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_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico

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

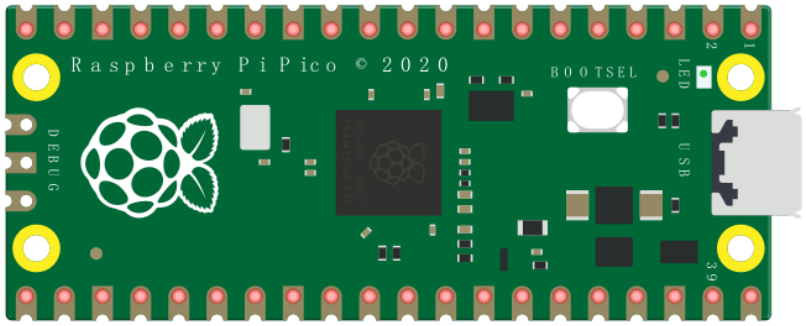
1, Component list: helps users to learn and find what components are needed in each project.
2, Component Knowledge: allows you to learn the features and usage of the components.
3, Circuit: assists to build circuit for each project.
4, Sketches and comments: makes it easier for users to learn to use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

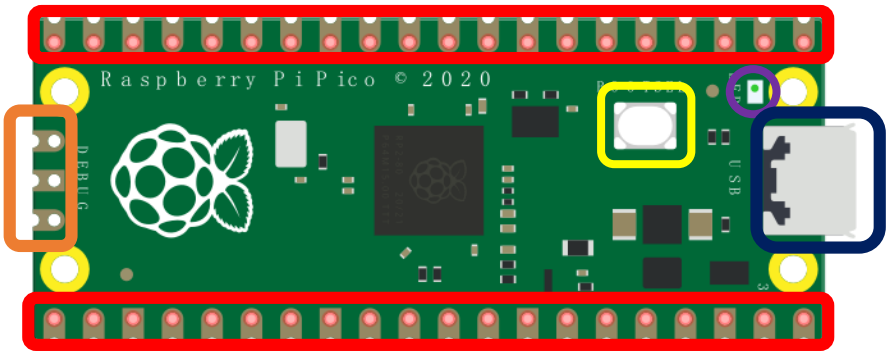
If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

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.

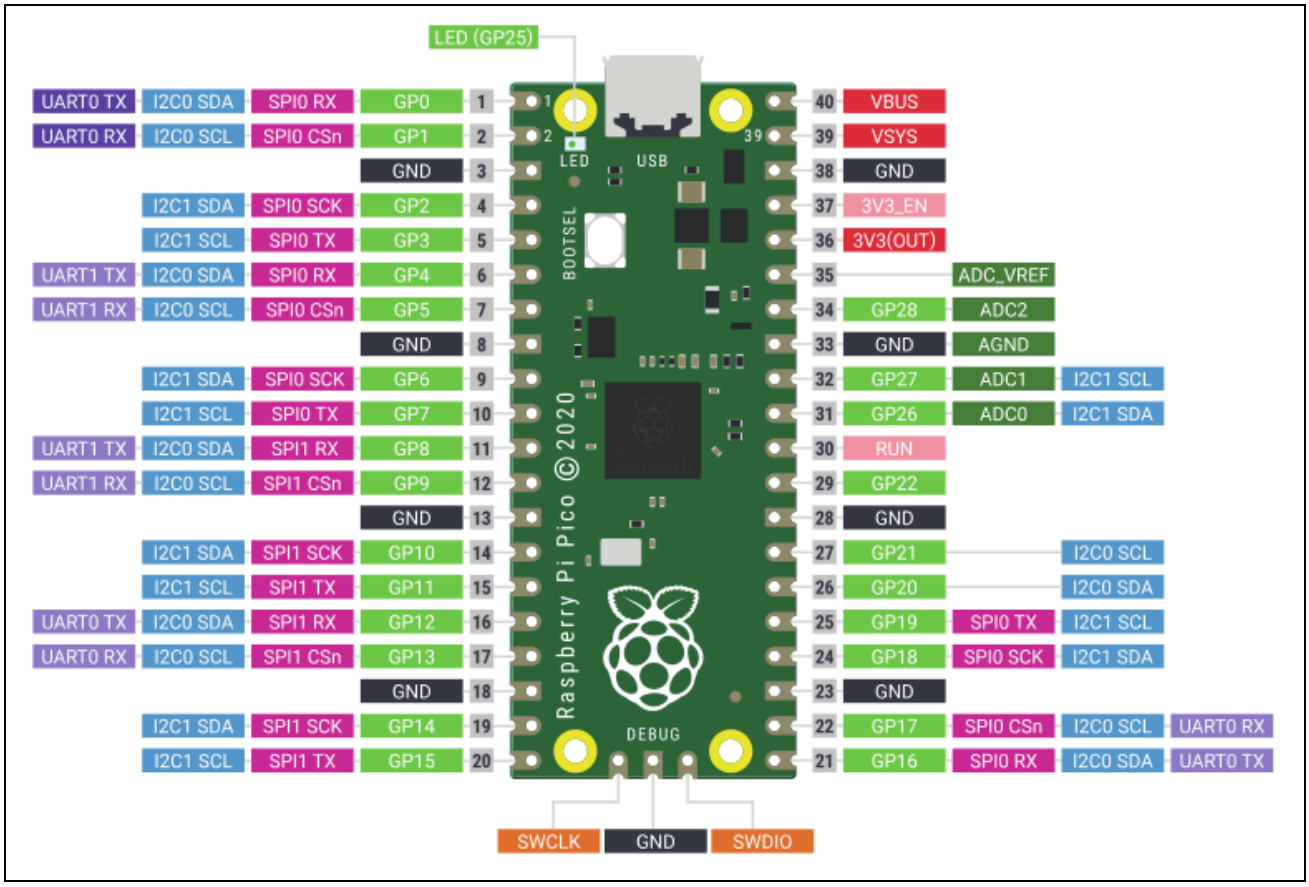


The hardware interfaces are distributed as follows:



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

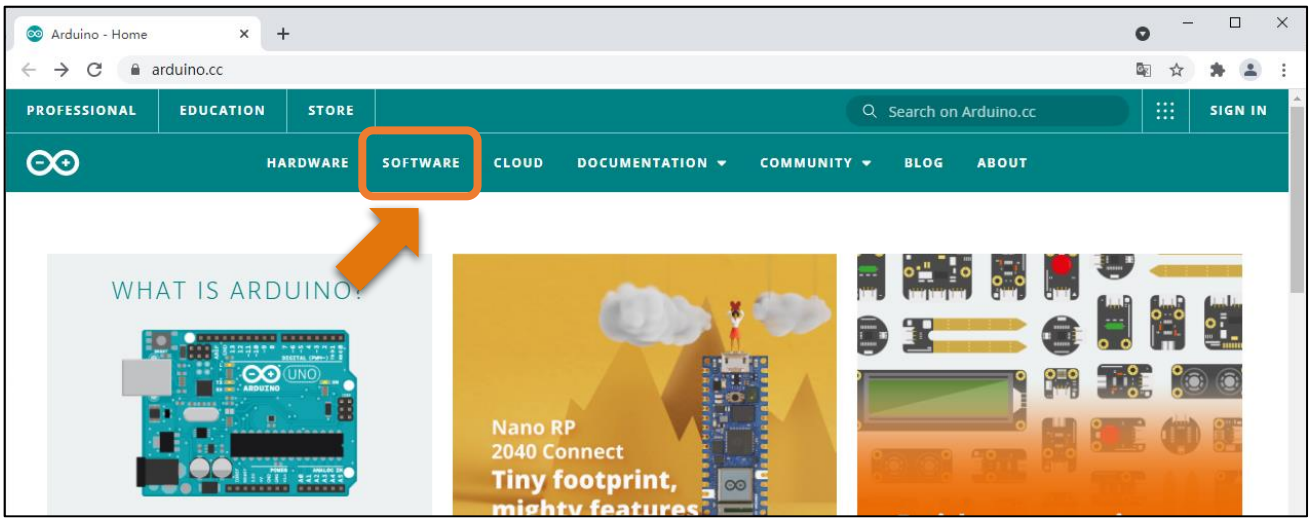
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 2
SPI_MOSI	Pin 3
SPI_MISO	Pin 4
SPI_SS	Pin 5

Chapter 0 Getting Ready (Important)

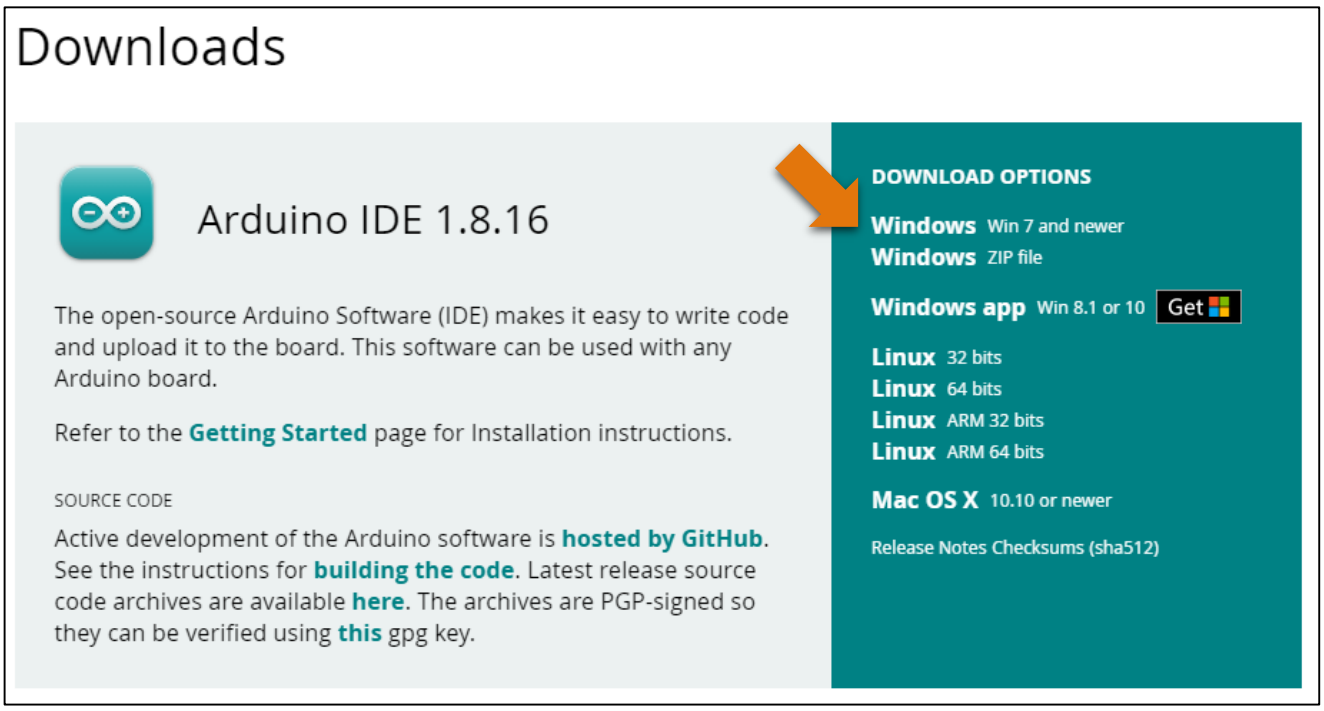
Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

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.

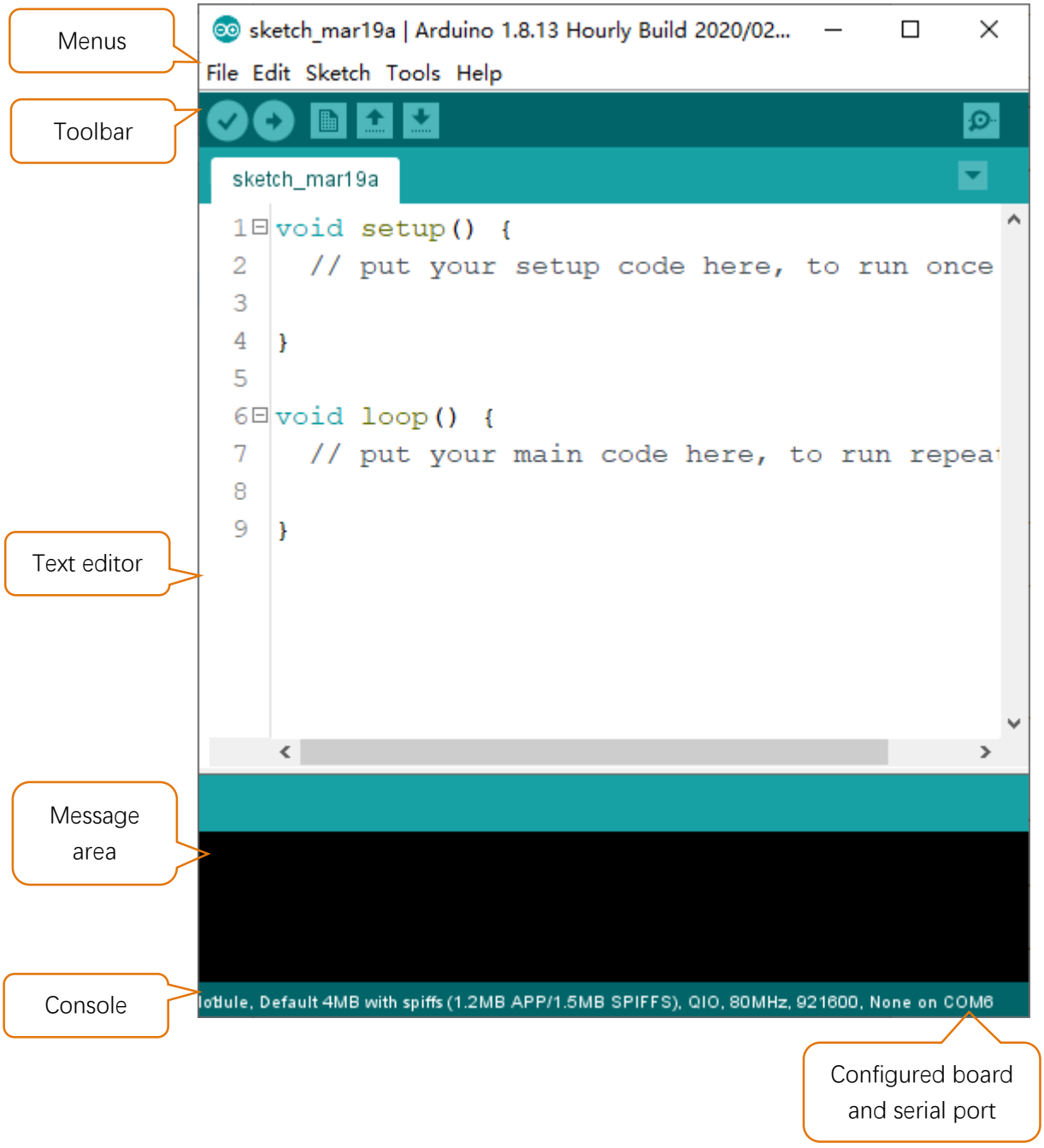


Select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer" to download to install the driver correctly.



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.

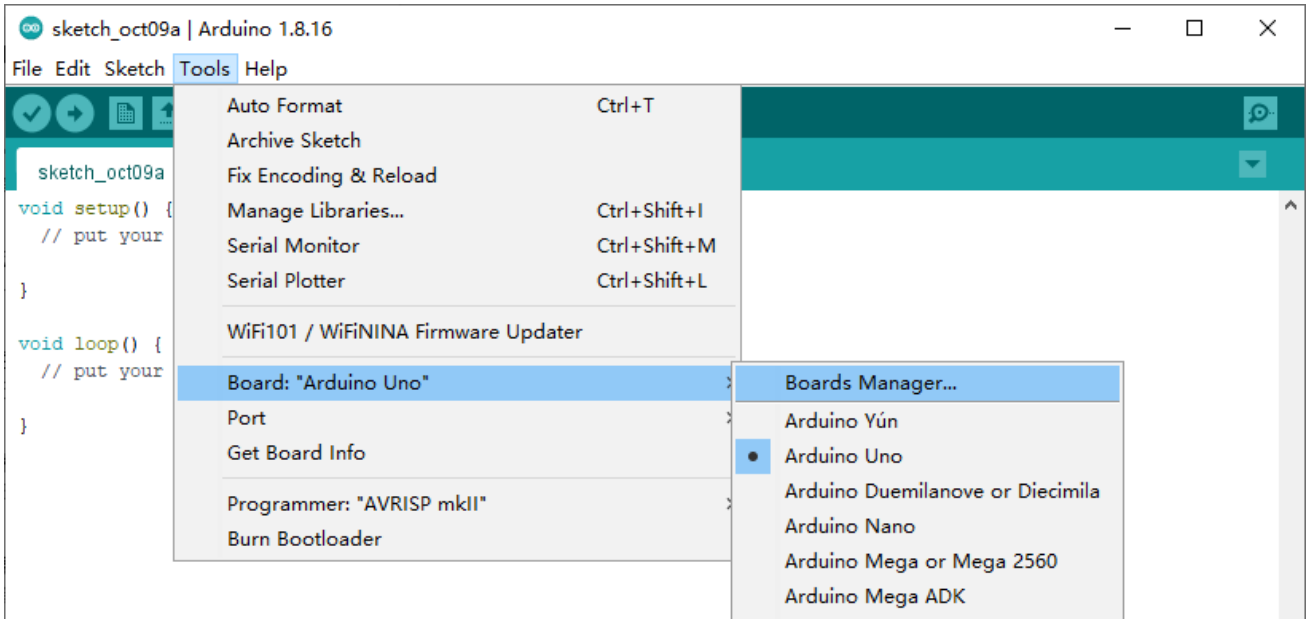
Serial Monitor
Open the serial monitor.

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.

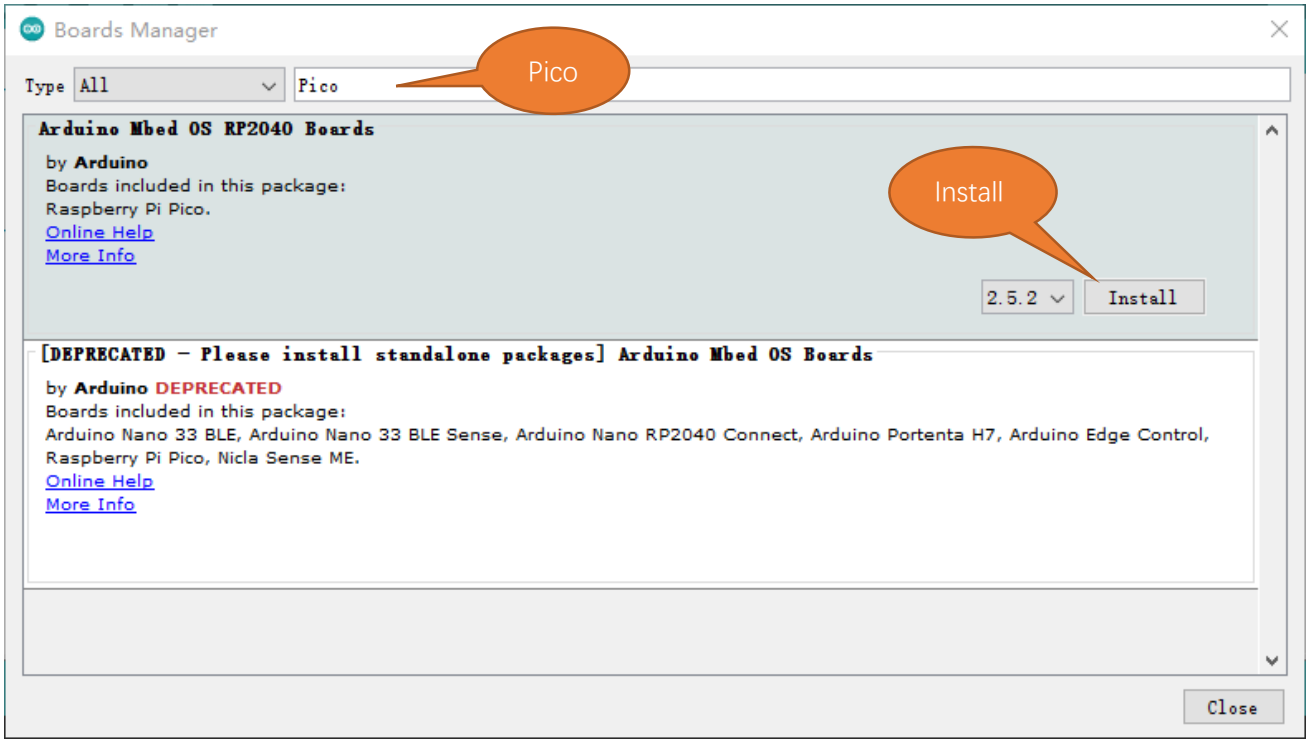
Installation of Development Board Support Package

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.

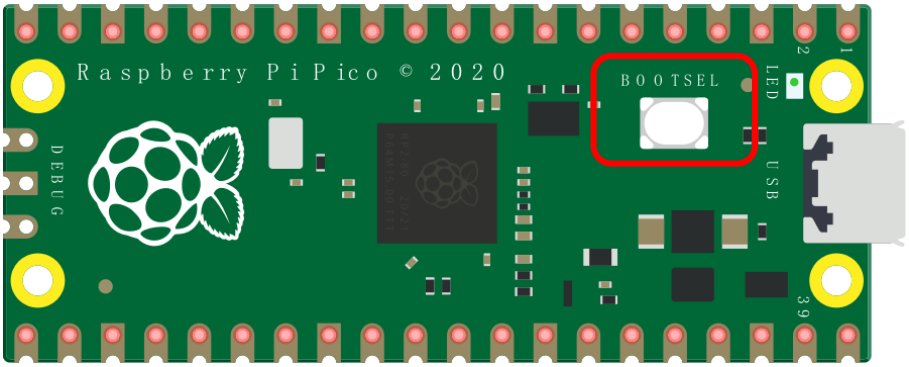


4, Click Yes in the pop-up "**dpinst-amd64.exe**"installation window. (Without it, you will fail to communicate with Arduino.) Thus far, we have finished installing the development support package.

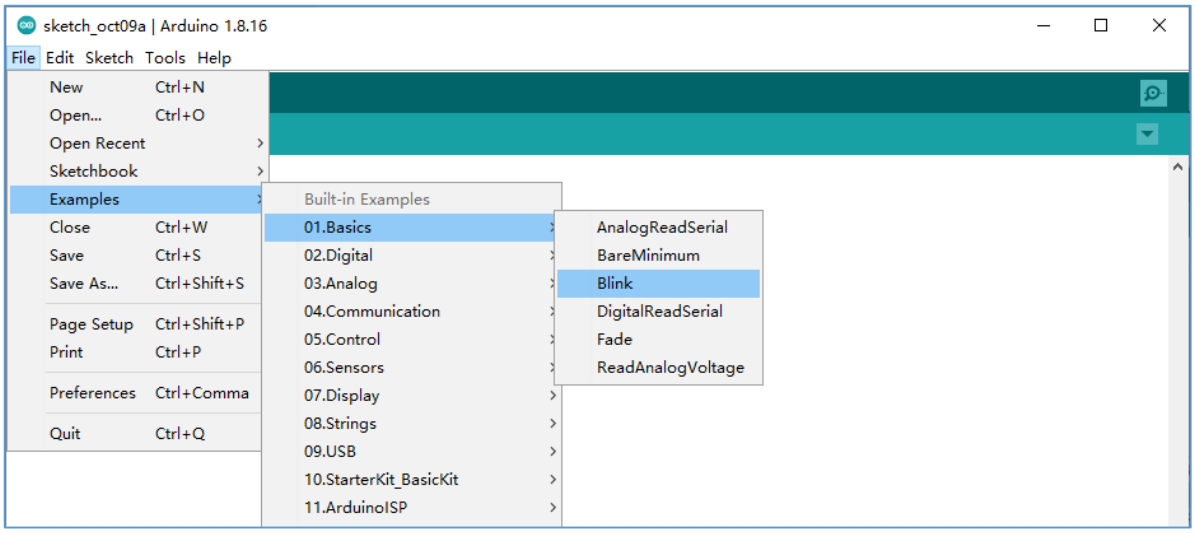
Uploading Adruino-compatible Firmware for Pico

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.

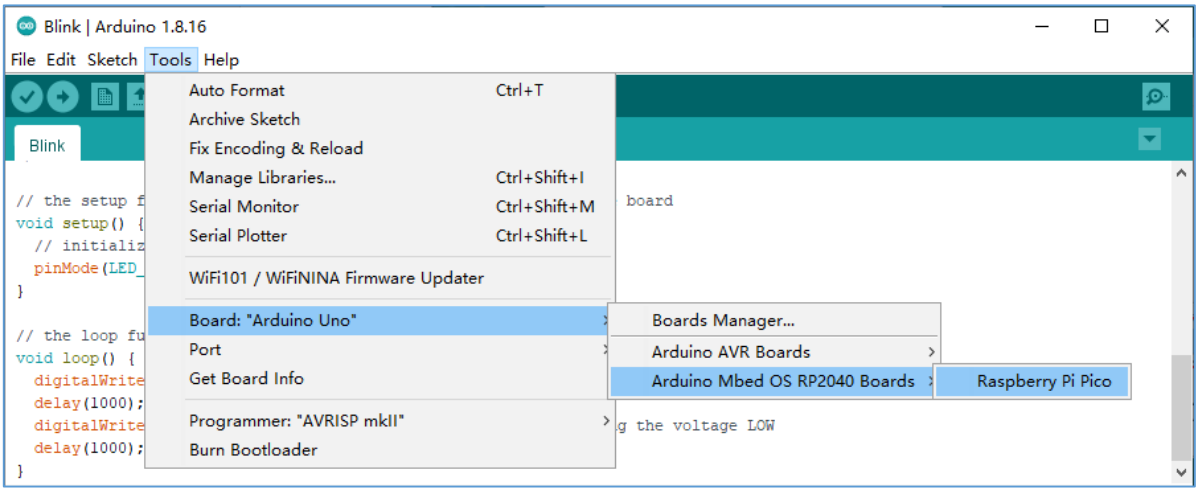
1, Disconnect Pico from computer. Keep pressing the white button(BOOTSEL) on Pico, and connect Pico to computer before releasing the button. (Note: Be sure to keep pressing the button before powering the Pico, otherwise the firmware will not download successfully)



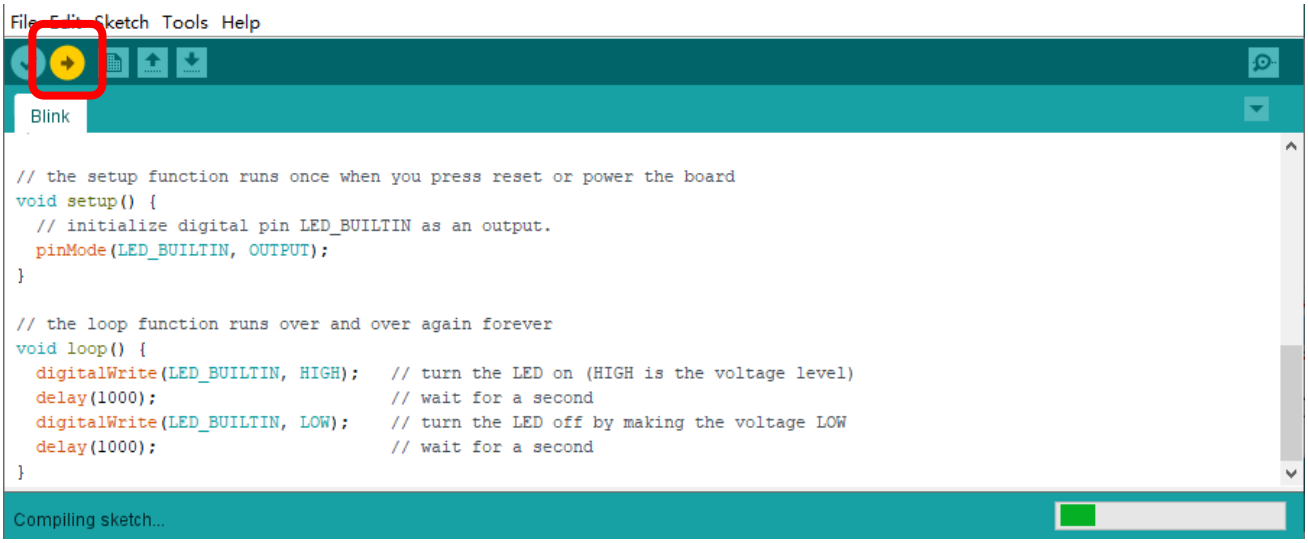
2, Open Arduino IDE. Click File>Examples>01.Basics>Blink.



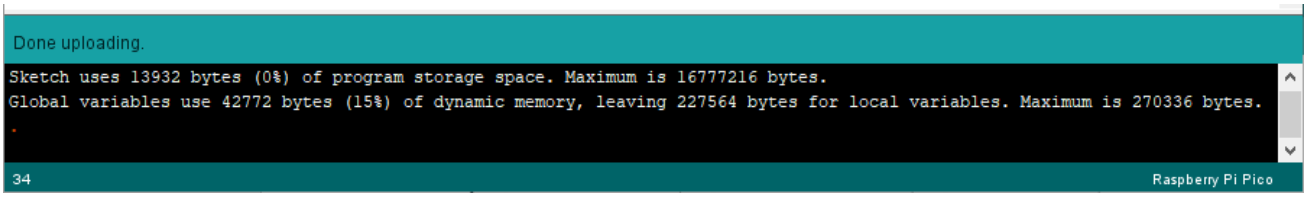
3, Click **Tools>Board>Arduino Mbed OS RP2040 Boards>Raspberry Pi Pico**.



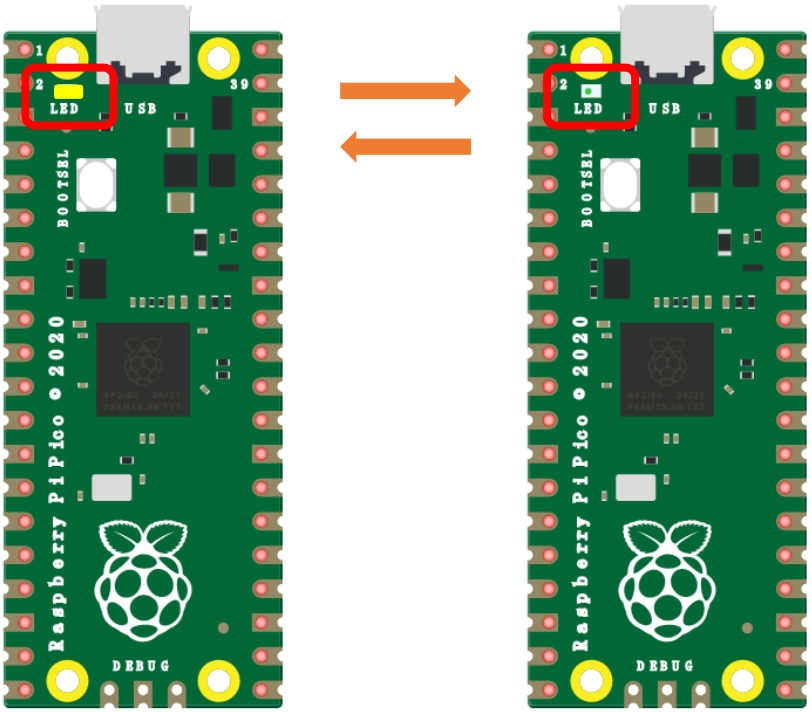
4, Upload sketch to Pico.



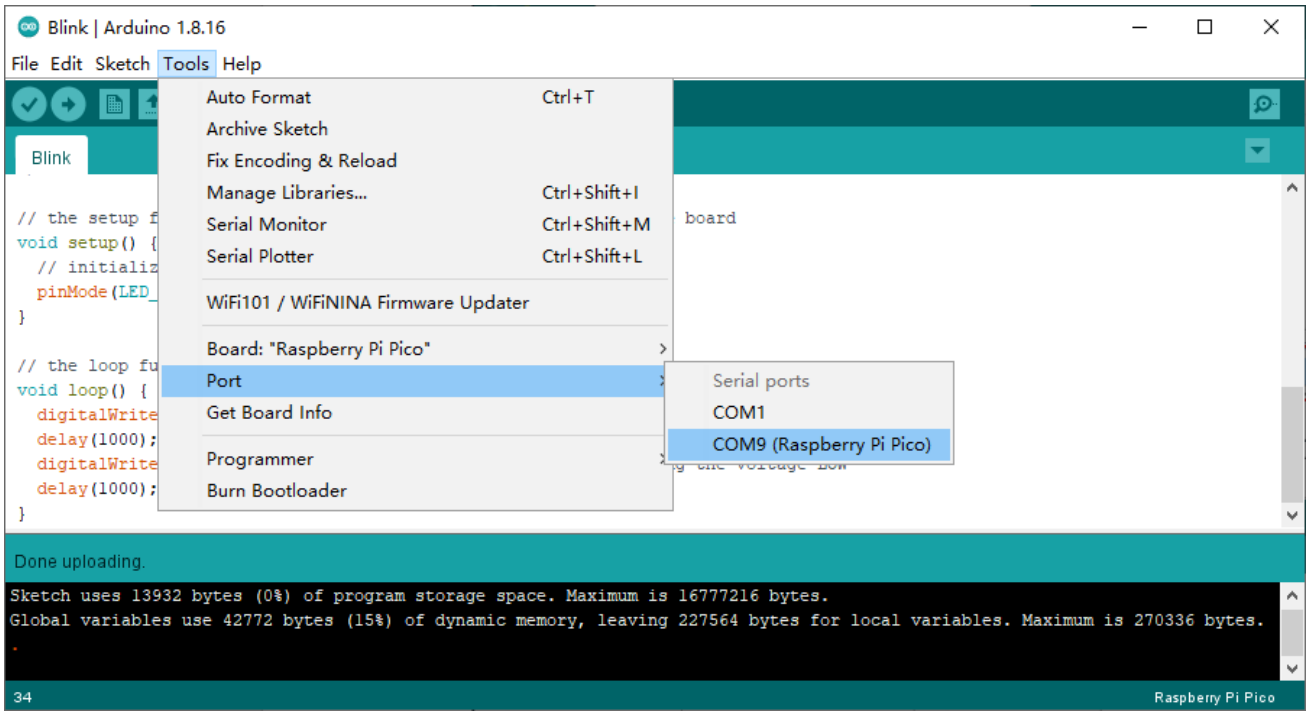
When the sketch finishes uploading, you can see the following prompt.



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.

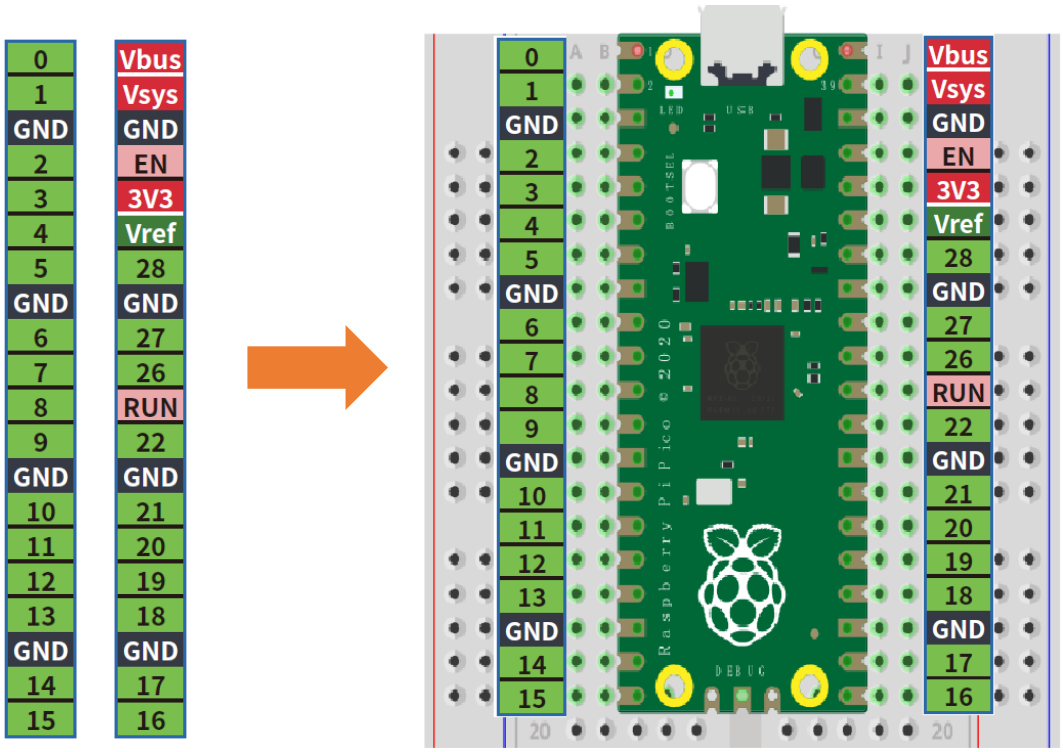


Note:

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.**

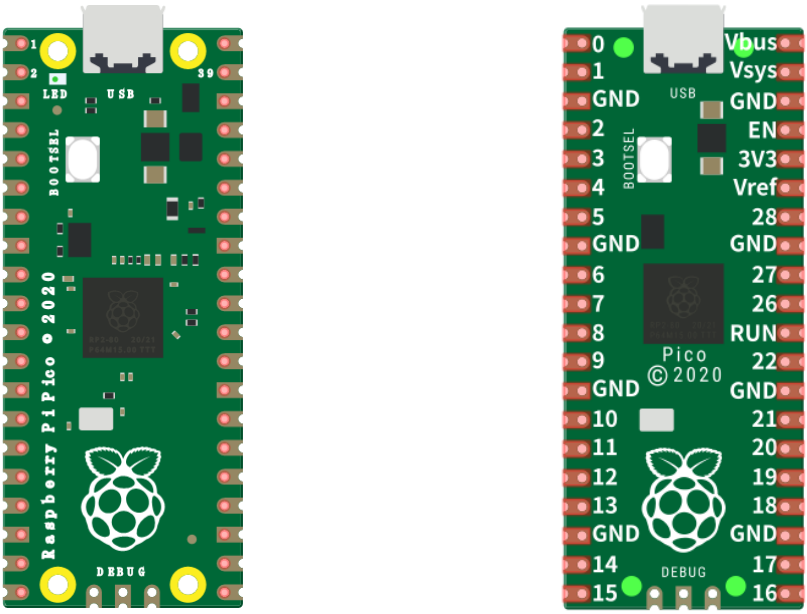
Paste the Sticker on the Breadboard

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)

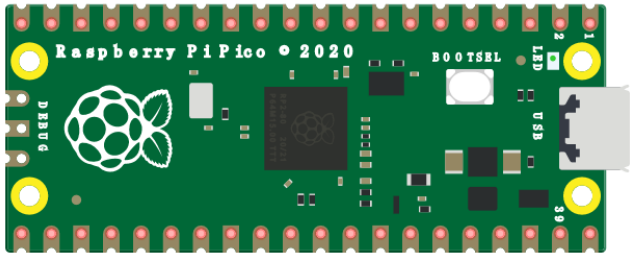
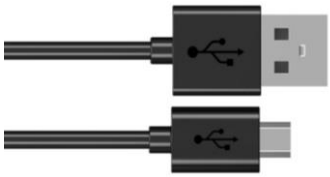
This chapter is the Start Point in the journey to build and explore Pico electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

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

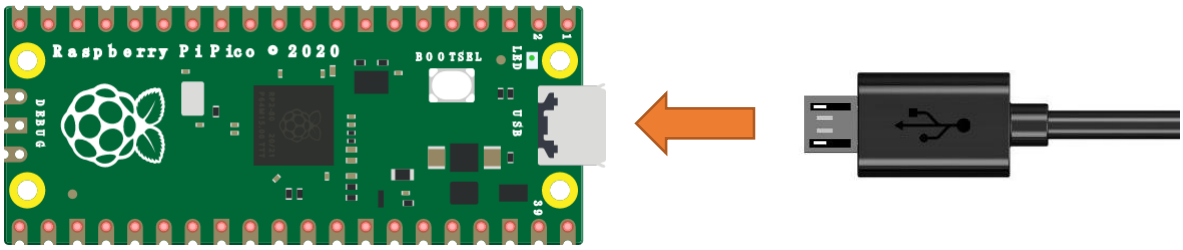
If you haven’t installed Arduino IDE, you can click Here.
If you haven’t uploaded firmware for Pico, you can click Here to upload.

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.

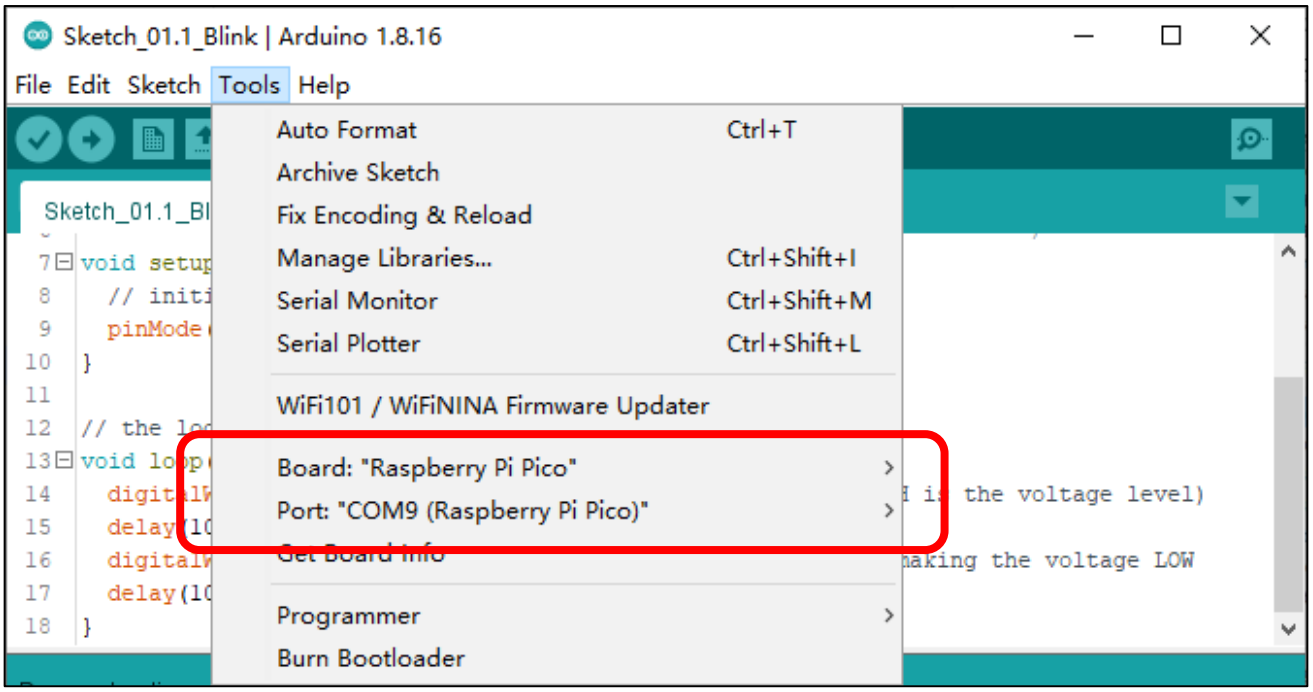


Sketch

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_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.1_Blink**.

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

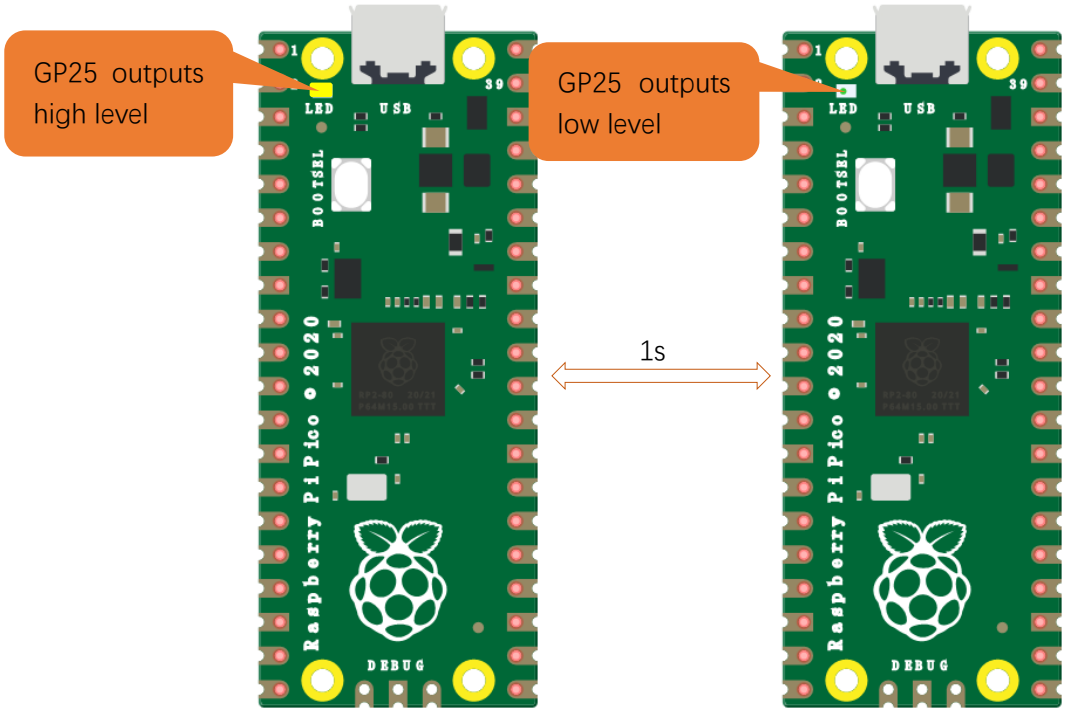
Click Tools, make sure Board and Port are as follows:



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.


```
// the setup function runs once when you press reset or power the board
void setup() {
  ...
}

// the loop function runs over and over again forever
void loop() {
  ...
}
```

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);

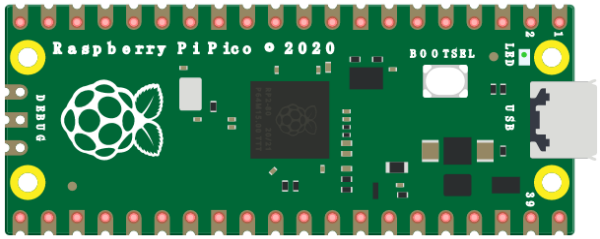
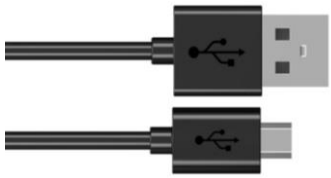
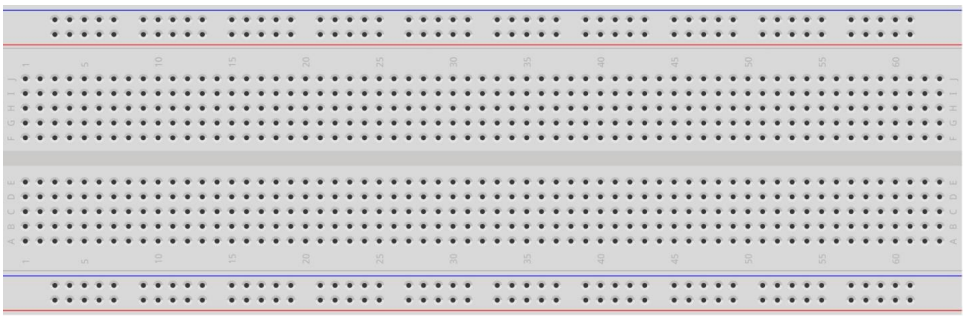
Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

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

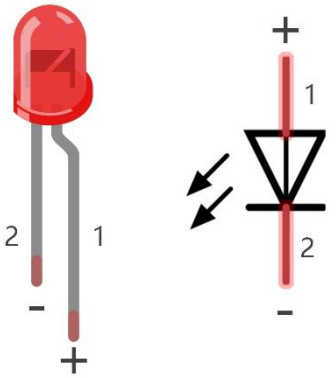
Raspberry Pi Pico x1 	USB Cable x1 	
Breadboard x1 		
LED x1 	Resistor 220Ω x1 	Jumper 

Component Knowledge

LED

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

All common 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.



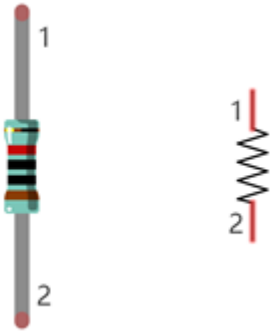
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

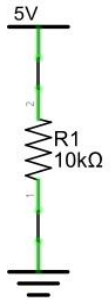
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

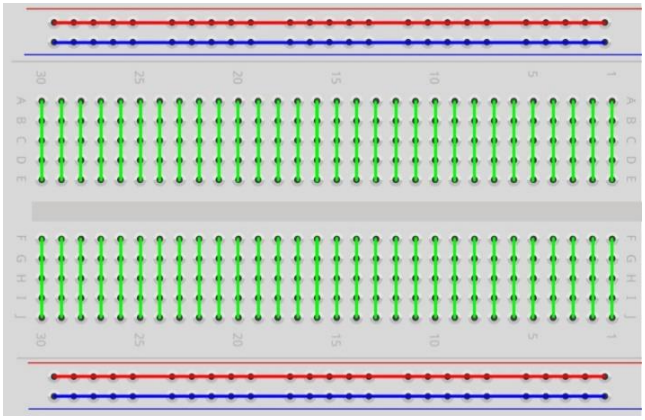


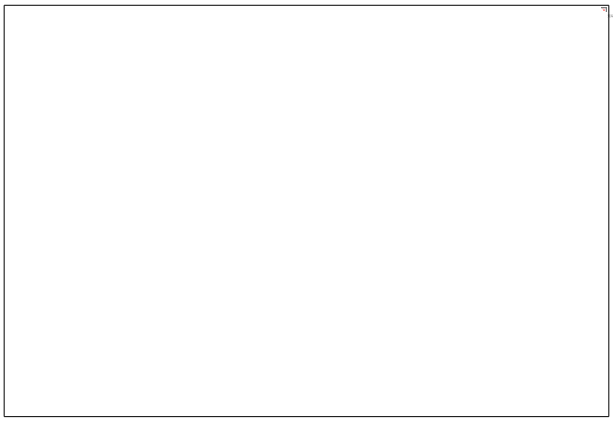
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

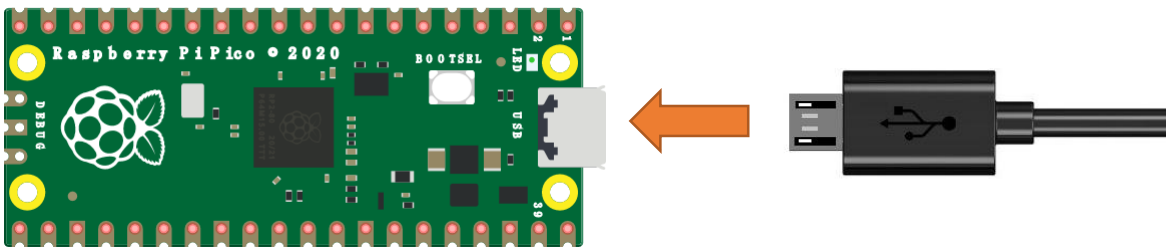
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

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

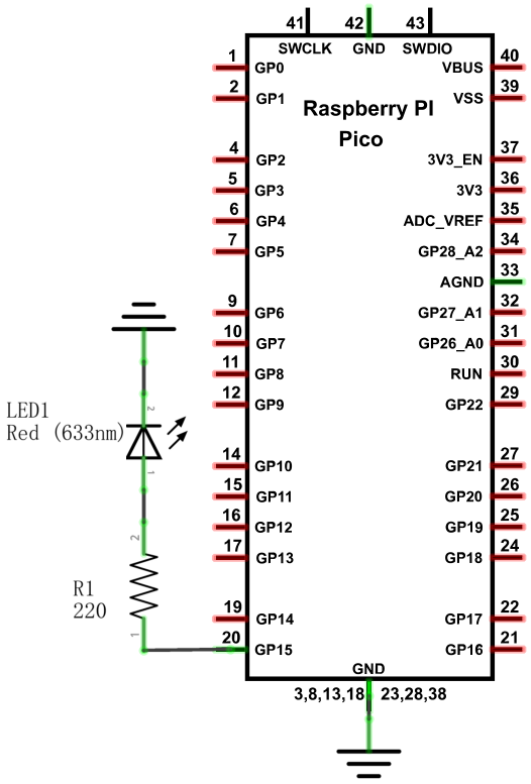


Circuit

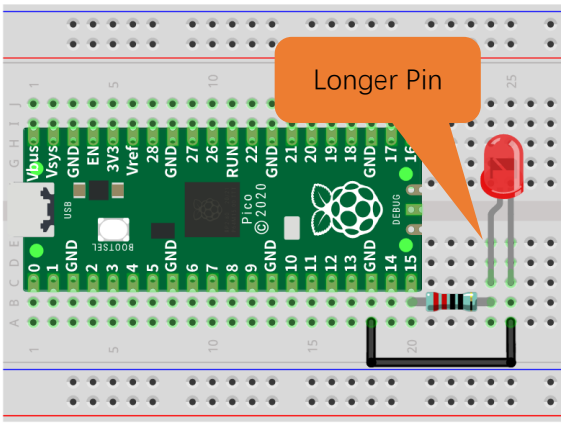
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

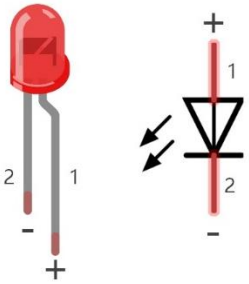
CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

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.

Any concerns? ✉ support@freenove.com

Sketch

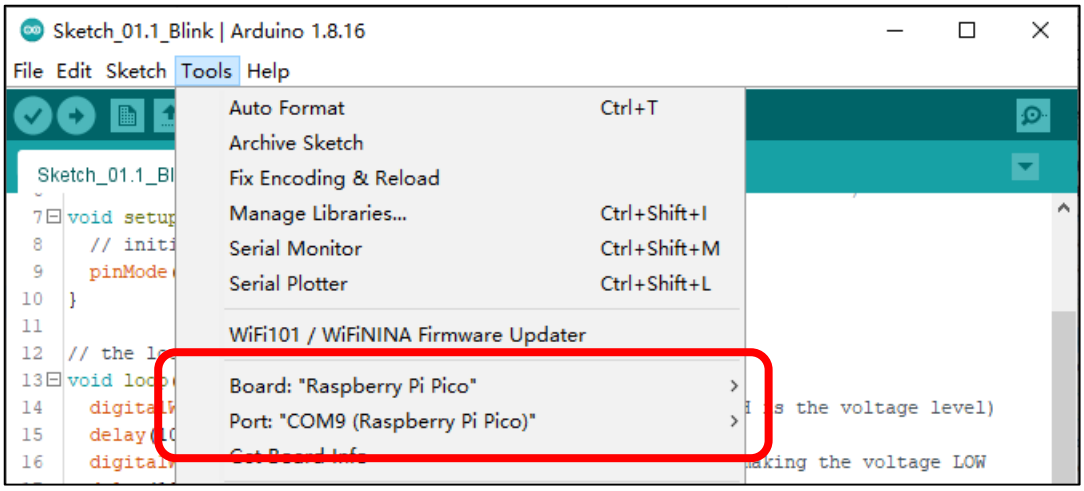
According to the circuit diagram, when GP15 of Pico outputs high level, LED lights up; when it outputs low, LED lights off. Therefore, we can make LED flash repeatedly by controlling GP15 to output high and low repeatedly.

You can open the provided code:

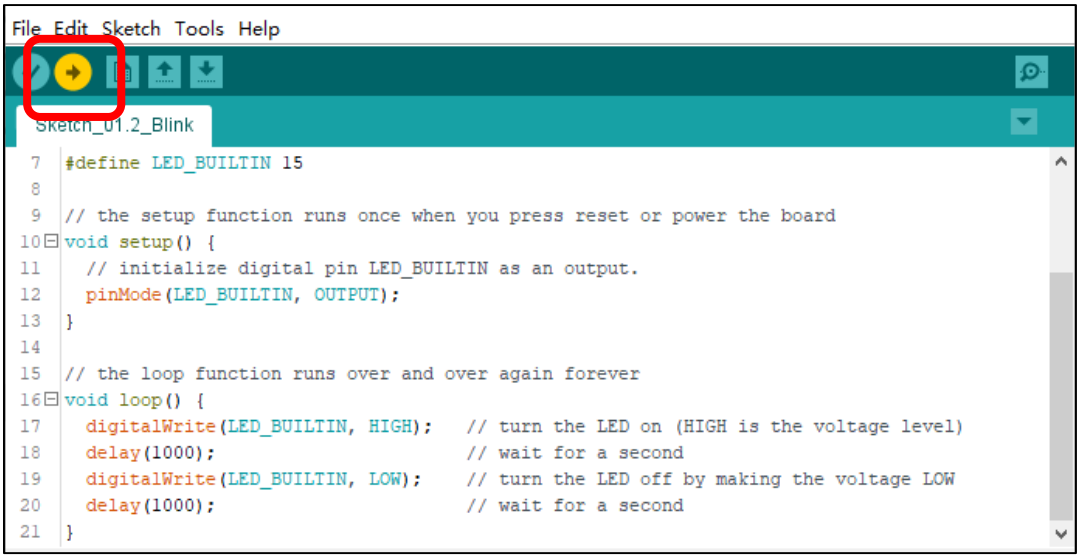
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_01.2_Blink.

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

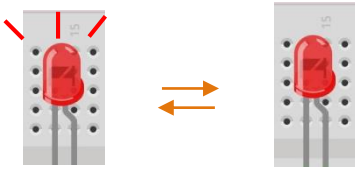
Click Tools, make sure Board and Port are as follows:



Click "Upload" to upload the sketch to Pico.



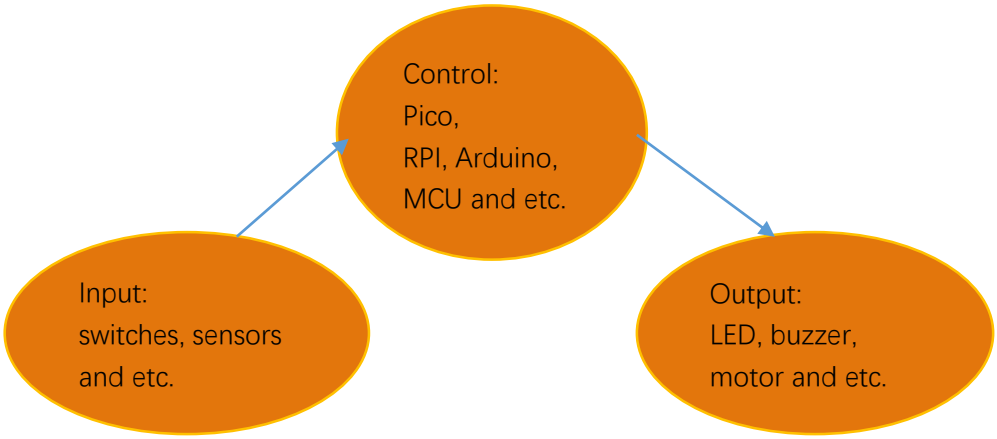
Click "Upload". Download the code to Pico and your LED in the circuit starts Blink.



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

Chapter 2 Button & LED

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

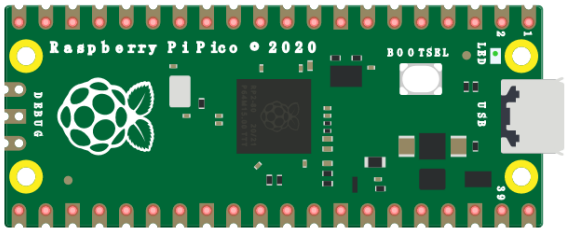
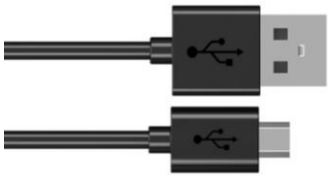
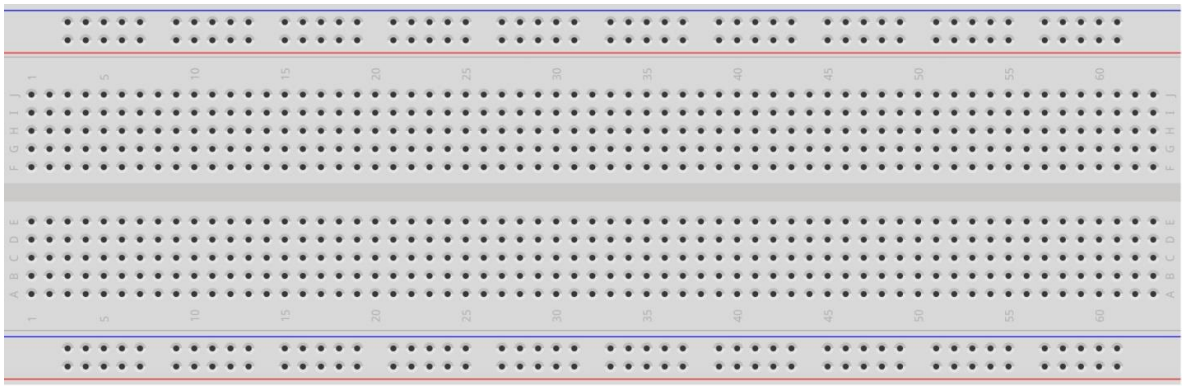


Next we make a simple project: build a control system with button, LED and Raspberry Pi Pico.
Input: Button
Control: Raspberry Pi Pico
Output: LED

Project 2.1 Button & LED

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.

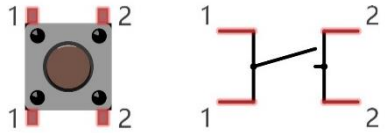
Component List

<table>
<tr><td colspan="3">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td>Jumper
  </td><td>LED x1
  </td><td>Resistor
220Ω x1
  </td><td>Resistor
10kΩ x2
  </td><td>Push button
x1
  </td></tr>
</table>

Component Knowledge

Push button

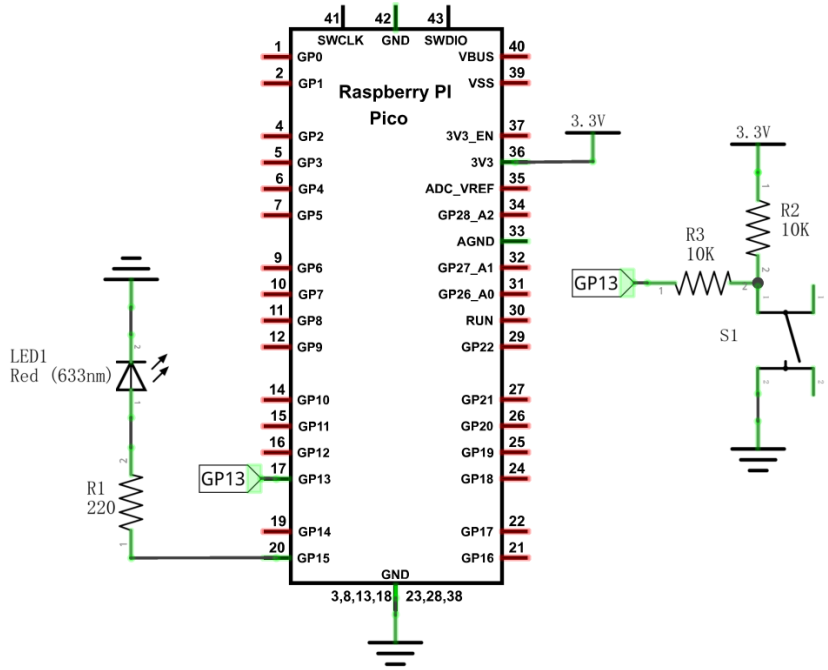
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



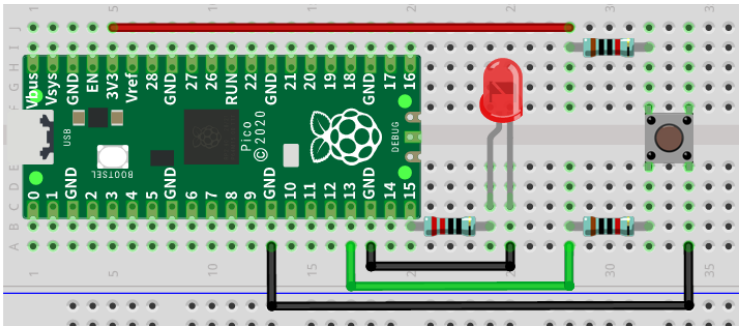
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

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.

Any concerns? ✉ support@freenove.com

Sketch

This project is designed for learning how to use push button switch to control an LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch.
Upload following sketch:
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.1_ButtonAndLed.
Sketch_02.1_ButtonAndLed

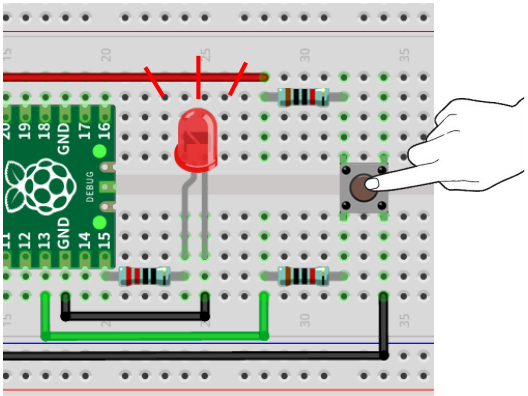
Sketch_02.1_ButtonAndLed | Arduino 1.8.12

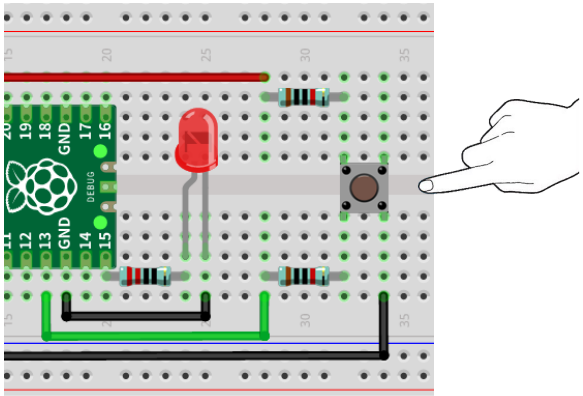
File Edit Sketch Tools Help

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);

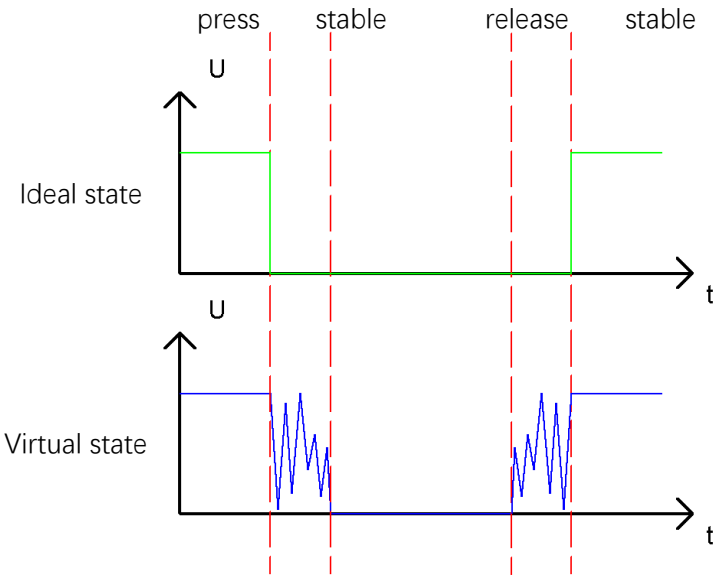
This function returns the value read at the given pin. It will be “HIGH” or “LOW”(1 or 0) depending on the logic level at the pin.

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_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Sketches\Sketch_02.2_TableLamp.
Sketch_02.2_TableLamp

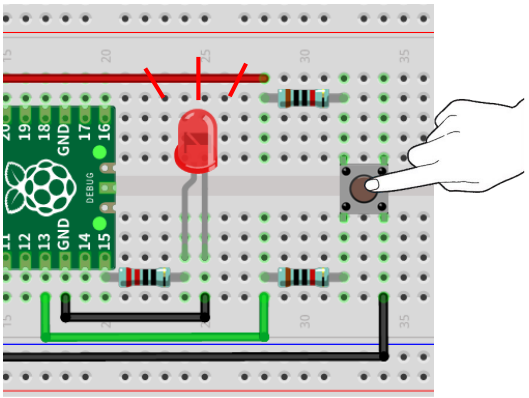
Sketch_02.2_TableLamp | Arduino 1.8.12

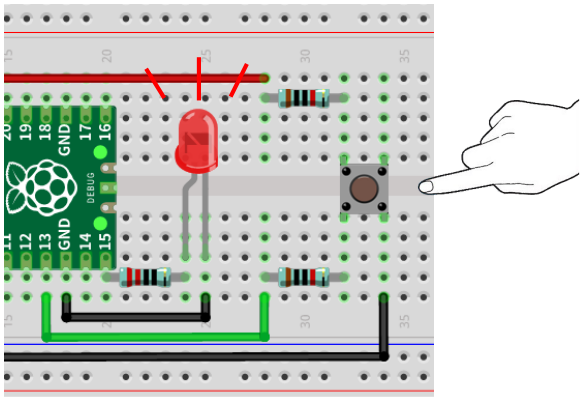
File Edit Sketch Tools Help

Sketch_02.2_TableLamp

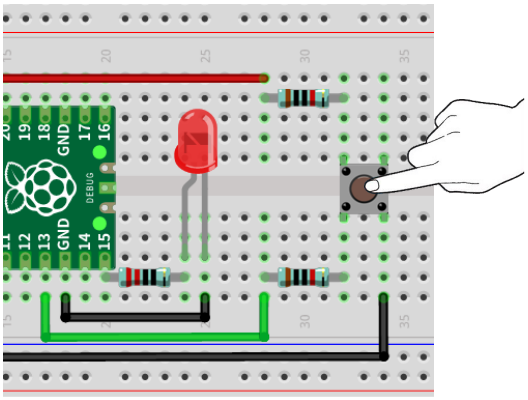
```
#define PIN_LED    15
#define PIN_BUTTON 13
bool led_state = false;

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);
}
```

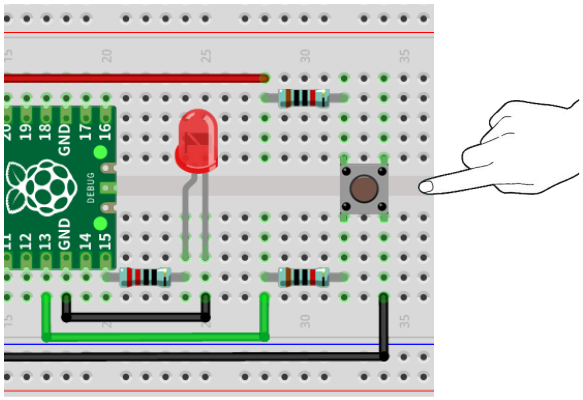
Upload the sketch to Pico. When the button is pressed, LED lights up; when the button is released, LED is still ON.





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  }
```

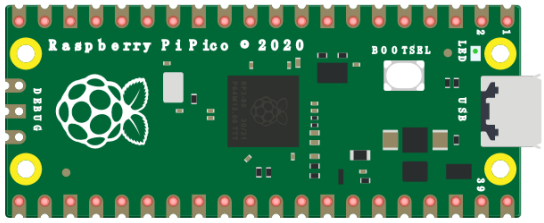
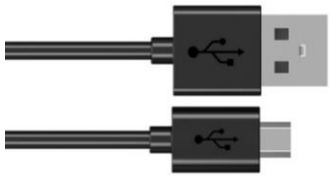
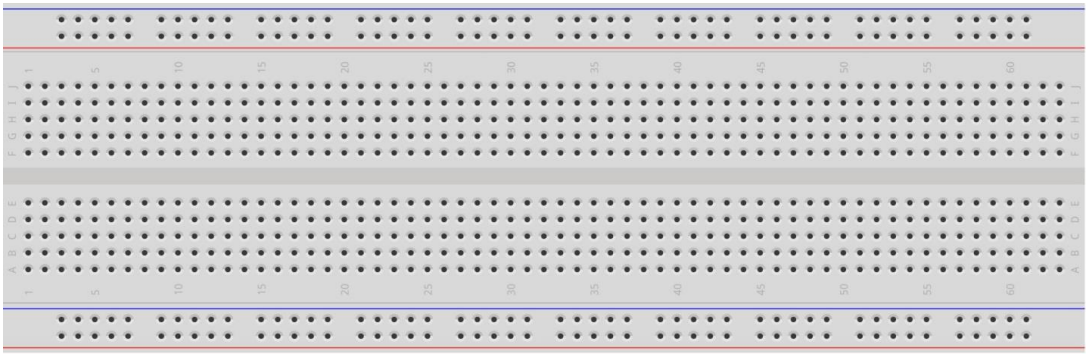
Chapter 3 LED Bar

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

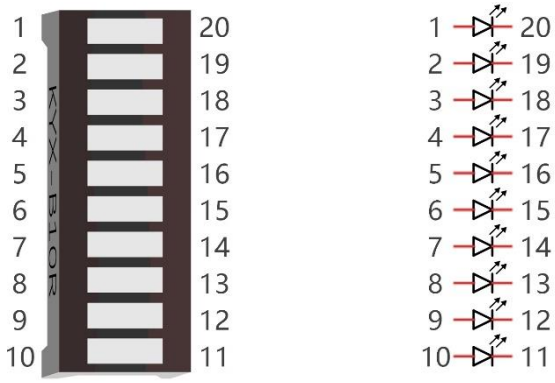
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Jumper 	LED bar graph x1 	Resistor 220Ω x10 

Component Knowledge

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

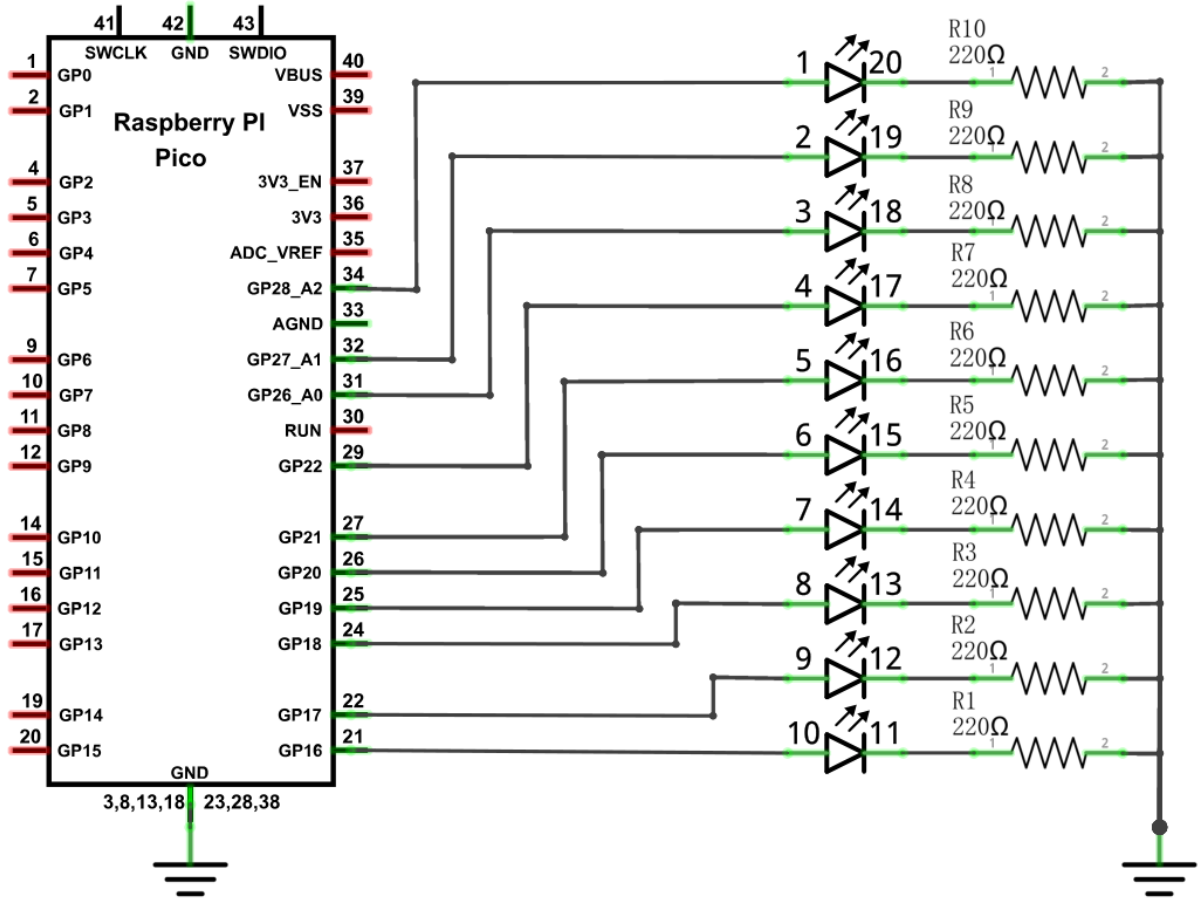
LED bar

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.

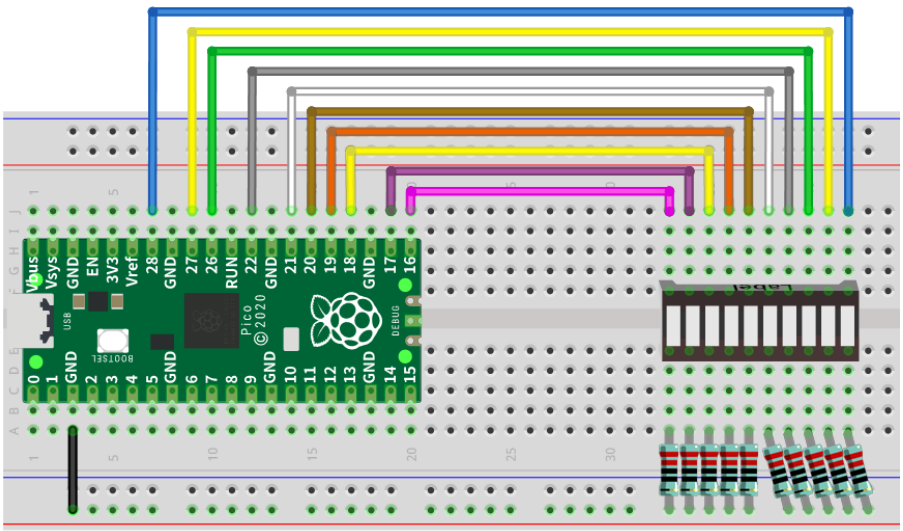


Circuit

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.

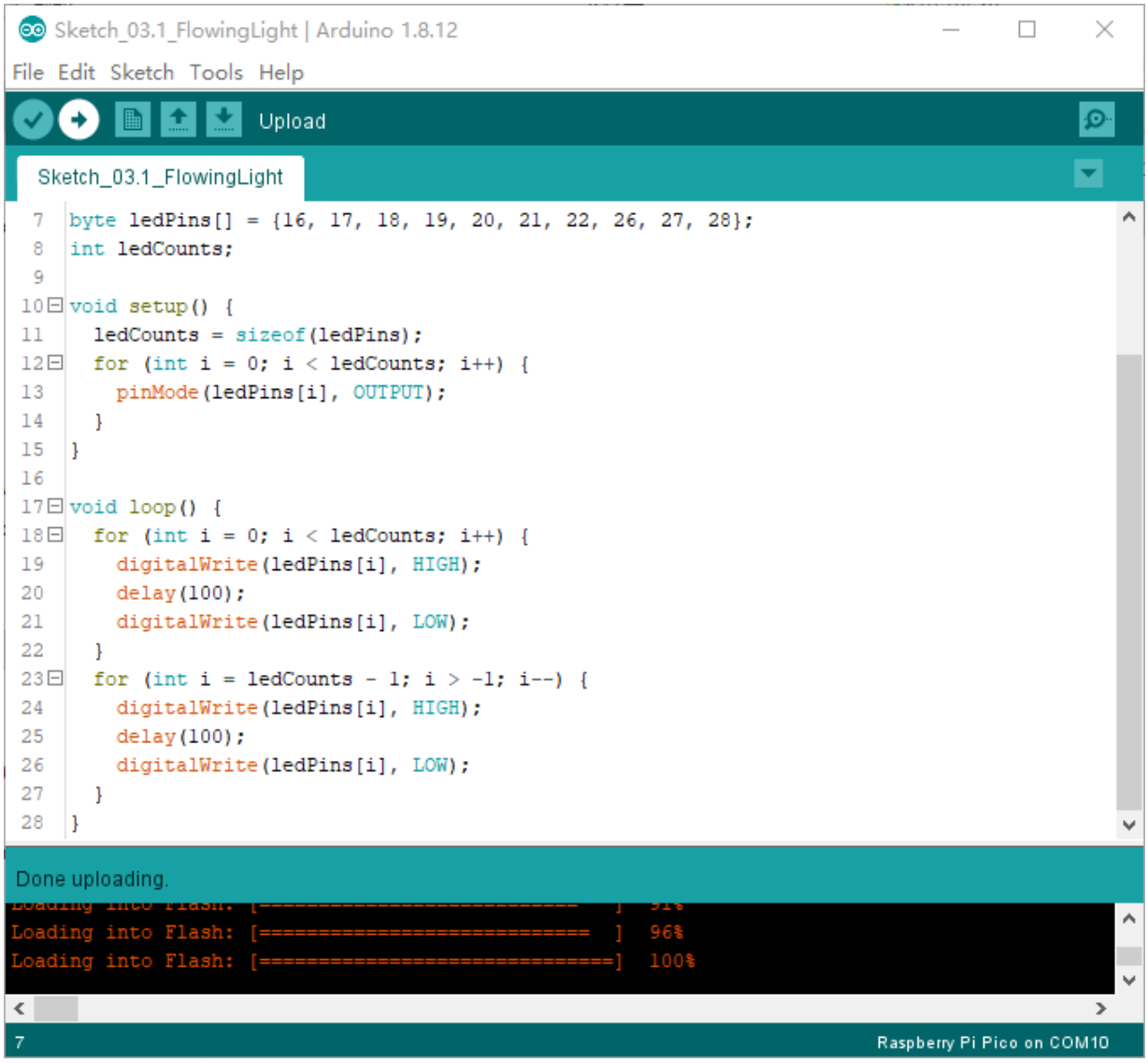
Sketch

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

Upload following sketch:

Freenove_Ultimate_Starter_Kit_for_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);
  }
```

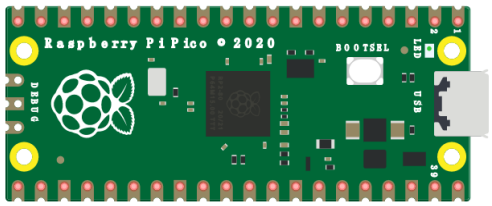
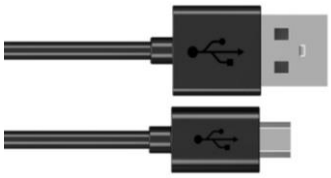
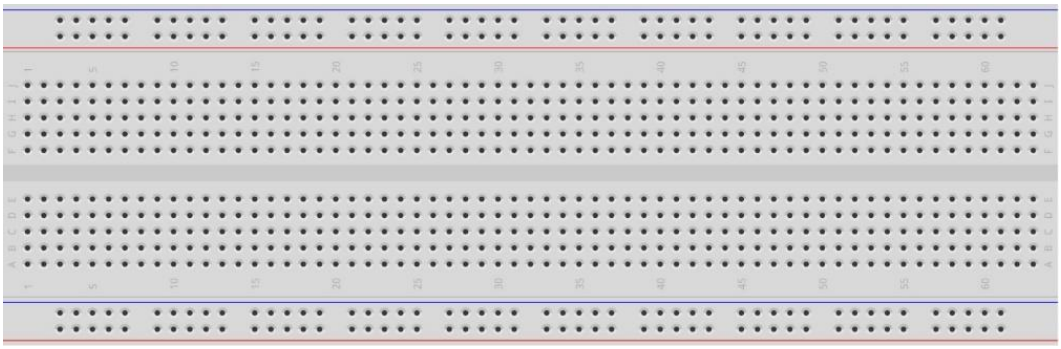
Chapter 4 Analog & PWM

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.

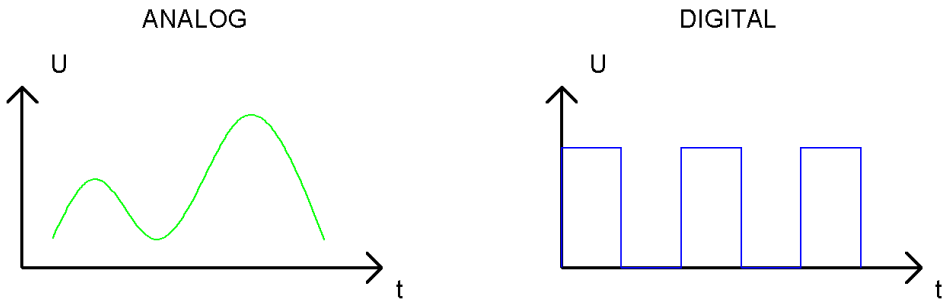
Component List

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

Related Knowledge

Analog & Digital

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



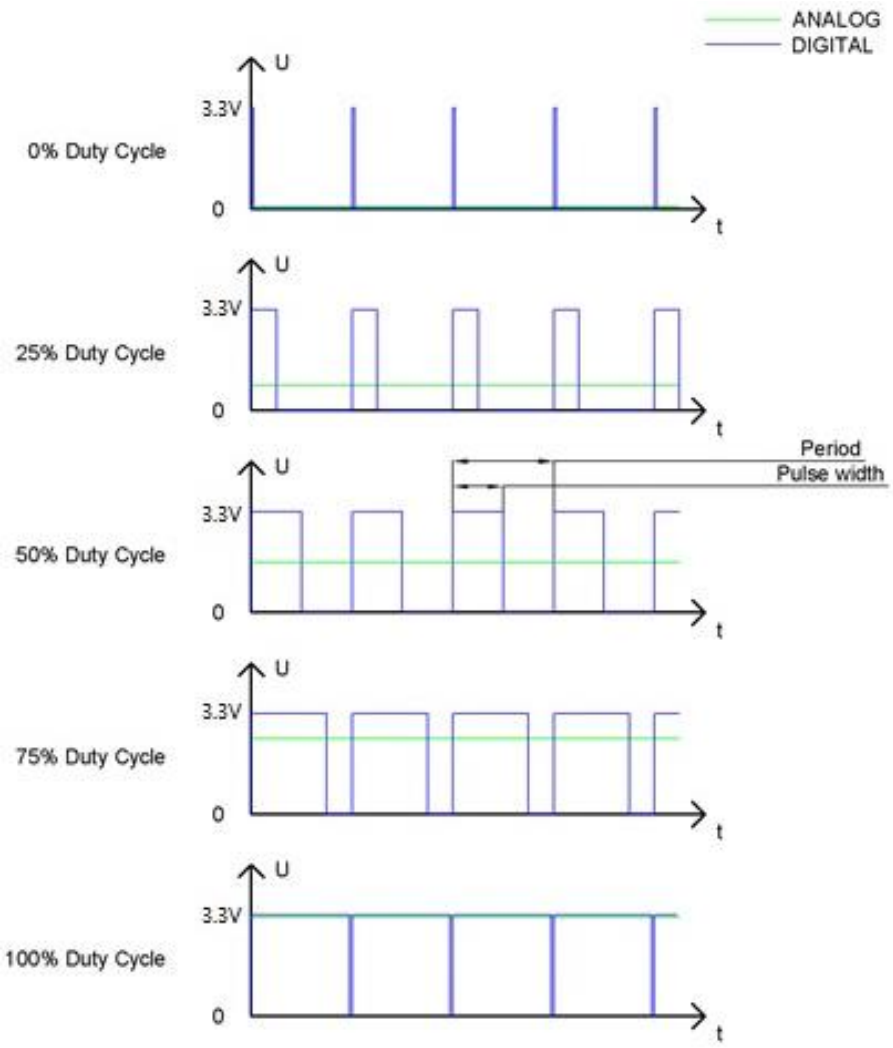
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

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

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

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



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

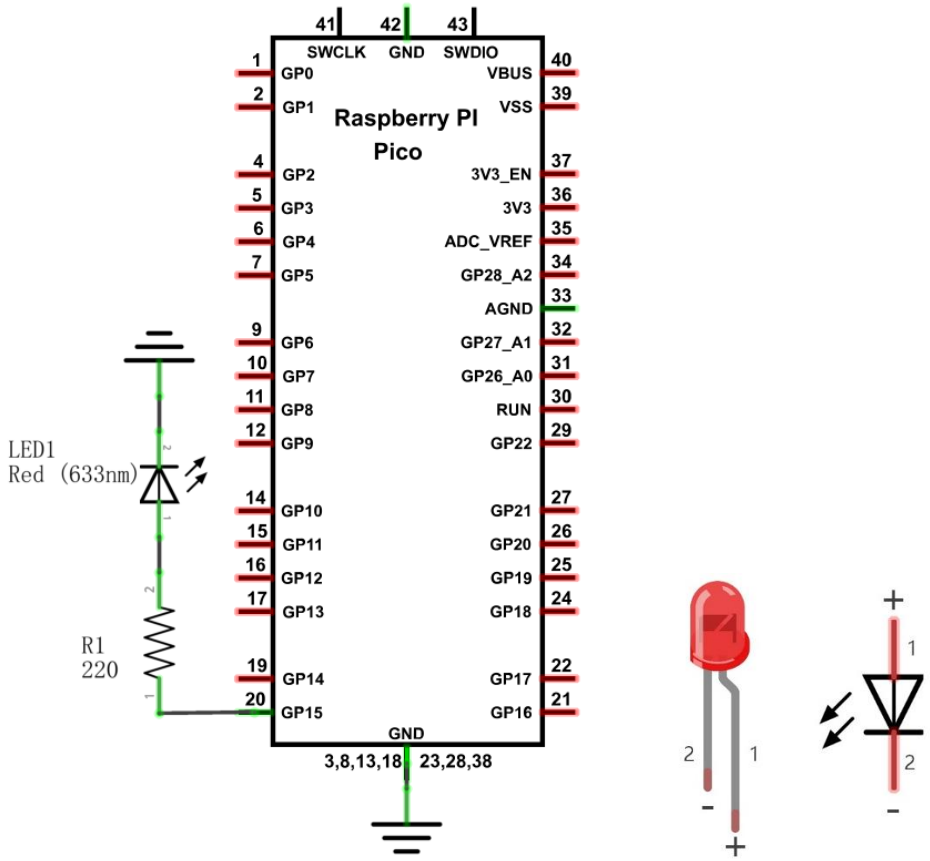
Raspberry Pi Pico and PWM

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

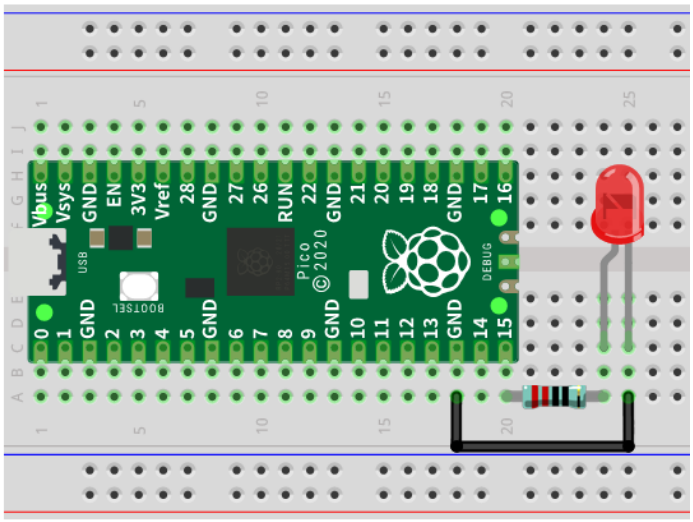
Circuit

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

Schematic diagram



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

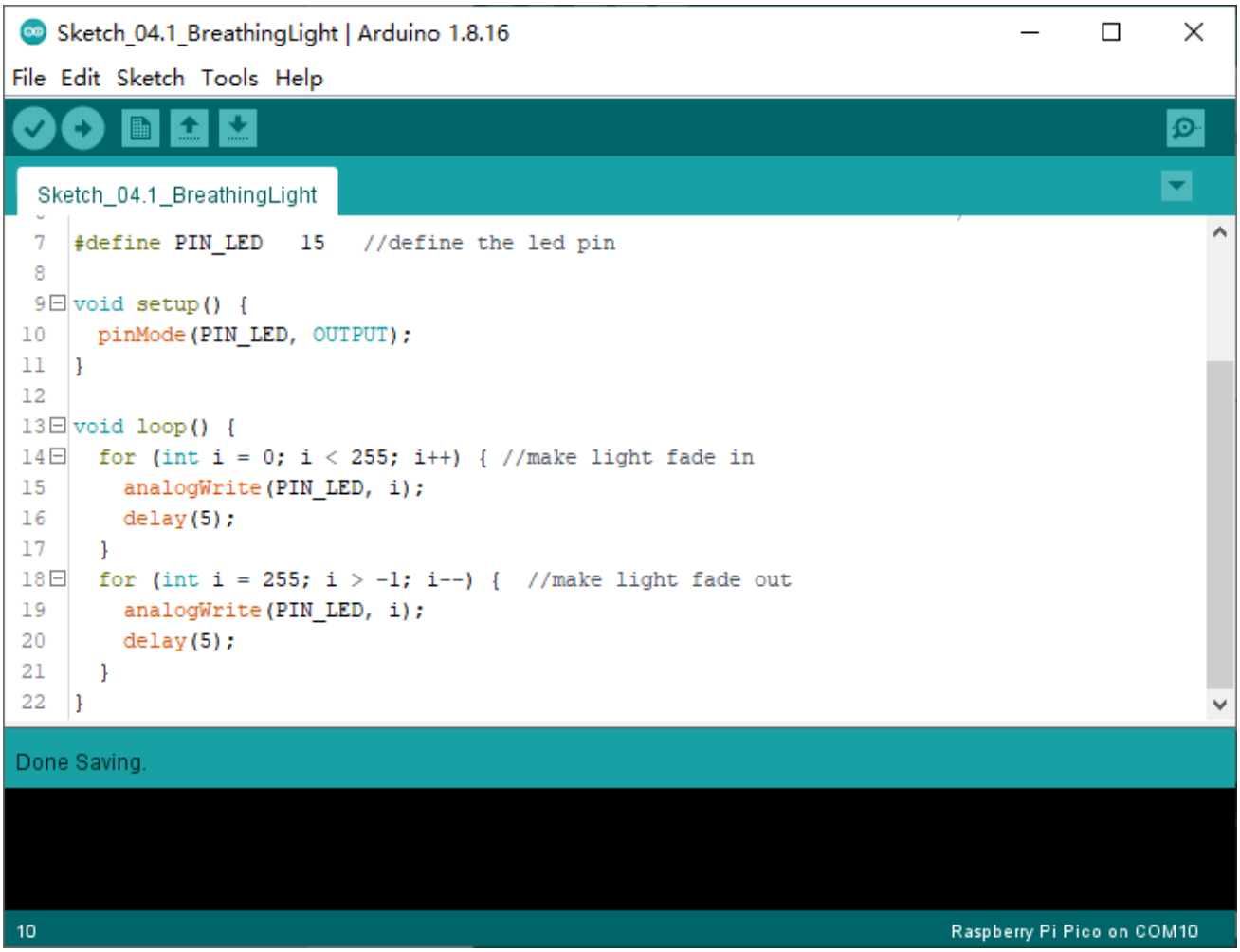


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Sketch

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

Sketch_04.1_BreathingLight



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

The following is the program code:

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

Set the pin controlling LED to output mode.

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

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

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

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

analogWrite(pin, value)

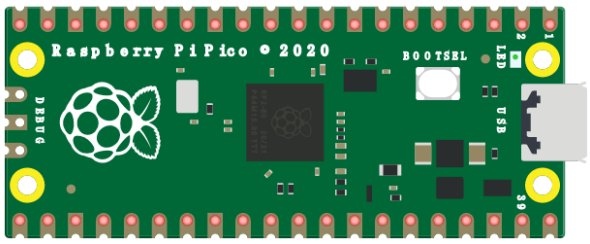
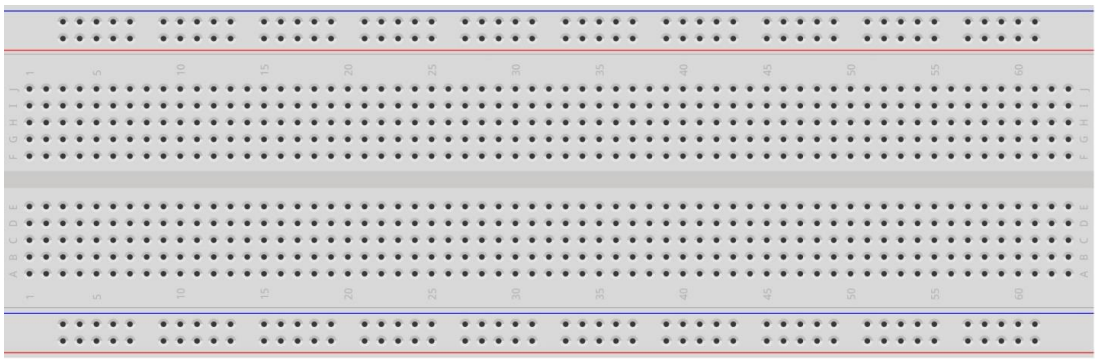
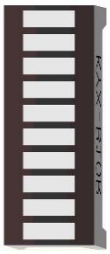
Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255,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

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

Project 4.2 Meteor Flowing Light

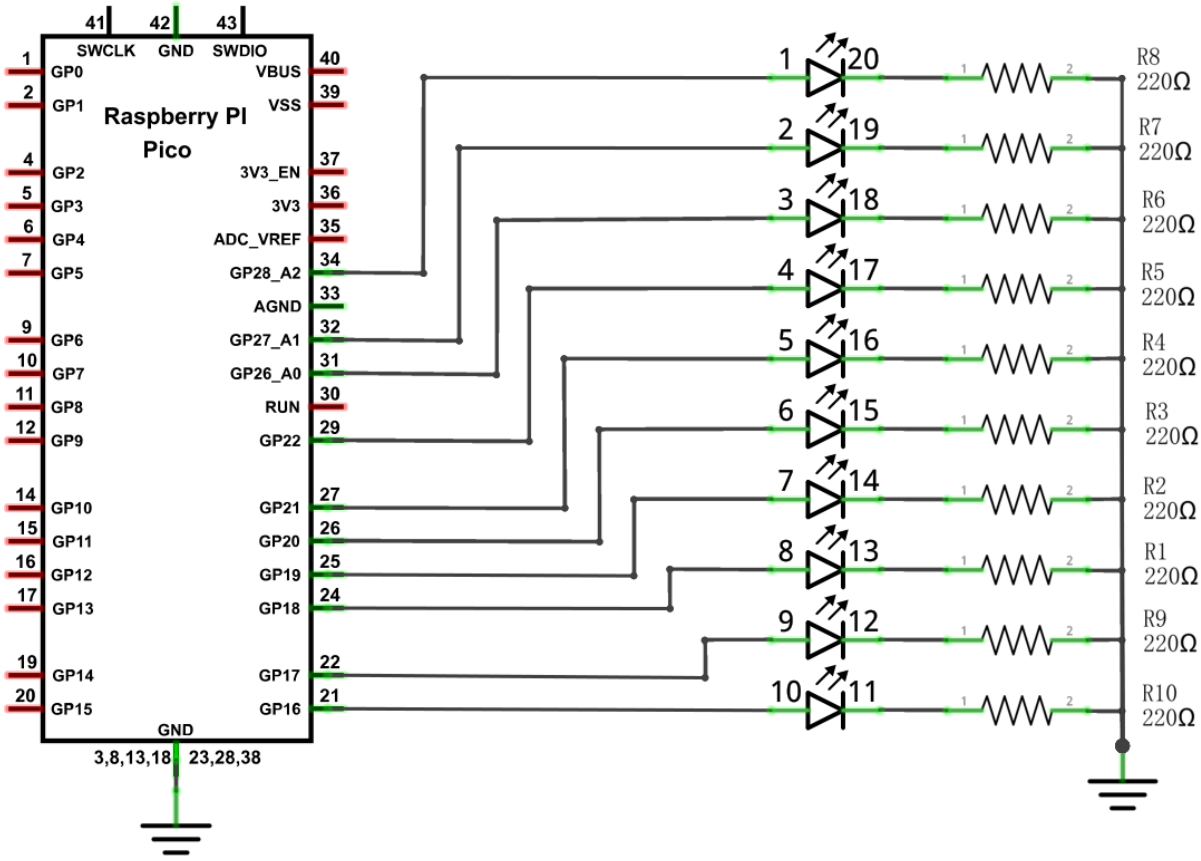
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light. The component list, circuit, and hardware are exactly consistent with the project Flowing Light.

Component List

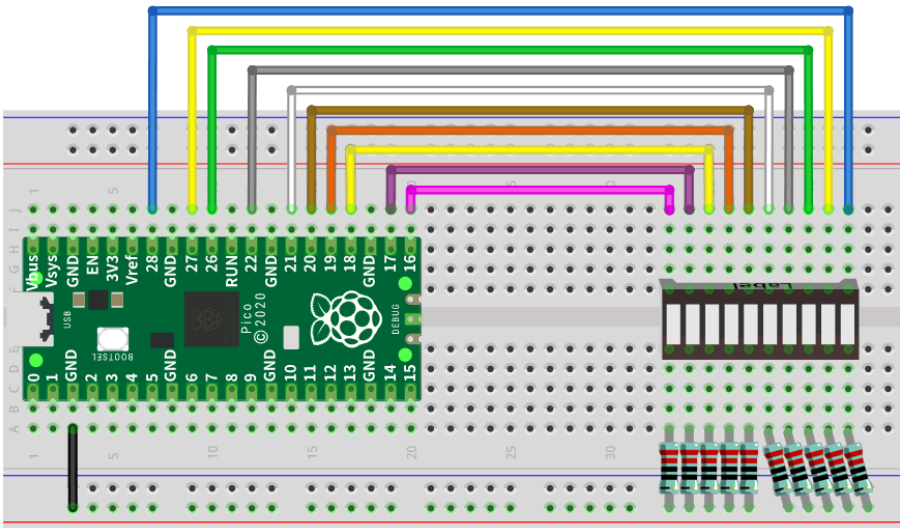
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Jumper 	LED bar graph x1 	Resistor 220Ω x10 

Circuit

Schematic diagram



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



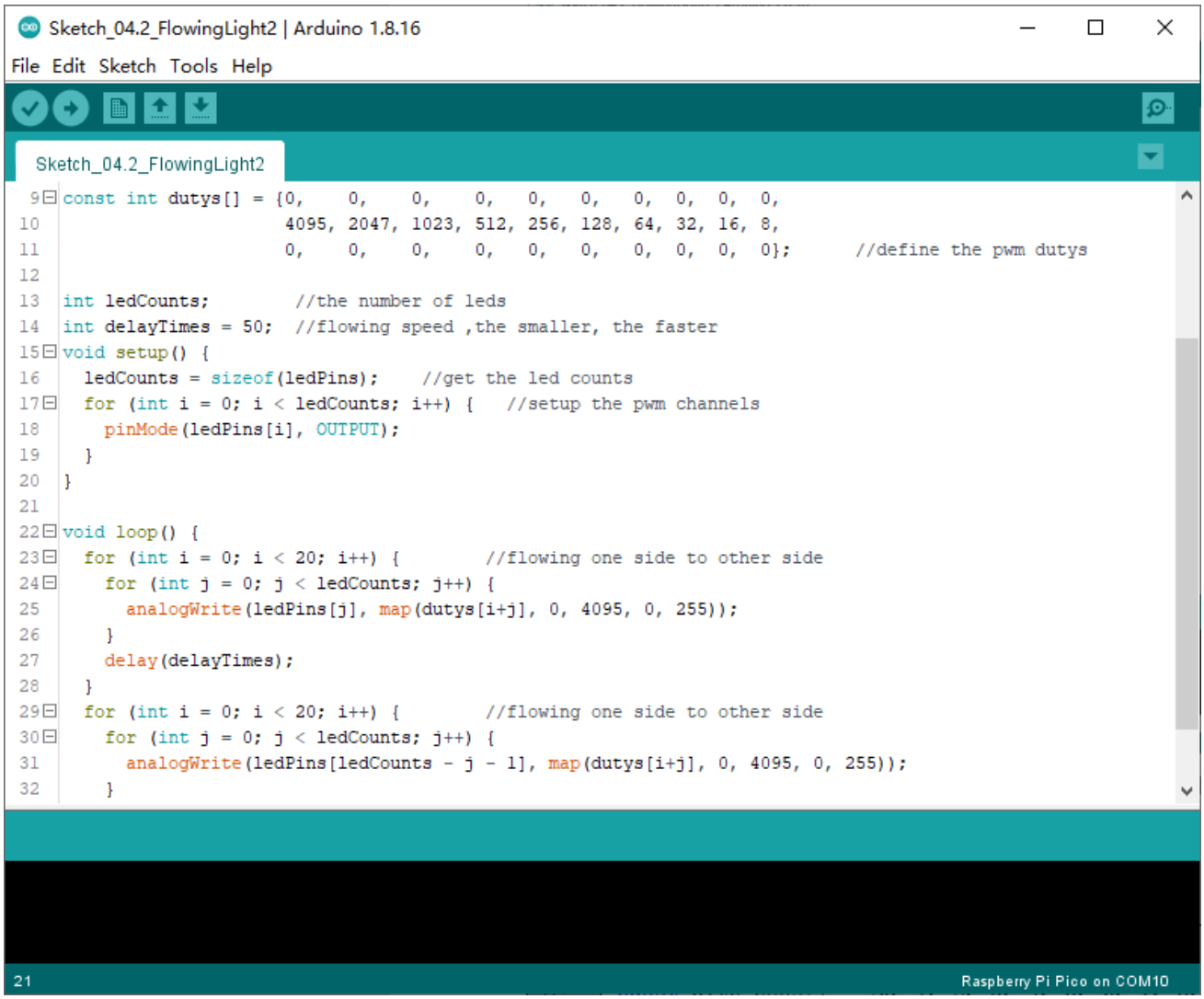
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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

Sketch

Meteor flowing light will be implemented with PWM.

Sketch_04.2_FlowingLight2




```
const int dutys[] = {0,    0,    0,    0,    0,    0,    0,  0,  0,  0,
                     4095, 2047, 1023, 512,  256,  128,  64, 32, 16, 8,
                     0,    0,    0,    0,    0,    0,    0,  0,  0,  0};      //define the pwm dutys

int ledCounts;         //the number of leds
int delayTimes = 50;  //flowing speed ,the smaller, the faster
void setup() {
  ledCounts = sizeof(ledPins);    //get the led counts
  for (int i = 0; i < ledCounts; i++) {   //setup the pwm channels
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
    }
    delay(delayTimes);
  }
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
    }
```

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() {
```

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

void loop() {
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
    }
    delay(delayTimes);
  }
  for (int i = 0; i < 20; i++) {        //flowing one side to other side
    for (int j = 0; j < ledCounts; j++) {
      analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
    }
    delay(delayTimes);
  }
}
```

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

```
const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};     //define led pins

const int dutys[] = {0,    0,    0,    0,    0,    0,    0,   0,   0,   0,
                     4095, 2047, 1023, 512,  256,  128,  64,  32,  16,  8,
                     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.

```
int ledCounts;        //the number of leds
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.

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

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

0	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
d i	0	0	0	0	0	0	0	0	0	1023	512	256	128	64	32	16	8	4	2	0	0	0	0	0	0	0	0	0	0
0																													
1																													
2																													
3																													
...																													
18																													
19																													
20																													

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

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

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

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

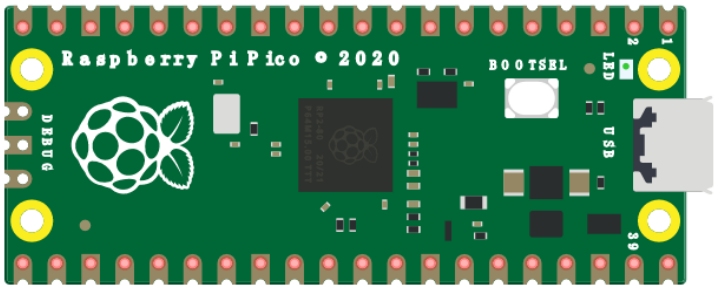
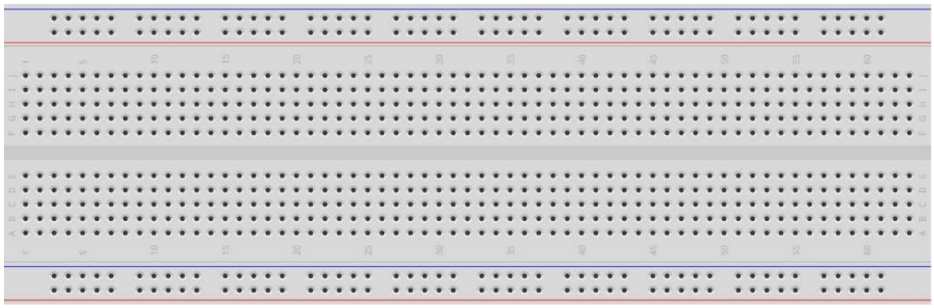
Chapter 5 RGBLED

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

Project 5.1 Random Color Light

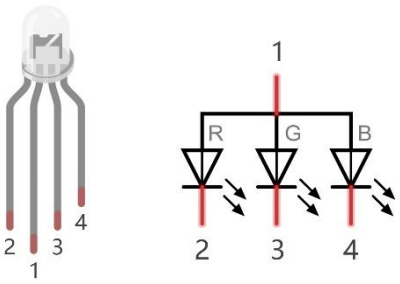
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

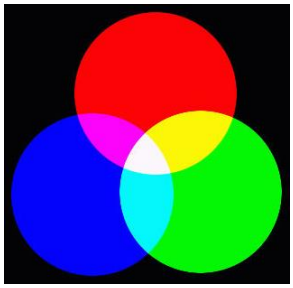
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>RGBLED x1
  </td><td>Resistor 220Ω x3
  </td><td>Jumper
  </td></tr>
</table>

Related Knowledge

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



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

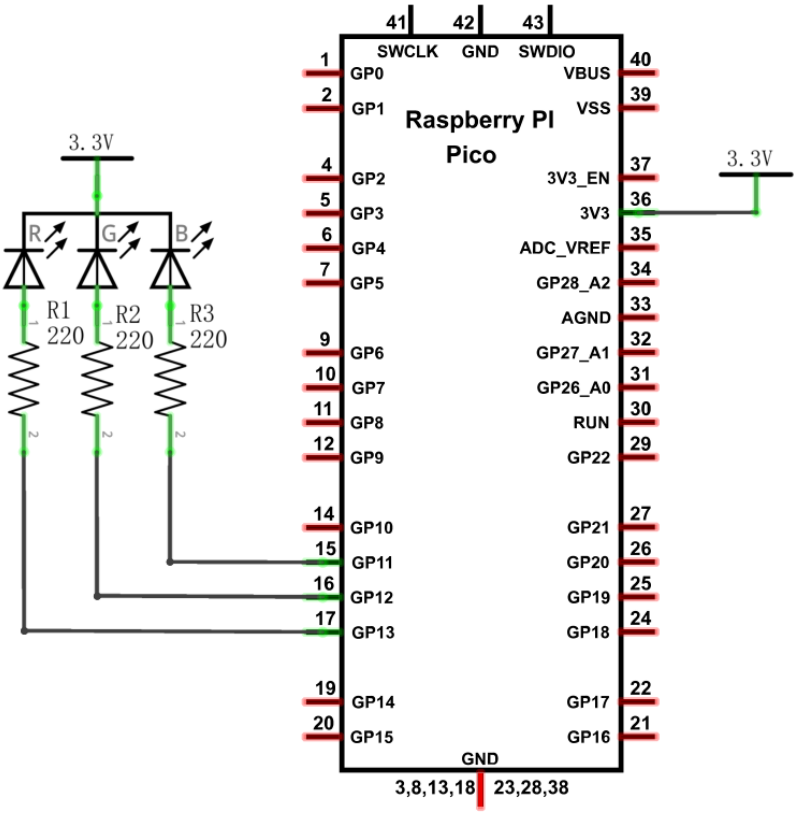


RGB

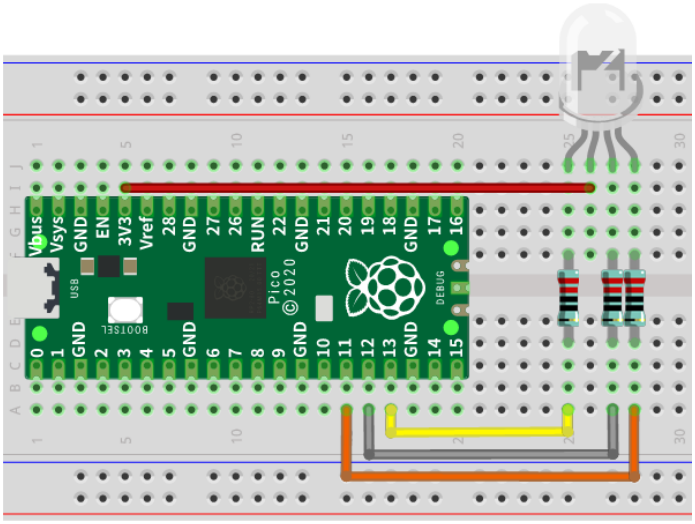
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations.

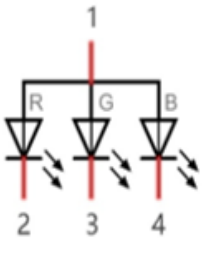
Circuit

Schematic diagram



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



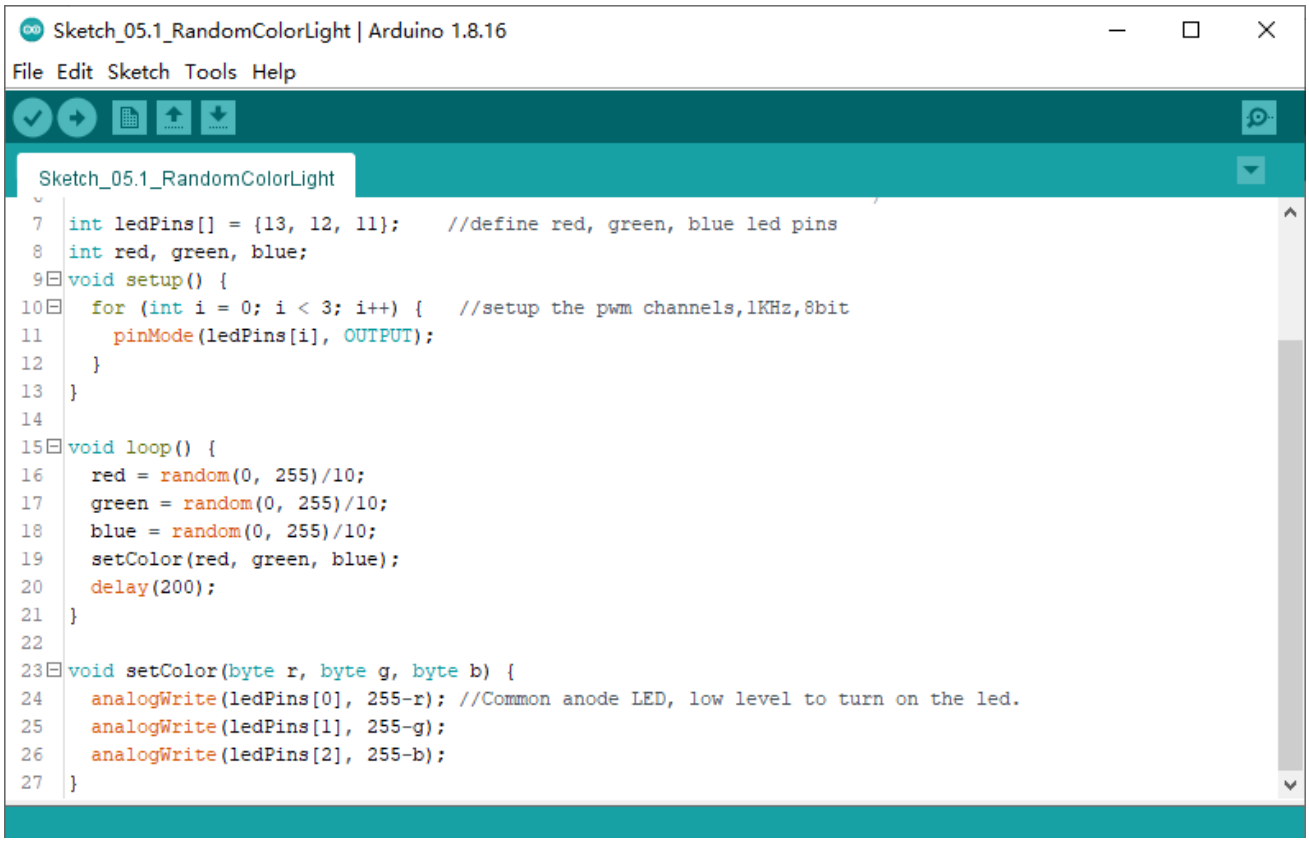


Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

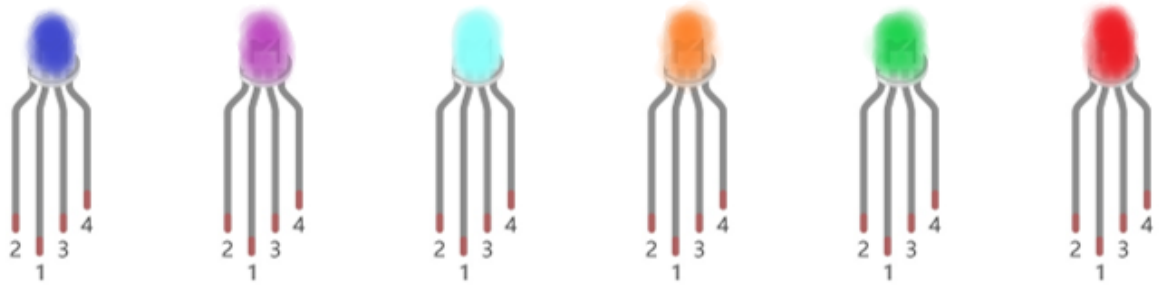
Sketch

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

Sketch_05.1_ColorfulLight



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



The following is the program code:

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

```

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

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

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

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

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

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

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

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

long random(min, max);

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

Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff. This project will realize a fashionable Light with soft color changes.

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

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

Sketch

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

Sketch_05.2_SoftColorfulLight

The following is the program code:

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

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

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

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

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

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

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

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

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

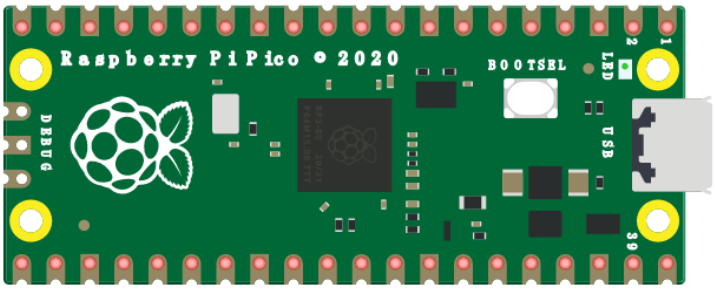
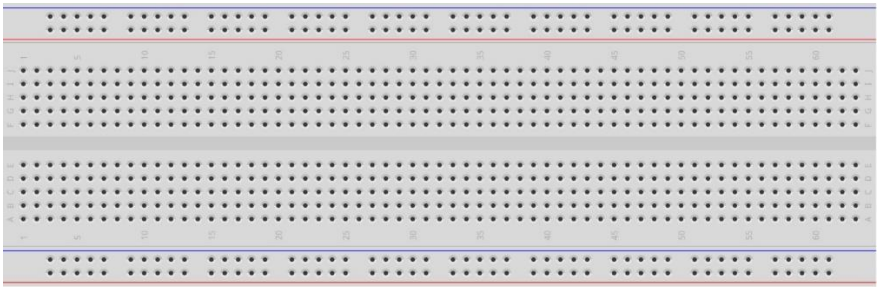
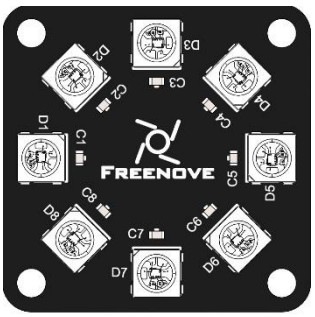
Chapter 6 NeoPixel

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

Project 6.1 NeoPixel

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

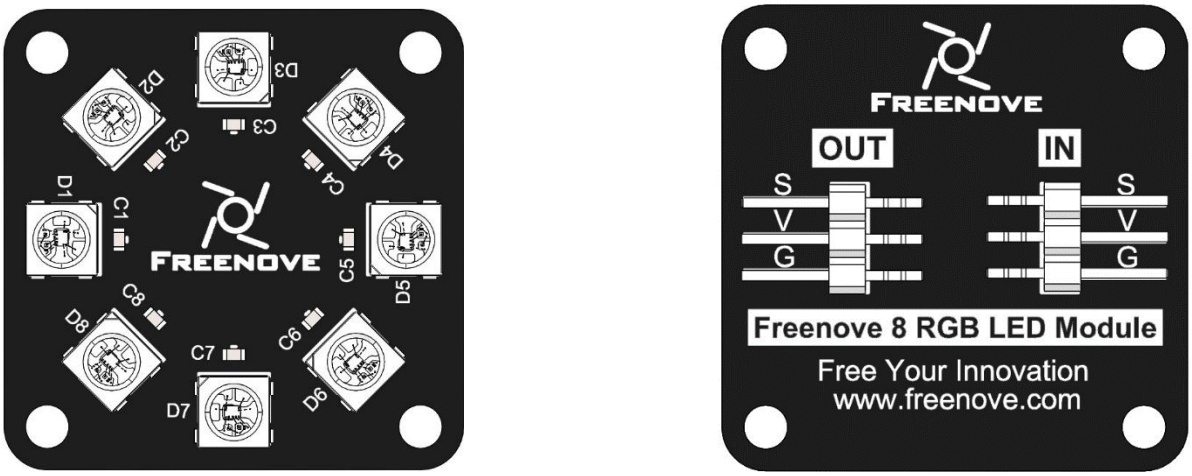
Component List

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

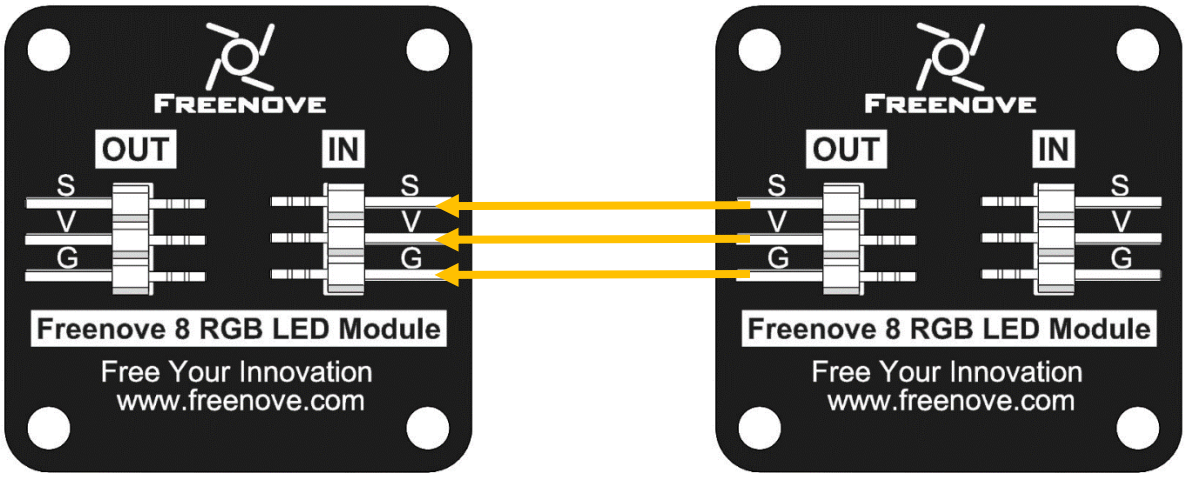
Related Knowledge

Freenove 8 RGB LED Module

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



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

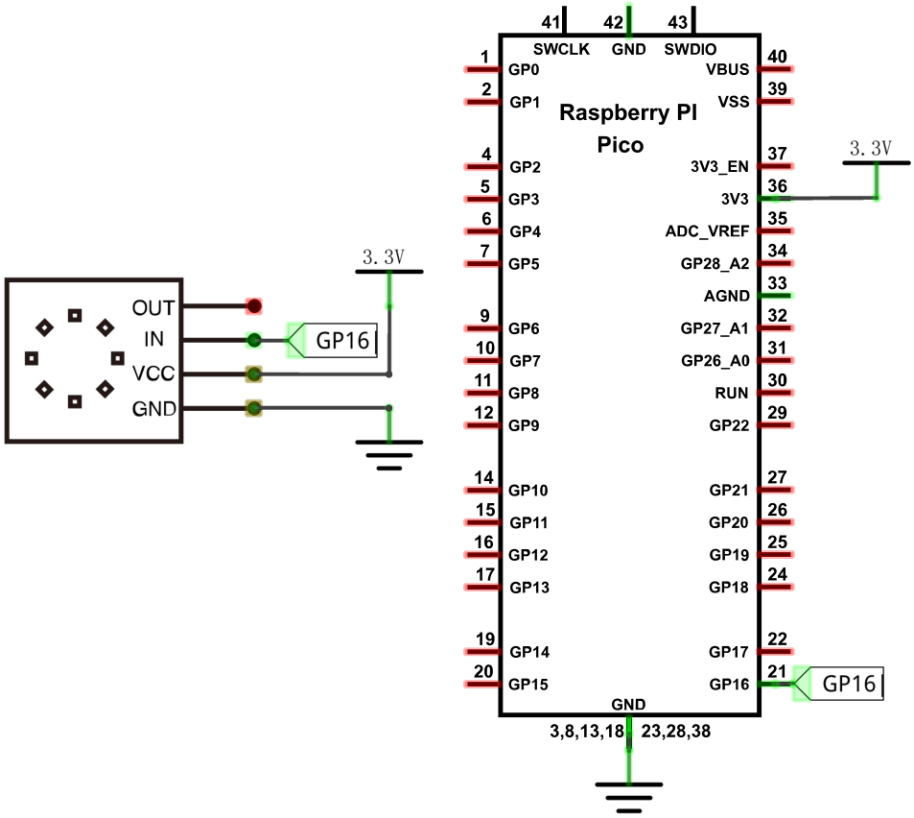


Pin description:

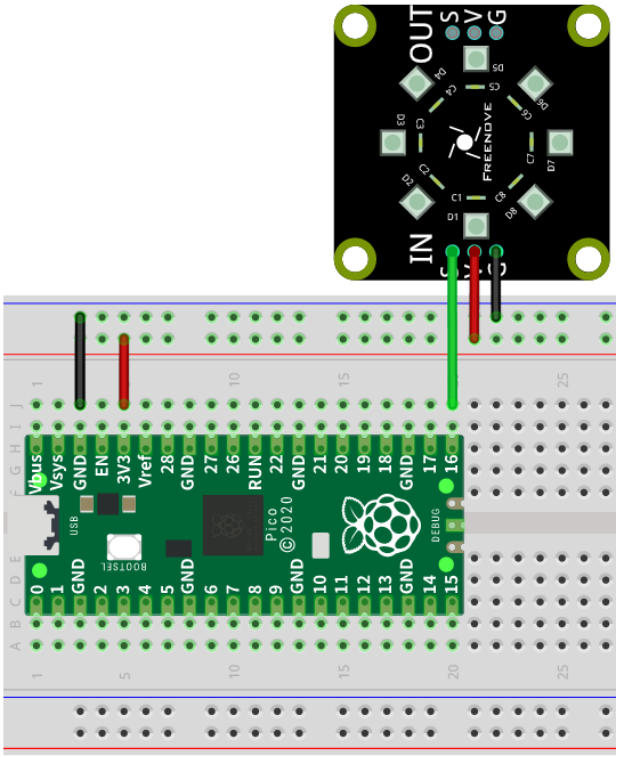
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.3V~5.5V	V	Power supply pin, +3.3V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



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

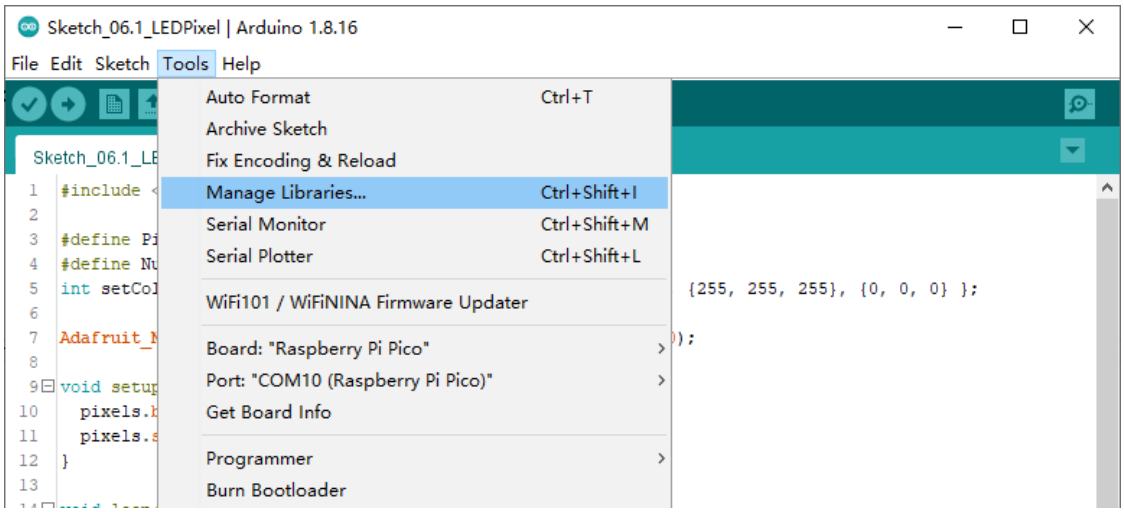


Sketch

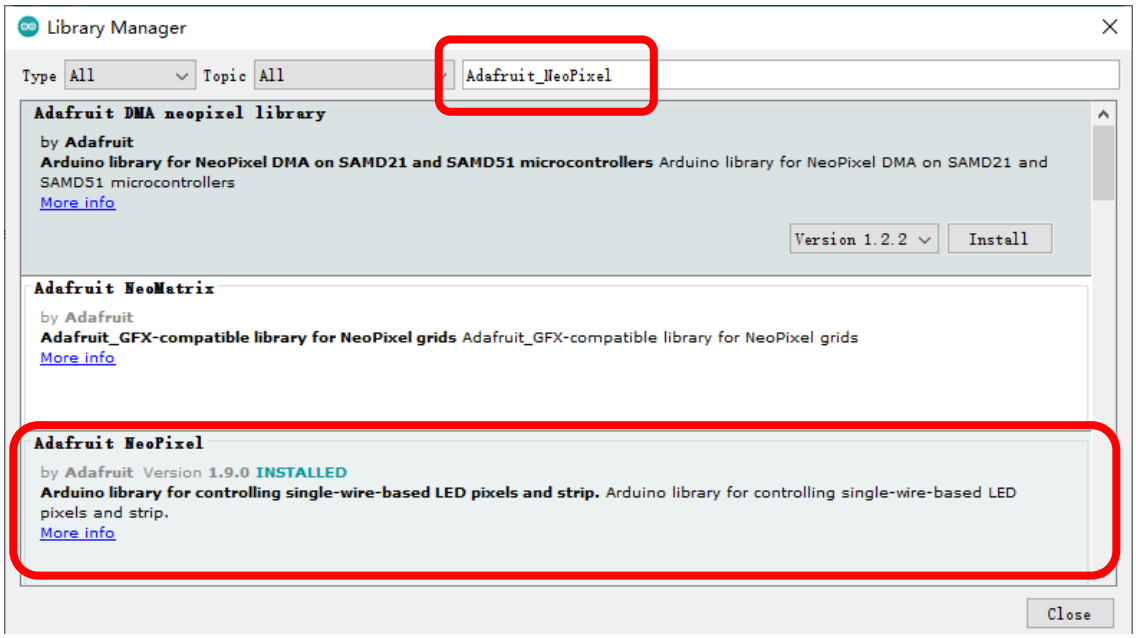
This code uses a library named "**Adafruit_NeoPixel**". If you have not installed it, please do so first.
Library is an important feature of the open source world, and we know that Arduino is an open source platform that everyone can contribute to. Libraries are generally licensed under the LGPL, which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

How to install the library

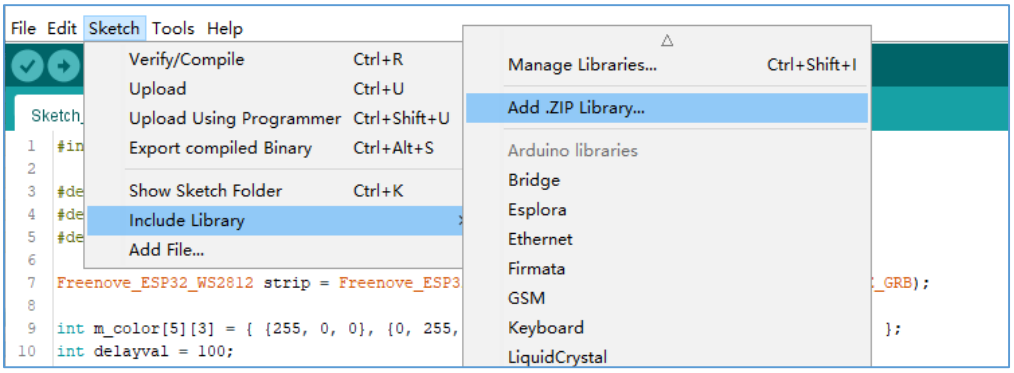
There are two ways to add libraries.
The first way, open the Arduino IDE, click Tools →Manager Libraries.



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

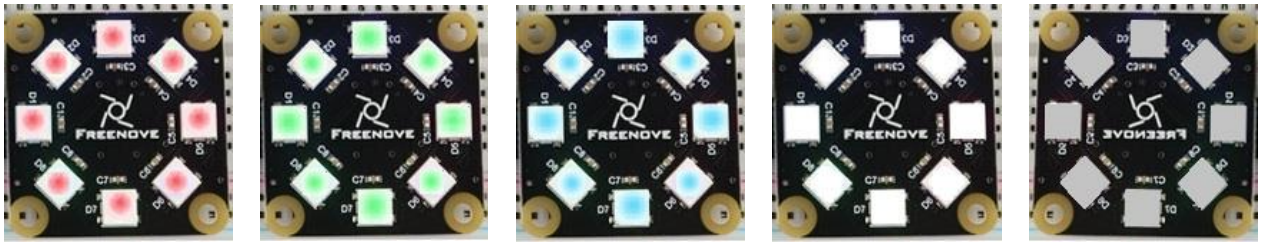


The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library. In the pop-up window, find the file named "./Libraries/ **Adafruit_NeoPixel.Zip**" which locates in this directory, and click OPEN.



Sketch_06.1_LEDPixel

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



The following is the program code:

```
#include <Adafruit_NeoPixel.h>

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

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

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

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

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

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

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

```
#define Pin        16
#define NumPixels 8
```

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

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

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

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

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

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

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

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

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

Reference

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

Constructor to create a NeoPixel object.

Before each use of the constructor, please add "Adafruit_NeoPixel.h"

Parameters

n: The number of led.

pin_gpio: A pin connected to an LED.

type: Types of LED.

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

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

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

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

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

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

`void begin(void);`

Initialize the NeoPixel object

```
void setPixelColor (u8 index, u8 r, u8 g, u8 b);
void setPixelColor (u8 index, u32 rgb);
void setPixelColor (u8 index, u8 r, u8 g, u8 b, u8 w);
```

Set the color of LED with order number n.

`void show(void);`

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

`void setBrightness(uint8_t);`

Set the brightness of the LED.

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

https://github.com/adafruit/Adafruit_NeoPixel

Project 6.2 Rainbow Light

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

Sketch

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

Sketch_06.2_RainbowLight

Sketch_06.2_RainbowLight | Arduino 1.8.16

File Edit Sketch Tools Help

Sketch_06.2_RainbowLight

```
#include <Adafruit_NeoPixel.h>

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

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

void loop() {
  for (int j = 0; j < 256 * 5; j++) {
    for (int i = 0; i < 8; i++) {
      Wheel(((i * 256 / 8) + j)%255);
      strip.setPixelColor(i, strip.Color(red, green, blue));
    }
    strip.show();
    delay(10);
  }
}
```

Done uploading.

```
Loading into Flash: [============================]  100%
```

9 Raspberry Pi Pico on COM10

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

The following is the program code:

```
#include <Adafruit_NeoPixel.h>

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

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

void loop() {
  for (int j = 0; j < 256 * 5; j++) {
    for (int i = 0; i < 8; i++) {
      Wheel(((i * 256 / 8) + j)%255);
      strip.setPixelColor(i, strip.Color(red, green, blue));
    }
    strip.show();
    delay(10);
  }
}

void Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    red = 255 - WheelPos * 3;
    green = 0;
    blue = WheelPos * 3;
  }
  else if (WheelPos < 170) {
    WheelPos -= 85;
    red = 0;
```

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

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

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

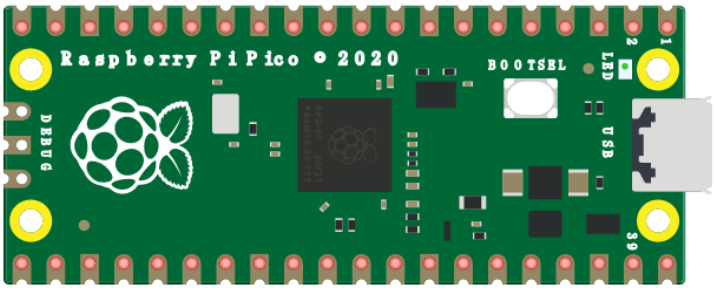
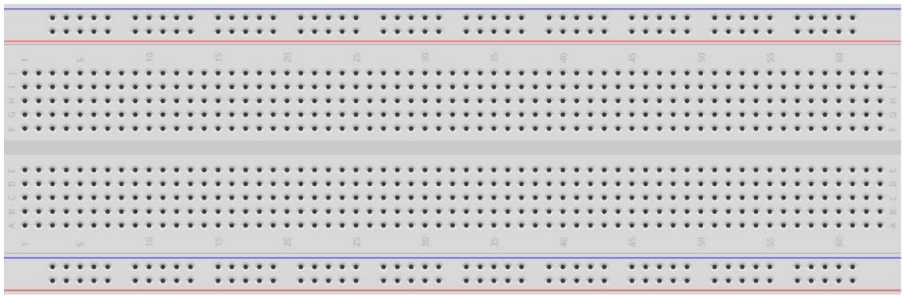
Chapter 7 Buzzer

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

Project 7.1 Doorbell

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

Component List

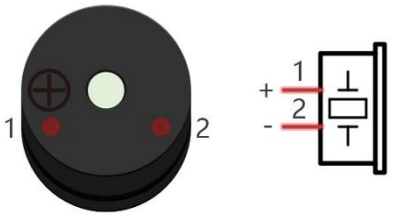
Raspberry Pi Pico x1 		USB cable x1 		
Breadboard x1 				
Jumper 				
NPN transistorx1 (S8050) 	Active buzzer x1 	Push button x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component Knowledge

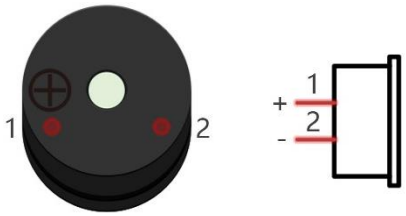
Buzzer

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

Active buzzer



Passive buzzer



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

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

How to identify active and passive buzzer?

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

Active buzzer

Passive buzzer

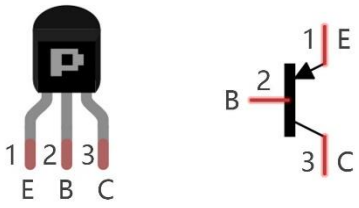
Transistor

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

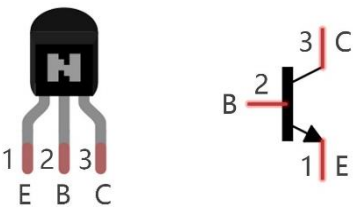
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

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

PNP transistor



NPN transistor

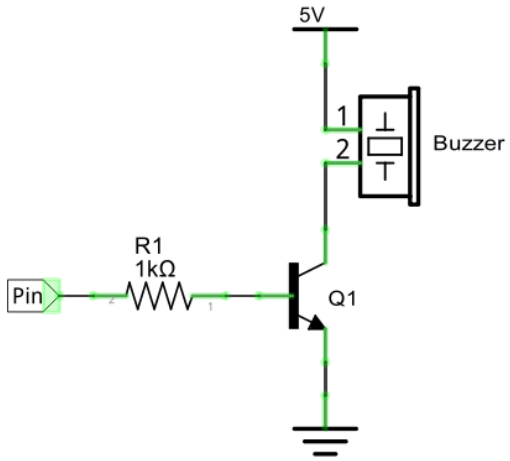
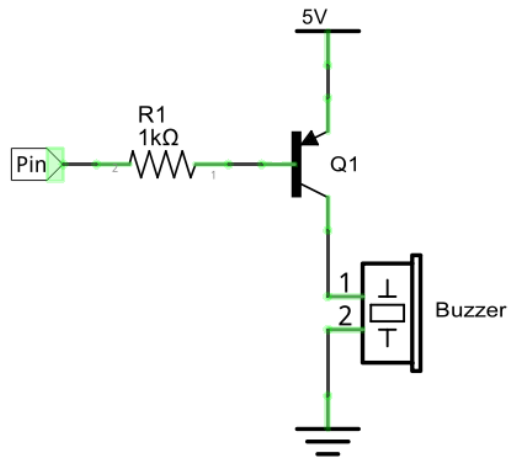


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

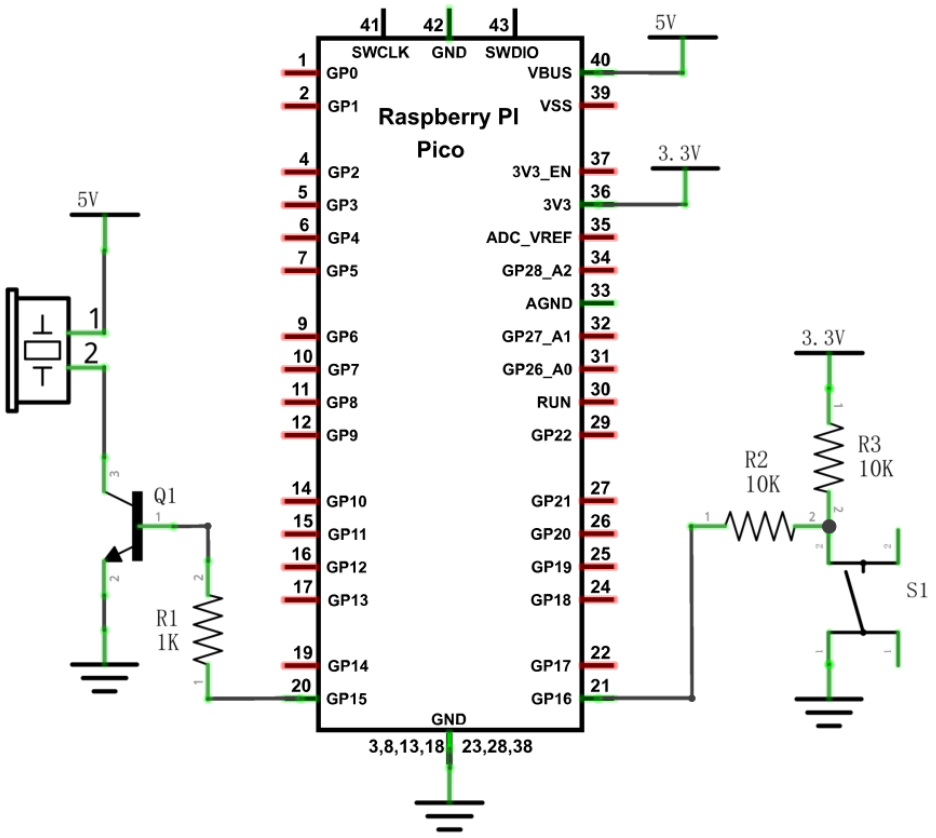
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

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

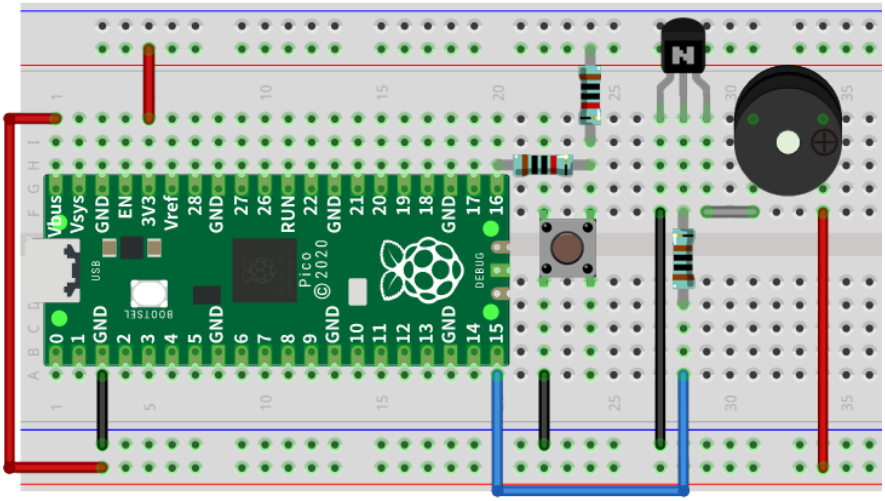
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

Circuit

Schematic diagram



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



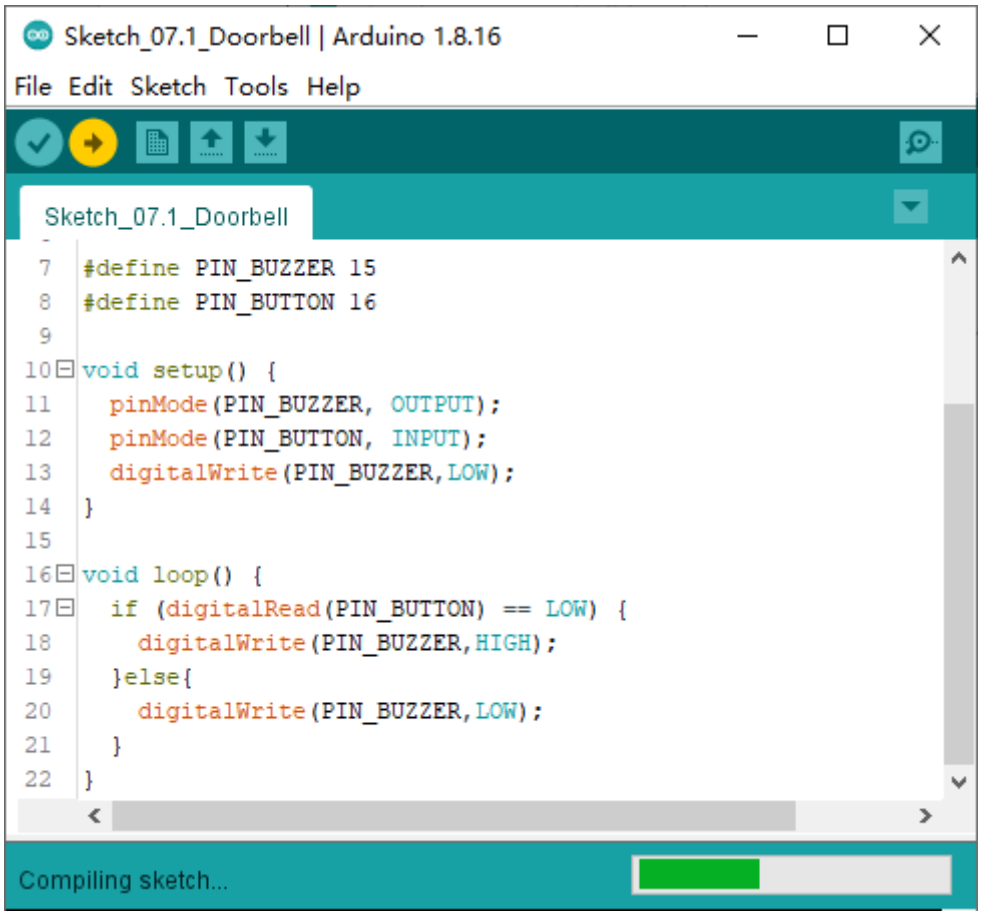
Note:

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

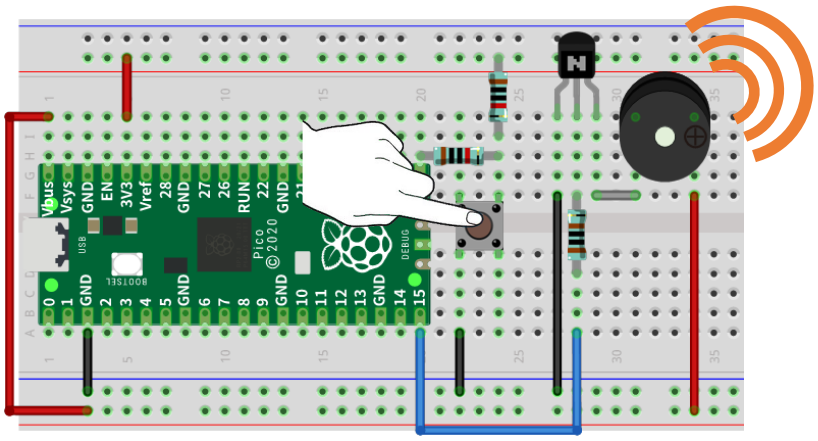
Sketch

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

Sketch_07.1_Doorbell



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



The following is the program code:

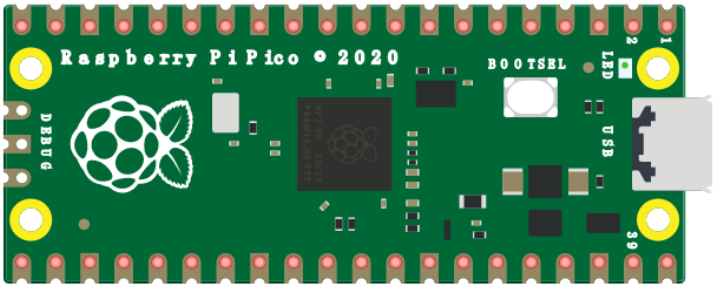
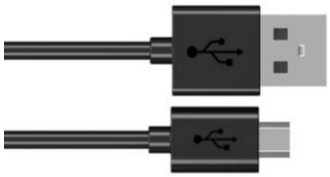
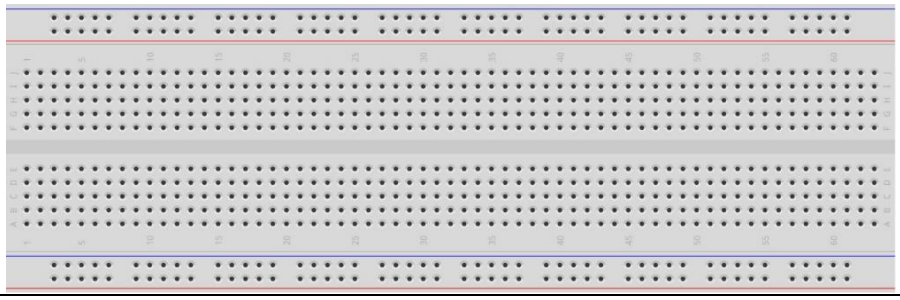
```
#define PIN_BUZZER 15
#define PIN_BUTTON 16

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

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

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

Project 7.2 Alertor

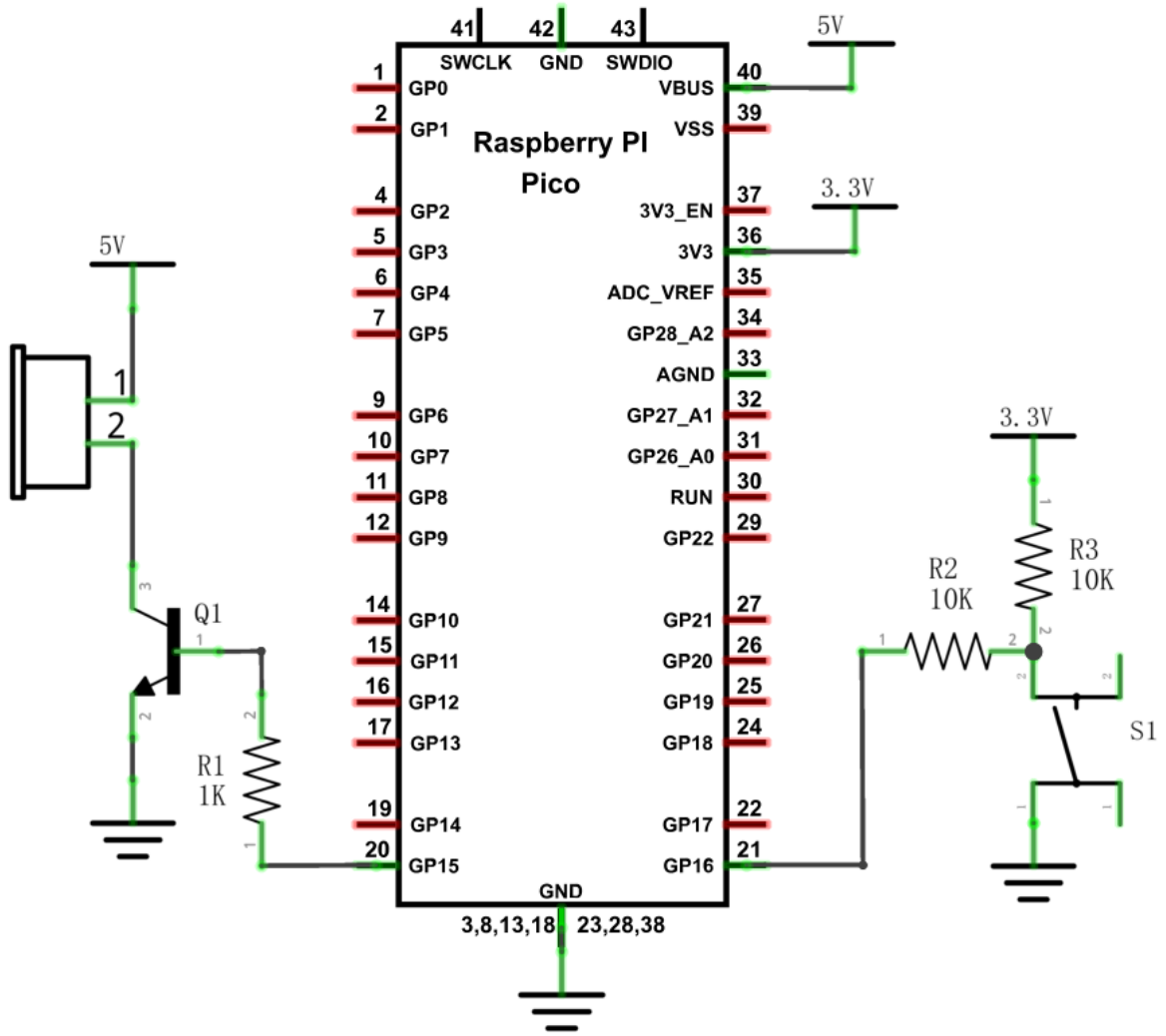
Next, we will use a passive buzzer to make an alarm.
Component list and the circuit part is similar to last section, only the **active buzzer** needs to be **replaced** with a **passive buzzer** for this project.

Component List

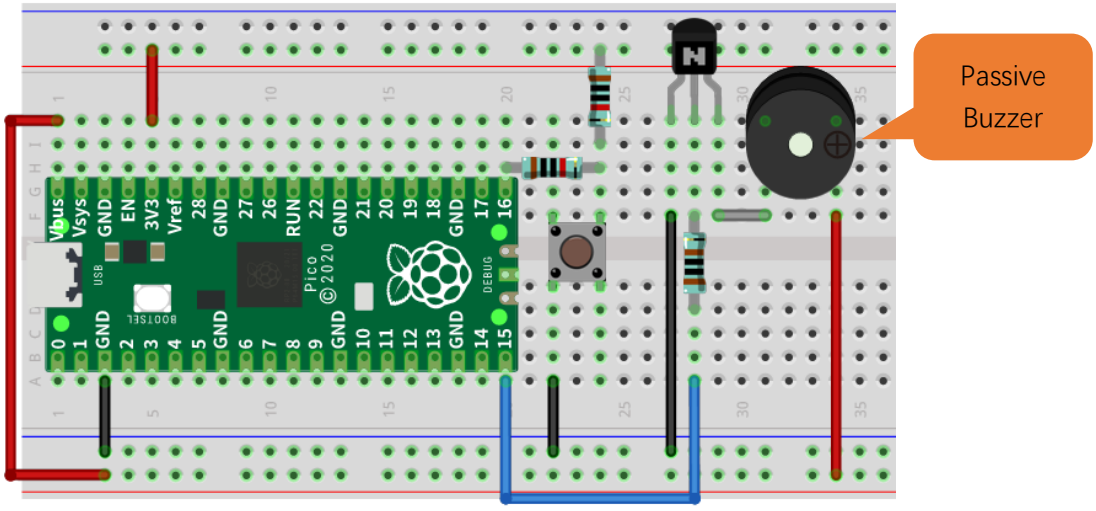
<table>
<tr><td colspan="3">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td colspan="5">Jumper
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Passive buzzer x1
  </td><td>Push button x1
  </td><td>Resistor
1kΩ x1
  </td><td>Resistor
10kΩ x2
  </td></tr>
</table>

Circuit

Schematic diagram



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

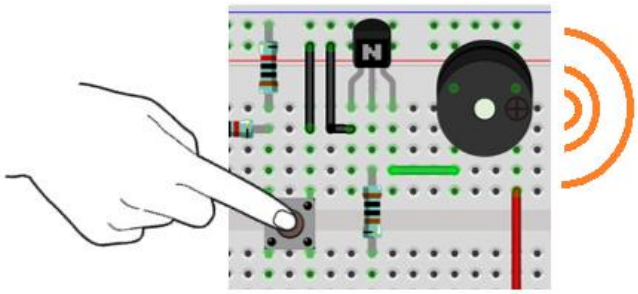


Sketch

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

Sketch_07.2_Alertor

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



The following is the program code:

```
#define PIN_BUZZER 15
#define PIN_BUTTON 16

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

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

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

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

```
        delayMicroseconds(1000000 / freqs / 2);
        digitalWrite(PIN, LOW);
        delayMicroseconds(1000000 / freqs / 2);
      }
    }
}
```

Define the button and pin to control the passive buzzer.

```
#define PIN_BUZZER 15
#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.

```
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);
      delayMicroseconds(1000000 / freqs / 2);
      digitalWrite(PIN, LOW);
      delayMicroseconds(1000000 / freqs / 2);
    }
  }
}
```

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.

```
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);
  }
}
```

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.

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

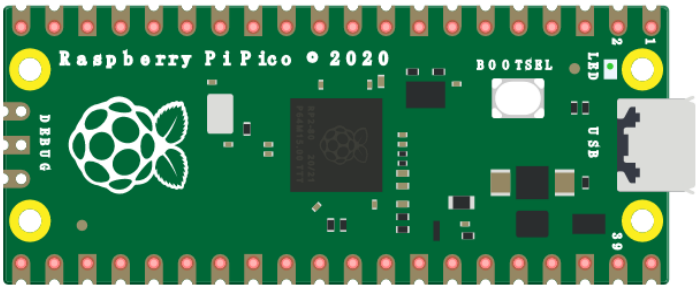
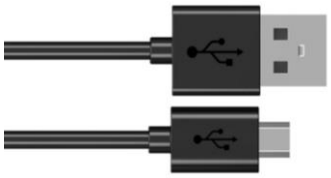
Chapter 8 Serial Communication

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

Project 8.1 Serial Print

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

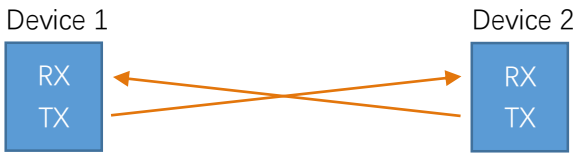
Component List

Raspberry Pi Pico x1 	USB cable x1 

Related Knowledge

Serial communication

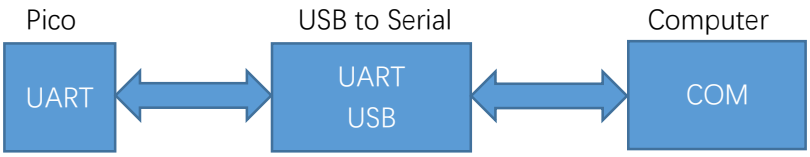
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



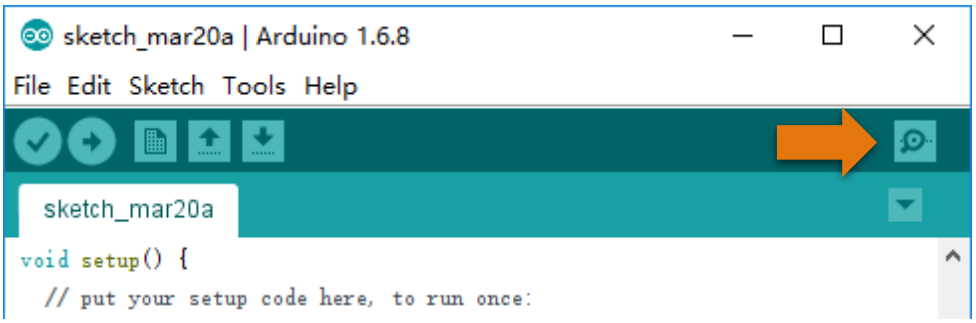
Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

Serial port on Raspberry Pi Pico

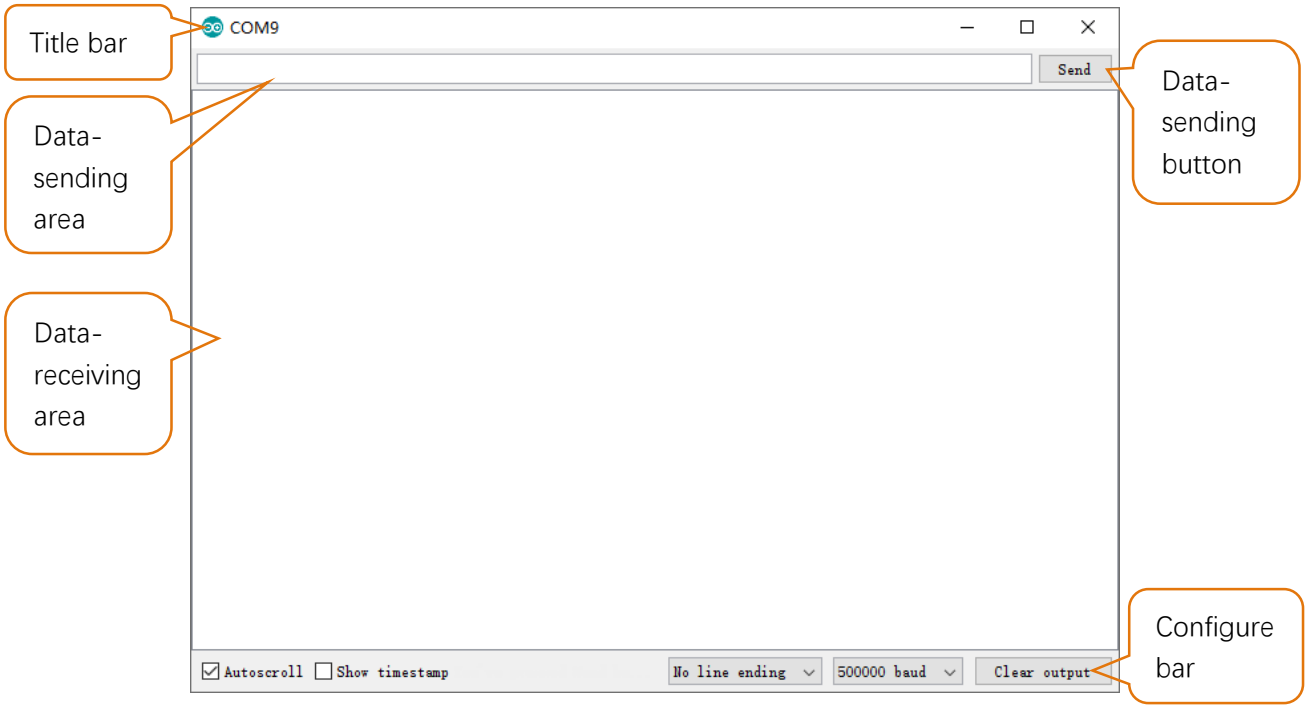
Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



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

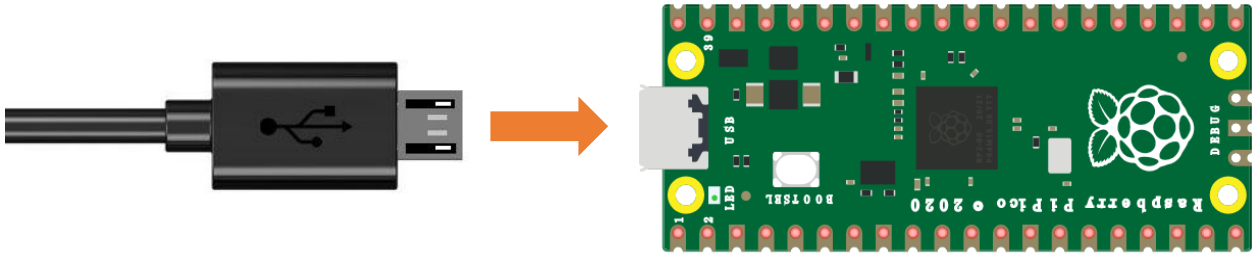


Interface of serial monitor window is as follows. If you can't open it, make sure Pico has been connected to the computer, and choose the right serial port in the menu bar "Tools".



Circuit

Connect Raspberry Pi Pico to the computer with USB cable.



Sketch

Sketch_08.1_SerialPrinter

Sketch_08.1_SerialPrinter | Arduino 1.8.16

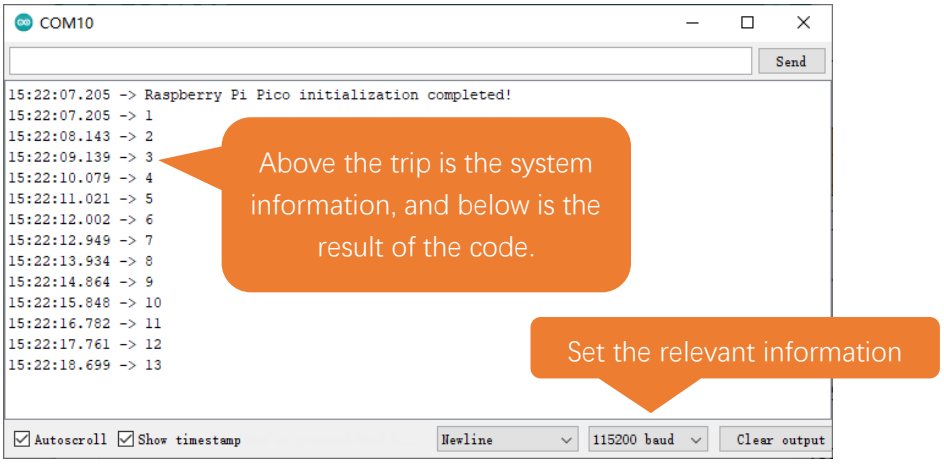
File Edit Sketch Tools Help

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:



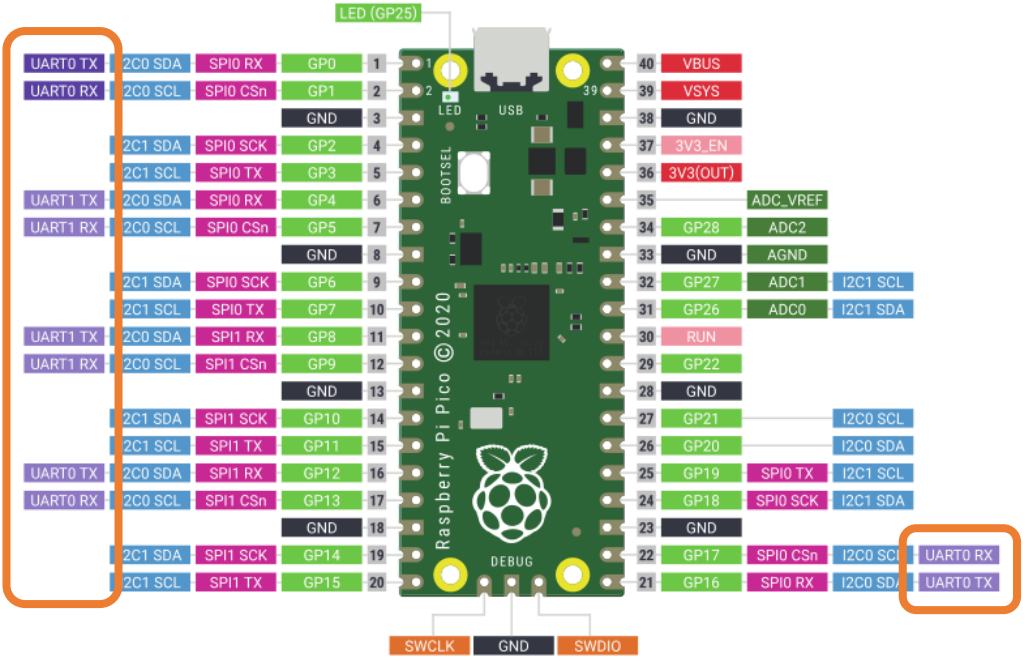
As shown above, when the code runs, the data is printed every one second.

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 8.2 Serial Read and Write

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

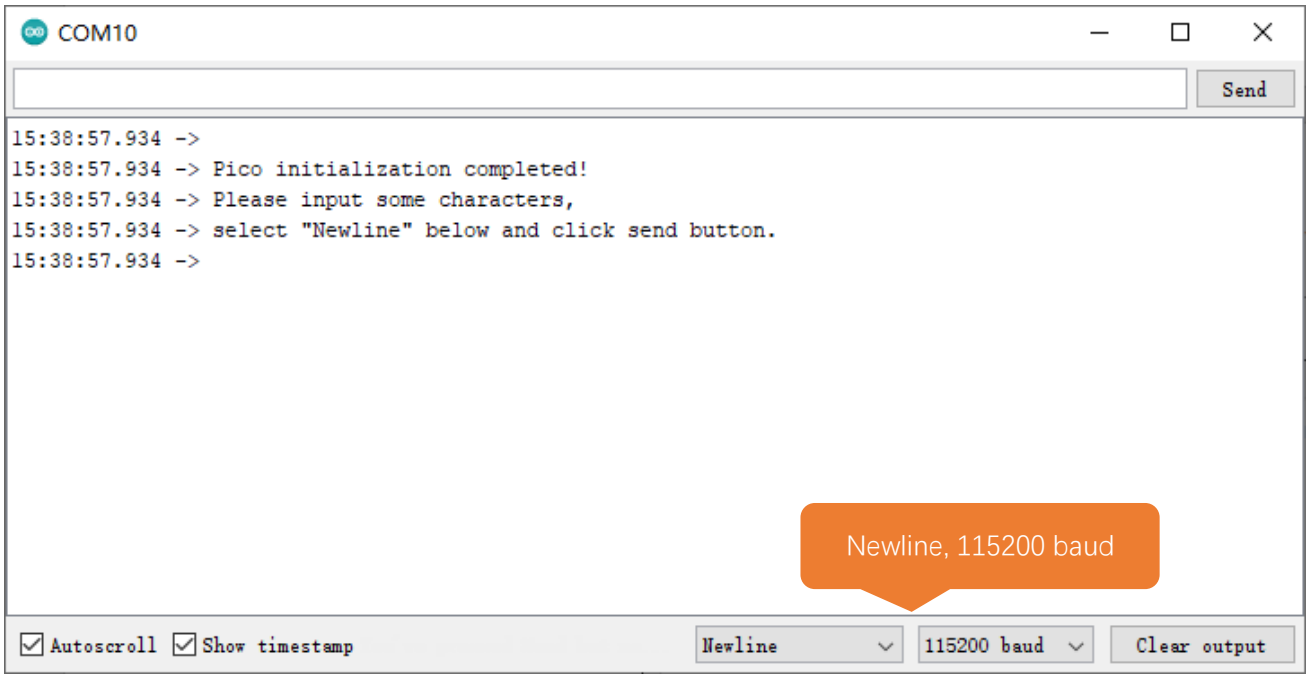
Component and circuit are the same as in the previous project.

Sketch

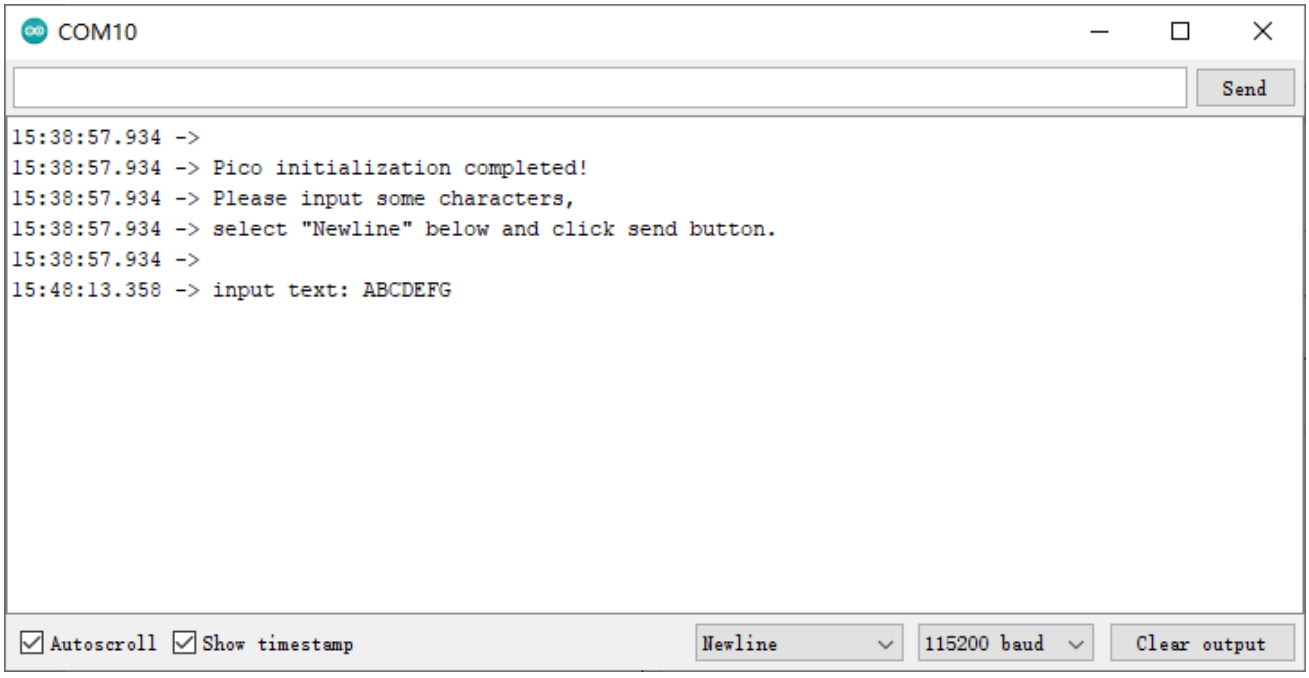
Sketch_08.2_SerialRW


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

void setup() {
  Serial.begin(115200);
  Serial.println(String("\nPico initialization completed!\n")
                 + String("Please input some characters,\n")
                 + String("select \"Newline\" below and click send button. \n"));
}

void loop() {
  if (Serial.available()) {         // judge whether data has been received
    char inChar = Serial.read();         // read one character
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
  if (stringComplete) {
    Serial.print("input text: ");
    Serial.print(inputString);
    inputString = "";
    stringComplete = false;
  }
}
```

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



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



The following is the program code:

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

void setup() {
  Serial.begin(115200);delay(1000);
  Serial.println(String("\nPico initialization completed!\n")
                + String("Please input some characters,\n")
                + String("select \"Newline\" below and click send button. \n"));
}

void loop() {
  if (Serial.available()) {         // judge whether data has been received
    char inChar = Serial.read();          // read one character
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
  if (stringComplete) {
    Serial.print("input text: ");
    Serial.print(inputString);
    inputString = "";
    stringComplete = false;
  }
}
```

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

Reference

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

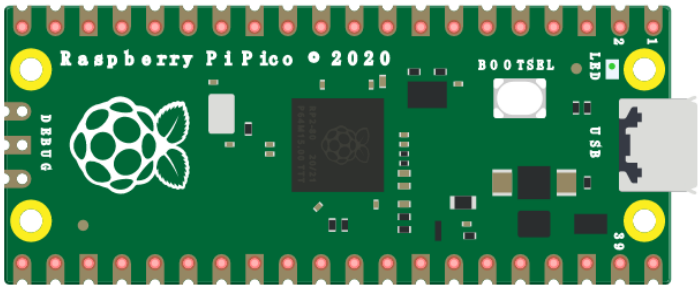
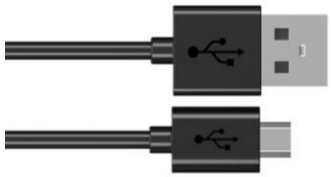
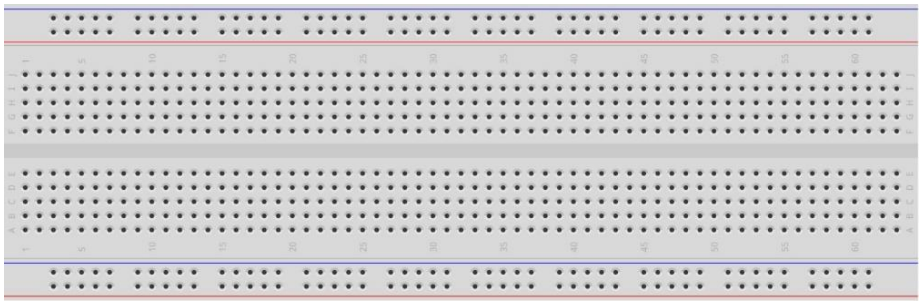
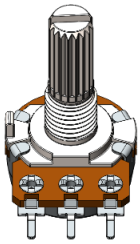
Chapter 9 AD Converter

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

Project 9.1 Read the Voltage of Potentiometer

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

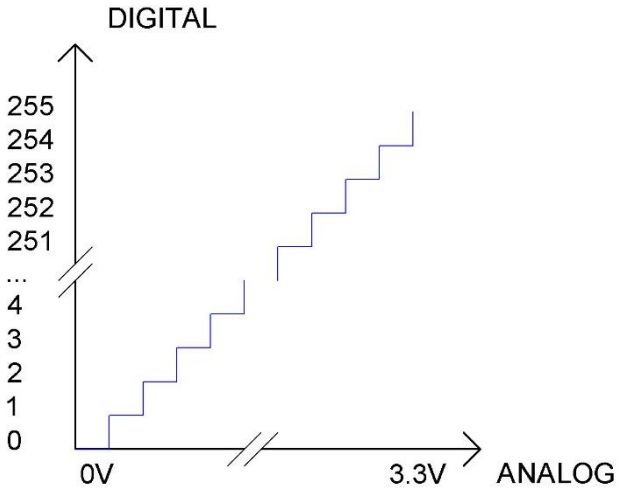
Component List

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

Related Knowledge

ADC

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



Subsection 1: the analog in rang of 0V---3.3/1023 V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3/1023V---2*3.3/1023V corresponds to digital 1;
...
The following analog will be divided accordingly.
The conversion formula is as follows:

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

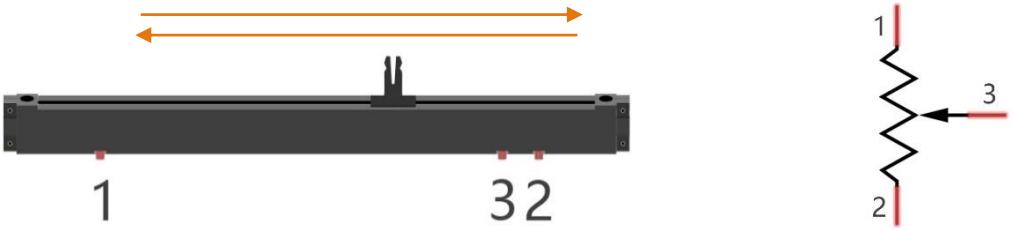
ADC ChannelsRaspberry Pi Pico

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

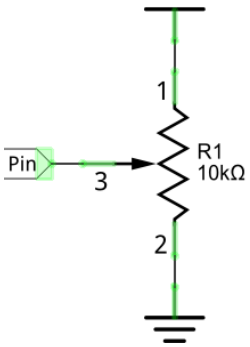
Potentiometer

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.



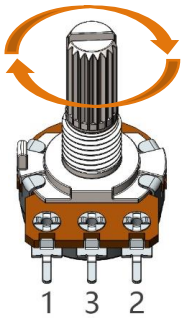
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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



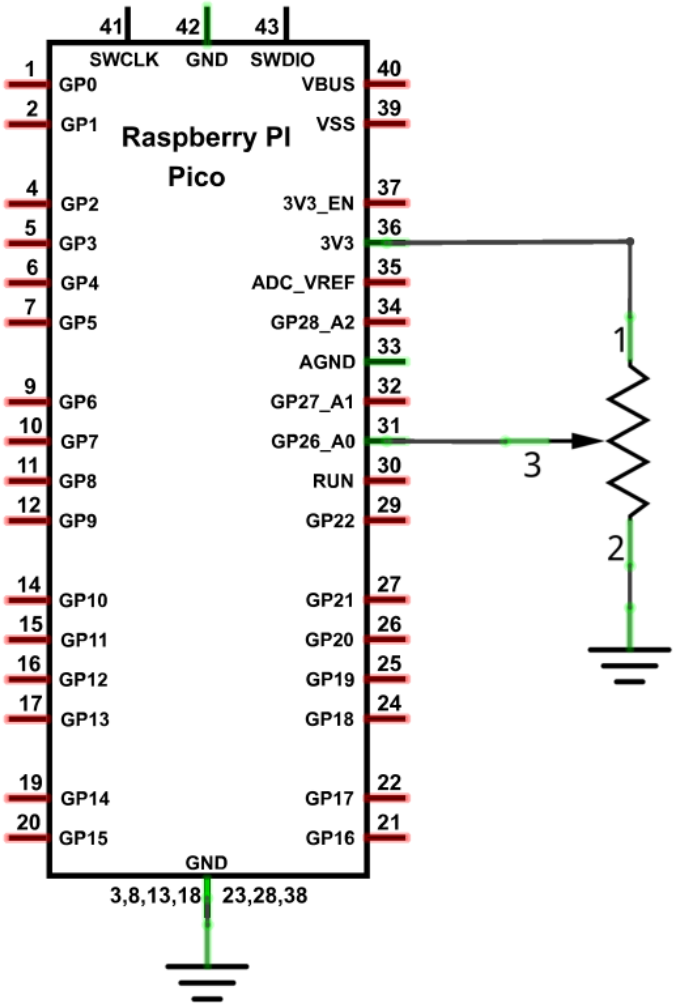
Rotary potentiometer

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

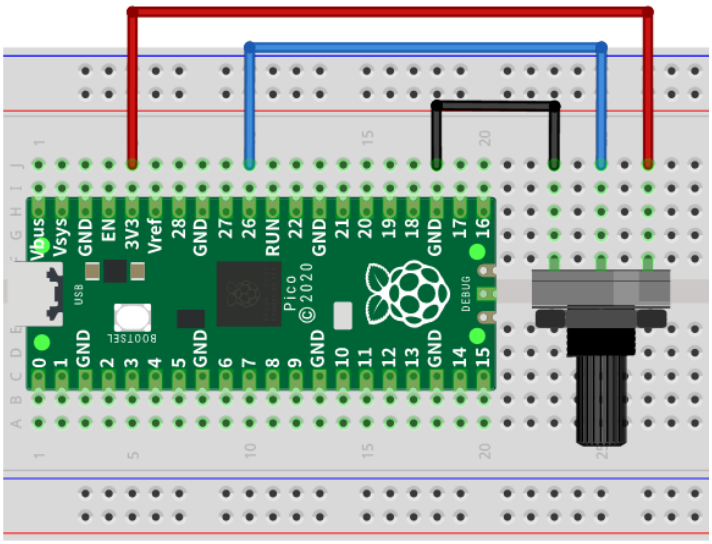


Circuit

Schematic diagram



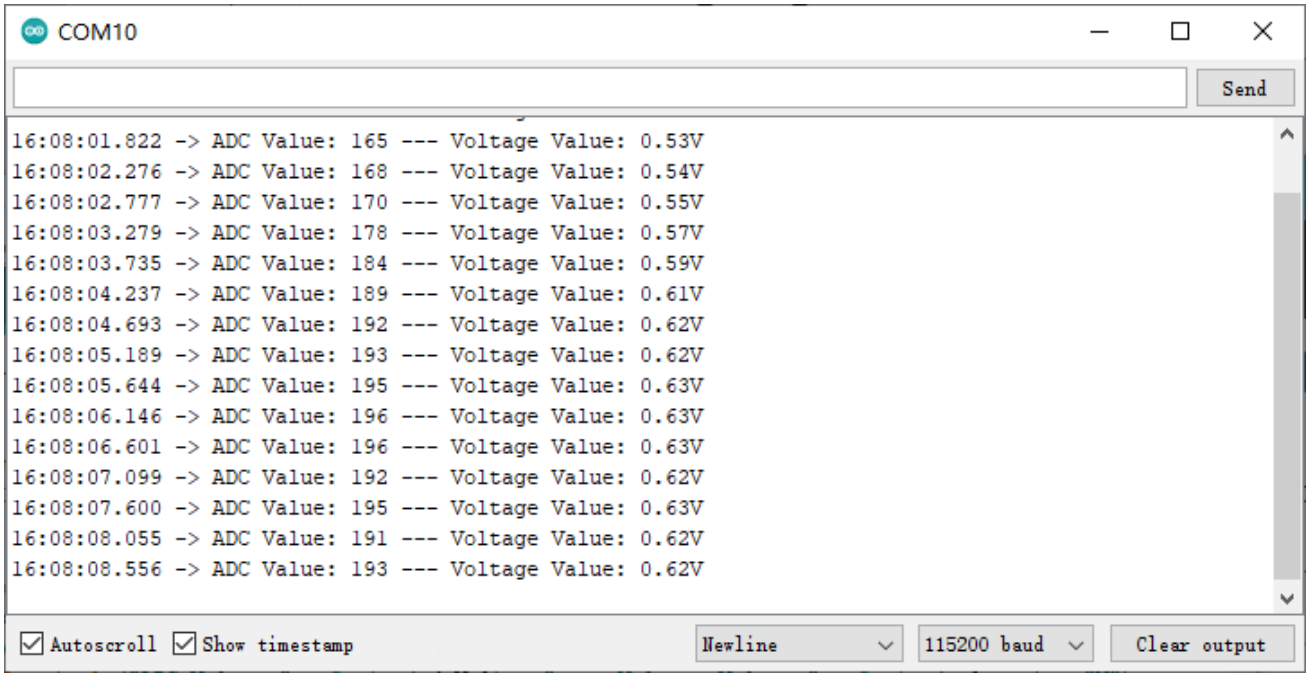
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_09.1_ADC

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



The following is the code:

```
#define PIN_ANALOG_IN  26

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

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

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

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

Reference

uint16_t analogRead(*uint8_t* pin);

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

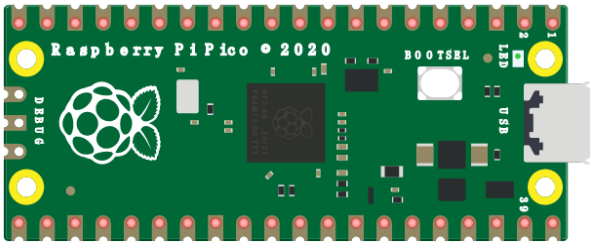
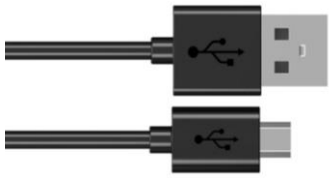
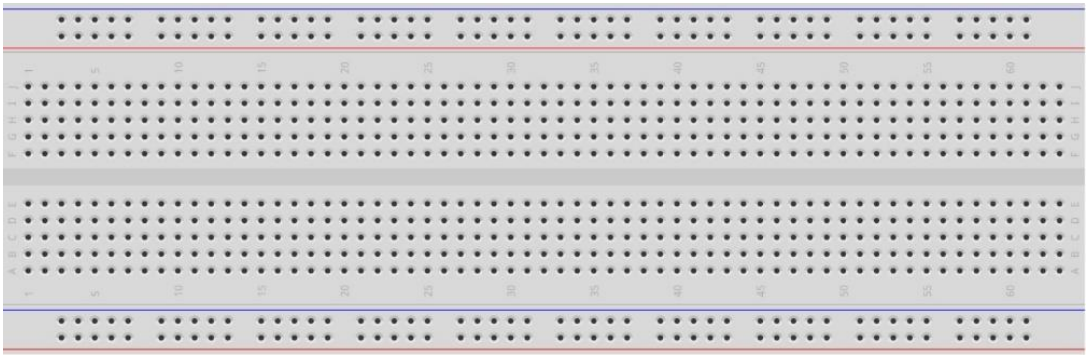
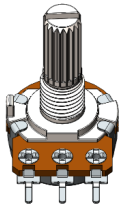
Chapter 10 Potentiometer & LED

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

Project 10.1 Soft Light

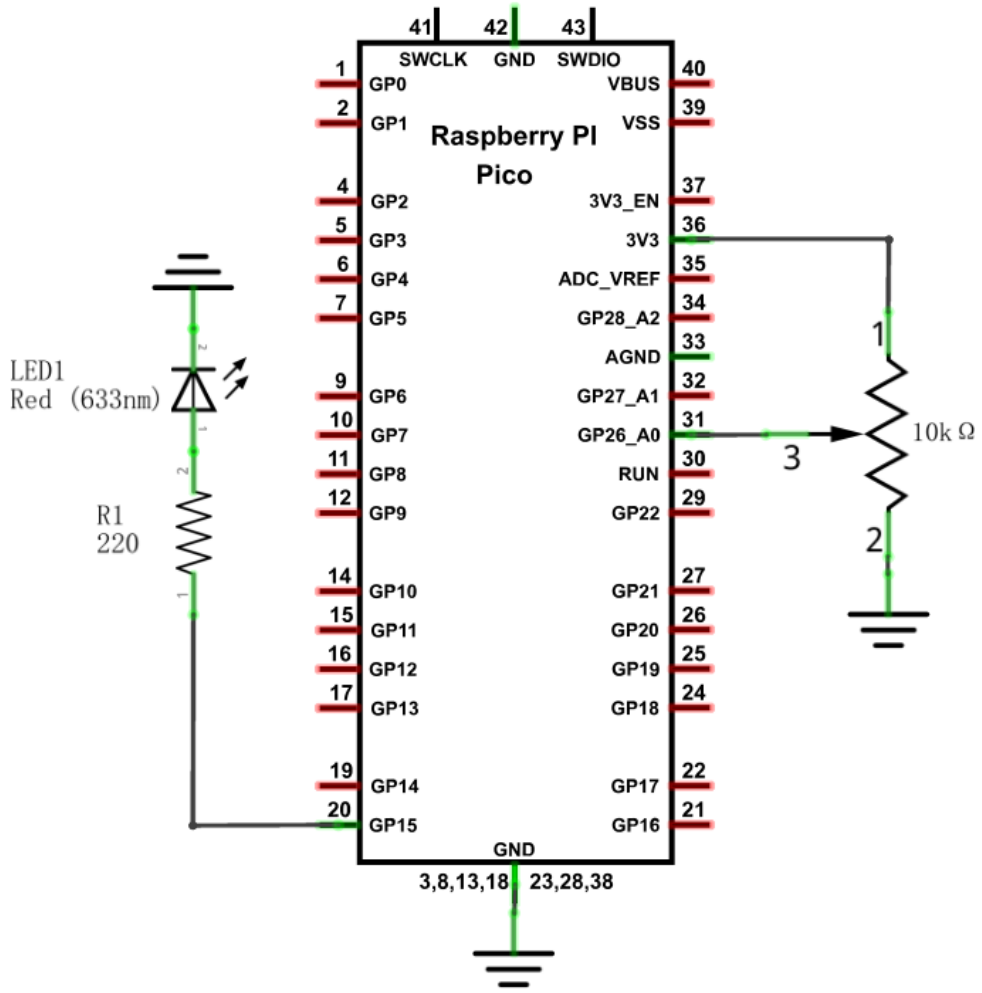
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

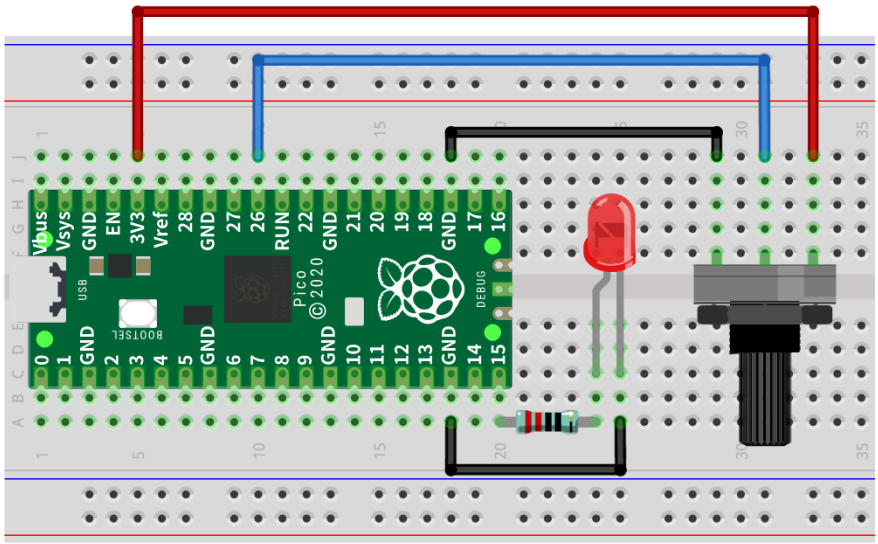
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td><td>Jumper
</td></tr>
</table>

Circuit

Schematic diagram



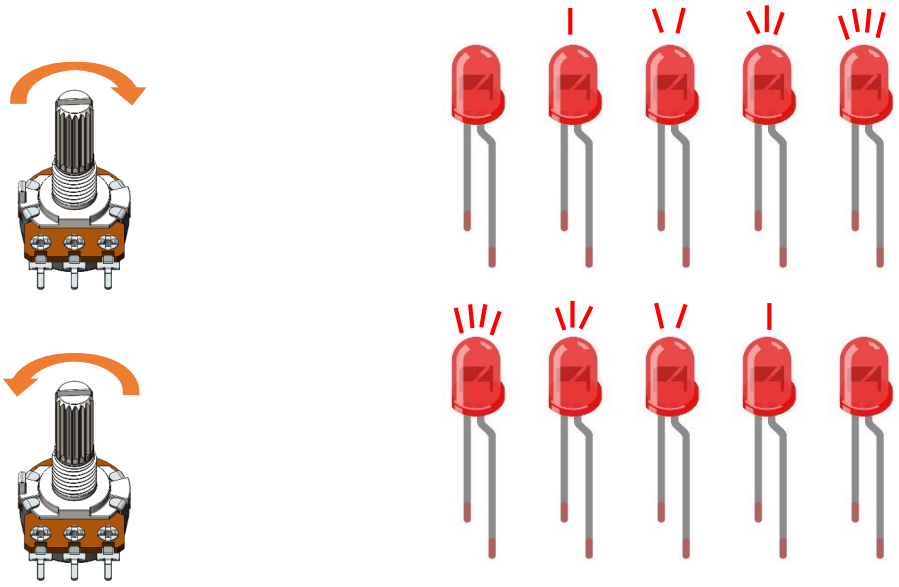
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_10.1_Softlight

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



The following is the code:

```
#define PIN_ADC0        26
#define PIN_LED         15

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

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

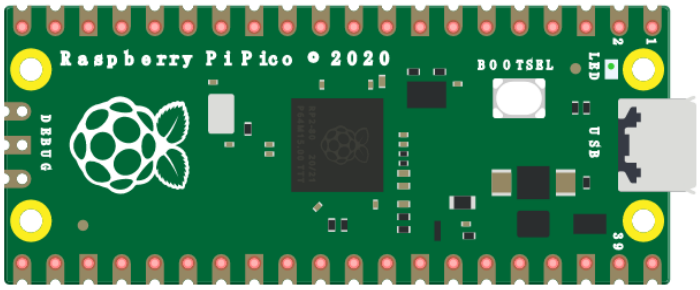
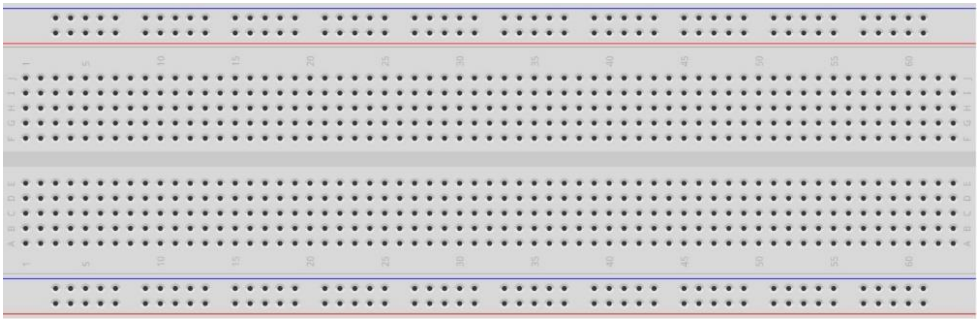
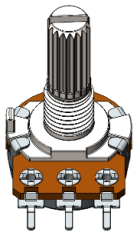
In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED brightness.

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

Project 10.2 Soft Colorful Light

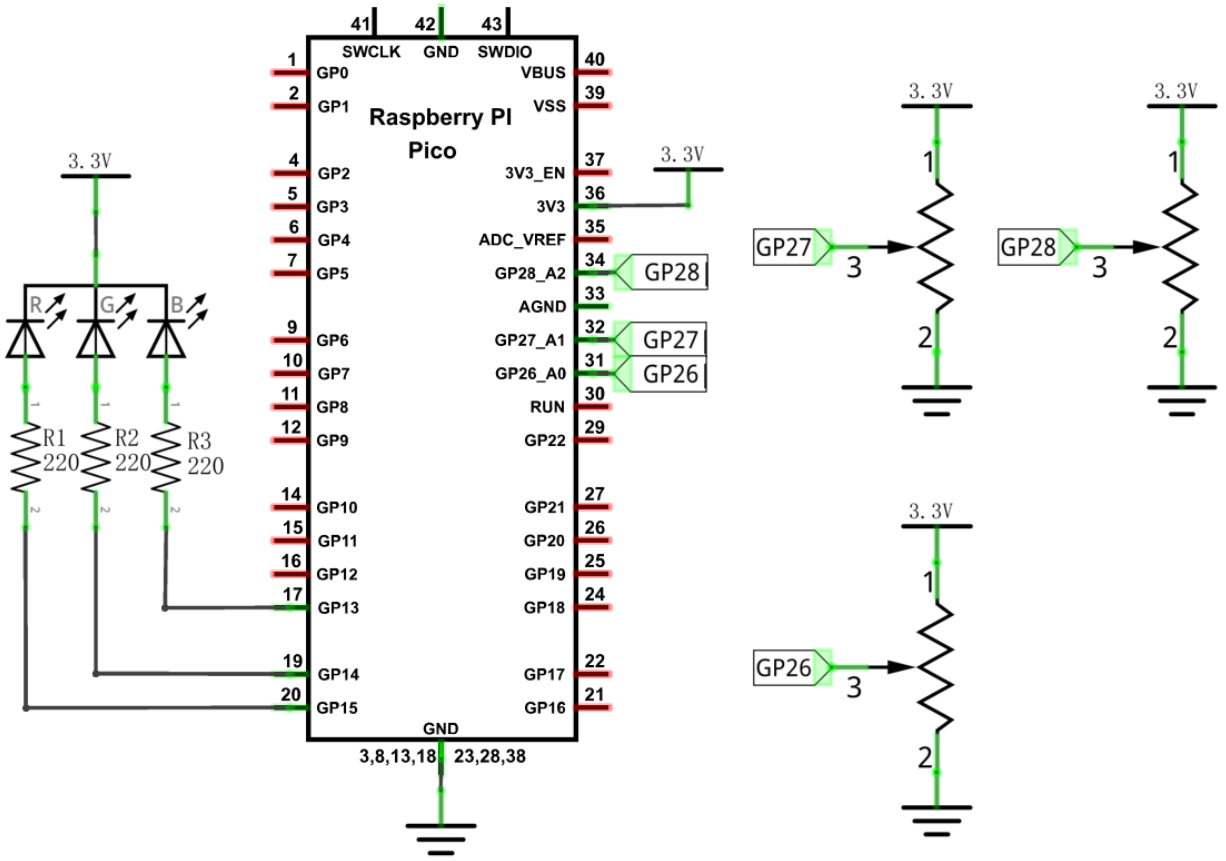
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED,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

Component List

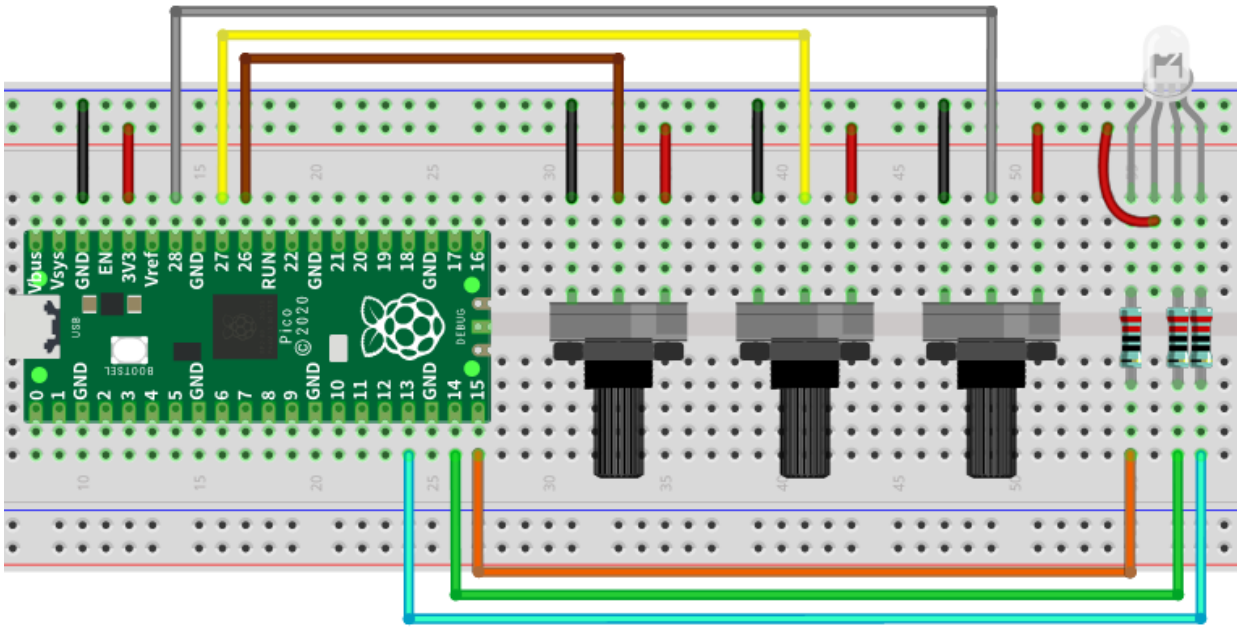
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td>Resistor 220Ω x3
  </td><td>RGBLED x1
  </td><td>Jumper
  </td></tr>
</table>

Circuit

Schematic diagram

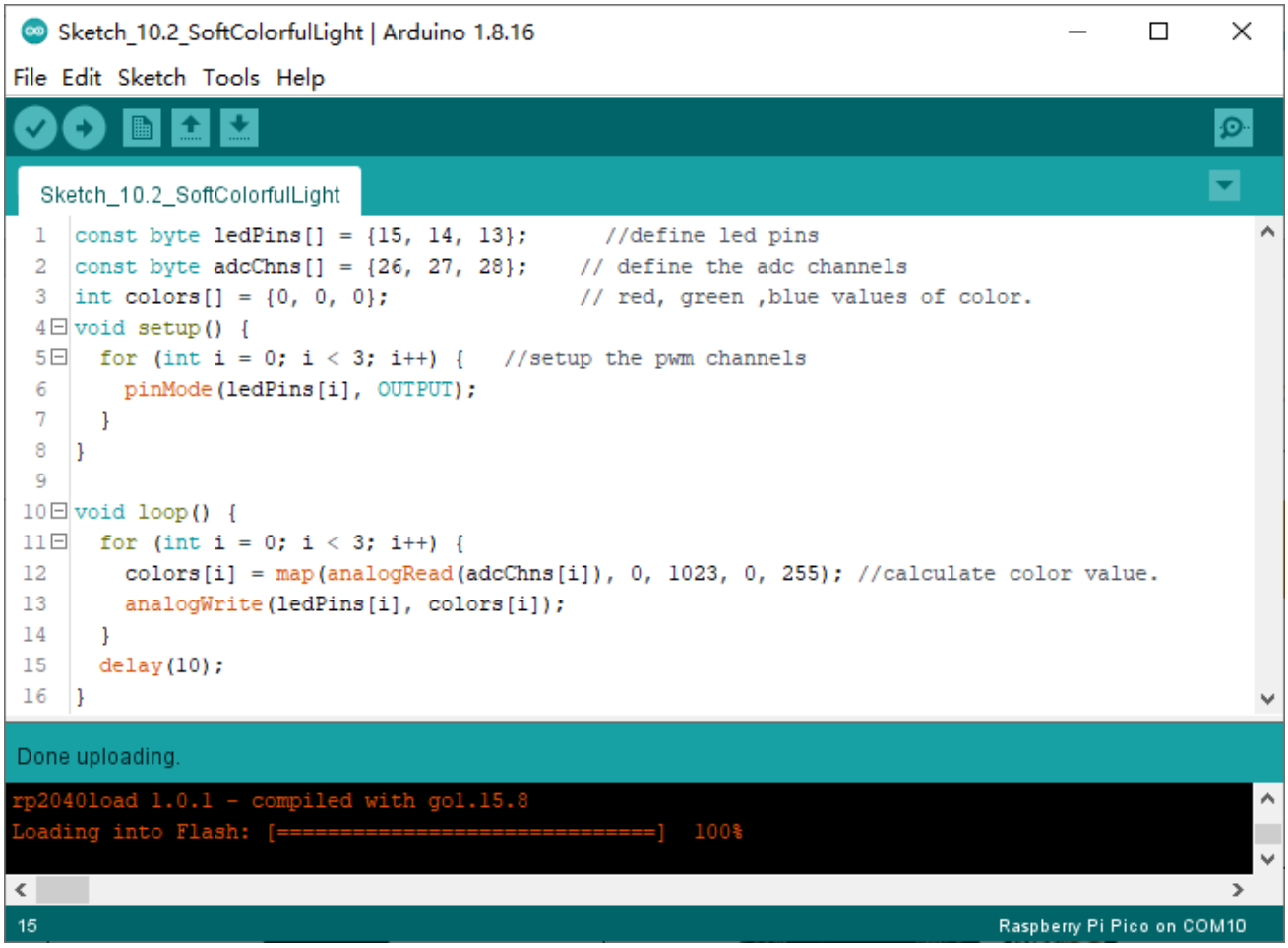


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_10.2_SoftColorfulLight



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

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

The following is the program code:

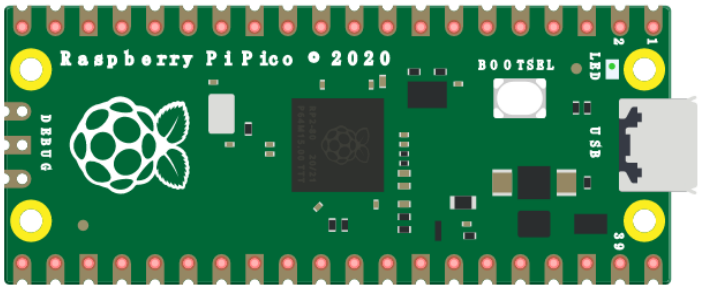
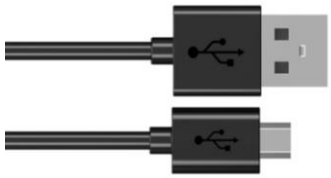
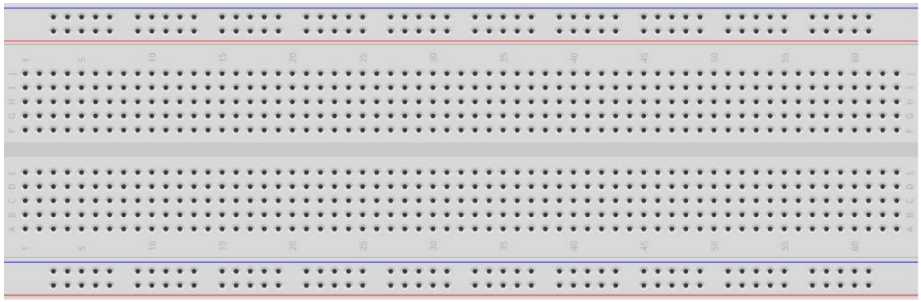
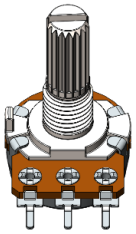
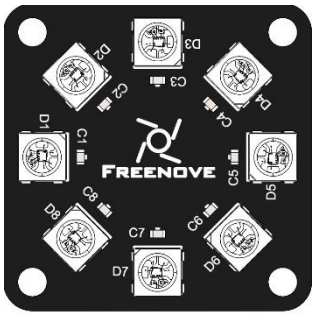
```
const byte ledPins[] = {15, 14, 13};      //define led pins
const byte adcChns[] = {26, 27, 28};    // define the adc channels
int colors[] = {0, 0, 0};               // red, green ,blue values of color.
void setup() {
  for (int i = 0; i < 3; i++) {   //setup the pwm channels
    pinMode(ledPins[i],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

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

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

Project 10.3 Soft Rainbow Light

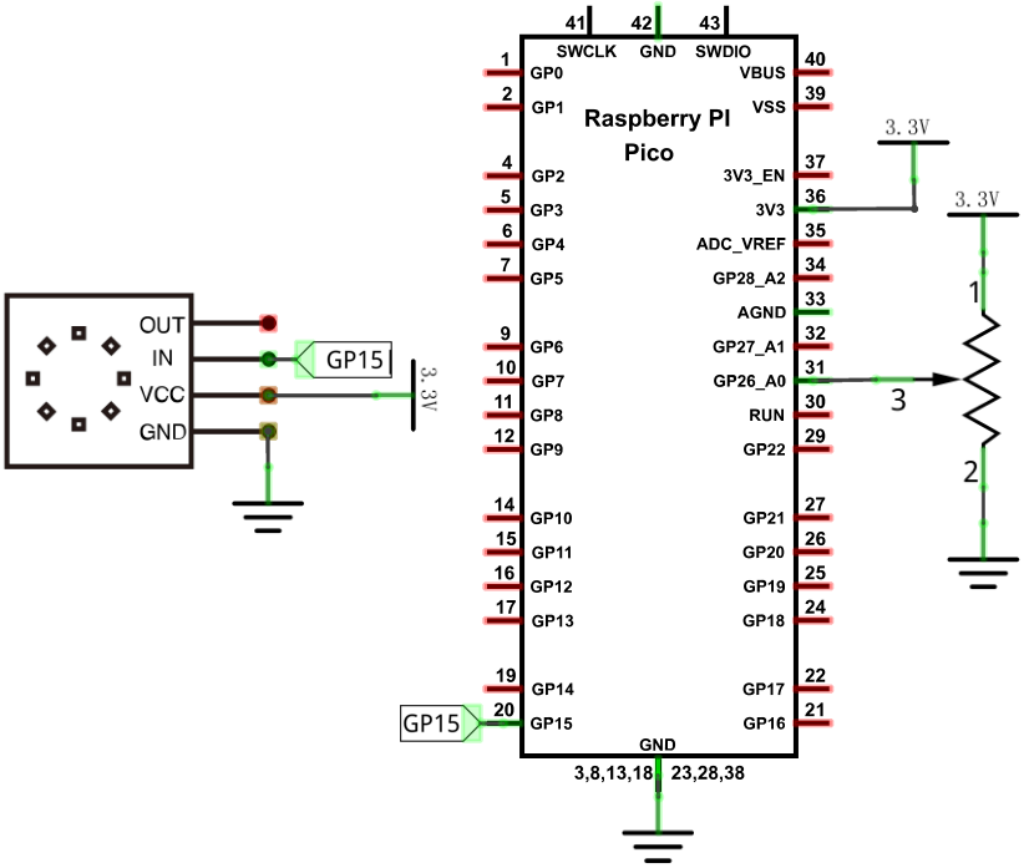
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

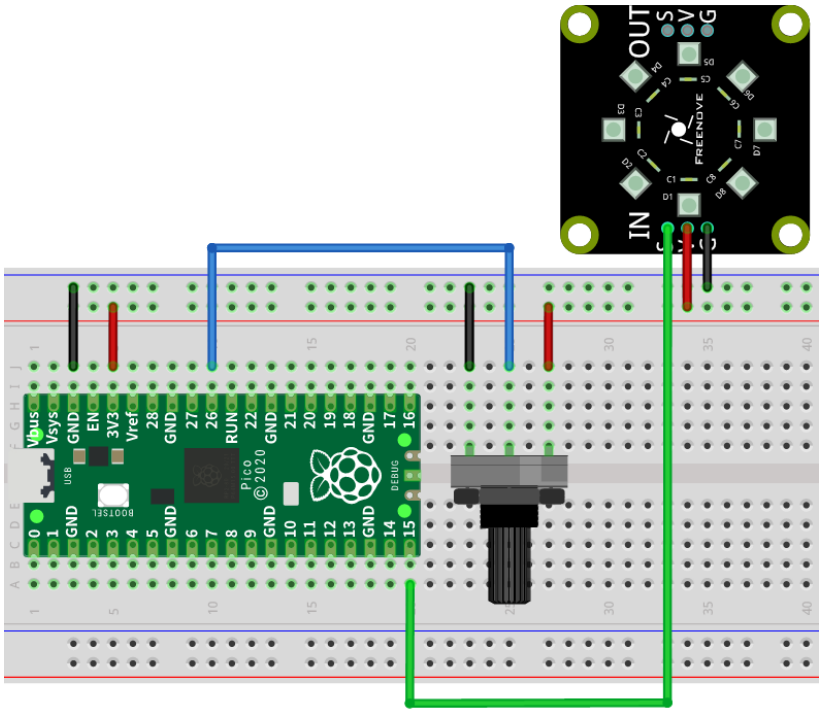
<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Freenove 8 RGB LED Module x1
  </td><td>Jumper
Jumper
  </td></tr>
</table>

Circuit

Schematic diagram



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



Sketch

Sketch_10.3_Soft_Rainbow_Light

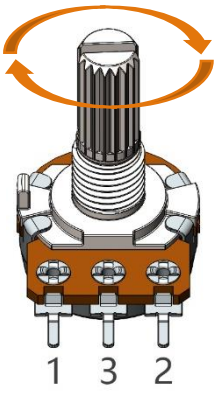
Sketch_10.3_SoftRainbowLight | Arduino 1.8.16

File Edit Sketch Tools Help

Sketch_10.3_SoftRainbowLight

```
#include <Adafruit_NeoPixel.h>

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

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

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



The following is the program code:

```
#include <Adafruit_NeoPixel.h>

#define Pin       16
#define NumPixels 8
#define Pin_ADC0  26
int red = 0;
```

```
int green = 0;
int blue = 0;
int adcVal = 0;
Adafruit_NeoPixel strip(NumPixels, Pin, NEO_GRB + NEO_KHZ800);

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

void Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    red = 255 - WheelPos * 3;
    green = 0;
    blue = WheelPos * 3;
  }else if(WheelPos < 170) {
    WheelPos -= 85;
    red = 0;
    green = WheelPos * 3;
    blue = 255 - WheelPos * 3;
  }else{
    WheelPos -= 170;
    red = WheelPos * 3;
    green = 255 - WheelPos * 3;
    blue = 0;
  }
}
```

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

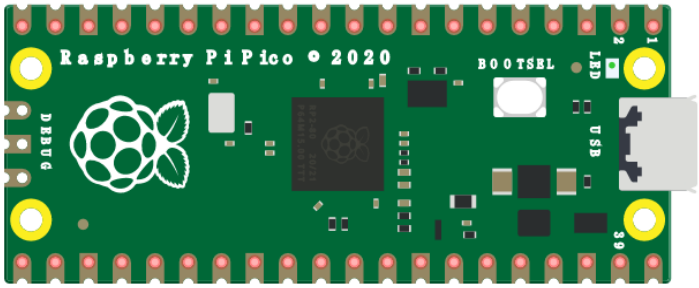
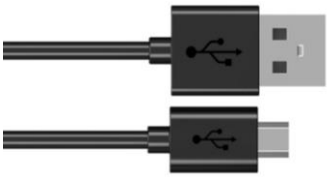
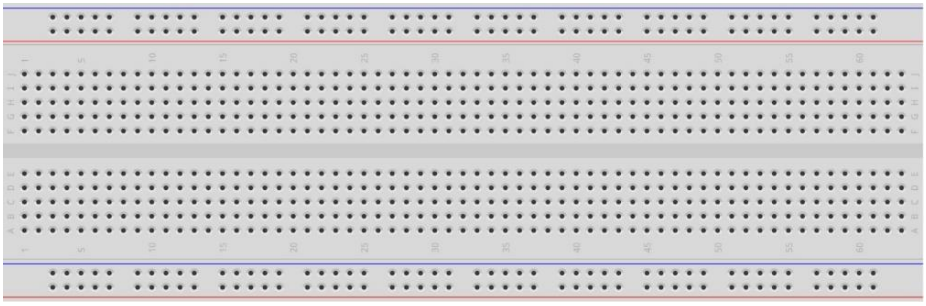
Chapter 11 Photoresistor & LED

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

Project 11.1 Control LED through Photoresistor

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

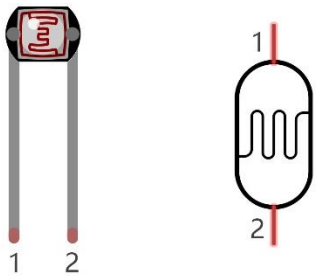
Component List

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

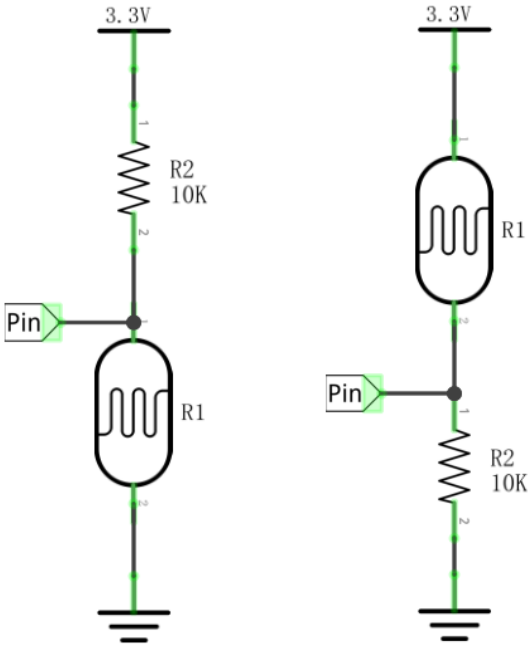
Component Knowledge

Photoresistor

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



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

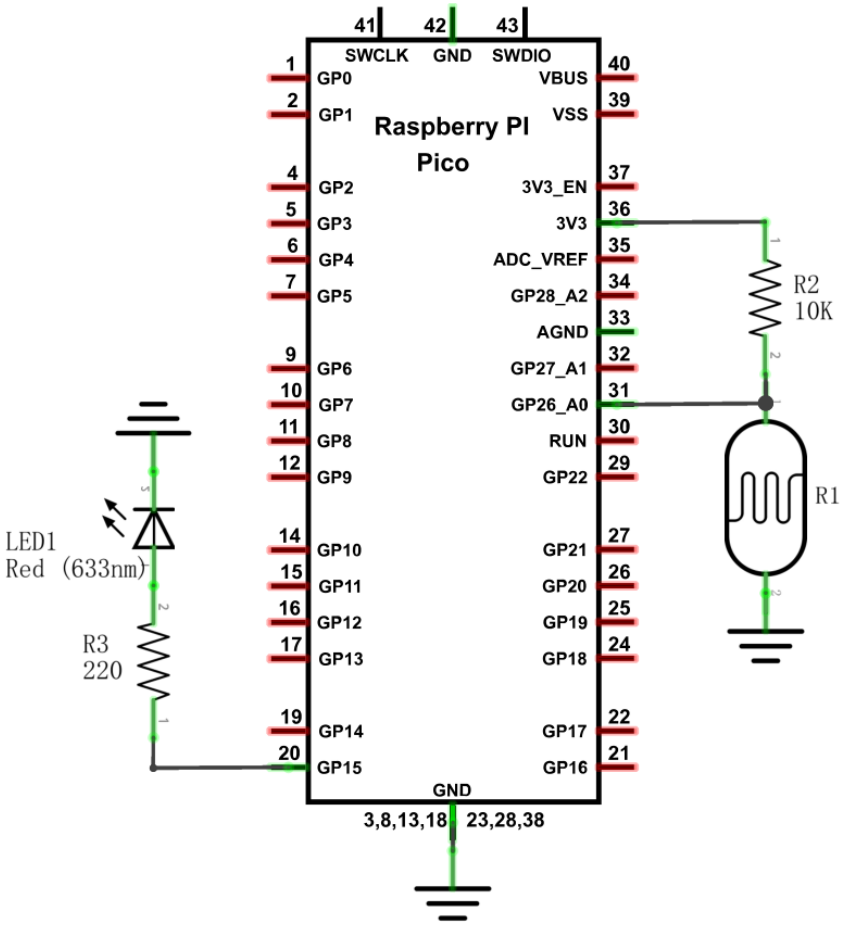


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

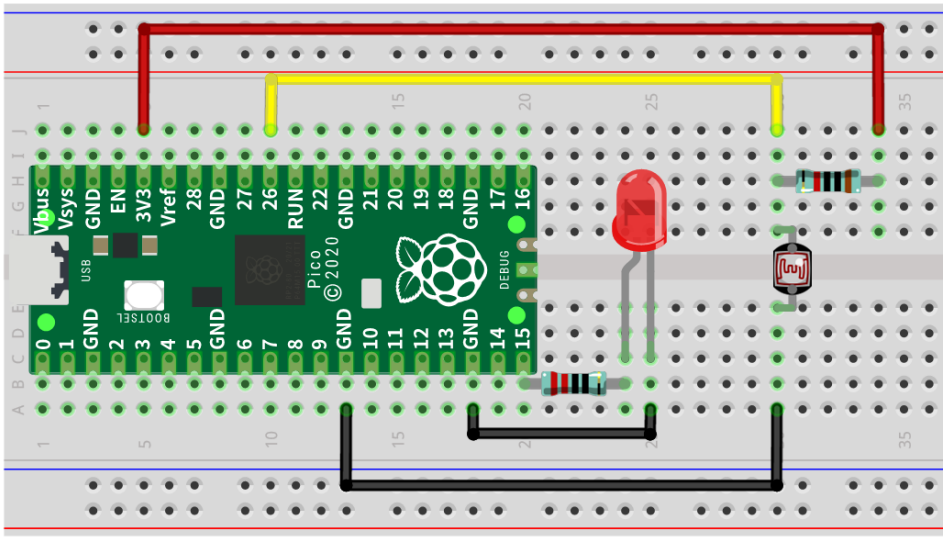
Circuit

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

Schematic diagram



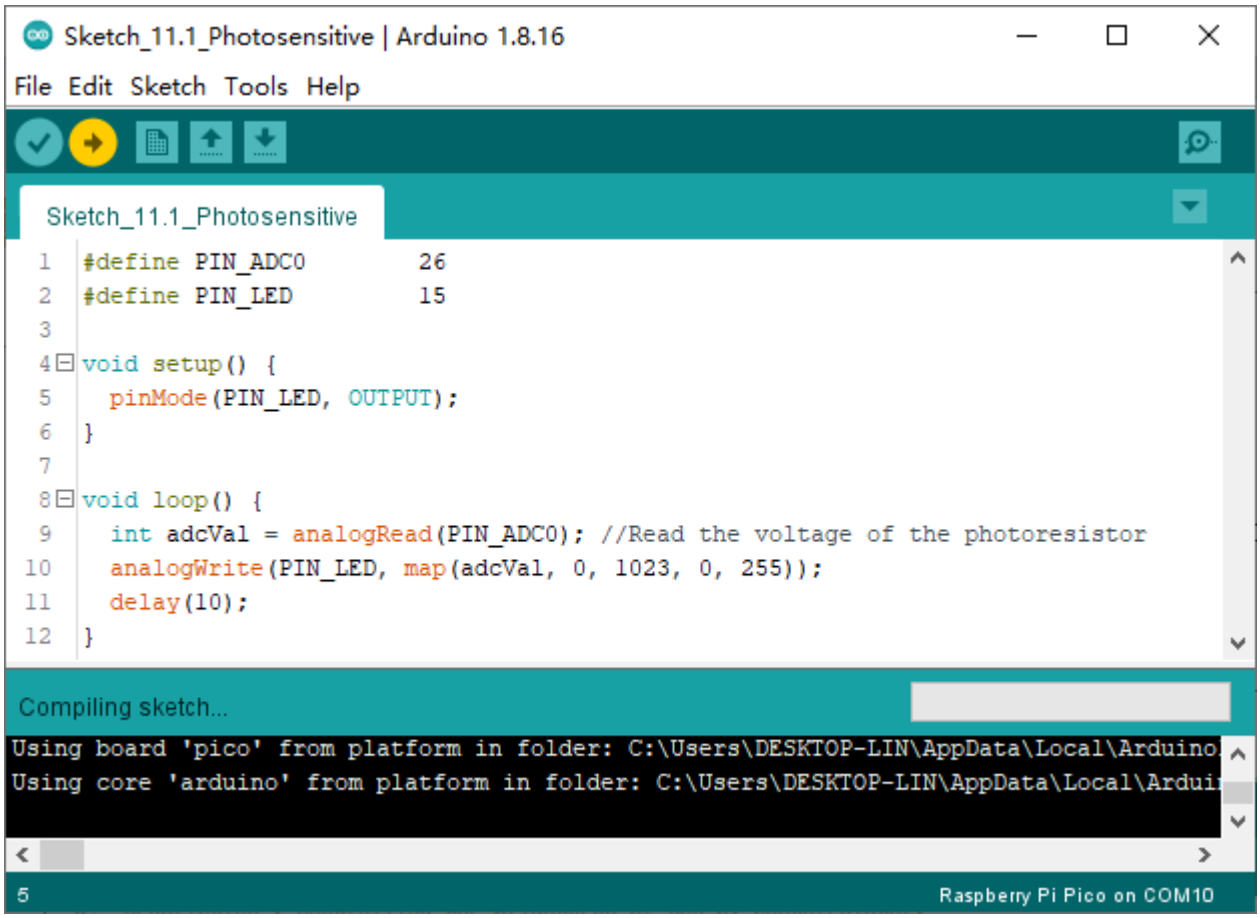
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

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

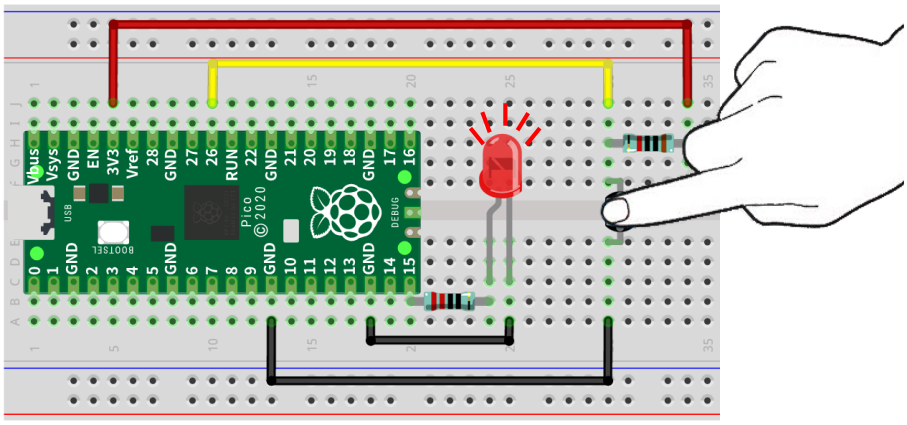
Sketch_11.1_Nightlamp



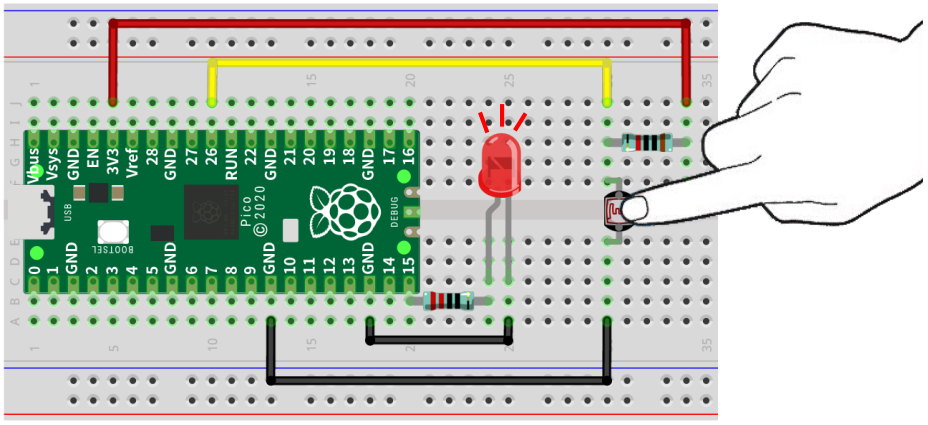
Download the code to Pico, if you cover the photoresistor or increase the light shining on it, the brightness of the LED changes accordingly.

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

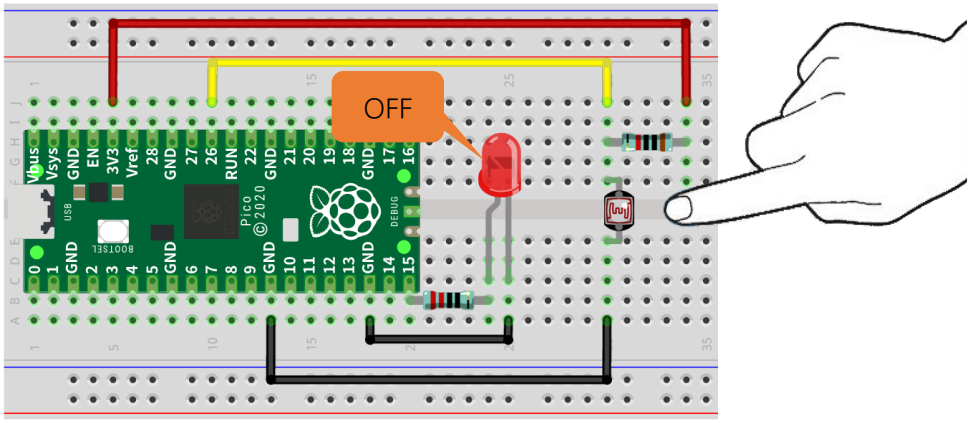
Fully cover the photoresistor:



Half cover the photoresistor:



Not cover the photoresistor:



The following is the program code:

```
#define PIN_ADC0          26
#define PIN_LED           15

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

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

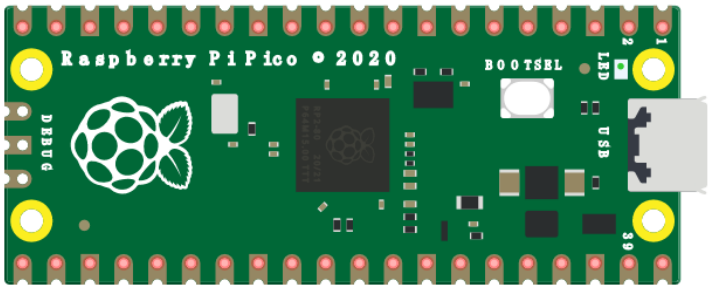
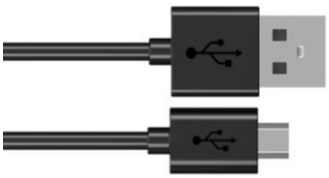
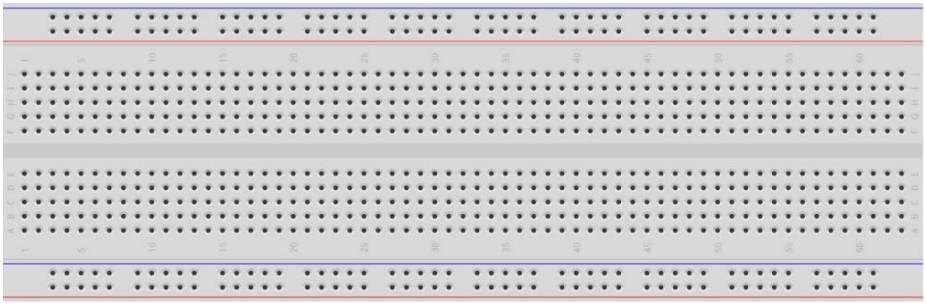
Chapter 12 Thermistor

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

Project 12.1 Thermometer

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

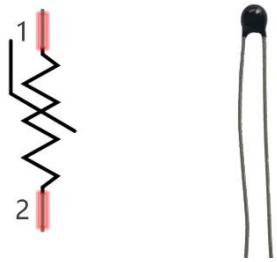
Component List

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;

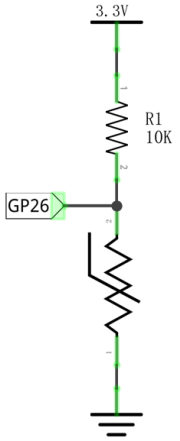
EXP[n] is nth power of e;

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.

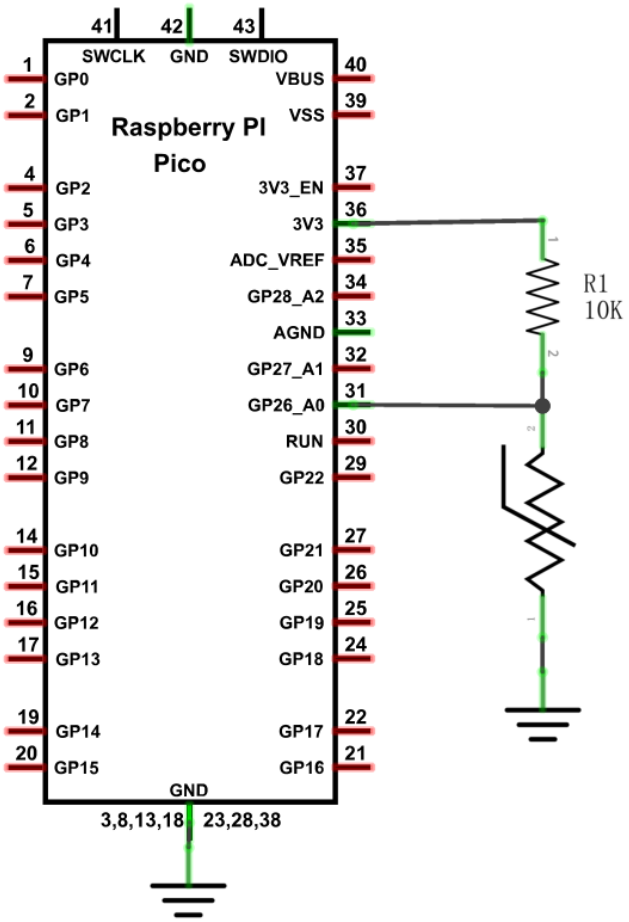
Therefore, the temperature formula can be derived as:

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

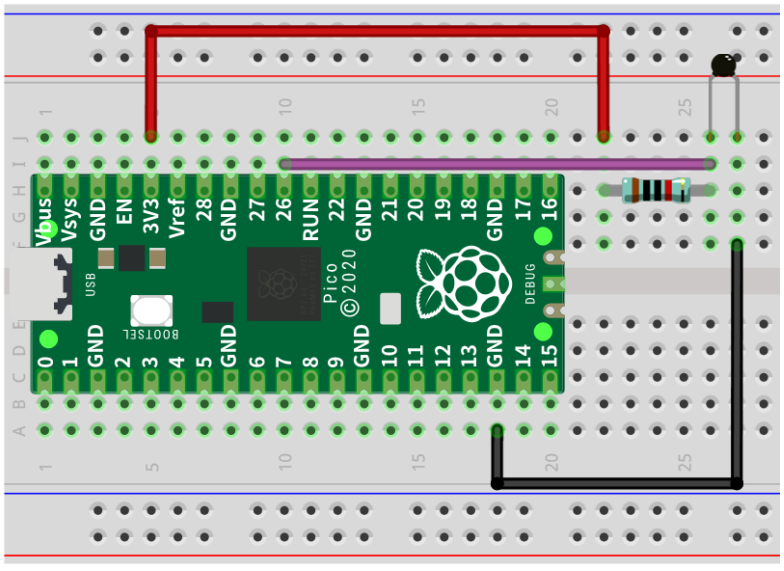
Circuit

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



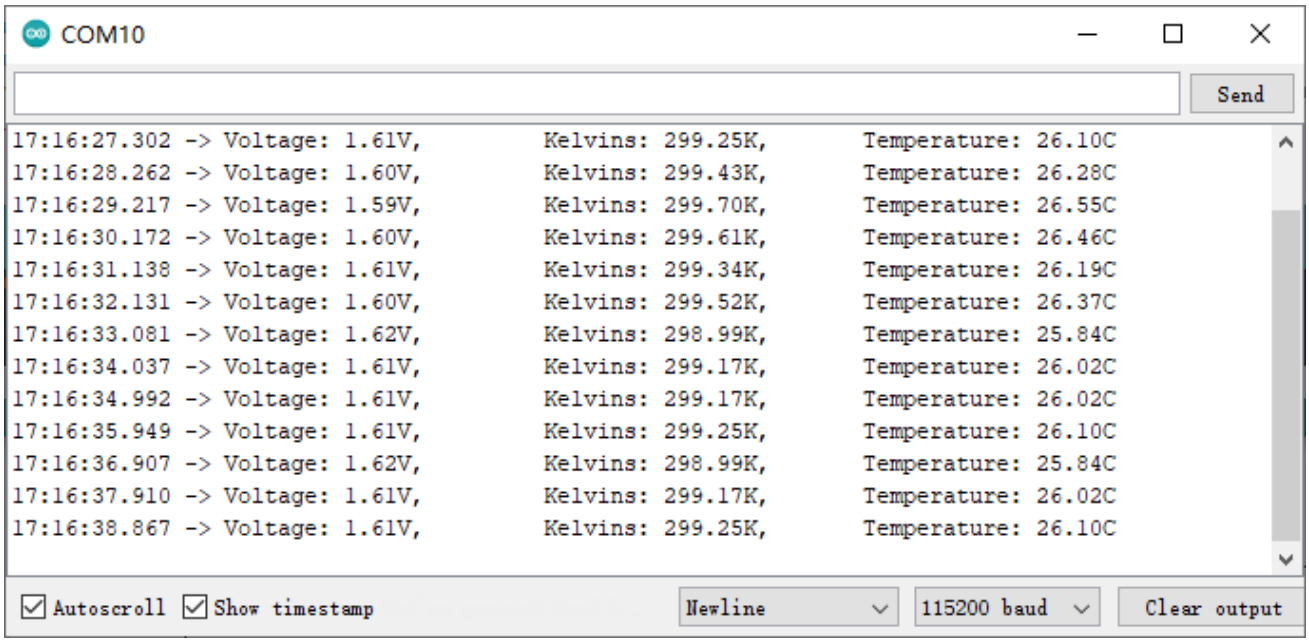
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_12.1_Thermometer

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



The following is the code:

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

void loop() {
  int adcValue = analogRead(PIN_ADC0); //read ADC pin
  double voltage = (float)adcValue / 1023.0 * 3.3;// calculate voltage
  double Rt = 10 * voltage / (3.3 - voltage);//calculate resistance value of thermistor
  double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0);//calculate temperature (Kelvin)
  double tempC = tempK - 273.15;//calculate temperature (Celsius)
  Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String(tempK) + "K,\t" + "Temperature: " + String(tempC) + "C");
  delay(1000);
}
```

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

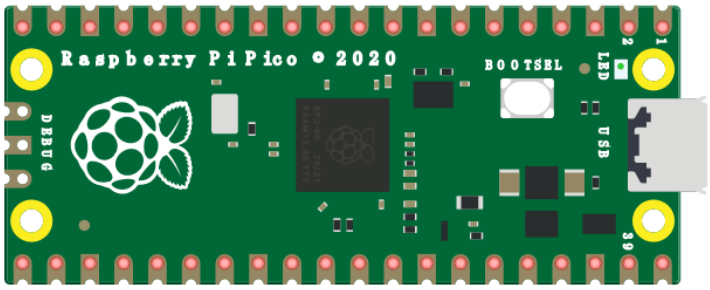
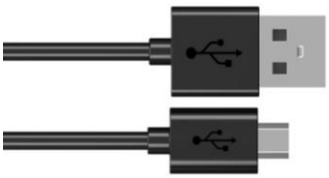
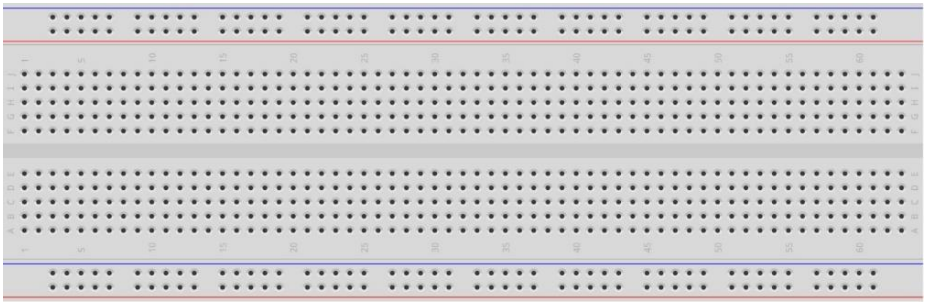
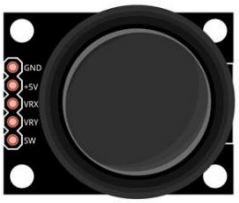
Chapter 13 Joystick

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

Project 13.1 Joystick

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

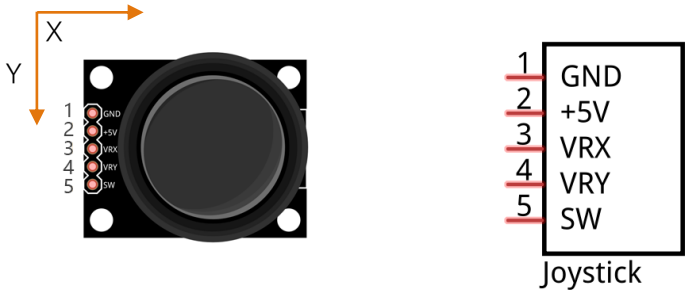
Component List

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

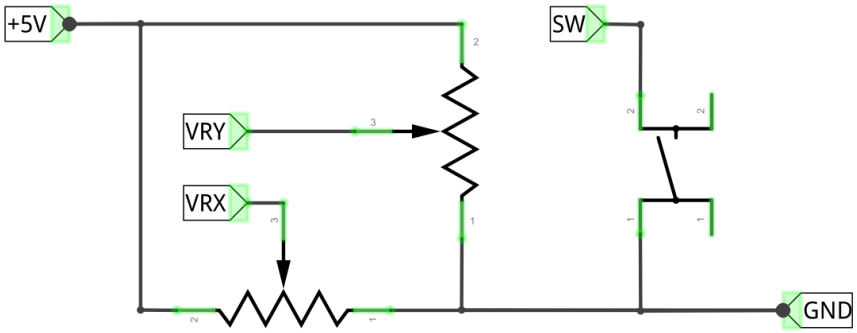
Component Knowledge

Joystick

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



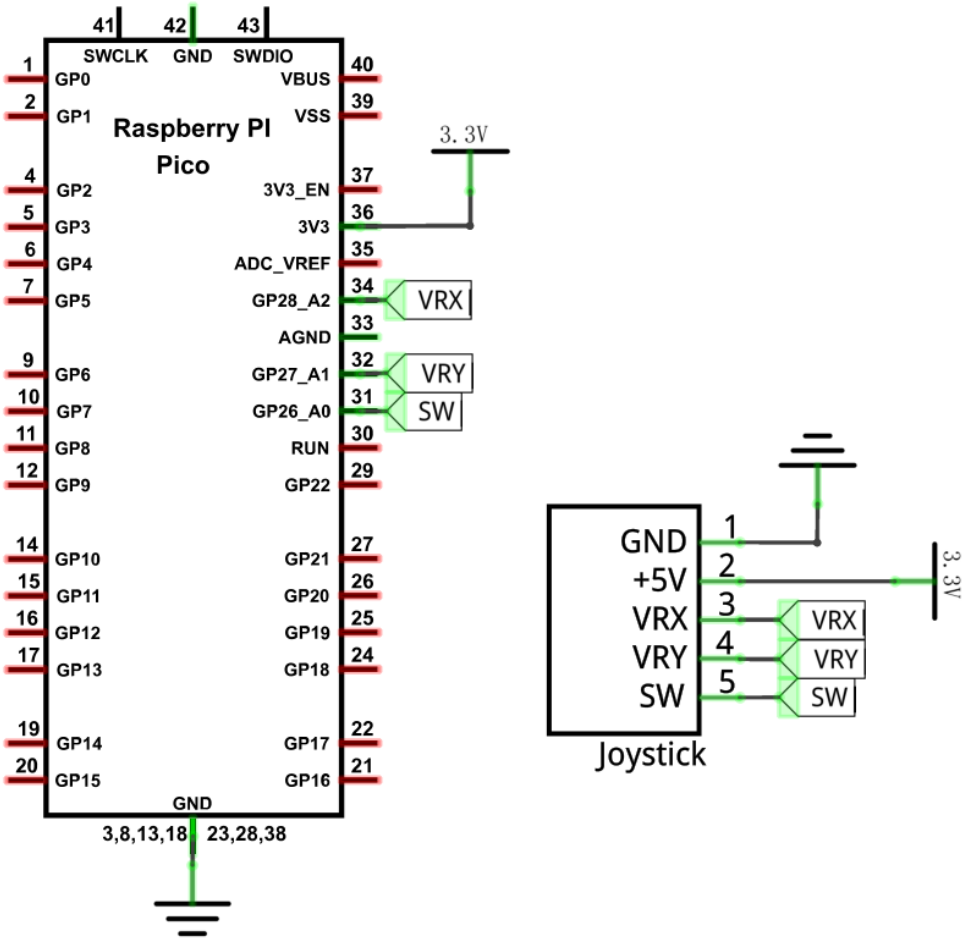
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



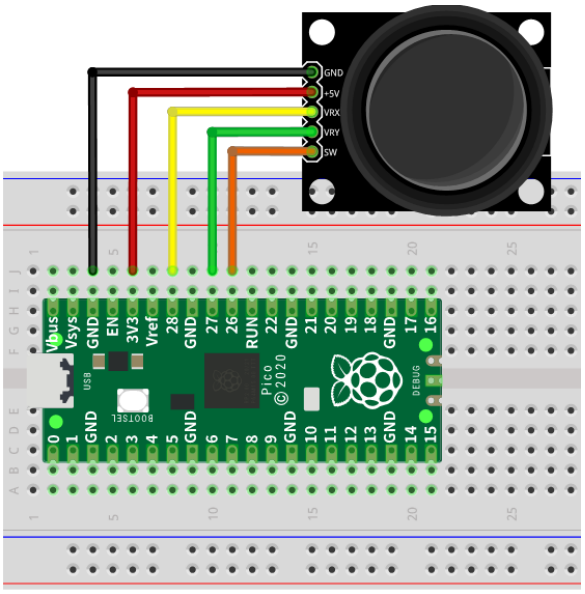
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



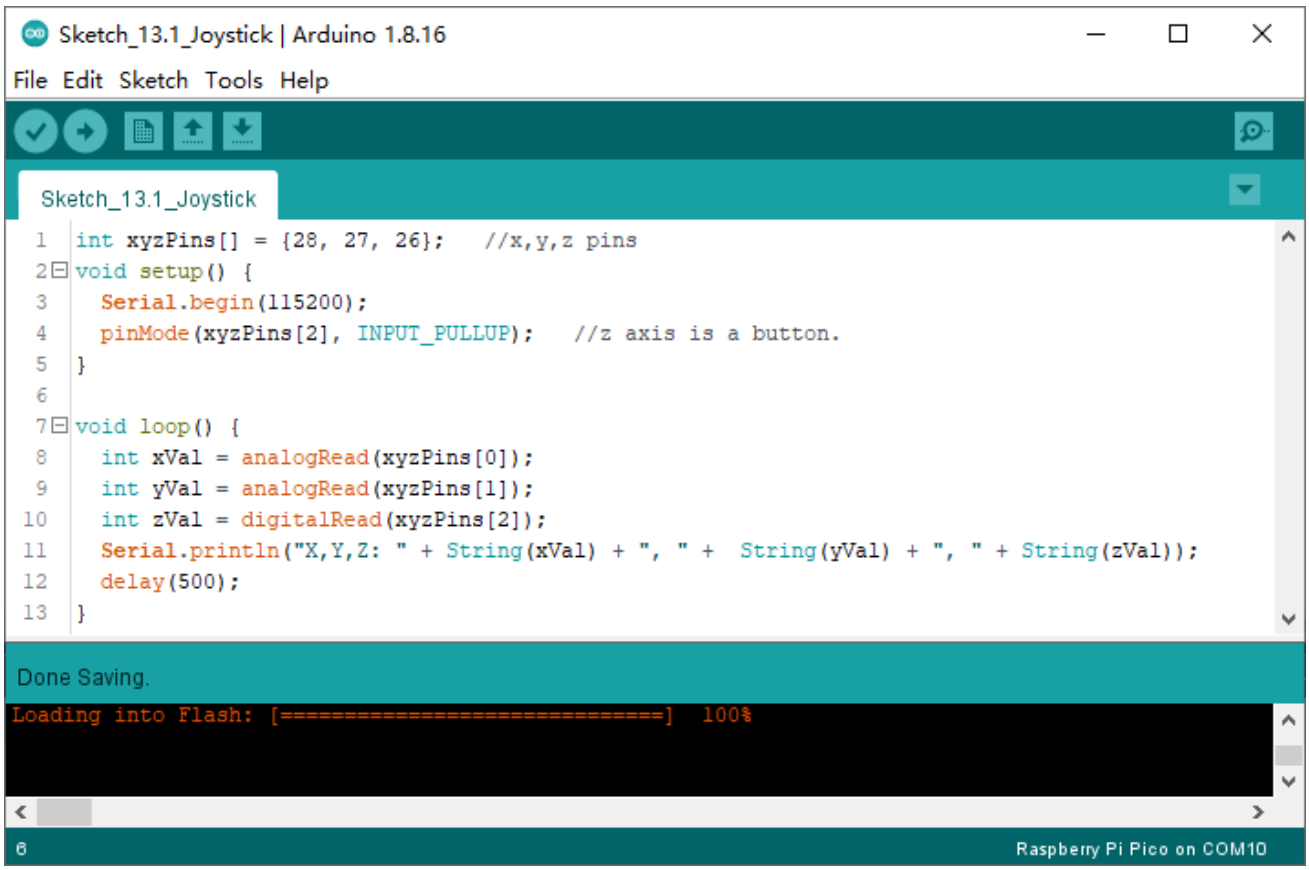
Hardware connection. If you need any support, please feel free to contact us via:support@freenove.com



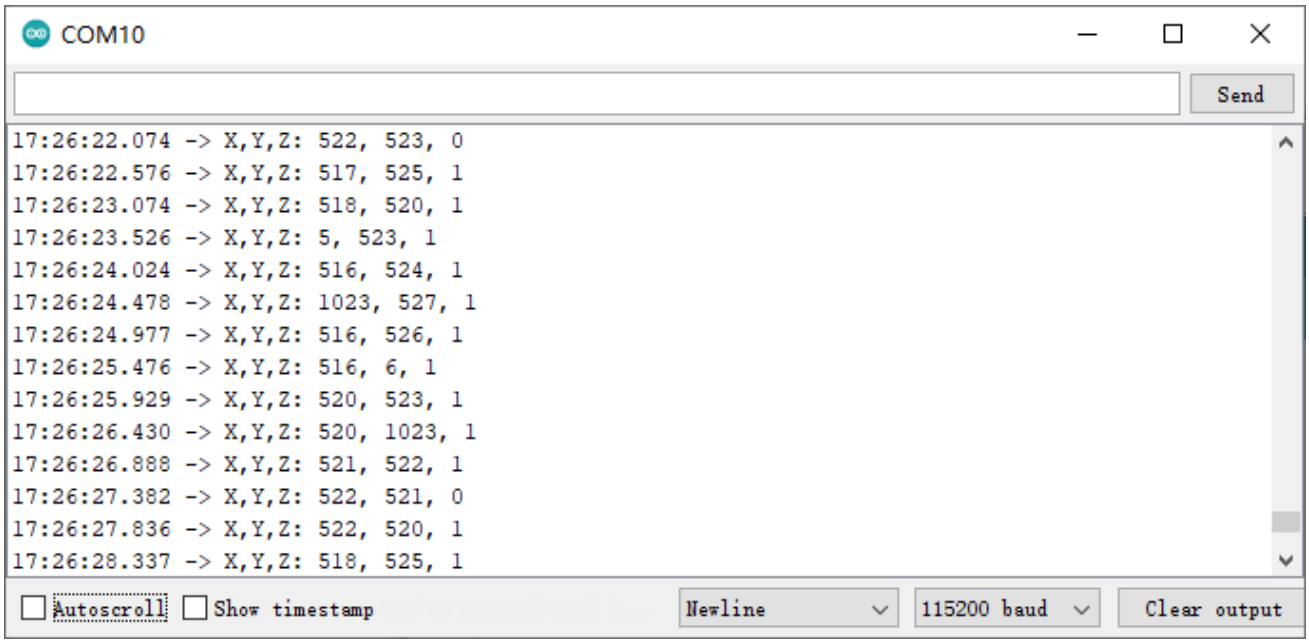
Sketch

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

Sketch_13.1_Joystick



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



The following is the code:

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

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

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

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

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

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

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

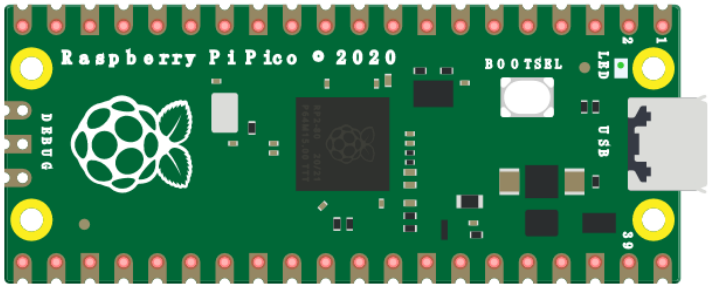
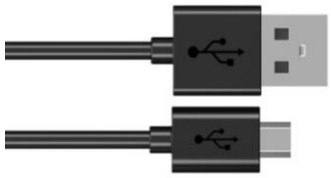
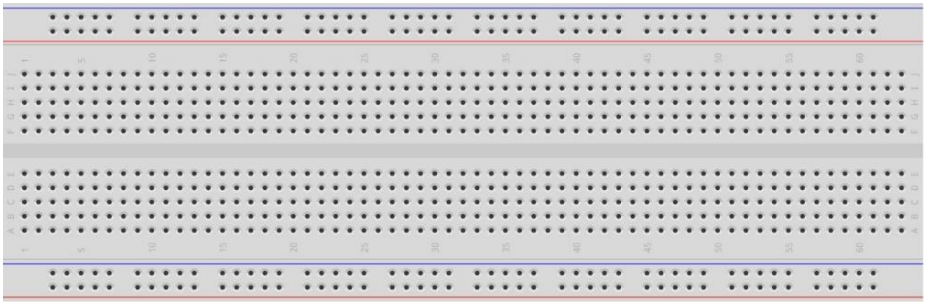
Chapter 14 74HC595 & LED Bar Graph

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

Project 14.1 Flowing Water Light

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

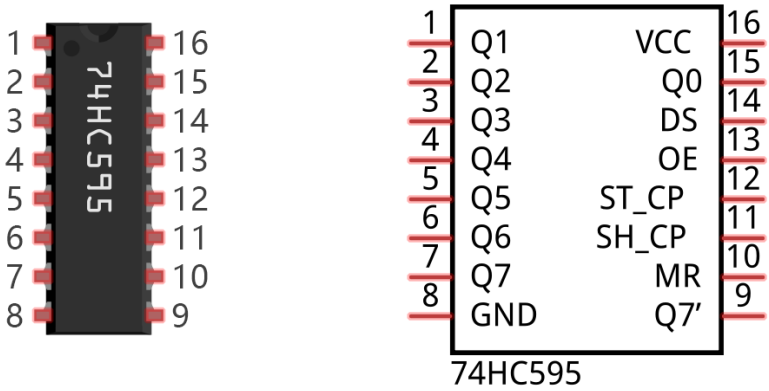
Component List

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

Related Knowledge

74HC595

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



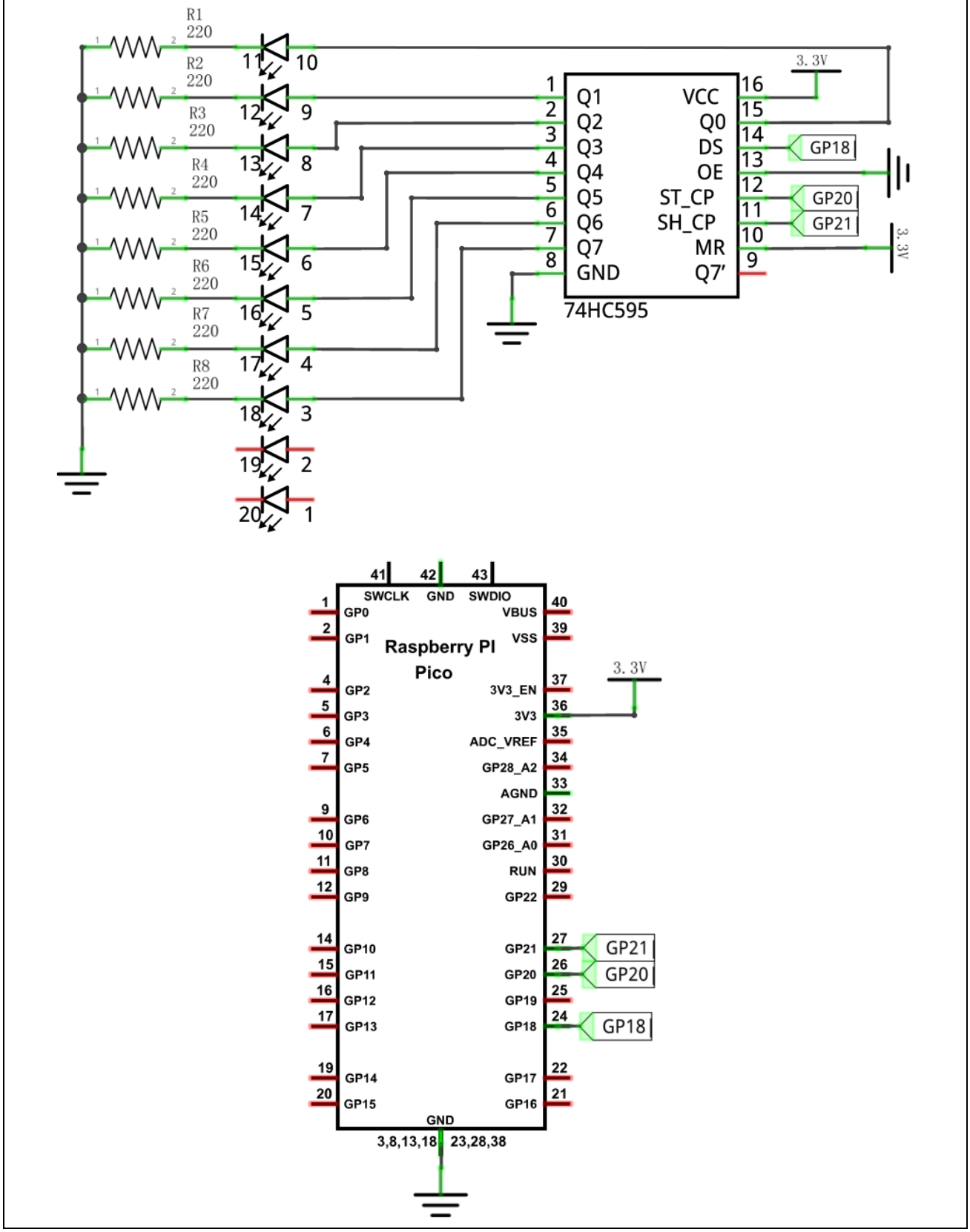
The ports of the 74HC595 chip are described as follows:

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

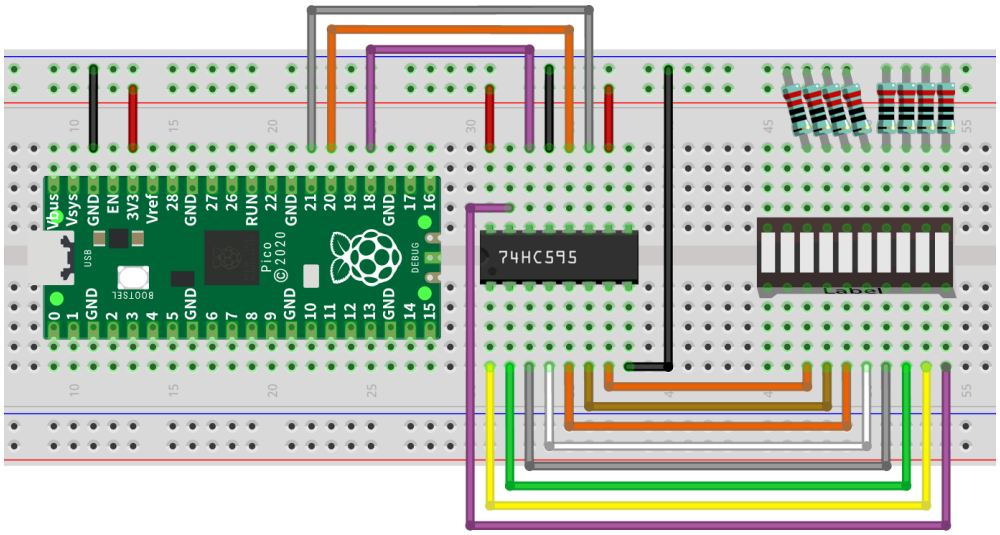
For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



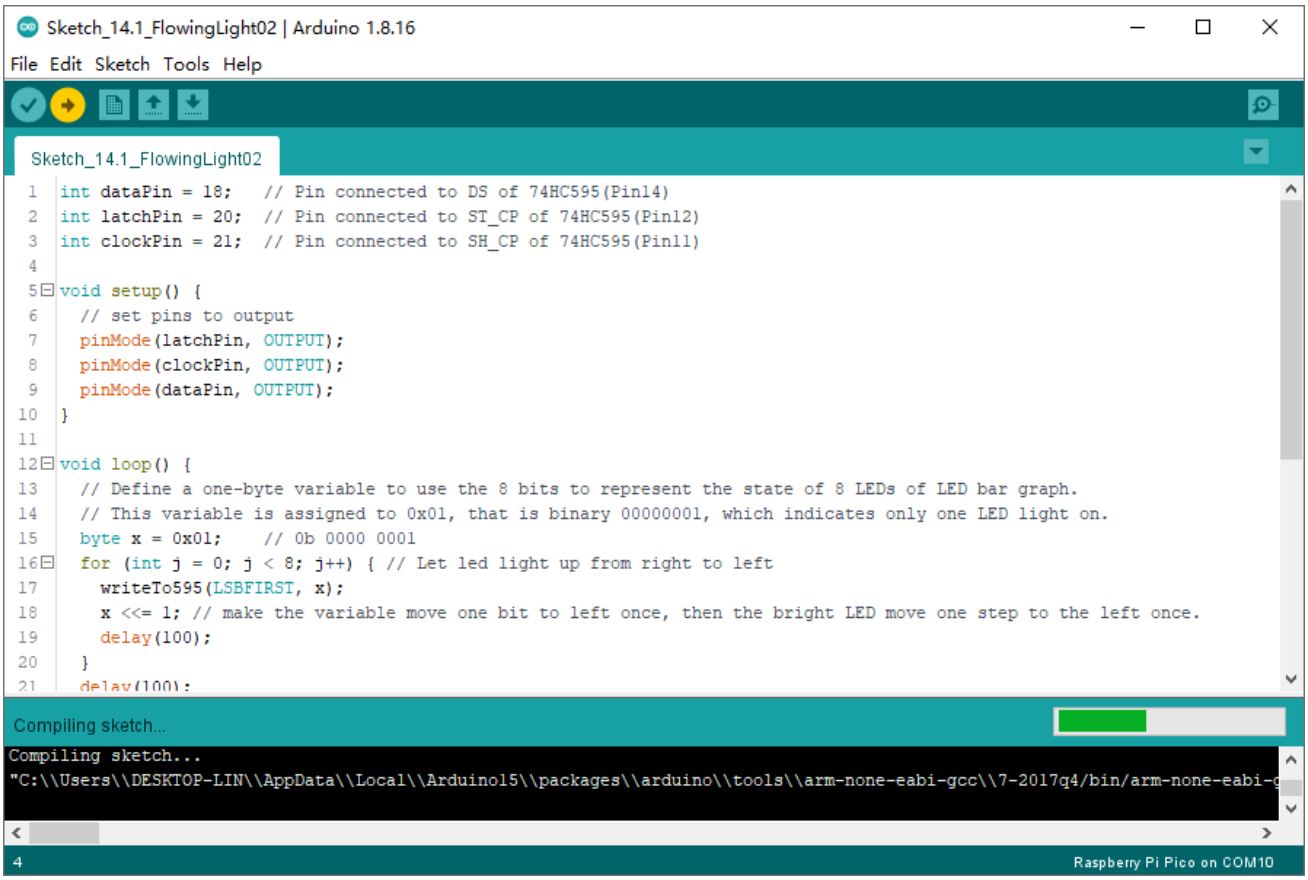
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

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

Sketch_14.1_FlowingLight2



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

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

The following is the program code:

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

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

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

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

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

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

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

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

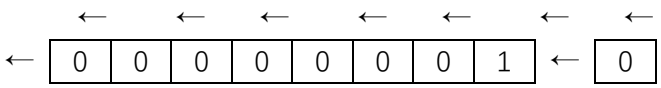
The subfunction writeTo595() is used to write data to 74HC595 and immediately output on the port of 74HC595.

Reference

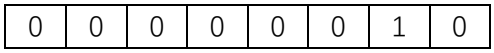
<< operator

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

byte x = 1 << 1;

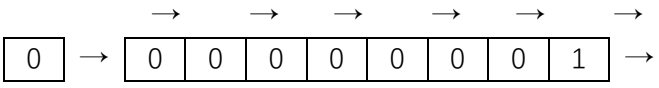


The result of x is 2（binary 00000010）.

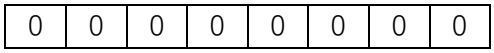


There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

byte x = 1 >> 1;



The result of x is 0（00000000）.



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

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

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

Parameters

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

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

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

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

For more details about shift function, please refer to:

https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/

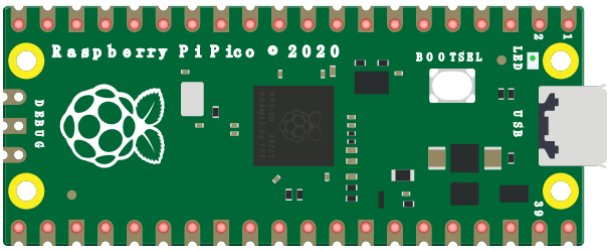
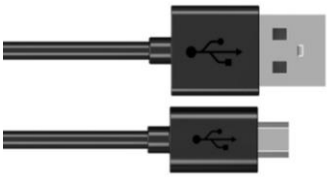
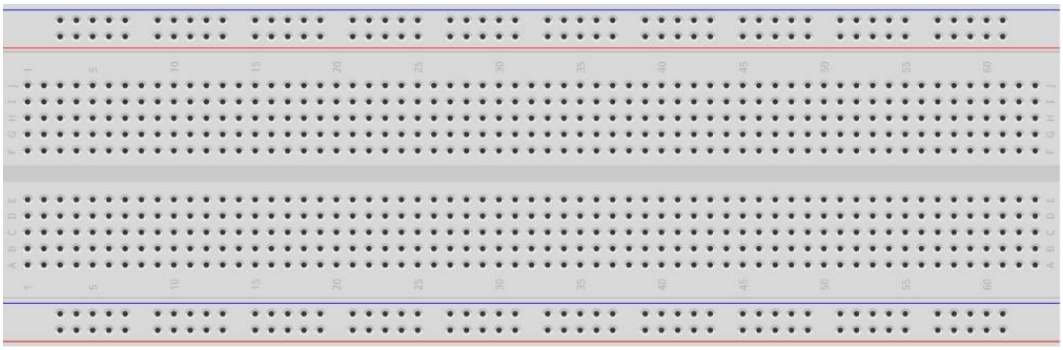
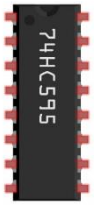
Chapter 15 74HC595 & 7-Segment Display.

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

Project 15.1 7-Segment Display.

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

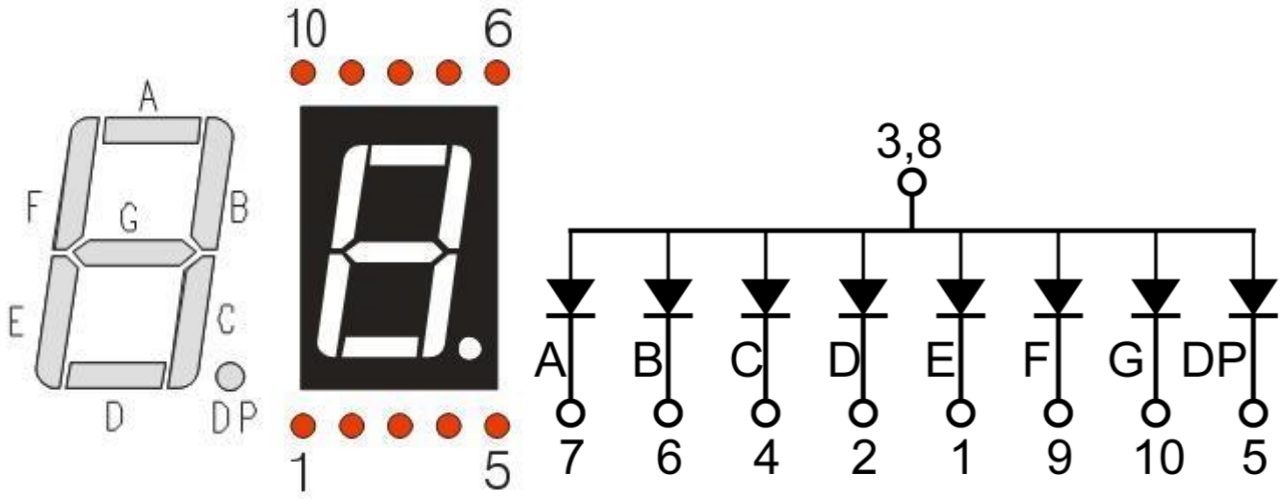
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x1
  </td><td>7-segment display x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper
</td></tr>
</table>

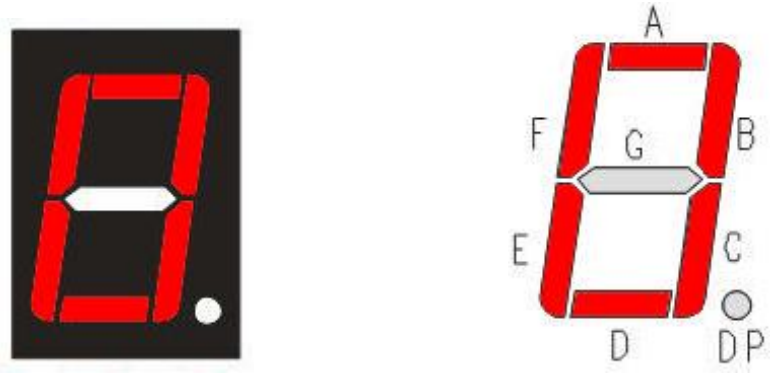
Component Knowledge

7-segment display

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



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



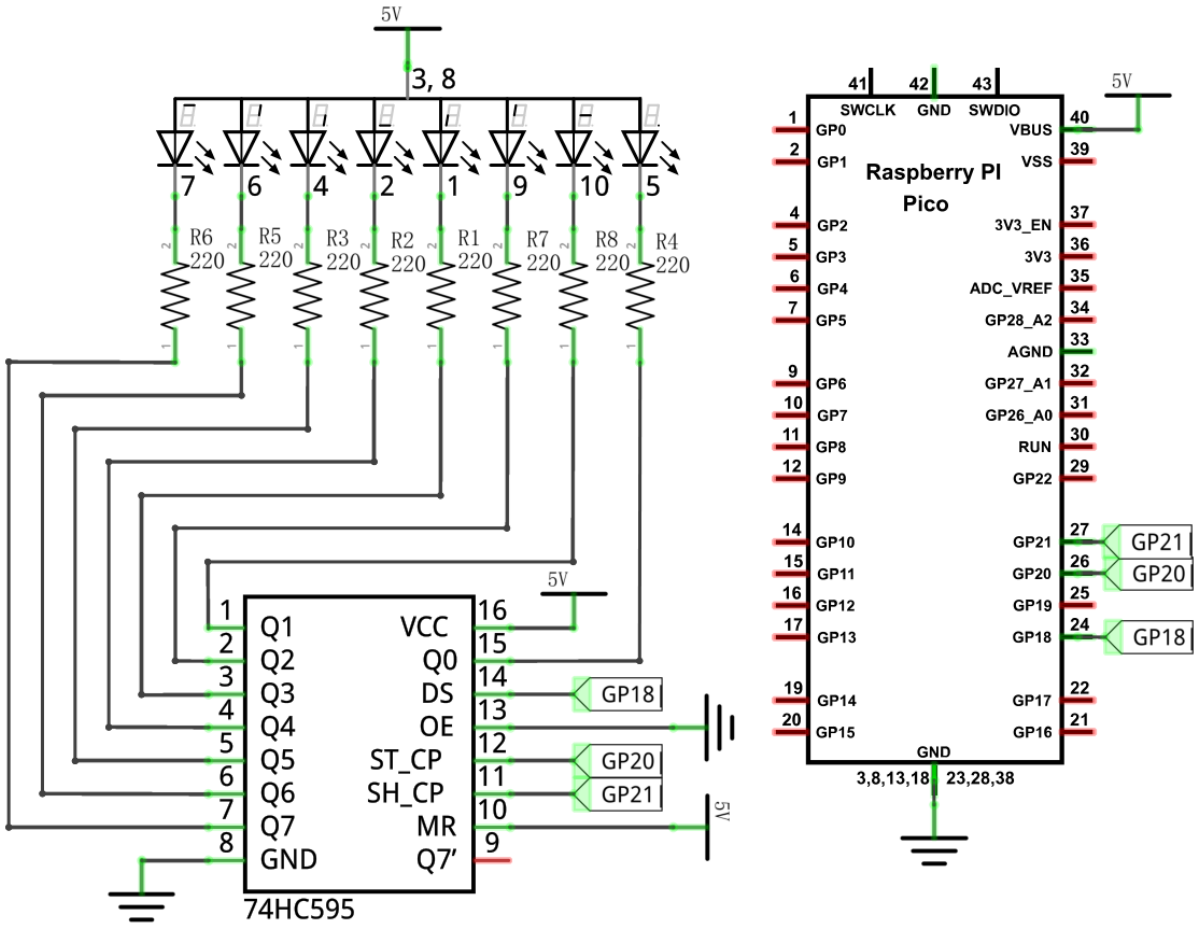
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”,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

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

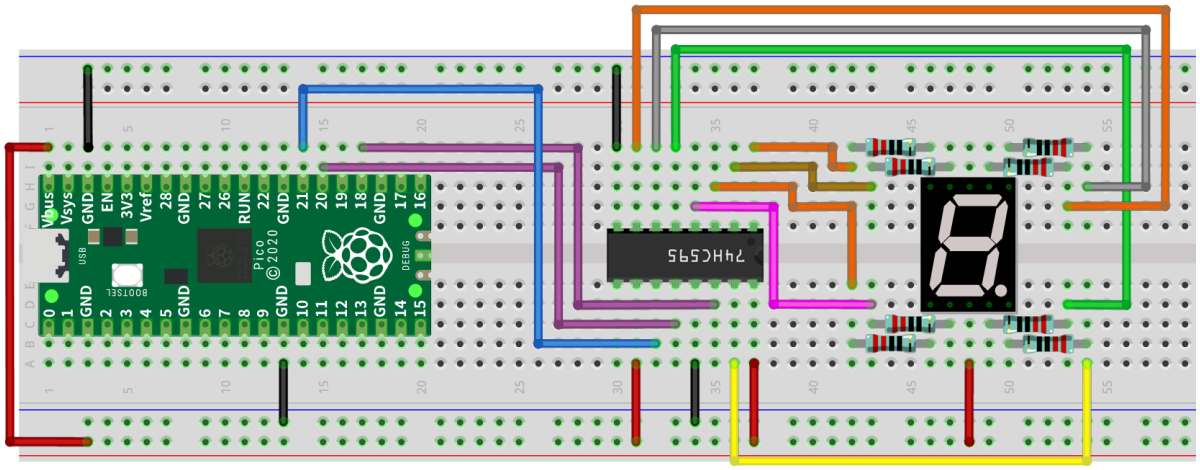
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0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

Circuit

Schematic diagram



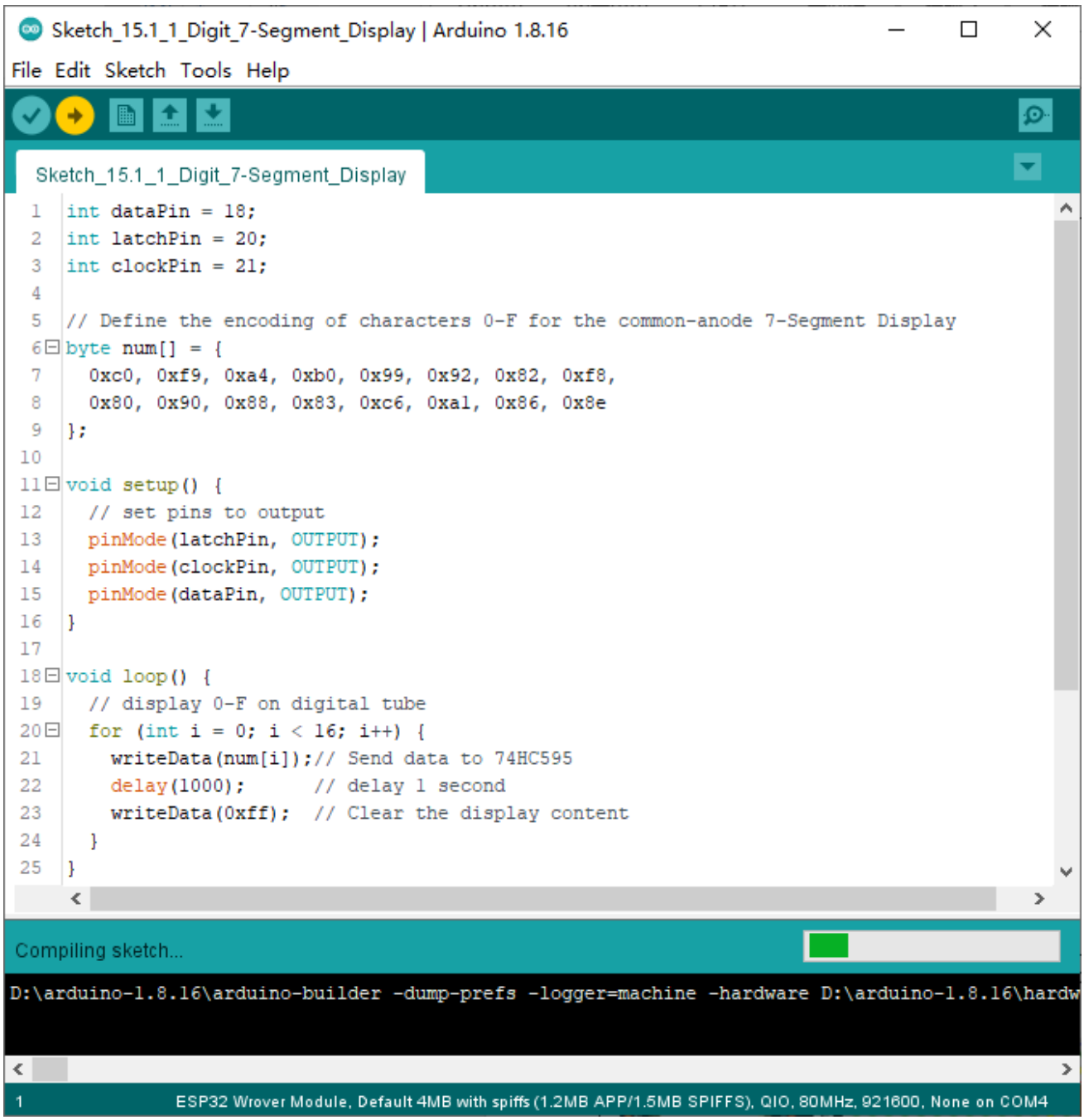
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



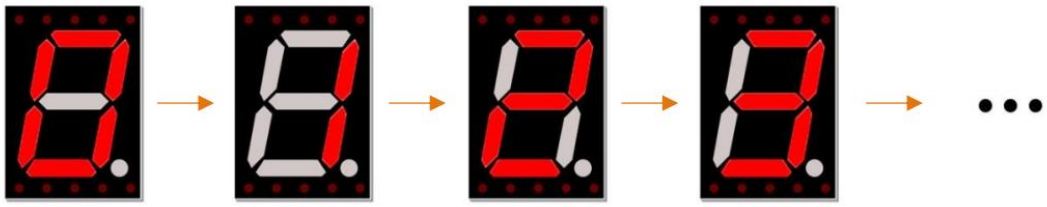
Sketch

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

Sketch_15.1_7_Segment_Display



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



The following is the program code:

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

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

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

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

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

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

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

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

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

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

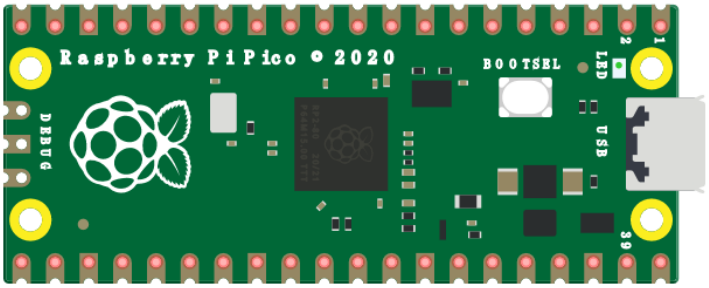
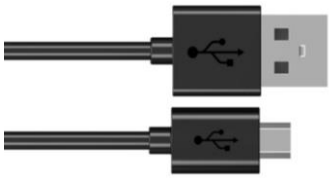
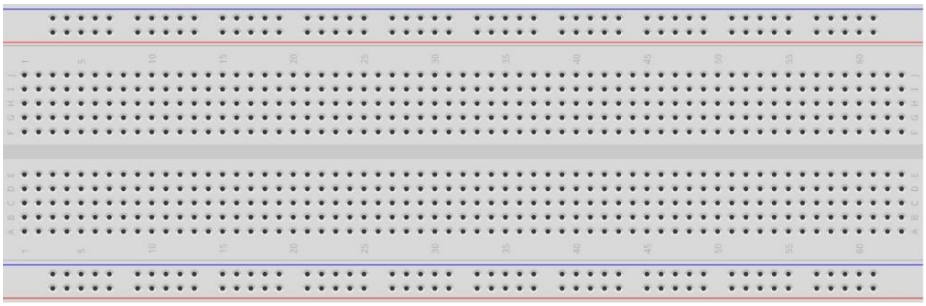
If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

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

Project 15.2 4-Digit 7-Segment Display

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

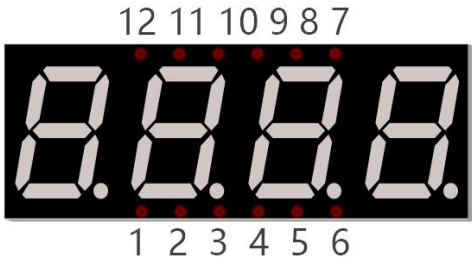
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x1
  </td><td>7-segment display x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper
</td></tr>
</table>

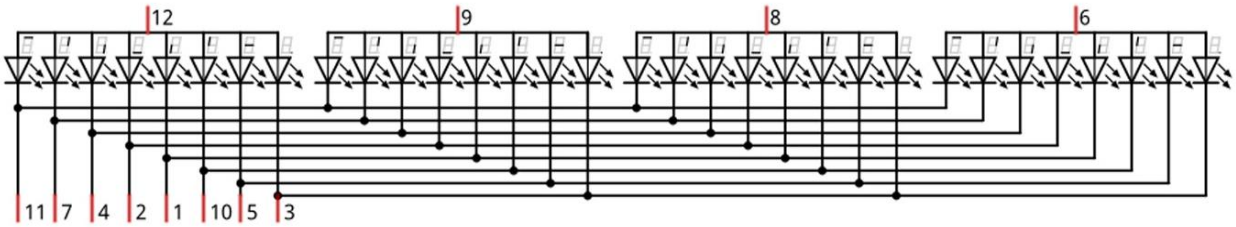
Component Knowledge

4 Digit 7-Segment Display

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



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

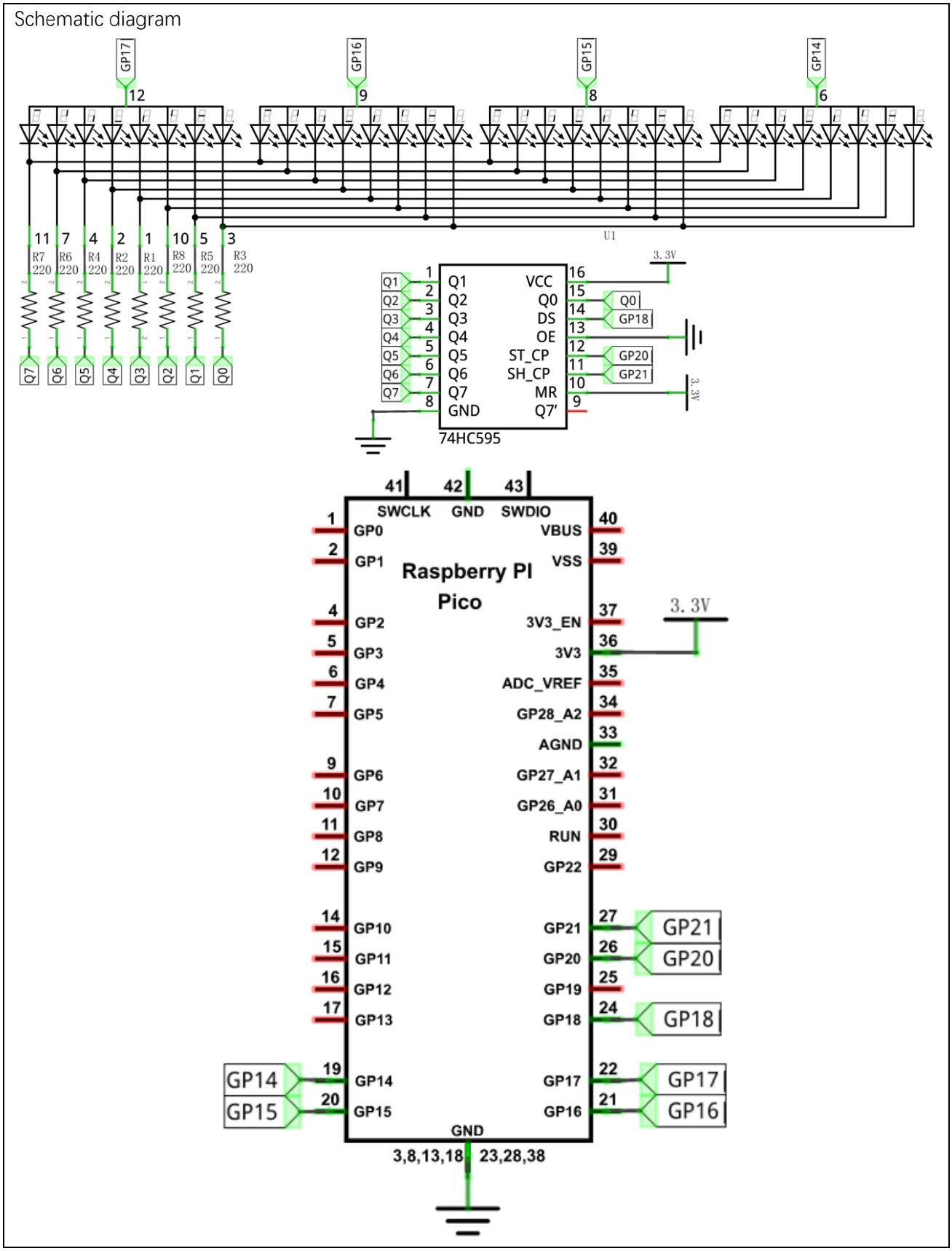


Display method of 4-Digit 7-segment display is similar to 1-Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit visibly in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the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

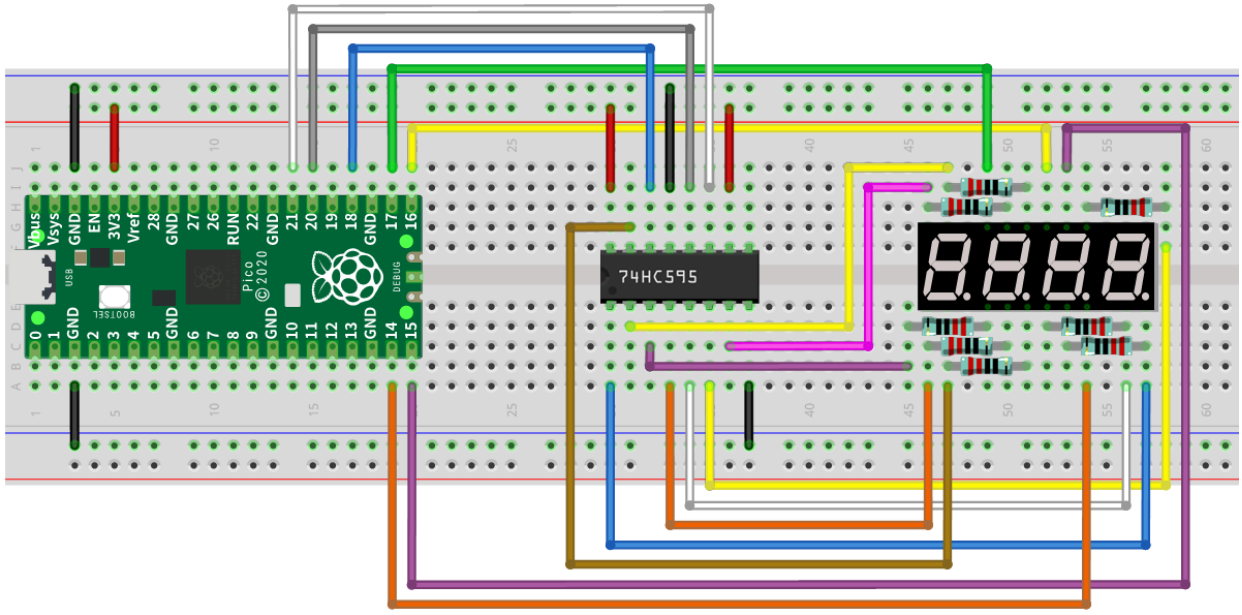
Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is imperceptible to the naked eye. This is due to the principle of optical afterglow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

Circuit

Schematic diagram



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



Sketch

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

Sketch_16.2_4_Dight_7-Segment_Display

Sketch_15.2_4_Digit_7-Segment_Display | Arduino 1.8.16

File Edit Sketch Tools Help

Sketch_15.2_4_Digit_7-Segment_Display

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

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

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

The following is the program code:

```
int latchPin = 18;           // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 20;           // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin =  21;           // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {17,16,15,14};// Common pin (anode) of 4 digit 7-segment display

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

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

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

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

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

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

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

Second, initialize all the pins to output mode.

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

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

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

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

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

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

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

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

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

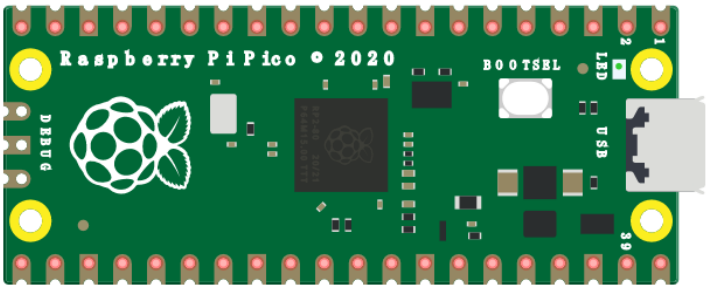
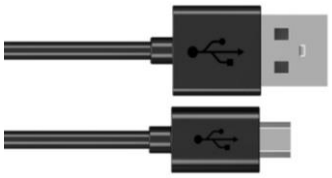
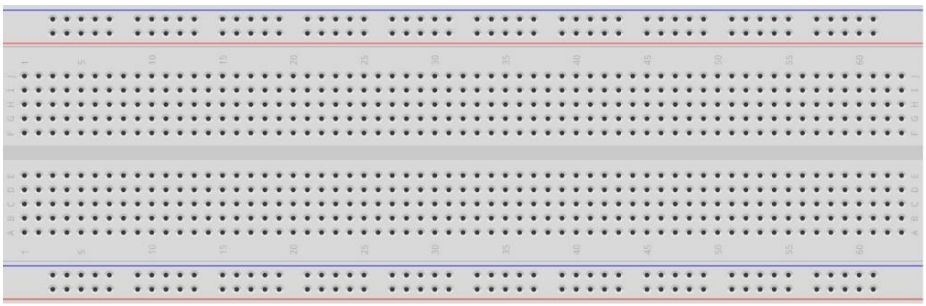
Chapter 16 74HC595 & LED Matrix

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

Project 16.1 LED Matrix

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

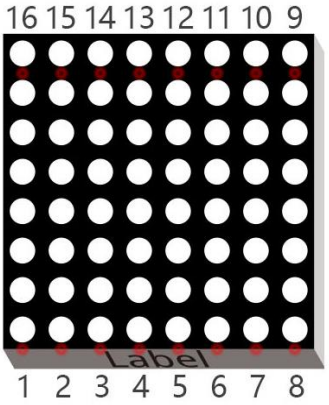
Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>74HC595 x2
  </td><td>8*8 LEDMatrix x1
  </td><td>Resistor 220Ω x8
  </td><td>Jumper
</td></tr>
</table>

Component Knowledge

LED matrix

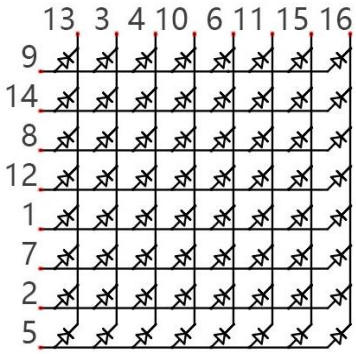
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



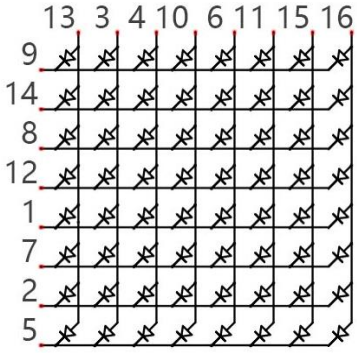
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

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

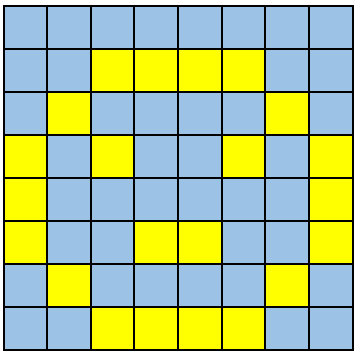
Connection mode of common anode



Connection mode of common cathode



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



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

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

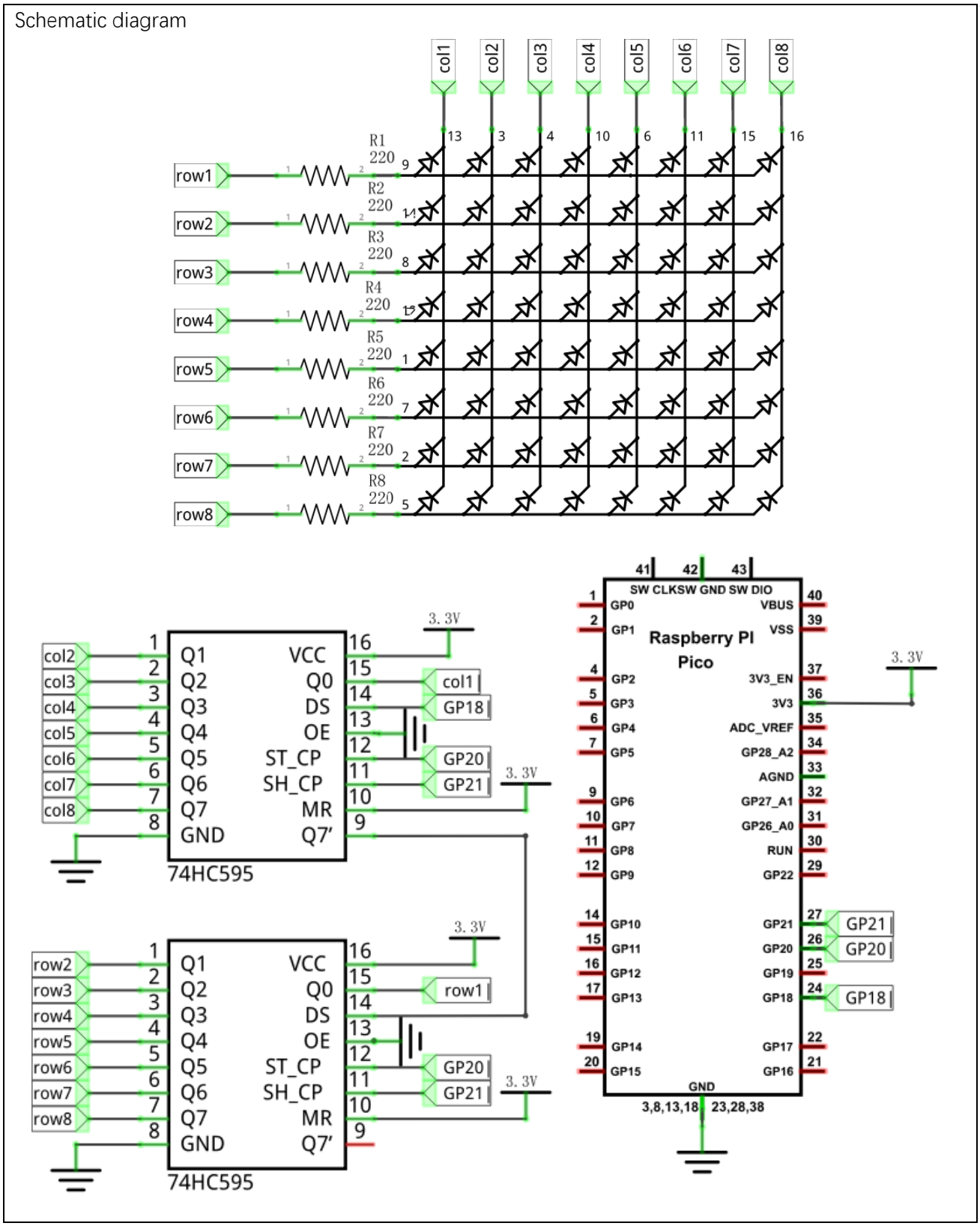
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated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

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

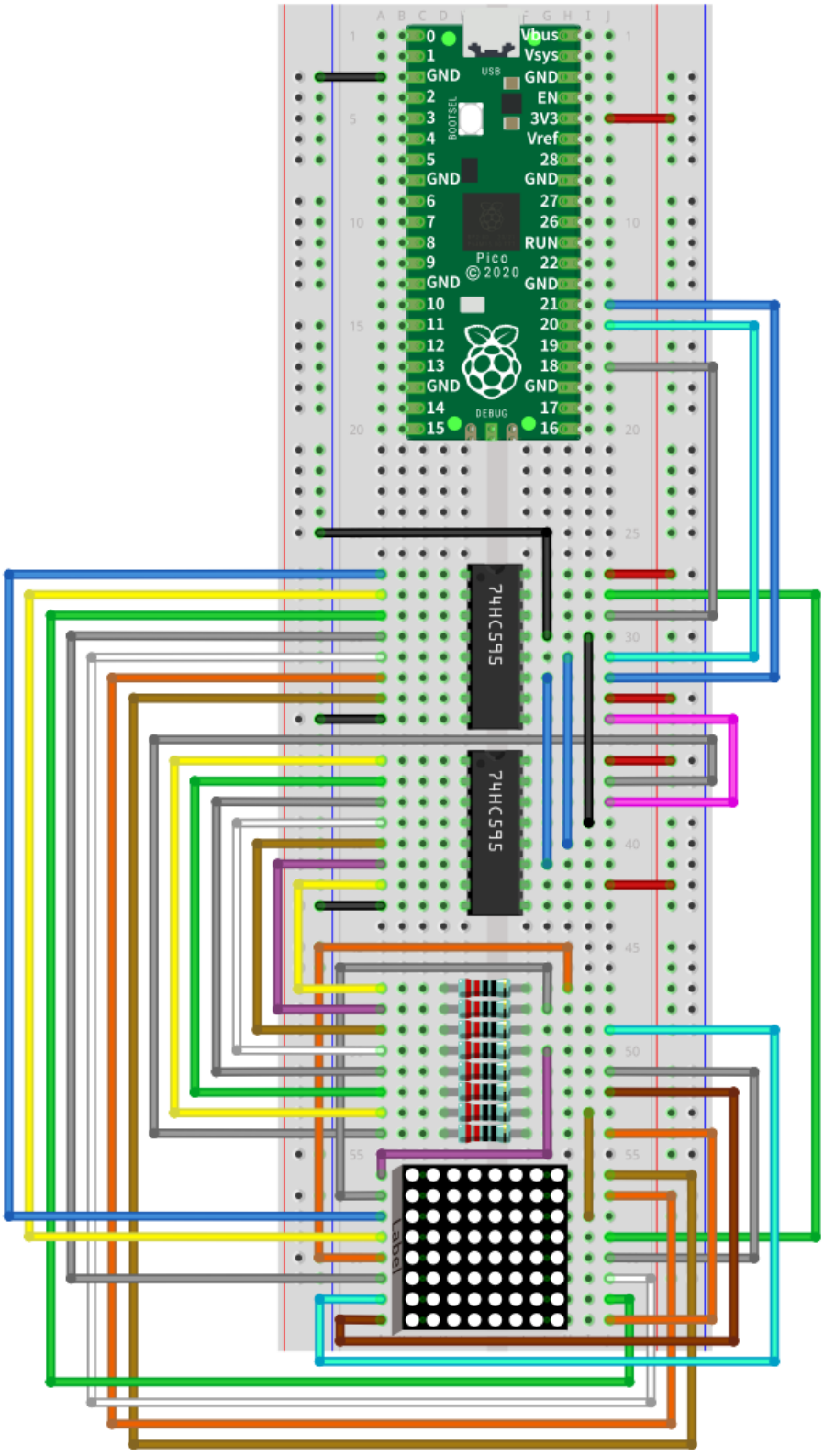
Circuit

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

Schematic diagram

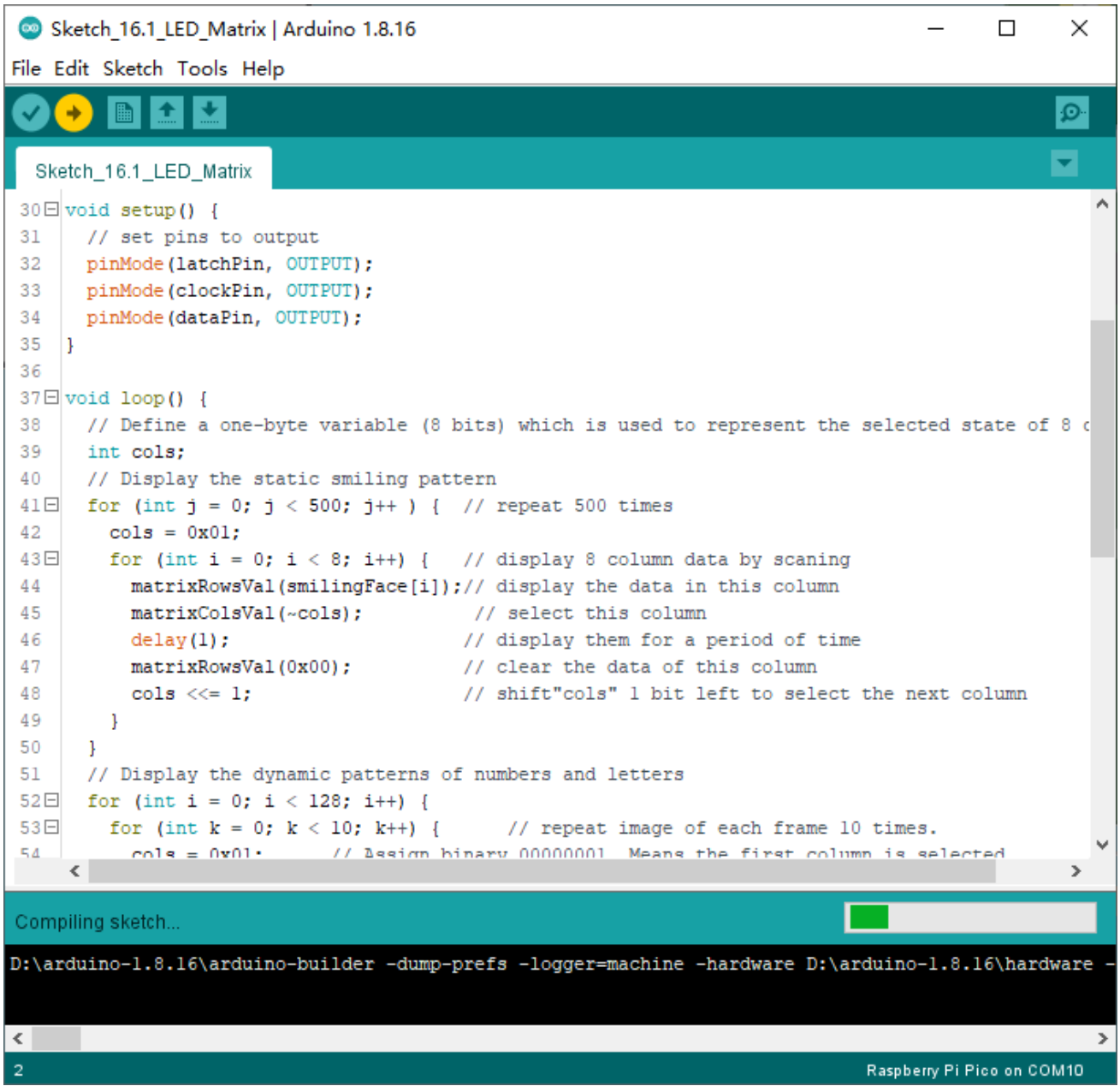


Hardware connection. If you need any support, please feel free to contact us via:support@freenove.com



Sketch

The following code will make LED matrix display a smiling face, and then display scrolling character "0-F".
Sketch_16.1_LED_Matrix



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

The following is the program code:

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

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

```
47        matrixRowsVal(0x00);          // clear the data of this column
48        cols <<= 1;                   // shift "cols" 1 bit left to select the next column
49      }
50    }
51    // Display the dynamic patterns of numbers and letters
52    for (int i = 0; i < 128; i++) {
53      for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
54        cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
55        for (int j = i; j < 8 + i; j++) { // display image of each frame
56          matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
57          matrixColsVal(~cols);             // select this column
58          delay(1);                         // display them for a period of time
59          matrixRowsVal(0x00);              // close the data of this column
60          cols <<= 1;                   // shift "cols" 1 bit left to select the next column
61        }
62      }
63    }
64  }
65
66  void matrixRowsVal(int value) {
67    // make latchPin output low level
68    digitalWrite(latchPin, LOW);
69    // Send serial data to 74HC595
70    shiftOut(dataPin, clockPin, LSBFIRST, value);
71    // make latchPin output high level, then 74HC595 will update the data to parallel output
72    digitalWrite(latchPin, HIGH);
73  }
74
75  void matrixColsVal(int value) {
76    // make latchPin output low level
77    digitalWrite(latchPin, LOW);
78    // Send serial data to 74HC595
79    shiftOut(dataPin, clockPin, MSBFIRST, value);
80    // make latchPin output high level, then 74HC595 will update the data to parallel output
81    digitalWrite(latchPin, HIGH);
82  }
```

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

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

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

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

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

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

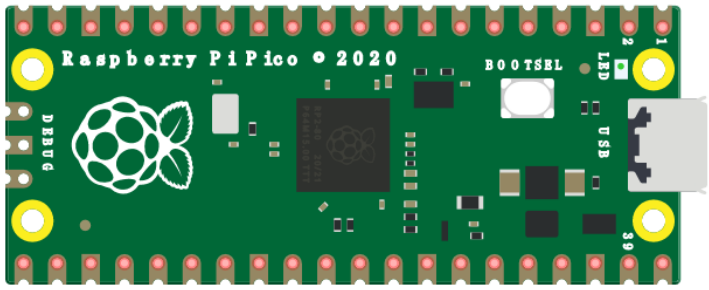
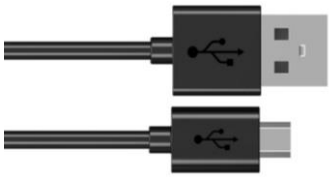
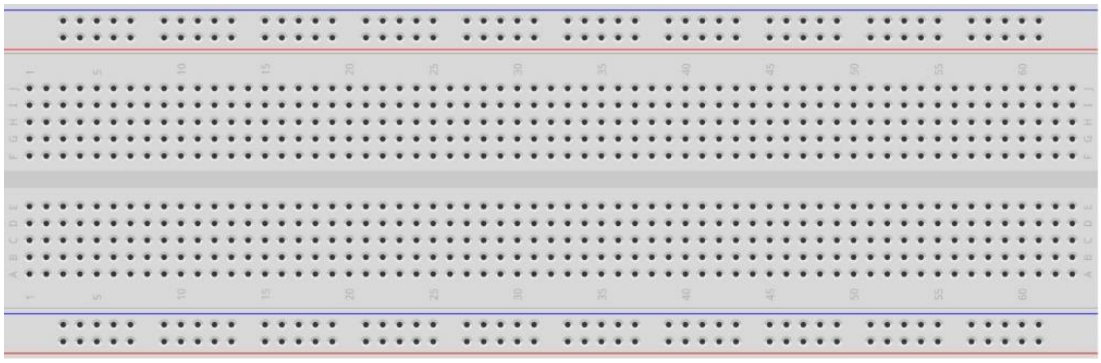
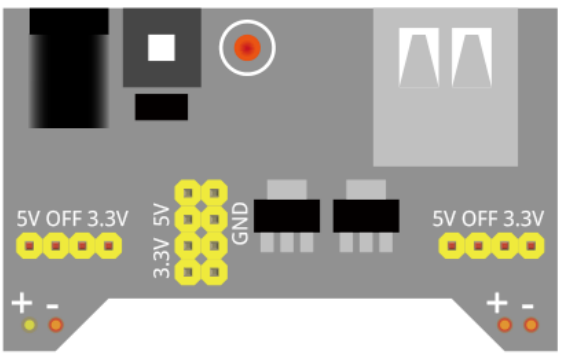
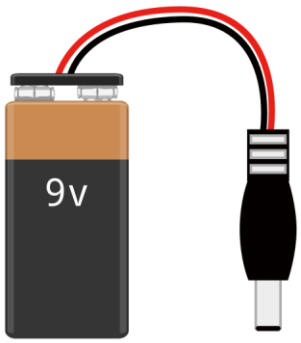
Chapter 17 Relay & Motor

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

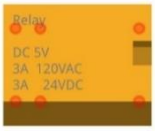
Project 17.1 Relay & Motor

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

Component List

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

Resistor 10kΩ x2 	Resistor 1kΩ x1 	Resistor 220Ω x1 

NPN transistor x1 	Relay x1 	Motor x1 	Push button x1 	LED x1 	Diode x1 

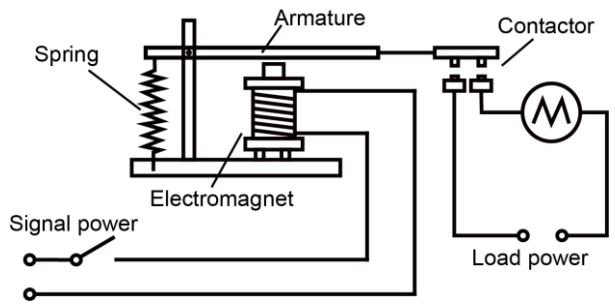
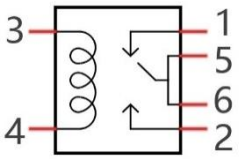
Jumper 

Component Knowledge

Relay

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

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

Diagram	Feature:	Symbol
		

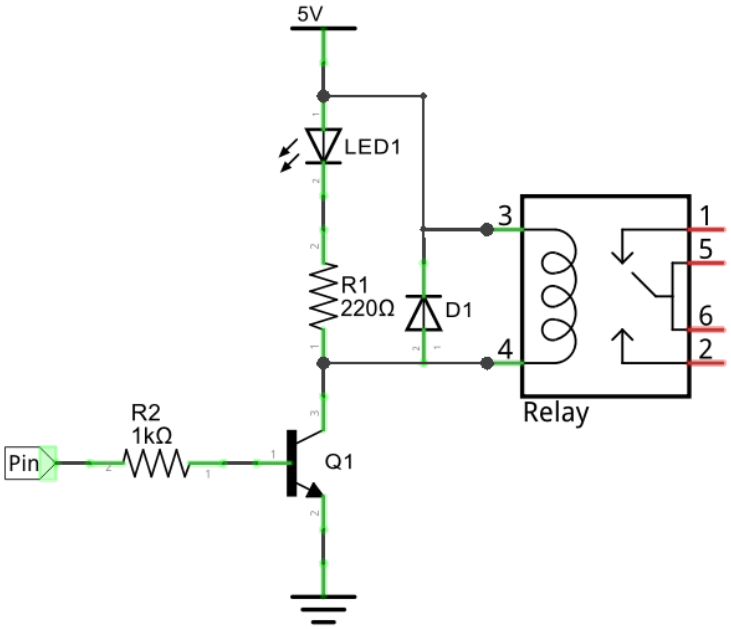
Pin 5 and pin 6 are connected to each other inside. When the coil pins 3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

Inductor

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

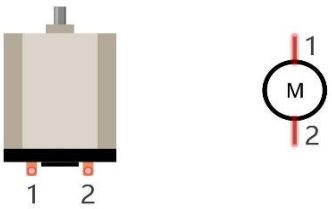
An inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductors hinder the change of current passing through it. When the current passing through it increases, it will attempt to hinder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

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

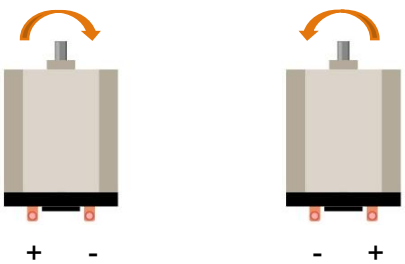


Motor

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

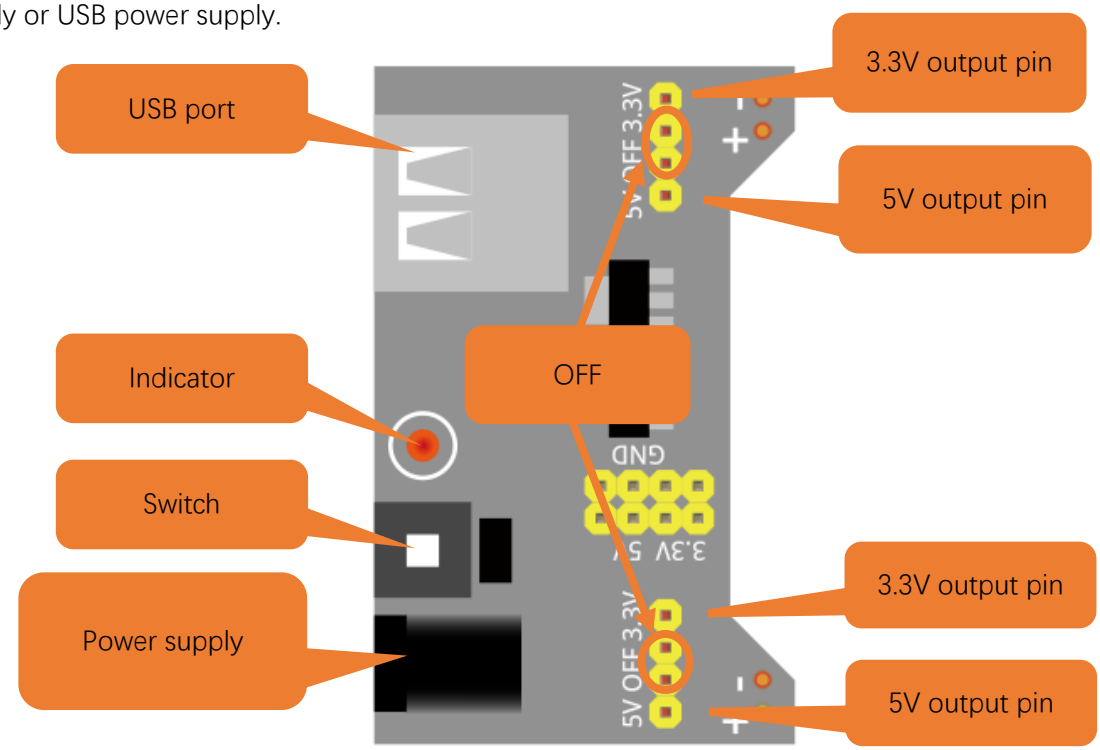


When a motor gets connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.



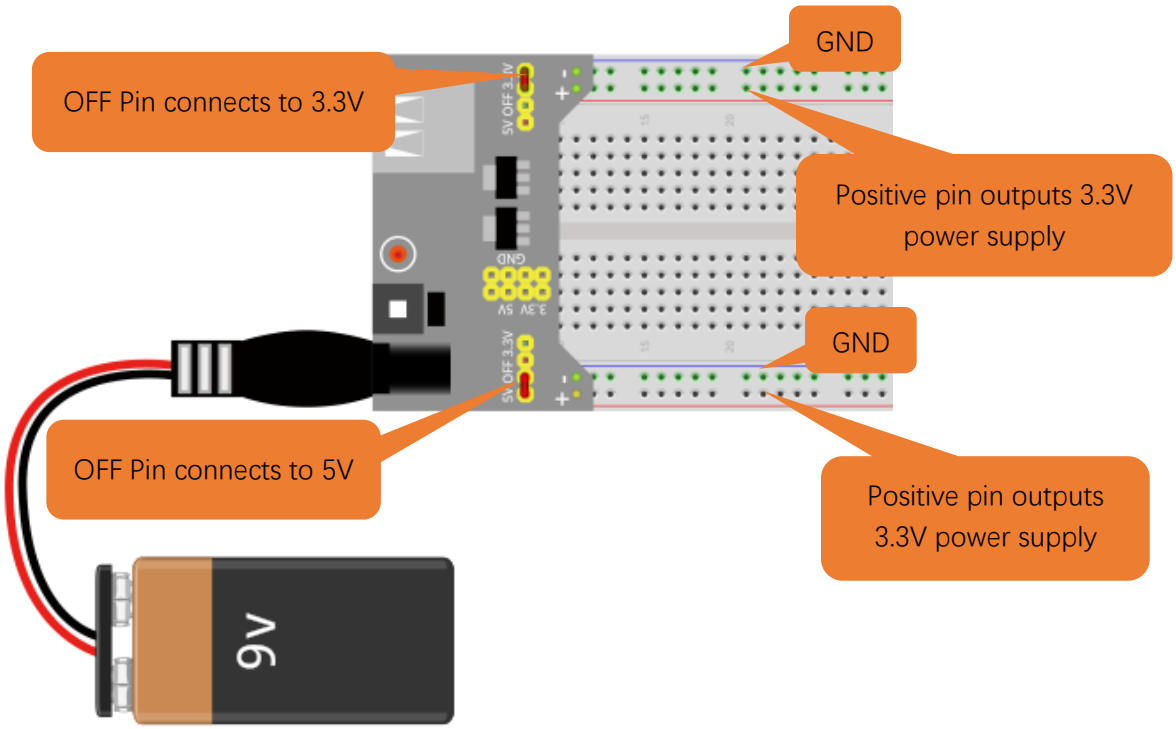
BreadBoardPower

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



Usage:

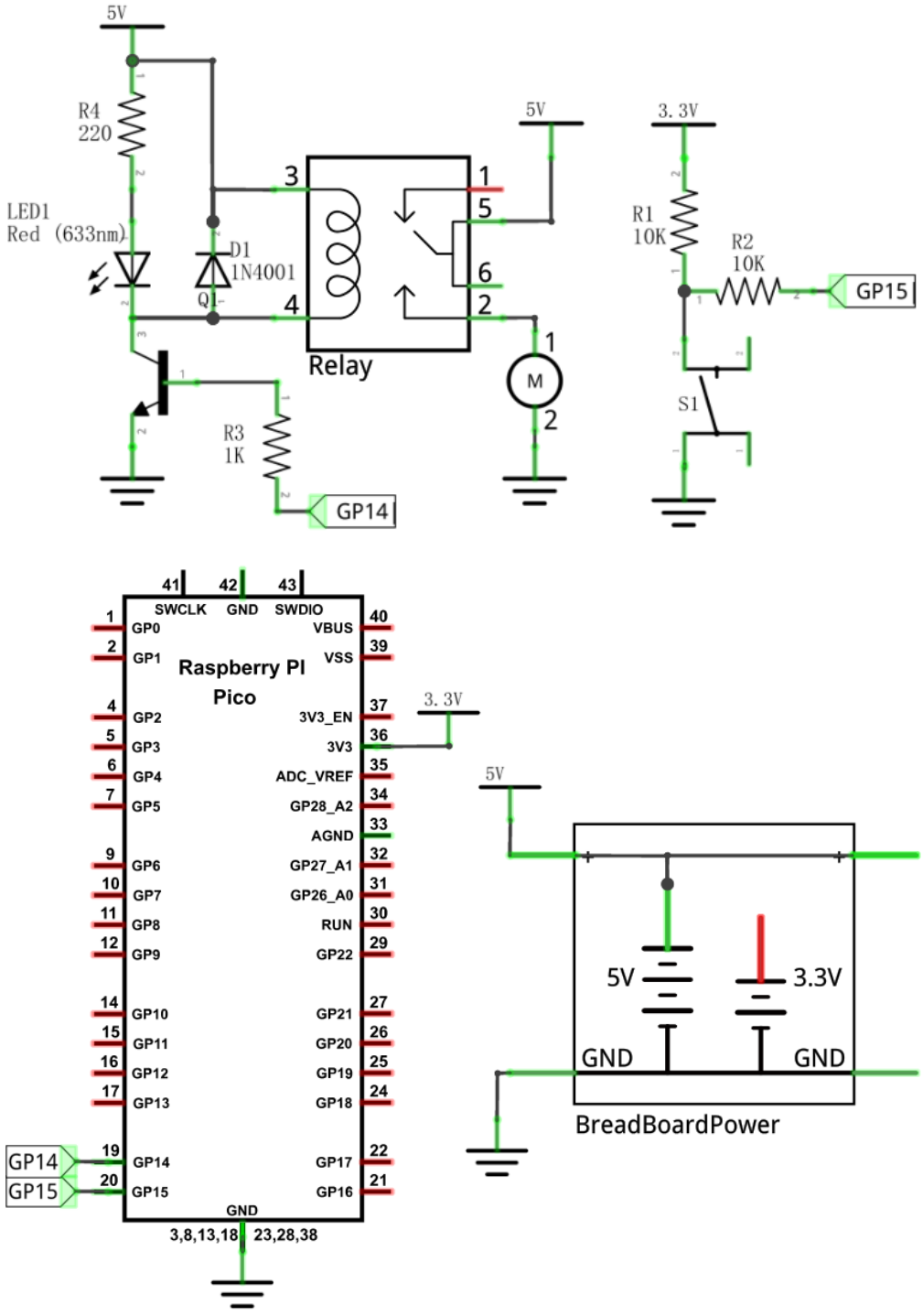
First connect BreadBoardPower to the power supply, and then selectively choose the connection between OFF and 5V, 3V to generate different voltage source outputs. as the picture shows.



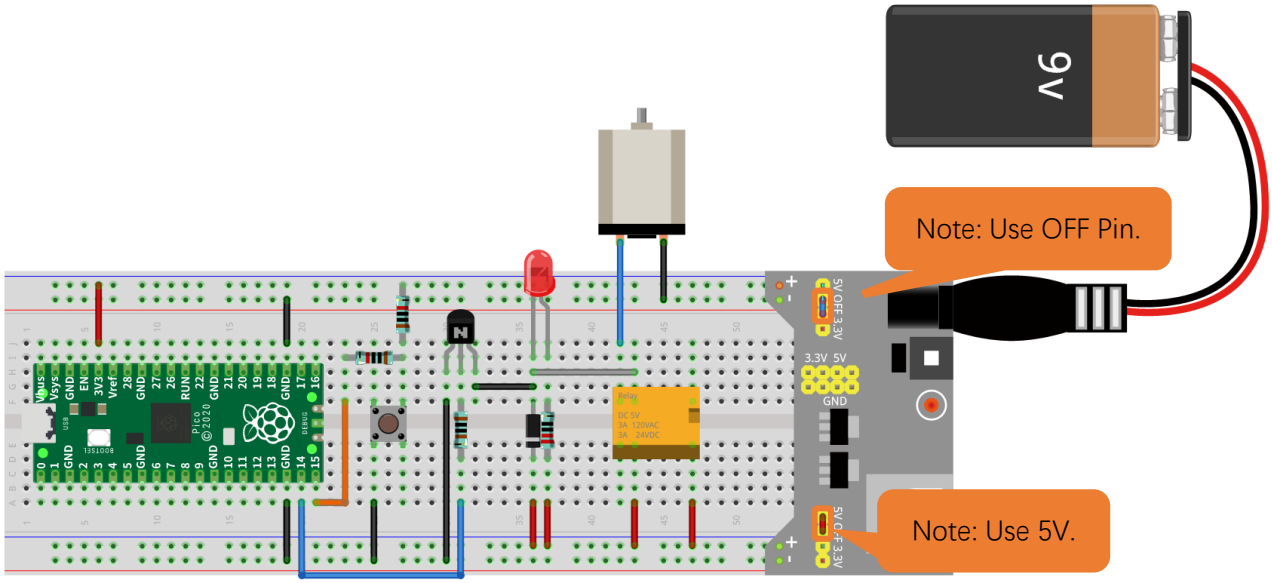
You can either select only one side of the output power supply, or choose to output 5V or 3.3V power supply at the same time on both sides, because the output circuits on both sides are separated and controlled independently and will not cause interference. Note that the GND should be connected to any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

Circuit

Schematic diagram



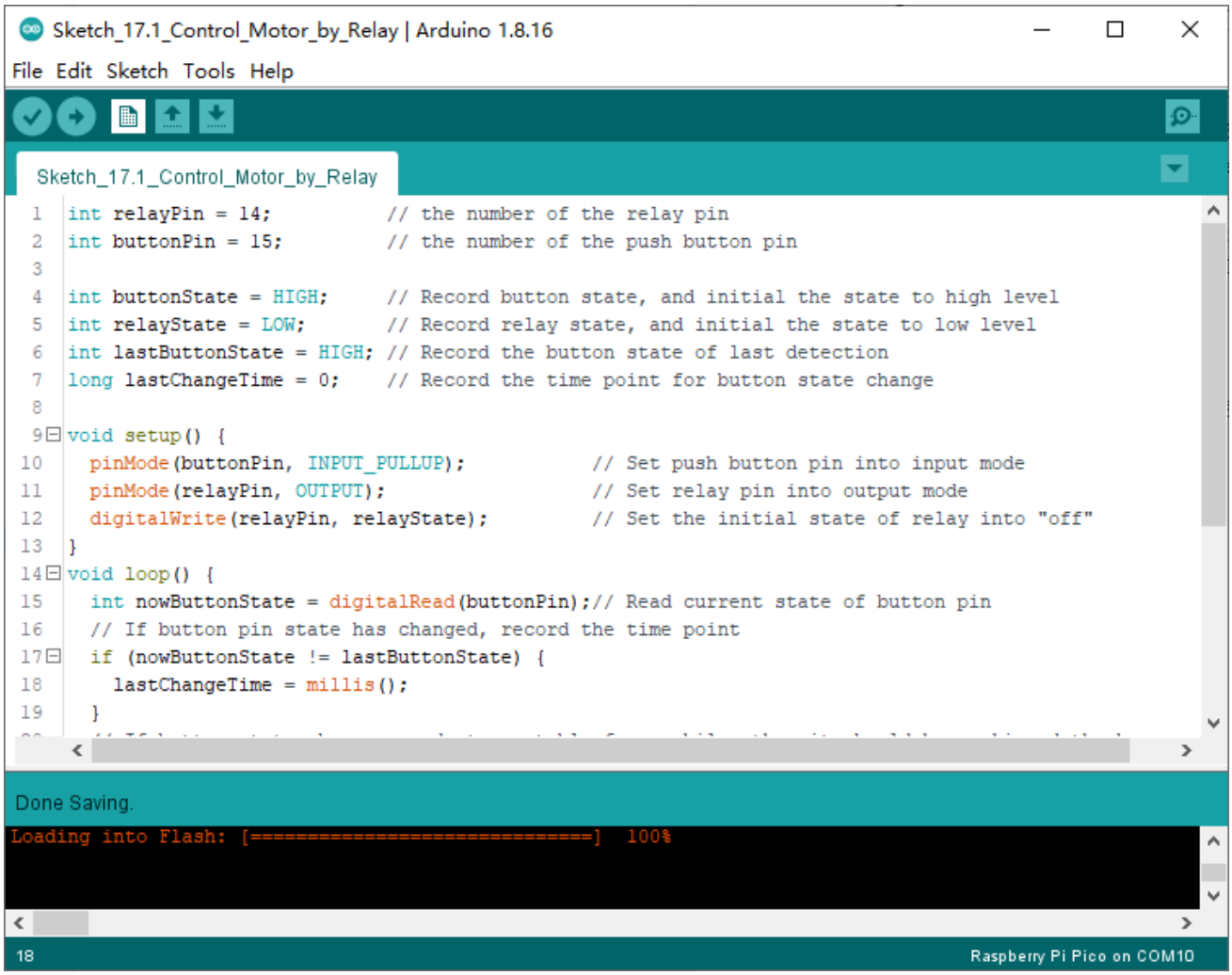
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Use buttons to control the relays and motors.

Sketch_17.1_Control_Motor_by_Relay



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

The following is the program code:

```
int relayPin = 14;          // the number of the relay pin
int buttonPin = 15;         // the number of the push button pin

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

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

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

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

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

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

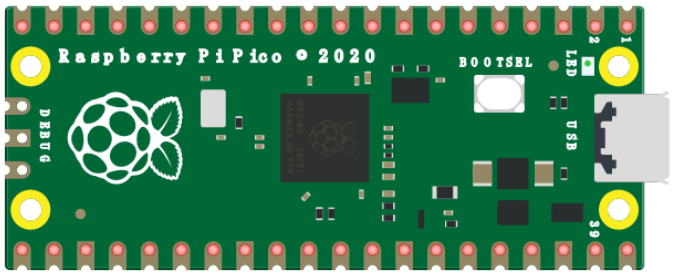
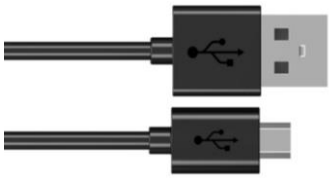
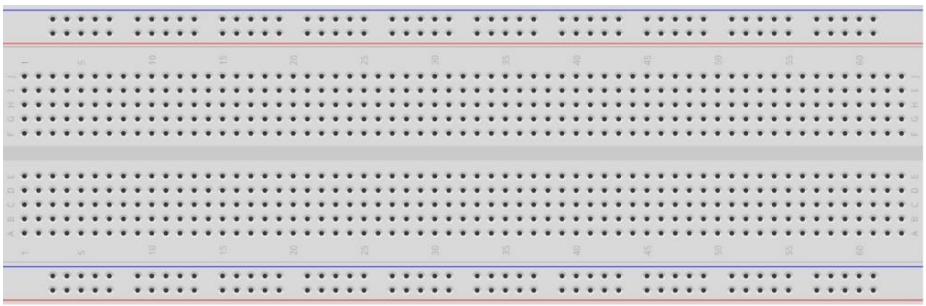
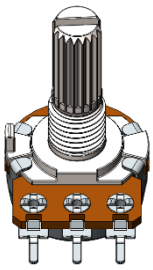
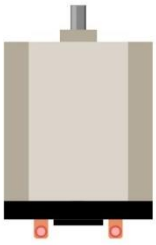
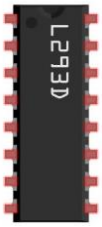
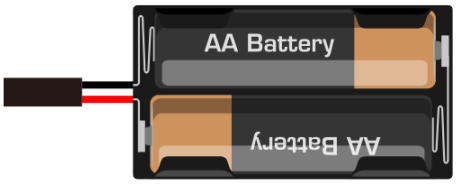
```
20   // If button state changes, and stays stable for a while, then it should have skipped the
     bounce area
21   if (millis() - lastChangeTime > 10) {
22     if (buttonState ! = nowButtonState) {    // Confirm button state has changed
23       buttonState = nowButtonState;
24       if (buttonState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

Chapter 18 L293D & Motor

Project 18.1 Control Motor with Potentiometer

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

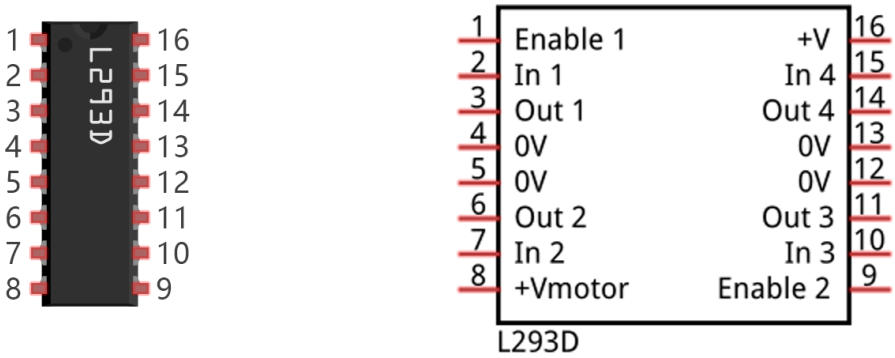
Component List

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

Component Knowledge

L293D

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

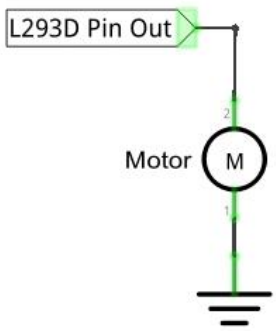


Port description of L293D module is as follows:

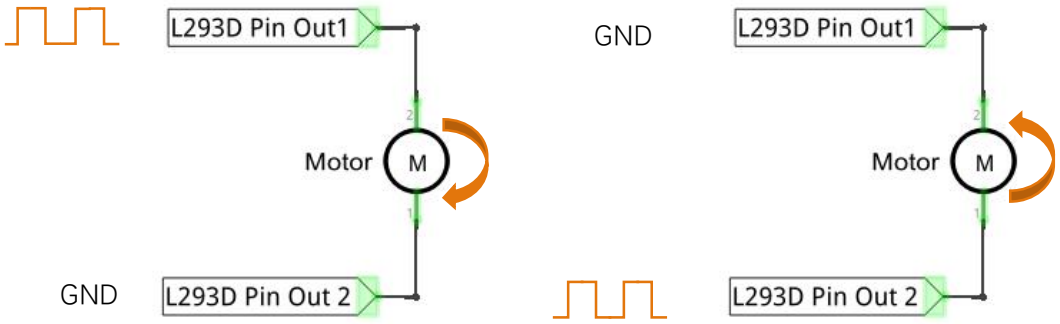
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4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

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

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



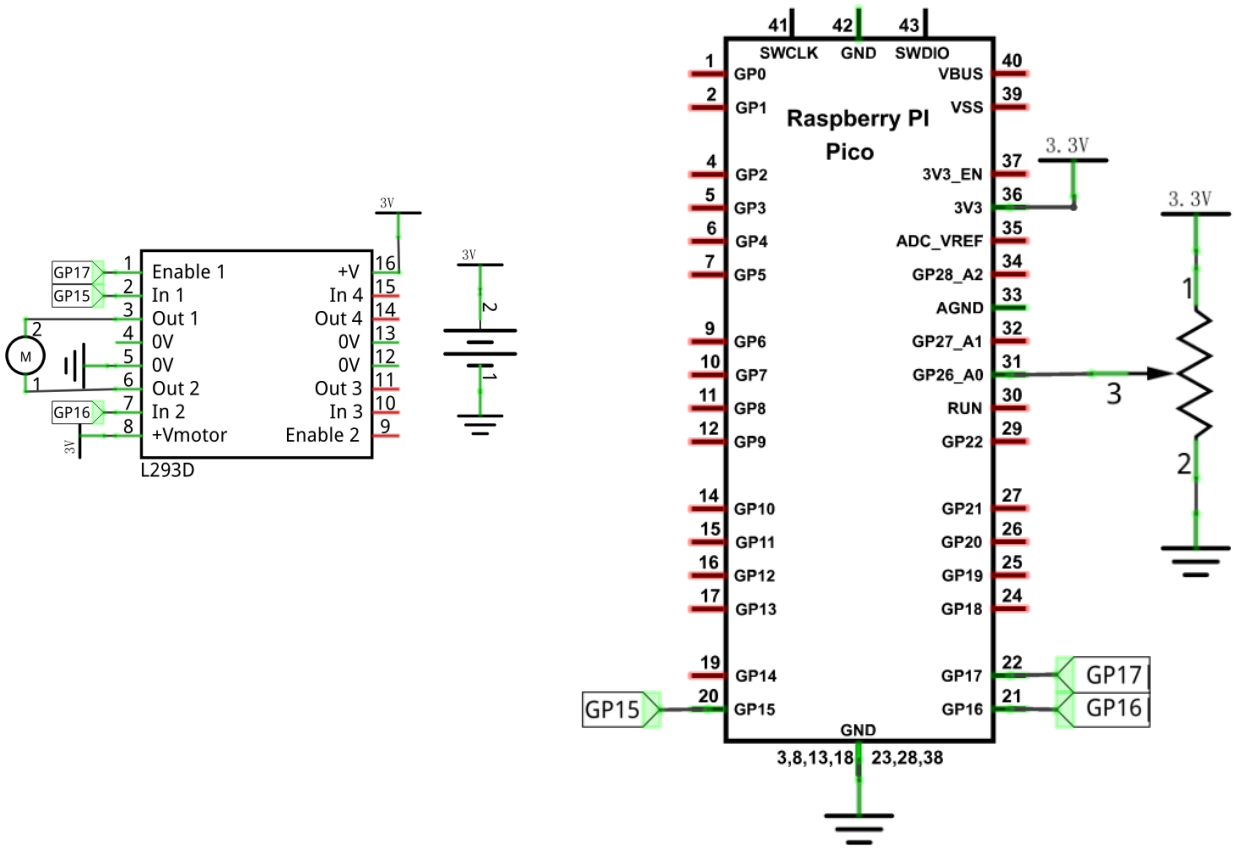
The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the direction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



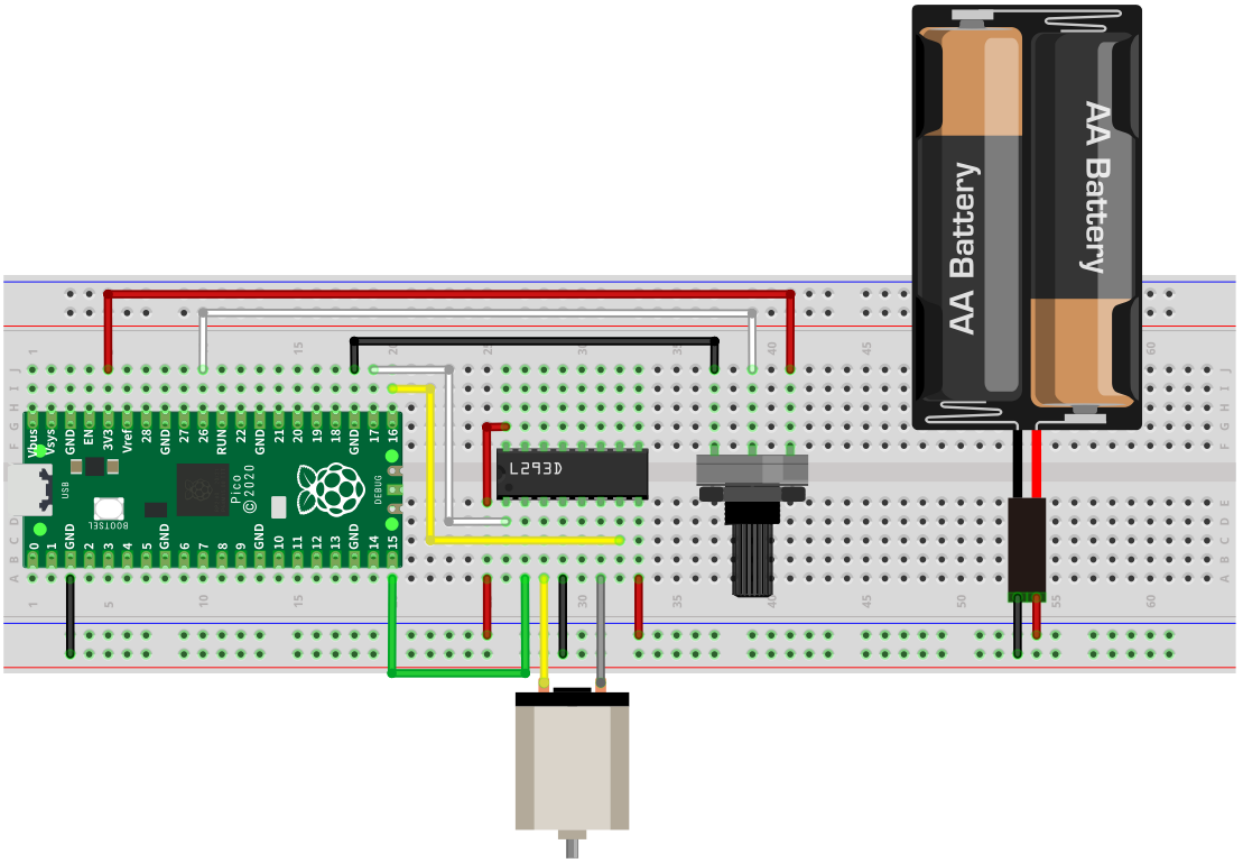
In practical use the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

Circuit

Schematic diagram



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

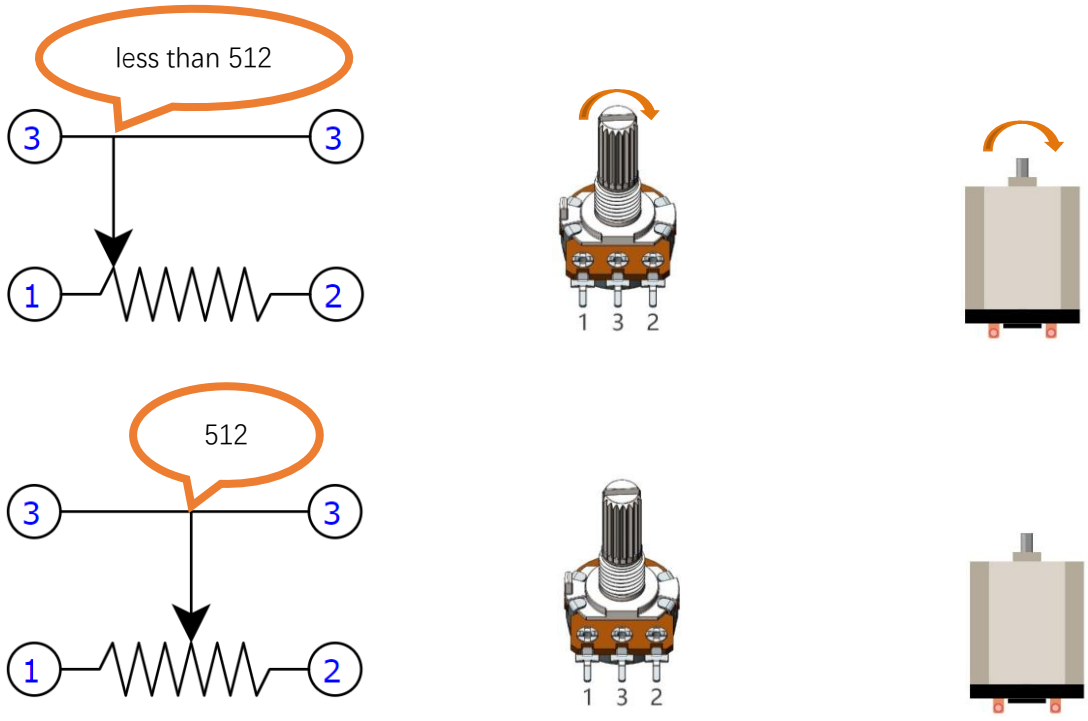


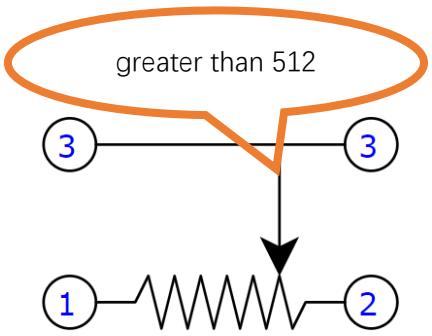
Sketch

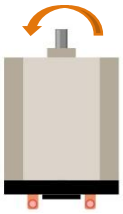
Sketch_18.1_Control_Motor_by_L293D

```
Sketch_18.1_Control_Motor_by_L293D | Arduino 1.8.18
File Edit Sketch Tools Help
Sketch_18.1_Control_Motor_by_L293D
15 void setup() {
16   // Initialize the pin into an output mode:
17   pinMode(in1Pin, OUTPUT);
18   pinMode(in2Pin, OUTPUT);
19   pinMode(enable1Pin, OUTPUT);
20 }
21
22 void loop() {
23   int potenVal = analogRead(ADC0_PIN);      // Convert the voltage of rotary potentiometer into digital
24   //Compare the number with value 512,
25   //if more than 512, clockwise rotates, otherwise, counter clockwise rotates
26   rotationSpeed = potenVal - 512;
27   if (potenVal > 512)
28     rotationDir = true;
29   else
30     rotationDir = false;
31   // Calculate the motor speed
32   rotationSpeed = abs(potenVal - 512);
33   //Control the steering and speed of the motor
34   driveMotor(rotationDir, constrain(rotationSpeed, 0, 512));
35 }
36
37 void driveMotor(boolean dir, int spd) {
Compiling sketch...
Compiling sketch...
"C:\\Users\\DESKTOP-LIN\\AppData\\Local\\Arduino15\\packages\\arduino\\tools\\arm-none-eabi-gcc\\7-2017q4/bin/arm-none-eabi-
29
Raspberry Pi Pico on COM10
```

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







The following is the sketch:

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

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

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

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

void driveMotor(boolean dir, int spd) {
  // Control motor rotation direction
```

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

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

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

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

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

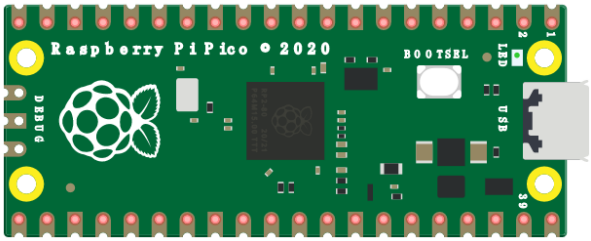
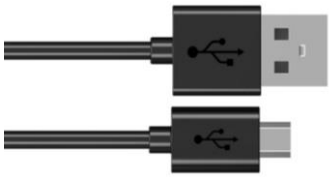
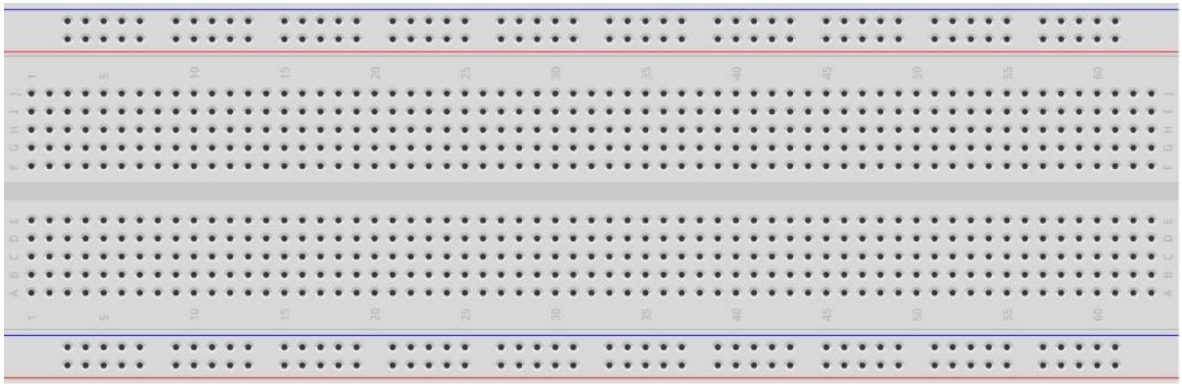
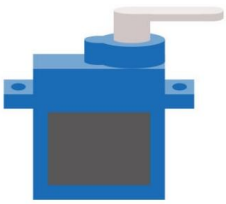
Chapter 19 Servo

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

Project 19.1 Servo Sweep

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

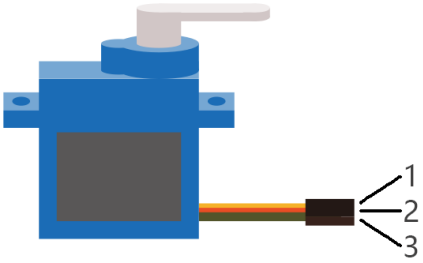
Component List

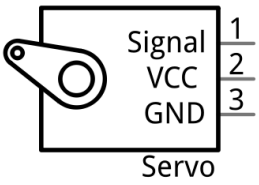
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Servo x1 	Jumper 

Component Knowledge

Servo

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





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

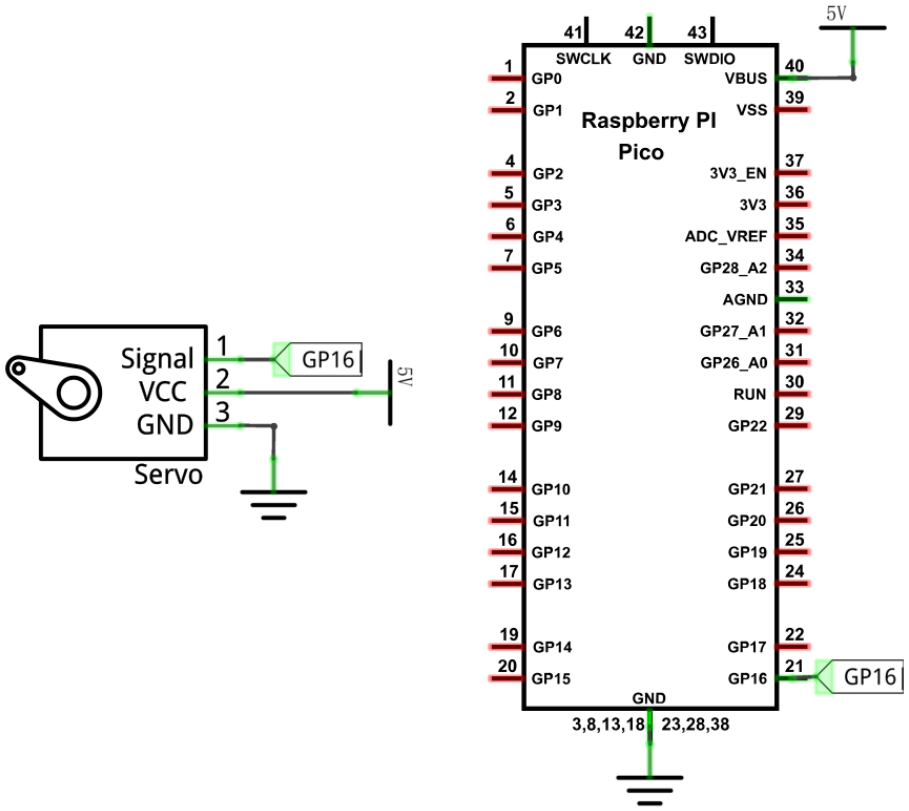
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

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

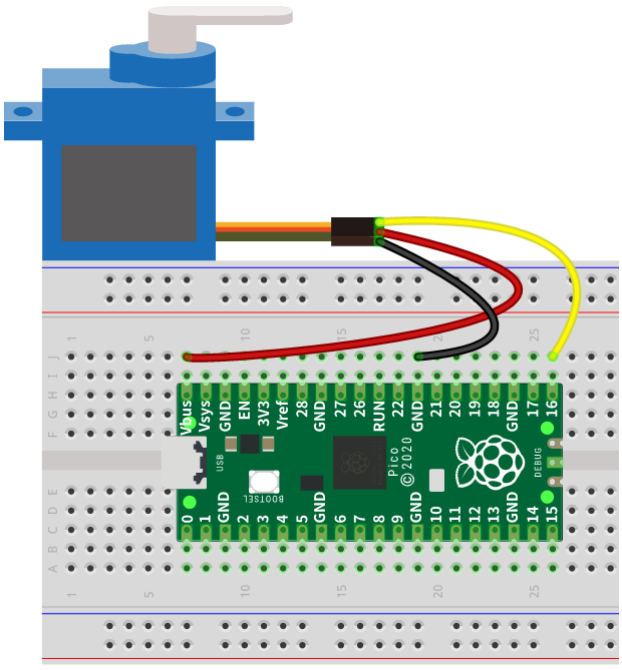
Circuit

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

Schematic diagram



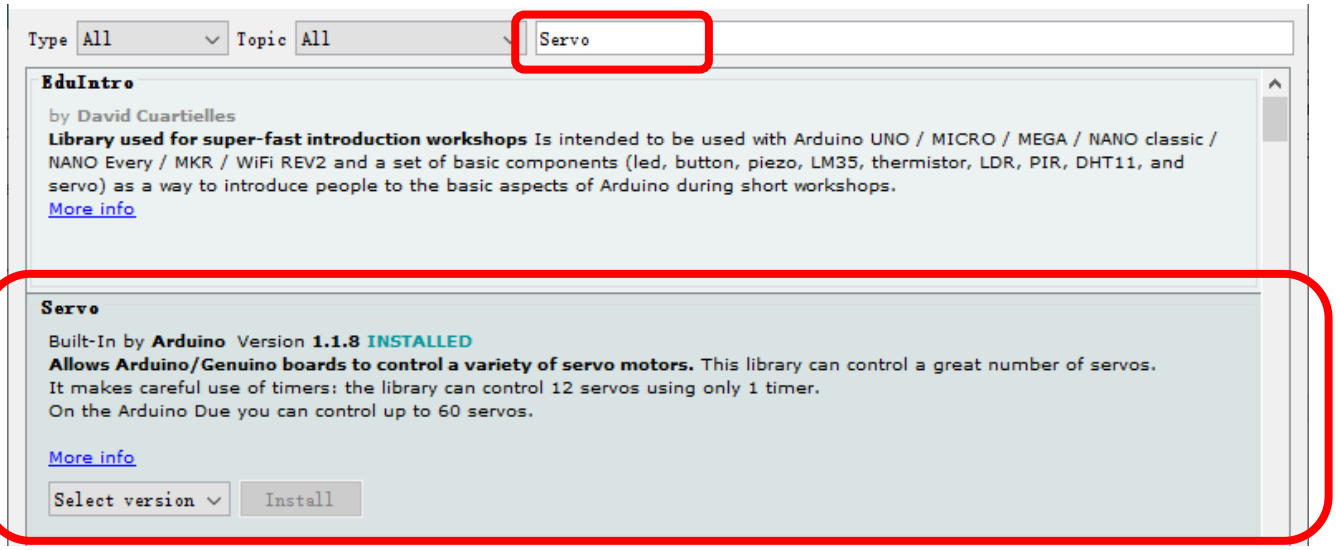
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

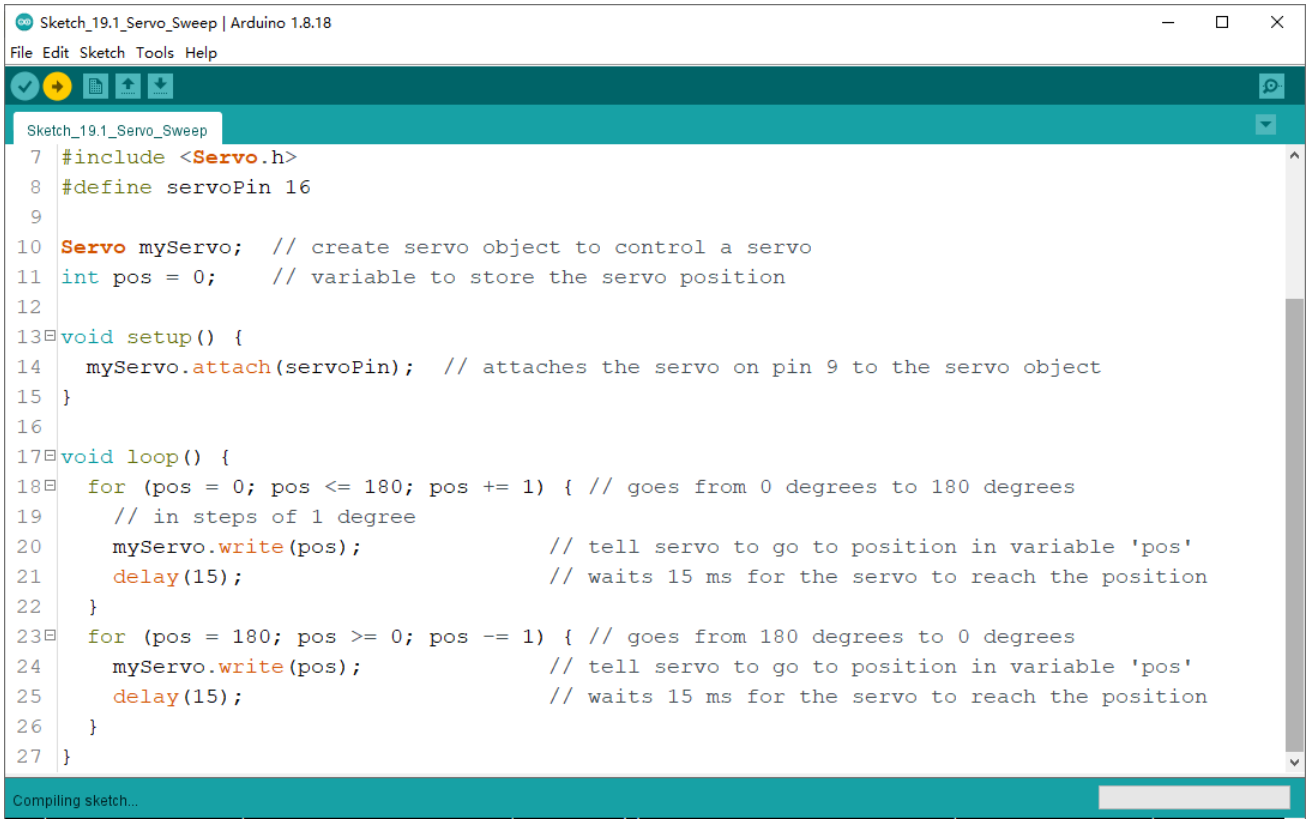
How to install the library

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

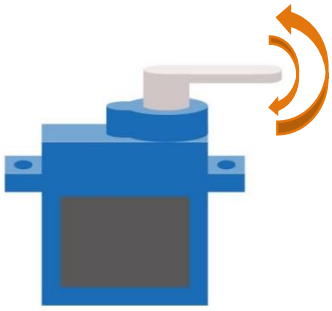


Use the Servo library to control the servo motor and let the servo motor rotate back and forth.

Sketch_19.1_Servo_Sweep



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



The following is the program code:

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

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

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

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

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

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

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

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

Set the control servo motor pin.

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

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

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

Reference

Servo Class

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

Servo myservo;

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

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

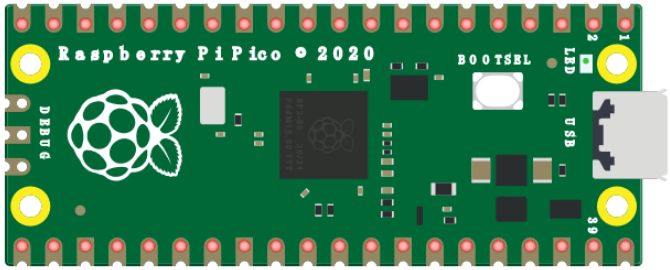
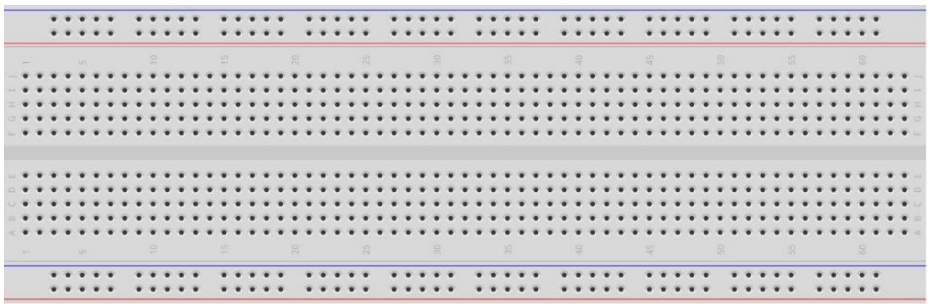
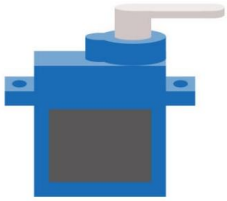
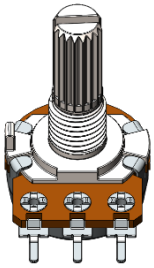
myservo.attach(pin): Initialize the servo, the parameter is the port connected to servo signal line;

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

Project 19.2 Servo Knob

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

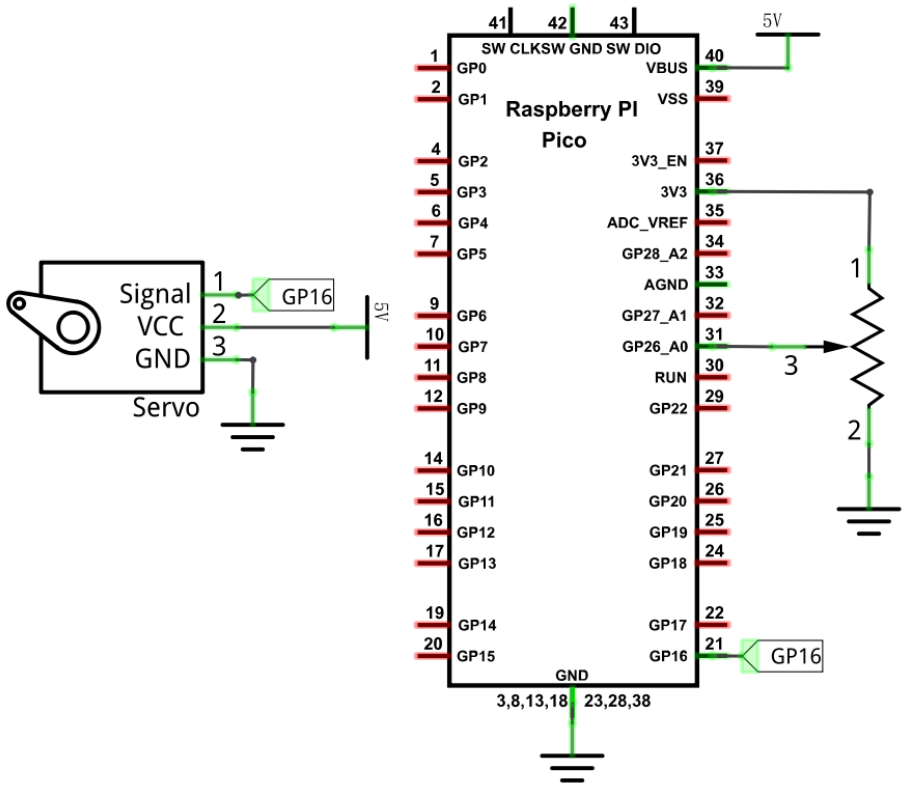
Component List

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

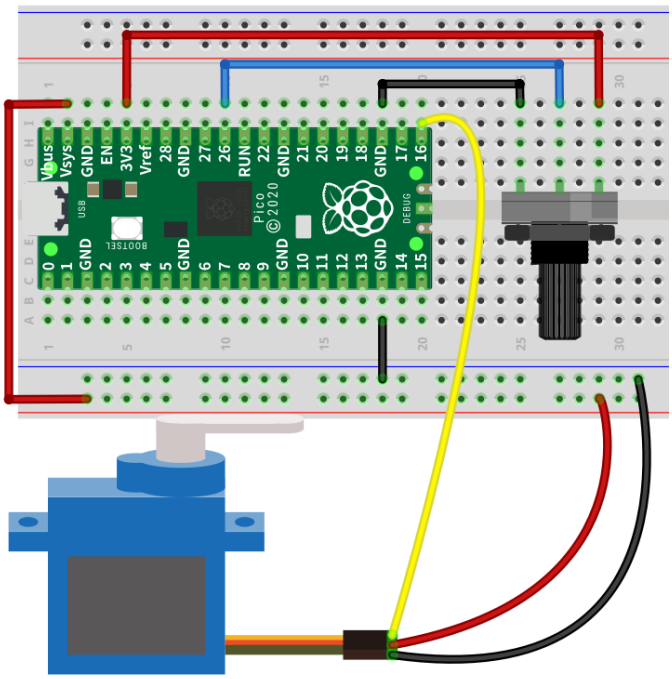
Circuit

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

Schematic diagram



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



Sketch

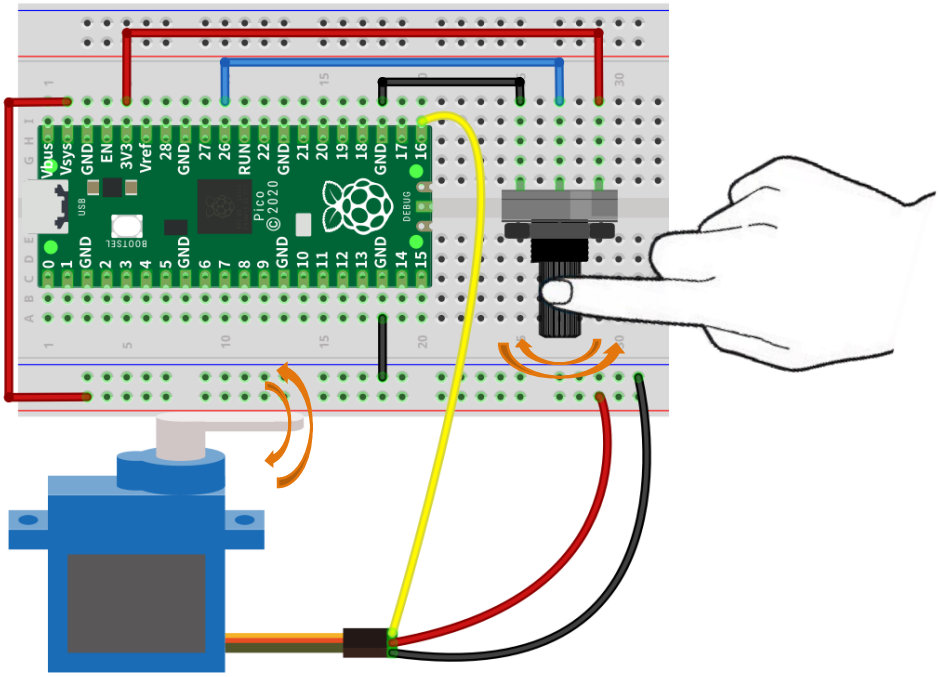
Sketch_19.2_Control_Servo_by_Potentiometer

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

```
#include <Servo.h>

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

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

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

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

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

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

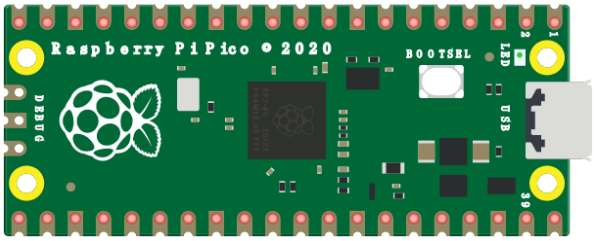
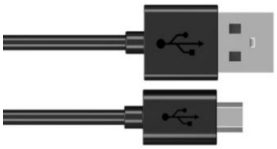
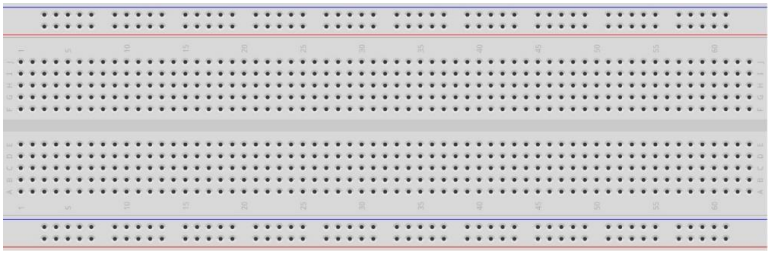
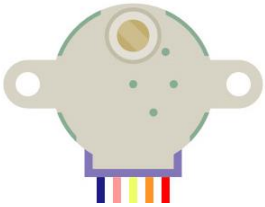
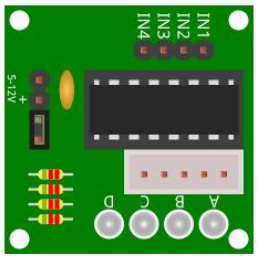
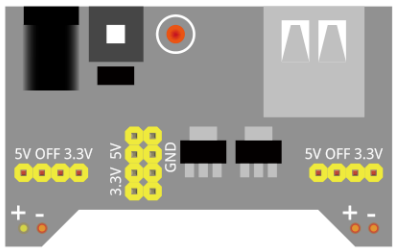
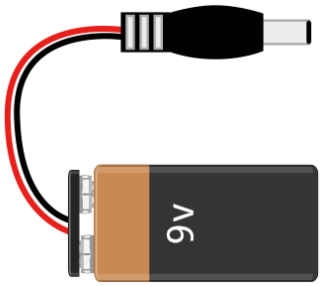


Chapter 20 Stepper Motor

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

Project 20.1 Stepper Motor

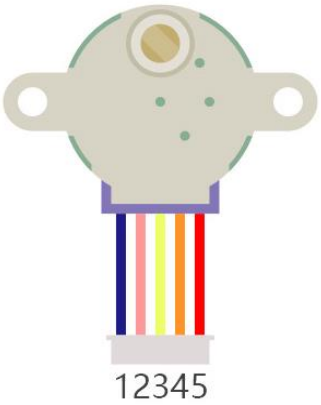
Component List

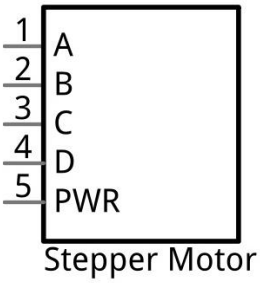
Raspberry Pi Pico x1 	USB cable x1 	
Breadboard x1 		
Stepper Motor x1 	ULN2003 Stepper motorDriver x1 	Jumper 
BreadBoardPower x1 	9V battery (prepared by yourself) & battery line x1 	

Component Knowledge

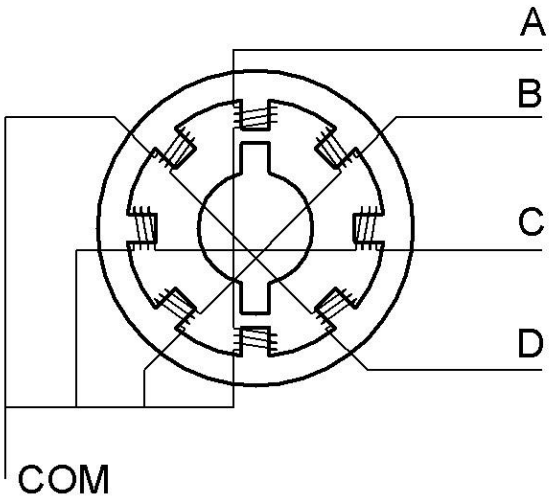
Stepper Motor x1

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



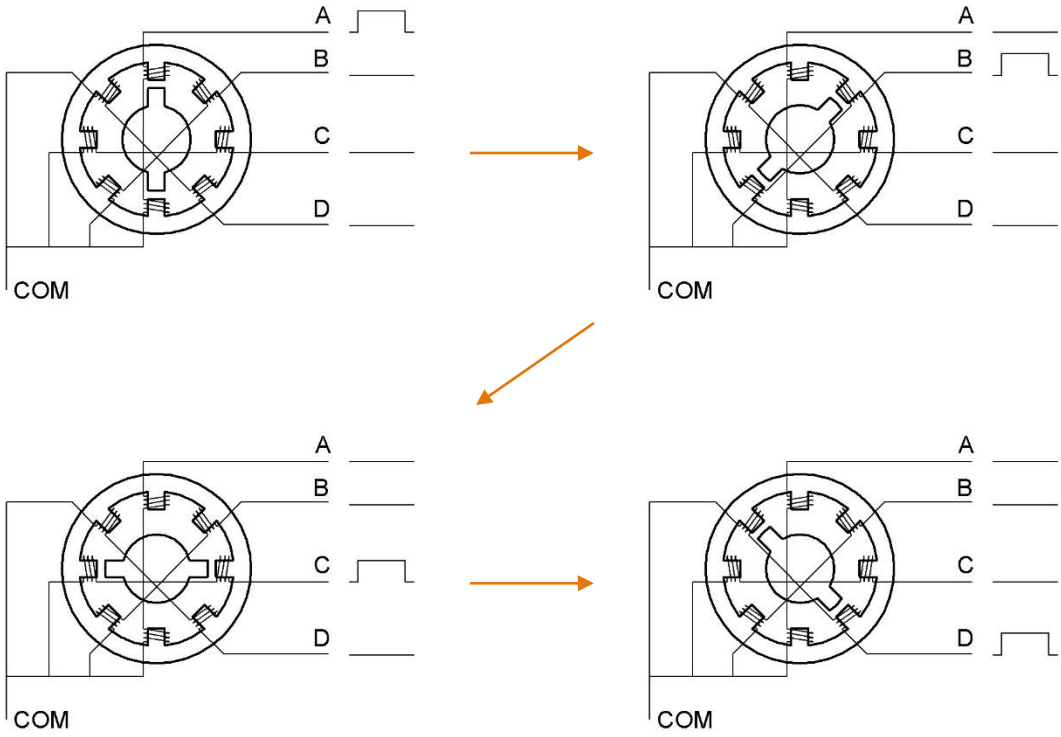


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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

A common driving process is as follows:



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

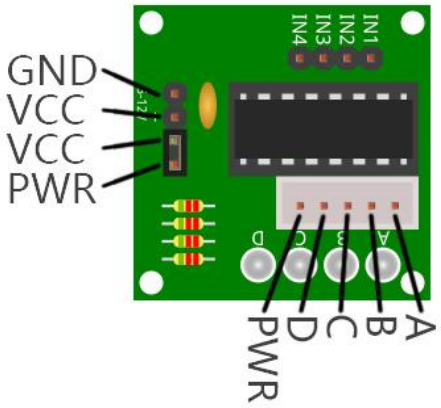
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. The sequence of powering the coils looks like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

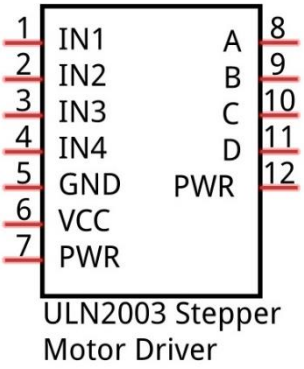
The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft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

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

ULN2003 Stepper motor driver

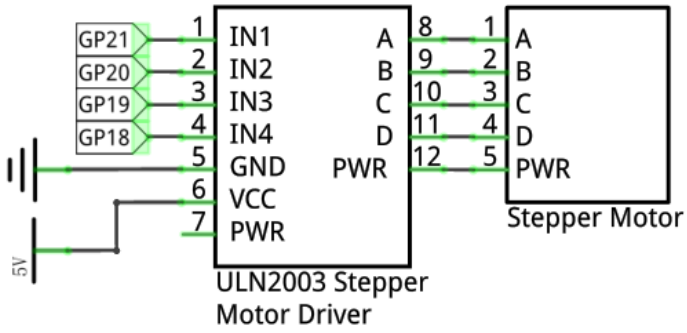
A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals.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

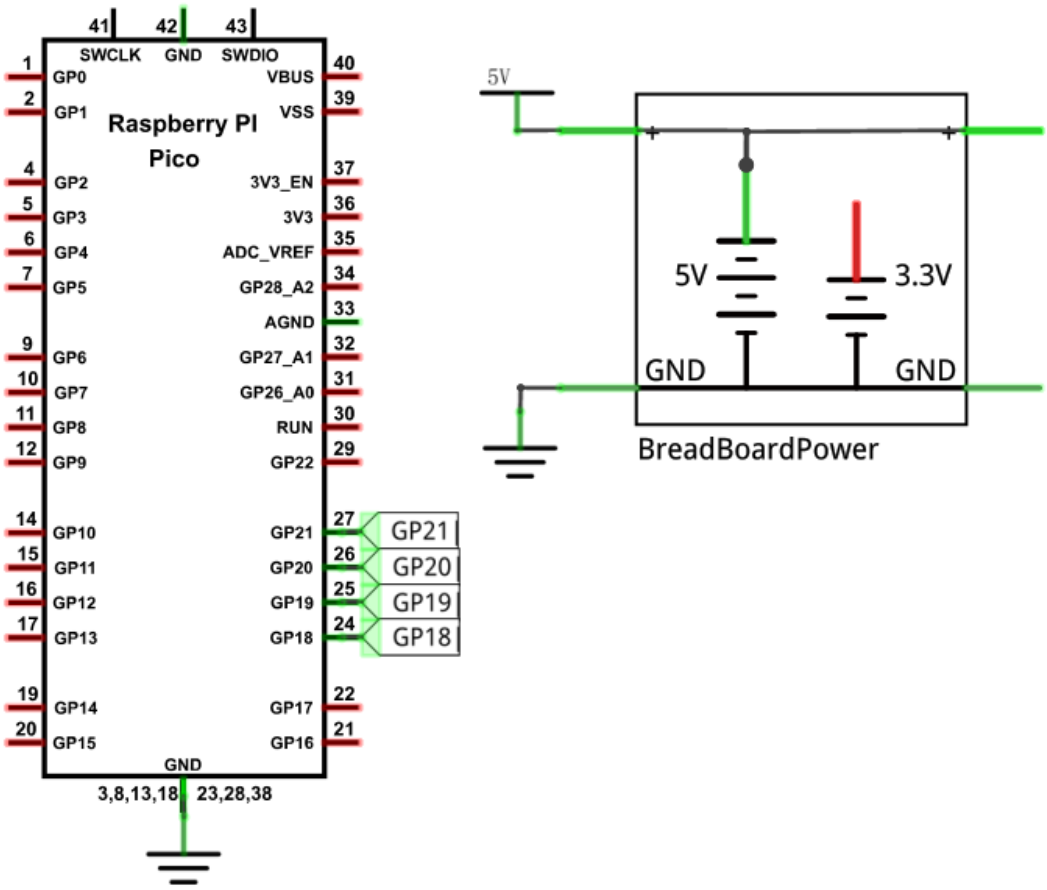




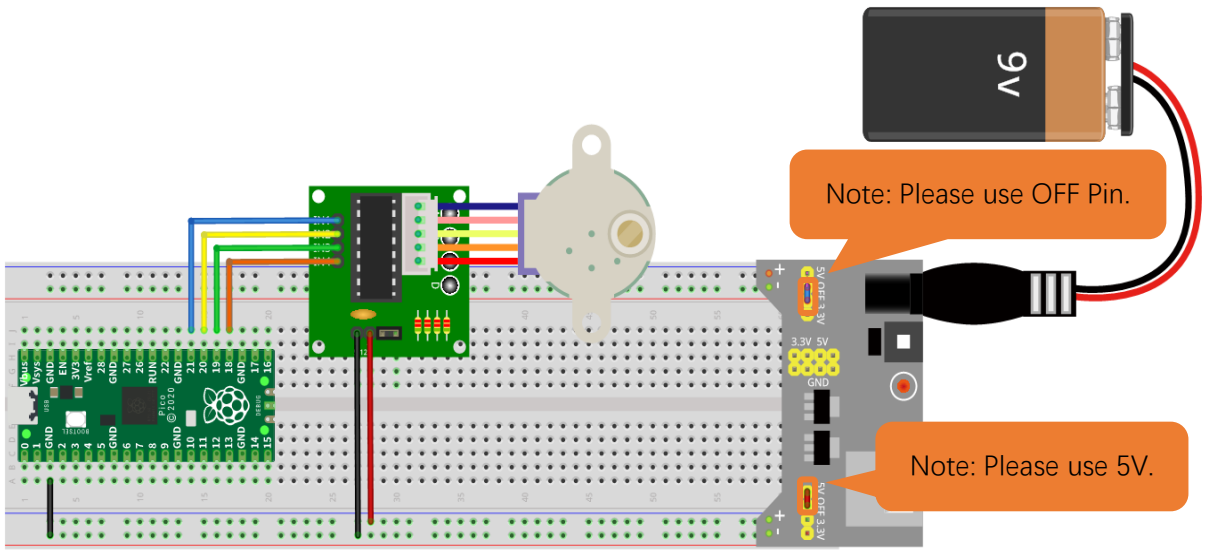
Circuit

Schematic diagram





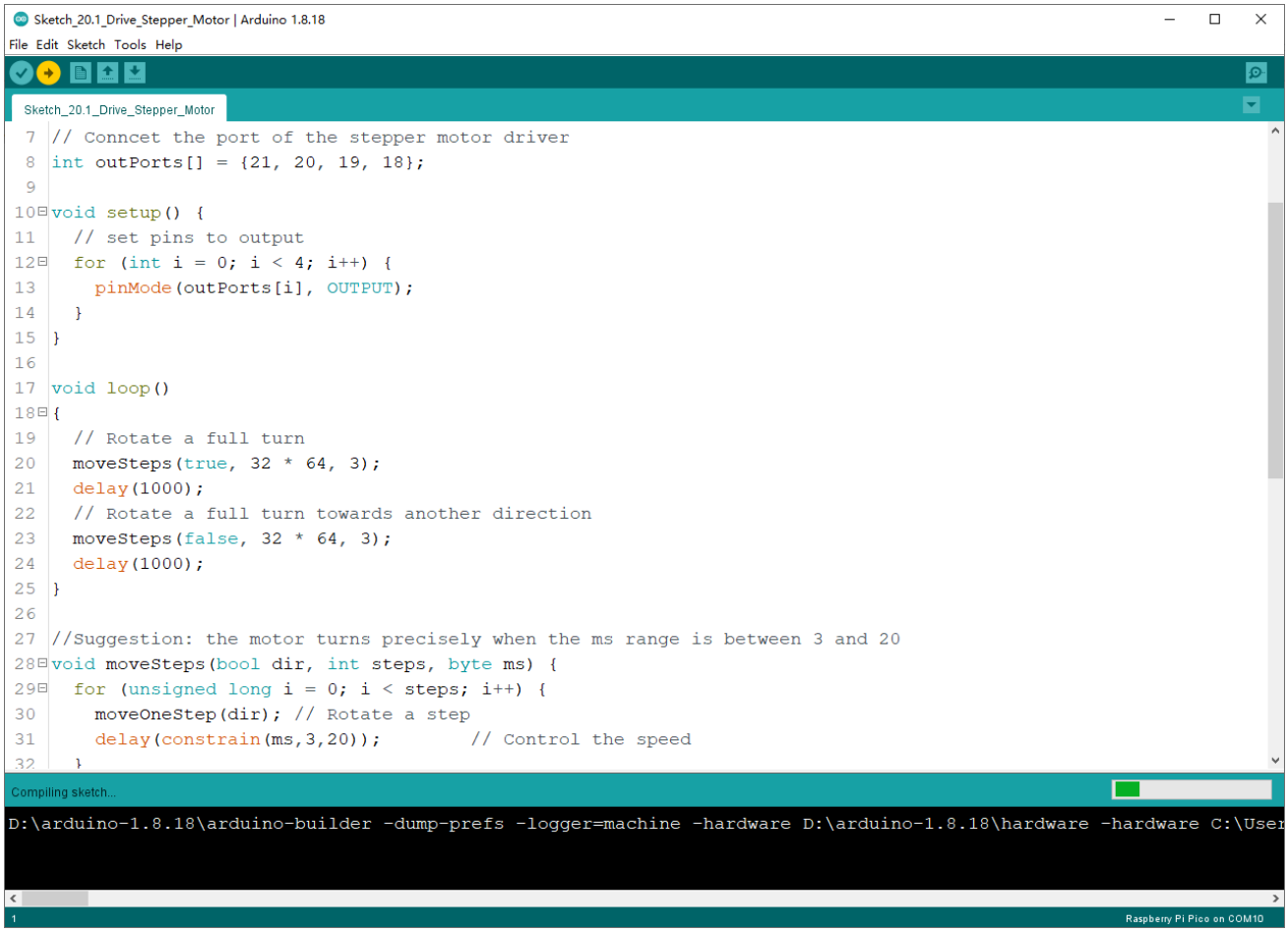
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



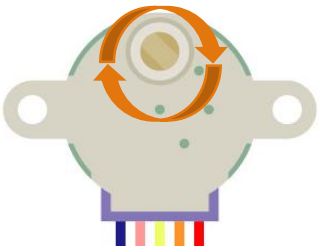
Sketch

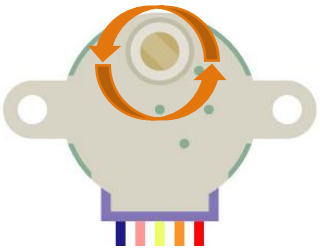
This code uses the four-step, four-part mode to drive the stepper motor in the clockwise and anticlockwise directions.

Sketch_20.1_Drive_Stepper_Motor



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





The following is the program code:

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

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

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

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

void moveOneStep(bool dir) {
  // Define a variable, use four low bit to indicate the state of port
  static byte out = 0x01;
  // Decide the shift direction according to the rotation direction
  if (dir) {  // ring shift left
    out ! = 0x08 ?  out = out << 1 : out = 0x01;
  }
  else {      // ring shift right
    out ! = 0x01 ?  out = out >> 1 : out = 0x08;
  }
  // Output singal to each port
  for (int i = 0; i < 4; i++) {
    digitalWrite(outPorts[i], (out & (0x01 << i)) ?  HIGH : LOW);
  }
}

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

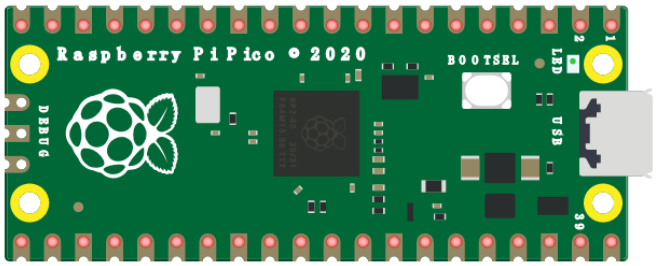
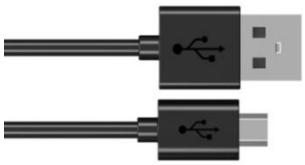
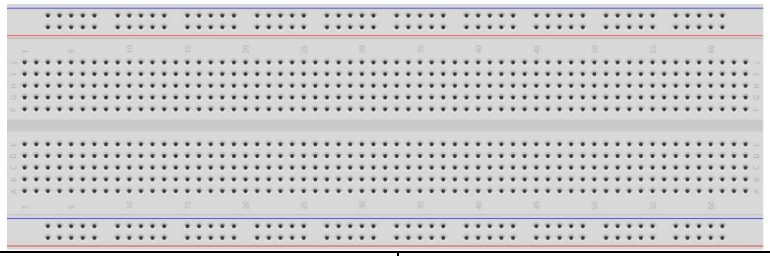
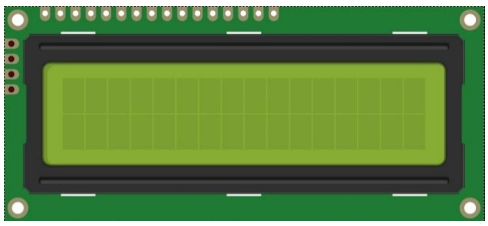
Chapter 21 LCD1602

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

Project 21.1 LCD1602

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

Component List

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

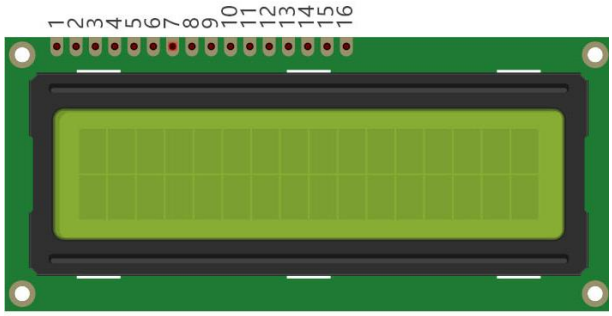
Component Knowledge

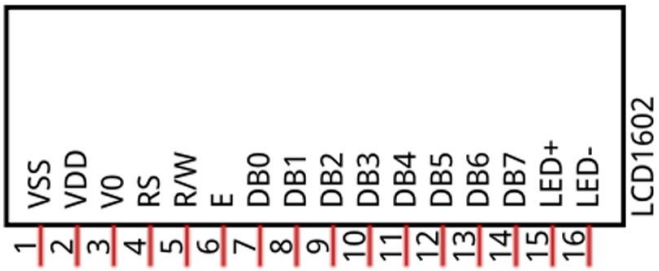
I2C communication

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

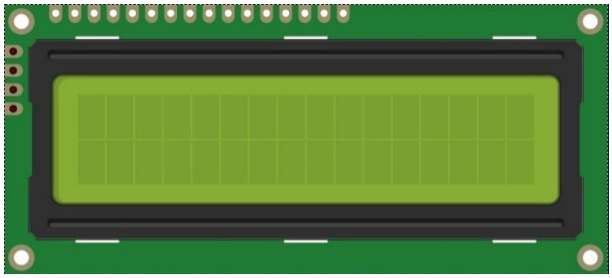
LCD1602 communication'

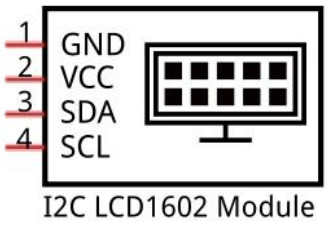
The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram.





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

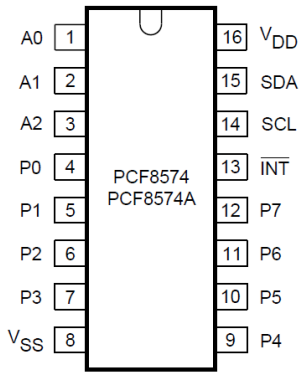




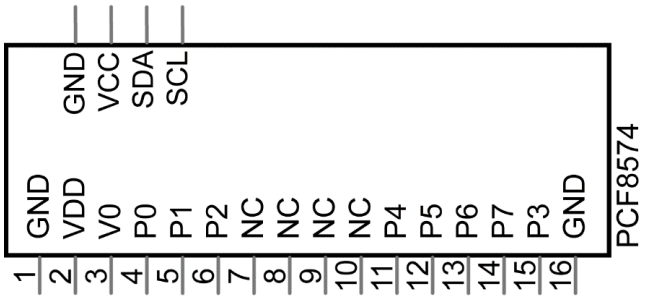
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

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

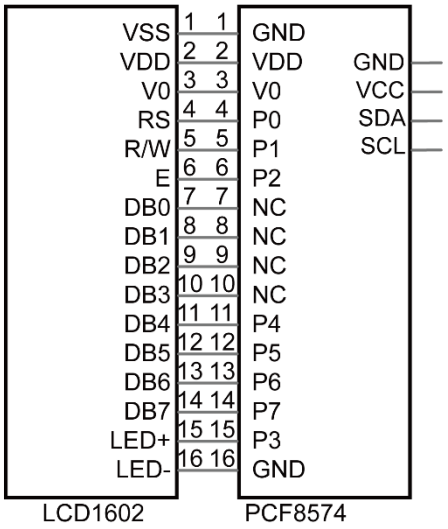
PCF8574 chip pin diagram:



PCF8574 module pin diagram



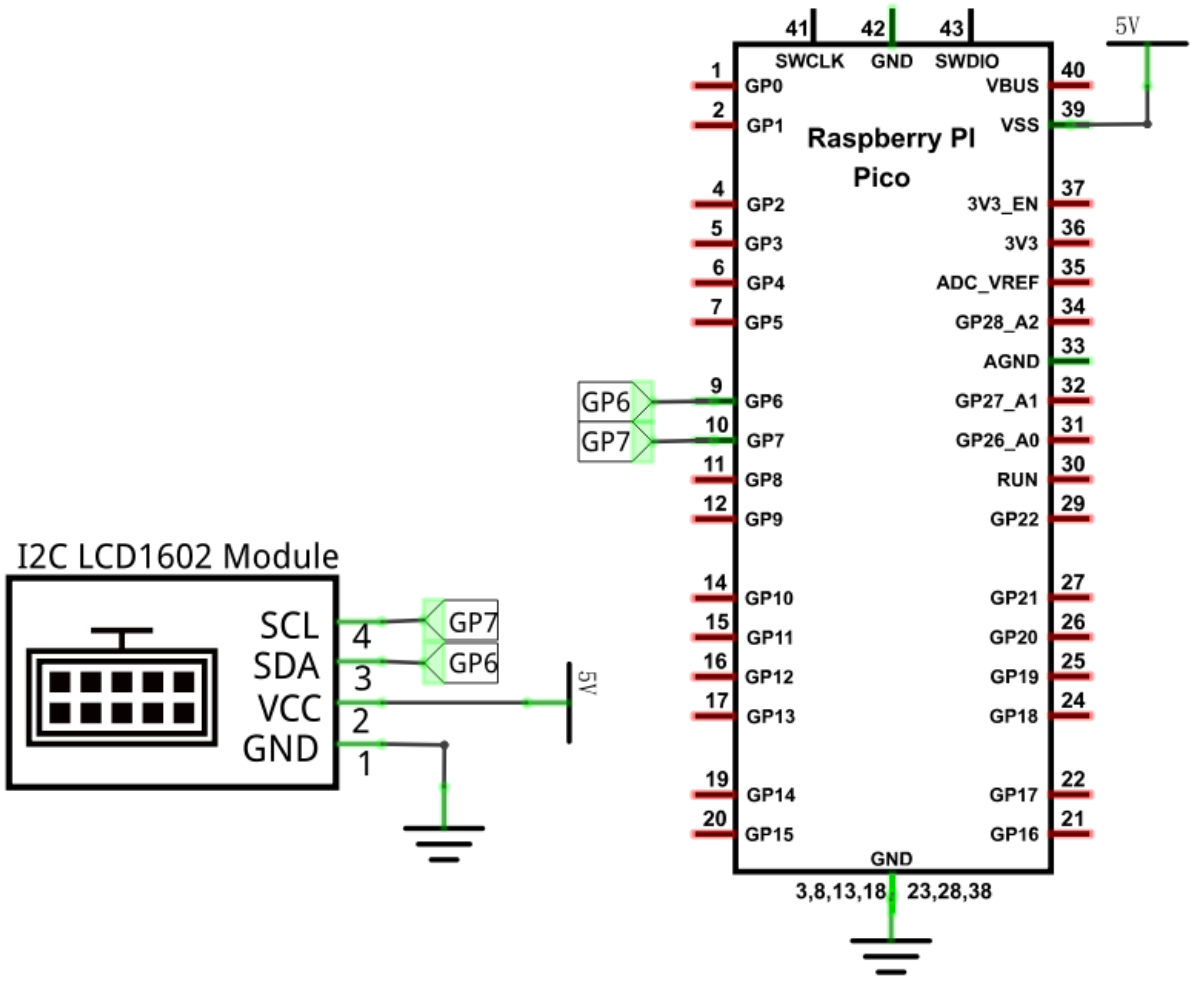
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



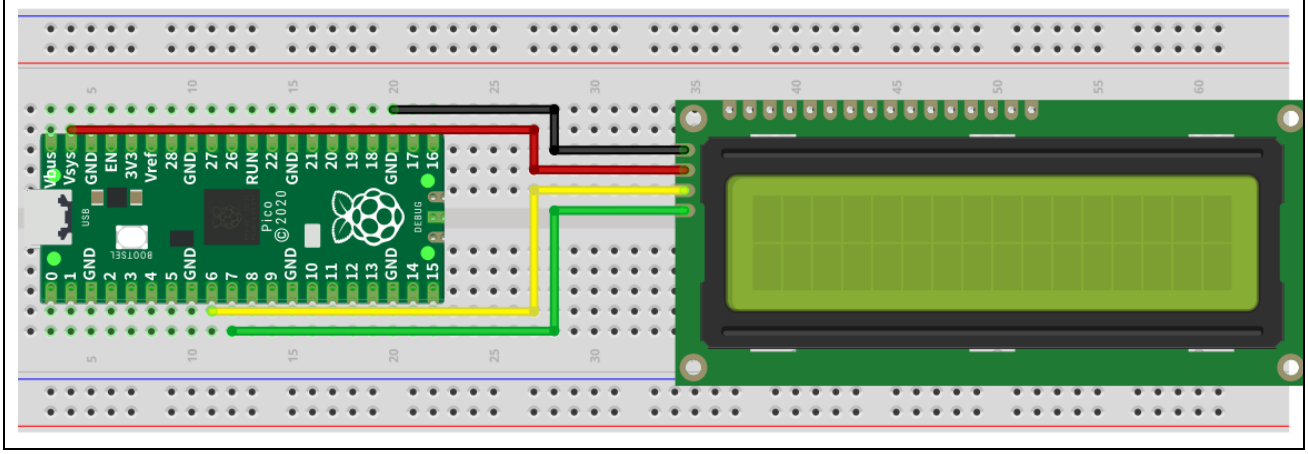
So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface. In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



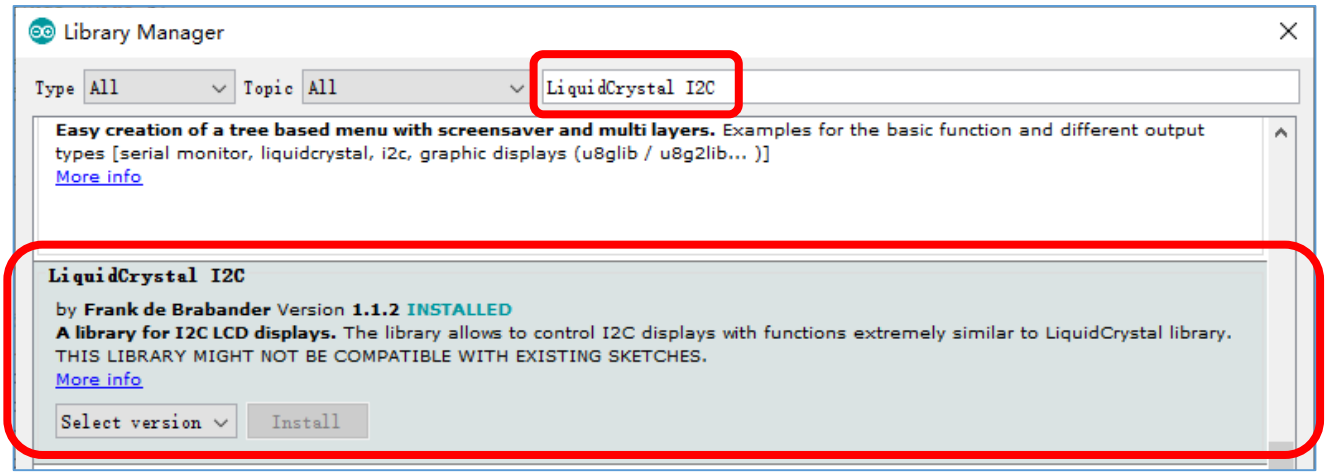
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

How to install the library

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



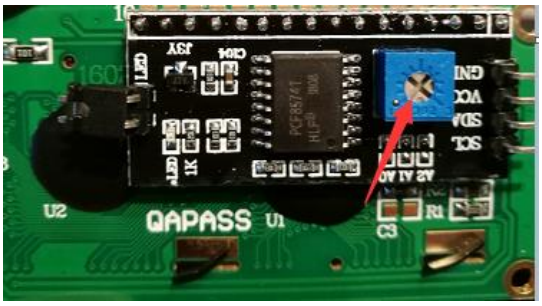
Use I2C LCD 1602 to display characters and variables.

Sketch_21.1_Display_the_string_on_LCD1602


```
#include <LiquidCrystal_I2C.h>

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

void setup() {
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  lcd.setCursor(0,0);             // Move the cursor to row 0, column 0
  lcd.print("hello world");       // The print content is displayed on the LCD
}

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

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

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



The following is the program code:

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

void setup() {
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
  lcd.setCursor(0,0);             // Move the cursor to row 0, column 0
  lcd.print("hello world");       // The print content is displayed on the LCD
}

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

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

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

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

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

Initialize LCD1602 and turn on the backlight of LCD.

```
10    lcd.init();                    // LCD driver initialization
11    lcd.backlight();               // Turn on the backlight
```

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

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

Print the number on the second line of LCD1602.

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

Reference

class LiquidCrystal

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

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

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

```
init();
```

Initializes the Lcd1602's device

```
backlight();
```

Turn on Lcd1602's backlight.

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

Sets the screen's column and row.

column: The range is 0 to 15.

row: The range is 0 to 1.

```
print(String);
```

Print the character string on Lcd1602

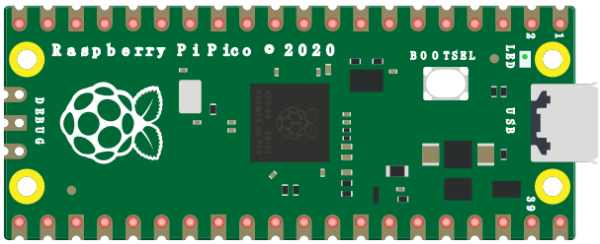
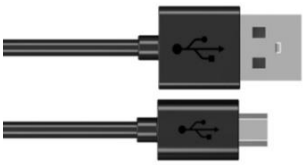
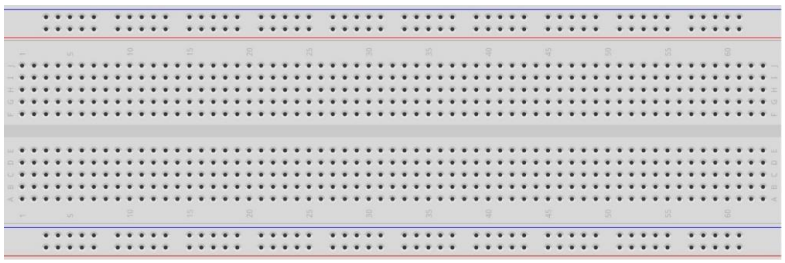
Chapter 22 Ultrasonic Ranging

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

Project 22.1 Ultrasonic Ranging

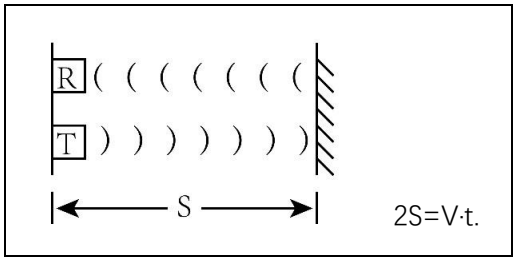
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

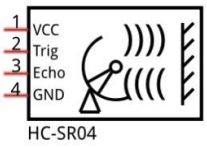
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	HC SR04 x1 

Component Knowledge

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



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



Pin description:

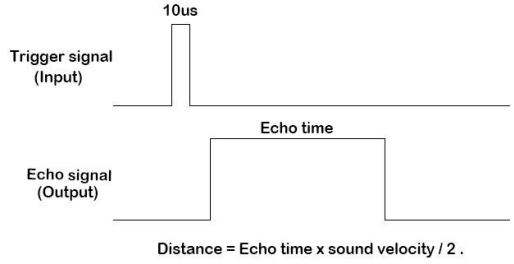
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

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

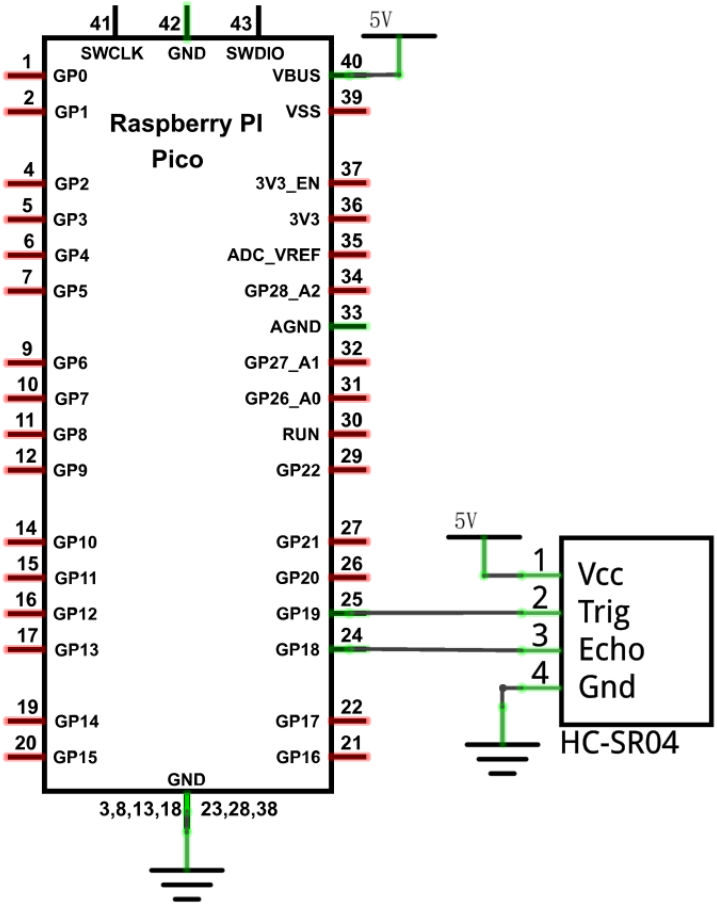
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The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



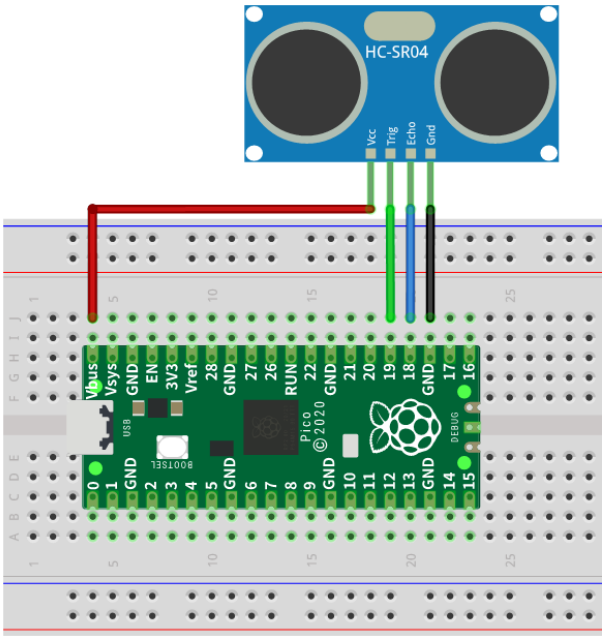
Circuit

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

Schematic diagram



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



Sketch

Sketch_22.1_Ultrasonic_Ranging

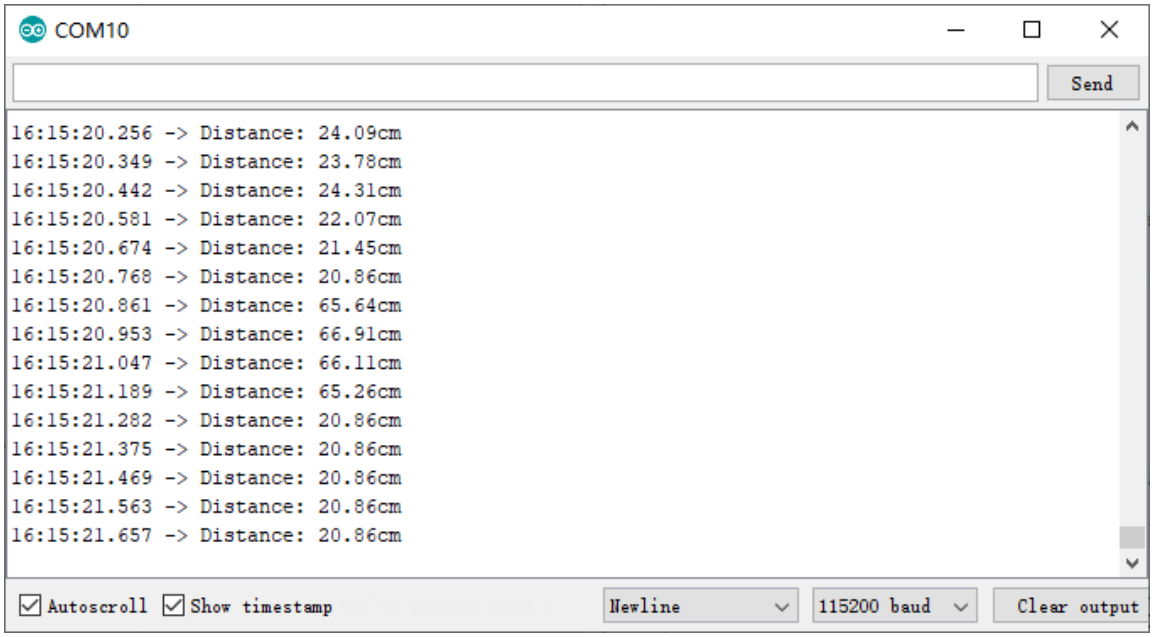
Sketch_22.1_Ultrasonic_Ranging | Arduino 1.8.18

File Edit Sketch Tools Help

Sketch_22.1_Ultrasonic_Ranging

```
#define trigPin 19 // define TrigPin
#define echoPin 18 // define EchoPin.
#define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s

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

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

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

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



The following is the program code:

```
#define trigPin 19 // define trigPin
#define echoPin 18 // define echoPin.
#define MAX_DISTANCE 700 // Maximum sensor distance is rated at 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s

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

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

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

```
28    // Wait HC-SR04 returning to the high level and measure out this waiting time
29    pingTime = pulseIn(echoPin, HIGH, timeOut);
30    // calculate the distance according to the time
31    distance = (float)pingTime * soundVelocity / 2 / 10000;
32    return distance; // return the distance value
33  }
```

First, define the pins and the maximum measurement distance.

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

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

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

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

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

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

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

About function **pulseIn**():

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

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

Project 22.2 Ultrasonic Ranging

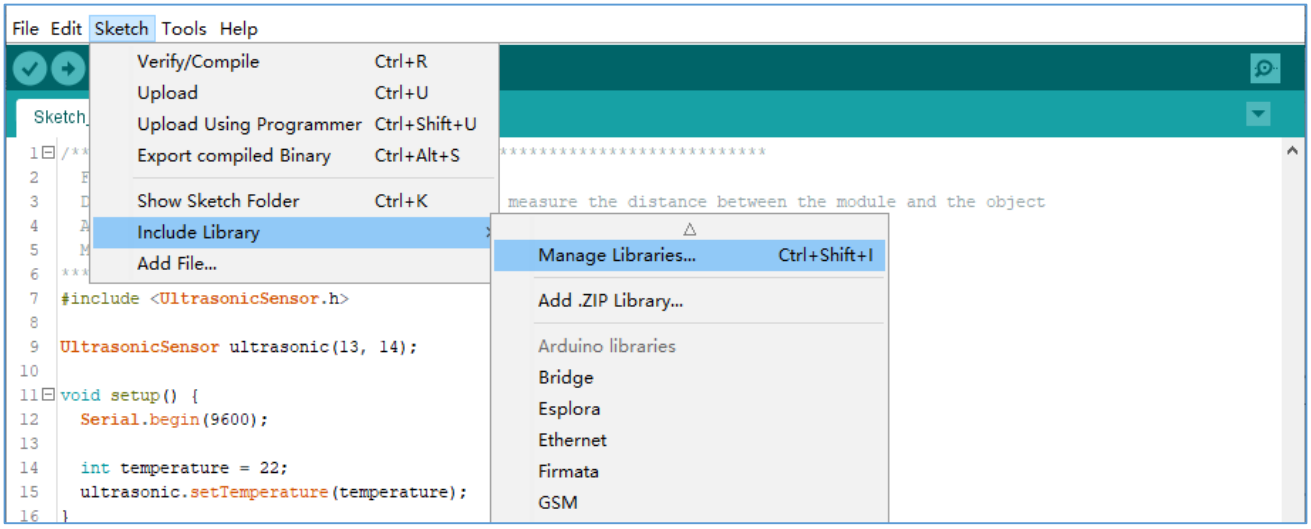
Component List and Circuit

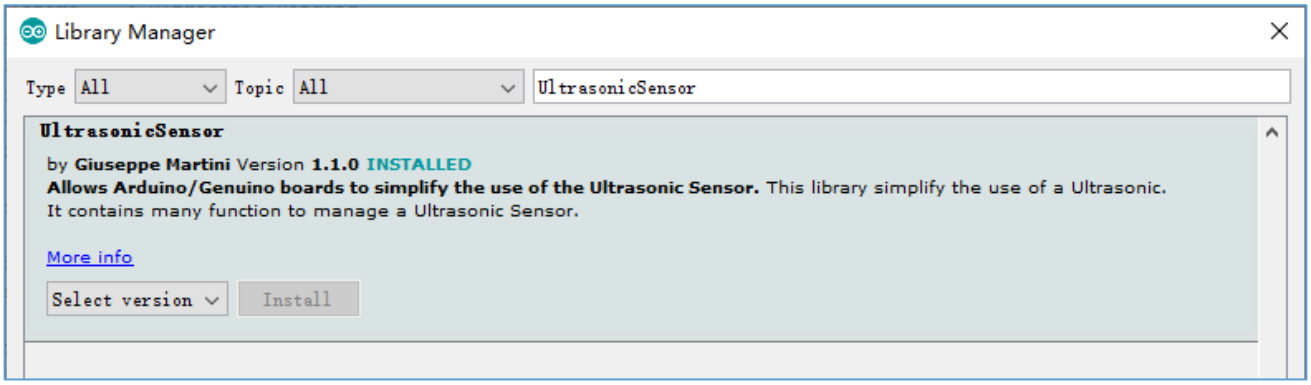
Component List and Circuit are the same as the previous section.

Sketch

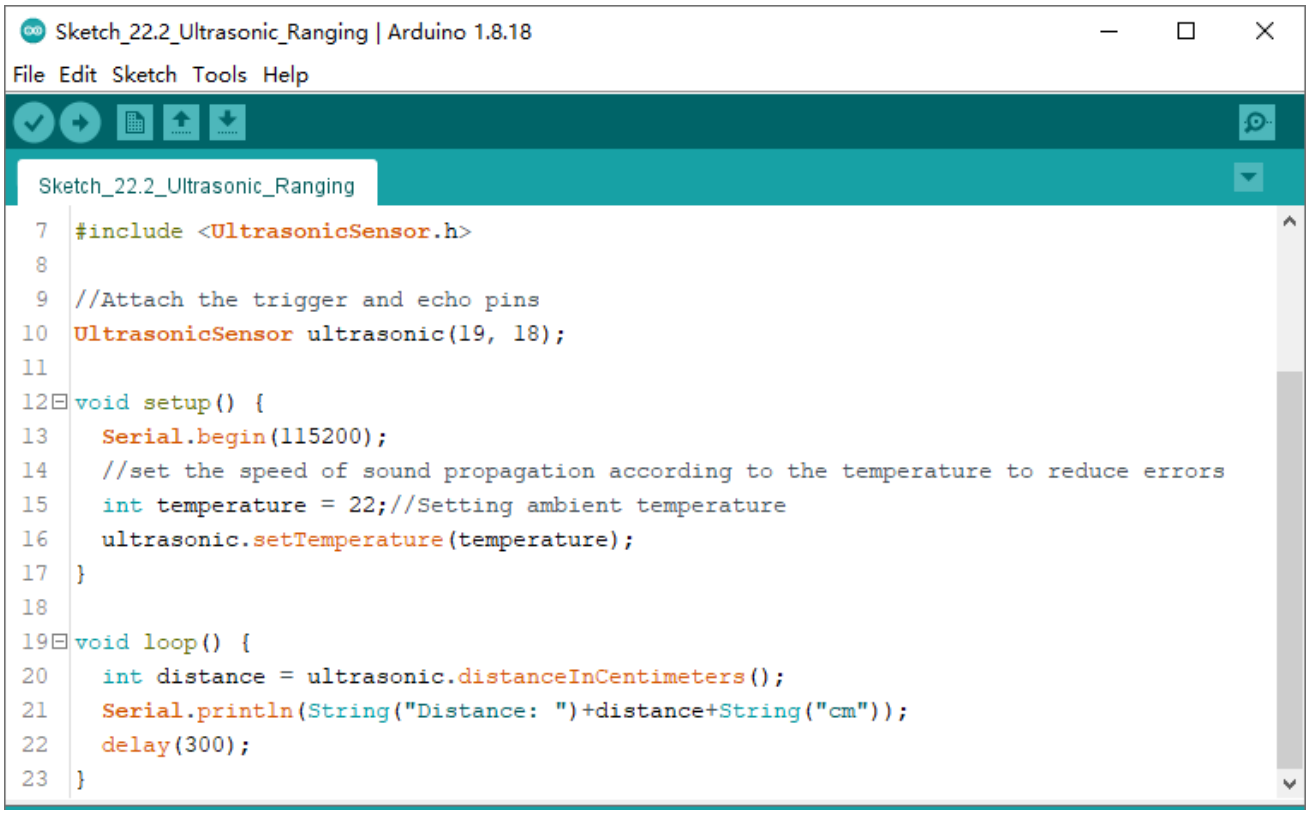
How to install the library

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

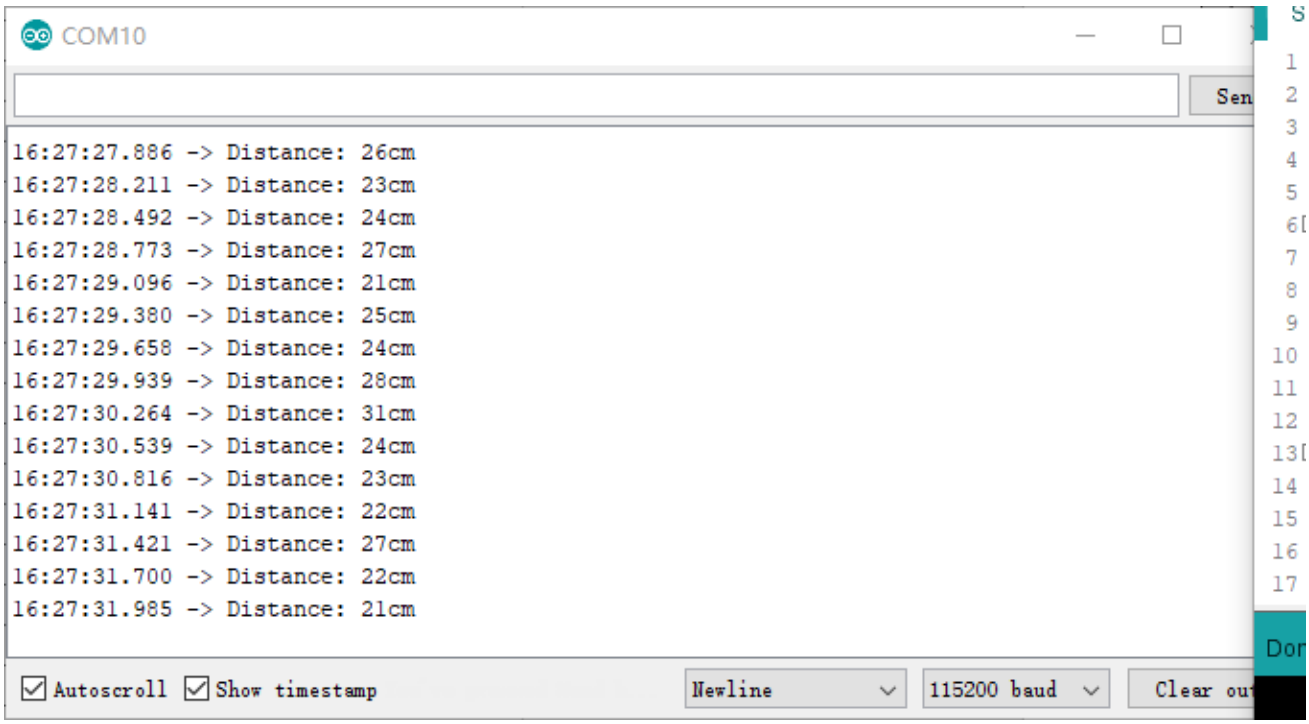




Sketch_22.2_Ultrasonic_Ranging



Upload the sketch to Pico, open the serial monitor and set the baud rate to 115200. Use the ultrasonic module to measure distance, as shown in the following picture:



The following is the program code:

```
#include <UltrasonicSensor.h>

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

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

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

First, add UltrasonicSensor library.

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

Define an ultrasonic object and associate it with the pins.

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

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

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

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

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

Reference

class UltrasonicSensor

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

UltrasonicSensor ultrasonic(19, 18);

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

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

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

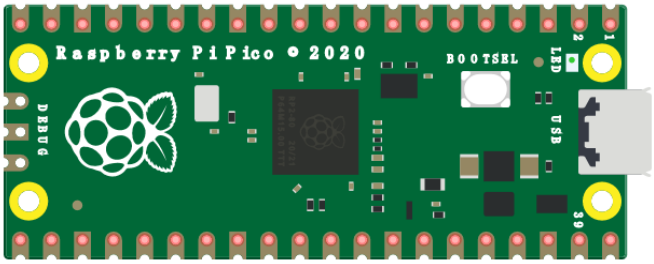
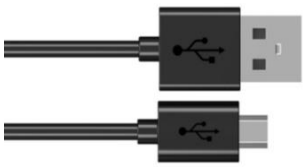
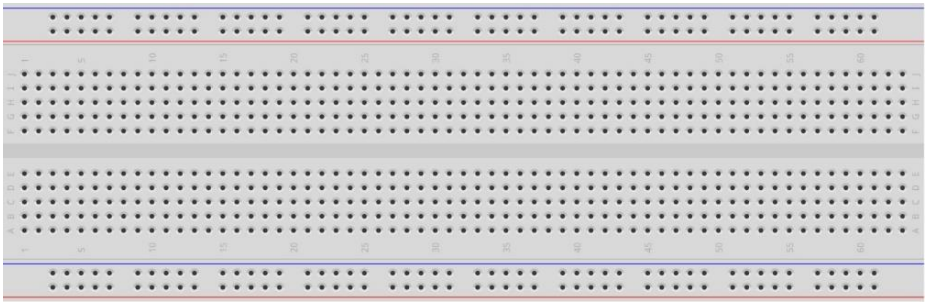
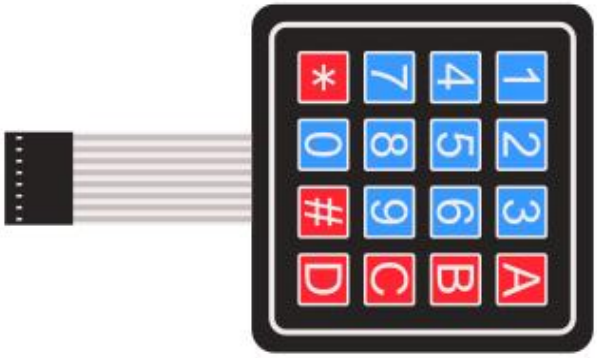
Chapter 23 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 23.1 Matrix Keypad

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

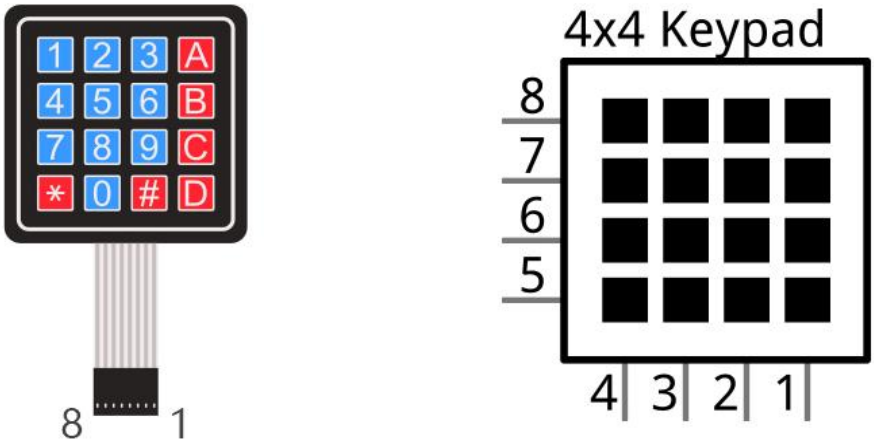
Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	4x4 Matrix Keypad x1 

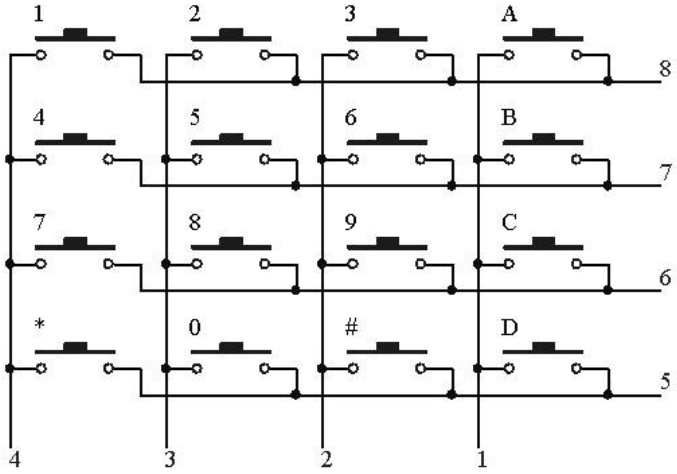
Component Knowledge

4x4 Matrix Keypad

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



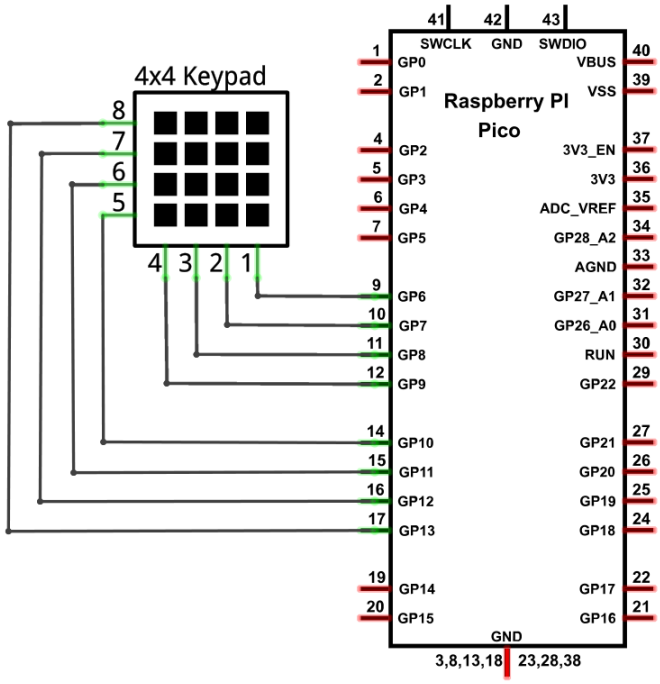
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



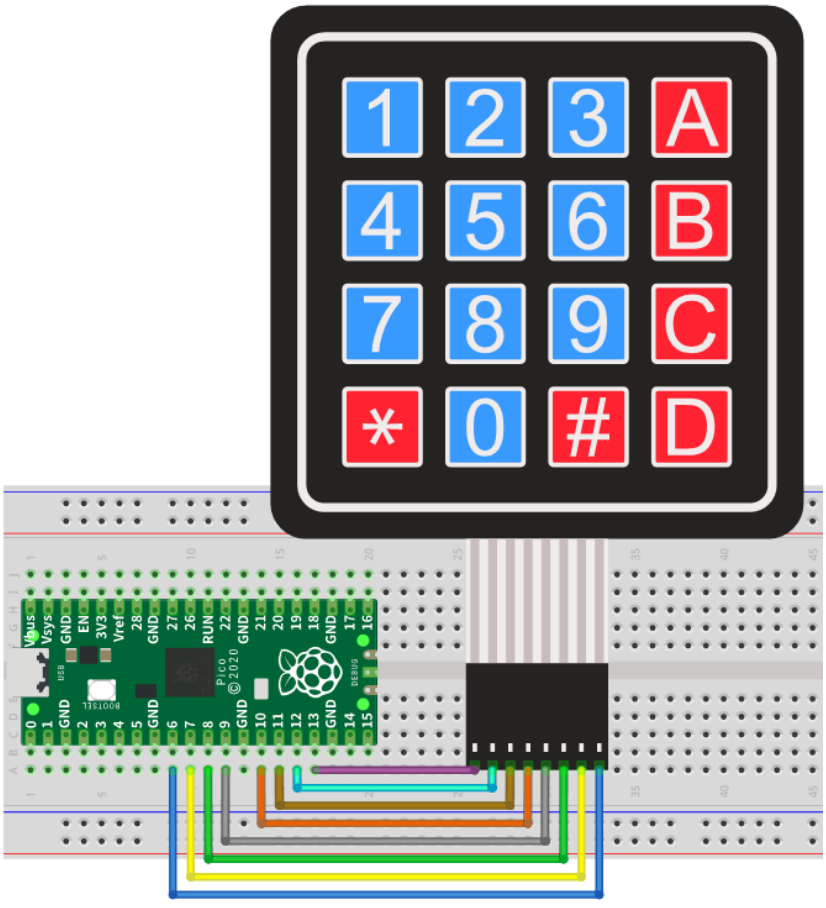
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to determine whether the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

Circuit

Schematic diagram



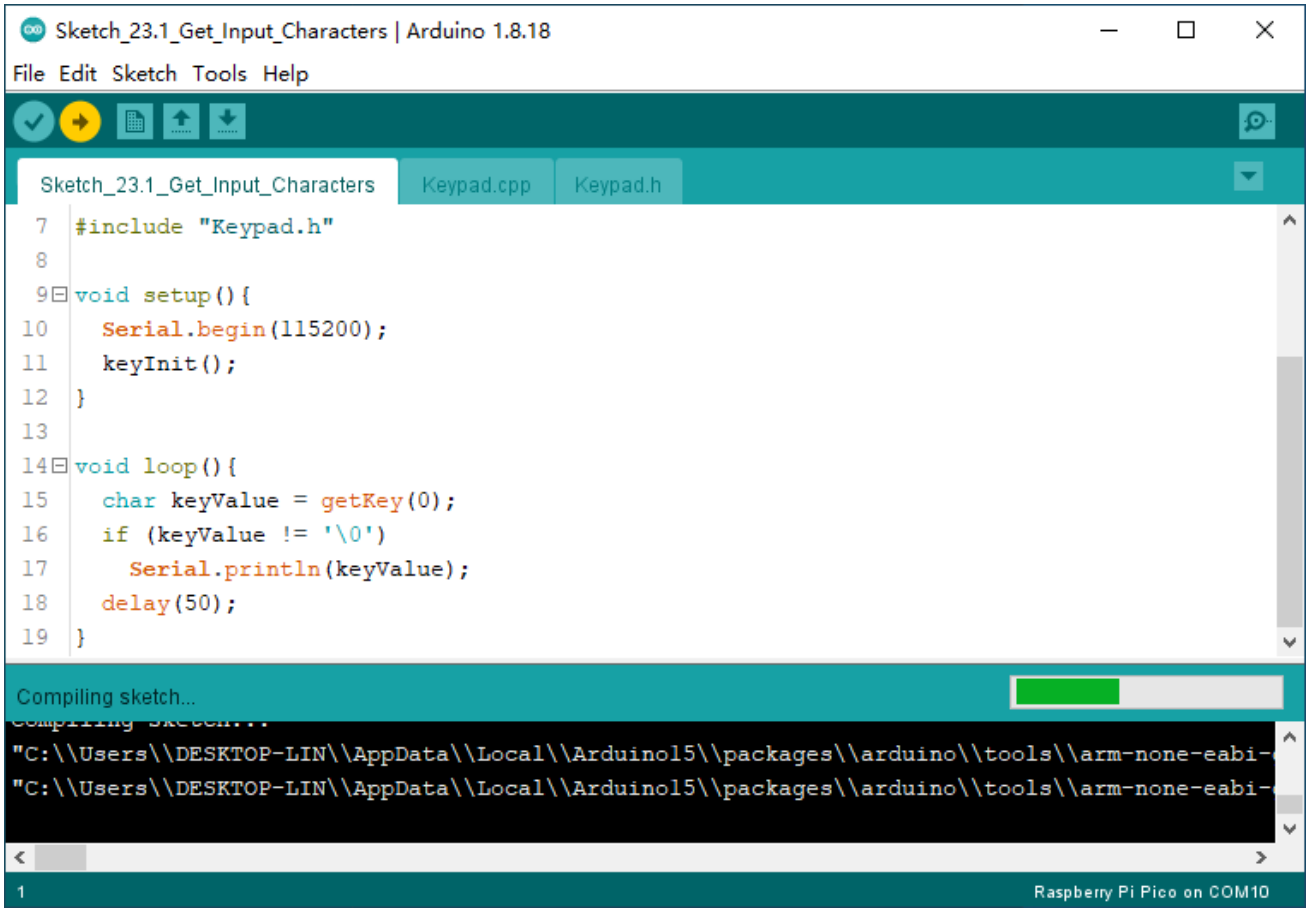
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



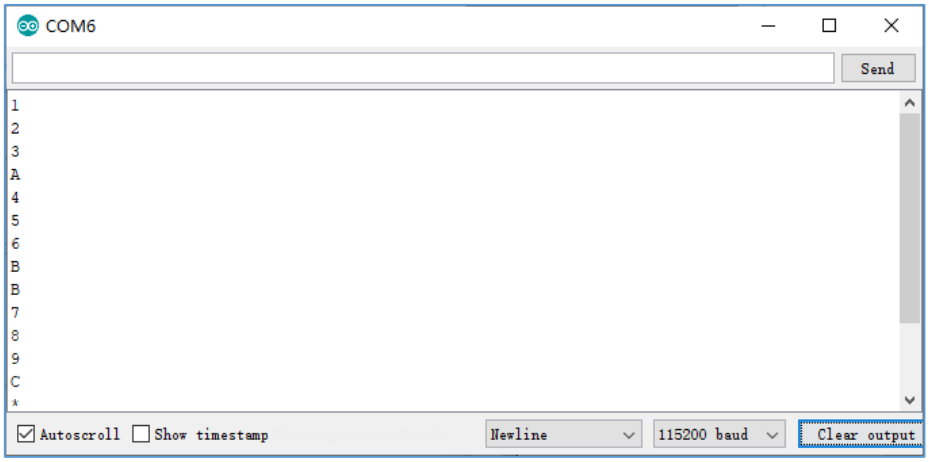
Sketch

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

Sketch_23.1_Get_Input_Characters



Download the code to Pico, open the serial port monitor, set the baud rate to 115200, press the keyboard, the value of the pressed keys will be printed out via the serial port, as shown in the following picture:



Keypad.cpp

```
#include "Keypad.h"

byte rowPin[4] = {13, 12, 11, 10};
byte colPin[4] = { 9,  8,  7,  6};

char keyStrings[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

int lastTime = 0;
int debounceTime = 20;

int pressKeyRow=0;
int pressKeyCol=0;
bool pressState=false;

void keyInit(void){
  for (int r = 0; r < sizeof(rowPin); r++){
    pinMode(colPin[r], OUTPUT);
    digitalWrite(colPin[r], HIGH);
    pinMode(rowPin[r], INPUT_PULLUP);
  }
  for (int c = 0; c < sizeof(colPin); c++){
    pinMode(colPin[c], OUTPUT);
    digitalWrite(colPin[c], HIGH);
  }
}

void keyScan(bool state){
  for (int c = 0; c < sizeof(colPin); c++){
    digitalWrite(colPin[c], LOW);
    for (int r = 0; r < sizeof(rowPin); r++){
      if (digitalRead(rowPin[r]) == LOW){
        while (state == true && digitalRead(rowPin[r]) == LOW);
        digitalWrite(colPin[c], HIGH);
        pressKeyRow=r;
        pressKeyCol=c;
        pressState=true;
      }
    }
```

```
    digitalWrite(colPin[c], HIGH);
  }
}

char getKey(bool state){
  if (millis() - lastTime > debounceTime){
    pressState = false;
    keyScan(state);
    lastTime = millis();
    if(pressState==true)
      return keyStrings[pressKeyRow][pressKeyCol];
    else
      return '\0';
  }
}
```

Include the header file, define the pins to control the keypad's rows and columns and define an array to store the key values of the keypad being pressed.

You can modify the following code to change key values and the pins controlling the keypad.

```
#include "Keypad.h"

byte rowPin[4] = {13, 12, 11, 10};
byte colPin[4] = { 9,  8,  7,  6};

char keyStrings[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```

The following is the program code:

```
#include "Keypad.h"

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

void loop(){
  char keyValue = getKey(0);
  if (keyValue != '\0')
    Serial.println(keyValue);
  delay(50);
}
```

keyInit() function is called to initialize the pins controlling the keypad.

```
5    keyInit();
```

getKey() is called to scan the matrix keyboard and return "\0" if no key is detected being pressed; and it returns the key value character of the pressed key when a key is detected to be pressed

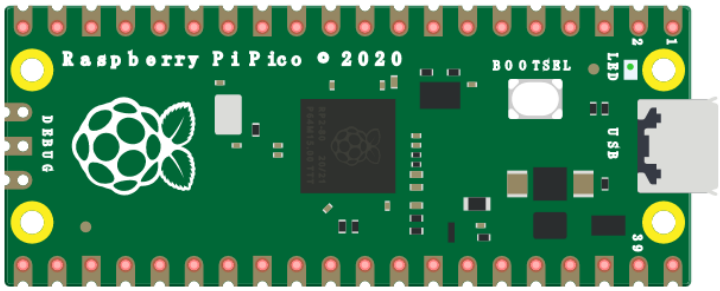
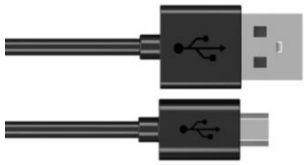
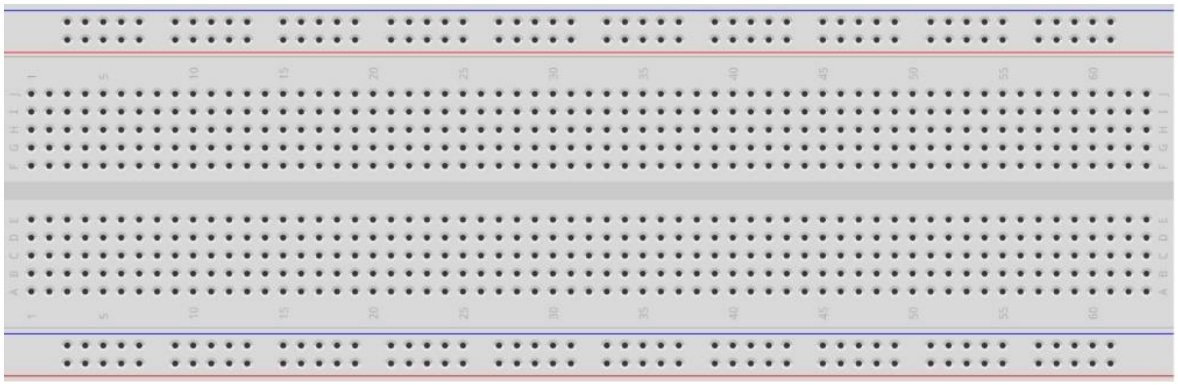
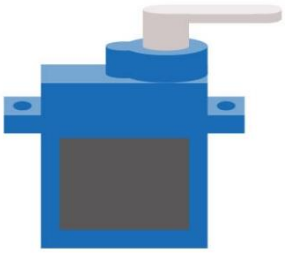
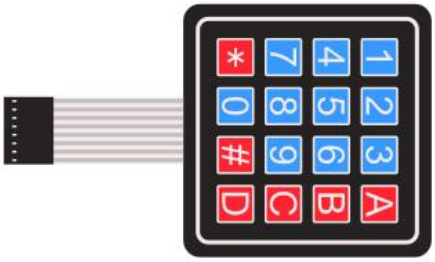
When the parameter of the getKey() function is 1, the matrix keyboard scans in an inching mode. In this mode, when the key is pressed and not released, the program will stop execution until the key is released. When the parameter is 0, the matrix keyboard scans in a continuous mode. In this mode, the program will not stop execution because the key is not released.

```
9    char keyValue = getKey(0);
```

Project 23.2 Keypad Door

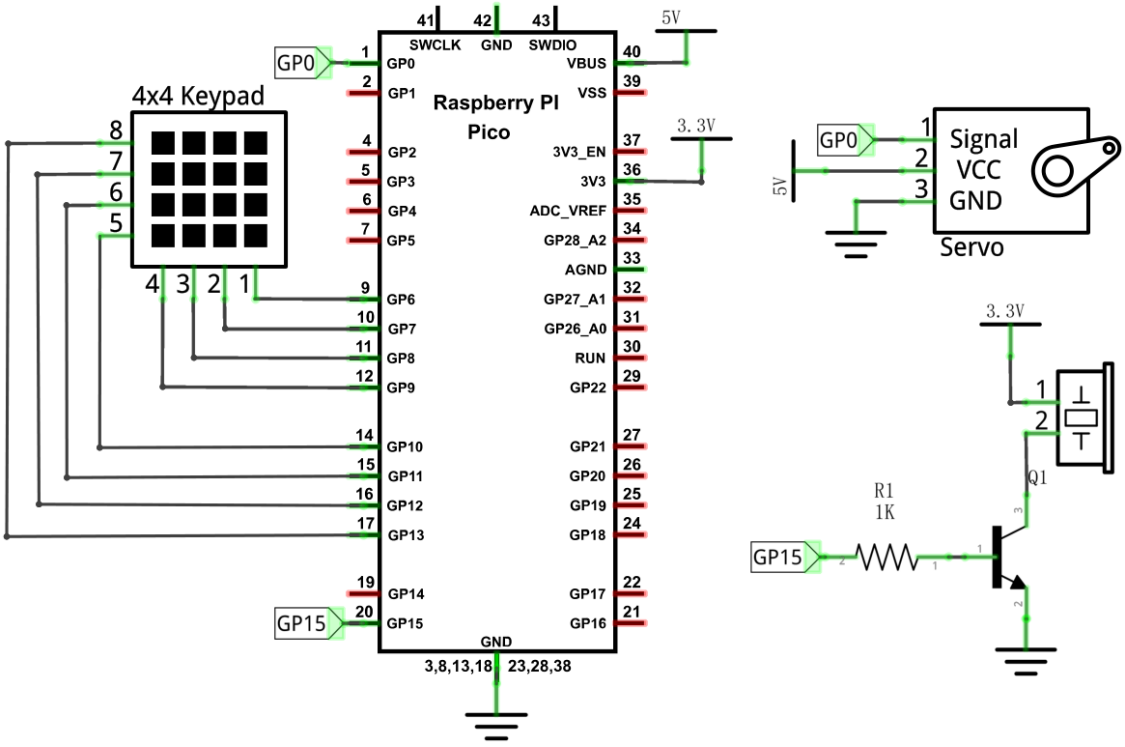
In this project, we use keypad as a keyboard to control the action of the servo motor.

Component List

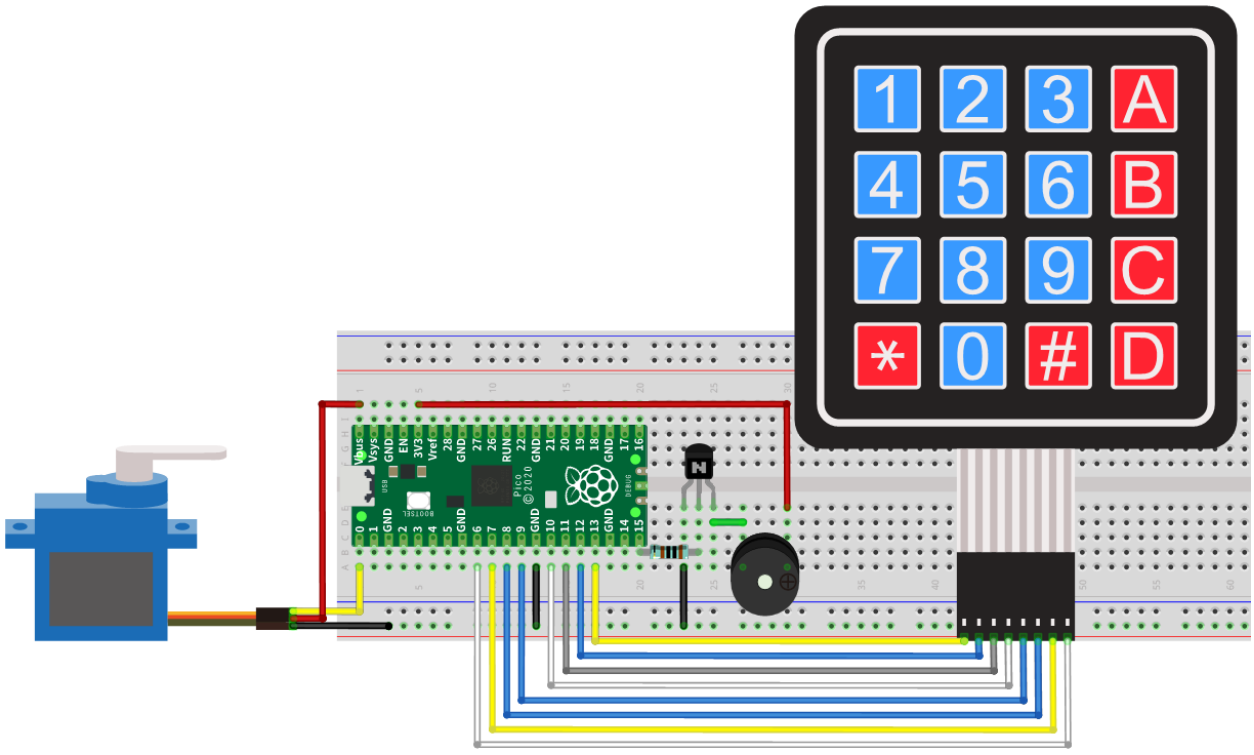
<table>
<tr><td>Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Jumper
  </td><td>Servo x1
  </td><td>4x4 Matrix Keypad x1
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Resistor 1kΩ x1
  </td></tr>
</table>

Circuit

Schematic diagram

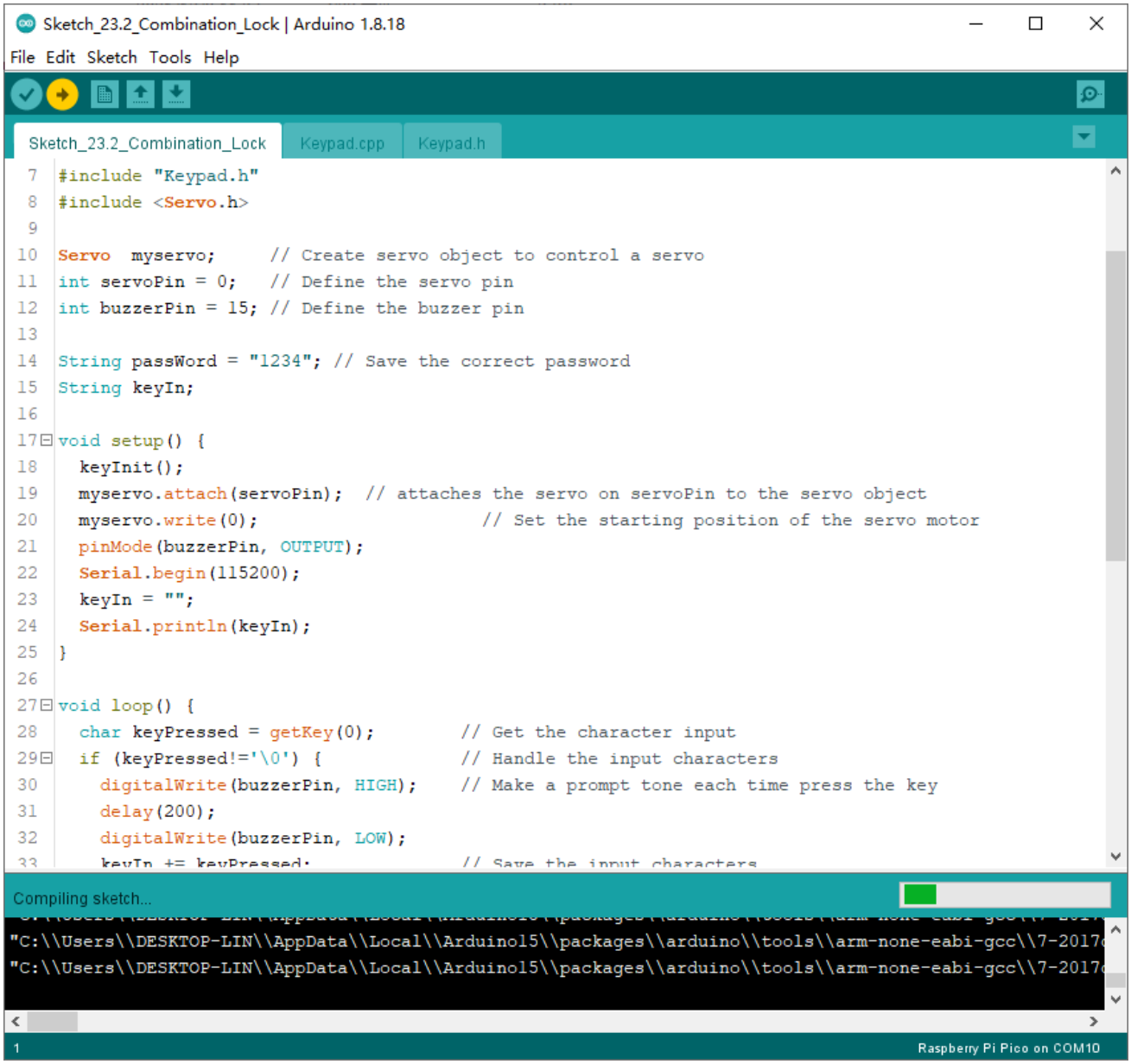


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

Sketch_23.2_Keypad_Door



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

The following is the program code:

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

Servo  myservo;     // Create servo object to control a servo
int servoPin = 0;   // Define the servo pin
int buzzerPin = 15; // Define the buzzer pin

String passWord = "1234"; // Save the correct password
String keyIn;

void setup() {
  keyInit();
  myservo.attach(servoPin);  // attaches the servo on servoPin to the servo object
  myservo.write(0);                    // Set the starting position of the servo motor
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(115200);
  keyIn = "";
  Serial.println(keyIn);
}

void loop() {
  char keyPressed = getKey(0);         // Get the character input
  if (keyPressed!='\0') {              // Handle the input characters
    digitalWrite(buzzerPin, HIGH);     // Make a prompt tone each time press the key
    delay(200);
    digitalWrite(buzzerPin, LOW);
    keyIn += keyPressed;               // Save the input characters
    Serial.println(keyPressed);        // Judge the correctness after input
    if (keyIn.length() == 4) {
      bool isRight = true;             // Save password is correct or not
      if( passWord != keyIn){
        isRight = !true;
      }
      if (isRight) {                   // If the input password is right
        myservo.attach(servoPin);
        myservo.write(90);             // Open the switch
        delay(2000);                   // Delay a period of time
        myservo.write(0);              // Close the switch
        Serial.println("passWord right!");
      }
      else {                           // If the input password is wrong
        digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
        delay(1000);
```

```
44            digitalWrite(buzzerPin, LOW);
45            Serial.println("passWord error!");
46          }
47          keyIn = ""; // Reset the number of the input characters to 0
48        }
49      }
50      delay(200);
51  }
```

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

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

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

```
22  char keyPressed = getKey(0);          // Get the character input
23    if (keyPressed!='\0') {             // Handle the input characters
24      digitalWrite(buzzerPin, HIGH);    // Make a prompt tone each time press the key
25      delay(200);
26      digitalWrite(buzzerPin, LOW);
27      keyIn += keyPressed;
```

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

```
29      if (keyIn.length() == 4) {
30        bool isRight = true;            // Save password is correct or not
31        if( passWord != keyIn){
32          isRight = !true;
33        }
```

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

```
34        if (isRight) {                  // If the input password is right
35          myservo.attach(servoPin);
36          myservo.write(90);            // Open the switch
37          delay(2000);                  // Delay a period of time
38          myservo.write(0);             // Close the switch
39          Serial.println("passWord right!");
40        }
41        else {                          // If the input password is wrong
42          digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
43          delay(1000);
44          digitalWrite(buzzerPin, LOW);
45          Serial.println("passWord error!");
46        }
```

Finally, remember to empty the keyInNum every time.

```
47        keyIn = ""; // Reset the number of the input characters to 0
```

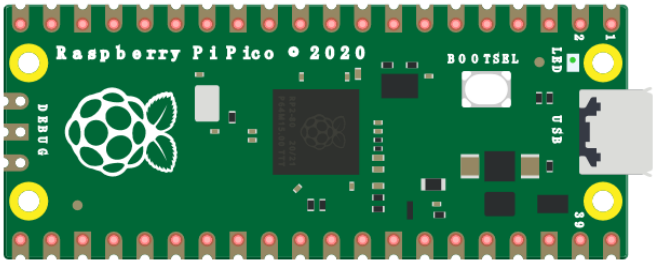
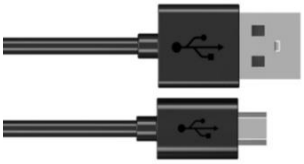
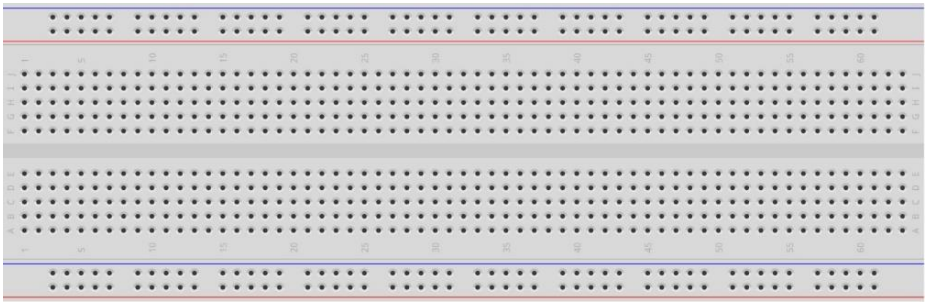
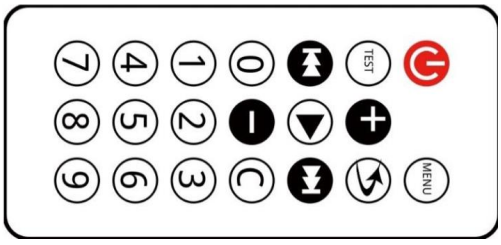
Chapter 24 Infrared Remote

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

Project 24.1 Infrared Remote Control

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

Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	Infrared Remote x1 (May need CR2025 battery x1, please check the holder) 
Infrared Remote x1 	Resistor 10kΩ x1 

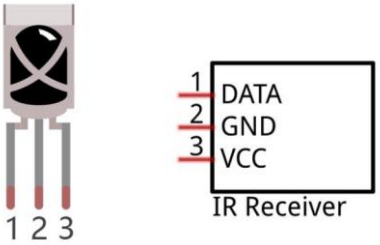
Component Knowledge

Infrared Remote

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

Infrared receiver

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

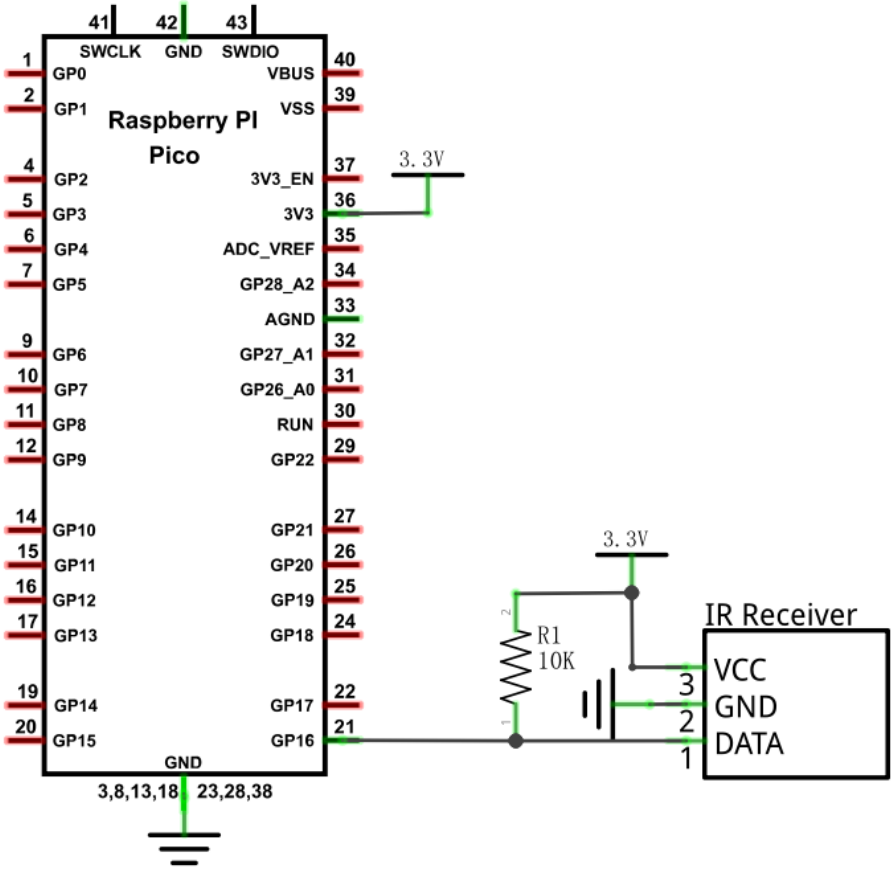


When you use the infrared remote control, it sends a key value to the receiving circuit according to the pressed key. We can program the Raspberry Pi Pico to do things like lighting, when a key value is received.
The following is the key value that the receiving circuit will receive when each key of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

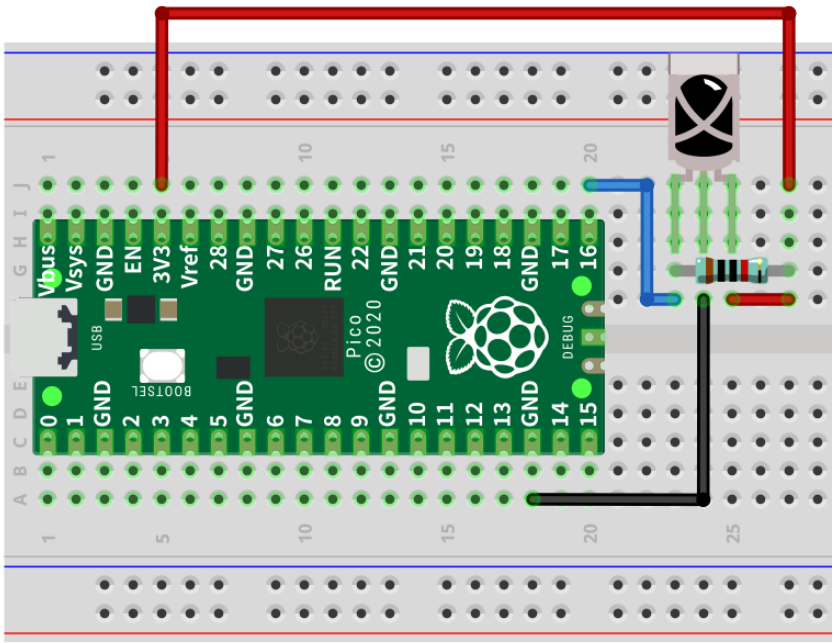
ICON	KEY Value	ICON	KEY Value
	FFA25D		FFB04F
	FFE21D		FF30CF
	FF22DD		FF18E7
	FF02FD		FF7A85
	FFC23D		FF10EF
	FFE01F		FF38C7
	FFA857		FF5AA5
	FF906F		FF42BD
	FF6897		FF4AB5
	FF9867		FF52AD

Circuit

Schematic diagram



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

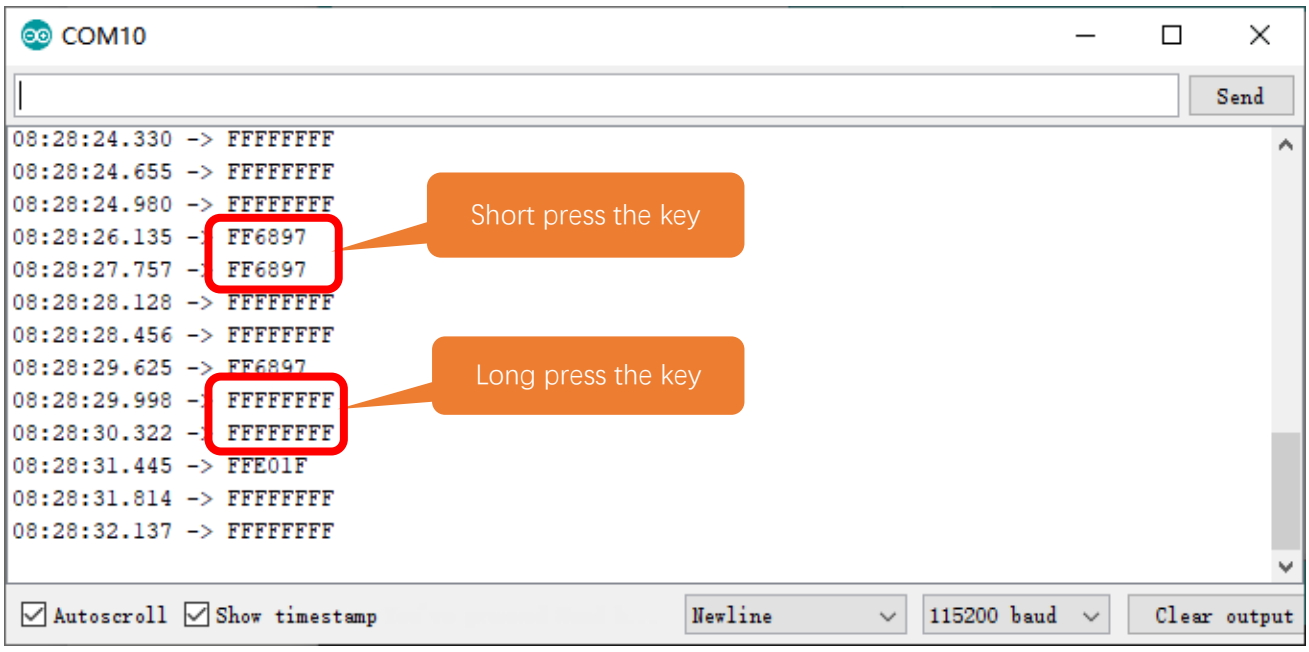


Sketch

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

Sketch_24.1_Infrared_Remote_Control

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



IR.cpp

```
#include "IR.h"

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

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

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

```
44            logList[num ++] = 0;
45          }
46          else {
47            logList[num ++] = 1;
48          }
49        }
50      }
51    }
52  }
53
54  unsigned long IR_Decode(int &code) {
55    unsigned long irData = 0;
56    irState=false;
57    if (code == 1) {
58      code = 0;
59      for (int i = 0; i < 32; i ++) {
60        if (logList[i] == 0) {
61          irData <<= 1;
62        }
63        else {
64          irData <<= 1;
65          irData ++;
66        }
67        logList[i] = 0;
68      }
69
70    }
71    if (code == 2) {
72      code = 0;
73      irData = 0xffffffff;
74    }
75    return irData;
76  }
77
78  void IR_Release(){
79    irState=true;
80  }
```

When the IR_Init() function is called, Pico initializes the infrared received pin and sets the external interrupt, associating it with the IR_Read() function. Every time the infrared receives data, external interrupt calls IR_Read() function to receive data, and resets the bit flag.

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

You can check whether flagCode has been reset, If it is reset, call IR_Decode() to decode the infrared data. Note: once IR_Decode() is called, infrared receiver won't receive data until IR_Release() is called.

```
unsigned long IR_Decode(int &code);
void IR_Release();
```

The following is the program code:

```
1  #include "IR.h"
2  #define IR_Pin 16
3
4  void setup() {
5    Serial.begin(115200);
6    IR_Init(IR_Pin);
7  }
8
9  void loop() {
10   if(flagCode){
11     int irValue = IR_Decode(flagCode);
12     Serial.println(irValue, HEX);
13     IR_Release();
14   }
15 }
```

IR_Init() is called to initialize infrared receiving pin GP16, enable external interrupt and associate it with GP16.

```
6    IR_Init(IR_Pin);
```

In loop(), determines whether infrared bit flag is reset. If it is, IR_Decode() is called to decode the data and print them out via serial monitor.

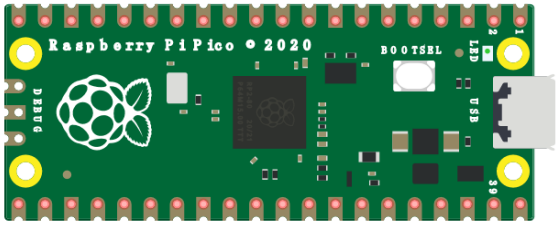
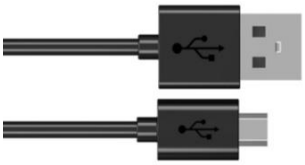
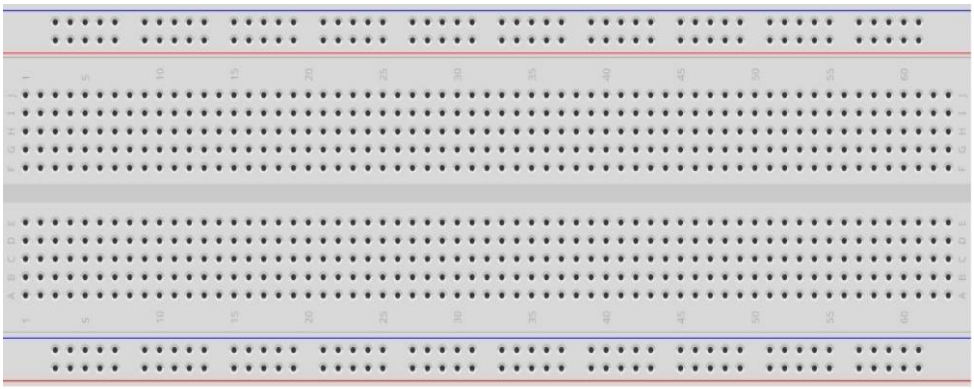
After using the infrared decoding function IR_Decode(), you need to call IR_Release() to release the infrared data receiving function. Otherwise, it won't receiver new infrared data again.

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

Project 24.2 Control LED through Infrared Remote

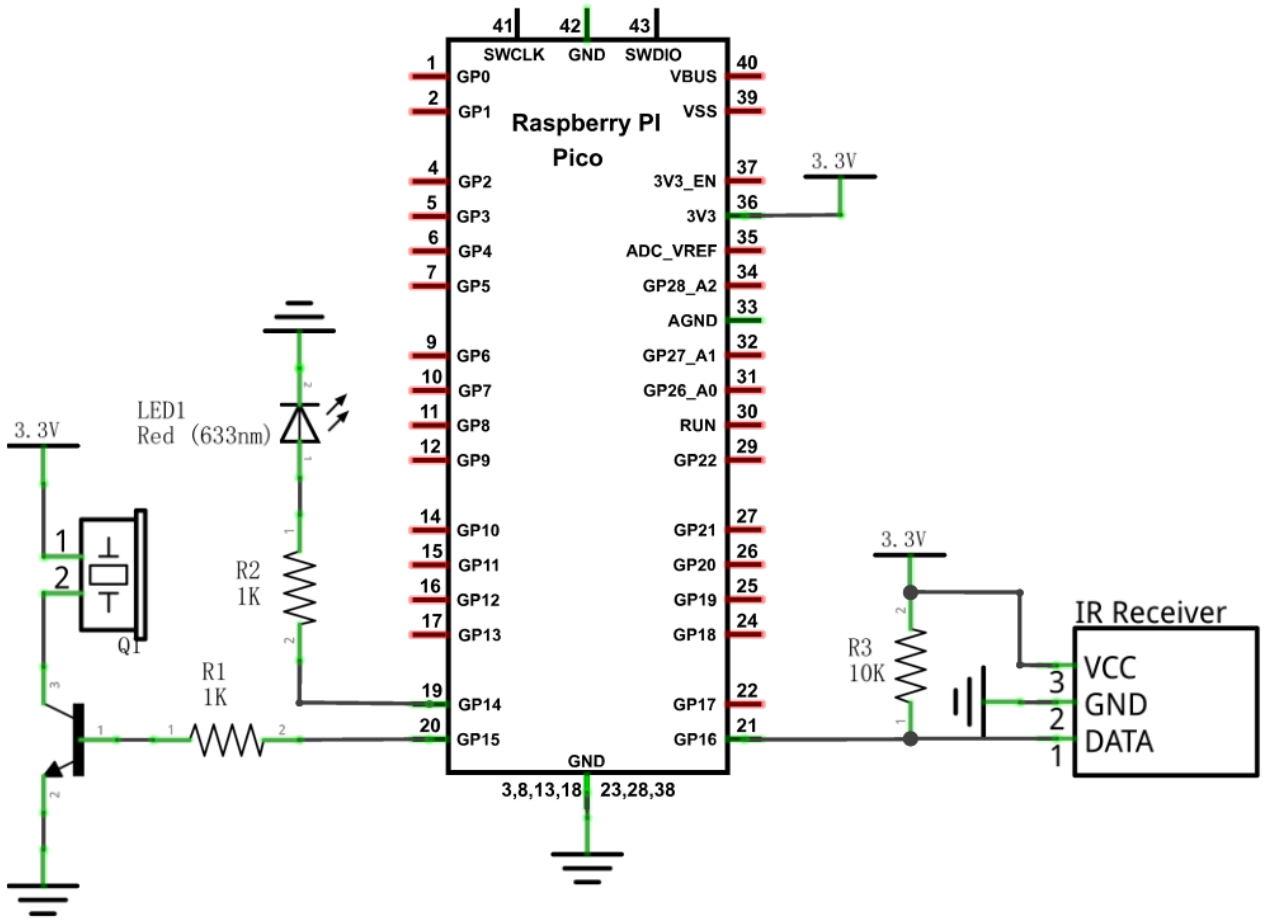
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

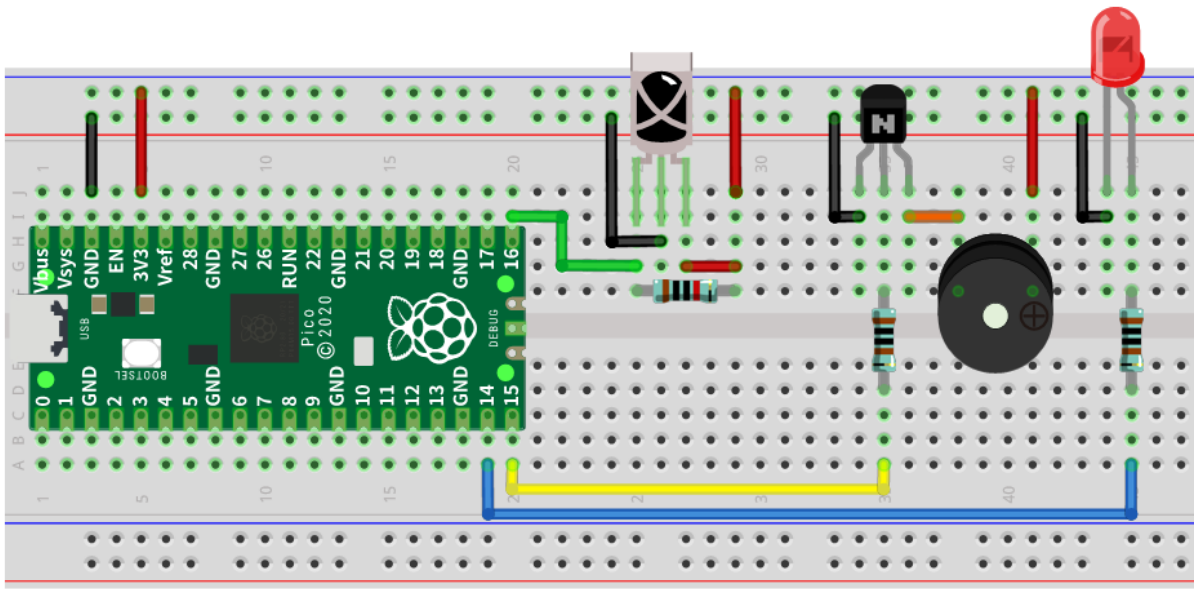
<table>
<tr><td>Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td colspan="2">Jumper
  </td><td rowspan="3">Infrared Remote x1
(May need CR2025 battery x1, please check the battery holder)
  </td></tr>
<tr><td>LED x1
  </td><td>Active buzzer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

Circuit

Schematic diagram



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



Sketch

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

Sketch_24.2_Control_LED_through_Infrared_Remote

Sketch_24.2_Control_LED_through_Infrared_Remote | Arduino 1.8.18

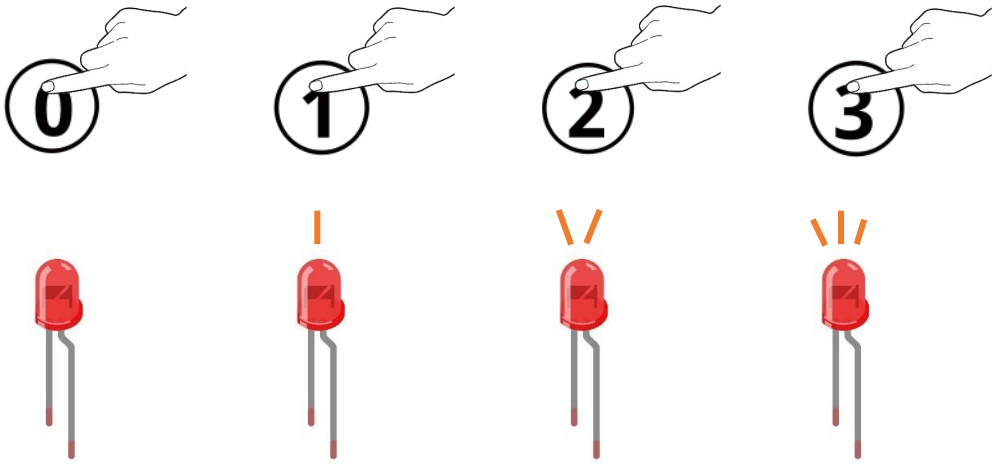
File Edit Sketch Tools Help

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

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

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

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

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

Rendering:



The following is the program code:

```
#include "IR.h"

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

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

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

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

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

```
23  void handleControl(unsigned long value) {
24    digitalWrite(buzzerPin, HIGH);
25    delay(100);
26    digitalWrite(buzzerPin, LOW);
27    // Handle the commands
28    switch (value) {
29      case 0xFF6897:              // Receive the number '0'
30        analogWrite(ledPin, 0);   // Turn off LED
31        break;
32      case 0xFF30CF:              // Receive the number '1'
33        analogWrite(ledPin, 50);  // Dimmest brightness
34        break;
35      case 0xFF18E7:              // Receive the number '2'
36        analogWrite(ledPin, 100); // Medium brightness
37        break;
38      case 0xFF7A85:              // Receive the number '3'
39        analogWrite(ledPin, 255); // Strongest brightnss
40        break;
41    }
42  }
```

In the loop() function, each time the infrared data is received, it is decoded and printed out through the serial monitor, and the handleControl() function is called to control the LED and buzzer to execute the corresponding code.

```
14  void loop() {
15    if(flagCode){
16      int irValue = IR_Decode(flagCode);
17      Serial.println(irValue, HEX);
18      handleControl(irValue);
19      IR_Release();
20    }
21  }
```

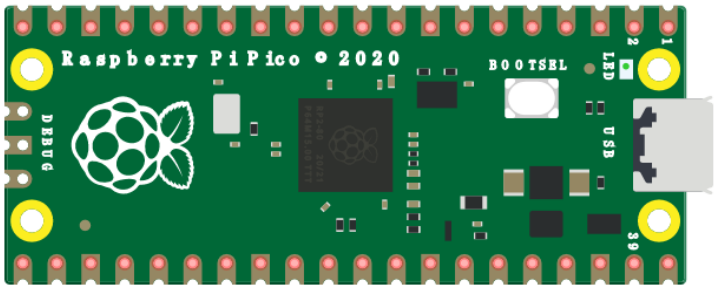
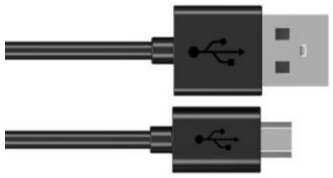
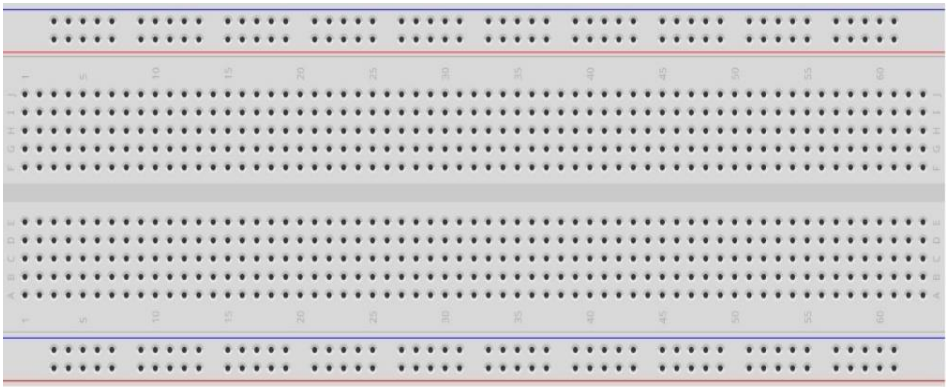
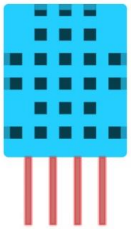
Chapter 25 Hygrothermograph DHT11

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

Project 25.1 Hygrothermograph

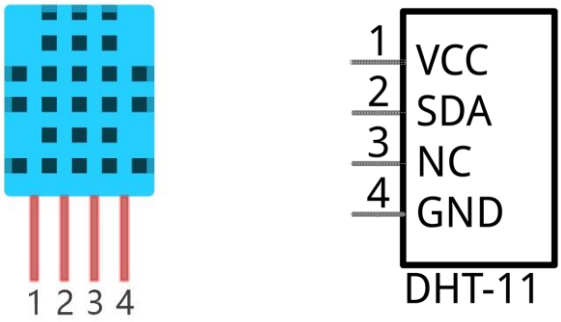
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the Raspberry Pi Pico to read Temperature and Humidity data of the DHT11 Module.

Component List

<table>
<tr><td>Raspberry Pi Pico x1
  </td><td>USB cable x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Jumper
  </td><td>DHT11 x1
  </td><td>Resistor 10kΩ x1
  </td></tr>
</table>

Component Knowledge

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



DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

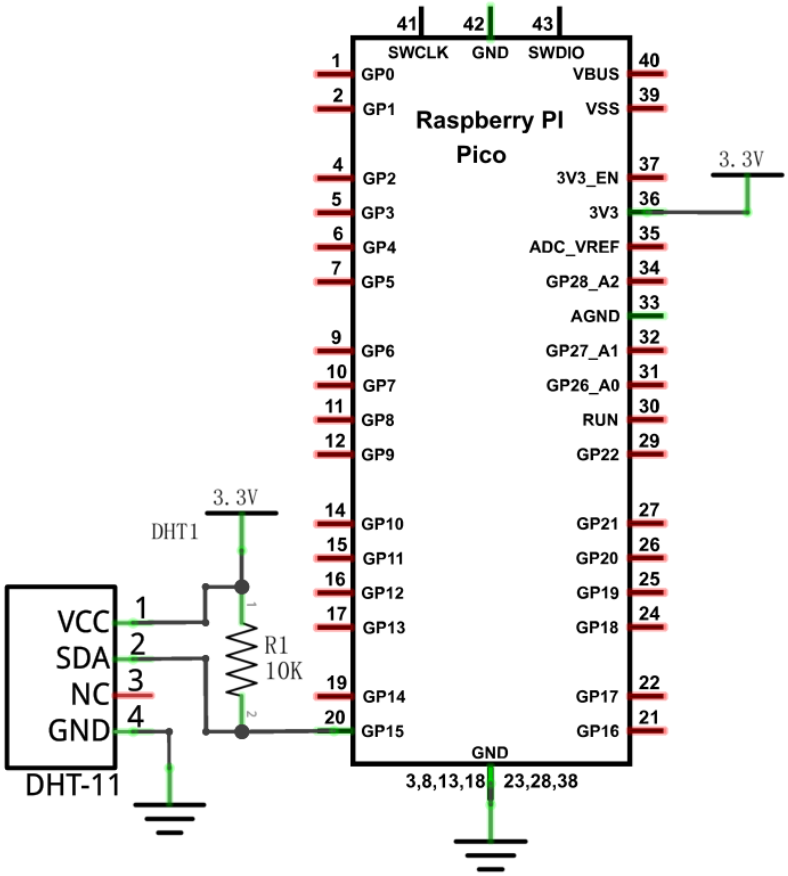
After being powered up, it will initialize in 1S's time. Its operating voltage is within the range of 3.3V-5.5V.

The SDA pin is a data pin, which is used to communicate with other devices.

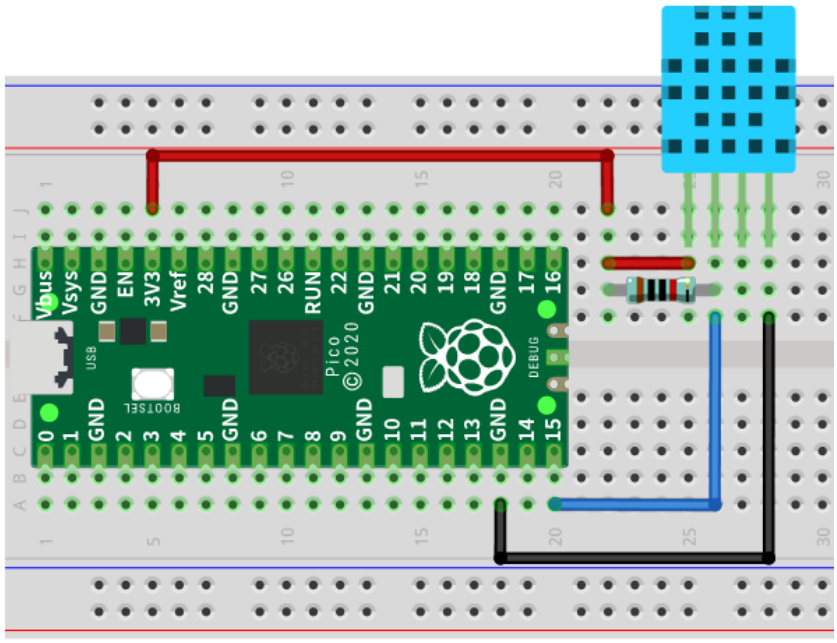
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



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



Sketch

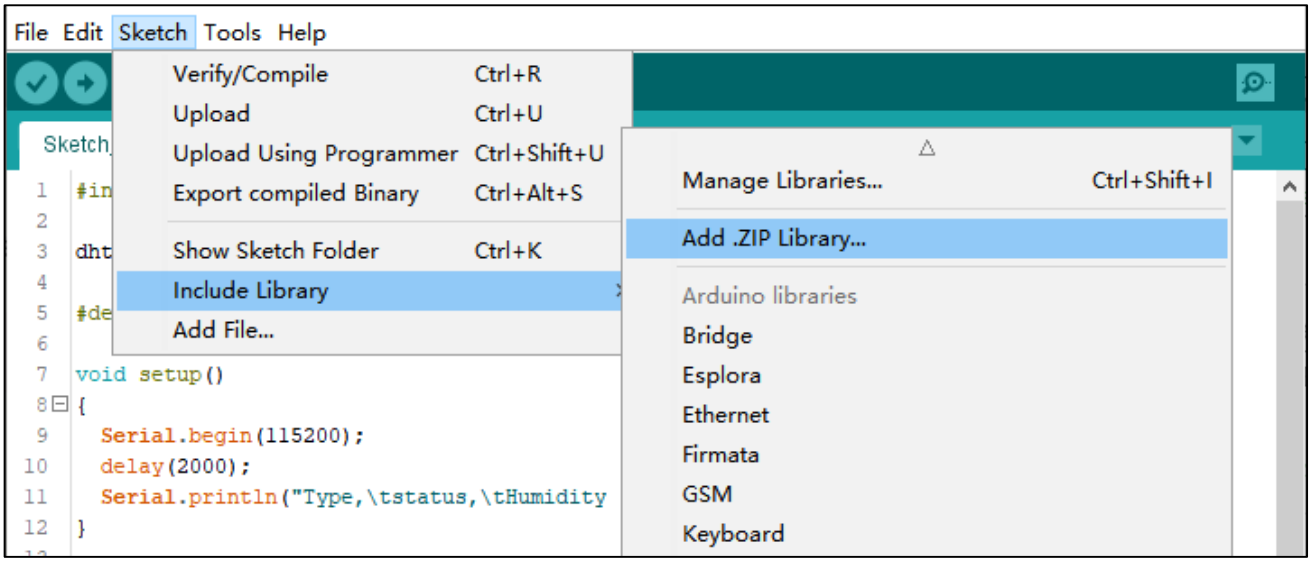
How to install the library

We use a third-party library DHT for this project. If you haven't installed it yet, please do so first.

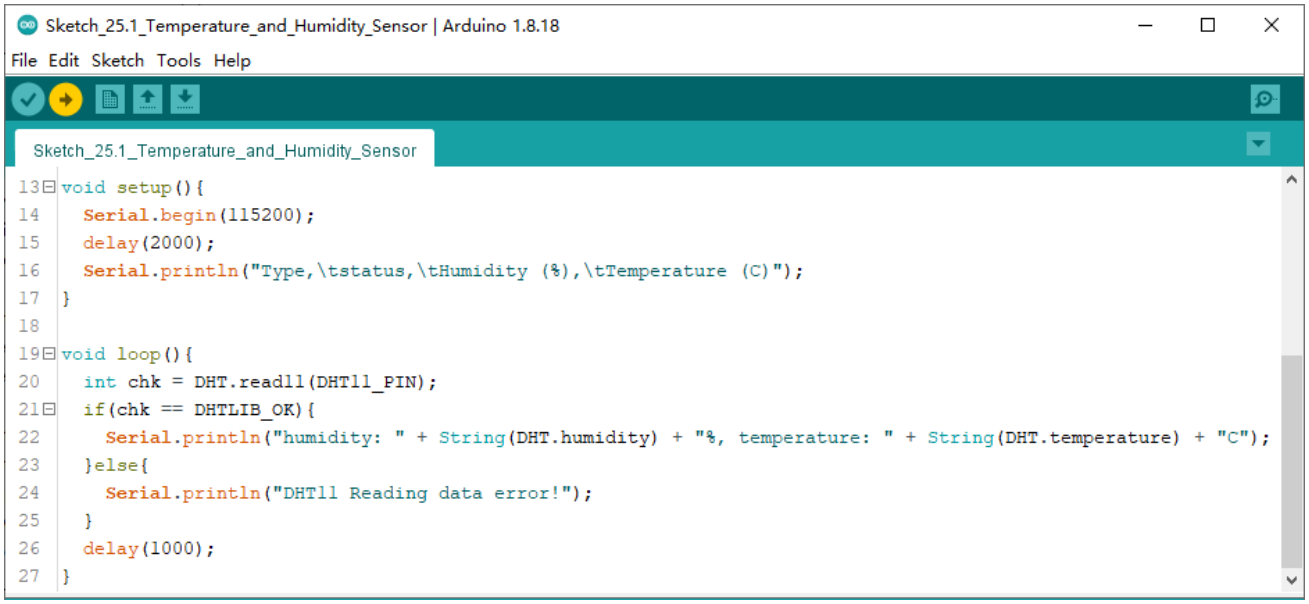
Steps are as follows:

Open **Arduino>Sketch>Include Library>Add .ZIP Library...**

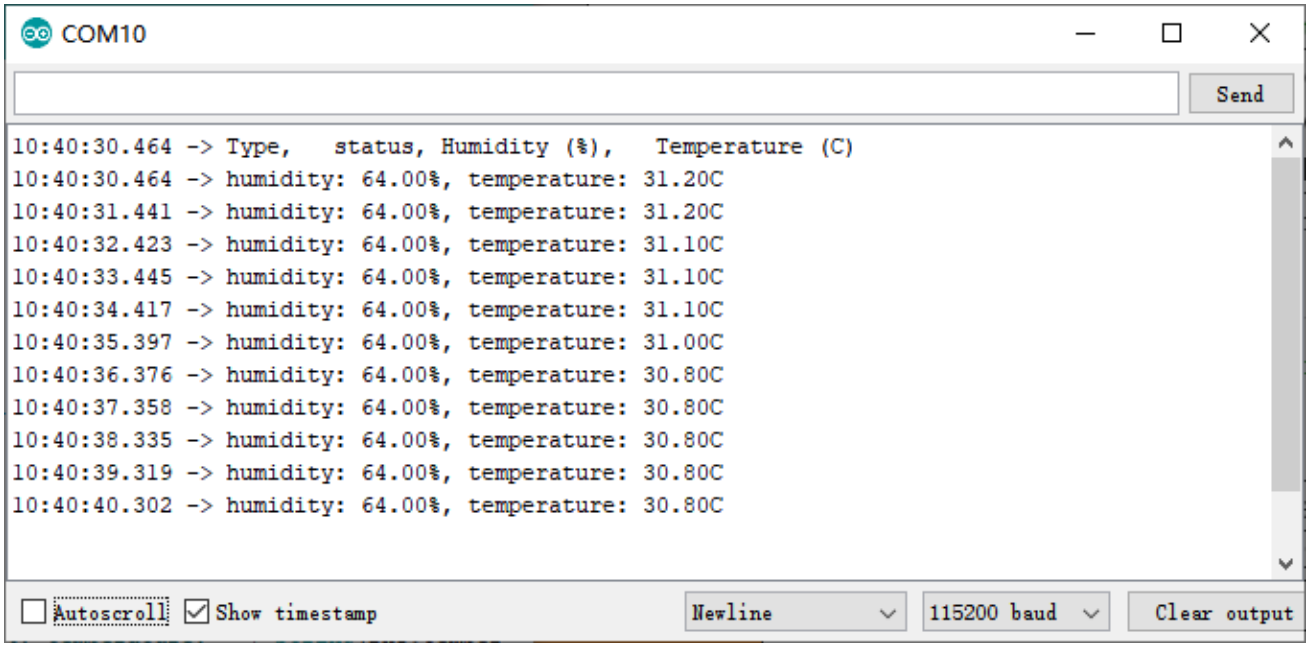
Select the provided **"Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\C\Libraries\DHT.zip"。**



Sketch_25.1_Temperature_and_Humidity_Sensor



Compile and upload the code to the Pico, open the serial monitor, and set the baud rate to 115200. Print out data of temperature and humidity sensor via the serial port.



The following is the program code:

```
#include <dht.h>

dht DHT;
#define DHT11_PIN 15

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Type,\tstatus,\tHumidity (%),\tTemperature (C)");
}

void loop(){
  int chk = DHT.read11(DHT11_PIN);
  if(chk == DHTLIB_OK){
    Serial.println("humidity: " + String(DHT.humidity) + "%, temperature: " +
String(DHT.temperature) + "C");
  }else{
    Serial.println("DHT11 Reading data error!");
  }
  delay(1000);
}
```

Before using dht11, we need to include a header file. Apply for a DHT object and define the pin controlling DHT as GP15.

```
#include <dht.h>

dht DHT;
#define DHT11_PIN 15
```

Read11() is used to read DHT11 data and assign the return value to variable chk.

```
  int chk = DHT.read11(DHT11_PIN);
```

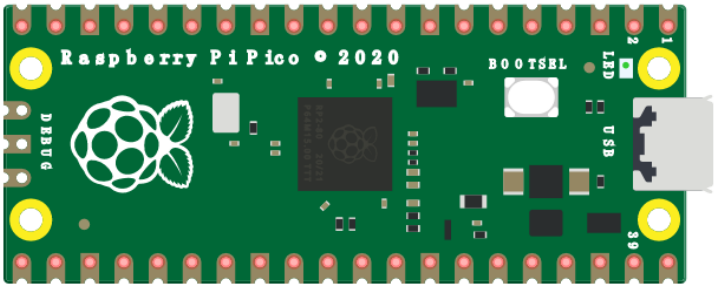
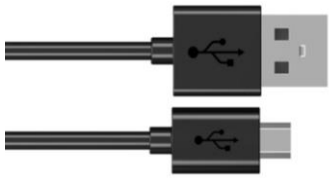
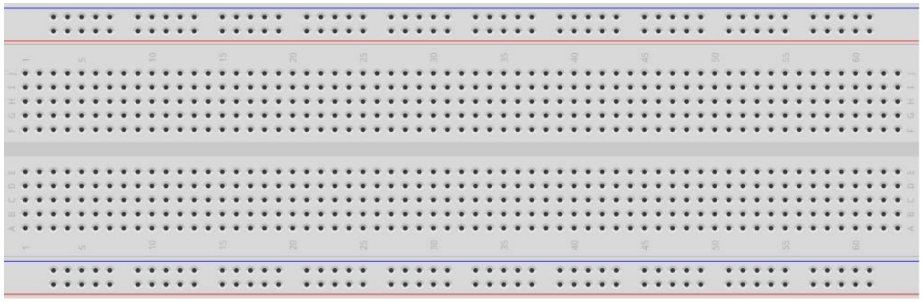
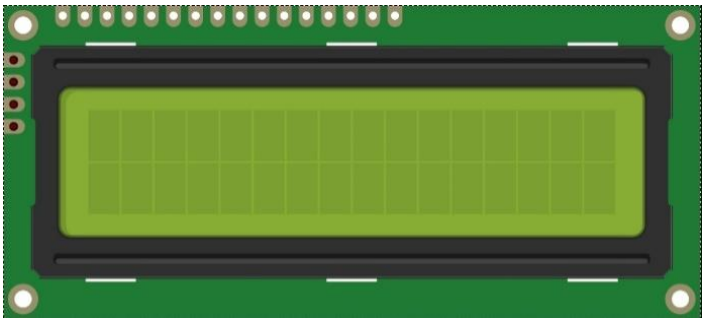
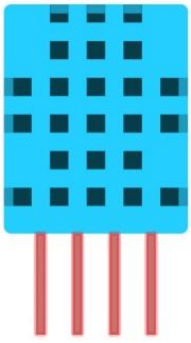
If the return value of the read11() function is not equal to DHTLIB_OK, it means that the data reading failed; If they equals, humuduty() and temperature() are called to obtain the temperature and humidity data of the current environment, and print it out through the serial port.

```
  if(chk == DHTLIB_OK){
    Serial.println("humidity: " + String(DHT.humidity) + "%, temperature: " +
String(DHT.temperature) + "C");
  }else{
    Serial.println("DHT11 Reading data error!");
  }
```

Project 25.2 Hygrothermograph

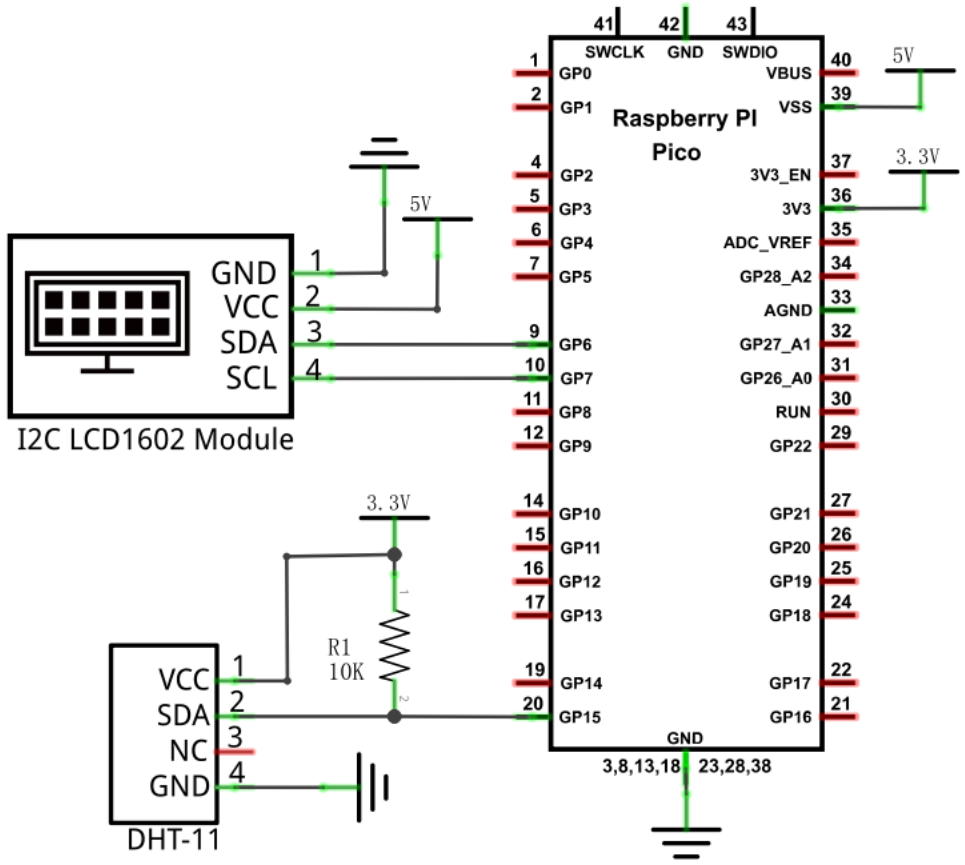
In this project, we use I2C-LCD1602 to display data collected by DHT11.

Component List

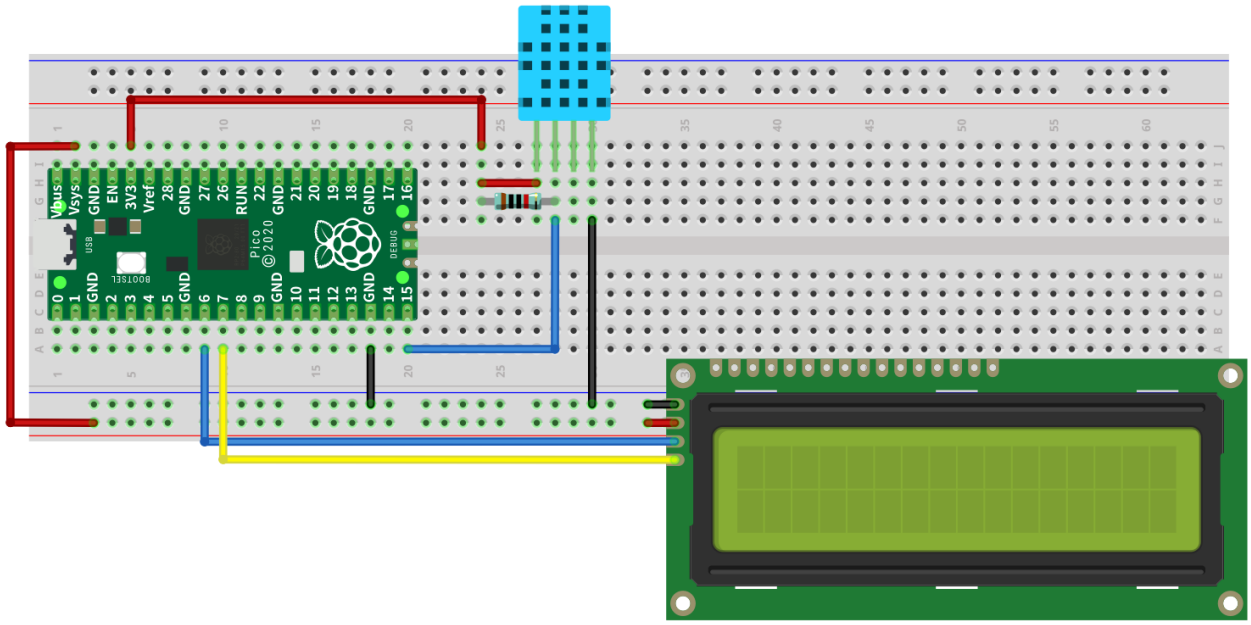
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
LCD1602 Module x1 	Resistor 10kΩ x1 
Jumper 	DHT11 x1 

Circuit

Schematic diagram



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

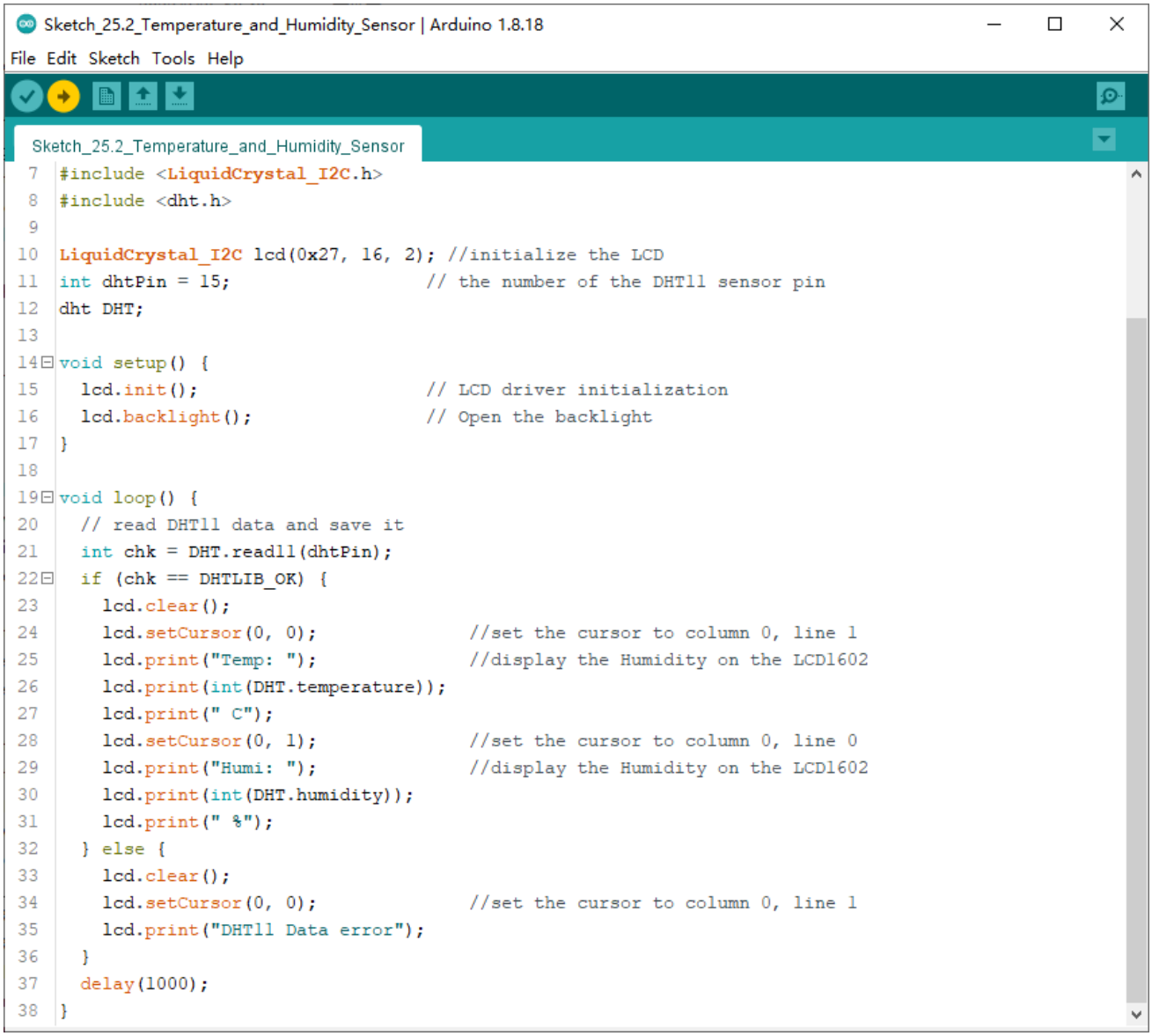


Obtain data of Hygrothermograph every second and display them on LCD1602. The first line displays

Sketch

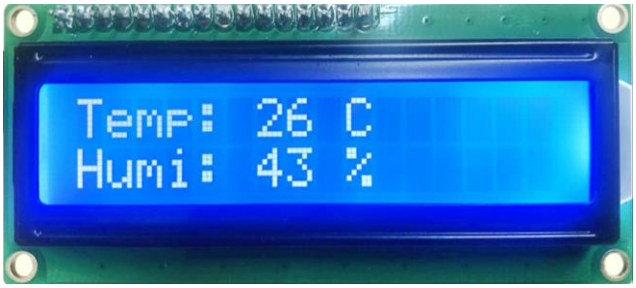
This code uses the DHT and LiquidCrystal_I2C libraries, so make sure the relevant library files are added before writing the program.

Sketch_25.2_Temperature_and_Humidity_Sensor




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

LiquidCrystal_I2C lcd(0x27, 16, 2); //initialize the LCD
int dhtPin = 15;                    // the number of the DHT11 sensor pin
dht DHT;

void setup() {
  lcd.init();                       // LCD driver initialization
  lcd.backlight();                  // Open the backlight
}

void loop() {
  // read DHT11 data and save it
  int chk = DHT.readl1(dhtPin);
  if (chk == DHTLIB_OK) {
    lcd.clear();
    lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
    lcd.print("Temp: ");              //display the Humidity on the LCD1602
    lcd.print(int(DHT.temperature));
    lcd.print(" C");
    lcd.setCursor(0, 1);              //set the cursor to column 0, line 0
    lcd.print("Humi: ");              //display the Humidity on the LCD1602
    lcd.print(int(DHT.humidity));
    lcd.print(" %");
  } else {
    lcd.clear();
    lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
    lcd.print("DHT11 Data error");
  }
  delay(1000);
}
```

Download the code to Pico. The first line of LCD1602 shows the temperature value, and the second line shows the humidity value. Try to "pinch" the DHT11(without touching the leads) with your index finger and thumb for a brief time to observe the change in the LCD display value.



The following is the program code:

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

LiquidCrystal_I2C lcd(0x27, 16, 2); //initialize the LCD
int dhtPin = 15;                    // the number of the DHT11 sensor pin
dht DHT;

void setup() {
  lcd.init();                     // LCD driver initialization
  lcd.backlight();                // Open the backlight
}

void loop() {
  // read DHT11 data and save it
  int chk = DHT.read11(dhtPin);
  if (chk == DHTLIB_OK) {
    lcd.clear();
    lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
    lcd.print("Temp: ");              //display the Humidity on the LCD1602
    lcd.print(int(DHT.temperature));
    lcd.print(" C");
    lcd.setCursor(0, 1);              //set the cursor to column 0, line 0
    lcd.print("Humi: ");              //display the Humidity on the LCD1602
    lcd.print(int(DHT.humidity));
    lcd.print(" %");
  } else {
    lcd.clear();
    lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
    lcd.print("DHT11 Data error");
  }
  delay(1000);
}
```

First, include the library function header file.

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

Initialize IIC-LCD1602 and turn ON the backlight.

```
9   lcd.init();                      // LCD driver initialization
10  lcd.backlight();                 // Open the backlight
```

Obtain the temperature and humidity data of the DHT11. If the data is obtained successfully, print it to the LCD1602 screen.

```
15  int chk = DHT.read11(dhtPin);
16  if (chk == DHTLIB_OK) {
17    lcd.clear();
18    lcd.setCursor(0, 0);              //set the cursor to column 0, line 1
19    lcd.print("Temp: ");              //display the Humidity on the LCD1602
20    lcd.print(int(DHT.temperature));
21    lcd.print(" C");
22    lcd.setCursor(0, 1);              //set the cursor to column 0, line 0
23    lcd.print("Humi: ");              //display the Humidity on the LCD1602
24    lcd.print(int(DHT.humidity));
25    lcd.print(" %");
26  }
```

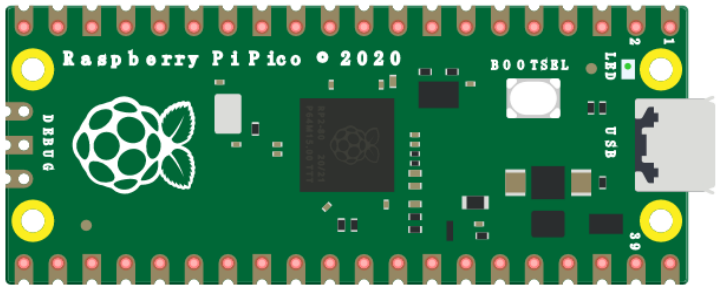
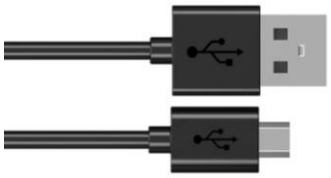
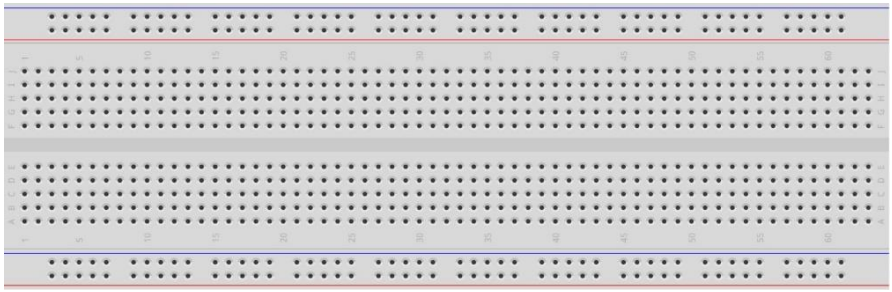
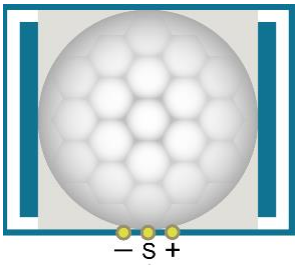
Chapter 26 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 26.1 Infrared Motion Detector with LED Indicator

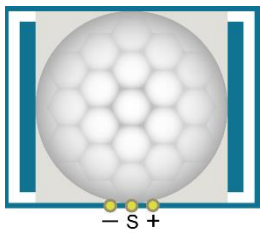
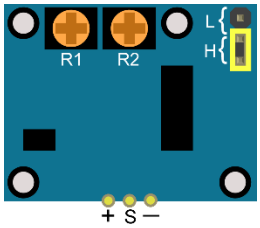
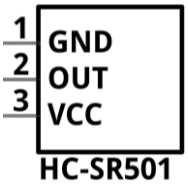
In this project, we will make a Motion Detector, with the human body infrared pyroelectric sensors. When someone is in close proximity to the Motion Detector, it will automatically light up and when there is no one close by, it will be out.
This Infrared Motion Sensor can detect the infrared spectrum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

Component List

<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>HC SR501 x1
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td>Jumper
  </td></tr>
</table>

Component Knowledge

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

Top	Bottom	Schematic
		

Description:

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

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

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

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

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

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

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

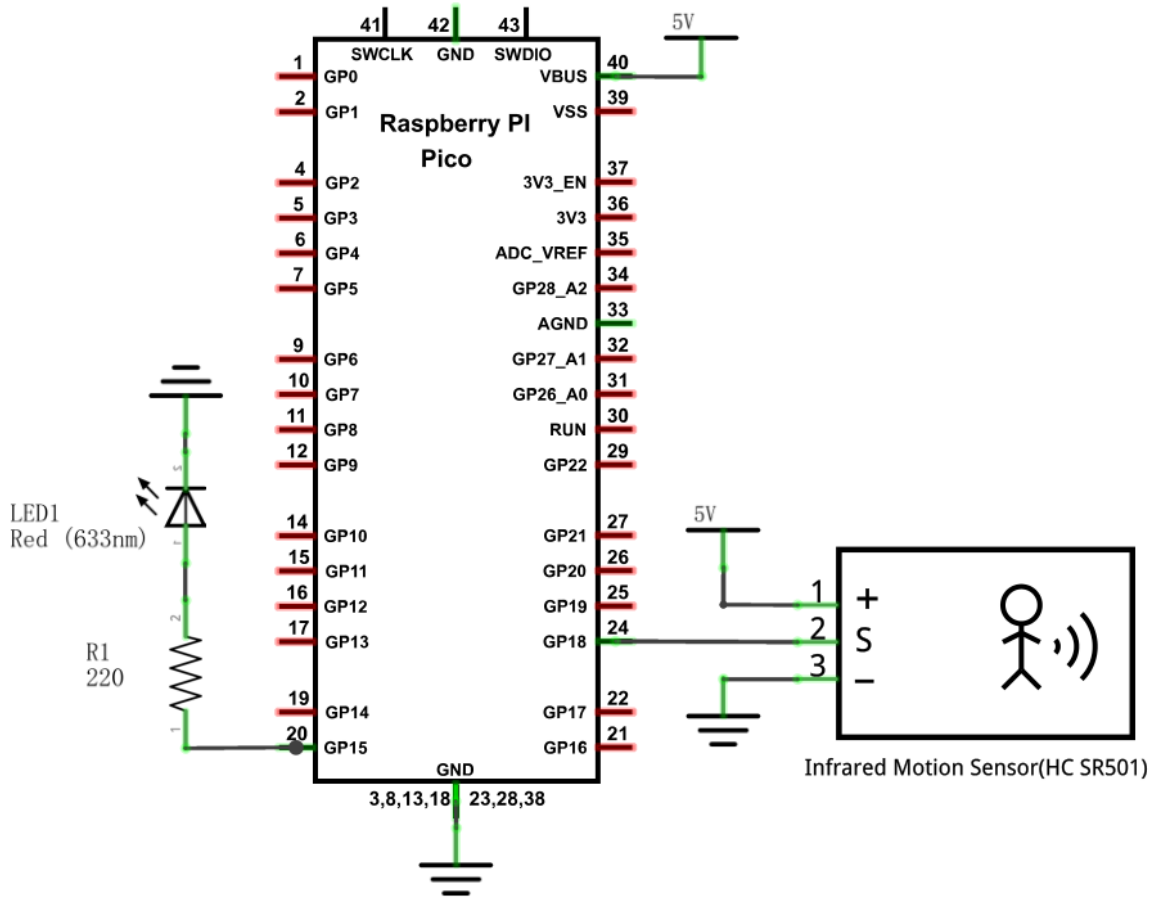
One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency).

Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

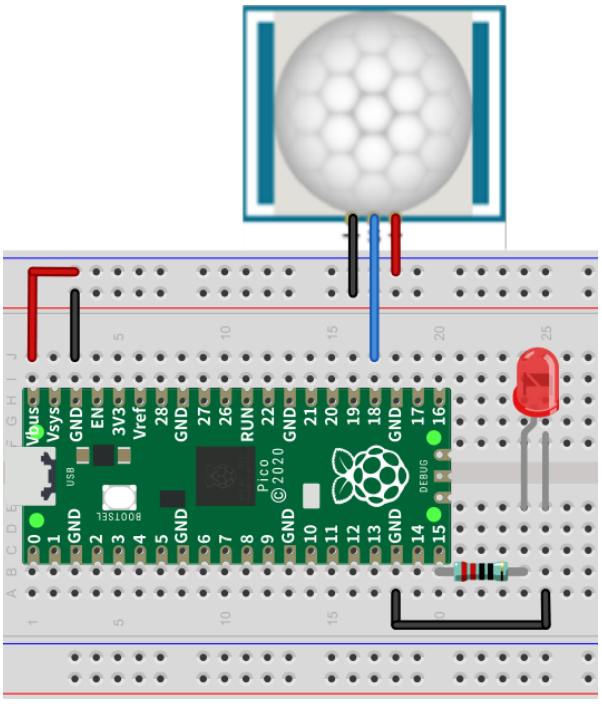
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



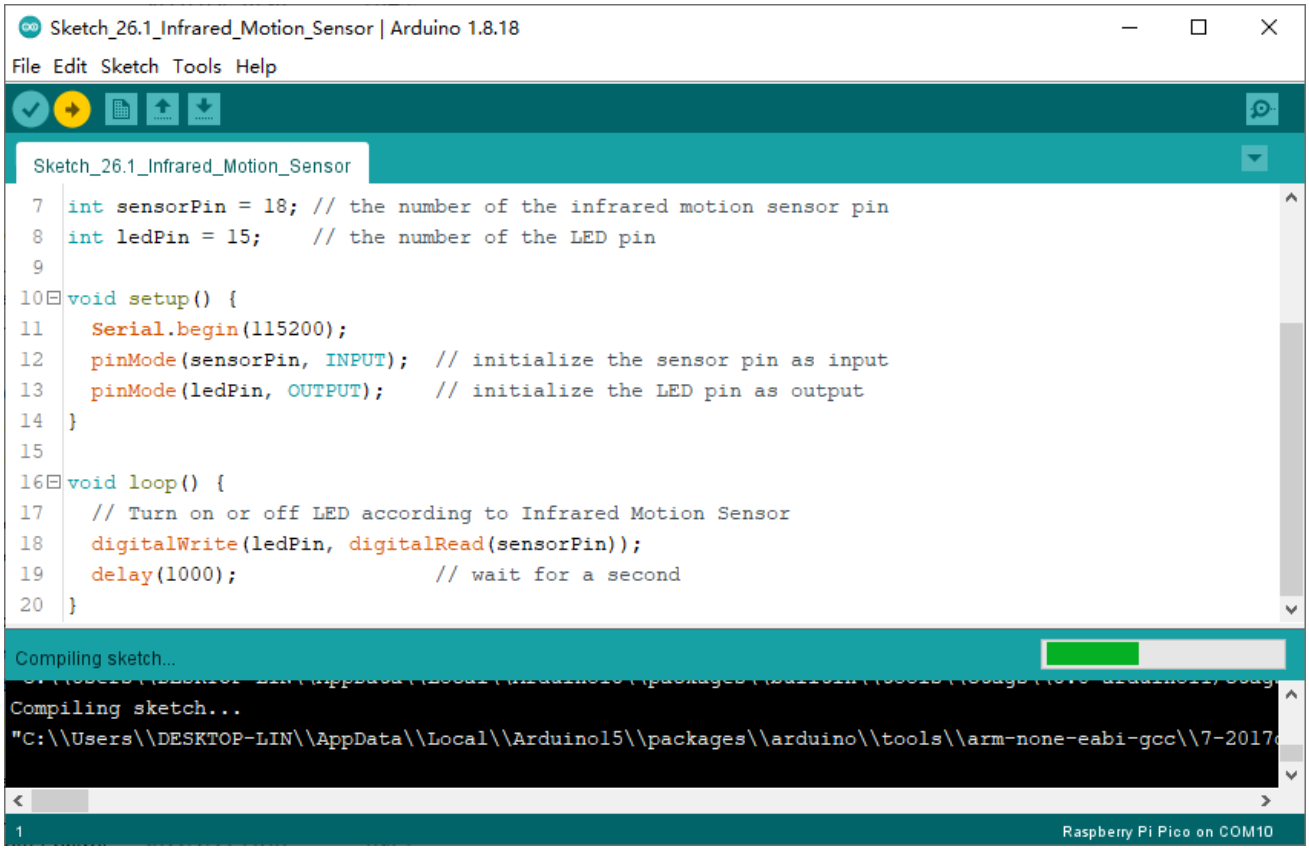
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Sketch

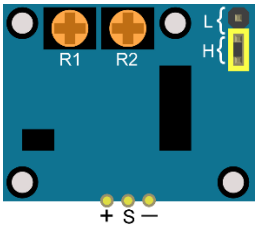
In this project, we will use the infrared motion sensor to trigger an LED, essentially making the infrared motion sensor act as a motion switch. Therefore, the code is very similar to the earlier project "push button switch and LED". The difference is that, when infrared motion sensor detects change,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

Sketch_26.1_Infrared_Motion_Sensor



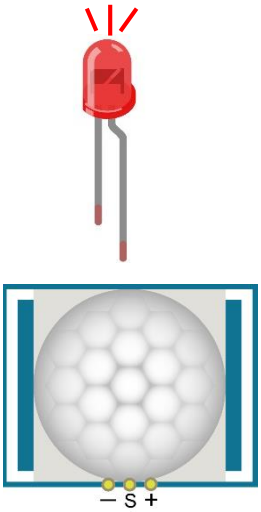
Verify and upload the code, and put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the infrared motion sensor and observe whether the LED turns ON or OFF automatically.

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

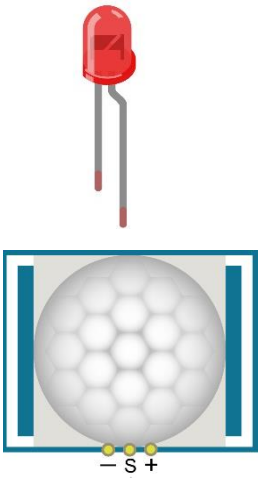


Apart from that, you can also use this sensor to control some other modules to implement different functions by reediting the code, such as the induction lamp, induction door.

Move to the Infrared Motion Sensor



Move away from the Infrared Motion Sensor



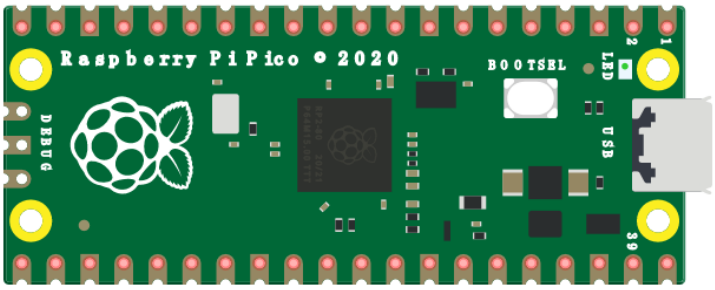
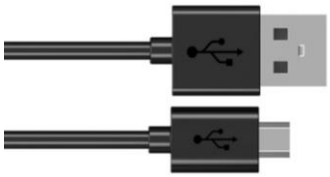
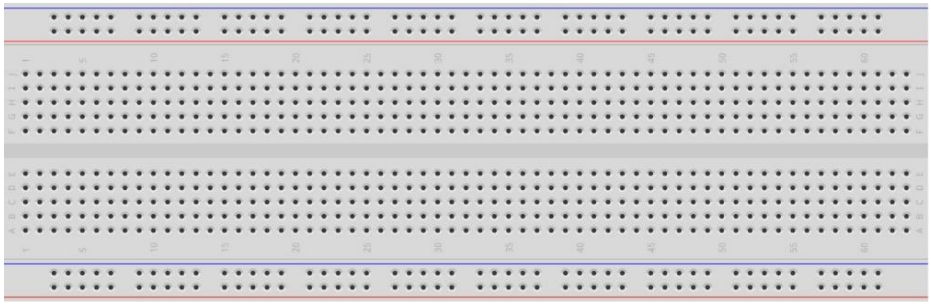
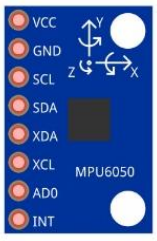
Chapter 27 Attitude Sensor MPU6050

In this chapter, we will learn about an MPU6050 Attitude Sensor which integrates an Accelerometer and Gyroscope.

Project 27.1 Read an MPU6050 Sensor Module

In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor.

Component List

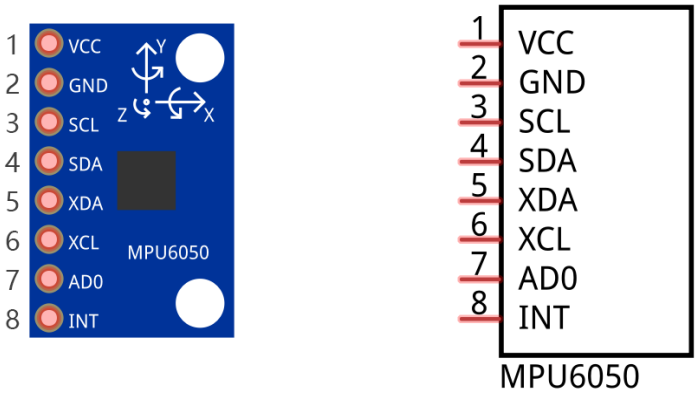
Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	MPU6050 x1 

Component Knowledge

MPU6050

MPU6050 Sensor Module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, a 3-axis Accelerometer and a DMP (Digital Motion Processor) all in a small package. The settings of the Accelerometer and Gyroscope of MPU6050 can be changed. A precision wide range digital temperature sensor is also integrated to compensate data readings for changes in temperature, and temperature values can also be read. The MPU6050 Module follows the I2C communication protocol and the default address is 0x68.

MPU6050 is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.



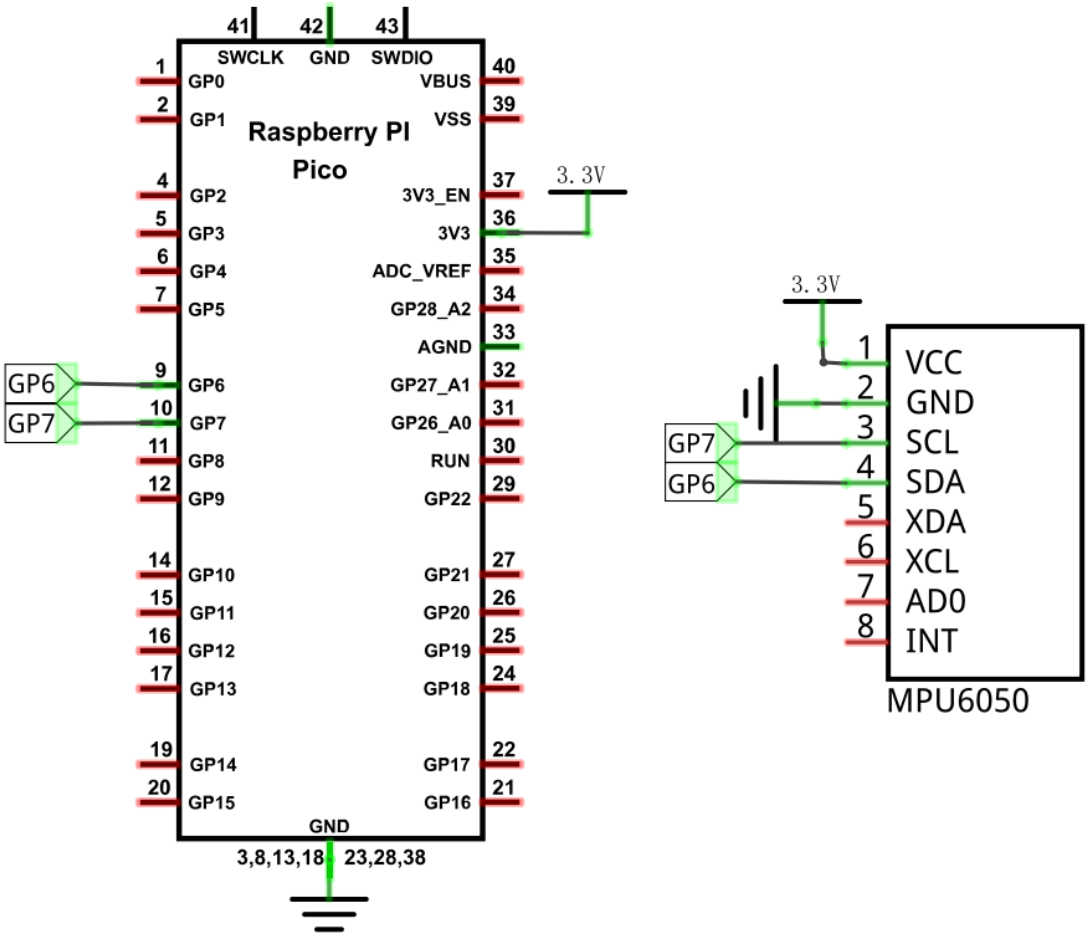
The port description of the MPU6050 module is as follows:

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

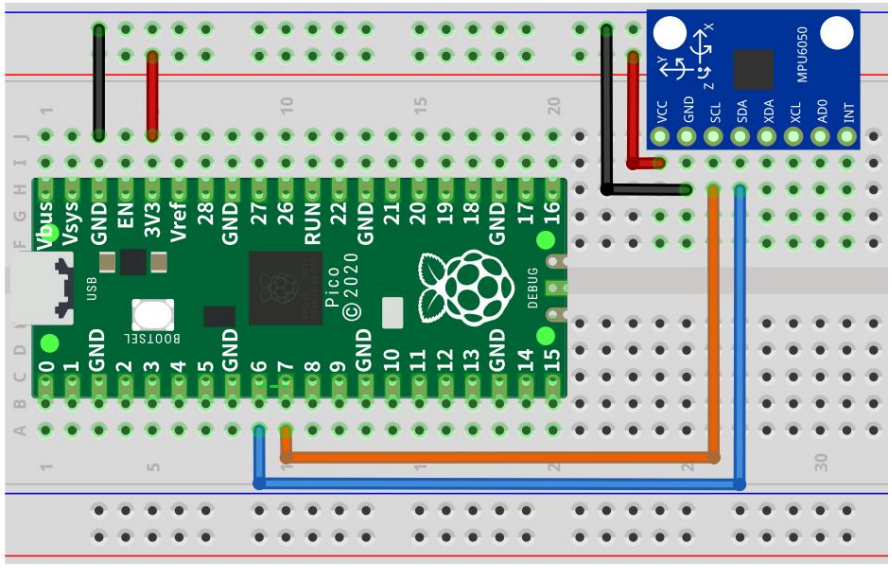
For more details, please refer to datasheet.

Circuit

Schematic diagram



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



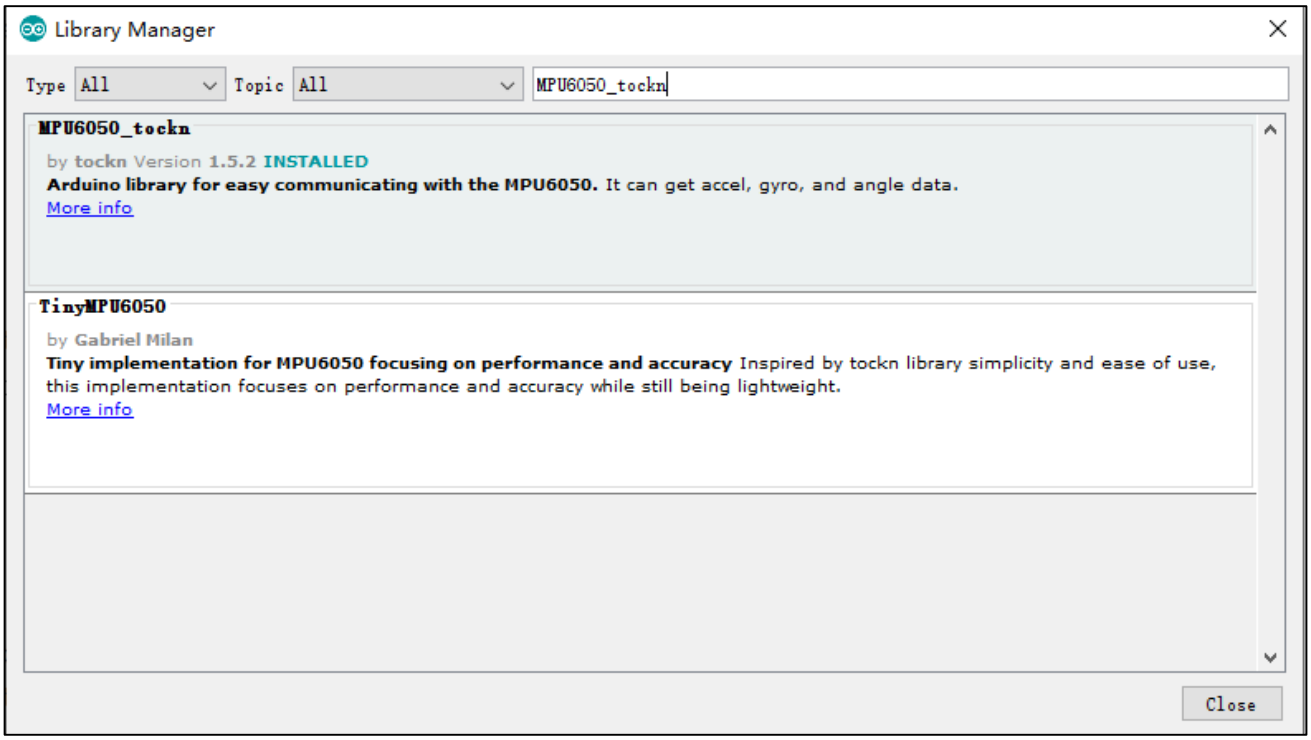
Sketch

How to install the library

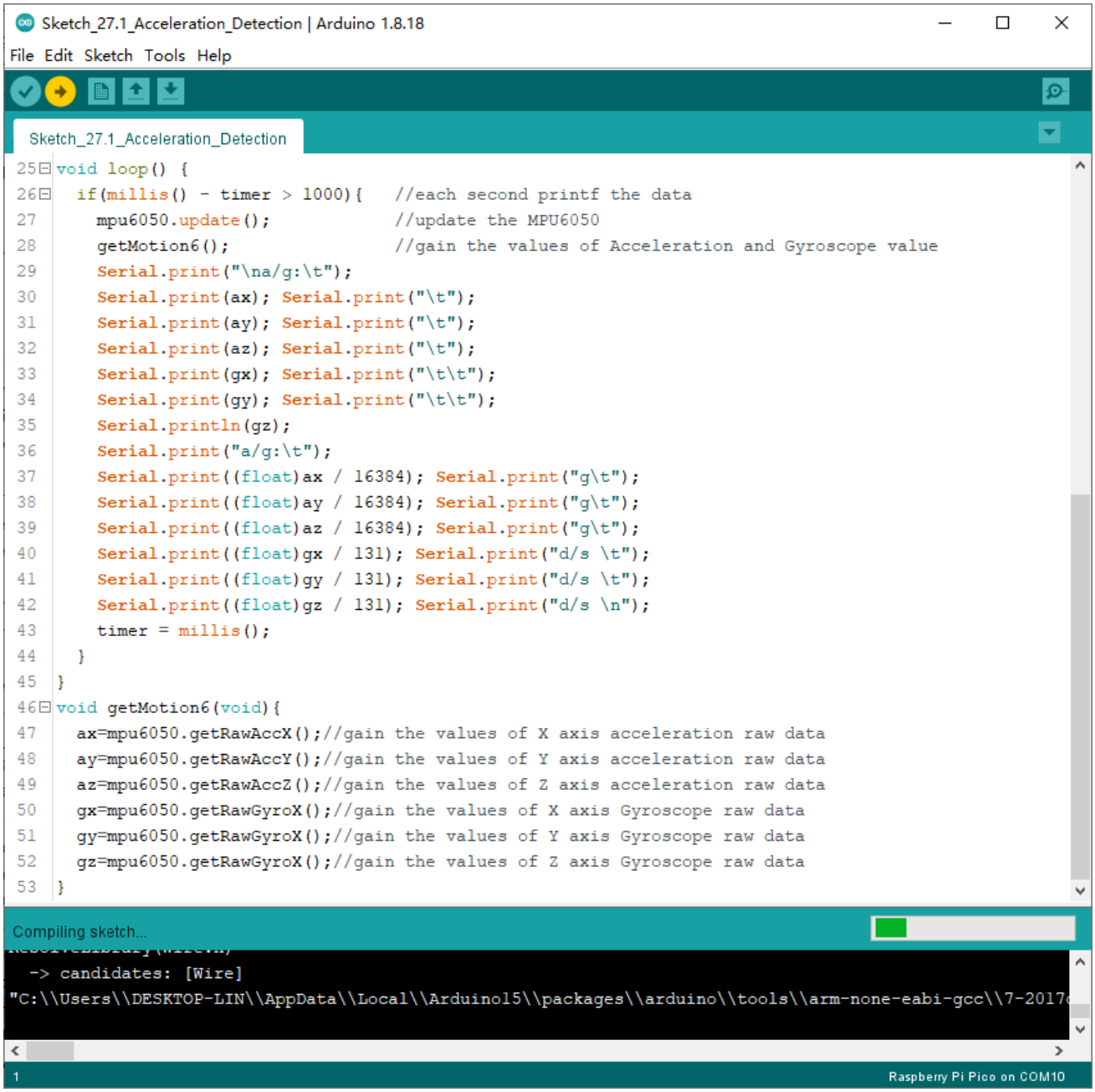
In this project, we will read the acceleration data and gyroscope data of MPU6050, and print them out.

We use the third party library MPU6050_tockn. If you haven't installed it yet, please do so now. The steps to add third-party Libraries are as follows: open arduino->Sketch->Include library-> Manage libraries. Enter " MPU6050_tockn " in the search bar and select " MPU6050_tockn " for installation.

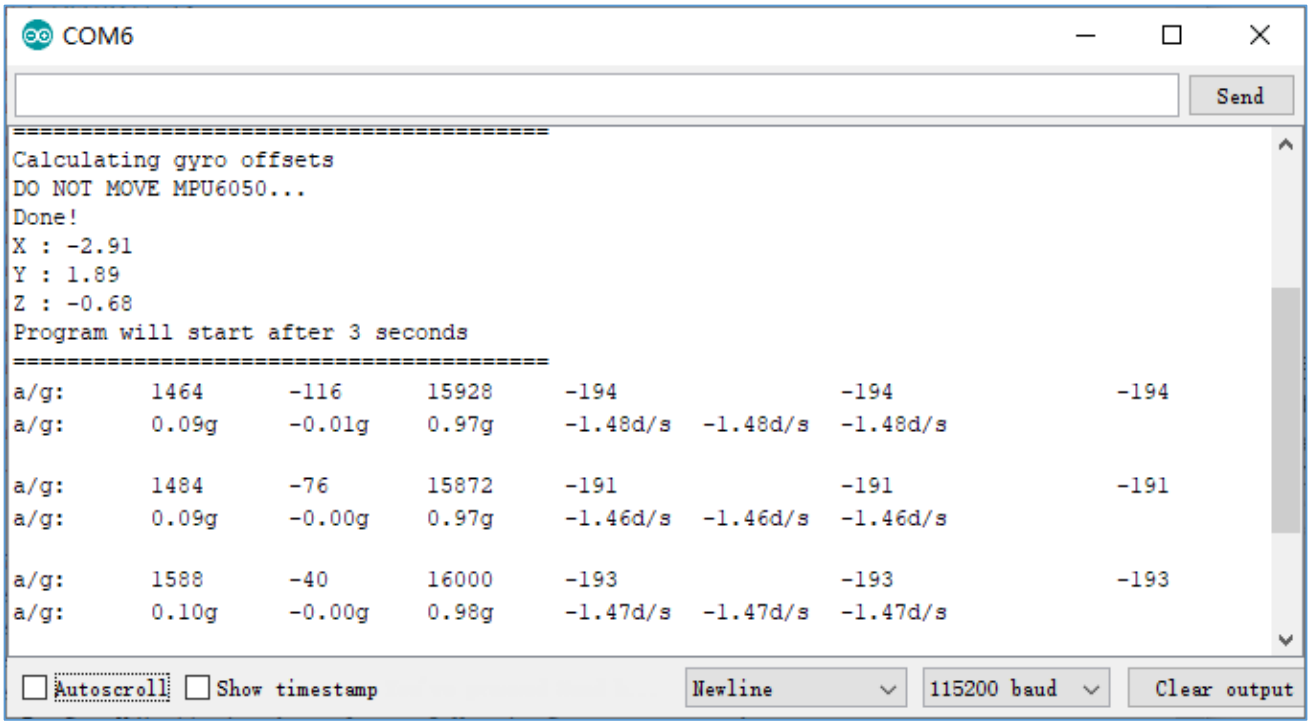
Refer to the following operations:



Sketch_27.1_Acceleration_Detection



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



The following is the program code:

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

MbedI2C iic(6,7);

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

long timer = 0;

void setup() {
  iic.begin();
  Serial.begin(115200);
  mpu6050.begin();                  //initialize the MPU6050
  mpu6050.calcGyroOffsets(true); //get the offsets value
}

void loop() {
  if(millis() - timer > 1000){   //each second printf the data
    mpu6050.update();            //update the MPU6050
    getMotion6();                //gain the values of Acceleration and Gyroscope value
    Serial.print("\na/g:\t");
```

```
24        Serial.print(ax); Serial.print("\t");
25        Serial.print(ay); Serial.print("\t");
26        Serial.print(az); Serial.print("\t");
27        Serial.print(gx); Serial.print("\t\t");
28        Serial.print(gy); Serial.print("\t\t");
29        Serial.println(gz);
30        Serial.print("a/g:\t");
31        Serial.print((float)ax / 16384); Serial.print("g\t");
32        Serial.print((float)ay / 16384); Serial.print("g\t");
33        Serial.print((float)az / 16384); Serial.print("g\t");
34        Serial.print((float)gx / 131); Serial.print("d/s \t");
35        Serial.print((float)gy / 131); Serial.print("d/s \t");
36        Serial.print((float)gz / 131); Serial.print("d/s \n");
37        timer = millis();
38      }
39    }
40    void getMotion6(void){
41      ax=mpu6050.getRawAccX();//gain the values of X axis acceleration raw data
42      ay=mpu6050.getRawAccY();//gain the values of Y axis acceleration raw data
43      az=mpu6050.getRawAccZ();//gain the values of Z axis acceleration raw data
44      gx=mpu6050.getRawGyroX();//gain the values of X axis Gyroscope raw data
45      gy=mpu6050.getRawGyroX();//gain the values of Y axis Gyroscope raw data
46      gz=mpu6050.getRawGyroX();//gain the values of Z axis Gyroscope raw data
47    }
```

Two library files "**MPU6050_tockn.h**" and "**Wire.h**" are used in the code and will be compiled with others.

Class MPU6050 is used to operate the MPU6050. When using it, please instantiate an object first.

Class MbedI2C is used to operate the IIC. When using it, please instantiate an object first.

```
4   MbedI2C iic(6,7);
6   MPU6050 mpu6050(iic);//Attach the IIC
```

In the setup function, IIC and MPU6050 are initialized and the offset difference of MPU6050 is obtained.

```
12  void setup(){
13    iic.begin();
14    Serial.begin(115200);
15    mpu6050.begin();                  //initialize the MPU6050
16    mpu6050.calcGyroOffsets(true); //get the offsets value
17  }
```

The getMotion6 function is used to obtain the x, y, z axis acceleration raw data and the Gyroscope raw data.

```
40  void getMotion6(void){
41    ax=mpu6050.getRawAccX();//gain the values of X axis acceleration raw data
42    ay=mpu6050.getRawAccY();//gain the values of Y axis acceleration raw data
43    az=mpu6050.getRawAccZ();//gain the values of Z axis acceleration raw data
44    gx=mpu6050.getRawGyroX();//gain the values of X axis Gyroscope raw data
45    gy=mpu6050.getRawGyroX();//gain the values of Y axis Gyroscope raw data
46    gz=mpu6050.getRawGyroX();//gain the values of Z axis Gyroscope raw data
```

```
47  }
```

Finally, the original data of the gyroscope is updated and acquired every second, and the original data, the processed acceleration and angular velocity data are printed out through the serial port.

```
19  void loop() {
20    if(millis() - timer > 1000){          //each second print the data
21      mpu6050.update();                   //update the MPU6050
22      getMotion6();                        //gain the values of Acceleration and Gyroscope value
23      Serial.print("\na/g:\t");
24      Serial.print(ax); Serial.print("\t");
25      Serial.print(ay); Serial.print("\t");cc
26      Serial.print(az); Serial.print("\t");
27      Serial.print(gx); Serial.print("\t\t");
28      Serial.print(gy); Serial.print("\t\t");
29      Serial.println(gz);
30      Serial.print("a/g:\t");
31      Serial.print((float)ax / 16384); Serial.print("g\t");
32      Serial.print((float)ay / 16384); Serial.print("g\t");
33      Serial.print((float)az / 16384); Serial.print("g\t");
34      Serial.print((float)gx / 131); Serial.print("d/s \t");
35      Serial.print((float)gy / 131); Serial.print("d/s \t");
36      Serial.print((float)gz / 131); Serial.print("d/s \n");
37      timer = millis();
38    }
39  }
```

Reference

Class MPU6050

This is a class library used to operate MPU6050, which can directly read and set MPU6050. Here are some member functions:

MPU6050 mpu6050(Wire): Associate MPU6050 with IIC.

begin(): Initialize the MPU6050.

calcGyroOffsets(true): If the parameter is true, get the gyro offset and automatically correct the offset. If the parameter is false, the offset value is not obtained and the offset is not corrected.

getRawAccX(): Gain the values of X axis acceleration raw data.

getRawAccY(): Gain the values of Y axis acceleration raw data.

getRawAccZ(): Gain the values of Z axis acceleration raw data.

getRawGyroX(): Gain the values of X axis Gyroscope raw data.

getRawGyroY(): Gain the values of Y axis Gyroscope raw data.

getRawGyroZ(): Gain the values of Z axis Gyroscope raw data.

getTemp(): Gain the values of MPU6050'temperature data.

update(): Update the MPU6050. If the updated function is not used, the IIC will not be able to retrieve the new data.

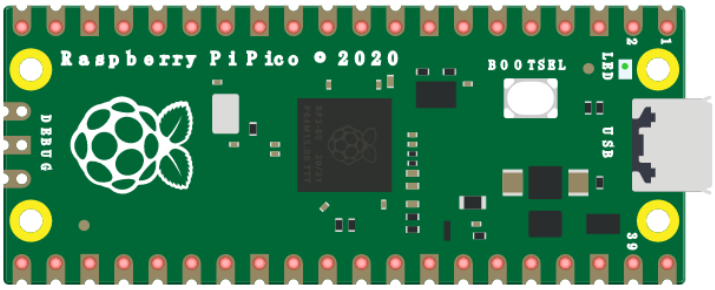
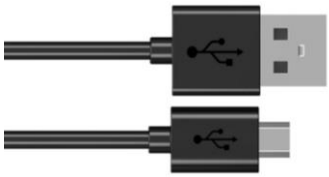
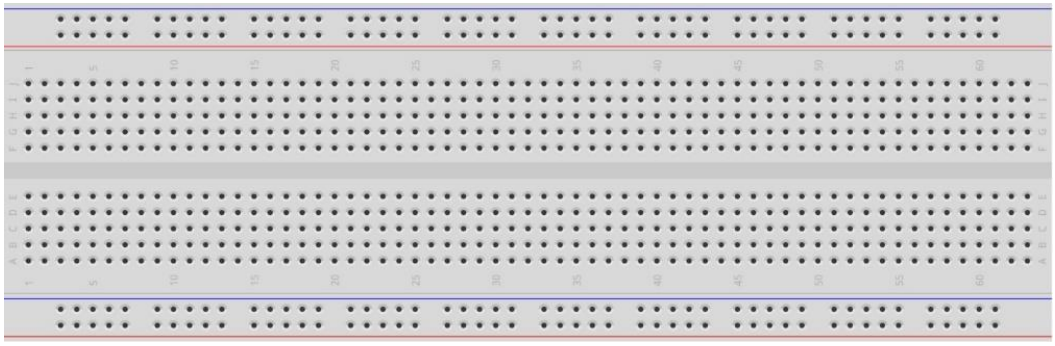
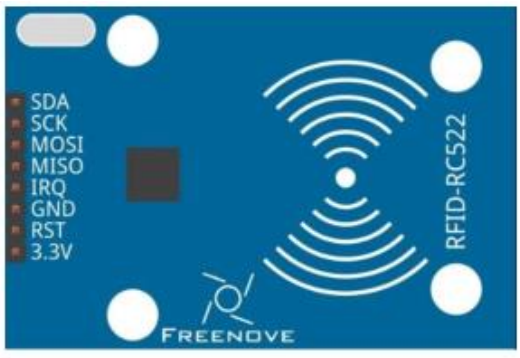
Chapter 28 RFID

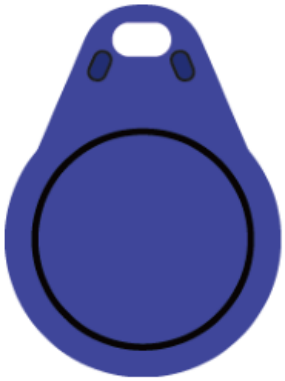
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 28.1 RFID read UID

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
Jumper 	RFID Module(RC522) x1 

Mifare1 S50 Standard card x1	Mifare1 S50 Non-standard card x1
	

Component Knowledge

RFID

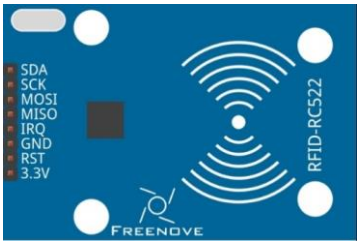
RFID (Radio Frequency Identification) is a wireless communication technology. A complete RFID system is generally composed of the responder and reader. Generally, we use tags as responders, and each tag has a unique code, which is attached to the object to identify the target object. The reader is a device for reading (or writing) tag information.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products. And Passive RFID products are the earliest, the most mature and most widely used products in the market among others. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of these belong to close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522 RFID Module

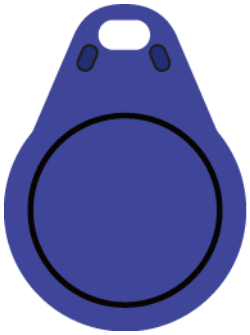
The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.
The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.
This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.



Mifare1 S50 Card

Mifare1 S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for this kit are shown below.

Mifare1 S50 Standard card



Mifare1 S50 Non-standard card

The Mifare1 S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according to absolute address, from 0 to 63). And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control which are put in the last block of each sector, and the block is also known as sector trailer, that is Block 3 in each sector. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the vendor code, which has been solidified and can't be changed, and the card serial number is stored here. In addition to the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:
(1) used as general data storage and can be operated for reading and writing.
(2) used as data value, and can be operated for initializing the value, adding value, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, 4-byte access control and 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally A0A1A2A3A4A5 for password A, B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

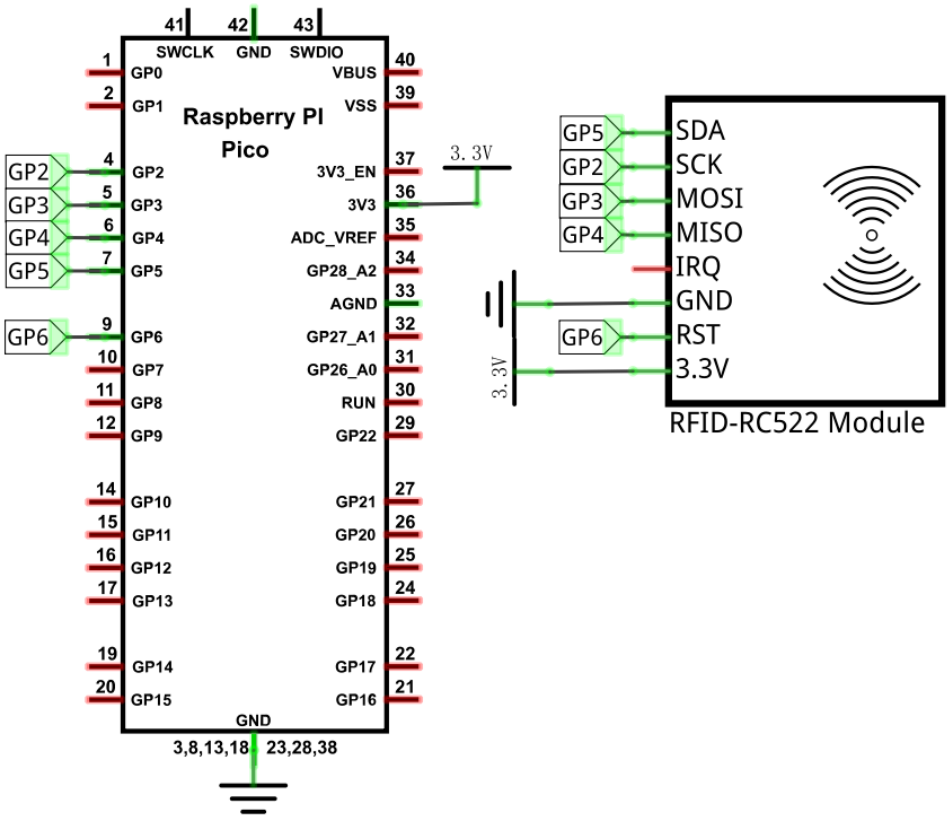
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read. If you choose to verify password A and then you forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

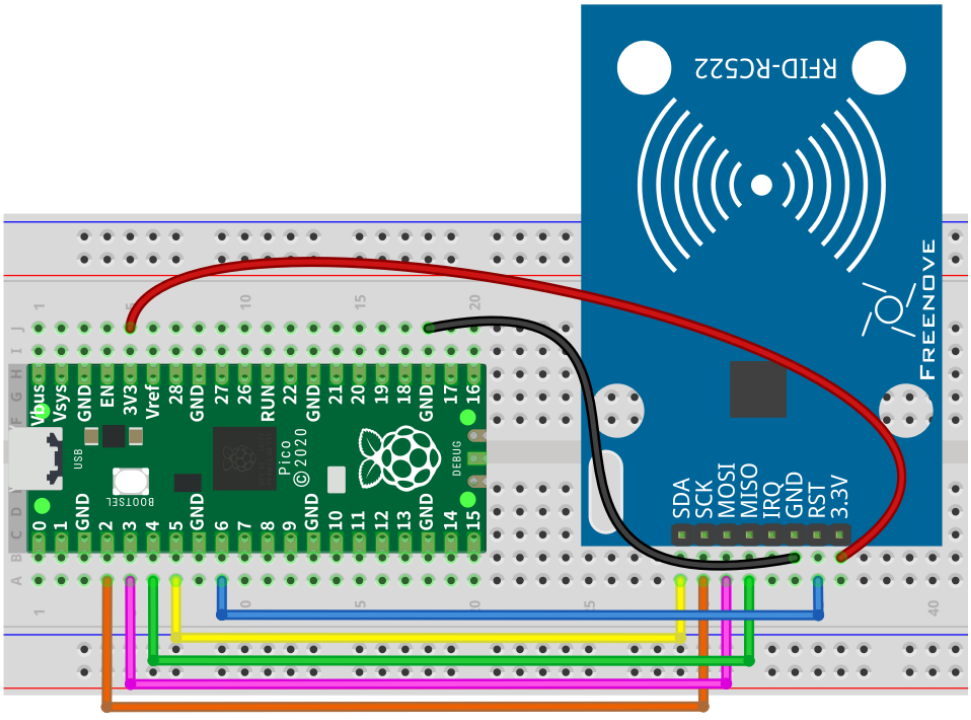
Circuit

The connection of control board and RFID module is shown below.

Schematic diagram



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



Sketch

Sketch_28.1_RFID_Read_UID

Before writing code, we need to import the library needed.

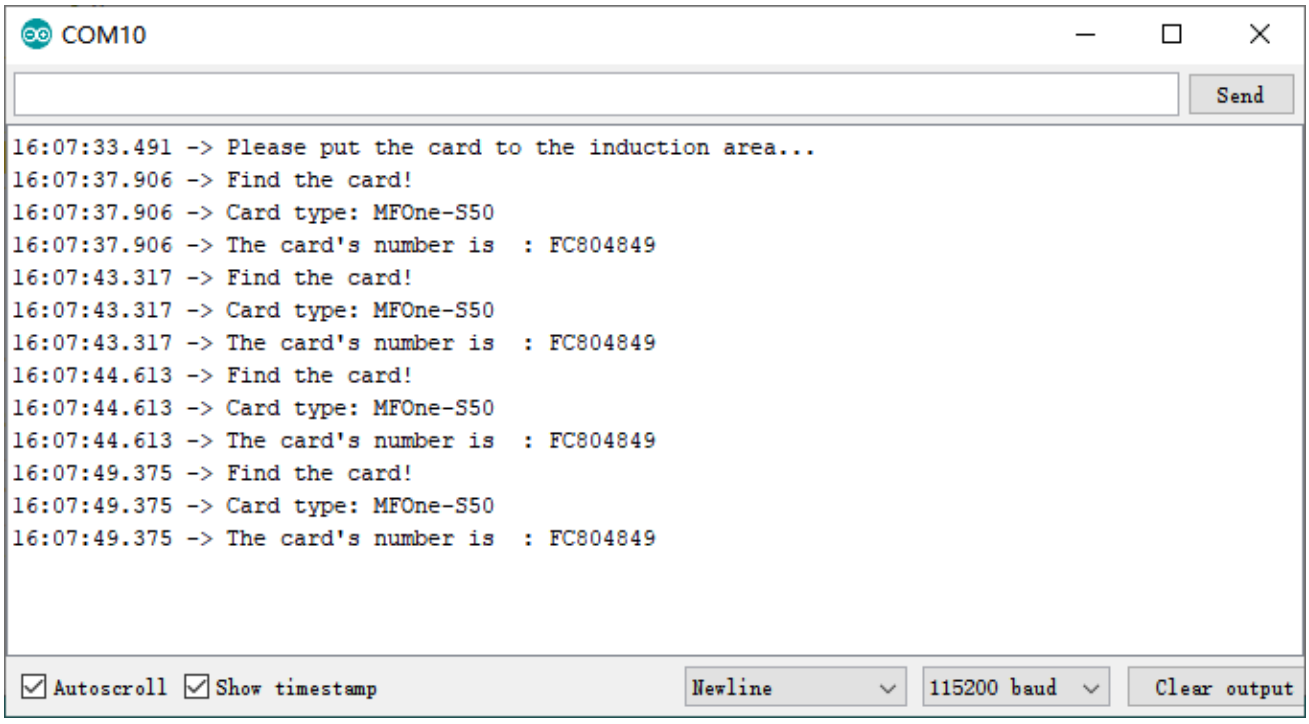
Click “Add .ZIP Library...” and then find **RFID.zip** in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library make it easy to operate RFID module.

This sketch will read the unique ID number (UID) of the card, recognize the type of the card and display the information through serial port.

```
#include <SPI.h>
#include <RFID.h>

RFID rfid(5, 6);
unsigned char status;
unsigned char str[MAX_LEN];  //MAX_LEN is 16: size of the array

void setup()
{
  Serial.begin(115200);
  SPI.begin();
  rfid.init(); //initialization
  Serial.println("Please put the card to the induction area...");
}

void loop()
{
  //Search card, return card types
  if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
    Serial.println("Find the card!");
    // Show card type
    ShowCardType(str);
    //Anti-collision detection, read card serial number
    if (rfid.anticoll(str) == MI_OK) {
      Serial.print("The card's number is  : ");
      //Display card serial number
      for (int i = 0; i < 4; i++) {
        Serial.print(0x0F & (str[i] >> 4), HEX);
        Serial.print(0x0F & str[i], HEX);
      }
      Serial.println("");
    }
    //card selection (lock card to prevent redundant read, removing the line will make the sketch read cards continually)
```

```
34        rfid.selectTag(str);
        }
35      rfid.halt();  // command the card to enter sleeping state
36    }
37    void ShowCardType(unsigned char * type)
38    {
39      Serial.print("Card type: ");
40      if (type[0] == 0x04 && type[1] == 0x00)
41        Serial.println("MFOne-S50");
42      else if (type[0] == 0x02 && type[1] == 0x00)
43        Serial.println("MFOne-S70");
44      else if (type[0] == 0x44 && type[1] == 0x00)
45        Serial.println("MF-UltraLight");
46      else if (type[0] == 0x08 && type[1] == 0x00)
47        Serial.println("MF-Pro");
48      else if (type[0] == 0x44 && type[1] == 0x03)
49        Serial.println("MF Desire");
50      else
51        Serial.println("Unknown");
52    }
```

After verifying and uploading the code, open the serial monitor and make a card approach the sensing area of RFID module. Then serial monitor will display the displacement ID number and the type of the card. If the induction time is too short, it may lead to incomplete-information display.



After including the RFID library, we need to construct a RFID class object before using the function in RFID library. Its constructor needs to be written to two pins, respectively to the SDA pin and the RST pin.

```
4   RFID rfid(5, 6);
```

In setup, initialize the serial port, SPI and RFID.

```
10    Serial.begin(115200);
11    SPI.begin();
12    rfid.init(); //initialization
```

In loop(), use findCard() waiting for the card approaching. Once it detects card contact, this function will return MI_OK and save the card type data in parameter str, and then enter the if statement.

```
19    if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
```

After entering if statement, call the sub function ShowCardType(). Then determine the type of the card according to the content of STR and print the type out through the serial port.

```
22      ShowCardType(str);
```

Then use the.anticoll() to read UID of the card and use serial port to print it out.

```
24      if (rfid.anticoll(str) == MI_OK) {
25        Serial.print("The card's number is  : ");
26        //Display card serial number
27        for (int i = 0; i < 4; i++) {
28          Serial.print(0x0F & (str[i] >> 4), HEX);
29          Serial.print(0x0F & str[i], HEX);
30        }
31        Serial.println("");
32      }
```

Project 28.2 Read and write

In this project, we will do reading and writing operations to the card.

Component list

Same with last section.

Circuit

Same with last section.

Sketch

Sketch_28.2_RFID_Read_And_Write

In this sketch, first read the data in particular location of the S50 M1 Card, then write data in that position and read it out. Display these data through the serial port.

```
#include <SPI.h>
#include <RFID.h>

//pin5:pin of card reader SDA.  pin6:pin of card reader RST
RFID rfid(5, 6);

// 4-byte card serial number, the fifth byte is check byte
unsigned char serNum[5];
unsigned char status;
unsigned char str[MAX_LEN];
unsigned char blockAddr;        //Select the operation block address: 0 to 63

// Write card data you want(within 16 bytes)
unsigned char writeDate[16] = "WelcomeFreenove";

// The A password of original sector: 16 sectors; the length of password in each sector is 6 bytes.
unsigned char sectorKeyA[16][16] = {
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
```

```
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
  { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
};

void setup()
{
  Serial.begin(115200);
  SPI.begin();
  rfid.init();
  Serial.println("Please put the card to the induction area...");
}

void loop()
{
  //find the card
  rfid.findCard(PICC_REQIDL, str);
  //Anti-collision detection, read serial number of card
  if (rfid.anticoll(str) == MI_OK) {
    Serial.print("The card's number is  : ");
    //print the card serial number
    for (int i = 0; i < 4; i++) {
      Serial.print(0x0F & (str[i] >> 4), HEX);
      Serial.print(0x0F & str[i], HEX);
    }
    Serial.println("");
    memcpy(rfid.serNum, str, 5);
  }
  //select card and return card capacity (lock the card to prevent multiple read and written)
  rfid.selectTag(rfid.serNum);
  //first, read the data of data block 4
  readCard(4);
  //write data(within 16 bytes) to data block
  writeCard(4);
  //then read the data of data block again
  readCard(4);
```

```
67
68    rfid.halt();
69  }
70
71  //write the card
72  void writeCard(int blockAddr) {
73    if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
    MI_OK) //authenticate
74    {
75      //write data
76      //status = rfid.write((blockAddr/4 + 3*(blockAddr/4+1)), sectorKeyA[0]);
77      Serial.print("set the new card password, and can modify the data of the Sector: ");
78      Serial.println(blockAddr / 4, DEC);
79      // select block of the sector to write data
80      if (rfid.write(blockAddr, writeDate) == MI_OK) {
81        Serial.println("Write card OK!");
82      }
83    }
84  }
85
86  //read the card
87  void readCard(int blockAddr) {
88    if ( rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
    MI_OK) // authenticate
89    {
90      // select a block of the sector to read its data
91      Serial.print("Read from the blockAddr of the card : ");
92      Serial.println(blockAddr, DEC);
93      if ( rfid.read(blockAddr, str) == MI_OK) {
94        Serial.print("The data is (char type display): ");
95        Serial.println((char *)str);
96        Serial.print("The data is (HEX type display): ");
97        for (int i = 0; i < sizeof(str); i++) {
98          Serial.print(str[i], HEX);
99          Serial.print(" ");
100       }
101       Serial.println();
102     }
103   }
104 }
```

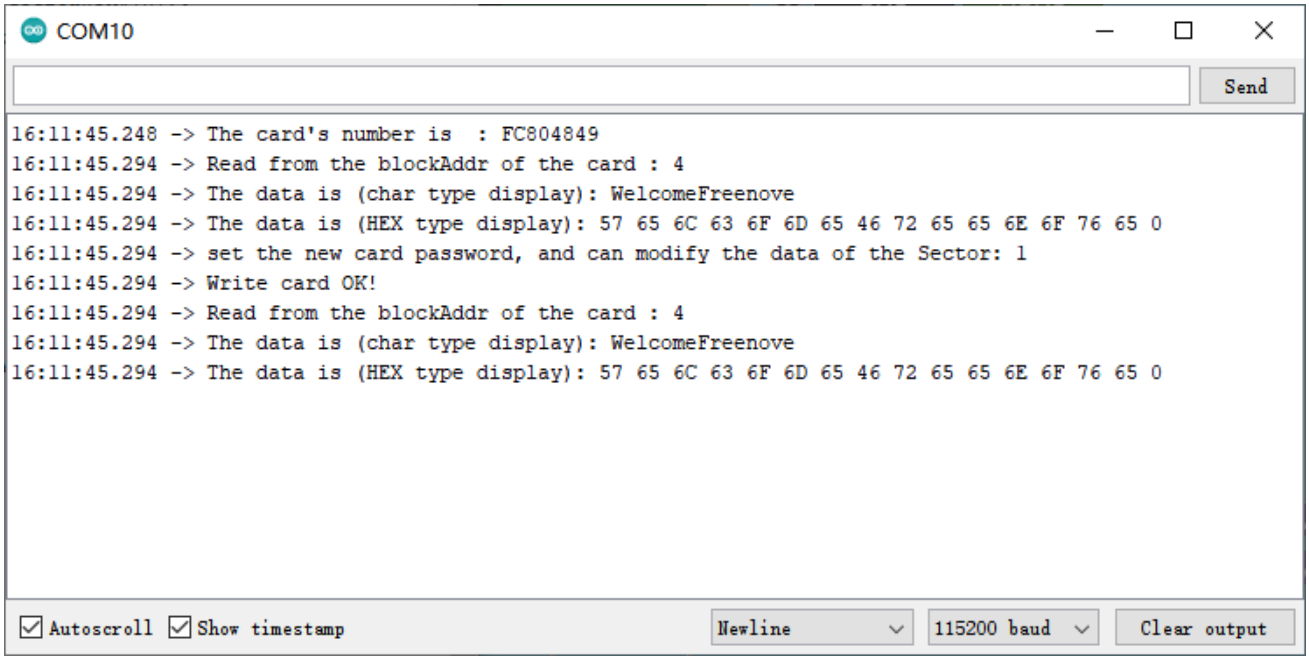
In the sub function of writeCard() and readCard(), we must first verify the password A, and then use the corresponding sub function to read and write. Here we do reading and writing operations to data block 0 (absolute NO.4) of the first sector.

```
73    if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
      MI_OK) //authenticate
```

In loop(), compare the contents of the data block NO.4 after written to the original contents.

```
61    //first, read the data of data block 4
62    readCard(4);
63    //write data(within 16 bytes) to data block
64    writeCard(4);
65    //then read the data of data block again
66    readCard(4);
```

After verifying and uploading the code, open the serial port monitor and make a card approach the sensing area of RFID module, then the serial port monitoring window will display displacement ID numbers of the card, the type of this card and the contents (before and after writing operation) of data block. If the induction time is too short, it may lead to incomplete information display.



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.