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_Basic_Starter_Kit_for_Raspberry_Pi_Pico

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

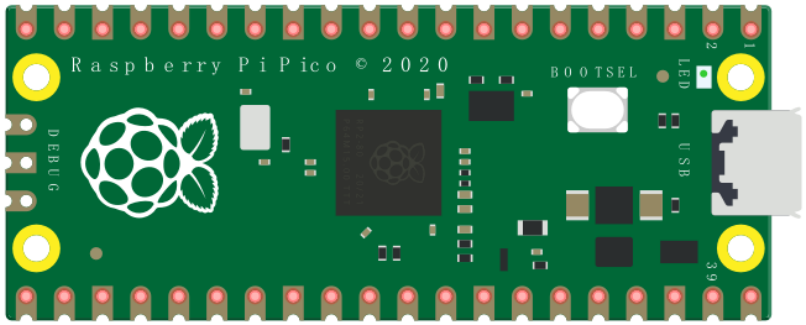
1, Component list: helps users to learn and find what components are needed in each project.
2, Component Knowledge: allows you to learn the features and usage of the components.
3, Circuit: assists to build circuit for each project.
4, Sketches and comments: makes it easier for users to learn to use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

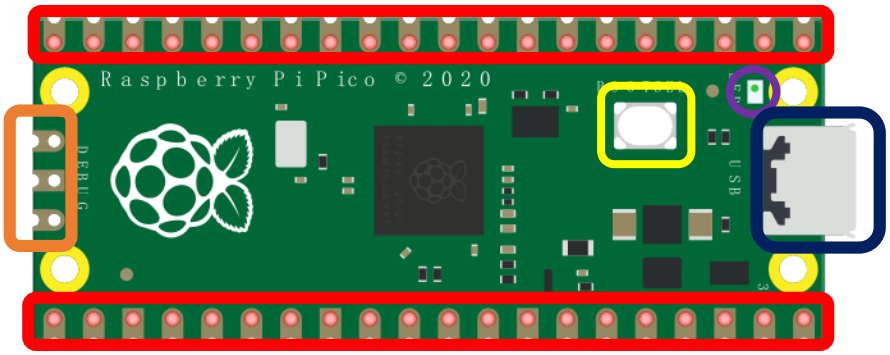
If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

Raspberry Pi Pico

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

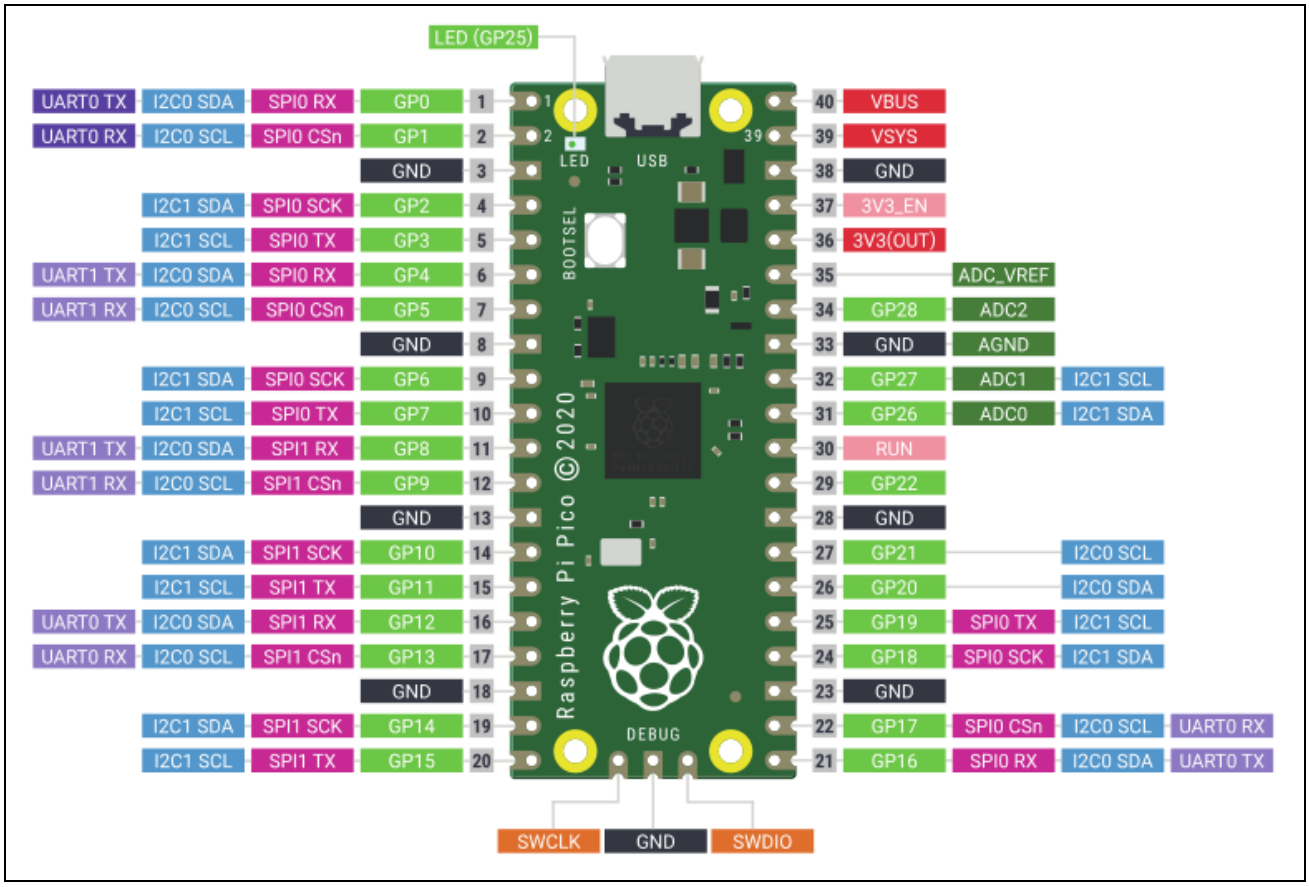


The hardware interfaces are distributed as follows:



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 6
I2C_SCL	Pin 7

SPI

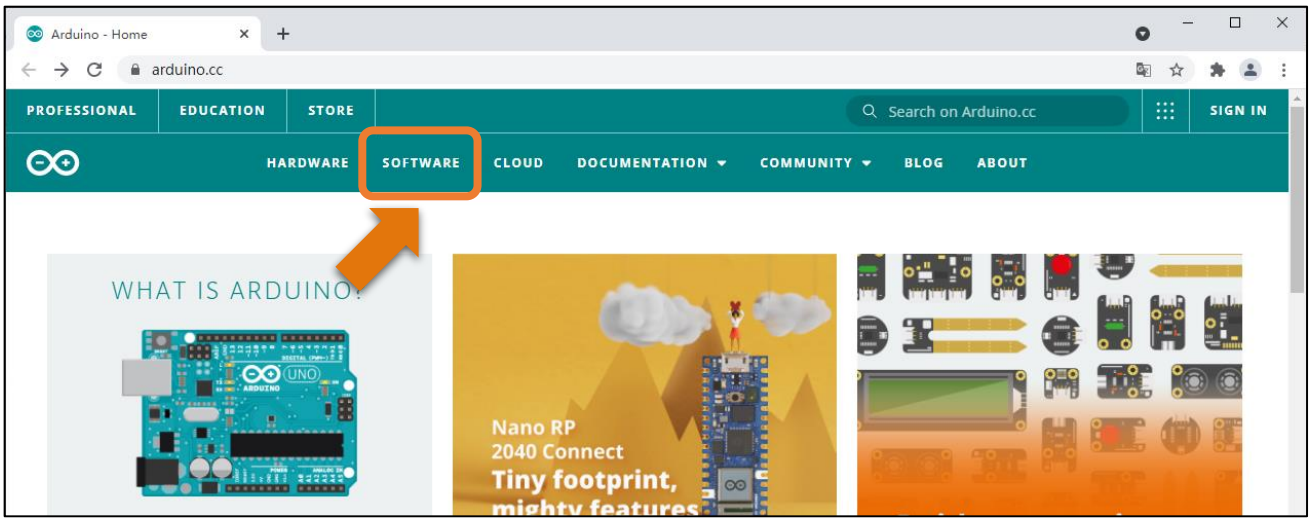
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 2
SPI_MOSI	Pin 3
SPI_MISO	Pin 4
SPI_SS	Pin 5

Chapter 0 Getting Ready (Important)

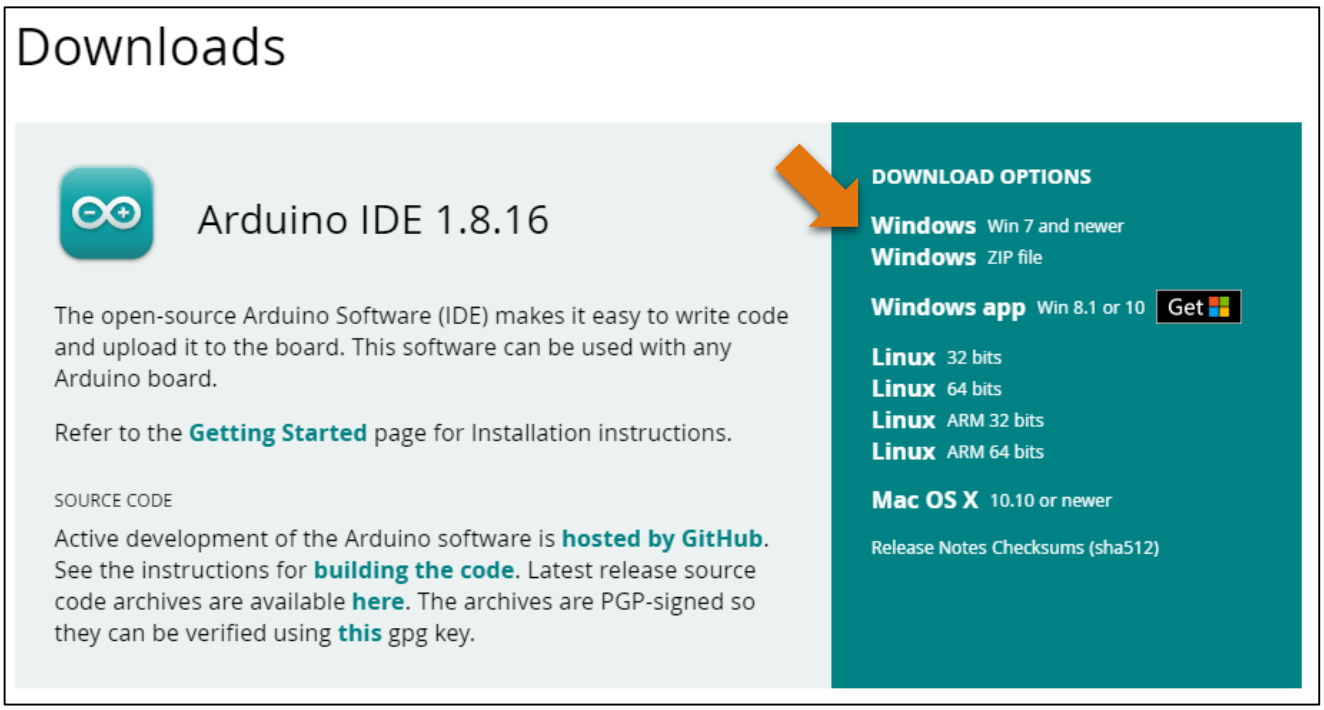
Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

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.

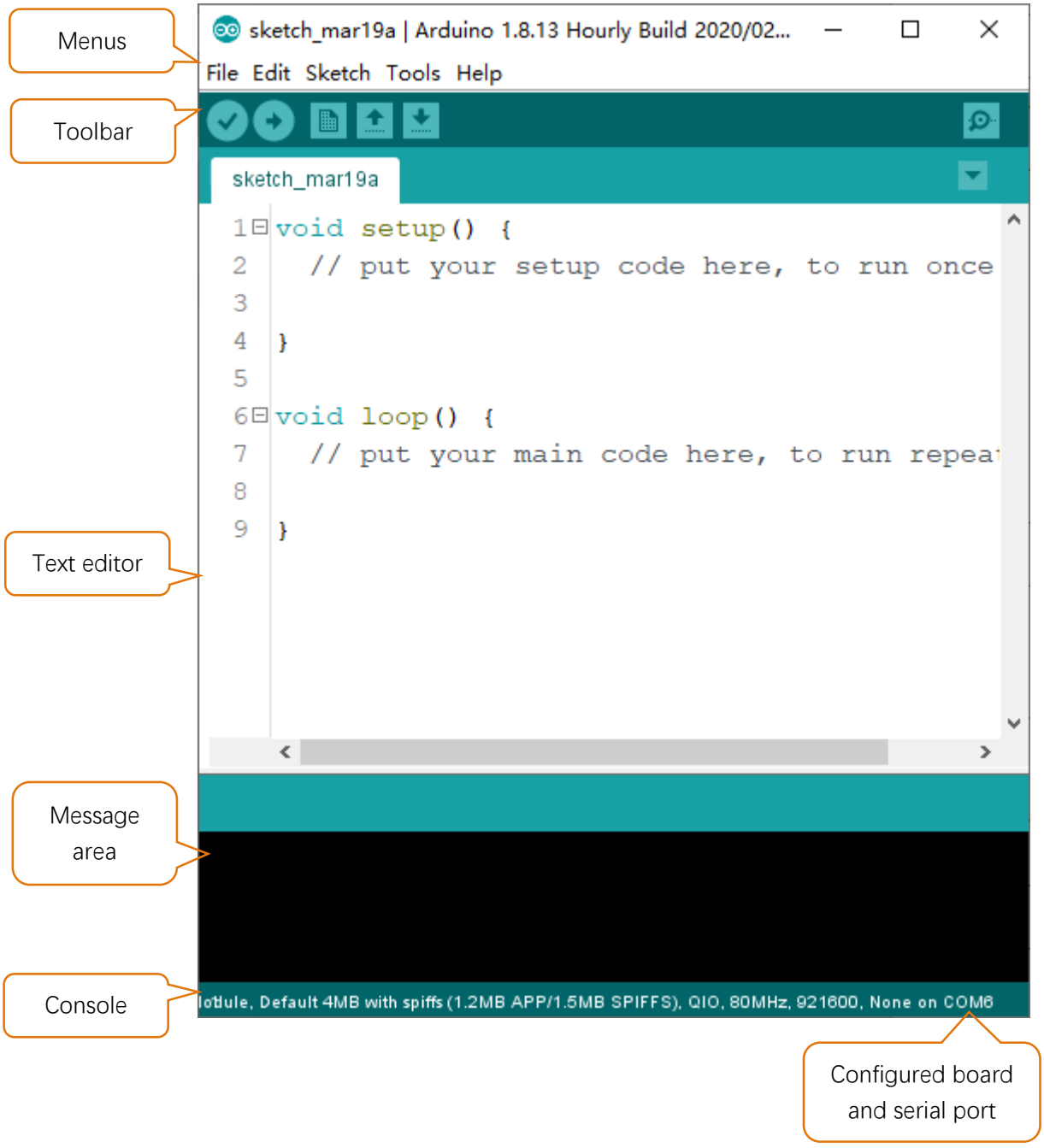


Select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer" to download to install the driver correctly.



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.

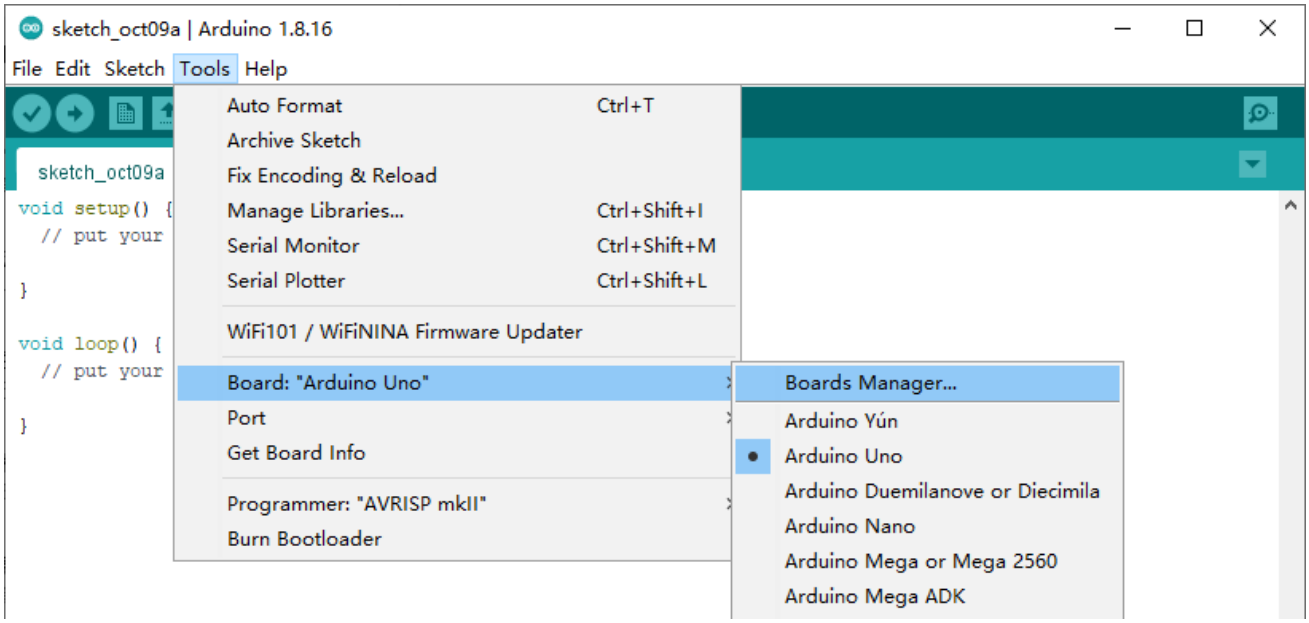
Serial Monitor
Open the serial monitor.

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.

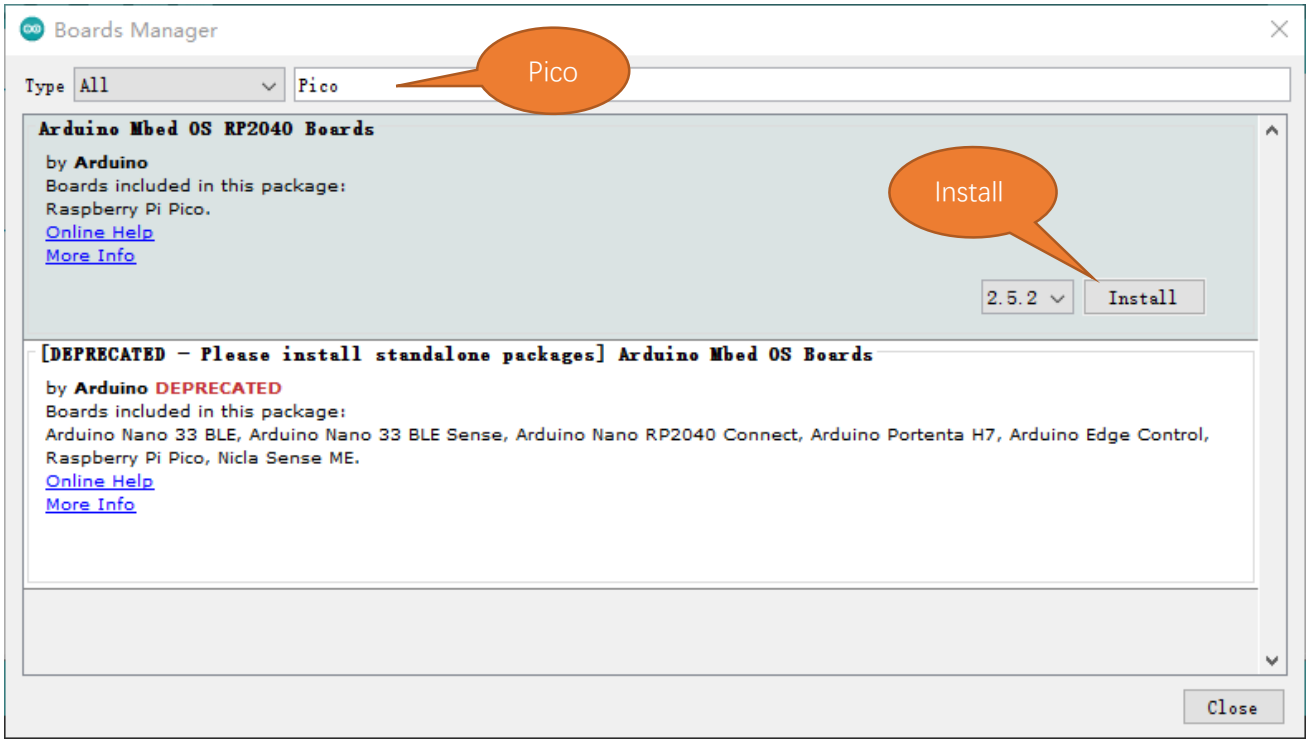
Installation of Development Board Support Package

1, Make sure your network is of good connection.

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



3, Enter Pico in the searching box, select "Arduino Mbed OS RP2040 Boards" and click on Install.

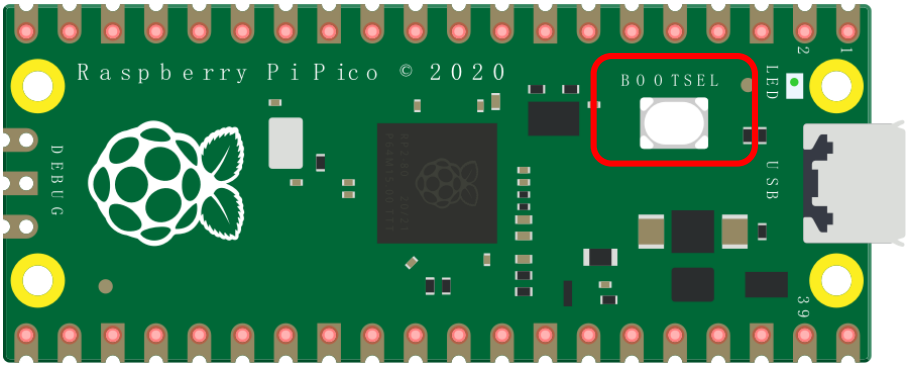


4, Click Yes in the pop-up "**dpinst-amd64.exe**"installation window. (Without it, you will fail to communicate with Arduino.) Thus far, we have finished installing the development support package.

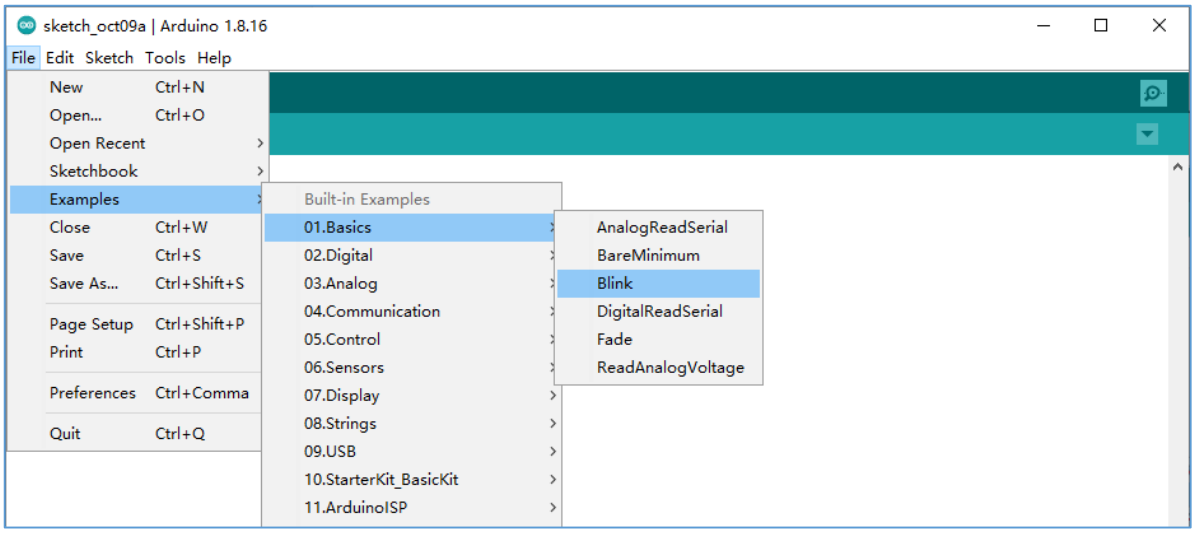
Uploading Adruino-compatible Firmware for Pico

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.

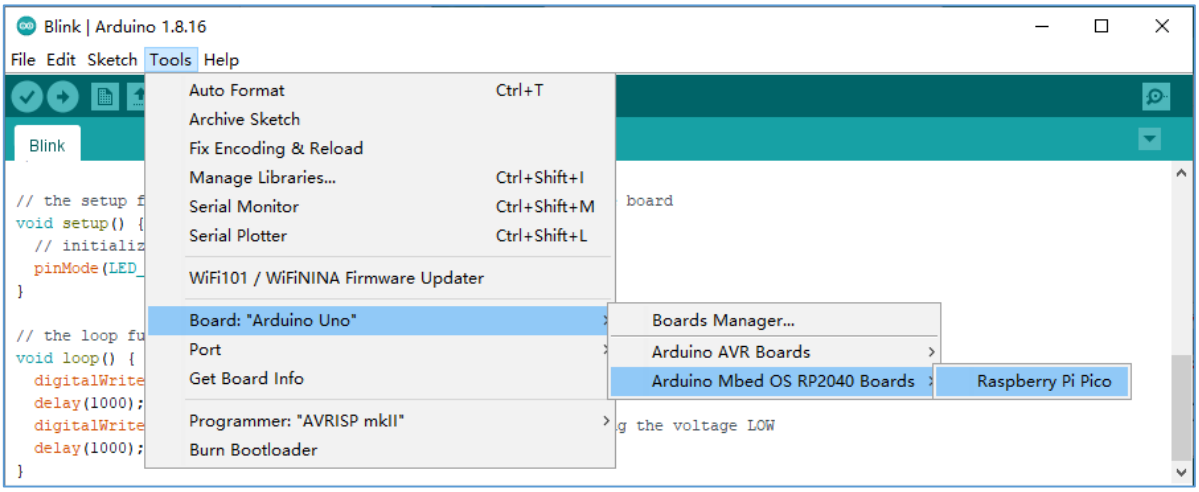
1, Disconnect Pico from computer. Keep pressing the white button(BOOTSEL) on Pico, and connect Pico to computer before releasing the button. (Note: Be sure to keep pressing the button before powering the Pico, otherwise the firmware will not download successfully)



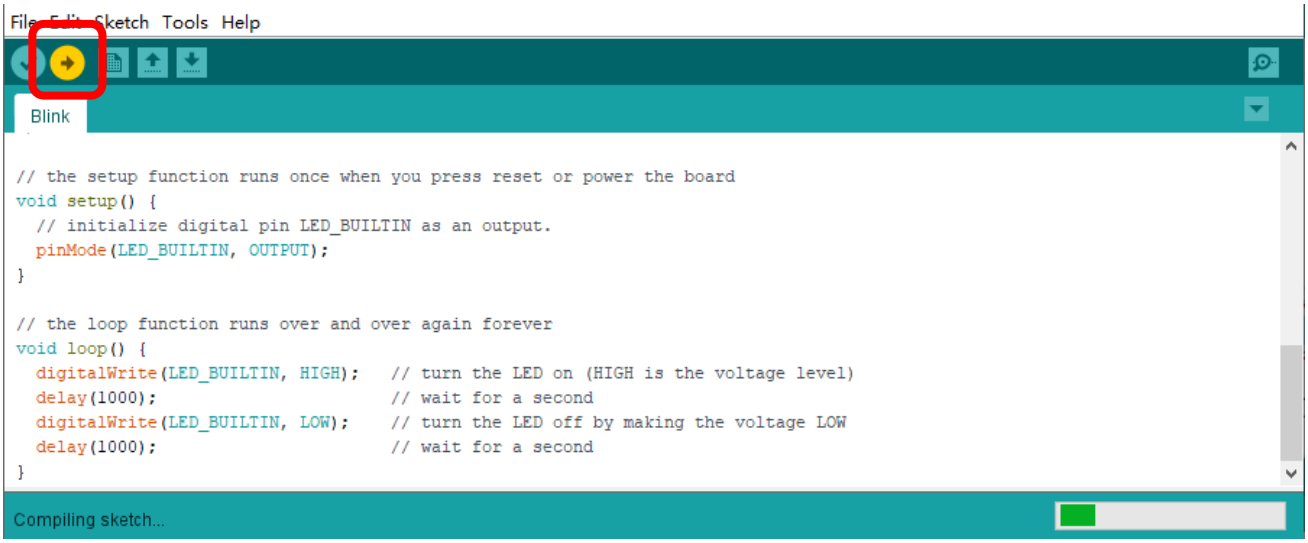
2, Open Arduino IDE. Click File>Examples>01.Basics>Blink.



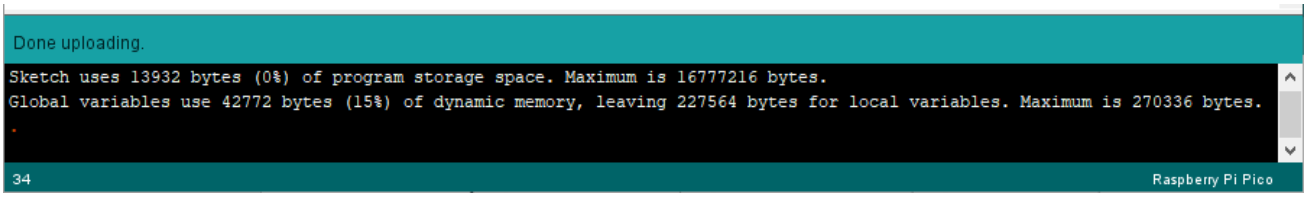
3, Click **Tools>Board>Arduino Mbed OS RP2040 Boards>Raspberry Pi Pico**.



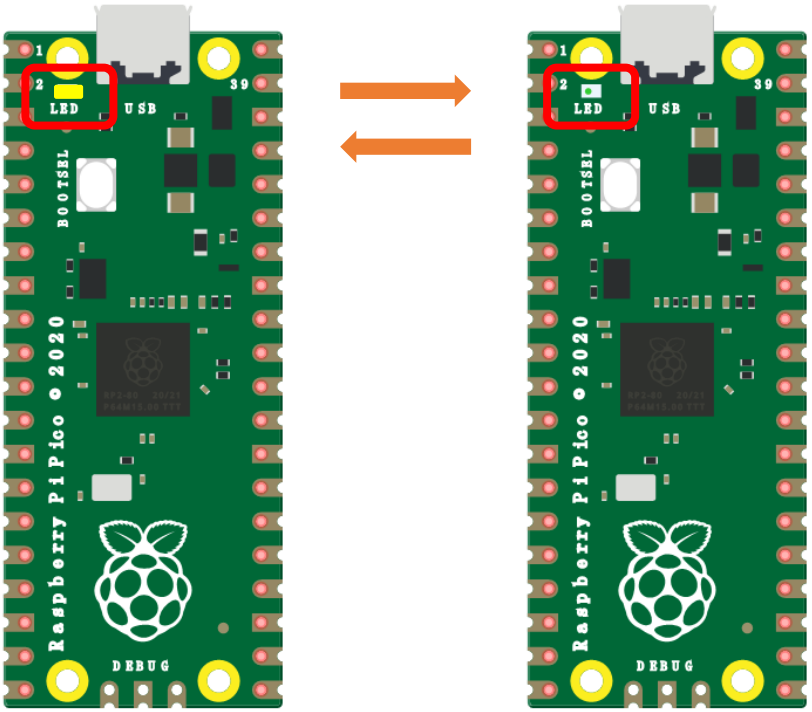
4, Upload sketch to Pico.



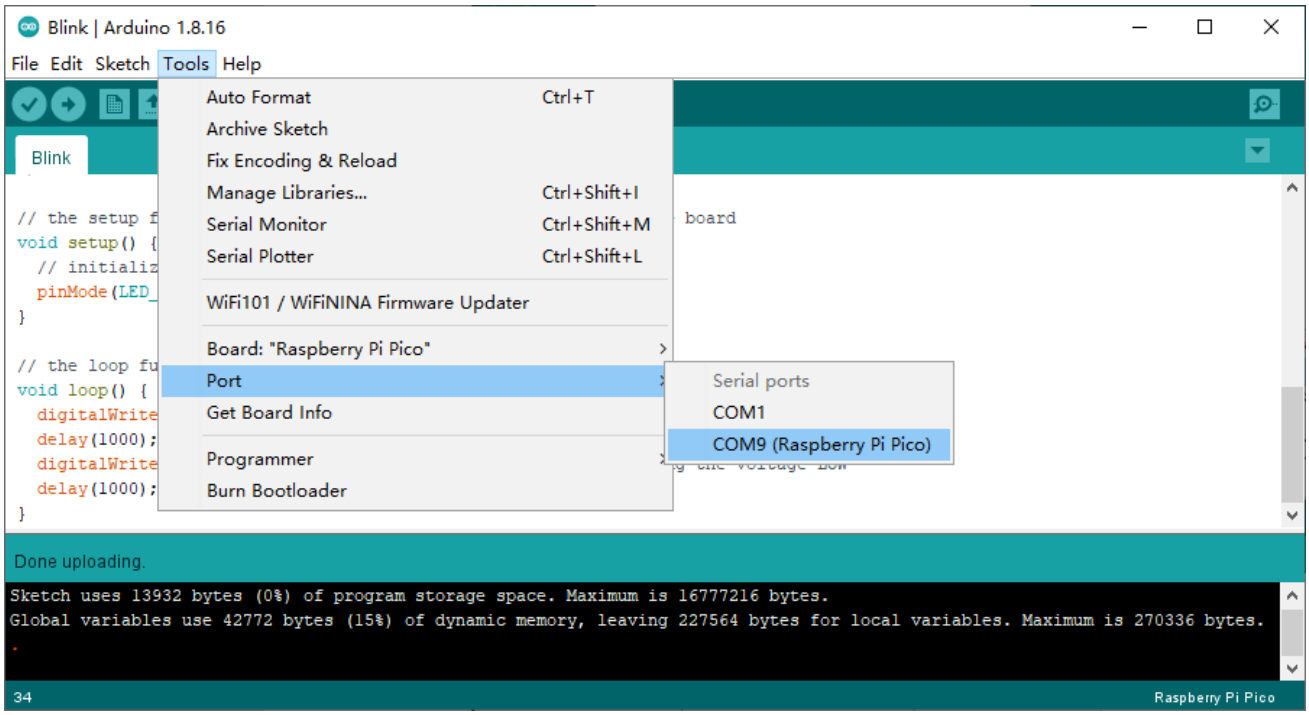
When the sketch finishes uploading, you can see the following prompt.



And 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.

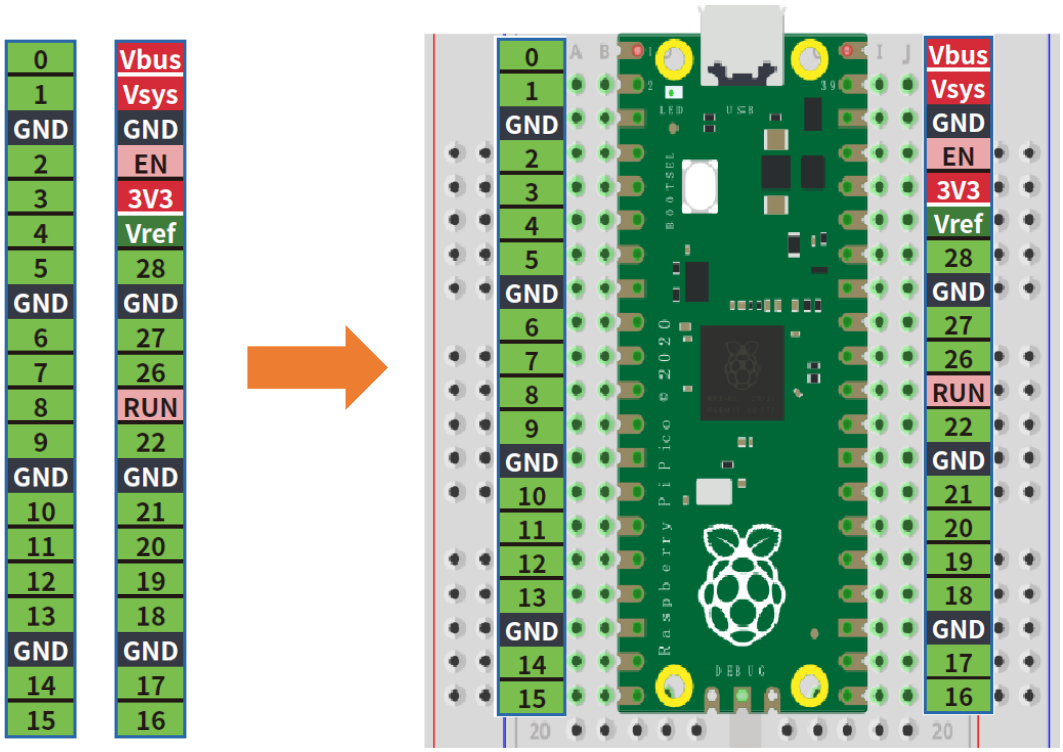


Note:

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.**

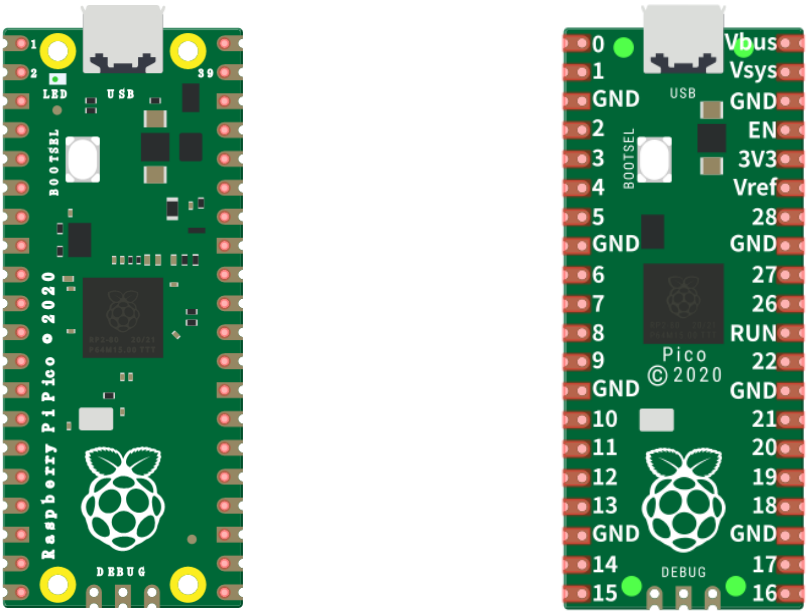
Paste the Sticker on the Breadboard

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.



Chapter 1 LED (Important)

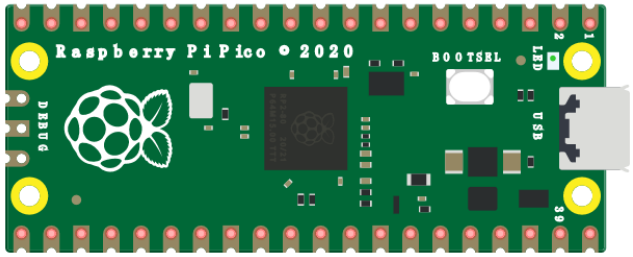
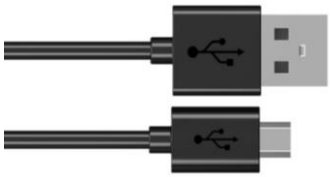
This chapter is the Start Point in the journey to build and explore Pico electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

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

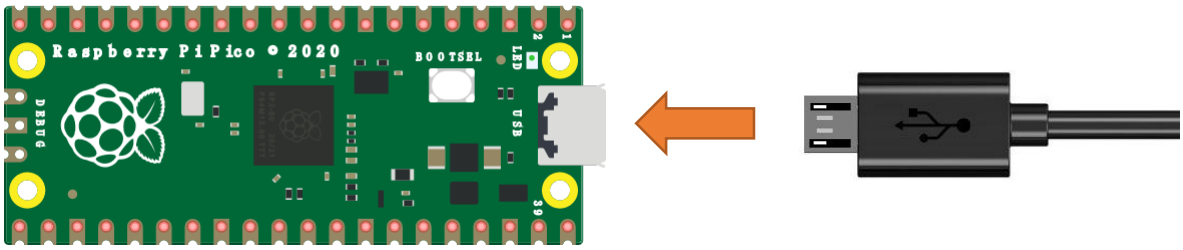
If you haven’t installed Arduino IDE, you can click Here.
If you haven’t uploaded firmware for Pico, you can click Here to upload.

Component List

Raspberry Pi Pico x1 	USB cable x1 

Power

Raspberry Pi Pico requires 5V power supply. You can either connect external 5V power supply to Vsys pin of Pico or connect a USB cable to the onboard USB base to power Pico.
In this tutorial, we use USB cable to power Pico and upload sketches.

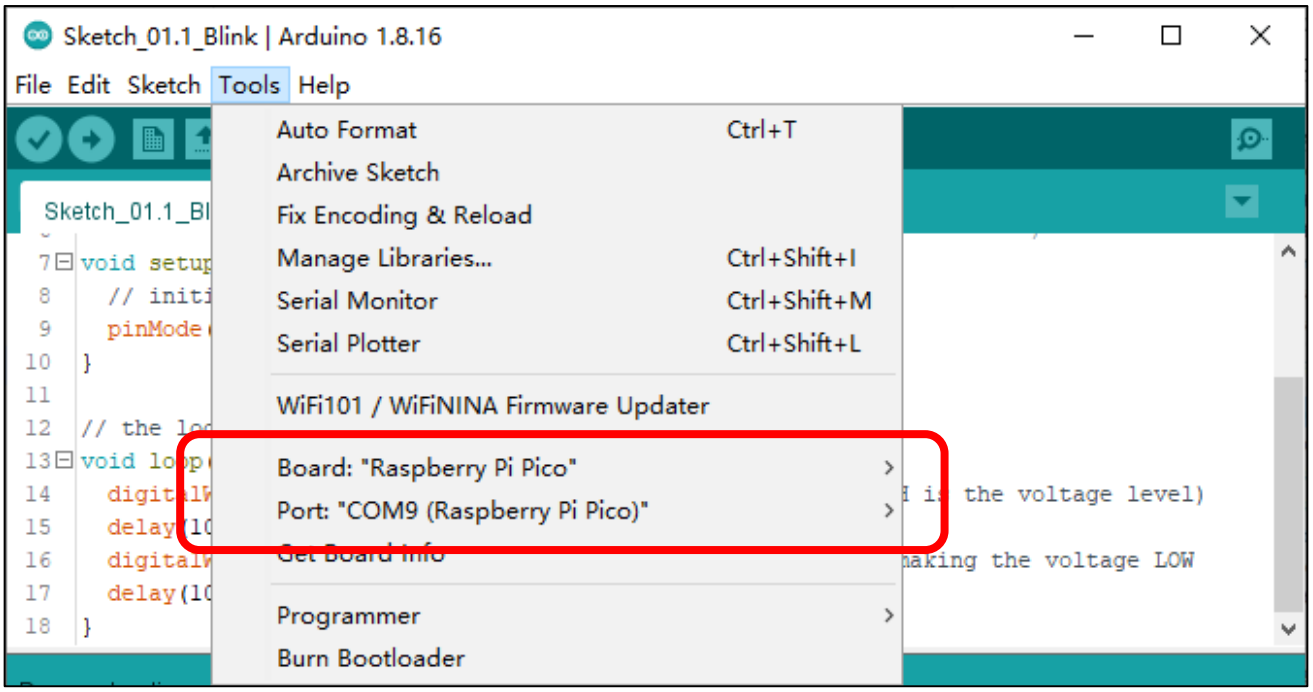


Sketch

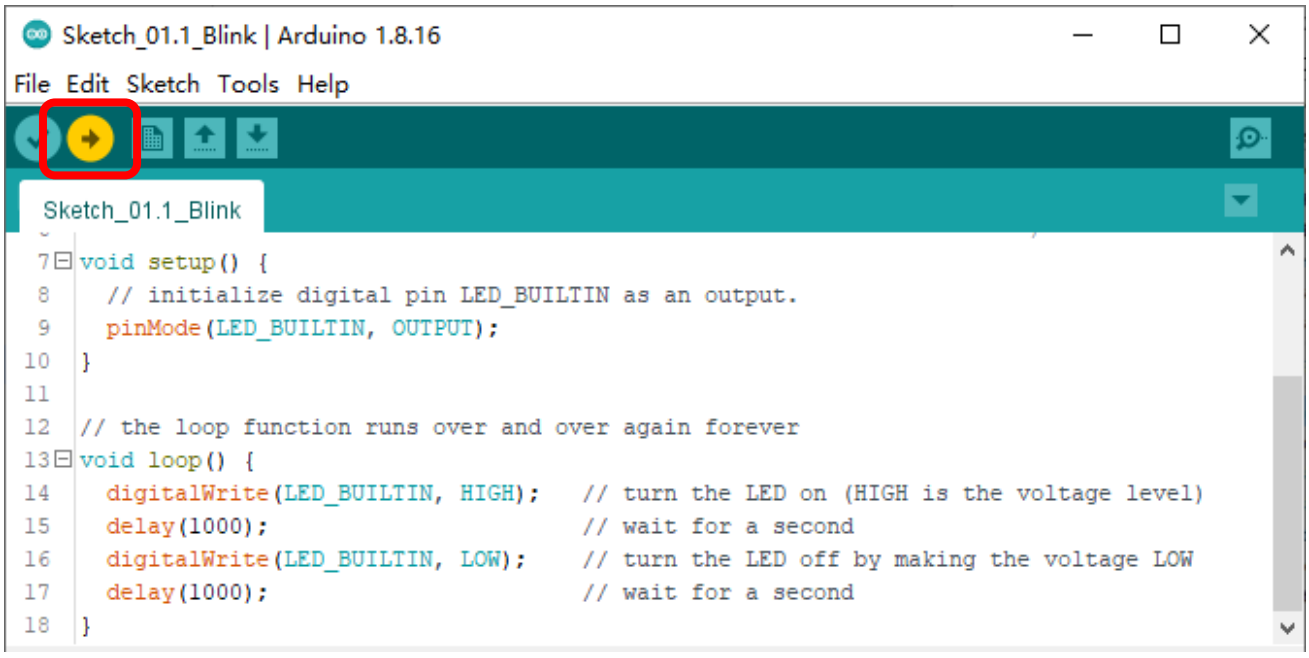
The onboard LED of Raspberry Pi Pico is controlled by GP25. When GP25 outputs high level, LED lights up; When it outputs low, LED lights off. You can open the provided code:
Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.1_Blink.

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

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

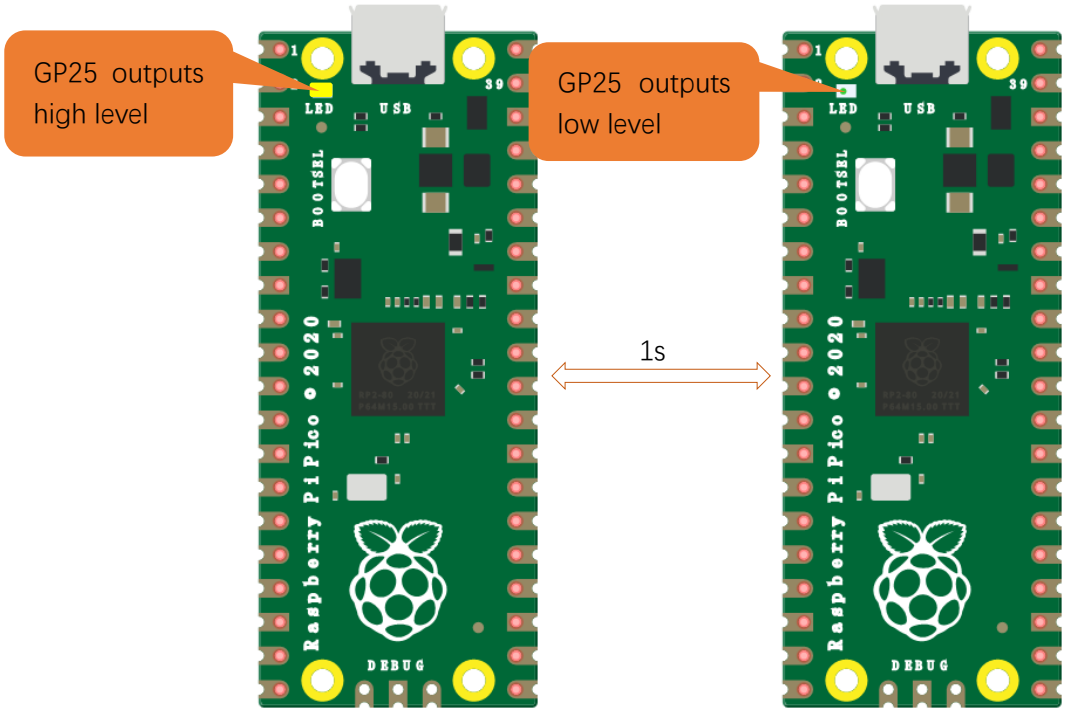


Click “Upload” to upload the sketch to Pico.



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

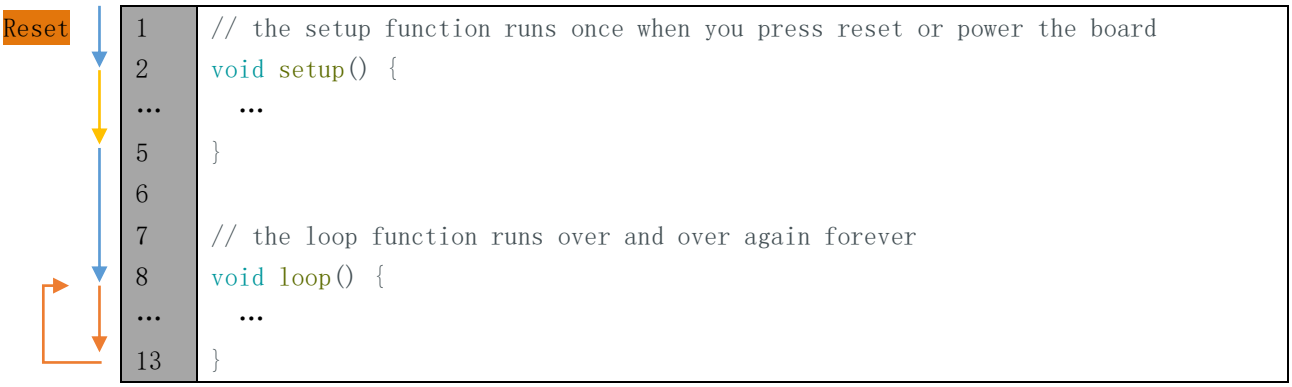
Pico's on-board LED lights on and off every 1s, flashing cyclically.



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);

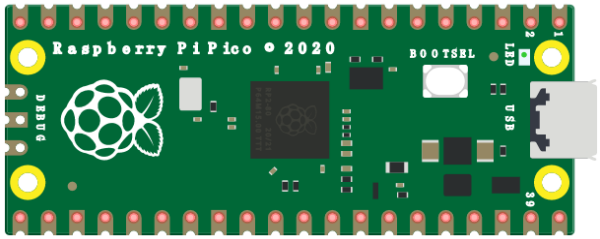
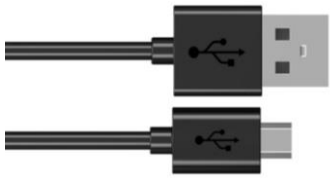
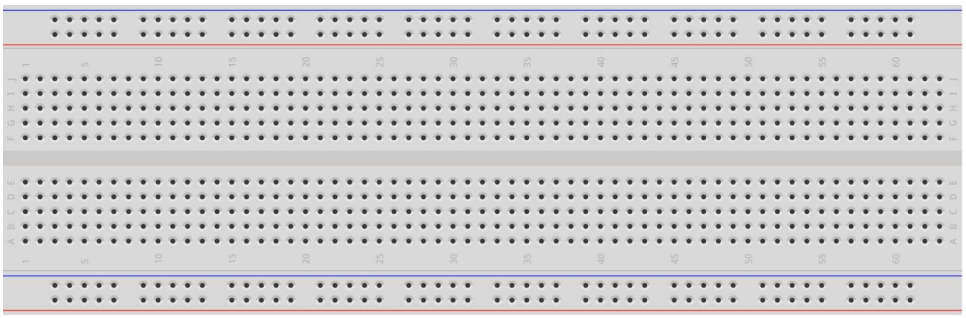
Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

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

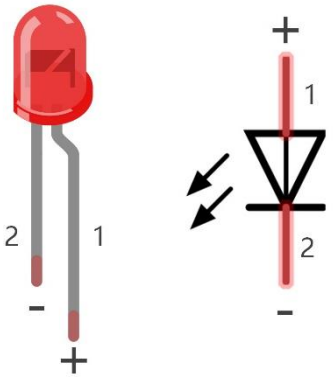
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB Cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper
  </td></tr>
</table>

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.



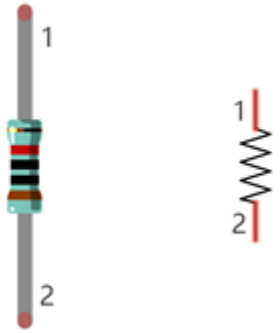
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

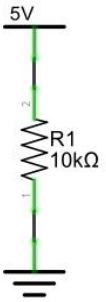
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

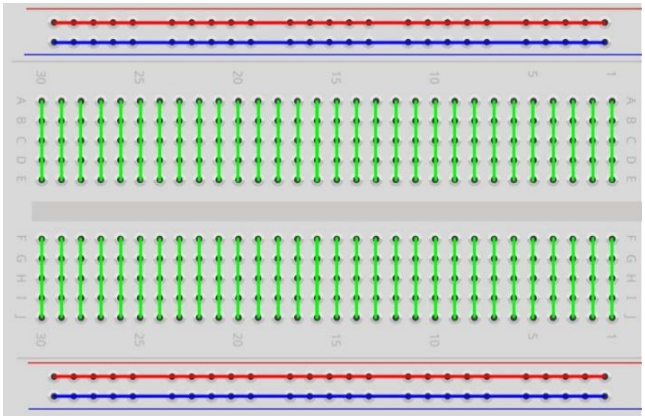


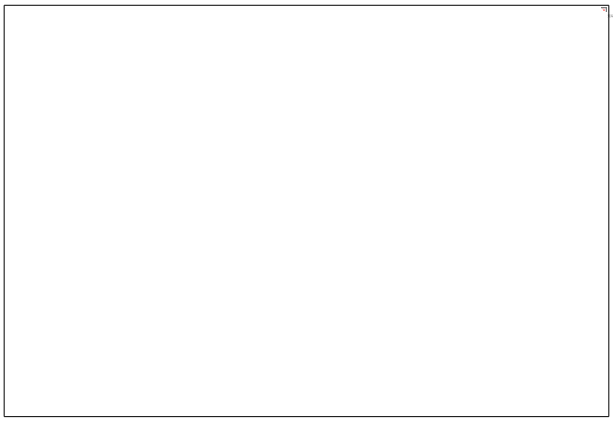
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

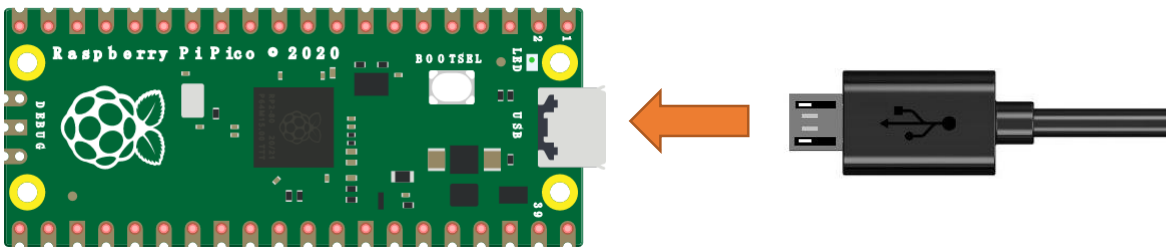
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

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

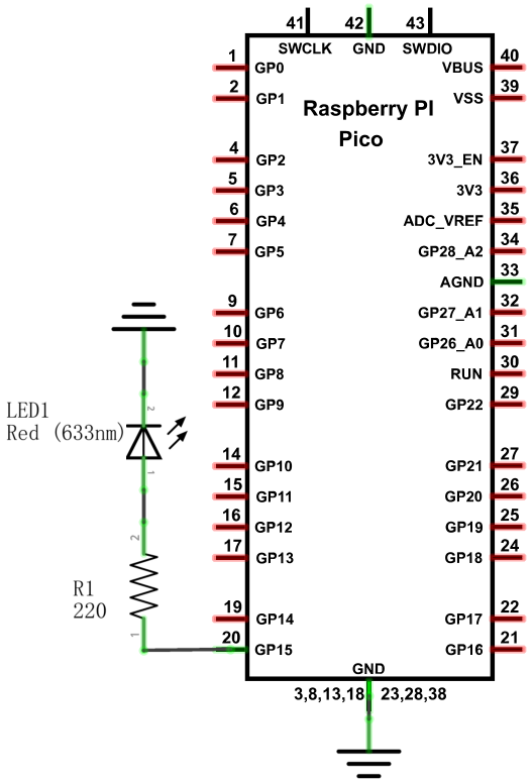


Circuit

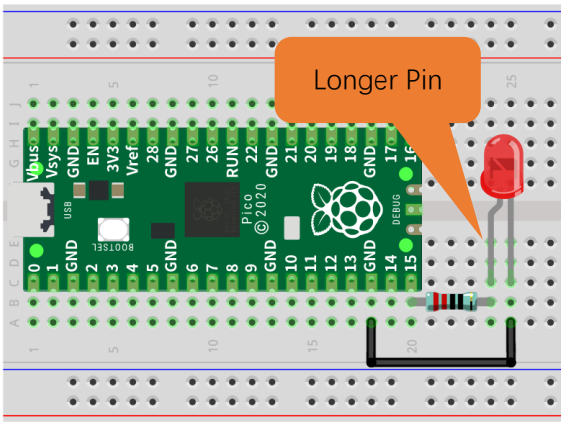
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

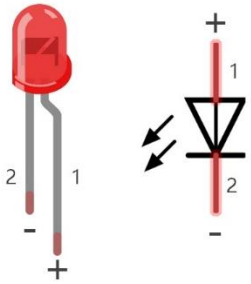
CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

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

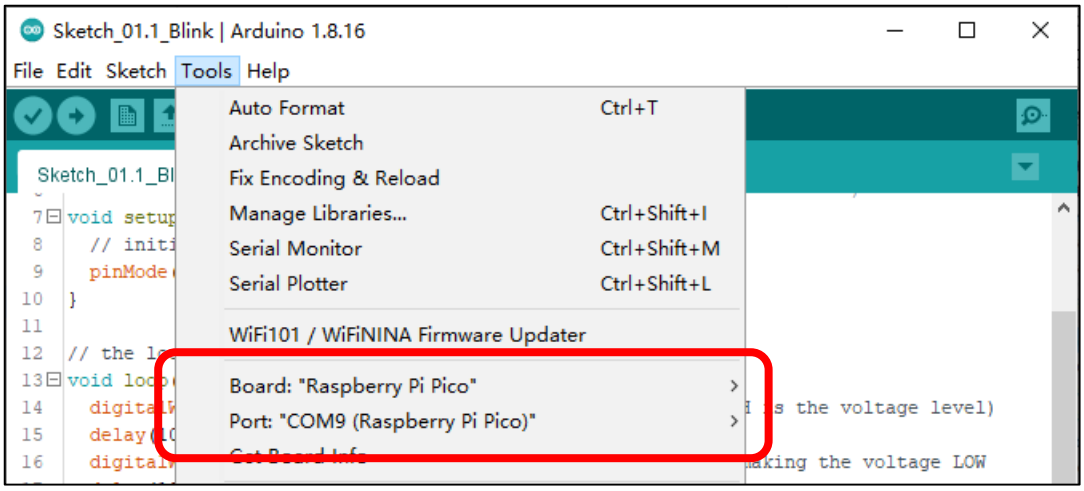
According to the circuit diagram, when GP15 of Pico outputs high level, LED lights up; when it outputs low, LED lights off. Therefore, we can make LED flash repeatedly by controlling GP15 to output high and low repeatedly.

You can open the provided code:

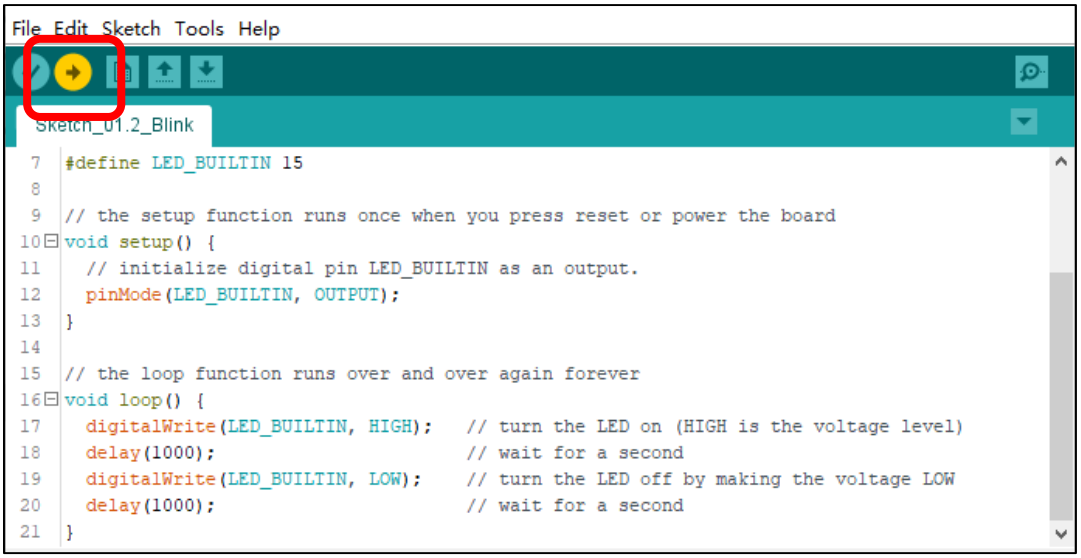
Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.2_Blink.

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

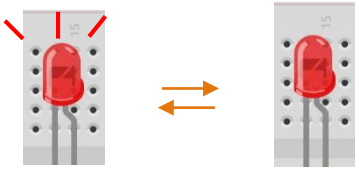
Click Tools, make sure Board and Port are as follows:



Click "Upload" to upload the sketch to Pico.



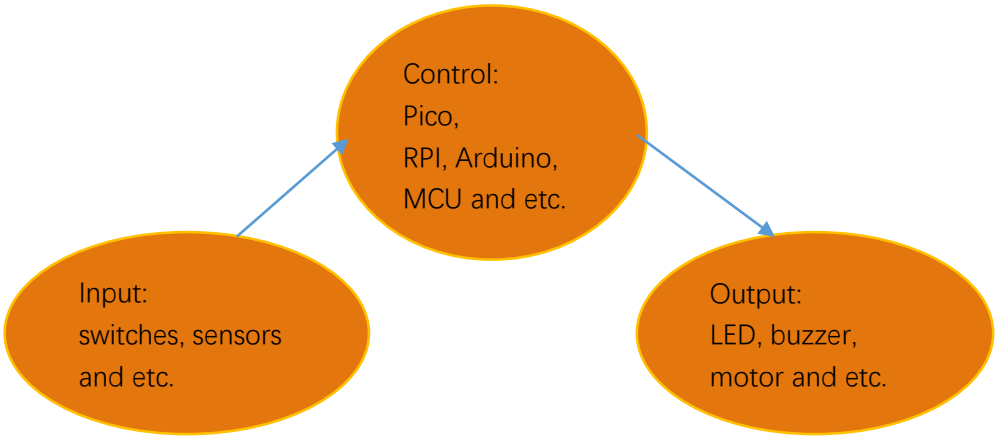
Click "Upload". Download the code to Pico and your LED in the circuit starts Blink.



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.

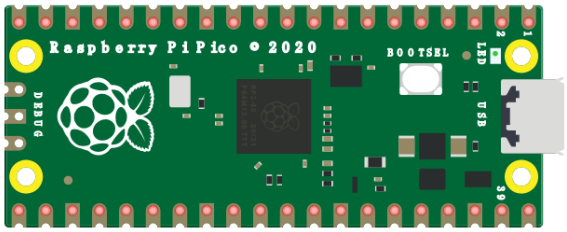
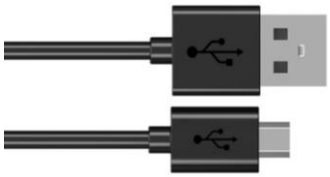
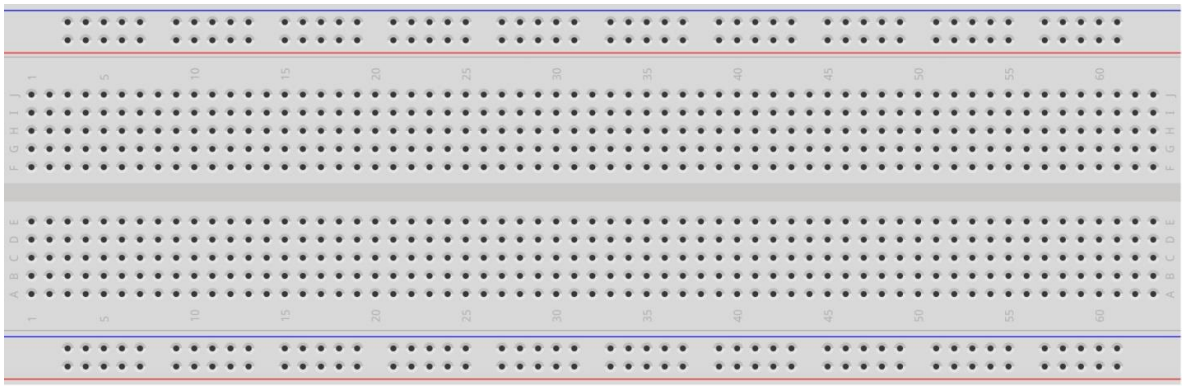


Next we make a simple project: build a control system with button, LED and Raspberry Pi Pico.
Input: Button
Control: Raspberry Pi Pico
Output: LED

Project 2.1 Button & LED

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.

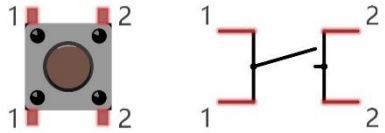
Component List

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

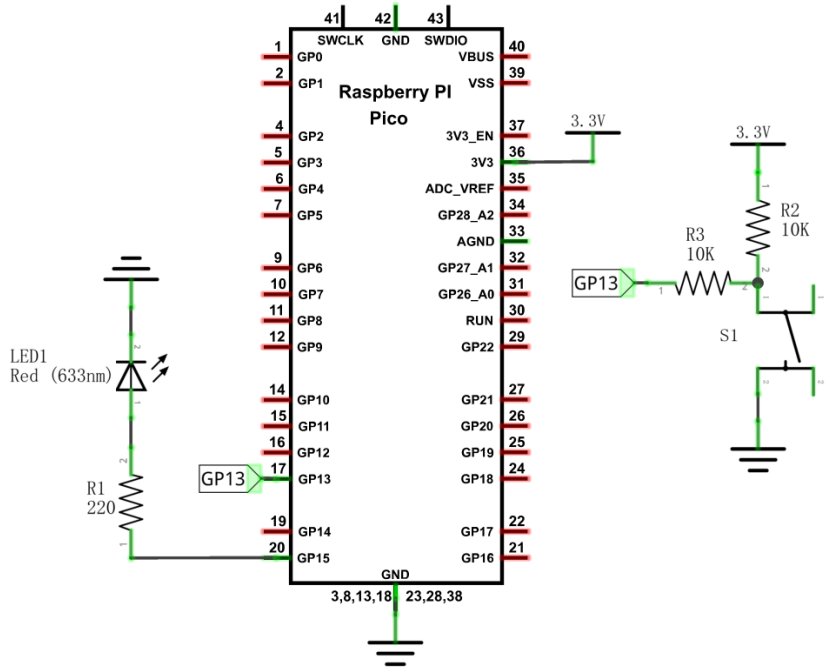
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



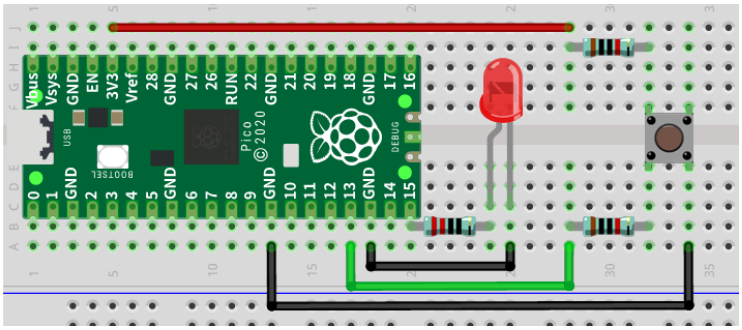
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

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_Basic_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.1_ButtonAndLed.
Sketch_02.1_ButtonAndLed

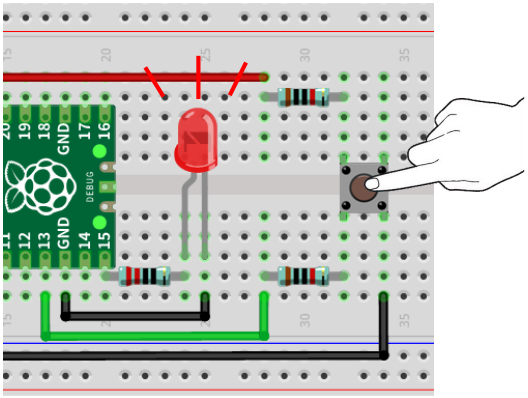
Sketch_02.1_ButtonAndLed | Arduino 1.8.12

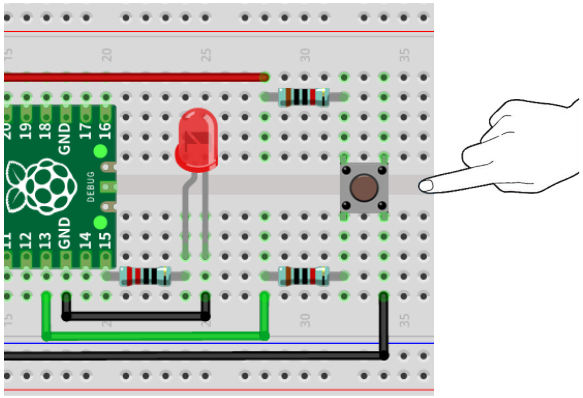
File Edit Sketch Tools Help

Sketch_02.1_ButtonAndLed

```
#define PIN_LED    15
#define PIN_BUTTON 13

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);
  }
}
```

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);

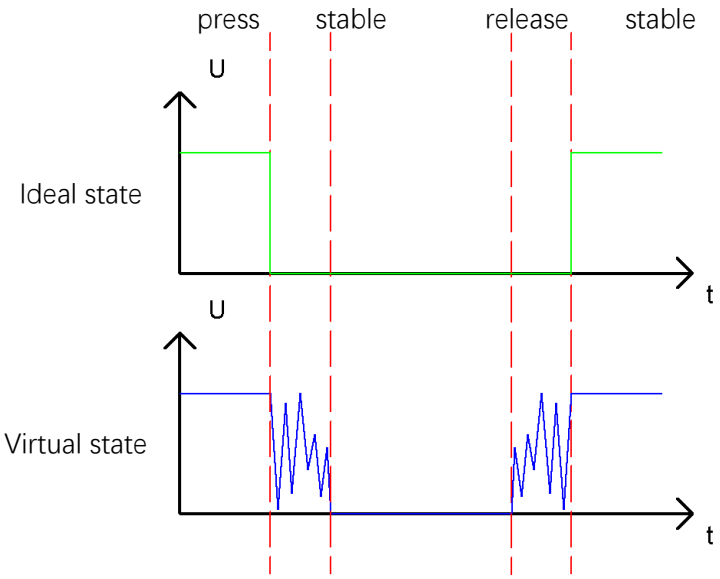
This function returns the value read at the given pin. It will be “HIGH” or “LOW”(1 or 0) depending on the logic level at the pin.

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_Basic_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.2_TableLamp.
Sketch_02.2_TableLamp

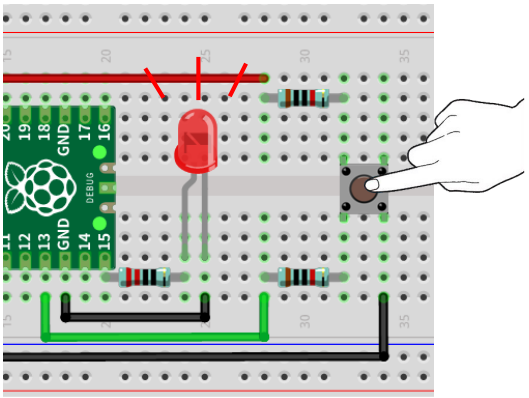
Sketch_02.2_TableLamp | Arduino 1.8.12

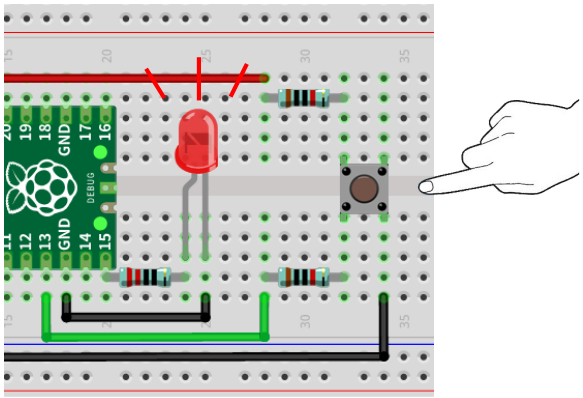
File Edit Sketch Tools Help

Sketch_02.2_TableLamp

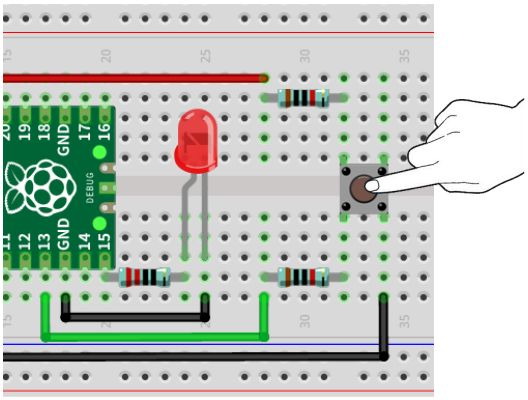
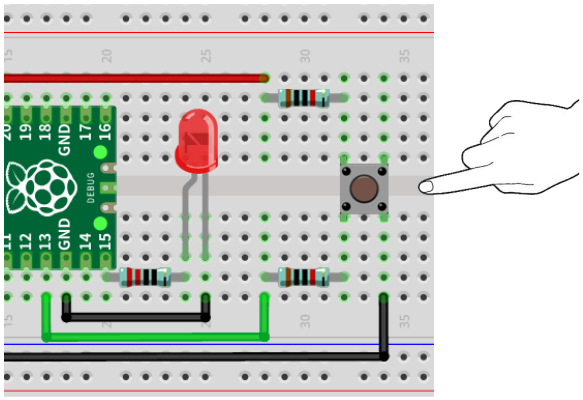
```
#define PIN_LED    15
#define PIN_BUTTON 13
bool led_state = false;

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  }
```

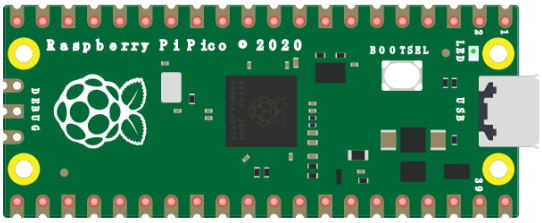
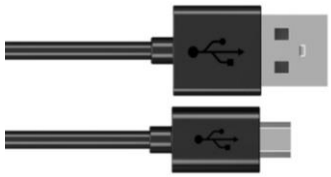
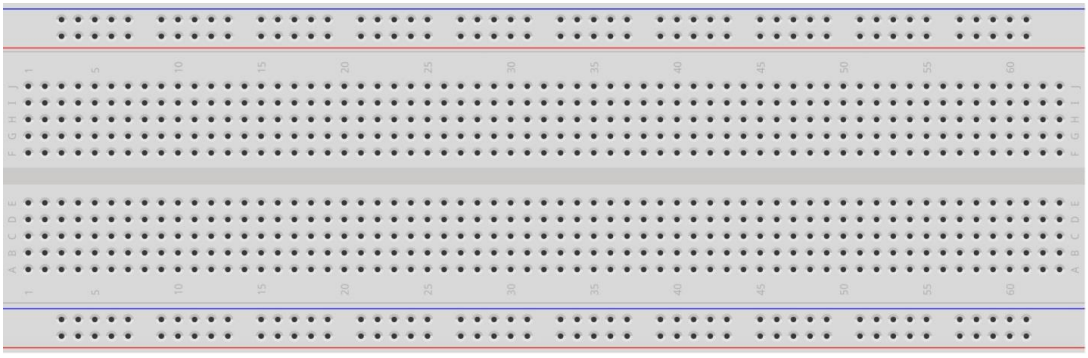
Chapter 3 LED Bar

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

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

Component List

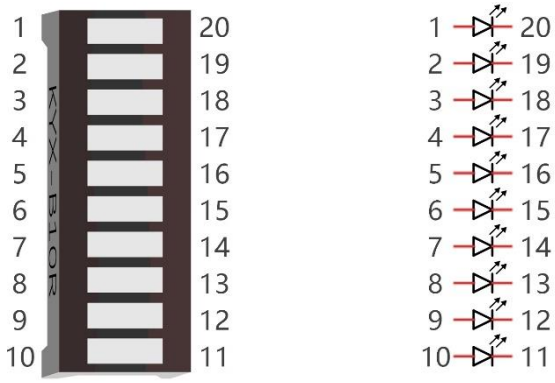
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Jumper 	LED bar graph x1 	Resistor 220Ω x10 

Component Knowledge

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

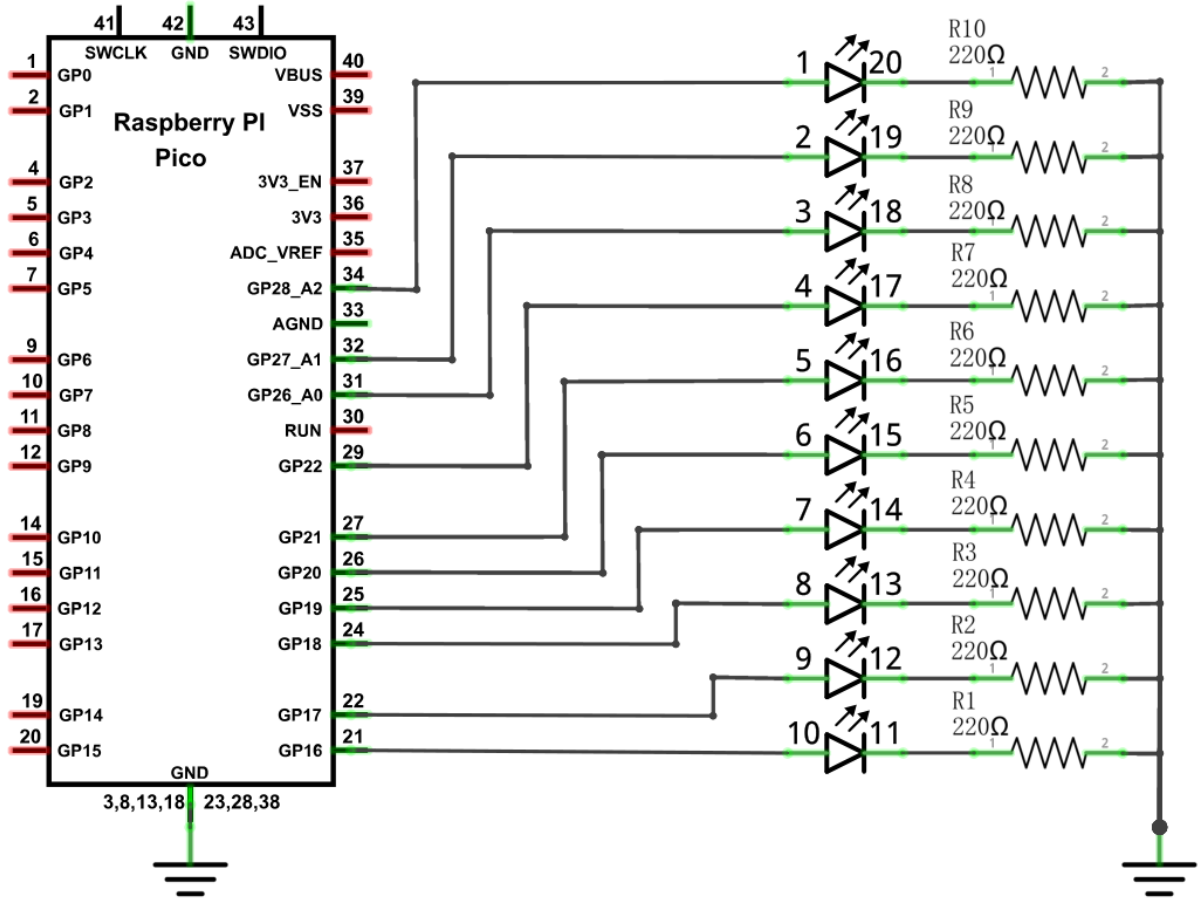
LED bar

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

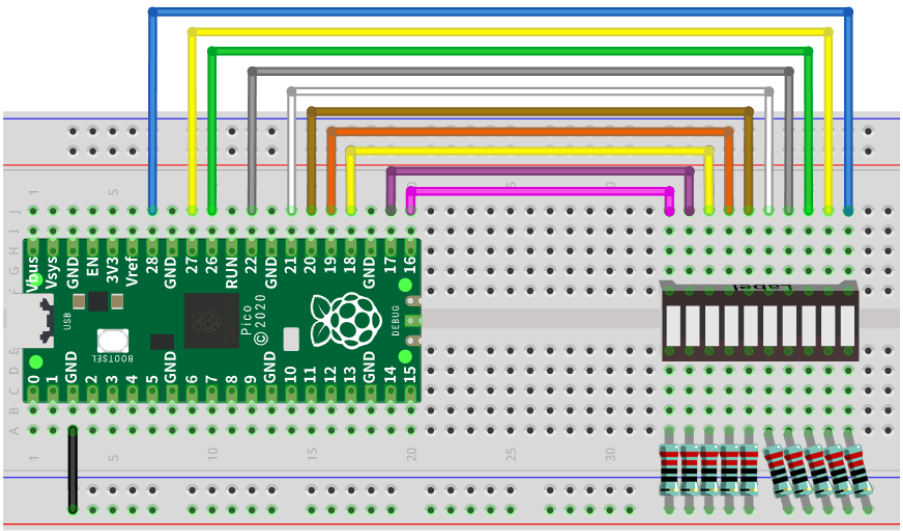


Circuit

Schematic diagram



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



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

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

Any concerns? ✉ support@freenove.com

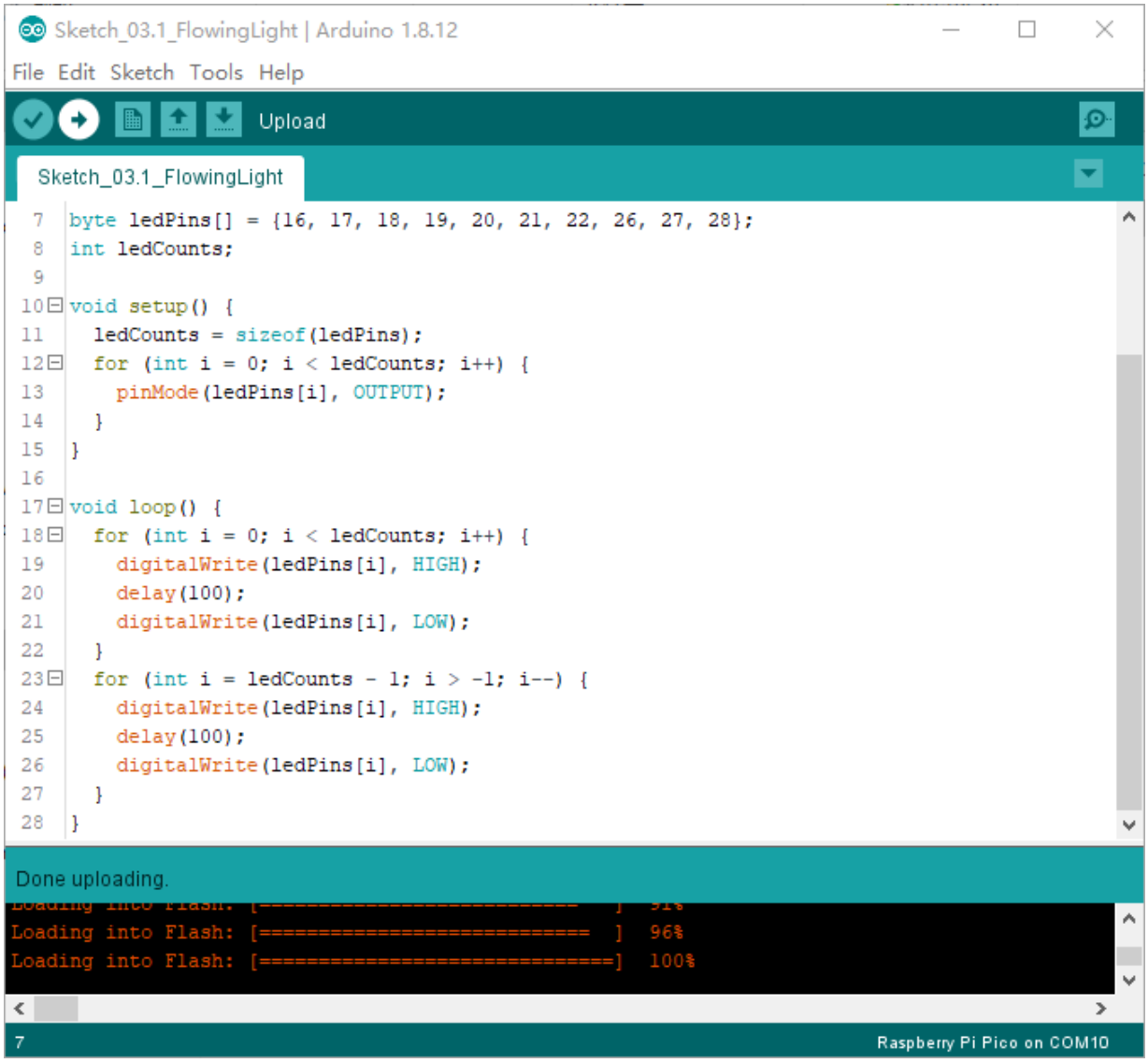
Sketch

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_Basic_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.

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);
  }
```

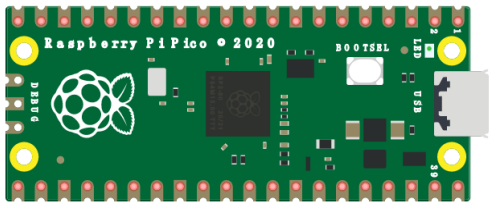
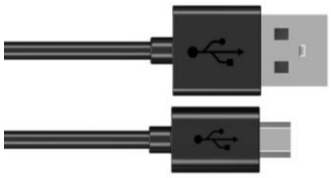
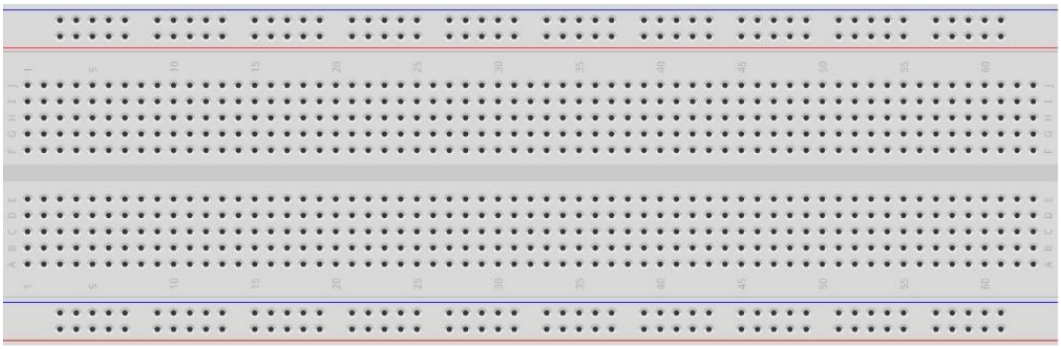
Chapter 4 Analog & PWM

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.

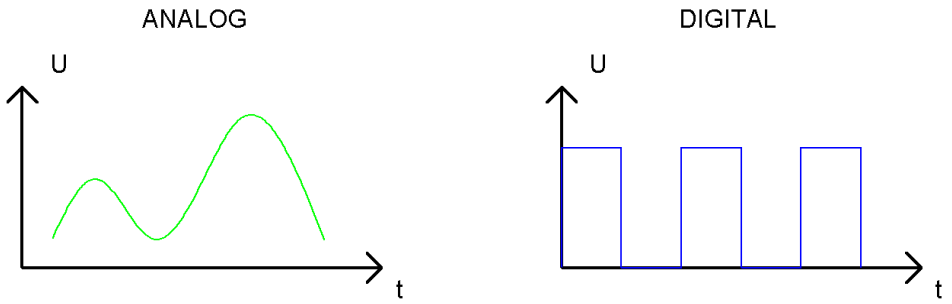
Component List

<table>
<tr><td>Raspberry Pi Pico x1
  </td><td colspan="2">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.



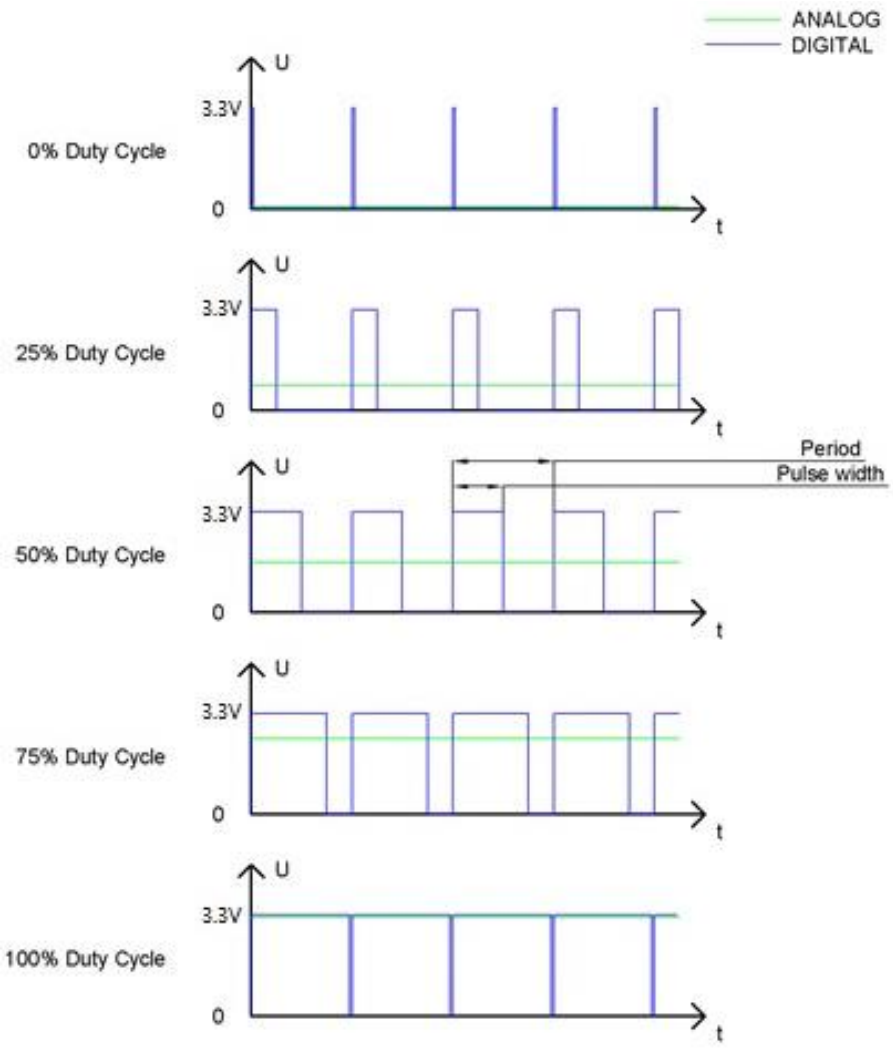
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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.

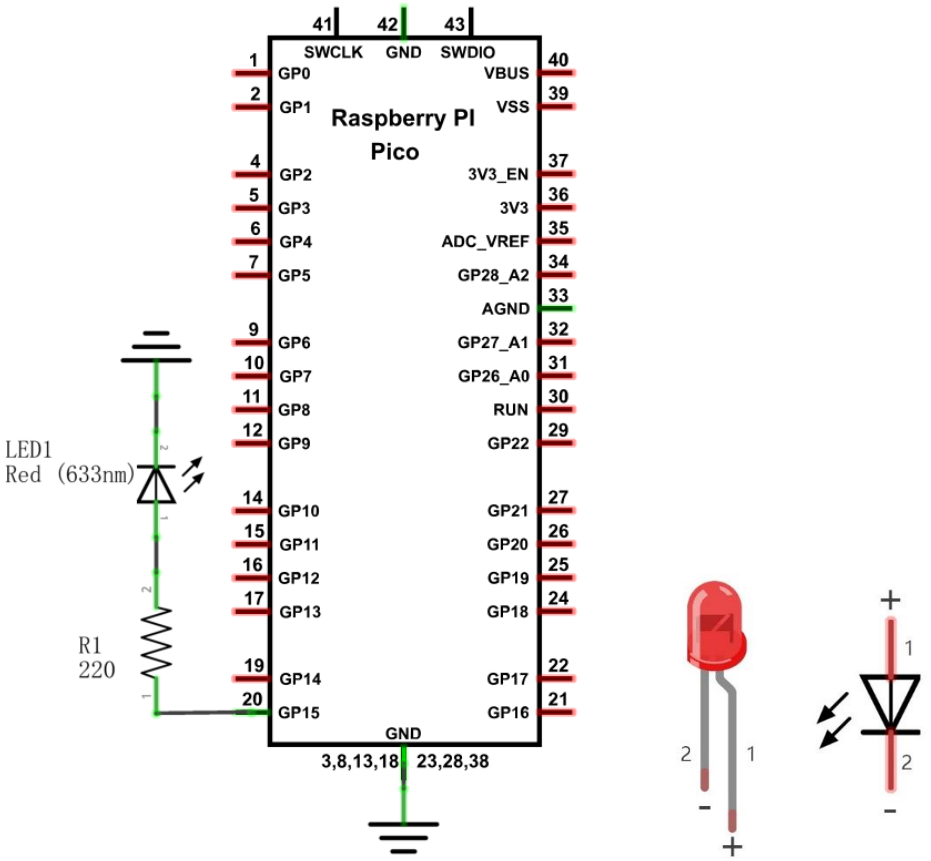
Raspberry Pi Pico and PWM

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.

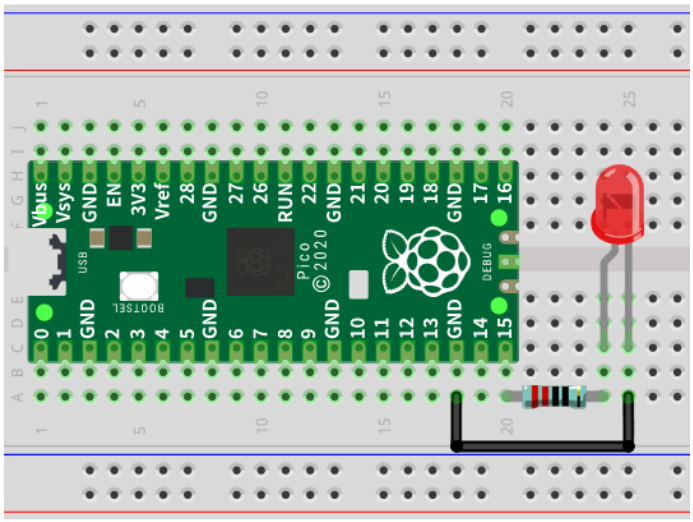
Circuit

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

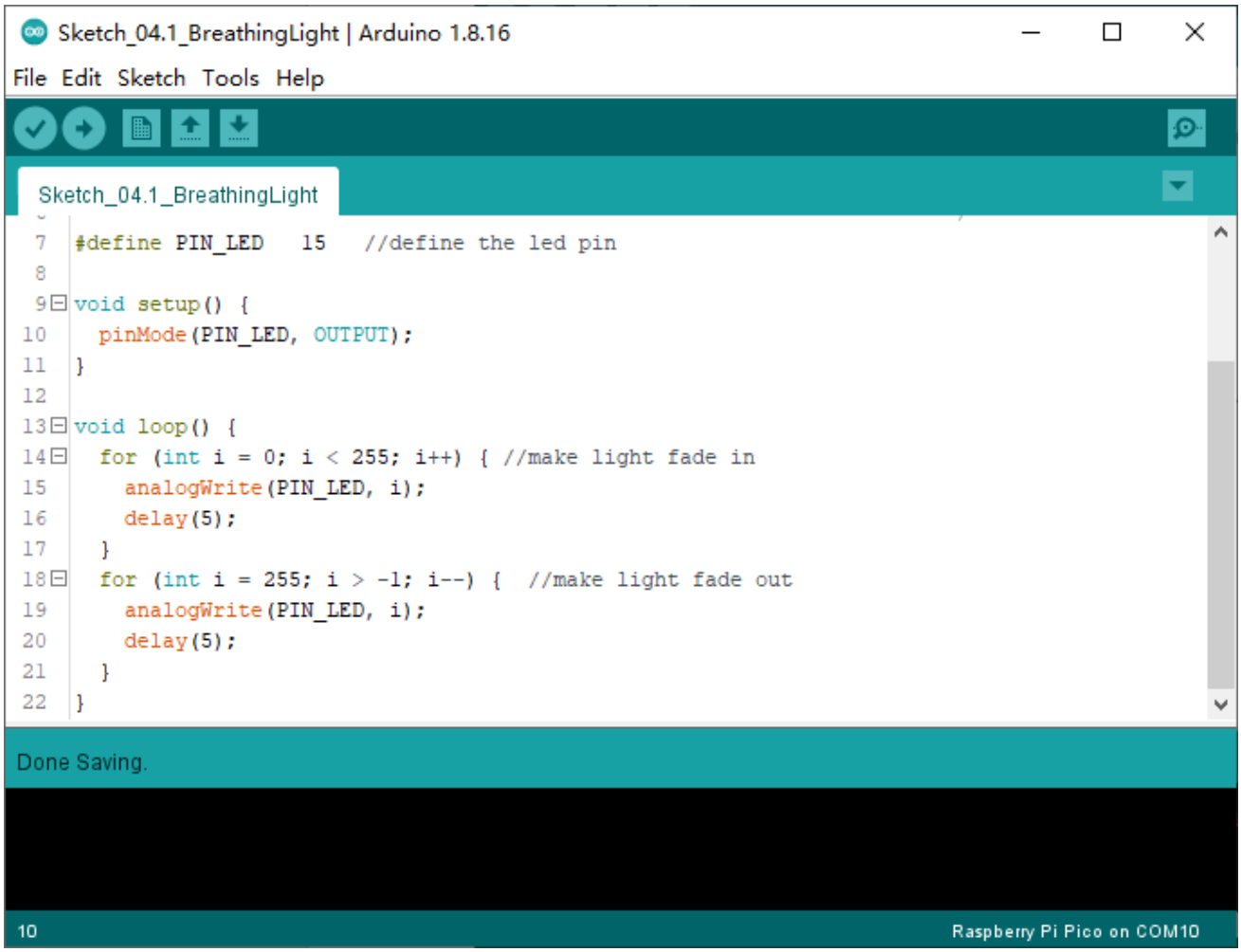


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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)

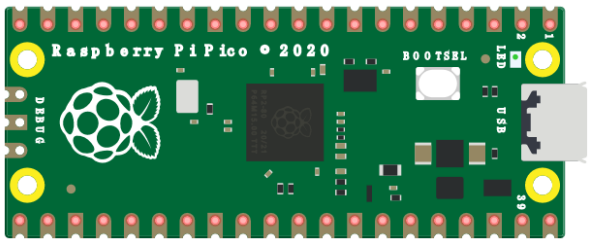
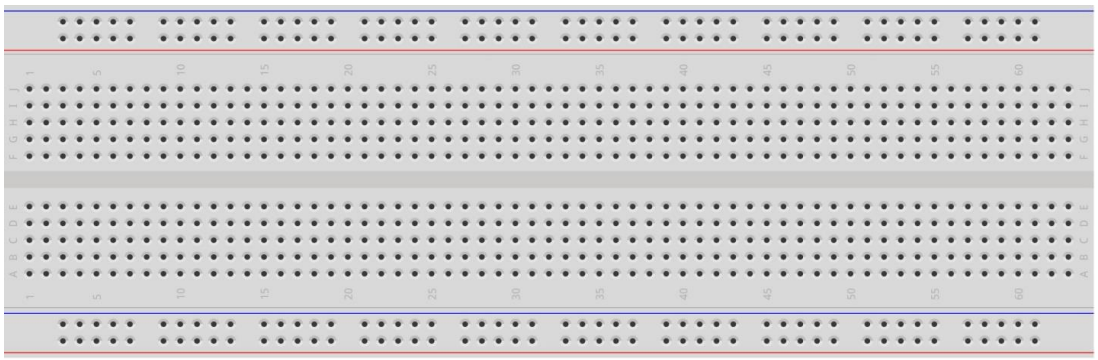
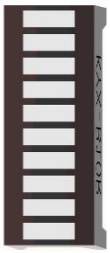
Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255,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

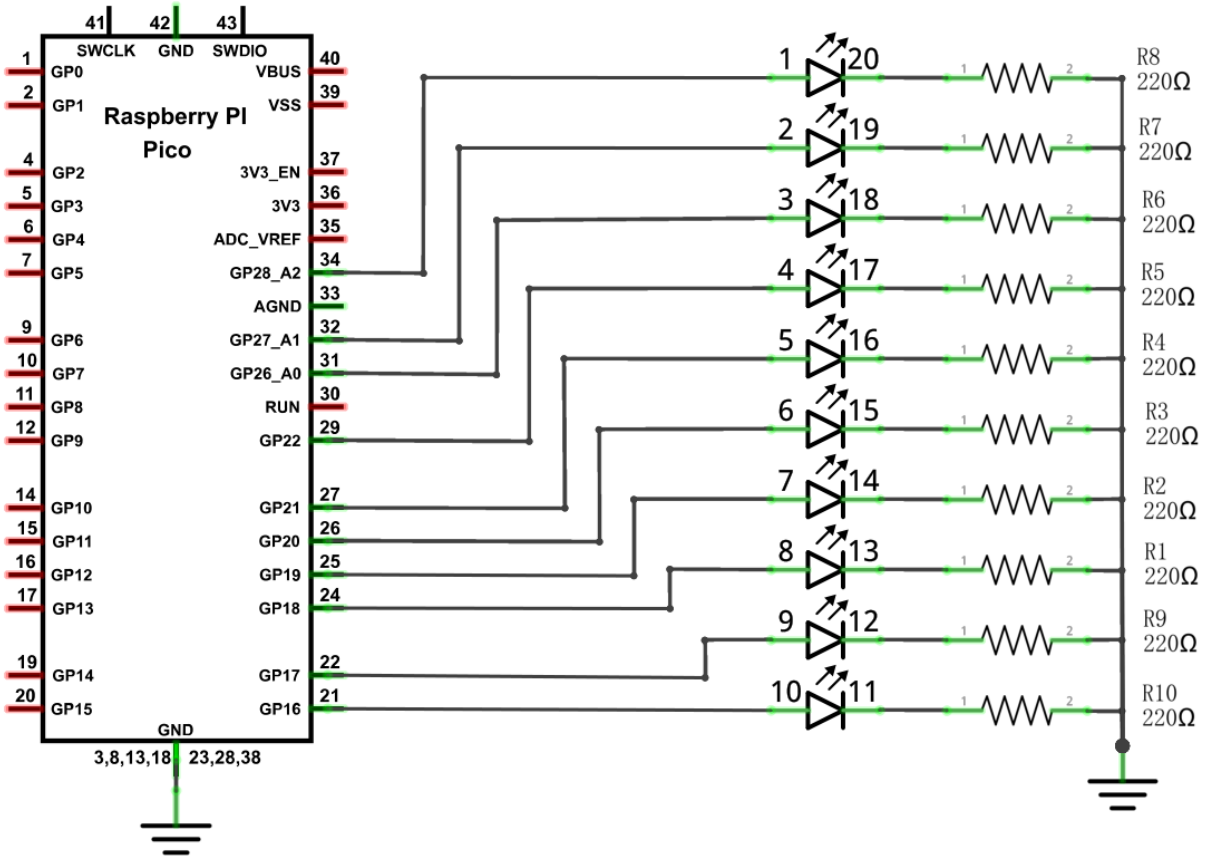
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light.
The component list, circuit, and hardware are exactly consistent with the project Flowing Light.

Component List

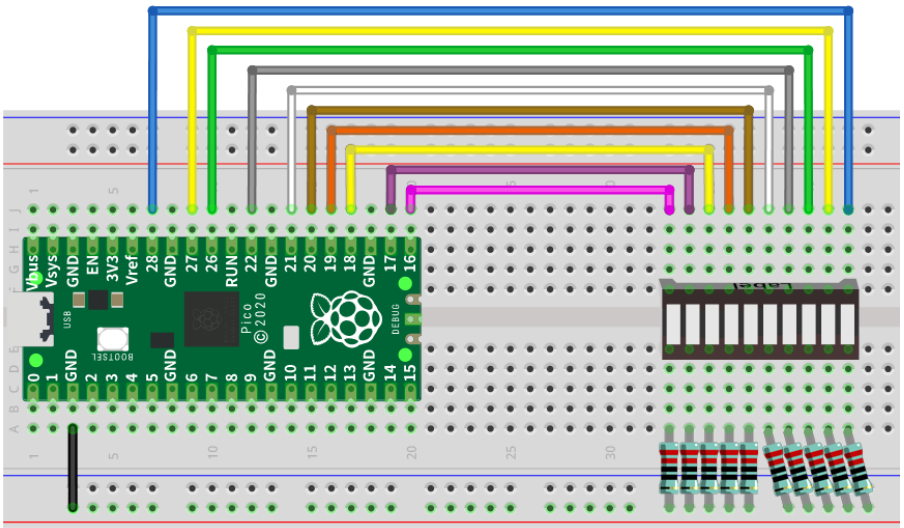
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Jumper 	LED bar graph x1 	Resistor 220Ω x10 

Circuit

Schematic diagram



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



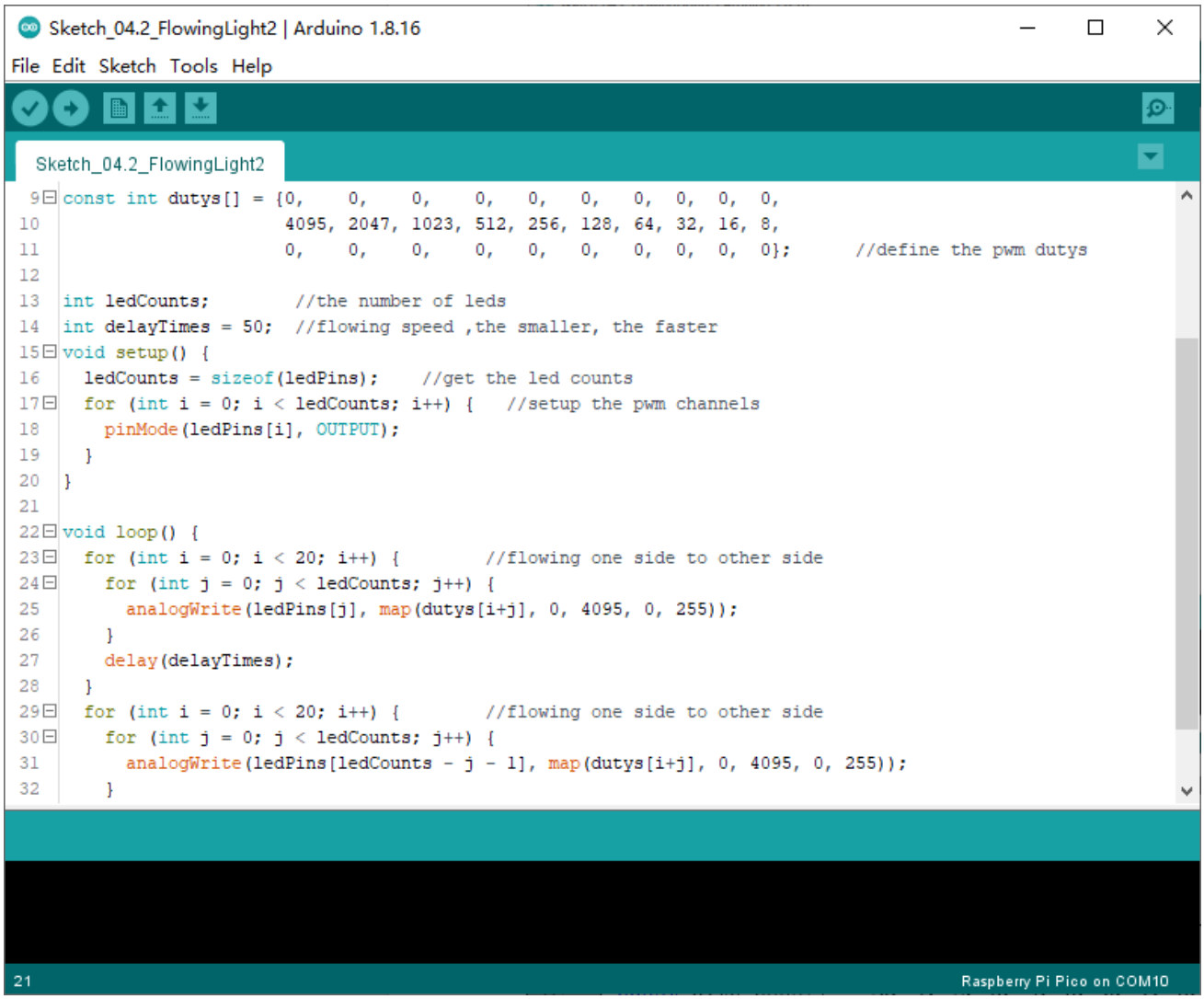
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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
duties 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.

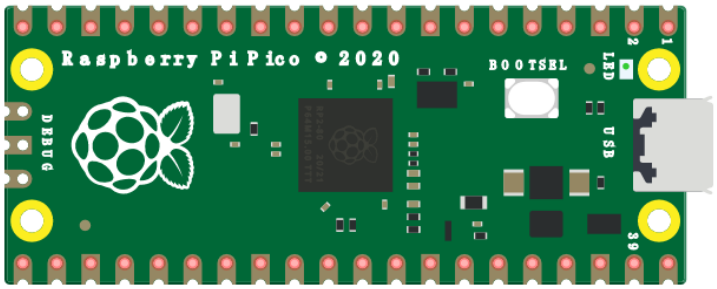
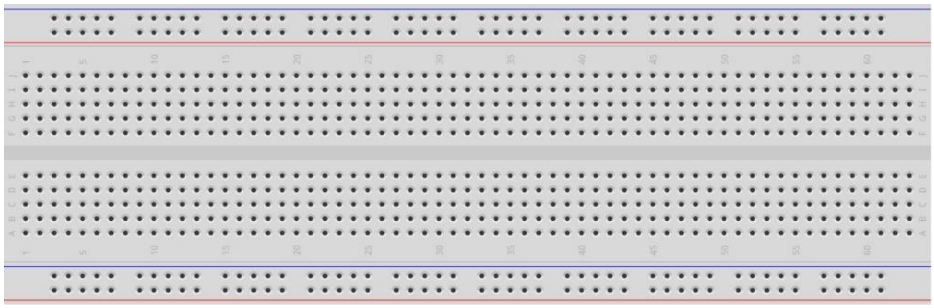
Chapter 5 RGBLED

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

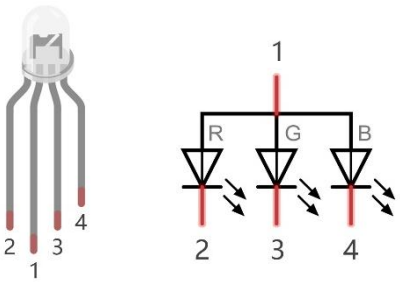
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

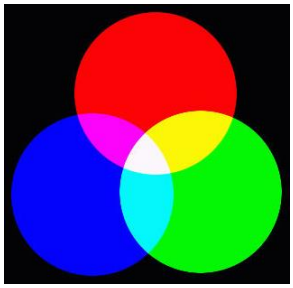
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>RGBLED x1
  </td><td>Resistor 220Ω x3
  </td><td>Jumper
  </td></tr>
</table>

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.

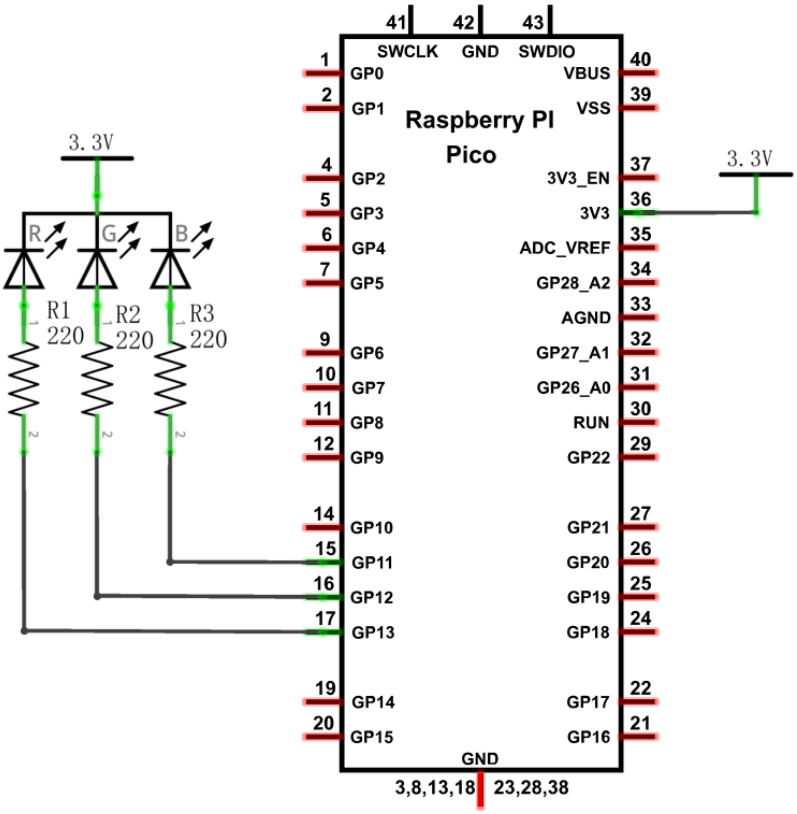


RGB

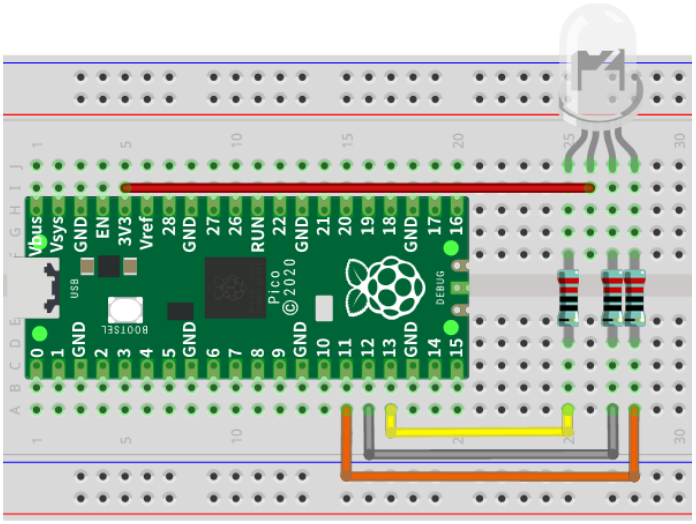
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

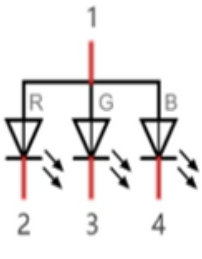
Circuit

Schematic diagram



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

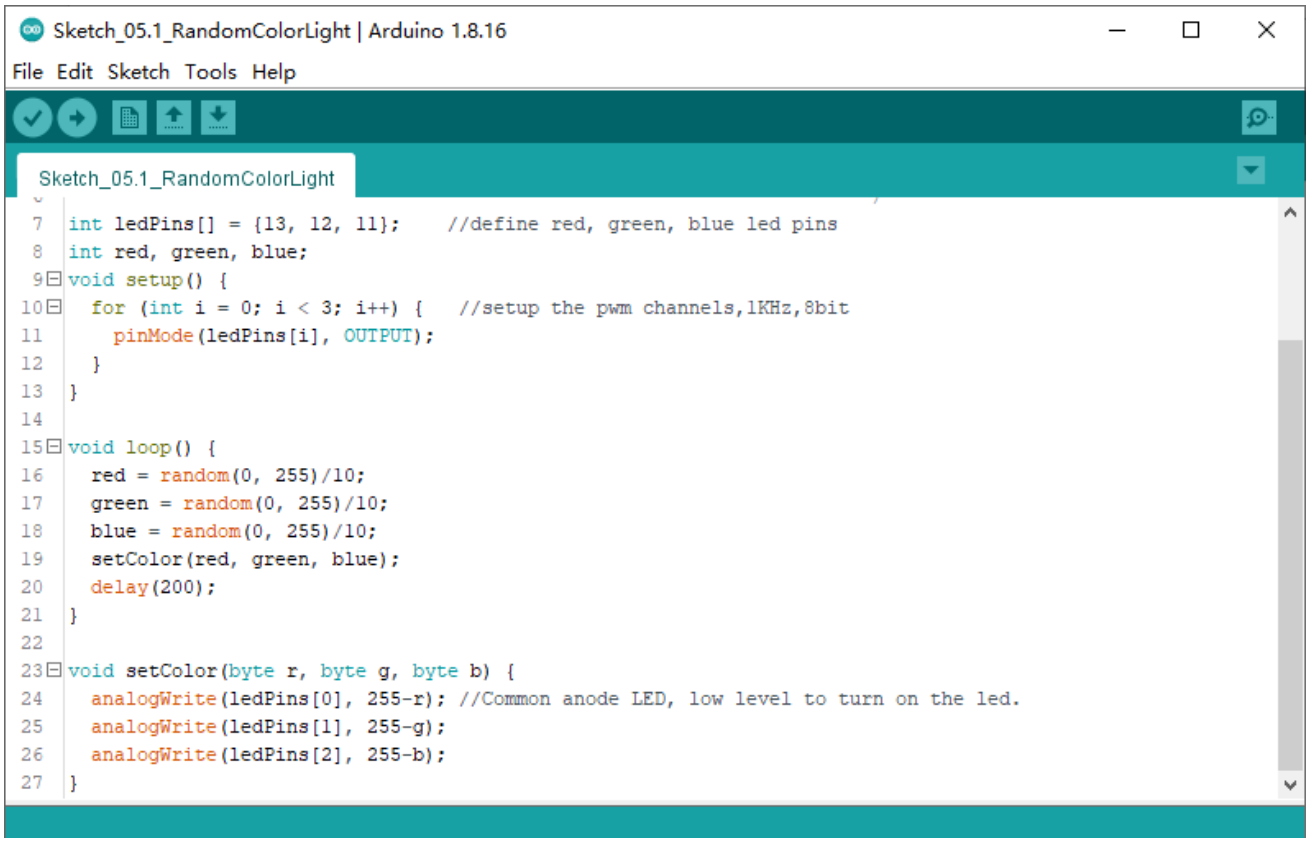




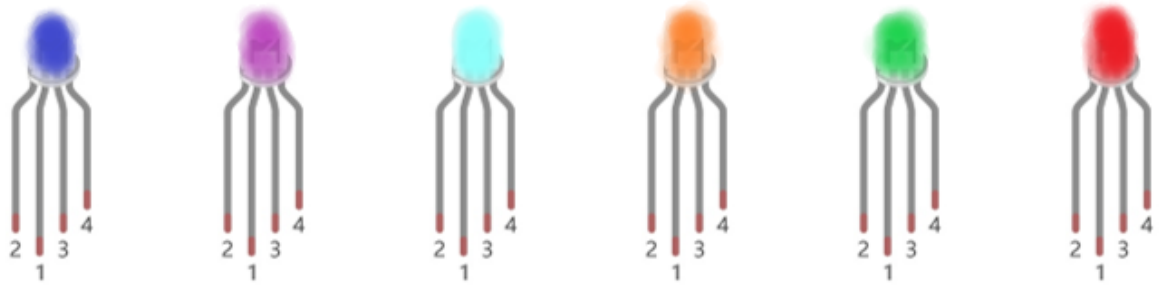
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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));
  }
}
```

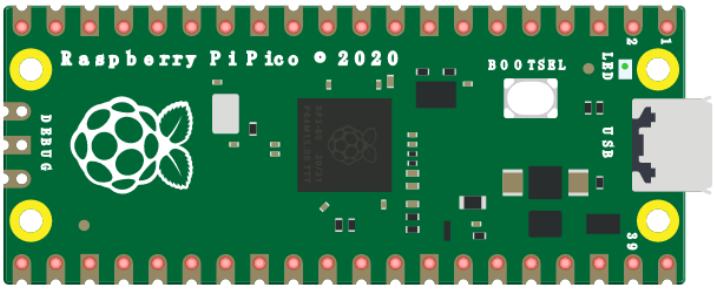
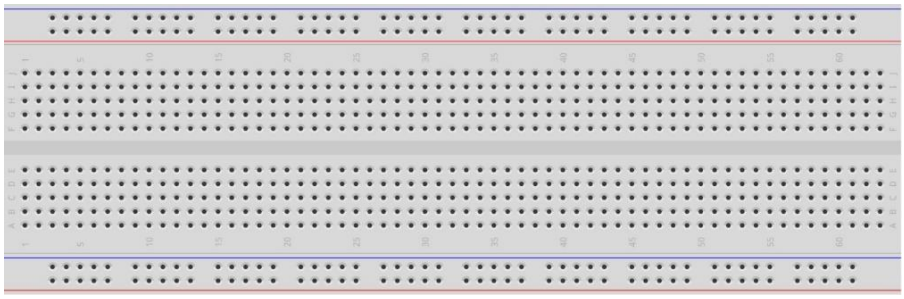
Chapter 6 Buzzer

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

Project 6.1 Doorbell

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

Component List

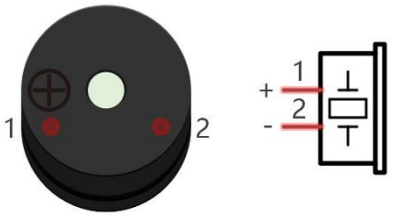
Raspberry Pi Pico x1 		USB cable x1 		
Breadboard x1 				
Jumper 				
NPN transistorx1 (S8050) 	Active buzzer x1 	Push button x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component Knowledge

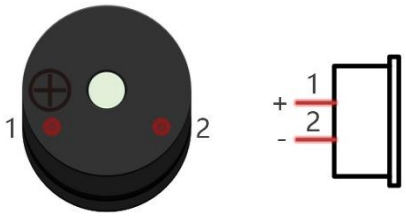
Buzzer

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

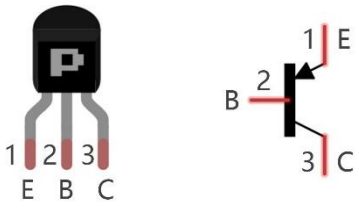
Transistor

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.

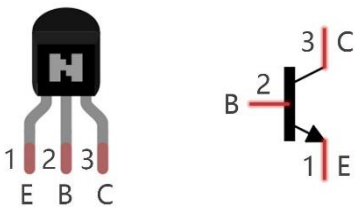
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

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

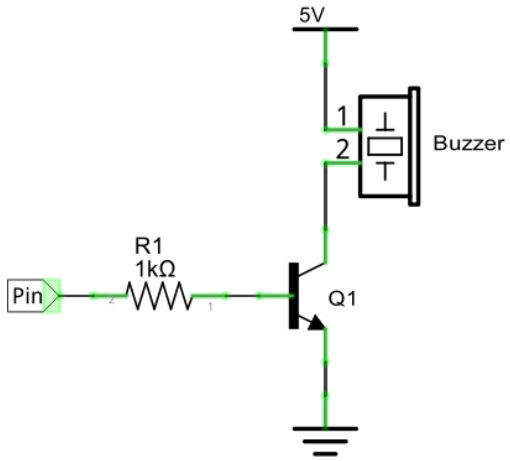
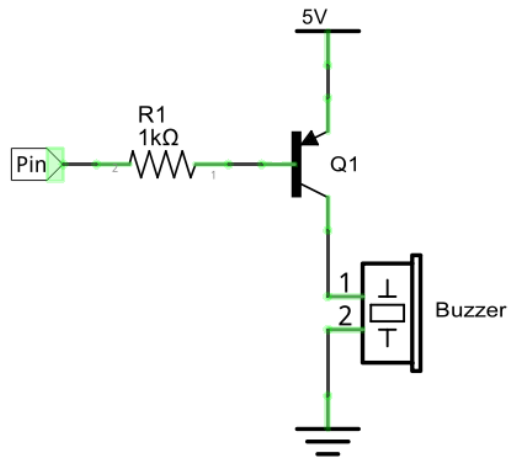


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

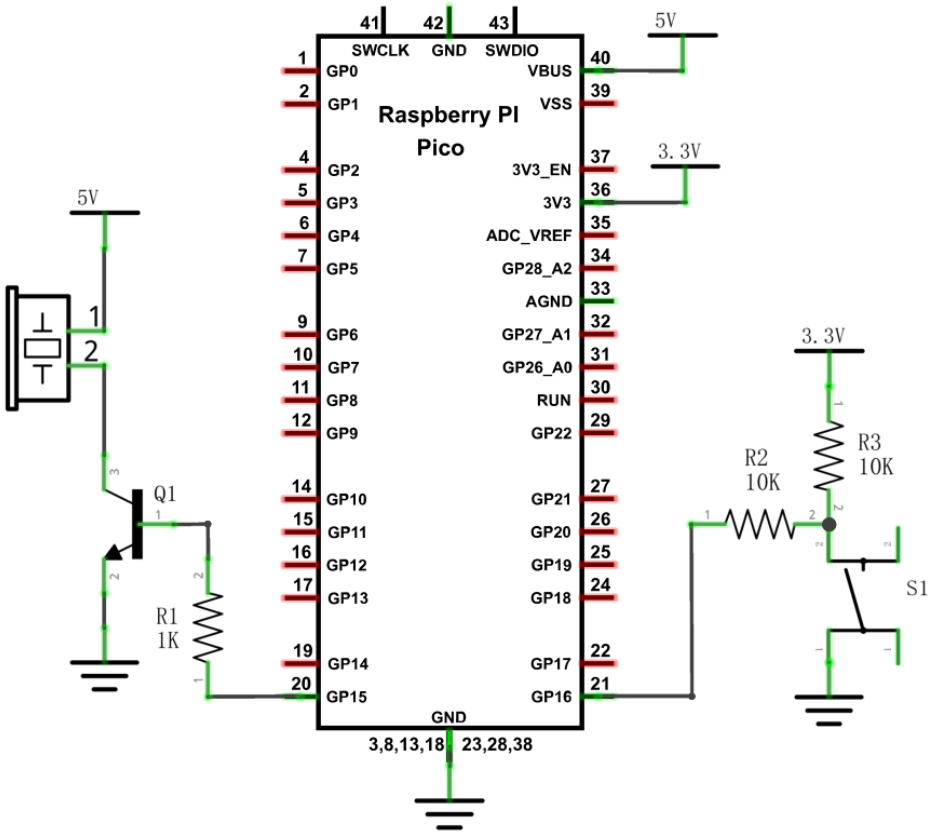
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

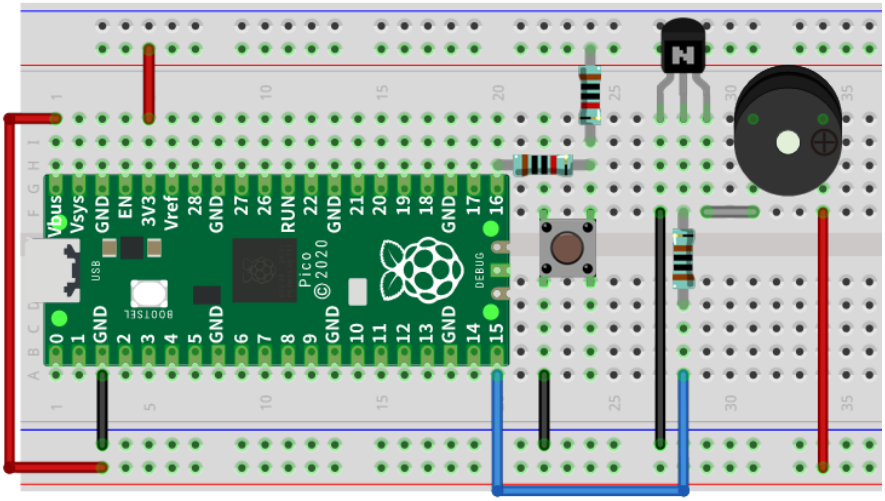
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

Circuit

Schematic diagram



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



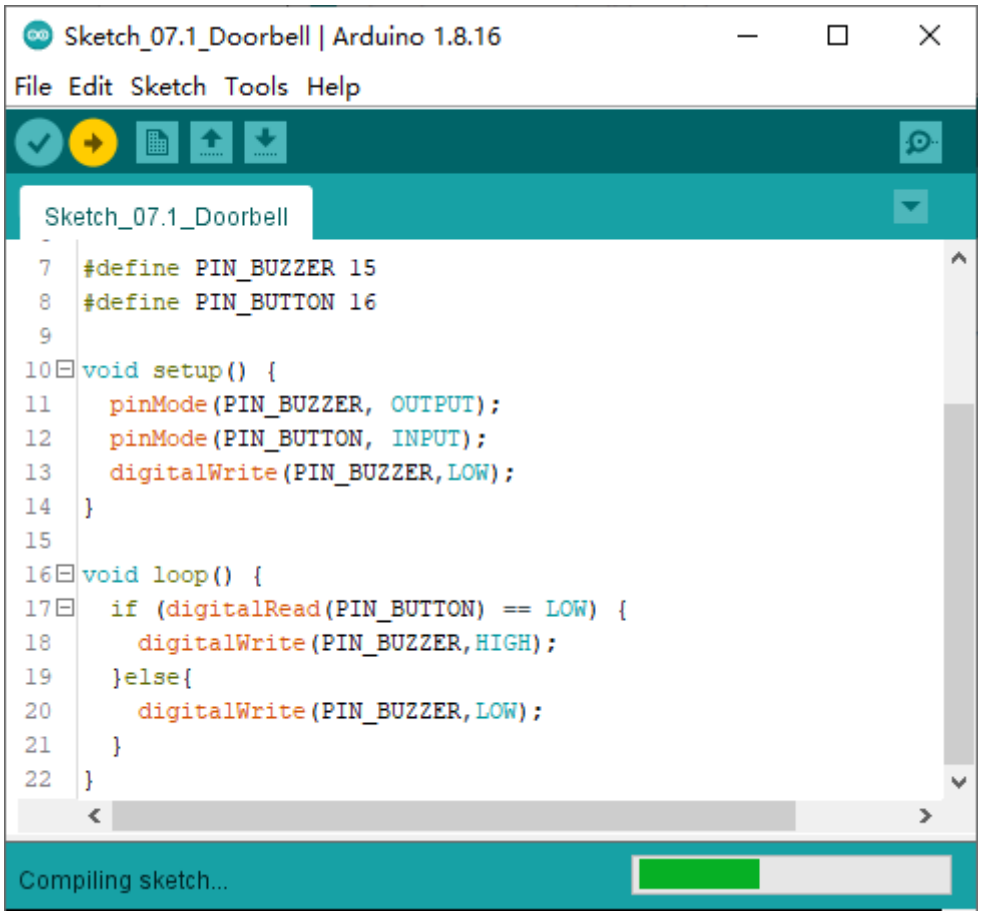
Note:

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.

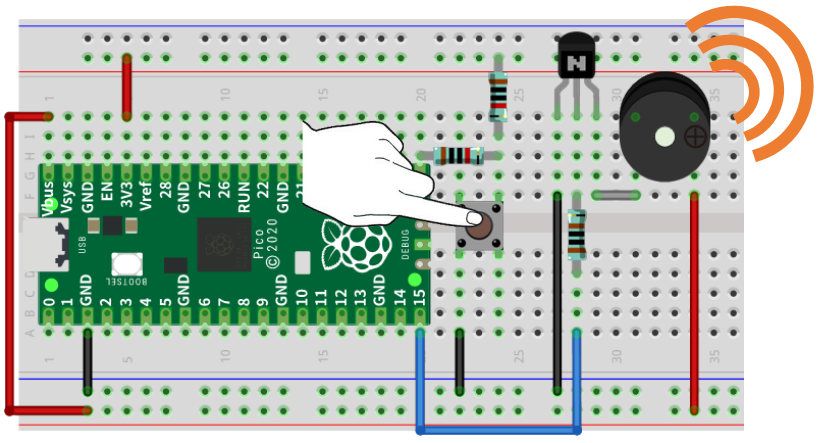
Sketch

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_06.1_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:

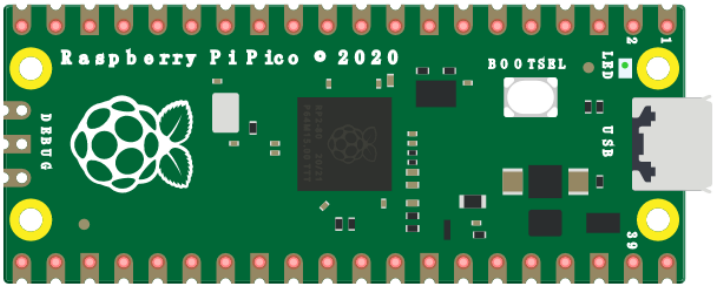
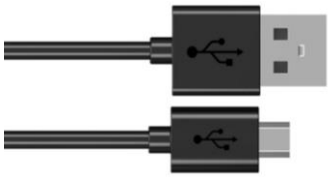
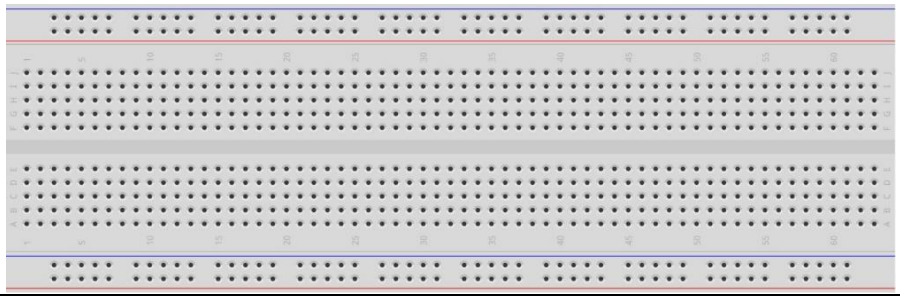
```
#define PIN_BUZZER 15
#define PIN_BUTTON 16

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 6.2 Alertor

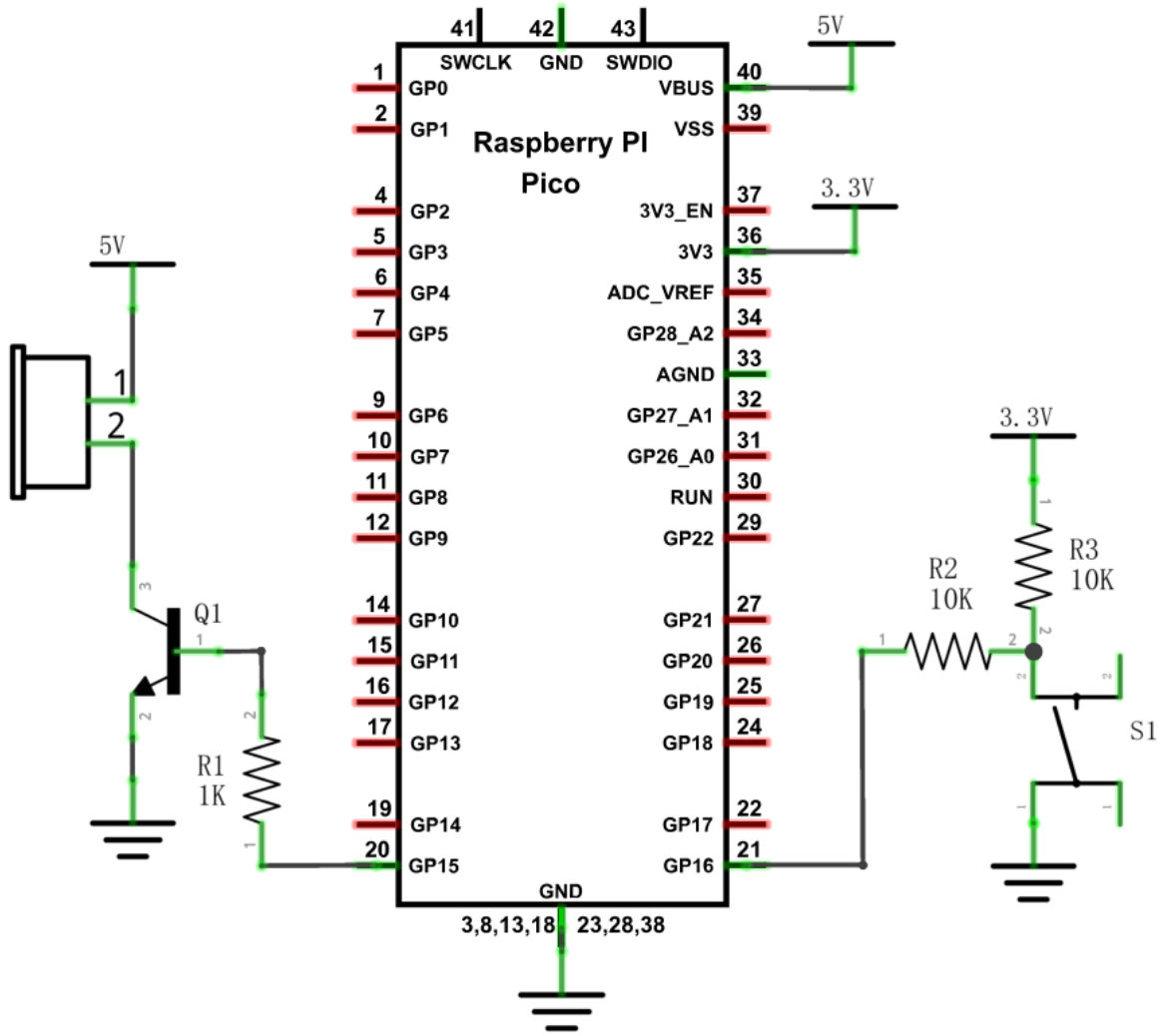
Next, we will use a passive buzzer to make an alarm.
Component list and the circuit part is similar to last section, only the **active buzzer** needs to be **replaced** with a **passive buzzer** for this project.

Component List

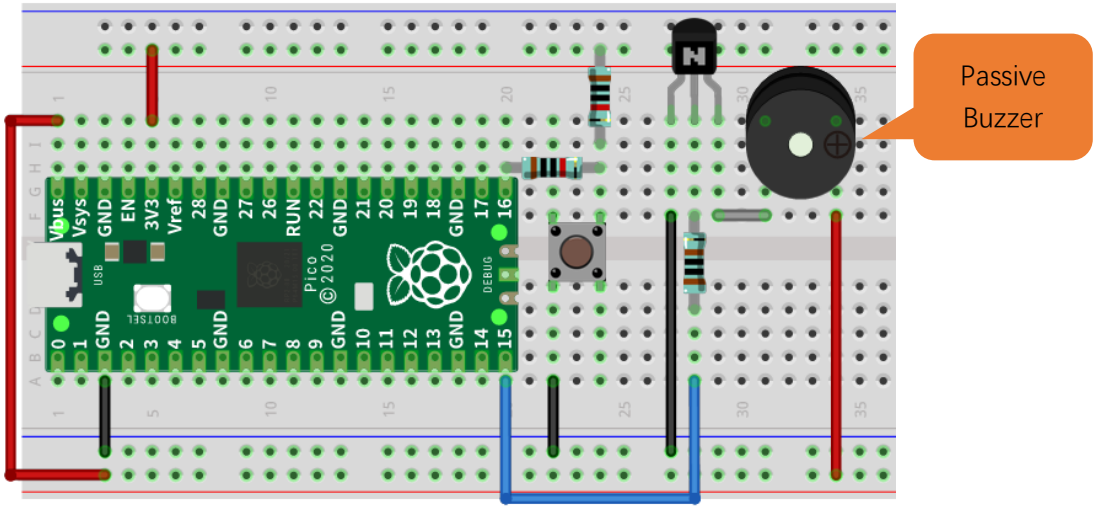
<table>
<tr><td colspan="3">Raspberry Pi Pico x1
  </td><td colspan="3">USB cable x1
  </td></tr>
<tr><td colspan="6">Breadboard x1
  </td></tr>
<tr><td colspan="6">Jumper
  </td></tr>
<tr><td colspan="2">NPN transistorx1
(S8050)
  </td><td>Passive buzzer x1
  </td><td>Push button x1
  </td><td>Resistor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

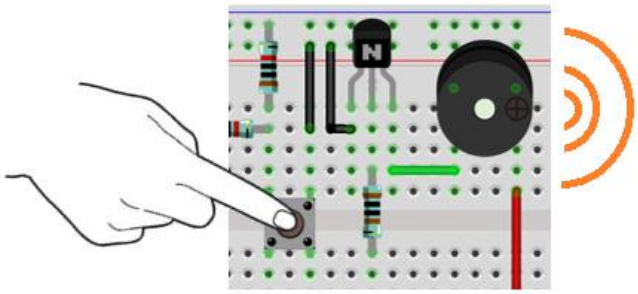


Sketch

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_06.2_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    }
```

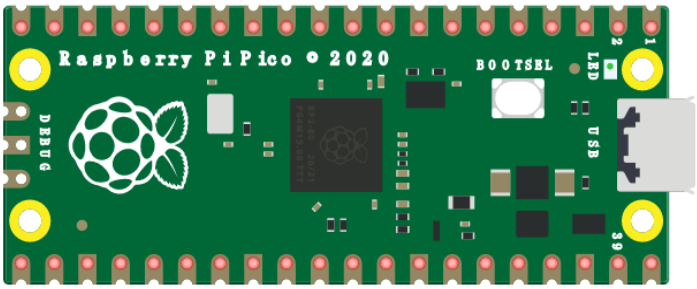
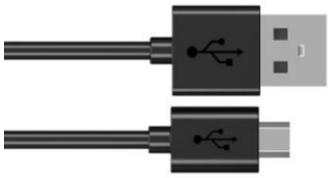
Chapter 7 Serial Communication

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

Project 7.1 Serial Print

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

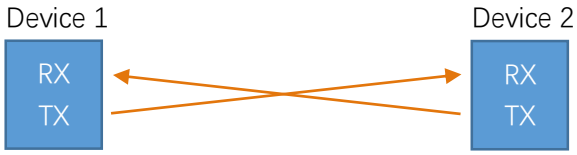
Component List

Raspberry Pi Pico x1 	USB cable x1 

Related Knowledge

Serial communication

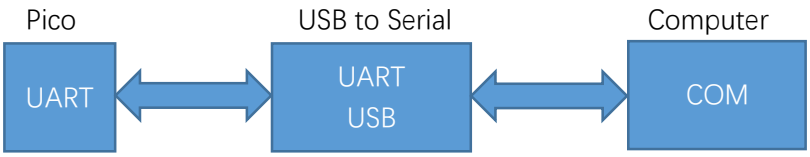
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



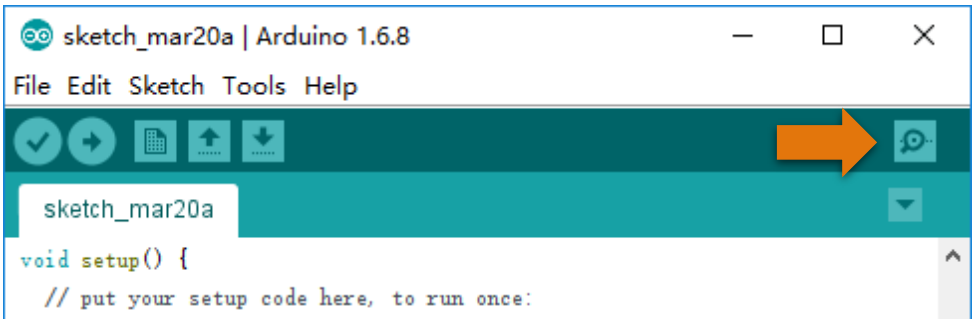
Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

Serial port on Raspberry Pi Pico

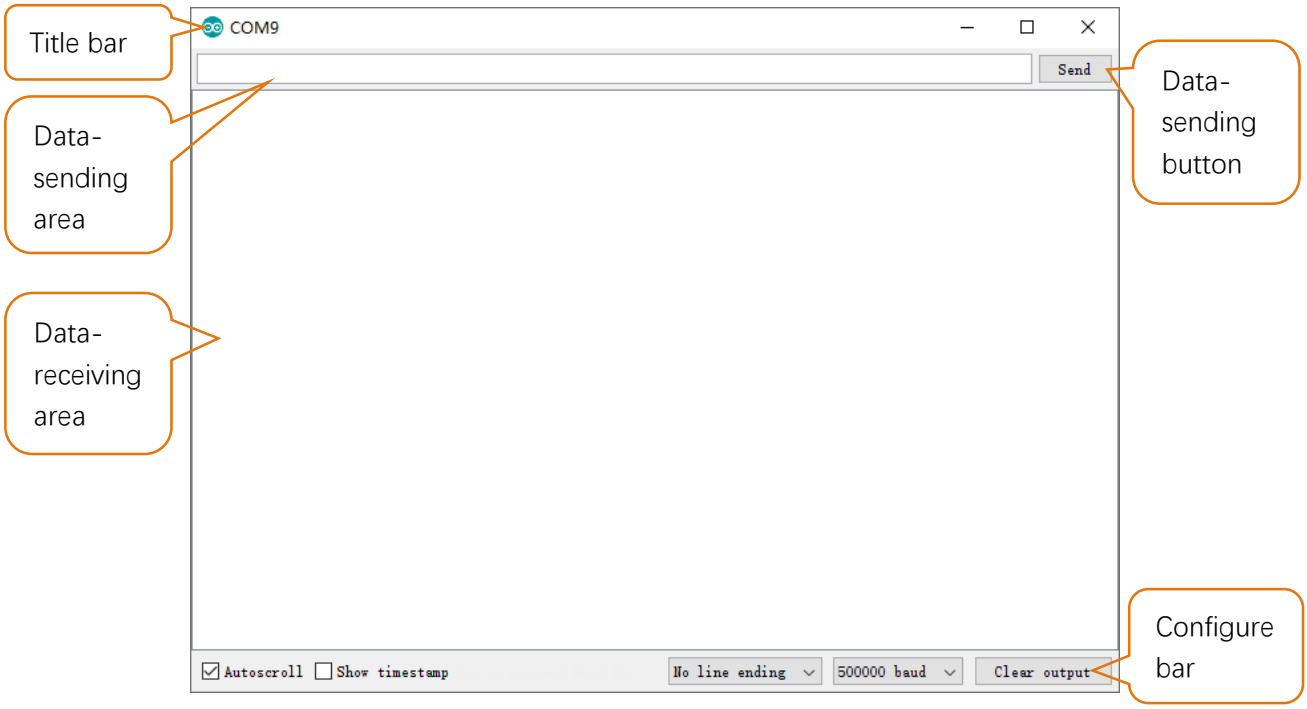
Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



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.

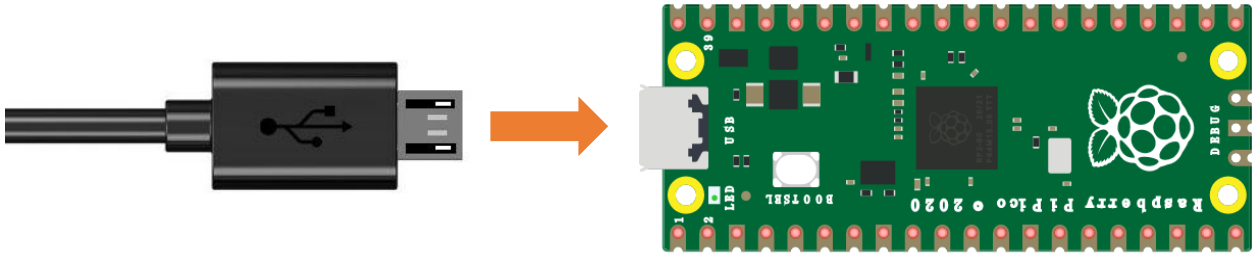


Interface of serial monitor window is as follows. If you can't open it, make sure Pico has been connected to the computer, and choose the right serial port in the menu bar "Tools".



Circuit

Connect Raspberry Pi Pico to the computer with USB cable.



Sketch

Sketch_07.1_SerialPrinter

Sketch_08.1_SerialPrinter | Arduino 1.8.16

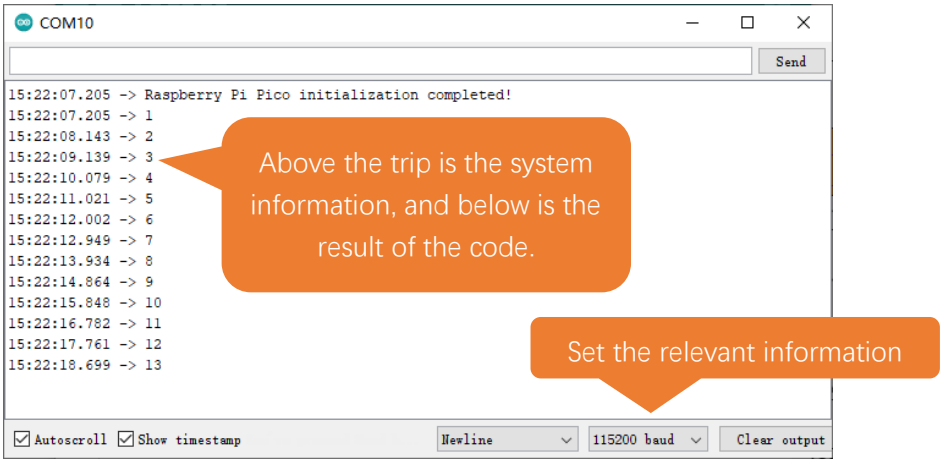
File Edit Sketch Tools Help

Sketch_08.1_SerialPrinter

```
/**********************************************************************
  Filename    : SerialPrinter
  Description : Use UART send some data to PC, and show them on serial monitor.
  Auther      : www.freenove.com
  Modification: 2020/07/11
**********************************************************************/

void setup() {
  Serial.begin(115200);
  delay(2000);
  Serial.println("Raspberry Pi Pico initialization completed!");
}

void loop() {
  Serial.println( millis() / 1000 % 60);
  delay(1000);
}
```

Done compiling.

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



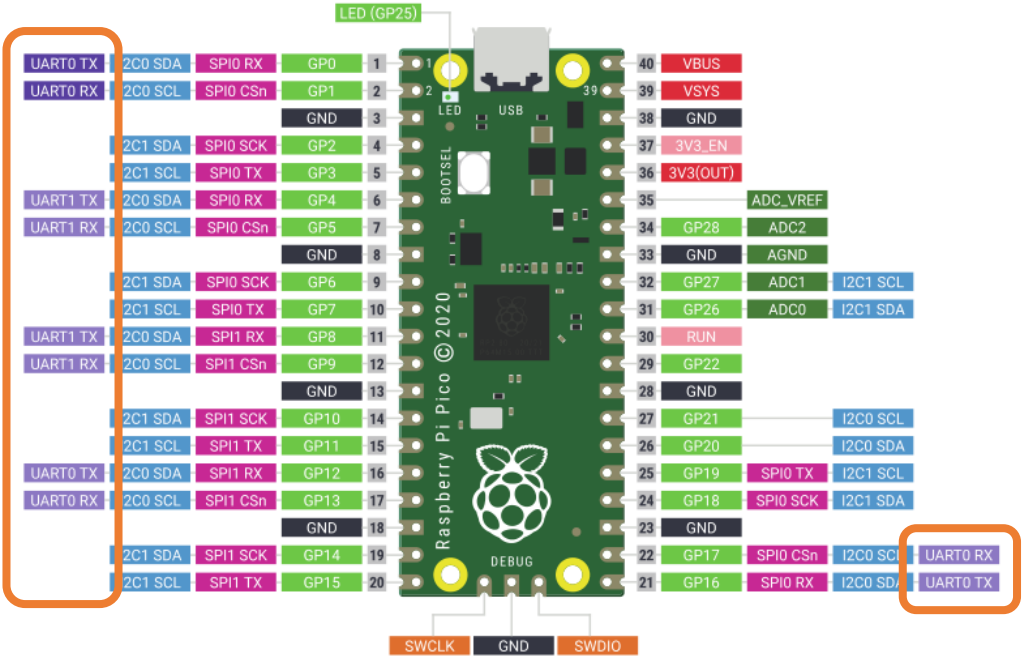
As shown above, when the code runs, the data is printed every one second.

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 7.2 Serial Read and Write

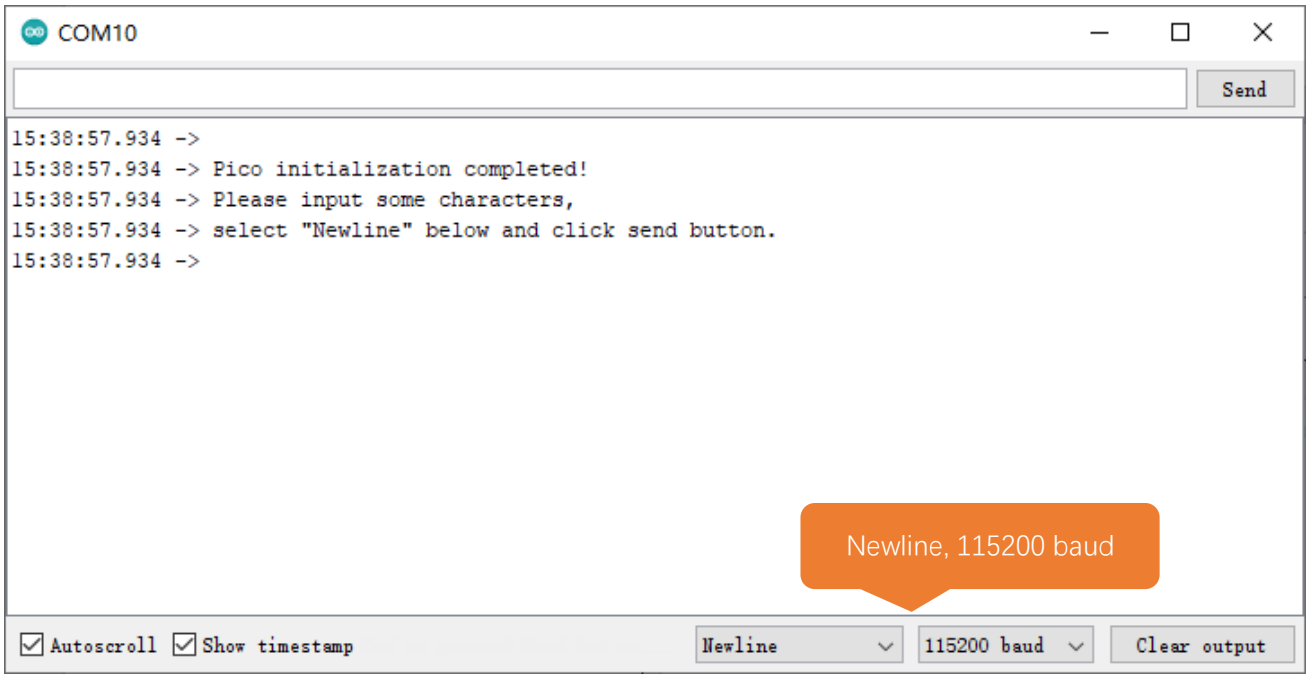
From last section, we use serial port on Pico to send data to a computer, now we will use that to receive data from computer.

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

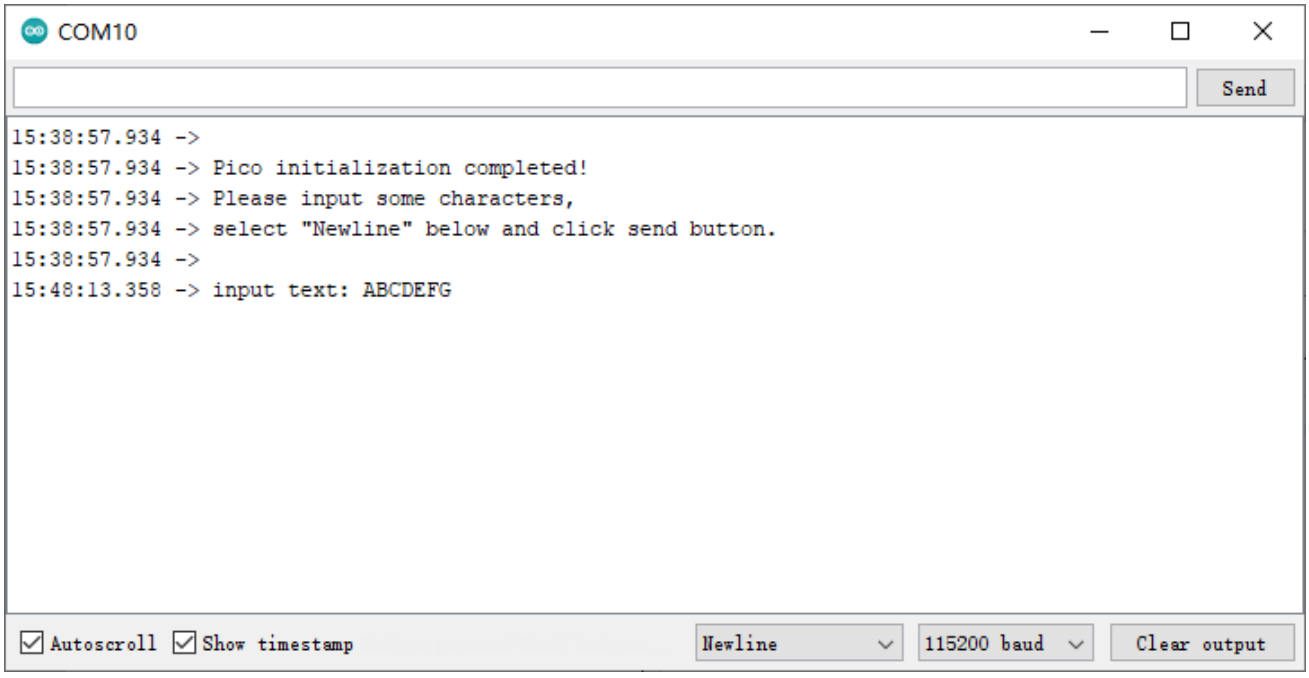
Sketch

Sketch_07.2_SerialRW

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.

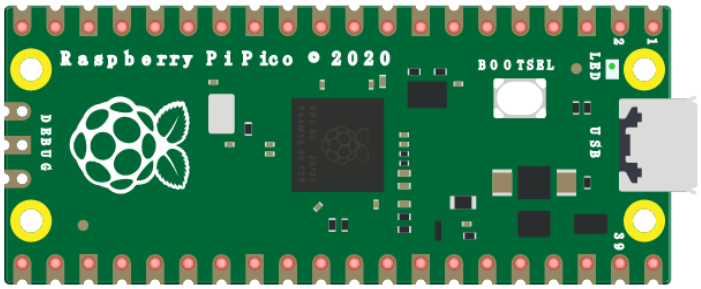
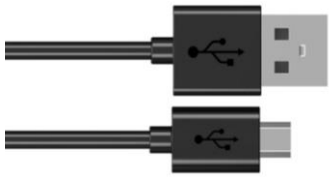
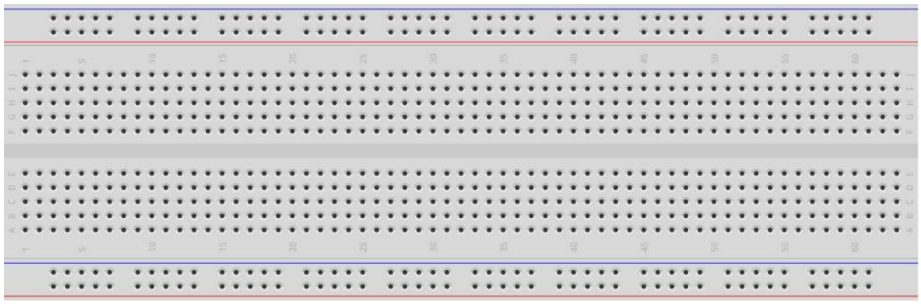
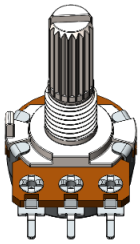
Chapter 8 AD Converter

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

Project 8.1 Read the Voltage of Potentiometer

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

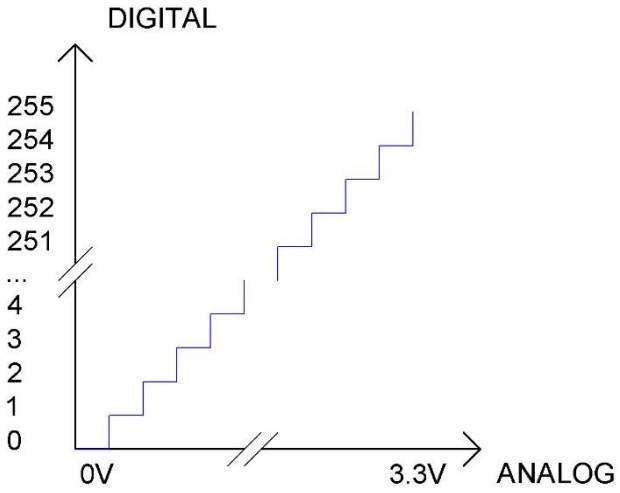
Component List

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:

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

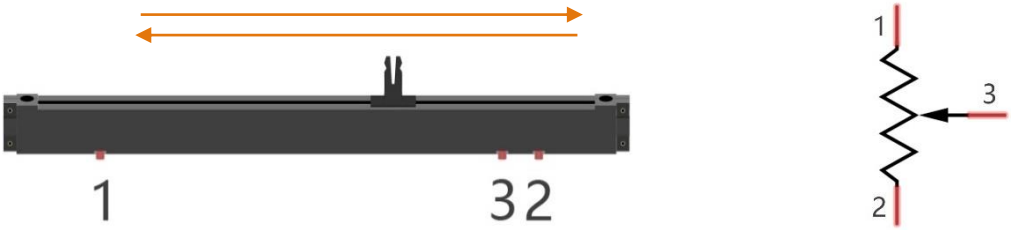
ADC ChannelsRaspberry Pi Pico

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

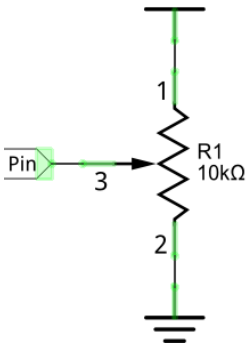
Potentiometer

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.



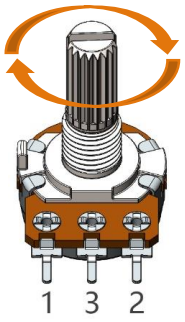
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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



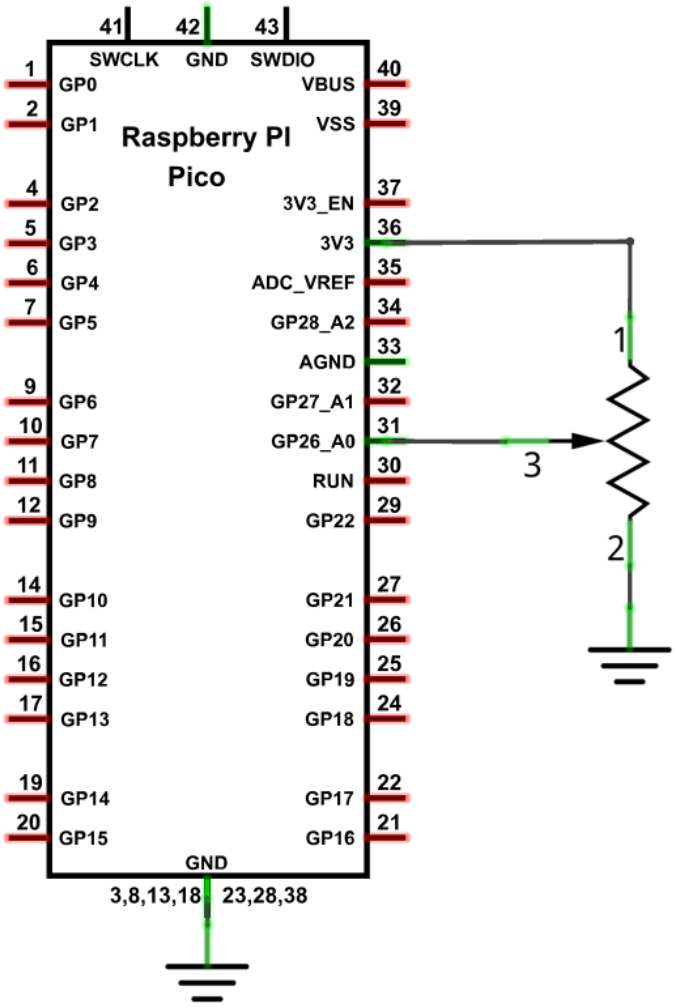
Rotary potentiometer

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

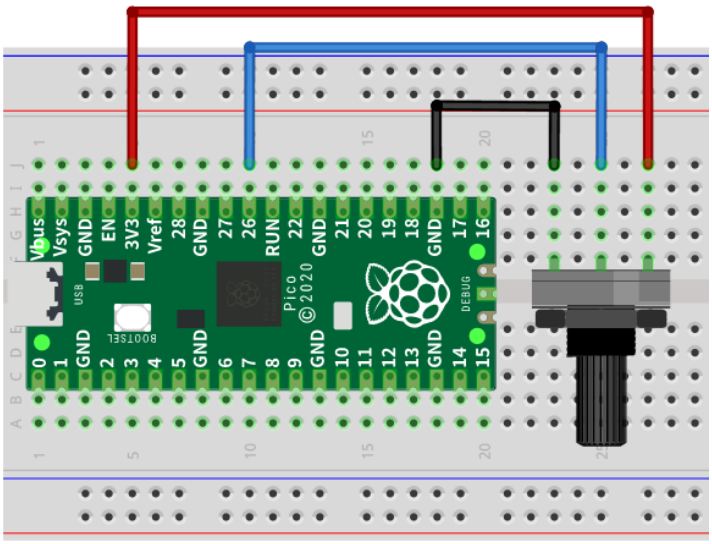


Circuit

Schematic diagram



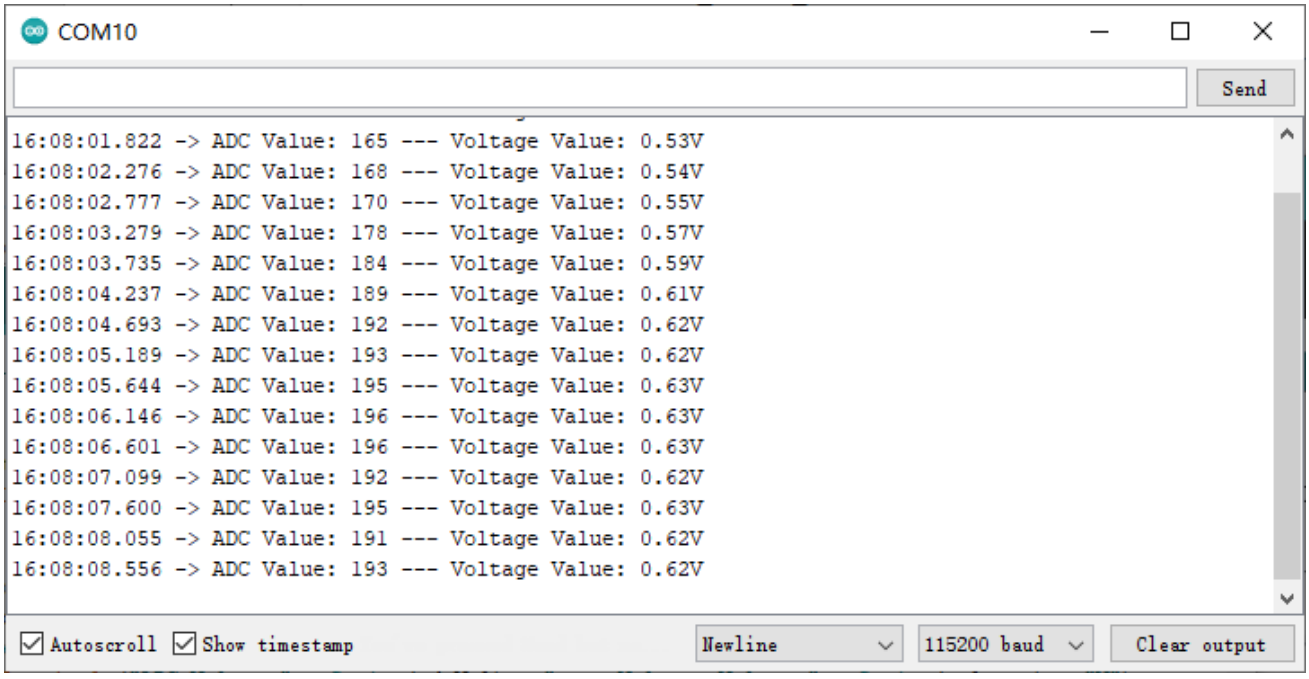
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_08.1_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).

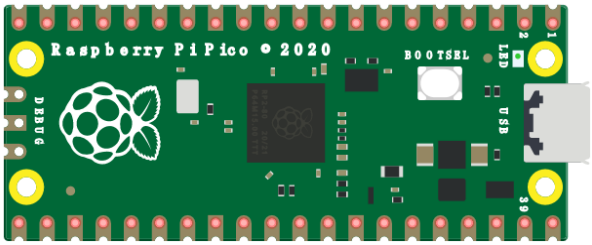
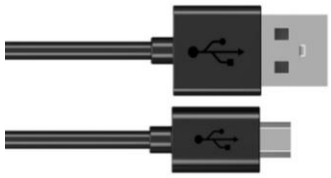
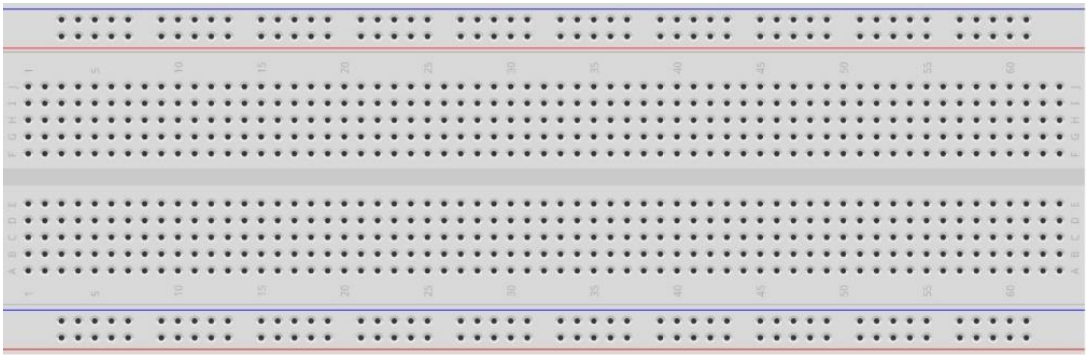
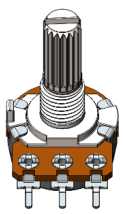
Chapter 9 Potentiometer & LED

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 9.1 Soft Light

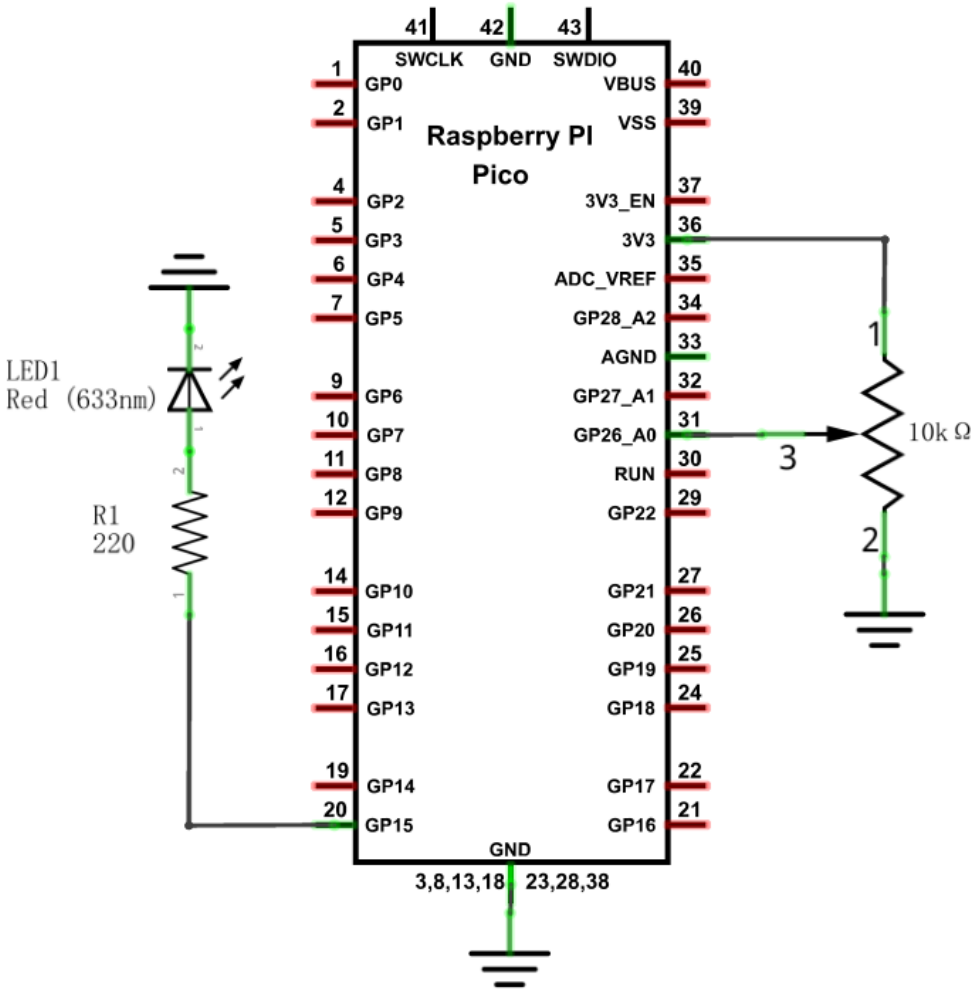
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

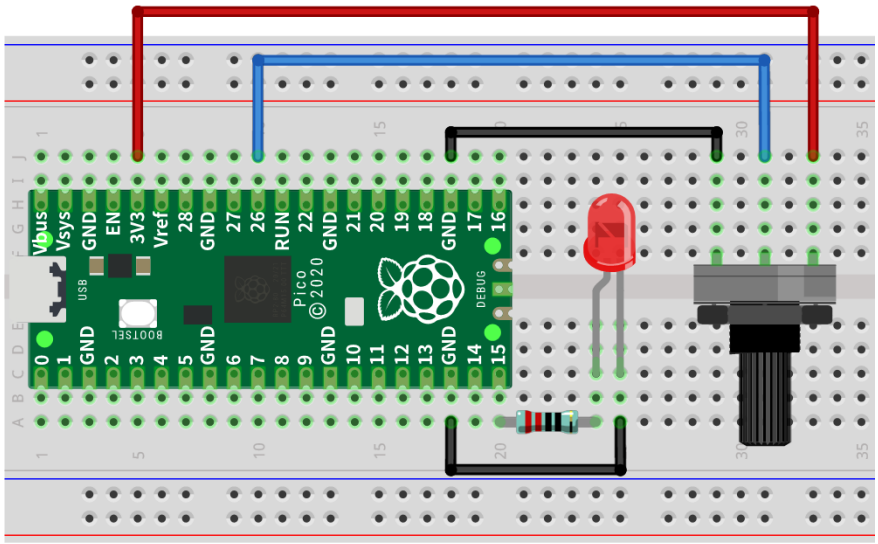
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td><td>Jumper
</td></tr>
</table>

Circuit

Schematic diagram



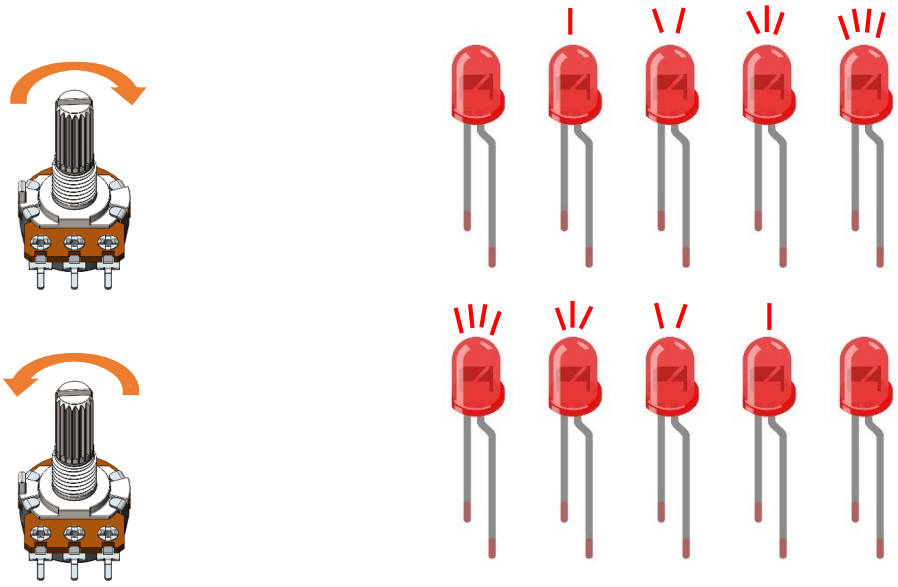
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_09.1_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.

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

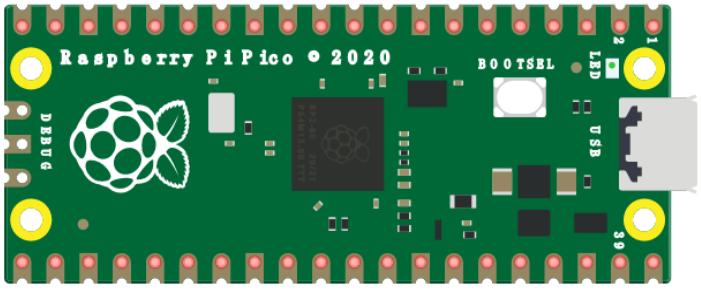
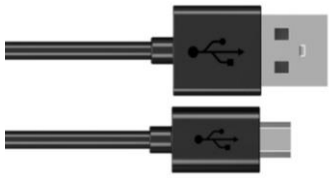
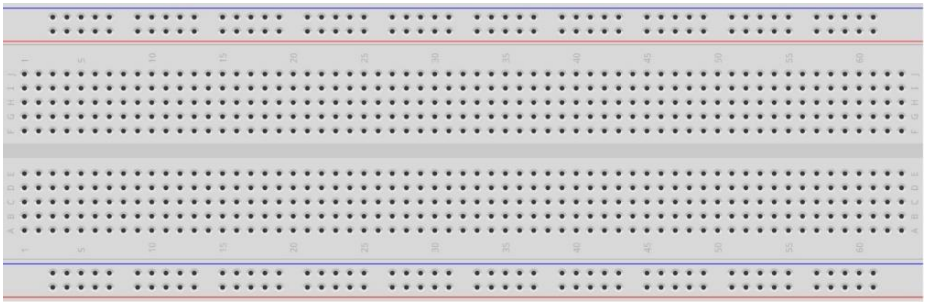
Chapter 10 Photoresistor & LED

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

Project 10.1 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.

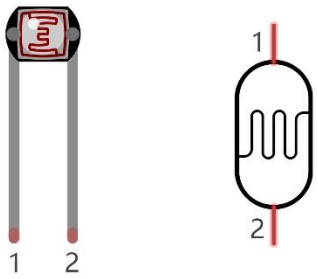
Component List

<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>

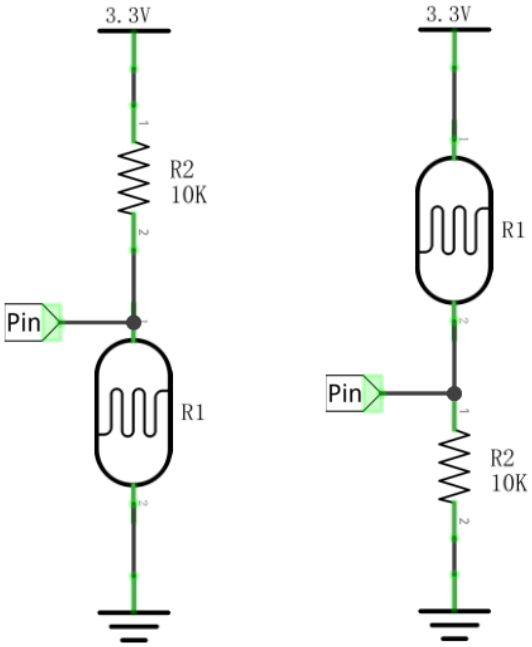
Component Knowledge

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:

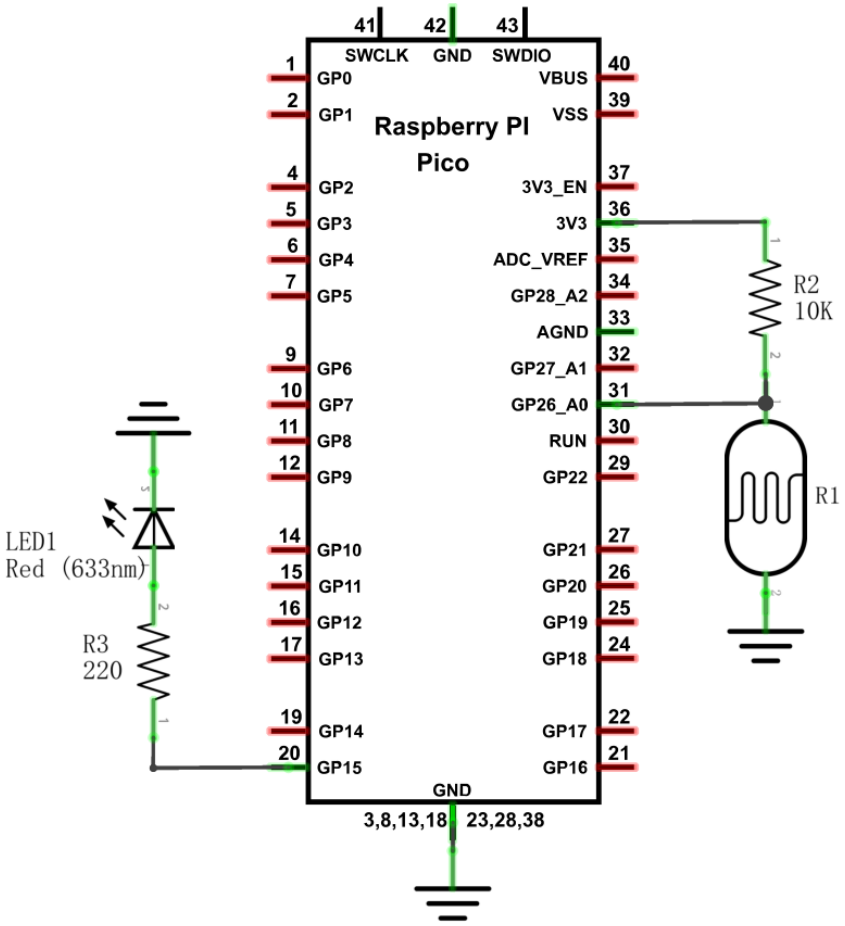


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

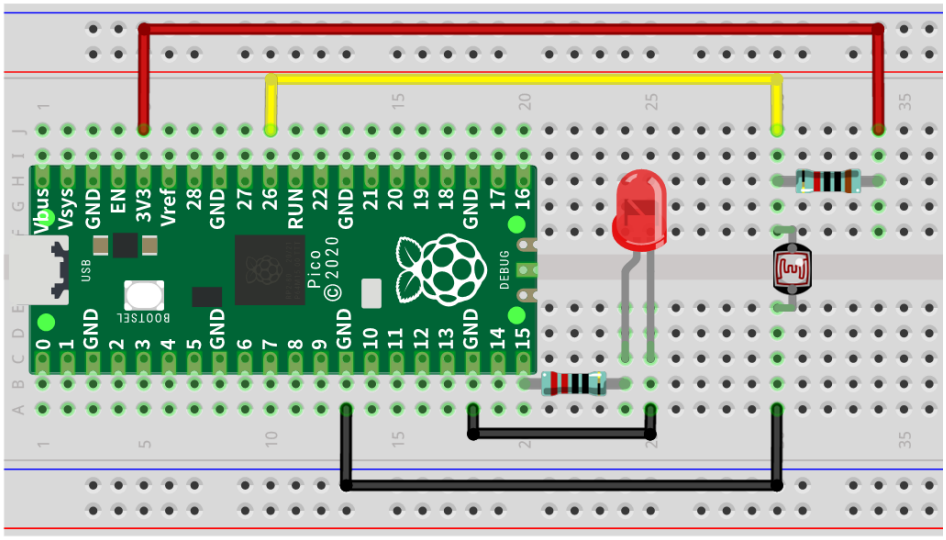
Circuit

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



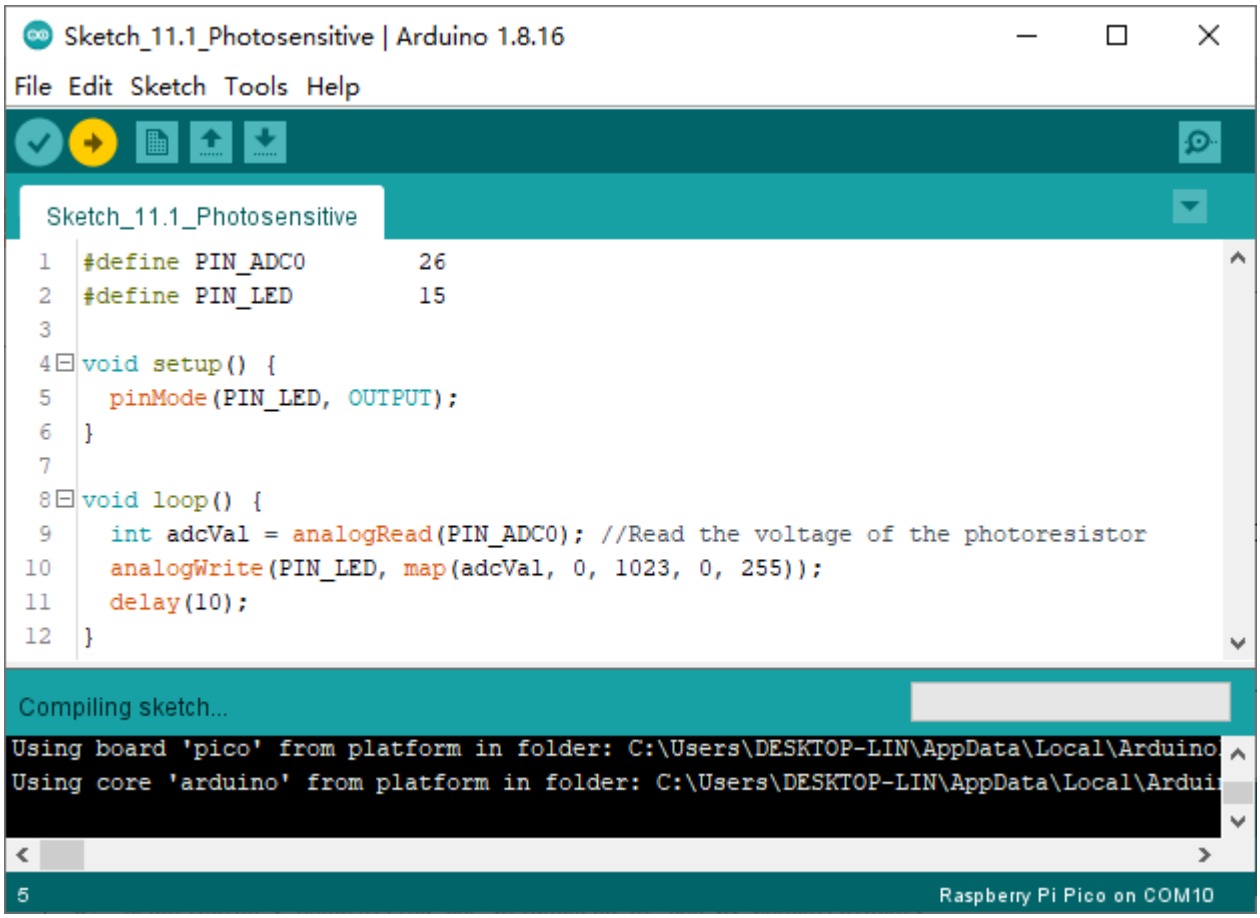
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



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.

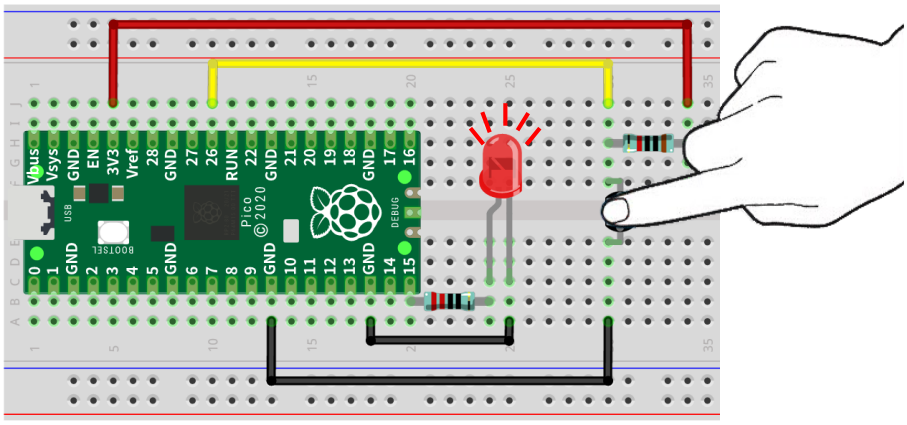
Sketch_10.1_Nightlamp



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

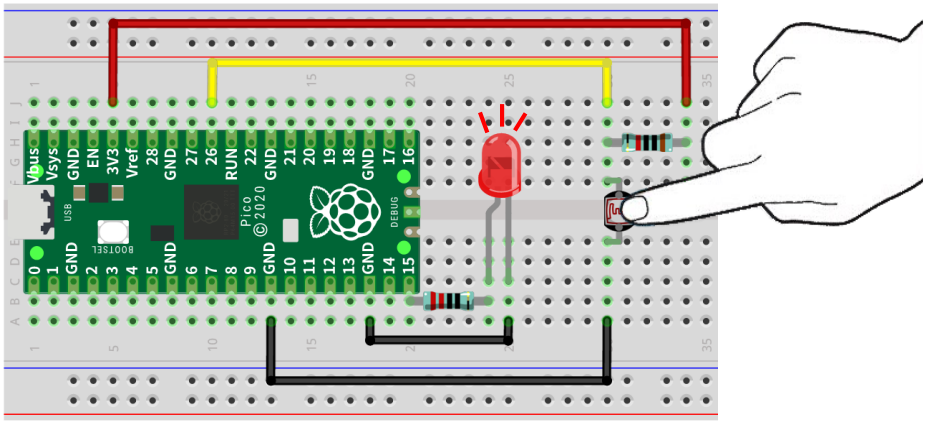
If you have any concerns, please contact us via: support@freenove.com

Fully cover the photoresistor:

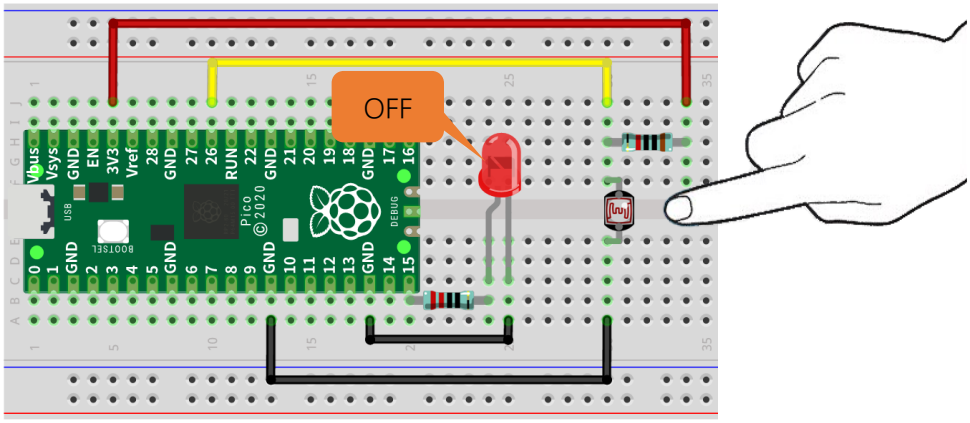


Any concerns? ✉ support@freenove.com

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);
}
```

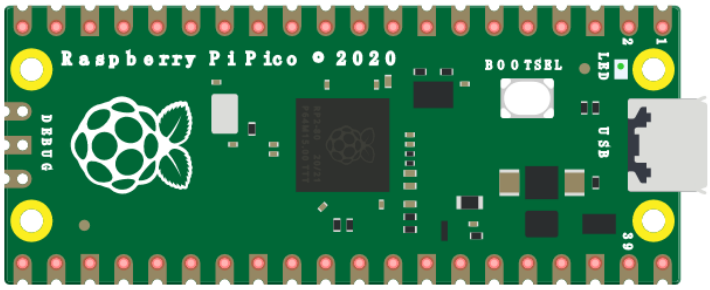
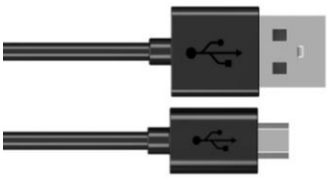
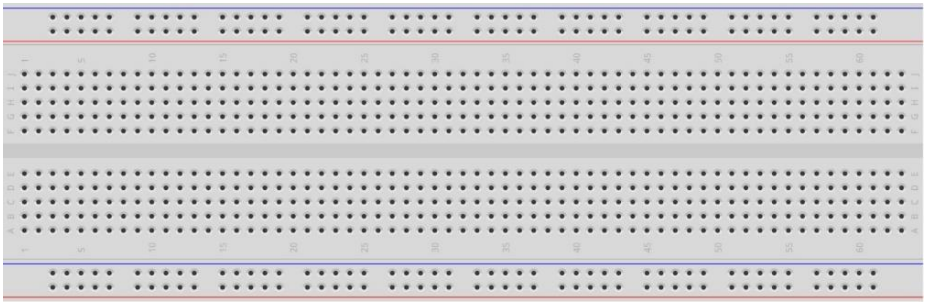
Chapter 11 Thermistor

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

Project 11.1 Thermometer

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

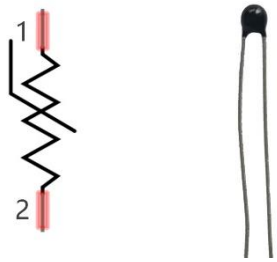
Component List

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

Component Knowledge

Thermistor

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



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

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

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

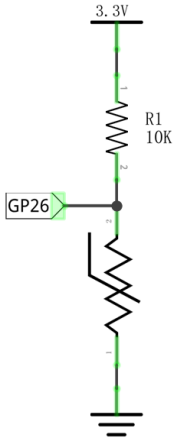
EXP[n] is nth power of e;

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.

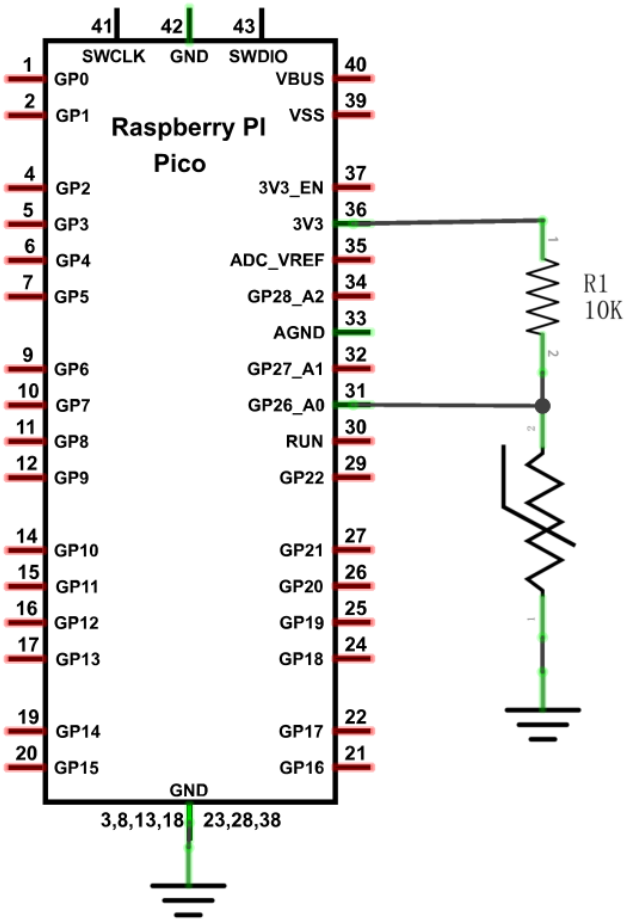
Therefore, the temperature formula can be derived as:

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

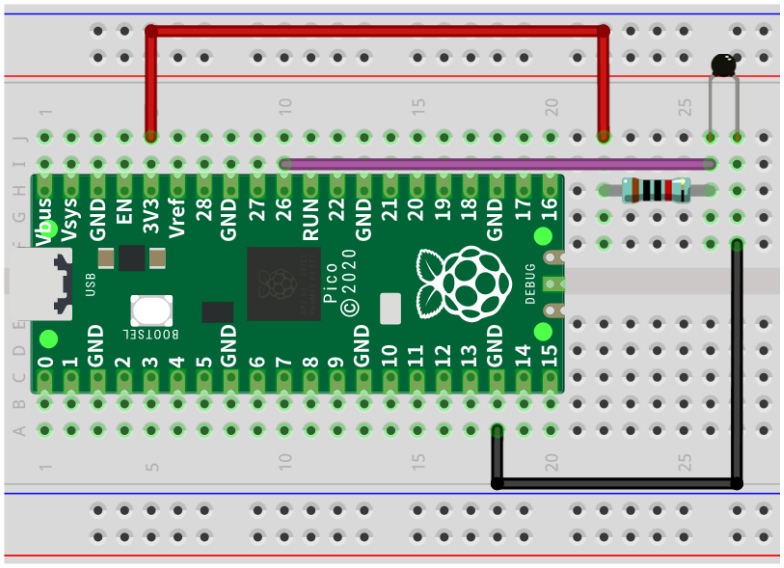
Circuit

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



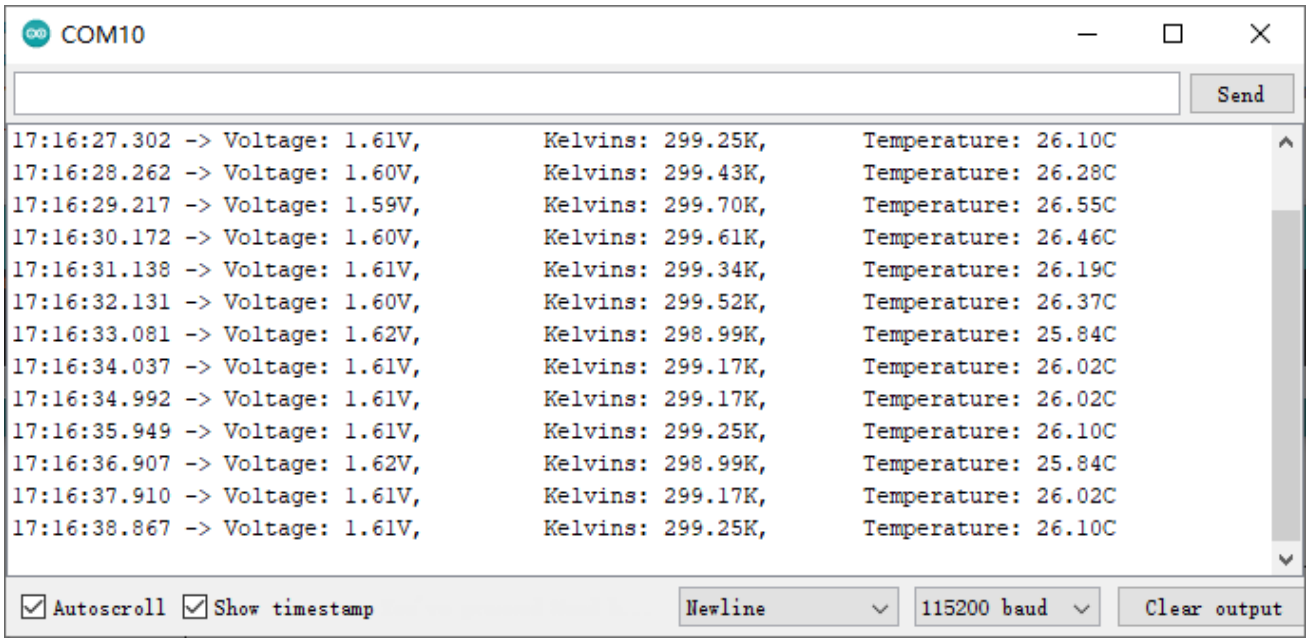
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_11.1_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.

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.