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.

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

Any concerns? ✉ support@freenove.com

Contents

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 haven't downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:
https://github.com/Freenove/Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico

In this tutorial, we devide 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.

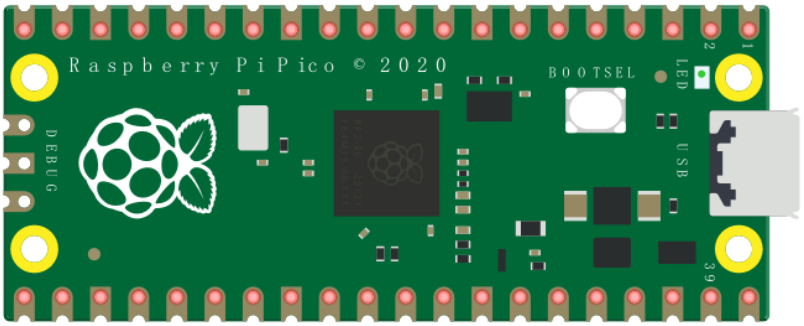
After completing the projects in this tutorial, you can also combine the components in different projects to make your own smart homes, smart car, robot, etc., bringing your imagination and creativity to life with Raspberry Pi Pico.

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

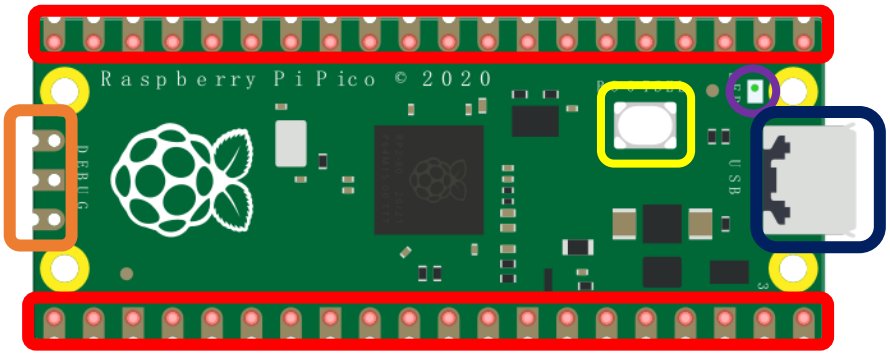
Raspberry Pi Pico

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.

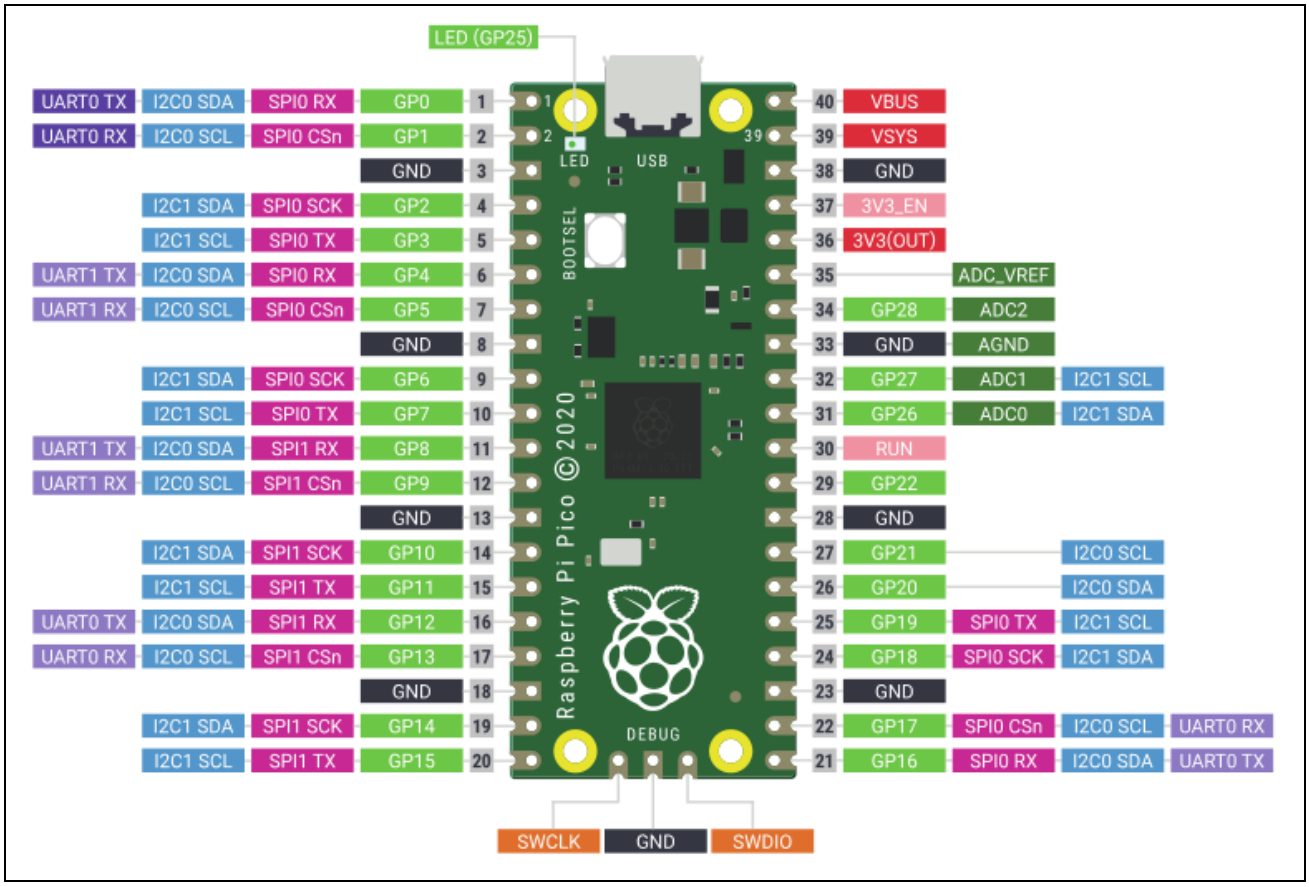


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSE button
	USB port
	LED
	Debugging

Function definition of pins:



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

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

UART, I2C, SPI Defalt 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

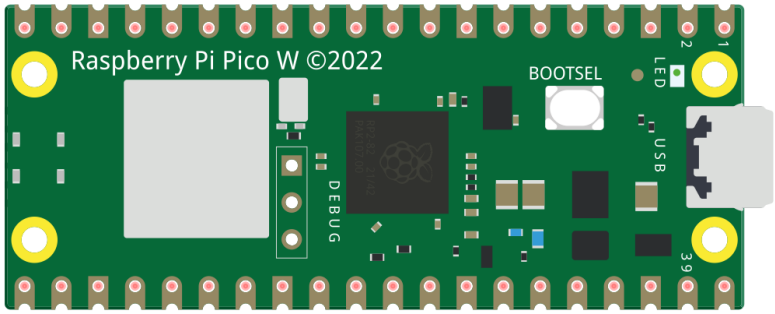
SPI

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

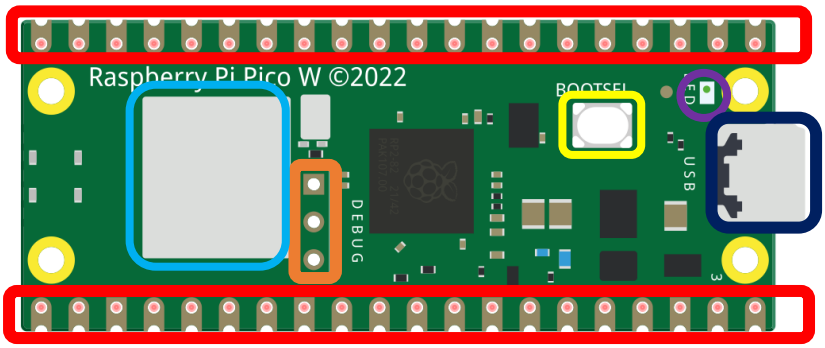
Raspberry Pi Pico W

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

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

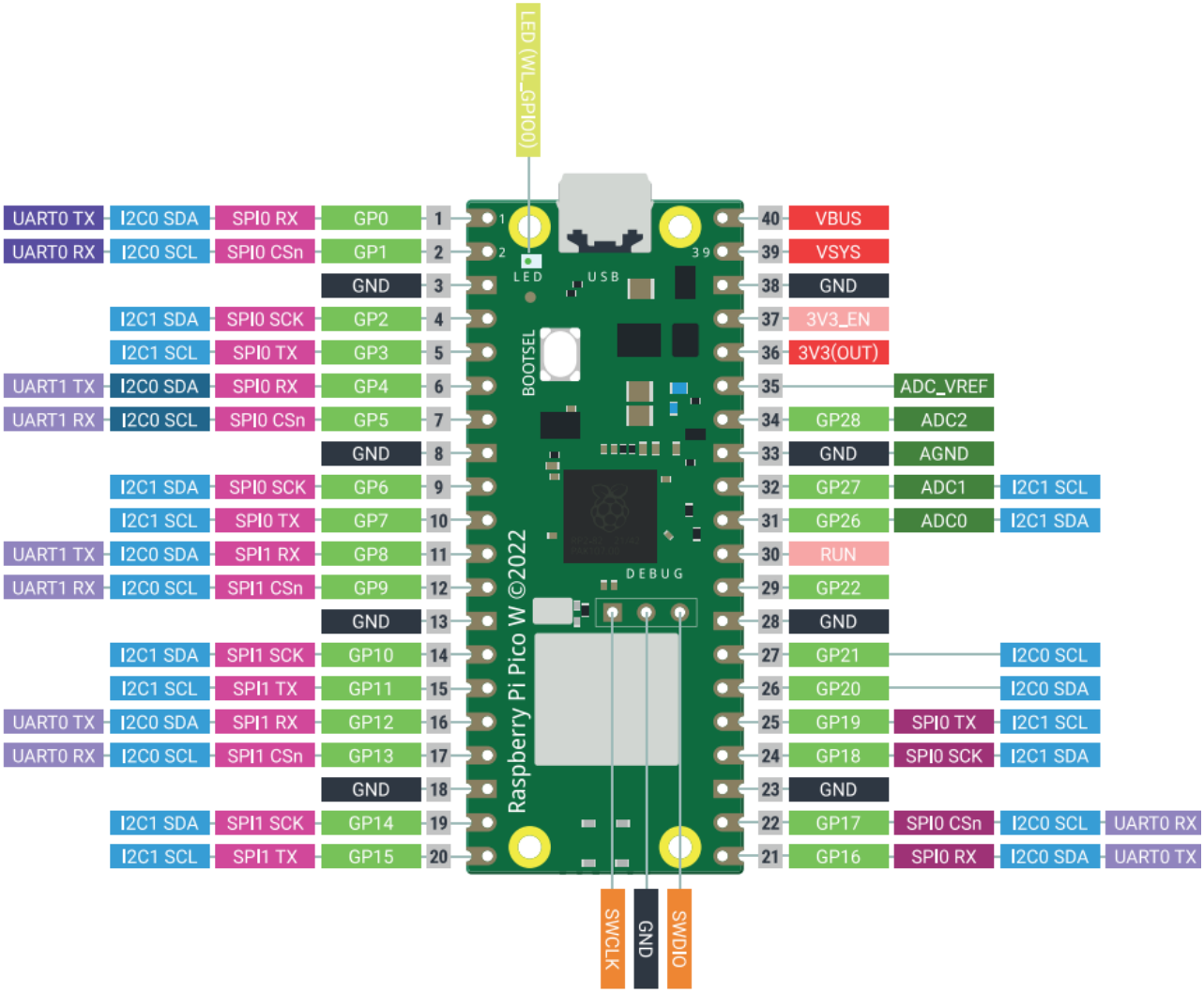


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSE button
	USB port
	LED
	Debugging
	Wireless

Function definition of pins:



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

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

UART, I2C, SPI, Wireless Defalt 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

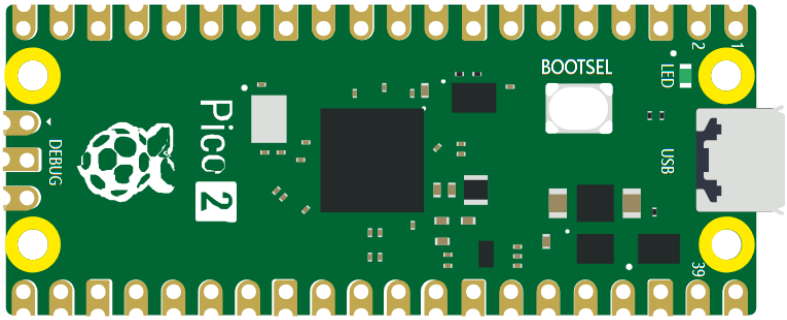
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Wireless

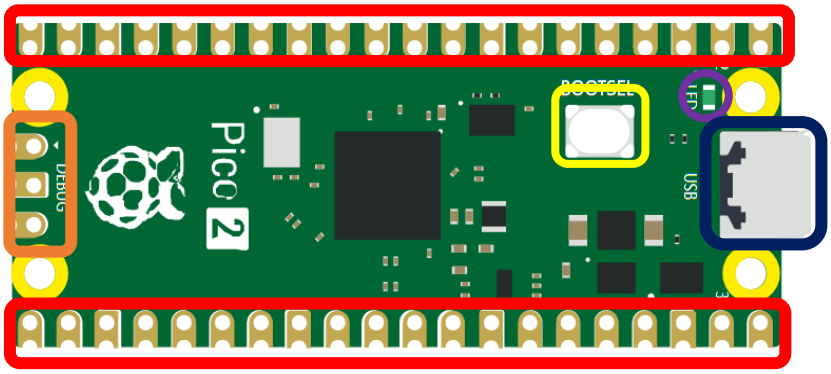
Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

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, whiich 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.

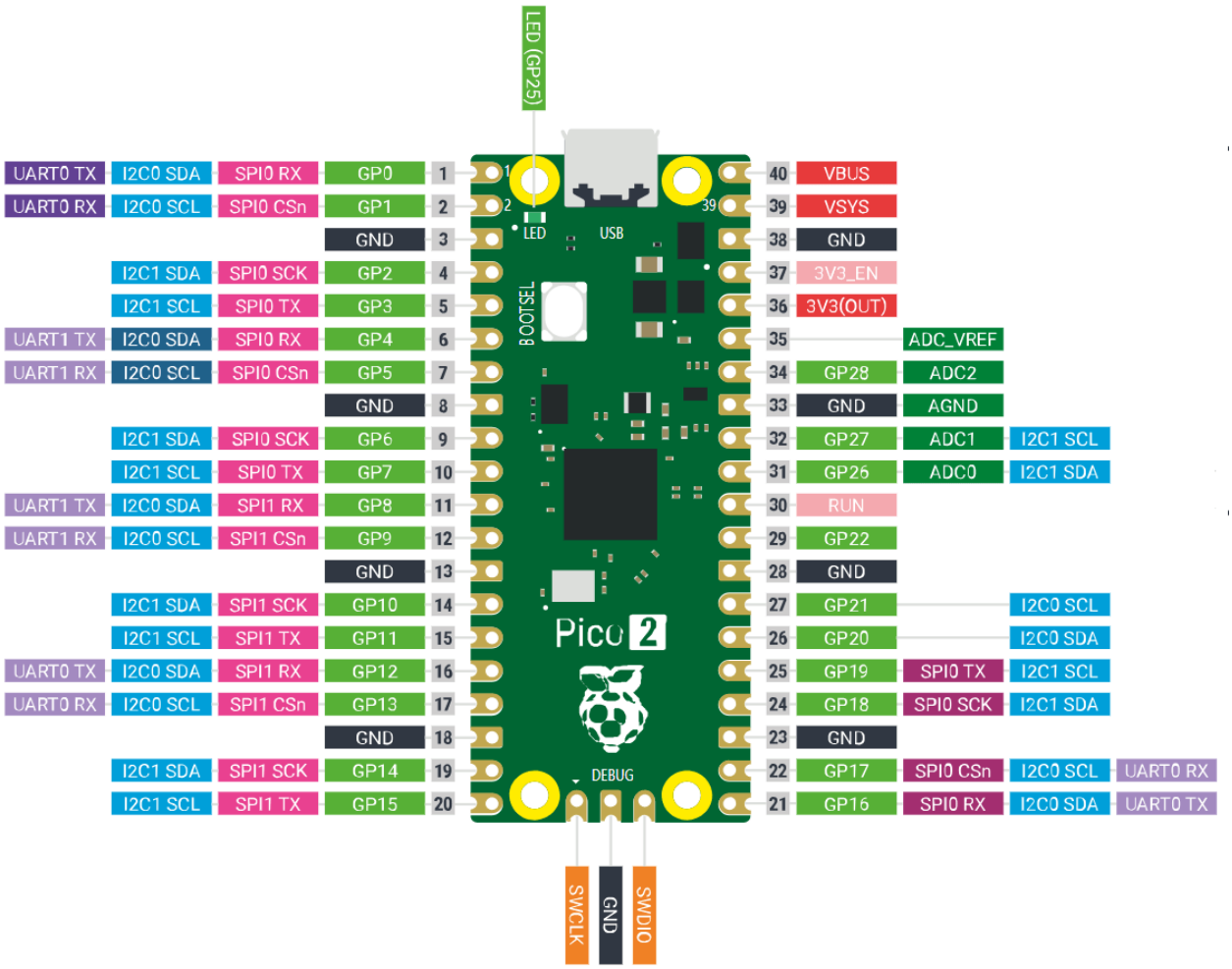


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSE button
	USB port
	LED
	Debugging

Function definition of pins:



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

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

UART, I2C, SPI Defalt 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

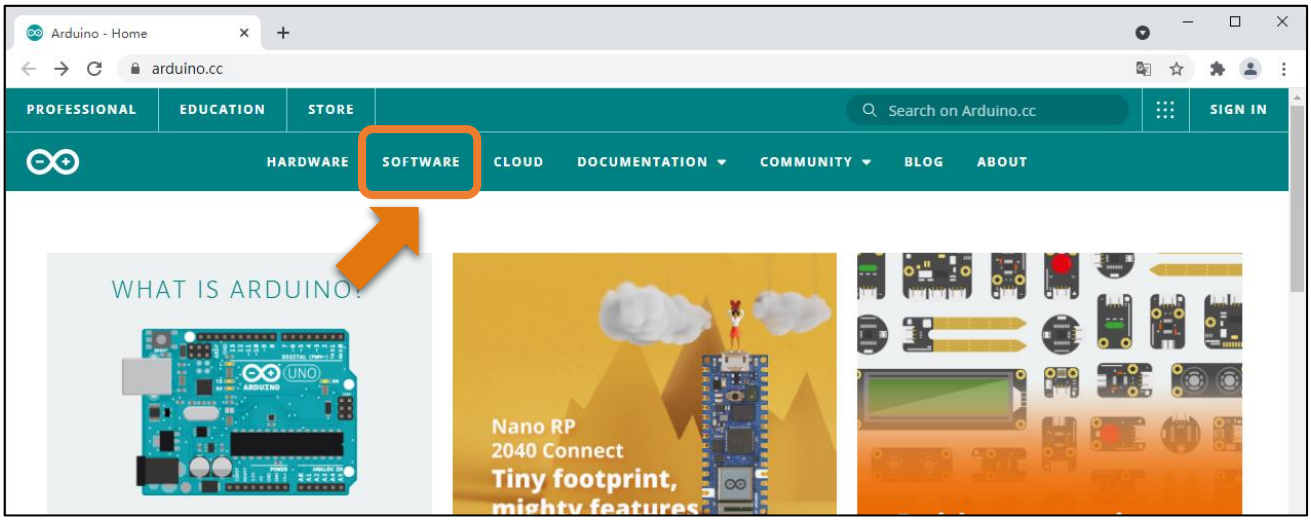
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Chapter 0 Getting Ready (Important)

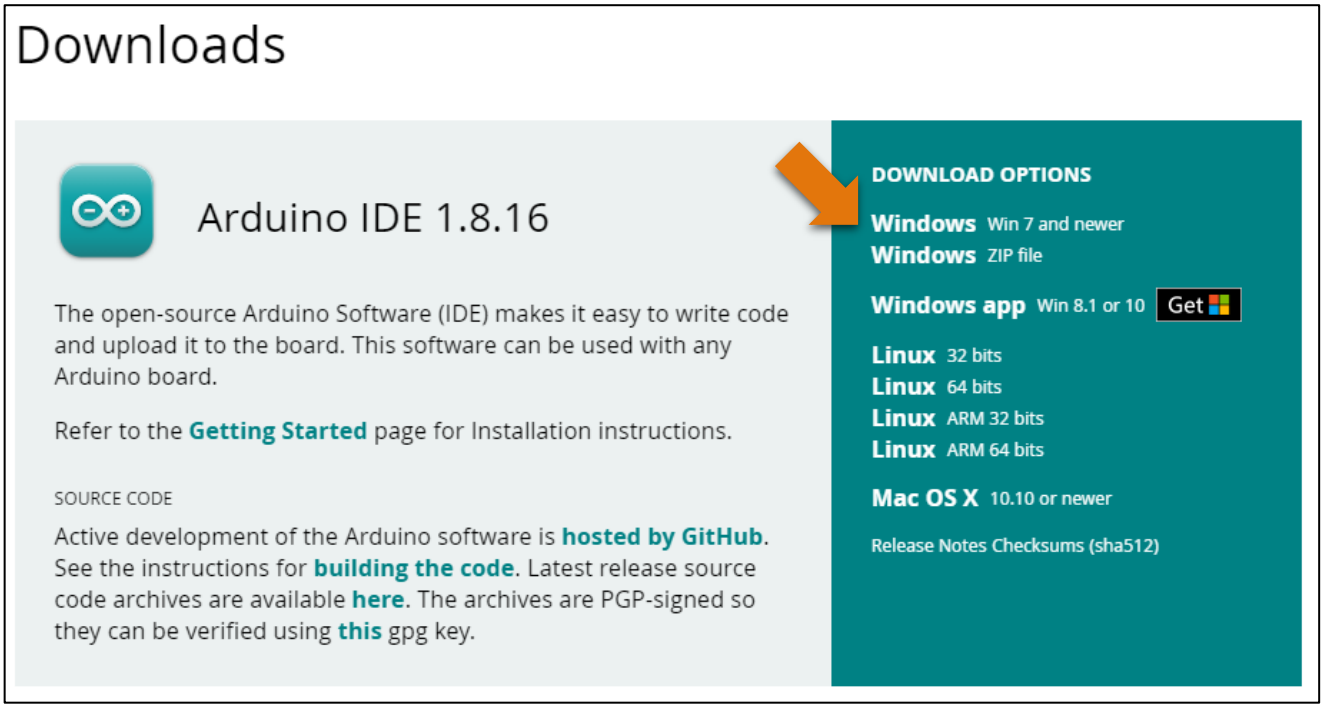
Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

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.

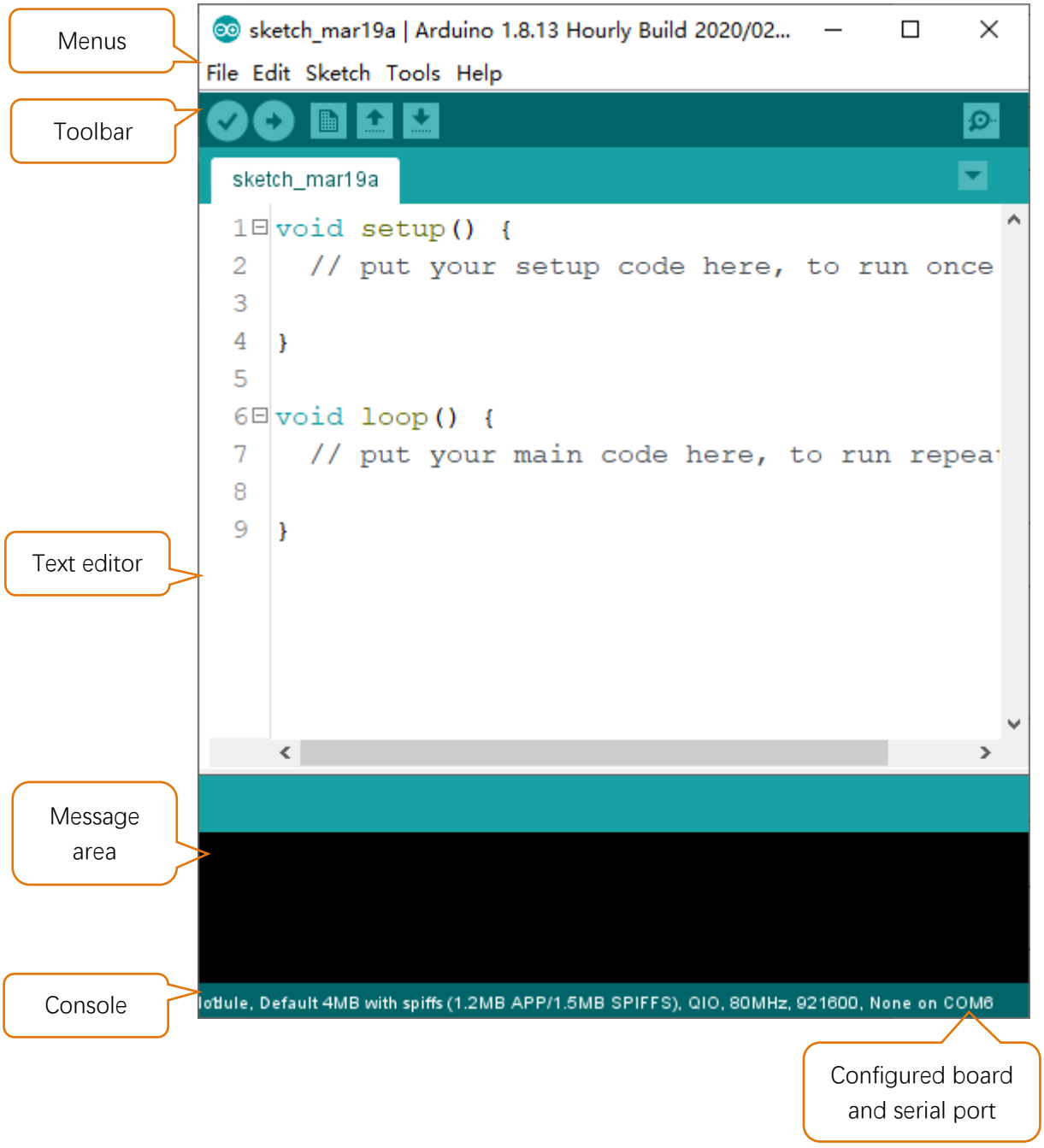


Select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer" to download to install the driver correctly.



After the download completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it popes up, please allow the installation.
After installation is complete, an Arduino Software shortcut will be generated in the desktop. Run the Arduino Software.

The interface of Arduino Software is as follows:



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

Verify
Check your code for compile errors .

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

New
Create a new sketch.

Open
Present a menu of all the sketches in your sketchbook. Clicking one will open it within the current window and overwrite its content.

Save
Save your sketch.

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.

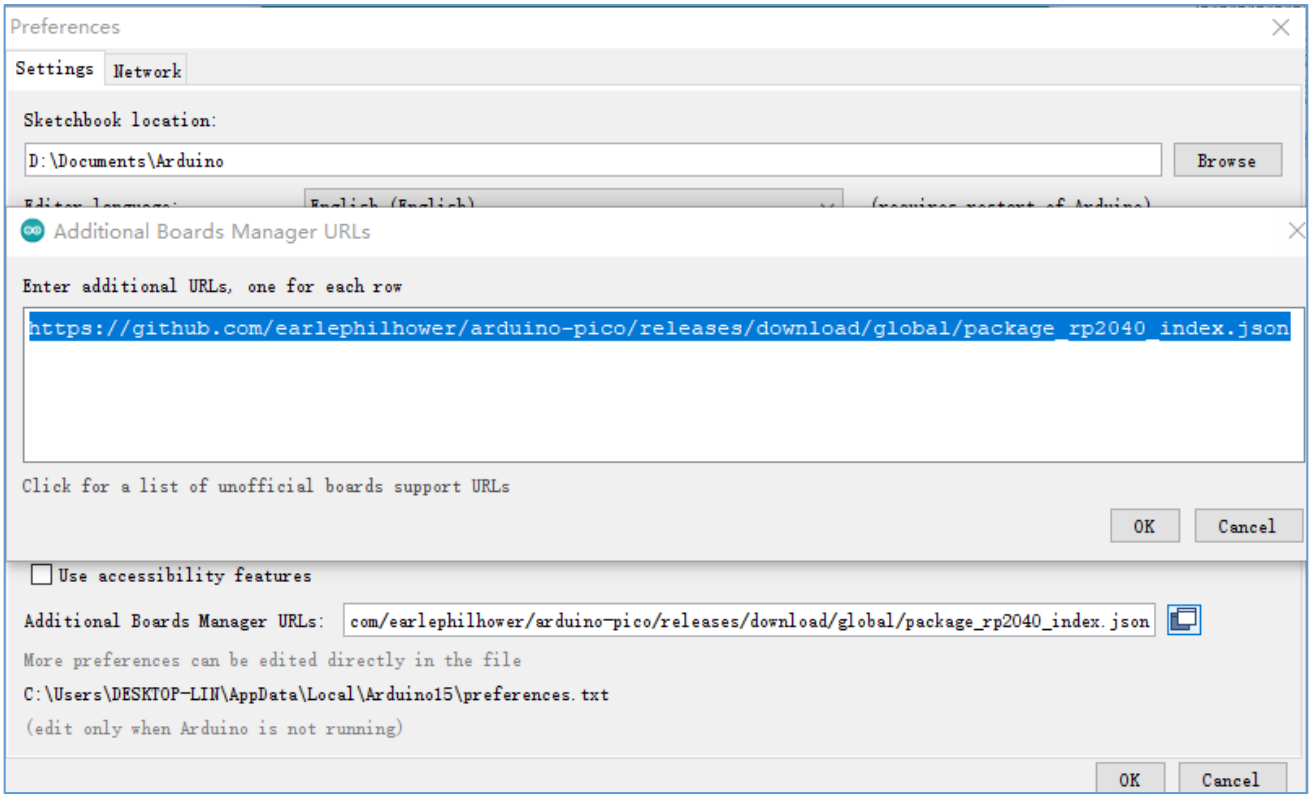
Installation of Development Board Support Package

1, Make sure your network is of good connection.

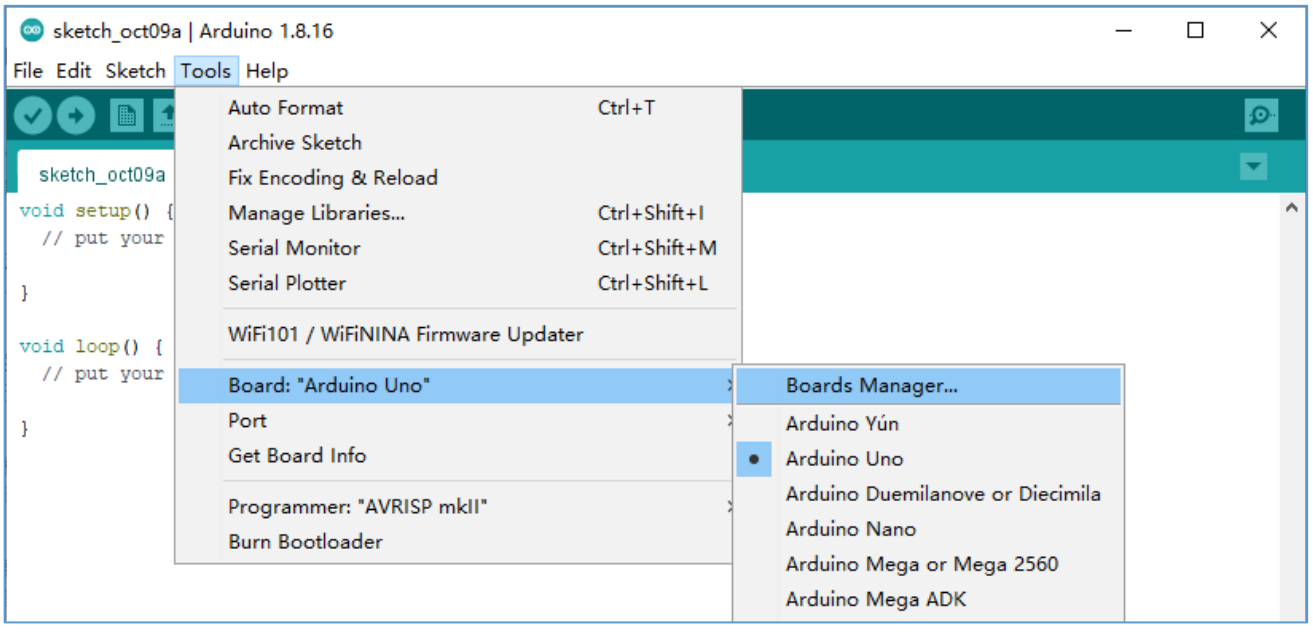
2, Open Arduino IDE, and click File>Preference. In new pop-up window, find "Additional Boards Manager URLs", and replace with a new line:

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

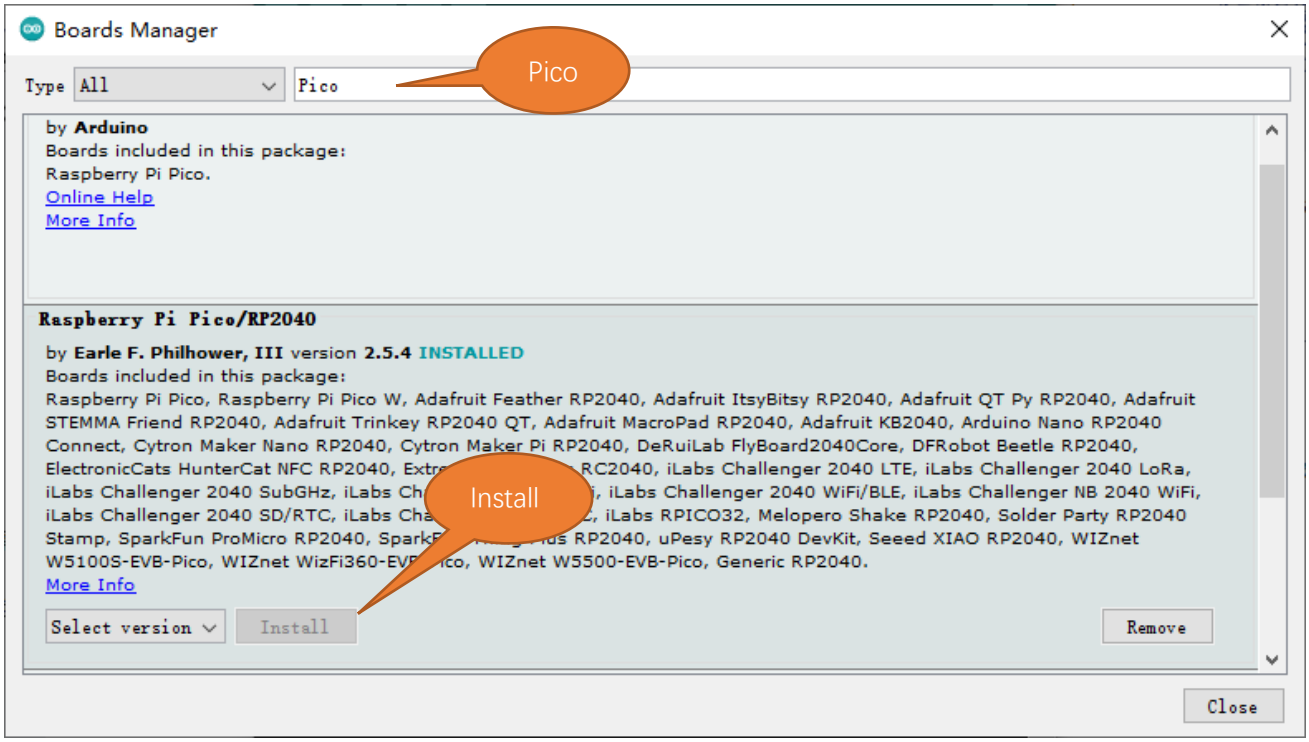
As shown below:



3, Open Arduino IDE. Click Tools>**Board**>**Boards Manager...**on the menu bar.



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

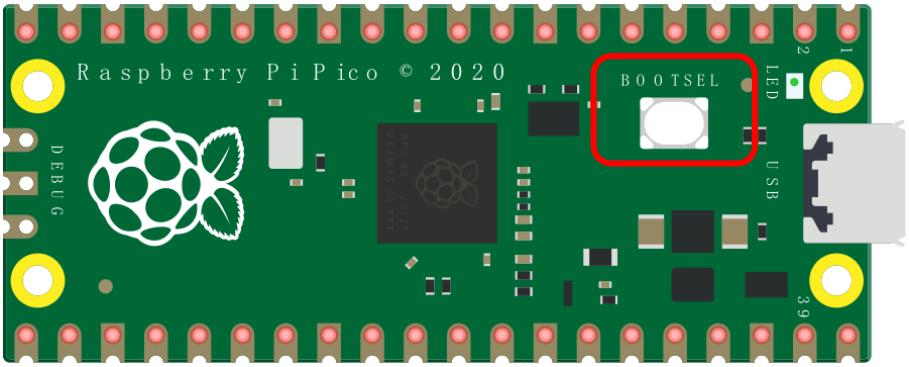


5, Click Yes in the pop-up "**dpinst-amd64.exe**"installation window. (Without it, you will fail to communicate with Arduino.) Thus far, we have finished installing the development support package.

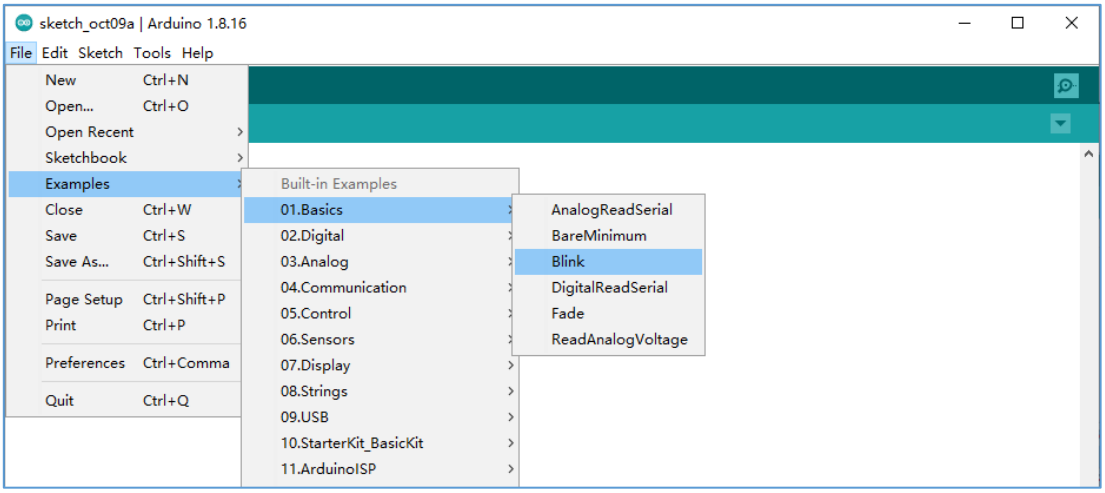
Uploading Adruino-compatible Firmware for Pico

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

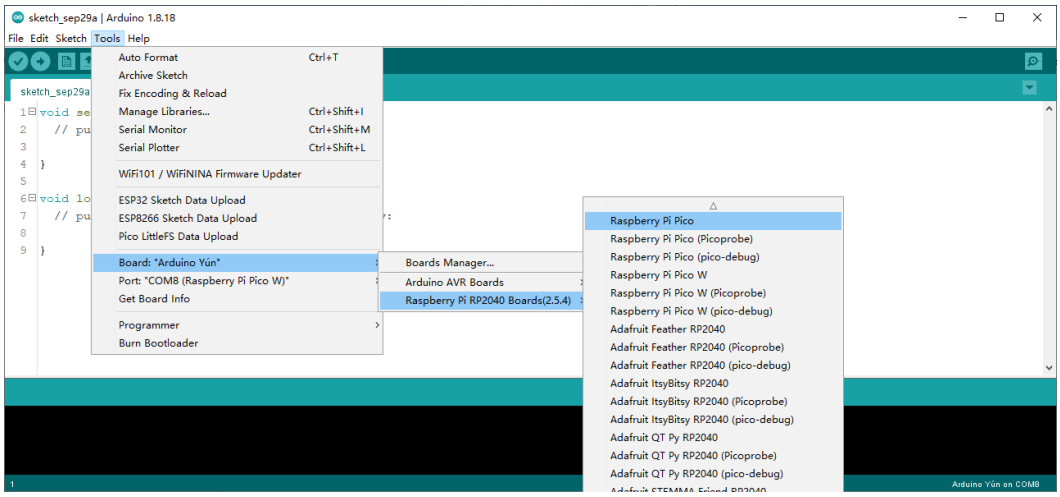
1. Disconnect Pico from computer. Keep pressing the white button (BOOTSEL) on Pico, and connect Pico to computer before releasing the button. (Note: Be sure to keep pressing the button before powering the Pico, otherwise the firmware will not download successfully)



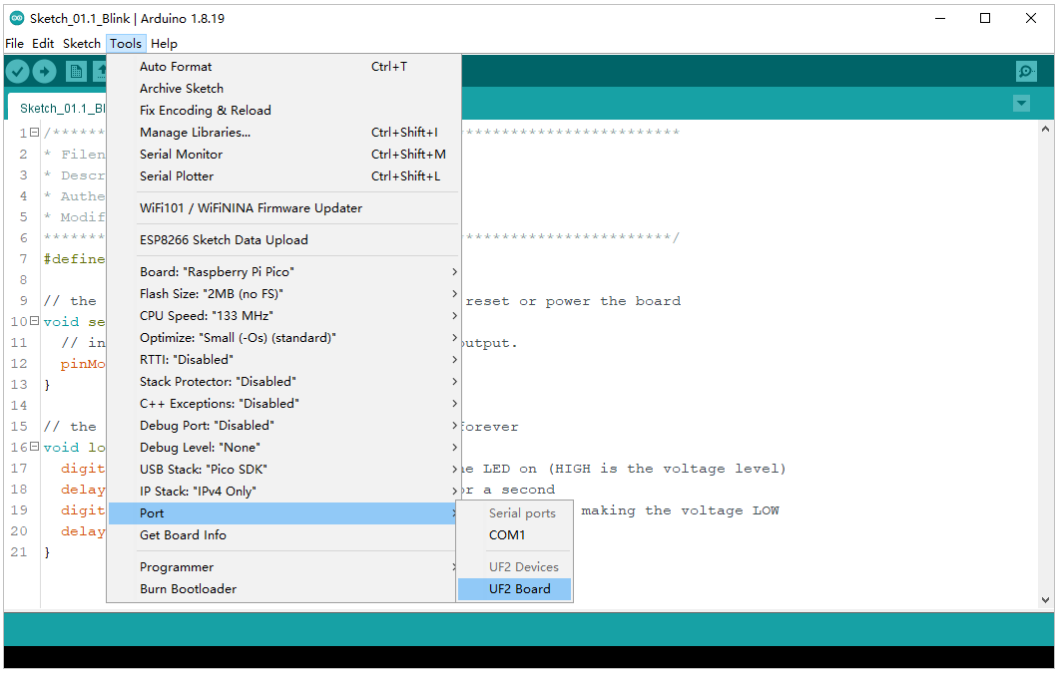
2. Open Arduino IDE. Click File>Examples>01.Basics>Blink.



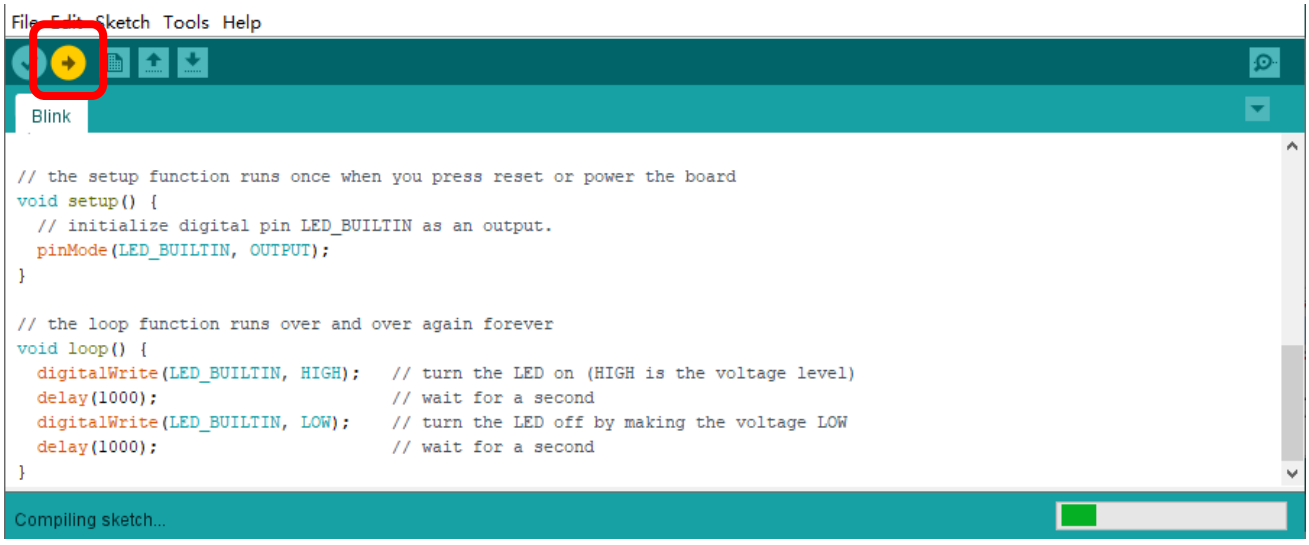
3. Click **Tools>Board>Raspberry Pi RP2040 Boards>Raspberry Pi Pico**.



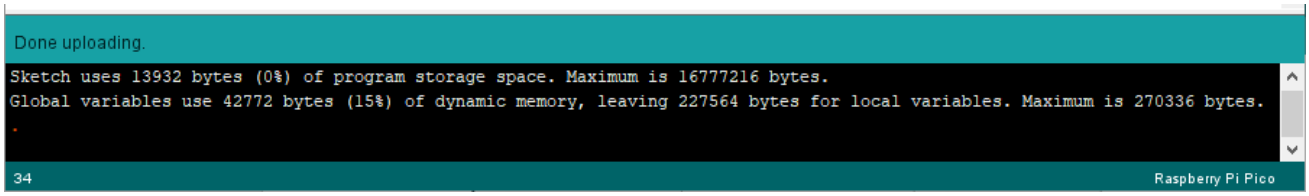
4. Click **Tools>Port>UF2 Board**.



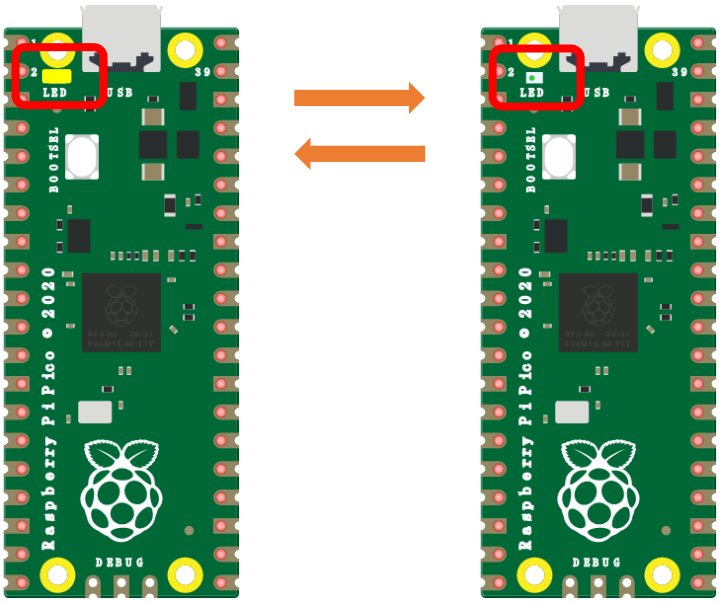
5. Upload sketch to Pico.



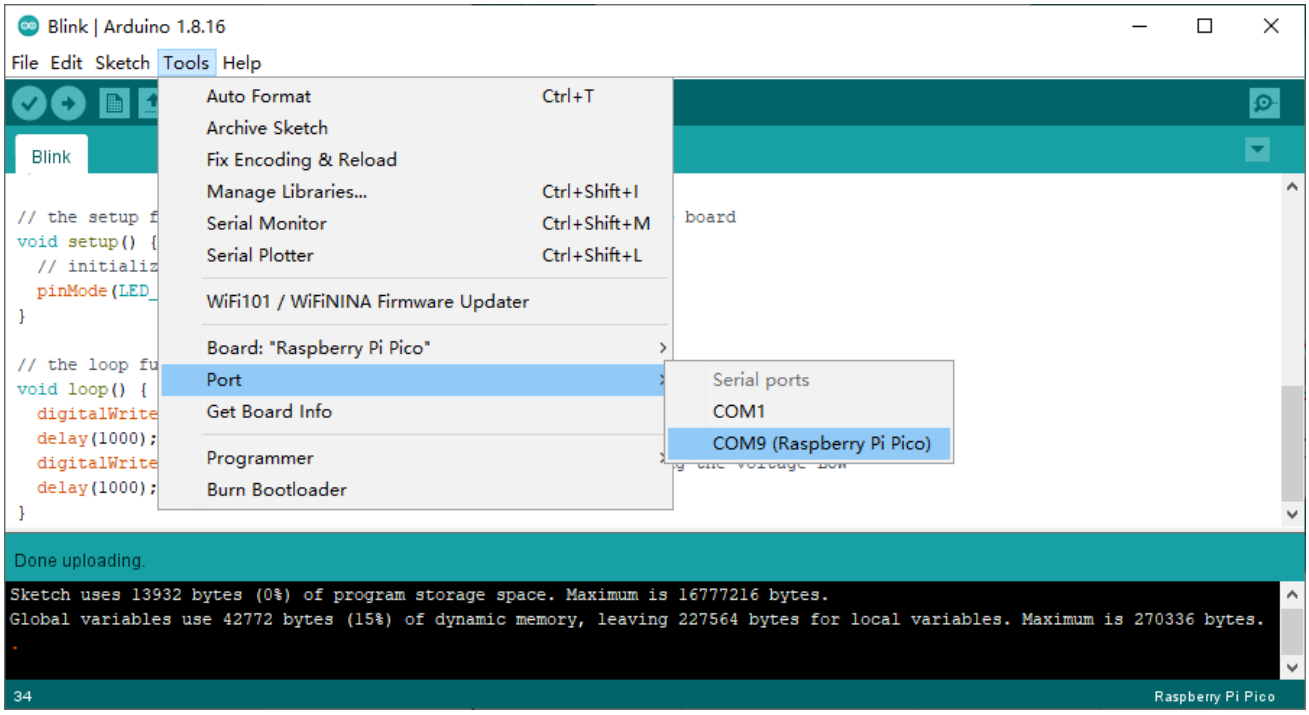
When the sketch finishes uploading, you can see the following prompt.



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 COM9.

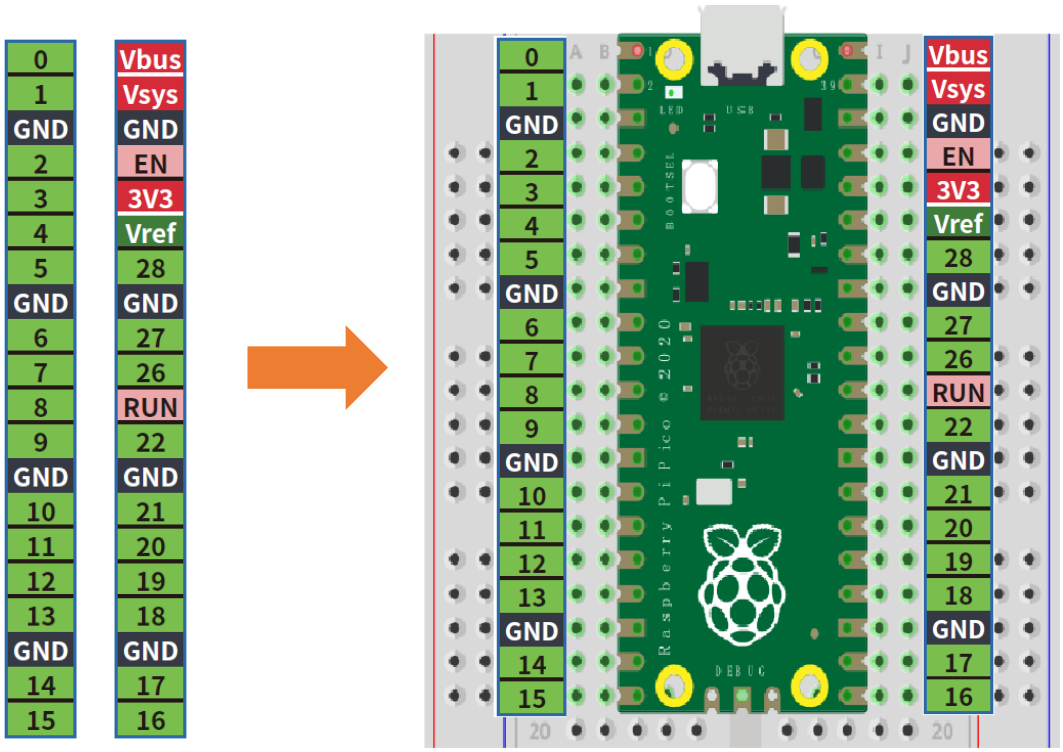


Note:

1. At the first time you use Arduino to upload sketch for Pico, you don't need to select port. After that, each time before uploading sketch, please check whether the port has beed 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.

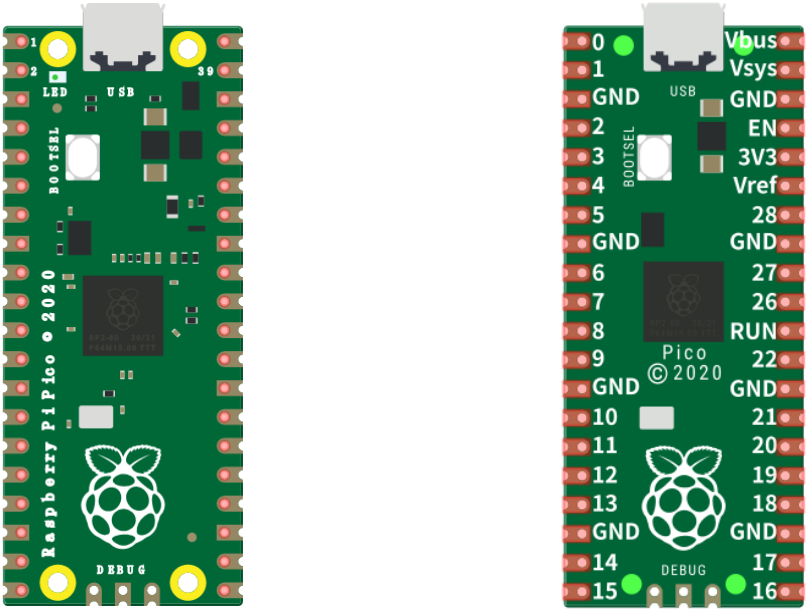
Paste the Sticker on the Breadboard

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've 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:
Rapsberry Pi Pico, Raspberry Pi Pico W, and Raspberry Pi Pico 2 are almost the same in use except for RFID and those involving WiFi function. In this book, except for the section related to WiFi, we use Raspberry Pi Pico as an example.

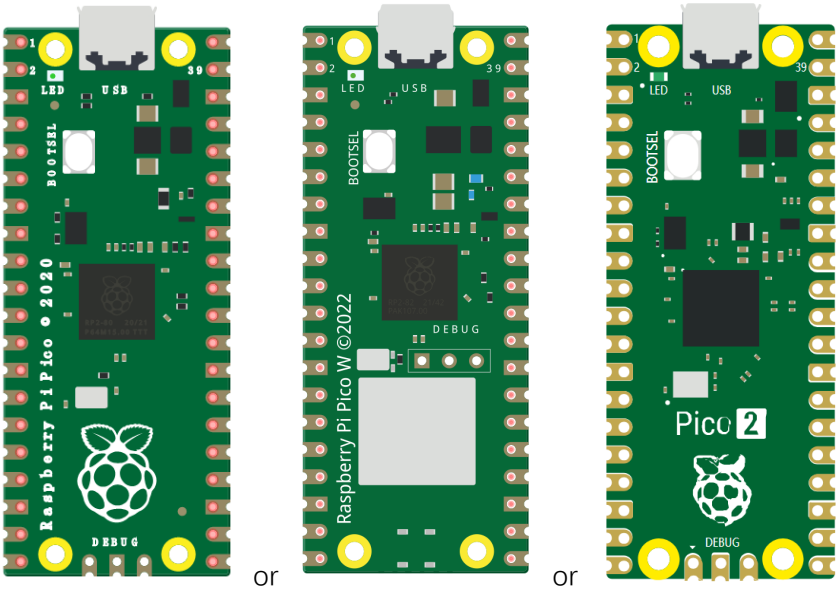
This chapter is the Start Point in the journey to build and explore Pico electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

In this project, we will use Raspberry Pi Pico to control blinking a common LED.
If you haven’t installed Arduino IDE, you can click Here.
If you haven’t uploaded firmware for Pico, you can click Here to upload.

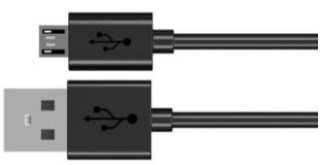
Component List

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



or or

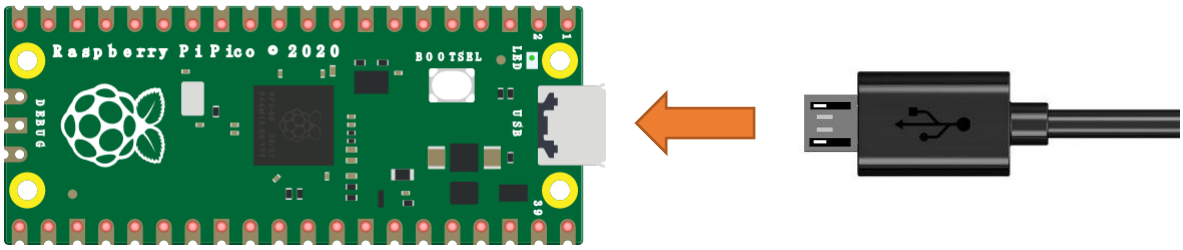
USB cable x1



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.

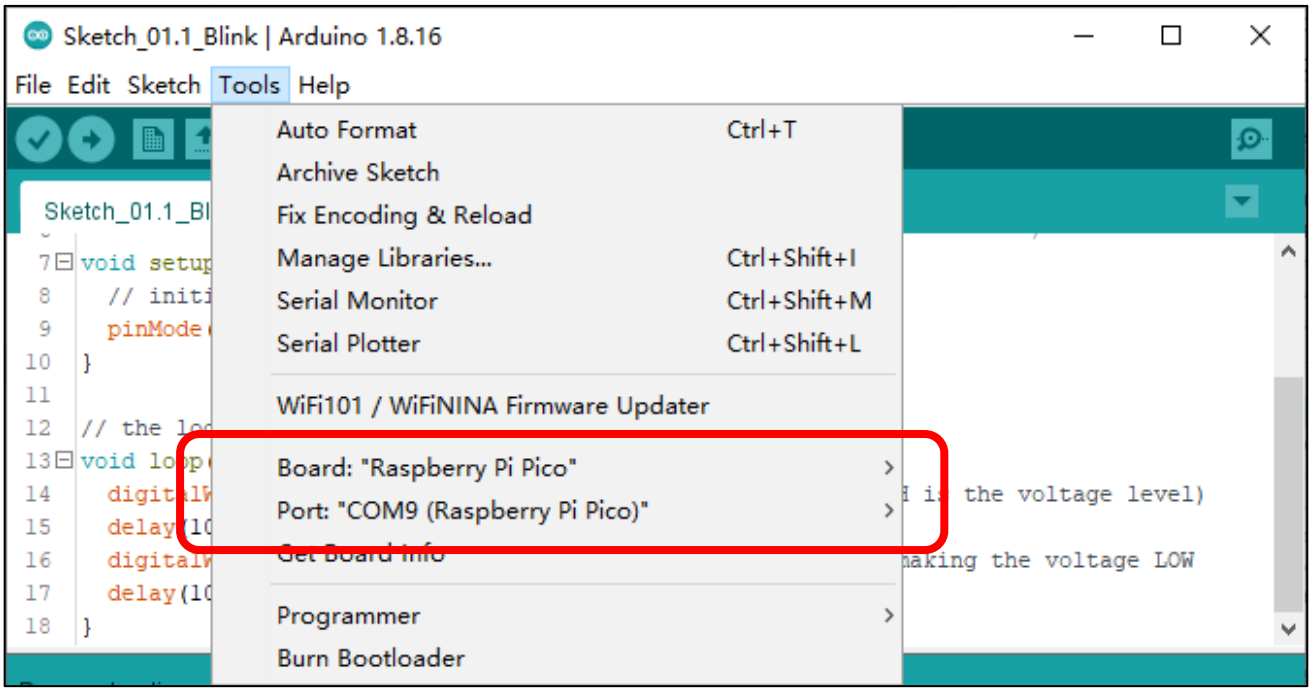


Sketch

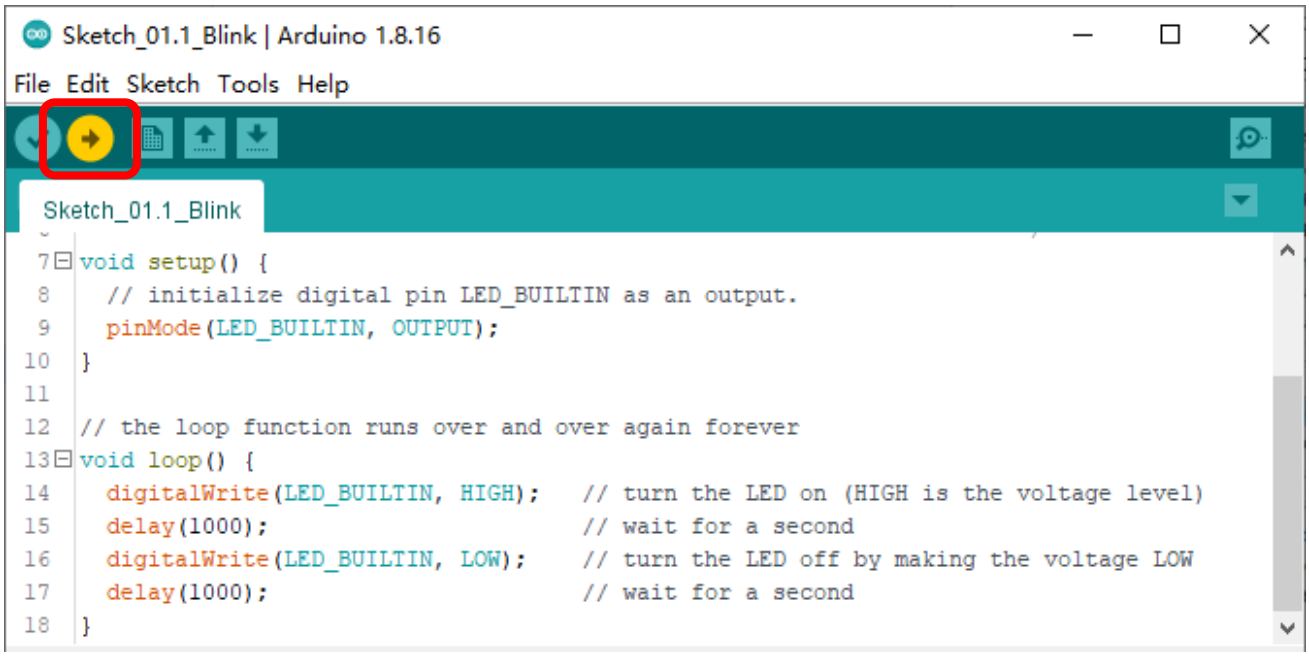
The onboard LED of Raspberry Pi Pico is controlled by GP25. When GP25 outputs high level, LED lights up; When it outputs low, LED lights off. You can open the provided code:
Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.1_Blink.

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

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

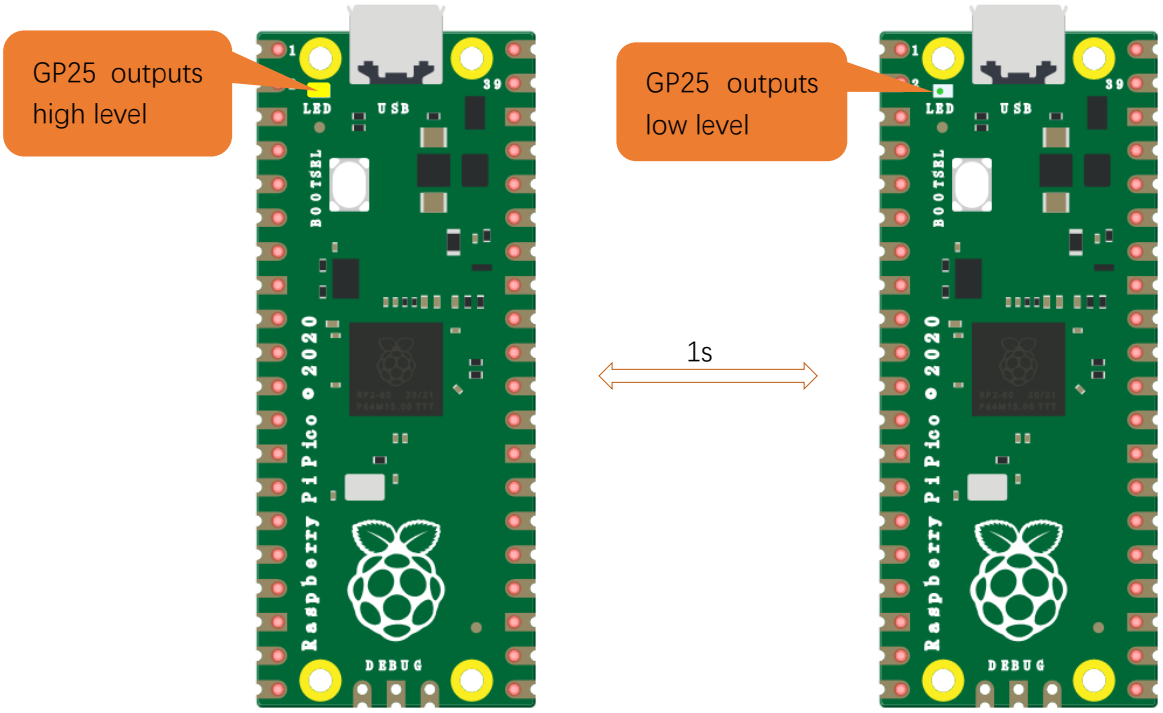


Click “Upload” to upload the sketch to Pico.



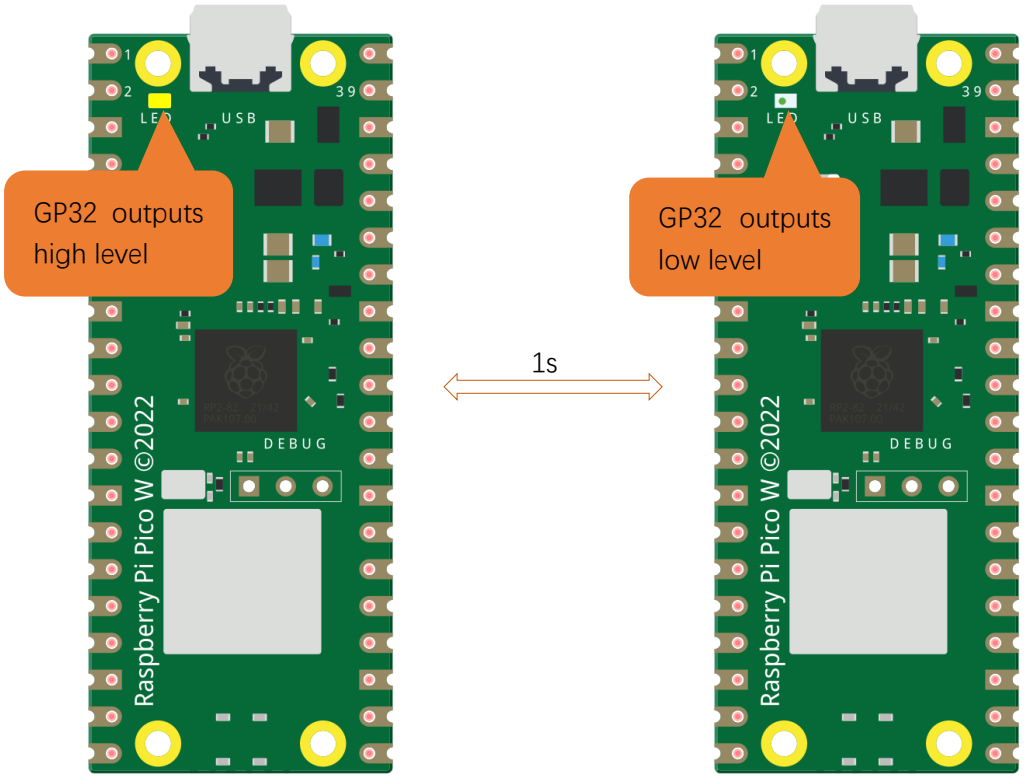
If you have any concerns, please contact us via: support@freenove.com

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.

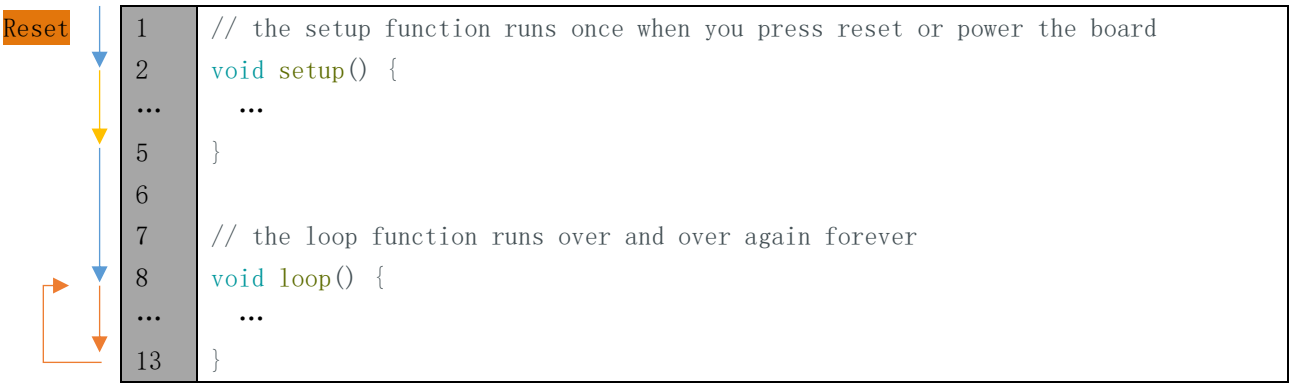
If you use Pico W, please change "# define LED_BUILTIN 25" to "# define LED_BUILTIN 32" in the code.



The following is the program code:

```
#define LED_BUILTIN 25

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

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

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

```
#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.

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

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

```
  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.

```
  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 either as an input or an output.

Parameters

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

mode: INPUT, OUTPUT, INPUT_PULLDOWM, or INPUT_PULLUP.

void digitalWrite (int pin, int value);

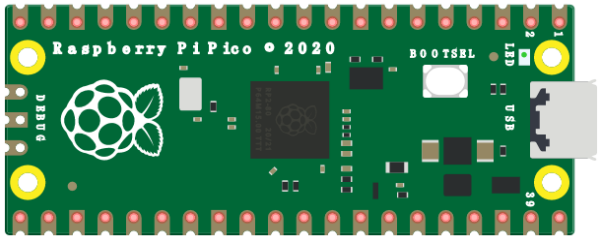
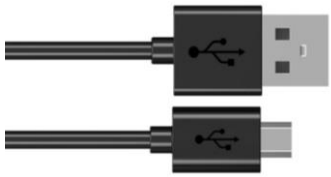
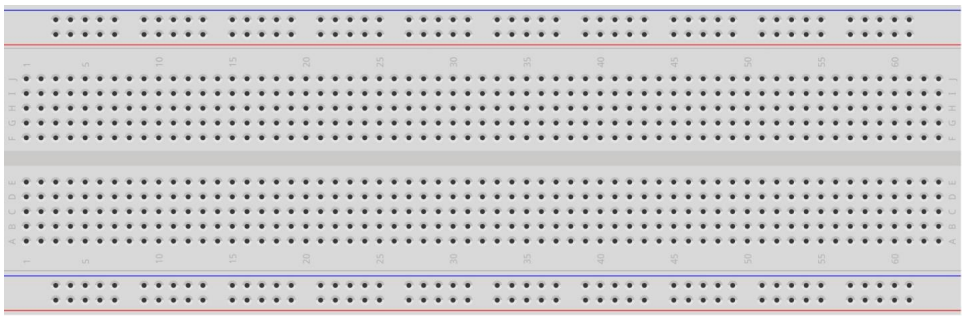
Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

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

Project 1.2 Blink

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

Component List

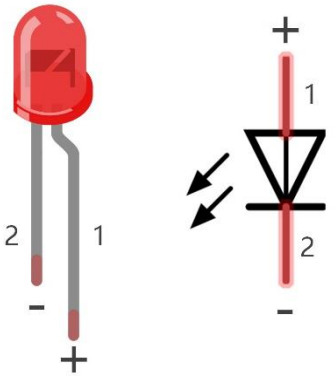
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB Cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper
  </td></tr>
</table>

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



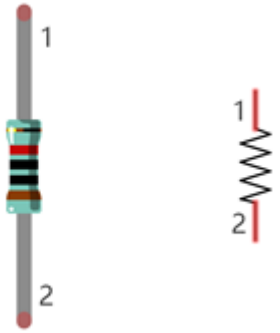
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

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

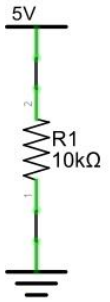
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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

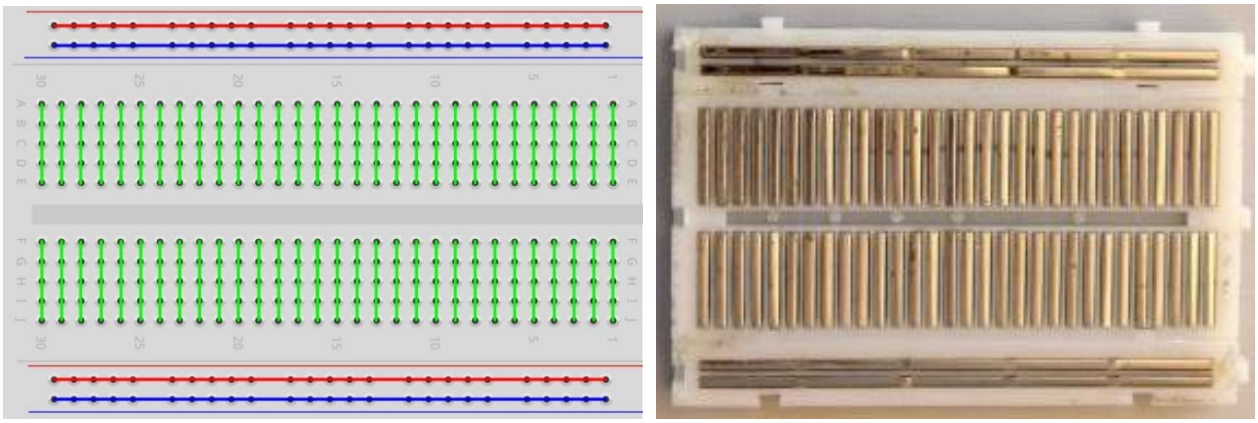


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

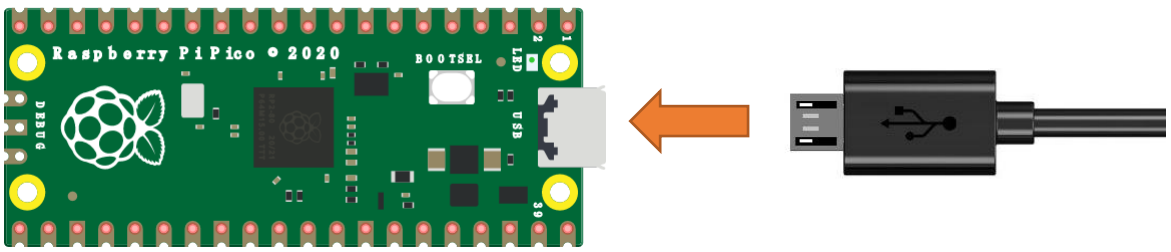
Breadboard

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



Power

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

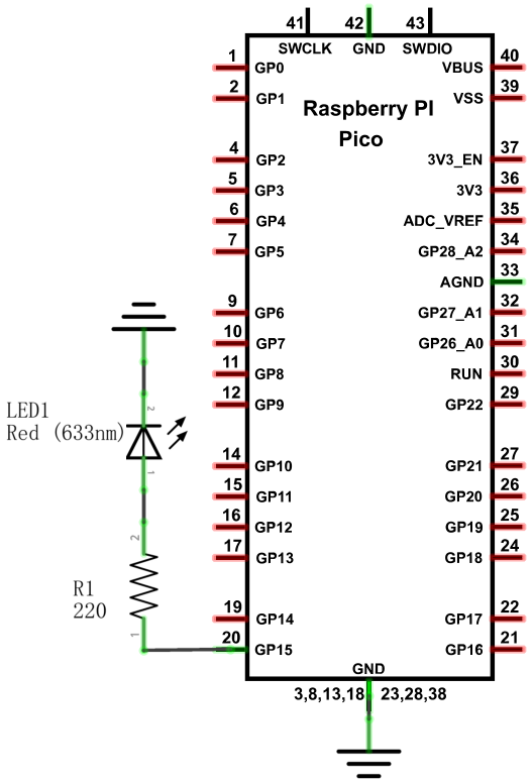


Circuit

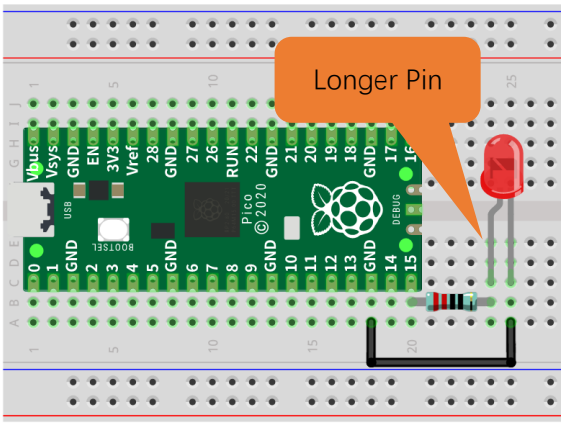
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

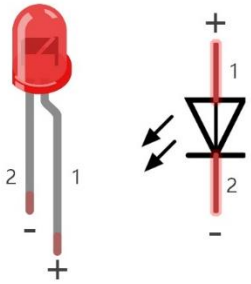
CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

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.

Sketch

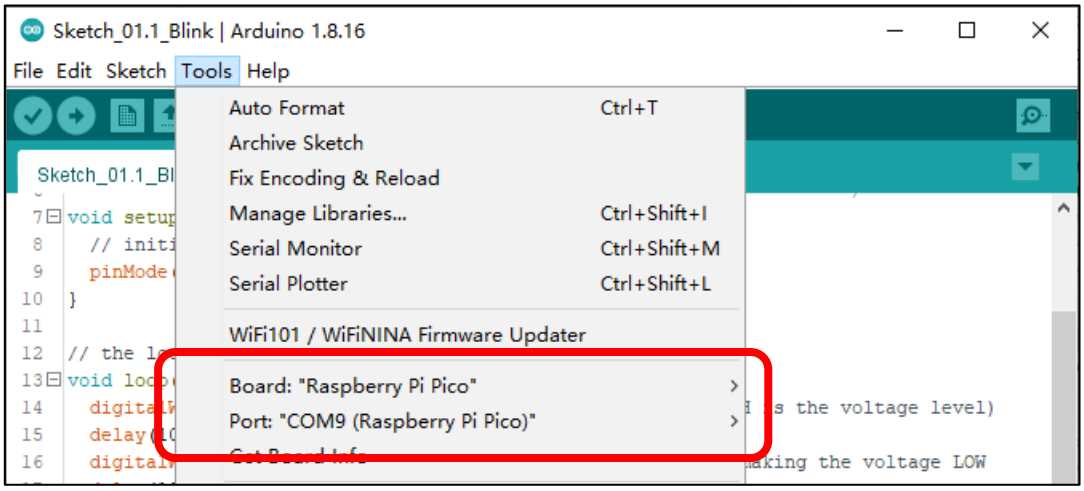
According to the circuit diagram, when GP15 of Pico outputs high level, LED lights up; when it outputs low, LED lights off. Therefore, we can make LED flash repeatedly by controlling GP15 to output high and low repeatedly.

You can open the provided code:

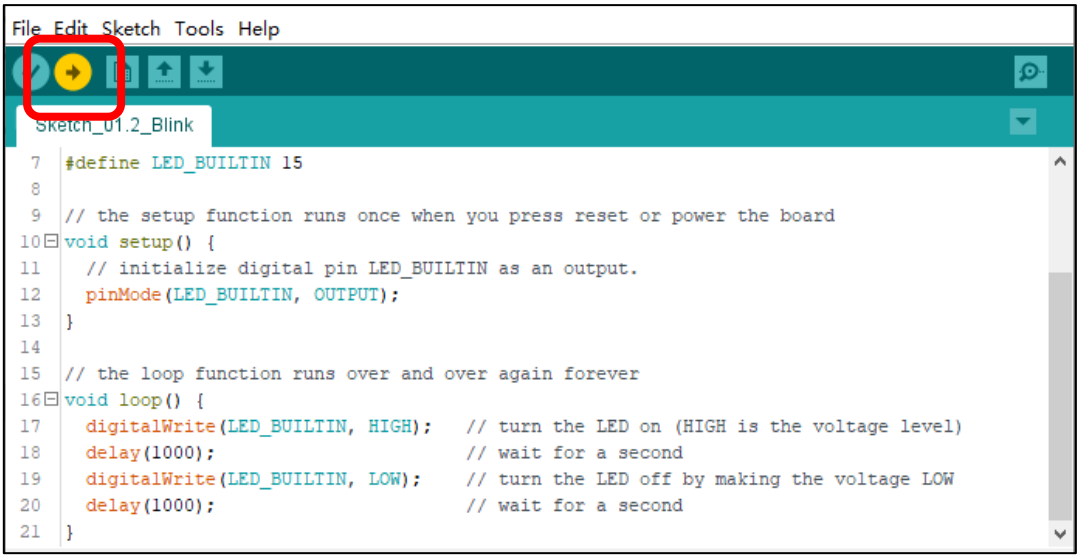
Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.2_Blink.

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

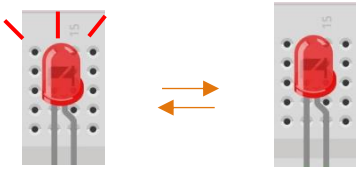
Click Tools, make sure Board and Port are as follows:



Click "Upload" to upload the sketch to Pico.



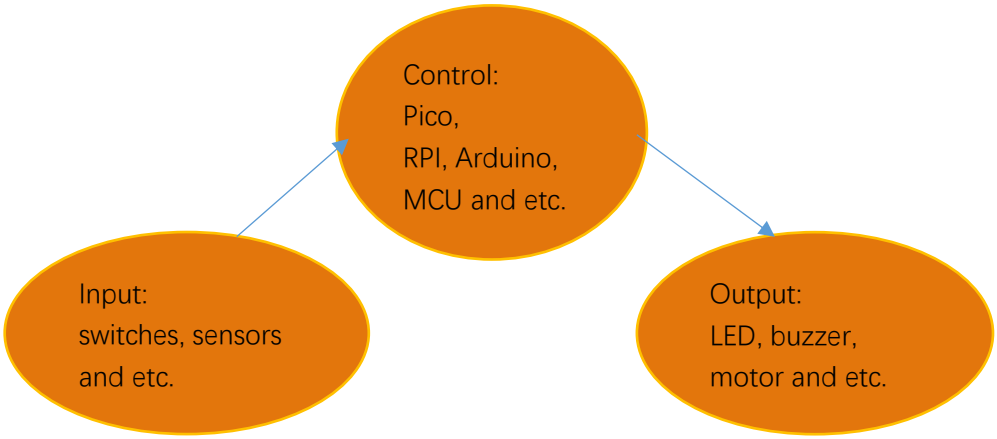
Click "Upload". Download the code to Pico and your LED in the circuit starts Blink.



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.



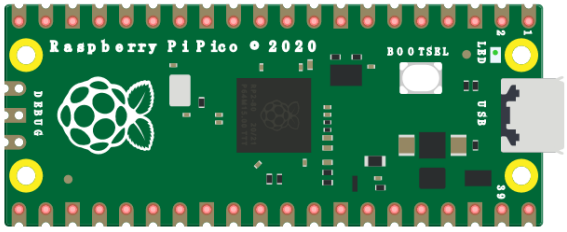
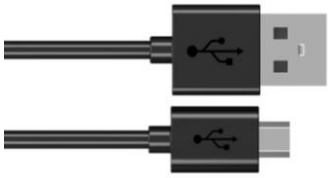
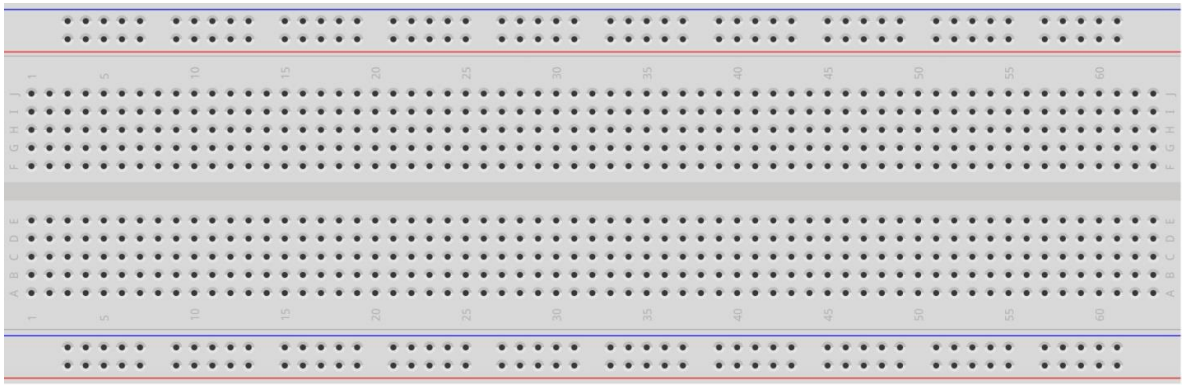
Next we make a simple project: build a control system with button, LED and Raspberry Pi Pico.
Input: Button
Control: Raspberry Pi Pico
Output: LED

Project 2.1 Button & LED

Note: Raspberry Pi Pico and Raspberry Pi Pico W 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.

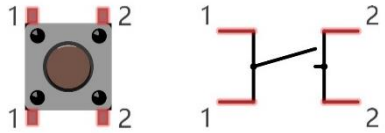
Component List

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

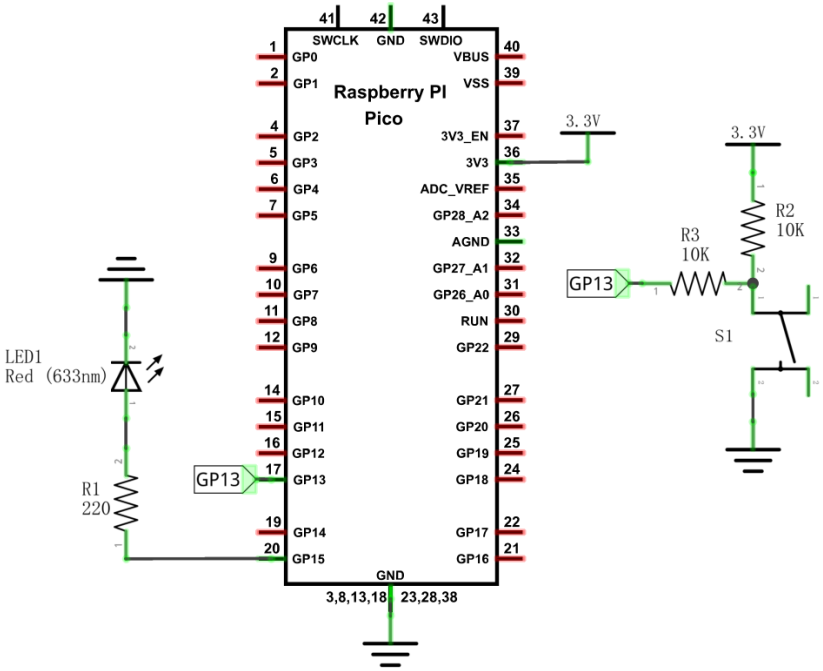
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



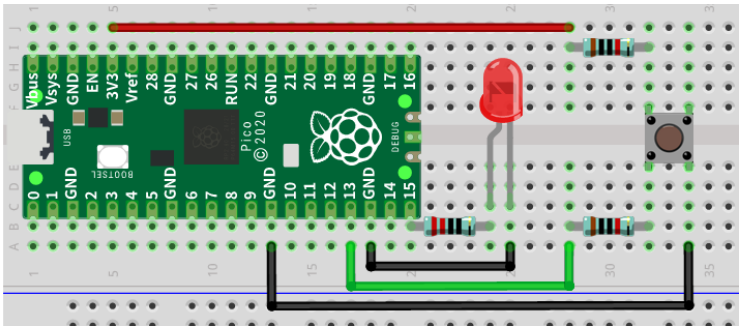
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



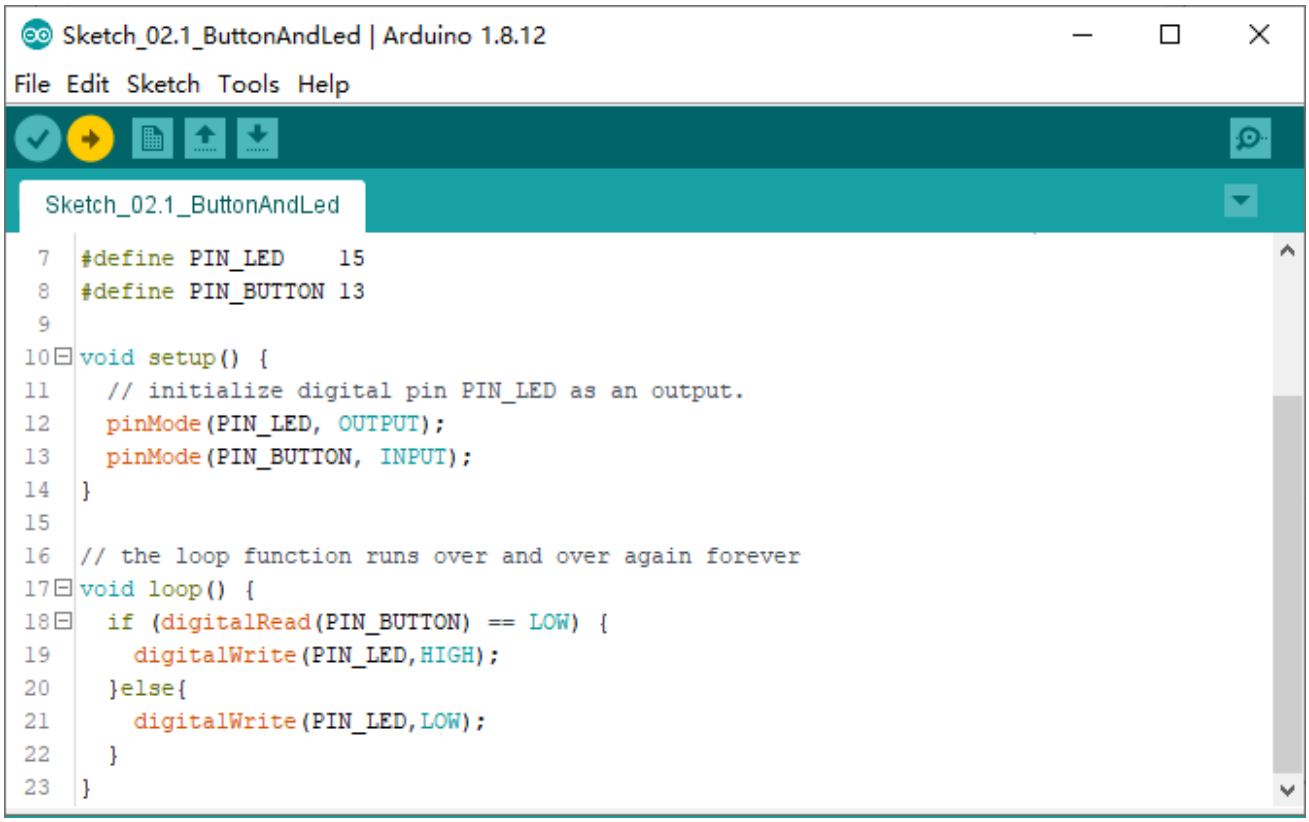
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



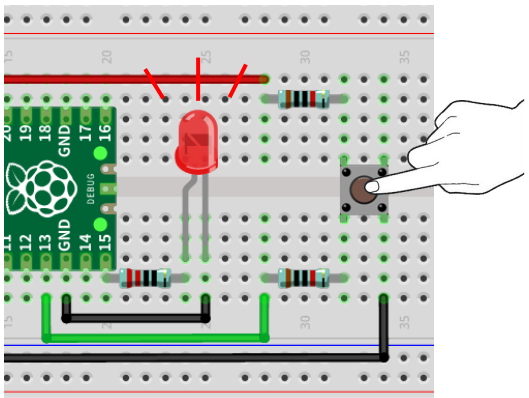
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

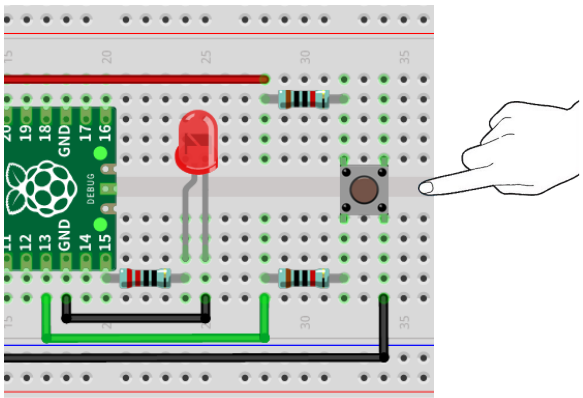
Sketch

This project is designed 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_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.1_ButtonAndLed.
Sketch_02.1_ButtonAndLed



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:

```
#define PIN_LED    15
#define PIN_BUTTON 13
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

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

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

```
#define PIN_LED    15
#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.

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

Reference

int digitalRead (int pin);

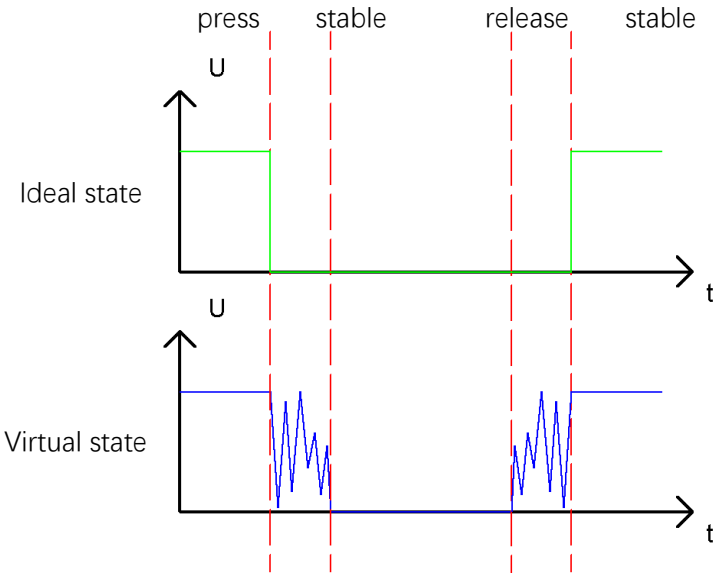
This function returns the value read at the given pin. It will be “HIGH” or “LOW”(1 or 0) depending on the logic level at the pin.

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_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.2_TableLamp.

Sketch_02.2_TableLamp

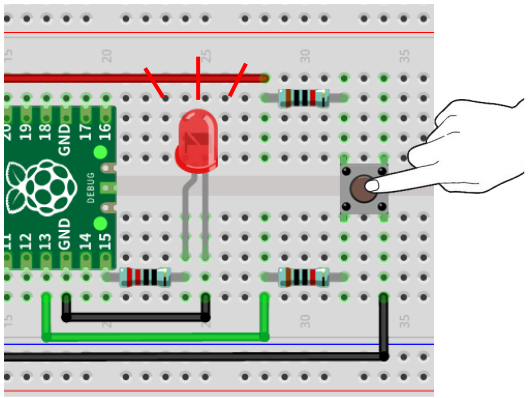
Sketch_02.2_TableLamp | Arduino 1.8.12

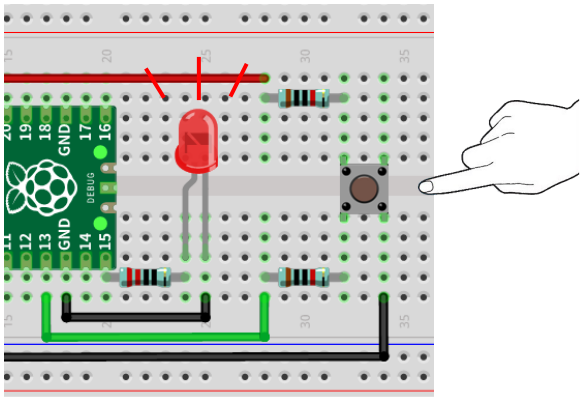
File Edit Sketch Tools Help

Sketch_02.2_TableLamp

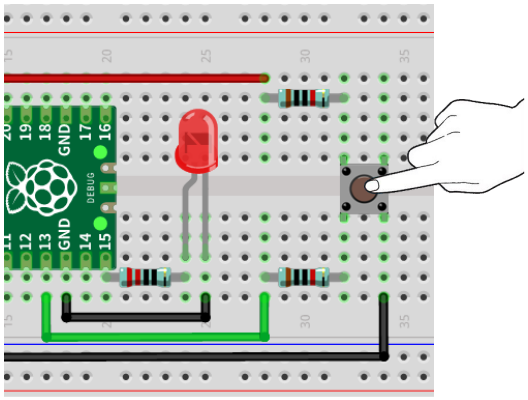
```
#define PIN_LED    15
#define PIN_BUTTON 13
bool led_state = false;

void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

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

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

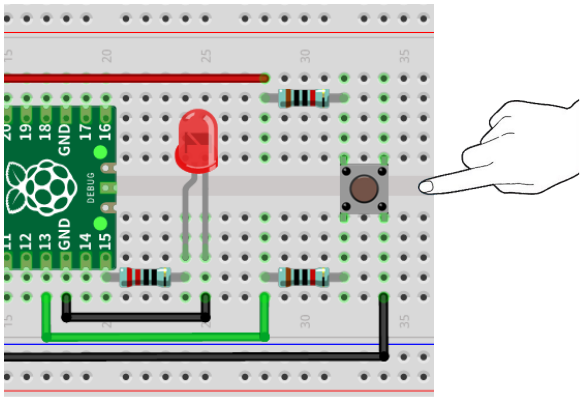
Upload the sketch to Pico. When the button is pressed, LED lights up; when the button is released, LED is still ON.





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





The following is the program code:

```
#define PIN_LED    15
#define PIN_BUTTON 13
bool ledState = false;

void setup() {
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

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

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

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

```
#define PIN_LED    15
#define PIN_BUTTON 13
```

Define a variable to store the status of LED.

```
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

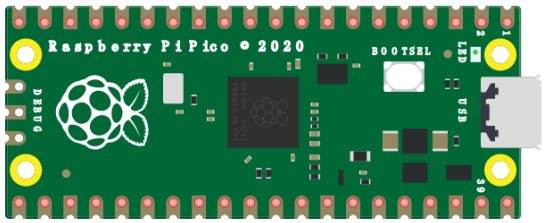
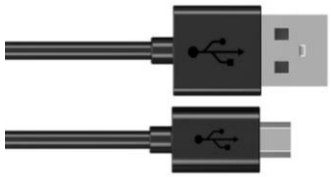
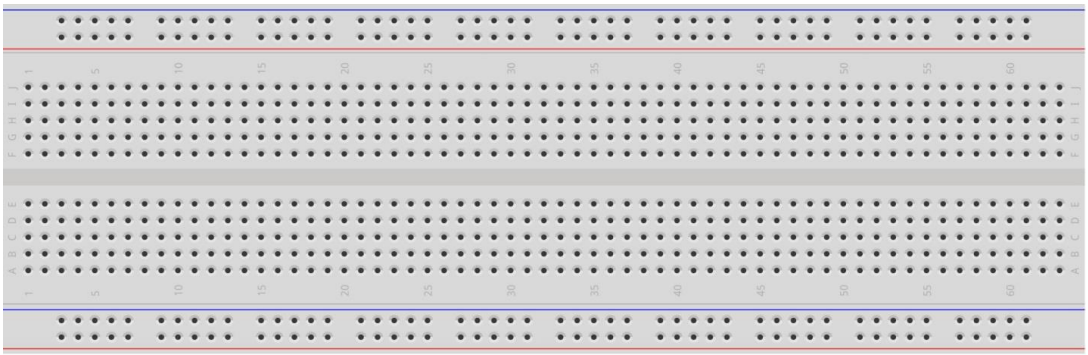
We have learned how to control an LED blinking, next we will learn how to control a number of LEDs.

Project 3.1 Flowing Light

Note: Raspberry Pi Pico and Raspberry Pi Pico W 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

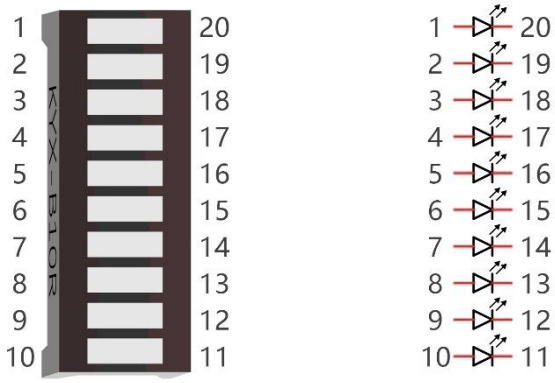
<table>
<tr><td>Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Jumper
  </td><td>LED bar graph x1
  </td><td>Resistor 220Ω x10
  </td></tr>
</table>

Component Knowledge

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

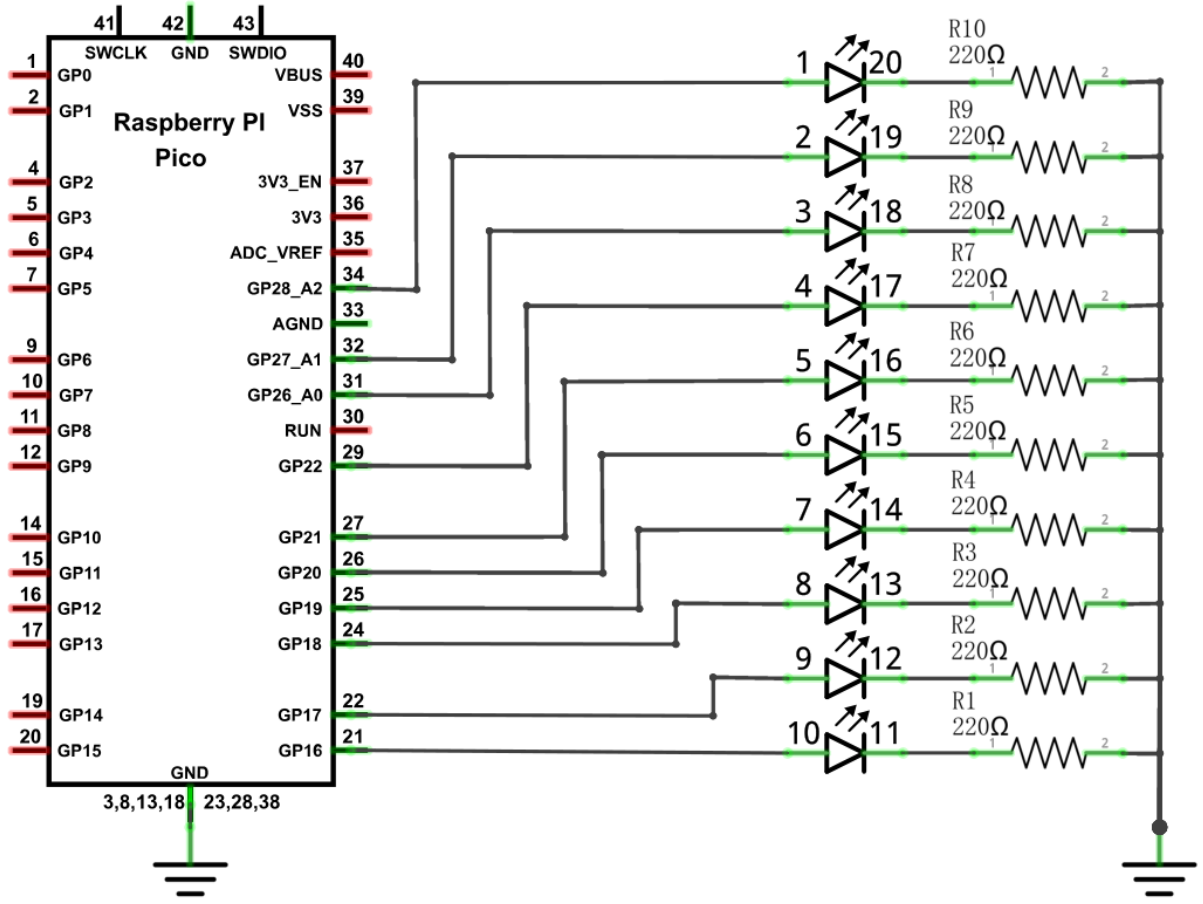
LED bar

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.

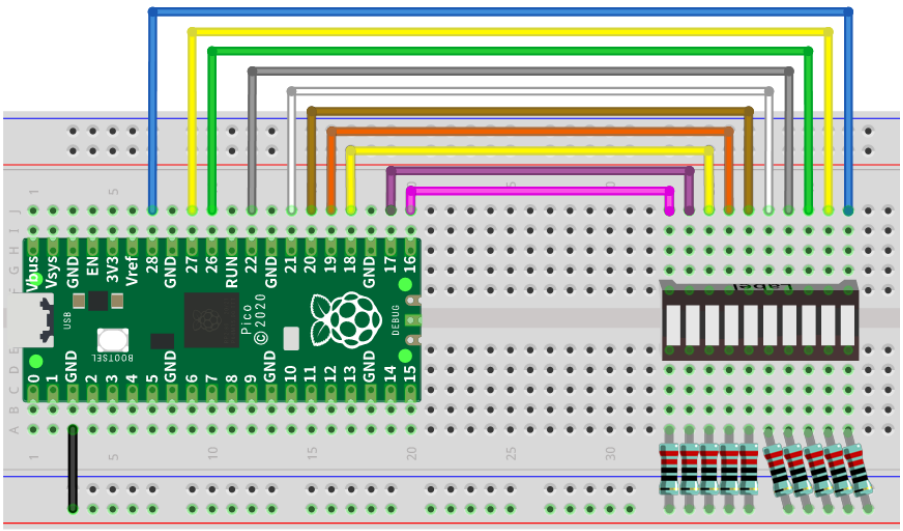


Circuit

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 doesn't work, try to rotate LEDbar for 180°. The label is random.

Any concerns? ✉ support@freenove.com

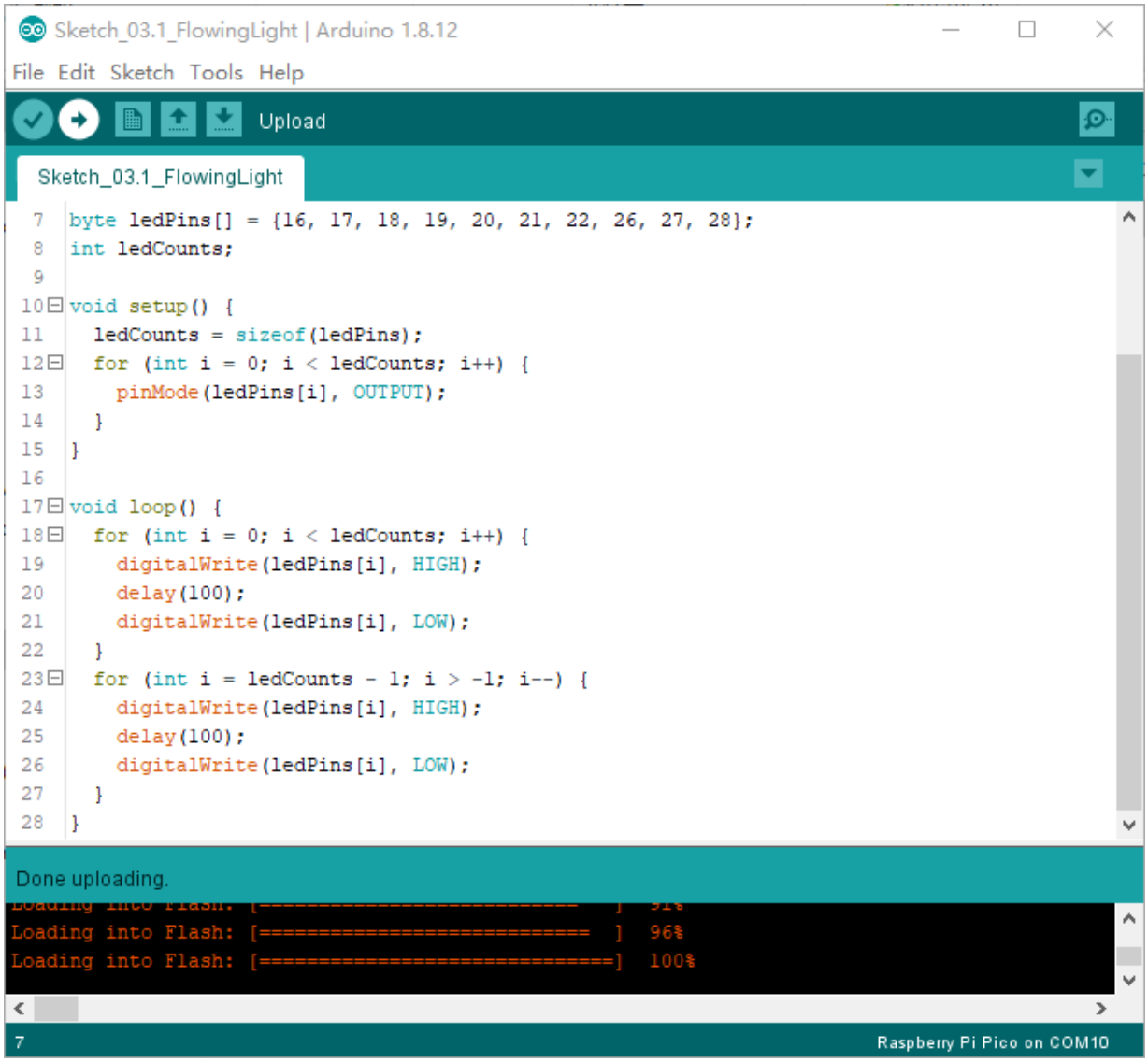
Sketch

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

Upload following sketch:

Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_03.1_FlowingLight.

Sketch_03.1_FlowingLight



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

The following is the program code:

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

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

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

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

```
byte ledPins[] = {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.

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

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

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

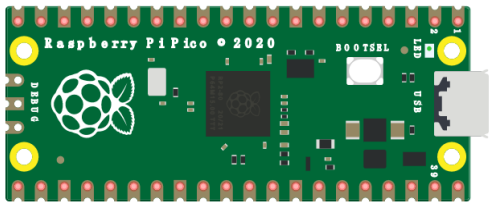
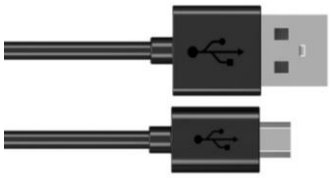
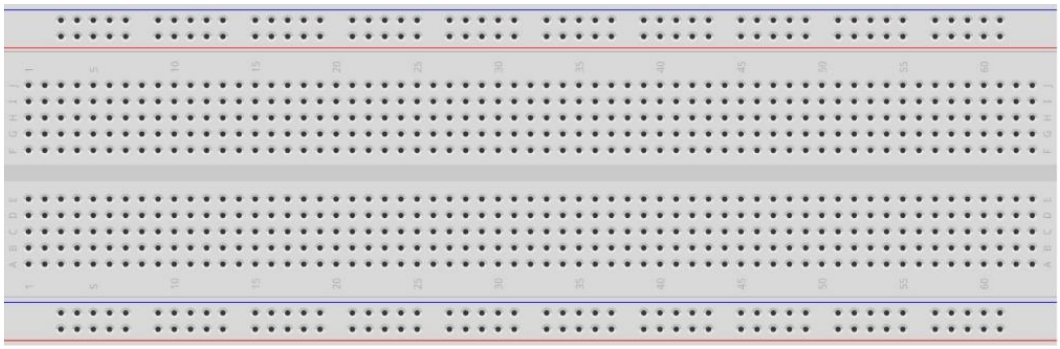
Chapter 4 Analog & PWM

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

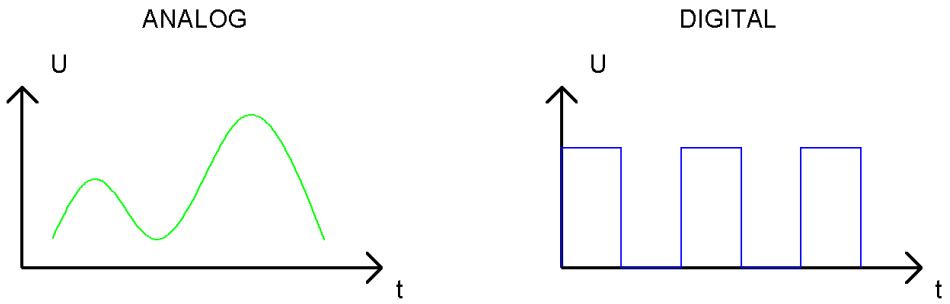
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper
  </td></tr>
</table>

Related Knowledge

Analog & Digital

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



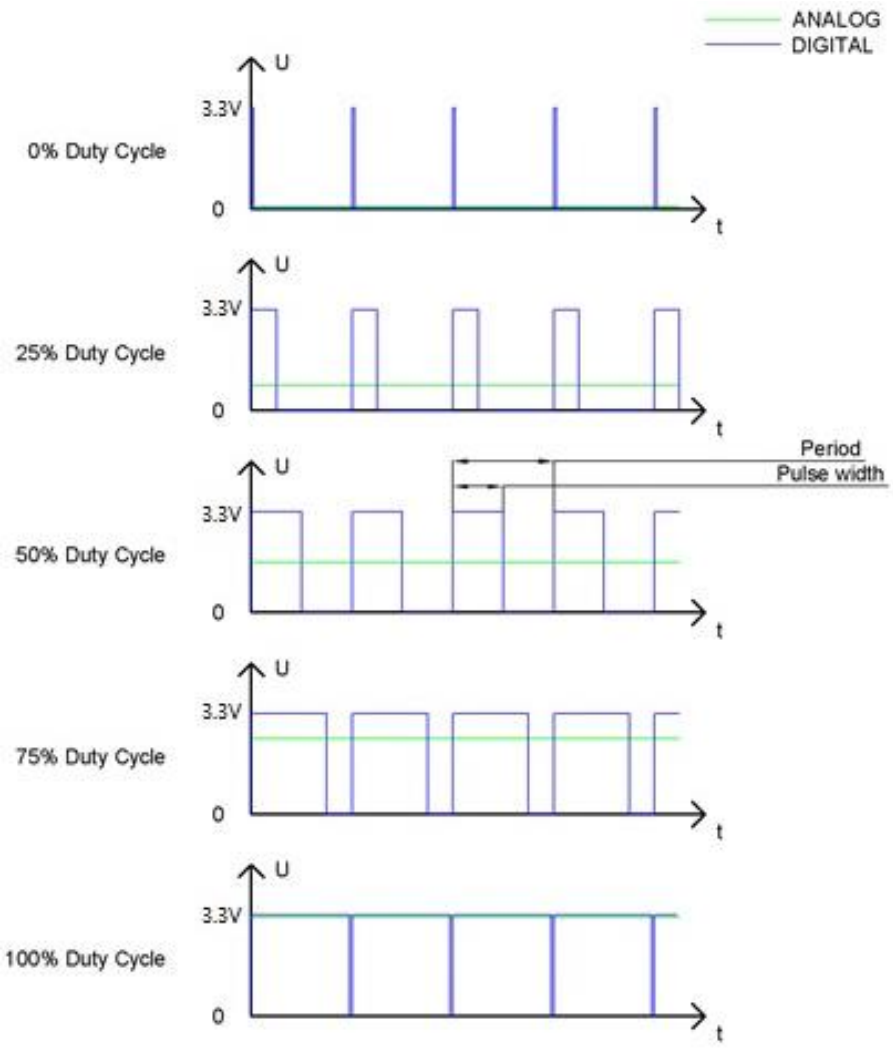
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

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

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

The longer the 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. So, we can control the output power of the LED and other output modules to achieve different effects.

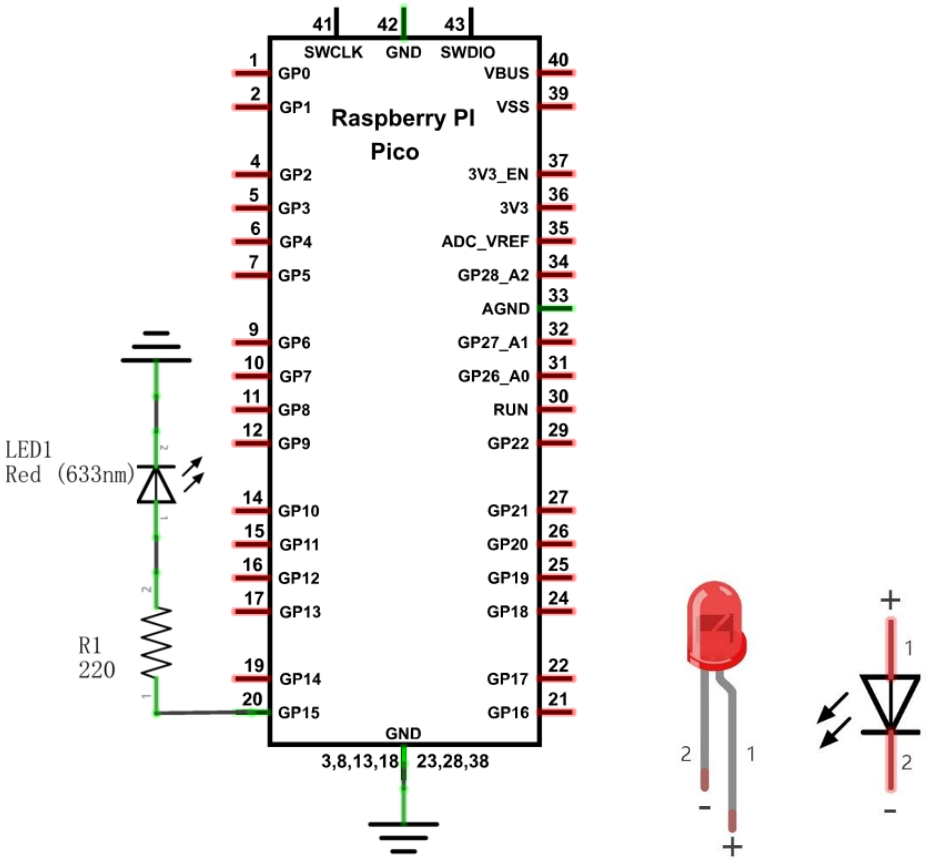
Raspberry Pi Pico and PWM

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.

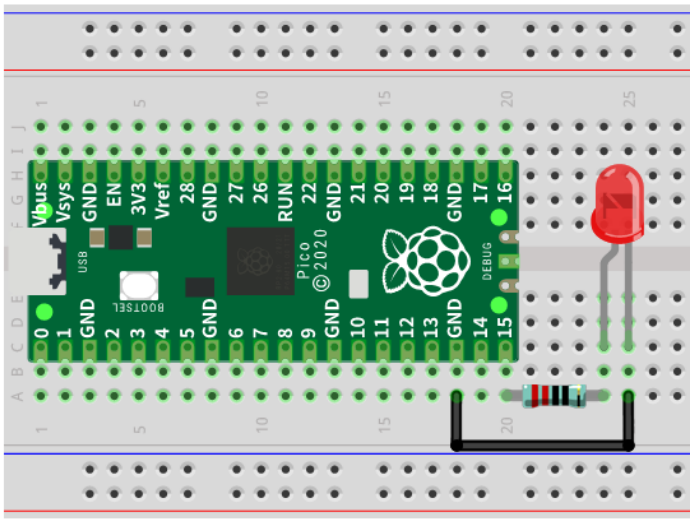
Circuit

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

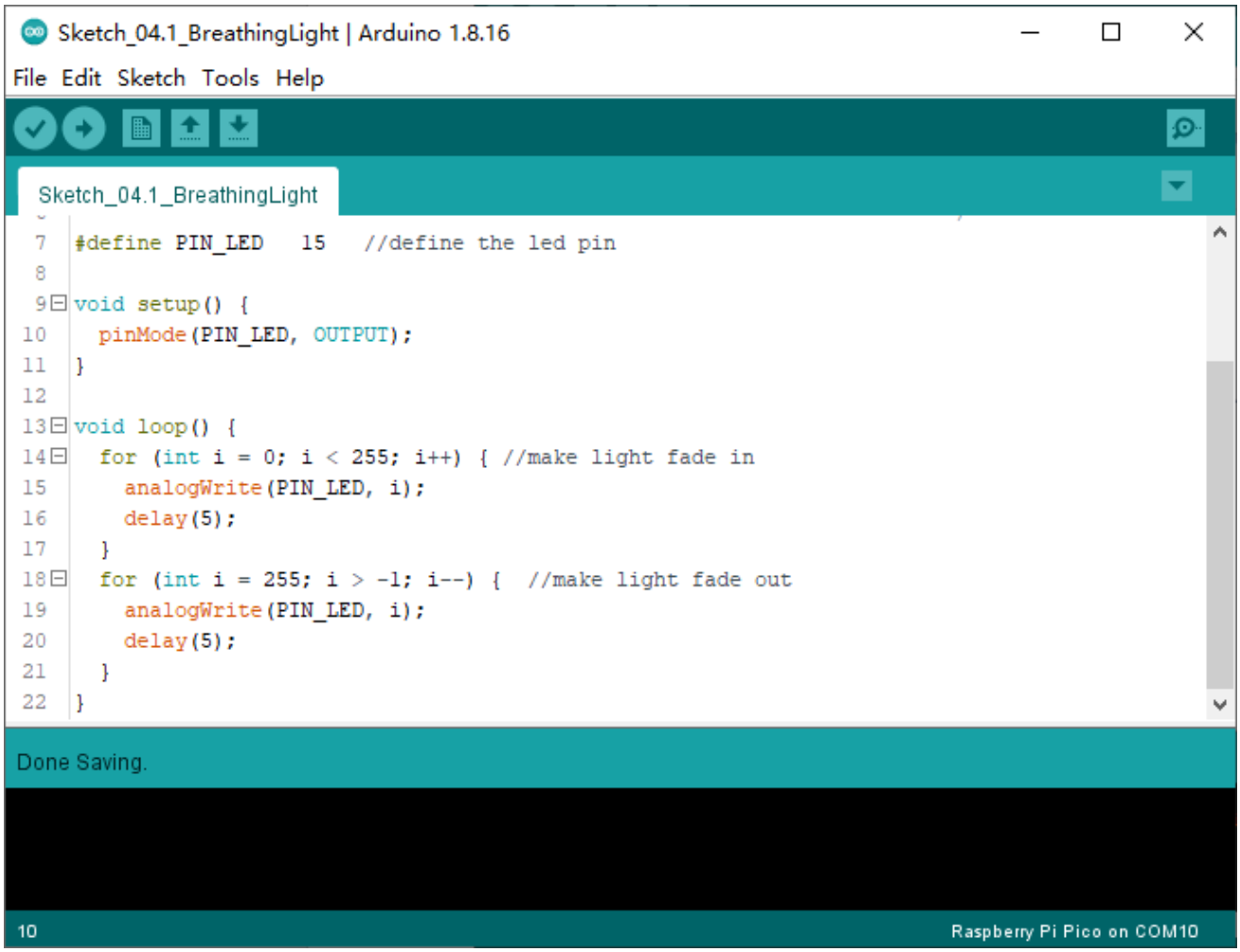


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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



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

The following is the program code:

```
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)

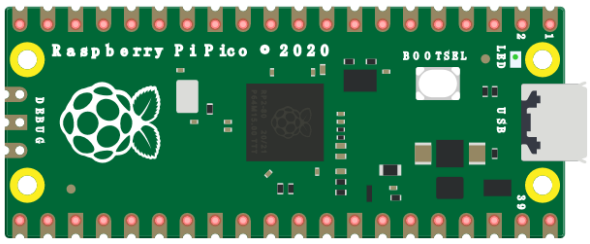
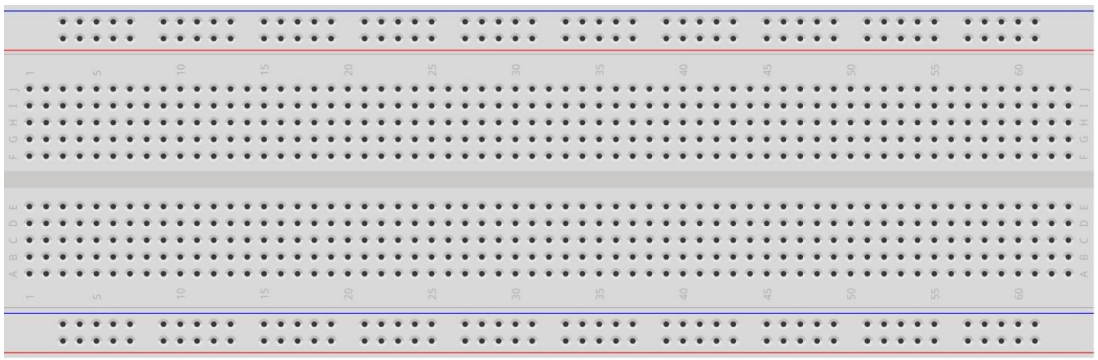
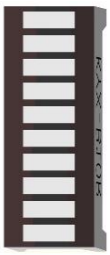
Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255,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

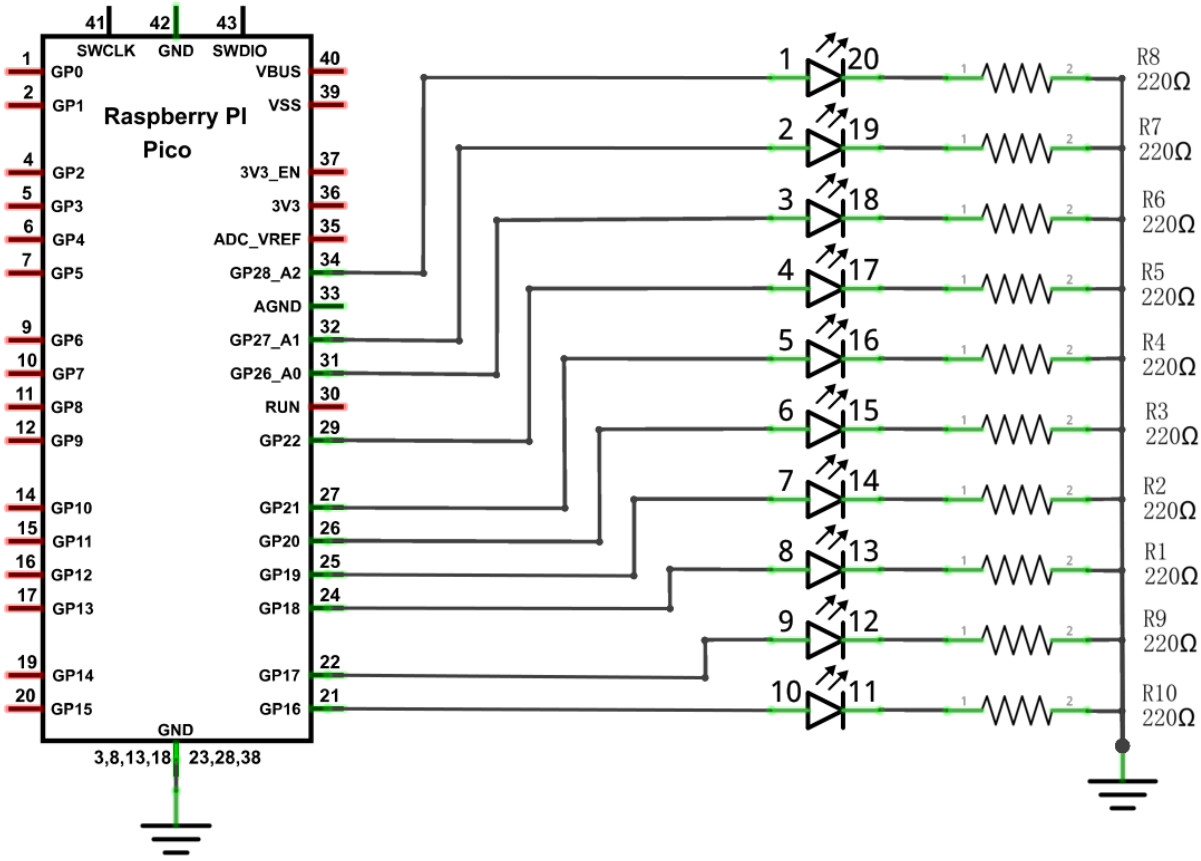
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light.
The component list, circuit, and hardware are exactly consistent with the project Flowing Light.

Component List

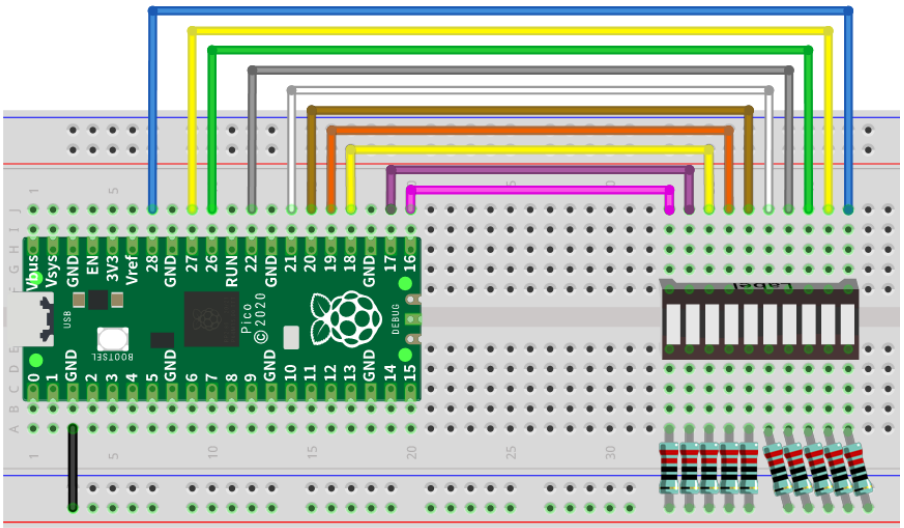
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Jumper 	LED bar graph x1 	Resistor 220Ω x10 

Circuit

Schematic diagram



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



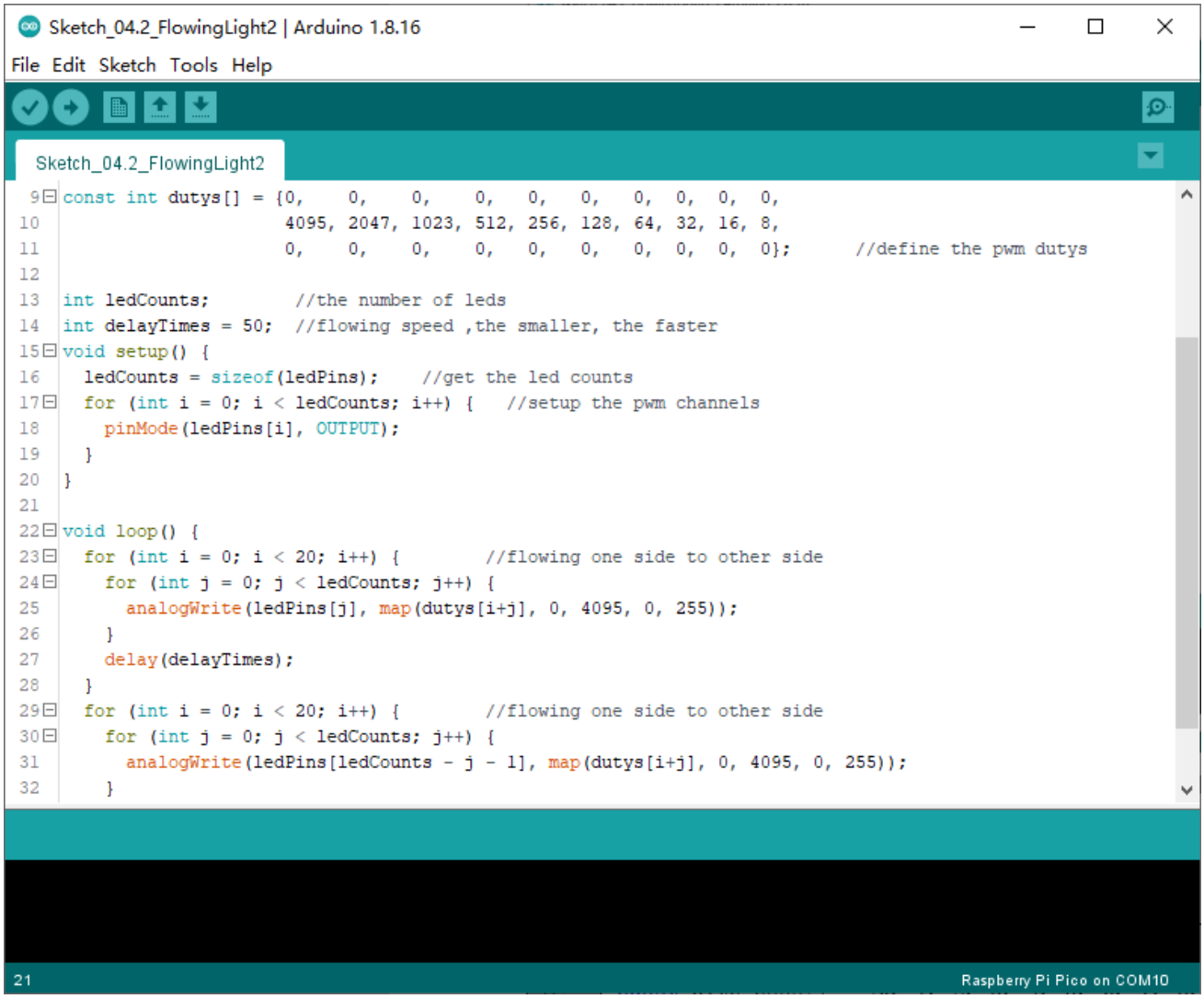
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

Sketch

Meteor flowing light will be implemented with PWM.

Sketch_04.2_FlowingLight2



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
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,   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	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
d i	0	0	0	0	0	0	0	0	0	1023	512	256	128	64	32	16	8	4	2	0	0	0	0	0	0	0	0	0	0
0																													
1																													
2																													
3																													
...																													
18																													
19																													
20																													

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

```
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.

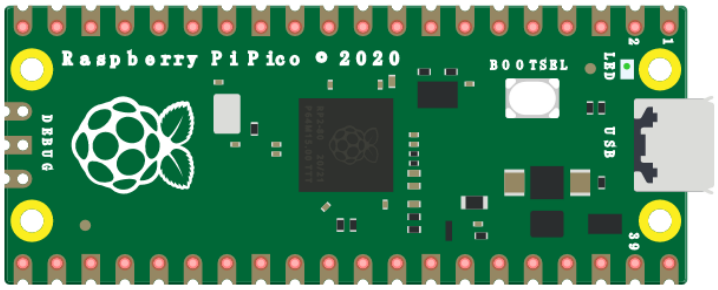
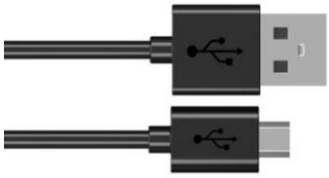
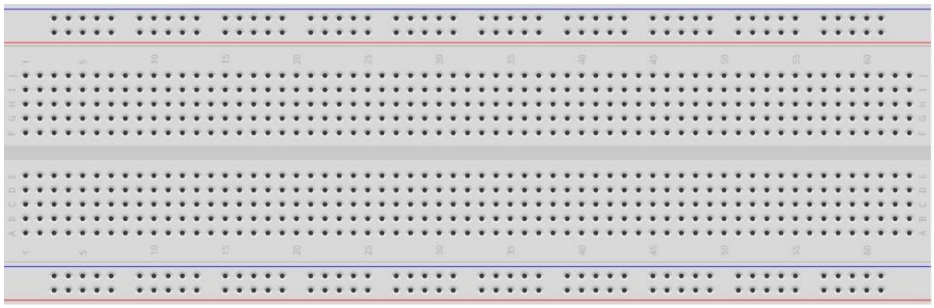
Chapter 5 RGBLED

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

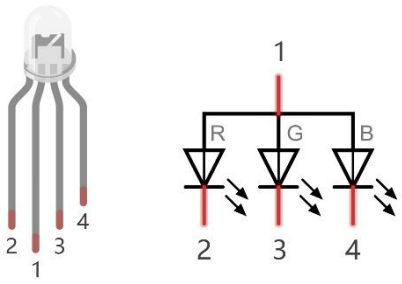
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

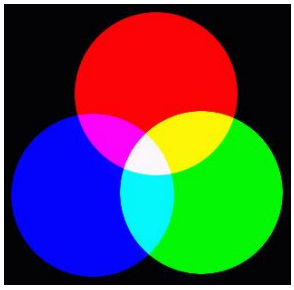
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>RGBLED x1
  </td><td>Resistor 220Ω x3
  </td><td>Jumper
  </td></tr>
</table>

Related Knowledge

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



Red, green, and blue 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, and etc. are working under this principle.

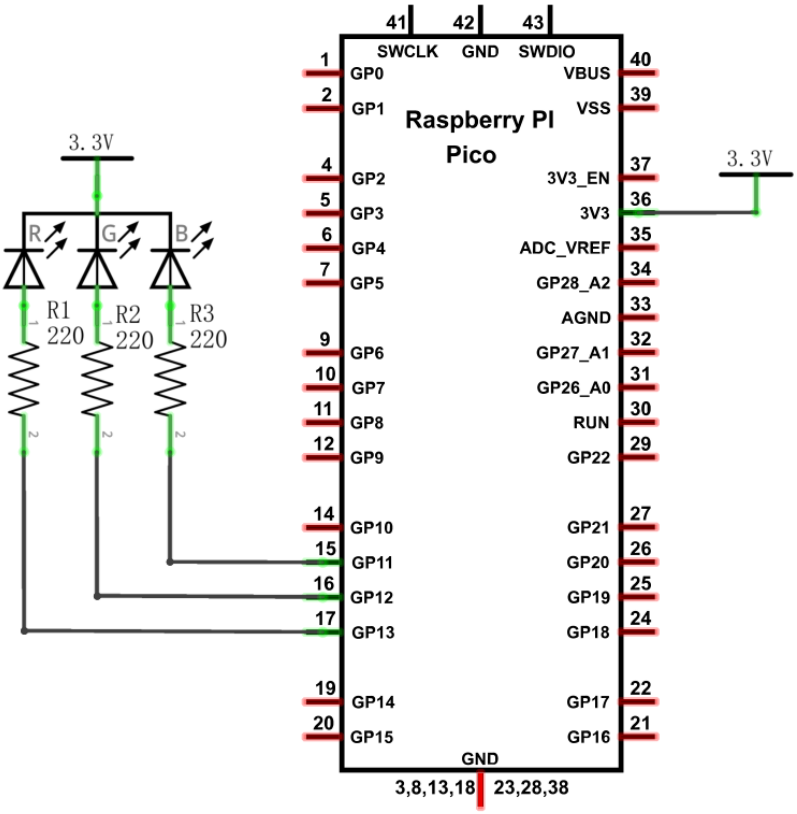


RGB

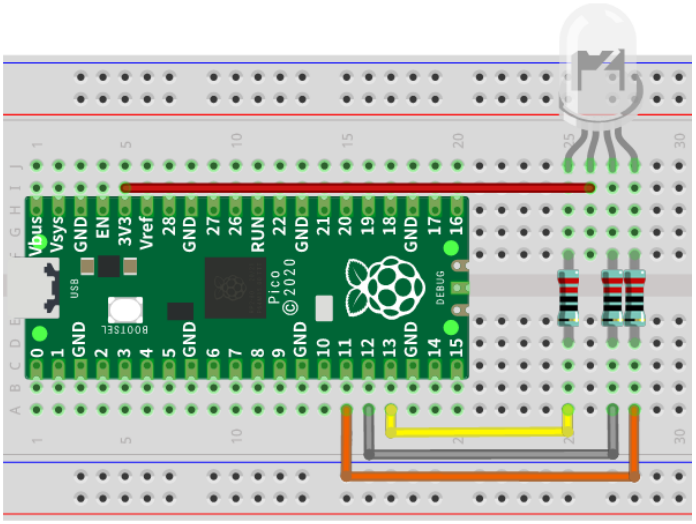
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

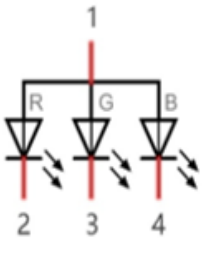
Circuit

Schematic diagram



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



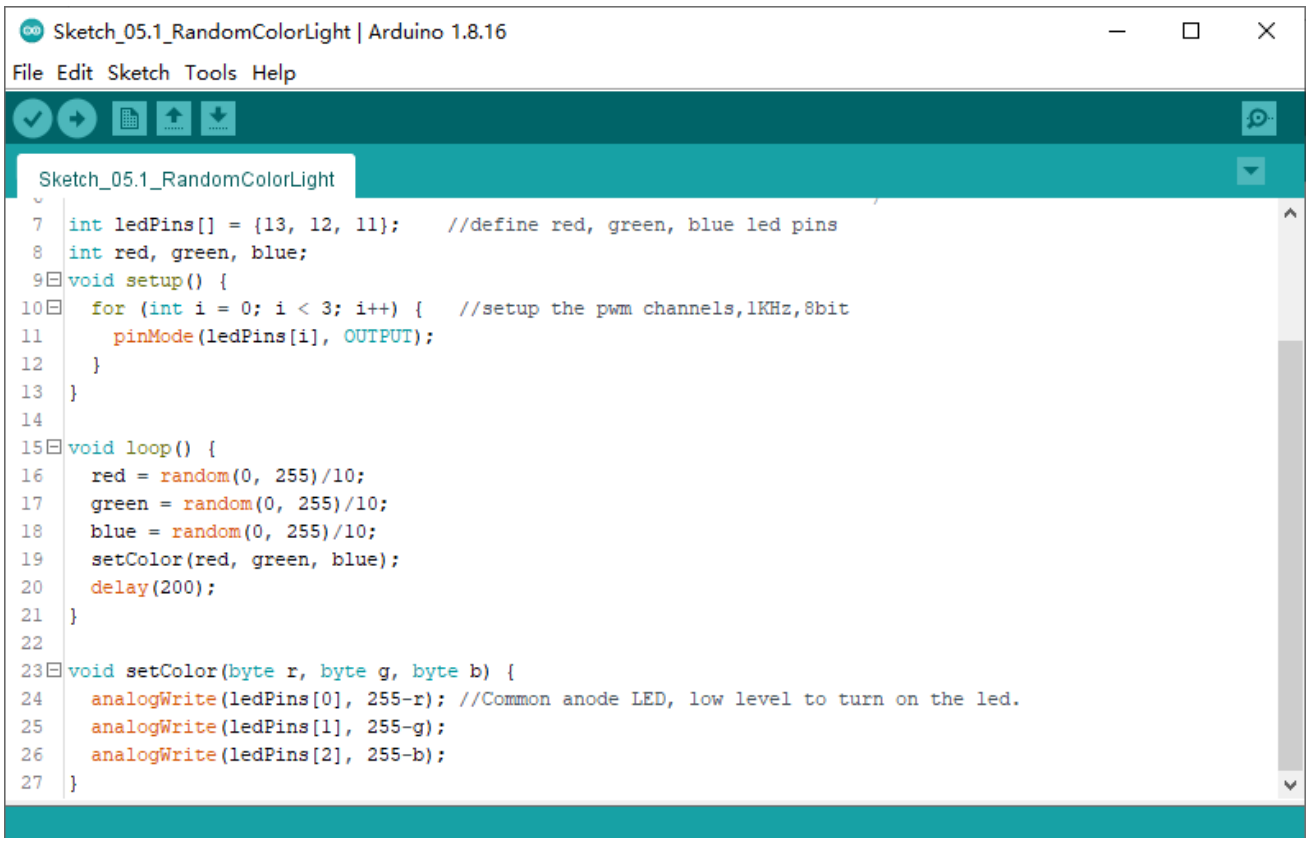


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

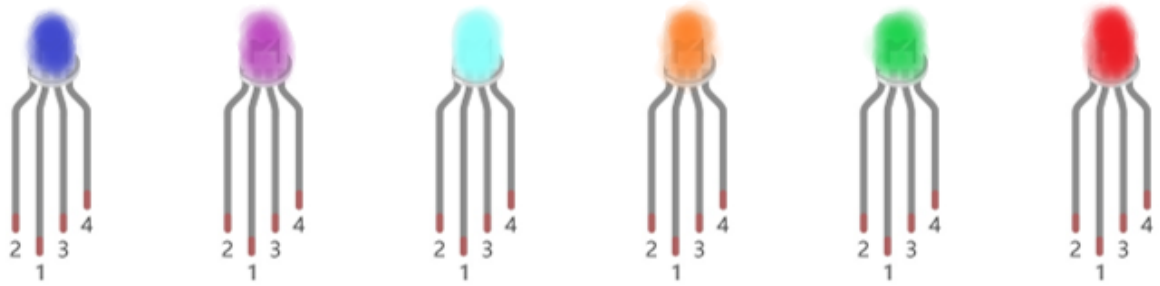
Sketch

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

Sketch_05.1_ColorfulLight



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



The following is the program code:

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

```

```
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 exactly 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:

```
const byte ledPins[] = {13, 12, 11};    //define led pins
void setup() {
  for (int i = 0; i < 3; i++) {   //setup the pwm channels
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 256; i++) {
    setColor(wheel(i));
    delay(100);
  }
}

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

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

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

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

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

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

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

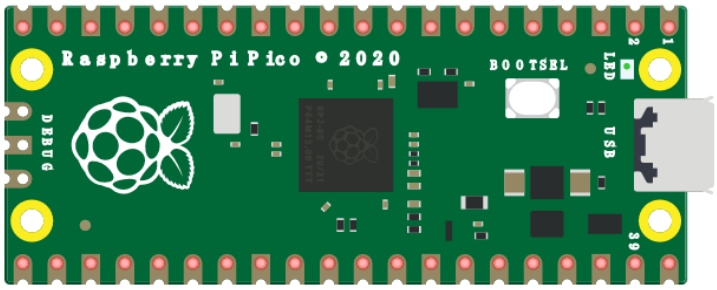
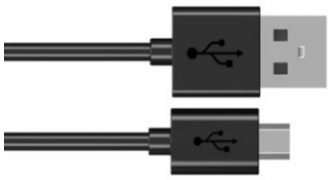
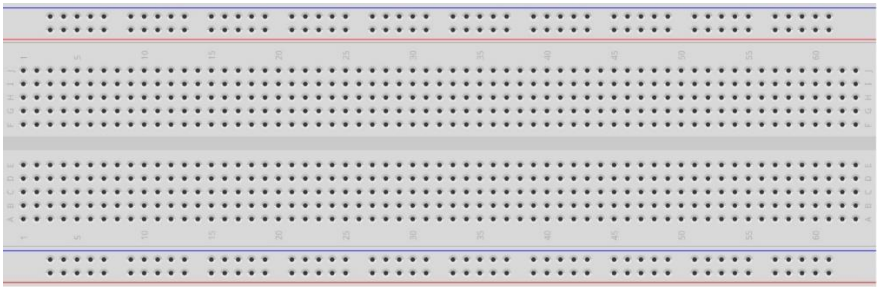
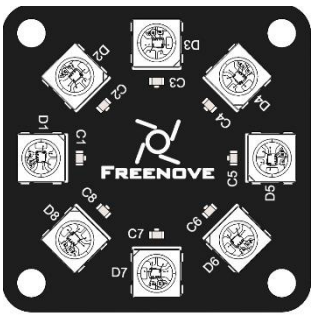
Chapter 6 NeoPixel

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.

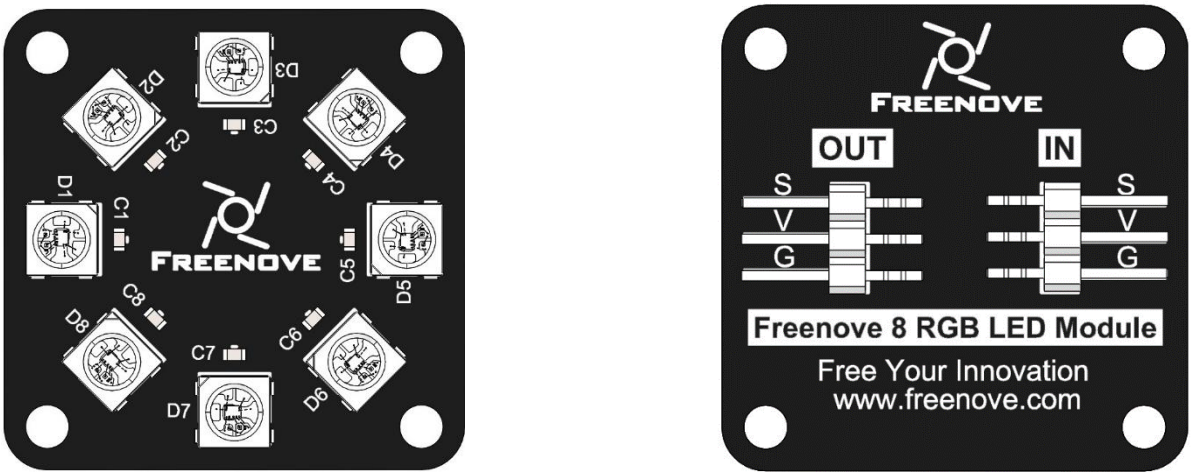
Component List

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

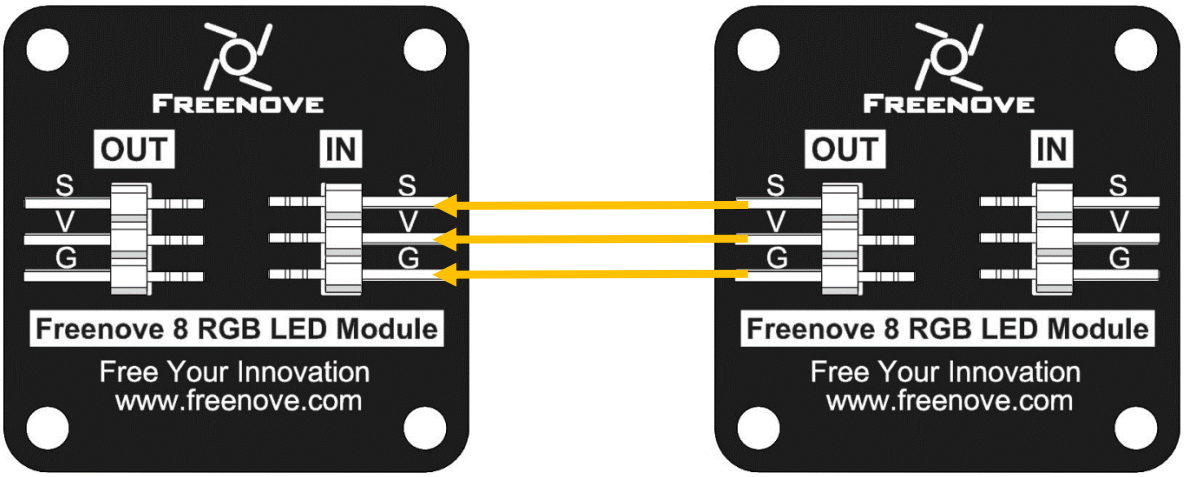
Related Knowledge

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.

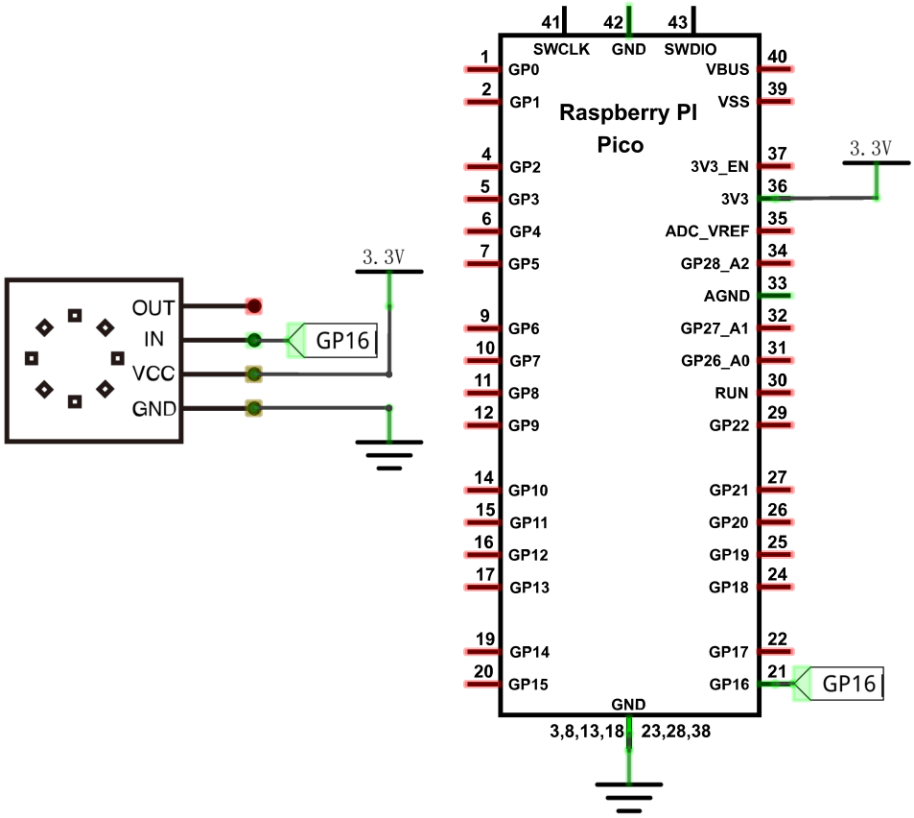


Pin description:

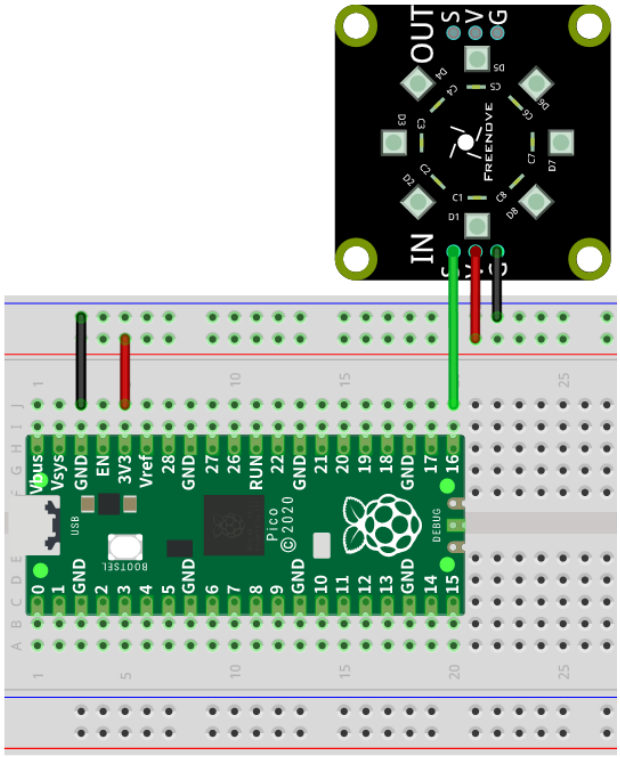
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.3V~5.5V	V	Power supply pin, +3.3V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



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

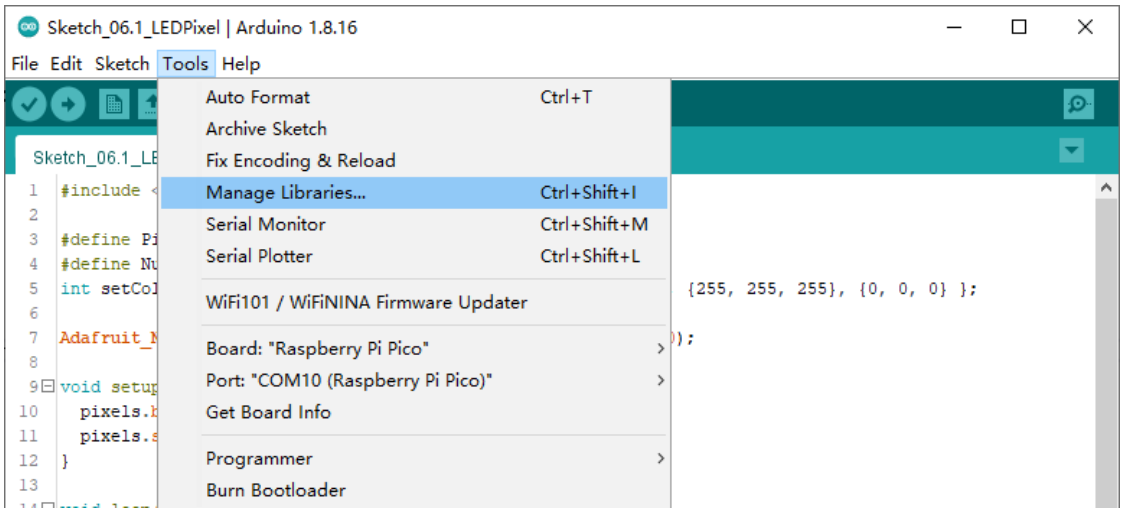


Sketch

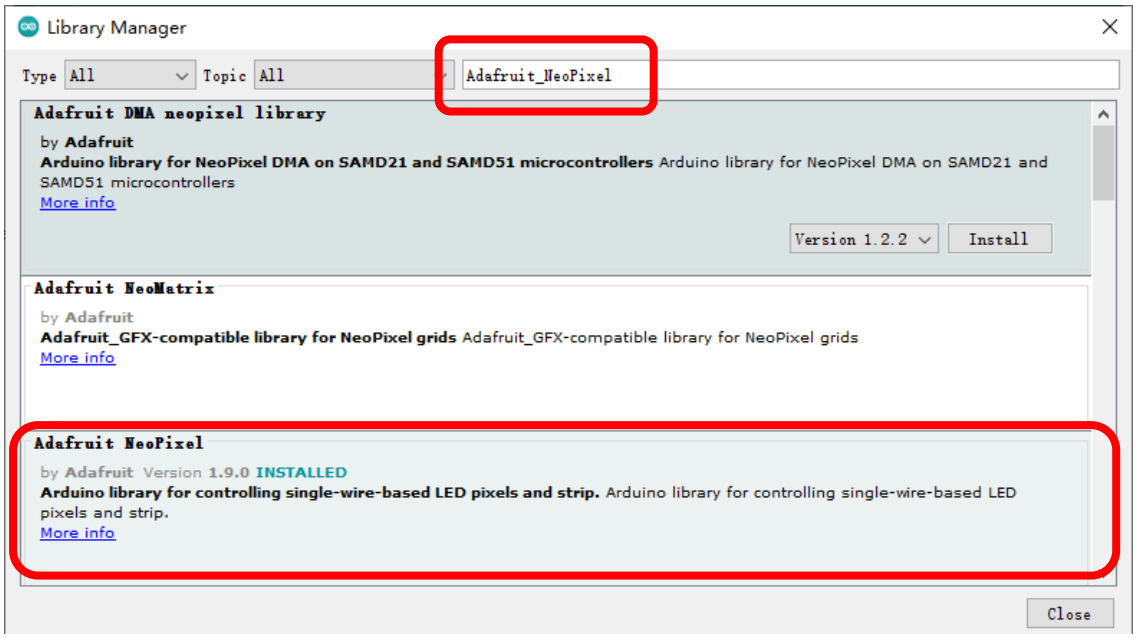
This code uses a library named "**Adafruit_NeoPixel**". If you have not installed it, please do so first.
Library is an important feature of the open source world, and we know that Arduino is an open source platform that everyone can contribute to. Libraries are generally licensed under the LGPL, which means you can use them for free to apply to your creations.

How to install the library

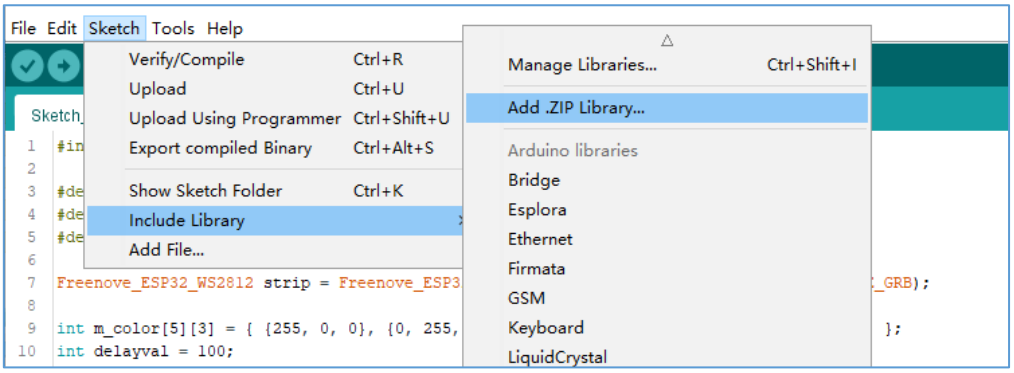
There are two ways to add libraries.
The first way, open the Arduino IDE, click Tools →Manager Libraries.



In the pop-up window, Library Manager, search for the name of the Library, "**Adafruit_NeoPixel**". Then click Install.

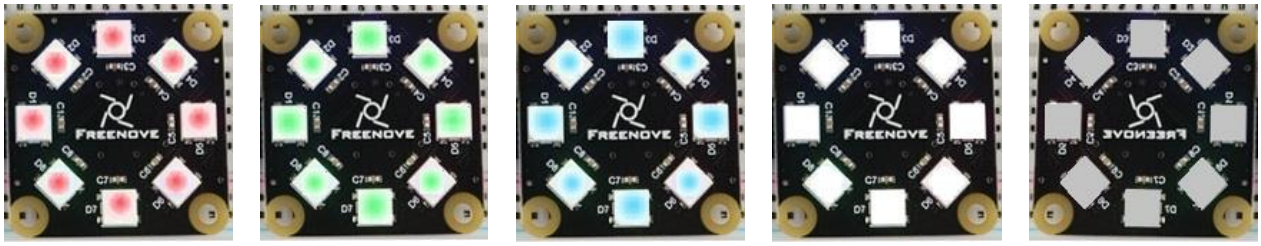


The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named "./Libraries/ **Adafruit_NeoPixel.Zip**" which locates in this directory, and click OPEN.



Sketch_06.1_LEDPixel

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:

```
#include <Adafruit_NeoPixel.h>

#define Pin       16
#define NumPixels 8
int setColor[5][3] = { {255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 255}, {0, 0, 0} };

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

void setup() {
  pixels.begin();
  pixels.setBrightness(20);
}

void loop() {
  for (int j = 0; j < 5; j++) {
    for (int i = 0; i < NumPixels; i++) {
      pixels.setPixelColor(i, setColor[j][0], setColor[j][1], setColor[j][2]);
      pixels.show();
      delay(100);
    }
    delay(500);
  }
}
```

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

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

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

```
#define Pin       16
#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.

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

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

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

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

```
  pixels.begin();
  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.

```
  for (int j = 0; j < 5; j++) {
    for (int i = 0; i < NumPixels; i++) {
      pixels.setPixelColor(i, setColor[j][0], setColor[j][1], setColor[j][2]);
      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 8 LEDs and gradually change.

Sketch_06.2_RainbowLight

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:

```
#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);
  }
}

void Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    red = 255 - WheelPos * 3;
    green = 0;
    blue = WheelPos * 3;
  }
  else if (WheelPos < 170) {
    WheelPos -= 85;
    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-j) is used to set the color of each LED, and the external “for” loop(for-i) is used to change the color, in which the self-increment value in i+=1 can be changed to change the color step distance. Changing the delay parameter changes the speed of the color change. Wheel(((i * 256 / 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    }
```

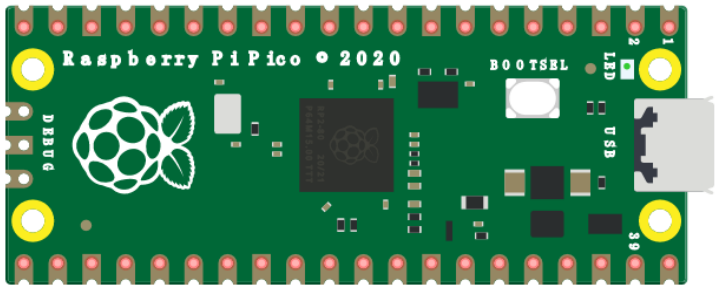
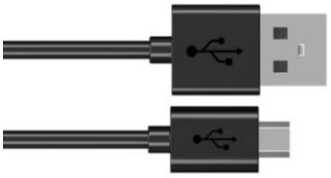
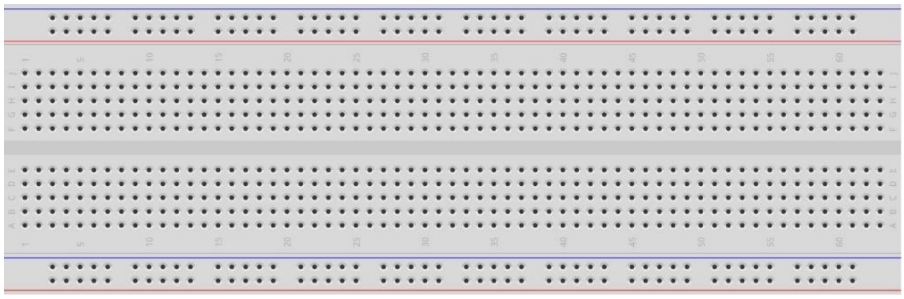
Chapter 7 Buzzer

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

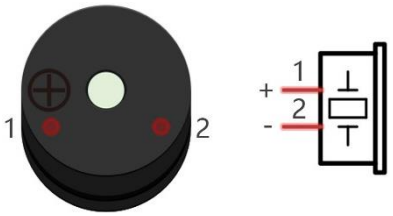
Raspberry Pi Pico x1 			USB cable x1 	
Breadboard x1 				
Jumper 				
NPN transistorx1 (S8050) 	Active buzzer x1 	Push button x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component Knowledge

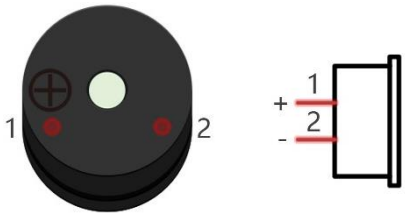
Buzzer

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

Active buzzer



Passive buzzer



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

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

How to identify active and passive buzzer?

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

Active buzzer

Passive buzzer

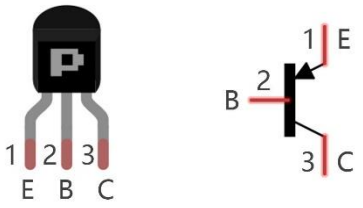
Transistor

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

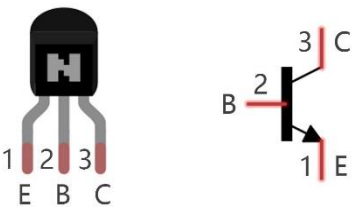
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

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



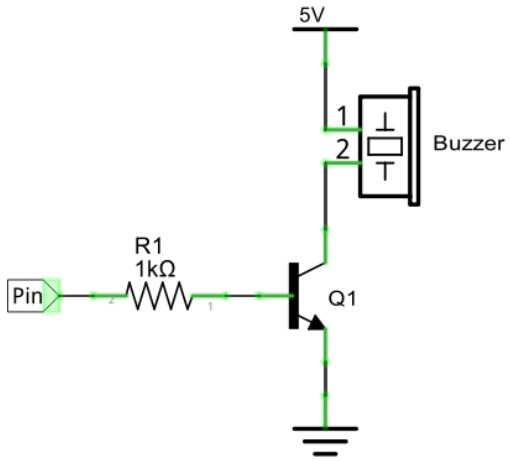
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

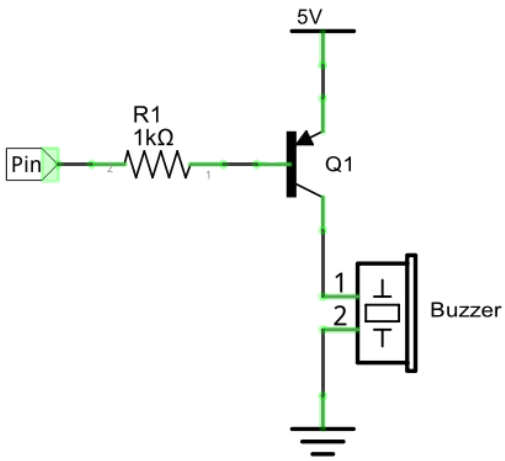
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

NPN transistor to drive buzzer

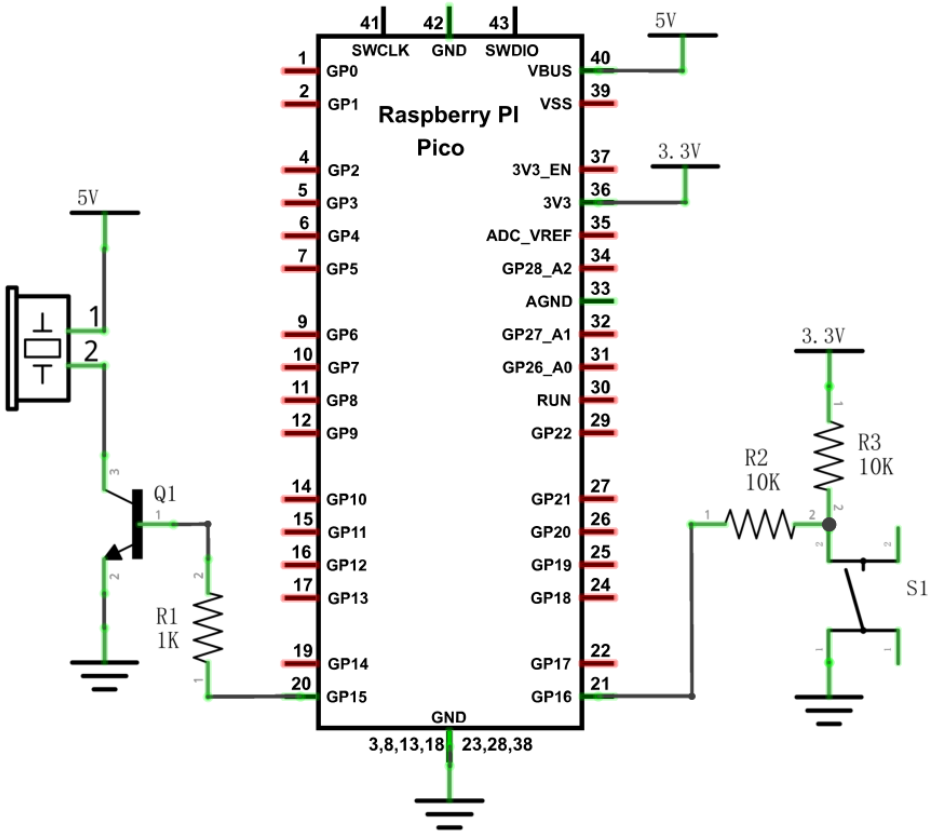


PNP transistor to drive buzzer

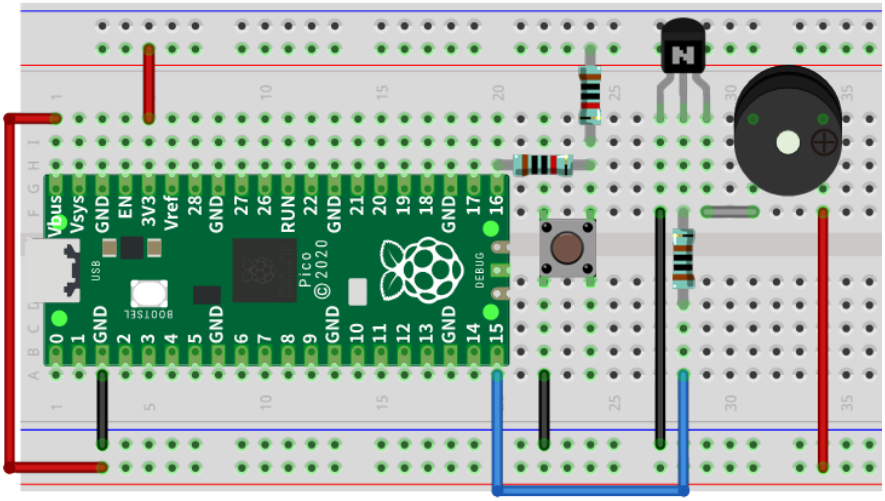


Circuit

Schematic diagram



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



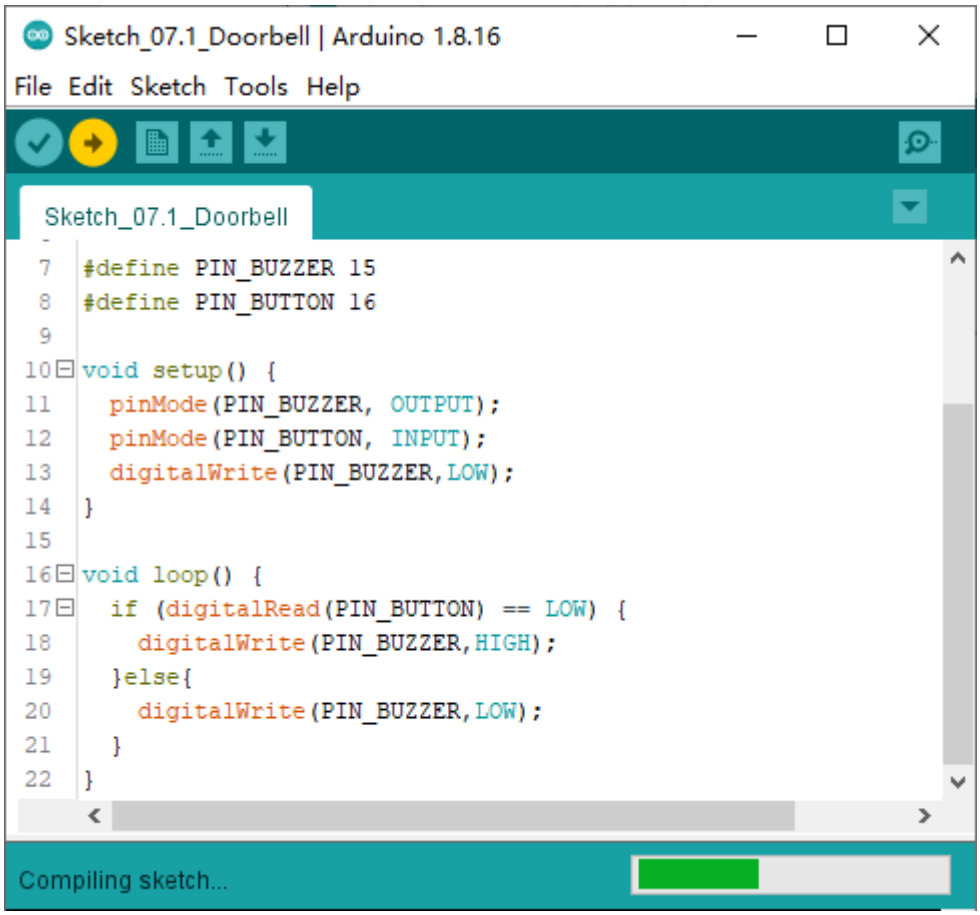
Note:

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.

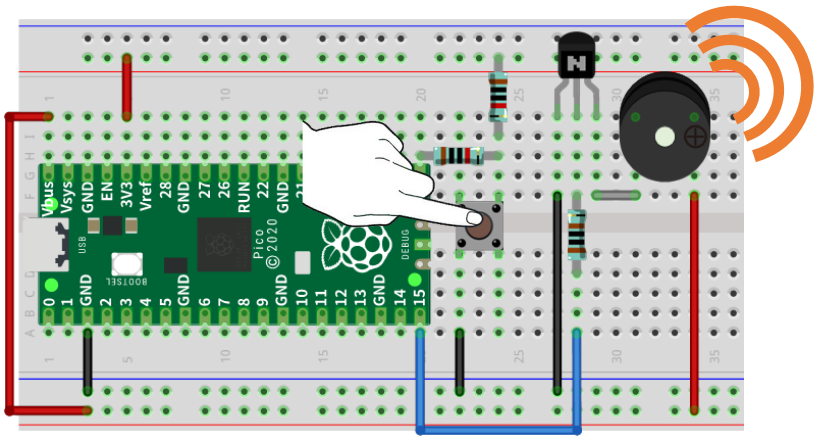
Sketch

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



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:

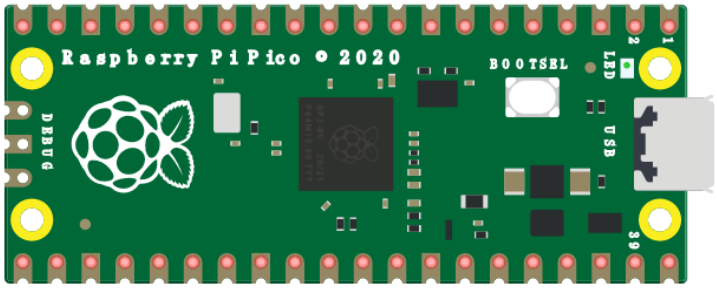
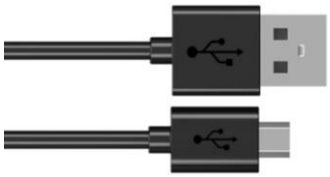
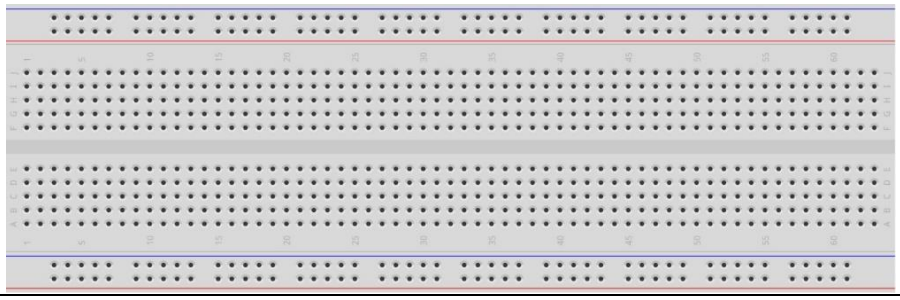
```
#define PIN_BUZZER 15
#define PIN_BUTTON 16

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

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

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

Project 7.2 Alertor

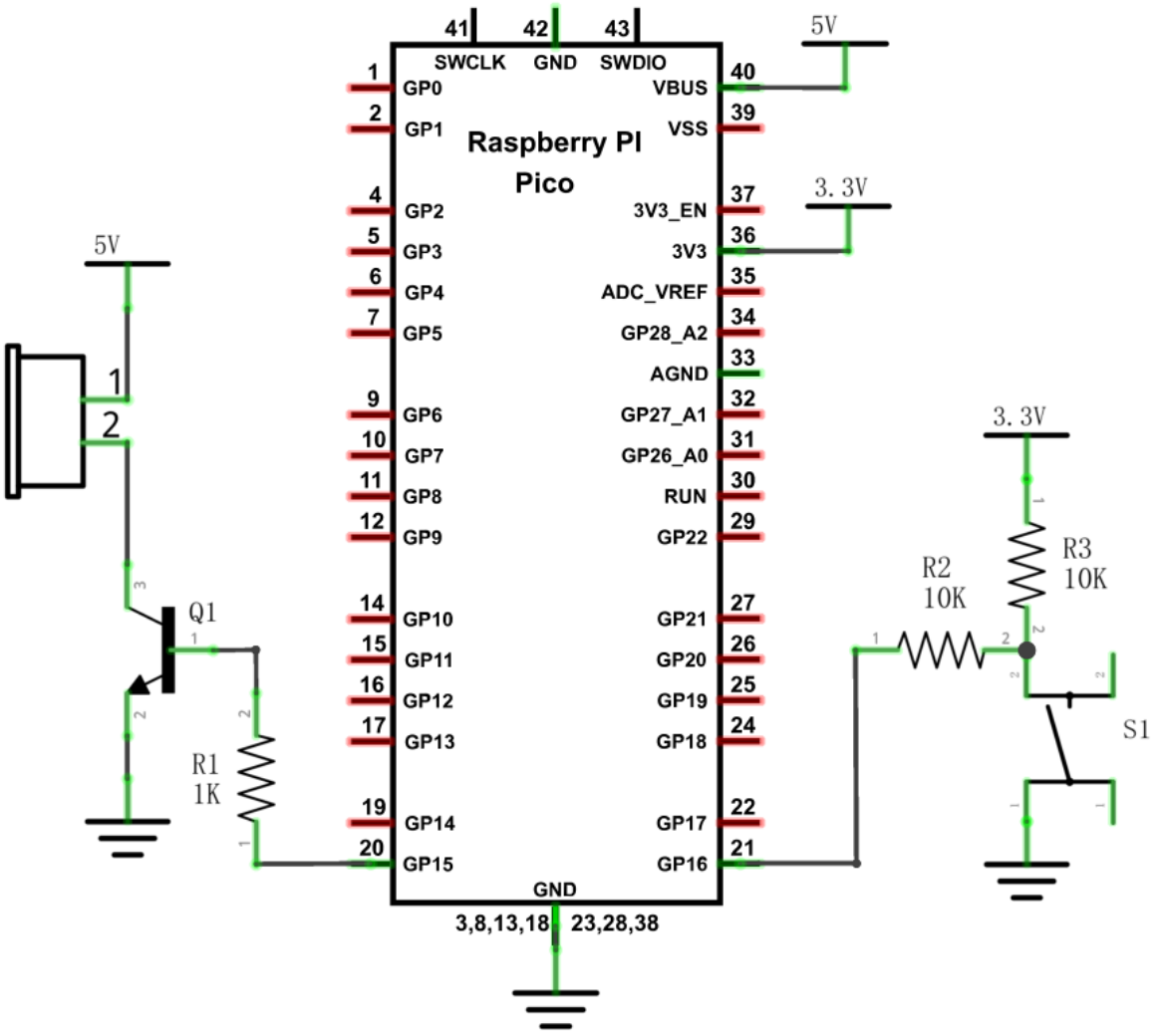
Next, we will use a passive buzzer to make an alarm.
Component list and the circuit part is similar to last section, only the **active buzzer** needs to be **replaced** with a **passive buzzer** for this project.

Component List

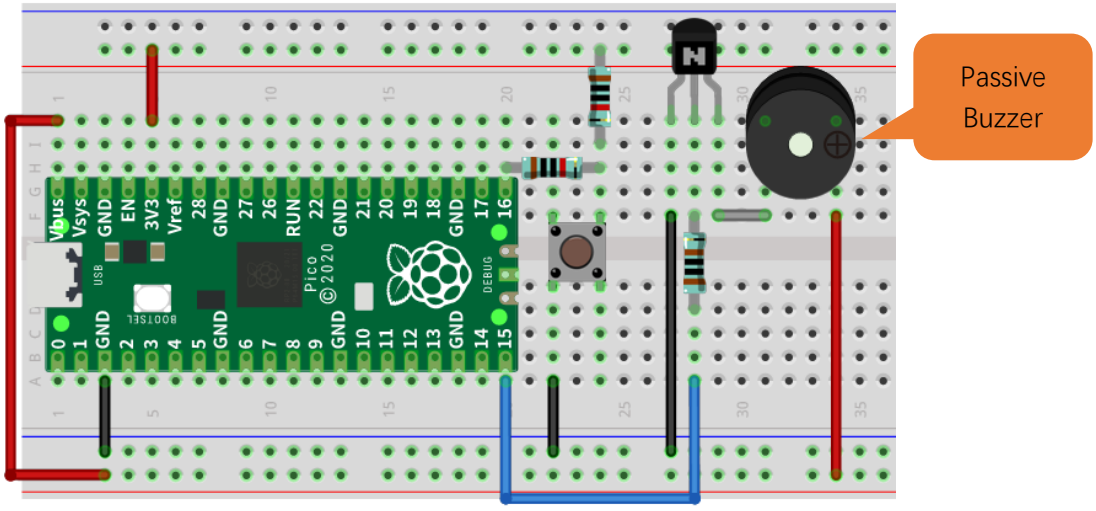
<table>
<tr><td colspan="3">Raspberry Pi Pico x1
  </td><td colspan="3">USB cable x1
  </td></tr>
<tr><td colspan="6">Breadboard x1
  </td></tr>
<tr><td colspan="6">Jumper
  </td></tr>
<tr><td colspan="2">NPN transistorx1
(S8050)
  </td><td>Passive buzzer x1
  </td><td>Push button x1
  </td><td>Resistor
1kΩ x1
  </td><td>Resistor
10kΩ x2
  </td></tr>
</table>

Circuit

Schematic diagram



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

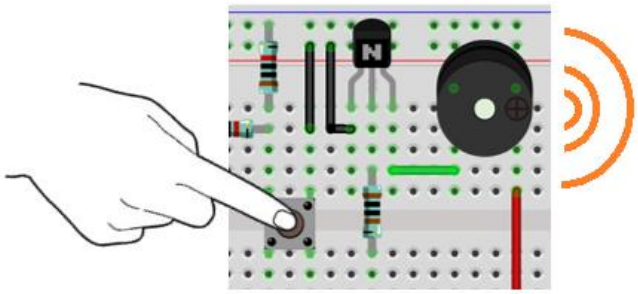


Sketch

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

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



The following is the program code:

```
#define PIN_BUZZER 15
#define PIN_BUTTON 16

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

void loop() {
  if (digitalRead(PIN_BUTTON) == LOW) {
    alert();
  }else {
    freq(PIN_BUZZER, 0, 10);
  }
}

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

void freq(int PIN, int freqs, int times) {
  if (freqs == 0) {
    digitalWrite(PIN, LOW);
  }
  else {
    for (int i = 0; i < times * freqs / 1000; i ++) {
      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 s = 1000000 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    }
```

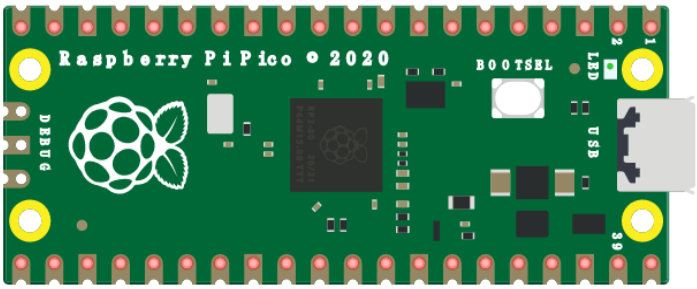
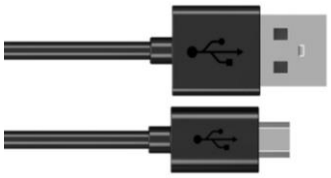
Chapter 8 Serial Communication

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.

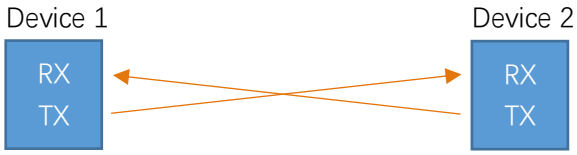
Component List

Raspberry Pi Pico x1 	USB cable x1 

Related Knowledge

Serial communication

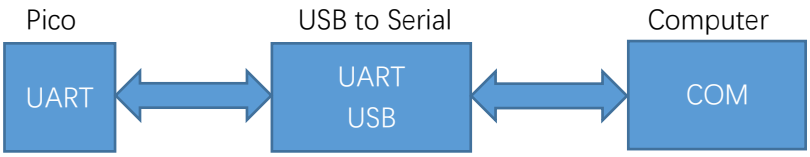
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



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

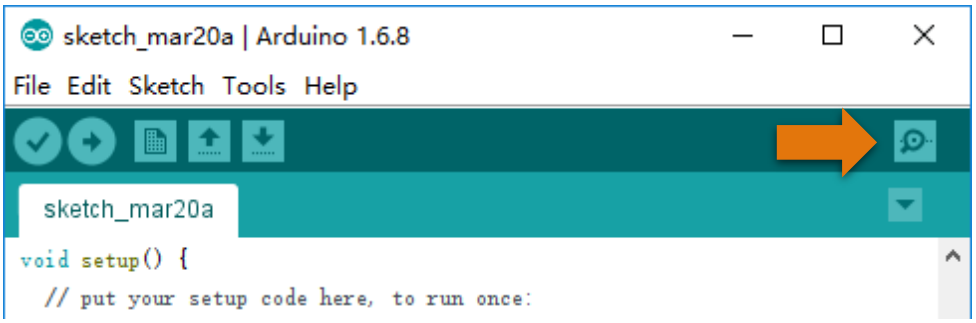
Serial port on Raspberry Pi Pico

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

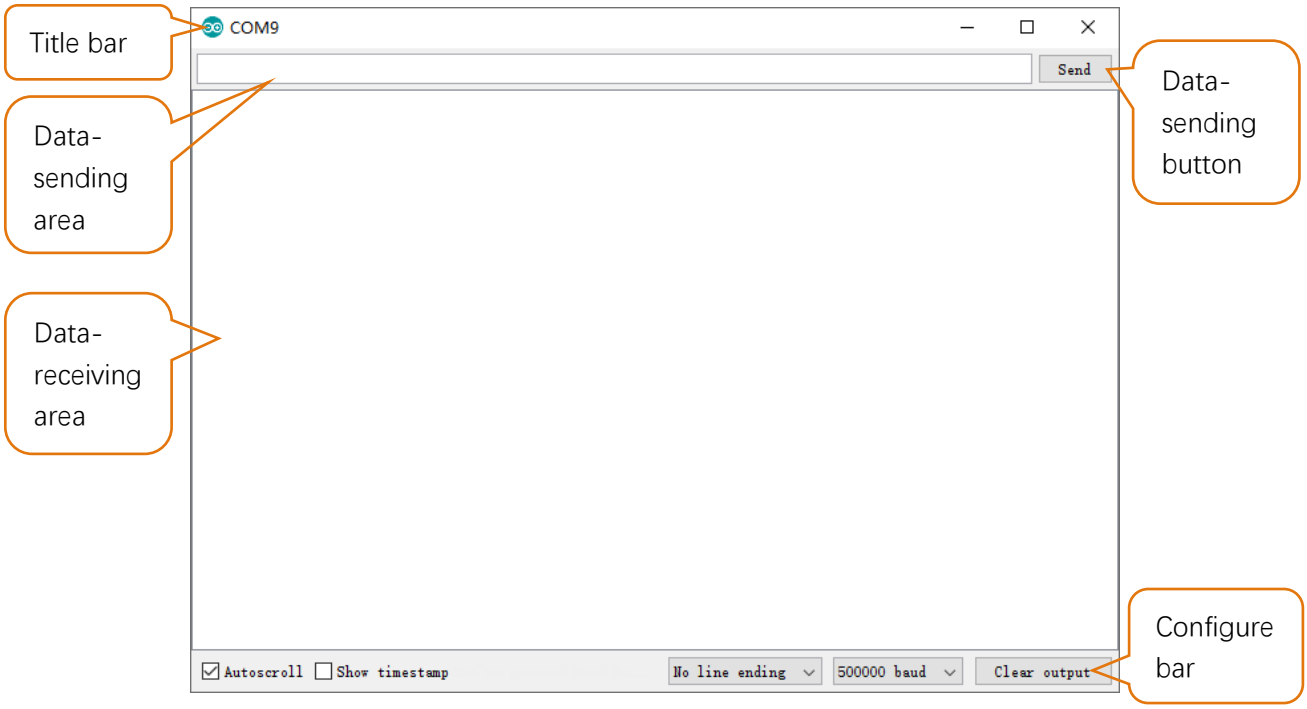


Arduino Software also uploads code to Pico through the serial connection.

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.

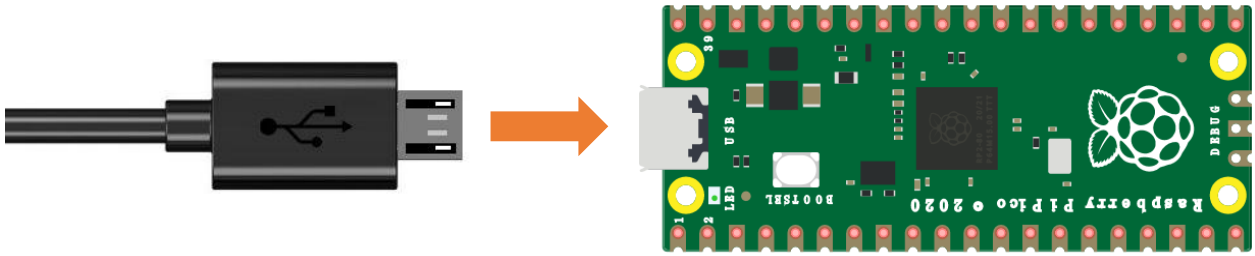


Interface of serial monitor window is as follows. If you can't open it, make sure Pico has been connected to the computer, and choose the right serial port in the menu bar "Tools".



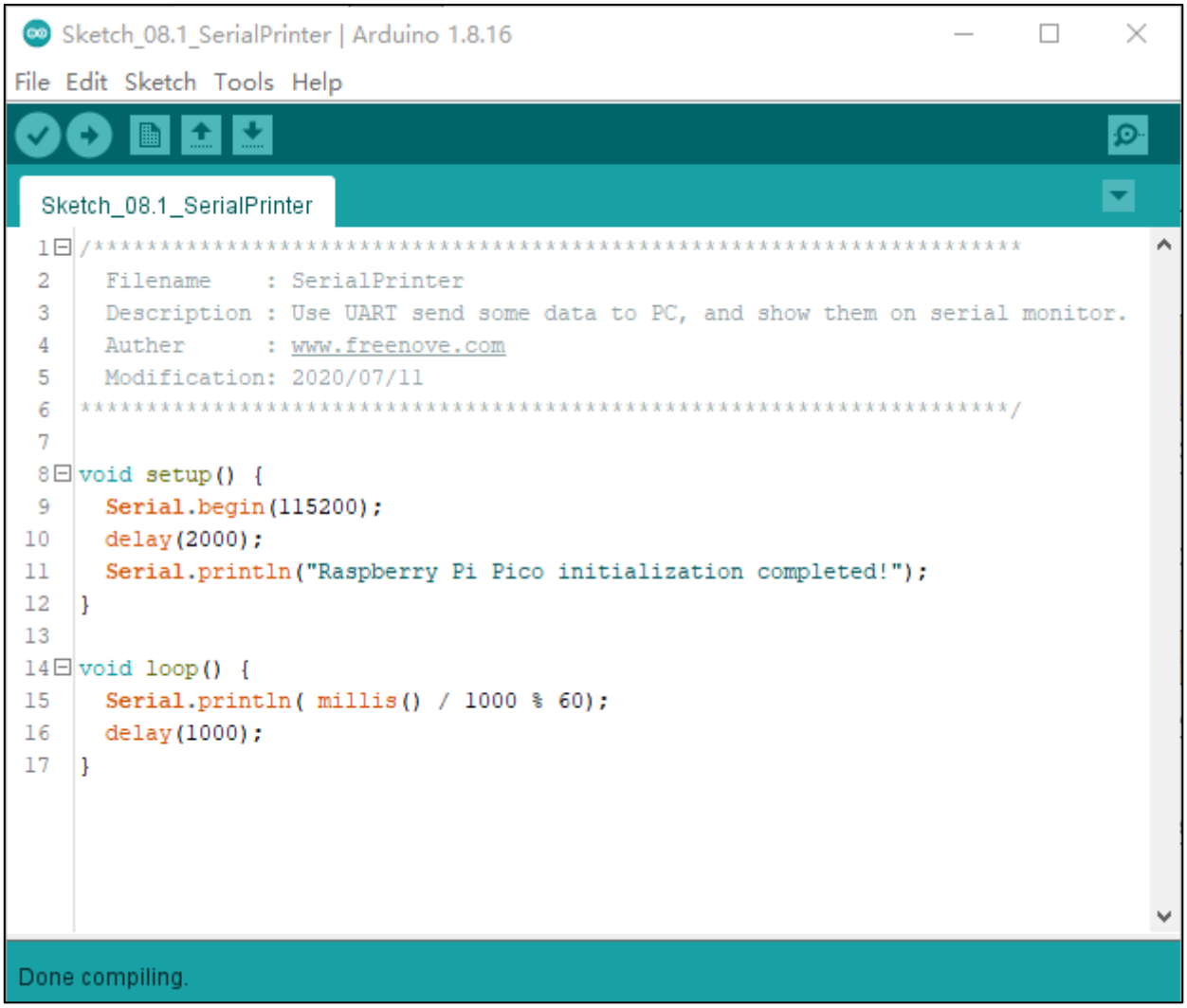
Circuit

Connect Raspberry Pi Pico to the computer with USB cable.

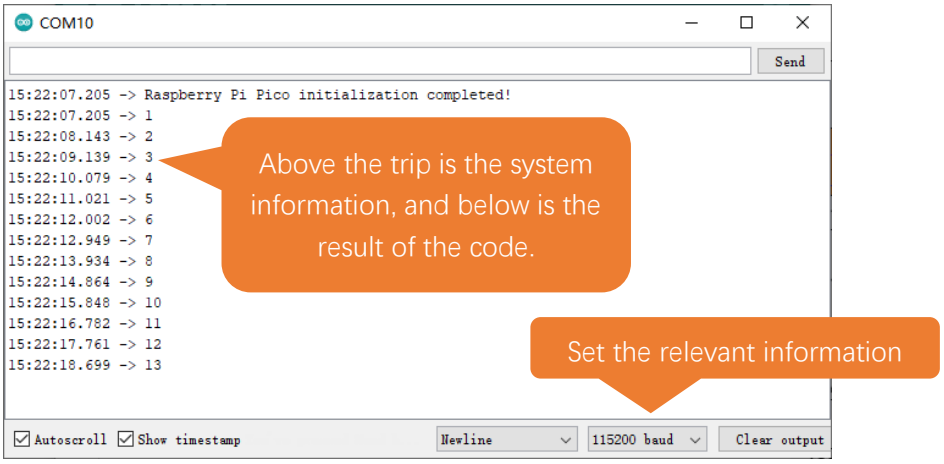


Sketch

Sketch_08.1_SerialPrinter



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



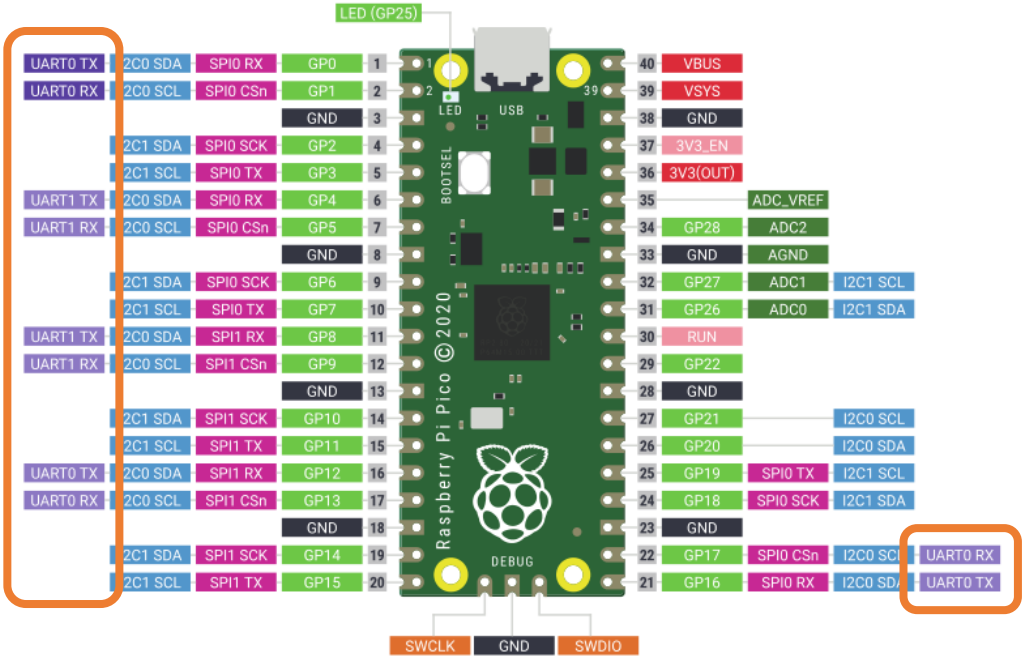
As shown above, when the code runs, the data is printed every one second.

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.

And you can also 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.

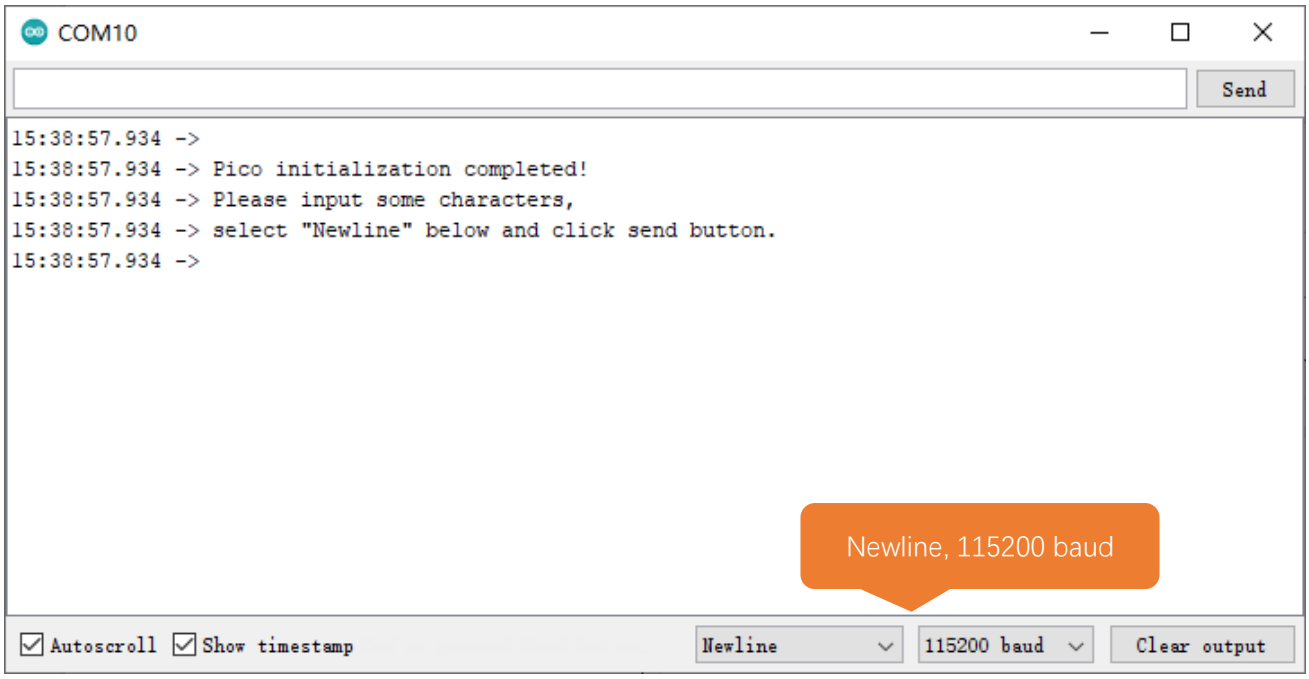
Component and circuit are the same as in the previous project.

Sketch

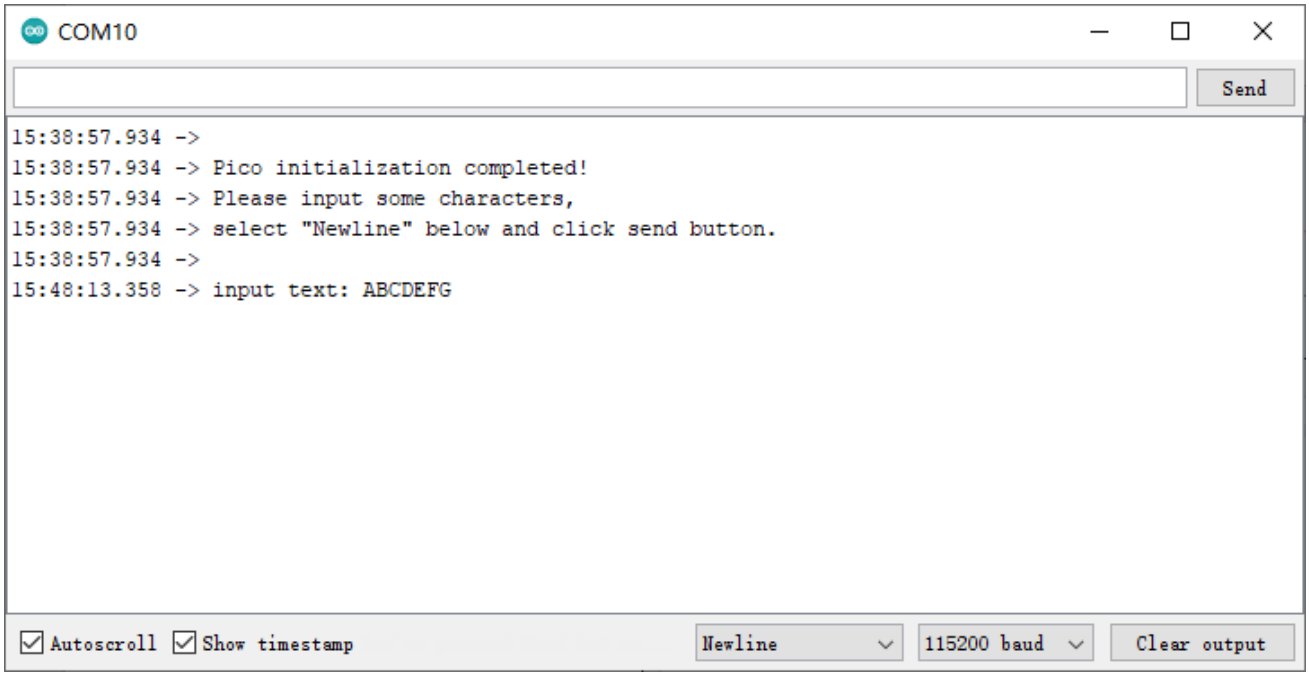
Sketch_08.2_SerialRW


```
/*****************************************************************************
  Filename    : SerialRW
  Description : Use UART read and write data between ESP32 and PC.
  Auther      : www.freenove.com
  Modification: 2020/07/11
*****************************************************************************/
String inputString = "";      //a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

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

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



Then type characters like 'ABCDEFG' into the data sent at the top and click the Send button to print out the data Pico receives.



The following is the program code:

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

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

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

Reference

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

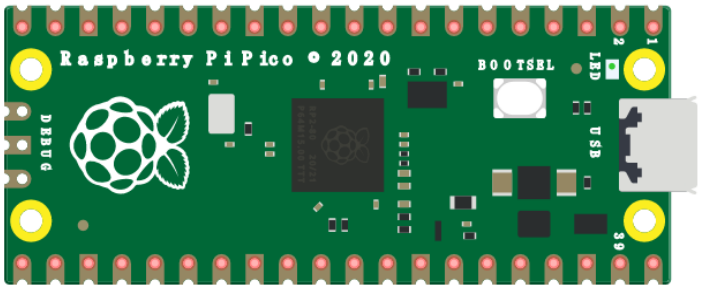
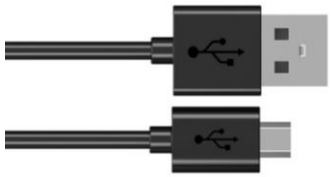
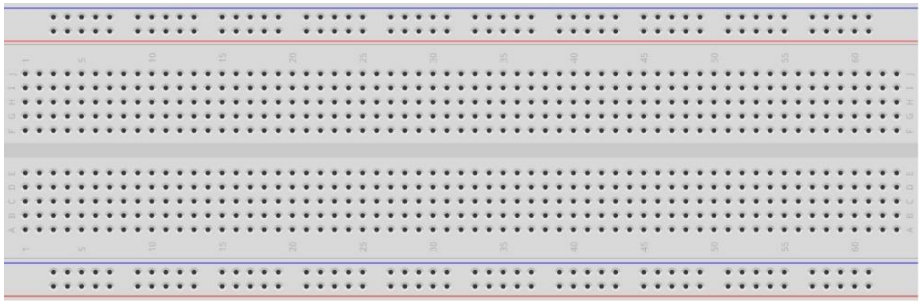
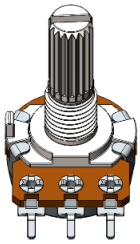
Chapter 9 AD Converter

This chapter we learn to use the ADC function of Rasepberry 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.

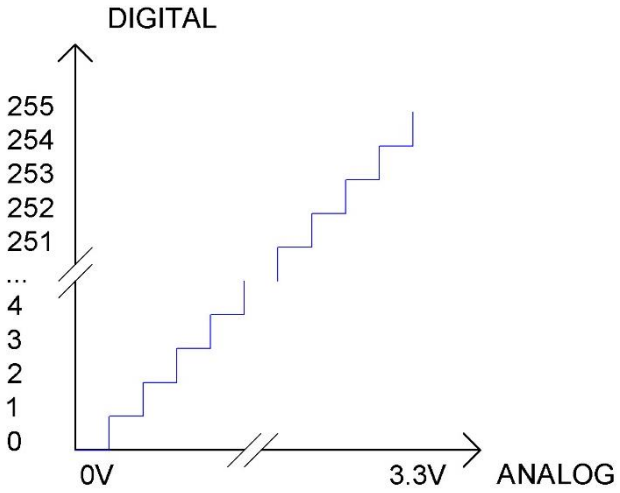
Component List

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, that means the resolution is 2^10=1024, and it represents a range (at 3.3V) will be divided equally to 1024 parts. The rage of analog values corresponds to ADC values. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in rang of 0V---3.3/1023 V corresponds to digital 0;
Subsection 2: the analog in rang 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:

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

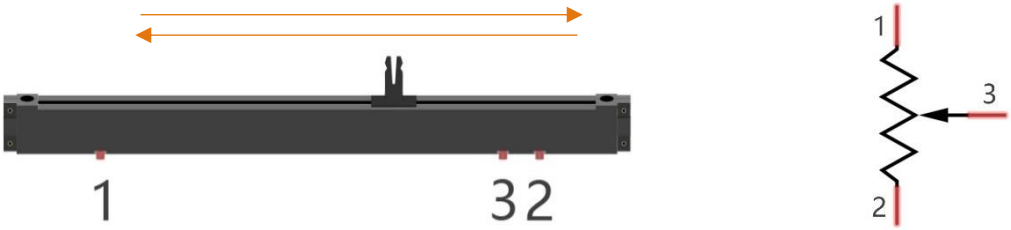
ADC ChannelsRaspberry Pi Pico

Raspberry Pi Pico has 4 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

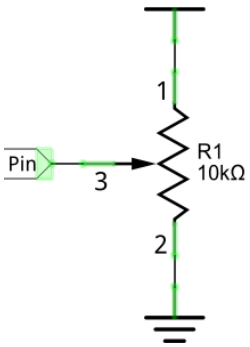
Potentiometer

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.



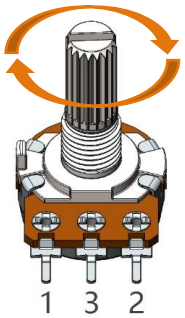
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (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.



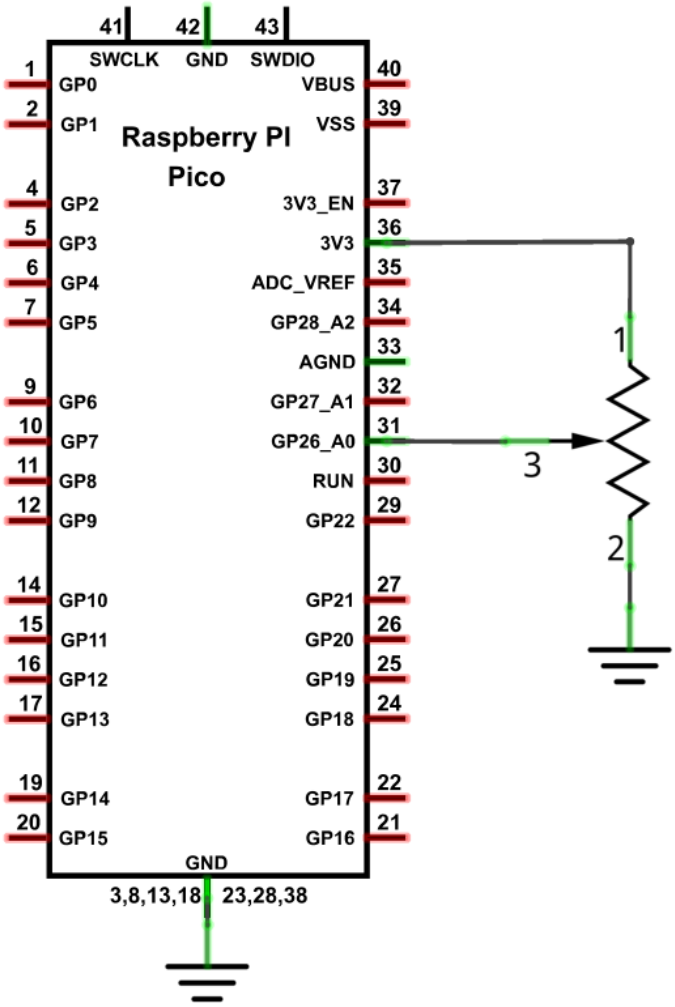
Rotary potentiometer

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

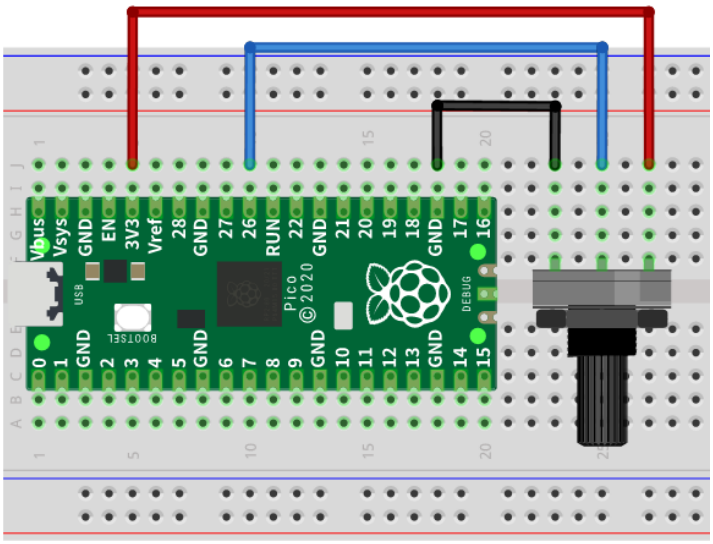


Circuit

Schematic diagram



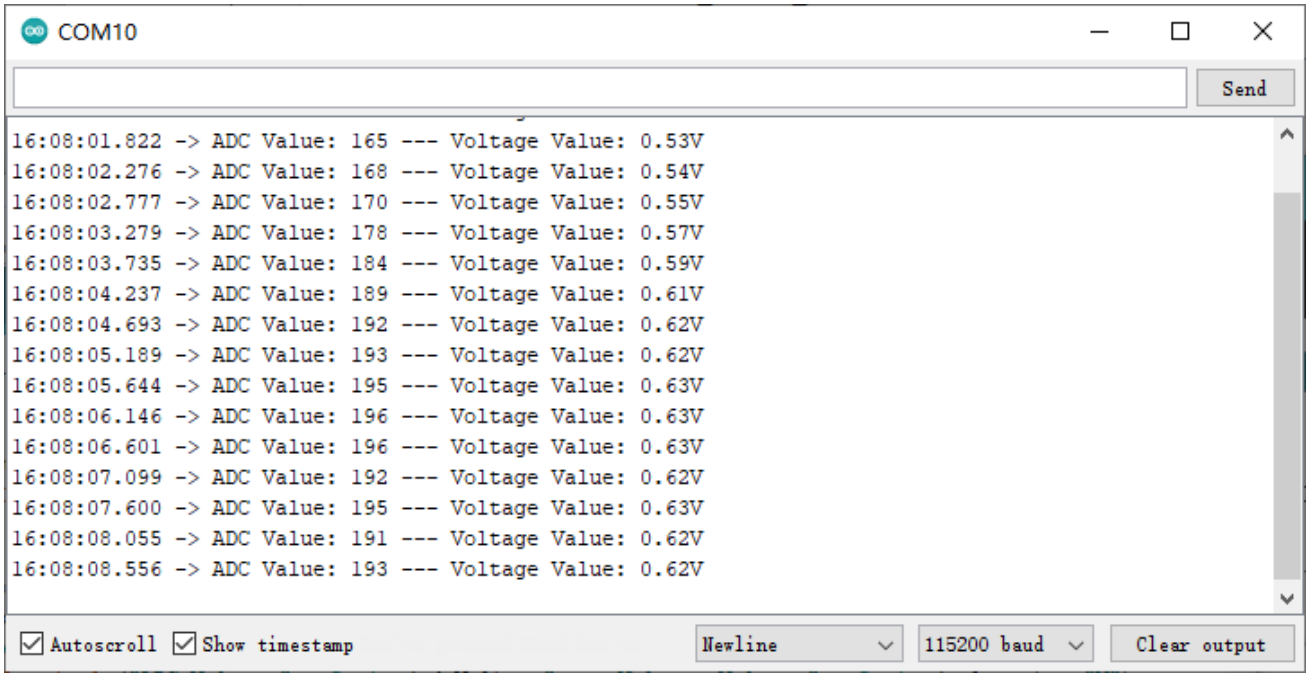
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_09.1_ADC

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



The following is the code:

```
#define PIN_ANALOG_IN  26

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

void loop() {
  int adcVal = analogRead(PIN_ANALOG_IN);
  double voltage = adcVal / 1023.0 * 3.3;
  Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: " + String(voltage) +
"V");
  delay(500);
}
```

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.

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

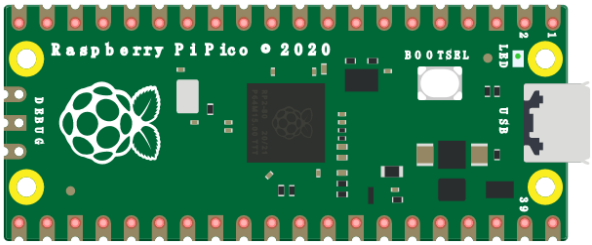
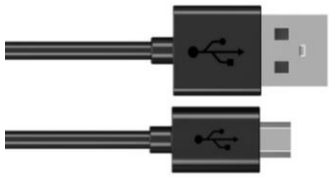
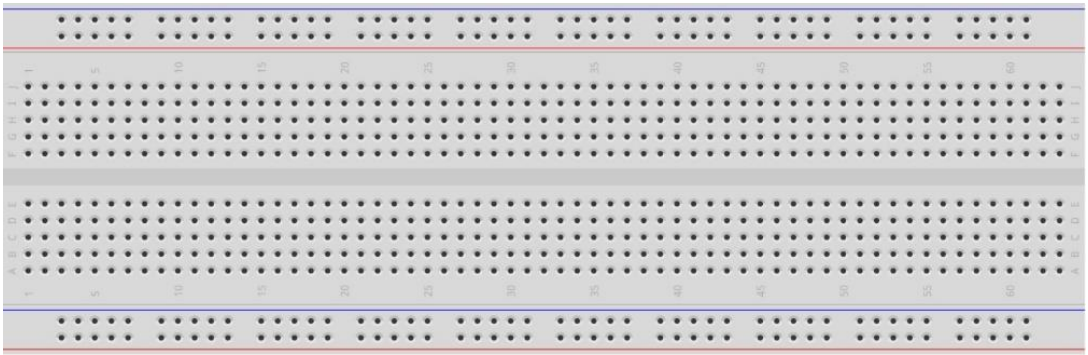
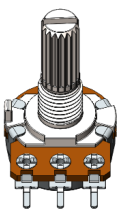
Chapter 10 Potentiometer & LED

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

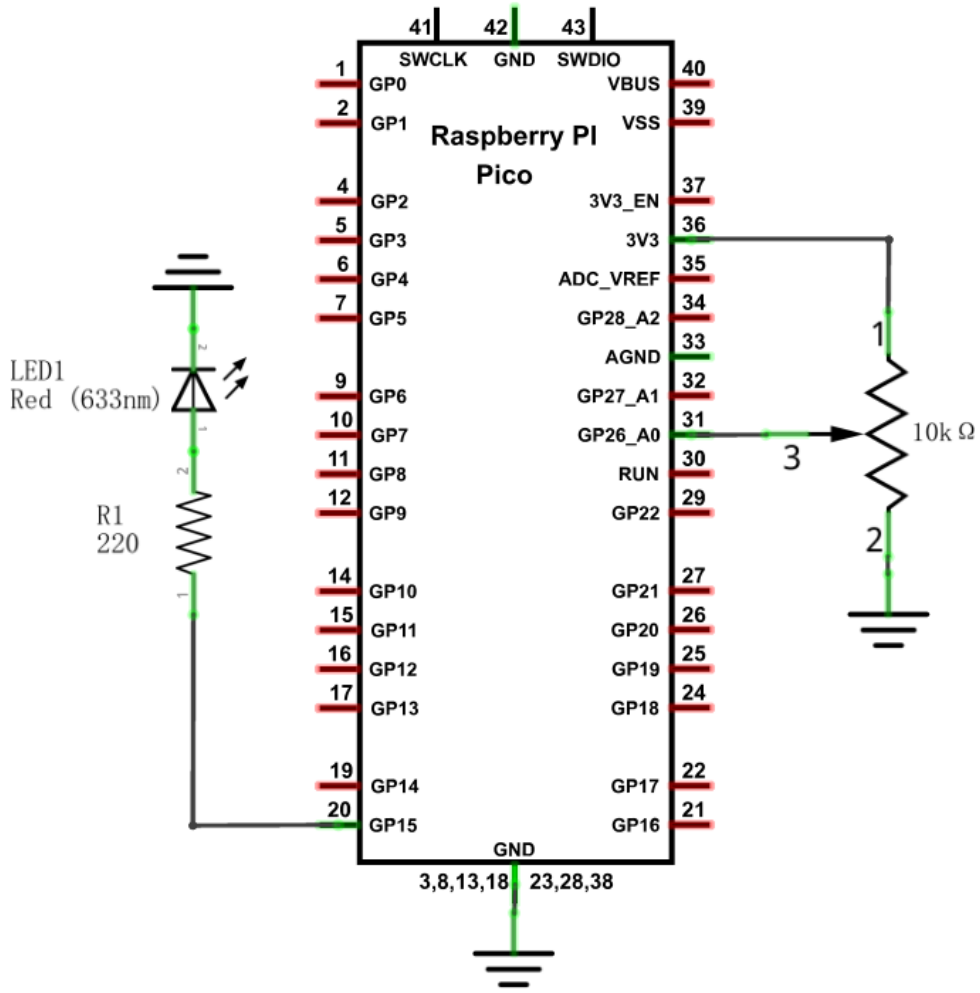
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

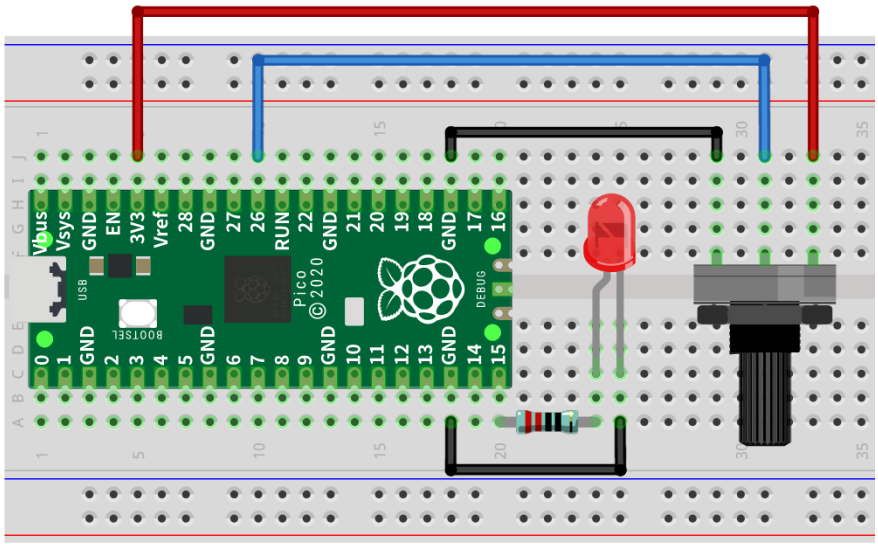
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td><td>Jumper
</td></tr>
</table>

Circuit

Schematic diagram



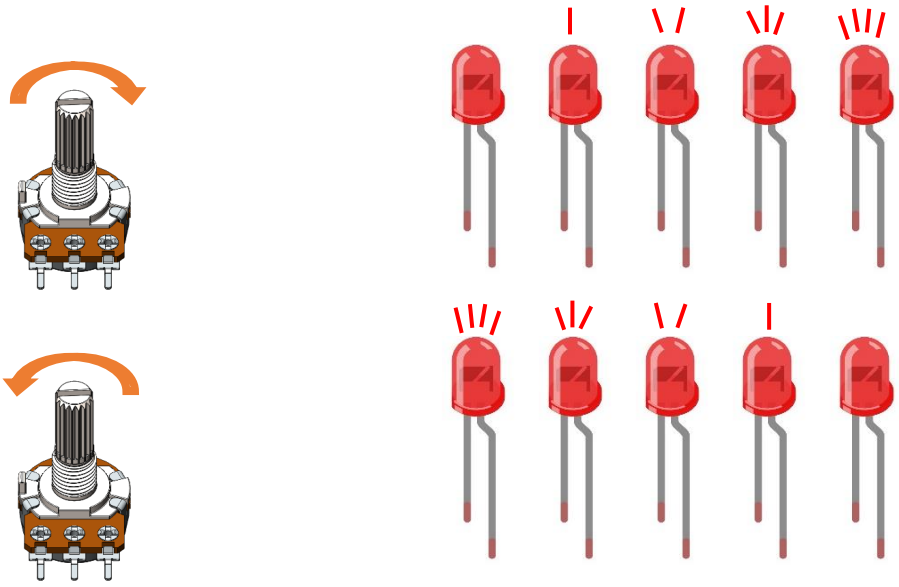
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_10.1_Softlight

Download the code to 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:

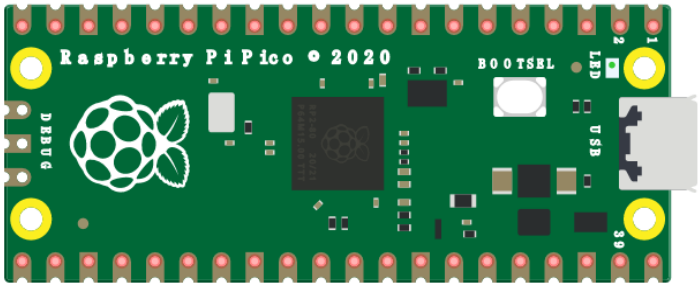
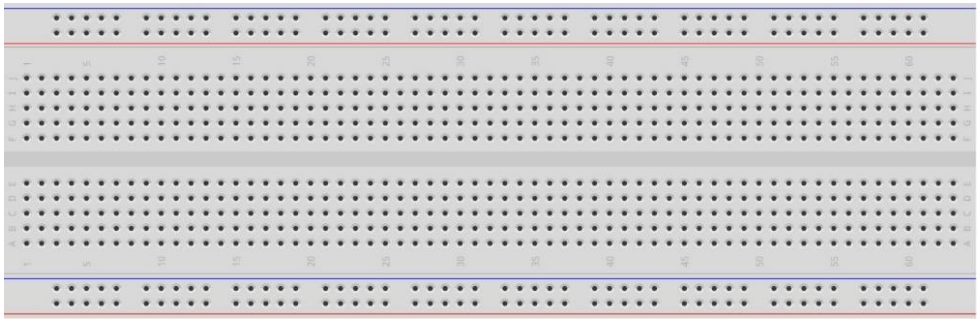
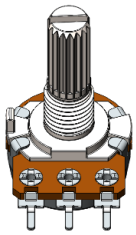
```
#define PIN_ADC0         26
#define PIN_LED          15

void setup() {
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  int adcVal = analogRead(PIN_ADC0); //read adc
  analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
  delay(10);
}
```

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

Project 10.2 Soft Colorful Light

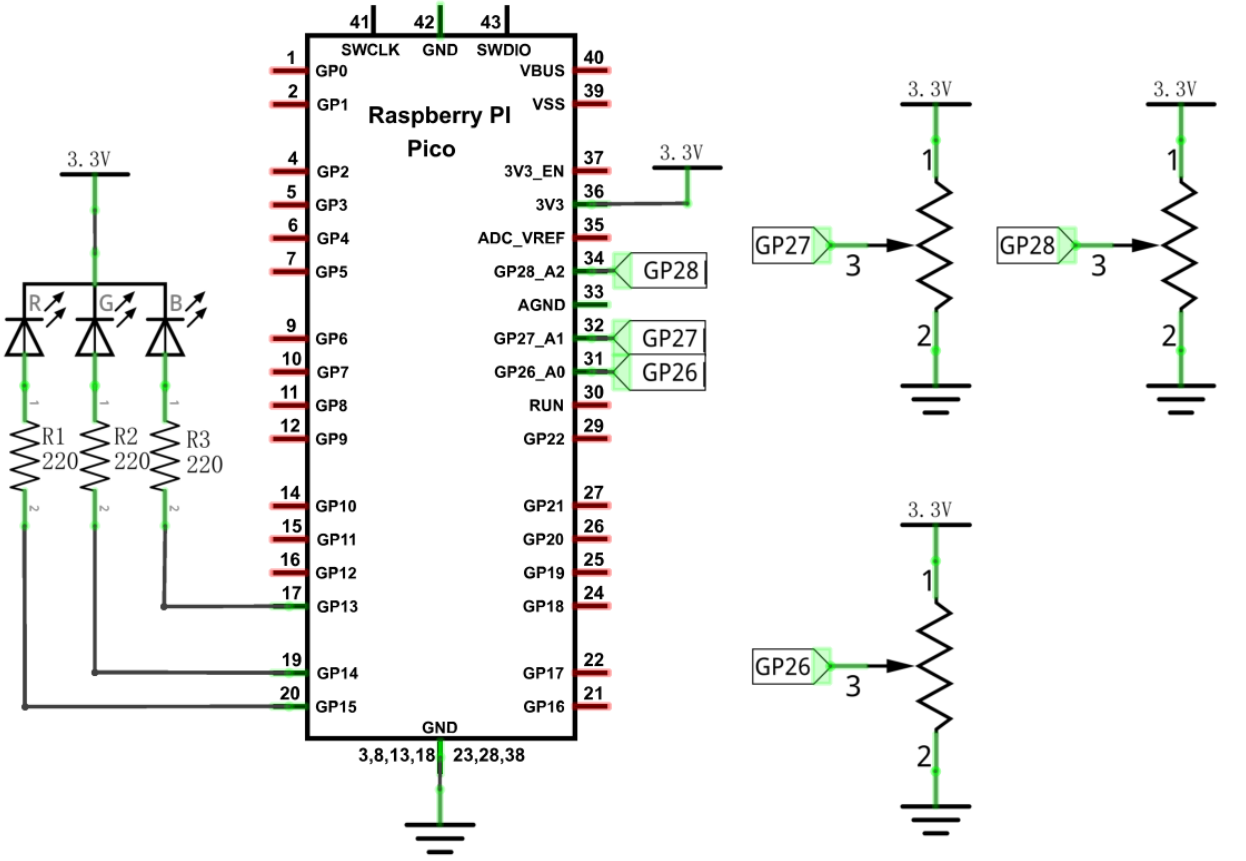
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED,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

Component List

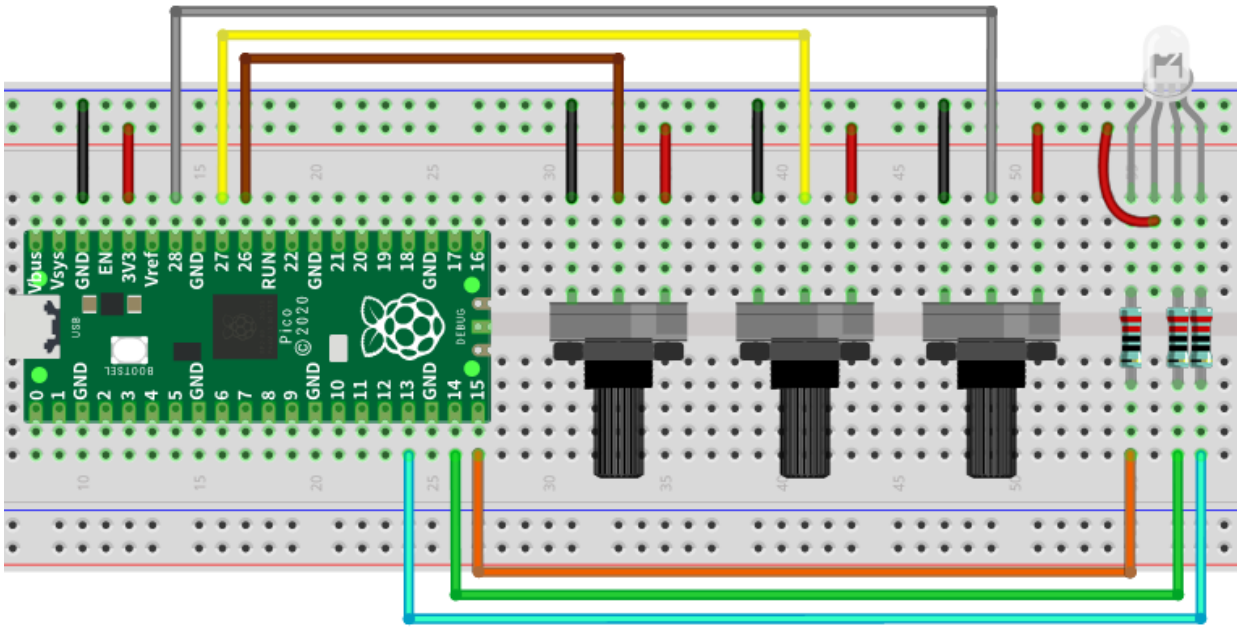
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td>Resistor 220Ω x3
  </td><td>RGBLED x1
  </td><td>Jumper
  </td></tr>
</table>

Circuit

Schematic diagram

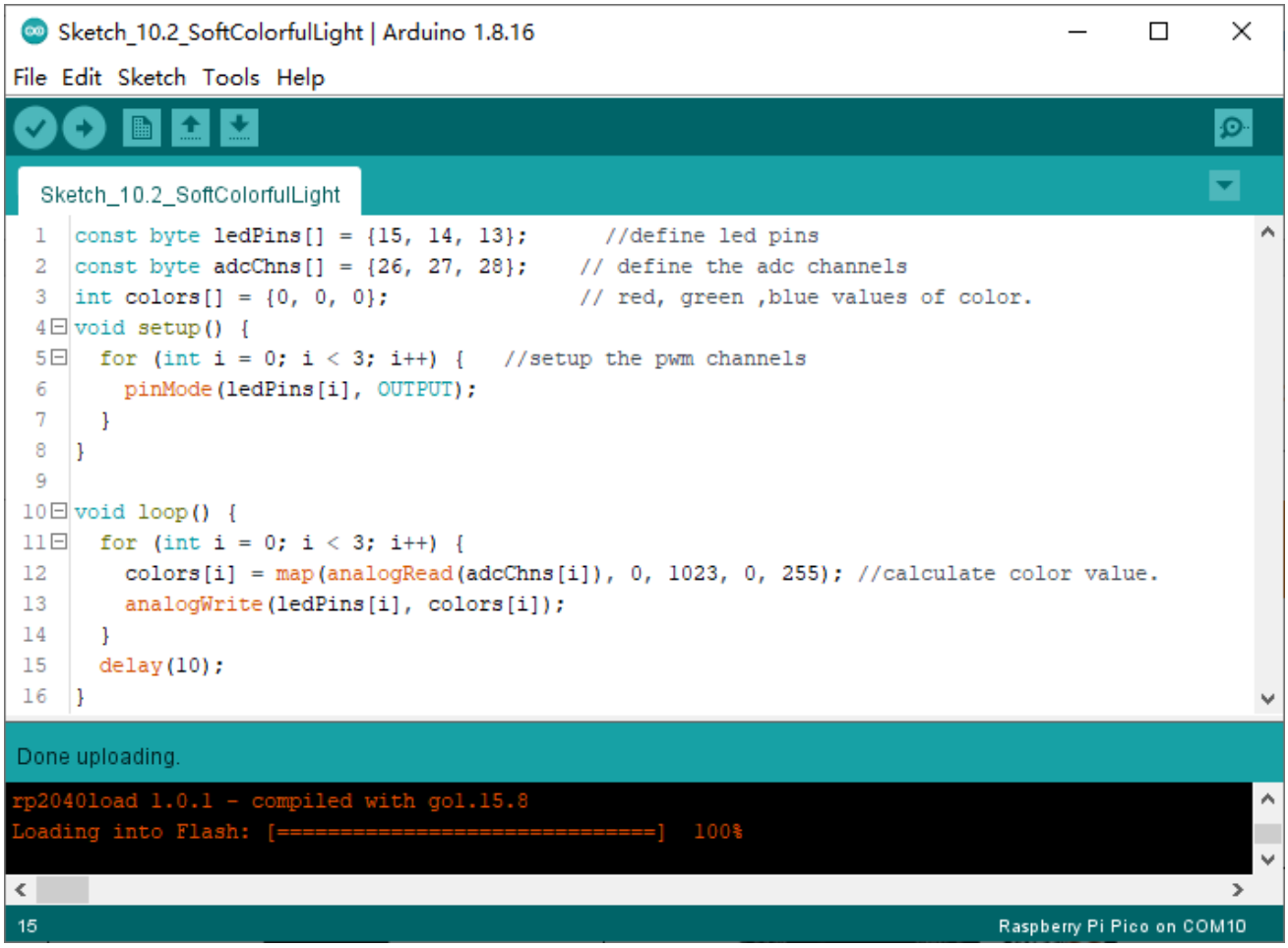


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_10.2_SoftColorfulLight



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

The following is the program code:

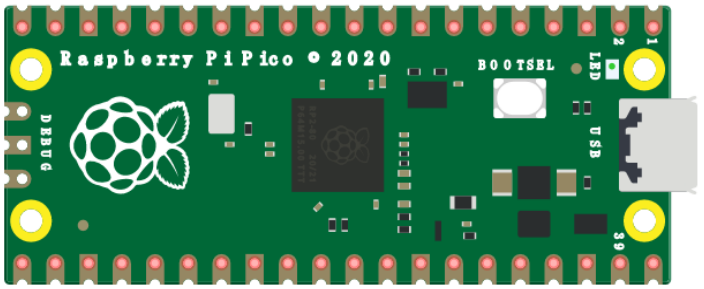
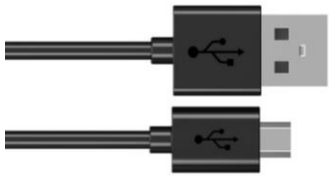
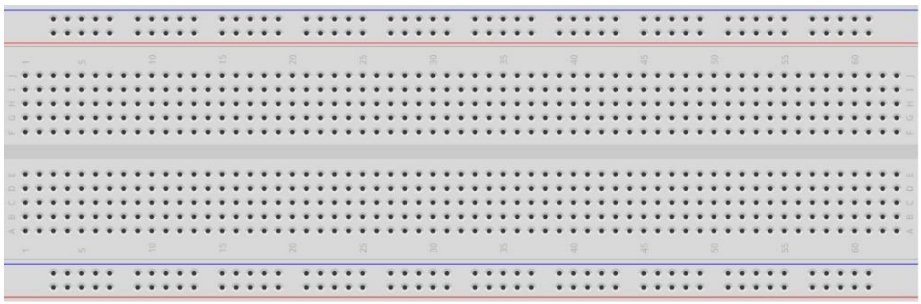
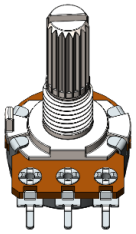
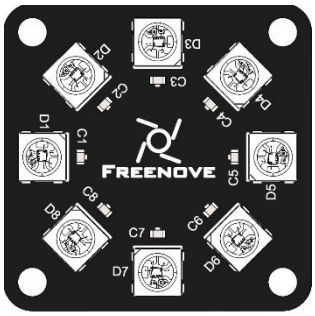
```
const byte ledPins[] = {15, 14, 13};      //define led pins
const byte adcChns[] = {26, 27, 28};    // define the adc channels
int colors[] = {0, 0, 0};               // red, green ,blue values of color.
void setup() {
  for (int i = 0; i < 3; i++) {   //setup the pwm channels
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 3; i++) {
    colors[i] = map(analogRead(adcChns[i]), 0, 1023, 0, 255); //calculate color value.
    analogWrite(ledPins[i], colors[i]);
  }
  delay(10);
}
```

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

Project 10.3 Soft Rainbow Light

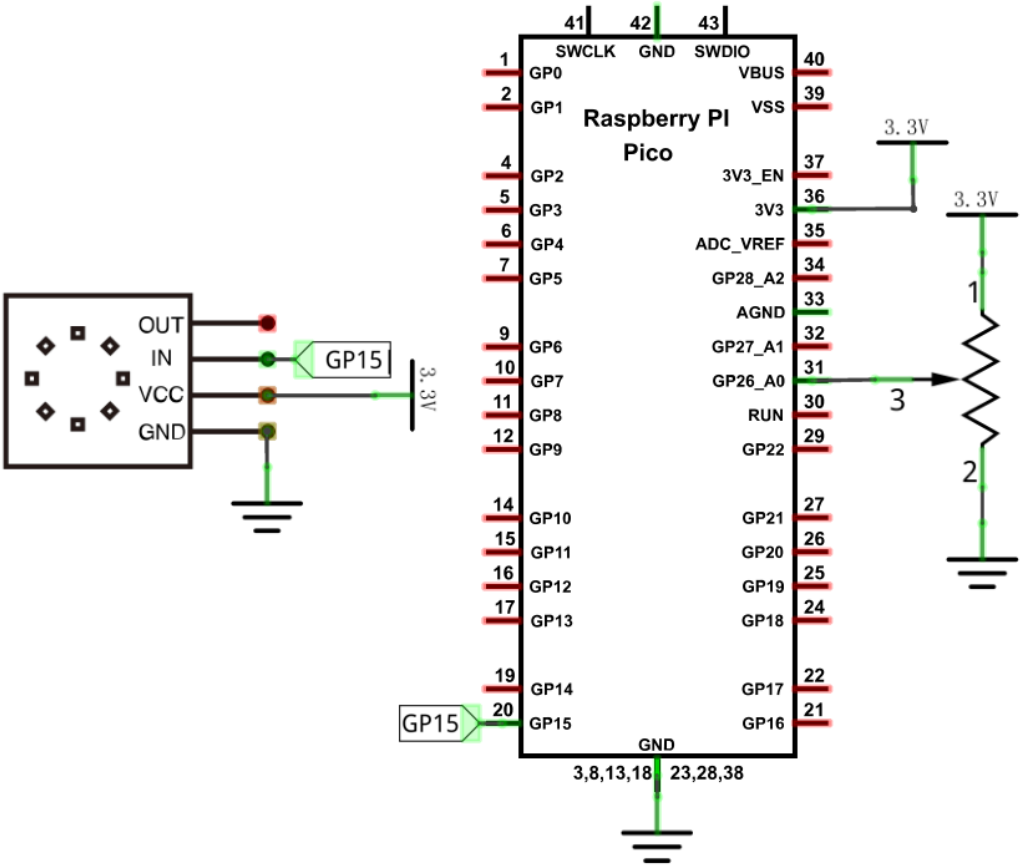
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

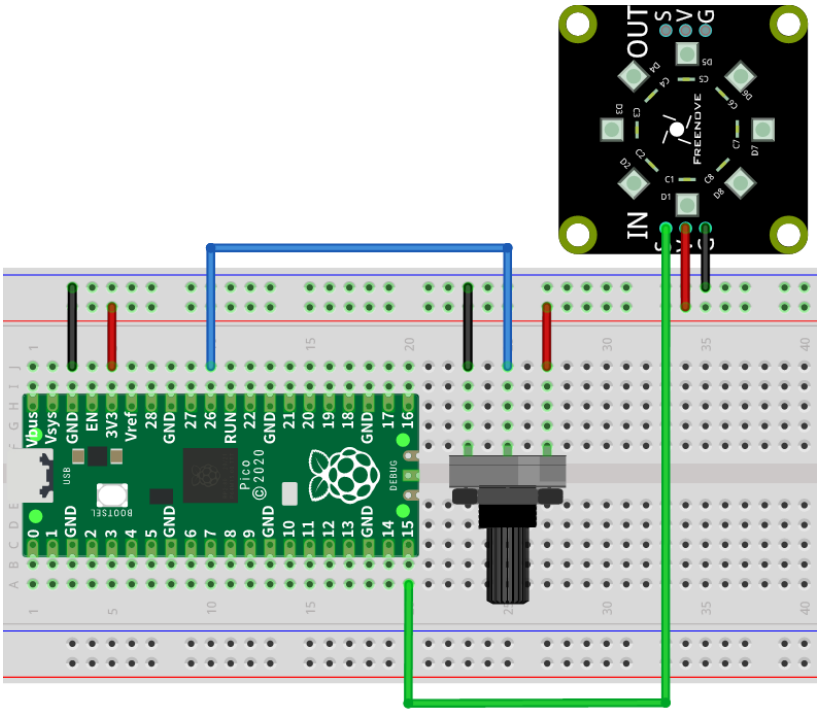
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Rotary potentiometer x1 	Freenove 8 RGB LED Module x1 	Jumper Jumper 

Circuit

Schematic diagram



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



Sketch

Sketch_10.3_Soft_Rainbow_Light

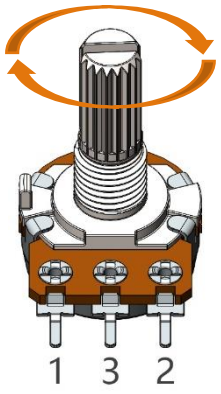
Sketch_10.3_SoftRainbowLight | Arduino 1.8.16

File Edit Sketch Tools Help

Sketch_10.3_SoftRainbowLight

```
#include <Adafruit_NeoPixel.h>

#define Pin       16
#define NumPixels 8
#define Pin_ADC0  26
int red = 0;
int green = 0;
int blue = 0;
int adcVal = 0;
Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);

void setup(){
  strip.begin();
  strip.setBrightness(20);
}
void loop(){
  adcVal = map(analogRead(Pin_ADC0), 0, 1023, 0, 255);
  for(int i=0; i< 8; i++) {
    Wheel(((i * 256 / 8) + adcVal) & 255);
    strip.setPixelColor(i, strip.Color(red, green, blue));
  }
  strip.show();
  delay(10);
}
```

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:

```
#include <Adafruit_NeoPixel.h>

#define Pin       16
#define NumPixels 8
#define Pin_ADC0  26
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.

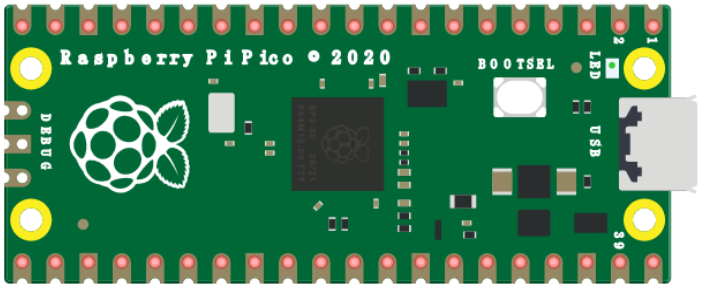
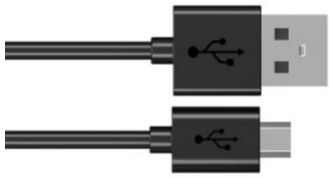
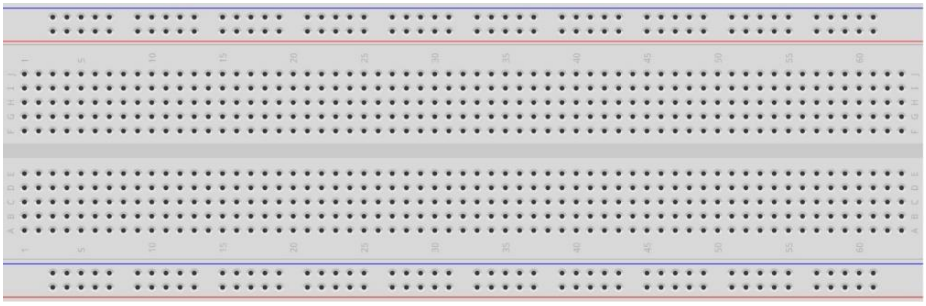
Chapter 11 Photoresistor & LED

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.

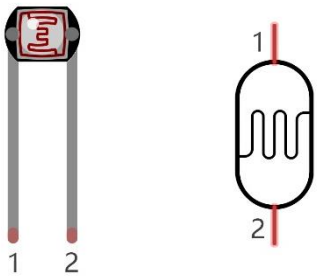
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">Resistor
220Ω x1
 
10KΩ x1
  </td><td>LED x1
  </td><td>Jumper
</td></tr>
</table>

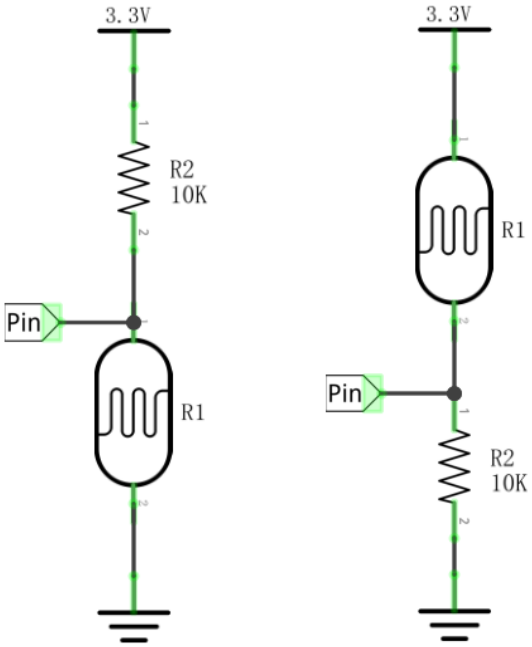
Component Knowledge

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:

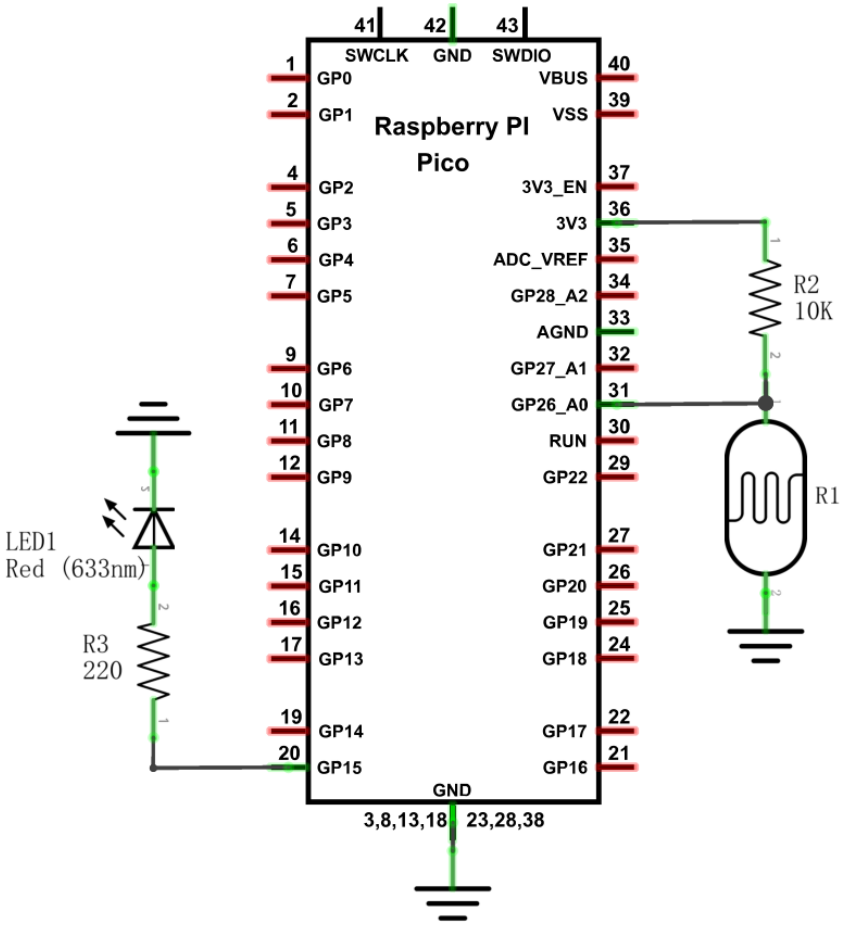


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

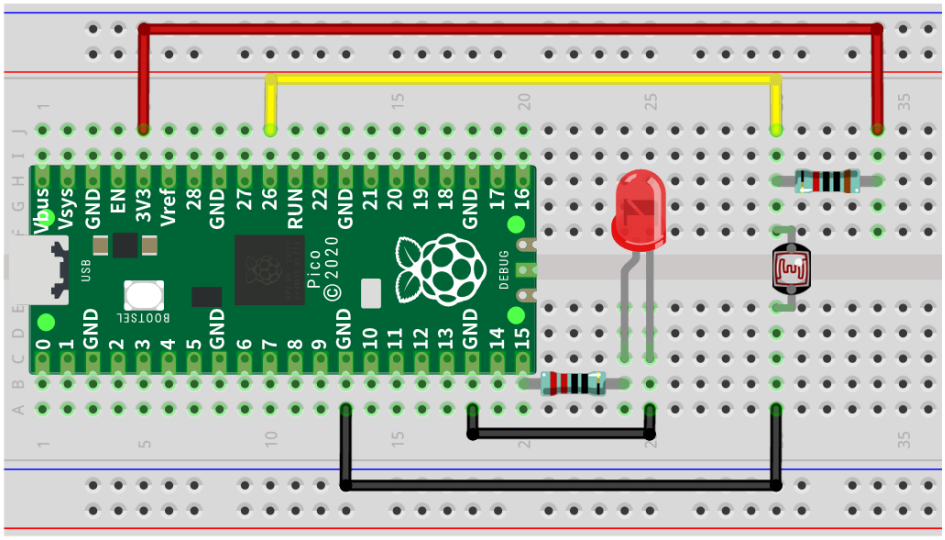
Circuit

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



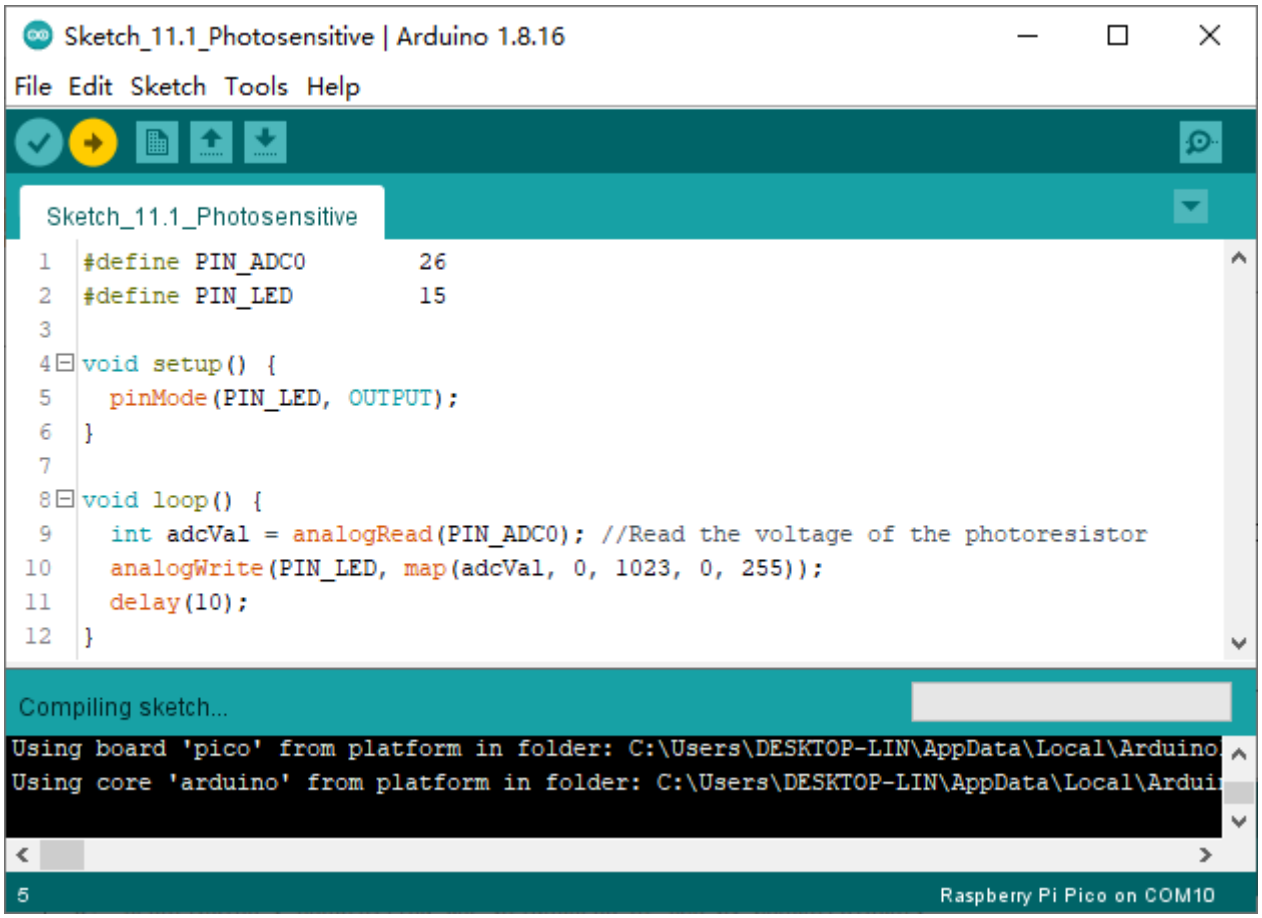
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



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.

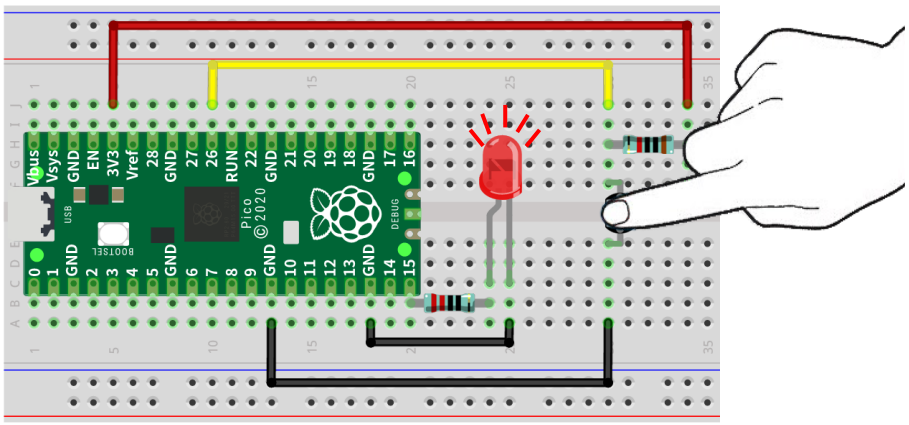
Sketch_11.1_Nightlamp



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

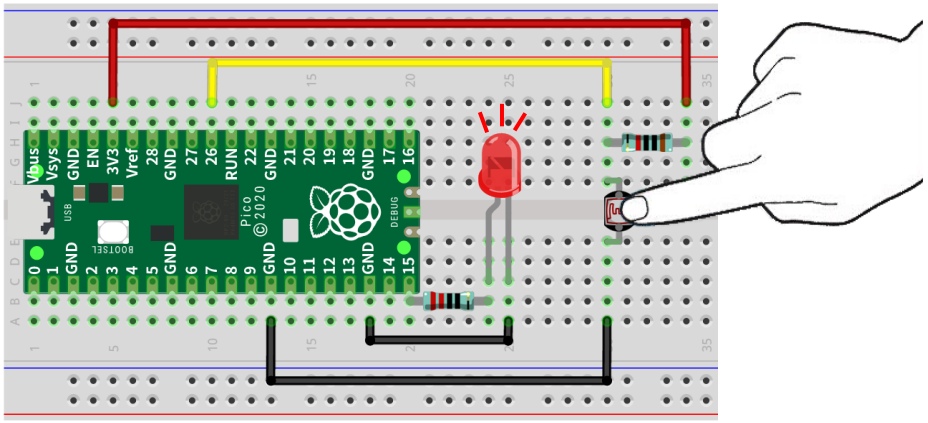
If you have any concerns, please contact us via: support@freenove.com

Fully cover the photoresistor:

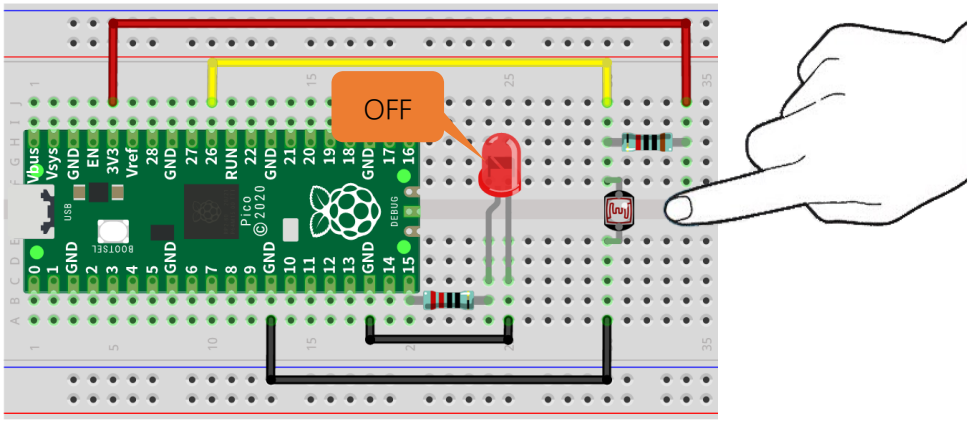


Any concerns? ✉ support@freenove.com

Half cover the photoresistor:



Not cover the photoresistor:



The following is the program code:

```
#define PIN_ADC0        26
#define PIN_LED         15

void setup() {
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  int adcVal = analogRead(PIN_ADC0); //Read the voltage of the photoresistor
  analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
  delay(10);
}
```

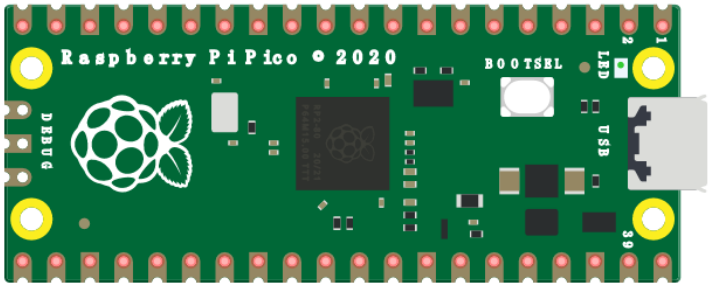
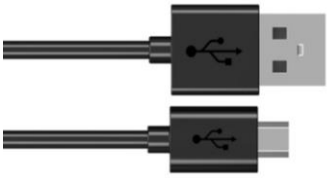
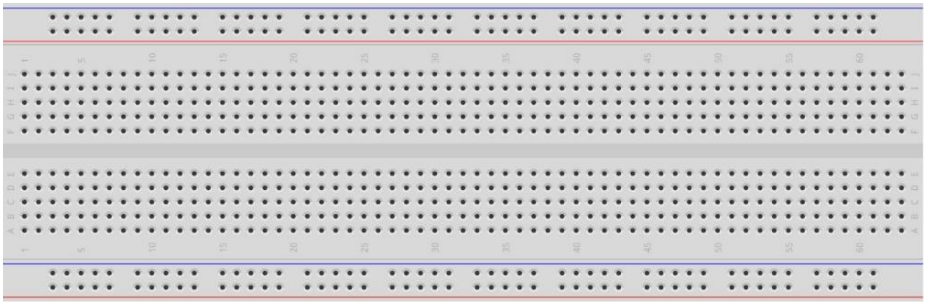
Chapter 12 Thermistor

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

Project 12.1 Thermometer

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

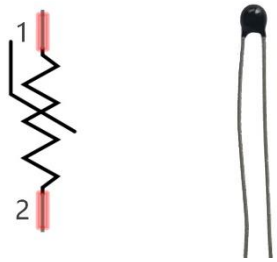
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Thermistor x1
  </td><td>Resistor 10kΩ x1
  </td><td>Jumper
  </td></tr>
</table>

Component Knowledge

Thermistor

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



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

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

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

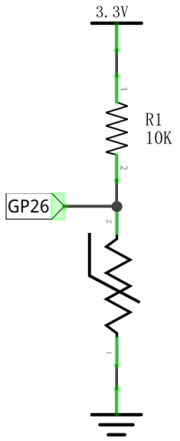
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



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

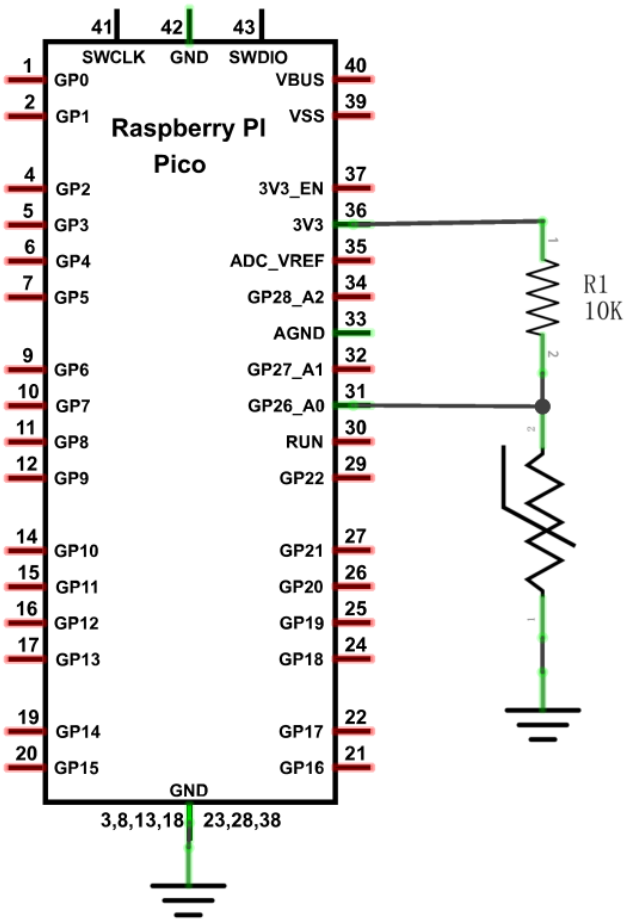
Therefore, the temperature formula can be derived as:

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

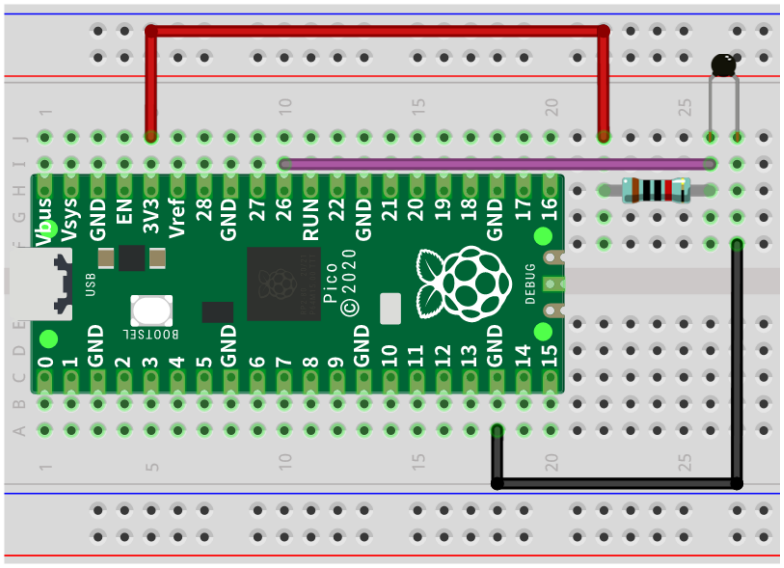
Circuit

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

Schematic diagram



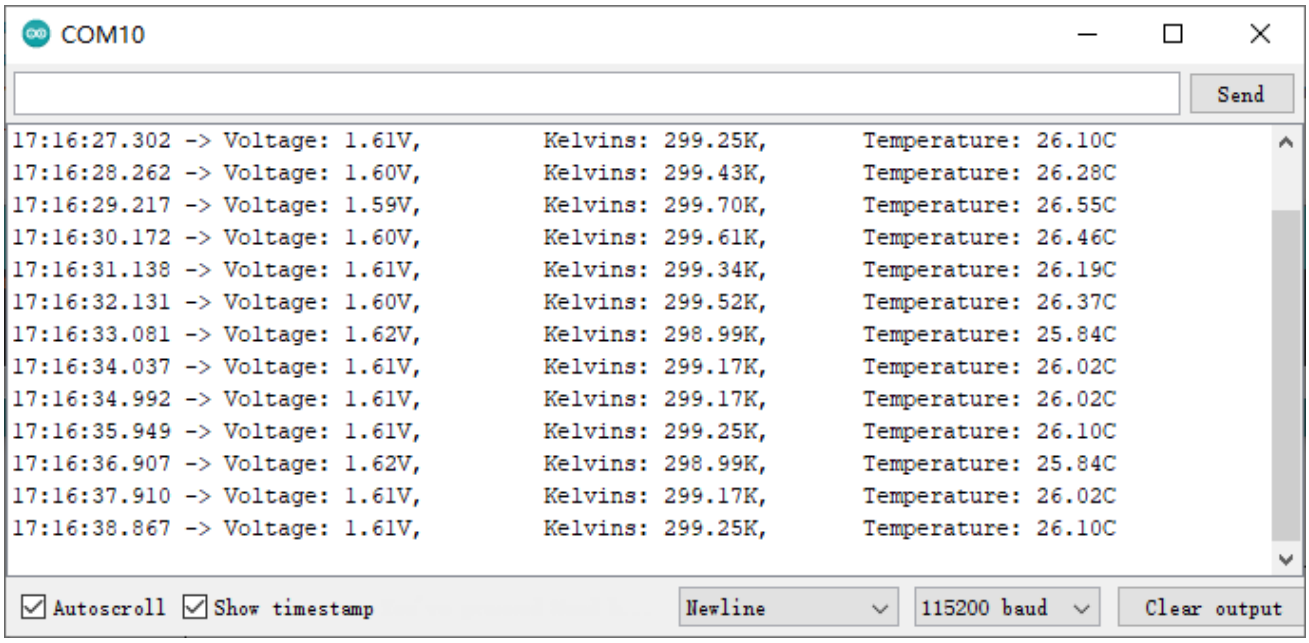
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_12.1_Thermometer

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.



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

The following is the code:

```
#define PIN_ADC0   26
void setup() {
  Serial.begin(115200);
}

void loop() {
  int adcValue = analogRead(PIN_ADC0); //read ADC pin
  double voltage = (float)adcValue / 1023.0 * 3.3;// calculate voltage
  double Rt = 10 * voltage / (3.3 - voltage);//calculate resistance value of thermistor
  double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0);//calculate temperature (Kelvin)
  double tempC = tempK - 273.15;//calculate temperature (Celsius)
  Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String(tempK) + "K,\t" + "Temperature: " + String(tempC) + "C");
  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.

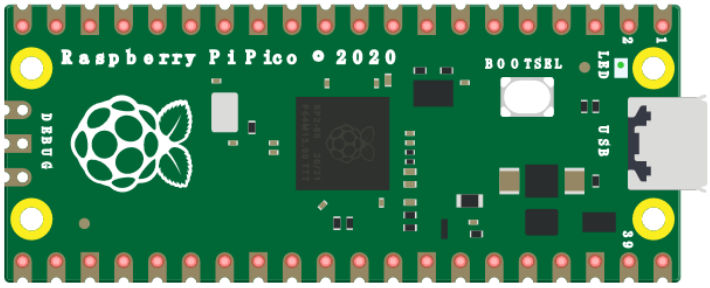
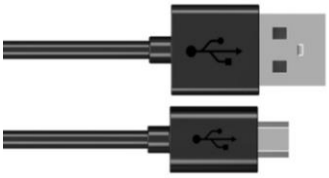
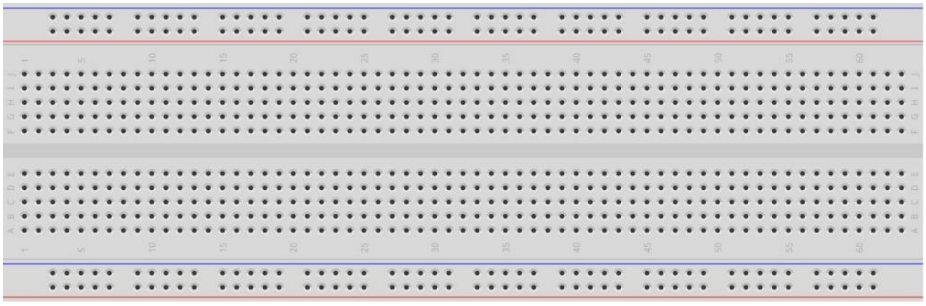
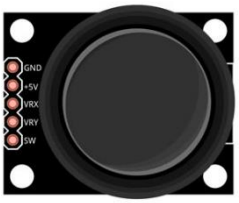
Chapter 13 Joystick

In the previous chapter, we have learned how to use rotary potentiometer. Now, let's learn a new electronic module Joystick which 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.

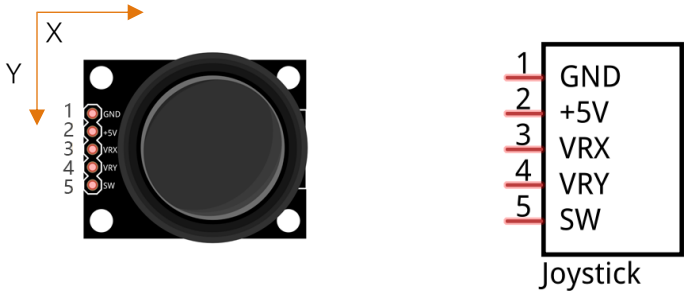
Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Joystick x1 	Jumper

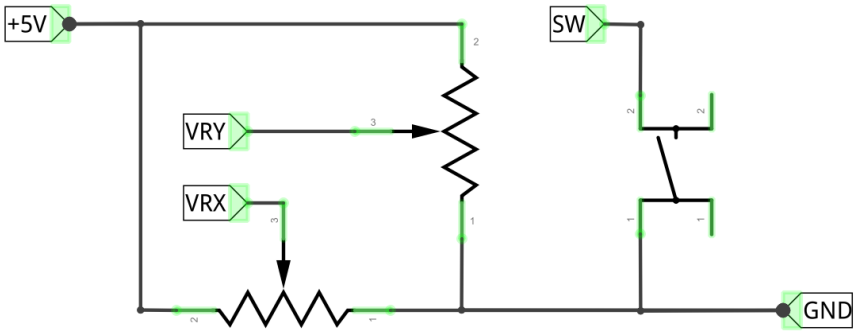
Component Knowledge

Joystick

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



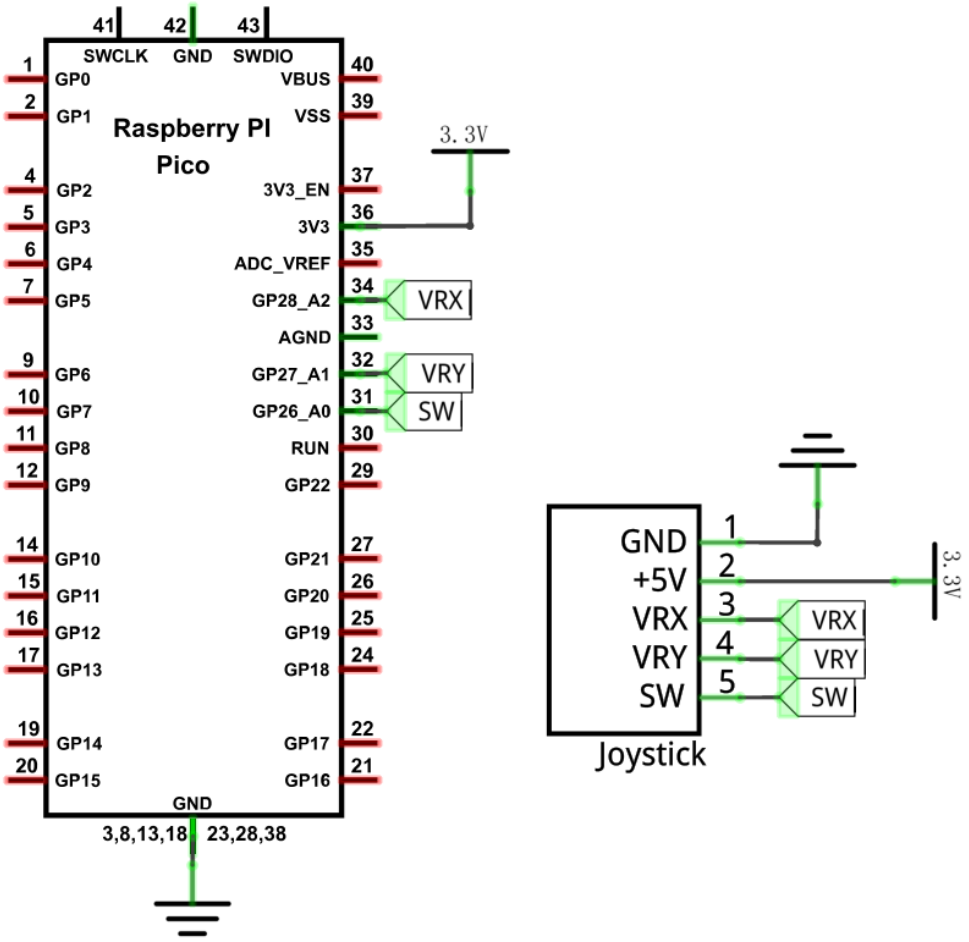
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



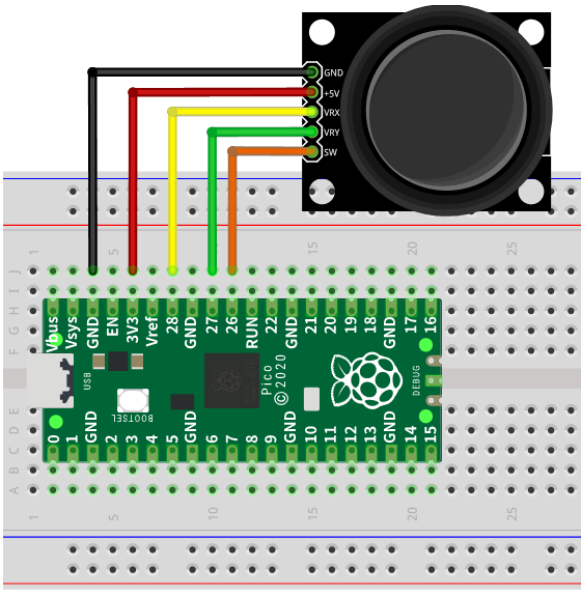
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



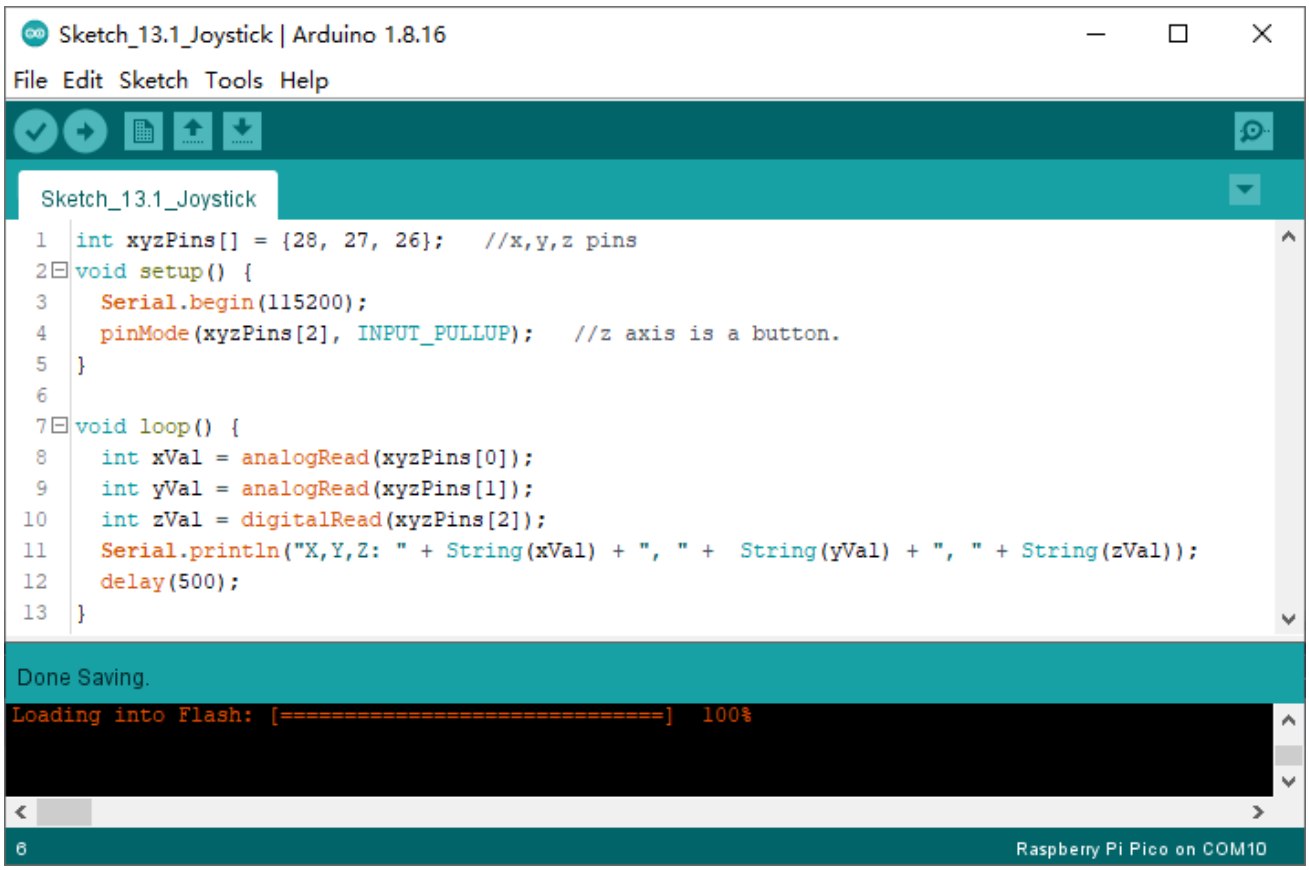
Hardware connection. If you need any support, please feel free to contact us via:support@freenove.com



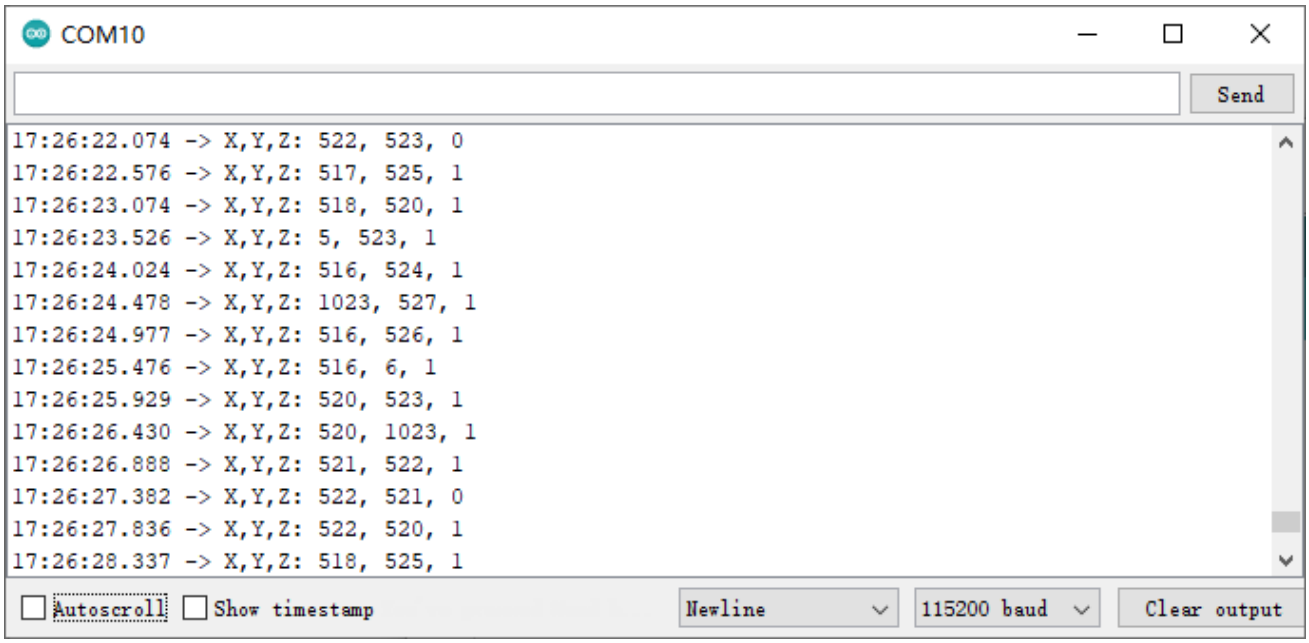
Sketch

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

Sketch_13.1_Joystick



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.



The following is the code:

```
int xyzPins[] = {28, 27, 26};   //x,y,z pins

void setup() {
  Serial.begin(115200);
  pinMode(xyzPins[2], INPUT_PULLUP);   //z axis is a button.
}

void loop() {
  int xVal = analogRead(xyzPins[0]);
  int yVal = analogRead(xyzPins[1]);
  int zVal = digitalRead(xyzPins[2]);
  Serial.println("X,Y,Z: " + String(xVal) + ", " +  String(yVal) + ", " + String(zVal));
  delay(500);
}
```

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

```
  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 digitalRead () to read the value of axis Z, then display them.

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

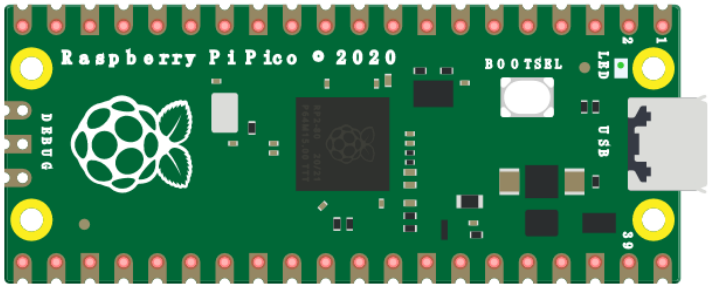
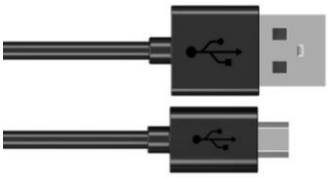
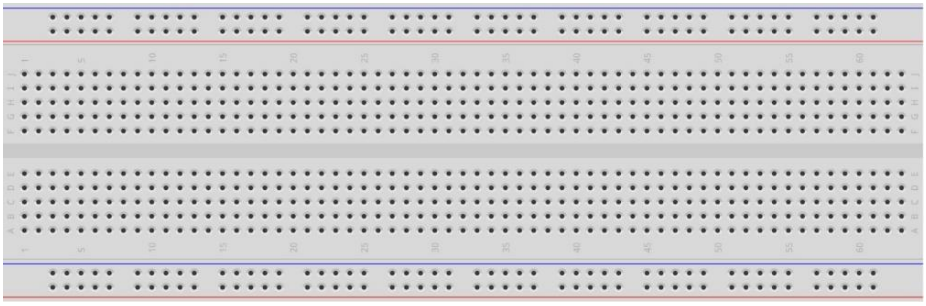
Chapter 14 74HC595 & LED Bar Graph

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

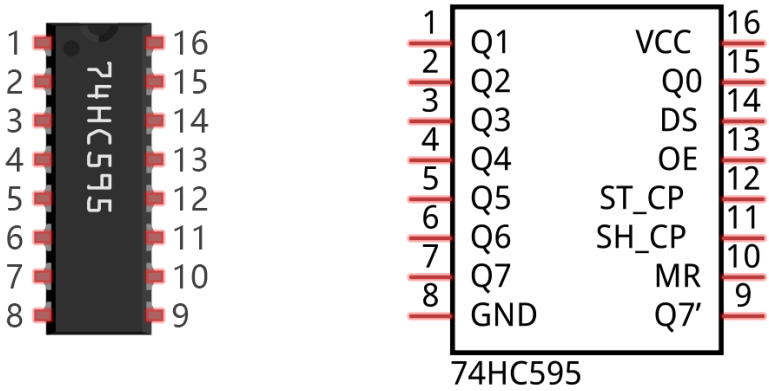
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x1
  </td><td>LED Bar Graph x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper
</td></tr>
</table>

Related Knowledge

74HC595

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



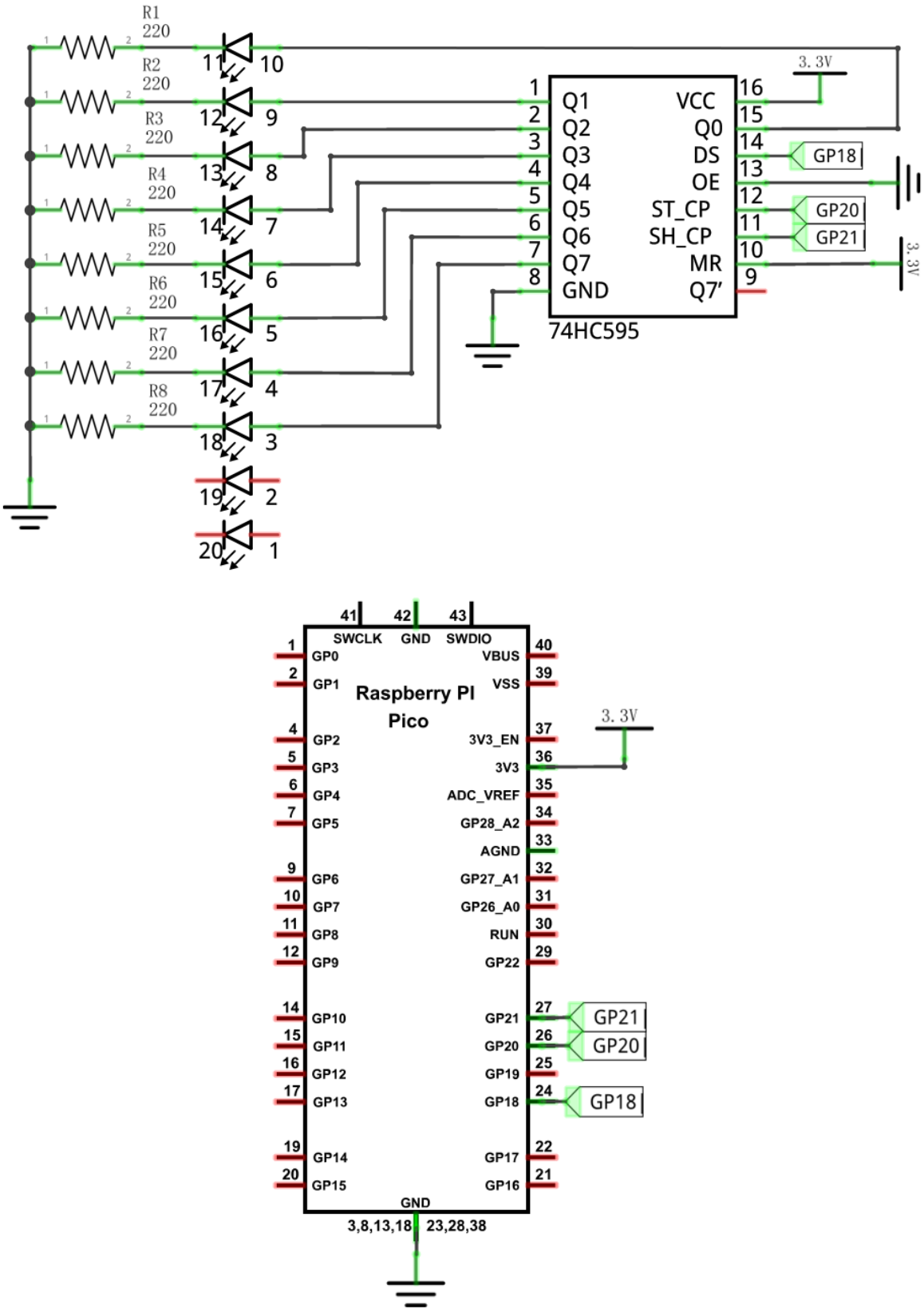
The ports of the 74HC595 chip are described as follows:

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

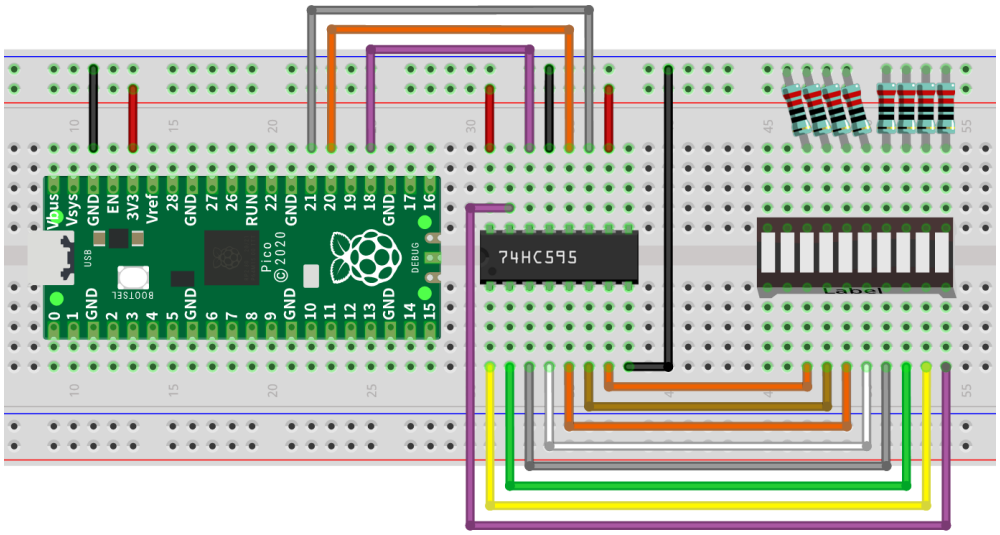
For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



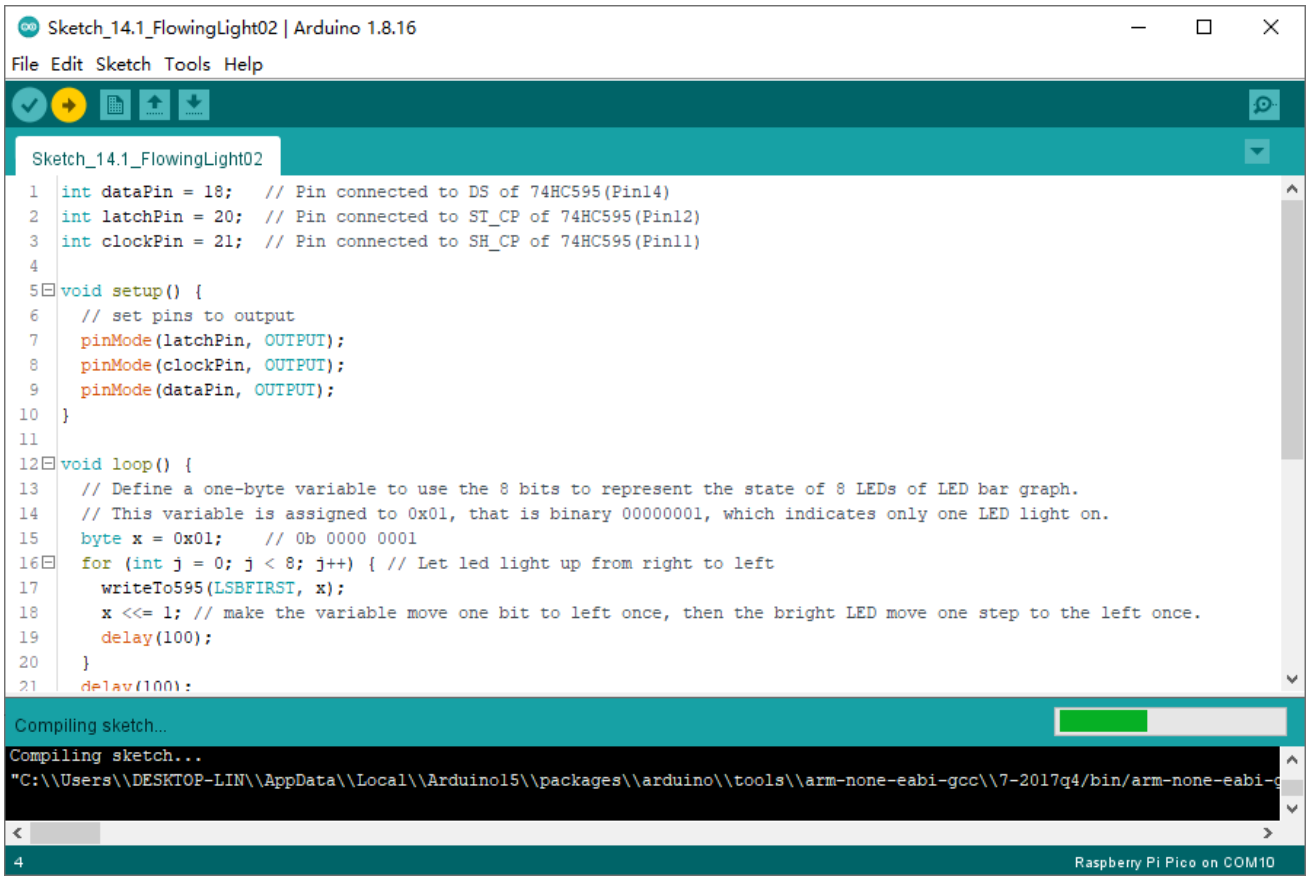
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

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

Sketch_14.1_FlowingLight2



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

The following is the program code:

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

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

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

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

```
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.

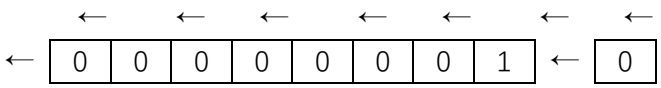
The subfunction writeTo595() is used to write data to 74HC595 and immediately output on the port of 74HC595.

Reference

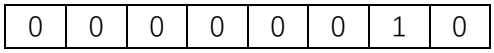
<< operator

"<<" 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;

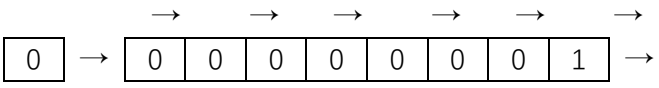


The result of x is 2（binary 00000010）.

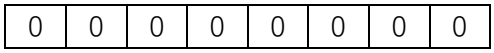


There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

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/shiftout/

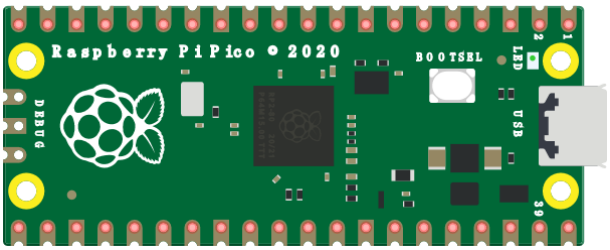
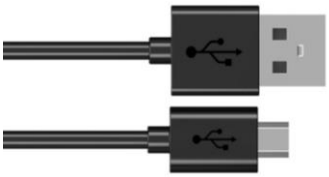
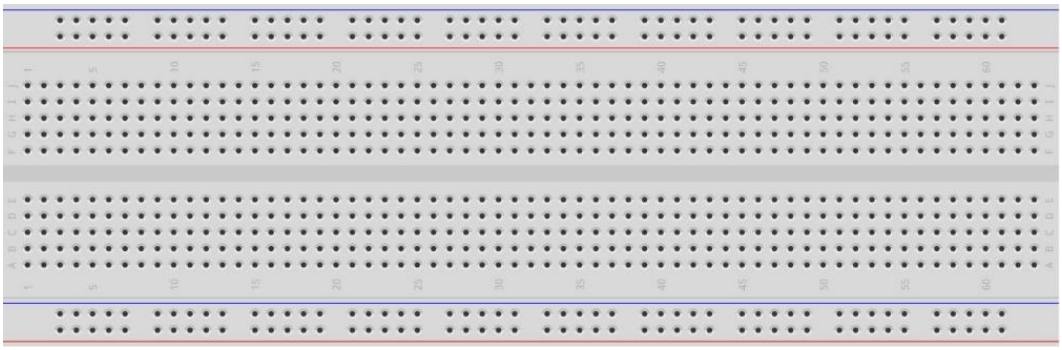
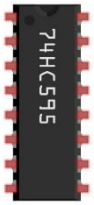
Chapter 15 74HC595 & 7-Segment Display.

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".

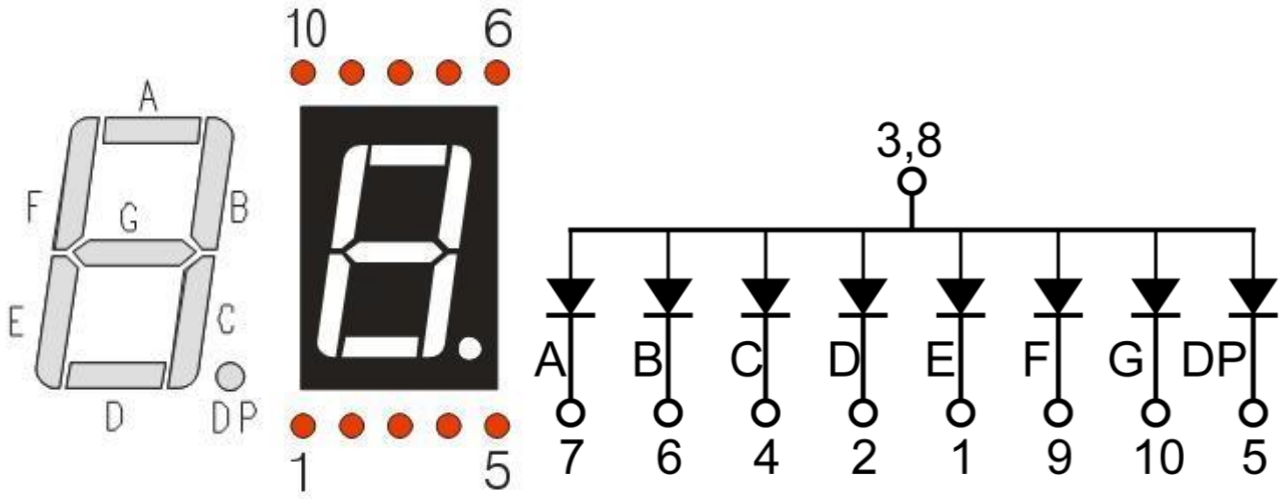
Component List

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

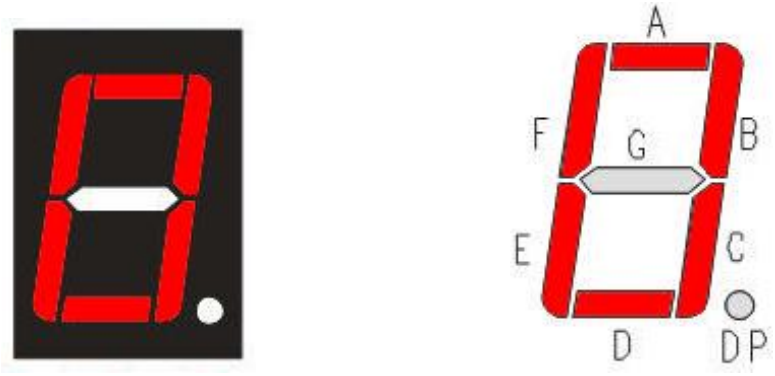
Component Knowledge

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.



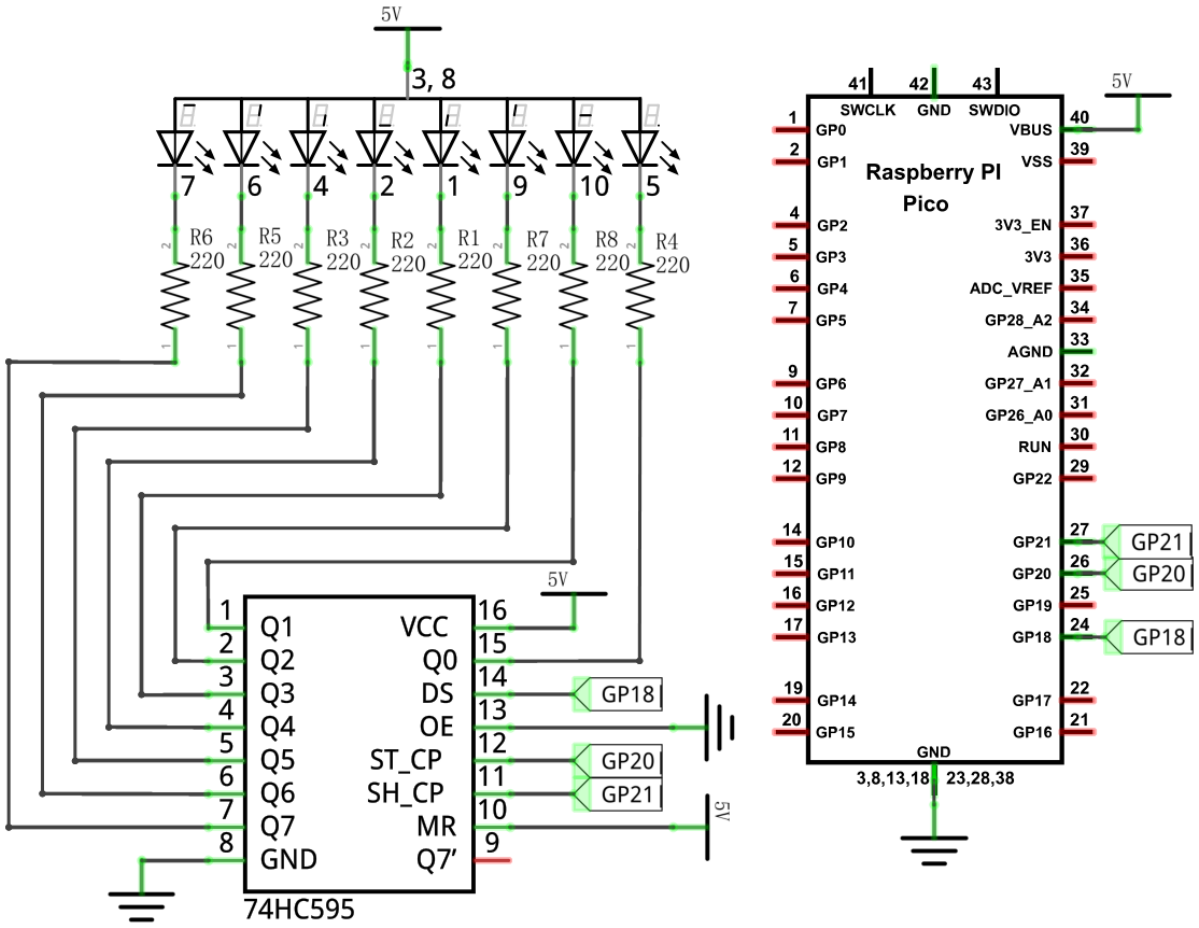
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”,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

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

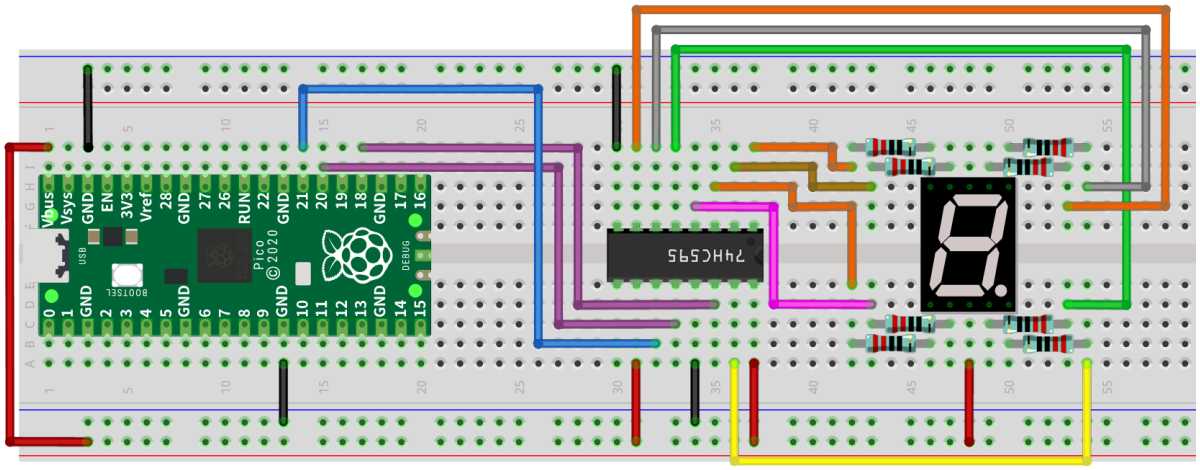
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0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

Circuit

Schematic diagram



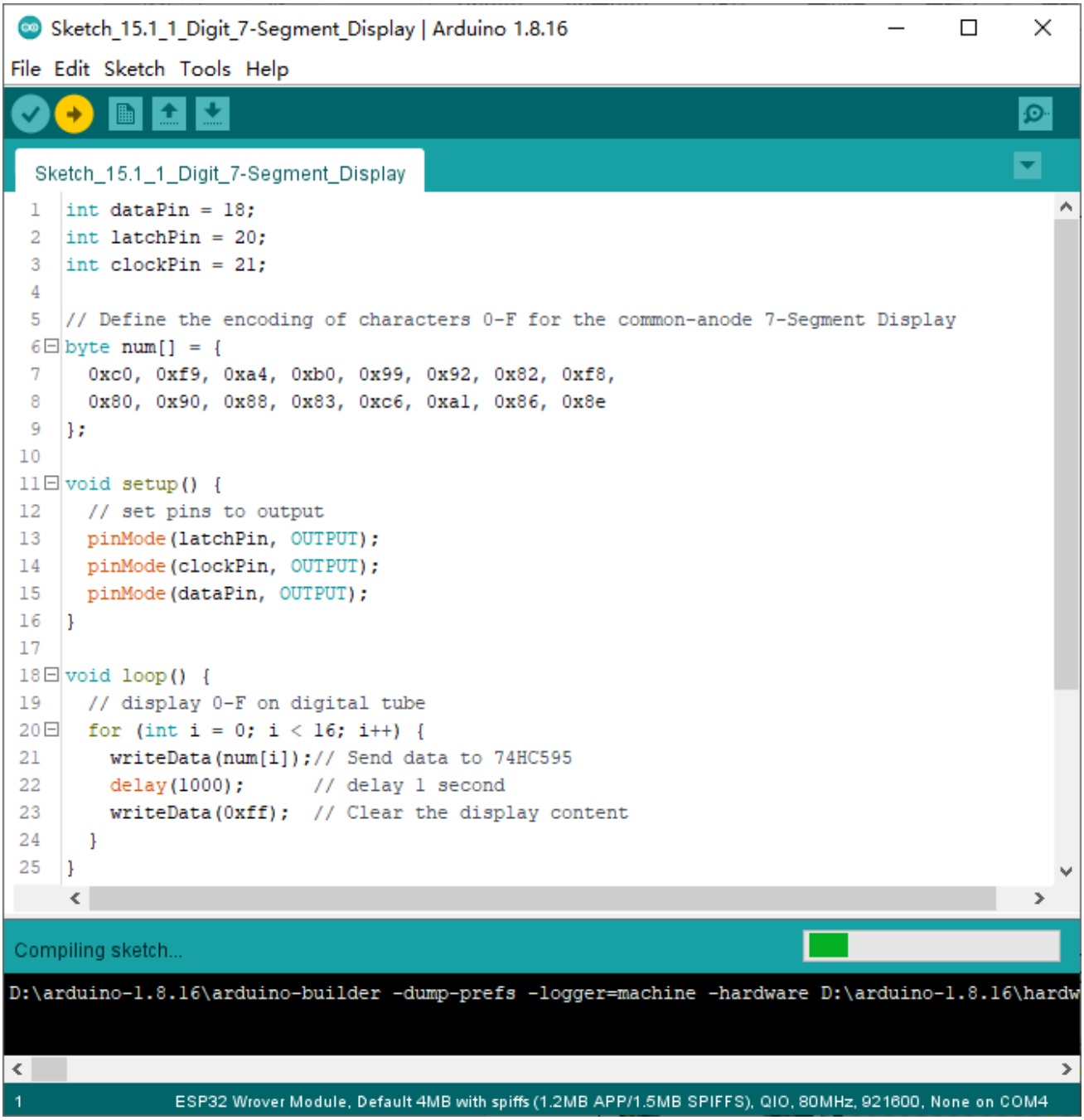
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



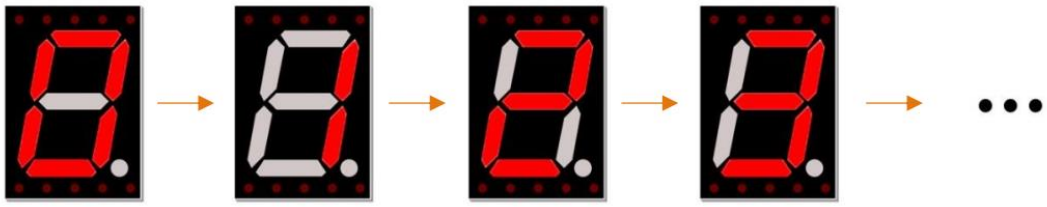
Sketch

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

Sketch_15.1_7_Segment_Display



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



The following is the program code:

```
int dataPin = 18;           // Pin connected to DS of 74HC595 (Pin14)
int latchPin = 20;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 21;          // Pin connected to SH_CP of 74HC595 (Pin11)
// Define the encoding of characters 0-F for the common-anode 7-Segment Display
byte num[] = {
  0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
  0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
};

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

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

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

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

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

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

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

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

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

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

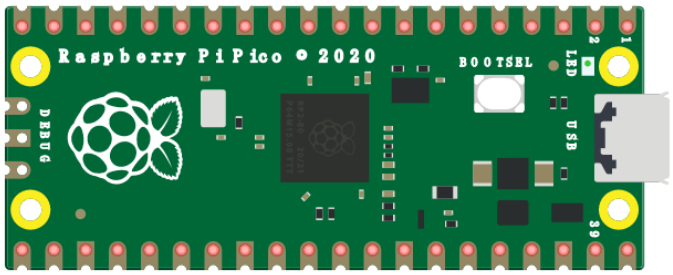
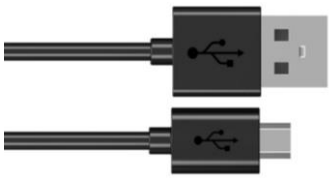
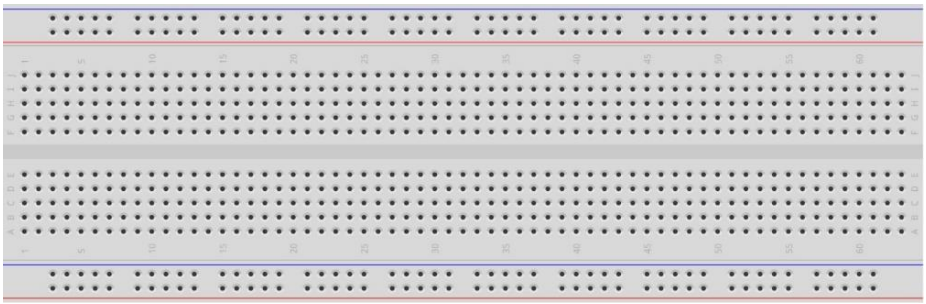
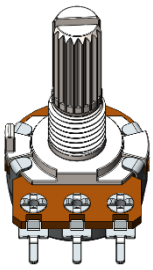
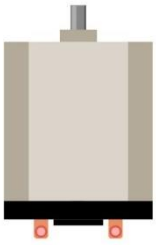
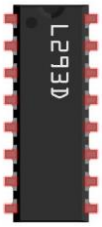
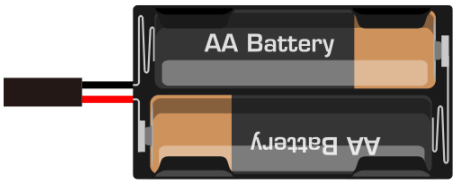
```
shiftOut(dataPin,clockPin, LSBFIRST, value & 0x7f);
```

Chapter 16 L293D & Motor

Project 16.1 Control Motor with Potentiometer

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

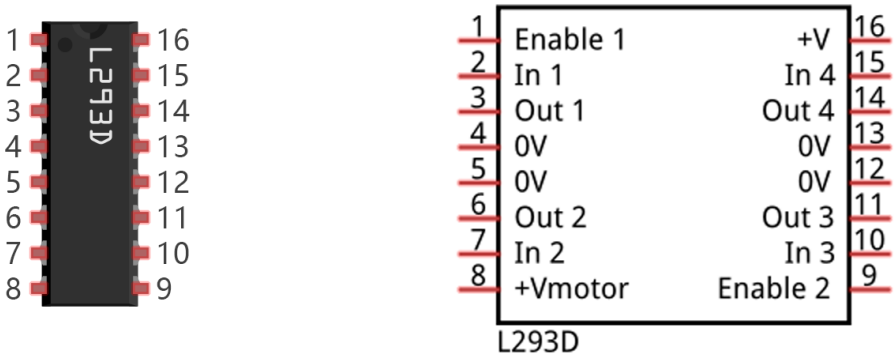
Component List

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).

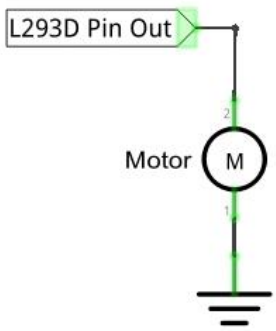


Port description of L293D module is as follows:

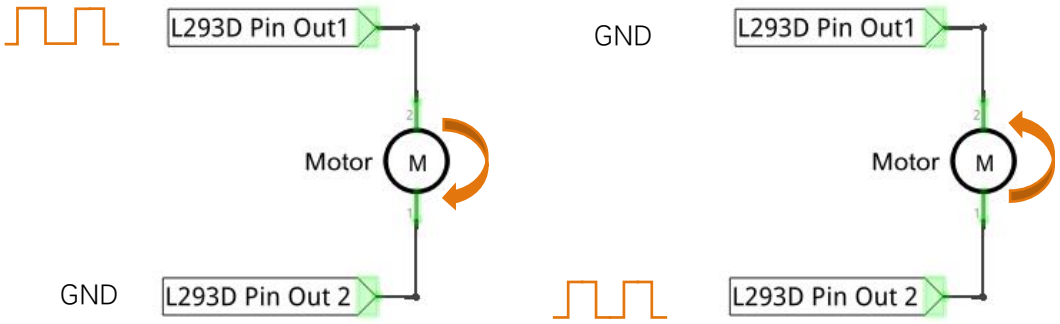
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4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



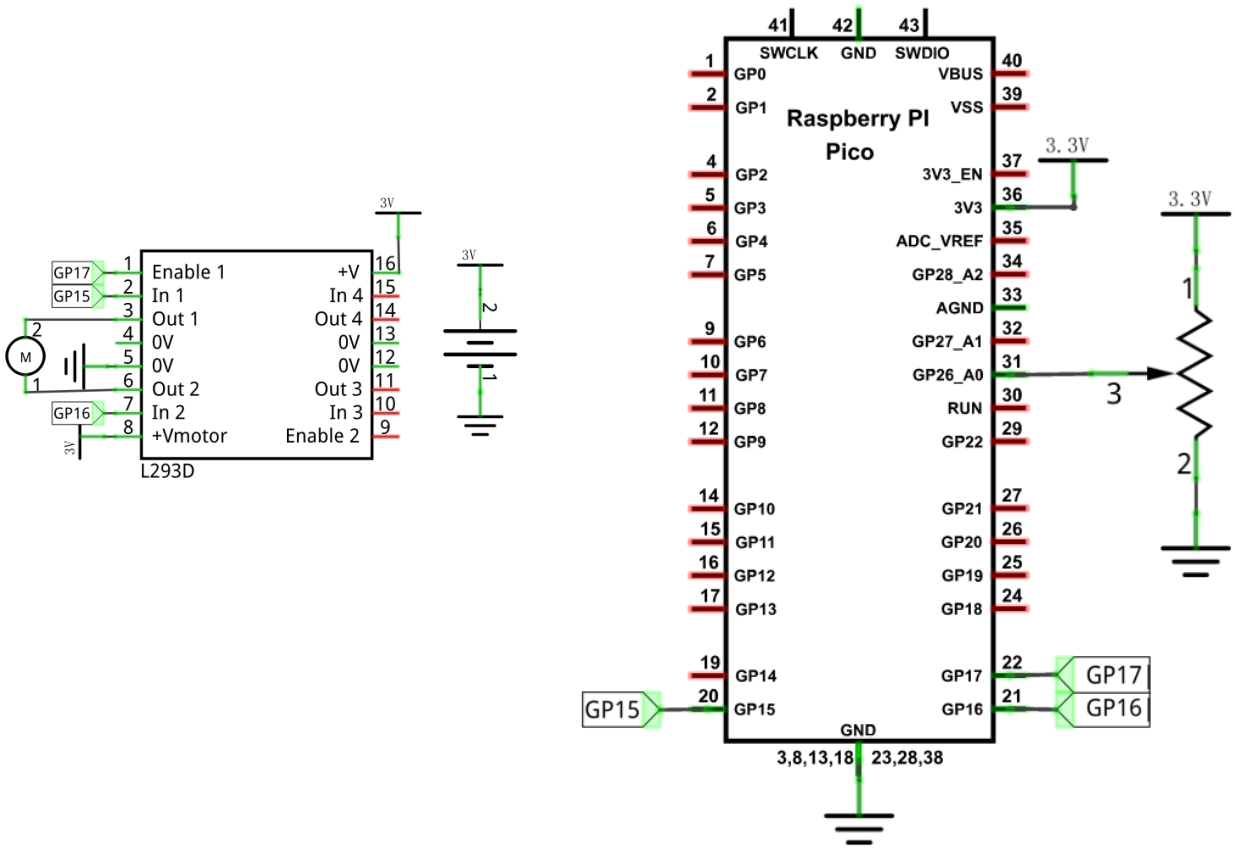
The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the direction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



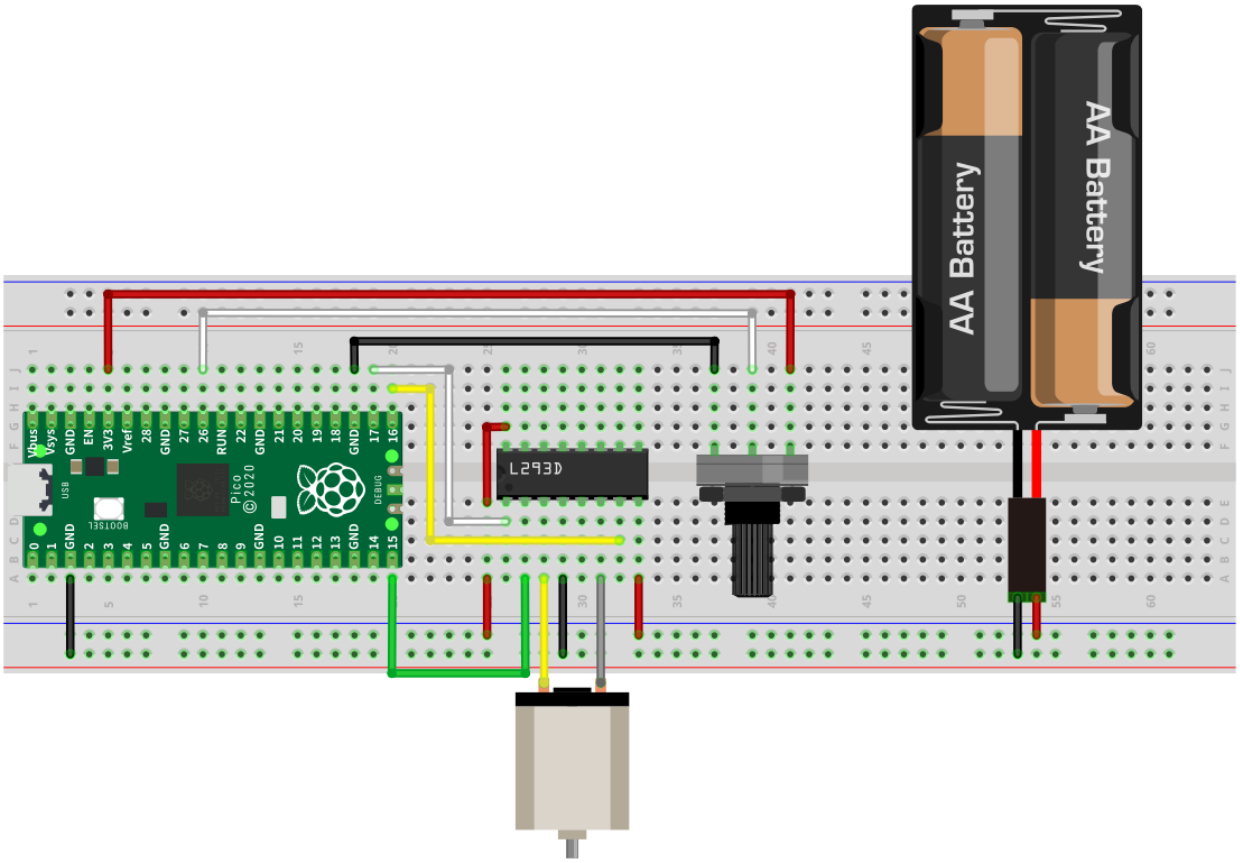
In practical use the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

Schematic diagram



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

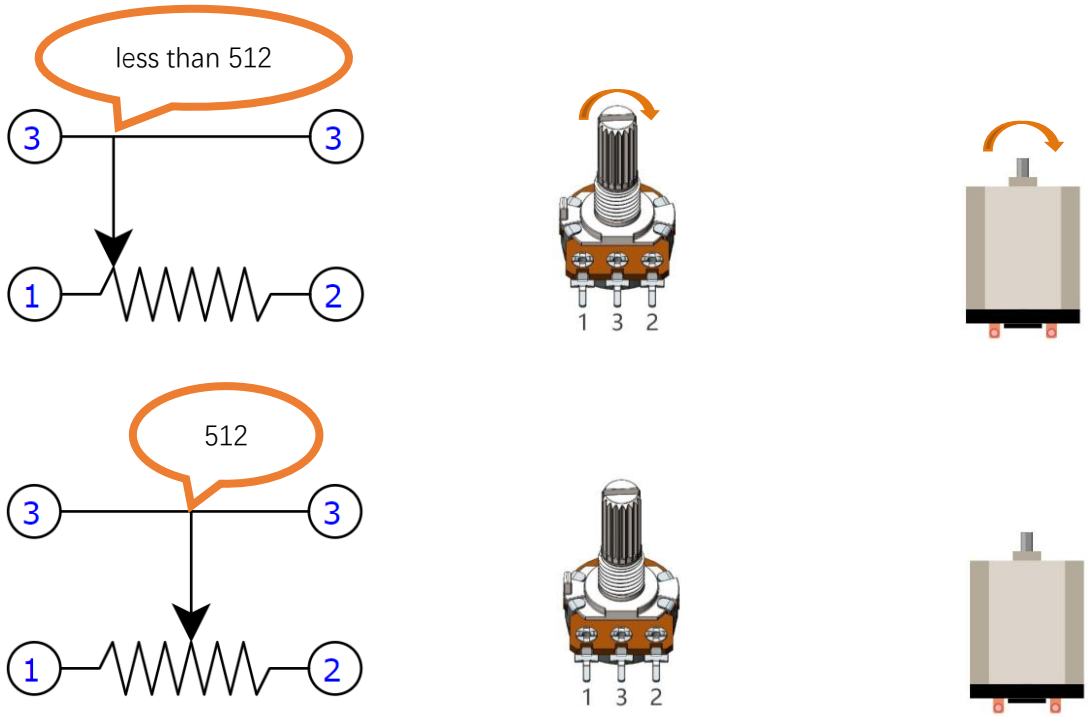


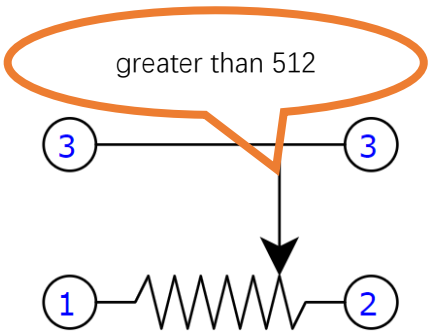
Sketch

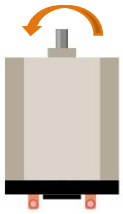
Sketch_16.1_Control_Motor_by_L293D

```
Sketch_18.1_Control_Motor_by_L293D | Arduino 1.8.18
File Edit Sketch Tools Help
Sketch_18.1_Control_Motor_by_L293D
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)
28     rotationDir = true;
29   else
30     rotationDir = false;
31   // Calculate the motor speed
32   rotationSpeed = abs(potenVal - 512);
33   //Control the steering and speed of the motor
34   driveMotor(rotationDir, constrain(rotationSpeed, 0, 512));
35 }
36
37 void driveMotor(boolean dir, int spd) {
Compiling sketch...
Compiling sketch...
"C:\\Users\\DESKTOP-LIN\\AppData\\Local\\Arduino15\\packages\\arduino\\tools\\arm-none-eabi-gcc\\7-2017q4/bin/arm-none-eabi-
29
Raspberry Pi Pico on COM10
```

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







The following is the sketch:

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

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

void setup() {
  // Initialize the pin into an output mode:
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
}

void loop() {
  int potenVal = analogRead(ADC0_PIN);      // Convert the voltage of rotary potentiometer
into digital
  //Compare the number with value 512,
  //if more than 512, clockwise rotates, otherwise, counter clockwise rotates
  rotationSpeed = potenVal - 512;
  if (potenVal > 512)
    rotationDir = true;
  else
    rotationDir = false;
  // Calculate the motor speed
  rotationSpeed = abs(potenVal - 512);
  //Control the steering and speed of the motor
  driveMotor(rotationDir, constrain(rotationSpeed, 0, 512));
}

void driveMotor(boolean dir, int spd) {
  // 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  }
```

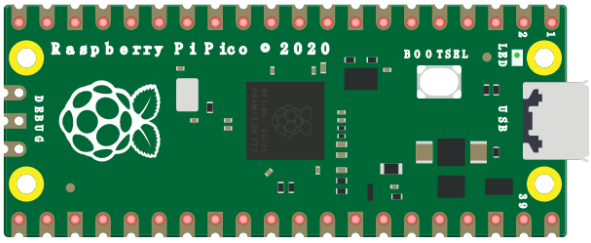
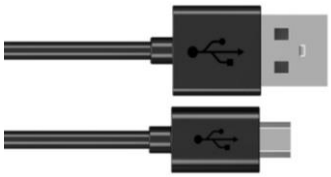
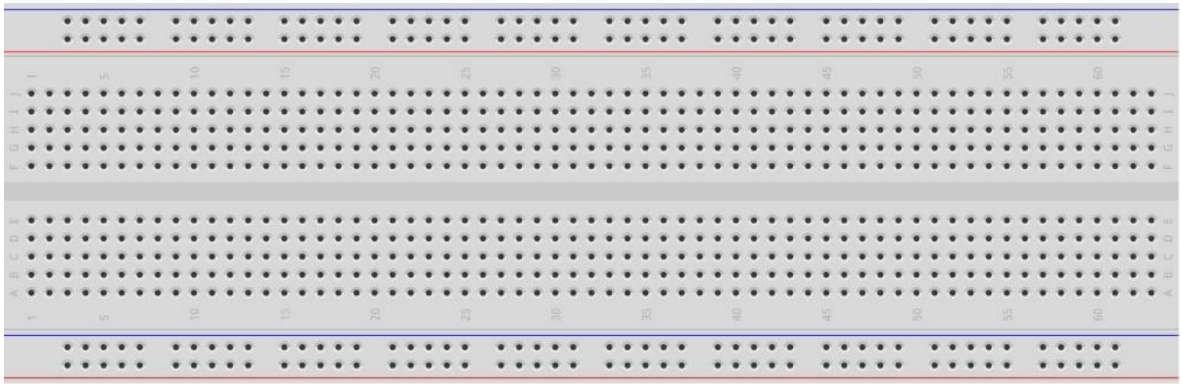
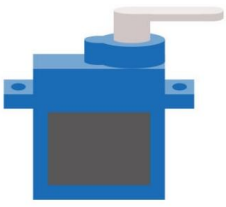
Chapter 17 Servo

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 17.1 Servo Sweep

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

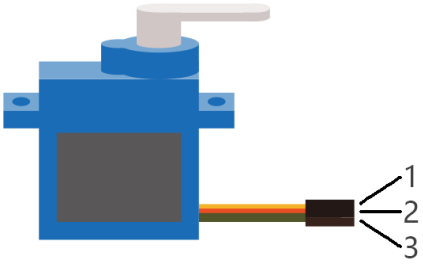
Component List

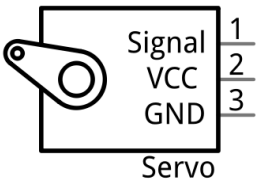
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Servo x1 	Jumper 

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:

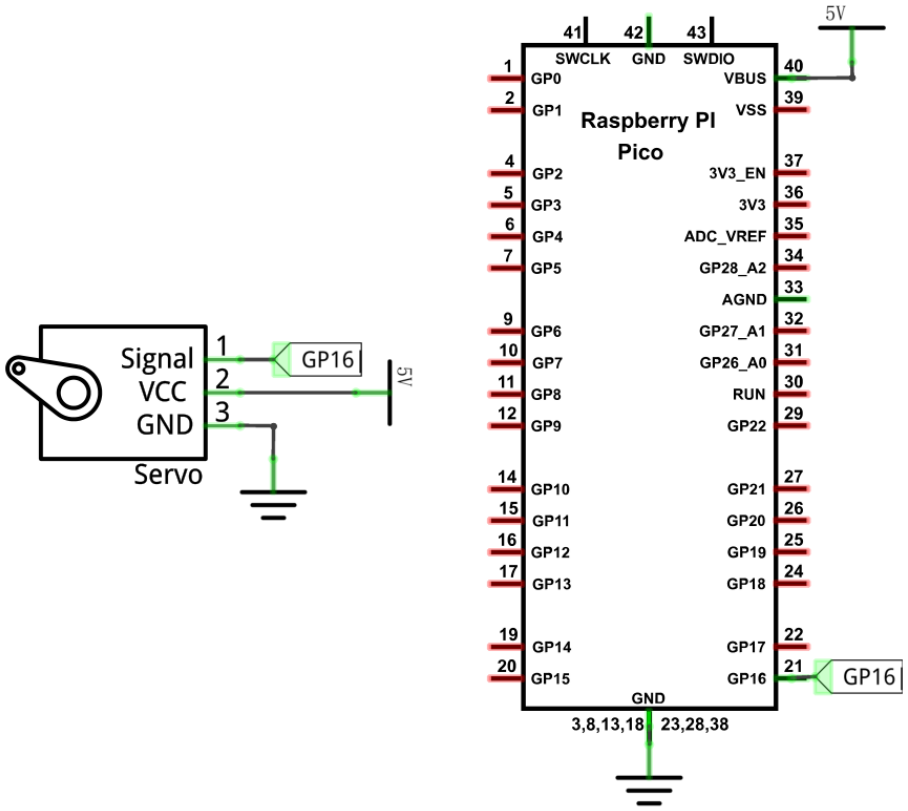
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

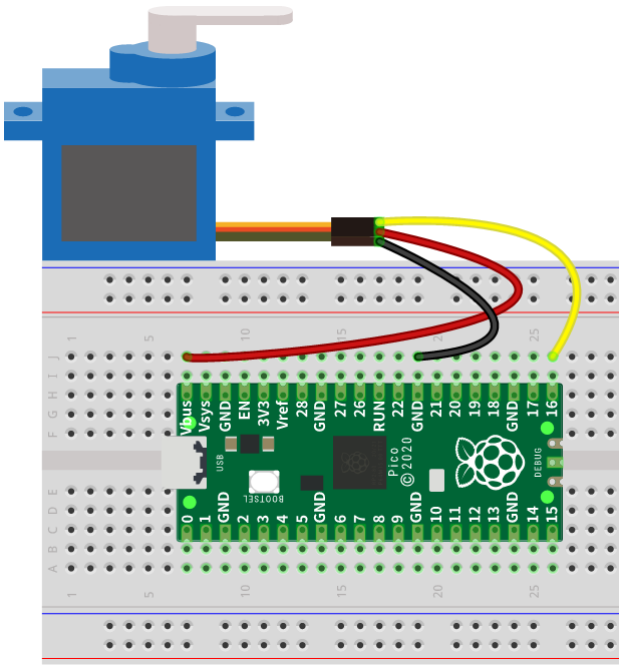
Circuit

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



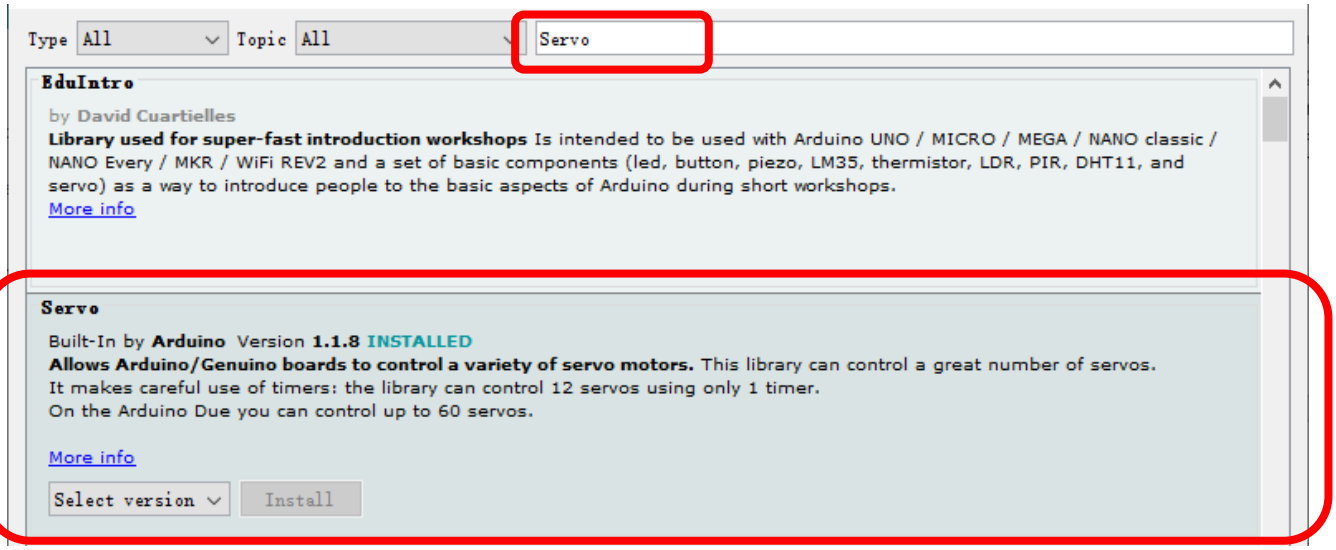
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

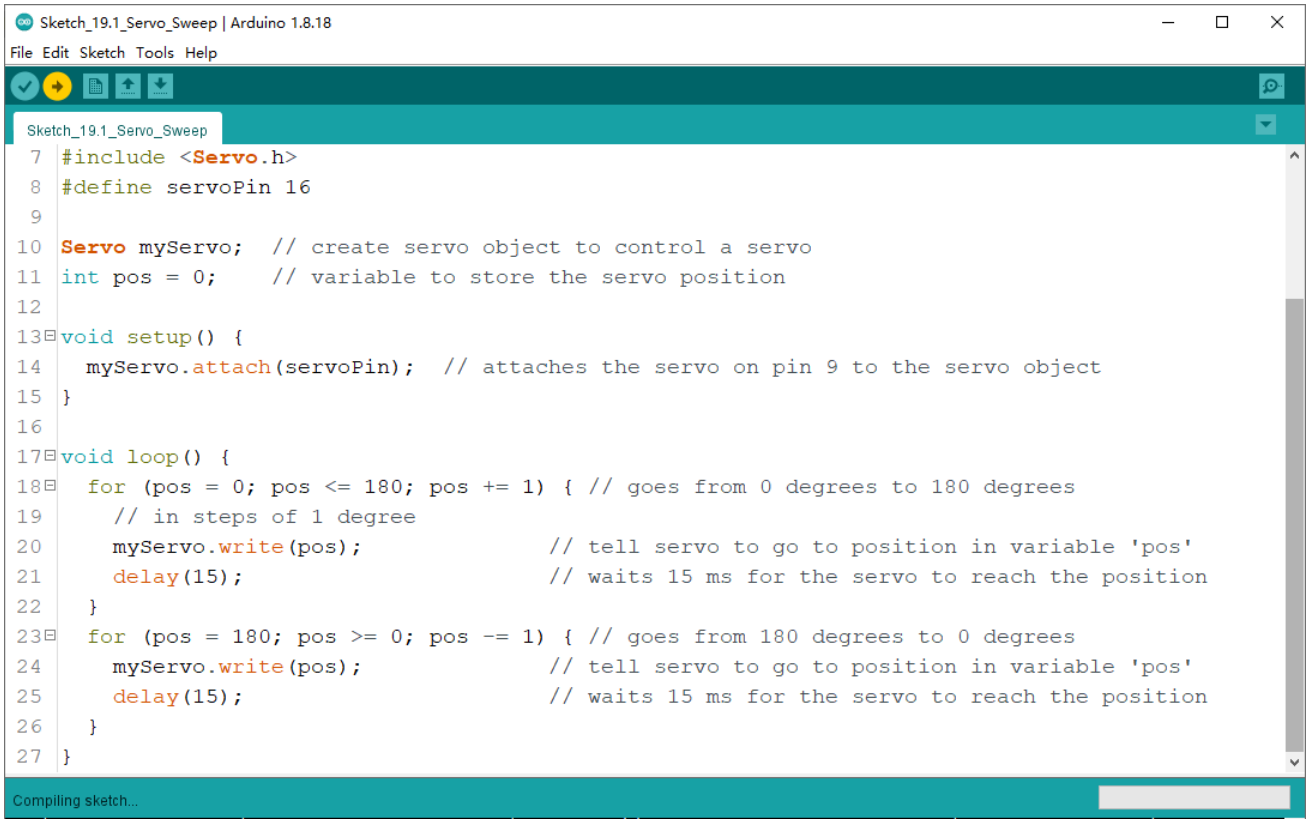
How to install the library

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

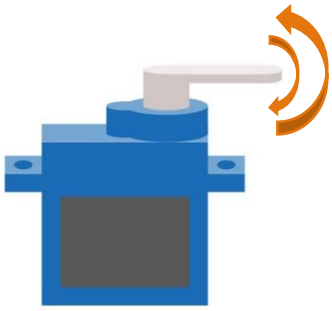


Use the Servo library to control the servo motor and let the servo motor rotate back and forth.

Sketch_17.1_Servo_Sweep



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:

```
#include <Servo.h>
#define servoPin 16

Servo myServo;  // create servo object to control a servo
int pos = 0;    // variable to store the servo position

void setup() {
  myServo.attach(servoPin);  // attaches the servo on pin 9 to the servo object
}

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

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

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

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

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

Set the control servo motor pin.

```
myServo.attach(servoPin);  // 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:

```
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:

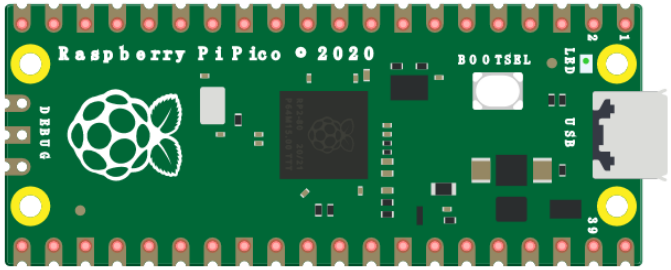
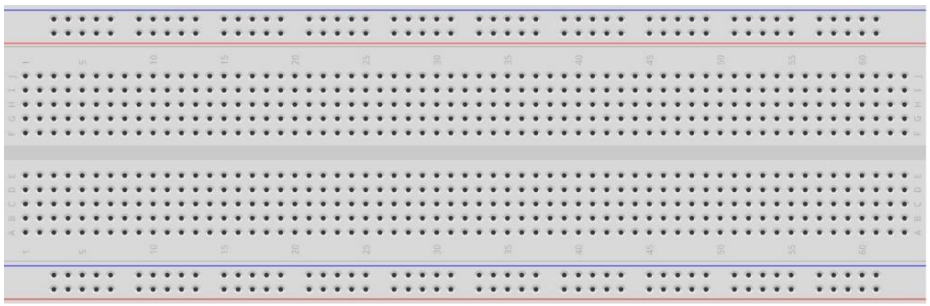
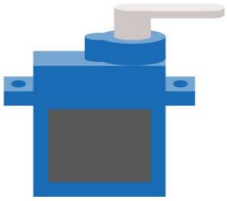
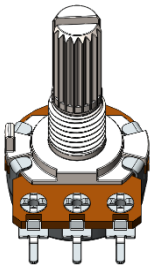
myservo.attach(pin): Initialize the servo, the parameter is the port connected to servo signal line;

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

Project 17.2 Servo Knob

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

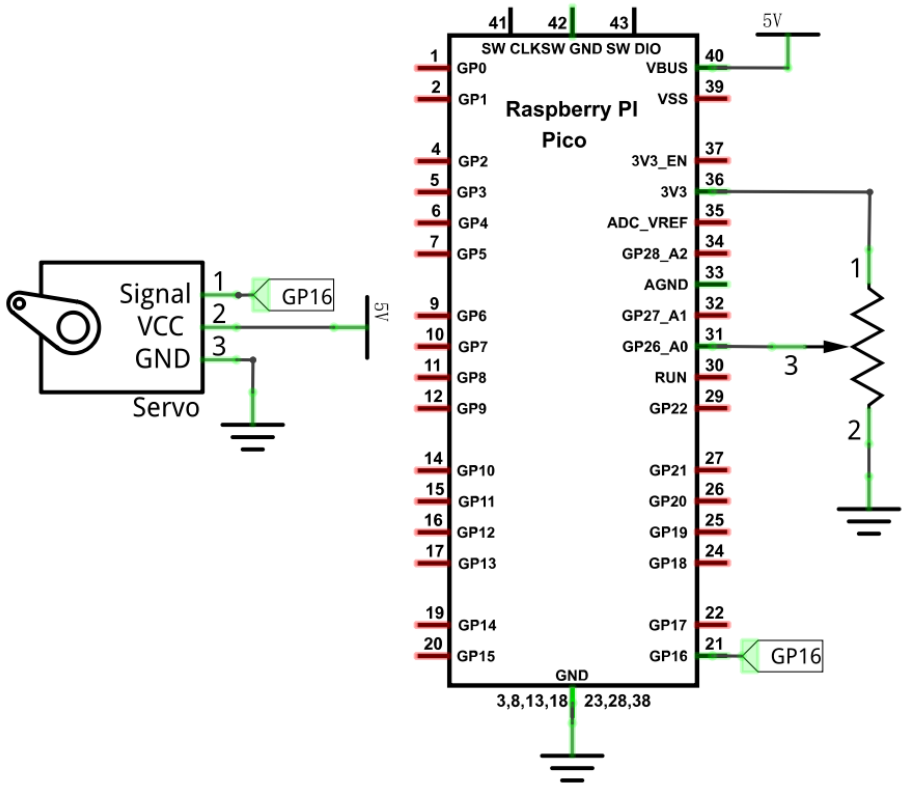
Component List

<table>
<tr><td>Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Servo x1
  </td><td>Jumper
  </td><td>Rotary potentiometer x1
  </td></tr>
</table>

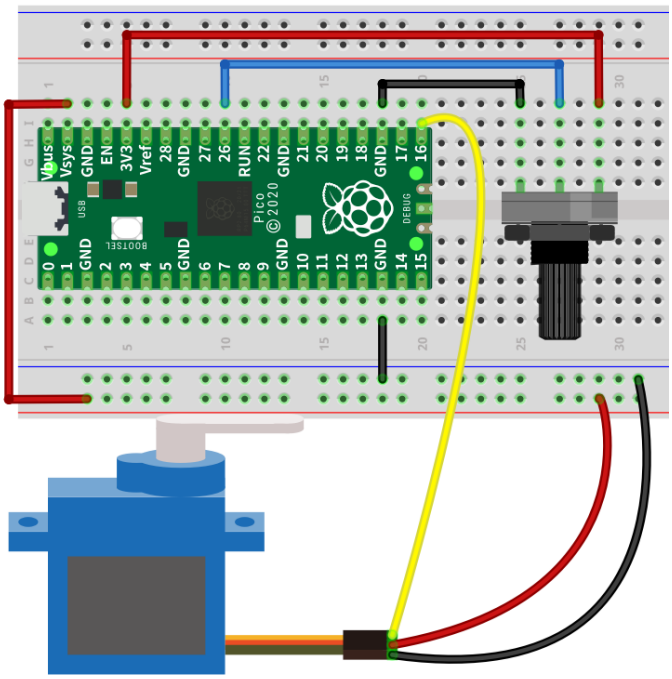
Circuit

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

Schematic diagram



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



Sketch

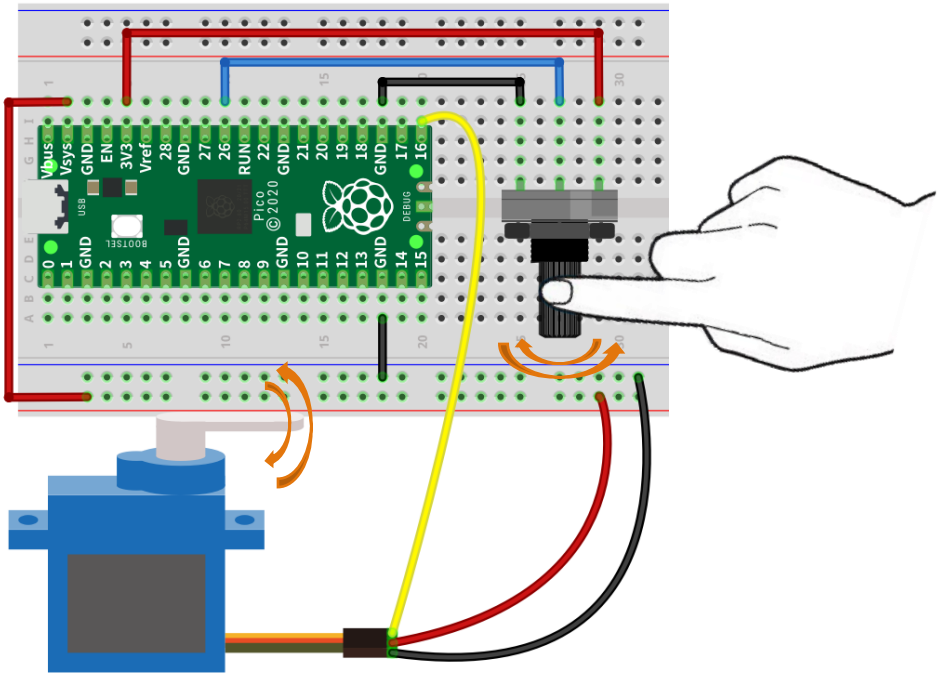
Sketch_17.2_Control_Servo_by_Potentiometer

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

```
#include <Servo.h>

#define servoPin 16          // define the pin of servo signal line
#define adcPin   26          // analog pin used to connect the potentiometer

Servo myservo;               // create servo object to control a servo
int potVal;                  // variable to read the potValue from the analog pin

void setup() {
  myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
}

void loop() {
  potVal = analogRead(adcPin);           // reads the potValue of the potentiometer
  potVal = map(potVal, 0, 1023, 0, 180);// scale it to use it with the servo
  myservo.write(potVal);                 // sets the servo position
  delay(15);                             // waits for the servo to get there
}
```

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, then the servo will rotate to a corresponding angle.



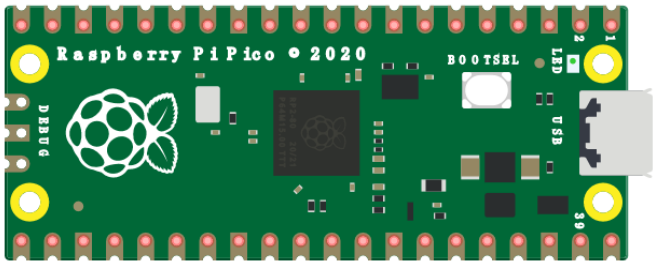
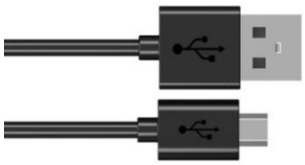
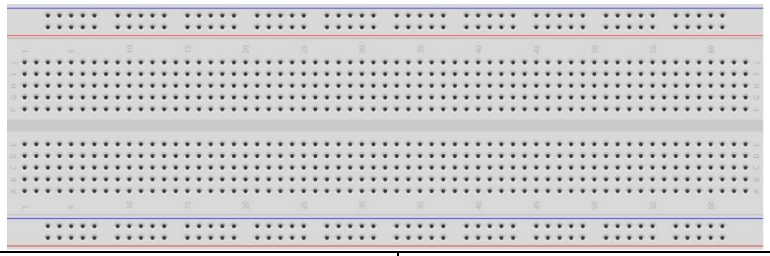
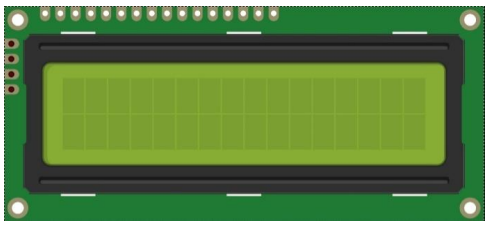
Chapter 18 LCD1602

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

Project 18.1 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 

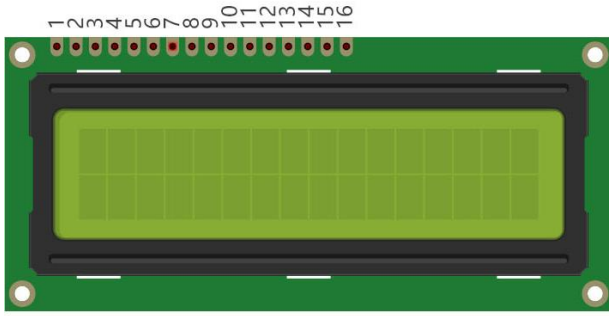
Component Knowledge

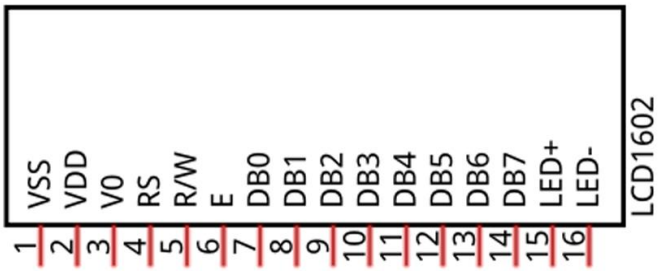
I2C communication

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.

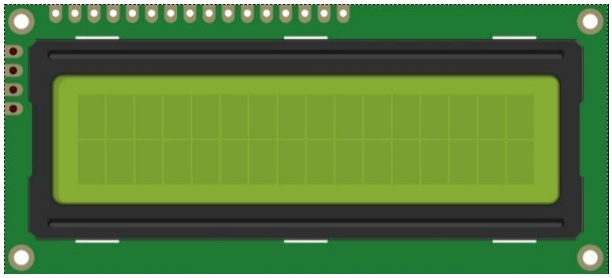
LCD1602 communication'

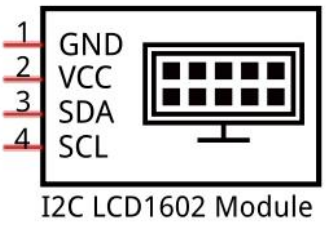
The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram.





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

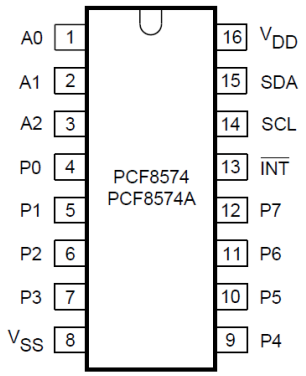




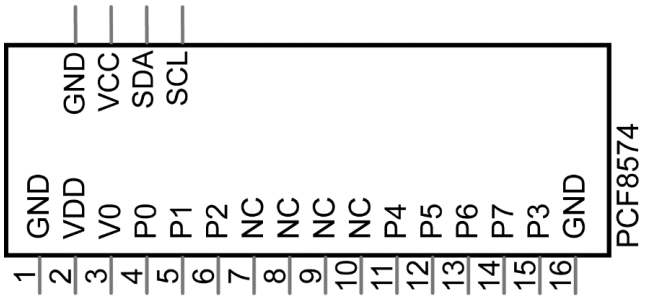
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

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

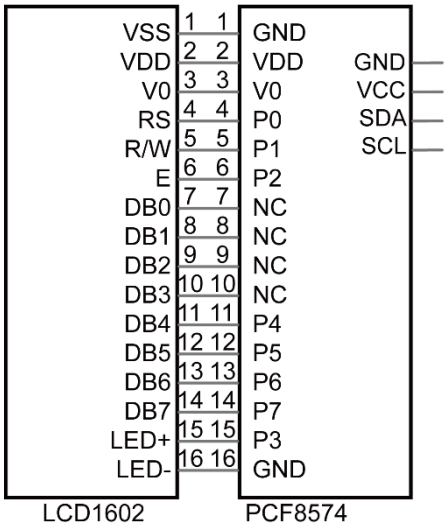
PCF8574 chip pin diagram:



PCF8574 module pin diagram



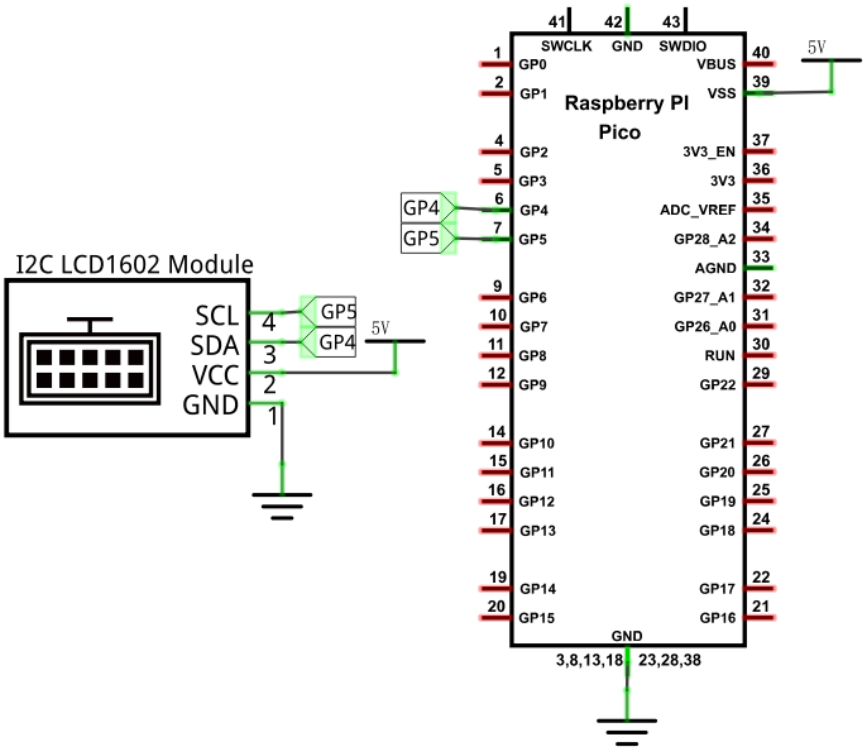
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



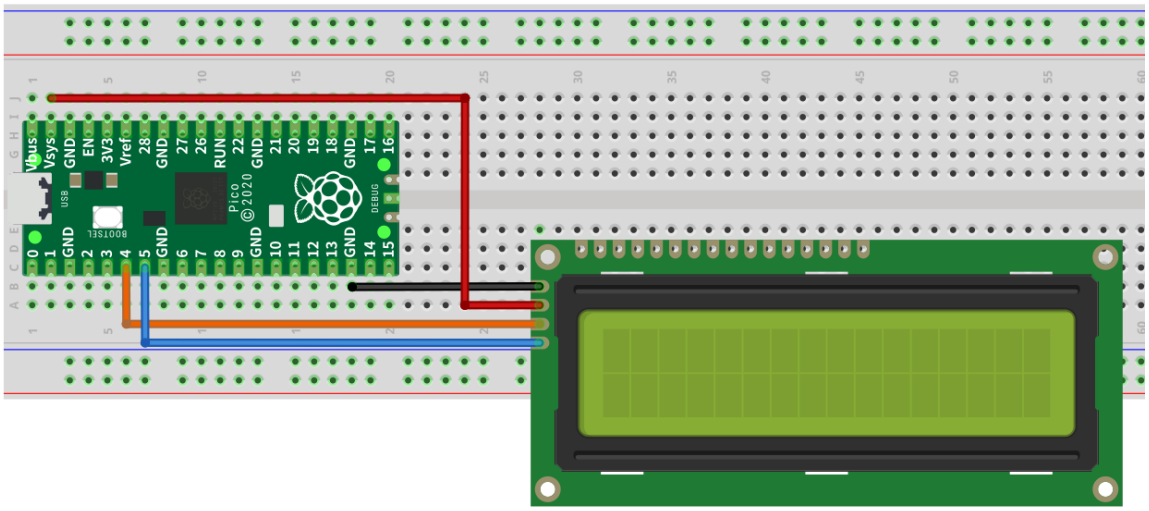
So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface. In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



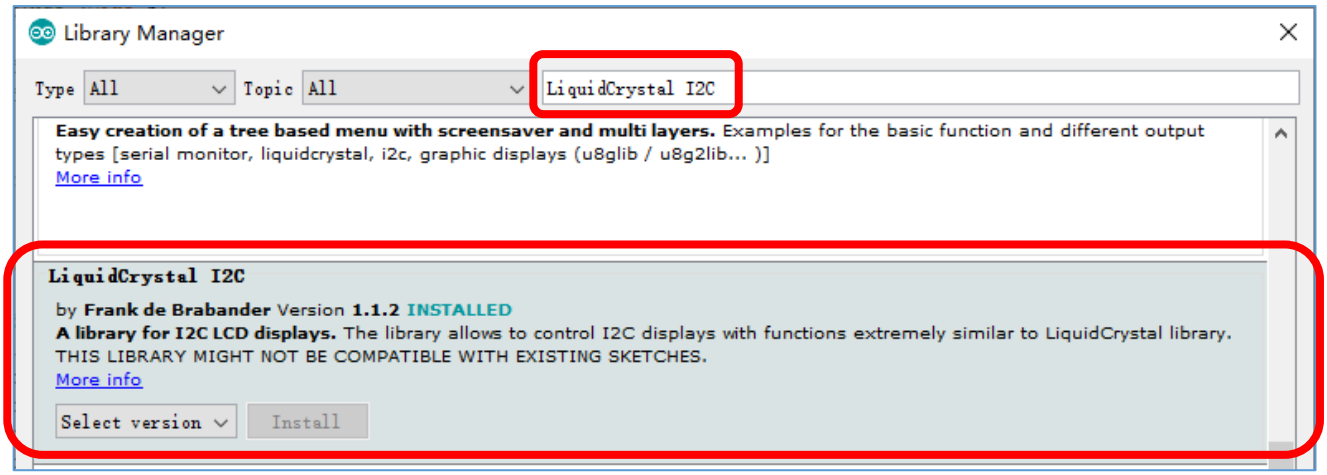
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

How to install the library

We use the third party library **LiquidCrystal I2C**. If you haven't installed it yet, please do so before learning. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter " LiquidCrystal I2C" in the search bar and select " LiquidCrystal I2C " for installation.



Use I2C LCD 1602 to display characters and variables.

Sketch_18.1_Display_the_string_on_LCD1602


```
#include <LiquidCrystal_I2C.h>

/*
 * note:If lcd1602 uses PCF8574T, IIC's address is 0x27,
 *      or lcd1602 uses PCF8574AT, IIC's address is 0x3F.
*/
LiquidCrystal_I2C lcd(0x27,16,2);

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

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

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.

The following is the program code:

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);

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

void loop() {
  lcd.setCursor(0,1);             // Move the cursor to row 1, column 0
```

```
17    lcd.print("Counter:");          // The count is displayed every second
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();
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

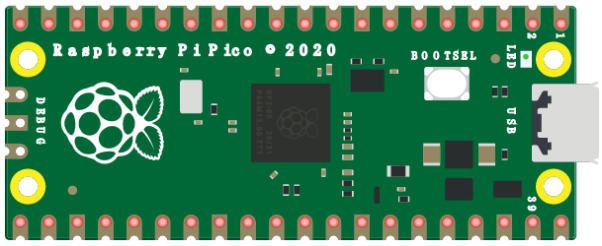
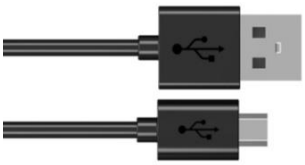
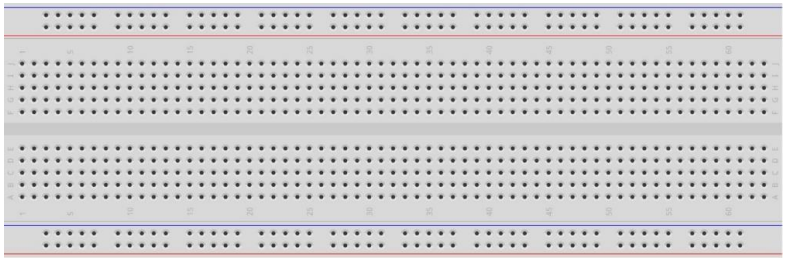
Chapter 19 Ultrasonic Ranging

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

Project 19.1 Ultrasonic Ranging

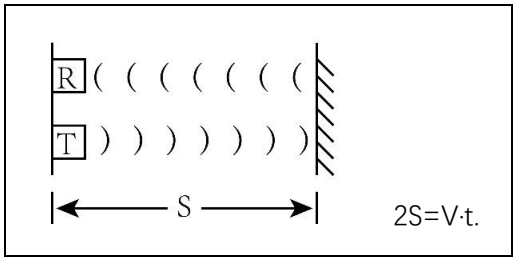
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

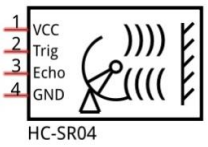
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	HC SR04 x1 

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=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



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



Pin description:

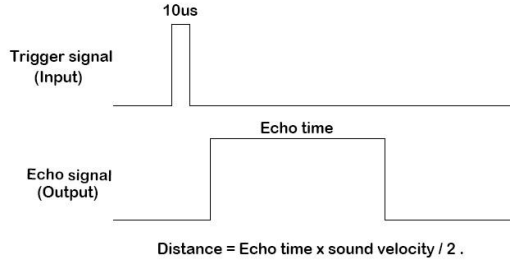
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

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

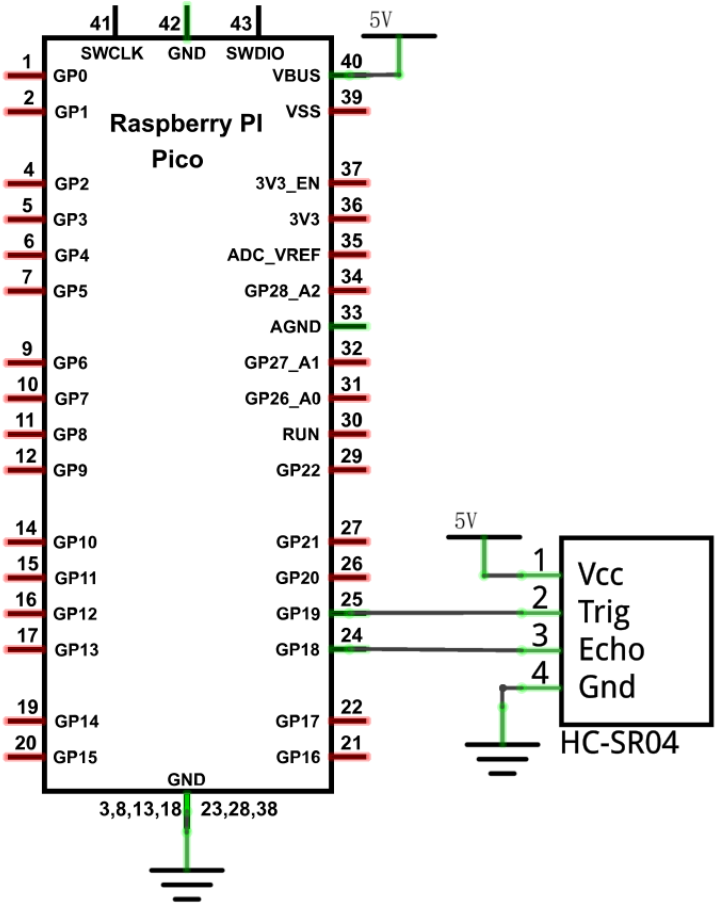
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The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



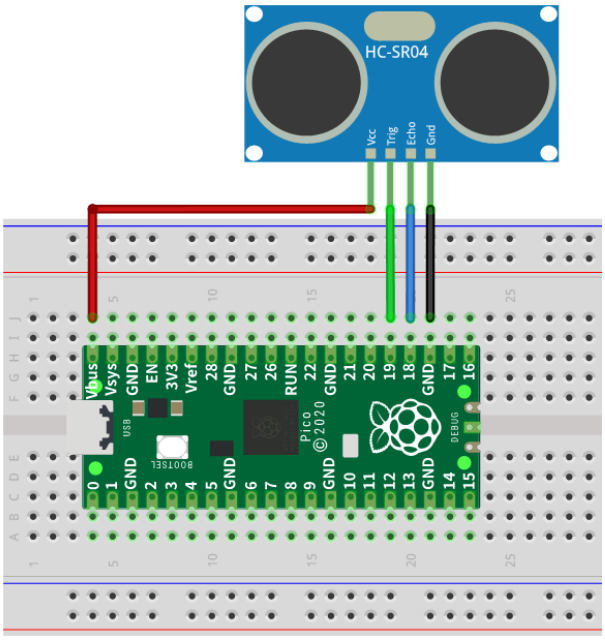
Circuit

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

Schematic diagram



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



Sketch

Sketch_19.1_Ultrasonic_Ranging

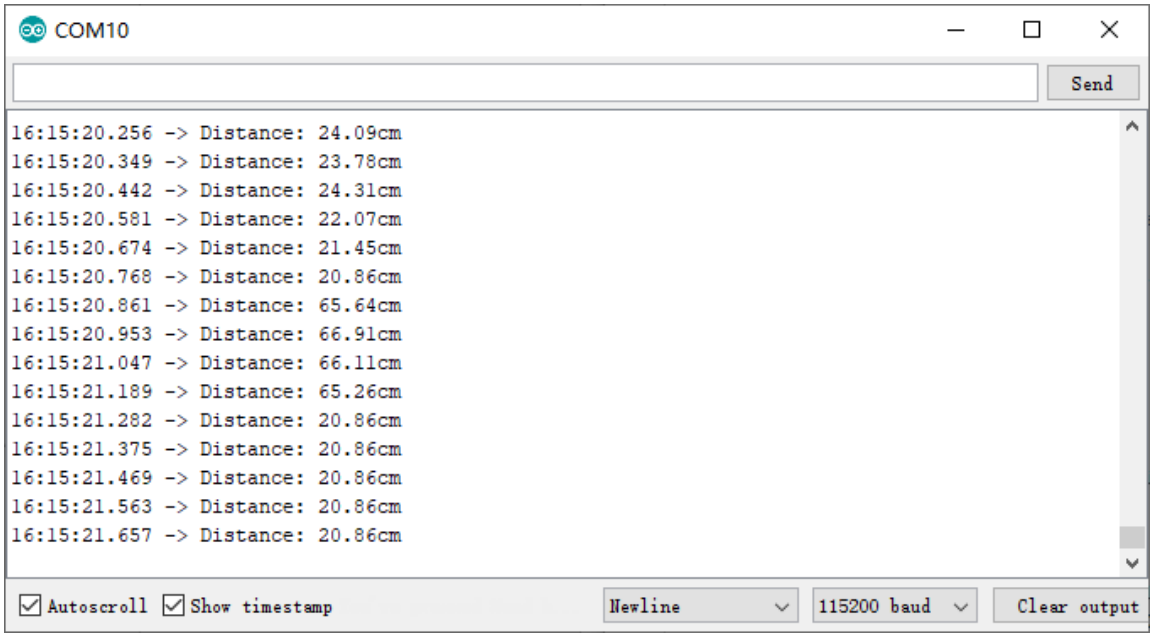
Sketch_22.1_Ultrasonic_Ranging | Arduino 1.8.18

File Edit Sketch Tools Help

Sketch_22.1_Ultrasonic_Ranging

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

Download the code to 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:

```
#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.printf("Distance: ");
  Serial.print(getSonar()); // Send ping, get distance in cm and print result
  Serial.println("cm");
}

float getSonar() {
  unsigned long pingTime;
  float distance;
  // make trigPin output high level lasting for 10us to trigger HC_SR04
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
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*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 triger 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 waitting 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 19.2 Ultrasonic Ranging

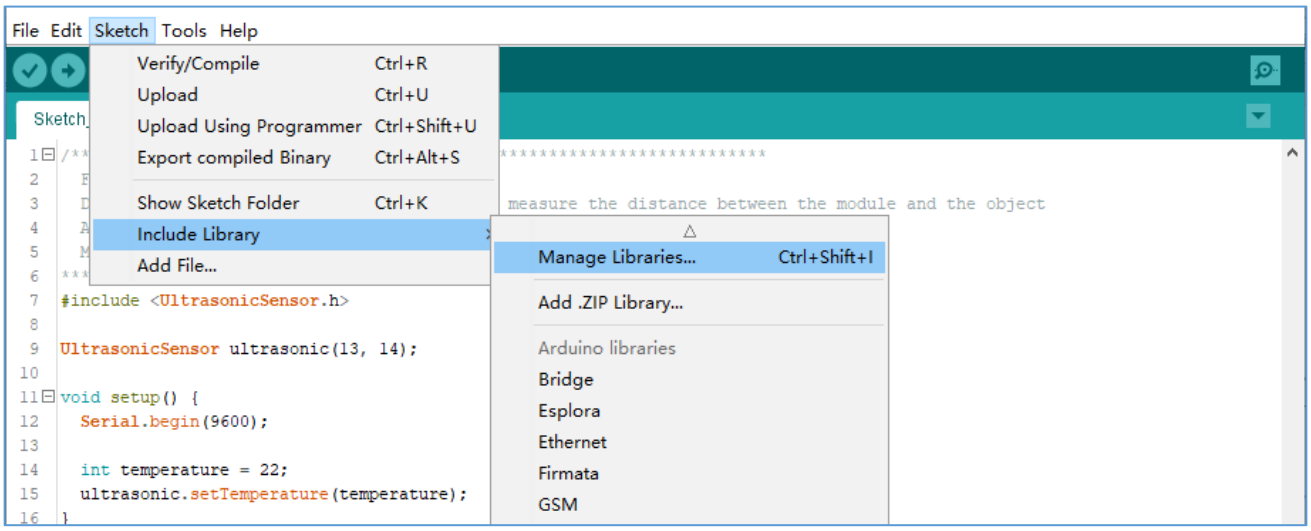
Component List and Circuit

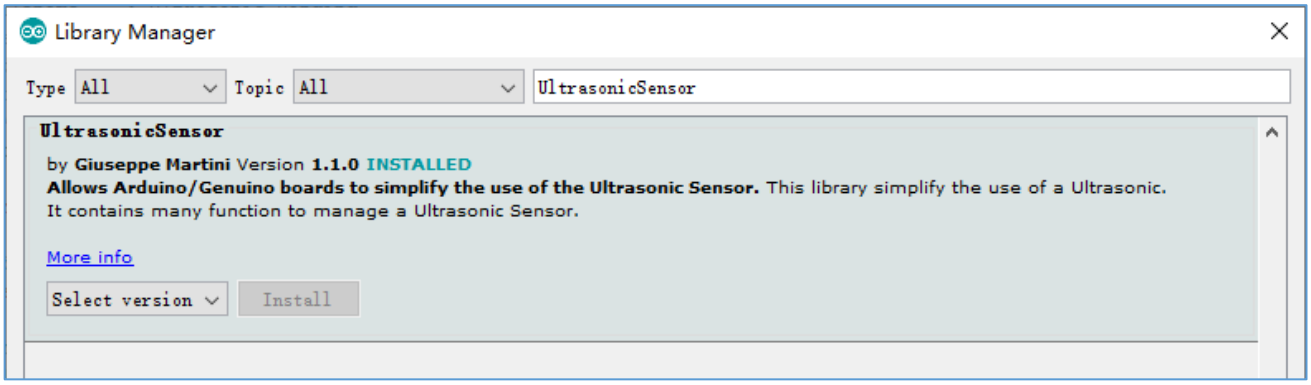
Component List and Circuit are the same as the previous section.

Sketch

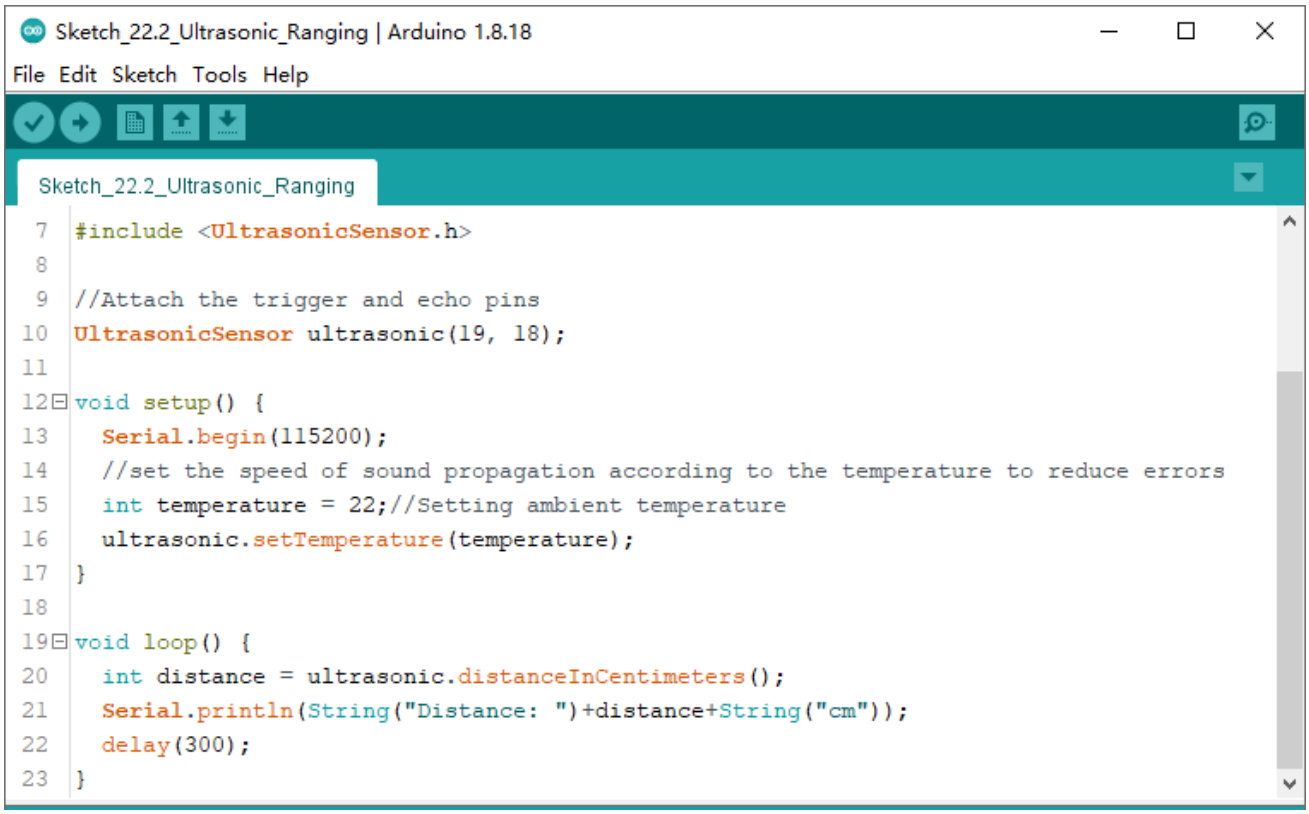
How to install the library

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

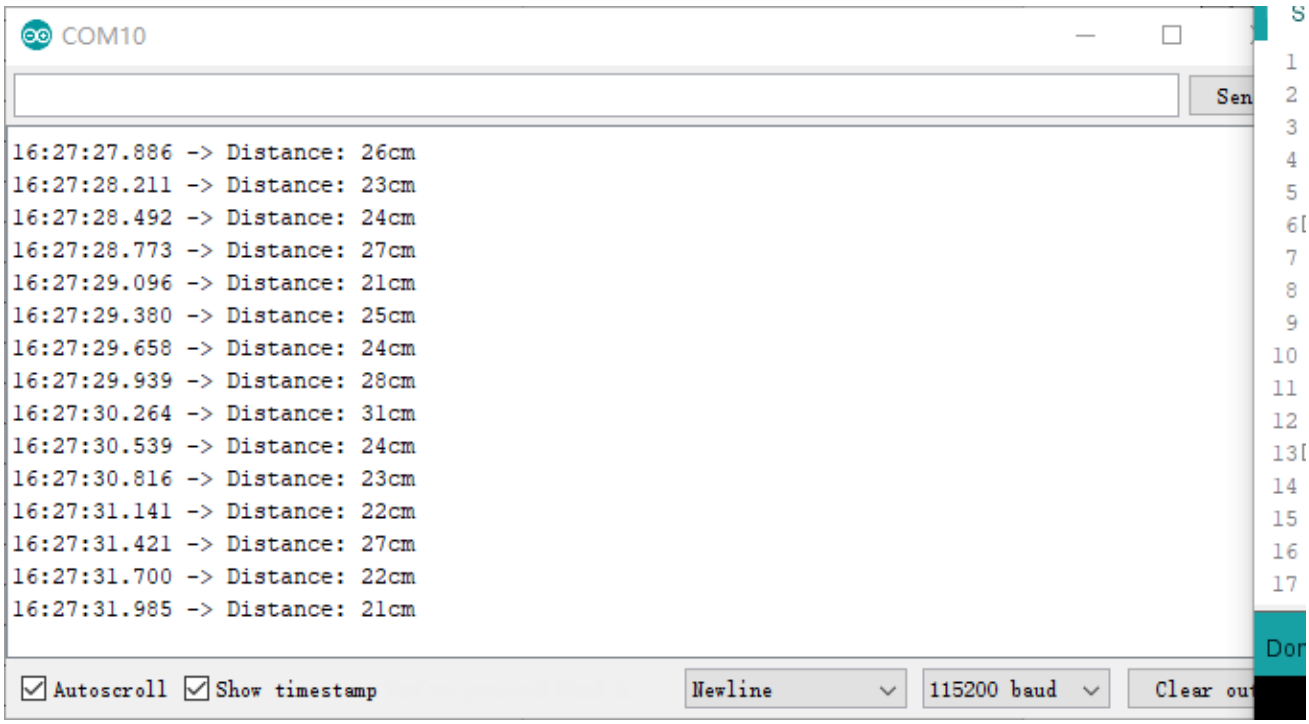




Sketch_19.2_Ultrasonic_Ranging



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:



The following is the program code:

```
#include <UltrasonicSensor.h>

//Attach the trigger and echo pins
UltrasonicSensor ultrasonic(19, 18);

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

void loop() {
  int distance = ultrasonic.distanceInCentimeters();
  Serial.print(String("Distance: ")+distance+String("cm\n"));
  delay(300);
}
```

First, add UltrasonicSensor library.

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

Define an ultrasonic object and associate it with the pins.

```
UltrasonicSensor ultrasonic(19, 18);
```

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

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

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

```
void loop() {
  int distance = ultrasonic.distanceInCentimeters();
  Serial.print(String("Distance: ")+distance+String("cm\n"));
  delay(300);
}
```

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.

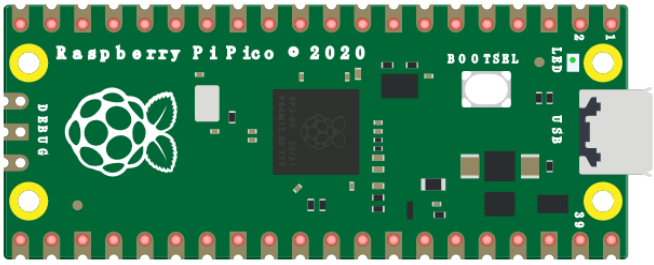
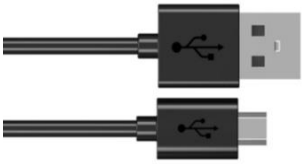
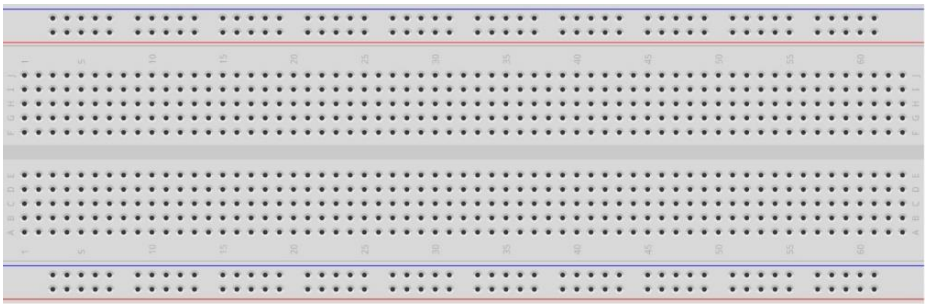
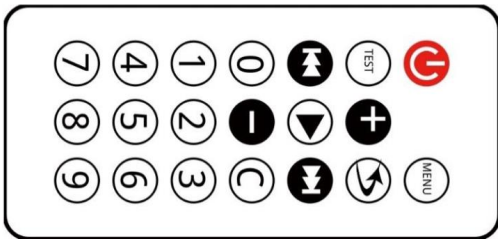
Chapter 20 Infrared Remote

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

Project 20.1 Infrared Remote Control

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

Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td colspan="2">Jumper
  </td><td rowspan="2">Infrared Remote x1
(May need CR2025 battery x1, please check the holder)
  </td></tr>
<tr><td>Infrared Remote x1
  </td><td>Resistor 10kΩ x1
  </td></tr>
</table>

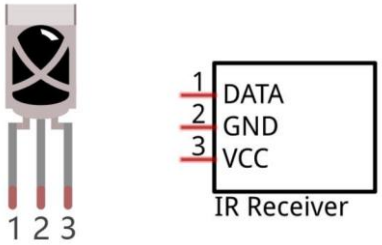
Component Knowledge

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.

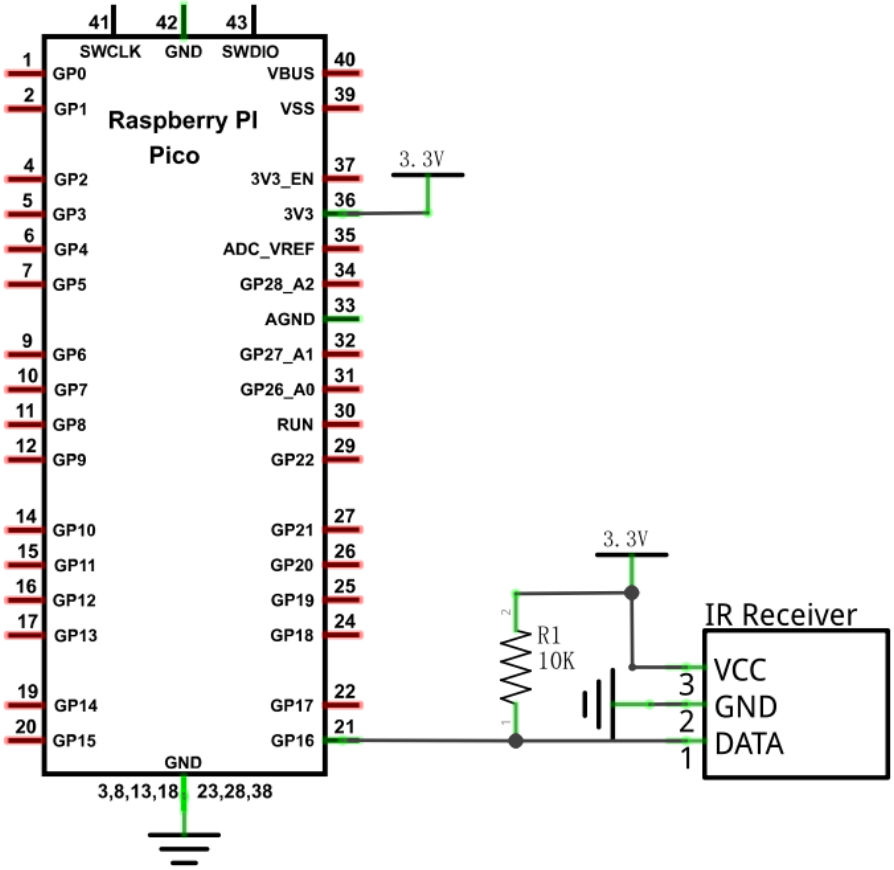


When you use the infrared remote control, it sends a key value to the receiving circuit according to the pressed key. We can program the Raspberry Pi Pico to do things like lighting, when a key value is received.
The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

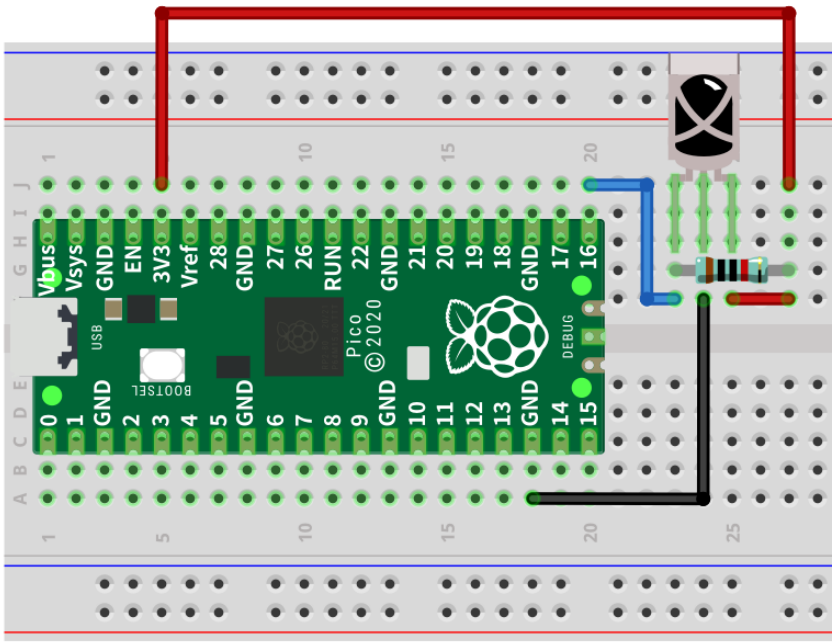
ICON	KEY Value	ICON	KEY Value
	FFA25D		FFB04F
	FFE21D		FF30CF
	FF22DD		FF18E7
	FF02FD		FF7A85
	FFC23D		FF10EF
	FFE01F		FF38C7
	FFA857		FF5AA5
	FF906F		FF42BD
	FF6897		FF4AB5
	FF9867		FF52AD

Circuit

Schematic diagram



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

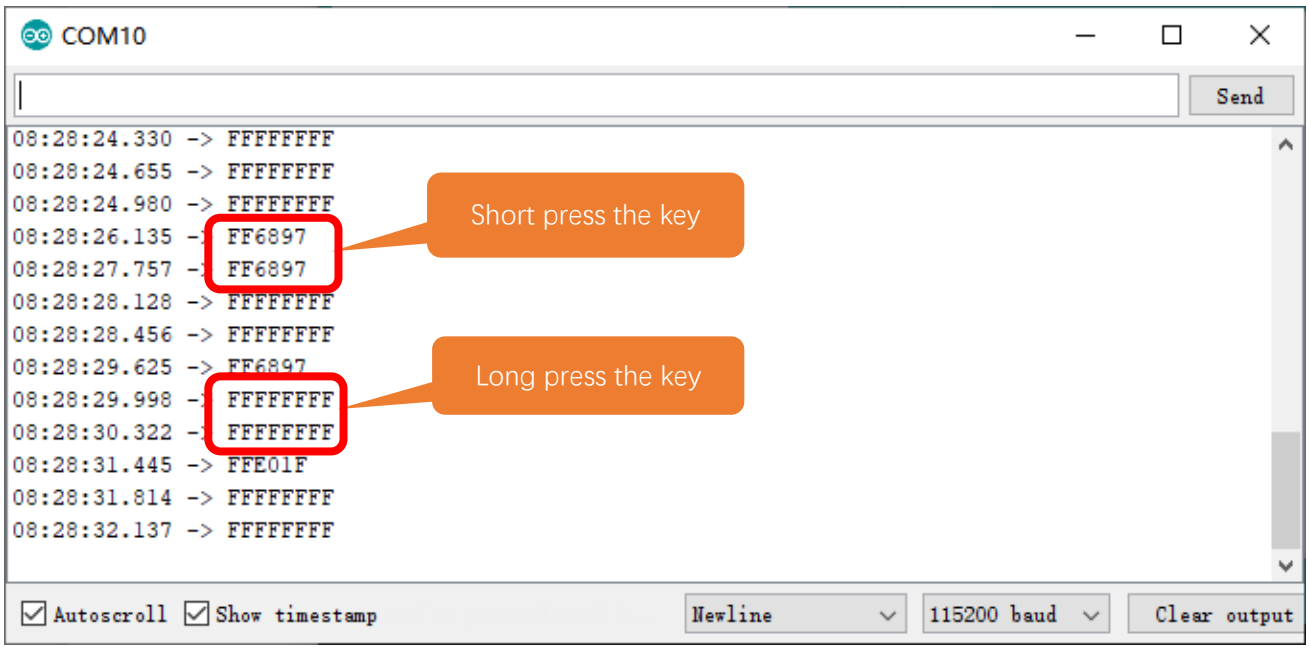


Sketch

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

Sketch_20.1_Infrared_Remote_Control

Download the code to Pico, open the serial port monitor, set the baud rate to 115200, press the IR remote control, the pressed keys value will be printed out through the serial port.



IR.cpp

```
#include "IR.h"

int logList[32];
unsigned long startTime;
int endTime, end2Time;
int flagCode = 0;
int irPin;
bool irState = true;

void IR_Init(int pin) {
  irPin = pin;
  pinMode(irPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(irPin), IR_Read, CHANGE);
}

void IR_Read() {
  if (irState == true) {
    unsigned long lowTime, highTime, intervalTime;
    int num = 0;
    while (digitalRead(irPin) == LOW) {
      startTime = micros();
      while (digitalRead(irPin) == LOW) {
        lowTime = micros();
      }
      intervalTime = lowTime - startTime;
      while (digitalRead(irPin) == HIGH) {
        highTime = micros();
        intervalTime = highTime - lowTime;
        if (intervalTime > 10000) {
          end2Time = millis();
          if (num == 32) {
            flagCode = 1;
            endTime = millis();
          }
          else if (num == 0 && end2Time - endTime > 300 && end2Time - endTime < 400) {
            flagCode = 2;
            endTime = millis();
          }
          return;
        }
      }
      if (intervalTime < 2000) {
        if (intervalTime < 700) {
```

```
            logList[num ++] = 0;
          }
          else {
            logList[num ++] = 1;
          }
        }
      }
    }

unsigned long IR_Decode(int &code) {
  unsigned long irData = 0;
  irState=false;
  if (code == 1) {
    code = 0;
    for (int i = 0; i < 32; i ++) {
      if (logList[i] == 0) {
        irData <<= 1;
      }
      else {
        irData <<= 1;
        irData ++;
      }
      logList[i] = 0;
    }

  }
  if (code == 2) {
    code = 0;
    irData = 0xffffffff;
  }
  return irData;
}

void IR_Release(){
  irState=true;
}
```

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.

```
extern int flagCode;
void IR_Init(int pin);
void IR_Read();
```

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 won't 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.

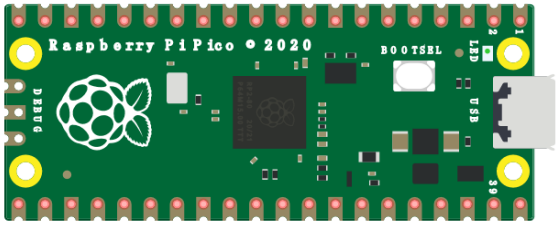
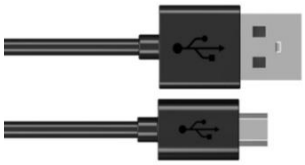
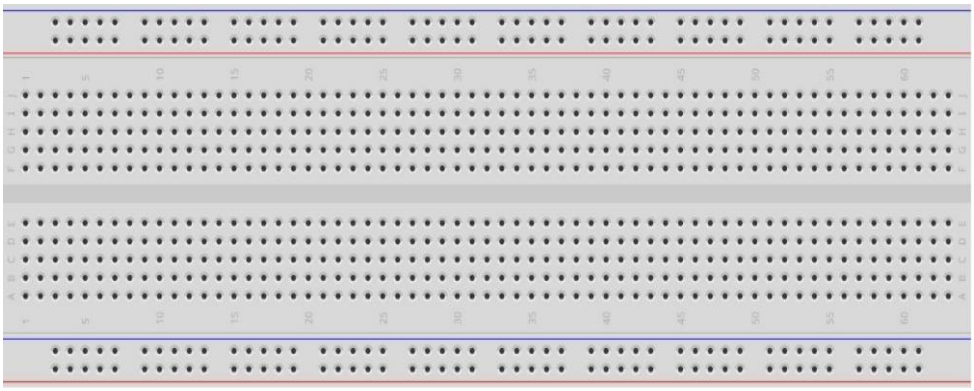
After using the infrared decoding function IR_Decode(), you need to call IR_Release() to release the infrared data receiving function. Otherwise, it won't receiver new infrared data again.

```
10   if(flagCode){
11     int irValue = IR_Decode(flagCode);
12     Serial.println(irValue, HEX);
13     IR_Release();
14   }
```

Project 20.2 Control LED through Infrared Remote

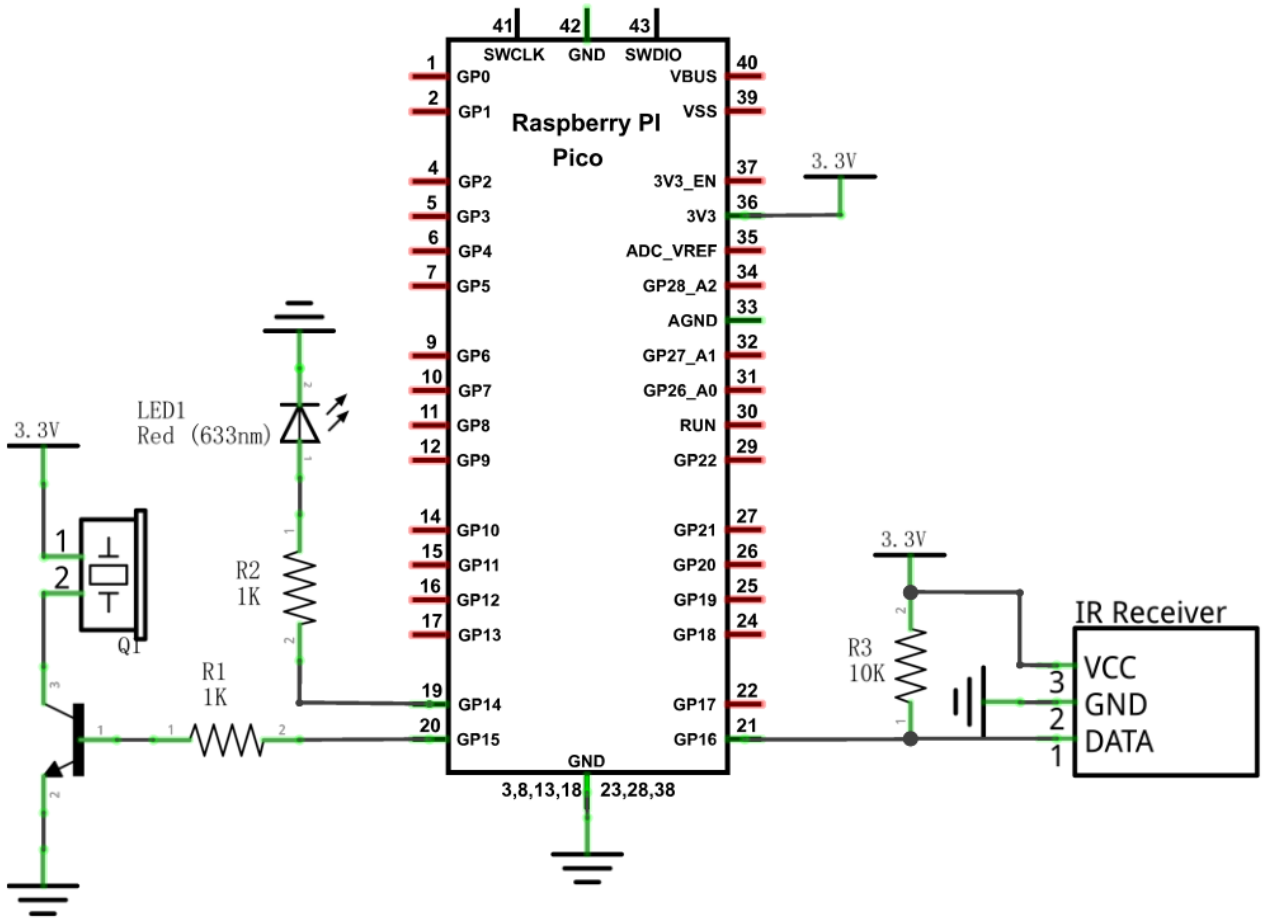
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

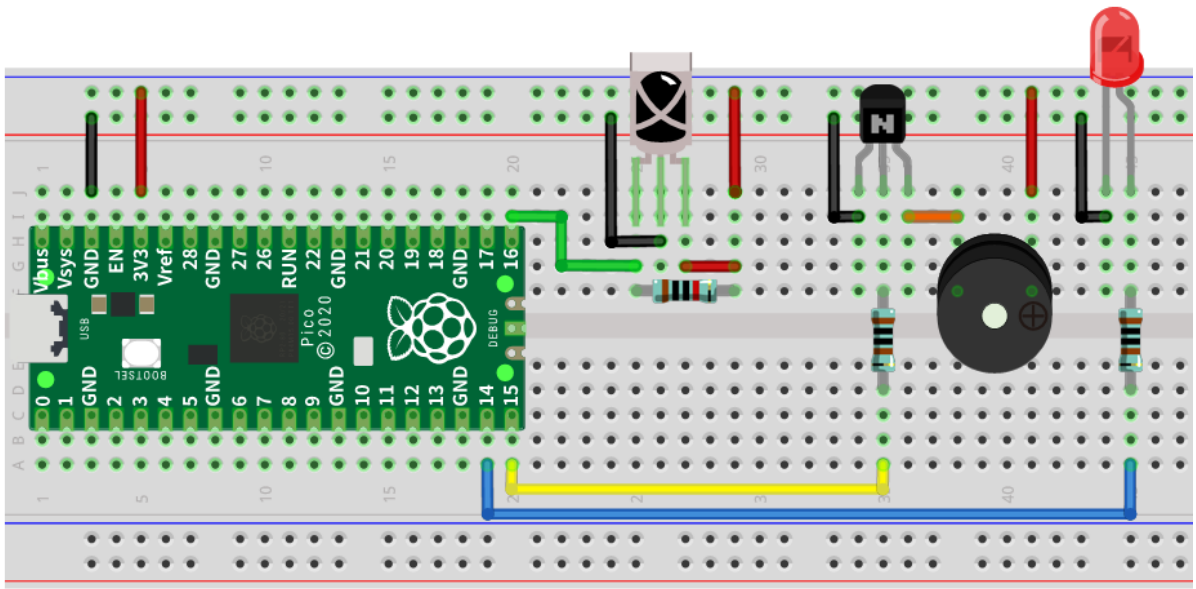
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td colspan="2">Jumper
  </td><td rowspan="3">Infrared Remote x1
(May need CR2025 battery x1,
please check the battery holder)
  </td></tr>
<tr><td>LED x1
  </td><td>Active buzzer x1
  </td><td>Resistor 1kΩ x2
  </td></tr>
<tr><td>Infrared receiver x1
  </td><td>NPN transistorx1
(S8050)
  </td><td>Resistor 10kΩ x1
  </td></tr>
</table>

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_20.2_Control_LED_through_Infrared_Remote

Sketch_24.2_Control_LED_through_Infrared_Remote | Arduino 1.8.18

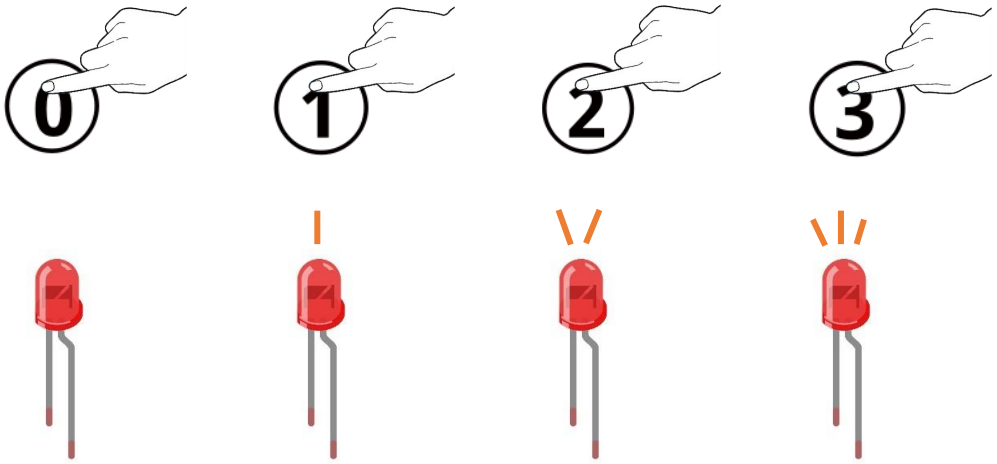
File Edit Sketch Tools Help

Sketch_24.2_Control_LED_through_Infrared_Remote | IR.cpp | IR.h

```
void setup() {
  Serial.begin(115200);
  IR_Init(irPin);
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  if(flagCode){
    int irValue = IR_Decode(flagCode);
    Serial.println(irValue, HEX);
    handleControl(irValue);
    IR_Release();
  }
}

void handleControl(unsigned long value) {
  digitalWrite(buzzerPin, HIGH);
  delay(100);
  digitalWrite(buzzerPin, LOW);
  // Handle the commands
  switch (value) {
    case 0xFF6897:              // Receive the number '0'
      analogWrite(ledPin, 0);   // Turn off LED
```

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:

```
#include "IR.h"

#define irPin 16
#define ledPin 14
#define buzzerPin 15

void setup() {
  Serial.begin(115200);
  IR_Init(irPin);
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  if(flagCode){
    int irValue = IR_Decode(flagCode);
    Serial.println(irValue, HEX);
    handleControl(irValue);
    IR_Release();
  }
}

void handleControl(unsigned long value) {
  digitalWrite(buzzerPin, HIGH);
  delay(100);
  digitalWrite(buzzerPin, LOW);
  // Handle the commands
  switch (value) {
    case 0xFF6897:              // Receive the number '0'
      analogWrite(ledPin, 0);   // Turn off LED
      break;
    case 0xFF30CF:              // Receive the number '1'
      analogWrite(ledPin, 50);  // Dimmest brightness
      break;
    case 0xFF18E7:              // Receive the number '2'
      analogWrite(ledPin, 100); // Medium brightness
      break;
    case 0xFF7A85:              // Receive the number '3'
      analogWrite(ledPin, 255); // Strongest brightnss
      break;
  }
}
```

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

```
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 brightnss
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  }
```

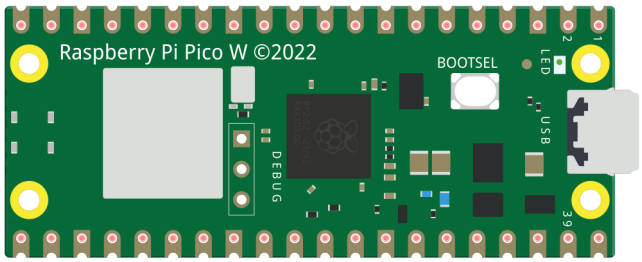
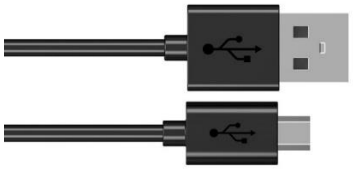
Chapter 21 WiFi Working Modes (Only for Pico W)

The biggest difference between the raspberry pi ico and the raspberry pie 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 21.1 Station mode

Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	

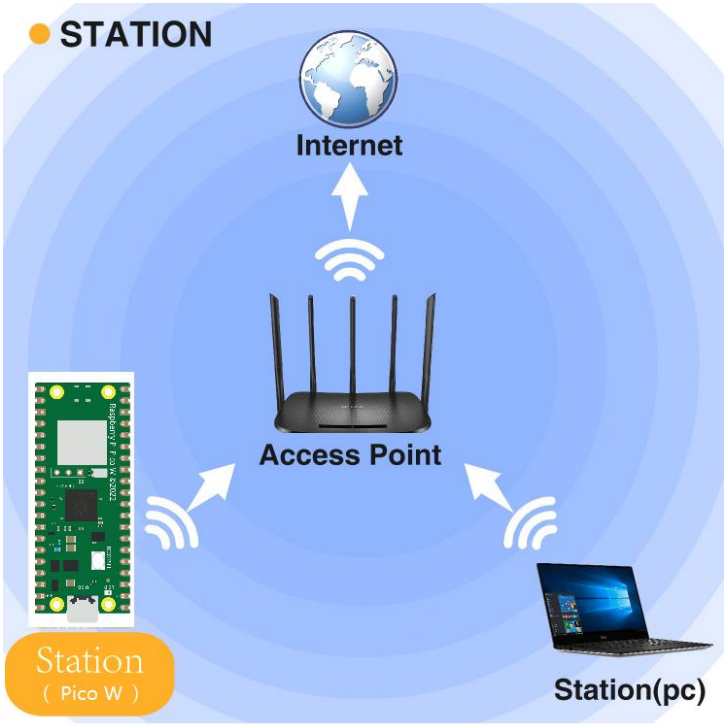
Component knowledge

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.

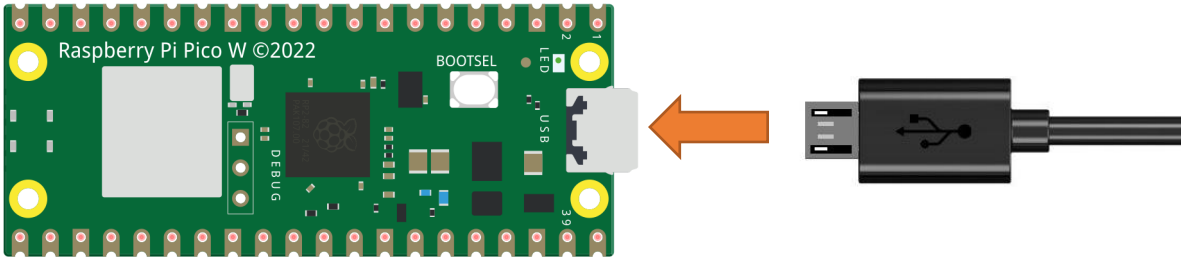
Station mode

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.



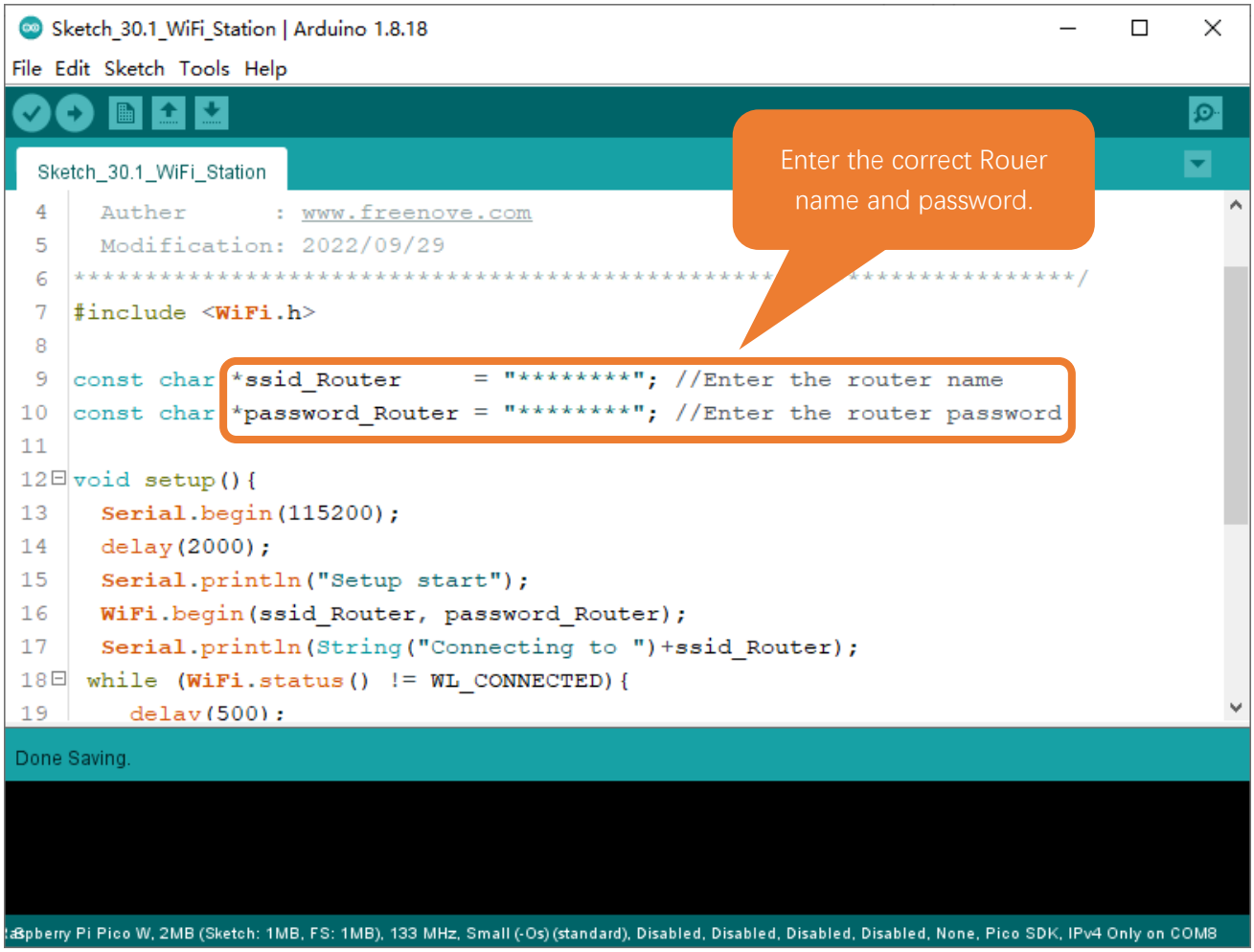
Circuit

Connect Pico W to the computer using the USB cable.



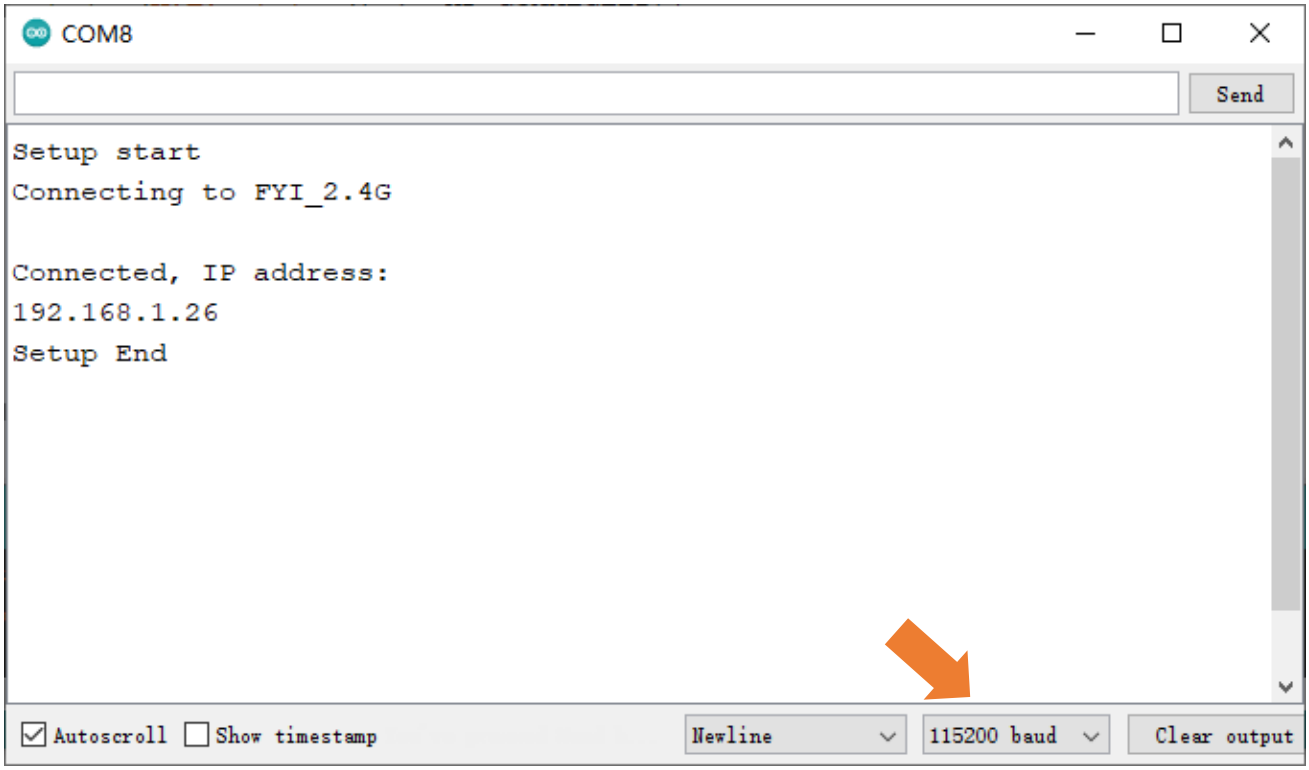
Sketch

Sketch_21.1_Station_mode



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. And 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:

```
#include <WiFi.h>

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

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

Include the WiFi Library header file of 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: blloean 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(): obtian IP address in Station mode

disconnect(): disconnect wifi

Project 21.2 AP mode

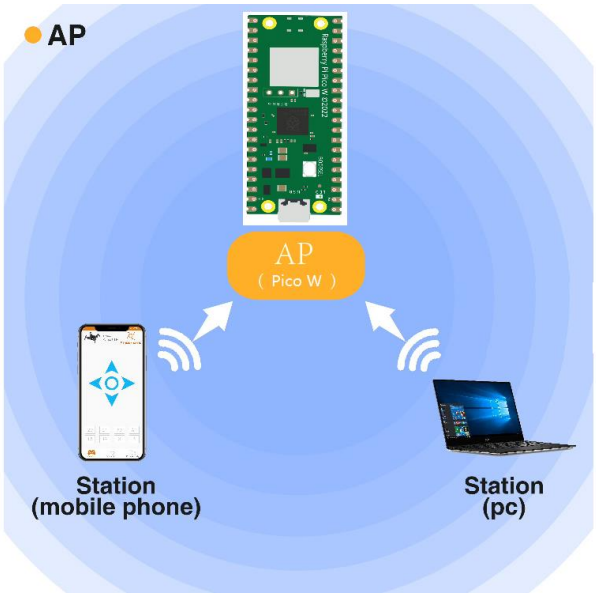
Component List & Circuit

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

Component knowledge

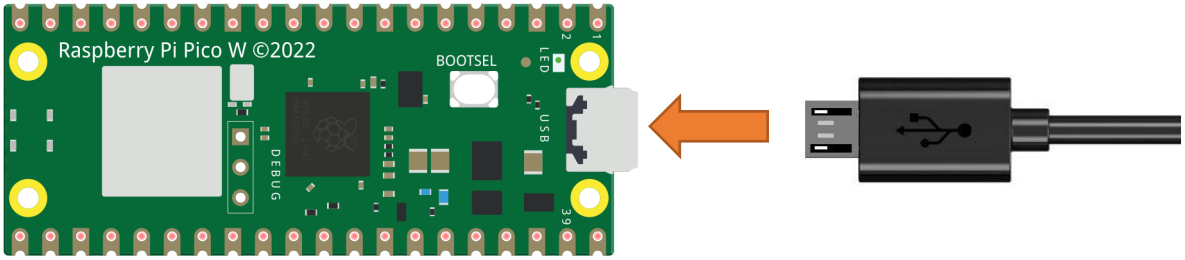
AP mode

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.



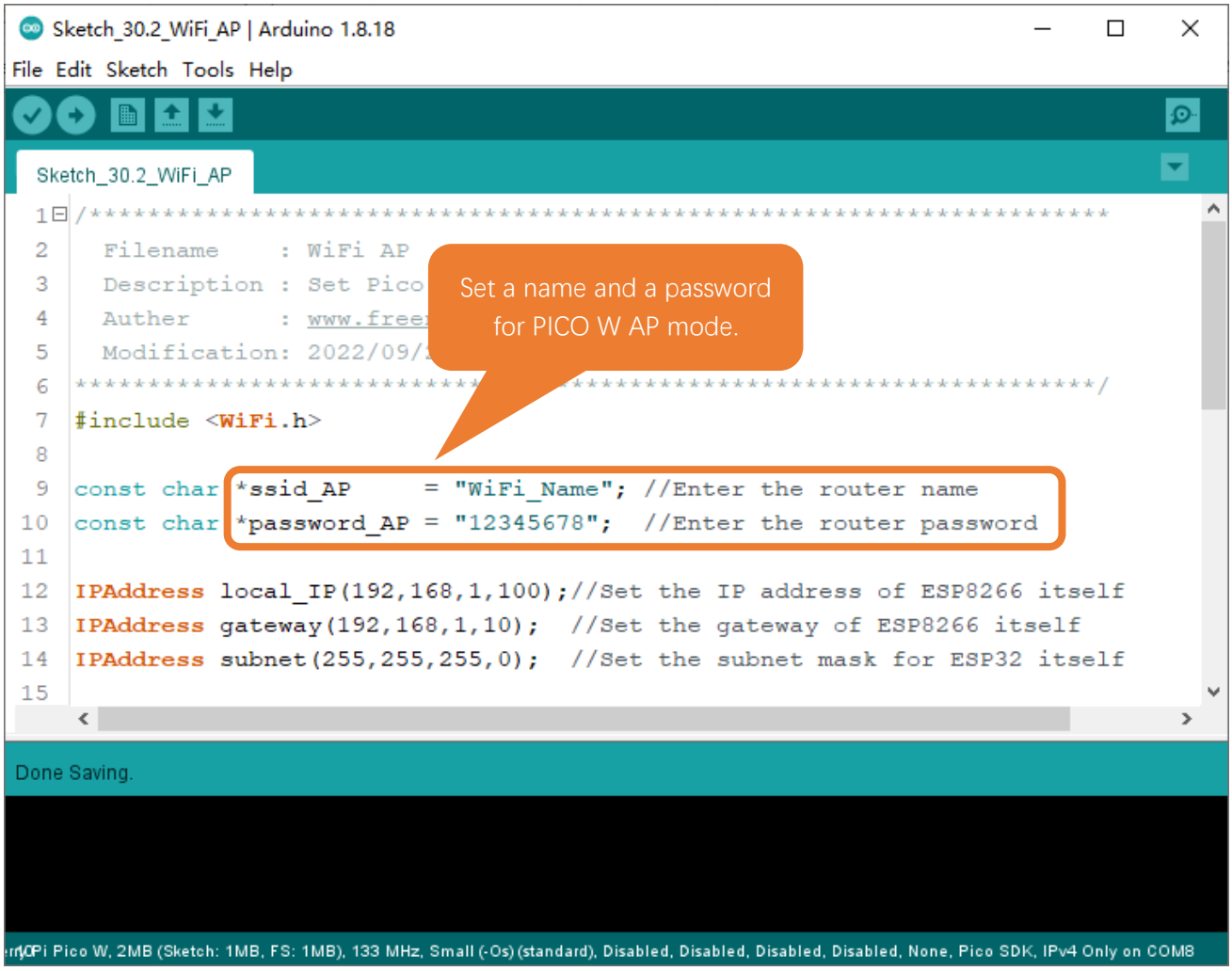
Circuit

Connect Pico W to the computer using the USB cable.



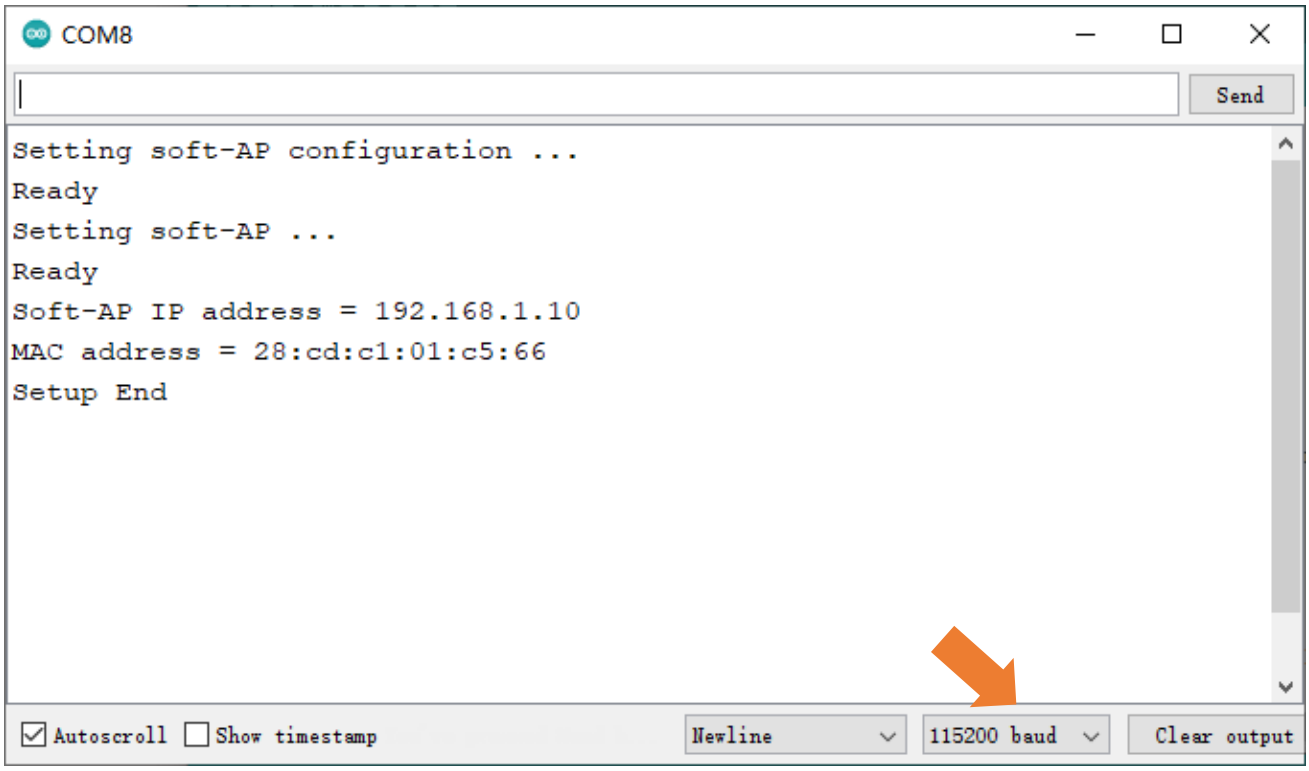
Sketch

Sketch_21.2_AP_mode

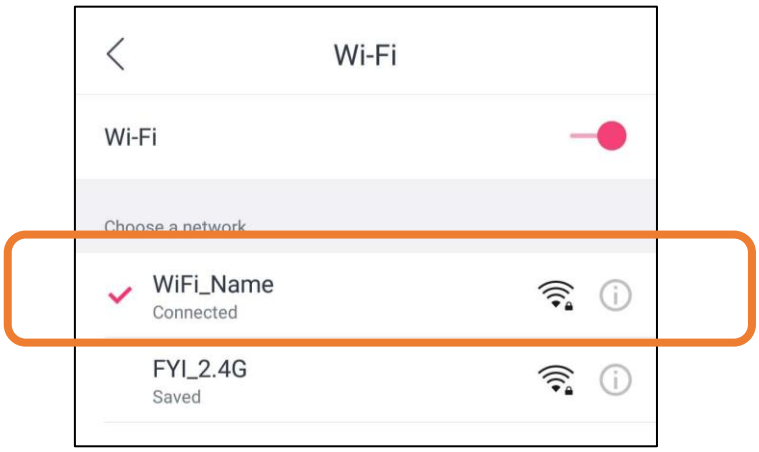


Before the Sketch runs, you can make any changes to the AP name and password for PICO W in the box as shown in the illustration above. Of course, you can leave it alone by default.

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



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

```
#include <WiFi.h>

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

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

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

void loop() {
}
```

Include WiFi Library header file of PICO W.

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

Enter correct AP name and password.

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

Set PICO W in AP mode.

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

Configure IP address, gateway and subnet mask for PICO W.

```
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.

```
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(): obtian IP address in AP mode

softAPdisconnect (): disconnect AP mode.

Project 21.3 AP+Station mode

Component List & Circuit

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

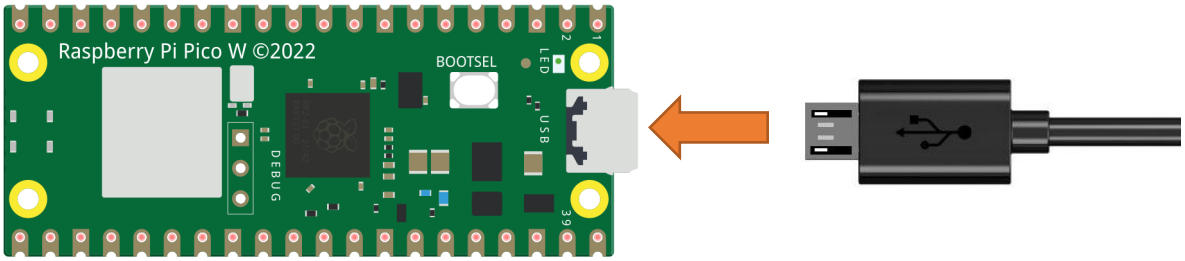
Component knowledge

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.

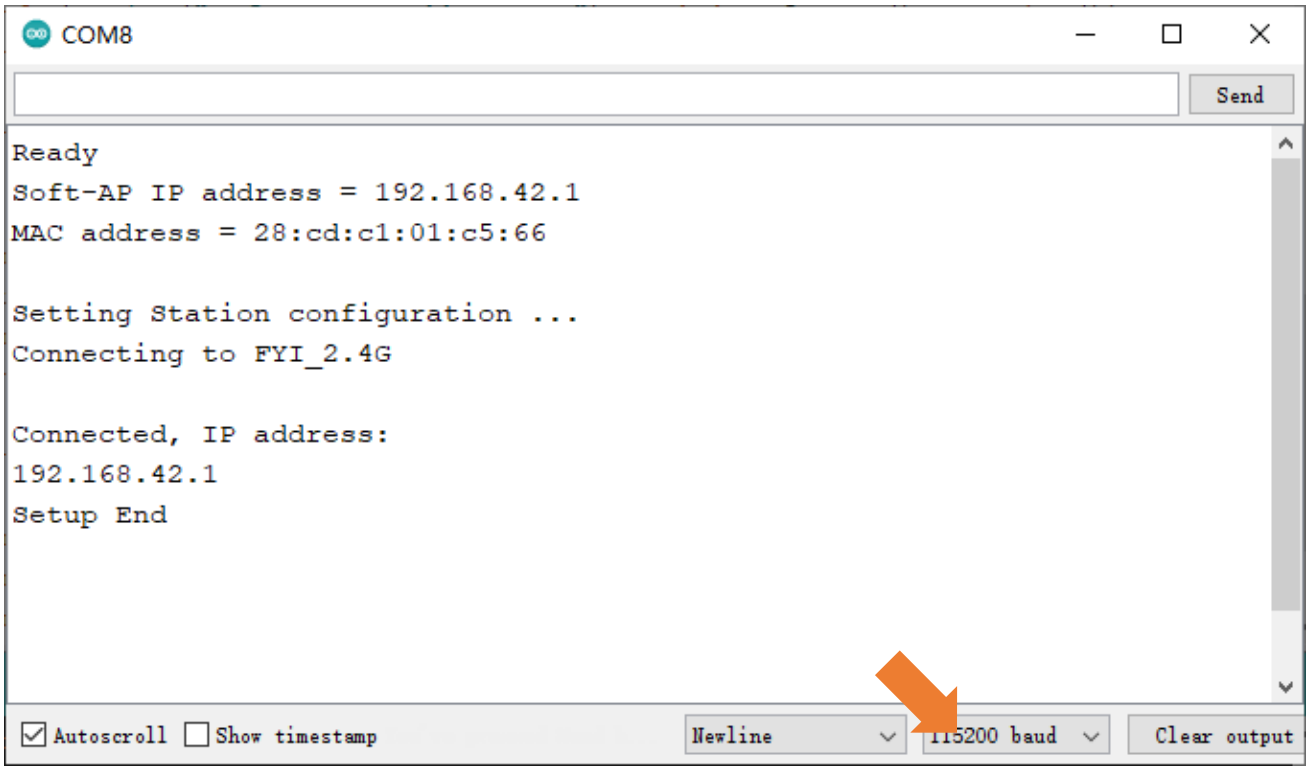


Sketch

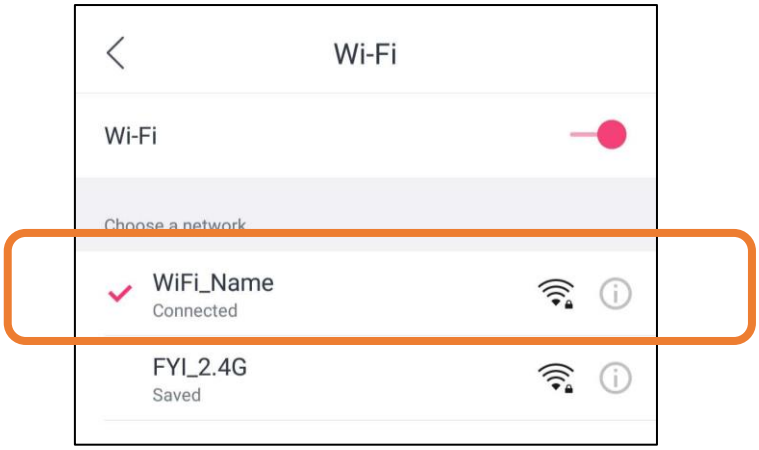
Sketch_21.3_AP_Station_mode

It is analogous to Project 21.1 and Project 21.2. 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. And then it will display as follows:



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



The following is the program code:

```
#include <WiFi.h>

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

void setup(){
```

```
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  }
```

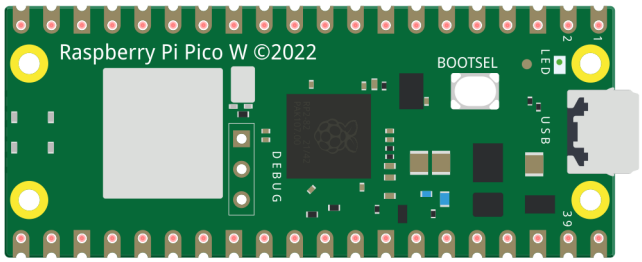
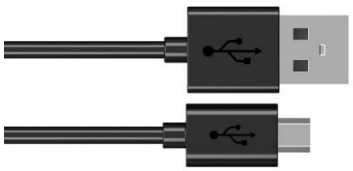
Chapter 22 TCP/IP (Only for Pico W)

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 22.1 As Client

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

Component List

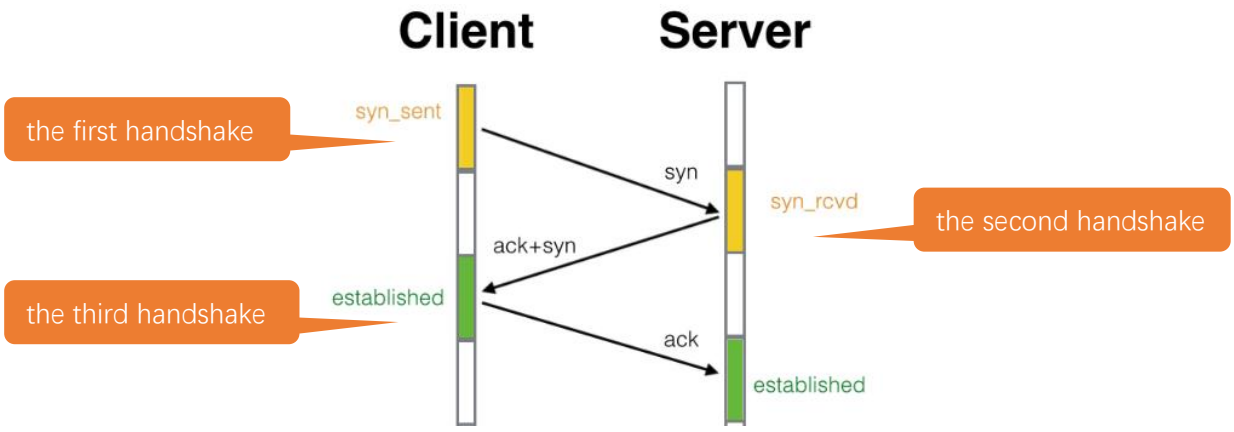
Raspberry Pi Pico W x1 	Micro USB Wire x1 

Component knowledge

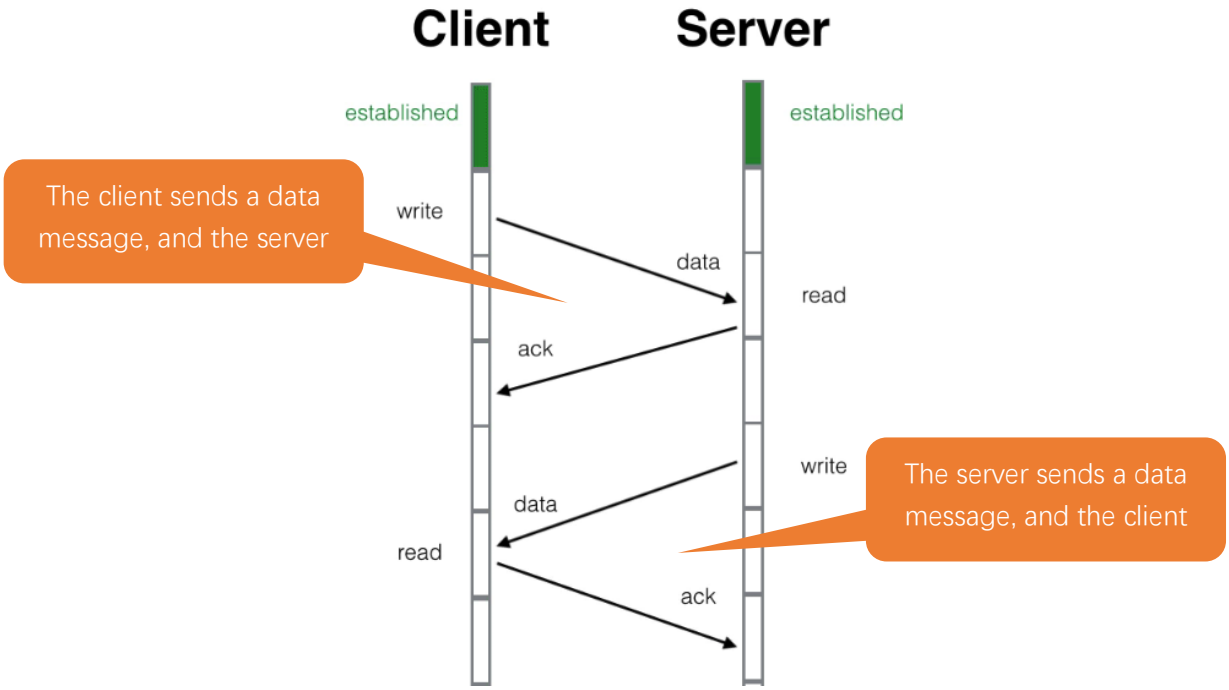
TCP connection

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-times handshake" is required.

Three-times handshake: In the TCP protocol, during the preparation phase of sending data, the client and the server interact three times to ensure the reliability of the connection, which is called "three-times handshake".
The first handshake, the client sends a connection request to the server and waits for the server to confirm.
The second handshake, the server sends a response back to the client informing that it has received the connection request.
The third handshake, the client sends a confirmation message to the server again to confirm the connection.



TCP is a connection-oriented, low-level transmission control protocol. After TCP establishes a connection, the client and server can send and receive messages to each other, and the connection will always exist as long as the client or server does not initiate disconnection. Each time one party sends a message, the other party will reply with an ack signal.



Install Processing

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

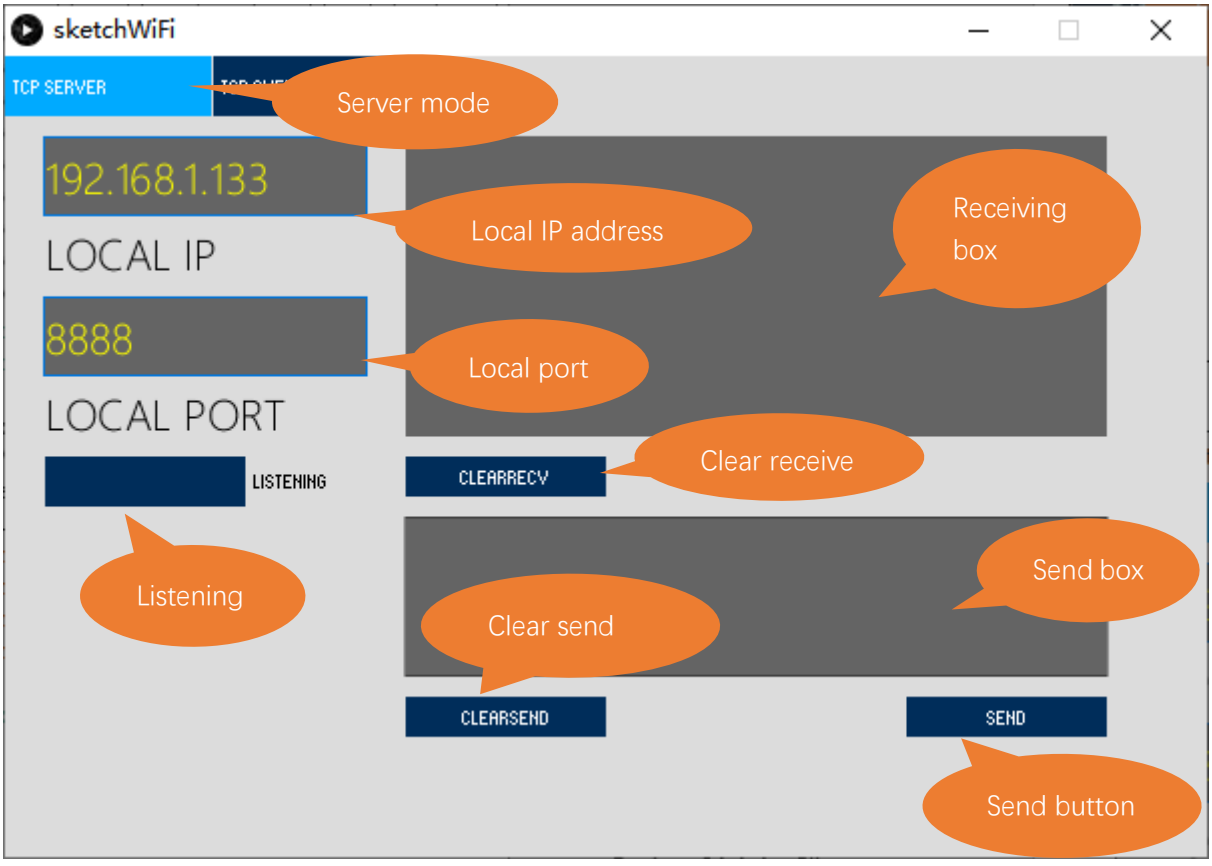
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

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

Use Server mode for communication

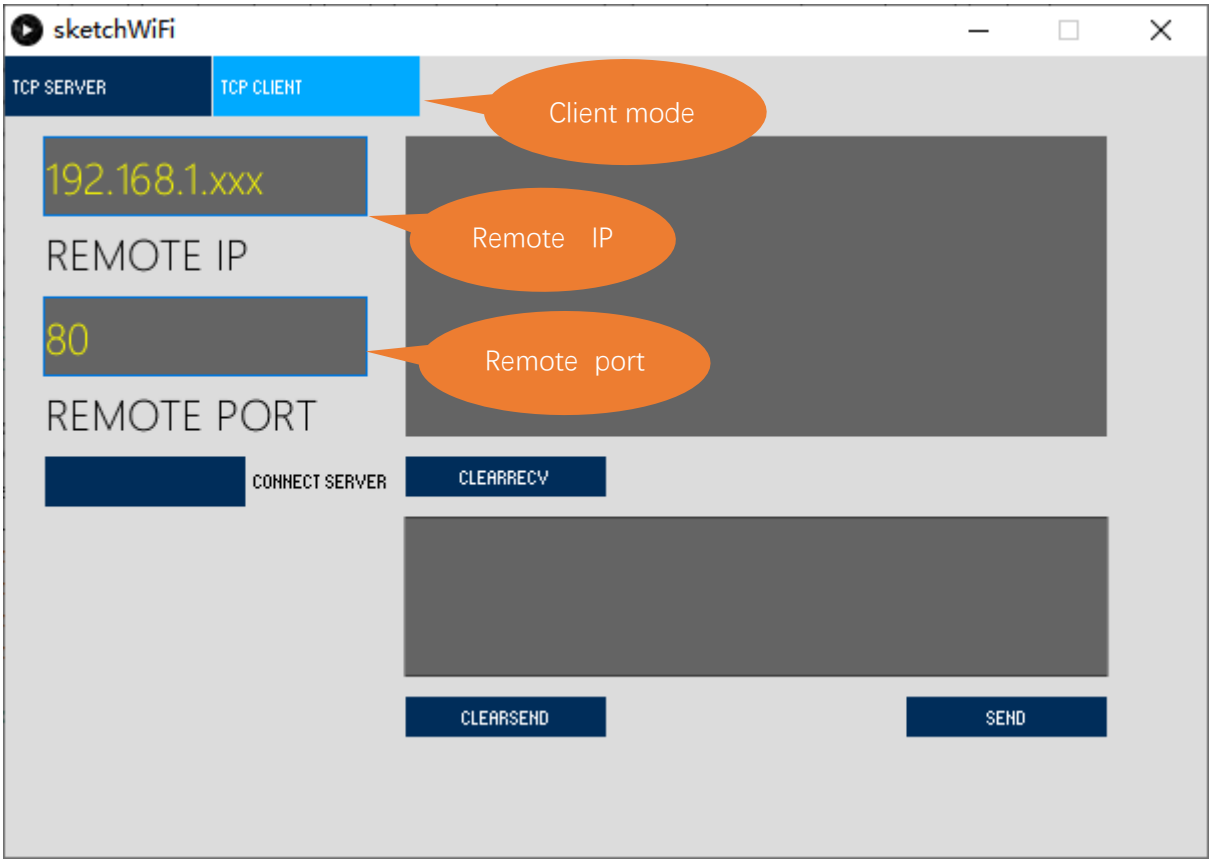
Open the "**Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_22.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

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.

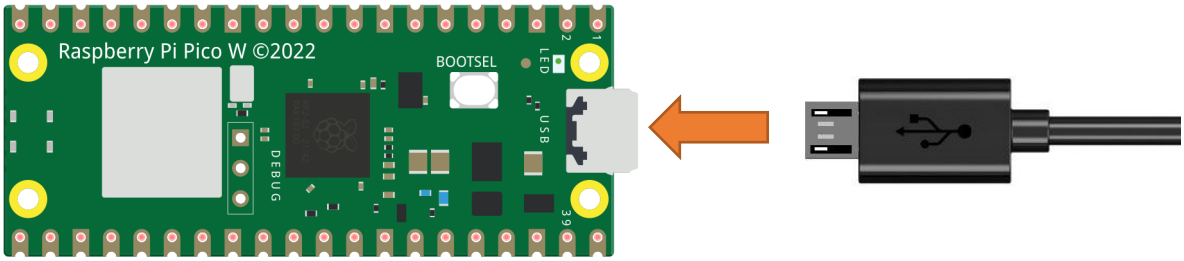
clear receive: clear out the content in the receiving text box

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

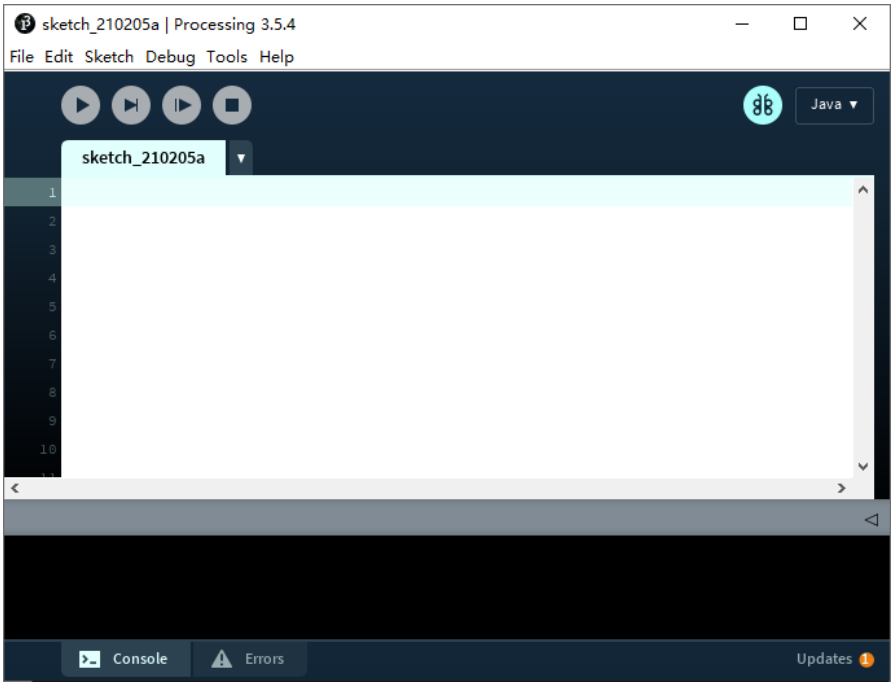
Connect Pico W to the computer using the USB cable.



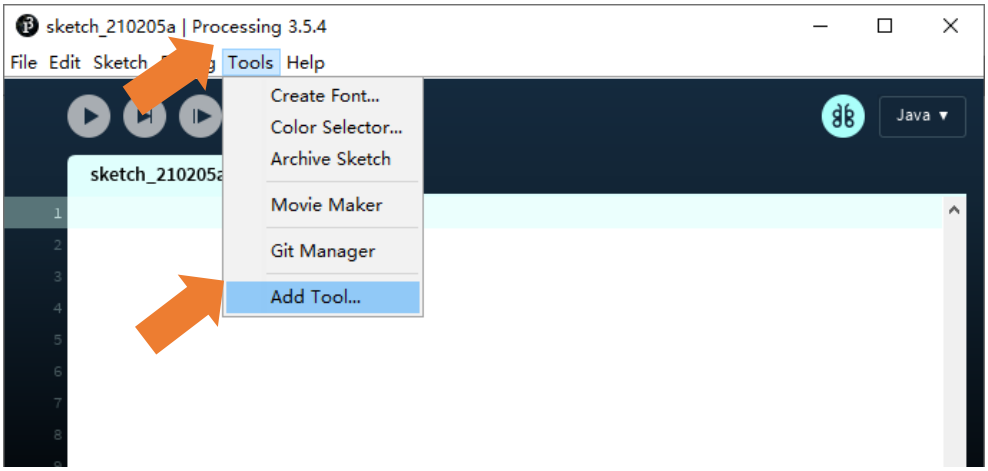
Sketch

If you have not installed "ControlP5", please follow the following steps to continue the installation, if you have installed, please skip this section.

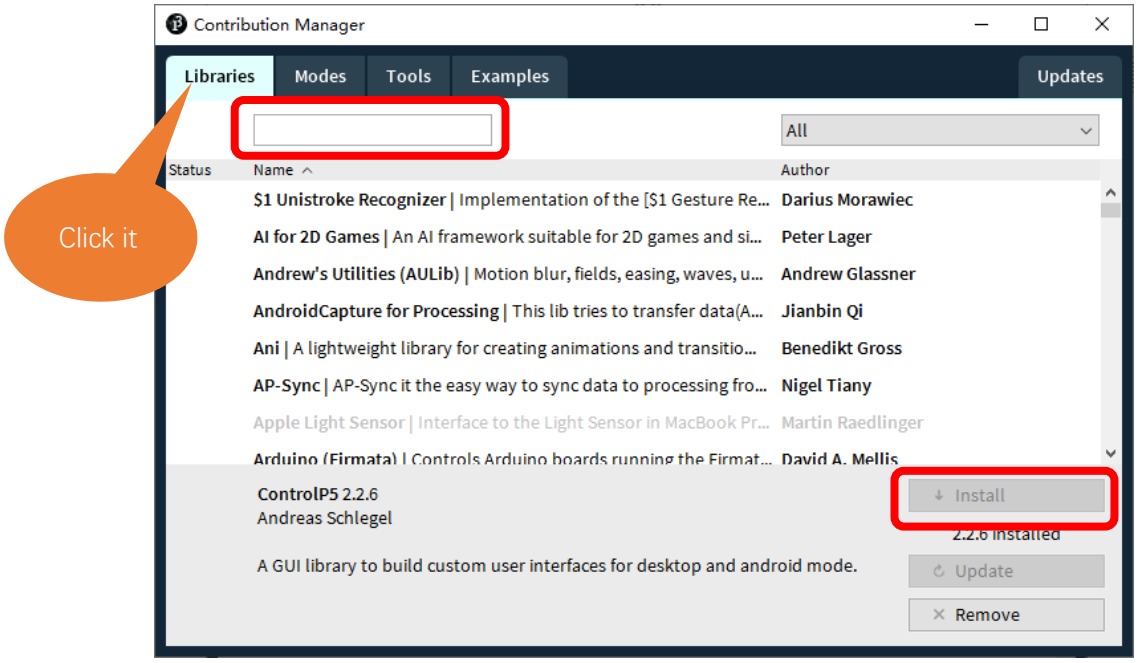
Open Processing.



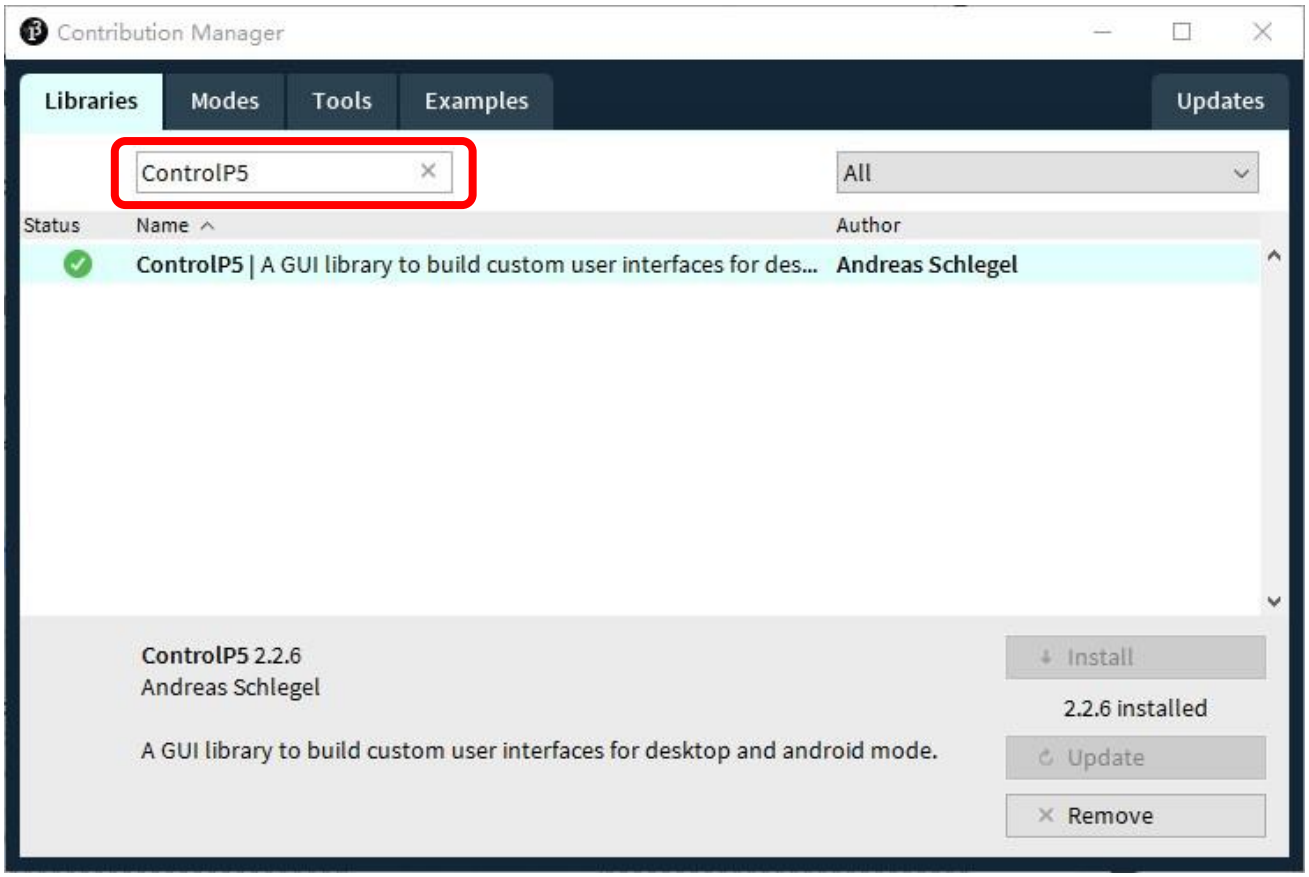
Click Add Tool under Tools.



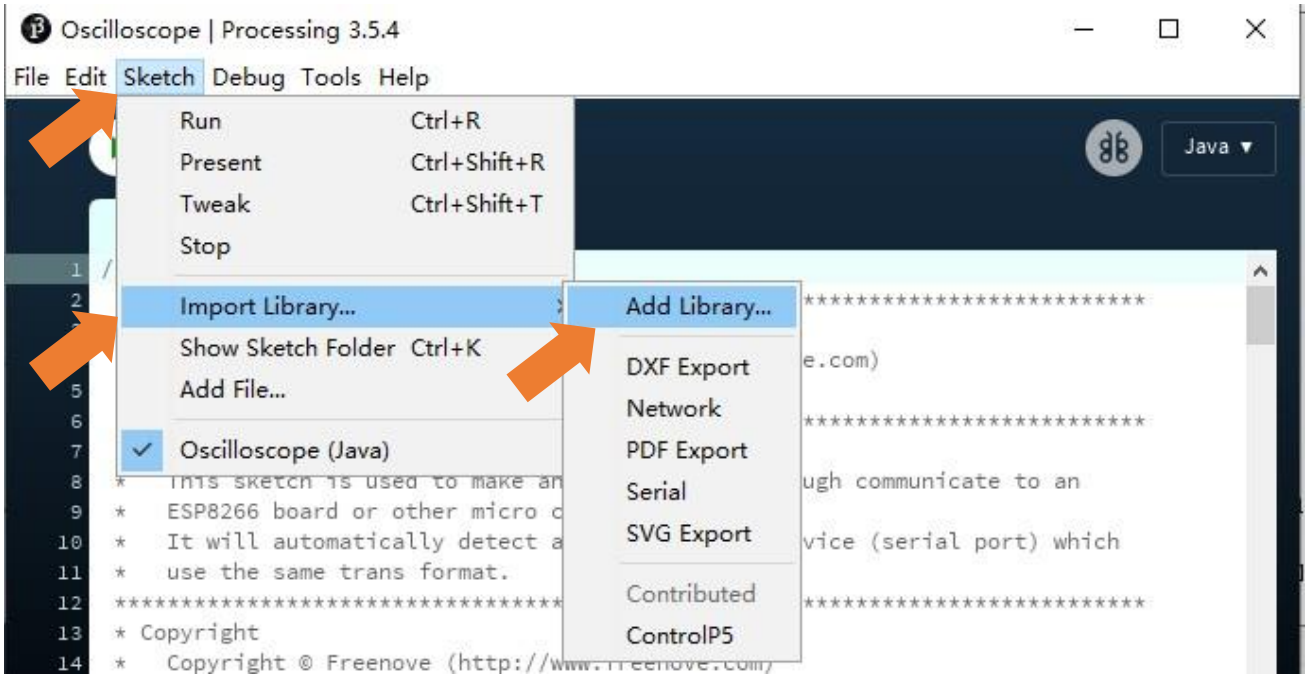
Select Libraries in the pop-up window.



Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

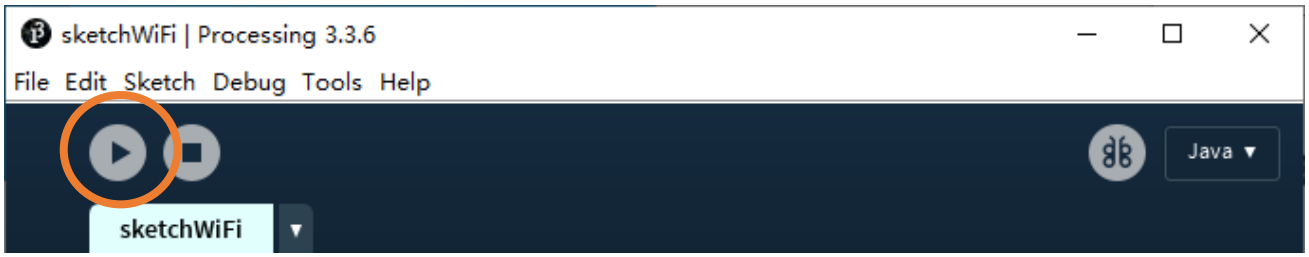


You can also click Add Library under 'Import Library' under 'Sketch'.

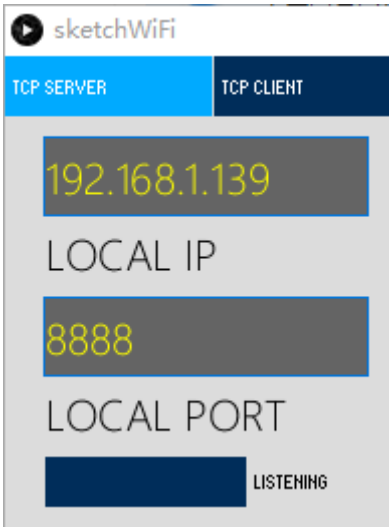


Sketch_22.1_As_Client

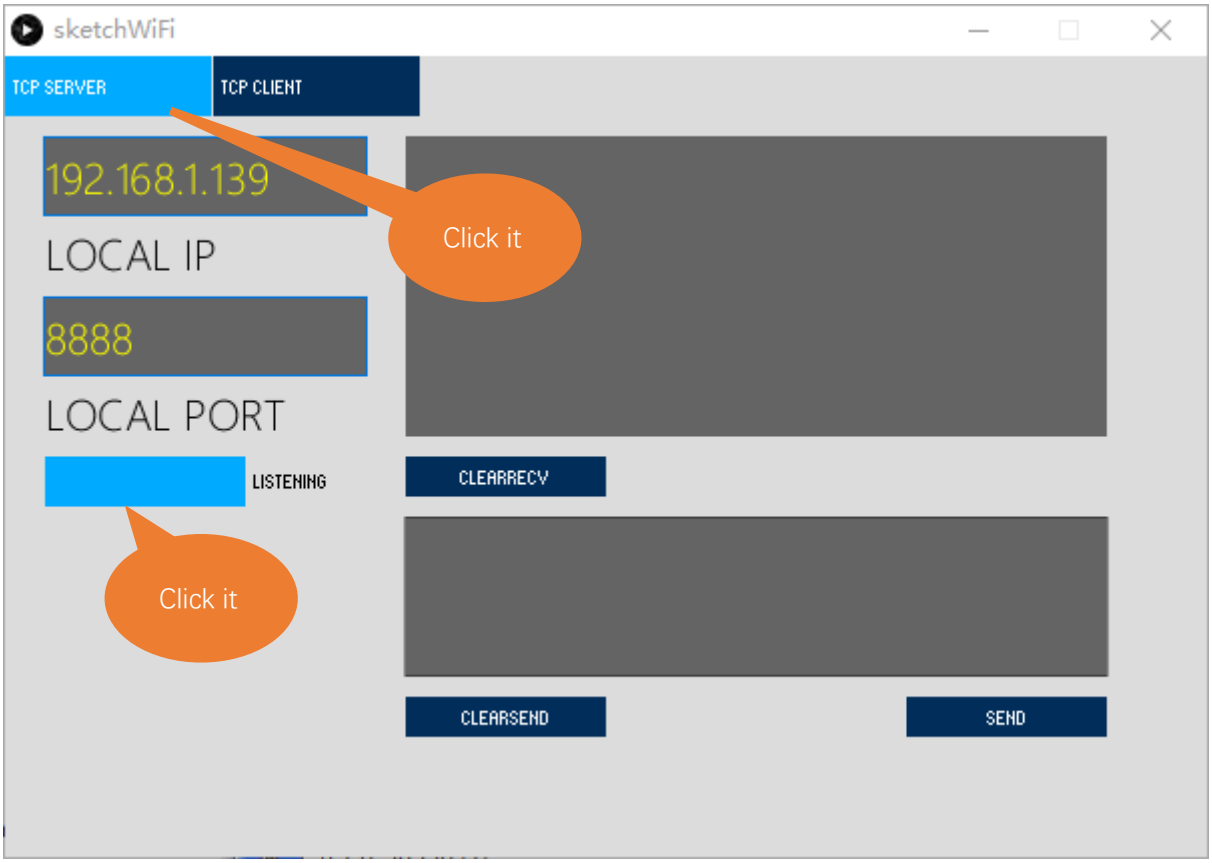
Before running the Sketch, please open "sketchWiFi.pde." first, and click "Run".



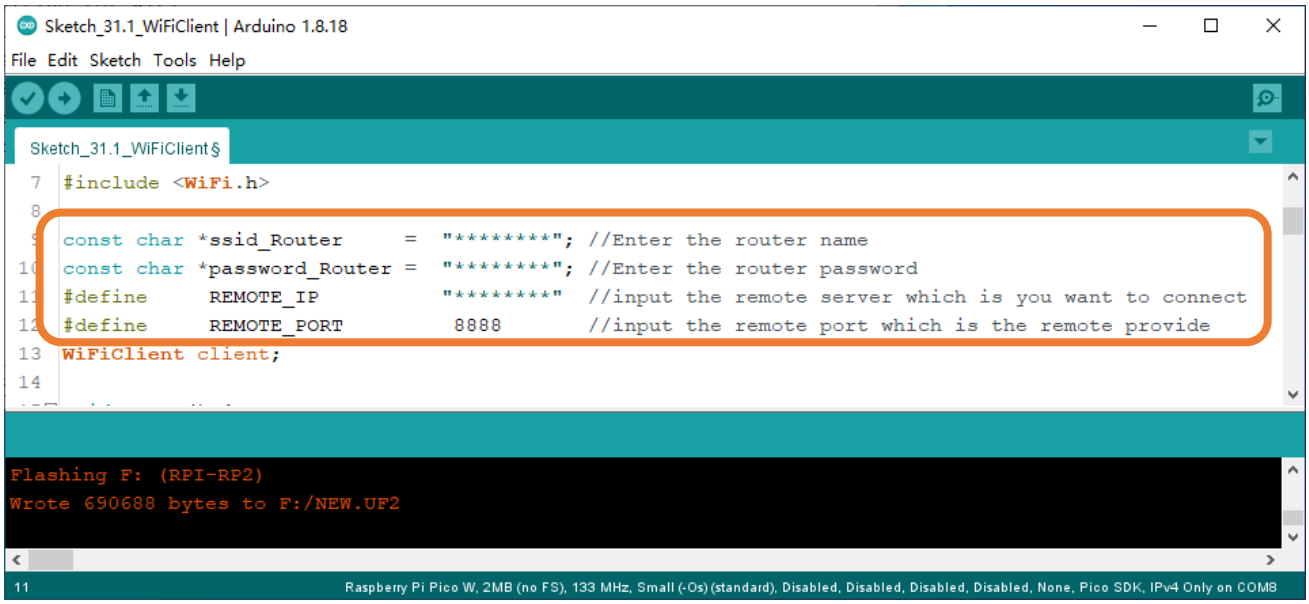
The newly pop up window will use the computer's IP address by default and open a data monitor port.



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

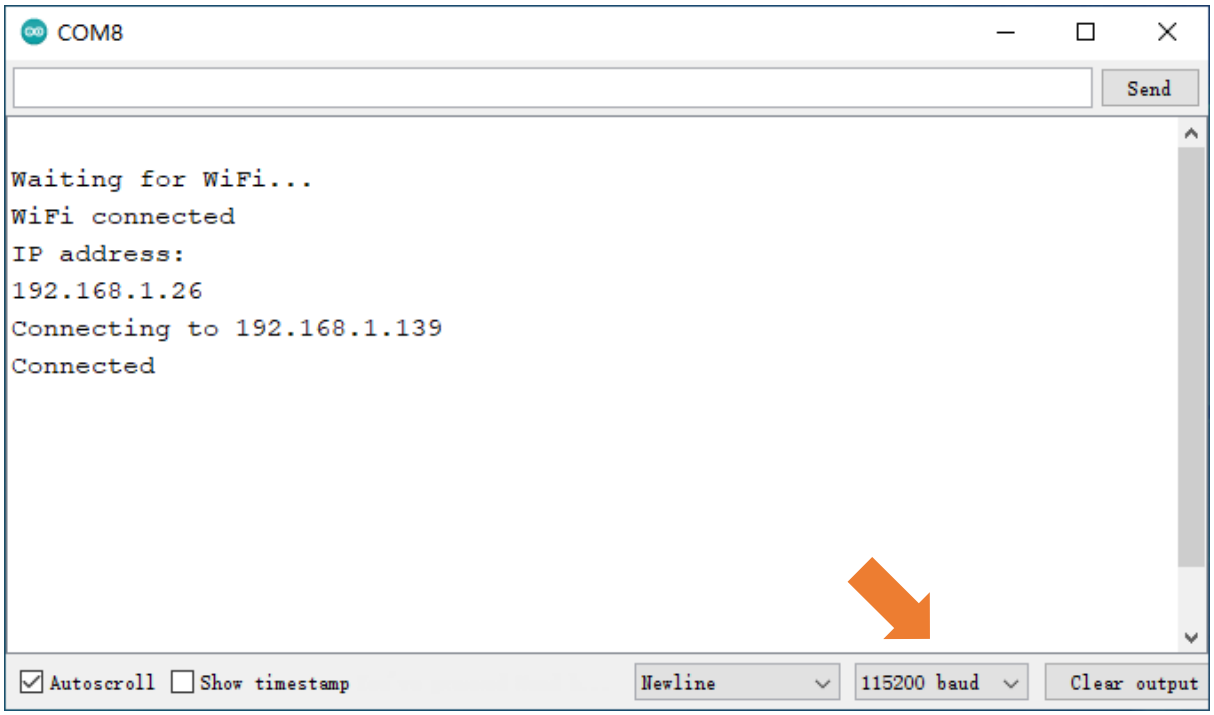


Next, open Sketch_22.1_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.

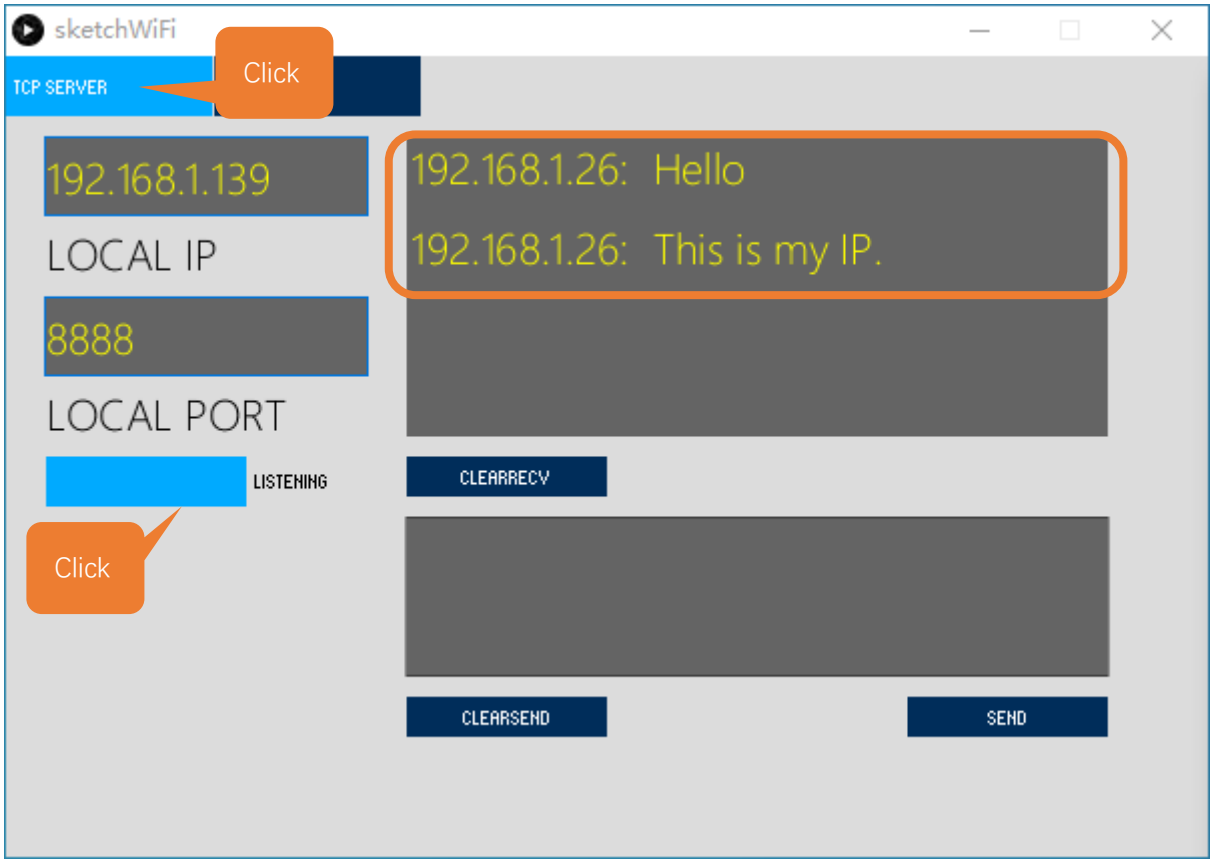


REMOTE_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE_IP is “192.168.1.139”. Generally, by default, the ports do not need to change its value.

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.



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:

```
#include <WiFi.h>

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

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

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

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

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

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

```
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) {
```

```
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.

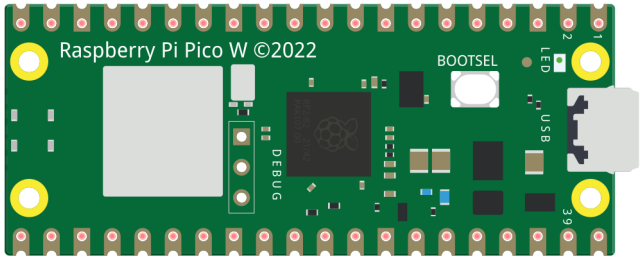
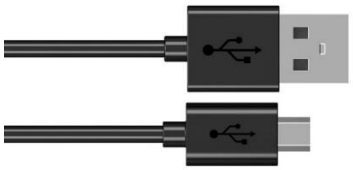
read(): read one byte of data in receive buffer

readString(): read string in receive buffer

Project 22.2 As Server

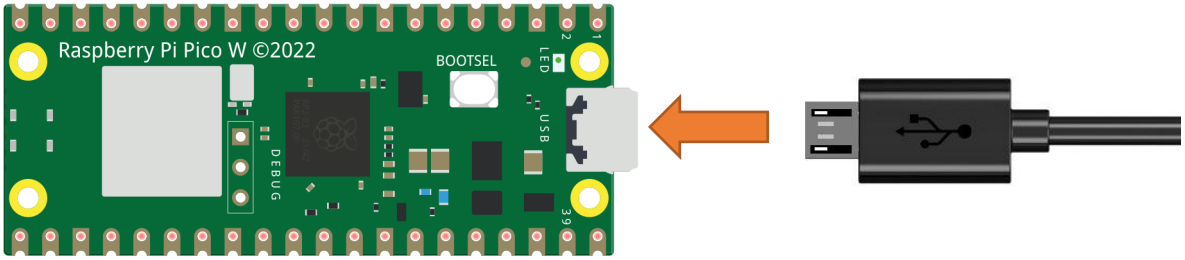
In this section, PICO W is used as a server to wait for the connection and communication of client on the same LAN.

Component List

Raspberry Pi Pico W x1 	Micro USB Wire x1 

Circuit

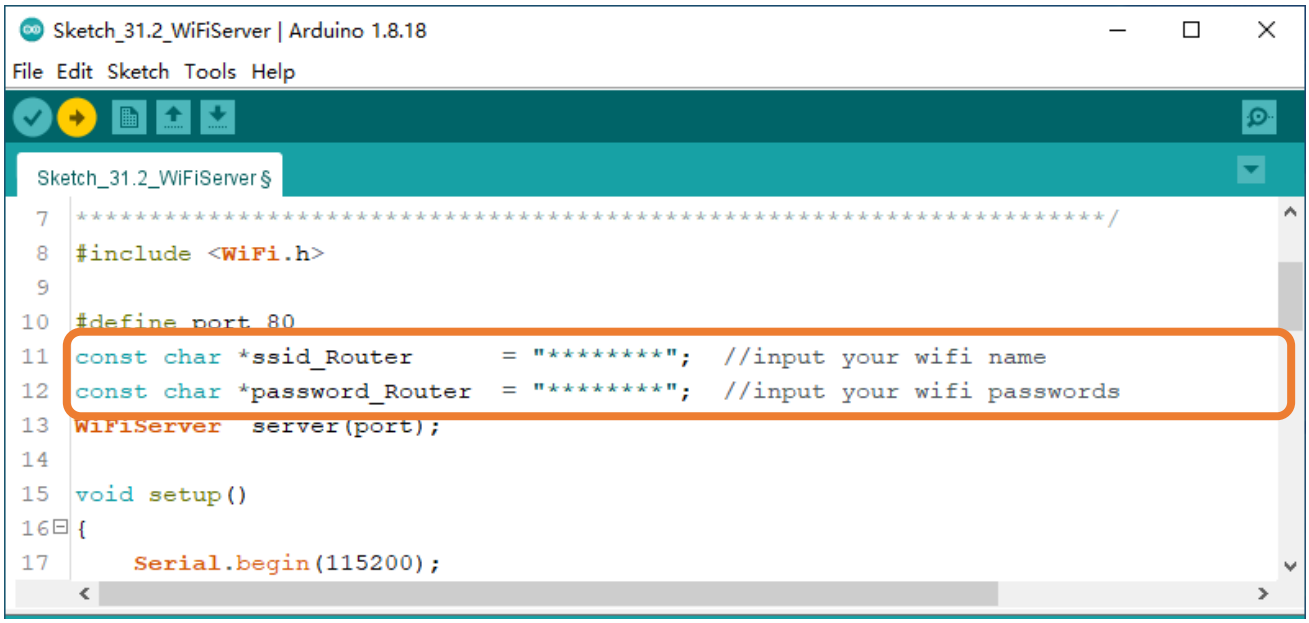
Connect Pico W to the computer using the USB cable.



Sketch

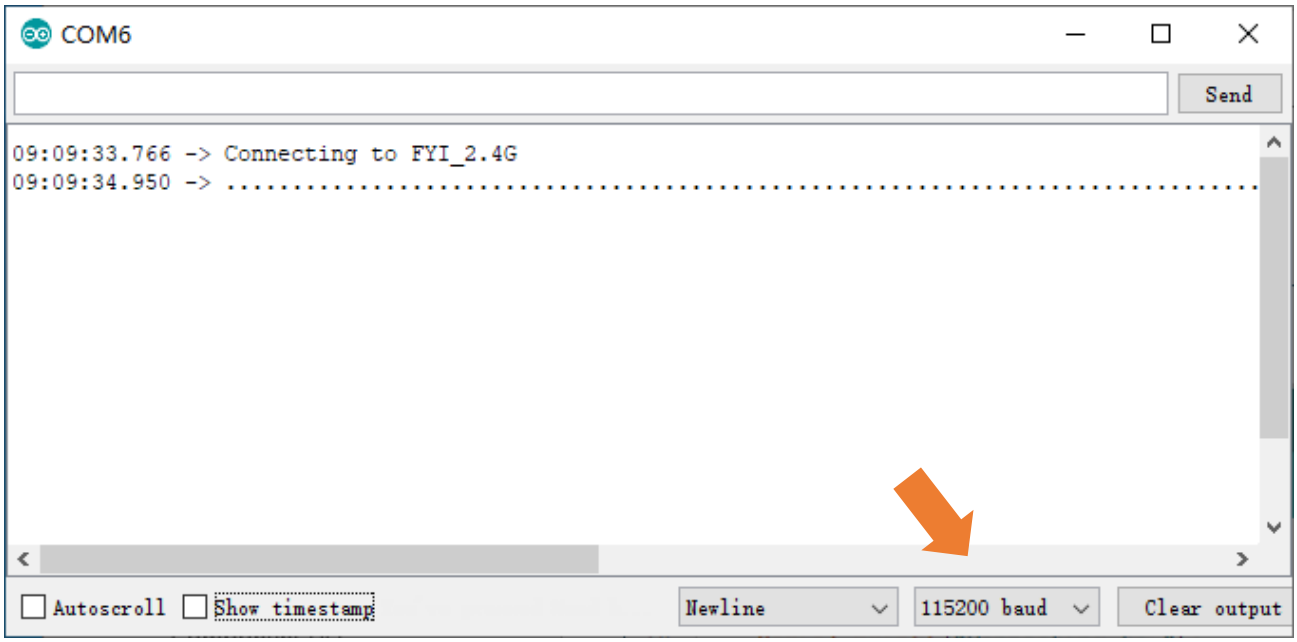
Before running Sketch, please modify the contents of the box below first.

Sketch_22.2_As_Server

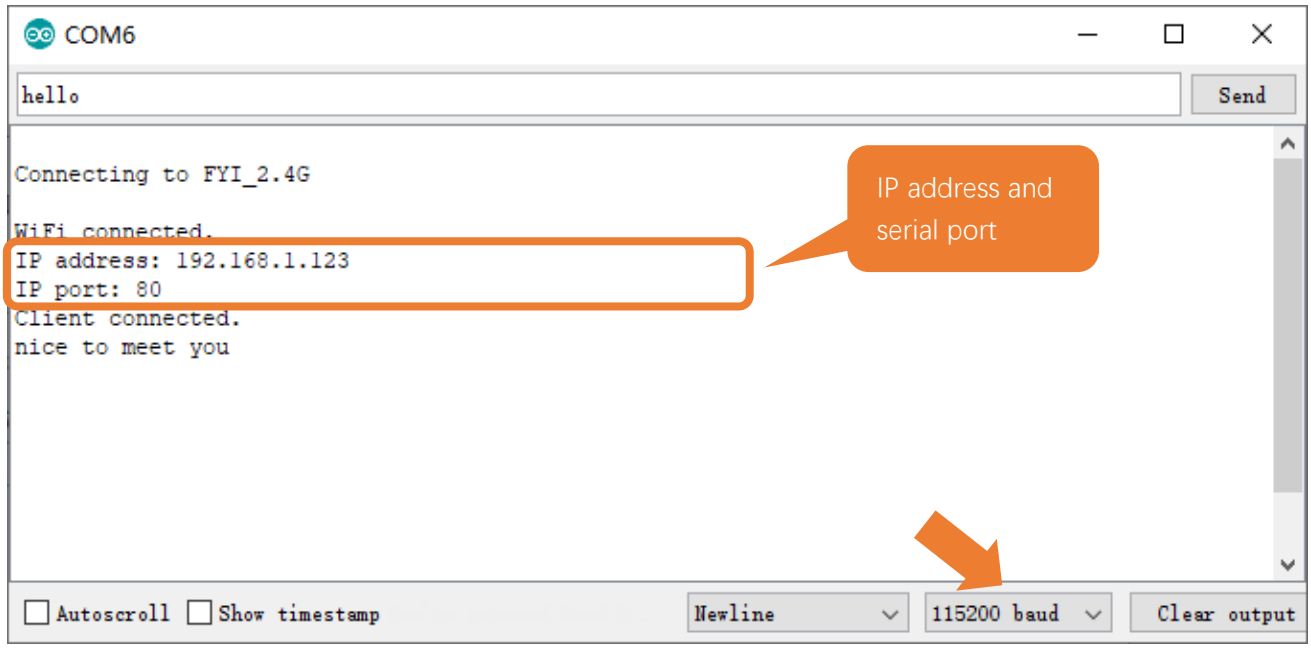


Compile and upload code to PICO W board, open the serial monitor and set the baud rate to 115200. Turn on server mode for PICO W, waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other.

If the Pico W fails to connect to router, please disconnect the power supply and try again several times.



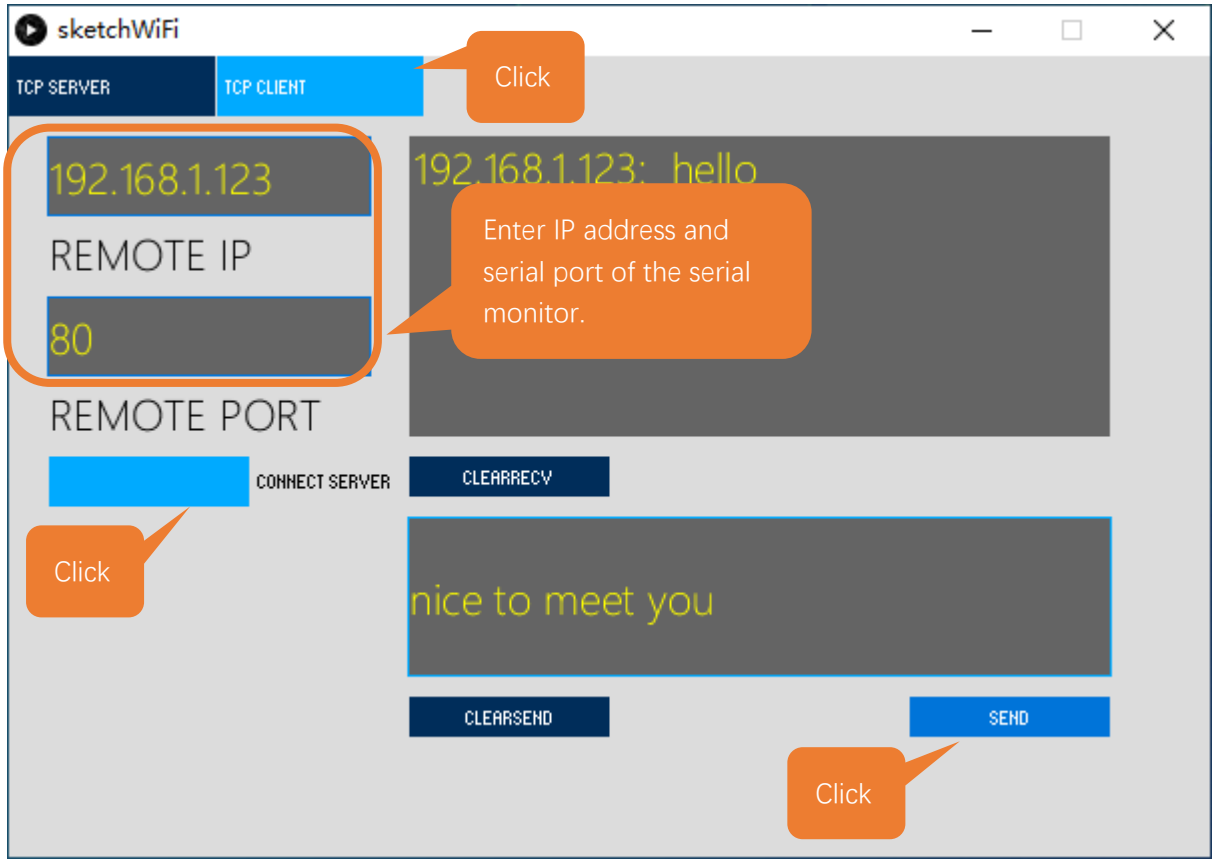
Serial Monitor



Processing:

Open the "**Freenove_Super_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_22.2_WiFiServer\sketchWiFi\sketchWiFi.pde**".

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



The following is the program code:

```
#include <WiFi.h>

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

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

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

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

Apply for method class of WiFiServer.

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

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please 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      Serial.print("IP address: ");
23      Serial.println(WiFi.localIP());                  //print out IP address of PICO W
24      Serial.printf("IP port: %d\n",port);             //Print out PICO W's port number
```

Turn on server mode of PICO W.

```
25      server.begin();                                  //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: ports of Server; range from 0 to 65535 with the default number as 0.

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

nodelay: true stands for forbidden Nagle algorithm.

close(): close tcp connection.

stop(): stop tcp connection.

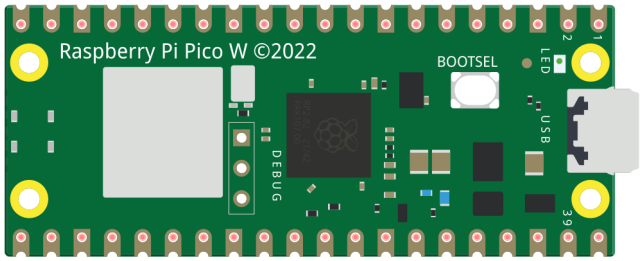
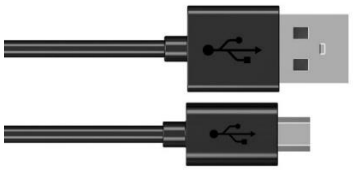
Chapter 23 Control LED with Web (Only for Pico W)

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 23.1 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

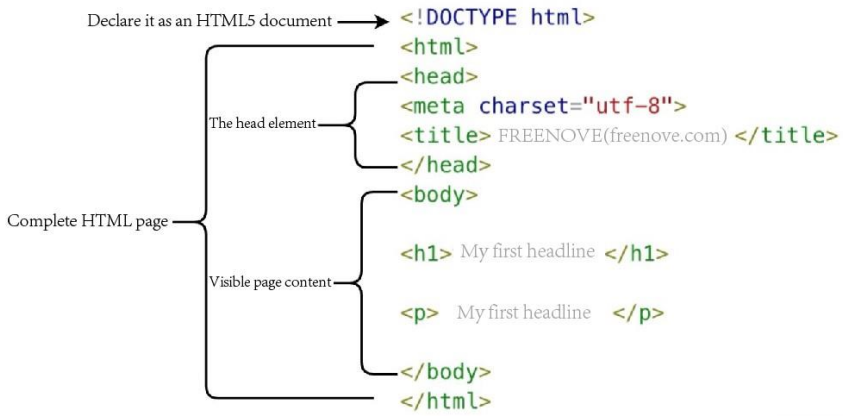
HTML

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.Hyper Text 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>: Notesthe title of the document

<body>: Contains visible page content

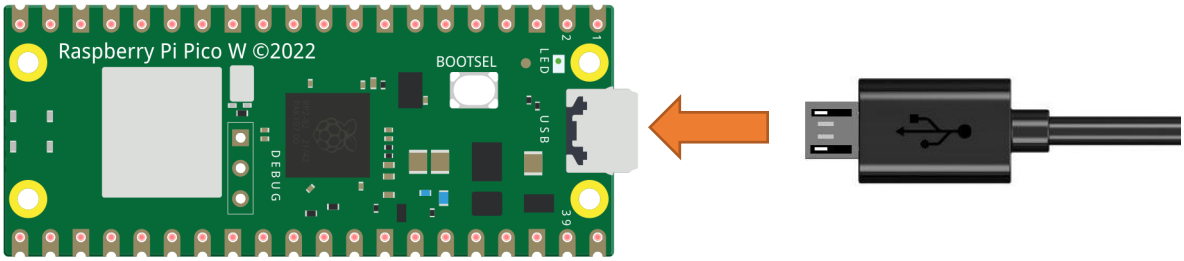
<h1>: Define a big heading

<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_23.1_Control_the_LED_with_Web

Sketch_32.1_Control_the_LED_with_Web | Arduino 1.8.18

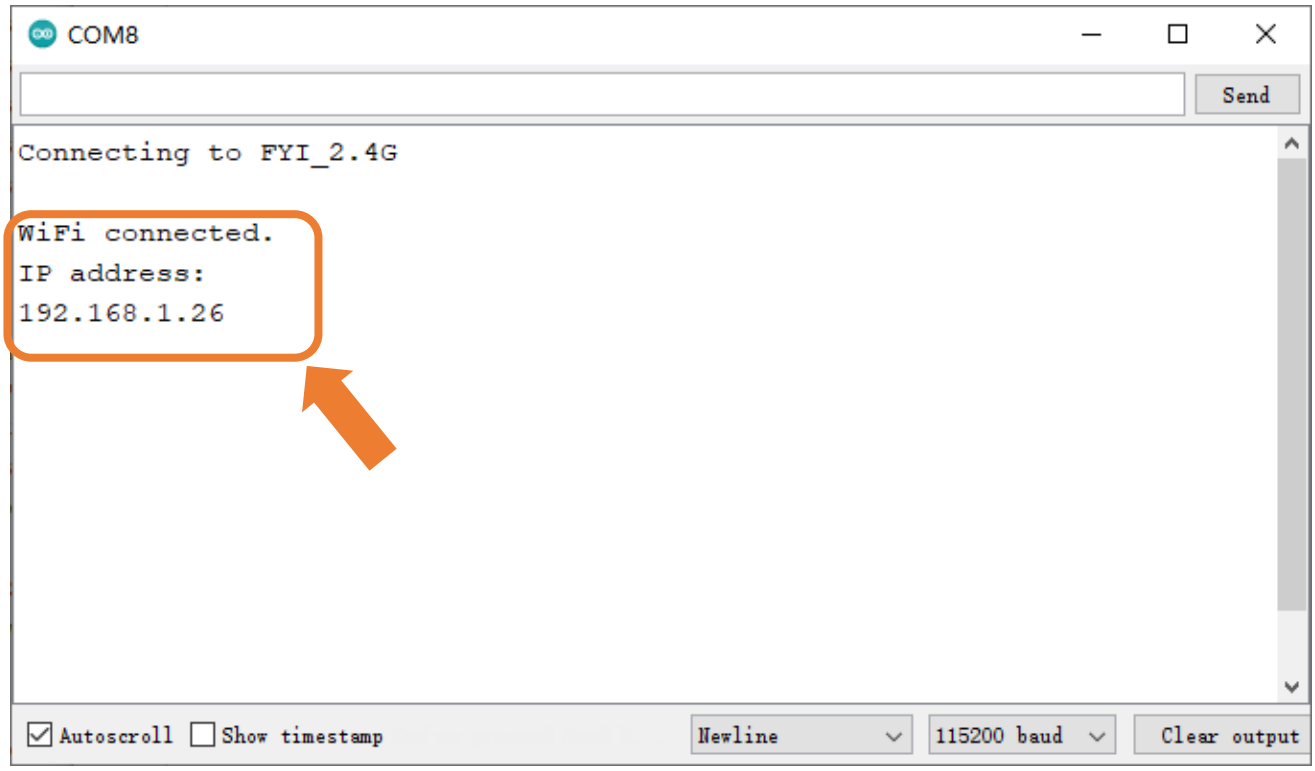
File Edit Sketch Tools Help

Sketch_32.1_Control_the_LED_with_Web

Enter the correct Router name and password.

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

// Replace with your network credentials
const char* ssid     = "********";
const char* password = "********";

// Set web server port number to 80
WiFiServer server(80);
// Variable to store the HTTP request
String header;
// Auxiliar variables to store the current output state
String PIN_LEDState = "OFF";

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  // Connect to Wi-Fi network with SSID and password
```

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:



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://192.168.1.26 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:

```
#include <WiFi.h>

// Replace with your network credentials
const char* ssid     = "********";
const char* password = "********";

// Set web server port number to 80
WiFiServer server(80);
// Variable to store the HTTP request
String header;
// Auxiliar variables to store the current output state
String PIN_LEDState = "OFF";

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
void loop() {
  WiFiClient client = server.available();  // Listen for incoming clients
```

```
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
      client
47      currentTime = millis();
48      previousTime = currentTime;
49      while (client.connected() && currentTime - previousTime <= timeoutTime) {  // loop while
      the client's connected
50        currentTime = millis();
51        if (client.available()) {  // if there's bytes to read from the client,
52          char c = client.read();  // read a byte, then
53          Serial.write(c);         // print it out the serial monitor
54          header += c;
55          if (c == '\n') {  // if the byte is a newline character
56            // if the current line is blank, you got two newline characters in a row.
57            // that's the end of the client HTTP request, so send a response:
58            if (currentLine.length() == 0) {
59              // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
60              // and a content-type so the client knows what's coming, then a blank line:
61              client.println("HTTP/1.1 200 OK");
62              client.println("Content-type:text/html");
63              client.println("Connection: close");
64              client.println();
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              }
75              // Display the HTML web page
76              client.println("<!DOCTYPE html><html>");
77              client.println("<head> <title>Pico W Web Server</title> <meta name=\"viewport\"
      content=\"width=device-width, initial-scale=1\">");
78              client.println("<link rel=\"icon\" href=\"data:,\">");
79              // CSS to style the on/off buttons
80              // Feel free to change the background-color and font-size attributes to fit your
      preferences
81              client.println("<style>html {font-family: Helvetica; display:inline-block; margin:
      0px auto; text-align: center;}");
82              client.println(" h1{color: #0F3376; padding: 2vh;} p{font-size: 1.5rem;}");
```

```
83              client.println(".button{background-color: #4286f4; display: inline-block; border:
                none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
                30px; margin: 2px; cursor: pointer;}");
84              client.println(".button2{background-color: #4286f4;display: inline-block; border:
                none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
                30px; margin: 2px; cursor: pointer;}</style></head>");
85              // Web Page Heading
86              client.println("<body><h1>Pico W Web Server</h1>");
87              client.println("<p>GPIO state: " + PIN_LEDState + "</p>");
88              client.println("<p><a href=\"/LED_BUILTIN/ON\"><button class=\"button
                button2\">ON</button></a></p>");
89              client.println("<p><a href=\"/LED_BUILTIN/OFF\"><button class=\"button
                button2\">OFF</button></a></p>");
90              client.println("</body></html>");
91              // The HTTP response ends with another blank line
92              client.println();
93              // Break out of the while loop
94              break;
95            } else {  // if you got a newline, then clear currentLine
96              currentLine = "";
97            }
98          } else if (c != '\r') {  // if you got anything else but a carriage return character,
99            currentLine += c;      // add it to the end of the currentLine
100         }
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);
```

```
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              }
```

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 you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: support@freenove.com
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.