

Welcome

Thank you for choosing Freenove products!

Getting Started

When reading this, you should have downloaded the ZIP file for this product.

Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

- ! Unzip the ZIP file instead of opening the file in the ZIP file directly.
- ! Do not move, delete or rename files in the folder just unzipped.

Get Support

Encounter problems? Don't worry! Refer to "TroubleShooting.pdf" or contact us.

When there are packaging damage, quality problems, questions encountering in use, etc., just send us an email. We will reply to you within one working day and provide a solution.

support@freenove.com

Safety and Precautions

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

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

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

About Freenove

Freenove provides open source electronic products and services worldwide.

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

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

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

<http://www.freenove.com>

Copyright

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



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

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



Other registered trademarks and their owners appearing in this document:

Arduino® is a trademark of Arduino LLC (<https://www.arduino.cc/>).

Raspberry Pi® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

Raspberry Pi Pico® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

micro:bit® is a trademark of Micro:bit Educational Foundation (<https://www.microbit.org/>).

ESPRESSIF® and ESP32® are trademarks of ESPRESSIF Systems (Shanghai) Co, Ltd (<https://www.espressif.com/>).

Any concerns? ✉ support@freenove.com

Contents

Welcome.....	1
Contents	1
Preface.....	5
Raspberry Pi Pico.....	6
Raspberry Pi Pico W	9
Raspberry Pi Pico 2.....	12
Chapter 0 Getting Ready (Important).....	15
Programming Software.....	15
Installation of Development Board Support Package	18
Uploading Arduino-compatible Firmware for Pico.....	20
Paste the Sticker on the Breadboard.....	24
Chapter 1 LED (Important).....	25
Project 1.1 Blink	25
Project 1.2 Blink	31
Chapter 2 Button & LED	36
Project 2.1 Button & LED.....	37
Project 2.2 MINI table lamp.....	41
Chapter 3 LED Bar	45
Project 3.1 Flowing Light	45
Chapter 4 Analog & PWM	50
Project 4.1 Breathing LED.....	50
Project 4.2 Meteor Flowing Light	56
Chapter 5 RGBLED	61
Project 5.1 Random Color Light.....	61
Project 5.2 Gradient Color Light.....	66
Chapter 6 NeoPixel	68
Project 6.1 NeoPixel	68
Project 6.2 Rainbow Light.....	75
Chapter 7 Buzzer	78



Project 7.1 Doorbell.....	78
Project 7.2 Alertor	84
Chapter 8 Serial Communication.....	89
Project 8.1 Serial Print.....	89
Project 8.2 Serial Read and Write	93
Chapter 9 AD Converter.....	96
Project 9.1 Read the Voltage of Potentiometer.....	96
Chapter 10 Potentiometer & LED	102
Project 10.1 Soft Light	102
Project 10.2 Soft Colorful Light	106
Project 10.3 Soft Rainbow Light.....	110
Chapter 11 Photoresistor & LED	114
Project 11.1 Control LED through Photoresistor.....	114
Chapter 12 Thermistor	119
Project 12.1 Thermometer	119
Chapter 13 Joystick	124
Project 13.1 Joystick	124
Chapter 14 74HC595 & LED Bar Graph	129
Project 14.1 Flowing Water Light.....	129
Chapter 15 74HC595 & 7-Segment Display.....	135
Project 15.1 7-Segment Display.....	135
Project 15.2 4-Digit 7-Segment Display.....	142
Chapter 16 74HC595 & LED Matrix	149
Project 16.1 LED Matrix.....	149
Chapter 17 Relay & Motor.....	158
Project 17.1 Relay & Motor	158
Chapter 18 L293D & Motor.....	167
Project 18.1 Control Motor with Potentiometer.....	167
Chapter 19 Servo	174
Project 19.1 Servo Sweep.....	174
Project 19.2 Servo Knob	180

Chapter 20 Stepper Motor.....	183
Project 20.1 Stepper Motor	183
Chapter 21 LCD1602	192
Project 21.1 LCD1602	192
Chapter 22 Ultrasonic Ranging	200
Project 22.1 Ultrasonic Ranging.....	200
Project 22.2 Ultrasonic Ranging.....	206
Chapter 23 Matrix Keypad	210
Project 23.1 Matrix Keypad.....	210
Project 23.2 Keypad Door	217
Chapter 24 Infrared Remote	222
Project 24.1 Infrared Remote Control.....	222
Project 24.2 Control LED through Infrared Remote	230
Chapter 25 Hygrothermograph DHT11.....	235
Project 25.1 Hygrothermograph.....	235
Project 25.2 Hygrothermograph.....	242
Chapter 26 Infrared Motion Sensor	247
Project 26.1 Infrared Motion Detector with LED Indicator.....	247
Chapter 27 Attitude Sensor MPU6050.....	252
Project 27.1 Read an MPU6050 Sensor Module	252
Chapter 28 RFID	260
Project 28.1 RFID read UID	260
Project 28.2 Read and write.....	269
Chapter 29 Play Music.....	273
Project 29.1 Play Music by Audio Converter & Amplifier.....	273
Chapter 30 WiFi Working Modes (Only for Pico W).....	288
Project 30.1 Station mode.....	288
Project 30.2 AP mode.....	293
Project 30.3 AP+Station mode	298
Chapter 31 TCP/IP (Only for Pico W).....	302
Project 31.1 as Client	302
Project 31.2 as Server	316



Chapter 32 Control LED with Web (Only for Pico W).....	322
Project 32.1 Control the LED with Web.....	322
Chapter 33 Bluetooth (Only for Pico W).....	330
Project 33.1 Bluetooth Passthrough	330
Project 33.2 Bluetooth Low Energy Data Passthrough.....	337
Project 33.3 Bluetooth Control LED	351
Project 33.4 Bluetooth Low Energy Control LED	357
Chapter 34 Bluetooth Audio (Only for Pico W).....	365
Project 33.1 Bluetooth Passthrough	365
What's Next?	373

Preface

Raspberry Pi Pico is a tiny, fast, and versatile board built using RP2040, a brand new microcontroller chip designed by Raspberry Pi in the UK. Supporting Python and C/C++ development, it is perfect for DIY projects. In this tutorial, we use Arduino to learn Pico. If you want to learn the Python version, please refer to another tutorial: [python_tutorial.pdf](#).

Using Arduino IDE as the development environment for Raspberry Pi Pico allows users to learn Pico better and more quickly, which is just like developing Arduino programs. In addition, resources such as Arduino's libraries can be directly used to greatly improve the efficiency of development.

If you have not downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:

https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico

In this tutorial, we divide each project into 4 sections:

- 1, Component list: helps users to learn and find what components are needed in each project.
- 2, Component Knowledge: allows you to learn the features and usage of the components.
- 3, Circuit: assists to build circuit for each project.
- 4, Sketches and comments: makes it easier for users to learn to use Raspberry Pi Pico and make secondary development.

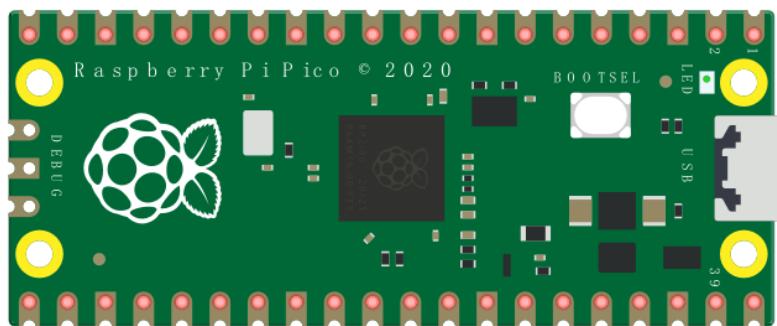
After completing the projects in this tutorial, you can also combine the components in different projects to make your own smart homes, smart car, robot, etc., bringing your imagination and creativity to life with Raspberry Pi Pico.

If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

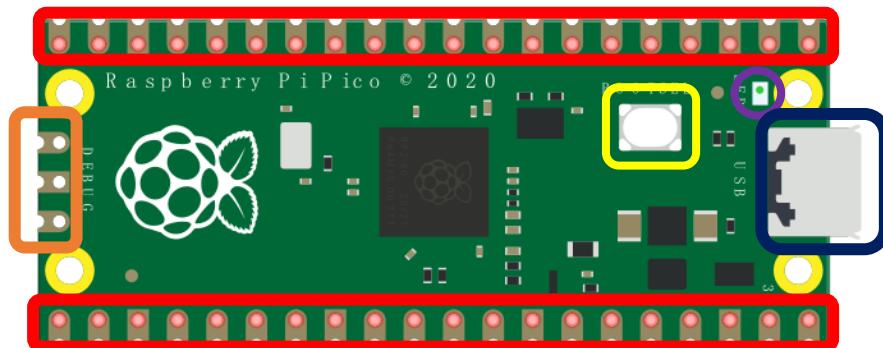
Raspberry Pi Pico

Raspberry Pi Pico applies to all chapters except Wireless in this tutorial.

Before learning Pico, we need to know about it. Below is an imitated diagram of Pico, which looks very similar to the actual Pico.

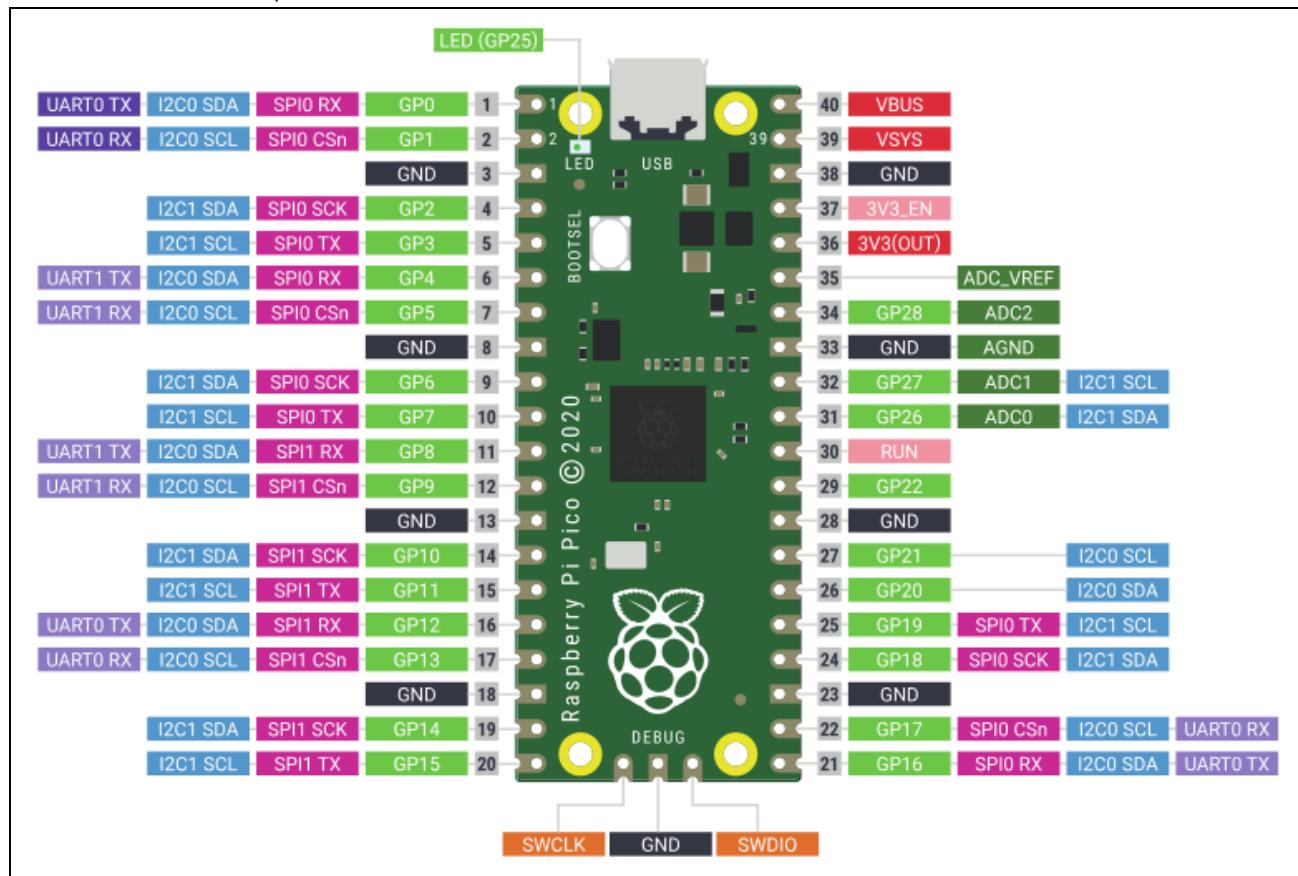


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: <https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf>

UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

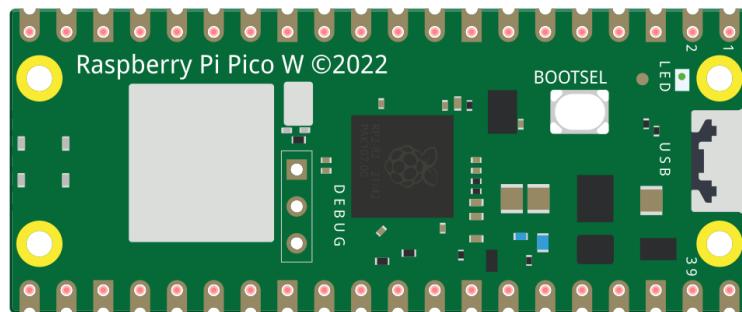
SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

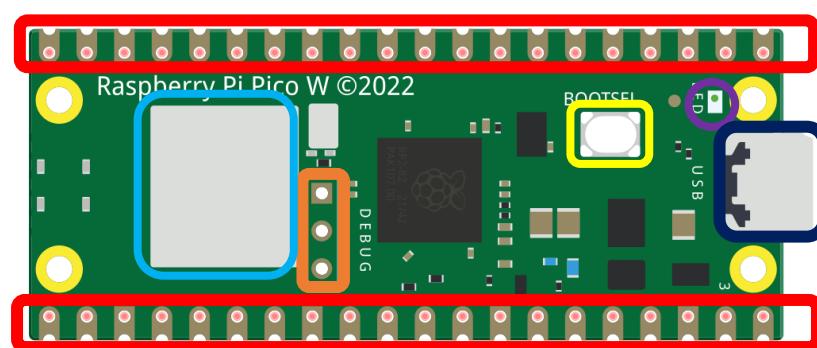
Raspberry Pi Pico W

Raspberry Pi Pico W applies to all chapters in this tutorial.

Raspberry Pi Pico W adds CYW43439 as the WiFi function based on Raspberry Pi Pico. It is connected to RP2040 chip through SPI interface.

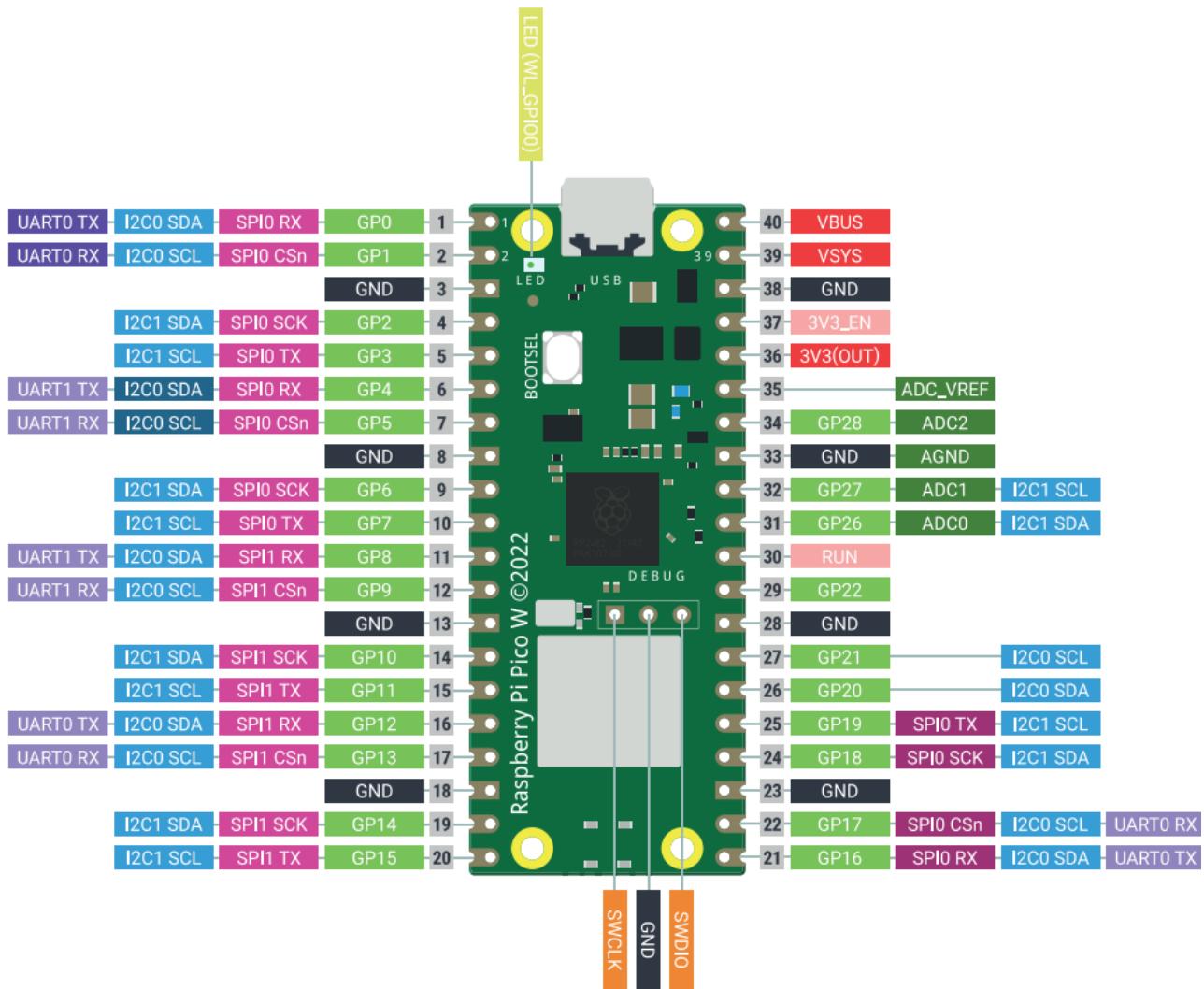


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging
	Wireless

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Purple	UART(default)	Lavender	UART
Magenta	SPI	Cyan	I2C
Pink	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>

UART, I2C, SPI, Wireless Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

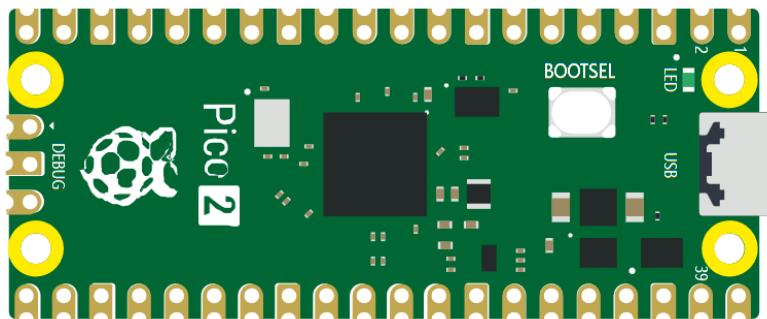
Wireless

Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

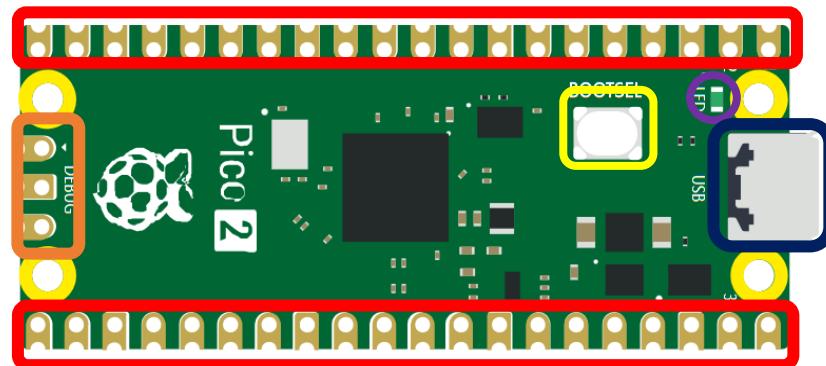
Raspberry Pi Pico 2

Raspberry Pi Pico 2 is applicable to all chapters in this tutorial except RFID and those involving WiFi.

Raspberry Pi Pico 2 uses RP2350 chip as the main controller, which equipped with dual Cortex-M33 or Hazard3 processors, capable of running up to 150 MHz, providing a significant boost in processing power, compared with the original Pico. It also doubles the memory with 520KB of SRAM and 4MB of onboard flash memory, with the ADC sampling frequency increasing to up to 500ksps. In addition, it adds 8 more PWM channels, and features additional interfaces like 2× Timer with 4 alarms, 1× AON Timer and 4 x PIO.

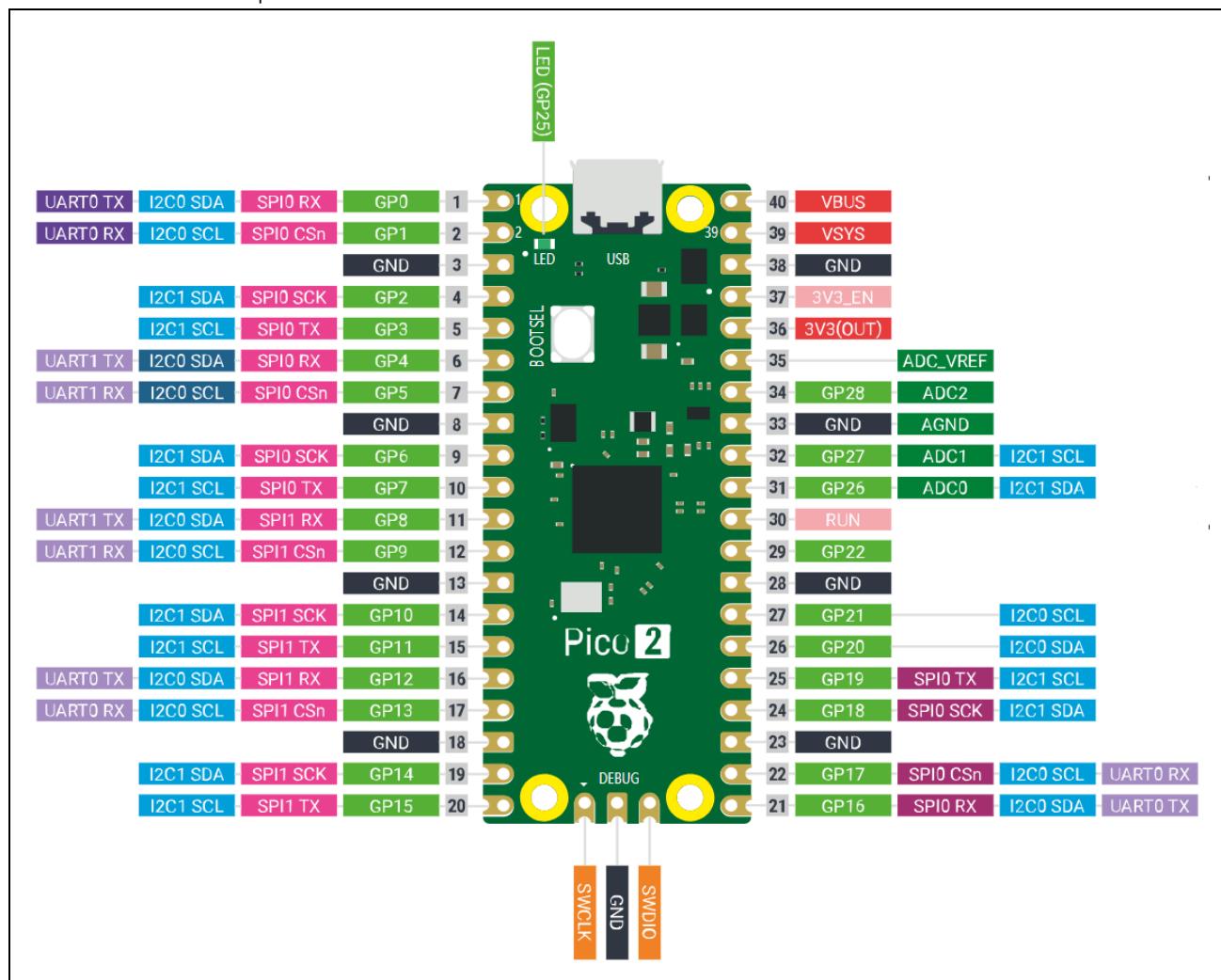


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: <https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf>

UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2350. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

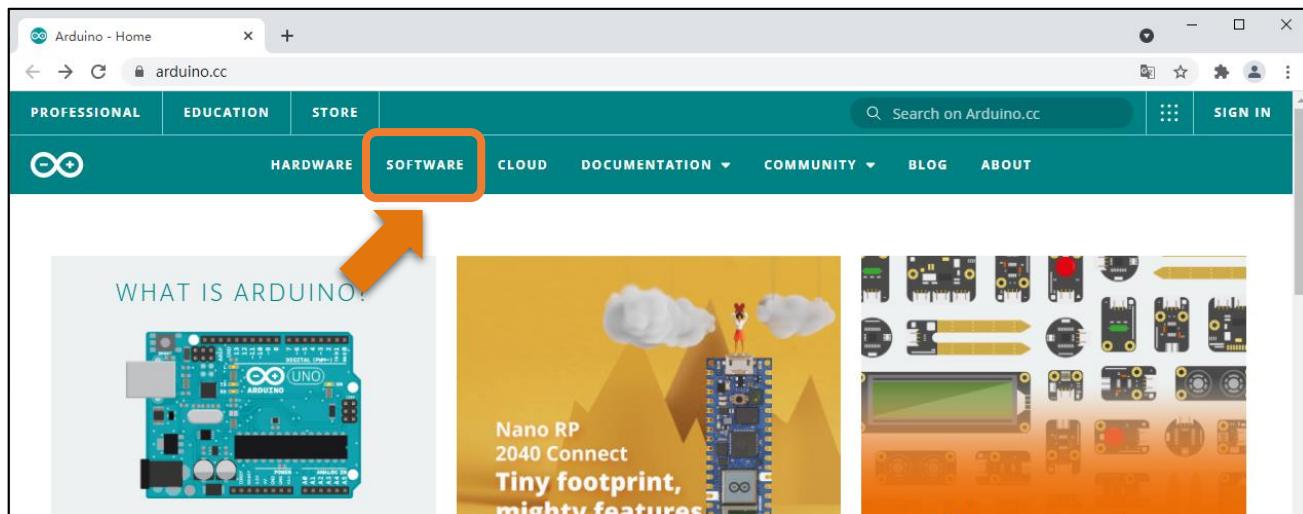
Chapter 0 Getting Ready (Important)

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

Programming Software

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

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



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

Downloads

The screenshot shows the download page for Arduino IDE 2.3.2. On the left, there's a thumbnail of the Arduino IDE icon, the title "Arduino IDE 2.3.2", and a brief description: "The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocomplete, code navigation, and even a live debugger." Below this, there's a link to "Arduino IDE 2.0 documentation" and a note about nightly builds. On the right, there's a large section titled "DOWNLOAD OPTIONS" with links for Windows (Win 10 and newer, 64 bits, MSI installer, ZIP file), Linux (AppImage 64 bits (X86-64), ZIP file 64 bits (X86-64)), and macOS (Intel, 10.15: "Catalina" or newer, 64 bits, Apple Silicon, 11: "Big Sur" or newer, 64 bits). A link to "Release Notes" is also present.

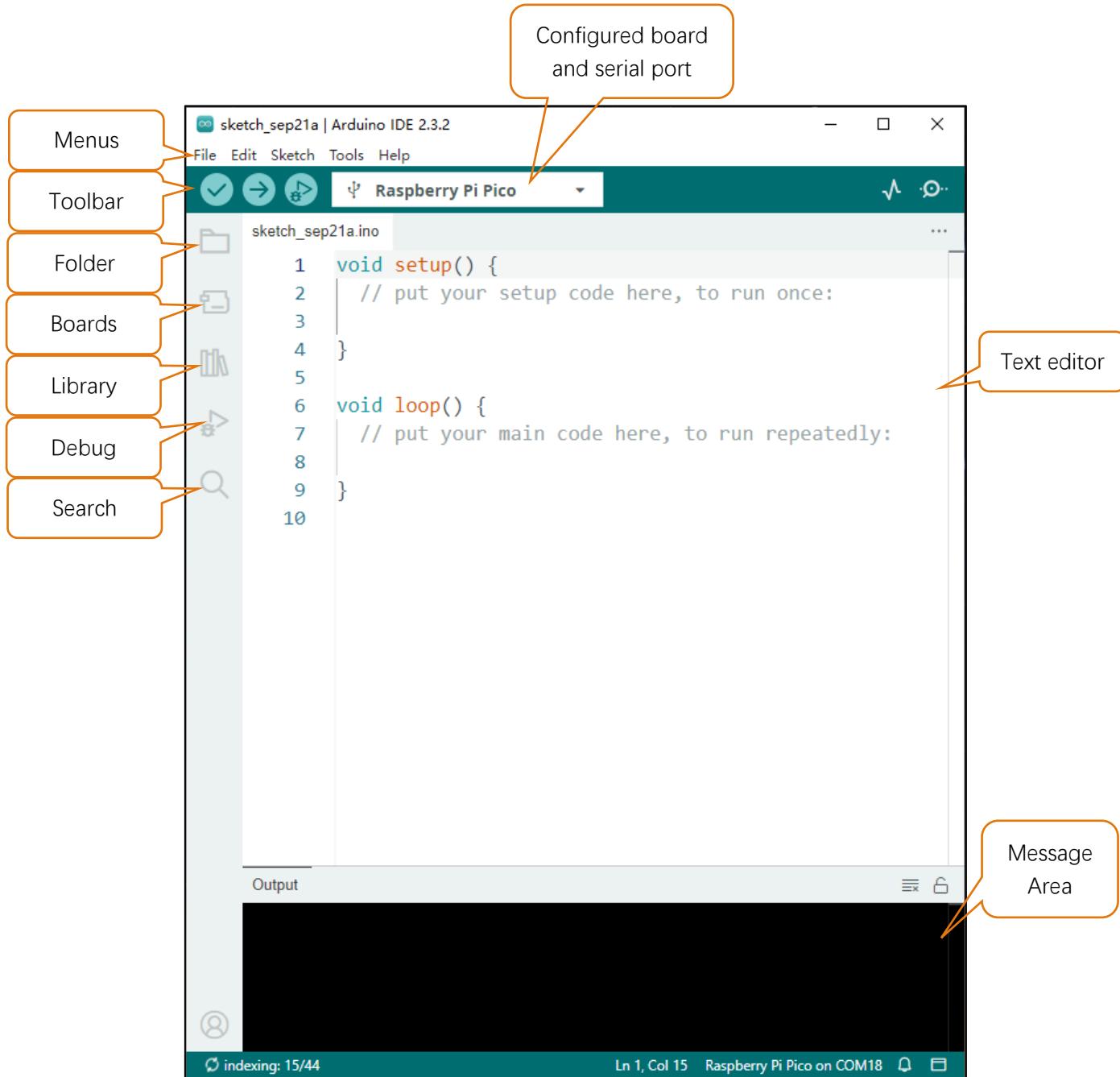


After the download completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it comes up, please allow the installation.

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



The interface of Arduino Software is as follows:



Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and saved with the file extension.**.ino**. The editor has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify, upload, and debug programs, select board & port, and open serial plotter and serial monitor.



Verify

Check your code for compile errors.



Upload

Compile your code and upload them to the configured board.



Debug

Test and debug programs in real time.



Select Board & Port

Detected Arduino boards automatically show up here, along with the port number.



Serial Plotter

Open the serial plotter.



Serial Monitor

Open the serial monitor.

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

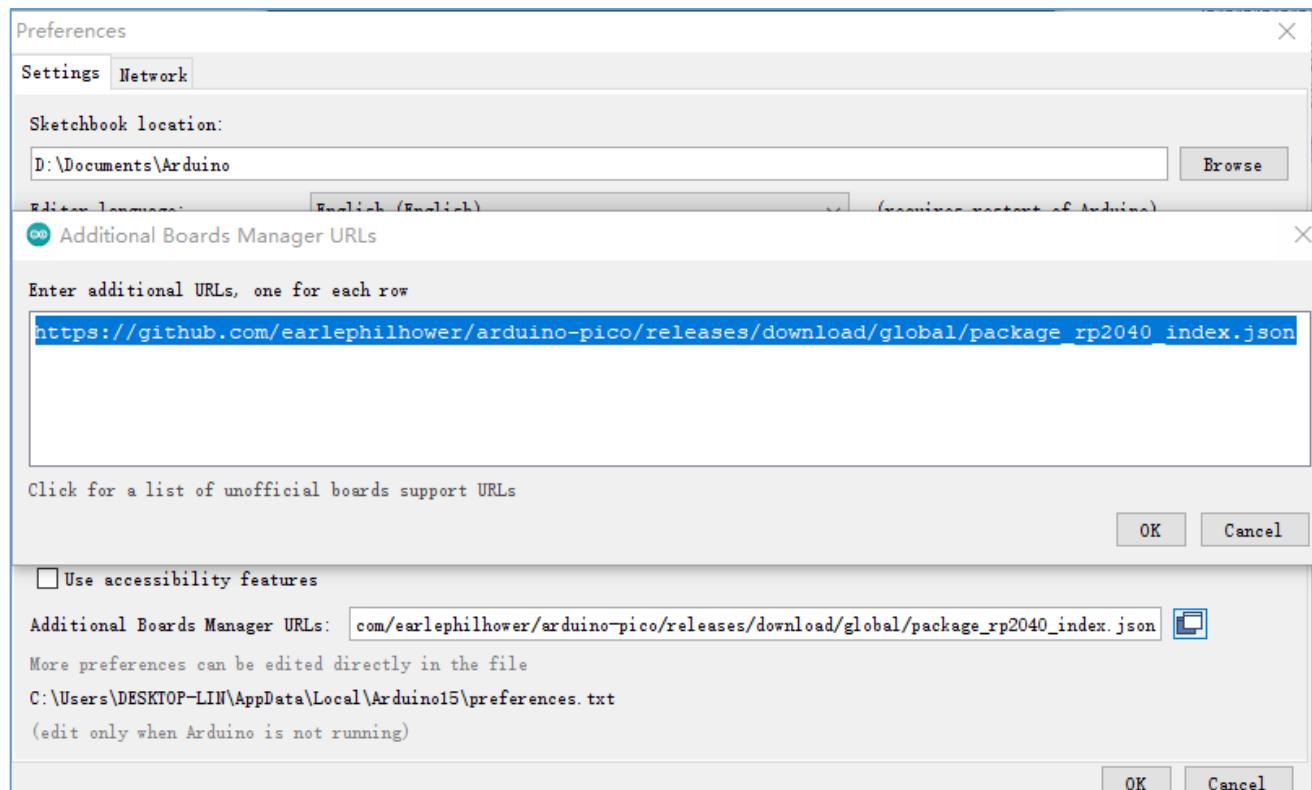


Installation of Development Board Support Package

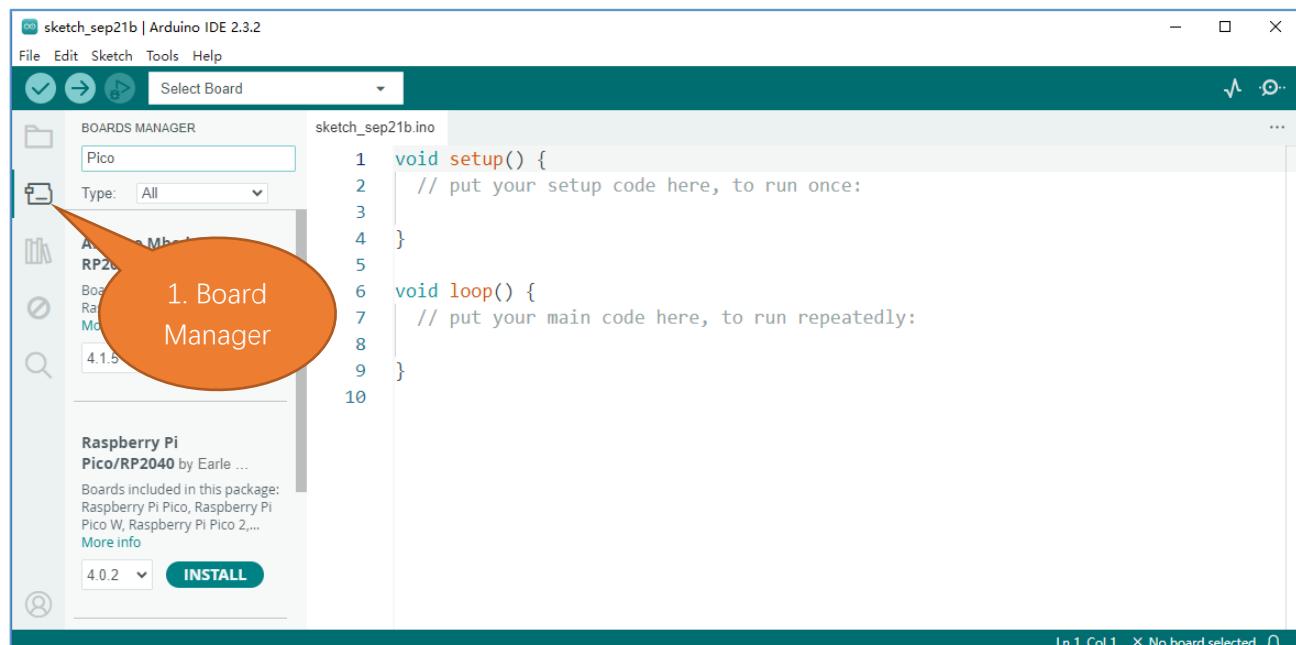
1. Make sure your network is of good connection.
2. Open Arduino IDE, and click File>Preference. In new pop-up window, find "Additional Boards Manager URLs", and replace with a new line:

https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json

As shown below:

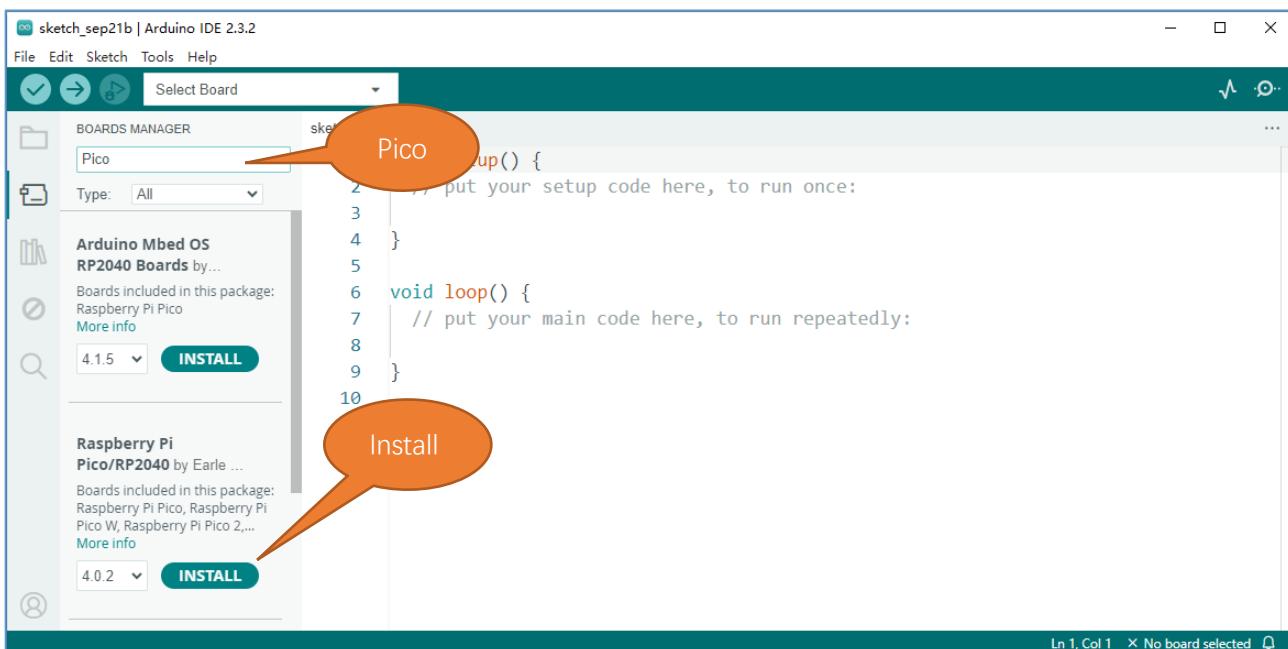


3. Open Arduino IDE; click Boards Manager on the left.



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

4. Enter Pico in the searching box, and select “Raspberry Pi Pico/RP2040” and click on Install.



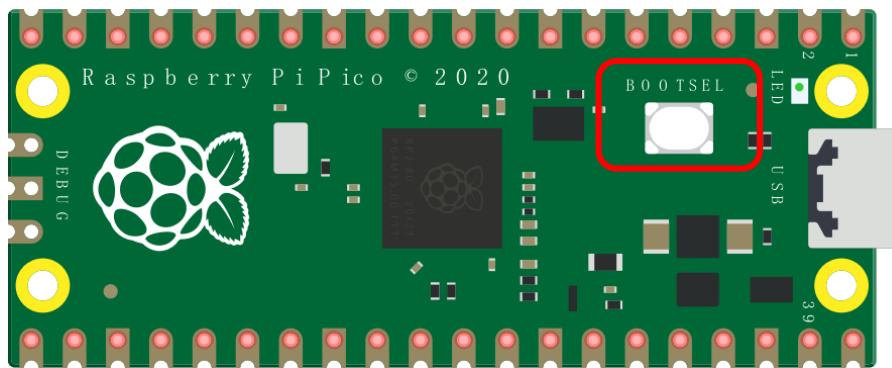
5. Click Yes in the pop-up “**dpinst-amd64.exe**” installation window. (Without it, you will fail to communicate with Arduino.) Thus far, we have finished installing the development support package.



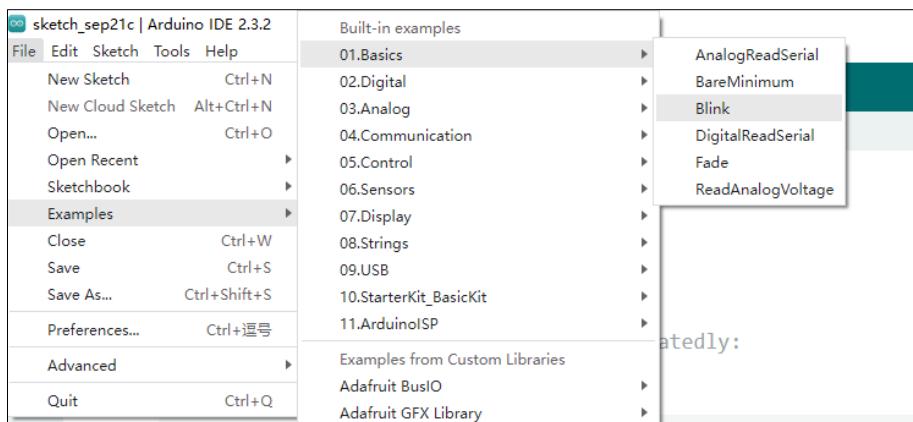
Uploading Arduino-compatible Firmware for Pico

If your Pico is new and you want to use Arduino to learn and develop, you need to upload an Arduino-compatible Firmware for it. Please refer to the following steps to configure.

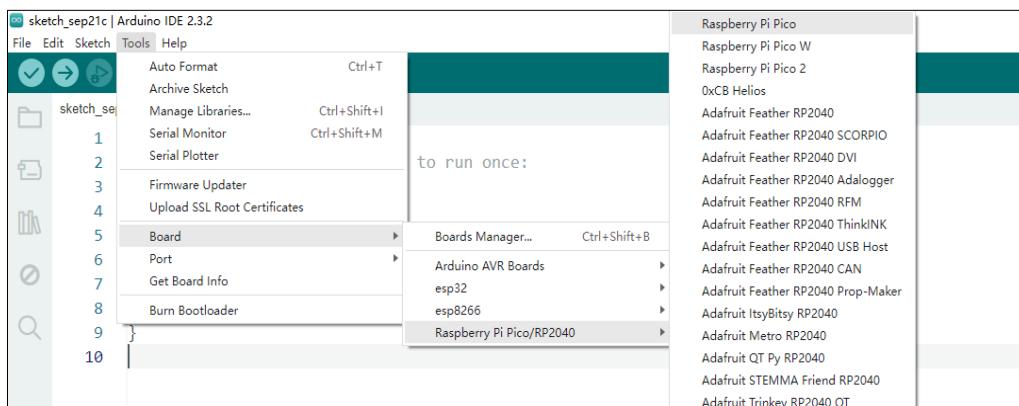
1. Disconnect Pico from computer. Keep pressing the white button (BOOTSEL) on Pico, and connect Pico to computer before releasing the button. (Note: Be sure to keep pressing the button before powering the Pico, otherwise the firmware will not download successfully)



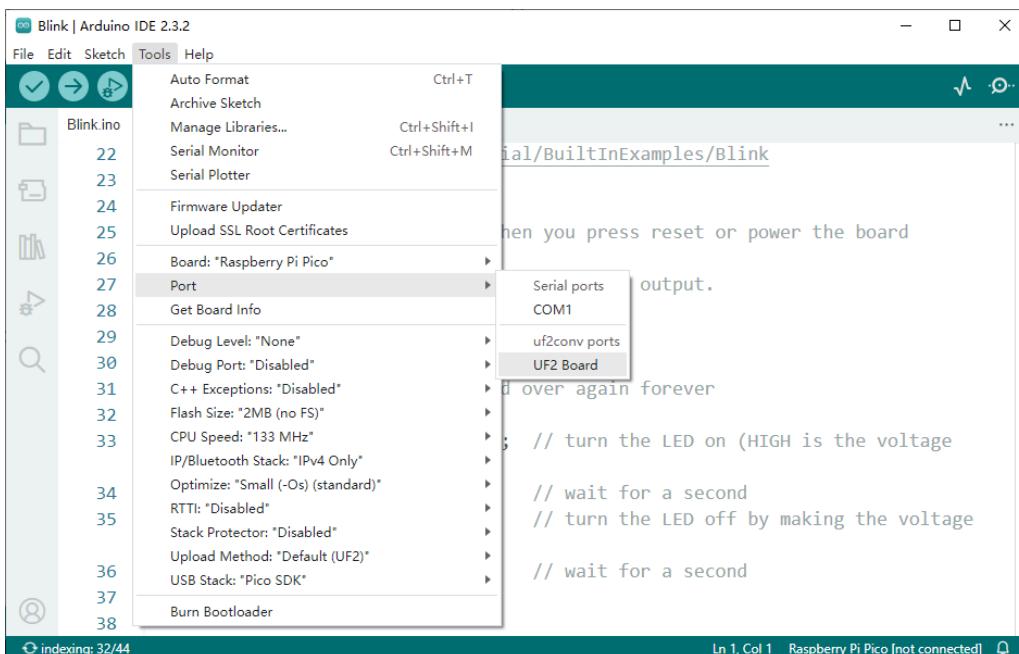
2. Open Arduino IDE. Click File>Examples>01.Basics>Blink.



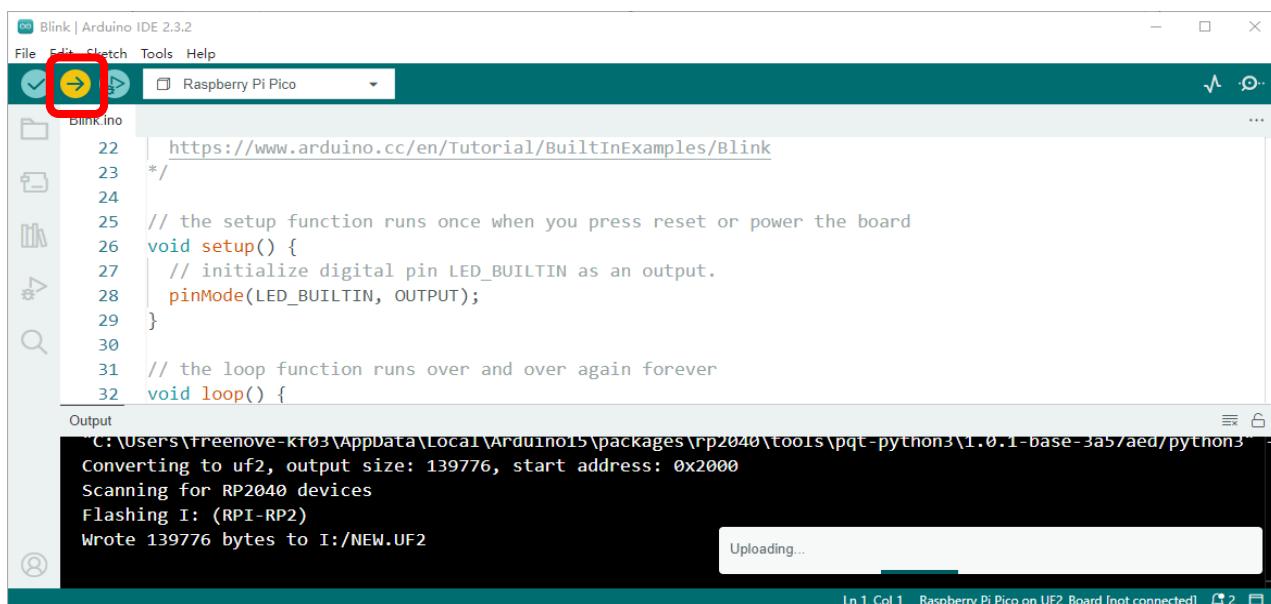
3. Click Tools>Board>Raspberry Pi RP2040 Boards>Raspberry Pi Pico.



4. Click Tools>Port>UF2 Board.



5. Upload sketch to Pico.



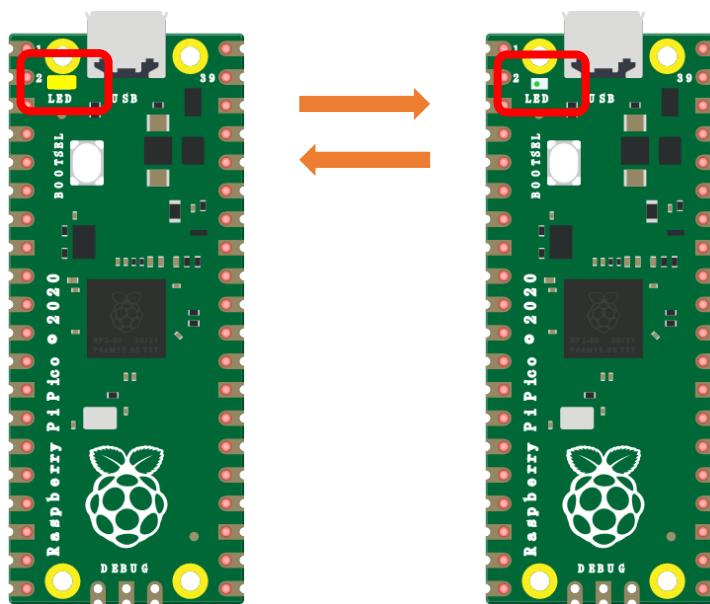
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



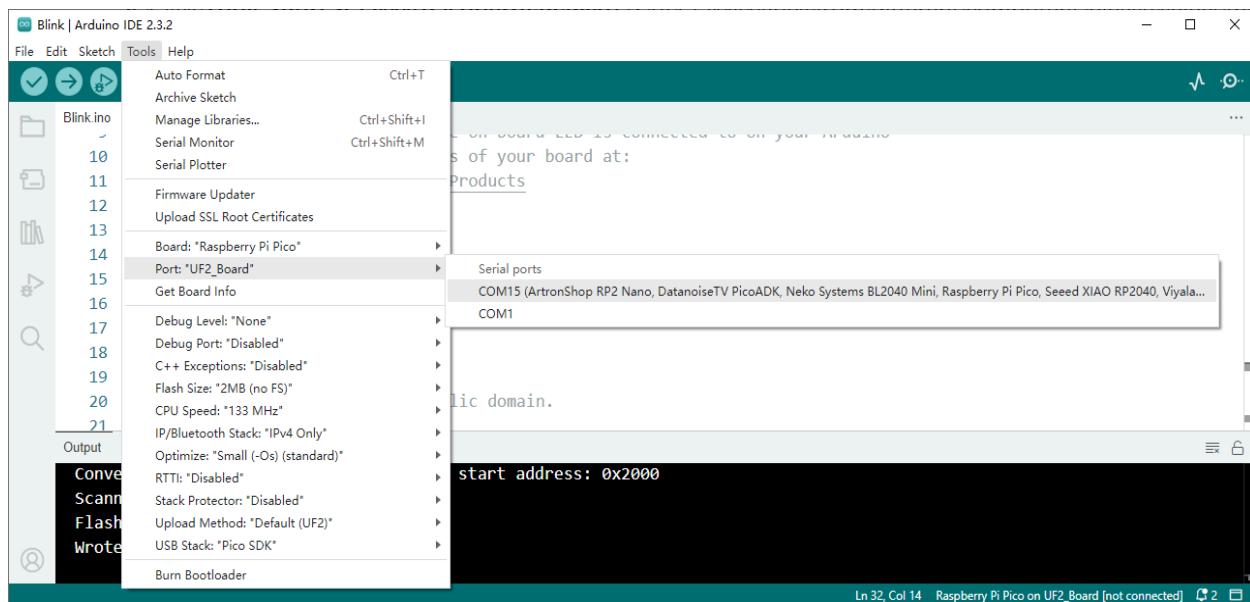
When the sketch finishes uploading, you can see the following prompt.

```
Output
Converting to uf2, output size: 139776, start address: 0x2000
Scanning for RP2040 devices
Flashing I: (RPI-RP2)
Wrote 139776 bytes to I:/NEW.UF2
Ln 32, Col 14 Raspberry Pi Pico on UF2_Board [not connected] 4:2
```

And you can see the indicator on Pico starts to flash.



5. Click **Tools>Port>COMx(Raspberry Pi Pico)**. X of COMx varies from different computers. Please select the correct one on your computer. In our case, it is COM15.

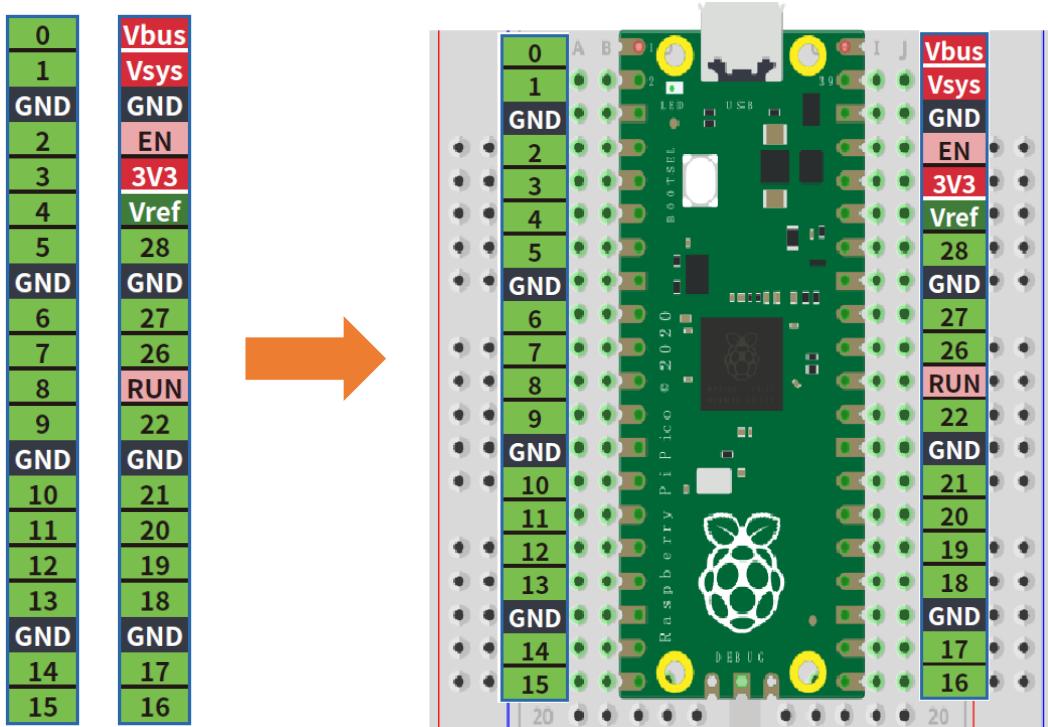


Note:

1. At the first time you use Arduino to upload sketch for Pico, you do not need to select port. After that, each time before uploading sketch, please check whether the port has been selected; otherwise, the downloading may fail.
2. Sometimes when using, Pico may lose firmware due to the code and fail to work. At this point, you can upload firmware for Pico as mentioned above.

Paste the Sticker on the Breadboard

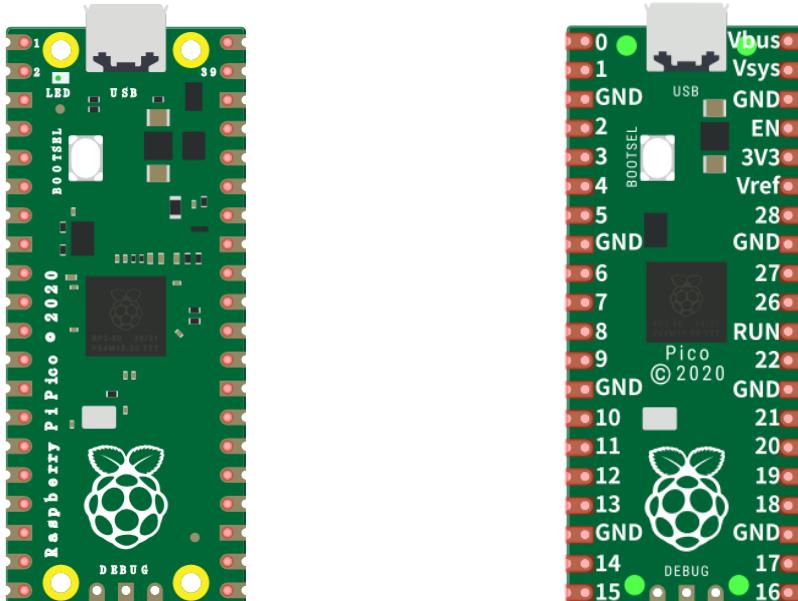
It is not difficult to use the Pico. However, officially, the pin functions are printed on the back of the board, which makes it inconvenient to use. To help users finish each project in the tutorial faster and easier, we provide stickers of the pin functions as follows:



You can paste the sticker on the blank area of the breadboard as above.

To make the tutorial more intuitive, we have made some changes to the simulation diagram as below. The left one is the actual Pico and the right one is its simulation diagram. Please note that to avoid misunderstanding.

In addition, the external pin interface functions of Pico, Pico W and Pico 2 are identical.



Chapter 1 LED (Important)

Note:

Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by one wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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

Project 1.1 Blink

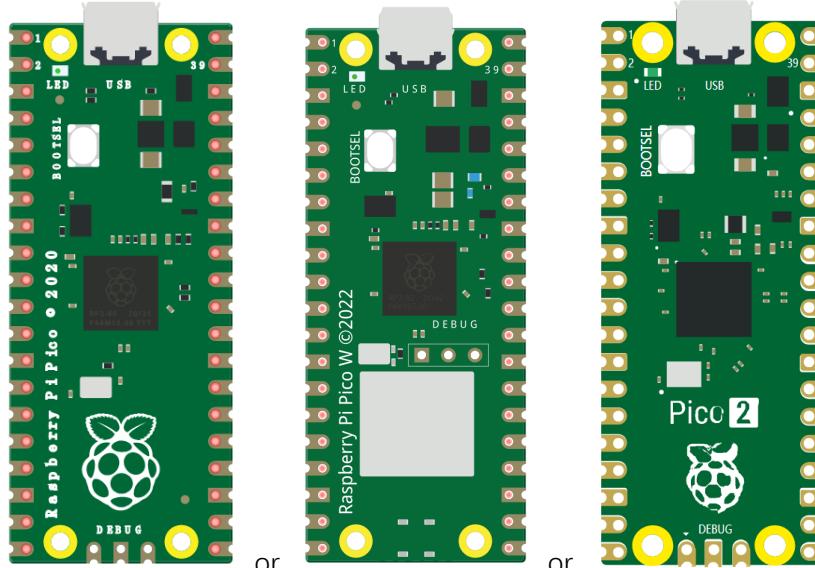
In this project, we will use Raspberry Pi Pico to control blinking a common LED.

If you have not installed Arduino IDE, you can click [Here](#).

If you have not uploaded firmware for Pico, you can click [Here](#) to upload.

Component List

Raspberry Pi Pico (or Pico W or Pico 2) x1



USB cable x1

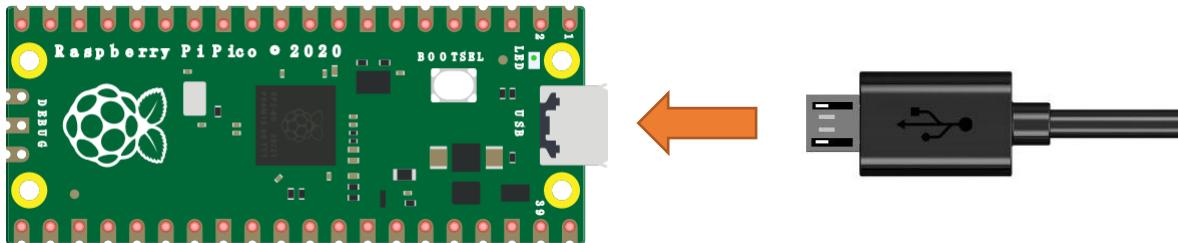




Power

Raspberry Pi Pico requires 5V power supply. You can either connect external 5V power supply to Vsys pin of Pico or connect a USB cable to the onboard USB base to power Pico.

In this tutorial, we use USB cable to power Pico and upload sketches.



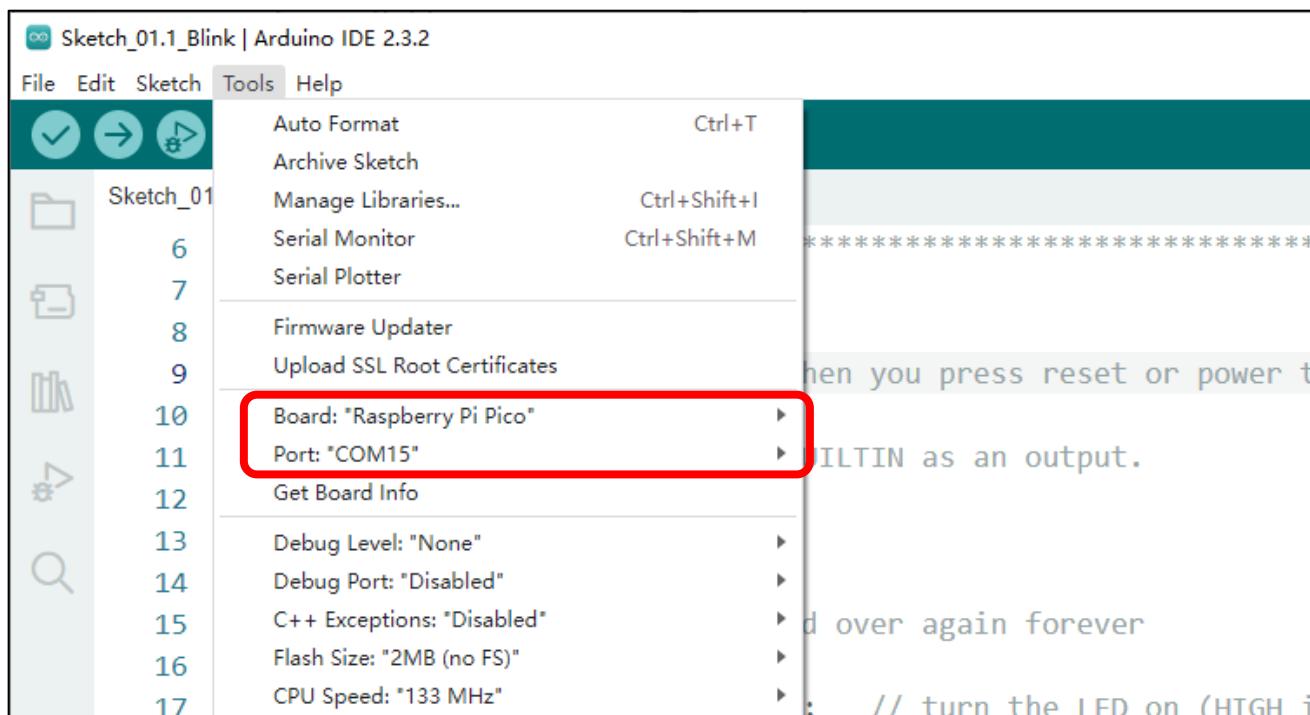
Sketch

The onboard LED of Raspberry Pi Pico is controlled by GP25. When GP25 outputs high level, LED lights up; when it outputs low, LED lights off. You can open the provided code:

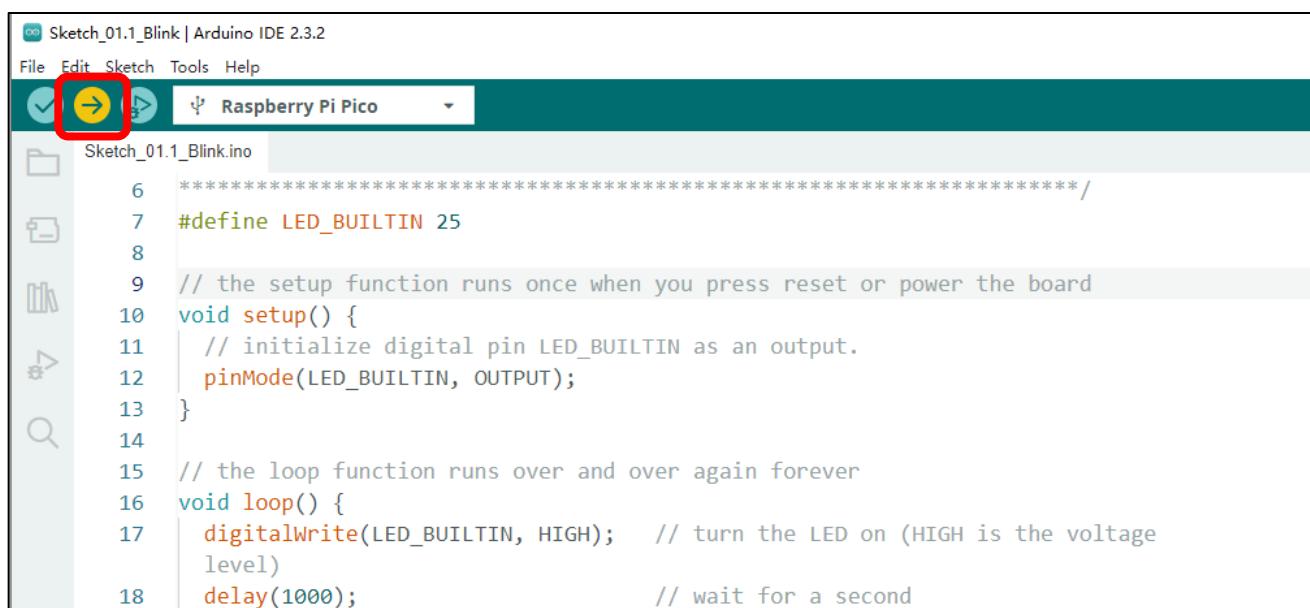
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\Sketches\Sketch_01.1_Blink.

Before uploading code to Pico, please check the configuration of Arduino IDE.

Click Tools, make sure Board and Port are as follows:

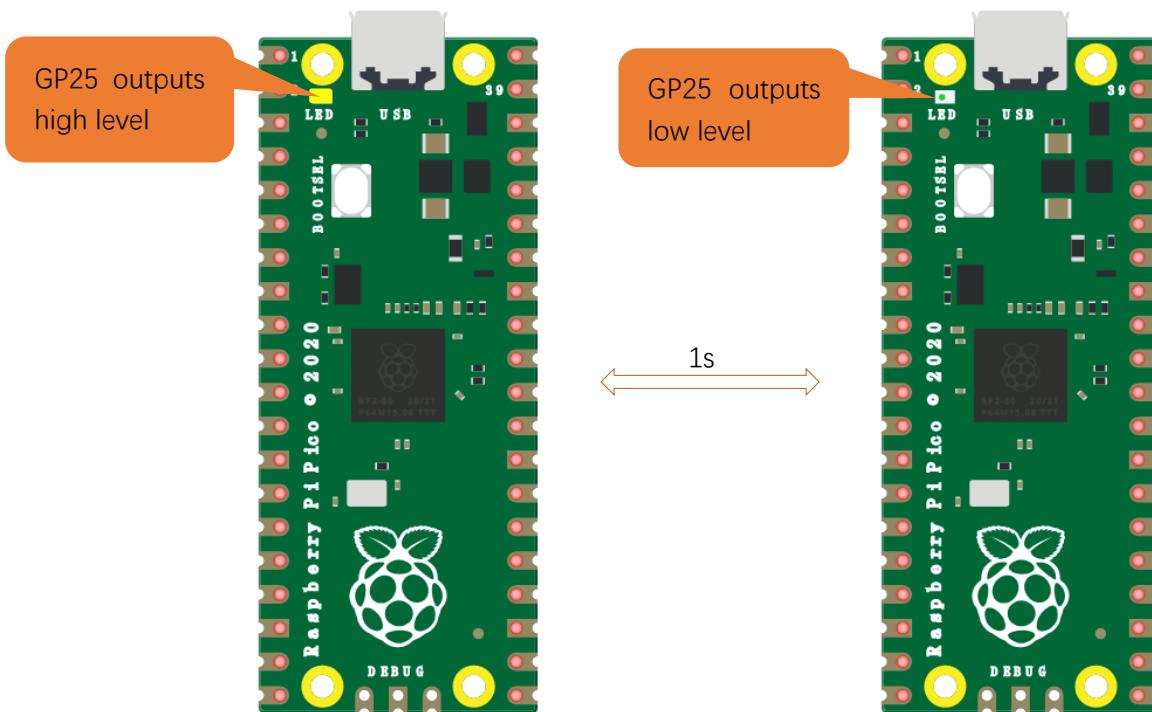


Click "Upload" to upload the sketch to Pico.



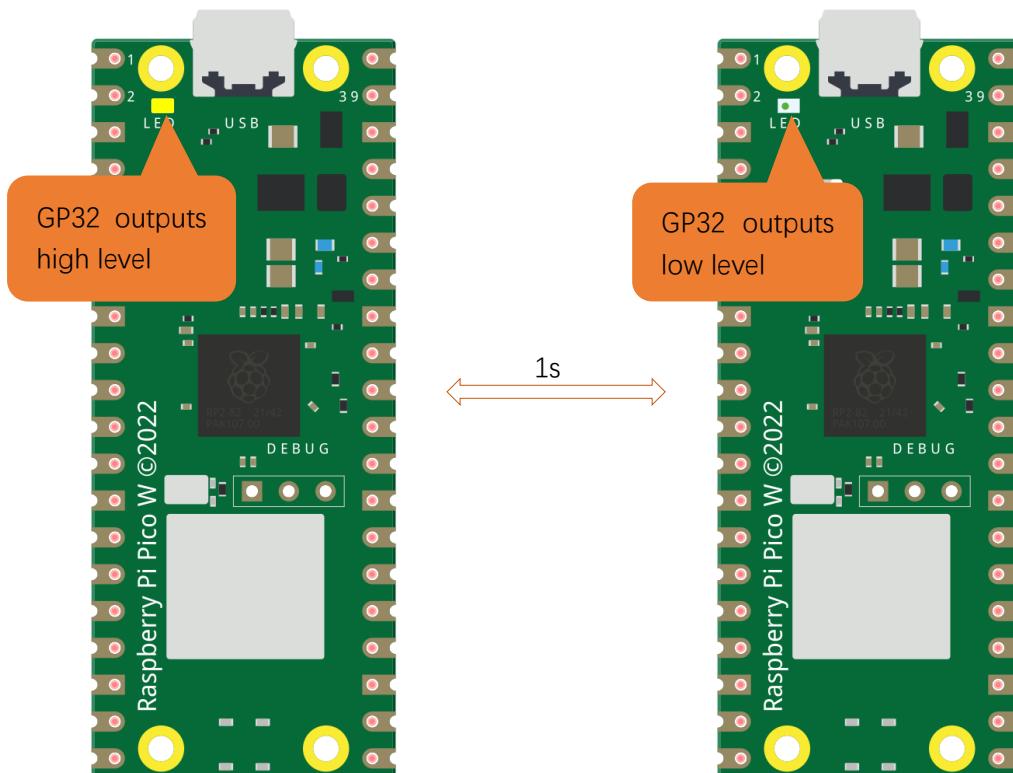
If you have any concerns, please contact us at support@freenove.com

Pico's on-board LED lights on and off every 1s, flashing cyclically.



Note: Pico's on-board LED is driven by GPIO25. Pico W's on-board LED uses WL_GPIO0, which is defined as GPIO32 on Arduino.

If you use Pico W, please change "# define LED_BUILTIN 25" to "# define LED_BUILTIN 32" in the code.



Any concerns? ✉ support@freenove.com

The following is the program code:

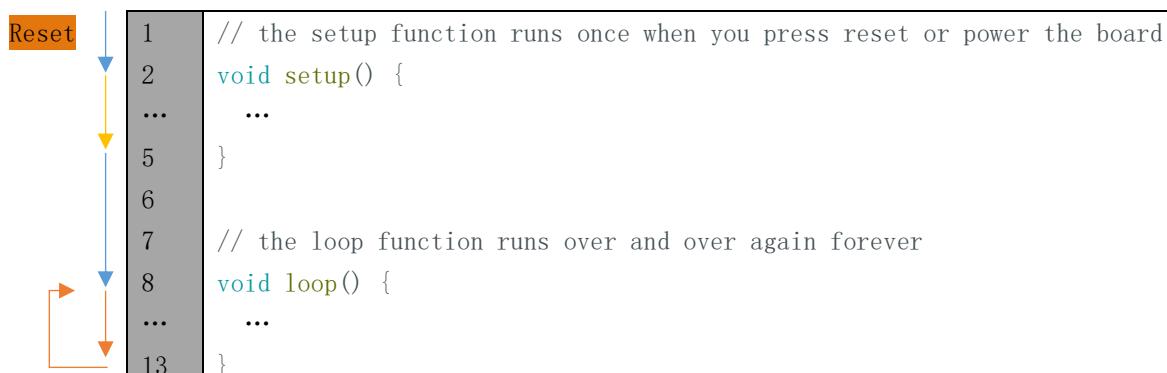
```

1 #define LED_BUILTIN 25
2
3 // the setup function runs once when you press reset or power the board
4 void setup() {
5     // initialize digital pin LED_BUILTIN as an output.
6     pinMode(LED_BUILTIN, OUTPUT);
7 }
8
9 // the loop function runs over and over again forever
10 void loop() {
11     digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on (HIGH is the voltage level)
12     delay(1000);                      // wait for a second
13     digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
14     delay(1000);                      // wait for a second
15 }
```

The Arduino IDE code usually contains two basic functions: void setup() and void loop().

After the board is reset, the setup() function will be executed firstly, and then the loop() function.

setup() function is generally used to write code to initialize the hardware. And loop() function is used to write code to achieve certain functions. loop() function is executed repeatedly. When the execution reaches the end of loop(), it will back to the beginning of loop() to run again.



In the circuit, GP25 of Pico is connected to the LED, so the LED pin is defined as 25.

```
1 #define LED_BUILTIN 25
```

This means that after this line of code, all LED_BUILTIN will be regarded as 25.

In the setup() function, first, we set the LED_BUILTIN as output mode, which can make the port output high or low level.

```

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

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

```
10 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
```

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

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



Then set the LED_BUILTIN to output low level, and LED lights off. One second later, the execution of loop() function will be completed.

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

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

Reference

void pinMode(int pin, int mode);

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

Parameters

pin: the pin number to set the mode of LED.

mode: INPUT, OUTPUT, INPUT_PULLDOWN, or INPUT_PULLUP.

void digitalWrite (int pin, int value);

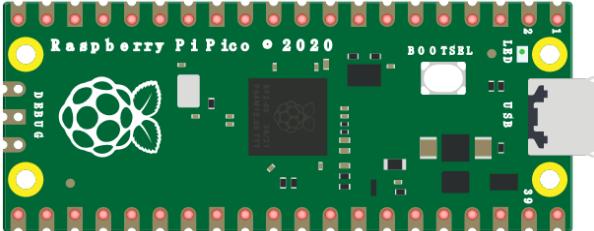
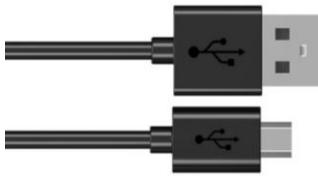
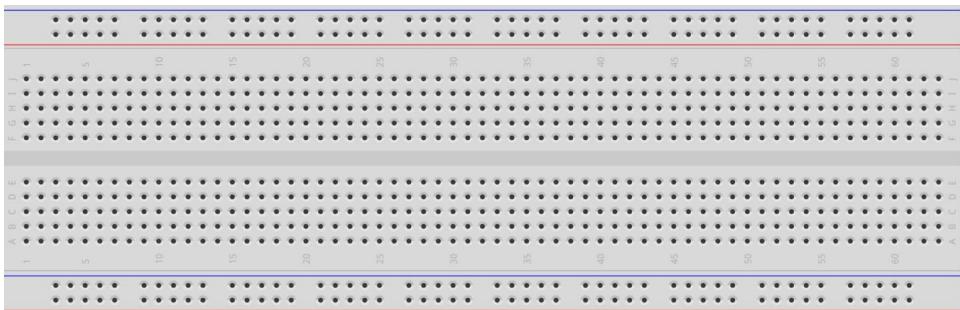
Writes the value HIGH or LOW (1 or 0) to the given pin that must have been previously set as an output.

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

Project 1.2 Blink

In this project, we will use Raspberry Pi Pico to control blinking a common LED.

Component List

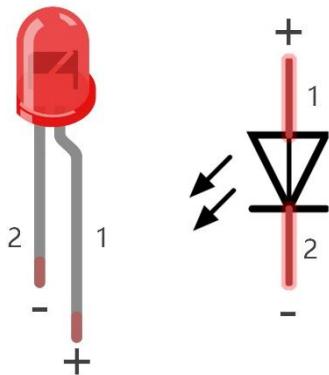
Raspberry Pi Pico x1	USB Cable x1
	
Breadboard x1	
	
LED x1	Resistor 220Ω x1
	
	Jumper
	

Component Knowledge

LED

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

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



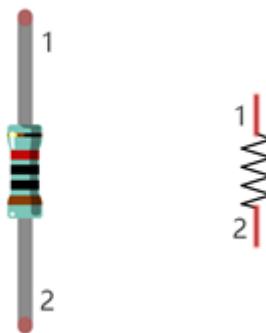
LED	Voltage	Maximum current	Recommended current
Red	1.9 - 2.2V	20mA	10mA
Green	2.9 - 3.4V	10mA	5mA
Blue	2.9 - 3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

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

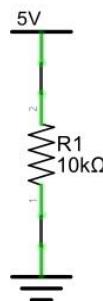
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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



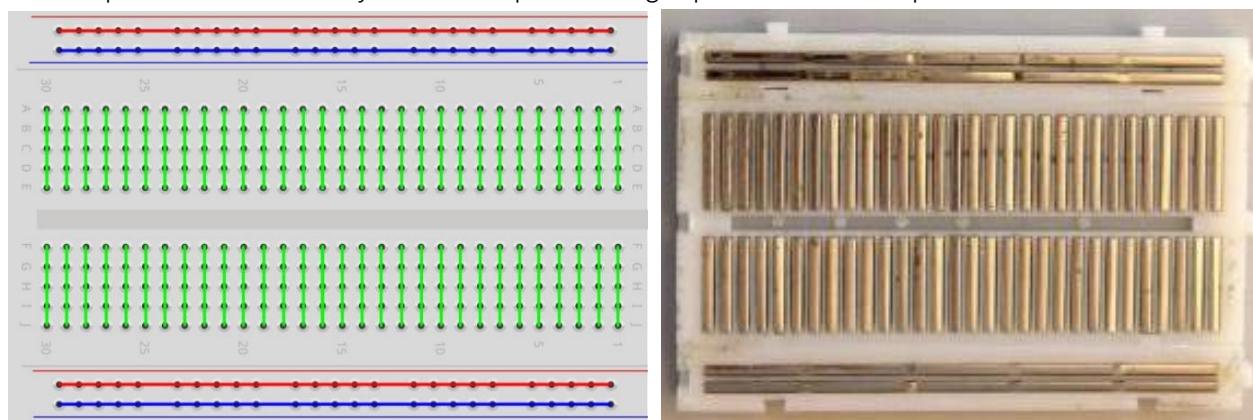
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

Breadboard

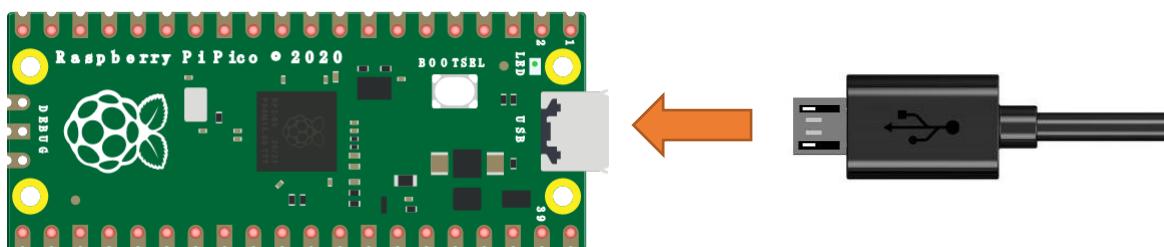
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached.

The left picture shows the way to connect pins. The right picture shows the practical internal structure.



Power

In this tutorial, we connect Raspberry Pi Pico and computer with a USB cable.



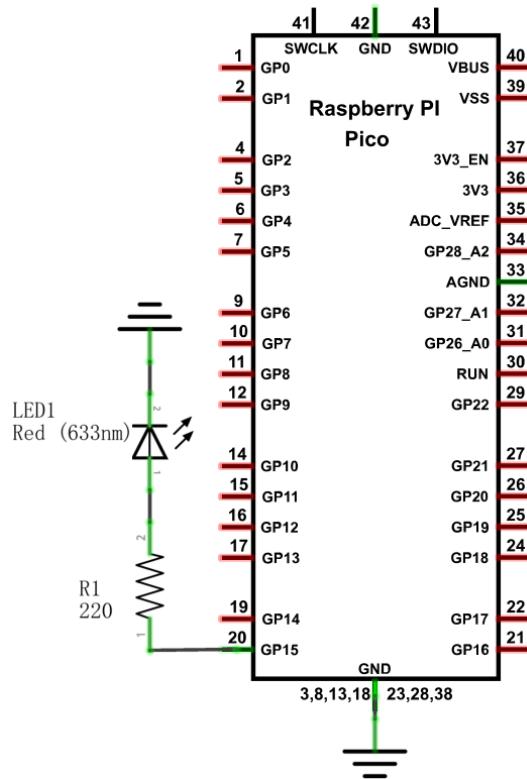
Circuit

First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

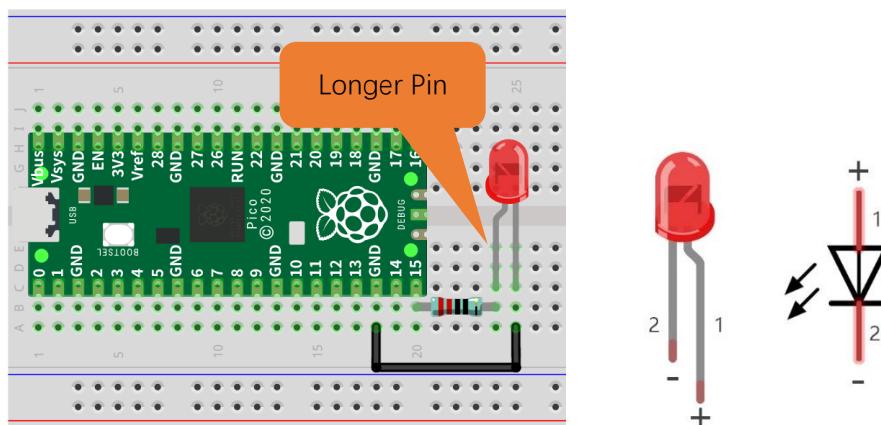
CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)!

WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram



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



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Any concerns? ✉ support@freenove.com

Sketch

According to the circuit diagram, when GP15 of Pico outputs high level, LED lights up; when it outputs low, LED lights off. Therefore, we can make LED flash repeatedly by controlling GP15 to output high and low repeatedly.

You can open the provided code:

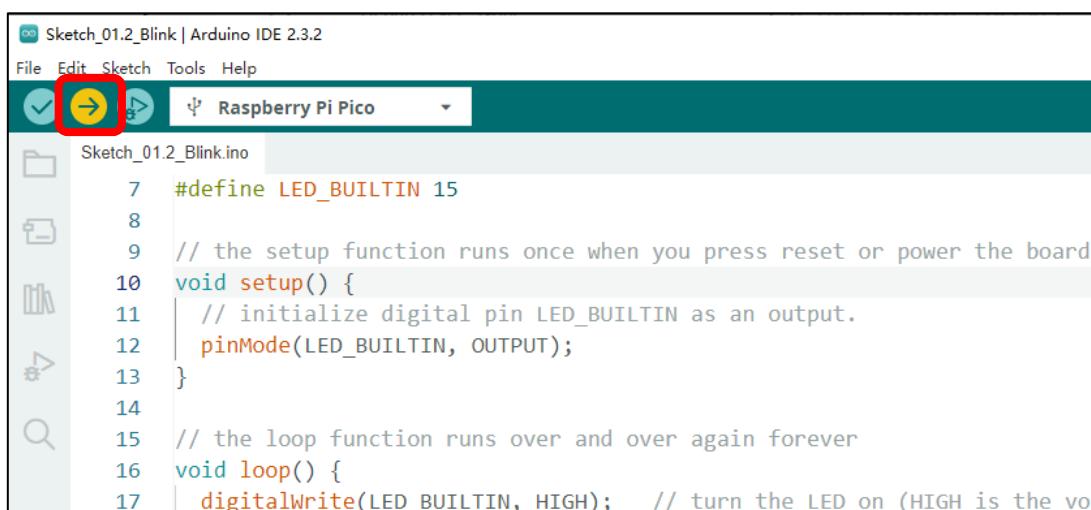
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.2_Blink.

Before uploading code to Pico, please check the configuration of Arduino IDE.

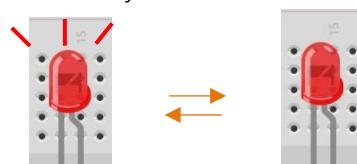
Click Tools, make sure Board and Port are as follows:



Click "Upload" to upload the sketch to Pico.



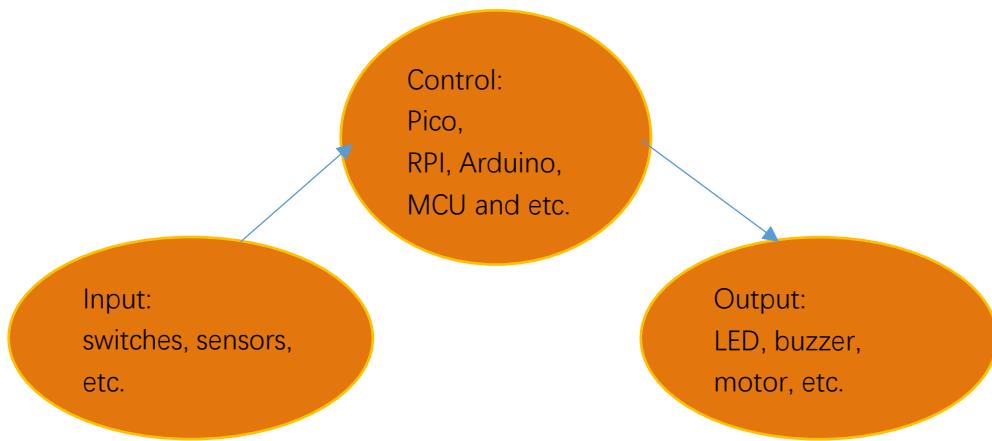
Click "Upload". Download the code to Pico and your LED in the circuit starts Blink.



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

Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and Raspberry Pi Pico was the control part. In practical applications, we not only make LEDs blink, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as lighting up LEDs, turning ON a buzzer and so on.



Next, we make a simple project: build a control system with button, LED and Raspberry Pi Pico.

Input: Button

Control: Raspberry Pi Pico

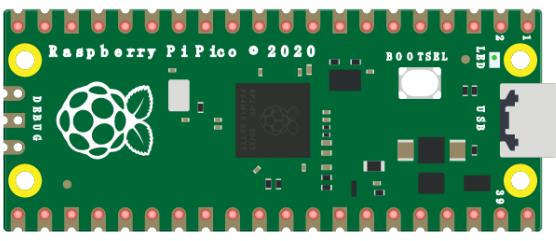
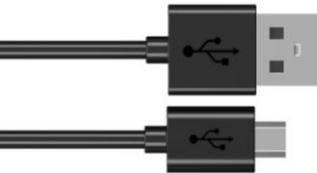
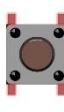
Output: LED

Project 2.1 Button & LED

Note: Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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

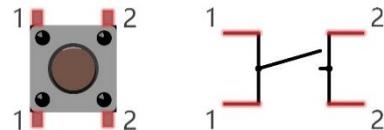
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper		LED x1	
		Resistor 220Ω x1	
		Resistor 10kΩ x2	
		Push button x1	

Component Knowledge

Push button

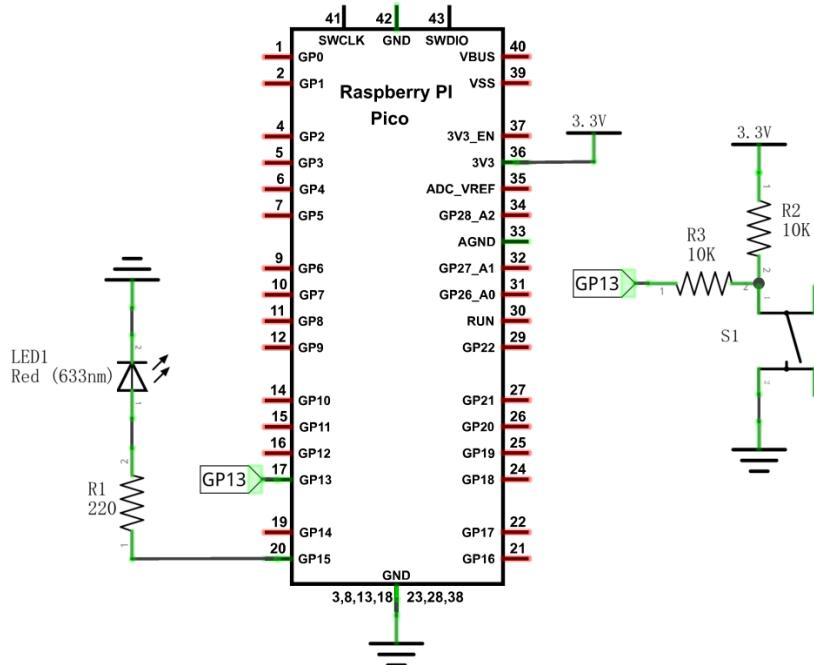
This type of Push Button Switch has four pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



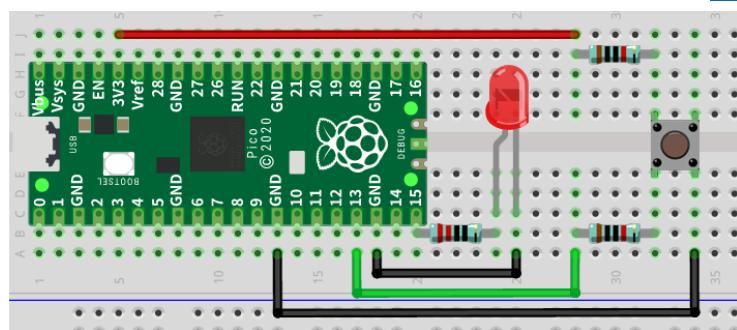
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Any concerns? ✉ support@freenove.com

Sketch

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

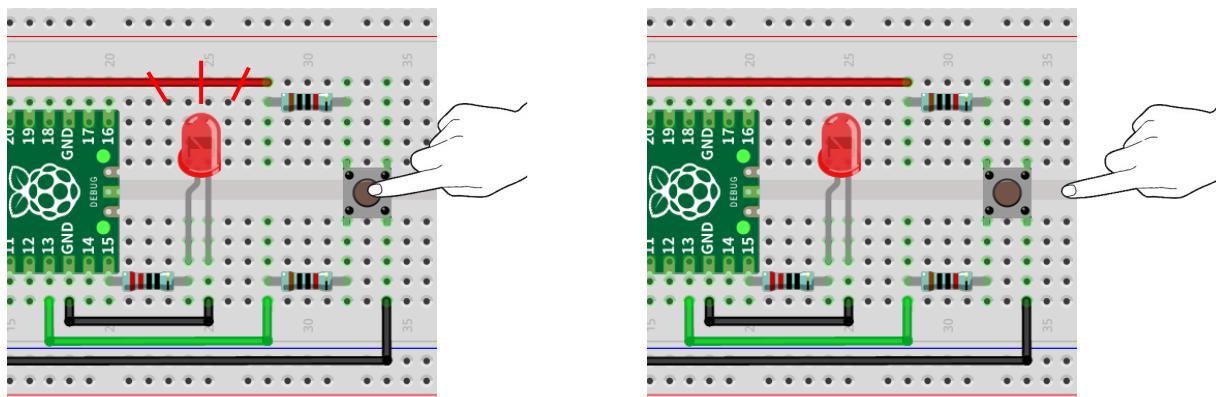
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\Sketches\Sketch_02.1_ButtonAndLed.

Sketch_02.1_ButtonAndLed



```
Sketch_02.1_ButtonAndLed | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_02.1_ButtonAndLed.ino
1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3
4 void setup() {
5     // initialize digital pin PIN_LED as an output.
6     pinMode(PIN_LED, OUTPUT);
7     pinMode(PIN_BUTTON, INPUT);
8 }
9
10 // the loop function runs over and over again forever
11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED,HIGH);
14     }else{
15         digitalWrite(PIN_LED,LOW);
16     }
17 }
```

Upload the sketch to Pico. When pressing the button, LED lights up; when releasing the button, LED lights OFF.





The following is the program code:

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3 // the setup function runs once when you press reset or power the board
4 void setup() {
5     // initialize digital pin PIN_LED as an output.
6     pinMode(PIN_LED, OUTPUT);
7     pinMode(PIN_BUTTON, INPUT);
8 }
9
10 // the loop function runs over and over again forever
11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED, HIGH);
14     }else{
15         digitalWrite(PIN_LED, LOW);
16     }
17 }
```

In the circuit connection, LED and button are connected with GP15 and GP13 respectively, so define ledPin and buttonPin as 15 and 13 respectively.

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
```

In the while cycle of main function, use digitalRead(buttonPin) to determine the state of button. When the button is pressed, the function returns low level and the result of "if" is true, so LED lights up. Otherwise, LED lights OFF.

```

11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED, HIGH);
14     }else{
15         digitalWrite(PIN_LED, LOW);
16     }
17 }
```

Reference

```
int digitalRead (int pin);
```

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

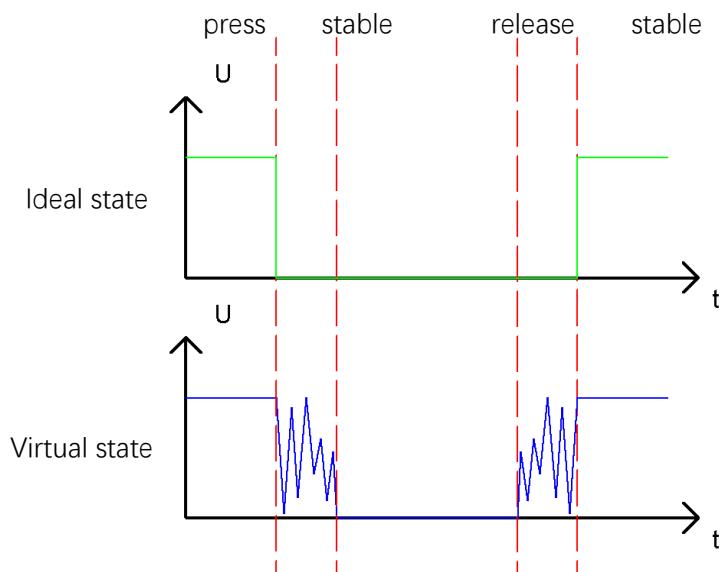
Project 2.2 MINI table lamp

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

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

Debounce for Push Button

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



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



Sketch

Upload following sketch:

Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.2_TableLamp.

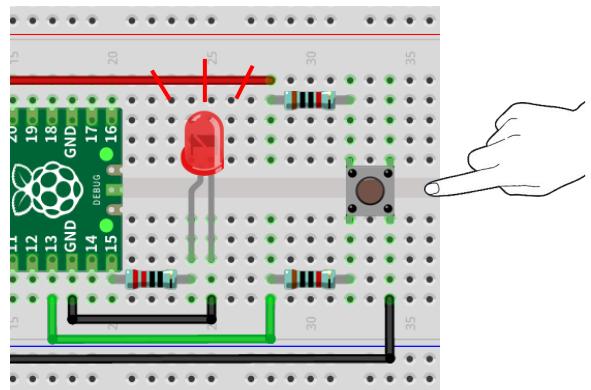
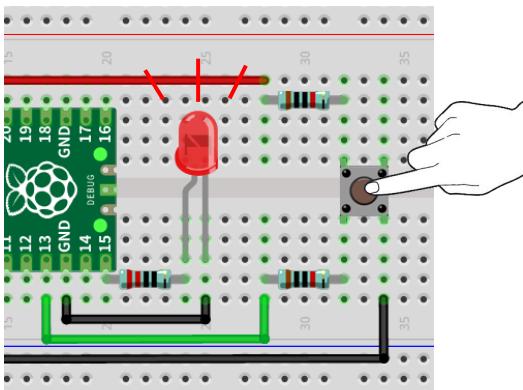
Sketch_02.2_TableLamp

```

Sketch_02.2_TableLamp | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_02.2_TableLamp.ino
  7 #define PIN_LED      15
  8 #define PIN_BUTTON  13
  9 bool ledState = false;
10
11 void setup() {
12     // initialize digital pin PIN_LED as an output.
13     pinMode(PIN_LED, OUTPUT);
14     pinMode(PIN_BUTTON, INPUT);
15 }
16
17 // the loop function runs over and over again forever
18 void loop() {
19     if (digitalRead(PIN_BUTTON) == LOW) {
20         delay(20);
21         if (digitalRead(PIN_BUTTON) == LOW) {
22             reverseGPIO(PIN_LED);
23         }
24         while (digitalRead(PIN_BUTTON) == LOW);
25     }
26 }
27
28 void reverseGPIO(int pin) {
29     ledState = !ledState;
30     digitalWrite(pin, ledState);
31 }

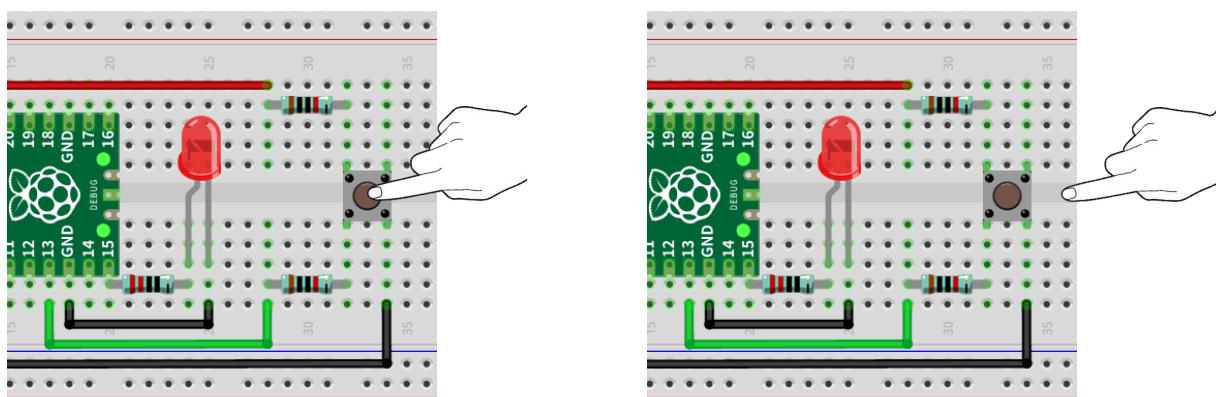
```

Upload the sketch to Pico. When the button is pressed, LED lights up; when the button is released, LED is still ON.



Any concerns? ✉ support@freenove.com

When the button is pressed again, LED turns OFF; when released, LED keeps OFF.



The following is the program code:

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3 bool ledState = false;
4
5 void setup() {
6     // initialize digital pin PIN_LED as an output.
7     pinMode(PIN_LED, OUTPUT);
8     pinMode(PIN_BUTTON, INPUT);
9 }
10
11 // the loop function runs over and over again forever
12 void loop() {
13     if (digitalRead(PIN_BUTTON) == LOW) {
14         delay(20);
15         if (digitalRead(PIN_BUTTON) == LOW) {
16             reverseGPIO(PIN_LED);
17         }
18         while (digitalRead(PIN_BUTTON) == LOW);
19     }
20 }
21
22 void reverseGPIO(int pin) {
23     ledState = !ledState;
24     digitalWrite(pin, ledState);
25 }
```

In the circuit connection, LED and button are connected with GP15 and GP13 respectively, so define ledPin and buttonPin as 15 and 13 respectively.

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
```

Define a variable to store the status of LED.

```
3 bool ledState = false;
```



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

```
13  if (digitalRead(PIN_BUTTON) == LOW) {  
14      delay(20);  
15      if (digitalRead(PIN_BUTTON) == LOW) {  
16          reverseGPIO(PIN_LED);  
17      }  
18      while (digitalRead(PIN_BUTTON) == LOW);  
19  }
```

When the button is pressed, reverseGPIO function is called to change the variable that controls LED's statue, and write it to Pico to reverse the pin's output state.

```
22  void reverseGPIO(int pin) {  
23      ledState = !ledState;  
24      digitalWrite(pin, ledState);  
25  }
```

Chapter 3 LED Bar

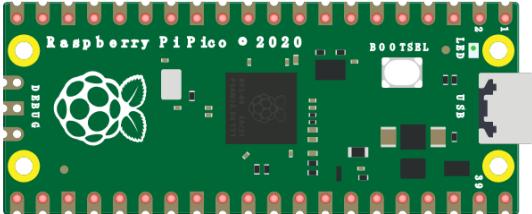
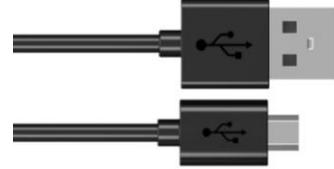
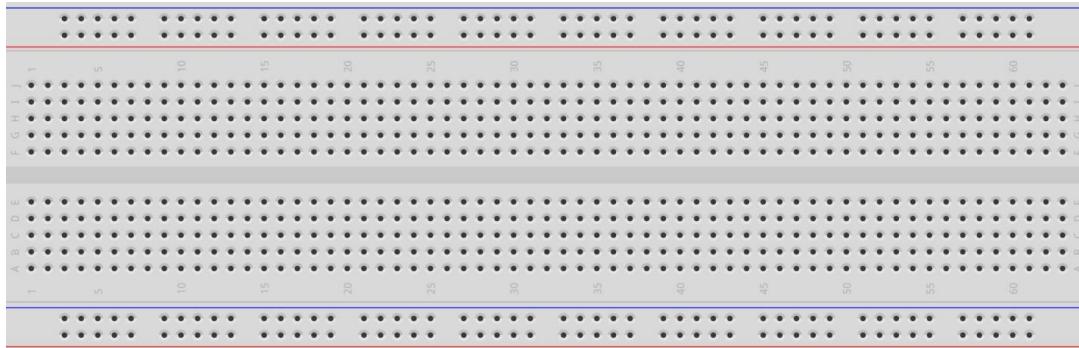
We have learned how to control an LED blinking, next we will learn how to control a number of LEDs.

Project 3.1 Flowing Light

Note: Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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

Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
Jumper	LED bar graph x1	Resistor 220Ω x10

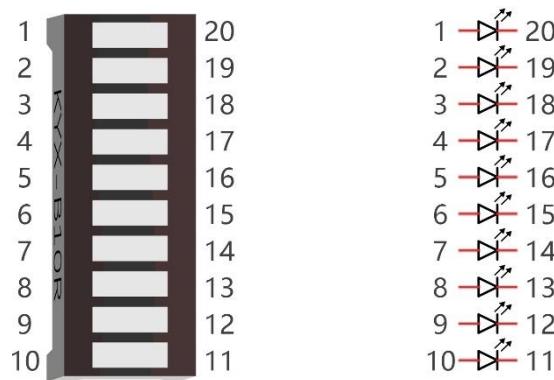


Component Knowledge

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

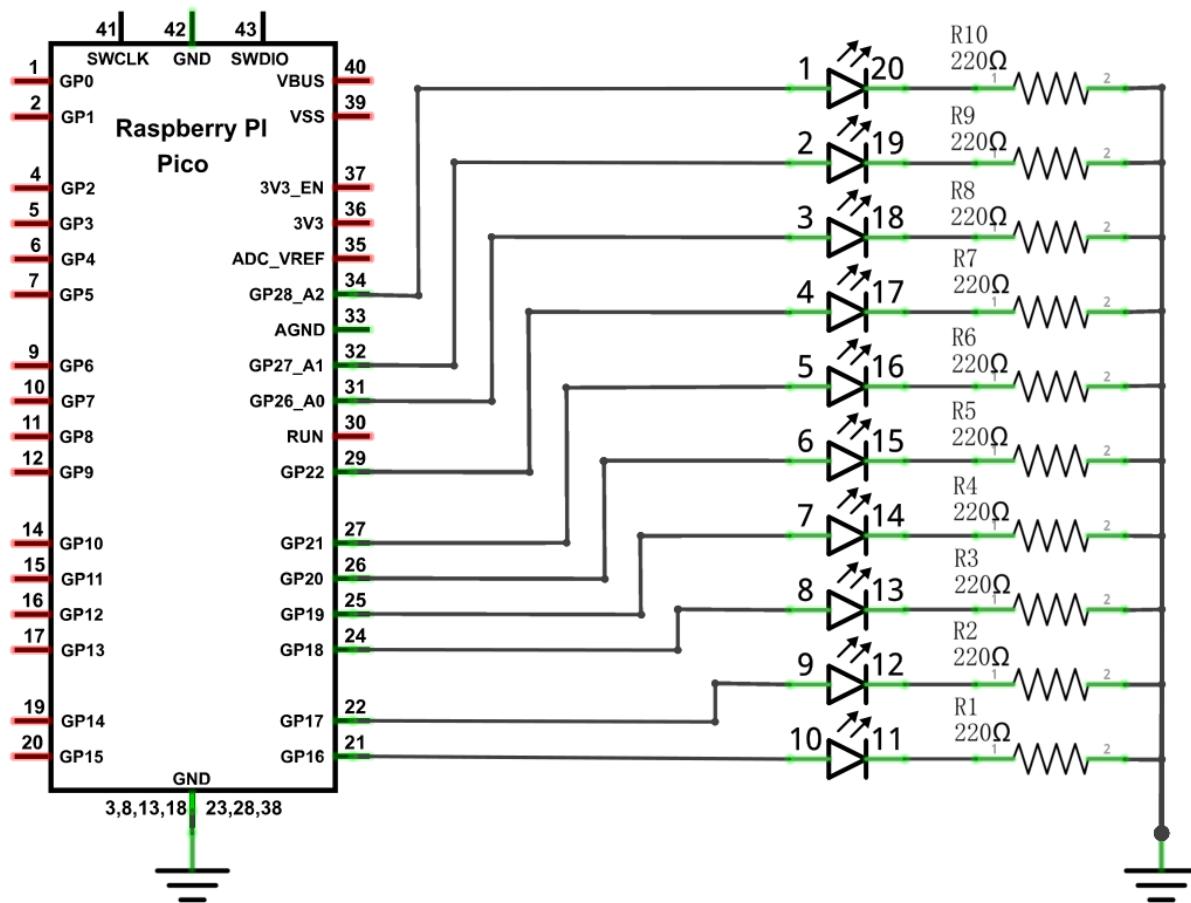
LED bar

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

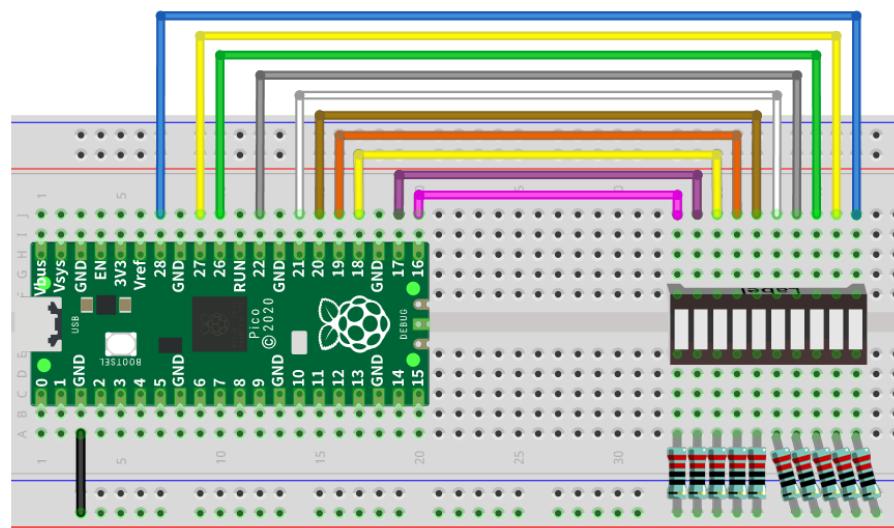


Circuit

Schematic diagram



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



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



Sketch

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

Upload following sketch:

Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_03.1_FlowingLight.

Sketch_03.1_FlowingLight

```

Sketch_03.1_FlowingLight | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_03.1_FlowingLight.ino
7 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};
8 int ledCounts;
9
10 void setup() {
11     ledCounts = sizeof(ledPins);
12     for (int i = 0; i < ledCounts; i++) {
13         pinMode(ledPins[i], OUTPUT);
14     }
15 }
16
17 void loop() {
18     for (int i = 0; i < ledCounts; i++) {
19         digitalWrite(ledPins[i], HIGH);
20         delay(100);
21         digitalWrite(ledPins[i], LOW);
22     }
23     for (int i = ledCounts - 1; i > -1; i--) {
24         digitalWrite(ledPins[i], HIGH);
25         delay(100);
26         digitalWrite(ledPins[i], LOW);
27     }
28 }

```

Click Upload to upload the sketch to Pico. LEDs of LED bar graph lights up one by one from left to right and then back from right to left.



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

Any concerns? ✉ support@freenove.com

The following is the program code:

```
1 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};  
2 int ledCounts;  
3  
4 void setup() {  
5     ledCounts = sizeof(ledPins);  
6     for (int i = 0; i < ledCounts; i++) {  
7         pinMode(ledPins[i], OUTPUT);  
8     }  
9 }  
10  
11 void loop() {  
12     for (int i = 0; i < ledCounts; i++) {  
13         digitalWrite(ledPins[i], HIGH);  
14         delay(100);  
15         digitalWrite(ledPins[i], LOW);  
16     }  
17     for (int i = ledCounts - 1; i > -1; i--) {  
18         digitalWrite(ledPins[i], HIGH);  
19         delay(100);  
20         digitalWrite(ledPins[i], LOW);  
21     }  
22 }
```

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

```
1 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};
```

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

```
5 ledCounts = sizeof(ledPins);  
6 for (int i = 0; i < ledCounts; i++) {  
7     pinMode(ledPins[i], OUTPUT);  
8 }
```

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

```
12 for (int i = 0; i < ledCounts; i++) {  
13     digitalWrite(ledPins[i], HIGH);  
14     delay(100);  
15     digitalWrite(ledPins[i], LOW);  
16 }  
17 for (int i = ledCounts - 1; i > -1; i--) {  
18     digitalWrite(ledPins[i], HIGH);  
19     delay(100);  
20     digitalWrite(ledPins[i], LOW);  
21 }
```



Chapter 4 Analog & PWM

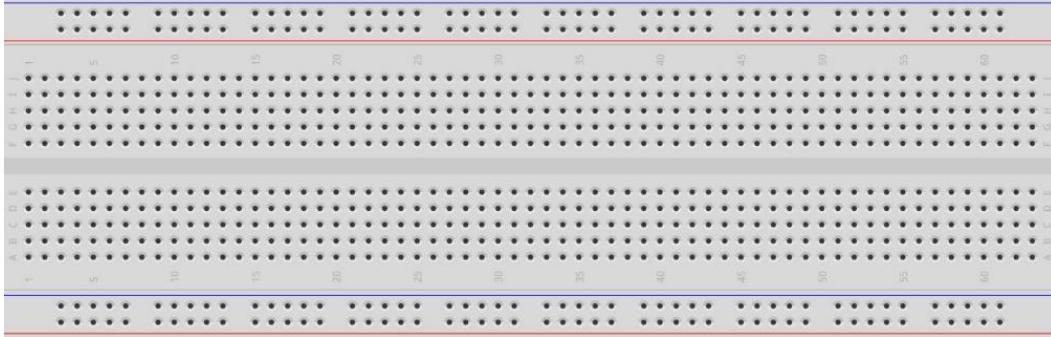
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That is what we are going to learn.

First, let us learn how to control the brightness of an LED.

Project 4.1 Breathing LED

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

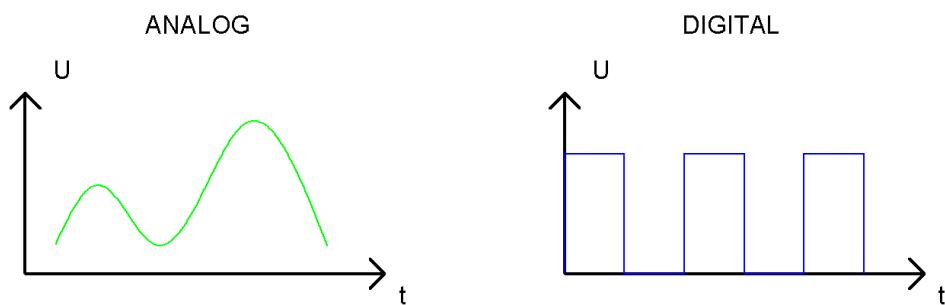
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
LED x1		Resistor 220Ω x1	Jumper

Related Knowledge

Analog & Digital

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



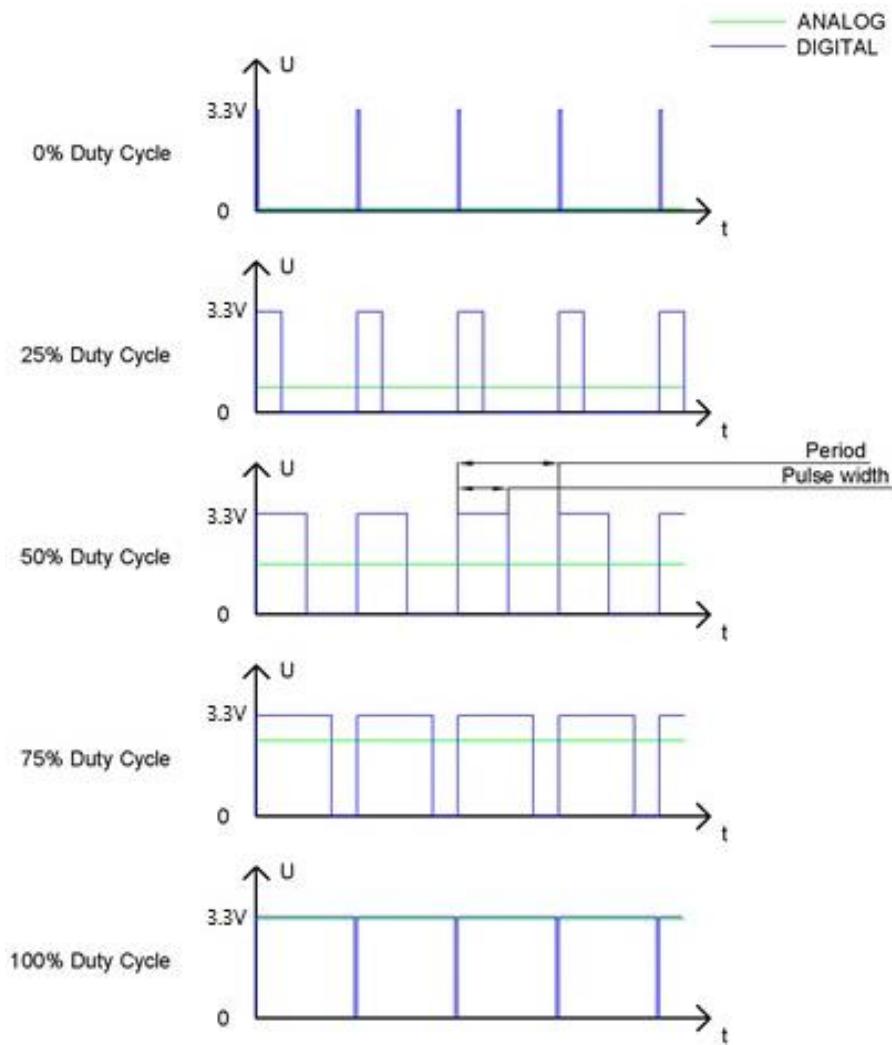
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Common processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals)

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on. It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent to the corresponding analog. Therefore, we can control the output power of the LED and other output modules to achieve different effects.

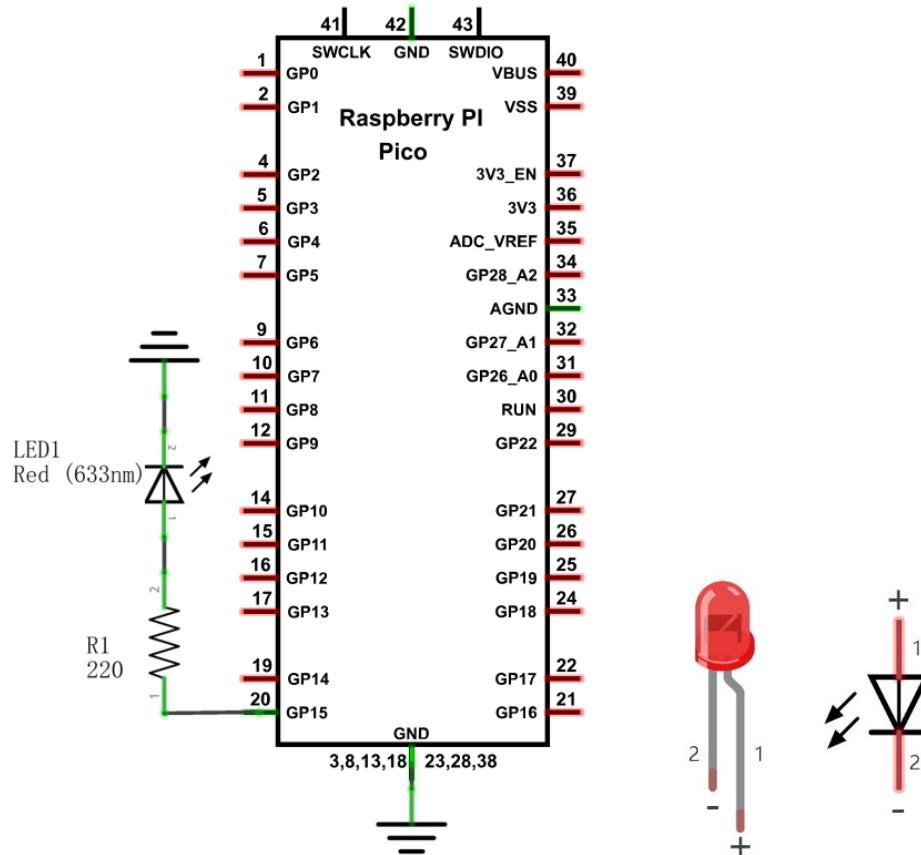
Raspberry Pi Pico and PWM

Raspberry Pi Pico has 16 PWM channels, each of which can control frequency and duty cycle independently. Every pin on Raspberry Pi Pico can be configured as PWM output. In Arduino, PWM frequency is set to 500Hz. You can change the PWM output by changing duty cycle.

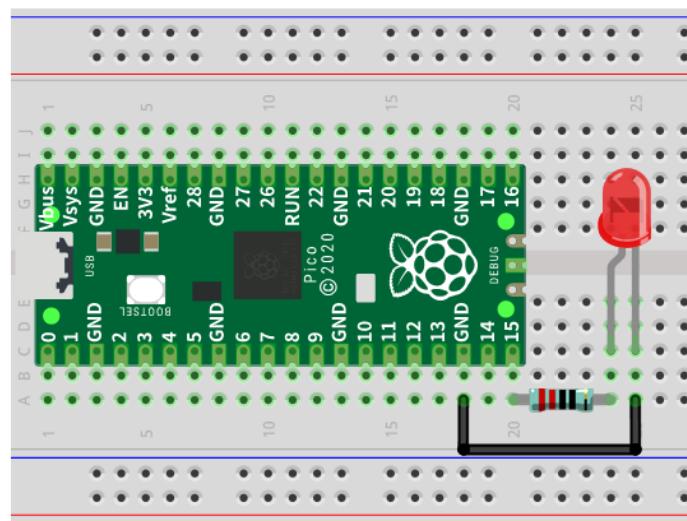
Circuit

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

Schematic diagram



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



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.



Sketch

This project is designed to make PWM output GP15 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

Sketch_04.1_BreathingLight

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

- Title Bar:** Sketch_04.1_BreathingLight | Arduino IDE 2.3.2
- Toolbar:** File, Edit, Sketch, Tools, Help
- Sketch Selection:** Sketch_04.1_BreathingLight.ino
- Code Area:** Contains the following C++ code:

```
7 #define PIN_LED 15 //define the led pin
8
9 void setup() {
10     pinMode(PIN_LED, OUTPUT);
11 }
12
13 void loop() {
14     for (int i = 0; i < 255; i++) { //make light fade in
15         analogWrite(PIN_LED, i);
16         delay(5);
17     }
18     for (int i = 255; i > -1; i--) { //make light fade out
19         analogWrite(PIN_LED, i);
20         delay(5);
21     }
22 }
```
- Output Area:** A large black box indicating no output.
- Status Bar:** Ln 3, Col 69 Raspberry Pi Pico on COM15

Download the code to Pico, and you will see that LED is turned from on to off and then from off to on gradually like breathing.



The following is the program code:

```
1 #define PIN_LED 15 //define the led pin
2
3 void setup() {
4     pinMode(PIN_LED, OUTPUT);
5 }
6
7 void loop() {
8     for (int i = 0; i < 255; i++) { //make light fade in
9         analogWrite(PIN_LED, i);
10        delay(5);
11    }
12    for (int i = 255; i > -1; i--) { //make light fade out
13        analogWrite(PIN_LED, i);
14        delay(5);
15    }
16 }
```

Set the pin controlling LED to output mode.

```
7 pinMode(PIN_LED, OUTPUT);
```

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

```
11 for (int i = 0; i < 255; i++) { //make light fade in
12     analogWrite(PIN_LED, i);
13     delay(5);
14 }
15 for (int i = 255; i > -1; i--) { //make light fade out
16     analogWrite(PIN_LED, i);
17     delay(5);
18 }
```

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

analogWrite(pin, value)

Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255, which represents the duty cycle of 0%-100%.

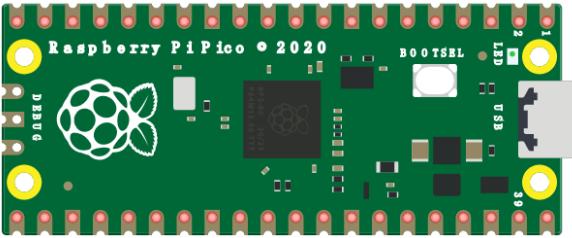
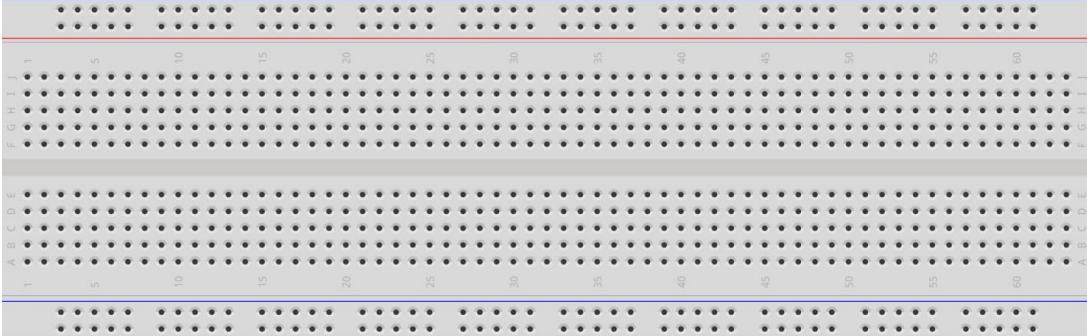
In order to use this function, we need to set the port to output mode.



Project 4.2 Meteor Flowing Light

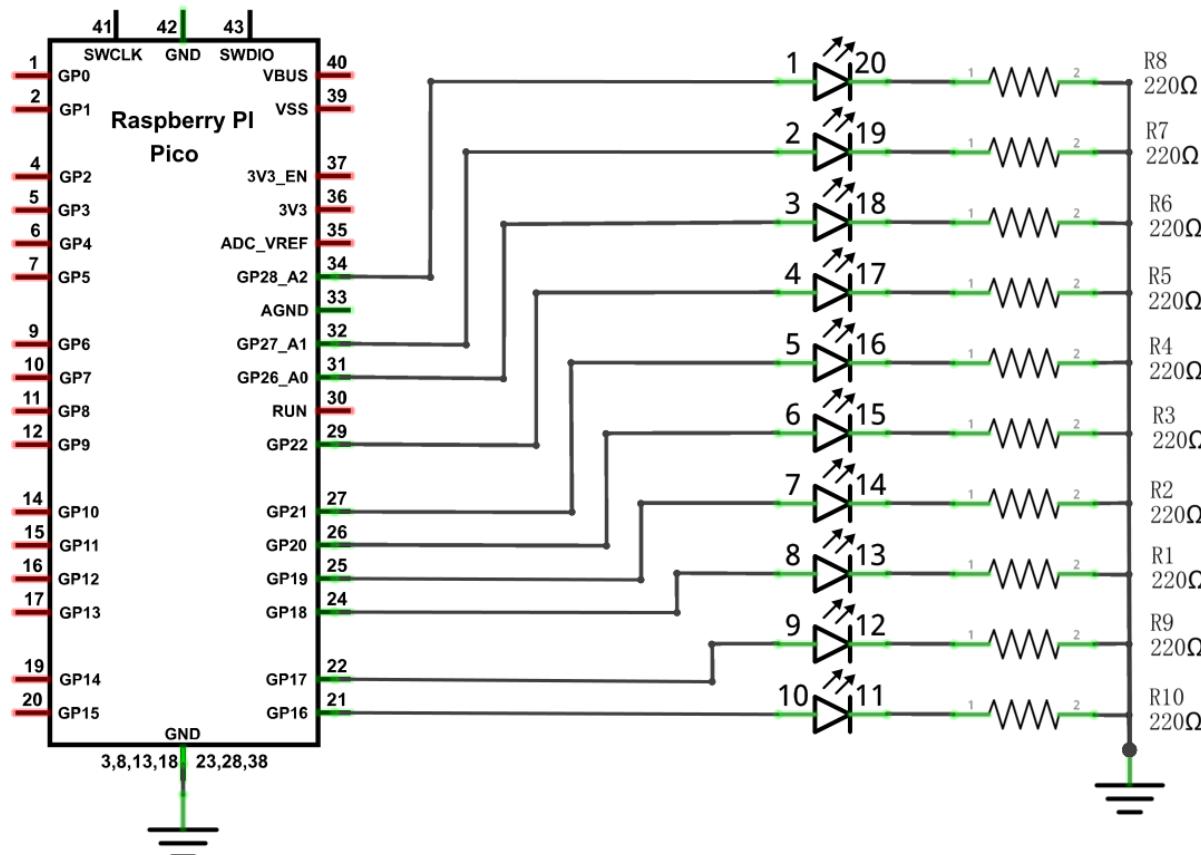
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light. The component list, circuit, and hardware are exactly consistent with the project [Flowing Light](#).

Component List

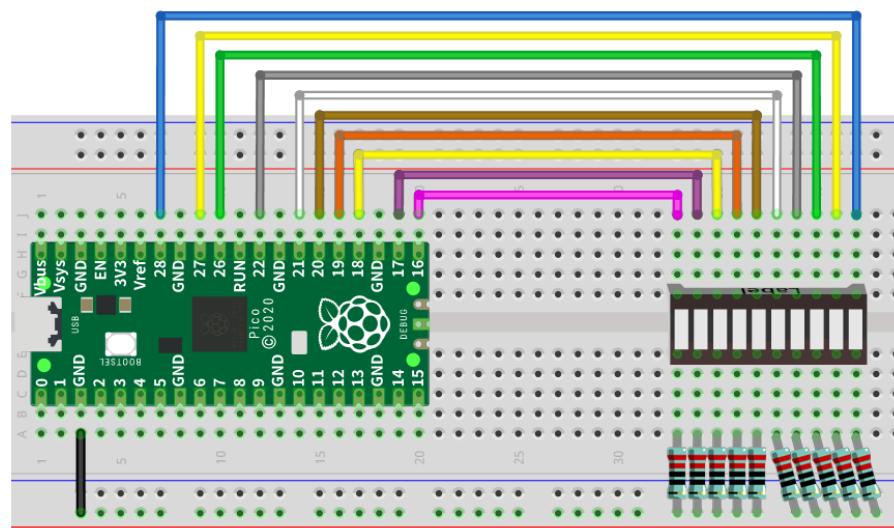
Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Jumper		LED bar graph x1
		Resistor 220Ω x10

Circuit

Schematic diagram



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



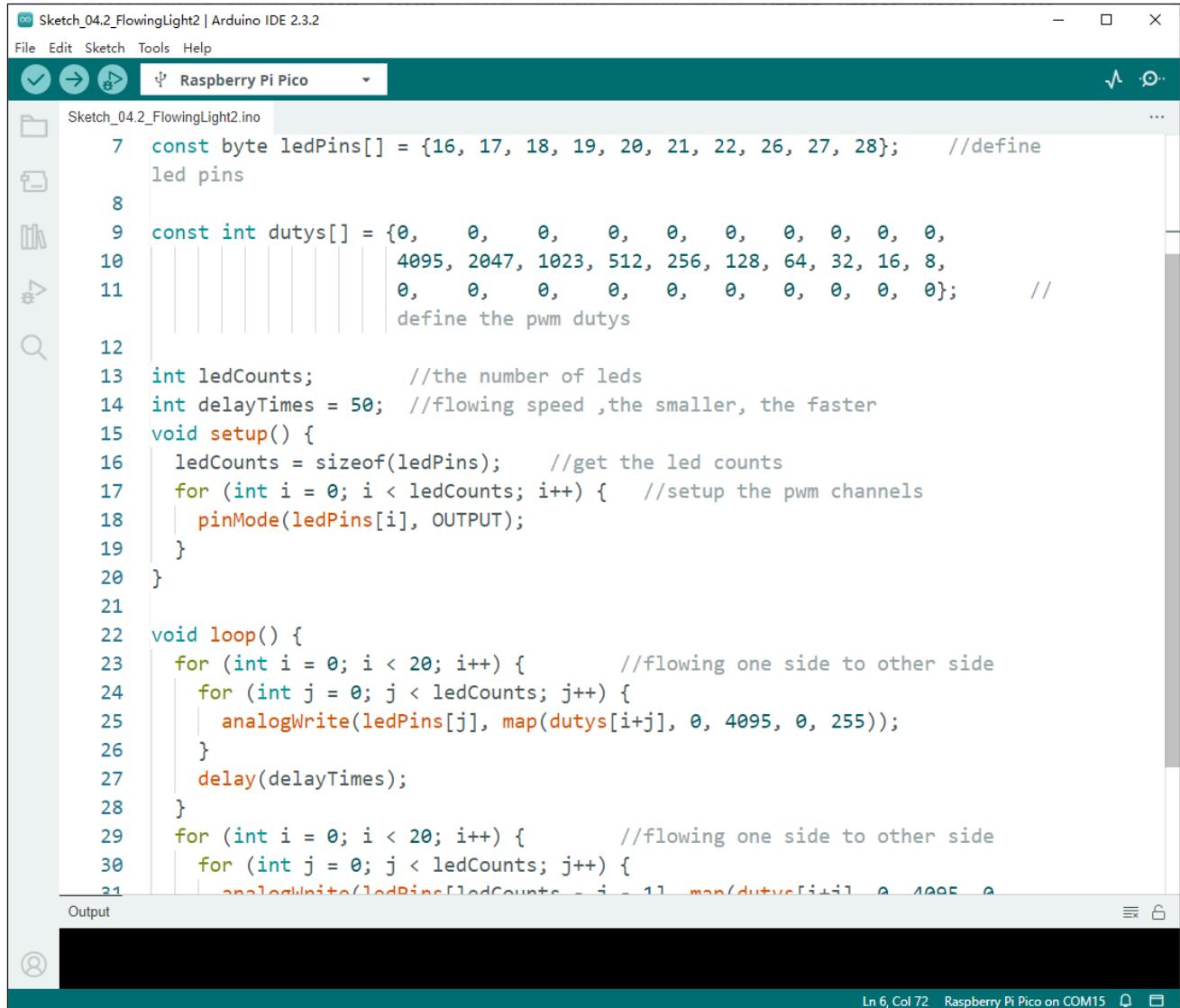
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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

Sketch

Meteor flowing light will be implemented with PWM.

Sketch_04.2_FlowingLight2



```

Sketch_04.2_FlowingLight2 | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_04.2_FlowingLight2.ino
7 const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define
8 led pins
9 const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
10 4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
11 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //define the pwm dutys
12
13 int ledCounts; //the number of leds
14 int delayTimes = 50; //flowing speed ,the smaller, the faster
15 void setup() {
16     ledCounts = sizeof(ledPins); //get the led counts
17     for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
18         pinMode(ledPins[i], OUTPUT);
19     }
20 }
21
22 void loop() {
23     for (int i = 0; i < 20; i++) { //flowing one side to other side
24         for (int j = 0; j < ledCounts; j++) {
25             analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
26         }
27         delay(delayTimes);
28     }
29     for (int i = 0; i < 20; i++) { //flowing one side to other side
30         for (int j = 0; j < ledCounts; j++) {
31             analogWrite(ledPins[ledCounts - i - 1], map(dutys[i+j], 0, 4095, 0,
32
Output
Ln 6, Col 72 Raspberry Pi Pico on COM15

```

Download the code to Pico, and LED bar graph will gradually light up and out from left to right, then back from right to left.

The following is the program code:

1	const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define led pins
2	
3	const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4	4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
5	0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //define the pwm dutys
6	
7	int ledCounts; //the number of leds

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

```

8 int delayTimes = 50; //flowing speed , the smaller, the faster
9 void setup() {
10 ledCounts = sizeof(ledPins); //get the led counts
11 for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
12   pinMode(ledPins[i], OUTPUT);
13 }
14 }
15
16 void loop() {
17   for (int i = 0; i < 20; i++) { //flowing one side to other side
18     for (int j = 0; j < ledCounts; j++) {
19       analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
20     }
21     delay(delayTimes);
22   }
23   for (int i = 0; i < 20; i++) { //flowing one side to other side
24     for (int j = 0; j < ledCounts; j++) {
25       analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
26     }
27     delay(delayTimes);
28   }
29 }
```

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

```

1 const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define led pins
2
3 const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4                     4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
5                     0, 0, 0, 0, 0, 0, 0, 0, 0};//define the pwm dutys
```

Define a variable to store the number of LEDs and another to control the flashing speed of the LED bar.

```

7 int ledCounts; //the number of leds
8 int delayTimes = 50; //flowing speed , the smaller, the faster
```

`Sizeof()` function is used to obtain the number of members of the array `ledPins` and assign it to `ledCount`. Use the 'for' loop to set all pins to output mode.

```

10 ledCounts = sizeof(ledPins); //get the led counts
11 for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
12   pinMode(ledPins[i], OUTPUT);
13 }
```



In loop(), a nested for loop is used to control the pulse width of the PWM, and LED bar graph moves one grid after each 1 is added in the first for loop, gradually changing according to the values in the array duties. As shown in the table below, the value of the second row is the value in the array duties, and the 10 green squares in each row below represent the 10 LEDs on the LED bar graph. Every 1 is added to I , the value of the LED bar graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

0	1	2	3	4	5	7	8	9	1	11	1	1	1	1	1	1	1	2	2	2	2	2	2	3
d	0	0	0	0	0	0	0	0	0	10	5	2	1	6	3	1	8	4	2	0	0	0	0	0
i										23	1	5	2	4	2	6								
0																								
1																								
2																								
3																								
...																								
1																								
8																								
1																								
9																								
2																								
0																								

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

```

17   for (int i = 0; i < 20; i++) {           //flowing one side to other side
18     for (int j = 0; j < ledCounts; j++) {
19       analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
20     }
21     delay(delayTimes);
22   }
23   for (int i = 0; i < 20; i++) {           //flowing one side to other side
24     for (int j = 0; j < ledCounts; j++) {
25       analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
26     }
27     delay(delayTimes);
28   }

```

map(value, fromLow, fromHigh, toLow, toHigh)

This function is used to remap a value, which will return a new value whose percentage in the range of toLow-toHigh is equal to the percentage of "value" in the range of fromLow-fromHigh. For example, 1 is the maximum in the range of 0-1 and the maximum value in the scope of 0-2 is 2, that is, the result value of map (1, 0, 1, 0, 2) is 2.

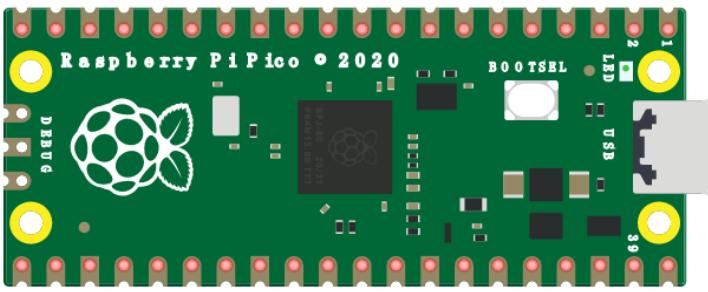
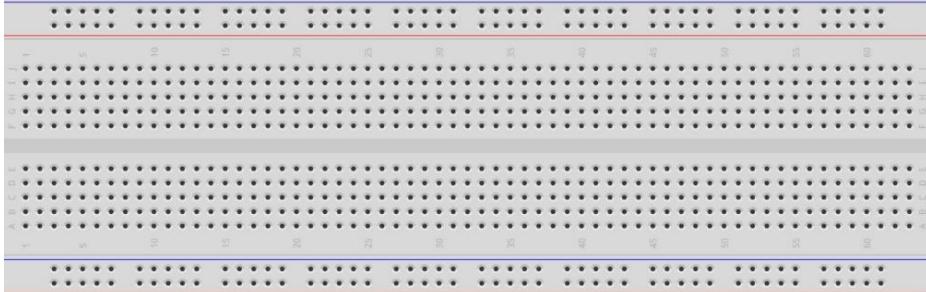
Chapter 5 RGBLED

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

Project 5.1 Random Color Light

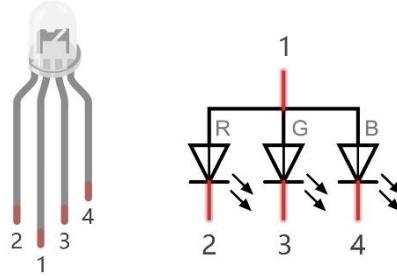
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

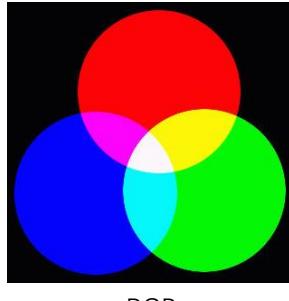
Raspberry Pi Pico x1	USB cable x1	
 A photograph of a Raspberry Pi Pico development board. It is a green printed circuit board with a central Broadcom SoC, labeled "Raspberry Pi Pico • 2020". It features four yellow circular pads on the left side, a USB Type-C port on the right, and various component markings like "BOOTSEL", "I2C", and "USB".		
Breadboard x1		
 A photograph of a standard breadboard with two rows of 40 pins each, labeled A through H along the top and 1 through 36 along the bottom.		
RGBLED x1	Resistor 220Ω x3	Jumper
 A photograph of an RGBLED component, which is a small cylindrical LED with four引脚 (pins).	 A photograph of a resistor component, showing its characteristic red band and color-coded value.	 A photograph of a jumper wire, consisting of two black plastic caps connected by a single continuous wire.

Related Knowledge

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



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

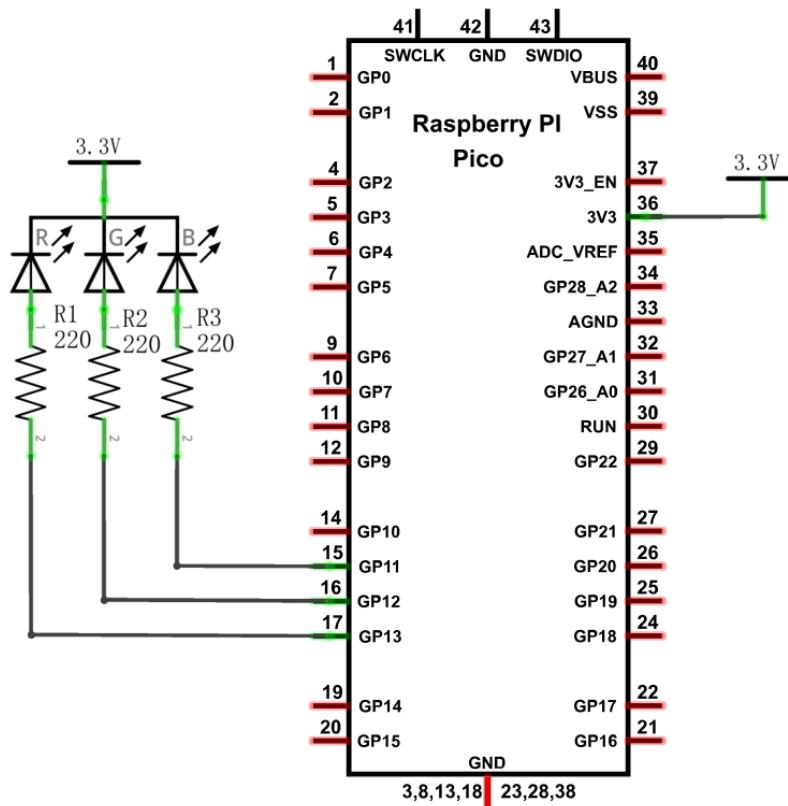


RGB

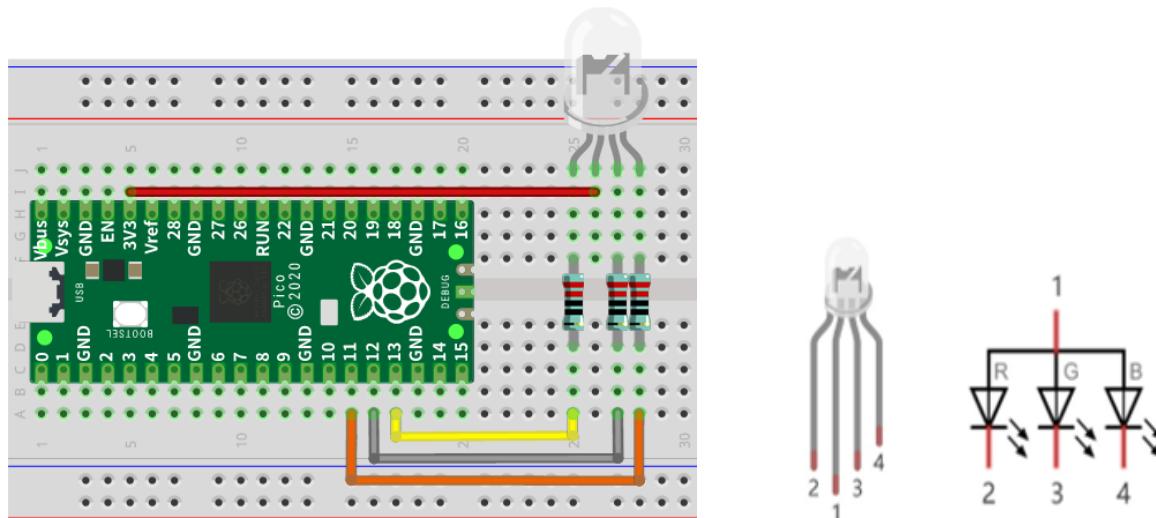
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8 * 2^8 * 2^8 = 16777216$ (16 million) colors through different combinations.

Circuit

Schematic diagram



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

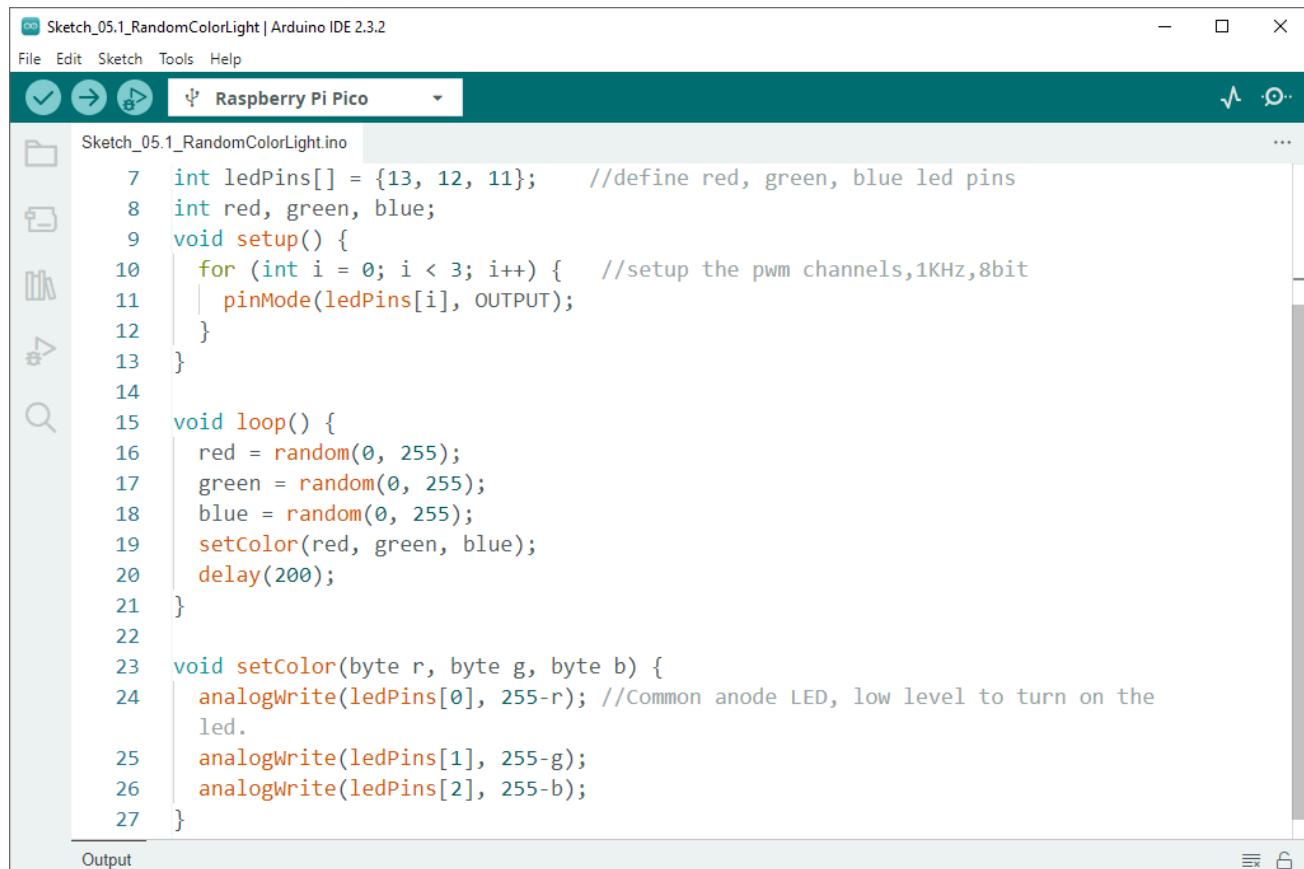


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Sketch

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

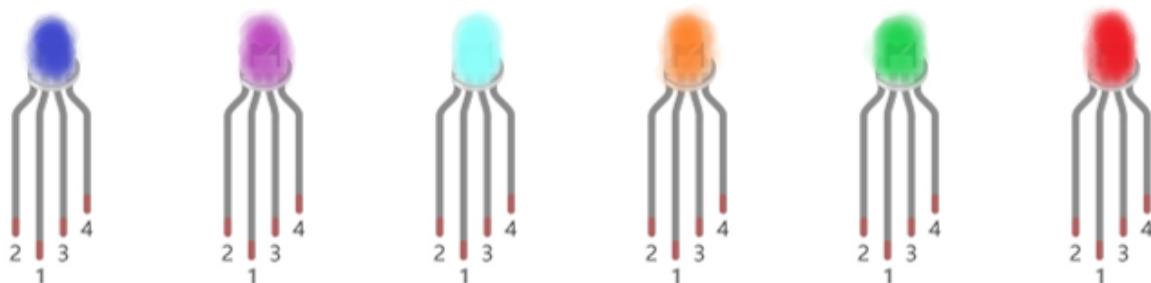
Sketch_05.1_ColorfulLight



```

Sketch_05.1_RandomColorLight | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_05.1_RandomColorLight.ino
1 int ledPins[] = {13, 12, 11};      //define red, green, blue led pins
2 int red, green, blue;
3 void setup() {
4     for (int i = 0; i < 3; i++) {    //setup the pwm channels,1KHz,8bit
5         pinMode(ledPins[i], OUTPUT);
6     }
7 }
8 void loop() {
9     red = random(0, 255);
10    green = random(0, 255);
11    blue = random(0, 255);
12    setColor(red, green, blue);
13    delay(200);
14 }
15 void setColor(byte r, byte g, byte b) {
16     analogWrite(ledPins[0], 255-r); //Common anode LED, low level to turn on the
17     analogWrite(ledPins[1], 255-g);
18     analogWrite(ledPins[2], 255-b);
19 }
20
21 }
22
23
24
25
26
27 }
```

With the code downloaded to Pico, RGB LED begins to display random colors.



The following is the program code:

1	<code>int ledPins[] = {13, 12, 11}; //define red, green, blue led pins</code>
2	<code>int red, green, blue;</code>
3	<code>void setup() {</code>
4	<code> for (int i = 0; i < 3; i++) { //setup the pwm channels,1KHz,8bit</code>
5	<code> pinMode(ledPins[i], OUTPUT);</code>
6	<code> }</code>
7	<code>}</code>
8	

```

9 void loop() {
10    red = random(0, 255);
11    green = random(0, 255);
12    blue = random(0, 255);
13    setColor(red, green, blue);
14    delay(200);
15 }
16
17 void setColor(byte r, byte g, byte b) {
18    analogWrite(ledPins[0], 255-r); //Common anode LED, low level to turn on the led.
19    analogWrite(ledPins[1], 255-g);
20    analogWrite(ledPins[2], 255-b);
21 }
```

Define pins to control RGB LED, and configure them as output mode.

```

1 int ledPins[] = {13, 12, 11};      //define red, green, blue led pins
2 int red, green, blue;
3 void setup() {
4     for (int i = 0; i < 3; i++) {    //setup the pwm channels, 1KHz, 8bit
5         pinMode(ledPins[i], OUTPUT);
6     }
7 }
```

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

```

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

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

```

12 red = random(0, 255);
13 green = random(0, 255);
14 blue = random(0, 255);
15 setColor(red, green, blue);
16 delay(200);
```

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

long random(min, max);

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



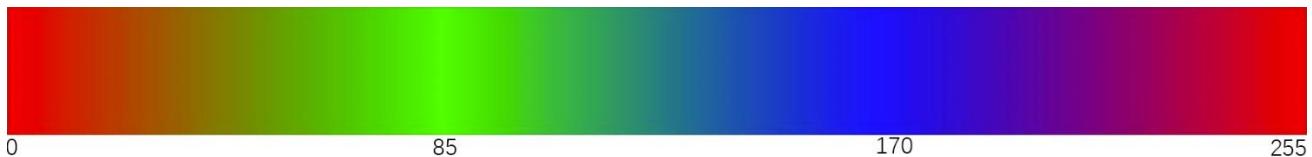
Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff.

This project will realize a fashionable Light with soft color changes.

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

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



Sketch

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

[Sketch_05.2_SoftColorfulLight](#)

The following is the program code:

```

1 const byte ledPins[] = {13, 12, 11};      //define led pins
2 void setup() {
3     for (int i = 0; i < 3; i++) {    //setup the pwm channels
4         pinMode(ledPins[i], OUTPUT);
5     }
6 }
7
8 void loop() {
9     for (int i = 0; i < 256; i++) {
10        setColor(wheel(i));
11        delay(100);
12    }
13 }
14
15 void setColor(long rgb) {
16     analogWrite(ledPins[0], 255 - (rgb >> 16) & 0xFF);
17     analogWrite(ledPins[1], 255 - (rgb >> 8) & 0xFF);
18     analogWrite(ledPins[2], 255 - (rgb >> 0) & 0xFF);
19 }
20
21 long wheel(int pos) {
22     long WheelPos = pos % 0xff;
23     if (WheelPos < 85) {
24         return ((255 - WheelPos * 3) << 16) | ((WheelPos * 3) << 8);
25     } else if (WheelPos < 170) {

```

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

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

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

```
15 void setColor(long rgb) {
16     ledcWrite(chns[0], 255 - (rgb >> 16) & 0xFF);
17     ledcWrite(chns[1], 255 - (rgb >> 8) & 0xFF);
18     ledcWrite(chns[2], 255 - (rgb >> 0) & 0xFF);
19 }
```

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

```
21 long wheel(int pos) {
22     long WheelPos = pos % 0xff;
23     if (WheelPos < 85) {
24         return (((255 - WheelPos * 3) << 16) | ((WheelPos * 3) << 8));
25     } else if (WheelPos < 170) {
26         WheelPos -= 85;
27         return (((255 - WheelPos * 3) << 8) | (WheelPos * 3));
28     } else {
29         WheelPos -= 170;
30         return ((WheelPos * 3) << 16 | (255 - WheelPos * 3));
31     }
32 }
```

Chapter 6 NeoPixel

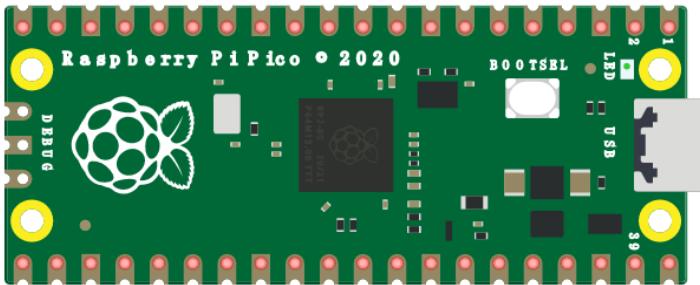
This chapter will help you learn to use a more convenient RGBLED lamp, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 6.1 NeoPixel

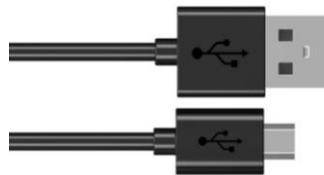
Learn the basic usage of NeoPixel and use it to blink red, green, blue and white.

Component List

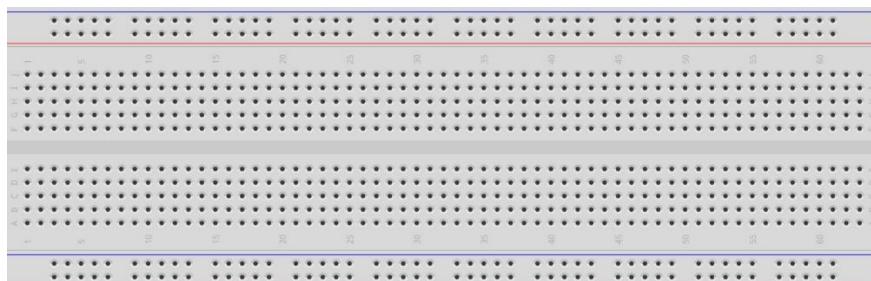
Raspberry Pi Pico x1



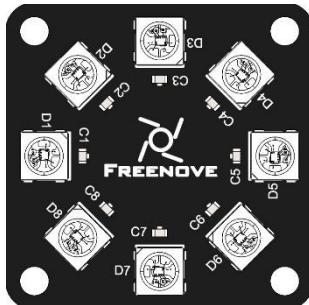
USB cable x1



Breadboard x1



Freenove 8 RGB LED Module x1



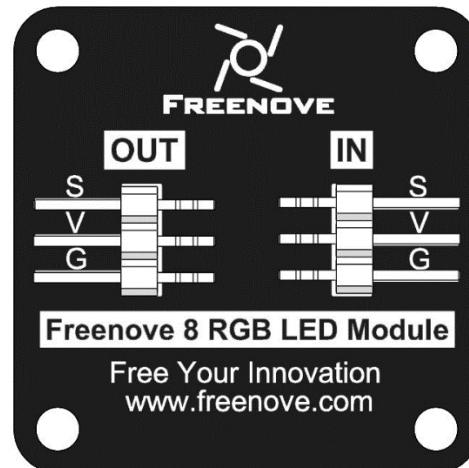
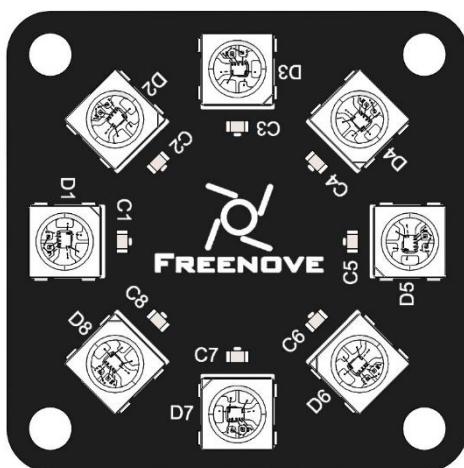
Jumper



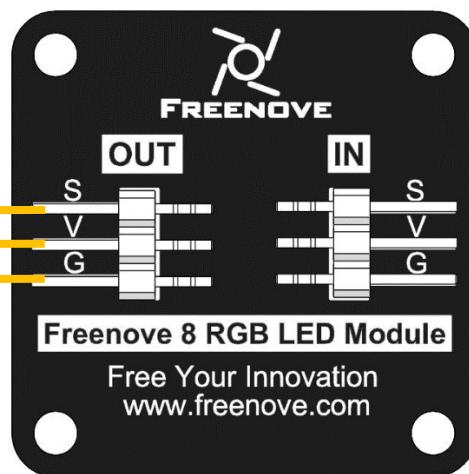
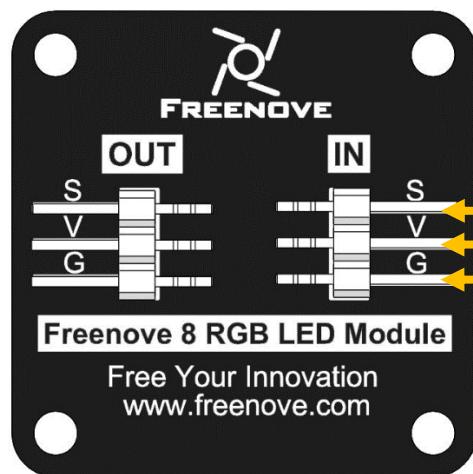
Related Knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control eight LEDs on the module. As shown below:



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

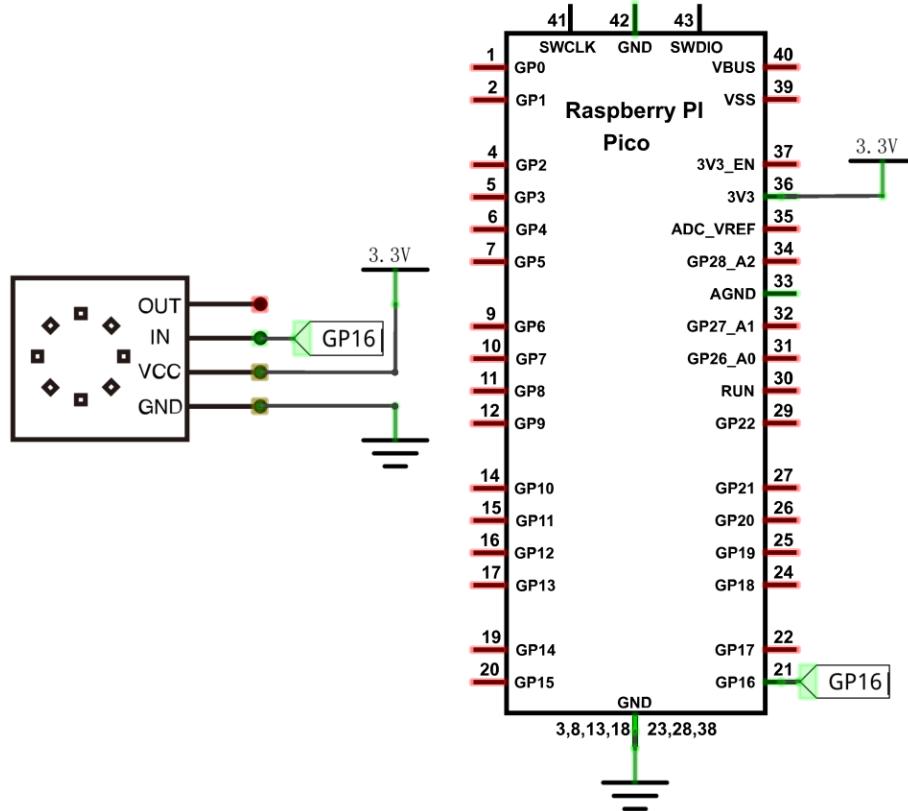


Pin description:

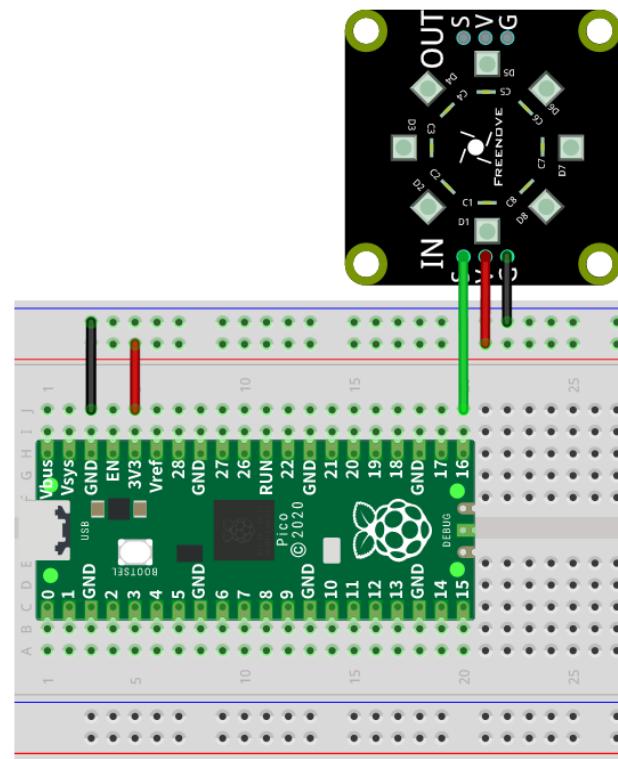
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.3V~5.5V	V	Power supply pin, +3.3V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



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



Any concerns? ✉ support@freenove.com

Sketch

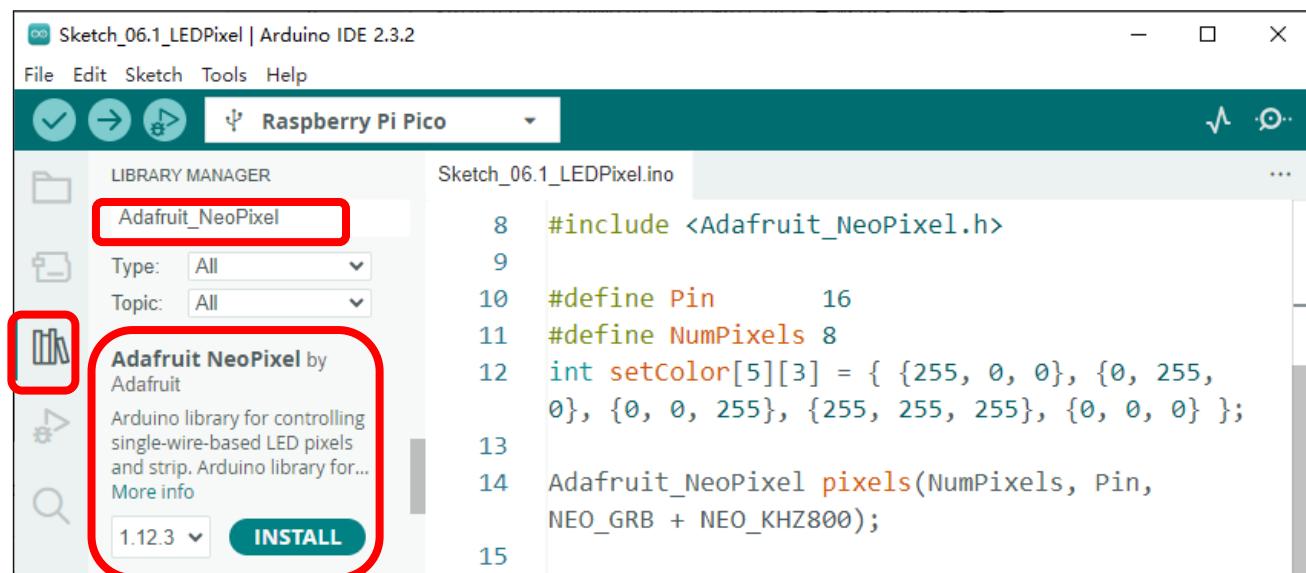
This code uses a library named "**Adafruit_NeoPixel**". If you have not installed it, please do so first.

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

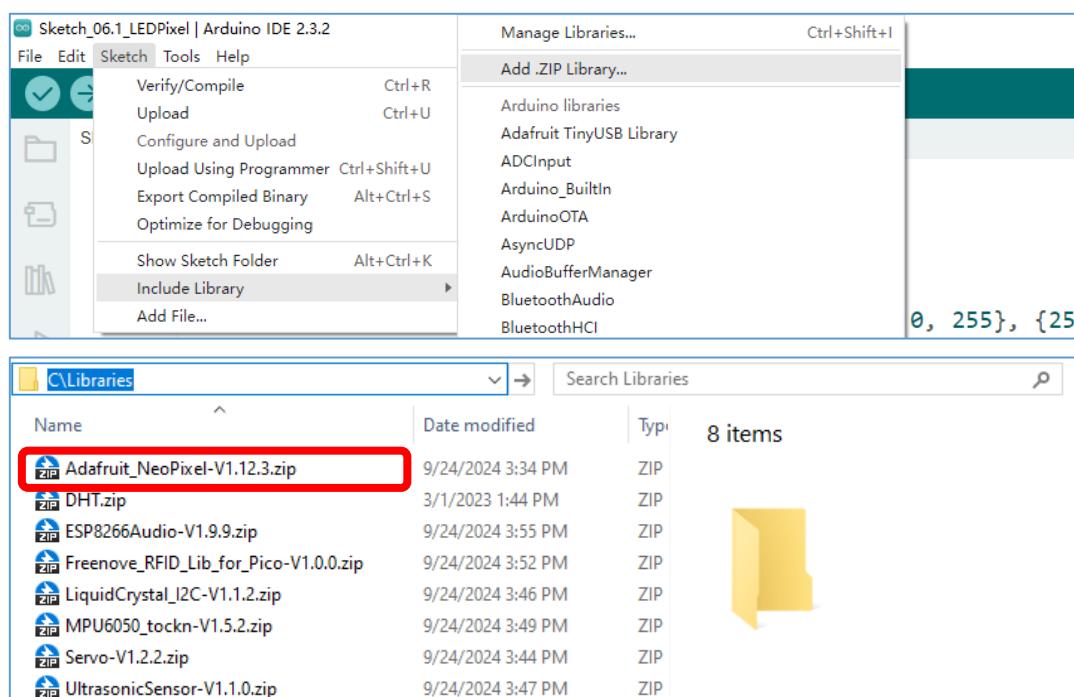
How to install the library

There are two ways to add libraries.

The first way, open Arduino IDE, click **Library Manager** on the left, input "**Adafruit_NeoPixel**" in the search bar and select "**Adafruit_NeoPixel**" to install, as shown below.



The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named ".Libraries/ **Adafruit_NeoPixel-V1.12.3.Zip**" which locates in this directory, and click OPEN.





Sketch_06.1_LEDPixel

Sketch_06.1_LEDPixel | Arduino IDE 2.3.2

File Edit Sketch Tools Help

Raspberry Pi Pico

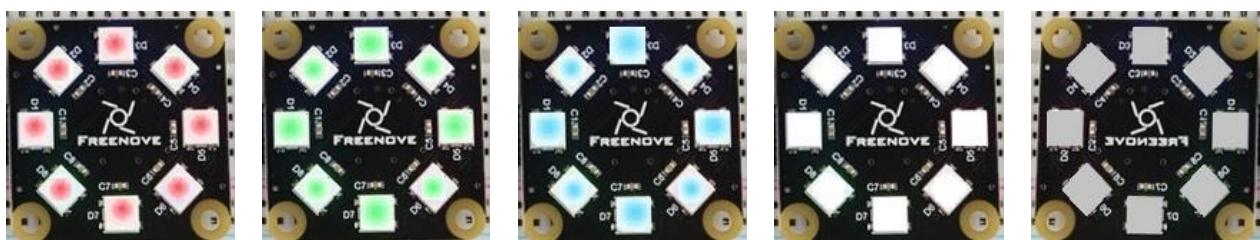
```

10 #define Pin      16
11 #define NumPixels 8
12 int setColor[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255},
13   {0, 0, 0} };
14 Adafruit_NeoPixel pixels(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
15
16 void setup() {
17   pixels.begin();
18   pixels.setBrightness(20);
19 }
20
21 void loop() {
22   for (int j = 0; j < 5; j++) {
23     for (int i = 0; i < NumPixels; i++) {
24       pixels.setPixelColor(i, setColor[j][0], setColor[j][1], setColor[j][2]);
25       pixels.show();
26       delay(100);
27     }
28     delay(500);
29   }
30 }
```

Output

Ln 5, Col 33 Raspberry Pi Pico on COM15 4 3

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



The following is the program code:

```

1 #include <Adafruit_NeoPixel.h>
2
3 #define Pin      16
4 #define NumPixels 8
5 int setColor[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255}, {0, 0, 0} };
6
7 Adafruit_NeoPixel pixels(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
8
9 void setup() {
10     pixels.begin();
11     pixels.setBrightness(20);
12 }
13
14 void loop() {
15     for (int j = 0; j < 5; j++) {
16         for (int i = 0; i < NumPixels; i++) {
17             pixels.setPixelColor(i, setColor[j][0], setColor[j][1], setColor[j][2]);
18             pixels.show();
19             delay(100);
20         }
21         delay(500);
22     }
23 }
```

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

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

Define the pins connected to the ring, the number of LEDs on the ring.

```
3 #define Pin      16
4 #define NumPixels 8
```

Apply for an object that controls the RGB LED ring, and assign the number of LEDs, the number of pins that control the LEDs, and the control mode of the LEDs to the object.

```
7 Adafruit_NeoPixel pixels(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
```

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

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

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

```
10     pixels.begin();
11     pixels.setBrightness(20);
```

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

```
15     for (int j = 0; j < 5; j++) {
16         for (int i = 0; i < NumPixels; i++) {
17             pixels.setPixelColor(i, setColor[j][0], setColor[j][1], setColor[j][2]);
18             pixels.show();
```

```

19     delay(100);
20 }
21     delay(500);
22 }
```

Reference

Adafruit_NeoPixel(uint16_t n, int16_t pin = 6, neoPixelType type = NEO_GRB + NEO_KHZ800)

Constructor to create a NeoPixel object.

Before each use of the constructor, please add “[Adafruit_NeoPixel.h](#)”

Parameters

n: The number of led.

pin_gpio: A pin connected to an LED.

type: Types of LED.

NEO_RGB: The sequence of NeoPixel module loading color is red, green and blue.

NEO_RBG: The sequence of NeoPixel module loading color is red, blue and green.

NEO_GRB: The sequence of NeoPixel module loading color is green, red and blue.

TYPE_GBR: The sequence of NeoPixel module loading color is green, blue and red.

NEO_BRG: The sequence of NeoPixel module loading color is blue, red and green.

NEO_BGR: The sequence of NeoPixel module loading color is blue, green and red.

void begin(void);

Initialize the NeoPixel object

void setPixelColor (u8 index, u8 r, u8 g, u8 b);

void setPixelColor (u8 index, u32 rgb);

void setPixelColor (u8 index, u8 r, u8 g, u8 b, u8 w);

Set the color of LED with order number n.

void show(void);

Send the color data to the led and display the set color immediately.

void setBrightness(uint8_t);

Set the brightness of the LED.

If you want to learn more about this library, you can visit the following website:

https://github.com/adafruit/Adafruit_NeoPixel

Project 6.2 Rainbow Light

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

Sketch

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



Sketch_06.2_RainbowLight

```
#include <Adafruit_NeoPixel.h>
#define Pin 16
#define NumPixels 8
int red = 0;
int green = 0;
int blue = 0;
Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
void setup() {
    strip.begin();
    strip.setBrightness(20);
}
void loop() {
    for (int j = 0; j < 256 * 5; j++) {
        for (int i = 0; i < 8; i++) {
            Wheel(((i * 256 / 8) + j)%255);
            strip.setPixelColor(i, strip.Color(red, green, blue));
        }
        strip.show();
        delay(10);
    }
}
```

The screenshot shows the Arduino IDE interface with the sketch file 'Sketch_06.2_RainbowLight.ino' open. The code uses the Adafruit_NeoPixel library to control an 8-pixel NeoPixel strip connected to pin 16. The setup function initializes the strip and sets the brightness to 20. The loop function iterates 256 times (5 full cycles of 512 steps each), with each iteration consisting of 8 nested loops (one for each pixel). Inside these loops, a 'Wheel' color palette is used to determine the color values for red, green, and blue, which are then set using the strip.setPixelColor method. A delay of 10ms is added between each iteration of the outer loop.



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



The following is the program code:

```
1 #include <Adafruit_NeoPixel.h>
2
3 #define Pin      16
4 #define NumPixels 8
5 int red = 0;
6 int green = 0;
7 int blue = 0;
8 Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
9
10 void setup() {
11     strip.begin();
12     strip.setBrightness(20);
13 }
14
15 void loop() {
16     for (int j = 0; j < 256 * 5; j++) {
17         for (int i = 0; i < 8; i++) {
18             Wheel(((i * 256 / 8) + j)%255);
19             strip.setPixelColor(i, strip.Color(red, green, blue));
20         }
21         strip.show();
22         delay(10);
23     }
24 }
25
26 void Wheel(byte WheelPos) {
27     WheelPos = 255 - WheelPos;
28     if (WheelPos < 85) {
29         red = 255 - WheelPos * 3;
30         green = 0;
31         blue = WheelPos * 3;
32     }
33     else if (WheelPos < 170) {
34         WheelPos -= 85;
35         red = 0;
```

```
36     green = WheelPos * 3;
37     blue = 255 - WheelPos * 3;
38 }
39 else {
40     WheelPos -= 170;
41     red = WheelPos * 3;
42     green = 255 - WheelPos * 3;
43     blue = 0;
44 }
45 }
```

In the loop(), two “for” loops are used, the internal “for” loop(for-i) is used to set the color of each LED, and the external “for” loop(for-i) is used to change the color, in which the self-increment value in i+=1 can be changed to change the color step distance. Changing the delay parameter changes the speed of the color change. Wheel(((i * 256 / 8) + j)%255) will take color from the color model at equal intervals starting from i.

```
16 for (int j = 0; j < 256 * 5; j++) {
17     for (int i = 0; i < 8; i++) {
18         Wheel(((i * 256 / 8) + j)%255);
19         strip.setPixelColor(i, strip.Color(red, green, blue));
20     }
21     strip.show();
22     delay(10);
23 }
```



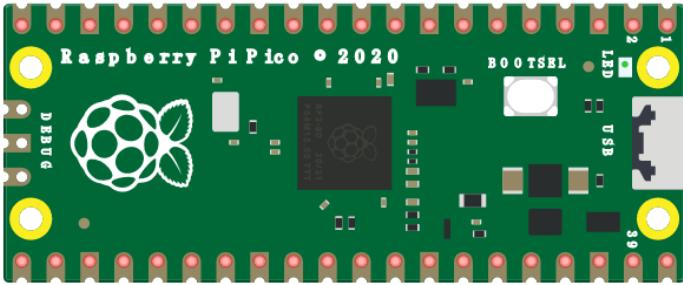
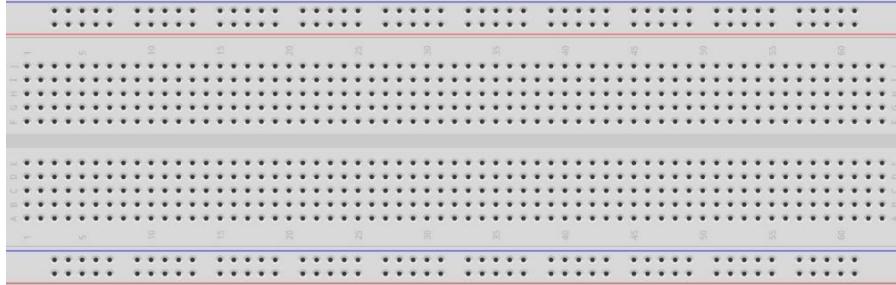
Chapter 7 Buzzer

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

Project 7.1 Doorbell

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

Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Jumper		
NPN transistor x1 (S8050)		Active buzzer x1
		Push button x1
		Resistor 1kΩ x1
		Resistor 10kΩ x2

Any concerns?  support@freenove.com

Component Knowledge

Buzzer

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

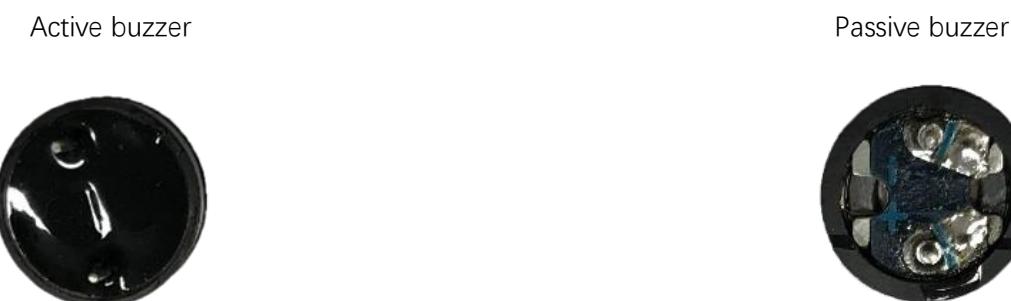


Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2 kHz, which means the passive buzzer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

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

How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes viewing of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model).



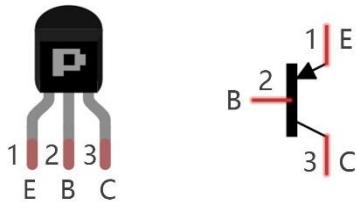
Transistor

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

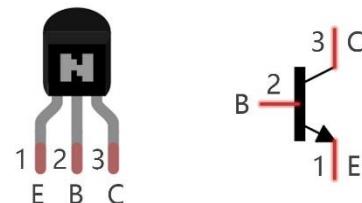
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

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

PNP transistor



NPN transistor

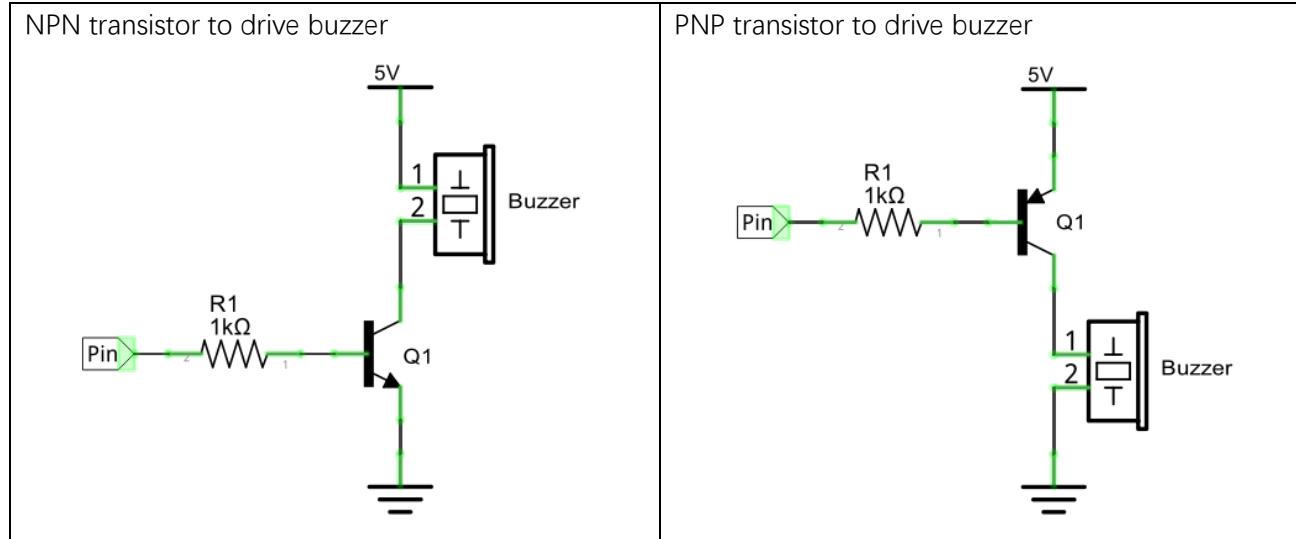


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

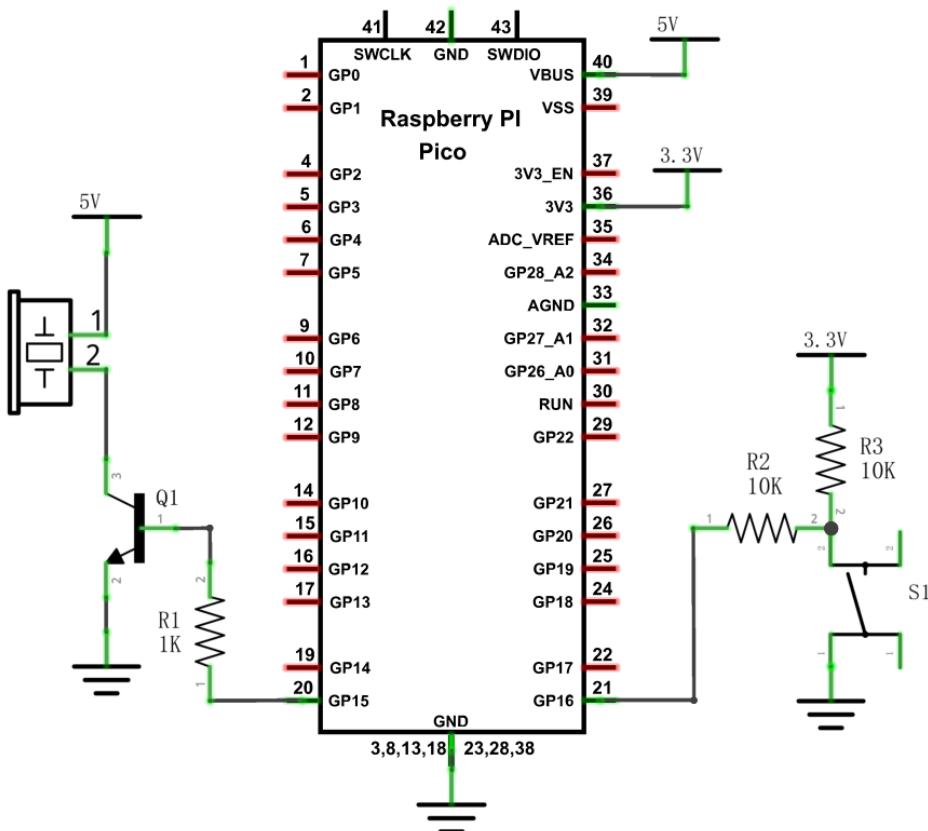
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will be conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will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

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

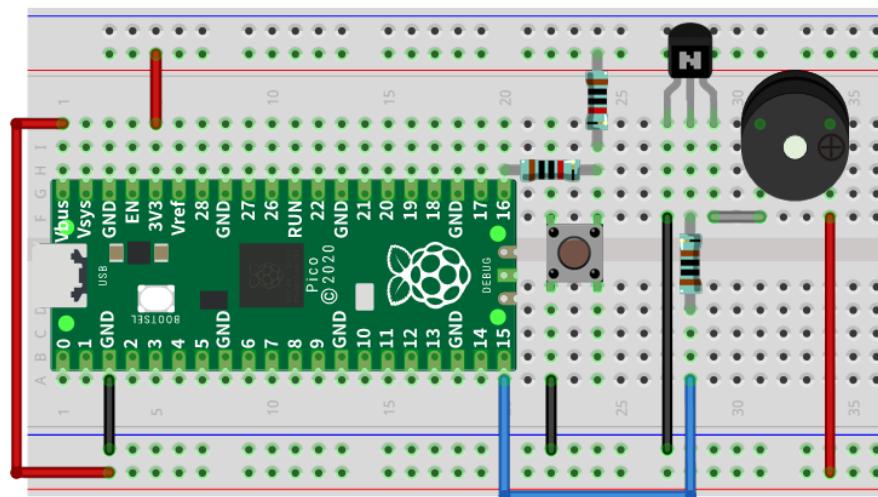


Circuit

Schematic diagram



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



Note:

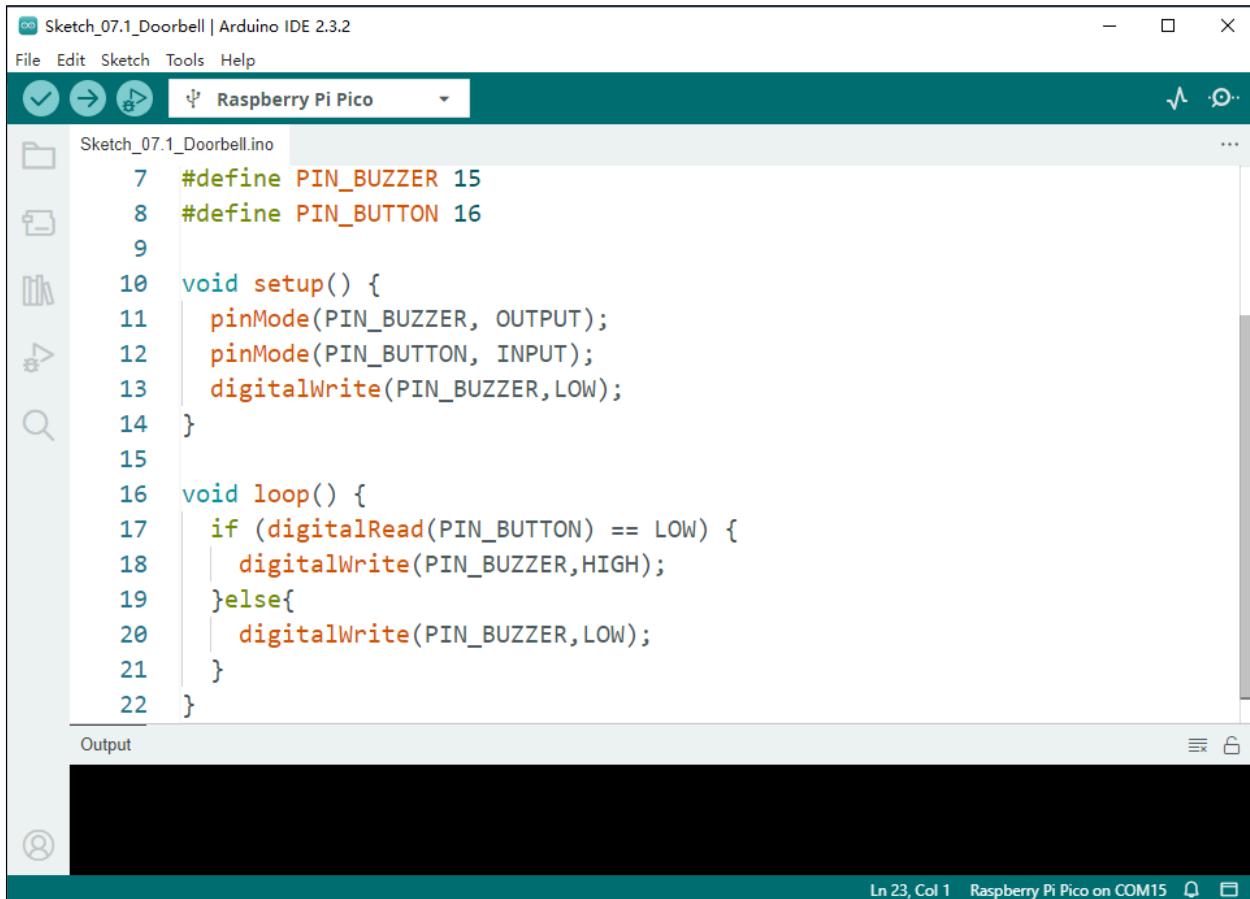
1. In this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.
 2. VBUS should be connect to the positive end of USB cable. If it connects to GND, it may burn the computer or Raspberry Pi Pico. Similarly, please be careful when wiring pins 36-40 of Pico to avoid short circuit.

Any concerns? support@freenove.com

Sketch

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

Sketch_07.1_Doorbell



```

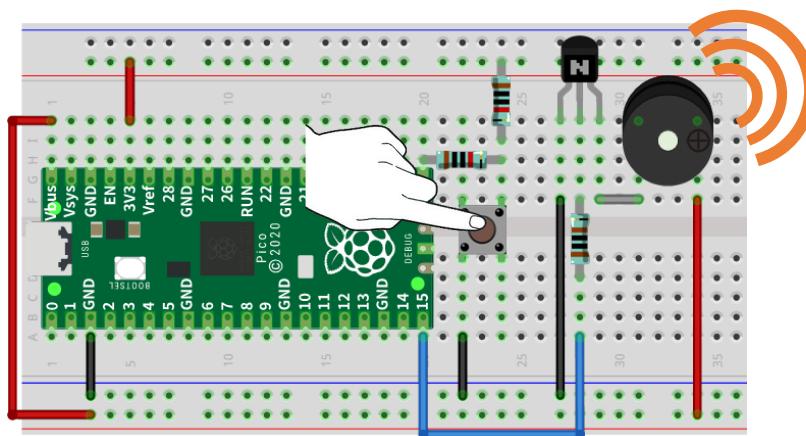
Sketch_07.1_Doorbell | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_07.1_Doorbell.ino
7 #define PIN_BUZZER 15
8 #define PIN_BUTTON 16
9
10 void setup() {
11     pinMode(PIN_BUZZER, OUTPUT);
12     pinMode(PIN_BUTTON, INPUT);
13     digitalWrite(PIN_BUZZER,LOW);
14 }
15
16 void loop() {
17     if (digitalRead(PIN_BUTTON) == LOW) {
18         digitalWrite(PIN_BUZZER,HIGH);
19     }else{
20         digitalWrite(PIN_BUZZER,LOW);
21     }
22 }

```

Output

Ln 23, Col 1 Raspberry Pi Pico on COM15

Download the code to Pico, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.



The following is the program code:

```
1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
3
4 void setup() {
5     pinMode(PIN_BUZZER, OUTPUT);
6     pinMode(PIN_BUTTON, INPUT);
7     digitalWrite(PIN_BUZZER, LOW);
8 }
9
10 void loop() {
11     if (digitalRead(PIN_BUTTON) == LOW) {
12         digitalWrite(PIN_BUZZER, HIGH);
13     } else{
14         digitalWrite(PIN_BUZZER, LOW);
15     }
16 }
```

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

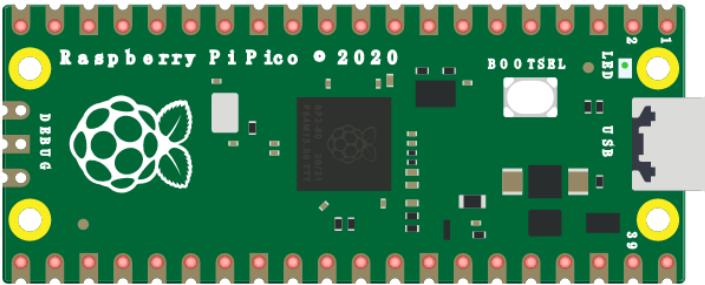
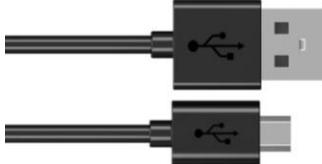
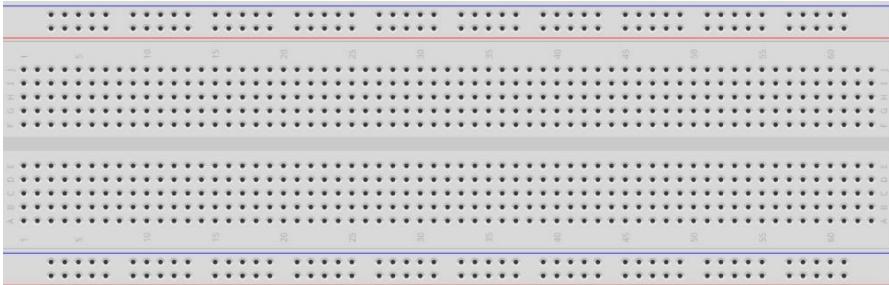


Project 7.2 Alertor

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

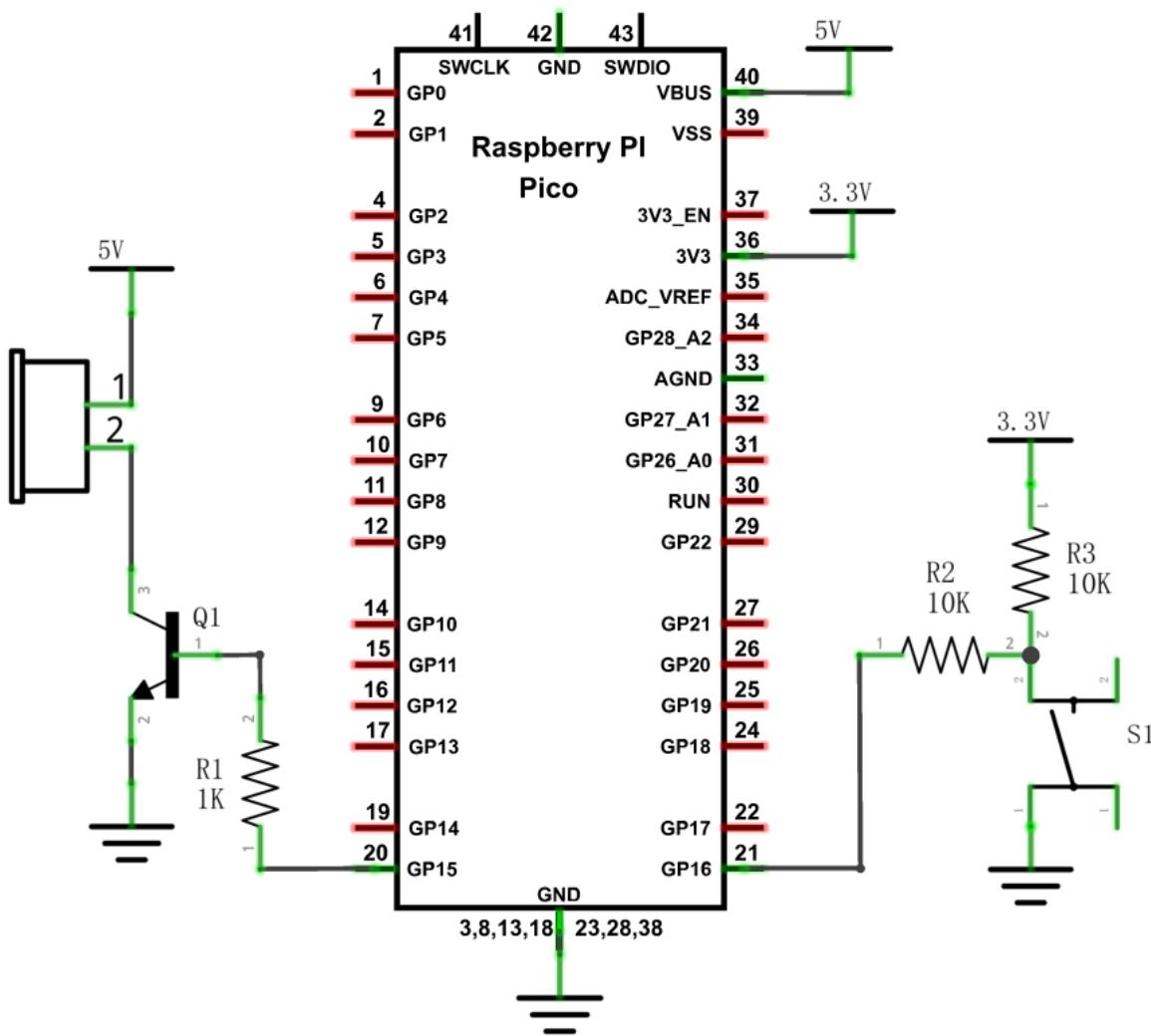
Component list and the circuit part is similar to last section, only the **active buzzer** needs to be **replaced** with a **passive buzzer** for this project.

Component List

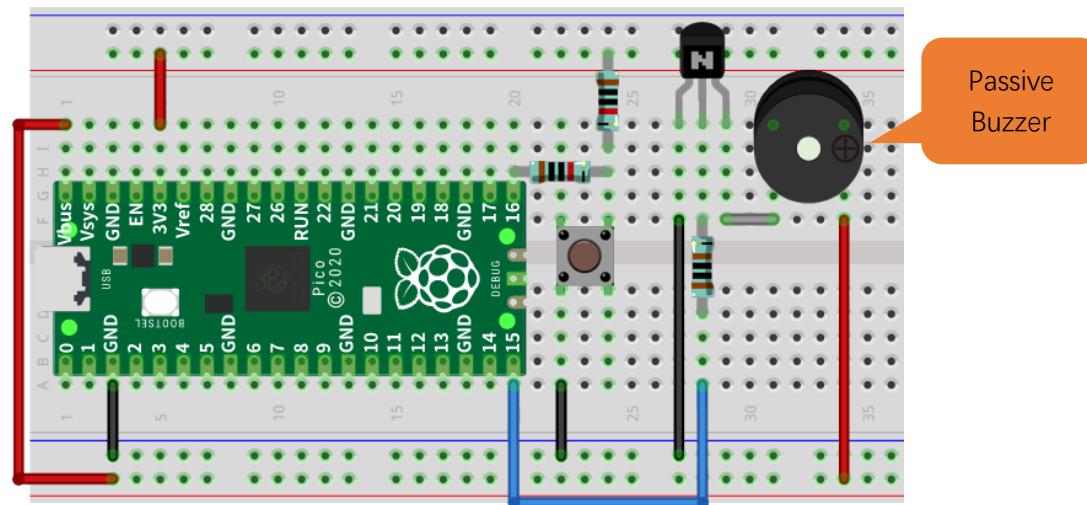
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper			
NPN transistorx1 (S8050)		Passive buzzer x1	
Push button x1		Resistor 1kΩ x1	
		Resistor 10kΩ x2	

Circuit

Schematic diagram



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

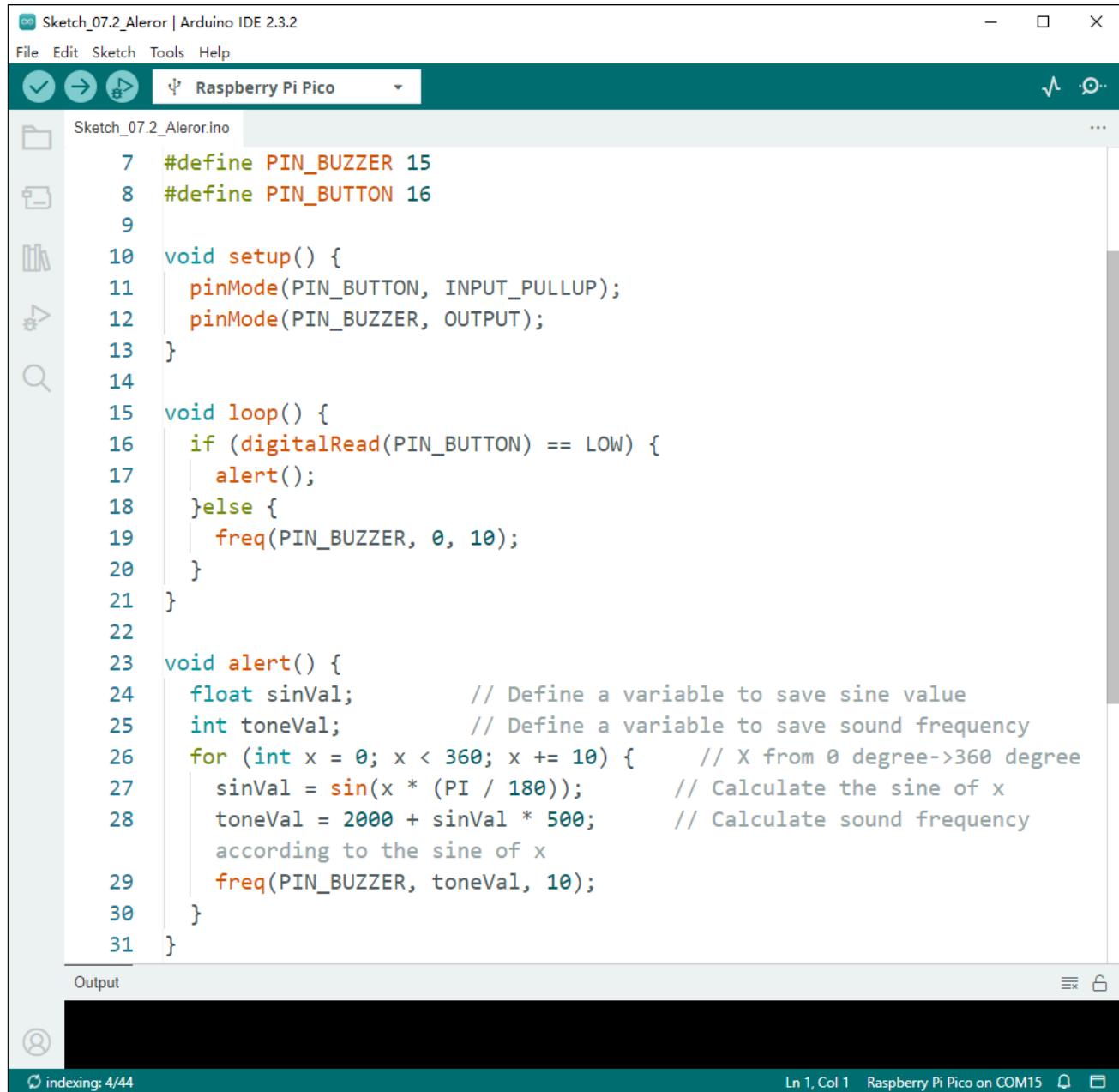


Any concerns? support@freenove.com

Sketch

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

Sketch_07.2_Alertor

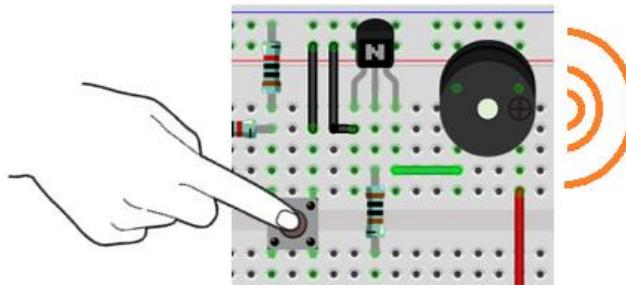


The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch_07.2_Aleror | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and a dropdown set to "Raspberry Pi Pico".
- Code Editor:** Displays the `Sketch_07.2_Aleror.ino` file content. The code uses `#define` statements to map pins to constants. It defines `PIN_BUZZER` to 15 and `PIN_BUTTON` to 16. The `setup()` function initializes the button as an input with pull-up and the buzzer as an output. The `loop()` function checks if the button is pressed. If it is, it calls the `alert()` function. Otherwise, it plays a sine wave at a frequency calculated based on the sine of the current angle (x) from 0 to 360 degrees. The `freq()` function is used to set the PWM frequency.

```
Sketch_07.2_Aleror.ino
7 #define PIN_BUZZER 15
8 #define PIN_BUTTON 16
9
10 void setup() {
11     pinMode(PIN_BUTTON, INPUT_PULLUP);
12     pinMode(PIN_BUZZER, OUTPUT);
13 }
14
15 void loop() {
16     if (digitalRead(PIN_BUTTON) == LOW) {
17         alert();
18     }else {
19         freq(PIN_BUZZER, 0, 10);
20     }
21 }
22
23 void alert() {
24     float sinVal;          // Define a variable to save sine value
25     int toneVal;           // Define a variable to save sound frequency
26     for (int x = 0; x < 360; x += 10) {    // X from 0 degree->360 degree
27         sinVal = sin(x * (PI / 180));      // Calculate the sine of x
28         toneVal = 2000 + sinVal * 500;       // Calculate sound frequency
29         freq(PIN_BUZZER, toneVal, 10);
30     }
31 }
```
- Output Panel:** Shows the status message "indexing: 4/44" and "Ln 1, Col 1 Raspberry Pi Pico on COM15".

Download the code to Pico, press the button, and then alarm sounds; when the button is released, the alarm will stop sounding.



The following is the program code:

```
1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
3
4 void setup() {
5     pinMode(PIN_BUTTON, INPUT_PULLUP);
6     pinMode(PIN_BUZZER, OUTPUT);
7 }
8
9 void loop() {
10    if (digitalRead(PIN_BUTTON) == LOW) {
11        alert();
12    }else {
13        freq(PIN_BUZZER, 0, 10);
14    }
15 }
16
17 void alert() {
18     float sinVal;          // Define a variable to save sine value
19     int toneVal;           // Define a variable to save sound frequency
20     for (int x = 0; x < 360; x += 10) { // X from 0 degree->360 degree
21         sinVal = sin(x * (PI / 180));      // Calculate the sine of x
22         toneVal = 2000 + sinVal * 500;      // Calculate sound frequency according to the sine of x
23         freq(PIN_BUZZER, toneVal, 10);
24     }
25 }
26
27 void freq(int PIN, int freqs, int times) {
28     if (freqs == 0) {
29         digitalWrite(PIN, LOW);
30     }
31     else {
32         for (int i = 0; i < times * freqs / 1000; i++) {
33             digitalWrite(PIN, HIGH);
```

```

34     delayMicroseconds(1000000 / freqs / 2);
35     digitalWrite(PIN, LOW);
36     delayMicroseconds(1000000 / freqs / 2);
37   }
38 }
39 }
```

Define the button and pin to control the passive buzzer.

```

1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
```

Write a function to drive the passive buzzer with a duty cycle of 50%. The `delayMicroseconds()` function is in

1us. $1\text{ s} = 1000000\text{ us}$. By the formula $T = \frac{1}{f}$, when the frequency is fixed, the PWM period T is also fixed.

```

27 void freq(int PIN, int freqs, int times) {
28   if (freqs == 0) {
29     digitalWrite(PIN, LOW);
30   }
31   else {
32     for (int i = 0; i < times * freqs / 1000; i++) {
33       digitalWrite(PIN, HIGH);
34       delayMicroseconds(1000000 / freqs / 2);
35       digitalWrite(PIN, LOW);
36       delayMicroseconds(1000000 / freqs / 2);
37     }
38   }
39 }
```

The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (here it is 500) and plus the resonant frequency of buzzer.

```

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

In the `loop()` function, when the button is pressed, subfunction `alert()` will be called and the alertor will issue a warning sound; otherwise, it stops the buzzer.

```

10 if (digitalRead(PIN_BUTTON) == LOW) {
11   alert();
12 }else {
13   freq(PIN_BUZZER, 0, 10);
14 }
```

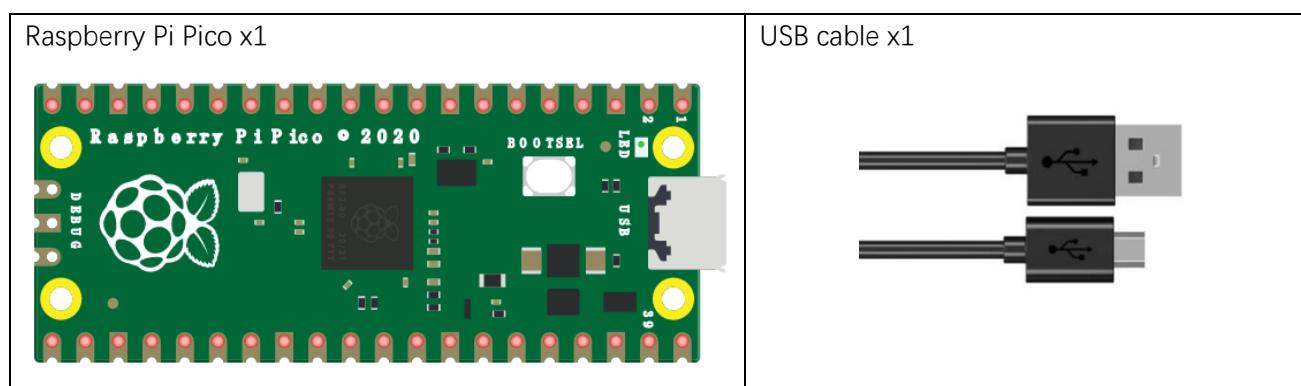
Chapter 8 Serial Communication

Serial Communication is a means of Communication between different devices. This section describes Raspberry Pi Pico Serial Communication.

Project 8.1 Serial Print

This project uses Raspberry Pi Pico serial communicator to send data to the computer and print it on the serial monitor.

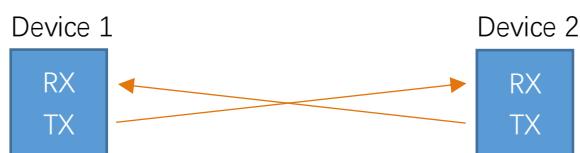
Component List



Related Knowledge

Serial communication

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



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

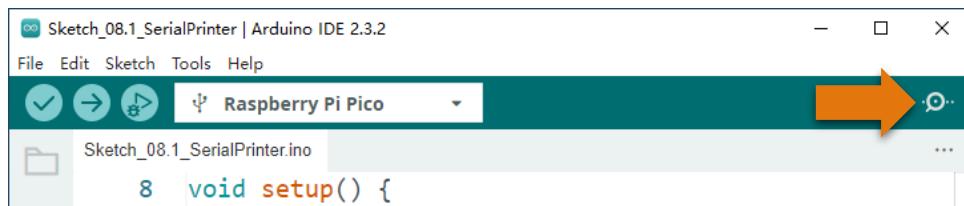
Serial port on Raspberry Pi Pico

Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.

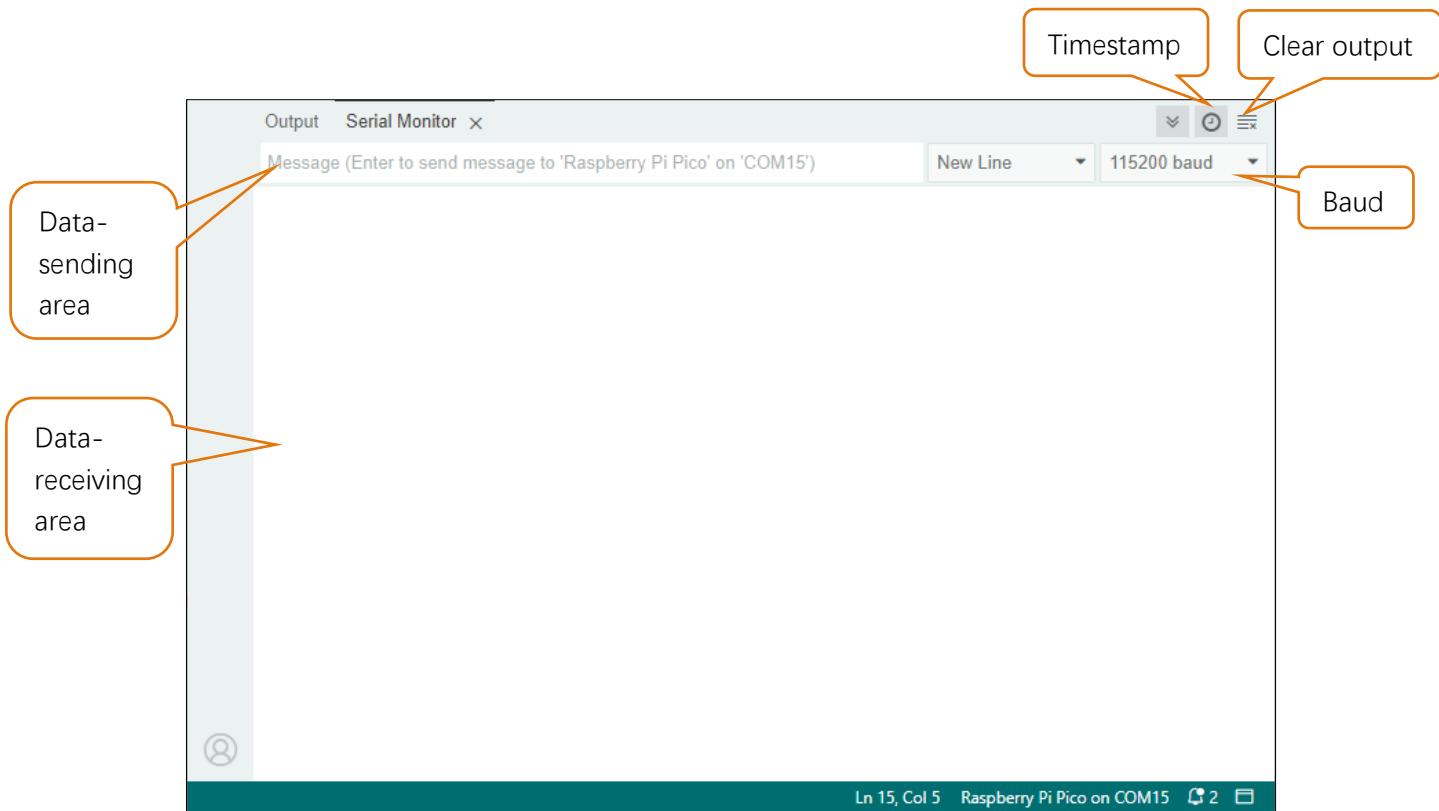


Arduino Software also uploads code to Pico through the serial connection.

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

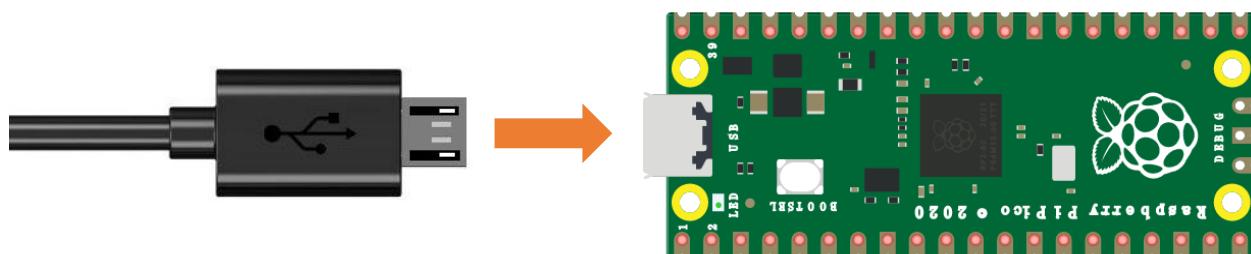


Interface of serial monitor window is as follows. If you cannot open it, make sure Pico has been connected to the computer, and choose the right serial port in the menu bar "Tools".



Circuit

Connect Raspberry Pi Pico to the computer with USB cable.



Sketch

Sketch_08.1_SerialPrinter

A screenshot of the Arduino IDE version 2.3.2. The window title is "Sketch_08.1_SerialPrinter | Arduino IDE 2.3.2". The top menu bar includes File, Edit, Sketch, Tools, Help, and a dropdown for boards set to "Raspberry Pi Pico". The main area shows the code for "Sketch_08.1_SerialPrinter.ino". The code is as follows:

```
1 //*****
2 // Filename    : SerialPrinter
3 // Description : Use UART send some data to PC, and show them on serial
4 // monitor.
5 // Author     : www.freenove.com
6 // Modification: 2021/10/13
7 //*****
8 void setup() {
9     Serial.begin(115200);
10    delay(2000);
11    Serial.println("Raspberry Pi Pico initialization completed!");
12 }
13
14 void loop() {
15     Serial.println( millis() / 1000 % 60);
16     delay(1000);
17 }
```

The status bar at the bottom indicates "Ln 18, Col 1 Raspberry Pi Pico on COM15 4 2".

Download the code to Pico, open the serial port monitor, set the baud rate to 115200. As shown in the following picture:

```

15 | Serial.println( millis() / 1000 % 60 );
16 | delay(1000);
17 |

```

Set the relevant information

Output Serial Monitor x

Message (Enter to send message to 'Raspberry Pi Pico' on 'COM15') New Line 115200 baud

13:59:58.860 -> Raspberry Pi Pico initialization completed!
13:59:58.860 -> 1
13:59:59.831 -> 2
14:00:00.830 -> 3
14:00:01.830 -> 4
14:00:02.868 -> 5
14:00:03.864 -> 6
14:00:04.830 -> 7
14:00:05.851 -> 8

Above the trip is the system information, and below is the result of the code.

Ln 10, Col 10 Raspberry Pi Pico on COM15

As shown above, when the code runs, the data is printed every one second.

Reference

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

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

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

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

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

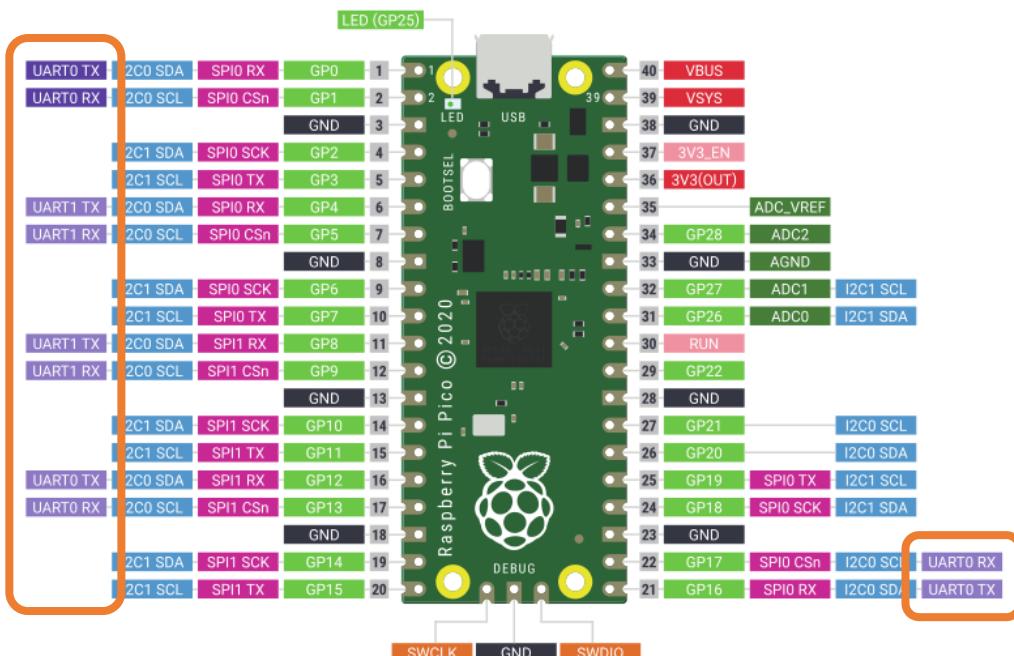
Print formatted content to the serial port in the same way as print in standard C.

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

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

For details, please refer to [UART, I2C, SPI default pin](#).

You can change settings according to the distribution of pins.



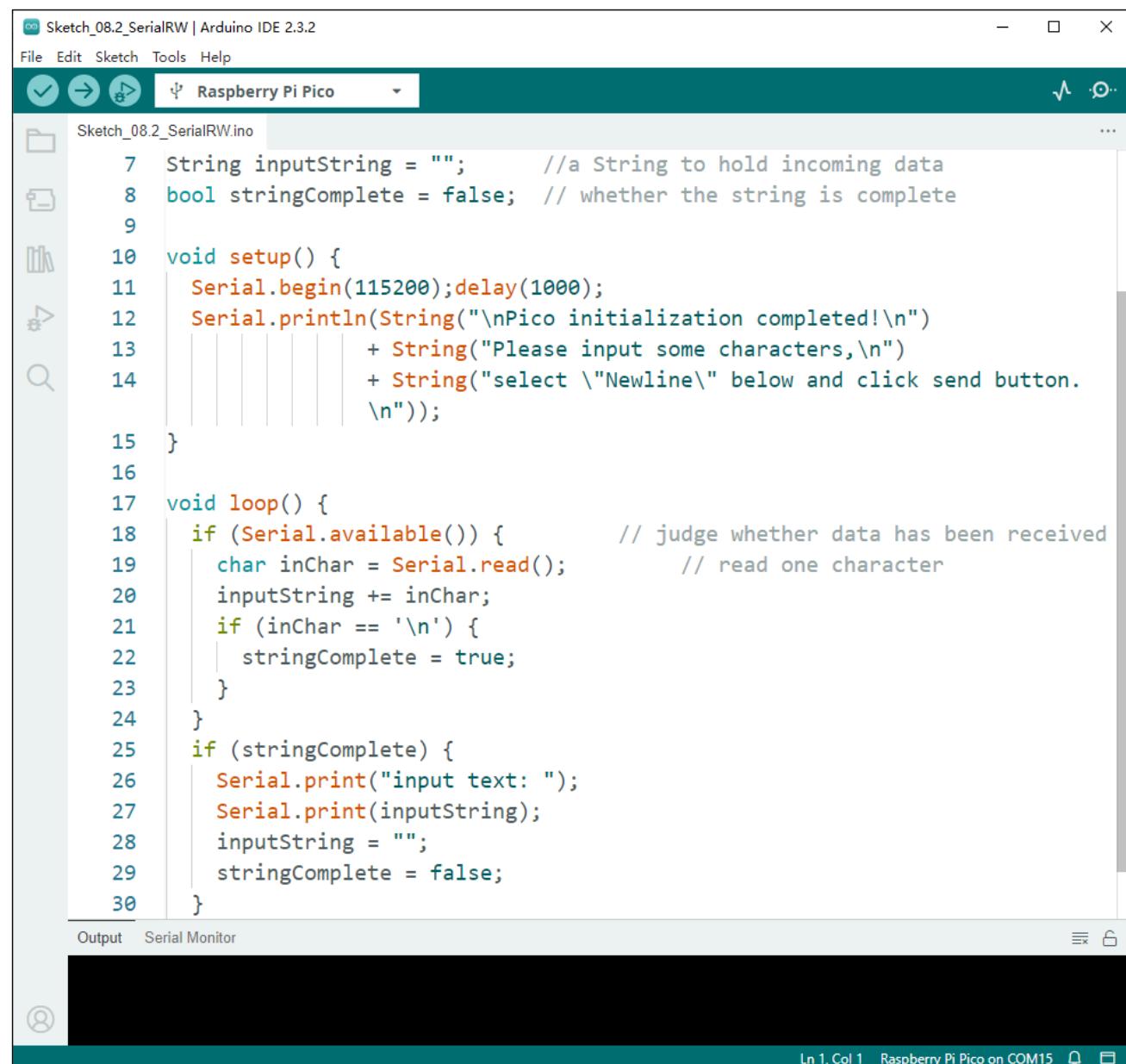
Project 8.2 Serial Read and Write

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

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

Sketch

Sketch_08.2_SerialRW



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

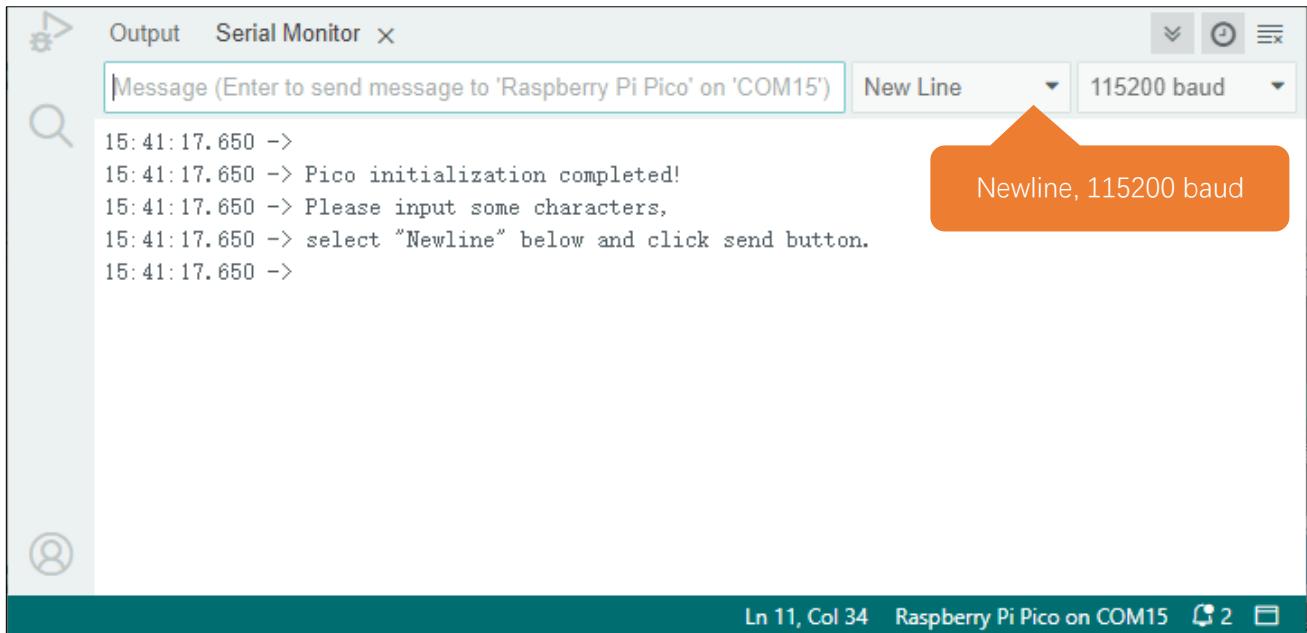
- Title Bar:** Sketch_08.2_SerialRW | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and a dropdown menu set to "Raspberry Pi Pico".
- Code Editor:** Displays the `Sketch_08.2_SerialRW.ino` file content. The code initializes the serial port at 115200 baud, prints a welcome message, and then enters a loop to read incoming characters. It checks if a newline character (\n) has been received to determine if a complete string is available. If so, it prints the string and resets the inputString and stringComplete variables.

```
String inputString = "";      //a String to hold incoming data
bool stringComplete = false; // whether the string is complete
void setup() {
    Serial.begin(115200);delay(1000);
    Serial.println(String("\nPico initialization completed!\n")
                  + String("Please input some characters,\n")
                  + String("select \"Newline\" below and click send button.\n"));
}
void loop() {
    if (Serial.available()) {      // judge whether data has been received
        char inChar = Serial.read(); // read one character
        inputString += inChar;
        if (inChar == '\n') {
            stringComplete = true;
        }
    }
    if (stringComplete) {
        Serial.print("input text: ");
        Serial.print(inputString);
        inputString = "";
        stringComplete = false;
    }
}
```

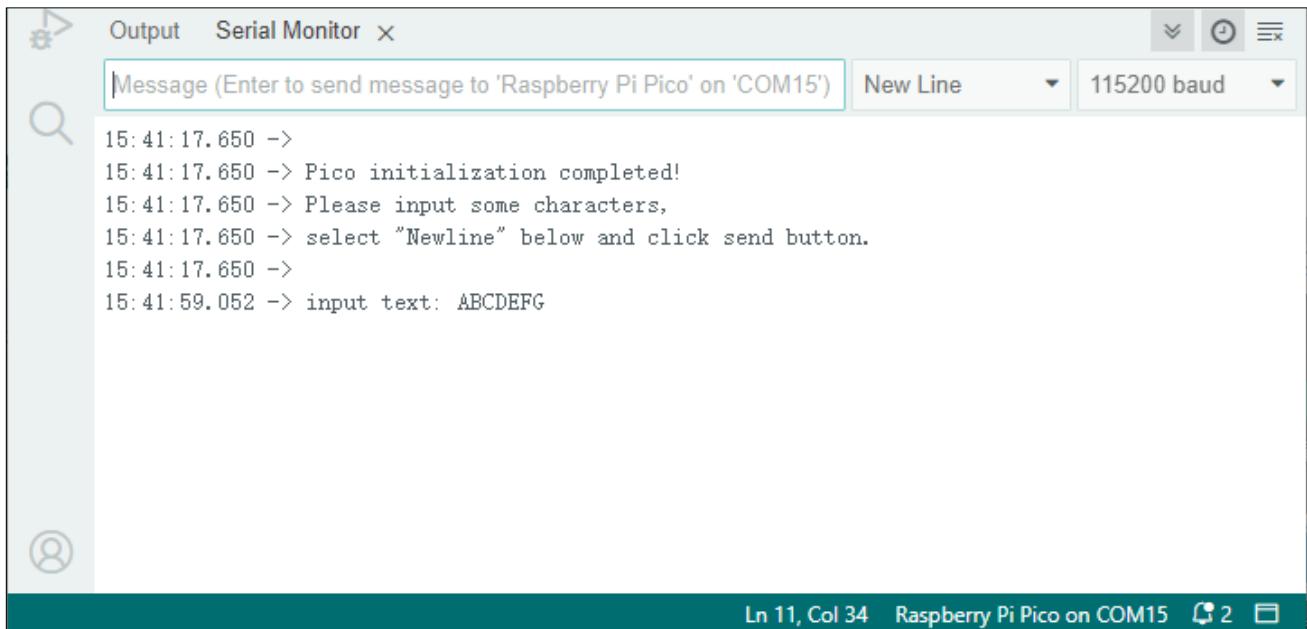
- Status Bar:** Shows "Ln 1, Col 1" and "Raspberry Pi Pico on COM15".



Download the code to Pico, open the serial monitor, and set the bottom to Newline, 115200, as shown in the following picture:



Input texts like "ABCDEFG" in the Message Bar and press Enter to print the data received by Pico.



The following is the program code:

```
1 String inputString = "";      //a String to hold incoming data
2 bool stringComplete = false; // whether the string is complete
3
4 void setup() {
5     Serial.begin(115200);delay(1000);
6     Serial.println(String("\nPico initialization completed!\n")
7                     + String("Please input some characters, \n")
8                     + String("select \"Newline\" below and click send button. \n"));
9 }
10
11 void loop() {
12     if (Serial.available()) {      // judge whether data has been received
13         char inChar = Serial.read();      // read one character
14         inputString += inChar;
15         if (inChar == '\n') {
16             stringComplete = true;
17         }
18     }
19     if (stringComplete) {
20         Serial.print("input text: ");
21         Serial.print(inputString);
22         inputString = "";
23         stringComplete = false;
24     }
25 }
```

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

Reference

String();

Constructs an instance of the String class.

For more information, please visit

<https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>

int available(void);

Get the number of bytes (characters) available for reading from the serial port. This is data that has already arrived and stored in the serial receive buffer.

Serial.read();

Reads incoming serial data.



Chapter 9 AD Converter

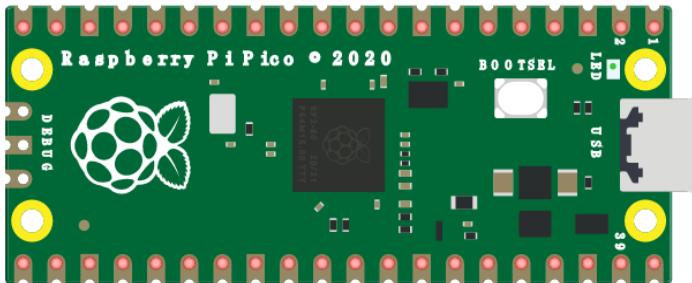
This chapter we learn to use the ADC function of Raspberry Pi Pico.

Project 9.1 Read the Voltage of Potentiometer

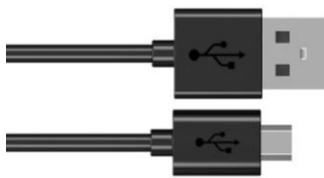
In this chapter, we use ADC function of Pico to read the voltage output by potentiometer.

Component List

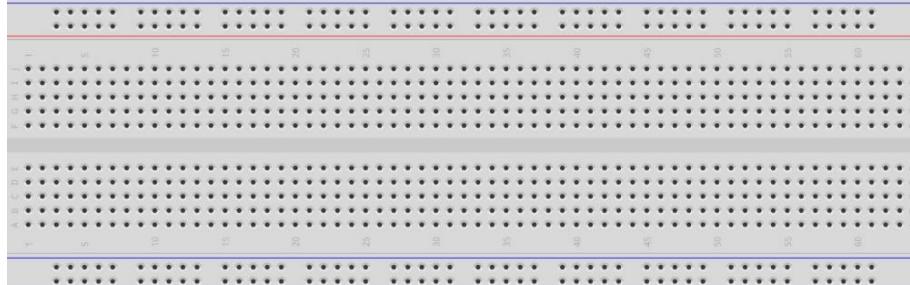
Raspberry Pi Pico x1



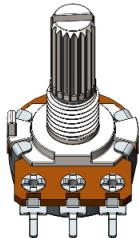
USB cable x1



Breadboard x1



Rotary potentiometer x1



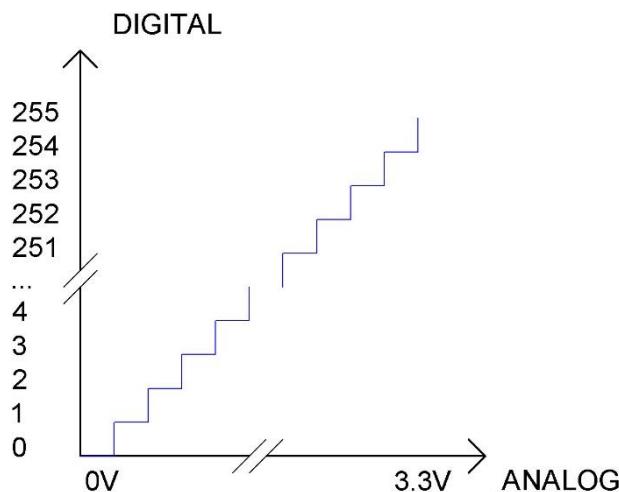
Jumper



Related Knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC on Pico is 10 bits, which means the resolution is $2^{10}=1024$, and it represents a range (at 3.3V) will be divided equally to 1024 parts. The range of analog values corresponds to ADC values. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in range of 0V---3.3/1023 V corresponds to digital 0;

Subsection 2: the analog in range of 3.3/1023V---2*3.3/1023V corresponds to digital 1;

...

The following analog will be divided accordingly.

The conversion formula is as follows:

$$ADC\ Value = \frac{Analog\ Voltage}{3.3} * 1023$$

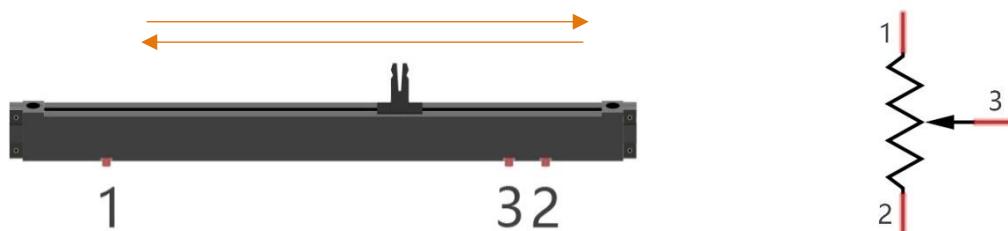
ADC Channels Raspberry Pi Pico

Raspberry Pi Pico has four ADC channels, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29). ADC3 used to measure VSYS on Pico board. Therefore, there are only three generic ADC channels that can be directly used, namely, ADC0, ADC1 and ADC2.

Component Knowledge

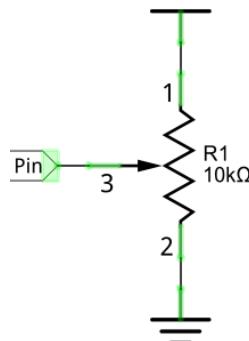
Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



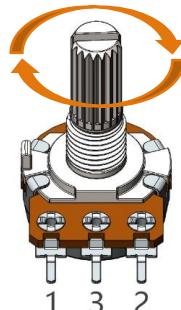
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider, the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



Rotary potentiometer

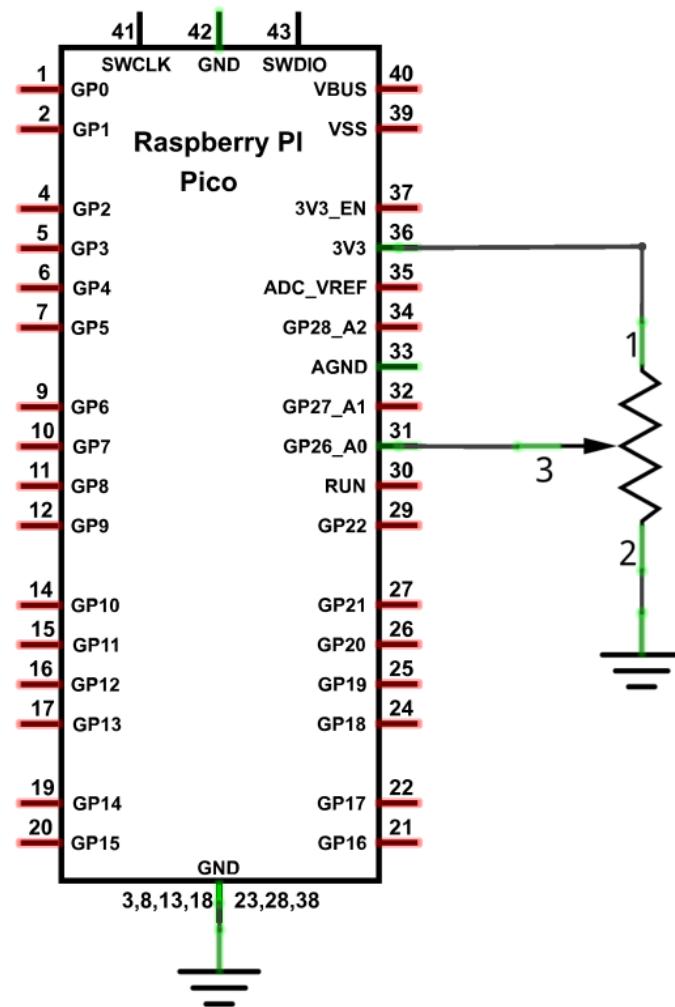
Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



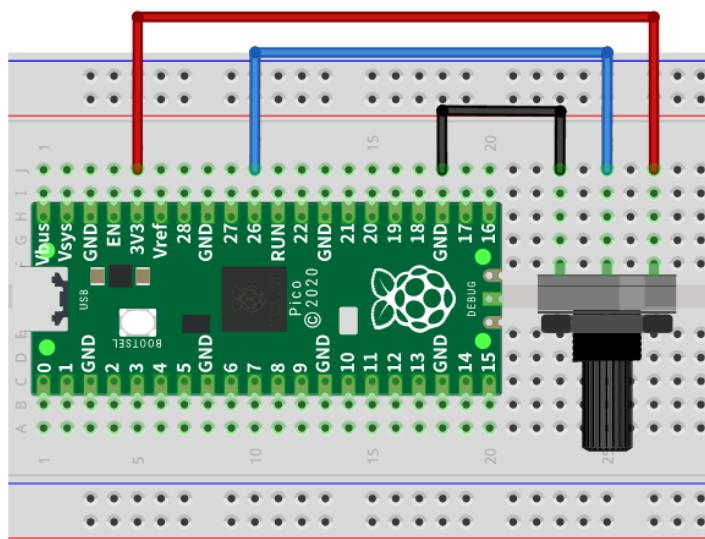
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Circuit

Schematic diagram



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



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



Sketch

Sketch_09.1_ADC

```

Sketch_09.1_ADC | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_09.1_ADC.ino
 7 #define PIN_ANALOG_IN 26
 8
 9 void setup() {
10   Serial.begin(115200);
11 }
12
13 void loop() {
14   int adcVal = analogRead(PIN_ANALOG_IN);
15   double voltage = adcVal / 1023.0 * 3.3;
16   Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: "
17     + String(voltage) + "V");
18   delay(500);
}

```

Output

Ln 1, Col 1 Raspberry Pi Pico on COM15 2

Download the code to Pico, open the serial monitor, and set the baud rate to 115200, as shown in the following picture,

Output Serial Monitor

Message (Enter to send message to 'Raspberry Pi Pico' on 'COM15')

New Line 115200 baud

```

14:21:22.183 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:22.725 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:23.183 -> ADC Value: 246 --- Voltage Value: 0.79V
14:21:23.719 -> ADC Value: 243 --- Voltage Value: 0.78V
14:21:24.198 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:24.711 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:25.206 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:25.684 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:26.184 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:26.717 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:27.228 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:27.685 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:28.231 -> ADC Value: 246 --- Voltage Value: 0.79V
14:21:28.685 -> ADC Value: 243 --- Voltage Value: 0.78V
14:21:29.223 -> ADC Value: 244 --- Voltage Value: 0.79V

```

Ln 1, Col 1 Raspberry Pi Pico on COM15 2

The following is the code:

```
1 #define PIN_ANALOG_IN 26
2
3 void setup() {
4     Serial.begin(115200);
5 }
6
7 void loop() {
8     int adcVal = analogRead(PIN_ANALOG_IN);
9     double voltage = adcVal / 1023.0 * 3.3;
10    Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: " + String(voltage) +
11        "V");
12    delay(500);
13 }
```

In loop() function, analogRead is called to get the ADC value of ADC0 and assign it to adcVal. Calculate the measured voltage value through the formula, and print these data through the serial port monitor.

```
8 int adcVal = analogRead(PIN_ANALOG_IN);
9     double voltage = adcVal / 1023.0 * 3.3;
10    Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: " + String(voltage) +
11        "V");
```

Reference

`uint16_t analogRead(uint8_t pin);`

Reads the value from the specified analog pin. Return the analog reading on the pin. (0-1023 for 10 bits).



Chapter 10 Potentiometer & LED

We have learnt to use ADC in the previous chapter. In this chapter, we will combine PWM and ADC to use potentiometer to control LED, RGBLED and Neopixel.

Project 10.1 Soft Light

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

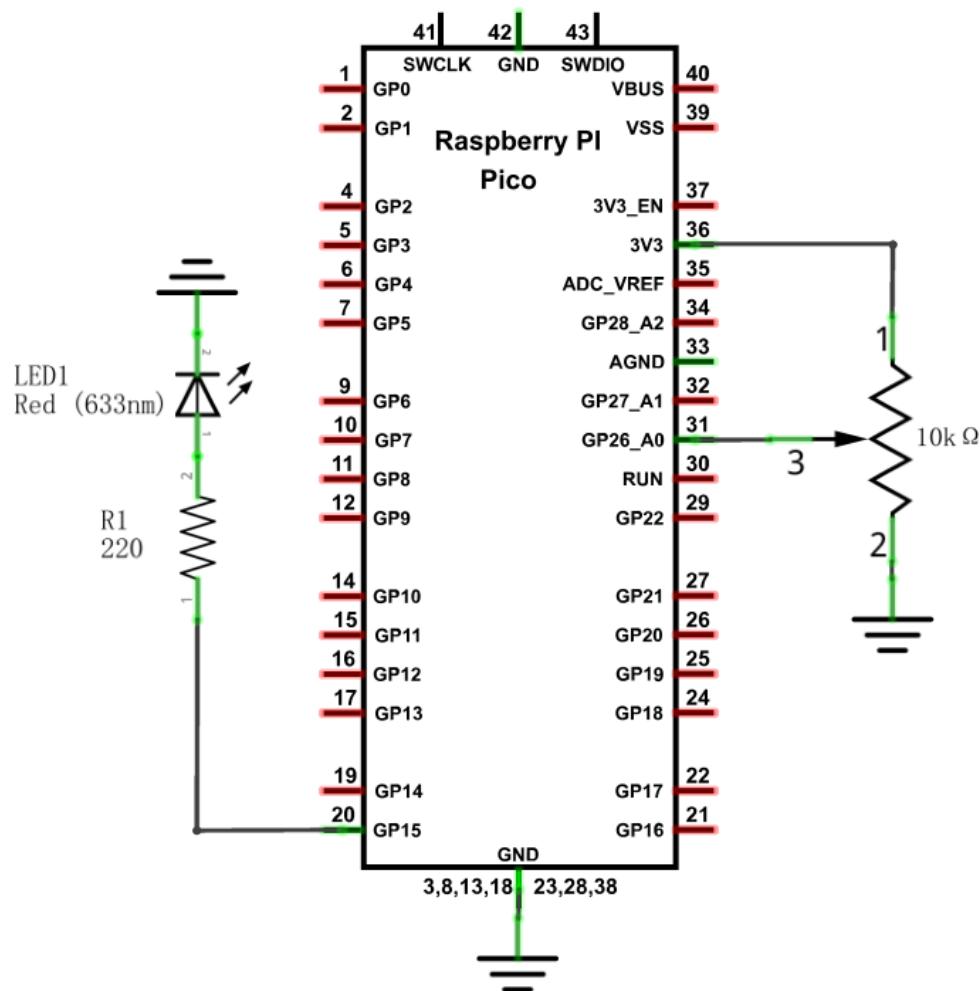
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	Jumper

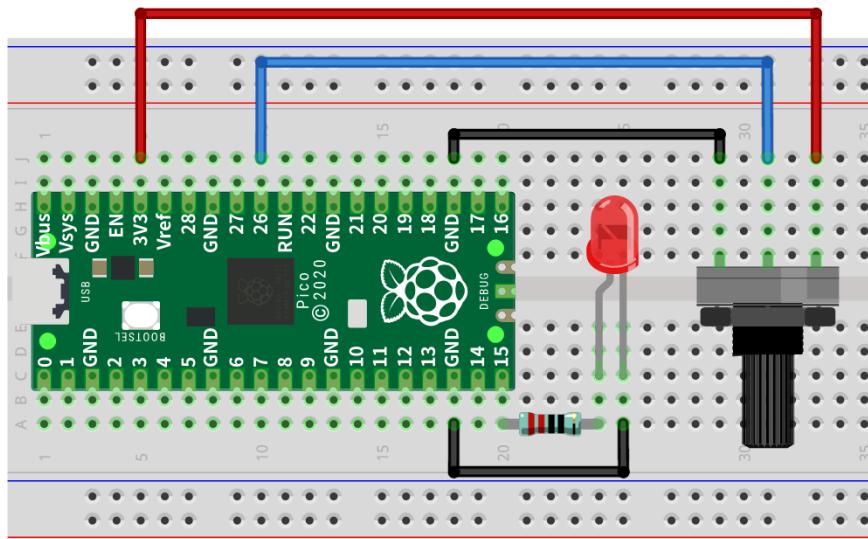
Any concerns? ✉ support@freenove.com

Circuit

Schematic diagram



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



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



Sketch

Sketch_10.1_Softlight

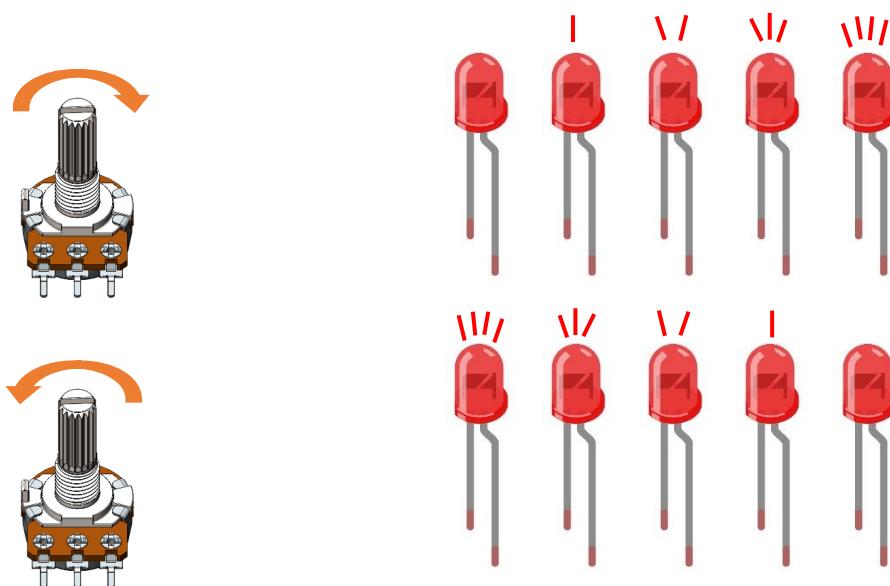
The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch_10.1_SoftLight | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and others.
- Sketch Area:** Displays the code for `Sketch_10.1_SoftLight.ino`. The code reads an analog value from pin GP26 (PIN_ADC0) and maps it to a digital output on pin GP15 (PIN_LED), which controls the brightness of an LED.

```
7 #define PIN_ADC0      26
8 #define PIN_LED       15
9
10 void setup() {
11     pinMode(PIN_LED, OUTPUT);
12 }
13
14 void loop() {
15     int adcVal = analogRead(PIN_ADC0); //read adc
16     analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
17     delay(10);
18 }
```

- Output Area:** Shows "indexing: 2/44".
- Status Bar:** Shows "Ln 1, Col 1" and "Raspberry Pi Pico on COM15".

Download the code to Pico, by turning the adjustable resistor to change the input voltage of GP26, Pico changes the output voltage of GP15 according to this voltage value, thus changing the brightness of the LED.



The following is the code:

```
1 #define PIN_ADC0      26
2 #define PIN_LED       15
3
4 void setup() {
5     pinMode(PIN_LED, OUTPUT);
6 }
7
8 void loop() {
9     int adcVal = analogRead(PIN_ADC0); //read adc
10    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
11    delay(10);
12 }
```

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

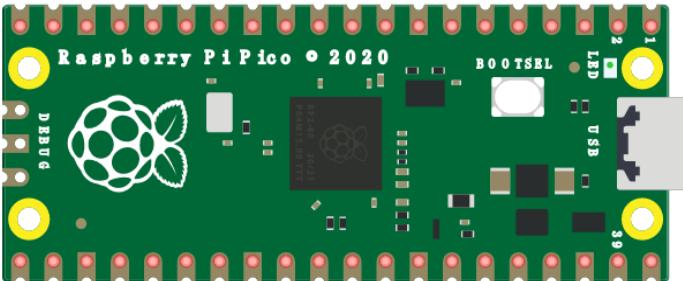
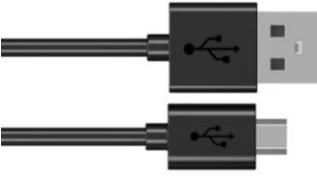
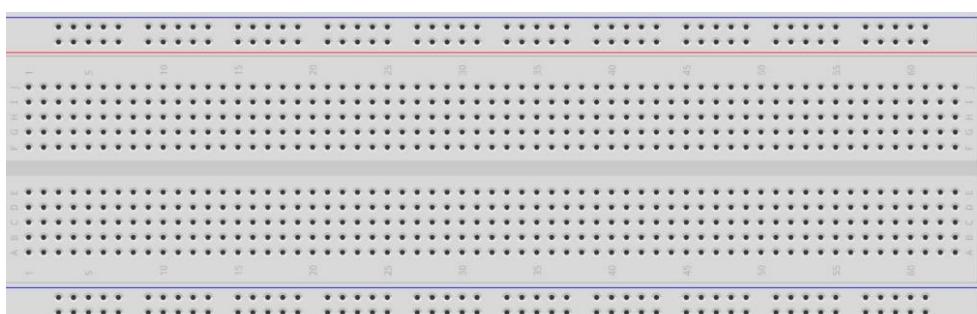
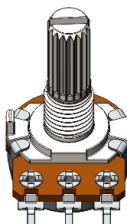
If you have any concerns, please contact us via support@freenove.com



Project 10.2 Soft Colorful Light

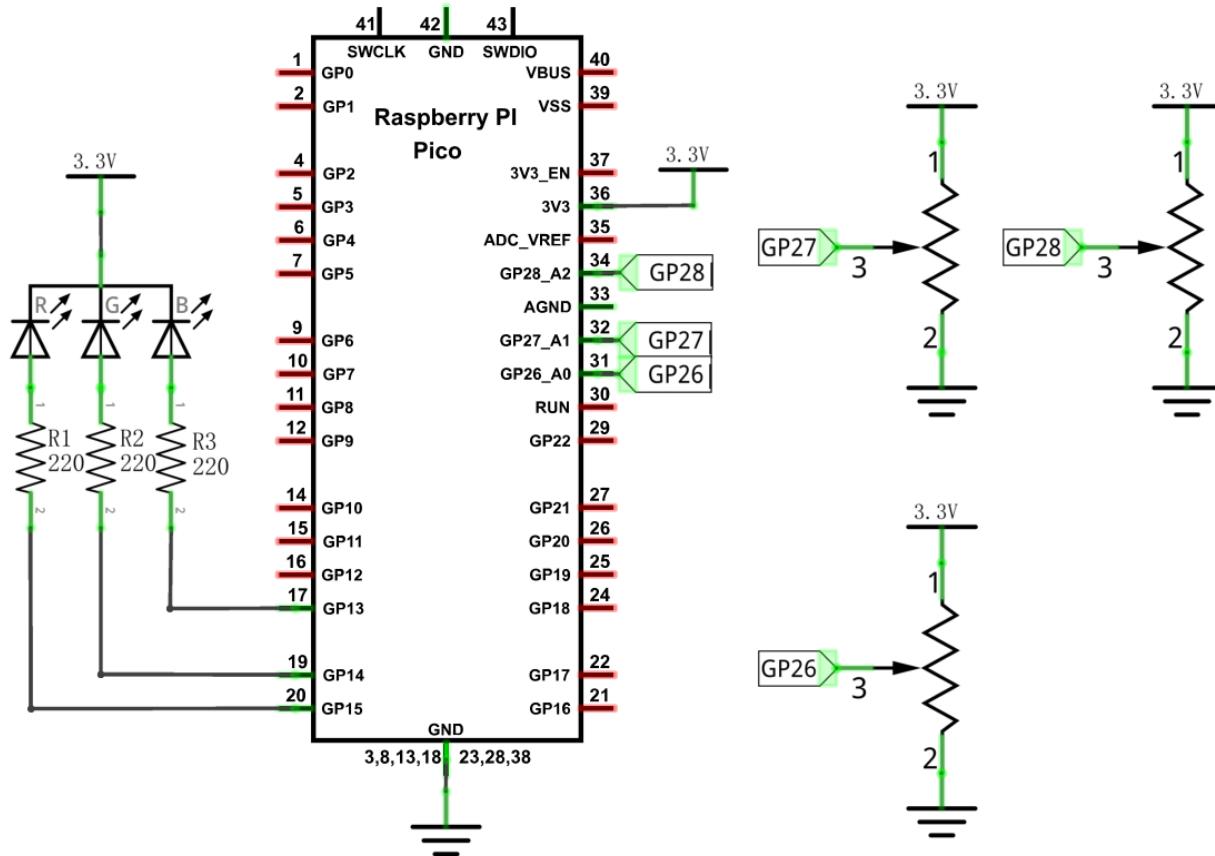
In this project, three potentiometers are used to control the RGB LED and in principle, it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one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

Component List

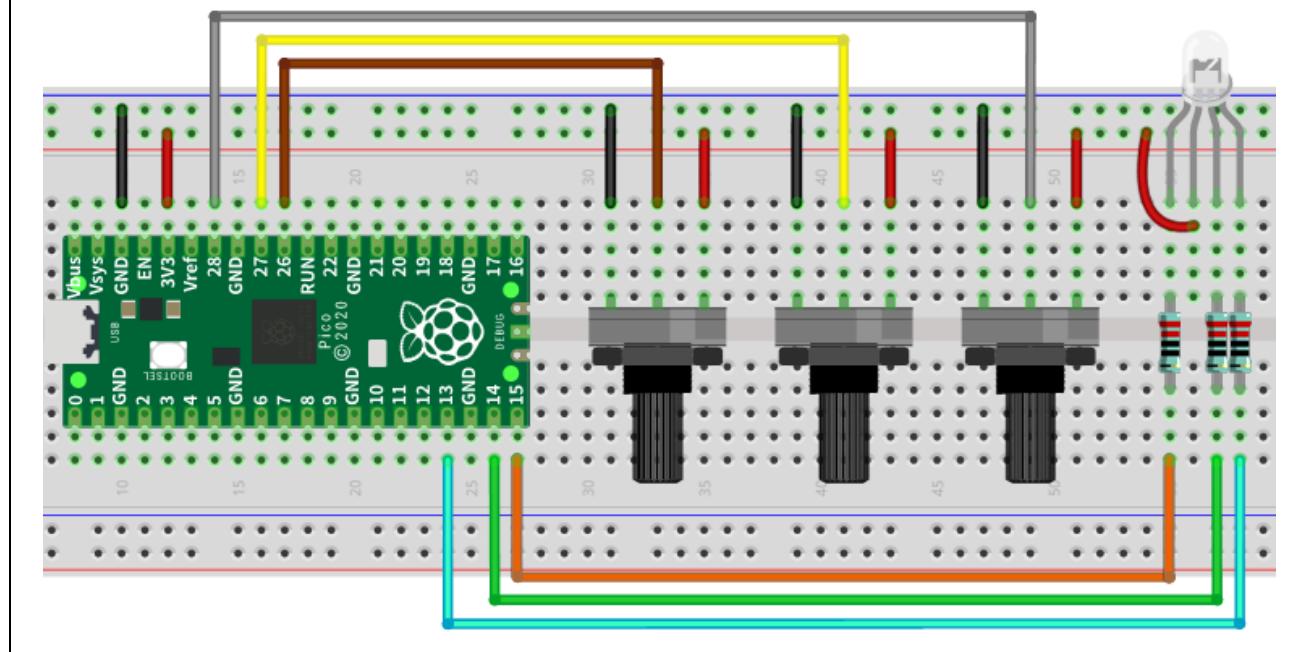
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Rotary potentiometer x3		Resistor 220Ω x3	
RGBLED x1		Jumper	

Circuit

Schematic diagram



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



Any concerns? ✉ support@freenove.com



Sketch

Sketch_10.2_SoftColorfullLight

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

- Title Bar:** Sketch_10.2_SoftColorfullLight | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Selection:** Sketch_10.2_SoftColorfullLight.ino
- Board Selection:** Raspberry Pi Pico
- Code Area:** Displays the following C++ code for the Sketch_10.2_SoftColorfullLight.ino file:

```
7 const byte ledPins[] = {15, 14, 13};      //define led pins
8 const byte adcChns[] = {26, 27, 28};      // define the adc channels
9 int colors[] = {0, 0, 0};                  // red, green ,blue values of
color.
10 void setup() {
11     for (int i = 0; i < 3; i++) {    //setup the pwm channels
12         pinMode(ledPins[i], OUTPUT);
13     }
14 }
15
16 void loop() {
17     for (int i = 0; i < 3; i++) {
18         colors[i] = map(analogReadadcChns[i]), 0, 1023, 0, 255); //calculate
color value.
19         analogWrite(ledPins[i], colors[i]);
20     }
21     delay(10);
22 }
```

Output Area: Shows "Ln 1, Col 1 Raspberry Pi Pico on COM15" at the bottom right.

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

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

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

The following is the program code:

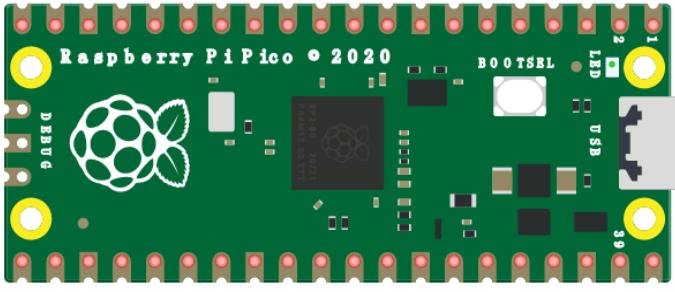
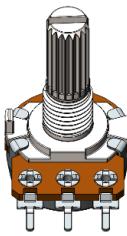
```
1 const byte ledPins[] = {15, 14, 13};      //define led pins
2 const byte adcChns[] = {26, 27, 28};    // define the adc channels
3 int colors[] = {0, 0, 0};                // red, green, blue values of color.
4 void setup() {
5     for (int i = 0; i < 3; i++) {    //setup the pwm channels
6         pinMode(ledPins[i], OUTPUT);
7     }
8 }
9
10 void loop() {
11     for (int i = 0; i < 3; i++) {
12         colors[i] = map(analogRead(adcChns[i]), 0, 1023, 0, 255); //calculate color value.
13         analogWrite(ledPins[i], colors[i]);
14     }
15     delay(10);
16 }
```

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

Project 10.3 Soft Rainbow Light

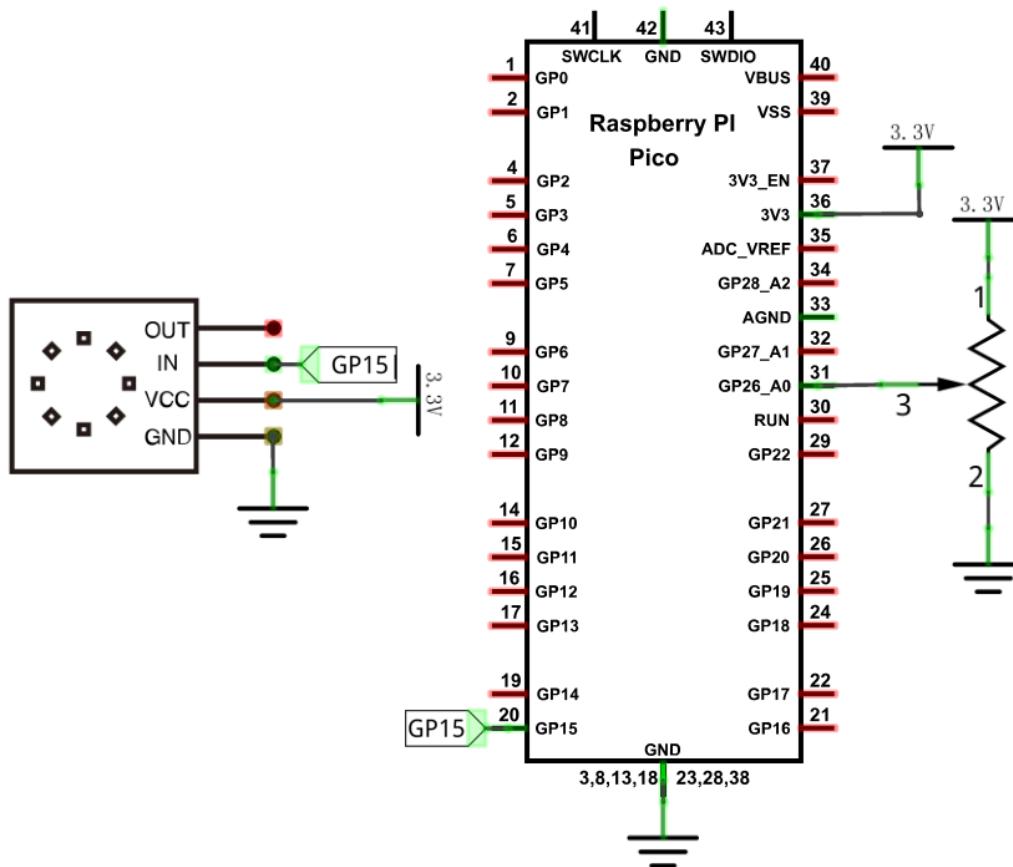
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

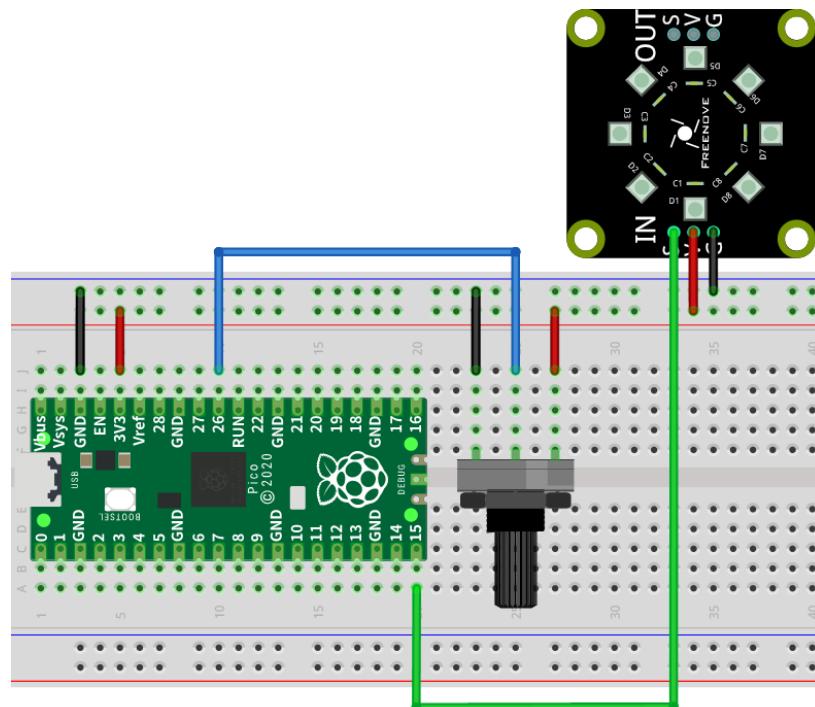
Raspberry Pi Pico x1	USB cable x1
Breadboard x1	
Rotary potentiometer x1	Freenove 8 RGB LED Module x1
	
	Jumper Jumper
	

Circuit

Schematic diagram



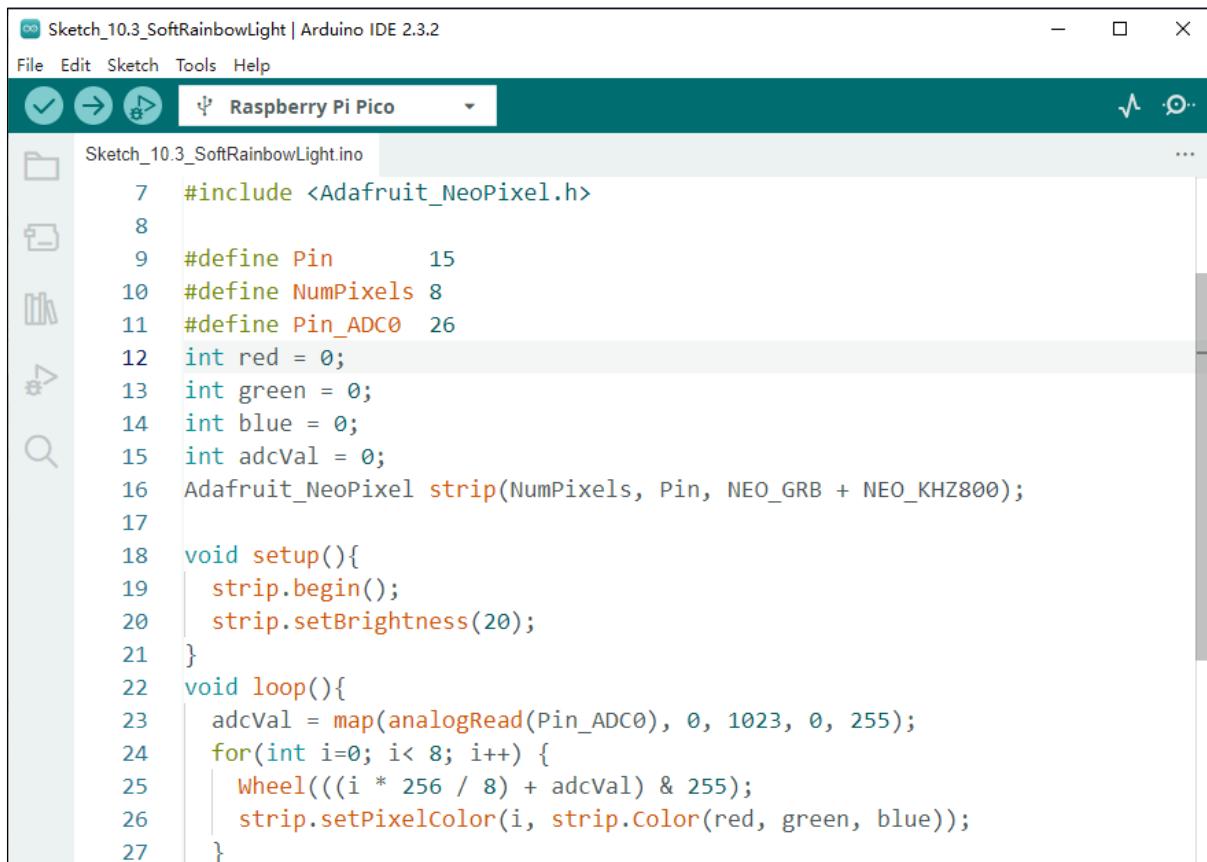
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Sketch

Sketch_10.3_Soft_Rainbow_Light

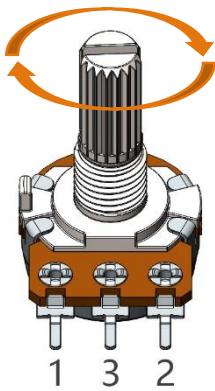


```

Sketch_10.3_SoftRainbowLight | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_10.3_SoftRainbowLight.ino
 7 #include <Adafruit_NeoPixel.h>
 8
 9 #define Pin      15
10 #define NumPixels 8
11 #define Pin_ADC0  26
12 int red = 0;
13 int green = 0;
14 int blue = 0;
15 int adcVal = 0;
16 Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
17
18 void setup(){
19     strip.begin();
20     strip.setBrightness(20);
21 }
22 void loop(){
23     adcVal = map(analogRead(Pin_ADC0), 0, 1023, 0, 255);
24     for(int i=0; i< 8; i++) {
25         wheel(((i * 256 / 8) + adcVal) & 255);
26         strip.setPixelColor(i, strip.Color(red, green, blue));
27     }
}

```

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



The following is the program code:

1	#include <Adafruit_NeoPixel.h>
2	
3	#define Pin 16
4	#define NumPixels 8
5	#define Pin_ADC0 26
6	int red = 0;

```
7 int green = 0;
8 int blue = 0;
9 int adcVal = 0;
10 Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);
11
12 void setup() {
13     strip.begin();
14     strip.setBrightness(20);
15 }
16 void loop() {
17     adcVal = map(analogRead(Pin_ADC0), 0, 1023, 0, 255);
18     for(int i=0; i< 8; i++) {
19         Wheel(((i * 256 / 8) + adcVal) & 255);
20         strip.setPixelColor(i, strip.Color(red, green, blue));
21     }
22     strip.show();
23     delay(10);
24 }
25
26 void Wheel(byte WheelPos) {
27     WheelPos = 255 - WheelPos;
28     if(WheelPos < 85) {
29         red = 255 - WheelPos * 3;
30         green = 0;
31         blue = WheelPos * 3;
32     } else if(WheelPos < 170) {
33         WheelPos -= 85;
34         red = 0;
35         green = WheelPos * 3;
36         blue = 255 - WheelPos * 3;
37     } else{
38         WheelPos -= 170;
39         red = WheelPos * 3;
40         green = 255 - WheelPos * 3;
41         blue = 0;
42     }
43 }
```

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



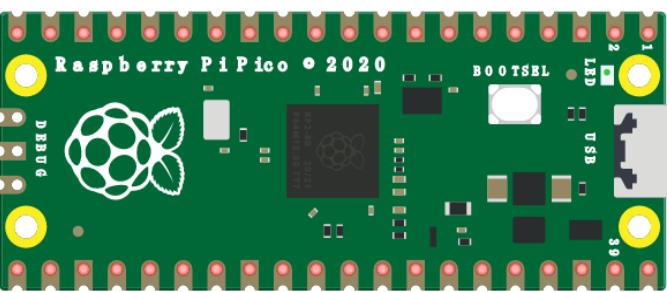
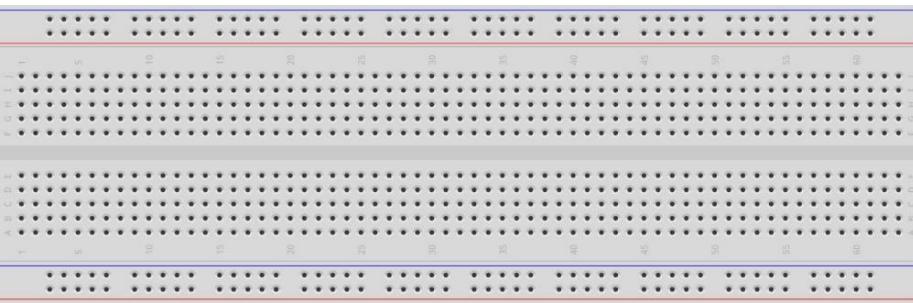
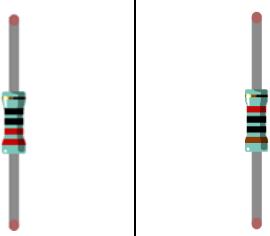
Chapter 11 Photoresistor & LED

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

Project 11.1 Control LED through Photoresistor

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

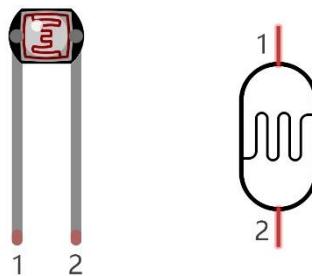
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Photoresistor x1 		Resistor 220Ω x1 10KΩ x1 
LED x1		Jumper

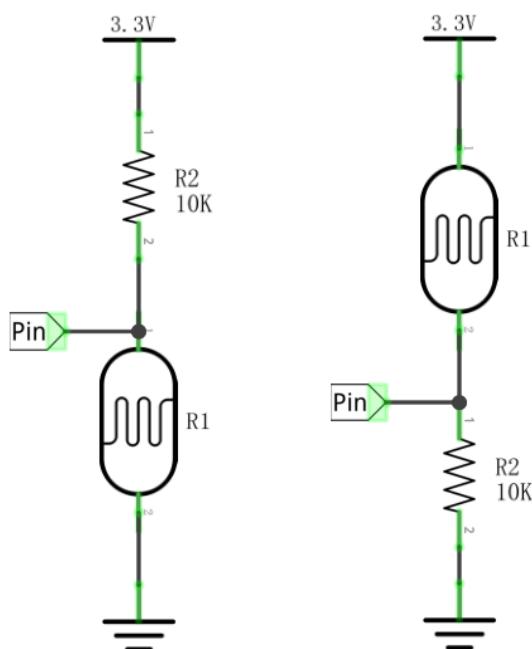
Component Knowledge

Photoresistor

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



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

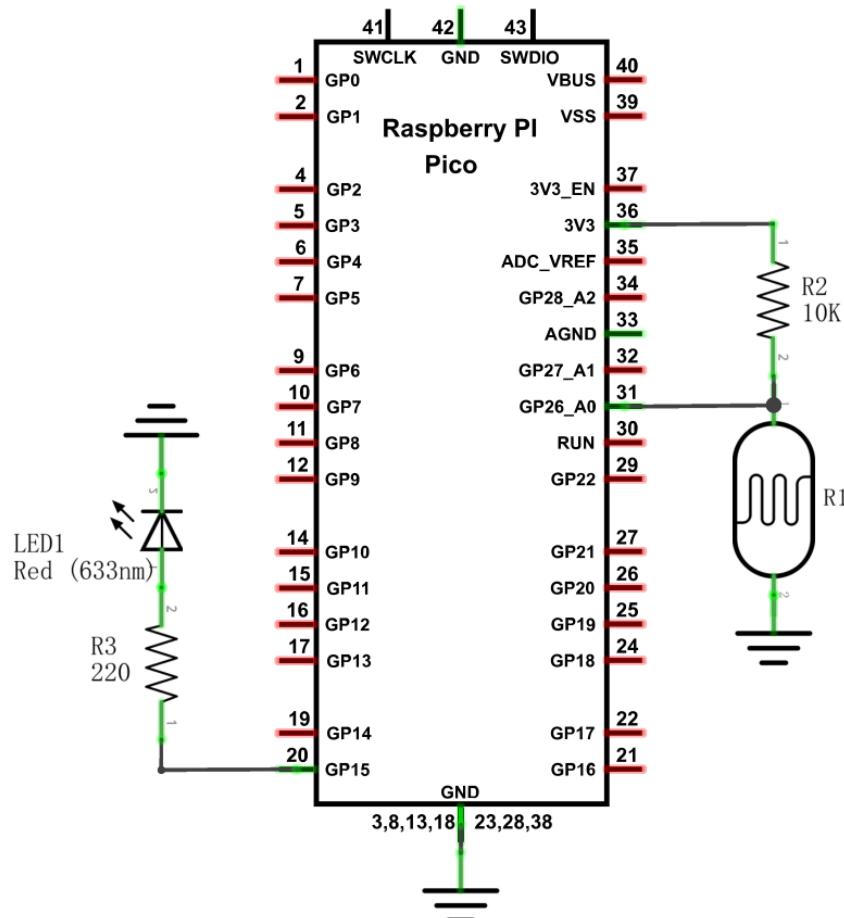


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

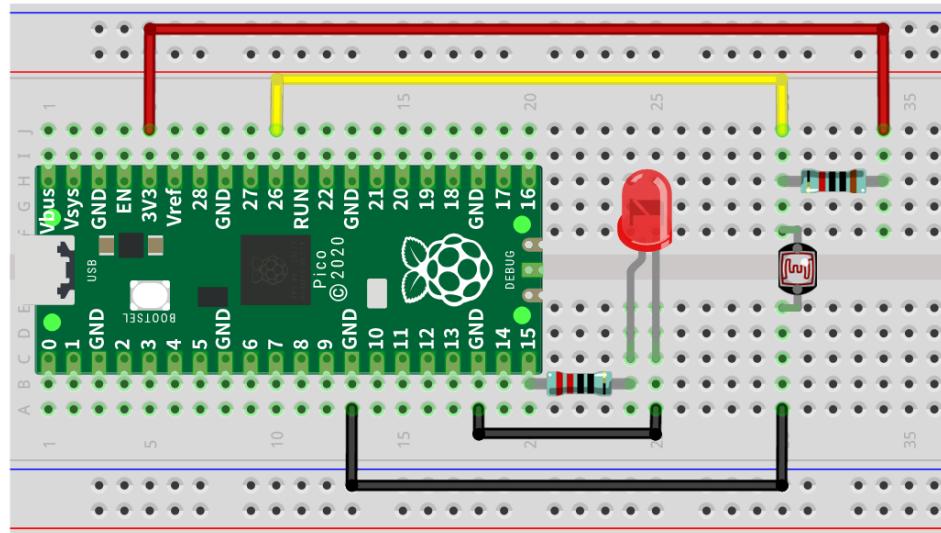
Circuit

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

Schematic diagram



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



Any concerns? ✉ support@freenove.com

Sketch

The circuit used is similar to the project Soft Light. The only difference is that the input signal of the ADC0 pin of ADC changes from a potentiometer to a combination of a photoresistor and a resistor.

Sketch_11.1_Nightlamp

```

Sketch_11.1_Photosensitive | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_11.1_Photosensitive.ino
1 #define PIN_ADC0      26
2 #define PIN_LED       15
3
4 void setup() {
5     pinMode(PIN_LED, OUTPUT);
6 }
7
8 void loop() {
9     int adcVal = analogRead(PIN_ADC0); //Read the voltage of the
10    photoresistor
11    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
12    delay(10);
13 }
14
15
16
17
18 }
```

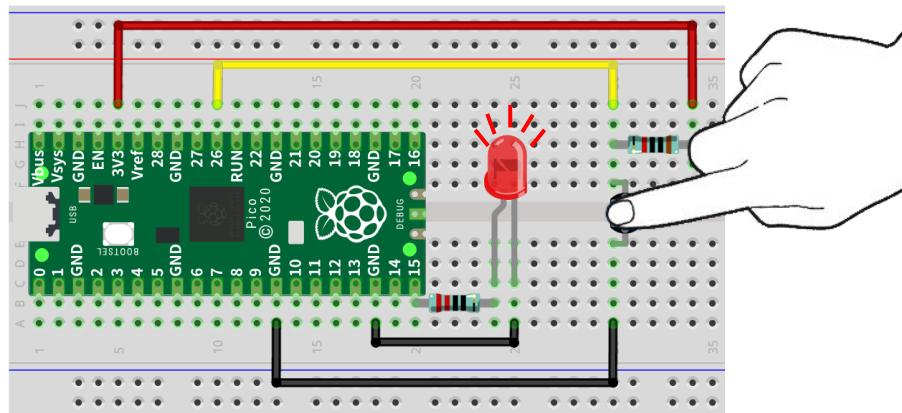
Output

Ln 1, Col 1 Raspberry Pi Pico on COM15

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

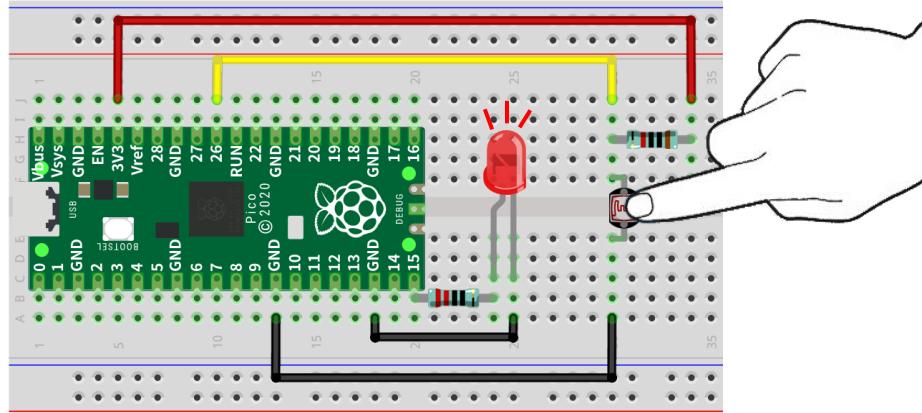
If you have any concerns, please contact us via: support@freenove.com

Fully cover the photoresistor:

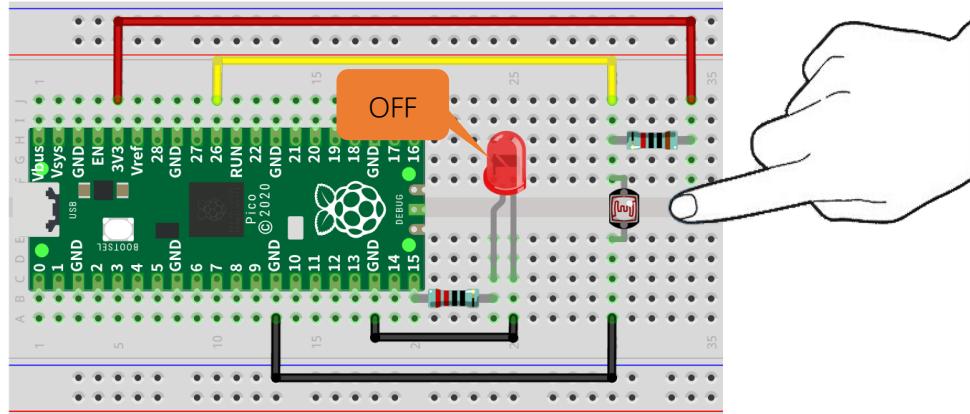


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Half cover the photoresistor:



Not cover the photoresistor:



The following is the program code:

```

1 #define PIN_ADC0      26
2 #define PIN_LED       15
3
4 void setup() {
5     pinMode(PIN_LED, OUTPUT);
6 }
7
8 void loop() {
9     int adcVal = analogRead(PIN_ADC0); //Read the voltage of the photoresistor
10    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
11    delay(10);
12 }
```

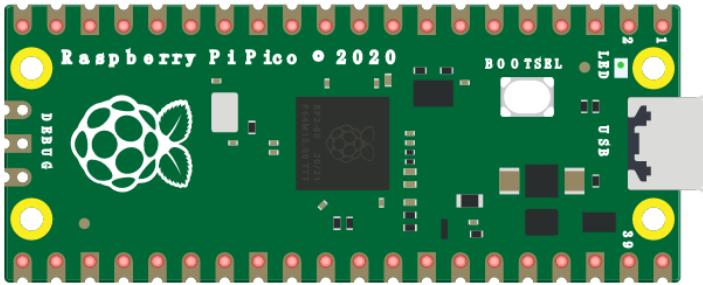
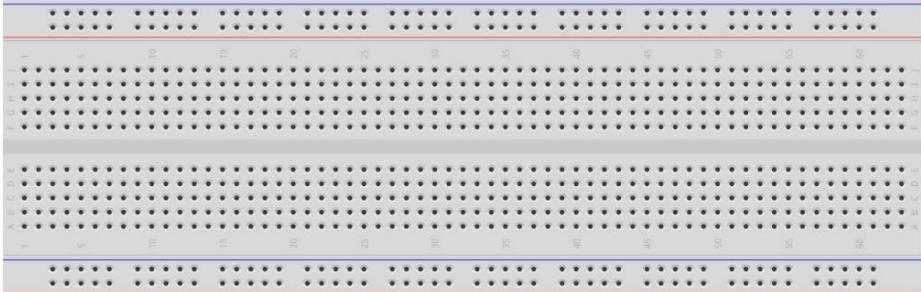
Chapter 12 Thermistor

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

Project 12.1 Thermometer

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

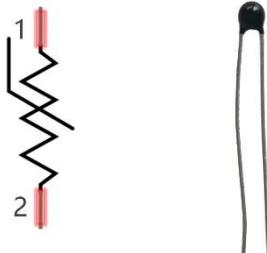
Component List

Raspberry Pi Pico x1	USB cable x1		
			
Breadboard x1			
	Thermistor x1	Resistor 10kΩ x1	Jumper

Component Knowledge

Thermistor

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



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

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

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

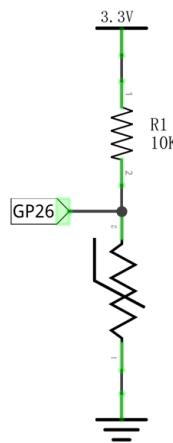
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



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

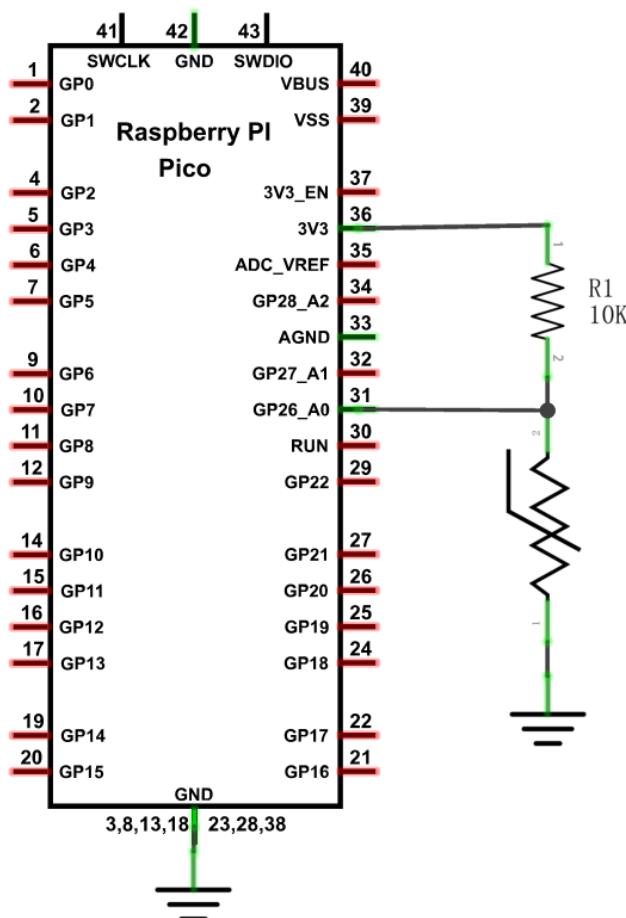
Therefore, the temperature formula can be derived as:

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

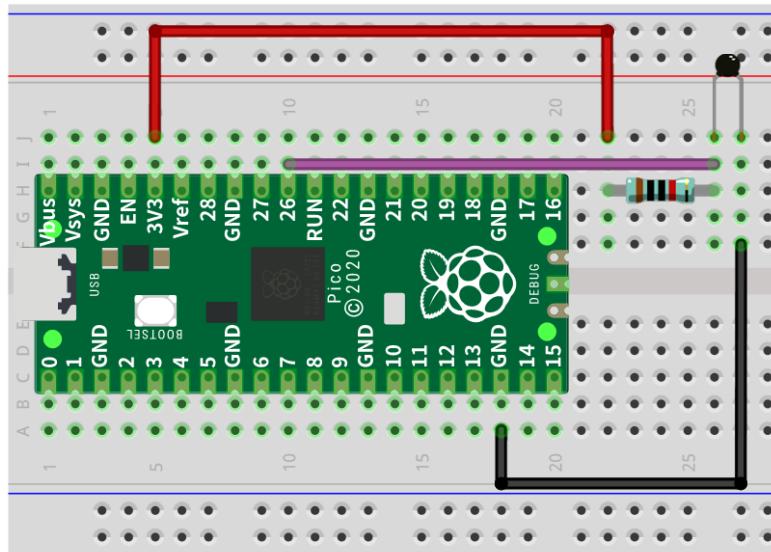
Circuit

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

Schematic diagram



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





Sketch

Sketch_12.1_Thermometer

```

Sketch_12.1_Thermometer | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_12.1_Thermometer.ino
7 #define PIN_ADC0 26
8 void setup() {
9   Serial.begin(115200);
10 }
11
12 void loop() {
13   int adcValue = analogRead(PIN_ADC0); //read ADC pin
14   double voltage = (float)adcValue / 1023.0 * 3.3; // calculate
15   double Rt = 10 * voltage / (3.3 - voltage); //calculate
16   double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0); //calculate
17   double tempC = tempK - 273.15; //calculate
18   Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String
19   (tempK) + "K,\t\t" + "Temperature: " + String(tempC) + "C");
20   delay(1000);
}

```

The screenshot shows the Arduino IDE interface with the sketch file open. The code reads the ADC value from pin 26, calculates the voltage, then uses the Steinhart-Hart formula to calculate temperature in Kelvin and Celsius. It then prints these values to the serial monitor.

Upload the code to Pico and serial monitor will display the current ADC, voltage and temperature values. Try to “pinch” the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

Time	Voltage (V)	Kelvins (K)	Temperature (C)
14:50:30.940	1.54V	301.51K	28.10C
14:50:36.952	1.52V	301.86K	28.71C
14:50:37.959	1.53V	301.58K	28.43C
14:50:38.935	1.52V	301.86K	28.71C
14:50:39.958	1.54V	301.31K	28.16C
14:50:40.953	1.52V	301.67K	28.52C
14:50:41.961	1.52V	301.67K	28.52C
14:50:42.917	1.52V	301.67K	28.52C
14:50:43.940	1.53V	301.49K	28.34C
14:50:44.946	1.54V	301.22K	28.07C
14:50:45.918	1.52V	301.67K	28.52C
14:50:46.919	1.54V	301.31K	28.16C
14:50:47.942	1.53V	301.49K	28.34C
14:50:48.947	1.53V	301.58K	28.43C
14:50:49.960	1.52V	301.67K	28.52C
14:50:50.920	1.53V	301.58K	28.43C

The serial monitor output shows a series of measurements. The first measurement is at 14:50:30.940. Subsequent measurements show fluctuations in voltage and temperature, with one notable increase in temperature (from 28.10C to 28.71C) corresponding to the pinch of the thermistor.

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

Any concerns? support@freenove.com

The following is the code:

```
1 #define PIN_ADC0 26
2 void setup() {
3     Serial.begin(115200);
4 }
5
6 void loop() {
7     int adcValue = analogRead(PIN_ADC0); //read ADC pin
8     double voltage = (float)adcValue / 1023.0 * 3.3;// calculate voltage
9     double Rt = 10 * voltage / (3.3 - voltage);//calculate resistance value of thermistor
10    double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0); //calculate temperature
11    (Kelvin)
12    double tempC = tempK - 273.15; //calculate temperature (Celsius)
13    Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String(tempK) +
14    "K,\t" + "Temperature: " + String(tempC) + "C");
15    delay(1000);
}
```

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



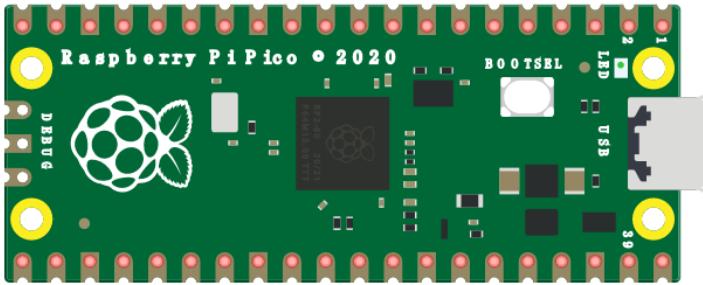
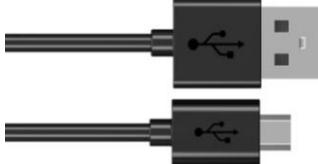
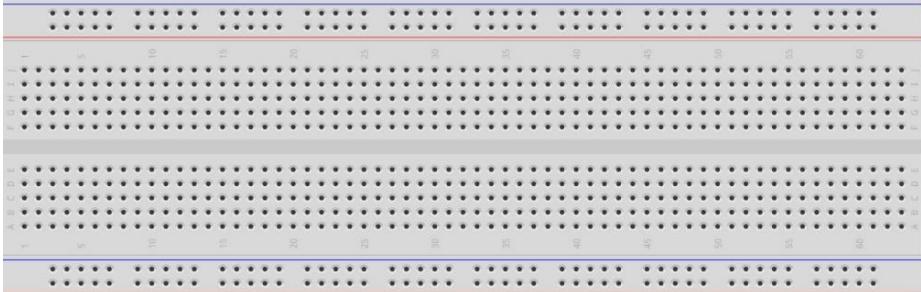
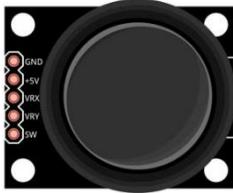
Chapter 13 Joystick

In the previous chapter, we have learned how to use rotary potentiometer. Now, let us learn a new electronic module Joystick that works on the same principle as rotary potentiometer.

Project 13.1 Joystick

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

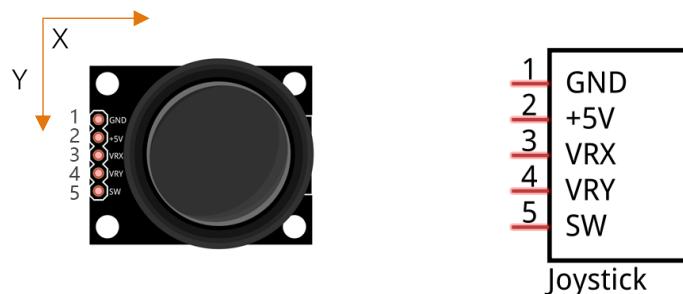
Component List

Raspberry Pi Pico x1	 A green printed circuit board (PCB) with a central Broadcom SoC (System-on-Chip). It features four yellow circular pads labeled 'GND', '3.3V', 'SW', and 'BOOTSEL'. A small white square component is visible near the center. The board has a standard 40-pin header along the bottom edge.	USB cable x1	 Two black USB cables, each with a standard A-type connector at one end and a micro-B connector at the other.	
Breadboard x1	 A schematic diagram of a breadboard. It shows a grid of 40 columns and 6 rows of connection points. The columns are labeled from 1 to 40. The rows are labeled F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z. The breadboard has a standard 40-pin header along the bottom edge.			
Joystick x1	 A black cylindrical joystick module with four pins labeled 'GND', '3.3V', 'VRX', and 'VRY' on the left side. It has two white circular pads on the right side.	Jumper		
		 A single green jumper wire with two black crimp ends.		

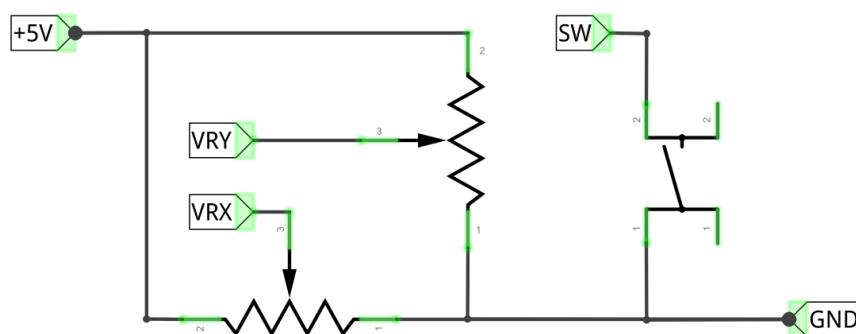
Component Knowledge

Joystick

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



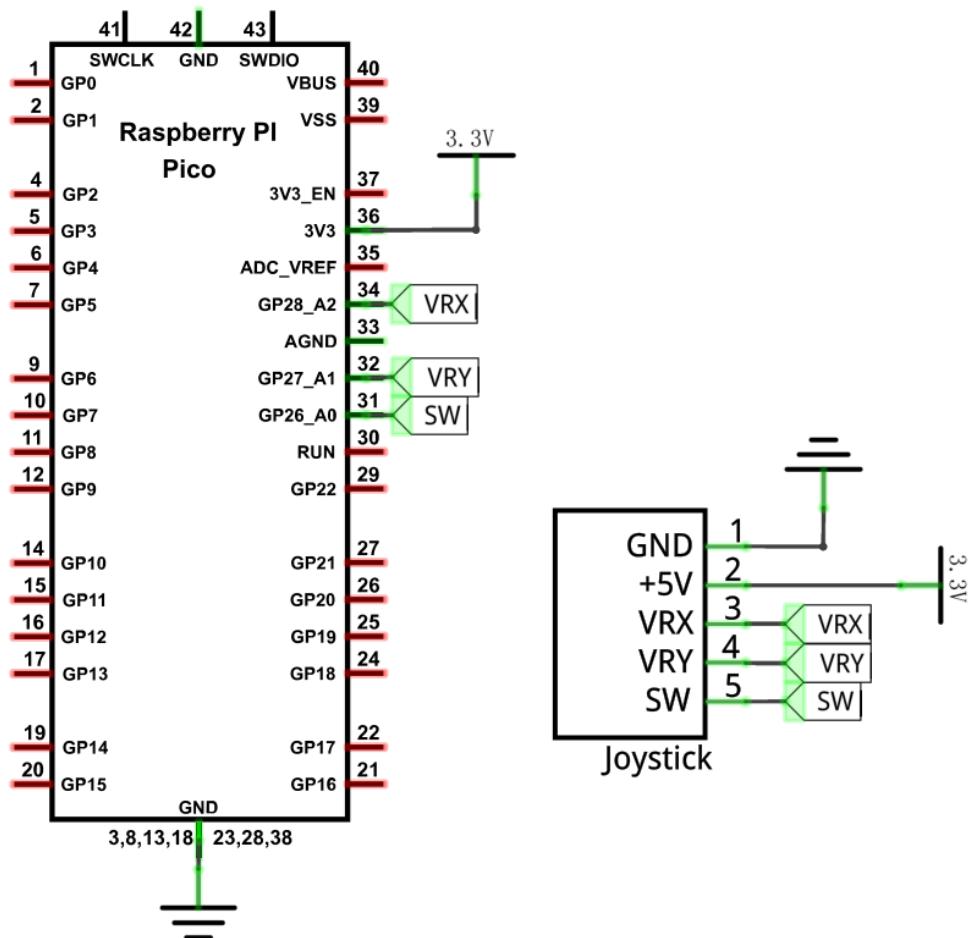
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



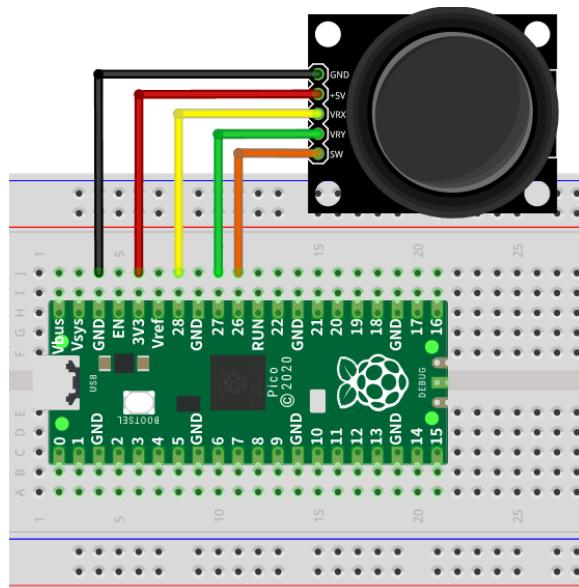
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



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

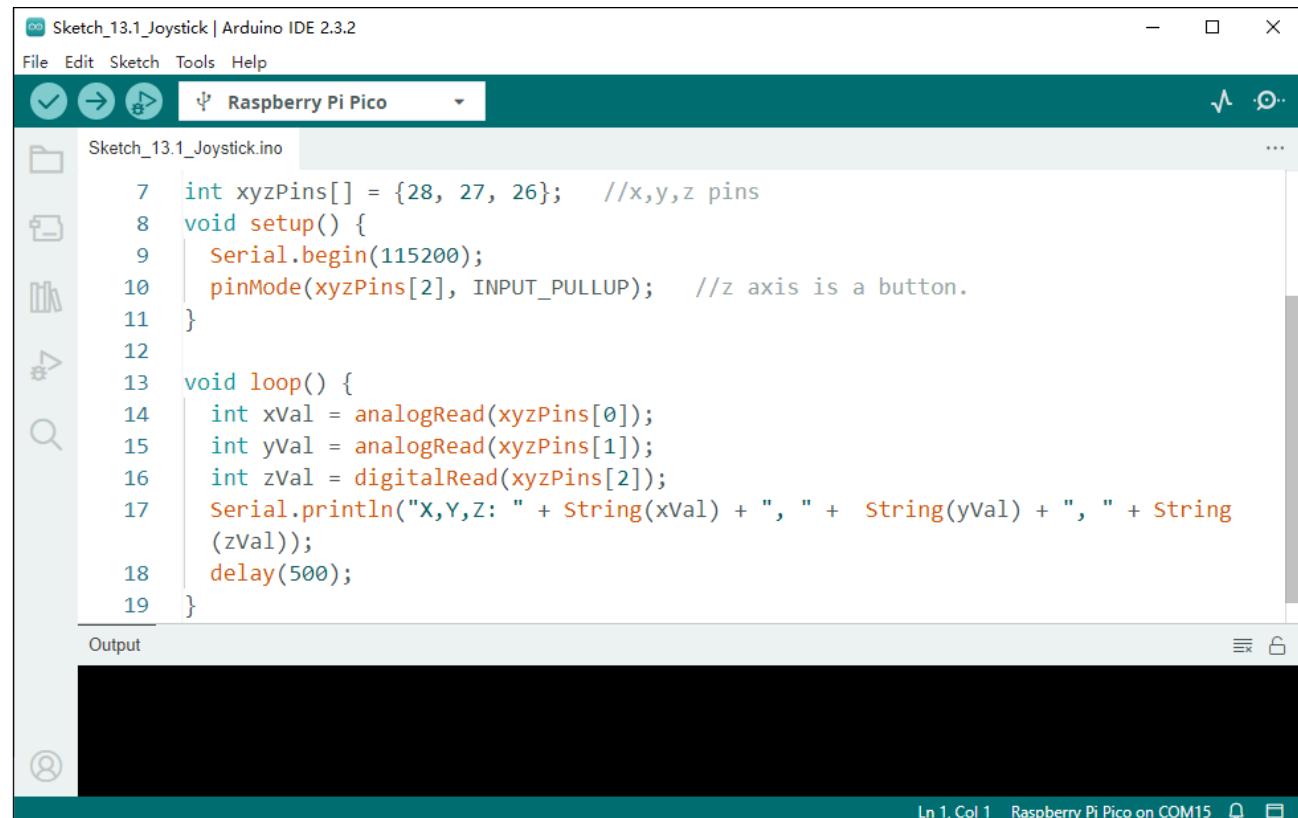


Any concerns? ✉ support@freenove.com

Sketch

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

Sketch_13.1_Joystick



```

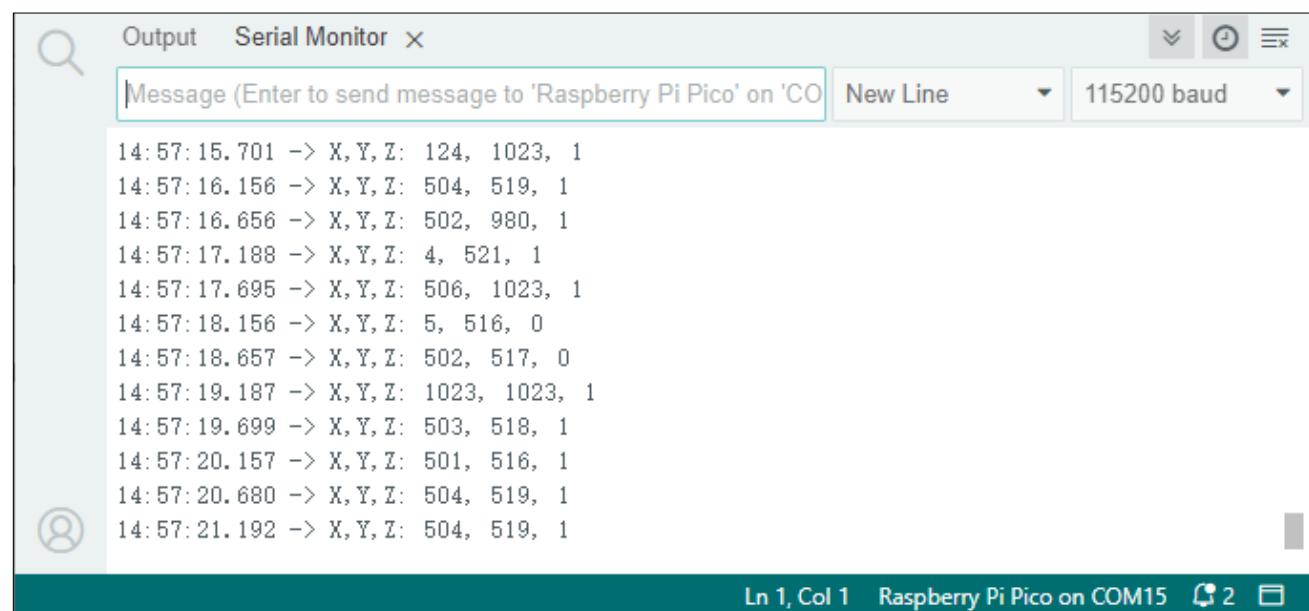
Sketch_13.1_Joystick | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ψ Raspberry Pi Pico
Sketch_13.1_Joystick.ino ...
7 int xyzPins[] = {28, 27, 26}; //x,y,z pins
8 void setup() {
9   Serial.begin(115200);
10  pinMode(xyzPins[2], INPUT_PULLUP); //z axis is a button.
11 }
12
13 void loop() {
14   int xVal = analogRead(xyzPins[0]);
15   int yVal = analogRead(xyzPins[1]);
16   int zVal = digitalRead(xyzPins[2]);
17   Serial.println("X,Y,Z: " + String(xVal) + ", " + String(yVal) + ", " + String(zVal));
18   delay(500);
19 }

```

Output

Ln 1, Col 1 Raspberry Pi Pico on COM15

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



Time	X	Y	Z
14:57:15.701	124	1023	1
14:57:16.156	504	519	1
14:57:16.656	502	980	1
14:57:17.188	4	521	1
14:57:17.695	506	1023	1
14:57:18.156	5	516	0
14:57:18.657	502	517	0
14:57:19.187	1023	1023	1
14:57:19.699	503	518	1
14:57:20.157	501	516	1
14:57:20.680	504	519	1
14:57:21.192	504	519	1

Ln 1, Col 1 Raspberry Pi Pico on COM15

The following is the code:

```
1 int xyzPins[] = {28, 27, 26}; //x, y, z pins
2
3 void setup() {
4     Serial.begin(115200);
5     pinMode(xyzPins[2], INPUT_PULLUP); //z axis is a button.
6 }
7
8 void loop() {
9     int xVal = analogRead(xyzPins[0]);
10    int yVal = analogRead(xyzPins[1]);
11    int zVal = digitalRead(xyzPins[2]);
12    Serial.println("X,Y,Z: " + String(xVal) + ", " + String(yVal) + ", " + String(zVal));
13    delay(500);
14 }
```

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

```
5 pinMode(xyzPins[2], INPUT_PULLUP); //z axis is a button.
```

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

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

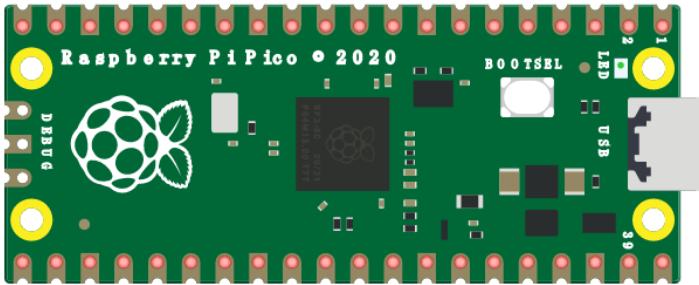
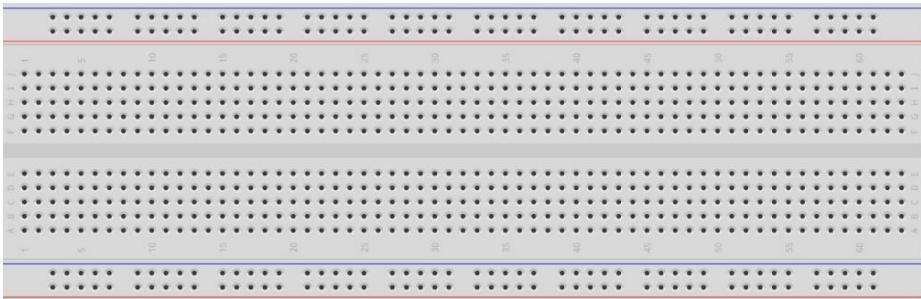
Chapter 14 74HC595 & LED Bar Graph

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

Project 14.1 Flowing Water Light

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

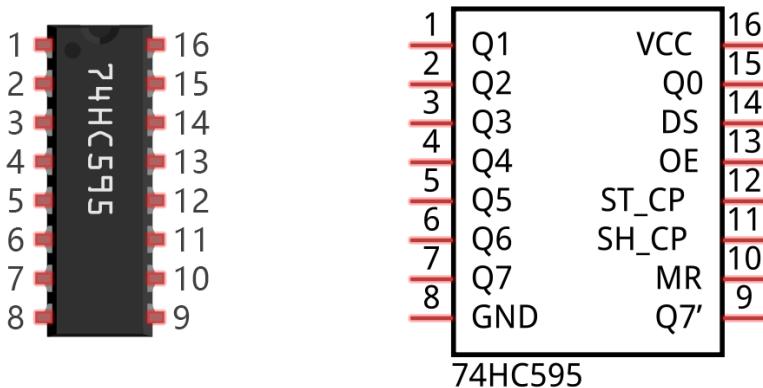
Component List

Raspberry Pi Pico x1		USB cable x1		
Breadboard x1				
74HC595 x1		LED Bar Graph x1	Resistor 220Ω x8	Jumper

Related Knowledge

74HC595

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

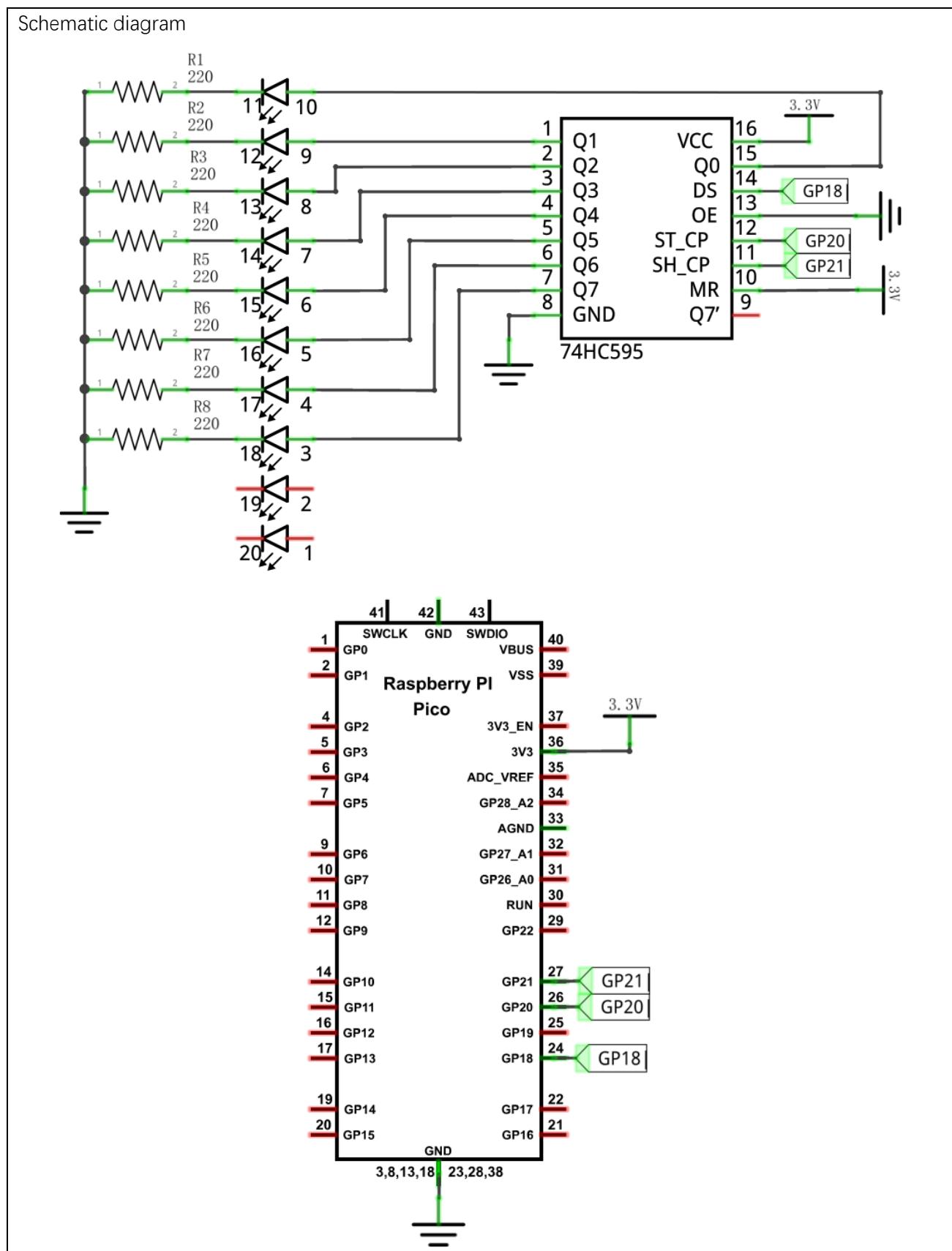


The ports of the 74HC595 chip are described as follows:

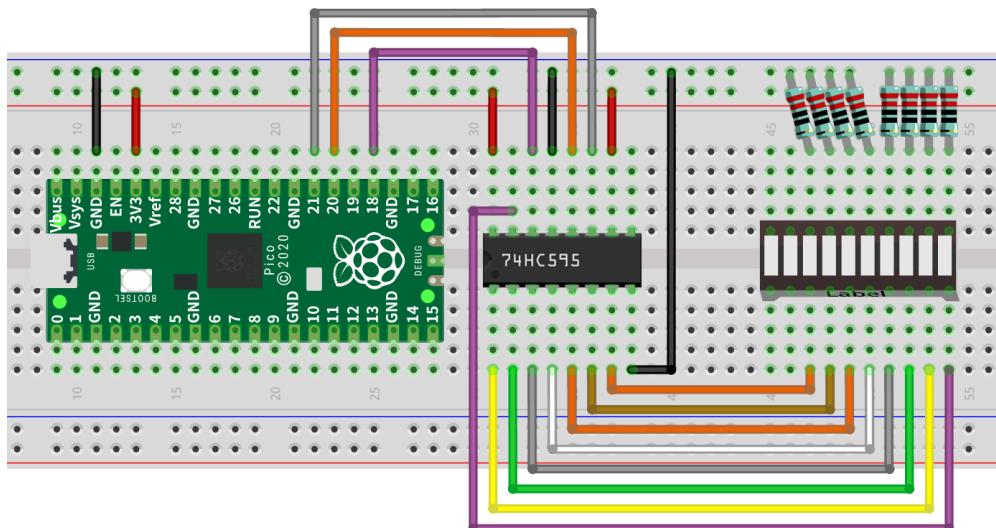
Pin name	GPIO number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this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

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

Circuit



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



Sketch

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

Sketch_14.1_FlowingLight2

```

Sketch_14.1_FlowingLight2 | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_14.1_FlowingLight2.ino
1 int dataPin = 18; // Pin connected to DS of 74HC595(Pin14)
2 int latchPin = 20; // Pin connected to ST_CP of 74HC595(Pin12)
3 int clockPin = 21; // Pin connected to SH_CP of 74HC595(Pin11)
4
5 void setup() { // set pins to output
6     pinMode(latchPin, OUTPUT);
7     pinMode(clockPin, OUTPUT);
8     pinMode(dataPin, OUTPUT);
9 }
10
11 void loop() {
12     // Define a one-byte variable to use the 8 bits to represent the state of 8
13     // LEDs of LED bar graph.
14     // This variable is assigned to 0x01, that is binary 00000001, which
15     // indicates only one LED light on.
16     byte x = 0x01; // 0b 0000 0001
17     for (int j = 0; j < 8; j++) { // Let led light up from right to left
18         writeTo595(LSBFIRST, x);
19         x <<= 1; // make the variable move one bit to left once, then the bright
20
21
22
23
Output
Ln 1, Col 1 Raspberry Pi Pico on COM15

```

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

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

Any concerns? ✉ support@freenove.com

The following is the program code:

```

1 int dataPin = 18; // Pin connected to DS of 74HC595(Pin14)
2 int latchPin = 20; // Pin connected to ST_CP of 74HC595(Pin12)
3 int clockPin = 21; // Pin connected to SH_CP of 74HC595(Pin11)
4
5 void setup() { // set pins to output
6     pinMode(latchPin, OUTPUT);
7     pinMode(clockPin, OUTPUT);
8     pinMode(dataPin, OUTPUT);
9 }
10
11 void loop() {
12     // Define a variable to use the 8 bits to represent the state of 8 LEDs of LED bar graph.
13     // This variable is assigned to 0x01, which indicates only one LED light on.
14     byte x = 0x01; // 0b 0000 0001
15     for (int j = 0; j < 8; j++) { // Let led light up from right to left
16         writeTo595(LSBFIRST, x);
17         x <<= 1; // make the variable move one bit to left once, then the bright LED move one step
18         to the left once.
19         delay(100);
20     }
21     delay(100);
22     x = 0x80; // 0b 1000 0000
23     for (int j = 0; j < 8; j++) { // Let led light up from left to right
24         writeTo595(LSBFIRST, x);
25         x >>= 1;
26         delay(100);
27     }
28 }
29 void writeTo595(BitOrder order, byte _data) {
30     // Output low level to latchPin
31     digitalWrite(latchPin, LOW);
32     // Send serial data to 74HC595
33     shiftOut(dataPin, clockPin, order, _data);
34     // Output high level to latchPin, and 74HC595 will update the data to the parallel output
35     // port.
36     digitalWrite(latchPin, HIGH);
}

```

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

14	byte x = 0x01; // 0b 0000 0001
----	--------------------------------

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

```

15   for (int j = 0; j < 8; j++) { // Let led light up from right to left
16     writeTo595(LSBFIRST, x);
17     x <<= 1;
18     delay(50);
19 }
```

In second "for" loop, the situation is the same. The difference is that x is shift from 0x80 to the RIGHT in order.

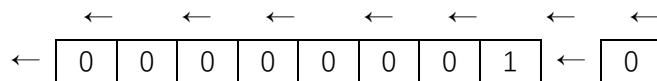
The subfunction `writeTo595()` is used to write data to 74HC595 and immediately output on the port of 74HC595.

Reference

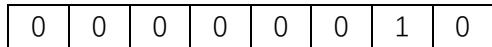
<< operator

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

`byte x = 1 << 1;`

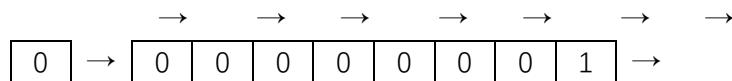


The result of x is 2 (binary 00000010) .



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

`byte x = 1 >> 1;`



The result of x is 0 (00000000) .



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

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

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

Parameters

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

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

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

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

For more details about shift function, please refer to:

<https://www.arduino.cc/reference/en/language/functions/advanced-io/shifout/>

Any concerns? ✉ support@freenove.com

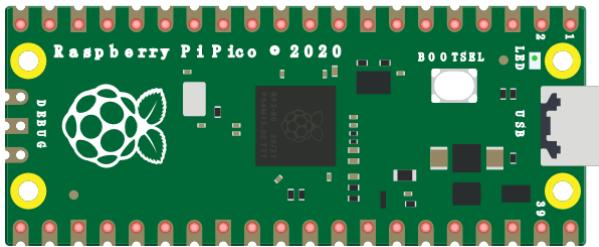
Chapter 15 74HC595 & 7-Segment Display.

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

Project 15.1 7-Segment Display.

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

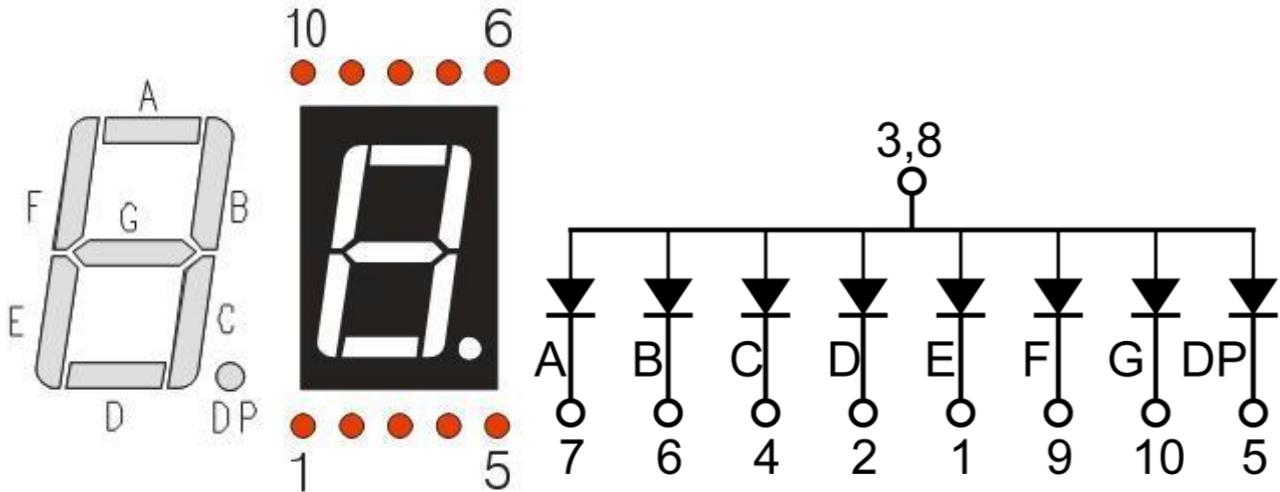
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
74HC595 x1	7-segment display x1	Resistor 220Ω x8	Jumper
			

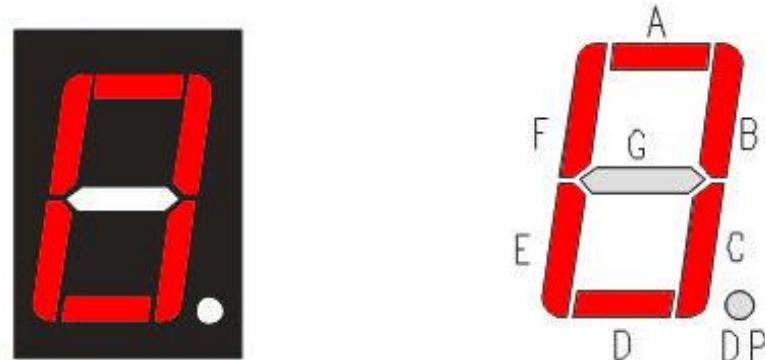
Component Knowledge

7-segment display

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



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



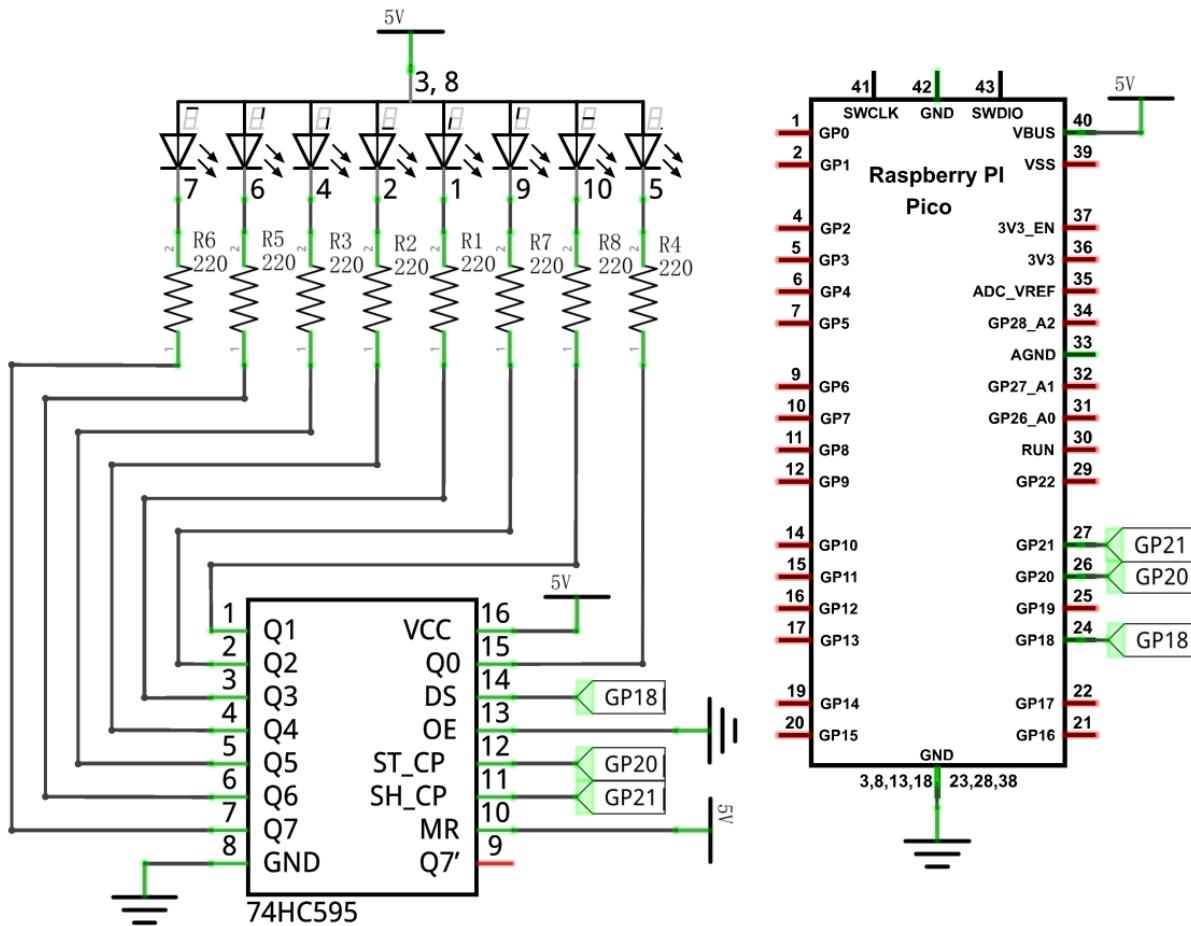
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment "A" as the lowest level and segment "DP" as the highest level, from high to low would look like this: "DP", "G", "F", "E", "D", "C",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

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

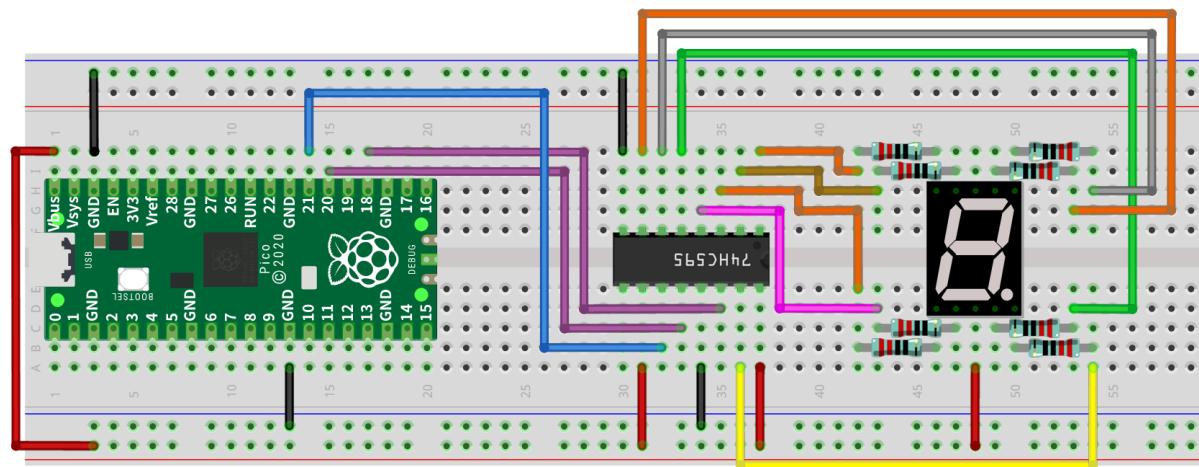
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0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

Circuit

Schematic diagram



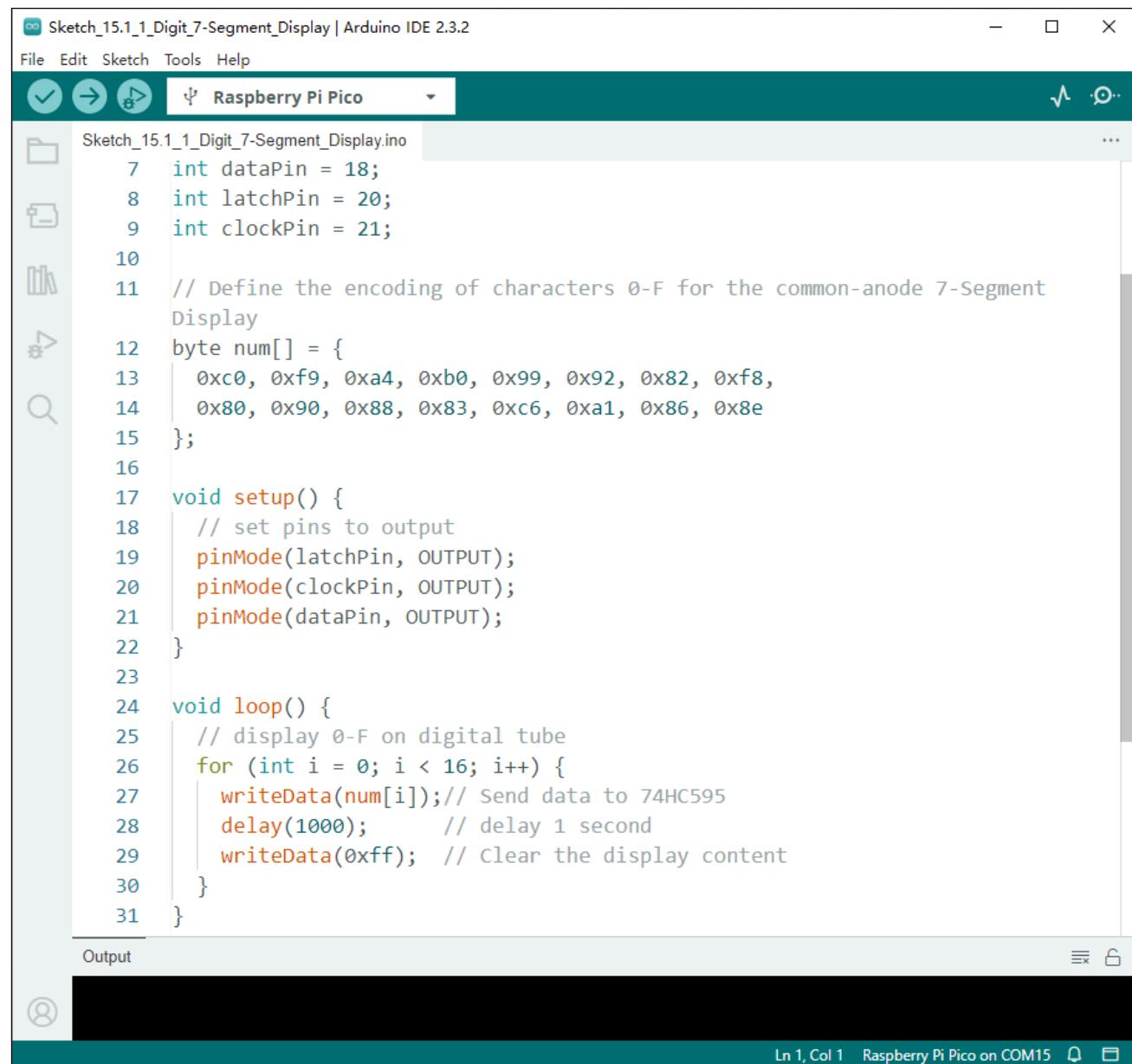
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

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

Sketch_15.1_7_Segment_Display



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

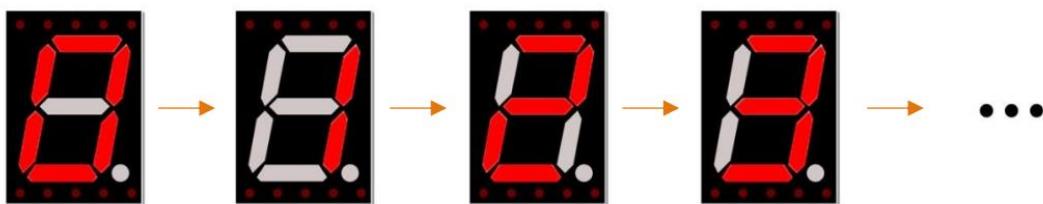
- Title Bar:** Sketch_15.1_1_Digit_7-Segment_Display | Arduino IDE 2.3.2
- Toolbar:** File, Edit, Sketch, Tools, Help
- Sketch Name:** Sketch_15.1_1_Digit_7-Segment_Display.ino
- Board:** Raspberry Pi Pico
- Code Area:** The code is for a common-anode 7-segment display using a 74HC595 shift register. It defines pin numbers and a lookup table for character codes, then sets up pins and loops to display characters from 0 to F.

```
Sketch_15.1_1_Digit_7-Segment_Display.ino
1 int dataPin = 18;
2 int latchPin = 20;
3 int clockPin = 21;
4
5 // Define the encoding of characters 0-F for the common-anode 7-Segment
6 // Display
7 byte num[] = {
8     0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
9     0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
10 };
11
12 void setup() {
13     // set pins to output
14     pinMode(latchPin, OUTPUT);
15     pinMode(clockPin, OUTPUT);
16     pinMode(dataPin, OUTPUT);
17 }
18
19 void loop() {
20     // display 0-F on digital tube
21     for (int i = 0; i < 16; i++) {
22         writeData(num[i]); // Send data to 74HC595
23         delay(1000); // delay 1 second
24         writeData(0xff); // Clear the display content
25     }
26 }
```

- Output Area:** Shows a black placeholder for the terminal window.
- Status Bar:** Ln 1, Col 1 Raspberry Pi Pico on COM15



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



The following is the program code:

```

1 int dataPin = 18;           // Pin connected to DS of 74HC595 (Pin14)
2 int latchPin = 20;          // Pin connected to ST_CP of 74HC595 (Pin12)
3 int clockPin = 21;          // Pin connected to SH_CP of 74HC595 (Pin11)
4 // Define the encoding of characters 0-F for the common-anode 7-Segment Display
5 byte num[] = {
6     0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7     0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
8 };
9
10 void setup() {
11     // set pins to output
12     pinMode(latchPin, OUTPUT);
13     pinMode(clockPin, OUTPUT);
14     pinMode(dataPin, OUTPUT);
15 }
16
17 void loop() {
18     // display 0-F on digital tube
19     for (int i = 0; i < 16; i++) {
20         writeData(num[i]); // Send data to 74HC595
21         delay(1000);      // delay 1 second
22         writeData(0xff); // Clear the display content
23     }
24 }
25
26 void writeData(int value) {
27     // Make latchPin output low level
28     digitalWrite(latchPin, LOW);
29     // Send serial data to 74HC595
30     shiftOut(dataPin, clockPin, LSBFIRST, value);
31     // Make latchPin output high level
32     digitalWrite(latchPin, HIGH);
33 }
```

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

```

4 // Define the encoding of characters 0-F for the common-anode 7-Segment Display
5 byte num[] = {
6     0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7     0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
8 };

```

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

```

17 void loop() {
18     // display 0-F on digital tube
19     for (int i = 0; i < 16; i++) {
20         writeData(num[i]); // Send data to 74HC595
21         delay(1000);      // delay 1 second
22         writeData(0xff); // Clear the display content
23     }
24 }

```

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

```

26 void writeData(int value) {
27     // Make latchPin output low level
28     digitalWrite(latchPin, LOW);
29     // Send serial data to 74HC595
30     shiftOut(dataPin, clockPin, LSBFIRST, value);
31     // Make latchPin output high level, then 74HC595 will update data to parallel output
32     digitalWrite(latchPin, HIGH);
33 }

```

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

```

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

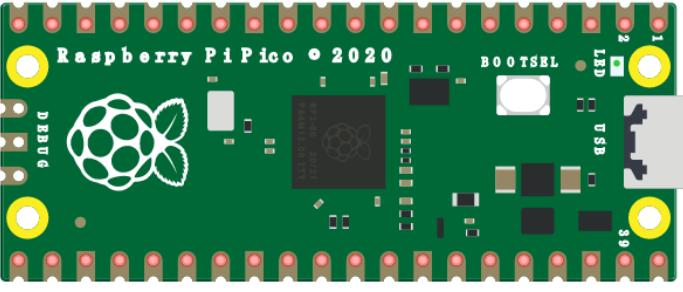
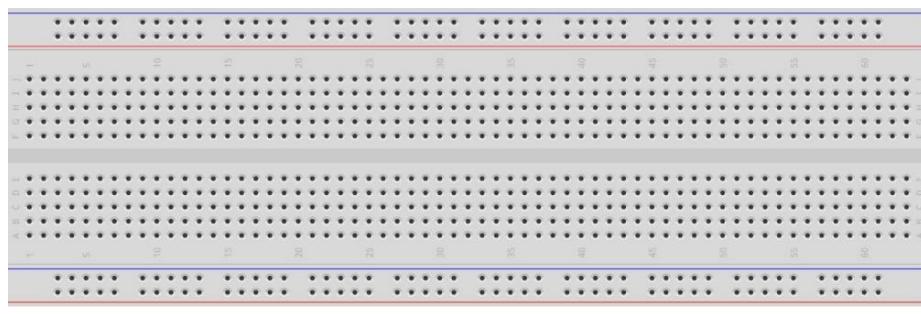
```



Project 15.2 4-Digit 7-Segment Display

Now, let us try to control a more-digit 7-segment display.

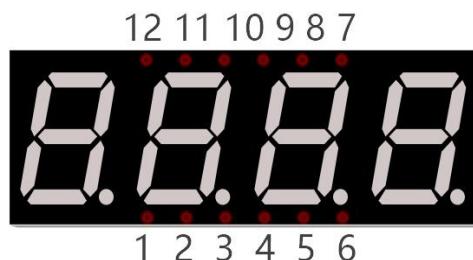
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
74HC595 x1	7-segment display x1	Resistor 220Ω x8	Jumper

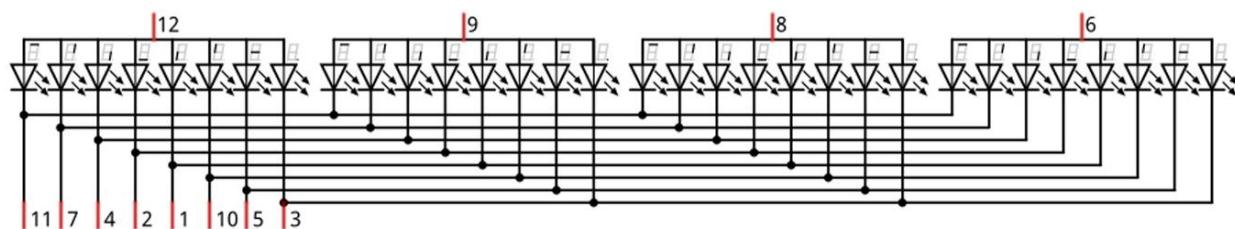
Component Knowledge

4 Digit 7-Segment Display

A 4-Digit 7-segment display integrates four 7-Segment Displays into one module. Therefore, it can display more characters. All the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



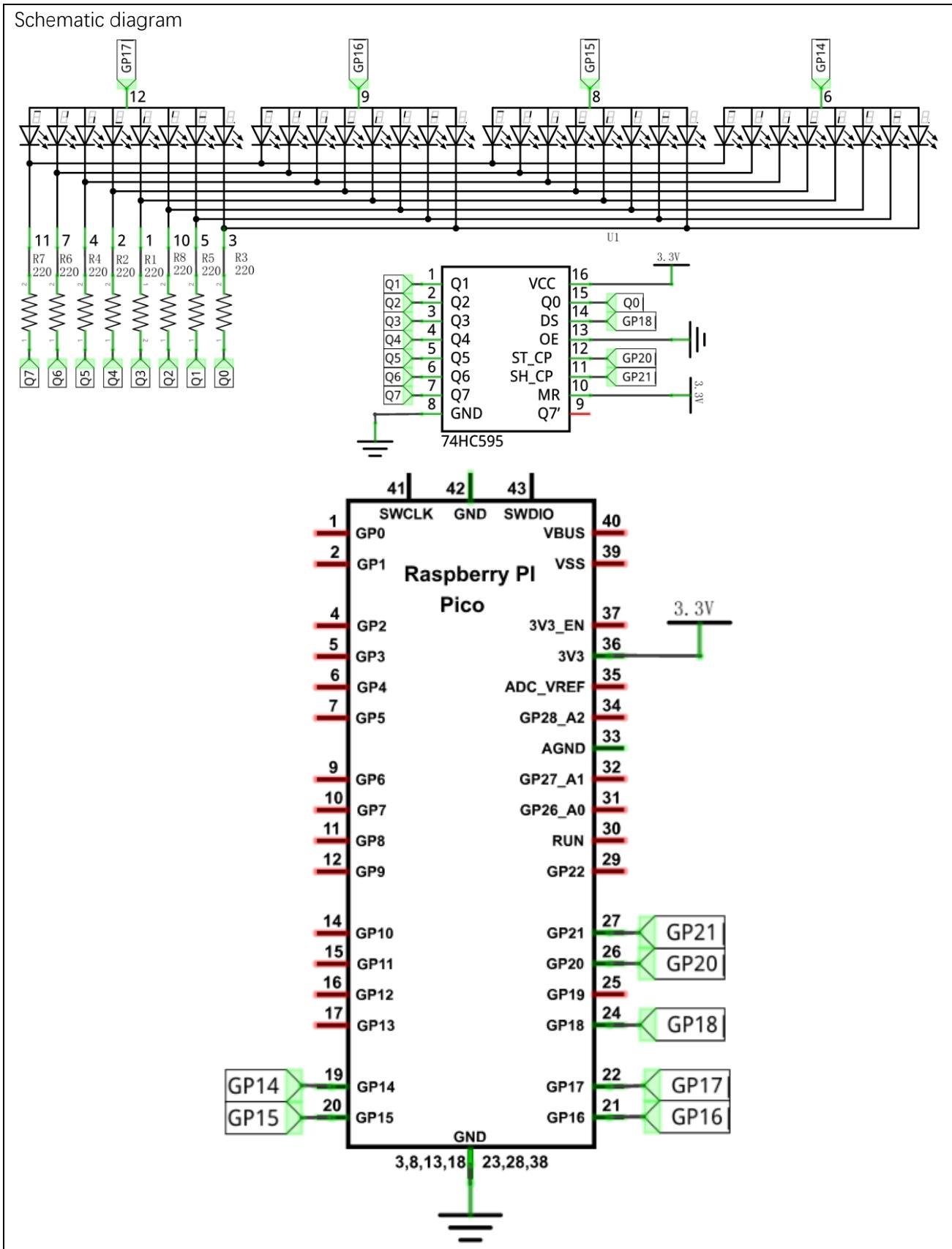
The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-Segment Display are connected together.



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

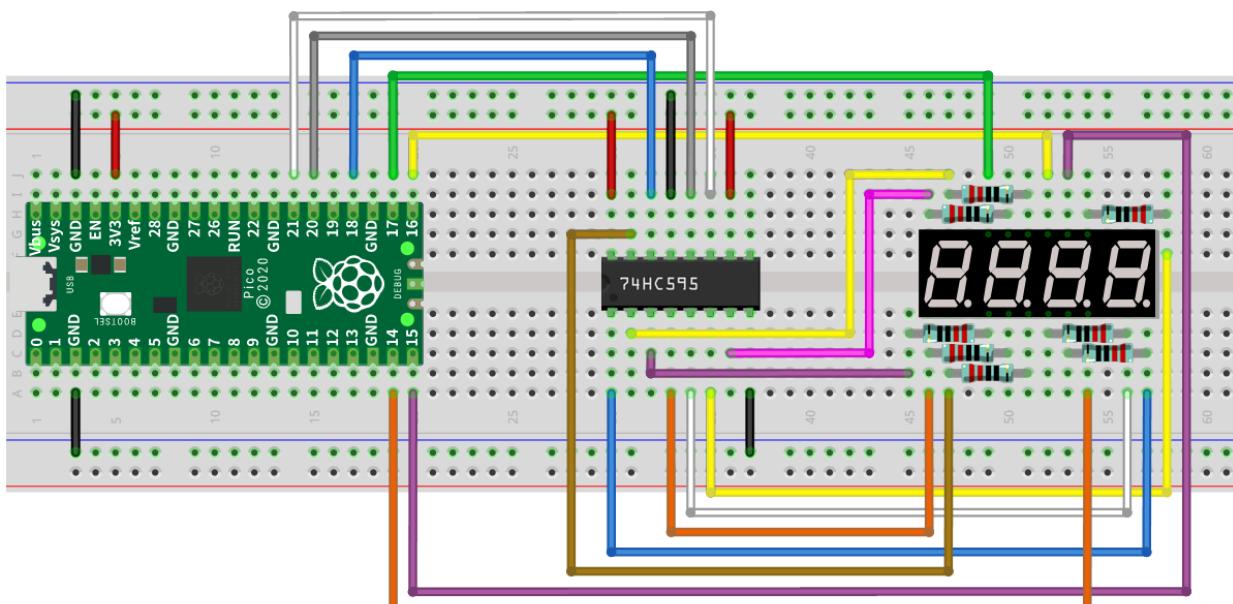
Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is imperceptible to the naked eye. This is due to the principle of optical afterglow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

Circuit



Any concerns? ✉ support@freenove.com

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



Sketch

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

[Sketch_15.2_4_Digit_7-Segment_Display](#)

```
Sketch_15.2_4_Digit_7-Segment_Display | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_15.2_4_Digit_7-Segment_Display.ino ...
16 void setup() {
17     // set pins to output
18     pinMode(latchPin, OUTPUT);
19     pinMode(clockPin, OUTPUT);
20     pinMode(dataPin, OUTPUT);
21     for (int i = 0; i < 4; i++) {
22         pinMode(comPin[i], OUTPUT);
23     }
24 }
25
26 void loop() {
27     for (int i = 0; i < 4; i++) {
28         // Select a single 7-segment display
29         electDigitalDisplay (i);
30         // Send data to 74HC595
31         writeData(num[i]);
32         delay(5);
33         // Clear the display content
34         writeData(0xff);
35     }
36 }
```

Compile and upload code to Pico, then the digital tube displays as shown.



The following is the program code:

```

1 int latchPin = 18;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 20;          // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 21;           // Pin connected to DS of 74HC595 (Pin14)
4 int comPin[] = {17, 16, 15, 14}; // Common pin (anode) of 4 digit 7-segment display
5
6 // Define the encoding of characters 0-F of the common-anode 7-Segment Display
7 byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
8                 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};
9
10 void setup() {
11     // set pins to output
12     pinMode(latchPin, OUTPUT);
13     pinMode(clockPin, OUTPUT);
14     pinMode(dataPin, OUTPUT);
15     for (int i = 0; i < 4; i++) {
16         pinMode(comPin[i], OUTPUT);
17     }
18 }
19
20 void loop() {
21     for (int i = 0; i < 4; i++) {
22         // Select a single 7-segment display
23         electDigitalDisplay (i);
24         // Send data to 74HC595
25         writeData(num[i]);
26         delay(5);
27         // Clear the display content
28         writeData(0xff);
29     }
30 }
31
32 void electDigitalDisplay(byte com) {
33     // Close all single 7-segment display
34     for (int i = 0; i < 4; i++) {
35         digitalWrite(comPin[i], LOW);
36     }

```

```

37 // Open the selected single 7-segment display
38 digitalWrite(comPin[com], HIGH);
39 }
40
41 void writeData(int value) {
42 // Make latchPin output low level
43 digitalWrite(latchPin, LOW);
44 // Send serial data to 74HC595
45 shiftOut(dataPin, clockPin, LSBFIRST, value); // Make latchPin output high level
46 // Make latchPin output high level, then 74HC595 will update data to parallel output
47 digitalWrite(latchPin, HIGH);
48 }
```

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

```

1 int latchPin = 18;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 20;          // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 21;           // Pin connected to DS of 74HC595 (Pin14)
4 int comPin[] = {17, 16, 15, 14}; // Common pin (anode) of 4 digit 7-segment display
5
6 // Define the encoding of characters 0-F of the common-anode 7-Segment Display
7 byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
8                 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};
```

Second, initialize all the pins to output mode.

```

10 void setup() {
11     // set pins to output
12     pinMode(latchPin, OUTPUT);
13     pinMode(clockPin, OUTPUT);
14     pinMode(dataPin, OUTPUT);
15     for (int i = 0; i < 4; i++) {
16         pinMode(comPin[i], OUTPUT);
17     }
18 }
```

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

```

32 void electDigitalDisplay(byte com) {
33     // Close all single 7-segment display
34     for (int i = 0; i < 4; i++) {
35         digitalWrite(comPin[i], LOW);
36     }
37     // Open the selected single 7-segment display
38     digitalWrite(comPin[com], HIGH);
39 }
```



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

```
41 void writeData(int value) {  
42     // Make latchPin output low level  
43     digitalWrite(latchPin, LOW);  
44     // Send serial data to 74HC595  
45     shiftOut(dataPin, clockPin, LSBFIRST, value);  
46     // Make latchPin output high level, then 74HC595 will update data to parallel output  
47     digitalWrite(latchPin, HIGH);  
48 }
```

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

```
20 void loop() {  
21     for (int i = 0; i < 4; i++) {  
22         // Select a single 7-segment display  
23         selectDigitalDisplay(i);  
24         // Send data to 74HC595  
25         writeData(num[i]);  
26         delay(5);  
27         // Clear the display content  
28         writeData(0xff);  
29     }  
30 }
```

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

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

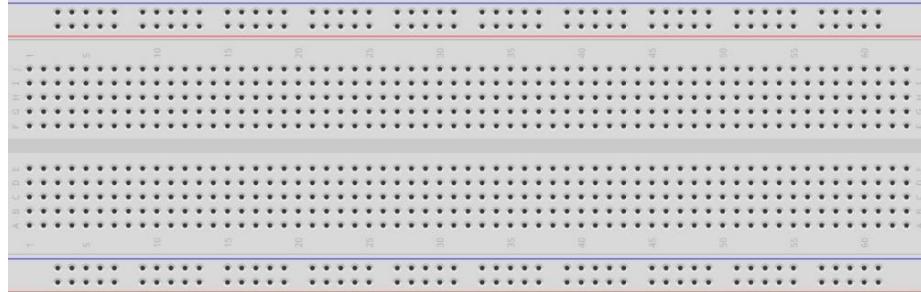
Chapter 16 74HC595 & LED Matrix

Thus far, we have learned how to use the 74HC595 IC Chip to control the Bar Graph LED and the 7-Segment Display. We will now use 74HC595 IC Chips to control an LED Matrix.

Project 16.1 LED Matrix

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

Component List

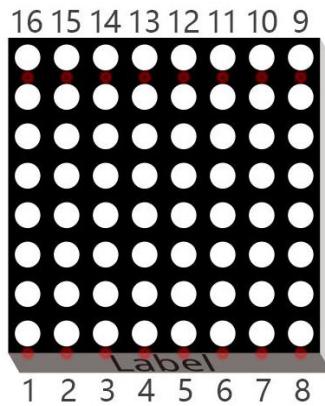
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
74HC595 x2	8*8 LED Matrix x1	Resistor 220Ω x8	Jumper



Component Knowledge

LED matrix

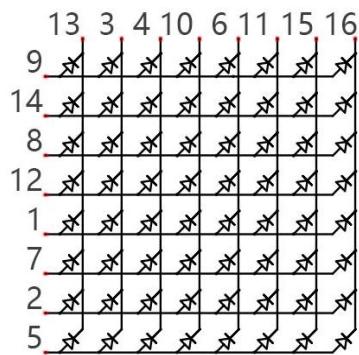
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



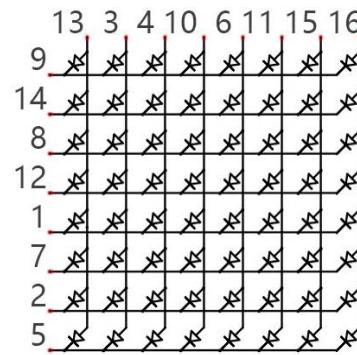
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

The LED Matrix that we use in this project is a Common Anode LED Matrix.

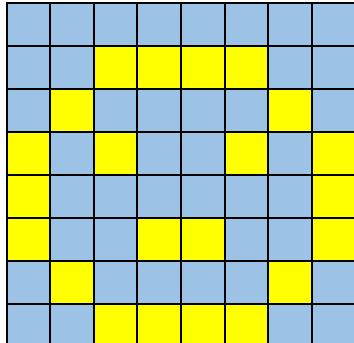
Connection mode of common anode



Connection mode of common cathode



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



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

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

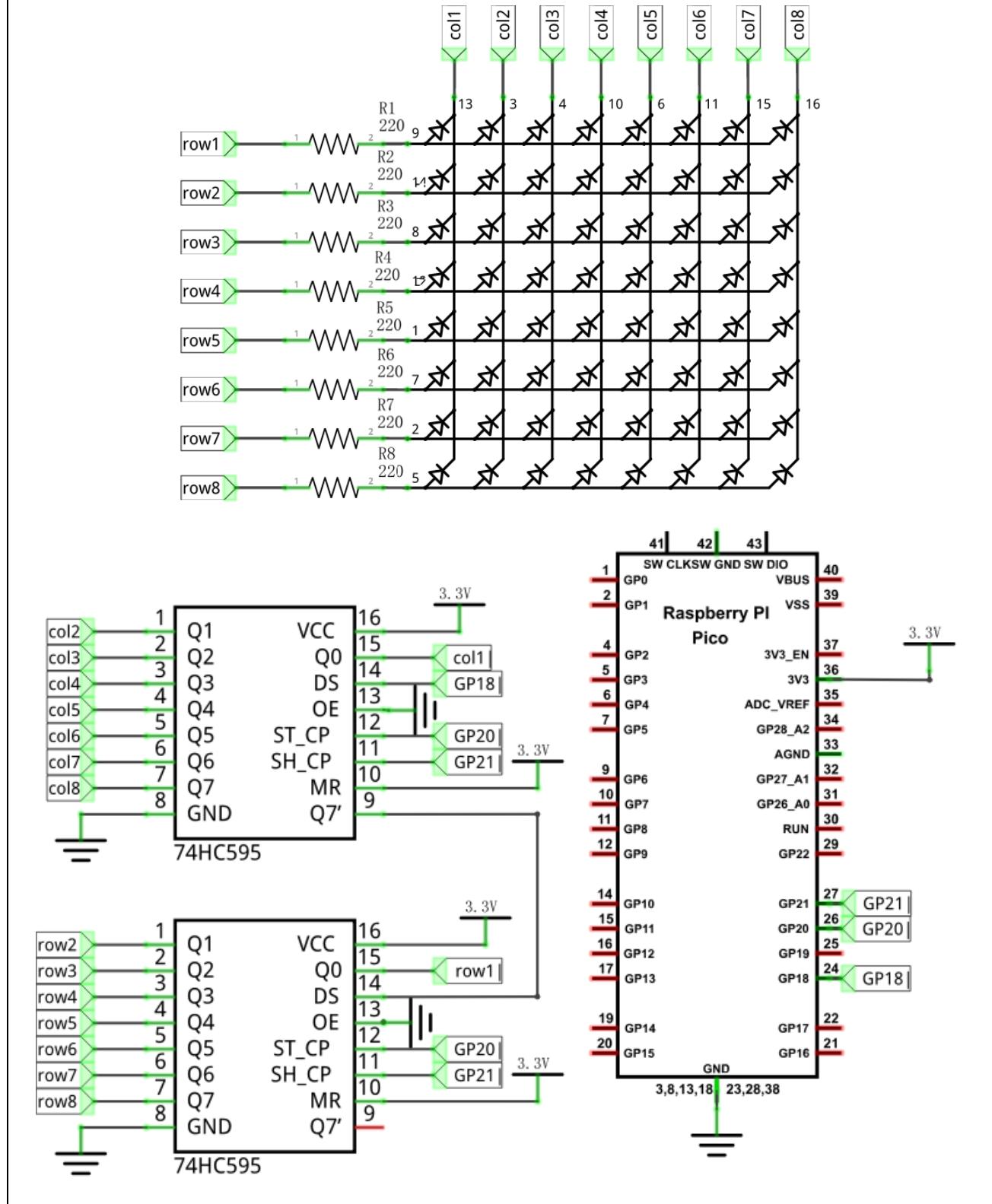
To begin, display the first column, then turn off the first column and display the second column. (and so on) turn off the seventh column and display the eighth column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated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

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

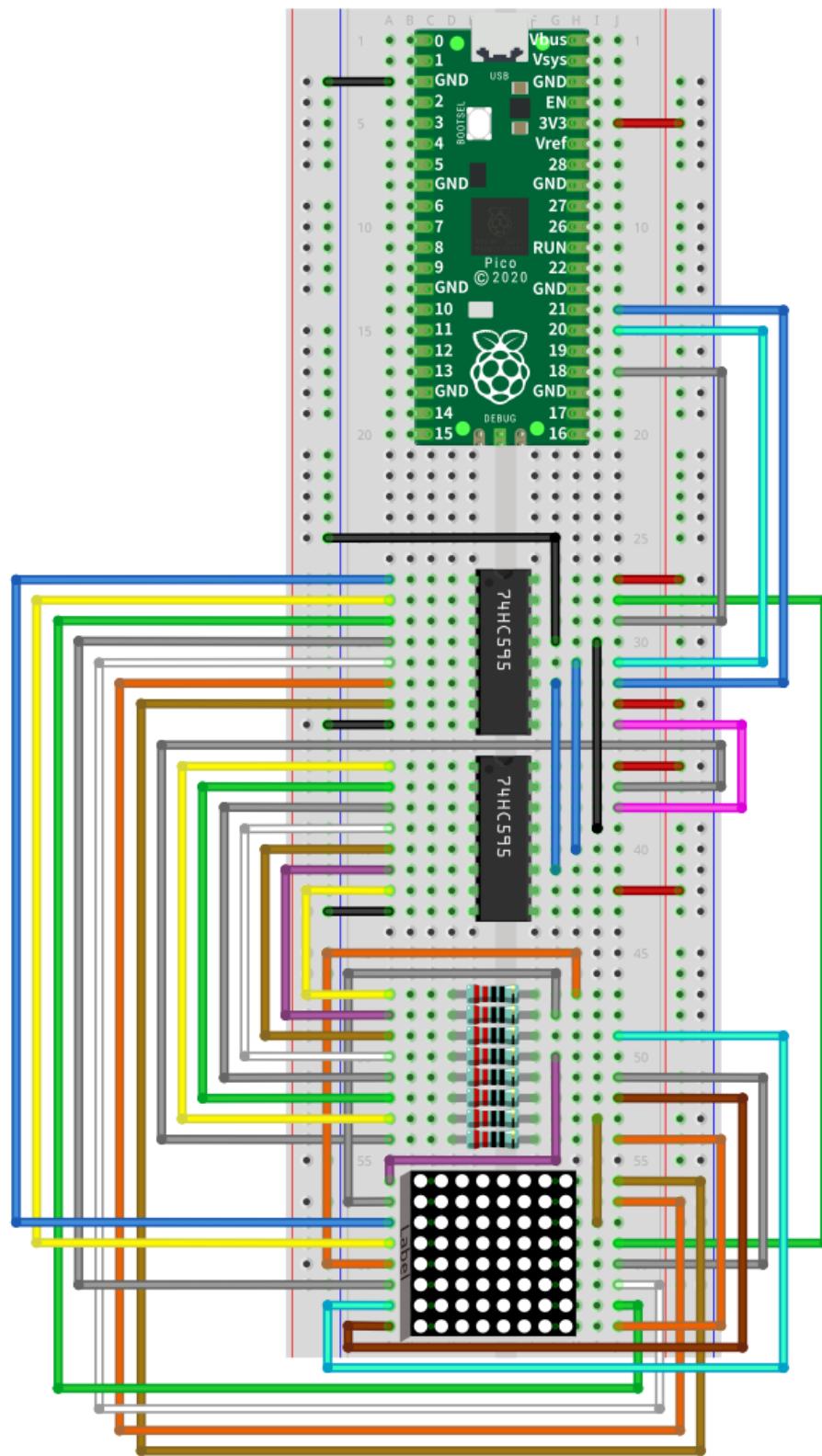
Circuit

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

Schematic diagram



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





Sketch

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

Sketch_16.1_LED_Matrix

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

- Title Bar:** Sketch_16.1_LED_Matrix | Arduino IDE 2.3.2
- Toolbar:** File, Edit, Sketch, Tools, Help
- Sketch Selection:** Raspberry Pi Pico
- Code Area:**

```

36 void setup() {
37     // set pins to output
38     pinMode(latchPin, OUTPUT);
39     pinMode(clockPin, OUTPUT);
40     pinMode(dataPin, OUTPUT);
41 }
42
43 void loop() {
44     // Define a one-byte variable (8 bits) which is used to represent
        the selected state of 8 column.
45     int cols;
46     // Display the static smiling pattern
47     for (int j = 0; j < 500; j++) { // repeat 500 times
48         cols = 0x01;
49         for (int i = 0; i < 8; i++) { // display 8 column data by
            scanning
50             matrixRowsVal(smilingFace[i]); // display the data in this column
51             matrixColsVal(~cols); // select this column
52             delay(1); // display them for a period of
            time
53             matrixRowsVal(0x00); // clear the data of this column
54             cols <= 1; // shift "cols" 1 bit left to
            select the next column
    
```
- Output Area:** Shows indexing: 1/44
- Status Bar:** Ln 89, Col 1 Raspberry Pi Pico on COM15

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

The following is the program code:

1	int latchPin = 18; // Pin connected to ST_CP of 74HC595 (Pin12)
2	int clockPin = 20; // Pin connected to SH_CP of 74HC595 (Pin11)
3	int dataPin = 21; // Pin connected to DS of 74HC595 (Pin14)

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

```
4 // Define the pattern data for a smiling face
5 const int smilingFace[] = {                                // " ^ v ^
6     0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x14
7 };
8 //
9 // Define the data of numbers and letters, and save them in flash area
10 const int data[] PROGMEM = {
11     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // "
12     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, // "1"
13     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
14     0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
15     0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
16     0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
17     0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
18     0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
19     0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
20     0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
21     0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
22     0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
23     0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
24     0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
25     0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
26     0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
27     0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00 // "F"
28 };
29
30 void setup() {
31     // set pins to output
32     pinMode(latchPin, OUTPUT);
33     pinMode(clockPin, OUTPUT);
34     pinMode(dataPin, OUTPUT);
35 }
36
37 void loop() {
38     // Define a one-byte variable (8 bits) which is used to represent the selected state of 8
39     // column.
40     int cols;
41     // Display the static smiling pattern
42     for (int j = 0; j < 500; j++) { // repeat 500 times
43         cols = 0x01;
44         for (int i = 0; i < 8; i++) { // display 8 column data by scanning
45             matrixRowsVal(smilingFace[i]); // display the data in this column
46             matrixColsVal(~cols); // select this column
47             delay(1); // display them for a period of time
48     }
49 }
```

```

47     matrixRowsVal(0x00);           // clear the data of this column
48     cols <= 1;                   // shift "cols" 1 bit left to select the next column
49 }
50 }
51 // Display the dynamic patterns of numbers and letters
52 for (int i = 0; i < 128; i++) {
53     for (int k = 0; k < 10; k++) {    // repeat image of each frame 10 times.
54         cols = 0x01;             // Assign binary 00000001. Means the first column is selected.
55         for (int j = i; j < 8 + i; j++) { // display image of each frame
56             matrixRowsVal(pgm_read_word_near(data + j)); // display the data in this column
57             matrixColsVal(~cols);           // select this column
58             delay(1);                  // display them for a period of time
59             matrixRowsVal(0x00);           // close the data of this column
60             cols <= 1;                   // shift "cols" 1 bit left to select the next column
61     }
62 }
63 }
64 }

65
66 void matrixRowsVal(int value) {
67     // make latchPin output low level
68     digitalWrite(latchPin, LOW);
69     // Send serial data to 74HC595
70     shiftOut(dataPin, clockPin, LSBFIRST, value);
71     // make latchPin output high level, then 74HC595 will update the data to parallel output
72     digitalWrite(latchPin, HIGH);
73 }

74
75 void matrixColsVal(int value) {
76     // make latchPin output low level
77     digitalWrite(latchPin, LOW);
78     // Send serial data to 74HC595
79     shiftOut(dataPin, clockPin, MSBFIRST, value);
80     // make latchPin output high level, then 74HC595 will update the data to parallel output
81     digitalWrite(latchPin, HIGH);
82 }

```

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

```

39 int cols;
40 // Display the static smiling pattern
41 for (int j = 0; j < 500; j++) { // repeat 500 times
42     cols = 0x01;
43     for (int i = 0; i < 8; i++) { // display 8 column data by scanning

```

```

44     matrixRowsVal(smilingFace[i]); // display the data in this column
45     matrixColsVal(~cols);        // select this column
46     delay(1);                  // display them for a period of time
47     matrixRowsVal(0x00);        // clear the data of this column
48     cols <= 1;                 // shift "cols" 1 bit left to select the next column
49   }
50 }
```

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

```

51 // Display the dynamic patterns of numbers and letters
52 for (int i = 0; i < 128; i++) {
53   for (int k = 0; k < 10; k++) {           // repeat image of each frame 10 times.
54     cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
55     for (int j = i; j < 8 + i; j++) { // display image of each frame
56       matrixRowsVal(pgm_read_word_near(data + j)); // display the data in this column
57       matrixColsVal(~cols);          // select this column
58       delay(1);                   // display them for a period of time
59       matrixRowsVal(0x00);         // close the data of this column
60       cols <= 1;                 // shift "cols" 1 bit left to select the next column
61     }
62   }
63 }
```

In this example, we use two 74HC595 to drive the LED matrix, requiring only 3 pins, so that we could save the rest of 13 pins.

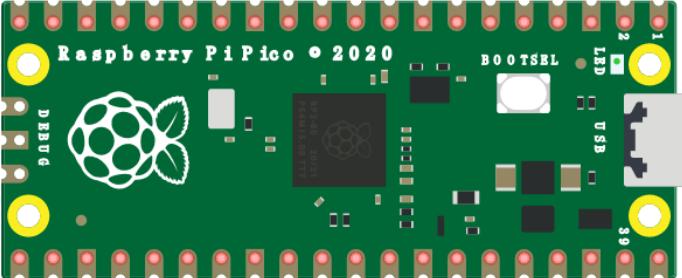
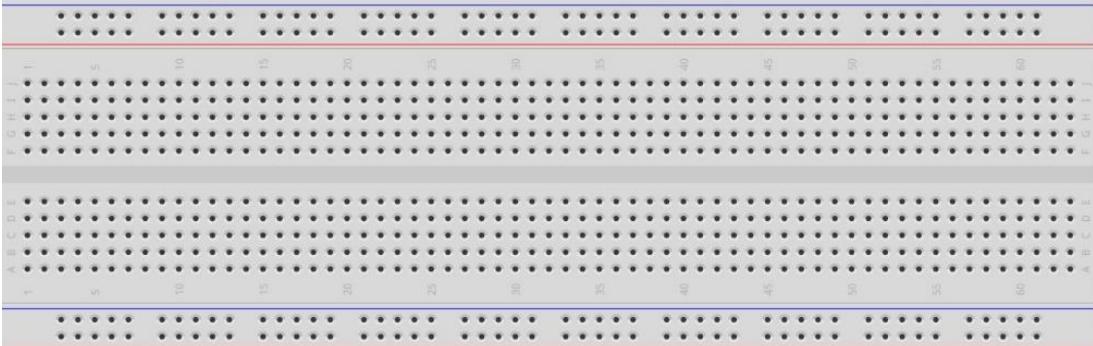
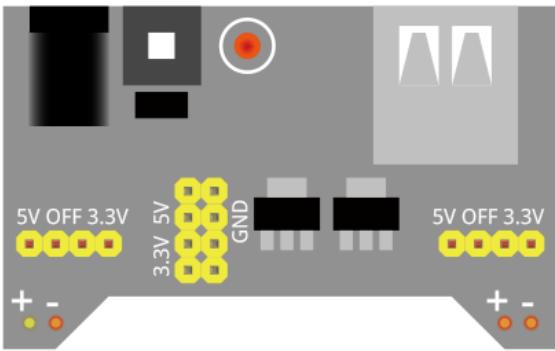
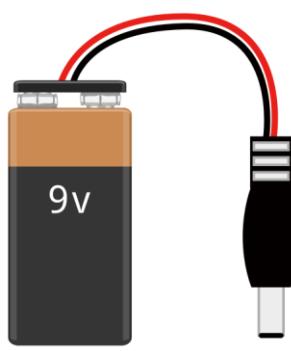
Chapter 17 Relay & Motor

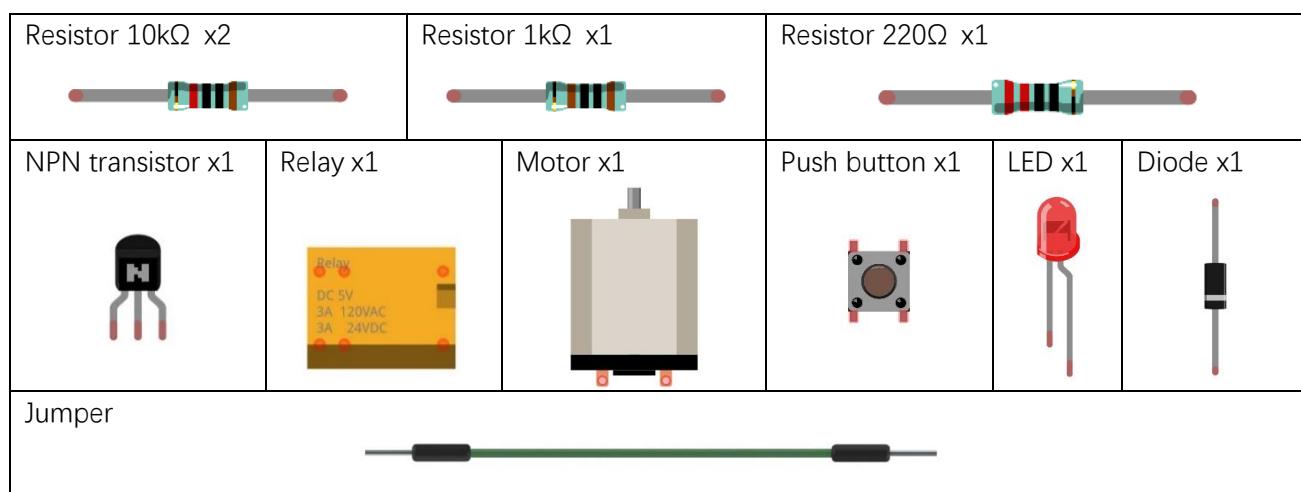
In this chapter, we will learn a kind of special switch module, Relay Module.

Project 17.1 Relay & Motor

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

Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Breadboard Power x1		9V battery (prepared by yourself) & battery line x1 

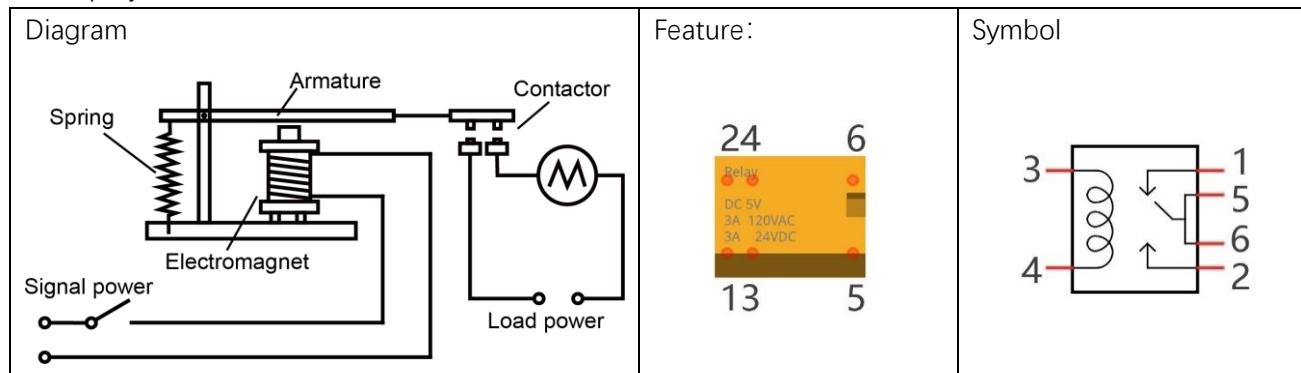


Component Knowledge

Relay

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

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



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

Inductor

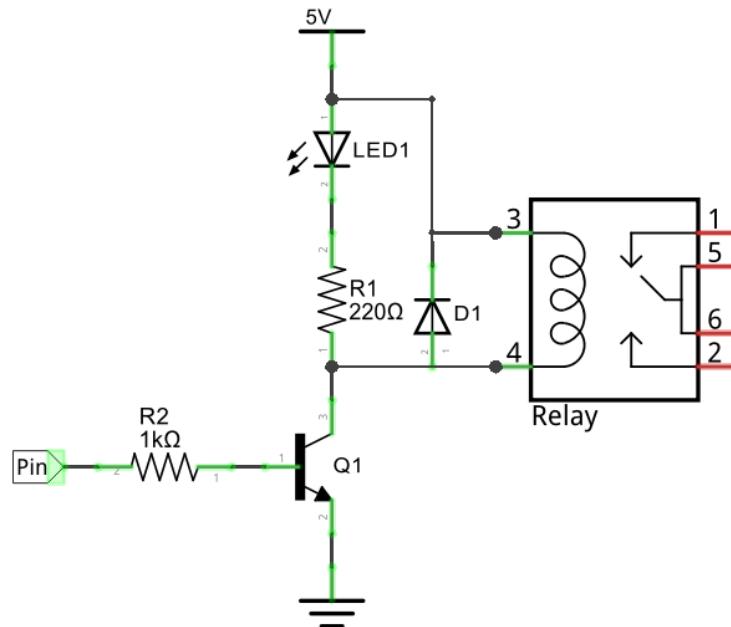
The symbol of Inductance is “L” and the unit of inductance is the “Henry” (H). Here is an example of how this can be encountered: $1H=1000mH$, $1mH=1000\mu H$.

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



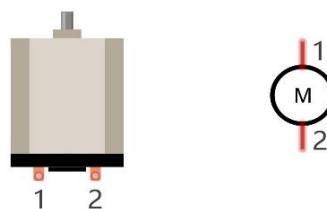


The reference circuit for relay is as follows. The coil of relays can be equivalent to that of inductors, when the transistor disconnects power supply of the relay, the current in the coil of the relay cannot stop immediately, causing an impact on power supply. Therefore, a parallel diode will be connected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

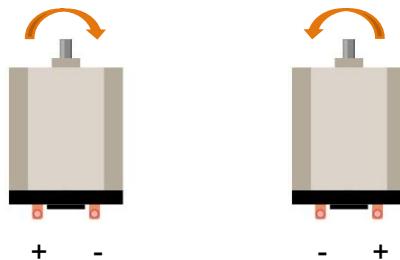


Motor

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

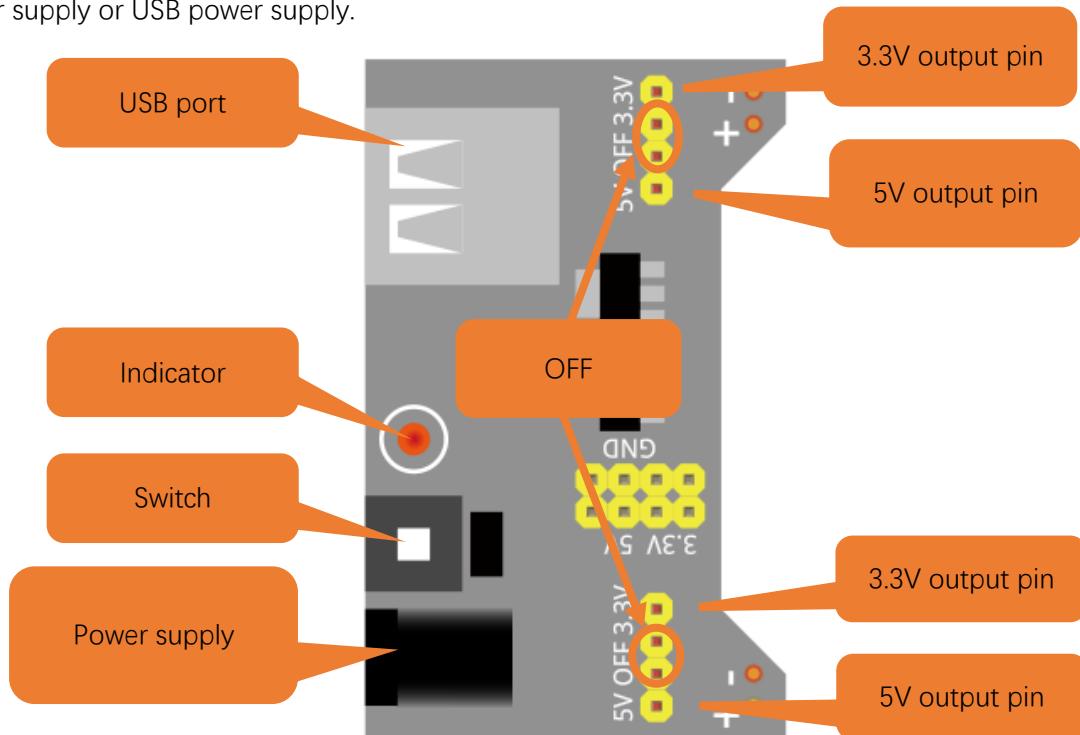


When a motor is connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.



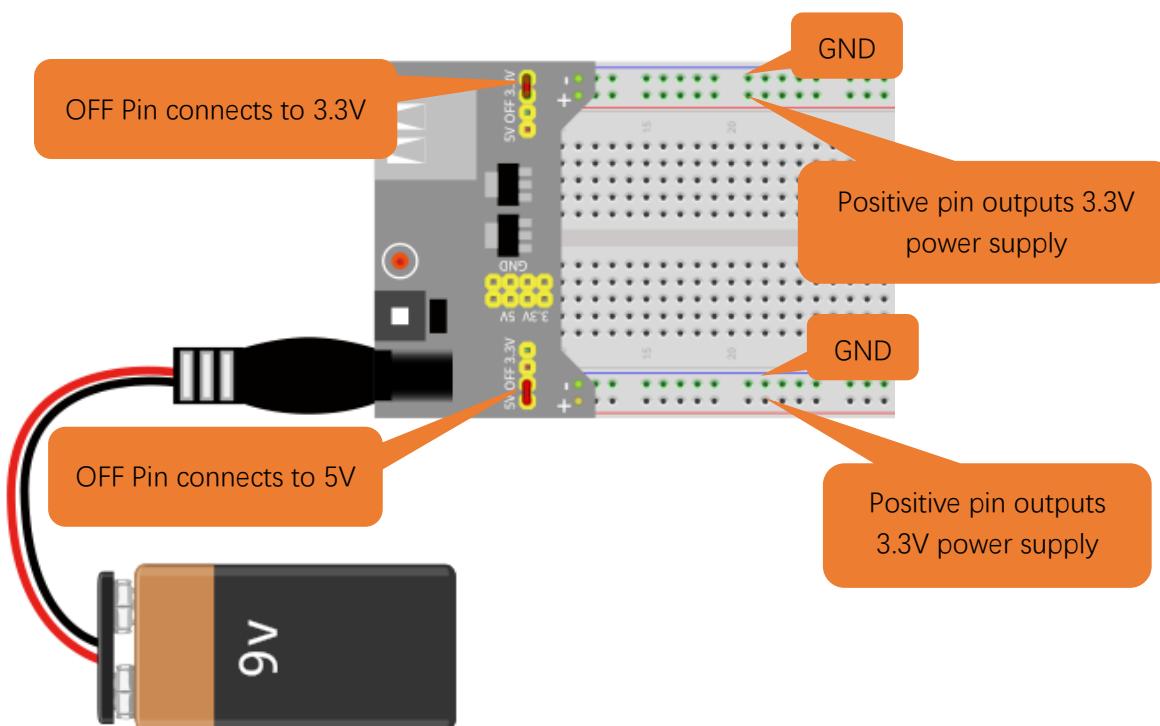
Breadboard Power

When the Raspberry Pi Pico outputs insufficient power or the power supply voltage and power consumption required by the device or module exceeds that provided by the Raspberry Pi Pico, you can choose Breadboard Power to either output 3.3V voltage source or 5V voltage source. Input power: DC 6~10V power supply or USB power supply.



Usage:

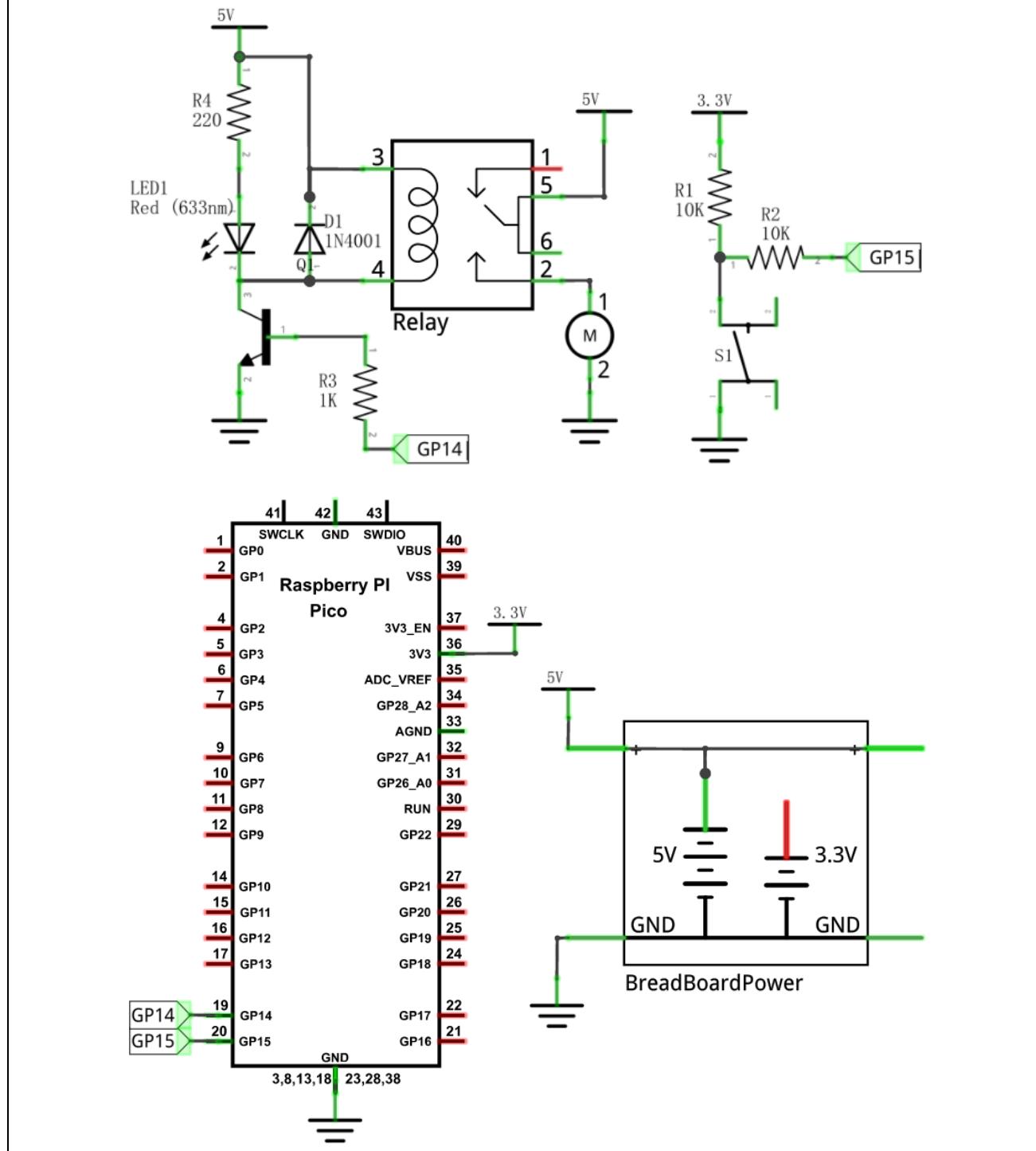
First connect Breadboard Power to the power supply, and then selectively choose the connection between OFF and 5V, 3V to generate different voltage source outputs, as the picture shows.



You can either select only one side of the output power supply, or choose to output 5V or 3.3V power supply at the same time on both sides, because the output circuits on both sides are separated and controlled independently and will not cause interference. Note that the GND should be connected to any GND of the Raspberry Pi Pico to form a Common ground connection.

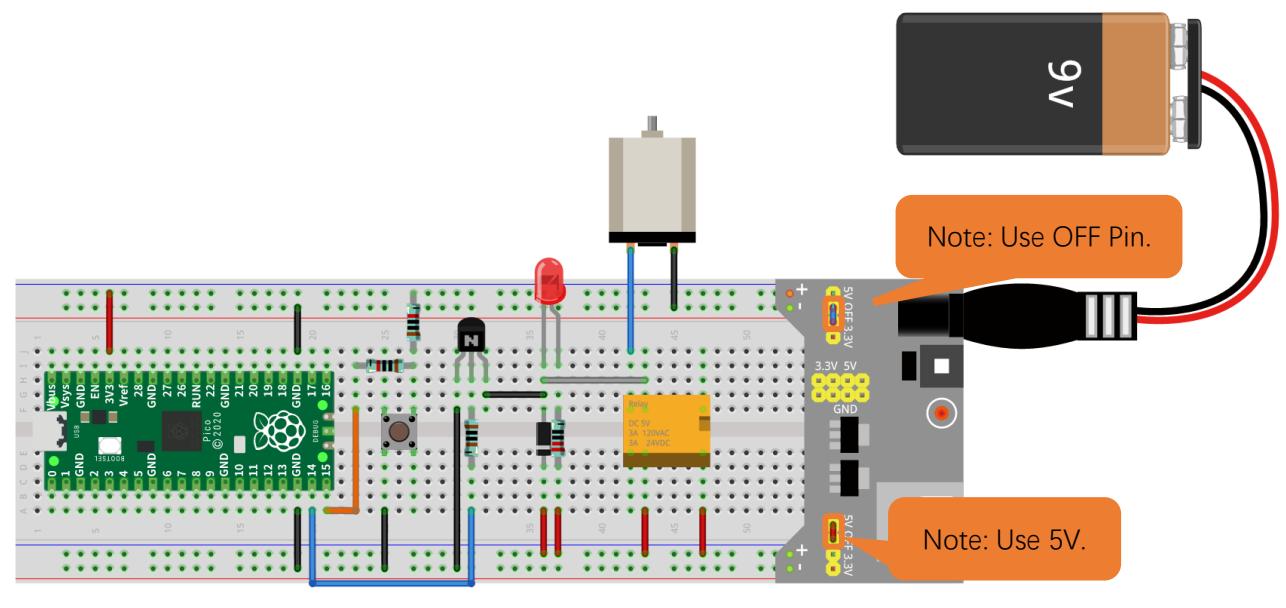
Circuit

Schematic diagram



Any concerns? ✉ support@freenove.com

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





Sketch

Use buttons to control the relays and motors.

Sketch_17.1_Control_Motor_by_Relay

The screenshot shows the Arduino IDE interface with the sketch file "Sketch_17.1_Control_Motor_by_Relay.ino" open. The code is written for the Raspberry Pi Pico. It defines relay and button pins, initializes them, and sets up the push button pin as INPUT_PULLUP and the relay pin as OUTPUT. The setup function also configures the serial port for output. The code is as follows:

```
Sketch_17.1_Control_Motor_by_Relay | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_17.1_Control_Motor_by_Relay.ino
1 int relayPin = 14;           // the number of the relay pin
2 int buttonPin = 15;          // the number of the push button pin
3
4 int buttonState = HIGH;      // Record button state, and initial the
5 state to high level
6 int relayState = LOW;        // Record relay state, and initial the
7 state to low level
8 int lastButtonstate = HIGH; // Record the button state of last
9 detection
10 long lastChangeTime = 0;    // Record the time point for button state
11 change
12
13 void setup() {
14     pinMode(buttonPin, INPUT_PULLUP);           // Set push button pin
15     into input mode
16     pinMode(relayPin, OUTPUT);                 // Set relay pin into
17     output mode
18
19     Serial.begin(9600);                         // Set serial port for output
20 }
```

Output

Ln 1, Col 1 Raspberry Pi Pico on COM15

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



The following is the program code:

```
1 int relayPin = 14;           // the number of the relay pin
2 int buttonPin = 15;          // the number of the push button pin
3
4 int buttonState = HIGH;      // Record button state, and initial the state to high level
5 int relayState = LOW;        // Record relay state, and initial the state to low level
6 int lastButtonState = HIGH; // Record the button state of last detection
7 long lastChangeTime = 0;    // Record the time point for button state change
8
9 void setup() {
10    pinMode(buttonPin, INPUT_PULLUP);           // Set push button pin into input mode
11    pinMode(relayPin, OUTPUT);                  // Set relay pin into output mode
12    digitalWrite(relayPin, relayState);         // Set the initial state of relay into "off"
13 }
14 void loop() {
15    int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
16    // If button pin state has changed, record the time point
17    if (nowButtonState != lastButtonState) {
18        lastChangeTime = millis();
19    }
20    // If button state changes, and stays stable for a while, then it should have skipped the
21    // bounce area
22    if (millis() - lastChangeTime > 10) {
23        if (buttonState != nowButtonState) { // Confirm button state has changed
24            buttonState = nowButtonState;
25            if (buttonState == LOW) {        // Low level indicates the button is pressed
26                relayState = ! relayState;   // Reverse relay state
27                digitalWrite(relayPin, relayState); // Update relay state
28            }
29        }
30    lastButtonState = nowButtonState; // Save the state of last button
31 }
```



In Chapter 2, the pressing and releasing of the button will result in mechanical vibrating. If we do not solve this problem, some unexpected consequences may happen to the procedure.

Click [here](#) to return to Chapter 2 Button & LED.

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

```
15 int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
16 // If button pin state has changed, record the time point
17 if (nowButtonState != lastButtonState) {
18     lastChangeTime = millis();
19 }
```

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

```
20 // If button state changes, and stays stable for a while, then it should have skipped the
bounce area
21 if (millis() - lastChangeTime > 10) {
22     if (buttonState != nowButtonState) { // Confirm button state has changed
23         buttonState = nowButtonState;
24         if (buttonState == LOW) { // Low level indicates the button is pressed
25             relayState = ! relayState; // Reverse relay state
26             digitalWrite(relayPin, relayState); // Update relay state
27         }
28     }
29 }
30 lastButtonState = nowButtonState; // Save the state of last button
```

Chapter 18 L293D & Motor

Project 18.1 Control Motor with Potentiometer

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

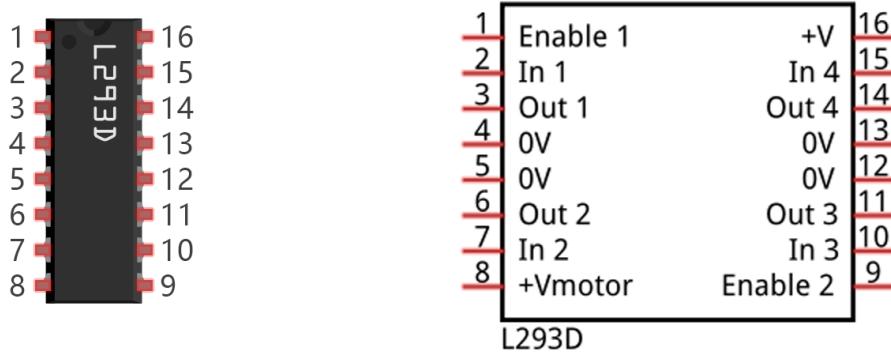
Component List

Raspberry Pi Pico x1	USB cable x1
Breadboard x1	
Rotary potentiometer x1	Motor x1
L293D x1	
Jumper	Battery box x1

Component Knowledge

L293D

L293D is an IC Chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a Unidirectional DC Motor with 4 ports or a Bi-Directional DC Motor with 2 ports or a Stepper Motor (Stepper Motors are covered later in this Tutorial).



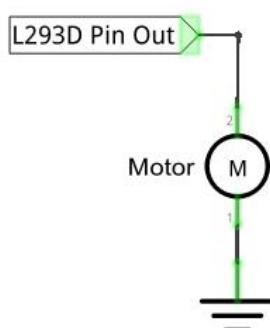
Port description of L293D module is as follows:

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

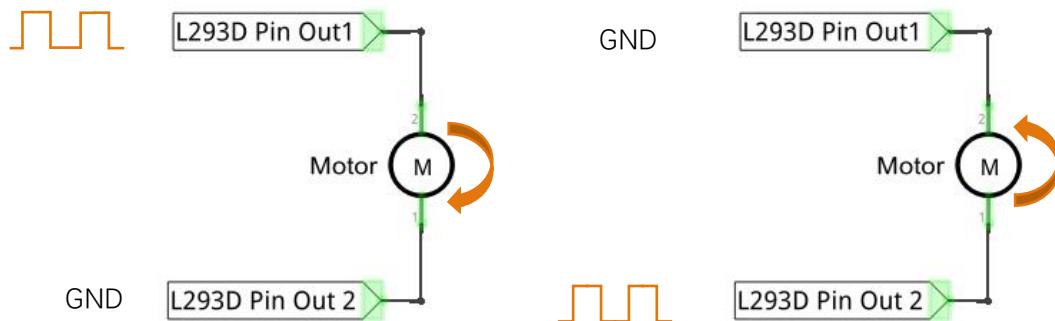
For more details, please refer to the datasheet for this IC Chip.

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

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

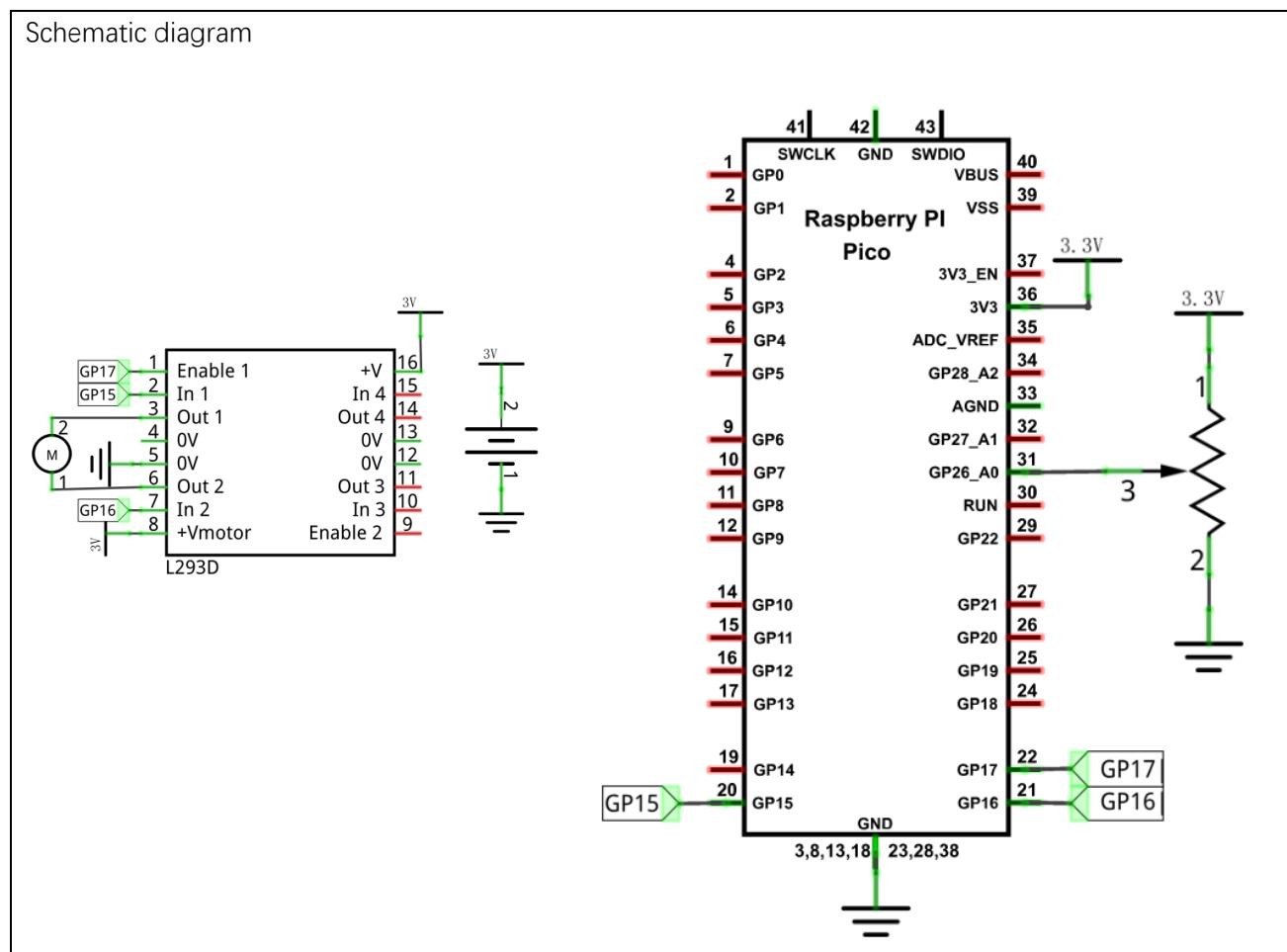


The following connection uses two channels of the L293D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the direction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

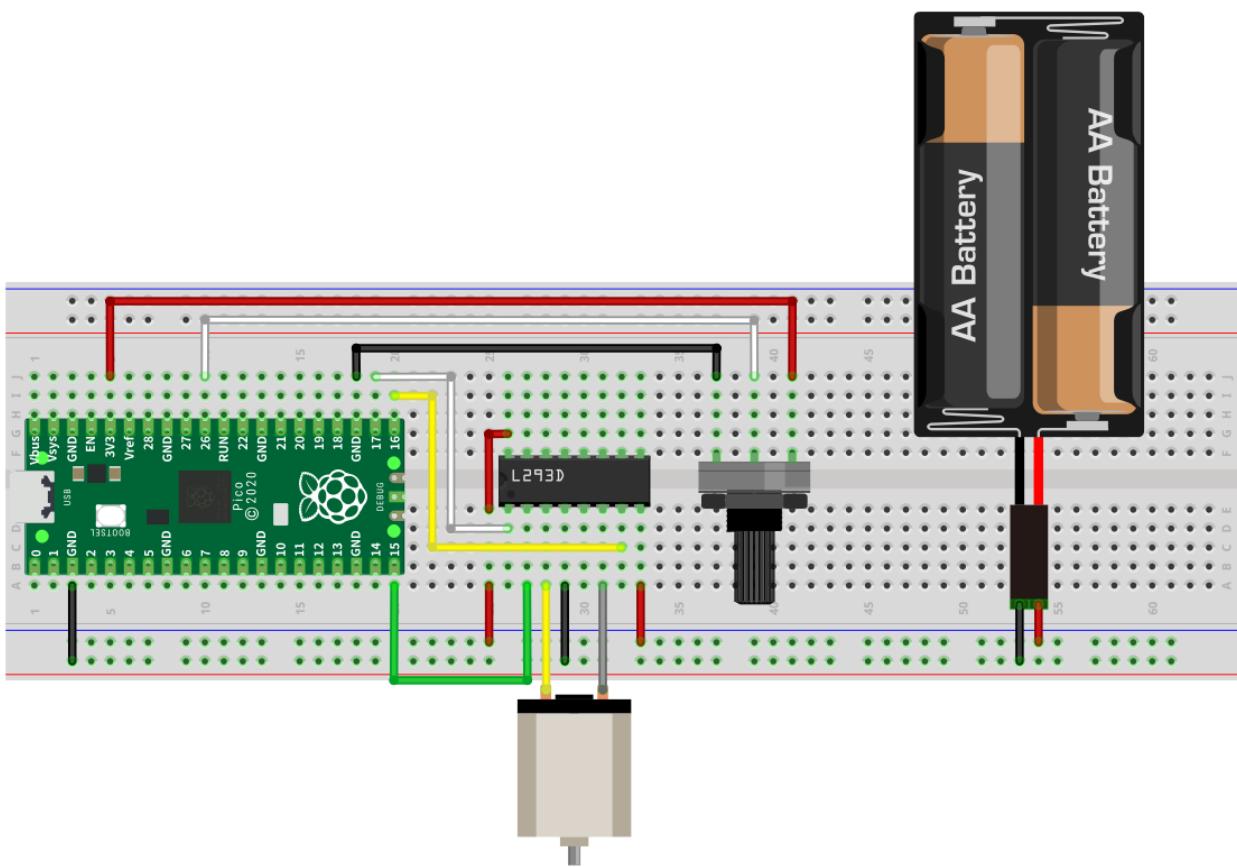


In practical use, the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

Circuit



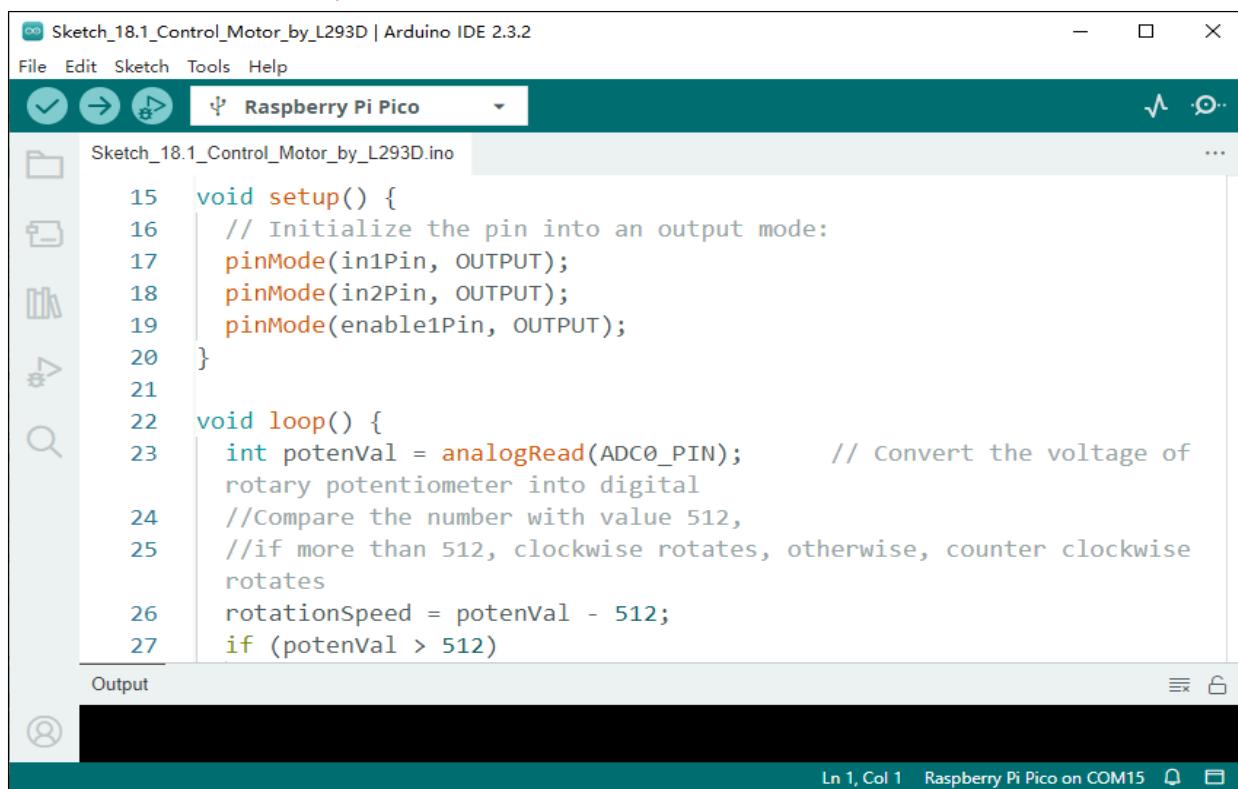
Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

Sketch

Sketch_18.1_Control_Motor_by_L293D



The screenshot shows the Arduino IDE interface with the sketch `Sketch_18.1_Control_Motor_by_L293D.ino` open. The code is as follows:

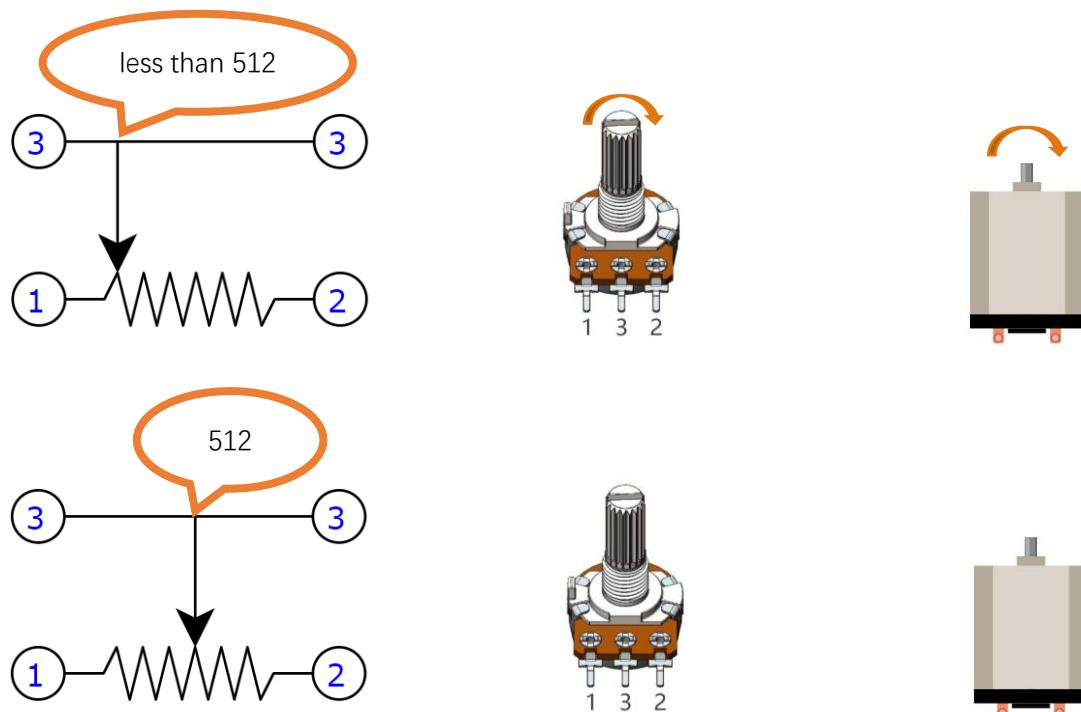
```

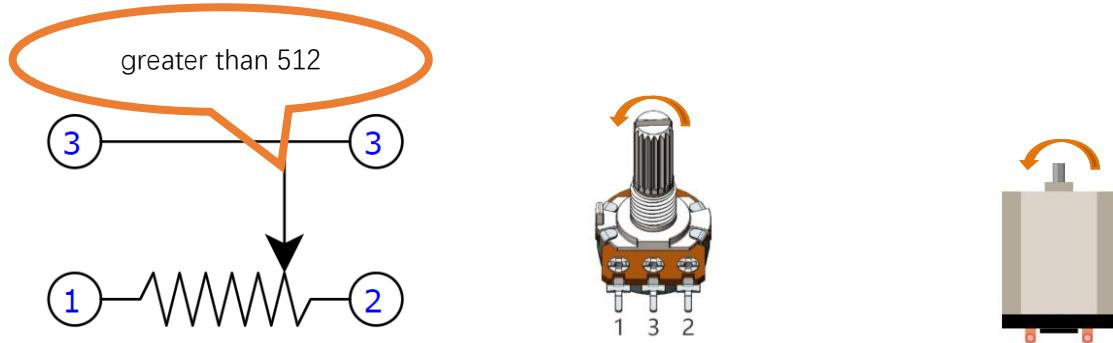
15 void setup() {
16     // Initialize the pin into an output mode:
17     pinMode(in1Pin, OUTPUT);
18     pinMode(in2Pin, OUTPUT);
19     pinMode(enable1Pin, OUTPUT);
20 }
21
22 void loop() {
23     int potenVal = analogRead(ADC0_PIN);      // Convert the voltage of
                                                 // rotary potentiometer into digital
24     //Compare the number with value 512,
25     //if more than 512, clockwise rotates, otherwise, counter clockwise
     //rotates
26     rotationSpeed = potenVal - 512;
27     if (potenVal > 512)

```

The code initializes three pins as outputs and reads the value from an analog pin (ADC0_PIN). It then compares this value to 512 to determine the direction of rotation for the motor.

Download code to Pico, rotate the potentiometer in one direction, and the motor speeds up slowly in one direction. Then rotate the potentiometer in the other direction and the motor will slow down to stop. Then rotate it in an inverse direction to accelerate the motor.





The following is the sketch:

```

1 int in1Pin = 15;           // Define L293D channel 1 pin
2 int in2Pin = 16;           // Define L293D channel 2 pin
3 int enable1Pin = 17;        // Define L293D enable 1 pin
4 int ADC0_PIN = 26;

5
6 boolean rotationDir; // Define a variable to save the motor's rotation direction
7 int rotationSpeed;    // Define a variable to save the motor rotation speed
8
9 void setup() {
10 // Initialize the pin into an output mode:
11 pinMode(in1Pin, OUTPUT);
12 pinMode(in2Pin, OUTPUT);
13 pinMode(enable1Pin, OUTPUT);
14 }
15
16 void loop() {
17 int potenVal = analogRead(ADC0_PIN);      // Convert the voltage of rotary potentiometer
into digital
18 //Compare the number with value 512,
19 //if more than 512, clockwise rotates, otherwise, counter clockwise rotates
20 rotationSpeed = potenVal - 512;
21 if (potenVal > 512)
22     rotationDir = true;
23 else
24     rotationDir = false;
25 // Calculate the motor speed
26 rotationSpeed = abs(potenVal - 512);
27 //Control the steering and speed of the motor
28 driveMotor(rotationDir, constrain(rotationSpeed, 0, 512));
29 }
30
31 void driveMotor(boolean dir, int spd) {
32 // Control motor rotation direction

```

```

33   if (dir) {
34     digitalWrite(in1Pin, HIGH);
35     digitalWrite(in2Pin, LOW);
36   }
37   else {
38     digitalWrite(in1Pin, LOW);
39     digitalWrite(in2Pin, HIGH);
40   }
41   // Control motor rotation speed
42   analogWrite(enable1Pin, spd);
43 }
```

The ADC of Pico has a 10-bit accuracy, corresponding to a range from 0 to 1023. In this program, set the number 512 as the midpoint. If the value of ADC is less than 512, make the motor rotate in one direction. If the value of ADC is greater than 512, make the motor rotate in the other direction. Subtract 512 from the ADC value and take the absolute value and use this result as the speed of the motor.

```

17   int potenVal = analogRead(ADC0_PIN);      // Convert the voltage of rotary potentiometer
into digital
18   //Compare the number with value 512,
19   //if more than 512, clockwise rotates, otherwise, counter clockwise rotates
20   rotationSpeed = potenVal - 512;
21   if (potenVal > 512)
22     rotationDir = true;
23   else
24     rotationDir = false;
25   // Calculate the motor speed
26   rotationSpeed = abs(potenVal - 512);
27   //Control the steering and speed of the motor
28   driveMotor(rotationDir, constrain(rotationSpeed, 0, 512));
```

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

```

31 void driveMotor(boolean dir, int spd) {
32   // Control motor rotation direction
33   if (dir) {
34     digitalWrite(in1Pin, HIGH);
35     digitalWrite(in2Pin, LOW);
36   }
37   else {
38     digitalWrite(in1Pin, LOW);
39     digitalWrite(in2Pin, HIGH);
40   }
41   // Control motor rotation speed
42   analogWrite(enable1Pin, spd);
43 }
```



Chapter 19 Servo

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

Project 19.1 Servo Sweep

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

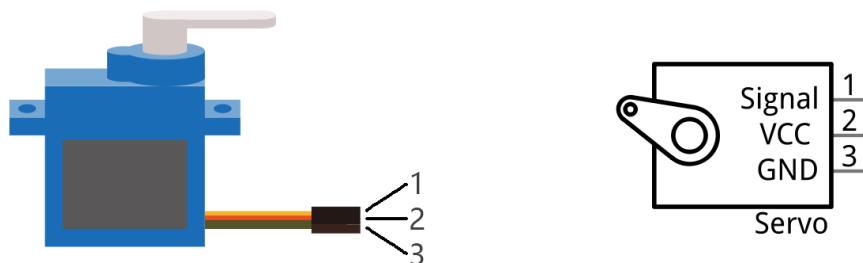
Component List

Raspberry Pi Pico x1	USB cable x1
A green printed circuit board (PCB) for the Raspberry Pi Pico. It features a central Broadcom SoC, a USB Type-C port, and several pins labeled with letters A through J and numbers 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, and 60.	Two standard black USB cables, each with a black USB-A male connector at one end and a grey USB-B male connector at the other.
Breadboard x1	A schematic diagram of a breadboard. It shows two parallel rows of 40 numbered holes (5 to 60) with vertical columns labeled A through J between them. Horizontal red and blue lines indicate power rails.
Servo x1	Jumper

Component Knowledge

Servo

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



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

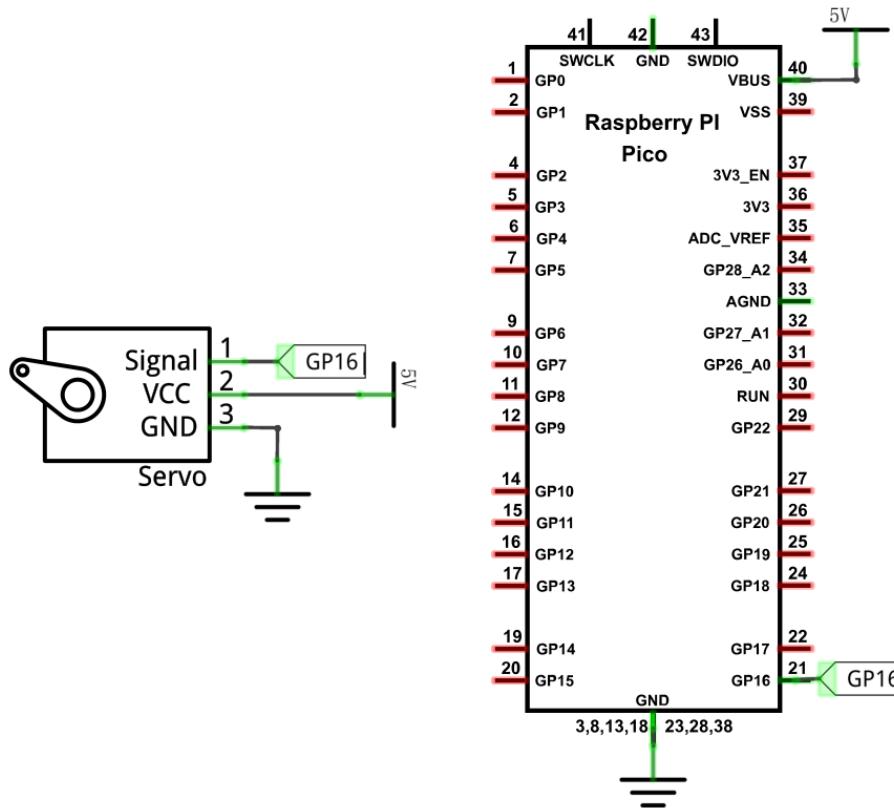
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

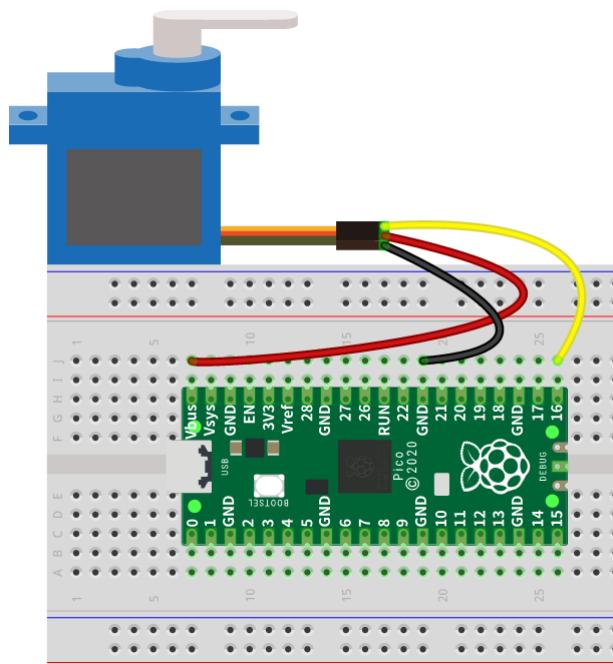
Circuit

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

Schematic diagram



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



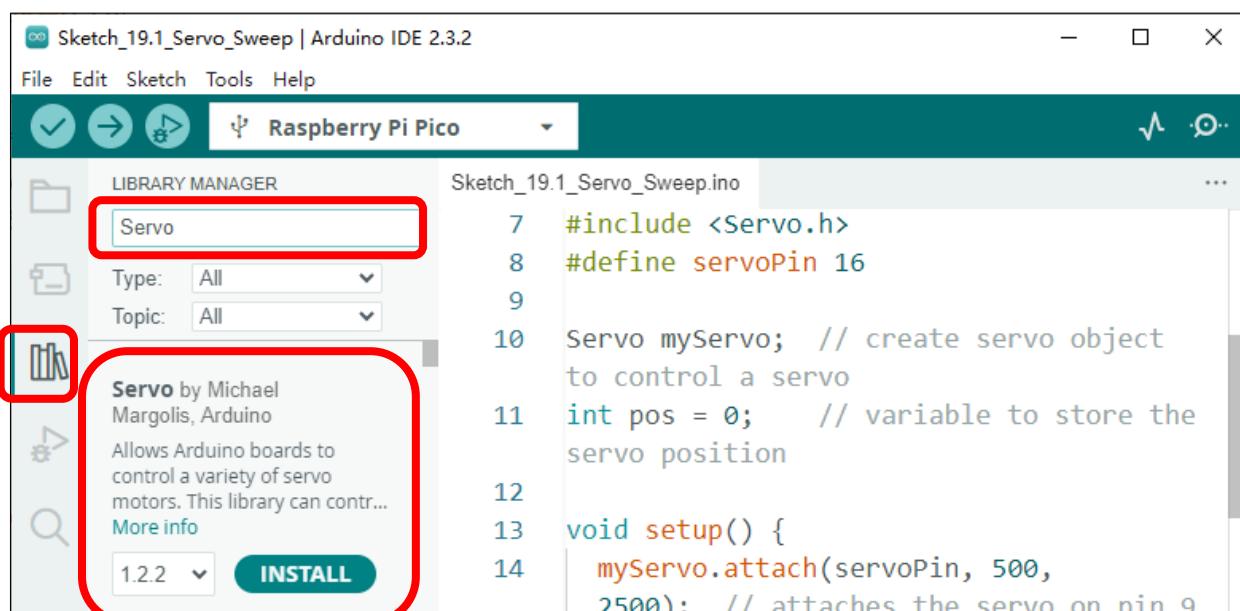
Any concerns? ✉ support@freenove.com

Sketch

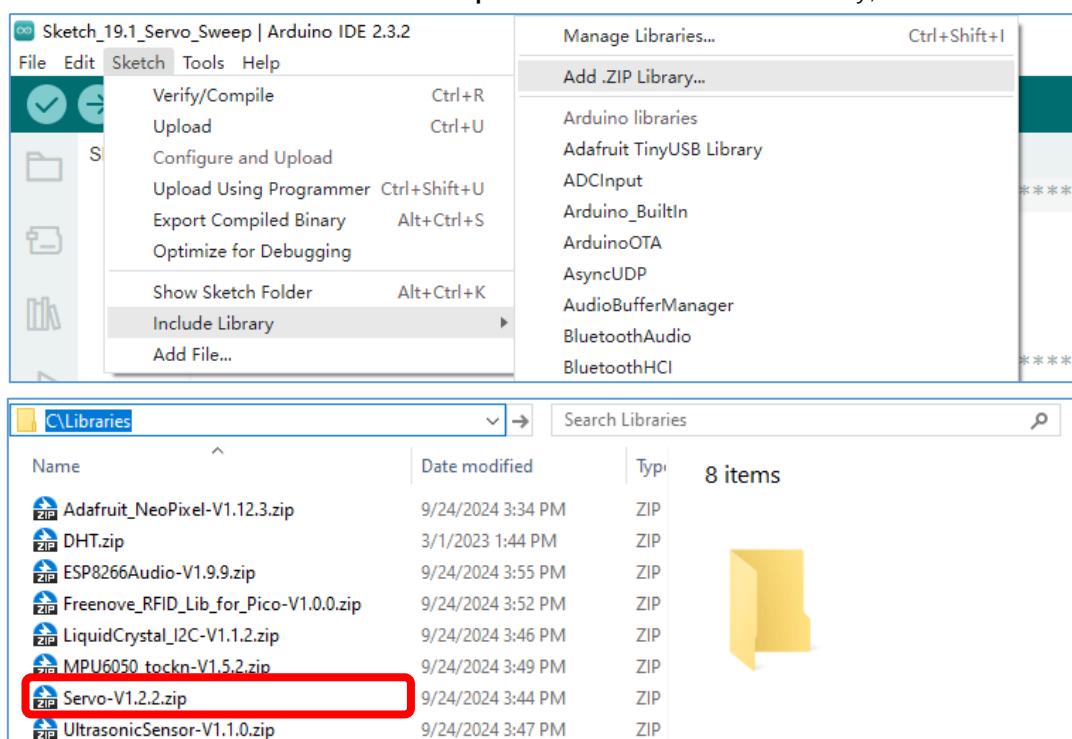
How to install the library

If you have not installed the Servo library, please do so first.

The first one: Open Arduino IDE, click **Library Manager** on the left, input “Servo” in the search bar, select it to install.



The second way, open Arduino IDE, click Sketch → Include Library → Add .ZIP Library. In the pop-up window, find the file named “./Libraries/ **Servo-V1.2.2.Zip**” which locates in this directory, and click OPEN.



Use the Servo library to control the servomotor and let the servomotor rotate back and forth.

Sketch_19.1_Servo_Sweep



The screenshot shows the Arduino IDE interface with the sketch `Sketch_19.1_Servo_Sweep.ino` open. The code is as follows:

```

1 #include <Servo.h>
2 #define servoPin 16
3
4 Servo myServo; // create servo object to control a servo
5 int pos = 0; // variable to store the servo position
6
7 void setup() {
8     myServo.attach(servoPin, 500, 2500); // attaches the servo on pin 9 to the
9         servo object
10 }
11
12 void loop() {
13     for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
14         // in steps of 1 degree
15         myServo.write(pos); // tell servo to go to position in
16 }
17 }
```

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



The following is the program code:

```

1 #include <Servo.h>
2 #define servoPin 16
3
4 Servo myServo; // create servo object to control a servo
5 int pos = 0; // variable to store the servo position
6
7 void setup() {
8     myServo.attach(servoPin); // attaches the servo on pin 9 to the servo object
9 }
10
11 void loop() {
12     for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
13         // in steps of 1 degree
14         myServo.write(pos); // tell servo to go to position in variable 'pos'
15 }
```

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

```
15     delay(15);           // waits 15 ms for the servo to reach the position
16 }
17 for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
18     myServo.write(pos);          // tell servo to go to position in variable 'pos'
19     delay(15);                 // waits 15 ms for the servo to reach the position
20 }
21 }
```

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

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

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

```
4 Servo myServo; // create servo object to control a servo
```

Set the control servo motor pin.

```
8 myServo.attach(servоСPin); // attaches the servo on pin 9 to the servo object
```

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

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

Reference

Servo Class

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

Servo myservo;

Most other boards can define 12 objects of Servo type, namely, they can control up to 12 servos.

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

myservo.attach(pin): Initialize the servo, the parameter is the port connected to servo signal line;

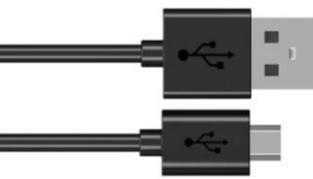
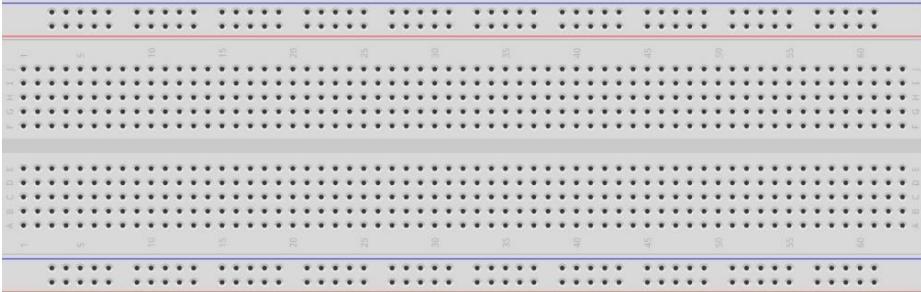
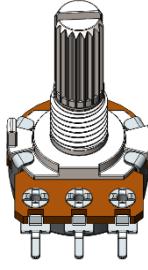
myservo.write(angle): Control servo to rotate to the specified angle; parameter here is to specify the angle.



Project 19.2 Servo Knob

Use a potentiometer to control the servomotor to rotate at any angle.

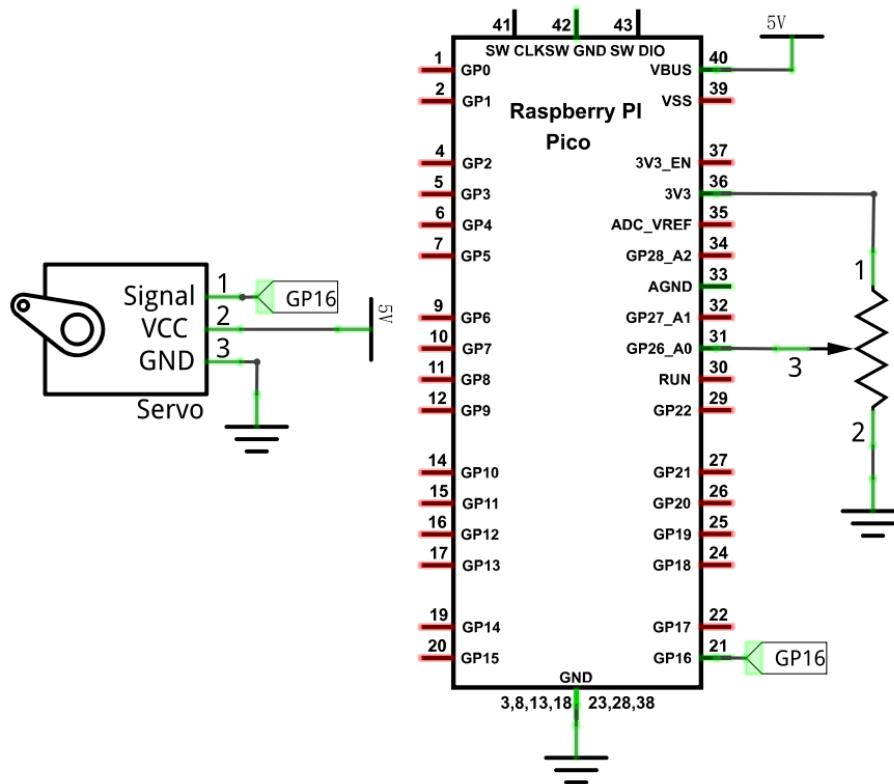
Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
Servo x1	Jumper	Rotary potentiometer x1
		

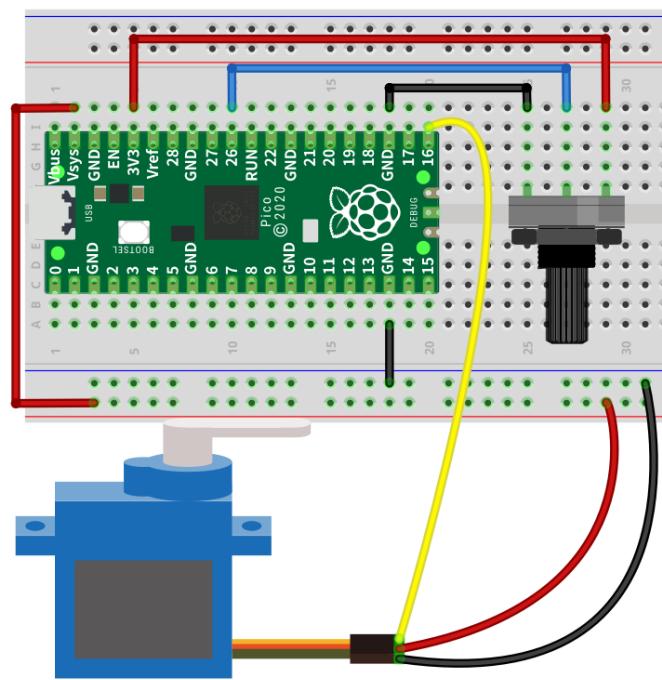
Circuit

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

Schematic diagram



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





Sketch

Sketch_19.2_Control_Servo_by_Potentiometer

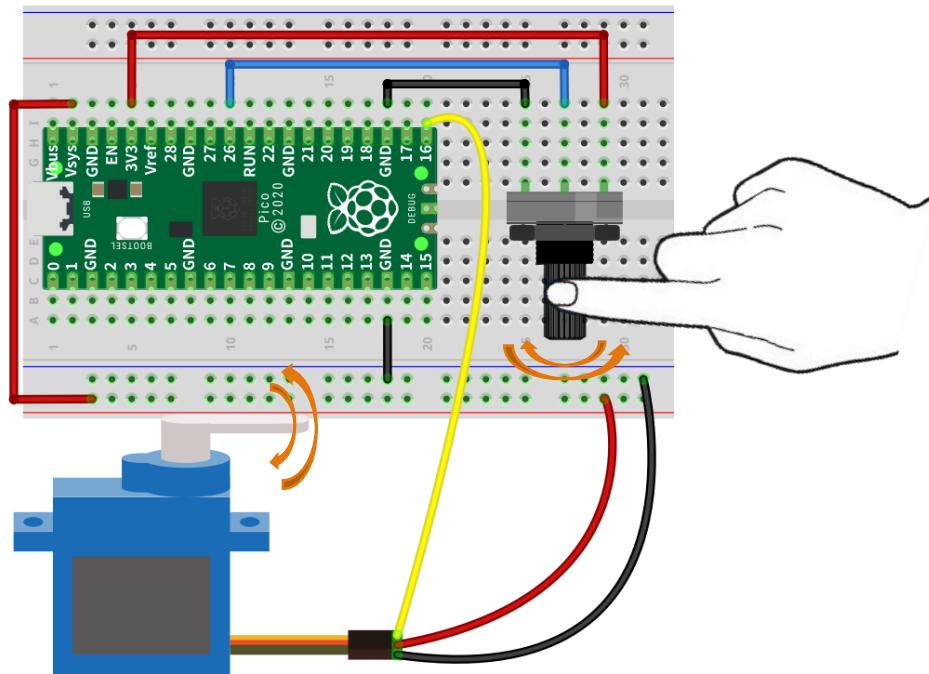
Now, write the code to detect the voltage of rotary potentiometer, and control servo to rotate to a different angle according to that.

```

1 #include <Servo.h>
2
3 #define servoPin 16          // define the pin of servo signal line
4 #define adcPin   26          // analog pin used to connect the potentiometer
5
6 Servo myservo;           // create servo object to control a servo
7 int potVal;              // variable to read the potValue from the analog pin
8
9 void setup() {
10     myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
11 }
12
13 void loop() {
14     potVal = analogRead(adcPin);      // reads the potValue of the potentiometer
15     potVal = map(potVal, 0, 1023, 0, 180); // scale it to use it with the servo
16     myservo.write(potVal);           // sets the servo position
17     delay(15);                   // waits for the servo to get there
18 }
```

In the code, we obtain the ADC value of GP26, and map it to the servo angle.

Verify and upload the code, turn the potentiometer shaft, and then the servo will rotate to a corresponding angle.

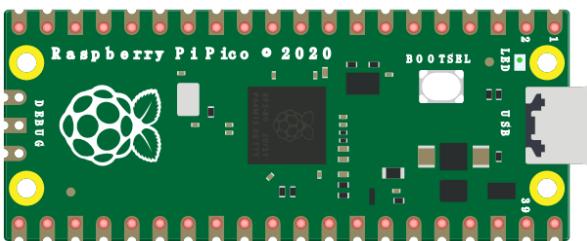
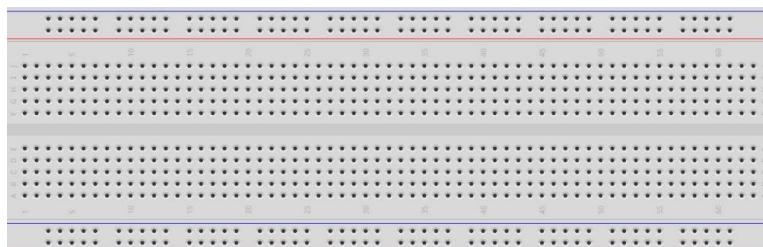
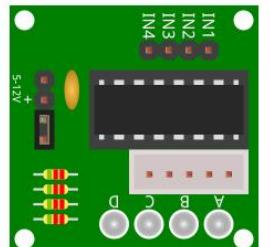
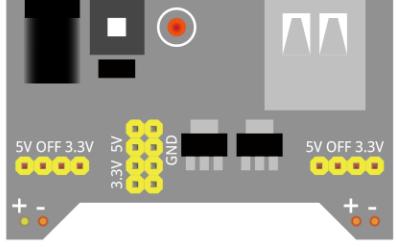


Chapter 20 Stepper Motor

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

Project 20.1 Stepper Motor

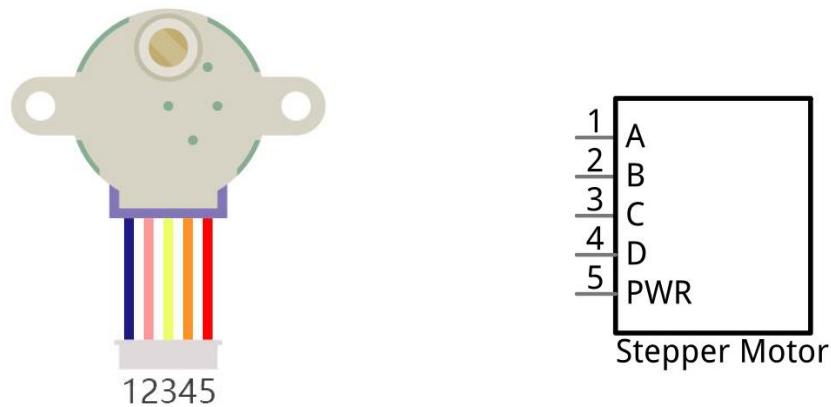
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Stepper Motor x1		ULN2003 Stepper motor Driver x1
		Jumper
Breadboard Power x1		9V battery (prepared by yourself) & battery line x1

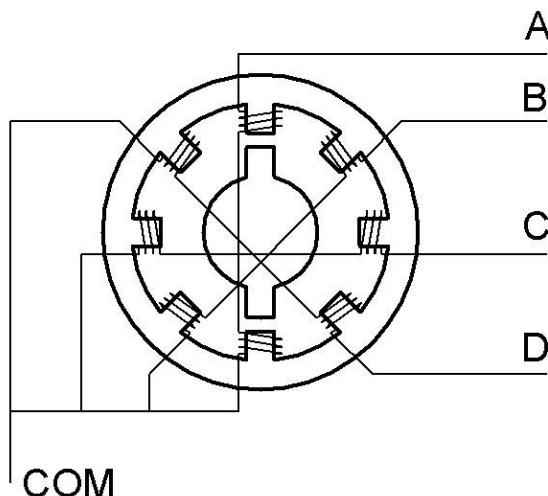
Component Knowledge

Stepper Motor x1

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

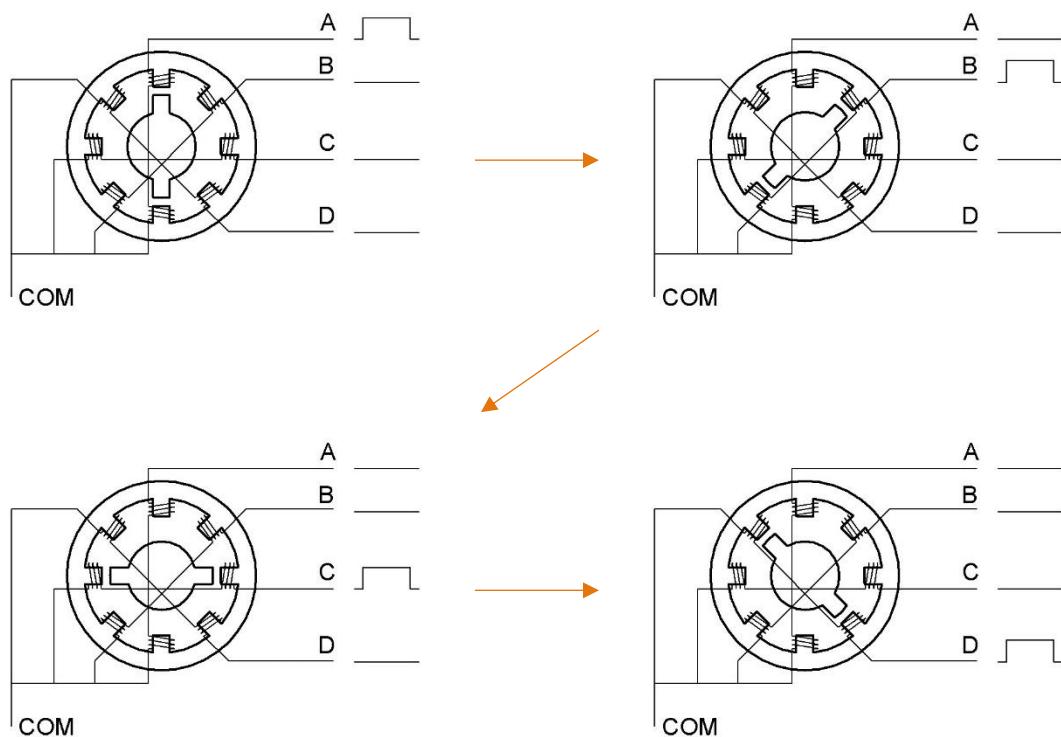


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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

A common driving process is as follows:



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

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

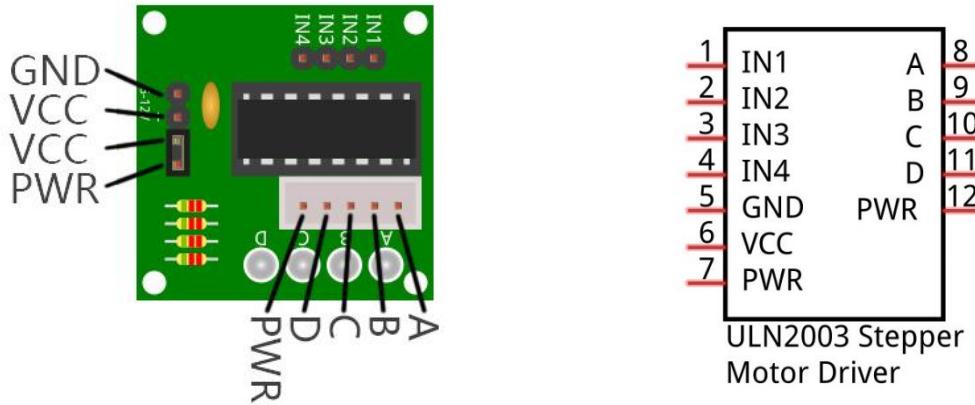
The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft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

The time required for each step of the stepper motor must be greater than 2ms to operate normally.



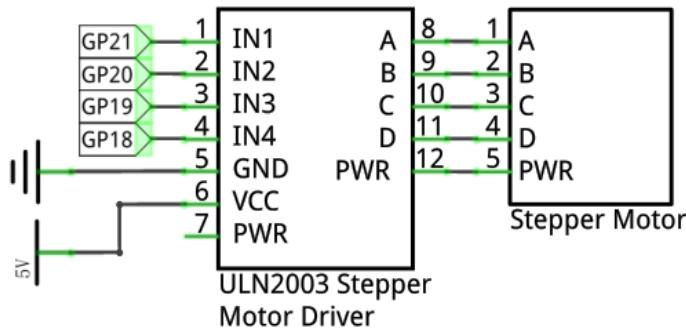
ULN2003 Stepper motor driver

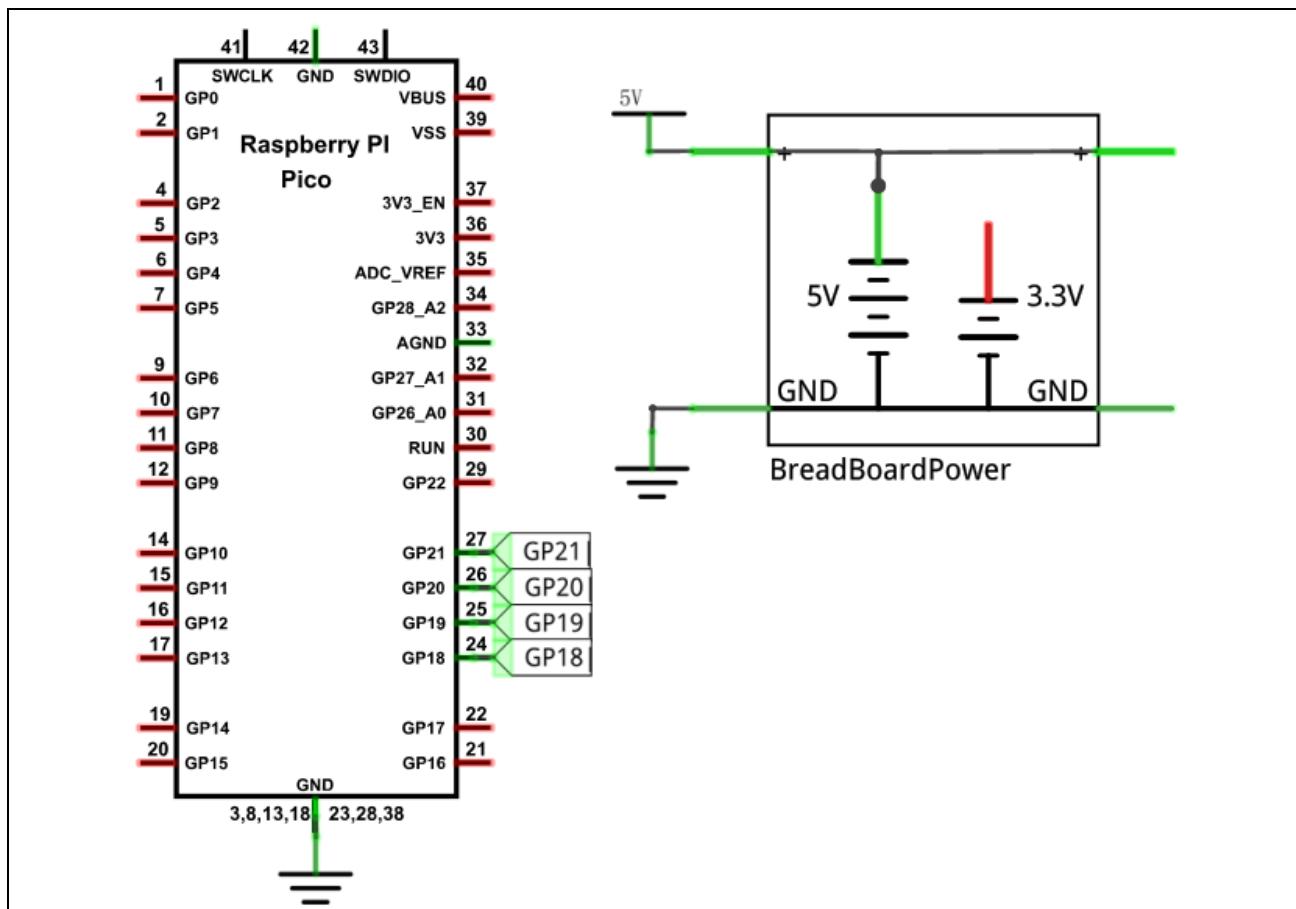
A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and four LEDs are integrated into the board to indicate the state of these signals.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



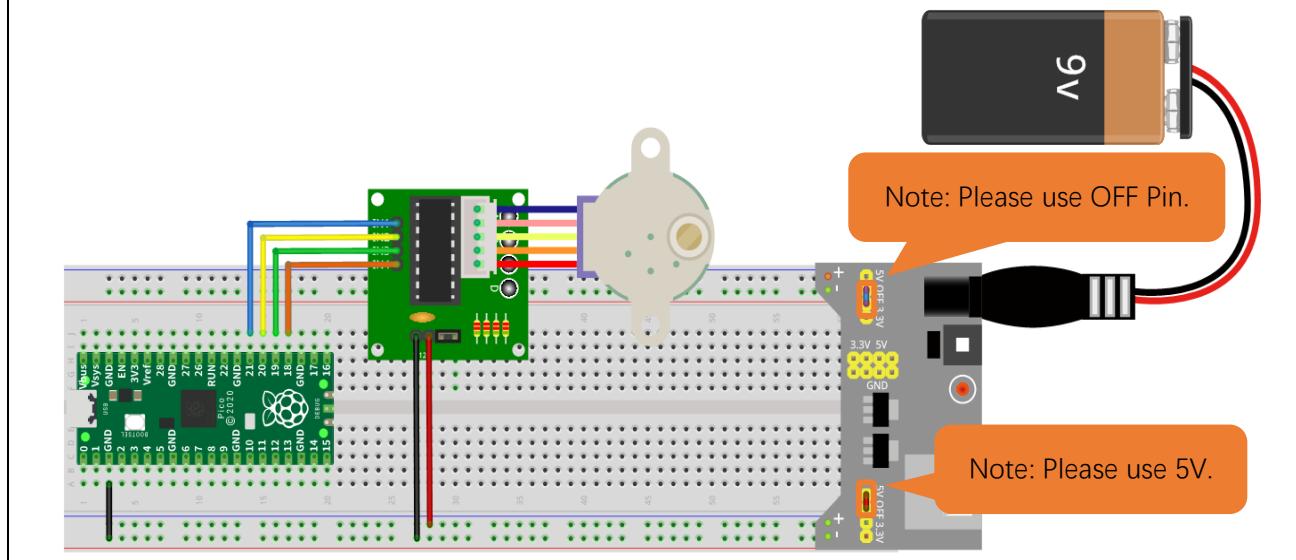
Circuit

Schematic diagram





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





Sketch

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

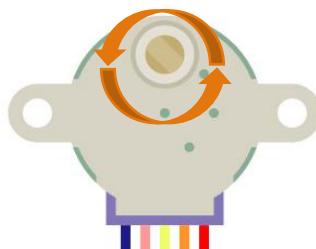
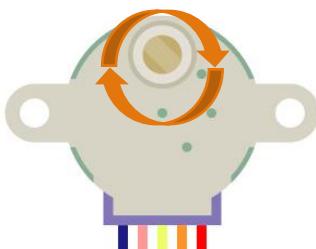
Sketch_20.1_Drive_Stepper_Motor

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

- Title Bar:** Sketch_20.1_Drive_Stepper_Motor | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Board Selection:** Raspberry Pi Pico
- Code Area:** Displays the following C++ code for a stepper motor driver:

```
8 int outPorts[] = {21, 20, 19, 18};
9
10 void setup() {
11     // set pins to output
12     for (int i = 0; i < 4; i++) {
13         pinMode(outPorts[i], OUTPUT);
14     }
15 }
16
17 void loop()
18 {
19     // Rotate a full turn
20     moveSteps(true, 32 * 64, 3);
21     delay(1000);
22     // Rotate a full turn towards another direction
```
- Output Area:** Shows a blacked-out log area with the message "Ln 1, Col 1 Raspberry Pi Pico on COM15 [not connected] 4" at the bottom right.

Compile and upload the code to the Pico, the stepper motor will rotate 360° clockwise and stop for 1s, and then rotate 360° anticlockwise and stop for 1s. This action is repeated in an endless loop.



The following is the program code:

```
1 // Connect the port of the stepper motor driver
2 int outPorts[] = {21, 20, 19, 18};
3
4 void setup() {
5     // set pins to output
6     for (int i = 0; i < 4; i++) {
7         pinMode(outPorts[i], OUTPUT);
8     }
9 }
10
11 void loop() {
12     // Rotate a full turn
13     moveSteps(true, 32 * 64, 3);
14     delay(1000);
15     // Rotate a full turn towards another direction
16     moveSteps(false, 32 * 64, 3);
17     delay(1000);
18 }
19
20 //Suggestion: the motor turns precisely when the ms range is between 3 and 20
21 void moveSteps(bool dir, int steps, byte ms) {
22     for (unsigned long i = 0; i < steps; i++) {
23         moveOneStep(dir); // Rotate a step
24         delay(constrain(ms, 3, 20)); // Control the speed
25     }
26 }
27
28 void moveOneStep(bool dir) {
29     // Define a variable, use four low bit to indicate the state of port
30     static byte out = 0x01;
31     // Decide the shift direction according to the rotation direction
32     if (dir) { // ring shift left
33         out != 0x08 ? out = out << 1 : out = 0x01;
34     }
35     else { // ring shift right
36         out != 0x01 ? out = out >> 1 : out = 0x08;
37     }
38     // Output singal to each port
39     for (int i = 0; i < 4; i++) {
40         digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
41     }
42 }
```

```

44 void moveAround(bool dir, int turns, byte ms) {
45     for(int i=0;i<turns;i++)
46         moveSteps(dir, 32*64, ms);
47 }
48 void moveAngle(bool dir, int angle, byte ms) {
49     moveSteps(dir, (angle*32*64/360), ms);
50 }
```

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

```

1 // Connect the port of the stepper motor driver
2 int outPorts[] = {21, 20, 19, 18};
```

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

```
23 moveOneStep(dir); // Rotate a step
```

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

```

29 // Define a variable, use four low bit to indicate the state of port
30 static byte out = 0x01;
31 // Decide the shift direction according to the rotation direction
32 if (dir) { // ring shift left
33     out != 0x08 ? out = out << 1 : out = 0x01;
34 }
35 else { // ring shift right
36     out != 0x01 ? out = out >> 1 : out = 0x08;
37 }
```

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

```

38 // Output singal to each port
39 for (int i = 0; i < 4; i++) {
40     digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
41 }
```

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

```

20 //Suggestion: the motor turns precisely when the ms range is between 3 and 20
21 void moveSteps(bool dir, int steps, byte ms) {
22     for (unsigned long i = 0; i < steps; i++) {
23         moveOneStep(dir); // Rotate a step
24         delay(constrain(ms, 3, 20)); // Control the speed
25     }
26 }
```

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

```
44 void moveAround(bool dir, int turns, byte ms) {  
45     for(int i=0;i<turns;i++)  
46         moveSteps(dir, 32*64, ms);  
47 }
```

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

```
48 void moveAngle(bool dir, int angle, byte ms) {  
49     moveSteps(dir, (angle*32*64/360), ms);  
50 }
```

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

```
11 void loop() {  
12     // Rotate a full turn  
13     moveSteps(true, 32 * 64, 3);  
14     delay(1000);  
15     // Rotate a full turn towards another direction  
16     moveSteps(false, 32 * 64, 3);  
17     delay(1000);  
18 }
```

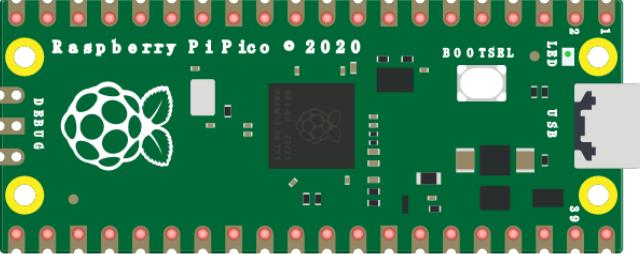
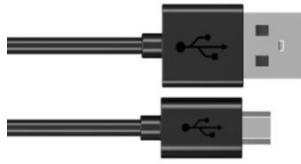
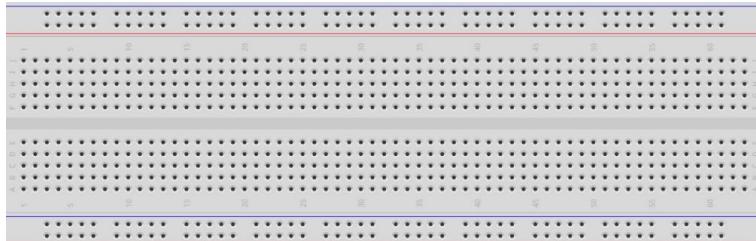
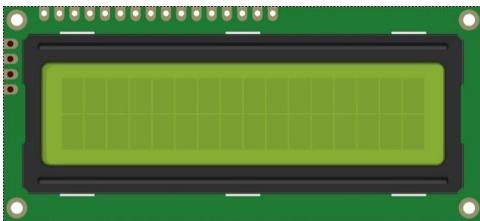
Chapter 21 LCD1602

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

Project 21.1 LCD1602

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

Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
LCD1602 Module x1		Jumper	

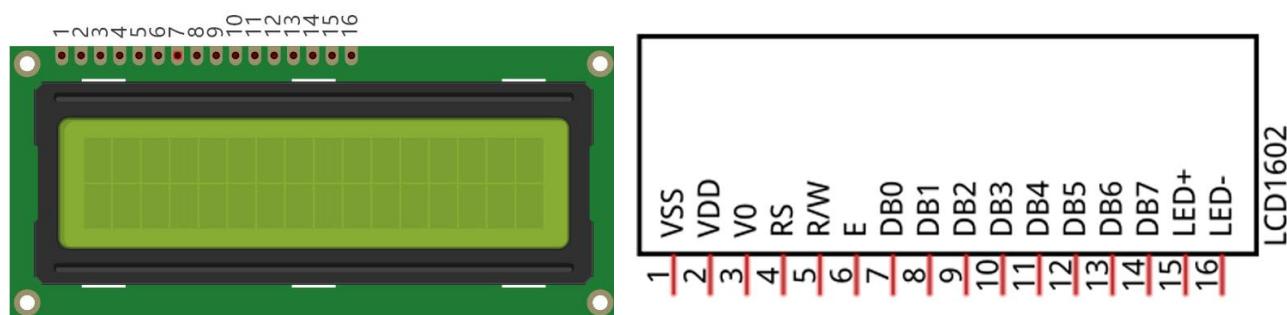
Component Knowledge

I2C communication

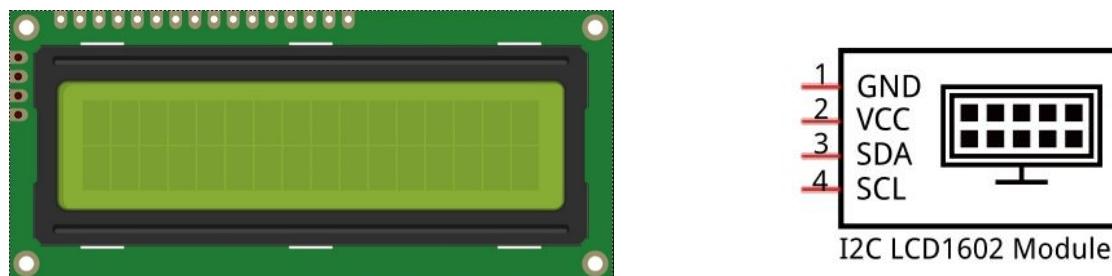
I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used for the connection of micro controllers and their peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device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

LCD1602 communication

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

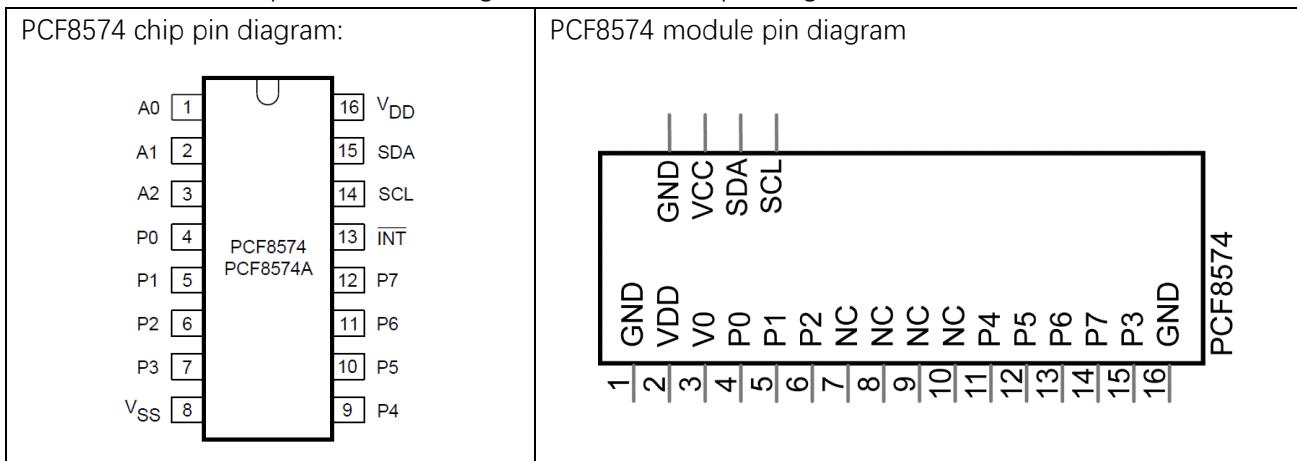


I2C LCD1602 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to use only four lines to operate the LCD1602.

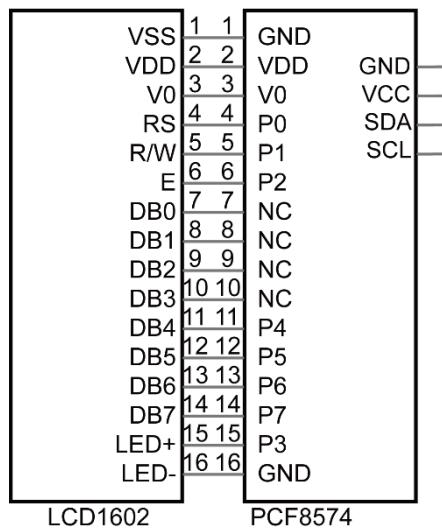


The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

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



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

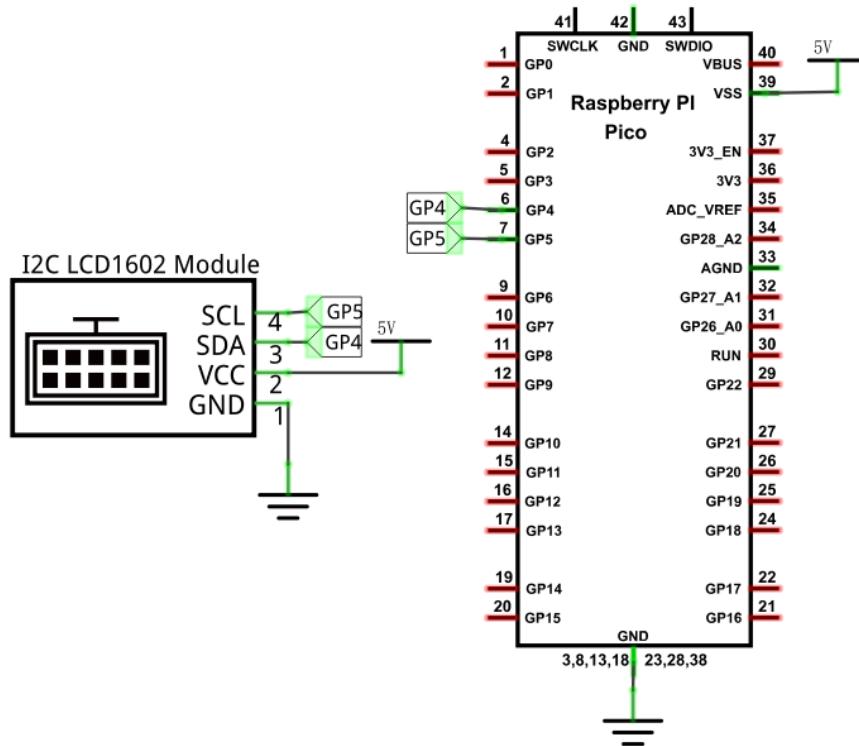


Therefore, we only need four pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface.

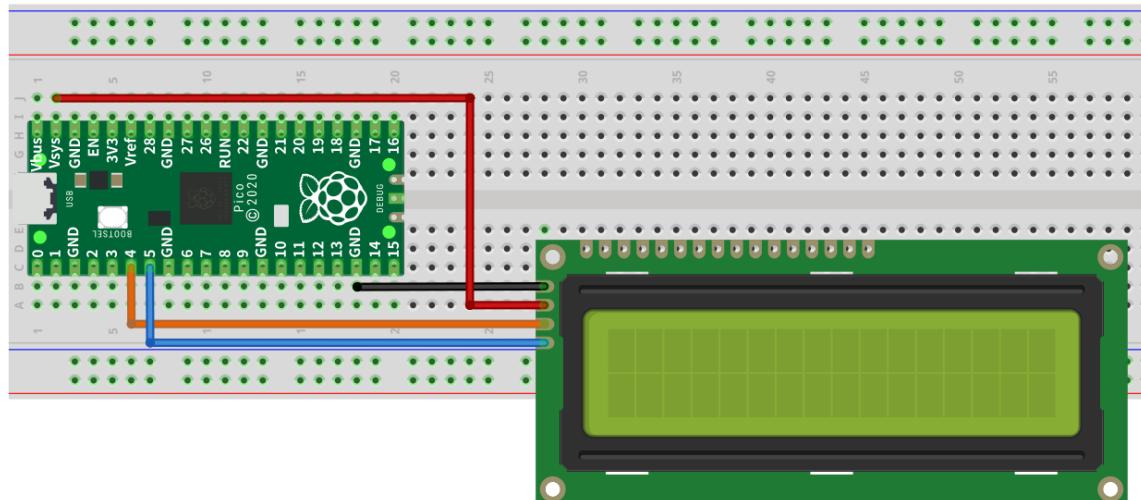
In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



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

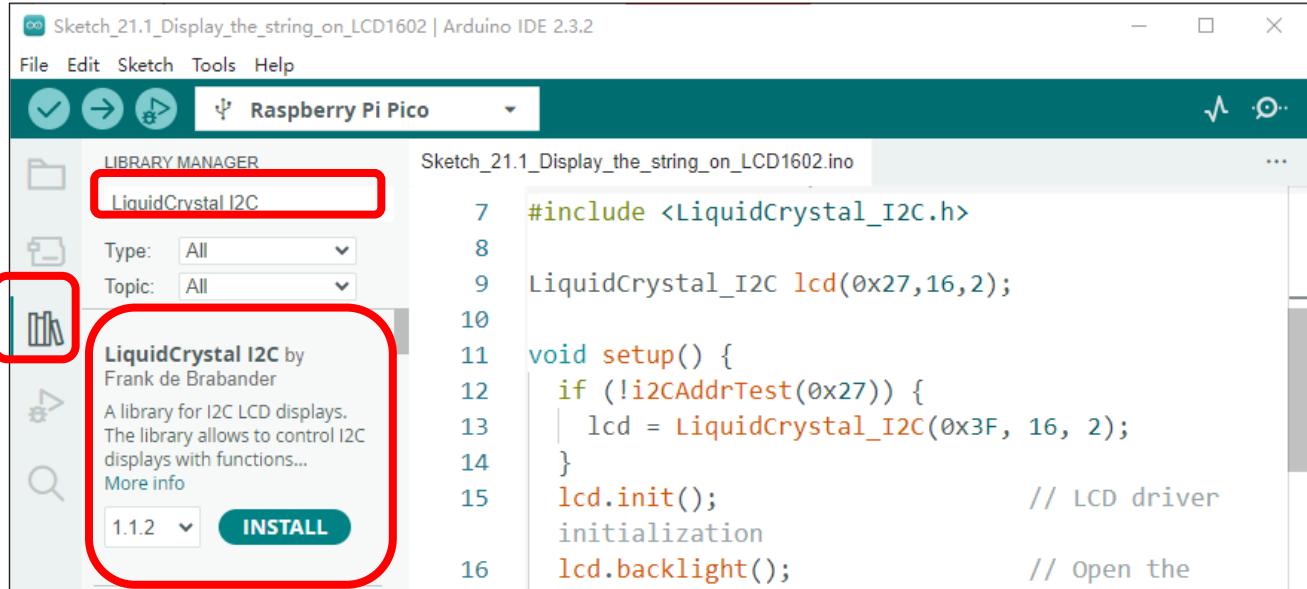


Sketch

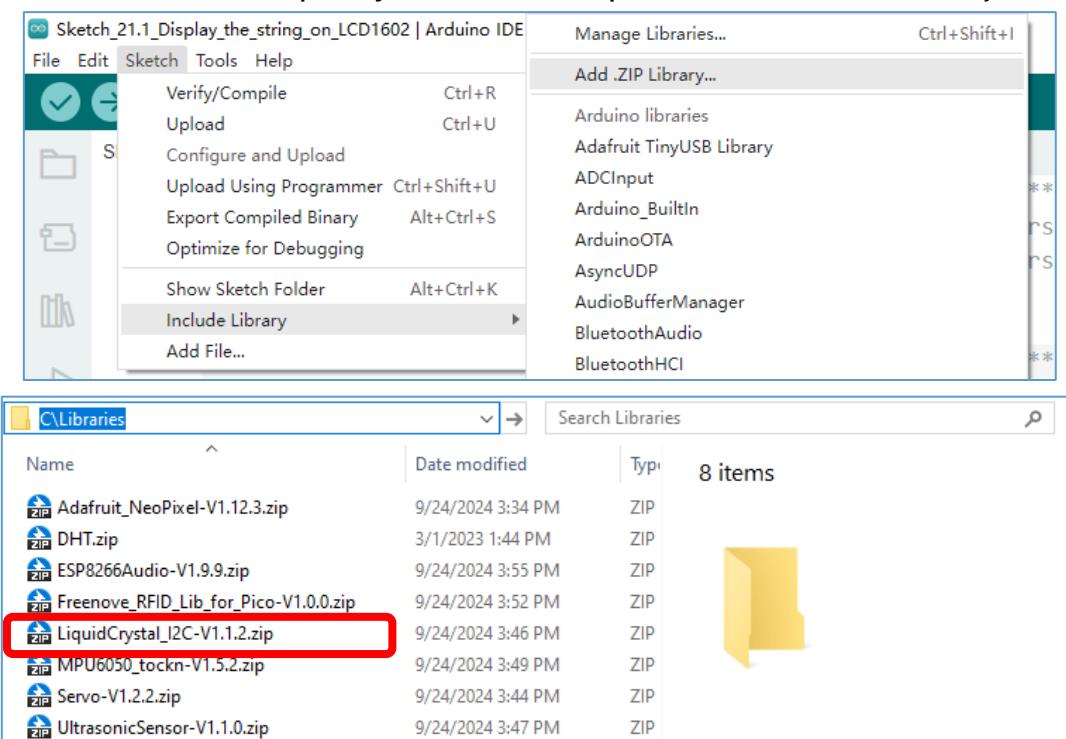
How to install the library

In this project, we use a third-party library named **LiquidCrystal I2C**. Here are two ways to install the library for your reference.

The first one: Open Arduino IDE; click **Library Manager** on the left, search "LiquidCrystal I2C" to install.



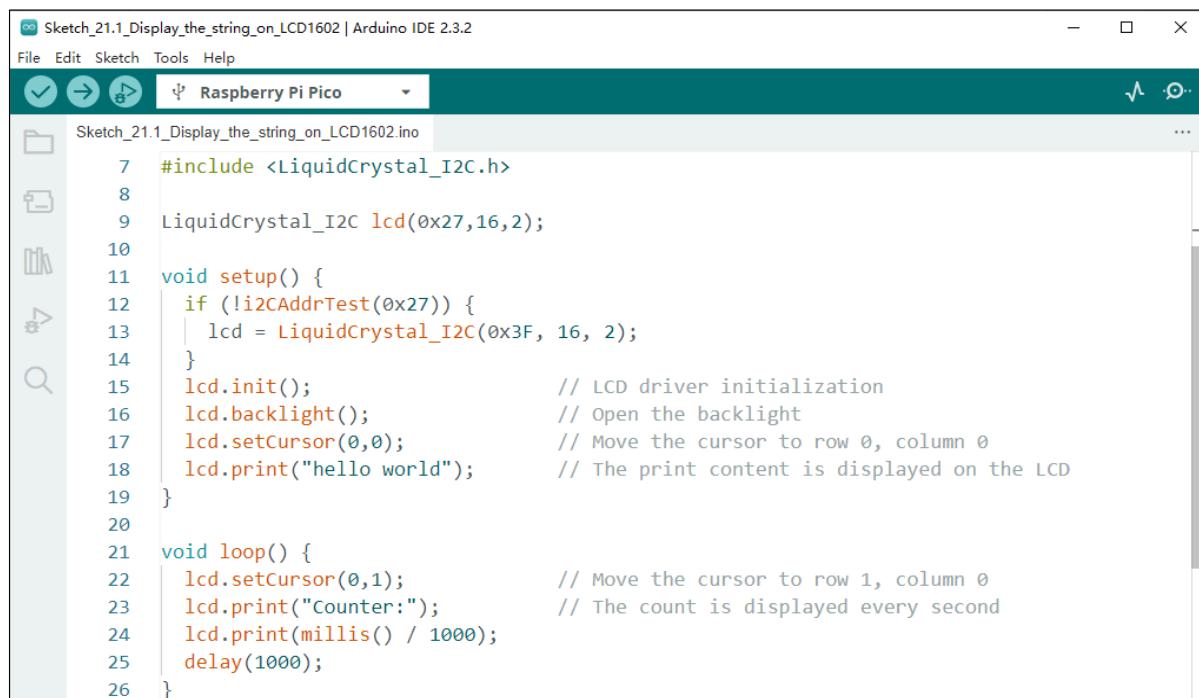
The second way, open Arduino IDE, click Sketch → Include Library → Add .ZIP Library. In the pop-up window, find the file named "../Libraries/ **LiquidCrystal_I2C-V1.1.2.Zip**" which locates in this directory, and click OPEN.



Use I2C LCD 1602 to display characters and variables.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Sketch_21.1_Display_the_string_on_LCD1602



```

Sketch_21.1_Display_the_string_on_LCD1602 | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_21.1_Display_the_string_on_LCD1602.ino
7 #include <LiquidCrystal_I2C.h>
8
9 LiquidCrystal_I2C lcd(0x27, 16, 2);
10
11 void setup() {
12     if (!i2cAddrTest(0x27)) {
13         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
14     }
15     lcd.init(); // LCD driver initialization
16     lcd.backlight(); // Open the backlight
17     lcd.setCursor(0,0); // Move the cursor to row 0, column 0
18     lcd.print("Hello world!"); // The print content is displayed on the LCD
19 }
20
21 void loop() {
22     lcd.setCursor(0,1); // Move the cursor to row 1, column 0
23     lcd.print("Counter:");
24     lcd.print(millis() / 1000);
25     delay(1000);
26 }

```

Compile and upload the code to Pico and the LCD1602 displays characters.



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

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



If the LCD1602 you received is the following one, and you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display clearly.



Any concerns? ✉ support@freenove.com



The following is the program code:

```

1 #include <LiquidCrystal_I2C.h>
2
3 LiquidCrystal_I2C lcd(0x27, 16, 2);
4
5 void setup() {
6     if (!i2CAddrTest(0x27)) {
7         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
8     }
9     lcd.init(); // LCD driver initialization
10    lcd.backlight(); // Open the backlight
11    lcd.setCursor(0,0); // Move the cursor to row 0, column 0
12    lcd.print("hello world"); // The print content is displayed on the LCD
13 }
14
15 void loop() {
16     lcd.setCursor(0,1); // Move the cursor to row 1, column 0
17     lcd.print("Counter:");
18     lcd.print(millis() / 1000);
19     delay(1000);
20 }
21
22 bool i2CAddrTest(uint8_t addr) {
23     Wire.begin();
24     Wire.beginTransmission(addr);
25     if (Wire.endTransmission() == 0) {
26         return true;
27     }
28     return false;
29 }
```

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

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

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

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

Initialize LCD1602 and turn on the backlight of LCD.

```

6     if (!i2CAddrTest(0x27)) {
7         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
8     }
9     lcd.init(); // LCD driver initialization
10    lcd.backlight(); // Turn on the backlight
```

Check whether the I2C address is responded by a device.

```

22 bool i2CAddrTest(uint8_t addr) {
23     Wire.begin();
```

Any concerns? ✉ support@freenove.com

```

24   Wire.beginTransmission(addr);
25   if (Wire.endTransmission() == 0) {
26     return true;
27   }
28   return false;
29 }
```

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

```

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

Print the number on the second line of LCD1602.

```

16 void loop() {
17   lcd.setCursor(0,1);           // Move the cursor to row 1, column 0
18   lcd.print("Counter:");       // The count is displayed every second
19   lcd.print(millis() / 1000);
20   delay(1000);
21 }
```

Reference

class LiquidCrystal

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

LiquidCrystal_I2C lcd(0x27, 16, 2);

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

init();

Initializes the Lcd1602's device

backlight();

Turn on Lcd1602's backlight.

setCursor(column, row);

Sets the screen's column and row.

Column: The range is 0 to 15.

Row: The range is 0 to 1.

print(String);

Print the character string on Lcd1602



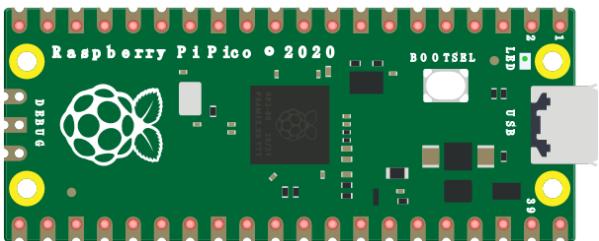
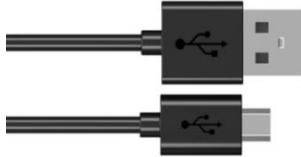
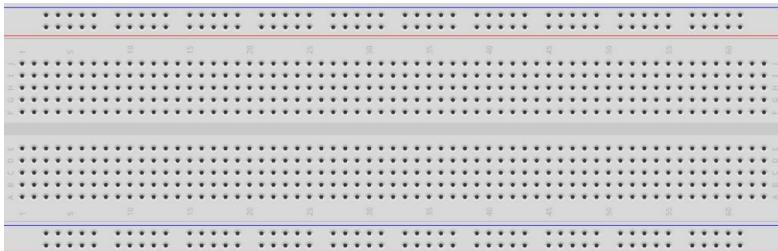
Chapter 22 Ultrasonic Ranging

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

Project 22.1 Ultrasonic Ranging

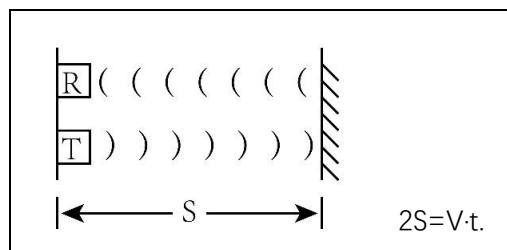
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

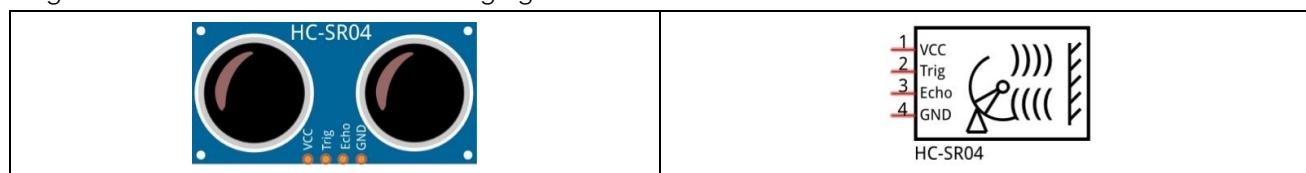
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper		HC SR04 x1	

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about $v=340\text{m/s}$, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: $s=vt/2$.



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



Pin description:

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

Technical specs:

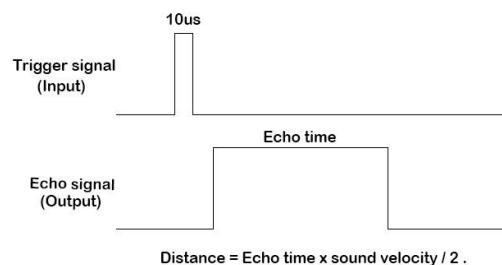
Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

Maximum measured distance: 200cm

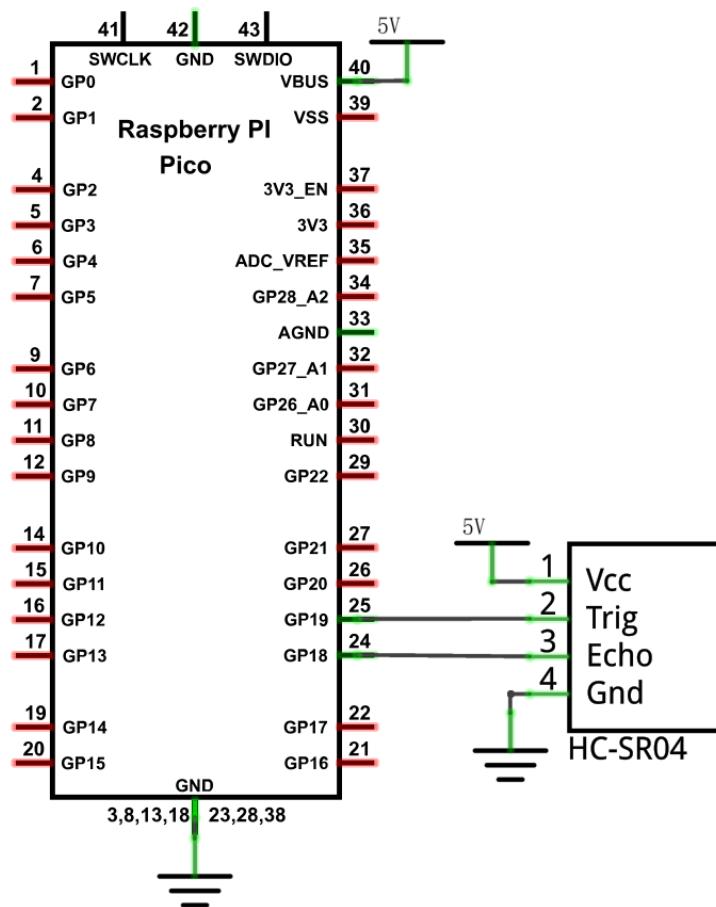
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The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



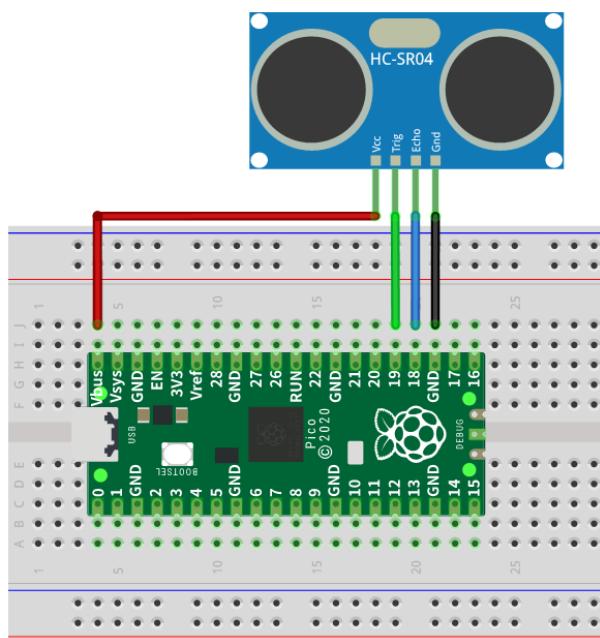
Circuit

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

Schematic diagram



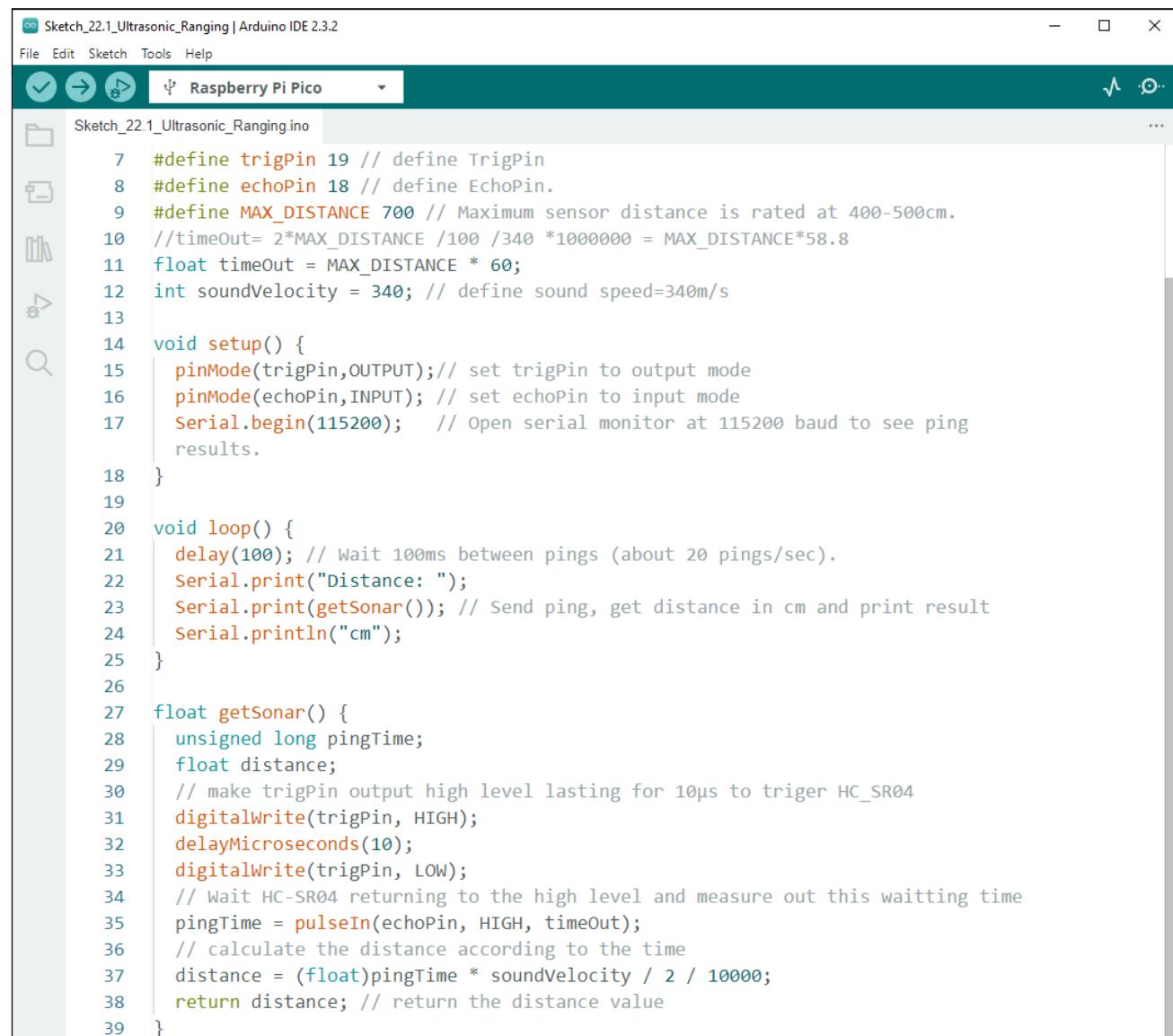
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

Sketch

Sketch_22.1_Ultrasonic_Ranging



The screenshot shows the Arduino IDE interface with the sketch file "Sketch_22.1_Ultrasonic_Ranging.ino" open. The code is written for a Raspberry Pi Pico. It defines pins for trigPin (19) and echoPin (18), sets MAX_DISTANCE to 700 cm, and initializes serial communication at 115200 baud. The setup() function configures the pins and starts the serial monitor. The loop() function sends a ping every 100ms, receives the echo time, calculates the distance, and prints it to the serial monitor in cm. The getSonar() function handles the pulse generation and measurement.

```
#define trigPin 19 // define TrigPin
#define echoPin 18 // define EchoPin.
#define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s
void setup() {
    pinMode(trigPin,OUTPUT);// set trigPin to output mode
    pinMode(echoPin,INPUT); // set echoPin to input mode
    Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.
}
void loop() {
    delay(100); // Wait 100ms between pings (about 20 pings/sec).
    Serial.print("Distance: ");
    Serial.print(getSonar()); // Send ping, get distance in cm and print result
    Serial.println("cm");
}
float getSonar() {
    unsigned long pingTime;
    float distance;
    // make trigPin output high level lasting for 10µs to trigger HC-SR04
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Wait HC-SR04 returning to the high level and measure out this waiting time
    pingTime = pulseIn(echoPin, HIGH, timeOut);
    // calculate the distance according to the time
    distance = (float)pingTime * soundVelocity / 2 / 10000;
    return distance; // return the distance value
}
```

Download the code to Pico, open the serial monitor, set the baud rate to 115200, and you can use it to measure the distance between the ultrasonic module and the object, as shown in the following picture:



The following is the program code:

```

1 #define trigPin 19 // define trigPin
2 #define echoPin 18 // define echoPin.
3 #define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400–500cm.
4 //timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
5 float timeOut = MAX_DISTANCE * 60;
6 int soundVelocity = 340; // define sound speed=340m/s
7
8 void setup() {
9     pinMode(trigPin,OUTPUT);// set trigPin to output mode
10    pinMode(echoPin,INPUT); // set echoPin to input mode
11    Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.
12 }
13
14 void loop() {
15     delay(100); // Wait 100ms between pings (about 20 pings/sec).
16     Serial.printf("Distance: ");
17     Serial.print(getSonar()); // Send ping, get distance in cm and print result
18     Serial.println(" cm");
19 }
20
21 float getSonar() {
22     unsigned long pingTime;
23     float distance;
24     // make trigPin output high level lasting for 10us to trigger HC_SR04
25     digitalWrite(trigPin, HIGH);
26     delayMicroseconds(10);
27     digitalWrite(trigPin, LOW);
28     // Wait HC-SR04 returning to the high level and measure out this waiting time
29     pingTime = pulseIn(echoPin, HIGH, timeOut);
30     // calculate the distance according to the time
31     distance = (float)pingTime * soundVelocity / 2 / 10000;

```

```

32     return distance; // return the distance value
33 }
```

First, define the pins and the maximum measurement distance.

```

1 #define trigPin 19 // define trigPin
2 #define echoPin 18 // define echoPin.
3 #define MAX_DISTANCE 700      //define the maximum measured distance
```

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

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

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

```

21 float getSonar() {
22     unsigned long pingTime;
23     float distance;
24     // make trigPin output high level lasting for 10μs to trigger HC_SR04?
25     digitalWrite(trigPin, HIGH);
26     delayMicroseconds(10);
27     digitalWrite(trigPin, LOW);
28     // Wait HC-SR04 returning to the high level and measure out this waiting time
29     pingTime = pulseIn(echoPin, HIGH, timeOut);
30     // calculate the distance according to the time
31     distance = (float)pingTime * soundVelocity / 2 / 10000;
32     return distance; // return the distance value
33 }
```

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

```

14 void loop() {
15     delay(100); // Wait 100ms between pings (about 20 pings/sec).
16     Serial.printf("Distance: ");
17     Serial.print(getSonar()); // Send ping, get distance in cm and print result
18     Serial.println("cm");
19 }
```

About function **pulseIn()**:

int pulseIn(int pin, int level, int timeout);

pin: the number of the Arduino pin on which you want to read the pulse. Allowed data types: int.

value: type of pulse to read: either HIGH or LOW. Allowed data types: int.

timeout (optional): the number of microseconds to wait for the pulse to start; default is one second.

Project 22.2 Ultrasonic Ranging

Component List and Circuit

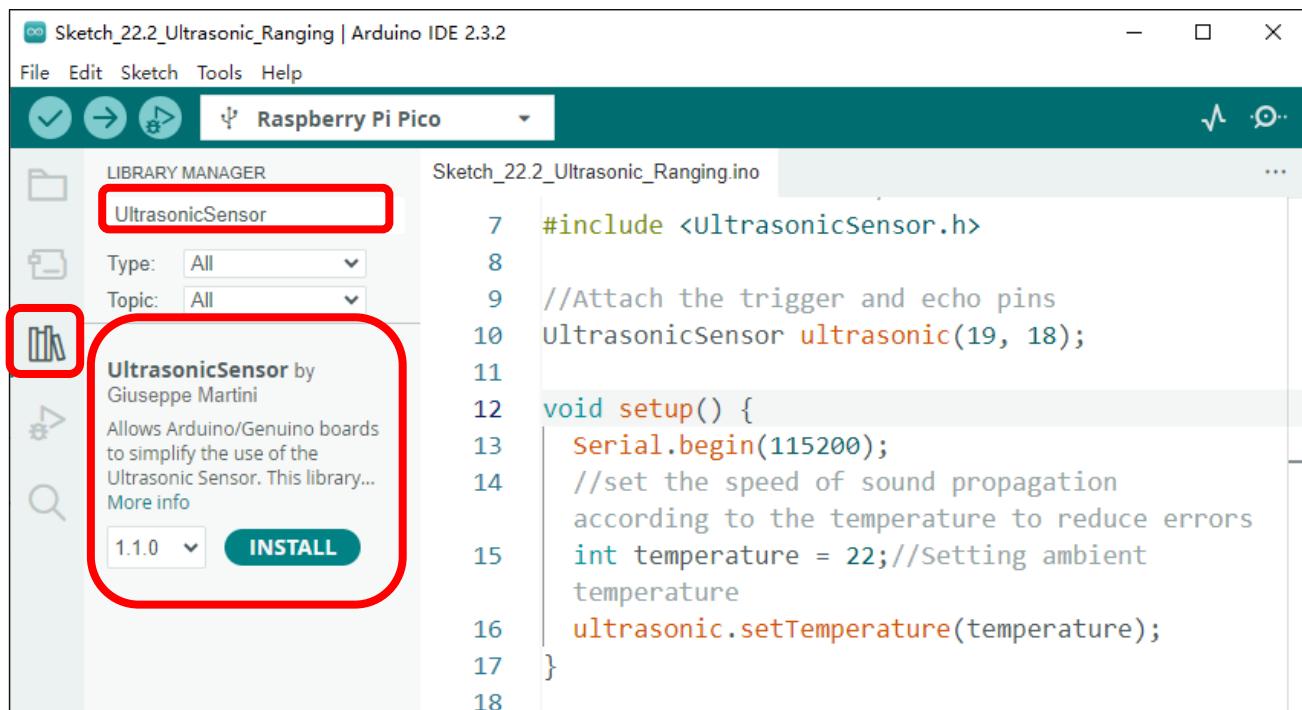
Component List and Circuit are the same as the previous section.

Sketch

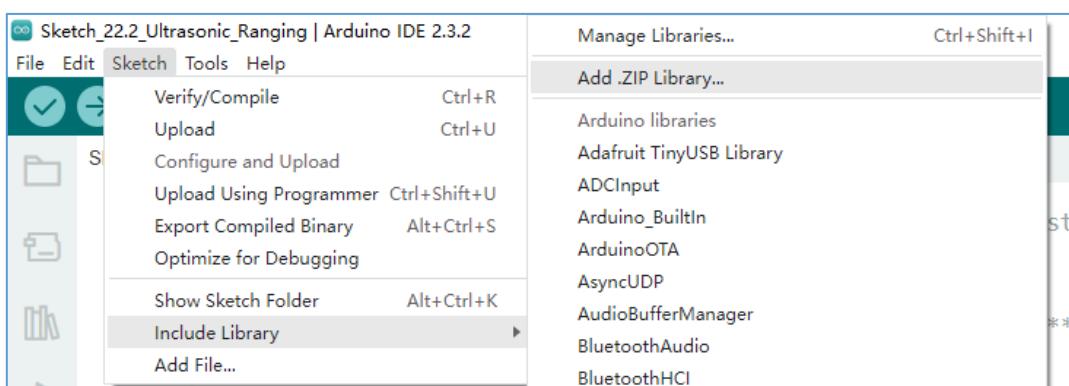
How to install the library

In this project, we use a third-party library named **UltrasonicSensor**. Here are two ways to install the library for your reference.

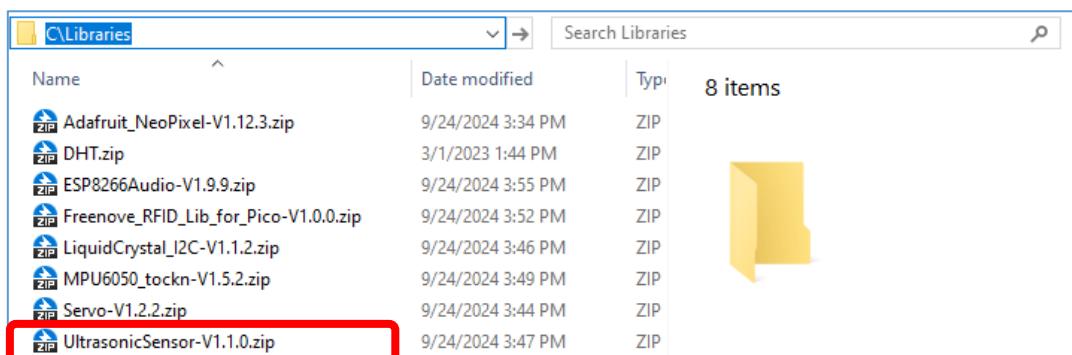
The first one: Open Arduino IDE, click **Library Manage** on the left, and search "**UltrasonicSensor**" to install.



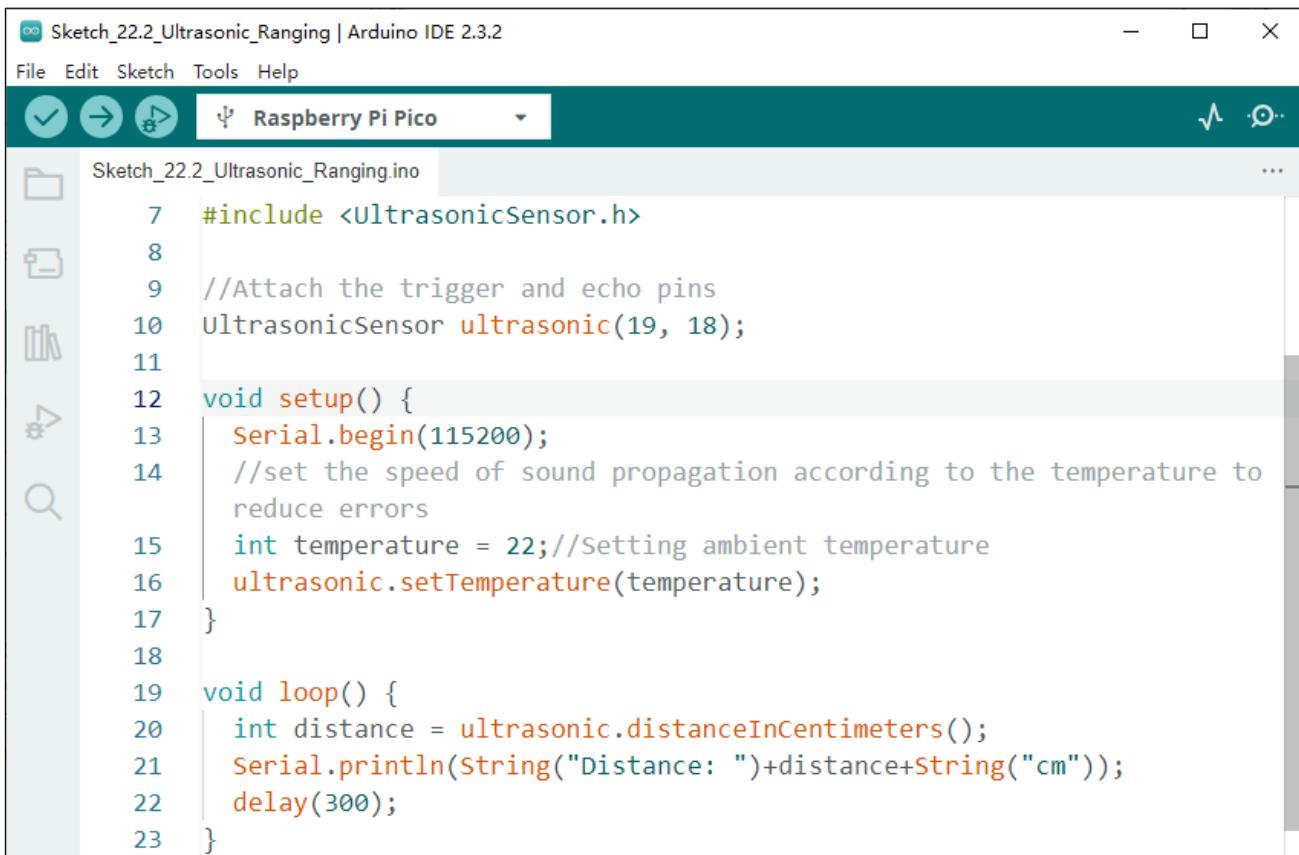
The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named “./Libraries/ **UltrasonicSensor-V1.1.0.Zip**” which locates in this directory, and click OPEN.



Any concerns?  support@freenove.com



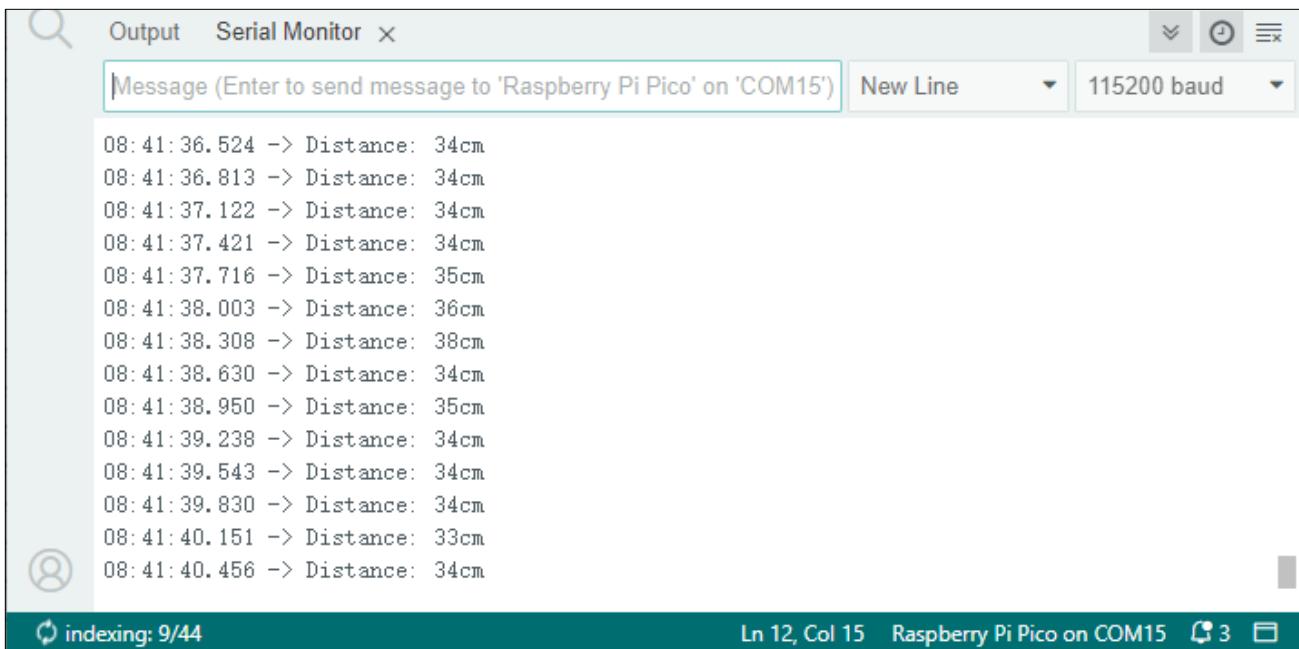
Sketch_22.2_Ultrasonic_Ranging



```

Sketch_22.2_Ultrasonic_Ranging | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_22.2_Ultrasonic_Ranging.ino
1 #include <UltrasonicSensor.h>
2
3 //Attach the trigger and echo pins
4 ultrasonicSensor ultrasonic(19, 18);
5
6 void setup() {
7     Serial.begin(115200);
8     //set the speed of sound propagation according to the temperature to
9     //reduce errors
10    int temperature = 22;//Setting ambient temperature
11    ultrasonic.setTemperature(temperature);
12 }
13
14 void loop() {
15     int distance = ultrasonic.distanceInCentimeters();
16     Serial.println(String("Distance: ")+distance+String("cm"));
17     delay(300);
18 }
19
20
21
22
23 }
```

Upload the sketch to Pico, open the serial monitor and set the baud rate to 115200. Use the ultrasonic module to measure distance, as shown in the following picture:



Time	Distance (cm)
08:41:36.524	34cm
08:41:36.813	34cm
08:41:37.122	34cm
08:41:37.421	34cm
08:41:37.716	35cm
08:41:38.003	36cm
08:41:38.308	38cm
08:41:38.630	34cm
08:41:38.950	35cm
08:41:39.238	34cm
08:41:39.543	34cm
08:41:39.830	34cm
08:41:40.151	33cm
08:41:40.456	34cm

The following is the program code:

```

1 #include <UltrasonicSensor.h>
2
3 //Attach the trigger and echo pins
4 UltrasonicSensor ultrasonic(19, 18);
5
6 void setup() {
7   Serial.begin(115200);
8   //set the speed of sound propagation according to the temperature to reduce errors
9   int temperature = 22; //Setting ambient temperature
10  ultrasonic.setTemperature(temperature);
11 }
12
13 void loop() {
14   int distance = ultrasonic.distanceInCentimeters();
15   Serial.print(String("Distance: ")+distance+String("cm\n"));
16   delay(300);
17 }
```

First, add UltrasonicSensor library.

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

Define an ultrasonic object and associate it with the pins.

```
4 UltrasonicSensor ultrasonic(19, 18);
```

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

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

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

```

13 void loop() {
14   int distance = ultrasonic.distanceInCentimeters();
15   Serial.print(String("Distance: ")+distance+String("cm\n"));
16   delay(300);
17 }
```

Reference

class UltrasonicSensor

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

UltrasonicSensor ultrasonic(19, 18);

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

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

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

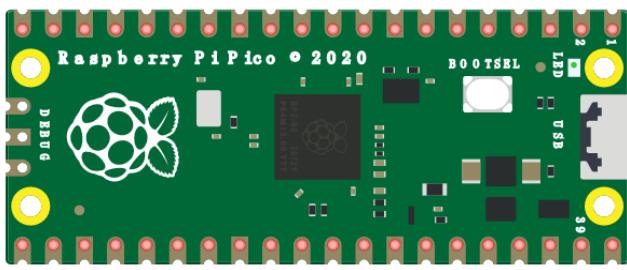
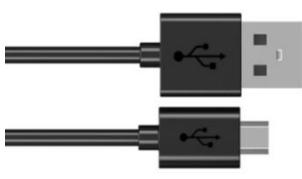
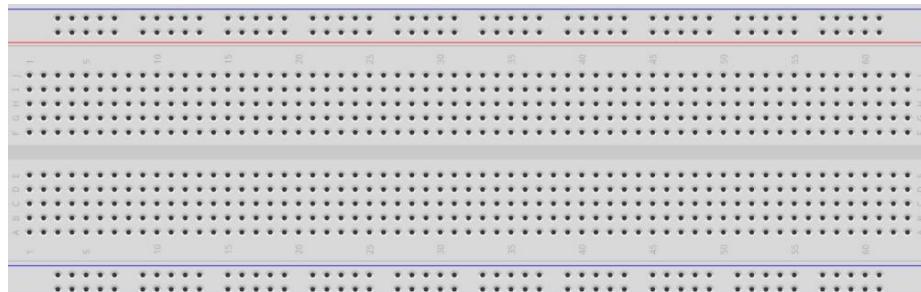
Chapter 23 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 23.1 Matrix Keypad

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

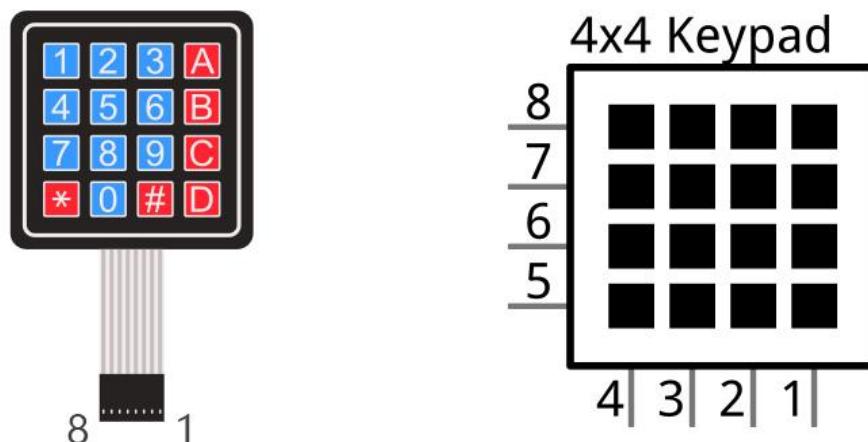
Component List

Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
Jumper	4x4 Matrix Keypad x1 

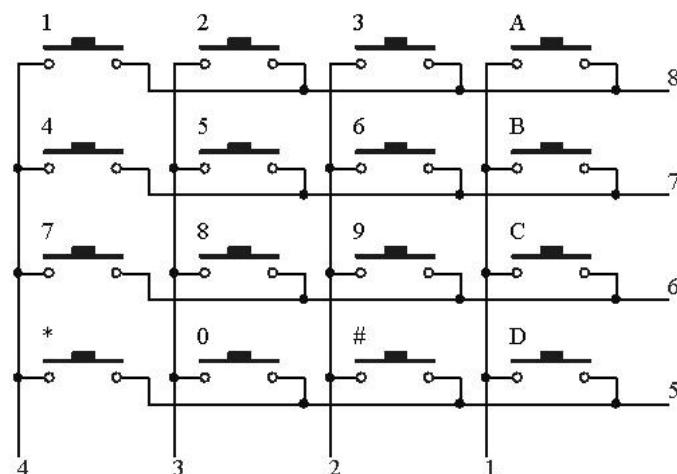
Component Knowledge

4x4 Matrix Keypad

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



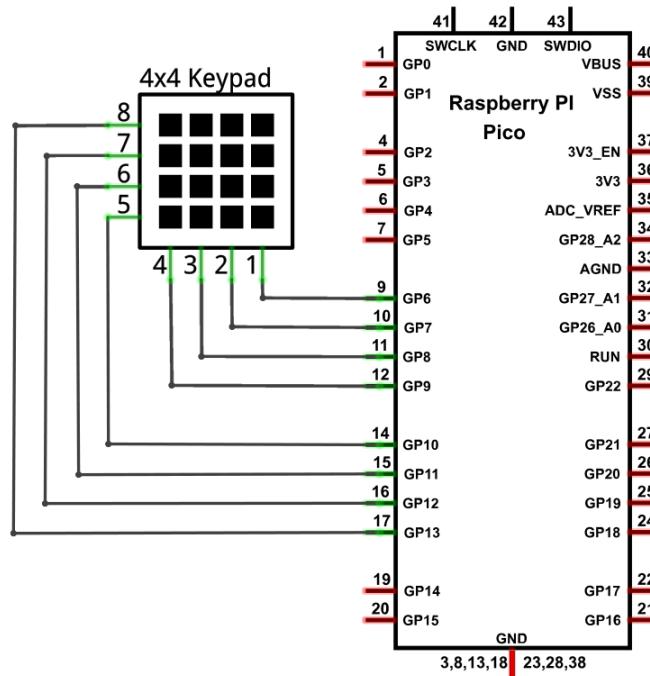
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



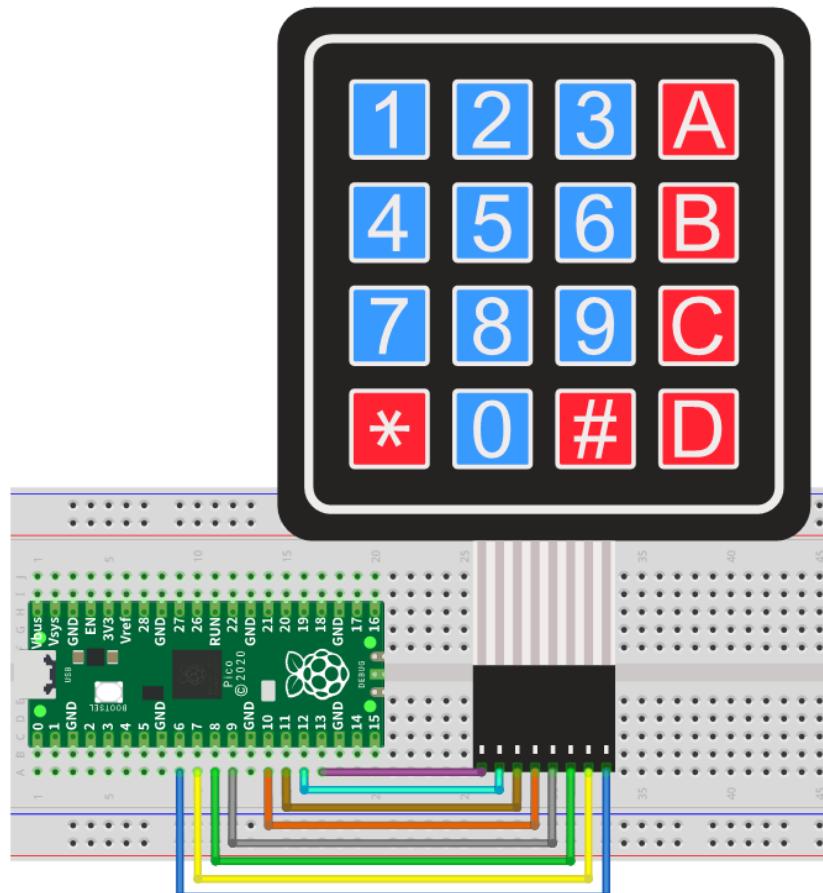
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to determine whether the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

Circuit

Schematic diagram



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

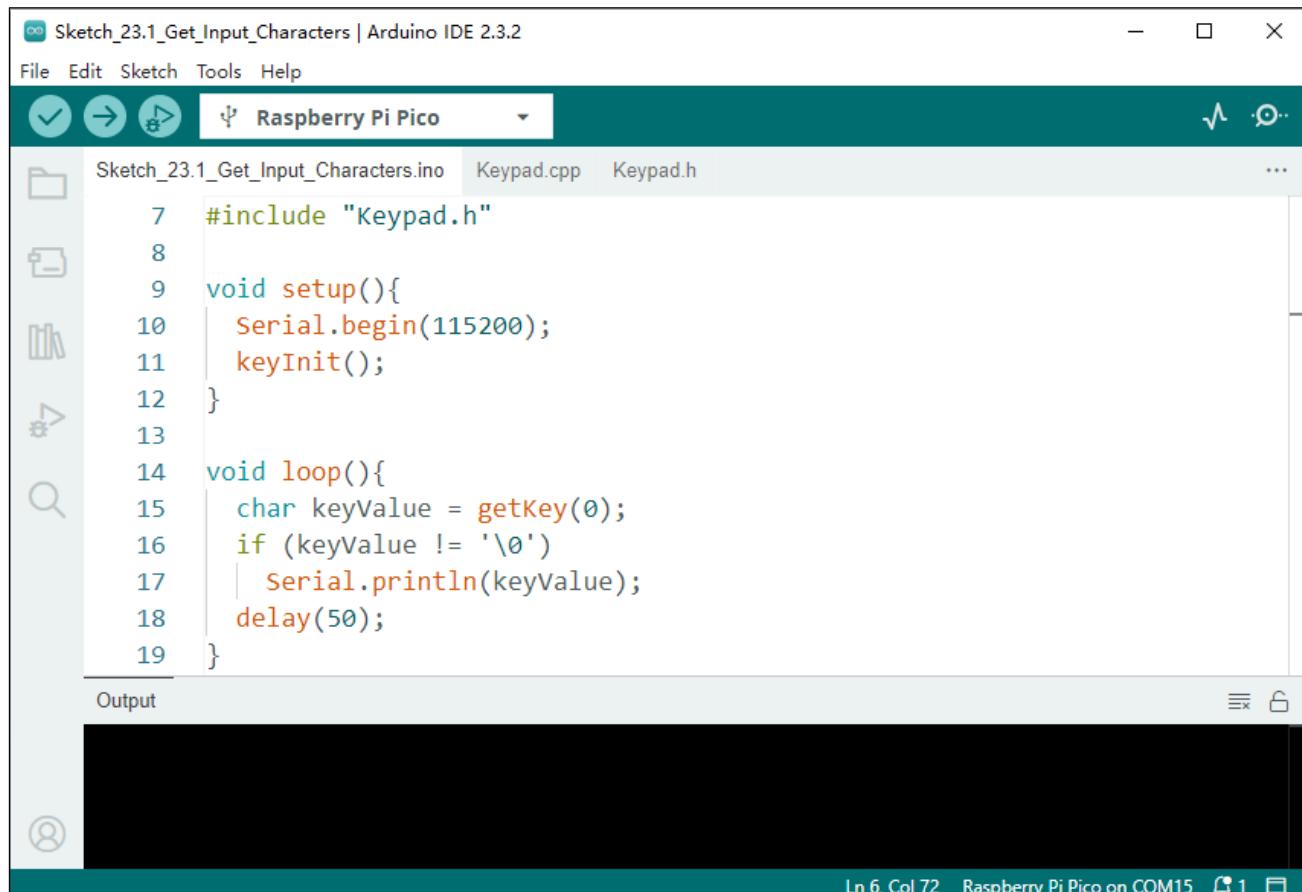


Any concerns? support@freenove.com

Sketch

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

Sketch_23.1_Get_Input_Characters

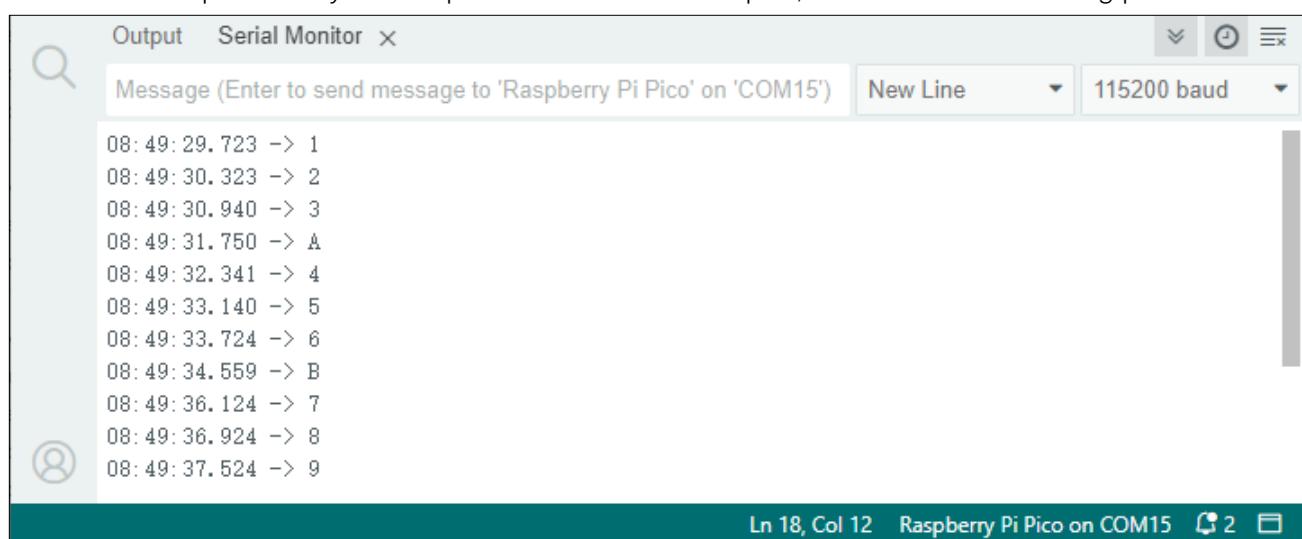


```

Sketch_23.1_Get_Input_Characters | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_23.1_Get_Input_Characters.ino Keypad.cpp Keypad.h
7 #include "Keypad.h"
8
9 void setup(){
10   Serial.begin(115200);
11   keyInit();
12 }
13
14 void loop(){
15   char keyValue = getKey(0);
16   if (keyValue != '\0')
17     Serial.println(keyValue);
18   delay(50);
19 }

```

Download the code to Pico, open the serial port monitor, set the baud rate to 115200, press the keyboard, the value of the pressed keys will be printed out via the serial port, as shown in the following picture:



Time	Key Pressed	Value
08:49:29.723		1
08:49:30.323		2
08:49:30.940		3
08:49:31.750		A
08:49:32.341		4
08:49:33.140		5
08:49:33.724		6
08:49:34.559		B
08:49:36.124		7
08:49:36.924		8
08:49:37.524		9

Keypad.cpp

```
1 #include "Keypad.h"
2
3 byte rowPin[4] = {13, 12, 11, 10};
4 byte colPin[4] = { 9, 8, 7, 6};
5
6 char keyStrings[4][4] = {
7     {'1', '2', '3', 'A'},
8     {'4', '5', '6', 'B'},
9     {'7', '8', '9', 'C'},
10    {'*', '0', '#', 'D'}
11 };
12
13 int lastTime = 0;
14 int debounceTime = 20;
15
16 int pressKeyRow=0;
17 int pressKeyCol=0;
18 bool pressState=false;
19
20 void keyInit(void) {
21     for (int r = 0; r < sizeof(rowPin); r++) {
22         pinMode(colPin[r], OUTPUT);
23         digitalWrite(colPin[r], HIGH);
24         pinMode(rowPin[r], INPUT_PULLUP);
25     }
26     for (int c = 0; c < sizeof(colPin); c++) {
27         pinMode(colPin[c], OUTPUT);
28         digitalWrite(colPin[c], HIGH);
29     }
30 }
31
32 void keyScan(bool state) {
33     for (int c = 0; c < sizeof(colPin); c++) {
34         digitalWrite(colPin[c], LOW);
35         for (int r = 0; r < sizeof(rowPin); r++) {
36             if (digitalRead(rowPin[r]) == LOW) {
37                 while (state == true && digitalRead(rowPin[r]) == LOW);
38                 digitalWrite(colPin[c], HIGH);
39                 pressKeyRow=r;
40                 pressKeyCol=c;
41                 pressState=true;
42             }
43         }
44     }
45 }
```

```

44     digitalWrite(colPin[c], HIGH);
45   }
46 }
47
48 char getKey(bool state) {
49   if (millis() - lastTime > debounceTime) {
50     pressState = false;
51     keyScan(state);
52     lastTime = millis();
53     if(pressState==true)
54       return keyStrings[pressKeyRow][pressKeyCol];
55     else
56       return '\0';
57   }
58 }
```

Include the header file, define the pins to control the keypad's rows and columns and define an array to store the key values of the keypad being pressed.

You can modify the following code to change key values and the pins controlling the keypad.

```

1 #include "Keypad.h"
2
3 byte rowPin[4] = {13, 12, 11, 10};
4 byte colPin[4] = { 9, 8, 7, 6};
5
6 char keyStrings[4][4] = {
7   {'1', '2', '3', 'A'},
8   {'4', '5', '6', 'B'},
9   {'7', '8', '9', 'C'},
10  {'*', '0', '#', 'D'}
11 };
```

The following is the program code:

```

1 #include "Keypad.h"
2
3 void setup() {
4   Serial.begin(115200);
5   keyInit();
6 }
7
8 void loop() {
9   char keyValue = getKey(0);
10  if (keyValue != '\0')
11    Serial.println(keyValue);
12  delay(50);
13 }
```



keyInit() function is called to initialize the pins controlling the keypad.

```
5     keyInit();
```

getKey() is called to scan the matrix keyboard and return "\0" if no key is detected being pressed; and it returns the key value character of the pressed key when a key is detected to be pressed

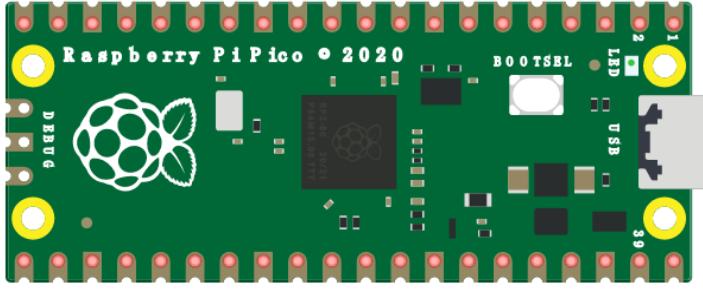
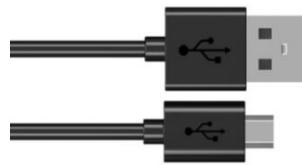
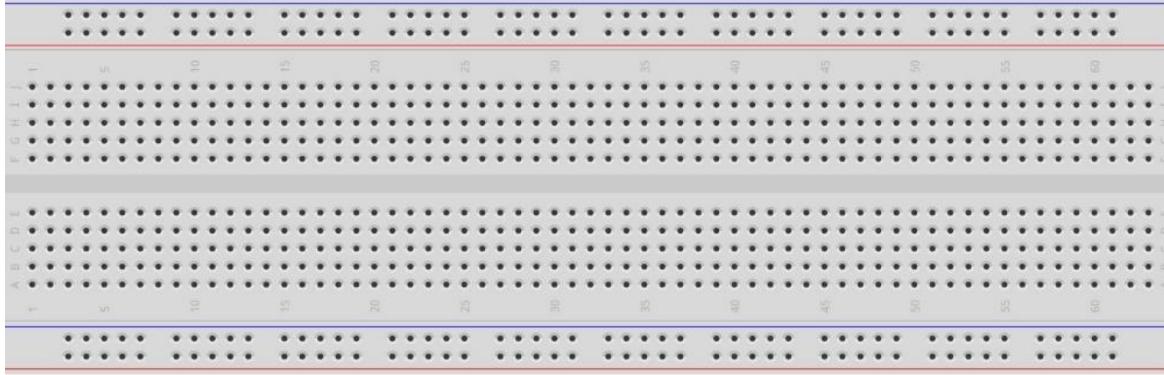
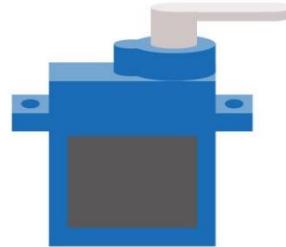
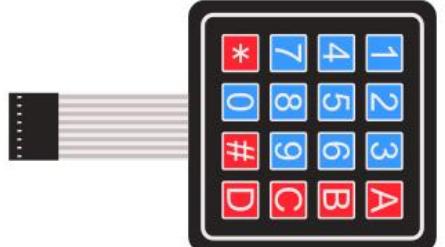
When the parameter of the getKey() function is 1, the matrix keyboard scans in an inching mode. In this mode, when the key is pressed and not released, the program will stop execution until the key is released. When the parameter is 0, the matrix keyboard scans in a continuous mode. In this mode, the program will not stop execution because the key is not released.

```
9     char keyValue = getKey(0);
```

Project 23.2 Keypad Door

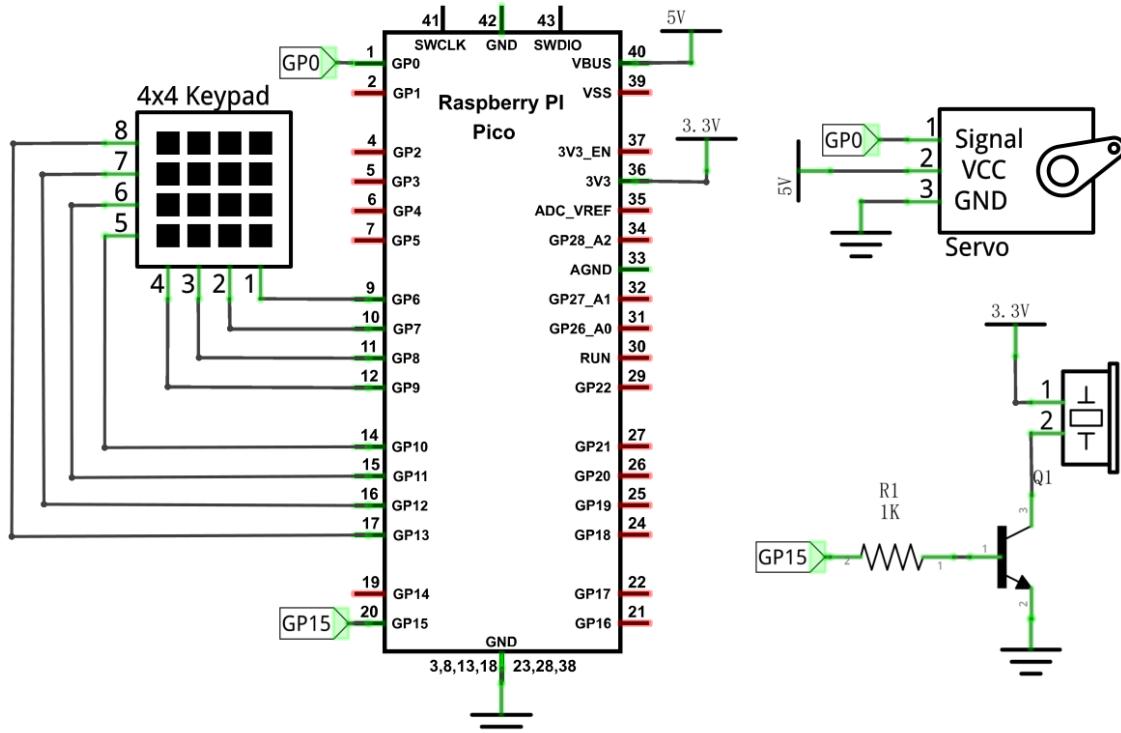
In this project, we use keypad as a keyboard to control the action of the servo motor.

Component List

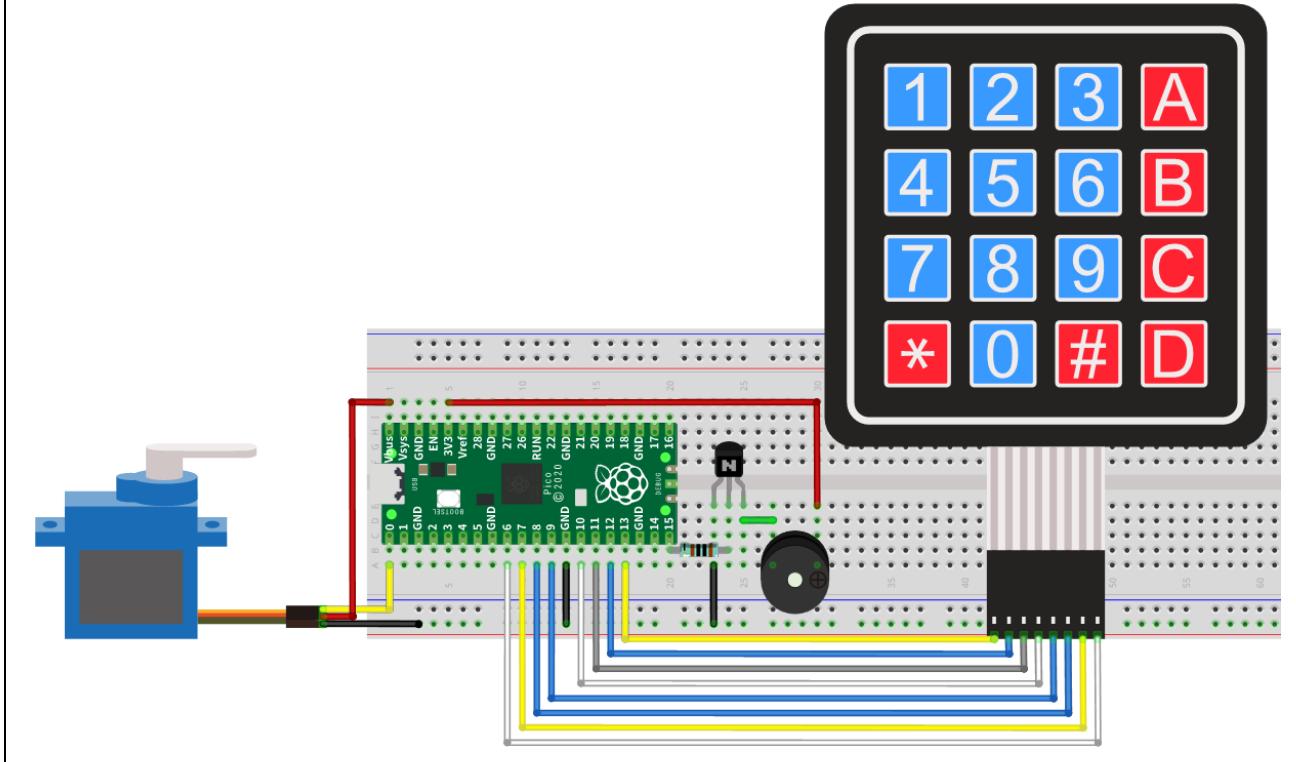
Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
	
Jumper	Servo x1
	
4x4 Matrix Keypad x1	
NPN transistor x1 (S8050)	Active buzzer x1
	
	Resistor 1kΩ x1
	

Circuit

Schematic diagram

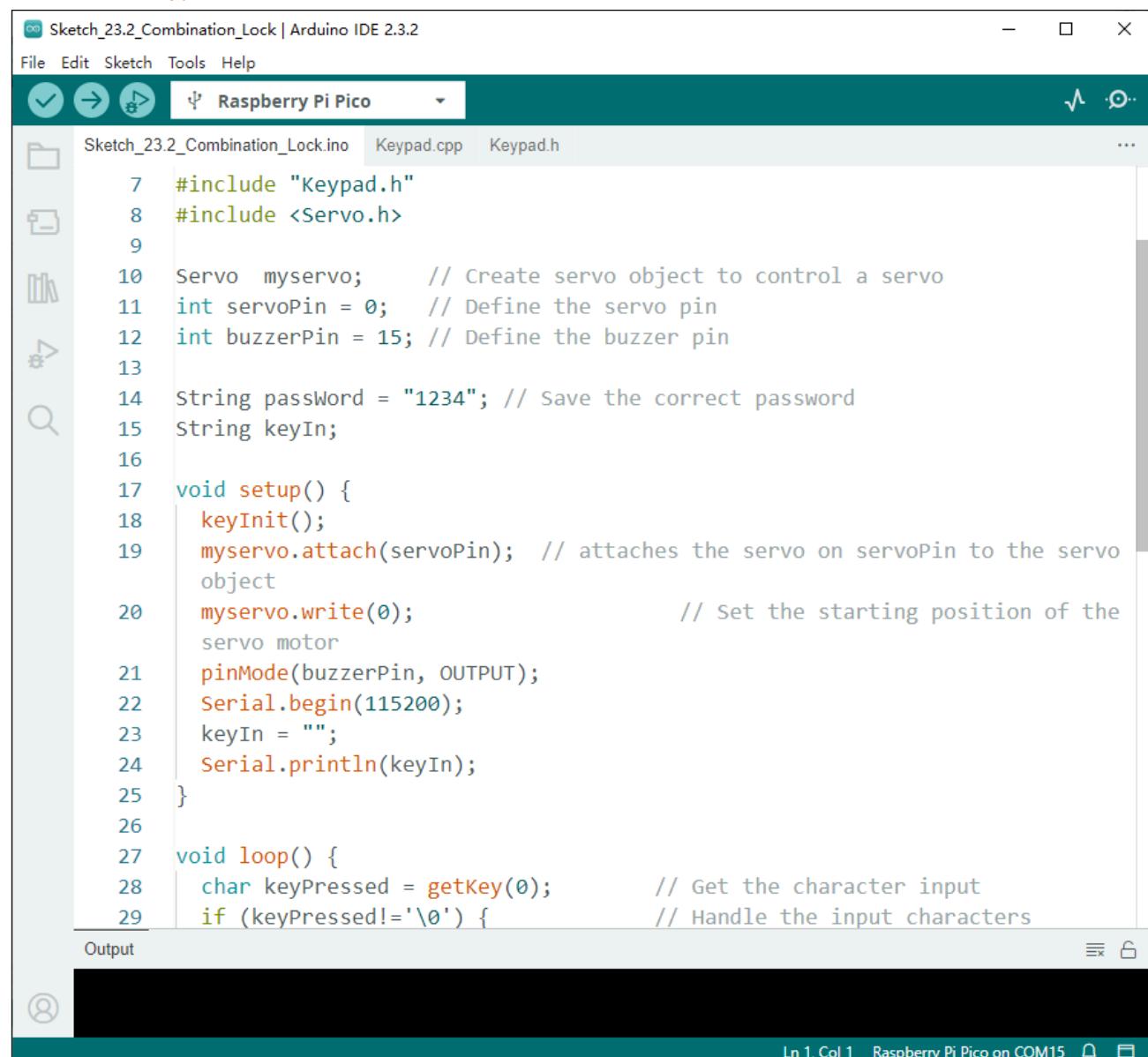


Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Sketch

Sketch_23.2_Keypad_Door



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

- Title Bar:** Sketch_23.2_Combination_Lock | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Selection:** Sketch_23.2_Combination_Lock.ino
- Board Selection:** Raspberry Pi Pico
- Code Area:** Displays the C++ code for the sketch. The code includes includes for Keypad.h and Servo.h, defines servo and buzzer pins, sets up a password ("1234"), initializes a servo, and handles character input in the loop.
- Output Area:** Shows the message "Ln 1, Col 1 Raspberry Pi Pico on COM15".

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

The following is the program code:

```
1 #include "Keypad.h"
2 #include <Servo.h>
3
4 Servo myservo; // Create servo object to control a servo
5 int servoPin = 0; // Define the servo pin
6 int buzzerPin = 15; // Define the buzzer pin
7
8 String passWord = "1234"; // Save the correct password
9 String keyIn;
10
11 void setup() {
12     keyInit();
13     myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
14     myservo.write(0); // Set the starting position of the servo motor
15     pinMode(buzzerPin, OUTPUT);
16     Serial.begin(115200);
17     keyIn = "";
18     Serial.println(keyIn);
19 }
20
21 void loop() {
22     char keyPressed = getKey(0); // Get the character input
23     if (keyPressed != '\0') { // Handle the input characters
24         digitalWrite(buzzerPin, HIGH); // Make a prompt tone each time press the key
25         delay(200);
26         digitalWrite(buzzerPin, LOW);
27         keyIn += keyPressed; // Save the input characters
28         Serial.println(keyPressed); // Judge the correctness after input
29         if (keyIn.length() == 4) {
30             bool isRight = true; // Save password is correct or not
31             if (passWord != keyIn) {
32                 isRight = !true;
33             }
34             if (isRight) { // If the input password is right
35                 myservo.attach(servoPin);
36                 myservo.write(90); // Open the switch
37                 delay(2000); // Delay a period of time
38                 myservo.write(0); // Close the switch
39                 Serial.println("passWord right!");
34             }
35             else { // If the input password is wrong
36                 digitalWrite(buzzerPin, HIGH); // Make a wrong password prompt tone
37                 delay(1000);
38             }
39         }
40     }
41 }
```

```

44     digitalWrite(buzzerPin, LOW);
45     Serial.println("passWord error!");
46   }
47   keyIn = ""; // Reset the number of the input characters to 0
48 }
49 }
50 delay(200);
51 }
```

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

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

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

```

22 char keyPressed = getKey(0);           // Get the character input
23 if (keyPressed != '\0') {             // Handle the input characters
24   digitalWrite(buzzerPin, HIGH);      // Make a prompt tone each time press the key
25   delay(200);
26   digitalWrite(buzzerPin, LOW);
27   keyIn += keyPressed;
```

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

```

29 if (keyIn.length() == 4) {
30   bool isRight = true;                // Save password is correct or not
31   if (passWord != keyIn) {
32     isRight = !true;
33 }
```

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

```

34 if (isRight) {                      // If the input password is right
35   myservo.attach(servoPin);
36   myservo.write(90);                 // Open the switch
37   delay(2000);                     // Delay a period of time
38   myservo.write(0);                  // Close the switch
39   Serial.println("passWord right!");
40 }
41 else {                            // If the input password is wrong
42   digitalWrite(buzzerPin, HIGH); // Make a wrong password prompt tone
43   delay(1000);
44   digitalWrite(buzzerPin, LOW);
45   Serial.println("passWord error!");
46 }
```

Finally, remember to empty the keyInNum every time.

```
47 keyIn = ""; // Reset the number of the input characters to 0
```



Chapter 24 Infrared Remote

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

Project 24.1 Infrared Remote Control

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

Component List

Raspberry Pi Pico x1	A green printed circuit board with a central Broadcom SoC, labeled "Raspberry Pi Pico • 2020". It has various pins, a USB port, and a power jack.	USB cable x1	Two standard black USB-A to USB-A cables.
Breadboard x1	A breadboard with a grid of 40 columns and 24 rows of holes, labeled with letters A through H and numbers 1 through 40.		
Jumper	A long, thin black jumper wire with two black alligator clips at the ends.	Infrared Remote x1 (May need CR2025 battery x1, please check the holder)	A white remote control with a numeric keypad (0-9), arrows, and function keys like TEST, C, and MENU.
Infrared Remote x1	The physical infrared remote control device.	Resistor 10kΩ x1	A cylindrical resistor with a red band indicating 10 kilohms.

Component Knowledge

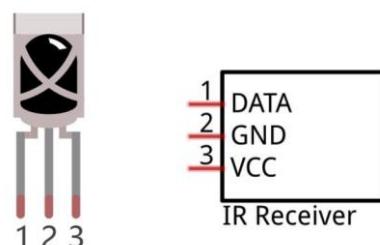
Infrared Remote

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



Infrared receiver

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





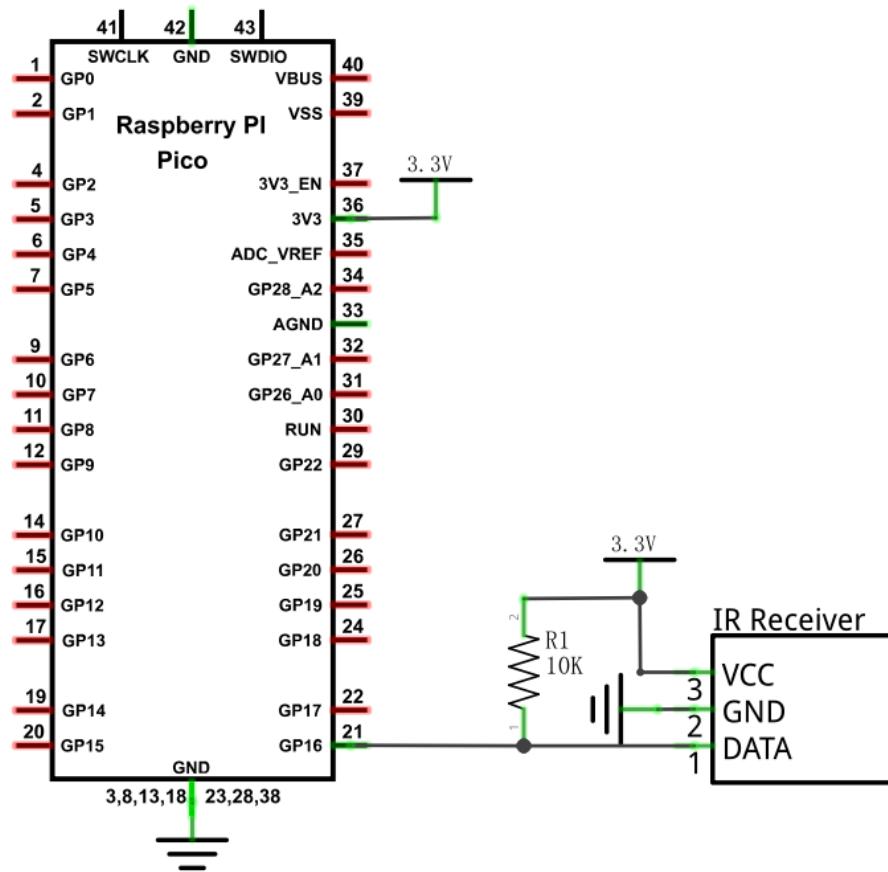
When you use the infrared remote control, it sends a key value to the receiving circuit according to the pressed key. We can program the Raspberry Pi Pico to do things like lighting, when a key value is received.

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

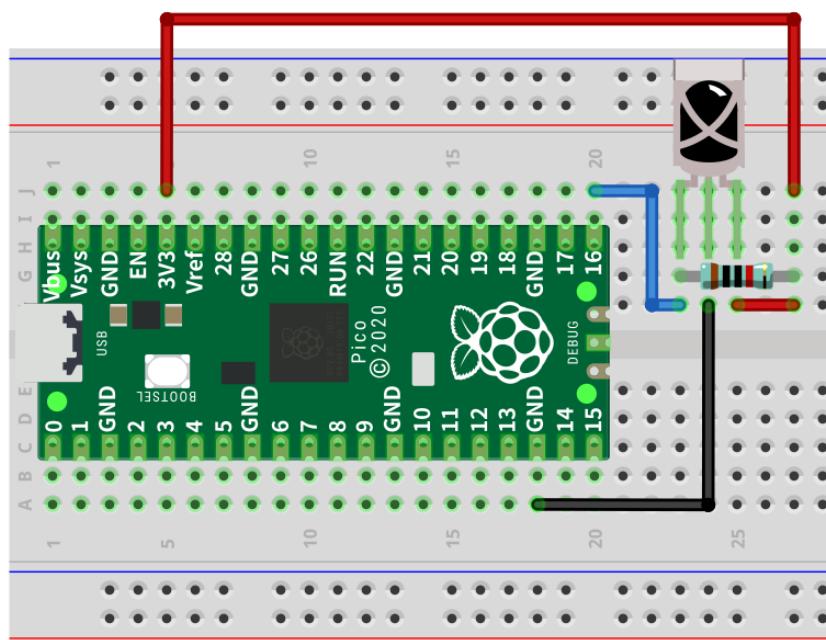
ICON	KEY Value	ICON	KEY Value
	FFA25D		FFB04F
	FFE21D		FF30CF
	FF22DD		FF18E7
	FF02FD		FF7A85
	FFC23D		FF10EF
	FFE01F		FF38C7
	FFA857		FF5AA5
	FF906F		FF42BD
	FF6897		FF4AB5
	FF9867		FF52AD

Circuit

Schematic diagram



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

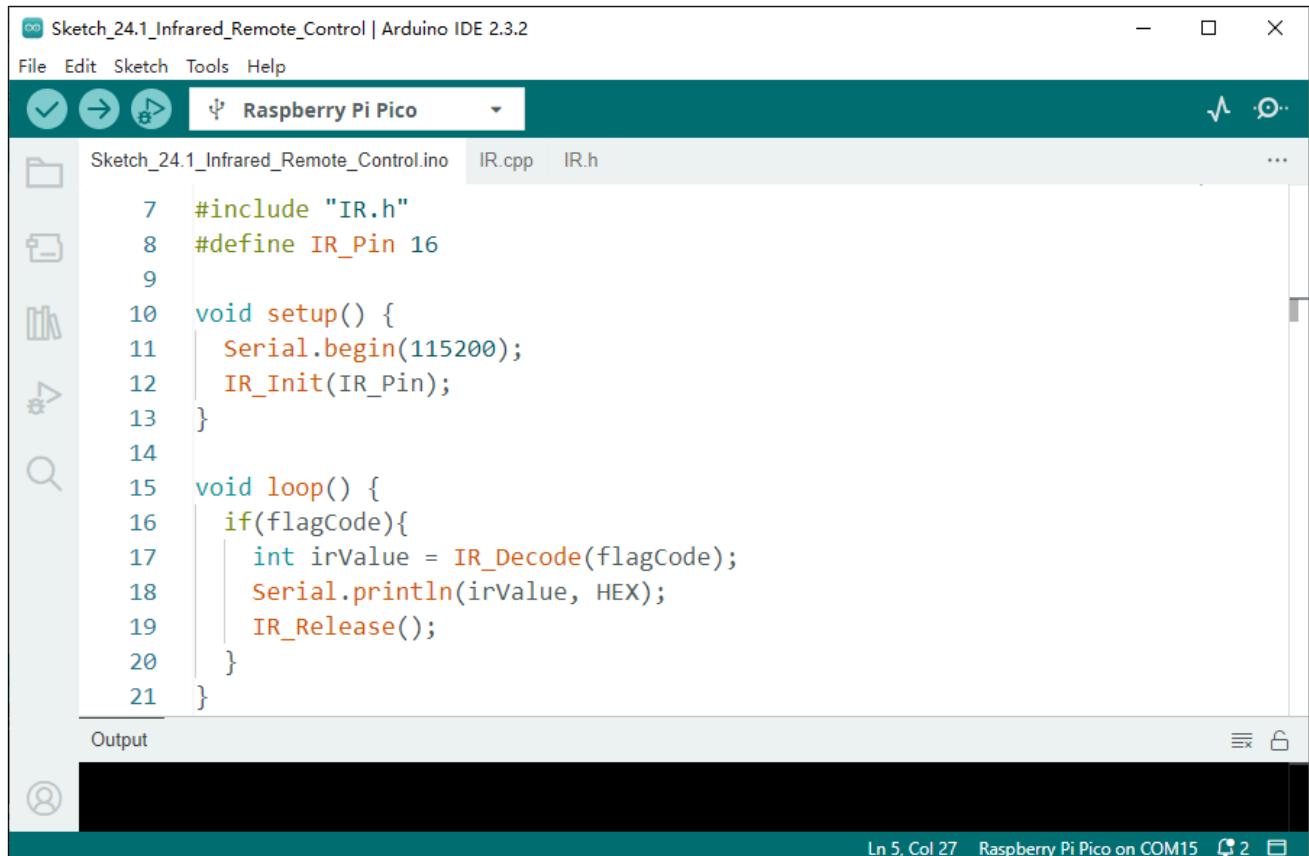


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Sketch

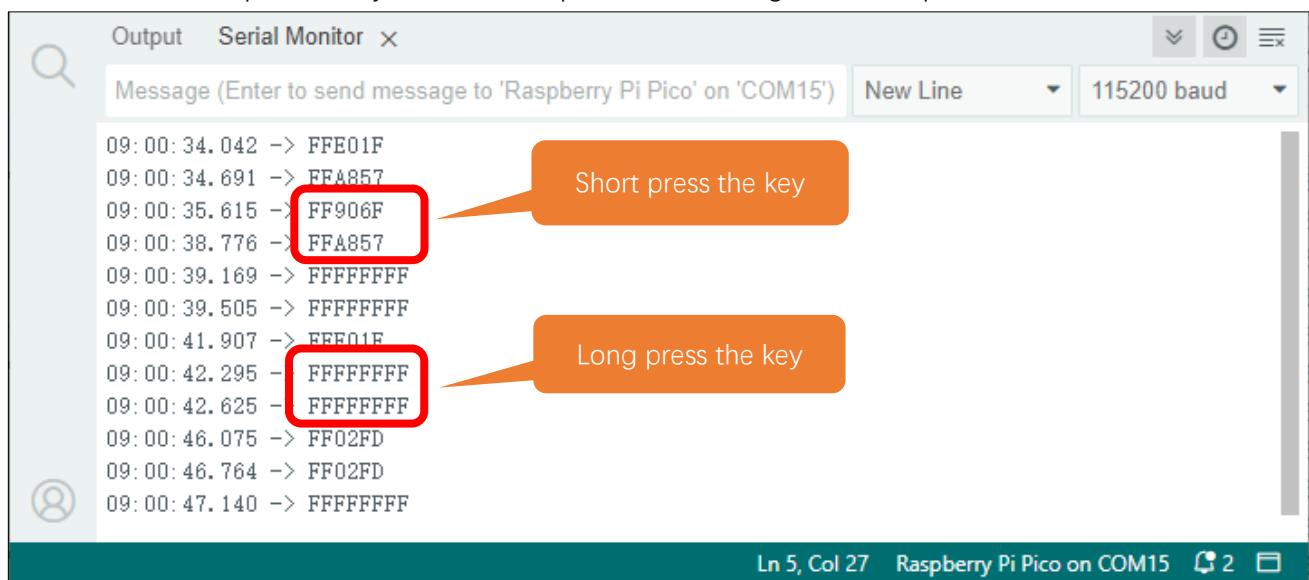
This sketch uses the infrared receiving tube to receive the value sent from the infrared remote control, and print it out via the serial port.

Sketch_24.1_Infrared_Remote_Control



```
#include "IR.h"
#define IR_Pin 16
void setup() {
    Serial.begin(115200);
    IR_Init(IR_Pin);
}
void loop() {
    if(flagCode){
        int irValue = IR_Decode(flagCode);
        Serial.println(irValue, HEX);
        IR_Release();
    }
}
```

Download the code to Pico, open the serial port monitor, set the baud rate to 115200, and press the IR remote control, the pressed keys value will be printed out through the serial port.



Time	Action	Hex Value
09:00:34.042	Short press the key	FFE01F
09:00:34.691		FFEA857
09:00:35.615	Short press the key	FF906F
09:00:38.776	Long press the key	FFA857
09:00:39.169		FFFFFF
09:00:39.505		FFFFFF
09:00:41.907	Short press the key	FFE01F
09:00:42.295	Long press the key	FFFFFF
09:00:42.625	Long press the key	FFFFFF
09:00:46.075		FF02FD
09:00:46.764		FF02FD
09:00:47.140		FFFFFF

IR.cpp

```
1 #include "IR.h"
2
3 int logList[32];
4 unsigned long startTime;
5 int endTime, end2Time;
6 int flagCode = 0;
7 int irPin;
8 bool irState = true;
9
10 void IR_Init(int pin) {
11     irPin = pin;
12     pinMode(irPin, INPUT_PULLUP);
13     attachInterrupt(digitalPinToInterrupt(irPin), IR_Read, CHANGE);
14 }
15
16 void IR_Read() {
17     if (irState == true) {
18         unsigned long lowTime, highTime, intervalTime;
19         int num = 0;
20         while (digitalRead(irPin) == LOW) {
21             startTime = micros();
22             while (digitalRead(irPin) == LOW) {
23                 lowTime = micros();
24             }
25             intervalTime = lowTime - startTime;
26             while (digitalRead(irPin) == HIGH) {
27                 highTime = micros();
28                 intervalTime = highTime - lowTime;
29                 if (intervalTime > 10000) {
30                     end2Time = millis();
31                     if (num == 32) {
32                         flagCode = 1;
33                         endTime = millis();
34                     }
35                     else if (num == 0 && end2Time - endTime > 300 && end2Time - endTime < 400) {
36                         flagCode = 2;
37                         endTime = millis();
38                     }
39                     return;
40                 }
41             }
42             if (intervalTime < 2000) {
43                 if (intervalTime < 700) {
```

```

44         logList[num ++] = 0;
45     }
46     else {
47         logList[num ++] = 1;
48     }
49 }
50 }
51 }
52 }

53

54 unsigned long IR_Decode(int &code) {
55     unsigned long irData = 0;
56     irState=false;
57     if (code == 1) {
58         code = 0;
59         for (int i = 0; i < 32; i++) {
60             if (logList[i] == 0) {
61                 irData <<= 1;
62             }
63             else {
64                 irData <<= 1;
65                 irData++;
66             }
67             logList[i] = 0;
68         }
69     }
70     if (code == 2) {
71         code = 0;
72         irData = 0xffffffff;
73     }
74     return irData;
75 }
76 }

77

78 void IR_Release() {
79     irState=true;
80 }
```

When the IR_Init() function is called, Pico initializes the infrared received pin and sets the external interrupt, associating it with the IR_Read() function. Every time the infrared receives data, external interrupt calls IR_Read() function to receive data, and resets the bit flag.

	<pre> extern int flagCode; void IR_Init(int pin); void IR_Read();</pre>
--	---

You can check whether flagCode has been reset, If it is reset, call IR_Decode() to decode the infrared data.

Note: once IR_Decode() is called, infrared receiver will not receive data until IR_Release() is called.

```
unsigned long IR_Decode(int &code);  
void IR_Release();
```

The following is the program code:

```
1 #include "IR.h"  
2 #define IR_Pin 16  
3  
4 void setup() {  
5     Serial.begin(115200);  
6     IR_Init(IR_Pin);  
7 }  
8  
9 void loop() {  
10    if(flagCode){  
11        int irValue = IR_Decode(flagCode);  
12        Serial.println(irValue, HEX);  
13        IR_Release();  
14    }  
15 }
```

IR_Init() is called to initialize infrared receiving pin GP16, enable external interrupt and associate it with GP16.

```
6     IR_Init(IR_Pin);
```

In loop(), determines whether infrared bit flag is reset. If it is, IR_Decode() is called to decode the data and print them out via serial monitor.

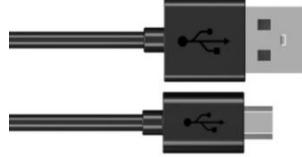
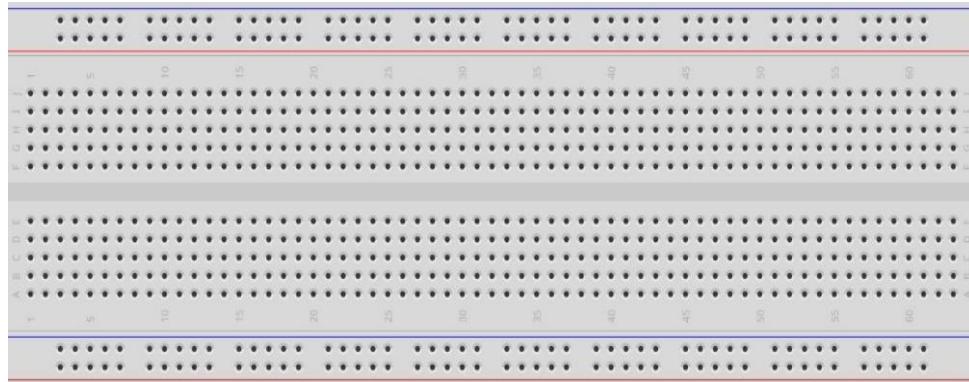
After using the infrared decoding function IR_Decode(), you need to call IR_Release() to release the infrared data receiving function. Otherwise, it will not receiver new infrared data again.

```
10 if(flagCode){  
11     int irValue = IR_Decode(flagCode);  
12     Serial.println(irValue, HEX);  
13     IR_Release();  
14 }
```

Project 24.2 Control LED through Infrared Remote

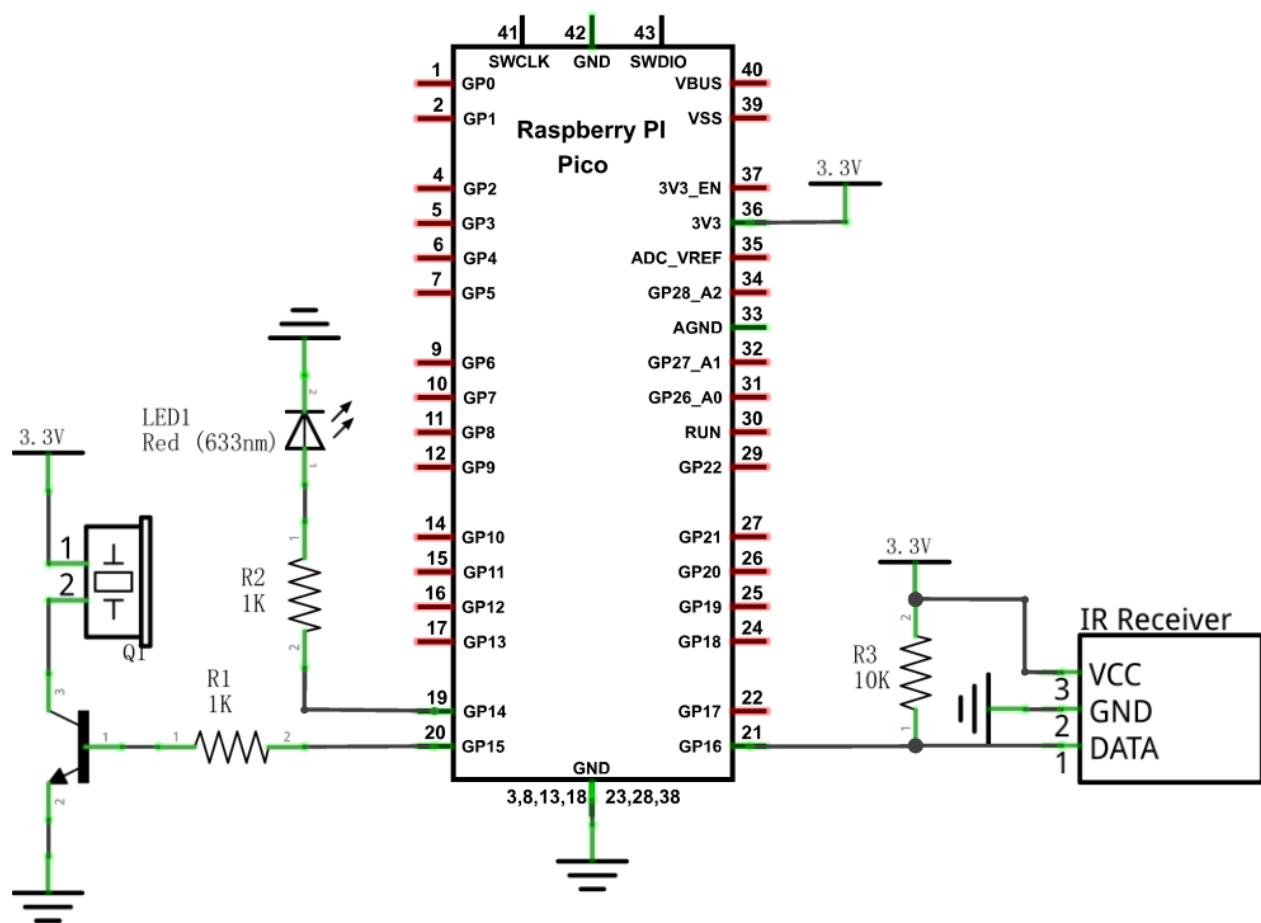
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

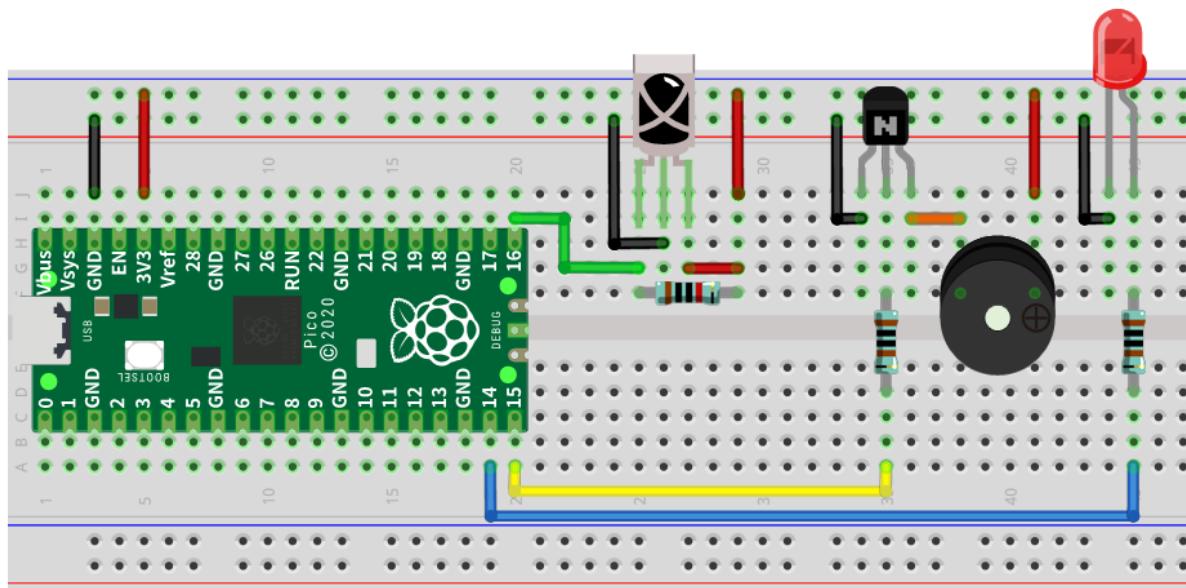
Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
Jumper	Infrared Remote x1 (May need CR2025 battery x1, please check the battery holder)
LED x1	Active buzzer x1
	
Resistor 1kΩ x2	
Infrared receiver x1	NPN transistor x1 (S8050)
	
Resistor 10kΩ x1	

Circuit

Schematic diagram



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





Sketch

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

[Sketch_24.2_Control_LED_through_Infrared_Remote](#)

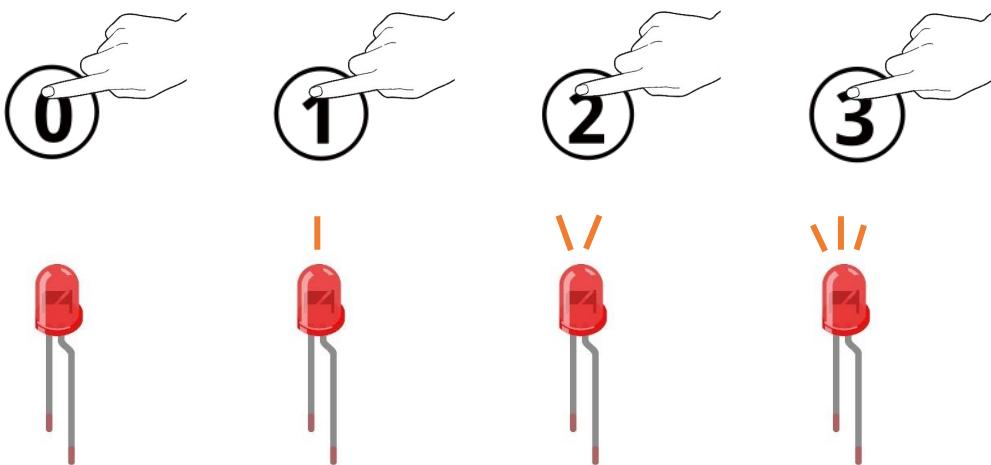
```

Sketch_24.2_Control_LED_through_Infrared_Remote | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_24.2_Control_LED_through_Infrared_Remote.ino IR.cpp IR.h ...
7 #include "IR.h"
8
9 #define irPin 16
10 #define ledPin 14
11 #define buzzerPin 15
12
13 void setup() {
14     Serial.begin(115200);
15     IR_Init(irPin);
16     pinMode(ledPin, OUTPUT);
17     pinMode(buzzerPin, OUTPUT);
18 }
19
20 void loop() {
21     if(flagCode){
22         int irValue = IR_Decode(flagCode);
23         Serial.println(irValue, HEX);
24         handleControl(irValue);
25         IR_Release();
26     }
27 }

```

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

Rendering:



The following is the program code:

```
1 #include "IR.h"
2
3 #define irPin 16
4 #define ledPin 14
5 #define buzzerPin 15
6
7 void setup() {
8     Serial.begin(115200);
9     IR_Init(irPin);
10    pinMode(ledPin, OUTPUT);
11    pinMode(buzzerPin, OUTPUT);
12 }
13
14 void loop() {
15     if(flagCode) {
16         int irValue = IR_Decode(flagCode);
17         Serial.println(irValue, HEX);
18         handleControl(irValue);
19         IR_Release();
20     }
21 }
22
23 void handleControl(unsigned long value) {
24     digitalWrite(buzzerPin, HIGH);
25     delay(100);
26     digitalWrite(buzzerPin, LOW);
27     // Handle the commands
28     switch (value) {
29         case 0xFF6897:           // Receive the number '0'
30             analogWrite(ledPin, 0); // Turn off LED
31             break;
32         case 0xFF30CF:           // Receive the number '1'
33             analogWrite(ledPin, 50); // Dimmest brightness
34             break;
35         case 0xFF18E7:           // Receive the number '2'
36             analogWrite(ledPin, 100); // Medium brightness
37             break;
38         case 0xFF7A85:           // Receive the number '3'
39             analogWrite(ledPin, 255); // Strongest brightness
40             break;
41     }
42 }
```



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

```

23 void handleControl(unsigned long value) {
24     digitalWrite(buzzerPin, HIGH);
25     delay(100);
26     digitalWrite(buzzerPin, LOW);
27     // Handle the commands
28     switch (value) {
29         case 0xFF6897:           // Receive the number '0'
30             analogWrite(ledPin, 0); // Turn off LED
31             break;
32         case 0xFF30CF:           // Receive the number '1'
33             analogWrite(ledPin, 50); // Dimmest brightness
34             break;
35         case 0xFF18E7:           // Receive the number '2'
36             analogWrite(ledPin, 100); // Medium brightness
37             break;
38         case 0xFF7A85:           // Receive the number '3'
39             analogWrite(ledPin, 255); // Strongest brightness
40             break;
41     }
42 }
```

In the loop() function, each time the infrared data is received, it is decoded and printed out through the serial monitor, and the handleControl() function is called to control the LED and buzzer to execute the corresponding code.

```

14 void loop() {
15     if(flagCode) {
16         int irValue = IR_Decode(flagCode);
17         Serial.println(irValue, HEX);
18         handleControl(irValue);
19         IR_Release();
20     }
21 }
```

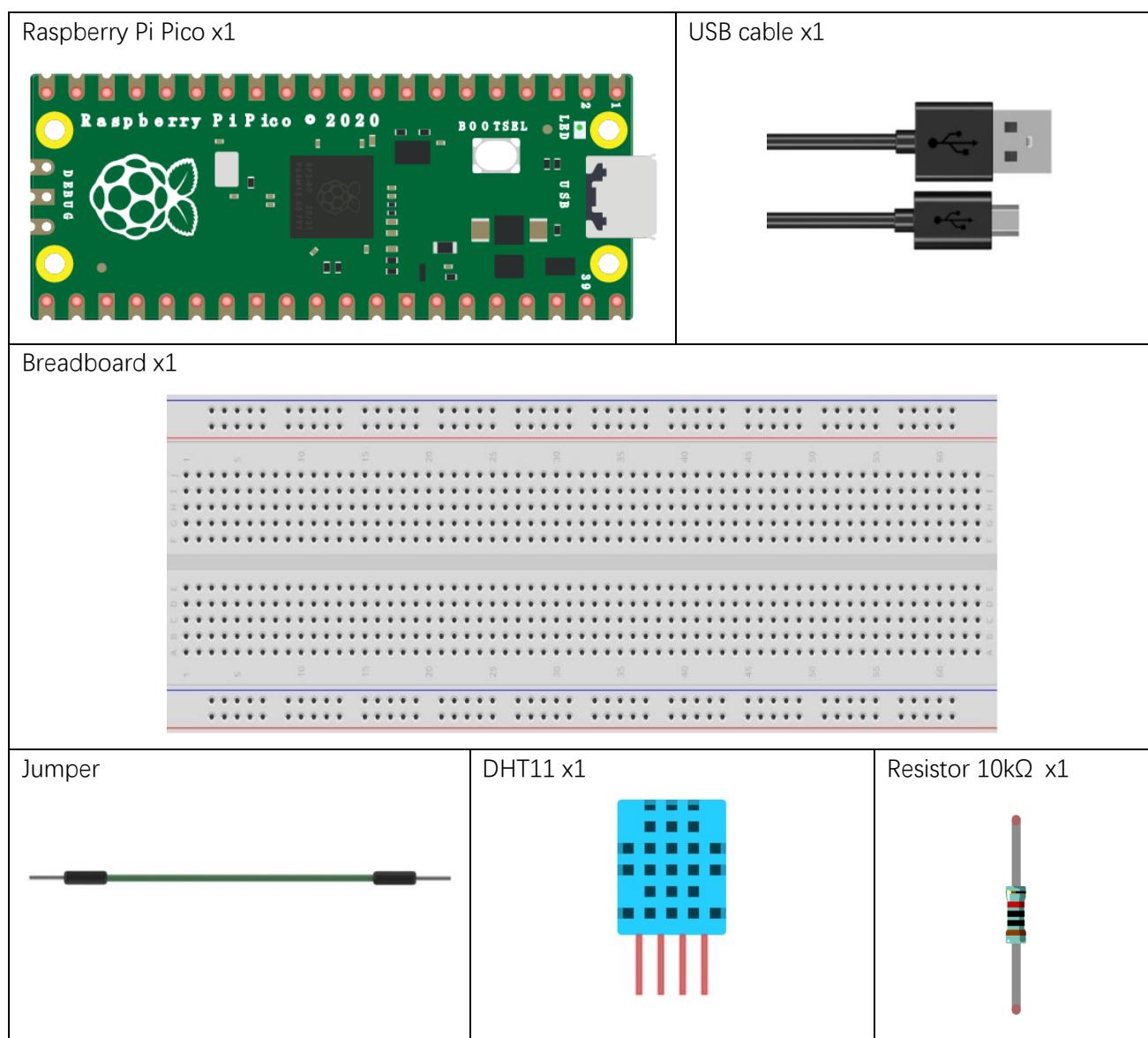
Chapter 25 Hygrothermograph DHT11

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

Project 25.1 Hygrothermograph

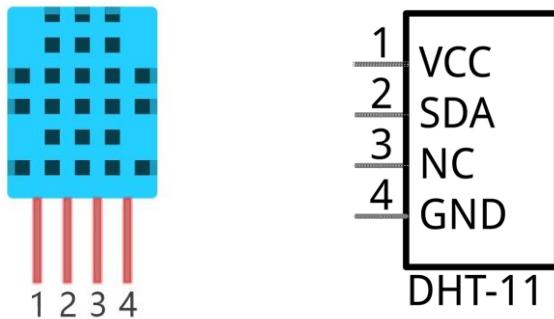
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the Raspberry Pi Pico to read Temperature and Humidity data of the DHT11 Module.

Component List



Component Knowledge

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



DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

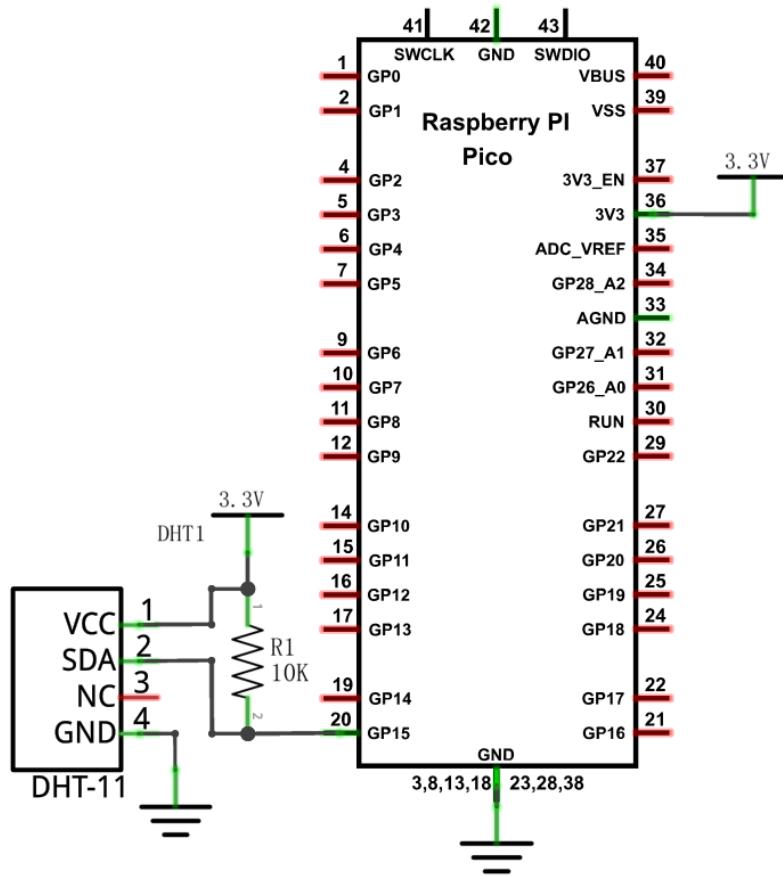
After being powered up, it will initialize in 1S's time. Its operating voltage is within the range of 3.3V-5.5V.

The SDA pin is a data pin, which is used to communicate with other devices.

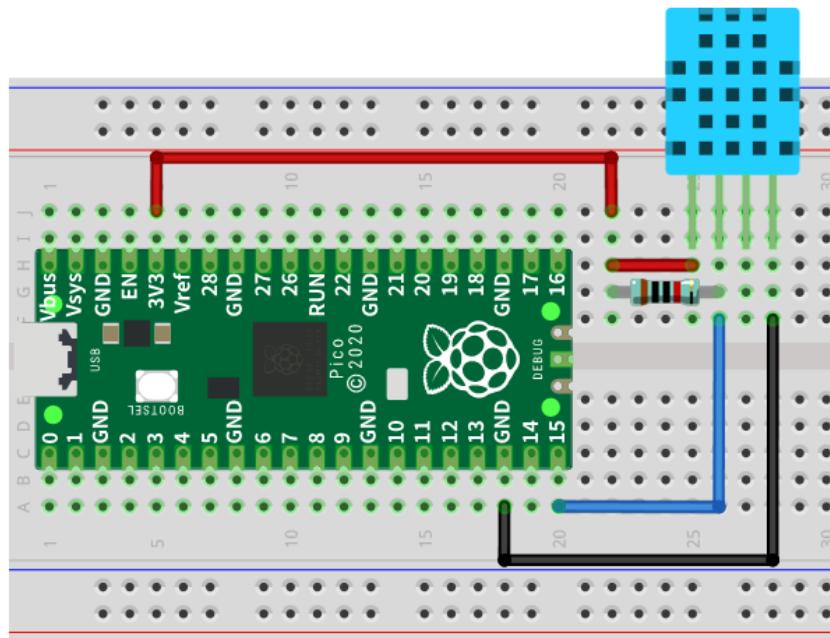
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



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





Sketch

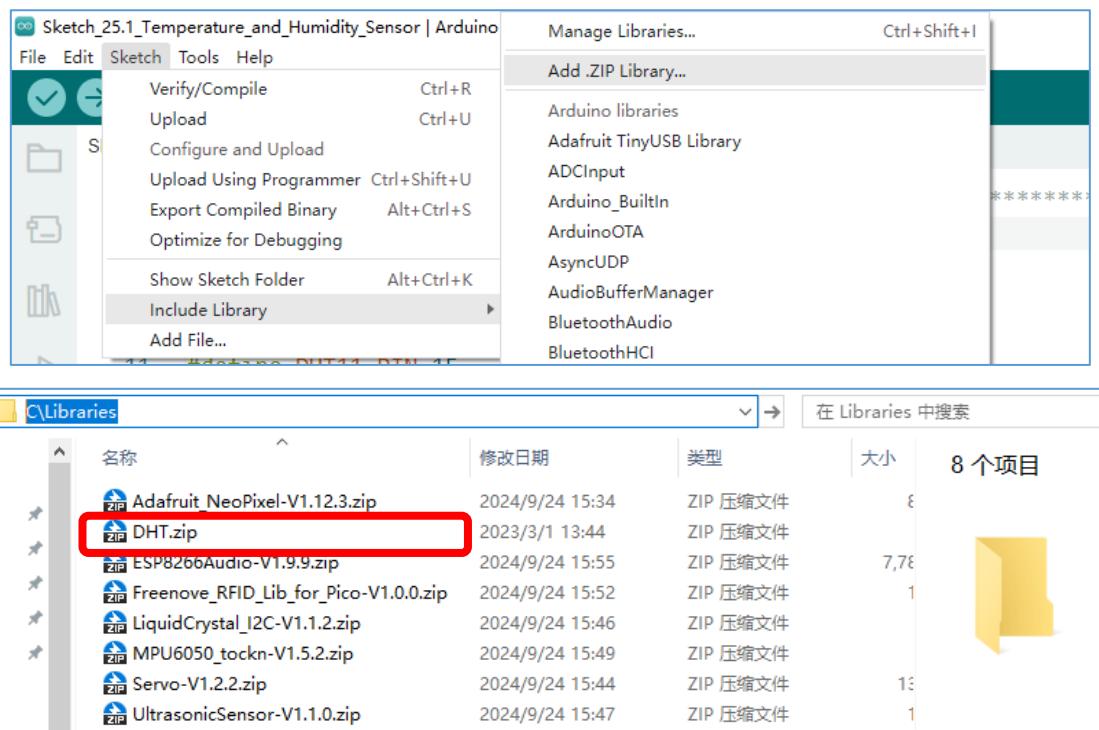
How to install the library

We use a third-party library DHT for this project. If you have not installed it yet, please do so first.

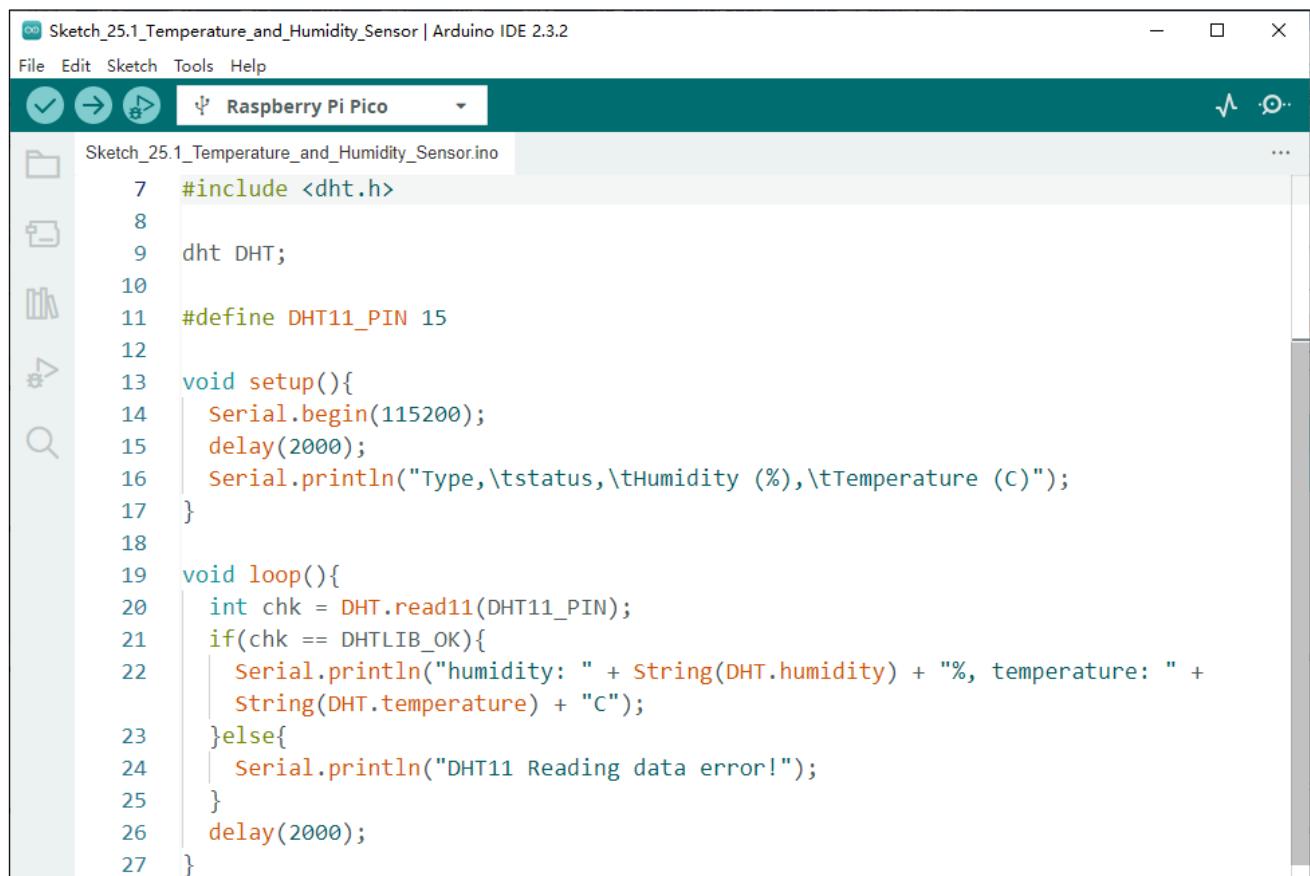
Steps are as follows:

Open **Arduino>Sketch>Include Library>Add .ZIP Library...**

Select the provided “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Libraries\DHT.zip**”.



Sketch_25.1_Temperature_and_Humidity_Sensor



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

- Title Bar:** Sketch_25.1_Temperature_and_Humidity_Sensor | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and others.
- Sketch Selection:** Raspberry Pi Pico
- Code Area:** Displays the following C++ code for a DHT11 sensor connected to pin 15:

```
Sketch_25.1_Temperature_and_Humidity_Sensor.ino
7 #include <dht.h>
8
9 dht DHT;
10
11 #define DHT11_PIN 15
12
13 void setup(){
14     Serial.begin(115200);
15     delay(2000);
16     Serial.println("Type,\tstatus,\tHumidity (%),\tTemperature (c)");
17 }
18
19 void loop(){
20     int chk = DHT.read11(DHT11_PIN);
21     if(chk == DHTLIB_OK){
22         Serial.println("humidity: " + String(DHT.humidity) + "%, temperature: " +
23             String(DHT.temperature) + "C");
24     }else{
25         Serial.println("DHT11 Reading data error!");
26     }
27 }
```



Compile and upload the code to the Pico, open the serial monitor, and set the baud rate to 115200. Print out data of temperature and humidity sensor via the serial port.

The screenshot shows the Arduino Serial Monitor interface. The title bar says "Output Serial Monitor". The message area contains the following text:

```

09:15:29.790 -> Type, status, Humidity (%), Temperature (C)
09:15:29.790 -> humidity: 59.00%, temperature: 28.00C
09:15:31.829 -> humidity: 62.00%, temperature: 27.80C
09:15:33.834 -> humidity: 61.00%, temperature: 27.80C
09:15:35.856 -> humidity: 95.00%, temperature: 28.10C
09:15:37.911 -> humidity: 95.00%, temperature: 28.00C
09:15:39.933 -> humidity: 95.00%, temperature: 28.00C
09:15:41.939 -> humidity: 85.00%, temperature: 28.00C
09:15:43.962 -> humidity: 79.00%, temperature: 28.00C
09:15:46.015 -> humidity: 75.00%, temperature: 28.00C
09:15:47.989 -> humidity: 71.00%, temperature: 28.10C
09:15:50.028 -> humidity: 69.00%, temperature: 28.00C
09:15:52.080 -> humidity: 67.00%, temperature: 28.00C

```

The status bar at the bottom right shows "Ln 7, Col 13 Raspberry Pi Pico on COM15" and icons for message count and refresh.

The following is the program code:

```

1 #include <dht.h>
2
3 dht DHT;
4 #define DHT11_PIN 15
5
6 void setup() {
7     Serial.begin(115200);
8     delay(2000);
9     Serial.println("Type, \tstatus, \tHumidity (%), \tTemperature (C)");
10 }
11
12 void loop() {
13     int chk = DHT.read11(DHT11_PIN);
14     if(chk == DHTLIB_OK){
15         Serial.println("humidity: " + String(DHT.humidity) + "%, temperature: " +
String(DHT.temperature) + "C");
16     }else{
17         Serial.println("DHT11 Reading data error!");
18     }
19     delay(2000);
20 }

```

Before using dht11, we need to include a header file. Apply for a DHT object and define the pin controlling DHT as GP15.

```
1 #include <dht.h>
2
3 dht DHT;
4 #define DHT11_PIN 15
```

Read11() is used to read DHT11 data and assign the return value to variable chk.

```
13 int chk = DHT.read11(DHT11_PIN);
```

If the return value of the read11() function is not equal to DHTLIB_OK, it means that the data reading failed; If they equals, humidity() and temperature() are called to obtain the temperature and humidity data of the current environment, and print it out through the serial port.

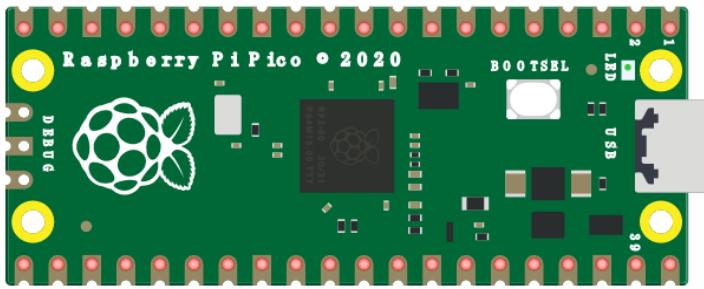
```
14 if(chk == DHTLIB_OK) {
15     Serial.println("humidity: " + String(DHT.humidity) + "%, temperature: " +
16     String(DHT.temperature) + "C");
17 } else{
18     Serial.println("DHT11 Reading data error!");
19 }
```

Project 25.2 Hygrothermograph

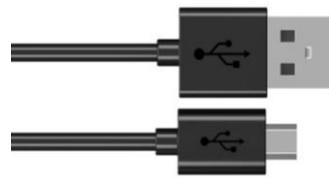
In this project, we use I2C-LCD1602 to display data collected by DHT11.

Component List

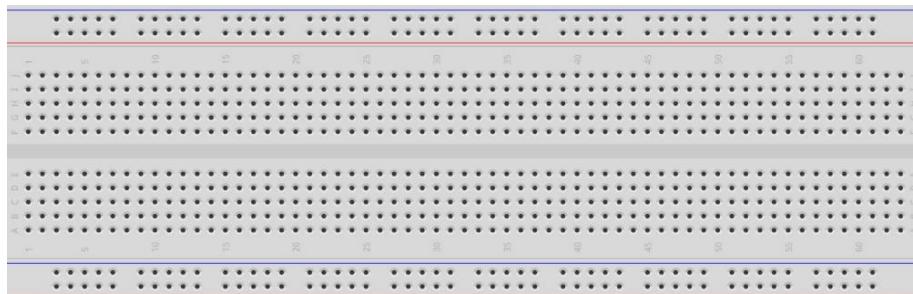
Raspberry Pi Pico x1



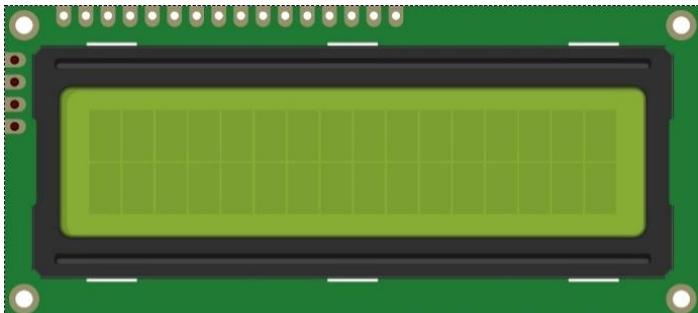
USB cable x1



Breadboard x1



LCD1602 Module x1



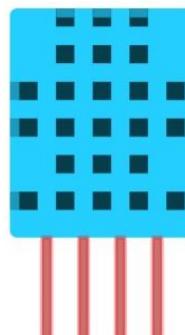
Resistor 10kΩ x1



Jumper

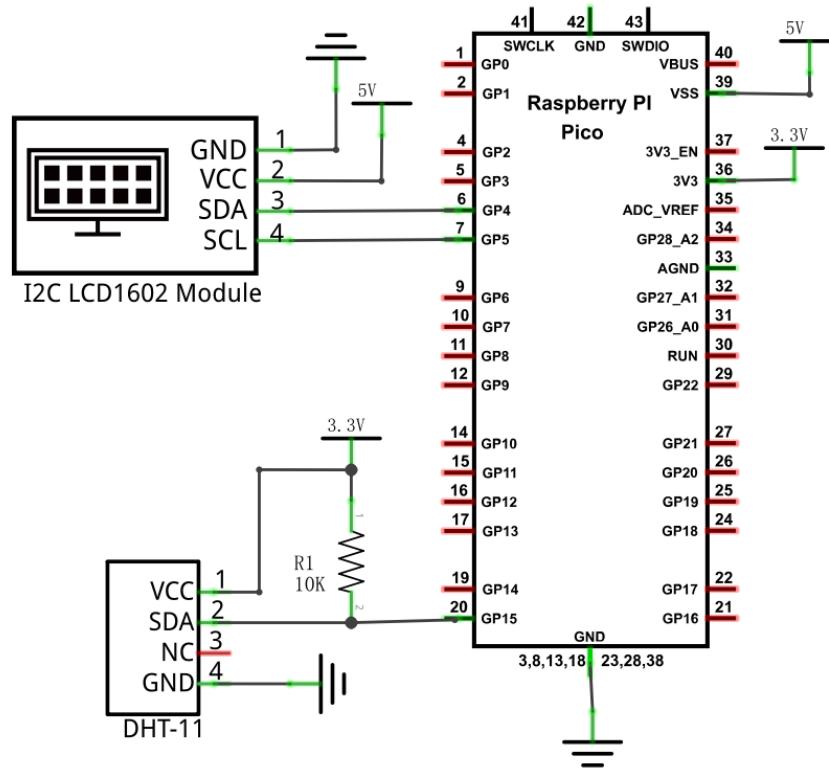


DHT11 x1

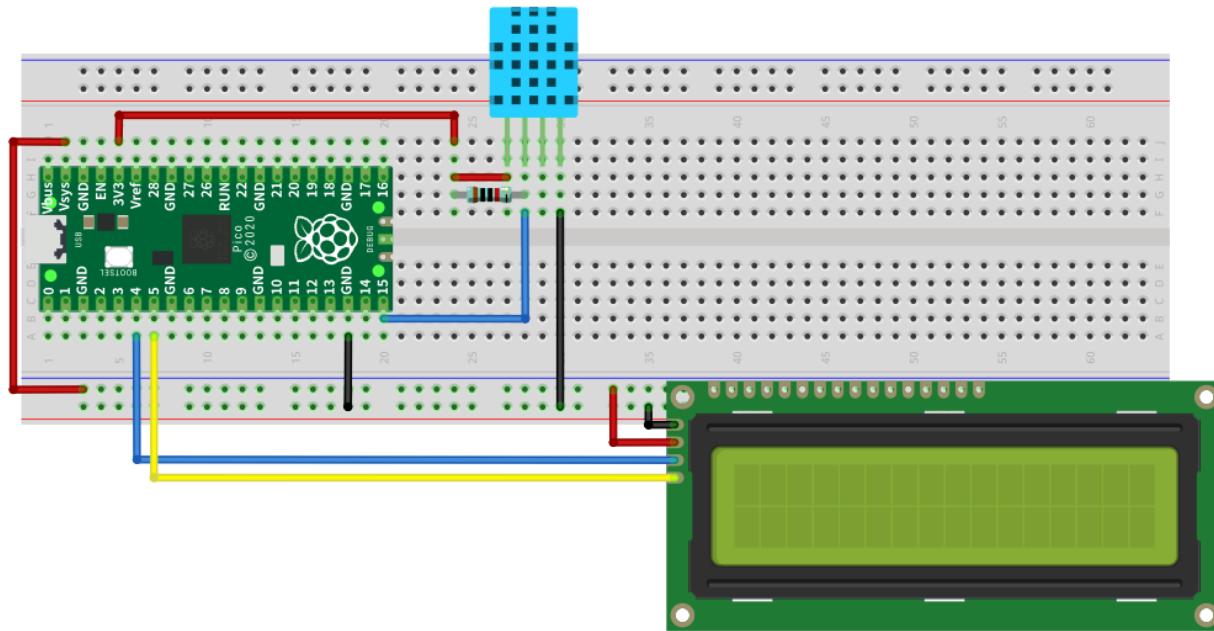


Circuit

Schematic diagram



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



Obtain data of Hygrothermograph every second and display them on LCD1602. The first line displays



Sketch

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

Sketch_25.2_Temperature_and_Humidity_Sensor

```

Sketch_25.2_Temperature_and_Humidity_Sensor | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_25.2_Temperature_and_Humidity_Sensor.ino
1 #include <LiquidCrystal_I2C.h>
2 #include <dht.h>
3
4 LiquidCrystal_I2C lcd(0x27, 16, 2); //initialize the LCD
5 int dhtPin = 15; // the number of the DHT11 sensor pin
6 dht DHT;
7
8 void setup() {
9     if (!i2CAddrTest(0x27)) {
10         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
11     }
12     lcd.init(); // LCD driver initialization
13     lcd.backlight(); // Open the backlight
14 }
15
16 void loop() {
17     // read DHT11 data and save it
18     int chk = DHT.read11(dhtPin);
19     if (chk == DHTLIB_OK) {
20         lcd.clear();
21         lcd.setCursor(0, 0); //set the cursor to column 0, line 1
22         lcd.print("Temp: "); //display the Humidity on the LCD1602
23         lcd.print(int(DHT.temperature));
24         lcd.print(" C");
25         lcd.setCursor(0, 1); //set the cursor to column 0, line 0
26         lcd.print("Humi: "); //display the Humidity on the LCD1602
27         lcd.print(int(DHT.humidity));
28         lcd.print(" %");
29     } else {
30
31
32
33
34
35
}

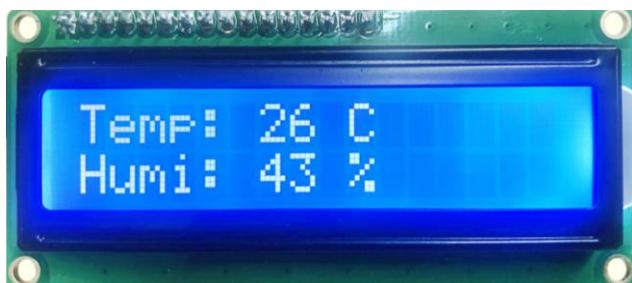
```

Output

indexing: 34/44

Ln 1, Col 1 Raspberry Pi Pico on COM15

Download the code to Pico. The first line of LCD1602 shows the temperature value, and the second line shows the humidity value. Try to “pinch” the DHT11(without touching the leads) with your index finger and thumb for a brief time to observe the change in the LCD display value.



The following is the program code:

```
1 #include <LiquidCrystal_I2C.h>
2 #include <dht.h>
3
4 LiquidCrystal_I2C lcd(0x27, 16, 2); //initialize the LCD
5 int dhtPin = 15; // the number of the DHT11 sensor pin
6 dht DHT;
7
8 void setup() {
9     if (!i2CAddrTest(0x27)) {
10         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
11     }
12     lcd.init(); // LCD driver initialization
13     lcd.backlight(); // Open the backlight
14 }
15
16 void loop() {
17     int chk = DHT.read11(dhtPin); // read DHT11 data and save it
18     if (chk == DHTLIB_OK) {
19         lcd.clear();
20         lcd.setCursor(0, 0); //set the cursor to column 0, line 1
21         lcd.print("Temp: "); //display the Humidity on the LCD1602
22         lcd.print(int(DHT.temperature));
23         lcd.print(" C");
24         lcd.setCursor(0, 1); //set the cursor to column 0, line 0
25         lcd.print("Humi: "); //display the Humidity on the LCD1602
26         lcd.print(int(DHT.humidity));
27         lcd.print(" %");
28     } else {
29         lcd.clear();
30         lcd.setCursor(0, 0); //set the cursor to column 0, line 1
31         lcd.print("DHT11 Data error");
32     }
33     delay(2000);
34 }
35
36 bool i2CAddrTest(uint8_t addr) {
37     Wire.begin();
38     Wire.beginTransmission(addr);
39     if (Wire.endTransmission() == 0) {
40         return true;
41     }
42     return false;
43 }
```



First, include the library function header file.

```
1 #include <LiquidCrystal_I2C.h>
2 #include <dht.h>
```

Initialize IIC-LCD1602 and turn ON the backlight.

```
9 if (!i2CAddrTest(0x27)) {
10     lcd = LiquidCrystal_I2C(0x3F, 16, 2);
11 }
12 lcd.init();           // LCD driver initialization
13 lcd.backlight();     // Open the backlight
```

Obtain the temperature and humidity data of the DHT11. If the data is obtained successfully, print it to the LCD1602 screen.

```
17 int chk = DHT.read11(dhtPin);
18 if (chk == DHTLIB_OK) {
19     lcd.clear();
20     lcd.setCursor(0, 0);          //set the cursor to column 0, line 1
21     lcd.print("Temp: ");        //display the Humidity on the LCD1602
22     lcd.print(int(DHT.temperature));
23     lcd.print(" C");
24     lcd.setCursor(0, 1);          //set the cursor to column 0, line 0
25     lcd.print("Humi: ");        //display the Humidity on the LCD1602
26     lcd.print(int(DHT.humidity));
27     lcd.print(" %");
28 }
```

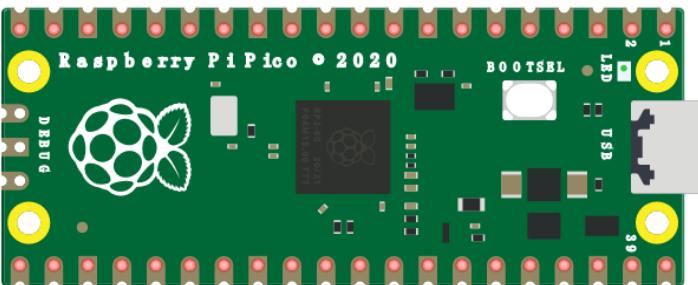
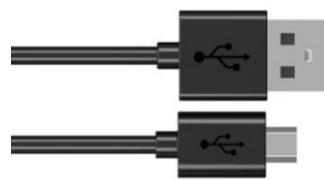
Chapter 26 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 26.1 Infrared Motion Detector with LED Indicator

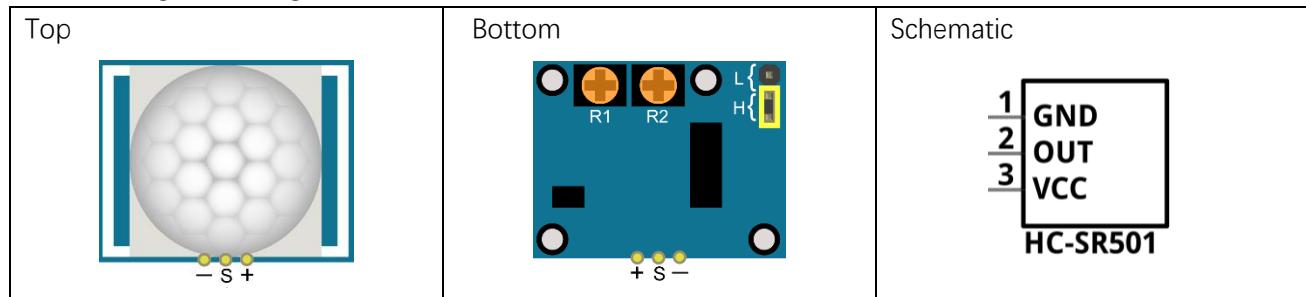
In this project, we will make a Motion Detector, with the human body infrared pyroelectric sensors. When someone is in close proximity to the Motion Detector, it will automatically light up and when there is no one close by, it will be out. This Infrared Motion Sensor can detect the infrared spectrum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

Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
HC SR501 x1	LED x1	Resistor 220Ω x1	Jumper

Component Knowledge

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



Description:

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

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

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

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

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

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

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

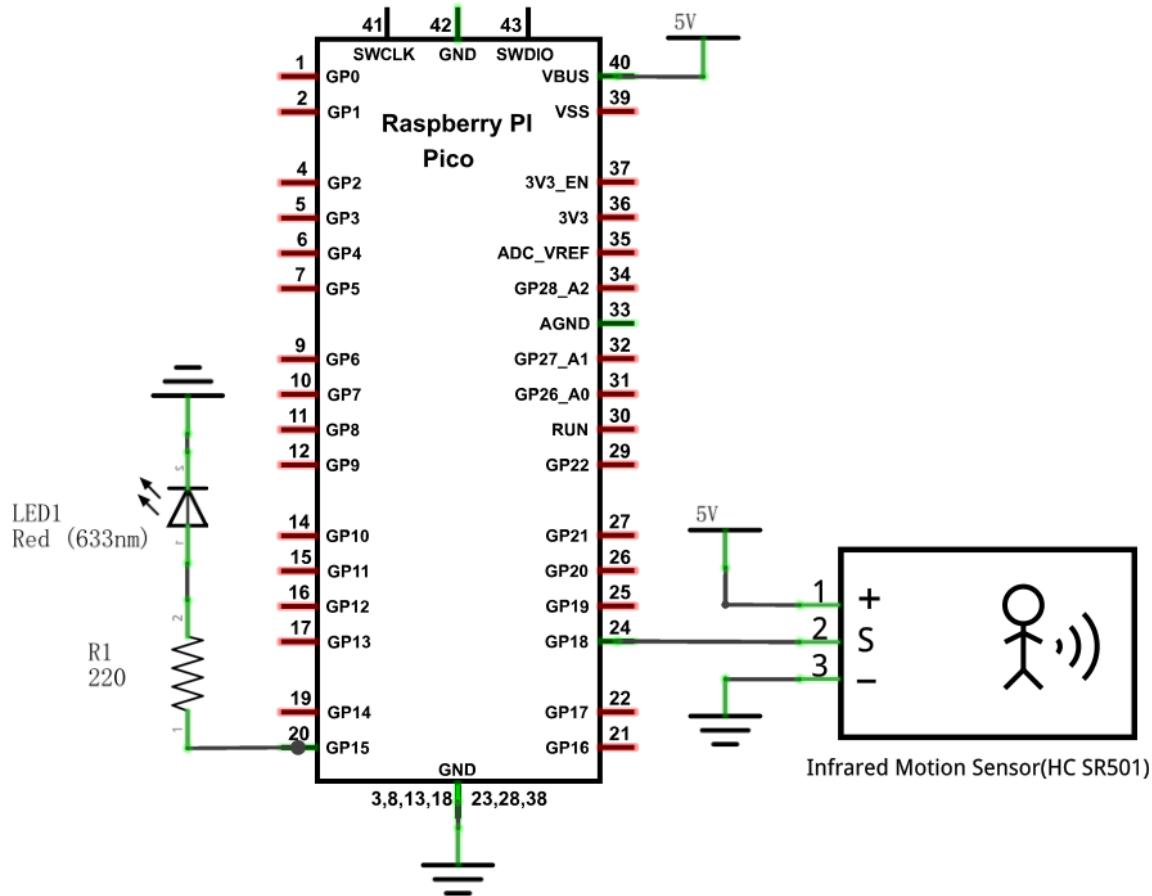
One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency).

Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

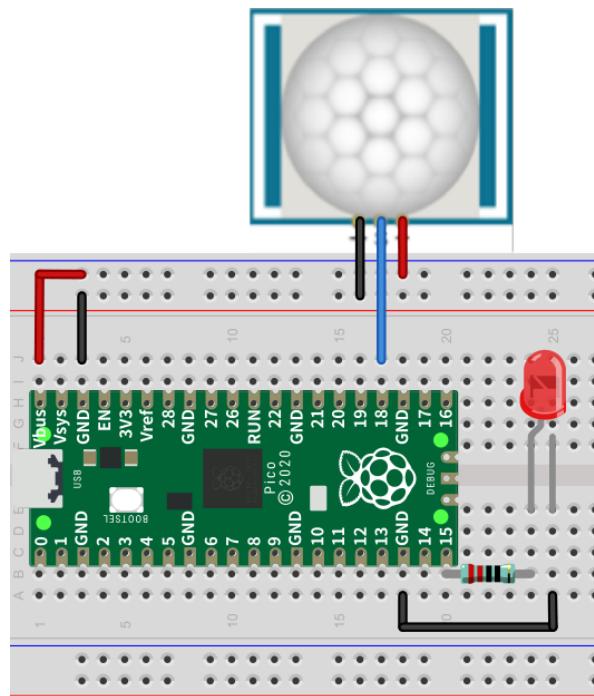
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



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



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



Sketch

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

Sketch_26.1_Infrared_Motion_Sensor

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

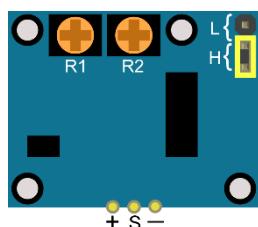
- Title Bar:** Sketch_26.1_Infrared_Motion_Sensor | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Upload, and Select Board (set to Raspberry Pi Pico).
- Sketch List:** Shows Sketch_26.1_Infrared_Motion_Sensor.ino
- Code Editor:**

```

7 int sensorPin = 18; // the number of the infrared motion sensor pin
8 int ledPin = 15;    // the number of the LED pin
9
10 void setup() {
11     Serial.begin(115200);
12     pinMode(sensorPin, INPUT); // initialize the sensor pin as input
13     pinMode(ledPin, OUTPUT);   // initialize the LED pin as output
14 }
15
16 void loop() {
17     // Turn on or off LED according to Infrared Motion Sensor
18     digitalWrite(ledPin, digitalRead(sensorPin));
19     delay(1000);           // wait for a second
20 }
```
- Output Panel:** Shows a black screen with the text "Ln 1, Col 1 Raspberry Pi Pico on COM15".

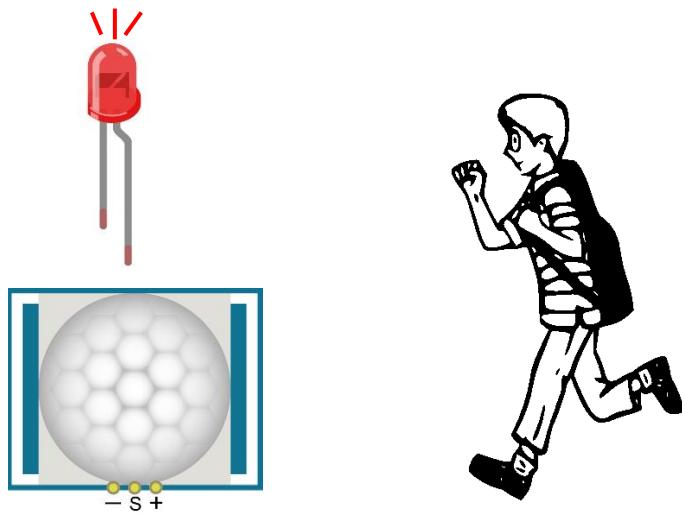
Verify and upload the code, and put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the infrared motion sensor and observe whether the LED turns ON or OFF automatically.

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



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

Move to the Infrared Motion Sensor



Move away from the Infrared Motion Sensor





Chapter 27 Attitude Sensor MPU6050

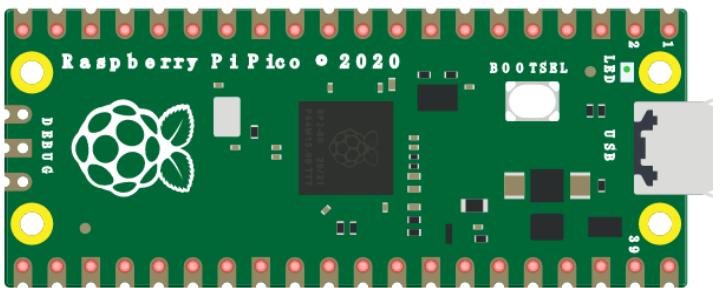
In this chapter, we will learn about an MPU6050 Attitude Sensor, which integrates an Accelerometer and Gyroscope.

Project 27.1 Read an MPU6050 Sensor Module

In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor.

Component List

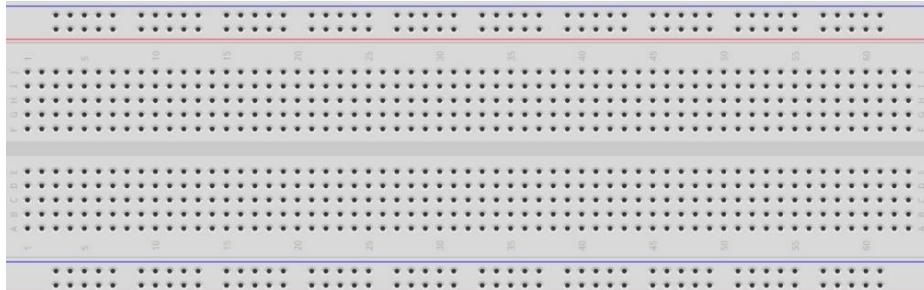
Raspberry Pi Pico x1



USB cable x1



Breadboard x1



Jumper



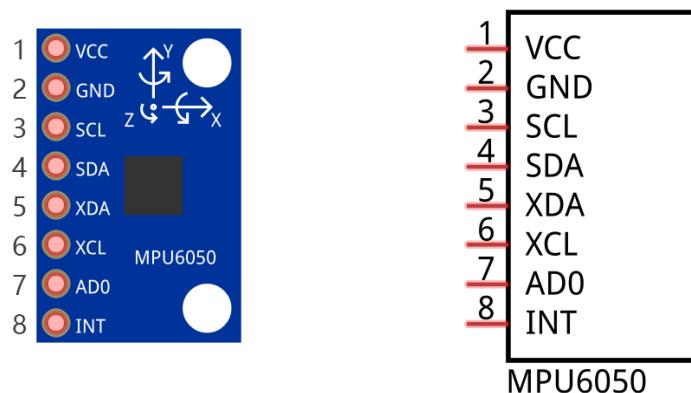
MPU6050 x1



Component Knowledge

MPU6050

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



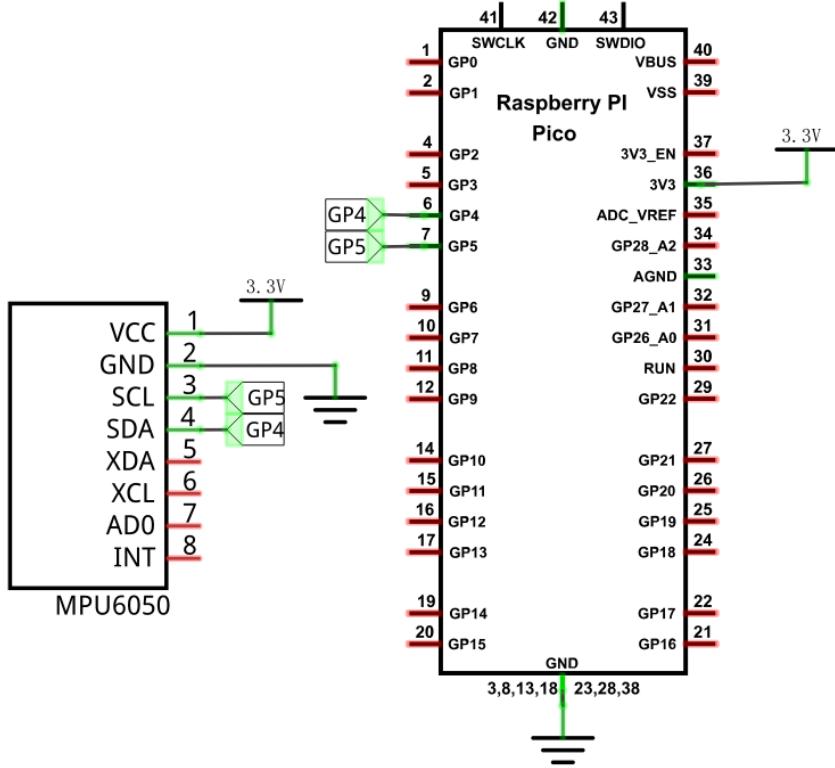
The port description of the MPU6050 module is as follows:

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

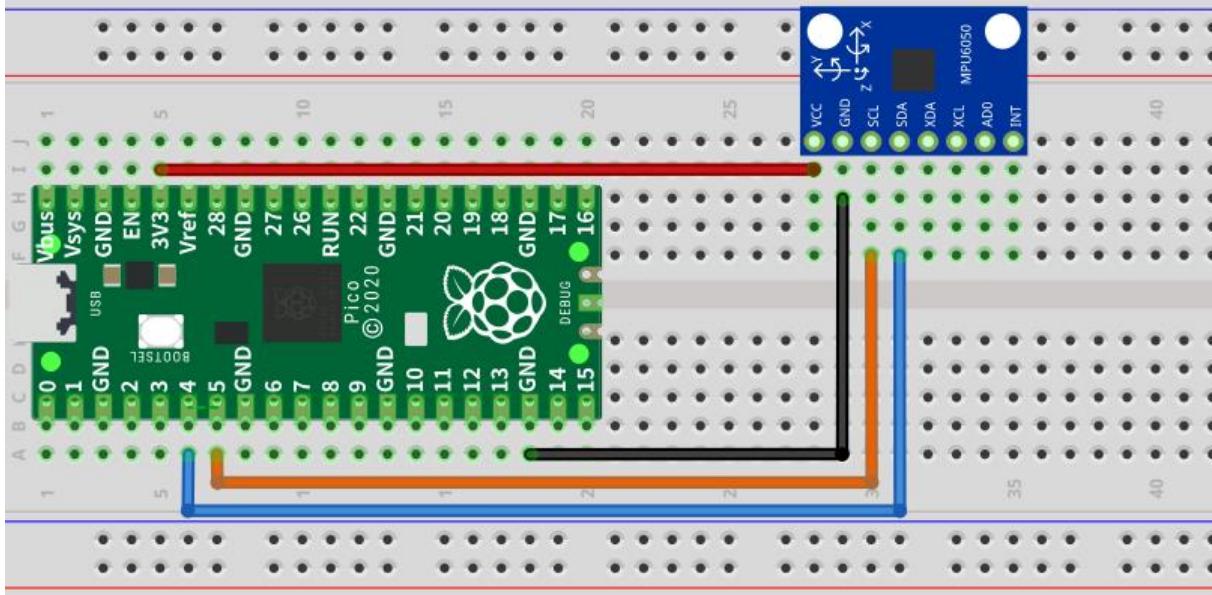
For more details, please refer to datasheet.

Circuit

Schematic diagram



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



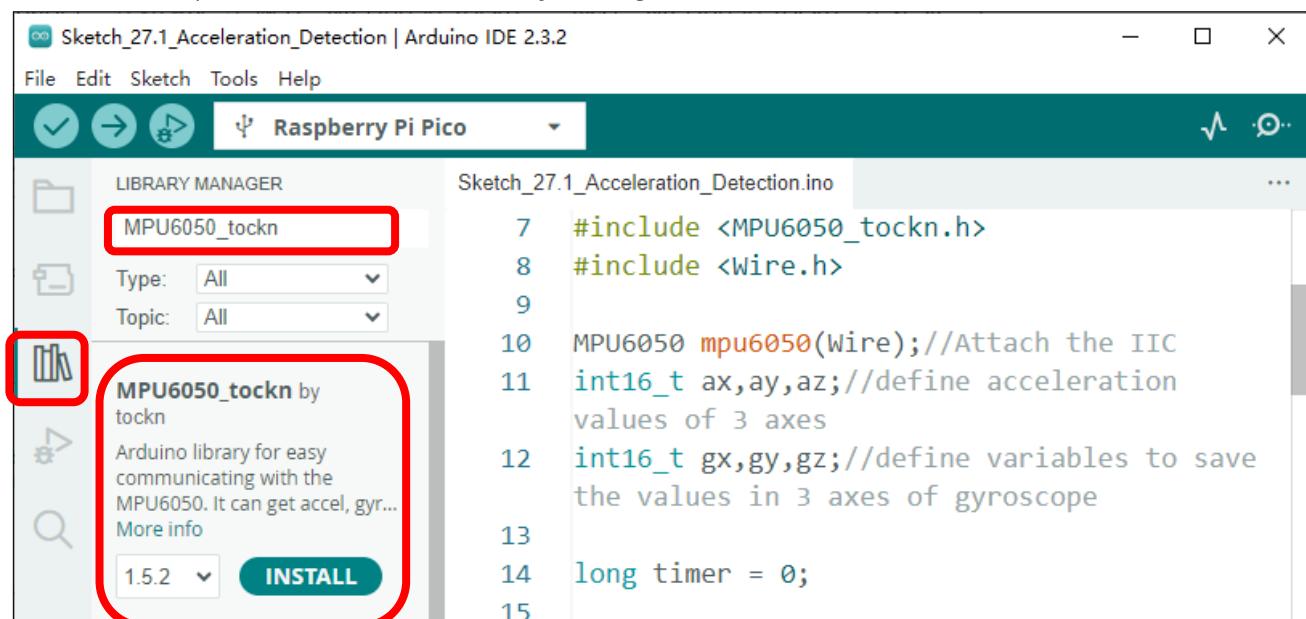
Sketch

How to install the library

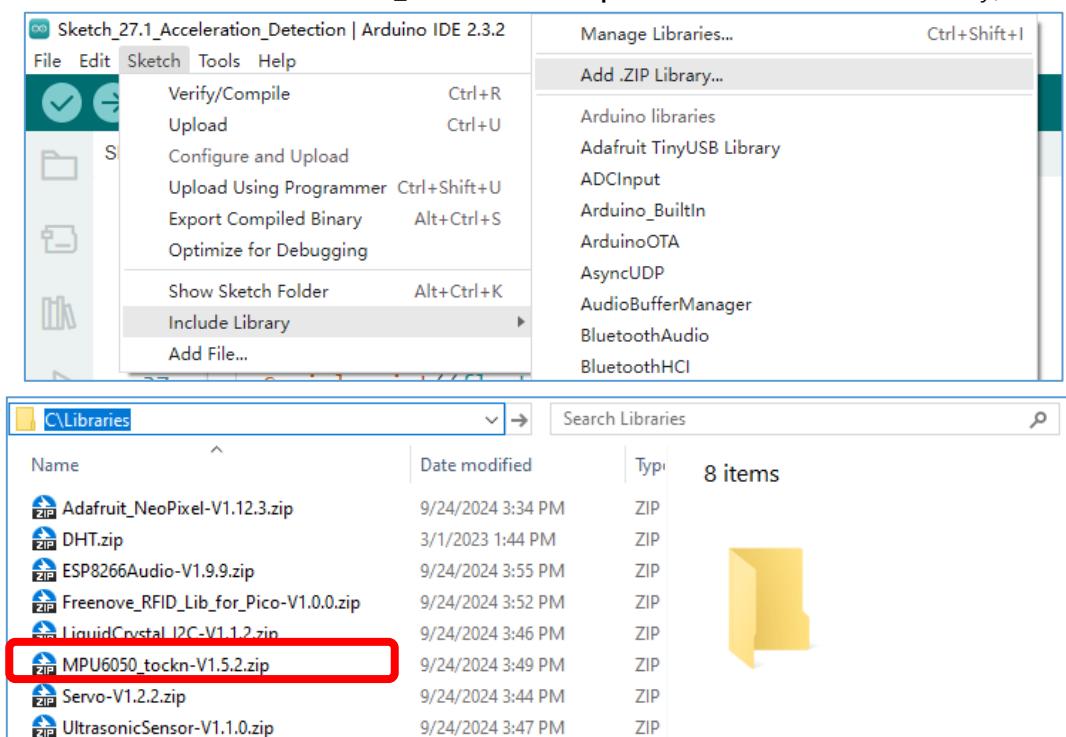
In this project, we will read the acceleration data and gyroscope data of MPU6050, and print them out.

We use a third-party library named **MPU6050_tockn**. Here are two ways to install the library for your reference.

The first one: Open Arduino IDE, click **Library Manager** on the left, search "**MPU6050_tockn**" to install.



The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named “./Libraries/**MPU6050_tockn-V1.5.2.Zip**” which locates in this directory, and click OPEN.



Sketch_27.1_Acceleration_Detection

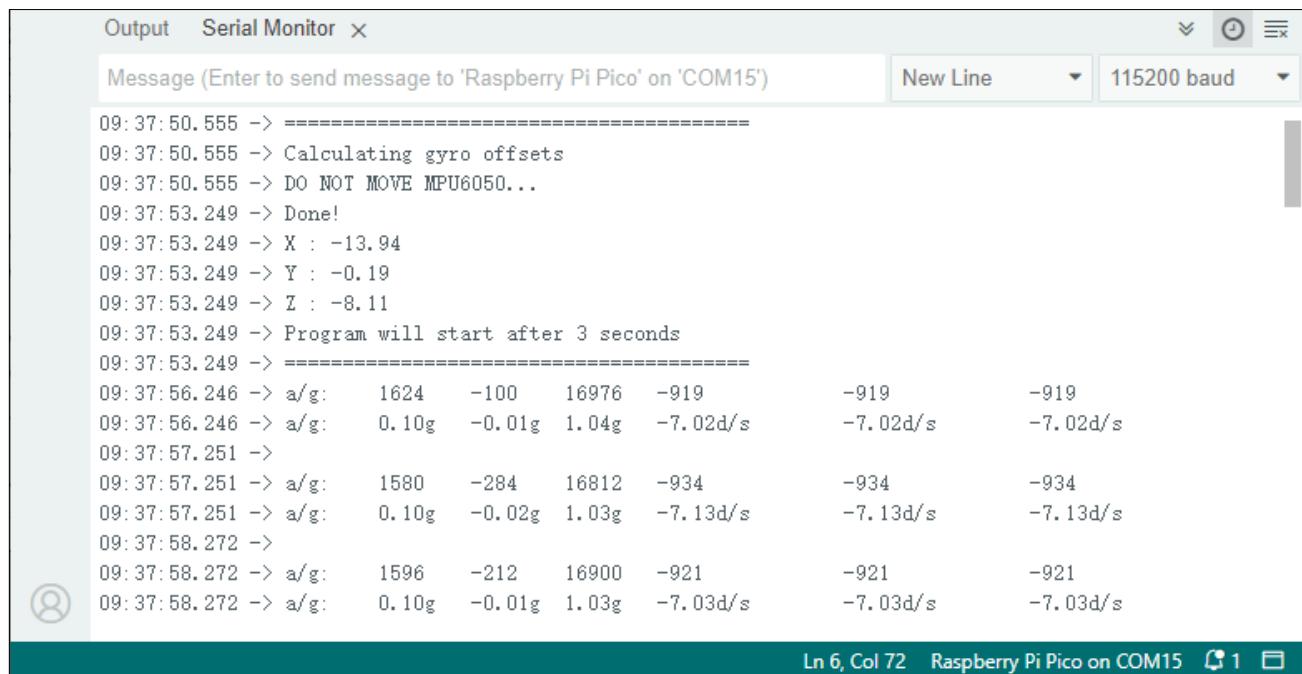
The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch_27.1_Acceleration_Detection | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Selection:** Raspberry Pi Pico
- Code Area:** Displays the C++ code for the sketch. The code reads data from an MPU6050 sensor and prints it to the Serial port. It includes a loop function that updates every second and a getMotion6 function that retrieves raw acceleration and gyroscope data.
- Output Area:** Shows the message "Ln 6, Col 72 Raspberry Pi Pico on COM15" indicating the connection status.

```
Sketch_27.1_Acceleration_Detection.ino

23 void loop() {
24     if(millis() - timer > 1000){ //each second printf the data
25         mpu6050.update(); //update the MPU6050
26         getMotion6(); //gain the values of Acceleration and
27         Gyroscope value
28         Serial.print("\na/g:\t");
29         Serial.print(ax); Serial.print("\t");
30         Serial.print/ay); Serial.print("\t");
31         Serial.print(az); Serial.print("\t");
32         Serial.print(gx); Serial.print("\t\t");
33         Serial.print(gy); Serial.print("\t\t");
34         Serial.println(gz);
35         Serial.print((float)ax / 16384); Serial.print("g\t");
36         Serial.print((float)ay / 16384); Serial.print("g\t");
37         Serial.print((float)az / 16384); Serial.print("g\t");
38         Serial.print((float)gx / 131); Serial.print("d/s \t");
39         Serial.print((float)gy / 131); Serial.print("d/s \t");
40         Serial.print((float)gz / 131); Serial.print("d/s \n");
41         timer = millis();
42     }
43 }
44 void getMotion6(void){
45     ax=mpu6050.getRawAccX(); //gain the values of X axis acceleration raw data
46     ay=mpu6050.getRawAccY(); //gain the values of Y axis acceleration raw data
47     az=mpu6050.getRawAccZ(); //gain the values of Z axis acceleration raw data
48     gx=mpu6050.getRawGyroX(); //gain the values of X axis Gyroscope raw data
49     gy=mpu6050.getRawGyroY(); //gain the values of Y axis Gyroscope raw data
50     gz=mpu6050.getRawGyroZ(); //gain the values of Z axis Gyroscope raw data
51 }
```

Download the code to Pico, open the serial monitor set the baud rate to 115200 and you can use it to measure the distance between the ultrasonic module and the object, as shown in the following picture:



The following is the program code:

```

1 #include <MPU6050_tockn.h>
2 #include <Wire.h>
3
4 MPU6050 mpu6050(Wire); //Attach the IIC
5 int16_t ax,ay,az;//define acceleration values of 3 axes
6 int16_t gx,gy,gz;//define variables to save the values in 3 axes of gyroscope
7
8 long timer = 0;
9
10 void setup() {
11     Wire.begin();
12     Serial.begin(115200);
13     mpu6050.begin();           //initialize the MPU6050
14     mpu6050.calcGyroOffsets(true); //get the offsets value
15 }
16
17 void loop() {
18     if(millis() - timer > 1000){ //each second printf the data
19         mpu6050.update();          //update the MPU6050
20         getMotion6();              //gain the values of Acceleration and Gyroscope value
21         Serial.print("\n a/g:\t");
22         Serial.print(ax); Serial.print("\t");
23         Serial.print(ay); Serial.print("\t");
24         Serial.print(az); Serial.print("\t");
}

```

```

25   Serial.print(gx); Serial.print("\t\t");
26   Serial.print(gy); Serial.print("\t\t");
27   Serial.println(gz);
28   Serial.print("a/g:\t");
29   Serial.print((float)ax / 16384); Serial.print("g\t");
30   Serial.print((float)ay / 16384); Serial.print("g\t");
31   Serial.print((float)az / 16384); Serial.print("g\t");
32   Serial.print((float)gx / 131); Serial.print("d/s \t");
33   Serial.print((float)gy / 131); Serial.print("d/s \t");
34   Serial.print((float)gz / 131); Serial.print("d/s \n");
35   timer = millis();
36 }
37 }
38 void getMotion6(void) {
39   ax=mpu6050.getRawAccX(); //gain the values of X axis acceleration raw data
40   ay=mpu6050.getRawAccY(); //gain the values of Y axis acceleration raw data
41   az=mpu6050.getRawAccZ(); //gain the values of Z axis acceleration raw data
42   gx=mpu6050.getRawGyroX(); //gain the values of X axis Gyroscope raw data
43   gy=mpu6050.getRawGyroY(); //gain the values of Y axis Gyroscope raw data
44   gz=mpu6050.getRawGyroZ(); //gain the values of Z axis Gyroscope raw data
45 }
```

Two library files "**MPU6050_tockn.h**" and "**Wire.h**" are used in the code and will be compiled with others. Class **MPU6050** is used to operate the **MPU6050**. When using it, please instantiate an object first.

4	<code>MPU6050 mpu6050(Wire); //Attach the IIC</code>
---	--

In the setup function, IIC and **MPU6050** are initialized and the offset difference of **MPU6050** is obtained.

10	<code>void setup() {</code>
11	<code> Wire.begin();</code>
12	<code> Serial.begin(115200);</code>
13	<code> mpu6050.begin(); //initialize the MPU6050</code>
14	<code> mpu6050.calcGyroOffsets(true); //get the offsets value</code>
15	<code>}</code>

The **getMotion6** function is used to obtain the x, y, z axis acceleration raw data and the Gyroscope raw data.

38	<code>void getMotion6(void){</code>
39	<code> ax=mpu6050.getRawAccX(); //gain the values of X axis acceleration raw data</code>
40	<code> ay=mpu6050.getRawAccY(); //gain the values of Y axis acceleration raw data</code>
41	<code> az=mpu6050.getRawAccZ(); //gain the values of Z axis acceleration raw data</code>
42	<code> gx=mpu6050.getRawGyroX(); //gain the values of X axis Gyroscope raw data</code>
43	<code> gy=mpu6050.getRawGyroY(); //gain the values of Y axis Gyroscope raw data</code>
44	<code> gz=mpu6050.getRawGyroZ(); //gain the values of Z axis Gyroscope raw data</code>
45	<code>}</code>

Finally, the original data of the gyroscope is updated and acquired every second, and the original data, the processed acceleration and angular velocity data are printed out through the serial port.

19	<code>void loop() {</code>
20	<code> if(millis() - timer > 1000){ //each second print the data</code>

Any concerns? ↗ support@freenove.com

```

21   mpu6050.update();           //update the MPU6050
22   getMotion6();              //gain the values of Acceleration and Gyroscope value
23   Serial.print("\n/a/g:\t");
24   Serial.print(ax); Serial.print("\t");
25   Serial.print(ay); Serial.print("\t");cc
26   Serial.print(az); Serial.print("\t");
27   Serial.print(gx); Serial.print("\t\t");
28   Serial.print(gy); Serial.print("\t\t");
29   Serial.println(gz);
30   Serial.print("a/g:\t");
31   Serial.print((float)ax / 16384); Serial.print("g\t");
32   Serial.print((float)ay / 16384); Serial.print("g\t");
33   Serial.print((float)az / 16384); Serial.print("g\t");
34   Serial.print((float)gx / 131); Serial.print("d/s \t");
35   Serial.print((float)gy / 131); Serial.print("d/s \t");
36   Serial.print((float)gz / 131); Serial.print("d/s \n");
37   timer = millis();
38 }
39 }
```

Reference

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

MPU6050 mpu6050 (Wire): Associate MPU6050 with IIC.

begin(): Initialize the MPU6050.

calcGyroOffsets (true): If the parameter is true, get the gyro offset and automatically correct the offset. If the parameter is false, the offset value is not obtained and the offset is not corrected.

getRawAccX(): Gain the values of X axis acceleration raw data.

getRawAccY(): Gain the values of Y axis acceleration raw data.

getRawAccZ(): Gain the values of Z axis acceleration raw data.

getRawGyroX(): Gain the values of X axis Gyroscope raw data.

getRawGyroY(): Gain the values of Y axis Gyroscope raw data.

getRawGyroZ(): Gain the values of Z axis Gyroscope raw data.

getTemp(): Gain the values of MPU6050's temperature data.

update(): Update the MPU6050. If the updated function is not used, the IIC will not be able to retrieve the new data.

Chapter 28 RFID

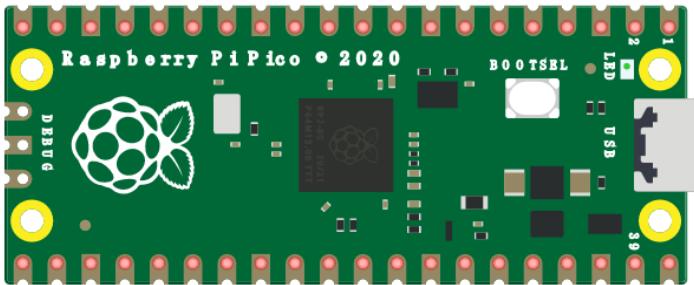
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 28.1 RFID read UID

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

Component List

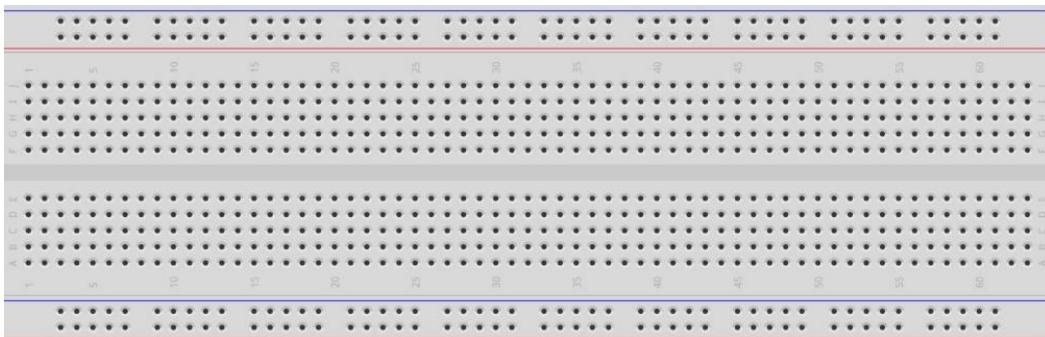
Raspberry Pi Pico x1



USB cable x1



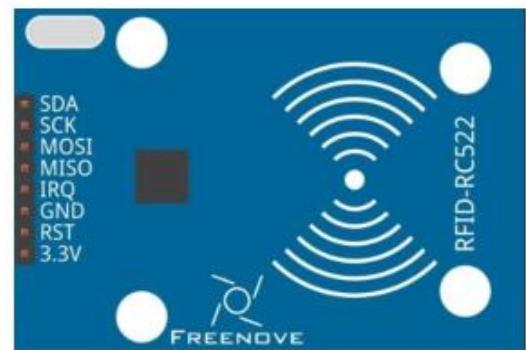
Breadboard x1

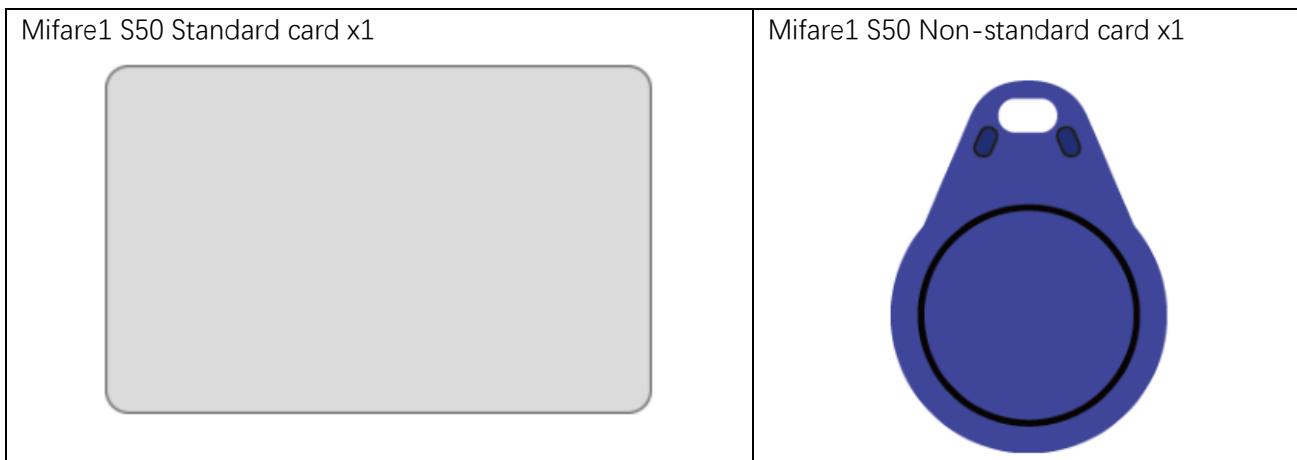


Jumper



RFID Module(RC522) x1





Component Knowledge

RFID

RFID (Radio Frequency Identification) is a wireless communication technology. A complete RFID system is generally composed of the responder and reader. Generally, we use tags as responders, and each tag has a unique code, which is attached to the object to identify the target object. The reader is a device for reading (or writing) tag information.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products. Moreover, Passive RFID products are the earliest, the most mature and most widely used products in the market among others. It can be seen everywhere in our daily life such as, the bus card, dining card, bankcard, hotel access cards, etc., and all of these belong to close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

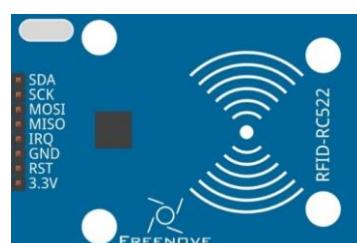
The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522 RFID Module

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.

This RFID Module uses MFRC522 as the control chip and SPI (Peripheral Interface Serial) as the reserved interface.



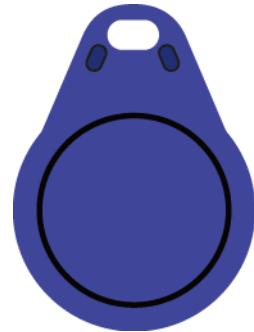
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Mifare1 S50 Card

Mifare1 S50 is often called Mifare Standard with the capacity of 1K bytes. Each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for this kit are shown below.



Mifare1 S50 Standard card



Mifare1 S50 Non-standard card

The Mifare1 S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains four data block (Block0-Block3). 64 blocks of 16 sectors will be numbered according to absolute address, from 0 to 63). Each block contains 16 bytes (Byte0-Byte15), $64 \times 16 = 1024$. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
.....
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control that are put in the last block of each sector, and the block is also known as sector trailer, that is Block 3 in each sector. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the vendor code, which has been solidified and cannot be changed, and the card serial number is stored here. In addition to the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications: (1) used as general data storage and can be operated for reading and writing. (2) used as data value, and can be operated for initializing the value, adding value, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, 4-byte access control and 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally A0A1A2A3A4A5 for password A, B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

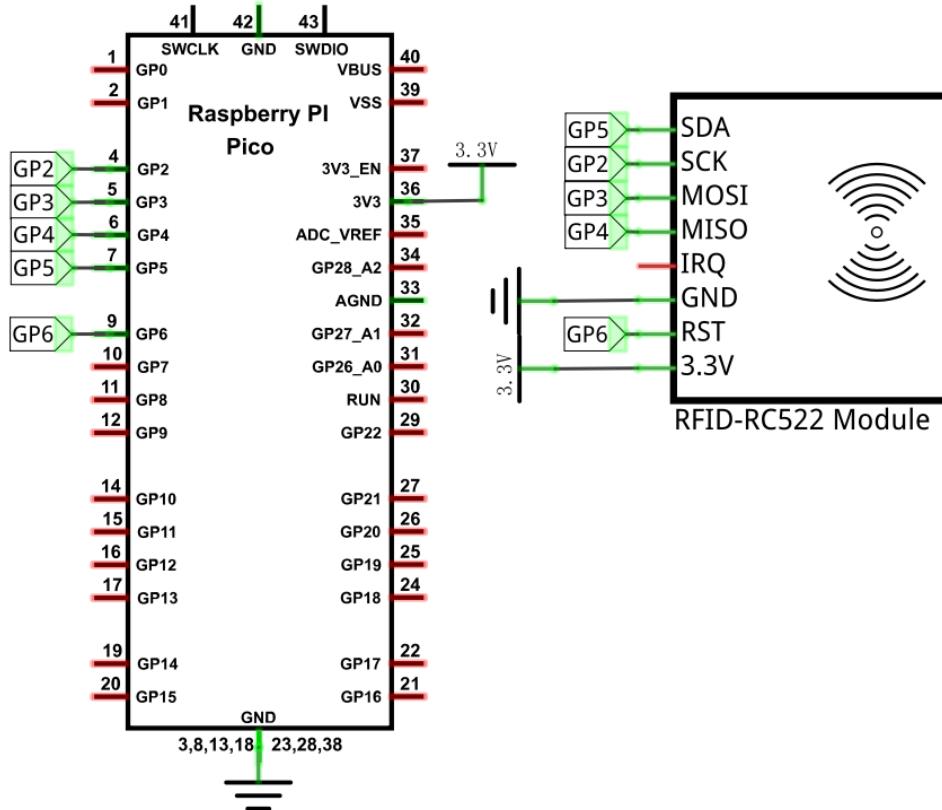
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. In addition, after verifying password A, we can do reading or writing operation to control blocks. However, password A can never be read. If you choose to verify password A and then you forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

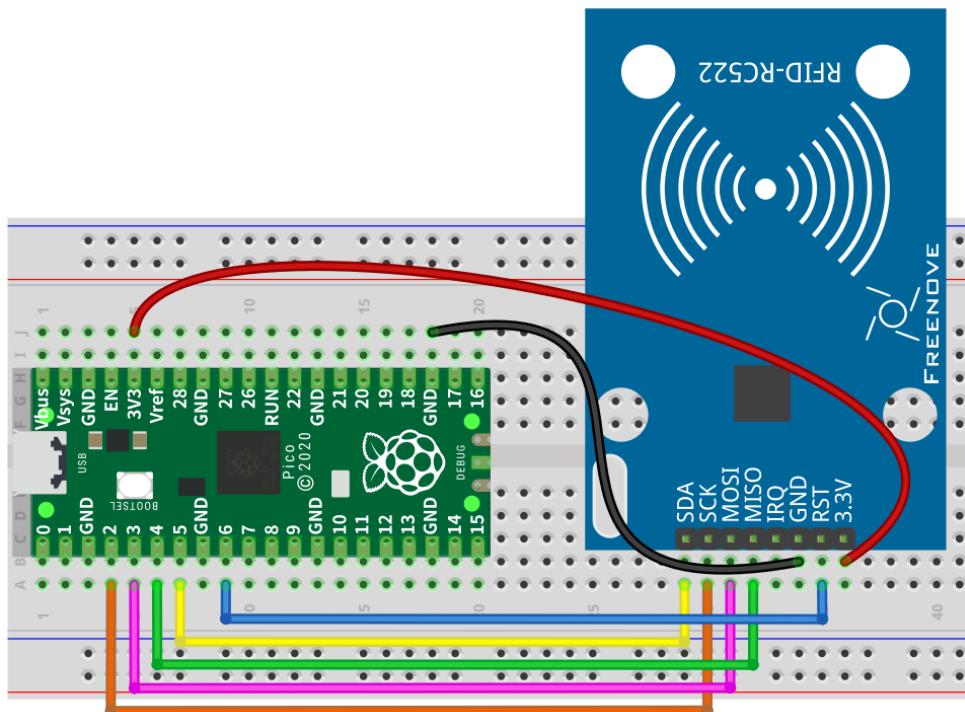
Circuit

The connection of control board and RFID module is shown below.

Schematic diagram



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



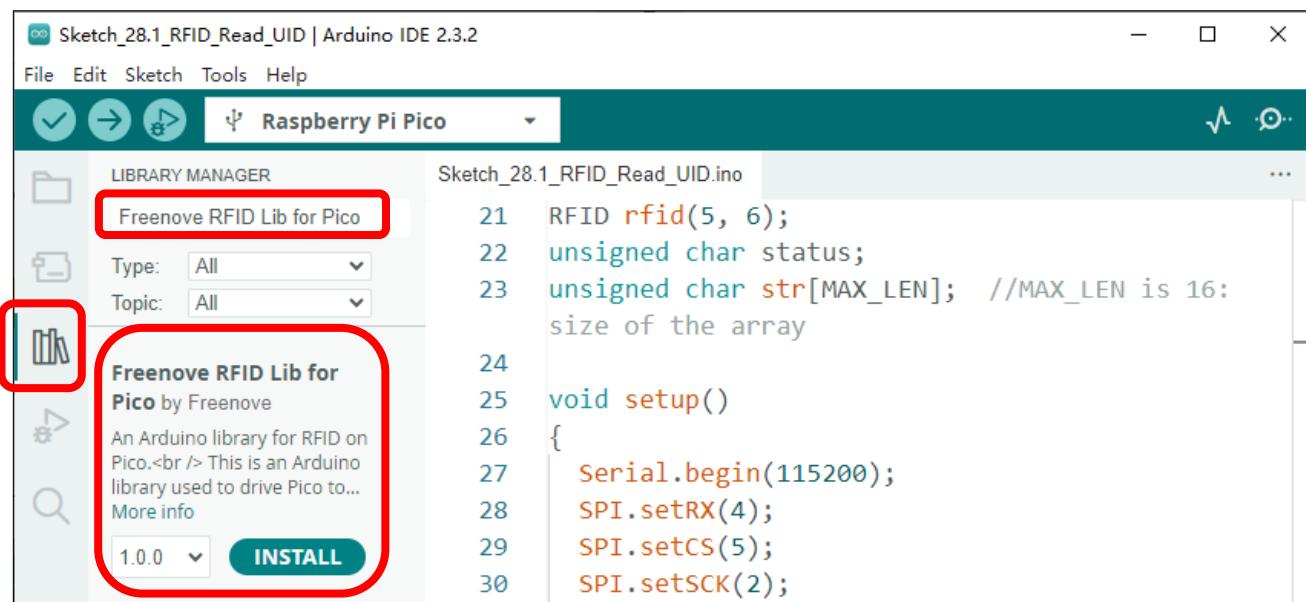
Any concerns? ✉ support@freenove.com

Sketch

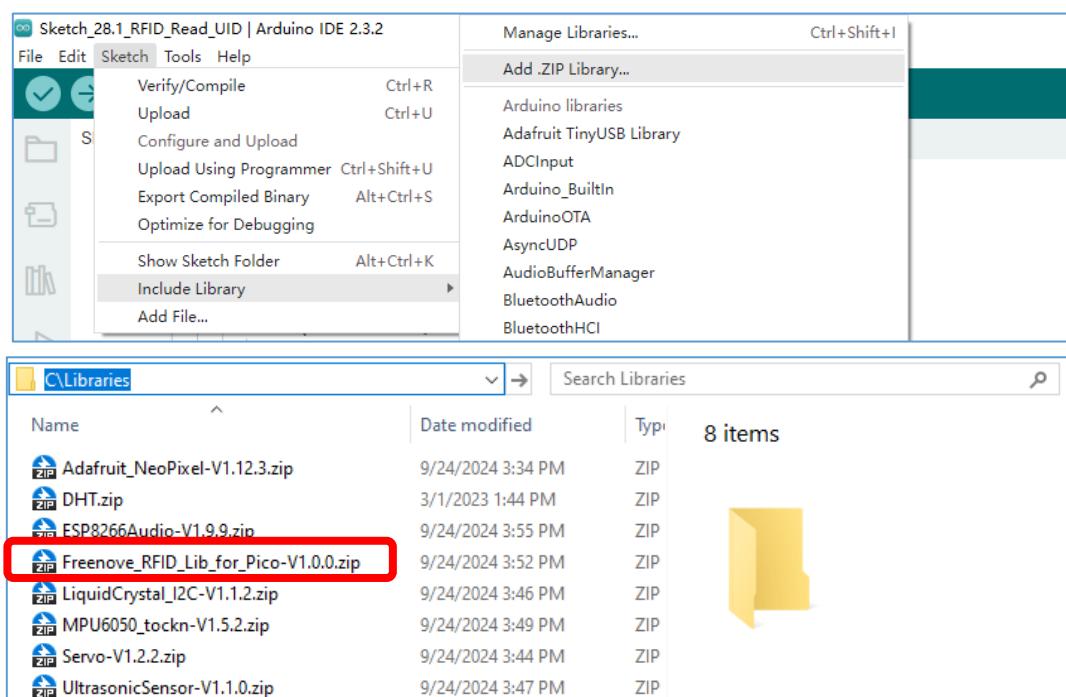
Sketch_28.1_RFID_Read_UID

In this project, we use a third-party library named **Freenove RFID Lib for Pico**. Here are two ways to install the library for your reference.

The first one: Open Arduino IDE, click **Library Manager** on the left, search "Freenove RFID Lib for Pico" to install.



The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named ".Libraries/ Freenove_RFID_Lib_for_Pico-V1.0.0.Zip" which locates in this directory, and click OPEN.





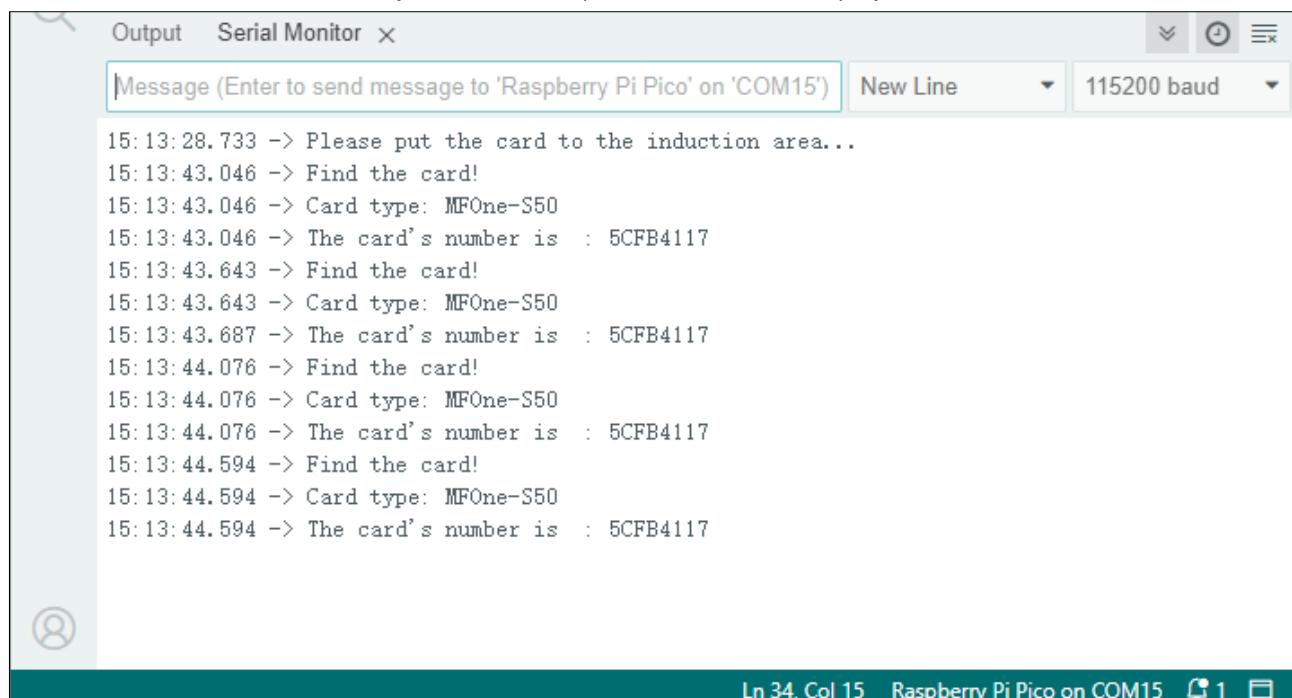
This sketch will read the unique ID number (UID) of the card, recognize the type of the card and display the information through serial port.

```
1 #include <SPI.h>
2 #include <RFID.h>
3
4 RFID rfid(5, 6);
5 unsigned char status;
6 unsigned char str[MAX_LEN]; //MAX_LEN is 16: size of the array
7
8 void setup()
9 {
10     Serial.begin(115200);
11     SPI.setRX(4);
12     SPI.setCS(5);
13     SPI.setSCK(2);
14     SPI.setTX(3);
15     SPI.begin();
16     rfid.init(); //initialization
17     Serial.println("Please put the card to the induction area...");
18 }
19
20 void loop()
21 {
22     //Search card, return card types
23     if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
24         Serial.println("Find the card!");
25         // Show card type
26         ShowCardType(str);
27         //Anti-collision detection, read card serial number
28         if (rfid.anticoll(str) == MI_OK) {
29             Serial.print("The card's number is : ");
30             //Display card serial number
31             for (int i = 0; i < 4; i++) {
32                 Serial.print(0x0F & (str[i] >> 4), HEX);
33                 Serial.print(0x0F & str[i], HEX);
34             }
35             Serial.println("");
36         }
37         //card selection (lock card to prevent redundant read, removing the line will make
38         //the sketch read cards continually)
39         rfid.selectTag(str);
40     }
41     rfid.halt(); // command the card to enter sleeping state
42 }
```

```

42 void ShowCardType(unsigned char * type)
43 {
44     Serial.print("Card type: ");
45     if (type[0] == 0x04 && type[1] == 0x00)
46         Serial.println("MFOne-S50");
47     else if (type[0] == 0x02 && type[1] == 0x00)
48         Serial.println("MFOne-S70");
49     else if (type[0] == 0x44 && type[1] == 0x00)
50         Serial.println("MF-UltraLight");
51     else if (type[0] == 0x08 && type[1] == 0x00)
52         Serial.println("MF-Pro");
53     else if (type[0] == 0x44 && type[1] == 0x03)
54         Serial.println("MF Desire");
55     else
56         Serial.println("Unknown");
57 }
```

After verifying and uploading the code, open the serial monitor and make a card approach the sensing area of RFID module. Then serial monitor will display the displacement ID number and the type of the card. If the induction time is too short, it may lead to incomplete-information display.



After including the RFID library, we need to construct a RFID class object before using the function in RFID library. Its constructor needs to be written to two pins, respectively to the SDA pin and the RST pin.

4	RFID rfid(5, 6);
---	------------------

In setup, initialize the serial port, SPI and RFID.

10	Serial.begin(115200);
11	SPI.setRX(4);
12	SPI.setCS(5);
13	SPI.setSCK(2);

```
14   SPI.setTX(3);  
15   SPI.begin();  
16   rfid.init(); //initialization
```

In loop(), use findCard() waiting for the card approaching. Once it detects card contact, this function will return MI_OK and save the card type data in parameter str, and then enter the "if" statement.

```
23   if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
```

After entering if statement, call the sub function ShowCardType(). Then determine the type of the card according to the content of STR and print the type out through the serial port.

```
26   ShowCardType(str);
```

Then use the.anticoll() to read UID of the card and use serial port to print it out.

```
28   if (rfid.anticoll(str) == MI_OK) {  
29       Serial.print("The card's number is : ");  
30       //Display card serial number  
31       for (int i = 0; i < 4; i++) {  
32           Serial.print(0x0F & (str[i] >> 4), HEX);  
33           Serial.print(0x0F & str[i], HEX);  
34       }  
35       Serial.println("");  
36   }
```

Project 28.2 Read and write

In this project, we will do reading and writing operations to the card.

Component list

Same with last section.

Circuit

Same with last section.

Sketch

Sketch_28.2_RFID_Read_And_Write

In this sketch, first read the data in particular location of the S50 M1 Card, then write data in that position and read it out. Display these data through the serial port.

```
1 #include <SPI.h>
2 #include <RFID.h>
3
4 //pin5:pin of card reader SDA. pin6:pin of card reader RST
5 RFID rfid(5, 6);
6
7 // 4-byte card serial number, the fifth byte is check byte
8 unsigned char serNum[5];
9 unsigned char status;
10 unsigned char str[MAX_LEN];
11 unsigned char blockAddr;           //Select the operation block address: 0 to 63
12
13 // Write card data you want(within 16 bytes)
14 unsigned char writeDate[16] = "WelcomeFreenove";
15
16 // The A password of original sector: 16 sectors; the length of password in each sector
17 // is 6 bytes.
17 unsigned char sectorKeyA[16][16] = {
18     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
19     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
20     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
21     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
22     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
23     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
```

```
24 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
25 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
26 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
27 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
28 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
29 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
30 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
31 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
32 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
33 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
34 };  
35  
36 void setup()  
37 {  
38     Serial.begin(115200);  
39     SPI.setRX(4);  
40     SPI.setCS(5);  
41     SPI.setSCK(2);  
42     SPI.setTX(3);  
43     SPI.begin();  
44     rfid.init();  
45     Serial.println("Please put the card to the induction area...");  
46 }  
47  
48 void loop()  
49 {  
50     //find the card  
51     rfid.findCard(PICC_REQIDL, str);  
52     //Anti-collision detection, read serial number of card  
53     if (rfid.anticoll(str) == MI_OK) {  
54         Serial.print("The card's number is : ");  
55         //print the card serial number  
56         for (int i = 0; i < 4; i++) {  
57             Serial.print(0x0F & (str[i] >> 4), HEX);  
58             Serial.print(0x0F & str[i], HEX);  
59         }  
60         Serial.println("");  
61         memcpy(rfid.serNum, str, 5);  
62     }  
63     //select card and return card capacity (lock the card to prevent multiple read and  
written)  
64     rfid.selectTag(rfid.serNum);  
65     //first, read the data of data block 4  
readCard(4);
```

```
67 //write data(within 16 bytes) to data block
68 writeCard(4);
69 //then read the data of data block again
70 readCard(4);
71
72 rfid.halt();
73 }
74
75 //write the card
76 void writeCard(int blockAddr) {
77     if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
78         MI_OK) //authenticate
79     {
80         //write data
81         //status = rfid.write((blockAddr/4 + 3*(blockAddr/4+1)), sectorKeyA[0]);
82         Serial.print("set the new card password, and can modify the data of the Sector: ");
83         Serial.println(blockAddr / 4, DEC);
84         // select block of the sector to write data
85         if (rfid.write(blockAddr, writeDate) == MI_OK) {
86             Serial.println("Write card OK!");
87         }
88     }
89
90 //read the card
91 void readCard(int blockAddr) {
92     if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
93         MI_OK) // authenticate
94     {
95         // select a block of the sector to read its data
96         Serial.print("Read from the blockAddr of the card : ");
97         Serial.println(blockAddr, DEC);
98         if (rfid.read(blockAddr, str) == MI_OK) {
99             Serial.print("The data is (char type display): ");
100            Serial.println((char *)str);
101            Serial.print("The data is (HEX type display): ");
102            for (int i = 0; i < sizeof(str); i++) {
103                Serial.print(str[i], HEX);
104                Serial.print(" ");
105            }
106        }
107    }
108 }
```



In the sub function of writeCard() and readCard(), we must first verify the password A, and then use the corresponding sub function to read and write. Here we do reading and writing operations to data block 0 (absolute NO.4) of the first sector.

```
77 if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==  
    MI_OK) //authenticate
```

In loop(), compare the contents of the data block NO.4 after written to the original contents.

```
65 //first, read the data of data block 4  
66 readCard(4);  
67 //write data(within 16 bytes) to data block  
68 writeCard(4);  
69 //then read the data of data block again  
70 readCard(4);
```

After verifying and uploading the code, open the serial port monitor and make a card approach the sensing area of RFID module, then the serial port monitoring window will display displacement ID numbers of the card, the type of this card and the contents (before and after writing operation) of data block. If the induction time is too short, it may lead to incomplete information display.

The screenshot shows the Arduino Serial Monitor interface. The top bar includes tabs for 'Output' and 'Serial Monitor' (which is active), along with settings for 'New Line' and '115200 baud'. The main window displays the following log output:

```
15:17:20.402 -> The card's number is : 5CFB4117
15:17:20.407 -> Read from the blockAddr of the card : 4
15:17:20.407 -> The data is (char type display): WelcomeFreenove
15:17:20.407 -> The data is (HEX type display): 57 65 6C 63 6F 6D 65 46 72 65 65 6E 6F 76 65 0
15:17:20.407 -> set the new card password, and can modify the data of the Sector: 1
15:17:20.407 -> Write card OK!
15:17:20.407 -> Read from the blockAddr of the card : 4
15:17:20.407 -> The data is (char type display): WelcomeFreenove
15:17:20.407 -> The data is (HEX type display): 57 65 6C 63 6F 6D 65 46 72 65 65 6E 6F 76 65 0
```

The bottom status bar indicates 'Ln 9, Col 1' and 'Raspberry Pi Pico on COM15'.

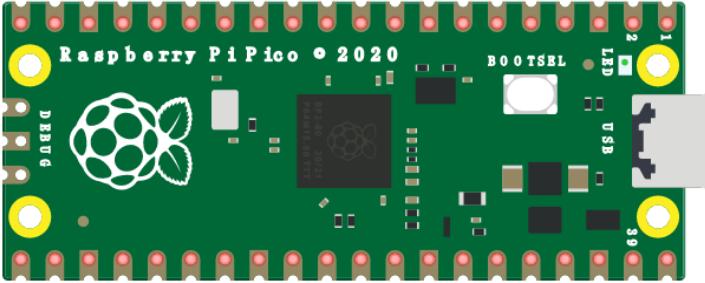
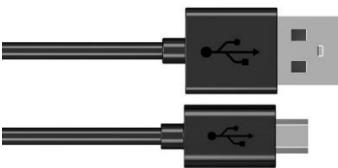
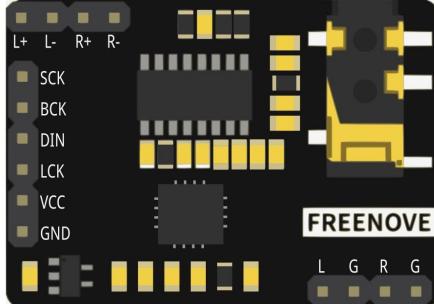
Chapter 29 Play Music

In this chapter, we use Raspberry Pi Pico and an IIS audio decoding module to play music.

This chapter is not applicable to FNK0058A/B. If your kit is one of these two models, please skip this chapter.

Project 29.1 Play Music by Audio Converter & Amplifier

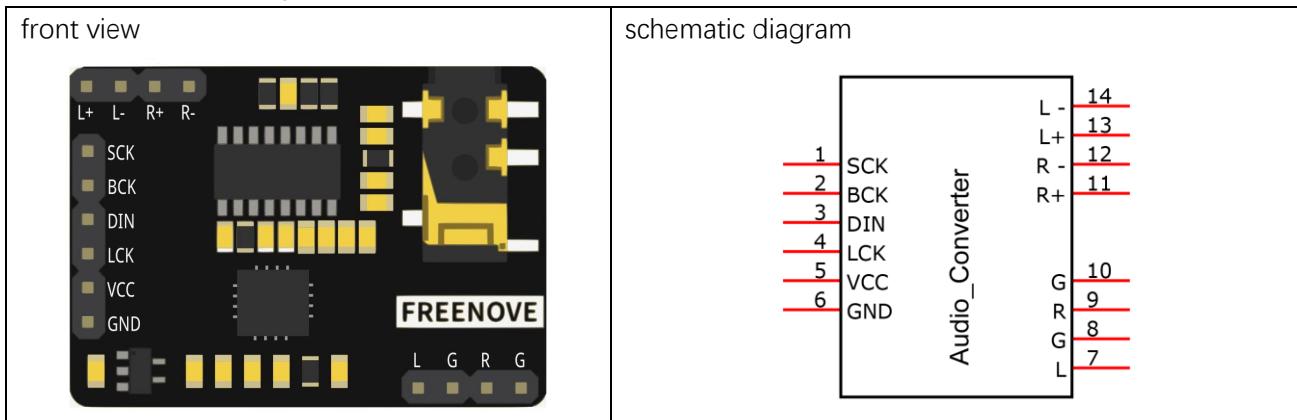
Component List

Raspberry Pi Pico x1	 A green printed circuit board (PCB) with a central Broadcom SoC chip. It features a USB-C port at the top, a microSD card slot, and various component pads along the edges. The text "Raspberry Pi Pico • 2020" is printed on the board.	Micro USB Wire x1	 Two black Micro USB cables, each with a standard A-type connector at one end and a Micro-B connector at the other.
Audio Converter & Amplifier	 A black PCB module with a central IC and various pins labeled with connection names: L+, L-, R+, R-, SCK, BCK, DIN, LCK, VCC, GND, and L, G, R, G.	Speaker*1	 A circular speaker icon with a central hole and two wires extending from it, one red and one black.
Jumper wire F/M*5			 A single green jumper wire with two black plastic crimp connectors at the ends.



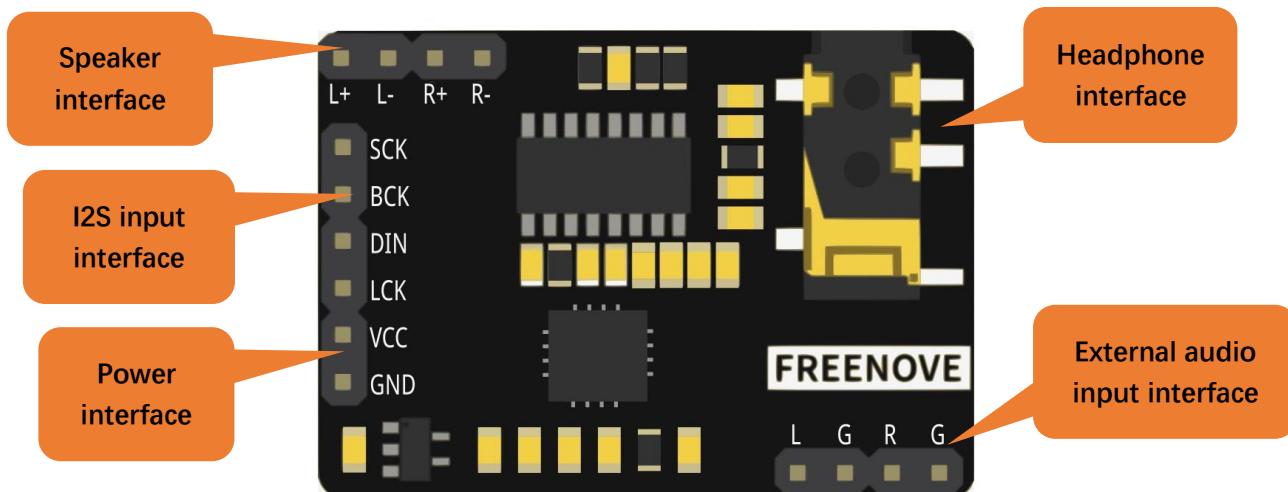
Component knowledge

Audio Converter & Amplifier module



Interface description for Audio Converter & Amplifier module

Pin	Name	Introductions
1	SCK	System clock input
2	BCK	Audio data bit clock input
3	DIN	Audio data input
4	LCK	Audio data word clock input
5	VCC	Power input, 3.3V~5.0V
6	GND	Power Ground
7	L	External audio left channel input
8	G	Power Ground
9	R	External audio right channel input
10	G	Power Ground
11	R+	Positive pole of right channel horn
12	R-	Negative pole of right channel horn
13	L+	Positive pole of left channel horn
14	L-	Negative pole of left channel horn



Any concerns? ✉ support@freenove.com

Speaker interface: Connect left channel speaker and right channel speaker. Group L: L+ & L-; Group R: R+ & R-. The two interfaces of the speaker can be connected to the interfaces of group L or group R. However, when one interface is connected to group L, the other cannot be connected to group R. Doing so may cause the module to malfunction.

Headphone interface: the interface to connect the headphones.

I2S input interface: connect to the device with I2S. Used to transcode audio data into DAC audio signals.

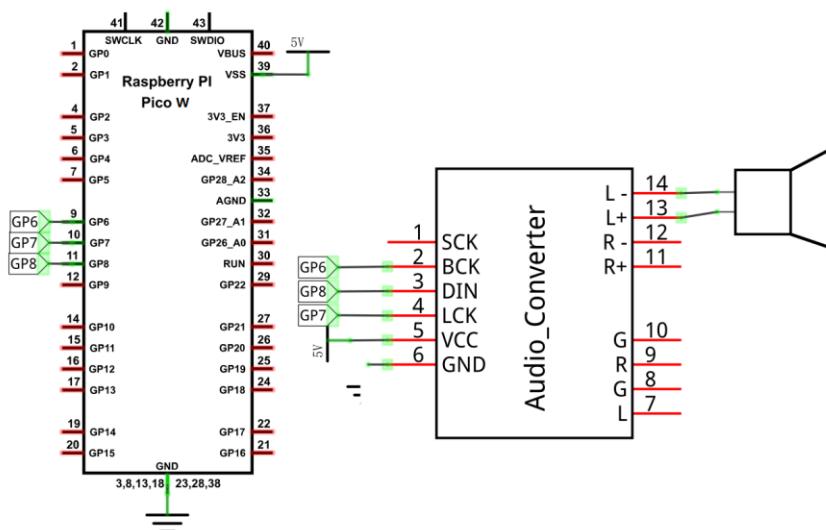
External audio input interface: connect to external audio equipment. Used to amplify externally input audio signals.

Power interface: connect to external power supply. External power supply selection range: 3.3V-5.0V.

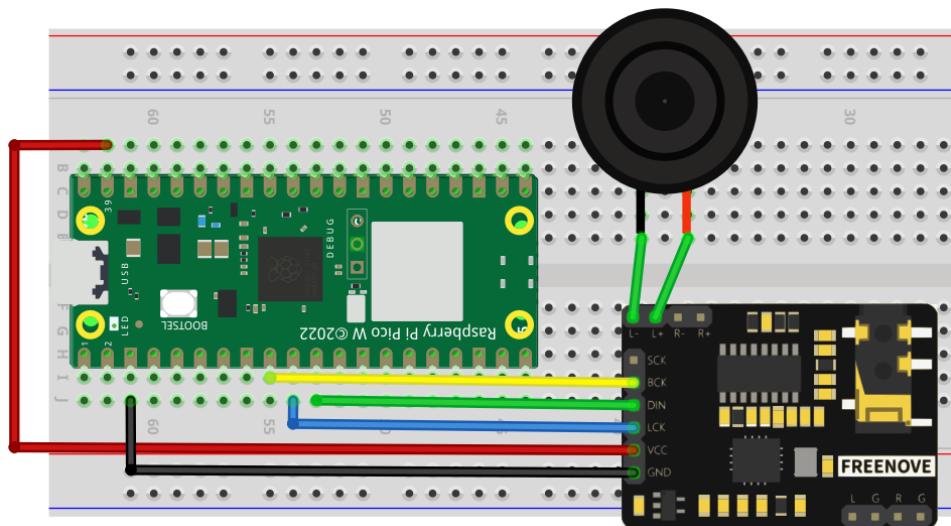
Circuit

The connection between the control board and the audio module is shown in the figure below.

Schematic diagram



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

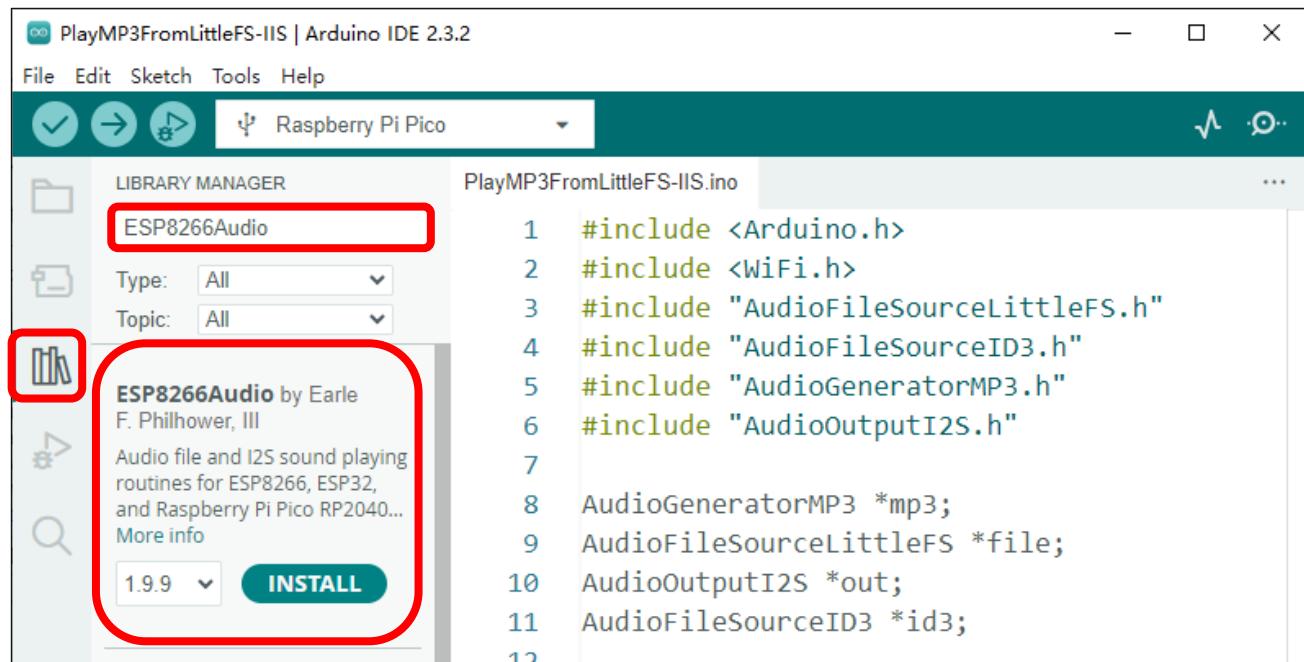


Sketch

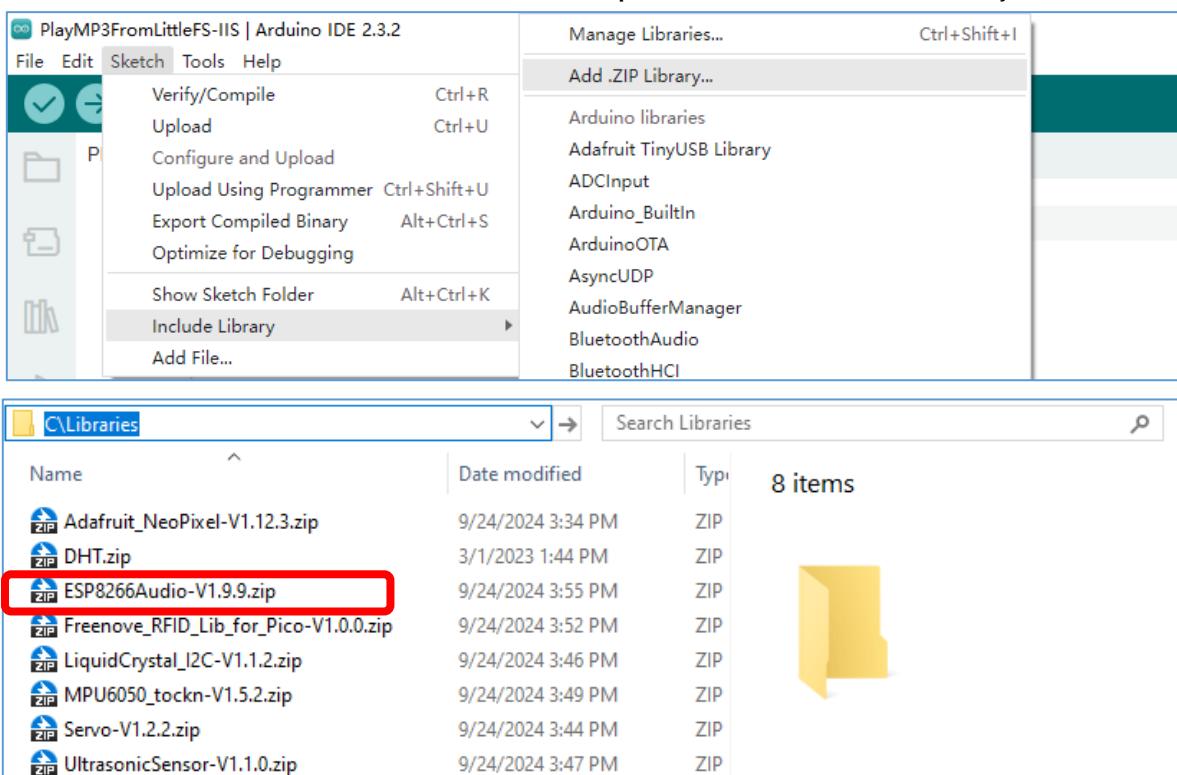
How to install the library

In this project, we use a third-party library named **ESP8266Audio**. Please install it first.

Open Arduino IDE, click **Library Manage** on the left, and search “**ESP8266Audio**” to install.



The second way, open Arduino IDE, click Sketch → Include Library → Add .ZIP Library. In the pop-up window, find the file named “./Libraries/**ESP8266Audio-V1.9.9.Zip**” which locates in this directory, and click OPEN.



Install the PicoLittleFS tool

The latest Arduino IDE has supported LittleFS plugins.

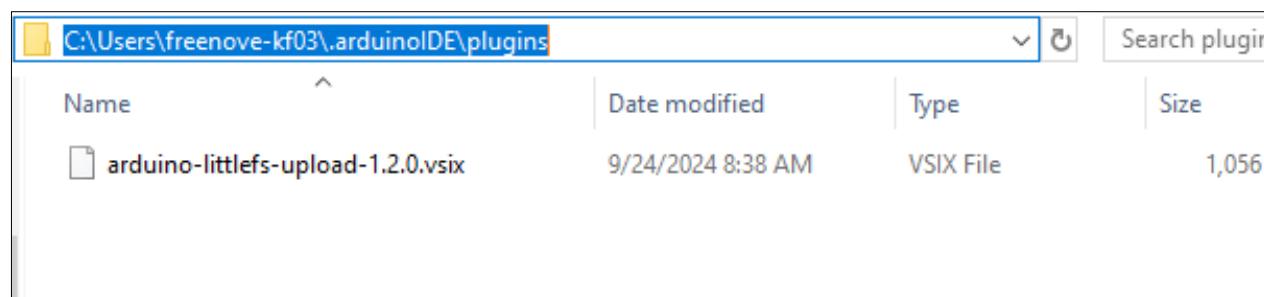
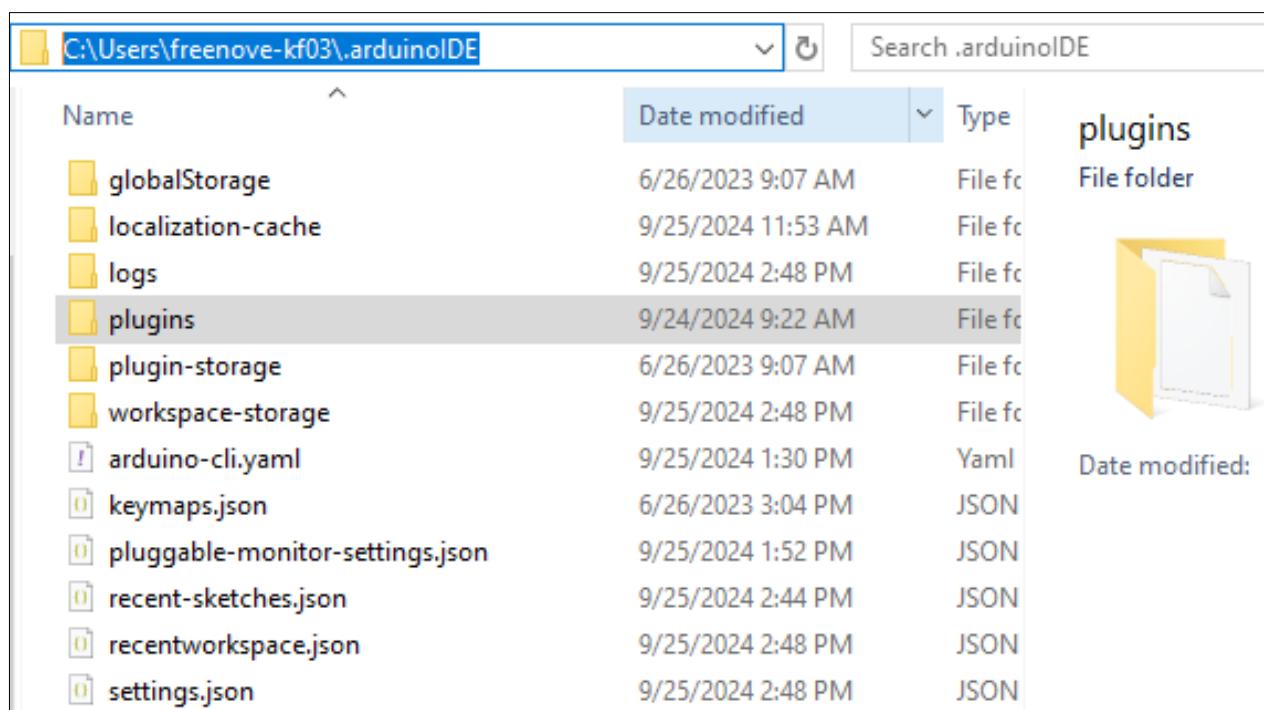
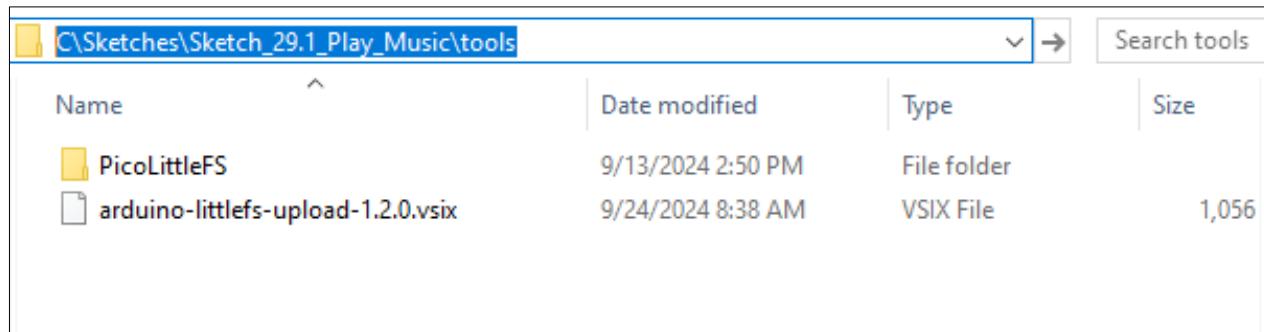
The steps to install PicoLittleFS is as follows:

Copy the file **arduino-littlefs-upload-1.2.0.vsix** under the directory

"Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\Sketches\Sketch_29.1_Play_Music\tools" to the computer's directory **~/.arduinoIDE/plugins/**

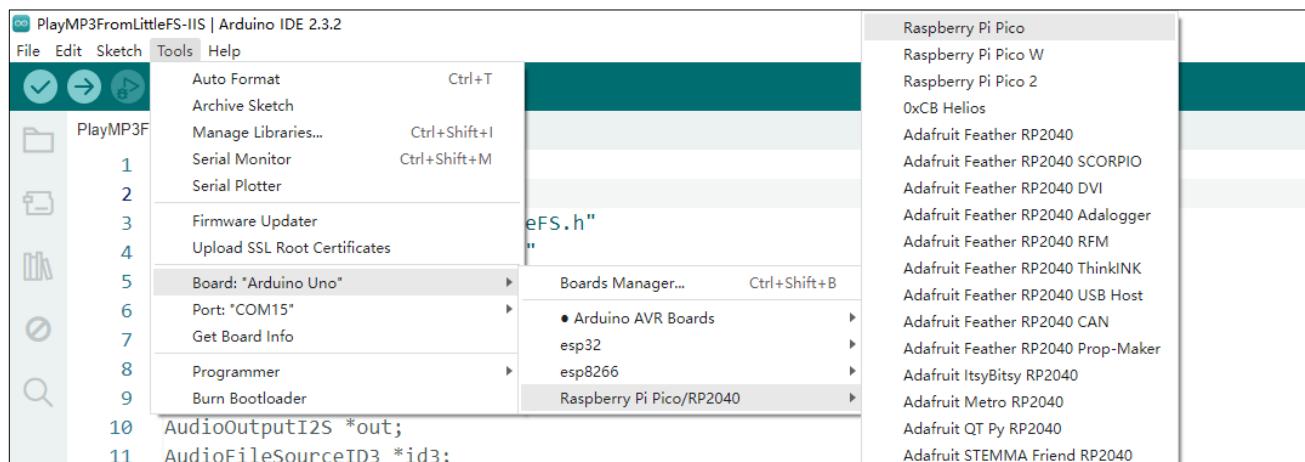
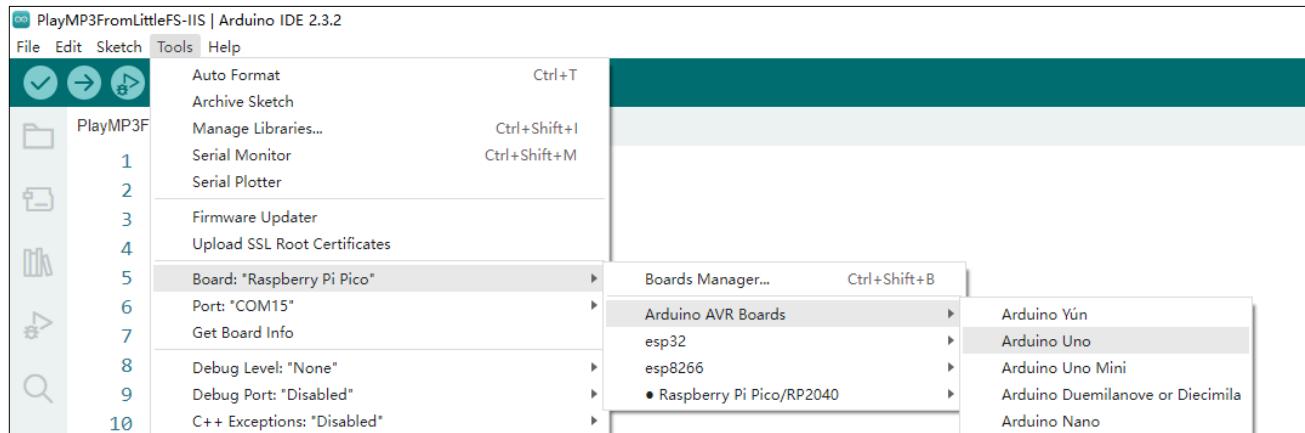
If there is no a folder name **plugins** in your computer, please create the folder before copying the file.

After copying the file, restart Arduino IDE.

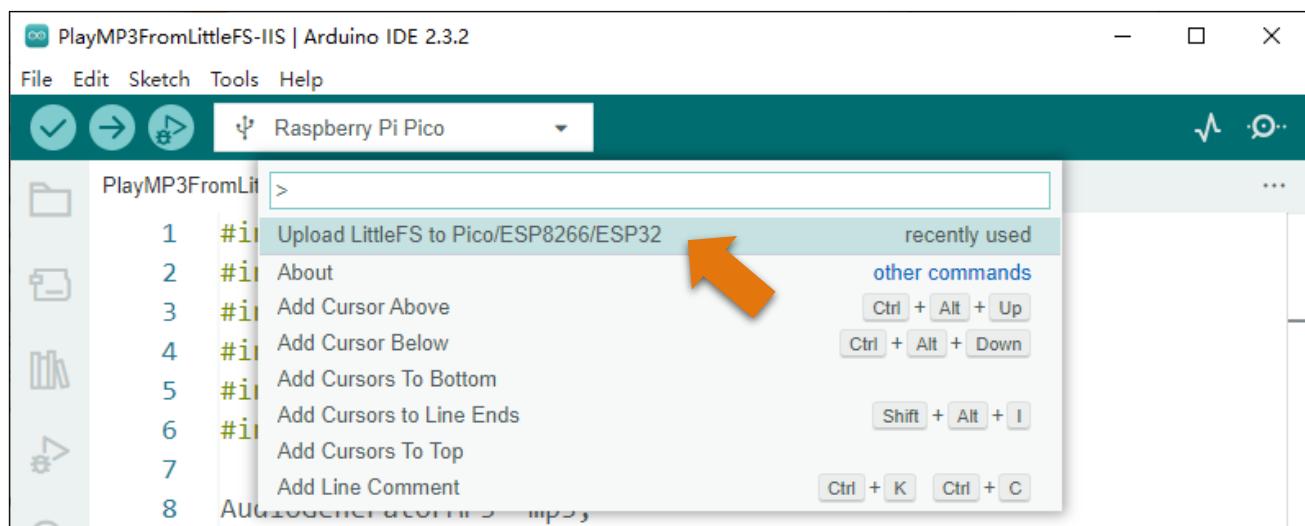


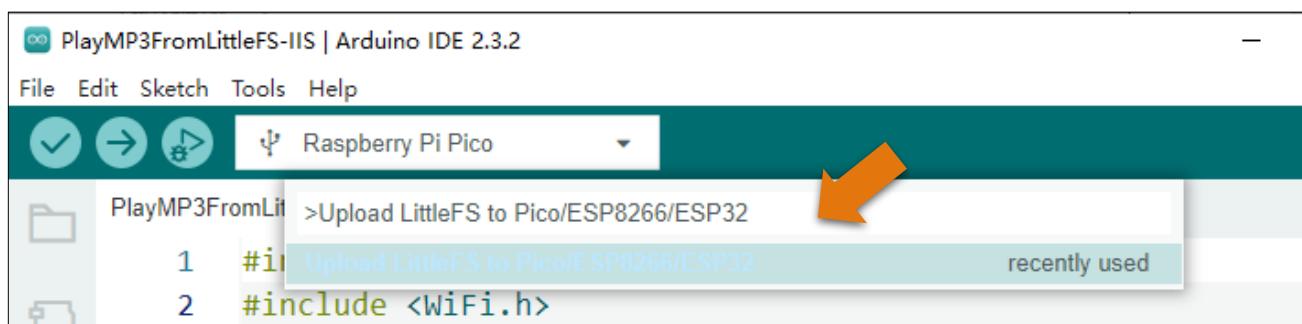


The sketch opened with the start of the Arduino IDE may be corrupted, which can lead to code uploading failure. To address this issue, you can change the board selection (switching to any board), and then select back the pico board. Alternatively, you can close the opened sketch and reopen it.



Here is how to use the PicoLittleFS tool, press [Ctrl]+[Shift]+[P] simultaneously, enter ***Upload LittleFS to Pico/ESP8266*** on the input field.



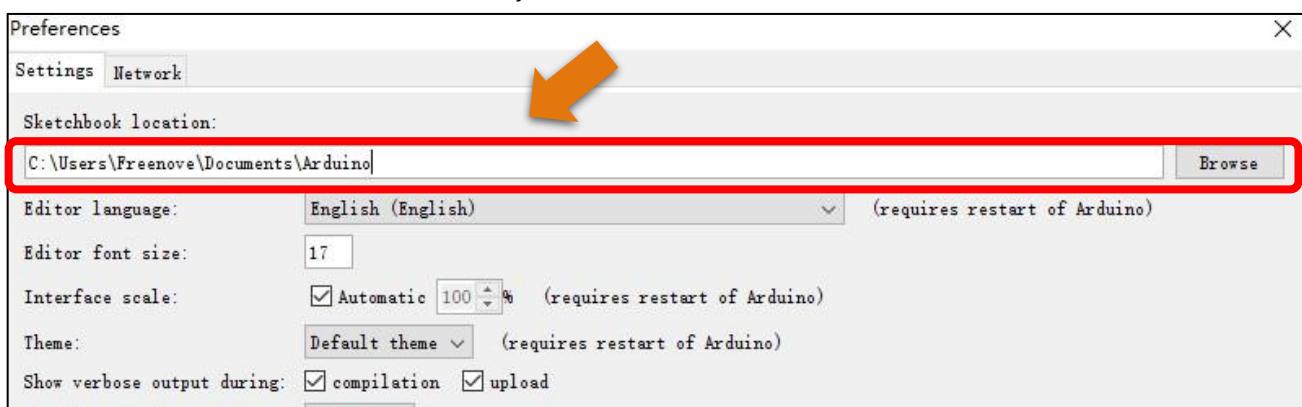


If your Arduino IDE is an old version one, like Arduino 1.x.x, please refer to the following steps to install the PicoLittleFS tool.

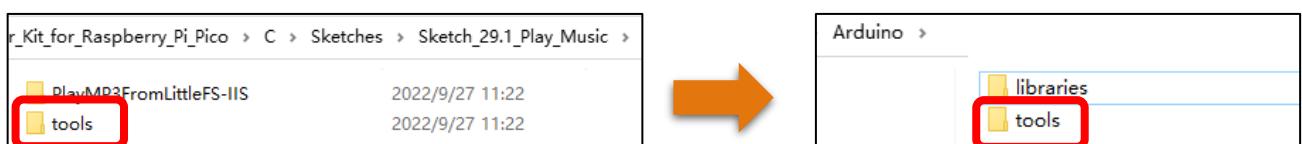
First, open the Arduino IDE, and then click File in Menus and select Preferences.



Find the Arduino IDE environment directory location.

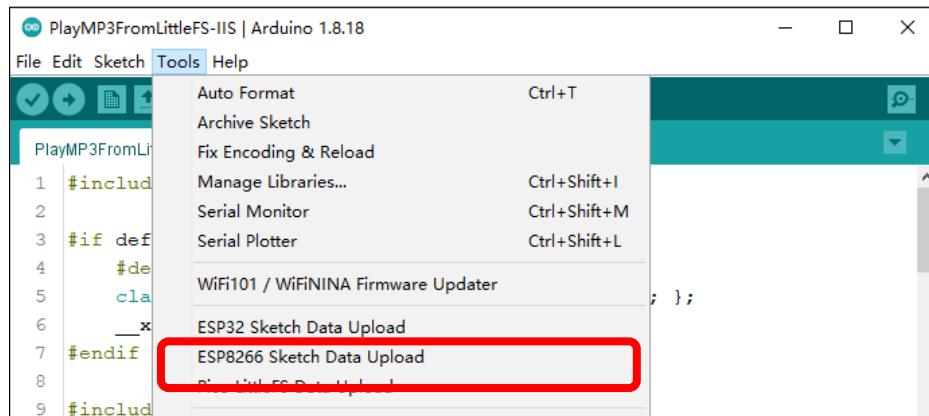


Copy the tools folder in the code folder to your Sketchbook location.





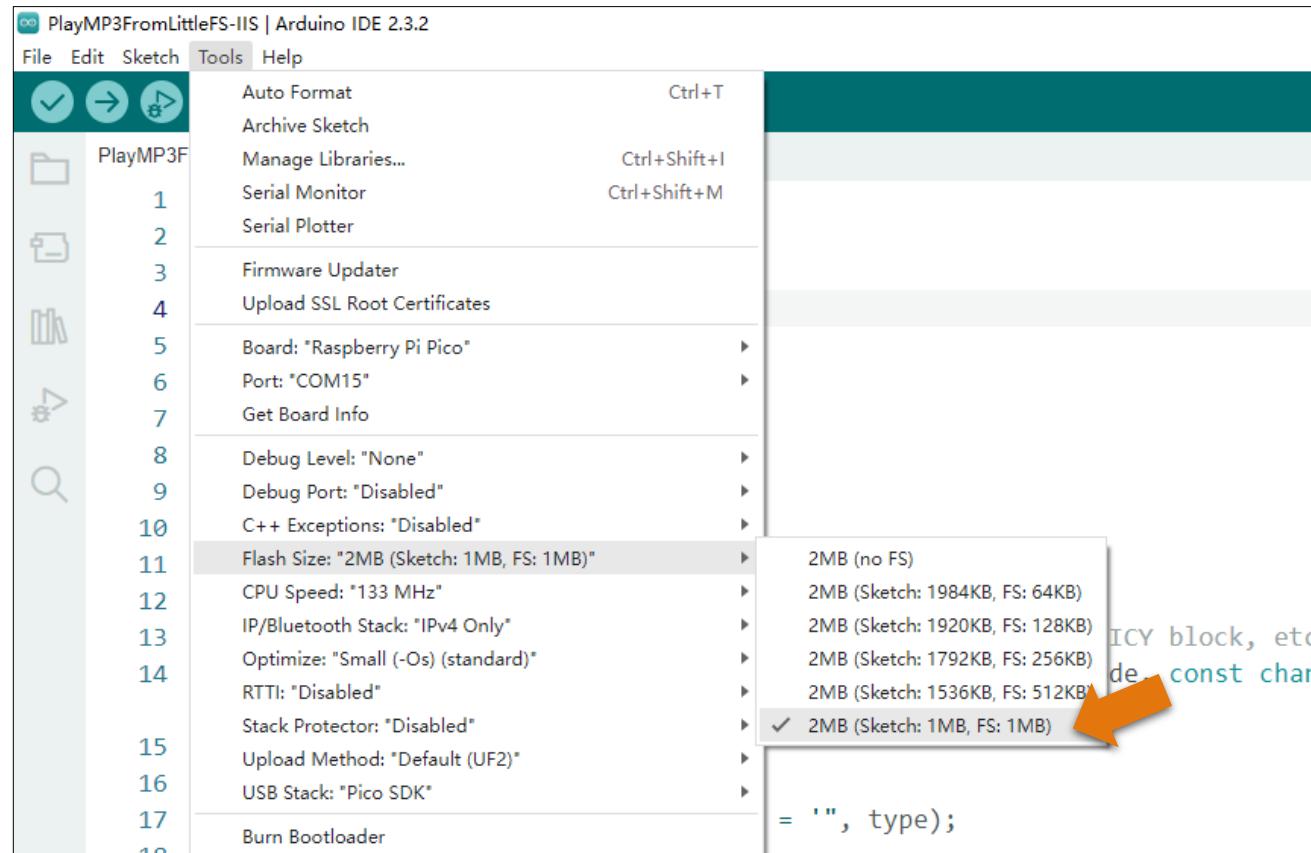
Finally, restart the Arduino IED. After restarting, you can see that the plug-in already exists in the interface.



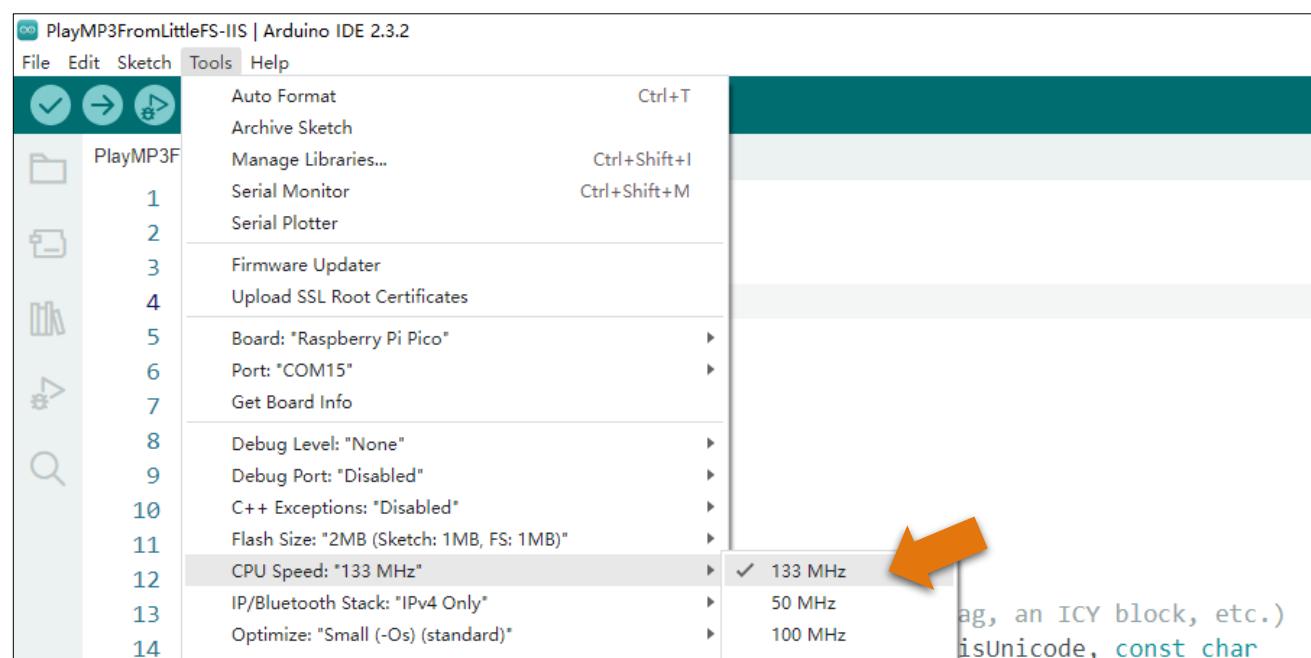
Upload music

Pico of Raspberry Pie has 2M Flash space. Generally, Arduino mode allocates it to the code area. Therefore, before starting, we need to modify the configuration of Flash Size.

Open Arduino, select Tools from the menu bar, select Flash Size, and allocate 1MB of Flash space to store codes and 1MB to store audio files.

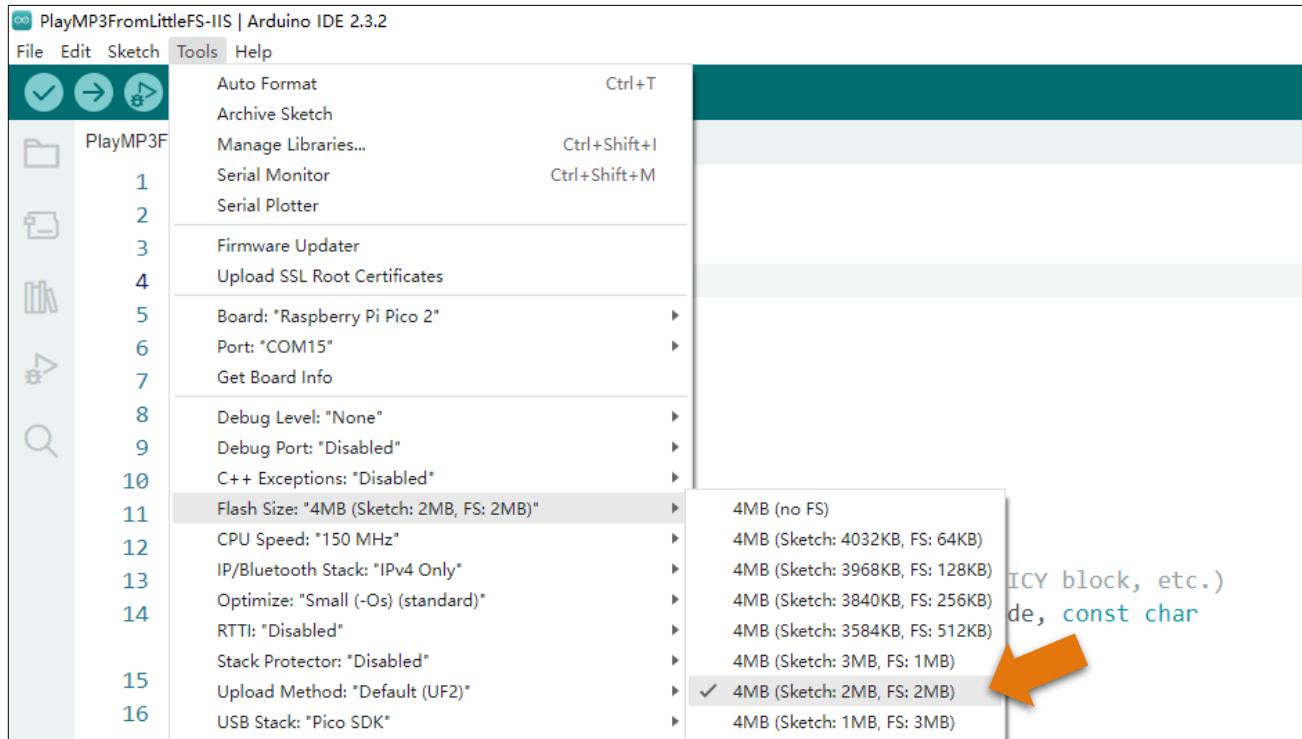


Make sure the CPU Speed of Raspberry Pie is 133 MHz. If the frequency is too low, the audio decoding speed may be too slow and the audio playback may not be continuous.

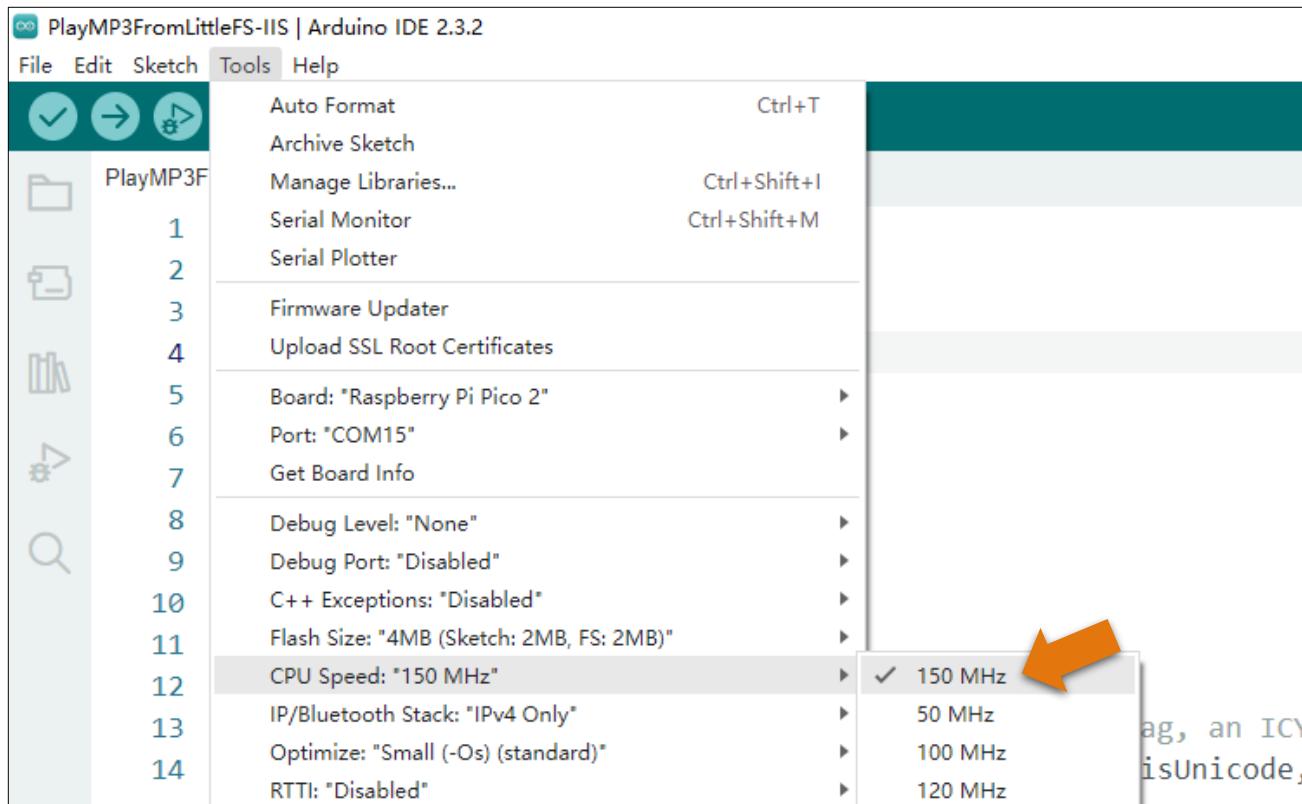




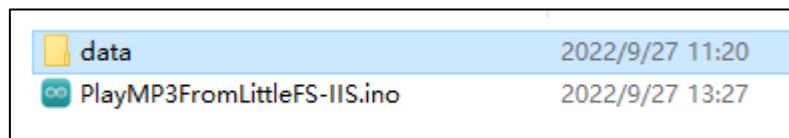
If your board is Raspberry Pi Pico 2, please select the Flash Size 4MB (Sketch: 2MB, FS: 2MB). As Pico 2 has 4M Flash space, so we assign 2M to store sketch and 2M to store audio files.



The CPU of Raspberry Pi Pico 2 can run up to 150MHz, so it is more appropriate to set the CPU speed to 150MHz.



Check the code file and audio file. We create a folder named data under the same level directory of the code file, and place the audio file directly in this folder.



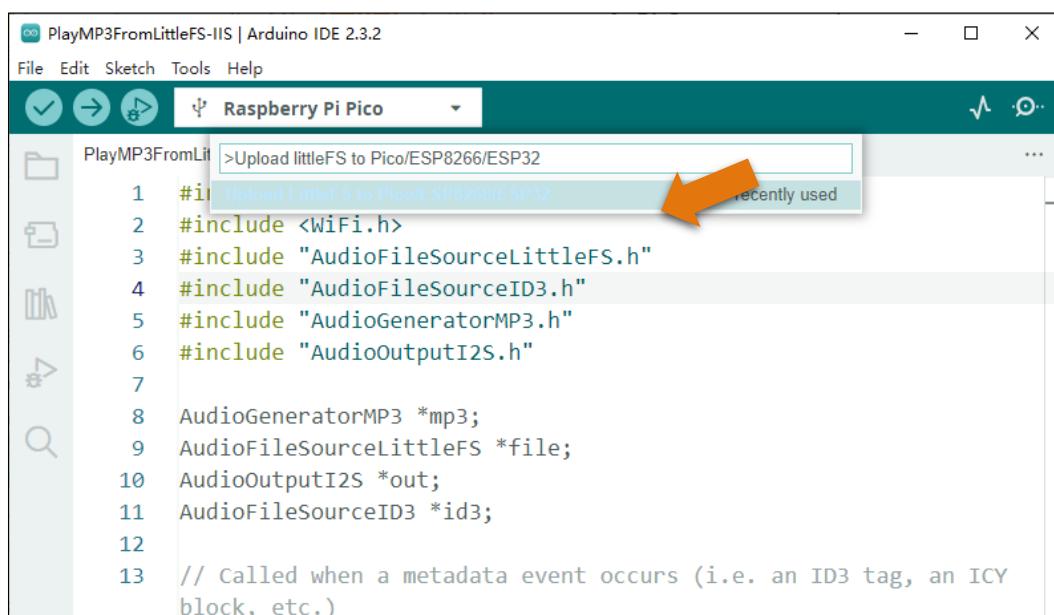
Note:

1. The name of the data folder cannot be changed; otherwise, the plug-in cannot be used to upload audio files to Pico.
2. The number of audio files in the data folder is unlimited, but the total size cannot exceed 1MB. If the file upload fails, please check whether the data folder size exceeds the range.

Click the upload button to upload the code to Pico:



After uploading the code, click Arduino IDE Tools and click following content:



```

Output LittleFS Upload x
LittleFS Filesystem Uploader v1.2.0 -- https://github.com/earlephilhower/arduino-littlefs-upload

Sketch Path: F:\FNK0058_63_64_65_Pico(w)_Kit\01.Pico全系列教程修改\All_Pico_Kit\Sketches\sketch_29_1_Play_Music\PlayMP3FromLittleFS-IIIS
  Data Path: F:\FNK0058_63_64_65_Pico(w)_Kit\01.Pico全系列教程修改\All_Pico_Kit\Sketches\sketch_29_1_Play_Music\PlayMP3FromLittleFS-IIIS\data
  Device: RP2040 series

Building LittleFS filesystem
Command Line: C:\Users\freenove-kf03\AppData\Local\Arduino15\packages\rp2040\tools\pqt-mklittlefs\2.2.0-d04e724\mklittlefs.exe -c F:\FNK0058_63_64_65_Pico(w)_Kit\01.Pico全系列教程修改\All_Pico_Kit\C\Sketches\sketch_29_1_Play_Music\PlayMP3FromLittleFS-IIIS\data -p 256 -b 4096 -s 1048576 C:\Users\freenove~1\AppData\Local\Temp\tmp-13032-8H3Ucxxf0a1o-.littlefs.bin

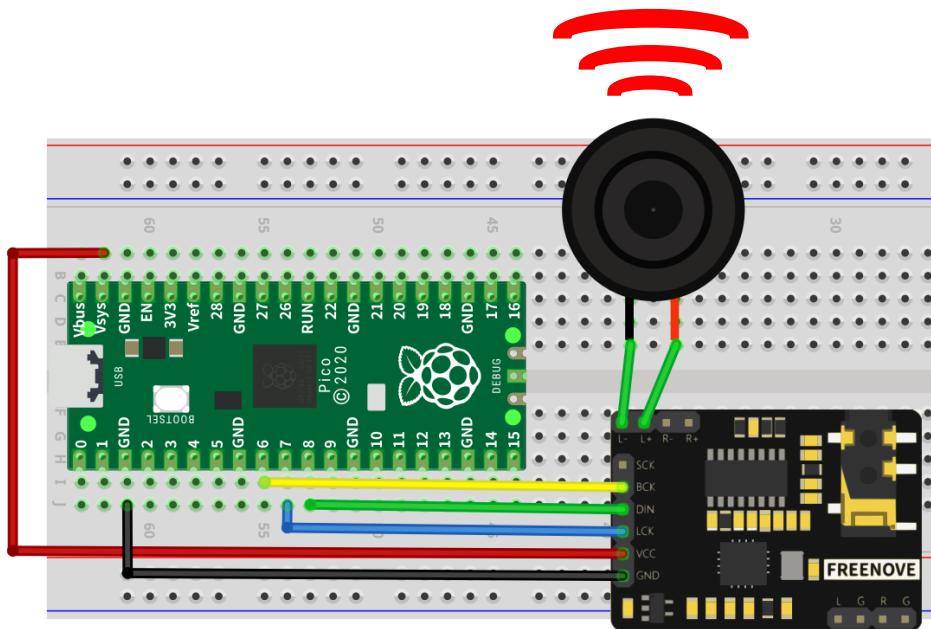
4.0 or above

Generating UF2 image
Command Line: C:\Users\freenove-kf03\AppData\Local\Arduino15\packages\rp2040\tools\pqt-picotool\2.2.0-8a9af99-2\picotool.exe uf2 convert C:\Users\freenove~1\AppData\Local\Temp\tmp-13032-8H3Ucxxf0a1o-.littlefs.bin -t bin C:\Users\freenove\kf03\AppData\Local\Temp\tmp-13032-8H3Ucxxf0a1o-.littlefs.uf2 -o 0x100ff000 --family data

Uploading LittleFS filesystem
Command Line: C:\Users\freenove-kf03\AppData\Local\Arduino15\packages\rp2040\tools\pqt-uf2conv.py --serial COM15 --family RP2040 C:\Users\freenove\kf03\AppData\Local\Temp\tmp-13032-8H3Ucxxf0a1o-.littlefs.uf2 --deploy
Resetting COM15
Converting to uf2, output size: 2097152, start address: 0x2000
Scanning for RP2040 devices
Flashing I: (RPI-RP2)
Wrote 2097152 bytes to I:/NEW.UF2
Completed upload.

```

After the music upload is completed, the speaker will play the audio file you uploaded.



Sketch_29.1_Play_Music

```
1 #include <Arduino.h>
2 #include <WiFi.h>
3 #include "AudioFileSourceLittleFS.h"
4 #include "AudioFileSourceID3.h"
5 #include "AudioGeneratorMP3.h"
6 #include "AudioOutputI2S.h"
7
8 AudioGeneratorMP3 *mp3;
9 AudioFileSourceLittleFS *file;
10 AudioOutputI2S *out;
11 AudioFileSourceID3 *id3;
12
13 // Called when a metadata event occurs (i.e. an ID3 tag, an ICY block, etc.)
14 void MDCallback(void *cbData, const char *type, bool isUnicode, const char *string)
15 {
16     (void)cbData;
17     Serial.printf("ID3 callback for: %s = '", type);
18
19     if (isUnicode)
20     {
21         string += 2;
22     }
23
24     while (*string)
25     {
26         char a = *(string++);
27         if (isUnicode)
28         {
29             string++;
30         }
31         Serial.printf("%c", a);
32     }
33     Serial.printf("\r\n");
34     Serial.flush();
35 }
36
37 void setup()
38 {
39     WiFi.mode(WIFI_OFF);
40     Serial.begin(115200);
41     delay(1000);
42     Serial.printf("Sample MP3 playback begins... \r\n");
43 }
```

```

44     audioLogger = &Serial;
45     out = new AudioOutputI2S();
46     out->SetPinout(6, 7, 8); // Set the pins for IIS(BCK, LCK, DIN)
47     out->SetGain(0.5);
48     file = new AudioFileSourceLittleFS("1.mp3");
49     id3 = new AudioFileSourceID3(file);
50     id3->RegisterMetadataCB(MDCallback, (void *)"ID3TAG");
51     mp3 = new AudioGeneratorMP3();
52     mp3->begin(id3, out);
53 }
54
55 void loop()
56 {
57     if (mp3->isRunning())
58     {
59         if (!mp3->loop())
60             mp3->stop();
61     }
62     else
63     {
64         Serial.printf("MP3 done\r\n");
65         delay(1000);
66     }
67 }
```

Check whether the file has finished playing in the main loop function.

```

55 void loop()
56 {
57     if (mp3->isRunning())
58     {
59         if (!mp3->loop())
60             mp3->stop();
61     }
62     else
63     {
64         Serial.printf("MP3 done\r\n");
65         delay(1000);
66     }
67 }
```

Associate the callback information output of the decoding library to the serial port. Initialize the IIS interface, and set the IIS pin interfaces to GP6, GP7, and GP8. Set the volume to 0.5.

```

44     audioLogger = &Serial;
45     out = new AudioOutputI2S();
46     out->SetPinout(6, 7, 8); // Set the pins for IIS(BCK, LCK, DIN)
47     out->SetGain(0.5); //Volume range: 0~3.9
```

Any concerns? ✉ support@freenove.com

Note that the name of the audio file in the data folder must be consistent with the name in the code; otherwise, music cannot be played because music cannot be retrieved.

```
48     file = new AudioFileSourceLittleFS("1.mp3");
49     id3 = new AudioFileSourceID3(file);
50     id3->RegisterMetadataCB(MDCallback, (void *)"ID3TAG");
51     mp3 = new AudioGeneratorMP3();
52     mp3->begin(id3, out);
```



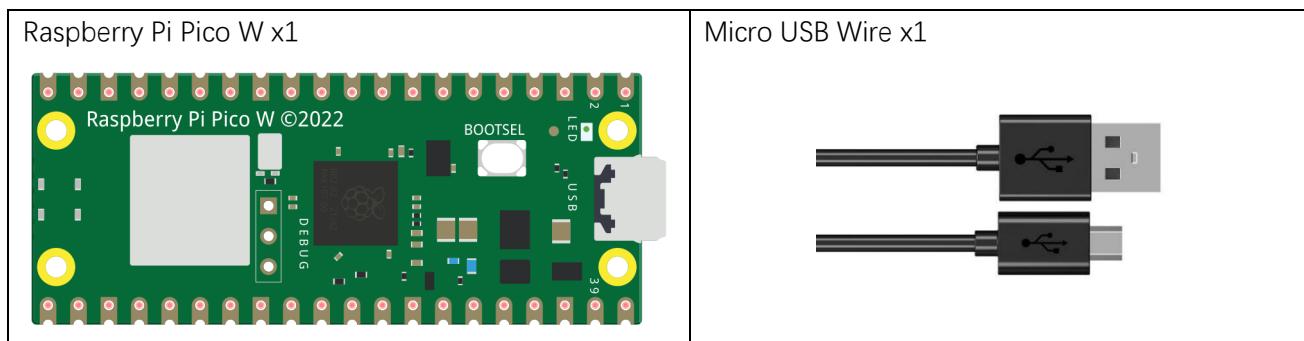
Chapter 30 WiFi Working Modes (Only for Pico W)

The biggest difference between the raspberry pi pico and the raspberry pi Pico W is that the raspberry pi pico W is equipped with a WiFi function module. At the beginning of this chapter, we will learn about the WiFi function of Pico W of Raspberry Pi.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

Project 30.1 Station mode

Component List



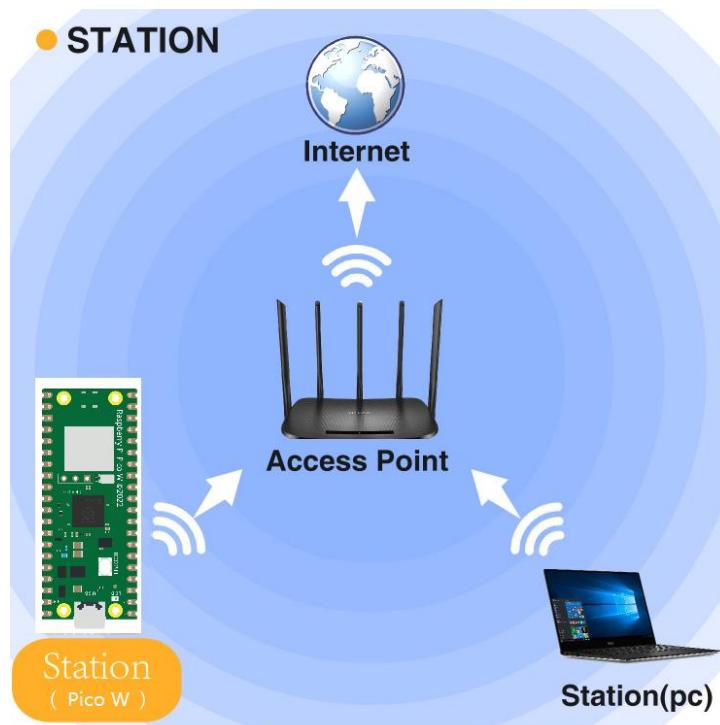
Component knowledge

Wireless

Pico W has an on-board 2.4GHz wireless interface using an Infineon CYW43439. The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The wireless interface is connected via SPI to the RP2040.

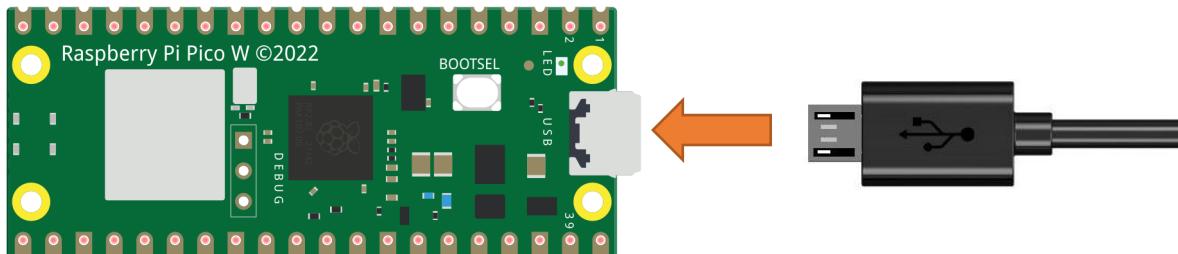
Station mode

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



Circuit

Connect Pico W to the computer using the USB cable.





Sketch

Sketch_30.1_Station_mode

```
Sketch_30.1_WiFi_Station | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ψ Raspberry Pi Pico W
Sketch_30.1_WiFi_Station.ino
7 #include <WiFi.h>
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11
12 void setup(){
13     Serial.begin(115200);
14     delay(2000);
15     Serial.println("Setup start");
16     WiFi.begin(ssid_Router, password_Router);
17     Serial.println(String("Connecting to ") + ssid_Router);
18     while (WiFi.status() != WL_CONNECTED){
19         delay(500);
20         Serial.print(".");
21     }
--
```

Output

Ln 9, Col 31 Raspberry Pi Pico W on COM9 2

Enter the correct Router name and password.

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

After making sure the router name and password are entered correctly, compile and upload codes to Pico W, open serial monitor and set baud rate to 115200. Then it will display as follows:



When PICO W successfully connects to “ssid_Router”, serial monitor will print out the IP address assigned to PICO W by the router.

The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5
6 void setup() {
7     Serial.begin(115200);
8     delay(2000);
9     Serial.println("Setup start");
10    WiFi.begin(ssid_Router, password_Router);
11    Serial.println(String("Connecting to ") + ssid_Router);
12    while (WiFi.status() != WL_CONNECTED) {
13        delay(500);
14        Serial.print(".");
15    }
16    Serial.println("\nConnected, IP address: ");
17    Serial.println(WiFi.localIP());
18    Serial.println("Setup End");
19 }
20 void loop() {
21 }

```

Include the WiFi Library header file of Pico W.

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

Enter correct router name and password.

```

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

```

Set Pico W in Station mode and connect it to your router.

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

Check whether Pico W has connected to router successfully every 0.5s.

```

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

```

Serial monitor prints out the IP address assigned to Pico W.

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

Reference

Class Station

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

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

ssid: WiFi hotspot name

password: WiFi hotspot password

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

bssid: mac address of WiFi hotspot, optional parameter

connect: boolean optional parameter, defaulting to true. If set as false, then PICO W won't connect WiFi.

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

local_ip: station fixed IP address.

subnet: subnet mask

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

status: obtain the connection status of WiFi

local IP(): obtain IP address in Station mode

disconnect(): disconnect wifi

Project 30.2 AP mode

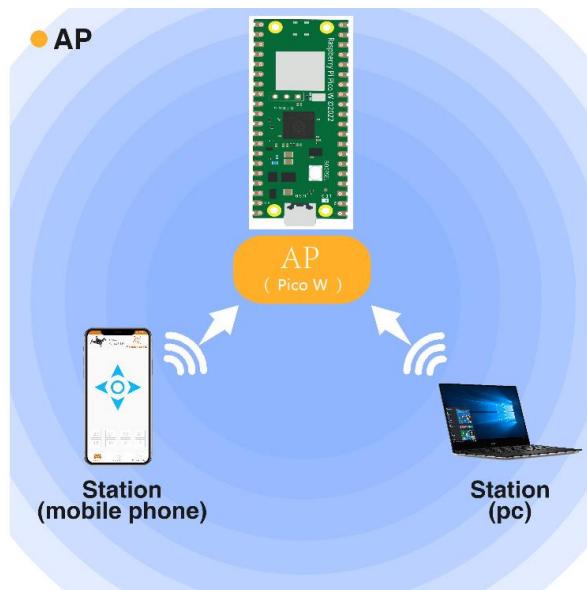
Component List & Circuit

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

Component knowledge

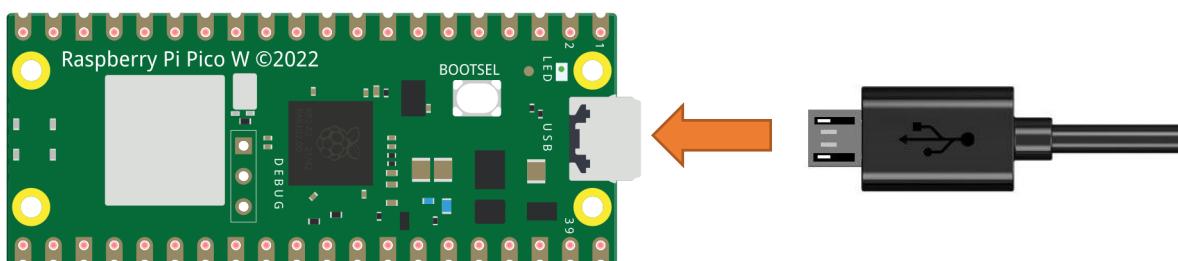
AP mode

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



Circuit

Connect Pico W to the computer using the USB cable.





Sketch

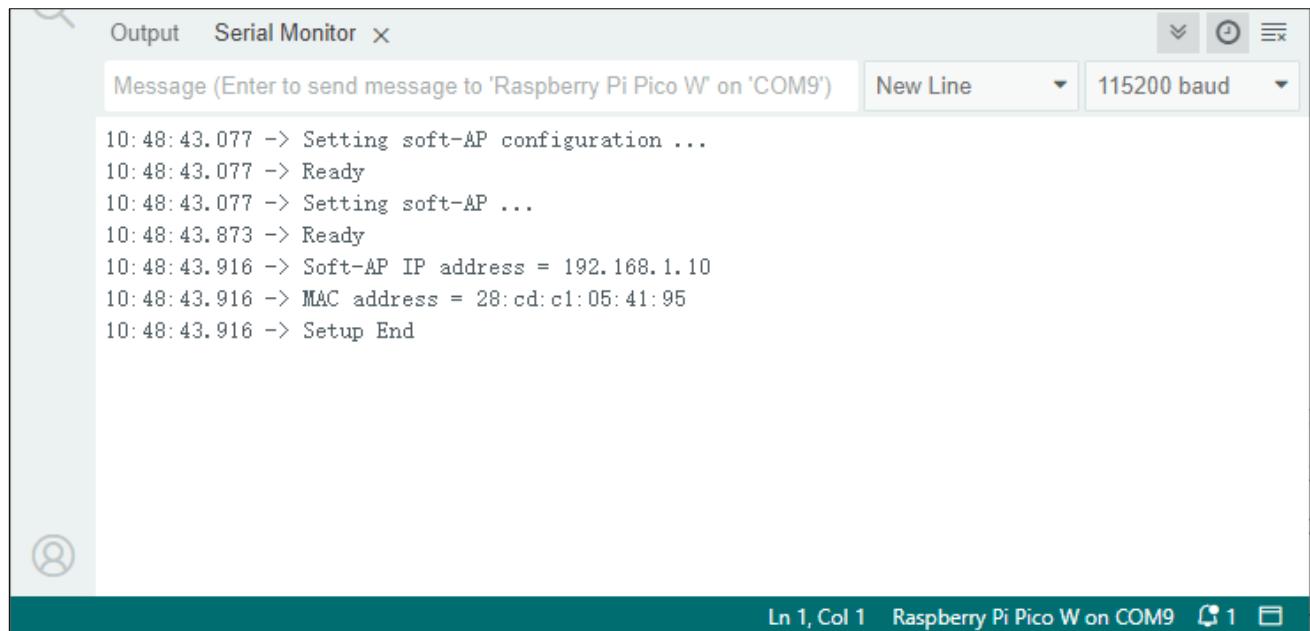
Sketch_30.2_AP_mode

```
Sketch_30.2_WIFI_AP | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_30.2_WIFI_AP.ino
7 #include <WiFi.h>
8
9 const char *ssid_AP      = "WiFi_Name"; //Enter the router name
10 const char *password_AP = "12345678"; //Enter the router password
11
12 IPAddress local_IP(192,168,1,100); //Set the IP address of ESP8266 itself
13 IPAddress gateway(192,168,1,10); //Set the gateway of ESP8266 itself
14 IPAddress subnet(255,255,255,0); //Set the subnet mask for itself
15
16 void setup(){
17     Serial.begin(115200);
18     delay(2000);
19     Serial.println("Setting soft-AP configuration ... ");
20     WiFi.disconnect();
```

Set a name and a password for PICO W AP mode.

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

Compile and upload codes to PICO W, open the serial monitor and set the baud rate to 115200. Then it will display as follows.



The screenshot shows the 'Serial Monitor' window with the title 'Output Serial Monitor'. The message field contains the following log output:

```

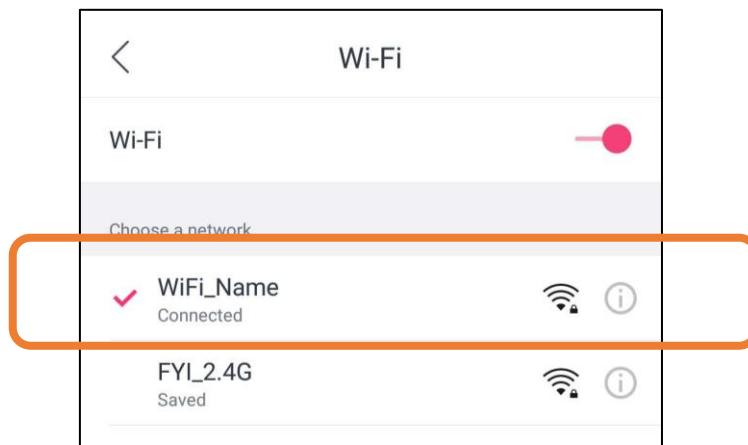
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')
New Line 115200 baud

10:48:43.077 -> Setting soft-AP configuration ...
10:48:43.077 -> Ready
10:48:43.077 -> Setting soft-AP ...
10:48:43.873 -> Ready
10:48:43.916 -> Soft-AP IP address = 192.168.1.10
10:48:43.916 -> MAC address = 28:cd:c1:05:41:95
10:48:43.916 -> Setup End

```

The status bar at the bottom shows 'Ln 1, Col 1 Raspberry Pi Pico W on COM9'.

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



Note:

1. Every time you change the WiFi name and password in the code, please power off and then on again, and then upload the code. It is possible that the WiFi name and password have not actually changed due to the direct uploading of code without power. This is because Pico W WiFi module and RP2040 chip are separated. Only when the power is cut off can the WiFi name and password be flashed to the WiFi module again.
2. Pico W executes this code only to open a WiFi hotspot, and does not configure the code related to online data transmission, so the mobile phone will display no data after connection.



The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_AP      = "WiFi_Name"; //Enter the router name
4 const char *password_AP = "12345678"; //Enter the router password
5
6 IPAddress local_IP(192, 168, 1, 100); //Set the IP address of PICO W itself
7 IPAddress gateway(192, 168, 1, 10); //Set the gateway of PICO W itself
8 IPAddress subnet(255, 255, 255, 0); //Set the subnet mask for PICO W itself
9
10 void setup() {
11     Serial.begin(115200);
12     delay(2000);
13     Serial.println("Setting soft-AP configuration ... ");
14     WiFi.disconnect();
15     WiFi.mode(WIFI_AP);
16     Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
17     Serial.println("Setting soft-AP ... ");
18     boolean result = WiFi.softAP(ssid_AP, password_AP);
19     if(result) {
20         Serial.println("Ready");
21         Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
22         Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
23     } else {
24         Serial.println("Failed!");
25     }
26     Serial.println("Setup End");
27 }
28
29 void loop() {
30 }
```

Include WiFi Library header file of PICO W.

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

Enter correct AP name and password.

```

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

Set PICO W in AP mode.

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

Configure IP address, gateway and subnet mask for PICO W.

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

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

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

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

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

Reference

Class AP

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

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

ssid: WiFi hotspot name

password: WiFi hotspot password

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

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

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

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

local_ip: station fixed IP address.

Gateway: gateway IP address

subnet: subnet mask

softAP(): obtain IP address in AP mode

softAPdisconnect (): disconnect AP mode.



Project 30.3 AP+Station mode

Component List & Circuit

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

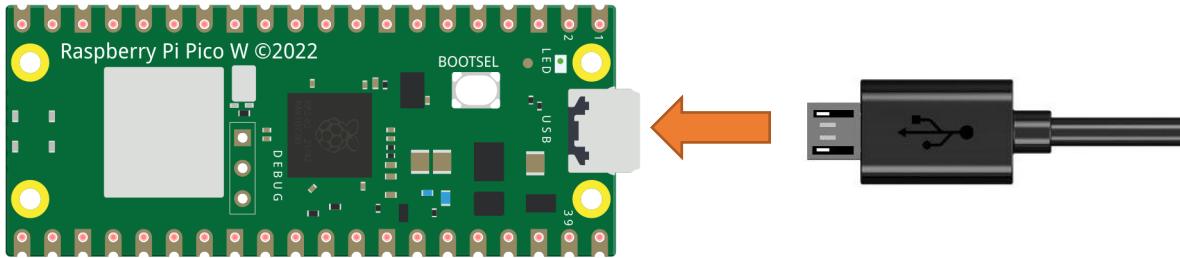
Component knowledge

AP+Station mode

PICO W currently does not support simultaneous use of AP mode and Station mode, so this section can be skipped. In the actual mode configuration, the last configured mode shall prevail.

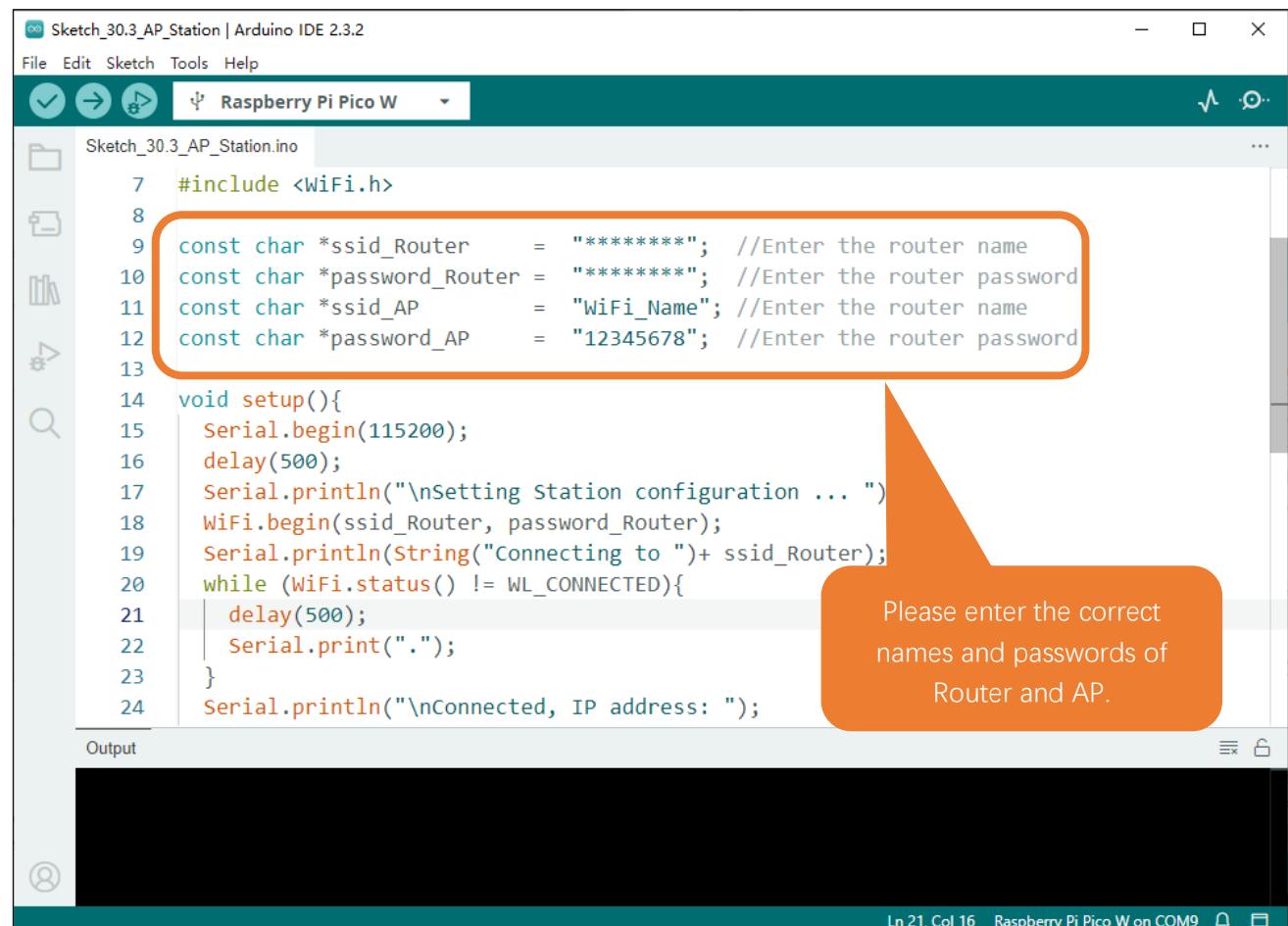
Circuit

Connect Pico W to the computer using the USB cable.



Sketch

Sketch_30.3_AP_Station_mode



The screenshot shows the Arduino IDE interface with the sketch file "Sketch_30.3_AP_Station.ino" open. The code is as follows:

```
#include <WiFi.h>
const char *ssid_Router      = "*****"; //Enter the router name
const char *password_Router = "*****"; //Enter the router password
const char *ssid_AP          = "WiFi_Name"; //Enter the router name
const char *password_AP      = "12345678"; //Enter the router password

void setup(){
    Serial.begin(115200);
    delay(500);
    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: ");
}
```

A callout bubble points to the configuration parameters (ssid_Router, password_Router, ssid_AP, password_AP) which are highlighted with a red box. The text inside the callout bubble reads: "Please enter the correct names and passwords of Router and AP."

It is analogous to project 30.1 and project 30.2. Before running the Sketch, you need to modify ssid_Router, password_Router, ssid_AP and password_AP shown in the box of the illustration above.



After making sure that Sketch is modified correctly, compile and upload codes to PICO W, open serial monitor and set baud rate to 115200. Then it will display as follows:

The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor". The message area contains the following log output:

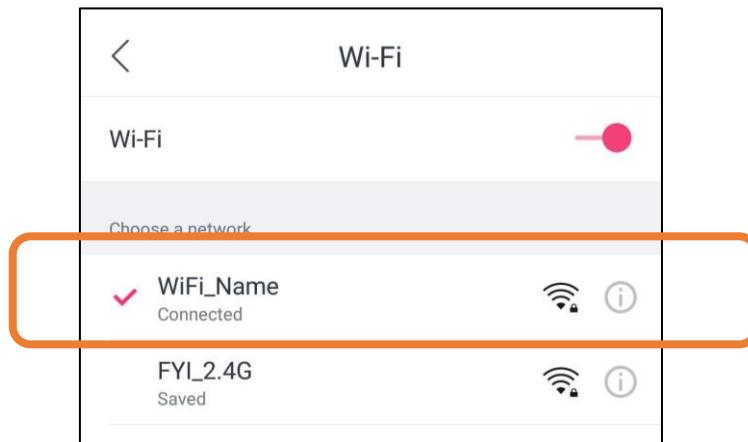
```

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')
10:53:45.024 -> Connecting to FYI_2.4G
10:53:45.024 ->
10:53:45.024 -> Connected, IP address:
10:53:45.024 -> 192.168.1.23
10:53:45.518 -> Setting soft-AP configuration ...
10:53:45.518 -> Setting soft-AP ...
10:53:46.357 -> Ready
10:53:46.357 -> Soft-AP IP address = 192.168.1.1
10:53:46.357 -> MAC address = 28:cd:c1:05:41:95
10:53:46.357 -> Setup End

```

The status bar at the bottom right shows "Ln 15, Col 24 Raspberry Pi Pico W on COM9" and icons for a bell and a refresh.

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



The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5 const char *ssid_AP          = "WiFi_Name"; //Enter the AP name
6 const char *password_AP      = "12345678"; //Enter the AP password
7
8 void setup() {
9     Serial.begin(115200);

```

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



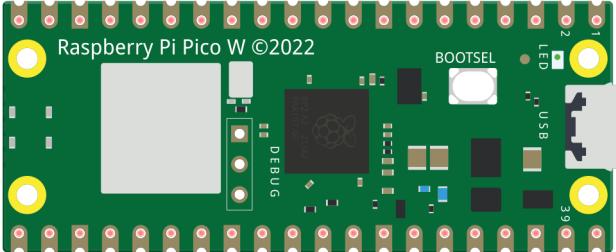
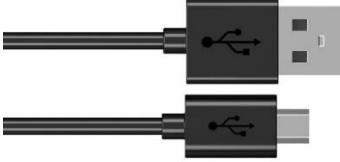
Chapter 31 TCP/IP (Only for Pico W)

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

Project 31.1 as Client

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

Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	

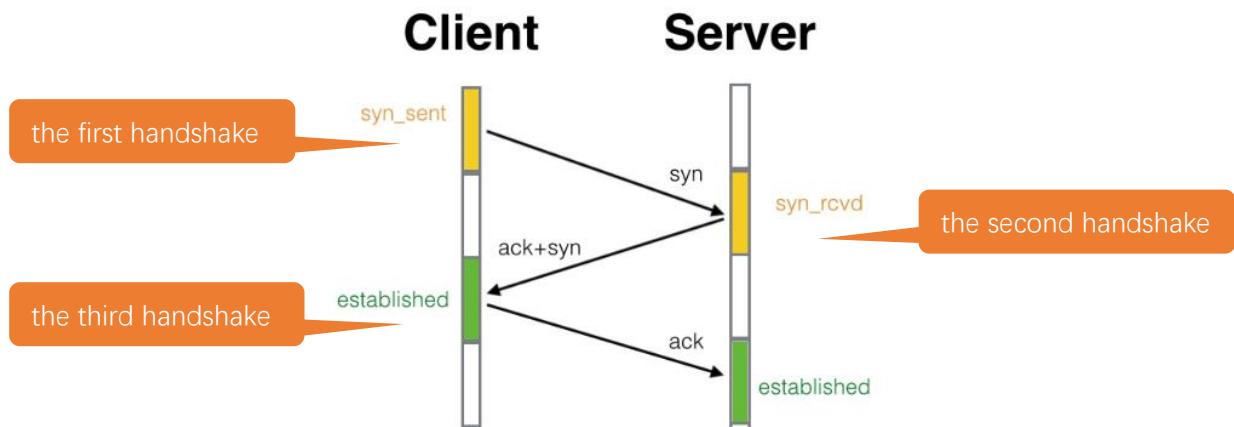
Component knowledge

TCP connection

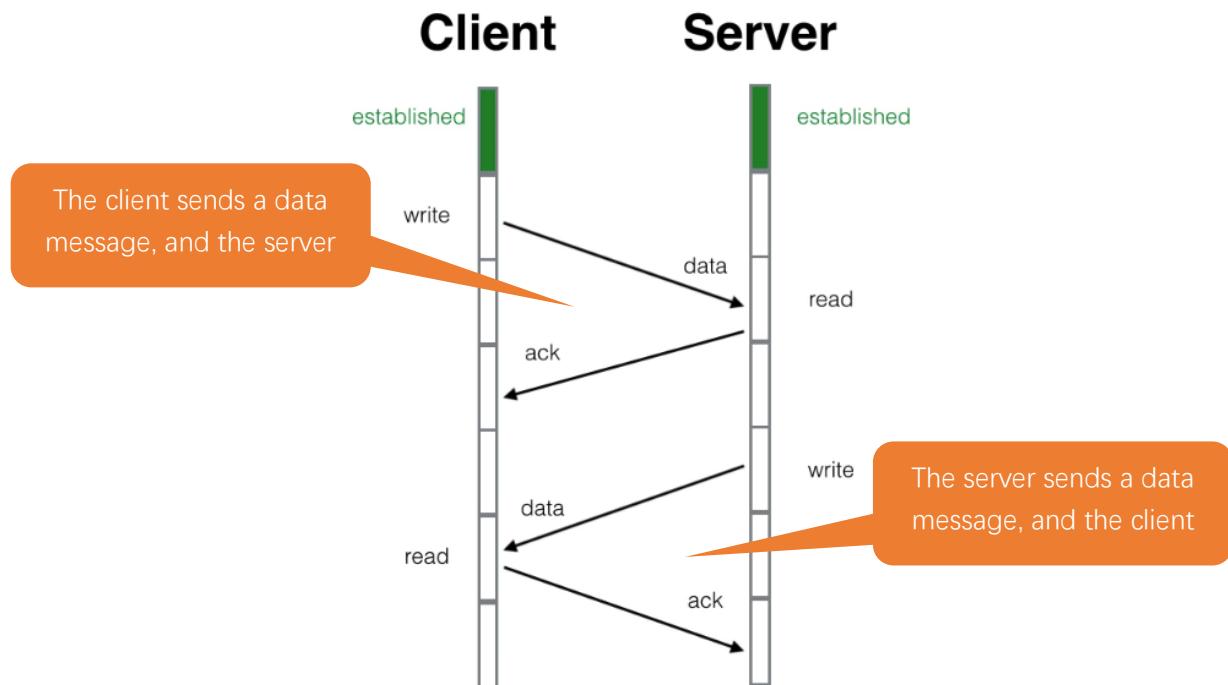
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request to the server, and each time such a request is proposed, a "three-time handshake" is required.

Three-time handshake: In the TCP protocol, during the preparation phase of sending data, the client and the server interact three times, to ensure the reliability of the connection, which is called "three-times handshake". The first handshake, the client sends a connection request to the server and waits for the server to confirm. The second handshake, the server sends a response back to the client informing that it has received the connection request.

The third handshake, the client sends a confirmation message to the server again to confirm the connection.



TCP is a connection-oriented, low-level transmission control protocol. After TCP establishes a connection, the client and server can send and receive messages to each other, and the connection will always exist as long as the client or server does not initiate disconnection. Each time one party sends a message, the other party will reply with an ack signal.





Install Processing

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

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

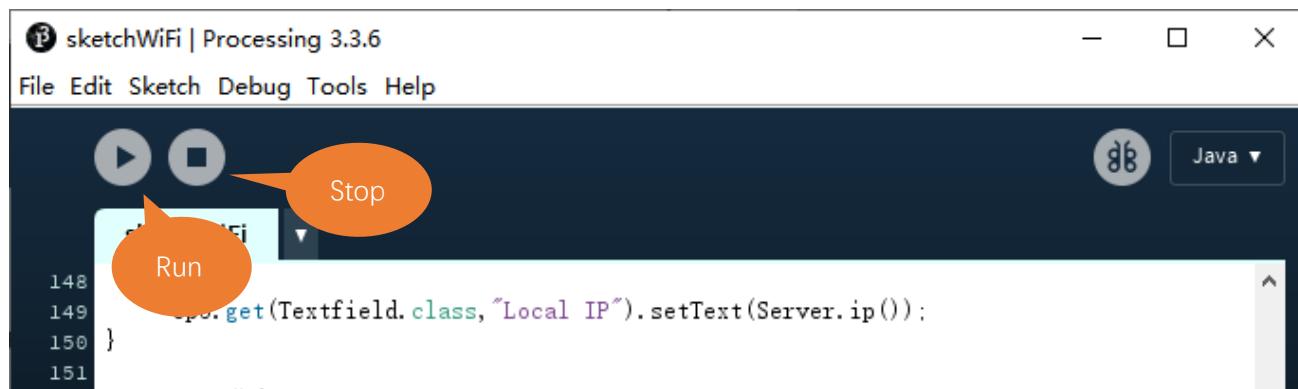
The screenshot shows the official Processing website's download section. At the top, there are tabs for "Processing", "p5.js", "Processing.py", "Processing for Android", "Processing for Pi", and "Processing Foundation". A search bar is located in the top right. The main content area features a large "Processing" logo with a geometric background. To the left is a sidebar with links: "Cover", "Download", "Donate", "Exhibition", "Reference", "Libraries", "Tools", "Environment", "Tutorials", "Examples", "Books", "Overview", and "People". In the center, it says "Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below." Below this, the "3.5.4 (17 January 2020)" release is listed with download links for "Windows 64-bit", "Windows 32-bit", "Linux 64-bit", and "Mac OS X". To the right of the release info, there are links to "» Github", "» Report Bugs", "» Wiki", "» Supported Platforms", and a note about "changes in 3.0".

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

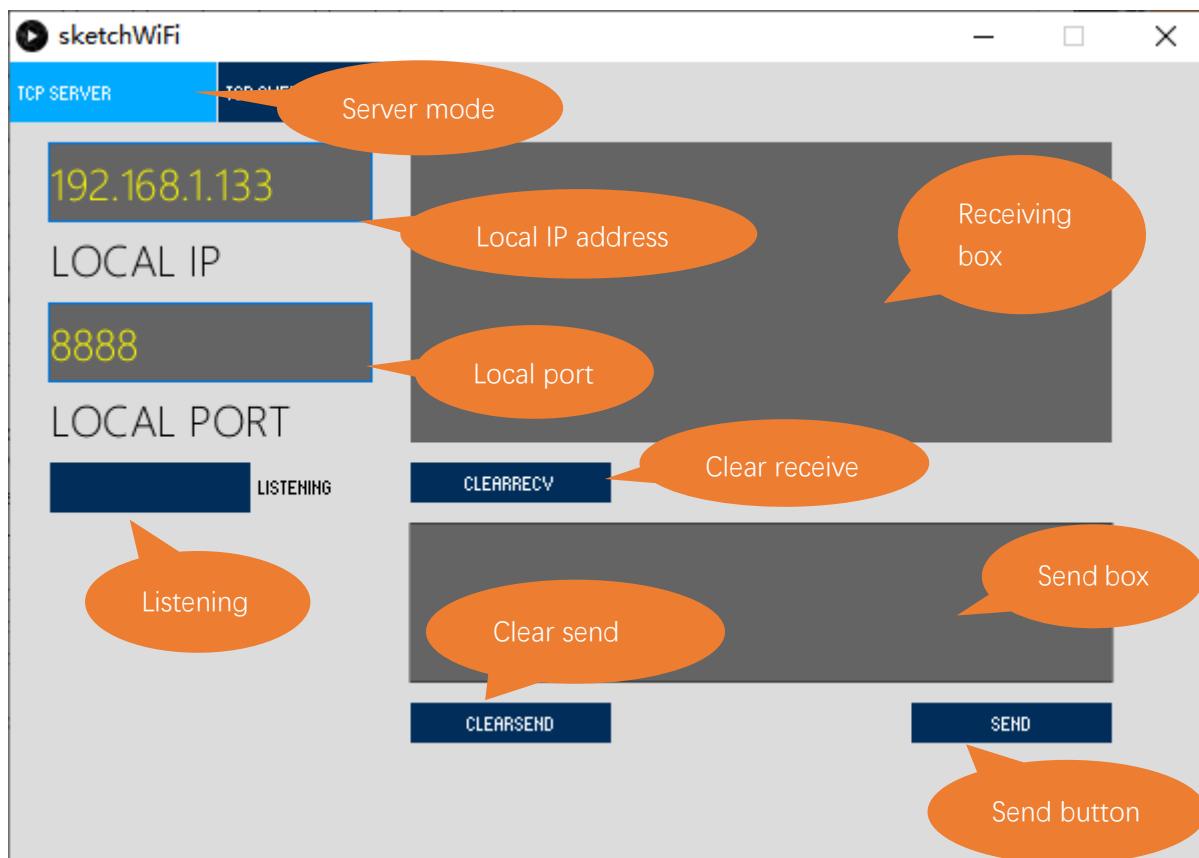
core	2020/1/17 12:16
java	2020/1/17 12:17
lib	2020/1/17 12:16
modes	2020/1/17 12:16
tools	2020/1/17 12:16
processing.exe	2020/1/17 12:16
processing-java.exe	2020/1/17 12:16
revisions.txt	2020/1/17 12:16

Use Server mode for communication

Open the “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_31.1_WiFiClient\sketchWiFi\sketchWiFi.pde**”, and click “Run”.

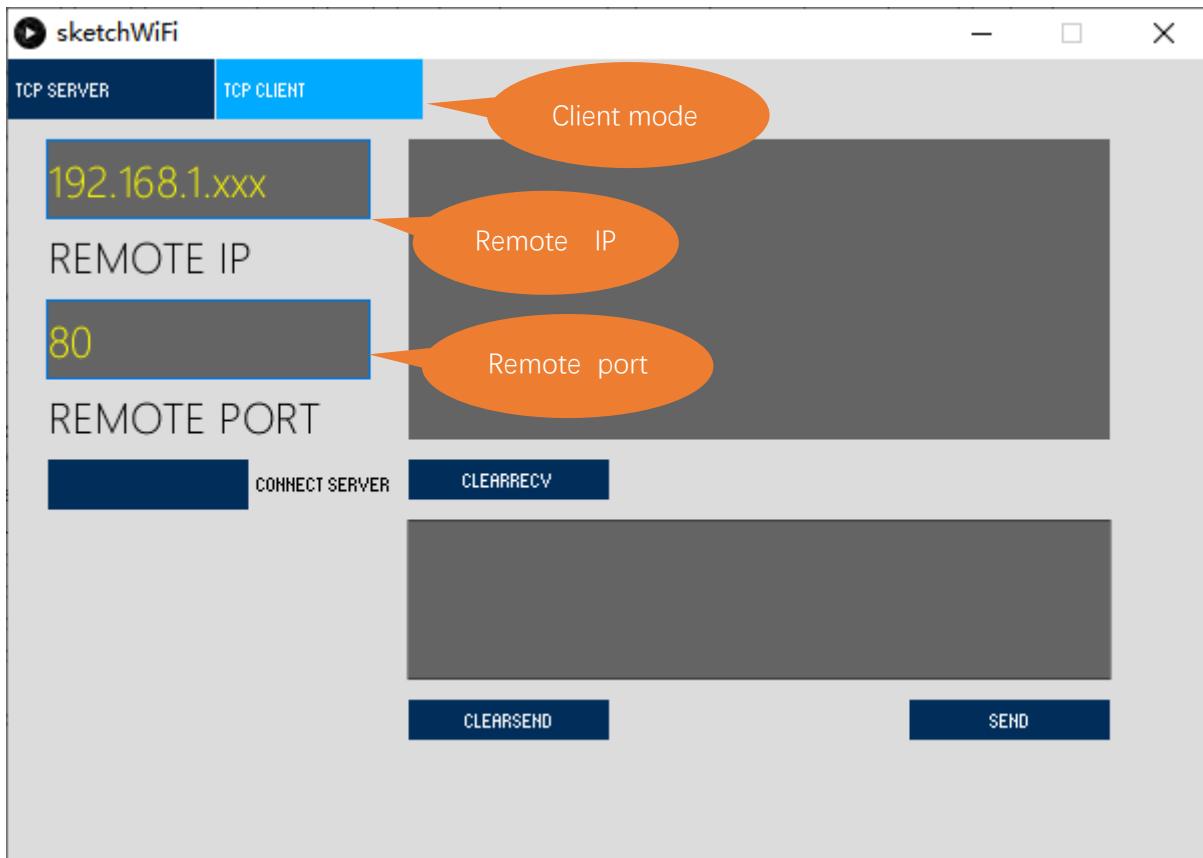


The new pop-up interface is as follows. If PICO W is used as client, select TCP SERVER mode for sketchWiFi.



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

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



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

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

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

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

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

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

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

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

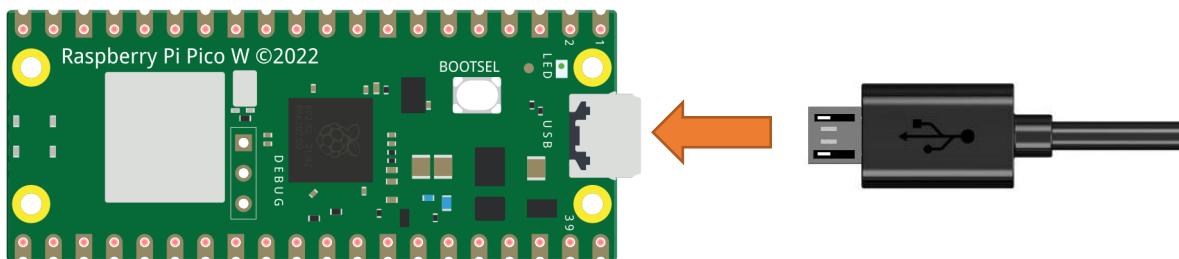
clear receive: clear out the content in the receiving text box

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

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

Circuit

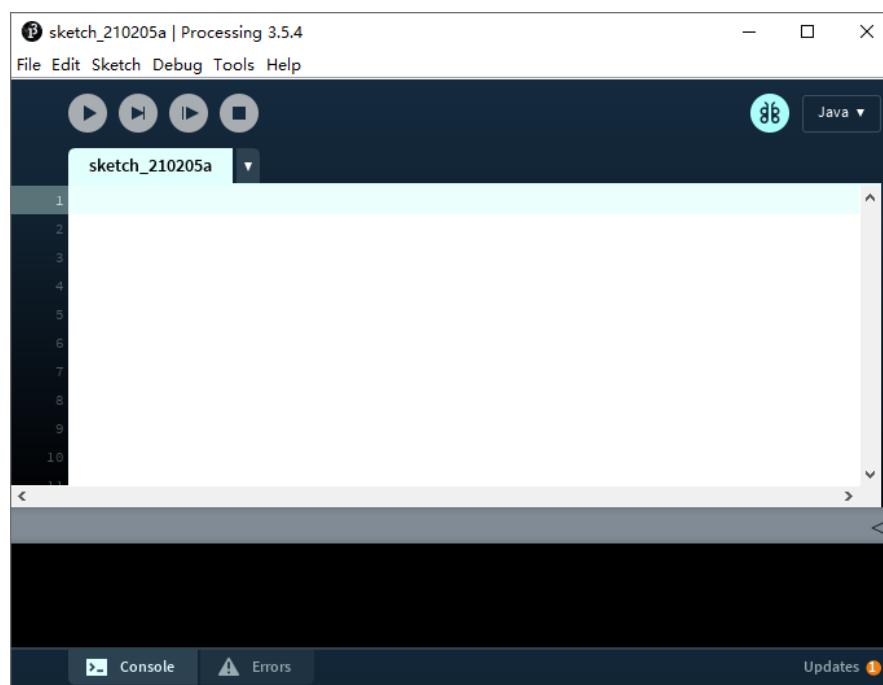
Connect Pico W to the computer using the USB cable.



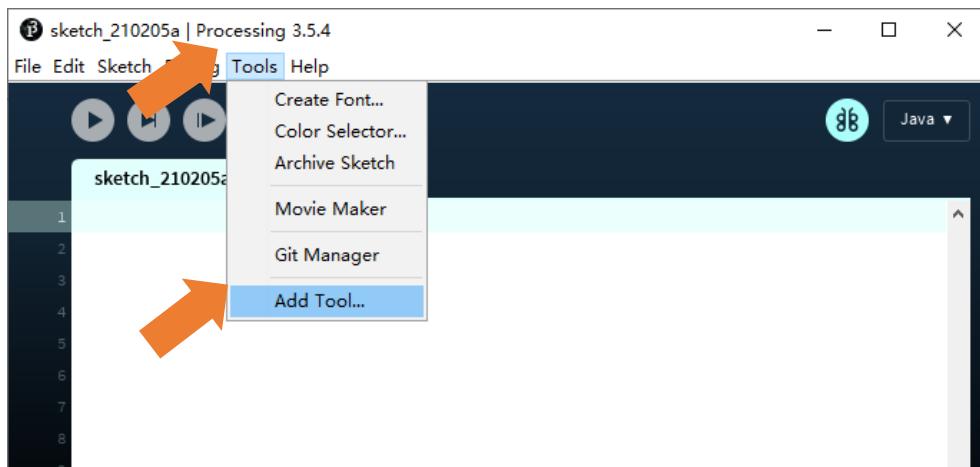
Sketch

If you have not installed “ControlIP5”, please follow the following steps to continue the installation, if you have installed, please skip this section.

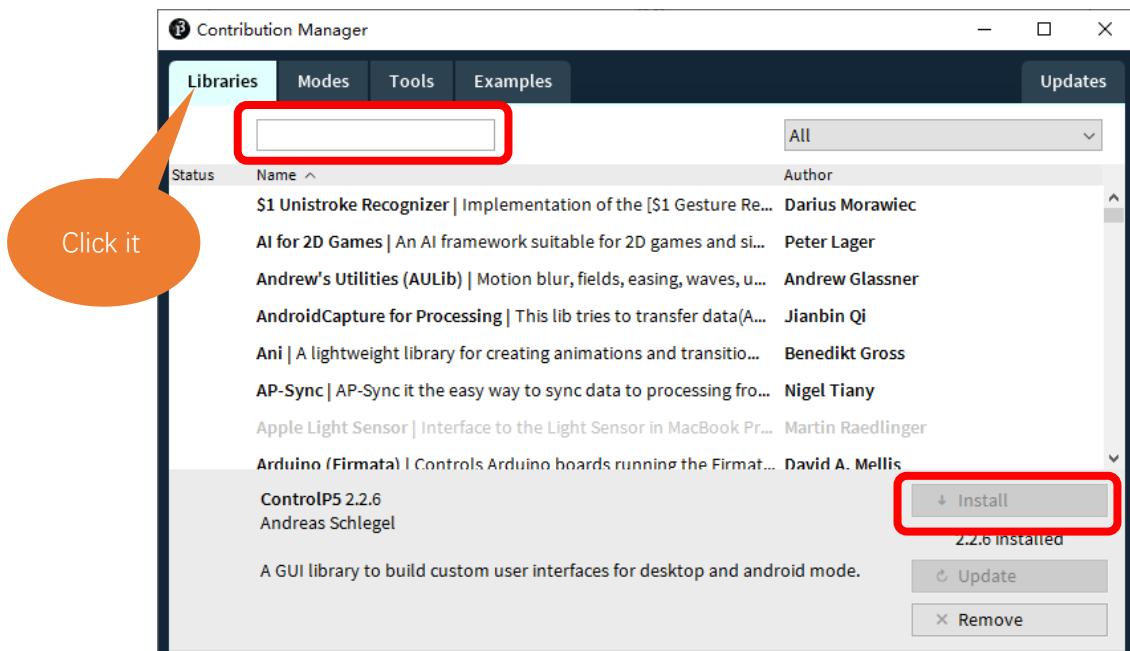
Open Processing.



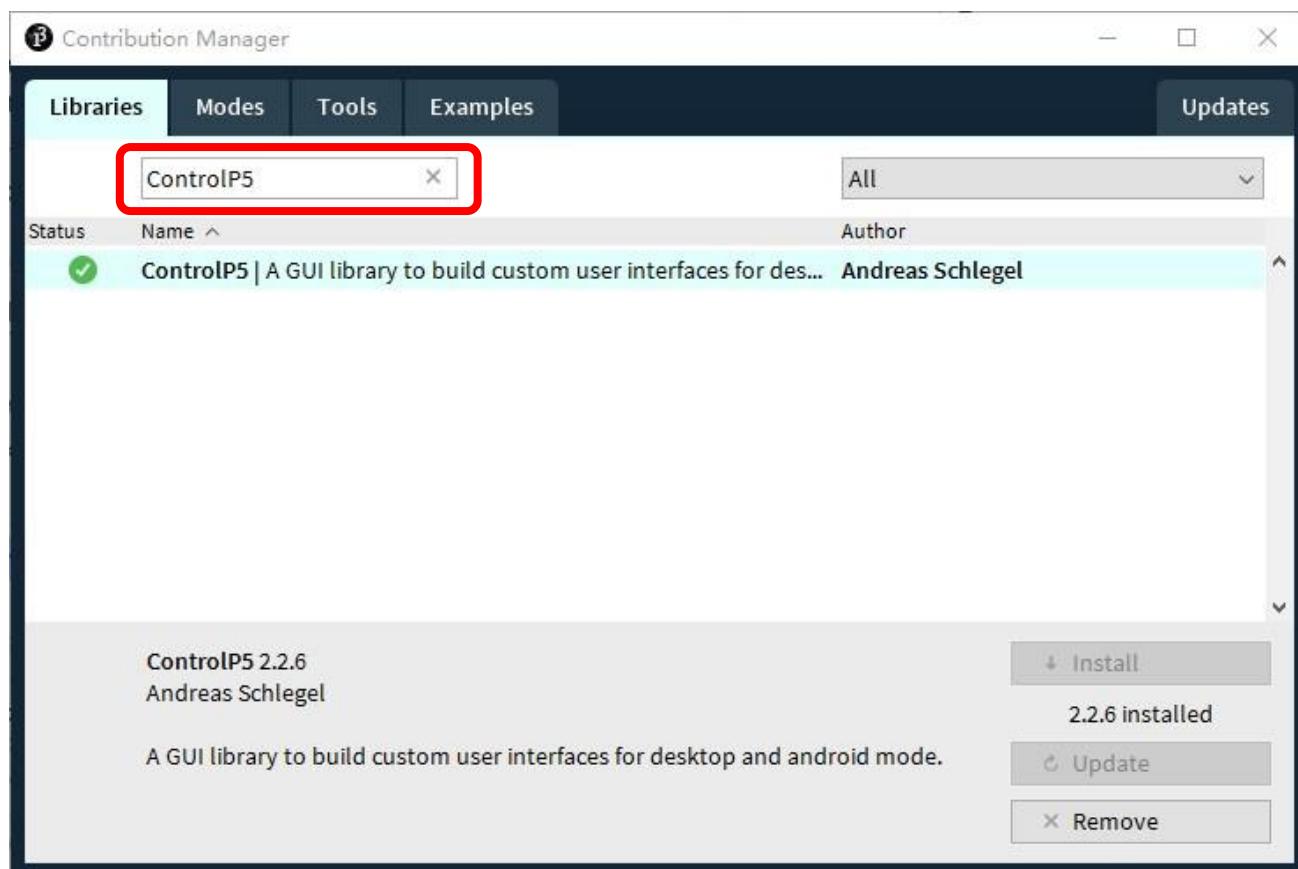
Click Add Tool under Tools.



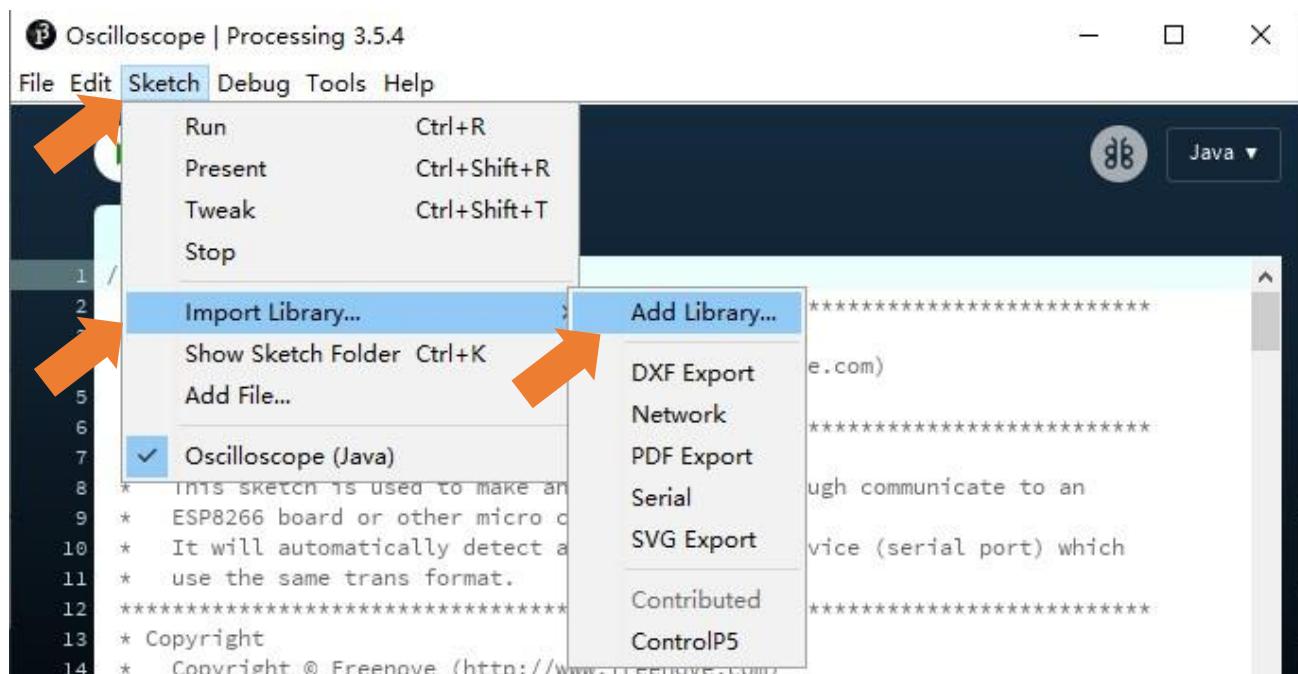
Select Libraries in the pop-up window.



Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

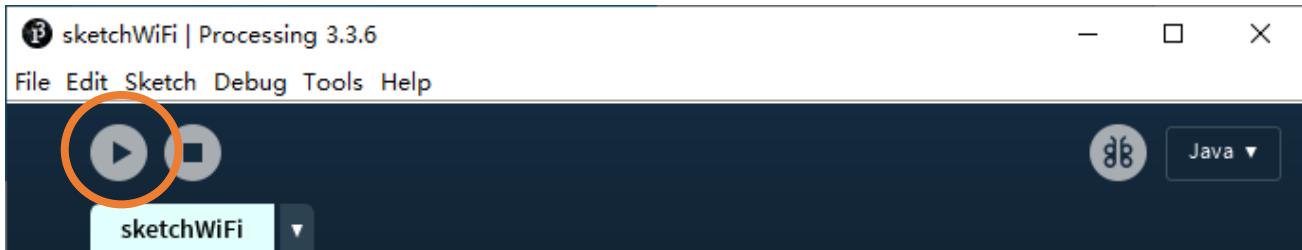


You can also click Add Library under 'Import Library' under 'Sketch'.

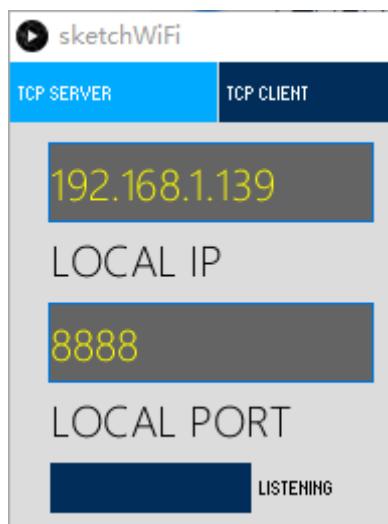


Sketch_31.1_As Client

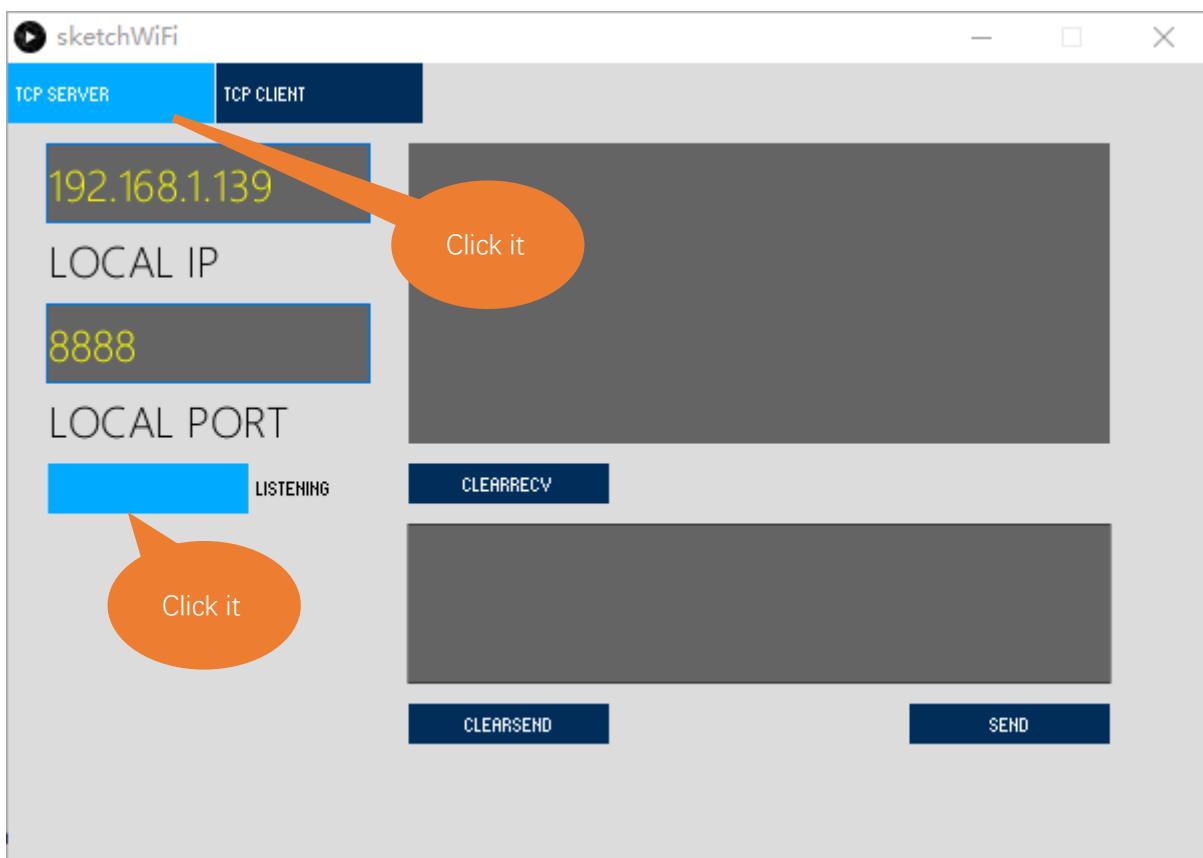
Before running the Sketch, please open “sketchWiFi.pde.” first, and click “Run”.



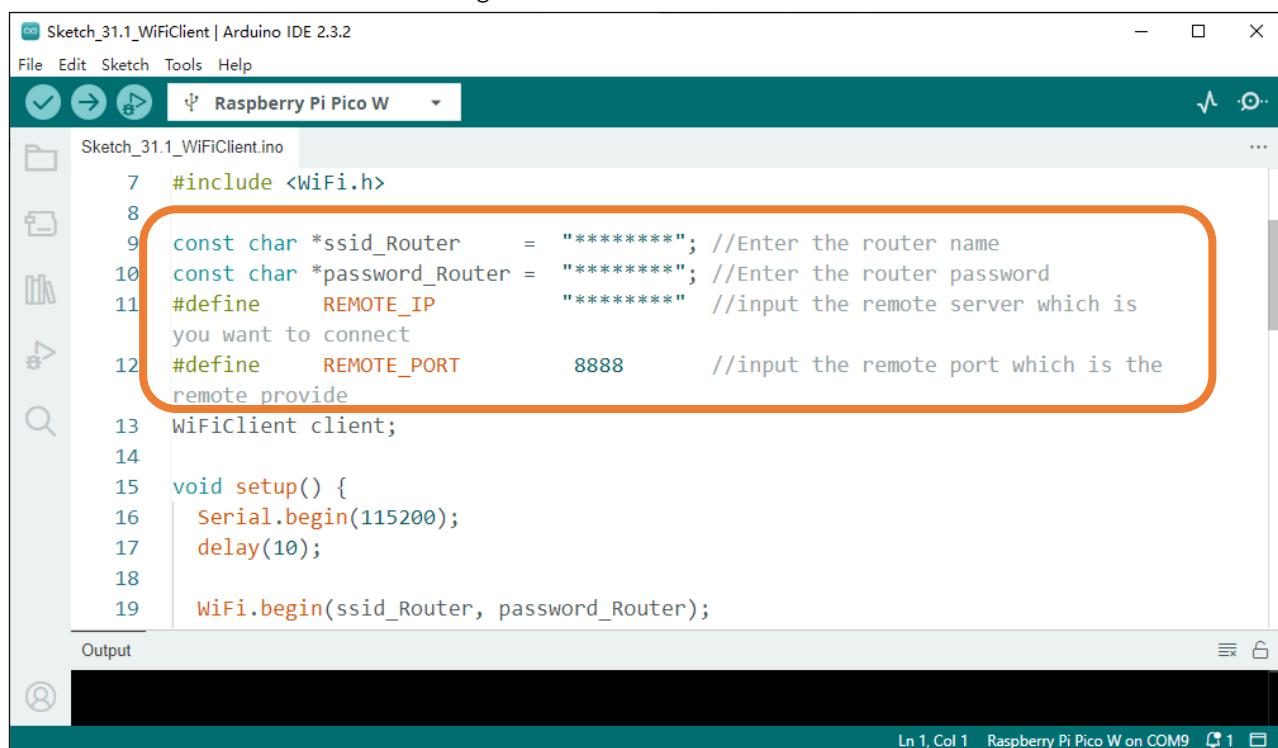
The newly pop up window will use the computer's IP address by default and open a data monitor port.



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



Next, open Sketch_31.1_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.



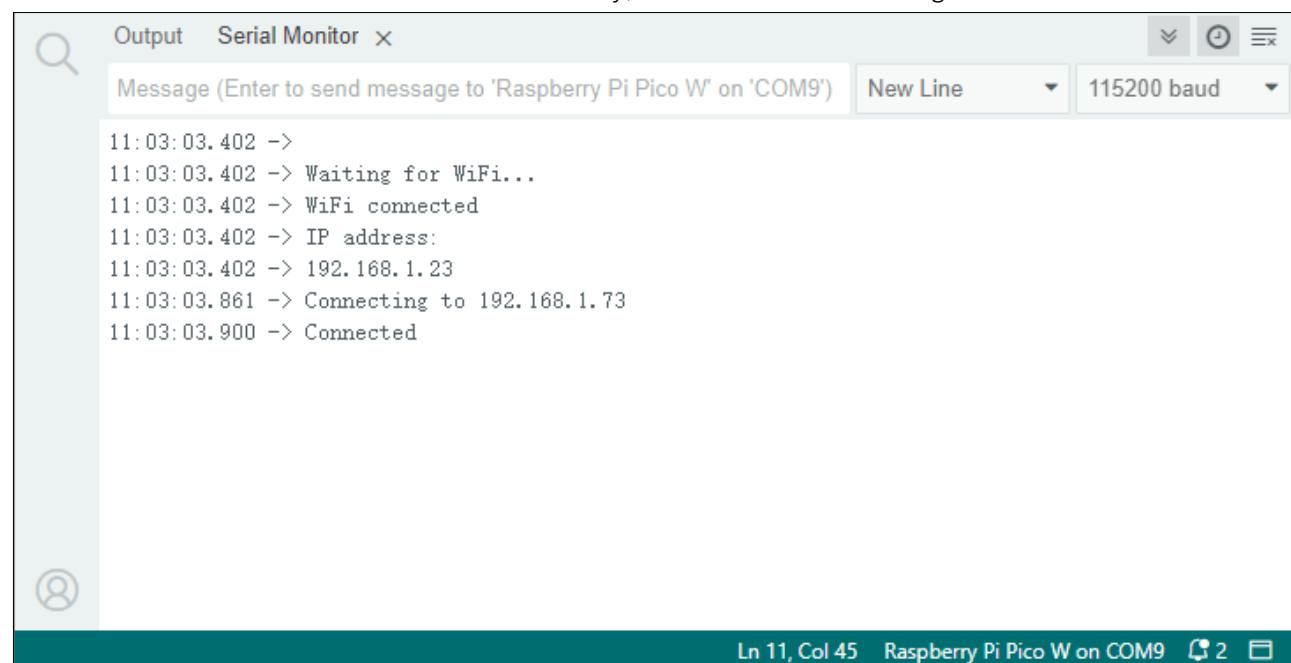
```

Sketch_31.1_WiFiClient | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico W
Sketch_31.1_WiFiClient.ino
7 #include <WiFi.h>
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11 #define    REMOTE_IP          "*****" //input the remote server which is
you want to connect
12 #define    REMOTE_PORT        8888     //input the remote port which is the
remote provide
13 WiFiClient client;
14
15 void setup() {
16   Serial.begin(115200);
17   delay(10);
18
19   WiFi.begin(ssid_Router, password_Router);

```

REMOTE_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE_IP is "192.168.1.73". Generally, by default, the ports do not need to change its value.

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



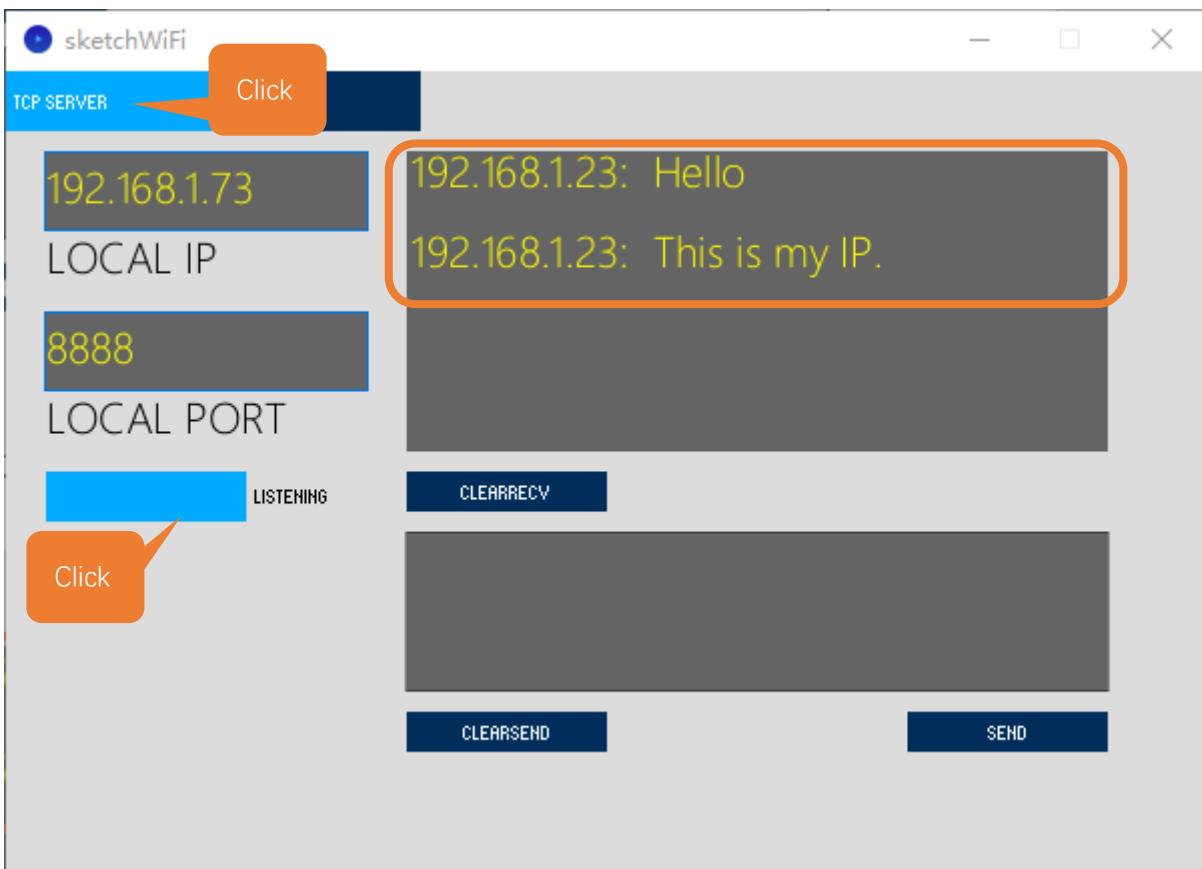
```

Output Serial Monitor ×
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud
11:03:03.402 ->
11:03:03.402 -> Waiting for WiFi...
11:03:03.402 -> WiFi connected
11:03:03.402 -> IP address:
11:03:03.402 -> 192.168.1.23
11:03:03.861 -> Connecting to 192.168.1.73
11:03:03.900 -> Connected

```



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



At this point, you can send data to Pico W through sketchWiFi. Pico W will send the received data back to sketchWiFi after receiving it.

The following is the program code:

```
1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5 #define    REMOTE_IP        "*****"   //input the remote server which is you want to connect
6 #define    REMOTE_PORT       8888      //input the remote port which is the remote provide
7 WiFiClient client;
8
9 void setup() {
10   Serial.begin(115200);
11   delay(10);
12
13   WiFi.begin(ssid_Router, password_Router);
14   Serial.print("\nWaiting for WiFi... ");
15   while (WiFi.status() != WL_CONNECTED) {
16     Serial.print(".");
17     delay(500);
18   }
19   Serial.println("");
20   Serial.println("WiFi connected");
21   Serial.println("IP address: ");
22   Serial.println(WiFi.localIP());
23   delay(500);
24
25   Serial.print("Connecting to ");
26   Serial.println(REMOTE_IP);
27
28   while (!client.connect(REMOTE_IP, REMOTE_PORT)) {
29     Serial.println("Connection failed.");
30     Serial.println("Waiting a moment before retrying... ");
31   }
32   Serial.println("Connected");
33   client.print("Hello\n");
34   client.print("This is my IP.\n");
35
36 void loop() {
37   if (client.available() > 0) {
38     delay(20);
39     //read back one line from the server
40     String line = client.readString();
41     Serial.println(REMOTE_IP + String(":") + line);
42   }
}
```

```

43   if (Serial.available() > 0) {
44     delay(20);
45     String line = Serial.readString();
46     client.print(line);
47   }
48   if (client.connected () == 0) {
49     client.stop();
50     WiFi.disconnect();
51   }
52 }
```

Add WiFi function header file.

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

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

```

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

Apply for the method class of WiFiClient.

```
7 WiFiClient client;
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect. Please disconnect the power supply and try again several times.

```

13 WiFi.begin(ssid_Router, password_Router);
14 Serial.print("\nWaiting for WiFi... ");
15 while (WiFi.status() != WL_CONNECTED) {
16   Serial.print(".");
17   delay(500);
18 }
```

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

```

28 while (!client.connect(REMOTE_IP, REMOTE_PORT)) {//Connect to Server
29   Serial.println("Connection failed.");
30   Serial.println("Waiting a moment before retrying... ");
31 }
32 Serial.println("Connected");
33 client.print("Hello\n");
```

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

```

37 if (client.available() > 0) {
38   delay(20);
39   //read back one line from the server
40   String line = client.readString();
41   Serial.println(REMOTE_IP + String(":") + line);
42 }
43 if (Serial.available() > 0) {
```

Any concerns? ✉ support@freenove.com

```
44     delay(20);  
45     String line = Serial.readString();  
46     client.print(line);  
47 }
```

If the server is disconnected, turn off WiFi of PICO W.

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

Reference

Class Client

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

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

ip, *host: ip address of target server

port: port number of target server

timeout: connection timeout

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

stop(): stop tcp connection

print(): send data to server connecting to client

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

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

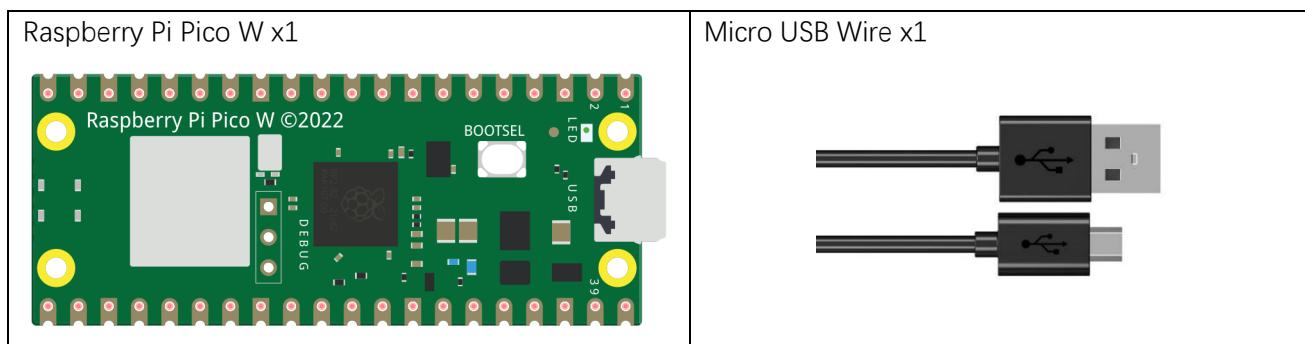
readString(): read string in receive buffer



Project 31.2 as Server

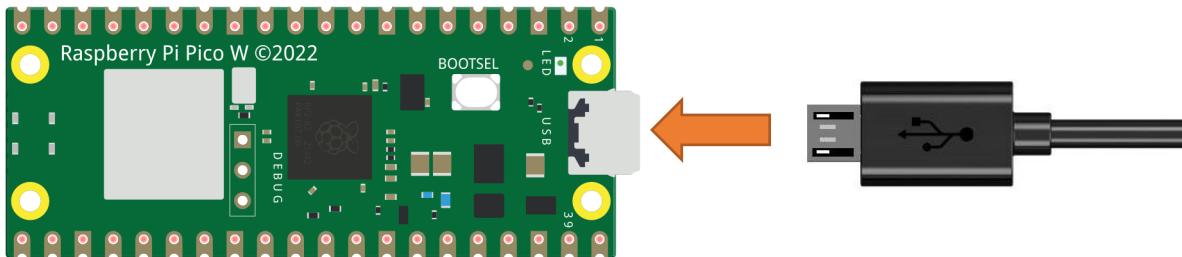
In this section, PICO W is used as a server to wait for the connection and communication of client on the same LAN.

Component List



Circuit

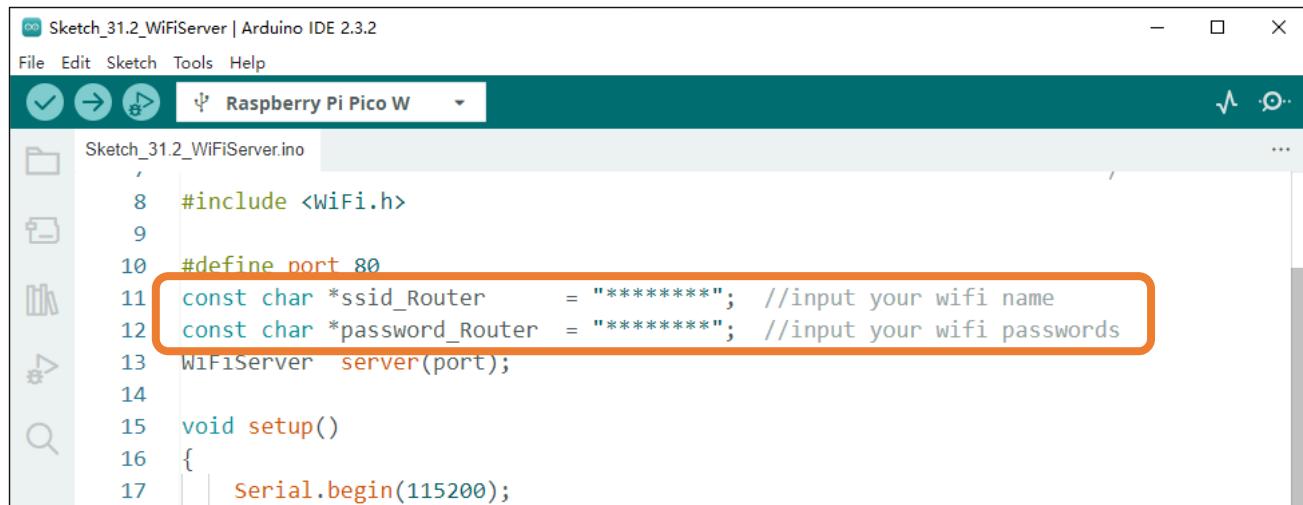
Connect Pico W to the computer using the USB cable.



Sketch

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

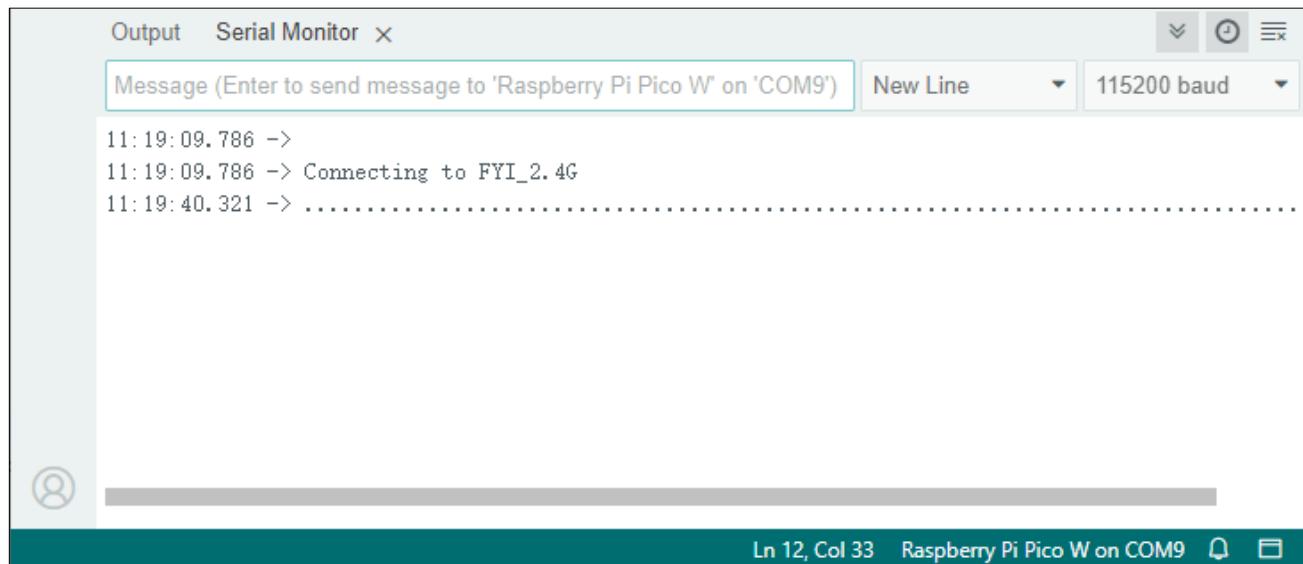
Sketch_31.2_As_Server



```
Sketch_31.2_WiFiServer | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_31.2_WiFiServer.ino
1
2   #include <WiFi.h>
3
4
5 #define port 80
6 const char *ssid_Router      = "*****"; //input your wifi name
7 const char *password_Router = "*****"; //input your wifi passwords
8 WiFiServer server(port);
9
10 void setup()
11 {
12     Serial.begin(115200);
```

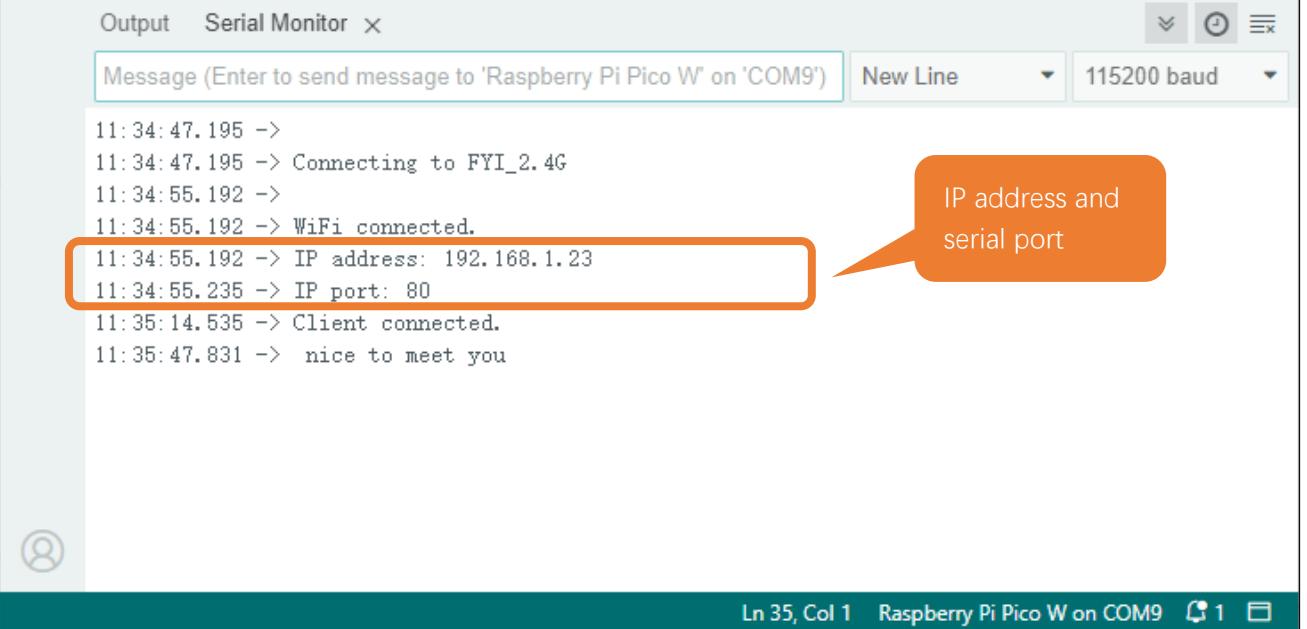
Compile and upload code to PICO W board, open the serial monitor and set the baud rate to 115200. Turn on server mode for PICO W, waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other.

If the Pico W fails to connect to router, please disconnect the power supply and try again several times.



Output	Serial Monitor
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')	New Line
11:19:09.786 ->	115200 baud
11:19:09.786 -> Connecting to FYI_2.4G	
11:19:40.321 ->	

Serial Monitor



Output Serial Monitor ×

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud

```

11:34:47.195 ->
11:34:47.195 -> Connecting to FYI_2.4G
11:34:55.192 ->
11:34:55.192 -> WiFi connected.
11:34:55.192 -> IP address: 192.168.1.23
11:34:55.235 -> IP port: 80
11:35:14.535 -> Client connected.
11:35:47.831 -> nice to meet you

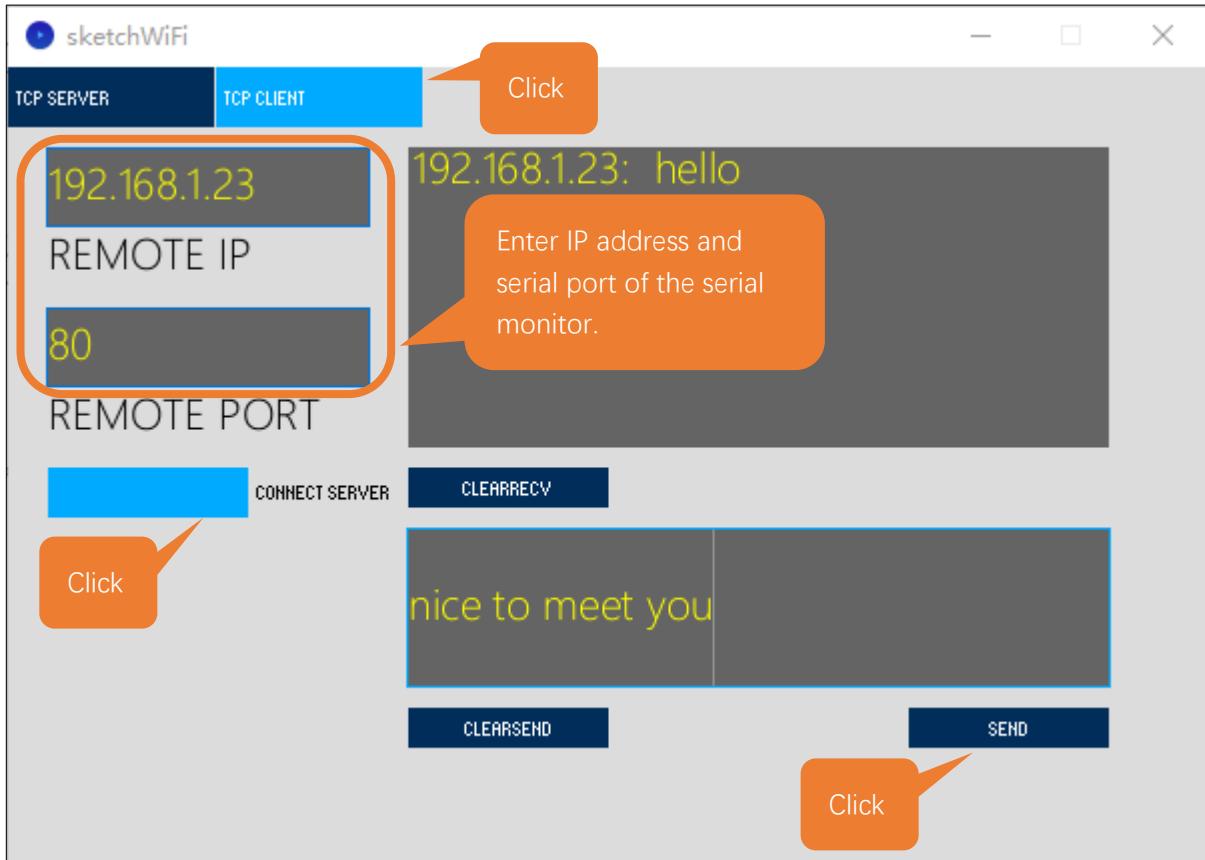
```

Ln 35, Col 1 Raspberry Pi Pico W on COM9 📲 1

Processing:

Open the “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_31.2_WiFiServer\sketchWiFi\sketchWiFi.pde**”.

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



The following is the program code:

```
1 #include <WiFi.h>
2
3 #define port 80
4 const char *ssid_Router      = "*****"; //input your wifi name
5 const char *password_Router  = "*****"; //input your wifi passwords
6 WiFiServer server(port);
7
8 void setup()
9 {
10    Serial.begin(115200);
11    Serial.printf("\nConnecting to ");
12    Serial.println(ssid_Router);
13    WiFi.disconnect();
14    WiFi.begin(ssid_Router, password_Router);
15    delay(1000);
16    while (WiFi.status() != WL_CONNECTED) {
17        delay(500);
18        Serial.print(".");
19    }
20    Serial.println("");
21    Serial.println("WiFi connected.");
22    Serial.print("IP address: ");
23    Serial.println(WiFi.localIP());
24    Serial.printf("IP port: %d\n", port);
25    server.begin(port);
26 }
27
28 void loop() {
29    WiFiClient client = server.available();           // listen for incoming clients
30    if (client) {                                     // if you get a client
31        Serial.println("Client connected.");
32        while (client.connected()) {                  // loop while the client's connected
33            if (client.available()) {                 // if there's bytes to read from the
34                Serial.println(client.readStringUntil('\n'));// print it out the serial monitor
35                while (client.read() > 0);               // clear the wifi receive area cache
36            }
37            if (Serial.available()) {                  // if there's bytes to read from the
38                client.print(Serial.readStringUntil('\n'));// print it out the client.
39                while (Serial.read() > 0);              // clear the wifi receive area cache
40            }
41        }
42    }
43 }
```

```

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

Apply for method class of WiFiServer.

6	<code>WiFiServer server(port);</code>	//Apply for a Server object whose port number is 80
---	---------------------------------------	---

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please disconnect the power supply and try again several times.

```

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

Print out the IP address and port number of PICO W.

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

Turn on server mode of PICO W.

25	<code>server.begin();</code>	//Turn ON PICO W as Server mode
----	------------------------------	---------------------------------

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

```

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

Reference

Class Server

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

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

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

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

begin(port): start the TCP Server.

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

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

nodelay: true stands for forbidden Nagle algorithm.

close(): close tcp connection.

stop(): stop tcp connection.



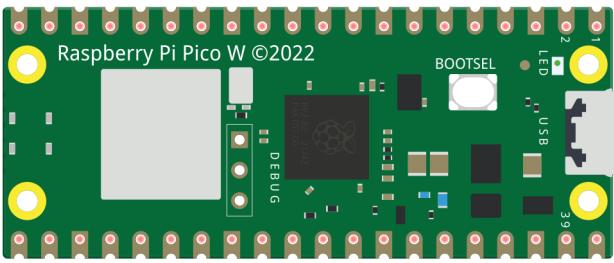
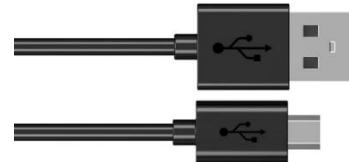
Chapter 32 Control LED with Web (Only for Pico W)

In this chapter, we will use PICO W to make a simple smart home. We will learn how to control LED lights through web pages.

Project 32.1 Control the LED with Web

In this project, we need to build a Web Service and then use PICO W to control the LED through the Web browser of the phone or PC. Through this example, you can remotely control the appliances in your home to achieve smart home.

Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	

Component knowledge

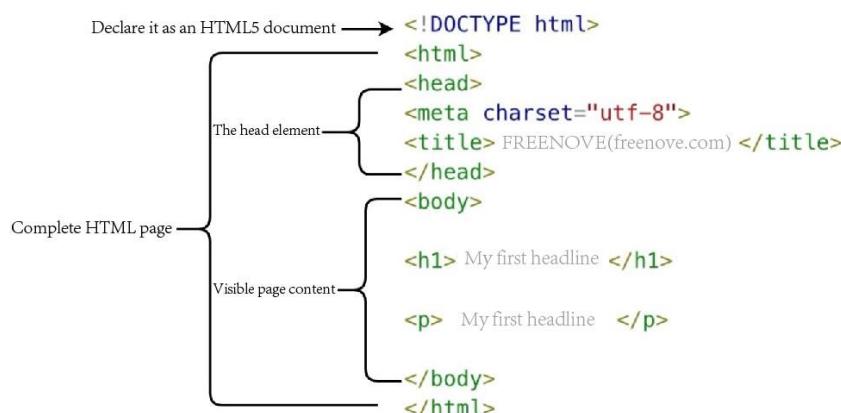
HTML

Hypertext Markup Language (HTML) is a standard Markup Language for creating web pages. It includes a set of tags that unify documents on the network and connect disparate Internet resources into a logical whole. HTML text is descriptive text composed of HTML commands that describe text, graphics, animations, sounds, tables, links, etc. The extension of the HTML file is HTM or HTML. Hypertext is a way to organize information. It uses hyperlinks to associate words and charts in Text with other information media. These related information media may be in the same Text, other files, or files located on a remote computer. This way of organizing information connects the information resources distributed in different places, which is convenient for people to search and retrieve information.

The nature of the Web is hypertext Markup Language (HTML), which can be combined with other Web technologies (e.g., scripting languages, common gateway interfaces, components, etc.) to create powerful Web pages. Thus, Hypertext Markup Language (HTML) is the foundation of World Wide Web (Web) programming, that is, the World Wide Web is based on hypertext. Hypertext Markup Language is called hypertext Markup language because the text contains so-called "hyperlink" points.

You can build your own WEB site using HTML, which runs on the browser and is parsed by the browser.

Example analysis is shown in the figure below:



<!DOCTYPE html>: Declare it as an HTML5 document

<html>: Is the root element of an HTML page

<head>: Contains meta data for the document, such as `<meta charset="utf-8">`; Define the web page encoding format to UTF-8.

<title>: Notes the title of the document

<body>: Contains visible page content

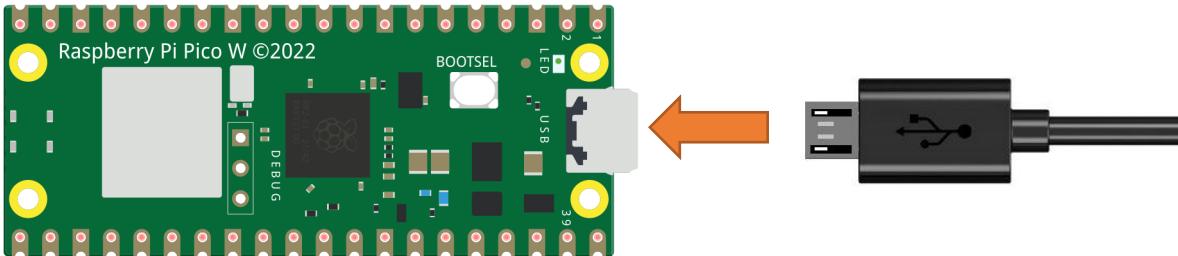
<h1>: Define a big heading

<p>: Define a paragraph

For more information, please visit: <https://developer.mozilla.org/en-US/docs/Web/HTML>

Circuit

Connect Pico W to the computer using the USB cable.



Sketch

Sketch_32.1_Control_the_LED_with_Web

 A screenshot of the Arduino IDE interface. The title bar says "Sketch_32.1_Control_the_LED_with_Web | Arduino IDE 2.3.2". The menu bar includes File, Edit, Sketch, Tools, and Help. The central area shows the code for "Sketch_32.1_Control_the_LED_with_Web.ino". The code includes comments for network credentials and a web server setup. A callout bubble with an orange border and arrow points to the lines of code for "ssid" and "password", which are both set to "*****". The text inside the bubble reads "Enter the correct Router name and password.".

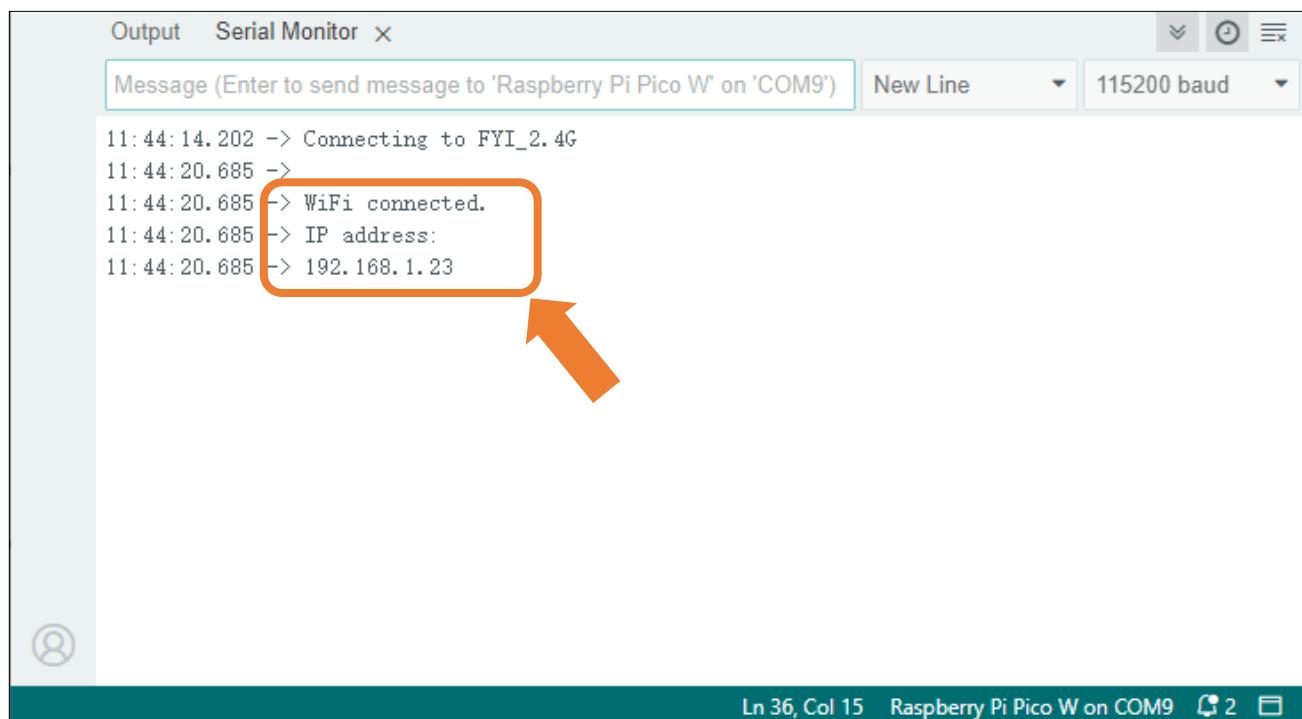
```

11 #include <WiFi.h>
12
13 // Replace with your network credentials
14 const char* ssid      = "*****";
15 const char* password = "*****";
16
17 // Set web server port number to 80
18 WiFiServer server(80);
19 // Variable to store the HTTP request
20 String header;
21 // Auxiliar variables to store the current output state
22 String PIN_LEDstate = "OFF";
23
24 // Current time
25 unsigned long currentTime = millis();
26 // Previous time
27 unsigned long previousTime = 0;
28 // Define timeout time in milliseconds (example: 2000ms = 2s)
29 const long timeoutTime = 2000;
30
31 void setup() {
32   Serial.begin(115200);
33   // Initialize the output variables as outputs
34   pinMode(LED_BUILTIN, OUTPUT);
35   digitalWrite(LED_BUILTIN, LOW);
36
37   // Connect to Wi-Fi network with SSID and password
38   Serial.print("Connecting to ");

```

Ln 1, Col 1 × No board selected

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



The screenshot shows the 'Serial Monitor' window with the title 'Output' and 'Serial Monitor'. The message area contains the following text:

```
11:44:14.202 -> Connecting to FYI_2.4G  
11:44:20.685 ->  
11:44:20.685 -> WiFi connected.  
11:44:20.685 -> IP address:  
11:44:20.685 -> 192.168.1.23
```

An orange arrow points from the text 'As shown in the following figure:' to the line '11:44:20.685 -> WiFi connected.' and the line '11:44:20.685 -> IP address: 192.168.1.23'.

When PICO W successfully connects to "ssid_Router", serial monitor will print out the IP address assigned to PICO W by the router. Access <http://192.168.1.23> in a computer browser on the LAN. [As shown in the following figure:](#)



You can click the corresponding button to control the LED on and off.



The following is the program code:

```
1 #include <WiFi.h>
2
3 // Replace with your network credentials
4 const char* ssid      = "*****";
5 const char* password = "*****";
6
7 // Set web server port number to 80
8 WiFiServer server(80);
9 // Variable to store the HTTP request
10 String header;
11 // Auxiliar variables to store the current output state
12 String PIN_LEDState = "OFF";
13
14 // Current time
15 unsigned long currentTime = millis();
16 // Previous time
17 unsigned long previousTime = 0;
18 // Define timeout time in milliseconds (example: 2000ms = 2s)
19 const long timeoutTime = 2000;
20
21 void setup() {
22     Serial.begin(115200);
23     // Initialize the output variables as outputs
24     pinMode(LED_BUILTIN, OUTPUT);
25     digitalWrite(LED_BUILTIN, LOW);
26
27     // Connect to Wi-Fi network with SSID and password
28     Serial.print("Connecting to ");
29     Serial.println(ssid);
30     WiFi.begin(ssid, password);
31     while (WiFi.status() != WL_CONNECTED) {
32         delay(500);
33         Serial.print(".");
34     }
35     // Print local IP address and start web server
36     Serial.println("");
37     Serial.println("WiFi connected.");
38     Serial.println("IP address: ");
39     Serial.println(WiFi.localIP());
40     server.begin();
41 }
42 void loop() {
43     WiFiClient client = server.available(); // Listen for incoming clients
```

Any concerns? ✉ support@freenove.com

```
44  if (client) {                                // If a new client connects,
45      Serial.println("New Client.");           // print a message out in the serial port
46      String currentLine = "";                 // make a String to hold incoming data from the
47      client
48      currentTime = millis();
49      previousTime = currentTime;
50      while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while
the client's connected
51          currentTime = millis();
52          if (client.available()) { // if there's bytes to read from the client,
53              char c = client.read(); // read a byte, then
54              Serial.write(c);       // print it out the serial monitor
55              header += c;
56              if (c == '\n') { // if the byte is a newline character
57                  // if the current line is blank, you got two newline characters in a row.
58                  // that's the end of the client HTTP request, so send a response:
59                  if (currentLine.length() == 0) {
60                      // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
61                      // and a content-type so the client knows what's coming, then a blank line:
62                      client.println("HTTP/1.1 200 OK");
63                      client.println("Content-type:text/html");
64                      client.println("Connection: close");
65                      client.println();
66                      // turns the GPIOs on and off
67                      if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
68                          Serial.println("LED_BUILTIN ON");
69                          PIN_LEDState = "ON";
70                          digitalWrite(LED_BUILTIN, HIGH);
71                      } else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
72                          Serial.println("LED_BUILTIN OFF");
73                          PIN_LEDState = "OFF";
74                          digitalWrite(LED_BUILTIN, LOW);
75                      }
76                      // Display the HTML web page
77                      client.println("<!DOCTYPE html><html>");
78                      client.println("<head> <title>Pico W Web Server</title> <meta name=\"viewport\""
content="width=device-width, initial-scale=1">");
79                      client.println("<link rel=\"icon\" href=\"data:, \">");
80                      // CSS to style the on/off buttons
81                      // Feel free to change the background-color and font-size attributes to fit your
preferences
82                      client.println("<style>html {font-family: Helvetica; display:inline-block; margin:
0px auto; text-align: center;}</style>");
```



```

83         client.println(".button{background-color: #4286f4; display: inline-block; border:
84             none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
85             30px; margin: 2px; cursor: pointer;}");
86         client.println(".button2{background-color: #4286f4;display: inline-block; border:
87             none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
88             30px; margin: 2px; cursor: pointer;}</style></head>");
89         // Web Page Heading
90         client.println("<body><h1>Pico W Web Server</h1>");
91         client.println("<p>GPIO state: " + PIN_LEDState + "</p>"); 
92         client.println("<p><a href=\"/LED_BUILTIN/ON\"><button class=\"button
button2\">ON</button></a></p>"); 
93         client.println("<p><a href=\"/LED_BUILTIN/OFF\"><button class=\"button
button2\">OFF</button></a></p>"); 
94         client.println("</body></html>"); 
95         // The HTTP response ends with another blank line
96         client.println(); 
97         // Break out of the while loop
98         break;
99     } else { // if you got a newline, then clear currentLine
100        currentLine = "";
101    }
102  }
103  // Clear the header variable
104  header = "";
105  // Close the connection
106  client.stop();
107  Serial.println("Client disconnected.");
108  Serial.println("");
109 }
110 }
```

Include the WiFi Library header file of PICO W.

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

Enter correct router name and password.

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

Set PICO W in Station mode and connect it to your router.

```
30 WiFi.begin(ssid, password);
```

Check whether PICO W has connected to router successfully every 0.5s.

```
31 while (WiFi.status() != WL_CONNECTED) {
32     delay(500);
```

Any concerns? ✉ support@freenove.com

```
33     Serial.print(".");
34 }
```

Serial monitor prints out the IP address assigned to PICO W.

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

Click the button on the web page to control the LED light on and off.

```
65 // turns the GPIOs on and off
66 if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
67     Serial.println("LED_BUILTIN ON");
68     PIN_LEDState = "ON";
69     digitalWrite(LED_BUILTIN, HIGH);
70 } else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
71     Serial.println("LED_BUILTIN OFF");
72     PIN_LEDState = "OFF";
73     digitalWrite(LED_BUILTIN, LOW);
74 }
```

Chapter 33 Bluetooth (Only for Pico W)

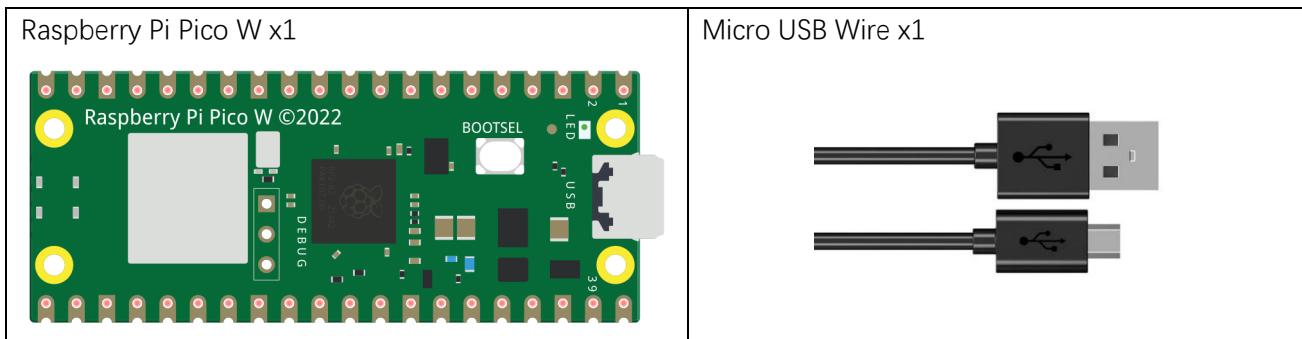
In June 2023, Raspberry Pi Official has updated to add Bluetooth support to the Pico W.

Pico W's Bluetooth 5.2 supports Bluetooth Classic and Bluetooth Low Energy (BLE) functionality. At the beginning of this chapter, we will learn the Pico W's Bluetooth function.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

Project 33.1 Bluetooth Passthrough

Component List



Component knowledge

Pico W's wireless functionality is provided by the Infineon CYW43439 device, which contains a 2.4 GHz radio providing both 802.11n Wi-Fi and Bluetooth 5.2, supporting Bluetooth Classic and Bluetooth Low Energy (BLE) functionality.

For simple data transfer, there are two modes:

Master mode

A device works in master mode can connect to one or more slave devices.

We can search and select the slave devices nearby to connect.

When a device initiates a connection request in master mode, it requires information about other Bluetooth devices, including their addresses and pairing keys.

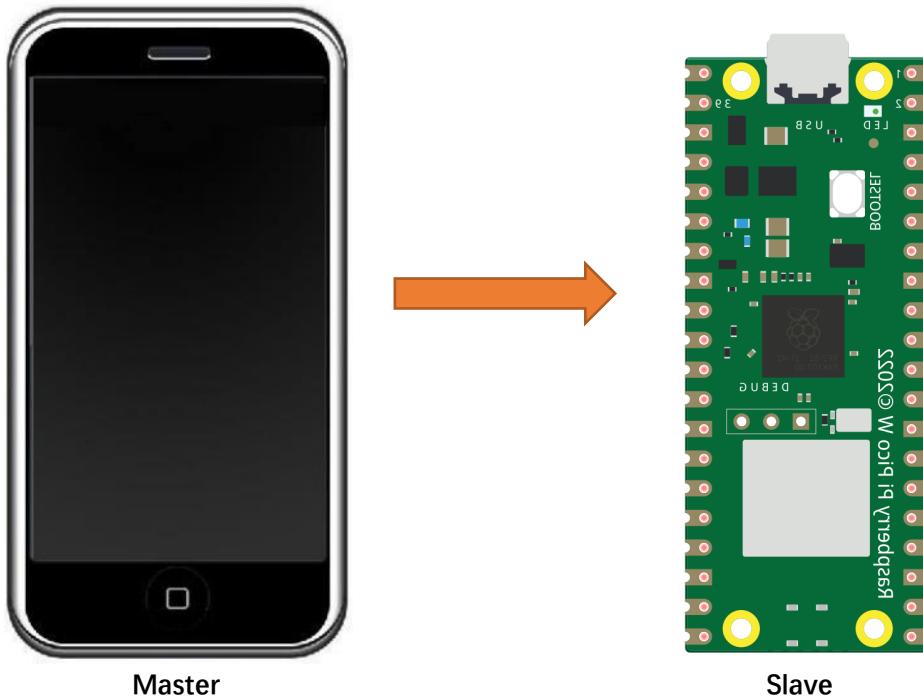
Once the devices are paired, a direct connection can be established.

Slave mode

A Bluetooth module operating in slave mode can only receive connection requests from a master device and cannot actively initiate connections.

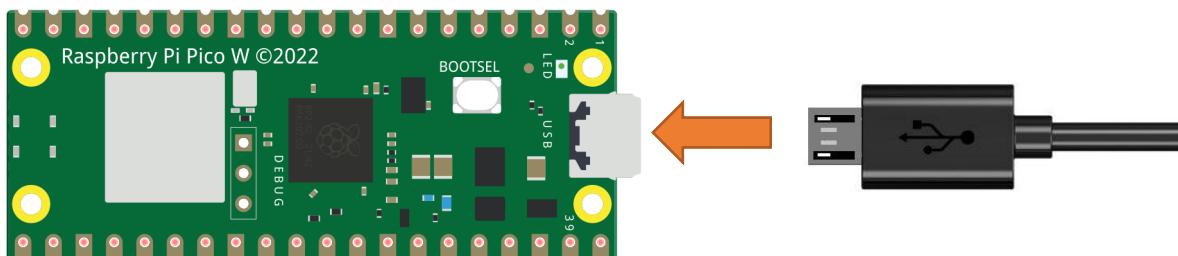
After establishing a connection with a master device, it can either send or receive data.

Bluetooth devices can interact with each other, with one in master mode and the other in slave mode. During data communication, the master device searches for and selects nearby devices to connect with. Once a connection is made, data can be exchanged between the devices. In the case of data exchange between a smartphone and a Raspberry Pi Pico W, the smartphone typically operates in master mode, while the Raspberry Pi Pico W functions in slave mode.



Circuit

Connect Pico W to the computer using the USB cable.

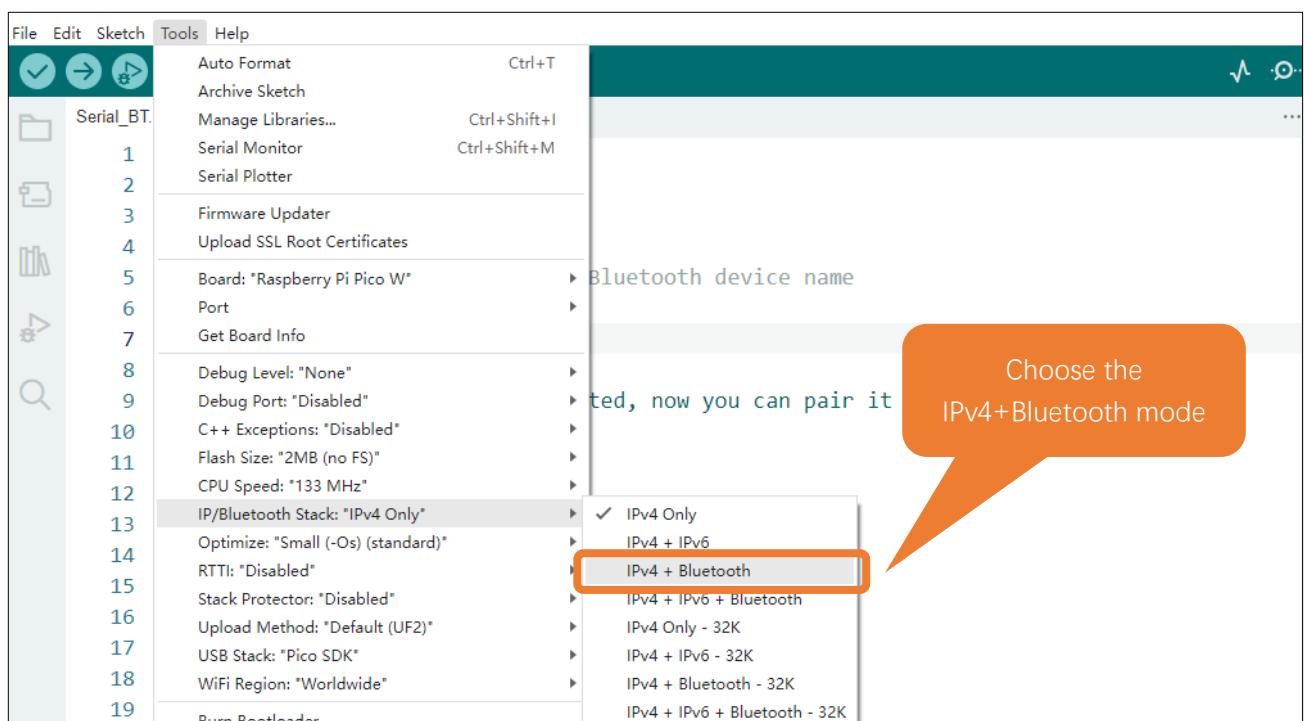
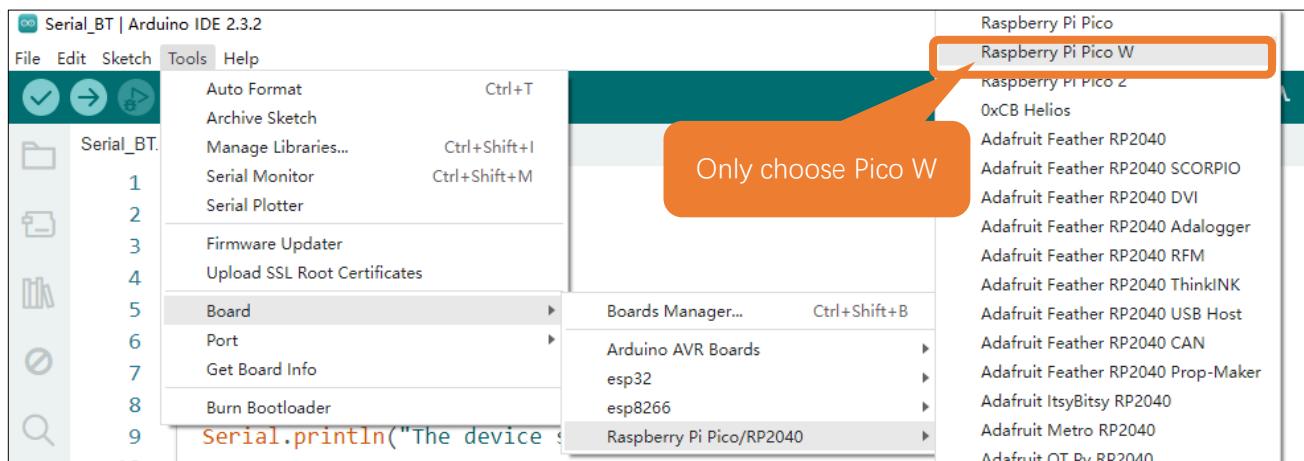




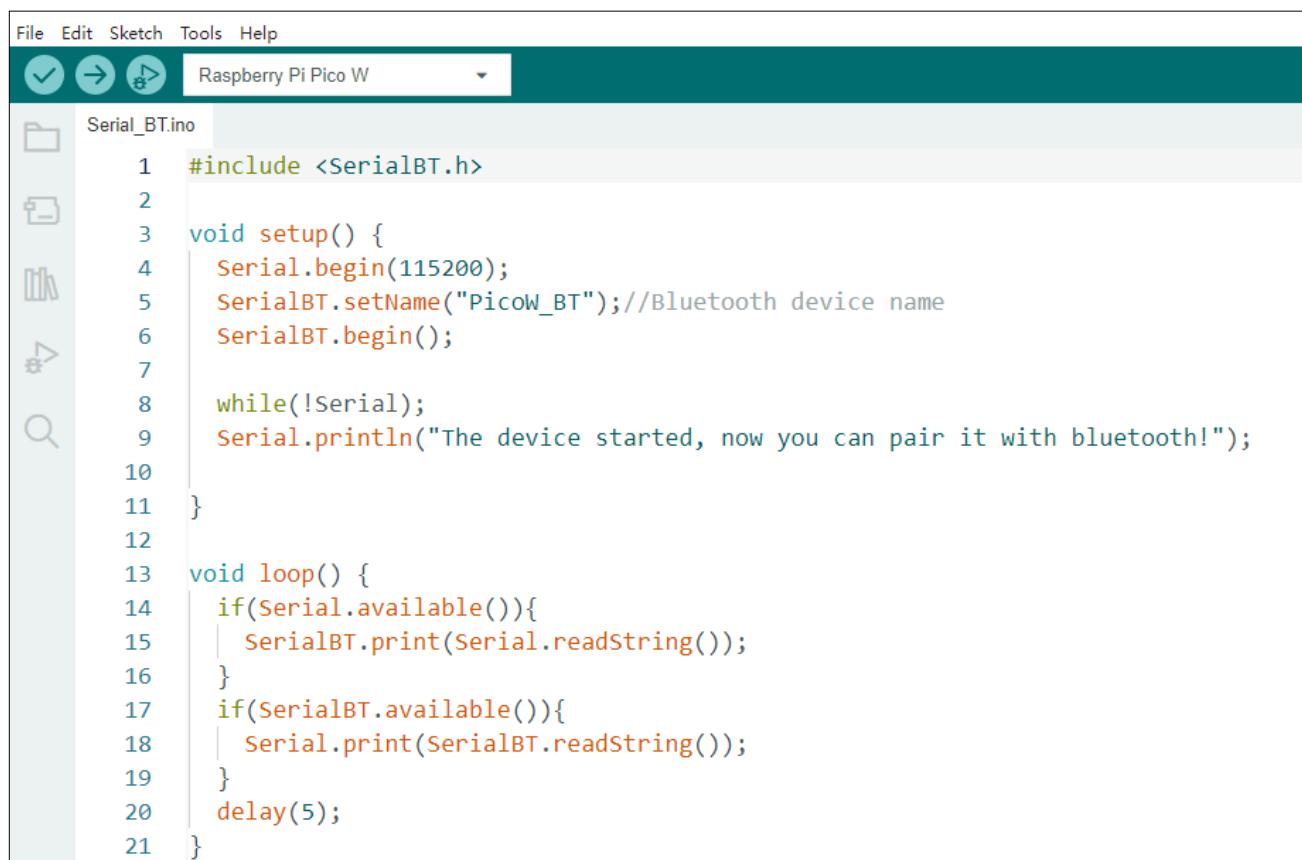
Sketch

How to enable Pico W's Bluetooth

In default setting, Pico W only enables the ipv4 function of its WiFi. To enable Bluetooth, please refer to the following steps:

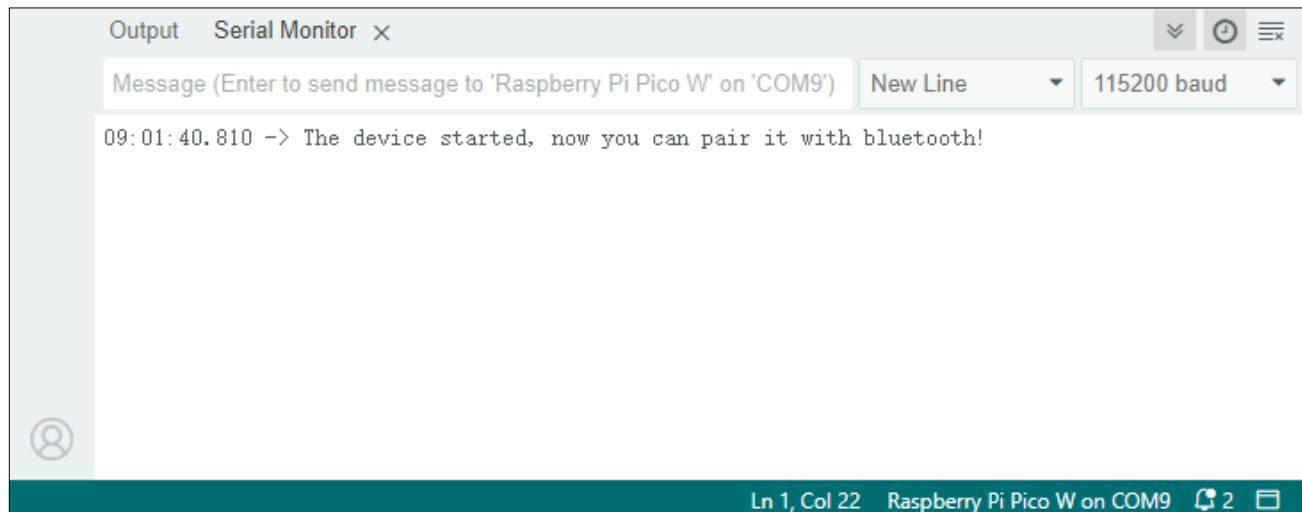


Sketch_33.1_Serial_BT



```
File Edit Sketch Tools Help
Serial_BT.ino
1 #include <SerialBT.h>
2
3 void setup() {
4     Serial.begin(115200);
5     SerialBT.setName("PicoW_BT"); //Bluetooth device name
6     SerialBT.begin();
7
8     while(!Serial);
9     Serial.println("The device started, now you can pair it with bluetooth!");
10 }
11
12 void loop() {
13     if(Serial.available()){
14         SerialBT.print(Serial.readString());
15     }
16     if(SerialBT.available()){
17         Serial.print(SerialBT.readString());
18     }
19     delay(5);
20 }
```

Upload the sketch to Pico W and open the serial monitor, set the baud rate to 115200. When you see the message as shown below, it indicates that, the Bluetooth of Pico W is ready.



Output Serial Monitor X

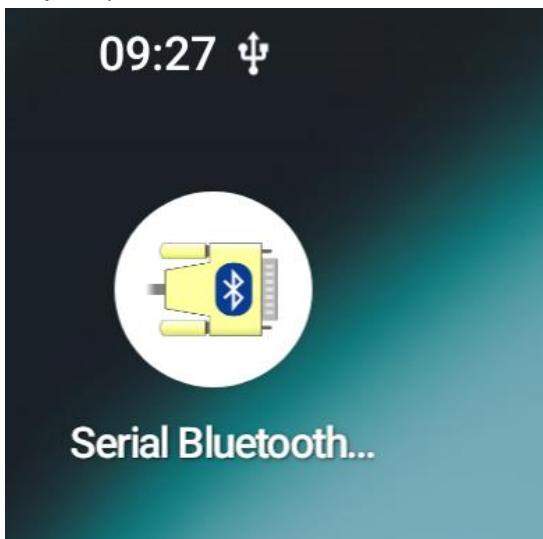
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud

09:01:40.810 -> The device started, now you can pair it with bluetooth!

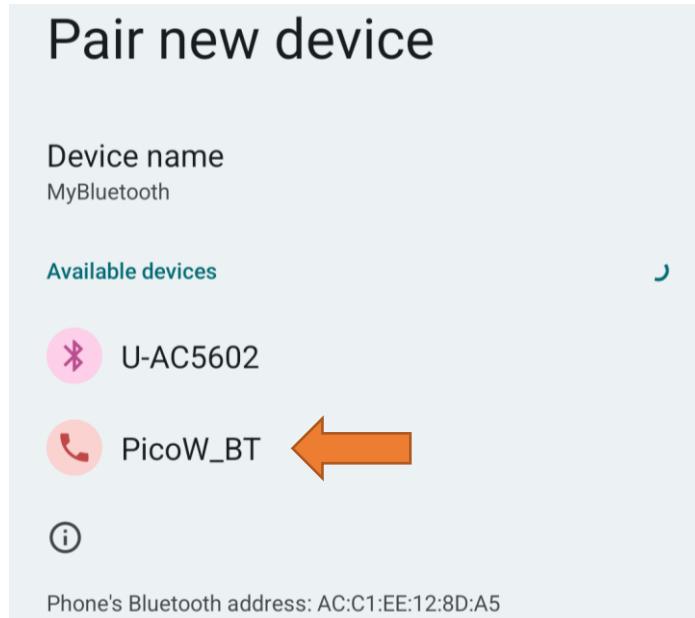
Ln 1, Col 22 Raspberry Pi Pico W on COM9 2



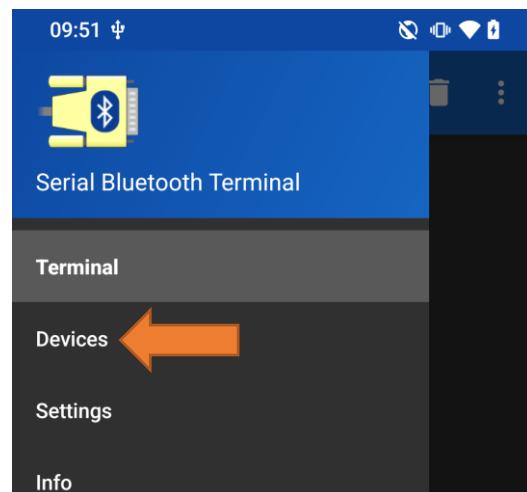
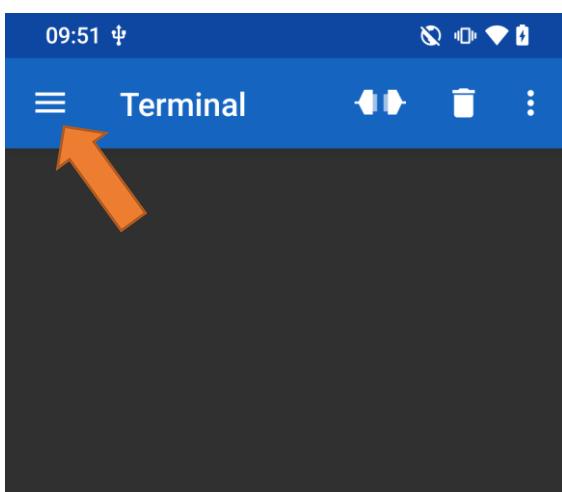
Please make sure the Bluetooth of your phone is turn ON and the App Serial Bluetooth Terminal has installed on your phone.



Click Pair new device and select "PicoW_BT" to connect.

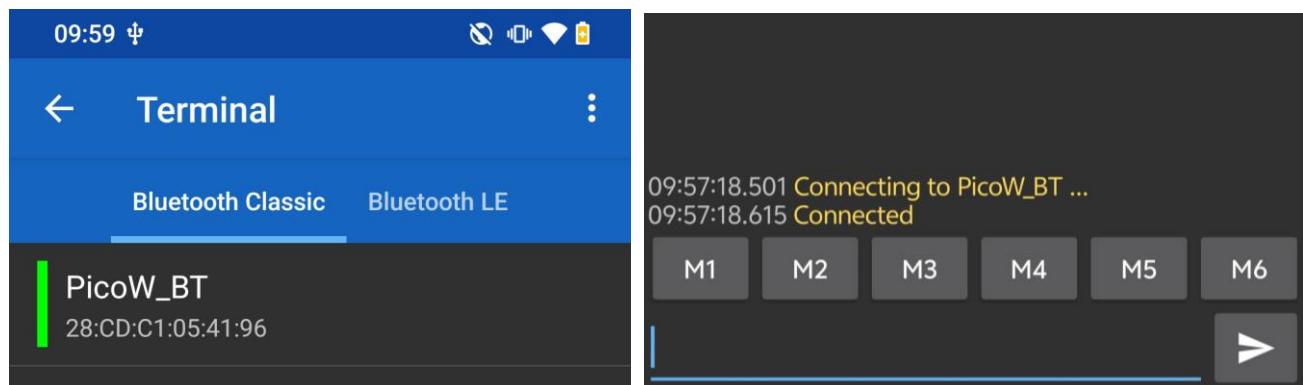


Open the App "Serial Bluetooth Terminal", expand the menu and select "Devices".



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

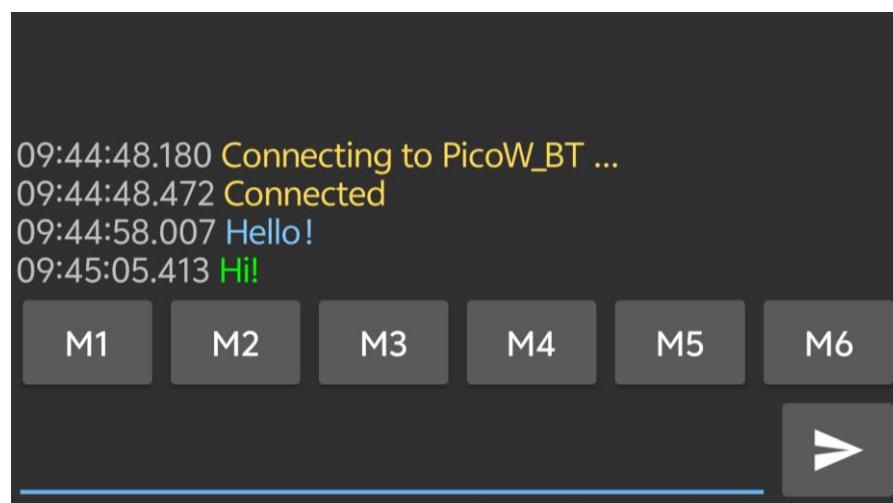
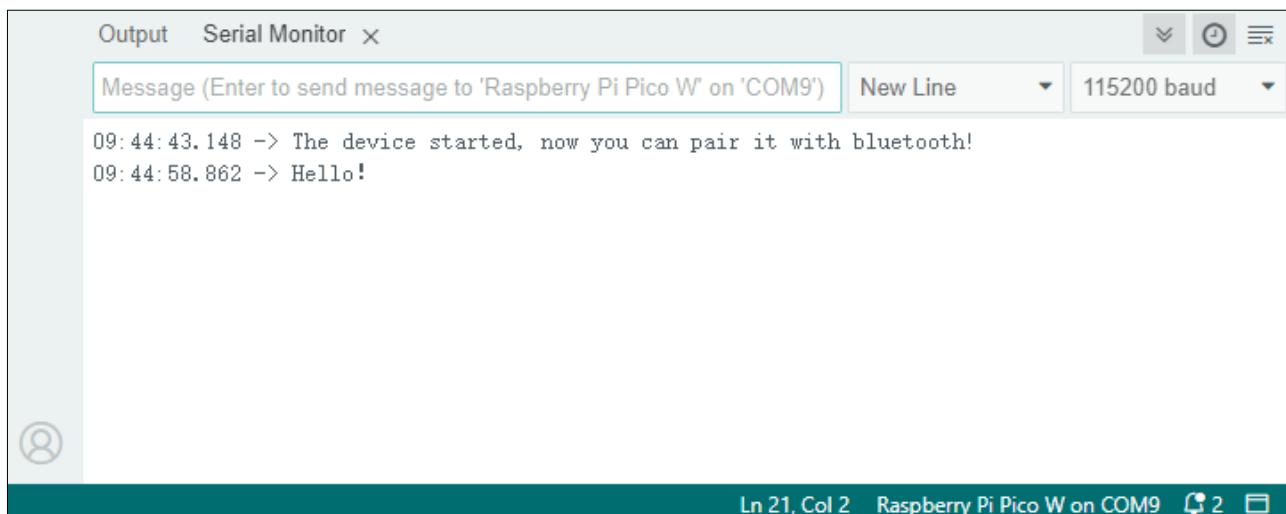
In Bluetooth Classic mode, select PicoW_BT, and you will see "Connected", which indicates a successful connection.



Now, the data can be transferred between your phone and computer (Pico W).

Input Hello at the sending bar on the phone app and tap the send icon to send message to pico W.

When the computer receives it, input "Hi" at the Message bar and hit Enter key to send message to your phone.





The following is the program code:

```
1 #include <SerialBT.h>
2
3 void setup() {
4     Serial.begin(115200);
5     SerialBT.setName("PicoW_BT"); //Bluetooth device name
6     SerialBT.begin();
7     while(!Serial);
8     Serial.println("The device started, now you can pair it with bluetooth!")
9 }
10
11 void loop() {
12     if(Serial.available()) {
13         SerialBT.print(Serial.readString());
14     }
15     if(SerialBT.available()) {
16         Serial.print(SerialBT.readString());
17     }
18     delay(5);
19 }
20 }
```

Reference

Class SerialBT

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

SetName(const char *name): Sets the Bluetooth module name.

begin(): Initializes the Bluetooth functionality.

available(): Retrieves data sent from the buffer; if none, returns 0.

read(): Reads data from Bluetooth, returns data as an int type.

readString(): Reads data from Bluetooth, returns data as a String type.

write(uint8_t c): Sends a single uint8_t type of data to Bluetooth.

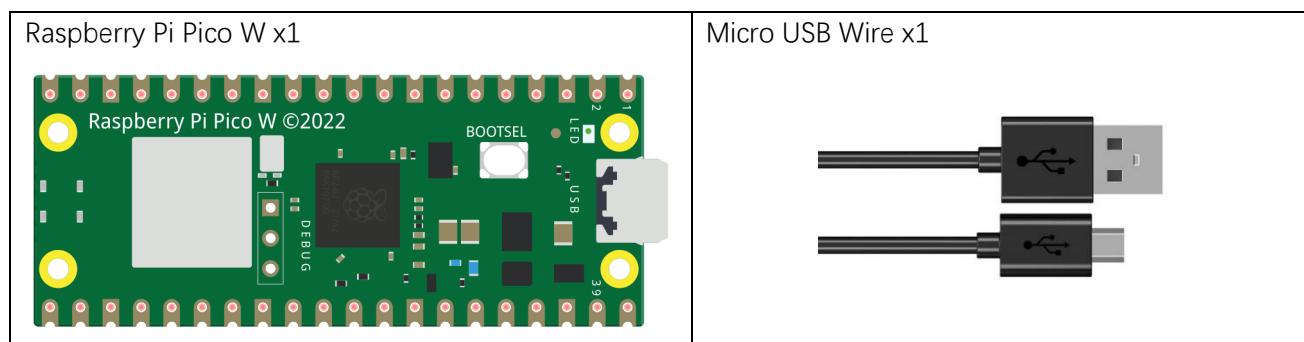
write(const uint8_t *p, size_t len): Sends the first len bytes of data stored at pointer address p to Bluetooth.

print(): Sends all types of data to Bluetooth for printing.

end(): Disconnects all Bluetooth devices and turns off Bluetooth, freeing up all occupied space.

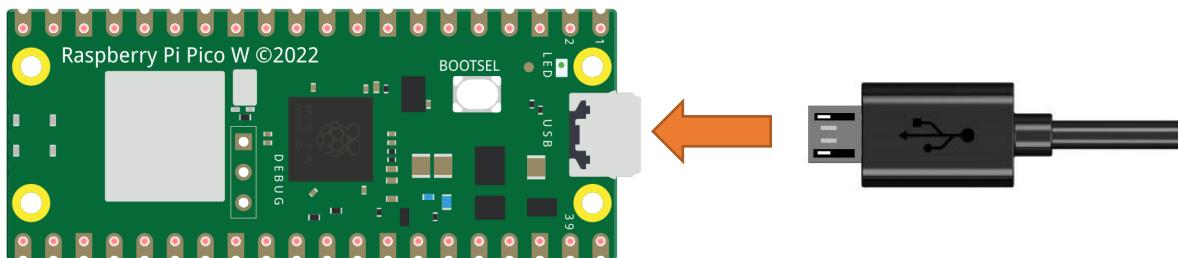
Project 33.2 Bluetooth Low Energy Data Passthrough

Component List

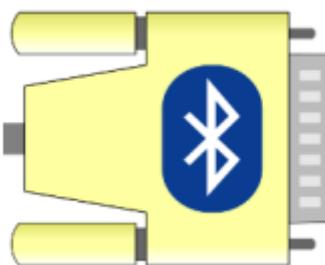


Circuit

Connect Pico W to the computer using the USB cable.



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





Sketch

Sketch_33.2_Serial_BT

```

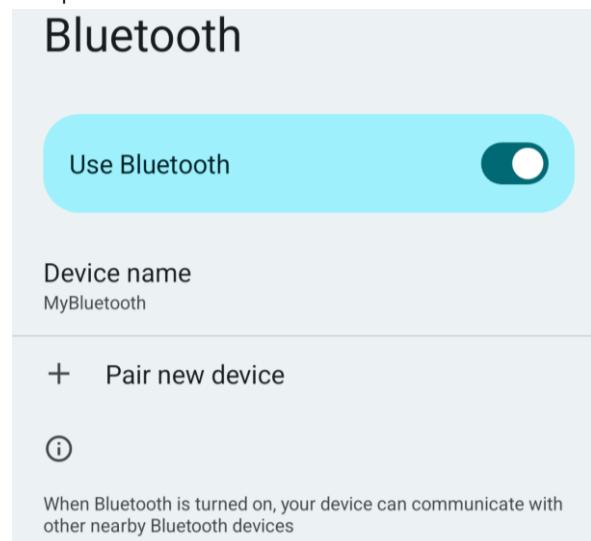
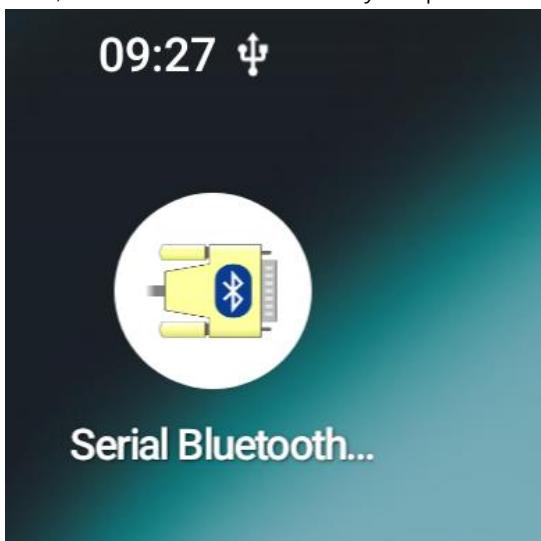
File Edit Sketch Tools Help
Serial_BLE.ino
42 void setup(void) {
43   Serial.begin(115200);
44   while(!Serial);
45
46   setupBLE("PicoW_BLE");
47 }
48
49 void loop(void) {
50   BTstack.loop();
51
52   if (Serial.available() > 0) {           // Check if there is serial
53     String input = Serial.readStringUntil('\n'); // Read until '\n'
54     input.trim();                         // Remove front and back blanks
55
56     size_t input_length = input.length();    // Get String length
57
58     // Copy input to characteristic_data and add line breaks
59     memcpy(characteristic_data, input.c_str(), input_length);
60     characteristic_data[input_length] = '\n'; // add linefeeds
61     characteristic_data[input_length + 1] = '\0'; // End of string
62
63     Serial.print("input data: ");
64     Serial.print(characteristic_data);

```

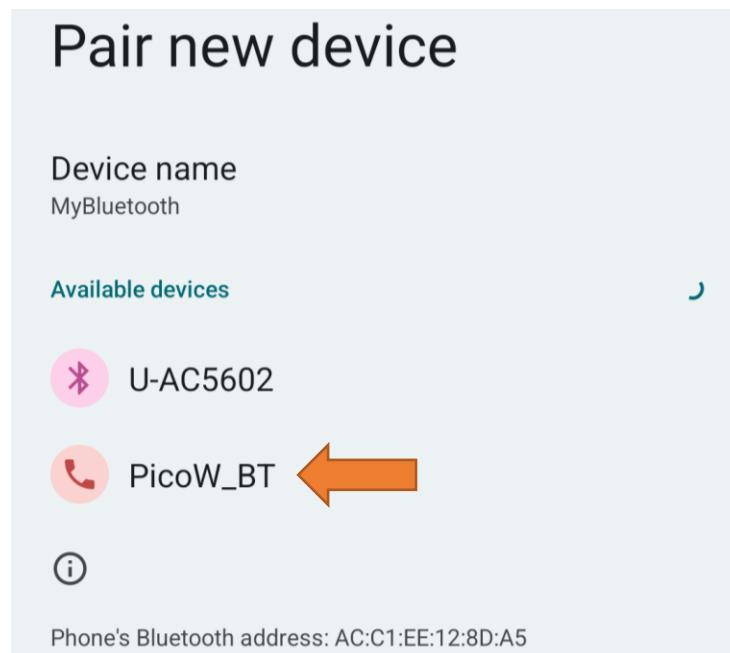
Before uploading the sketch, please make sure the Bluetooth of Pico W is enabled.

Serial Bluetooth

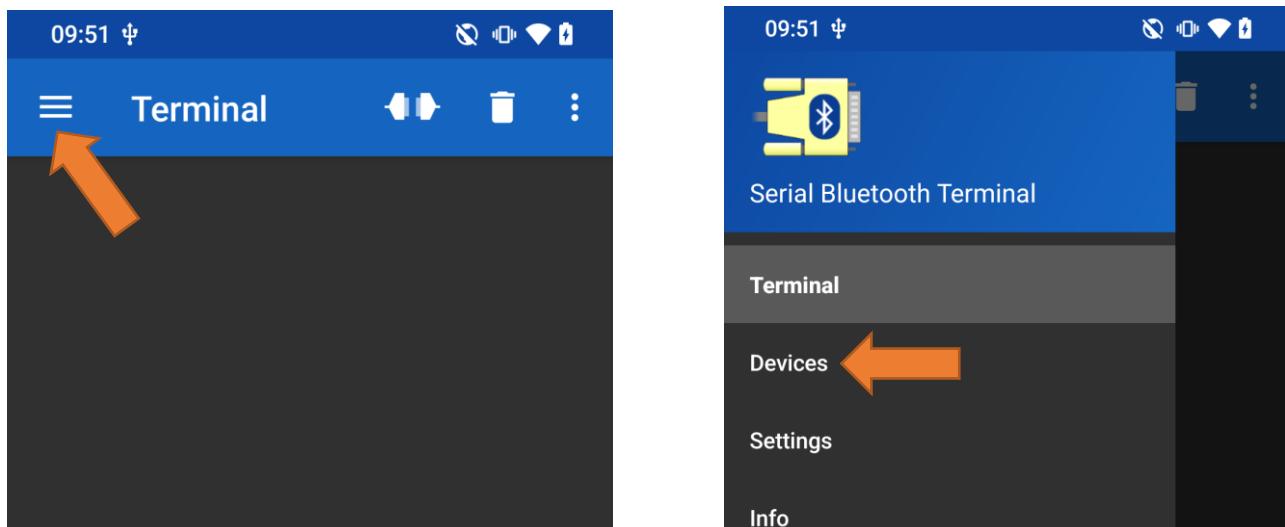
Compile and upload sketch th Pico W. The opration is similar to that in the previous section.
First, make sure Bluetooth on your phone is turned ON, and open Serial Bluetooth Terminal.



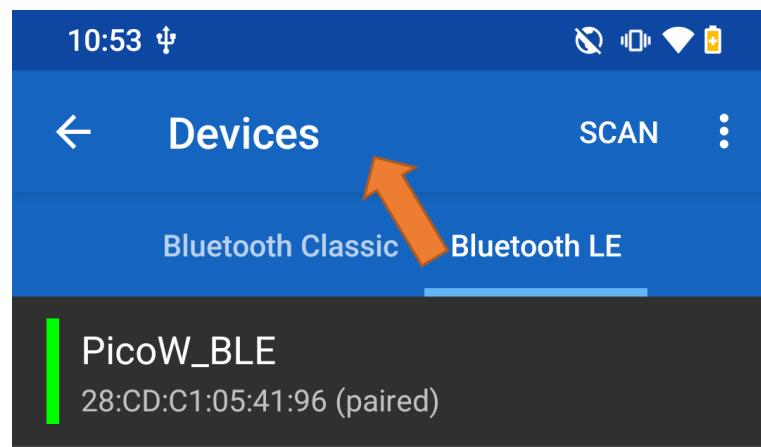
Click Pair new device, select "PicoW_BT" to connect.



Expand the menu at the left of the app, and select "Devices".

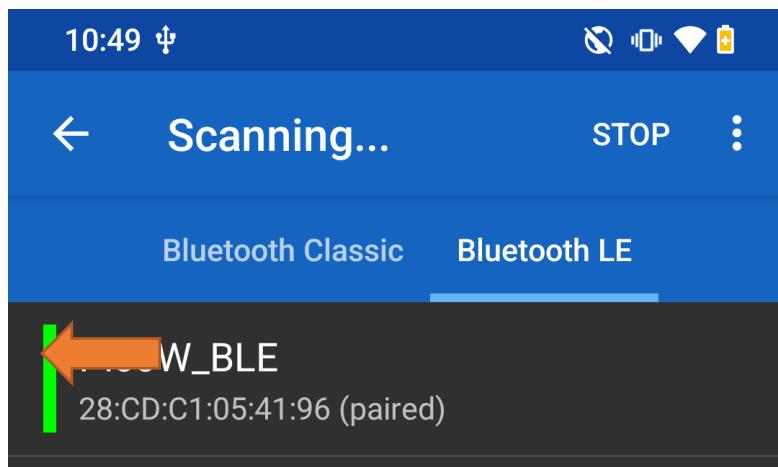


Select BLUETOOTH LE and click scan to search for BLE devices nearby.





Select “PicoW_BLE”.



LightBlue

If Serial Bluetooth cannot be installed on your phone, try LightBlue. If you do not have this software installed on your phone, you can refer to this link:

iPhone: <https://apps.apple.com/us/app/lightblue/id557428110?platform=iphone>



LightBlue® 4+

The go-to BLE development tool
Punch Through

★★★★★ 4.0 • 4 Ratings

Free

Screenshots Mac iPhone iPad

Step 1. Upload sketch 33.1 to Pico W.

Step 2. Open Serial Monitor.

```

File Edit Sketch Tools Help
Raspberry Pi Pico W
Sketch 33.1 BLE.ino
41 void setup(void) {
42     Serial.begin(115200);
43     while(!Serial);
44
45     setupBLE("PicoW_BLE");
46 }
47
48 void loop(void) {
49     BTstack.loop();
50
51     if (Serial.available() > 0) { // Check if there is serial
52         String input = Serial.readStringUntil('\n');
53         input.trim(); // Remove front and back bla
54
55         size_t input_length = input.length(); // Get String length
56     }
57 }
```

Step 3. Set the baud rate to 115200.

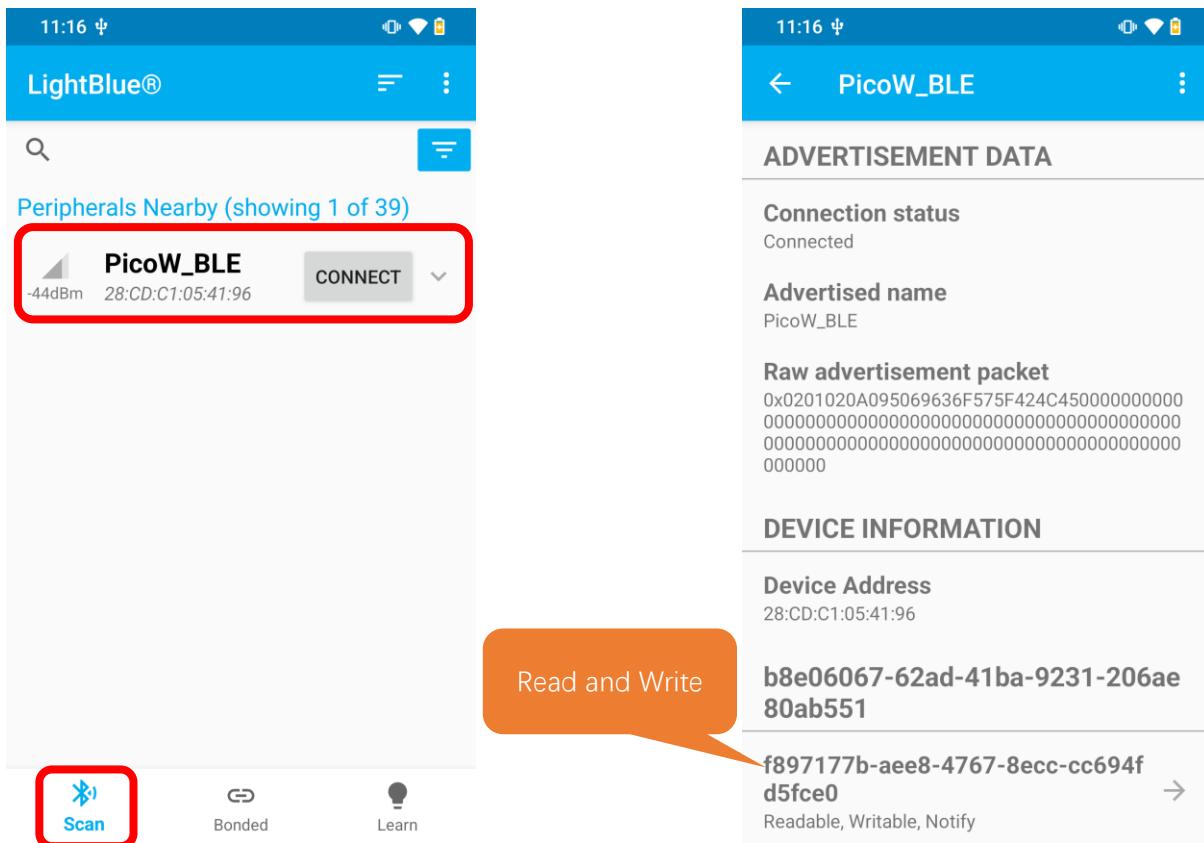


Turn ON Bluetooth on your phone and open LightBlue App.



Any concerns? ✉ support@freenove.com

At the device scan page, scroll down to refresh the devices nearby. Select PicoW_BLE to connect.



Click "Read and Write". Click on "Read and Write". Select the corresponding data format in the box on the right side of the data format, such as HEX hexadecimal, utf-string string, Binary binary, etc. Then click SUBSCRIBE.

Characteristic UUID
f897177b-aee8-4767-8ecc-cc694fd5fce0

- ✓ **Readable**
Able to be read from
- ✓ **Writable**
Able to be written to
- ✓ **Supports notifications/indications**
Able to be subscribed to for notifications/indications on changes to the characteristic

Data format

READ/INDICATED VALUES

READ AGAIN SUBSCRIBE

No value read recently
Tap on one of the buttons above — if available — to begin

WRITTEN VALUES

Return to the serial monitor on Arduino IDE, you can input any message and hit the Enter key to send.



Then you can see LightBlue on your phone receive the data.

The screenshot shows the LightBlue app interface. At the top, it displays the time "11:17" and battery level. Below is a list of characteristics:

- Readable**: Able to be read from
- Writable**: Able to be written to
- Supports notifications/indications**: Able to be subscribed to for notifications/indications on changes to the characteristic

A red box highlights the "Data format" dropdown set to "UTF-8 String".

READ/INDICATED VALUES

READ AGAIN **UNSUBSCRIBE**

A red box highlights the value "Hello" and the timestamp "Wed Oct 09 11:17:37 GMT+08:00 2024".

WRITTEN VALUES

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.



Similarly, you can send data from your phone to computer. Input any content and click WRITE to send the data.

11:18 ⓘ

← f897177b-aee8-4767-8ecc...

Supports notifications/indications

✓ Able to be subscribed to for notifications/indications on changes to the characteristic

Data format **UTF-8 String**

READ/INDICATED VALUES

READ AGAIN **UNSUBSCRIBE**

Hello

Wed Oct 09 11:17:37 GMT+08:00 2024

WRITTEN VALUES

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.

hello

WRITE

No value written recently

Input some data and tap on the "Write" button to begin

You can see the data received.

Output Serial Monitor ×

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')

11:44:49.498 -> Waiting a client connection to notify...

11:44:53.984 -> Device connected!

11:45:05.162 -> Received data: hello

New Line 115200 baud

Ln 40, Col 2 Raspberry Pi Pico W on COM9 1

Now the data can be transfer between your phone and pico W.

The following is the program code:

```
1 #include <BTstackLib.h>
2 #include <SPI.h>
3 #include <btstack.h>
4
5 #define MAX_LENGTH 256
6 static char characteristic_data[MAX_LENGTH] = "Pico W Bluetooth";
7
8 // Track subscription status
9 bool isSubscribed = false;
10
11 // The handle of the connection
12 hci_con_handle_t connection_handle = HCI_CON_HANDLE_INVALID;
13
14 // characteristic handle
15 uint16_t characteristic_handle = 0;
16
17 // Assuming the handle of CCCD is a characteristic handle+1
18 #define CCCD_HANDLE (characteristic_handle + 1)
19
20 void setupBLE(const char *BLEName) {
21     //Set callback function
22     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);
23     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);
24     BTstack.setGATTCharacteristicRead(gattReadCallback);
25     BTstack.setGATTCharacteristicWrite(gattWriteCallback);
26
27     //Set GATT database
28     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
29     characteristic_handle = BTstack.addGATTCharacteristicDynamic(
30         new UUID("f897177b-ae8-4767-8ecc-cc694fd5fce0"),
31         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
32         0
33     );
34
35     //Activate Bluetooth and broadcast
36     BTstack.setup(BLEName);
37     BTstack.startAdvertising();
38
39     Serial.println("Waiting a client connection to notify... ");
40 }
41
42 void setup() {
43     Serial.begin(115200);
```

```
44     while(!Serial);
45
46     setupBLE("PicoW_BLE");
47 }
48
49 void loop() {
50     BTstack.loop();
51
52     if (Serial.available() > 0) {           // Check if there is serial data
53         String input = Serial.readStringUntil('\n');
54         input.trim();                      // Remove front and back blank spaces
55
56         size_t input_length = input.length(); // Get String length
57
58         // Copy input to characteristic_data and add line breaks
59         memcpy(characteristic_data, input.c_str(), input_length);
60         characteristic_data[input_length] = '\n'; // add linefeeds
61         characteristic_data[input_length + 1] = '\0'; // End of string
62
63         Serial.print("input data: ");
64         Serial.print(characteristic_data);
65
66         sendNotificationToSubscribers();        //Send notifications to subscribed devices
67     }
68     //Delay by 5ms to prevent data loss due to receiving too quickly
69     delay(5);
70 }
71
72 void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
73     if(status == BLE_STATUS_OK) {
74         Serial.println("Device connected!");
75         connection_handle = device->getHandle(); // Get connection handle
76     }
77 }
78
79 void deviceDisconnectedCallback(BLEDevice * device) {
80     (void) device;
81     Serial.println("Disconnected.");
82     connection_handle = HCI_CON_HANDLE_INVALID;
83     isSubscribed = false;                  // Reset subscription status
84 }
85
86 // Callback function for reading data
87 uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
```

```
88     (void) value_handle;
89     if (buffer && buffer_size > 0) {
90         Serial.print("Read data: ");
91         Serial.println(characteristic_data);
92
93         size_t data_length = strlen(characteristic_data); // Get characteristic_data length
94         if (data_length > buffer_size) // Limit length
95             data_length = buffer_size;
96         }
97
98         memcpy(buffer, characteristic_data, data_length); // Copy String
99         return data_length;
100    }
101    return 0;
102 }
103
104 // Callback function for writing data
105 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {
106     if (value_handle == CCCD_HANDLE) { // Processing CCCD writing
107         if (size >= 2) {
108             uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
109             isSubscribed = true; //Subscription status is true
110         }
111     } else { // Writing of processing feature data
112         size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
113         memcpy(characteristic_data, buffer, copy_size); // Copy String
114         characteristic_data[copy_size] = '\0'; // Ensure that the string ends
115
116         Serial.print("Received data: ");
117         Serial.println(characteristic_data);
118     }
119
120     return 0;
121 }
122
123 void sendNotificationToSubscribers() {
124     if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
125         // Send notifications
126         att_server_notify(connection_handle, characteristic_handle,
127                           (uint8_t*)characteristic_data, strlen(characteristic_data));
128     }
129 }
130 }
```



Define the specified UUID number for BLE vendor.

```

28  BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
29  characteristic_handle = BTstack.addGATTCharacteristicDynamic(
30      new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),
31      ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
32      0
33  );

```

Write a Callback function for the BLE server to manage BLE connections.

```

72  void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
73      if(status == BLE_STATUS_OK) {
74          Serial.println("Device connected!");
75          connection_handle = device->getHandle();           // Get connection handle
76      }
77  }

```

Write a Callback function for the BLE server to handle BLE disconnections.

```

79  void deviceDisconnectedCallback(BLEDevice * device) {
80      (void) device;
81      Serial.println("Disconnected.");
82      connection_handle = HCI_CON_HANDLE_INVALID;
83      isSubscribed = false;                                // Reset subscription status
84  }

```

Write a Callback function to read data. When BLE needs to read data, print the read data on the phone.

```

86  // Callback function for reading data
87  uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
88      (void) value_handle;
89      if (buffer && buffer_size > 0) {
90          Serial.print("Read data: ");
91          Serial.println(characteristic_data);
92
93          size_t data_length = strlen(characteristic_data); // Get characteristic_data length
94          if (data_length > buffer_size) {                  // Limit length
95              data_length = buffer_size;
96          }
97
98          memcpy(buffer, characteristic_data, data_length); // Copy String
99          return data_length;
100     }
101
102 }

```

Create a Callback function for writing data. When data is written to BLE, check if it is the CCCD characteristic handle or a write characteristic handle. If it is the CCCD, it indicates a subscription state; otherwise, print the written data to the serial console.

```

104 // Callback function for writing data
105 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {

```

Any concerns? ✉ support@freenove.com

```

106 if (value_handle == CCCD_HANDLE) {           // Processing CCCD writing
107     if (size >= 2) {
108         uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
109         isSubscribed = true;                      //Subscription status is true
110     }
111 } else { // Writing of processing feature data
112     size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
113     memcpy(characteristic_data, buffer, copy_size); // Copy String
114     characteristic_data[copy_size] = '\0';          // Ensure that the string ends
115
116     Serial.print("Received data: ");
117     Serial.println(characteristic_data);
118 }
119
120 return 0;
121 }
```

Write a function to send subscription notifications. When data is received through the serial port, send a notification to let BLE receive the data.

```

123 void sendNotificationToSubscribers() {
124     if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
125         // Send notifications
126         att_server_notify(connection_handle, characteristic_handle,
127                            (uint8_t*)characteristic_data, strlen(characteristic_data));
128     }
129 }
```

Process serial port data: Upon receiving data from the serial port, remove any leading and trailing whitespace, append a newline character to the end of the string, and then use the notification function to transmit the data to the mobile device via BLE.

```

52     if (Serial.available() > 0) {           // Check if there is serial data
53         String input = Serial.readStringUntil('\n');
54         input.trim();                      // Remove front and back blank spaces
55
56         size_t input_length = input.length(); // Get String length
57
58         // Copy input to characteristic_data and add line breaks
59         memcpy(characteristic_data, input.c_str(), input_length);
60         characteristic_data[input_length] = '\n'; // add linefeeds
61         characteristic_data[input_length + 1] = '\0'; // End of string
62
63         Serial.print("input data: ");
64         Serial.print(characteristic_data);
65
66         sendNotificationToSubscribers();        //Send notifications to subscribed devices
67     }
```



The design for creating the BLE server is:

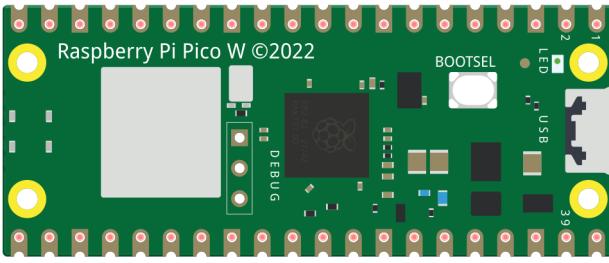
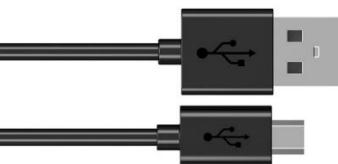
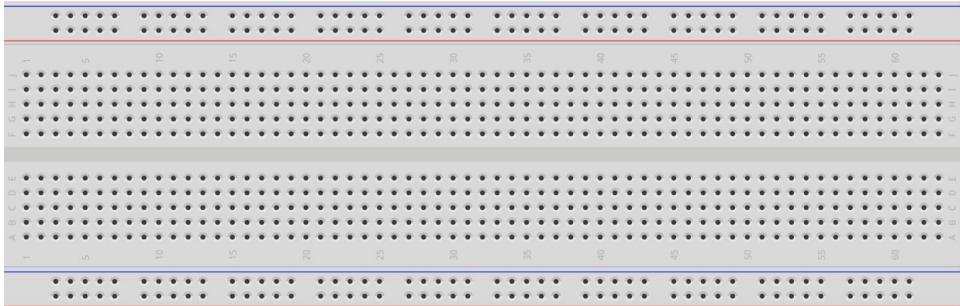
1. Set up a connection Callback function.
2. Set up a Callback function for disconnection.
3. Set up GATT read characteristic.
4. Set up GATT write characteristic.
5. Add GATT service.
6. Add GATT dynamic characteristic (read, write, notify).
7. Initialize BLE.
8. Start advertising.

```
20 void setupBLE(const char *BLEName) {  
21     //Set callback function  
22     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);  
23     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);  
24     BTstack.setGATTCharacteristicRead(gattReadCallback);  
25     BTstack.setGATTCharacteristicWrite(gattWriteCallback);  
26  
27     //Set GATT database  
28     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));  
29     characteristic_handle = BTstack.addGATTCharacteristicDynamic(  
30         new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),  
31         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,  
32         0  
33     );  
34  
35     //Activate Bluetooth and broadcast  
36     BTstack.setup(BLEName);  
37     BTstack.startAdvertising();  
38  
39     Serial.println("Waiting a client connection to notify...");  
40 }
```

Project 33.3 Bluetooth Control LED

In this project, we will control an LED via Pico W's Bluetooth function.

Component List

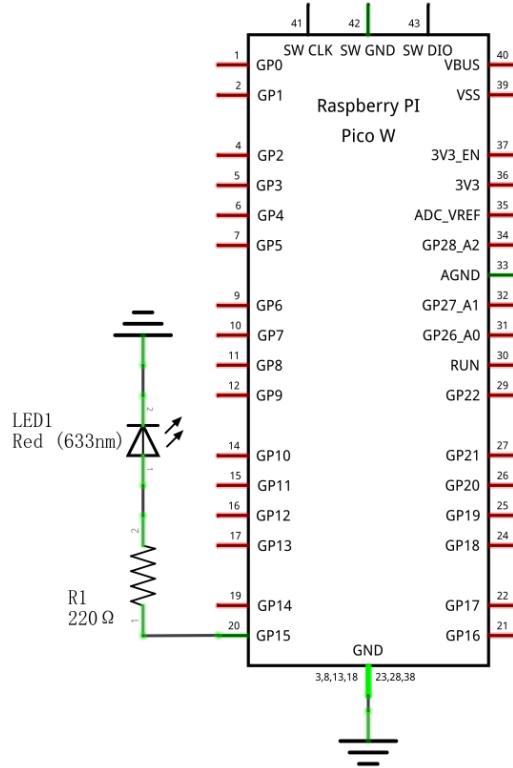
Raspberry Pi Pico W x1	Micro USB Wire x1
	
Breadboard x1	
	  



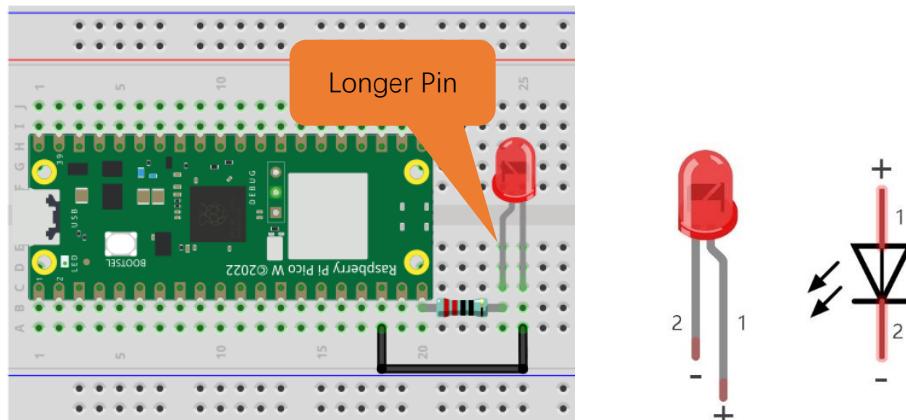
Circuit

Connect Pico W to your phone with the USB cable.

Schematic diagram



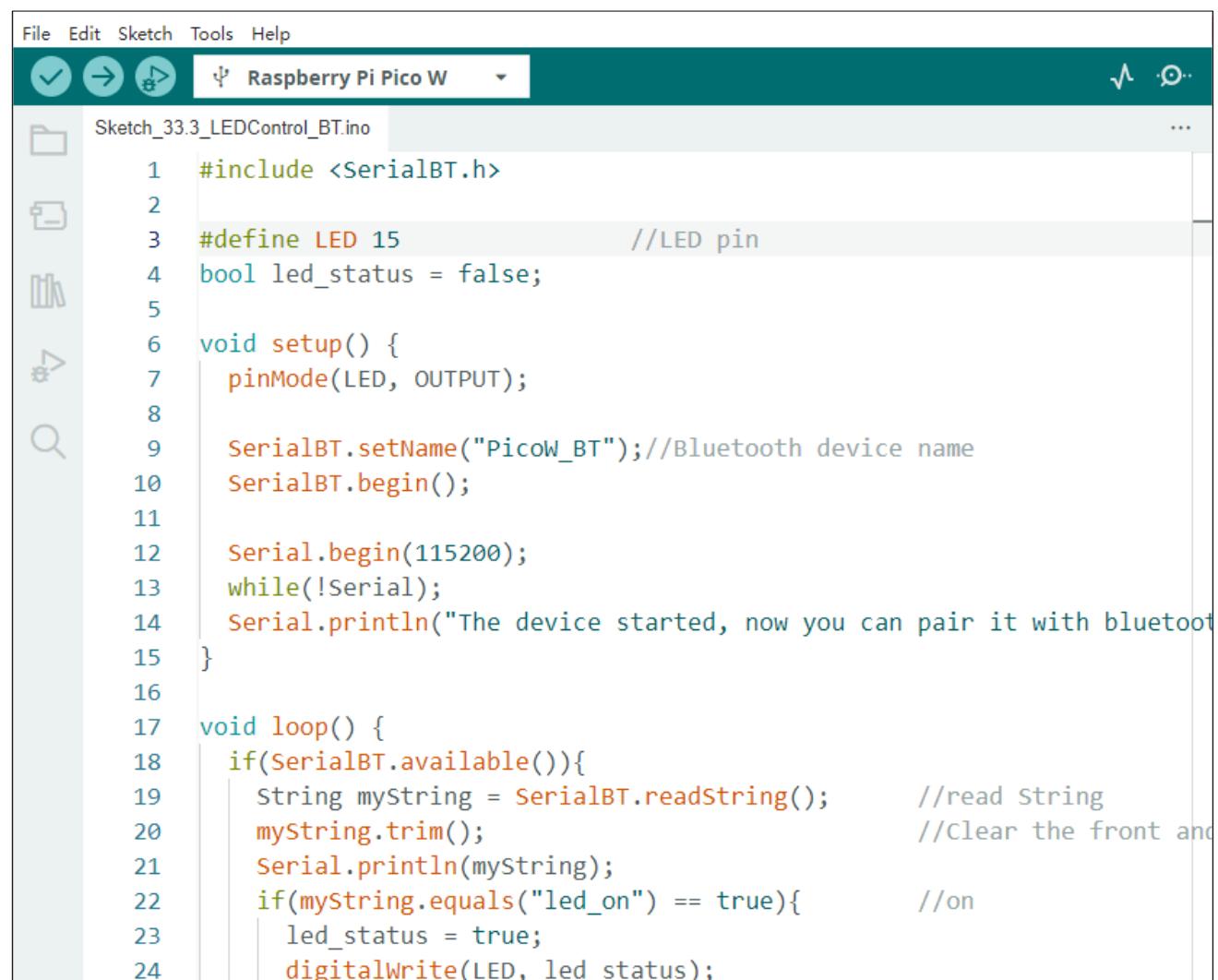
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Sketch

Sketch_33.3_LEDControl_BT



The screenshot shows the Arduino IDE interface with the sketch file `Sketch_33.3_LEDControl_BT.ino` open. The code is written in C++ and uses the `SerialBT.h` library to control a LED connected to pin 15. The sketch sets up the serial connection and begins listening for commands. It reads strings from the serial port, trims them, and then checks if they are "led_on", "led_off", or "led_toggle". If "led_on" is found, it sets the LED status to true and writes it to the pin. If "led_off" is found, it sets the LED status to false and writes it to the pin. If "led_toggle" is found, it toggles the LED status and writes it to the pin.

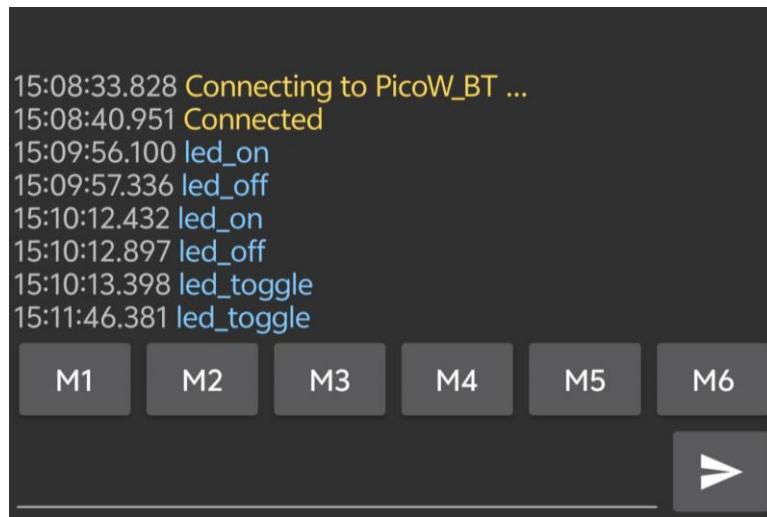
```
#include <SerialBT.h>
#define LED 15 //LED pin
bool led_status = false;

void setup() {
    pinMode(LED, OUTPUT);
    SerialBT.setName("PicoW_BT");//Bluetooth device name
    SerialBT.begin();
    Serial.begin(115200);
    while(!Serial);
    Serial.println("The device started, now you can pair it with bluetooth");
}

void loop() {
    if(SerialBT.available()){
        String myString = SerialBT.readString(); //read String
        myString.trim(); //Clear the front and back spaces
        Serial.println(myString);
        if(myString.equals("led_on") == true){ //on
            led_status = true;
            digitalWrite(LED, led_status);
        }
        if(myString.equals("led_off") == true){ //off
            led_status = false;
            digitalWrite(LED, led_status);
        }
        if(myString.equals("led_toggle") == true){ //toggle
            led_status = !led_status;
            digitalWrite(LED, led_status);
        }
    }
}
```

Upload the sketch to Pico W.

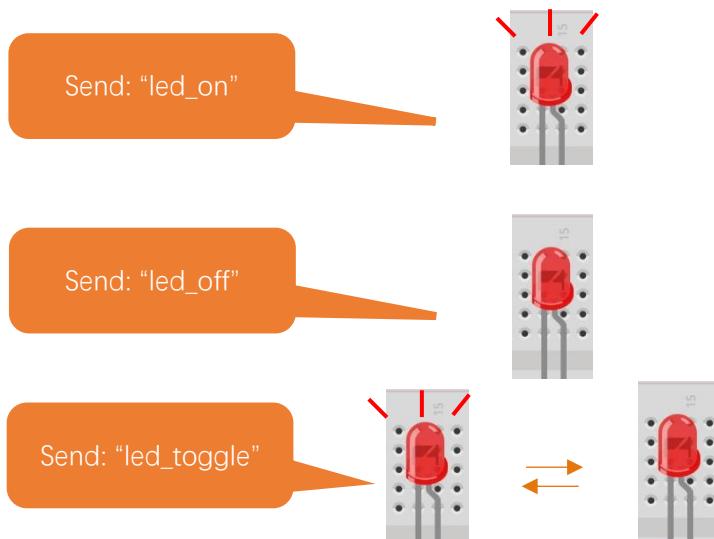
The operation on the phone app is similar to that in Project 33.1. You just need to change the sending messages to "led_on", "led_off" and "led_toggle" to control the status of the LED.



Displays on the serial monitor.



LED status.



Note: If the messages you send are not "led_on", "led_off", or "led toggle", the status of the LED won't change. For example, when the LED is already ON, it will remain ON unless the message "led_off", or "led toggle" is received.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

The following is the program code:

```

1 #include <SerialBT.h>
2
3 #define LED 15           //LED pin
4 bool led_status = false;
5
6 void setup() {
7     pinMode(LED, OUTPUT);
8
9     SerialBT.setName("PicoW_BT"); //Bluetooth device name
10    SerialBT.begin();
11
12    Serial.begin(115200);
13    while(!Serial);
14    Serial.println("The device started, now you can pair it with bluetooth!");
15 }
16
17 void loop() {
18     if(SerialBT.available()) {
19         String myString = SerialBT.readString(); //read String
20         myString.trim(); //Clear the front and back blank spaces
21         Serial.println(myString);
22         if(myString.equals("led_on") == true){ //on
23             led_status = true;
24             digitalWrite(LED, led_status);
25         }
26         if(myString.equals("led_off") == true){ //off
27             led_status = false;
28             digitalWrite(LED, led_status);
29         }
30         if(myString.equals("led_toggle") == true){ //toggle
31             led_status = !led_status;
32             digitalWrite(LED, led_status);
33         }
34     }
35 }
```

Define the pin of LED and set its default status as false.

```

3 #define LED 15           //LED pin
4 bool led_status = false;
```

Set the LED pin to output mode.

```

3 pinMode(LED, OUTPUT);
```

Process the received Bluetooth data, determine the status of the LED light. If it is "led_on", the LED light is on; if it is "led_off", the LED light is off; if it is "led_toggle", the LED status is opposite to the current status.

```

18 if(SerialBT.available()) {
```

```

19   String myString = SerialBT.readString();      //read String
20   myString.trim();                            //Clear the front and back blank spaces
21   Serial.println(myString);
22   if(myString.equals("led_on") == true){        //on
23       led_status = true;
24       digitalWrite(LED, led_status);
25   }
26   if(myString.equals("led_off") == true){        //off
27       led_status = false;
28       digitalWrite(LED, led_status);
29   }
30   if(myString.equals("led_toggle") == true){    //toggle
31       led_status = !led_status;
32       digitalWrite(LED, led_status);
33   }
34 }
```

Reference

The ‘trim()’ function is used to remove whitespace from both sides of a string, converting "led_on\n" to "led_on" for easier subsequent judgment.

void String: :trim(void)

removes whitespace from both sides of a string.

The ‘equals()’ function is used to compare if two strings are identical, providing a simple and fast way to compare strings.

unsigned char String: :equals(const String &s2)

s2: The other string to be compared.

Return value: If s2 is not equal to the current string, the return value is 0.

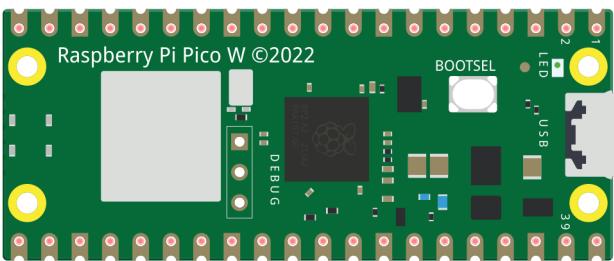
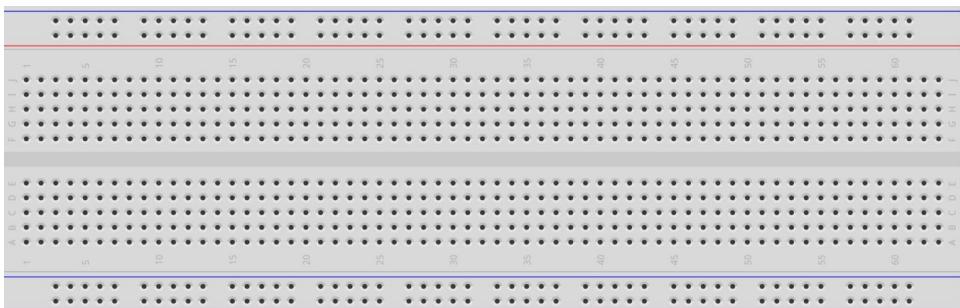
If s2 is equal to the current string, the return value is 1.

Note: Both data being compared must be of string type.

Project 33.4 Bluetooth Low Energy Control LED

This project is basically the same as the previous one. You do not need to change the circuit wiring. Just upload the corresponding sketch.

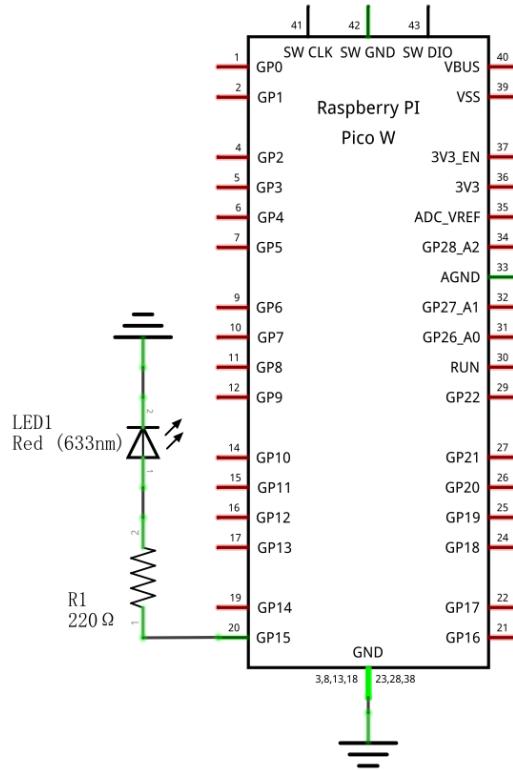
Component List

Raspberry Pi Pico W x1	Micro USB Wire x1	
		
Breadboard x1		
LED x1	Resistor 220Ω x1	Jumper

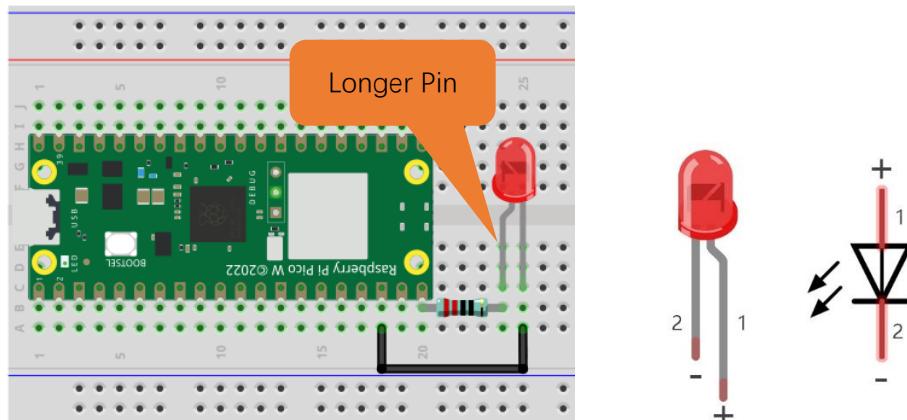
Circuit

Connect the Pico W to your computer with a USB cable.

Schematic diagram



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



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Sketch

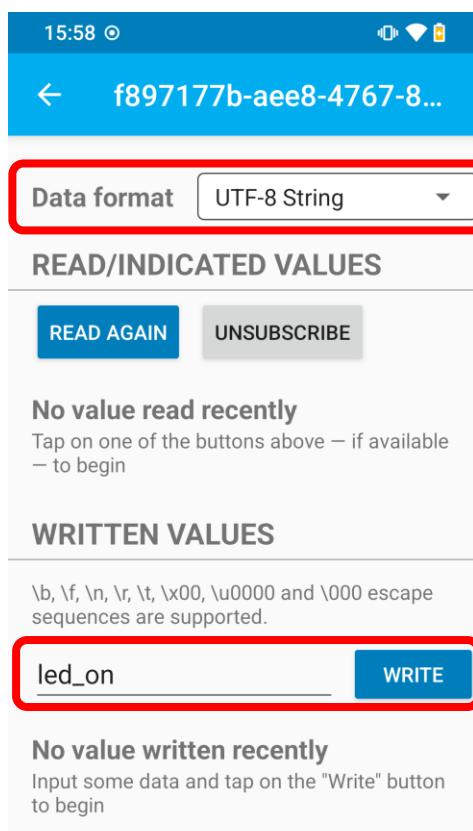
Sketch_33.4_LEDControl_BLE



```

File Edit Sketch Tools Help
Sketch_33.4_LEDControl_BLE.ino
Sketch_33.4_LEDControl_BLE.ino ...
46 void setup(void) {
47     pinMode(LED, OUTPUT);
48
49     Serial.begin(115200);
50     while(!Serial);
51
52     setupBLE("PicoW_BLE");
53 }
54
55 void loop(void) {
56     BTstack.loop();
57
58     if (Serial.available() > 0) { // Check if there is serial
59         String input = Serial.readStringUntil('\n');
60         input.trim(); //Remove front and back blank
61
62         size_t input_length = input.length(); // Get String length
    
```

Upload the sketch to Pico W. As previously mentioned, you can input “led_on”, “led_off”, or “led_toggle” to change the status of the LED.





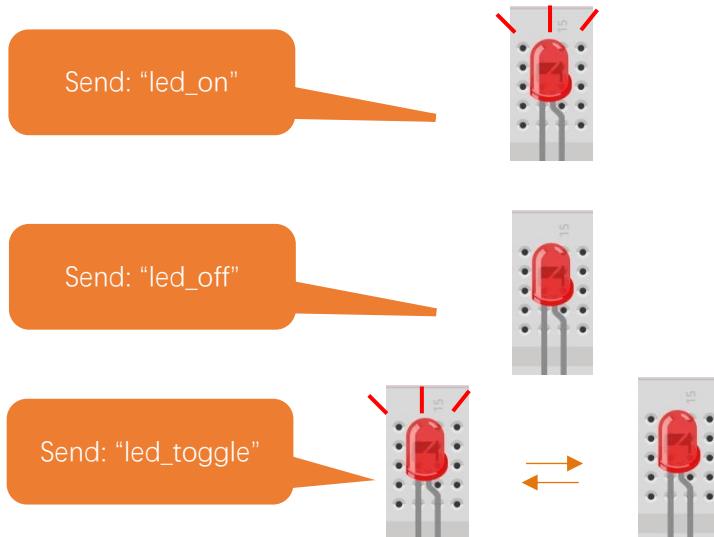
Displays on the serial monitor.

The screenshot shows the 'Serial Monitor' window with the title 'Output' and 'Serial Monitor'. The message area displays the following log:

```
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')  
15:59:27.800 -> Waiting a client connection to notify...  
15:59:28.952 -> Device connected!  
15:59:49.320 -> Received data: led_on  
15:59:54.347 -> Received data: led_off  
15:59:58.369 -> Received data: led_on  
16:00:01.319 -> Received data: led_off  
16:00:06.918 -> Received data: led_toggle  
16:00:07.991 -> Received data: led_toggle
```

At the bottom right of the window, it says 'Ln 47, Col 18 Raspberry Pi Pico W on COM9' with a speech bubble icon and a '2'.

LED Status.



Note: If the messages you send are not "led_on", "led_off", or "led toggle", the status of the LED will not change.

For example, when the LED is already ON, it will remain ON unless the message "led_off", or "led toggle" is received.

The following is the program code:

```
1 #include <BTstackLib.h>
2 #include <SPI.h>
3 #include <btstack.h>
4
5 #define LED 15           // LED pin
6 bool led_status = false;
7
8 #define MAX_LENGTH 256
9 static char characteristic_data[MAX_LENGTH] = "Pico W Bluetooth";
10
11 // Track subscription status
12 bool isSubscribed = false;
13
14 // The handle of the connection
15 hci_con_handle_t connection_handle = HCI_CON_HANDLE_INVALID;
16
17 // characteristic handle
18 uint16_t characteristic_handle = 0;
19
20 // Assuming the handle of CCCD is a characteristic handle+1
21 #define CCCD_HANDLE (characteristic_handle + 1)
22
23 void setupBLE(const char *BLEName) {
24     //Set callback function
25     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);
26     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);
27     BTstack.setGATTCharacteristicRead(gattReadCallback);
28     BTstack.setGATTCharacteristicWrite(gattWriteCallback);
29
30     //Set GATT database
31     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
32     characteristic_handle = BTstack.addGATTCharacteristicDynamic(
33         new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),
34         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
35         0
36     );
37
38     //Activate Bluetooth and broadcast
39     BTstack.setup(BLEName);
40     BTstack.startAdvertising();
41
42     Serial.println("Waiting a client connection to notify...");
43 }
```

```
44
45 void setup() {
46     pinMode(LED, OUTPUT);
47
48     Serial.begin(115200);
49     while(!Serial);
50
51     setupBLE("PicoW_BLE");
52 }
53
54 void loop() {
55     BTstack.loop();
56
57     if (Serial.available() > 0) { // Check if there is serial data
58         String input = Serial.readStringUntil('\n');
59         input.trim(); // Remove front and back blank spaces
60
61         size_t input_length = input.length(); // Get String length
62
63         // Copy input to characteristic_data and add line breaks
64         memcpy(characteristic_data, input.c_str(), input_length);
65         characteristic_data[input_length] = '\n'; // add linefeeds
66         characteristic_data[input_length + 1] = '\0'; // End of string
67
68         Serial.print("input data: ");
69         Serial.print(characteristic_data);
70
71         sendNotificationToSubscribers(); //Send notifications to subscribed devices
72     }
73     //Delay by 5ms to prevent data loss due to receiving too quickly
74     delay(5);
75 }
76
77 void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
78     if(status == BLE_STATUS_OK) {
79         Serial.println("Device connected!");
80         connection_handle = device->getHandle(); // Get connection handle
81     }
82 }
83
84 void deviceDisconnectedCallback(BLEDevice * device) {
85     (void) device;
86     Serial.println("Disconnected.");
87     connection_handle = HCI_CON_HANDLE_INVALID;
```

```
88     isSubscribed = false;           // Reset subscription status
89 }
90
91 // Callback function for reading data
92 uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
93     (void) value_handle;
94     if (buffer && buffer_size > 0) {
95         Serial.print("Read data: ");
96         Serial.println(characteristic_data);
97
98         size_t data_length = strlen(characteristic_data); // Get characteristic_data length
99         if (data_length > buffer_size) {                  // Limit length
100             data_length = buffer_size;
101         }
102
103         memcpy(buffer, characteristic_data, data_length); // Copy String
104         return data_length;
105     }
106     return 0;
107 }

108
109 // Callback function for writing data
110 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {
111     if (value_handle == CCCD_HANDLE) {                // Processing CCCD writing
112         if (size >= 2) {
113             uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
114             isSubscribed = true;                      //Subscription status is true
115         }
116     } else {                                         // Writing of processing feature data
117         size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
118         memcpy(characteristic_data, buffer, copy_size); // Copy String
119         characteristic_data[copy_size] = '\0';          // Ensure that the string ends
120
121         String myString = String(characteristic_data)    // Convert to String
122         myString.trim();                                // Remove front and back blank spaces
123         if(myString.equals("1led_on") == true){          // led on
124             led_status = true;
125             digitalWrite(LED, led_status);
126         }
127         if(myString.equals("1led_off") == true){          // led off
128             led_status = false;
129             digitalWrite(LED, led_status);
130         }
131         if(myString.equals("1led_toggle") == true){        // led toggle
```

```
132     led_status = !led_status;
133     digitalWrite(LED, led_status);
134 }
135
136     Serial.print("Received data: ");
137     Serial.println(characteristic_data);
138 }
139
140 return 0;
141 }
142
143 void sendNotificationToSubscribers() {
144     if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
145         // Send notifications
146         att_server_notify(connection_handle, characteristic_handle,
147                            (uint8_t*)characteristic_data, strlen(characteristic_data));
148     }
149 }
```

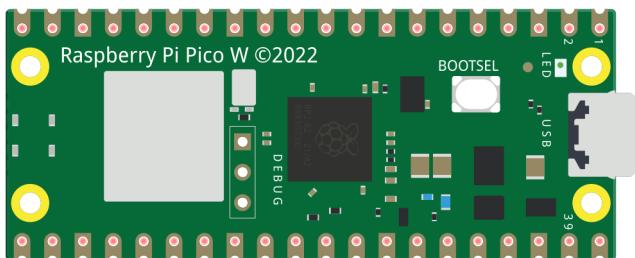
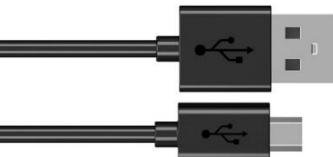
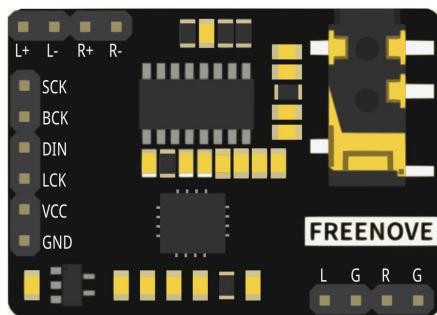
Chapter 34 Bluetooth Audio (Only for Pico W)

Raspberry Pi Pico W integrates both Classic Bluetooth and Bluetooth Low Energy (BLE), enabling it to transmit not only simple data and commands but also files including text and audio. In this section, we will use the Bluetooth reception function to receive music from a smartphone and play it.

Project 33.1 Bluetooth Passthrough

Utilizing the Bluetooth audio reception capability of the Raspberry Pi Pico W, we can decode audio data from a smartphone using the A2DP protocol, allowing the Raspberry Pi Pico W to receive high-quality audio. In this project, we will employ an audio converter and amplifier to transform the audio data into stereo and output it.

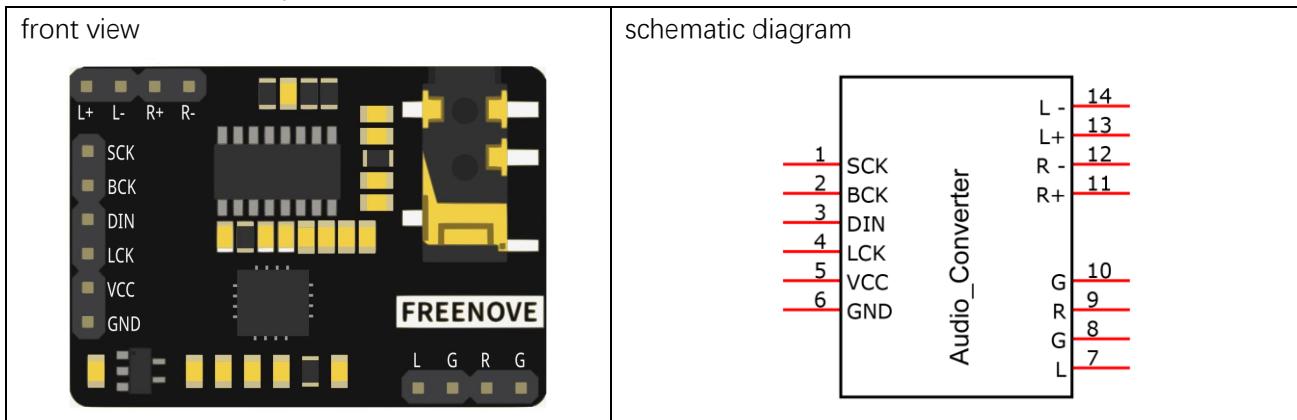
Component List

Raspberry Pi Pico W x1	 A photograph of the Raspberry Pi Pico W development board. It is a green PCB with a central Broadcom chip, various capacitors, resistors, and connectors. Two yellow circular pads are visible on the left side. Labels include "Raspberry Pi Pico W ©2022", "DEBUG", "BOOTSEL", "LED", and "USB".	Micro USB Wire x1	 A photograph of a standard black Micro USB cable with two ends, each featuring a black plastic housing and a metal connector.
Audio Converter & Amplifier	 A photograph of the Freenove Audio Converter & Amplifier module. It is a black PCB with several surface-mount components, including a large black integrated circuit and smaller resistors and capacitors. Pin labels on the left include L+, L-, R+, R-, SCK, BCK, DIN, LCK, VCC, GND, and GND. On the right, pins are labeled L, G, R, G.	Speaker*1	 A photograph of a black speaker with a circular grille and a central black driver unit. Two wires are visible at the bottom.
Jumper wire F/M*4			 A photograph of a long, thin black jumper wire with four pins at each end, designed for connecting between the Pico W and the audio converter module.



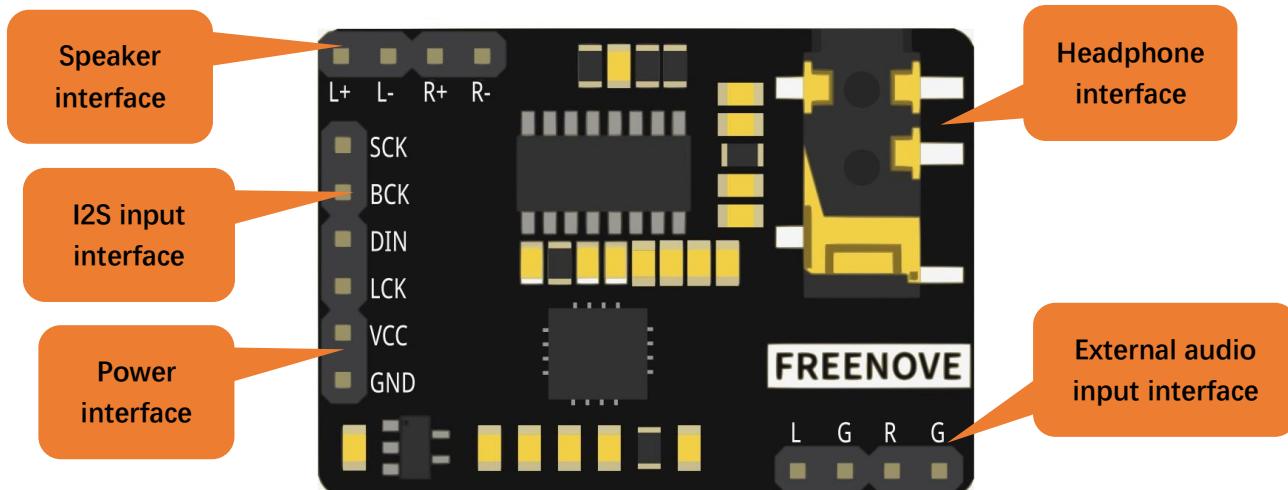
Component knowledge

Audio Converter & Amplifier module



Interface description for Audio Converter & Amplifier module

Pin	Name	Introductions
1	SCK	System clock input
2	BCK	Audio data bit clock input
3	DIN	Audio data input
4	LCK	Audio data word clock input
5	VCC	Power input, 3.3V~5.0V
6	GND	Power Ground
7	L	External audio left channel input
8	G	Power Ground
9	R	External audio right channel input
10	G	Power Ground
11	R+	Positive pole of right channel horn
12	R-	Negative pole of right channel horn
13	L+	Positive pole of left channel horn
14	L-	Negative pole of left channel horn



Speaker interface: Connect left channel speaker and right channel speaker. Group L: L+ & L-; Group R: R+ & R-. The two interfaces of the speaker can be connected to the interfaces of group L or group R. However, when one interface is connected to group L, the other cannot be connected to group R. Doing so may cause the module to malfunction.

Headphone interface: the interface to connect the headphones.

I2S input interface: connect to the device with I2S. Used to transcode audio data into DAC audio signals.

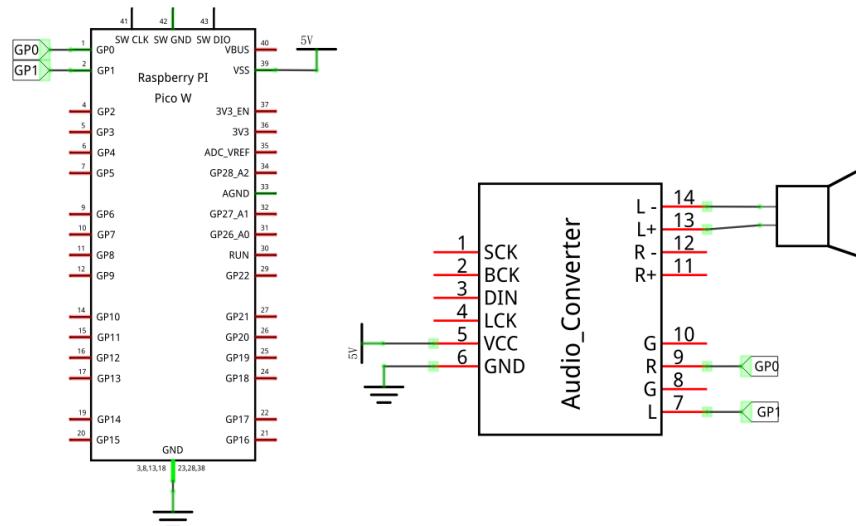
External audio input interface: connect to external audio equipment. Used to amplify externally input audio signals.

Power interface: connect to external power supply. External power supply selection range: 3.3V-5.0V.

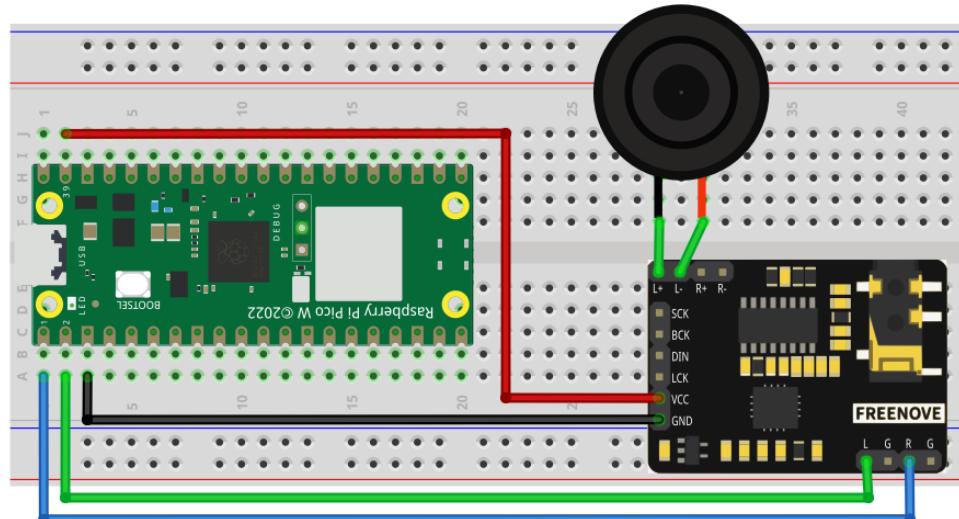
Circuit

The connection between the control board and the audio module is shown in the figure below.

Schematic diagram



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





Sketch

Sketch_33.4_Bluetooth_By_PCM5102A

```

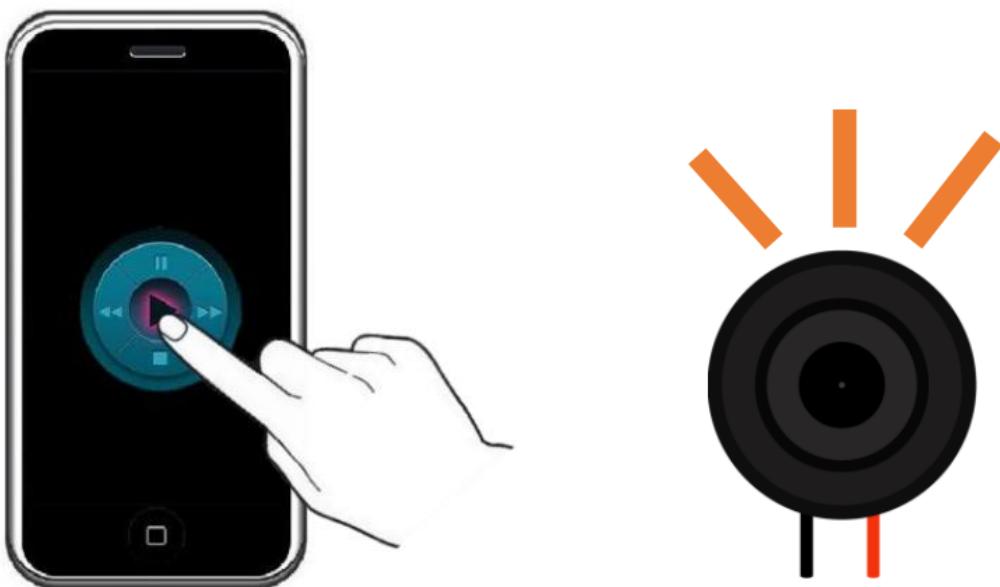
File Edit Sketch Tools Help
Sketch_33.4_Bluetooth_By_PCM5102A.ino
1 #include <BluetoothAudio.h>
2 #include <PWMAudio.h>
3
4 PWMAudio pwm;
5 A2DPSink a2dp;
6
7 // Define a playback state
8 volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
9
10 // Volume callback function
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%\n", pct);
14 }
15
16 // Connect callback function
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
26
27 // Play status callback function
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }

```

When you see the messages as shown below, it indicates that the Pico W's Bluetooth is ready for connection and playing music.

Message	Time
17:38:41.169 -> Starting, connect to the PicoW and start playing music	17:38:41.169
17:38:41.169 -> Use BOOTSEL to pause/resume playback	17:38:41.169
17:38:41.918 -> NOW PLAYING:	17:38:41.918

Search “Pico W Boom” on your phone to connect. Upon successful connection, you can display audio via Pico W.



The following is the program code:

```
1 #include <BluetoothAudio.h>
2 #include <PWMAudio.h>
3
4 PWMAudio pwm;
5 A2DPSink a2dp;
6
7 // Define a playback state
8 volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
9
10 // Volume callback function
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%%\n", pct);
14 }
15
16 // Connect callback function
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
```

```

27 // Play status callback function
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }
32
33 void setup() {
34     Serial.begin(115200);
35     while(!Serial);
36
37     Serial.printf("Starting, connect to the PicoW and start playing music\n");
38     Serial.printf("Use BOOTSEL to pause/resume playback\n");
39
40     a2dp.setName("PicoW Boom 00:00:00:00:00:00");
41     // Set the audio consumer to PWM audio output
42     a2dp.setConsumer(new BluetoothAudioConsumerPWM(pwm));
43     a2dp.onVolume(volumeCB);
44     a2dp.onConnect(connectCB);
45     a2dp.onPlaybackStatus(playbackCB);
46     a2dp.begin();
47 }
48
49 char *nowPlaying = nullptr; //Store the name of the currently playing track
50
51 void loop() {
52     if (BOOTSEL) {
53         if (status == A2DPSink::PAUSED) {           // if the playback status is paused
54             a2dp.play();                            // play
55             Serial.printf("Resuming\n");
56         } else if (status == A2DPSink::PLAYING) { // if the playback status is playing
57             a2dp.pause();                          // pause
58             Serial.printf("Pausing\n");
59         }
60         while (BOOTSEL);
61     }
62     if (!nowPlaying || strcmp(nowPlaying, a2dp.trackTitle())) {
63         free(nowPlaying);
64         nowPlaying = strdup(a2dp.trackTitle());
65         Serial.printf("NOW PLAYING: %s\n", nowPlaying);
66     }
67 }
```

Include two libraries: "BluetoothAudio.h" and "PWMAudio.h".

1	#include <BluetoothAudio.h>
2	#include <PWMAudio.h>

Any concerns? ✉ support@freenove.com

Before using the functions in these two libraries, it is necessary to construct objects of the PWMAudio class and the A2DPSink class.

```
4  PWMAudio pwm;
5  A2DPSink a2dp;
```

Define the playback status of Bluetooth audio, the default is stop status.

```
8  volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
```

Write a volume callback function that is called when the volume changes and prints the volume percentage.

```
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%%\n", pct);
14 }
```

Write a connection callback function to print the Bluetooth device address when connected to Pico W Bluetooth.

```
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
```

Write a playback status Callback function to control the playback status of Bluetooth audio.

```
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }
```

The steps for creating A2DP Bluetooth audio are:

1. Set the Bluetooth name and MAC address.
2. Set the audio consumer to PWM audio output.
3. Enable callbacks for volume, connection, and playback status.
4. Initialize A2DP.

```
40 a2dp.setName("PicoW Boom 00:00:00:00:00:00");
41 // Set the audio consumer to PWM audio output
42 a2dp.setConsumer(new BluetoothAudioConsumerPWM(pwm));
43 a2dp.onVolume(volumeCB);
44 a2dp.onConnect(connectCB);
45 a2dp.onPlaybackStatus(playbackCB);
46 a2dp.begin();
```



Reference

Class A2DPSink

SetName(const char *name): Sets the Bluetooth module name and MAC address.

SetConsumer(BluetoothAudioConsumer_ *c): Sets the audio consumer as an A2DP receiver.

onVolume(void (*cb)(void *, int), void *cbData = nullptr): Enables the Bluetooth audio volume Callback.

onConnect(void (*cb)(void *, bool), void *cbData = nullptr): Enables the Bluetooth audio connection Callback.

onPlaybackStatus(void (*cb)(void *, PlaybackStatus), void *cbData = nullptr): Enables the Bluetooth audio playback status Callback.

begin(): Initializes A2DP.

play(): Changes the Bluetooth audio playback status to playing.

pause(): Changes the Bluetooth audio playback status to paused.

trackTitle(): Obtains the title of the current Bluetooth audio track.

What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions, or if you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.

<https://www.freenove.com/>

Thank you again for choosing Freenove products.