fact, its working principle is very simple. Its inside structure is equivalent to two adjustable potentiometers and a button. When this button is not pressed and the module is pulled down by R1, low levels will be output ; on the contrary, when the button is pressed, VCC will be connected (high levels). When we move the joystick, the internal potentiometer will adjust to output different voltages, and we can read the analog value.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Joystick Module*1	5P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Joystick
 * Description : Read data from Rocker.
 * Auther      : http//www.keyestudio.com
*/
int xyzPins[] = {34, 35, 13};   //x,y,z pins
void setup() {
  Serial.begin(9600);
  pinMode(xyzPins[0], INPUT); //x axis.
  pinMode(xyzPins[1], INPUT); //y axis.
  pinMode(xyzPins[2], INPUT_PULLUP);   //z axis is a button.
}

// 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.
void loop() {
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.println("X,Y,Z: " + String(xVal) + ", " +  String(yVal) + ", " + String(zVal));
  delay(500);
}
//**********************************************************************************
```

Code Explanation

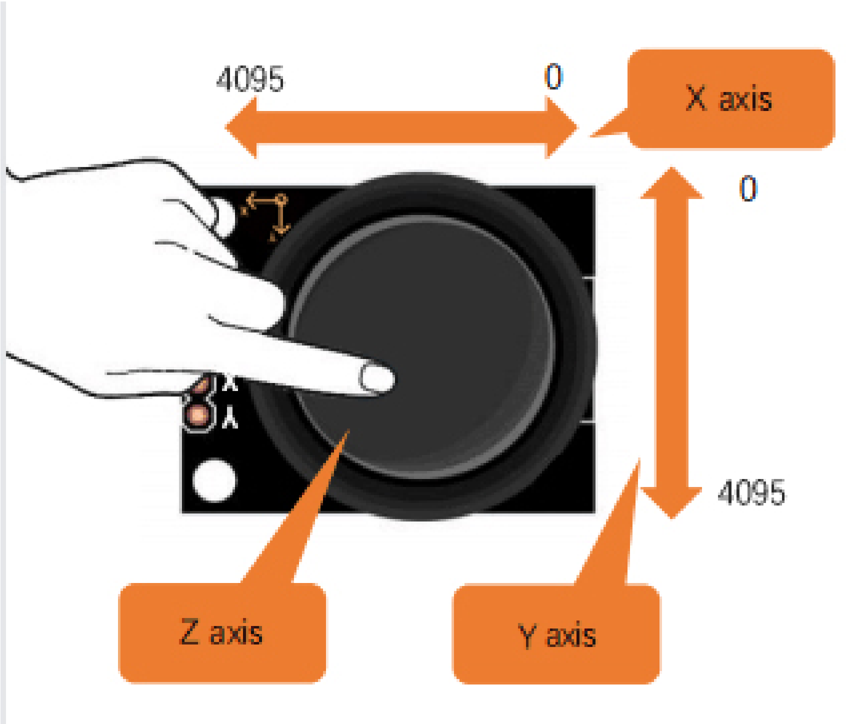
In the experiment, according to the wiring diagram, the x pin is set to GPIO34, the y pin is set to GPIO35 and the pin of the joystick is set to GPIO13.

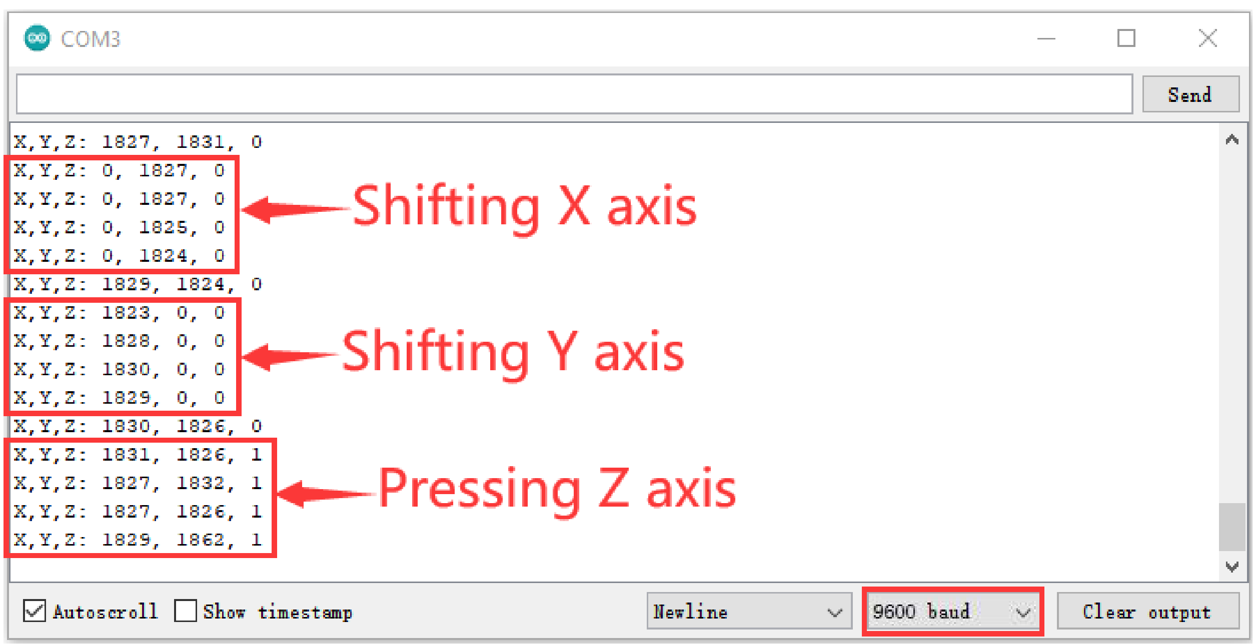
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After

uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600;

The serial monitor will show the corresponding value. Moving the joystick or pressing it will change the analog and digital values in the serial monitor .





6.2.31 Project 31: Relay Module

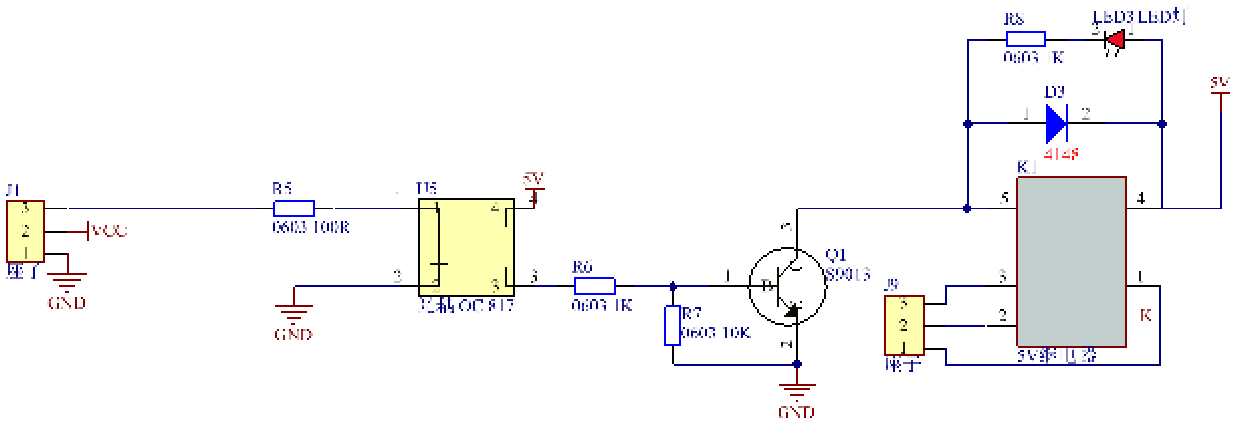
Overview

In our daily life, we usually use communication to drive electrical equipment, and sometimes we use switches to control electrical equipment. If the switch is connected directly to the ac circuit, leakage occurs and people are in danger. Therefore, from the perspective of safety, we specially designed this relay module with NO(normally open) end and NC(normally closed) end.

Working Principle

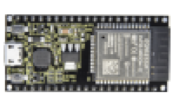
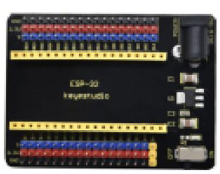
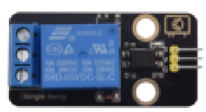
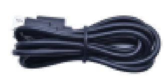
Relay is compatible with a variety of microcontroller control board, such as Arduino series microcontroller, which is a small current to control the operation of large current "automatic switch".

Input Voltage3.3V-5V

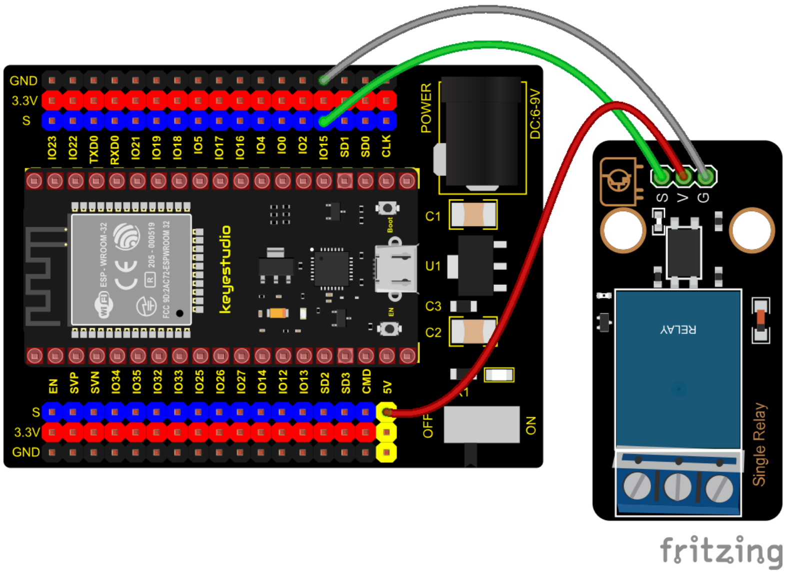


It can let the MCU control board drive 3A load, such as an LED lamp belt, a DC motor, a micro water pump and a solenoid valve pluggable interface design, which is easy to use.

Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Relay Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



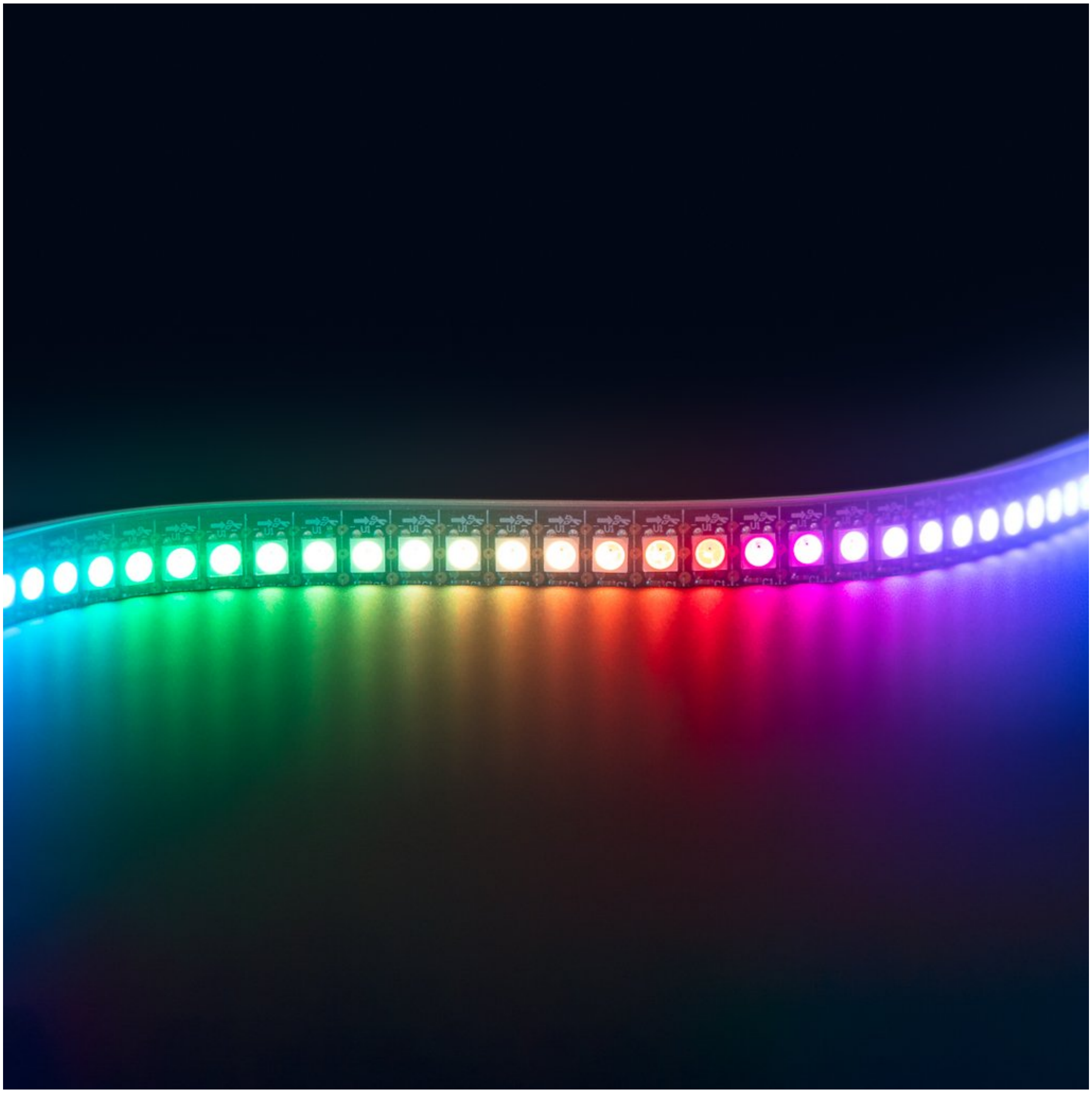
Test Code

```
//*************************************************************************************
/*
 * Filename    : Relay
 * Description : Relay turn on and off.
 * Auther      : http//www.keyestudio.com
*/
#define  Relay  15 // defines digital 15
void setup()
{
pinMode(Relay, OUTPUT); // sets "Relay" to "output"
}
void loop()
{
digitalWrite(Relay, HIGH); // turns on the relay
delay(1000); //delays 1 seconds
digitalWrite(Relay, LOW); // turns off the relay
delay(1000); // delays 1 seconds
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The relay will cycle on and off, on for 1 second, off for 1 second. At the same time, you can hear the sound of the relay on and off as well as see the change of the indicator light on the relay .

6.2.32 Project 32: SK6812 RGB Module



Overview

In previous lessons, we learned about the plug-in RGB module and used PWM signals to color the three pins of the module.

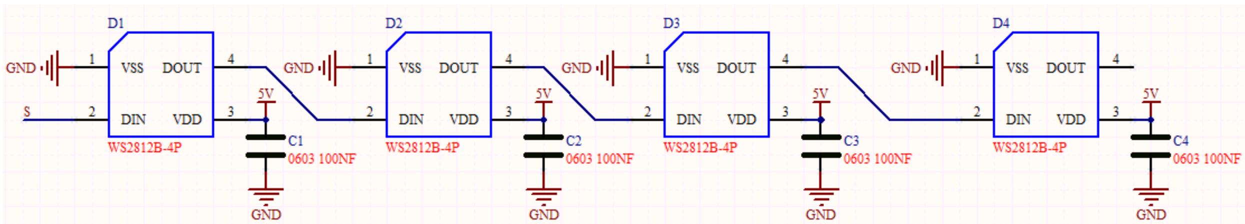
There is a Keyestudio 6812 RGB module whose the driving principle is different from the plug-in RGB module. It can only control with one pin. This is a set. It is an intelligent externally controlled LED light source with the control circuit and the light-emitting circuit. Each LED element is the same as a 5050 LED lamp bead, and each component is a pixel. There are four lamp beads on the module, which indicates four pixels.

In the experiment, we make different lights show different colors.

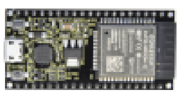
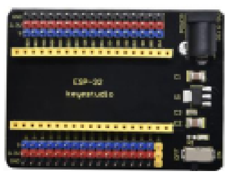
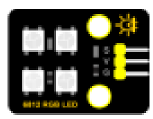
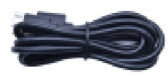
Working Principle

From the schematic diagram, we can see that these four pixel lighting beads are all connected in series. In fact, no matter how many they are, we can use a pin to control a light and let it display any color. The pixel point contains a data latch signal shaping amplifier drive circuit, a high-precision internal oscillator and a 12V high-voltage programmable constant current control part, which effectively ensures the color of the pixel point light is highly consistent.

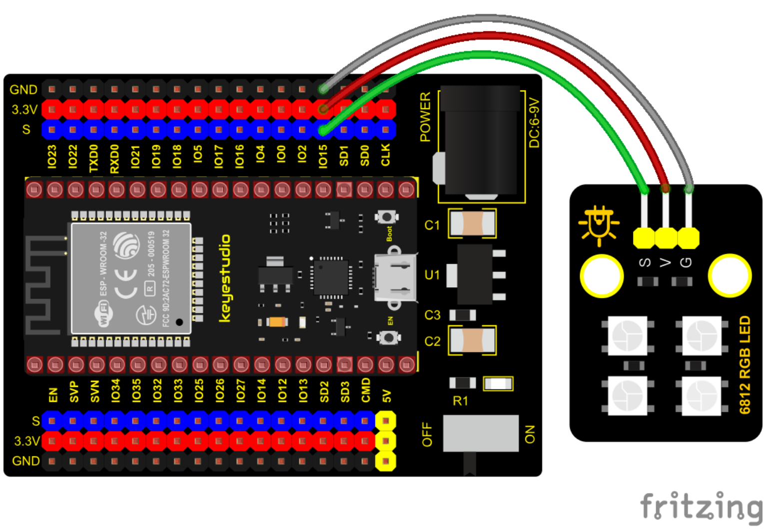
The data protocol adopts a single-wire zero-code communication method. After the pixel is powered up and reset, the S terminal receives the data transmitted from the controller. The first 24bit data sent is extracted by the first pixel and sent to the data latch of the pixel.



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 6812 RGB Module*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : sk6812 RGB LED
 * Description : turn on sk6812 RGB LED
 * Auther      : http//www.keyestudio.com
*/
#include <Adafruit_NeoPixel.h>

#define PIN 15

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);

// IMPORTANT: To reduce NeoPixel burnout risk, add 1000 uF capacitor across
// pixel power leads, add 300 - 500 Ohm resistor on first pixel's data input
// and minimize distance between Arduino and first pixel.  Avoid connecting
// on a live circuit...if you must, connect GND first.

void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  // Send a theater pixel chase in...
  theaterChase(strip.Color(127, 127, 127), 50); // White
  theaterChase(strip.Color(127,   0,   0), 50); // Red
  theaterChase(strip.Color(  0,   0, 127), 50); // Blue

  rainbow(20);
  rainbowCycle(20);
  theaterChaseRainbow(50);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, c);
      strip.show();
      delay(wait);
  }
}
```

(continues on next page)

(continued from previous page)

```
void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Theatre-style crawling lights.
void theaterChase(uint32_t c, uint8_t wait) {
  for (int j=0; j<10; j++) {  //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}

//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait) {
  for (int j=0; j < 256; j++) {     // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
        for (int i=0; i < strip.numPixels(); i=i+3) {
          strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel on
        }
        strip.show();

```

(continues on next page)

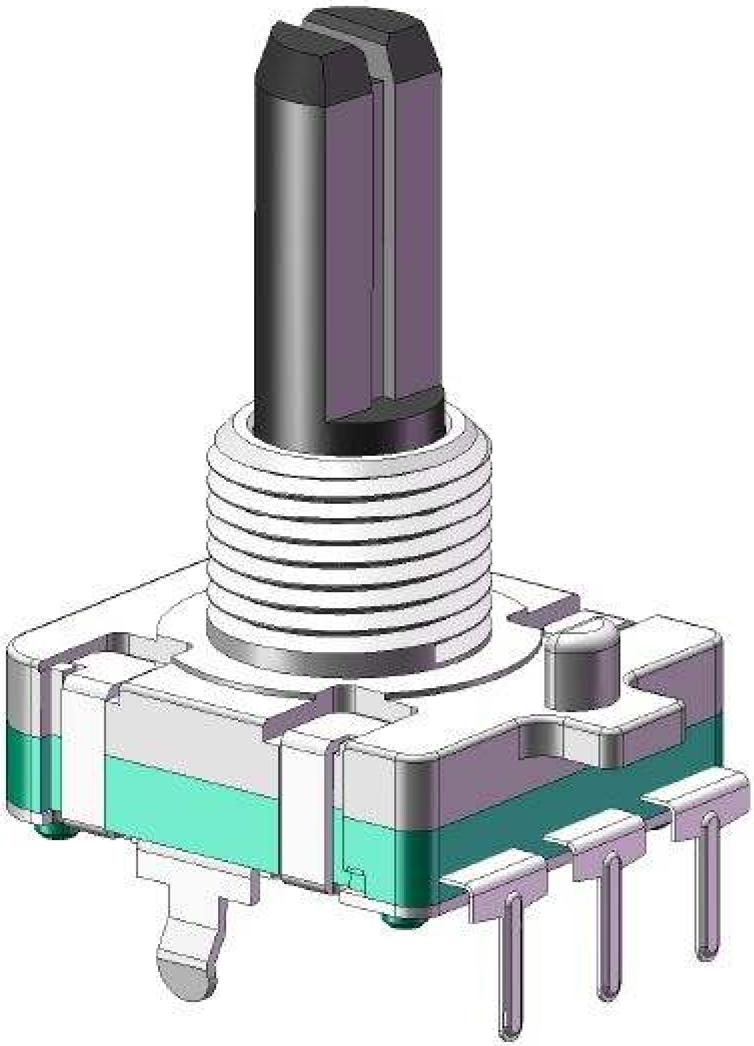
(continued from previous page)

```
        delay(wait);

        for (int i=0; i < strip.numPixels(); i=i+3) {
          strip.setPixelColor(i+q, 0);        //turn every third pixel off
        }
    }
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
   return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else {
   WheelPos -= 170;
   return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
//*************************************************************************************
```

Test Code

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Then we can see the four RGB LED are lighting in various colors.

6.2.33 Project 33: Rotary Encoder

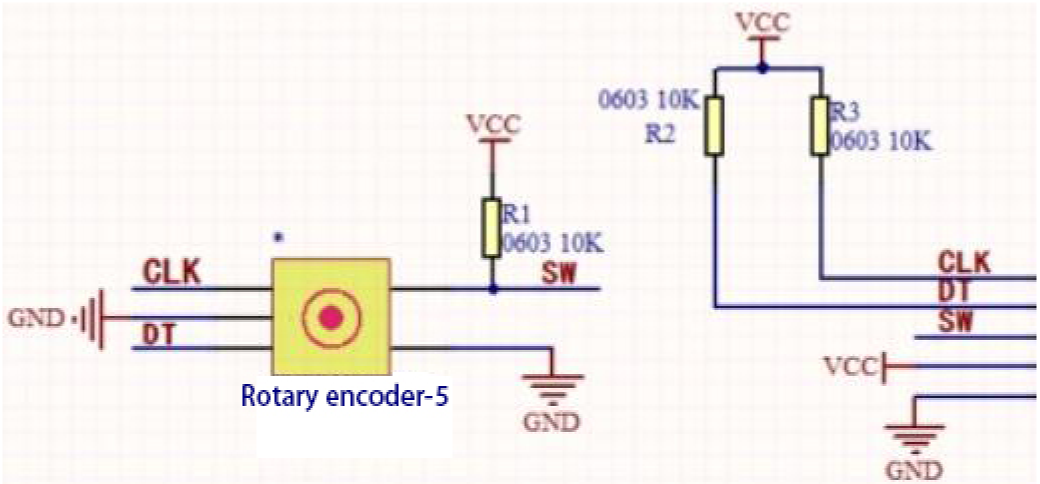


Overview

In this kit, there is a Keyestudio rotary encoder, dubbed as switch encoder. It is applied to automotive electronics, multimedia audio, instrumentation, household appliances, smart home, medical equipment and so on.

In the experiment, it is used for counting. When we rotate the rotary encoder clockwise, the set data adds 1; if you rotate it anticlockwise, the set data substructs1; and when the middle button is pressed, the value will be show in the serial monitor.

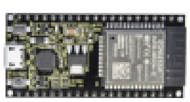
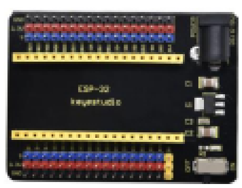
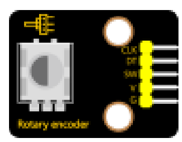
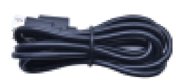
Working Principle



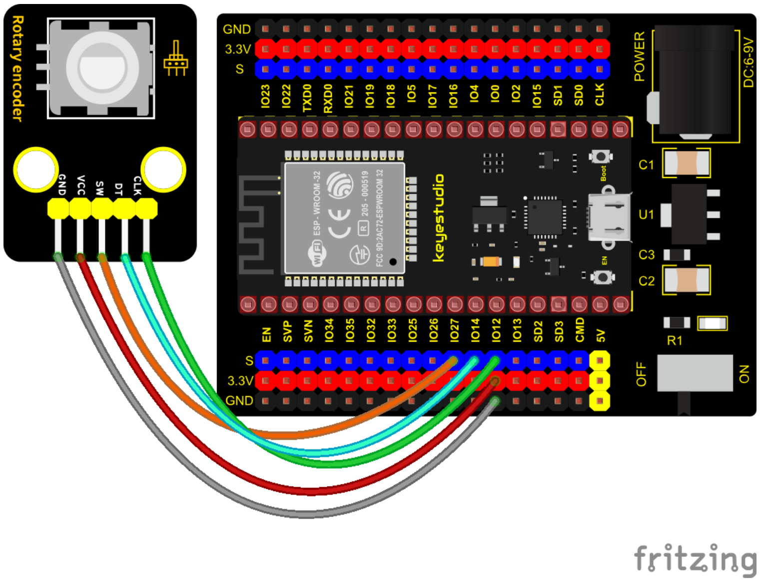
The incremental encoder converts the displacement into a periodic electric signal, and then converts this signal into a counting pulse, and the number of pulses indicates the size of the displacement.

This module mainly uses 20-pulse rotary encoder components. It can calculate the number of pulses output during clockwise and reverse rotation. There is no limit to count rotation. It resets to the initial state, that is, starts counting from 0.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Encoder*1	5P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : Encoder
 * Description : Rotary encoder module counting.
 * Auther      : http//www.keyestudio.com
*/
//Interfacing Rotary Encoder with Arduino
//Encoder Switch -> pin 27
//Encoder DT -> pin 14
//Encoder CLK -> pin 12

int Encoder_DT  = 14;
int Encoder_CLK  = 12;
int Encoder_Switch = 27;

int Previous_Output;
int Encoder_Count;

void setup() {
  Serial.begin(9600);

  //pin Mode declaration
  pinMode (Encoder_DT, INPUT);
  pinMode (Encoder_CLK, INPUT);
  pinMode (Encoder_Switch, INPUT);

  Previous_Output = digitalRead(Encoder_DT); //Read the inital value of Output A
}

void loop() {
  //aVal = digitalRead(pinA);

  if (digitalRead(Encoder_DT) != Previous_Output)
```

(continues on next page)

(continued from previous page)

```
  {
    if (digitalRead(Encoder_CLK) != Previous_Output)
    {
      Encoder_Count ++;
      Serial.println(Encoder_Count);
    }
    else
    {
      Encoder_Count--;
      Serial.println(Encoder_Count);
    }
  }

  Previous_Output = digitalRead(Encoder_DT);

  if (digitalRead(Encoder_Switch) == 0)
  {
    delay(5);
    if (digitalRead(Encoder_Switch) == 0) {
      Serial.println("Switch pressed");
      while (digitalRead(Encoder_Switch) == 0);
    }
  }
}
//*************************************************************************************
```

Code Explanation

Set CLK to GPIO12 and DAT to GPIO14.

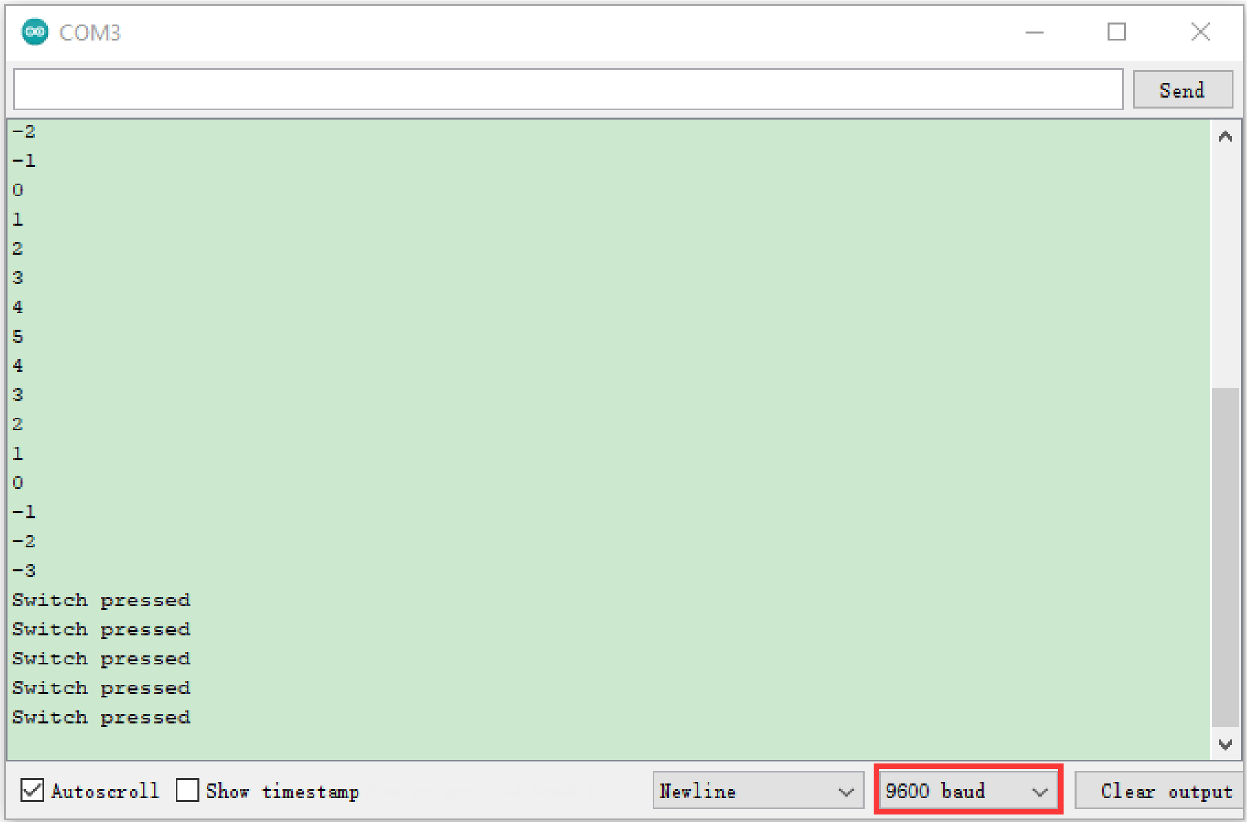
This code is set well in the library file. When CLK descends, read the voltage of DAT, when DAT is a HIGH level, the value of the rotary encoder is added by 1; when DAT is a LOW level, the value of the rotary encoder is cut down 1.

Set the pin of the button(GPIO27) to LOW and print.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600;

Rotate the knob on the rotary encoder clockwise, the displayed data will increase; on the contrary, in anticlockwise way, the data will decrease. Equally, press the button on the rotary encoder,“Switch pressed”will be shown.



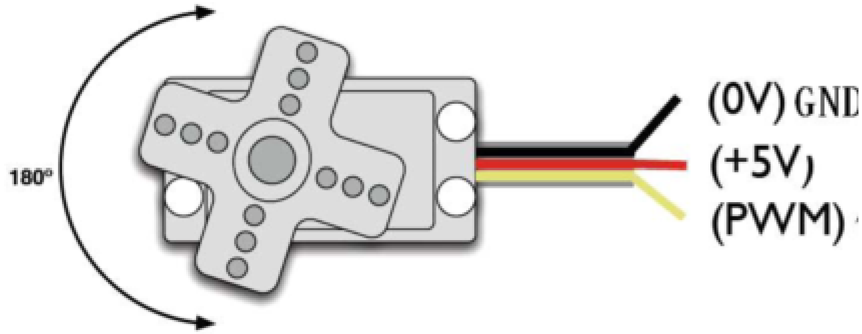
6.2.34 Project 34: Servo Control

Overview

Servo motor is a position control rotary actuator. It mainly consists of a housing, a circuit board, a core-less motor, a gear and a position sensor.

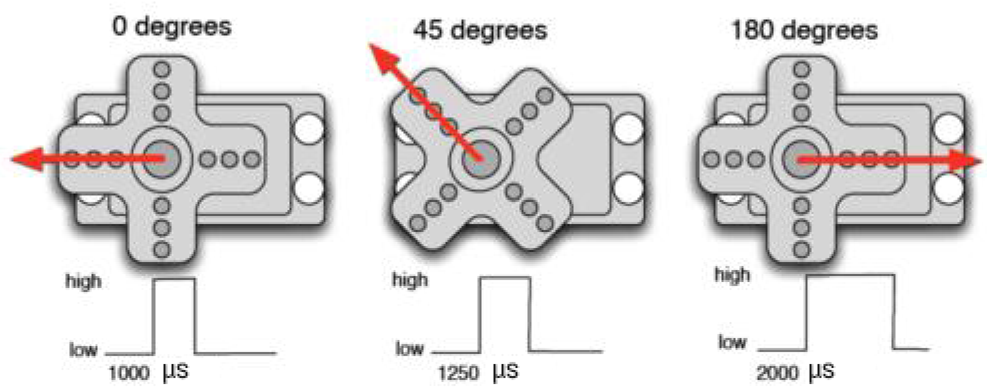
In general, servo has three lines in brown, red and orange. The brown wire is grounded, the red one is a positive pole line and the orange one is a signal line.

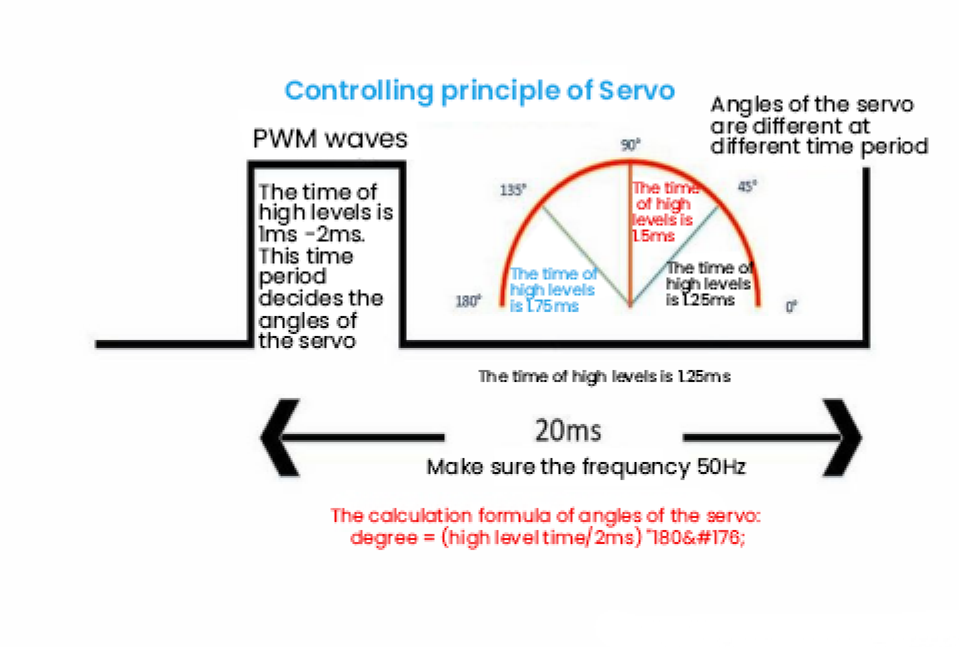
Working Principle



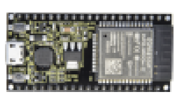
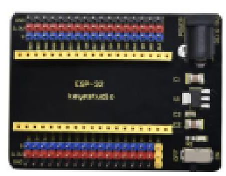
The rotation angle of servo motor is controlled by regulating the duty cycle of PWM (Pulse-Width Modulation) signal. The standard cycle of PWM signal is 20ms (50Hz).

Theoretically, the width is distributed between 1ms-2ms, but in fact, it's between 0.5ms-2.5ms. The width corresponds the rotation angle from 0° to 180°. But note that for different brand motors, the same signal may have different rotation angles.

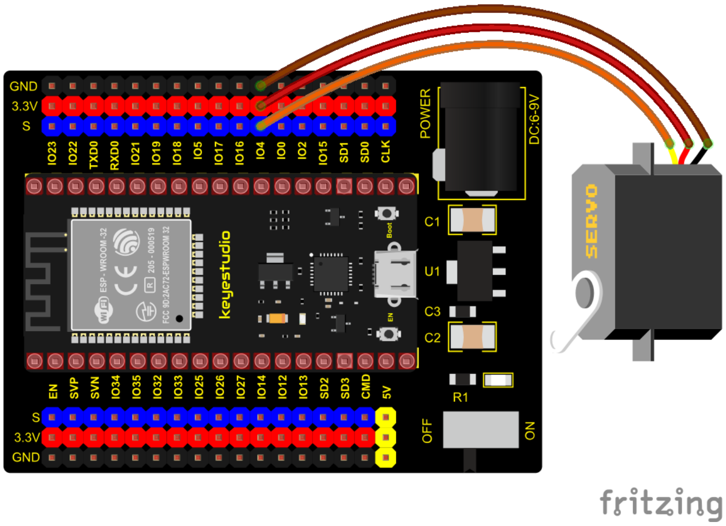




Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Servo*1	Micro USB Cable*1

Connection Diagram



Test Code 1

```
//**********************************************************************
/*
 * Filename    : Servo_1
 * Description : Steering gear rotation Angle 0-90-180, repeatly
 * Auther      : http//www.keyestudio.com
*/
int servoPin = 4;//steering gear PIN

void setup() {
  pinMode(servoPin, OUTPUT);//steering pin is set to output
}
void loop() {
  servopulse(servoPin, 0);//Rotate it to zero degrees
  delay(1000);//delay 1S
  servopulse(servoPin, 90);//Rotate it to 90 degrees
  delay(1000);
  servopulse(servoPin, 180);//Rotate it to 180 degrees
  delay(1000);
}

void servopulse(int pin, int myangle) { //Impulse function
  int pulsewidth = map(myangle, 0, 180, 500, 2500); //Map Angle to pulse width
  for (int i = 0; i < 10; i++) { //Output a few more pulses
    digitalWrite(pin, HIGH);//Set the steering gear interface level to high
    delayMicroseconds(pulsewidth);//The number of microseconds of delayed pulse width value
    digitalWrite(pin, LOW);//Lower the level of steering gear interface
    delay(20 - pulsewidth / 1000);
  }
}
//**********************************************************************
```

Code Explanation 1

map(value, fromLow, fromHigh, toLow, toHigh)

Value is the value we map. fromLow, fromHigh is the maximum and minimum value, toLow, toHigh are the upper limit and lower limit we map.

For example, map(myangle, 0, 180, 500, 2500) means that an angle value myangle (0°-180°the mapping range is from 500us to 2500us.

We use the function **servopulse()** to make the servo move. We also make the servo rotate 0°, 90°and 180°cyclically.

Test Result 1

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, the servo will rotate 0°90° and 180° cyclically.

Test Code 2

```
//**********************************************************************
/*
 * Filename    : Servo Sweep
 * Description : Control the servo motor for sweeping
 * Auther      : http//www.keyestudio.com
*/
#include <ESP32Servo.h>

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

int posVal = 0;    // variable to store the servo position
int servoPin = 4; // 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
  }
}
//********************************************************************************
```

Code Explanation 2

myservo. write (pos) is the rotation angle to POS. myservo. read () reads the current angle value of the servo.

Test Result 2

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, the servo will rotate from 0° to 180° by moving 1° for each 15ms.

6.2.35 Project 35: Ultrasonic Sensor

Bats and some marine animals are able to use high frequencies of sound for echolocation or communication. They can emit ultrasonic waves from the larynx through the mouth or nose and use the sound waves that bounce back to orient and determine the position, size and whether nearby objects are moving.

Ultrasonic is a frequency higher than 20000 Hz sound wave, which has a good direction, a strong penetration ability, and is easy to obtain more concentrated sound energy as well as spread far in the water. It can be used for ranging, speed measurement, cleaning, welding, gravel, sterilization and disinfection. What's more, it has many applications in medicine, military, industry and agriculture.

Overview

In this kit, there is a keyes HC-SR04 ultrasonic sensor, which can detect obstacles in front and the detailed distance between the sensor and the obstacle. Its principle is the same as that of bat flying. It can emit the ultrasonic signals that cannot be heard by humans. When these signals hit an obstacle and come back immediately. The distance between the sensor and the obstacle can be calculated by the time gap of emitting signals and receiving signals.

In the experiment, we use the sensor to detect the distance between the sensor and the obstacle, and print the test result.

Working Principle

The most common ultrasonic ranging method is the echo detection. As shown below; when the ultrasonic emitter emits the ultrasonic waves towards certain direction, the counter will count. The ultrasonic waves travel and reflect back once encountering the obstacle. Then the counter will stop counting when the receiver receives the ultrasonic waves coming back.

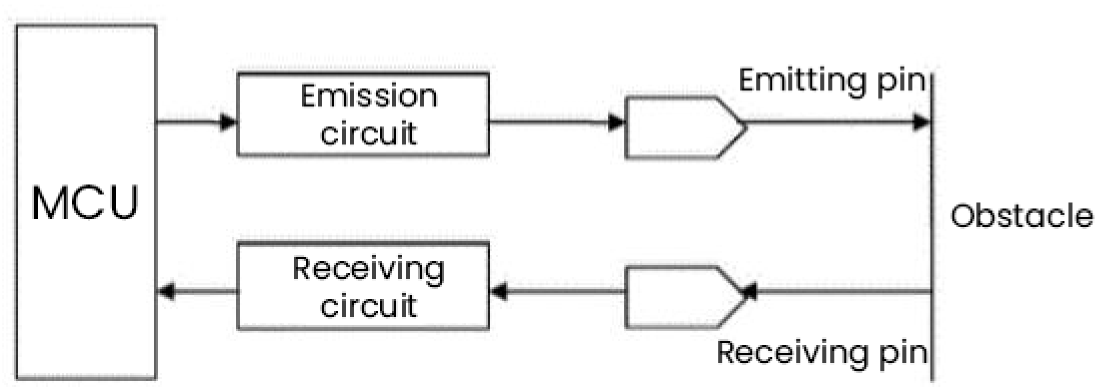
The ultrasonic wave is also sound wave, and its speed of sound V is related to temperature. Generally, it travels 340m/s in the air. According to time t, we can calculate the distance s from the emitting spot to the obstacle. $s = 340t/2$ The HC-SR04 ultrasonic ranging module can provide a non-contact distance sensing function of 2cm-400cm, and the ranging accuracy can reach as high as 3mm; the module includes an ultrasonic transmitter, receiver and control circuit. Basic working principle:

1). First pull down the TRIG, and then trigger it with at least 10us high level signal;

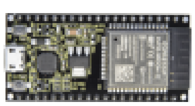
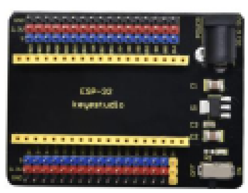
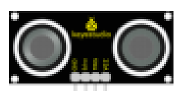
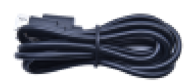
2). After triggering, the module will automatically transmit eight 40KHZ square waves, and automatically detect whether there is a signal to return.

3). If there is a signal returned back, through the ECHO to output a high level, the duration time of high level is actually the time from emission to reception of ultrasonic.

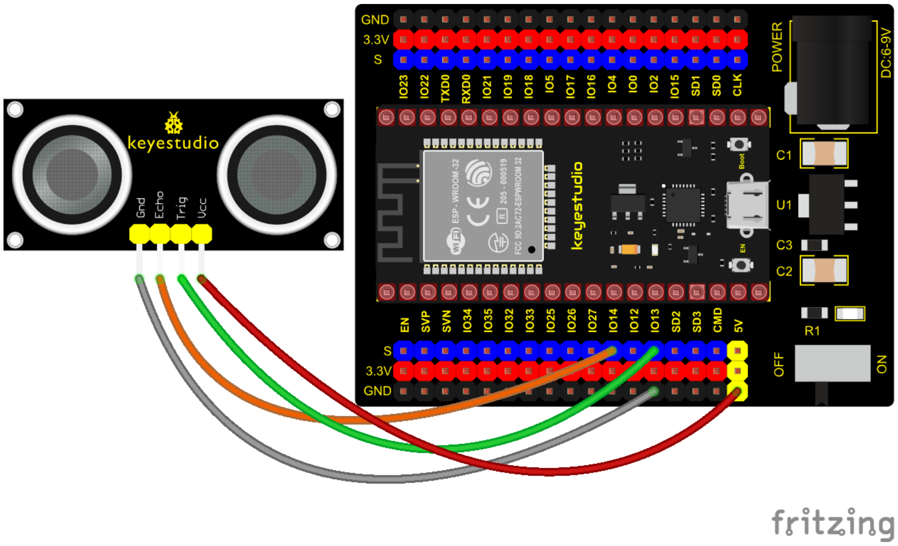
$$TestDistance = HighLevelDuration * 340m/s * 0.5$$



Components

				
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio SR01 Ultrasonic Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



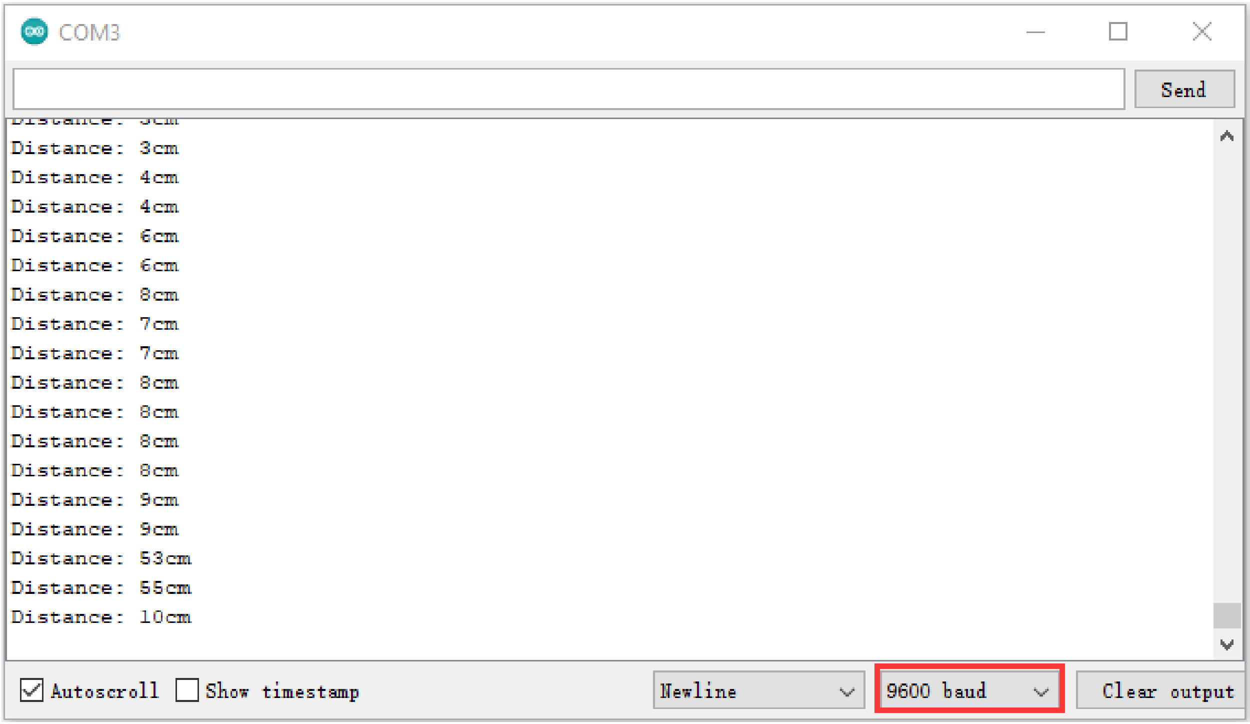
Test Code

```
//**********************************************************************************
/*
 * Filename    : Ultrasonic
 * Description : Use the ultrasonic module to measure the distance.
 * Auther      : http//www.keyestudio.com
*/
const int TrigPin = 13; // define TrigPin
const int EchoPin = 14; // define EchoPin.
int duration = 0; // Define the initial value of the duration to be 0
int distance = 0;//Define the initial value of the distance to be 0
void setup()
{
  pinMode(TrigPin , OUTPUT); // set trigPin to output mode
  pinMode(EchoPin , INPUT); // set echoPin to input mode
  Serial.begin(9600);  // Open serial monitor at 9600 baud to see ping results.
}
void loop()
{
 // make trigPin output high level lasting for 10s 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
  duration = pulseIn(EchoPin , HIGH);
  // calculate the distance according to the time
  distance = (duration/2) / 28.5 ;
  Serial.print("Distance: ");
  Serial.print(distance); //Serial port print distance value
  Serial.println("cm");
  delay(300); // Wait 100ms between pings (about 20 pings/sec).
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600.

The serial monitor will print the distance between the ultrasonic sensor and the object.



6.2.36 Project 36: IR Receiver Module

Overview

Infrared remote control is currently the most widely used means of communication and remote control, which has the characteristics of small volume, low power consumption, strong function and low cost. Therefore, recorder, audio equipment, air conditioning machine and toys and other small electrical devices have also used the infrared remote control.

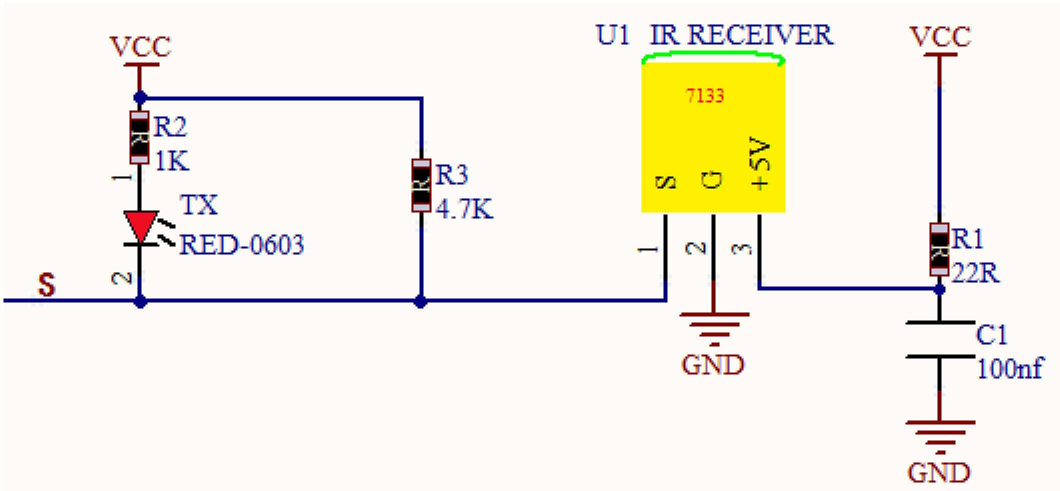
Its transmitting circuit is the use of infrared light emitting diode to emit modulated infrared light wave. The circuit is composed of infrared receiving diode, triode or silicon photocell. They convert infrared light emitted by infrared emitter into corresponding electrical signal, and then send back amplifier.

In this experiment, we need to know how to use the infrared receiving sensor. The infrared receiving sensor mainly uses the VS1838B infrared receiving sensor element. It integrates receiving, amplifying, and demodulating. The internal IC has already completed the demodulation, and the output is a digital signal. It can receive 38KHz modulated remote control signal.

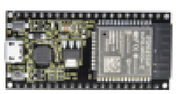
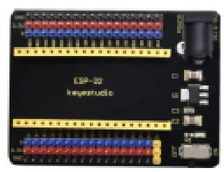
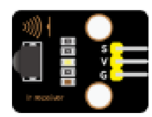
In the experiment, we use the IR receiver to receive the infrared signal emitted by the external infrared transmitting device, and display the received signal in the shell.

Working Principle

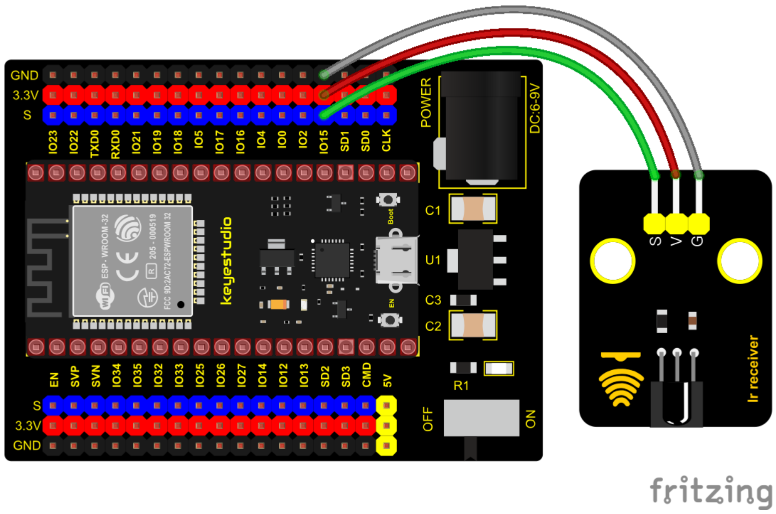
The main part of the IR remote control system is modulation, transmission and reception. The modulated carrier frequency is generally between 30khz and 60khz, and most of them use a square wave of 38kHz and a duty ratio of 1/3. A 4.7K pull-up resistor R3 is added to the signal end of the infrared receiver.



Components

		
ESP32 Board*11	ESP32 Expansion Board*1	DIY IR Receiver*1
		
3P Dupont Wire*1	Micro USB Cable*1	Remote Control*1

Connection Diagram



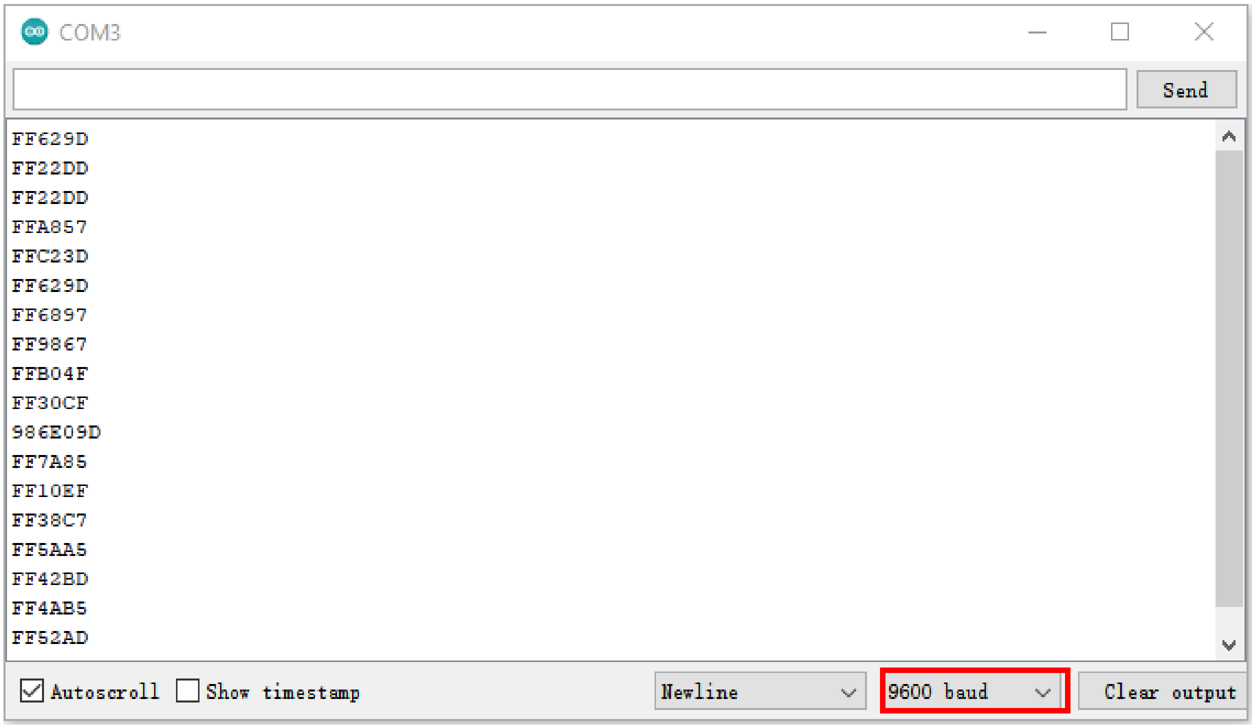
Test Code

```
//**********************************************************************************
/*
 * Filename    : IR Receiver
 * Description : Decode the infrared remote control and print it out through the serial port.
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

const uint16_t recvPin = 15; // Infrared receiving pin
IRrecv irrecv(recvPin);      // Create a class object used to receive class
decode_results results;       // Create a decoding results class object

void setup() {
  Serial.begin(9600);       // Initialize the serial port and set the baud rate to 9600
  irrecv.enableIRIn();        // Start the receiver
  Serial.print("IRrecvDemo is now running and waiting for IR message on Pin ");
  Serial.println(recvPin);   //print the infrared receiving pin
}

void loop() {
  if (irrecv.decode(&results)) {          // Waiting for decoding
    serialPrintUint64(results.value, HEX);// Print out the decoded results
    Serial.println("");
    irrecv.resume();                      // Release the IRremote. Receive the next value
  }
  delay(1000);
}
//**********************************************************************************
```

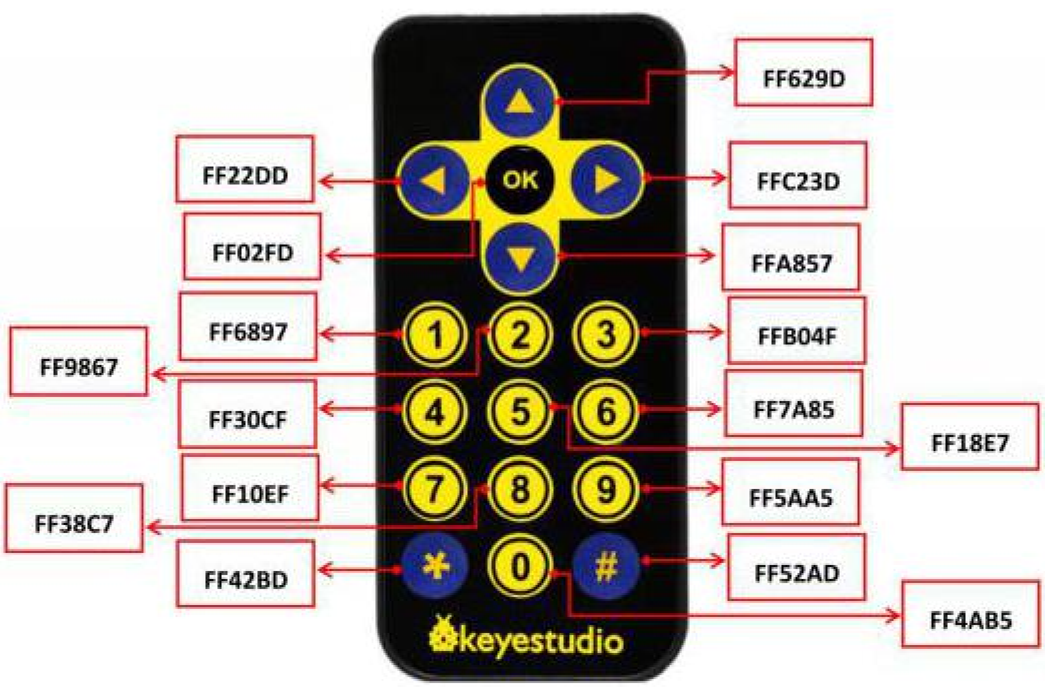
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32.

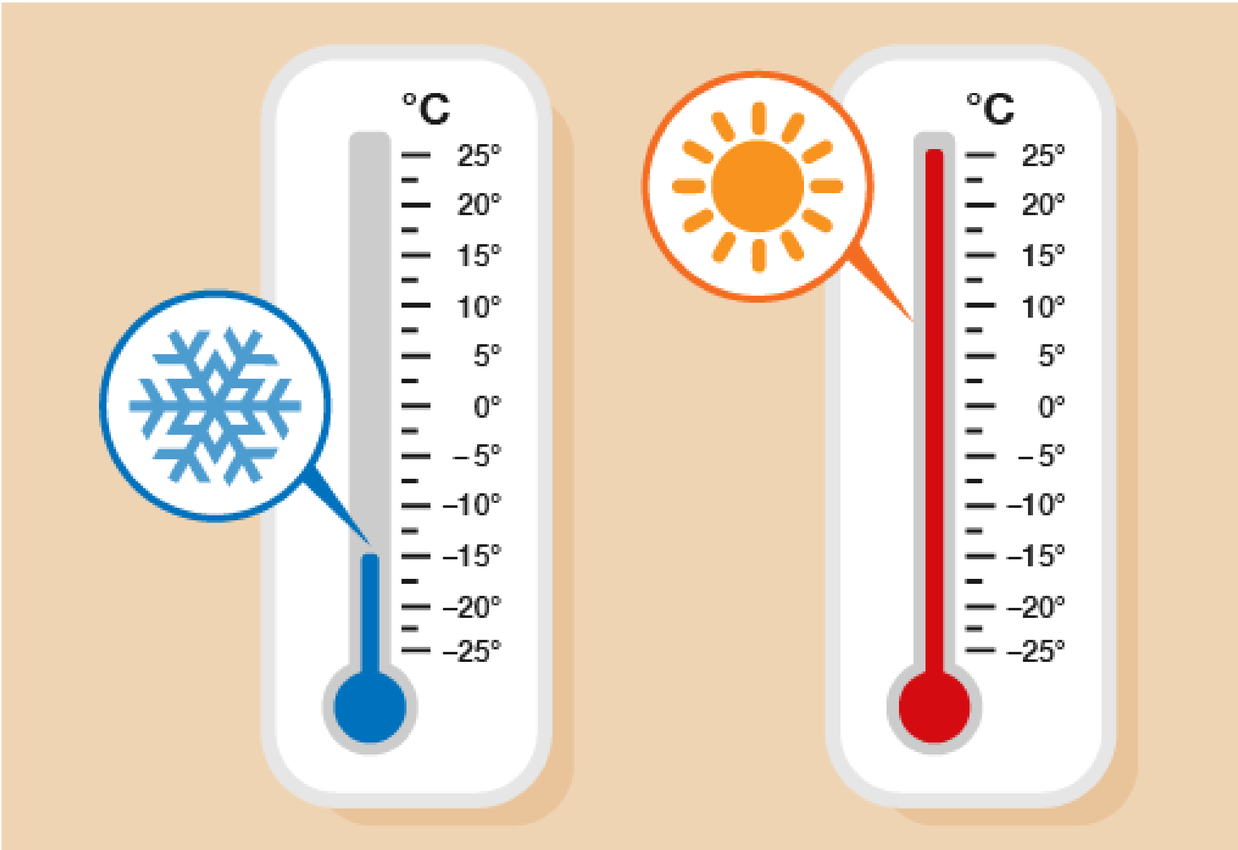
After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set baud rate to 9600; Find the infrared remote control, pull out the insulating sheet, and press the button at the receiving head of the infrared receiving sensor. After receiving the signal, the LED on the infrared receiving sensor also starts to flash, as shown in the figure below.



Write down the key code value associated with the infrared remote with each key, as you will need this information later.



6.2.37 Project 37: DS18B20 Temperature Sensor

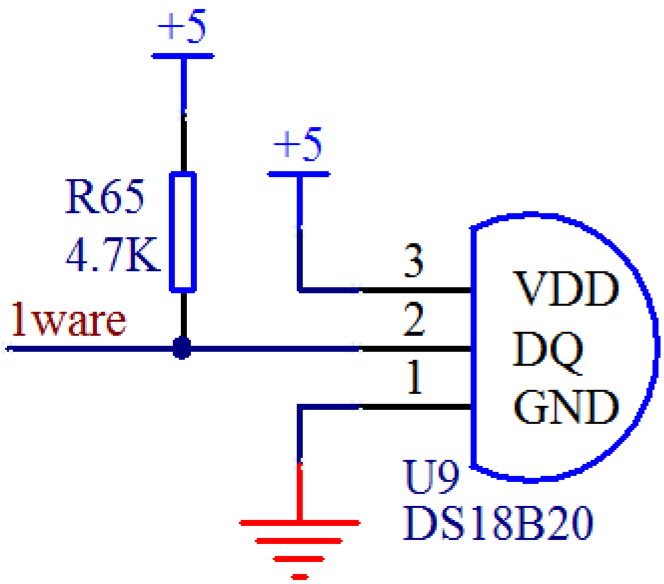


Description

In this kit, there is a DS18B20 temperature sensor, which is from maxim. The MCU can communicate with the DS18B20 through 1-Wire protocol, and finally read the temperature.

In this experiment, we will use this temperature sensor to measure the temperature in the current environment. The test result is °C, ranging from -55°C to +125°C. We will display the test result on shell.

Working Principle



The hardware interface of the 1-Wire bus is very simple, just connect the data pin of the DS18B20 to an IO port of the microcontroller. The timing of the 1-Wire bus is relatively complex. Many students can't understand the timing diagram independently here. We have encapsulated the complex timing operations in the library, and you can use the library functions directly.

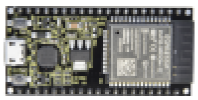
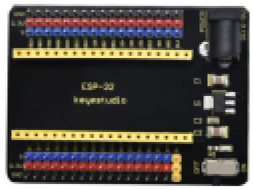
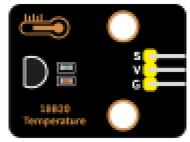
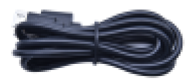
Schematic Diagram of DS18B20

This can save up to 12-bit temperature vale. In the register, save in code complement. As shown below;

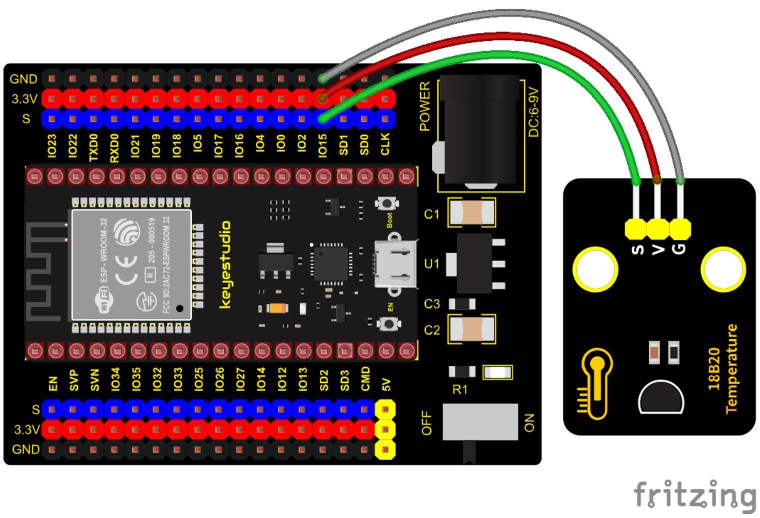
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB
MSb			(unit = °C)				LSb	
S	S	S	S	S	2^6	2^5	2^4	MSB

A total of 2 bytes, LSB is the low byte, MSB is the high byte, where MSb is the high byte of the byte, LSb is the low byte of the byte. As you can see, the binary number, the meaning of the temperature represented by each bit, is expressed. Among them, S represents the sign bit, and the lower 11 bits are all powers of 2, which are used to represent the final temperature. The temperature measurement range of DS18B20 is from -55 degrees to +125 degrees, and the expression form of temperature data, S represents positive and negative temperature, and the resolution is 2, which is 0.0625.

Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY 18B20 Temperature Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

Required Components



Test Code

```
//**********************************************************************************
/*
 * Filename    : ds18b20
 * Description : Read the temperature of ds18B20
 * Auther      : http//www.keyestudio.com
*/
#include <DS18B20.h>

//ds18b20 pin to 15
DS18B20 ds18b20(15);

void setup() {
  Serial.begin(9600);
}

void loop() {
  double temp = ds18b20.GetTemp();//Read the temperature
```

(continues on next page)

(continued from previous page)

```
  temp *= 0.0625;//The conversion accuracy is 0.0625/LSB
  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println("C");
  delay(1000);
}
//*************************************************************************************
```

Code Explanation

We set the pin to GPIO15 and obtain the temperature in the unit of ℃.

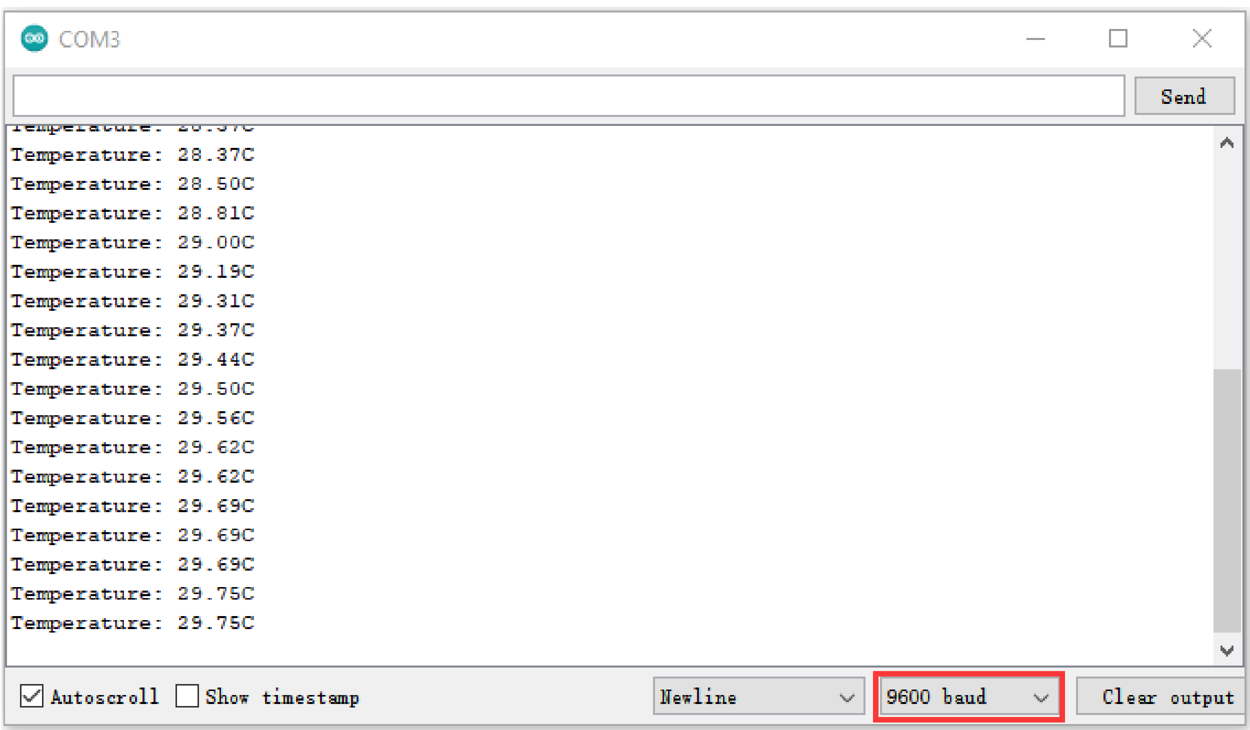
Set a double decimal variable to temp, and assign the measured result to temp.

The serial monitor displays the temp value, and the baud rate needs to be set before displaying (our default setting is 9600, which can be changed).

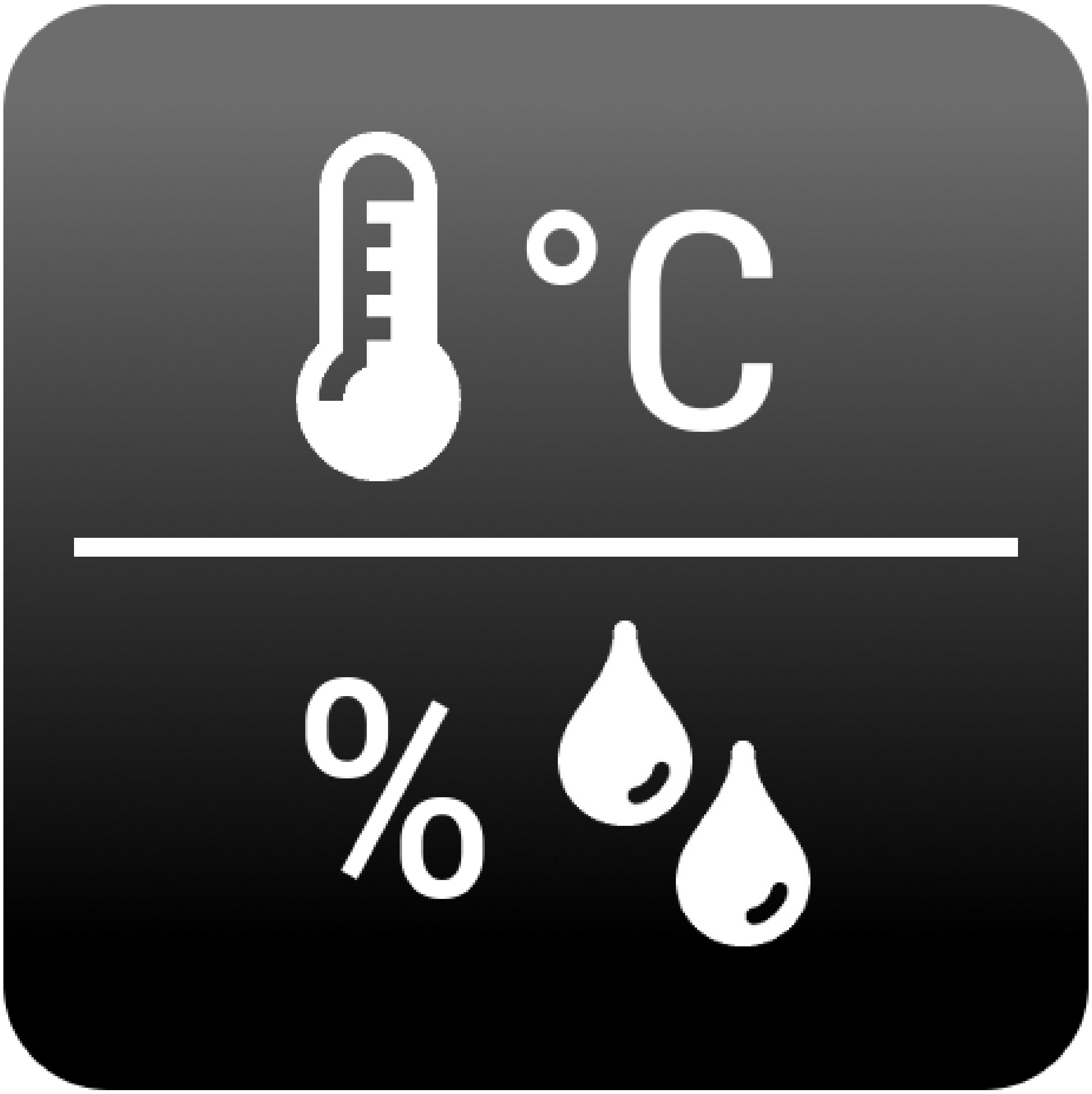
We add the unit behind the data. If the unit is directly set to ℃, the test result will be garbled. So we directly replace ℃ with C.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600; The monitor will display the temperature of the current environment, as shown below.



6.2.38 Project 38: XHT11 Temperature and Humidity Sensor

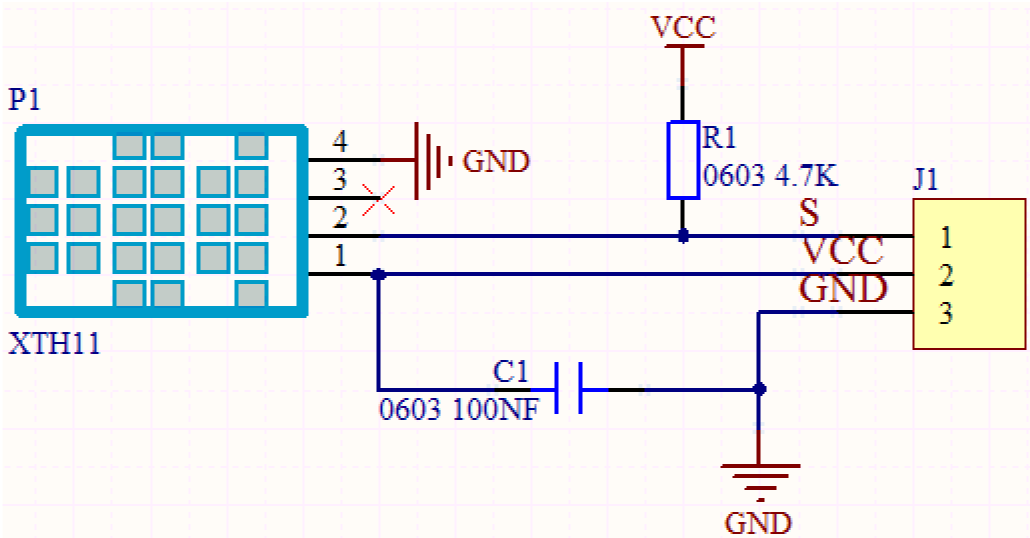


Description

This DHT11 temperature and humidity sensor is a composite sensor which contains a calibrated digital signal output of the temperature and humidity.

DHT11 temperature and humidity sensor uses the acquisition technology of the digital module and temperature and humidity sensing technology, ensuring high reliability and excellent long-term stability.

It includes a resistive element and a NTC temperature measuring device.



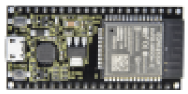
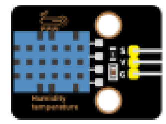
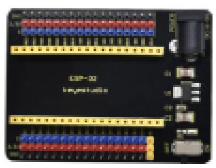
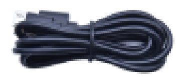
Working Principle

The communication and synchronization between the single-chip microcomputer and XHT11 adopts the single bus data format. The communication time is about 4ms. The data is divided into fractional part and integer part.

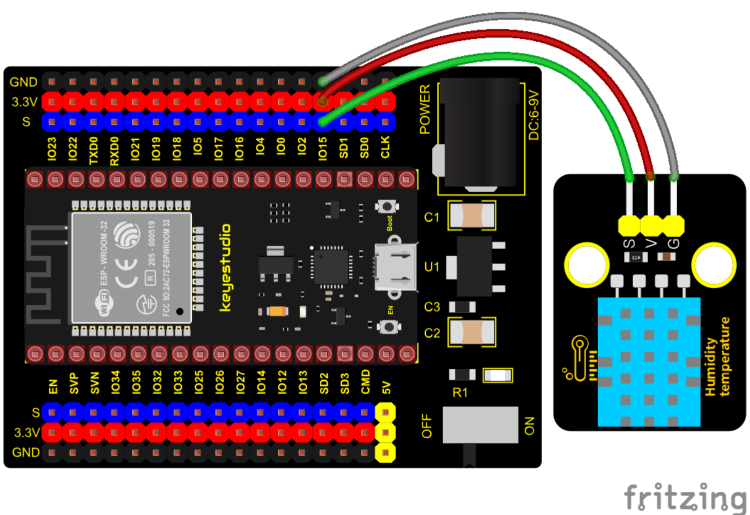
Operation process: A complete data transmission is 40bit, high bit first out. Data format: 8bit humidity integer data + 8bit humidity decimal data + 8bit temperature integer data + 8bit temperature decimal data + 8bit checksum

8-bit checksum: 8-bit humidity integer data + 8-bit humidity decimal data + 8-bit temperature integer data + 8-bit temperature decimal data "Add the last 8 bits of the result.

Required Components

		
ESP32Board*1	Keyestudio XHT11 Temperature and Humidity Sensor *1 (compatible with DHT11)	
		
ESP32 Expansion Board*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : xht11
 * Description : Read the temperature and humidity values of XHT11.
 * Auther      : http//www.keyestudio.com
```

(continues on next page)

(continued from previous page)

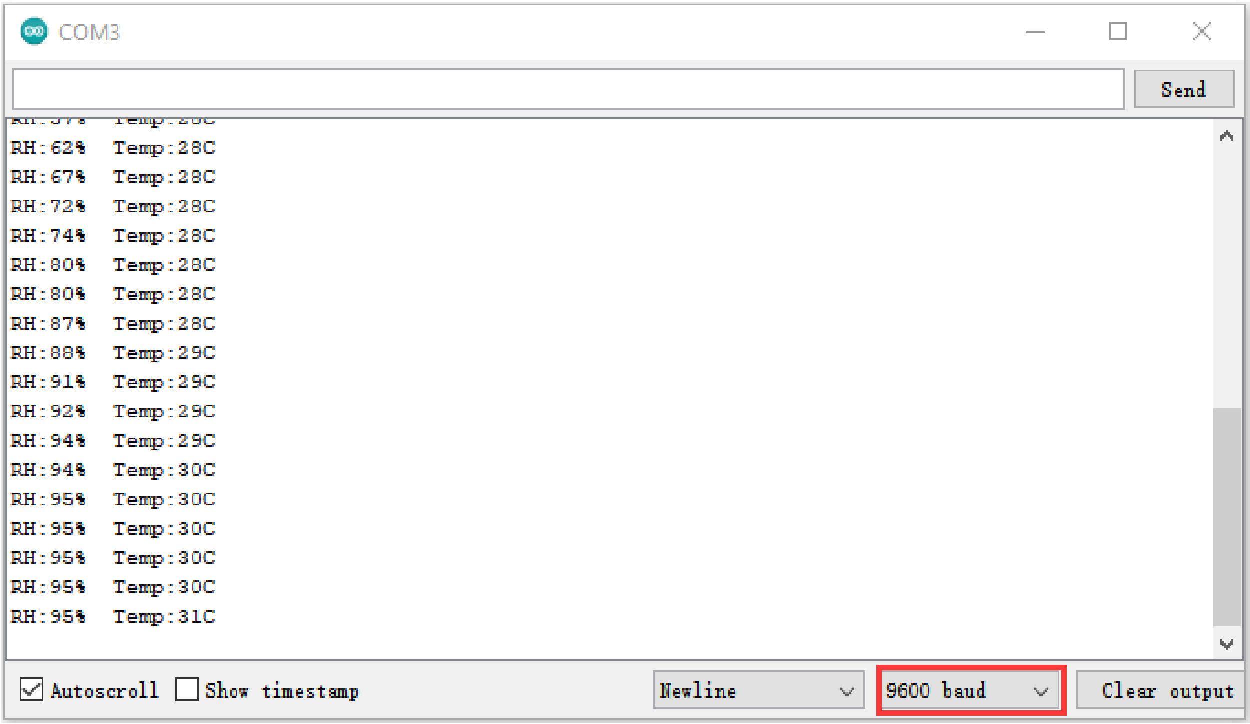
```
*/
#include "xht11.h"
//gpio15
xht11 xht(15);

unsigned char dht[4] = {0, 0, 0, 0};//Only the first 32 bits of data are received, not␣
↪the parity bits
void setup() {
  Serial.begin(9600);//Start the serial port monitor and set baud rate to 9600
}

void loop() {
  if (xht.receive(dht)) { //Returns true when checked correctly
    Serial.print("RH:");
    Serial.print(dht[0]); //The integral part of humidity, DHT [1] is the fractional part
    Serial.print("%  ");
    Serial.print("Temp:");
    Serial.print(dht[2]); //The integral part of temperature, DHT [3] is the fractional␣
↪part
    Serial.println("C");
  } else {    //Read error
    Serial.println("sensor error");
  }
  delay(1000);  //It takes 1000ms to wait for the device to read
}
//*************************************************************************************
```

Code Explanation

1). We set the pin to GPIO15, and store the detected temperature and humidity data in the dht[4] array. 2). We add units behind the data. If the temperature unit is directly set to °C, the test results may be wrong, so we directly replace °C with C; the humidity unit is directly set to %.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set baud rate to 9600; The monitor will display the temperature and humidity data of the current environment, as shown below.


COM3
Send
RH:62% Temp:28C
RH:67% Temp:28C
RH:72% Temp:28C
RH:74% Temp:28C
RH:80% Temp:28C
RH:80% Temp:28C
RH:87% Temp:28C
RH:88% Temp:29C
RH:91% Temp:29C
RH:92% Temp:29C
RH:94% Temp:29C
RH:94% Temp:30C
RH:95% Temp:30C
RH:95% Temp:30C
RH:95% Temp:30C
RH:95% Temp:30C
RH:95% Temp:31C
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

6.2.39 Project 39: DS1307 Clock Module

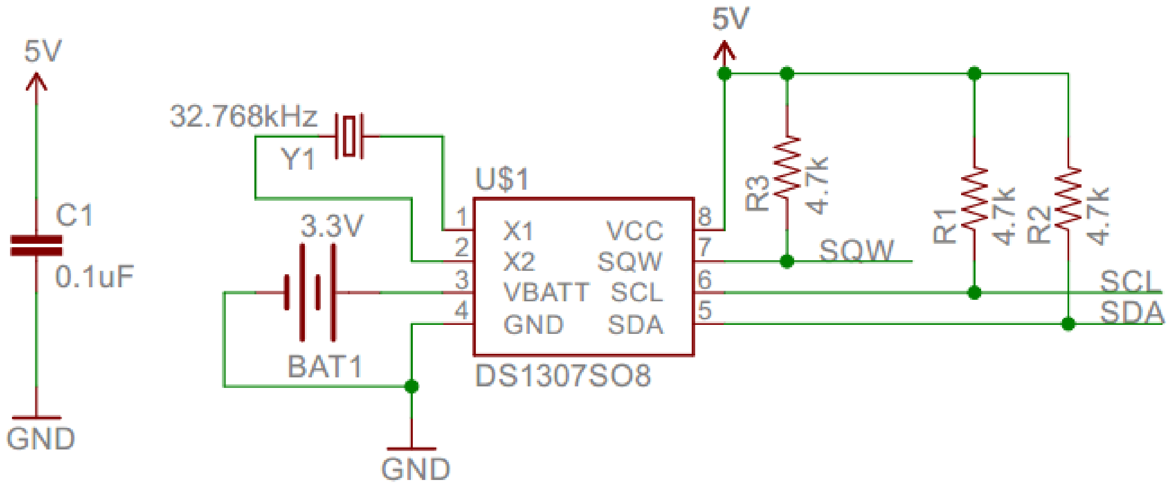
Overview

This module mainly uses the real-time clock chip DS1307, which is the I2C bus interface chip that has second, minute, hour, day, month, year and other functions as well as leap year automatic adjustment function introduced by DALLAS. It can work independently of CPU, and won't' affected by the CPU main crystal oscillator and capacitance as well as keep accurate time. What's more, monthly cumulative error is generally less than 10 s. The chip also has a clock protection circuit in case of main power failure and runs on a back-up battery that denies the CPU read and write access.

At the same time, it contains automatic switching control circuit of standby power supply, making it guarantees the accuracy of system clock in case of power failure of main power supply and other bad environment.

Going forward, the DS1307 chip internal integration has a certain capacity, with power failure protection characteristics of static RAM, which can be used to save some key data.

In the experiment, we use the DS1307 clock module to obtain the system time and print the test results.



Working Principle

Serial real-time clock records year, month, day, hour, minute, second and week; AM and PM indicate morning and afternoon respectively; 56 bytes of NVRAM store data; 2-wire serial port; programmable square wave output; power failure detection and automatic switching circuit; battery current is less than 500nA.

Pins description:

X1, X2: 32.768kHz crystal terminal

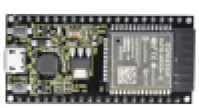
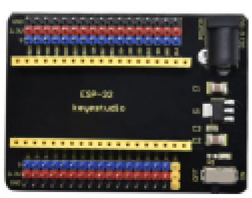
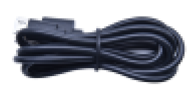
VBAT: +3V input

SDA: serial data

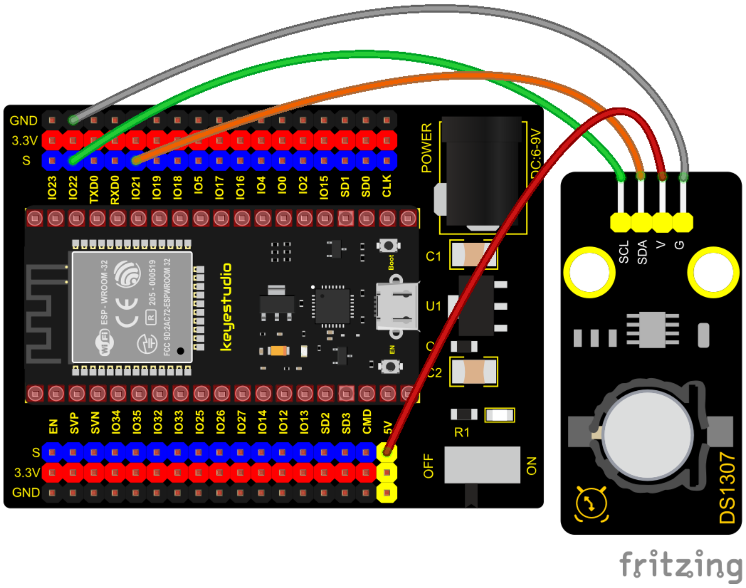
SCL: serial clock

SQW/OUT: square waves/output drivers

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DS1307 Clock Module*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : DS1307 Real Time Clock
 * Description : Read the year/month/day/hour/minute/second/week of DS1307 clock module
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "RtcDS1307.h"  //DS1307 clock module library

RtcDS1307<TwoWire> Rtc(Wire);//i2cport

void setup(){
  Serial.begin(57600);//Set baud rate to 57600
  Rtc.Begin();
  Rtc.SetIsRunning(true);

  Rtc.SetDateTime(RtcDateTime(__DATE__, __TIME__));
}

void loop(){
  // Print year/month/day/hour/minute/second/week
  Serial.print(Rtc.GetDateTime().Year());
  Serial.print("/");
  Serial.print(Rtc.GetDateTime().Month());
  Serial.print("/");
  Serial.print(Rtc.GetDateTime().Day());
  Serial.print("    ");
  Serial.print(Rtc.GetDateTime().Hour());
  Serial.print(":");
  Serial.print(Rtc.GetDateTime().Minute());
  Serial.print(":");
  Serial.print(Rtc.GetDateTime().Second());
```

(continues on next page)

(continued from previous page)

```
  Serial.print("    ");
  Serial.println(Rtc.GetDateTime().DayOfWeek());
  delay(1000);//Delay 1 second
}
//**********************************************************************************
```

Code Explanation

Rtc.GetDateTime(): the obtained current time and date.

Rtc.Begin(): enable DS1307 real-time clock.

Rtc.SetIsRunning(true): run the DS1307 real-time clock, if true changes into false, time will stop.

Rtc.SetDateTime(): set time.

Rtc.GetDateTime().Year(): return year.

Rtc.GetDateTime().Month(): return month.

Rtc.GetDateTime().Day(): return date.

Rtc.GetDateTime().Hour(): return hour.

Rtc.GetDateTime().Minute(): return minute.

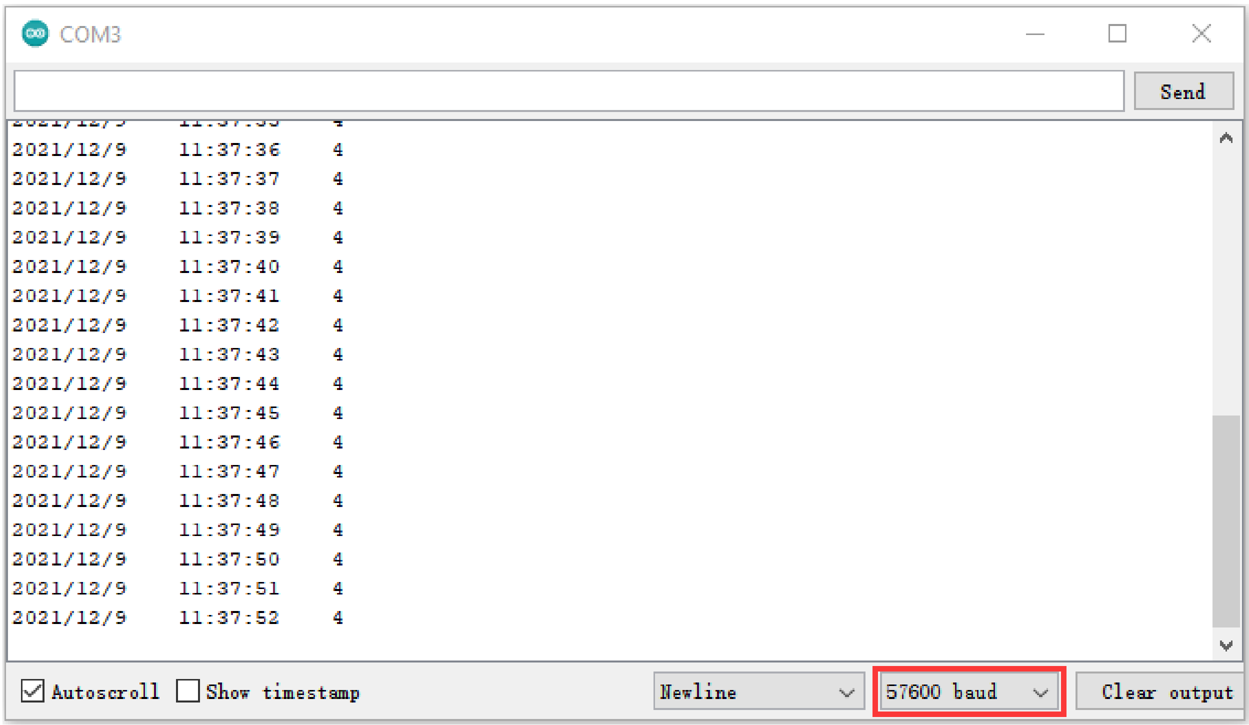
Rtc.GetDateTime().Second(): return second.

Rtc.GetDateTime().DayOfWeek(): return week.

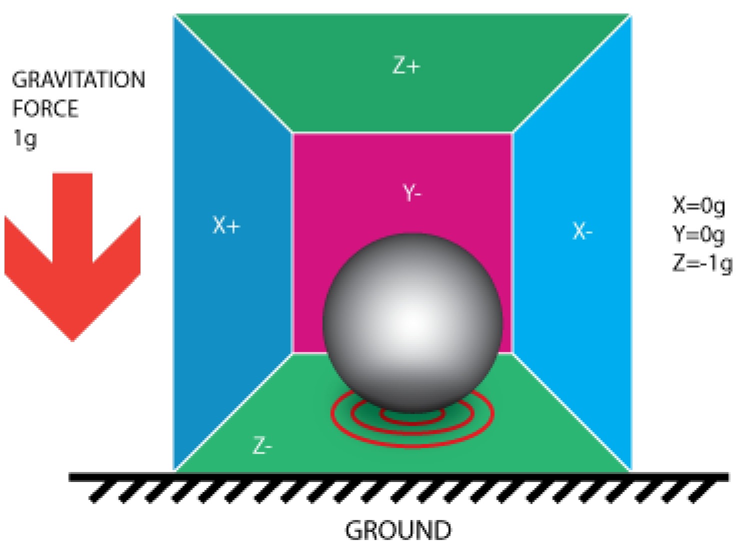
Test Result

Connect the wires according to the experimental wiring diagram, attach the DS1307 sensor to a battery, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on.

Open the serial monitor and set baud rate to 57600. We can see the displayed year, month, day, hour, minute, second and week on the serial monitor, and set the time and date to refresh every second, as shown below:



6.2.40 Project 40: ADXL345 Acceleration Sensor



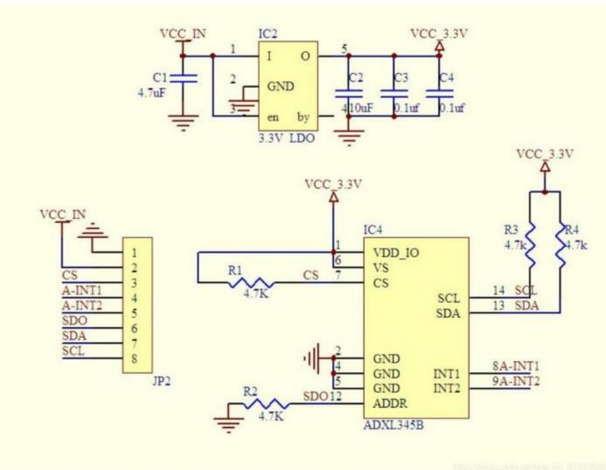
Overview

In this kit, there is a DIY electronic building block ADXL345 acceleration sensor module, which uses the ADXL345BCCZ chip. The chip is a small, thin, low-power 3-axis accelerometer with a high resolution (13 bits) and a measurement range of ±16g that can measure both dynamic acceleration due to motion or impact as well as stationary acceleration such as gravitational acceleration, making the device usable as a tilt sensor.

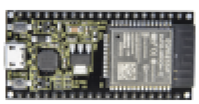
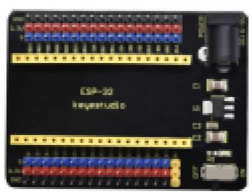
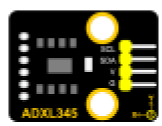
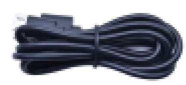
Working Principle

The ADXL345 is a complete 3-axis acceleration measurement system with a selection of measurement ranges of ±2 g, ±4 g, ±8 g or ±16 g. Its digital output data is in 16-bit binary complement format and can be accessed through an SPI (3-wire or 4-wire) or I2C digital interface.

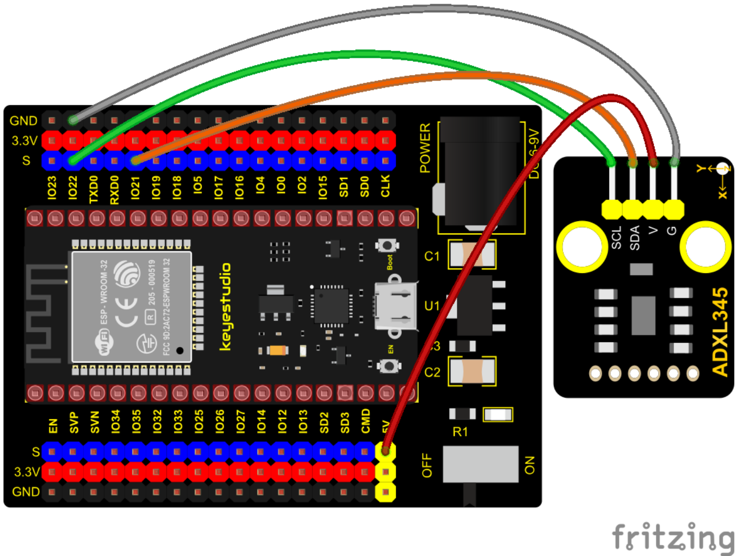
The sensor can measure static acceleration due to gravity in tilt detection applications, as well as dynamic acceleration due to motion or impact. Its high resolution (3.9mg/LSB) enables measurement of tilt Angle changes of less than 1.0°.



Components Required

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio ADXL345 Acceleration Module*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : ADXL345
 * Description : Read the X/Y/Z value of ADXL345
 * Auther      : http//www.keyestudio.com
*/
#include "adxl345_io.h"
//The port is sda-->21,scl-->22
adxl345 adxl345(21, 22);

float out_X, out_Y, out_Z;

void setup() {
  Serial.begin(57600);//Start serial port monitoring and set baud rate to 57600
  adxl345.Init();
}

void loop() {
  adxl345.readXYZ(&out_X, &out_Y, &out_Z);
  Serial.print(out_X);
  Serial.print("g   ");
  Serial.print(out_Y);
  Serial.print("g   ");
  Serial.print(out_Z);
  Serial.println("g");
  delay(100);
}
//**********************************************************************************
```

Code Explanation

Set 3 decimal variables out_X out_Y out_Z, and assign the measured result to out_X out_Y out_Z. The serial monitor displays the value of out_X out_Y out_Z, and the baud rate needs to be set before displaying (our default setting is 57600, which can be changed).

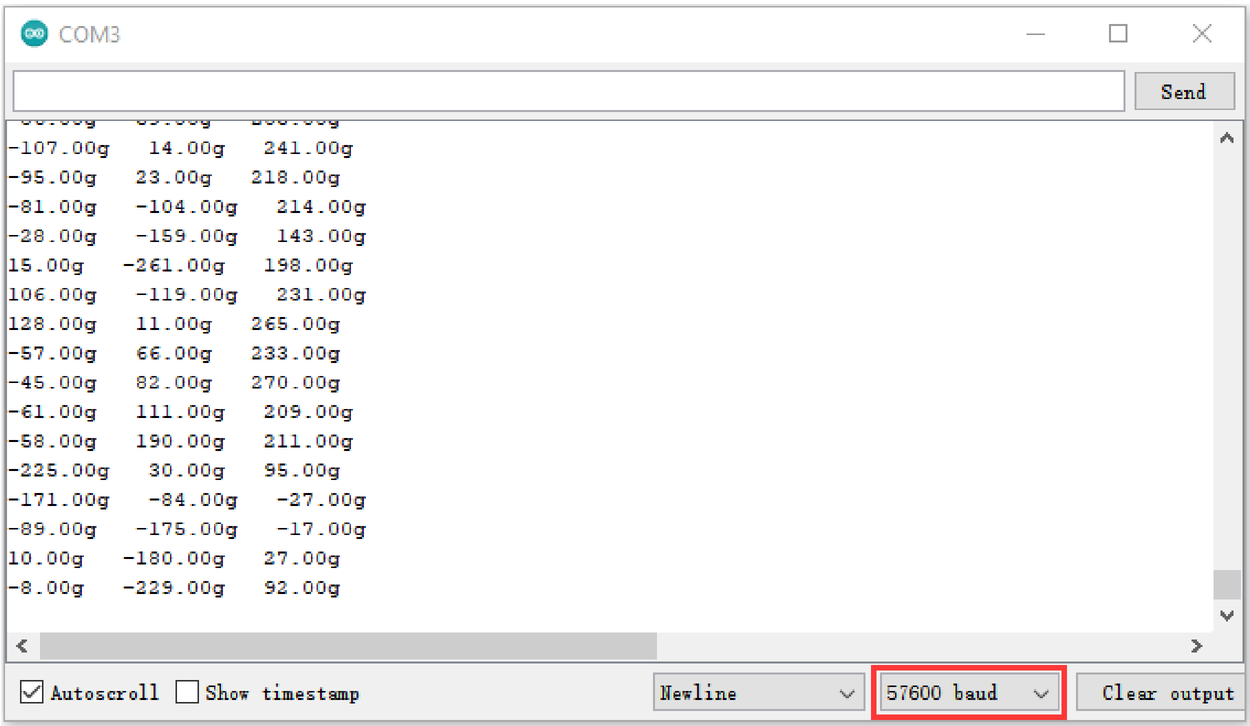
Adxl345.Init; Initialize the ADXX345 accelerometer.

adxl345.readXYZ(&out_X, &out_Y, &out_Z); Get the acceleration value of the X axis and return it to the variables out_X, out_Y, out_Z.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set baud rate to 57600.

The serial monitor displays the value corresponding to the sensor, the unit is mg, as shown in the figure below.



6.2.41 Project 41: TM1650 4-Digit Tube Display

Overview

This module is mainly composed of a 0.36 inch red common cathode 4-digit digital tube, and its driver chip is TM1650. When using it, we only need two signal lines to make the single-chip microcomputer control a 4-bitdigit tube, which greatly saves the IO port resources of the control board.

TM1650 is a special circuit for LED (light emitting diode display) drive control. It integrates MCU input and output control digital interface, data latch, LED drivers, keyboard scanning, brightness adjustment and other circuits.

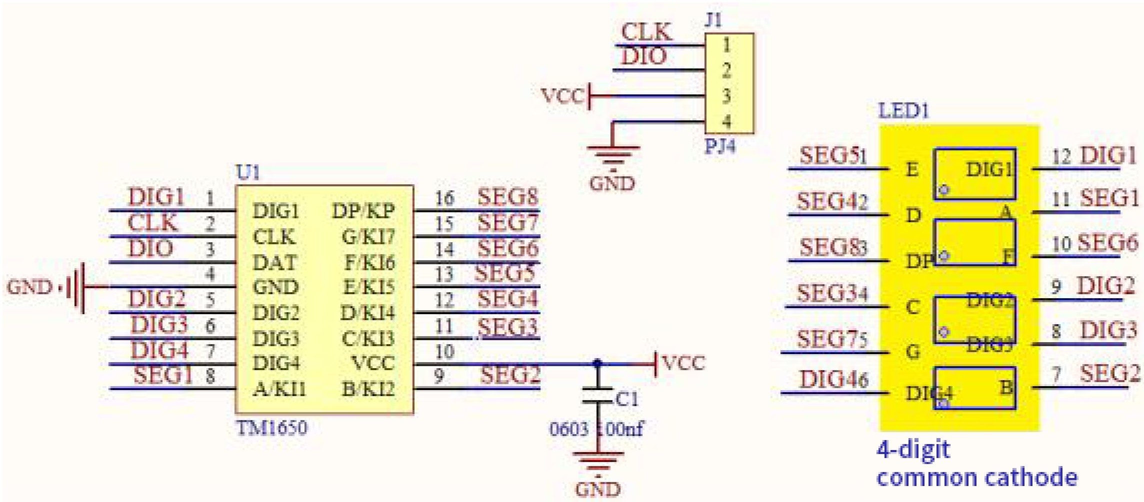
TM1650 has stable performance, reliable quality and strong anti-interference ability.

It can be applied to the application of long-term continuous working for 24 hours.

TM1650 uses 2-wire serial transmission protocol for communication (note that this data transmission protocol is not a standard I2C protocol). The chip can drive the digital tube and save MCU pin resources through two pins and MCU communication.

Working Principle

TM1650 adopts IIC treaty, which uses DIO and CLK buses.



Data command setting: 0x48 means that we light up the digital tube, instead of enable the function of key scanning

B7	B6	B5	B4	B3	B2	B1	B0	Function	Description
×	0	0	0		×	×		Brightness setting	Eight-level brightness
×	0	0	1		×	×			One-level brightness
×	0	1	0		×	×			Two-level brightness
×	0	1	1		×	×			Three-level brightness
×	1	0	0		×	×			Four-level brightness
×	1	0	1		×	×			Five-level brightness
×	1	1	0		×	×			Six-level brightness
×	1	1	1		×	×			Seven-level brightness
×				0	×	×		7/8 segment display control bit	8-segment display way
×				1	×	×			7-segment display way
×					×	×	0	ON/OFF display bit	Off display
×					×	×	1		On display

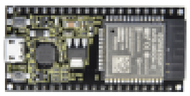
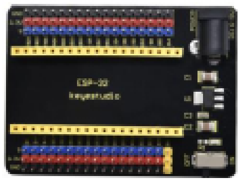
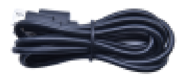
Command display setting:

bit[6:4]: set the brightness of tube display, and 000 is brightest

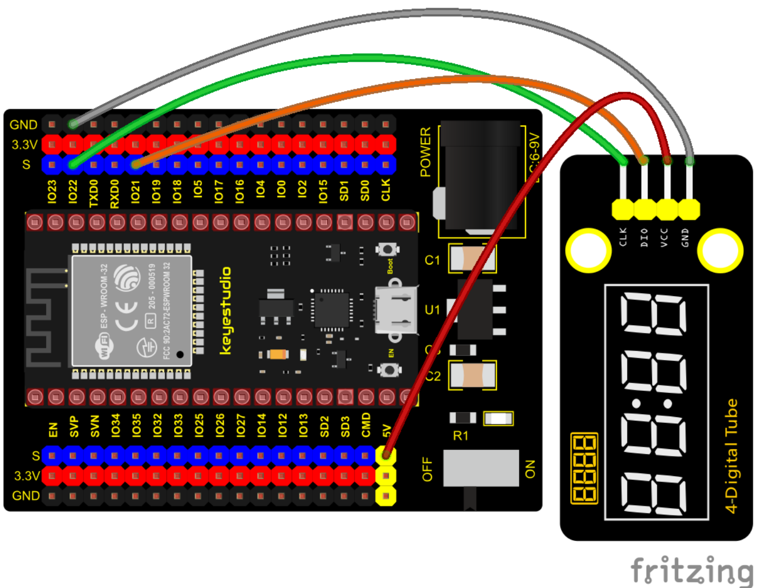
bit[3]: set to show decimal points

bit[0]: start the display of the tube display

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio TM16504-Digit Segment Display*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : TM1650 Four digital tube
 * Description : TM1650 Four Digital Tube shows 0-9999
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h"
#define CLK 22    //pins definitions for TM1650 and can be changed to other ports
#define DIO 21
TM1650 DigitalTube(CLK,DIO);

void setup(){
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
→default : 1
```

(continues on next page)

(continued from previous page)

```
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
}

void loop(){
  for(int num=0; num<10000; num++){
    displayFloatNum(num);
    delay(100);
  }
}

void displayFloatNum(float num){
  if(num > 9999)
    return;
  int dat = num*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The 4-digit tube display will show integer from 0 to 99999, add 1 for each 10ms. Increase to 9999 then start from 0.

6.2.42 Project 42: HT16K33_8X8 Dot Matrix Module

Overview

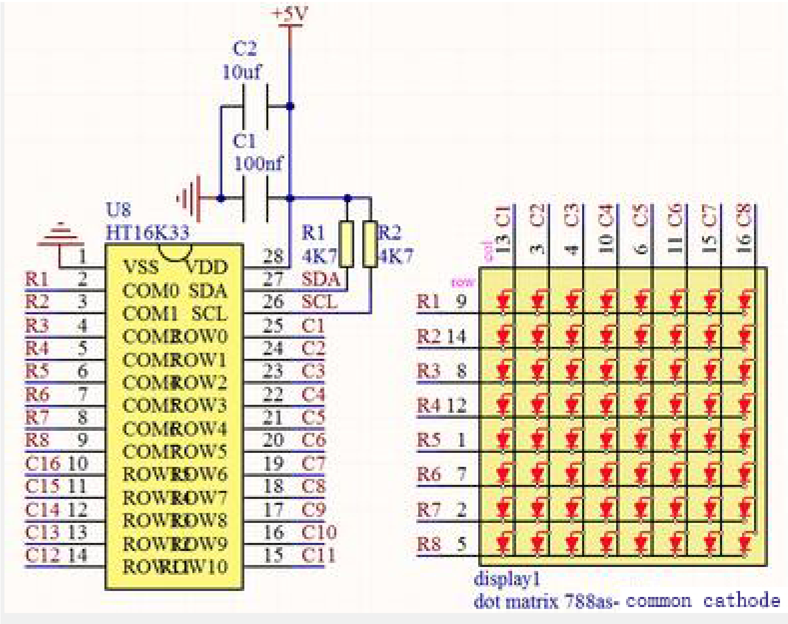
What is the dot matrix display? If we apply the previous circuit, there will be must one IO port to control only one LED. When more LED need to be controlled, we may adopt a dot matrix. The 8X8 dot matrix is composed of 64 light-emitting diodes, and each light-emitting diode is placed at the intersection of the row line and the column line. Refer to the experimental schematic diagram below, when the corresponding column is set to a high level and a certain row to low, the corresponding diode will light up. For instance, set pin 13 to a high level and pin 9 to low, and then the first LED will light up. In the experiment, we display icons via this dot matrix.

Working Principle

As the schematic diagram shown, to light up the LED at the first row and column, we only need to set C1 to high level and R1 to low level. To turn on LEDs at the first row, we set R1 to low level and C1-C8 to high level.

16 IO ports are needed, which will highly waste the MCU resources.

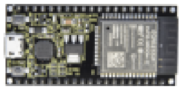
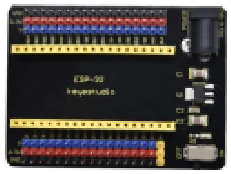
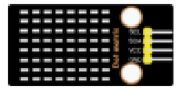
Therefore, we designed this module, using the HT16K33 chip to drive an 8*8 dot matrix, which greatly saves the resources of the single-chip microcomputer.



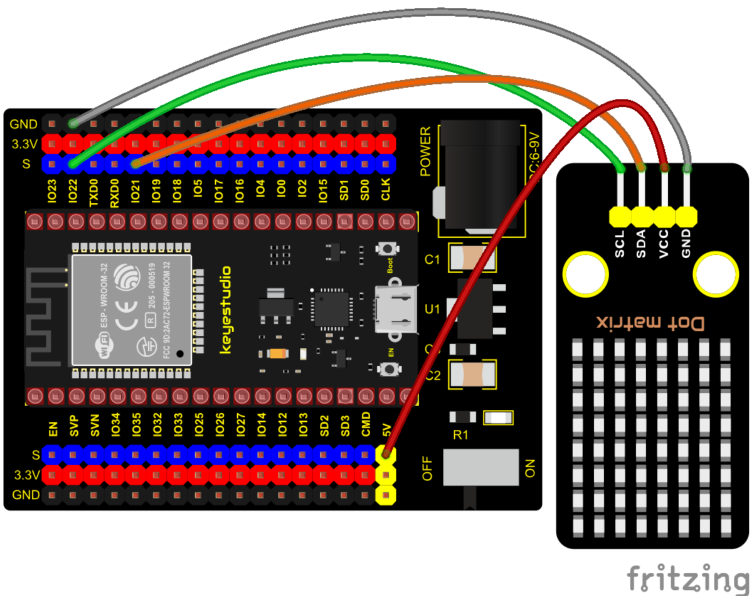
There are three DIP switches on the module, all of which are set to I2C communication address. The setting method is shown below. A0A1 and A2 are grounded, that is, the address is 0x70.

A0（1）	A1（2）	A2（3）	0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）
0X70			0X71			0X72		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
1（ON）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	1（ON）	0（OFF）	1（ON）
0X73			0X74			0X75		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）			
0（OFF）	1（ON）	1（ON）	1（ON）	1（ON）	1（ON）			

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HT16K33_ 8X8 Dot Matrix*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : 8×8 Dot-matrix Display
 * Description : 8x8 LED dot matrix display“Heart” pattern.
 * Auther      : http//www.keyestudio.com
*/
#include "HT16K33_Lib_For_ESP32.h"

#define SDA 21
#define SCL 22

ESP32_HT16K33 matrix = ESP32_HT16K33();

//The brightness values can be set from 1 to 15, with 1 darkest and 15 brightest
#define  A  15

```

(continues on next page)

(continued from previous page)

```
byte result[8][8];
byte test1[8] = {0x00,0x42,0x41,0x09,0x09,0x41,0x42,0x00};

void setup()
{
  matrix.init(0x70, SDA, SCL);//Initialize matrix
  matrix.showLedMatrix(test1,0,0);
  matrix.show();
}

void loop()
{
  for (int i = 0; i <= 7; i++)
  {
    matrix.setBrightness(i);
    delay(100);
  }
  for (int i = 7; i > 0; i--)
  {
    matrix.setBrightness(i);
    delay(100);
  }
}
//**************************************************************************************
```

Code Explanation

First we need to import the library file.

The pattern in our code is an array of byte data type, which is shown in the table below.

We convert {0x00,0x42,0x41,0x09,0x09,0x41,0x42,0x00} into binary, and fill in the 8*8 form below to make it clear. 1 means on, 0 means off. Then we can see that it is a smile shape.

0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Then the dot matrix displays a " smile "pattern.

6.2.43 Project 43: LCD_128X32_DOT Module

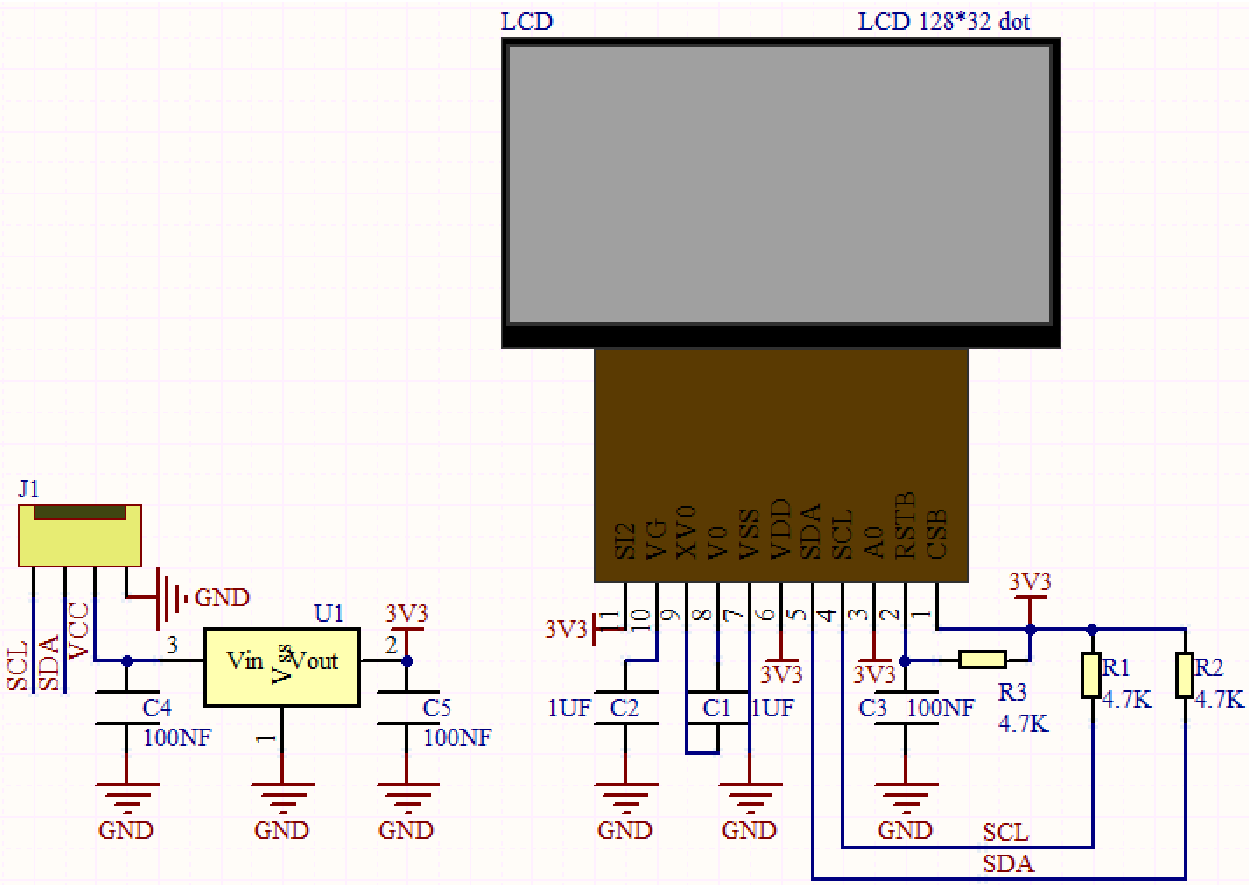
Description

This is a 128*32 pixel LCD module, which uses IIC communication mode and ST7567A driver chip . At the same time, the code contains all the English letters and common symbols of the library that can be directly called. When used, we can also set English letters and symbols to display different text sizes in our code.

To make it easy to set up the pattern display, we also provide a mold capture software that can convert a specific pattern into control code and then copy it directly into the test code for use.

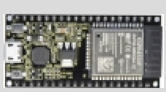
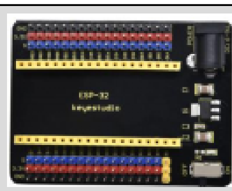
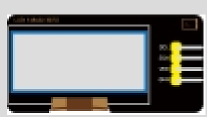
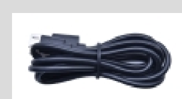
In the experiment, we will set up the display screen to display various English words, common symbols and numbers.

Working Principle

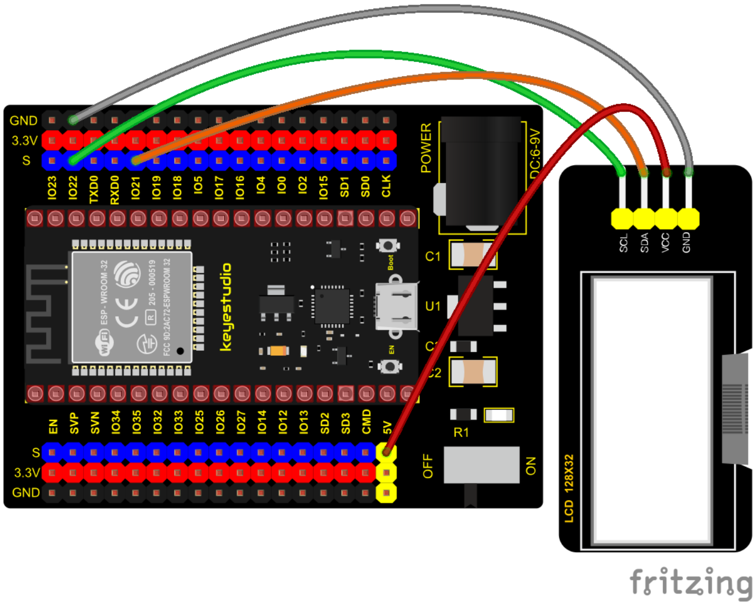


The module uses the IIC communication principle, the underlying functions have been encapsulated in the library surface, we can directly call the library function, if interested, you can also go to understand the underlying driver of the module.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio LCD_128X32_DOT Module*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : lcd128*32
 * Description : Lcd128 *32 Displays character strings
 * Auther      : http//www.keyestudio.com
*/
#include "lcd128_32_io.h"

//Create lcd12832 examples,sda--->21 scl--->22
lcd lcd(21, 22);

void setup() {
  lcd.Init(); //initialize
  lcd.Clear();  //cls
}

void loop() {
  lcd.Cursor(0, 7); //Set display position
  lcd.Display("KEYES"); //Setting the display
  lcd.Cursor(1, 0);
  lcd.Display("ABCDEFGHIJKLMNOPQR");
  lcd.Cursor(2, 0);
  lcd.Display("123456789+-*/<>=$@");
  lcd.Cursor(3, 0);
  lcd.Display("%^&(){}:;'|?,.~\\[]");
}
//**********************************************************************************
```

Code Explanation

First import the library file 1.Init() initializes the display screen; .Clear() clears the display; .Cursor() sets the display position; .Display() displays characters.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The first line of the 128X32LCD module display shows “KEYES”, the second line shows “ABCDEFGHIJKLMNOPQR”, and the third line shows “123456789±*/<> =$@”, the fourth line displays “%^&(){}:;’|?,.~\[]”, as shown in the following figure:

6.2.44 Project 44: RFID Module

Description

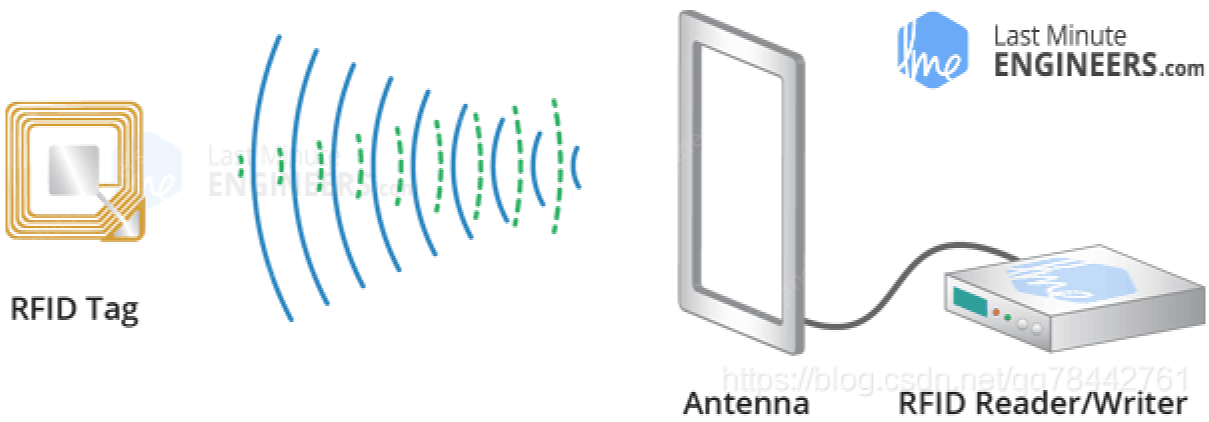
RFIDRFID-RC522 radio frequency module adopts a Philips MFRC522 original chip to design card reading circuit, easy to use and low cost, suitable for equipment development and card reader development and so on.

RFID or Radio Frequency Identification system consists of two main components, a transponder/tag attached to an object to be identified, and a transceiver also known as interrogator/Reader.

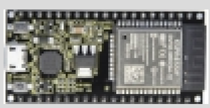
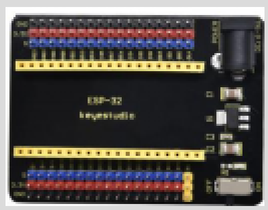
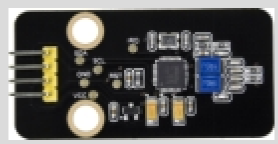
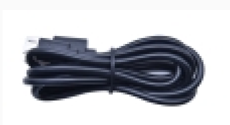
In the experiment, the data read by the card swipe module is 4 hexadecimal numbers, and we print these four hexadecimal numbers as strings. For example, we read the data of the IC card below: 0xED0xF70x940x5A and the information string displayed in the serial monitor is ED F7 94 5A ; the data read from the keychain is: 0x4C0x090x6B0x6E . Different IC cards and different key chains have diverse data.

Working Principle

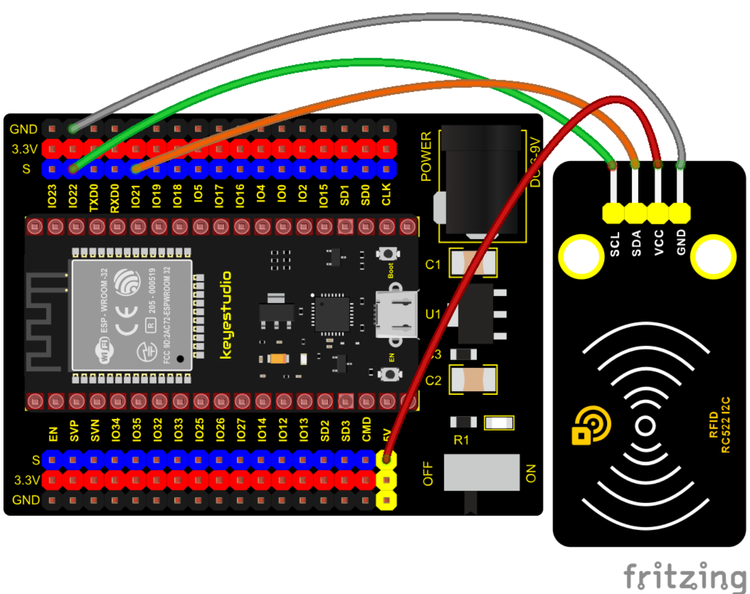
Radio frequency identification, the card reader is composed of a radio frequency module and a high-level magnetic field. The Tag transponder is a sensing device, and this device does not contain a battery. It only contains tiny integrated circuit chips and media for storing data and antennas for receiving and transmitting signals. To read the data in the tag, first put it into the reading range of the card reader. The reader will generate a magnetic field, and because the magnetic energy generates electricity according to Lenz’s law, the RFID tag will supply power, thereby activating the device.



Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY RFID Module*1	4P Dupont Wire*1
			
Micro USB Cable*1	Key*1	IC Card*1	

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : RFID
 * Description : RFID reader UID
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "MFRC522_I2C.h"
// IIC pins default to GPIO21 and GPIO22 of ESP32
// 0x28 is the i2c address of SDA, if doesn't matchplease check your address with i2c.
MFRC522 mfrc522(0x28);   // create MFRC522.

void setup() {
  Serial.begin(115200);           // initialize and PC's serial communication
  Wire.begin();                   // initialize I2C
  mfrc522.PCD_Init();             // initialize MFRC522
  ShowReaderDetails();            // dispaly PCD - MFRC522 read carder
  Serial.println(F("Scan PICC to see UID, type, and data blocks..."));
}

void loop() {
  //
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }

  // select one of door cards. UID and SAK are mfrc522.uid.

  // save UID
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println();
}

void ShowReaderDetails() {
  //  attain the MFRC522 software
  byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
  Serial.print(F("MFRC522 Software Version: 0x"));
  Serial.print(v, HEX);
  if (v == 0x91)
    Serial.print(F(" = v1.0"));
  else if (v == 0x92)
    Serial.print(F(" = v2.0"));
  else
    Serial.print(F(" (unknown)"));
  Serial.println("");
  // when returning to 0x00 or 0xFF, may fail to transmit communication signals
```

(continues on next page)

(continued from previous page)

```
  if ((v == 0x00) || (v == 0xFF)) {
    Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?
→"));
  }
}
//*************************************************************************************
```

Code Explanation

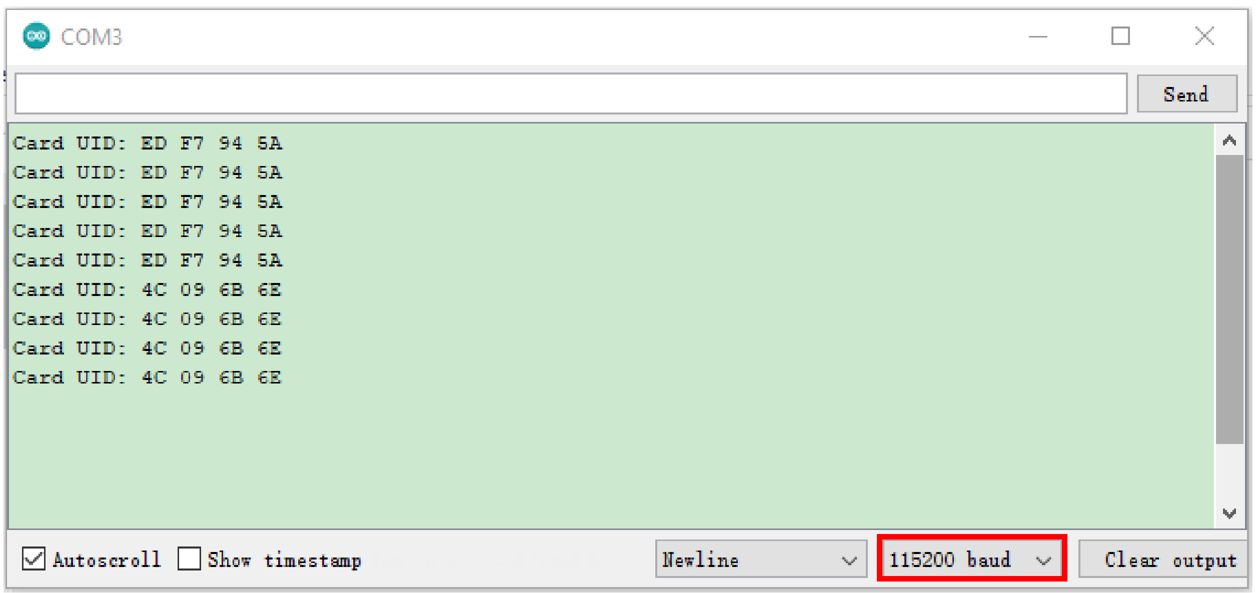
Wire.begin(); The module we use is the IIC interface, so we first initialize the IIC

mfrc522.PCD_Init(); initialize MFRC522

String(mfrc522.uid.uidByte[i], HEX); A string to convert the value read into hexadecimal format.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 115200. When we make the IC card close to the RFID module, the information will be printed out, as shown in the figure below.



Note: Different **RFID-RC522** door cards and key chains have diverse values.

6.3 3. Comprehensive Experiments:

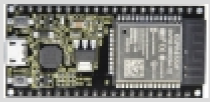
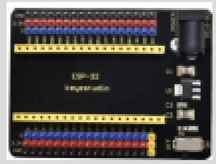
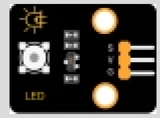
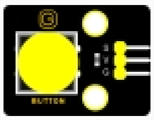
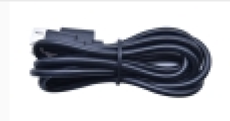
The previous projects are related to single sensor or module. In the following part, we will combine various sensors and modules to create some comprehensive experiments to perform special functions.

6.3.1 Project 45: Button-controlled LED

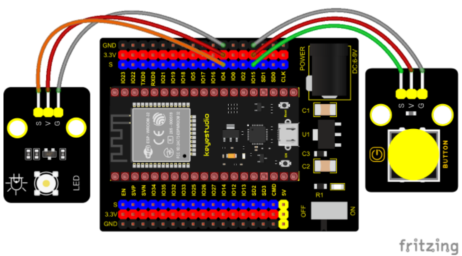
Overview

In this lesson, we will make an extension experiment with a button and an LED. When the button is pressed and low levels are output, the LED will light up; when the button is released, the LED will go off. Then we can control a module with another module.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1
		
Keyestudio DIY Button Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : button_control_LED
 * Description : Make a table lamp.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_LED    4
#define PIN_BUTTON 15
bool ledState = false;

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);
  }
}
```

(continues on next page)

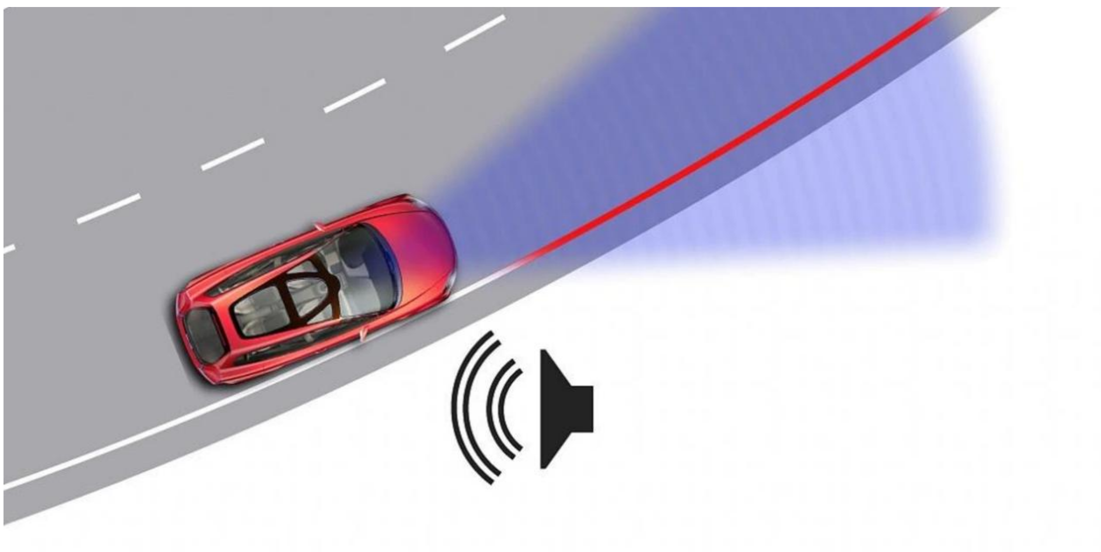
(continued from previous page)

```
void reverseGPIO(int pin) {
  ledState = !ledState;
  digitalWrite(pin, ledState);
}
//**********************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When the button is pressed, the LED will light up; when pressed again, the LED will go off.

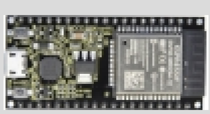
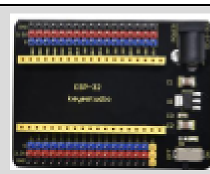
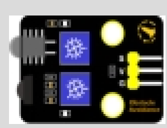
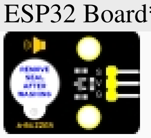
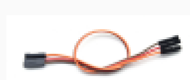
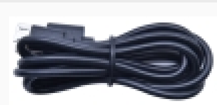
6.3.2 Project 46: Alarm Experiment



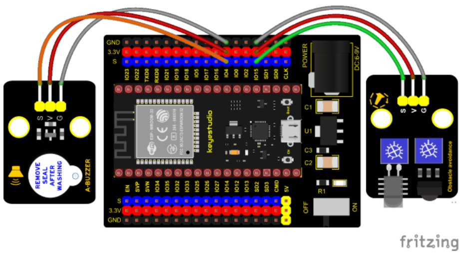
Overview

In the previous experiment, we control an output module though an input module. In this lesson, we will make an experiment that the active buzzer will emit sounds once an obstacle appears.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Obstacle Avoidance Sensor*1
		
Keyestudio Active Buzzer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Avoiding alarm
 * Description : Obstacle avoidance sensor controls the buzzer
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
  pinMode(15, INPUT);  //Obstacle avoidance sensor is connected to GPIO15 and set to␣
↪input mode
  pinMode(4, OUTPUT); //The buzzer is connected to GPIO4 and set to output mode
}

void loop() {
  item = digitalRead(15);//Read the level value output by the obstacle avoidance sensor
  if (item == 0) {//Obstruction detected
    digitalWrite(4, HIGH);//The buzzer sounded
  } else {//No obstacles detected
    digitalWrite(4, LOW);//The buzzer is off
  }
  delay(100);//Delay 100ms
}
//**********************************************************************************
```

Code Explanation

Set IO ports according to connection diagram then configure pins mode.

The value is 0 when pressing the button, So, we can determine the key value(0through if (item == 0) and make the buzzer beep digitalWrite(4, HIGH).

Test Result

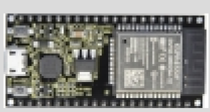
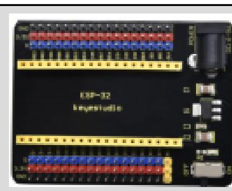
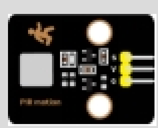
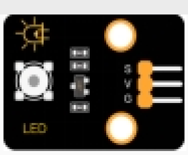
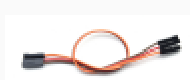
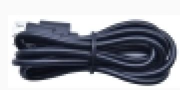
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. If the obstacle is detected, the active buzzer will chime; if not, it won't beep.

6.3.3 Project 47: Intrusion Detection

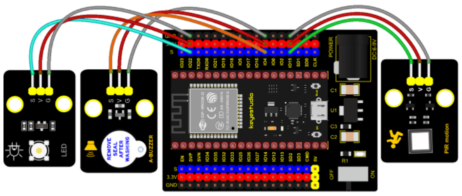
Description

In this experiment, we use a PIR motion sensor to control an active buzzer to emit sounds and the onboard LED to flash rapidly.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY PIR Motion Sensor*1	Keyestudio DIY Active Buzzer*1
			
Keyestudio Purple LED Module*1	3P Dupont Wire*3	Micro USB Cable*1	

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : PIR alarm
 * Description : PIR control buzzer
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
  pinMode(15, INPUT);  //PIR motion sensor is connected to GPIO15 and set as the input mode
  pinMode(4, OUTPUT);//The active buzzer is connected to GPIO4 and set to output mode
  pinMode(22, OUTPUT);//LED is connected to GPIO22 and set to output mode
}

void loop() {
  item = digitalRead(15);//Read digital level signal output by infrared pyrorelease sensor
  if (item == 1) {  //Movement detected
    digitalWrite(4, HIGH); //Turn on the buzzer
    digitalWrite(22, HIGH); //Turn on the LED
    delay(200);//Delay 200ms
    digitalWrite(4, LOW); //Turn off the buzzer
    digitalWrite(22, LOW); //Turn off the LED
    delay(200);//Delay 200ms
  } else {  //No detection
    digitalWrite(4, LOW); //Turn off the buzzer
    digitalWrite(22, LOW); //Turn off the LED
  }
}
//*************************************************************************************
```

Test Result

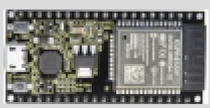
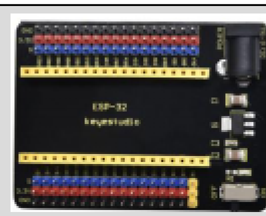
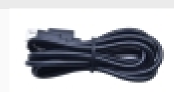
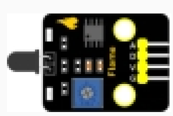
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. If the sensor detects people moving, the buzzer will emit an alarm, and the LED will flash continuously.

6.3.4 Project 48: Extinguishing Robot

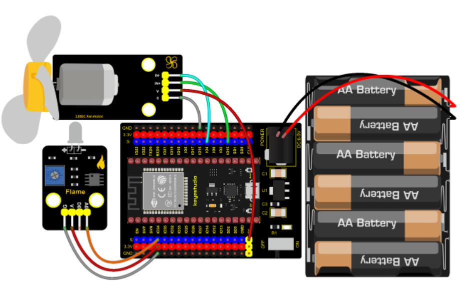
Description

Today we will use Arduino simulation to build an extinguishing robot that will automatically sense the fire and start the fan. In this project we will learn how to build a very simple robot using ESP32, (detecting flames with a flame sensor, blowing out candles with a fan) can teach us basic concepts about robotics. Once you understand the basics below, you can build more complex robots.

Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	130 Motor*1
		
3P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1
		
Battery(provide for yourself)*6	Flame Sensor*1	Battery Holder*1

Connection Diagram



Test Code

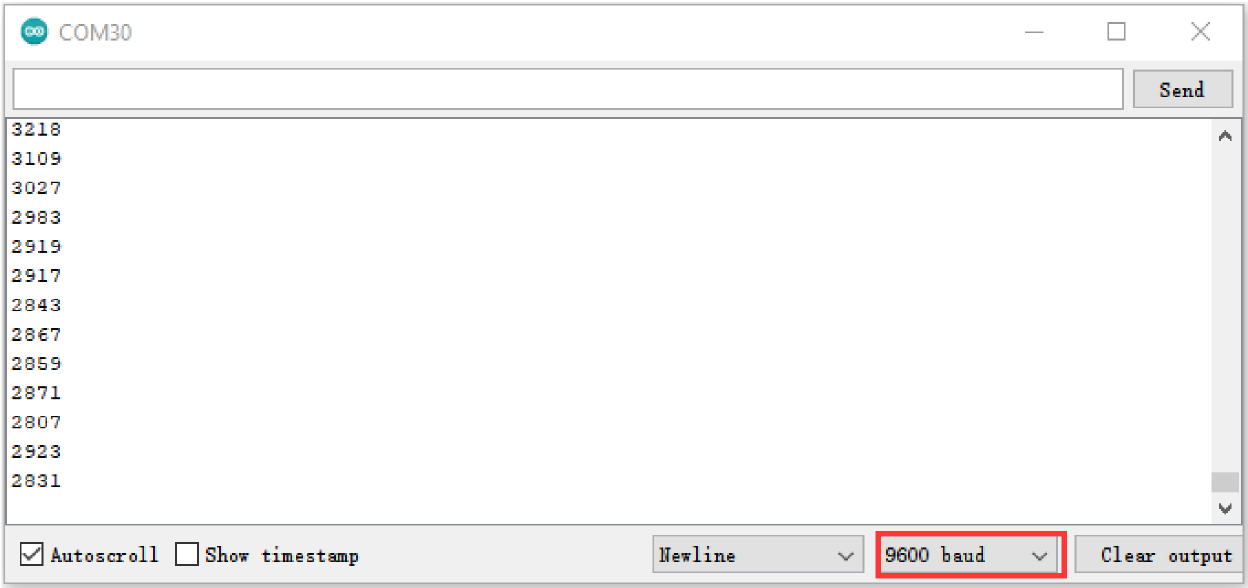
```
//*************************************************************************************
/*
 * Filename    : Fire-fighting robot
 * Description : Flame sensor controls the 130 fan module
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
  Serial.begin(9600);
  pinMode(15, OUTPUT);//INA corresponds to IN+, and sets GPIO15 to output mode
  pinMode(4, OUTPUT);//INB corresponds to IN-, and sets GPIO4 to output mode
}

void loop() {
  item = analogRead(34);//The flame sensor is connected to GPIO34, and read the␣
↪simulated value to Item
  Serial.println(item);//Serial port display analog value
  if (item < 3000) {//Less than 3000
    digitalWrite(15, LOW);//Turn on the fan
    digitalWrite(4, HIGH);
  } else {//Otherwise, turn off the fan.
    digitalWrite(15, LOW);
    digitalWrite(4, LOW);
  }
  delay(100);
}
//*************************************************************************************
```

Code Explanation

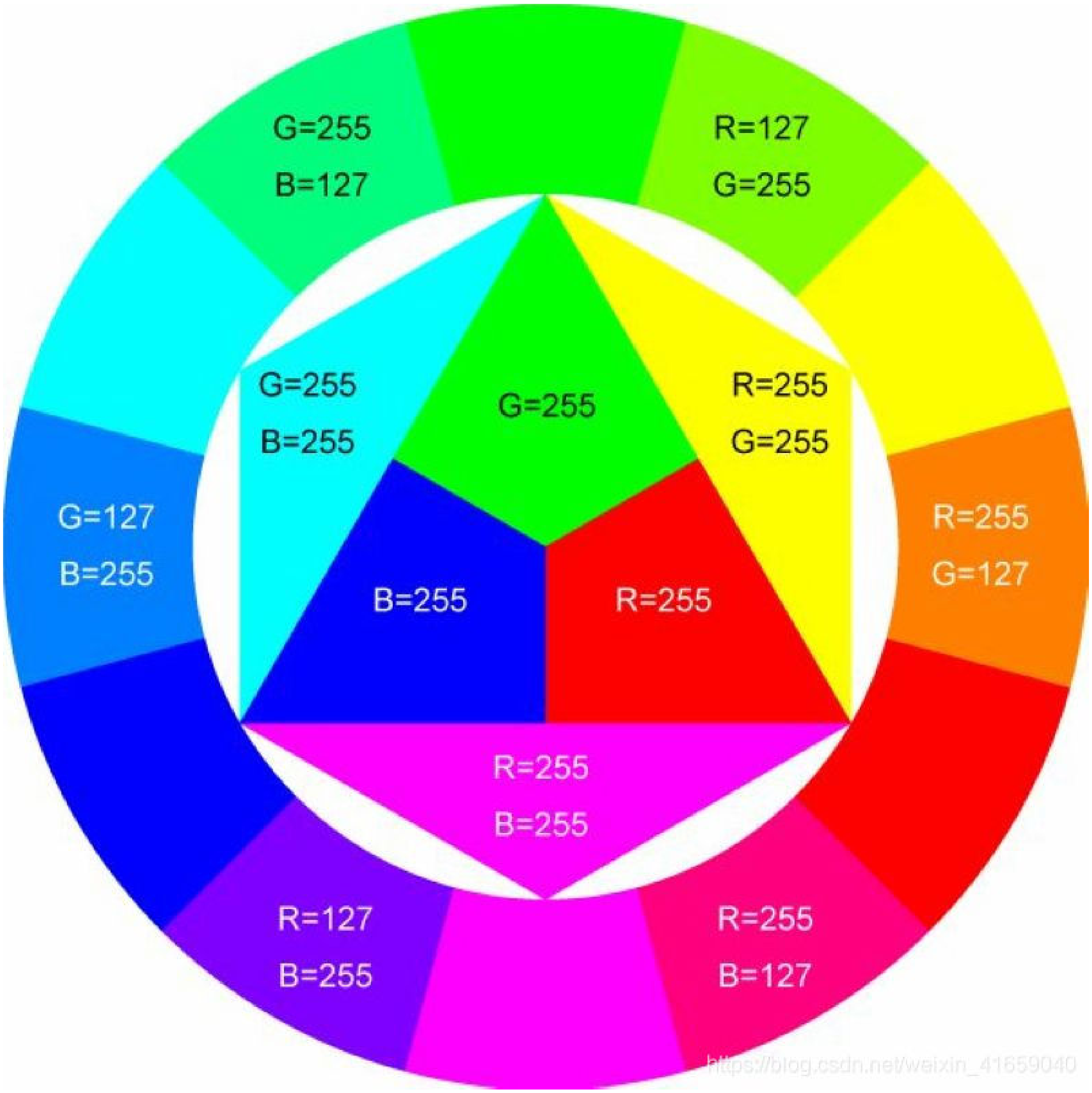
In the code, we set the threshold value to 3000. When the ADC value detected by the flame sensor is lower than the threshold value, the fan will be automatically turned on; otherwise, it will be turned off. For the driving method of the fan, please refer to the 130 Motor.

Test Result

Connect the wires according to the experimental wiring diagram, switch the DIP switch on the ESP32 expansion board to the ON end and power up, compile and upload the code to the ESP32. After uploading successfully, open the serial monitor and set baud rate to 9600, then the ADC value of the flame will be printed. When this value is less than 3000, the fan will work to blow out the fire, otherwise, it will be turned off. Basically, the ADC value can be set by yourself.


COM30
Send
3218
3109
3027
2983
2919
2917
2843
2867
2859
2871
2807
2923
2831
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

6.3.5 Project 49: Rotary Encoder control RGB

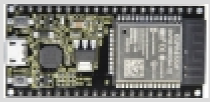
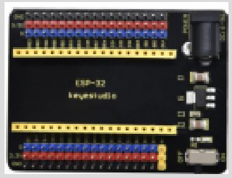
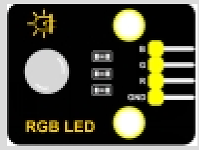
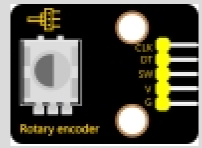
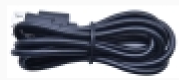


Introduction

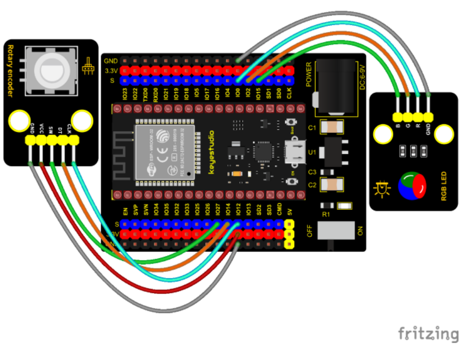
In this lesson, we will control the LED on the RGB module to show different colors through a rotary encoder.

When designing the code, we need to divide the obtained values by 3 to get the remainders. The remainder is 0 and the LED will become red. The remainder is 1, the LED will become green. The remainder is 2, the LED will turn blue.

Components

			
ESP32Board*1	ESP32 Expansion Board*1	KeyestudioCommon Cathode RGB Module*1	KeyestudioRotary Encoder Module*1
			
5P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Encoder control RGB
 * Description : Rotary encoder controls RGB to present different effects
 * Auther      : http//www.keyestudio.com
*/
//Interfacing Rotary Encoder with Arduino
//Encoder Switch -> pin 27
//Encoder DT -> pin 14
//Encoder CLK -> pin 12
int Encoder_DT  = 14;
int Encoder_CLK  = 12;
int Encoder_Switch = 27;

int Previous_Output;
int Encoder_Count;

int ledPins[] = {0, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;

int val;
```

(continues on next page)

(continued from previous page)

```
void setup() {
  Serial.begin(9600);

  //pin Mode declaration
  pinMode (Encoder_DT, INPUT);
  pinMode (Encoder_CLK, INPUT);
  pinMode (Encoder_Switch, INPUT);

  Previous_Output = digitalRead(Encoder_DT); //Read the inital value of Output A
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
   }
}

void loop() {
  //aVal = digitalRead(pinA);

  if (digitalRead(Encoder_DT) != Previous_Output)
  {
    if (digitalRead(Encoder_CLK) != Previous_Output)
    {
      Encoder_Count ++;
      Serial.print(Encoder_Count);
      Serial.print("  ");
      val = Encoder_Count % 3;
      Serial.println(val);
    }
    else
    {
      Encoder_Count--;
      Serial.print(Encoder_Count);
      Serial.print("  ");
      val = Encoder_Count % 3;
      Serial.println(val);
    }
  }

  Previous_Output = digitalRead(Encoder_DT);

  if (digitalRead(Encoder_Switch) == 0)
  {
    delay(5);
    if (digitalRead(Encoder_Switch) == 0) {
      Serial.println("Switch pressed");
      while (digitalRead(Encoder_Switch) == 0);
    }
  }
  if (val == 0) {
    //RED(255, 0, 0)
    ledcWrite(chns[0], 255 );
    ledcWrite(chns[1], 0);
```

(continues on next page)

(continued from previous page)

```
    ledcWrite(chns[2], 0);
  } else if (val == 1) {
    //GREEN(0, 255, 0)
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 255);
    ledcWrite(chns[2], 0);
  } else {
    //BLUE(0, 0, 255)
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 255);
  }
}
//*************************************************************************************
```

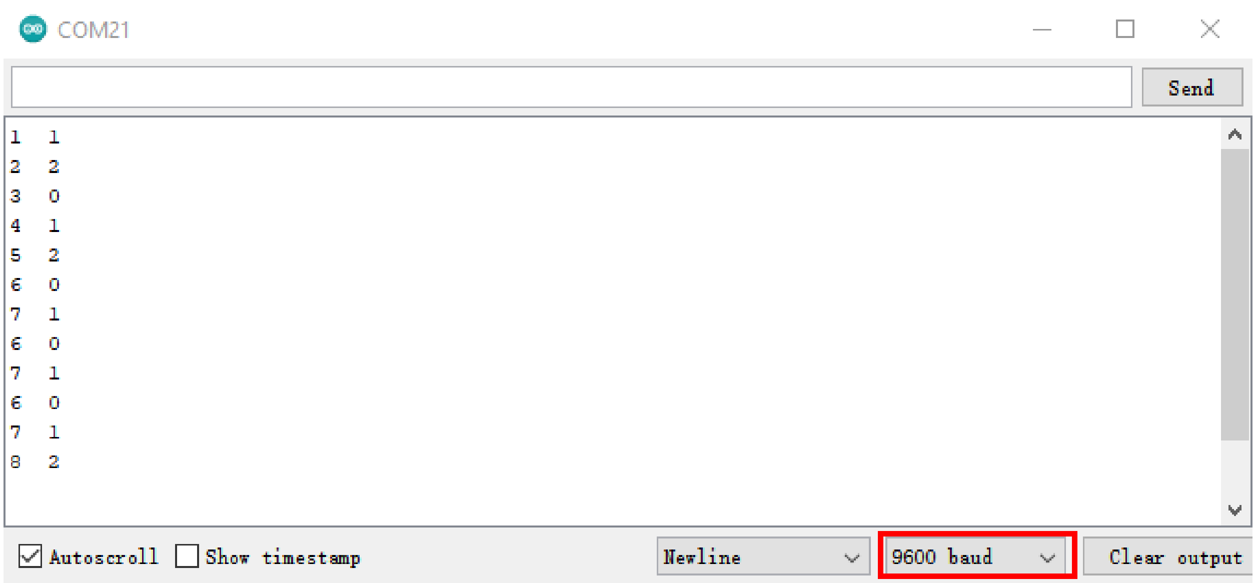
Code Explanation

1). In the experiment, we set the val to the remainder of Encoder_Count divided by 3. Encoder_Count is the value of the encoder. Then we can set pin GPIO0 (red), GPIO2 (green) and GPIO15 (blue) according to remainders.

2). Referring to the control method learned in the previous experiment, use the LED on the remainder control module to display the corresponding light color. The value obtained by taking the remainder of 3 for any number is 0 or 1 or 2. We use these three values to judge, and display the corresponding color.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600, then rotate the knob of the rotary encoder to display the reminders, which can control colors of LED(red green blue).



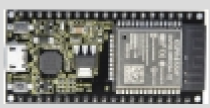
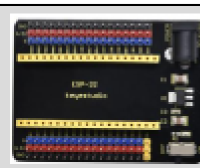
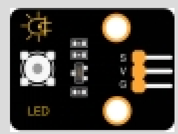
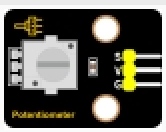
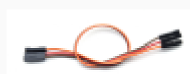
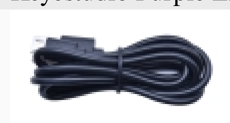
6.3.6 Project 50: Rotary Potentiometer

Introduction

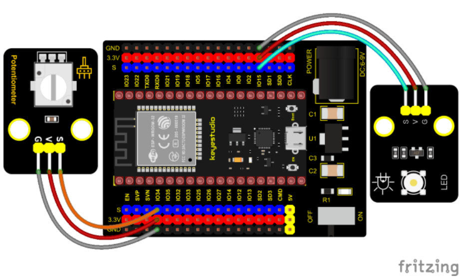
In the previous courses, we did experiments of breathing light and controlling LED with button. In this course, we do these two experiments by controlling the brightness of LED through an adjustable potentiometer. The brightness of LED is controlled by PWM values, and the range of analog values is 0 to 4095 and the PWM value range is 0-255.

After the code is set successfully, we can control the brightness of the LED on the module by rotating the potentiometer.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED*1
		
Keyestudio Rotary Potentiometer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : adjust the light
 * Description : Controlling the brightness of LED by potentiometer.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN    34  //the pin of the potentiometer
#define PIN_LED      15  // the pin of the LED
```

(continues on next page)

(continued from previous page)

```
#define CHAN            0
void setup() {
  ledcSetup(CHAN, 1000, 12);
  ledcAttachPin(PIN_LED, CHAN);
}

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

Code Explanation

In the experiment, the mapping function maps adcVal from the range of 0-4095 to 0-255, and assigns it to pwmVal.

Test Result

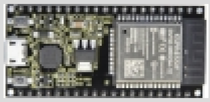
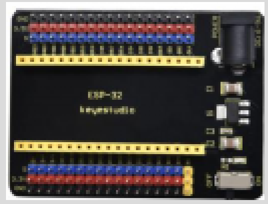
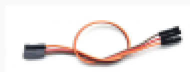
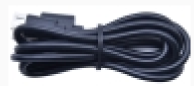
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the module can adjust the brightness of the LED on the LED module.

6.3.7 Project 51: Smart Windows

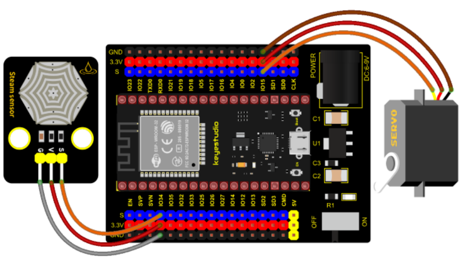
Description

In life, we can see all kinds of smart products, such as smart home. Smart homes include smart curtains, smart windows, smart TVs, smart lights, and more. In this experiment, we use a steam sensor to detect rainwater, and then achieve the effect of closing and opening the window by a servo.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Steam Sensor*1
		
Servo*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : smart window
 * Description : Water drop sensor controls steering gear rotation.
 * Auther      : http//www.keyestudio.com
*/
#include <ESP32Servo.h>//Import the steering gear library file
int adcVal = 0;//A variable that holds the ADC value output by the droplet sensor
int servoPin = 15;  // Define the servo pin
Servo myservo;//Defines an instance of the steering gear class

#define PIN_ADC    34  //the pin of the Water drop sensor

void setup(){
  Serial.begin(9600);
  pinMode(PIN_ADC, INPUT);
  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo object
}

void loop(){
```

(continues on next page)

(continued from previous page)

```
  adcVal = analogRead(PIN_ADC);//The droplet sensor is connected to the analog port GP34
  Serial.println(adcVal);
  if (adcVal > 2000) {//The simulated value is greater than 2000
    myservo.write(0);//close the window
    delay(500);//Give the steering gear time to turn
  } else {// no rain
    myservo.write(180);//open the window
    delay(500);//Delay 500ms
  }
}
//*************************************************************************************
```

Code Explanation

We can control a servo to rotate by a threshold.

Test Result

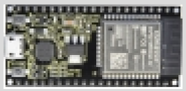
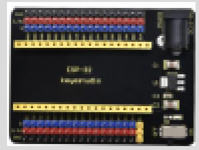
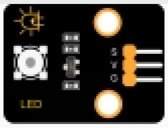
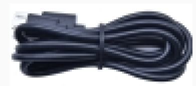
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When the sensor detects a certain amount of water, the servo rotates to achieve the effect of closing or opening windows.

6.3.8 Project 52: Sound Activated Light

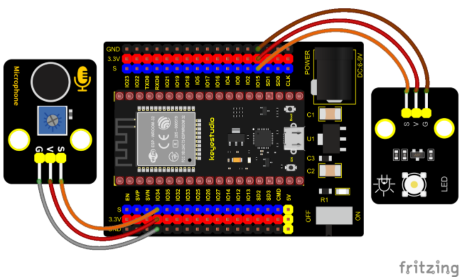
Introduction

In this lesson, we will make a smart sound activated light using a sound sensor and an LED module. When we make a sound, the light will automatically turn on; when there is no sound, the light will automatically turn off. How it works? Because the sound-controlled light is equipped with a sound sensor, and this sensor converts the intensity of external sound into a corresponding value. Then set a threshold, when the threshold is exceeded, the light will go on, and when it is not exceeded, the light will go off.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Sound Sensor*1
		
Keyestudio Purple LED Module*1	3P Dupont Wire*2	MicroUSBCable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : sound-controlled lights
 * Description : Sound sensor controls LED on and off
 * Auther      : http//www.keyestudio.com
*/
int ledPin = 15;//LED is connected to GP15
int microPin = 34;//Sound sensor is connected to GPIO34
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(ledPin, OUTPUT);//LED is the output mode
}

void loop() {
  int val = analogRead(microPin);//Read analog value
  Serial.print(val);// Serial port print
  if(val > 600){//exceed the threshold value
    digitalWrite(ledPin, HIGH);//Lighting LED 3sand print the corresponding information
    Serial.println("  led on");
    delay(3000);
  }else{//otherwise
    digitalWrite(ledPin, LOW);//Turn off the LED and print the corresponding information
    Serial.println("  led off");
  }
  delay(100);
}
//**********************************************************************************
```

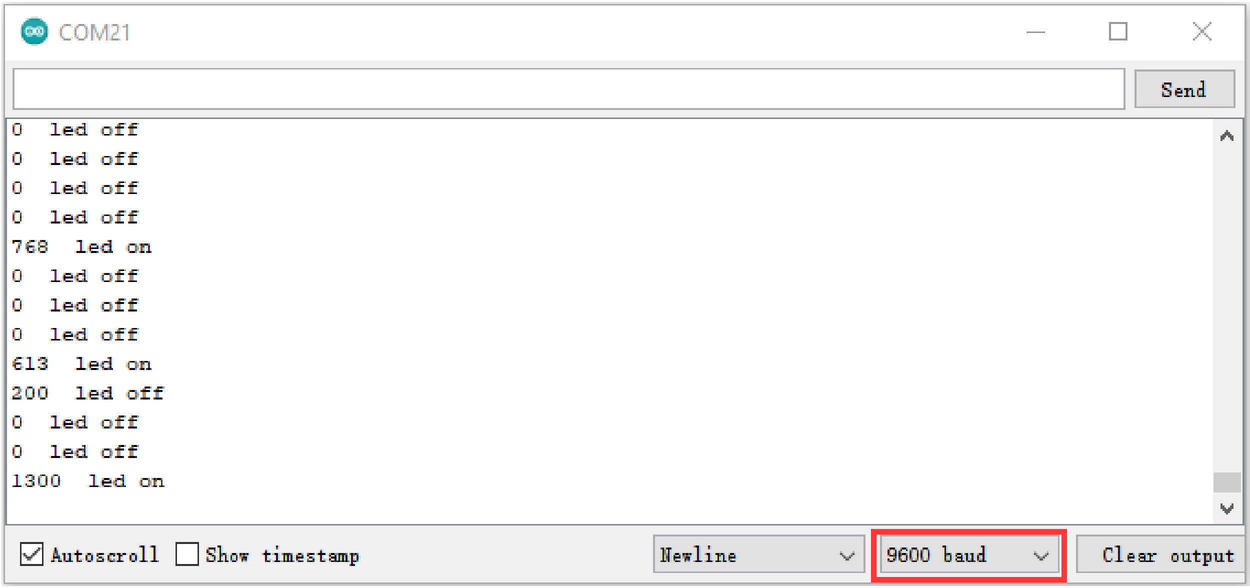
Code Explanation

We set the ADC threshold value to 600. If more than 600, LED will be on 3s; on the contrary, it will be off.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600, then the corresponding volume ADC value will be displayed.

When the analog value of sound is greater than 600, the LED on the LED module will light up 3s, otherwise it will go off.

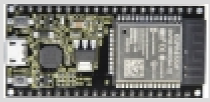
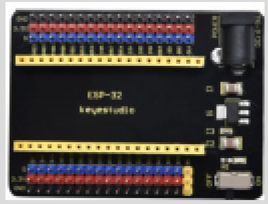
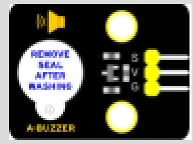
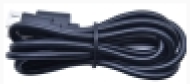
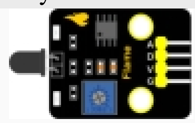


6.3.9 Project 53: Fire Alarm

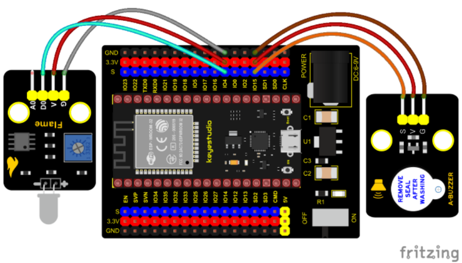
Description

In this experiment, we will make a fire alarm system. Just use a flame sensor to control an active buzzer to emit sounds.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIYActive Buzzer*1
		
Micro USB Cable*1	3P Dupont Wire*2	keyestudio DIY Flame Sensor*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Flame Alarm
 * Description : Controlling the buzzer by flame sensor.
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
  Serial.begin(9600);
  pinMode(4, INPUT);//Flame sensor digital pin is connected to GPIO4
  pinMode(15, OUTPUT);//Buzzer pin is connected to GPIO15
}

void loop() {
  item = digitalRead(4);//Read the digital level output by the flame sensor
  Serial.println(item);//Newline print level signal
  if (item == 0) {//Flame detected
    digitalWrite(15, HIGH);//Turn on the buzzer
  } else {//Otherwise, turn off the buzzer
    digitalWrite(15, LOW);
  }
  delay(100);//Delay 100ms
}
//**********************************************************************************
```

Code Explanation

This flame sensor uses an analog pin and a digital pin. When a flame is detected, the digital pin outputs a low level. In this experiment we will use the digital port.

Test Result

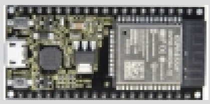
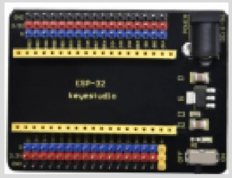
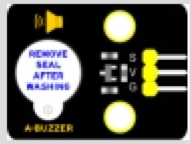
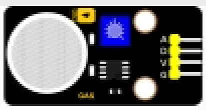
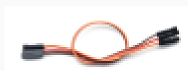
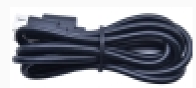
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When the sensor detects the flame, the external active buzzer will emit sounds, otherwise the active buzzer will not emit sounds.

6.3.10 Project 54: Smoke Alarm

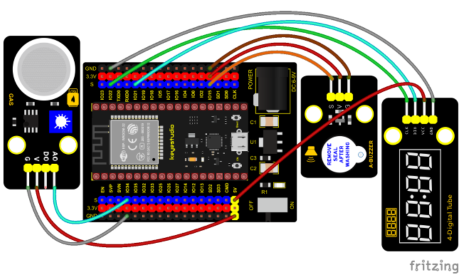
Description

In this experiment, we will make a smoke alarm by a TM16504-Digit segment module, a gas sensor and an active buzzer.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1	Keyestudio TM16504-Digit Segment Module*1
			
keyestudio Analog Gas Senso*1	3P Dupont Wire*2	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : smoke alarm
 * Description : MQ2 controls a buzzer and a four-digit analog smoke tester
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
int adcVal = 0; //display ADC value
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on, default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
```

(continues on next page)

(continued from previous page)

```
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(15, OUTPUT);//the buzzer is connected to GPIO15
}

void loop() {
  adcVal = analogRead(34);//Read the ADC values of MQ2
  displayFloatNum(adcVal);;//Four digit tube display adcVal values
  if (adcVal > 1000) {//ADC value is greater than 1000
    digitalWrite(15, HIGH); // buzzer alarming
  } else {//or else
    digitalWrite(15, LOW); //Turn off the buzzer
  }
  delay(100);//delay 100ms
}

void displayFloatNum(float adcVal){
  if(adcVal > 9999)
    return;
  int dat = adcVal*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Code Explanation

Define an integer variable val to store the analog value of the smoke sensor, and then we display the analog value in the four-digit digital tube, and then set a threshold, and when the threshold is reached, the buzzer will sound.

Test Result

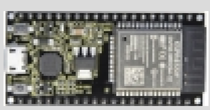
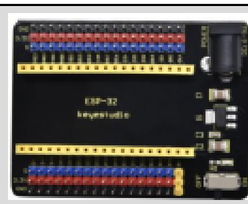
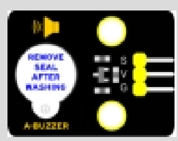
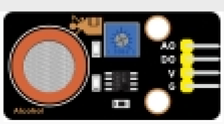
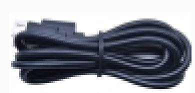
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When the concentration of combustible gas exceeds the standard, the active buzzer module will give an alarm, and the four-digit digital tube will display the concentration value.

6.3.11 Project 55: Alcohol Sensor

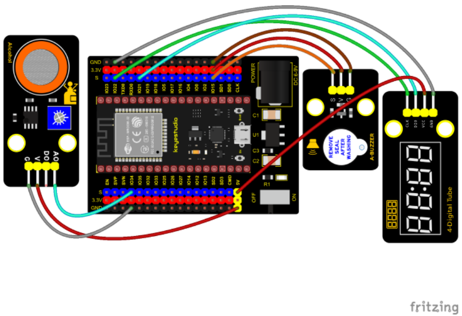
Description

In the last experiment, we made a smoke alarm. In this experiment, we combine the active buzzer, the MQ-3 alcohol sensor, and a four-digit digital tube to test the alcohol concentration through the alcohol sensor. Then, the concentration to control the active buzzer alarm and the four-digit digital tube to display the concentration. So as to achieve the simulation effect of alcohol detector.

Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Active Buzzer*1	Keyestudio DIY TM1650 4-Digit Tube Display*1
			
keyestudio Alcohol Sensor*1	3P Dupont Wire*2	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : breathalyzer
 * Description : MQ3 controls a buzzer and a four-digit tube to simulate a breathalyzer.
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
int adcVal = 0; //display ADC value
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(15, OUTPUT);//the buzzer is connected to GPIO15
}

void loop() {
  adcVal = analogRead(34);//Read the ADC values of MQ3
  displayFloatNum(adcVal);//Four digit tube display adcVal values
  if (adcVal > 1000) {//ADC value is greater than 1000
    digitalWrite(15, HIGH); // buzzer alarming
  } else {//or else
    digitalWrite(15, LOW); //Turn off the buzzer
  }
  delay(100);//delay 100ms
}

void displayFloatNum(float adcVal){
  if(adcVal > 9999)
    return;
  int dat = adcVal*10;
```

(continues on next page)

(continued from previous page)

```
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//**************************************************************************************
```

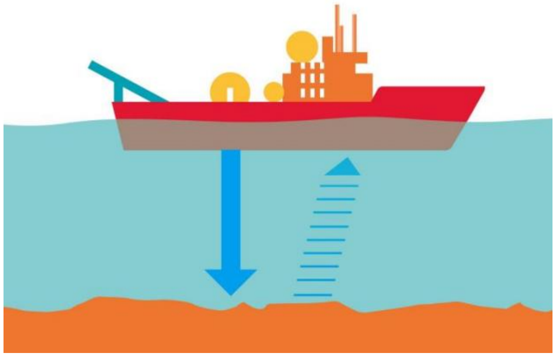
Code Explanation

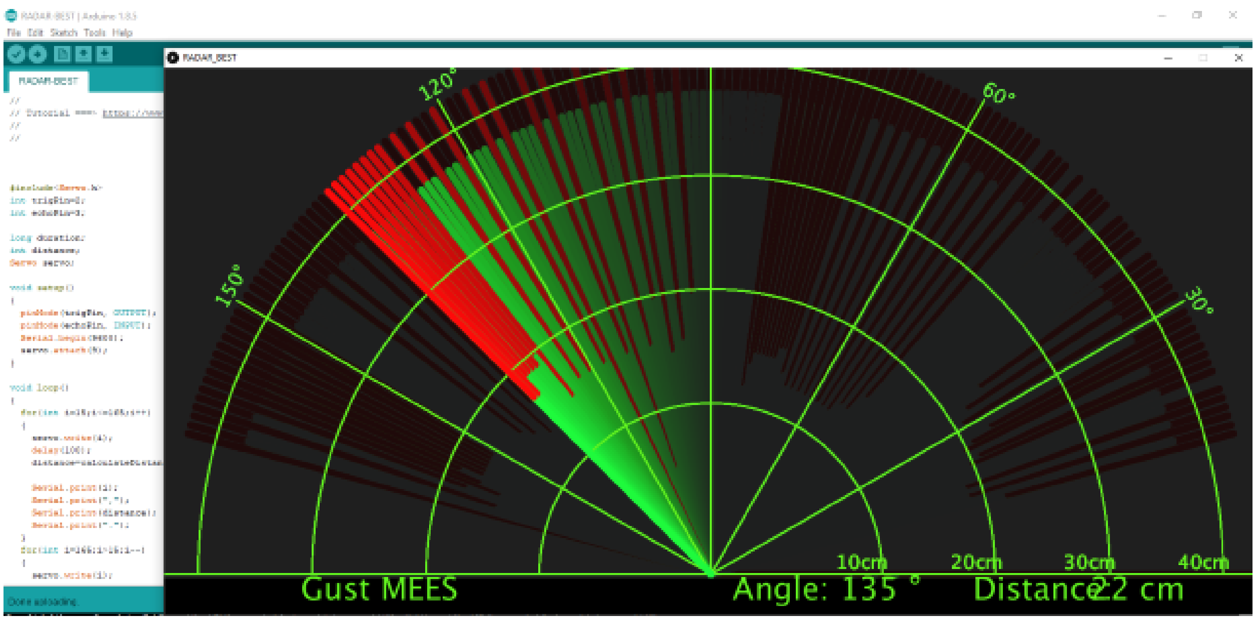
Define an integer variable val to store the ADC value of the alcohol sensor, then we display the analog value in the four-digit display module and set a threshold.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When different alcohol concentrations are detected, the active buzzer module will alarm, and the four-digit digital display will show the concentration value.

6.3.12 Project 56: Ultrasonic Radar



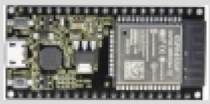
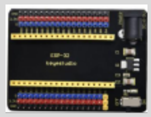
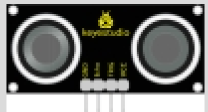
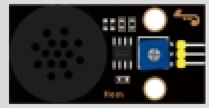
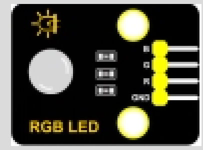
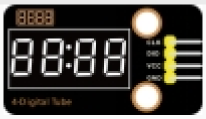
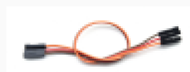
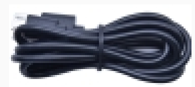


Description

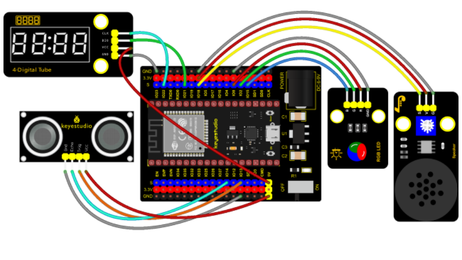
We know that bats use echoes to determine the direction and the location of their preys. In real life, sonar is used to detect sounds in the water. Since the attenuation rate of electromagnetic waves in water is very high, it cannot be used to detect signals, however, the attenuation rate of sound waves in the water is much smaller, so sound waves are most commonly used underwater for observation and measurement.

In this experiment, we will use a speaker module, an RGB module and a 4-digit tube display to make a device for detection through ultrasonic.

Required Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HC-SR04 Ultrasonic Sensor*1	Keyestudio 8002b Power Amplifier*1	Keyestudio DIY Common Cathode RGB Module *1
				
Keyestudio DIY TM1650 4-Digit Tube Display*1	4P Dupont Wire*3	3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Ultrasonic radar
 * Description : Ultrasonic control four digit tube, buzzer and RGB analog ultrasonic radar.
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

int beeppin = 18; //Define the horn pin as GPIO18

int TrigPin = 13; //Set the Trig pin to GPIO13
int EchoPin = 14; //Set the Echo pin to GPIO14
int distance;//Distance measured by ultrasound

int ledPins[] = {0, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels

```

(continues on next page)

(continued from previous page)

```
float checkdistance() { //get distance
  // A short low level is given beforehand to ensure a clean high pulse:
  digitalWrite(TrigPin, LOW);
  delayMicroseconds(2);
  // The sensor is triggered by a high pulse of 10 microseconds or more
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);
  // Read the signal from the sensor: a high level pulse
  //Its duration is the time (in microseconds) from sending the ping command to␣
↪receiving the echo from the object
  float distance = pulseIn(EchoPin, HIGH) / 58.00;  //Convert to distance
  delay(10);
  return distance;
}

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(TrigPin, OUTPUT);//Sets the Trig pin as output
  pinMode(EchoPin, INPUT);  //Set the Echo pin as input
  ledcSetup(3, 1000, 8);//setup the pwm channels,1KHz,8bit
  ledcAttachPin(18, 3);
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
  }
}

void loop() {
  distance = checkdistance(); //Ultrasonic ranging
  displayFloatNum(distance);  //Nixie tube shows distance
  if (distance <= 10) {
    ledcWrite(3, 100);
    delay(100);
    ledcWrite(3, 0);
    ledcWrite(chns[0], 255); //Common cathode LED, high level to turn on the led.
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 0);

  } else if (distance > 10 && distance <= 20) {
    ledcWrite(3, 200);
    delay(200);
    ledcWrite(3, 150);
    ledcWrite(chns[0], 0);
```

(continues on next page)

(continued from previous page)

```
    ledcWrite(chns[1], 255);
    ledcWrite(chns[2], 0);
  } else {
    ledcWrite(3, 0);
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 255);
  }
}

void displayFloatNum(float distance){
  if(distance > 9999)
    return;
  int dat = distance*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*********************************************************************************
```

Code Explanation

We set sound frequency and light color by adjusting different distance range.

We can adjust the distance range in the code.

Test Result

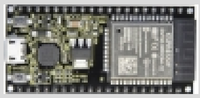
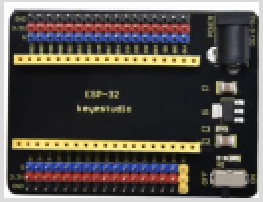
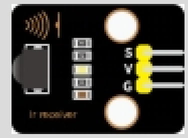
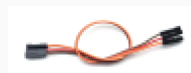
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. When the ultrasonic sensor detects different distances(within 20 cm), the buzzer will produce different frequencies of sound, the RGB will show different colors, and the measured distances are displayed on the 4-digit tube display.

6.3.13 Project 57: IR Remote Control

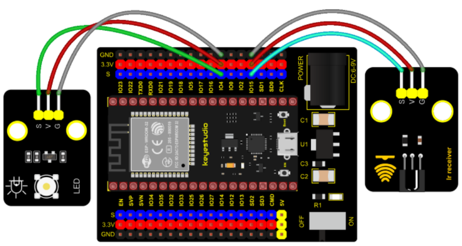
Introduction

In the previous experiments, we learned how to turn on/off the LED and adjust its brightness via PWM and print the button value of the IR remote control in the serial monitor window. Herein, we use an infrared remote control to turn on/off an LED.

Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Purple LED Module*1	Keyestudio DIY IR Receiver*1
			
Micro USB Cable*1	IR Remote Control*1	3P Dupont Wire*2	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : IR Control LED
 * Description : Remote controls LED on and off
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

const uint16_t recvPin = 15; // Infrared receiving pin 15
IRrecv irrecv(recvPin);      // Create a class object used to receive class
decode_results results;       // Create a decoding results class object
int led = 4;//LED connect to GP4

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();                    // Start the receiver
  pinMode(led, OUTPUT);
}
////////////////////
```

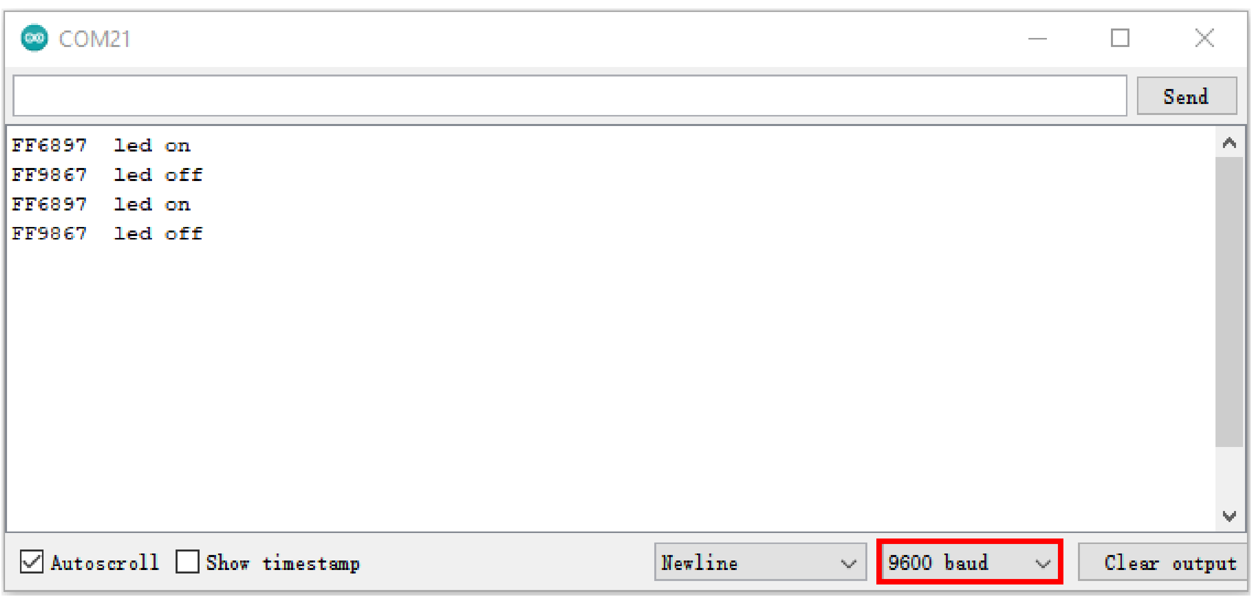
(continues on next page)

(continued from previous page)

```
void loop() {
  if(irrecv.decode(&results)) {         // Waiting for decoding
    serialPrintUint64(results.value, HEX);// Print out the decoded results
    Serial.print("");
    handleControl(results.value);       // Handle the commands from remote control
    irrecv.resume();                    // Receive the next value
  }
}
  void handleControl(unsigned long value) {
    if (value == 0xFF6897) // Receive the number '1'
     {
       digitalWrite(led, HIGH);//turn on LED
       Serial.println("  led on");
     }
    else if (value == 0xFF9867) // Receive the number '2'
     {
        digitalWrite(led, LOW);//turn off LED
       Serial.println("  led off");
    }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600. Press the button 1 of the remote, which will be displayed on the monitor, and the LED will be on. Similarly, press the button 2 , the LED will be off.

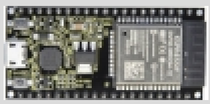
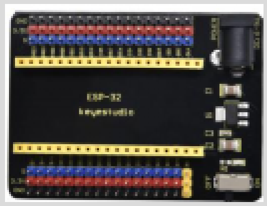
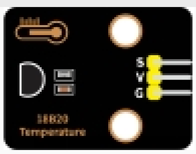
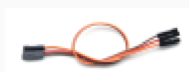
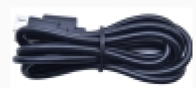
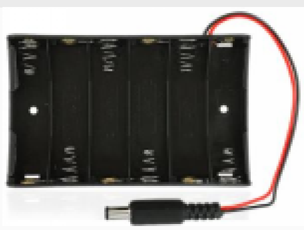


6.3.14 Project 58: Heat Dissipation Device

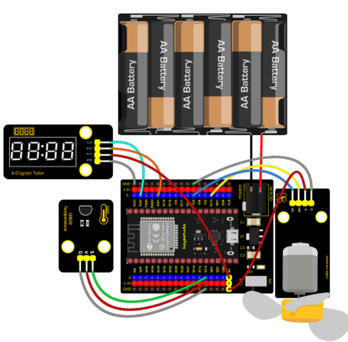
Description

We will use a temperature sensor and some modules to make a smart cooling device in this experiment. When the ambient temperature is higher than a certain value, the motor is turned on, thereby reducing the ambient temperature and achieving the heat dissipation effect. Then display the temperature value in the four-digit segment display.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 130 Motor*1	Keyestudio TM1650 4-Digit Segment Display*1
			
Keyestudio 18B20 Temperature Sensor*1	3P Dupont Wire*1	4P Dupont Wire*2	Micro USB Cable*1
			
Battery Holder*1	Battery(provided by yourself)*6		

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : heat abstractor
 * Description : DS18B20 controls a four digit tube and a motor that simulates Heat
```

(continues on next page)

(continued from previous page)

```
↪Abstractor
 * Auther      : http//www.keyestudio.com
*/
#include <DS18B20.h>
#include "TM1650.h" //Import the TM1650 library file
//The two ports are GP21 and GP22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

//ds18b20 pin to 13
DS18B20 ds18b20(13);
void setup() {
  Serial.begin(9600);
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  //Motor is connected to 15 4
  pinMode(15, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  double temp = ds18b20.GetTemp();//Read the temperature
  temp *= 0.0625;//The conversion accuracy is 0.0625/LSB
  Serial.println(temp);
  displayFloatNum(temp);//4- digit tube display temperature value
  if (temp > 25) {//When the temperature exceeds 25 degrees Celsius, turn on the fan
    digitalWrite(15, LOW);
    digitalWrite(4, HIGH);
  } else {//Otherwise, turn off the fan.
    digitalWrite(15, LOW);
    digitalWrite(4, LOW);
  }
  delay(100);
}

void displayFloatNum(float temp){
  if(temp > 9999)
    return;
  int dat = temp*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
```

(continues on next page)

(continued from previous page)

```
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Code Explanation

The setting of variables and the storage of detection values are the same as what we learned earlier. We also set a temperature threshold and control the rotation of the motor when the threshold is exceeded, and then we use the digital tube to display the temperature value.

Test Result

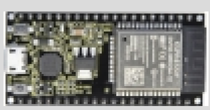
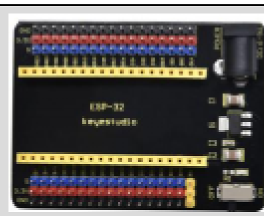
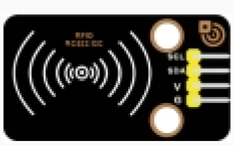
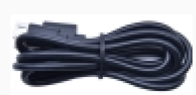
Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch on the ESP32 expansion board to the ON end, compile and upload the code to the ESP32. After uploading successfully, we can see the temperature of the current environment (unit is Celsius) on the four-digit segment display, as shown in the figure below. If this value exceeds the value we set, the fan will rotate to dissipate heat.

6.3.15 Project 59: Intelligent Entrance Guard System

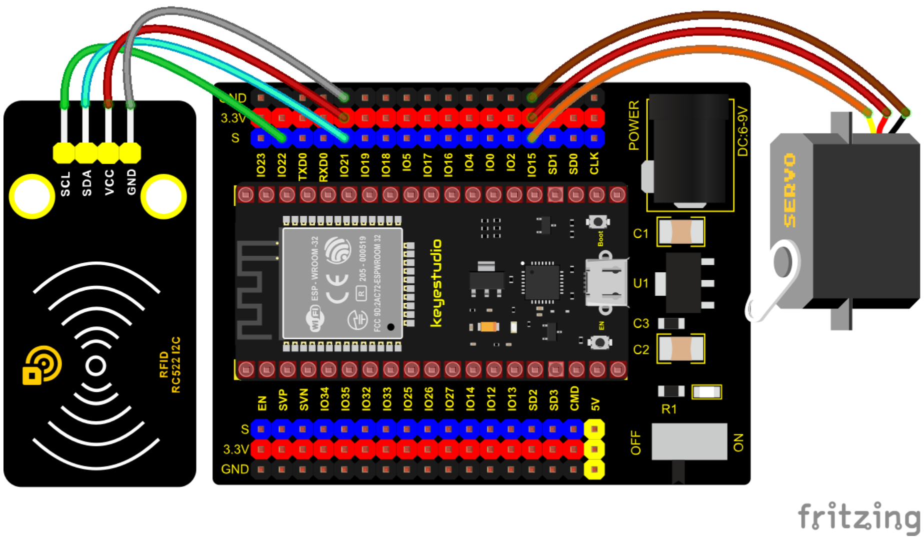
Description

In this project, we use the RFID522 card swiping module and the servo to set up an intelligent access control system. The principle is very simple. We use RFID522 swipe card module, an IC card or key card to unlock.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Key*1	IC Card*1
			
Keyestudio RFID Module*1	Servo*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

Note: Different RFID-MFRC522 IC cards and keys have diverse values. You can substitute your own IC cards and keys values for the corresponding values read by the RFID-MFRC522 module in the program, otherwise the servo can't be controlled when uploading the test code to the ESP32.

For example: You can replace the rfid_str of the `if (rfid_str == "edf7945a" || rfid_str == "4c96b6e"` in the program code with your own IC cards and keys values read by the RFID-MFRC522 module.

```
//**********************************************************************************
/*
 * Filename    : Intelligent_access_control
 * Description : RFID controlled steering gear simulated door opening
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "MFRC522_I2C.h"
// IIC pins default to GPIO21 and GPIO22 of ESP32
// 0x28 is the i2c address of SDA, if doesn't matchplease check your address with i2c.
MFRC522 mfrc522(0x28);   // create MFRC522.

#include <ESP32Servo.h>
Servo myservo;  // create servo object to control a servo
int servoPin = 15; // Servo motor pin

String rfid_str = "";

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mfrc522.PCD_Init();
  ShowReaderDetails();            // dispaly PCD - MFRC522 read carder
  Serial.println(F("Scan PICC to see UID, type, and data blocks..."));

  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo
```

(continues on next page)

(continued from previous page)

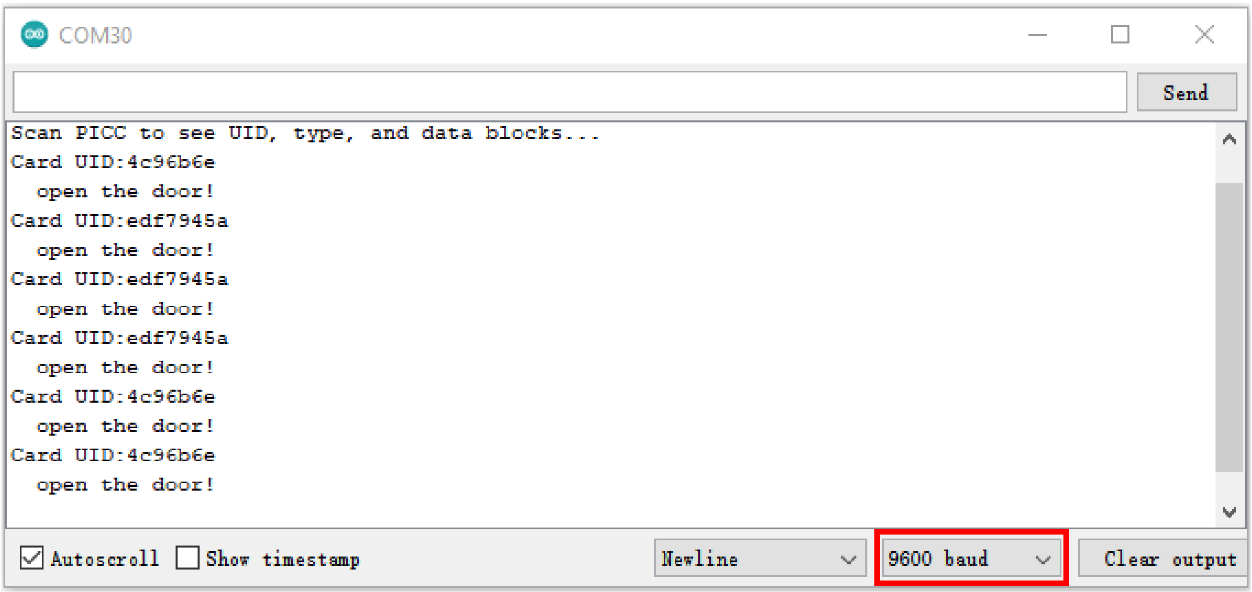
```
↪object
  myservo.write(0);
  delay(500);
}

void loop() {
   if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }

  // select one of door cards. UID and SAK are mfrc522.uid.

  // save UID
  rfid_str = ""; //String emptying
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    rfid_str = rfid_str + String(mfrc522.uid.uidByte[i], HEX);  //Convert to string
    //Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    //Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println(rfid_str);

  if (rfid_str == "edf7945a" || rfid_str == "4c96b6e") {
    myservo.write(180);
    delay(500);
    Serial.println("  open the door!");
    }
}

void ShowReaderDetails() {
  //  attain the MFRC522 software
  byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
  Serial.print(F("MFRC522 Software Version: 0x"));
  Serial.print(v, HEX);
  if (v == 0x91)
    Serial.print(F(" = v1.0"));
  else if (v == 0x92)
    Serial.print(F(" = v2.0"));
  else
    Serial.print(F(" (unknown)"));
  Serial.println("");
  // when returning to 0x00 or 0xFF, may fail to transmit communication signals
  if ((v == 0x00) || (v == 0xFF)) {
    Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?
↪"));
  }
}
//*************************************************************************************
```

Code Explanation

In the previous experiment, our card swipe module has tested the information of IC card and key. Then we use this corresponding information to control the door.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600. When we use the IC card or blue key to swipe the card, the monitor displays the card and the key information and "open the door", at the same time, the servo rotates to the corresponding angle to simulate opening the door.

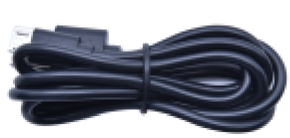
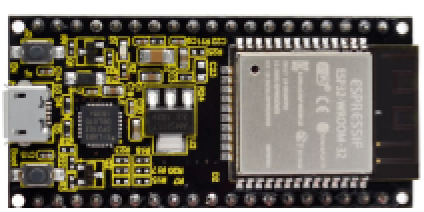


6.3.16 Project 60Bluetooth

This chapter mainly introduces how to use the Bluetooth of ESP32 for simple data transmission with mobile phone. Experiment 60.1 is conventional Bluetooth, and experiment 60.2 is Bluetooth control LED.

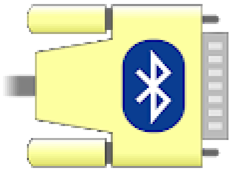
Project 60.1Classic Bluetooth

Components

	
USB Cable*1	ESP32*1

In this experiment, we need to use a Bluetooth dabbed serial Bluetooth terminal for a study. If you haven't install it, please click the installation: https://www.appsapk.com/serial-bluetooth-terminal/ .

Here is its sign:



Component Knowledge

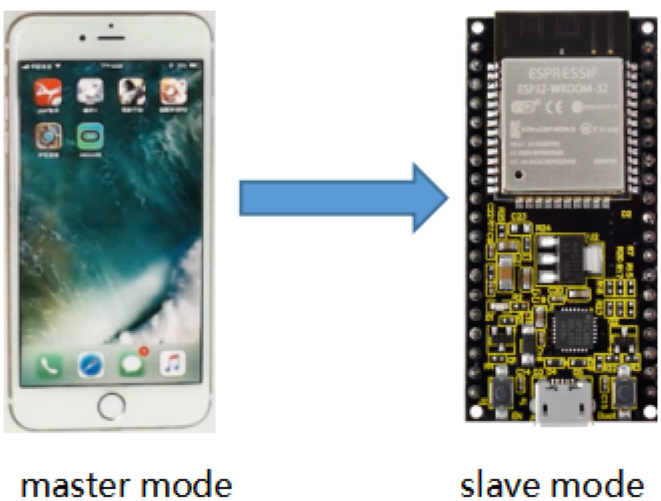
Bluetooth is a short-distance communication system that can be divided into two types, namely low power Bluetooth (BLE) and classic Bluetooth. There are two modes for simple data transfer: master mode and slave mode.

Master Mode:

In this mode, work is done on the master device and can be connected to the slave device. When the device initiates a connection request in the main mode, information such as the address and pairing password of other Bluetooth devices are required. Once paired, you can connect directly to them.

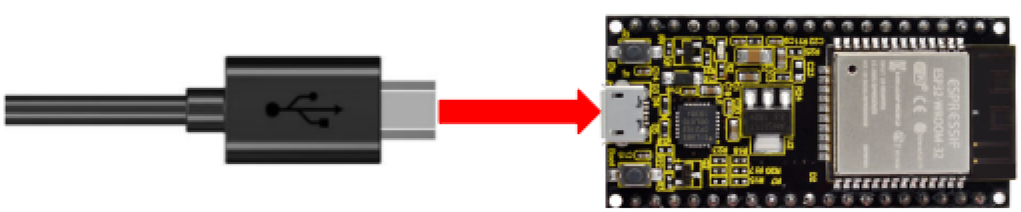
Slave Mode:

A Bluetooth module in the slave mode can only accept connection requests from the host, but cannot initiate connection requests. After being connected to a host device, it can send and receive data through the host device. Bluetooth devices can interact with each other, when they interact, the Bluetooth device in the main mode searches for nearby devices. While a connection is established, they can exchange data. For example, when a mobile phone exchanges data with ESP32, the mobile phone is usually in master mode and the ESP32 is in slave mode.



Wiring Diagram

We can use a USB cable to connect ESP32 mainboard to the USB port on a computer.

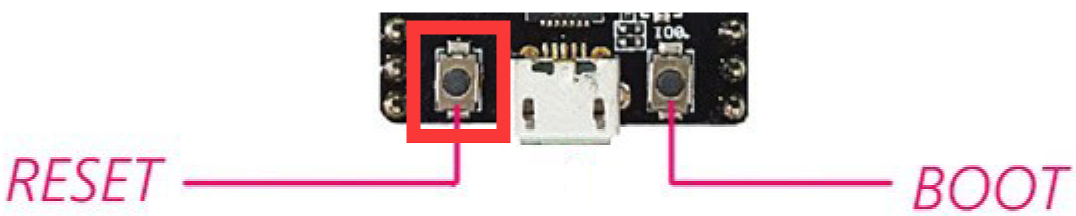


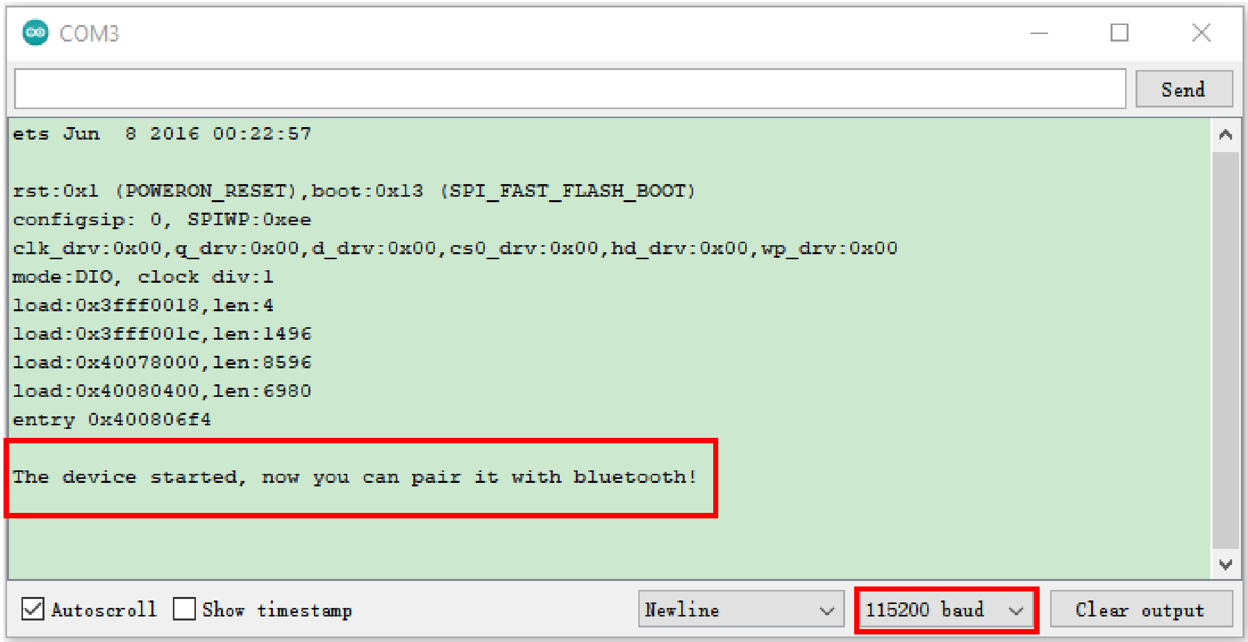
Test Code

```
//**********************************************************************************
/*
 * Filename    : Classic Bluetooth
 * Description : ESP32 communicates with the phone by bluetooth and print phone's data
↪via a serial port
 * Auther      : http//www.keyestudio.com
*/
#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);
}
//**********************************************************************************
```

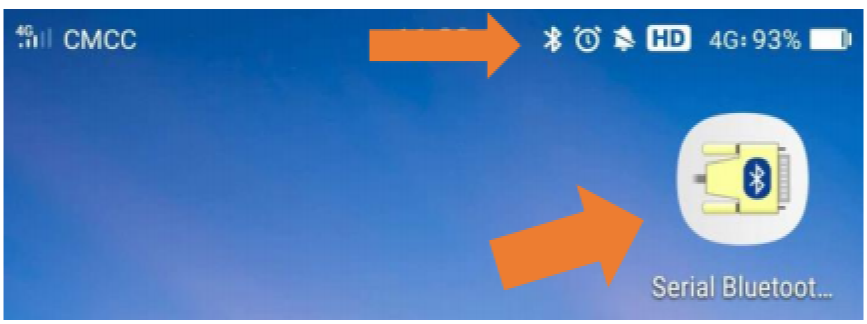
Test Result

Compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200. When you see the serial prints the character, as shown below, it means that the ESP32's bluetooth is waiting for connect ion with a phone. (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the RESET button of the ESP32)

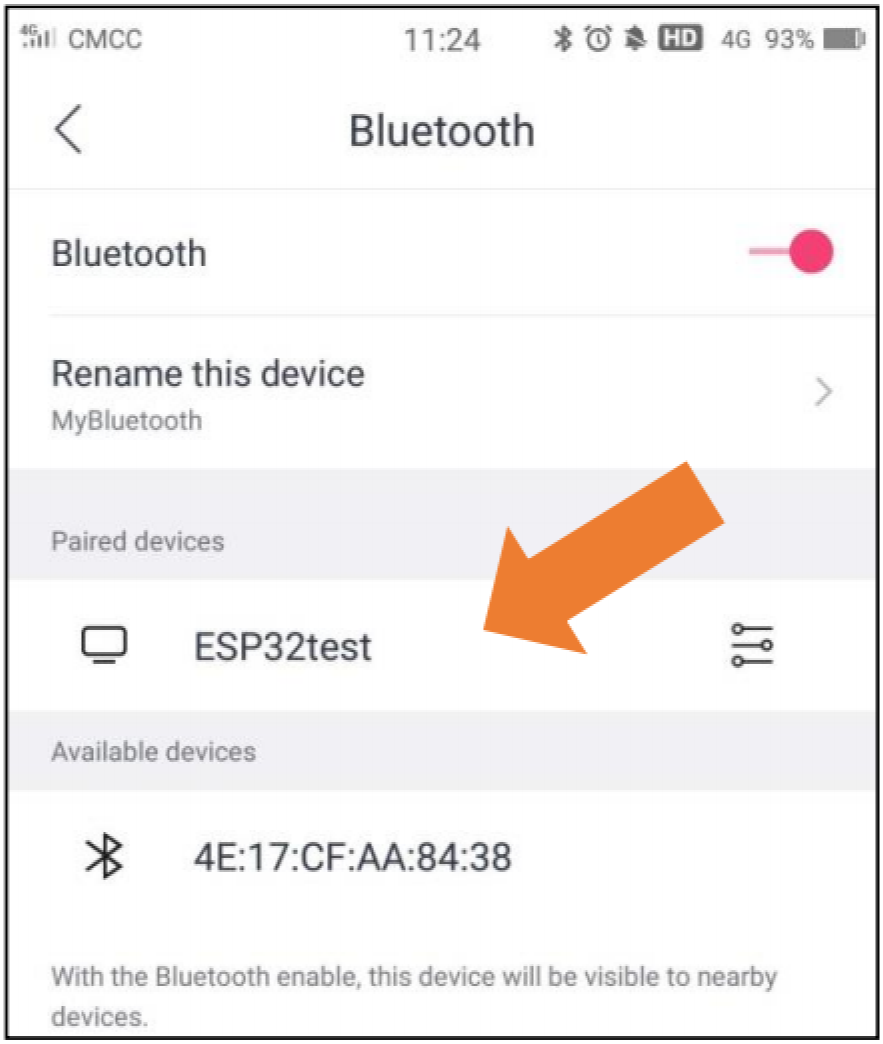




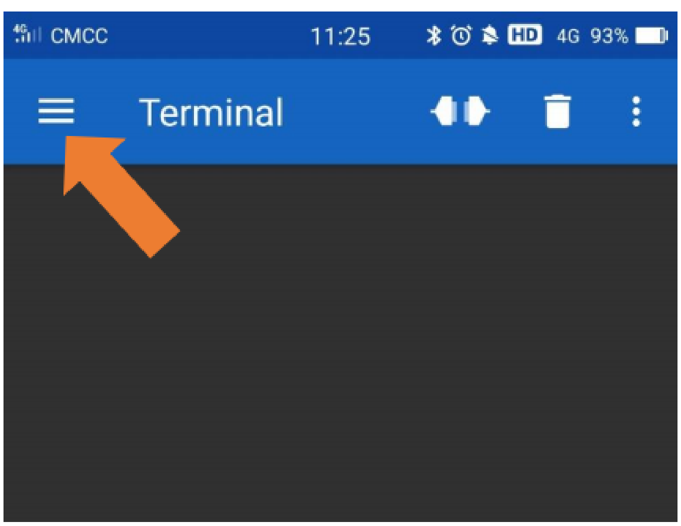
Ensure that your mobile phone Bluetooth is enabled and the Bluetooth application of "Serial Bluetooth Terminal" is installed.

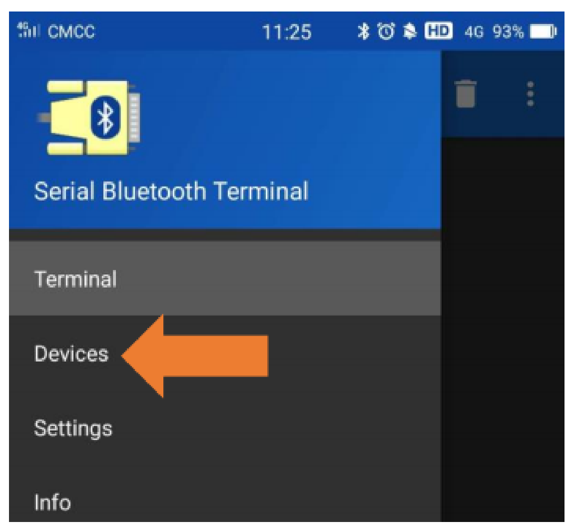


Click"Search"search for the nearby Bluetooth and select to connect the"ESP32 test".

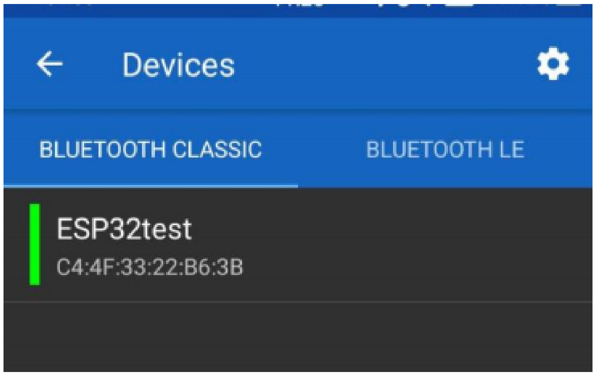


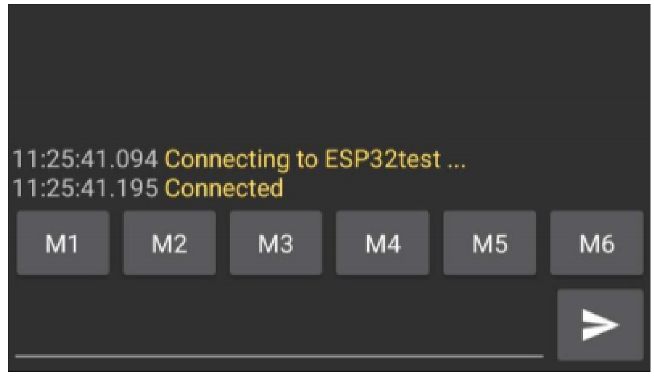
Open the software APP and click the left side of the terminal, select “Devices”.





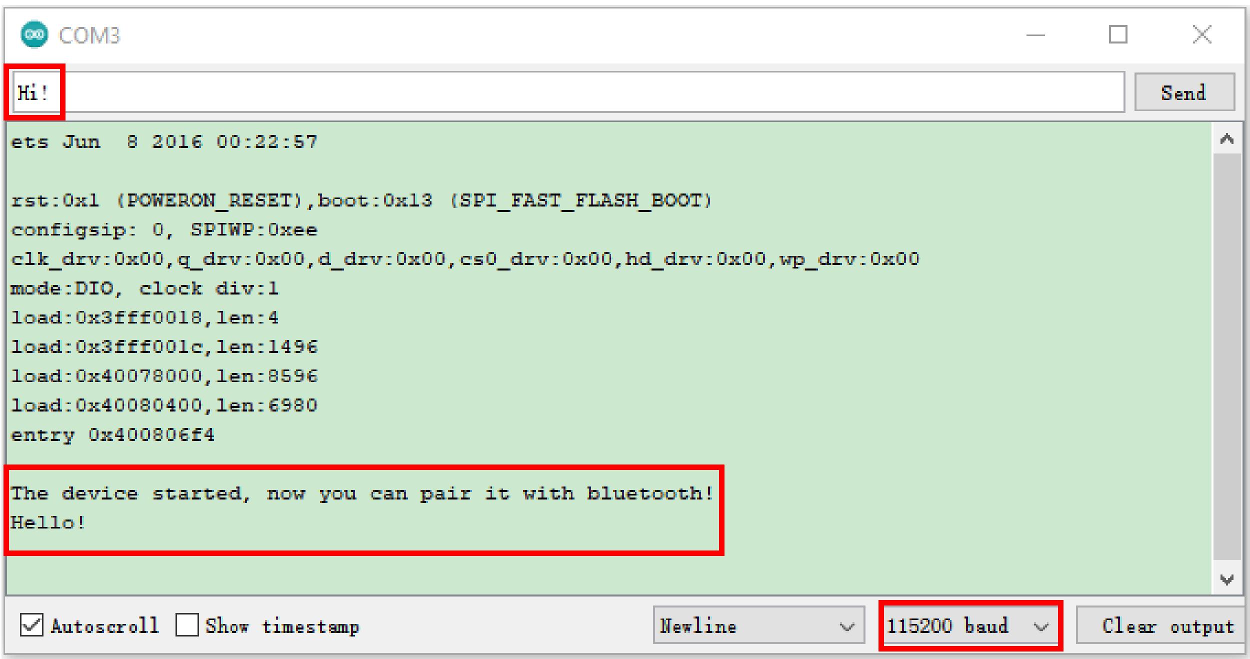
If you select ESP32test in classic bluetooth mode, a successful connection message will appear as shown below.

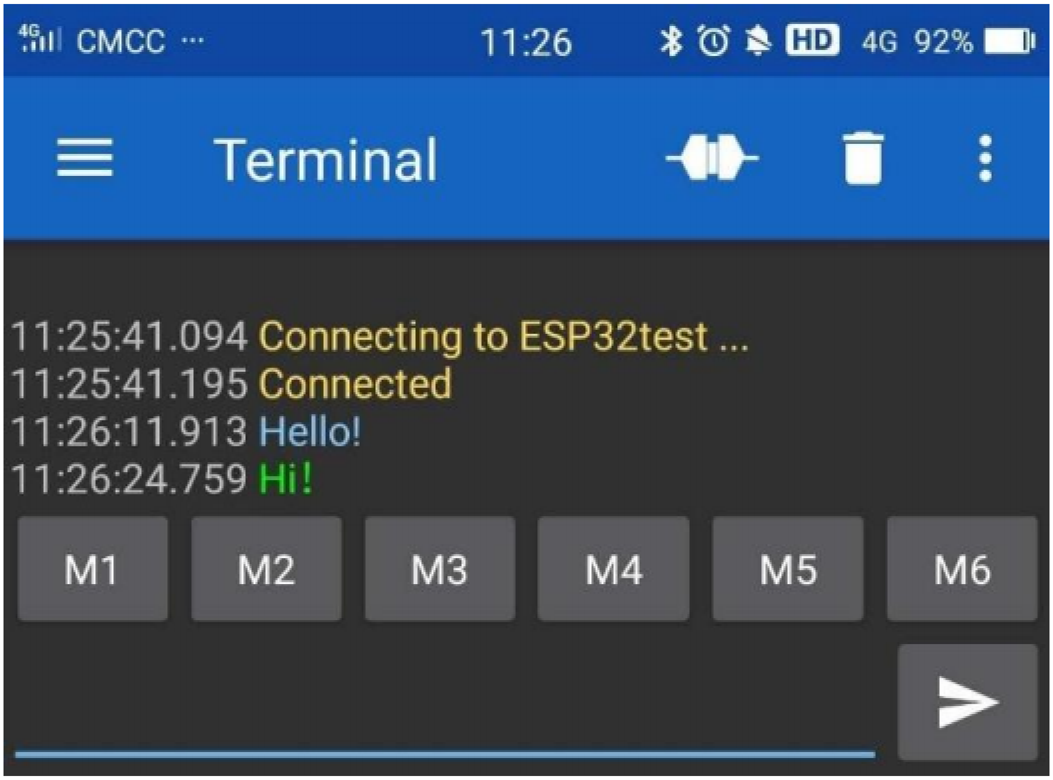




Data can be transferred between your phone and a computer via ESP32 now.

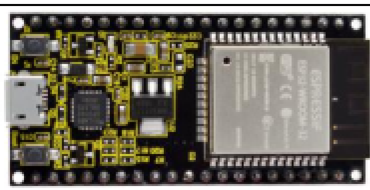
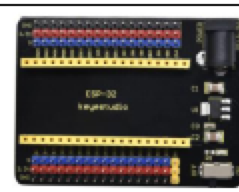
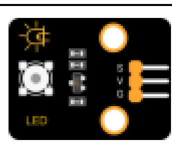
Send "Hello", When the computer receives it, which will reply with "Hi!".



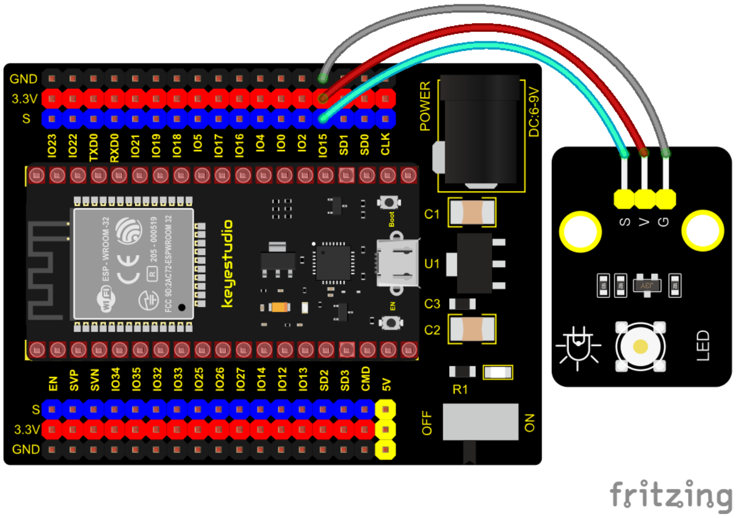


Project 60.2Bluetooth Control LED

Components

		
ESP32*1		ESP32 Expansion Board*1
		
Keyestudio Purple LED Module*1	3P Dupont*1	MicroUSB Cable*1

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Bluetooth Control LED
 * Description : The phone controls esp32's led via bluetooth.
                When the phone sends "LED_on," ESP32's LED lights turn on.
                When the phone sends "LED_off," ESP32's LED lights turn off.
 * Auther      : http//www.keyestudio.com
*/
#include "BluetoothSerial.h"
#include "string.h"
#define LED 15
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,HIGH);
    }
    if(strncmp(buffer,"led_off",7)==0){
      digitalWrite(LED,LOW);
```

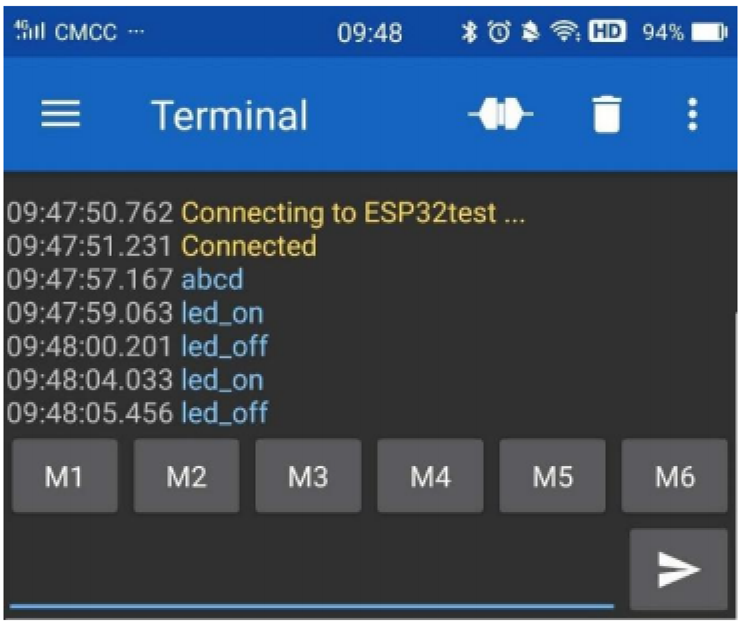
(continues on next page)

(continued from previous page)

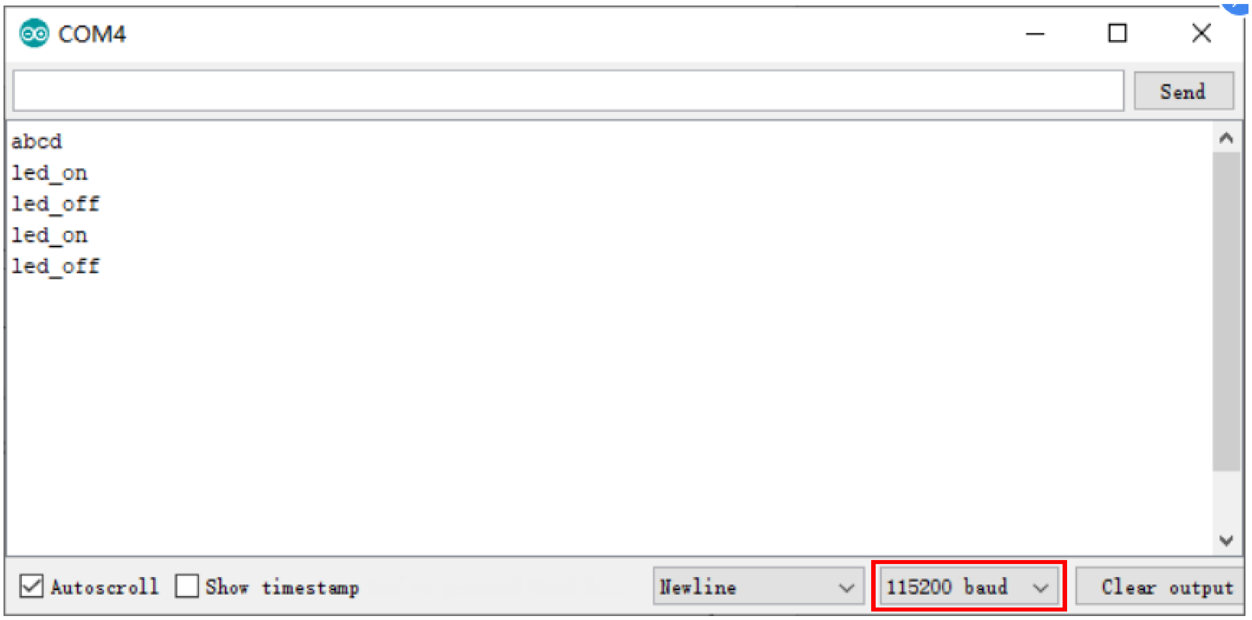
```
    }
    count=0;
    memset(buffer,0,20);
  }
}
//*************************************************************************************
```

Test Result

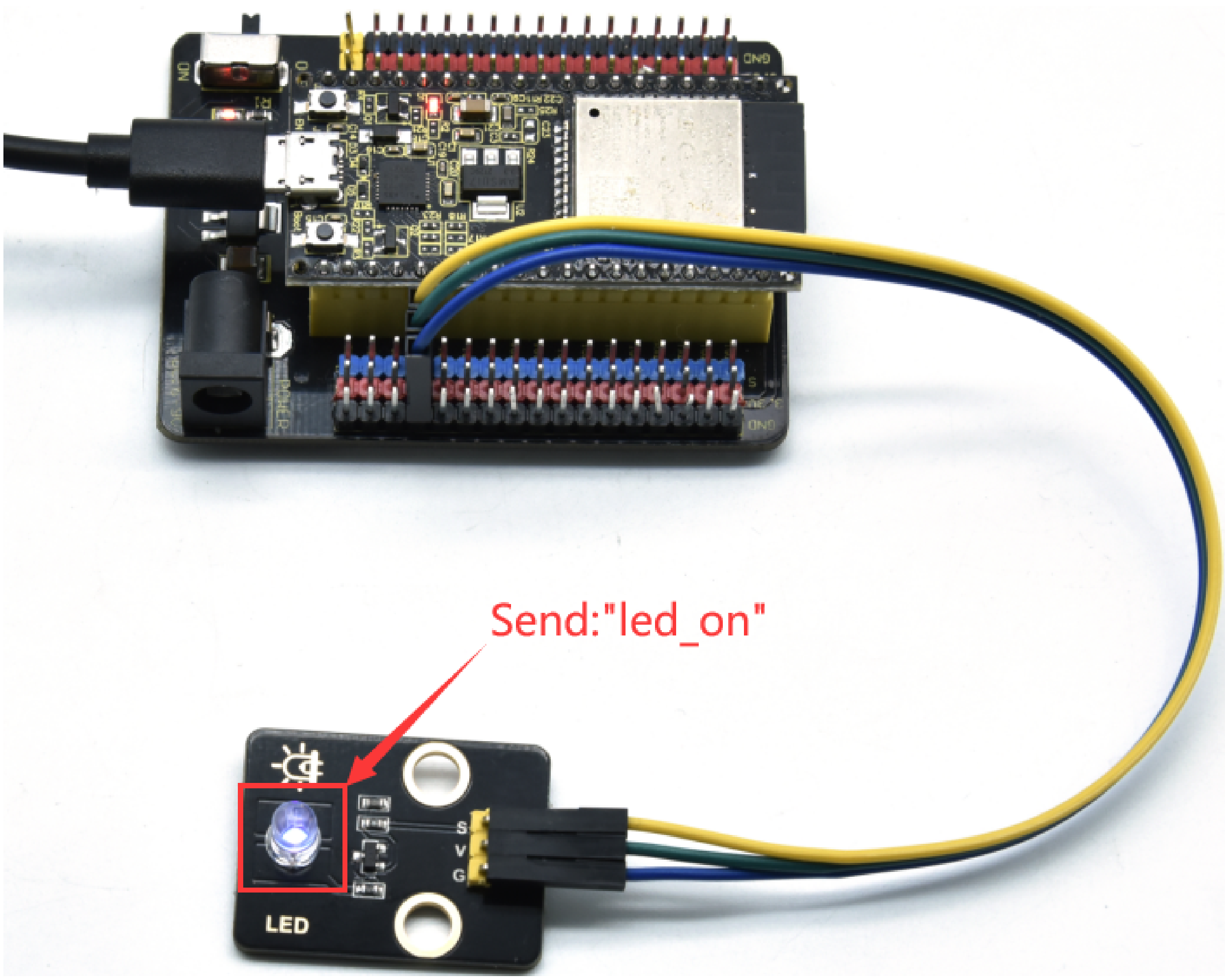
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The APP operation is the same as the **project 60.1**. To make the external LED on and off, simply change the sending content to "led_on" and "led_off". Moving the APP to send data:

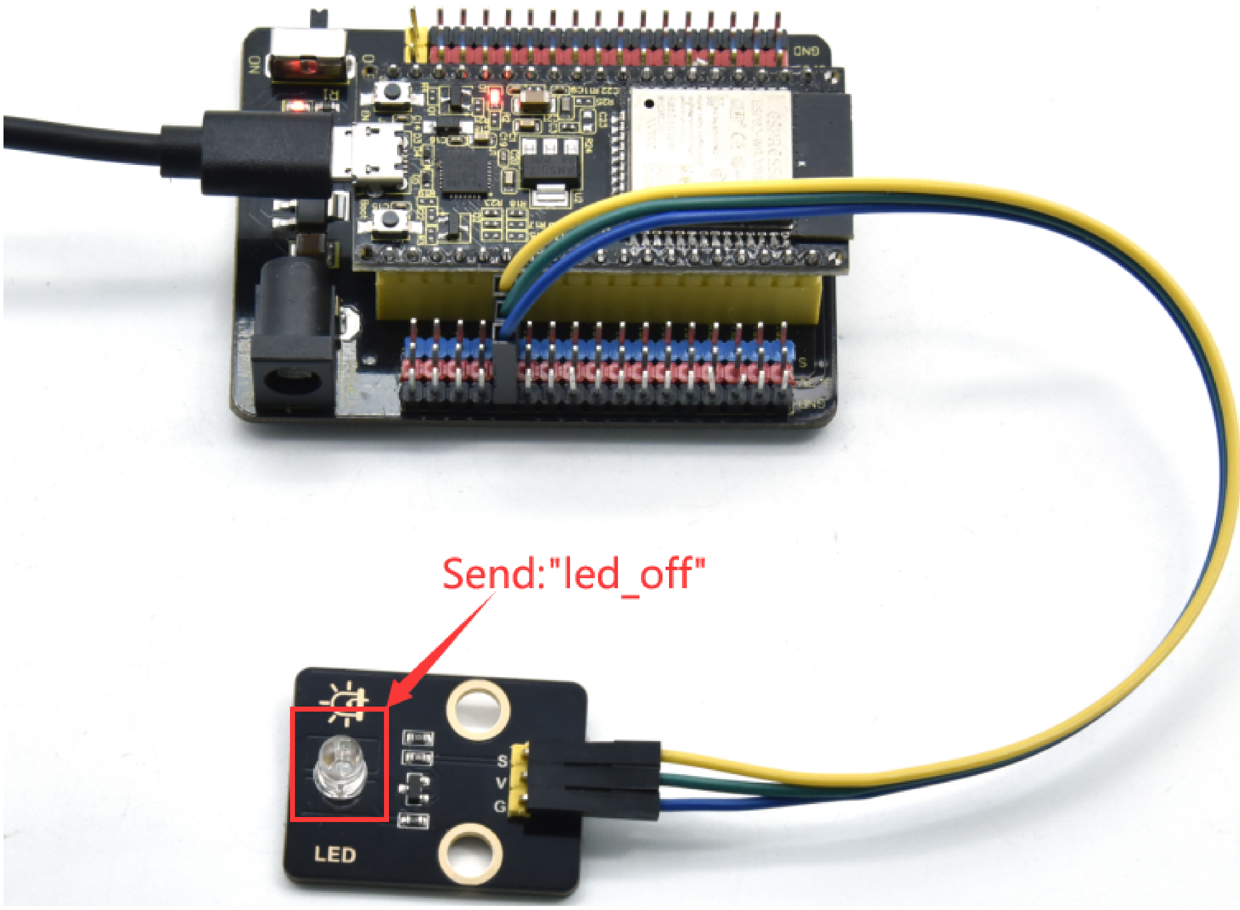


The serial monitor will display as follows:



LED Circumstance





Note:

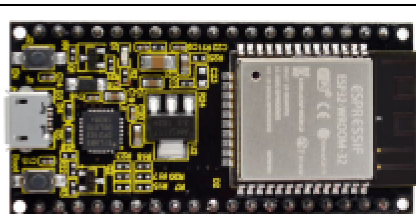
If the sent content is not "led-on 'or" led-off ", the status of the LED will not change. If the LED is on, it remains on when irrelevant content is received; Conversely, if the LED is off, it continues to be off when irrelevant content is received.

6.3.17 Project 61WIFI Station Mode

Description

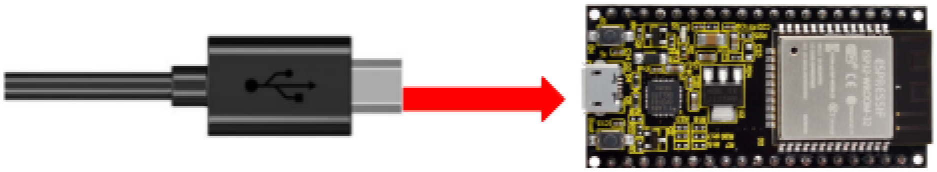
ESP32 has three different WiFi modes: Station mode, AP mode and AP+Station mode. All WiFi programming projects must be configured with WiFi running mode before using, otherwise the WiFi cannot be used. In this project, we are going to learn the WiFi Station mode of the ESP32.

Components

	
MicroUSB Cable*1	ESP32*1

Wiring Diagram

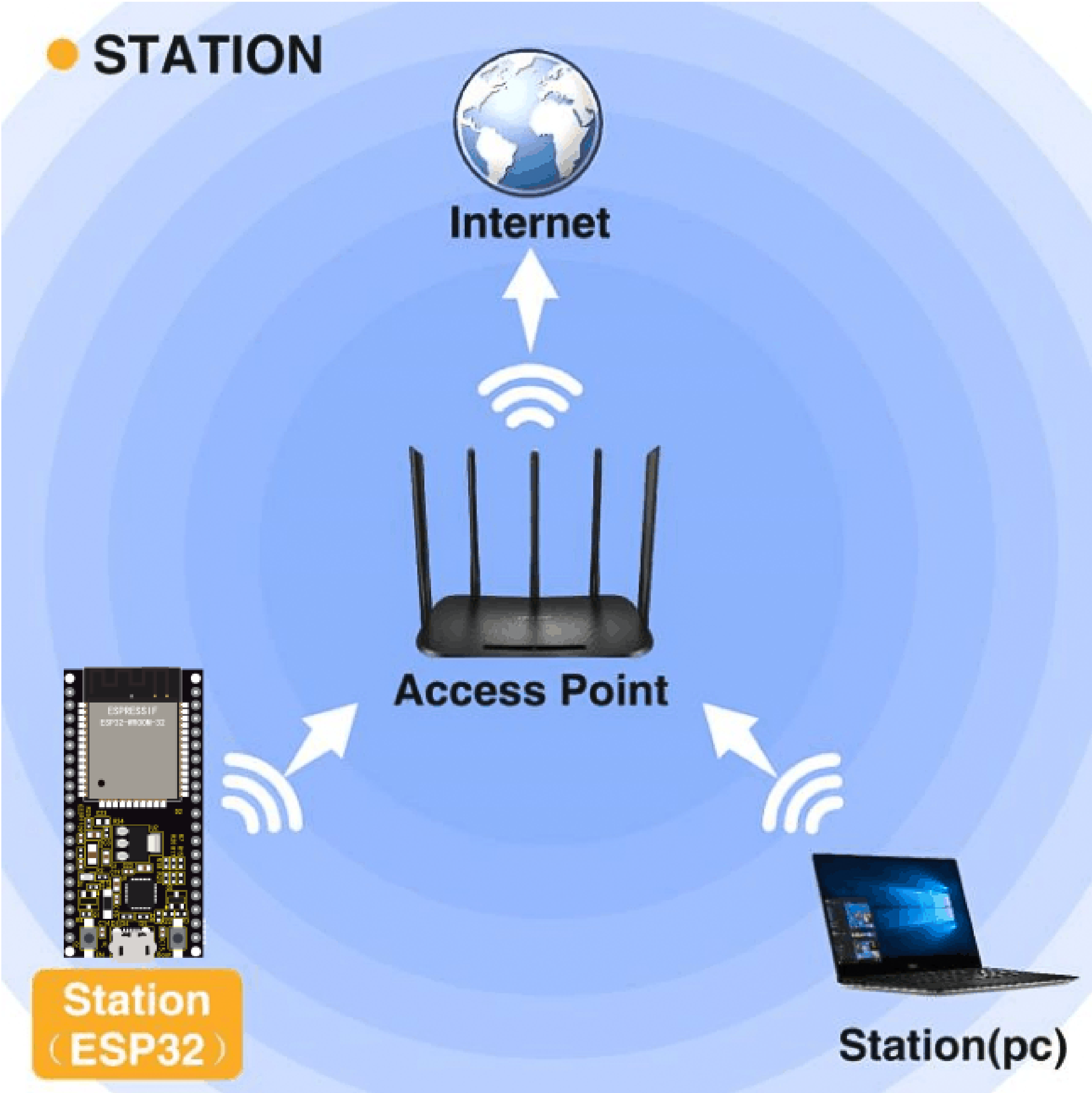
Plug the ESP32 to the USB port of your PC.



Component Knowledge

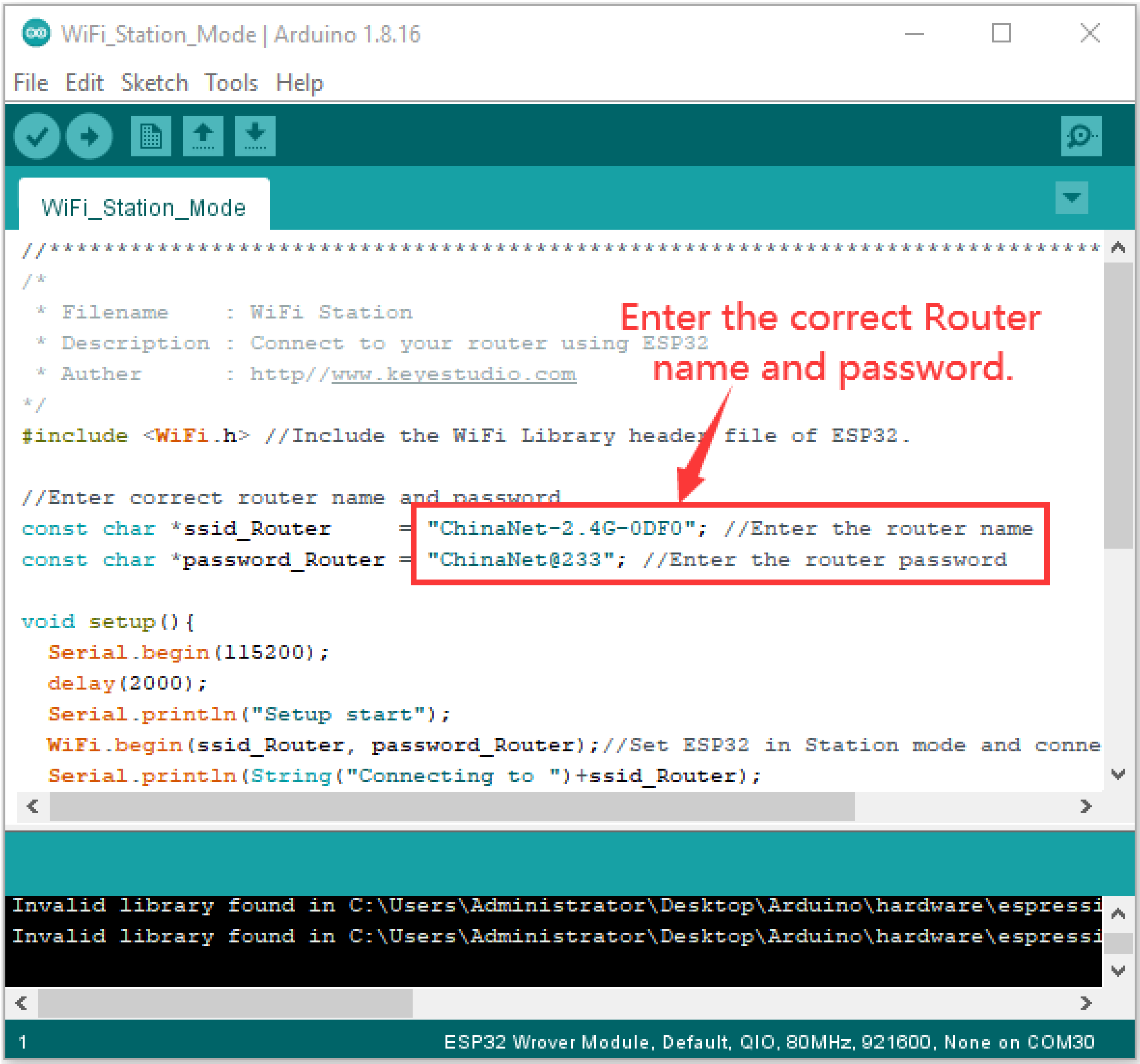
Station mode

When setting Station mode, the ESP32 is taken as a WiFi client. It can connect to the router network and communicate with other devices on the router via a WiFi connection. As shown in the figure below, the PC and the router have been connected. If the ESP32 wants to communicate with the PC, the PC and the router need to be connected.



Test Code

Since WiFi names and passwords vary from place to place, thereby users need to enter the correct WiFi names and passwords in the box shown below before the program code runs.




```
//*****************************************************************************
/*
 * Filename    : WiFi Station
 * Description : Connect to your router using ESP32
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h> //Include the WiFi Library header file of ESP32.

//Enter correct router name and password.
const char *ssid_Router     = "ChinaNet-2.4G-0DF0"; //Enter the router name
const char *password_Router = "ChinaNet@233"; //Enter the router password

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setup start");
  WiFi.begin(ssid_Router, password_Router);//Set ESP32 in Station mode and connect it to
↪your router.
```

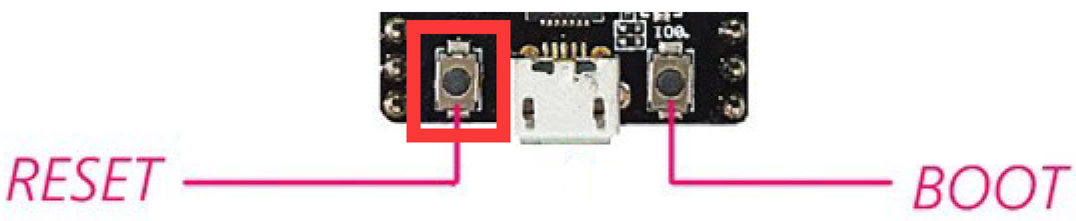
(continues on next page)

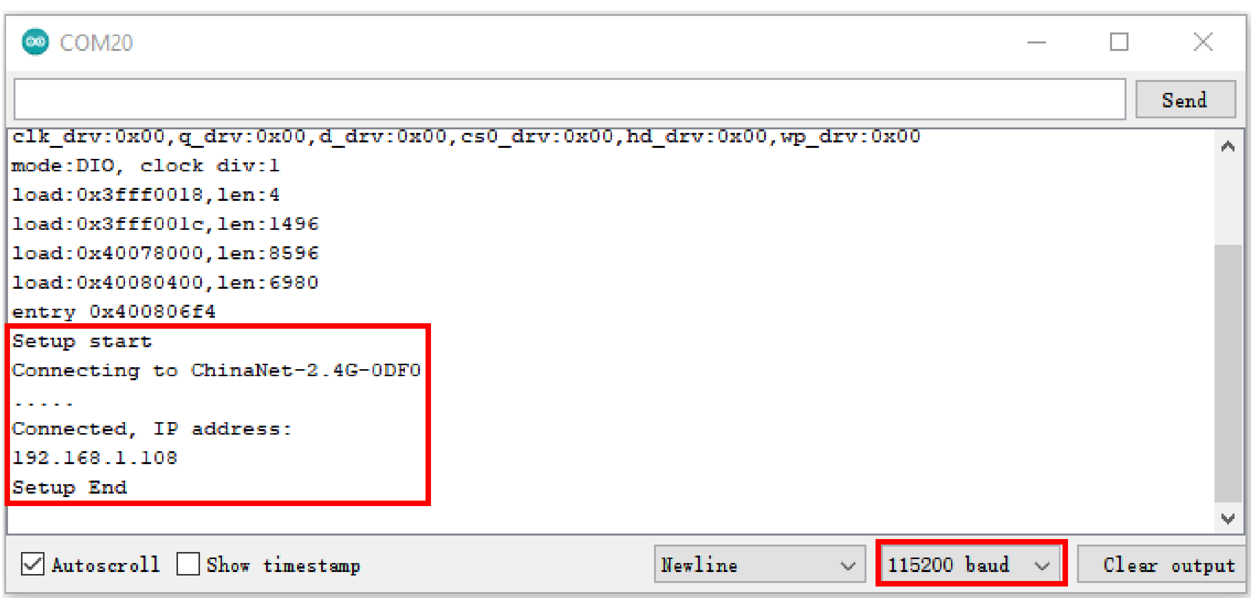
(continued from previous page)

```
  Serial.println(String("Connecting to ")+ssid_Router);
//Check whether ESP32 has connected to router successfully every 0.5s.
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected, IP address: ");
  Serial.println(WiFi.localIP());//Serial monitor prints out the IP address assigned to␣
↪ESP32.
  Serial.println("Setup End");
}

void loop() {
}
//*************************************************************************************
```

Test Result

After entering the correct WiFi names and passwords, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200. When the ESP32 successfully connects to ssid_WiFi, the serial monitor prints out the IP address, then monitor will display as follows: (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)



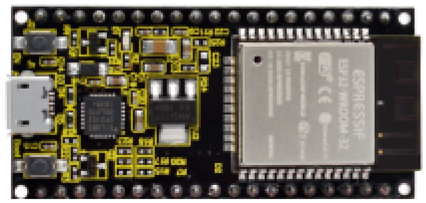


6.3.18 Project 62WIFI AP Mode

Description

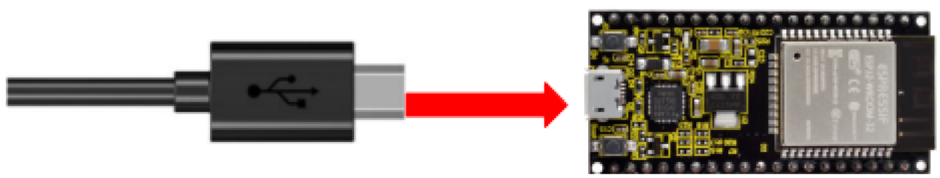
In this project, we are going to learn the WiFi AP mode of the ESP32.

Components

	
MicroUSB Cable*1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your PC.



Component Knowledge

AP Mode:

When setting AP mode, a hotspot network will be created, waiting for other WiFi devices to connect. As shown below:

Take the ESP32 as the hotspot, if a phone or PC needs to communicate with the ESP32, it must be connected to the ESP32's hotspot. Communication is only possible after a connection is established via the ESP32.

Test Code

Before running the code, you can make any changes to the ESP32 AP name and password in the box as shown below, but in a default circumstance, it doesn't need to modify.


```
//**********************************************************************************
/*
 * Filename    : WiFi AP
 * Description : Set ESP32 to open an access point
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h> //Include the WiFi Library header file of ESP32.

const char *ssid_AP     = "ESP32_Wifi"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password

IPAddress local_IP(192,168,1,108);//Set the IP address of ESP32 itself
IPAddress gateway(192,168,1,1);   //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);
```

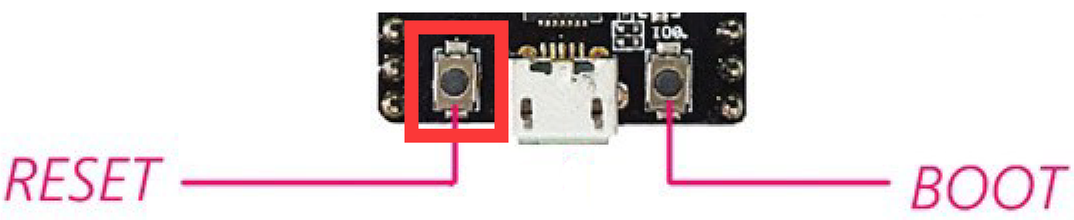
(continues on next page)

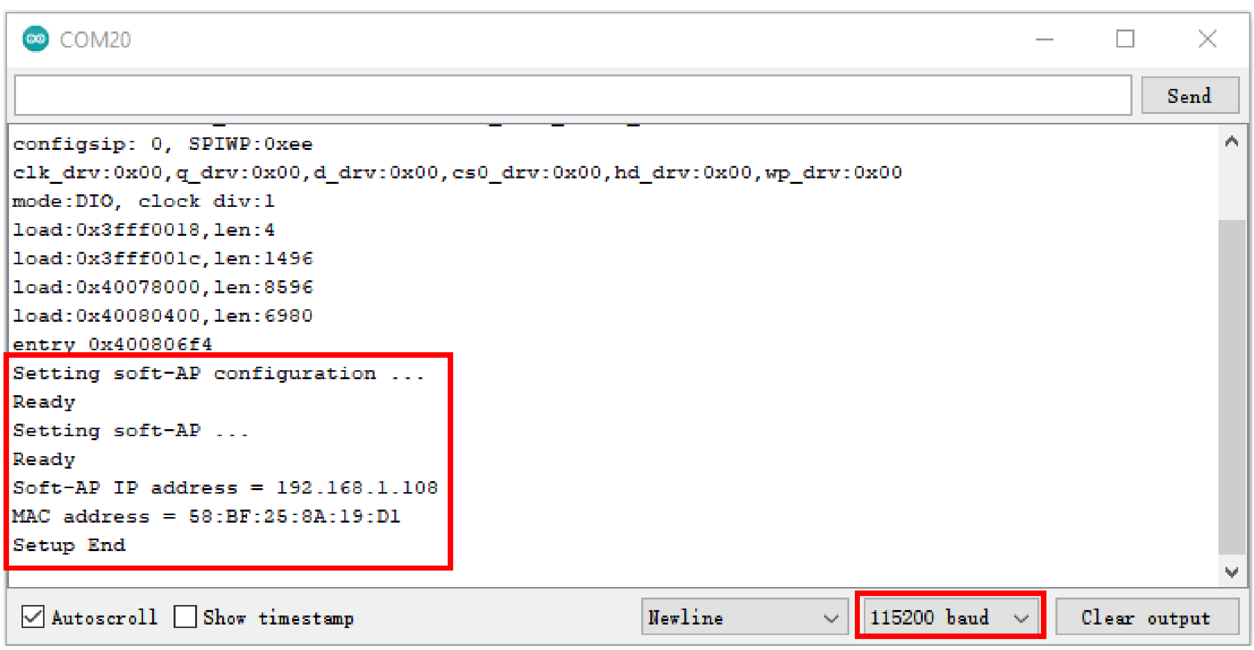
(continued from previous page)

```
  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() {
}
//*************************************************************************************
```

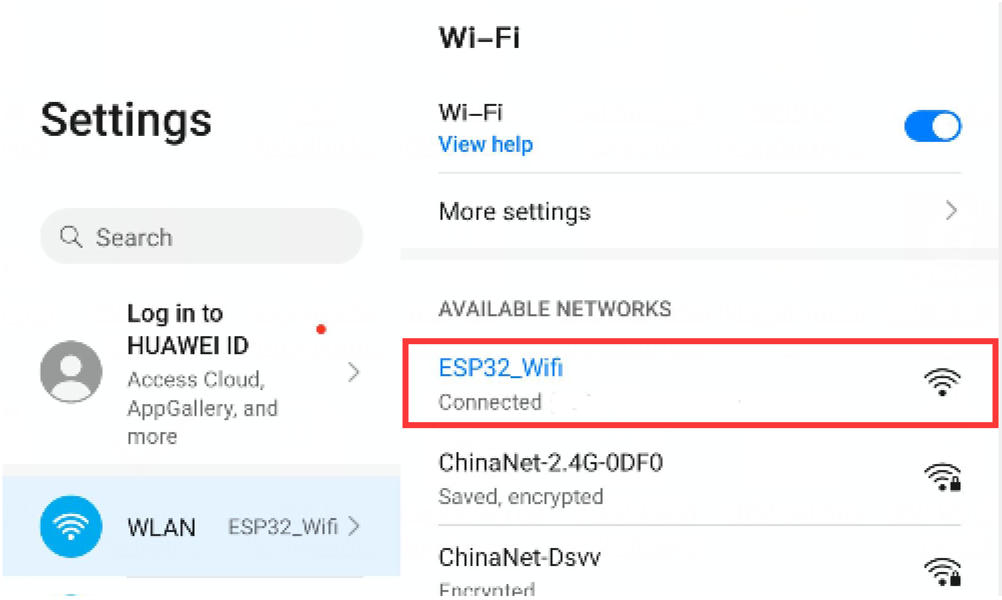
Test Result

Compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200, then monitor will display as follows: (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the RESET button of the ESP32)





When observing the printed information of the serial monitor, turn on the WiFi scanning function of the mobile phone, you can see the ssid_AP on ESP32, which is dubbed "ESP32_Wifi" in this program code. You can connect to it either by typing the password "12345678" or by modifying the program code to change its AP name and password.

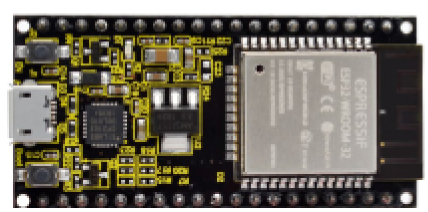


6.3.19 Project 63WIFI AP+Station Mode

Description

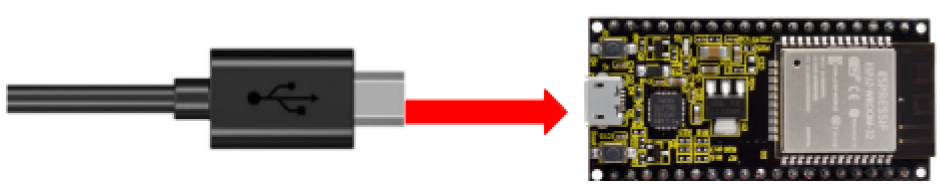
In this project, we are going to learn the AP+Station mode of the ESP32.

Components

	
MicroUSB Cable*1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your PC



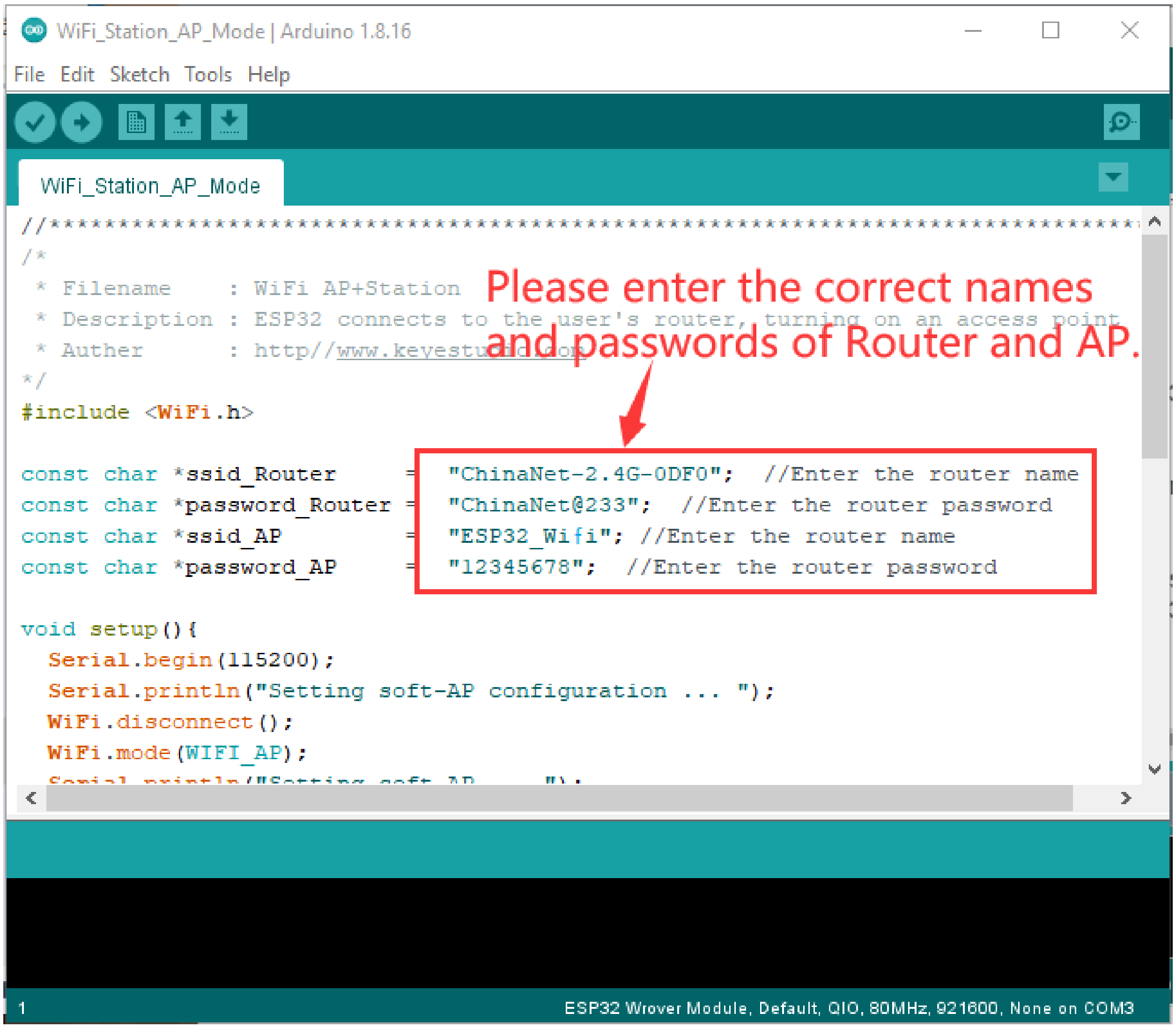
Component Knowledge

AP+Station mode:

In addition to the AP mode and the Station mode, AP+Station mode can be used at the same time. Turn on the Station mode of the ESP32, connect it to the router network, and it can communicate with the Internet through the router. Then turn on the AP mode to create a hotspot network. Other WiFi devices can be connected to the router network or the hotspot network to communicate with the ESP32.

Test Code

Before running the code, you need to modify the ssid_Routerpassword_Routerssid_AP and password_AP, as shown in the box below:




```
//**********************************************************************************
/*
 * Filename    : WiFi AP+Station
 * Description : ESP32 connects to the user's router, turning on an access point
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h>
```

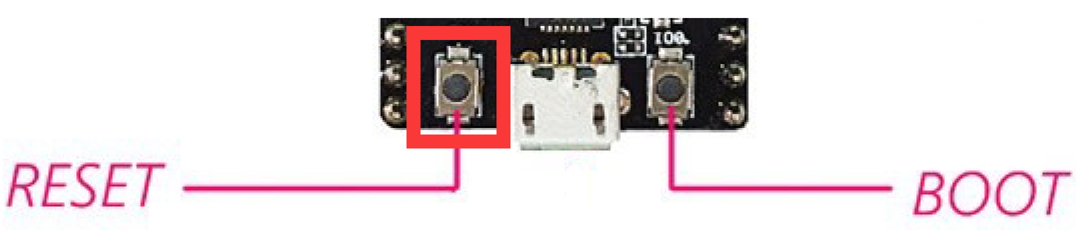
(continues on next page)

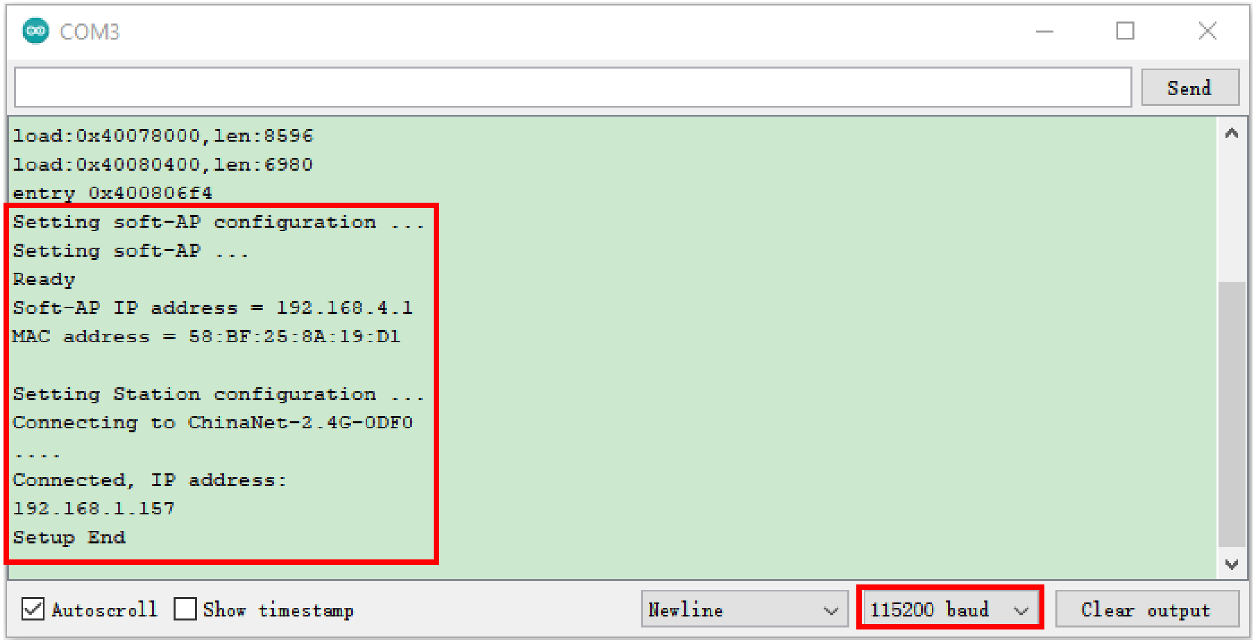
(continued from previous page)

```
const char *ssid_Router     =  "ChinaNet-2.4G-0DF0";  //Enter the router name
const char *password_Router =  "ChinaNet@233";  //Enter the router password
const char *ssid_AP         =  "ESP32_Wifi"; //Enter the router name
const char *password_AP     =  "12345678";  //Enter the router 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() {
}
//************************************************************************************
```

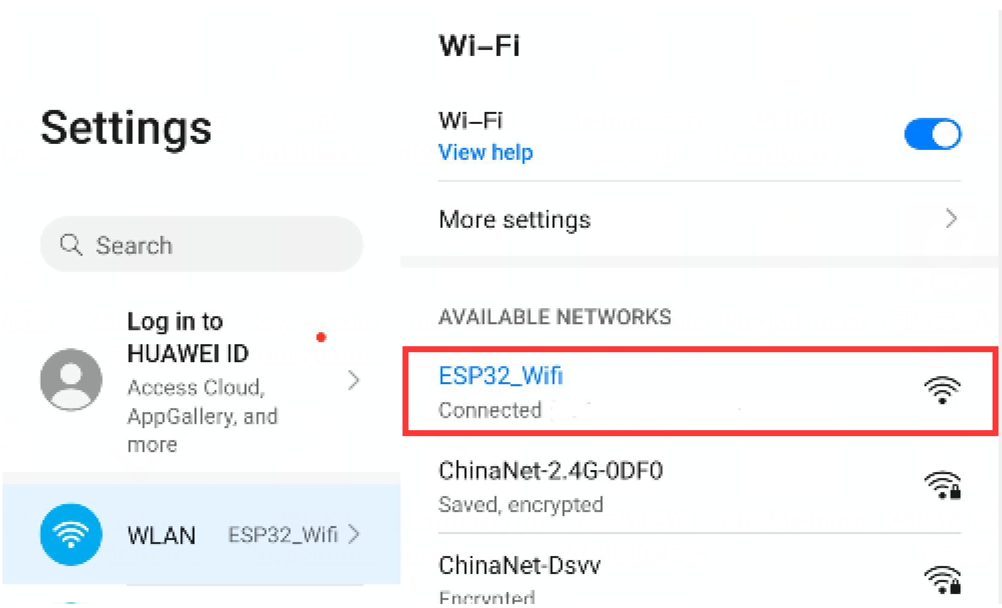
Test Result

Ensure that the code in the program has been modified correctly, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200, then monitor will display as follows: (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the RESET button of the ESP32)

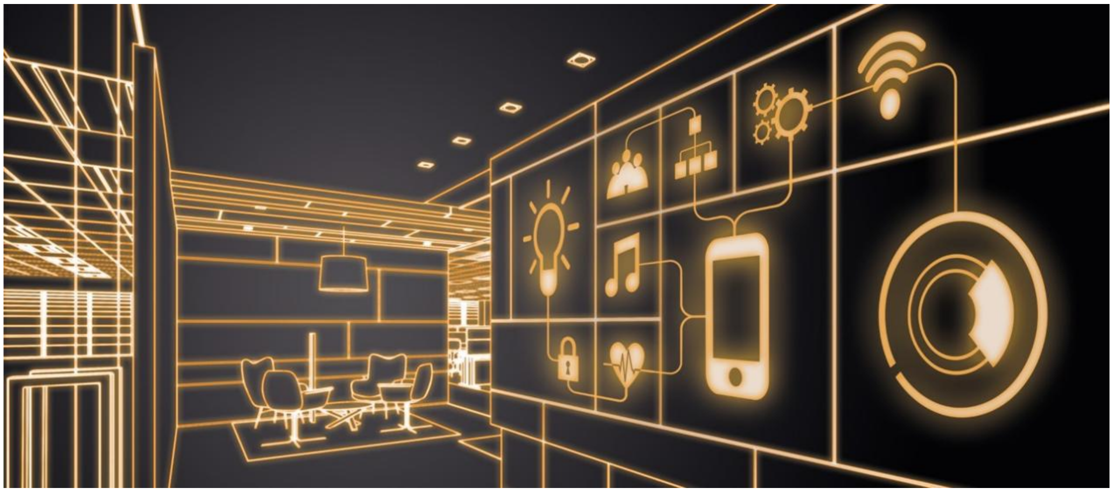




Open the WiFi scanning function of the mobile phone, you can see the ssid_AP.



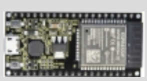
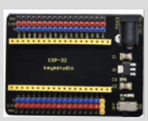
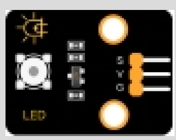
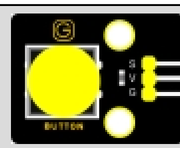
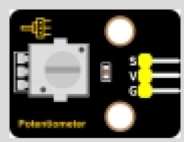
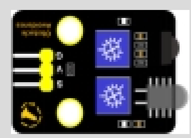
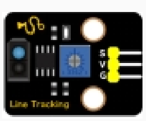
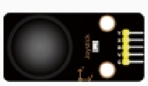
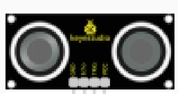
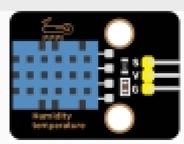
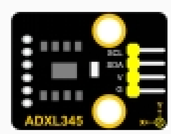
6.3.20 Project 64: Comprehensive Experiment



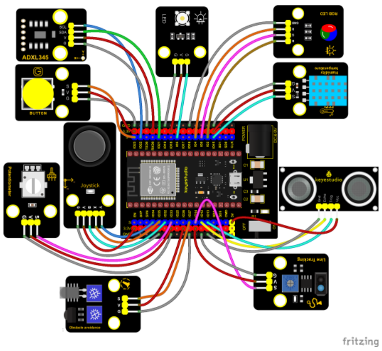
Introduction

We did a lot of experiments, and for each one we needed to re-upload the code, so can we achieve different functions through an experiment? In this experiment, we will use an external button module to achieve different functions.

Components Required

					
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Purple LED Module*1	Keyestudio Button Module*1	Keyestudio Rotary Potentiometer*1	Keyestudio Obstacle Avoidance Sensor*1
					
Keyestudio Line Tracking Sensor*1	Keyestudio DIY Joystick Module*1	Keyestudio HC-SR04 Ultrasonic sensor *1	Keyestudio DIY-Common Cathode RGB Module *1	Keyestudio XHT11 Temperature and Humidity Sensor *1	Keyestudio ADXL345 Acceleration Sensor*1
					
MicroUSB Cable*1	3PDupont Wire*6	4PDupont Wire*3	5PDupont Wire*1		

Wiring Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : Comprehensive experiment
 * Description : Multiple sensors/modules work together
 * Auther      : http//www.keyestudio.com
*/
#include "xht11.h"
#include "adxl345_io.h"

//ADXL345 sda-->21,scl-->22
adxl345 adxl345(21, 22);

//xht11 to gpio15
xht11 xht(15);

//rgb is connected to 4,0,2
int ledPins[] = {4, 0, 2};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;

//Rocker module port
int X = 35;
int Y = 34;
int KEY = 32;

//Potentiometer pin is connected to analog port 33
int resPin = 33;

//Trace sensor pin connected to IO port 14
int TrackingPin = 14;

//LED is Connected to GP5
#define PIN_LED   5  // the pin of the LED
#define CHAN     3

//Obstacle avoidance sensor is connected to GP27
int Avoid = 27;

//Ultrasonic sensor port
int Trig = 13;
```

(continues on next page)

(continued from previous page)

```
int Echo = 12;

//Key module port
int button = 23;

int PushCounter = 0;//Store the number of times a key is pressed
int yushu = 0;
unsigned char dht[4] = {0, 0, 0, 0};//Only the first 32 bits of data are received, not␣
↪the parity bits
bool ir_flag = 1;
float out_X, out_Y, out_Z;

void counter() {
  delay(10);
  ir_flag = 0;
  if (!digitalRead(button)) {
    PushCounter++;
  }
}

void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(KEY, INPUT);//Button of remote sensing module
  ledcSetup(CHAN, 1000, 12);
  ledcAttachPin(PIN_LED, CHAN);
  pinMode(button, INPUT);//The key module
  attachInterrupt(digitalPinToInterrupt(button), counter, FALLING);  //External␣
↪interrupt 0, falling edge fired
  pinMode(Avoid, INPUT);//Obstacle avoidance sensor
  pinMode(Trig, OUTPUT);//Ultrasonic module
  pinMode(Echo, INPUT);
  adxl345.Init();
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
  delay(1000);
 }
}

void loop() {
  yushu = PushCounter % 8;
  if (yushu == 0) {  //The remainder is 0
    yushu_0();  //rgb displays
  } else if (yushu == 1) {  //The remainder is 1
    yushu_1();  //Displays the high and low levels read by the tracking sensor
  } else if (yushu == 2) {  //The remainder is 2
    yushu_2();  //Display temperature and humidity value
  } else if (yushu == 3) {  //The remainder is 3
    yushu_3();  //Displays the rocker value
  }else if (yushu == 4) {  //The remainder is 4
    yushu_4();  //Display potentiometer ADC value and potentiometer control LED
  } else if (yushu == 5) {  //The remainder is 5
```

(continues on next page)

(continued from previous page)

```
    yushu_5();  //Obstacle avoidance sensor detects obstacles
  } else if (yushu == 6) {  //The remainder is 6
    yushu_6();  //Shows the distance detected by ultrasound
  } else if (yushu == 7) {  //The remainder is 7
    yushu_7();  //ADXL345 triaxial acceleration value
  }
}

//RGB
void yushu_0() {
  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);
}

void yushu_1() {
  int val = digitalRead(TrackingPin);//Read the digital level output by the tracking␣
↪sensor
  Serial.print(val);//Serial port print value
  if (val == 0) {//White val is 0 detected
    Serial.print("         ");
    Serial.println("White");
    delay(100);
  }
  else {//Black val is 1 detected
    Serial.print("         ");
    Serial.println("Black");
    delay(100);
  }
}

void yushu_2() {
  if (xht.receive(dht)) { //Returns true when checked correctly
    Serial.print("RH:");
    Serial.print(dht[0]); //The integral part of humidity, DHT [1] is the fractional part
    Serial.print("%  ");
    Serial.print("Temp:");
    Serial.print(dht[2]); //The integral part of temperature, DHT [3] is the fractional␣
↪part
    Serial.println("C");
  } else {    //read error
    Serial.println("sensor error");
  }
  delay(1200);
}
```

(continues on next page)

(continued from previous page)

```
void yushu_3() {
  int x = analogRead(X);
  int y = analogRead(Y);
  int key = digitalRead(KEY);
  Serial.print("X:");
  Serial.print(x);
  Serial.print("    Y:");
  Serial.print(y);
  Serial.print("    KEY:");
  Serial.println(key);
  delay(100);
}

void yushu_4() {
  int adcVal = analogRead(resPin); //read adc
  Serial.println(adcVal);
  int pwmVal = adcVal;        // adcVal re-map to pwmVal
  ledcWrite(CHAN, pwmVal);    // set the pulse width.
  delay(10);
}

void yushu_5() {
  int val = digitalRead(Avoid);
  if (val == 0) {//Obstruction detected
    Serial.println("There are obstacles");
  }
  else {//No obstructions detected
    Serial.println("All going well");
  }
  delay(100);
}

void yushu_6() {
  float distance = checkdistance();
  Serial.print("distance:");
  Serial.print(distance);
  Serial.println("cm");
  delay(100);
}

void yushu_7() {
  adxl345.readXYZ(&out_X, &out_Y, &out_Z);
  Serial.print(out_X);
  Serial.print("g   ");
  Serial.print(out_Y);
  Serial.print("g   ");
  Serial.print(out_Z);
  Serial.println("g");
  delay(100);
}
```

(continues on next page)

(continued from previous page)

```
float checkdistance() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  float distance = pulseIn(Echo, HIGH) / 58.00;
  delay(10);
  return distance;
}
//*************************************************************************************
```

Code Explanation

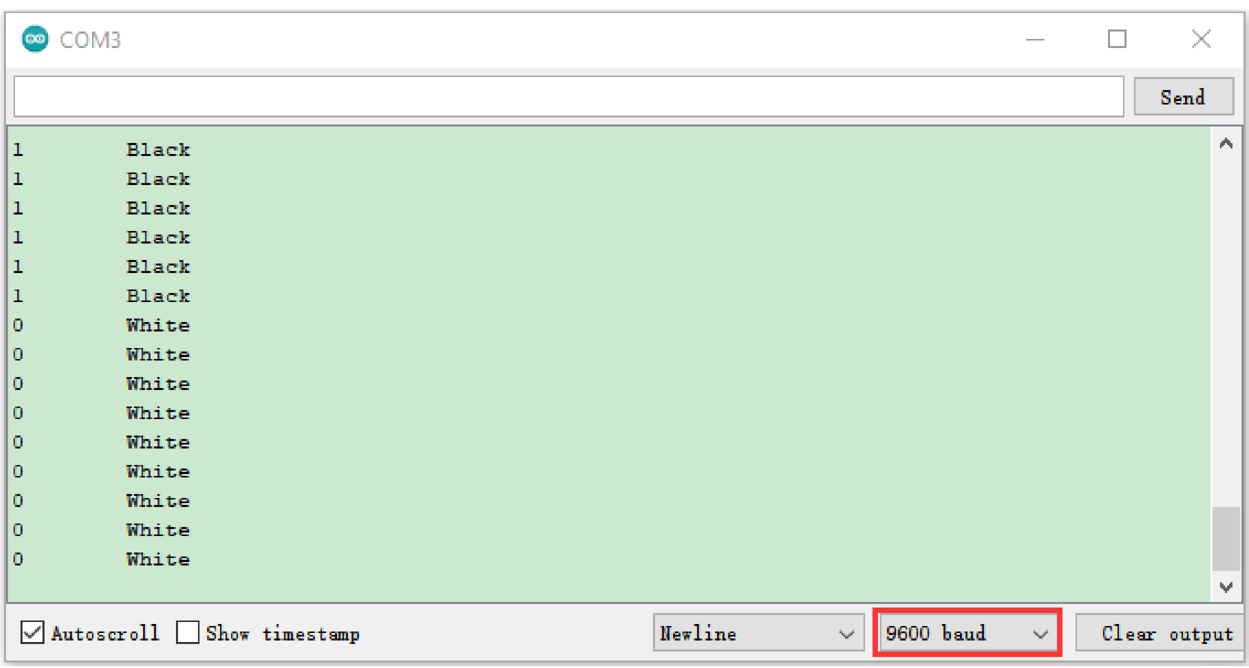
1). Calculate how many times the button is pressed, divide it by 8, and get the remainder which is 0, 1 2, 3, 4, 5 , 6 and 7. According to different remainders, construct eight unique functions to control the experiment and realize different functions.

2). Following the instructions, we can add or remove sensors/modules in the wiring, and then change the experimental function in the code.

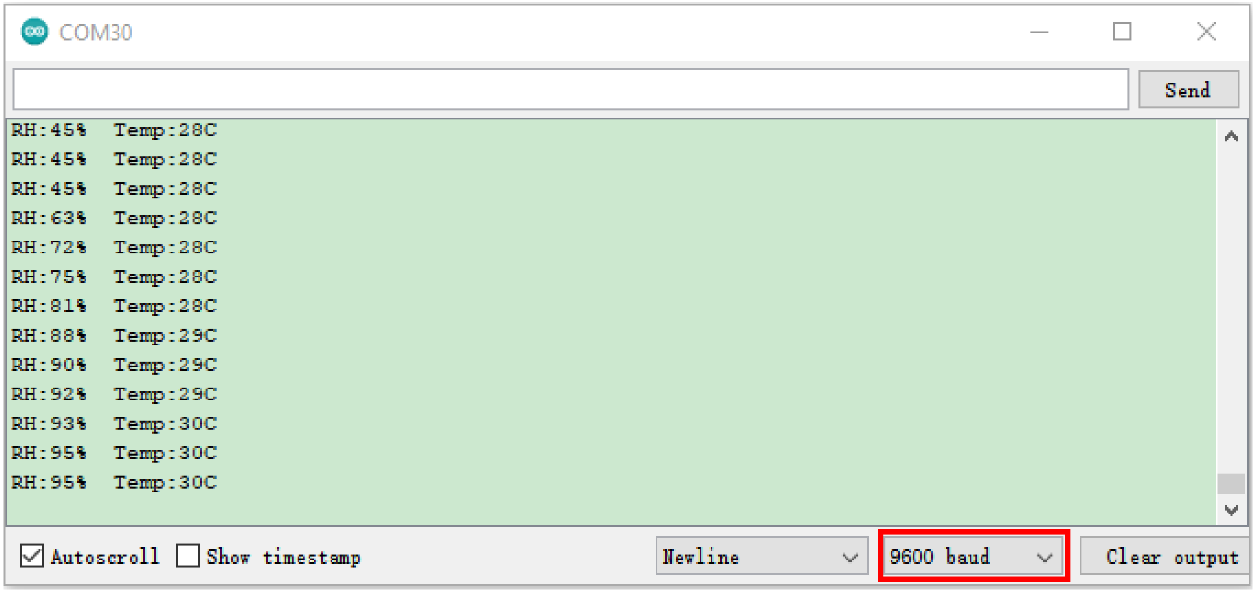
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. At the beginning, the number of the button is 0 and remainder is 0. Open the monitor and set baud rate to 9600.

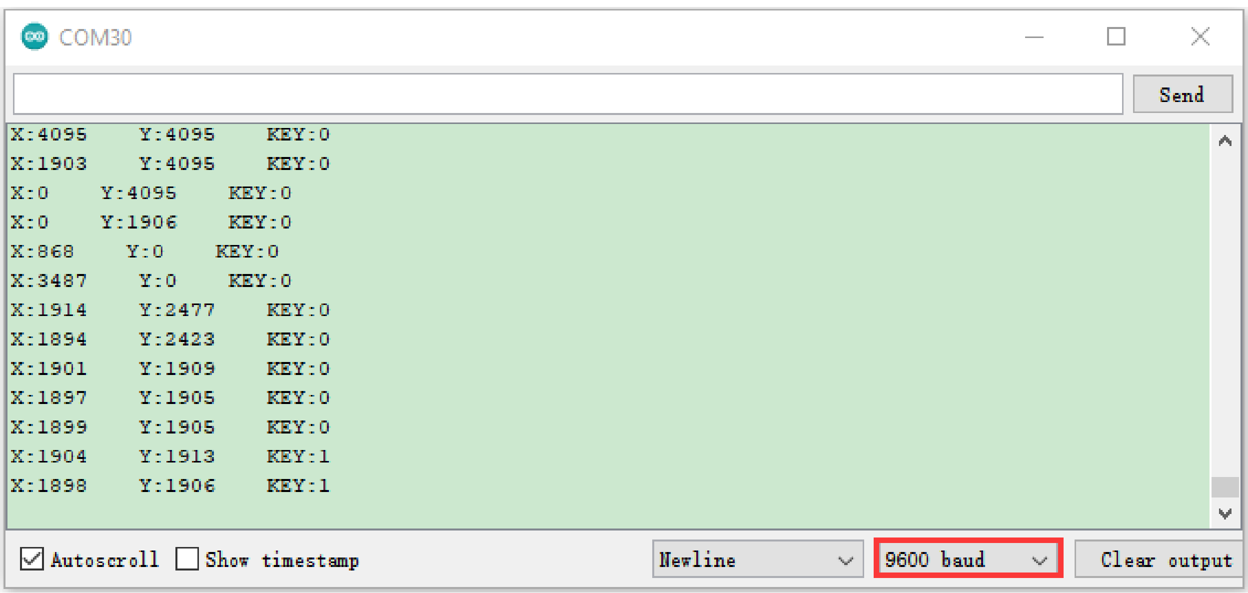
Press the button, the RGB stops flashing, press once, the remainder is 1. The function of the experiment is to detect black objects and white objects by a line tracking sensor. If the sensor does not detect an object or detects a black object, val is 1, and the serial monitor displays the character "1 Black". When a white object (reflective) is detected, val is 0 and the serial monitor displays the character "0 White", the serial monitor will display as follows:



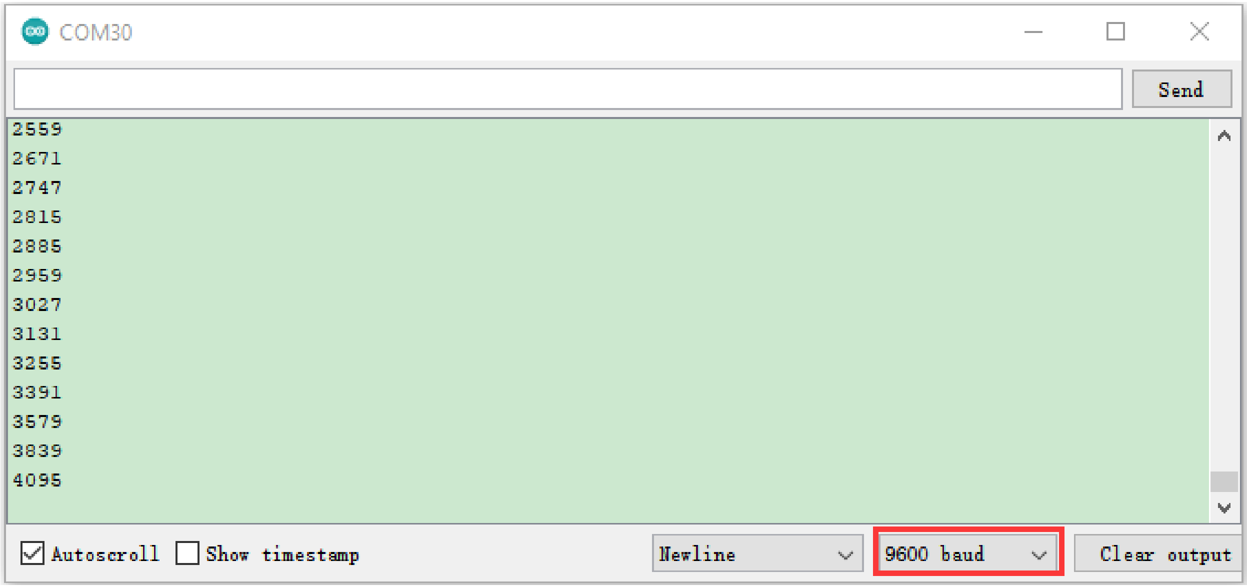
Press a key twice, the time of pressing buttons is 2 and the remainder is 2. Read temperature and humidity values. As shown below:



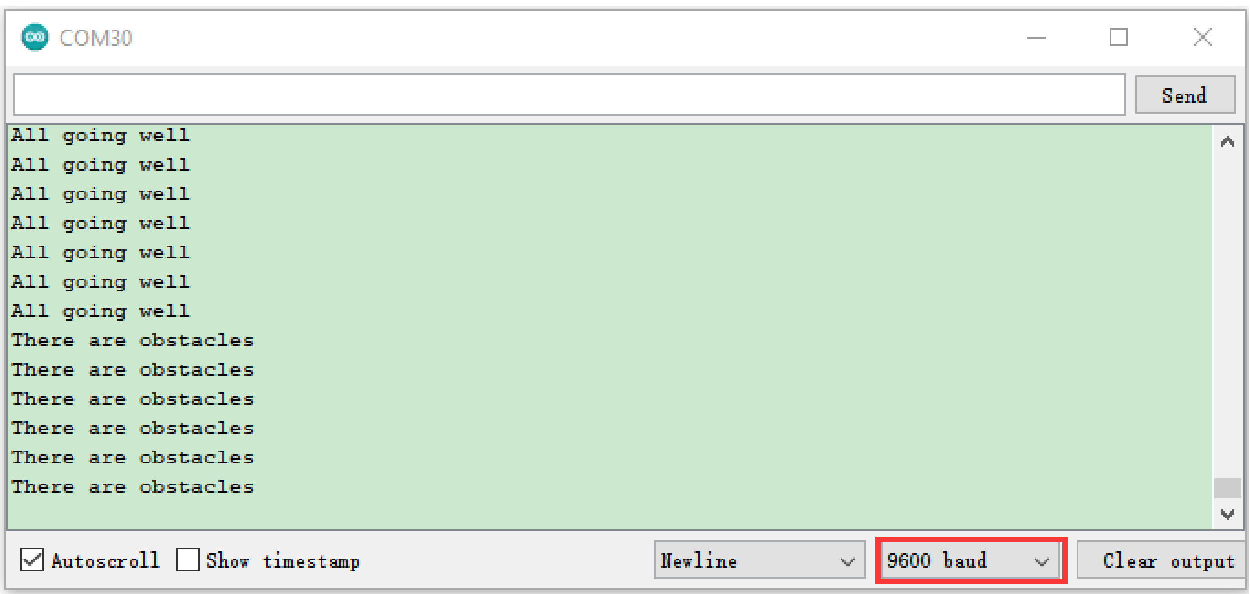
Press a key again, the time of pressing buttons is 3 and the remainder is 3. Read digital values at x, y and z axis of the joystick module. As shown below:



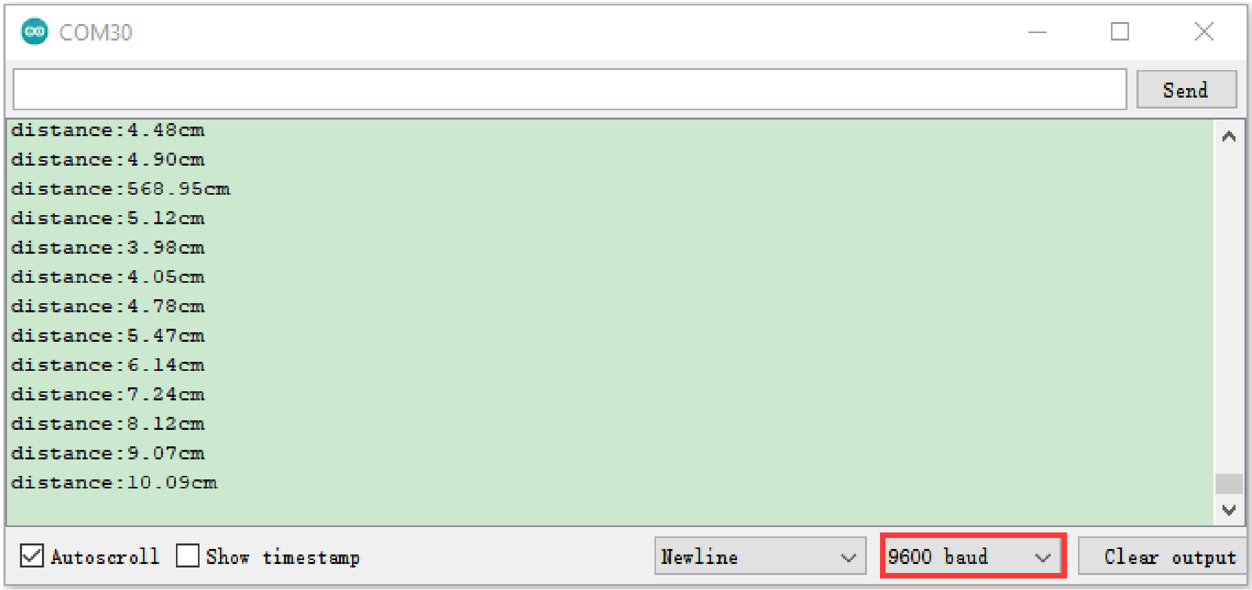
Press the key for the fourth time, the remainder is 4. Then the potentiometer can adjust the PWM value at the GPI05 port to control LED brightness of the purple LED.



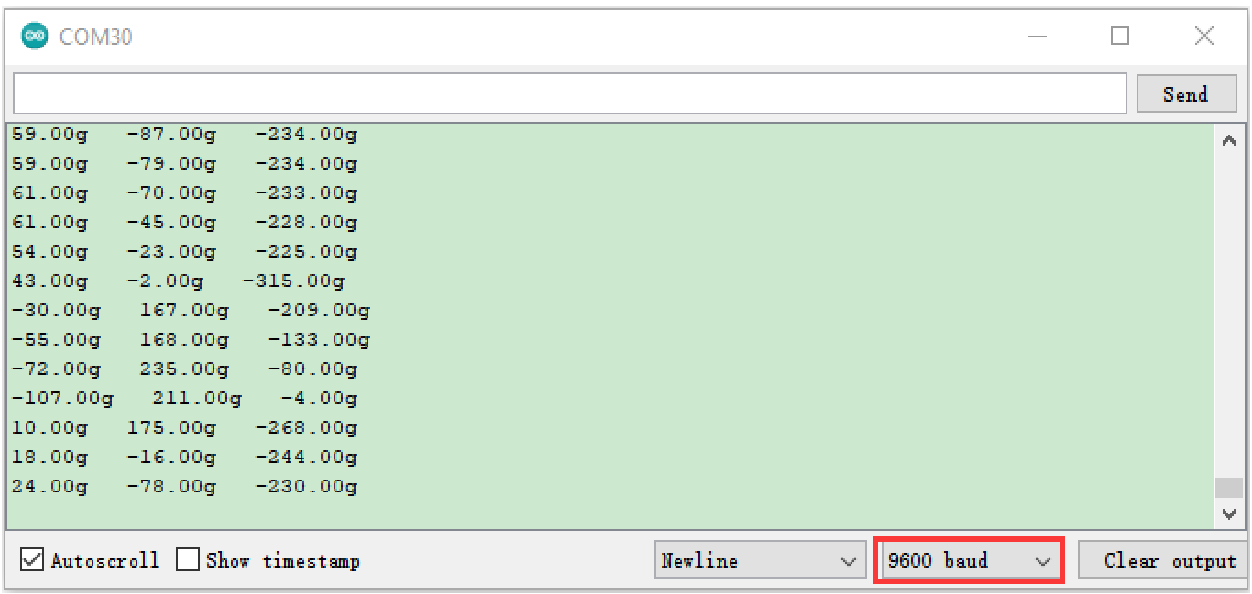
Press the key for the fifth time, the remainder is 5. Then the ultrasonic sensor can detect obstacles, as shown below:



Press the key for the sixth time, the remainder is 6. Then the ultrasonic sensor can detect distance away from obstacles, as shown below:



Press the key for seventh time and the remainder is 7. The monitor will print out the acceleration values.



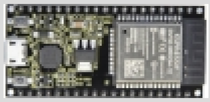
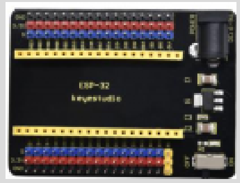
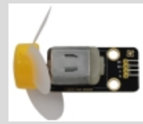
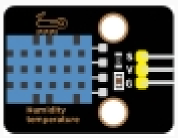
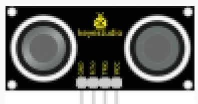
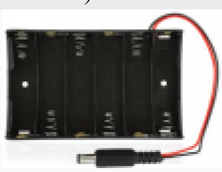
Press the key for eighth time and the remainder is 0. Then the RGB will flash. If you press keys incessantly, remainders will change in a loop way. So does functions.

6.3.21 Project 65: WiFi

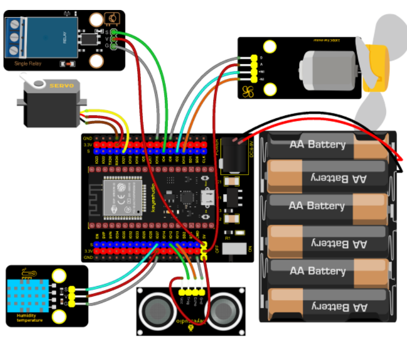
Description

In the previous experiment, we have learned the WiFi Station mode, WiFi AP mode and WiFi AP+Station mode of the ESP32. In this project, We will use ESP32's WiFi Station mode to control the work of multiple sensors/modules through APP connection with WiFi to achieve the effect of WiFi smart home.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 130 Motor*1	Keyestudio5V Relay Module*1	Servo*1
				
Keyestudio XHT11Temperature and Humidity Sensor*1compatible DHT11)	Keyestudio HC-SR04 Ultrasonic Sensor*1	3P Dupont*2	4P Dupont*2	Smart Phone/PC*1
				
Battery Holder*1	Battery (provided by yourself)*6	Micro USB Cable*1		

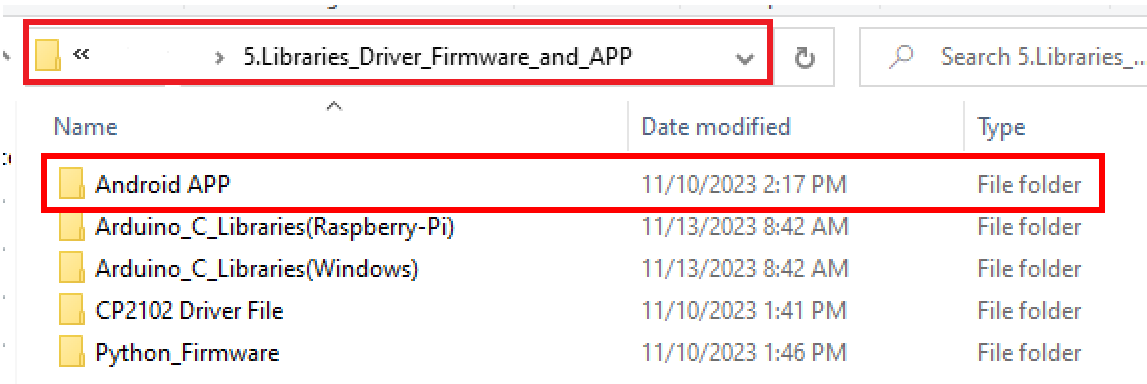
Wiring Diagram



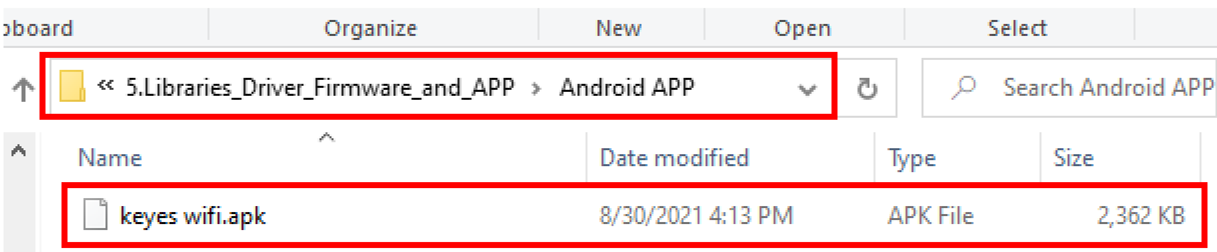
Install APP

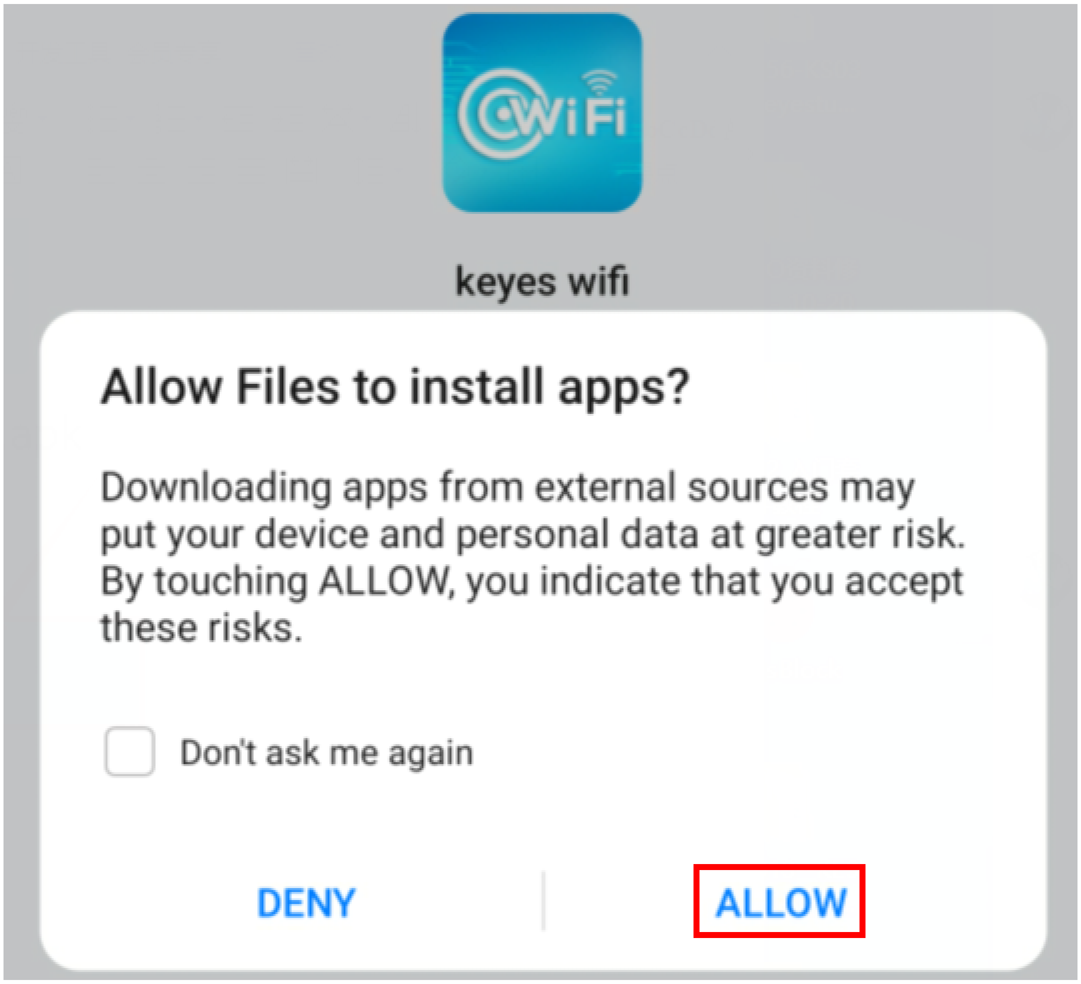
(1) Android device (mobile phone/PC) APP:

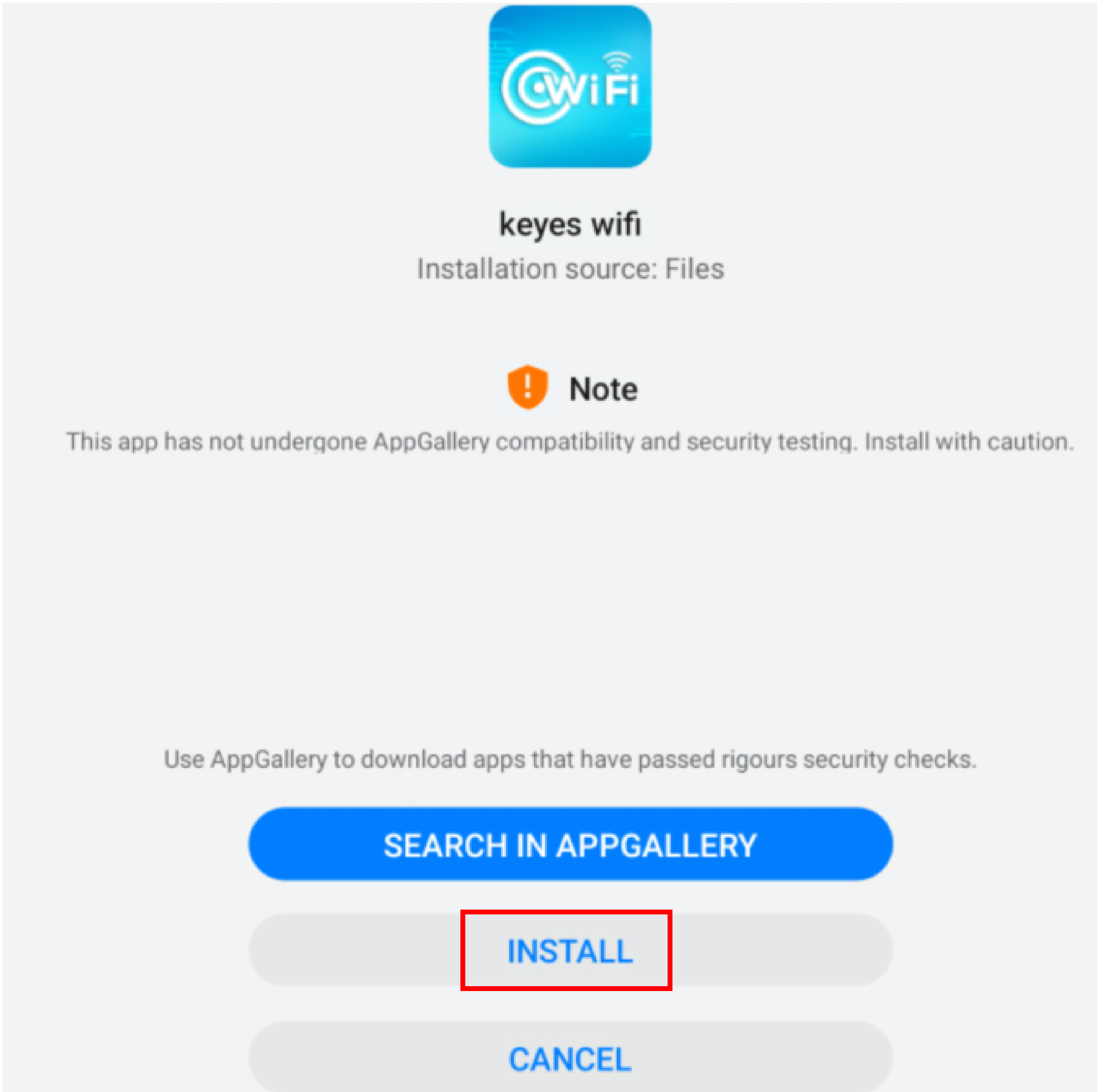
A. We provide the Android APP installation package:

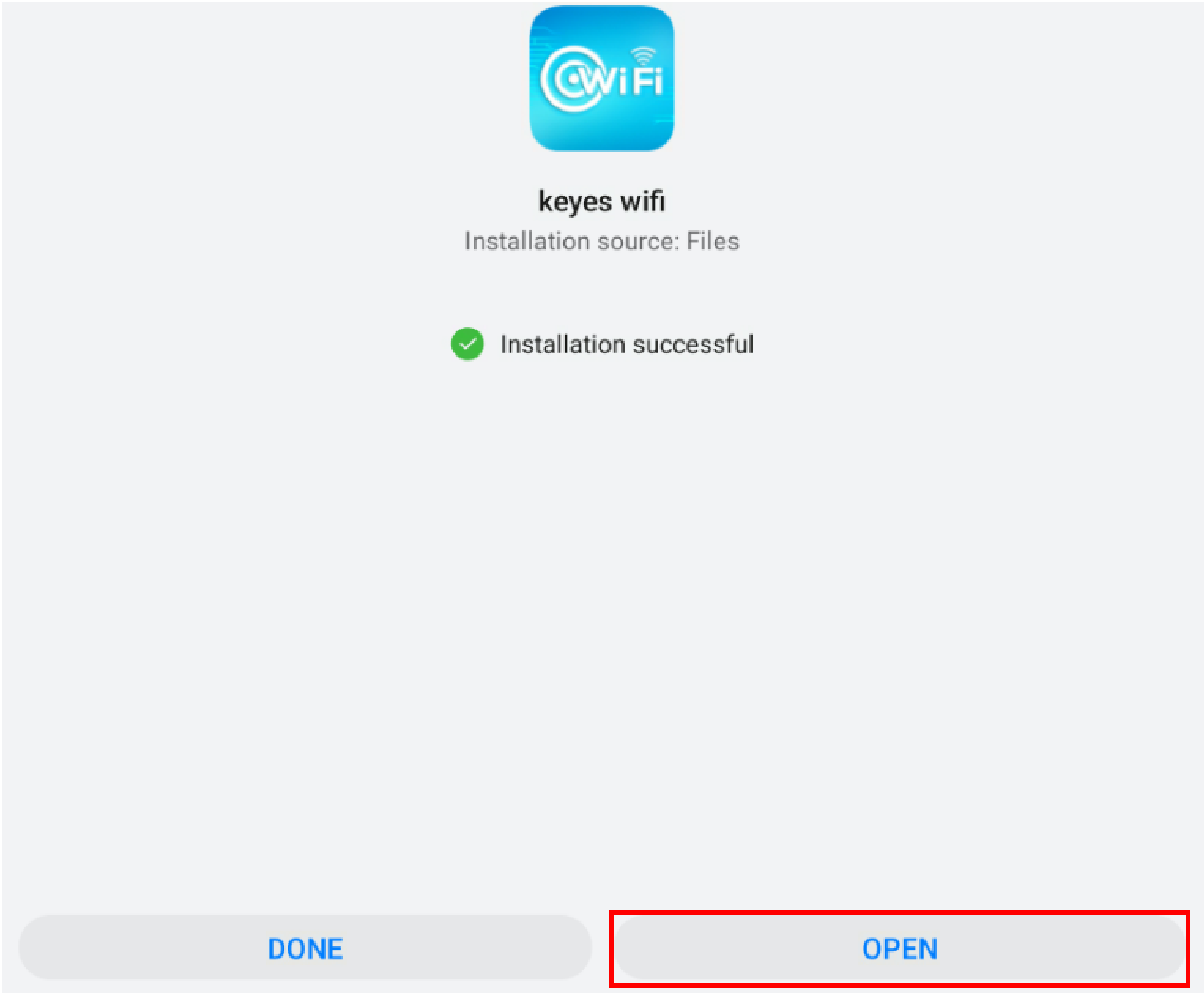


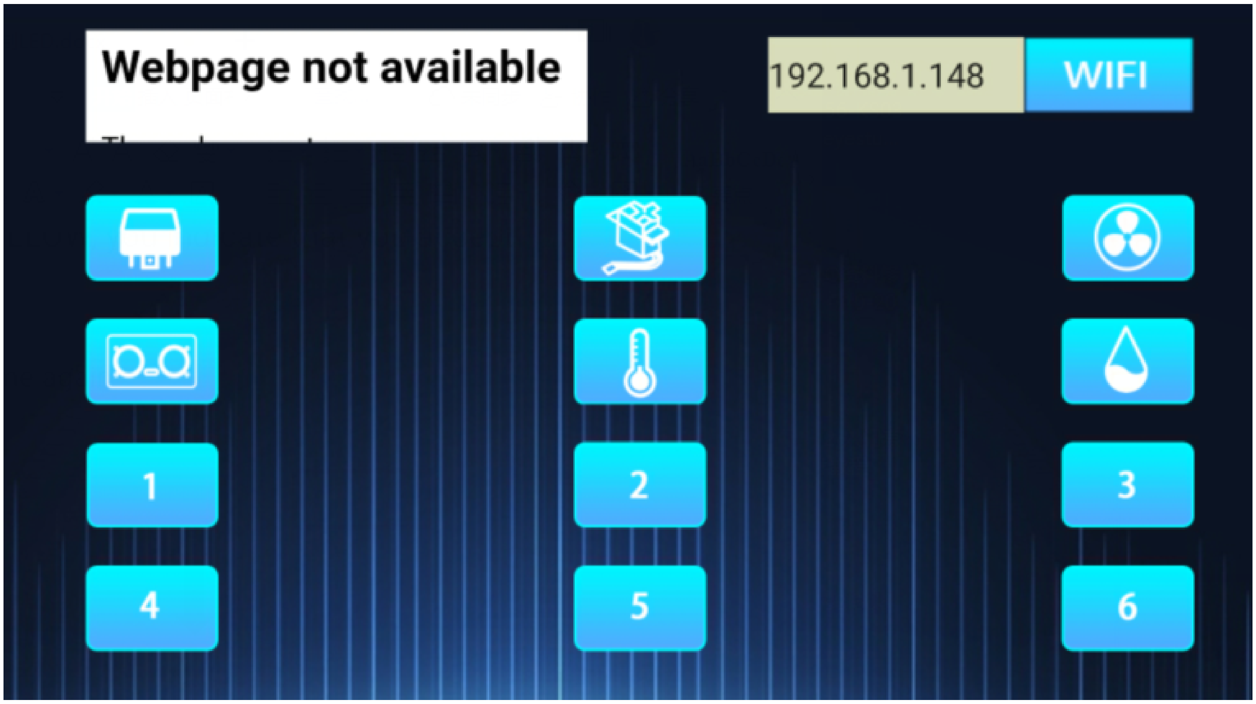
B. Now transfer the **keyes wifi.apk** file in the Android APP installation package to the Android phone or PC, click the **keyes wifi.apk** file to enter the installation page, click "**ALLOW**" key, and then click "**INSTALL**" button. After installation, click "**OPEN**" button to enter the APP interface.




keyes wifi
Allow Files to install apps?
Downloading apps from external sources may put your device and personal data at greater risk. By touching ALLOW, you indicate that you accept these risks.
Don't ask me again
DENY
ALLOW


WiFi
keyes wifi
Installation source: Files
Note
This app has not undergone AppGallery compatibility and security testing. Install with caution.
Use AppGallery to download apps that have passed rigours security checks.
SEARCH IN APPGALLERY
INSTALL
CANCEL


WiFi
keyes wifi
Installation source: Files
Installation successful
DONE
OPEN



(2) IOS device (mobile phone /iPad) APP:

A. Open App Store

B. Enter keyes link in the search box and click search, the download interface appears. Click “  ” to download and install the APP of the keyes link. The following operations are similar to those of Android system. You can refer to the steps of Android system above for operation.

Test Code

Note: You need to change the Wifi name and default Wifi password of the experimental code to your own Wifi name and Wifi password.

```
const char* ssid = "ChinaNet-2.4G-0DF0"; //the name of user's wifi
const char* password = "ChinaNet@233";  //the password of user's wifi
```

```
//**********************************************************************************
/*
 * Filename    : WiFi Smart Home.
 * Description : WiFi APP controls Multiple sensors/modules work to achieve the effect␣
↪of WiFi smart home.
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
```

(continues on next page)

(continued from previous page)

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

#include "xht11.h"
//gpio15
xht11 xht(27);
unsigned char dht[4] = {0, 0, 0, 0};

#include <ESP32Servo.h>
Servo myservo;
int servoPin = 21;
#define Relay  4
#define IN1 2  //IN1 corresponds to IN+
#define IN2 15 //IN2 corresponds to IN-
#define trigPin  12
#define echoPin  13

int distance1;
String dis_str;
int ip_flag = 1;
int ultra_state = 1;
int temp_state = 1;
int humidity_state = 1;

String item = "0";
const char* ssid = "ChinaNet-2.4G-0DF0"; //the name of user's wifi
const char* password = "ChinaNet@233";   //the password of user's wifi
WiFiServer server(80);
String unoData = "";

void setup() {
  Serial.begin(115200);
  pinMode(Relay, OUTPUT);
  myservo.setPeriodHertz(50);
  myservo.attach(servoPin, 500, 2500);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
  Serial.println("TCP server started");
  MDNS.addService("http", "tcp", 80);
```

(continues on next page)

(continued from previous page)

```
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(Relay, LOW);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  WiFiClient client = server.available();
  if (!client) {
      return;
  }
  while(client.connected() && !client.available()){
      delay(1);
  }
  String req = client.readStringUntil('\r');
  int addr_start = req.indexOf(' ');
  int addr_end = req.indexOf(' ', addr_start + 1);
  if (addr_start == -1 || addr_end == -1) {
      Serial.print("Invalid request: ");
      Serial.println(req);
      return;
  }
  req = req.substring(addr_start + 1, addr_end);
  item=req;
  Serial.println(item);
  String s;
  if (req == "/")
  {
      IPAddress ip = WiFi.localIP();
      String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' +
↪String(ip[3]);
      s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>
↪Hello from ESP32 at ";
      s += ipStr;
      s += "</html>\r\n\r\n";
      Serial.println("Sending 200");
      client.println(s);
  }
  else if(req == "/btn/0")
  {
    Serial.write('a');
    client.println(F("turn on the relay"));
    digitalWrite(Relay, HIGH);
  }
  else if(req == "/btn/1")
  {
    Serial.write('b');
    client.println(F("turn off the relay"));
    digitalWrite(Relay, LOW);
  }
```

(continues on next page)

(continued from previous page)

```
else if(req == "/btn/2")
{
  Serial.write('c');
  client.println("Bring the steering gear over 180 degrees");
  myservo.write(180);
  delay(200);
}
else if(req == "/btn/3")
{
  Serial.write('d');
  client.println("Bring the steering gear over 0 degrees");
  myservo.write(0);
  delay(200);
}
else if(req == "/btn/4")
{
  Serial.write('e');
  client.println("esp32 already turn on the fans");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
}
else if(req == "/btn/5")
{
  Serial.write('f');
  client.println("esp32 already turn off the fans");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}
else if(req == "/btn/6")
{
  Serial.write('g');
  while(Serial.available() > 0)
  {
    unoData = Serial.readStringUntil('#');
    client.println("Data");
  }
  while(ultra_state>0)
     {
        Serial.print("Distance = ");
        Serial.print(checkdistance());
        Serial.println("#");
        Serial1.print("Distance = ");
        Serial1.print(checkdistance());
        Serial1.println("#");
        int t_val1 = checkdistance();
        client.print("Distance(cm) = ");
        client.println(t_val1);
        ultra_state = 0;
      }
}
else if(req == "/btn/7")
{
```

(continues on next page)

(continued from previous page)

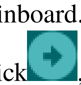
```
    Serial.write('h');
    client.println("turn off the ultrasonic");
    ultra_state = 1;
  }
  else if(req == "/btn/8")
  {
    Serial.write('i');
    while(Serial.available() > 0)
     {
      unoData = Serial.readStringUntil('#');
      client.println(unoData);
     }
    while(temp_state>0)
      {
        if (xht.receive(dht)) {
          Serial.print("Temperature = ");
          Serial.print(dht[2],1);
          Serial.println("#");
          Serial1.print("Temperature = ");
          Serial1.print(dht[2],1);
          Serial1.println("#");
          int t_val2 = dht[2];
          client.print("Temperature(℃) = ");
          client.println(t_val2);
        }
        temp_state = 0;
      }
  }
  else if(req == "/btn/9")
  {
    Serial.write('j');
    client.println("turn off the temperature");
    temp_state = 1;
  }
  else if(req == "/btn/10")
  {
    Serial.write('k');
    while(Serial.available() > 0)
     {
       unoData = Serial.readStringUntil('#');
       client.println(unoData);
     }
    while(humidity_state > 0)
      {
        if (xht.receive(dht)) {
          Serial.print("Humidity = ");
          Serial.print(dht[0],1);
          Serial.println("#");
          Serial1.print("Humidity = ");
          Serial1.print(dht[0],1);
          Serial1.println("#");
          int t_val3 = dht[0];
```

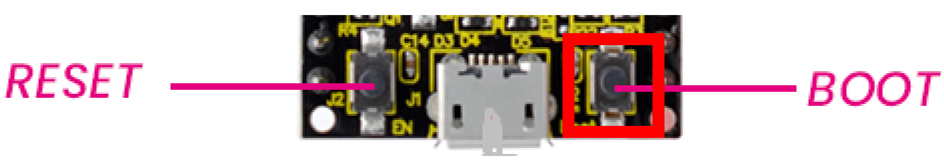
(continues on next page)

(continued from previous page)

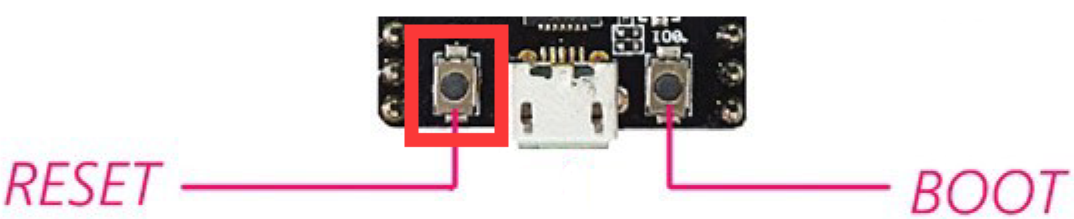
```
          client.print("Humidity(%) = ");
          client.println(t_val3);
        }
        humidity_state = 0;
      }
  }
  else if(req == "/btn/11")
  {
    Serial.write('l');
    client.println("turn off the humidity");
    humidity_state = 1;
    }
  //client.print(s);
  client.stop();
}

int checkdistance() {
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(10);
  digitalWrite(12, LOW);
  int distance = pulseIn(13, HIGH) / 58;

  delay(10);
  return distance;
}
//*********************************************************************************
```

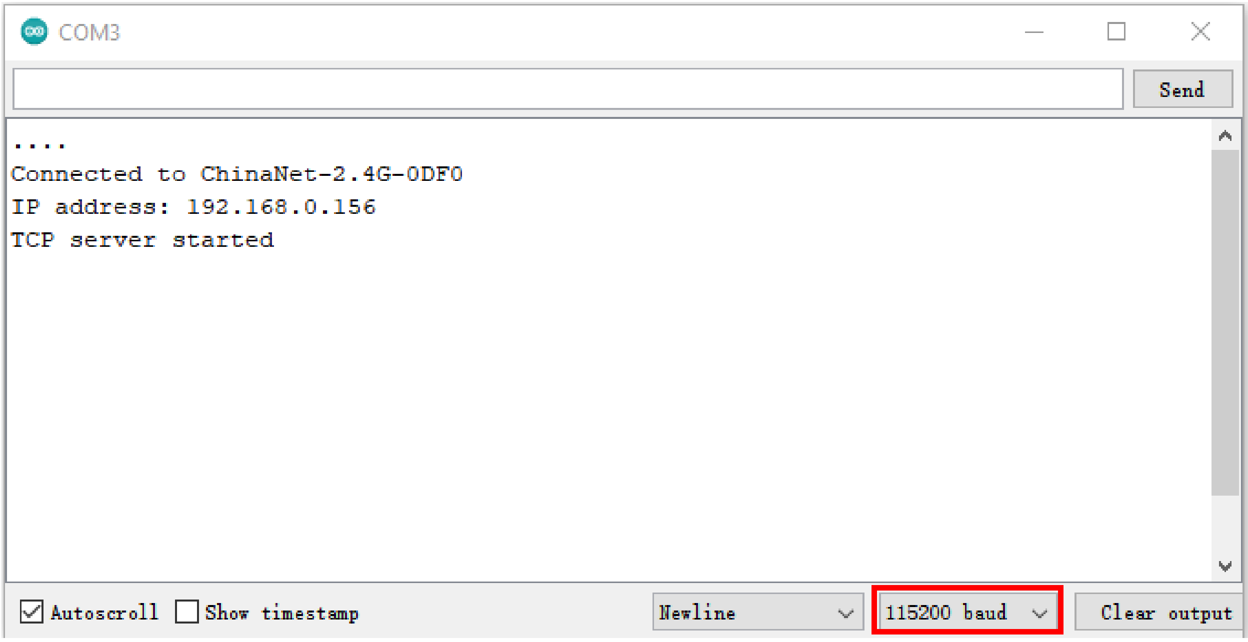
Test Result

After the code has been modified correctly, connect the external power supply and power on. Switch the DIP switch ON the ESP32 expansion board to the ON end, compile and upload the code to the ESP32 mainboard.If uploading the code is not successfulpress the Boot button on the ESP32 mainboard with your hand after click, release it when the upload progress percentage appears.)

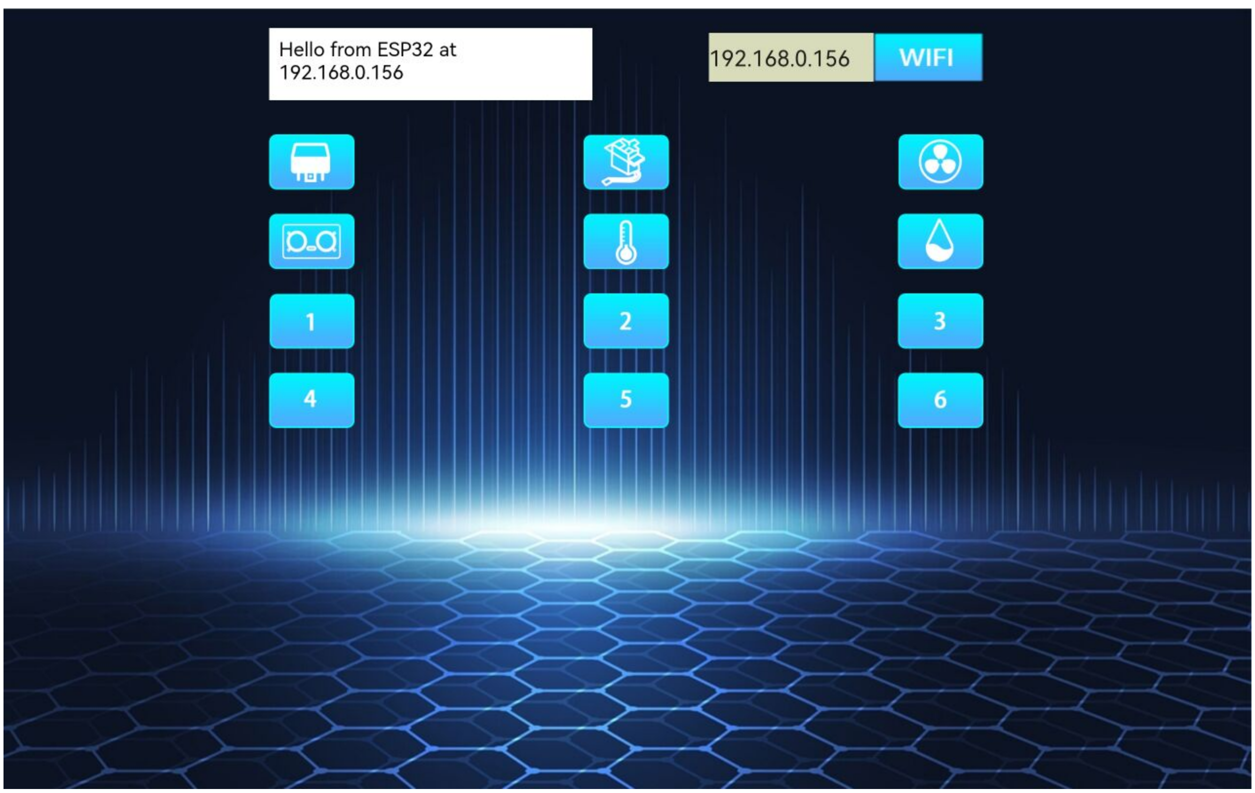


Open the serial monitor and set baud rate to 115200, then the monitor prints the detected WiFi IP address. (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)





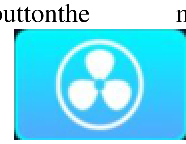
Open WiFi APP, enter the detected WIFI IP address in the text box in front of the WIFI button (for example, the IP address detected by the serial monitor above is 192.168.0.156). Next, click the WIFI button to connect to WIFI, at the same time, the corresponding WiFi IP address will be displayed in the text box :"Hello from ESP32 at 192.168.0.156", then the APP has connected to WiFi.(WiFi IP address sometimes changes, if the original IP address can not use, you need to re-check it.)

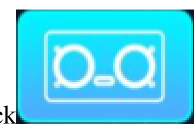


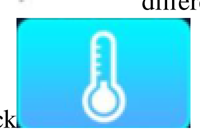
After the APP is connected to WiFi, the following operations are performed:

1) Click  button, the relay will be opened, the APP will display turn on the relay and the indicator lights up on the module. Click  again, the relay will be closed , the APP will display turn off the relay and the indicator on the module is off.

2) Click  buttonthe servo rotates 180°the APP will display  againthe APP will display Bring the steering gear over 0 degrees the servo rotates 0°.

3) Click  buttonthe motorwith small fan bladesrotatesthe APP will display  againclose the motorthe APP will display esp32 already turn off the fans

4) Click  buttonthe ultrasonic sensor detects the distance, put an object in front of the ultrasonic sensor, the APP will display Distance(cm) = 6 different distances show different numbers, the distance between the object and the ultrasonic sensor is 14cmclick  again, turn off the sensor, the APP will display turn off the ultrasonic .

5) Click  buttonthe temperature and humidity sensor measures the temperature in the environment, the APP will display Temperature(℃) = 30 different temperatures show different temperature valuesthe ambient temperature is 28 ° C., click  again, turn off the sensorthe APP will display turn off the temperature .

6) Click  buttonthe temperature and humidity sensor measures the humidity in the environment,the APP will display `Humidity(%) = 55` different humidity show different humidity values, the ambient humidity is 52%clickagainturn off the sensor, the APP will display `turn off the humidity`.

CHAPTER

SEVEN

ARDUINO(RASPBERRY-PI) TUTORIAL

7.1 1. Install Raspberry Pi OS System

7.1.1 1.1. Hardware Tool

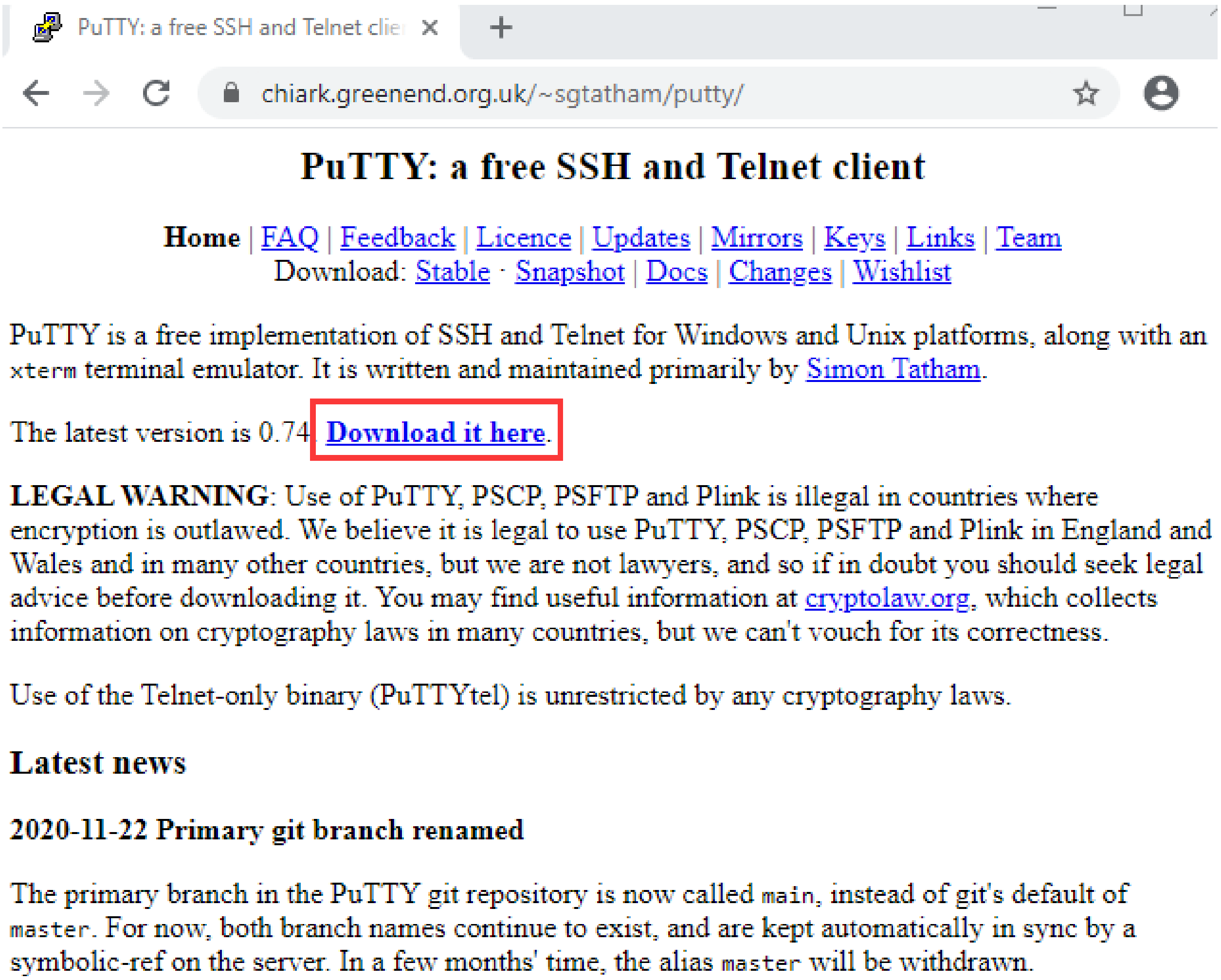
- Raspberry Pi 4B/3B/2B
- Above 8G TFT SD Card
- Card Reader
- Computer and other parts

7.1.2 1.2. Software Tool

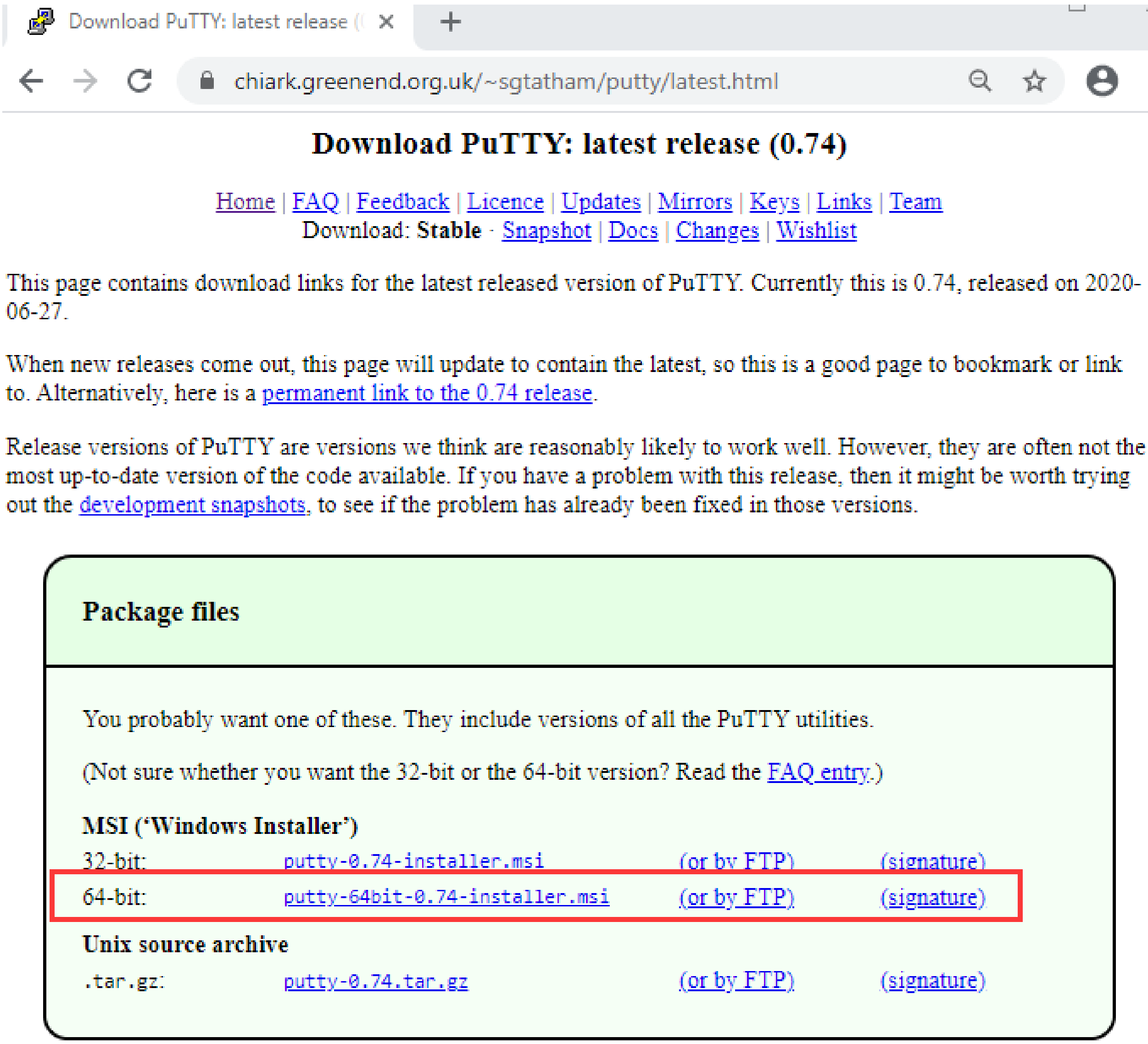
Windows System

(1) Install putty:

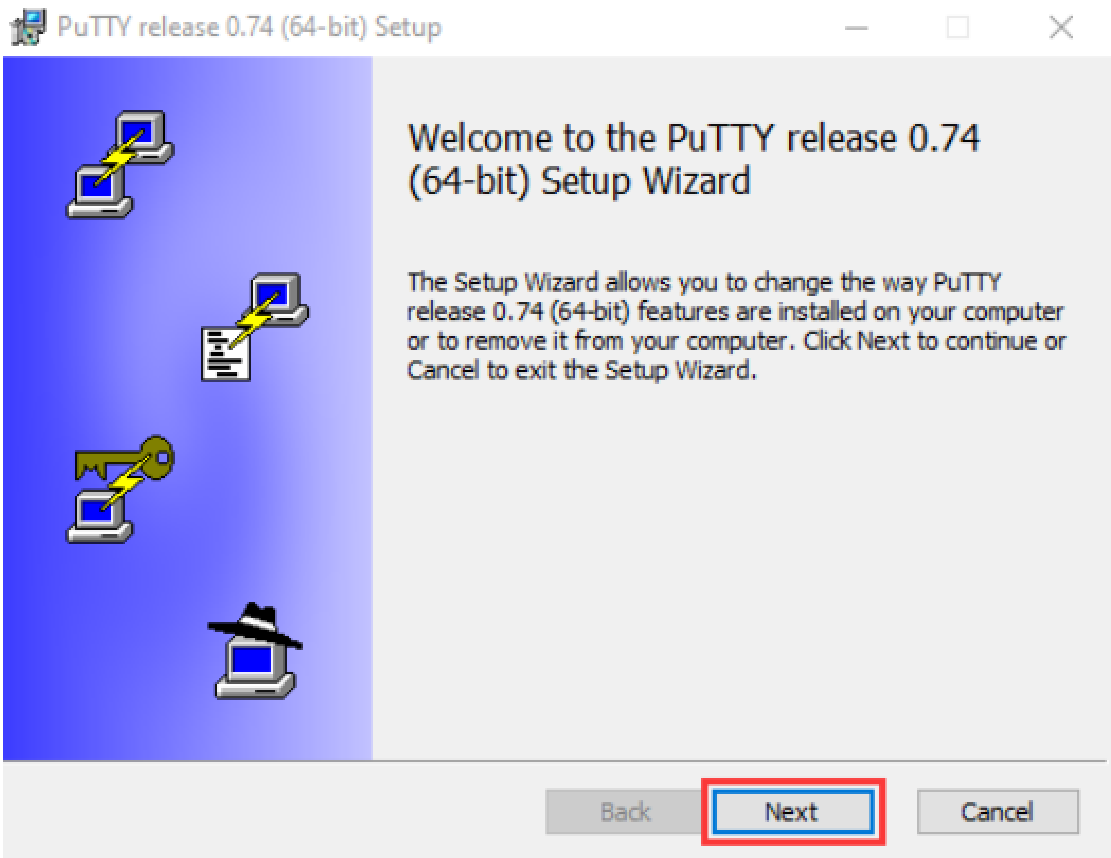
Download Putty: https://www.chiark.greenend.org.uk/~sgtatham/putty/



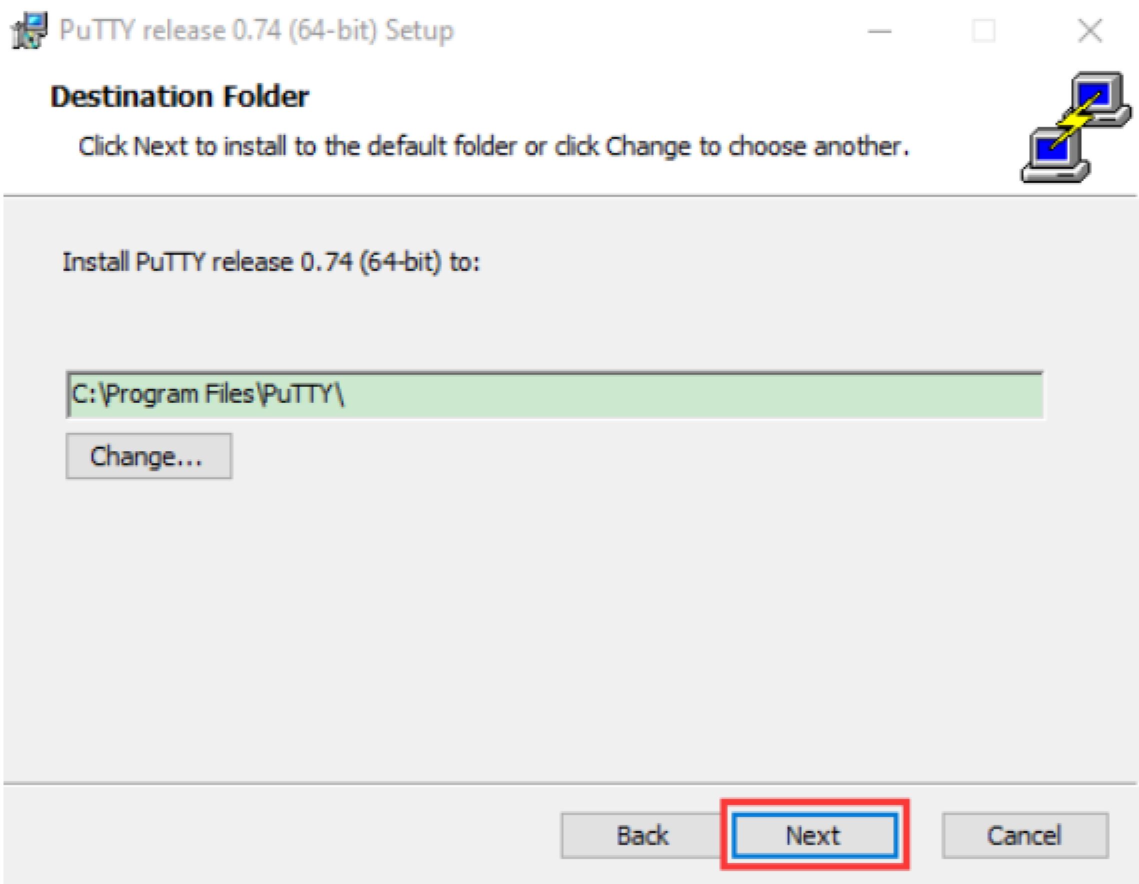
PuTTY: a free SSH and Telnet clie
chiark.greenend.org.uk/~sgtatham/putty/
PuTTY: a free SSH and Telnet client
Home | FAQ | Feedback | Licence | Updates | Mirrors | Keys | Links | Team
Download: Stable · Snapshot | Docs | Changes | Wishlist
PuTTY is a free implementation of SSH and Telnet for Windows and Unix platforms, along with an xterm terminal emulator. It is written and maintained primarily by Simon Tatham.
The latest version is 0.74. Download it here.
LEGAL WARNING: Use of PuTTY, PSCP, PSFTP and Plink is illegal in countries where encryption is outlawed. We believe it is legal to use PuTTY, PSCP, PSFTP and Plink in England and Wales and in many other countries, but we are not lawyers, and so if in doubt you should seek legal advice before downloading it. You may find useful information at cryptolaw.org, which collects information on cryptography laws in many countries, but we can't vouch for its correctness.
Use of the Telnet-only binary (PuTTYtel) is unrestricted by any cryptography laws.
Latest news
2020-11-22 Primary git branch renamed
The primary branch in the PuTTY git repository is now called main, instead of git's default of master. For now, both branch names continue to exist, and are kept automatically in sync by a symbolic-ref on the server. In a few months' time, the alias master will be withdrawn.



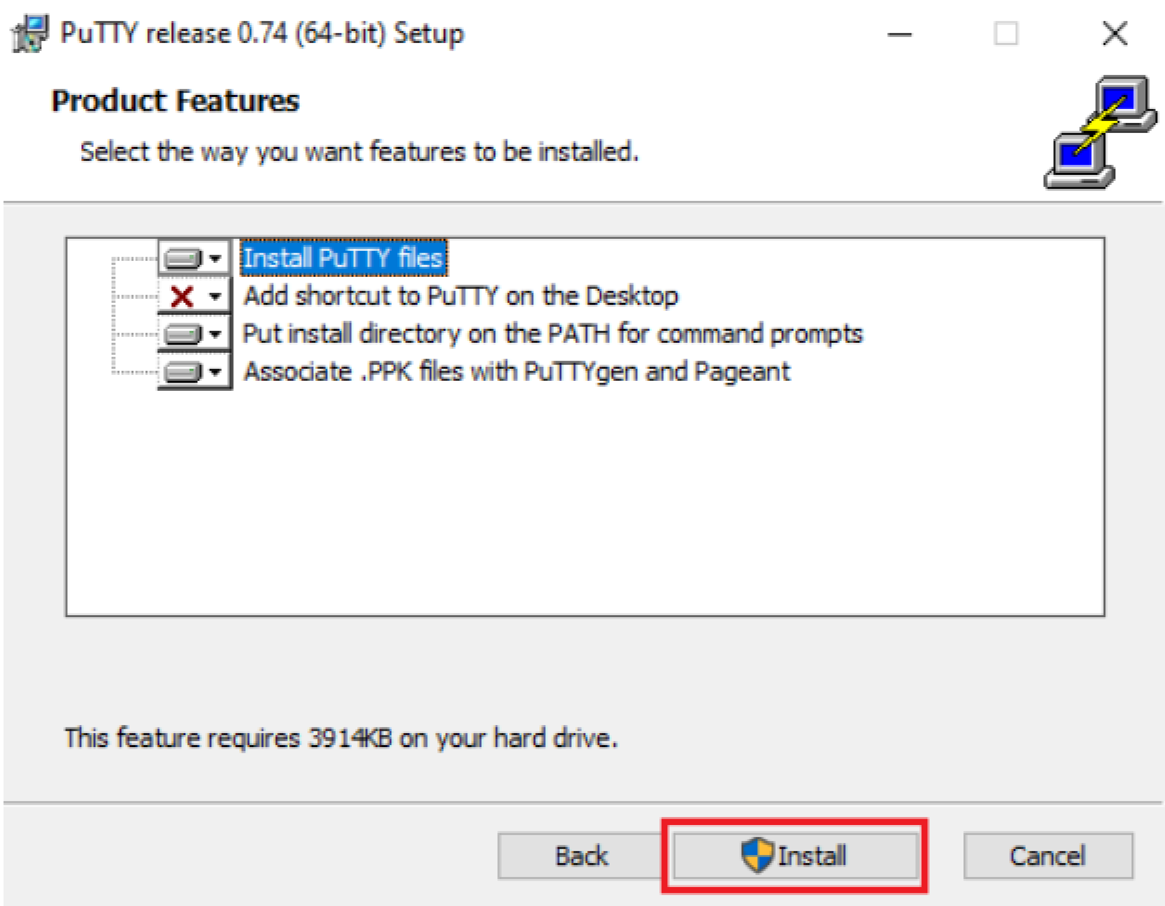
After downloading the driver file putty-64bit-0.74-installer double-click it and tap "Next".


PuTTY release 0.74 (64-bit) Setup
Welcome to the PuTTY release 0.74 (64-bit) Setup Wizard
The Setup Wizard allows you to change the way PuTTY release 0.74 (64-bit) features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel

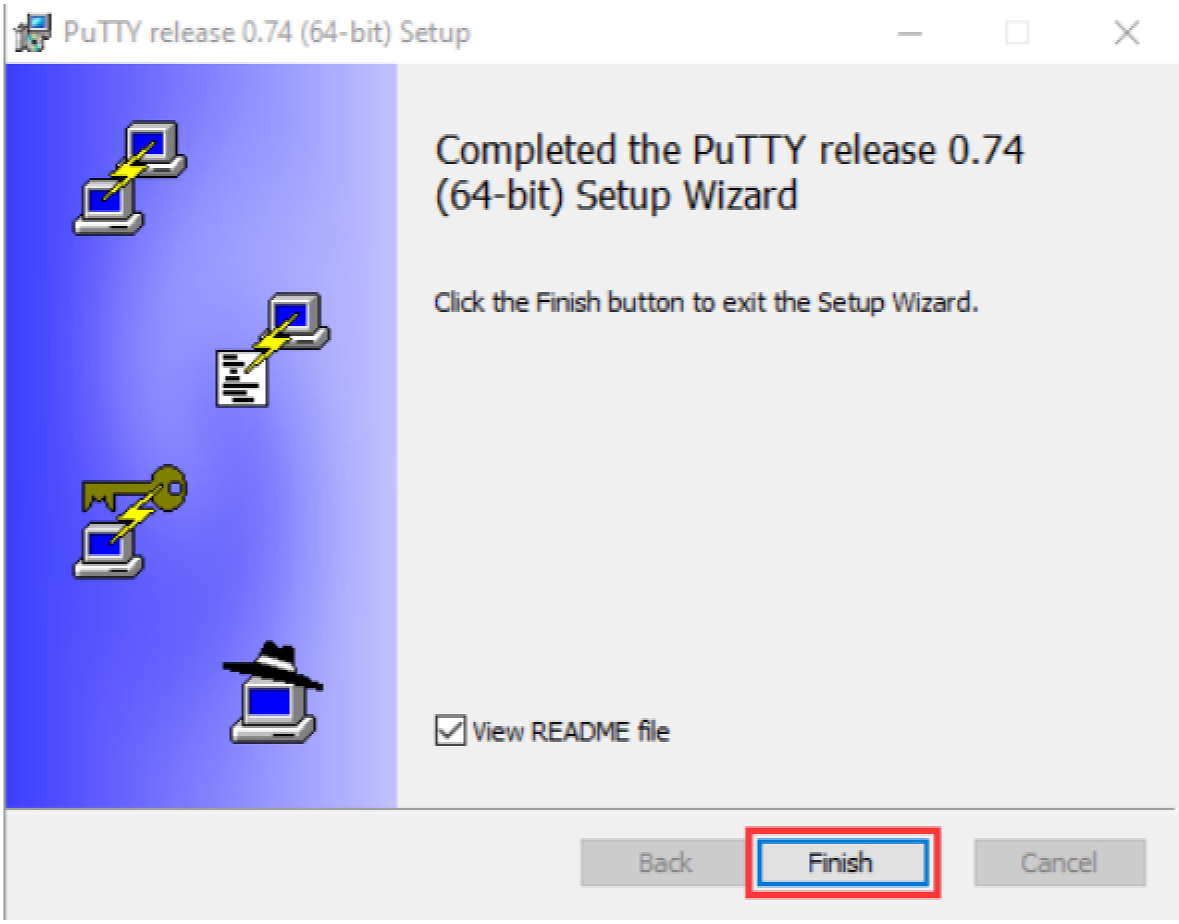
Click "Next".



Select "Install Putty files" and click "Install".

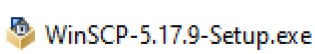


After a few seconds, click "Finish".



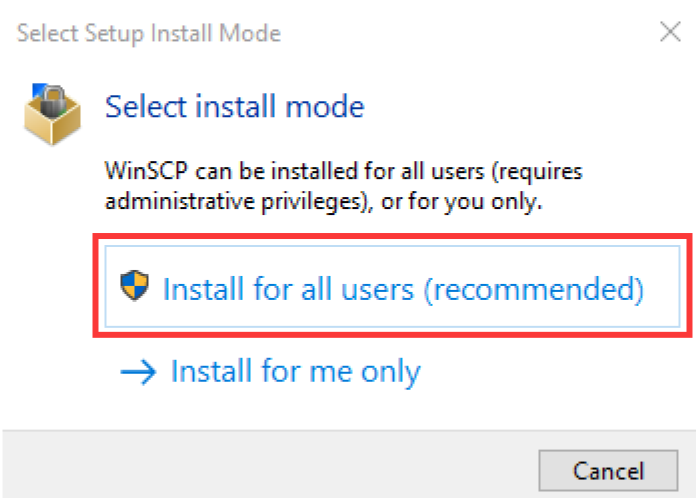
(2) SSH Remote Login software -WinSCP

Download WinSCP: https://winscp.net/eng/download.php

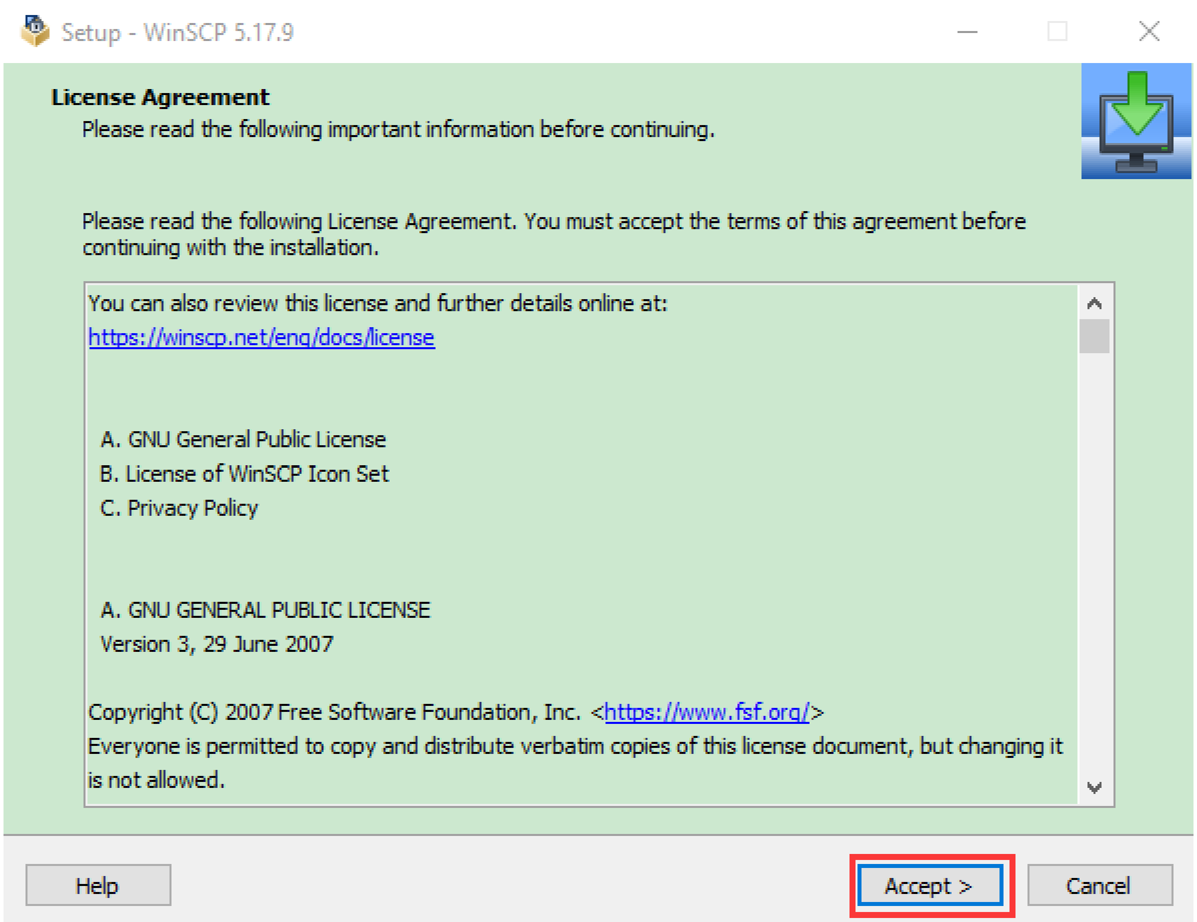
After the download, click 

and 

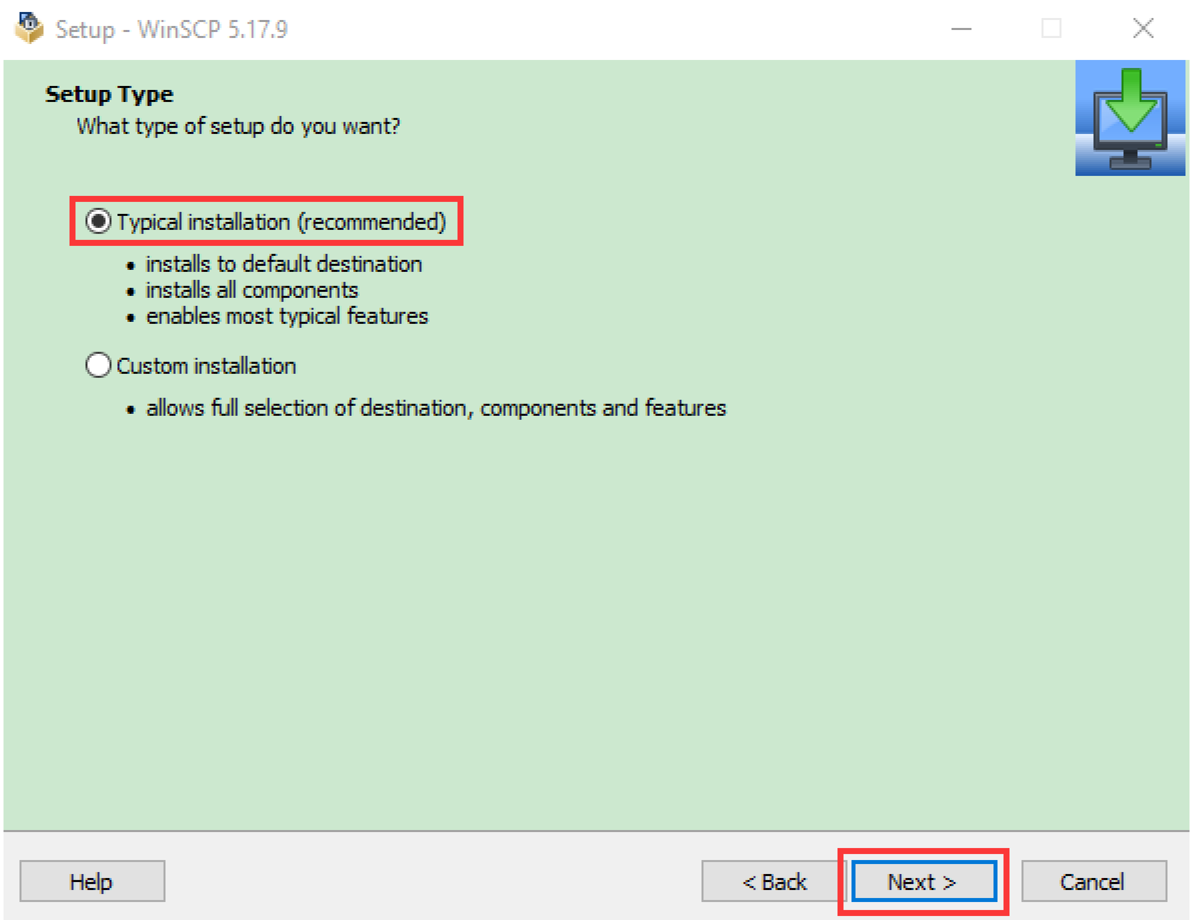
.

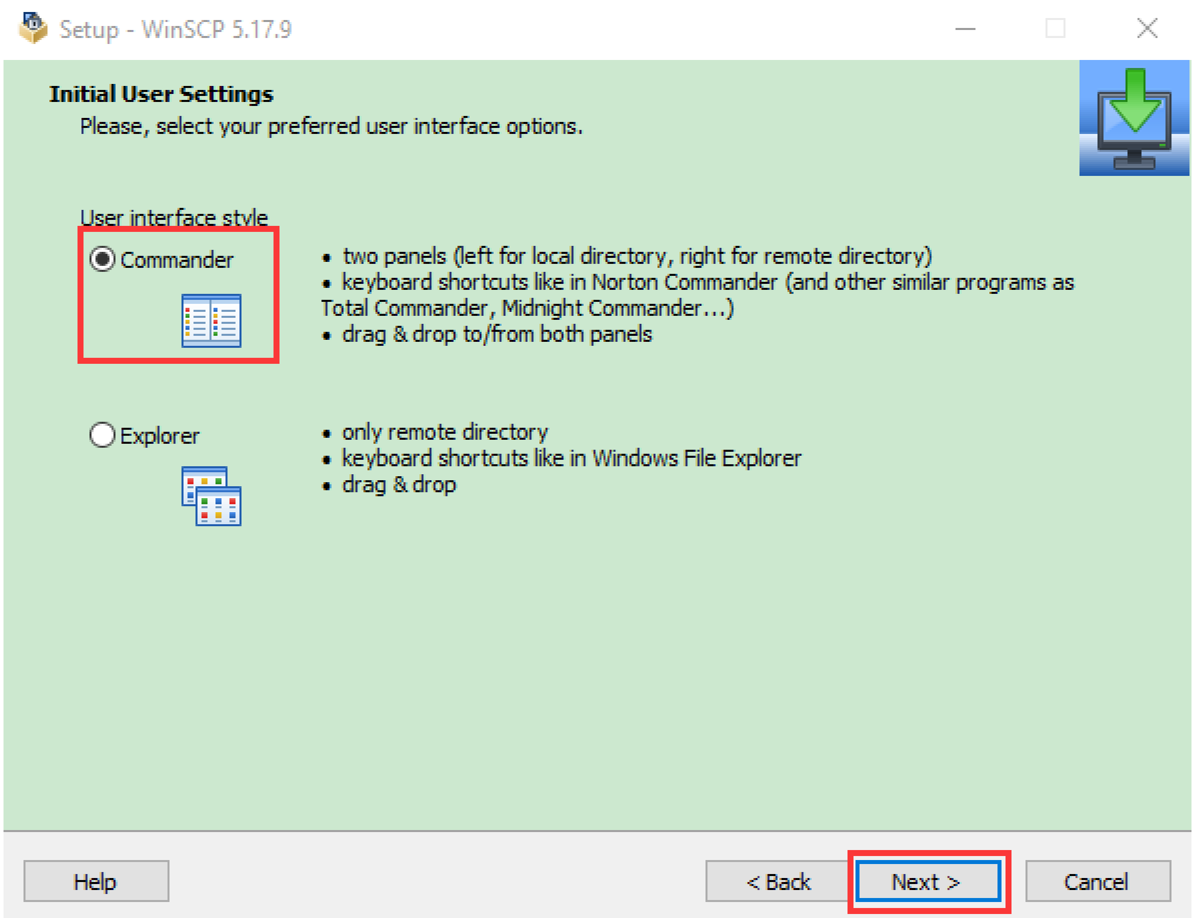


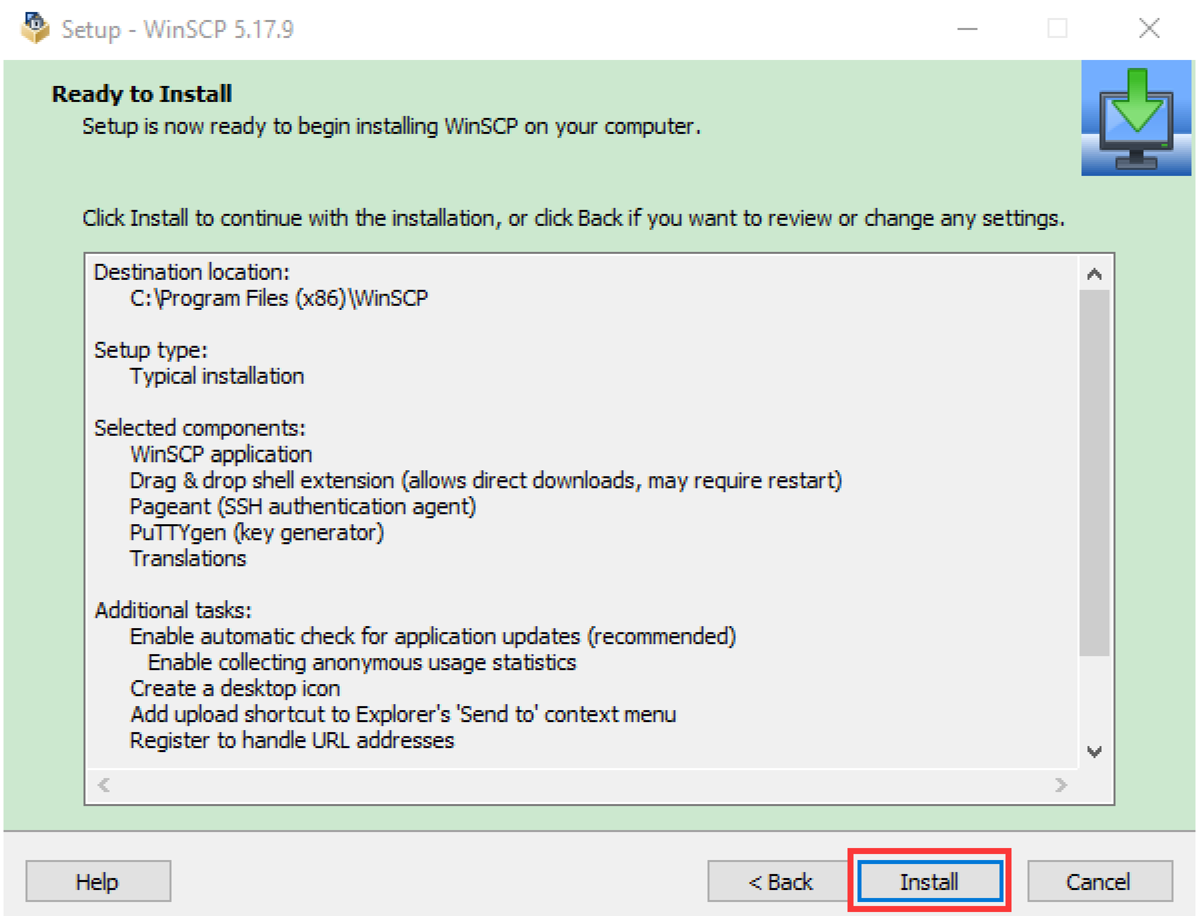
Click "Accept".



Follow the below steps to finish the installation.


Setup - WinSCP 5.17.9
Setup Type
What type of setup do you want?
Typical installation (recommended)
• installs to default destination
• installs all components
• enables most typical features
Custom installation
• allows full selection of destination, components and features
Help
< Back
Next >
Cancel


Setup - WinSCP 5.17.9
Initial User Settings
Please, select your preferred user interface options.
User interface style
Commander
• two panels (left for local directory, right for remote directory)
• keyboard shortcuts like in Norton Commander (and other similar programs as Total Commander, Midnight Commander…)
• drag & drop to/from both panels
Explorer
• only remote directory
• keyboard shortcuts like in Windows File Explorer
• drag & drop
Help
< Back
Next >
Cancel

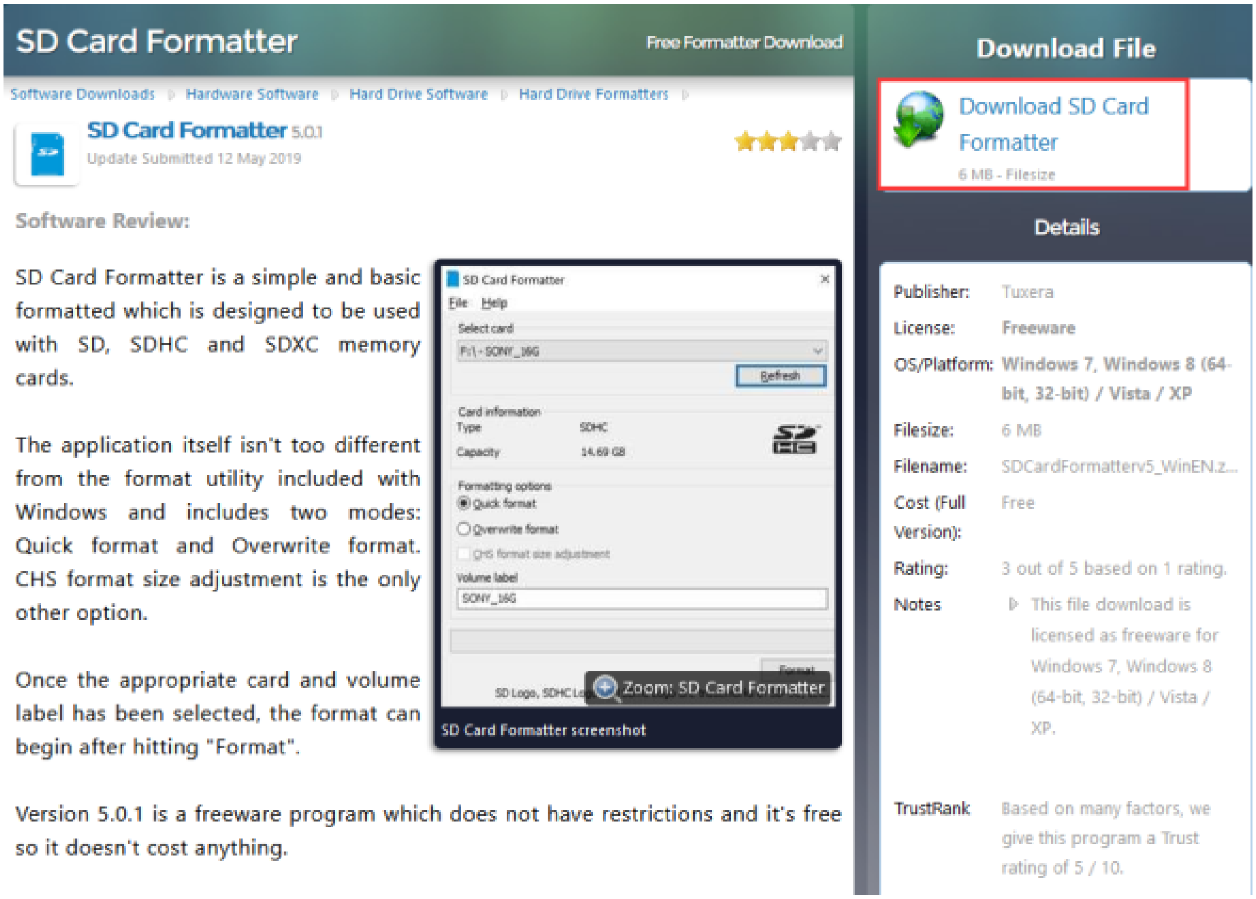

Setup - WinSCP 5.17.9
Ready to Install
Setup is now ready to begin installing WinSCP on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files (x86)\WinSCP
Setup type:
Typical installation
Selected components:
WinSCP application
Drag & drop shell extension (allows direct downloads, may require restart)
Pageant (SSH authentication agent)
PuTTYgen (key generator)
Translations
Additional tasks:
Enable automatic check for application updates (recommended)
Enable collecting anonymous usage statistics
Create a desktop icon
Add upload shortcut to Explorer's 'Send to' context menu
Register to handle URL addresses
Help
< Back
Install
Cancel

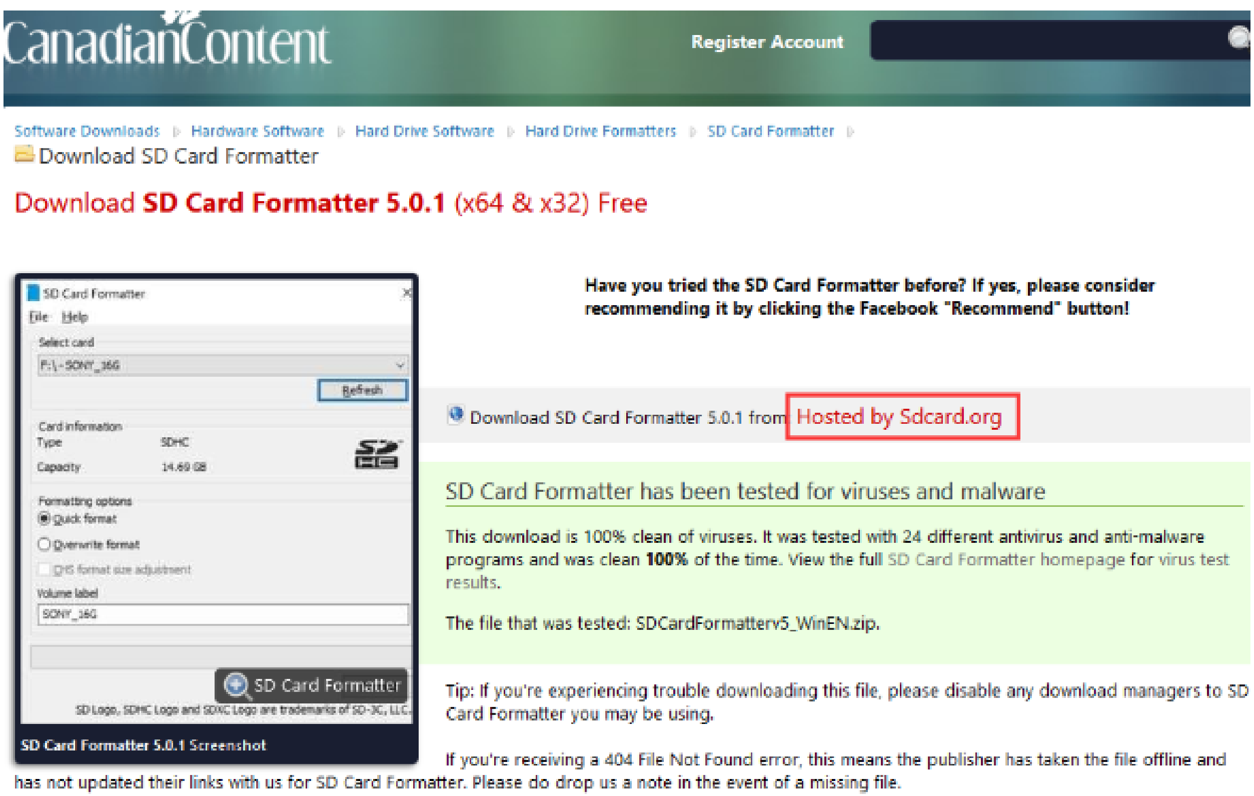
(3) SD Card Formatter

Format TFT card tool

Download SD Card Formatter

http://www.canadiancontent.net/tech/download/SD_Card_Formatter.html

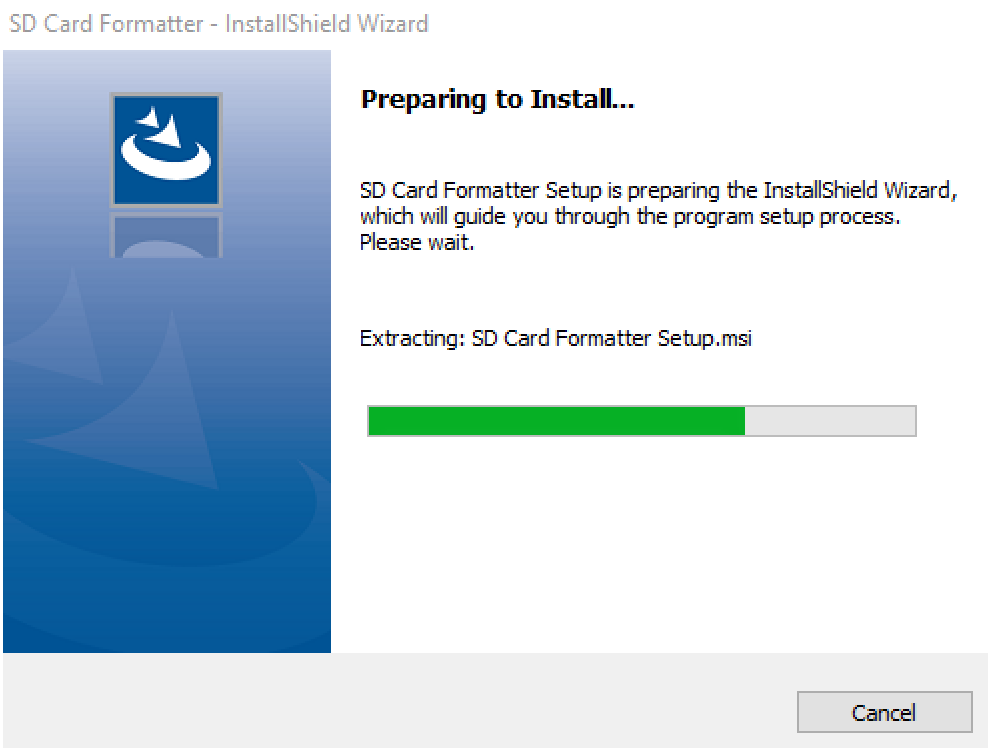

SD Card Formatter
Free Formatter Download
Software Downloads > Hardware Software > Hard Drive Software > Hard Drive Formatters
SD Card Formatter 5.0.1
Update Submitted 12 May 2019
Software Review:
SD Card Formatter is a simple and basic formatted which is designed to be used with SD, SDHC and SDXC memory cards.
The application itself isn't too different from the format utility included with Windows and includes two modes: Quick format and Overwrite format. CHS format size adjustment is the only other option.
Once the appropriate card and volume label has been selected, the format can begin after hitting "Format".
Version 5.0.1 is a freeware program which does not have restrictions and it's free so it doesn't cost anything.
Zoom: SD Card Formatter
SD Card Formatter screenshot
Download File
Download SD Card Formatter
6 MB - Filesize
Details
Publisher: Tuxera
License: Freeware
OS/Platform: Windows 7, Windows 8 (64-bit, 32-bit) / Vista / XP
Filesize: 6 MB
Filename: SDCardFormatterv5_WinEN.z...
Cost (Full Version): Free
Rating: 3 out of 5 based on 1 rating.
Notes: This file download is licensed as freeware for Windows 7, Windows 8 (64-bit, 32-bit) / Vista / XP.
TrustRank: Based on many factors, we give this program a Trust rating of 5 / 10.

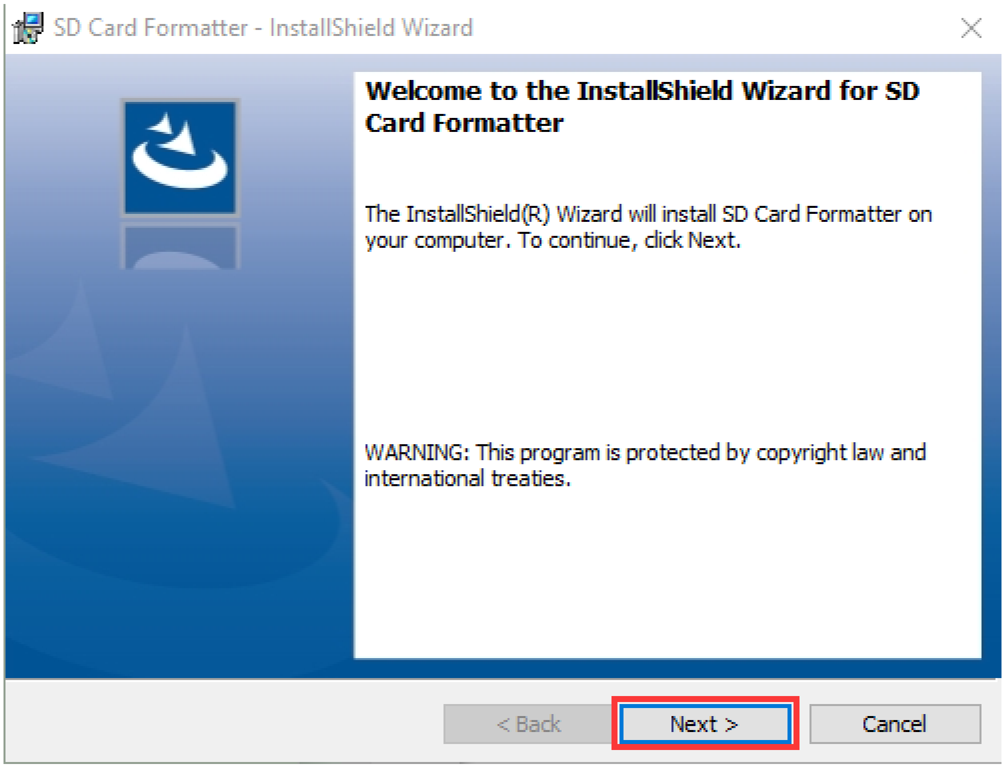

CanadianContent
Register Account
Software Downloads > Hardware Software > Hard Drive Software > Hard Drive Formatters > SD Card Formatter
Download SD Card Formatter
Download SD Card Formatter 5.0.1 (x64 & x32) Free
SD Card Formatter
SD Card Formatter 5.0.1 Screenshot
Have you tried the SD Card Formatter before? If yes, please consider recommending it by clicking the Facebook "Recommend" button!
Download SD Card Formatter 5.0.1 from Hosted by Sdcard.org
SD Card Formatter has been tested for viruses and malware
This download is 100% clean of viruses. It was tested with 24 different antivirus and anti-malware programs and was clean 100% of the time. View the full SD Card Formatter homepage for virus test results.
The file that was tested: SDCardFormatterv5_WinEN.zip.
Tip: If you're experiencing trouble downloading this file, please disable any download managers to SD Card Formatter you may be using.
If you're receiving a 404 File Not Found error, this means the publisher has taken the file offline and has not updated their links with us for SD Card Formatter. Please do drop us a note in the event of a missing file.

Unzip the SDCardFormatterv5_WinEN package, double-click SD Card Formatter 5.0.1 Setup.exe to run it.

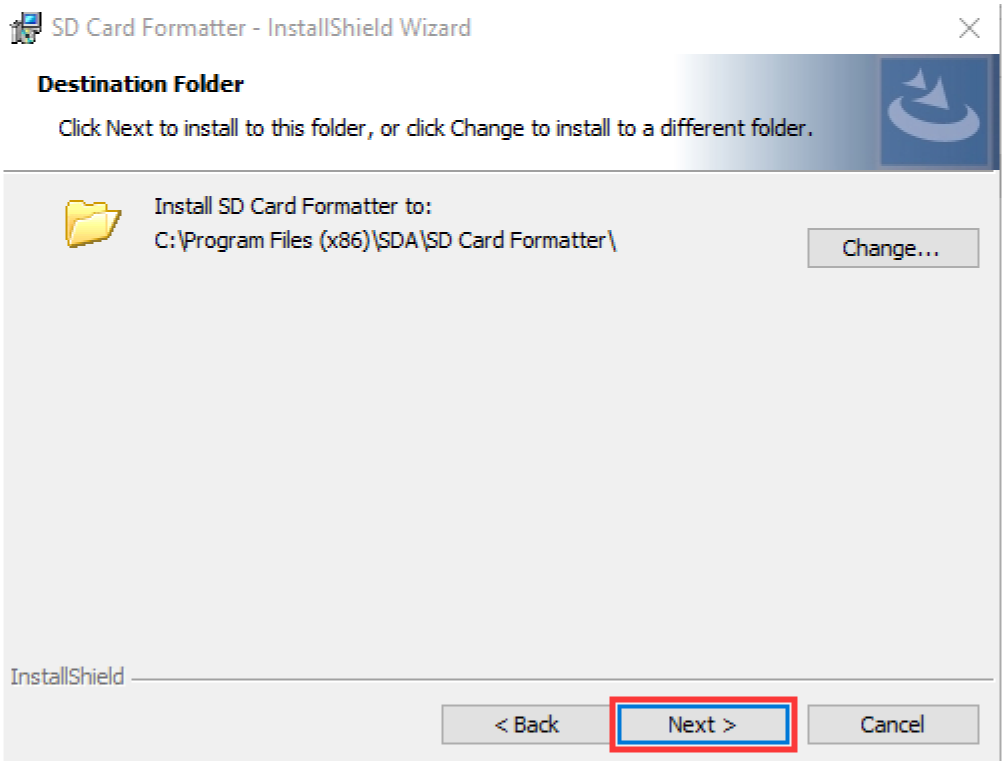
docs/4.Arduino_C_Tutorial(Raspberry-Pi)/media/046c67e4072093ee3dad27e8088fcf9f.png

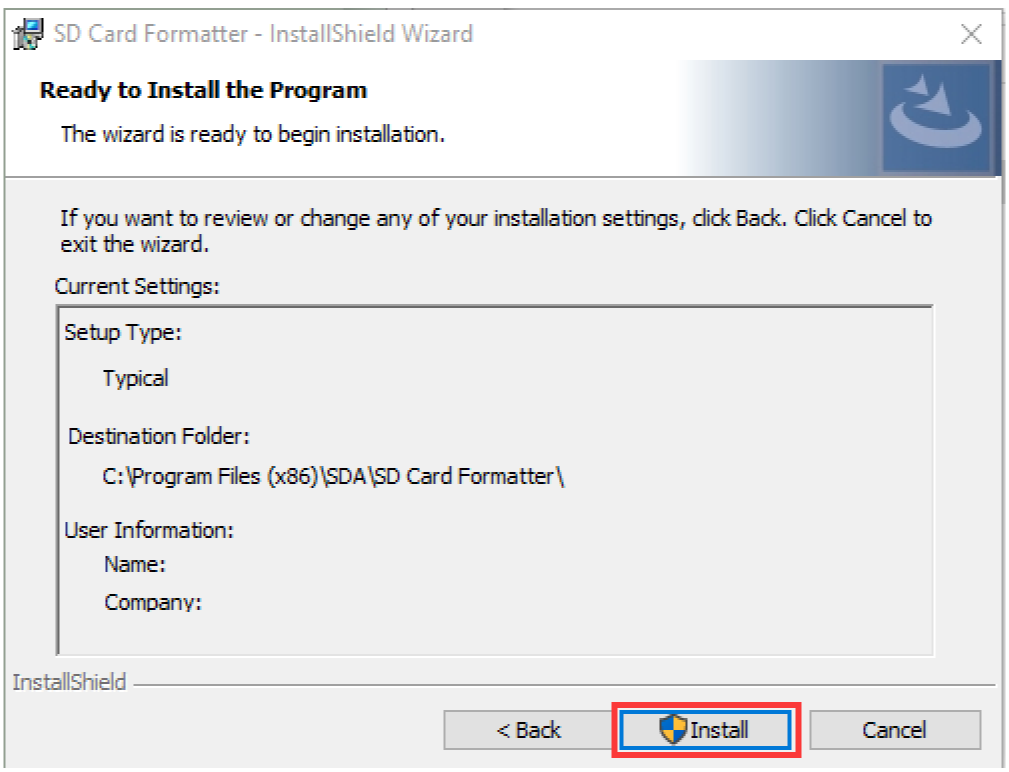
Click "Next" and choose docs/4.Arduino_C_Tutorial(Raspberry-Pi)/media/13dc08ae2b5cb52ae3d7ea198134d778.png, then tap "Next" .



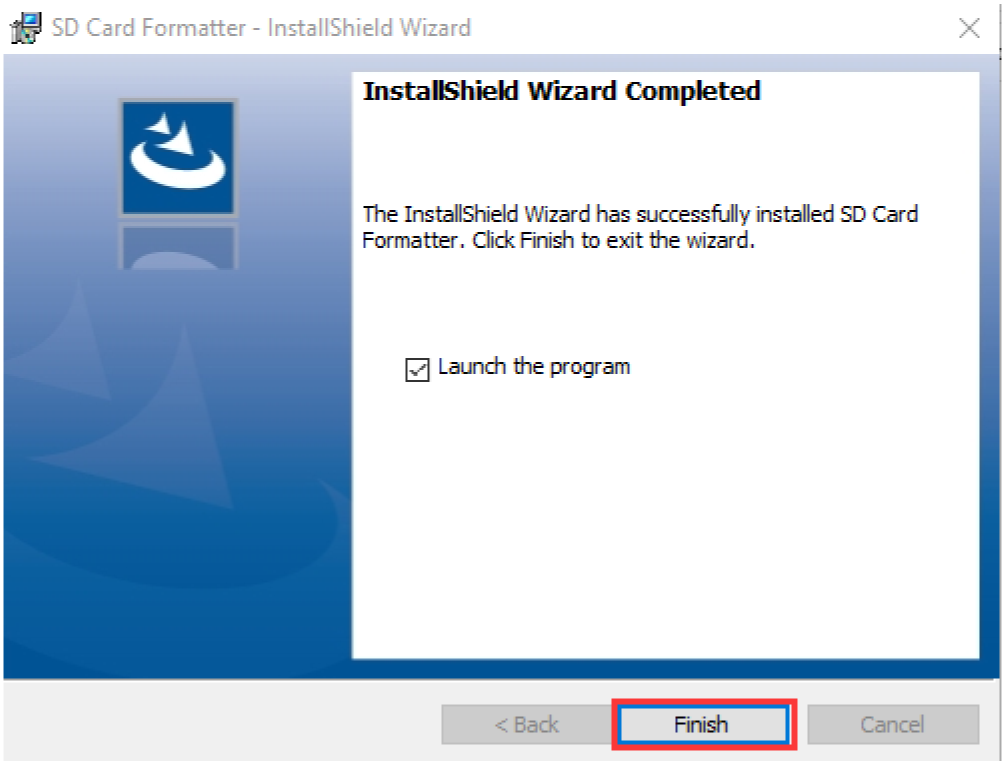


Click “Next” and “Install”.



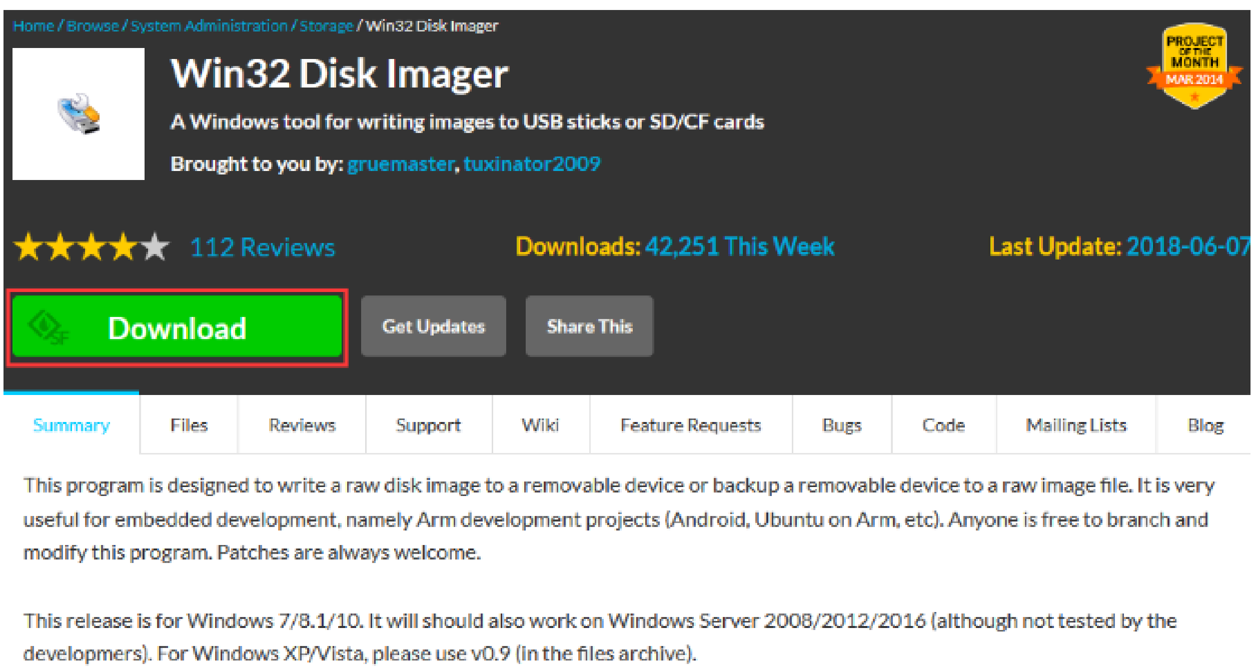


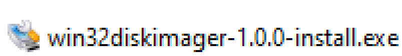
After a few seconds, click "Finish".



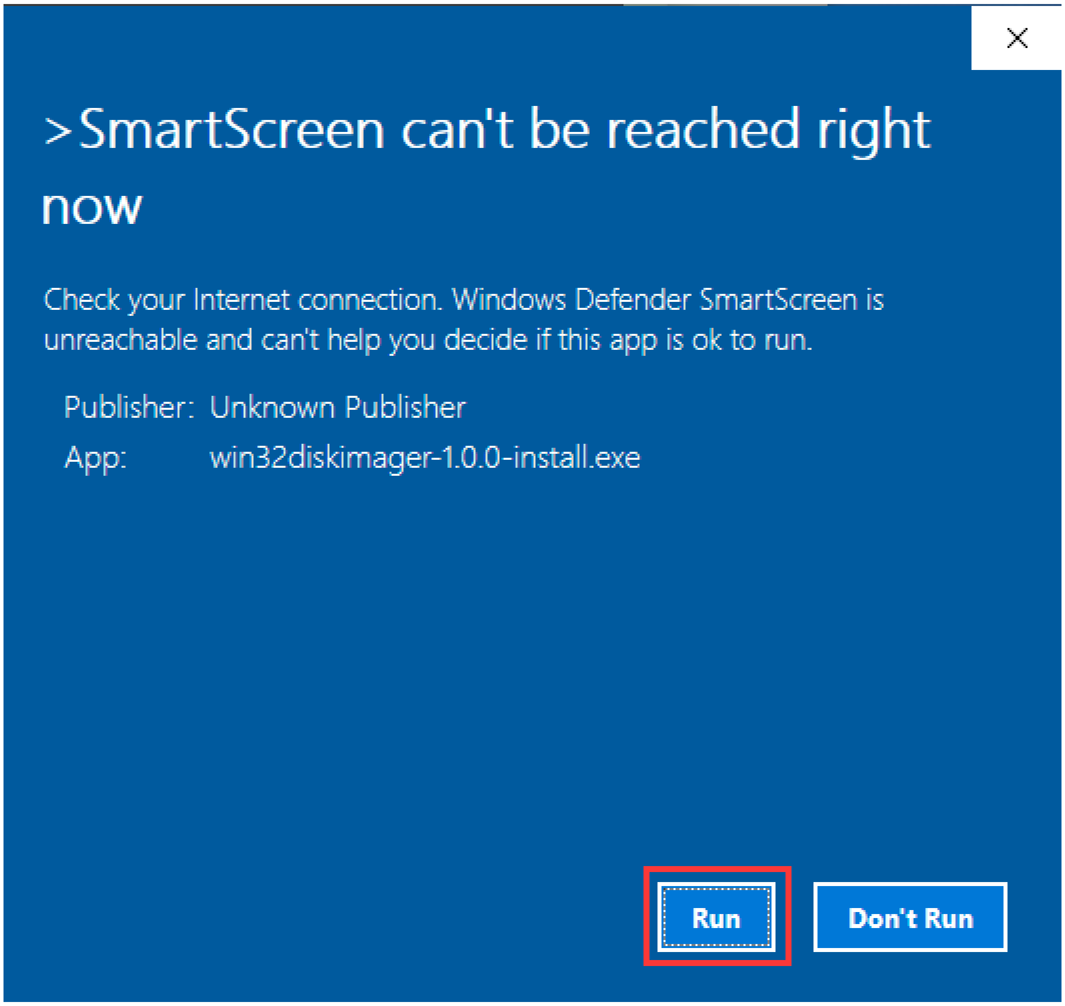
(4) Burn Win32DiskImager

Download Linkhttps://sourceforge.net/projects/win32diskimager/

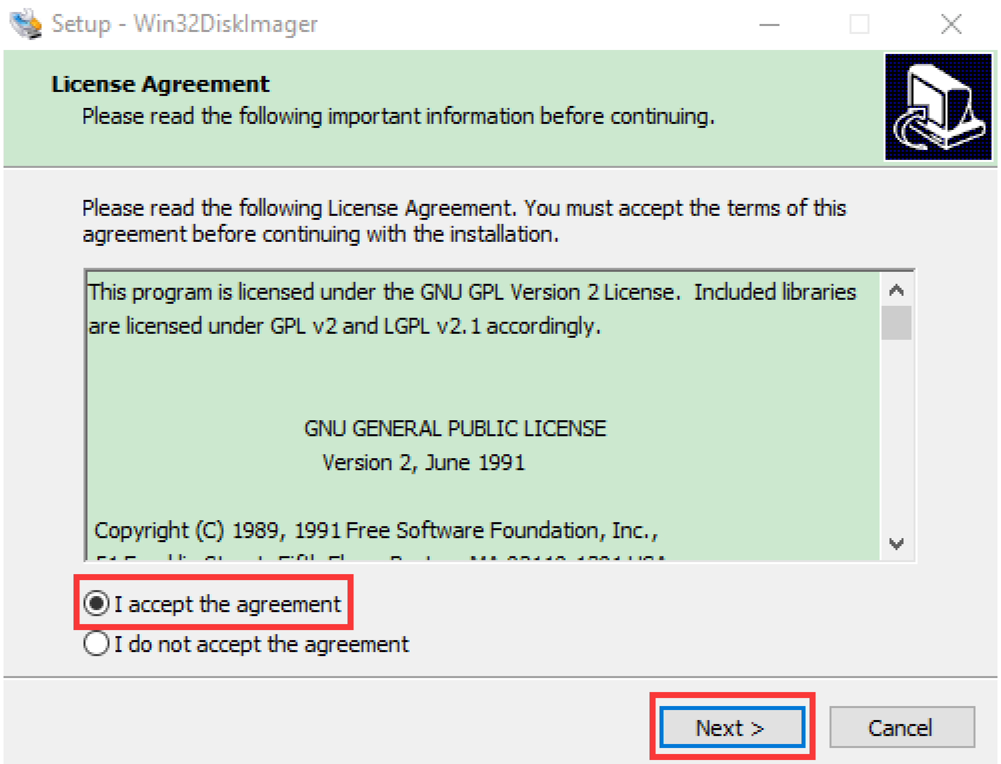


After the download, double-click 

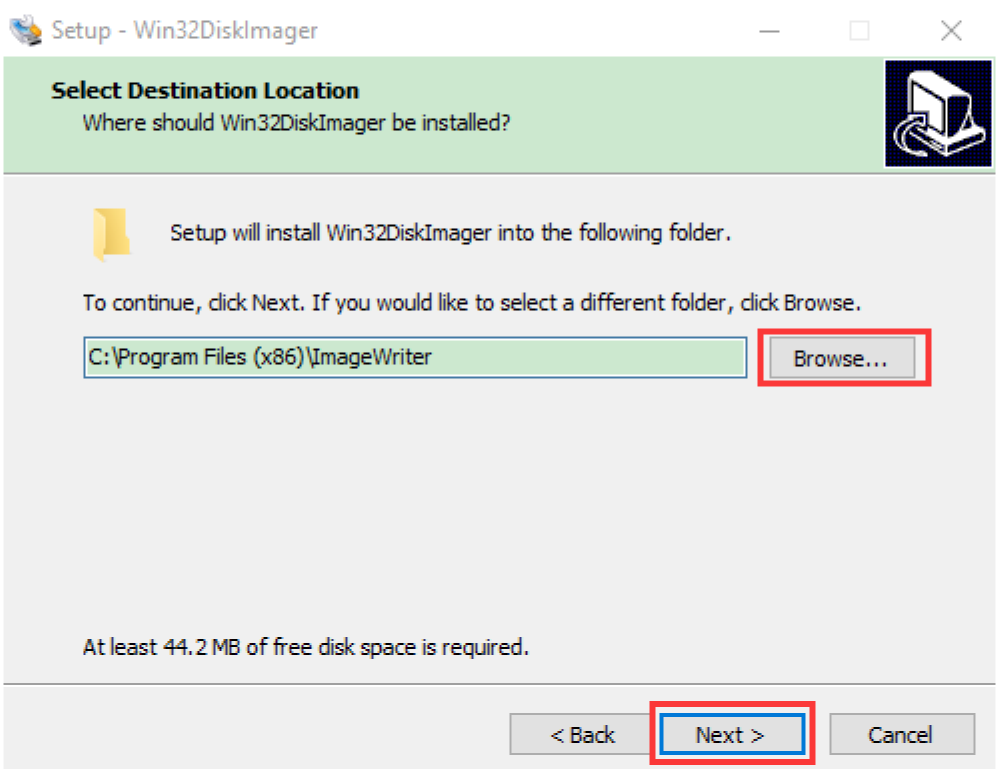
and tap "Run".

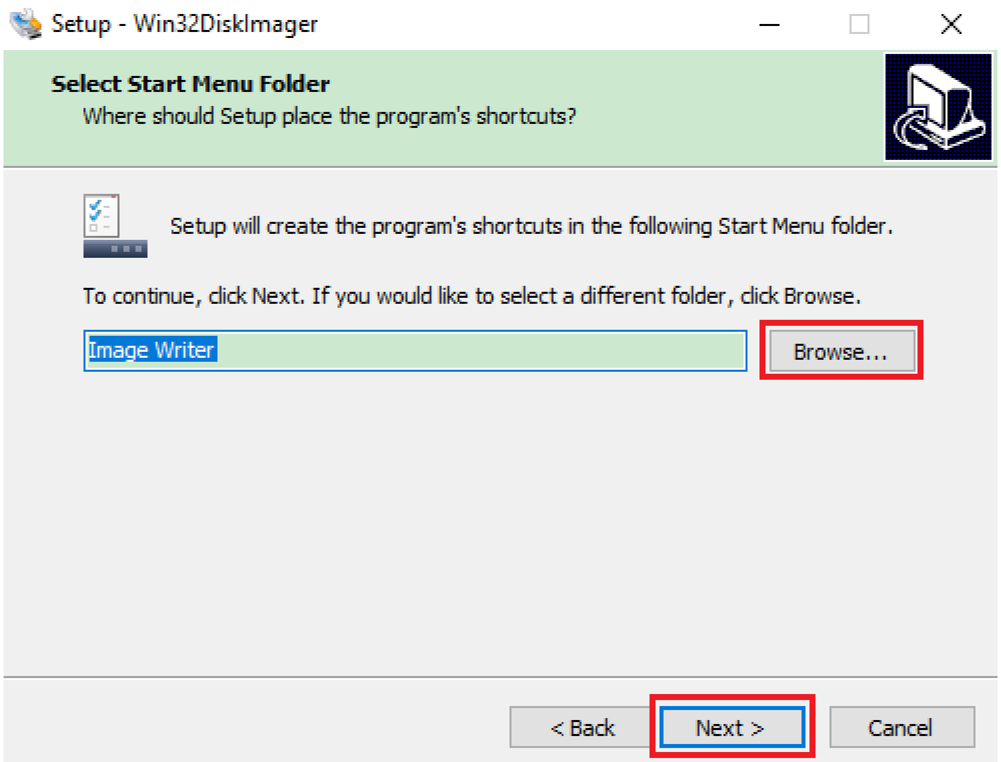


Select I accept the agreement and tap “Next”.

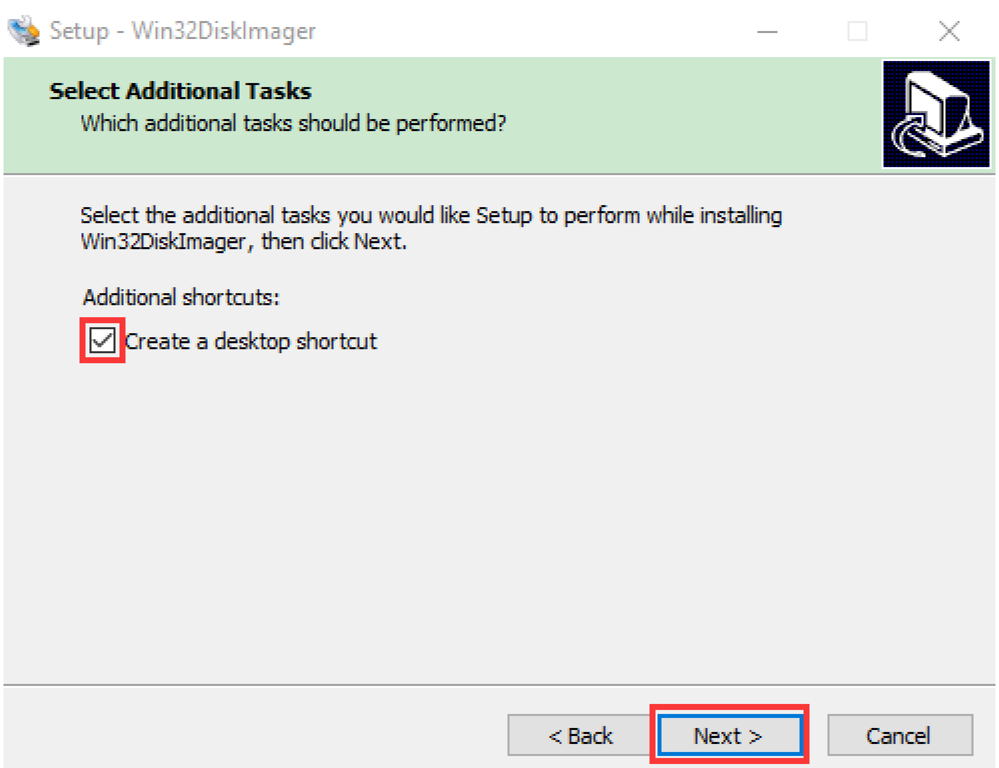


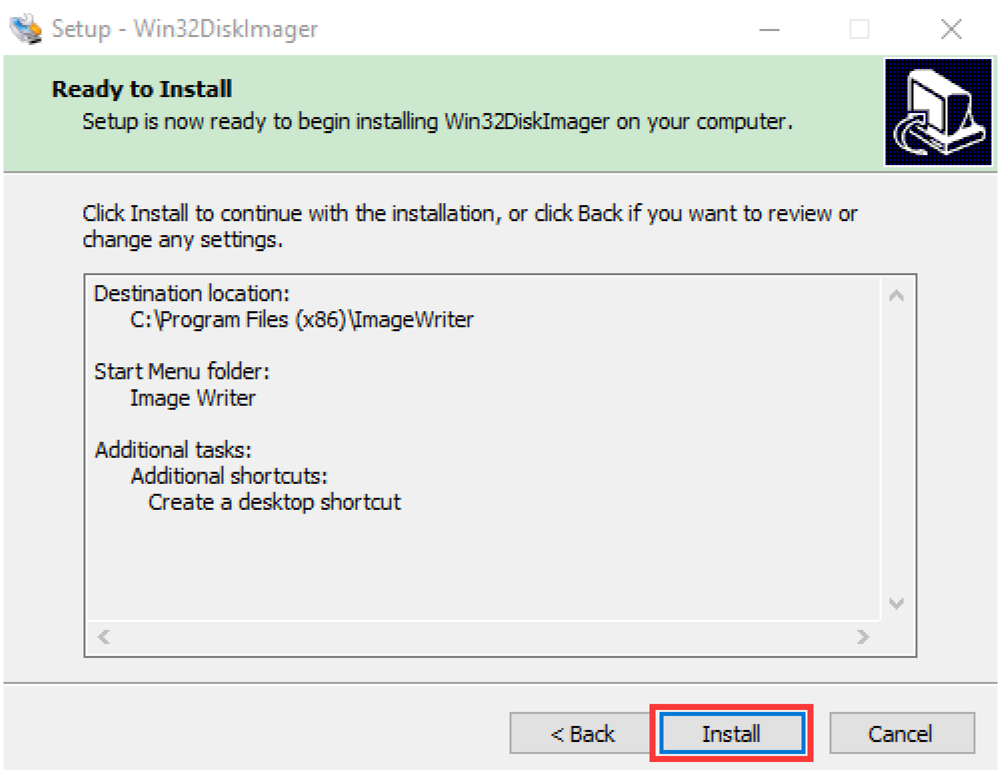
Click "Browse…" and find out the folder where the Win32DiskImager is located, tap "Next" .



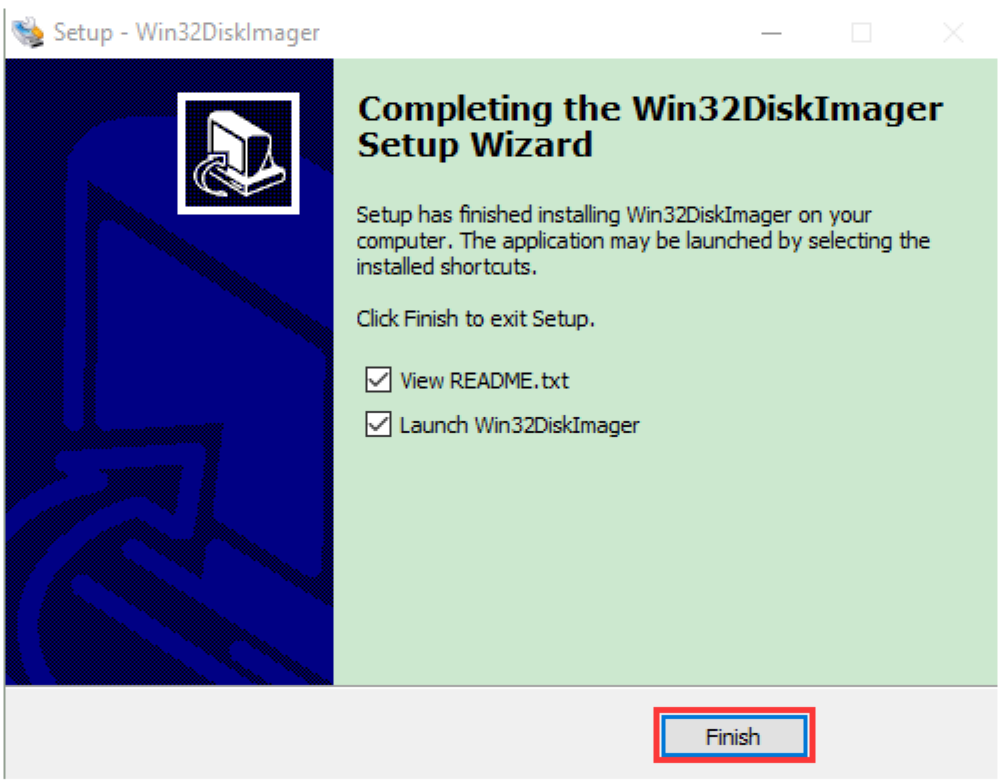


Tick Create a desktop shortcut , click "Next" and "Install".





After a few seconds, click "Finish".



The installation is finished.

(5) WNetWatcher

Scan to search ip address software tool—WNetWatcher

Download Linkhttp://www.nirsoft.net/utils/wnetwatcher.zip

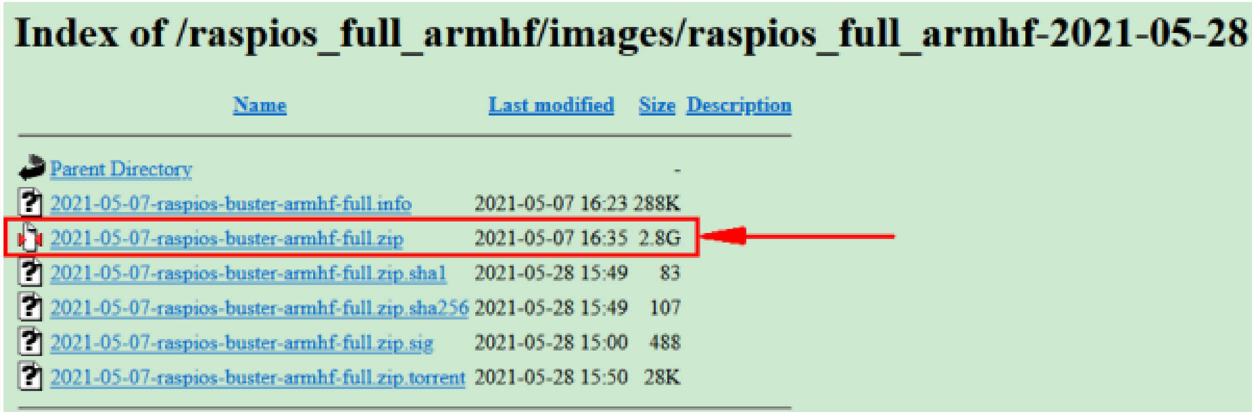
(6) Raspberry Pi Imager

Download Address

https://www.raspberrypi.org/downloads/raspberry-pi-os/

Old Version:

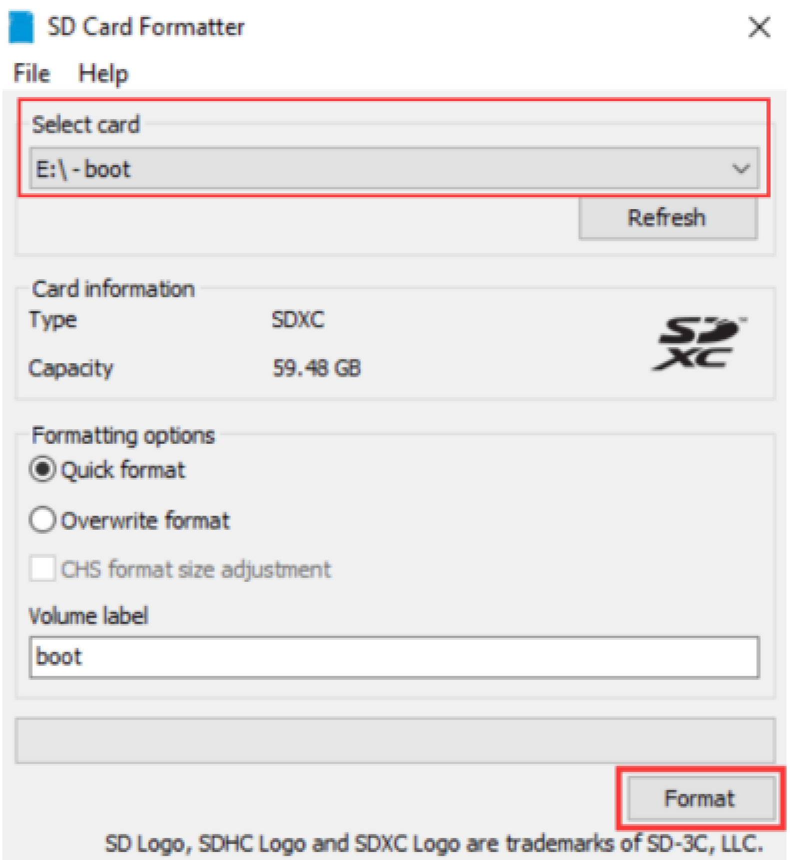
Raspbianhttps://downloads.raspberrypi.org/raspbian/images/ Raspbian fullhttps://downloads.raspberrypi.org/raspbian_full/images/ Raspbian litehttps://downloads.raspberrypi.org/raspbian_lite/images/ We use the 2020.05.28 version in the tutorial and recommend you to use this version. (Please download this version as shown in the picture below.) https://downloads.raspberrypi.org/raspios_full_armhf/images/raspios_full_armhf-2021-05-28/

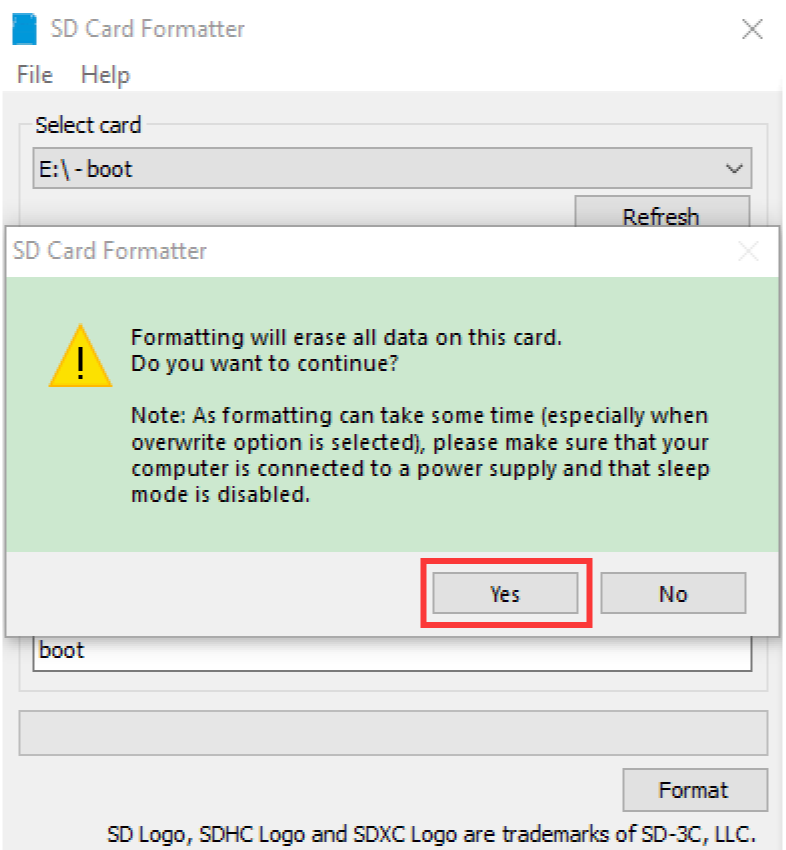


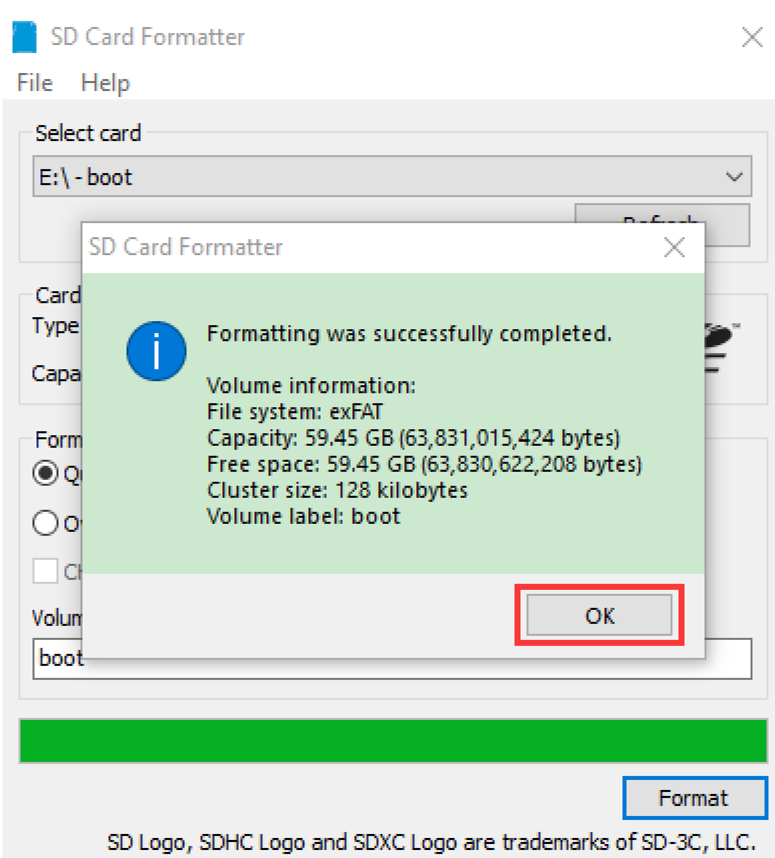
7.1.3 1.3. Install Raspberry Pi OS on Raspberry Pi 4B:

Insert TFT RAM card to card reader, then interface card reader to USB port of computer.

Format TFT RAM card with SD Card Formatter software, as shown below:

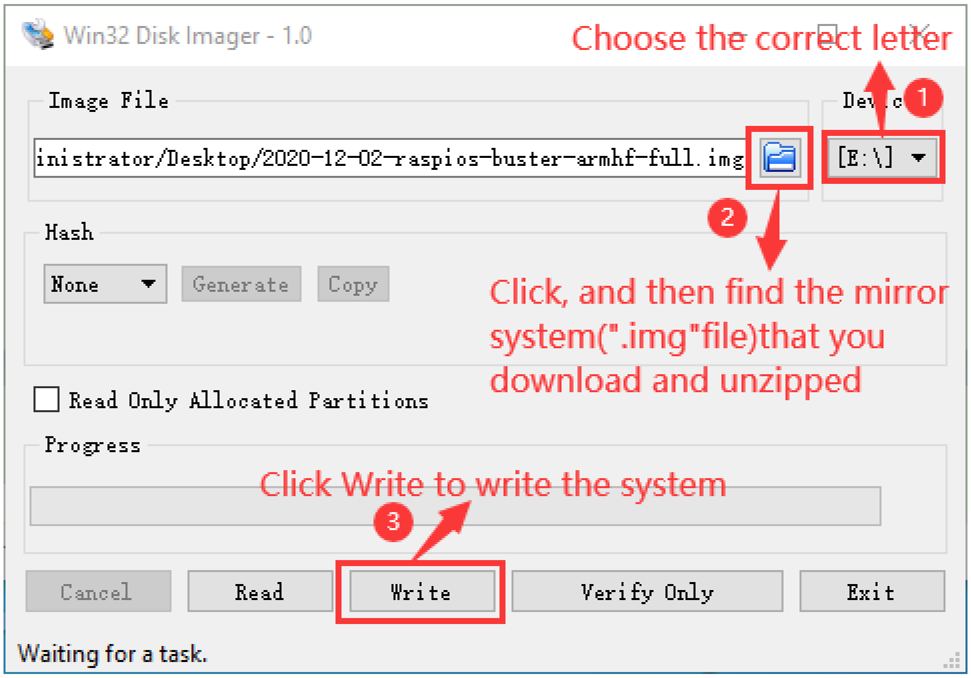

SD Card Formatter
File Help
Select card
E:\ - boot
Refresh
Card information
Type SDXC
Capacity 59.48 GB
Formatting options
Quick format
Overwrite format
CHS format size adjustment
Volume label
boot
Format
SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

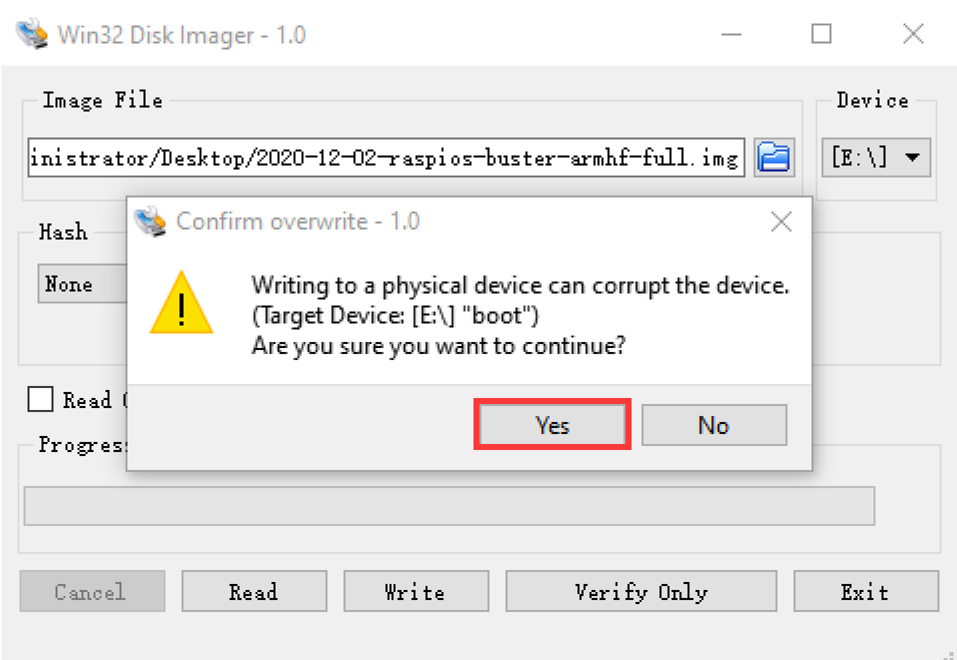

SD Card Formatter
File Help
Select card
E:\ - boot
Refresh
SD Card Formatter
Formatting will erase all data on this card.
Do you want to continue?
Note: As formatting can take some time (especially when overwrite option is selected), please make sure that your computer is connected to a power supply and that sleep mode is disabled.
Yes
No
boot
Format
SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

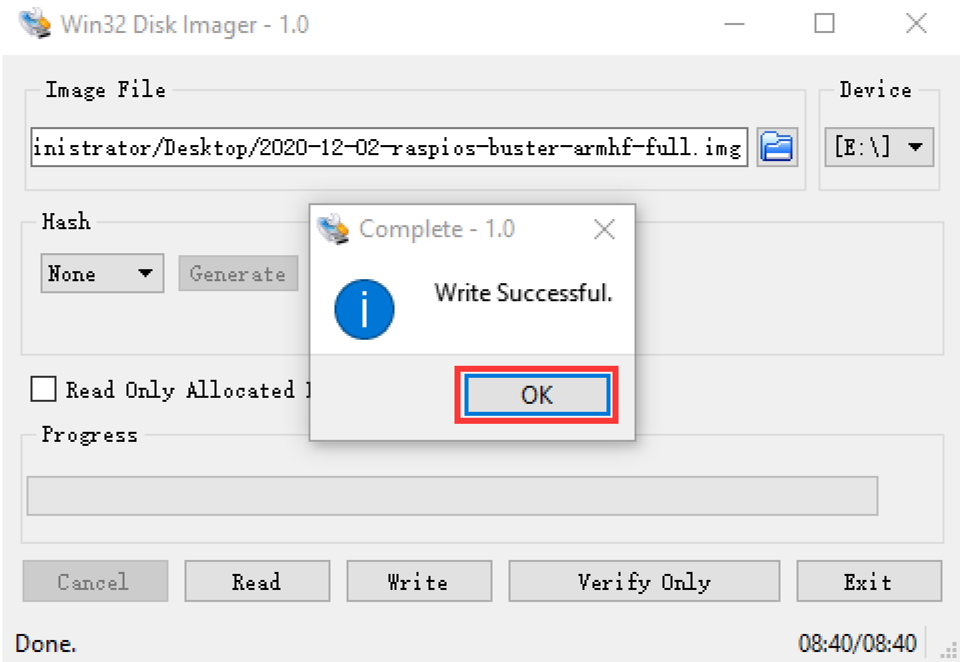

SD Card Formatter
File Help
Select card
E:\ - boot
SD Card Formatter
Formatting was successfully completed.
Volume information:
File system: exFAT
Capacity: 59.45 GB (63,831,015,424 bytes)
Free space: 59.45 GB (63,830,622,208 bytes)
Cluster size: 128 kilobytes
Volume label: boot
OK
boot
Format
SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

(1) Burn System

Burn the Raspberry Pi OS system to TFT card using Win32DiskImager software

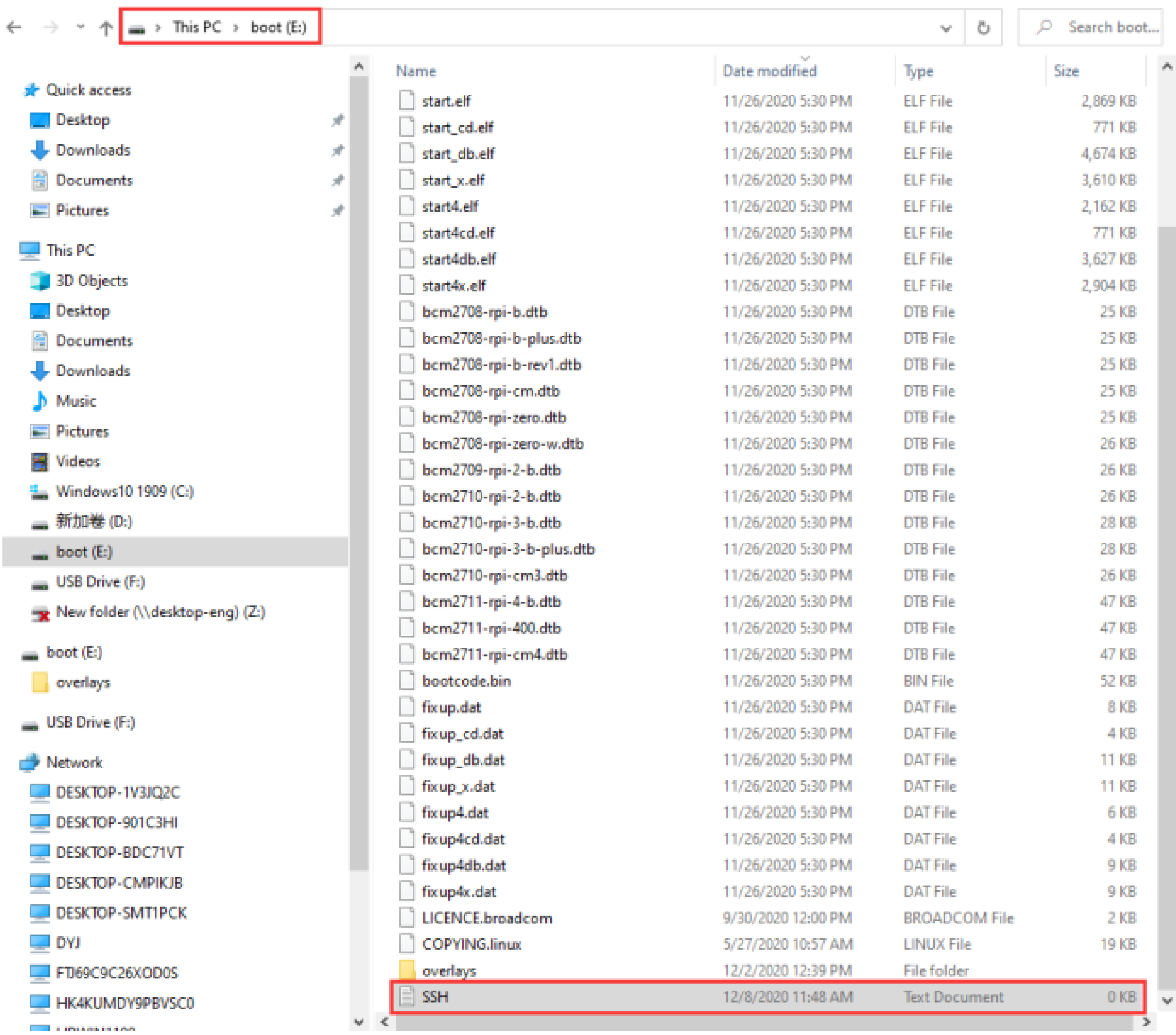






Don't eject card reader after burning mirror system, build a file named SSH, then delete .txt .

The SSH login function can be activated by copying SSH file to boot category, as shown below.

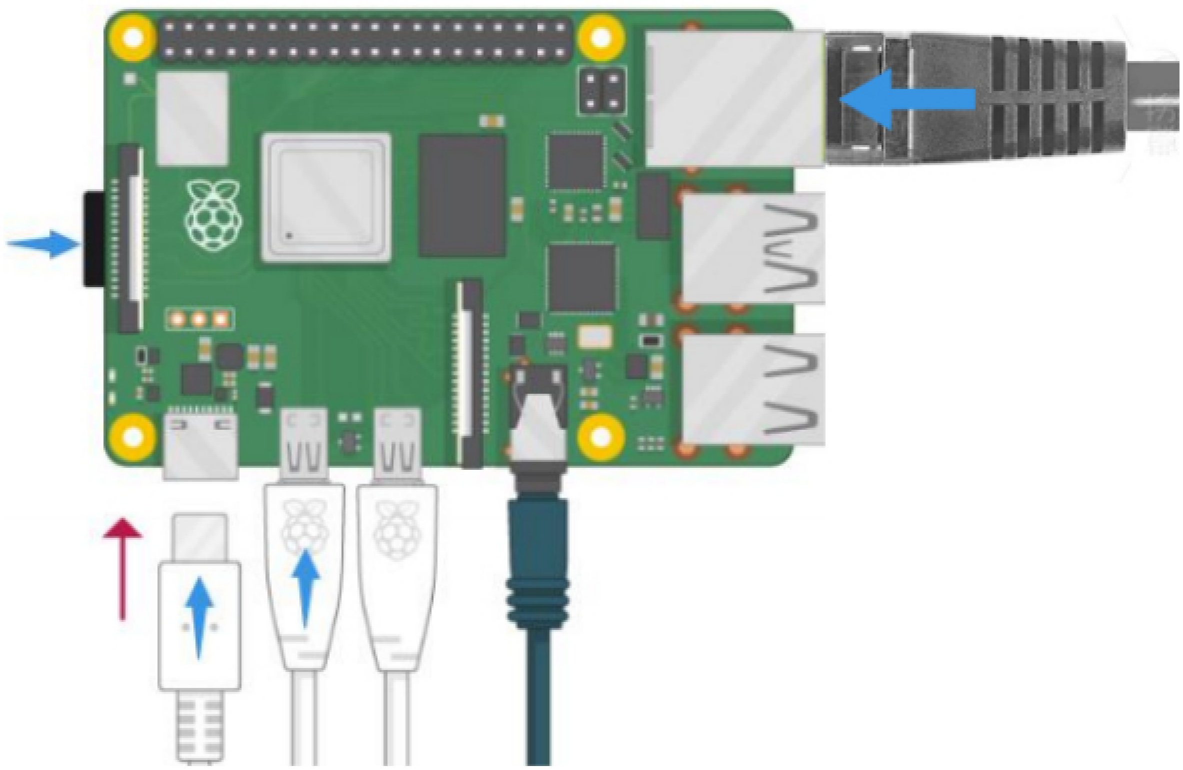


Eject Card Reader

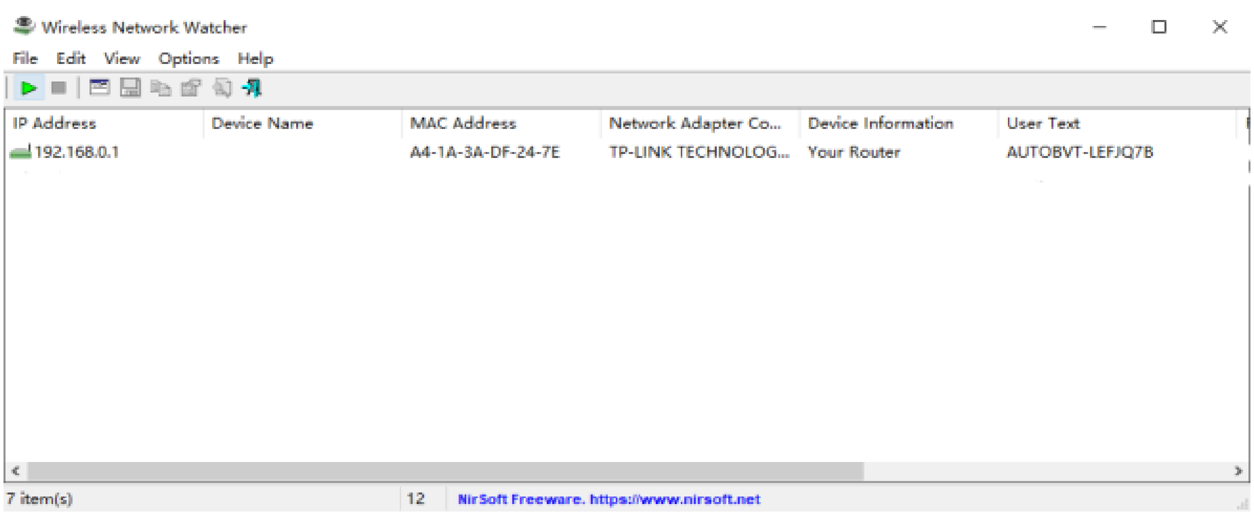
(2) Log in system

(Raspberry and PC should be in the same local area network.)

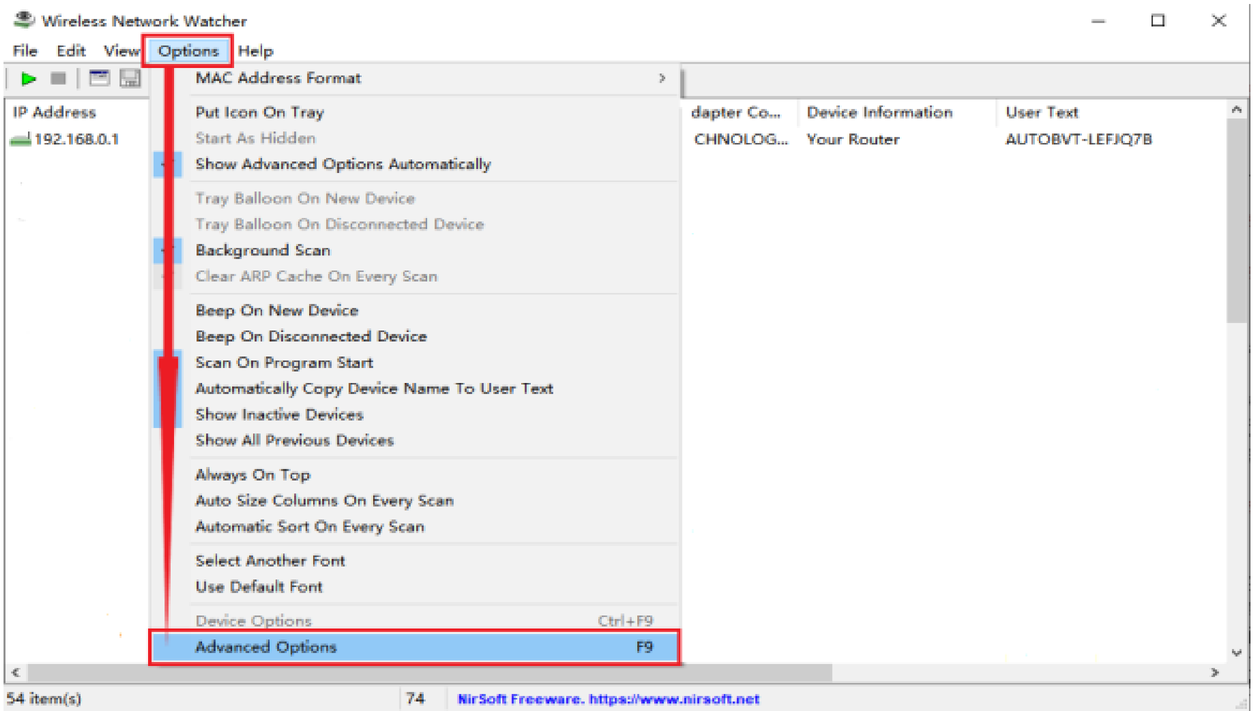
1). Insert TFT memory card into Raspberry Pi, connect internet cable and plug in power. If you have screen and HDMI cable of Raspberry Pi, you could view Raspberry Pi OS activating. If not, you can enter the desktop of Raspberry Pi via SSH remote login software—WinSCP and xrdp.

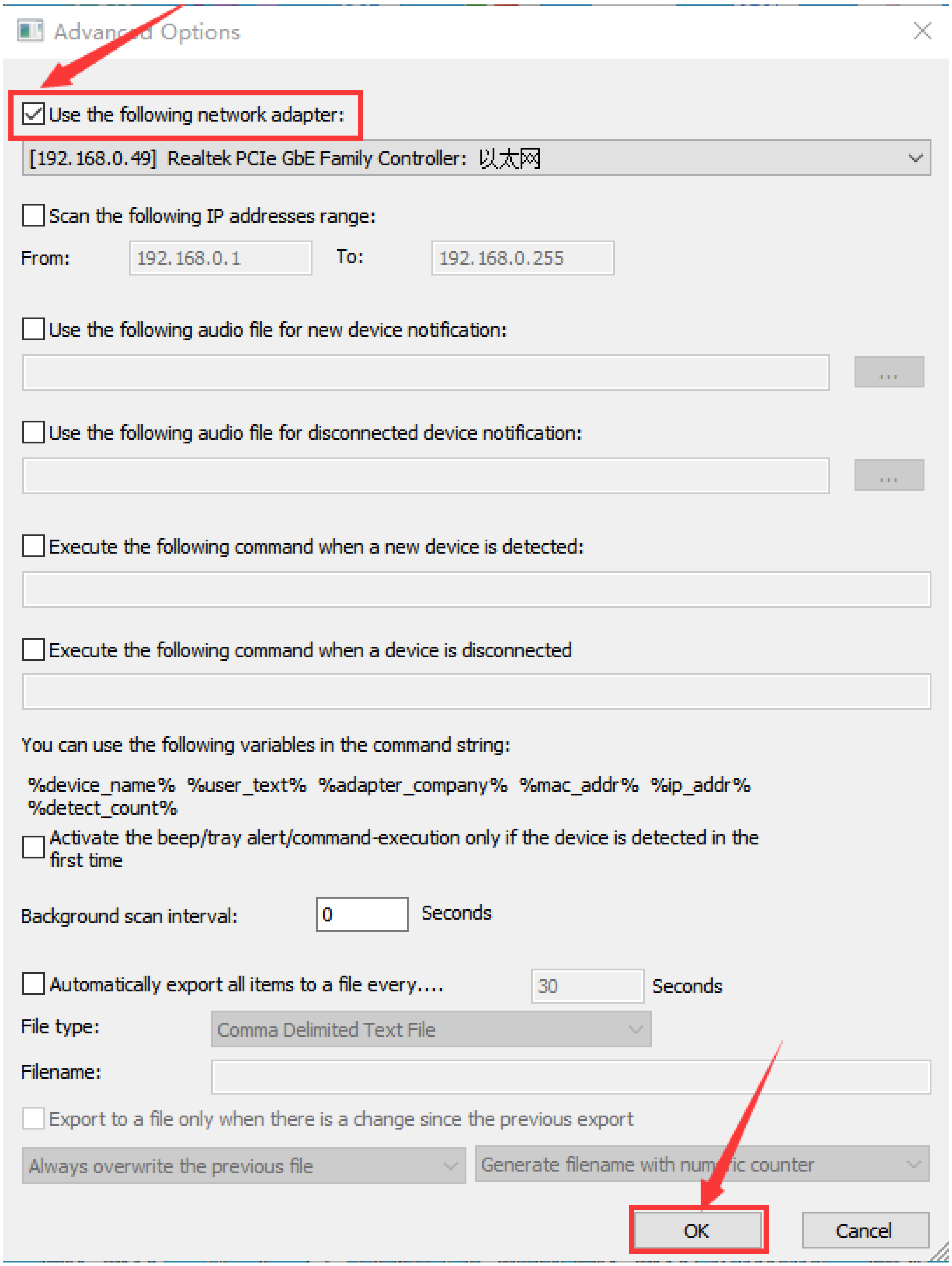


2). Use the WNetWatcher software to find the IP address of the Raspberry Pi.

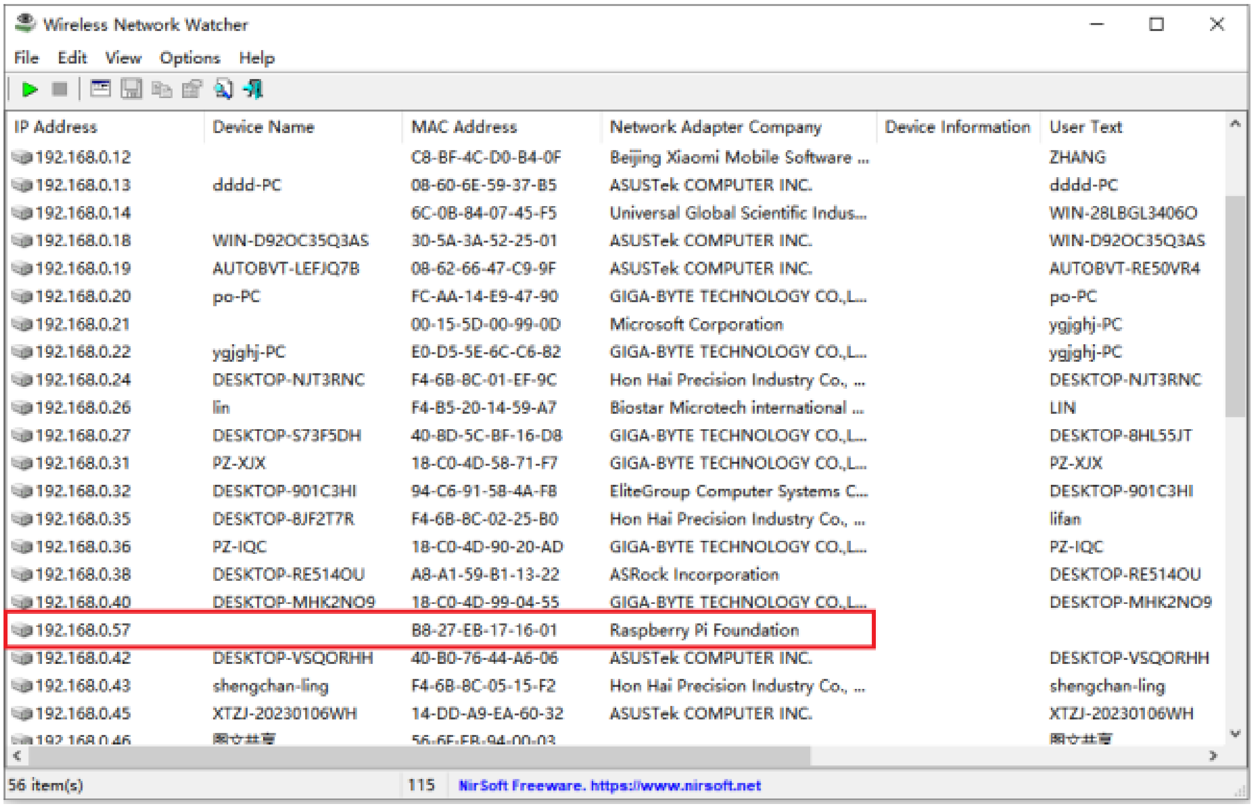


If there is no IP address as shown in the figure above, follow the following steps to set it.


Wireless Network Watcher
File
Edit
View
Options
Help
MAC Address Format
Put Icon On Tray
Start As Hidden
Show Advanced Options Automatically
Tray Balloon On New Device
Tray Balloon On Disconnected Device
Background Scan
Clear ARP Cache On Every Scan
Beep On New Device
Beep On Disconnected Device
Scan On Program Start
Automatically Copy Device Name To User Text
Show Inactive Devices
Show All Previous Devices
Always On Top
Auto Size Columns On Every Scan
Automatic Sort On Every Scan
Select Another Font
Use Default Font
Device Options Ctrl+F9
Advanced Options F9
IP Address
192.168.0.1
dapter Co...
CHNOLOG...
Device Information
Your Router
User Text
AUTOBVT-LEFJQ7B
54 item(s)
74
NirSoft Freeware. https://www.nirsoft.net



Once the setup is complete, record the IP and MAC addresses of the Raspberry PI. As shown in the red box below, the MAC address of the Raspberry PI is b8:27:eb:17:16:01, and the ip address is 192.168.0.57.

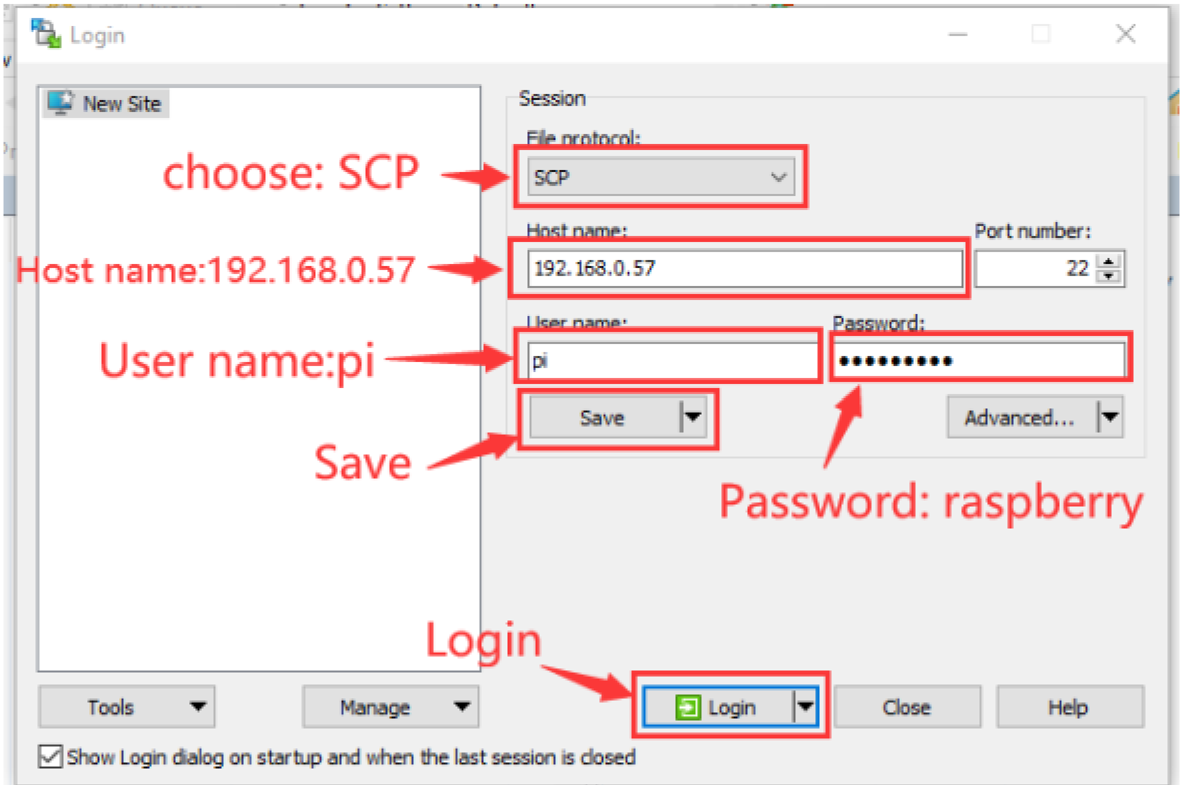


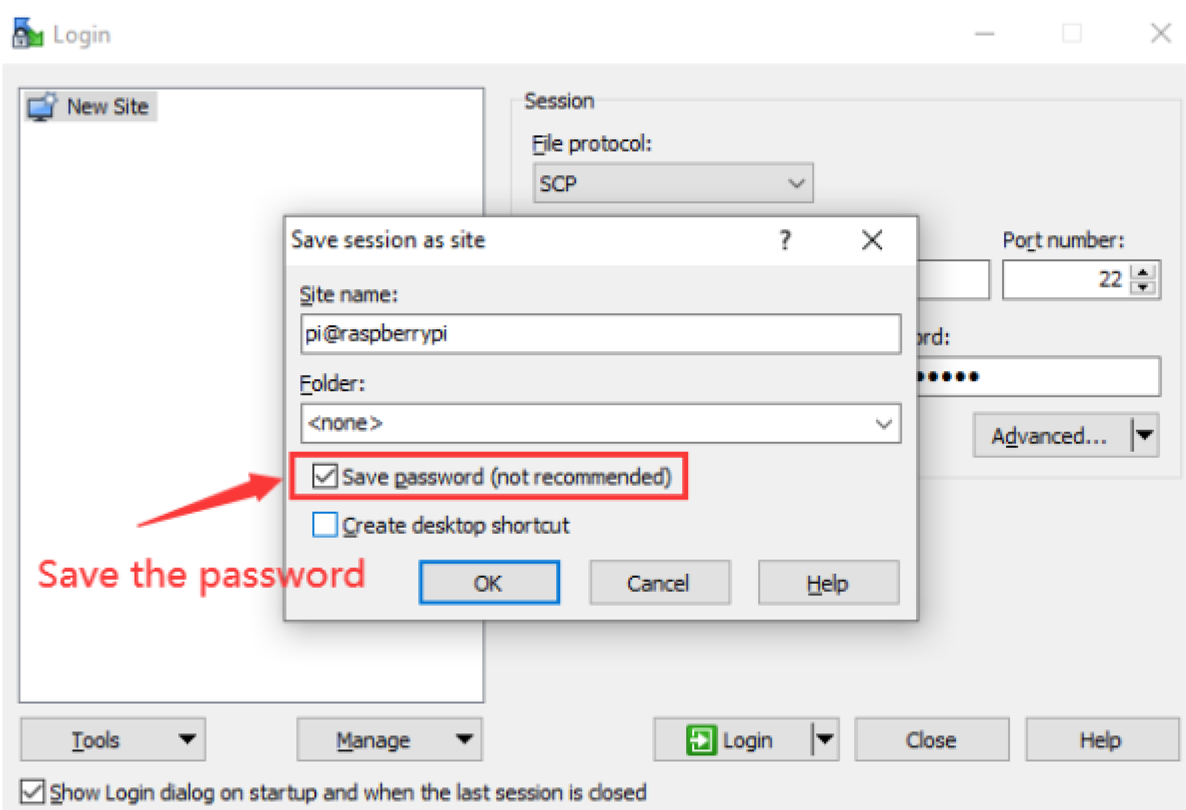
If you do not know the mac address and the ip address of the Raspberry PI, then unplug the network cable of the Raspberry PI first, open the **WNetWatcher** query, and the detection times will be displayed on the right side of the interface. Connect the Raspberry PI cable and query it once using **WNetWatcher**, and the Raspberry PI address is detected one less time than the other addresses. Then write down the ip and mac addresses.

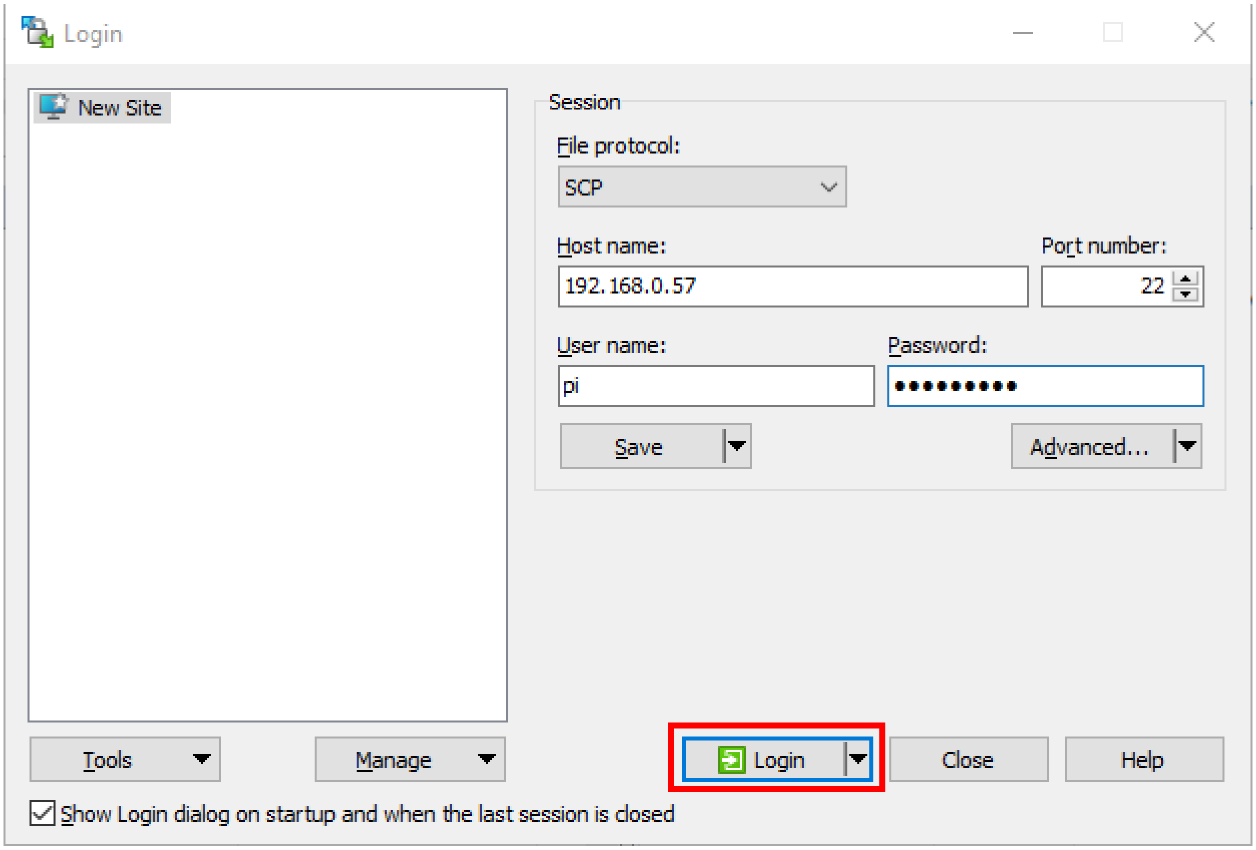
(3) Remote Login

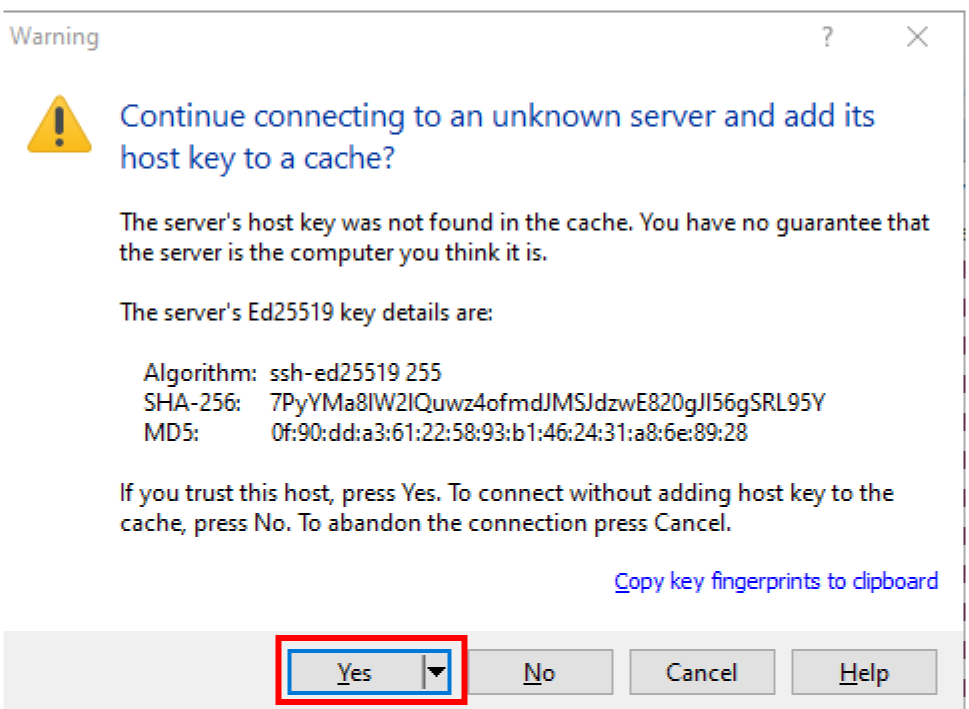
Enter default user name, password and host name on WinSCP to log in.

The same network only receives one Raspberry Pi.

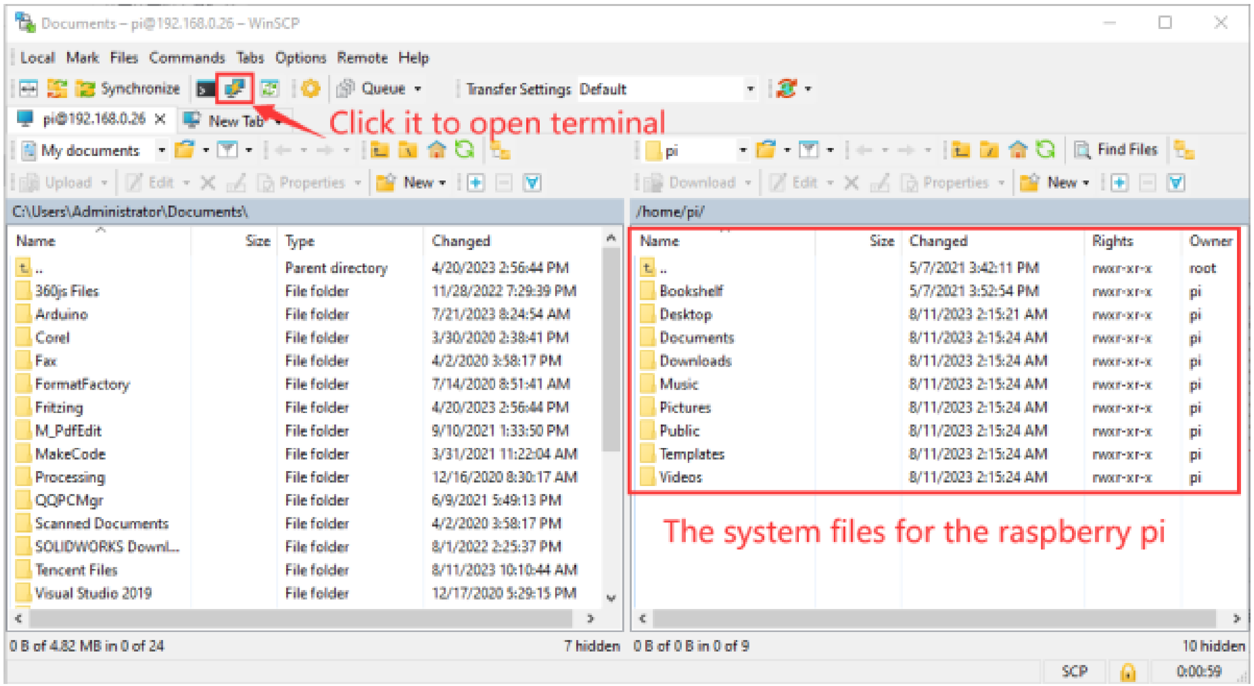

Login
New Site
Session
File protocol:
SCP
Host name:
192.168.0.57
Port number:
22
User name:
pi
Password:
Save
Advanced...
choose: SCP
Host name:192.168.0.57
User name:pi
Save
Password: raspberry
Login
Tools
Manage
Login
Close
Help
Show Login dialog on startup and when the last session is closed


Login
New Site
Session
File protocol:
SCP
Save session as site
Site name:
pi@raspberrypi
Folder:
<none>
Save password (not recommended)
Create desktop shortcut
OK
Cancel
Help
Port number:
22
Advanced...
Save the password
Tools
Manage
Login
Close
Help
Show Login dialog on startup and when the last session is closed

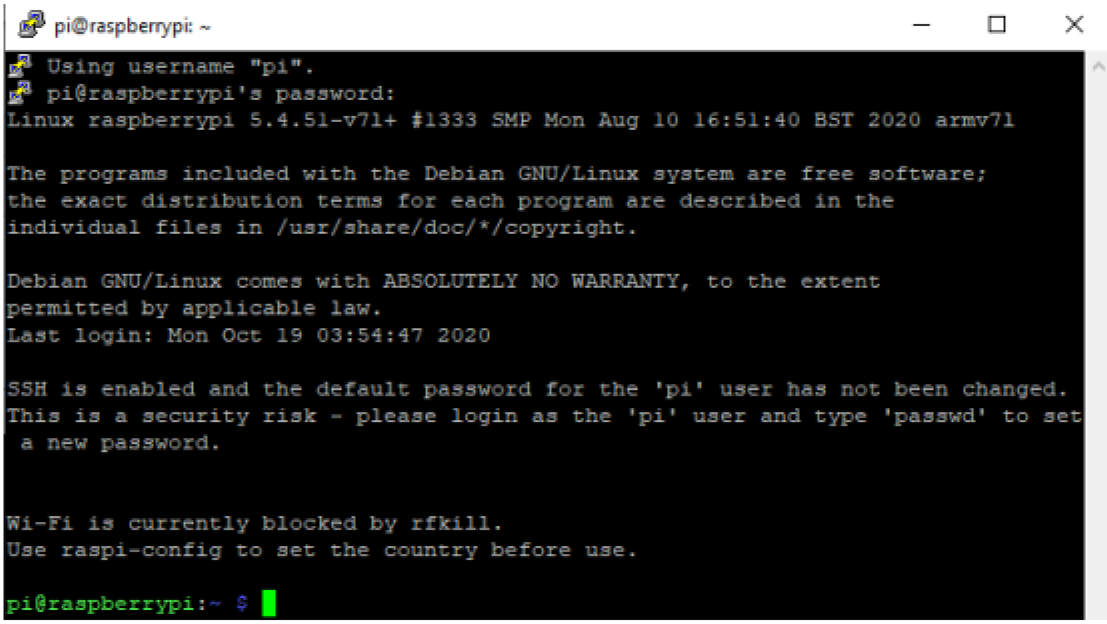

Login
New Site
Session
File protocol:
SCP
Host name:
192.168.0.57
Port number:
22
User name:
pi
Password:
Save
Advanced...
Tools
Manage
Login
Close
Help
Show Login dialog on startup and when the last session is closed


Warning
Continue connecting to an unknown server and add its host key to a cache?
The server's host key was not found in the cache. You have no guarantee that the server is the computer you think it is.
The server's Ed25519 key details are:
Algorithm: ssh-ed25519 255
SHA-256: 7PyYMa8lW2lQuwz4ofmdJMSJdzwE820gJI56gSRL95Y
MD5: 0f:90:dd:a3:61:22:58:93:b1:46:24:31:a8:6e:89:28
If you trust this host, press Yes. To connect without adding host key to the cache, press No. To abandon the connection press Cancel.
Copy key fingerprints to clipboard
Yes
No
Cancel
Help

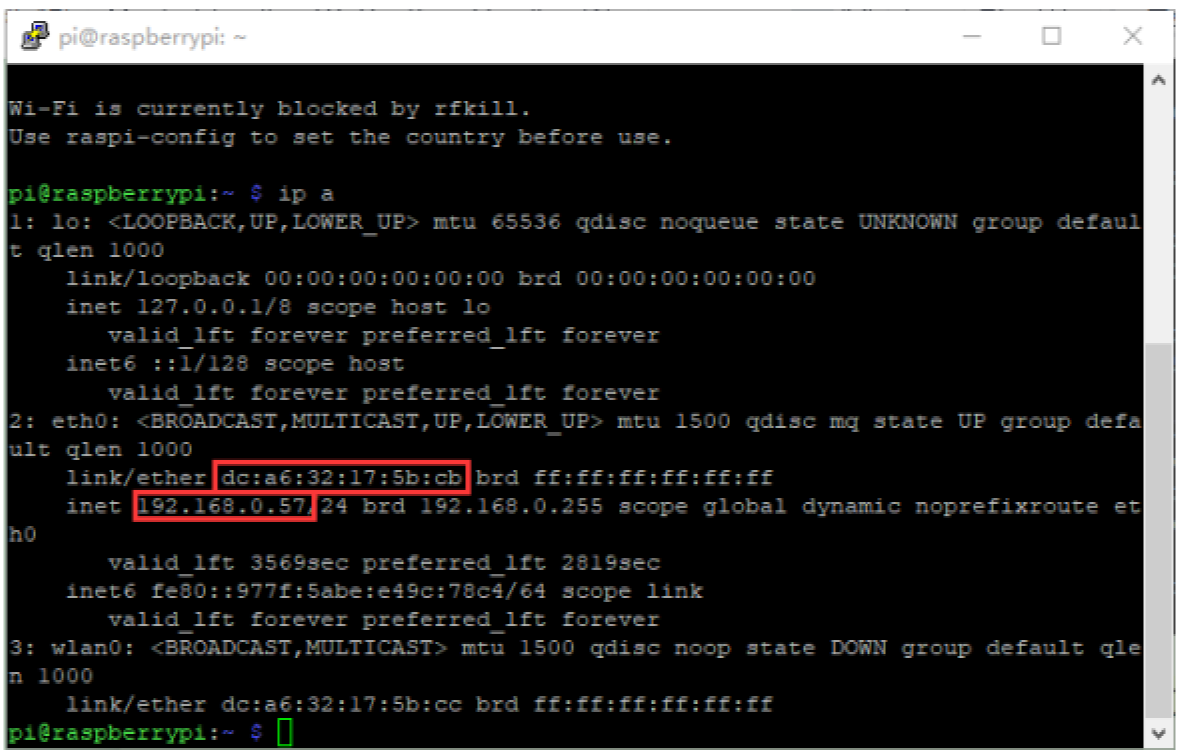
(4) Check ip and mac address



Click to open terminal input the passwordraspberry, and press“**Enter**”on keyboard.



Logging in successfully, open the terminal, input ip a and tap"**Enter**"to check ip and mac address.



(5) Fix ip address of Raspberry Pi

Ip address is changeable, therefore, we need to make ip address fixed for convenient use.

Follow the below steps

Switch to root user

If without root user's password

 Set root passward

Input passwordin the terminalsudo passwd root to set password

 Switch to root user

Input su root

 Fix the configuration file of ip address

Firstly change ip address of the following configuration file.

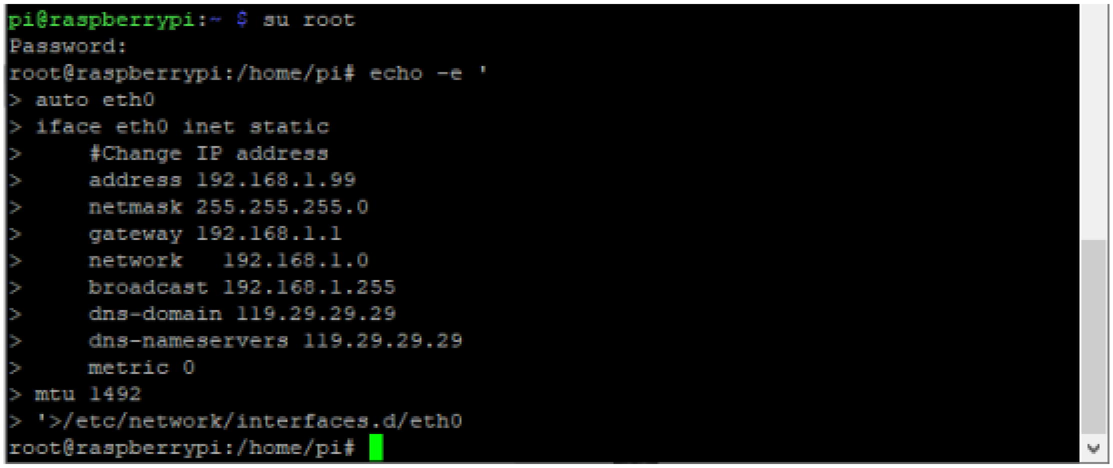
#New ip addressaddress 192.168.0.57

Copy the above new address to terminal and press“**Enter**”.

Configuration File

```
echo -e '

auto eth0

iface eth0 inet static

\#Change IP address

address 192.168.0.57

netmask 255.255.255.0

gateway 192.168.1.1

network 192.168.1.0

broadcast 192.168.1.255

dns-domain 119.29.29.29

dns-nameservers 119.29.29.29

metric 0

mtu 1492

'\>/etc/network/interfaces.d/eth0
```

As shown below:

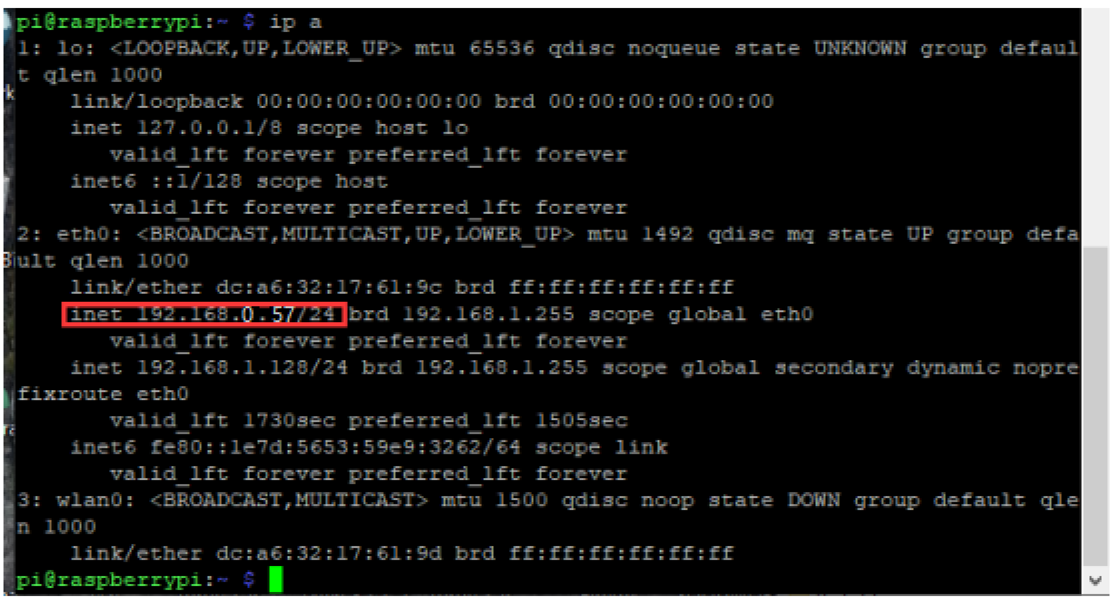


Reboot the system and activate the configuration file

Input the restart command in the terminal: sudo reboot

You could log in via fixed ip afterwards.

Check IP and insure ip address fixed well



(6) Log in Desktop on Raspberry Pi Wirelessly

In fact, we can log in desktop on Raspberry Pi Wirelessly even without screen and HDMI cable.

VNC and Xrdp are commonly used to log in desktop of Raspberry Pi wirelessly.

Install Xrdp Service in the terminal

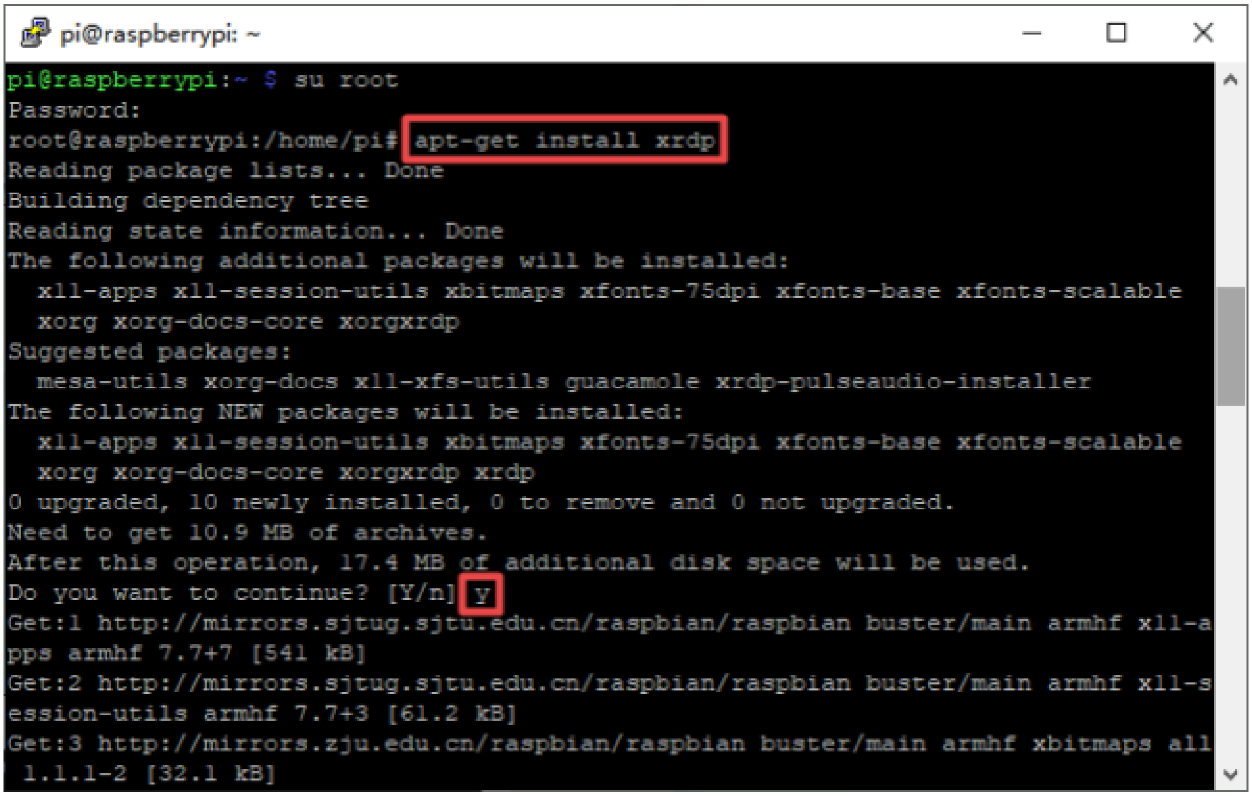
Installation commands:

Switch to Root User: su root

Install apt-get install xrdp

Enter y and press "**Enter**"

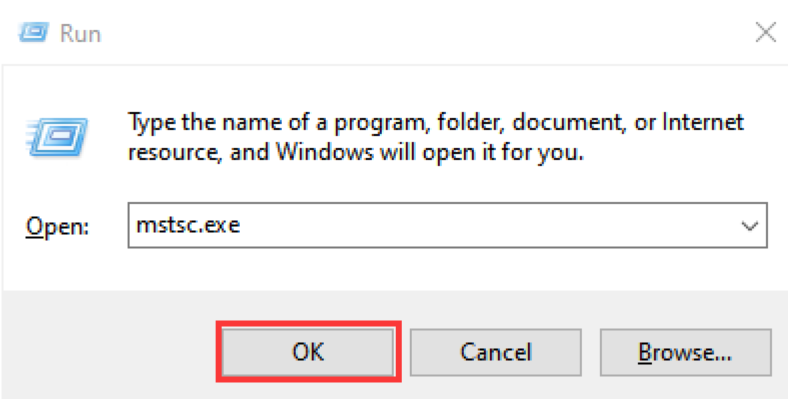
As shown below:



(7) Open the remote desktop connection on Windows

Press **WIN+R** on keyboard and enter mstsc.exe

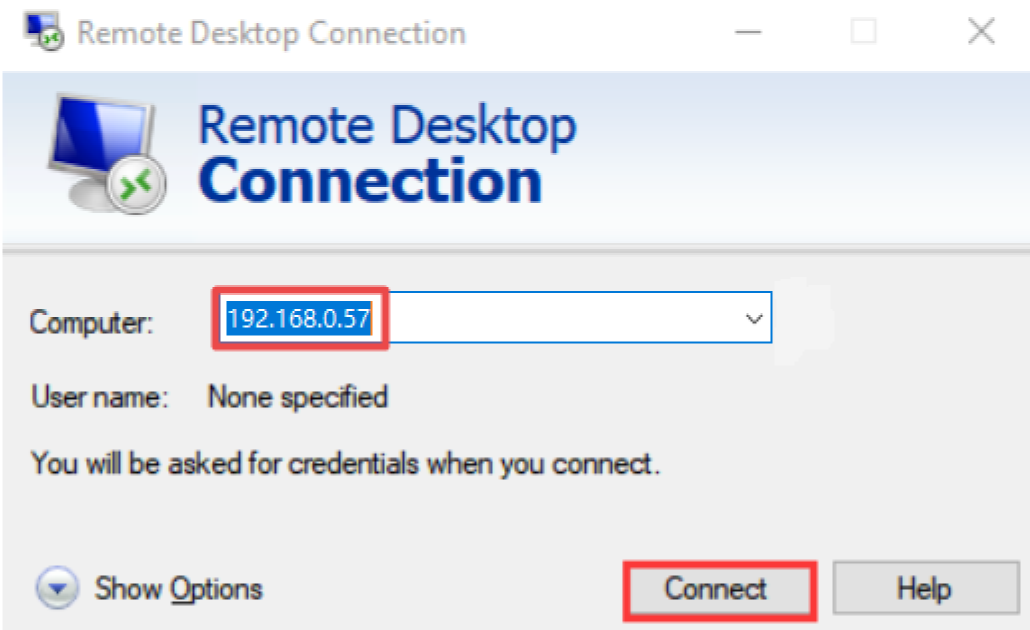
As shown below



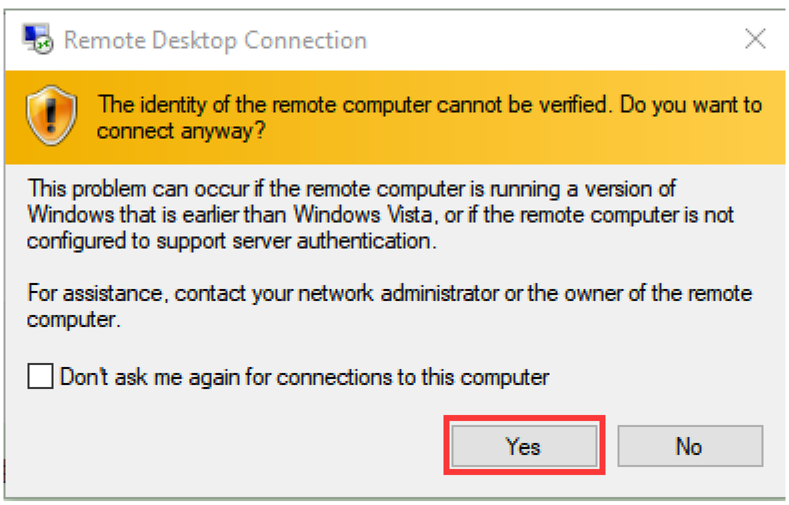
Input ip address of Raspberry Pi, as shown below.

Click“**Connect**”and tap“**Connect**”.

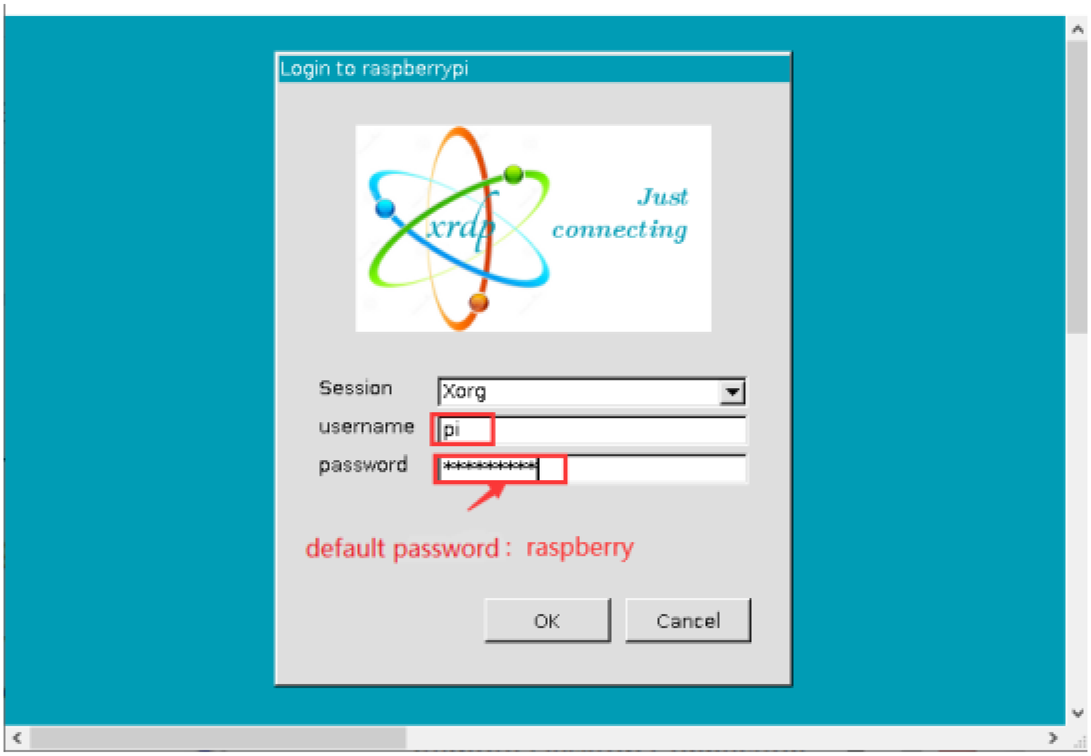
192.168.0.57 is ip address we use, you could change into yours ip address.



Click “**Yes**”.



Input user name: pi, default password: raspberry, as shown below:



Click "OK" or "Enter" , you will view the desktop of Raspberry Pi OS, as shown below:

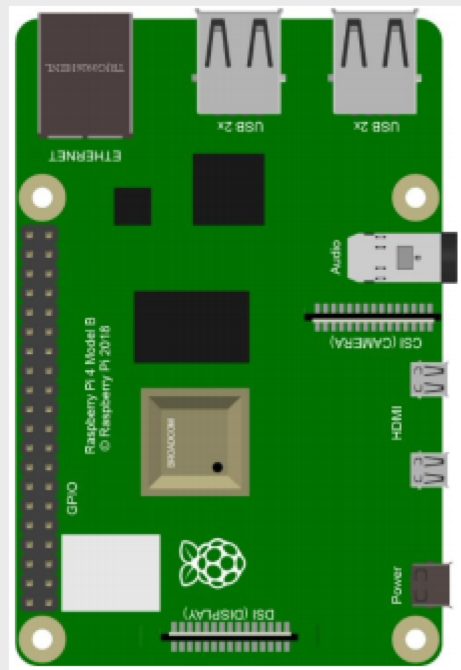
Now, we finish the basic configuration of Raspberry Pi OS.

7.2 2. Preparations for C language:

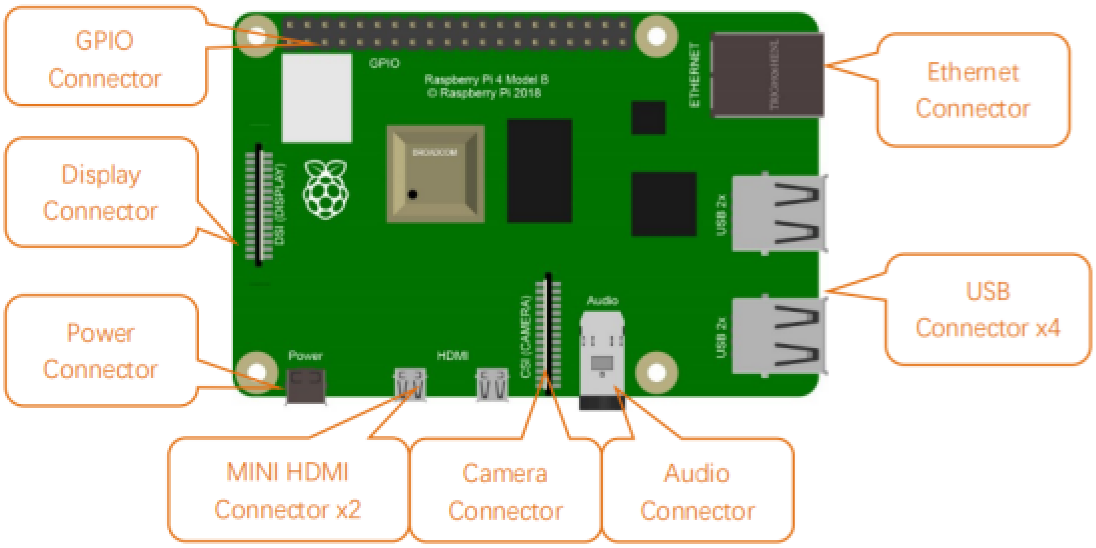
C language is a programming language with a considerably fast running speed. There are numerous software and system core code written in it, such as Linux system. Notably, hardware MCU and embedded class are not exception. Thereby, it makes sense to learn the C language to control hardware.

7.2.1 2.1. Hardware

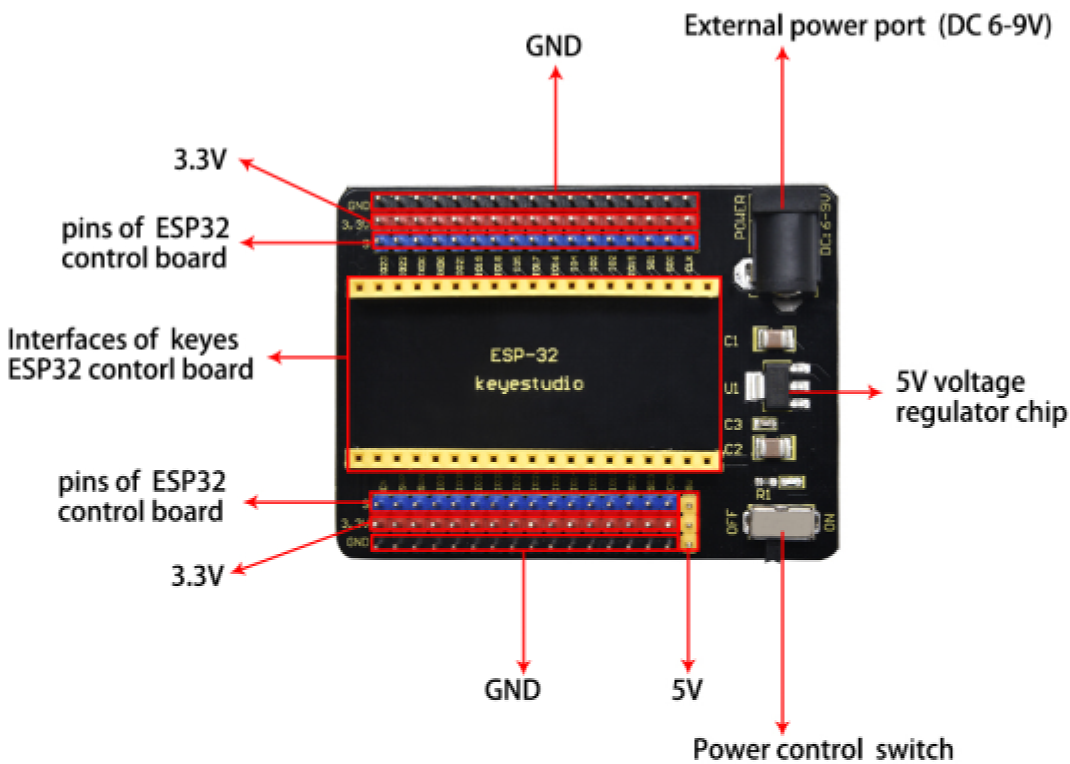
(1) Raspberry Pi 4B:

Raspberry Pi 4B	Raspberry Pi 4B Model
	

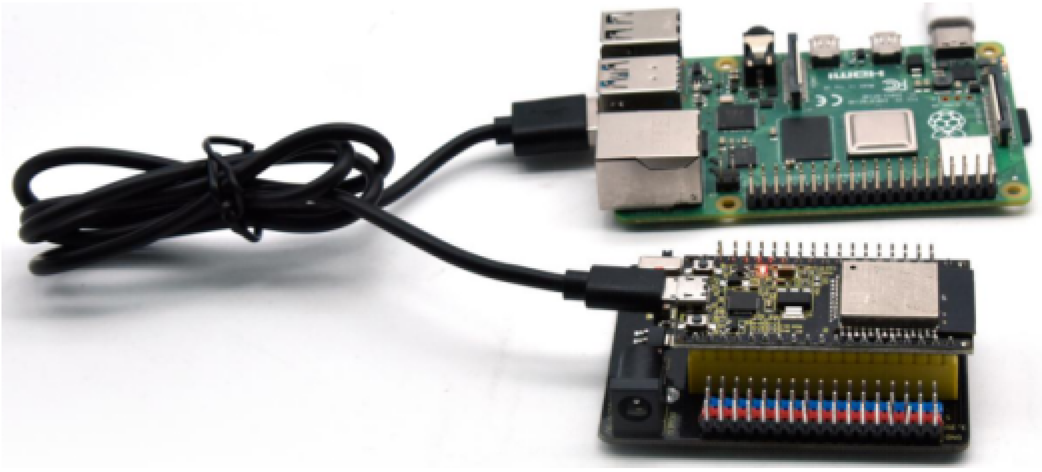
Hardware Interfaces

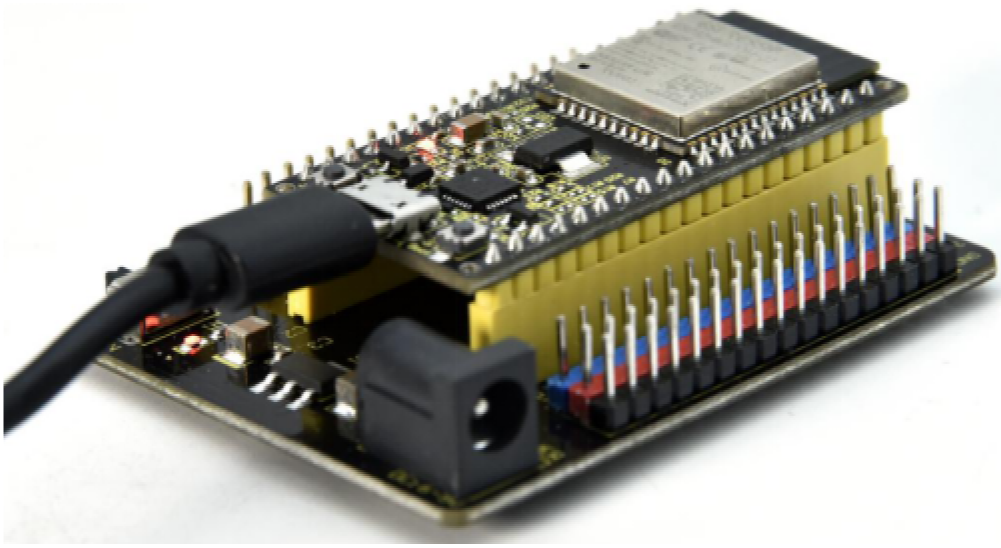


(2) ESP32 Expansion Board:



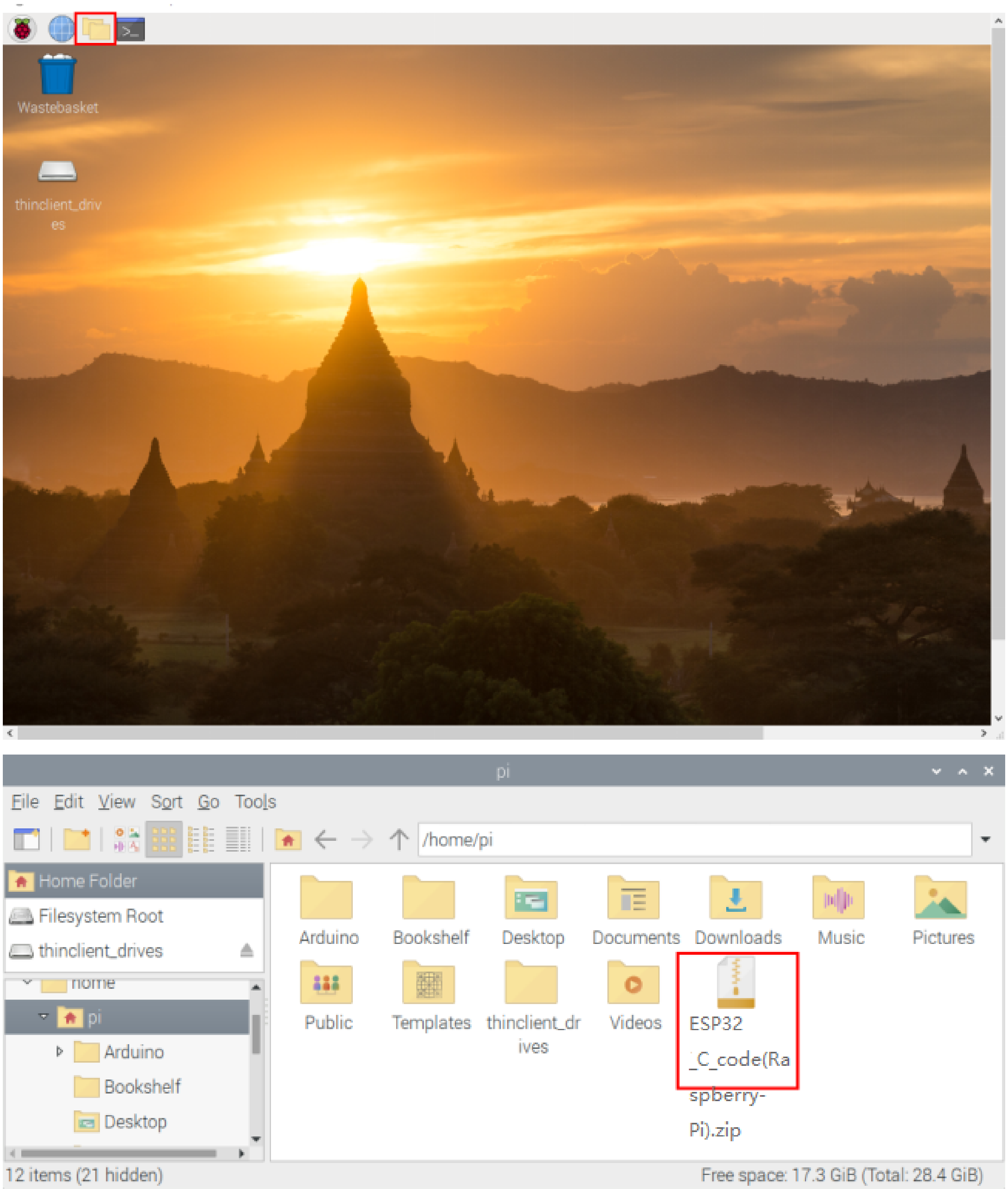
(3) Raspberry Pi+ESP32 mainboard+ESP32 Expansion Board+USB Cableare as follows

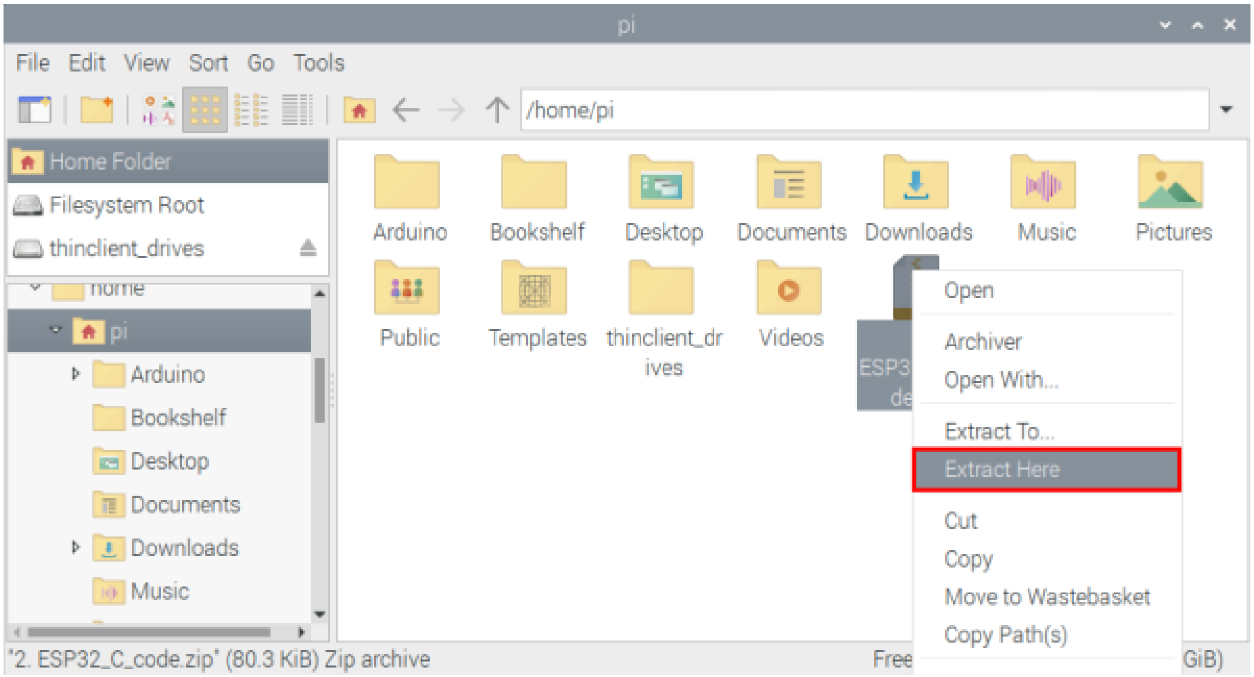


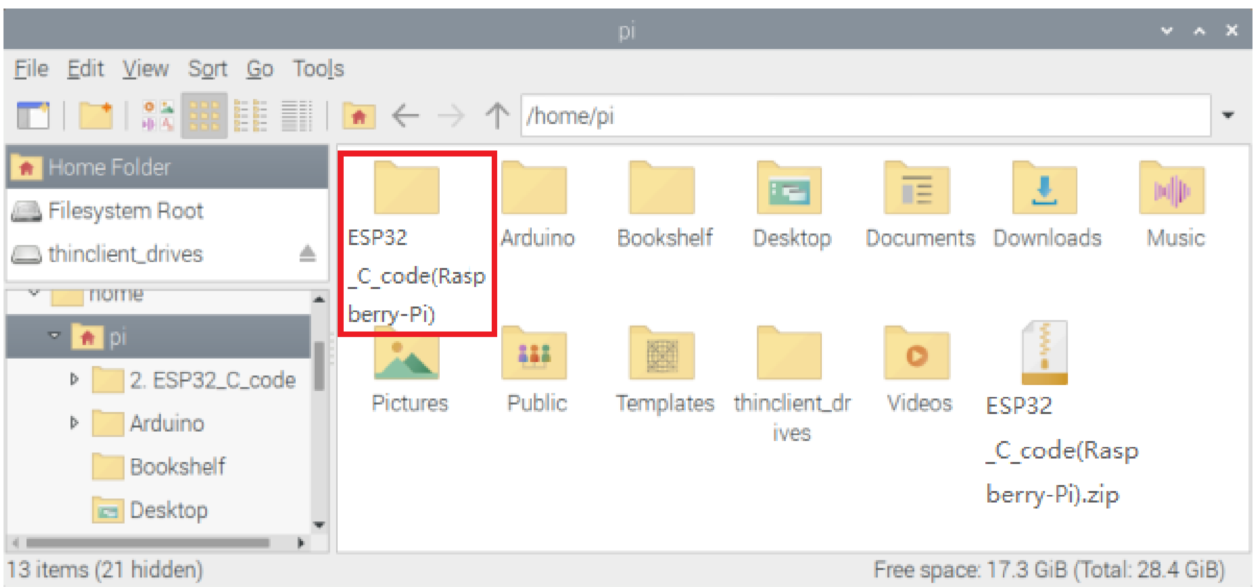


7.2.2 2.2. Copy Example Code Folder to Raspberry Pi:

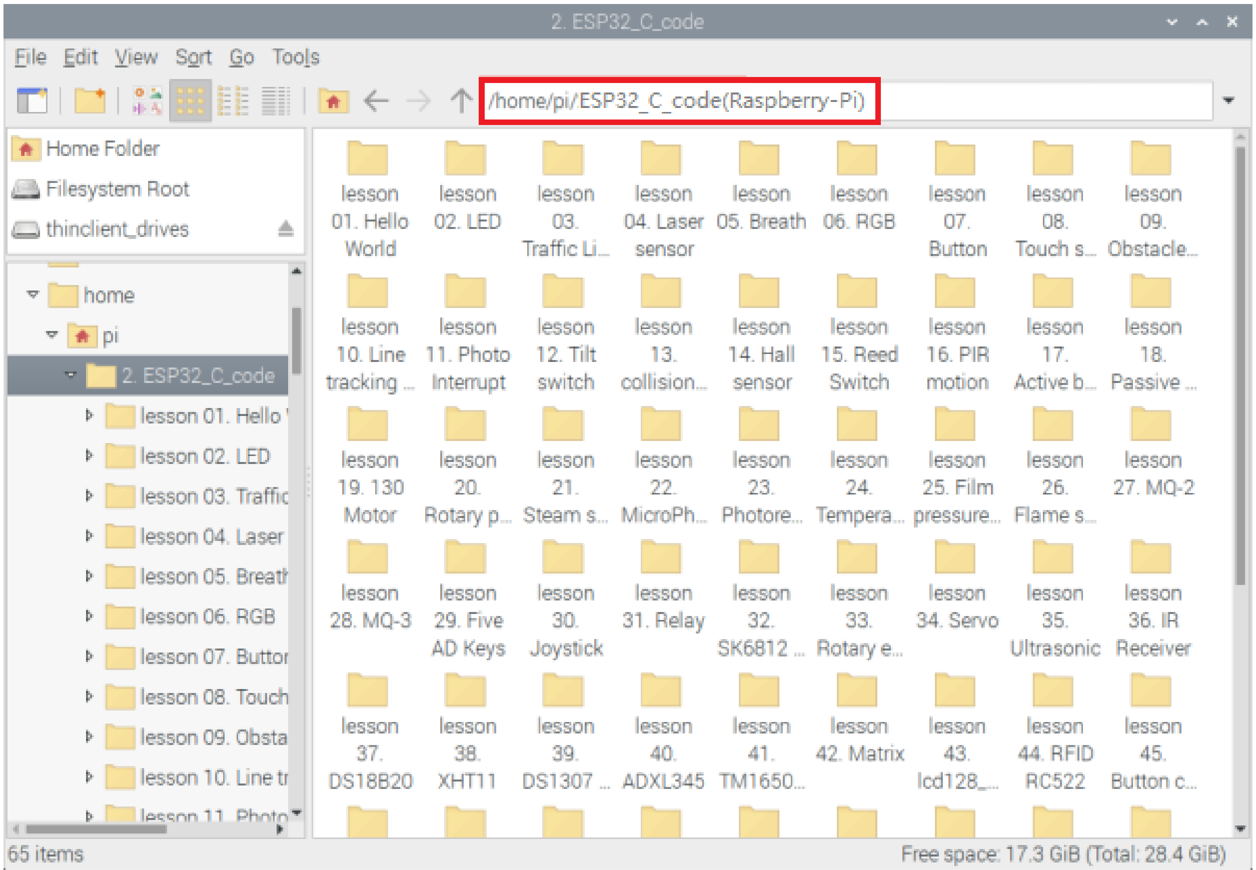
Place example code folder to the pi folder of Raspberry Pi. and extract the example code from **ESP32_C_code(Raspberry-Pi).zip** file(the default is .zip file), as shown below:


Wastebasket
thinclient_drives
pi
File Edit View Sort Go Tools
/home/pi
Home Folder
Filesystem Root
thinclient_drives
home
pi
Arduino
Bookshelf
Desktop
Arduino
Bookshelf
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
thinclient_drives
Videos
ESP32_C_code(Raspberry-Pi).zip
12 items (21 hidden)
Free space: 17.3 GiB (Total: 28.4 GiB)





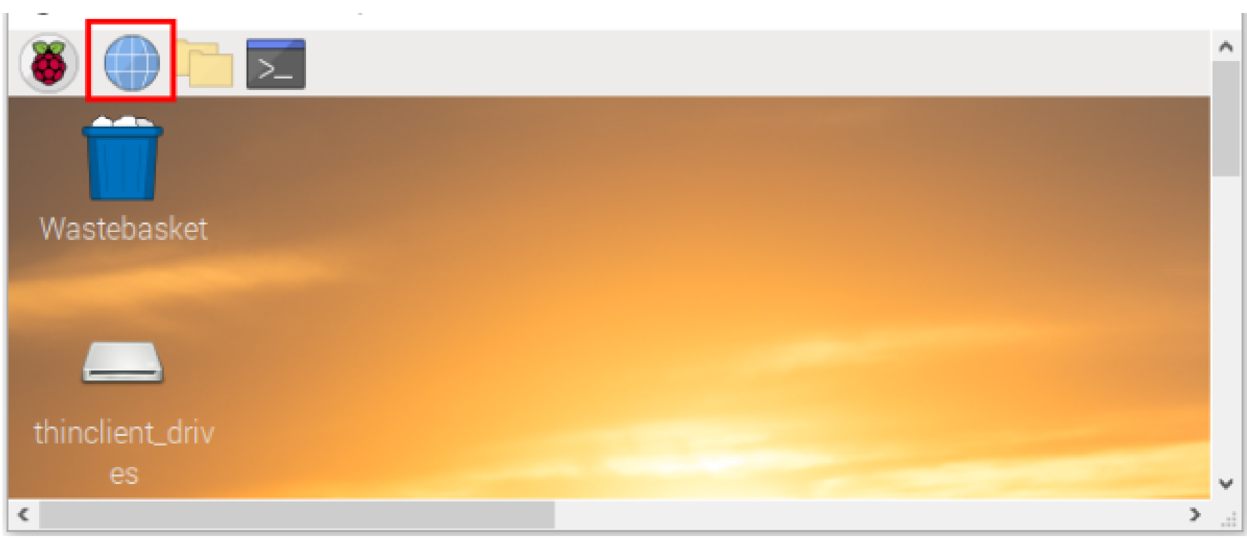
Double-click **ESP32_C_code(Raspberry-Pi)**, as shown below.



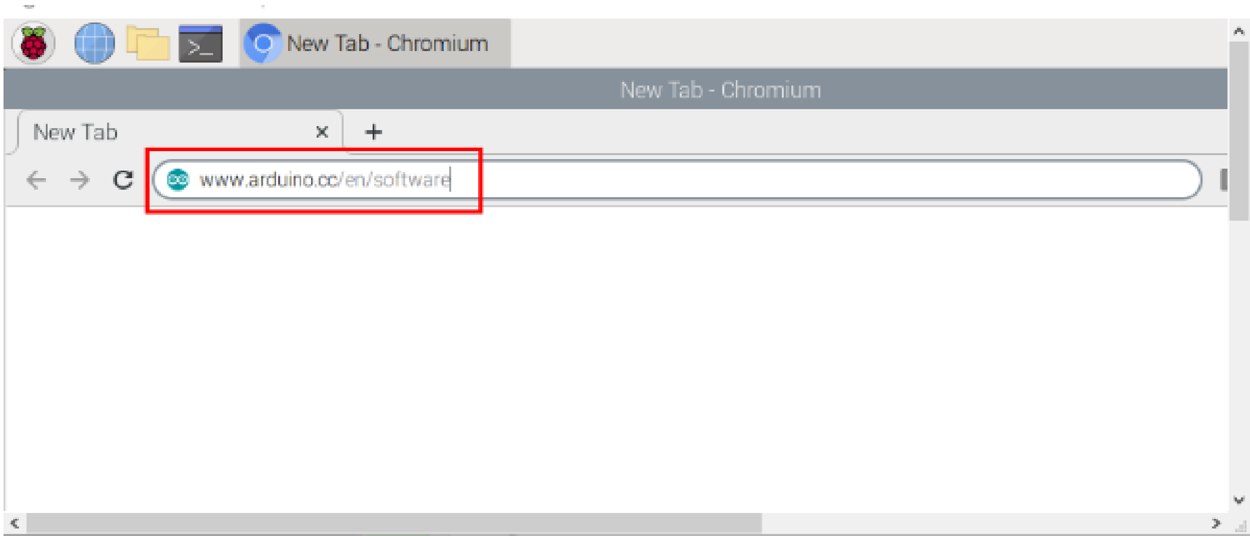
7.3 3. Linux SystemRaspberry Pi:

7.3.1 3.1. Download and install Arduino IDE

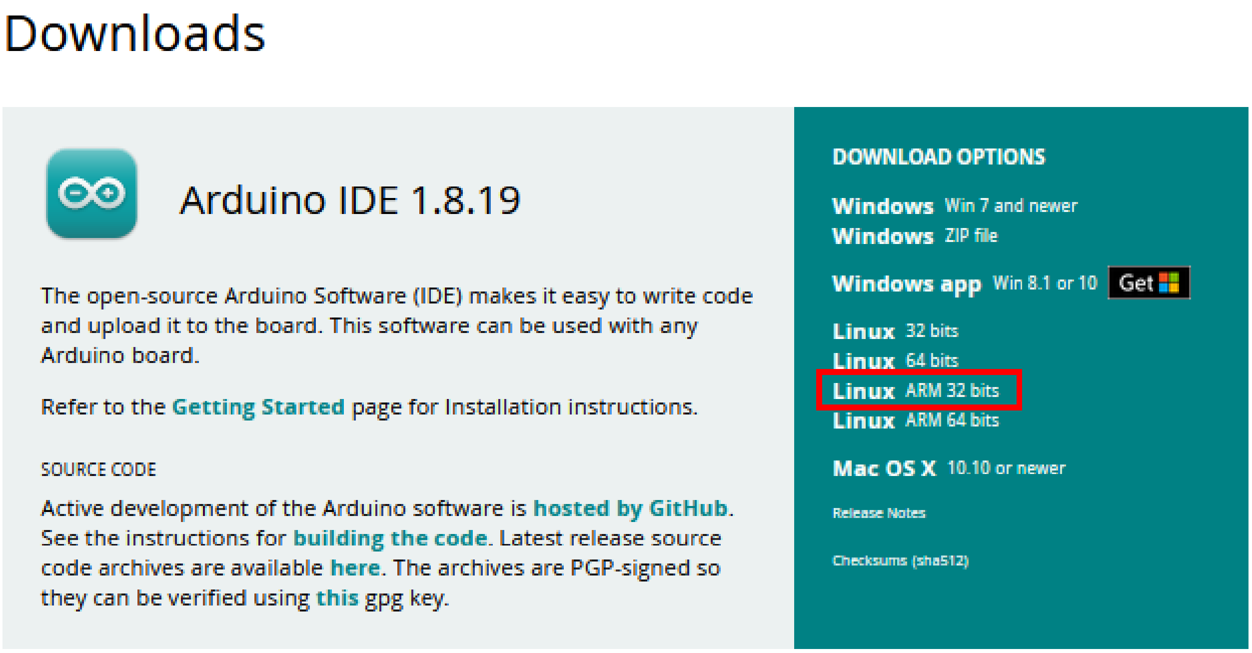
1First, click on Raspberry Pi's browser.



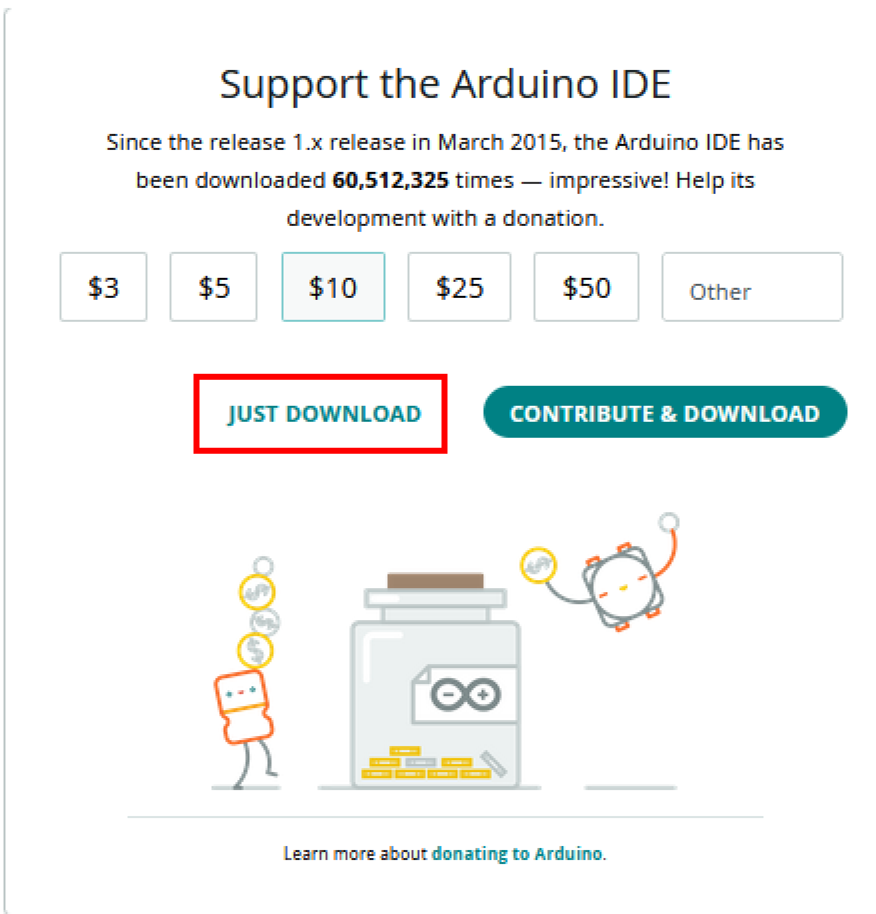
2Download Arduino IDE from the Arduino official website*www.arduino.cc/en/software* , as shown below:



(3) There are various versions of IDE for Arduino. Just download a version compatible with your system. (install the lasted Arduino IDE 1.8.19) and click "**Linux ARM 32 bits**".

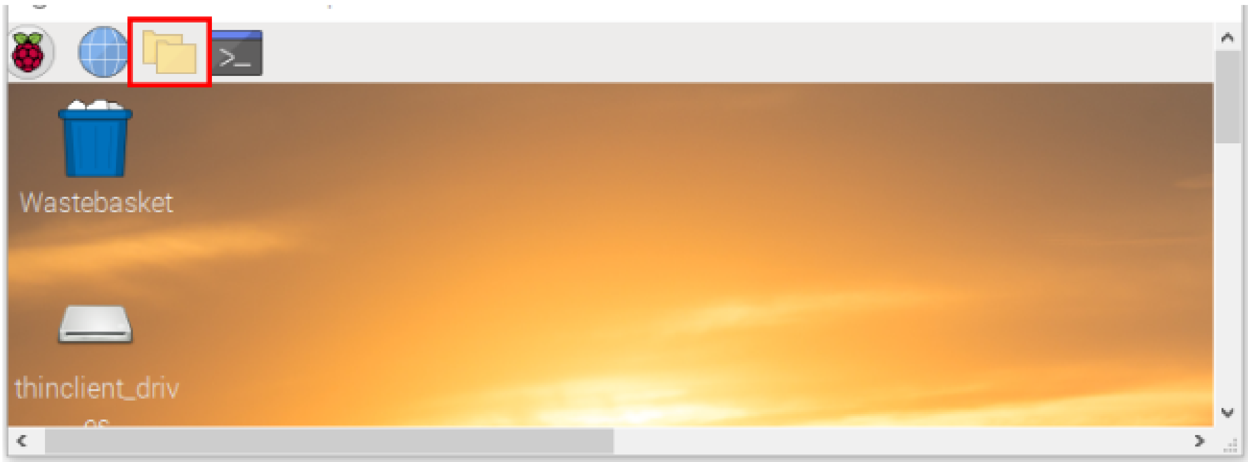


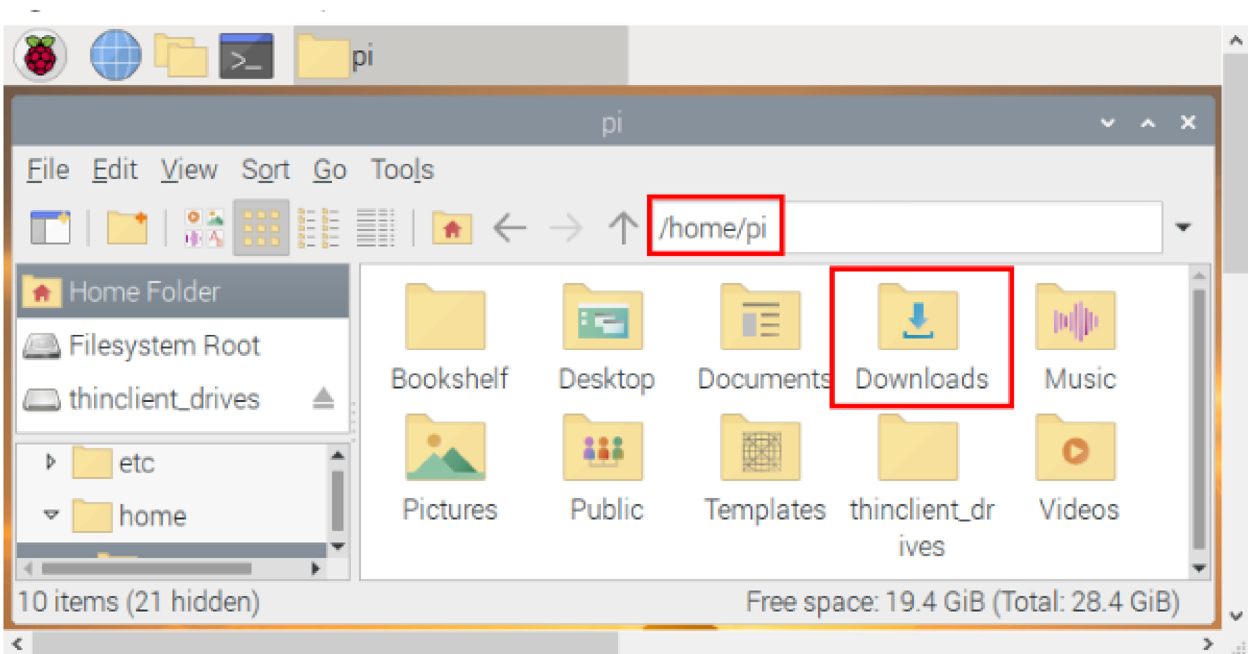
(4) You just need to click “**JUST DOWNLOAD**”.

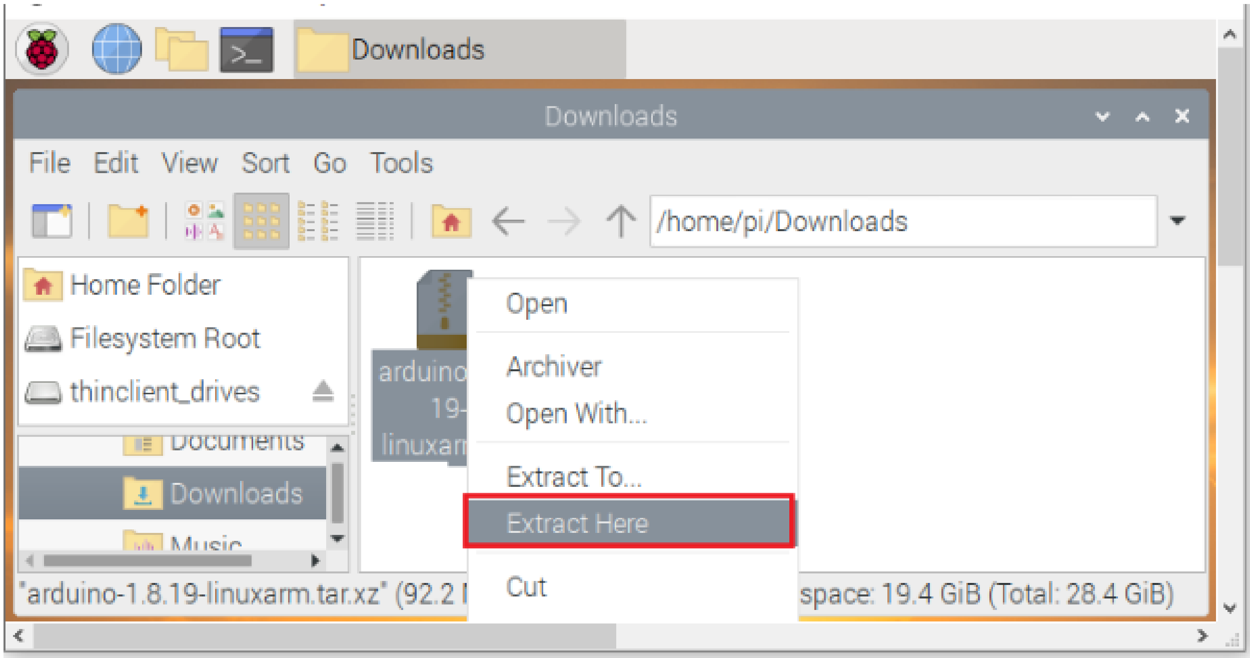


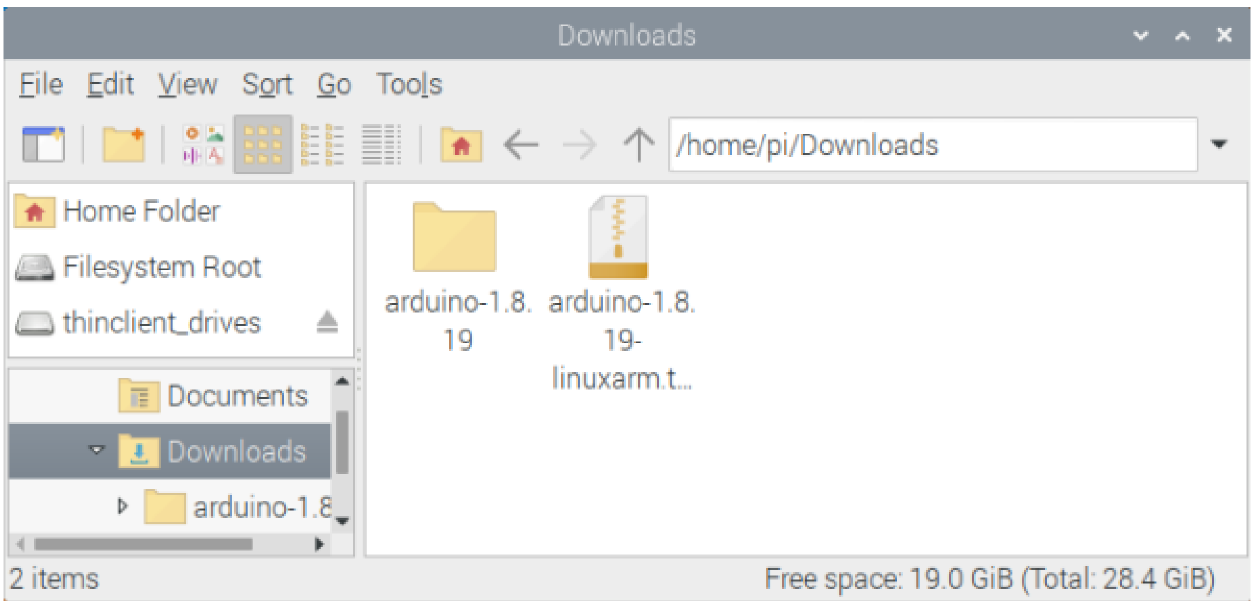
After a few seconds, the lasted Arduino IDEArduino 1.8.19 versionzip file can be directly downloaded.

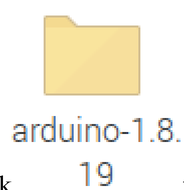
(5) Click , then find the Downloads file from the pi and tap it. Then we can see the downloaded package "**arduino-1.8.19-linuxarm.tar.xz**" and unzip it.





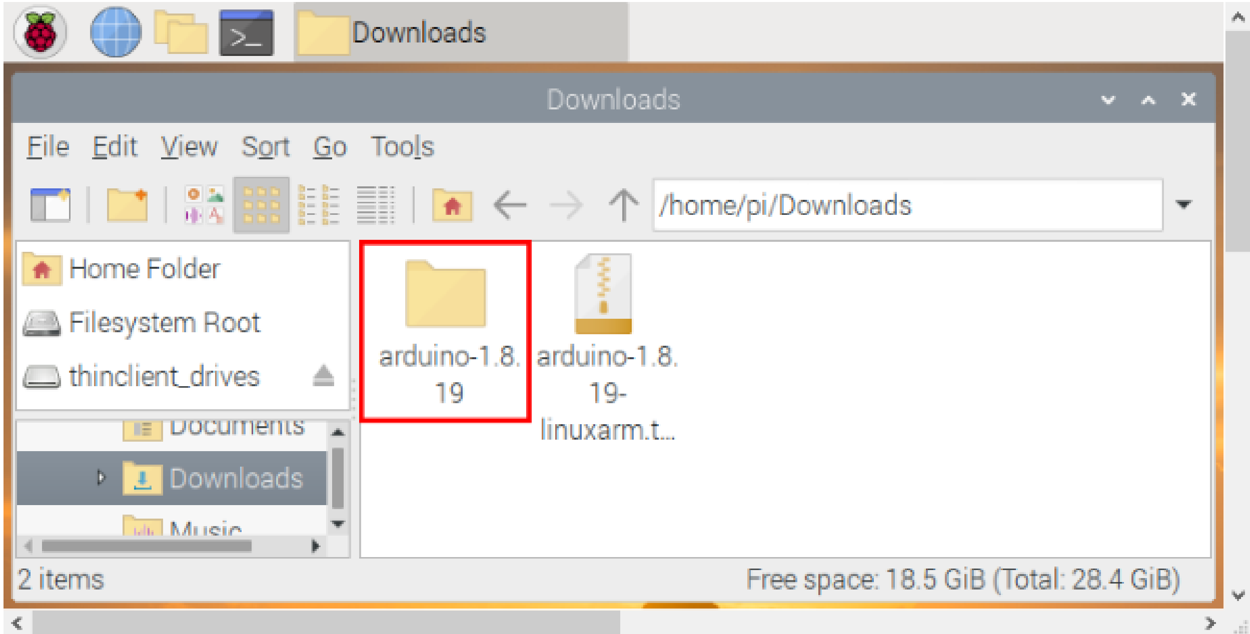


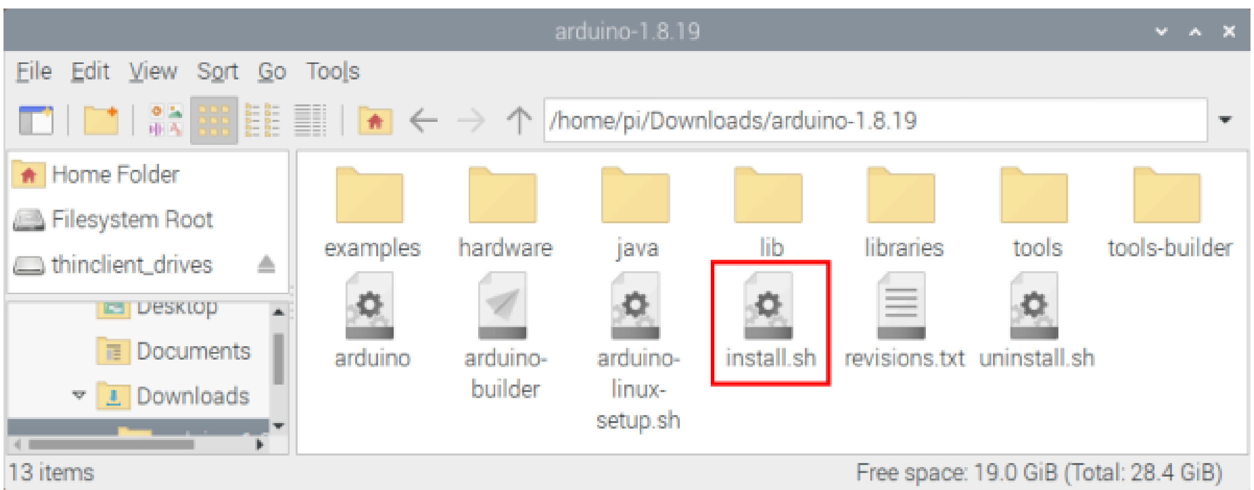


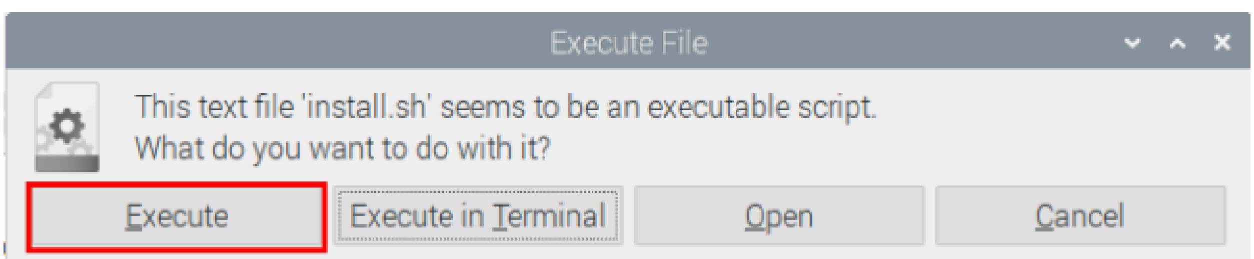
6Click 

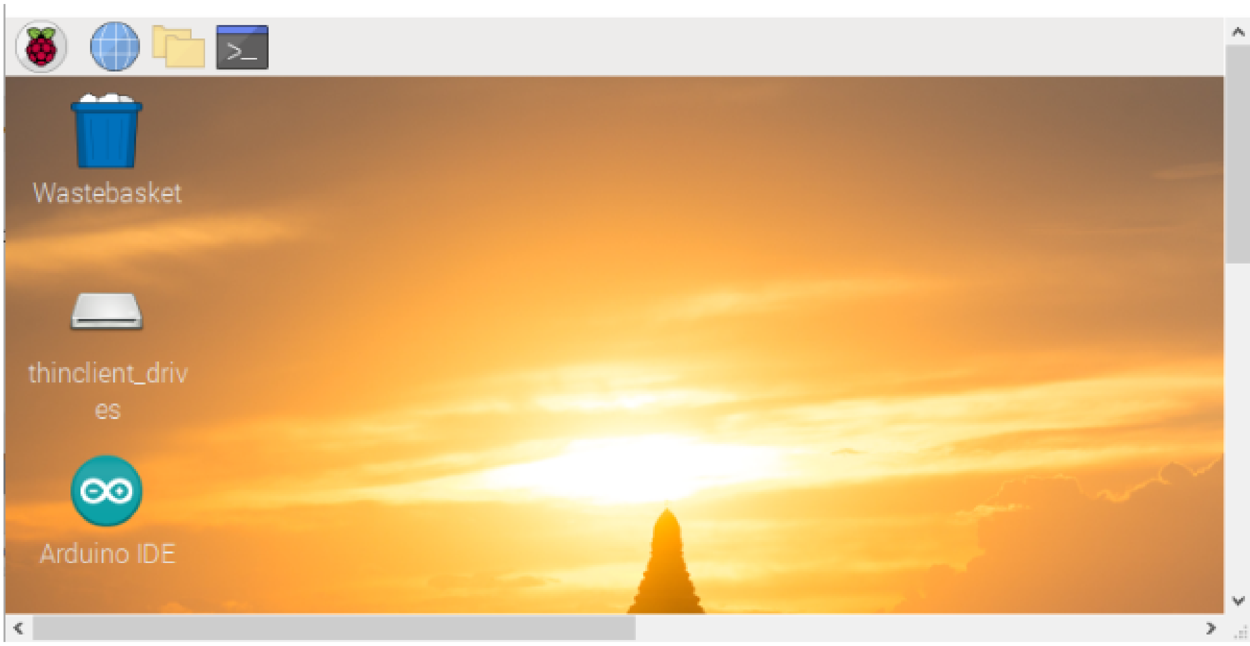
find"**install.sh**"

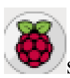
file and tap itclick "**Execute**" to install the Arduino IDE.


Downloads
Downloads
File Edit View Sort Go Tools
/home/pi/Downloads
Home Folder
Filesystem Root
thinclient_drives
Documents
Downloads
Music
arduino-1.8.19
arduino-1.8.19-linuxarm.t...
2 items
Free space: 18.5 GiB (Total: 28.4 GiB)


arduino-1.8.19
File Edit View Sort Go Tools
/home/pi/Downloads/arduino-1.8.19
Home Folder
Filesystem Root
thinclient_drives
Desktop
Documents
Downloads
examples
hardware
java
lib
libraries
tools
tools-builder
arduino
arduino-builder
arduino-linux-setup.sh
install.sh
revisions.txt
uninstall.sh
13 items
Free space: 19.0 GiB (Total: 28.4 GiB)

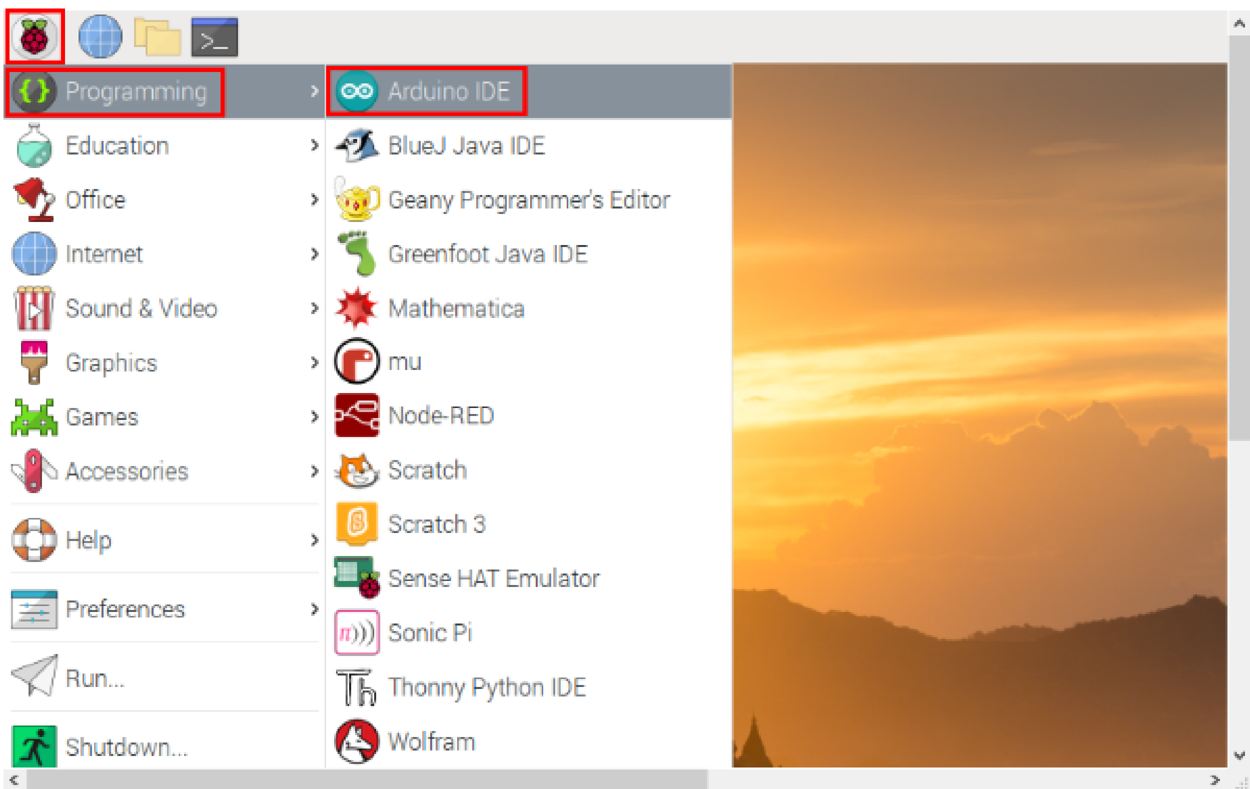

Execute File
This text file 'install.sh' seems to be an executable script.
What do you want to do with it?
Execute
Execute in Terminal
Open
Cancel

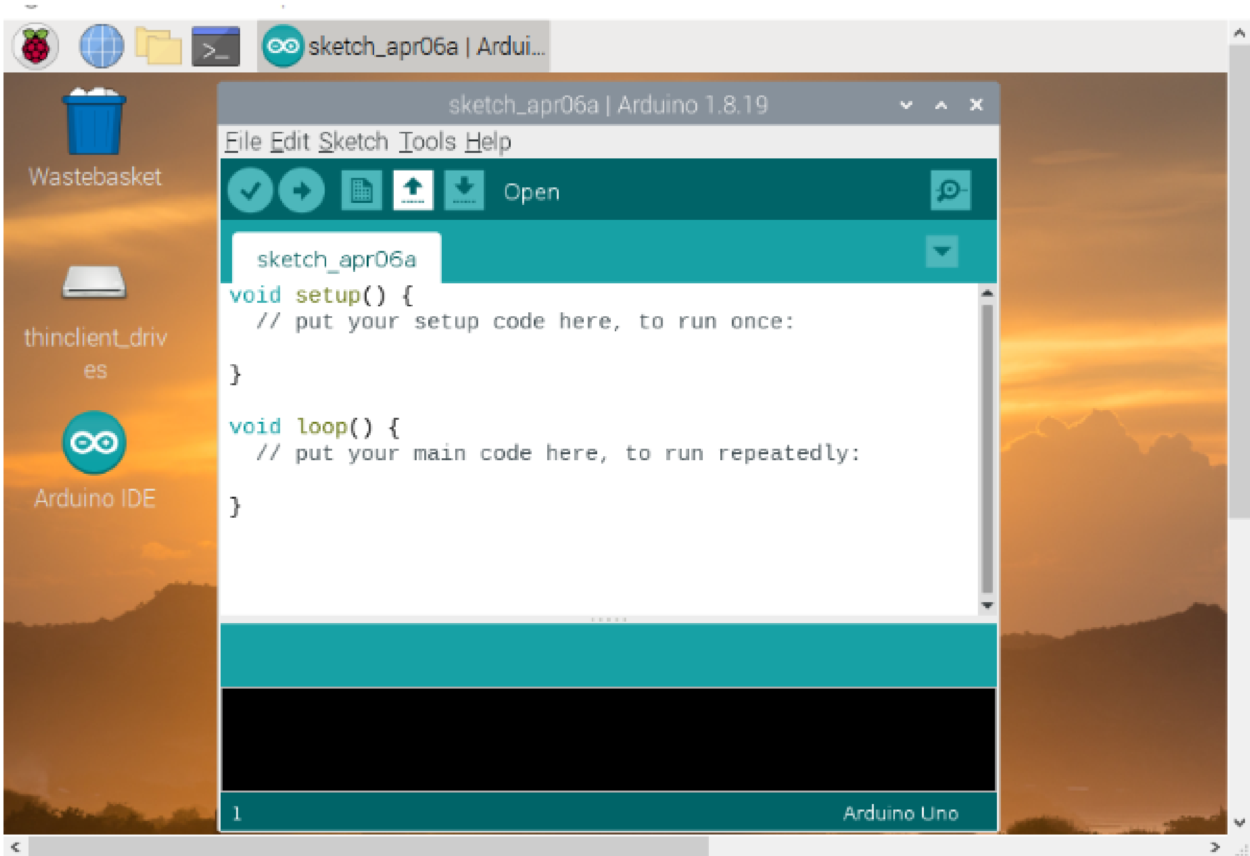


7Click  select 

and click 

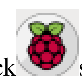
to open the Arduino IDE.





7.3.2 3.2. Install the ESP32 on Arduino IDE

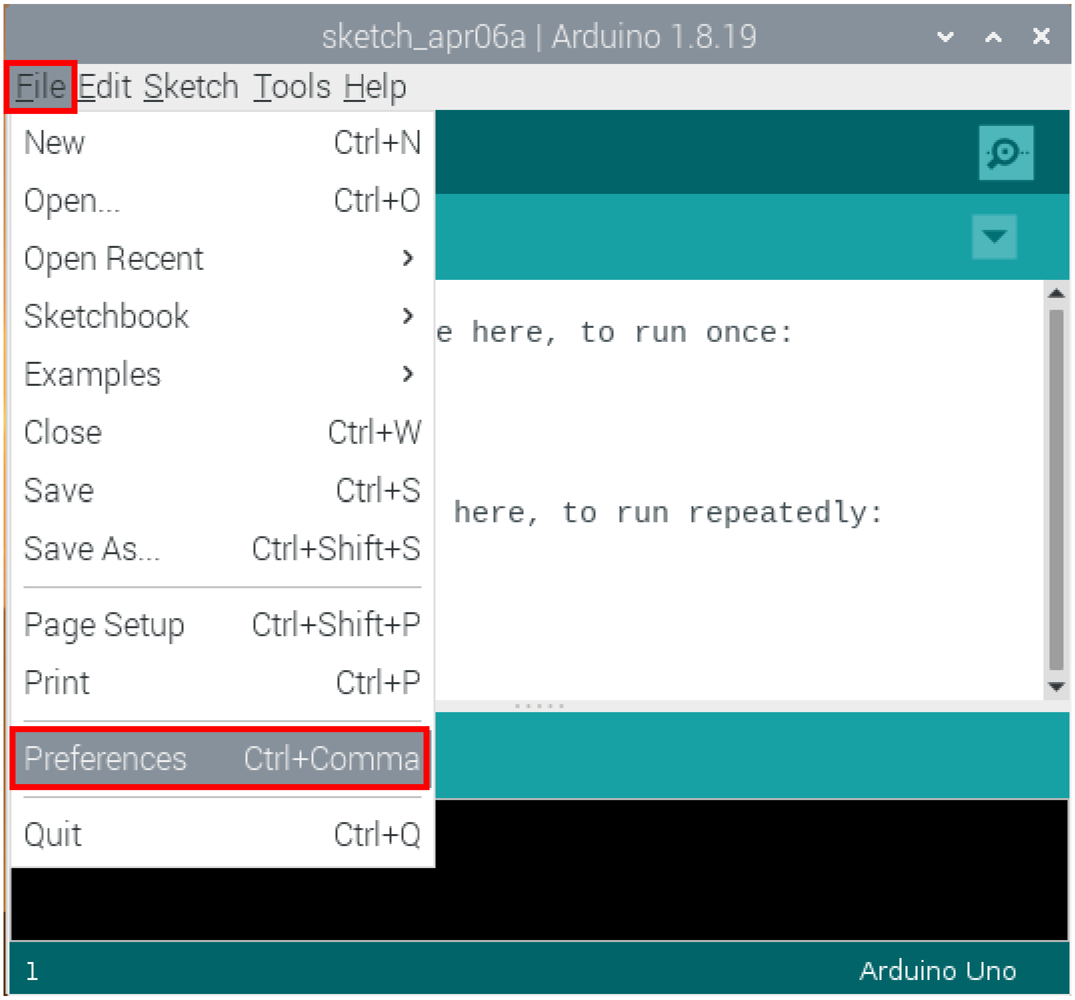
Noteyou need to download Arduino IDE 1.8.5 or advanced version to install the ESP32.

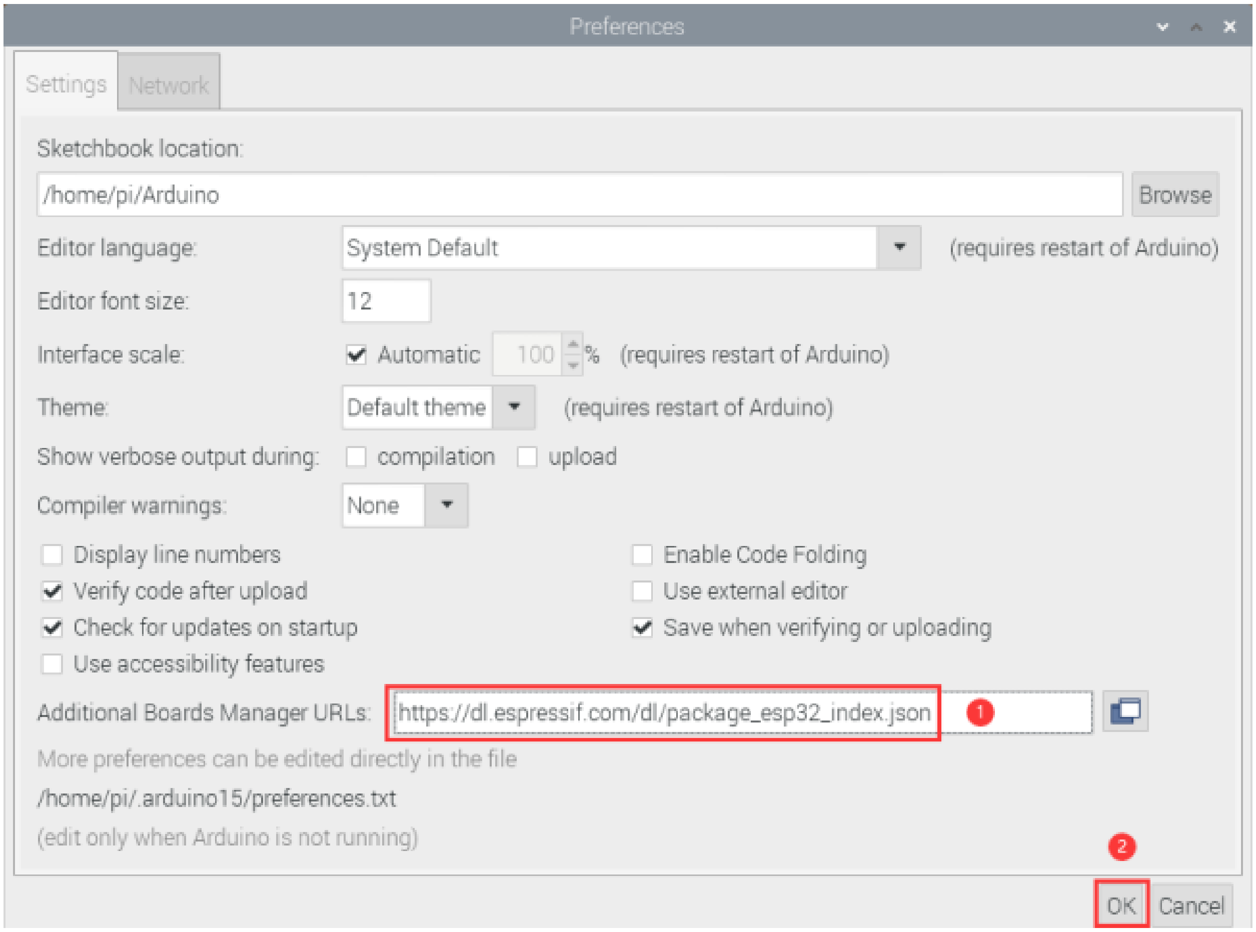
(1Click  select 

and click 

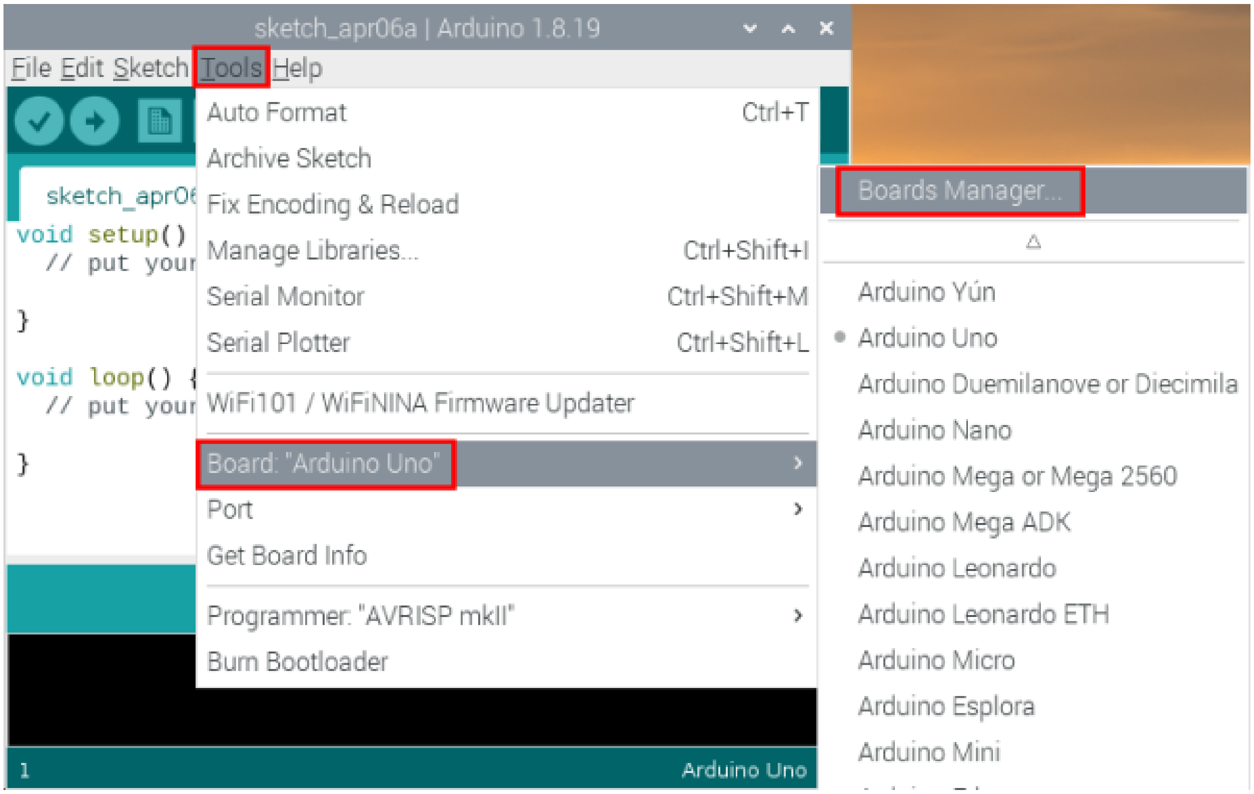
to open the Arduino IDE.

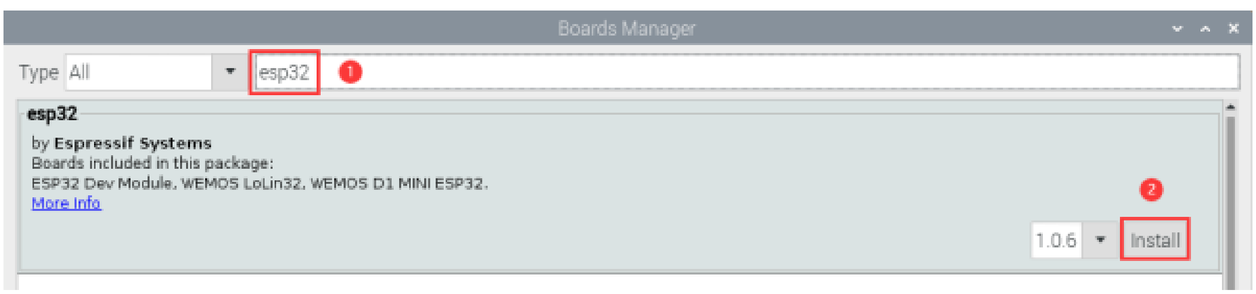
(2 Click "**File**" →"**Preferences**"copy the website address https://dl.espressif.com/dl/package_esp32_index.json in the "**Additional Boards Manager URLs:**" and click "**OK**" .

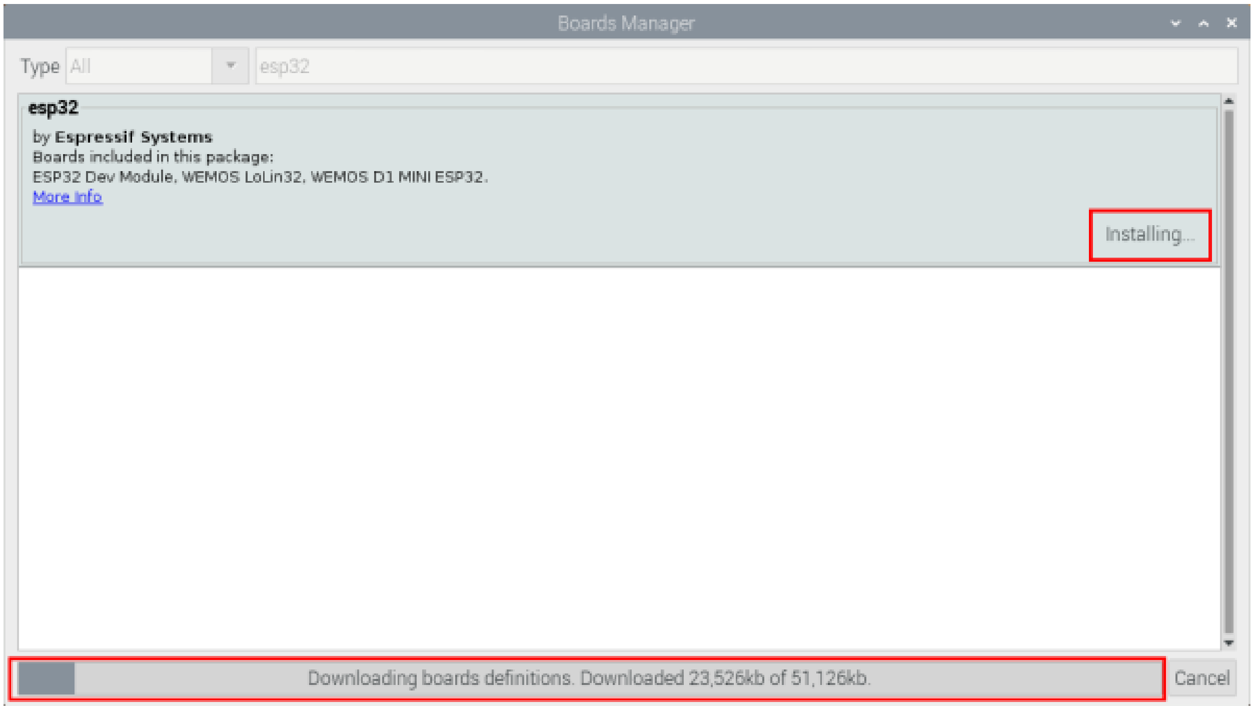

sketch_apr06a | Arduino 1.8.19
File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
e here, to run once:
here, to run repeatedly:
1
Arduino Uno



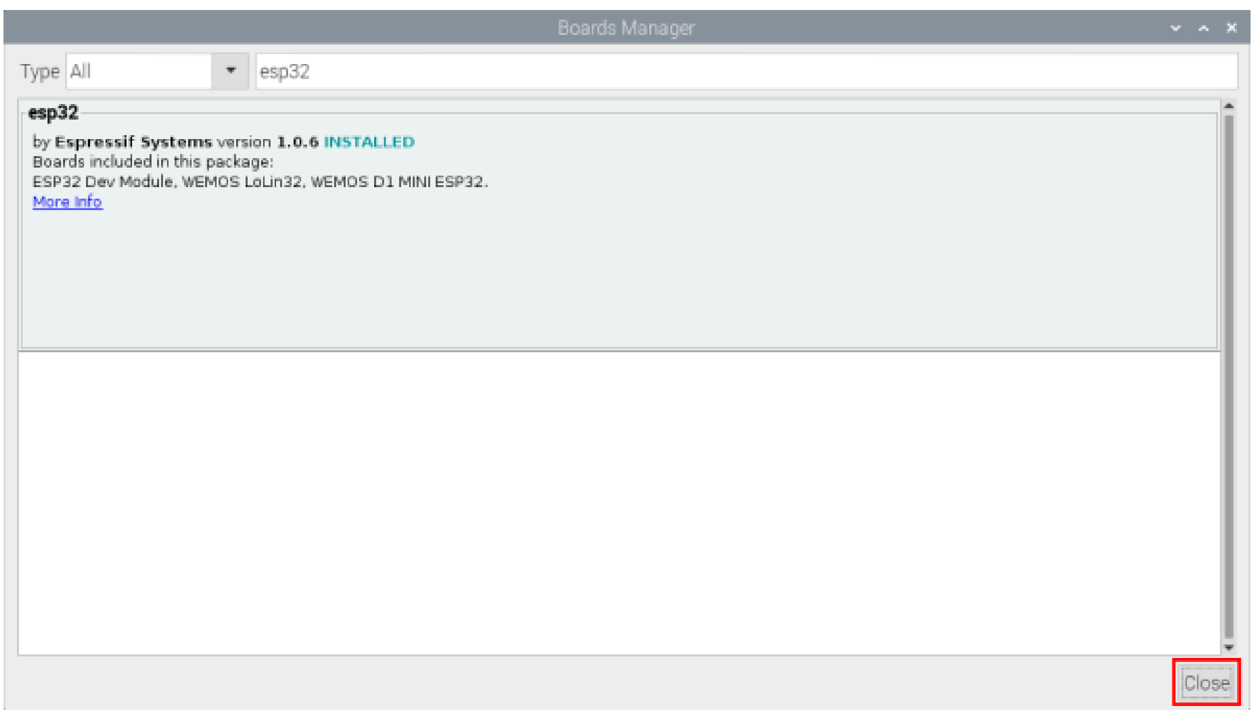
3 Click "**Tools**" → "**Board:**" then click "**Boards Manager…**"to enter "**Boards Manager**". Enter "**ESP32**" as follows, then click "**Install**" .


sketch_apr06a | Arduino 1.8.19
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino Yún
Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo
Arduino Leonardo ETH
Arduino Micro
Arduino Esplora
Arduino Mini
sketch_apr06
void setup()
// put your
}
void loop() {
// put your
}
1
Arduino Uno


Boards Manager
Type All
esp32
1
esp32
by Espressif Systems
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32, WEMOS D1 MINI ESP32.
More Info
2
1.0.6
Install



(4) After installing, click "**Close**".

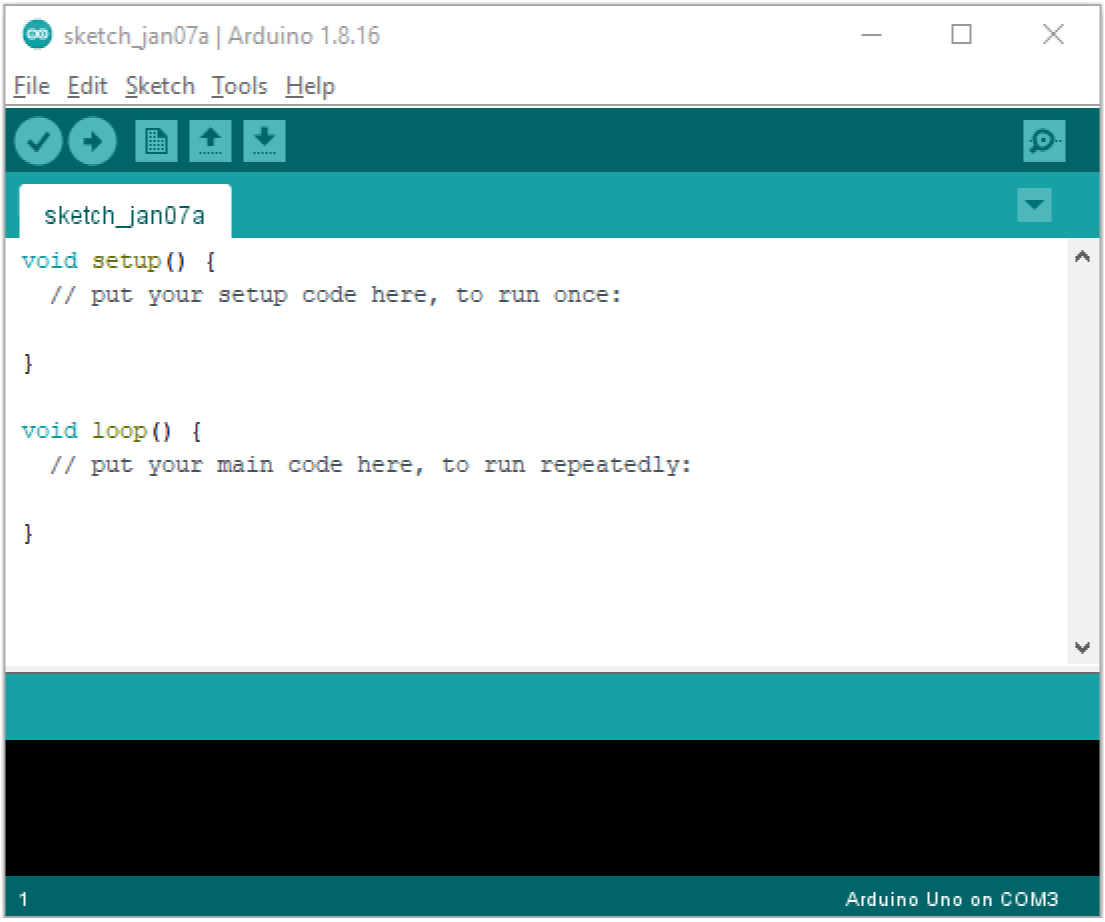


7.3.3 3.3. Arduino IDE Setting

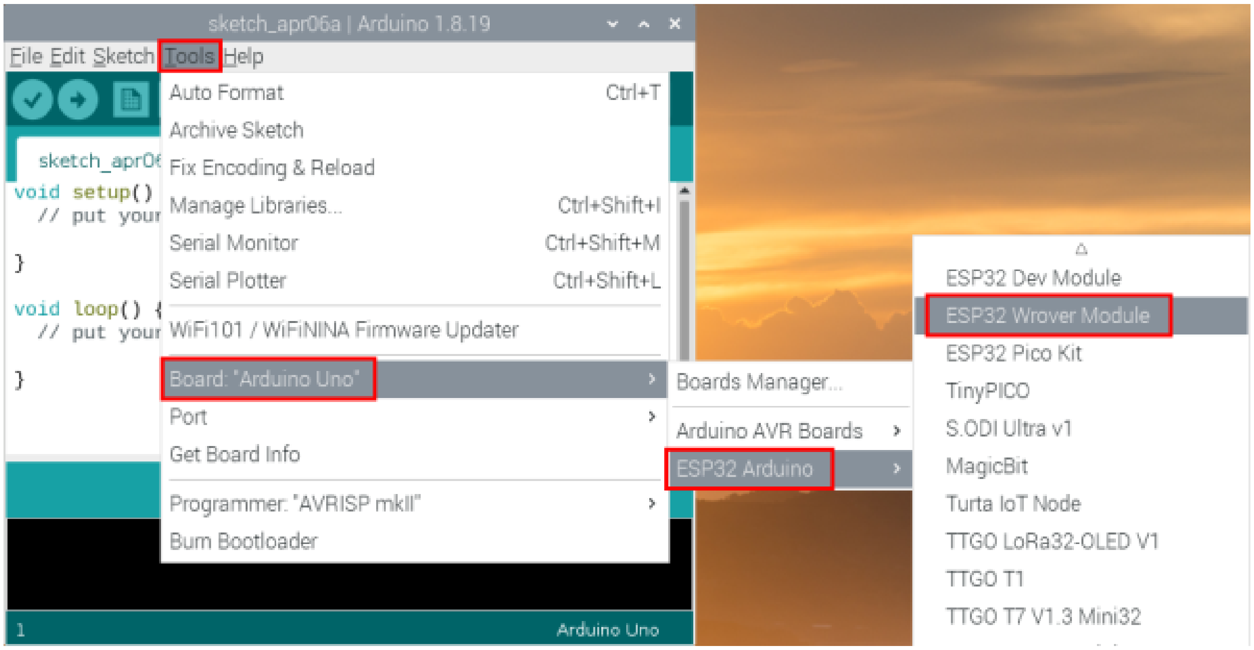
Click select 

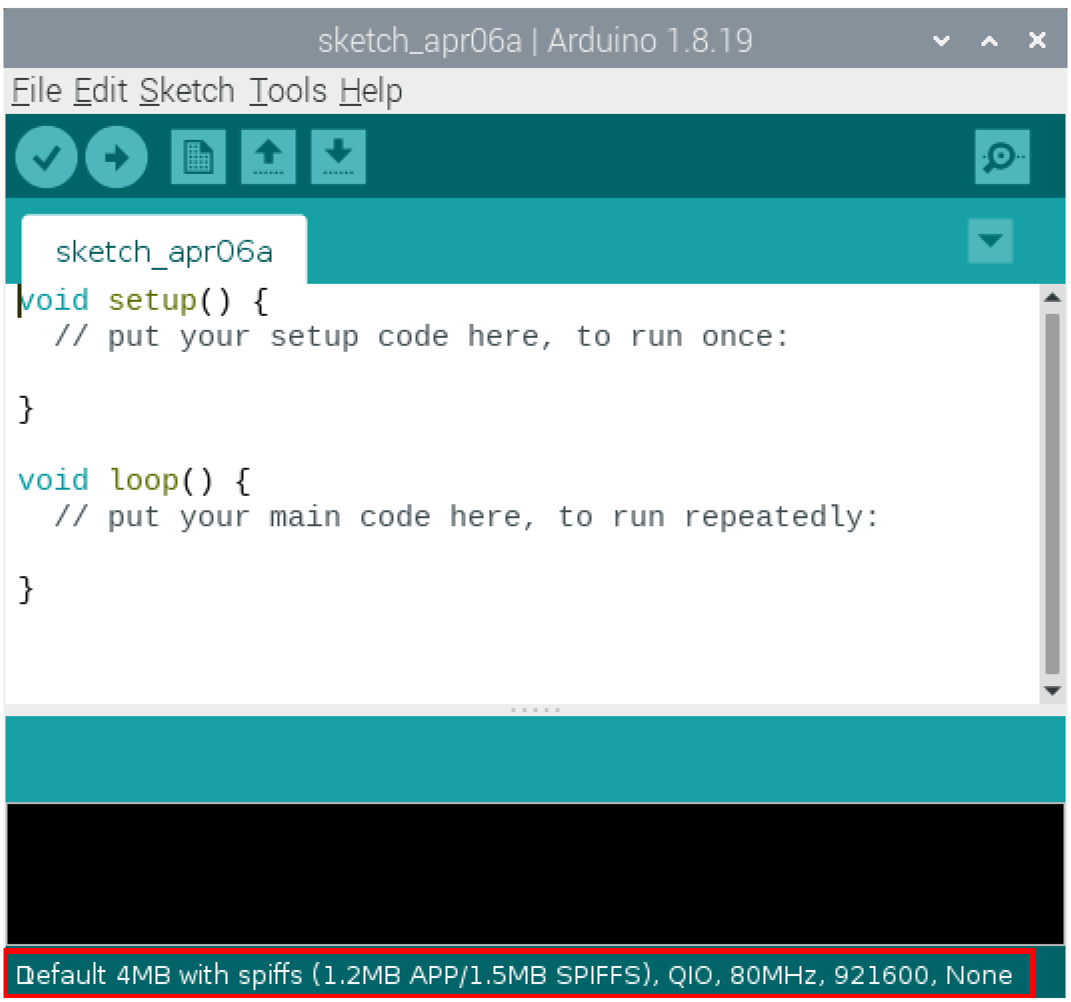
and click 

to open the Arduino IDE.



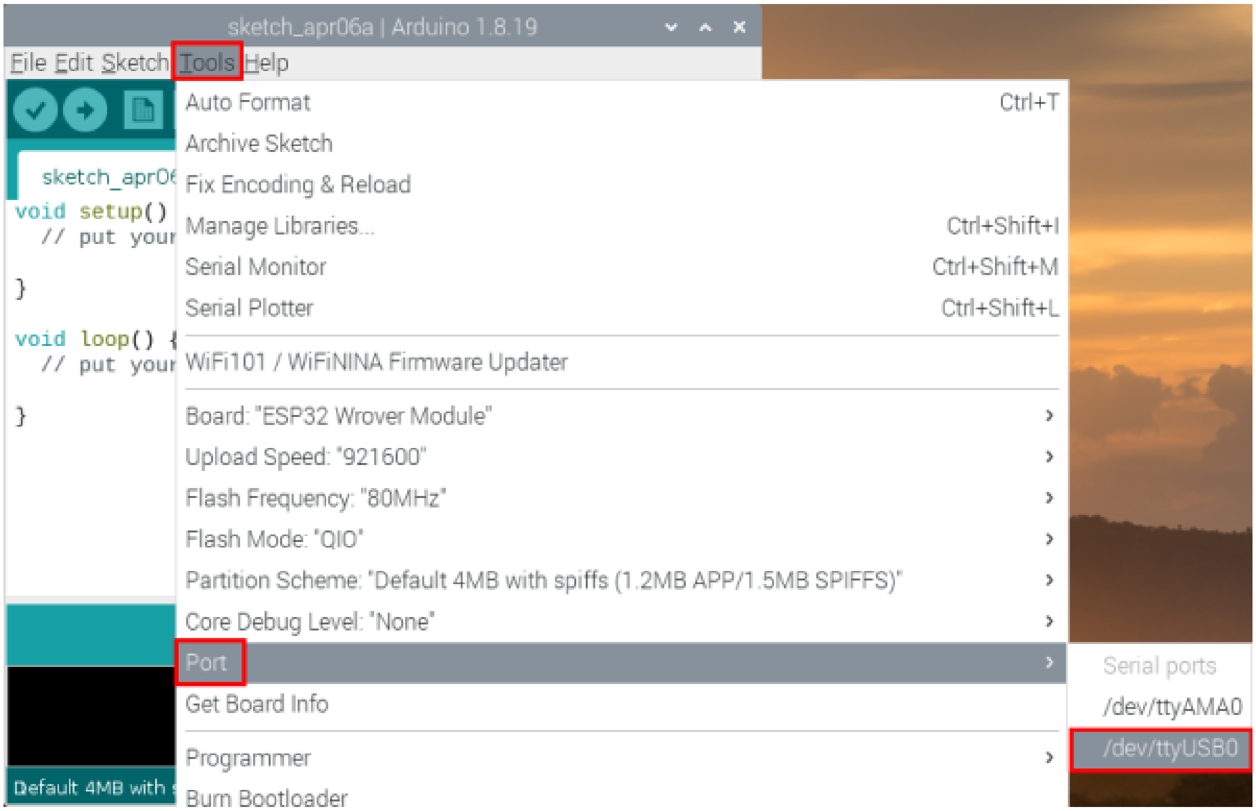
When downloading the sketch to the board, you must select the correct name of Arduino board that matches the board connected to your computer. As shown below: (Note: we use the ESP32 board in this tutorial; therefore, we select ESP32)



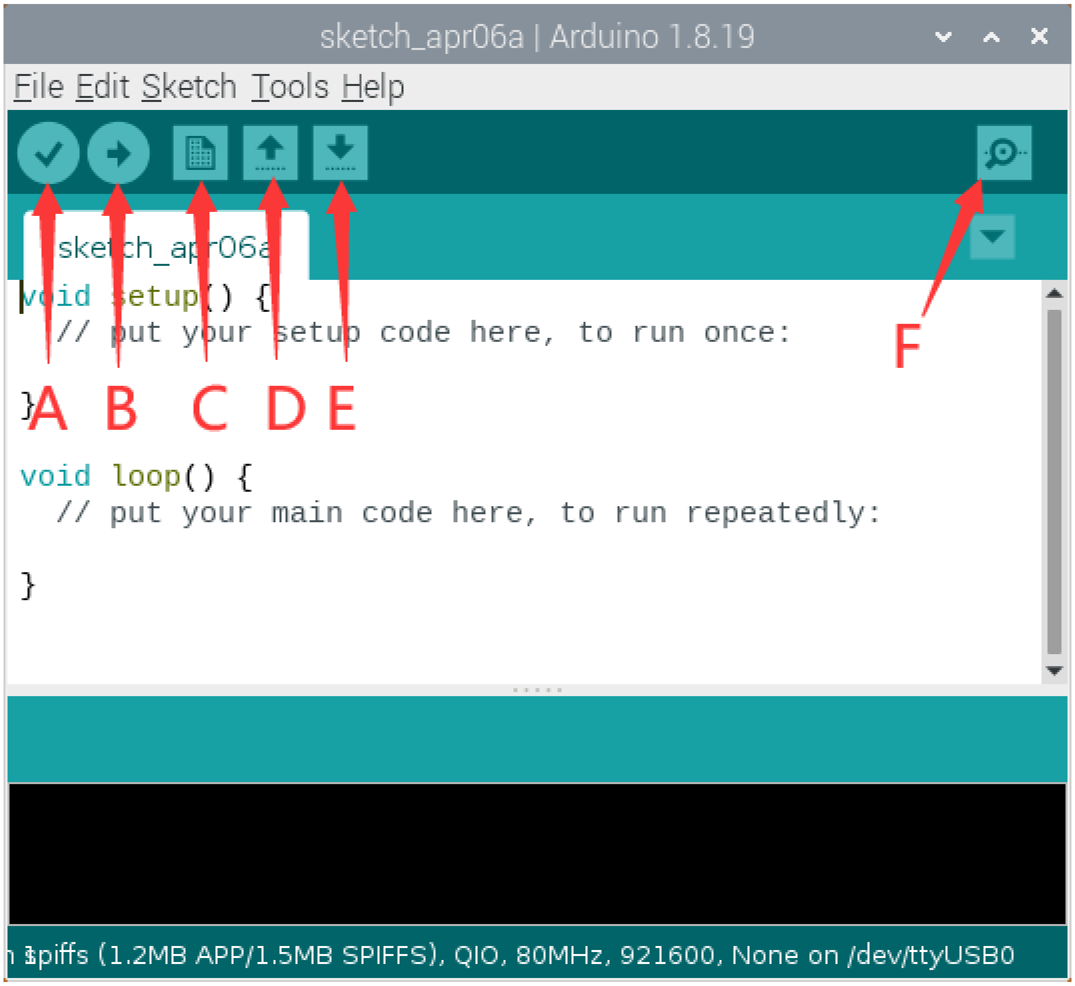


Then select the correct COM port (you can see the corresponding COM port after the ESP32 is connected to the

Raspberry Pi via a USB cable.).




sketch_apr06a | Arduino 1.8.19
File Edit Sketch Tools Help
sketch_apr06a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
spiffs (1.2MB APP/1.5MB SPIFFS), QIO, 80MHz, 921600, None on /dev/ttyUSB0



A- Used to verify whether there is any compiling mistakes or not. B- Used to upload the sketch to your Arduino board. C- Used to create shortcut window of a new sketch. D- Used to directly open an example sketch. E- Used to save the sketch. F- Used to send the serial data received from board to the serial monitor.

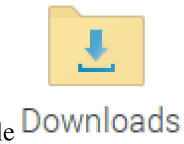
7.4 4. How to Add Libraries? :

7.4.1 4.1. What are Libraries ?:

Librariesare a collection of code that make it easy for you to connect sensors,displays, modules, etc. For example, the built-in LiquidCrystal library helps talk to LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. (https://www.arduino.cc/en/Reference/Libraries)

7.4.2 4.2. How to Install a Library ?:

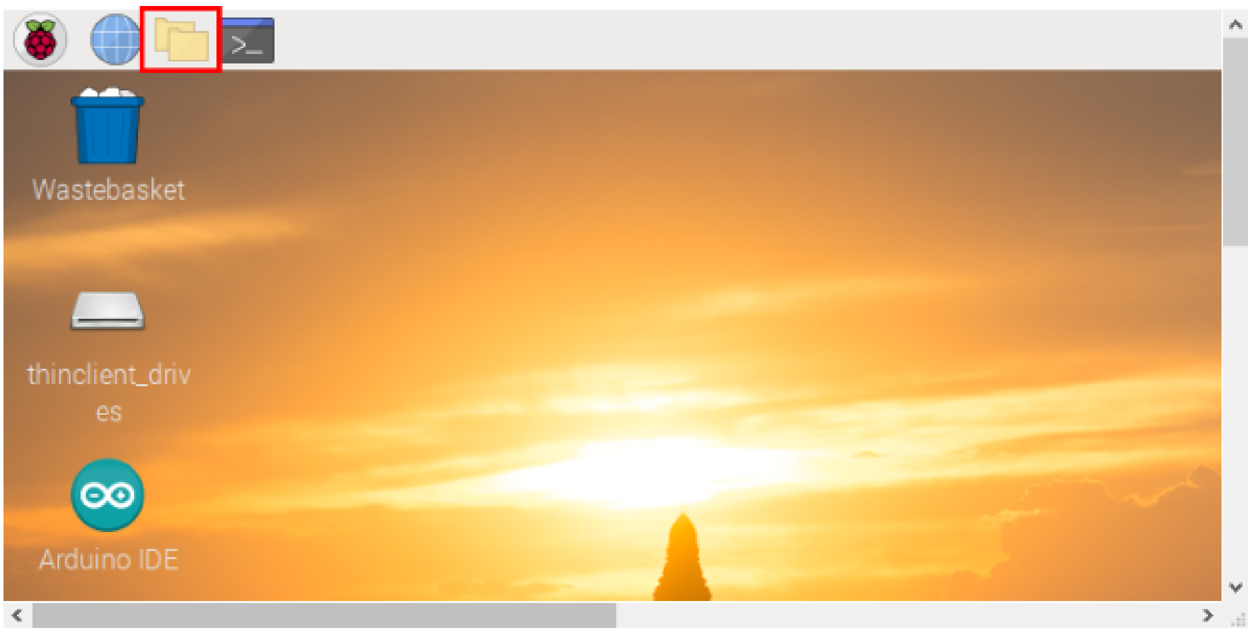
Here we will introduce the most simple way to add libraries .

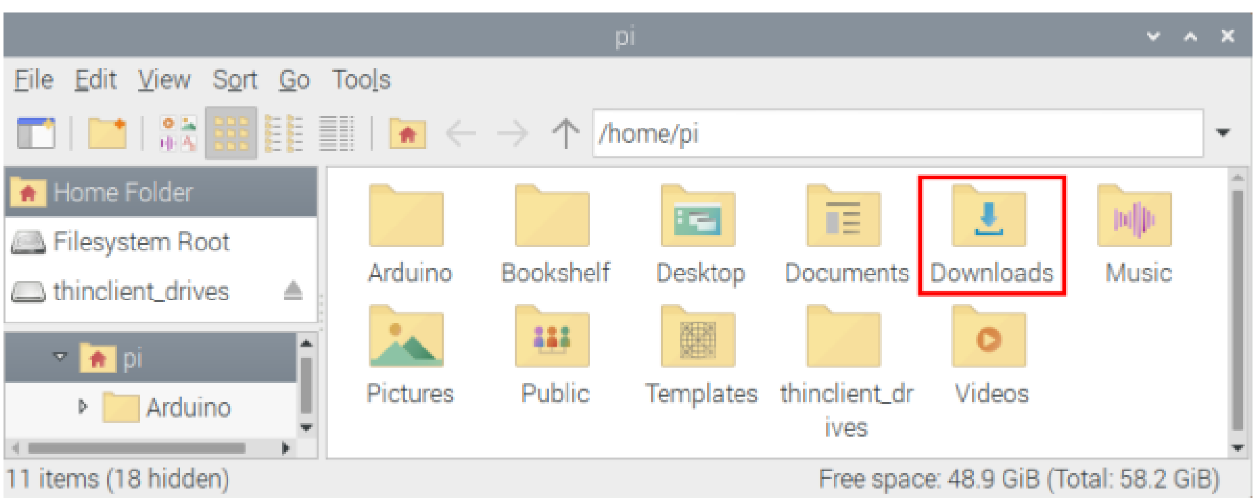
Step 1: Click  tap "Downloads" file 

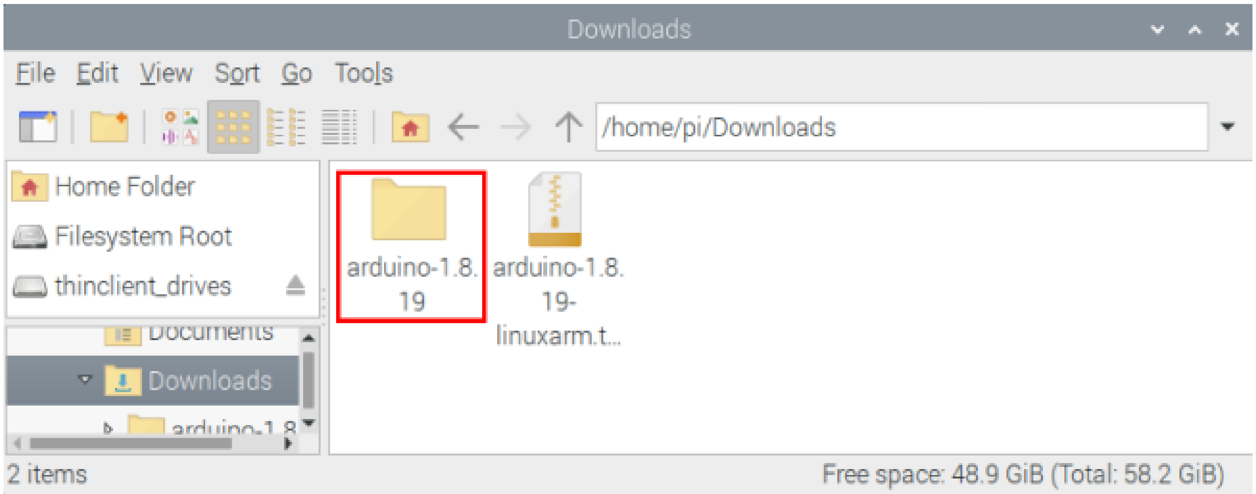
and click "**arduino-1.8.19**" file 

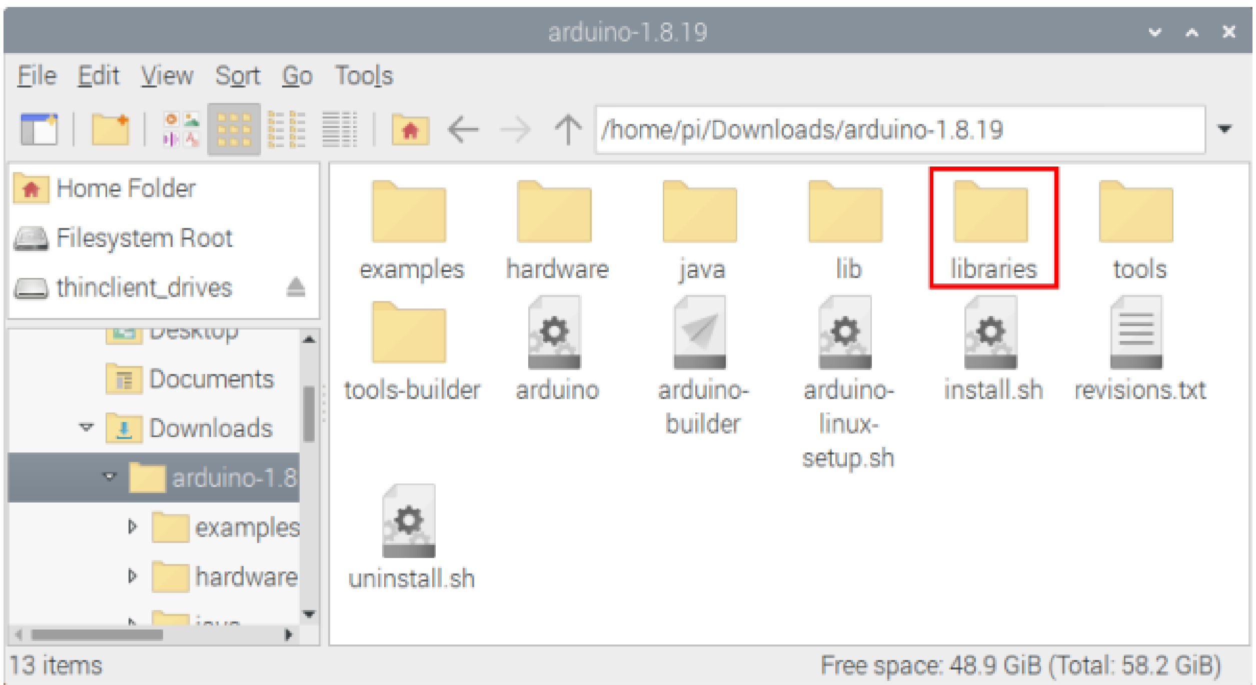
then find and click"**libraries**" file 

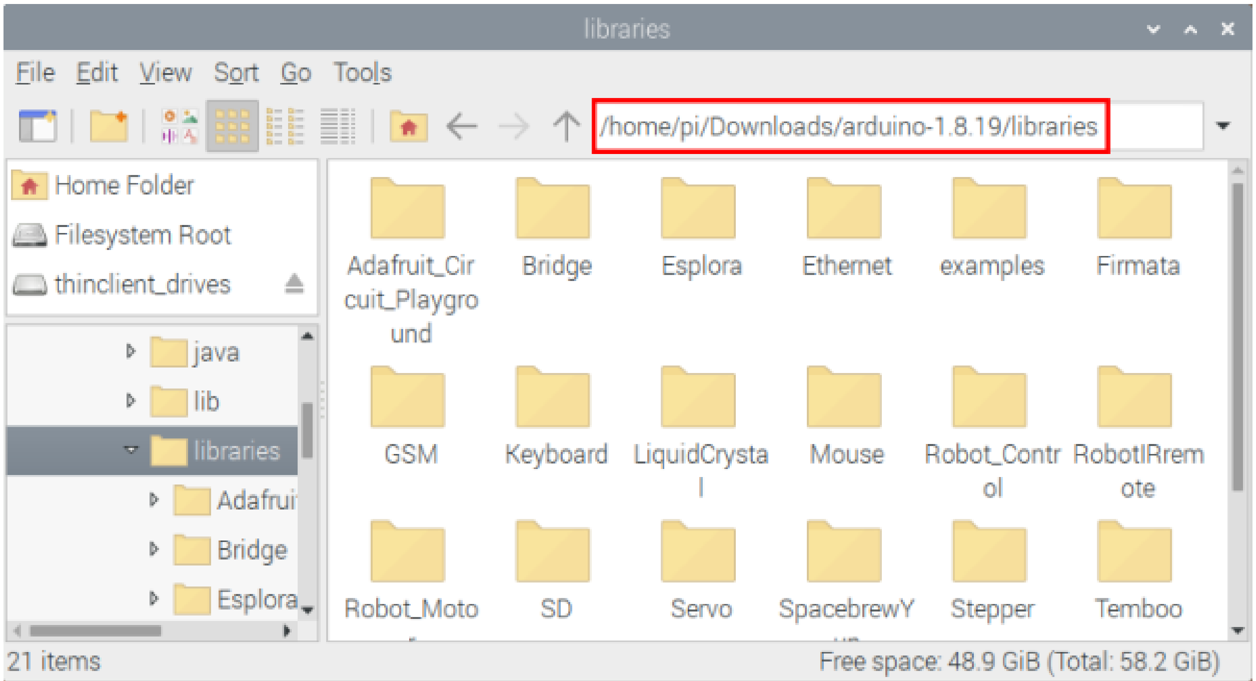
from the "**arduino-1.8.19**" file.



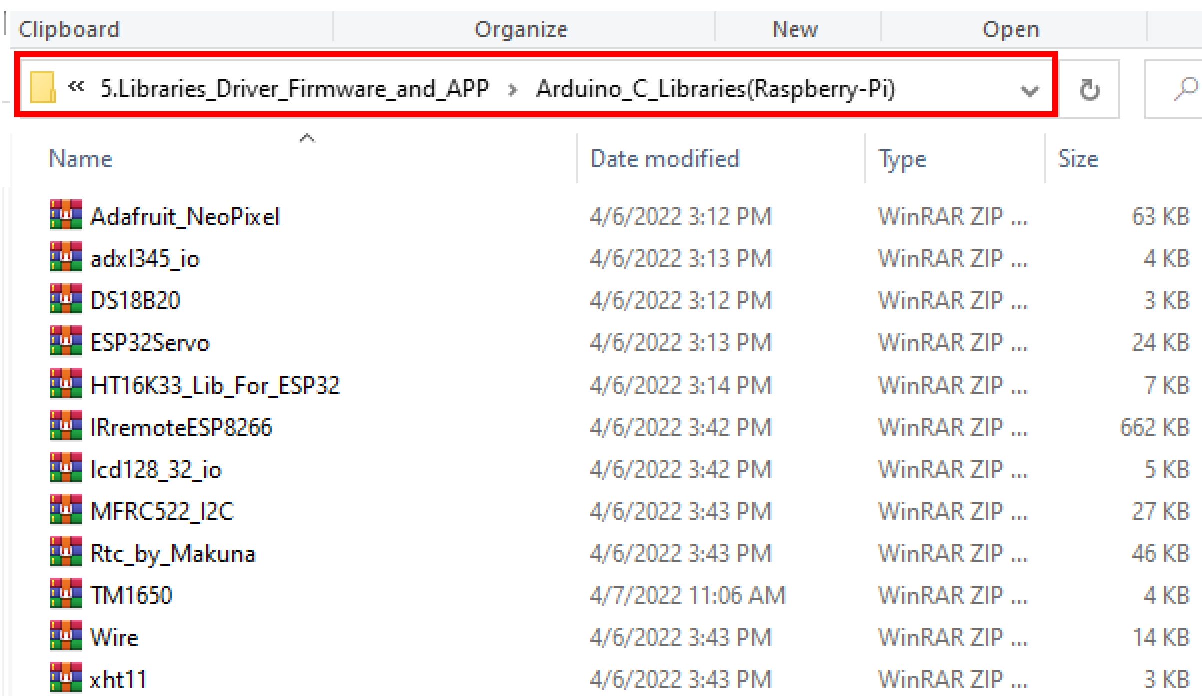


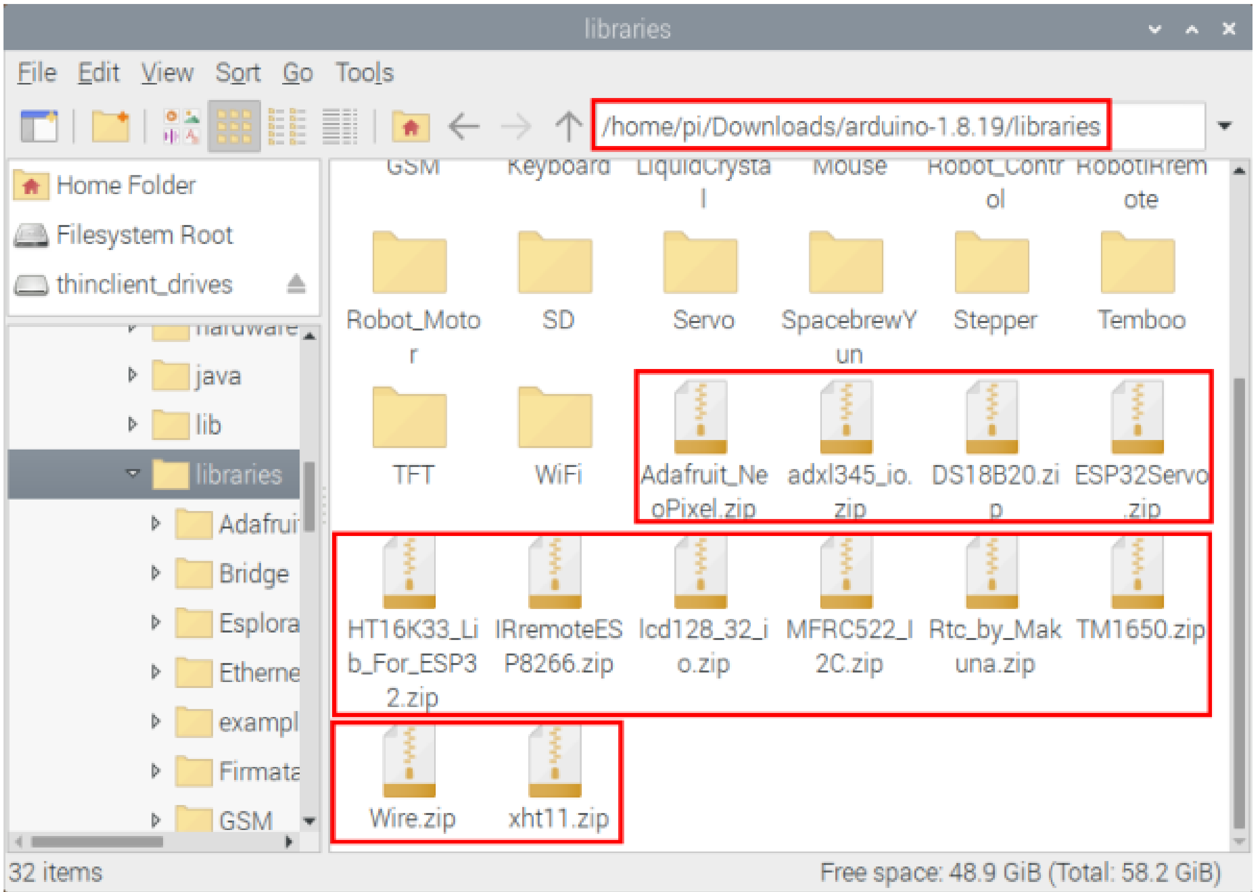

Downloads
File Edit View Sort Go Tools
/home/pi/Downloads
Home Folder
Filesystem Root
thinclient_drives
Documents
Downloads
arduino-1.8.19
arduino-1.8.19-linuxarm.t...
2 items
Free space: 48.9 GiB (Total: 58.2 GiB)


arduino-1.8.19
File Edit View Sort Go Tools
/home/pi/Downloads/arduino-1.8.19
Home Folder
Filesystem Root
thinclient_drives
Documents
Downloads
arduino-1.8
examples
hardware
examples
hardware
java
lib
libraries
tools
tools-builder
arduino
arduino-builder
arduino-linux-setup.sh
install.sh
revisions.txt
uninstall.sh
13 items
Free space: 48.9 GiB (Total: 58.2 GiB)

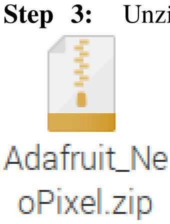


Step 2: Copy and paste the **Arduino_C_Libraries(Raspberry-Pi)** file (default .ZIP file) from the provided Arduino Libraries folder into the Libraries file opened in the first stepthe route is/home/pi/Downloads/arduino-1.8.19/libraries.

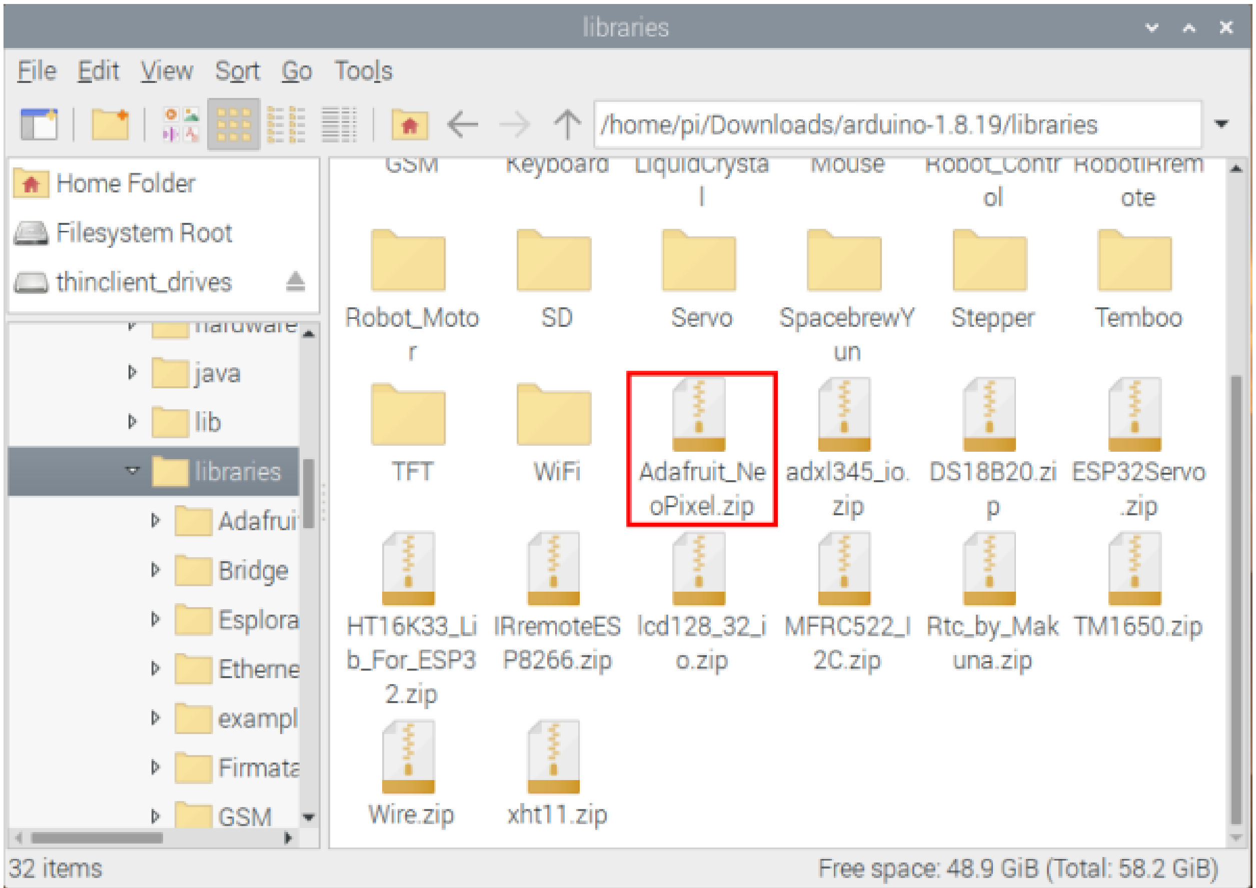


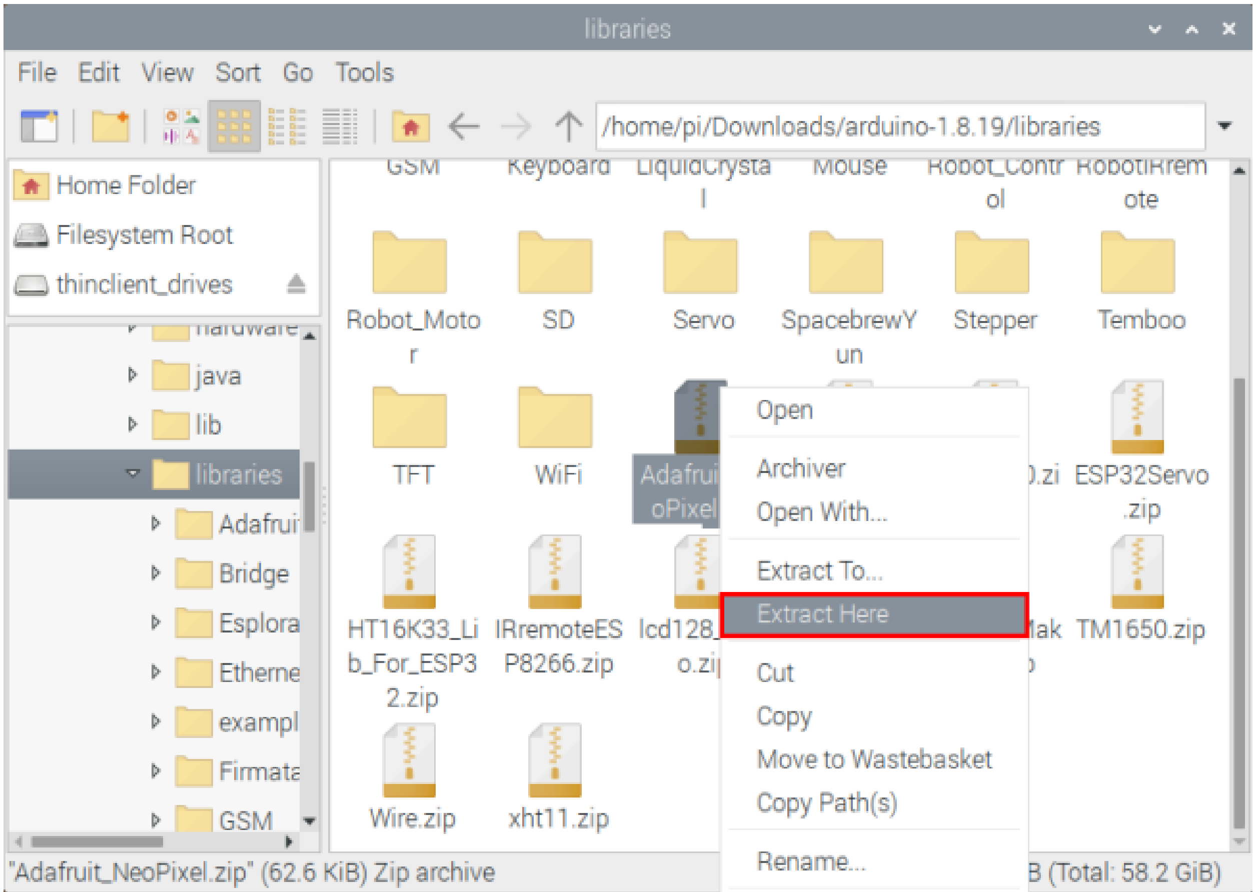


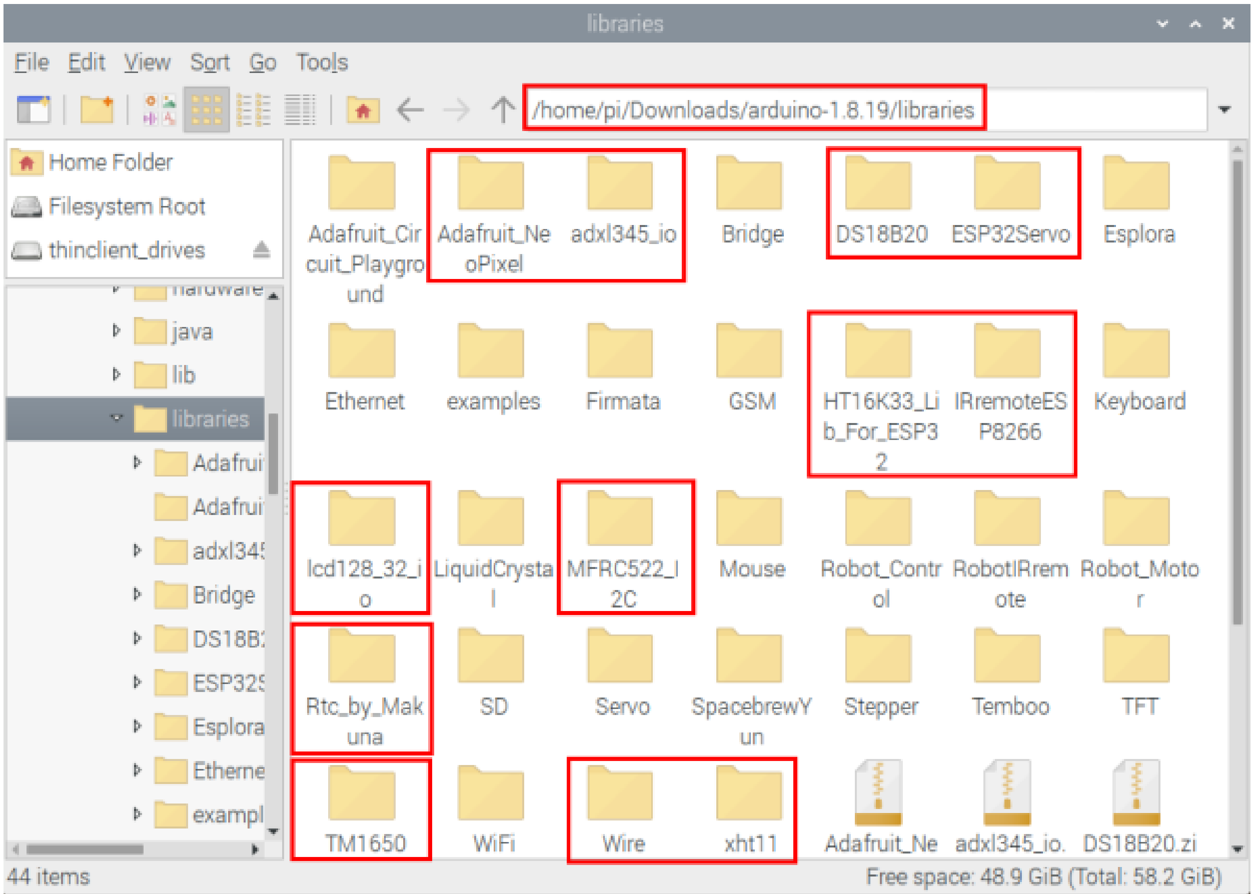
Step 3: Unzip the Arduino C package in the libraries folderfor exampleclick "Adafruit_NeoPixel.zip"file



select and tap"**Extract Here**"to unzip the "**Adafruit_NeoPixel.zip**"file.


libraries
File Edit View Sort Go Tools
/home/pi/Downloads/arduino-1.8.19/libraries
Home Folder
Filesystem Root
thinclient_drives
java
lib
libraries
Adafrui
Bridge
Esplora
Etherne
exampl
Firmata
GSM
Robot_Motor
SD
Servo
SpacebrewYun
Stepper
Temboo
TFT
WiFi
Adafruit_NeoPixel.zip
adxl345_io.zip
DS18B20.zip
ESP32Servo.zip
HT16K33_Lib_For_ESP32.zip
IRremoteESP8266.zip
lcd128_32_io.zip
MFRC522_I2C.zip
Rtc_by_Makuna.zip
TM1650.zip
Wire.zip
xht11.zip
32 items
Free space: 48.9 GiB (Total: 58.2 GiB)


libraries
File Edit View Sort Go Tools
/home/pi/Downloads/arduino-1.8.19/libraries
Home Folder
Filesystem Root
thinclient_drives
java
lib
libraries
Adafrui
Bridge
Esplora
Etherne
exampl
Firmata
GSM
Robot_Moto r
SD
Servo
SpacebrewY un
Stepper
Temboo
TFT
WiFi
ESP32Servo .zip
HT16K33_Li b_For_ESP3 2.zip
IRremoteES P8266.zip
TM1650.zip
Wire.zip
xht11.zip
Open
Archiver
Open With...
Extract To...
Extract Here
Cut
Copy
Move to Wastebasket
Copy Path(s)
Rename...
"Adafruit_NeoPixel.zip" (62.6 KiB) Zip archive



7.5 5. Basic Projects

When we get the kit, we can see that there are 42 sensors/modules in the kit, which contain the corresponding ESP32 mainboard, ESP32 Expansion Board and wirings. Here, we will connect the 42 sensors individually to the ESP32 mainboard and the ESP32 Expansion Board using wirings. Then run the corresponding test code to test the function of each sensor separately. Our next lesson is to study the principles of individual modules/sensors from simple to complex as well as some extended applications of sensors to consolidate and deepen our understanding of the kits.

Note: When connecting the module/sensor wirings in the projects, the wiring method and position must be followed in the document. What's more, do not misconnect the power supply and signal pin, otherwise there may be no experimental results or damage to the modules/sensors.

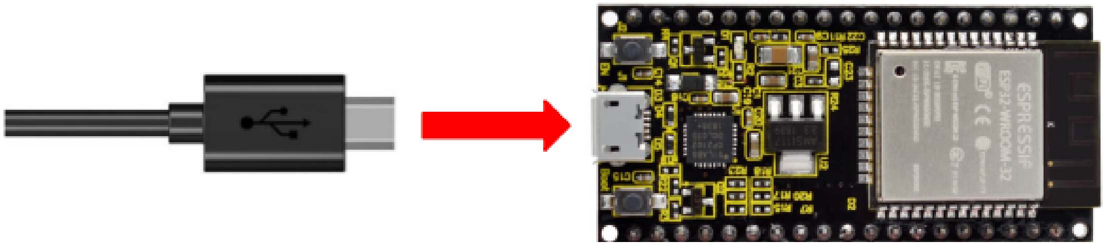
7.5.1 Project 1: Hello World

Overview

For ESP32 beginners, we will start with some simple things. In this project, you only need a ESP32 mainboard, a USB cable and Raspberry Pi to complete the "Hello World!" project, which is a test of communication between the ESP32 mainboard and the Raspberry Pi as well as a primary project.

Wiring Diagram

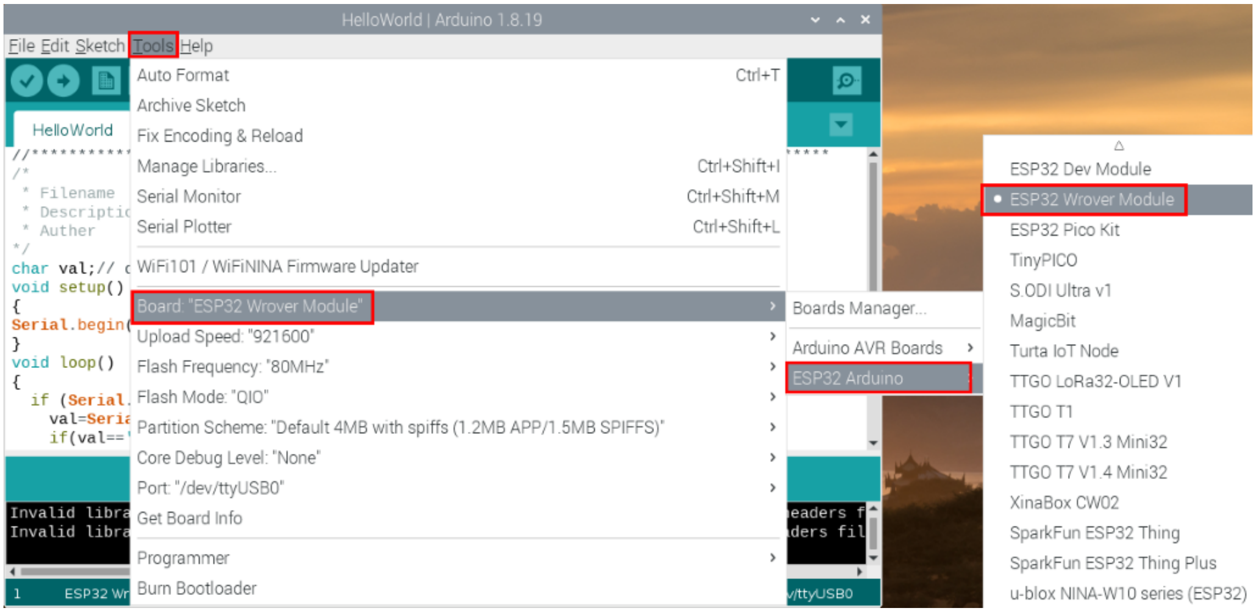
In this project, we will use a USB cable to connect the ESP32 to Raspberry Pi.



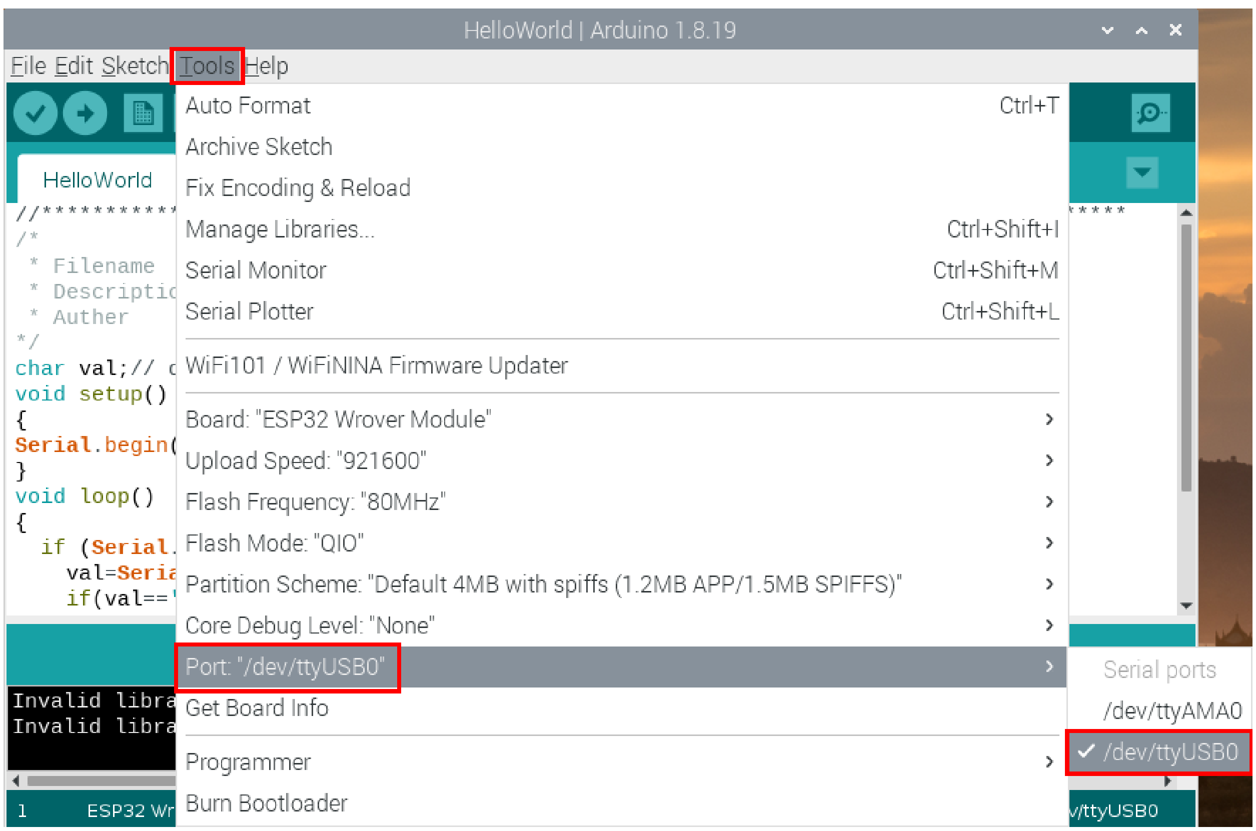
Test Code

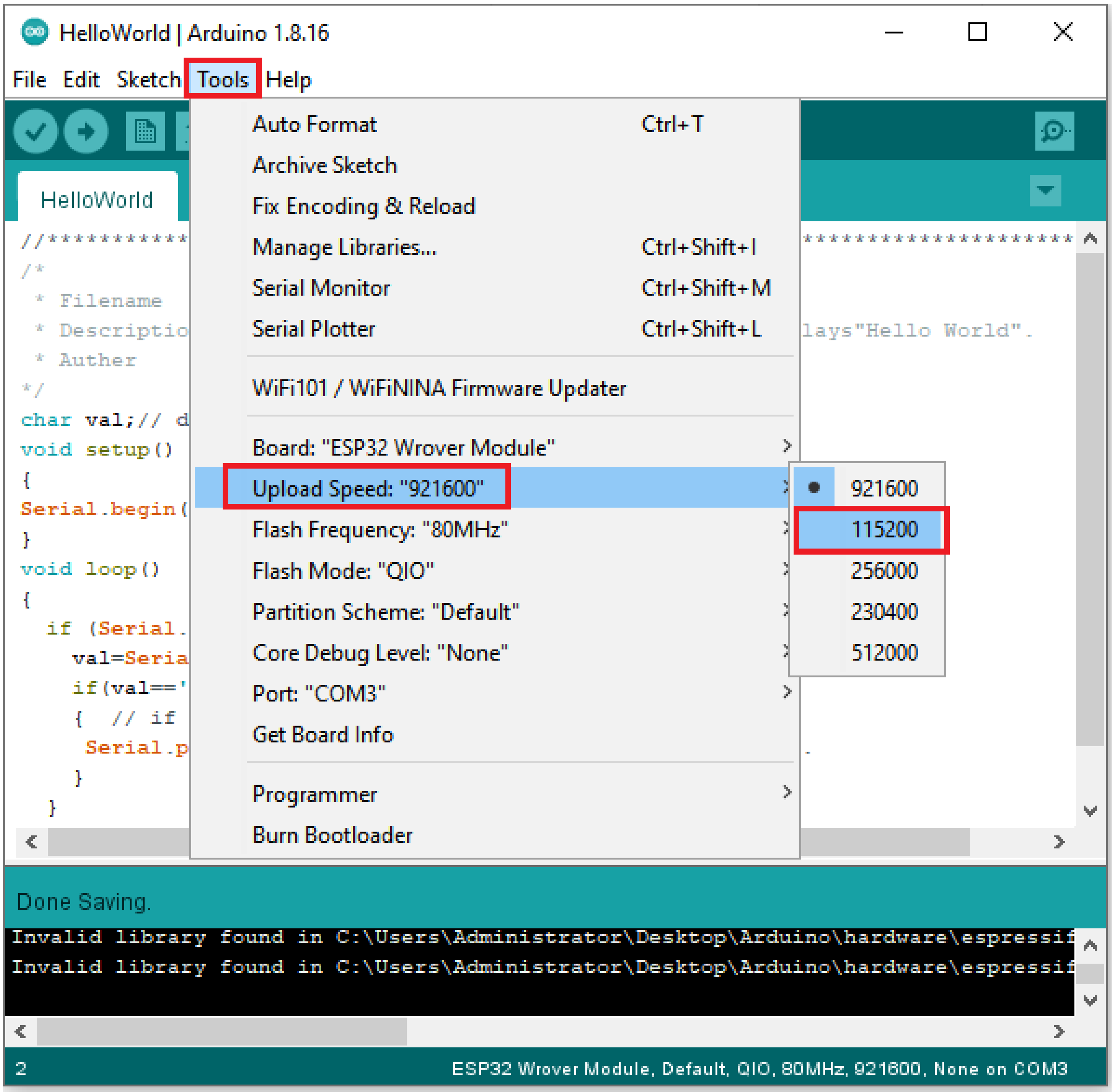
```
//*************************************************************************************
/*
 * Filename    : Hello World
 * Description : Enter the letter R,and the serial port displays"Hello World".
 * Auther      :http//www.keyestudio.com
*/
char val;// defines variable "val"
void setup()
{
Serial.begin(9600);// sets baudrate to 9600
}
void loop()
{
  if (Serial.available() > 0) {
    val=Serial.read();// reads symbols assigns to "val"
    if(val=='R')// checks input for the letter "R"
    {  // if so,
     Serial.println("Hello World!");// shows "Hello World !".
    }
  }
}
//*************************************************************************************
```

Before uploading the test code to the ESP32click “**Tools**” → “**Board**”select “**ESP32 Wrover Module**”.



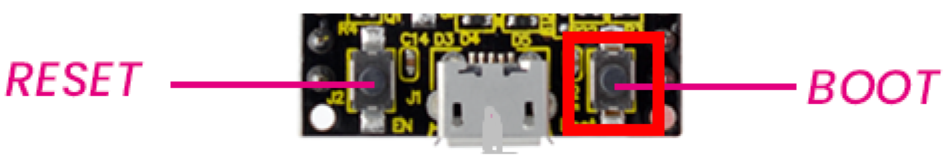
Select the correct serial port

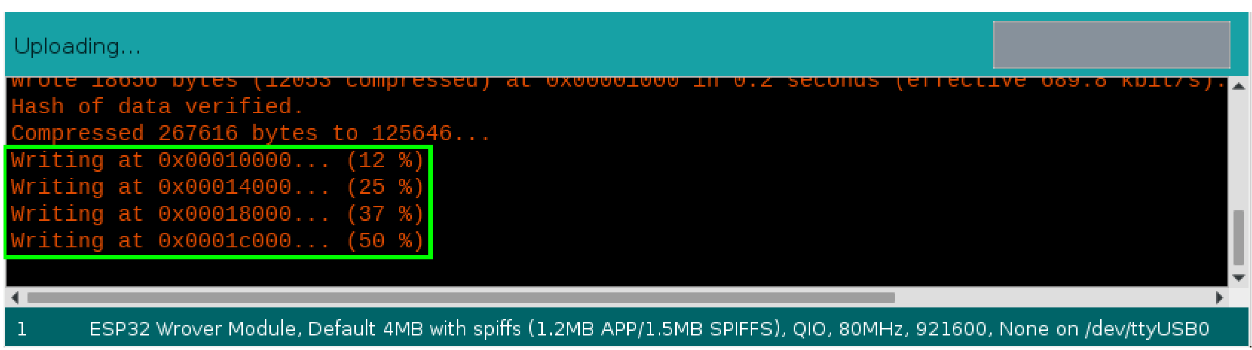




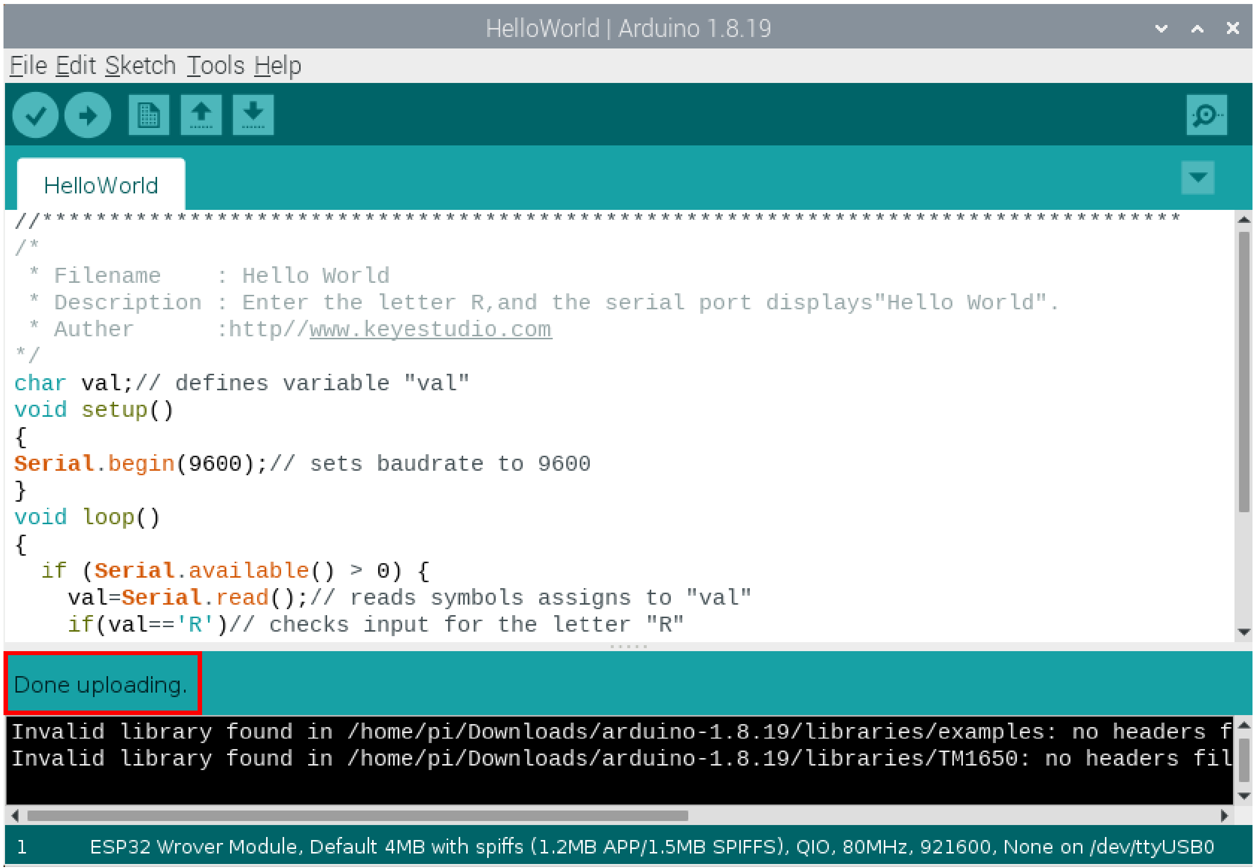
Click  to upload the test code to the ESP32.

Note: If the uploading code fails, you can press and hold the Boot button on the ESP32 after clicking  and release the Boot button after the percentage of uploading progress appears, as shown below:





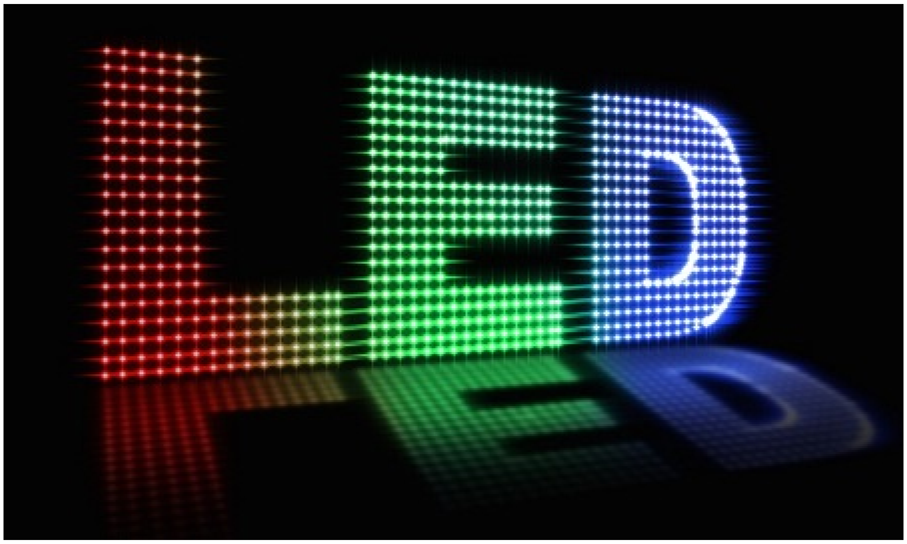
The code is uploaded successfully



Test Result

After uploading successfullywe will use a USB cable to power onclick, set the baud rate to 9600we need to press the reset button on the ESP32 motherboard and enter the letter “R”click “Send”then the serial monitor prints “Hello World!”.

7.5.2 Project 2: Lighting up LED



Overview

In this kit, we have a Keyestudio Purple Module, which is very simple to control. If you want to light up the LED, you just need to make a certain voltage across it.

In the project, we will control the high and low level of the signal end S through programming, so as to control the LED on and off.

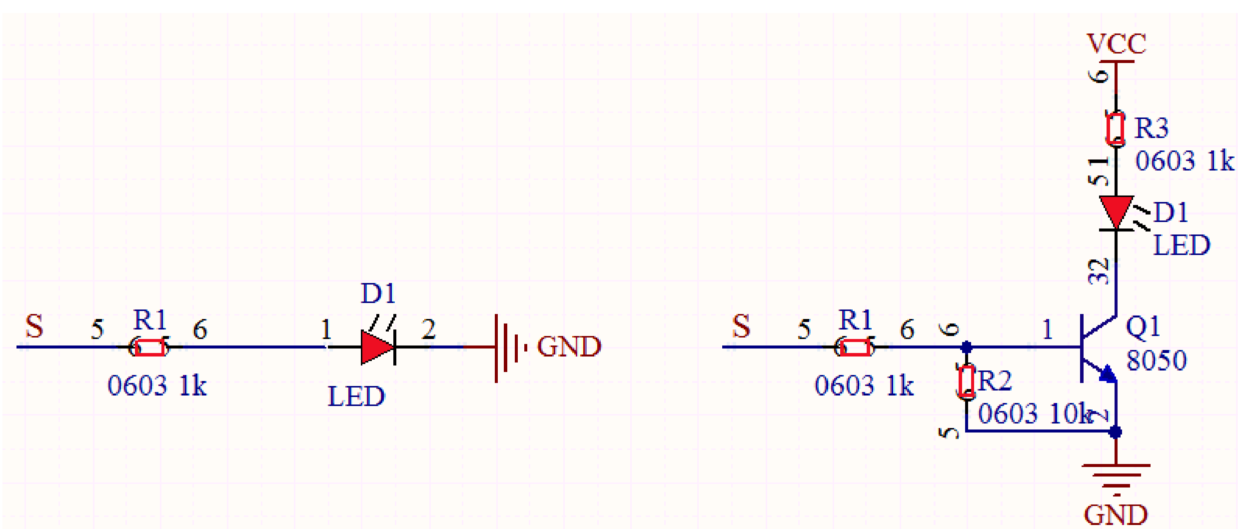
Working Principle

The two circuit diagrams are given.

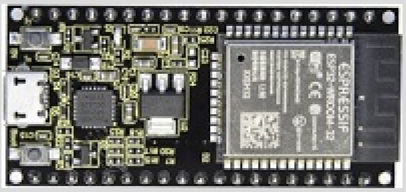
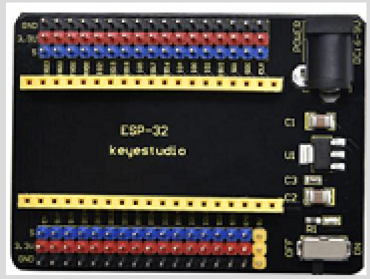
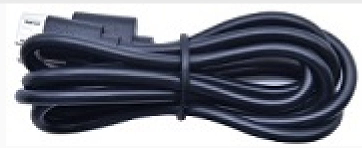
The left one is wrong wiring-up diagram. Why? Theoretically, when the S terminal outputs high levels, the LED will receive the voltage and light up.

Due to limitation of IO ports of ESP32 board, weak current can't make LED brighten.

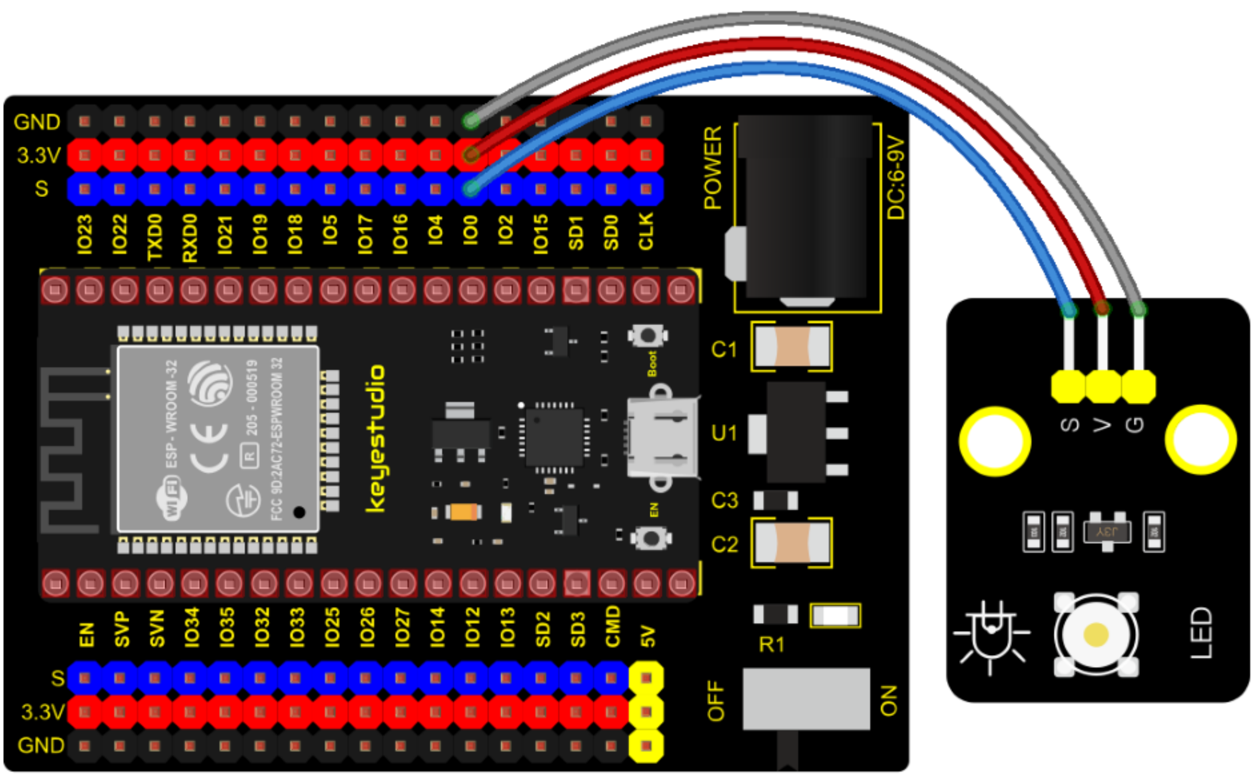
The right one is correct wiring-up diagram. GND and VCC are powered up. When the S terminal is a high level, the triode Q1 will be connected and LED will light up(note: current passes through LED and R3 to reach GND by VCC not IO ports). Conversely, when the S terminal is a low level, the triode Q1 will be disconnected and LED will go off.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Blink
```

(continues on next page)

(continued from previous page)

```
 * Description : led Flashing 1 s
 * Auther      : http://www.keyestudio.com
*/
int ledPin = 0; //Define LED pin connection to GPIO0
void setup() {
  pinMode(ledPin, OUTPUT);//Set mode to output
}

void loop() {
  digitalWrite(ledPin, HIGH); //Output high level, turn on led
  delay(1000);//Delay 1000 ms
  digitalWrite(ledPin, LOW); //Output low level,turn off led
  delay(1000);//Delay 1000 ms
}
//*************************************************************************************
```

Code Explanation

1). PinMode(pin,mode): Pin is the ESP32 GPIO pin number used to set the mode, here we set pin 0 as output mode.

2). DigitalWrite(pin, value): Pin is the GPIO pin, which is defined GP0 here. Valueis the digital level that we will outputHIGH/LOW. If the pin is configured to OUTPUT using pinMode(), its voltage is set to the corresponding value: 3.3V is HIGH,low level is 0V (ground). When connect the LEDs to the pins, using the digitalWriteHIGH, the LEDs will get dim.

3). Setup() executes once, while loop() executes all the time. Delay (ms) is delay function, ms is the number of milliseconds to pause. Data type: unsigned longrange 0~ 4,294,967,295 (2^32 - 1).

4). Firstly, we connect the module signal to ledPIN, namely GP0, and set it to a high level to light the LEDs on the module. Then delay 1000 ms, controlling the LEDs on the module light up for 1s and off for 1s to achieve the flashing effect.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the LED in the circuit will flash alternately.

7.5.3 Project 3: Traffic Lights Module

Overview

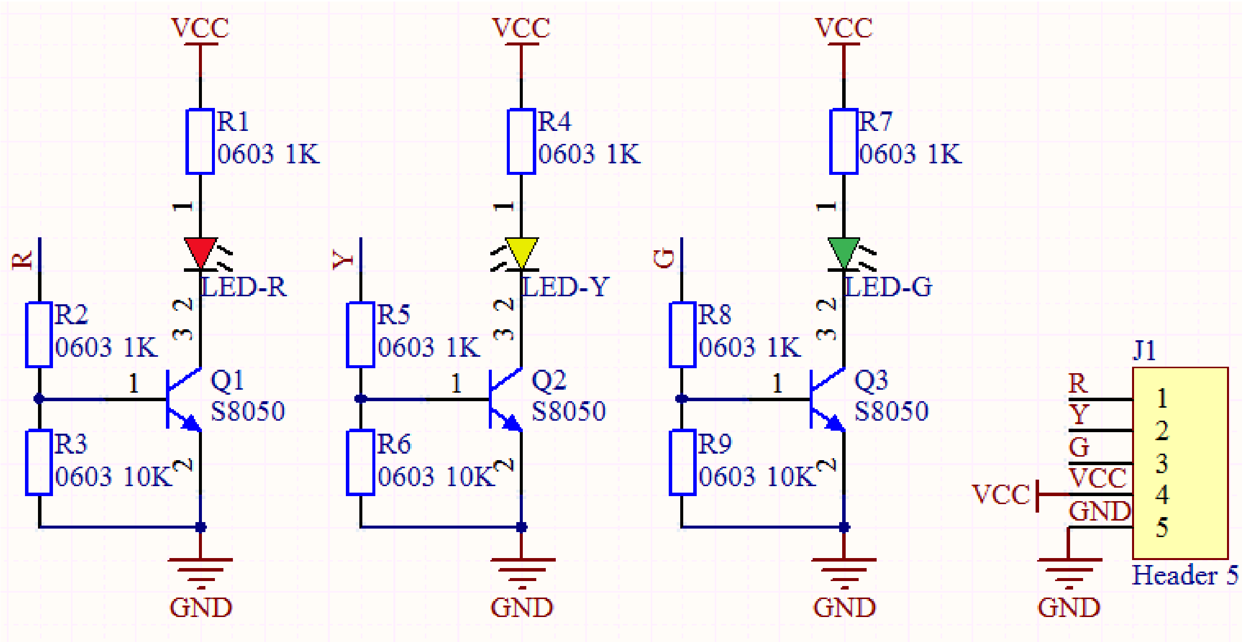
In this lesson, we will learn how to control multiple LED lights and simulate the operation of traffic lights.

Traffic lights are signal devices positioned at road intersections, pedestrian crossings, and other locations to control flows of traffic.

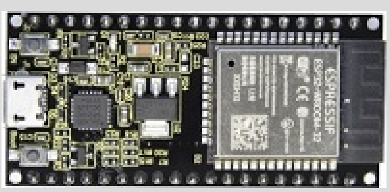
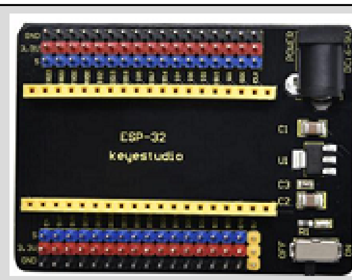
In this kit, we will use the traffic light module to simulate the traffic light.

Working Principle

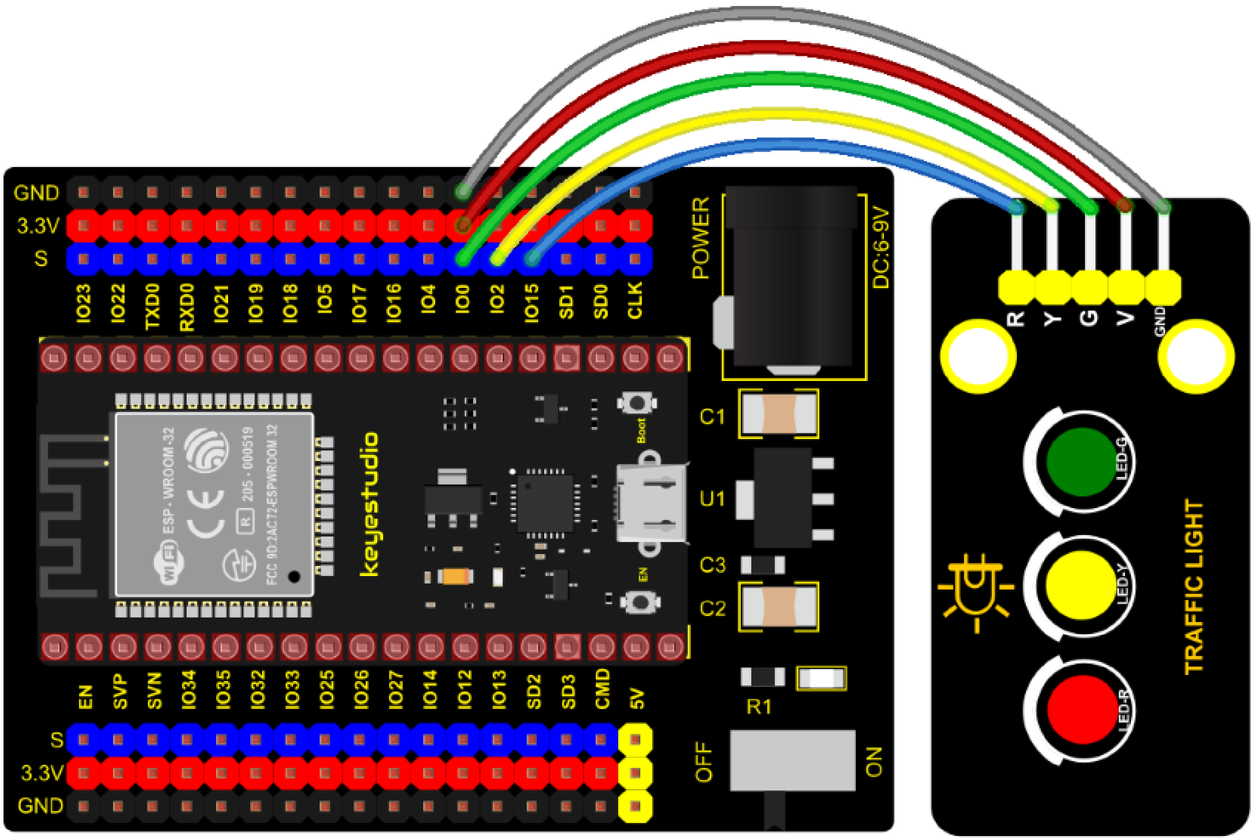
In previous lesson, we already know how to control an LED. In this part, we only need to control three separated LEDs. Input high levels to the signal R(3.3V), then the red LED will be on.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Traffic Lights Module*1
		
5P Dupont Wire*1	Micro USB Cable*1	

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Traffic_Light
 * Description : Simulated traffic lights
 * Auther      : http://www.keyestudio.com
*/
int redPin = 15;   //Red LED connected to GPIO15
int yellowPin = 2; //Yellow LED connected to GPIO2
int greenPin = 0;   //Green LED connected to GPIO0

void setup() {
  //LED interfaces are set to output mode
  pinMode(greenPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(redPin, OUTPUT);
}

void loop() {
  digitalWrite(greenPin, HIGH); //Lighting green LED
  delay(5000);  //Delay for 5 seconds
  digitalWrite(greenPin, LOW); //Turn off green LEDS
  for (int i = 1; i <= 3; i = i + 1) {  //run three times
    digitalWrite(yellowPin, HIGH); //Lighting yellow LED
```

(continues on next page)

(continued from previous page)

```
    delay(500); //Delay for 0.5 seconds
    digitalWrite(yellowPin, LOW); //Turn off yellow LED
    delay(500); //Delay for 0.5 seconds
  }
  digitalWrite(redPin, HIGH); //Lighting red LED
  delay(5000);  //Delay5s
  digitalWrite(redPin, LOW); //Turn off red LED
}
//*************************************************************************************
```

Code Explanation

Create pins, set pins mode and delayed functions.

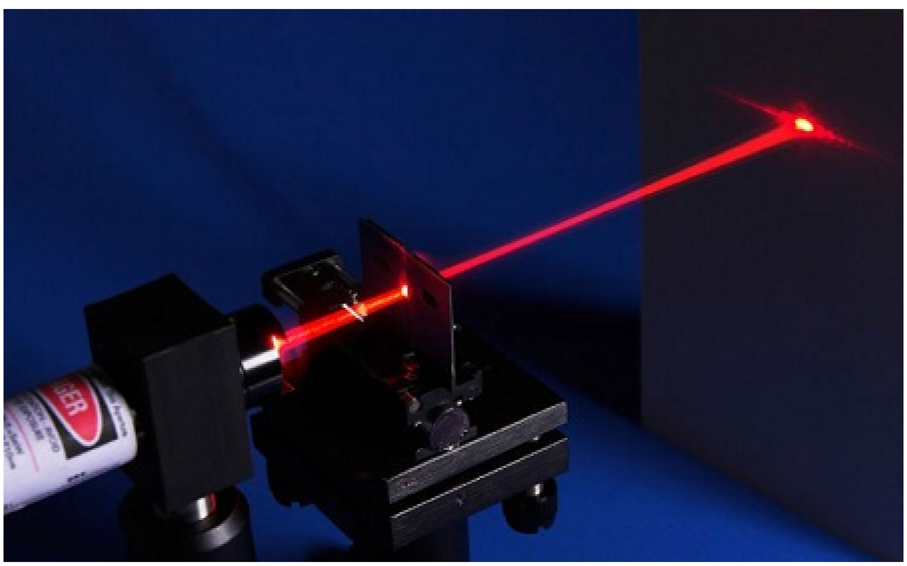
We use the function for(). for (int i = 1; i <= 3; i = i + 1) represents the variable i adds 1 fir each time from 1 to 3.

The function for (int i = 255; i >= 0; i = i - 1) indicates that i reduces by 1 each time. When i<0, exit the for() loop and execute 256 times.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the green LED will be on for 5s then off, the yellow LED will flash for 3s then go off and the red one will be on for 5s then off, the three LED modules will simulate the circulation of traffic lights automatically .

7.5.4 Project 4: Laser Sensor

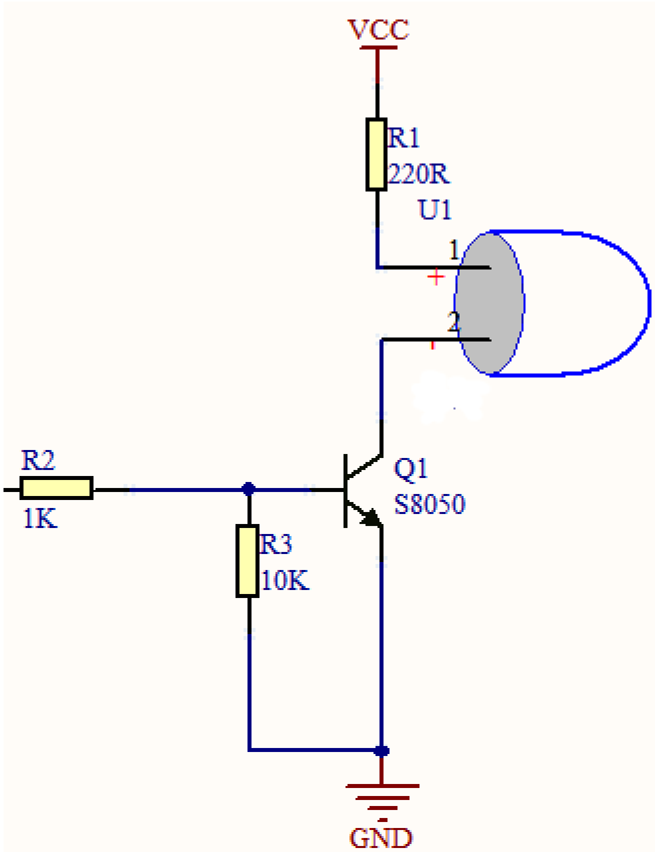


Description

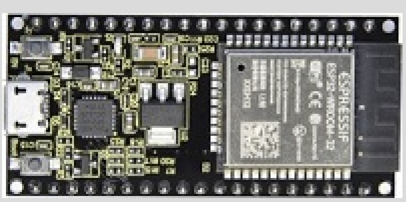
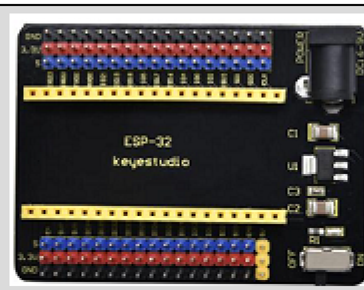
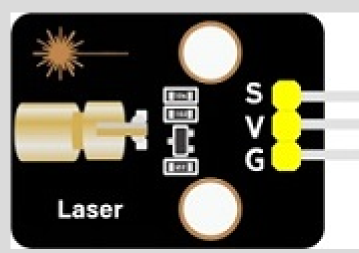
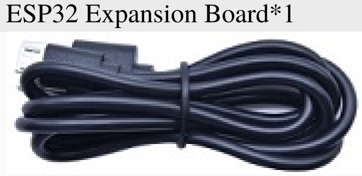
Lasers are widely used to cut, weld, surface treat, and more on specific materials. The energy of the laser is very high. The toy laser pointer may cause glare to the human eye, and it may cause retinal damage for a long time. my country also prohibits the use of laser to illuminate the aircraft.

Working Principle

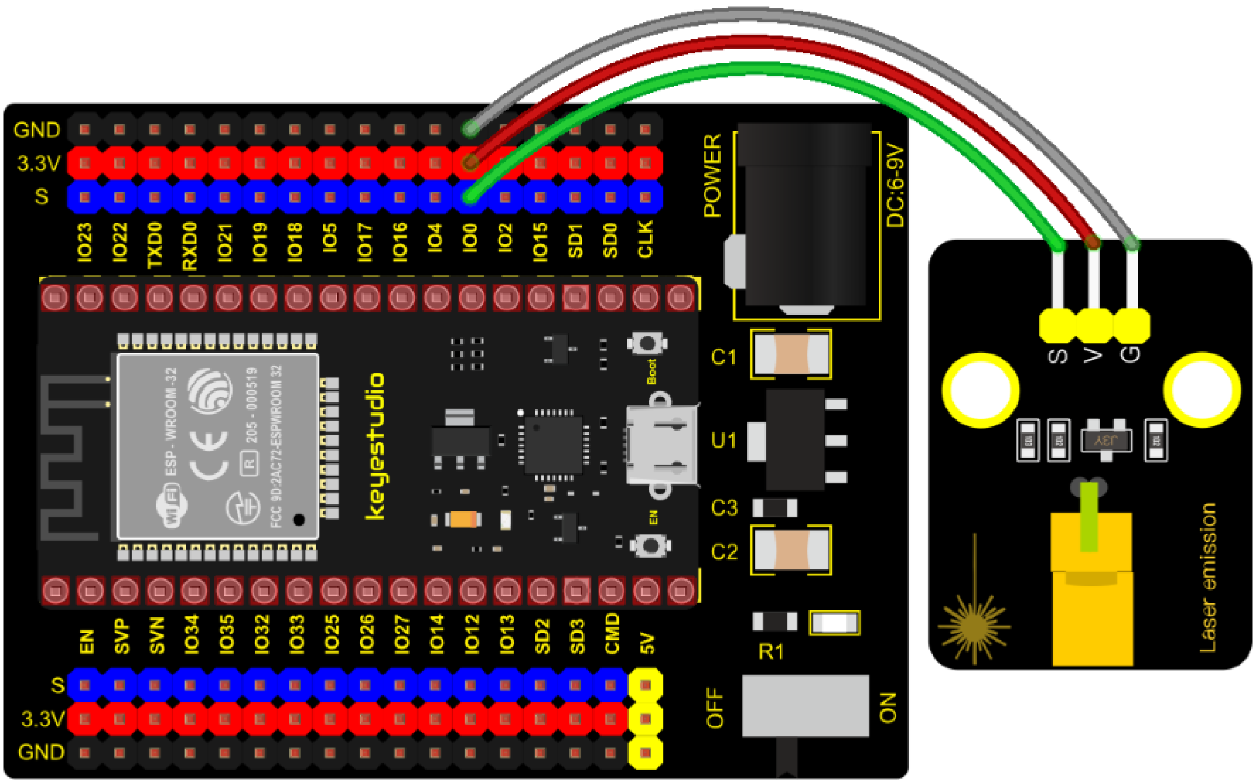
The laser head sensor module is mainly composed of a laser head with a light-emitting die, a condenser lens, and a copper adjustable sleeve. We can see the circuit schematic diagram of this module which is very similar to the LED we have learned. They are all driven by triodes. A high-level digital signal is directly input at the signal end, then the sensor will start to work; if inputting low levels, the sensor won't work.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Laser Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Laser sensor
 * Description : Laser light flashing
 * Auther      : http://www.keyestudio.com
*/
int laserPin = 0; //Define the laser pin as GPIO 0
void setup() {
  pinMode(laserPin, OUTPUT);//Define laser pin as output mode
}

void loop() {
  digitalWrite(laserPin, HIGH); //Open the laser
  delay(2000);  //Delay 2 seconds
  digitalWrite(laserPin, LOW); //Shut down the laser
  delay(2000);  //Delay 2 seconds
}
//**********************************************************************************
```

Test Result

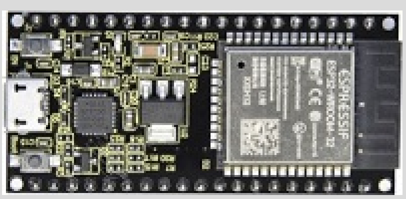
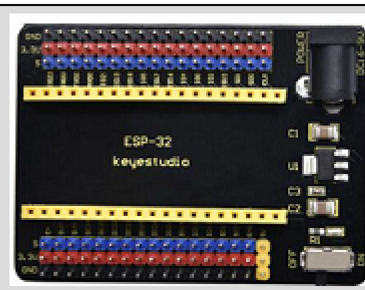
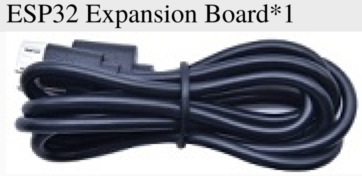
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the laser module will emit red laser signals for 2 seconds and stop emitting signals for 2 seconds on a cycle.

7.5.5 Project 5: Breathing LED

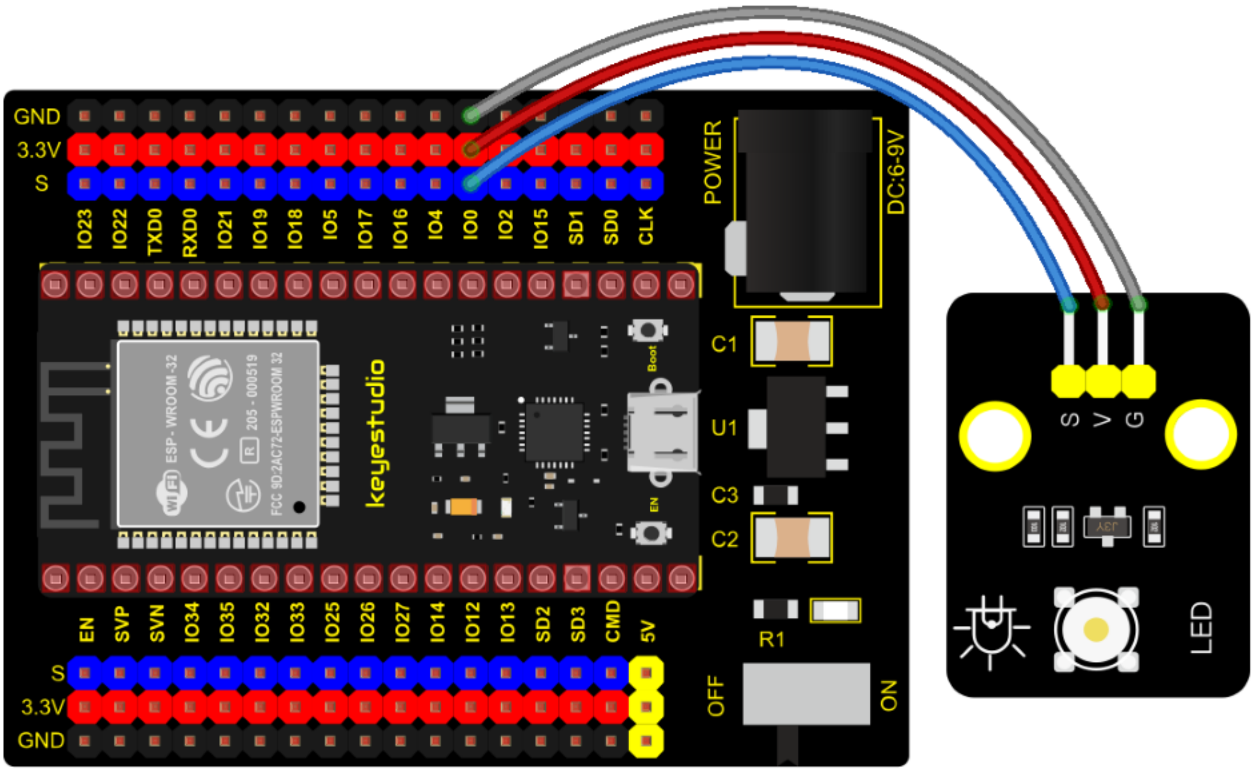
Overview

A“breathing LED”is a phenomenon where an LED’s brightness smoothly changes from dark to bright and back to dark, continuing to do so and giving the illusion of an LED“breathing. This phenomenon is similar to a lung breathing in and out. So how to control LED’s brightness? We need to take advantage of PWMyou can refer to experiment six.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1
		
3P Dupont Wire*1	MicroUSB Cable*1	

Connection Diagram



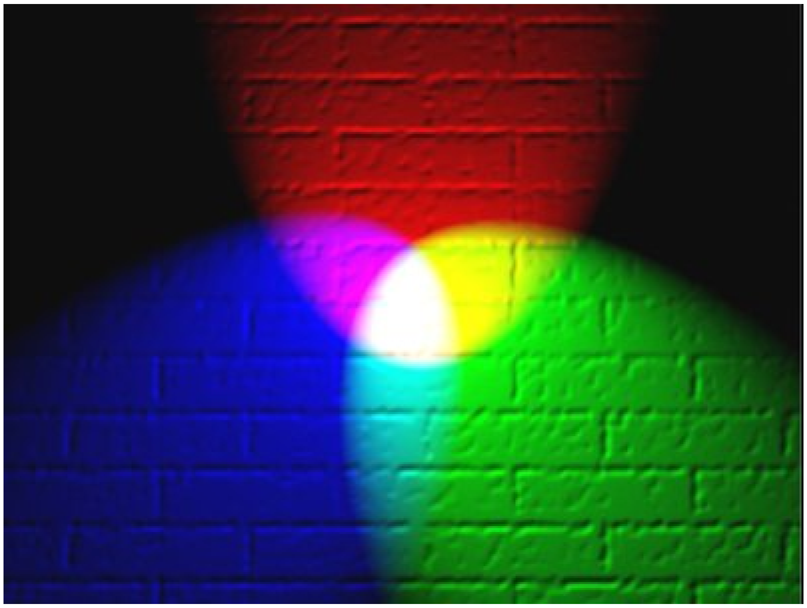
Test Code

```
//**********************************************************************
/*
 * Filename    : Breathing Led
 * Description : Make led light fade in and out, just like breathing.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_LED   0   //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() {
  ledcSetup(CHN, FRQ, PWM_BIT); //setup pwm channel
  ledcAttachPin(PIN_LED, CHN);  //attach the led pin to pwm channel
}

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

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the LED on the module gradually gets dimmer then brighter, cyclically, like human breathe.

7.5.6 Project 6: RGB Module



Overview

Among these modules is a RGB module. It adopts a F10-full color RGB foggy common cathode LED. We connect the RGB module to the PWM port of MCU and the other pin to GND(for common anode RGB, the rest pin will be connected to VCC). So what is PWM?

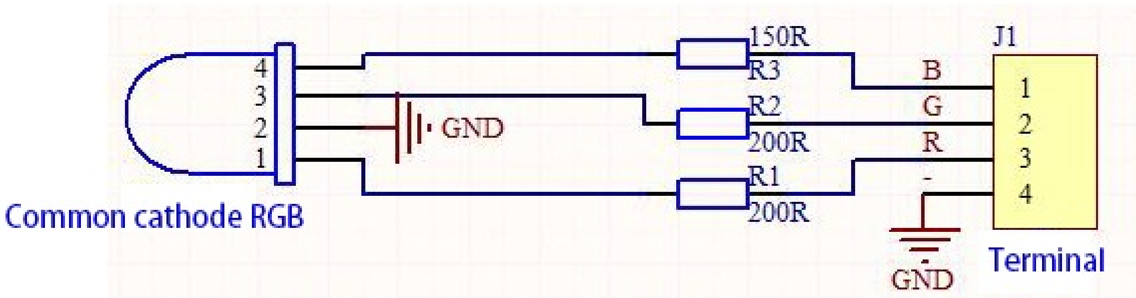
PWM is a means of controlling the analog output via digital means. Digital control is used to generate square waves with different duty cycles (a signal that constantly switches between high and low levels) to control the analog output. In general, the input voltages of ports are 0V and 5V. What if the 3V is required? Or a switch among 1V, 3V and 3.5V? We cannot change resistors constantly. For this reason, we resort to PWM.

For Arduino digital port voltage outputs, there are only LOW and HIGH levels, which correspond to the voltage outputs of 0V and 5V respectively. You can define LOW as "0" and HIGH as "1", and let the Arduino output five hundred "0" or "1" within 1 second. If output five hundred "1", that is 5V; if all of which is "0",that is 0V; if output 250 01 pattern, that is 2.5V.

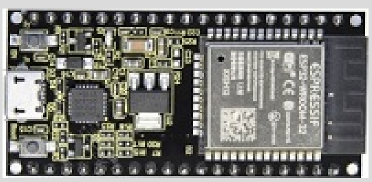
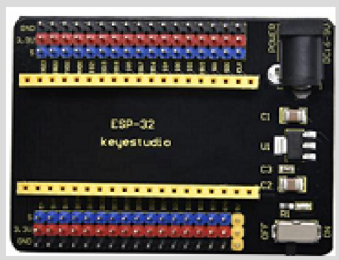
This process can be likened to showing a movie. The movie we watch are not completely continuous. Actually, it generates 25 pictures per second, which cannot be told by human eyes. Therefore, we mistake it as a continuous process. PWM works in the same way. To output different voltages, we need to control the ratio of 0 and 1. The more'0'or'1' output per unit time, the more accurate the control.

Working Principle

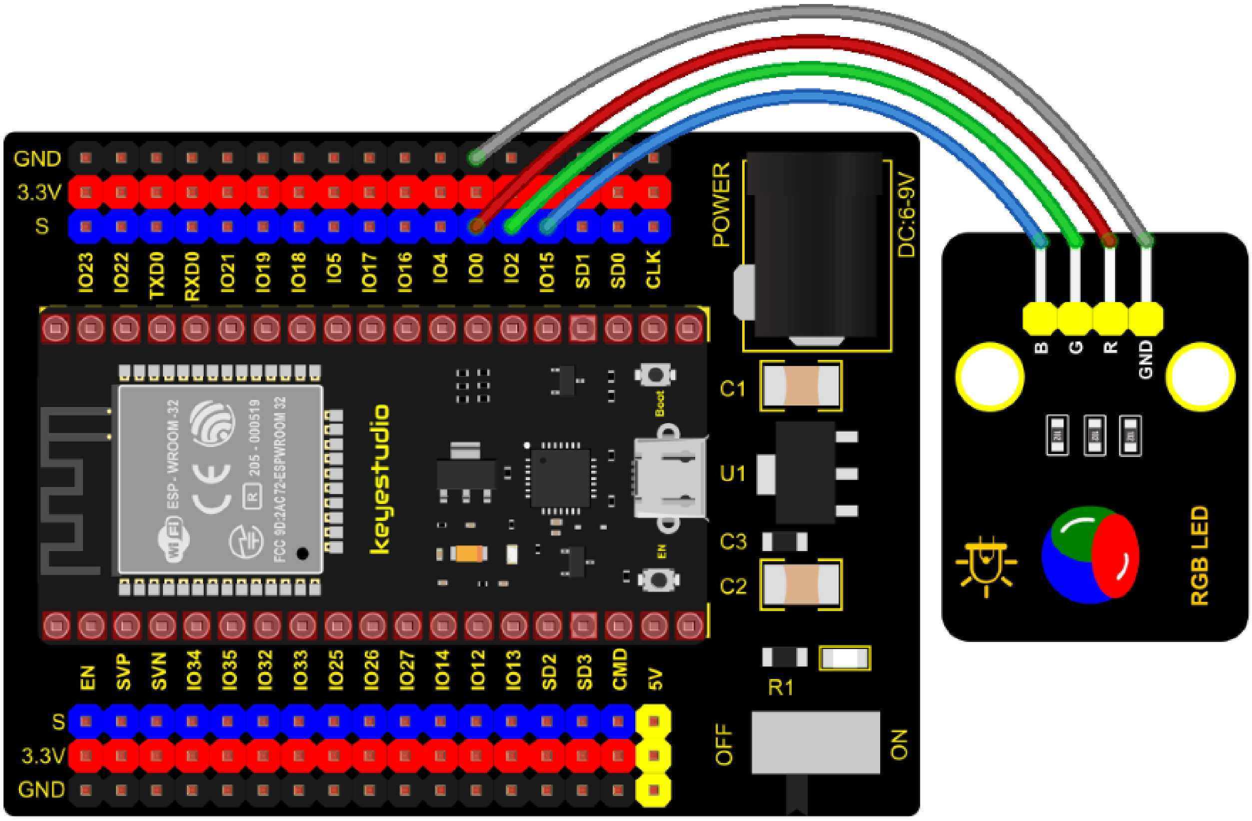
For our experiment, we will control the RGB module to display different colors through three PWM values.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Common Cathode RGB Module *1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : RGB LED
 * Description : Use RGBLED to show random color.
 * Auther      : http//www.keyestudio.com
*/
int ledPins[] = {0, 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
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], 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) {
```

(continues on next page)

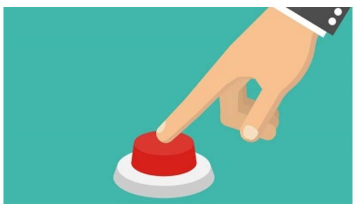
(continued from previous page)

```
  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);
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onwe will see that the RGB LED on the module starts to display random colors.

7.5.7 Project 7: Button Sensor



Overview

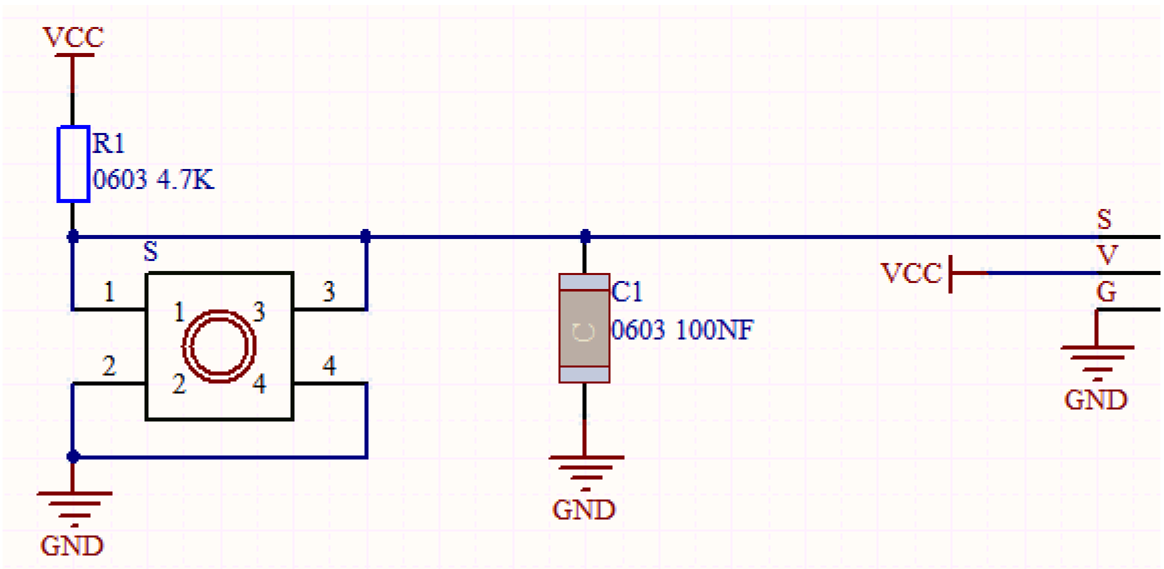
In this kit, there is a Keyestudio single-channel button module, which mainly uses a tact switch and comes with a yellow button cap.

In previous lessons, we learned how to make the pins of our single-chip microcomputer output a high level or low level. In this experiment, we will read the high level (3.3V) and low level (0V).

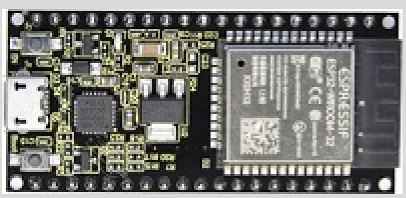
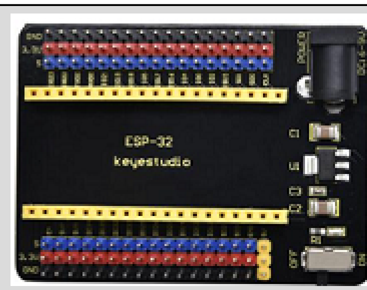
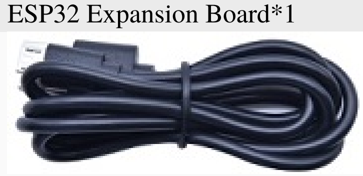
We can determine whether the button on the sensor is pressed by reading the high and low level of the S terminal on the sensor.

Working Principle

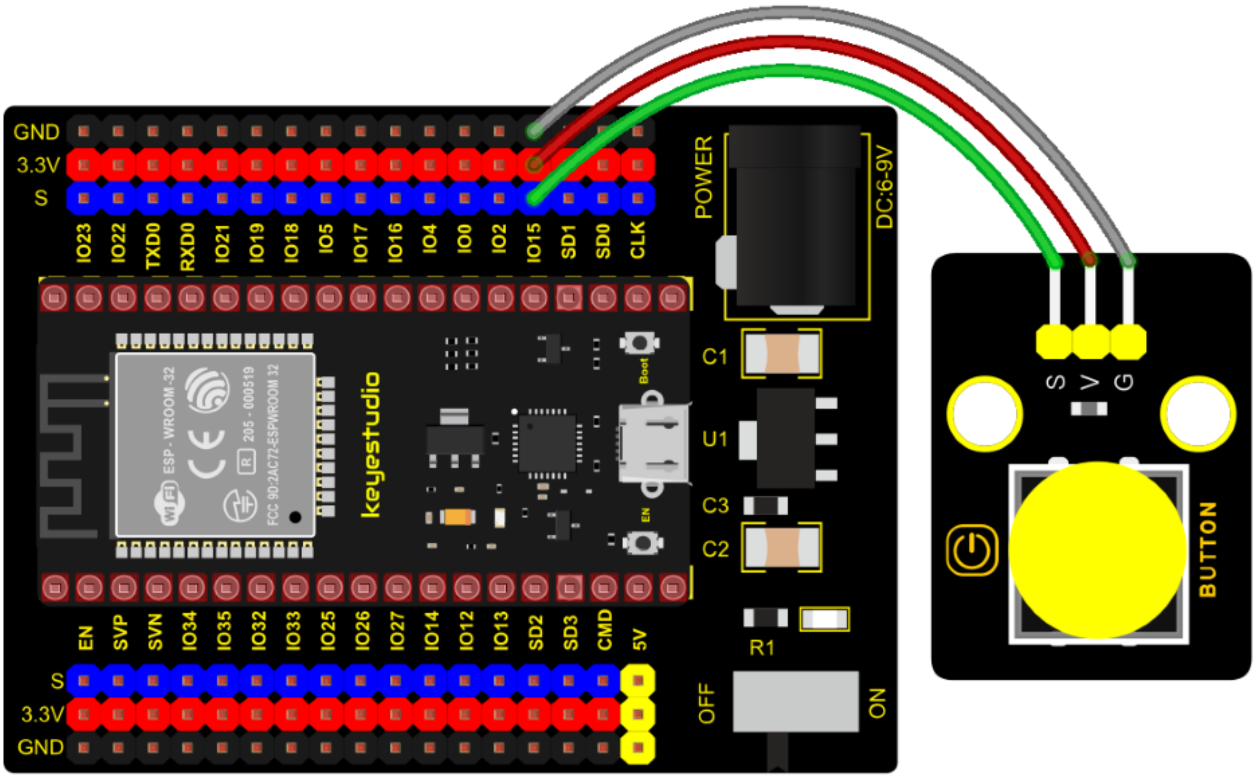
The button module has four pins. The pin 1 is connected to the pin 3 and the pin 2 is linked with the pin 4. When the button is not pressed, they are disconnected. Yet, when the button is pressed, they are connected. If the button is released, the signal end is high level.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Button Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : button
 * Description : Read key value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;  //Useto store key values
int button = 15; //The pin of the button is connected to GP15
void setup() {
  Serial.begin(9600); //Start the serial port monitor and set baud rate to 9600
  pinMode(button, INPUT); //Set key pin to input mode
}

void loop() {
  val = digitalRead(button);  //Read the value of the key and assign it to the variable val
  Serial.print(val);  //Print it on the serial port
  if (val == 0) { //Press the key to read the low level and print the press related information
    Serial.print("        ");
    Serial.println("Press the botton");
    delay(100);
  }

  else {  //Print information about key release
    Serial.print("        ");
```

(continues on next page)

(continued from previous page)

```
    Serial.println("Loosen the botton");
    delay(100);
  }
}
//*********************************************************************************
```

Code Explanation

1). **pinMode(button, INPUT)**; set the pin of the button module to GP15 and INPUT.

Configure INPUT through pinMode(). INPUT must use the pull-up or pull-down resistor(ours module has the pull-up resistor R1).

2). **Serial.begin(9600)**: Initialize serial communication and set the baud rate to 9600.

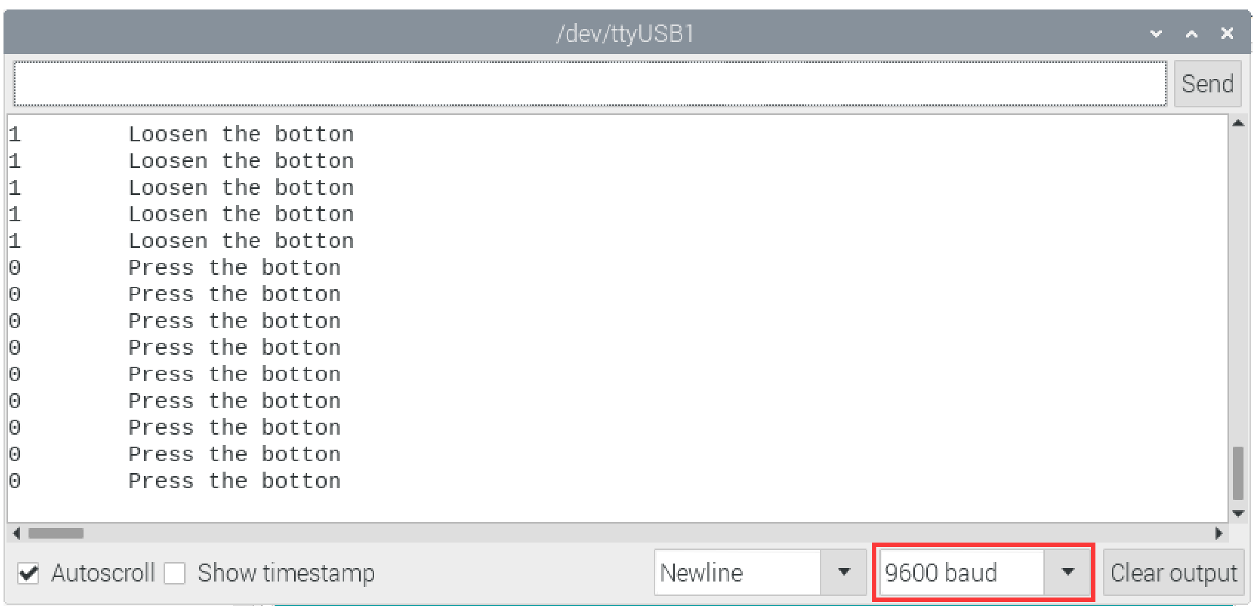
3). **digitalRead(button)**: read the digital level of the button(HIGH or LOW). If this pin is not connected to pins, the digitalRead() will return HIGH or LOW.

4). **if…else…**if the logic behind () is true, execute the code of (); otherwise execute the code of **else**.

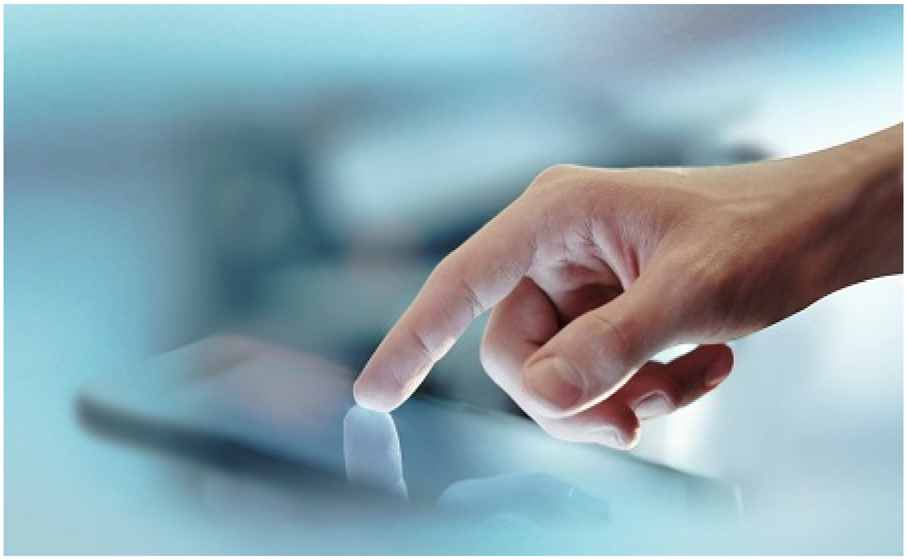
5). If the button is pressed, the signal end is low level, GP15 is low level and Val is 0. Then the monitor will show the corresponding value and characters; otherwise, the sensor is released, val is 1 and monitor will show 1 and other characters

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters. When the button is pressed, val is 0, the monitor will show "Press the button"when the button is released, val is 1the monitor will show "Loosen the button"; as shown below



7.5.8 Project 8: Capacitive Sensor

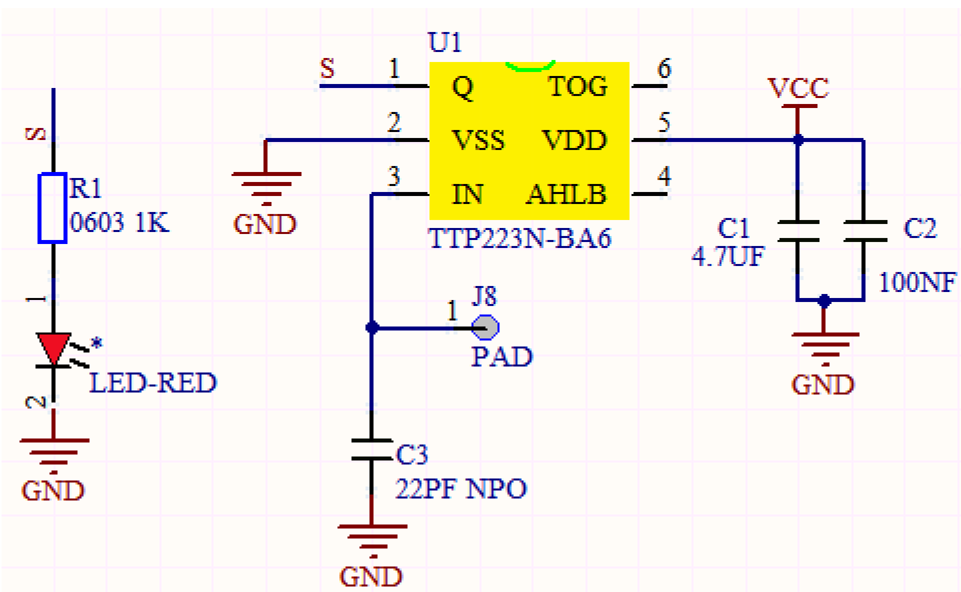


Description

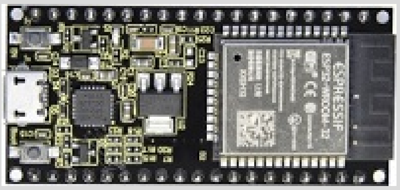
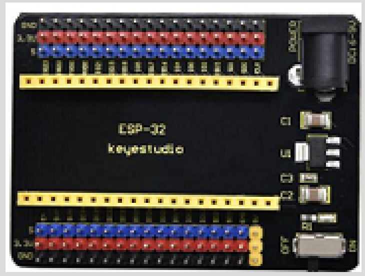
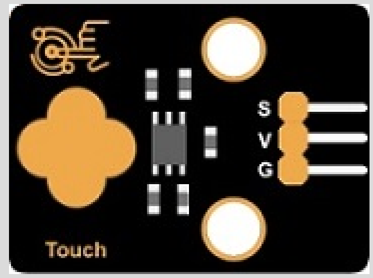
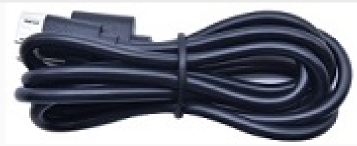
In this kit, there is a capacitive touch module which mainly uses a TTP223-BA6 chip. It is a touch detection chip, which provides a touch button, and its function is to replace the traditional button with a variable area button. When we power on, the sensor needs about 0.5 seconds to stabilize. Do not touch the keys during this time period. At this time, all functions are disabled, and self-calibration is always performed. The calibration period is about 4 seconds. We display the test results in the shell.

Working Principle

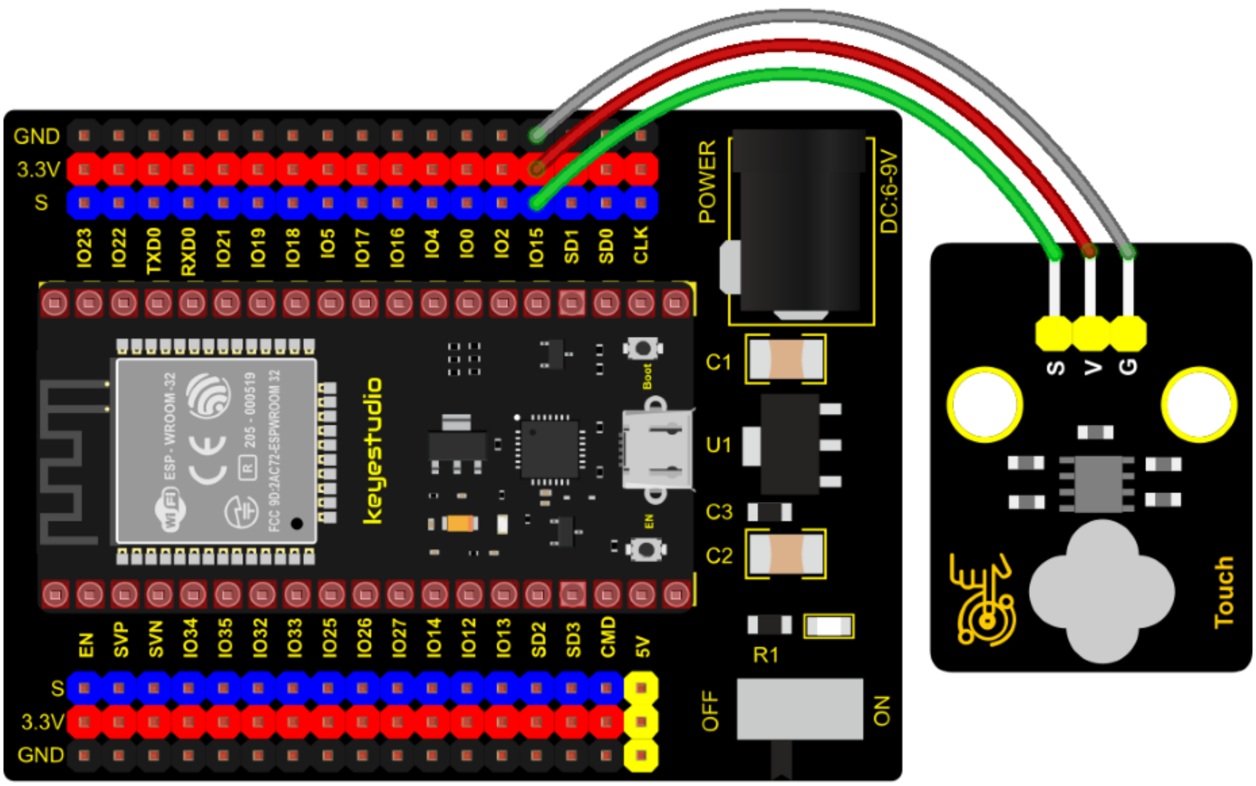
When our fingers touch the module, the signal S outputs high levels, the red LED on the module flashes. We can determine if the button is pressed or not by reading high and low levels on the sensor.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Capacitive Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Touch sensor
 * Description : Reading touch value
```

(continues on next page)

(continued from previous page)

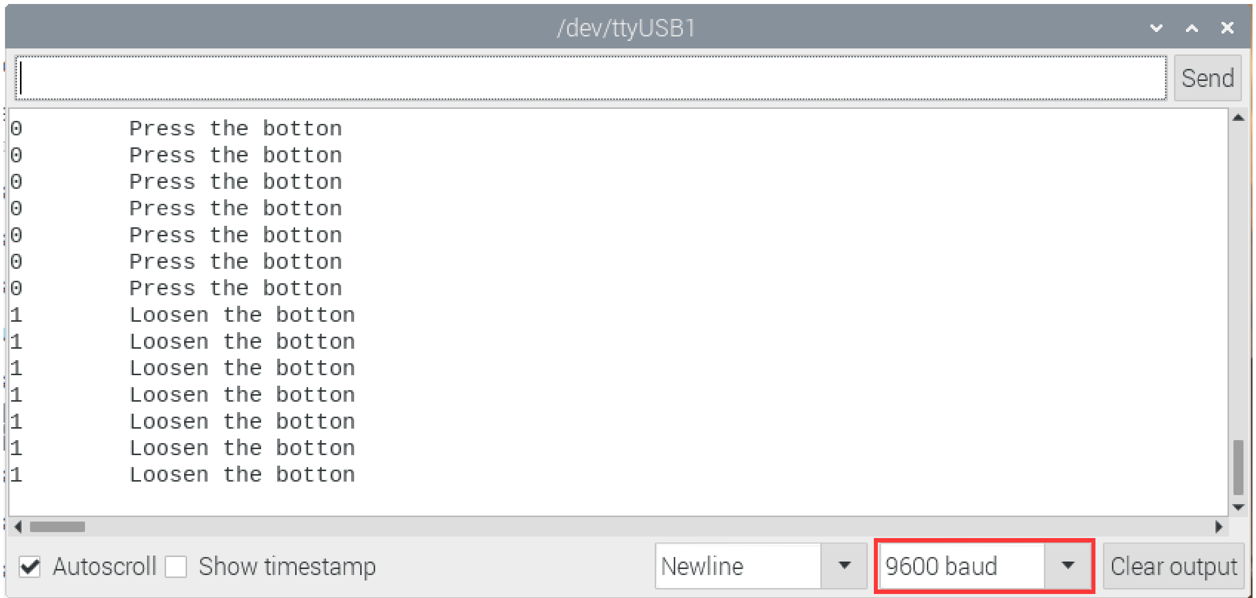
```
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int touch = 15; //The key of PIN
void setup() {
  Serial.begin(9600);//Baud rate is 9600
  pinMode(touch, INPUT);//Setting input mode
}

void loop() {
  val = digitalRead(touch);//Read the value of the key
  Serial.print(val);//Print out key values
  if (val == 1) {//Press for high level
    Serial.print("        ");
    Serial.println("Press the button");
    delay(100);
  }
  else {//Release to low level
    Serial.print("        ");
    Serial.println("Loosen the button");
    delay(100);
  }
}
//**************************************************************************************
```

Code Explanation

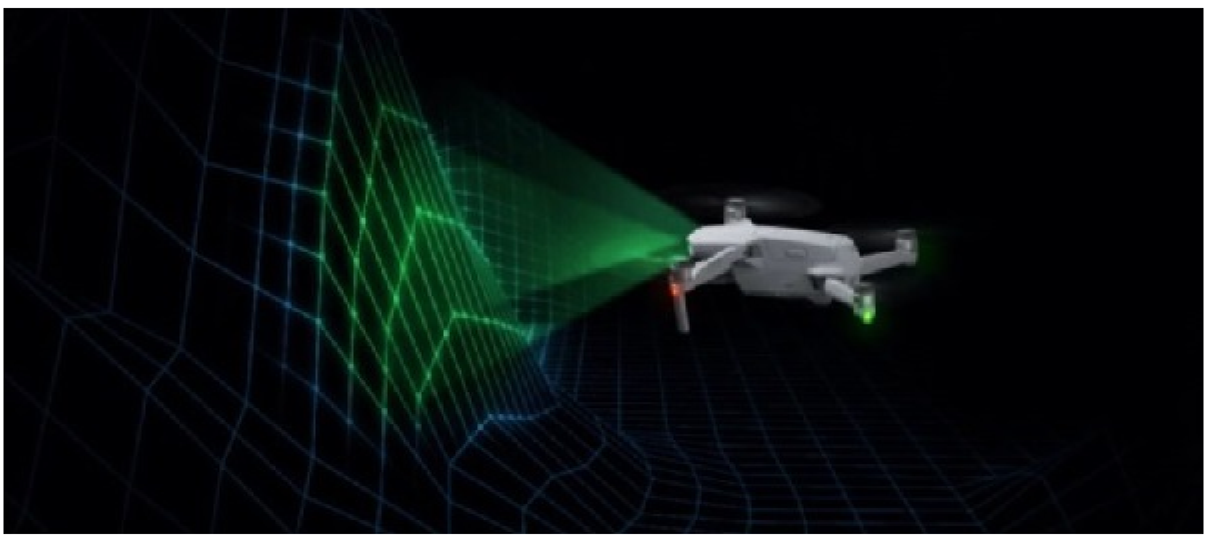
When we touch the sensor, the Shell monitor will show "Pressed the button!", if not, "Loosen the button!" will be shown on the monitor.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters. when the button is pressed, the red LED lights up and val is 1. Then the shell shows "Pressed the button!"; if the button is released, the red LED is off and val is 0, "Loosen the button!" will be displayed.



7.5.9 Project 9: Obstacle Avoidance Sensor



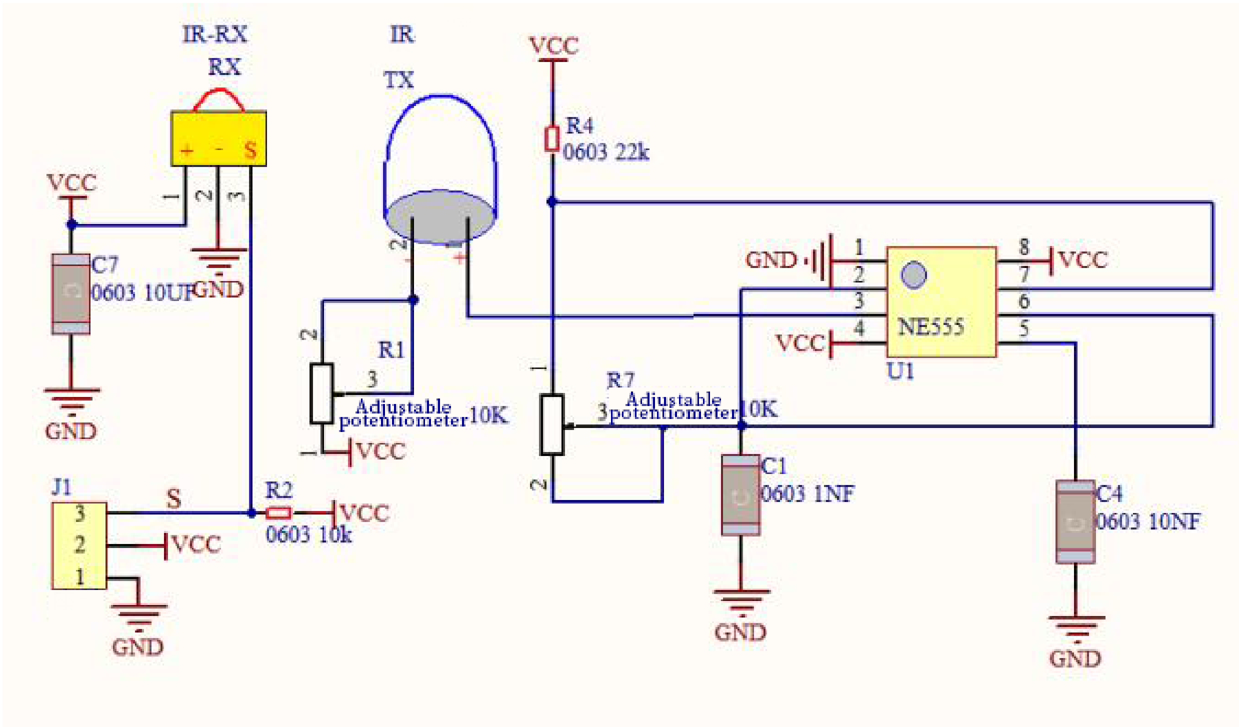
Overview

In this kit, there is a Keyestudio obstacle avoidance sensor, which mainly uses an infrared emitting and a receiving tube. In the experiment, we will determine whether there is an obstacle by reading the high and low level of the S terminal on the sensor.

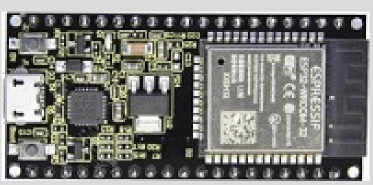
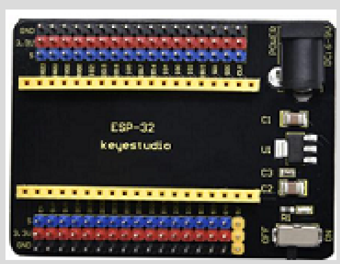
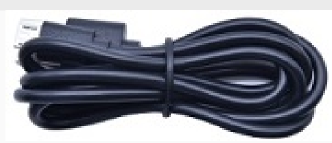
Working Principle

NE555 circuit provides IR signals with frequency to the emitter TX, then the IR signals will fade with the increase of transmission distance. If encountering the obstacle, it will be reflected back.

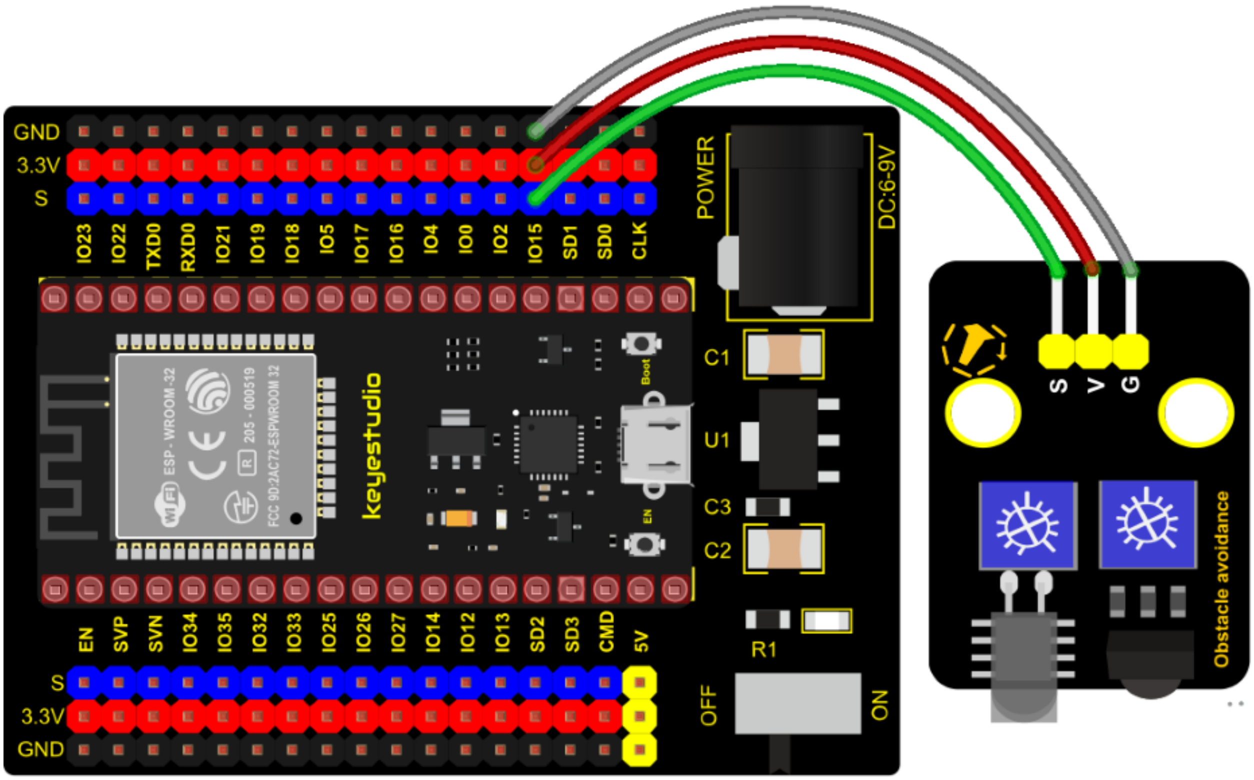
When the receiver RX meets the weak signals reflected back, the receiving pin will output high levels, which indicates the obstacle is far away. On the contrary, it the reflected signals are stronger, low levels will be output, which represents the obstacle is close. There are 2 potentiometers on the sensor, and by adjusting the 2 potentiometers, we can adjust its effective distance.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Obstacle Avoidance Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//
↪**************************************************************************************
/*
 * Filename    : Touch sensor
 * Description : Reading touch value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int touch = 15; //The key of PIN
void setup() {
  Serial.begin(9600);//Baud rate is 9600
  pinMode(touch, INPUT);//Setting input mode
}

void loop() {
  val = digitalRead(touch);//Read the value of the key
  Serial.print(val);//Print out key values
  if (val == 1) {//Press for high level
    Serial.print("        ");
    Serial.println("Press the button");
    delay(100);
  }
  else {//Release to low level
    Serial.print("        ");
    Serial.println("Loosen the button");
    delay(100);
```

(continues on next page)

(continued from previous page)

```
        }
    }
    //
↪***********************************************************************************
```

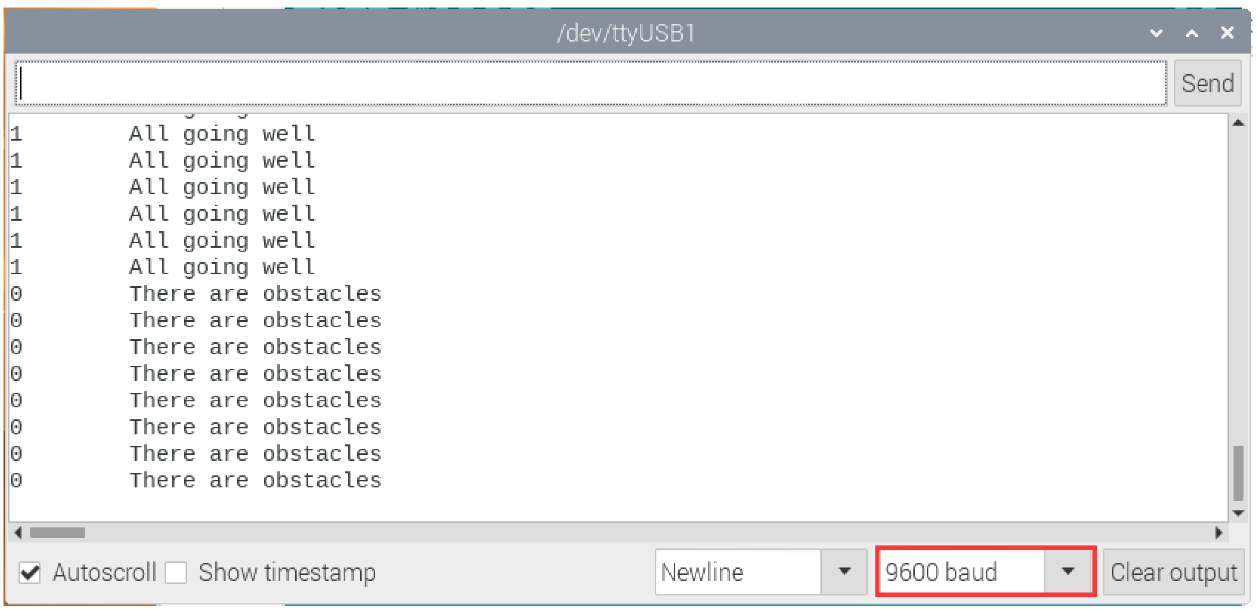
Code Explanation

Note:

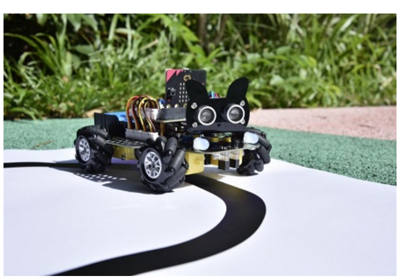
Upload the test code and wire up according to the connection diagram. After powering on, we start to adjust the two potentiometers to sense distance.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters. When the sensor detects the obstacle, the val is 0,the monitor will show“There are obstacles”; if the obstacle is not detected, the val is 1,“All going well” will be shown.



7.5.10 Project 10: Line Tracking Sensor



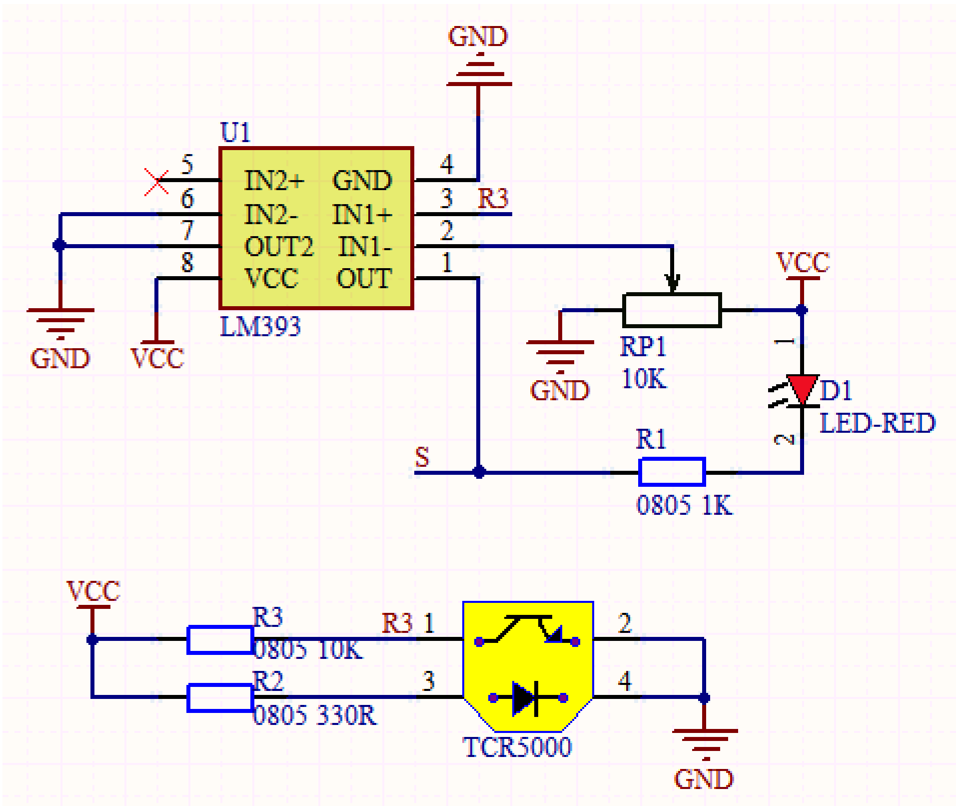
Description

In this kit, there is a DIY electronic building block single-channel line tracking sensor which mainly uses a TCRT5000 reflective black and white line recognition sensor element.

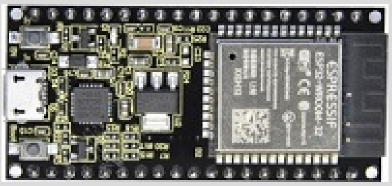
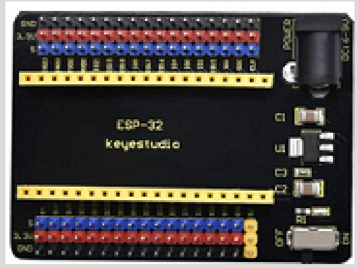
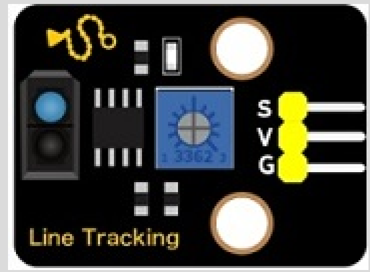
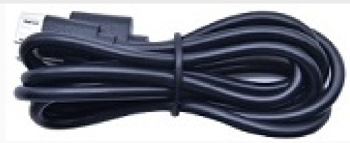
In the experiment, we judge the color (black and white) of the object detected by the sensor by reading the high and low levels of the S terminal on the module; and display the test results on the shell.

Working Principle

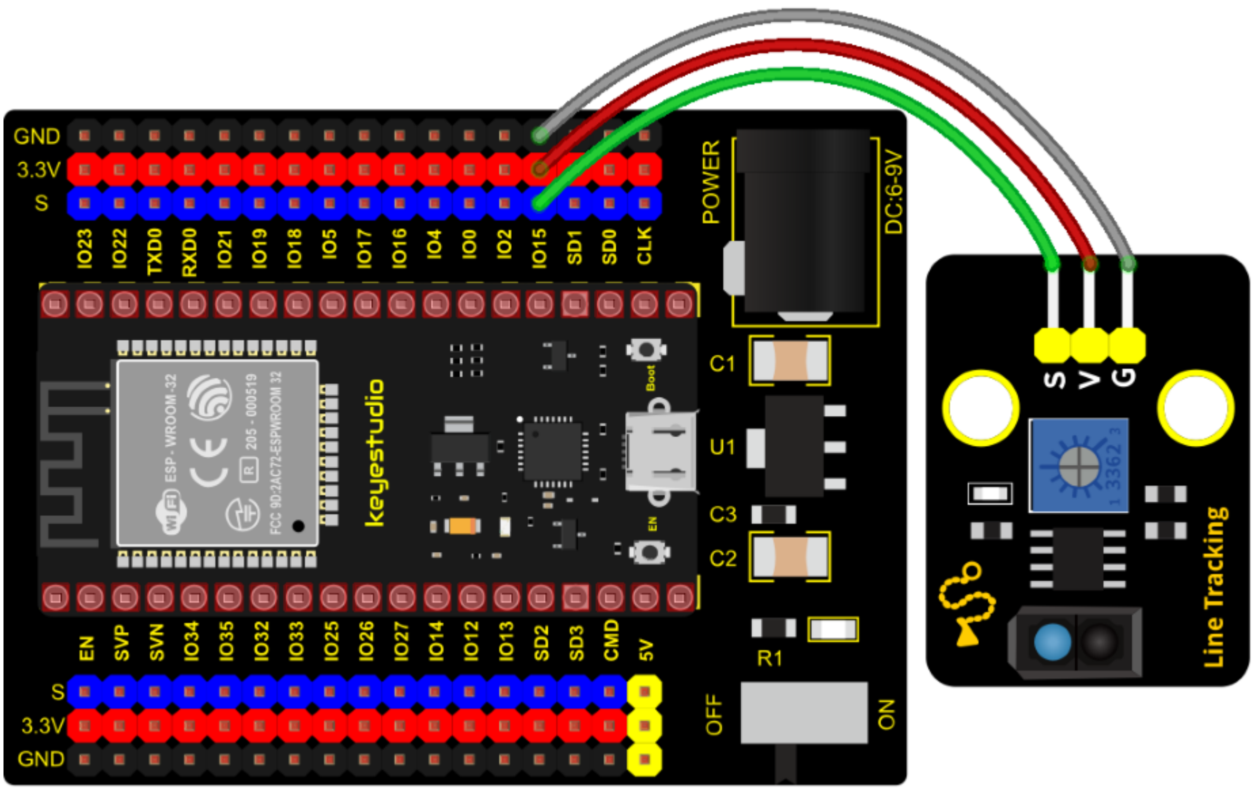
When a black or no object is detected, the signal terminal will output high levels; when white object is detected, the signal terminal is low level; its detection height is 0-3cm. We can adjust the sensitivity by rotating the potentiometer on the sensor. When the potentiometer is rotated, the sensitivity is best when the red LED on the sensor is at the critical point between off and on.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Line Tracking Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : line tracking
```

(continues on next page)

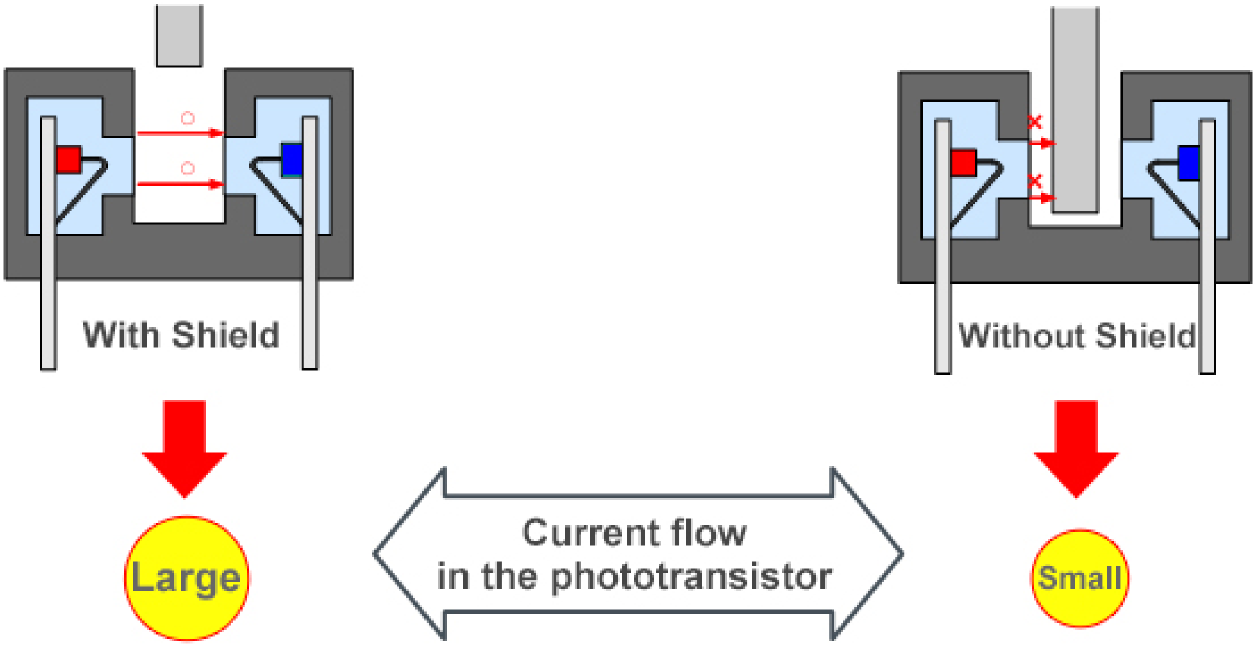
(continued from previous page)

```
 * Description : Reading the tracking sensor value
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(15, INPUT);//Sets sensor pin to input mode
}

void loop() {
  val = digitalRead(15);//Read the digital level output by the patrol sensor
  Serial.print(val);//Serial port print value
  if (val == 0) {//White val is 0 detected
    Serial.print("        ");
    Serial.println("White");
    delay(100);
  }
  else {//Black val is 1 detected
    Serial.print("        ");
    Serial.println("Black");
    delay(100);
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters. when the sensor doesn't detect an object or detects a black object, the val is 1, and the monitor will display "1 Black" ; when a white object (can reflect light) is detected, the val is 0, and the monitor will display "0 White" ;

7.5.11 Project 11: Photo Interrupter

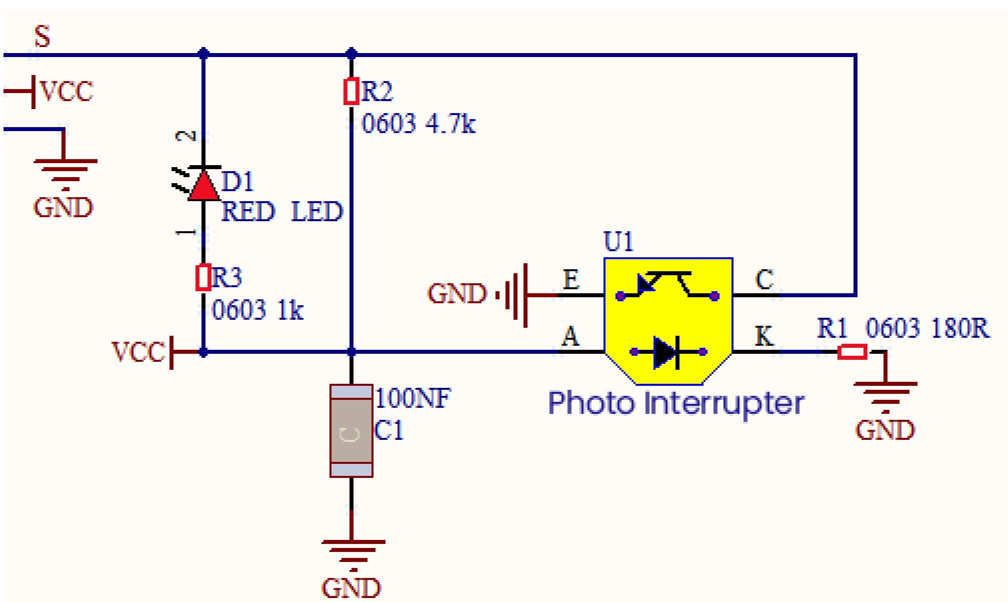


Description

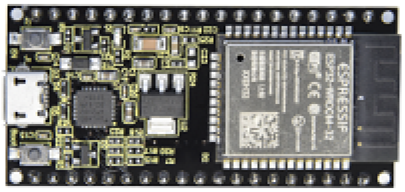
This kit contains a photo interrupter which mainly uses 1 ITR-9608 photoelectric switch. It is a photoelectric switch optical switch sensor.

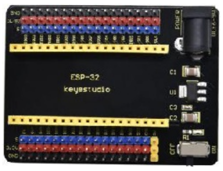
Working Principle

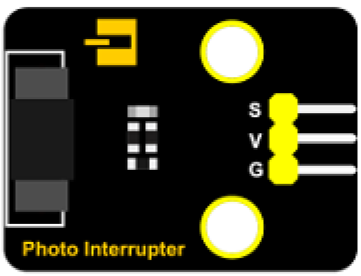
When the paper is put in the slot, C is connected with VCC and the signal end S of the sensor are high levels; then the red LED will be off. Otherwise, the red LED will be on.

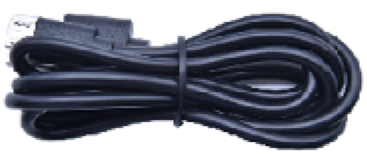


Required Components

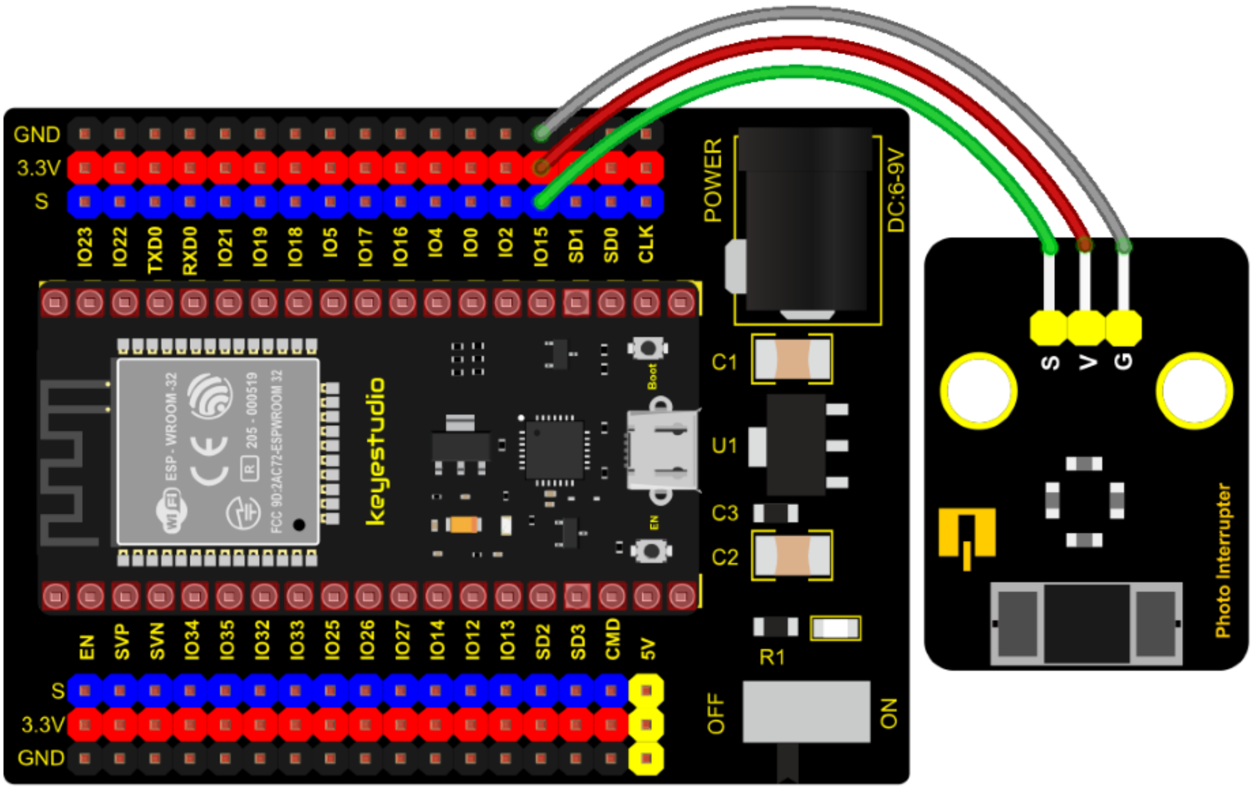








Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : Photo_Interrupt
 * Description : Light snap sensor counting
 * Auther      : http://www.keyestudio.com
*/
int PushCounter = 0; //The count variable is assigned an initial value of 0
int State = 0; //Store the current state of the sensor output
int lastState = 0; //Stores the state of the last sensor output
void setup() {
  Serial.begin(9600);//Set the baud rate to 9600
  pinMode(15, INPUT);//Set the light snap sensor pin to input mode
}

void loop() {
  State = digitalRead(15);//Read current state
  if (State != lastState) {//If the state is different from the last read
    if (State == 1) {//block the light
      PushCounter = PushCounter + 1;//Count + 1
      Serial.println(PushCounter);//Print count
    }
  }
  lastState = State;//Update state
}
//*************************************************************************************
```

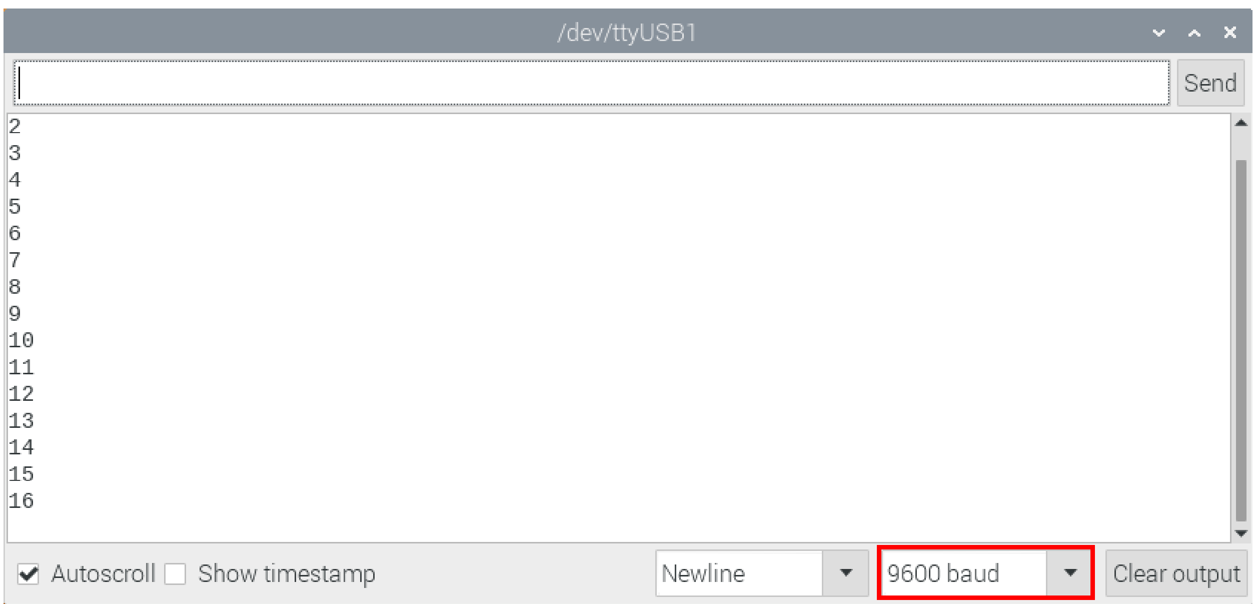
Code Explanation

Logic setting:

Initial Setting	Set PushCounter to 0Set State to 0 (value of the sensor)Set lastState to 0	
when an object enters the slot	lastState is 0State turns into 1; lastState turns into 1	Set PushCounter to PushCounter+1print the value of PushCounter
when the object leaves the slot	lastState is 1State becomes 0two data are not equallastState turns into 0.	PushCounterdoesn't change;Don't print the value of PushCounter
When the object goes through this slot again	lastState is 0, State becomes 1two data are not equallastState turns into 1.	SetPushCounter to PushCounter+1. And print the value of PushCounter
When the object leaves this slot again	lastState is 1State turns into 0two data are not equal lastState turns into 0	PushCounter doesn't change;Don't print the PushCounter value

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the PushCounter data. Every time when the object passes through the slot of the sensor, the PushCounter data will increase by 1 continuously, as shown below;



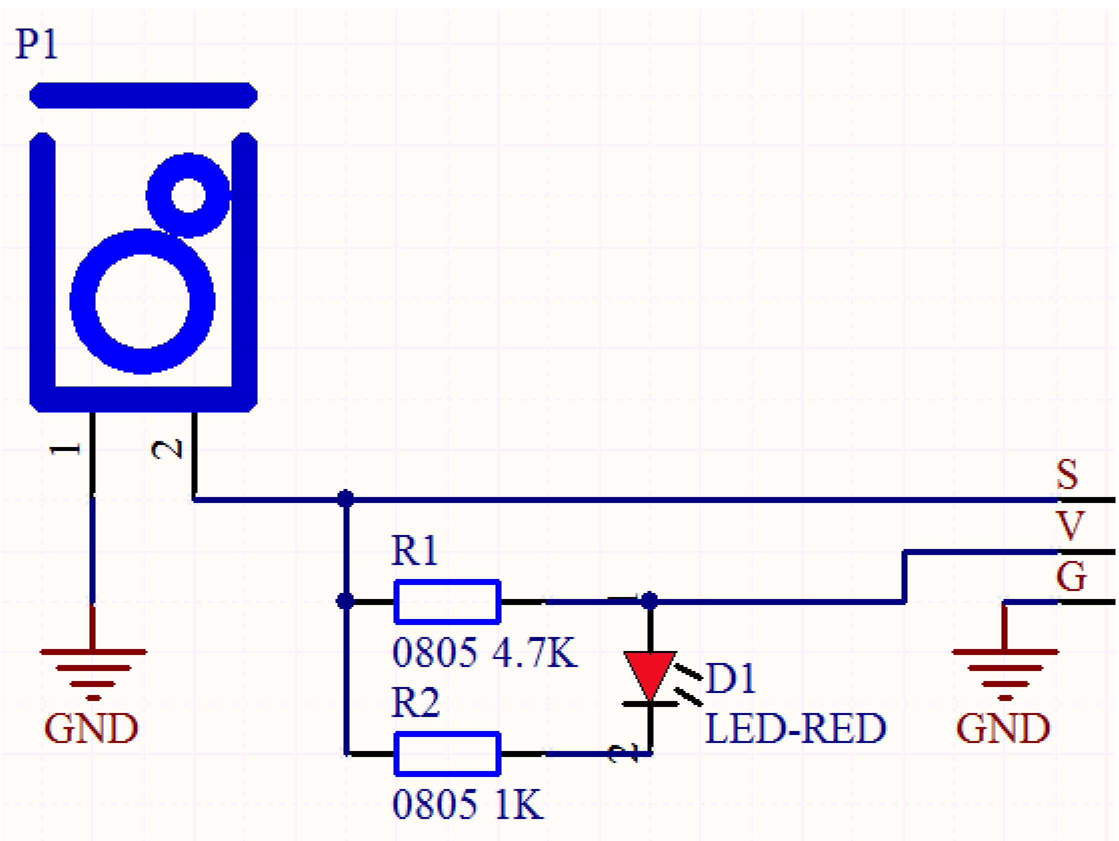
7.5.12 Project 12: Tilt Module

Overview

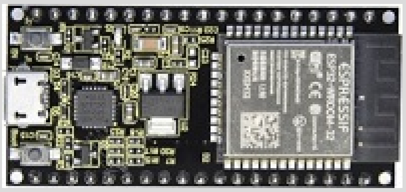
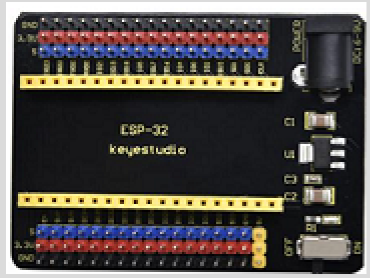
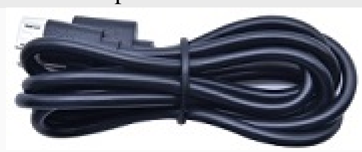
In this kit, there is a Keyestudio tilt sensor. The tilt switch can output signals of different levels according to whether the module is tilted. There is a ball inside. When the switch is higher than the horizontal level, the switch is turned on, and when it is lower than the horizontal level, the switch is turned off. This tilt module can be used for tilt detection, alarm or other detection.

Working Principle

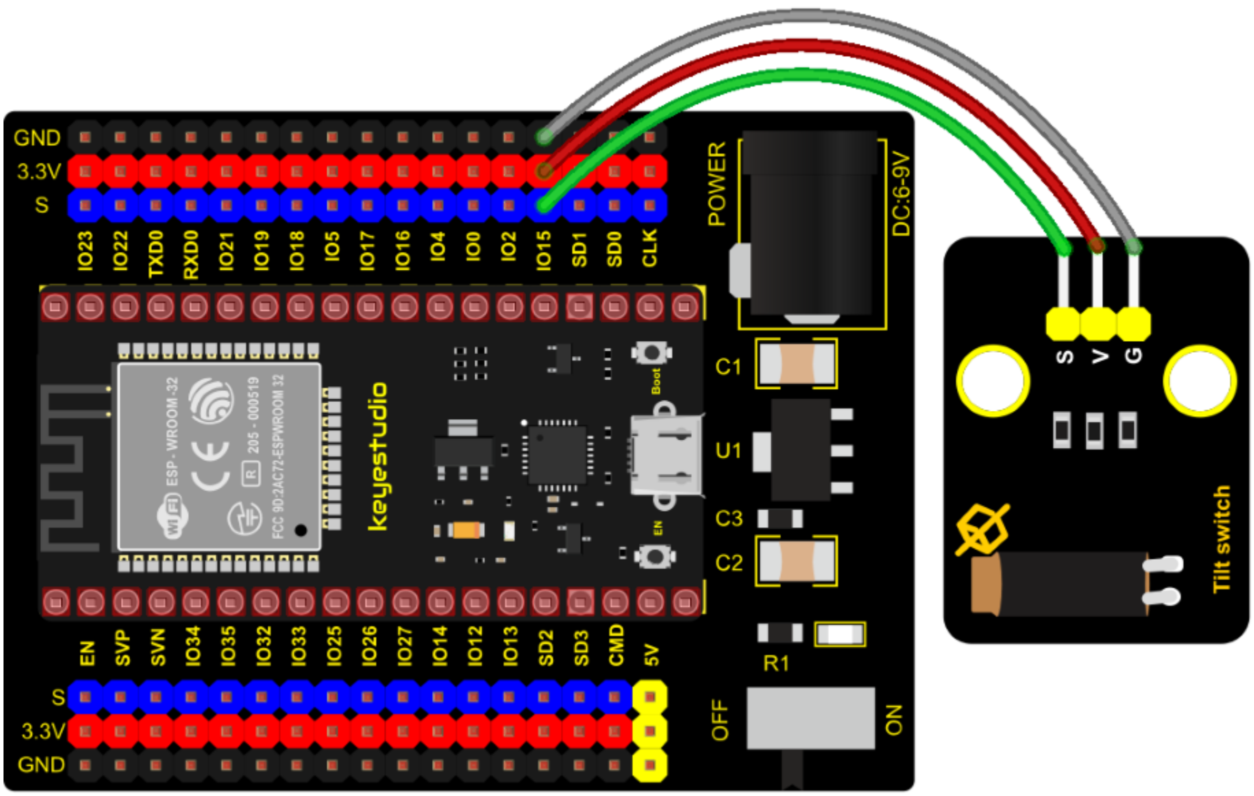
The working principle is pretty simple. When pin 1 and 2 of the ball switch P1 are connected, the signal S is low level and the red LED will light up; when they are disconnected, the pin will be pulled up by the 4.7K R1 and make S a high level, then LED will be off.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	KeyestudioTilt Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Tilt switch
```

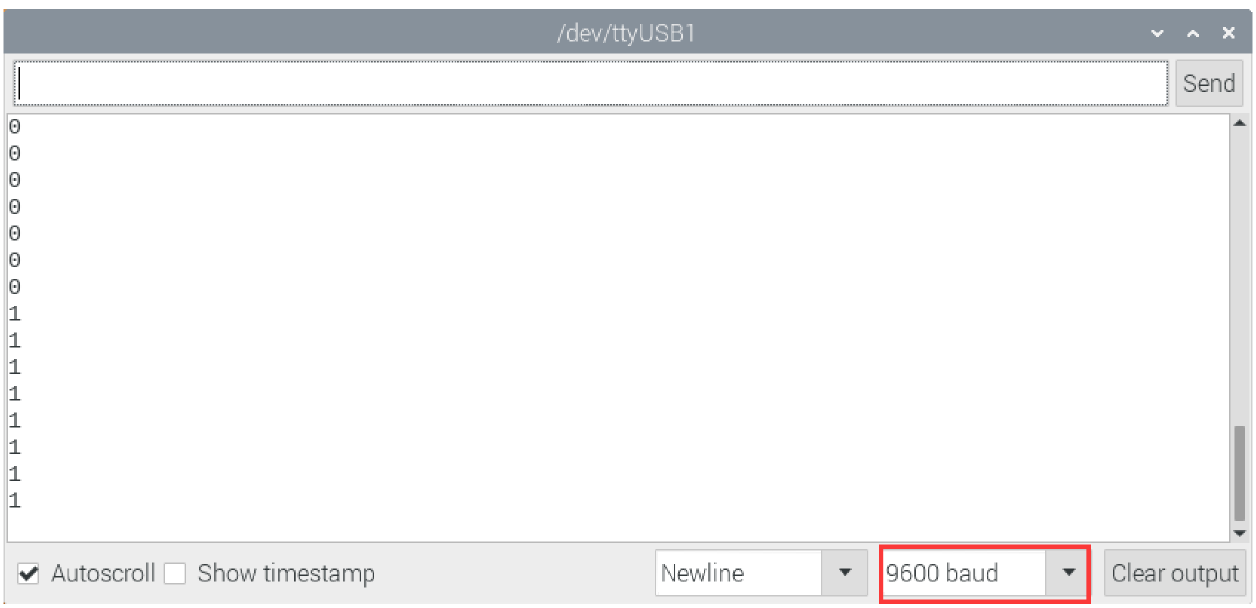
(continues on next page)

(continued from previous page)

```
 * Description : Reading the tilt sensor value
 * Auther      :http://www.keyestudio.com
*/
int val; //Store the level value output by the tilt sensor

void setup() {
  Serial.begin(9600);
  pinMode(15, INPUT);  //Connect the pin of the tilt sensor to GP15 and set GP15 to the input mode
}

void loop() {
  val = digitalRead(15); //Read module level signal
  Serial.println(val);  //Newline print
  delay(100); //Delay for 100 ms
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then make the tilt module incline to one side, the red LED on the module will be off and the monitor will display "1". In contrast, if you make it incline the other side, the red LED will light up and the monitor will display "0".



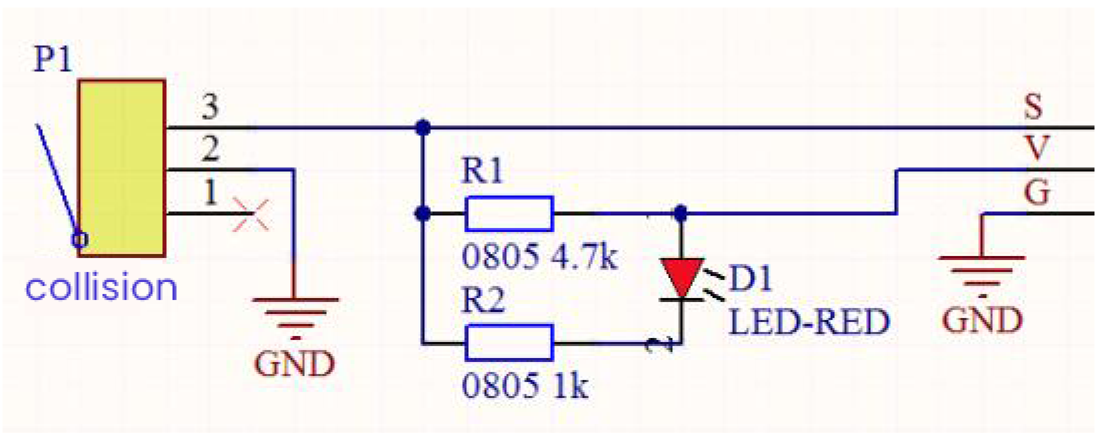
7.5.13 Project 13: Collision Sensor

Description

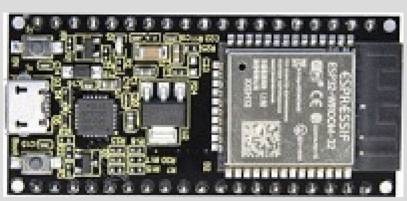
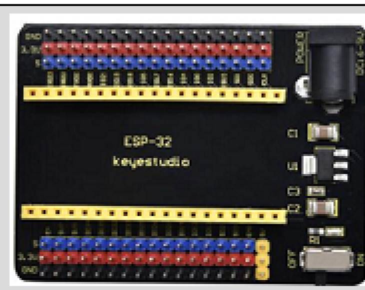
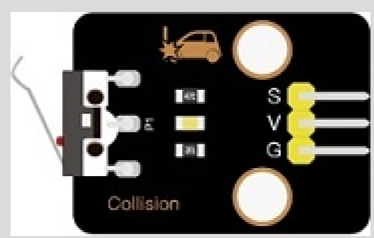
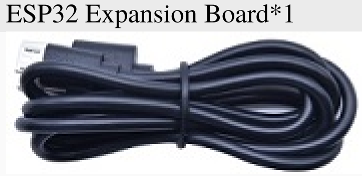
The collision sensor uses a tact switch. This sensor is often used as a limit switch in 3D printers. In the experiment, we judge whether the sensor shrapnel is pressed down by reading the high and low levels of the S terminal on the module; and, we display the test results in the shell.

Working Principle

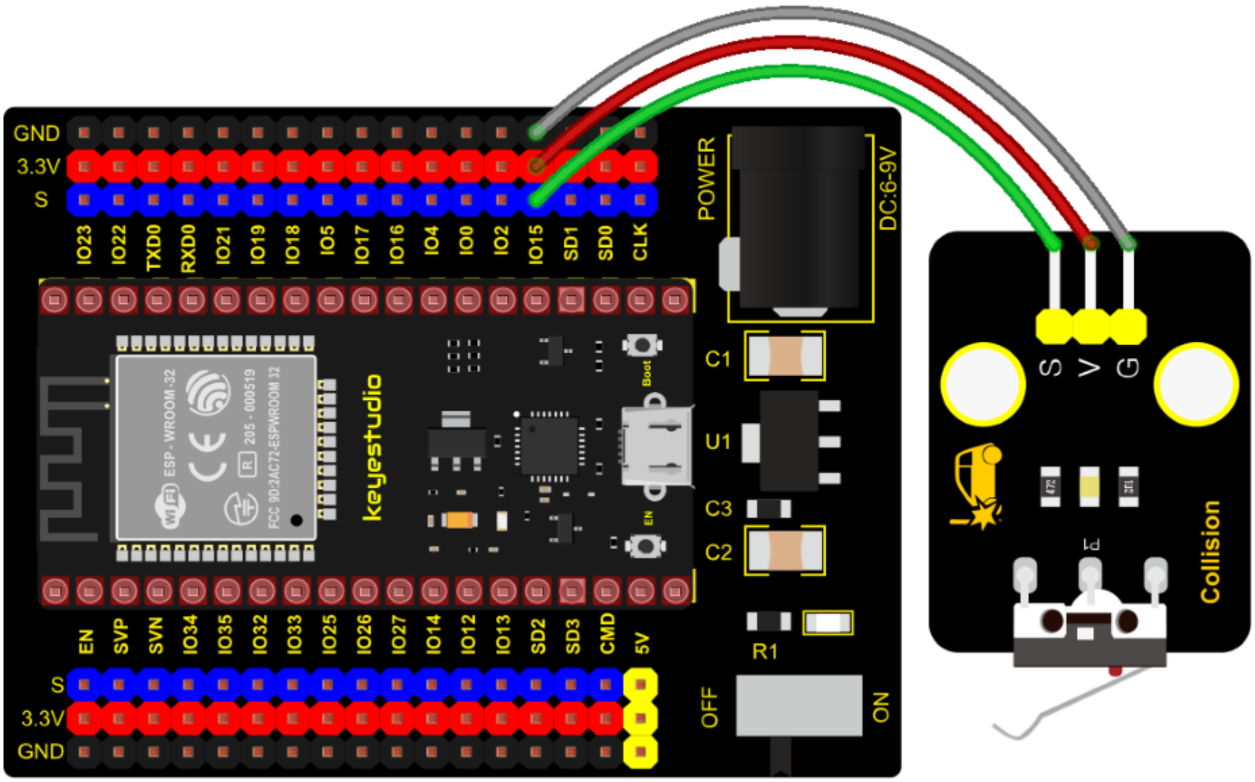
It mainly uses a tact switch. When the shrapnel of the tact switch is pressed, 2 and 3 are connected, the signal terminal S is low level, and the red LED on the module lights up; when the touch switch is not pressed, 2 and 3 are not connected, and 3 is pulled up to a high level by the 4.7K resistor R1, that is, the sensor signal terminal S is a high level, and the built-in red LED will be off at this time.



Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Collision Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : collision sensor
 * Description : Reading the value of the collision sensor
 * Auther      :  http://www.keyestudio.com
*/
int val = 0;
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(15, INPUT);//Set collision sensor pin 15 to input mode
}

void loop() {
  val = digitalRead(15);//Read the value of the collision sensor
  Serial.print(val);//Newline print
  if (val == 0) {//Collision val is 0
    Serial.print("        ");
    Serial.println("The end of his!");
    delay(100);
  }
  else {// No collision val is 1
    Serial.print("        ");
    Serial.println("All going well");
    delay(100);
  }
```

(continues on next page)

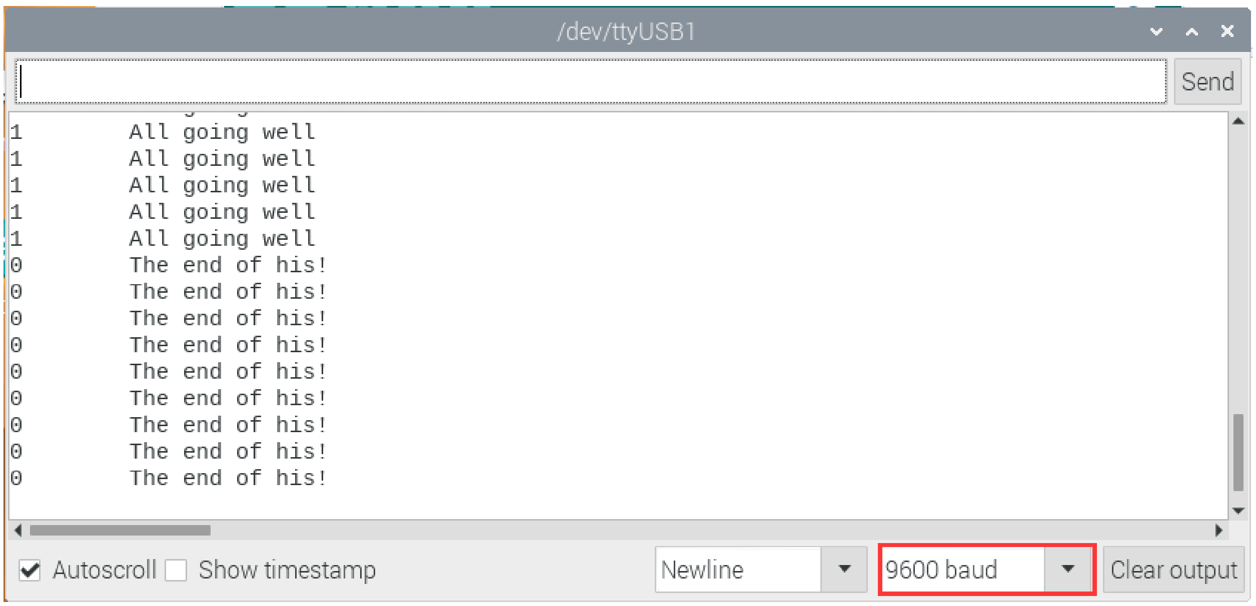
(continued from previous page)

```
}
//*********************************************************************************
```

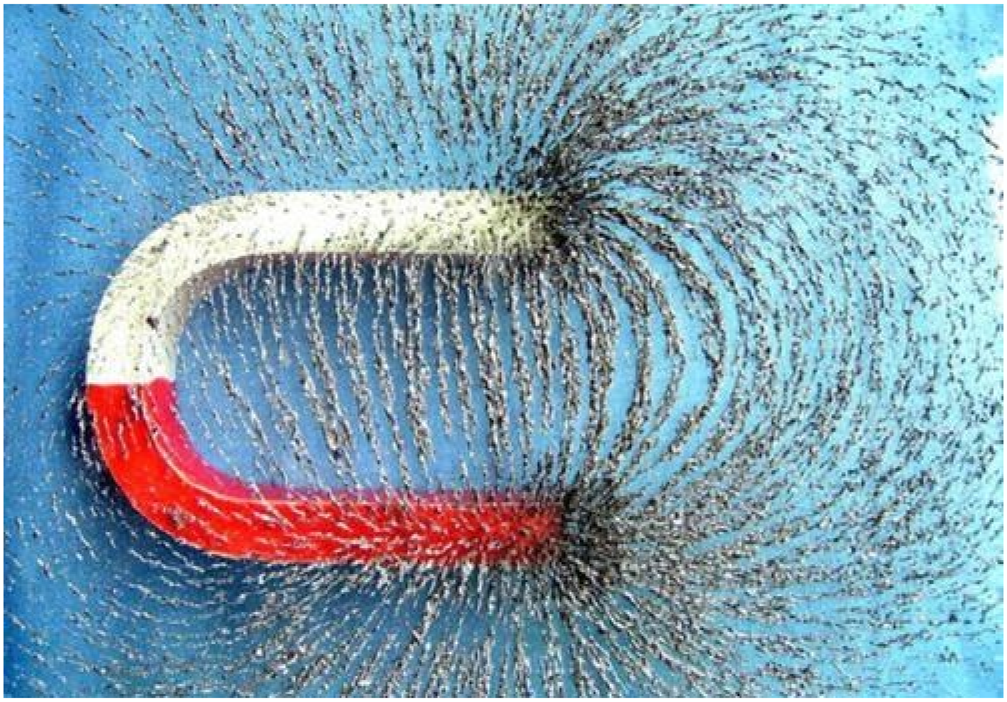
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters.

In the experiment, when the shrapnel on the sensor is pressed down, val is 0, the red LED of the module is on, and "0 The end of his!" is printed; when the shrapnel is released, the val is 1, the red LED of the module is off, and "1 All going well" is printed. "!" character, as shown below.



7.5.14 Project 14: Hall Sensor

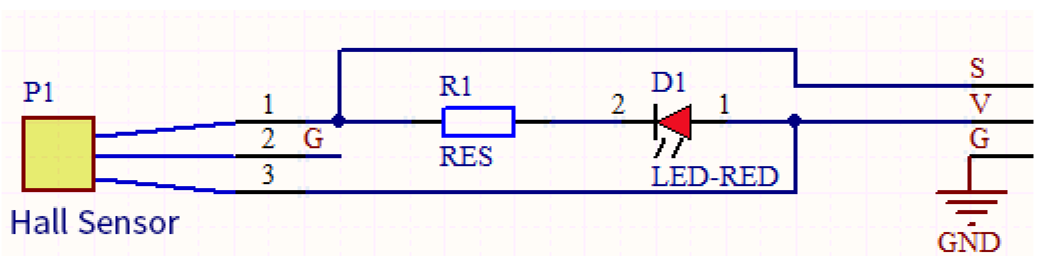


Description

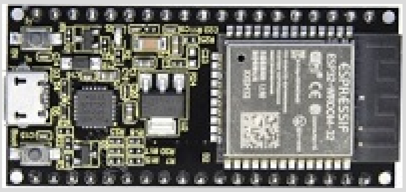
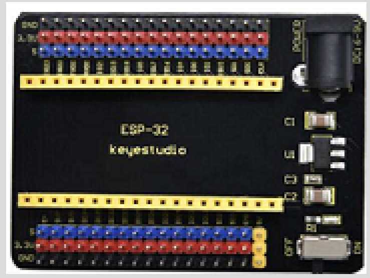
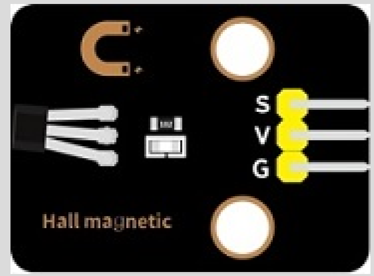
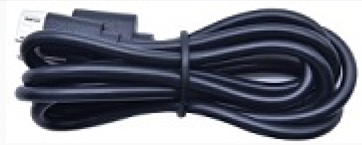
In this kit, there is a Hall sensor which mainly adopts a A3144 linear Hall element. The element P1 is composed of a voltage regulator, a Hall voltage generator, a differential amplifier, a Schmitt trigger, a temperature compensation circuit and an open-collector output stage. In the experiment, we use the Hall sensor to detect the magnetic field and display the test results on the shell.

Working Principle

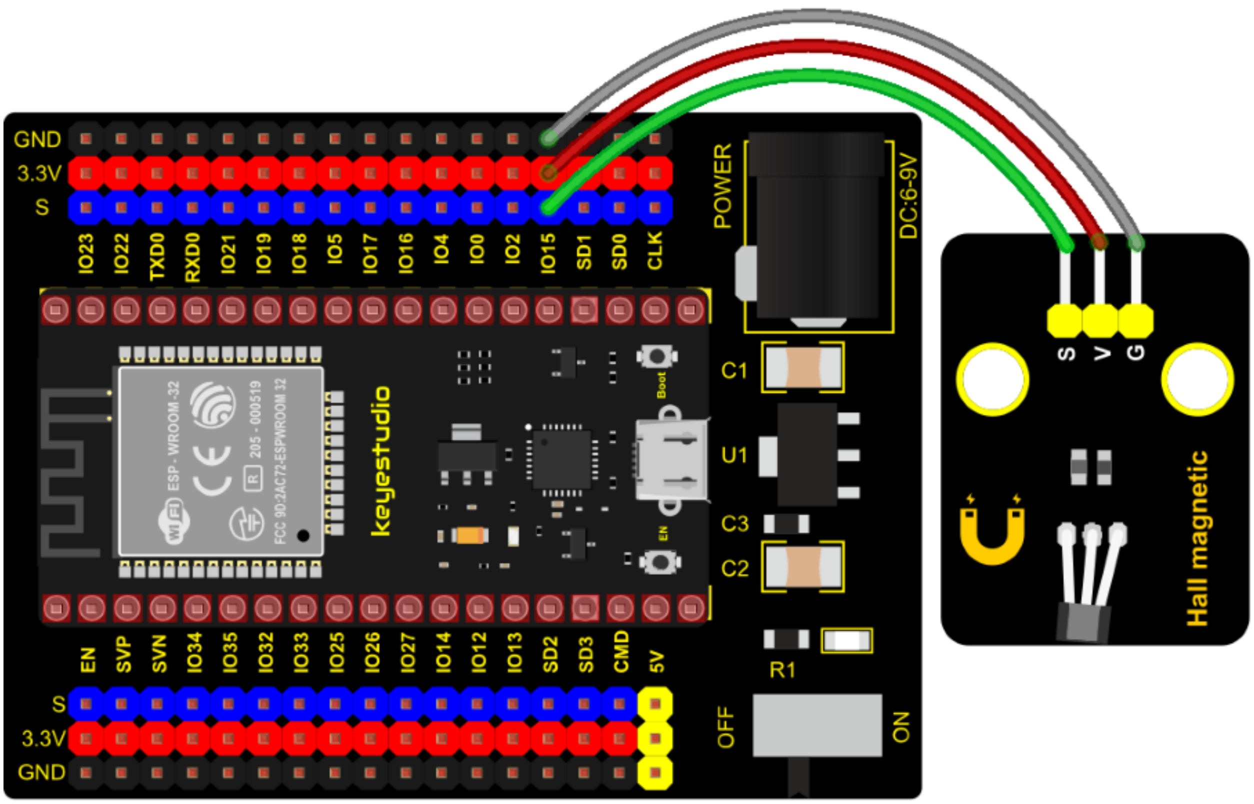
When the sensor detects no magnetic field or a north pole magnetic field, the signal terminal will be high level; when it senses a south pole magnetic field, the signal terminal will be low levels. The stronger the magnetic field strength is, induction distance is longer.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Hall Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Hall magnetic
```

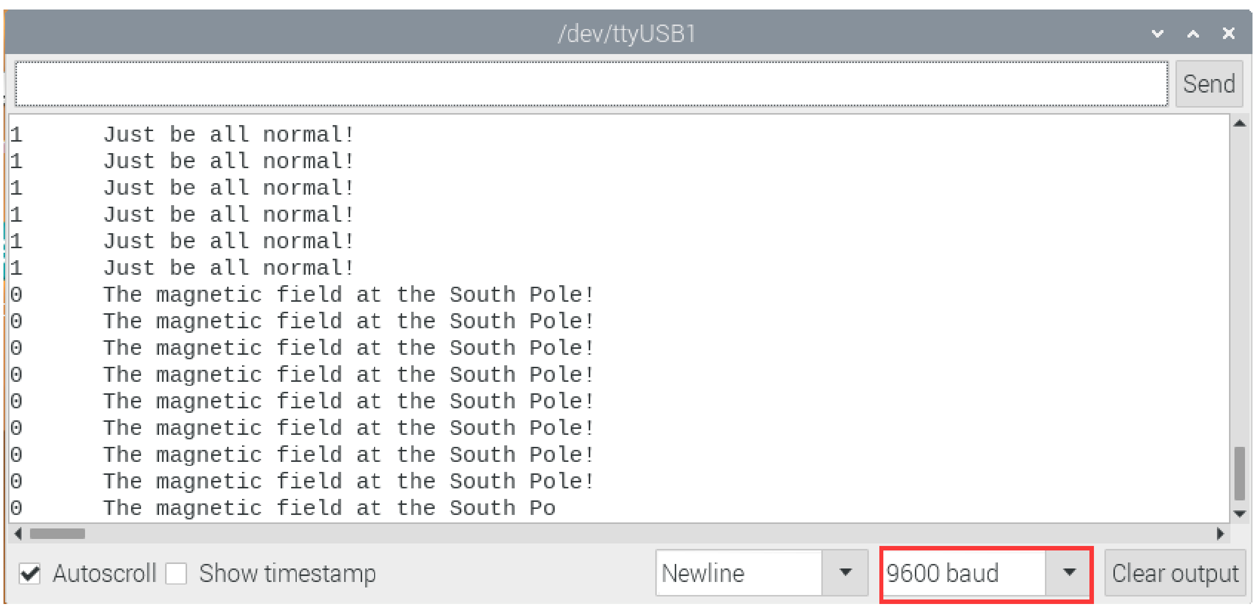
(continues on next page)

(continued from previous page)

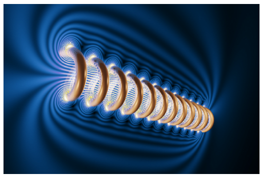
```
 * Description : Reading the value of hall magnetic sensor
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int hallPin = 15;  //Hall sensor pin is connected to GPIO15
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(hallPin, INPUT);//Set pin to input mode
}

void loop() {
  val = digitalRead(hallPin);//Read the level value of hall sensor
  Serial.print(val);//Print val
  if (val == 0) {//There is a South Pole magnetic field
    Serial.println("      The magnetic field at the South Pole!");
  }
  else {//If not
    Serial.println("      Just be all normal!");
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, when the sensor detects no magnetic fields or the north pole magnetic field, the monitor l will show"1 Just be all normal!"and the LED on the sensor will be off; When it detects the south pole magnetic field,"0 The magnetic field at the South Pole!"and the LED on the sensor will be on.



7.5.15 Project 15: Reed Switch Module



Overview

In this kit, there is a Keyestudio reed switch module, which mainly uses a MKA10110 green reed component.

The reed switch is the abbreviation of the dry reed switch. It is a passive electronic switch element with contacts.

It has the advantages of simple structure, small size and easy control.

Its shell is a sealed glass tube with two iron elastic reed electric plates.

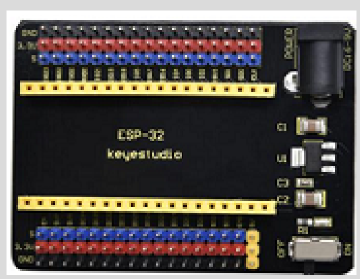
In the experiment, we will determine whether there is a magnetic field near the module by reading the high and low level of the S terminal on the module; and, we display the test result in the shell.

Working Principle

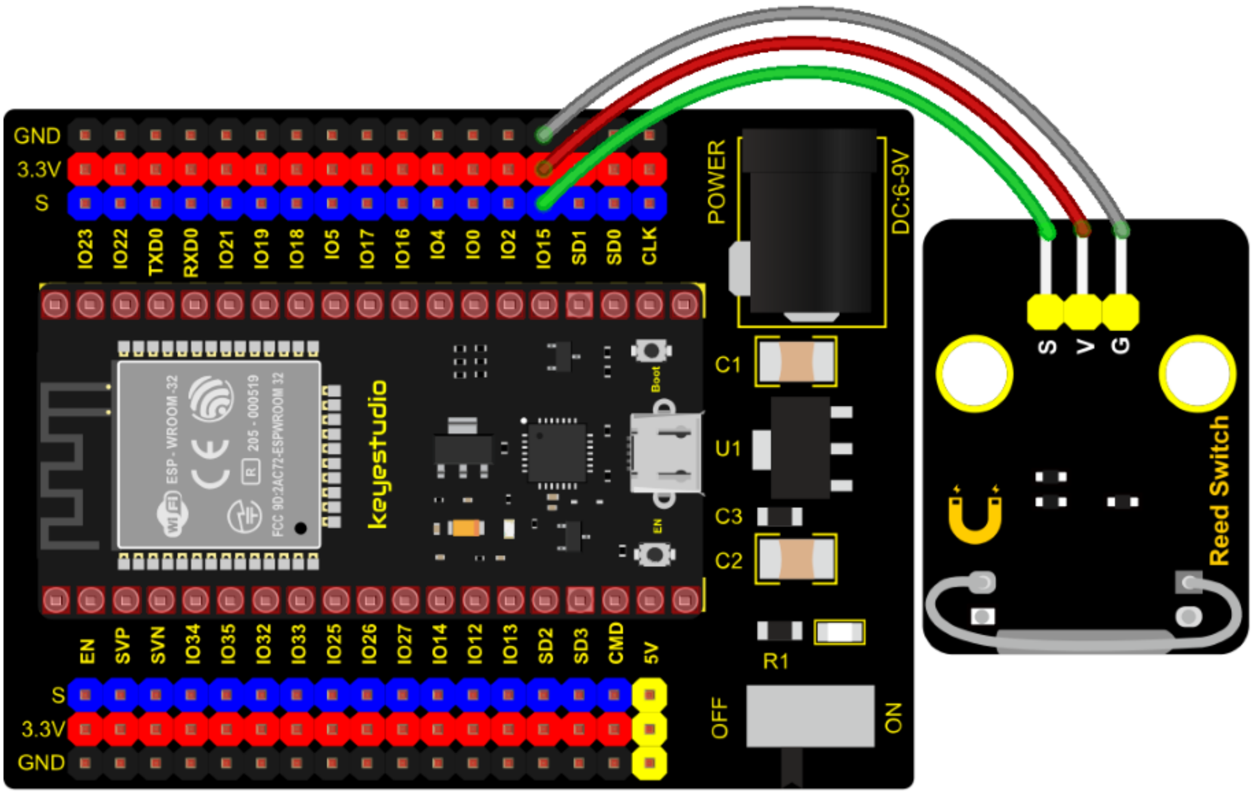
In normal conditions, the glass tube in the two reeds made of special materials are separated. When a magnetic substance close to the glass tube, in the role of the magnetic field lines, the pipe within the two reeds are magnetized to attract each other in contact, the reed will suck together, so that the junction point of the connected circuit communication.

After the disappearance of the outer magnetic reed because of their flexibility and separate, the line is disconnected. The sensor uses this characteristic to build a circuit to convert magnetic field signal into high and low level signal.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Reed Switch Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Reed Switch
 * Description : Read the value of the reed sensor
 * Auther      : http://www.keyestudio.com
*/
int val = 0;
int reedPin = 15; //Define dry reed module signal pin connected to GPIO15
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(reedPin, INPUT);//Set mode to input
}

void loop() {
  val = digitalRead(reedPin);//Read digital level
  Serial.print(val);//Serial port shows up

  if (val == 0) {//There's a magnetic field nearby
    Serial.print("        ");
    Serial.println("A magnetic  field");
    delay(100);
  }
  else {//There is no magnetic field
    Serial.print("        ");
    Serial.println("There is no magnetic field");
```

(continues on next page)

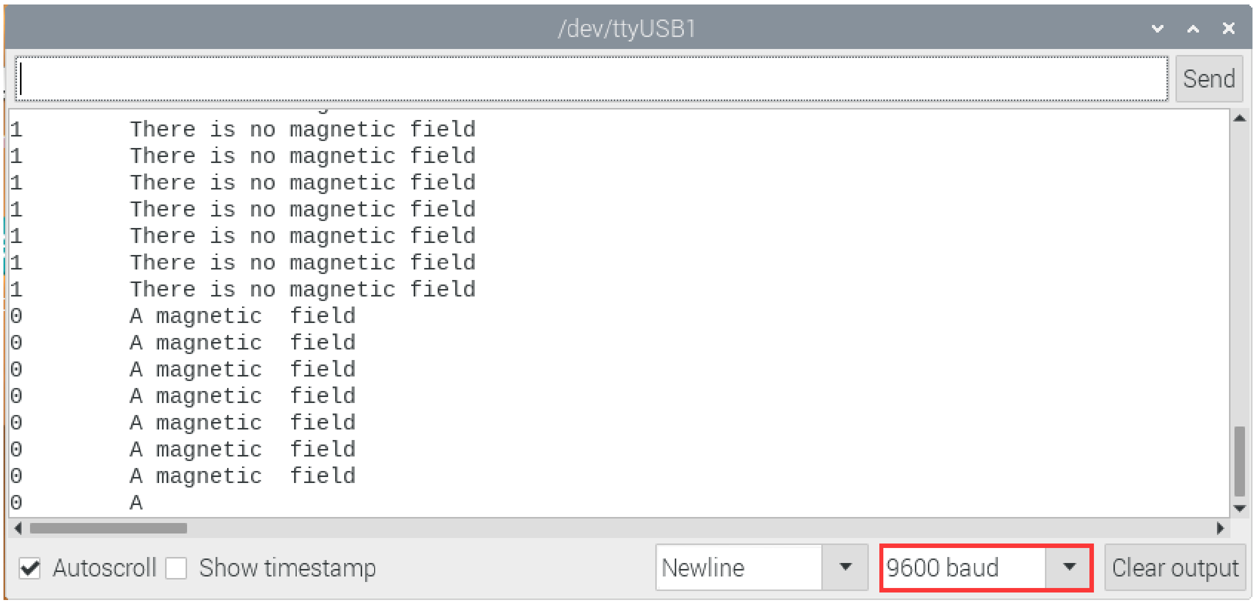
(continued from previous page)

```
    delay(100);
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onopen the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters.

When the sensor detects a magnetic field, val is 0 and the red LED of the module lights up, "0 A magnetic field" will be displayed; when no magnetic field is detected, val is 1, and the LED on the module goes out, "1 There is no magnetic field" will be shown, as shown below.



7.5.16 Project 16: PIR Motion Sensor

Overview

In this kit, there is a Keyestudio PIR motion sensor, which mainly uses an RE200B-P sensor elements. It is a human body pyroelectric motion sensor based on pyroelectric effect, which can detect infrared rays emitted by humans or animals, and the Fresnel lens can make the sensor's detection range farther and wider.

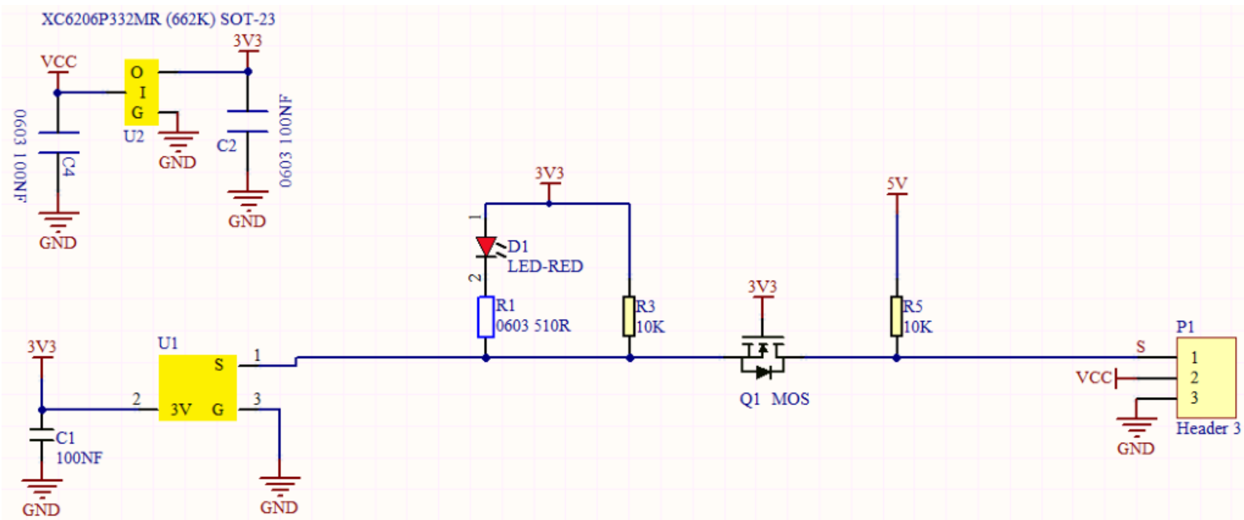
In the experiment, we determine if there is someone moving nearby by reading the high and low levels of the S terminal on the module. The detected results will be displayed on the Shell.

Working Principle

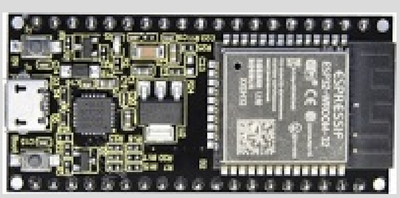
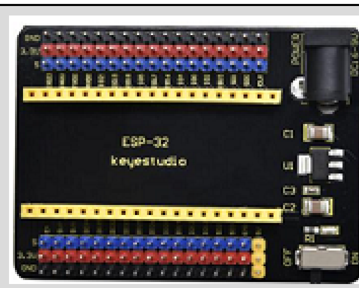
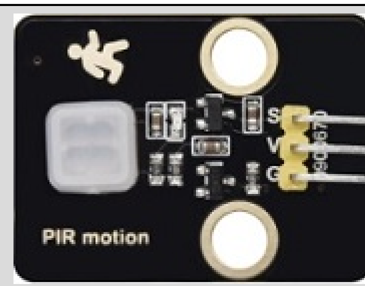
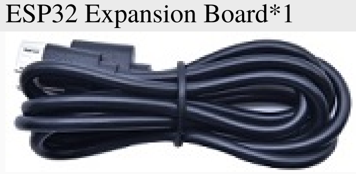
The upper left part is voltage conversion(VCC to 3.3V). The working voltage of sensors we use is 3.3V, therefore we can't use 5V directly. The voltage conversion circuit is needed.

When no person is detected or no infrared signal is received, and pin 1 of the sensor outputs low level. At this time, the LED on the module will light up and the MOS tube Q1 will be connected and the signal terminal S will detect Low levels.

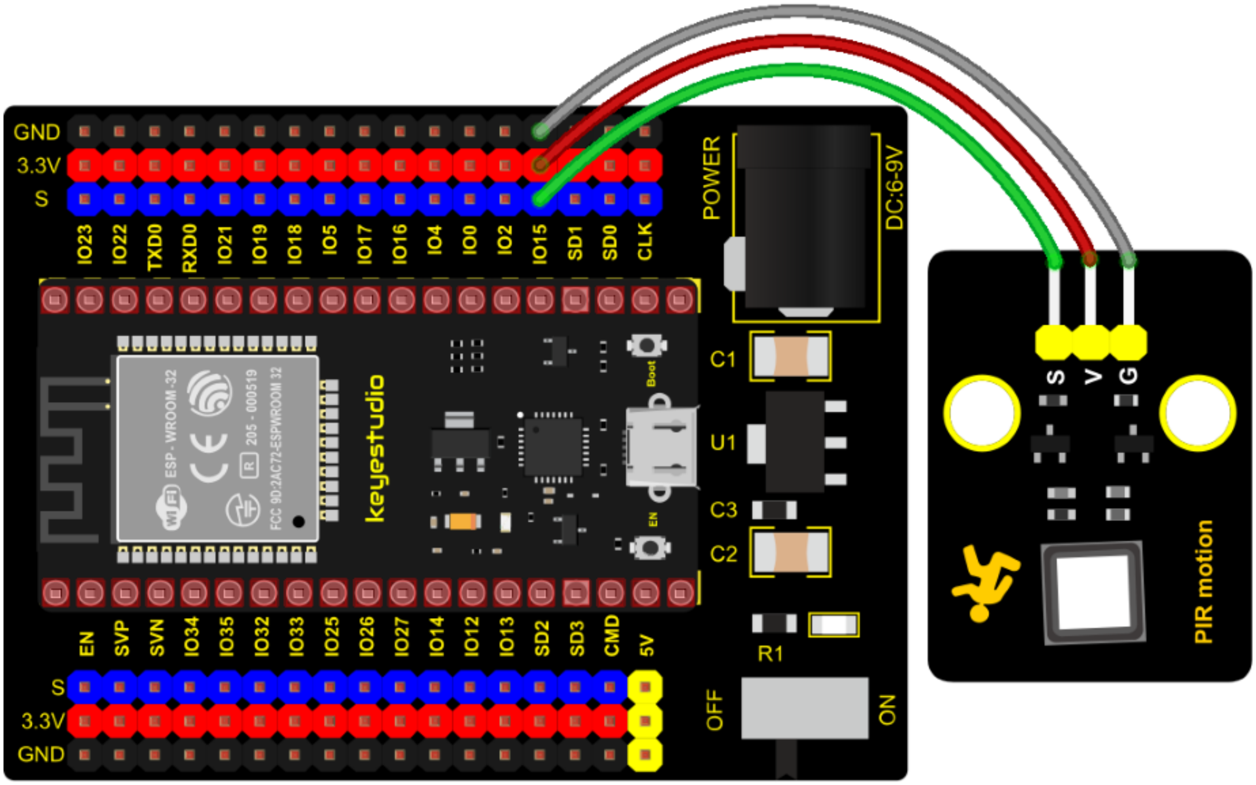
When one is detected or an infrared signal is received, and pin 1 of the sensor outputs a high level. Then LED on the module will go off, the MOS tube Q1 is disconnected and the signal terminal S will detect high levels.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY PIR Motion Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : PIR motion
 * Description : Reading the value of the human body infrared sensor
 * Auther      :  http://www.keyestudio.com
*/
int val = 0;
int pirPin = 15; //The pin of PIR motion sensor is defined as GPIO15
void setup() {
  Serial.begin(9600); //Set baud rate to 9600
  pinMode(pirPin, INPUT);  //Set the sensor to input mode
}

void loop() {
  val = digitalRead(pirPin);  //Read the sensor value
  Serial.print(val);//Print val value
  if (val == 1) {//There is movement nearby, output high level
    Serial.print("        ");
    Serial.println("Some body is in this area!");
    delay(100);
  }
  else {//If no  movement nearby, output low level
    Serial.print("        ");
    Serial.println("No one!");
```

(continues on next page)

(continued from previous page)

```
    delay(100);
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the corresponding data and characters.

When the sensor detects someone nearby, value is 1, the LED will go off and the monitor will show "1 Somebody is in this area!". In contrast, the value is 0, the LED will go up and "0 No one!" will be shown.

7.5.17 Project 17: Active Buzzer

Overview

In this kit, it contains an active buzzer module and a power amplifier module (the principle is equivalent to a passive buzzer).

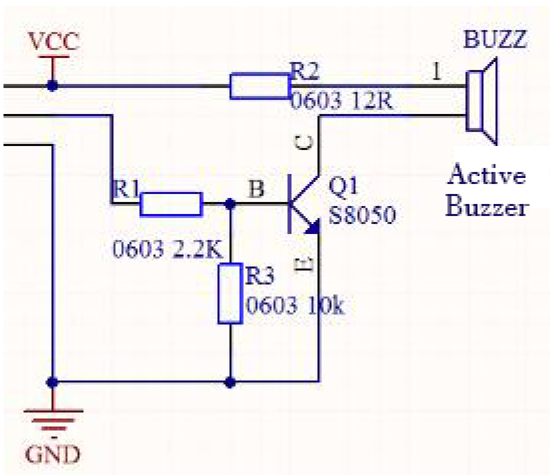
In this experiment, we control the active buzzer to emit sounds. Since it has its own oscillating circuit, the buzzer will automatically sound if given large voltage.

Working Principle

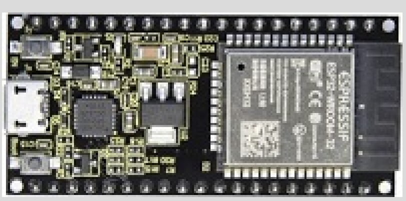
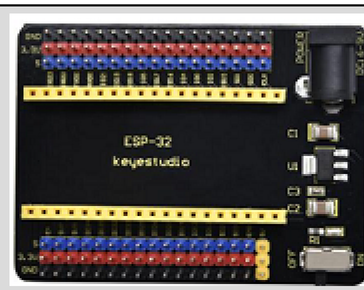
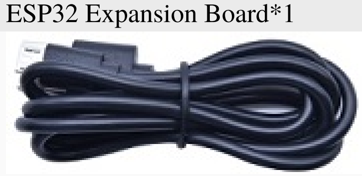
From the schematic diagram, the pin of buzzer is connected to a resistor R2 and another port is linked with a NPN triode Q1. So, if this triode Q1 is powered, the buzzer will sound.

If the base electrode of the triode connected to the R1 resistor is a high level, the triode Q1 will be connected.If the base electrode is pulled down by the resistor R3, the triode is disconnected.

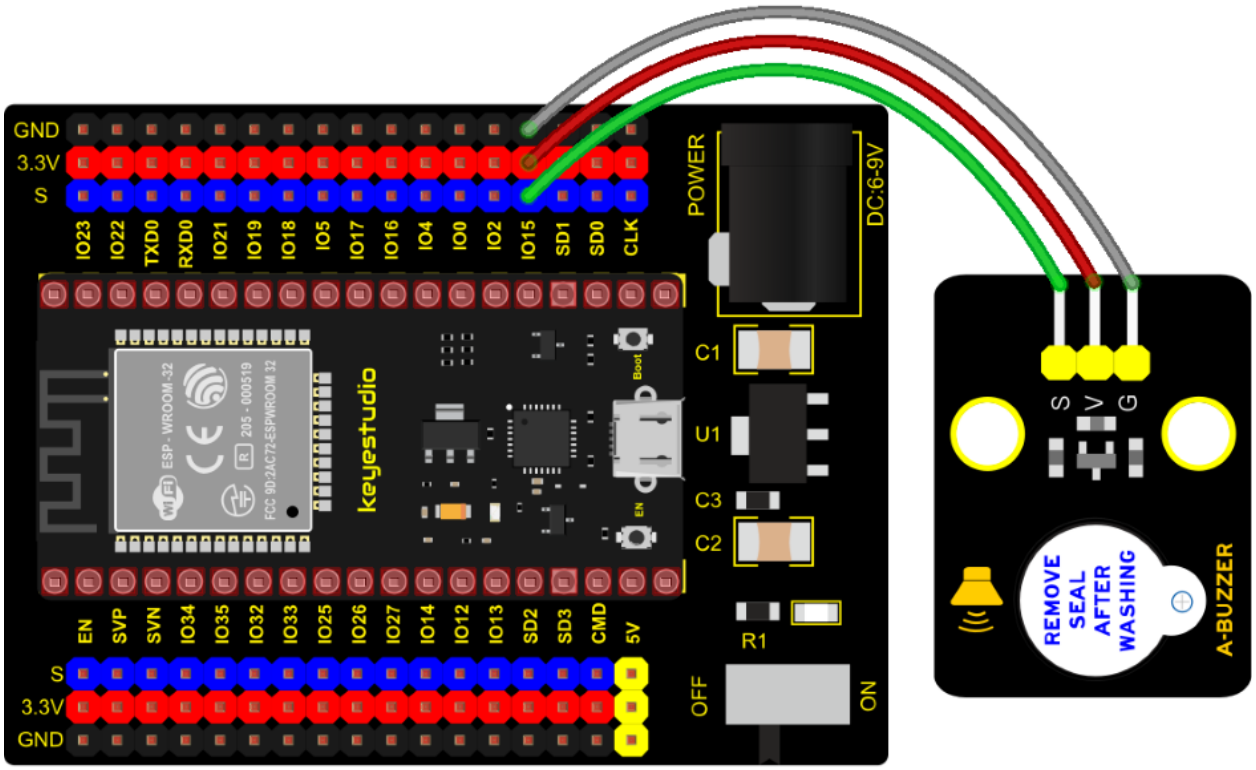
When we output a high level from the IO port to the triode, the buzzer will emit sounds; if outputting low levels, the buzzer won't emit sounds.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Active buzzer
 * Description : An active buzzer produces sound
 * Auther      : http://www.keyestudio.com
*/
int buzzer = 15; //Define buzzer receiver pin GPIO15
void setup() {
  pinMode(buzzer, OUTPUT);//Set the output mode
}

void loop() {
  digitalWrite(buzzer, HIGH); //sound production
  delay(1000);
  digitalWrite(buzzer, LOW); //Stop the sound
  delay(1000);
}
//**********************************************************************************
```

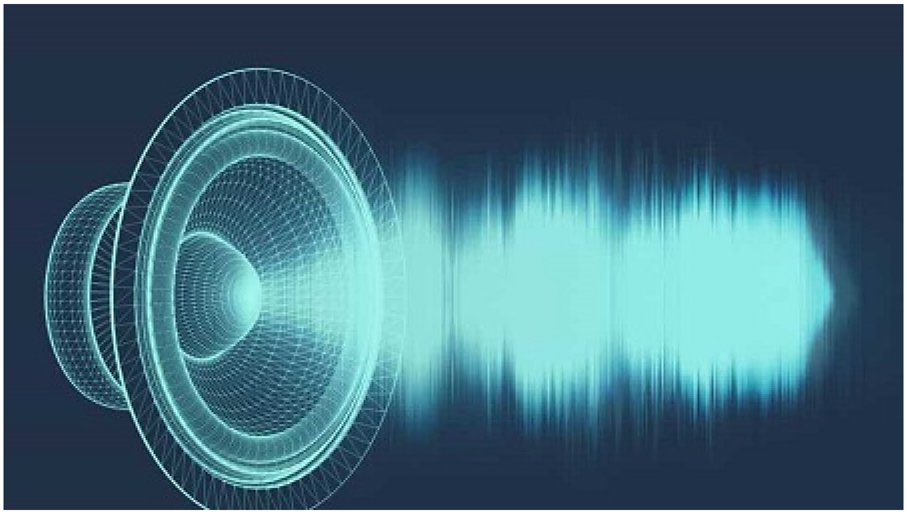
Code Explanation

In the experiment, we set the pin to GPIO15. When setting to high, the active buzzer will beep; when setting to low, the active buzzer will stop emitting sounds.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. The active buzzer will emit sound for 1 second, and stop for 1 second.

7.5.18 Project 18: 8002b Audio Power Amplifier



Overview

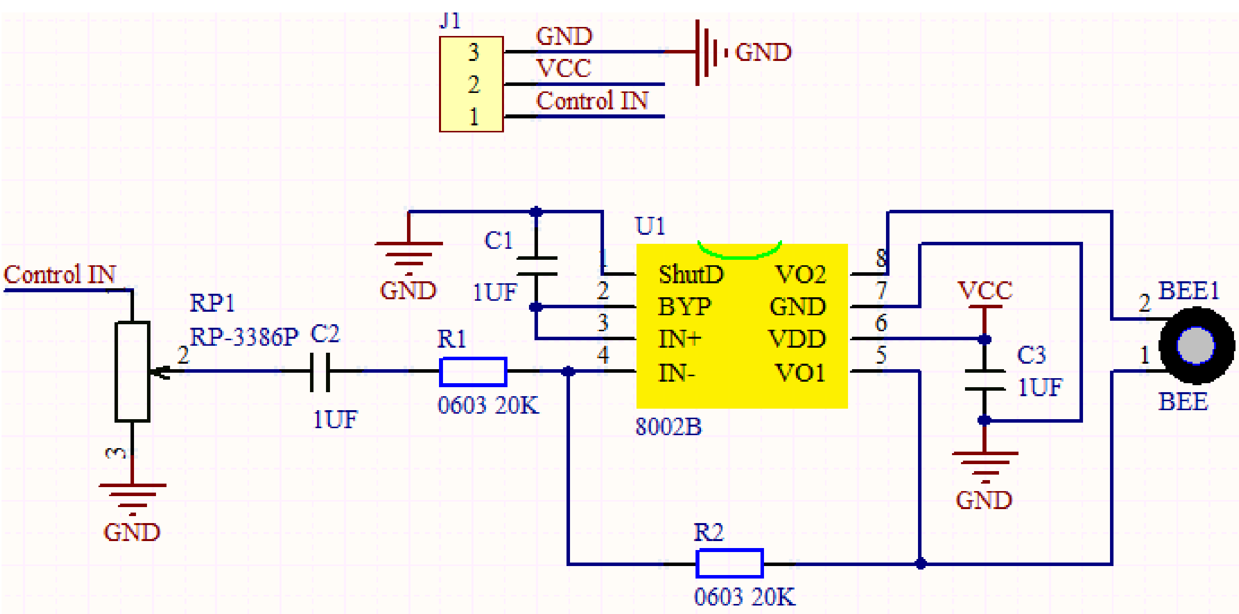
In this kit, there is a Keyestudio 8002b audio power amplifier. The main components of this module are an adjustable potentiometer, a speaker, and an audio amplifier chip;

The main function of this module is: it can amplify the output audio signal, with a magnification of 8.5 times, and play sound or music through the built-in low-power speaker, as an external amplifying device for some music playing equipment.

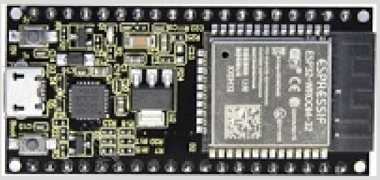
In the experiment, we used the 8002b power amplifier speaker module to emit sounds of various frequencies.

Working Principle

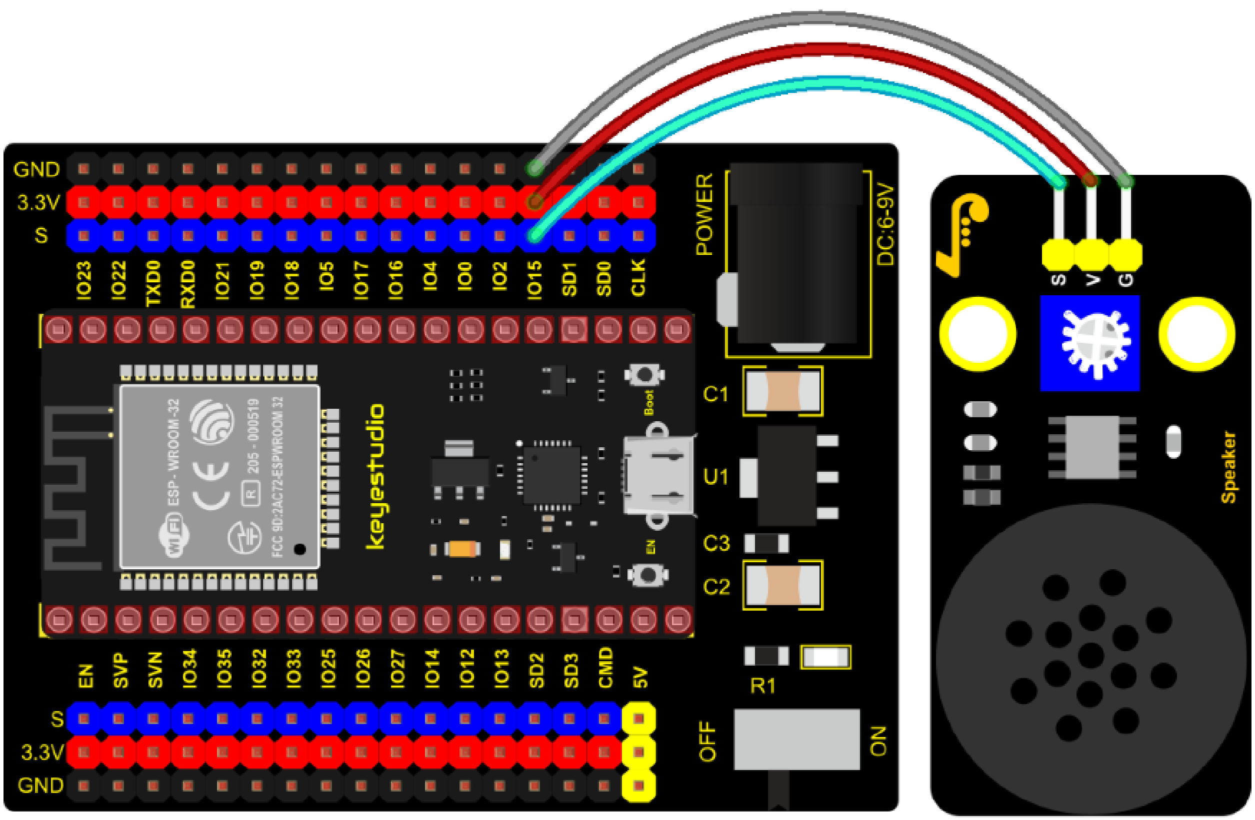
In fact, it is similar to a passive buzzer. The active buzzer has its own oscillation source.Yet, the passive buzzer does not have internal oscillation. When controlling the circuit, we need to input square waves of different frequencies to the positive pole of the component and ground the negative pole to control the buzzer to chime sounds of different frequencies.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 8002b Audio Power Amplifier*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : Passive Buzzer
 * Description : Passive Buzzer sounds the alarm.
 * Auther      : http//www.keyestudio.com
*/
#define LEDC_CHANNEL_0 0

// LEDC timer uses 13 bit accuracy
#define LEDC_TIMER_13_BIT  13

// Define tool I/O ports
#define BUZZER_PIN  15

//Create a musical melody list, Super Mario
int melody[] = {330, 330, 330, 262, 330, 392, 196, 262, 196, 165, 220, 247, 233, 220, 196, 330, 392, 440, 349, 392, 330, 262, 294, 247, 262, 196, 165, 220, 247, 233, 220, 196, 330, 392,440, 349, 392, 330, 262, 294, 247, 392, 370, 330, 311, 330, 208, 220, 262, 220, 262, 294, 392, 370, 330, 311, 330, 523, 523, 523, 392, 370, 330, 311, 330, 208, 220, 262,220, 262, 294, 311, 294, 262, 262, 262, 262, 262, 294, 330, 262, 220, 196, 262, 262,262, 262, 294, 330, 262, 262, 262, 262, 294, 330, 262, 220, 196};

//Create a list of tone durations
int noteDurations[] = {8,4,4,8,4,2,2,3,3,3,4,4,8,4,8,8,8,4,8,4,3,8,8,3,3,3,3,4,4,8,4,8,8,8,4,8,4,3,8,8,2,8,8,8,4,4,8,8,4,8,8,3,8,8,8,4,4,4,8,2,8,8,8,4,4,8,8,4,8,8,3,3,3,1,8,4,4,8,4,8,4,8,2,8,4,4,8,4,1,8,4,4,8,4,8,4,8,2};
void setup() {
pinMode(BUZZER_PIN, OUTPUT); // Set the buzzer to output mode
}

void loop() {

  int noteDuration; //Create a variable of noteDuration

  for (int i = 0; i < sizeof(noteDurations); ++i)

  {
      noteDuration = 800/noteDurations[i];

      ledcSetup(LEDC_CHANNEL_0, melody[i]*2, LEDC_TIMER_13_BIT);

      ledcAttachPin(BUZZER_PIN, LEDC_CHANNEL_0);

      ledcWrite(LEDC_CHANNEL_0, 50);

      delay(noteDuration * 1.30); //delay
  }
}
//**********************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power onthen the power amplifier module will emit the sound on a loop.

7.5.19 Project 19: 130 Motor

Description

The 130 motor driver module is compatible with servo motors, which has high efficiency and good quality fans.

It adopts a HR1124S motor control chip. HR1124S is a single-channel H-bridge driver chip for DC motor solutions. In addition, this chip has low standby current and low quiescent current.

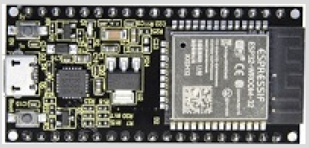
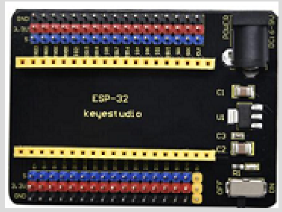
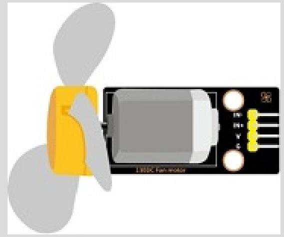
The module is compatible with various single-chip control boards. In the experiment, we can control the rotation direction of the motor by outputting the voltage directions of the two signal terminals IN+ and IN- to make the motor rotate.

Working Principle

The chip is used to help drive the motor. We can't drive it with a triode or an IO port due to its a large current of need. It is very simple to make the motor rotate. Just apply voltage to both ends of the motor. The direction of the motor is different in different voltage directions. Within the rated voltage, the higher the voltage, the faster the motor rotates; on the contrary, the lower the voltage, the slower the motor rotates, or even unable to rotate.

So we can use the PWM port to control the speed of the motor. We haven't learned PWM here, so we use the high and low levels to control the motor first.

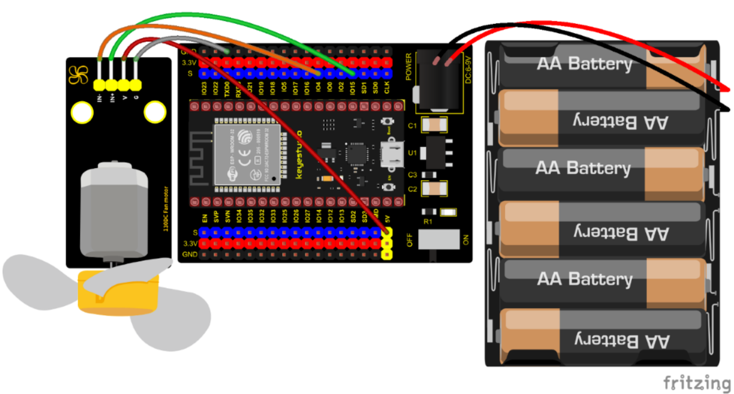
Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY 130 Motor*1	4P Dupont Wire*1
			
Micro USB Cable*1	Battety Holder*1	Battety (provide for yourself)*6	

Note: the motor is separated with its fan, you need to assemble it first.

Connection Diagram

130 Motor	ESP32 Expansion Board
G	G
V	5V
IN+	IO15
IN-	IO4



Test Code

```
//**********************************************************************************
/*
 * Filename    : 130DC Fan motor
 * Description : Motor positive and negative rotation
```

(continues on next page)

(continued from previous page)

```
 * Auther      : http://www.keyestudio.com
*/
//Define two pins interfaces of the motor, respectively 15 and 4
int INA = 15; //INA corresponds to IN+
int INB = 4;  //INB corresponds to IN-
void setup() {
  //Set the motor pins as output
  pinMode(INA, OUTPUT);
  pinMode(INB, OUTPUT);
}

void loop() {
  //Turn counterclockwise
  digitalWrite(INA, HIGH);
  digitalWrite(INB, LOW);
  delay(2000);
  //stop
  digitalWrite(INA, LOW);
  digitalWrite(INB, LOW);
  delay(1000);
  //clockwise rotation
  digitalWrite(INA, LOW);
  digitalWrite(INB, HIGH);
  delay(2000);
  //stop
  digitalWrite(INA, LOW);
  digitalWrite(INB, LOW);
  delay(1000);
}
//*************************************************************************************
```

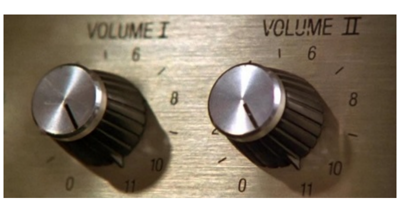
Code Explanation

Set pins to GPIO4GPIO15, when the pin GPIO4 outputs low levels and the pin GPIO15 outputs high levels, the motor will rotate counterclockwise; when both pins are set to low, the motor stops rotating.

Test Result

Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch ON the ESP32 expansion board to the ON end, after powering on, compile and upload the code to the ESP32. After uploading successfullythe fan will rotate counterclockwise for 2 seconds, stop for 1 second and clockwise for 2 seconds and stop for 1 second; cycle alternately.

7.5.20 Project 20: Potentiometer

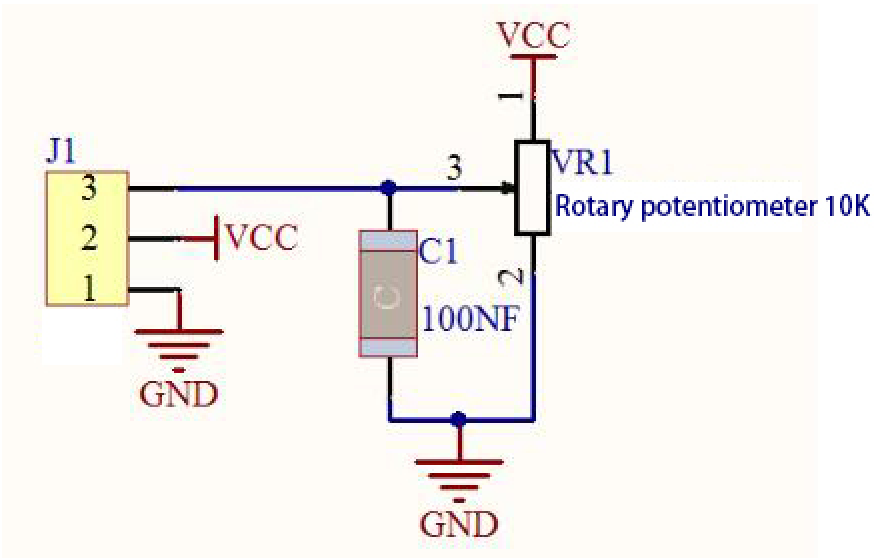


Overview

The following we will introduce is the Keyestudio rotary potentiometer which is an analog sensor.

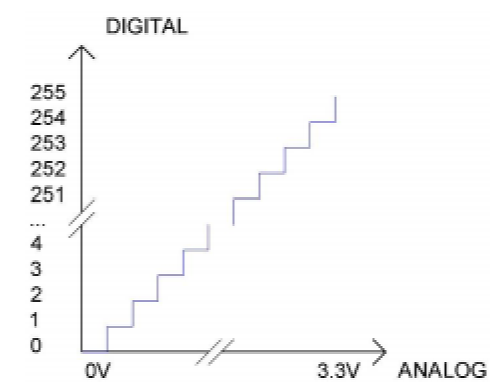
The digital IO ports can read the voltage value between 0 and 3.3V and the module only outputs high levels. However, the analog sensor can read the voltage value through 16 ADC analog ports on the ESP32 board. In the experiment, we will display the test results on the Shell.

Working Principle



It uses a 10K adjustable resistor. We can change the resistance by rotating the potentiometer. The signal S can detect the voltage changes(0-3.3V) which are analog quantity.

ADC The more bits an ADC has, the denser the partitioning of the simulation, the higher the accuracy of the final conversion.



Section 1: 0V – 3.3/4095 V analog quantity corresponding to digital 0;

Section 2: Analog quantities in the range 3.3/4095V – 2* 3.3/4095V correspond to digital 1;

...

The conversion formula is as follows:

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

DAC The higher the precision of DAC, the higher the precision of the output voltage value.

The conversion formula is as follows:

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

ADC on ESP32

The ESP32 has 16 pins that can be used to measure analog signals. GPIO pin serial numbers and analog pin definitions are shown below:

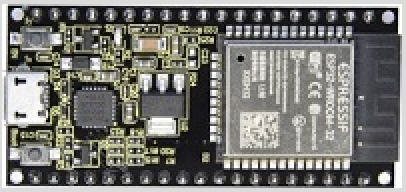
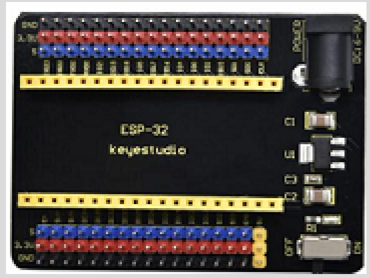
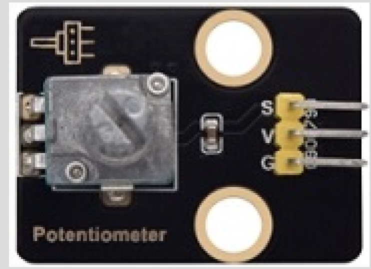
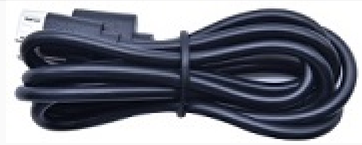
ADC number in ESP32	ESP32 GPIO number
ADC0	GPIO 36
ADC3	GPIO 39
ADC4	GPIO 32
ADC5	GPIO33
ADC6	GPIO34
ADC7	GPIO 35
ADC10	GPIO 4
ADC11	GPIO0
ADC12	GPIO2
ADC13	GPIO15
ADC14	GPIO13
ADC15	GPIO 12
ADC16	GPIO 14
ADC17	GPIO27
ADC18	GPIO25
ADC19	GPIO26

DAC on ESP32

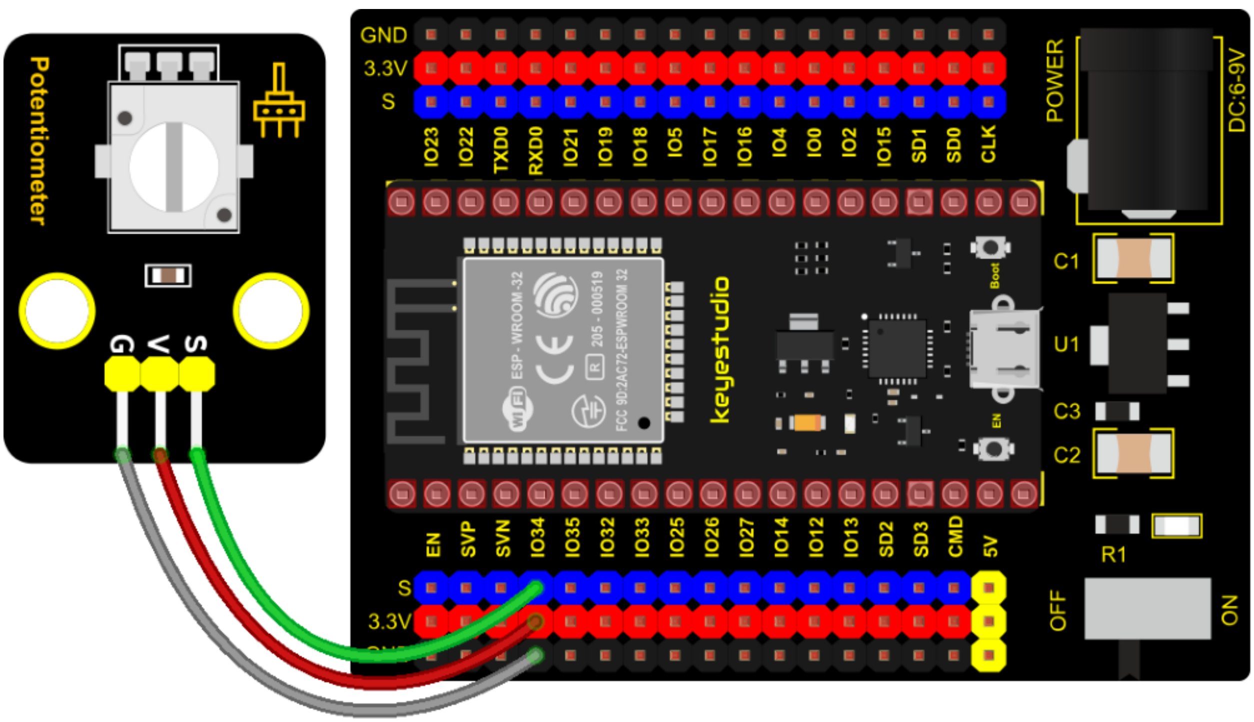
The ESP32 has two 8-bit digital-to-analog converters connected to GPIO25 and GPIO26 pins, which are immutable, as shown below :

Simulate pin number	GPIO number
DAC1	GPIO25
DAC2	GPIO26

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Potentiometer*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Rotary_potentiometer
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
```

(continues on next page)

(continued from previous page)

```
#define PIN_ANALOG_IN  34  //the pin of the Potentiometer

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal,␣
↪voltage);
  delay(200);
}
//*********************************************************************************
```

Code Explanation

1). analogVal means analog value. The rotary potentiometer outputs analog values(0~4095), therefore, we set pins to analog ports. For example, we connect to GPIO34.

2). analogRead(pin): read the value of the specified analog pin. The ESP32 contains a multi-channel, 12-bit converter. This means that it will map the input voltage between 0 and the working voltage (5V or 3.3V) to an integer value between 0 and 4095. For example, this will produce a resolution among readings: 3.3V/4096 stands for 0.0008V per unit.

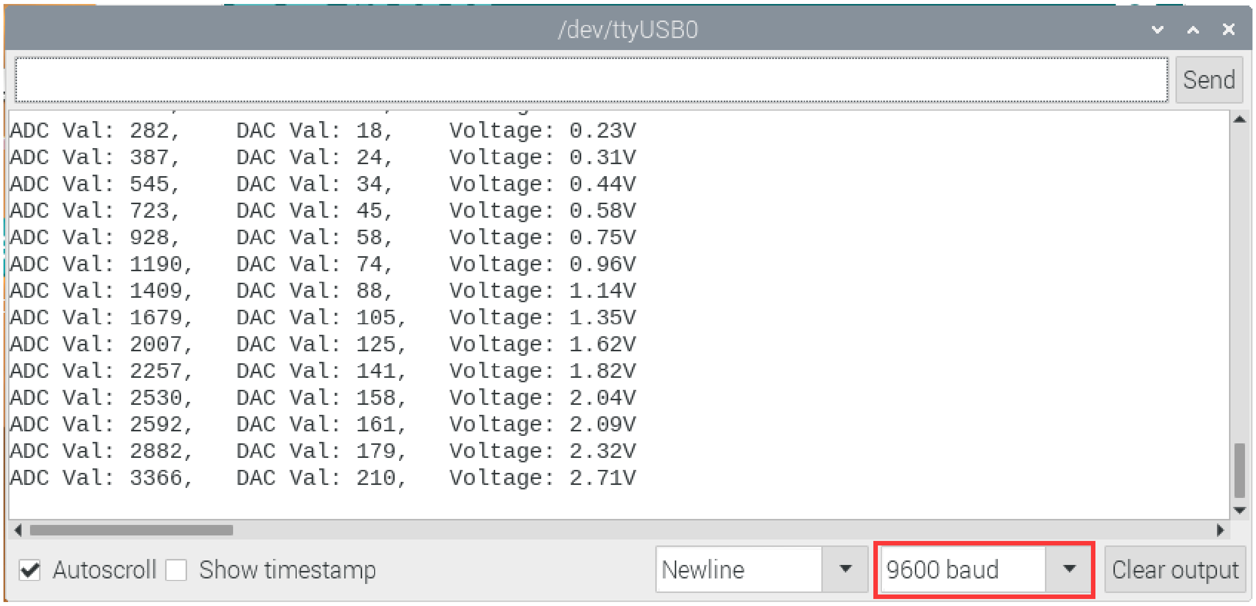
3). The map() function converts this 12-bit DAC value to an 8-bit DAC value.

4). Pin: the name of analog input pin.

5). The serial monitor displays the values of adcVal, dacVal, voltage, the baud rate must be set before display (we default to 9600,which can be changed).

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the potentiometer's ADC value, DAC value and voltage value. Rotate the potentiometer handle, the analog values will change.



7.5.21 Project 21: Steam Sensor

Description

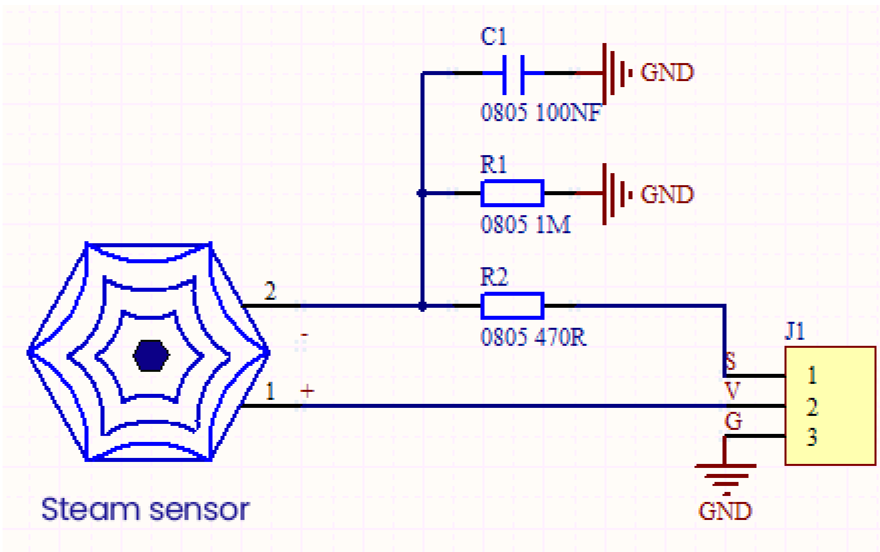
This is a DIY electronic building block water drop sensor. It is an analog (digital) input module, also called rain, rain sensor. It can be used to monitor various weather conditions, detect whether it is raining and the amount of rain, convert it into digital signal (DO) and analog signal (AO) output, and is widely used in Arduino robot kits, raindrops, rain sensors, and can be used for various It can monitor various weather conditions, and convert it into digital signal and AO output, and can also be used for automobile automatic wiper system, intelligent lighting system and intelligent sunroof system.

In the experiment, we input the sensor signal terminal (S terminal) to the analog port of the ESP32 development board, sense the change of the analog value, and display the corresponding analog value on the shell.

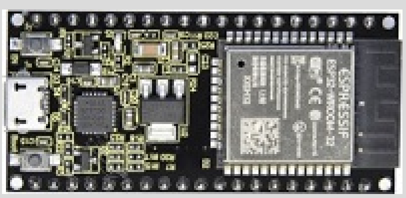
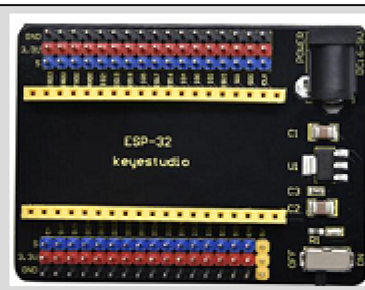
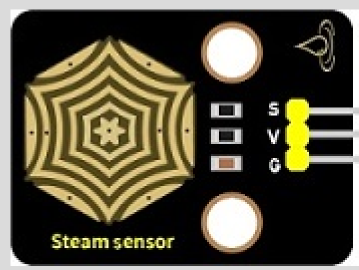
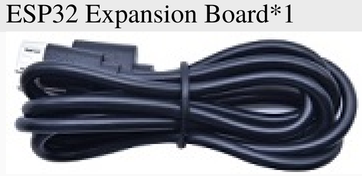
Working Principle

Its principle is to detect the amount of water through the exposed printed parallel lines on the circuit board. The more water there is, the more wires will be connected, and the conductive contact area increases. The voltage output by pin 2 will gradually increase. The larger the analog value detected by the signal terminal S is.

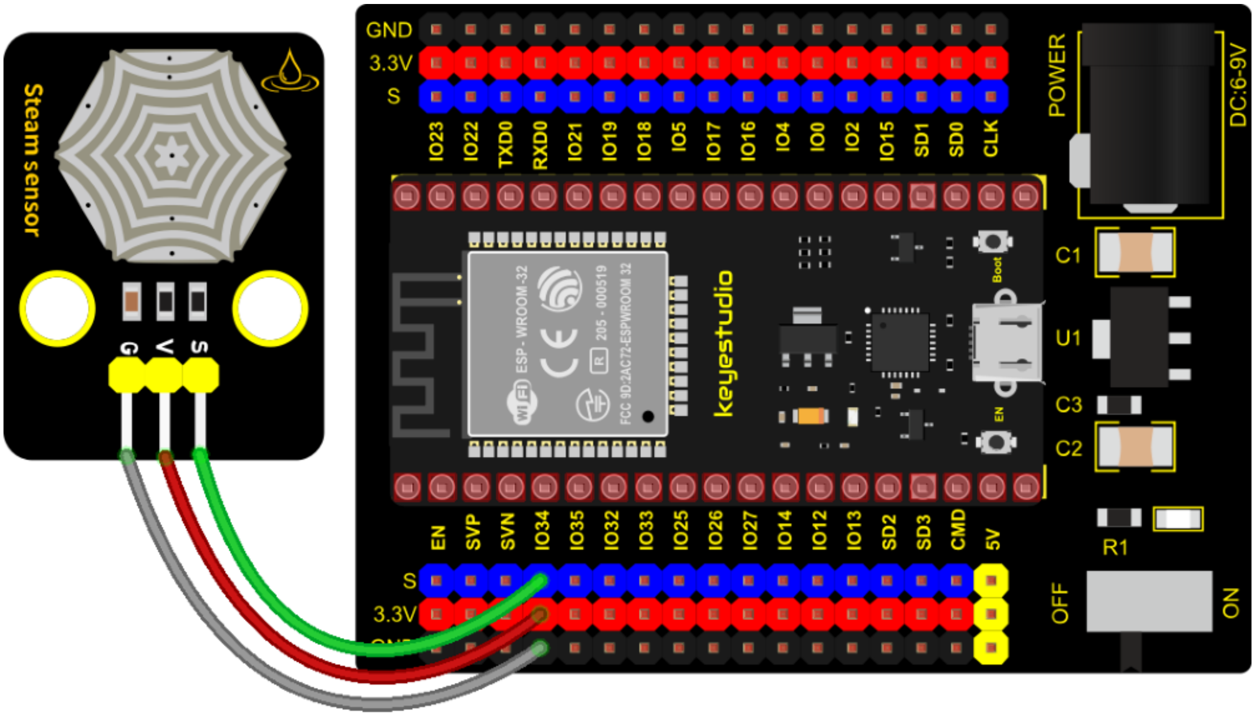
It can also detect steam in the air. Two position holes are used to install on the other devices.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Steam Sensor *1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram

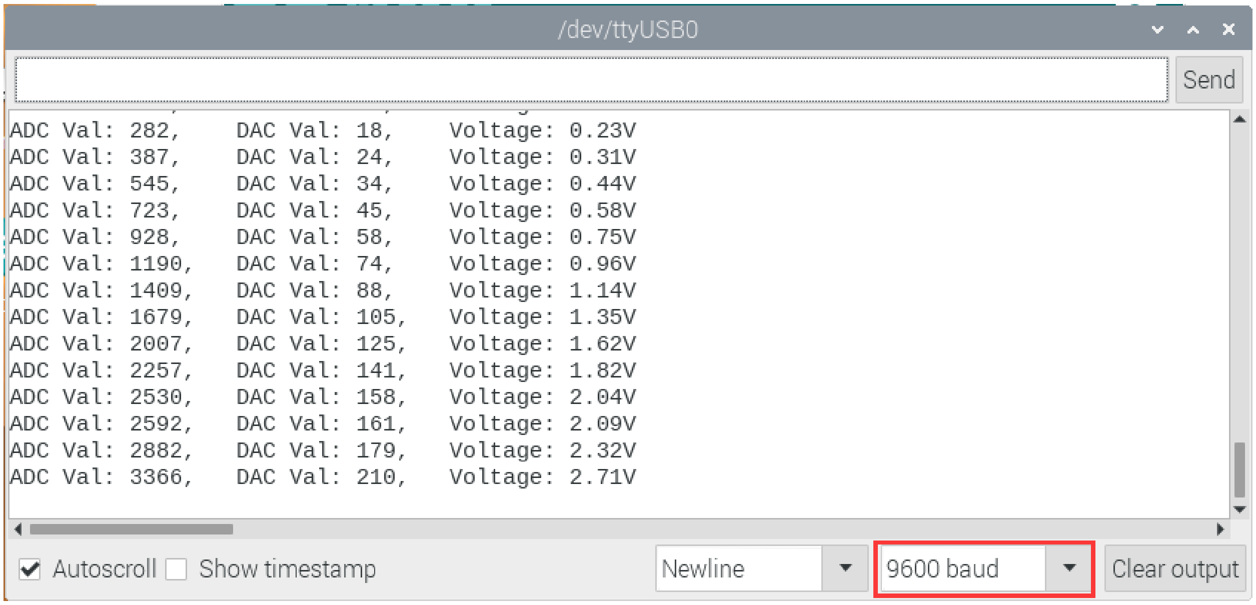


Test Code

```
//**********************************************************************************
/*
 * Filename    : Steam sensor
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN   34  //the pin of the Steam sensor

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the steam sensor's ADC value, DAC value and voltage value. When a few drops of water are placed in the sensor sensing area, the values will change. The more water volume, the greater the output voltage value , ADC value and the DAC value .



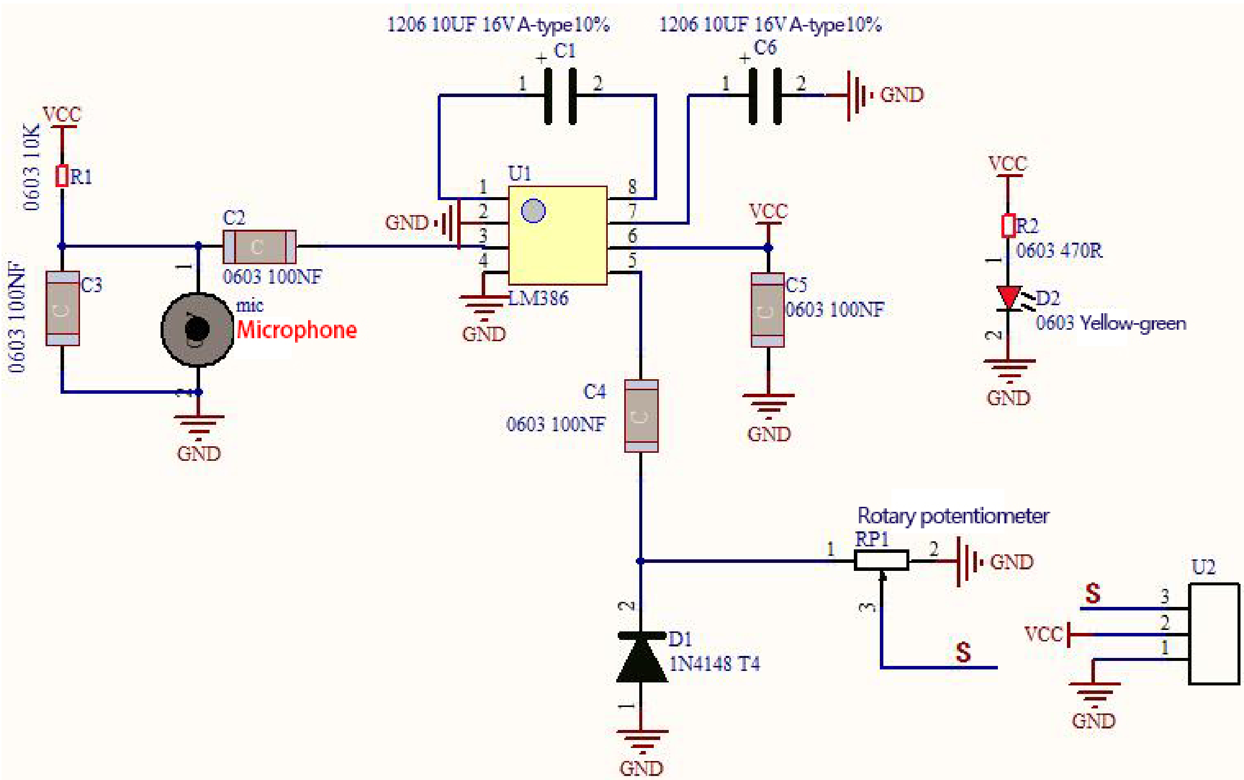
7.5.22 Project 22: Sound Sensor

Overview

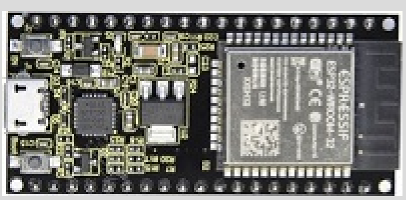
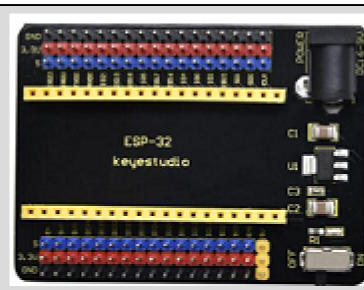
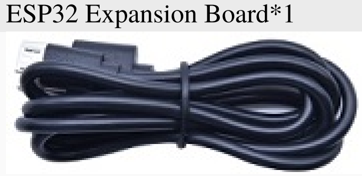
In this kit, there is a Keyestudio DIY electronic block and a sound sensor. In the experiment, we test the analog value corresponding to the sound level in the current environment with it. The louder the sound, the larger the ADC, DAC and the voltage value, and the "shell" window will display the test results.

Working Principle

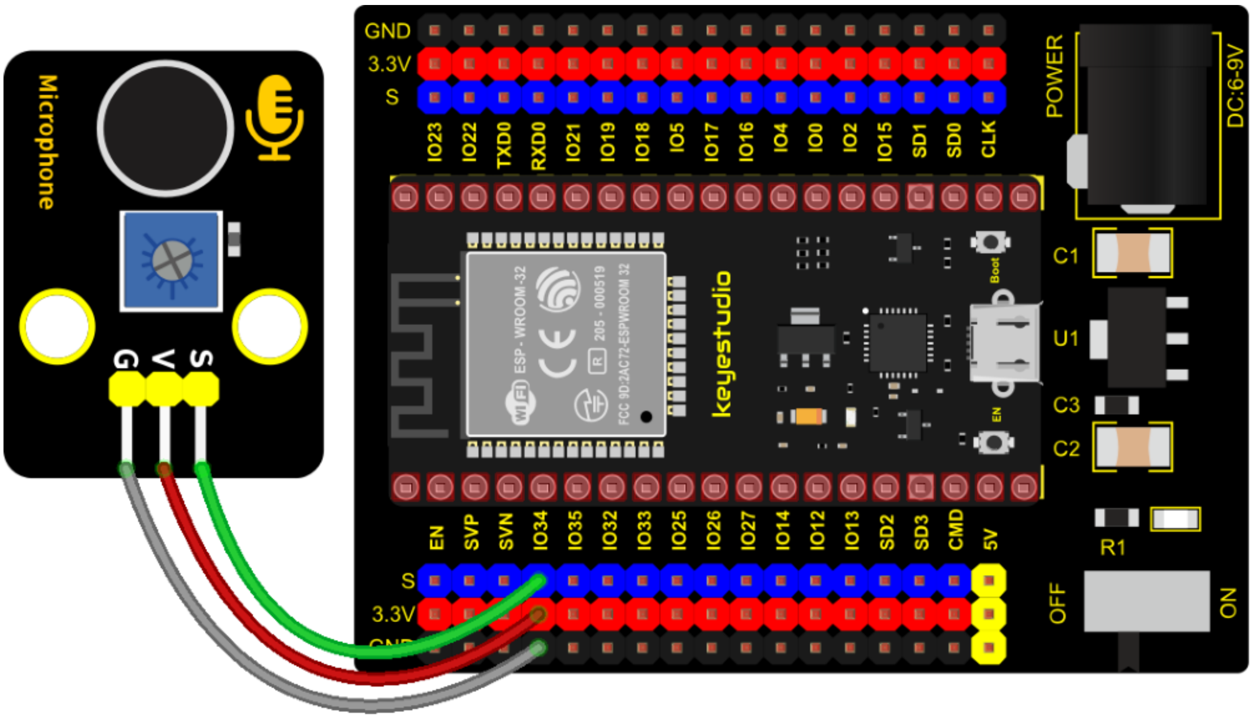
It uses a high-sensitive microphone component and an LM386 chip. We build the circuit with the LM386 chip and amplify the sound through the high-sensitive microphone. In addition, we can adjust the sound volume by the potentiometer. Rotate it clockwise, the sound will get louder.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Sound Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram

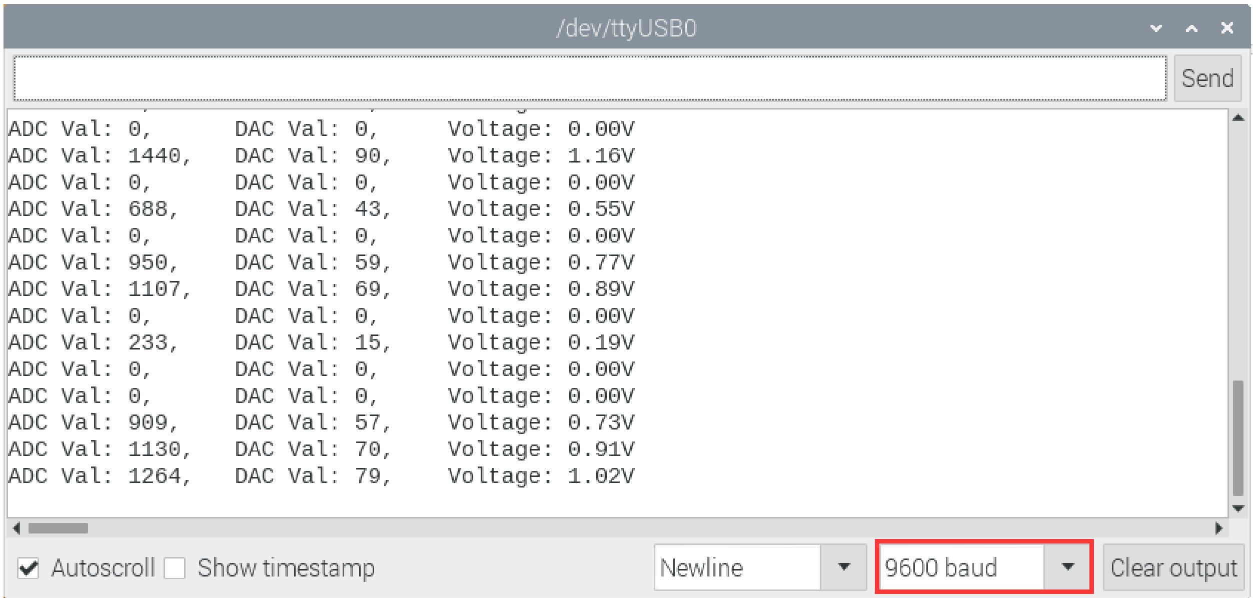


Test Code

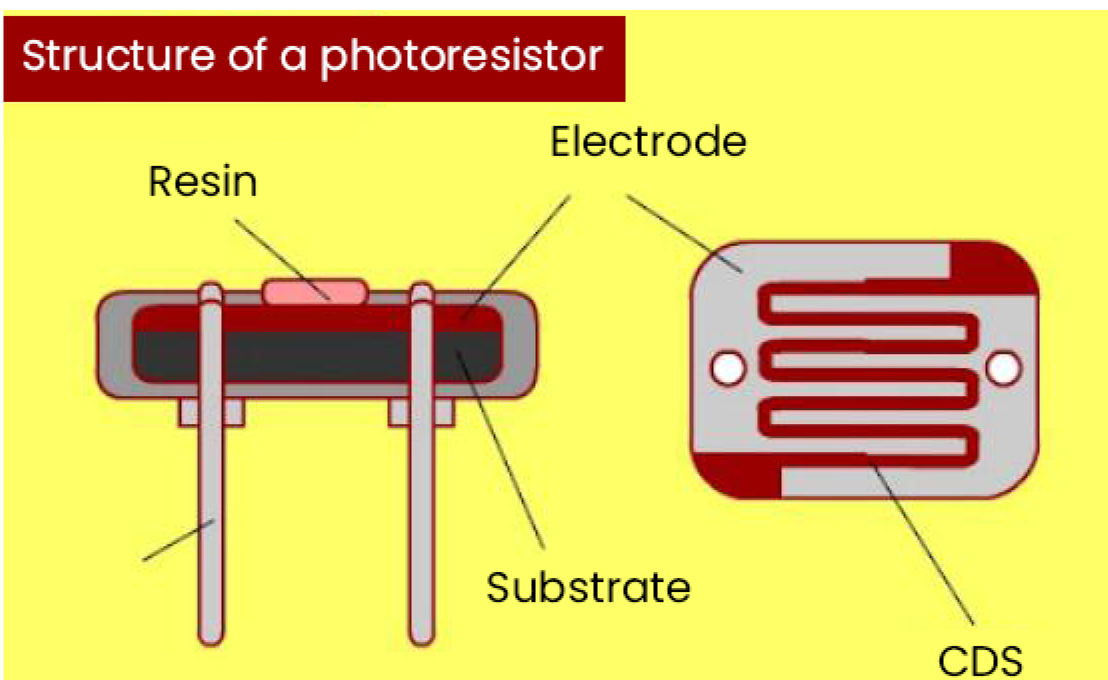
```
//**********************************************************************************
/*
 * Filename    : MicroPhone
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Sound Sensor

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal, voltage);
  delay(200);
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the sound sensor's ADC value, DAC value and voltage value. Rotate clockwise the potentiometer and speak at the MIC. Then you can see the analog value get larger, as shown below:



7.5.23 Project 23: Photoresistor

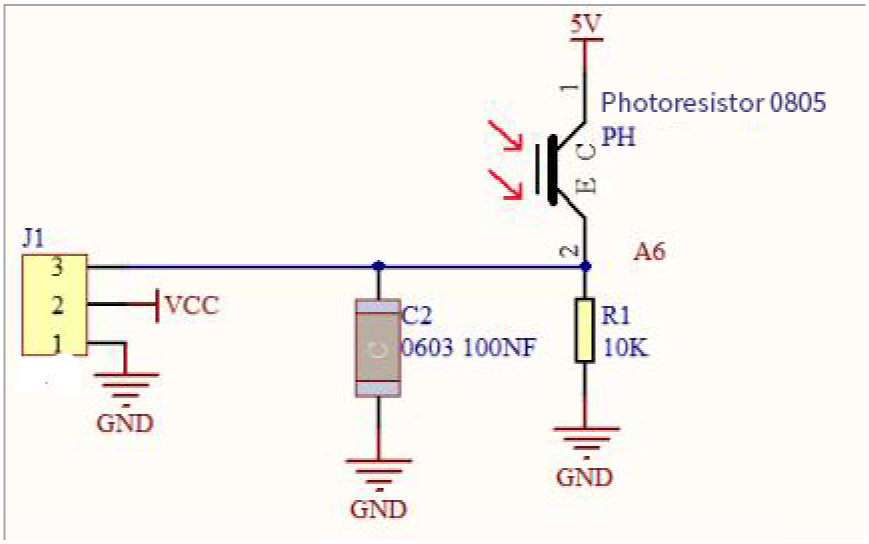


Description

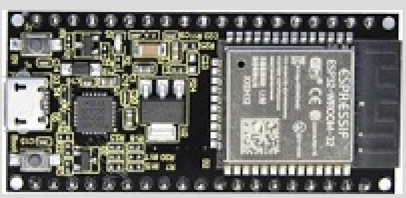
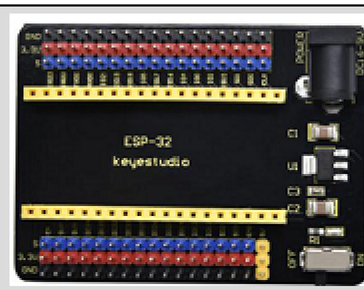
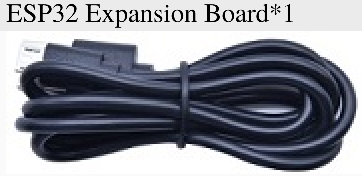
In this kit, there is a photoresistor which consists of photosensitive resistance elements. Its resistance changes with the light intensity. Also, it converts the resistance change into a voltage change through the characteristic of the photosensitive resistive element. When wiring it up, we interface its signal terminal (S terminal) with the analog port of ESP32 , so as to sense the change of the analog value, and display the corresponding analog value in the shell.

Working Principle

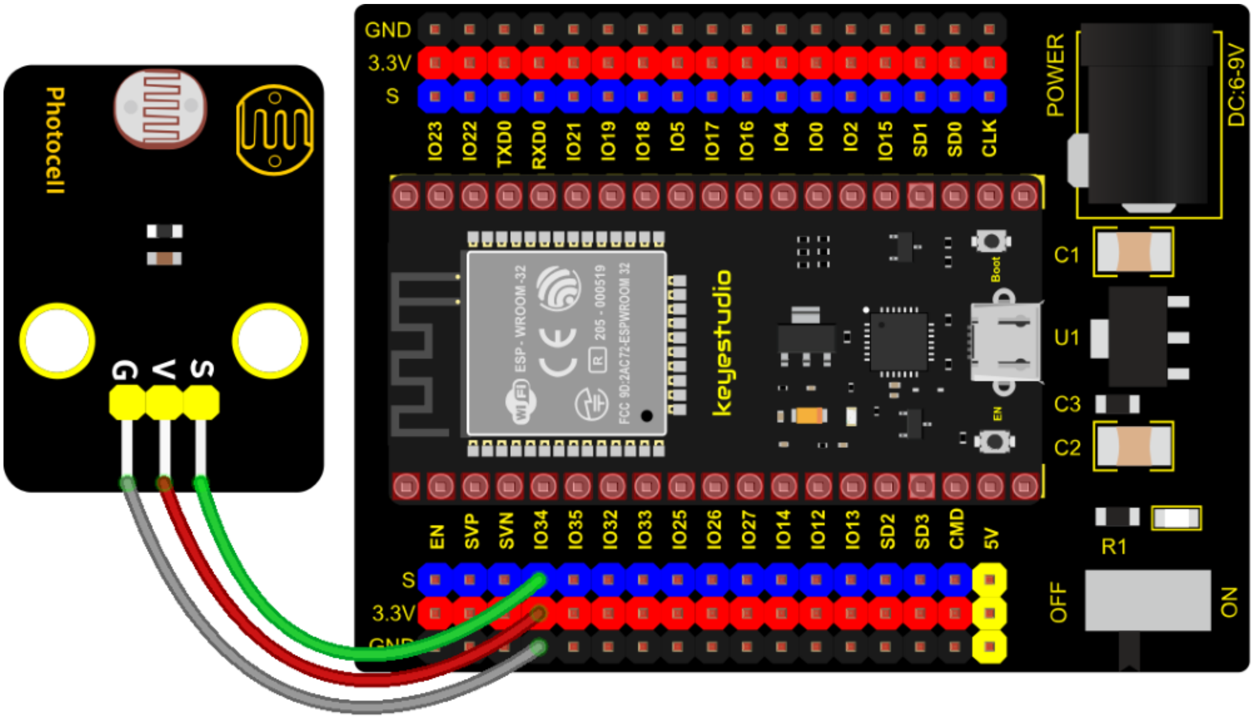
If there is no light, the resistance is 0.2M and the detected voltage at the terminal 2 is close to 0. When the light intensity increases, the resistance value of the light sensor is getting smaller and smaller, so the voltage detected at the signal end is getting larger and larger. . .



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Photoresistor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



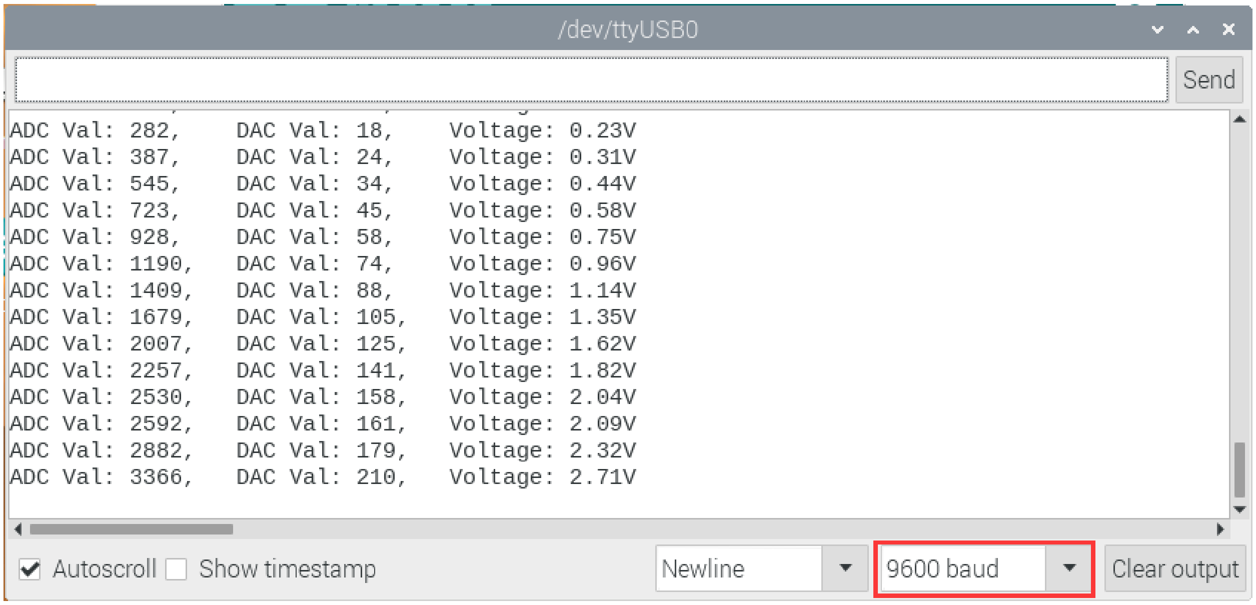
Test Code

```
//**********************************************************************************
/*
 * Filename    : Photoresistance
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Photoresistance

void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula, and the␣
↪information is finally printed out.
void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", adcVal, dacVal,␣
↪voltage);
  delay(200);
}
//**********************************************************************************
```

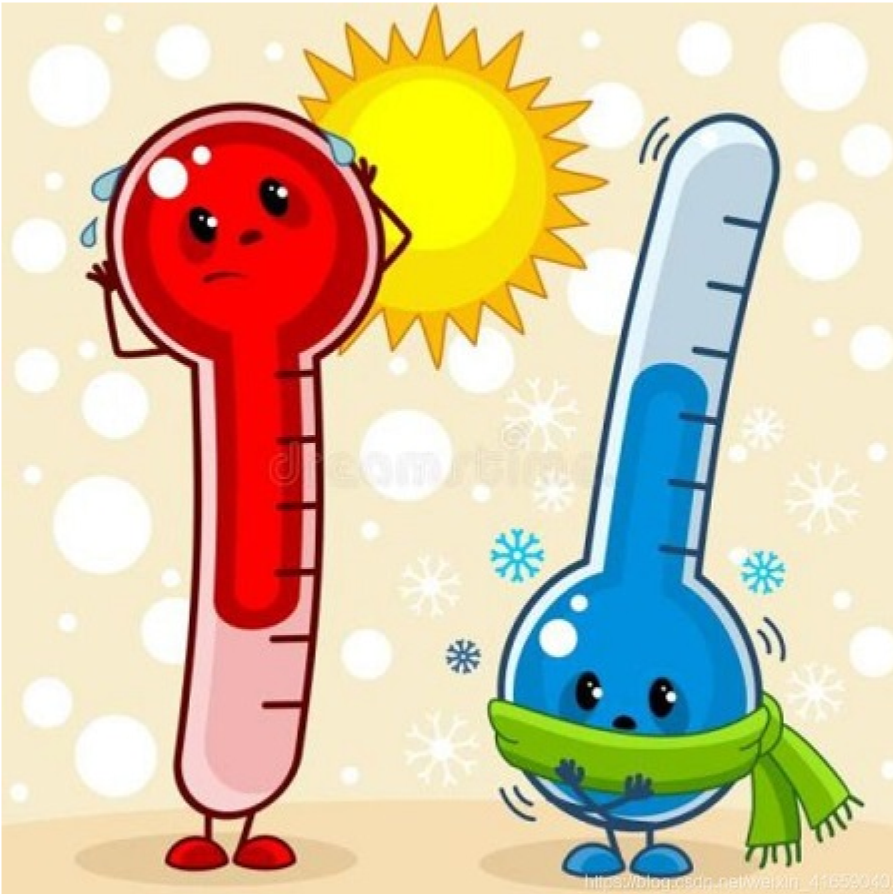
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After

uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the photoresistor's ADC value, DAC value and voltage value. When the light intensity gets stronger, the analog values will get larger, as shown below:



7.5.24 Project 24: NTC-MF52AT Thermistor

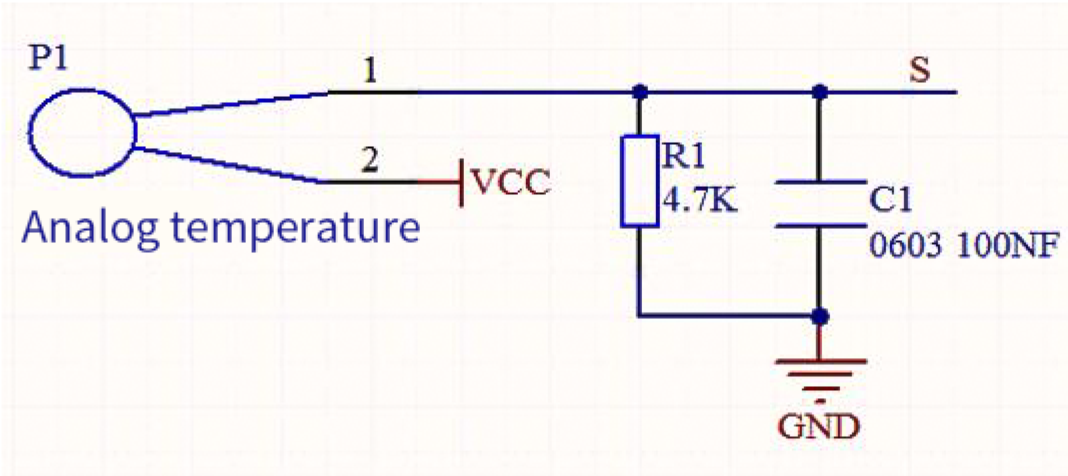


Overview

In the experiment, there is a NTC-MF52AT analog thermistor. We connect its signal terminal to the analog port of the ESP32 mainboard and read the corresponding ADC value, voltage value and thermistor value.

We can use analog values to calculate the temperature of the current environment through specific formulas. Since the temperature calculation formula is more complicated, we only read the corresponding analog value.

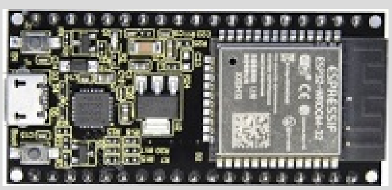
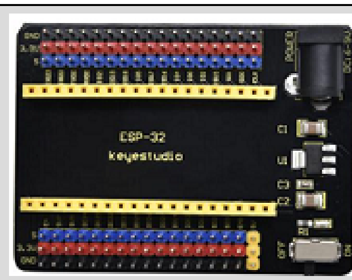
Working Principle



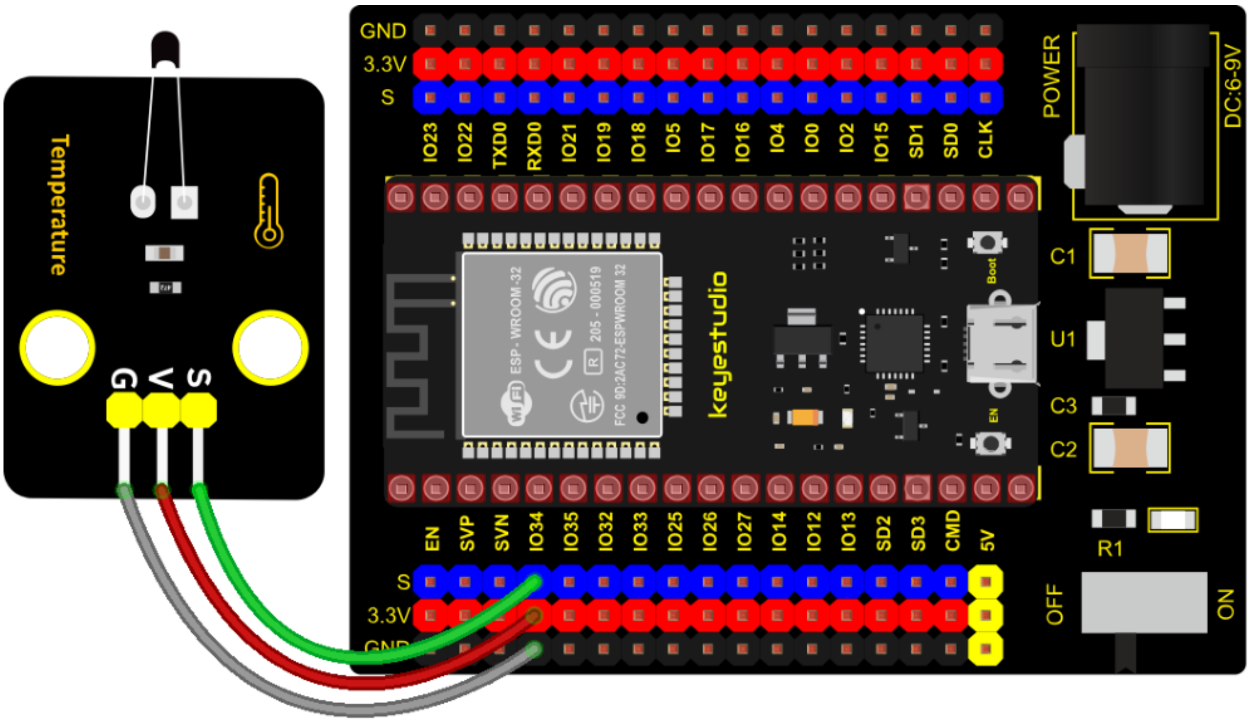
This module mainly uses NTC-MF52AT thermistor element, which can sense the changes of the surrounding environment temperature. Resistance changes with the temperature, causing the voltage of the signal terminal S to change.

This sensor uses the characteristics of NTC-MF52AT thermistor element to convert resistance changes into voltage changes.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio NTC-MF52AT Thermistor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



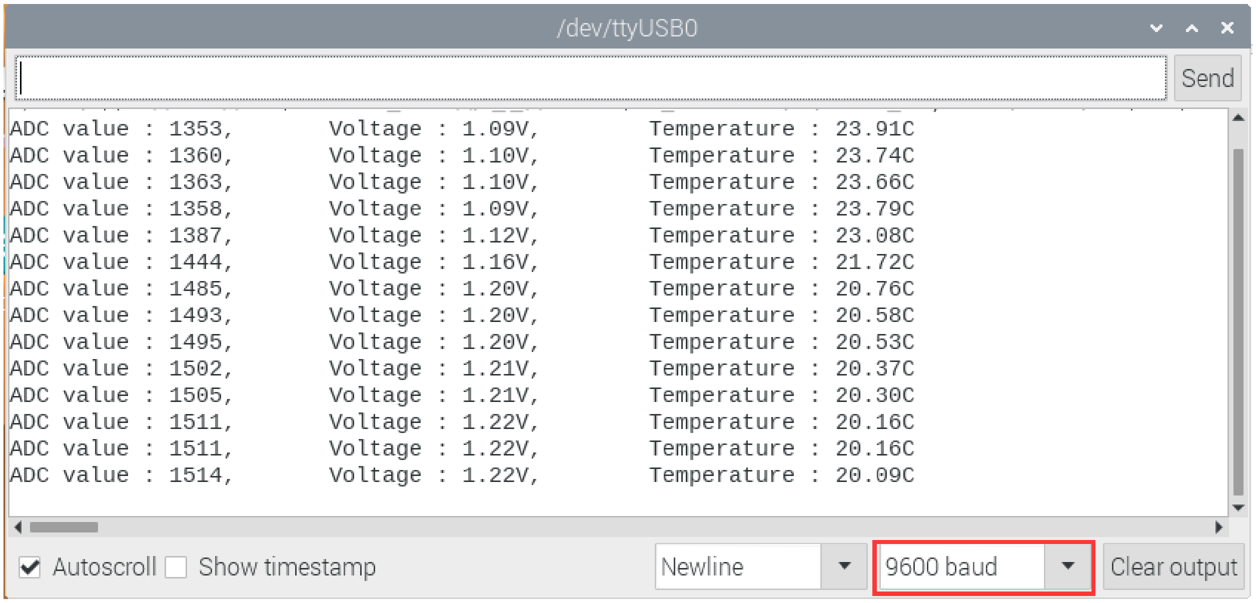
Test Code

```
//**********************************************************************************
/*
 * Filename    : Temperature sensor
 * Description : Making a thermometer by thermistor.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN   34
void setup() {
  Serial.begin(9600);
}

void loop() {
  int adcValue = analogRead(PIN_ANALOG_IN);                       //read ADC pin
  double voltage = (float)adcValue / 4095.0 * 3.3;                // calculate voltage
  double Rt = (3.3 - voltage) / voltage * 4.7;                     //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);
}
//**********************************************************************************
```

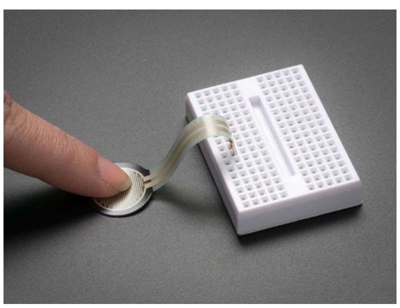
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After

uploading successfully, we will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the thermistor's ADC value, DAC value and voltage value, as shown below:



7.5.25 Project 25: Thin-film Pressure Sensor

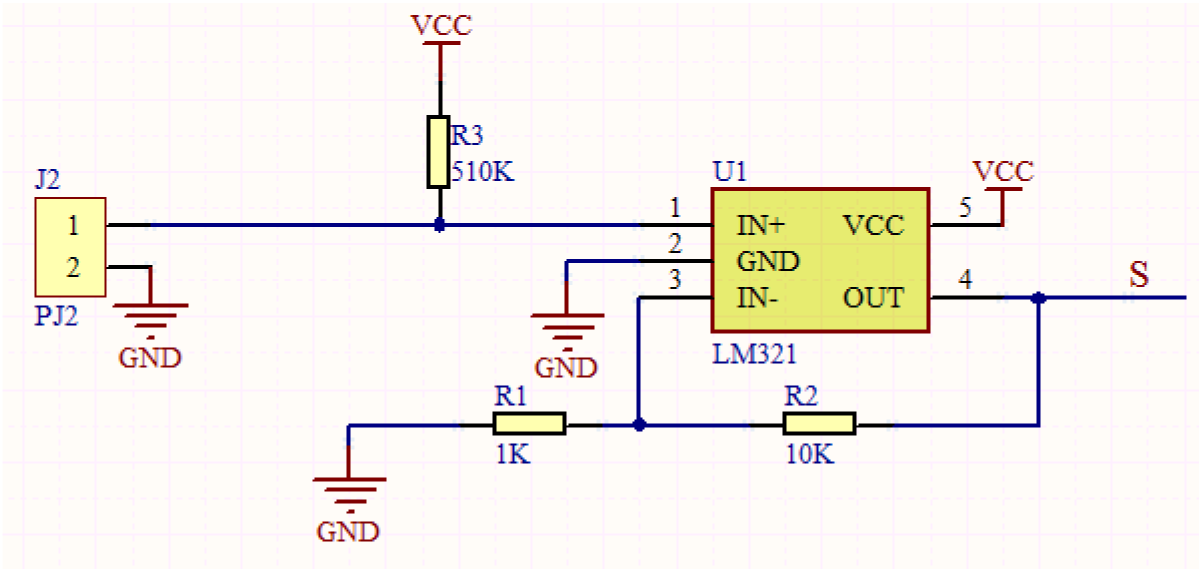


Overview

In this kit, there is a Keyestudio thin-film pressure sensor. The thin-film pressure sensor composed of a new type of nano pressure-sensitive material and a comfortable ultra-thin film substrate, has waterproof and pressure-sensitive functions.

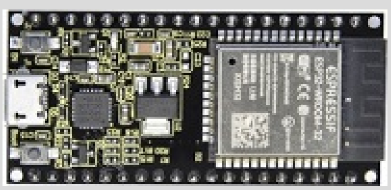
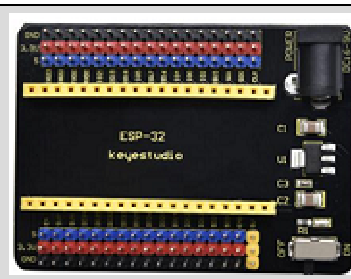
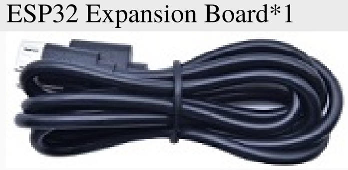
In the experiment, we determine the pressure by collecting the analog signal on the S end of the module. The smaller the ADC value, DAC value and voltage value, the greater the pressure; and the displayed results will shown on the Shell.

Working Principle

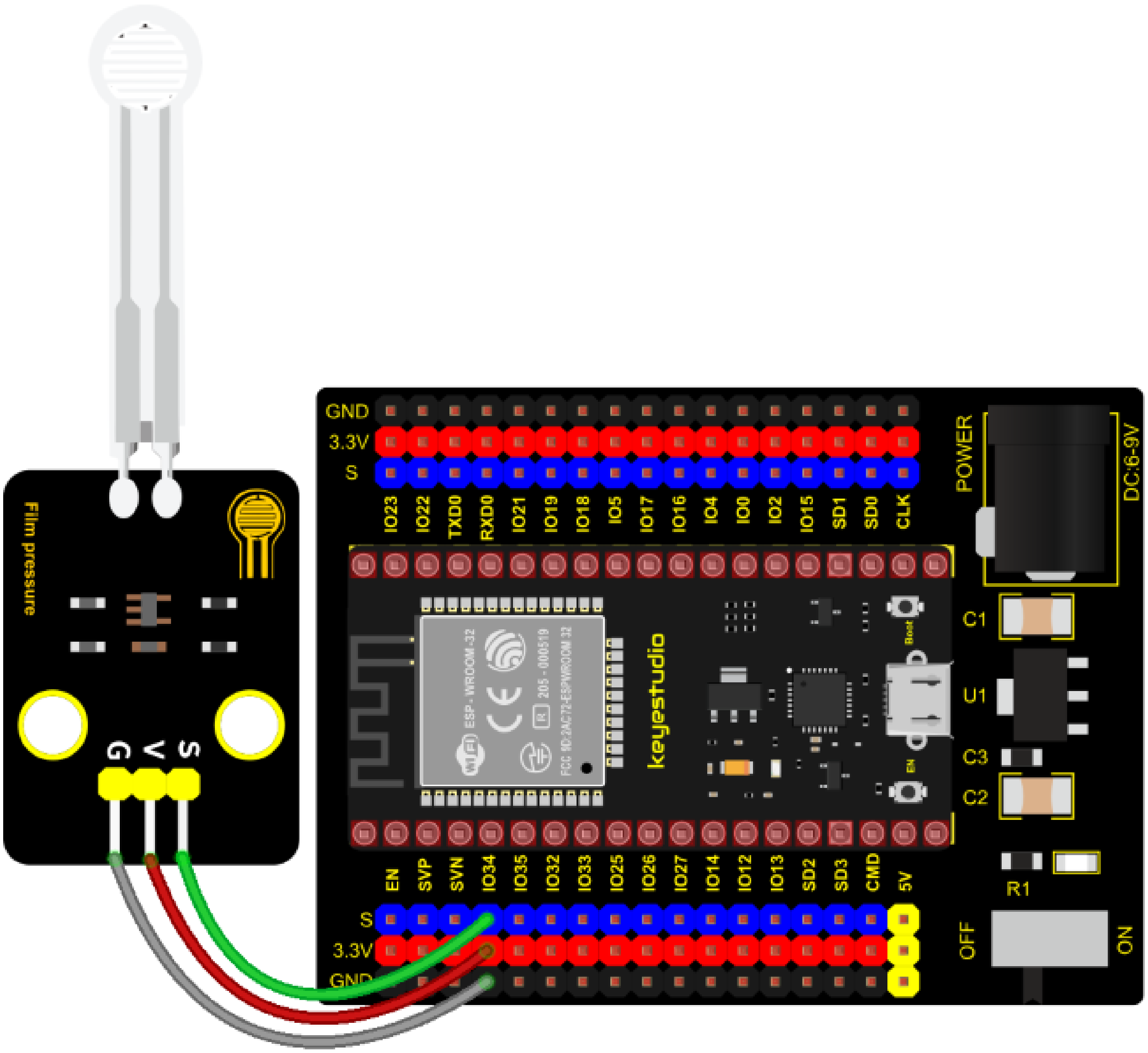


When the sensor is pressed by external forces, the resistance value of sensor will vary. We convert the pressure signals detected by the sensor into the electric signals through a circuit. Then we can obtain the pressure changes by detecting voltage signal changes.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	KeyestudioThin-film Pressure Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Film pressure sensor
 * Description : Read the basic usage of ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN  34  //the pin of the Film pressure sensor
void setup() {
  Serial.begin(9600);
}

//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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
```

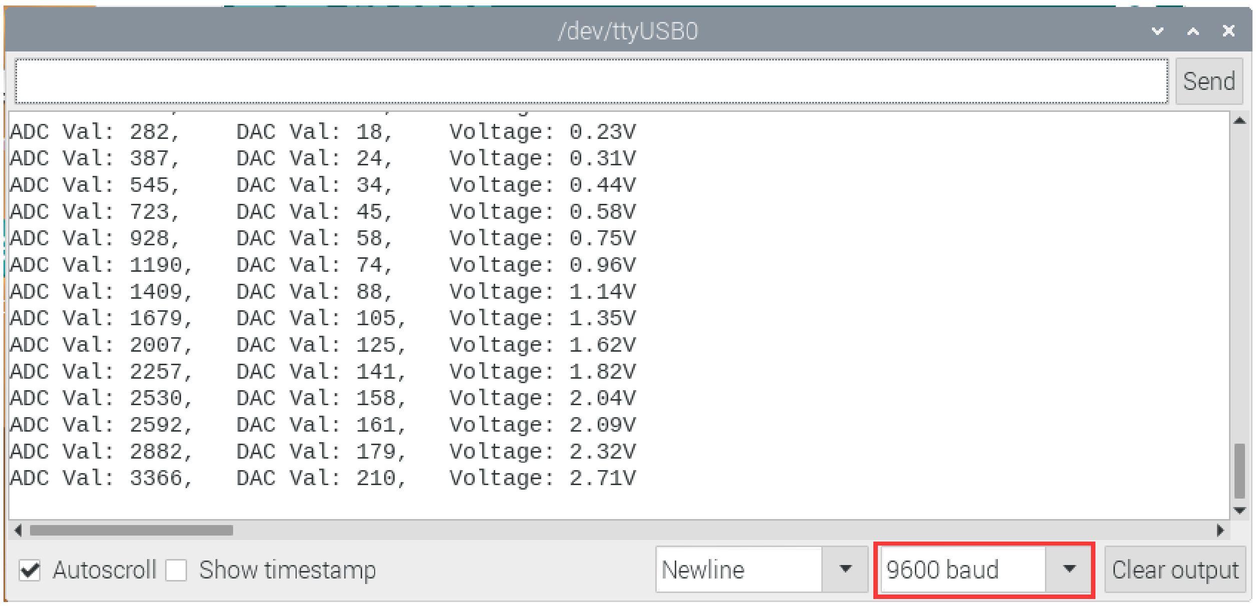
(continues on next page)

(continued from previous page)

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

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then the serial monitor will display the thin-film's ADC value, DAC value and voltage value, when the thin-film is pressed by fingers, the analog value will decrease, as shown below;



7.5.26 Project 26: Flame Sensor

Description

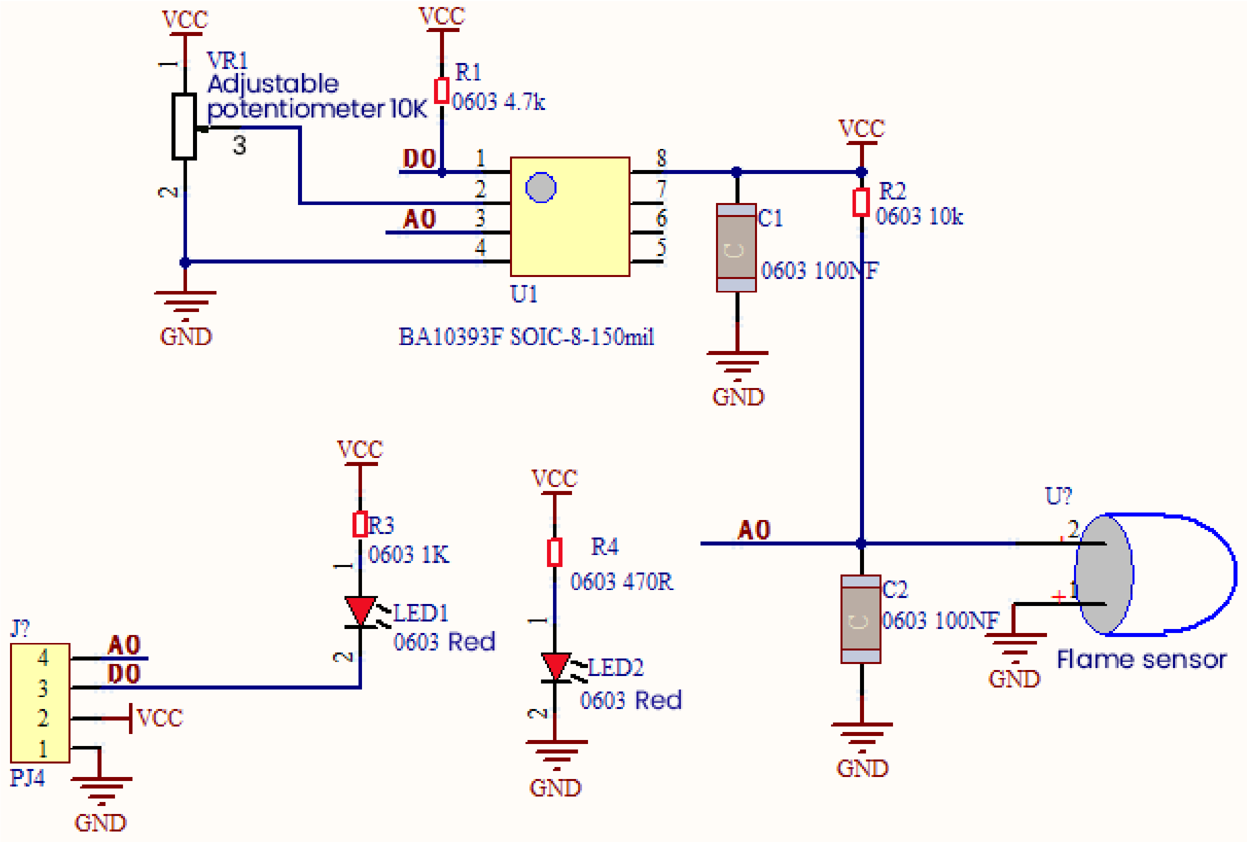
In daily life, it is often seen that a fire broke out without any precaution. It will cause great economic and human loss. So how can we avoid this situation? Right, install a flame sensor and a speaker in those places that easily break out a fire. When the flame sensor detects a fire, the speaker will alarm people quickly to put out the fire.

So in this project, you will learn how to use a flame sensor and an active buzzer module to simulate the fire alarm system.

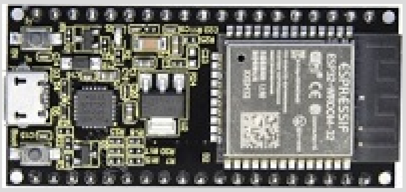
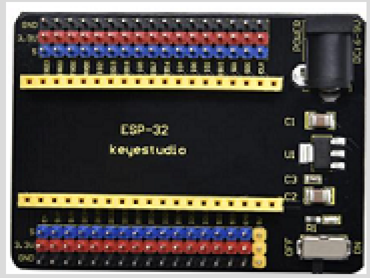
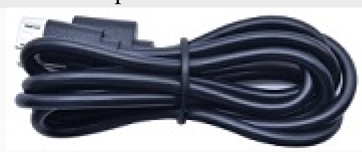
Working Principle

This flame sensor can be used to detect fire or other light sources with wavelength stands at 700nm ~ 1000nm. Its detection angle is about 60°. You can rotate the potentiometer on the sensor to control its sensitivity. Adjust the potentiometer to make the LED at the critical point between on and off state. The sensitivity is the best.

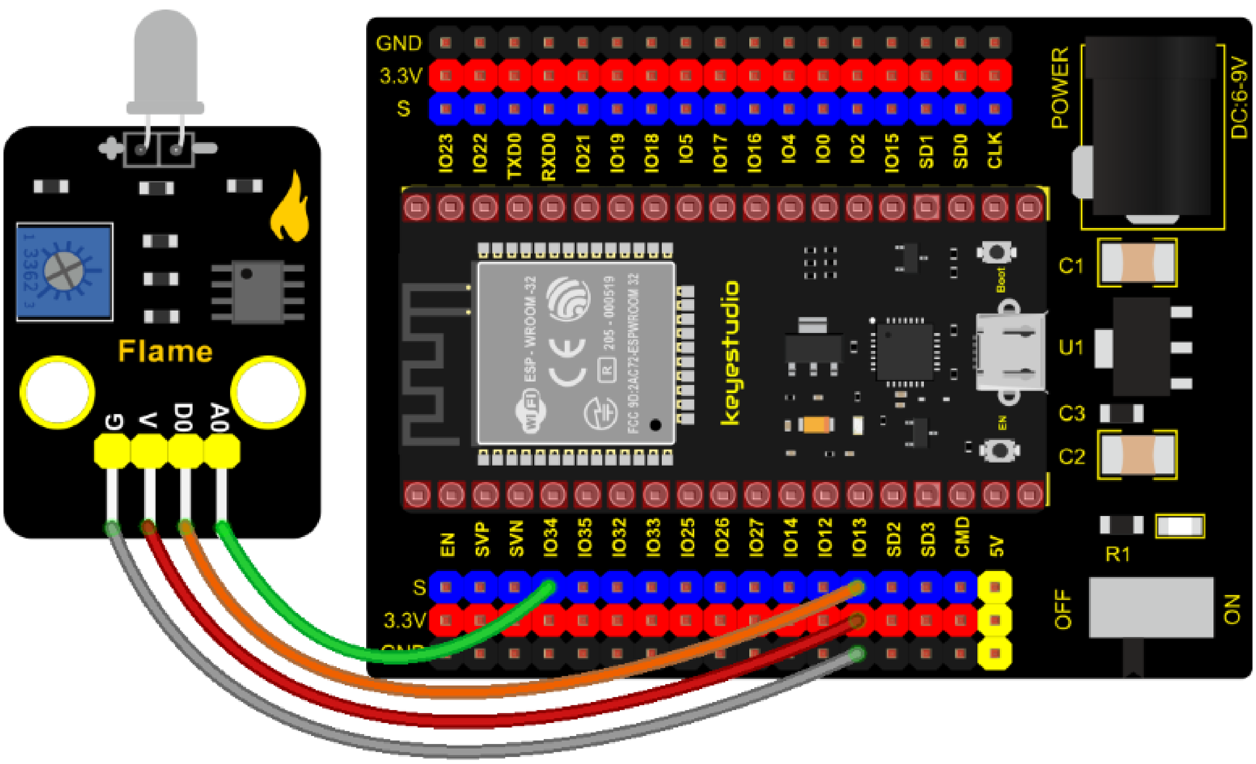
From the below figure, power up. When detecting fire, the digital pin outputs low levels, the red LED2 will light up first, the digital signal terminal D0 outputs a low level, and the red LED1 will light up. The stronger the external infrared light, the smaller the value; the weaker the infrared light, the larger the value.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Flame Sensor*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Flame sensor
 * Description : Read the basic usage of DigitalADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
```

(continues on next page)

(continued from previous page)

```
*/
//Flame sensor two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin = 13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value,
//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 input and output voltage are calculated according to the previous formula,
//and the information is finally printed out.
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n",
↪digitalVal, adcVal, dacVal, voltage);
  delay(200);
}
//*************************************************************************************
```

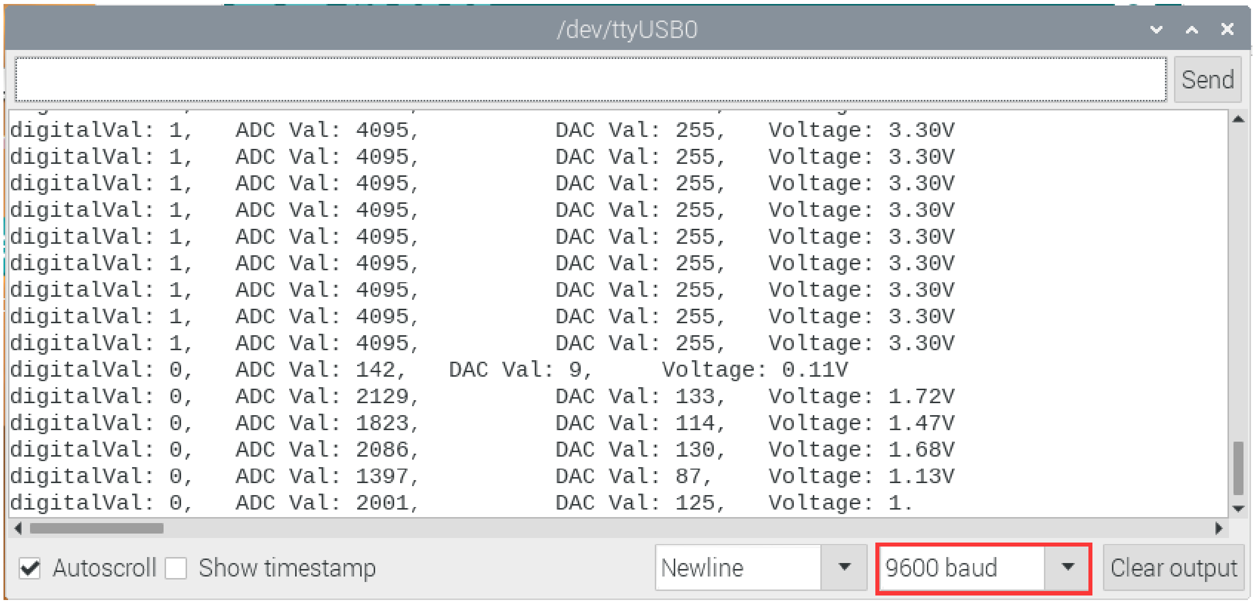
Code Explanation

Two pins we use are defined as GPIO13 and GPIO34 according to the wiring-up diagram, and print digital signals and analog signals respectively.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point. The red LED2 on the sensor module is lit, while the red LED1 is not. Open the monitor and set baud rate to 9600.

We need to press the reset button on the ESP32, then the "Shell" window will display the digital value, ADC value, DAC value and voltage value of the flame sensor. When fire is detected, the LED1 will be on. the digital value will change from 1 to 0, and the analog value will become smaller, as shown below.



7.5.27 Project 27: MQ-2 Gas Sensor

Description

This analog gas sensor - MQ2 is used in gas leakage detecting equipment in consumer electronics and industrial markets.

This sensor is suitable for detecting LPG, I-butane, propane, methane, alcohol, Hydrogen and smoke. It has high sensitivity and quick response.

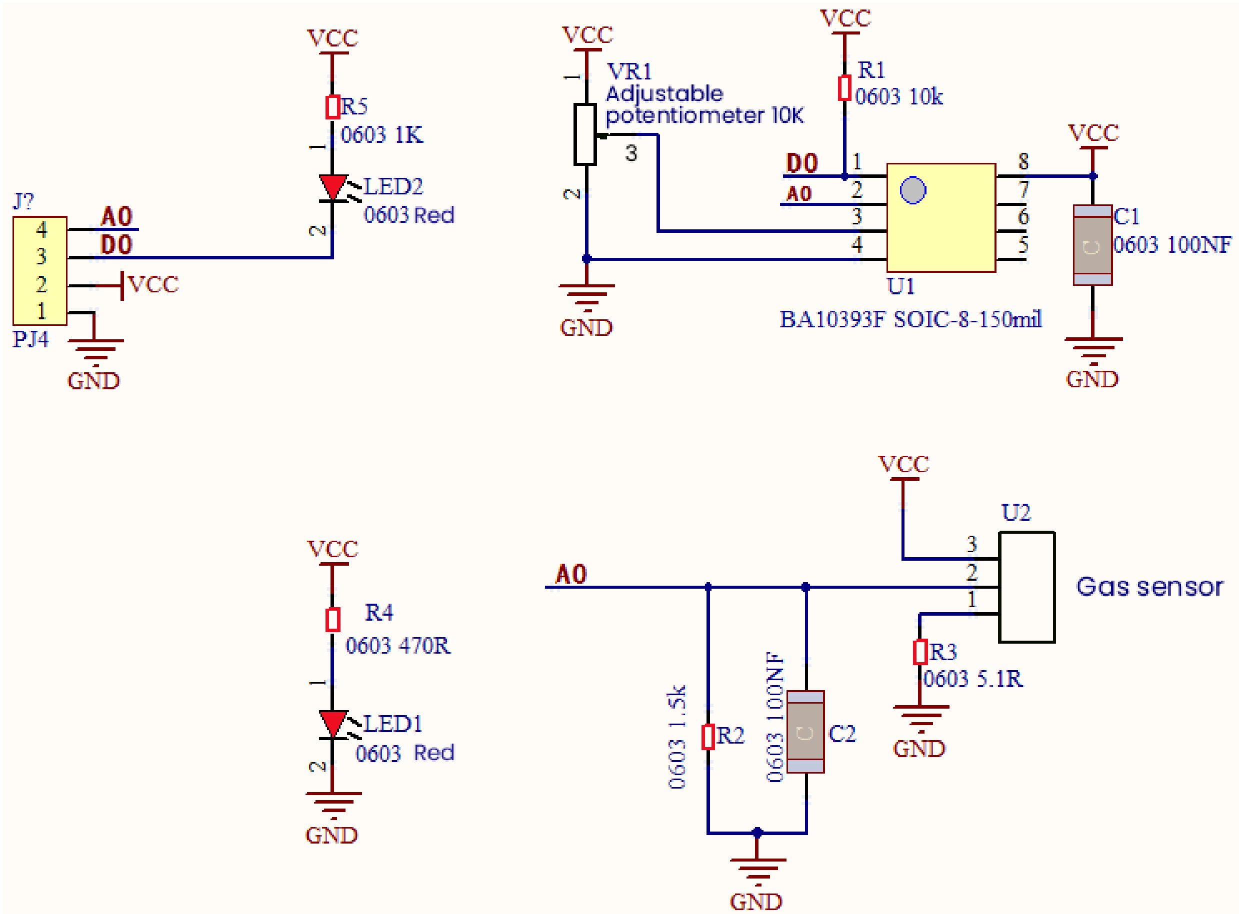
In addition, the sensitivity can be adjusted by rotating the potentiometer.

In the experiment, we read the analog value at the A0 port and the D0 port to determine the content of gas.

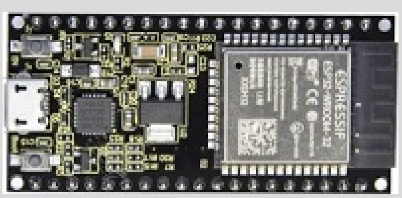
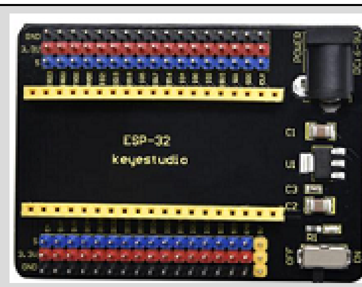
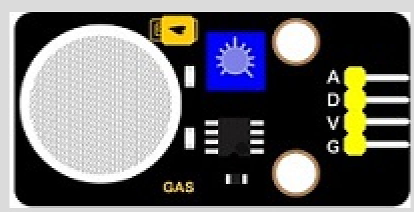
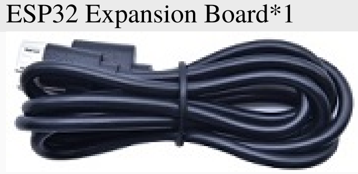
Working Principle

The greater the concentration of smoke, the greater the conductivity, the lower the output resistance, the greater the output analog signal.

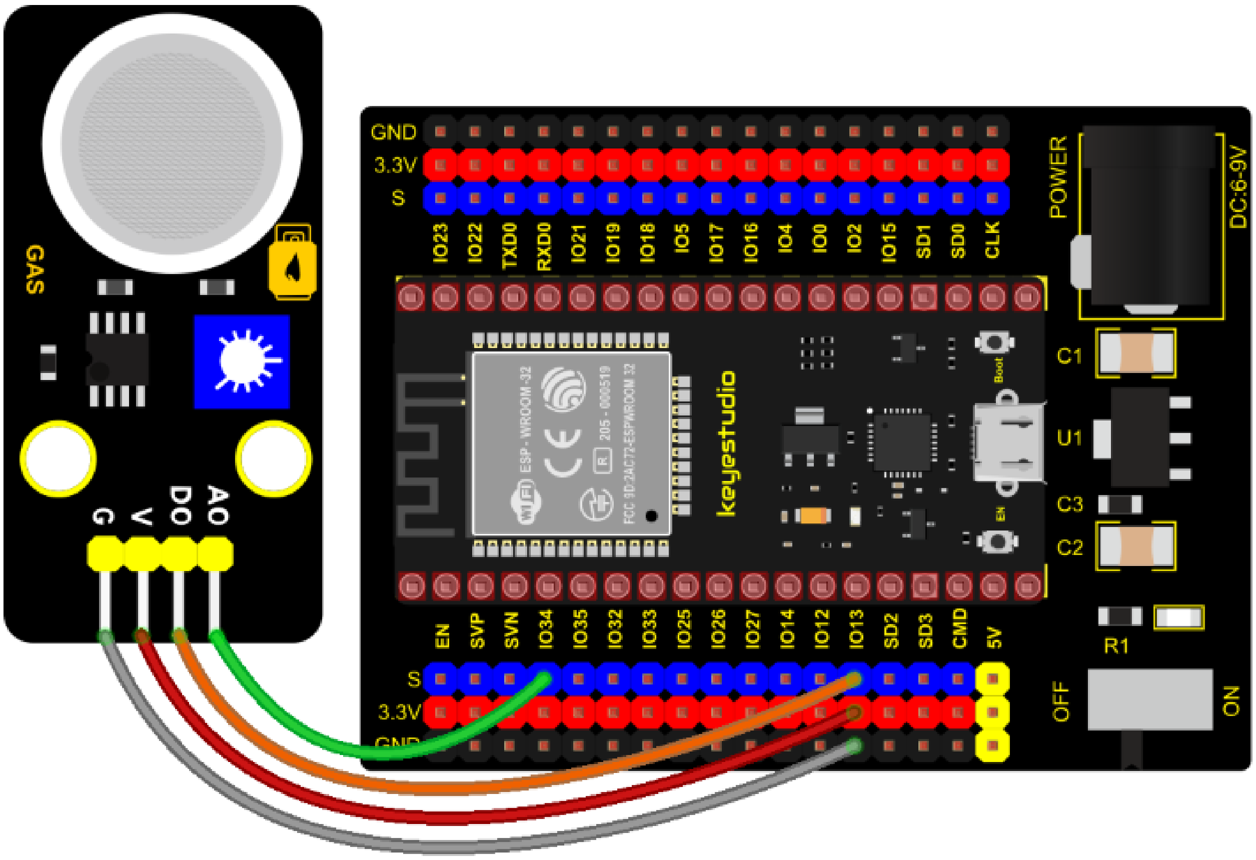
When in use, the A0 terminal reads the analog value of the corresponding gas; the D0 terminal is connected to an LM393 chip (voltage comparator), we can adjust the alarm threshold of the measured gas through the potentiometer, and output the digital value at D0. When the measured gas content exceeds the critical point, the D0 terminal outputs a low level; when the measured gas content does not exceed the critical point, the D0 terminal outputs a high level.



Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio DIY Analog Gas Sensor*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : MQ2
 * Description : Read the basic usage of Digital, ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
//MQ_2 two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin =  13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value, 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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
```

(continues on next page)

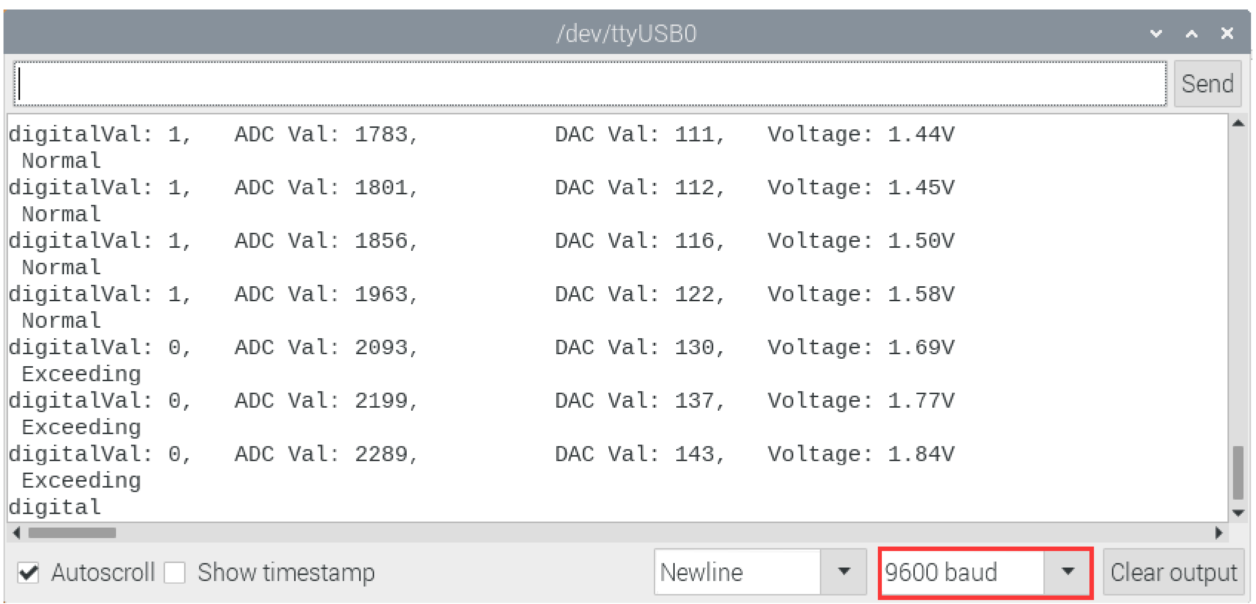
(continued from previous page)

```
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", digitalVal, adcVal, dacVal, voltage);
  if (digitalVal == 1) {
    Serial.println(" Normal");
  }
  else {
    Serial.println(" Exceeding");
  }
  delay(100); //Delay time 100 ms
}
//*********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the red LED bright and not bright critical point. Open the monitor , set baud rate to 9600.

We need to press the reset button on the ESP32, then the monitor displays the corresponding data and characters. When the sensor detects the smoke or combustible gas, the red LED lights up and the digital value changes from 1 to 0, the ADC value, DAC value and voltage value increase, as shown below.



7.5.28 Project 28: MQ-3 Alcohol Sensor

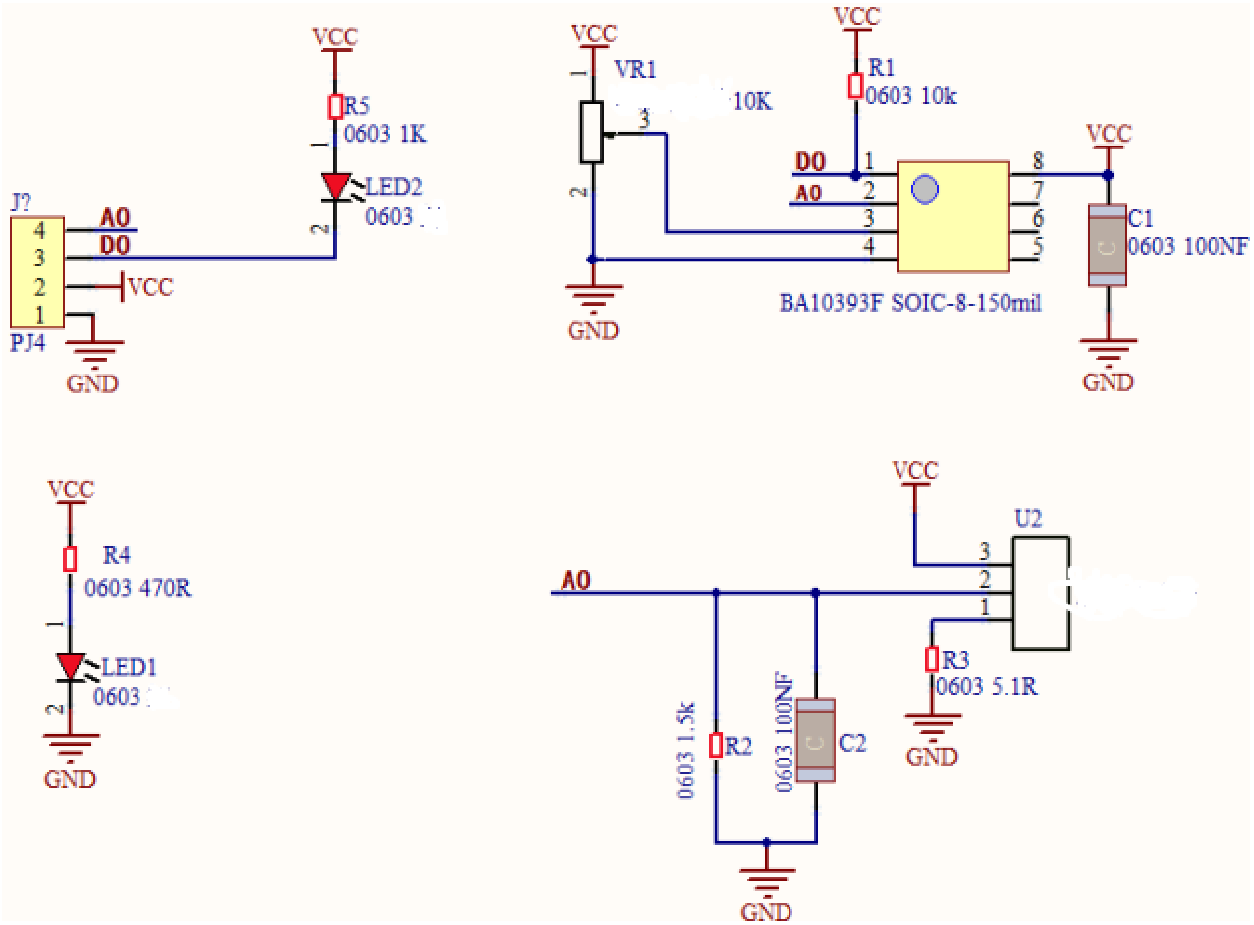
Description

In this kit, there is a MQ-3 alcohol sensor, which uses the gas-sensing material is tin dioxide (SnO2) which has a low conductivity in clean air. When there is alcohol vapor in the environment where the sensor is located, the conductivity of the sensor increases with the increase of the alcohol gas concentration in the air. The change in conductivity can be converted into an output signal corresponding to the gas concentration using a simple circuit.

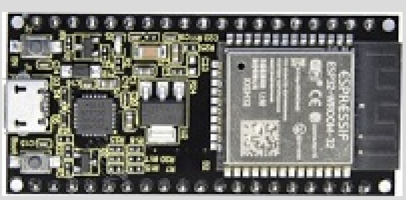
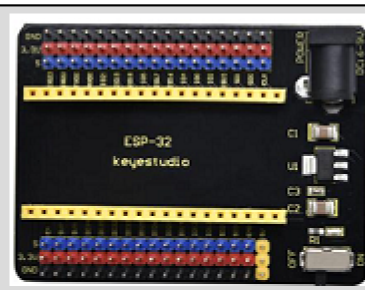
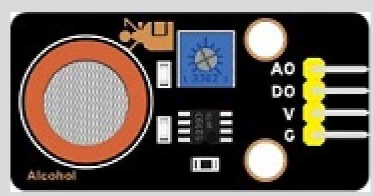
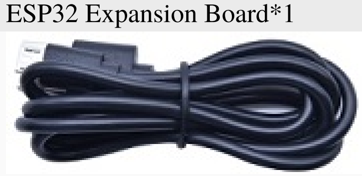
In the experiment, we read the analog value at the A0 end of the sensor and the digital value at the D0 end to judge the content of alcohol vapor in the air and whether they exceed the standard.

Working Principle

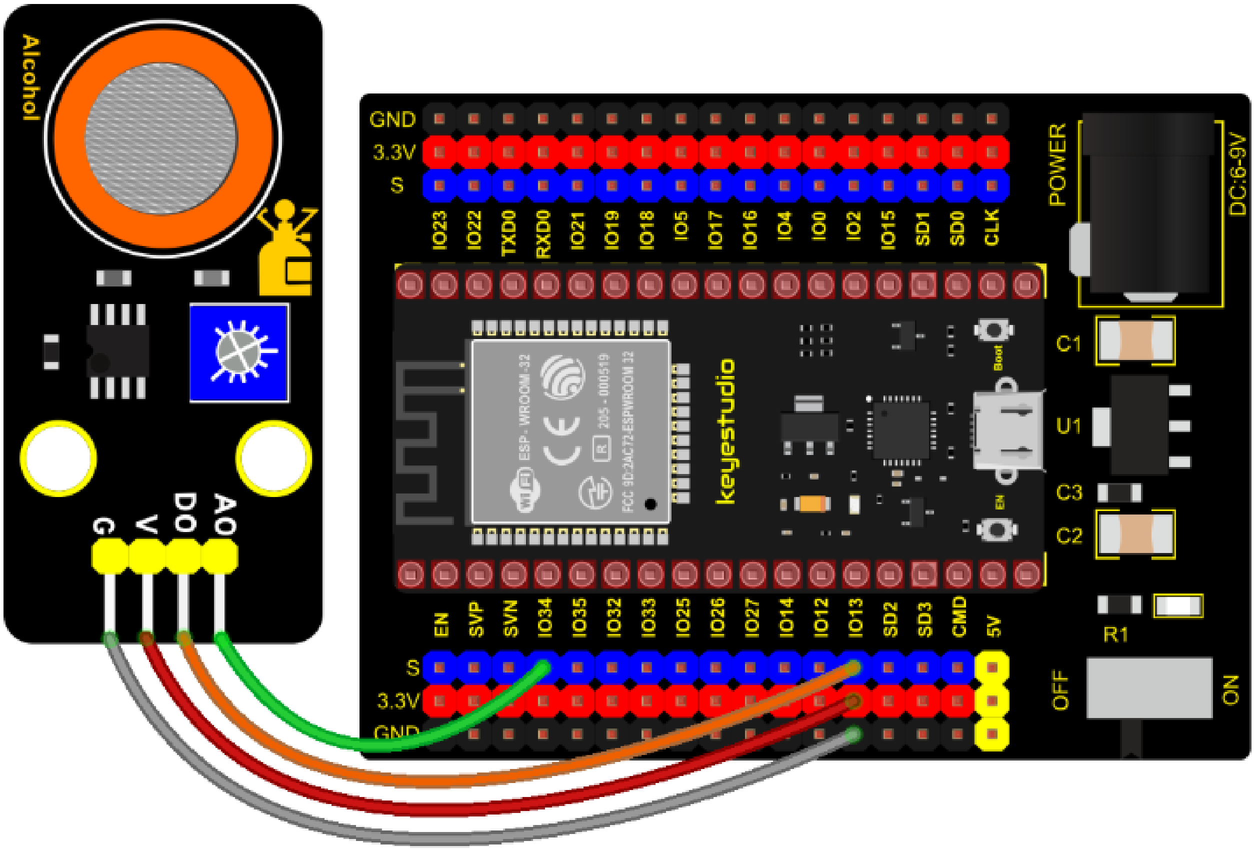
At a certain temperature, the conductivity changes with the composition of the ambient gas. When in use, A0 terminal reads the analog value corresponding to alcohol vapor; D0 terminal is connected to an LM393 chip (comparator), we can adjust and measure the alcohol vapor alarm threshold through the potentiometer, and output the digital value at D0. When the measured alcohol vapor content exceeds the critical point, the D0 terminal outputs a low level; when the measured alcohol vapor content does not exceed the critical point, the D0 terminal outputs a high level.



Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio Alcohol Sensor*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : MQ3
 * Description : Read the basic usage of Digital, ADCDAC and Voltage
 * Auther      : http//www.keyestudio.com
*/
//MQ_3 two pins 13, 34, respectively
#define PIN_ANALOG_IN  34
int digitalPin =  13;

//The following two variables hold the digital signal and adc values respectively
int analogVal = 0;
int adcVal = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalPin, INPUT); //Digital pin 13 is set to input mode
}

//In loop()the digitalRead()function is used to obtain the digital value, 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 input and output voltage are calculated according to the previous formula, and the information is finally printed out.
```

(continues on next page)

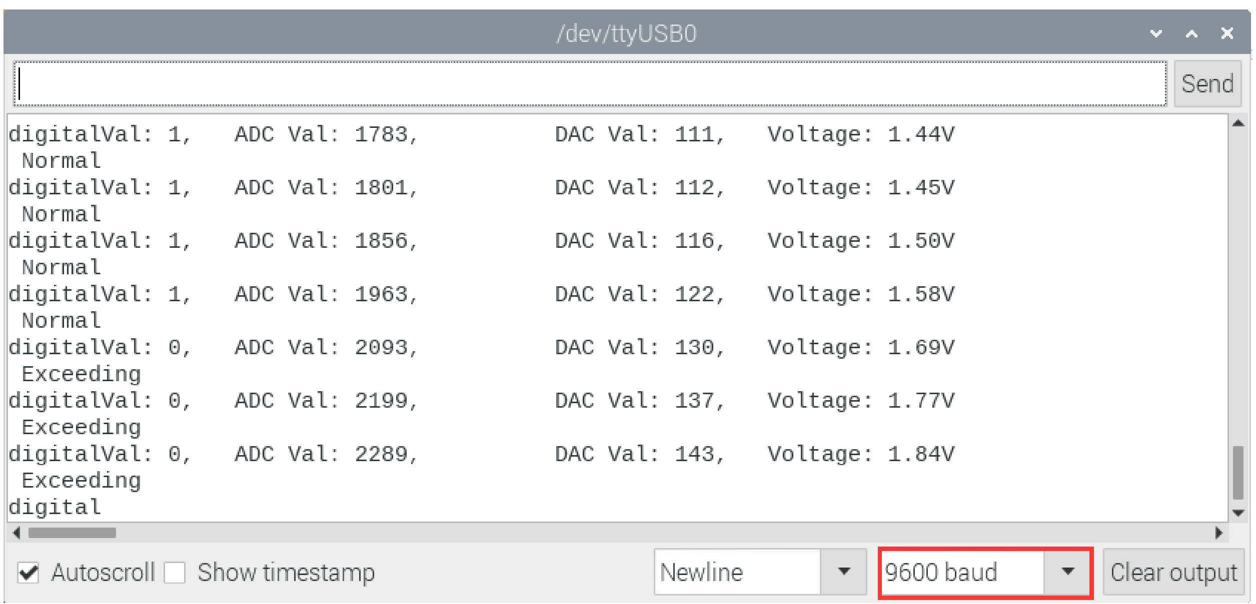
(continued from previous page)

```
void loop() {
  int digitalVal = digitalRead(digitalPin);  //Read digital signal;
  int adcVal = analogRead(PIN_ANALOG_IN);
  int dacVal = map(adcVal, 0, 4095, 0, 255);
  double voltage = adcVal / 4095.0 * 3.3;
  Serial.printf("digitalVal: %d, \t ADC Val: %d, \t DAC Val: %d, \t Voltage: %.2fV\n", digitalVal, adcVal, dacVal, voltage);
  if (digitalVal == 1) {
    Serial.println("  Normal");
  }
  else {
    Serial.println("  Exceeding");
  }
  delay(100); //Delay time 100 ms
}
//*********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Rotating the potentiometer on the sensor, we can adjust the yellow and green LED bright and not bright critical point. Open the monitor, set baud rate to 9600.

We need to press the reset button on the ESP32, then the monitor displays the corresponding data and characters. When the sensor detects the alcohol gas, the yellow and green LED lights up and the digital value changes from 1 to 0, the ADC value, DAC value and voltage value decrease, as shown below.



7.5.29 Project 29: Five-key AD Button Module

Description

When we talked about analog and digital sensors earlier, we talked about the single-channel key module. When we press the key, it outputs a low level, and when we release the key, it outputs a high level. We can only read these two digital signals. In fact, the key module ADC acquisition can also be performed. In this kit, a DIY electronic building block five-way AD button module is included.

We can judge which key is pressed through the analog value. In the experiment, we print out the key press information in the shell.

Working Principle

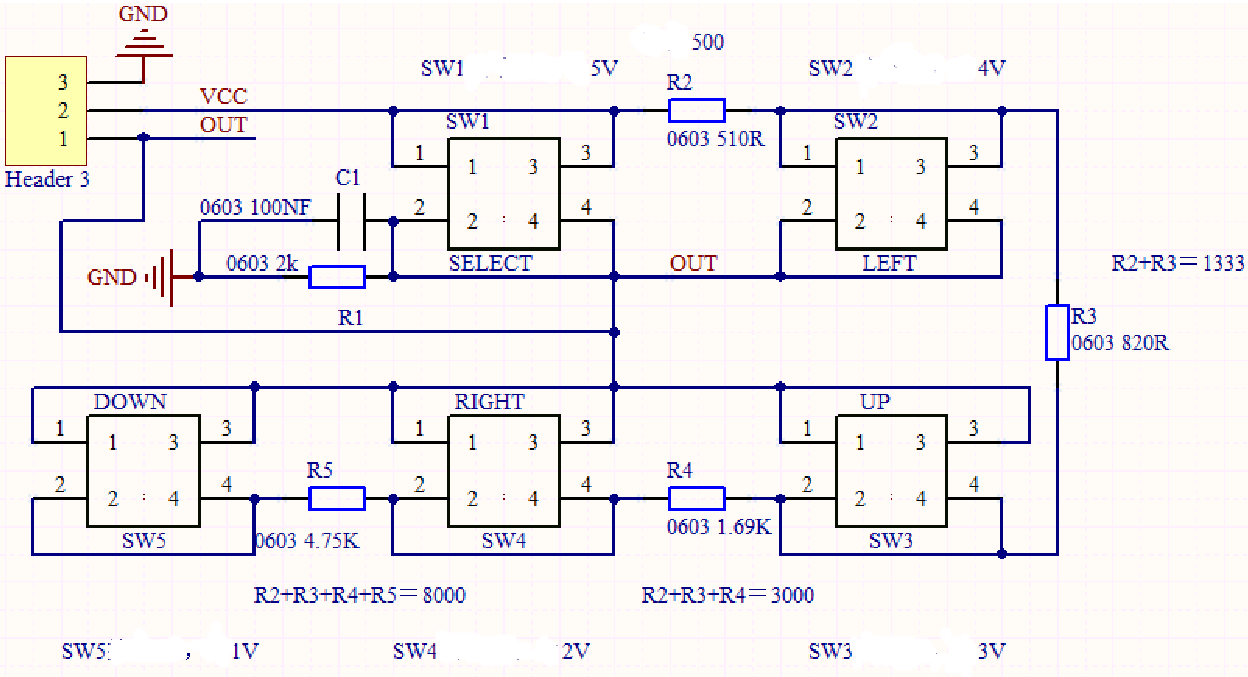
Let's look at the schematic diagram, when we do not press the key, the OUT of S output to the signal end is pulled down by R1. At this time, we read the low level 0V. When we press the key SW1, the OUT of the output to the signal end S is directly connected to the VCC. At this time, we read the high level 3.3V(the figure is marked as a 12-bit ADC(0~4095) and VCC is 5V. The principle is the same. Here we have VCC of 3.3V and ADC mapped to 12 bits), which is an analog value of 4095.

Next,when we press the key SW2, the OUT terminal voltage of the signal we read is the voltage between R2 and R1, namely VCC*R1/(R2+R1), which is about 2.64V, and the analog value is about 3276.

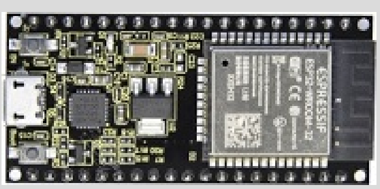
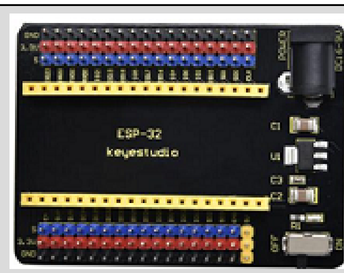
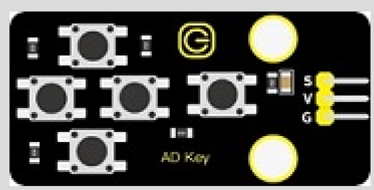
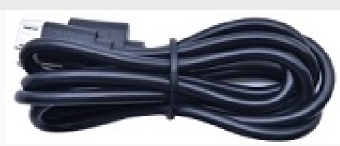
When we press the key SW3, the OUT terminal voltage of the signal we read is the voltage between R2+R3 and R1, namely VCC*R1/(R3+R2+R1), which is about 1.99V, and the analog value is about 2469.

When we press the key SW4, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4 and R1, namely VCC*R1/(R4+R3+R2+R1), about 1.31V, and the analog value is about 1626.

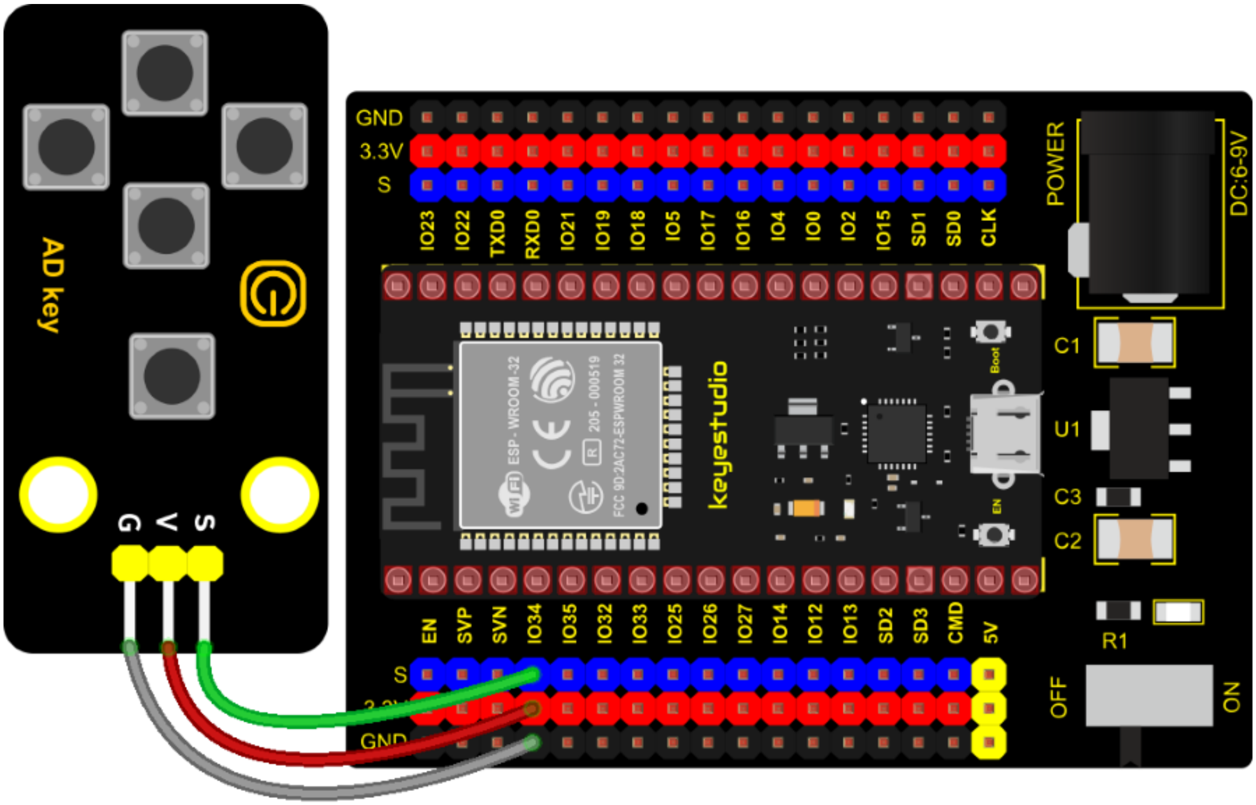
Similarly, when we press the key SW5, the OUT terminal voltage of the signal we read is the voltage between R2+R3+R4+R5 and R1, namely VCC*R1/(R5+R4+R3+R2+R1), which is about 0.68V, and the analog value is about 844.



Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 5-Channel AD Button Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Five AD Keys
 * Description : Read the value of Five AD Keys
 * Auther      : http//www.keyestudio.com
*/
int val = 0;
int ADkey = 34; //Define five AD keys connected to GPIO36
void setup() {
  Serial.begin(9600); //Set baud rate to 9600
}

void loop() {
  val = analogRead(ADkey);  //Read the simulated value of the AD key and assign it to␣
↪the variable val
  Serial.print(val);  //A newline prints the variable val
  if (val <= 500) { //Val is less than or equal to 500 when no button is pressed
    Serial.println("   no key  is pressed");
  } else if (val <= 1000) { //When key 5 is pressed,val is between 500 and 1000
    Serial.println("   SW5 is pressed");
  } else if (val <= 2000) { //When pressed,val is between 1000 and 2000
    Serial.println("   SW4 is pressed");
  } else if (val <= 3000) { //When pressed,val is between 2000 and 3000
    Serial.println("   SW3 is pressed");
  } else if (val <= 4000) { //When key 2 is pressed,val is between 3000 and 4000
    Serial.println("   SW2 is pressed");
```

(continues on next page)

(continued from previous page)

```
  } else {  //When key 1 is pressed,val is greater than 4000
    Serial.println("   SW1 is pressed");
  }
}
//*********************************************************************************
```

Code Explanation

We assign the read analog value to the variable val, and the serial monitor displays the value of val, (we set to 9600).

When the analog value is in the range of 500 and 1000, the button SW5 is pressed; when the analog value is in the 1000 and 2000, the button SW4 is pressed; when the analog value is between 2000 and 3000, the button SW3 is pressed; when the analog value is between 3000 and 4000, the button SW2 is pressed. When the analog value is above 4000, we judge that the button SW1 is pressed.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600. We need to press the reset button on the ESP32, when the button is pressed, the serial monitor prints out the corresponding information, as shown in the figure below.

7.5.30 Project 30: Joystick Module

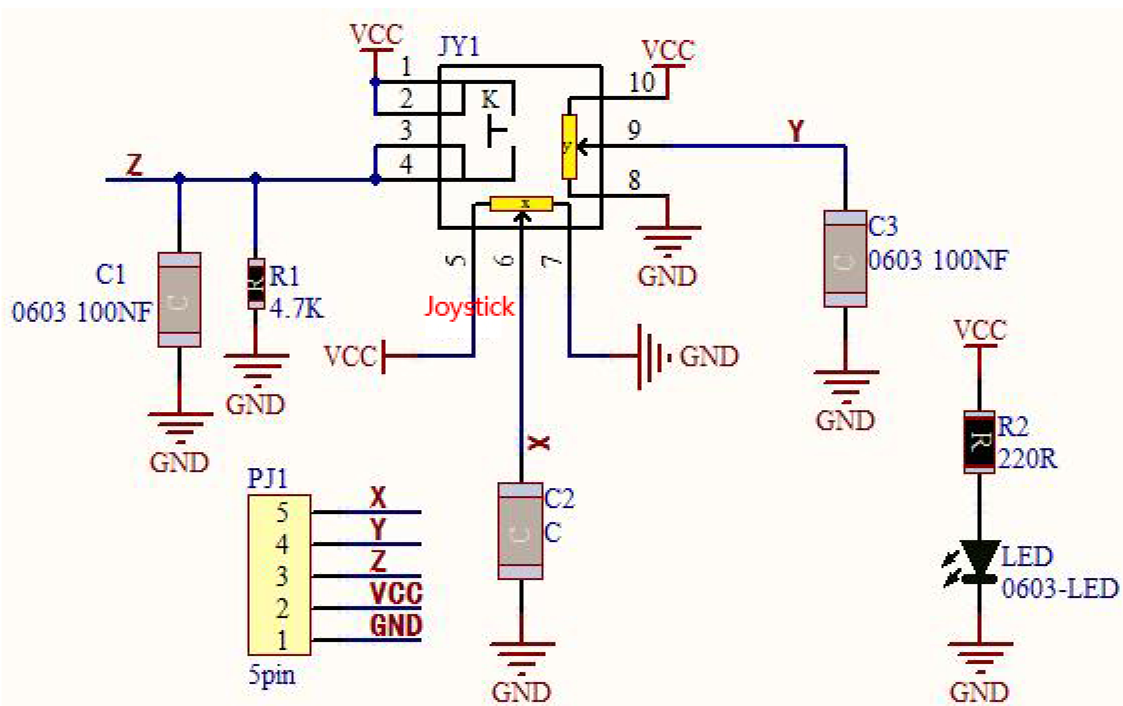
Overview

Game handle controllers are ubiquitous.

It mainly uses PS2 joysticks. When controlling it, we need to connect the X and Y ports of the module to the analog port of the single-chip microcomputer, port B to the digital port of the single-chip microcomputer, VCC to the power output port(3.3-5V), and GND to the GND of the MCU. We can read the high and low levels of two analog values and one digital port) to determine the working status of the joystick on the module.

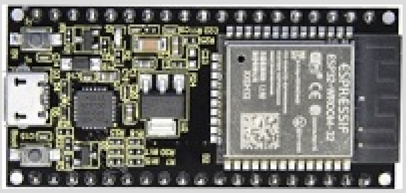
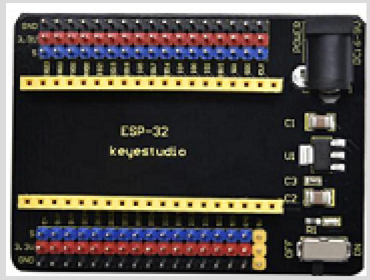
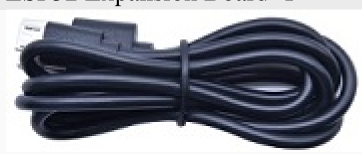
In the experiment, two analog values(x axis and y axis) will be shown on Shell.

Working Principle

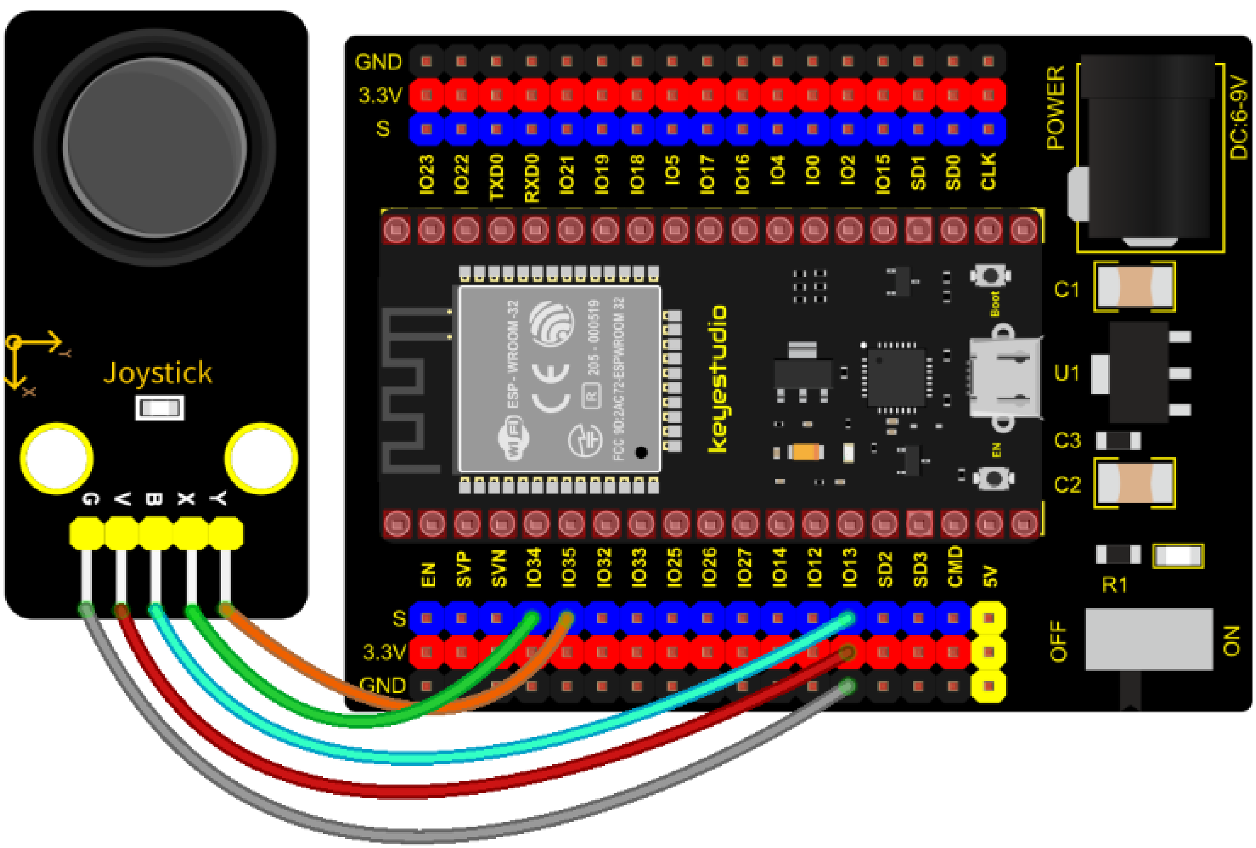


In fact, its working principle is very simple. Its inside structure is equivalent to two adjustable potentiometers and a button. When this button is not pressed and the module is pulled down by R1, low levels will be output ; on the contrary, when the button is pressed, VCC will be connected (high levels), When we move the joystick, the internal potentiometer will adjust to output different voltages, and we can read the analog value.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Joystick Module*1
		
5P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Joystick
 * Description : Read data from Rocker.
 * Auther      : http//www.keyestudio.com
*/
int xyzPins[] = {34, 35, 13};   //x,y,z pins
void setup() {
  Serial.begin(9600);
  pinMode(xyzPins[0], INPUT); //x axis.
  pinMode(xyzPins[1], INPUT); //y axis.
  pinMode(xyzPins[2], INPUT_PULLUP);   //z axis is a button.
}

// 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.
void loop() {
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.println("X,Y,Z: " + String(xVal) + ", " +  String(yVal) + ", " + String(zVal));
  delay(500);
}
//**********************************************************************************
```

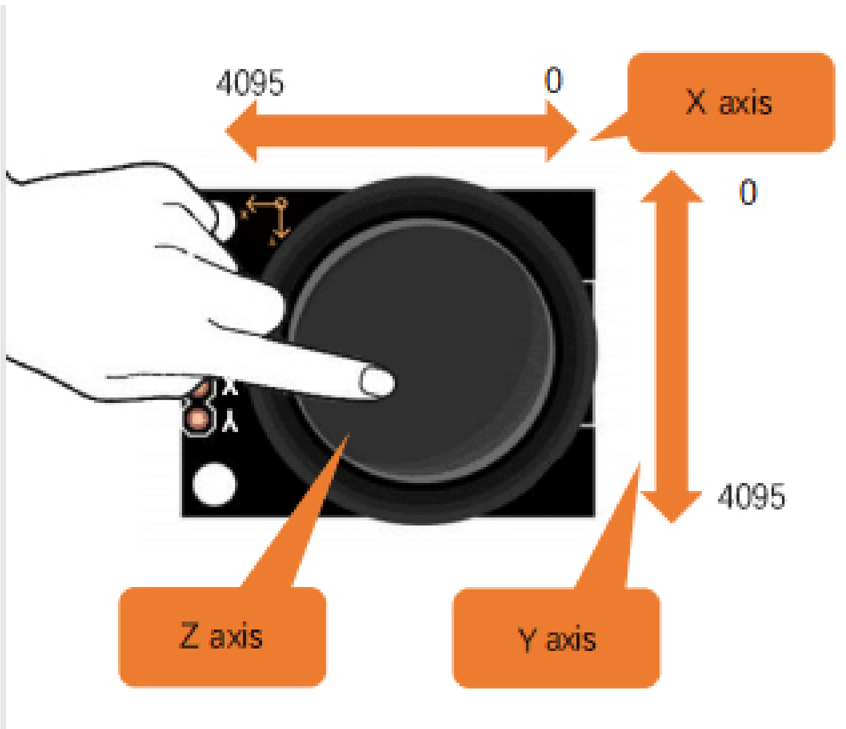
Code Explanation

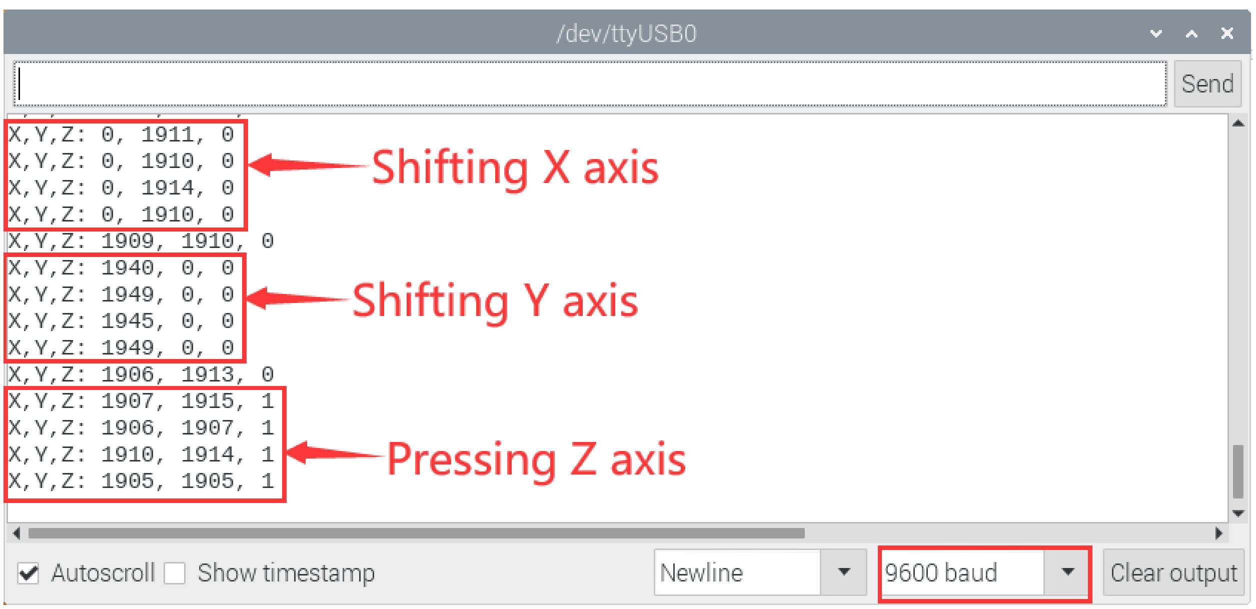
In the experiment, according to the wiring diagram, the x pin is set to GPIO34, the y pin is set to GPIO35 and the pin of the joystick is set to GPIO13.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600.

We need to press the reset button on the ESP32, then the serial monitor will show the corresponding value. Moving the joystick or pressing it will change the analog and digital values in the serial monitor .





7.5.31 Project 31: Relay Module

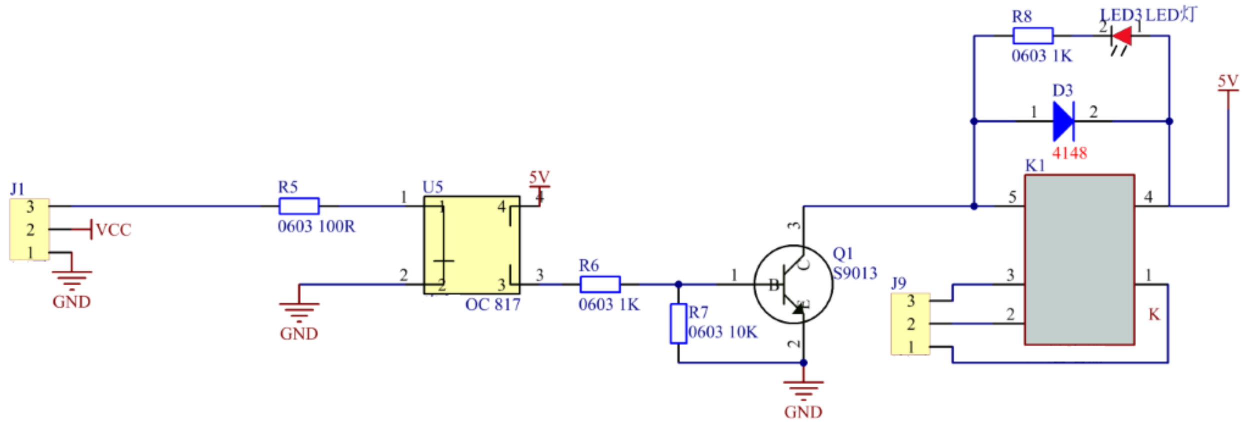
Overview

In our daily life, we usually use communication to drive electrical equipment, and sometimes we use switches to control electrical equipment. If the switch is connected directly to the ac circuit, leakage occurs and people are in danger. Therefore, from the perspective of safety, we specially designed this relay module with NO(normally open) end and NC(normally closed) end.

Working Principle

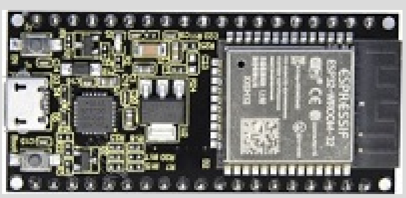
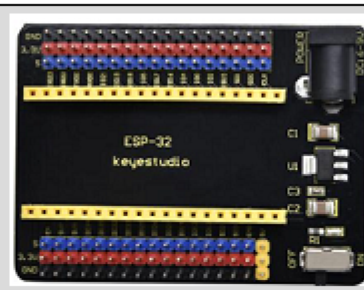
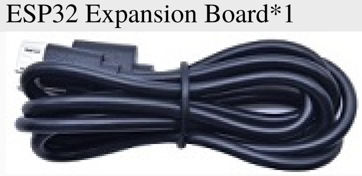
Relay is compatible with a variety of microcontroller control board, such as Arduino series microcontroller, which is a small current to control the operation of large current “automatic switch”.

Input Voltage3.3V-5V

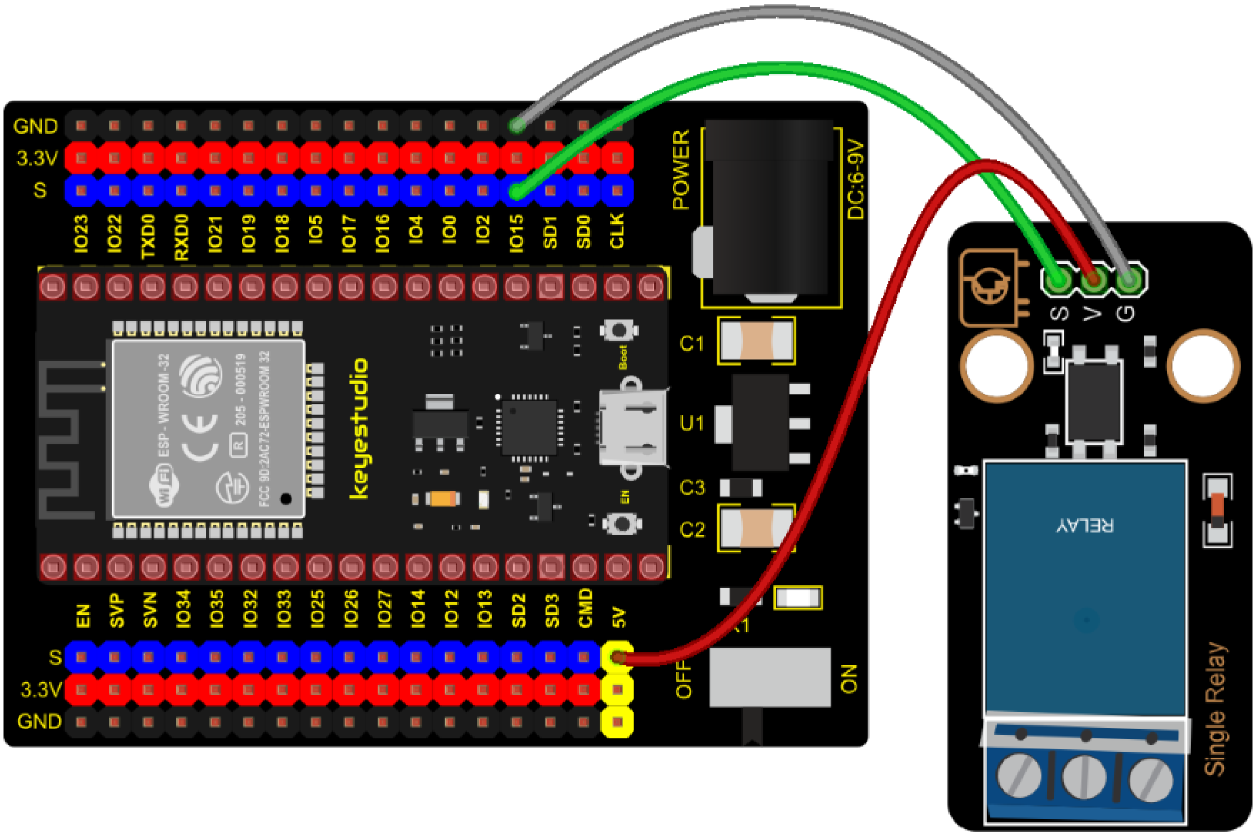


It can let the MCU control board drive 3A load, such as an LED lamp belt, a DC motor, a micro water pump and a solenoid valve plugable interface design, which is easy to use.

Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Relay Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Relay
 * Description : Relay turn on and off.
 * Auther      : http//www.keyestudio.com
*/
#define  Relay  15 // defines digital 15
void setup()
{
pinMode(Relay, OUTPUT); // sets "Relay" to "output"
}
void loop()
{
digitalWrite(Relay, HIGH); // turns on the relay
delay(1000); //delays 1 seconds
digitalWrite(Relay, LOW); // turns off the relay
delay(1000); // delays 1 seconds
}
//**********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The relay will cycle on and off, on for 1 second, off for 1 second. At the same time, you can hear the sound of the relay on and off as well as see the change of the indicator light on the relay.

7.5.32 Project 32: SK6812 RGB Module

Overview

In previous lessons, we learned about the plug-in RGB module and used PWM signals to color the three pins of the module.

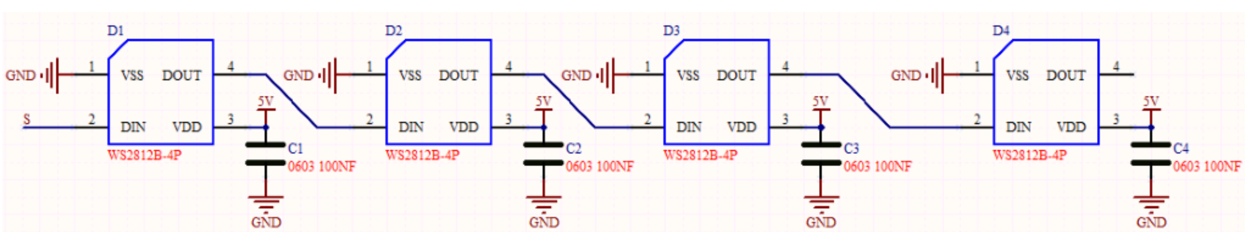
There is a Keyestudio 6812 RGB module whose the driving principle is different from the plug-in RGB module. It can only control with one pin. This is a set. It is an intelligent externally controlled LED light source with the control circuit and the light-emitting circuit. Each LED element is the same as a 5050 LED lamp bead, and each component is pixel. There are four lamp beads on the module, which indicates four pixels.

In the experiment, we make different lights show different colors.

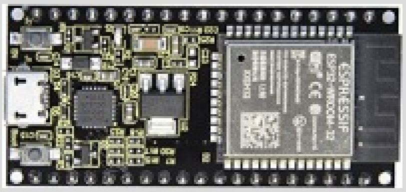
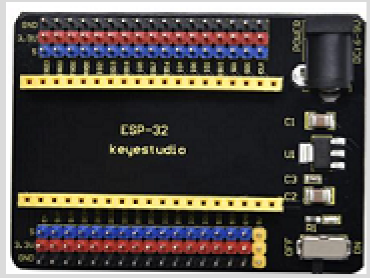
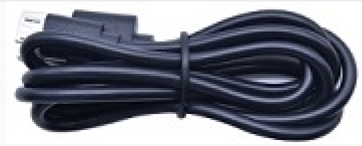
Working Principle

From the schematic diagram, we can see that these four pixel lighting beads are all connected in series. In fact, no matter how many they are, we can use a pin to control a light and let it display any color. The pixel point contains a data latch signal shaping amplifier drive circuit, a high-precision internal oscillator and a 12V high-voltage programmable constant current control part, which effectively ensures the color of the pixel point light is highly consistent.

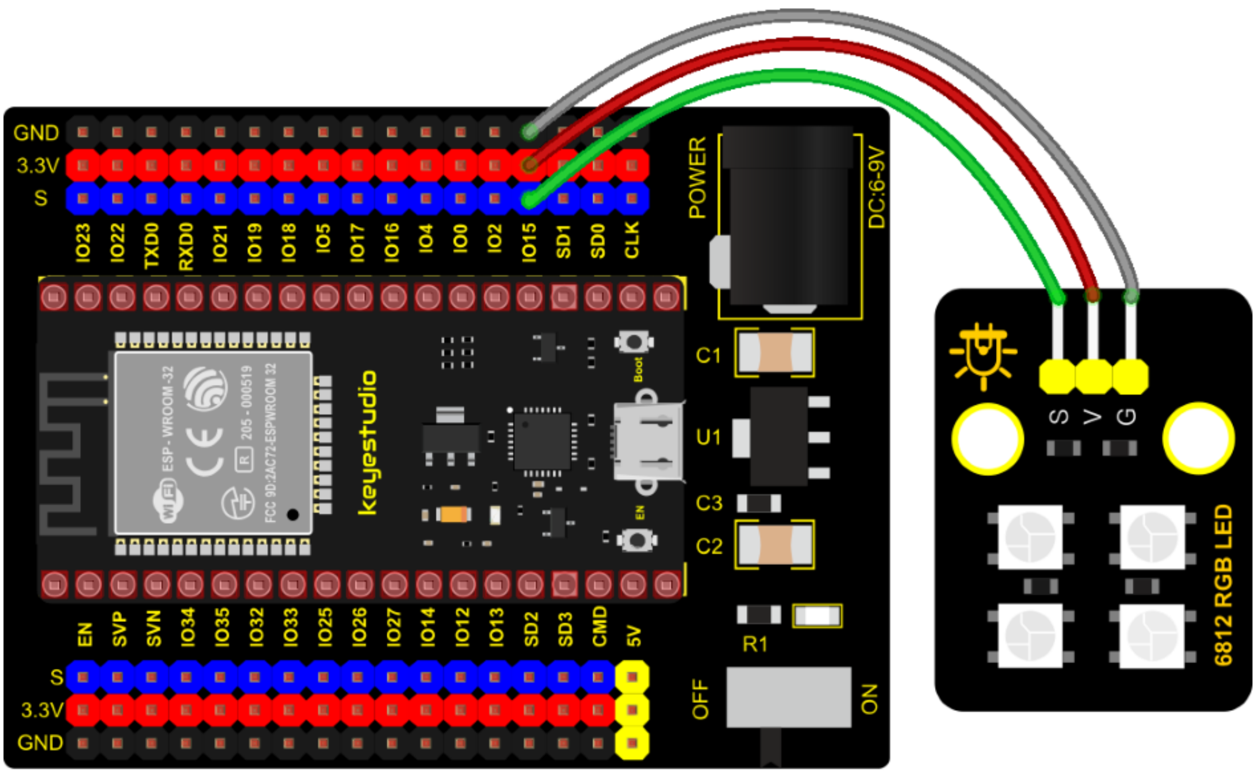
The data protocol adopts a single-wire zero-code communication method. After the pixel is powered up and reset, the S terminal receives the data transmitted from the controller. The first 24bit data sent is extracted by the first pixel and sent to the data latch of the pixel.



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 6812 RGB Module*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : sk6812 RGB LED
 * Description : turn on sk6812 RGB LED
```

(continues on next page)

(continued from previous page)

```
 * Auther      : http//www.keyestudio.com
*/
#include <Adafruit_NeoPixel.h>

#define PIN 15

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);

// IMPORTANT: To reduce NeoPixel burnout risk, add 1000 uF capacitor across pixel power leads, add 300 - 500 Ohm resistor on first pixel's data input and minimize distance between Arduino and first pixel.  Avoid connecting on a live circuit...if you must, connect GND first.

void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  // Send a theater pixel chase in...
  theaterChase(strip.Color(127, 127, 127), 50); // White
  theaterChase(strip.Color(127,   0,   0), 50); // Red
  theaterChase(strip.Color(  0,   0, 127), 50); // Blue

  rainbow(20);
  rainbowCycle(20);
  theaterChaseRainbow(50);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, c);
      strip.show();
      delay(wait);
  }
}

void rainbow(uint8_t wait) {
  uint16_t i, j;
```

(continues on next page)

(continued from previous page)

```
  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Theatre-style crawling lights.
void theaterChase(uint32_t c, uint8_t wait) {
  for (int j=0; j<10; j++) {  //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}

//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait) {
  for (int j=0; j < 256; j++) {     // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
        for (int i=0; i < strip.numPixels(); i=i+3) {
          strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel on
        }
        strip.show();

        delay(wait);

        for (int i=0; i < strip.numPixels(); i=i+3) {
```

(continues on next page)

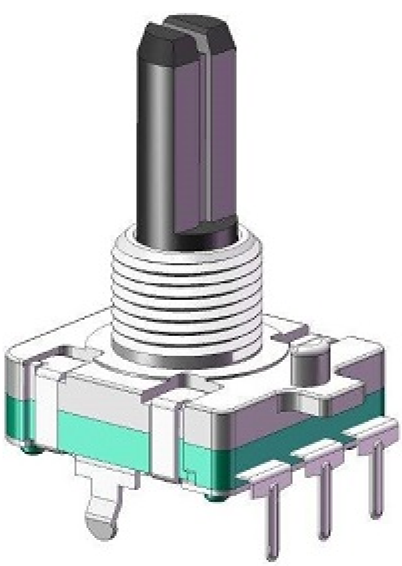
(continued from previous page)

```
          strip.setPixelColor(i+q, 0);        //turn every third pixel off
        }
    }
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
   return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else {
   WheelPos -= 170;
   return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
//************************************************************************************
```

Test Code

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Then We can see 4 RGB leds on the module emitting various color lighting effects.

7.5.33 Project 33: Rotary Encoder

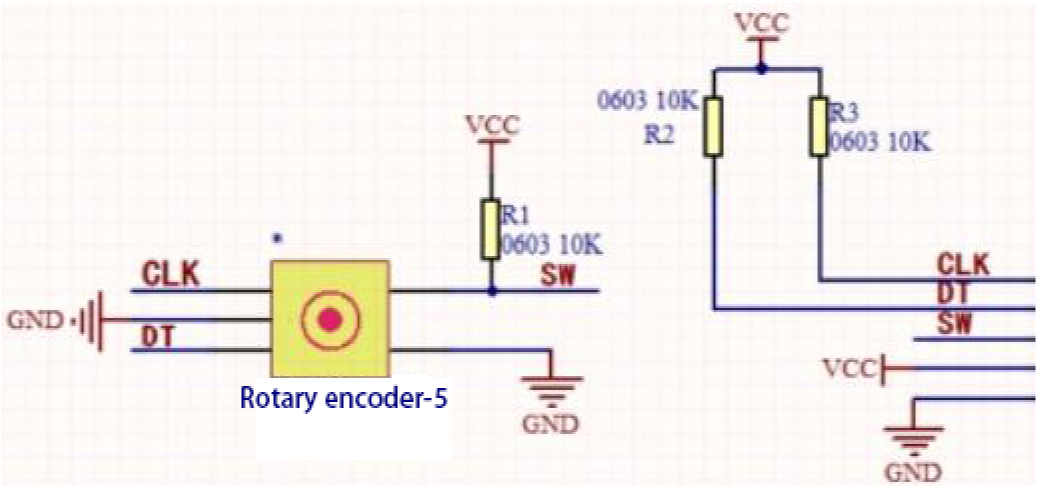


Overview

In this kit, there is a Keyestudio rotary encoder, dubbed as switch encoder. It is applied to automotive electronics, multimedia audio, instrumentation, household appliances, smart home, medical equipment and so on.

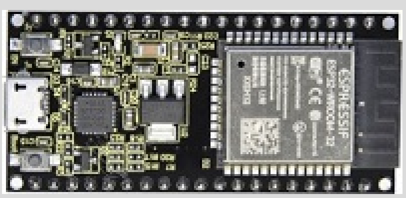
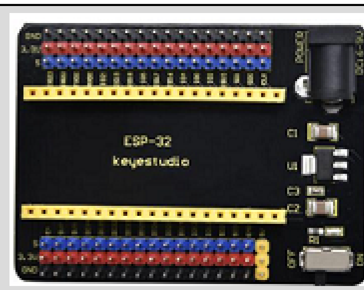
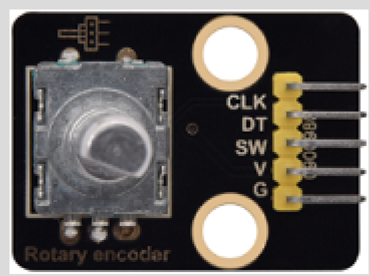
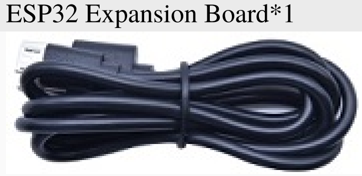
In the experiment, it it used for counting. When we rotate the rotary encoder clockwise, the set data is up 1; if you rotate it anticlockwise, the set data falls by 1; and when the middle button is pressed, the value will be show in the serial monitor.

Working Principle

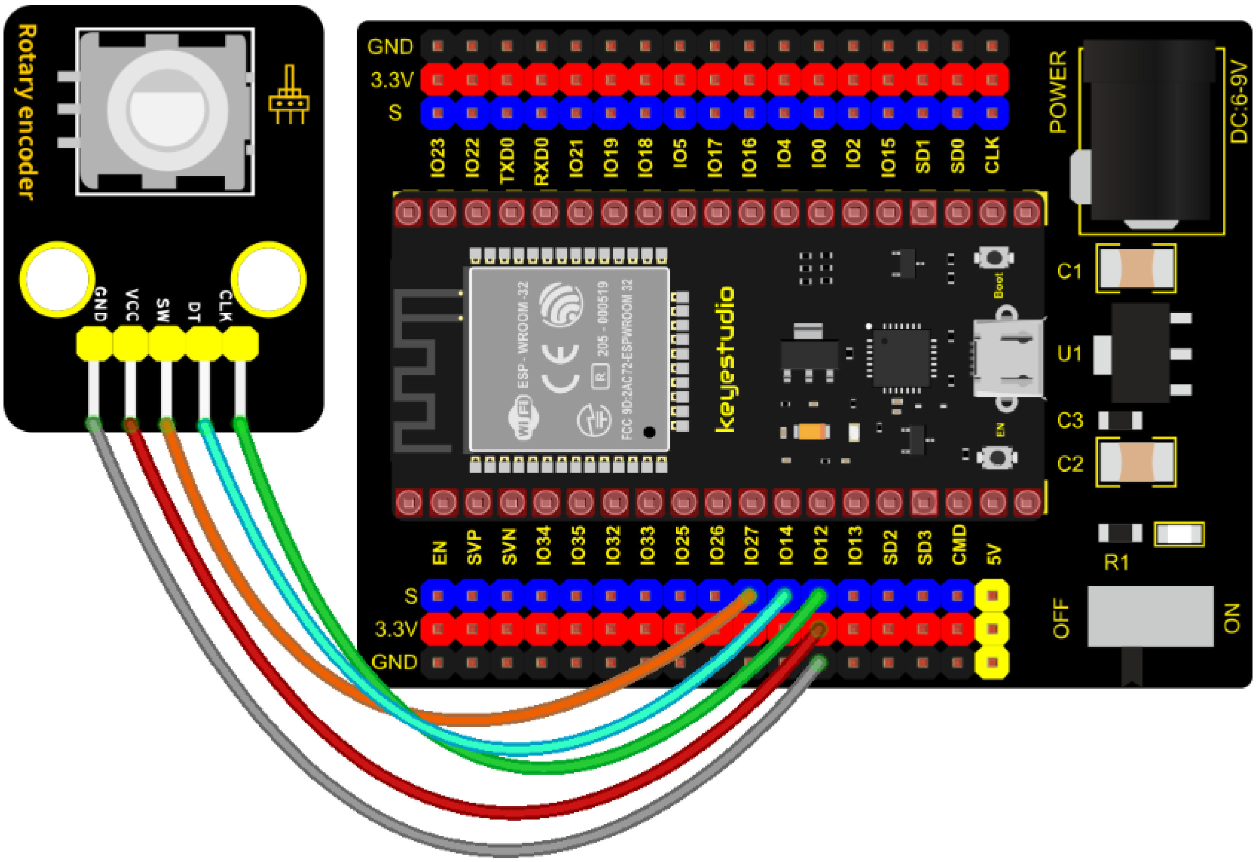


The incremental encoder converts the displacement into a periodic electric signal, and then converts this signal into a counting pulse, and the number of pulses indicates the size of the displacement. This module mainly uses 20-pulse rotary encoder components. It can calculate the number of pulses output during clockwise and reverse rotation. There is no limit to count rotation. It resets to the initial state, that is, starts counting from 0.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Rotary Encoder*1
		
5P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Encoder
 * Description : Rotary encoder module counting.
 * Auther      : http//www.keyestudio.com
*/
//Interfacing Rotary Encoder with Arduino
//Encoder Switch -> pin 27
//Encoder DT -> pin 14
//Encoder CLK -> pin 12

int Encoder_DT  = 14;
int Encoder_CLK  = 12;
int Encoder_Switch = 27;

int Previous_Output;
int Encoder_Count;

void setup() {
  Serial.begin(9600);

  //pin Mode declaration
  pinMode (Encoder_DT, INPUT);
  pinMode (Encoder_CLK, INPUT);
```

(continues on next page)

(continued from previous page)

```
  pinMode (Encoder_Switch, INPUT);

  Previous_Output = digitalRead(Encoder_DT); //Read the inital value of Output A
}

void loop() {
  //aVal = digitalRead(pinA);

  if (digitalRead(Encoder_DT) != Previous_Output)
  {
    if (digitalRead(Encoder_CLK) != Previous_Output)
    {
      Encoder_Count ++;
      Serial.println(Encoder_Count);
    }
    else
    {
      Encoder_Count--;
      Serial.println(Encoder_Count);
    }
  }

  Previous_Output = digitalRead(Encoder_DT);

  if (digitalRead(Encoder_Switch) == 0)
  {
    delay(5);
    if (digitalRead(Encoder_Switch) == 0) {
      Serial.println("Switch pressed");
      while (digitalRead(Encoder_Switch) == 0);
    }
  }
}
//*********************************************************************************
```

Code Explanation

Set CLK to GPIO12 and DAT to GPIO14.

This code is set well in the library file. When CLK descends, read the voltage of DAT, when DAT is a HIGH level, the value of the rotary encoder is added by 1; when DAT is a LOW level, the value of the rotary encoder is cut down 1.

Set the pin of the button(GPIO27) to LOW and print.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600.

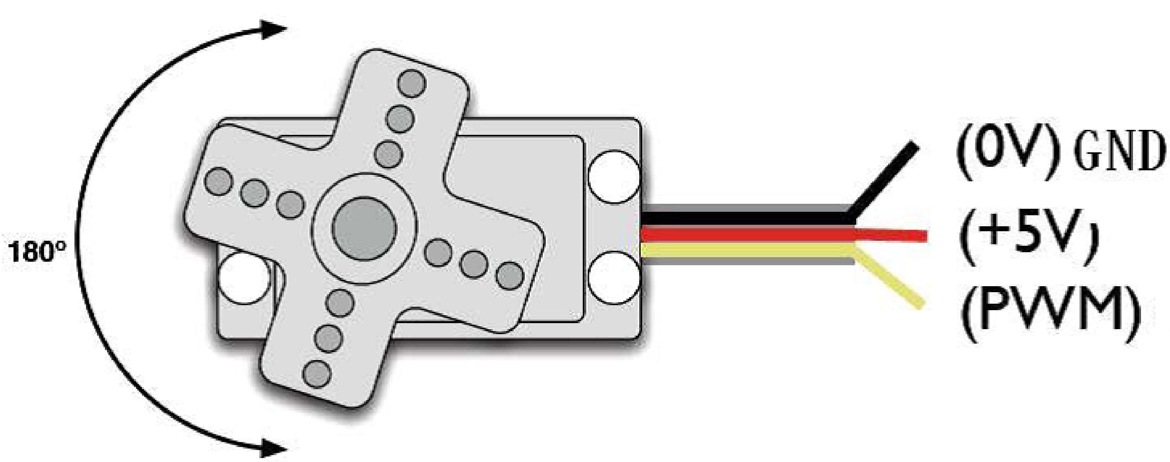
We need to press the reset button on the ESP32, then rotate the knob on the rotary encoder clockwise, the displayed data will rise; on the contrary, in anticlockwise way, the data will decrease. Equally, press the button on the rotary encoder, "Switch pressed" will be shown.

7.5.34 Project 34: Servo Control

Overview

Servo is a position control rotary actuator. It mainly consists of a housing, a circuit board, a core-less motor, a gear and a position sensor.

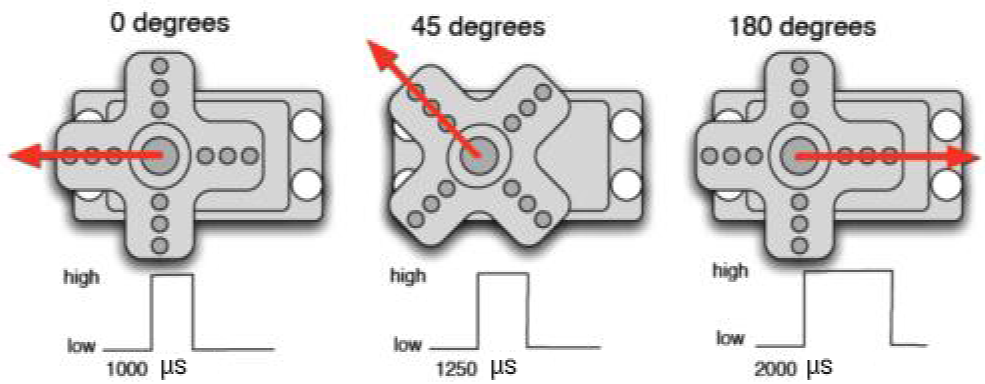
In general, servo has three lines in brown, red and orange. The brown wire is grounded, the red one is a positive pole line and the orange one is a signal line.

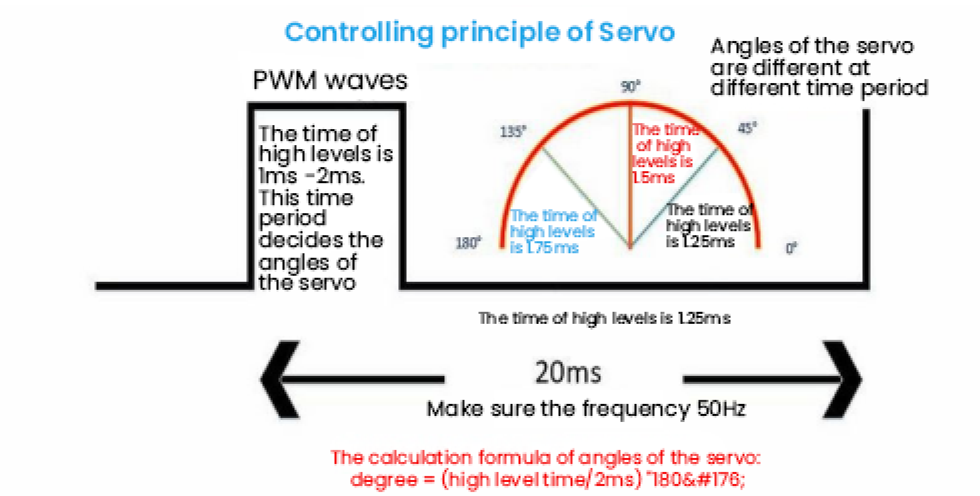


Working Principle

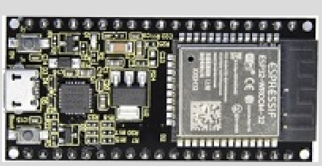
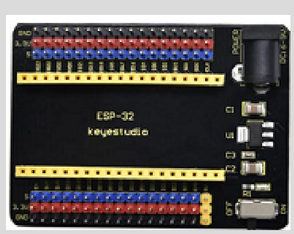
When the motor speed is constant, the potentiometer is driven to rotate through the cascade reduction gear, which leads that the voltage difference is 0, and the motor stops rotating. Generally, the angle range of servo rotation is 0° –180 °

The rotation angle of servo motor is controlled by regulating the duty cycle of PWM (Pulse-Width Modulation) signal. The standard cycle of PWM signal is 20ms (50Hz). Theoretically, the width is distributed between 1ms-2ms, but in fact, it's between 0.5ms-2.5ms. The width corresponds the rotation angle from 0° to 180°. But note that for different brand motors, the same signal may have different rotation angles.

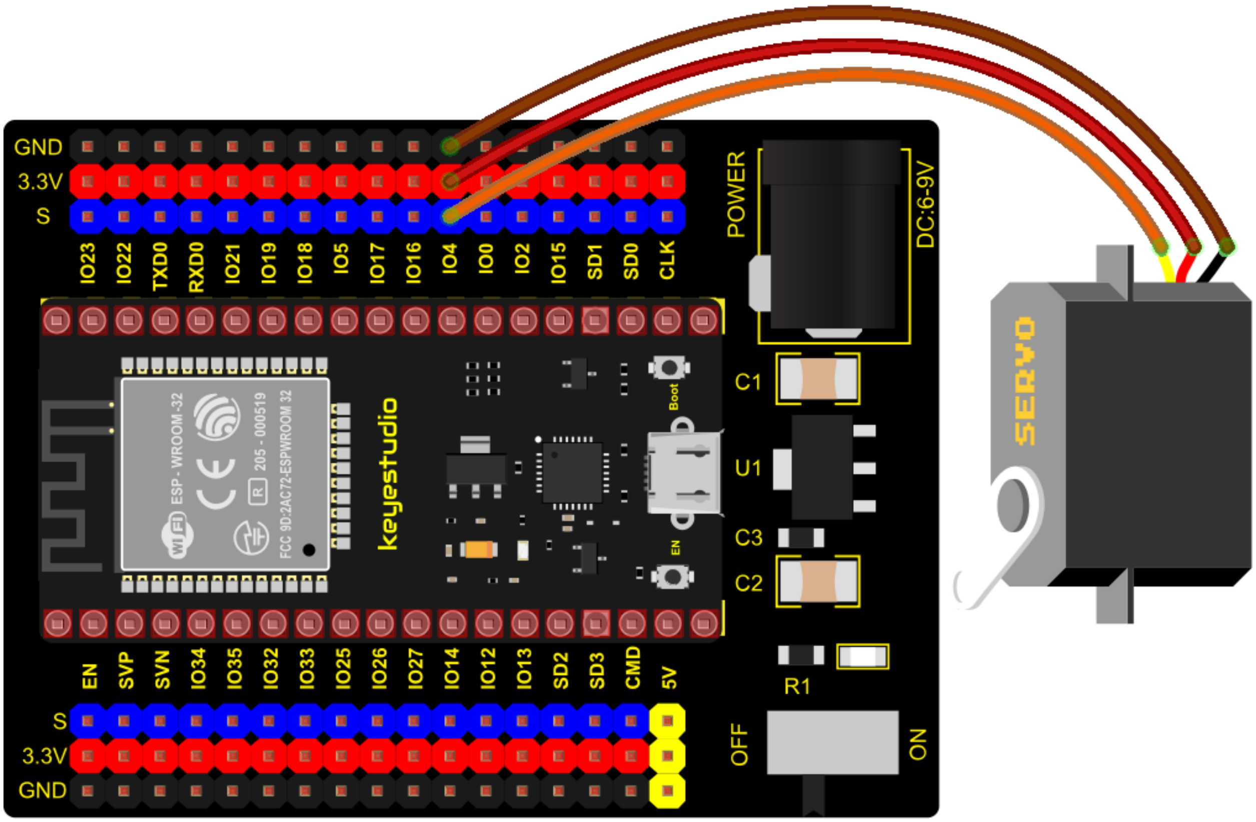




Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Servo*1	Micro USB Cable*1

Connection Diagram



Test Code 1

```
//**********************************************************************
/*
 * Filename    : Servo_1
 * Description : Steering gear rotation Angle 0-90-180, repeatly
 * Auther      : http//www.keyestudio.com
*/
int servoPin = 4;//steering gear PIN

void setup() {
  pinMode(servoPin, OUTPUT);//steering pin is set to output
}
void loop() {
  servopulse(servoPin, 0);//Rotate it to zero degrees
  delay(1000);//delay 1S
  servopulse(servoPin, 90);//Rotate it to 90 degrees
  delay(1000);
  servopulse(servoPin, 180);//Rotate it to 180 degrees
  delay(1000);
}

void servopulse(int pin, int myangle) { //Impulse function
  int pulsewidth = map(myangle, 0, 180, 500, 2500); //Map Angle to pulse width
  for (int i = 0; i < 10; i++) { //Output a few more pulses
```

(continues on next page)

(continued from previous page)

```
    digitalWrite(pin, HIGH);//Set the steering gear interface level to high
    delayMicroseconds(pulsewidth);//The number of microseconds of delayed pulse width␣
↪value
    digitalWrite(pin, LOW);//Lower the level of steering gear interface
    delay(20 - pulsewidth / 1000);
  }
}
//*************************************************************************
```

Code Explanation 1

1). **map(value, fromLow, fromHigh, toLow, toHigh)**

Value is the value we map. fromLow, fromHigh is the maximum and minimum value

toLow, toHigh are the upper limit and lower limit we map. For example, map(myangle, 0, 180, 500, 2500) means that an angle value myangle (0°-180°the mapping range is from 500us to 2500us.

2). **servopulse()**

We use the function servopulse() to make the servo move. We also make the servo rotate 0°, 90° and 180°cyclically.

Test Result 1

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, the servo will rotate 0°90° and 180° cyclically.

Test Code 2

```
//*************************************************************************
/*
 * Filename    : Servo Sweep
 * Description : Control the servo motor for sweeping
 * Auther      : http//www.keyestudio.com
*/
#include <ESP32Servo.h>

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

int posVal = 0;    // variable to store the servo position
int servoPin = 4; // 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'
```

(continues on next page)

(continued from previous page)

```
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
//*************************************************************************************
```

Code Explanation 2

myservo. write (pos) is the rotation angle to POS. myservo. read () reads the current angle value of the servo.

Test Result 2

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on, the servo will rotate from 0° to 180° by moving 1° for each 15ms.

7.5.35 Project 35: Ultrasonic Sensor

Overview

Bats and some marine animals are able to use high frequencies of sound for echolocation or communication. They can emit ultrasonic waves from the larynx through the mouth or nose and use the sound waves that bounce back to orient and determine the position, size and whether nearby objects are moving.

Ultrasonic is a frequency higher than 20000 Hz sound wave, which has a good direction, a strong penetration ability, and is easy to obtain more concentrated sound energy as well as spread far in the water. It can be used for ranging, speed measurement, cleaning, welding, gravel, sterilization and disinfection. What's more, it has many applications in medicine, military, industry and agriculture.

In this kit, there is a keyes HC-SR04 ultrasonic sensor, which can detect obstacles in front and the detailed distance between the sensor and the obstacle. Its principle is the same as that of bat flying. It can emit the ultrasonic signals that cannot be heard by humans. When these signals hit an obstacle and come back immediately. The distance between the sensor and the obstacle can be calculated by the time gap of emitting signals and receiving signals.

In the experiment, we use the sensor to detect the distance between the sensor and the obstacle, and print the test result.

Working Principle

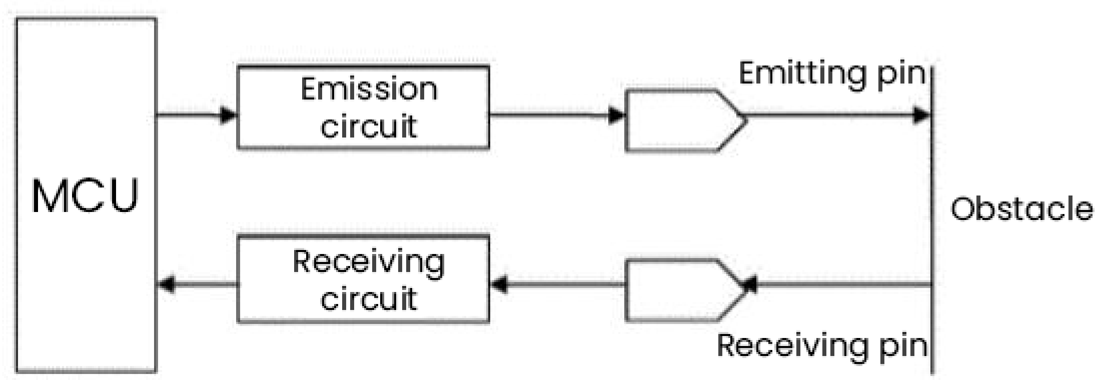
The most common ultrasonic ranging method is the echo detection. As shown below; when the ultrasonic emitter emits the ultrasonic waves towards certain direction, the counter will count. The ultrasonic waves travel and reflect back once encountering the obstacle. Then the counter will stop counting when the receiver receives the ultrasonic waves coming back.

The ultrasonic wave is also sound wave, and its speed of sound V is related to temperature. Generally, it travels 340m/s in the air. According to time t, we can calculate the distance s from the emitting spot to the obstacle. $s = 340t/2$ The HC-SR04 ultrasonic ranging module can provide a non-contact distance sensing function of 2cm-400cm, and the ranging accuracy can reach as high as 3mm; the module includes an ultrasonic transmitter, receiver and control circuit. Basic working principle:

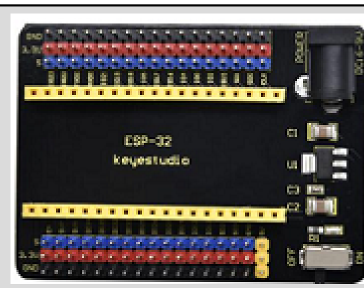
1). First pull down the TRIG, and then trigger it with at least 10us high level signal;

2). After triggering, the module will automatically transmit eight 40KHZ square waves, and automatically detect whether there is a signal to return.

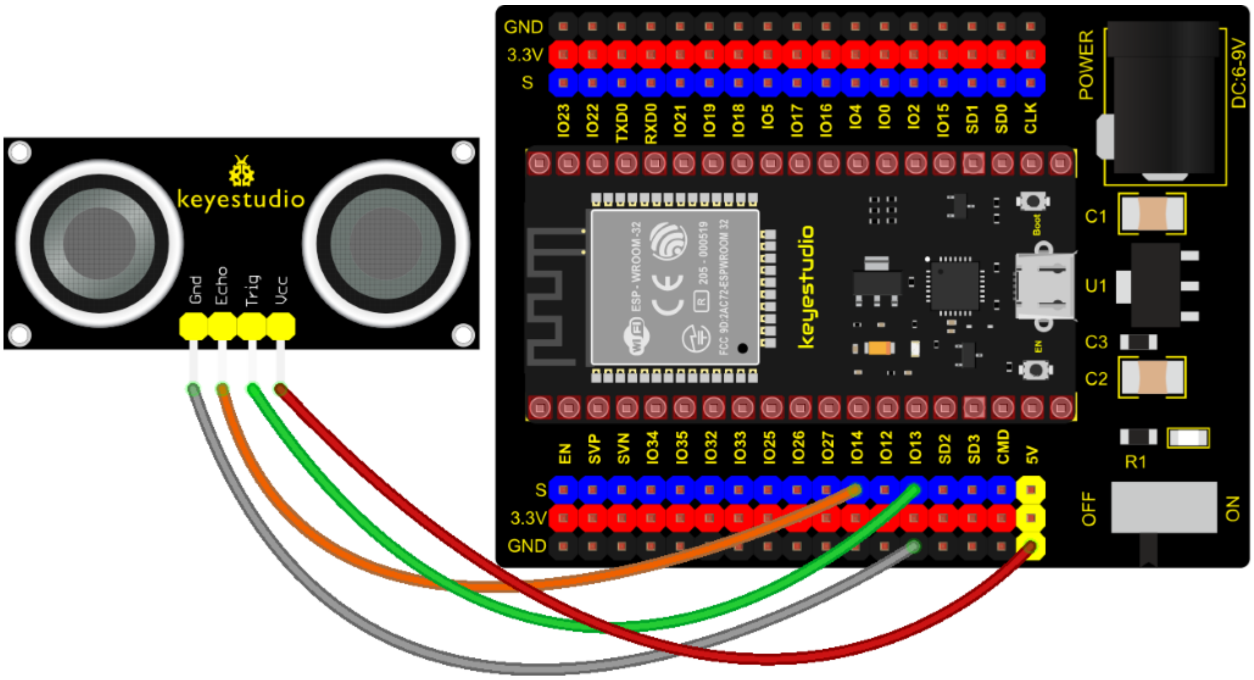
3). If there is a signal returned back, through the ECHO to output a high level, the duration time of high level is actually the time from emission to reception of ultrasonic. $TestDistance = HighLevelDuration * 340m/s * 0.5$



Components

		
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio SR01 Ultrasonic Sensor*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Ultrasonic
 * Description : Use the ultrasonic module to measure the distance.
 * Auther      : http//www.keyestudio.com
*/
const int TrigPin = 13; // define TrigPin
const int EchoPin = 14; // define EchoPin.
int duration = 0; // Define the initial value of the duration to be 0
int distance = 0;//Define the initial value of the distance to be 0
void setup()
{
  pinMode(TrigPin , OUTPUT); // set trigPin to output mode
  pinMode(EchoPin , INPUT); // set echoPin to input mode
  Serial.begin(9600);  // Open serial monitor at 9600 baud to see ping results.
}
void loop()
{
 // make trigPin output high level lasting for 10s 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
  duration = pulseIn(EchoPin , HIGH);
  // calculate the distance according to the time
  distance = (duration/2) / 28.5 ;
  Serial.print("Distance: ");
  Serial.print(distance); //Serial port print distance value
  Serial.println("cm");
```

(continues on next page)

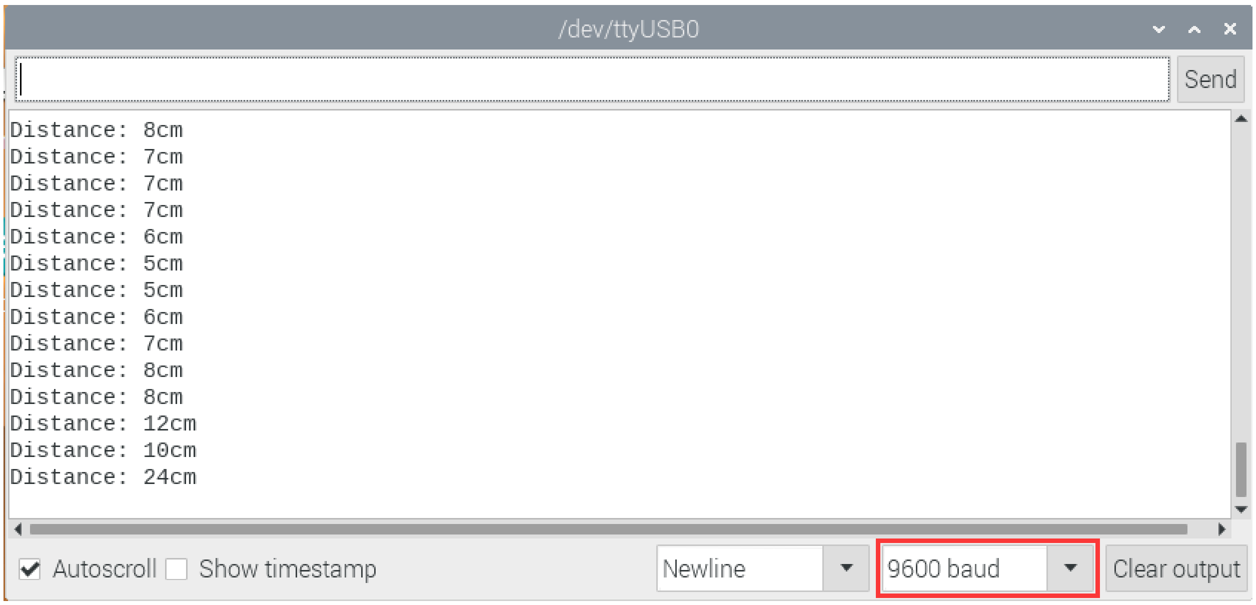
(continued from previous page)

```
  delay(300); // Wait 100ms between pings (about 20 pings/sec).
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600.

We need to press the reset button on the ESP32, then the serial monitor will print the distance between the ultrasonic sensor and the object.



7.5.36 Project 36: IR Receiver Module

Overview

Infrared remote control is currently the most widely used means of communication and remote control, which has the characteristics of small volume, low power consumption, strong function and low cost. Therefore, recorder, audio equipment, air conditioning machine and toys and other small electrical devices have also used the infrared remote control.

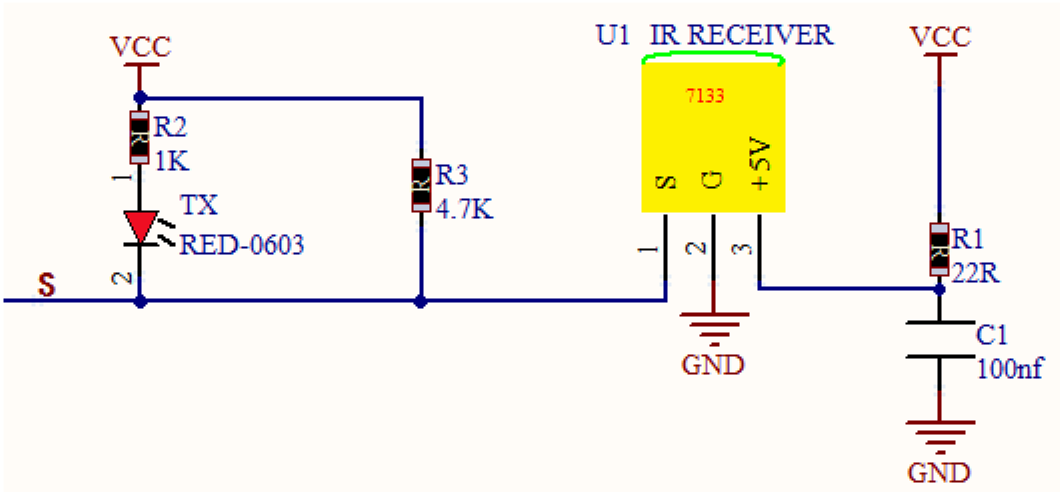
Its transmitting circuit is the use of infrared light emitting diode to emit modulated infrared light wave. The circuit is composed of infrared receiving diode, triode or silicon photocell. They convert infrared light emitted by infrared emitter into corresponding electrical signal, and then send back amplifier.

In this experiment, we need to know how to use the infrared receiving sensor. The infrared receiving sensor mainly uses the VS1838B infrared receiving sensor element. It integrates receiving, amplifying, and demodulating. The internal IC has already completed the demodulation, and the output is a digital signal. It can receive 38KHz modulated remote control signal.

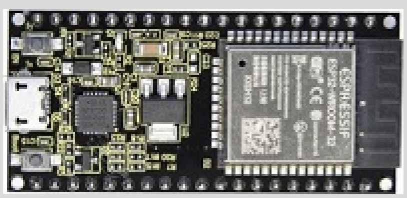
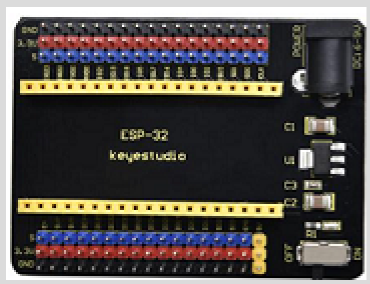
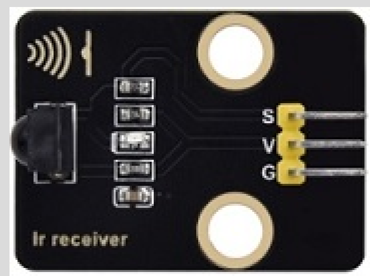
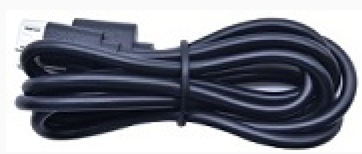
In the experiment, we use the IR receiver to receive the infrared signal emitted by the external infrared transmitting device, and display the received signal in the shell.

Working Principle

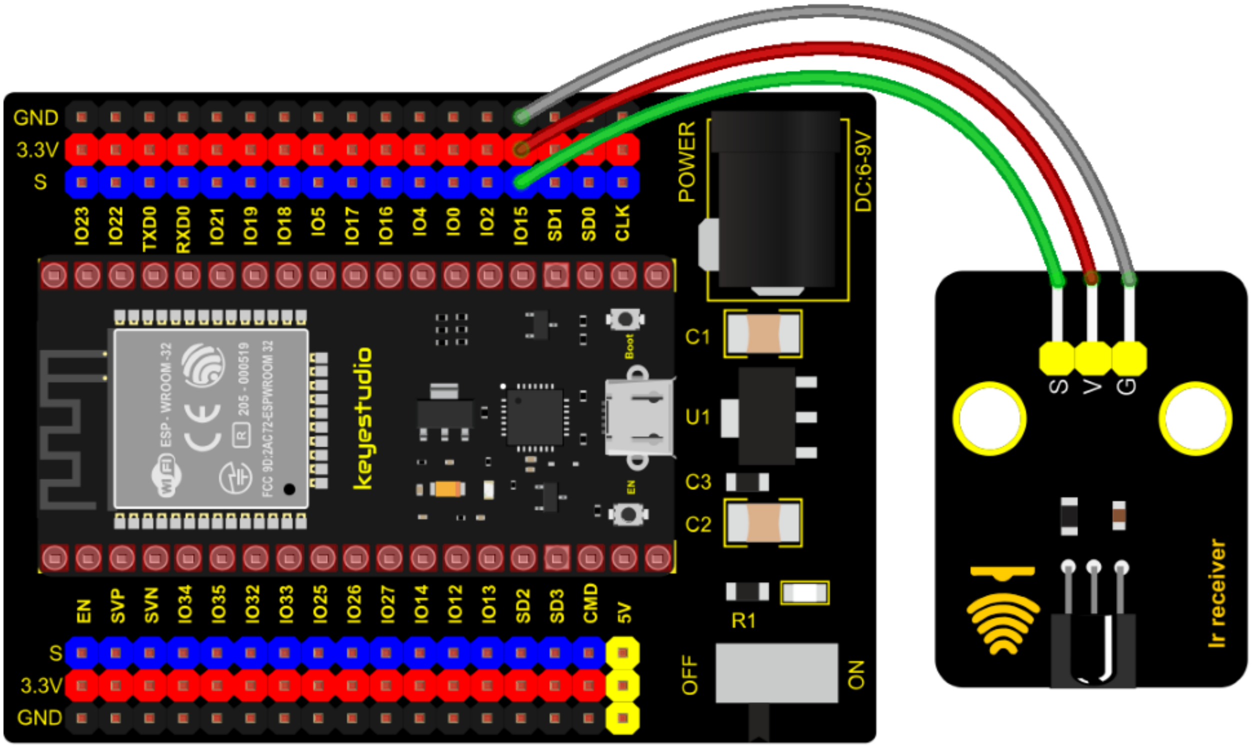
The main part of the IR remote control system is modulation, transmission and reception. The modulated carrier frequency is generally between 30khz and 60khz, and most of them use a square wave of 38kHz and a duty ratio of 1/3. A 4.7K pull-up resistor R3 is added to the signal end of the infrared receiver.



Components

		
ESP32 Board*11	ESP32 Expansion Board*1	Keyestudio DIYIR Receiver*1
		
3P Dupont Wire*1	Micro USB Cable*1	Remote Control*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : IR Receiver
 * Description : Decode the infrared remote control and print it out through the serial port.
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

const uint16_t recvPin = 15; // Infrared receiving pin
IRrecv irrecv(recvPin);      // Create a class object used to receive class
decode_results results;       // Create a decoding results class object

void setup() {
  Serial.begin(9600);       // Initialize the serial port and set the baud rate to 9600
  irrecv.enableIRIn();        // Start the receiver
  Serial.print("IRrecvDemo is now running and waiting for IR message on Pin ");
  Serial.println(recvPin);   //print the infrared receiving pin
}

void loop() {
  if (irrecv.decode(&results)) {          // Waiting for decoding
    serialPrintUint64(results.value, HEX);// Print out the decoded results
    Serial.println("");
```

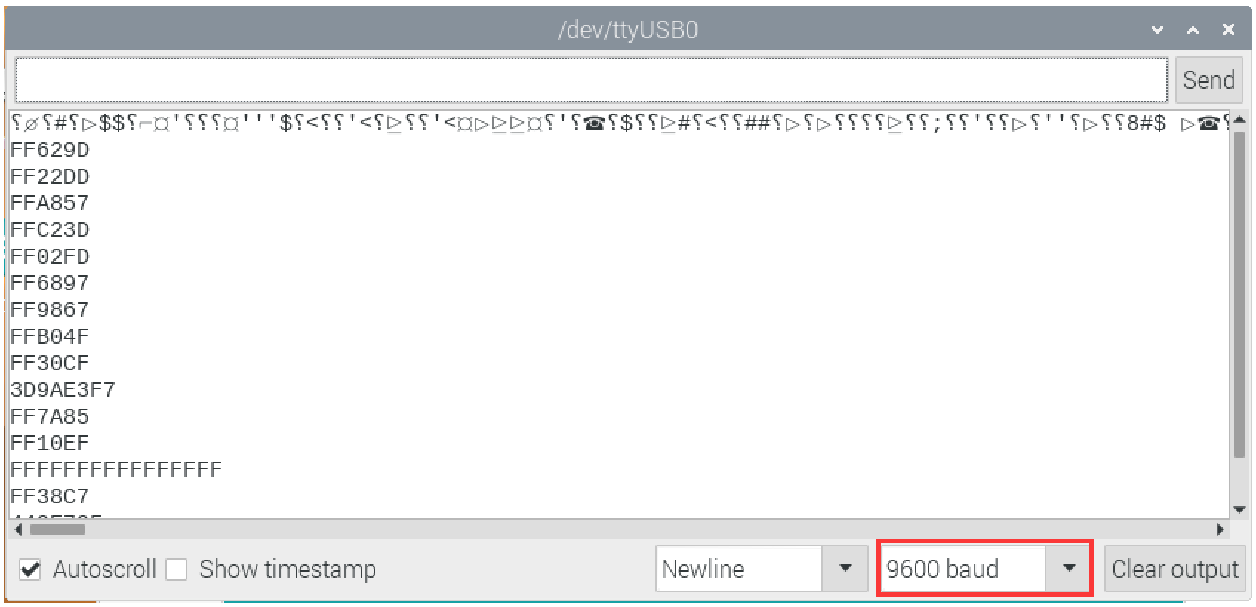
(continues on next page)

(continued from previous page)

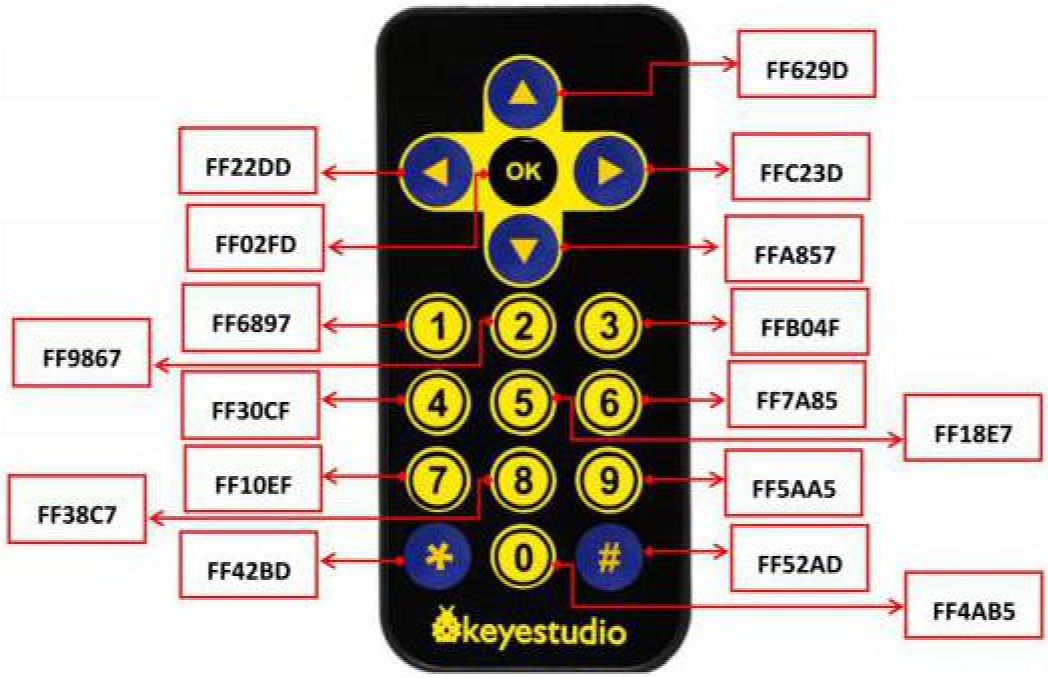
```
    irrecv.resume();                      // Release the IRremote. Receive the next value
  }
  delay(1000);
}
//*************************************************************************************
```

Test Result

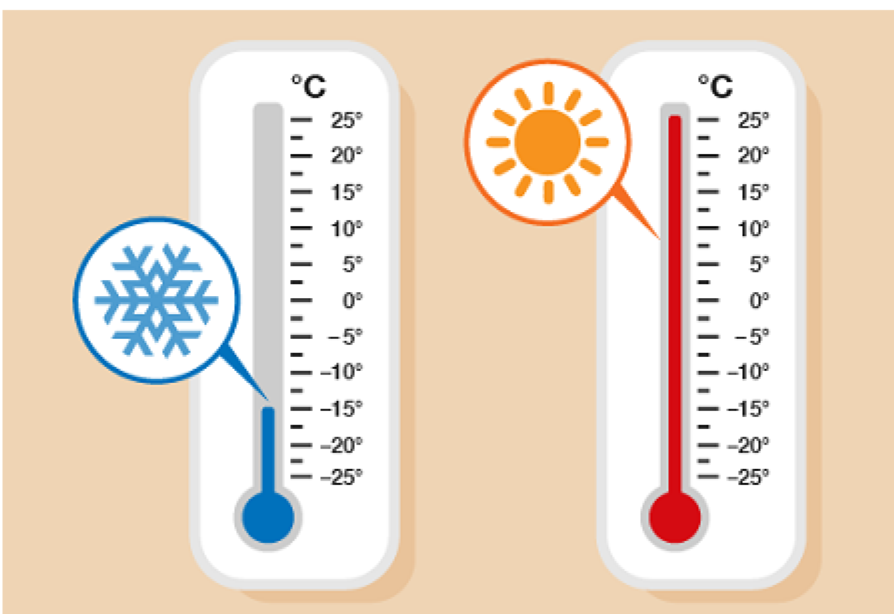
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600; Find the infrared remote control, pull out the insulating sheet, and press the button at the receiving head of the infrared receiving sensor. After receiving the signal, the LED on the infrared receiving sensor also starts to flash, as shown in the figure below.



Write down the key code value associated with the infrared remote with each key, as you will need this information later.



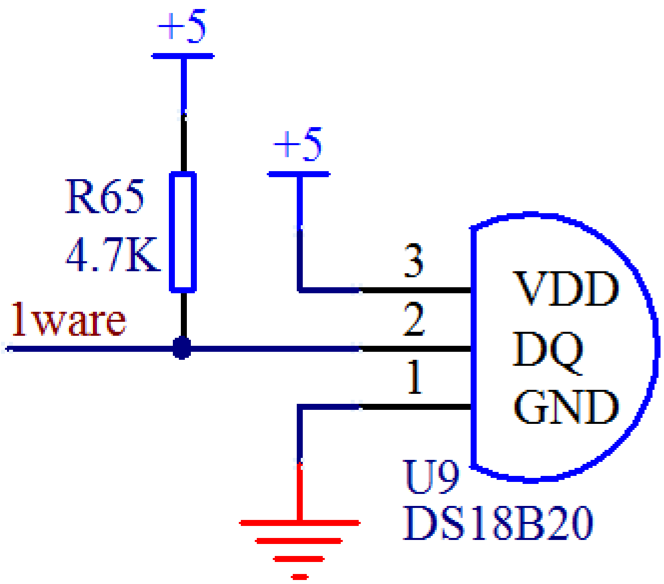
7.5.37 Project 37: DS18B20 Temperature Sensor



Description

In this kit, there is a DS18B20 temperature sensor, which is from maxim. The MCU can communicate with the DS18B20 through 1-Wire protocol, and finally read the temperature. In this experiment, we will use this temperature sensor to measure the temperature in the current environment. The test result is ℃, ranging from -55℃ to +125℃. We will display the test result on shell.

Working Principle



The hardware interface of the 1-Wire bus is very simple, just connect the data pin of the DS18B20 to an IO port of the microcontroller. The timing of the 1-Wire bus is relatively complex. Many students can't understand the timing diagram independently here. We have encapsulated the complex timing operations in the library, and you can use the library functions directly.

Schematic Diagram of DS18B20

This can save up to 12-bit temperature vale. In the register, save in code complement. As shown below;

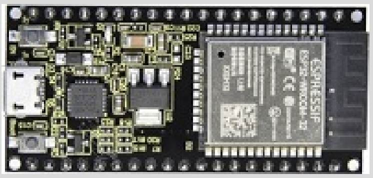
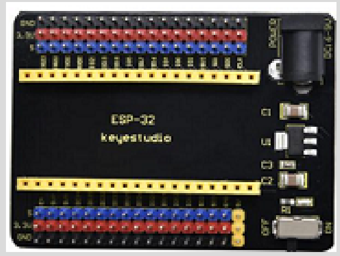
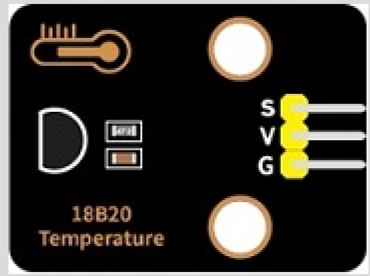
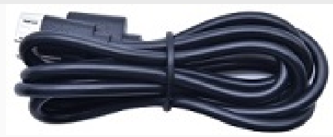
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB

MSb (unit = °C) LSb

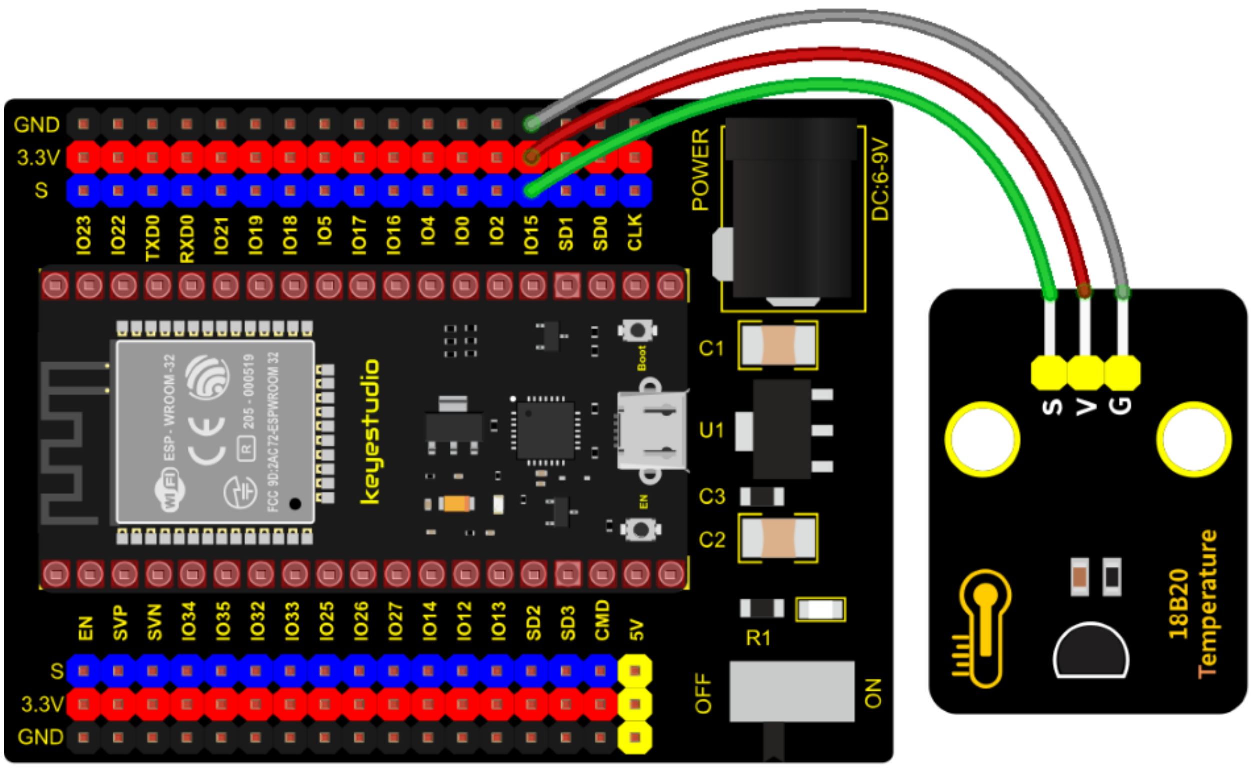
S	S	S	S	S	2^6	2^5	2^4	MSB

A total of 2 bytes, LSB is the low byte, MSB is the high byte, where MSb is the high byte of the byte, LSb is the low byte of the byte. As you can see, the binary number, the meaning of the temperature represented by each bit, is expressed. Among them, S represents the sign bit, and the lower 11 bits are all powers of 2, which are used to represent the final temperature. The temperature measurement range of DS18B20 is from -55 degrees to +125 degrees, and the expression form of temperature data, S represents positive and negative temperature, and the resolution is 2, which is 0.0625.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY 18B20 Temperature Sensor*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Required Components



Test Code

```
//**********************************************************************************
/*
 * Filename    : ds18b20
 * Description : Read the temperature of ds18B20
```

(continues on next page)

(continued from previous page)

```
 * Auther      : http//www.keyestudio.com
*/
#include <DS18B20.h>

//ds18b20 pin to 15
DS18B20 ds18b20(15);

void setup() {
  Serial.begin(9600);
}

void loop() {
  double temp = ds18b20.GetTemp();//Read the temperature
  temp *= 0.0625;//The conversion accuracy is 0.0625/LSB
  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println("C");
  delay(1000);
}
//**********************************************************************************
```

Code Explanation

1). We set the pin to GPIO15 and obtain the temperature in the unit of °C.

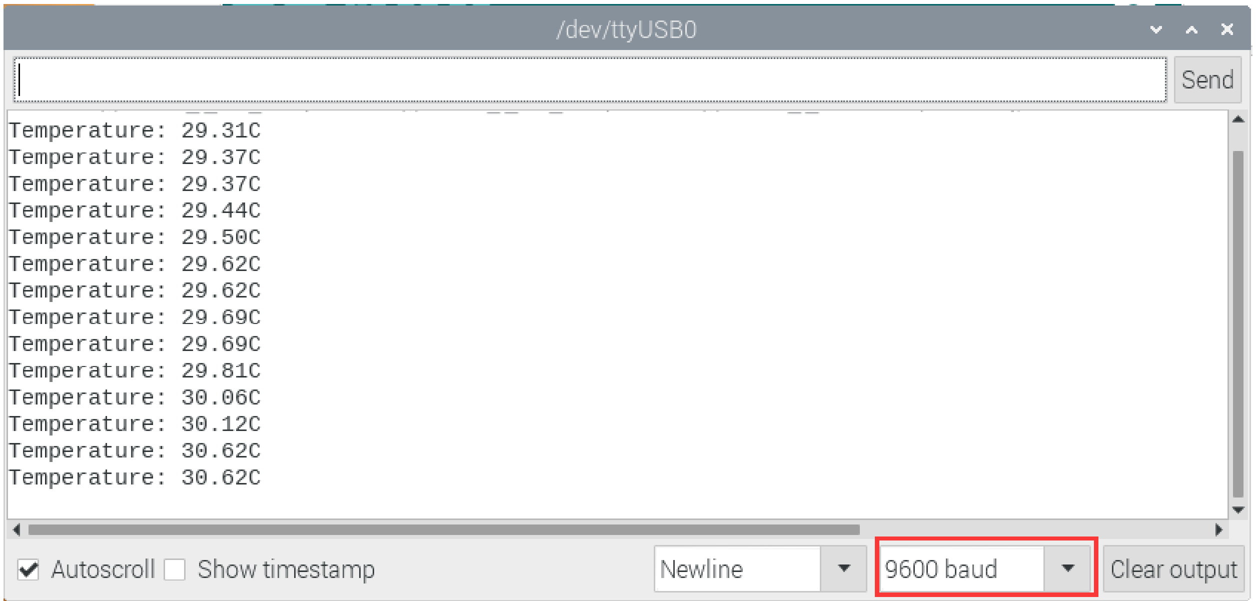
2). Set a double decimal variable to temp, and assign the measured result to temp.

3). The serial monitor displays the temp value, and the baud rate needs to be set before displaying (our default setting is 9600, which can be changed).

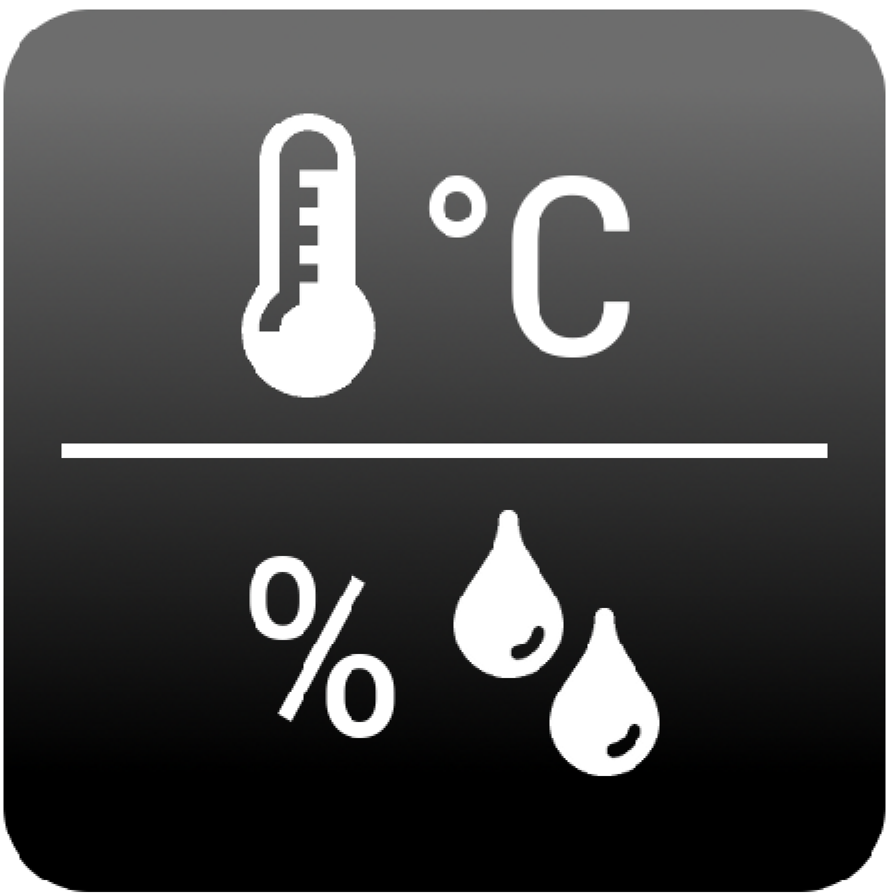
4). We add the unit behind the data. If the unit is directly set to °C, the test result will be garbled. So we directly replace °C with C.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600. We need to press the reset button on the ESP32, then the monitor will display the temperature of the current environment, as shown below.



7.5.38 Project 38: XHT11 Temperature and Humidity Sensor

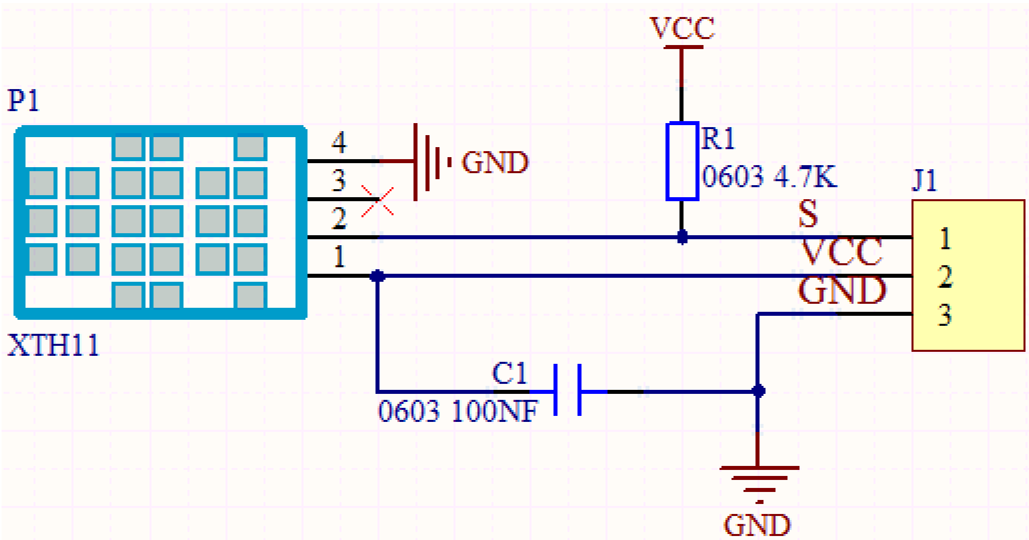


Description

This DHT11 temperature and humidity sensor is a composite sensor which contains a calibrated digital signal output of the temperature and humidity.

DHT11 temperature and humidity sensor uses the acquisition technology of the digital module and temperature and humidity sensing technology, ensuring high reliability and excellent long-term stability.

It includes a resistive element and a NTC temperature measuring device.

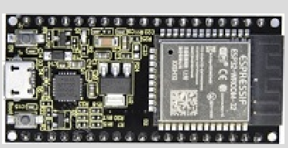
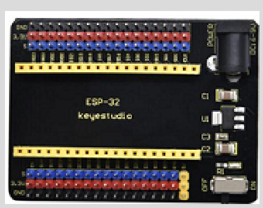
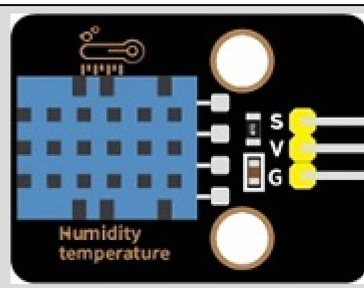


Working Principle

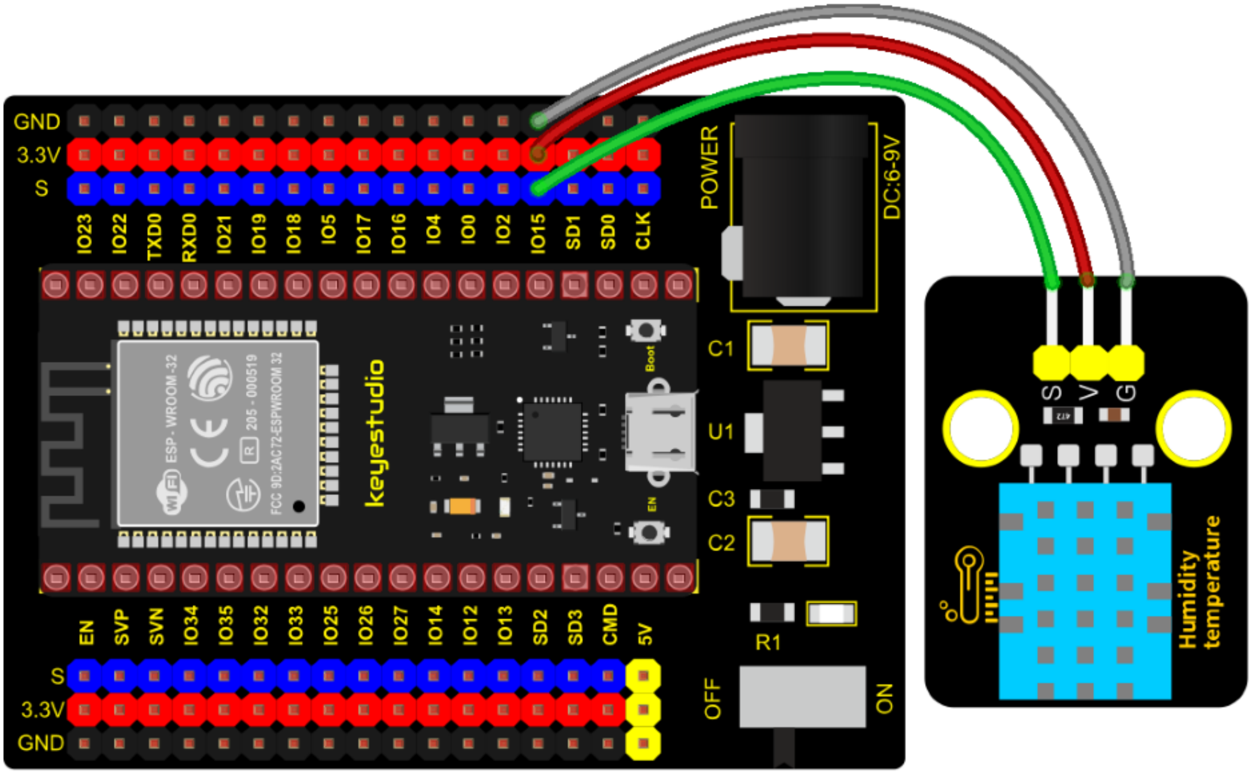
The communication and synchronization between the single-chip microcomputer and XHT11 adopts the single bus data format. The communication time is about 4ms. The data is divided into fractional part and integer part.

Operation process: A complete data transmission is 40bit, high bit first out. Data format: 8bit humidity integer data + 8bit humidity decimal data + 8bit temperature integer data + 8bit temperature decimal data + 8bit checksum 8-bit checksum: 8-bit humidity integer data + 8-bit humidity decimal data + 8-bit temperature integer data + 8-bit temperature decimal data “Add the last 8 bits of the result.

Required Components

		
ESP32Board*1	ESP32 Expansion Board*1	Keyestudio XHT11 Temperature and Humidity Sensor-compatible with DHT11)*1
		
3P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : xht11
 * Description : Read the temperature and humidity values of XHT11.
 * Auther      : http//www.keyestudio.com
*/
#include "xht11.h"
//gpio15
xht11 xht(15);

unsigned char dht[4] = {0, 0, 0, 0};//Only the first 32 bits of data are received, not the parity bits
void setup() {
  Serial.begin(9600);//Start the serial port monitor and set baud rate to 9600
}

void loop() {
  if (xht.receive(dht)) { //Returns true when checked correctly
    Serial.print("RH:");
    Serial.print(dht[0]); //The integral part of humidity, DHT [1] is the fractional part
    Serial.print("%  ");
    Serial.print("Temp:");
    Serial.print(dht[2]); //The integral part of temperature, DHT [3] is the fractional part
    Serial.println("C");
  } else {    //Read error
```

(continues on next page)

(continued from previous page)

```
    Serial.println("sensor error");
  }
  delay(1000);  //It takes 1000ms to wait for the device to read
}
//*********************************************************************************
```

Code Explanation

1). We set the pin to GPIO15, and store the detected temperature and humidity data in the dht[4] array.

2). We add units behind the data. If the temperature unit is directly set to °C, the test results may be wrong, so we directly replace °C with C; the humidity unit is directly set to %.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 9600. We need to press the reset button on the ESP32, then the monitor will display the temperature and humidity data of the current environment, as shown below.

7.5.39 Project 39: DS1307 Clock Module

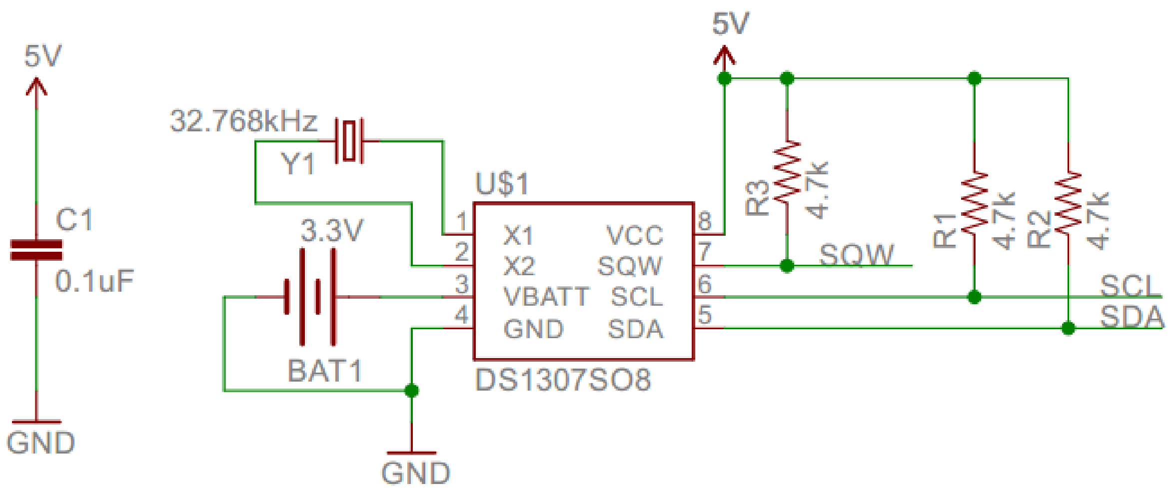
Overview

This module mainly uses the real-time clock chip DS1307, which is the I2C bus interface chip that has second, minute, hour, day, month, year and other functions as well as leap year automatic adjustment function introduced by DALLAS. It can work independently of CPU, and won't' affected by the CPU main crystal oscillator and capacitance as well as keep accurate time. What's more, monthly cumulative error is generally less than 10 s. The chip also has a clock protection circuit in case of main power failure and runs on a back-up battery that denies the CPU read and write access.

At the same time, it contains automatic switching control circuit of standby power supply, making it guarantees the accuracy of system clock in case of power failure of main power supply and other bad environment.

Going forward, the DS1307 chip internal integration has a certain capacity, with power failure protection characteristics of static RAM, which can be used to save some key data.

In the experiment, we use the DS1307 clock module to obtain the system time and print the test results.



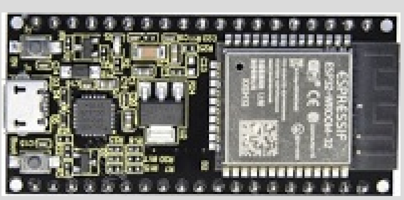
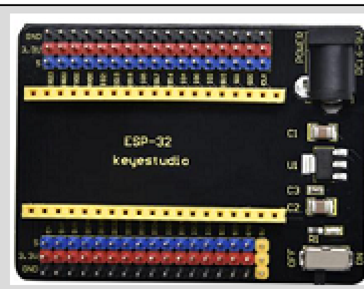
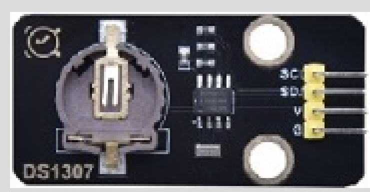
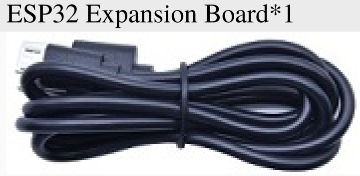
Working Principle

Serial real-time clock records year, month, day, hour, minute, second and week; AM and PM indicate morning and afternoon respectively; 56 bytes of NVRAM store data; 2-wire serial port; programmable square wave output; power failure detection and automatic switching circuit; battery current is less than 500nA.

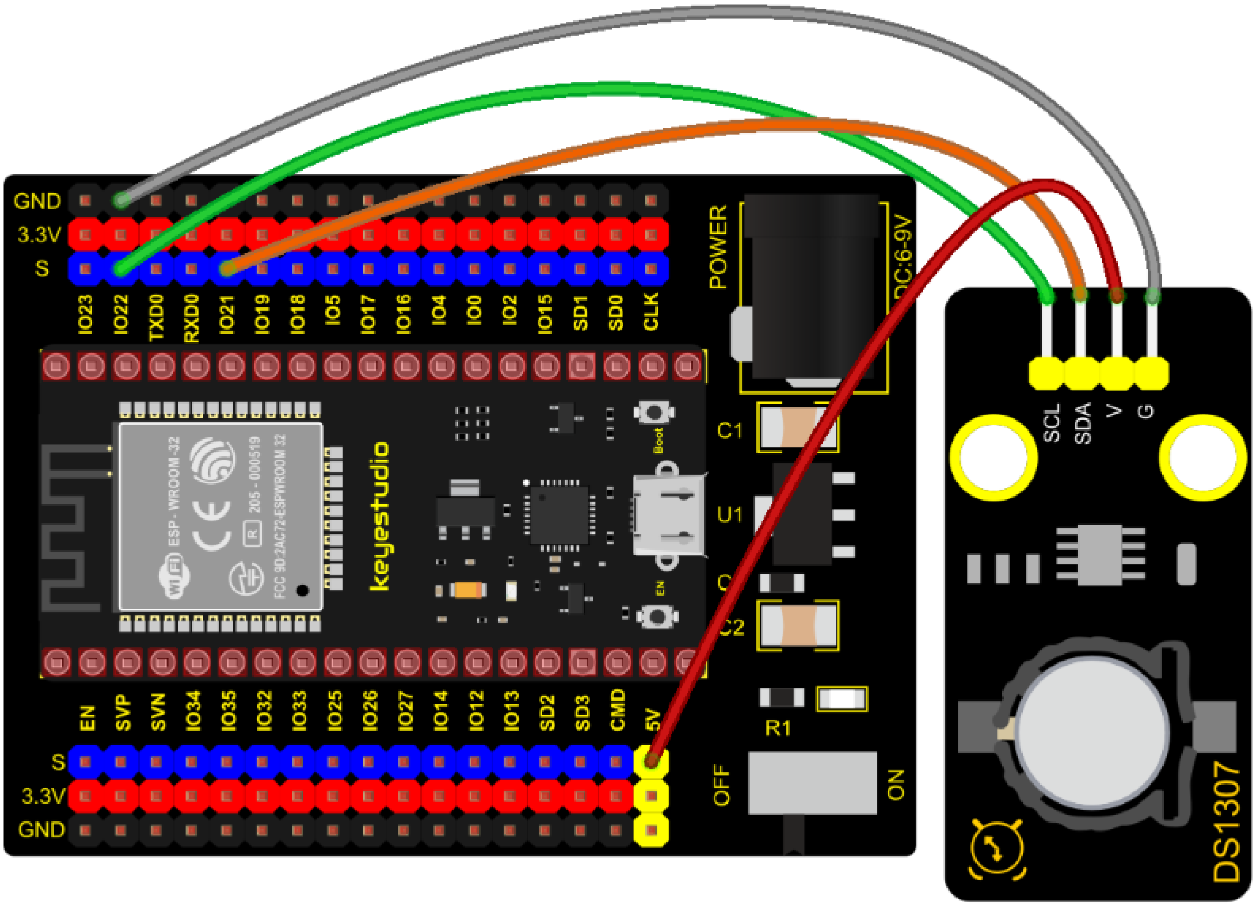
Pins description

- X1, X2: 32.768kHz crystal terminal ;
- VBAT+3V input;
- SDAserial data;
- SCLserial clock;
- SQW/OUTsquare waves/output drivers

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DS1307 Clock Module*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : DS1307 Real Time Clock
 * Description : Read the year/month/day/hour/minute/second/week of DS1307 clock module
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "RtcDS1307.h"  //DS1307 clock module library

RtcDS1307<TwoWire> Rtc(Wire);//i2cport

void setup(){
  Serial.begin(57600);//Set baud rate to 57600
  Rtc.Begin();
  Rtc.SetIsRunning(true);

  Rtc.SetDateTime(RtcDateTime(__DATE__, __TIME__));
}

void loop(){
  // Print year/month/day/hour/minute/second/week
```

(continues on next page)

(continued from previous page)

```
  Serial.print(Rtc.GetDateTime().Year());
  Serial.print("/");
  Serial.print(Rtc.GetDateTime().Month());
  Serial.print("/");
  Serial.print(Rtc.GetDateTime().Day());
  Serial.print("    ");
  Serial.print(Rtc.GetDateTime().Hour());
  Serial.print(":");
  Serial.print(Rtc.GetDateTime().Minute());
  Serial.print(":");
  Serial.print(Rtc.GetDateTime().Second());
  Serial.print("    ");
  Serial.println(Rtc.GetDateTime().DayOfWeek());
  delay(1000);//Delay 1 second
}
//*************************************************************************************
```

Code Explanation

Rtc.GetDateTime(): the obtained current time and date.

Rtc.Begin(); enable DS1307 real-time clock.

Rtc.SetIsRunning(true); run the DS1307 real-time clock, if true changes into false, time will stop.

Rtc.SetDateTime()set time.

Rtc.GetDateTime().Year(): return year.

Rtc.GetDateTime().Month(): return month.

Rtc.GetDateTime().Day(): return date.

Rtc.GetDateTime().Hour(): return hour.

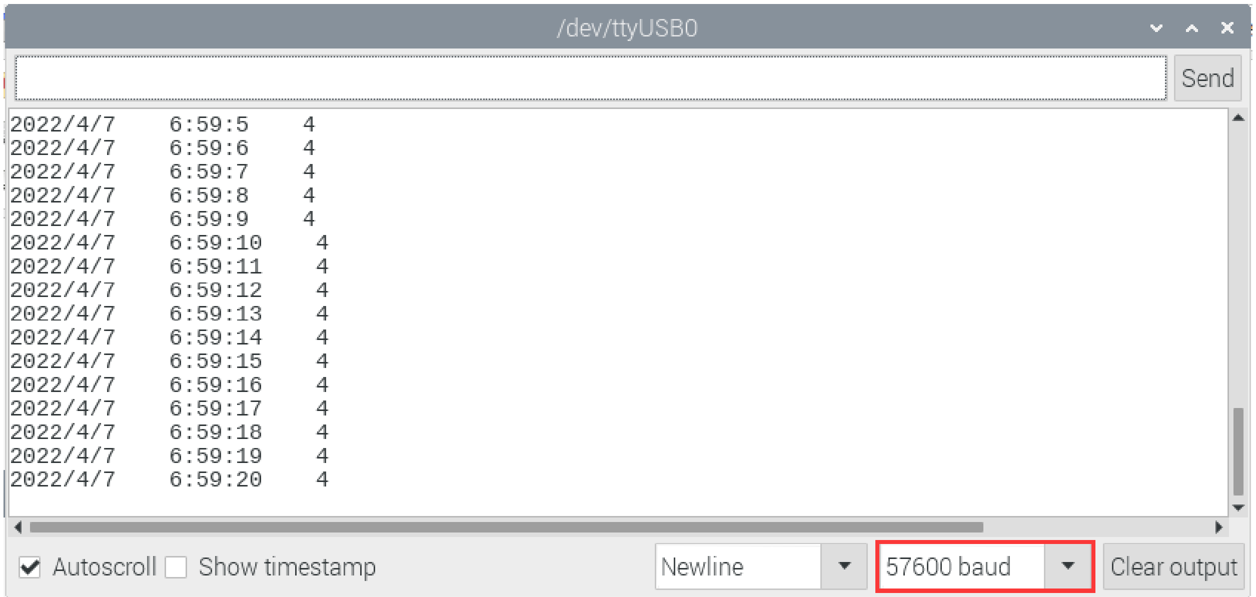
Rtc.GetDateTime().Minute(): return minute.

Rtc.GetDateTime().Second(): return second.

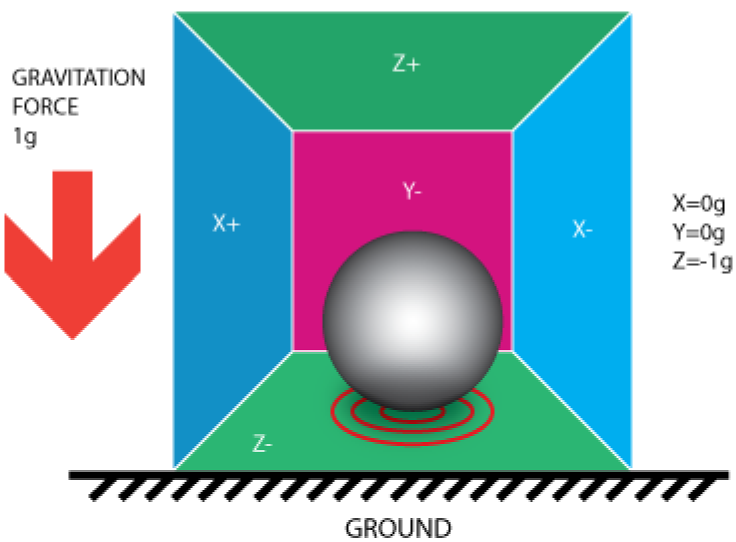
Rtc.GetDateTime().DayOfWeek(): return week.

Test Result

Connect the wires according to the experimental wiring diagram, attach the DS1307 sensor to a battery, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 57600. We need to press the reset button on the ESP32, then we can see the displayed year, month, day, hour, minute, second and week on the monitor, and set the time and date to refresh every second, as shown below:



7.5.40 Project 40: ADXL345 Acceleration Sensor



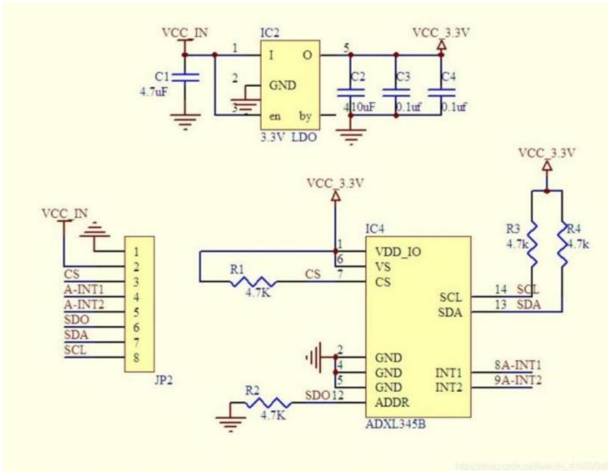
Overview

In this kit, there is a DIY electronic building block ADXL345 acceleration sensor module, which uses the ADXL345BCCZ chip. The chip is a small, thin, low-power 3-axis accelerometer with a high resolution (13 bits) and a measurement range of ±16g that can measure both dynamic acceleration due to motion or impact as well as stationary acceleration such as gravitational acceleration, making the device usable as a tilt sensor.

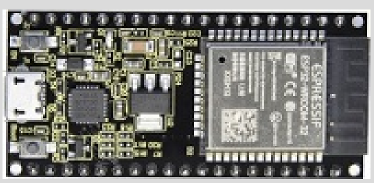
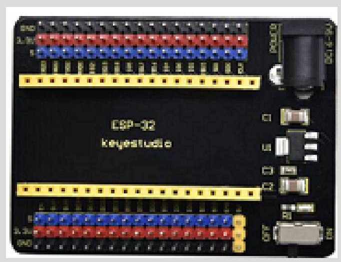
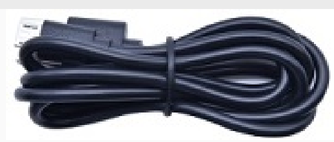
Working Principle

The ADXL345 is a complete 3-axis acceleration measurement system with a selection of measurement ranges of ±2 g, ±4 g, ±8 g or ±16 g. Its digital output data is in 16-bit binary complement format and can be accessed through an SPI (3-wire or 4-wire) or I2C digital interface.

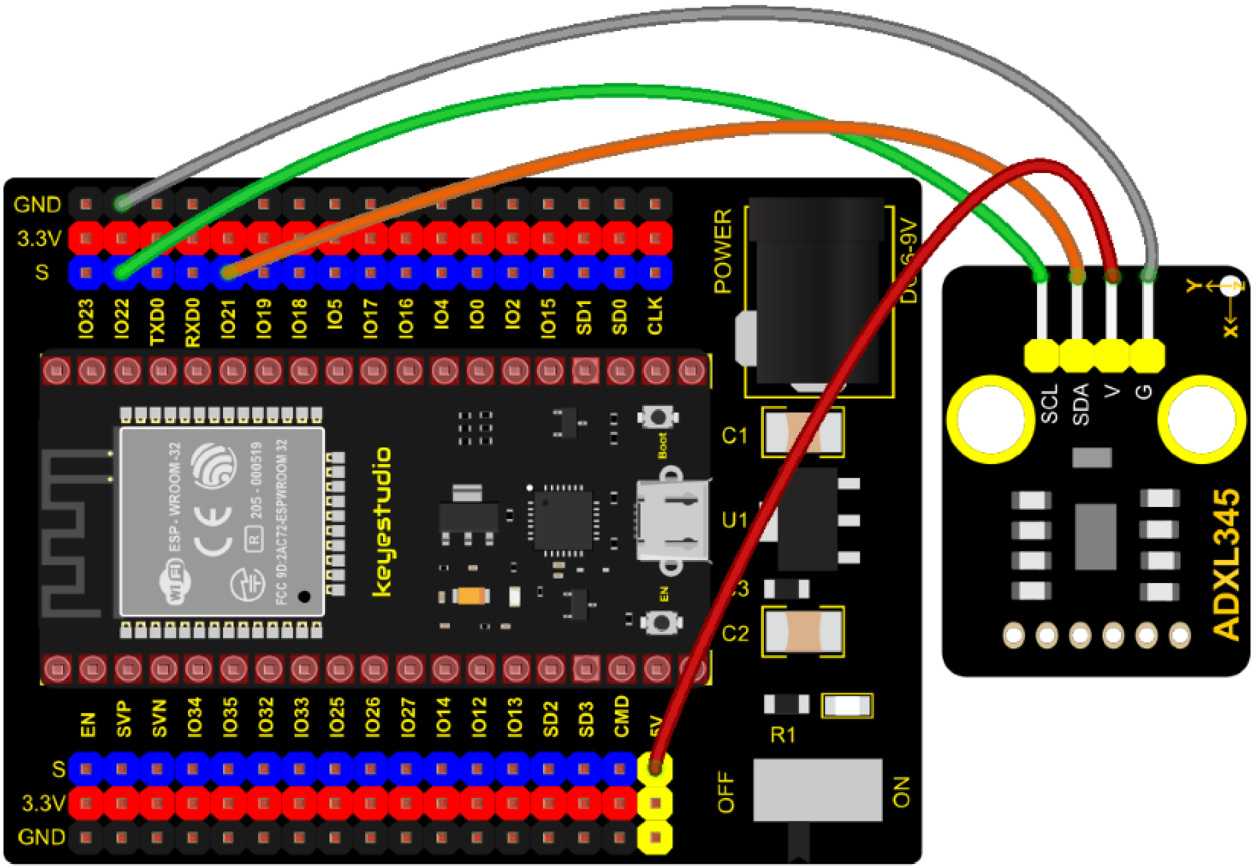
The sensor can measure static acceleration due to gravity in tilt detection applications, as well as dynamic acceleration due to motion or impact. Its high resolution (3.9mg/LSB) enables measurement of tilt Angle changes of less than 1.0°.



Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio ADXL345 Acceleration Module*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : ADXL345
 * Description : Read the X/Y/Z value of ADXL345
 * Auther      : http//www.keyestudio.com
*/
#include "adxl345_io.h"
//The port is sda-->21,scl-->22
adxl345 adxl345(21, 22);

float out_X, out_Y, out_Z;

void setup() {
  Serial.begin(57600);//Start serial port monitoring and set baud rate to 57600
  adxl345.Init();
}

void loop() {
  adxl345.readXYZ(&out_X, &out_Y, &out_Z);
  Serial.print(out_X);
  Serial.print("g   ");
  Serial.print(out_Y);
```

(continues on next page)

(continued from previous page)

```
  Serial.print("g   ");
  Serial.print(out_Z);
  Serial.println("g");
  delay(100);
}
//*************************************************************************************
```

Code Explanation

Set 3 decimal variables out_X out_Y out_Z, and assign the measured result to out_X out_Y out_Z. The serial monitor displays the value of out_X out_Y out_Z, and the baud rate needs to be set before displaying (our default setting is 9600, which can be changed).

Adxl345.Init; Initialize the ADXX345 accelerometer

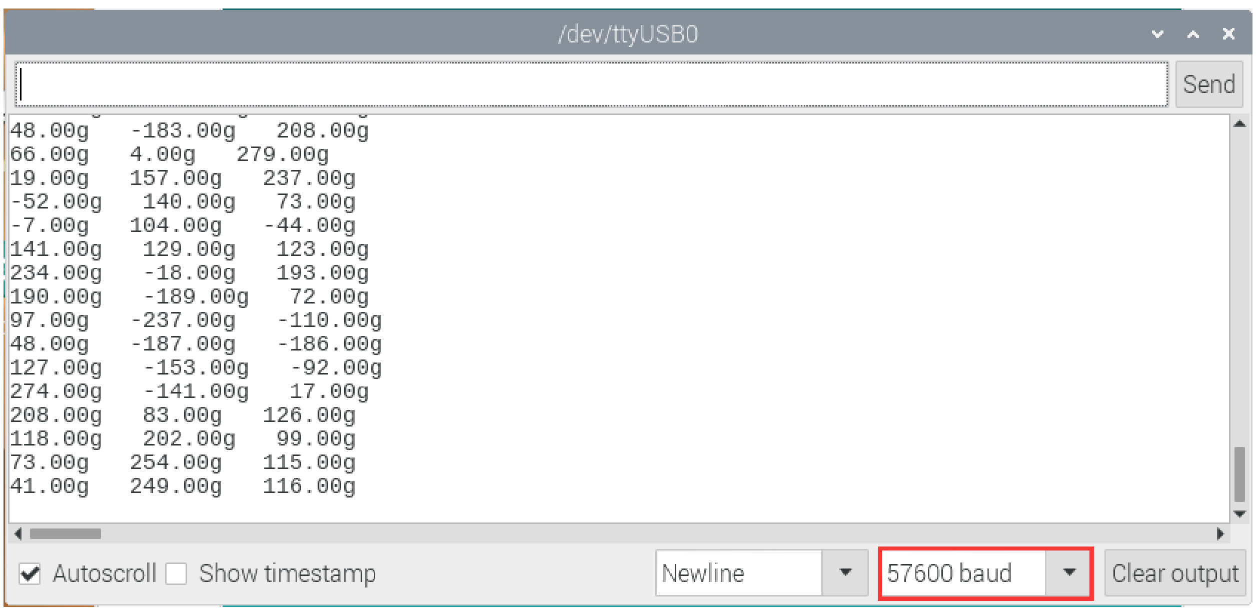
adxl345.readXYZ(&out_X, &out_Y, &out_Z);**

Get the acceleration value of the X axis and return it to the variables out_X, out_Y, out_Z

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set baud rate to 57600.

We need to press the reset button on the ESP32, then the serial monitor displays the value corresponding to the sensor, the unit is mg, as shown in the figure below.



7.5.41 Project 41: TM1650 4-Digit Tube Display

Overview This module is mainly composed of a 0.36 inch red common cathode 4-digit digital tube, and its driver chip is TM1650. When using it, we only need two signal lines to make the single-chip microcomputer control a 4-bitdigit tube, which greatly saves the IO port resources of the control board.

TM1650 is a special circuit for LED (light emitting diode display) drive control. It integrates MCU input and output control digital interface, data latch, LED drivers, keyboard scanning, brightness adjustment and other circuits.

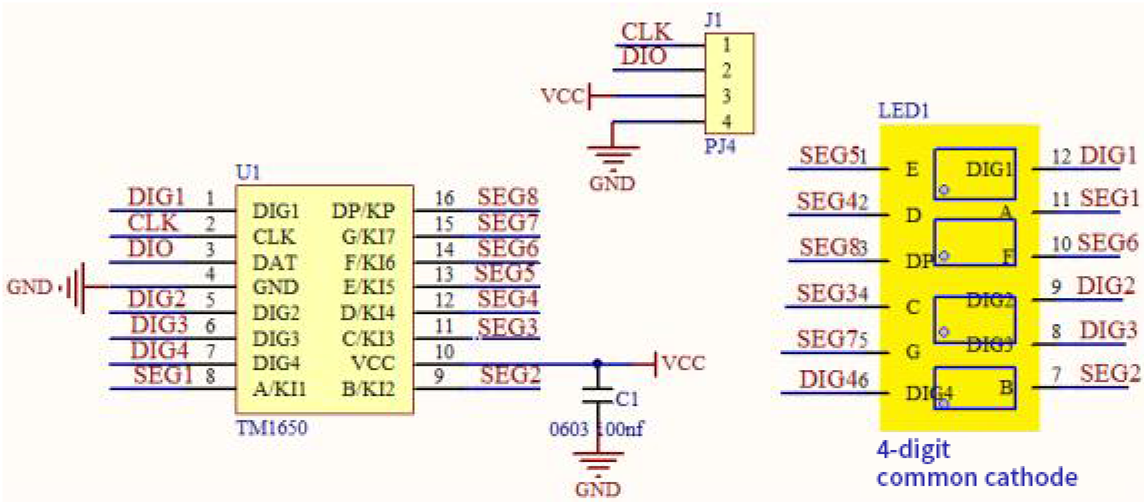
TM1650 has stable performance, reliable quality and strong anti-interference ability.

It can be applied to the application of long-term continuous working for 24 hours.

TM1650 uses 2-wire serial transmission protocol for communication (note that this data transmission protocol is not a standard I2C protocol). The chip can drive the digital tube and save MCU pin resources through two pins and MCU communication.

Working Principle

TM1650 adopts IIC treaty, which uses DIO and CLK buses.



Data command setting: 0x48 means that we light up the digital tube, instead of enable the function of key scanning

B7	B6	B5	B4	B3	B2	B1	B0	Function	Description
×	0	0	0		×	×		Brightness setting	Eight-level brightness
×	0	0	1		×	×			One-level brightness
×	0	1	0		×	×			Two-level brightness
×	0	1	1		×	×			Three-level brightness
×	1	0	0		×	×			Four-level brightness
×	1	0	1		×	×			Five-level brightness
×	1	1	0		×	×			Six-level brightness
×	1	1	1		×	×			Seven-level brightness
×				0	×	×		7/8 segment display control bit	8-segment display way
×				1	×	×			7-segment display way
×					×	×	0	ON/OFF display bit	Off display
×					×	×	1		On display

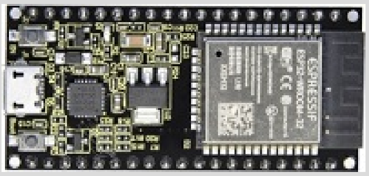
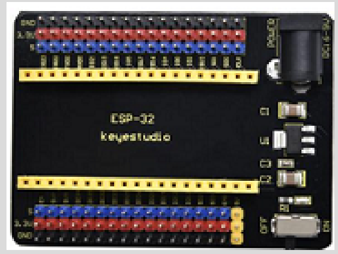
Command display setting:

bit[6:4]: set the brightness of tube display, and 000 is brightest

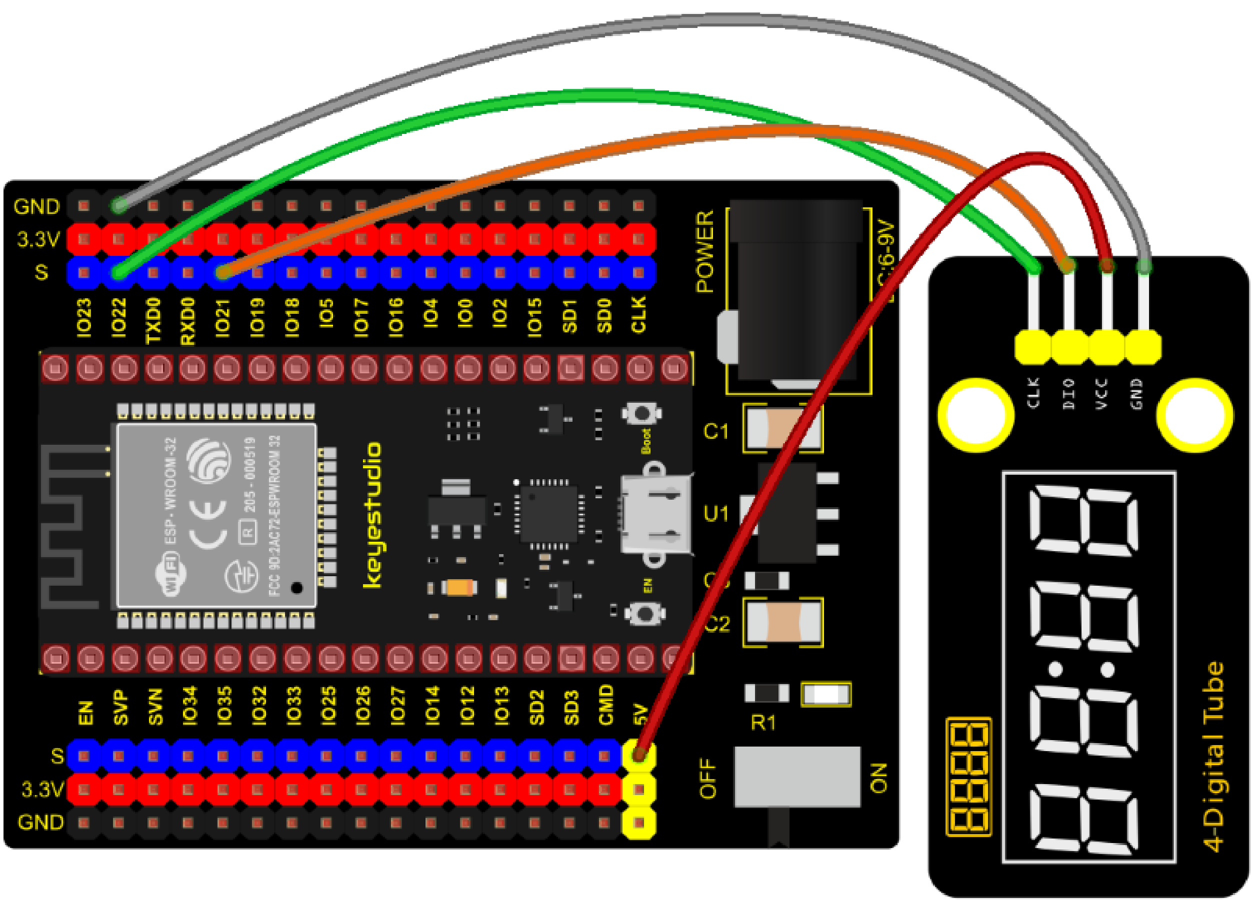
bit[3]: set to show decimal points

bit[0]: start the display of the tube display

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio TM16504-Digit Segment Display*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**************************************************************************************
/*
 * Filename    : TM1650 Four digital tube
 * Description : TM1650 Four Digital Tube shows 0-9999
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h"
#define CLK 22    //pins definitions for TM1650 and can be changed to other ports
#define DIO 21
TM1650 DigitalTube(CLK,DIO);

void setup(){
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
}

void loop(){
  for(int num=0; num<10000; num++){
    displayFloatNum(num);
    delay(100);
  }
}

void displayFloatNum(float num){
  if(num > 9999)
    return;
  int dat = num*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
```

(continues on next page)

(continued from previous page)

```
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*********************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The 4-digit tube display will show integer from 0 to 99999, add 1 for each 10ms. Increase to 9999 then start from 0.

7.5.42 Project 42: HT16K33_8X8 Dot Matrix Module

Overview

What is the dot matrix display?

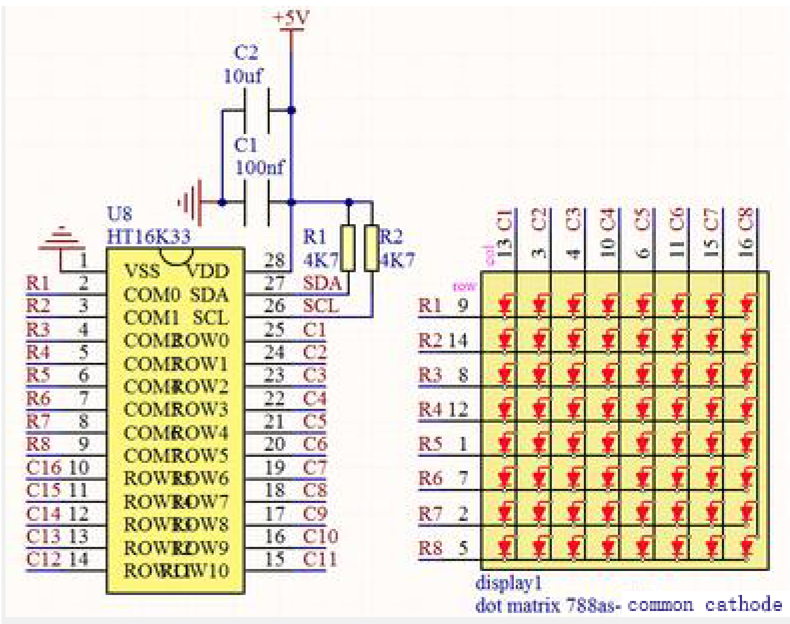
If we apply the previous circuit, there will be must one IO port to control only one LED. When more LED need to be controlled, we may adopt a dot matrix. The 8X8 dot matrix is composed of 64 light-emitting diodes, and each light-emitting diode is placed at the intersection of the row line and the column line. Refer to the experimental schematic diagram below, when the corresponding column is set to a high level and a certain row to low, the corresponding diode will light up… For instance, set pin 13 to a high level and pin 9 to low, and then the first LED will light up. In the experiment, we display icons via this dot matrix.

Working Principle

As the schematic diagram shown, to light up the LED at the first row and column, we only need to set C1 to high level and R1 to low level. To turn on LEDs at the first row, we set R1 to low level and C1-C8 to high level.

16 IO ports are needed, which will highly waste the MCU resources.

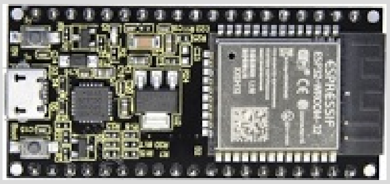
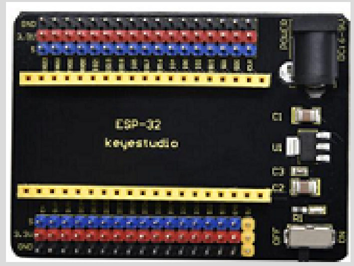
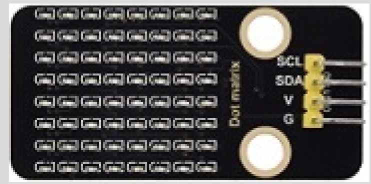
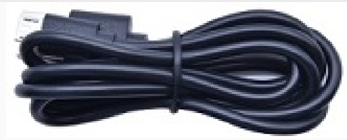
Therefore, we designed this module, using the HT16K33 chip to drive an 8*8 dot matrix, which greatly saves the resources of the single-chip microcomputer.



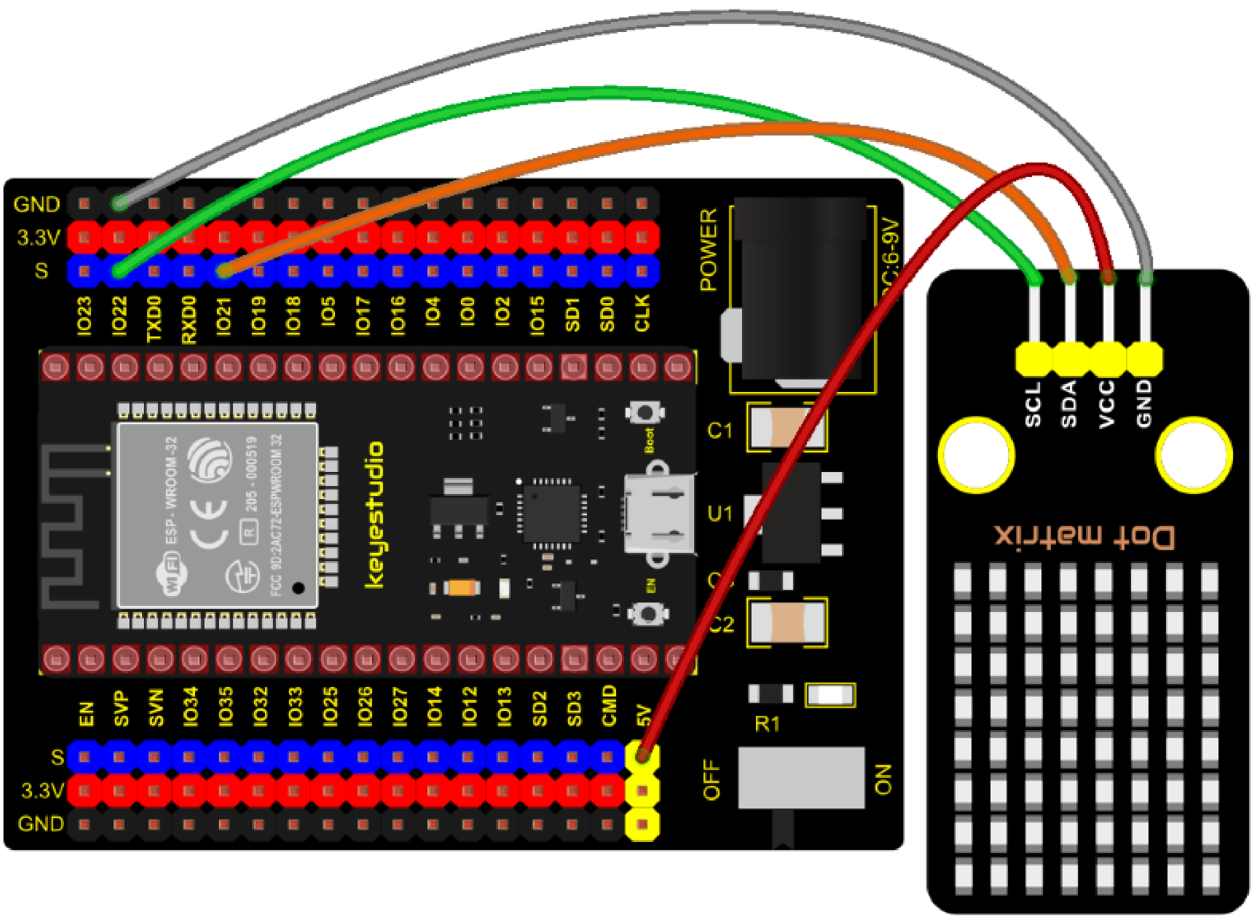
There are three DIP switches on the module, all of which are set to I2C communication address. The setting method is shown below. A0A1 and A2 are grounded, that is, the address is 0x70.

A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	0（OFF）
0X70			0X71			0X72		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）
1（ON）	1（ON）	0（OFF）	0（OFF）	0（OFF）	1（ON）	1（ON）	0（OFF）	1（ON）
0X73			0X74			0X75		
A0（1）	A1（2）	A2（3）	A0（1）	A1（2）	A2（3）			
0（OFF）	1（ON）	1（ON）	1（ON）	1（ON）	1（ON）			
0X76			0X77					

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HT16K33_8X8 Dot Matrix*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : 8×8 Dot-matrix Display
 * Description : 8x8 LED dot matrix display“Heart” pattern.
 * Auther      : http//www.keyestudio.com
*/
#include "HT16K33_Lib_For_ESP32.h"

#define SDA 21
#define SCL 22

ESP32_HT16K33 matrix = ESP32_HT16K33();

//The brightness values can be set from 1 to 15, with 1 darkest and 15 brightest
#define  A  15

byte result[8][8];
byte test1[8] = {0x00,0x42,0x41,0x09,0x09,0x41,0x42,0x00};

void setup()
{
  matrix.init(0x70, SDA, SCL);//Initialize matrix
  matrix.showLedMatrix(test1,0,0);
  matrix.show();
}

void loop()
{
  for (int i = 0; i <= 7; i++)
  {
    matrix.setBrightness(i);
    delay(100);
  }
  for (int i = 7; i > 0; i--)
  {
    matrix.setBrightness(i);
    delay(100);
  }
}
//**********************************************************************************
```

Code Explanation

First we need to import the library file.

The pattern in our code is an array of byte data type, which is shown in the table below. We convert into binary, and fill in the 8*8 form below to make it clear. 1 means on, 0 means off. Then we can see that it is a smile shape.

0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Then the dot matrix displays a " smile" pattern.

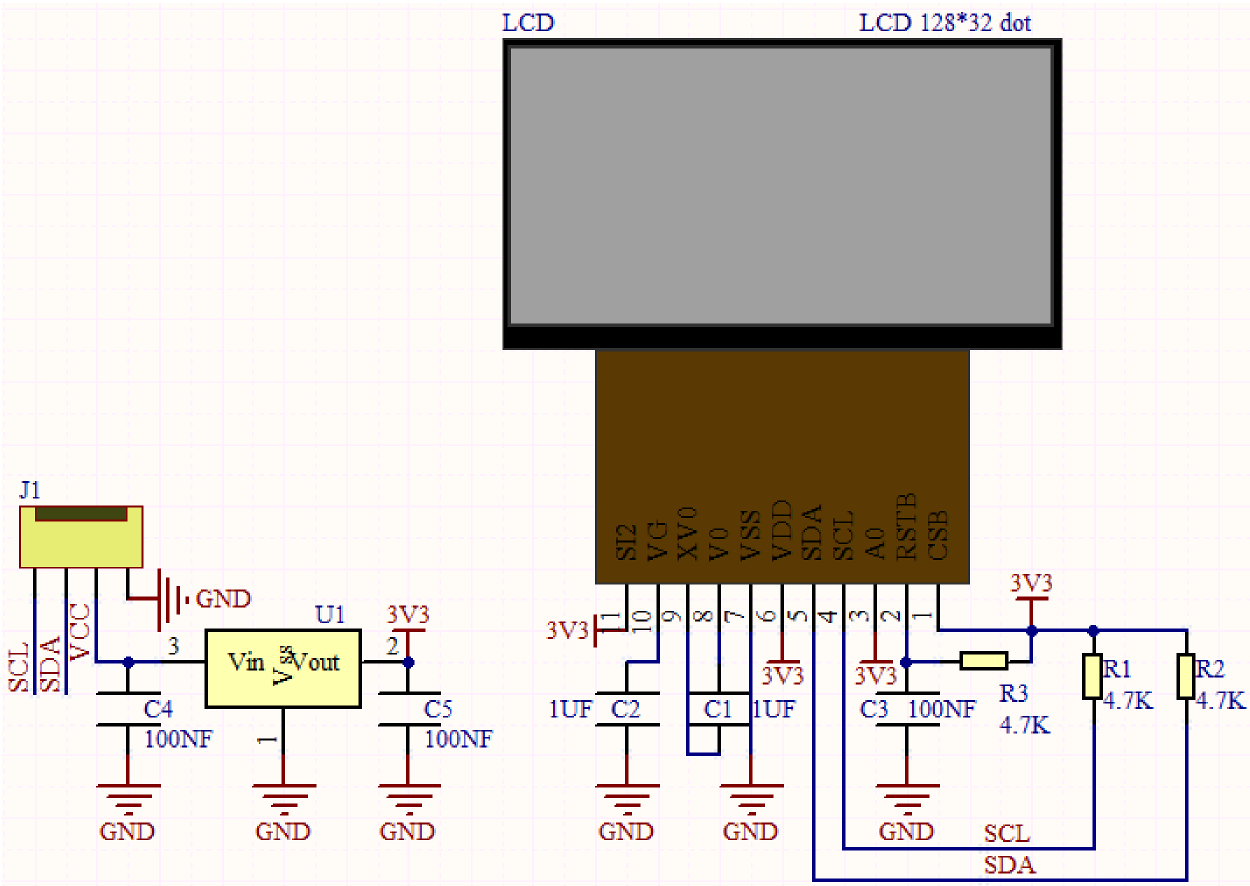
7.5.43 Project 43: LCD_128X32_DOT Module

Description

This is a 128*32 pixel LCD module, which uses IIC communication mode and ST7567A driver chip . At the same time, the code contains all the English letters and common symbols of the library that can be directly called. When used, we can also set English letters and symbols to display different text sizes in our code. To make it easy to set up the pattern display, we also provide a mold capture software that can convert a specific pattern into control code and then copy it directly into the test code for use.

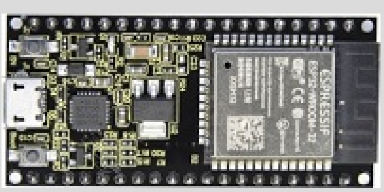
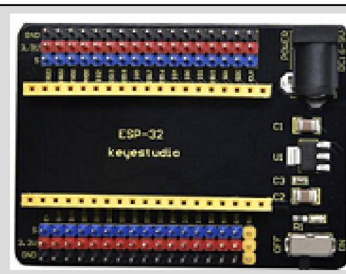
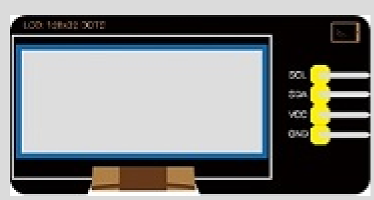
In the experiment, we will set up the display screen to display various English words, common symbols and numbers.

Working Principle

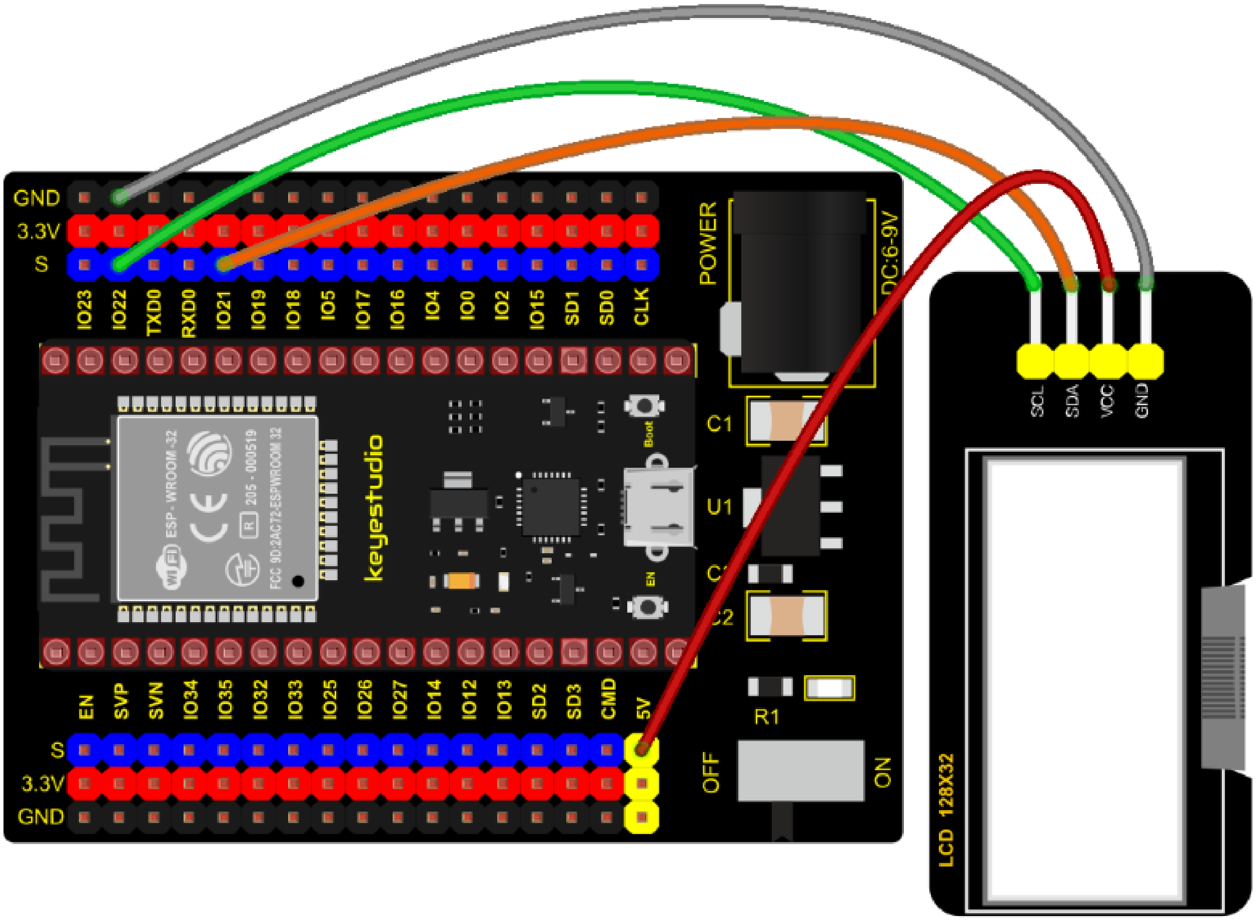


The module uses the IIC communication principle, the underlying functions have been encapsulated in the library surface, we can directly call the library function, if interested, you can also go to understand the underlying driver of the module.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio LCD_128X32_DOT Module*1
		
4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
* Filename    : lcd128*32
* Description : Lcd128 *32 Displays character strings
* Auther      : http//www.keyestudio.com
*/
#include "lcd128_32_io.h"

//Create lcd12832 examples,sda--->21 scl--->22
lcd lcd(21, 22);

void setup() {
    lcd.Init(); //initialize
    lcd.Clear();  //cls
  }

void loop() {
    lcd.Cursor(0, 7); //Set display position
    lcd.Display("KEYES"); //Setting the display
```

(continues on next page)

(continued from previous page)

```
    lcd.Cursor(1, 0);
    lcd.Display("ABCDEFGHIJKLMNOPQR");
    lcd.Cursor(2, 0);
    lcd.Display("123456789+-*/<>=$@");
    lcd.Cursor(3, 0);
    lcd.Display("%^&(){}:;'|?,.~\\[]");
  }
//*********************************************************************************
```

Code Explanation

First import the library file

.Init(): initializes the display screen;

.Clear(): clears the display;

.Cursor(): sets the display position;

.Display(): displays characters.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32.

After uploading successfully, we will use a USB cable to power on. The first line of the 128X32LCD module display shows "KEYES", the second line shows "ABCDEFGHIJKLMNOPQR", and the third line shows "123456789±*/<> =$@", the fourth line displays "%^&(){}:;'|?,.~\[]".

7.5.44 Project 44: RFID Module

Description

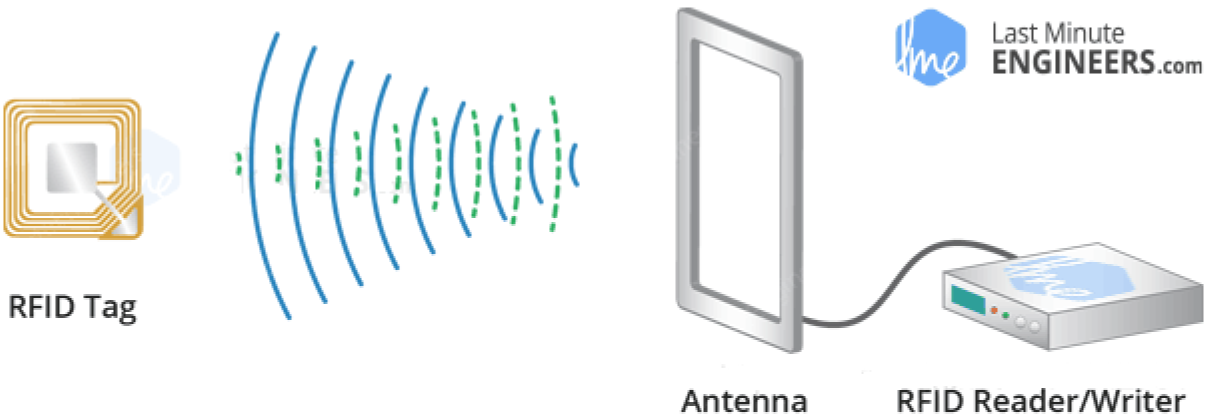
RFIDRFID-RC522 radio frequency module adopts a Philips MFRC522 original chip to design card reading circuit, easy to use and low cost, suitable for equipment development and card reader development and so on.

RFID or Radio Frequency Identification system consists of two main components, a transponder/tag attached to an object to be identified, and a transceiver also known as interrogator/Reader.

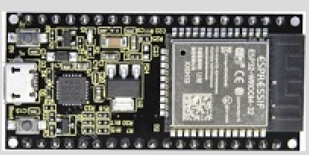
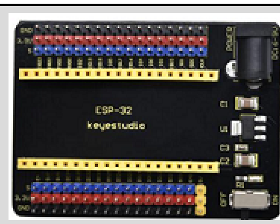
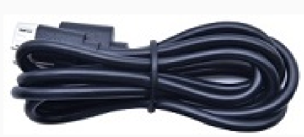
In the experiment, the data read by the card swipe module is 4 hexadecimal numbers, and we print these four hexadecimal numbers as strings. For example, we read the data of the IC card below: 0xED0xF70x940x5A and the information string displayed in the serial monitor is ED F7 94 5A ; the data read from the keychain is: 0x4C0x090x6B0x6E . Different IC cards and different key chains have diverse data.

Working Principle

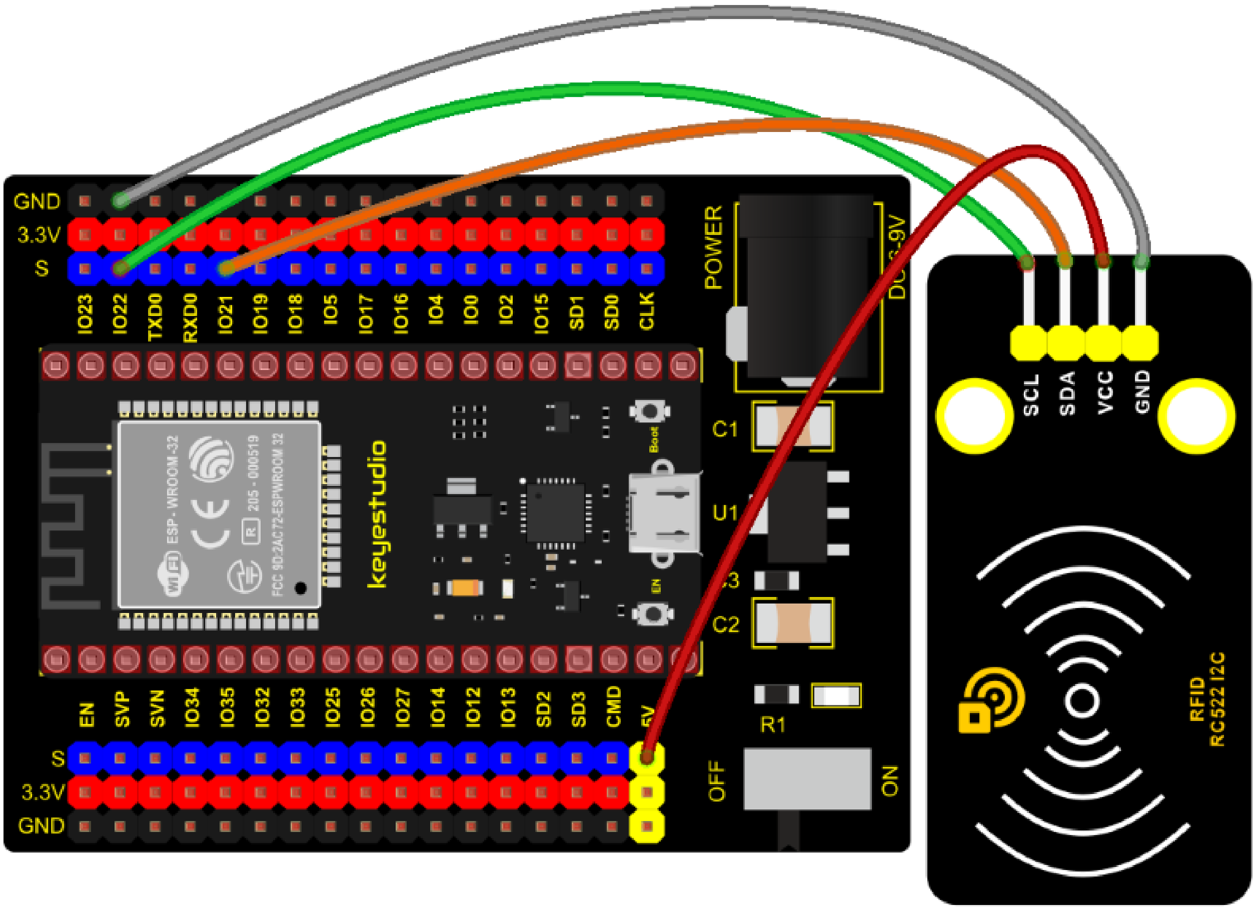
Radio frequency identification, the card reader is composed of a radio frequency module and a high-level magnetic field. The Tag transponder is a sensing device, and this device does not contain a battery. It only contains tiny integrated circuit chips and media for storing data and antennas for receiving and transmitting signals. To read the data in the tag, first put it into the reading range of the card reader. The reader will generate a magnetic field, and because the magnetic energy generates electricity according to Lenz's law, the RFID tag will supply power, thereby activating the device.



Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY RFID Module*1	4P Dupont Wire*1
			
Micro USB Cable*1	Key*1	IC Card*1	

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : RFID
 * Description : RFID reader UID
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "MFRC522_I2C.h"
// IIC pins default to GPIO21 and GPIO22 of ESP32
// 0x28 is the i2c address of SDA, if doesn't matchplease check your address with i2c.
MFRC522 mfrc522(0x28);   // create MFRC522.

void setup() {
  Serial.begin(115200);           // initialize and PC's serial communication
  Wire.begin();                   // initialize I2C
  mfrc522.PCD_Init();             // initialize MFRC522
  ShowReaderDetails();            // dispaly PCD - MFRC522 read carder
  Serial.println(F("Scan PICC to see UID, type, and data blocks..."));
}

void loop() {
```

(continues on next page)

(continued from previous page)

```
  //
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }

  // select one of door cards. UID and SAK are mfrc522.uid.

  // save UID
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println();
}

void ShowReaderDetails() {
  //  attain the MFRC522 software
  byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
  Serial.print(F("MFRC522 Software Version: 0x"));
  Serial.print(v, HEX);
  if (v == 0x91)
    Serial.print(F(" = v1.0"));
  else if (v == 0x92)
    Serial.print(F(" = v2.0"));
  else
    Serial.print(F(" (unknown)"));
  Serial.println("");
  // when returning to 0x00 or 0xFF, may fail to transmit communication signals
  if ((v == 0x00) || (v == 0xFF)) {
    Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?
↪"));
  }
}
//*****************************************************************************************
```

Code Explanation

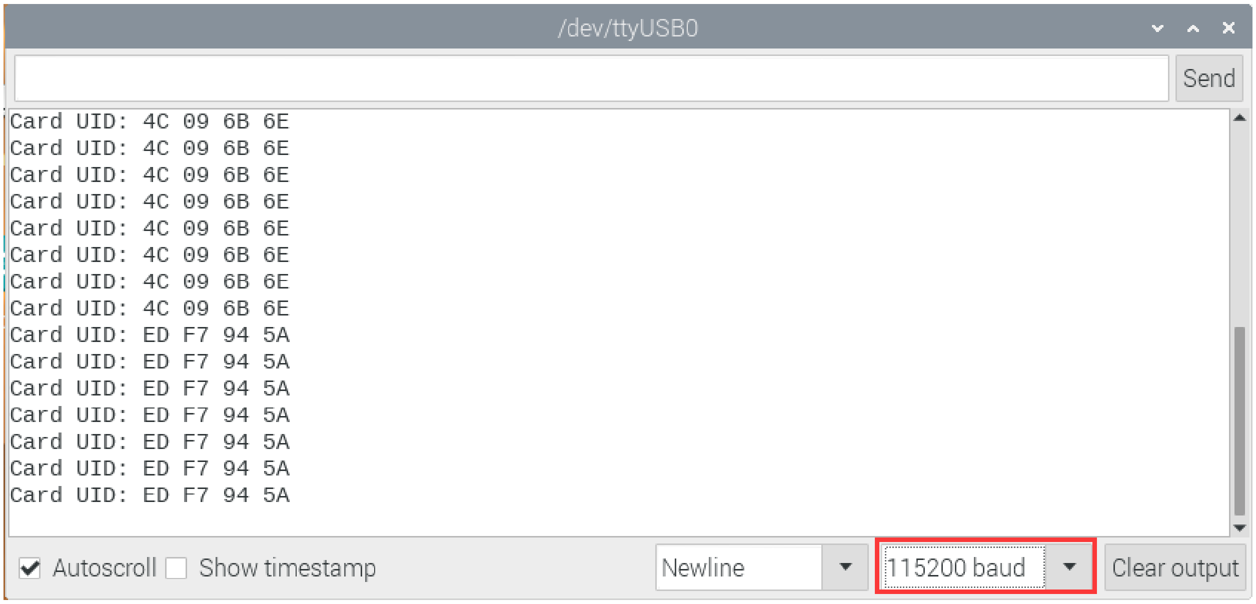
Wire.begin(); The module we use is the IIC interface, so we first initialize the IIC

mfrc522.PCD_Init(); initialize MFRC522

String(mfrc522.uid.uidByte[i], HEX); A string to convert the value read into hexadecimal format.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set baud rate to 115200. We need to press the reset button on the ESP32, when we make the IC card close to the RFID module, the information will be printed out, as shown in the figure below.



7.6 6. Comprehensive Projects:

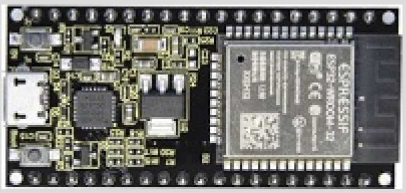
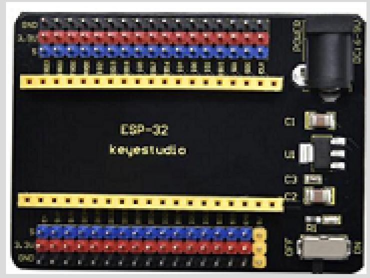
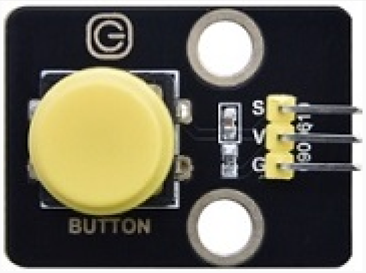
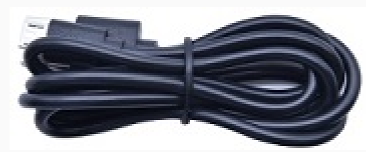
The previous projects are related to single sensor or module. In the following part, we will combine various sensors and modules to create some comprehensive experiments to perform special functions.

7.6.1 Project 45: Button-controlled LED

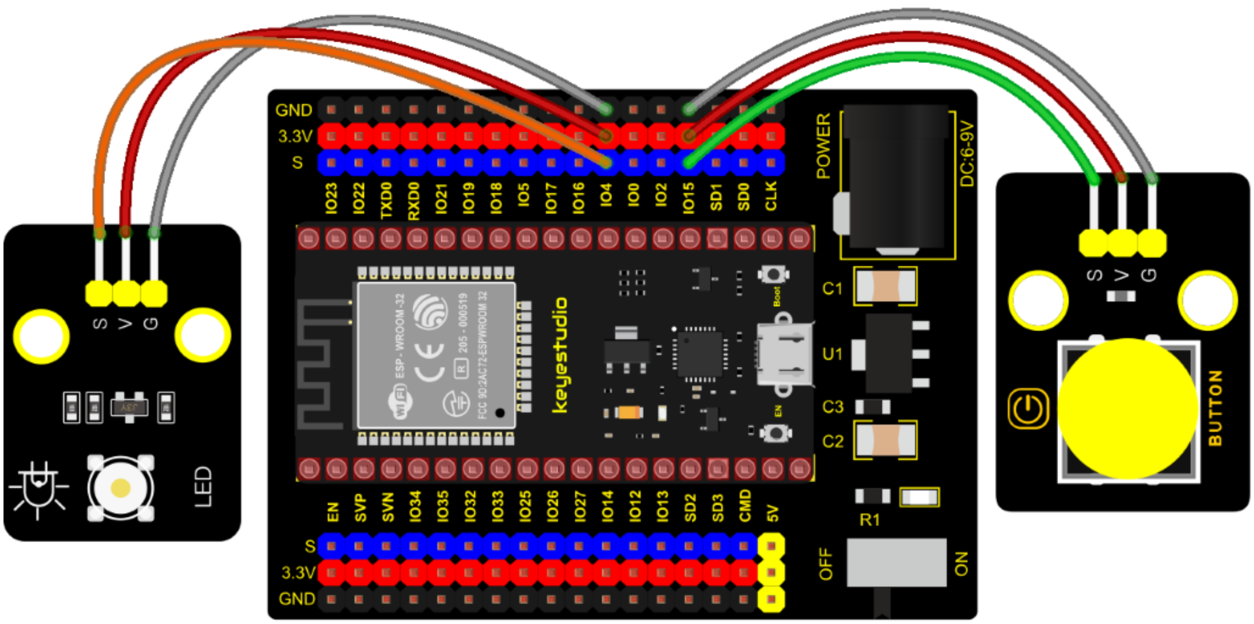
Overview

In this lesson, we will make an extension experiment with a button and an LED. When the button is pressed and low levels are output, the LED will light up; when the button is released, the LED will go off. Then we can control a module with another module.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED Module*1
		
Keyestudio DIY Button Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : button_control_LED
 * Description : Make a table lamp.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_LED    4
```

(continues on next page)

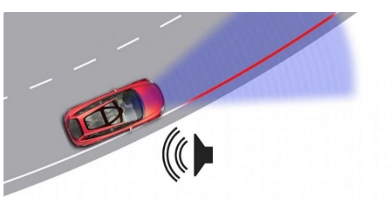
(continued from previous page)

```
#define PIN_BUTTON 15
bool ledState = false;

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);
  }
}

void reverseGPIO(int pin) {
  ledState = !ledState;
  digitalWrite(pin, ledState);
}
//*************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When the button is pressed, the LED will light up; when pressed again, the LED will go off.

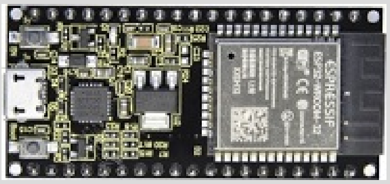
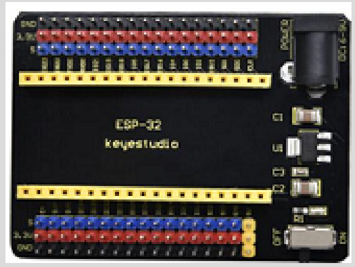
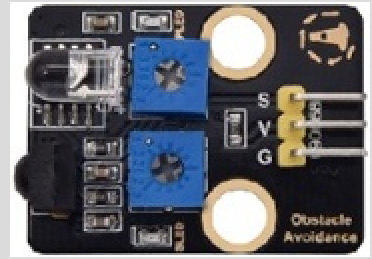
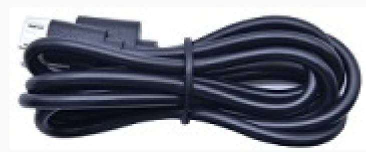
7.6.2 Project 46: Alarm Experiment



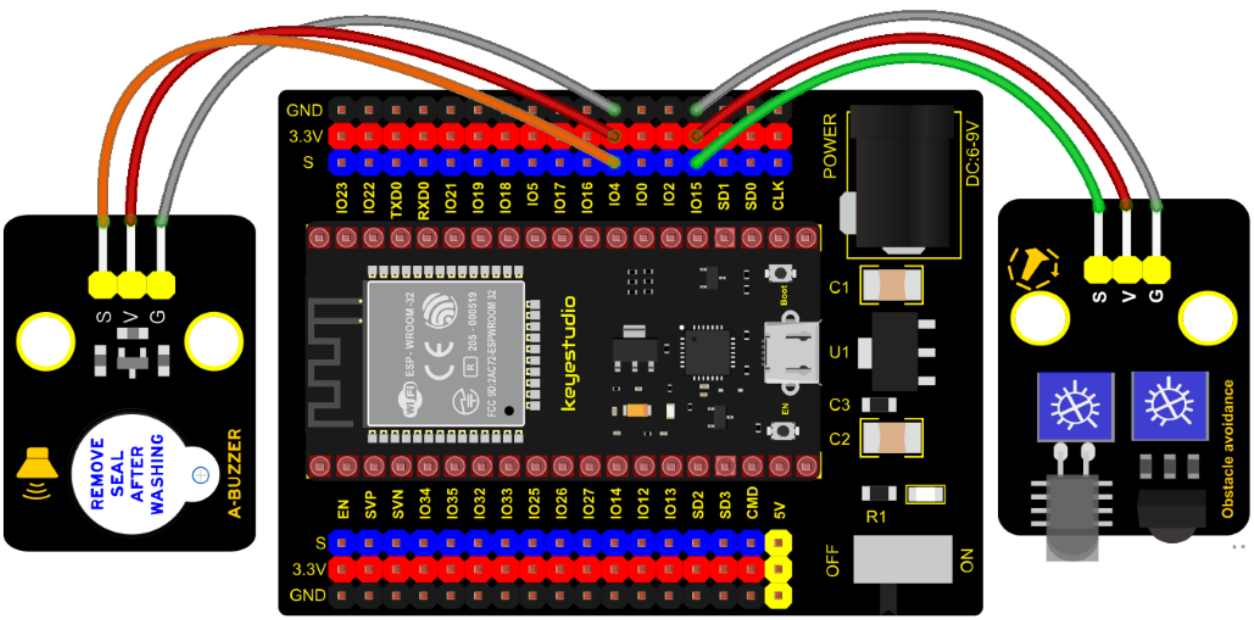
Overview

In the previous experiment, we control an output module though an input module. In this lesson, we will make an experiment that the active buzzer will emit sounds once an obstacle appears.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Obstacle Avoidance Sensor*1
		
Keyestudio Active Buzzer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Avoiding alarm
 * Description : Obstacle avoidance sensor controls the buzzer
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
```

(continues on next page)

(continued from previous page)

```
void setup() {
  pinMode(15, INPUT);  //Obstacle avoidance sensor is connected to GPIO15 and set to␣
↪input mode
  pinMode(4, OUTPUT); //The buzzer is connected to GPIO4 and set to output mode
}

void loop() {
  item = digitalRead(15);//Read the level value output by the obstacle avoidance sensor
  if (item == 0) {//Obstruction detected
    digitalWrite(4, HIGH);//The buzzer sounded
  } else {//No obstacles detected
    digitalWrite(4, LOW);//The buzzer is off
  }
  delay(100);//Delay 100ms
}
//*************************************************************************************
```

Code Explanation

Set IO ports according to connection diagram then configure pins mode

The value is 0 when pressing the button, So, we can determine the key value(0through if (item == 0) and make the buzzer beep digitalWrite(4, HIGH).

Test Result

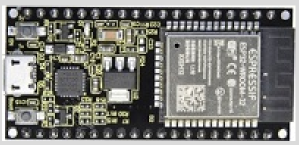
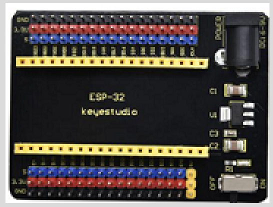
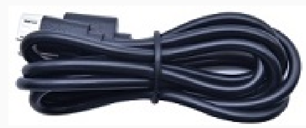
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. If the obstacle is detected, the active buzzer will chime; if not, it won't beep.

7.6.3 Project 47: Intrusion Detection

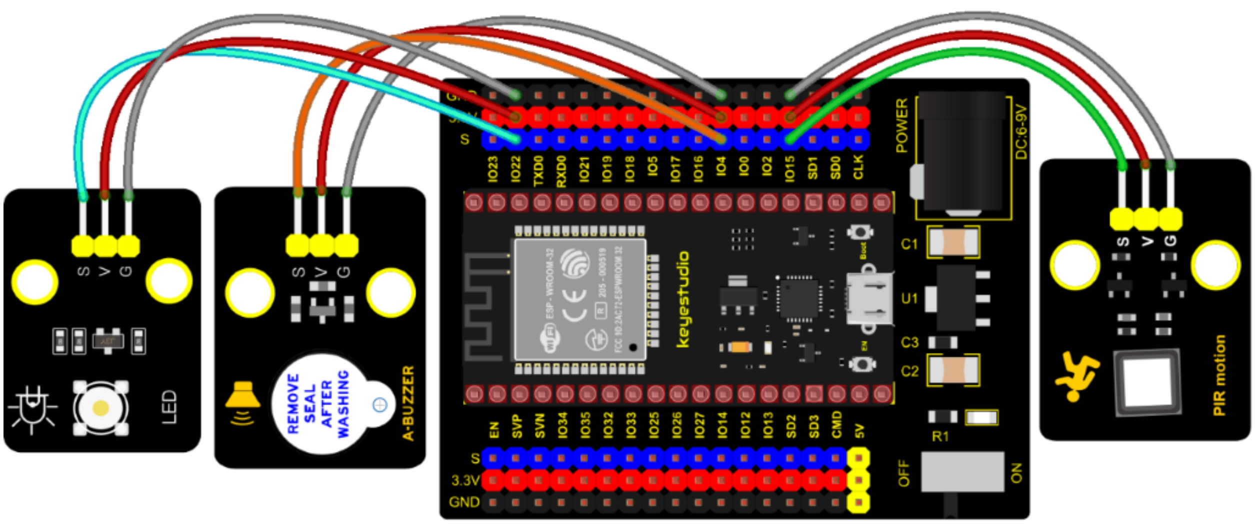
Description

In this experiment, we use a PIR motion sensor to control an active buzzer to emit sounds and the onboard LED to flash rapidly.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY PIR Motion Sensor*1	Keyestudio DIY Active Buzzer*1
			
Keyestudio Purple LED Module*1	3P Dupont Wire*3	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : PIR alarm
 * Description : PIR control buzzer
 * Auther      : http//www.keyestudio.com
*/
```

(continues on next page)

(continued from previous page)

```
int item = 0;
void setup() {
  pinMode(15, INPUT);  //PIR motion sensor is connected to GPIO15 and set as the input␣
↪mode
  pinMode(4, OUTPUT);//The active buzzer is connected to GPIO4 and set to output mode
  pinMode(22, OUTPUT);//LED is connected to GPIO22 and set to output mode
}

void loop() {
  item = digitalRead(15);//Read digital level signal output by infrared pyrorelease␣
↪sensor
  if (item == 1) {  //Movement detected
    digitalWrite(4, HIGH); //Turn on the buzzer
    digitalWrite(22, HIGH); //Turn on the LED
    delay(200);//Delay 200ms
    digitalWrite(4, LOW); //Turn off the buzzer
    digitalWrite(22, LOW); //Turn off the LED
    delay(200);//Delay 200ms
  } else {  //No detection
    digitalWrite(4, LOW); //Turn off the buzzer
    digitalWrite(22, LOW); //Turn off the LED
  }
}
//*************************************************************************************
```

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. If the sensor detects people moving, the buzzer will emit an alarm , and the LED will flash continuously.

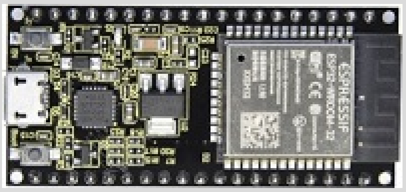
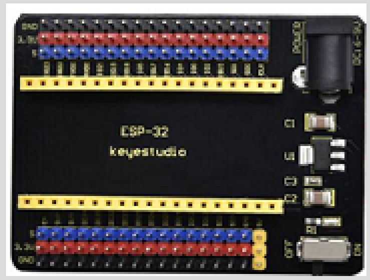
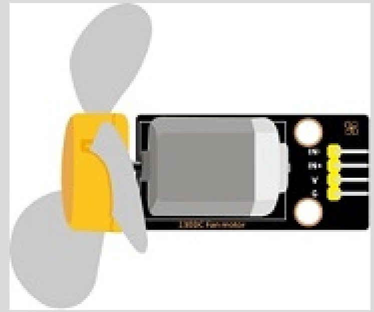
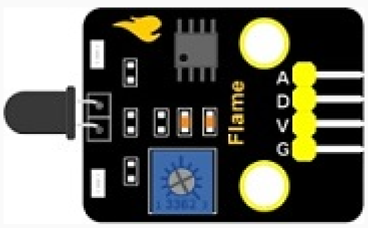
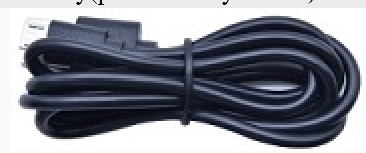
7.6.4 Project 48: Extinguishing Robot

Description

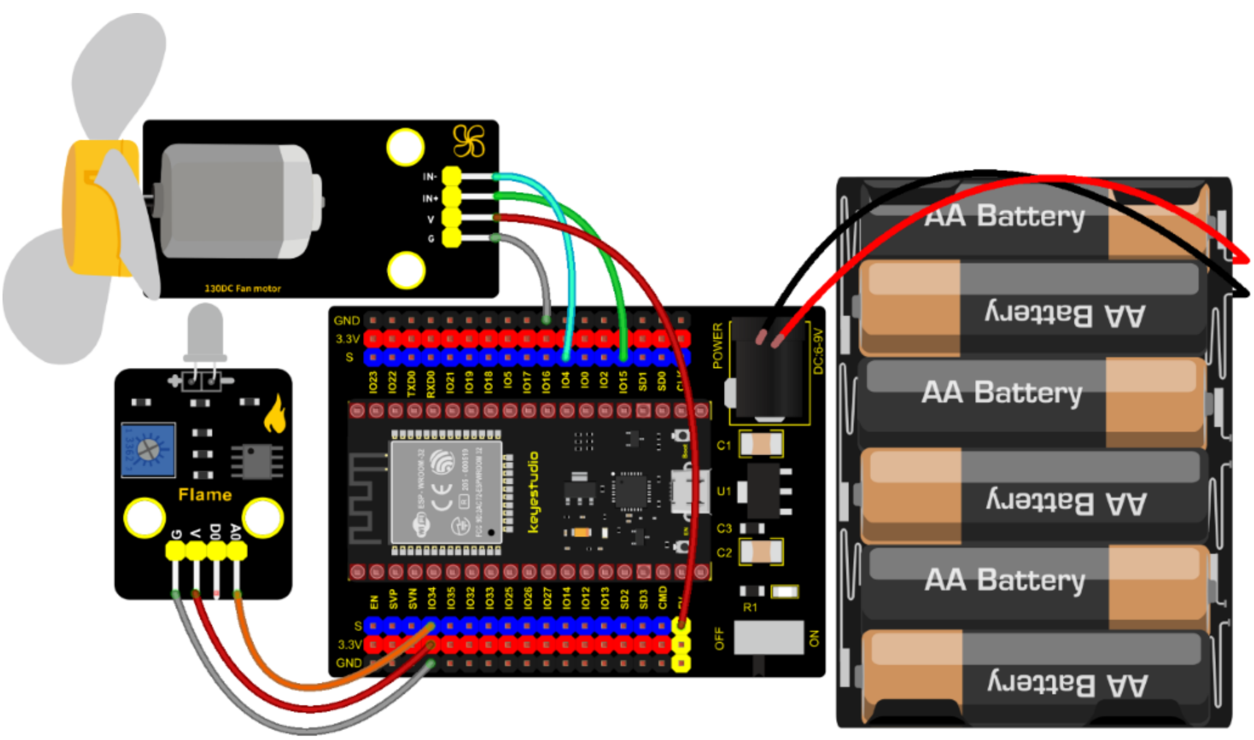
Today we will use Arduino simulation to build an extinguishing robot that will automatically sense the fire and start the fan.

In this project, we will learn how to build a very simple robot using ESP32, (detecting flames with a flame sensor, blowing out candles with a fan) can teach us basic concepts about robotics. Once you understand the basics below, you can build more complex robots.

Components Required

		
ESP32 Board*1	ESP32 Expansion Board*1	130 Motor*1
		
Flame Sensor*1	Battery Holder*1	Battery(privode for yourself)*6
		
3P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Fire-fighting robot
 * Description : Flame sensor controls the 130 fan module
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
  Serial.begin(9600);
  pinMode(15, OUTPUT);//INA corresponds to IN+, and sets GPIO15 to output mode
  pinMode(4, OUTPUT);//INB corresponds to IN-, and sets GPIO4 to output mode
}

void loop() {
  item = analogRead(34);//The flame sensor is connected to GPIO34, and read the␣
↪simulated value to Item
  Serial.println(item);//Serial port display analog value
  if (item < 3000) {//Less than 3000
    digitalWrite(15, LOW);//Turn on the fan
    digitalWrite(4, HIGH);
  } else {//Otherwise, turn off the fan.
    digitalWrite(15, LOW);
    digitalWrite(4, LOW);
  }
  delay(100);
}
//**********************************************************************************
```

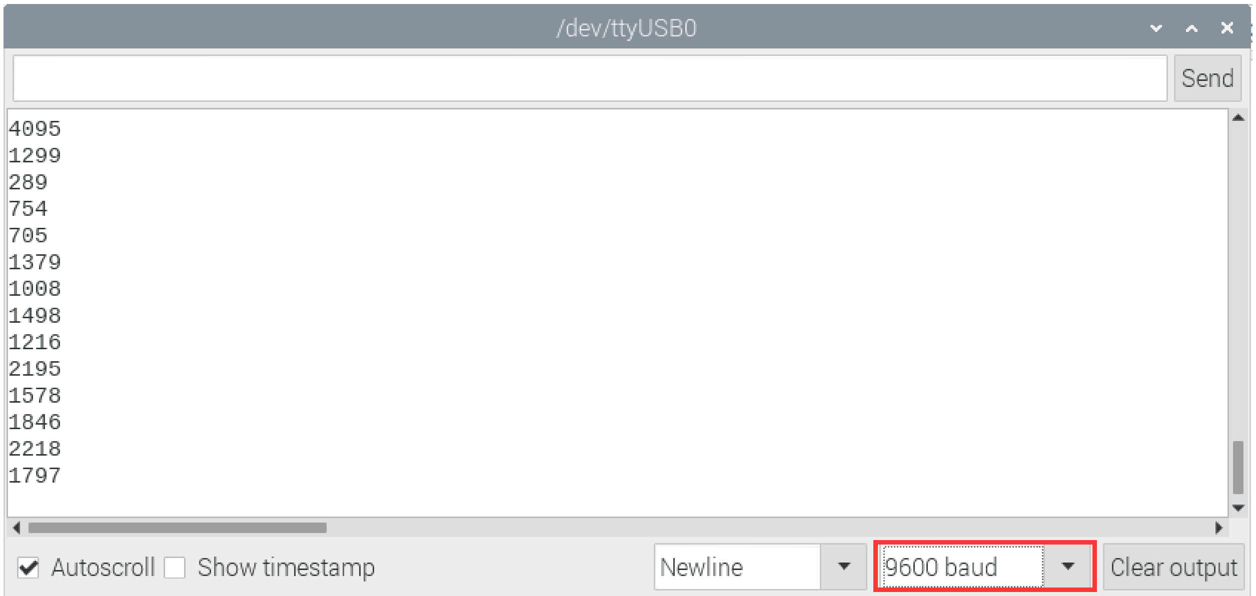
Code Explanation

In the code, we set the threshold value to 3000. When the ADC value detected by the flame sensor is lower than the threshold value, the fan will be automatically turned on; otherwise, it will be turned off. For the driving method of the fan, please refer to the 130 Motor.

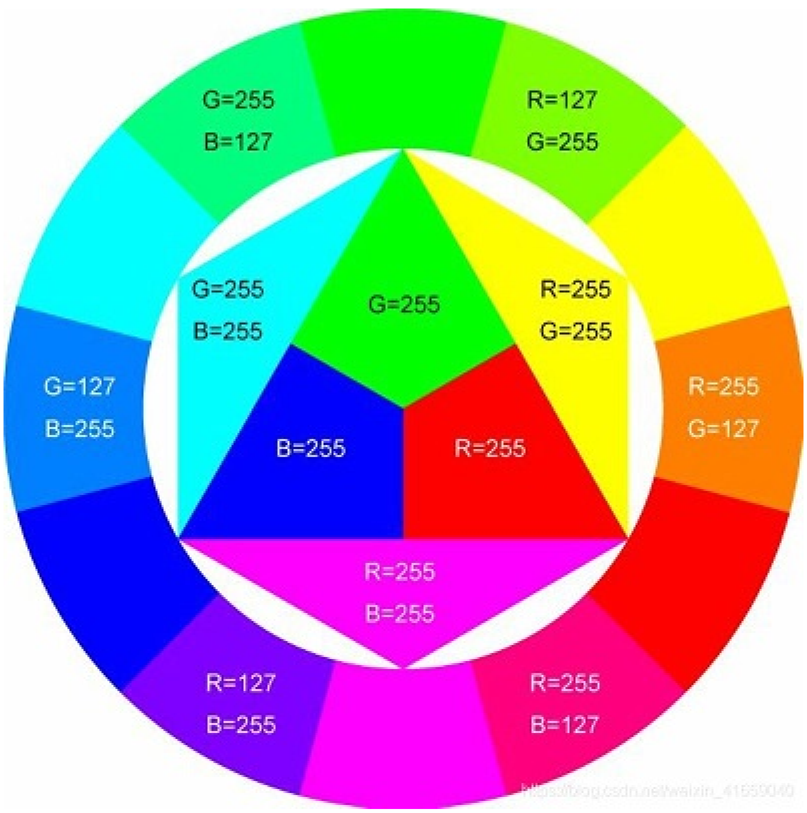
Test Result

Connect the wires according to the experimental wiring diagram, switch the DIP switch on the ESP32 expansion board to the ON end and power up, compile and upload the code to the ESP32. After uploading successfully, open the serial monitor and set baud rate to 9600.

We need to press the reset button on the ESP32, then the ADC value of the flame will be printed. When this value is less than 3000, the fan will work to blow out the fire, otherwise, it will be turned off. Basically, the ADC value can be set by yourself.



7.6.5 Project 49: Rotary Encoder control RGB

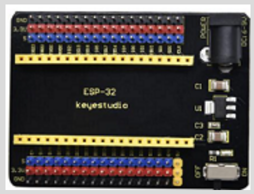
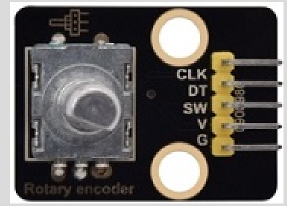
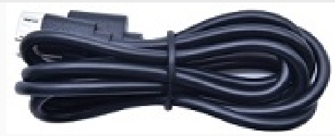


Introduction

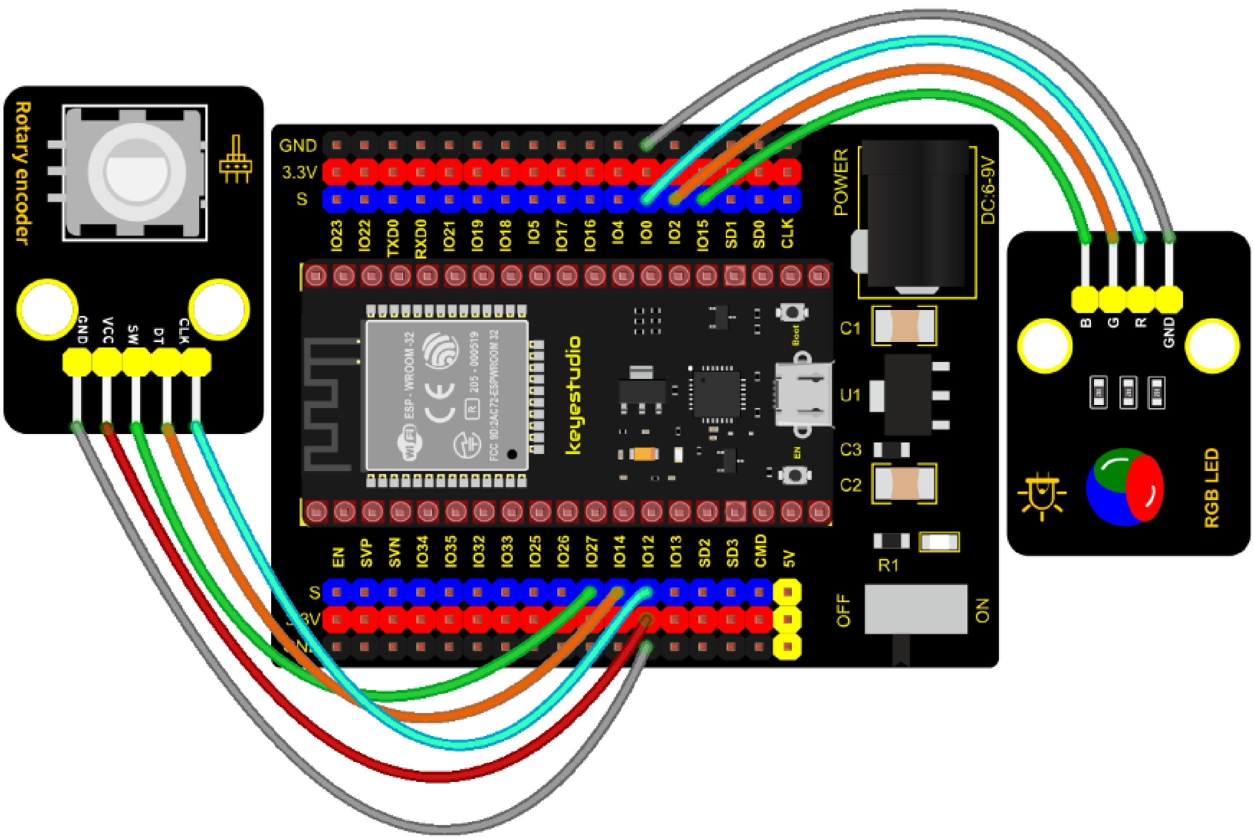
In this lesson, we will control the LED on the RGB module to show different colors through a rotary encoder.

When designing the code, we need to divide the obtained values by 3 to get the remainders. The remainder is 0 and the LED will become red. The remainder is 1, the LED will become green. The remainder is 2, the LED will turn blue.

Components

			
ESP32Board*1	ESP32 Expansion Board*1	KeyestudioCommon Cathode RGB Module*1	KeyestudioRotary Encoder Module*1
			
5P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Encoder control RGB
 * Description : Rotary encoder controls RGB to present different effects
 * Auther      : http//www.keyestudio.com
*/
//Interfacing Rotary Encoder with Arduino
//Encoder Switch -> pin 27
//Encoder DT -> pin 14
//Encoder CLK -> pin 12
int Encoder_DT  = 14;
int Encoder_CLK  = 12;
int Encoder_Switch = 27;

int Previous_Output;
int Encoder_Count;

int ledPins[] = {0, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;

int val;
void setup() {
  Serial.begin(9600);

  //pin Mode declaration
  pinMode (Encoder_DT, INPUT);
  pinMode (Encoder_CLK, INPUT);
  pinMode (Encoder_Switch, INPUT);

  Previous_Output = digitalRead(Encoder_DT); //Read the inital value of Output A
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
   }
}

void loop() {
  //aVal = digitalRead(pinA);

  if (digitalRead(Encoder_DT) != Previous_Output)
  {
    if (digitalRead(Encoder_CLK) != Previous_Output)
    {
      Encoder_Count ++;
      Serial.print(Encoder_Count);
      Serial.print("  ");
      val = Encoder_Count % 3;
      Serial.println(val);
    }
    else
    {
      Encoder_Count--;
```

(continues on next page)

(continued from previous page)

```
      Serial.print(Encoder_Count);
      Serial.print("  ");
      val = Encoder_Count % 3;
      Serial.println(val);
    }
  }

  Previous_Output = digitalRead(Encoder_DT);

  if (digitalRead(Encoder_Switch) == 0)
  {
    delay(5);
    if (digitalRead(Encoder_Switch) == 0) {
      Serial.println("Switch pressed");
      while (digitalRead(Encoder_Switch) == 0);
    }
  }
  if (val == 0) {
    //RED(255, 0, 0)
    ledcWrite(chns[0], 255 );
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 0);
  } else if (val == 1) {
    //GREEN(0, 255, 0)
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 255);
    ledcWrite(chns[2], 0);
  } else {
    //BLUE(0, 0, 255)
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 255);
  }
}
//*************************************************************************************
```

Code Explanation

1). In the experiment, we set the val to the remainder of Encoder_Count divided by 3. Encoder_Count is the value of the encoder. Then we can set pin GPIO0 (red), GPIO2 (green) and GPIO15 (blue) according to remainders.

2). Referring to the control method learned in the previous experiment, use the LED on the remainder control module to display the corresponding light color. The value obtained by taking the remainder of 3 for any number is 0 or 1 or 2. We use these three values to judge, and display the corresponding color.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then rotate the knob of the rotary encoder to display the reminders, which can control colors of LED(red green blue).

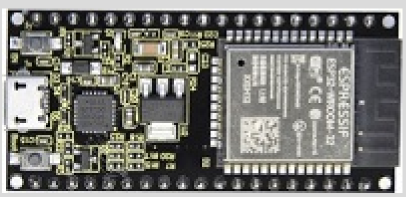
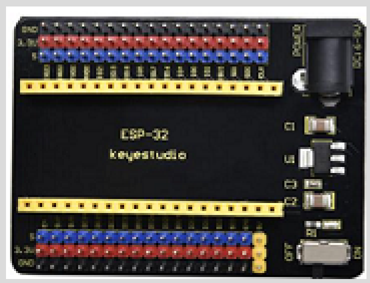
7.6.6 Project 50: Rotary Potentiometer

Introduction

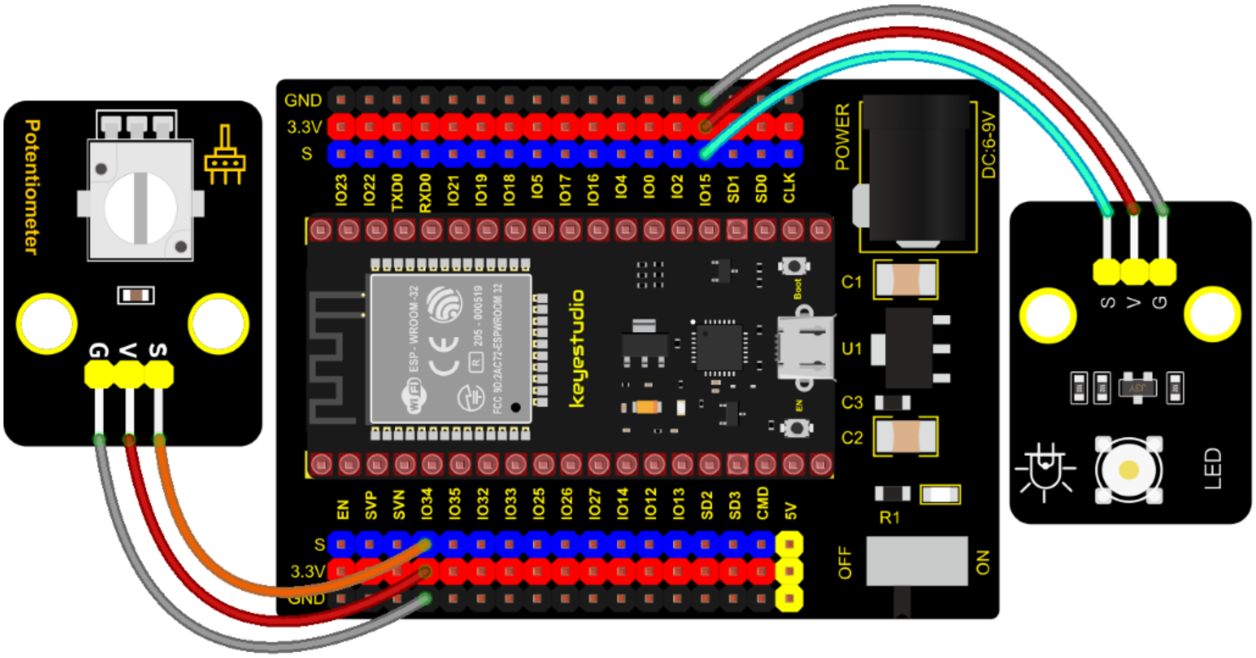
In the previous courses, we did experiments of breathing light and controlling LED with button. In this course, we do these two experiments by controlling the brightness of LED through an adjustable potentiometer. The brightness of LED is controlled by PWM values, and the range of analog values is 0 to 4095 and the PWM value range is 0-255.

After the code is set successfully, we can control the brightness of the LED on the module by rotating the potentiometer.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Purple LED*1
		
Keyestudio Rotary Potentiometer*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : adjust the light
 * Description : Controlling the brightness of LED by potentiometer.
 * Auther      : http//www.keyestudio.com
*/
#define PIN_ANALOG_IN    34  //the pin of the potentiometer
#define PIN_LED     15  // the pin of the LED
#define CHAN             0
void setup() {
  ledcSetup(CHAN, 1000, 12);
  ledcAttachPin(PIN_LED, CHAN);
}

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

Code Explanation

In the experiment, the mapping function maps adcVal from the range of 0-4095 to 0-255, and assigns it to pwmVal.

Test Result

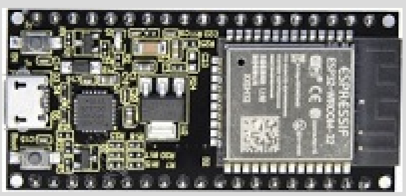
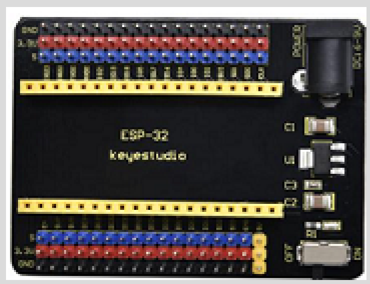
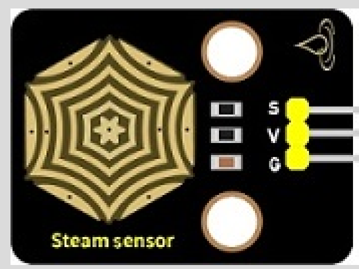
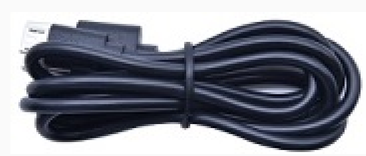
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Rotating the potentiometer on the module can adjust the brightness of the LED on the LED module.

7.6.7 Project 51: Smart Windows

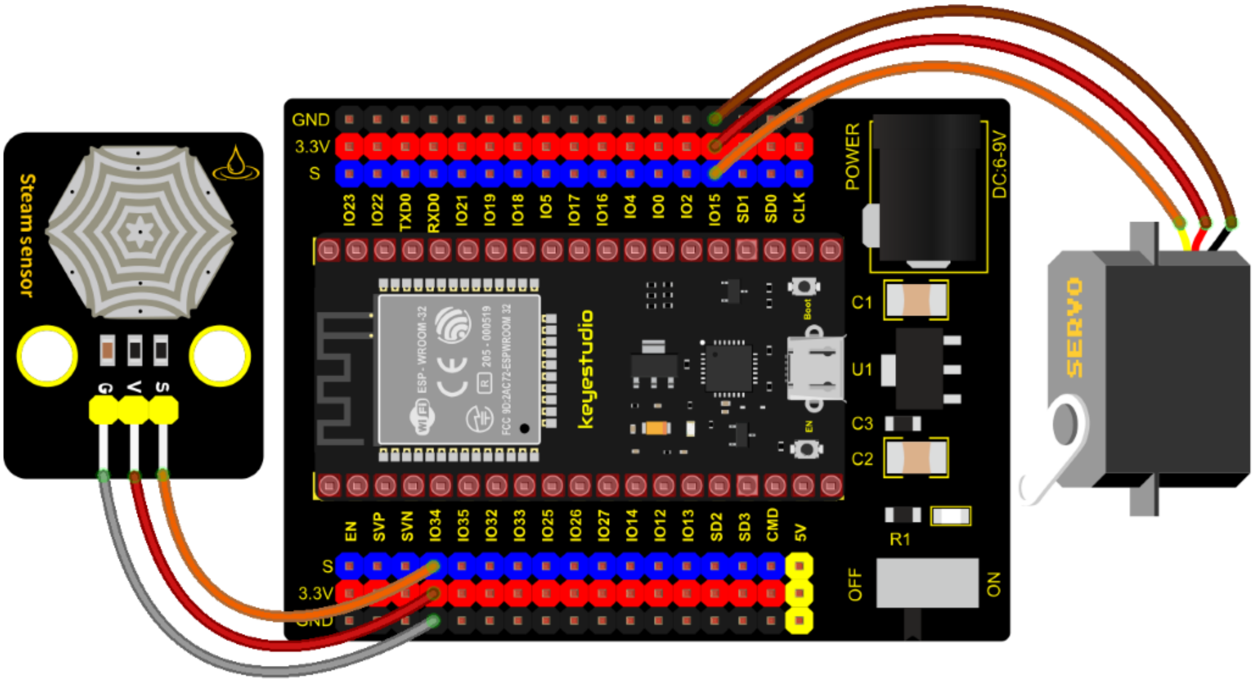
Description

In life, we can see all kinds of smart products, such as smart home. Smart homes include smart curtains, smart windows, smart TVs, smart lights, and more. In this experiment, we use a steam sensor to detect rainwater, and then achieve the effect of closing and opening the window by a servo.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Steam Sensor*1
		
Servo*1	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : smart window
 * Description : Water drop sensor controls steering gear rotation.
 * Auther      : http//www.keyestudio.com
*/
#include <ESP32Servo.h>//Import the steering gear library file
int adcVal = 0;//A variable that holds the ADC value output by the droplet sensor
int servoPin = 15;  // Define the servo pin
Servo myservo;//Defines an instance of the steering gear class

#define PIN_ADC    34  //the pin of the Water drop sensor

void setup(){
  Serial.begin(9600);
  pinMode(PIN_ADC, INPUT);
  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo object
}

void loop(){
  adcVal = analogRead(PIN_ADC);//The droplet sensor is connected to the analog port GP34
  Serial.println(adcVal);
  if (adcVal > 2000) {//The simulated value is greater than 2000
    myservo.write(0);//close the window
    delay(500);//Give the steering gear time to turn
  } else {// no rain
```

(continues on next page)

(continued from previous page)

```
    myservo.write(180);//open the window
    delay(500);//Delay 500ms
  }
}
//*************************************************************************************
```

Code Explanation

We can control a servo to rotate by a threshold.

Test Result

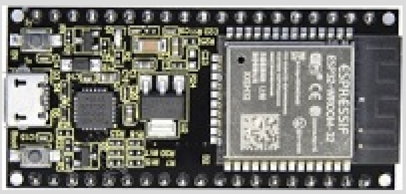
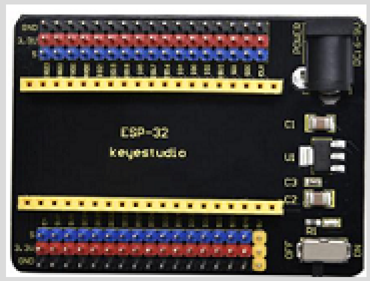
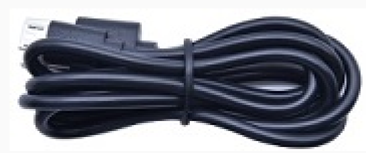
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When the sensor detects a certain amount of water, the servo rotates to achieve the effect of closing or opening windows.

7.6.8 Project 52: Sound Activated Light

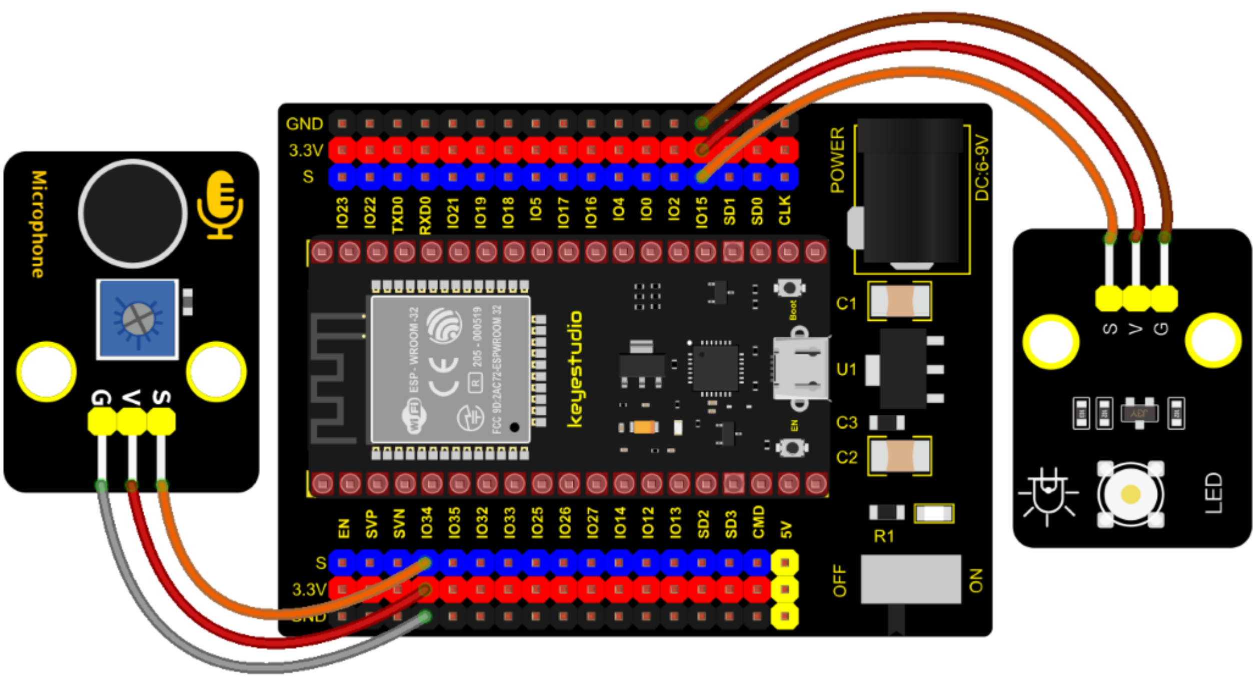
Introduction

In this lesson, we will make a smart sound activated light using a sound sensor and an LED module. When we make a sound, the light will automatically turn on; when there is no sound, the lights will automatically turn off. How it works? Because the sound-controlled light is equipped with a sound sensor, and this sensor converts the intensity of external sound into a corresponding value. Then set a threshold, when the threshold is exceeded, the light will turn on, and when it is not exceeded, the light will go out.

Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Sound Sensor*1
		
Keyestudio Purple LED Module*1	3P Dupont Wire*2	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : sound-controlled lights
 * Description : Sound sensor controls LED on and off
 * Auther      : http//www.keyestudio.com
*/
int ledPin = 15;//LED is connected to GP15
int microPin = 34;//Sound sensor is connected to GPIO34
void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(ledPin, OUTPUT);//LED is the output mode
}

void loop() {
  int val = analogRead(microPin);//Read analog value
  Serial.print(val);// Serial port print
  if(val > 600){//exceed the threshold value
    digitalWrite(ledPin, HIGH);//Lighting LED 3sand print the corresponding information
    Serial.println("  led on");
    delay(3000);
  }else{//otherwise
    digitalWrite(ledPin, LOW);//Turn off the LED and print the corresponding information
    Serial.println("  led off");
  }
  delay(100);
}
//**********************************************************************************
```

Code Explanation

We set the ADC threshold value to 600. If more than 600, LED will be on 3s; on the contrary, it will be off.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600.

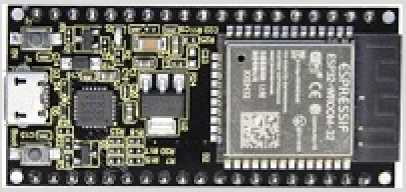
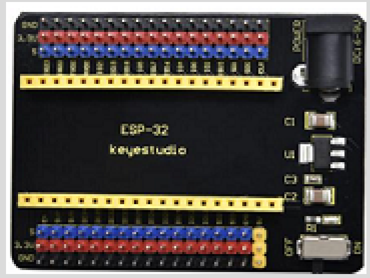
We need to press the reset button on the ESP32, then the corresponding volume ADC value will be displayed. When the analog value of sound is greater than 600, the LED on the LED module will light up 3s, otherwise it will go off.

7.6.9 Project 53: Fire Alarm

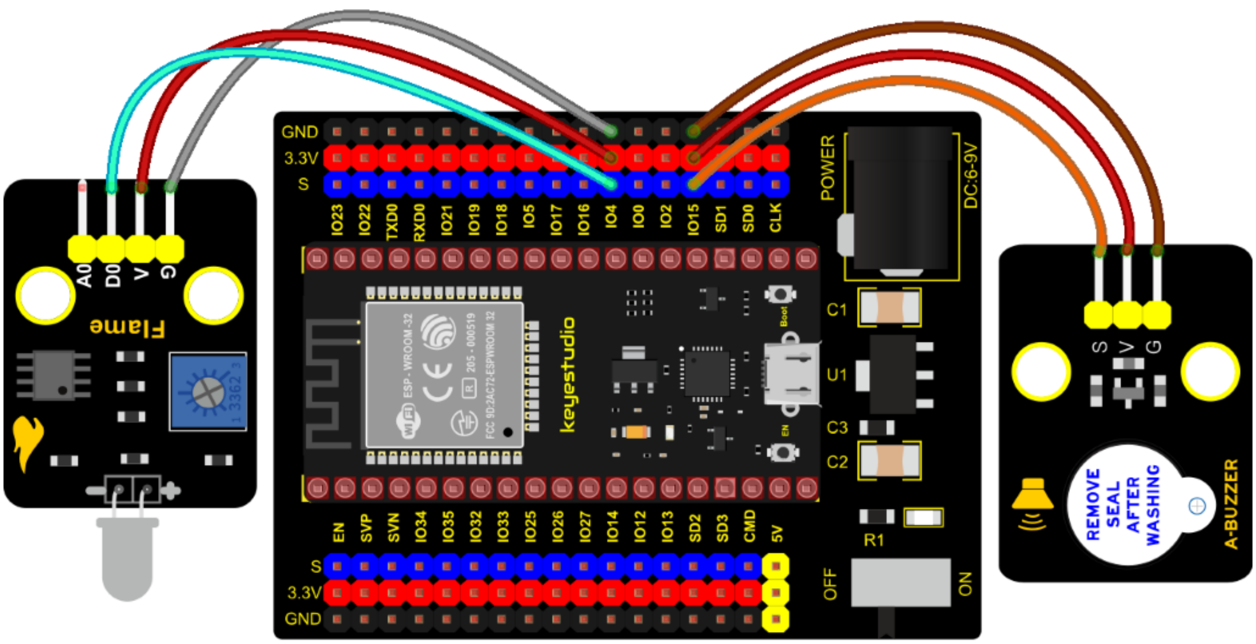
Description

In this experiment, we will make a fire alarm system. Just use a flame sensor to control an active buzzer to emit sounds.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Active Buzzer*1
		
Micro USB Cable*1	3P Dupont Wire*2	keyestudio DIY Flame Sensor*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Flame Alarm
 * Description : Controlling the buzzer by flame sensor.
 * Auther      : http//www.keyestudio.com
*/
int item = 0;
void setup() {
```

(continues on next page)

(continued from previous page)

```
  Serial.begin(9600);
  pinMode(4, INPUT);//Flame sensor digital pin is connected to GPIO4
  pinMode(15, OUTPUT);//Buzzer pin is connected to GPIO15
}

void loop() {
  item = digitalRead(4);//Read the digital level output by the flame sensor
  Serial.println(item);//Newline print level signal
  if (item == 0) {//Flame detected
    digitalWrite(15, HIGH);//Turn on the buzzer
  } else {//Otherwise, turn off the buzzer
    digitalWrite(15, LOW);
  }
  delay(100);//Delay 100ms
}
//**********************************************************************************
```

Code Explanation

This flame sensor uses an analog pin and a digital pin. When a flame is detected, the digital pin outputs a low level. In this experiment we will use the digital port.

Test Result

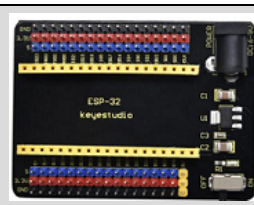
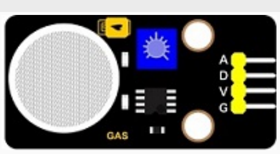
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When the sensor detects the flame, the external active buzzer will emit sounds, otherwise the active buzzer will not emit sounds.

7.6.10 Project 54: Smoke Alarm

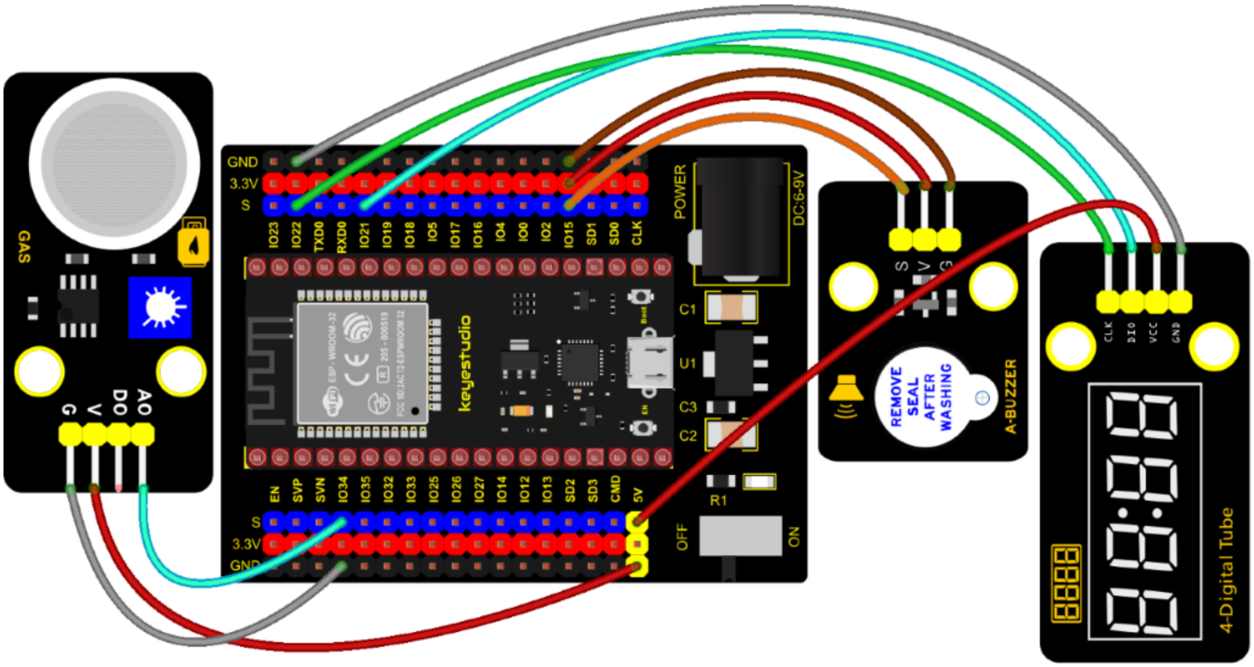
Description

In this experiment, we will make a smoke alarm by a TM16504-Digit segment module, a gas sensor and an active buzzer.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio Active Buzzer*1	Keyestudio TM16504-Digit Segment Module*1
			
keyestudio Analog Gas Senso*1	3P Dupont Wire*2	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//*************************************************************************************
/*
 * Filename    : smoke alarm
 * Description : MQ2 controls a buzzer and a four-digit analog smoke tester
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
int adcVal = 0; //display ADC value
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(15, OUTPUT);//the buzzer is connected to GPIO15
}

void loop() {
```

(continues on next page)

(continued from previous page)

```
  adcVal = analogRead(34);//Read the ADC values of MQ2
  displayFloatNum(adcVal);;//Four digit tube display adcVal values
  if (adcVal > 1000) {//ADC value is greater than 1000
    digitalWrite(15, HIGH); // buzzer alarming
  } else {//or else
    digitalWrite(15, LOW); //Turn off the buzzer
  }
  delay(100);//delay 100ms
}

void displayFloatNum(float adcVal){
  if(adcVal > 9999)
    return;
  int dat = adcVal*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Code Explanation

Define an integer variable val to store the analog value of thesmoke sensor, and then we display the analog value in the four-digit digital tube, and then set a threshold, and when the threshold is reached, the buzzer will sound.

Test Result

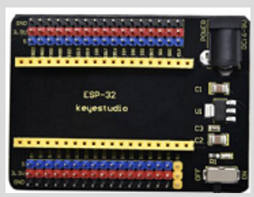
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When the concentration of combustible gas exceeds the standard, the active buzzer module will give an alarm, and the four-digit digital tube will display the concentration value.

7.6.11 Project 55: Alcohol Sensor

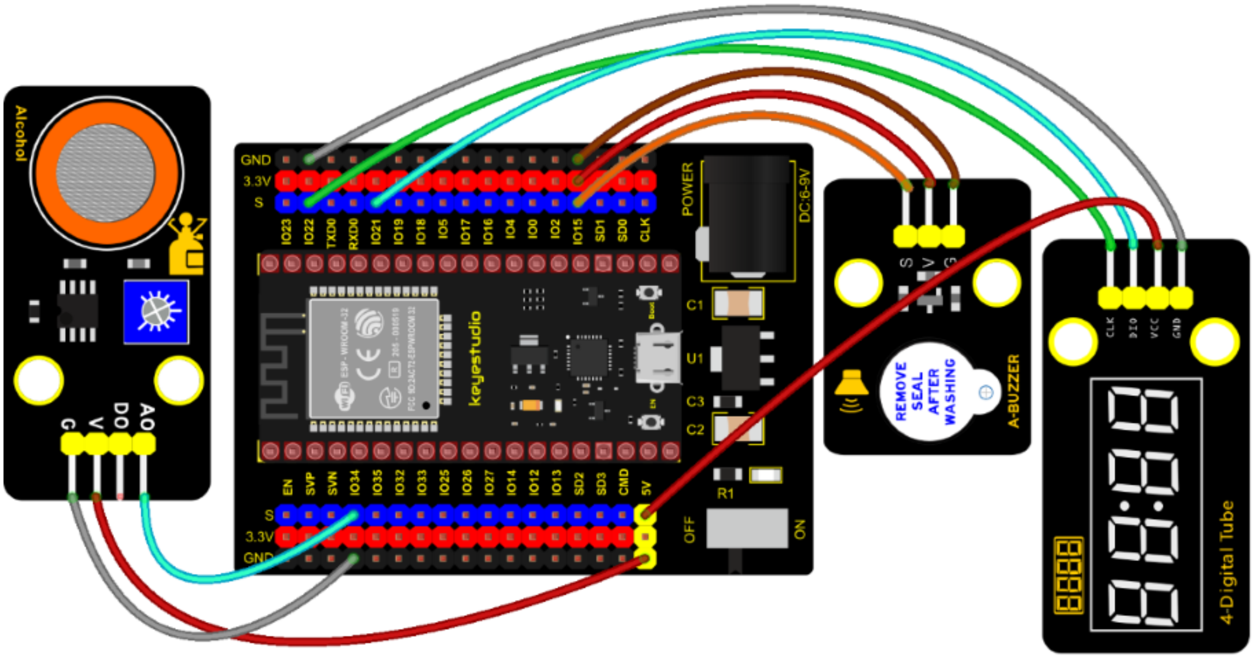
Description

In the last experiment, we made a smoke alarm. In this experiment, we combine the active buzzer, the MQ-3 alcohol sensor, and a four-digit digital tube to test the alcohol concentration through the alcohol sensor. Then, the concentration to control the active buzzer alarm and the four-digit digital tube to display the concentration. So as to achieve the simulation effect of alcohol detector.

Components Required

			
ESP32 Board*1	ESP32 Expansion Board*1	Active Buzzer*1	Keyestudio DIY TM1650 4-Digit Tube Display*1
			
keyestudio Alcohol Sensor*1	3P Dupont Wire*2	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : breathalyzer
 * Description : MQ3 controls a buzzer and a four-digit tube to simulate a breathalyzer.
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
int adcVal = 0; //display ADC value
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(15, OUTPUT);//the buzzer is connected to GPIO15
}

void loop() {
  adcVal = analogRead(34);//Read the ADC values of MQ3
  displayFloatNum(adcVal);//Four digit tube display adcVal values
```

(continues on next page)

(continued from previous page)

```
  if (adcVal > 1000) {//ADC value is greater than 1000
    digitalWrite(15, HIGH); // buzzer alarming
  } else {//or else
    digitalWrite(15, LOW); //Turn off the buzzer
  }
  delay(100);//delay 100ms
}

void displayFloatNum(float adcVal){
  if(adcVal > 9999)
    return;
  int dat = adcVal*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*********************************************************************************
```

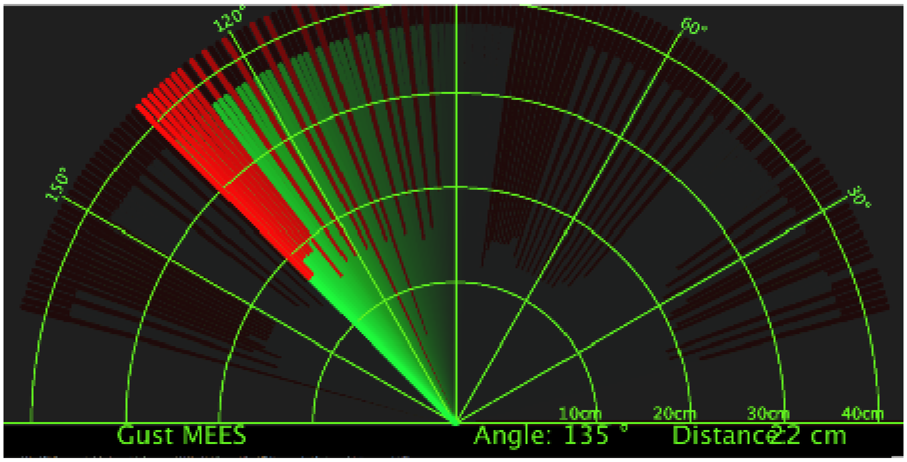
Code Explanation

Define an integer variable val to store the ADC value of the alcohol sensor, then we display the analog value in the four-digit display module and set a threshold.

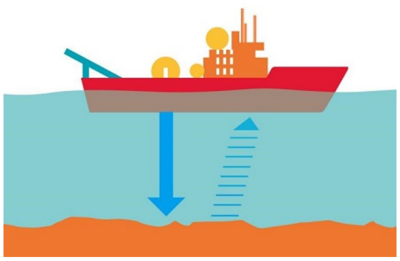
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When different alcohol concentrations are detected, the active buzzer module will alarm, and the four-digit digital display will show the concentration value.

7.6.12 Project 56: Ultrasonic Radar



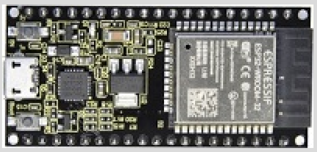
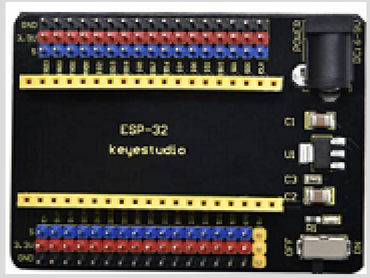
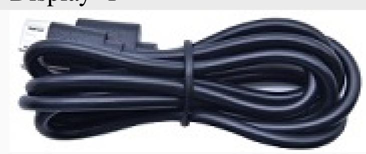
Description



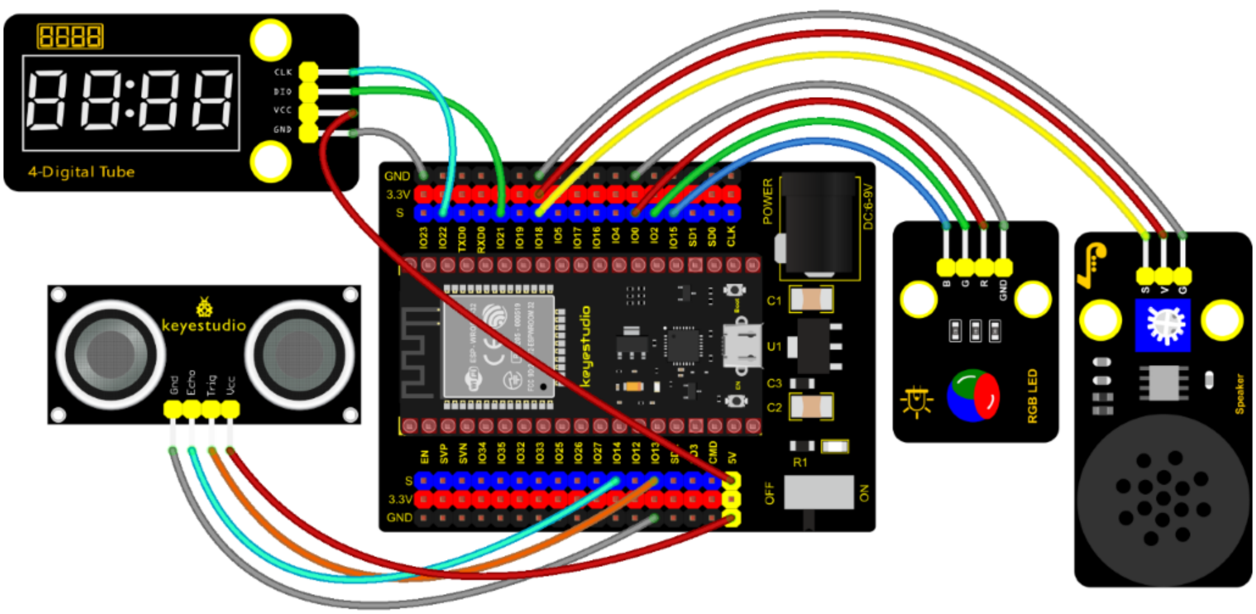
We know that bats use echoes to determine the direction and the location of their preys. In real life, sonar is used to detect sounds in the water. Since the attenuation rate of electromagnetic waves in water is very high, it cannot be used to detect signals, however, the attenuation rate of sound waves in the water is much smaller, so sound waves are most commonly used underwater for observation and measurement.

In this experiment, we will use a speaker module, an RGB module and a 4-digit tube display to make a device for detection through ultrasonic.

Required Components

		
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio HC-SR04 Ultrasonic Sensor*1
		
Keyestudio 8002b Power Amplifier*1	Keyestudio DIY Common Cathode RGB Module *1	Keyestudio DIY TM1650 4-Digit Tube Display*1
		
4P Dupont Wire*3	3P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Ultrasonic radar
 * Description : Ultrasonic control four digit tube, buzzer and RGB analog ultrasonic␣
↪radar.
 * Auther      : http//www.keyestudio.com
*/
#include "TM1650.h" //Import the TM1650 library file
//the interfaces are GPIO21 and GPIO22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);

int beeppin = 18; //Define the horn pin as GPIO18

int TrigPin = 13; //Set the Trig pin to GPIO13
int EchoPin = 14; //Set the Echo pin to GPIO14
int distance;//Distance measured by ultrasound

int ledPins[] = {0, 2, 15};    //define red, green, blue led pins
const byte chns[] = {0, 1, 2};        //define the pwm channels

float checkdistance() { //get distance
  // A short low level is given beforehand to ensure a clean high pulse:
  digitalWrite(TrigPin, LOW);
  delayMicroseconds(2);
  // The sensor is triggered by a high pulse of 10 microseconds or more
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);
  // Read the signal from the sensor: a high level pulse
  //Its duration is the time (in microseconds) from sending the ping command to␣
↪receiving the echo from the object
  float distance = pulseIn(EchoPin, HIGH) / 58.00;  //Convert to distance
  delay(10);
  return distance;
}

void setup() {
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  pinMode(TrigPin, OUTPUT);//Sets the Trig pin as output
  pinMode(EchoPin, INPUT);  //Set the Echo pin as input
  ledcSetup(3, 1000, 8);//setup the pwm channels,1KHz,8bit
  ledcAttachPin(18, 3);
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
```

(continues on next page)

(continued from previous page)

```
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
  }
}

void loop() {
  distance = checkdistance(); //Ultrasonic ranging
  displayFloatNum(distance);  //Nixie tube shows distance
  if (distance <= 10) {
    ledcWrite(3, 100);
    delay(100);
    ledcWrite(3, 0);
    ledcWrite(chns[0], 255); //Common cathode LED, high level to turn on the led.
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 0);

  } else if (distance > 10 && distance <= 20) {
    ledcWrite(3, 200);
    delay(200);
    ledcWrite(3, 150);
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 255);
    ledcWrite(chns[2], 0);
  } else {
    ledcWrite(3, 0);
    ledcWrite(chns[0], 0);
    ledcWrite(chns[1], 0);
    ledcWrite(chns[2], 255);
  }
}

void displayFloatNum(float distance){
  if(distance > 9999)
    return;
  int dat = distance*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
```

(continues on next page)

(continued from previous page)

```
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Code Explanation

We set sound frequency and light color by adjusting different distance range.

We can adjust the distance range in the code.

Test Result

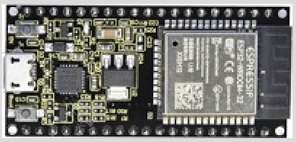
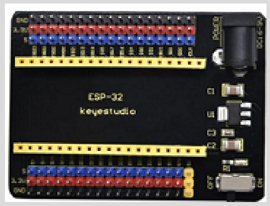
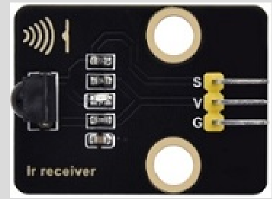
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. When the ultrasonic sensor detects different distances, the buzzer will produce different frequencies of sound (within 20 cm) , the RGB will show different colors, and the measured distances are displayed on the 4-digit tube display.

7.6.13 Project 57: IR Remote Control

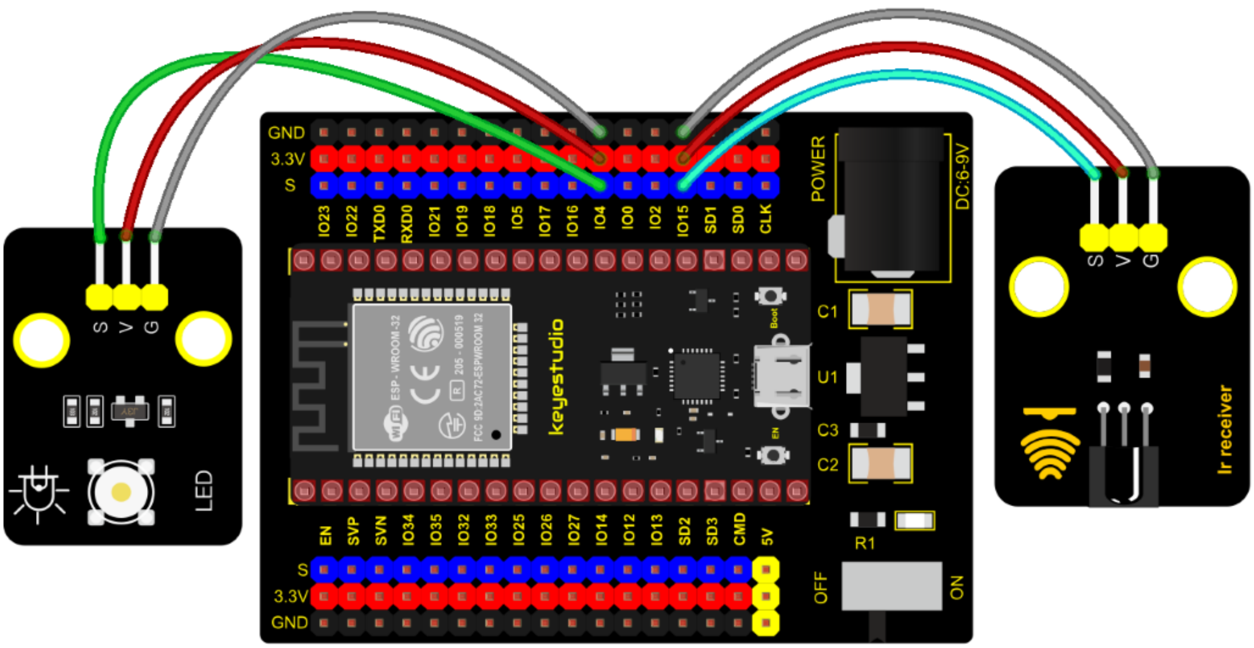
Introduction

In the previous experiments, we learned how to turn on/off the LED and adjust its brightness via PWM and print the button value of the IR remote control in the serial monitor window. Herein, we use an infrared remote control to turn on/off an LED.

Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio DIY Purple LED Module*1	Keyestudio DIY IR Receiver*1
			
Micro USB Cable*1	IR Remote Control*1	3P Dupont Wire*2	

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : IR Control LED
 * Description : Remote controls LED on and off
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
```

(continues on next page)

(continued from previous page)

```
#include <IRutils.h>

const uint16_t recvPin = 15; // Infrared receiving pin 15
IRrecv irrecv(recvPin);      // Create a class object used to receive class
decode_results results;       // Create a decoding results class object
int led = 4;//LED connect to GP4

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();                  // Start the receiver
  pinMode(led, OUTPUT);
}
////////////////////
void loop() {
  if(irrecv.decode(&results)) {        // Waiting for decoding
    serialPrintUint64(results.value, HEX);// Print out the decoded results
    Serial.print("");
    handleControl(results.value);       // Handle the commands from remote control
    irrecv.resume();                    // Receive the next value
  }
}
  void handleControl(unsigned long value) {
    if (value == 0xFF6897) // Receive the number '1'
     {
       digitalWrite(led, HIGH);//turn on LED
       Serial.println("  led on");
     }
    else if (value == 0xFF9867) // Receive the number '2'
     {
        digitalWrite(led, LOW);//turn off LED
       Serial.println("  led off");
    }
}
//**********************************************************************************
```

Test Result

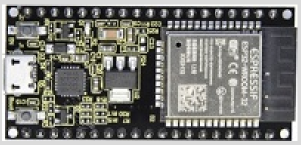
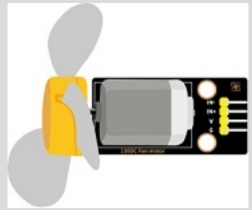
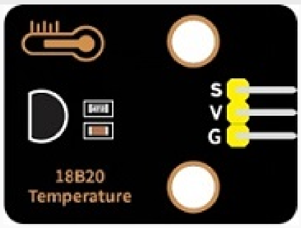
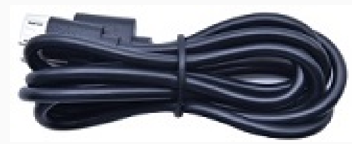
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600. We need to press the reset button on the ESP32, then press the button 1 of the remote, which will be displayed on the monitor, and the LED will be on. Similarly, press the button 2 , the LED will be off.


/dev/ttyUSB0
Send
FF6897   led on
FF9867   led off
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

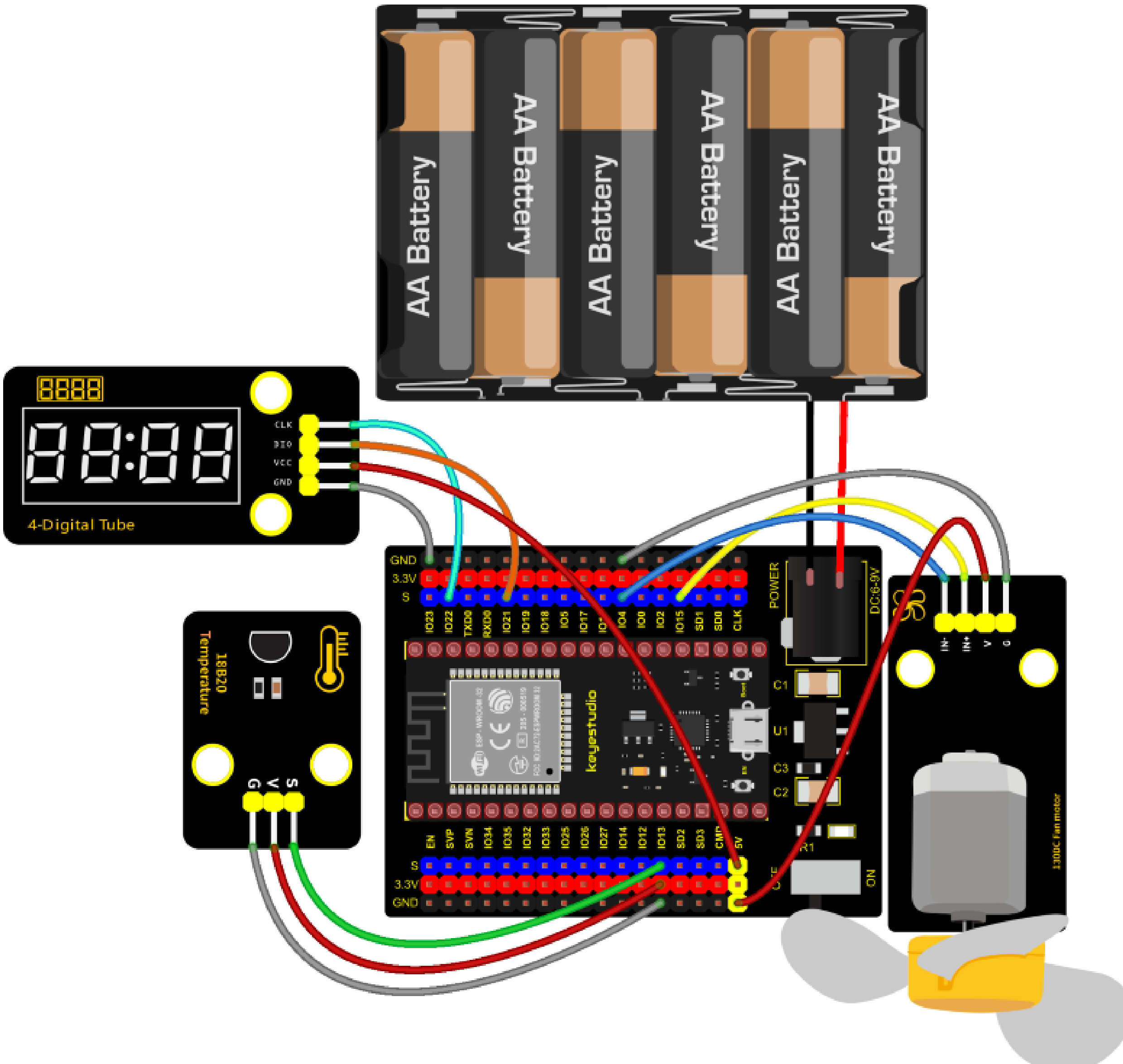
Description

We will use a temperature sensor and some modules to make a smart cooling device in this experiment. When the ambient temperature is higher than a certain value, the motor is turned on, thereby reducing the ambient temperature and achieving the heat dissipation effect. Then display the temperature value in the four-digit segment display.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	keyestudio 130 Motor*1	Keyestudio TM1650 4-Digit Segment Display*1
			
Keyestudio 18B20 Temperature Sensor*1	3P Dupont Wire*1	4P Dupont Wire*2	Micro USB Cable*1
			
Battery Holder*1	Battery (provide for yourself)*6		

Connection Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : heat abstractor
 * Description : DS18B20 controls a four digit tube and a motor that simulates Heat Abstractor
 * Auther      : http//www.keyestudio.com
*/
#include <DS18B20.h>
#include "TM1650.h" //Import the TM1650 library file
//The two ports are GP21 and GP22
#define DIO 21
#define CLK 22
TM1650 DigitalTube(CLK,DIO);
```

(continues on next page)

(continued from previous page)

```
//ds18b20 pin to 13
DS18B20 ds18b20(13);
void setup() {
  Serial.begin(9600);
  DigitalTube.setBrightness();  //set brightness, 0---7, default : 2
  DigitalTube.displayOnOFF();   //display on or off, 0=display off, 1=display on,␣
↪default : 1
  for(char b=1;b<5;b++){
    DigitalTube.clearBit(b);      //DigitalTube.clearBit(0 to 3); Clear bit display.
  }
  // DigitalTube.displayDot(1,true); //Bit0 display dot. Use before displayBit().
  DigitalTube.displayBit(1,0);    //DigitalTube.Display(bit,number); bit=0---3  number=0-
↪--9
  //Motor is connected to 15 4
  pinMode(15, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  double temp = ds18b20.GetTemp();//Read the temperature
  temp *= 0.0625;//The conversion accuracy is 0.0625/LSB
  Serial.println(temp);
  displayFloatNum(temp);//4- digit tube display temperature value
  if (temp > 25) {//When the temperature exceeds 25 degrees Celsius, turn on the fan
    digitalWrite(15, LOW);
    digitalWrite(4, HIGH);
  } else {//Otherwise, turn off the fan.
    digitalWrite(15, LOW);
    digitalWrite(4, LOW);
  }
  delay(100);
}

void displayFloatNum(float temp){
  if(temp > 9999)
    return;
  int dat = temp*10;
   //DigitalTube.displayDot(2,true); //Bit0 display dot. Use before displayBit().
  if(dat/10000 != 0){
    DigitalTube.displayBit(1, dat%100000/10000);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
  if(dat%10000/1000 != 0){
    DigitalTube.clearBit(1);
    DigitalTube.displayBit(2, dat%10000/1000);
    DigitalTube.displayBit(3, dat%1000/100);
    DigitalTube.displayBit(4, dat%100/10);
    return;
  }
```

(continues on next page)

(continued from previous page)

```
  if(dat%1000/100 != 0){
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.displayBit(3, dat%1000/100);
  DigitalTube.displayBit(4, dat%100/10);
  return;
}
  DigitalTube.clearBit(1);
  DigitalTube.clearBit(2);
  DigitalTube.clearBit(3);
  DigitalTube.displayBit(4, dat%100/10);
}
//*************************************************************************************
```

Code Explanation

The setting of variables and the storage of detection values are the same as what we learned earlier. We also set a temperature threshold and control the rotation of the motor when the threshold is exceeded, and then we use the digital tube to display the temperature value.

Test Result

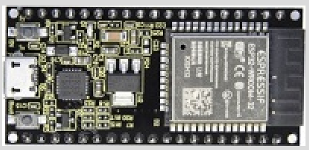
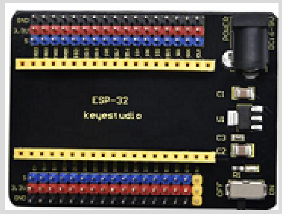
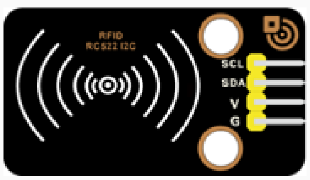
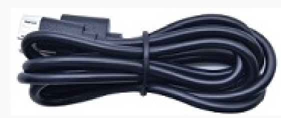
Connect the wires according to the experimental wiring diagram and power on. Switch the DIP switch on the ESP32 expansion board to the ON end, compile and upload the code to the ESP32. After uploading successfully, we can see the temperature of the current environment (unit is Celsius) on the four-digit segment display, as shown in the figure below. If this value exceeds the value we set, the fan will rotate to dissipate heat.

7.6.15 Project 59: Intelligent Entrance Guard System

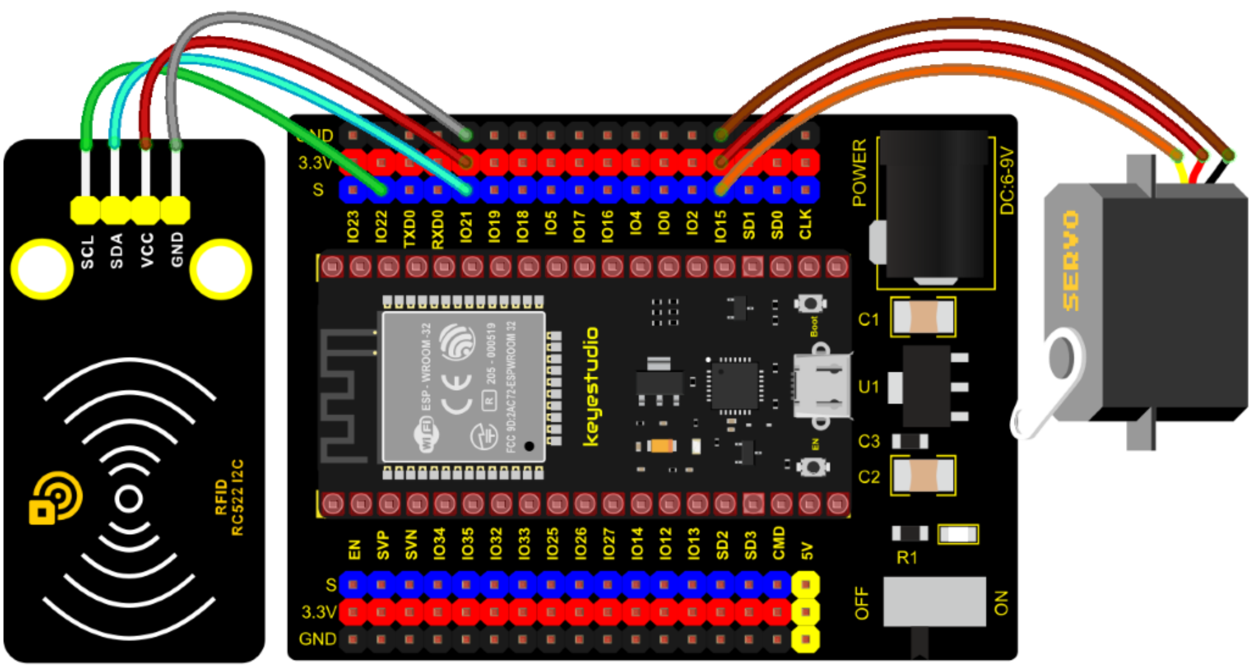
Description

In this project, we use the RFID522 card swiping module and the servo to set up an intelligent access control system. The principle is very simple. We use RFID522 swipe card module, an IC card or key card to unlock.

Required Components

			
ESP32 Board*1	ESP32 Expansion Board*1	Key*1	IC Card*1
			
Keyestudio RFID Module*1	Servo*1	4P Dupont Wire*1	Micro USB Cable*1

Connection Diagram



Test Code

Note: Different RFID-MFRC522 IC cards and keys have diverse values.You can substitute your own IC cards and keys values for the corresponding values read by the RFID-MFRC522 module in the program, otherwise the servo can't be controlled when uploading the test code to the ESP32.

For example: You can replace the rfid_str of the `if (rfid_str == "edf7945a" || rfid_str == "4c96b6e")` in the program code with your own IC cards and keys values read by the RFID-MFRC522 module.

```
//*************************************************************************************
/*
 * Filename    : Intelligent_access_control
 * Description : RFID controlled steering gear simulated door opening
 * Auther      : http//www.keyestudio.com
*/
#include <Wire.h>
#include "MFRC522_I2C.h"
// IIC pins default to GPIO21 and GPIO22 of ESP32
// 0x28 is the i2c address of SDA, if doesn't matchplease check your address with i2c.
MFRC522 mfrc522(0x28);   // create MFRC522.

#include <ESP32Servo.h>
Servo myservo;  // create servo object to control a servo
int servoPin = 15; // Servo motor pin

String rfid_str = "";

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mfrc522.PCD_Init();
  ShowReaderDetails();            // dispaly PCD - MFRC522 read carder
  Serial.println(F("Scan PICC to see UID, type, and data blocks..."));

  myservo.setPeriodHertz(50);           // standard 50 hz servo
  myservo.attach(servoPin, 500, 2500);  // attaches the servo on servoPin to the servo␣
↪object
  myservo.write(0);
  delay(500);
}

void loop() {
   if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }

  // select one of door cards. UID and SAK are mfrc522.uid.

  // save UID
  rfid_str = ""; //String emptying
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    rfid_str = rfid_str + String(mfrc522.uid.uidByte[i], HEX);  //Convert to string
    //Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    //Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println(rfid_str);

  if (rfid_str == "edf7945a" || rfid_str == "4c96b6e") {
```

(continues on next page)

(continued from previous page)

```
    myservo.write(180);
    delay(500);
    Serial.println("  open the door!");
    }
}

void ShowReaderDetails() {
  //  attain the MFRC522 software
  byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
  Serial.print(F("MFRC522 Software Version: 0x"));
  Serial.print(v, HEX);
  if (v == 0x91)
    Serial.print(F(" = v1.0"));
  else if (v == 0x92)
    Serial.print(F(" = v2.0"));
  else
    Serial.print(F(" (unknown)"));
  Serial.println("");
  // when returning to 0x00 or 0xFF, may fail to transmit communication signals
  if ((v == 0x00) || (v == 0xFF)) {
    Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?
↪"));
  }
}
//*************************************************************************************
```

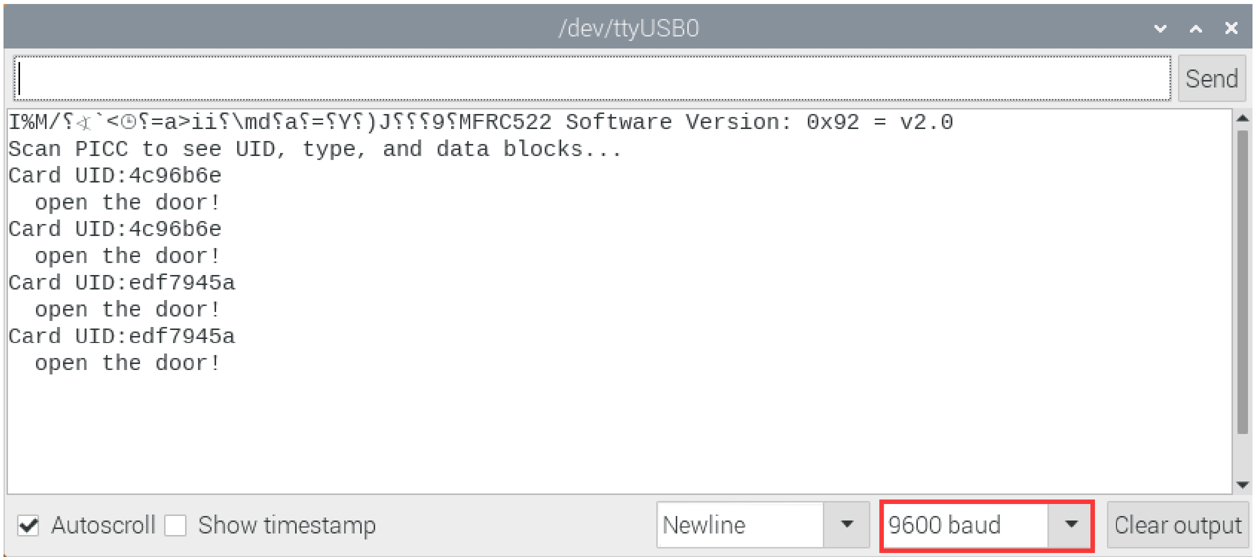
Code Explanation

In the previous experiment, our card swipe module has tested the information of IC card and key. Then we use this corresponding information to control the door.

Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 9600.

We need to press the reset button on the ESP32, when we use the IC card or blue key to swipe the card, the monitor displays the card and the key information and "open the door", at the same time, the servo rotates to the corresponding angle to simulate opening the door.

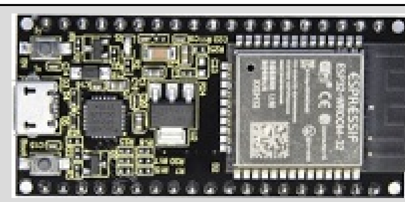


7.6.16 Project 60Bluetooth

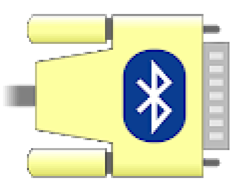
This chapter mainly introduces how to use the bluetooth of ESP32 for simple data transmission with mobile phone. Project 60.1 is conventional bluetooth, and Project 60.2 is bluetooth control LED.

Project 60.1Classic Bluetooth

Components

	
USB Cable*1	ESP32*1

In this experiment, we need to use a bluetooth dobbed serial bluetooth terminal for a study. If you haven't install it, please click the installation: https://www.appsapk.com/serial-bluetooth-terminal/.

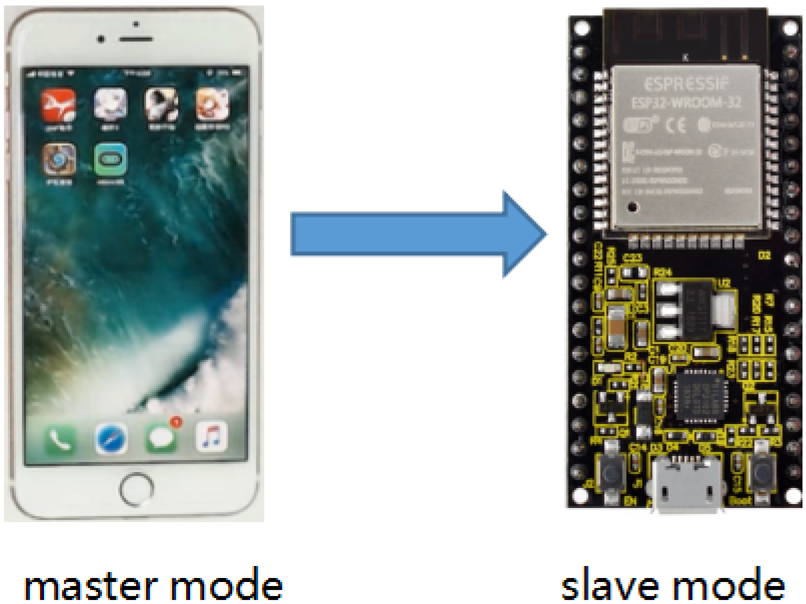
Here is its sign: 

Component Knowledge

Bluetooth is a short-distance communication system that can be divided into two types, namely low power bluetooth (BLE) and classic bluetooth. There are two modes for simple data transfer: master mode and slave mode.

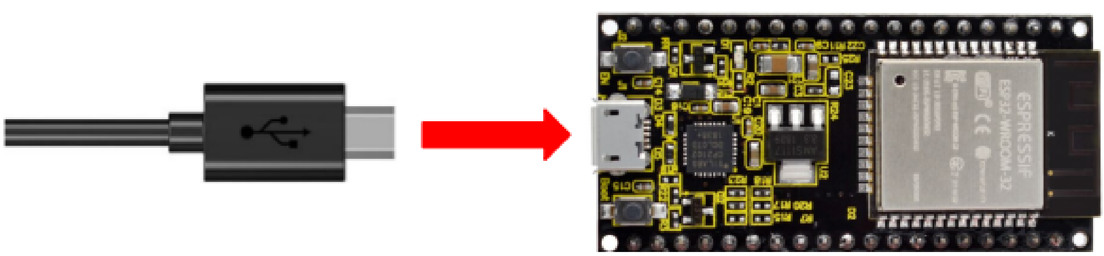
Master Mode: In this mode, work is done on the master device and can be connected to the slave device. When the device initiates a connection request in the main mode, information such as the address and pairing password of other bluetooth devices are required. Once paired, you can connect directly to them.

Slave Mode: A bluetooth module in the slave mode can only accept connection requests from the host, but cannot initiate connection requests. After being connected to a host device, it can send and receive data through the host device . Bluetooth devices can interact with each other, when they interact, the bluetooth device in the main mode searches for nearby devices. While a connection is established, they can exchange data. For example, when a mobile phone exchanges data with ESP32, the mobile phone is usually in master mode and the ESP32 is in slave mode.



Wiring Diagram

We can use a USB cable to connect ESP32 mainboard to the USB port on the Raspberry Pi.



Test Code

```
//**********************************************************************************
/*
 * Filename    : Classic Bluetooth
 * Description : ESP32 communicates with the phone by bluetooth and print phone's data via a serial port
 * Auther      : http//www.keyestudio.com
*/
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;
String buffer;
void setup() {
  Serial.begin(115200);
```

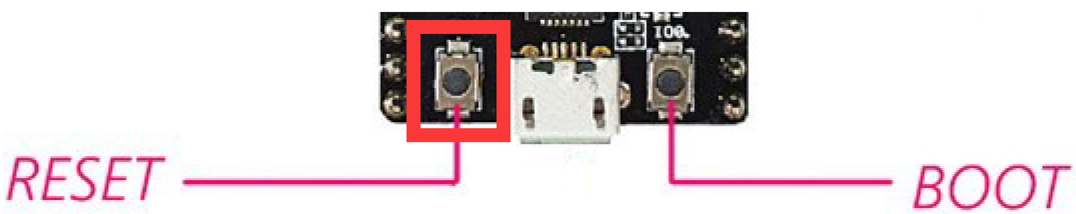
(continues on next page)

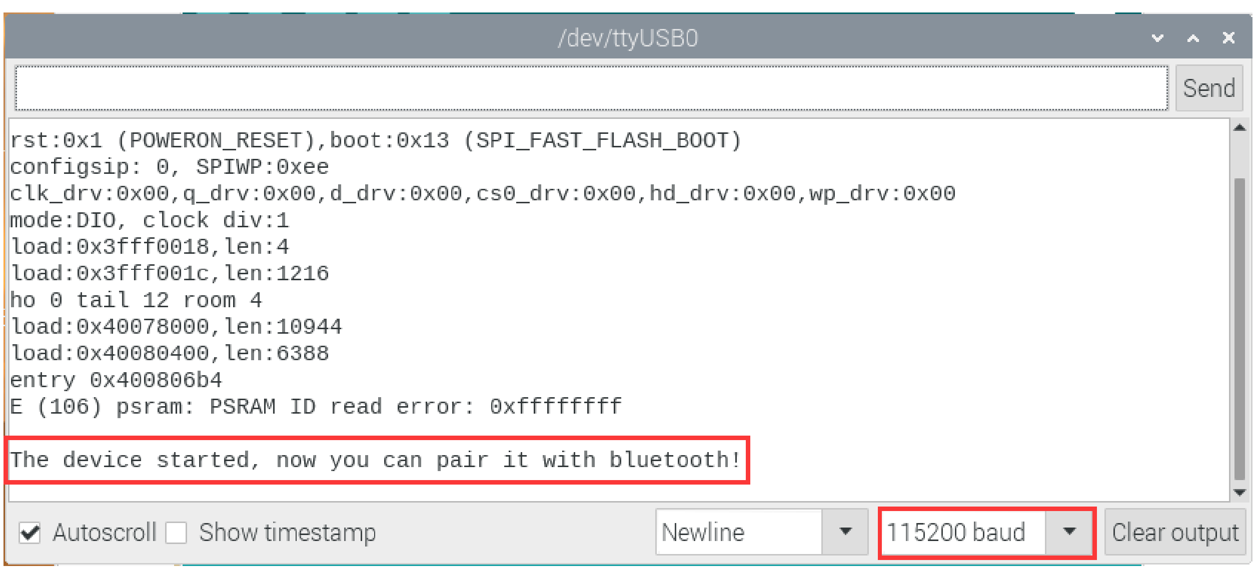
(continued from previous page)

```
  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);
}
//*************************************************************************************
```

Test Result

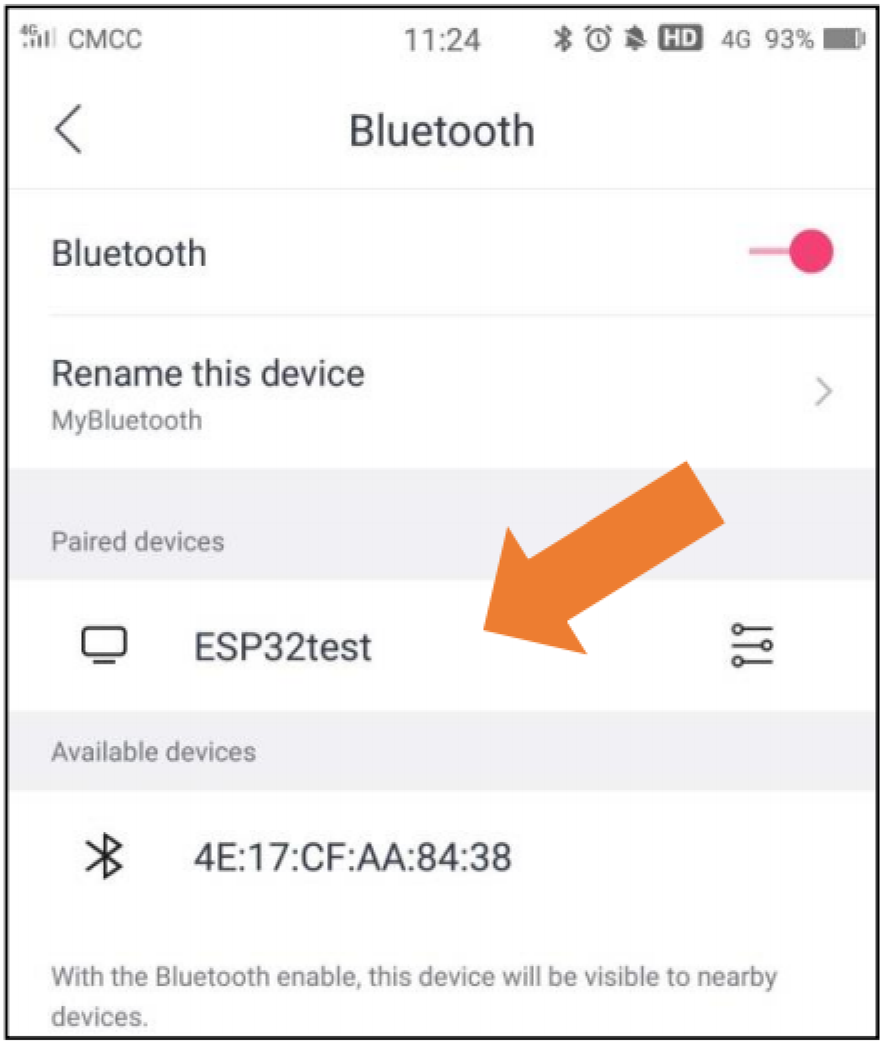
Compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200. We need to press the reset button on the ESP32, when you see the serial prints the character, as shown below, it means that the ESP32's bluetooth is waiting for connection with a phone. (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)



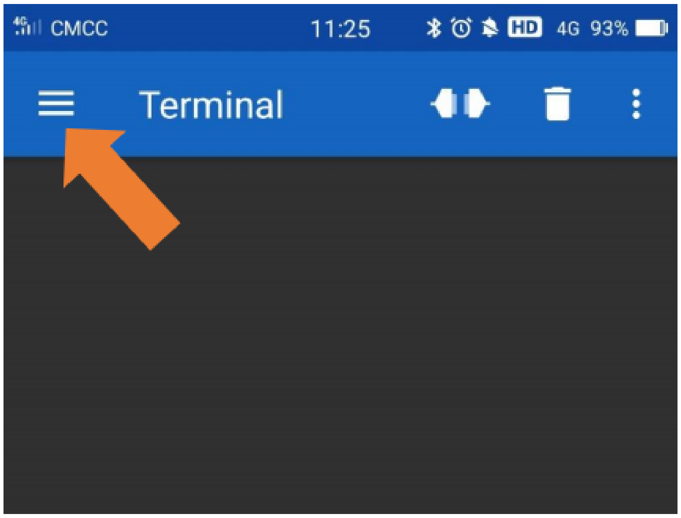


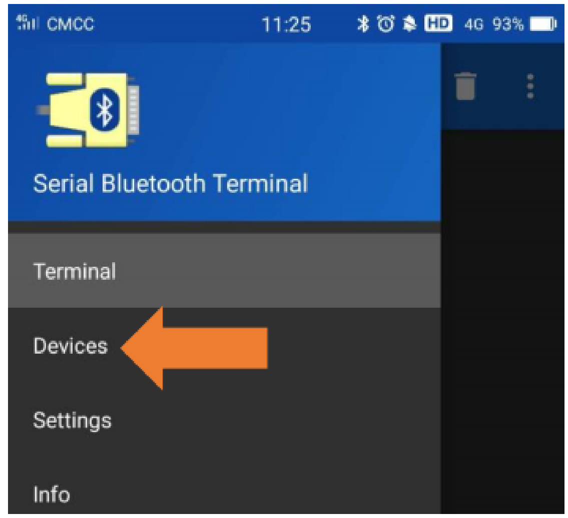
Ensure that your mobile phone bluetooth is enabled and the bluetooth application of "Serial Bluetooth Terminal" is installed.

Click"Search"search for the nearby bluetooth and select to connect the"ESP32 test".

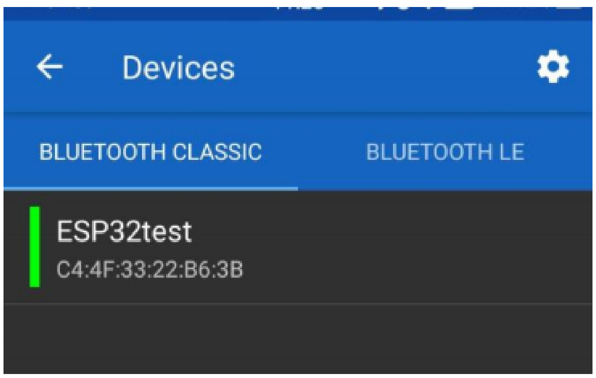


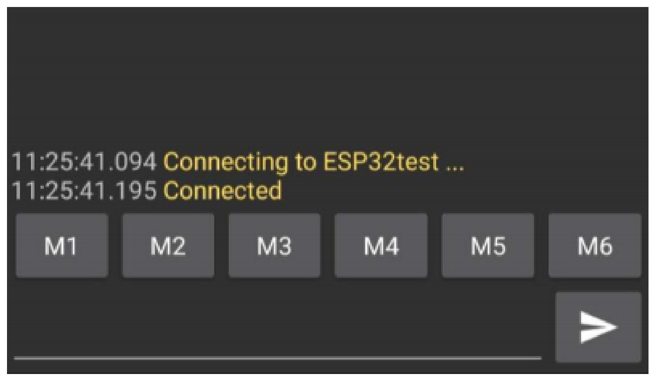
Open the software APP and click the left side of the terminal, select "Devices".





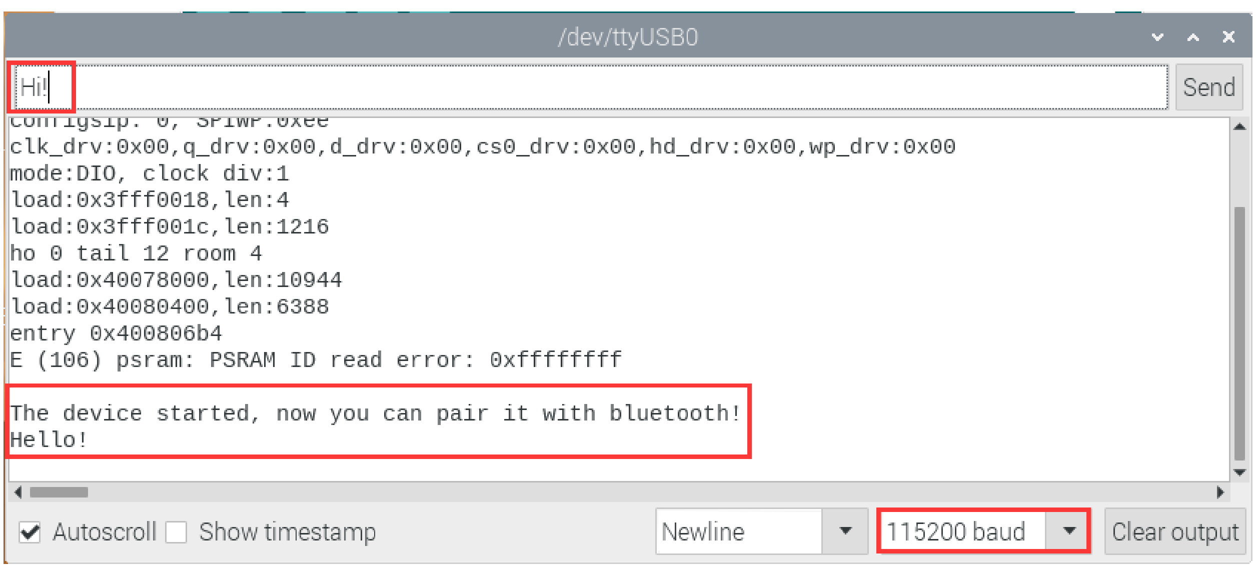
If you select ESP32test in classic bluetooth mode, a successful connection message will appear as shown below.

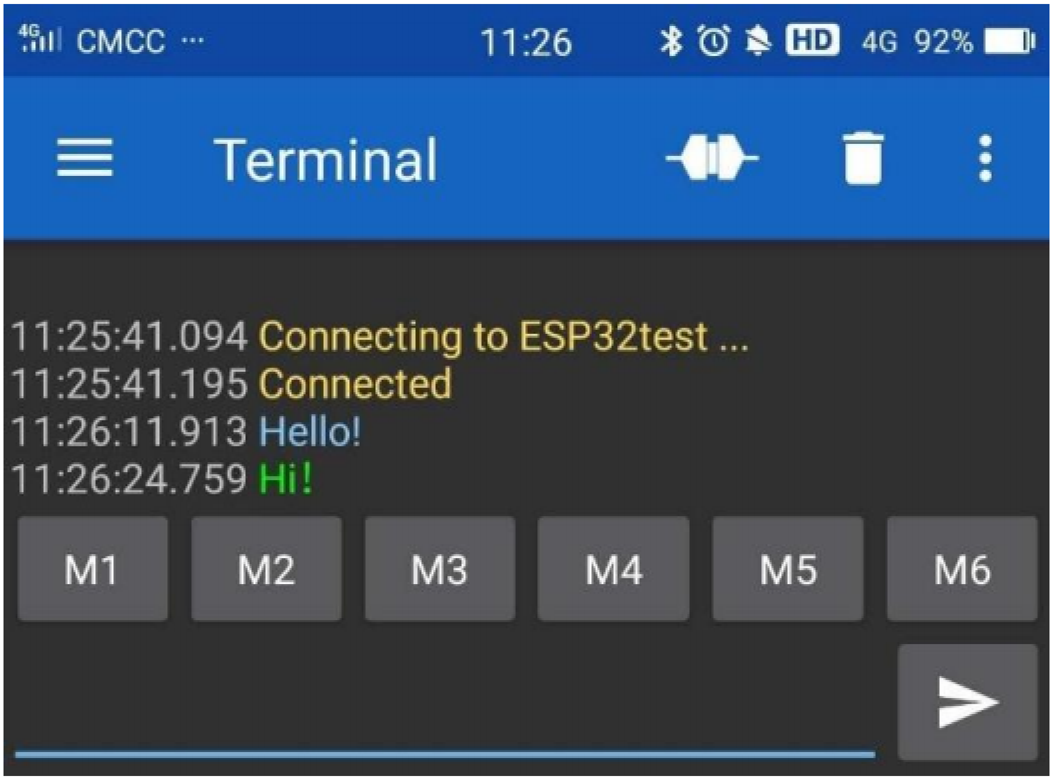




Data can be transferred between your phone and Raspberry Pi via ESP32 now.

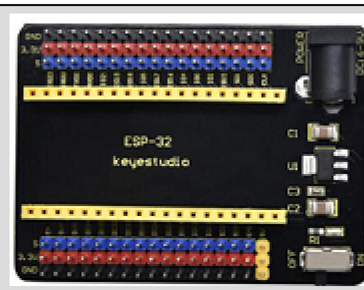
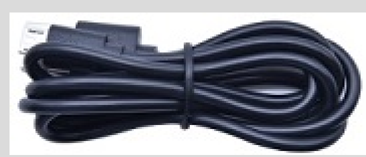
Send "Hello", When the Raspberry Pi receives it, which will reply with "Hi!".



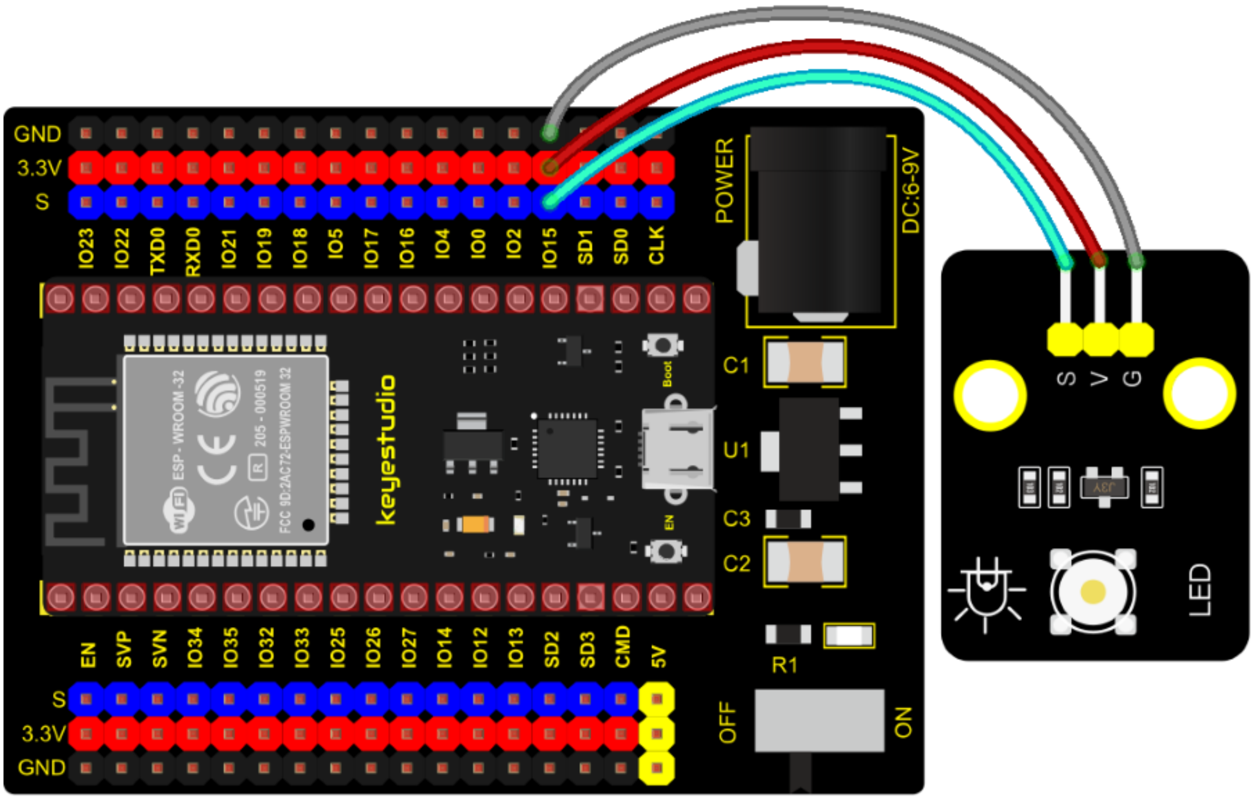


Project 60.2Bluetooth Control LED

Components

		
ESP32*1	ESP32 Expansion Board*1	MicroUSB Cable*1
		
Keyestudio Purple LED Module*1	3P Dupont*1	

Wiring Diagram



Test Code

```
//**********************************************************************************
/*
 * Filename    : Bluetooth Control LED
 * Description : The phone controls esp32's led via bluetooth.
                When the phone sends "LED_on," ESP32's LED lights turn on.
                When the phone sends "LED_off," ESP32's LED lights turn off.
 * Auther      : http//www.keyestudio.com
*/
#include "BluetoothSerial.h"
#include "string.h"
#define LED 15
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())
```

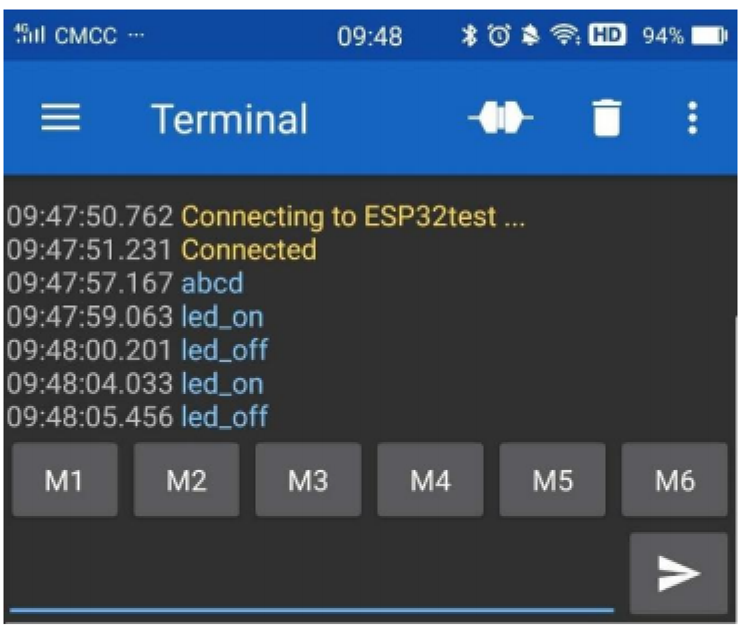
(continues on next page)

(continued from previous page)

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

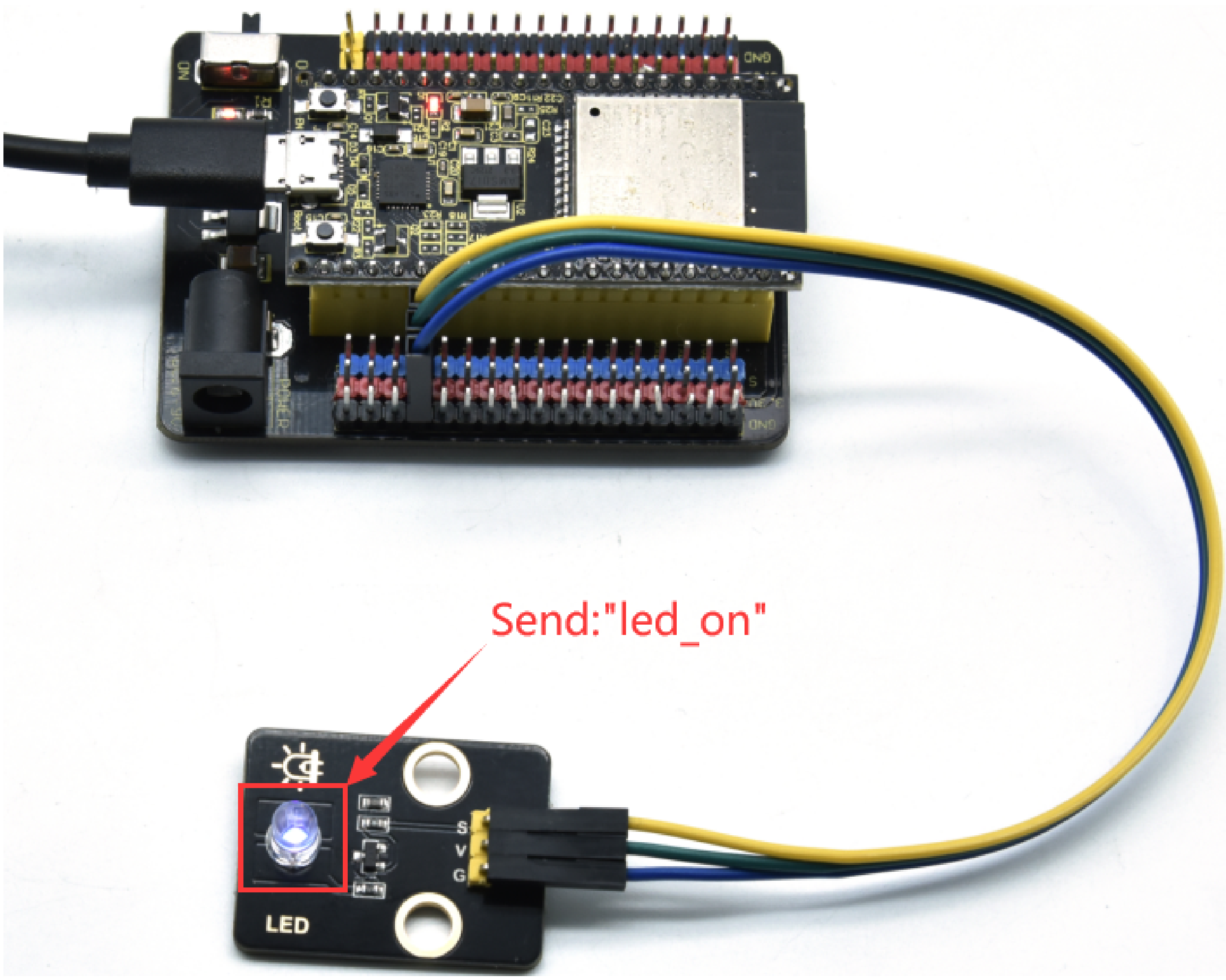
Test Result

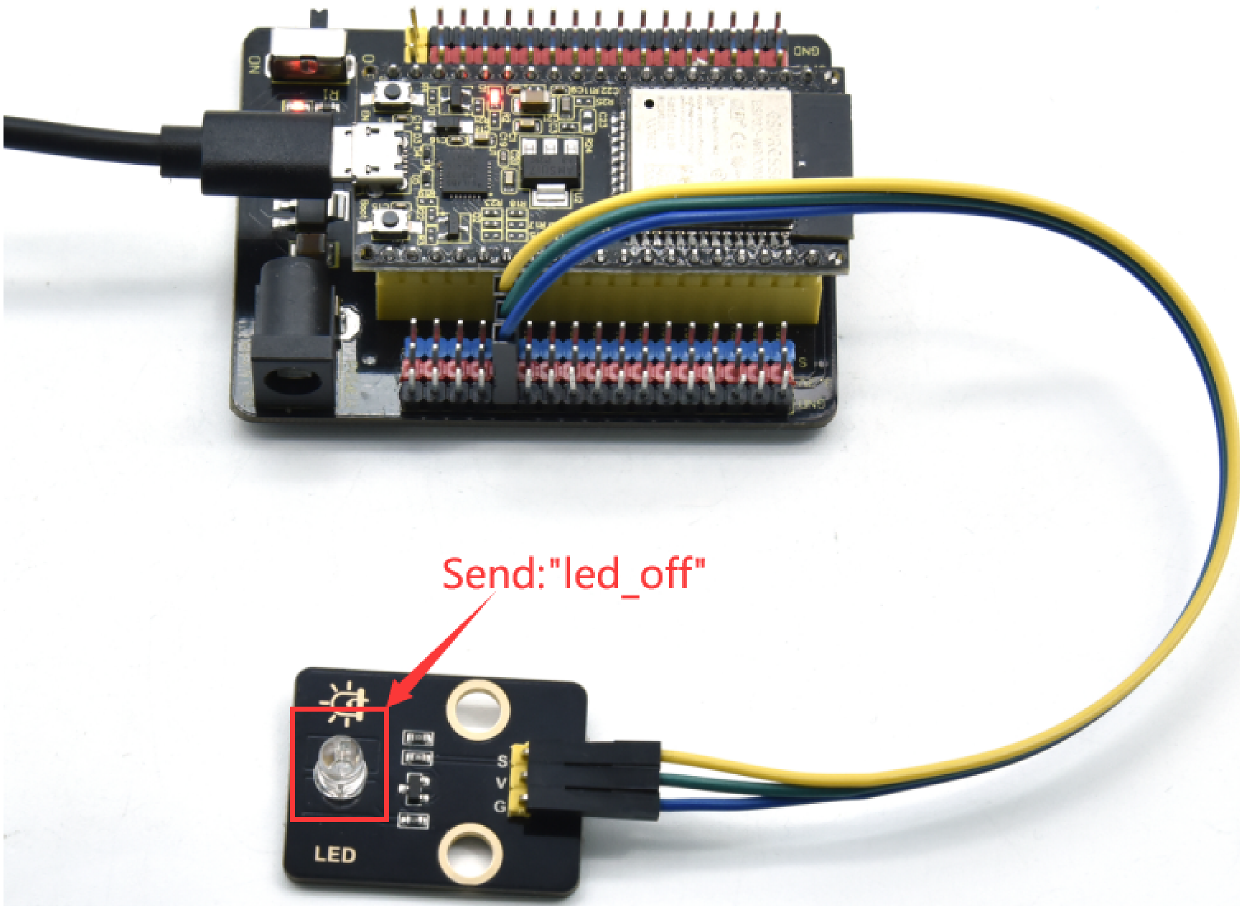
Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. The APP operation is the same as the project 60.1. We need to press the reset button on the ESP32, if you want to make the external LED on and off, simply change the sending content to "LED_on" and "LED_Off". Moving the APP to send data:



The serial monitor will display as follows:

LED Circumstance





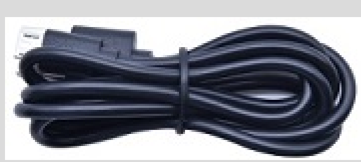
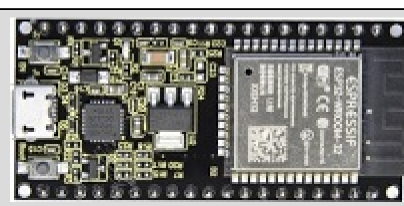
Note: If the sent content is not "led_on 'or" led_off ", the status of the LED will not change. If the LED is on, it remains on when irrelevant content is received; Conversely, if the LED is off, it continues to be off when irrelevant content is received.

7.6.17 Project 61WIFI Station Mode

Description

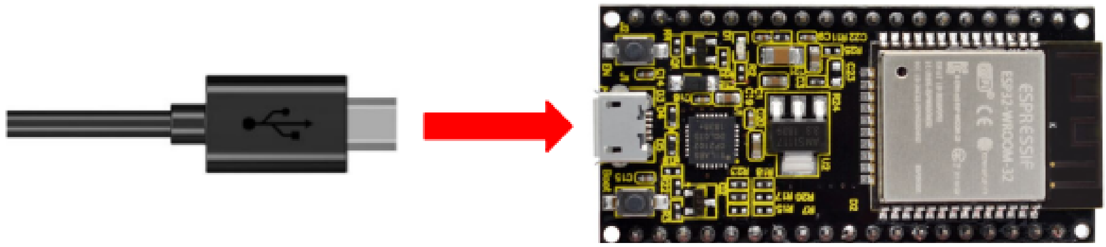
ESP32 has three different WiFi modes: Station mode, AP mode and AP+Station mode. All WiFi programming projects must be configured with WiFi running mode before using, otherwise the WiFi cannot be used. In this project, we are going to learn the WiFi Station mode of the ESP32.

Components

	
Micro USB Cable*1	ESP32 Board*1

Wiring Diagram

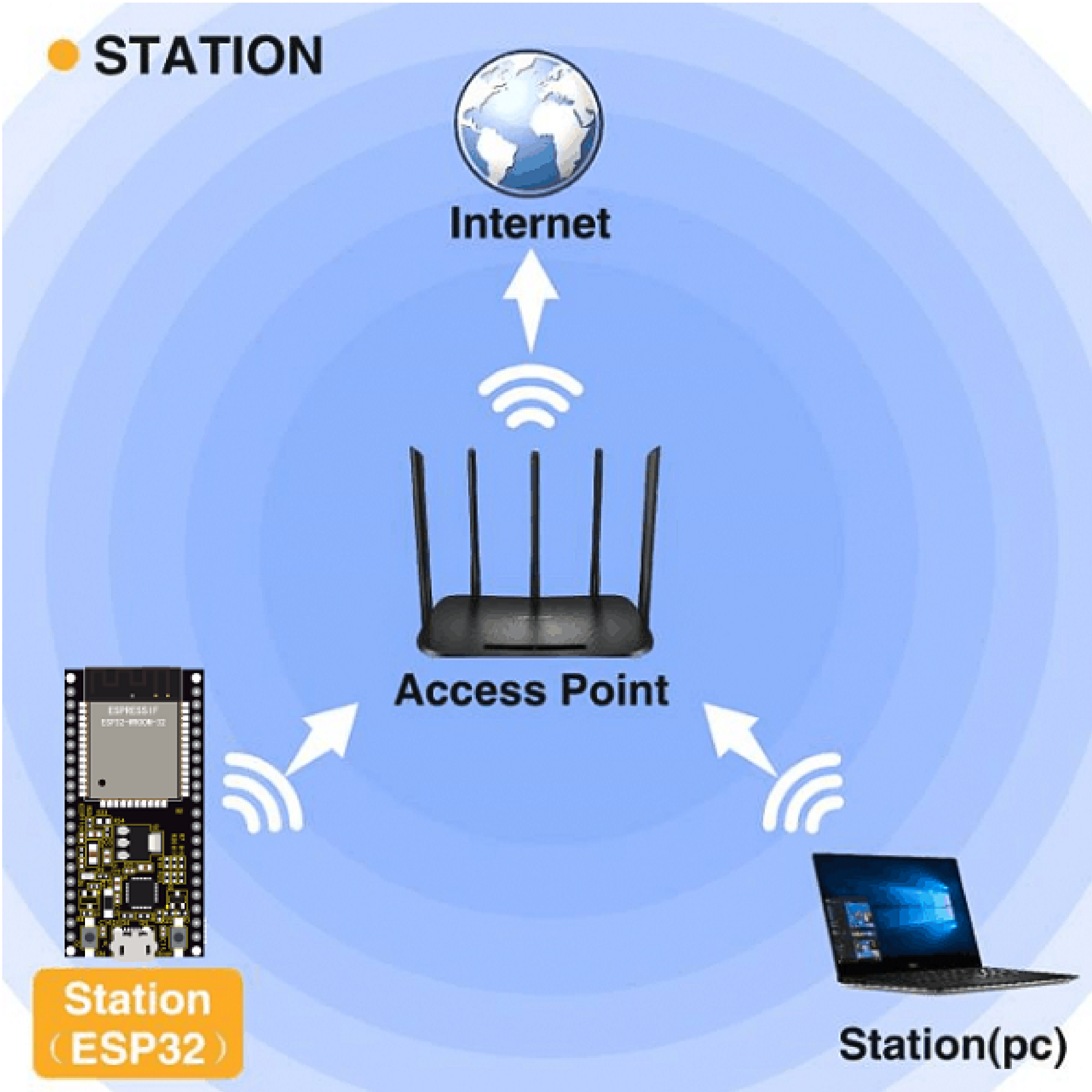
Plug the ESP32 to the USB port of your Raspberry Pi.



Component Knowledge

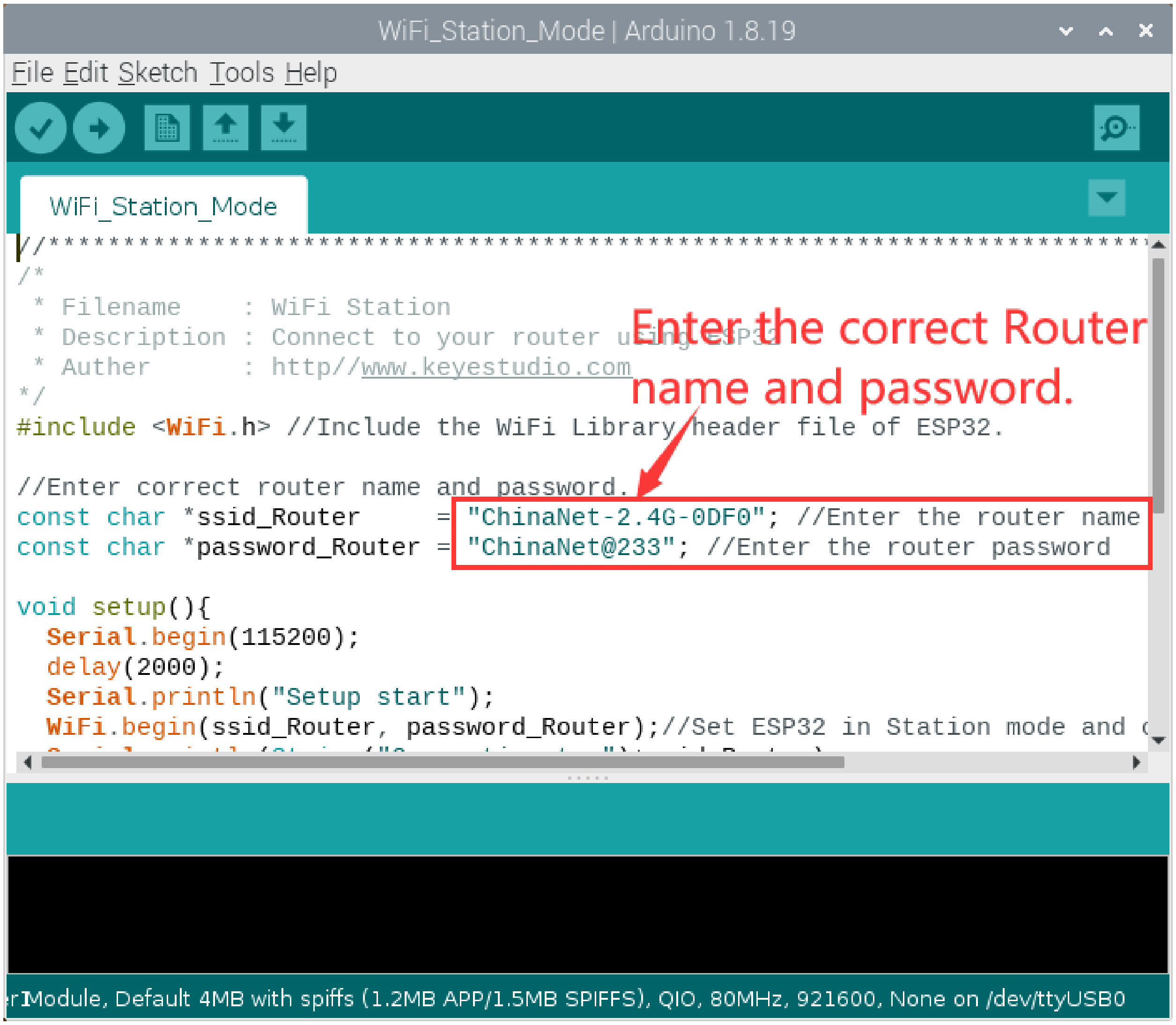
Station mode

When setting Station mode, the ESP32 is taken as a WiFi client. It can connect to the router network and communicate with other devices on the router via a WiFi connection. As shown in the figure below, the PC and the router have been connected. If the ESP32 wants to communicate with the PC, the PC and the router need to be connected.



Test Code

Since WiFi names and passwords vary from place to place, thereby users need to enter the correct WiFi names and passwords in the box shown below before the program code runs.




```
//**********************************************************************************
/*
 * Filename    : WiFi Station
 * Description : Connect to your router using ESP32
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h> //Include the WiFi Library header file of ESP32.

//Enter correct router name and password.
const char *ssid_Router     = "ChinaNet-2.4G-0DF0"; //Enter the router name
const char *password_Router = "ChinaNet@233"; //Enter the router password

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setup start");
  WiFi.begin(ssid_Router, password_Router);//Set ESP32 in Station mode and connect it to␣
↪your router.
```

(continues on next page)

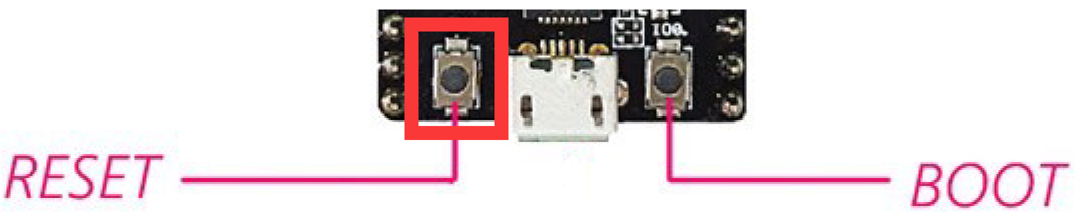
(continued from previous page)

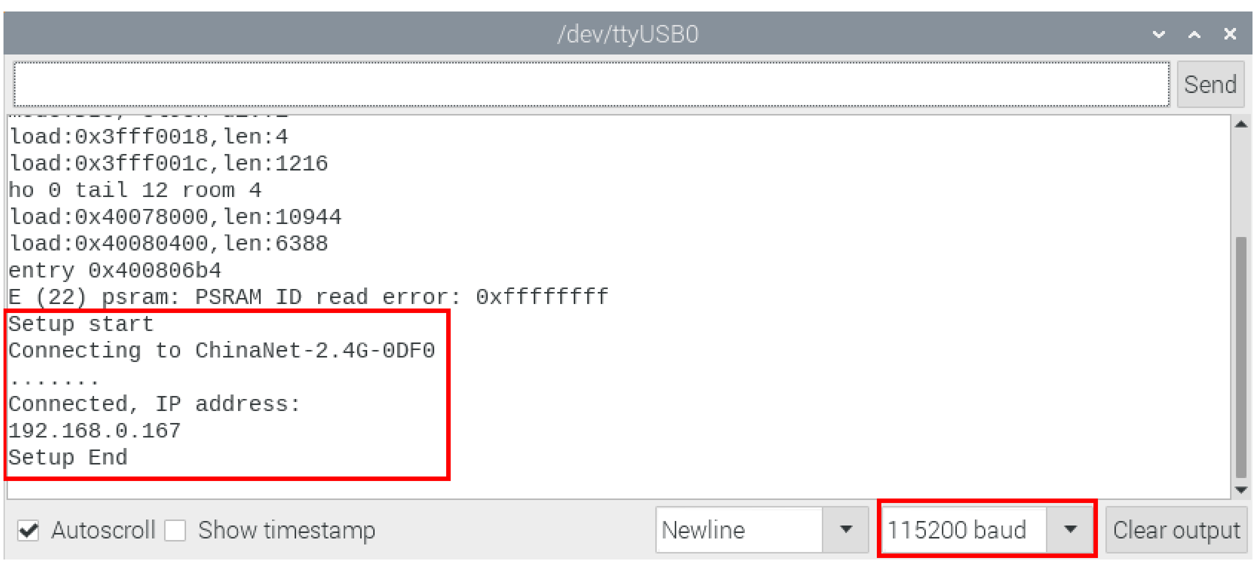
```
  Serial.println(String("Connecting to ")+ssid_Router);
//Check whether ESP32 has connected to router successfully every 0.5s.
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected, IP address: ");
  Serial.println(WiFi.localIP());//Serial monitor prints out the IP address assigned to ESP32.
  Serial.println("Setup End");
}

void loop() {
}
//*************************************************************************************
```

Test Result

After entering the correct WiFi names and passwords, compile and upload the code to the ESP32. After uploading successfullywe will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200.

When the ESP32 successfully connects to ssid_WiFi, the serial monitor prints out the IP address, then monitor will display as follows: (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)



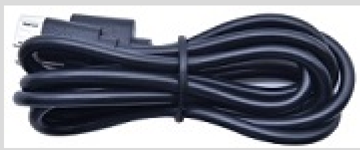
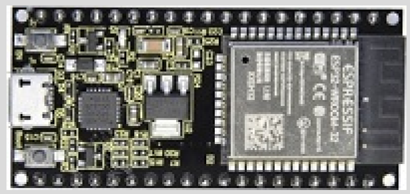


7.6.18 Project 62WIFI AP Mode

Description

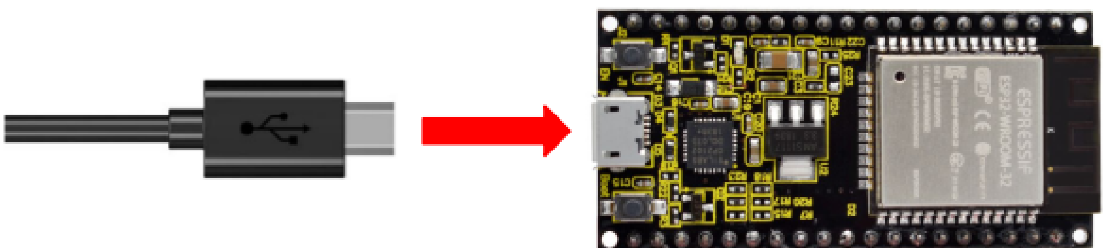
ESP32 has three different WiFi modes: Station mode, AP mode and AP+Station mode. All WiFi programming projects must be configured with WiFi running mode before using, otherwise the WiFi cannot be used. In this project, we are going to learn the WiFi AP mode of the ESP32.

Components

	
Micro USB Cable*1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your Raspberry Pi



Component Knowledge

AP Mode:

When setting AP mode, a hotspot network will be created, waiting for other WiFi devices to connect. As shown below;

Take the ESP32 as the hotspot, if a phone or PC needs to communicate with the ESP32, it must be connected to the ESP32's hotspot. Communication is only possible after a connection is established via the ESP32.

Test Code

Before the program code runs, you can make any changes to the ESP32 AP name and password in the box as shown below, but in a default circumstance, it doesn't need to modify.


```
//**************************************************************************
/*
 * Filename    : WiFi AP
 * Description : Set ESP32 to open an access point
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h> //Include the WiFi Library header file of ESP32.

const char *ssid_AP     = "ESP32_Wifi"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password

IPAddress local_IP(192,168,1,108);//Set the IP address of ESP32 itself
IPAddress gateway(192,168,1,1);   //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);
```

(continues on next page)

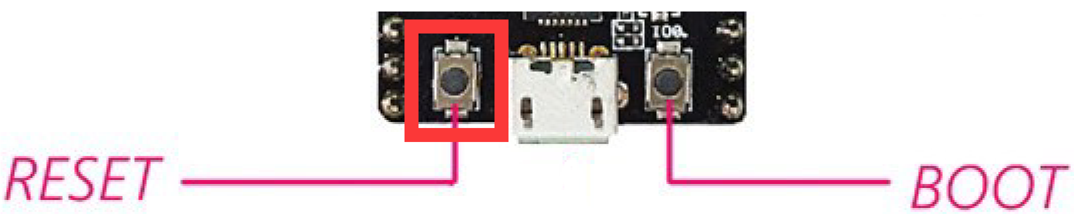
(continued from previous page)

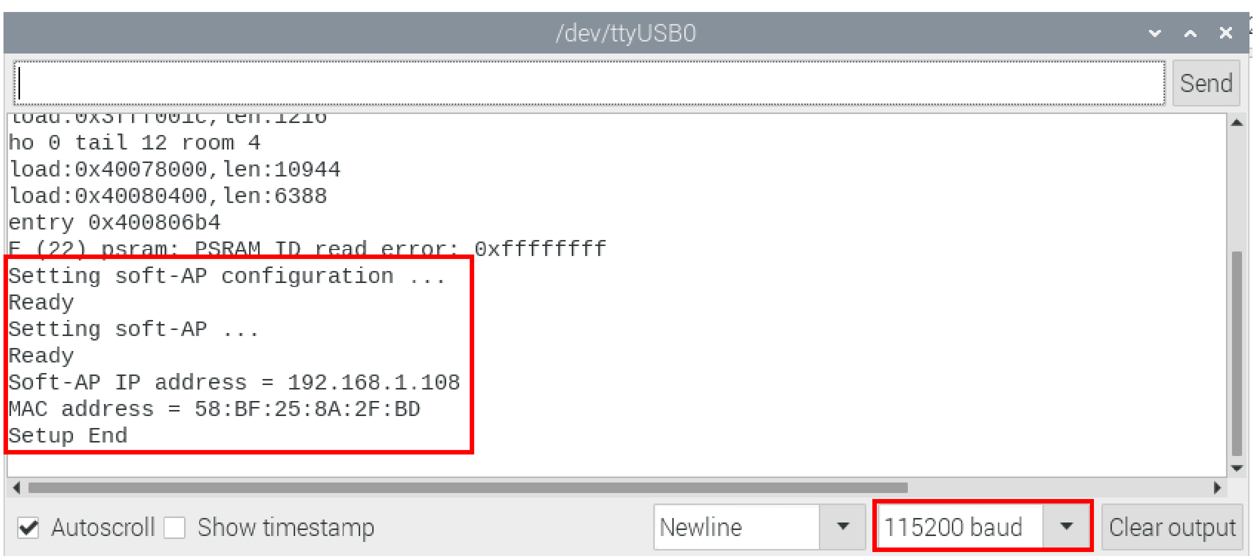
```
  Serial.println("Setting soft-AP configuration ... ");
  WiFi.disconnect();
  WiFi.mode(WIFI_AP);
  Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
  Serial.println("Setting soft-AP ... ");
  boolean result = WiFi.softAP(ssid_AP, password_AP);
  if(result){
    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() {
}
//*********************************************************************************
```

Test Result

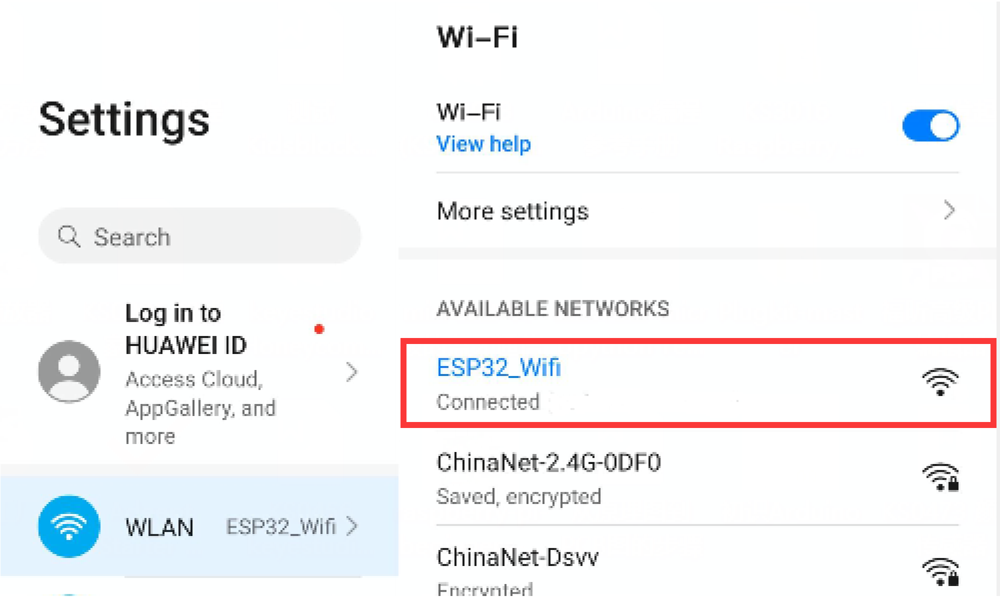
Compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200, then monitor will display as follows:

(If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)





When observing the printed information of the serial port monitor, turn on the WiFi scanning function of the mobile phone, you can see the ssid_AP on ESP32, which is dubbed "ESP32_Wifi" in this program code. You can connect to it either by typing the password "12345678" or by modifying the program code to change its AP name and password.

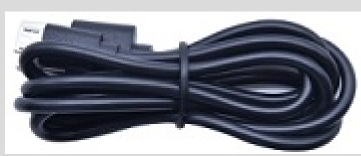
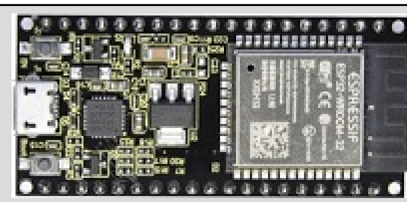


7.6.19 Project 63WIFI AP+Station Mode

Description

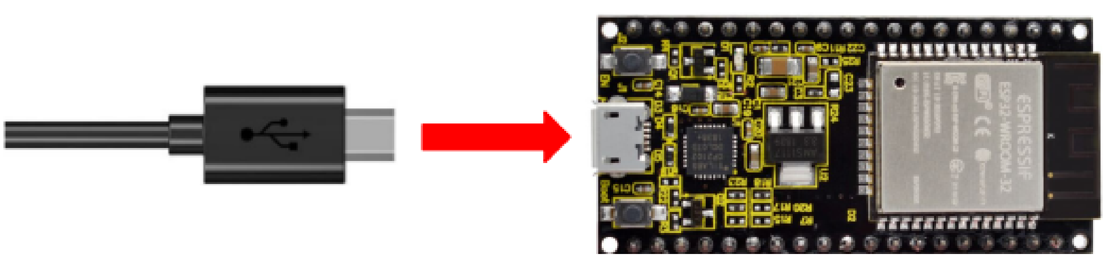
In this project, we are going to learn the AP+Station mode of the ESP32.

Components

	
Micro USB Cable*1	ESP32*1

Wiring Diagram

Plug the ESP32 mainboard to the USB port of your Raspberry Pi



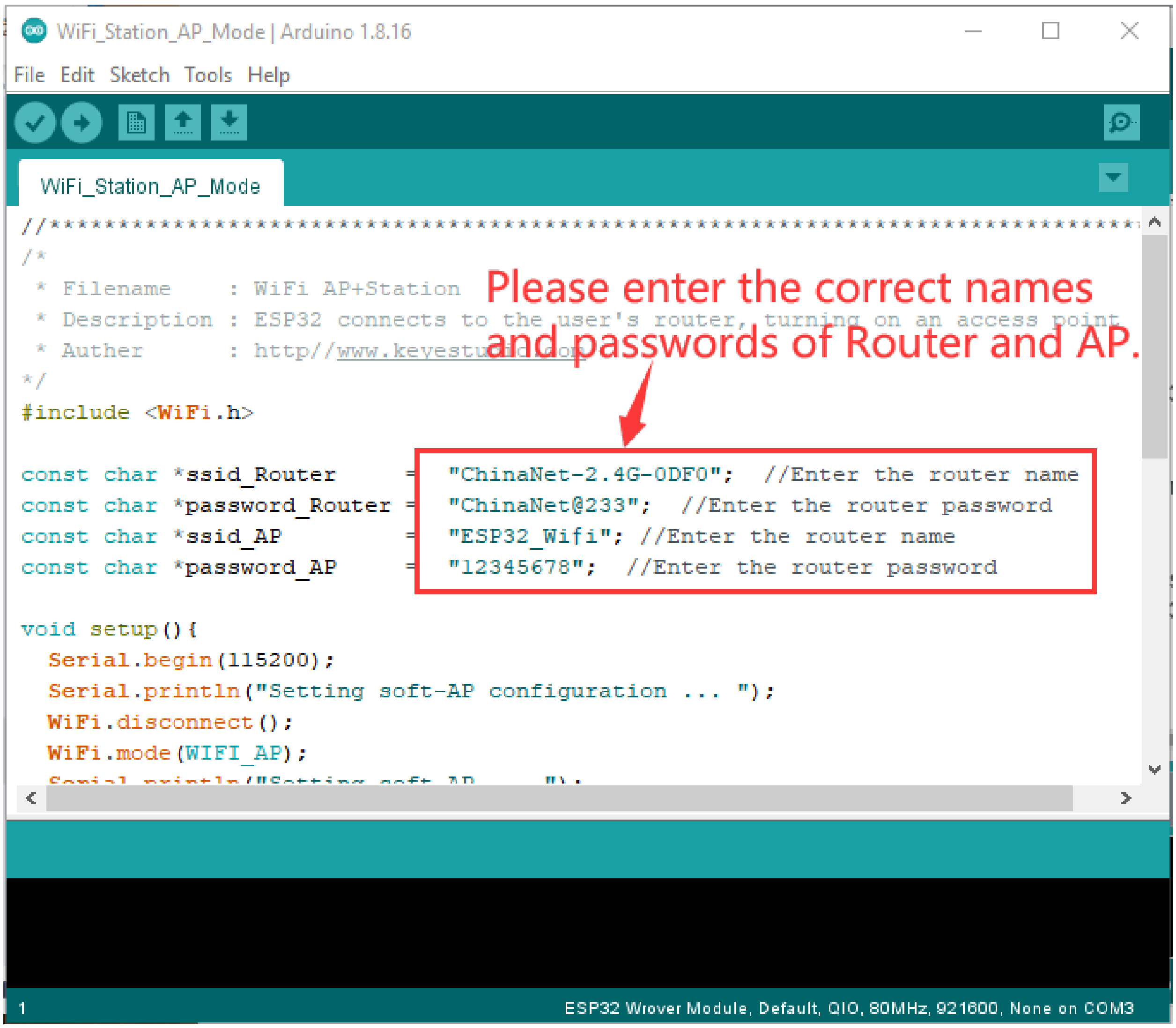
Component Knowledge

AP+Station mode

In addition to the AP mode and the Station mode, **AP+Station mode** can be used at the same time. Turn on the Station mode of the ESP32, connect it to the router network, and it can communicate with the Internet through the router. Then turn on the AP mode to create a hotspot network. Other WiFi devices can be connected to the router network or the hotspot network to communicate with the ESP32.

Test Code

Before the program code runs, you need to modify the ssid_Router, password_Router, ssid_AP and password_AP, as shown in the box below:




```
//**********************************************************************************
/*
 * Filename    : WiFi AP+Station
 * Description : ESP32 connects to the user's router, turning on an access point
 * Auther      : http//www.keyestudio.com
*/
#include <WiFi.h>
```

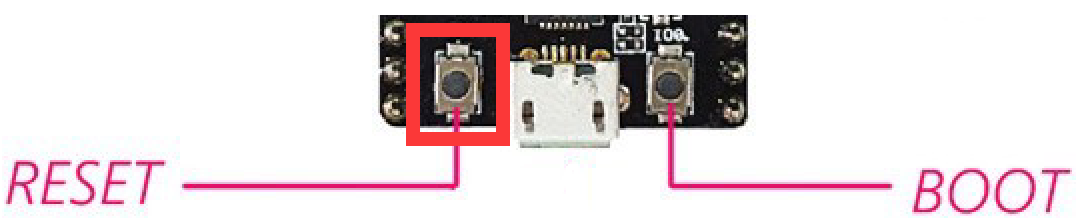
(continues on next page)

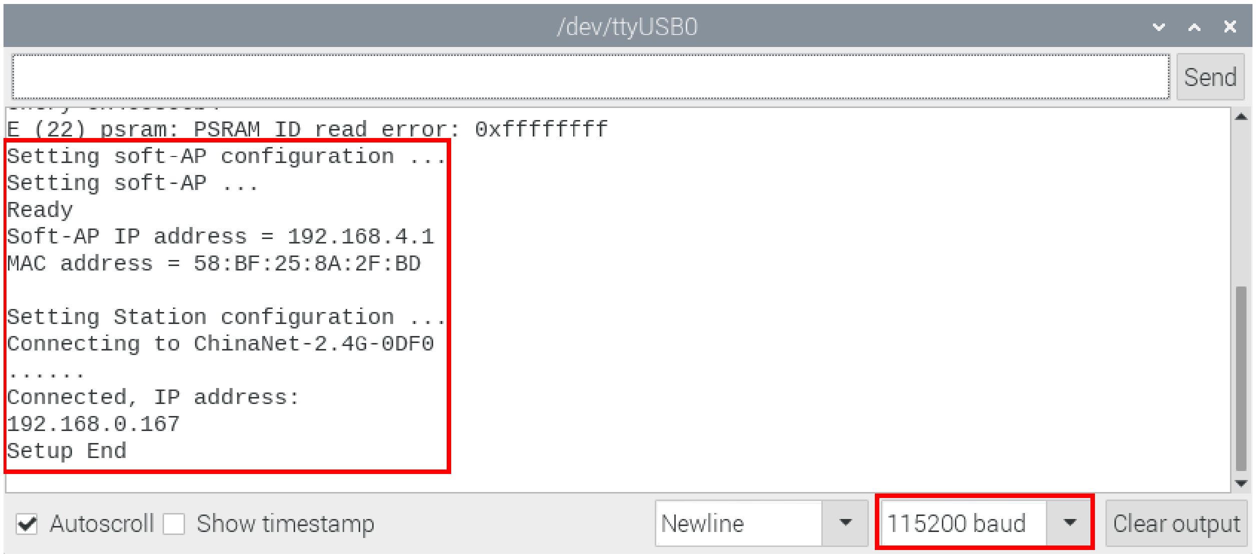
(continued from previous page)

```
const char *ssid_Router     =  "ChinaNet-2.4G-0DF0";  //Enter the router name
const char *password_Router =  "ChinaNet@233";  //Enter the router password
const char *ssid_AP         =  "ESP32_Wifi"; //Enter the router name
const char *password_AP     =  "12345678";  //Enter the router 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() {
}
//*************************************************************************************
```

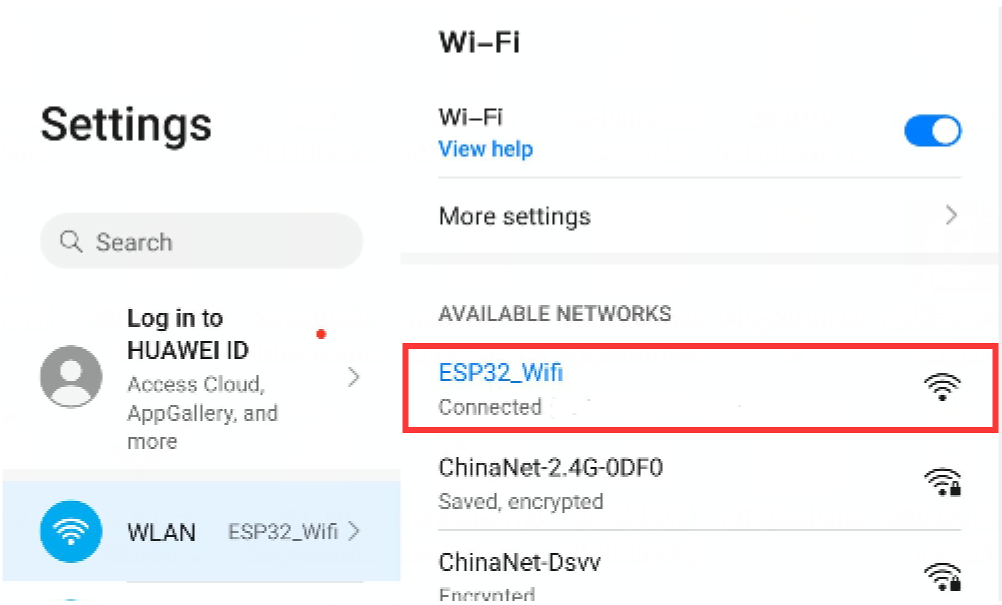
Test Result

Ensure that the code in the program has been modified correctly, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. Open the serial monitor and set the baud rate to 115200, then monitor will display as follows: (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)

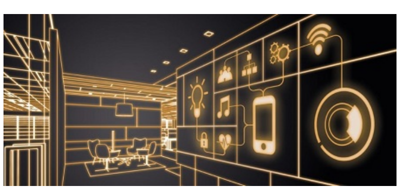




Open the WiFi scanning function of the mobile phone, you can see the ssid_AP.



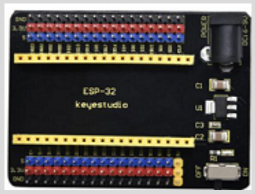
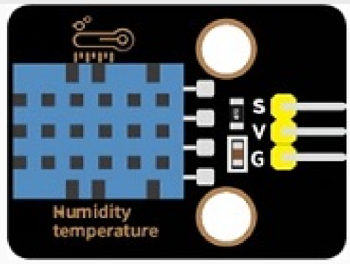
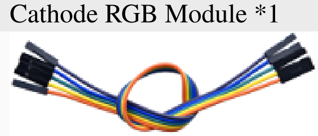
7.6.20 Project 64: Comprehensive Experiment



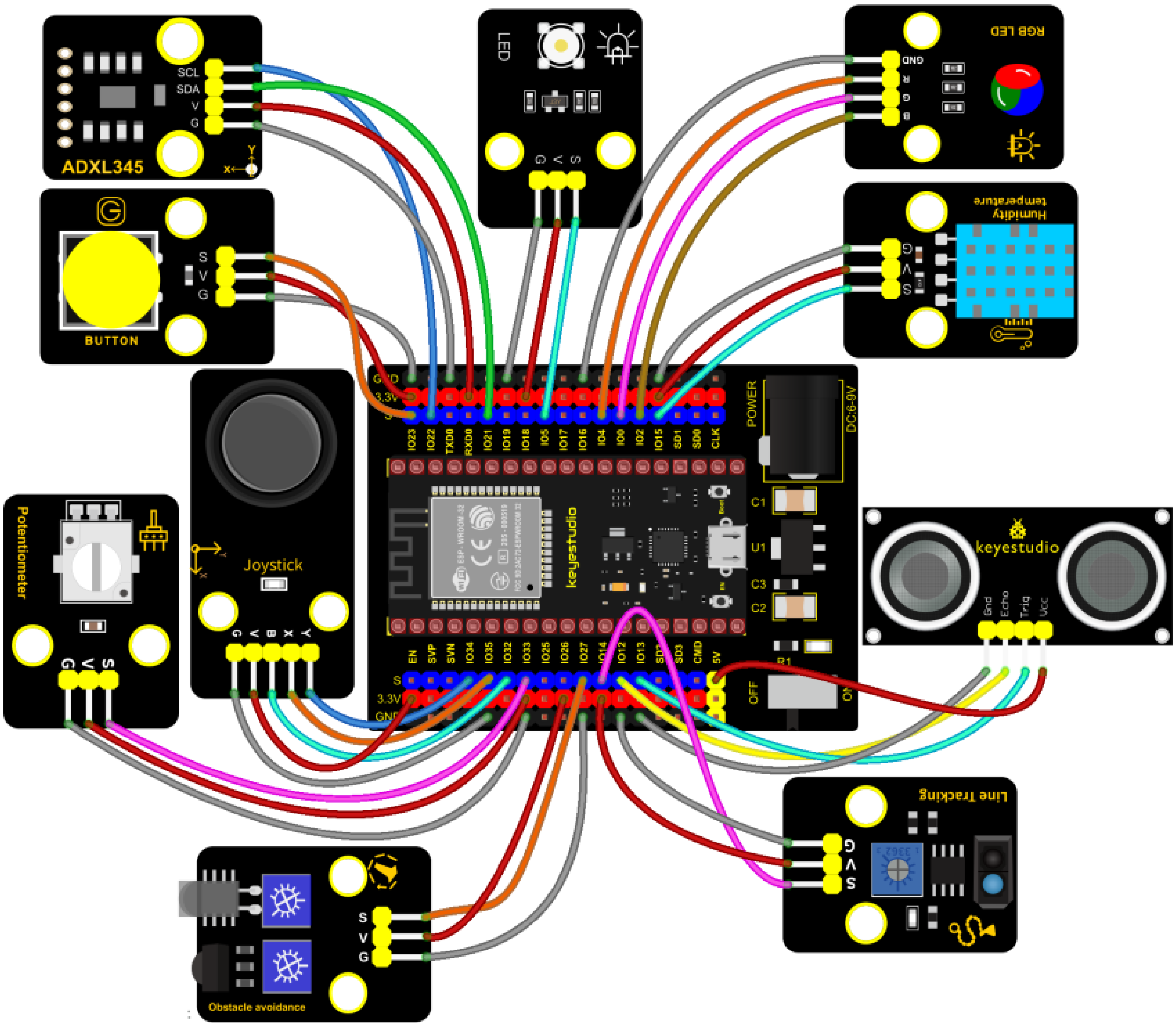
Introduction

We did a lot of experiments, and for each one we needed to re-upload the code, so can we achieve different functions through an experiment? In this experiment, we will use an external button module to achieve different functions.

Components Required

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>ESP32 Board*1</td><td>ESP32 Expansion Board*1</td><td>Keyestudio DIY Purple LED Module*1</td><td>Keyestudio Button Module*1</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Keyestudio Poten-tiometer*1</td><td>Keyestudio Obstacle Avoidance Sensor*1</td><td>Keyestudio XHT11 Tempera-ture and Humidity Sensor *1</td><td>Keyestudio ADXL345 Accel-eration Sensor*1</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Keyestudio Line Tracking Sensor*1</td><td>Keyestudio DIY Joy-stick Module*1</td><td>Keyestudio HC-SR04 Ultra-sonic sensor *1</td><td>Keyestudio DIY Common Cathode RGB Module *1</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Micro USB Cable*1</td><td>3P Dupont Wire*6</td><td>4P Dupont Wire*3</td><td>5P Dupont Wire*1</td></tr>
</table>

Wiring Diagram



Test Code

```
//**********************************************************************
/*
 * Filename    : Comprehensive experiment
 * Description : Multiple sensors/modules work together
 * Auther      : http//www.keyestudio.com
*/
#include "xht11.h"
#include "adxl345_io.h"

//ADXL345 sda-->21,scl-->22
adxl345 adxl345(21, 22);

//xht11 to gpio15
xht11 xht(15);

//rgb is connected to 4,0,2
int ledPins[] = {4, 0, 2};    //define red, green, blue led pins
```

(continues on next page)

(continued from previous page)

```
const byte chns[] = {0, 1, 2};        //define the pwm channels
int red, green, blue;

//Rocker module port
int X = 35;
int Y = 34;
int KEY = 32;

//Potentiometer pin is connected to analog port 33
int resPin = 33;

//Trace sensor pin connected to IO port 14
int TrackingPin = 14;

//LED is Connected to GP5
#define PIN_LED   5  // the pin of the LED
#define CHAN    3

//Obstacle avoidance sensor is connected to GP27
int Avoid = 27;

//Ultrasonic sensor port
int Trig = 13;
int Echo = 12;

//Key module port
int button = 23;

int PushCounter = 0;//Store the number of times a key is pressed
int yushu = 0;
unsigned char dht[4] = {0, 0, 0, 0};//Only the first 32 bits of data are received, not␣
↪the parity bits
bool ir_flag = 1;
float out_X, out_Y, out_Z;

void counter() {
  delay(10);
  ir_flag = 0;
  if (!digitalRead(button)) {
    PushCounter++;
  }
}

void setup() {
  Serial.begin(9600);//Set baud rate to 9600
  pinMode(KEY, INPUT);//Button of remote sensing module
  ledcSetup(CHAN, 1000, 12);
  ledcAttachPin(PIN_LED, CHAN);
  pinMode(button, INPUT);//The key module
  attachInterrupt(digitalPinToInterrupt(button), counter, FALLING);  //External␣
↪interrupt 0, falling edge fired
  pinMode(Avoid, INPUT);//Obstacle avoidance sensor
```

(continues on next page)

(continued from previous page)

```
  pinMode(Trig, OUTPUT);//Ultrasonic module
  pinMode(Echo, INPUT);
  adxl345.Init();
  for (int i = 0; i < 3; i++) {   //setup the pwm channels,1KHz,8bit
    ledcSetup(chns[i], 1000, 8);
    ledcAttachPin(ledPins[i], chns[i]);
  delay(1000);
 }
}

void loop() {
  yushu = PushCounter % 8;
  if (yushu == 0) {  //The remainder is 0
    yushu_0();  //rgb displays
  } else if (yushu == 1) {  //The remainder is 1
    yushu_1();  //Displays the high and low levels read by the tracking sensor
  } else if (yushu == 2) {  //The remainder is 2
    yushu_2();  //Display temperature and humidity value
  } else if (yushu == 3) {  //The remainder is 3
    yushu_3();  //Displays the rocker value
  }else if (yushu == 4) {  //The remainder is 4
    yushu_4();  //Display potentiometer ADC value and potentiometer control LED
  } else if (yushu == 5) {  //The remainder is 5
    yushu_5();  //Obstacle avoidance sensor detects obstacles
  } else if (yushu == 6) {  //The remainder is 6
    yushu_6();  //Shows the distance detected by ultrasound
  } else if (yushu == 7) {  //The remainder is 7
    yushu_7();  //ADXL345 triaxial acceleration value
  }
}

//RGB
void yushu_0() {
  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);
}

void yushu_1() {
  int val = digitalRead(TrackingPin);//Read the digital level output by the tracking␣
↪sensor
  Serial.print(val);//Serial port print value
  if (val == 0) {//White val is 0 detected
    Serial.print("        ");
    Serial.println("White");
```

(continues on next page)

(continued from previous page)

```
    delay(100);
  }
  else {//Black val is 1 detected
    Serial.print("        ");
    Serial.println("Black");
    delay(100);
  }
}

void yushu_2() {
  if (xht.receive(dht)) { //Returns true when checked correctly
    Serial.print("RH:");
    Serial.print(dht[0]); //The integral part of humidity, DHT [1] is the fractional part
    Serial.print("%  ");
    Serial.print("Temp:");
    Serial.print(dht[2]); //The integral part of temperature, DHT [3] is the fractional␣
↪part
    Serial.println("C");
  } else {    //read error
    Serial.println("sensor error");
  }
  delay(1200);
}

void yushu_3() {
  int x = analogRead(X);
  int y = analogRead(Y);
  int key = digitalRead(KEY);
  Serial.print("X:");
  Serial.print(x);
  Serial.print("    Y:");
  Serial.print(y);
  Serial.print("    KEY:");
  Serial.println(key);
  delay(100);
}

void yushu_4() {
  int adcVal = analogRead(resPin); //read adc
  Serial.println(adcVal);
  int pwmVal = adcVal;        // adcVal re-map to pwmVal
  ledcWrite(CHAN, pwmVal);    // set the pulse width.
  delay(10);
}

void yushu_5() {
  int val = digitalRead(Avoid);
  if (val == 0) {//Obstruction detected
    Serial.println("There are obstacles");
  }
  else {//No obstructions detected
    Serial.println("All going well");
```

(continues on next page)

(continued from previous page)

```
  }
  delay(100);
}

void yushu_6() {
  float distance = checkdistance();
  Serial.print("distance:");
  Serial.print(distance);
  Serial.println("cm");
  delay(100);
}

void yushu_7() {
  adxl345.readXYZ(&out_X, &out_Y, &out_Z);
  Serial.print(out_X);
  Serial.print("g   ");
  Serial.print(out_Y);
  Serial.print("g   ");
  Serial.print(out_Z);
  Serial.println("g");
  delay(100);
}

float checkdistance() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  float distance = pulseIn(Echo, HIGH) / 58.00;
  delay(10);
  return distance;
}
//*************************************************************************************
```

Code Explanation

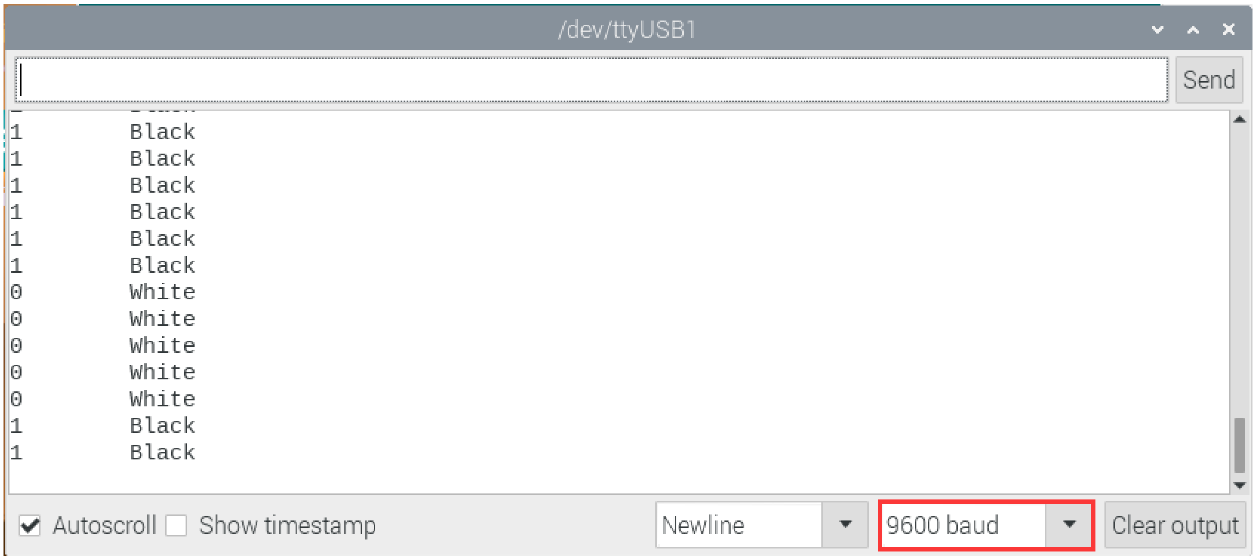
1). Calculate how many times the button is pressed, divide it by 8, and get the remainder which is 0, 1 2, 3, 4, 5 , 6 and 7. According to different remainders, construct eight unique functions to control the experiment and realize different functions.

2). Following the instructions, we can add or remove sensors/modules in the wiring, and then change the experimental function in the code.

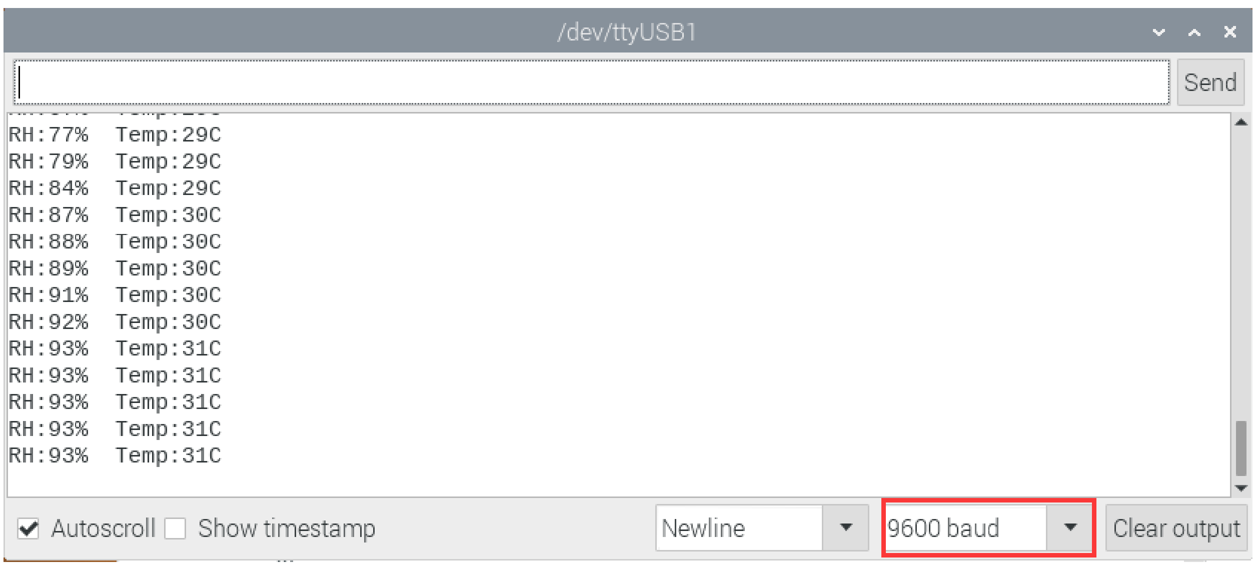
Test Result

Connect the wires according to the experimental wiring diagram, compile and upload the code to the ESP32. After uploading successfully, we will use a USB cable to power on. At the beginning, the number of the button is 0 and remainder is 0. Open the monitor and set baud rate to 9600.

Press the button, the RGB stops flashing, press once, the remainder is 1. The function of the experiment is to detect black objects and white objects by a line tracking sensor. If the sensor does not detect an object or detects a black object, val is 1, and the serial monitor displays the character “1 Black”. When a white object (reflective) is detected, val is 0 and the serial monitor displays the character “0 White”, the serial monitor will display as follows:



Press a key twice, the time of pressing buttons is 2 and the remainder is 2. Read temperature and humidity values. As shown below;



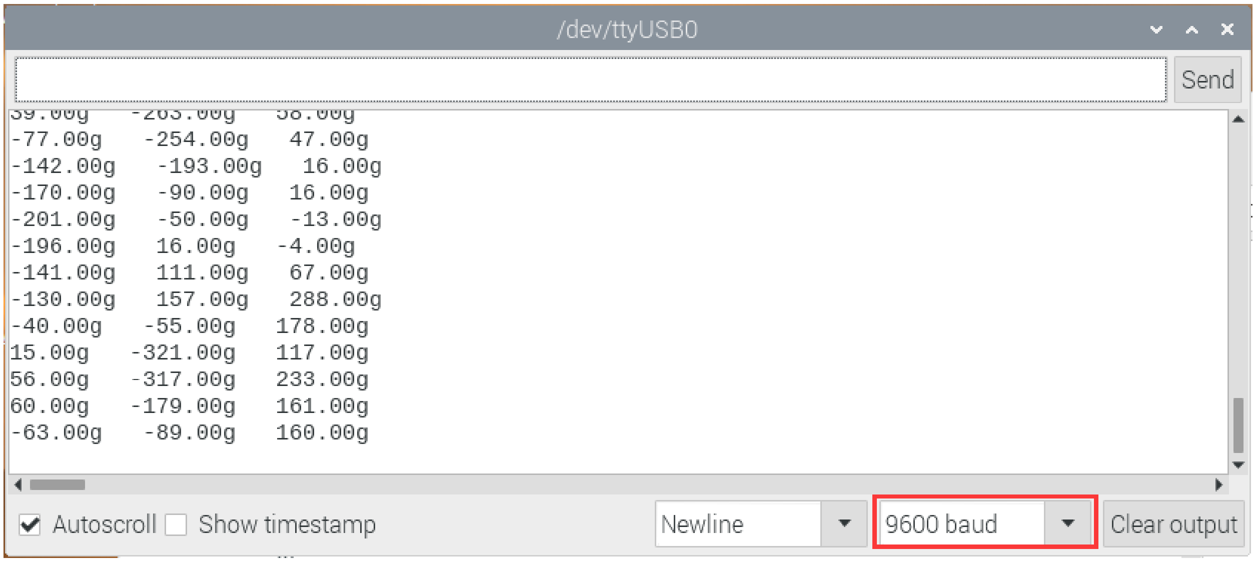
Press a key again, the time of pressing buttons is 3 and the remainder is 3. Read digital values at x, y and z axis of the joystick module. As shown below;

Press the key for the fourth time, the remainder is 4. Then the potentiometer can adjust the PWM value at the GPI05 port to control LED brightness of the purple LED.

Press the key for the fifth time, the remainder is 5. Then the ultrasonic sensor can detect obstacles, as shown below;

Press the key for the sixth time, the remainder is 6. Then the ultrasonic sensor can detect distance away from obstacles, as shown below;

Press the key for seventh time and the remainder is 7. The monitor will print out the acceleration values.



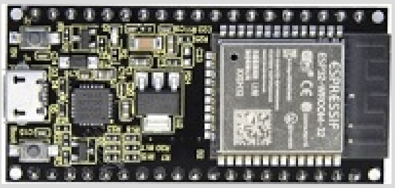
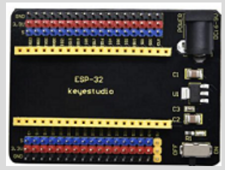
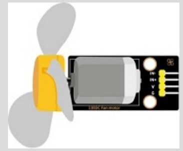
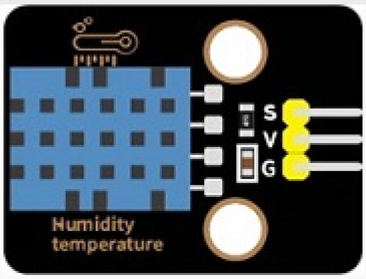
Press the key for eighth time and the remainder is 0. Then the RGB will flash. If you press keys incessantly, remainders will change in a loop way. So does functions.

7.6.21 Project 65: WiFi

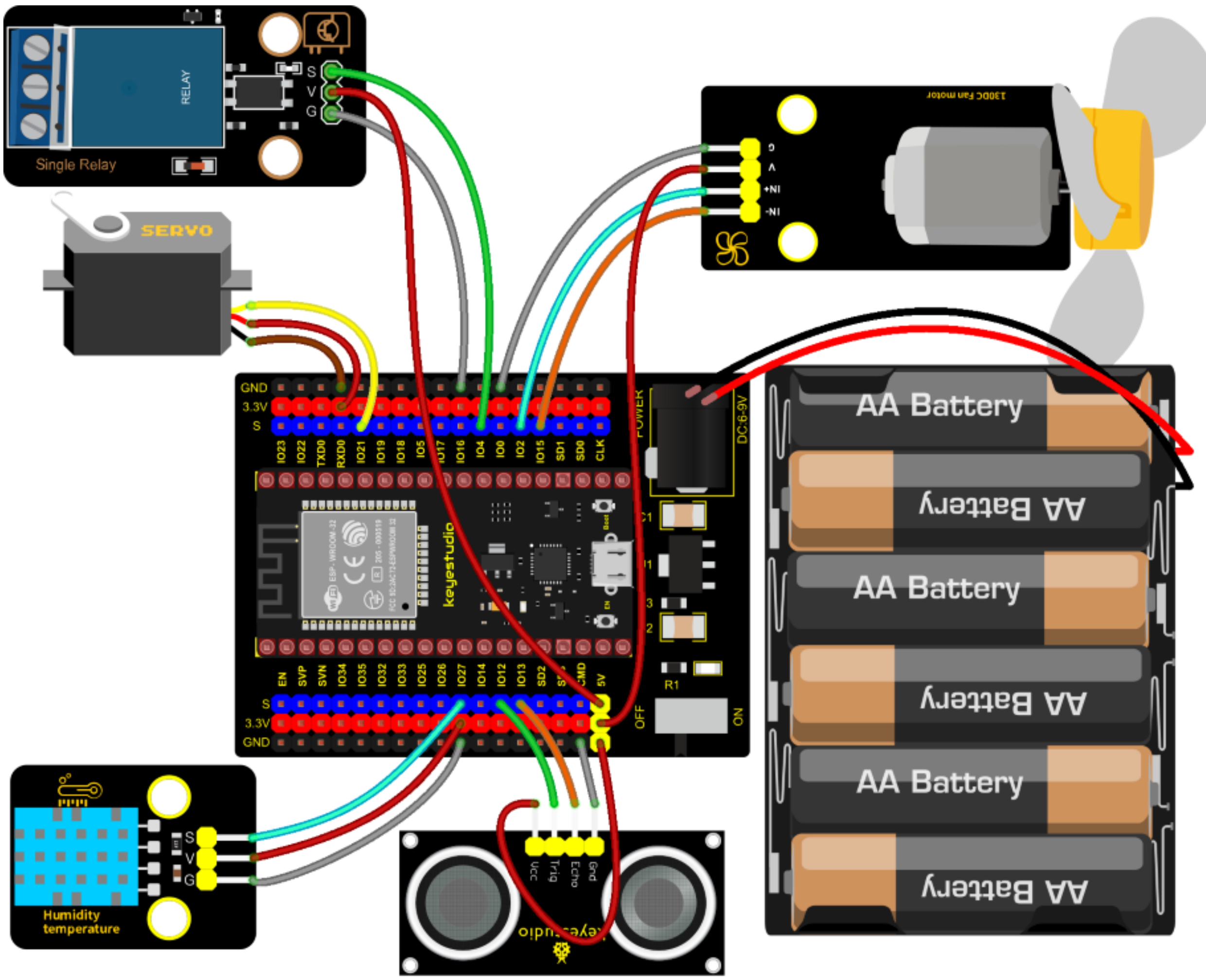
Description

In the previous experiment, we have learned the WiFi Station mode, WiFi AP mode and WiFi AP+Station mode of the ESP32. In this project, We will use ESP32's WiFi Station mode to control the work of multiple sensors/modules through APP connection with WiFi to achieve the effect of WiFi smart home.

Components

				
ESP32 Board*1	ESP32 Expansion Board*1	Keyestudio 130 Motor*1	Keyestudio 5V Relay Module*1	Servo*1
				
Keyestudio XHT11 Temperature and Humidity Sensor*1compatible DHT11)	Keyestudio HC-SR04 Ultrasonic Sensor*1	3P Dupont*2	4P Dupont*2	Smart Phone/PC*1
				
Battery Holder*1	Battery (provide for yourself)*6	Micro USB Cable*1		

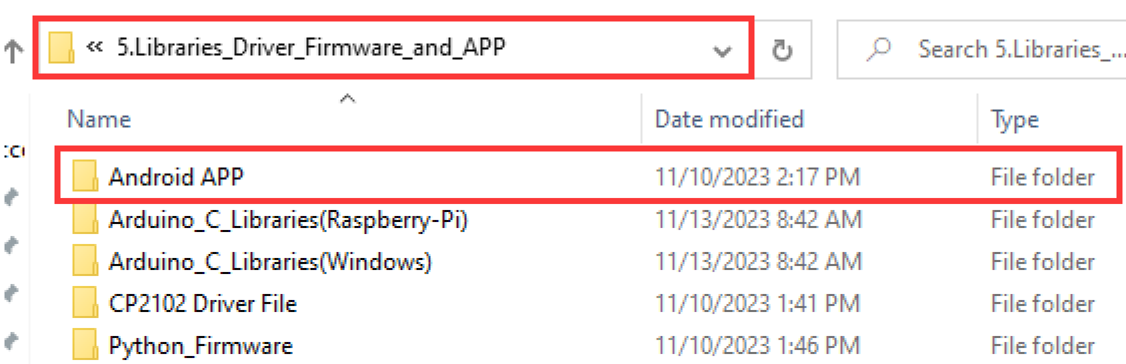
Wiring Diagram



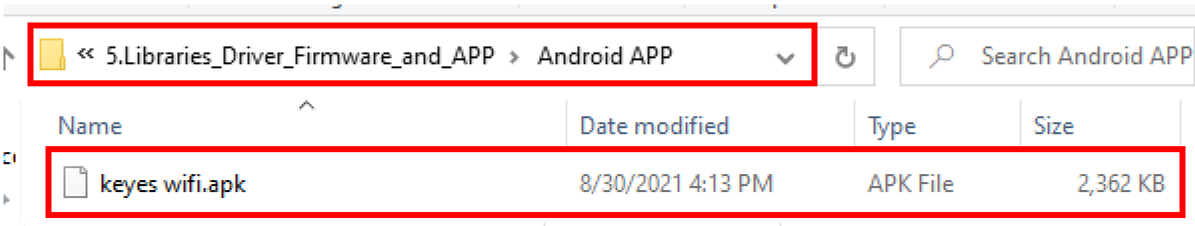
Install APP

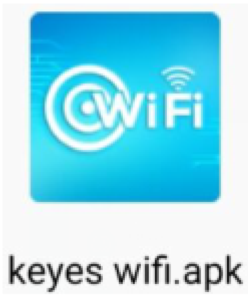
(1) Android device (mobile phone/PC) APP:

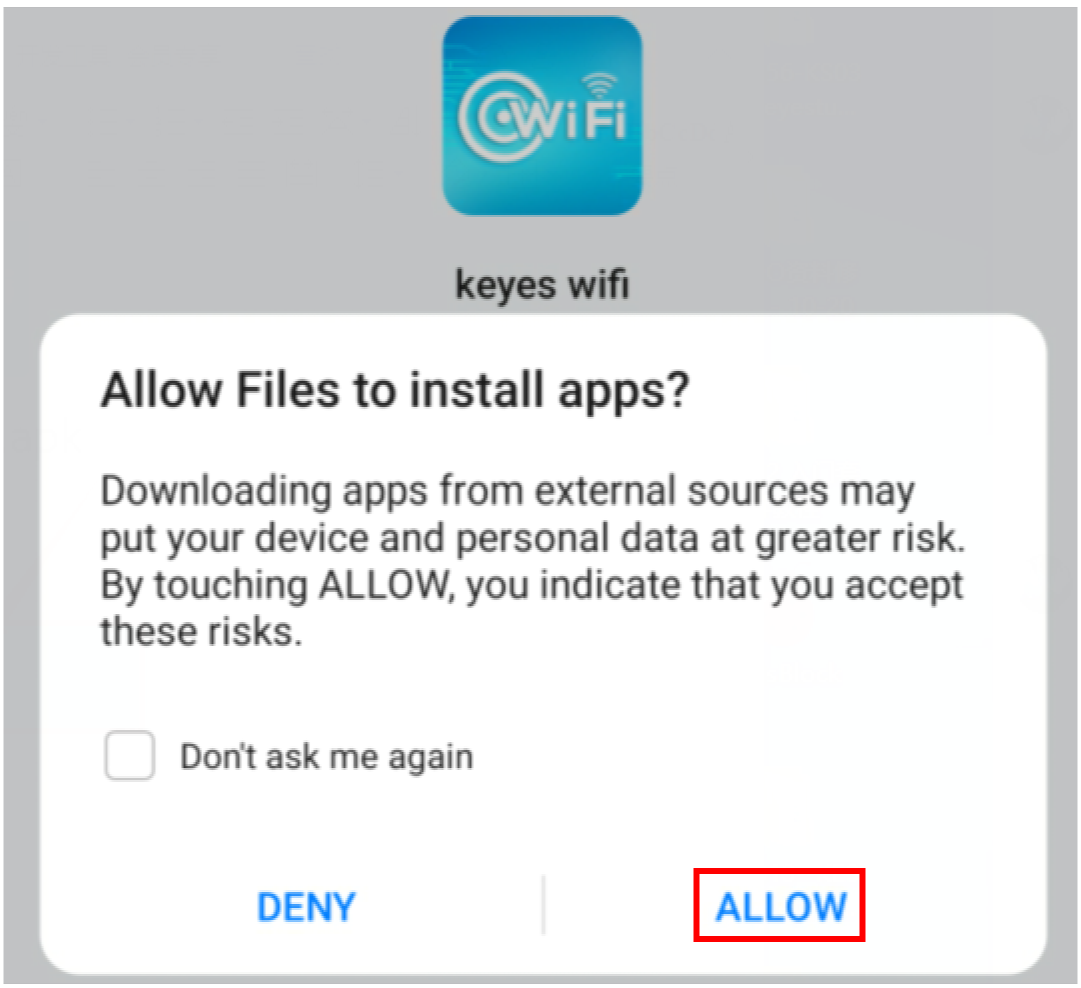
A. We provide the Android APP installation package.

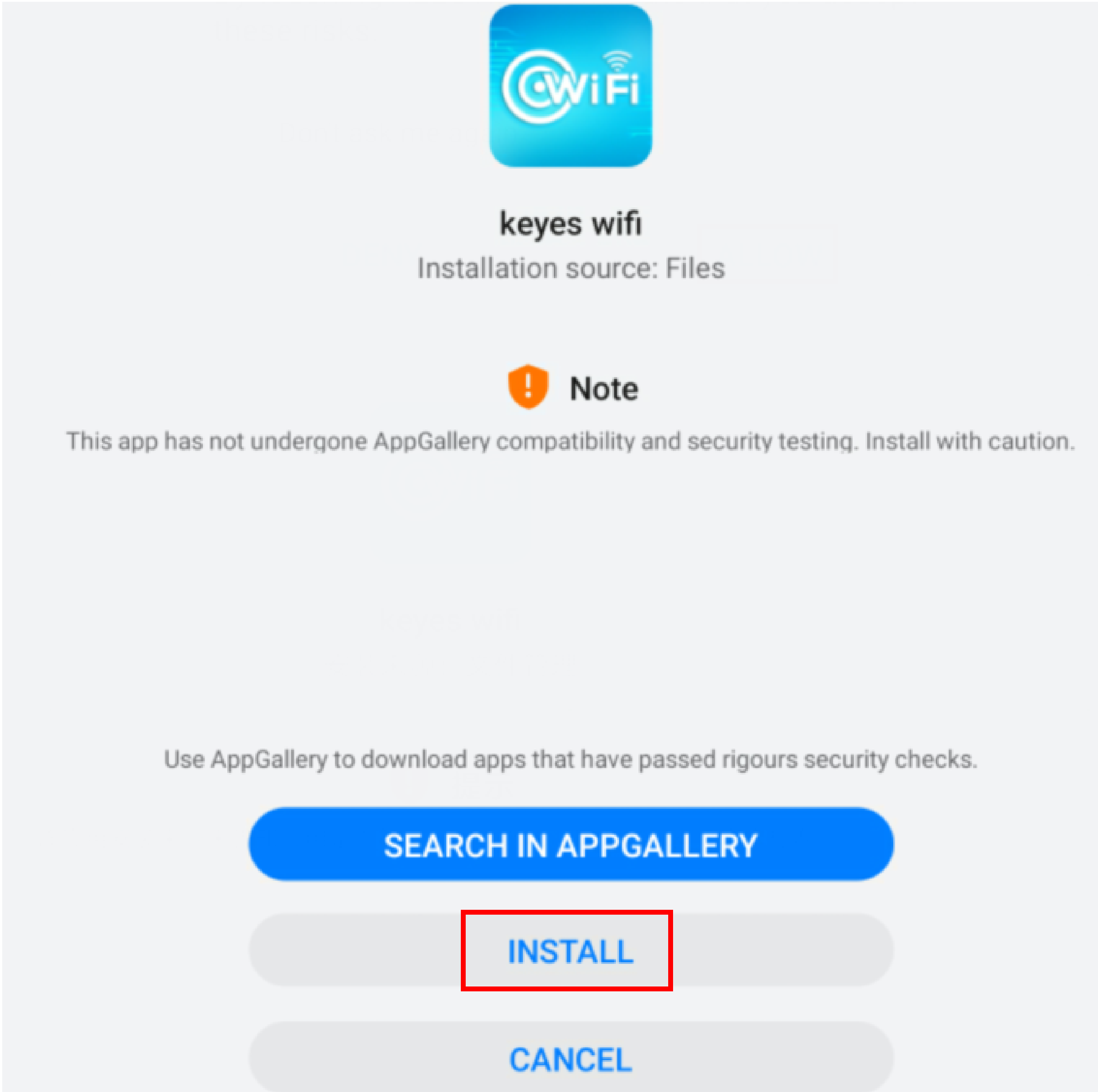


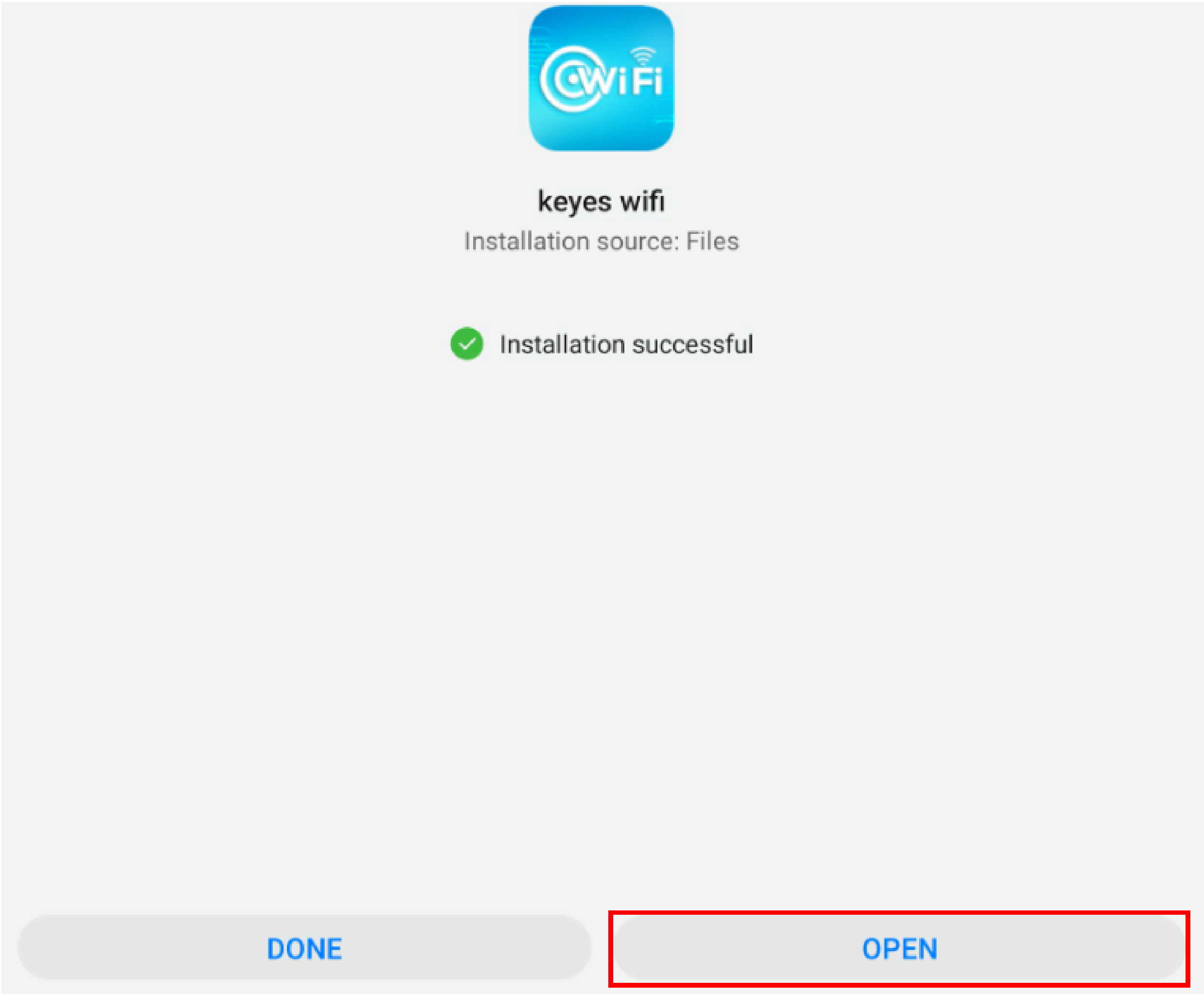
B. Now transfer the keyes wifi.apk file in the Android APP installation package to the Android phone or PC, click the keyes wifi.apk file to enter the installation page, click “ALLOW” key, and then click “INSTALL” button. After installation, click “OPEN” button to enter the APP interface.

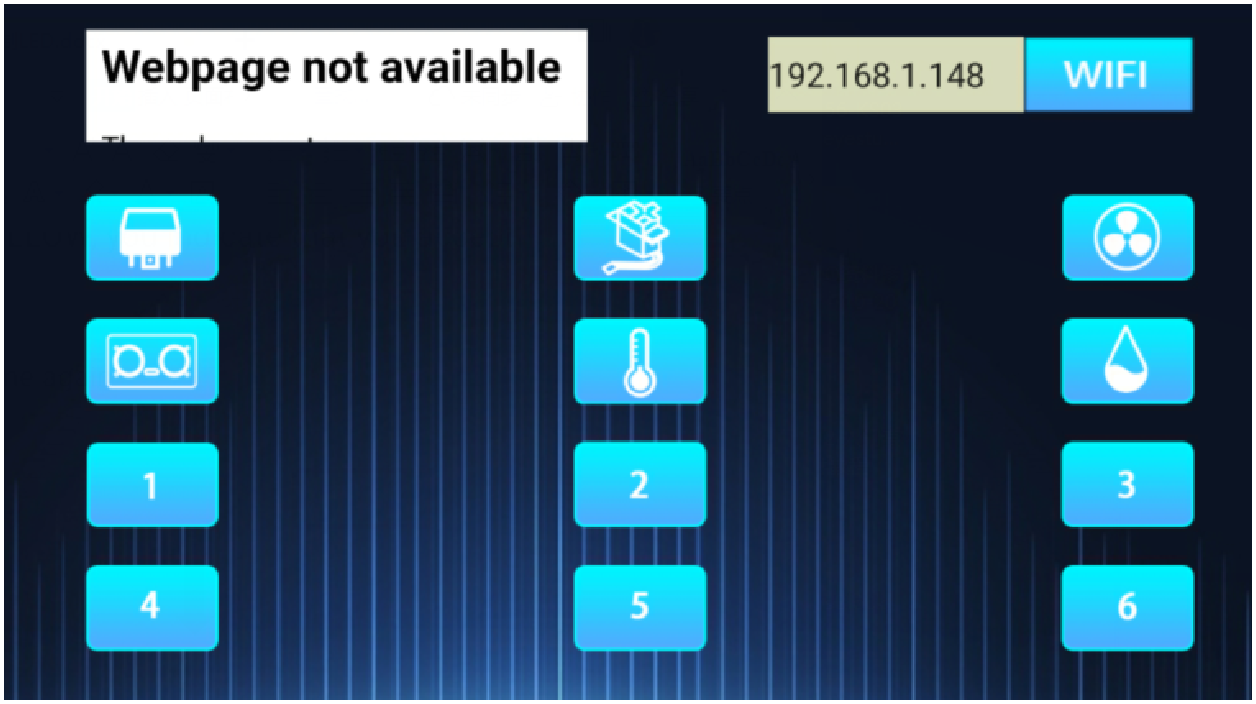

5.Libraries_Driver_Firmware_and_APP > Android APP
Search Android APP
Name
Date modified
Type
Size
keyes wifi.apk
8/30/2021 4:13 PM
APK File
2,362 KB


WiFi
keyes wifi.apk


WiFi
keyes wifi
Allow Files to install apps?
Downloading apps from external sources may put your device and personal data at greater risk. By touching ALLOW, you indicate that you accept these risks.
Don't ask me again
DENY
ALLOW


WiFi
keyes wifi
Installation source: Files
Note
This app has not undergone AppGallery compatibility and security testing. Install with caution.
Use AppGallery to download apps that have passed rigours security checks.
SEARCH IN APPGALLERY
INSTALL
CANCEL


WiFi
keyes wifi
Installation source: Files
Installation successful
DONE
OPEN



(2) IOS device (mobile phone /iPad) APP:

A. Open App Store 

B. Enter keyes link in the search box and click search, the download interface appears. Click "" to download and install the APP of the keyes link. The following operations are similar to those of Android system. You can refer to the steps of Android system above for operation.

Test Code

```
//**********************************************************************************
/*
 * Filename    : WiFi Smart Home.
 * Description : WiFi APP controls Multiple sensors/modules work to achieve the effect of WiFi smart home.
 * Auther      : http//www.keyestudio.com
*/
#include <Arduino.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiClient.h>

#include "xht11.h"
//gpio15
xht11 xht(27);
unsigned char dht[4] = {0, 0, 0, 0};
```

(continues on next page)

(continued from previous page)

```
#include <ESP32Servo.h>
Servo myservo;
int servoPin = 21;
#define Relay  4
#define IN1 2  //IN1 corresponds to IN+
#define IN2 15 //IN2 corresponds to IN-
#define trigPin  12
#define echoPin  13

int distance1;
String dis_str;
int ip_flag = 1;
int ultra_state = 1;
int temp_state = 1;
int humidity_state = 1;

String item = "0";
const char* ssid = "ChinaNet-2.4G-0DF0"; //the name of user's wifi
const char* password = "ChinaNet@233";   //the password of user's wifi
WiFiServer server(80);
String unoData = "";

void setup() {
  Serial.begin(115200);
  pinMode(Relay, OUTPUT);
  myservo.setPeriodHertz(50);
  myservo.attach(servoPin, 500, 2500);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
  Serial.println("TCP server started");
  MDNS.addService("http", "tcp", 80);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(Relay, LOW);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
```

(continues on next page)

(continued from previous page)

```
void loop() {
  WiFiClient client = server.available();
  if (!client) {
      return;
  }
  while(client.connected() && !client.available()){
      delay(1);
  }
  String req = client.readStringUntil('\r');
  int addr_start = req.indexOf(' ');
  int addr_end = req.indexOf(' ', addr_start + 1);
  if (addr_start == -1 || addr_end == -1) {
      Serial.print("Invalid request: ");
      Serial.println(req);
      return;
  }
  req = req.substring(addr_start + 1, addr_end);
  item=req;
  Serial.println(item);
  String s;
  if (req == "/")
  {
      IPAddress ip = WiFi.localIP();
      String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' +␣
↪String(ip[3]);
      s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>
↪Hello from ESP32 at ";
      s += ipStr;
      s += "</html>\r\n\r\n";
      Serial.println("Sending 200");
      client.println(s);
  }
  else if(req == "/btn/0")
  {
    Serial.write('a');
    client.println(F("turn on the relay"));
    digitalWrite(Relay, HIGH);
  }
  else if(req == "/btn/1")
  {
    Serial.write('b');
    client.println(F("turn off the relay"));
    digitalWrite(Relay, LOW);
  }
  else if(req == "/btn/2")
  {
    Serial.write('c');
    client.println("Bring the steering gear over 180 degrees");
    myservo.write(180);
    delay(200);
  }
  else if(req == "/btn/3")
```

(continues on next page)

(continued from previous page)

```
  {
    Serial.write('d');
    client.println("Bring the steering gear over 0 degrees");
    myservo.write(0);
    delay(200);
  }
  else if(req == "/btn/4")
  {
    Serial.write('e');
    client.println("esp32 already turn on the fans");
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
  }
  else if(req == "/btn/5")
  {
    Serial.write('f');
    client.println("esp32 already turn off the fans");
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
  }
  else if(req == "/btn/6")
  {
    Serial.write('g');
    while(Serial.available() > 0)
    {
      unoData = Serial.readStringUntil('#');
      client.println("Data");
    }
    while(ultra_state>0)
       {
          Serial.print("Distance = ");
          Serial.print(checkdistance());
          Serial.println("#");
          Serial1.print("Distance = ");
          Serial1.print(checkdistance());
          Serial1.println("#");
          int t_val1 = checkdistance();
          client.print("Distance(cm) = ");
          client.println(t_val1);
          ultra_state = 0;
        }
  }
  else if(req == "/btn/7")
  {
    Serial.write('h');
    client.println("turn off the ultrasonic");
    ultra_state = 1;
  }
  else if(req == "/btn/8")
  {
    Serial.write('i');
    while(Serial.available() > 0)
```

(continues on next page)

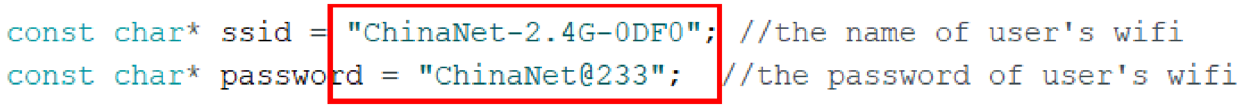
(continued from previous page)

```
    {
     unoData = Serial.readStringUntil('#');
     client.println(unoData);
    }
   while(temp_state>0)
     {
       if (xht.receive(dht)) {
         Serial.print("Temperature = ");
         Serial.print(dht[2],1);
         Serial.println("#");
         Serial1.print("Temperature = ");
         Serial1.print(dht[2],1);
         Serial1.println("#");
         int t_val2 = dht[2];
         client.print("Temperature(℃) = ");
         client.println(t_val2);
       }
       temp_state = 0;
     }
 }
 else if(req == "/btn/9")
 {
   Serial.write('j');
   client.println("turn off the temperature");
   temp_state = 1;
 }
 else if(req == "/btn/10")
 {
   Serial.write('k');
   while(Serial.available() > 0)
    {
      unoData = Serial.readStringUntil('#');
      client.println(unoData);
    }
   while(humidity_state > 0)
     {
       if (xht.receive(dht)) {
         Serial.print("Humidity = ");
         Serial.print(dht[0],1);
         Serial.println("#");
         Serial1.print("Humidity = ");
         Serial1.print(dht[0],1);
         Serial1.println("#");
         int t_val3 = dht[0];
         client.print("Humidity(%) = ");
         client.println(t_val3);
       }
       humidity_state = 0;
     }
 }
 else if(req == "/btn/11")
 {
```

(continues on next page)

(continued from previous page)

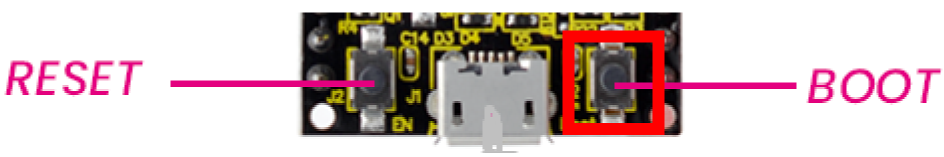
```
    Serial.write('l');
    client.println("turn off the humidity");
    humidity_state = 1;
    }
  //client.print(s);
  client.stop();
}

int checkdistance() {
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(10);
  digitalWrite(12, LOW);
  int distance = pulseIn(13, HIGH) / 58;

  delay(10);
  return distance;
}
//*********************************************************************************
```

Note: You need to 

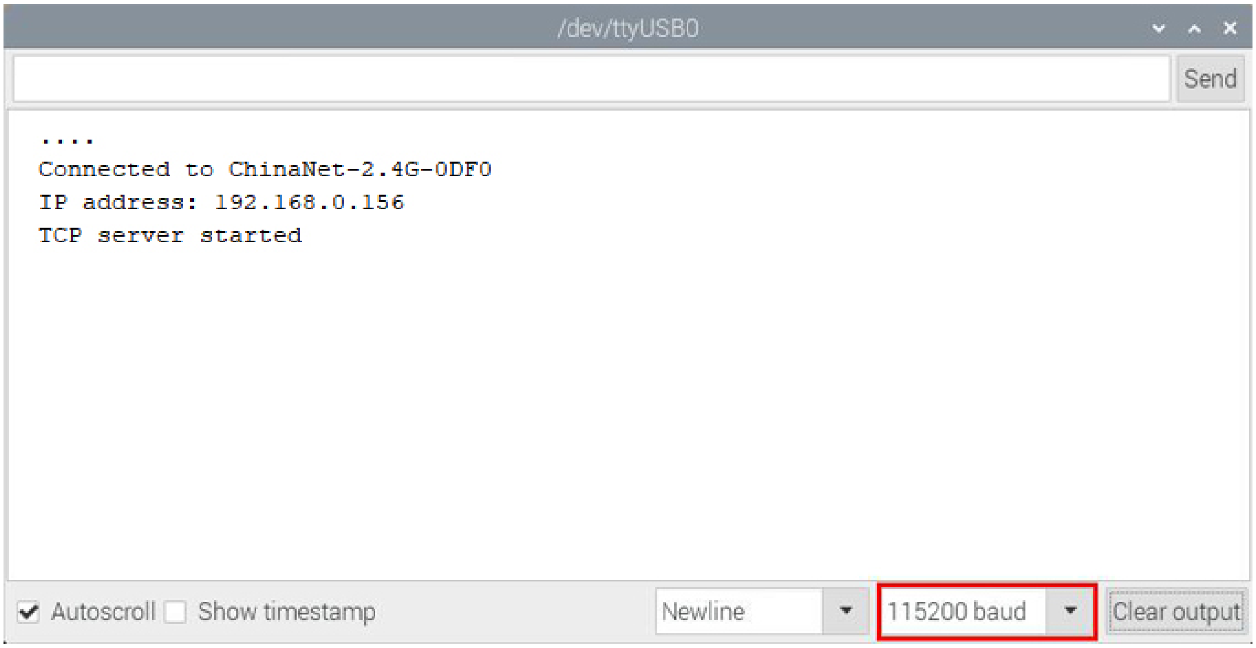
change the Wifi name and default Wifi password of the experimental code to your own Wifi name and Wifi password.

Test Result

After the code has been modified correctly, connect the external power supply and power on. Switch the DIP switch ON the ESP32 expansion board to the ON end, compile and upload the code to the ESP32 mainboard.If uploading the code is not successful, press the Boot button on the ESP32 mainboard with your hand after click , release it when the upload progress percentage appears.)

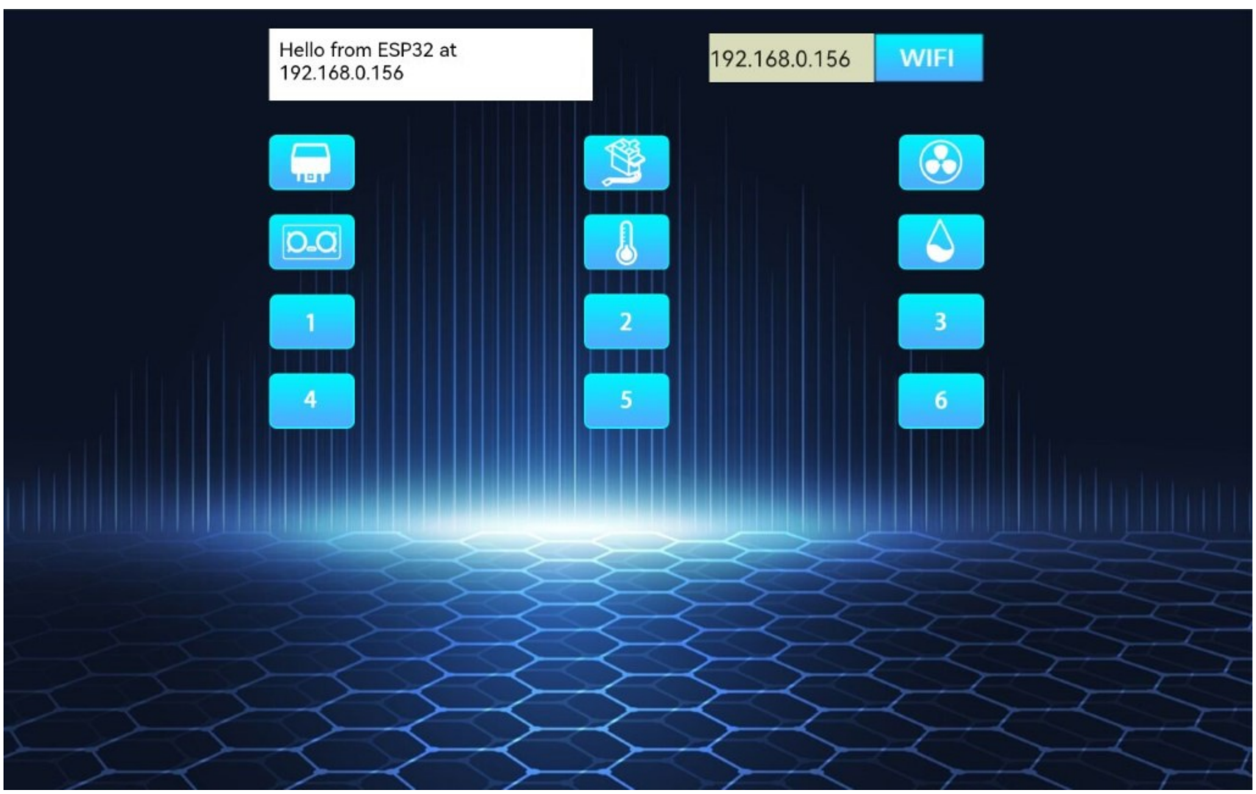


Open the serial monitor and set baud rate to 115200, then the monitor prints the detected WiFi IP address. (If open the serial monitor and set the baud rate to 115200, the information is not displayed, please press the button RESET of the ESP32)

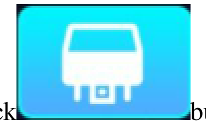
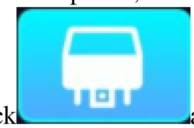


Open WiFi APP, enter the detected WIFI IP address in the text box in front of the WIFI button (for example, the IP address detected by the serial monitor above is 192.168.0.156).

Next, click the WIFI button to connect to WIFI, at the same time, the corresponding WiFi IP address will be displayed in the text box :"Hello from ESP32 at 192.168.0.156", then the APP has connected to WiFi. (WiFi IP address sometimes changes, if the original IP address can not use, you need to re-check it.)

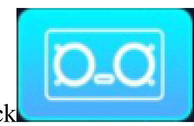
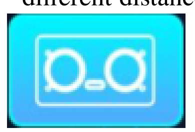


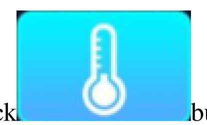
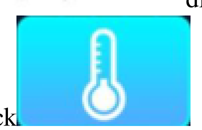
After the APP is connected to WiFi, the following operations are performed:

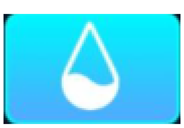
1). Click  button, the relay will be opened, the APP will display `turn on the relay` and the indicator lights up on the module. Click  again, the relay will be closed , the APP will display `turn off the relay` and the indicator on the module is off.

2). Click  buttonthe servo rotates 180°the APP will display  againthe APP will display `Bring the steering gear over 0 degrees` the servo rotates 0°.

3). Click  buttonthe motorwith small fan bladesrotatesthe APP will display  againclose the motorthe APP will display `esp32 already turn off the fans`

4). Click  buttonthe ultrasonic sensor detects the distance, put an object in front of the ultrasonic sensor, the APP will display `Distance(cm) = 6` different distances show different numbers, the distance between the object and the ultrasonic sensor is 14cmclick  again, turn off the sensor, the APP will display `turn off the ultrasonic` .

5). Click  buttonthe temperature and humidity sensor measures the temperature in the environment, the APP will display `Temperature(℃) = 30` different temperatures show different temperature valuesthe ambient temperature is 28 ° C., click  again, turn off the sensorthe APP will display `turn off the temperature` .

6). Click  buttonthe temperature and humidity sensor measures the humidity in the environment,the APP will display `Humidity(%) = 55` different humiditys show different humiditys values, the ambient humidity is 52%clickagainturn off the sensor, the APP will display turn off the humidity.