

Welcome

Thank you for choosing Freenove products!

How to Start

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.

Unpack

Before taking out all the parts, please read the file “Unpack.pdf”.

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

Attention

Pay attention to safety when using and storing this product:

- This product is not suitable for children under 12 years of age because of small parts and sharp parts.
- Minors should use this product under the supervision and guidance of adults.
- This product contains small and sharp parts. Do not swallow, prick and scratch to avoid injury.
- This product contains conductive parts. Do not hold them to touch power supply and other circuits.
- To avoid personal injury, do not touch parts rotating or moving while working.
- The wrong operation may cause overheat. Do not touch and disconnect the power supply immediately.
- Operate in accordance with the requirements of the tutorial. Fail to do so may damage the parts.
- Store this product in a dry and dark environment. Keep away from children.
- Turn off the power of the circuit before leaving.

About

Freenove provides open source electronic products and services.

Freenove is committed to helping customers learn programming and electronic knowledge, quickly implement product prototypes, realize their creativity and launch innovative products. Our services include:

- Kits for learning programming and electronics
- Kits compatible with Arduino®, Raspberry Pi®, micro:bit®, etc.
- Kits for robots, smart cars, drones, etc.
- Components, modules and tools
- Design and customization

To learn more about us or get our latest information, please visit our website:

<http://www.freenove.com>

Copyright

All the files provided in the ZIP file are released under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). You can find a copy of the license in the ZIP file.



It means you can use these files on your own derived works, in part or completely. But not for commercial use.

Freenove® brand and logo are trademarks of Freenove Creative Technology Co., Ltd. Must not be used without permission.



Other registered trademarks and their owners appearing in this document:

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

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

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

Contents

Welcome	i
Contents	1
Preface	1
Control Board.....	1
Chapter 0 Getting Ready (Important).....	4
Programming Software.....	4
Installation of Development Board Support Package	7
First Use	8
How to install the library.....	11
Chapter 1 LED Blink	13
Project 1.1 Control LED with Manual Button	13
Project 1.2 Control LED with Control Board	20
How to Use the Expanding GPIO Pins	26
Chapter 2 Two LEDs Blink.....	29
Project 2.1 Two LEDs Blink.....	29
Chapter 3 LED Bar Graph	36
Project 3.1 LED Bar Graph Display	36
Chapter 4 LED Blink Smoothly	44
Project 4.1 LEDs Emit Different Brightness.....	44
Project 4.2 LED Blinking Smoothly	50
Chapter 5 Control LED with Push Button Switch	52
Project 5.1 Control LED with Push Button Switch	52
Project 5.2 Change LED State with Push Button Switch.....	56
Chapter 6 Serial.....	58
Project 6.1 Send Data through Serial	58
Project 6.2 Receive Data through Serial Port	63
Project 6.3 Application of Serial	65
Chapter 7 Timer	68
Project 7.1 Serial print using timer	68
Project 7.2 Using timer to implement LED blinking	72
Chapter 8 ADC	74
Project 8.1 ADC.....	74
Project 8.2 Control LED by Potentiometer	80
Project 8.3 Control LED by Potentiometer	83
Chapter 9 RGB LED	87
Project 9.1 Control RGB LED through Potentiometer.....	87
Project 9.2 Multicolored LED	91
Chapter 10 Buzzer	94
Project 10.1 Active Buzzer.....	94
Project 10.2 Passive Buzzer	100
Chapter 11 DAC.....	104

Project 11.1 DAC	104
Chapter 12 RTC.....	109
Project 12.1 RTC	109
Chapter 13 Motor	112
Project 13.1 Control Motor by Relay.....	112
Project 13.2 Control Motor with L293D	119
Chapter 14 Servo.....	125
Project 14.1 Servo Sweep.....	125
Project 14.2 Control Servo with Potentiometer.....	129
Chapter 15 Temperature Sensor.....	132
Project 15.1 Detect the Temperature	132
Chapter 16 Joystick.....	136
Project 16.1 Joystick	136
Chapter 17 Acceleration sensor	140
Project 17.1 Acceleration Detection	140
Chapter 18 LED Matrix	148
Project 18.1 74HC595.....	148
Project 18.2 LED Matrix.....	154
Chapter 19 Onboard LED Matrix (WiFi Board).....	164
Project 19.1 LED Matrix.....	164
Project 19.2 LED Matrix.....	170
Project 19.3 Play the game with LED matrix.....	172
Project 19.3.1 LED Matrix Bounce Game	172
Project 19.3.2 LED Matrix Snake Game	176
Chapter 20 I2C LCD1602	183
Project 20.1 Display the String on I2C LCD1602	184
Project 20.2 I2C LCD1602 Clock.....	190
Chapter 21 Digital Display.....	200
Project 21.1 1-digit 7-segment Display.....	200
Project 21.2 4-digit 7-segment Display.....	205
Chapter 22 Stepper Motor.....	213
Project 22.1 Drive Stepper Motor.....	213
Chapter 23 Matrix Keypad	220
Project 23.1 Get Input Characters	220
Project 23.2 Combination Lock	226
Chapter 24 Vibration Switch.....	231
Project 24.1 Detect Vibration.....	231
Chapter 25 Infrared Remote	236
Project 25.1 Infrared Remote Control	236
Project 25.2 Control LED through Infrared Remote	243
Chapter 26 Temperature & Humidity Sensor.....	248
Project 26.1 Temperature & Humidity Sensor	248
Chapter 27 Infrared Motion Sensor	253
Project 27.1 Infrared Motion Sensor	253

Chapter 28 Ultrasonic Ranging.....	257
Project 28.1 Ultrasonic Ranging.....	257
Chapter 29 LEDpixel.....	264
Project 29.1 LEDpixel	264
Project 29.2 RainbowLight	271
Chapter 30 RFID.....	275
Project 30.1 RFID read UID	275
Project 30.2 Read and write.....	283
Chapter 31 WiFi Working Modes (WiFi Board).....	287
Project 31.1 Station mode.....	287
Project 31.2 AP mode.....	292
Chapter 32 TCP/IP (WiFi Board)	298
Project 32.1 As Client.....	298
Project 32.2 As Server.....	310
Chapter 33 Control LED with Web (WiFi Board).....	315
Project 33.1 Control the LED with Web.....	315
Chapter 34 Bluetooth (WiFi Board).....	323
Project 34.1 Bluetooth Low Energy Data Passthrough.....	323
Project 34.2 Control LED with Bluetooth	335
Chapter 35 USB HID	343
Project 35.1 Mouse control.....	343
Project 35.2 Keypad Control	347
Chapter 36 Soldering Circuit Board	351
Project 36.1 Solder a Buzzer	351
Project 36.2 Solder a Flowing Water Light	355
Other Components.....	360
What's Next?	362
Appendix	363
ASCII Table	363
Resistor Color Code	364

Preface

If you want to make some interesting projects or want to learn electronics and programming, this document will greatly help you.

Projects in this document usually contains two parts: the circuit and the code. No experience at all? Don't worry, this document will show you how to start from scratch.

If you encounter any problems, please feel free to send us an email, we will try our best to help you.

Support email: support@freenove.com

To complete these projects, you need to use a control board and software to program it, as well as some commonly used components.

Control Board

The control board is the core of a circuit. After programming, it can be used to control other components in the circuit to achieve intended functions.

There are multiple versions of Freenove control board. Your purchase may be one of the following:

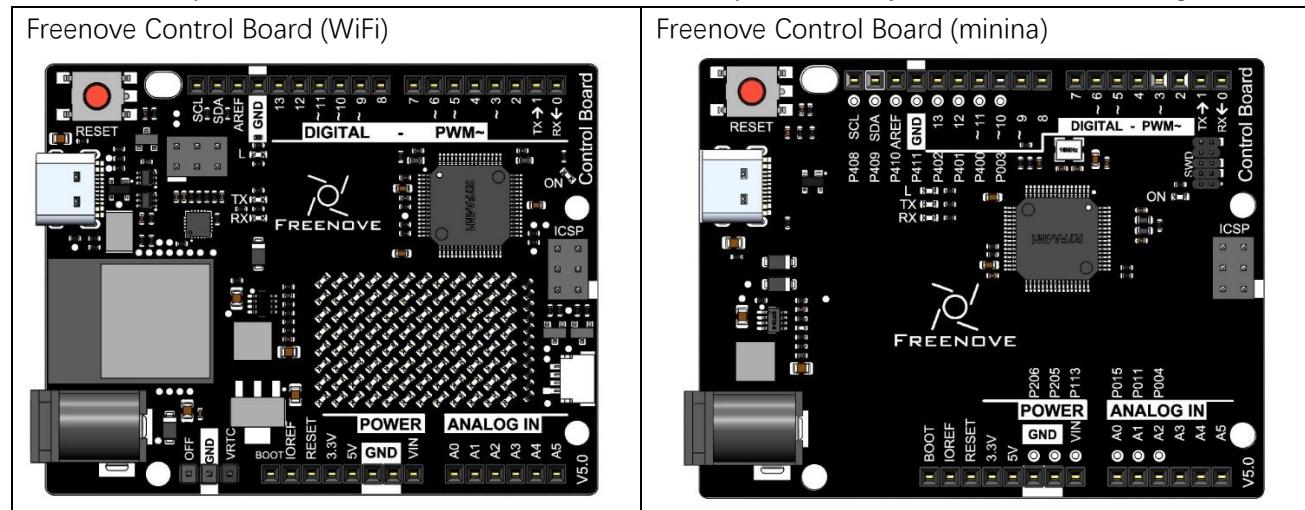
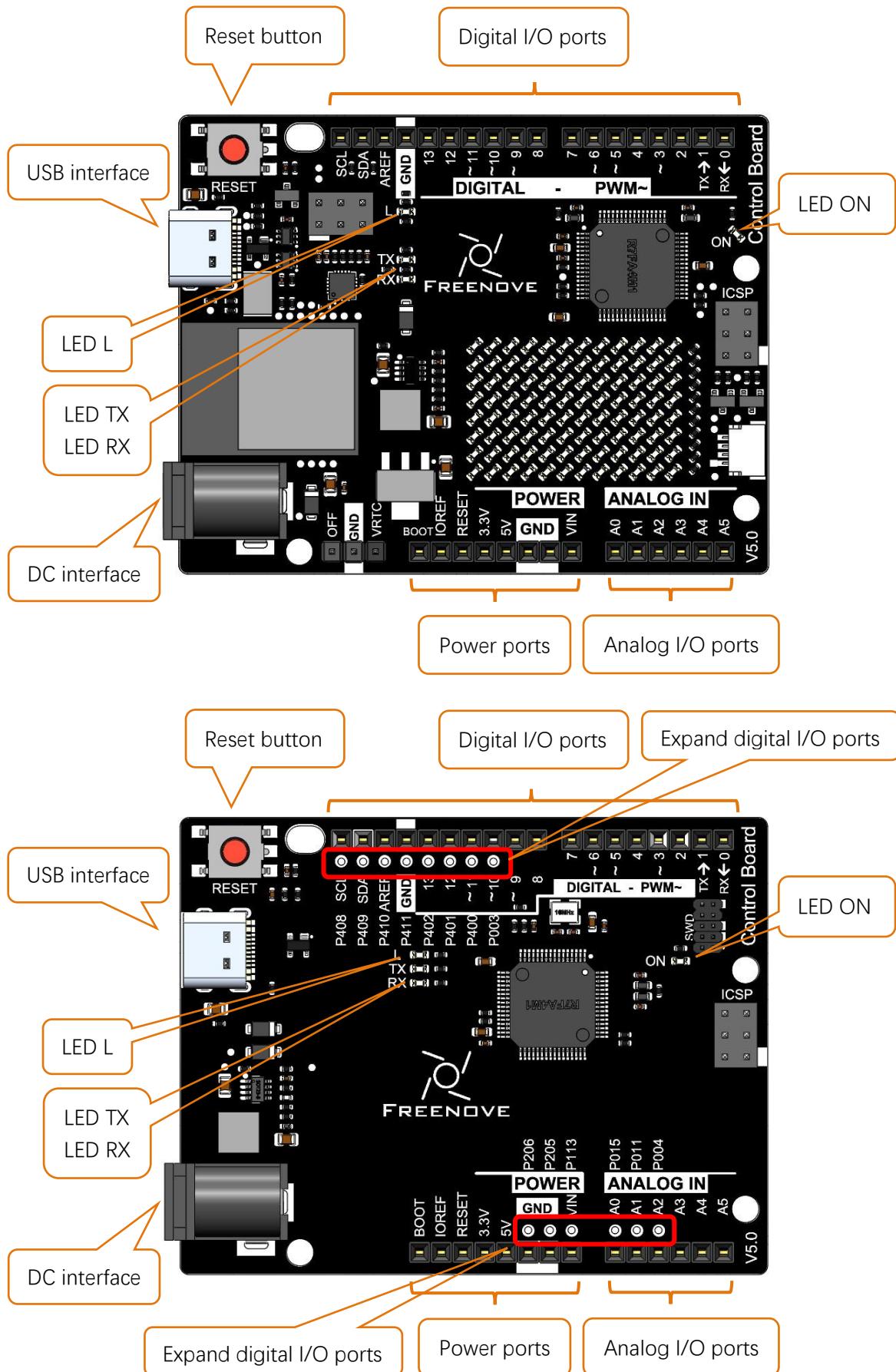


Diagram of the Freenove control board is shown below:



- Digital I/O ports is used to connect to other components or modules, to receive an input signal, or to send a control signal. Usually, we name it by adding a "D" in front of the number, such as D13 (pin 13).
- USB interface is used to provide power, upload code or communicate with PC.
- LED L is connected to digital I/O port 13 (pin 13).
- LED TX, RX is used to indicate the state of the serial communication.
- DC interface is connected DC power to provide power for the board.
- Power ports can provide power for electronic components and modules.
- Analog I/O ports can be used to measure analog signals.
- LED ON is used to indicate the power state.

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

We use Arduino® IDE to write and upload code for the control board, which is a free and open source.
(Arduino® is a trademark of Arduino LLC.)

Arduino IDE uses C/C++ programming language. Don't worry if you have never used it, because this document contains programming knowledge and detailed explanation of the code.

First, install Arduino IDE. Visit <https://www.arduino.cc/en/software>. Scroll down and find **Arduino IDE (2.3.X)**. Then select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer".



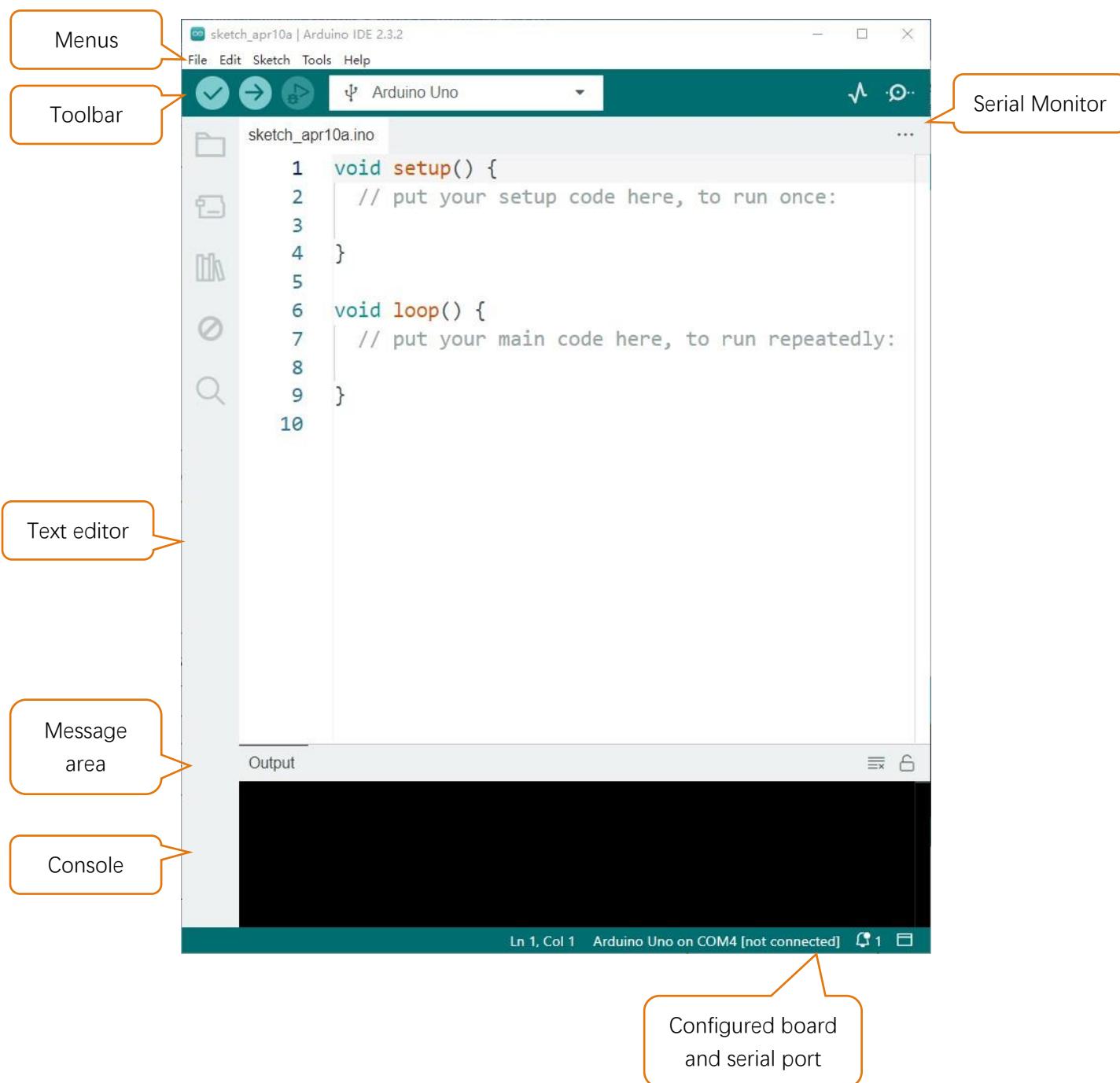
Downloads

A screenshot of the Arduino IDE 2.3.2 download page. It shows the Arduino IDE 2.3.2 icon and the title 'Arduino IDE 2.3.2'. Below the title is a description: 'The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.' There are links for 'SOURCE CODE' and 'GitHub'. To the right is a 'DOWNLOAD OPTIONS' sidebar with links for Windows (MSI installer, ZIP file), Linux (AppImage 64 bits, ZIP file 64 bits), and macOS (Intel, Apple Silicon). Orange arrows point from the 'Software' tab in the header to this page and from the 'Arduino IDE 2.3.2' icon to the 'DOWNLOAD OPTIONS' sidebar.

After the downloading completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it is popped up, please allow the installation. After installation is completed, an shortcut will be generated in the desktop.



Run it. The interface of the software is as follows:

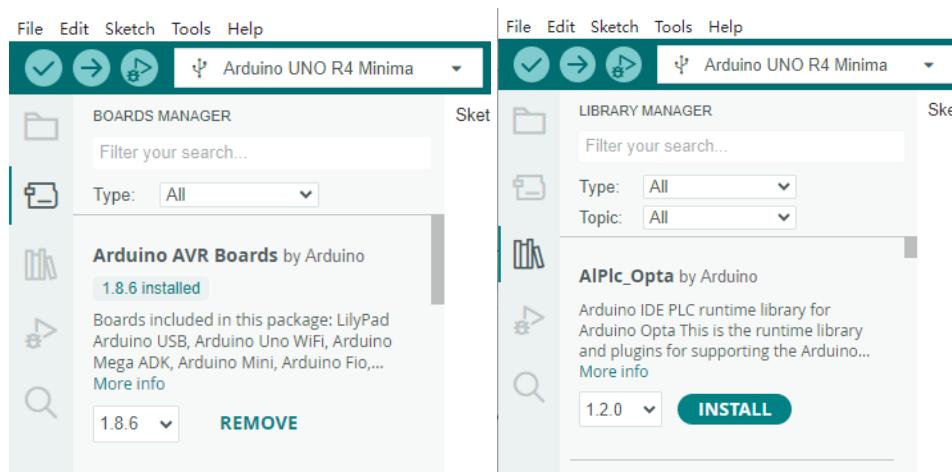


Programs written with Arduino IDE are called **sketches**. These sketches are written in a text editor and are saved with the file extension **.ino**. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino 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
Checks your code for errors compiling it.
-  Upload
Compiles your code and uploads it to the configured board.
-  New
Creates a new sketch.
-  Open
Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.
-  Save
Saves your sketch.
-  Serial Monitor
Opens the serial monitor.

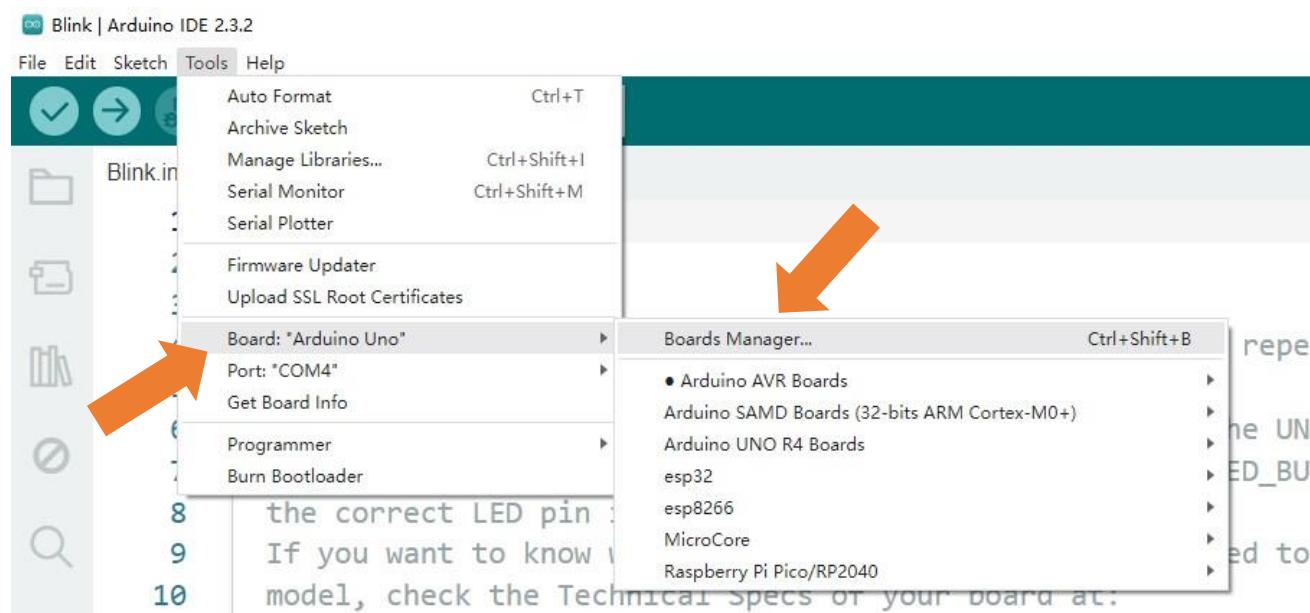
Additional commands are found within five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

In addition, the Arduino 2.X.X version provides a quick search bar for control board management and a quick search bar for libraries, so you can easily install the library files and control board support package you want.

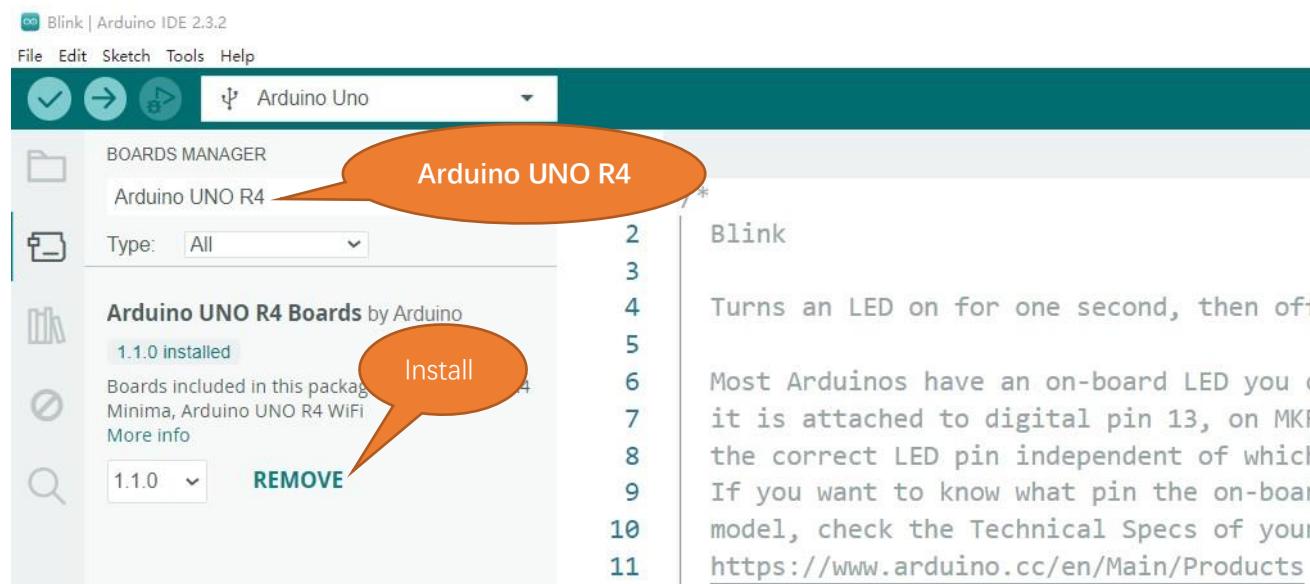


Installation of Development Board Support Package

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



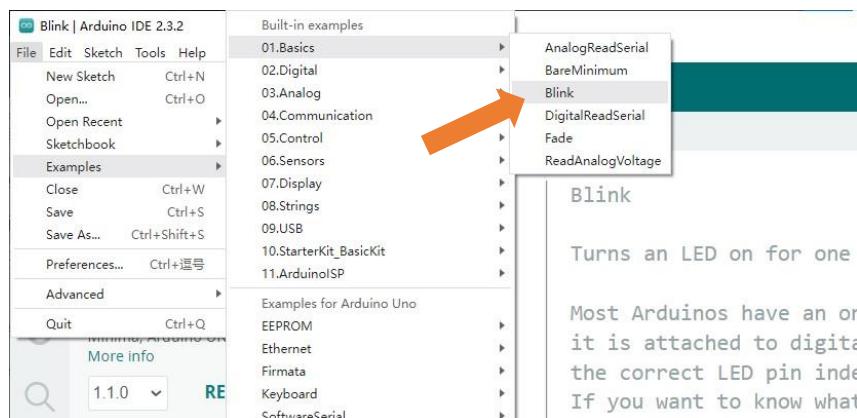
Enter **Arduino UNO R4** in the searching box, and select "**Arduino UNO R4**" and click on Install.



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.

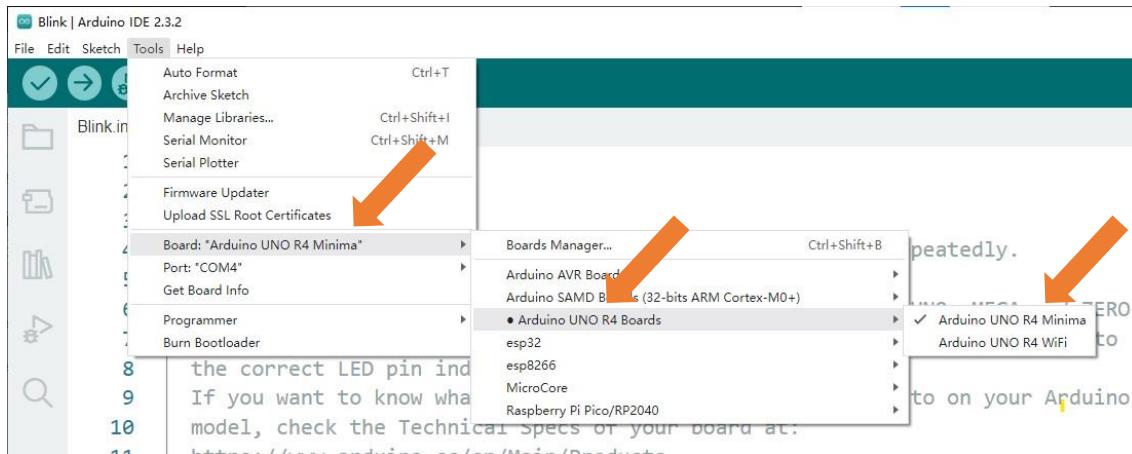
First Use

Open the example sketch "Blink".

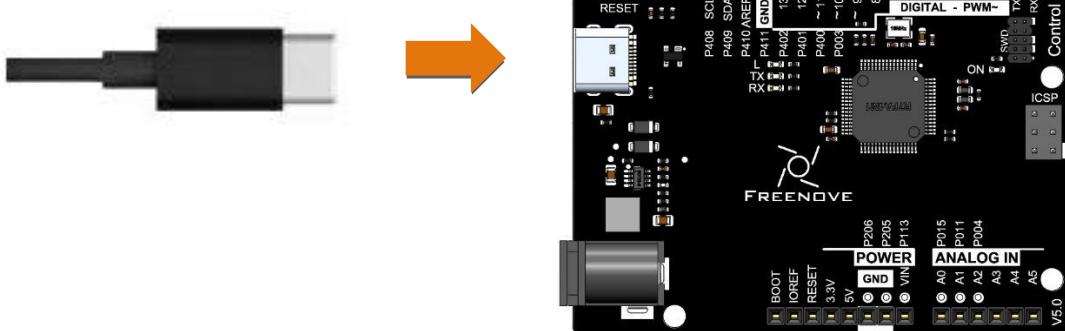


Select the board corresponding to the board you have in hands. Here we take Freenove Control Board (minina) as an example:

Select board "Arduino Uno R4 Minima". (Freenove control board is compatible with this board.)



Connect control board to your computer with USB cable.



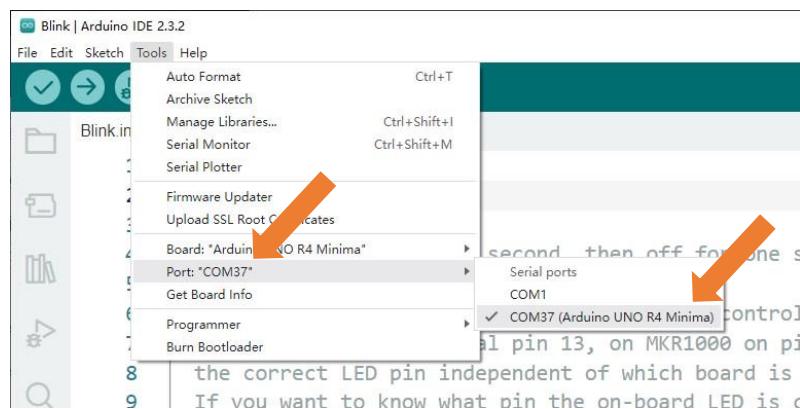
Select the port.

Note: Your port may be different from the following figure.

On Windows: It may be COM4, COM5 (Arduino Uno R4 Minima) or something like that.

On Mac: It may be /dev/cu.usbserial-710, /dev/cu.usbmodem7101 (Arduino Uno R4 Minima) or something like that.

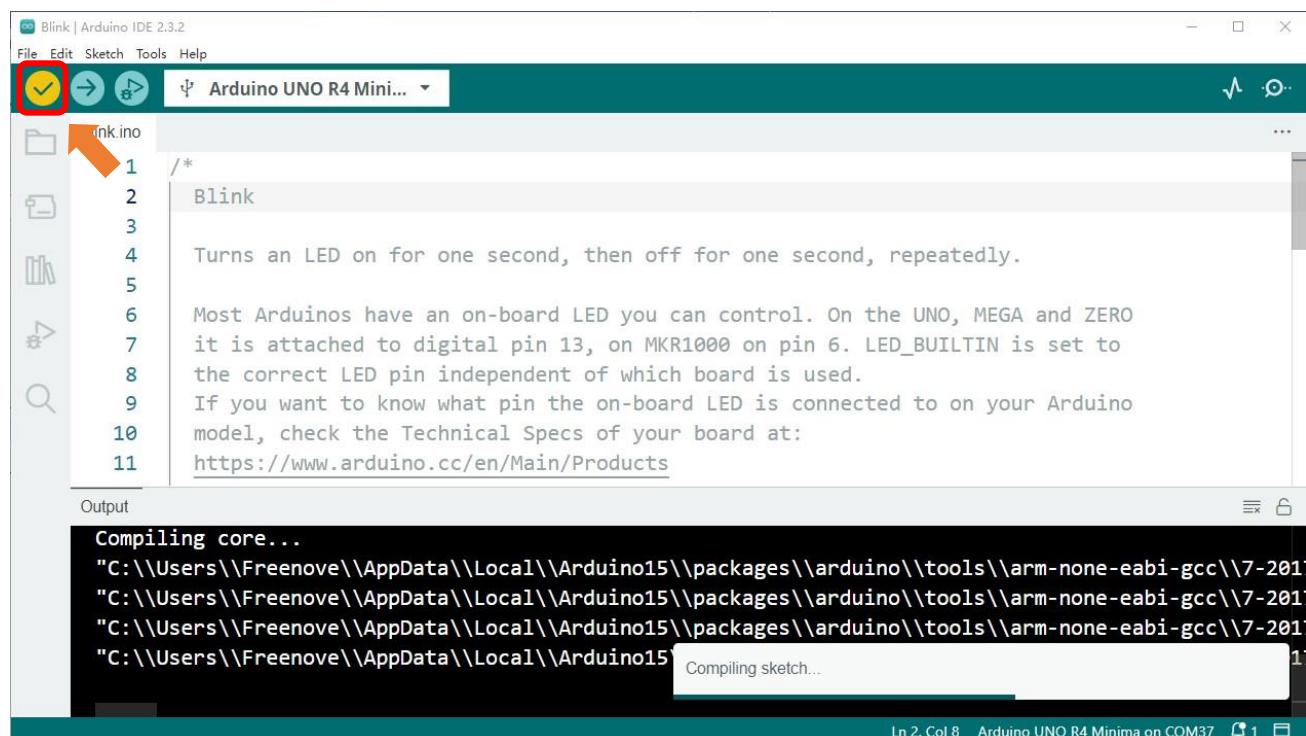
On Linux: It may be `/dev/ttyUSB0`, `/dev/ttyACM0` or something like that.



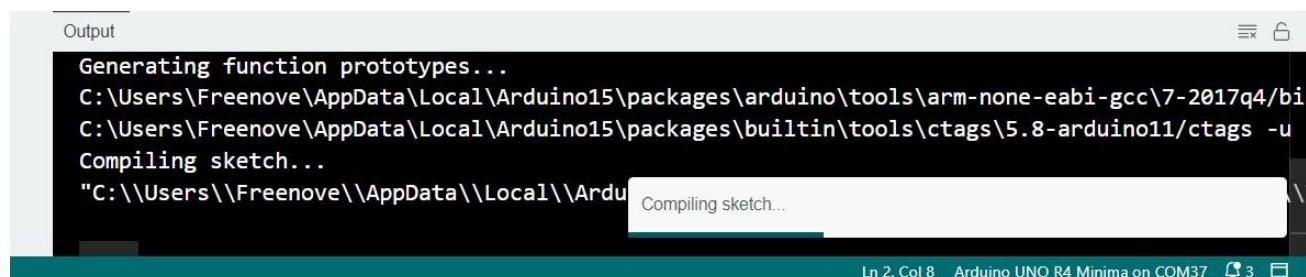
Note: If there is more than one port and you cannot decide which one to choose, disconnect the USB cable and check the port. Then connect the USB cable and check the port again. The new one is the correct port.

Having problems? Contact us for help! Send mail to: support@freenove.com

Click "Verify" button.



The following figure shows the code being compiled.



Wait a moment for the compiling to be completed. Figure below shows the code size and percentage of

Need help? Contact support@freenove.com

space occupation. If there is an error in the code, the compilation will fail and the details are shown here.



```
"C:\Users\Freenove\AppData\Local\Arduino15\packages\arduino\tools\arm-none-eabi-gcc\

"C:\Users\Freenove\AppData\Local\Arduino15\packages\arduino\tools\arm-none-eabi-gcc\

Sketch uses 54024 bytes (20%) of program storage space. Maximum is 262144 bytes.
Global variables use 4552 bytes (13%) of dynamic memory, leaving 28216 bytes for local variables

Ln 2, Col 8  Arduino Uno R4 Minima on COM37  1  1
```

Click "Upload" button.

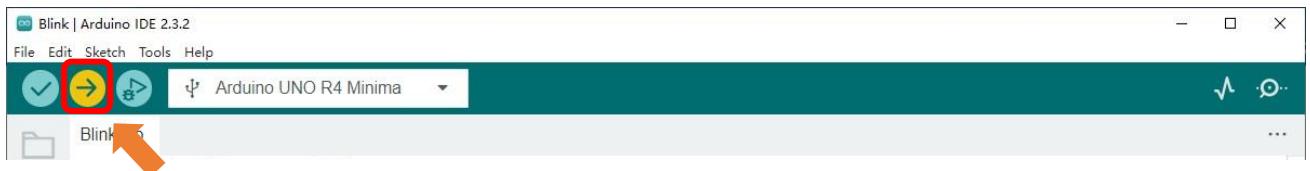
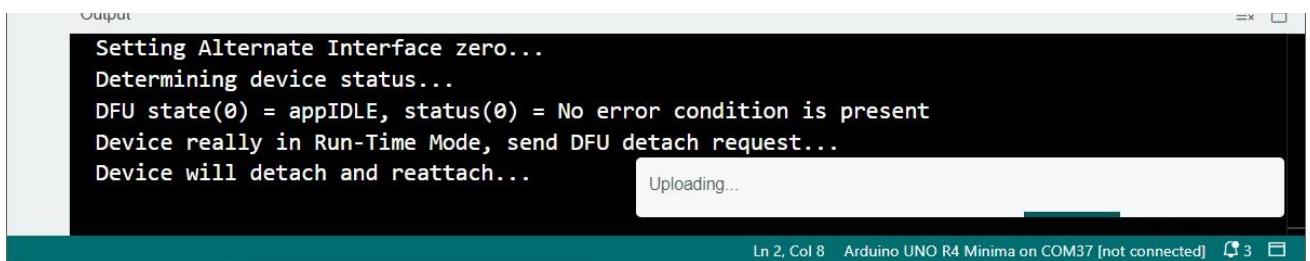


Figure below shows code are uploading.



```
Setting Alternate Interface zero...
Determining device status...
DFU state(0) = appIDLE, status(0) = No error condition is present
Device really in Run-Time Mode, send DFU detach request...
Device will detach and reattach...
Uploading...
```

Ln 2, Col 8 Arduino Uno R4 Minima on COM37 [not connected] 3 1

Wait a moment, and then the uploading is completed.

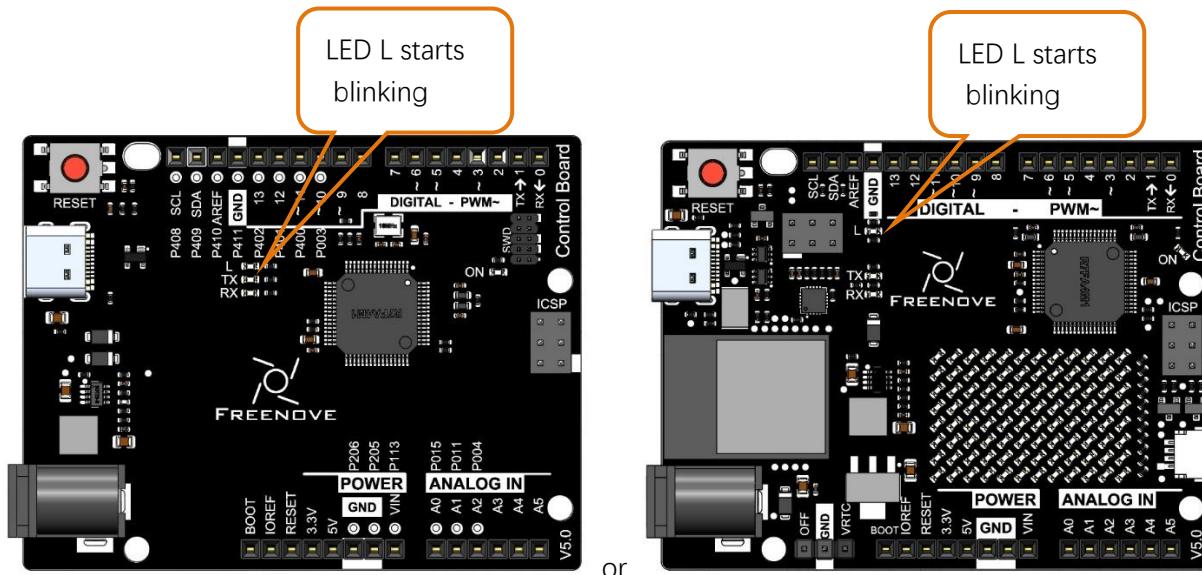


```
Download [=====] 100%      54032 bytes
Download done.
DFU state(7) = dfuMANIFEST, status(0) = No error condition is present
DFU state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

Ln 2, Col 8 Arduino Uno R4 Minima on COM37 2 1

Having problems? Contact us for help! Send mail to: support@freenove.com

After that, we will see the LED marked with "L" on the control board start blinking. It indicates that the code is running now!

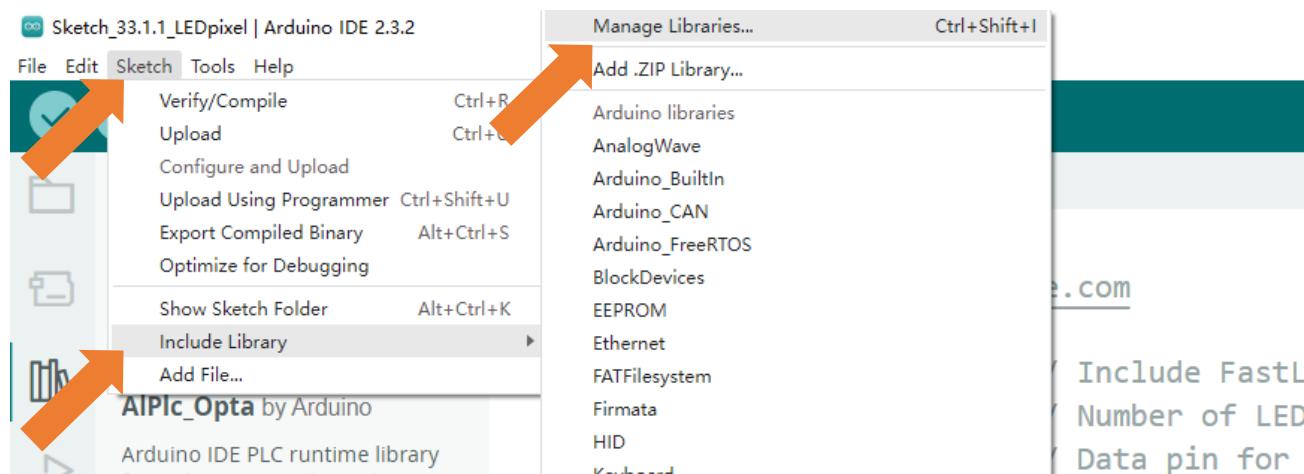


So far, we have completed the first use. I believe you have felt the joy of it. Next, we will carry out a series of projects, from easy to difficult, taking you to learn programming and the building of electronic circuit.

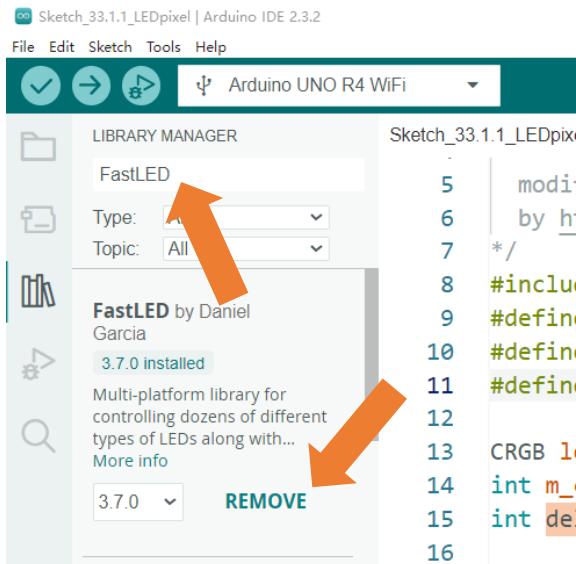
How to install the library

There are two ways to include libraries on Arduino IDE.

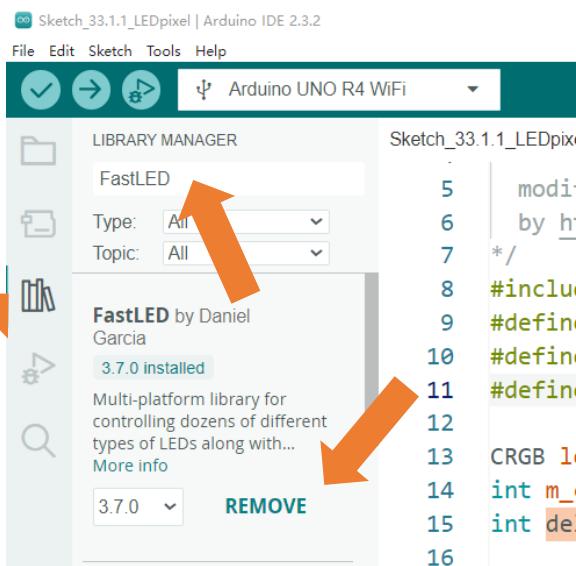
The first way, open the Arduino IDE, click Tools → Manager Libraries.



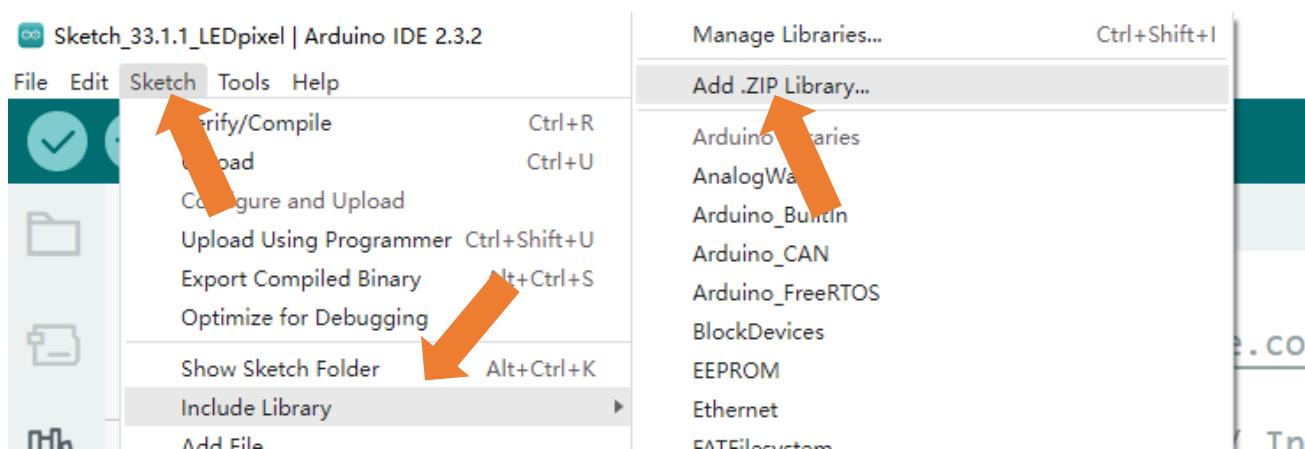
Here we take installing the “FastLED” library as an example. In the pop-up window, Library Manager, search for the name of the Library, “FastLED”. Then click Install.



Or, you can click the Library icon on the left of Arduino IDE, and type in "FastLED" on the search bar to install.



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



Chapter 1 LED Blink

We have previously tried to make the LED marked with "L" blink on the control board. Now let us use electronic components and codes to reproduce the phenomenon, and try to understand their principle.

Project 1.1 Control LED with Manual Button

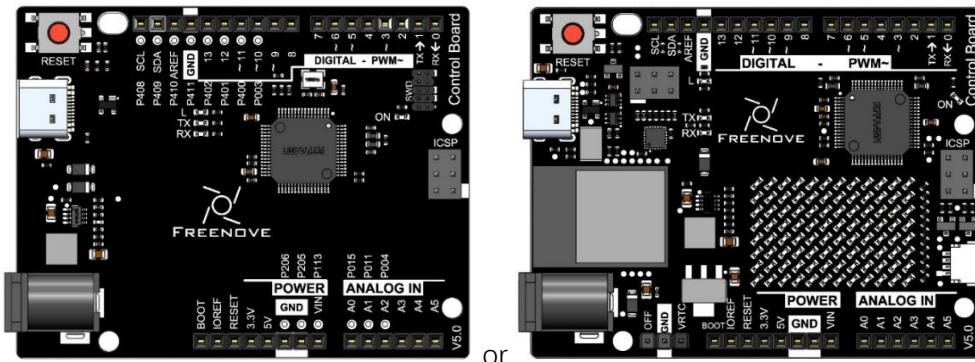
First, try using a push button switch to make the LED blink manually.

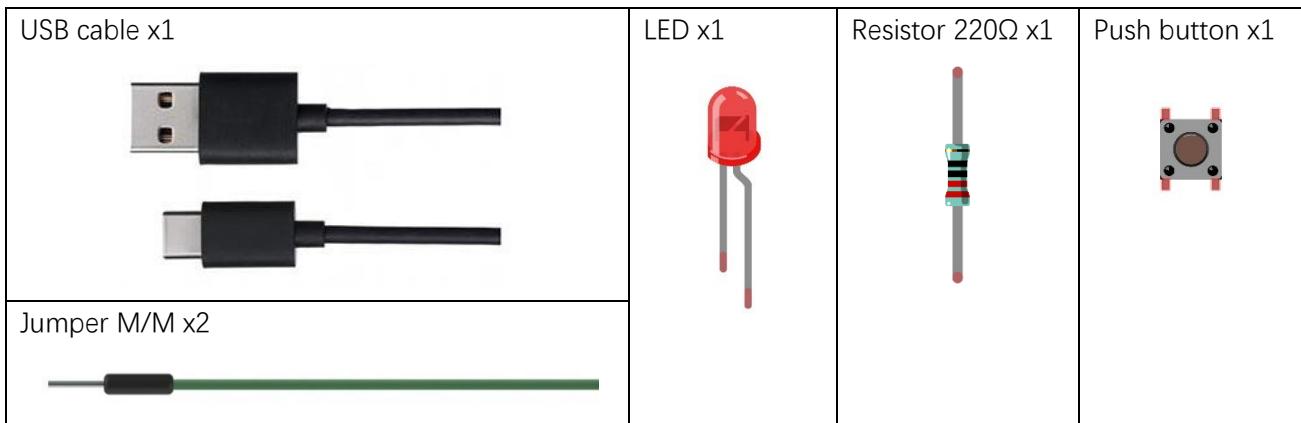
Component List

Note: It is worth noting that the board is compatible with the Arduino UNO R4 Minima and the Arduino UNO R4 WiFi board, and if you have one of them in hand, you can also use it for experiments in this tutorial.

Only the black control board is used to display the hardware connection in this document.

Control board x1





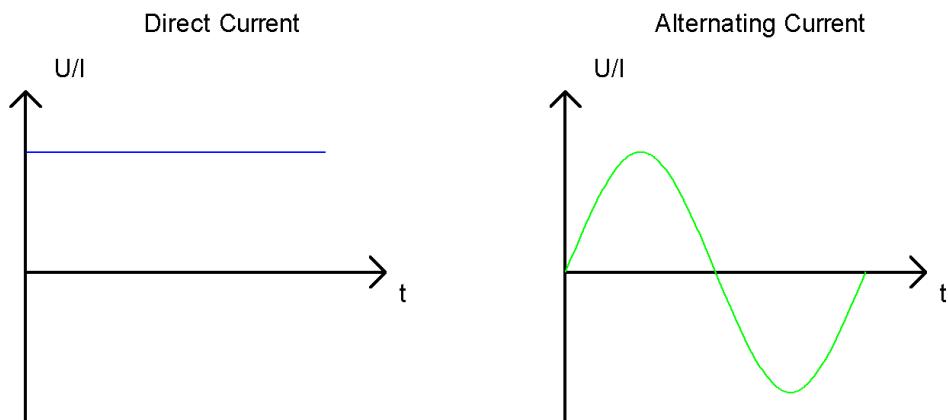
Circuit Knowledge

Power supply

Power supply provides energy for the circuit, and it is divided into DC power and AC power.

Voltage and current of DC power supply remains the same all the time, such as battery, power adapter.

Alternating Current (AC) describes the flow of charge that changes direction periodically. As a result, the voltage level also reverses along with the current. Its basic form is sinusoidal voltage(current). AC power is suitable for long-distance transmission of electric energy and it is used to supply power to houses.



Generally, electronic circuits use DC. Home appliances have rectifiers to convert AC into DC before they are used.

Battery or battery pack can be represented by the following symbols:

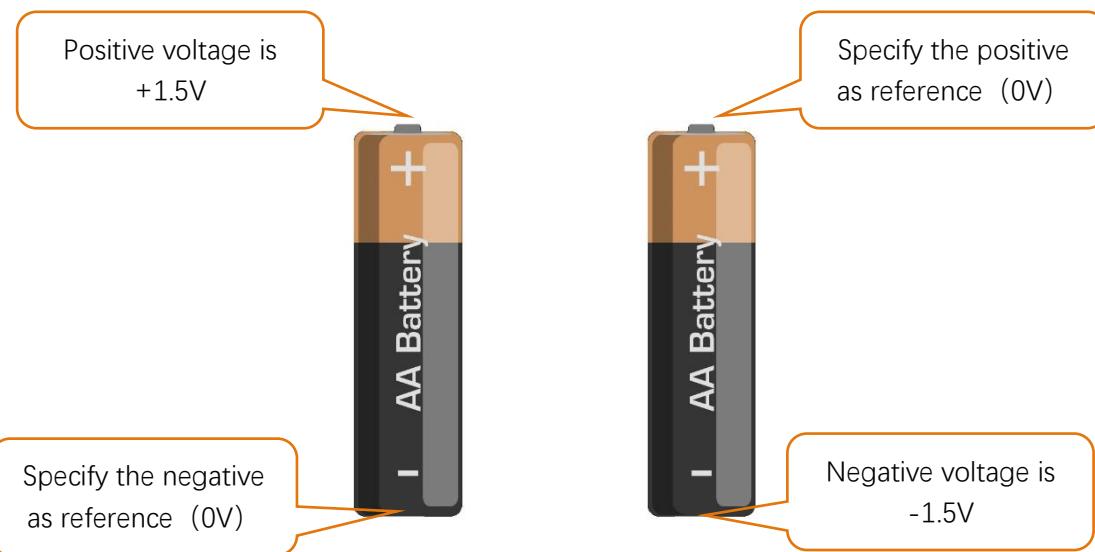


The positive and negative poles of the power supply must not be directly connected, otherwise it may scald you and cause damage to the battery.

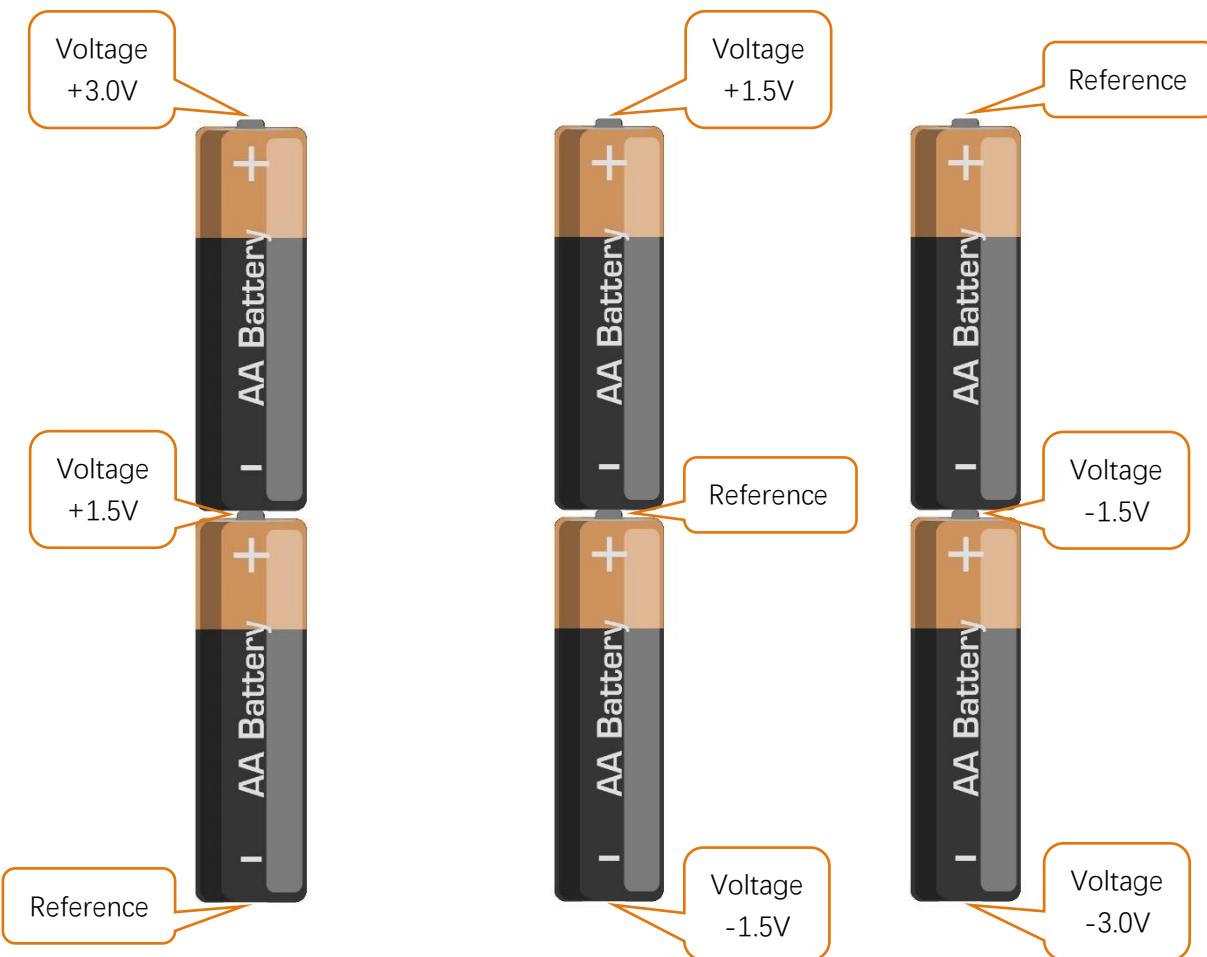
Voltage

The unit of voltage(U) is volt(V). $1kV=1000V$, $1V=1000mV$, $1mV=1000\mu V$.

Voltage is relative. As to a dry battery marked with "1.5V", its positive (+) voltage is 1.5V higher than the negative (-) voltage. If you specify the negative as reference (0V), the positive voltage will be +1.5V.



When two dry batteries are connected in series, the voltage of each point is as follows:



In practical circuits, we usually specify negative as reference voltage (0V), which is called "Ground". The positive is usually called "VCC". The positive and negative poles of power supply is usually represented by the following two symbols:

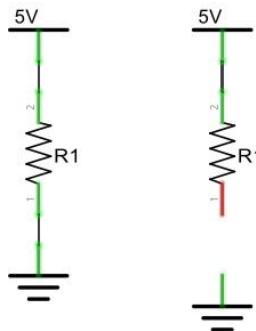


Current

The unit of current(I) is ampere(A). $1A=1000mA$, $1mA=1000\mu A$.

Closed loop consisting of electronic components is necessary for current to flow.

In the figures below: the left one is a loop circuit, so current flows through the circuit. The right one is not a loop circuit, so there is no current.

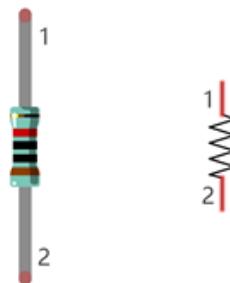


Resistor

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

A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit.

On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.

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

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



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

Component Knowledge

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

Jumper

Jumper is a kind of wire designed to connect the components together with its two terminals by inserting them onto breadboard or control board.

Jumpers have male end (pin) and female end (slot), so jumpers can be divided into the following 3 types.

Jumper M/M



Jumper F/F



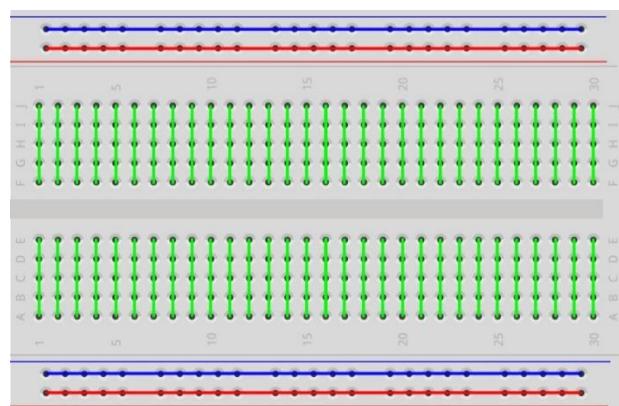
Jumper F/M



Breadboard

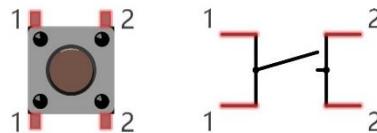
There are many small holes on breadboard to connect Jumpers.

Some small holes are connected inside breadboard. The following figure shows the inner links among those holes.



Push Button Switch

This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left



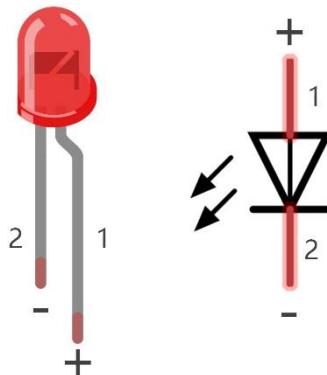
and right sides are the same per the illustration:

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

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 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.



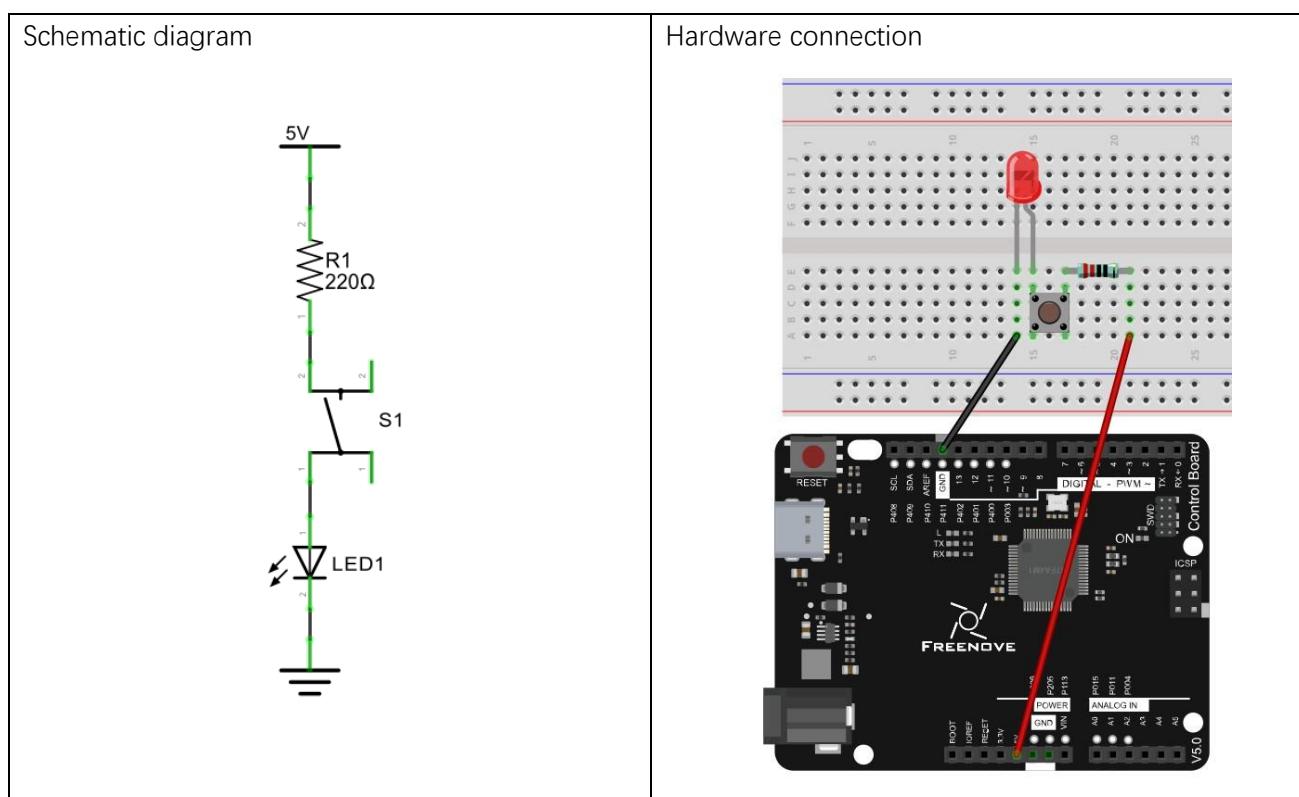
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Circuit

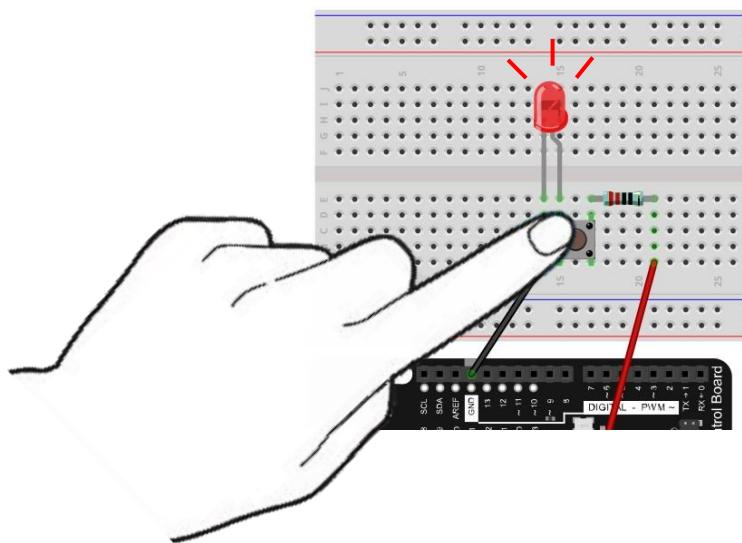
In this project, the LED is controlled by a push button switch, and the control board here only plays the role of power supply in the circuit.

Firstly, connect components with jumpers according to "hardware connection". Secondly, check the connection to confirm that there are no mistakes. Finally, connect the control board to computer with USB cable to avoid short circuit caused by contacting the wires.

Note: In this book, we use the regular board as an example to make the circuits. The connection is the same on the control board with WiFi function.



LED lights up when you press the push button switch, and it lights off when you release the button.



Project 1.2 Control LED with Control Board

Now, try using control board to make LED blink through programming.

Component List

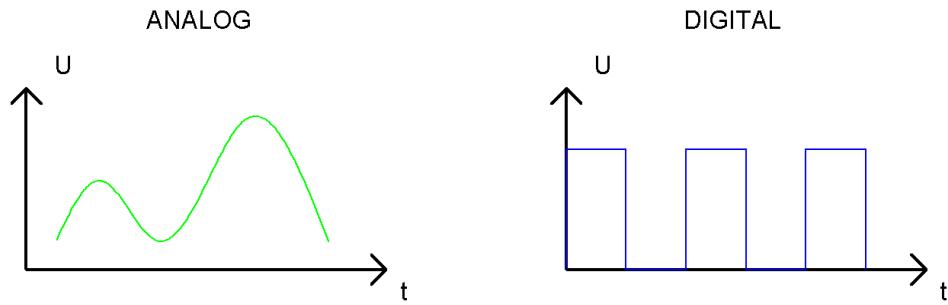
Components are basically the same with those in last section. Push button switch is no more needed.

Circuit Knowledge

Analog signal and Digital signal

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0°C to 10°C.

However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



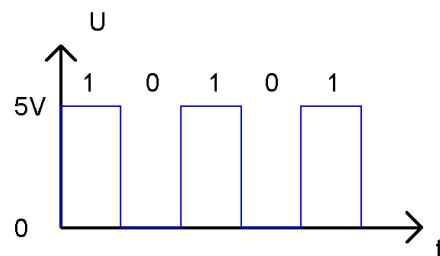
In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

Low level and high level

In a circuit, the form of binary (0 and 1) is presented as low level and high level.

Low level is generally equal to ground voltage(0V). High level is generally equal to the operating voltage of components.

The low level of the control board is 0V and high level is 5V, as shown below. When IO port on control board outputs high level, components of small power can be directly lit, like LED.



Code Knowledge

Before start writing code, we should learn about the basic programming knowledge.

Comments

Comments are the words used to explain for the sketches, and they won't affect the running of code.

There are two ways to use comments of sketches.

1. Symbol "://"

Contents behind "://" comment out the code in a single line.

```
1 // this is a comment area in this line.
```

The content in front of "://" will not be affected.

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

2. Symbol "/*"and "*/"

Code can also be commented out by the contents starting with a "/*" and finishing with a "*/" and you can place it anywhere in your code, on the same line or several lines.

```
1 /* this is comment area. */
```

Or

```
1 /*
2      this is a comment line.
3      this is a comment line.
4 */
```

Data type

When programming, we often use digital, characters and other data. C language has several basic data types as follows:

int: A number that does not have a fractional part, an integer, such as 0, 12, -1;

float: A number that has a fractional part, such as 0.1, -1.2;

char: It means character, such as 'a', '@', '0';

For more about date types, please visit the website: <https://www.Arduino.cc-Resources-Reference-Data Types>.

Constant

A constant is a kind of data that cannot be changed, such as int type 0, 1, float type 0.1, -0.1, char type 'a', 'B'.

Variable

A variable is a kind of data that can be changed. It consists of a name, a value, and a type. Variables need to be defined before using, such as:

```
1 int i;
```

"int" indicates the type, ";" indicates the end of the statement. The statement is usually written in one single line; and these statements form the code.

After declaration of the variable, you can use it. The following is an assignment to a variable:

```
1 i = 0; // after the execution, the value of i is 0
```

"=" is used to pass the value of a variable or constant on the right side to the variable on the left.

A certain number of variables can be declared in one statement, and a variable can be assigned multiple times. Also, the value of a variable can be passed to other variables. For example:

```

1 int i, j;
2 i = 0;           // after the execution, the value of i is 0
3 i = 1;           // after the execution, the value of i is 1
4 j = i;           // after the execution, the value of j is 1

```

Function

A function is a collection of statements with a sequence of order, which performs a defined task. Let's define a function void blink() as follows:

```

1 void blink() {
2     digitalWrite(13, HIGH);
3     delay(1000);
4     digitalWrite(13, LOW);
5     delay(1000);
6 }

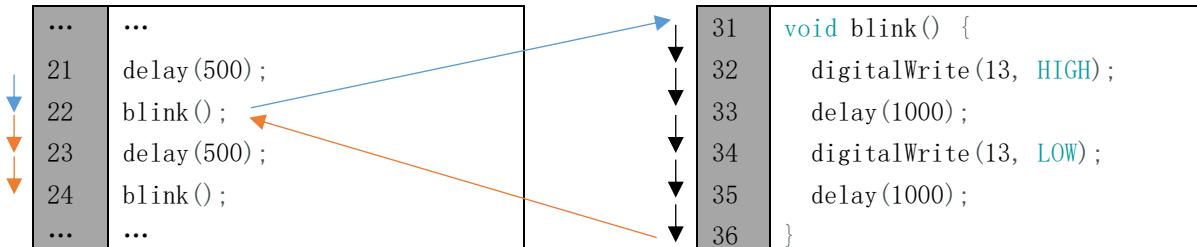
```

"void" indicates that the function does not return a value (Chapter 4 will detail the return value of functions); "()" its inside is parameters of a function (Chapter 2 will detail the parameters of the functions). No content inside it indicates that this function has no parameters;
"{}" contains the entire code of the function.

After the function is defined, it is necessary to be called before it is executed. Let's call the function void blink(), as shown below.

```
1 blink();
```

When the code is executed to a statement calling the function, the function will be executed. After execution of the function is finished, it will go back to the statement and execute the next statement.



Some functions have one or more parameters. When you call such functions, you need to write parameters inside "()":

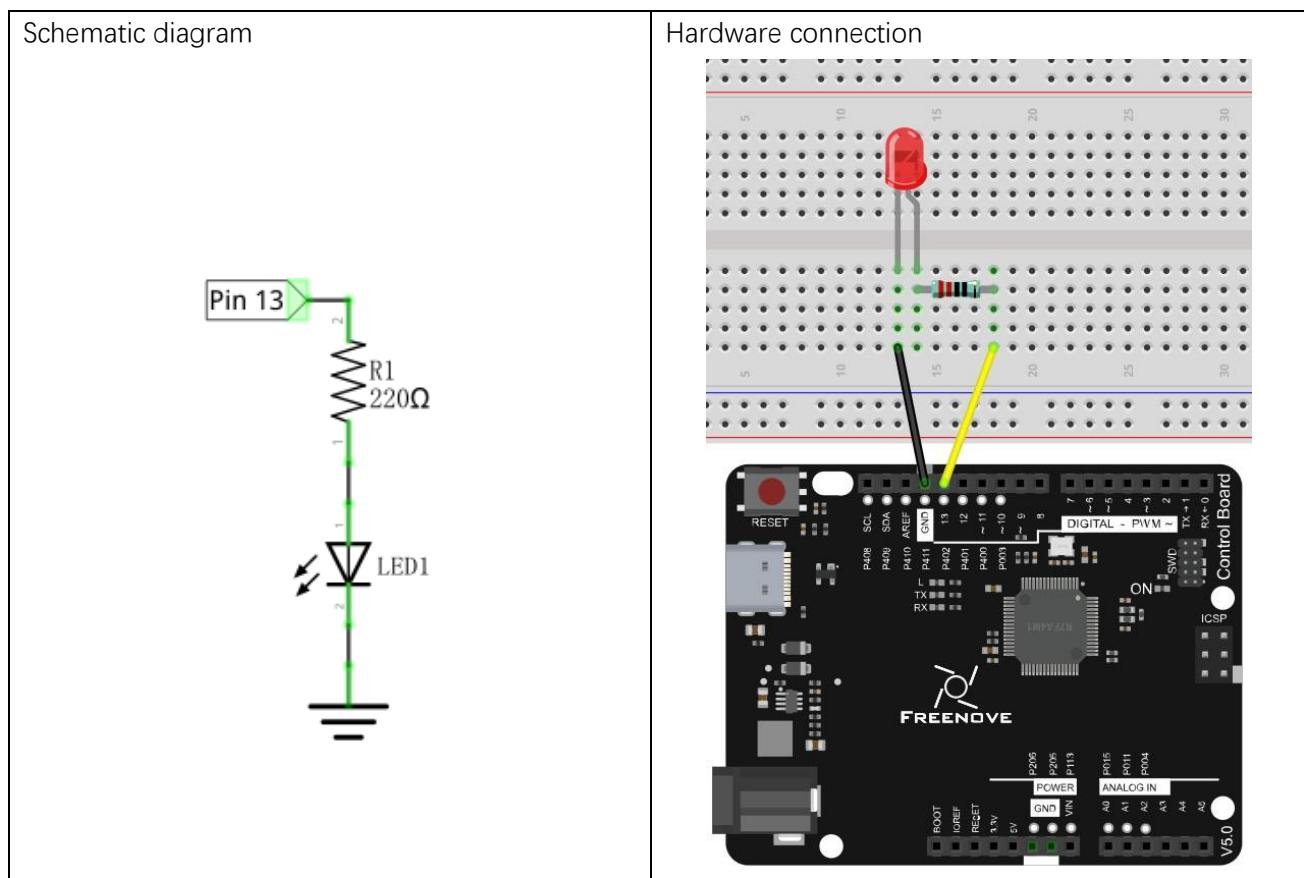
```

1 digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
2 delay(1000);           // wait for a second

```

Circuit

Now, we will use IO port of control board to provide power for the LED. Pin 13 of the control board is the digital pin. It can output high level or low level. In this way, control board can control the state of LED.



Sketch

Sketch 1.2.1

In order to make the LED blink, we need to make pin 13 of the control board output high and low level alternately.

We highly recommend you type the code manually instead of copying and pasting, so that you can develop your coding skills and get more knowledge.

```

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

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

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

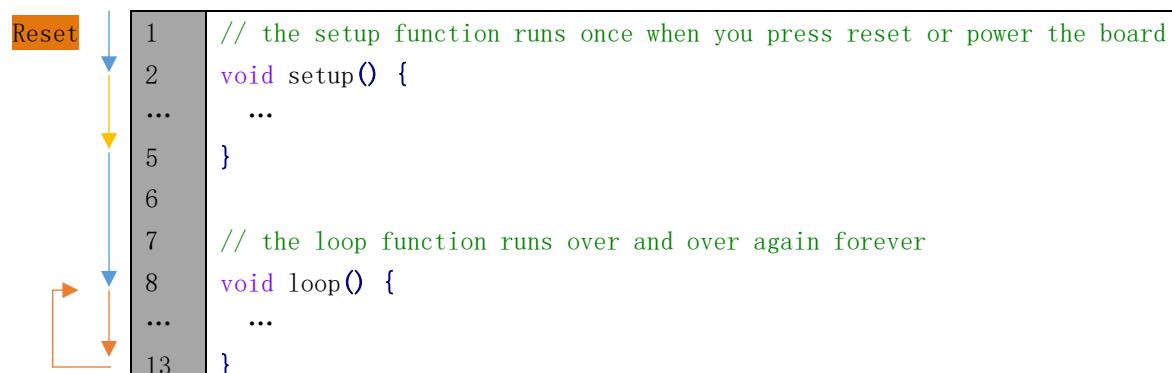
setup() function is generally used to write code to initialize the hardware.

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 jump to the beginning of loop() to run again.

Reset

Reset operation will lead the code to be executed from the beginning. Switching on the power, finishing uploading the code and pressing the reset button will trigger reset operation.



In the setup () function, first, we set pin 13 of the control board as output mode, which can make the port output high level or low level.

```
3 // initialize digital pin 13 as an output  
4 pinMode(13, OUTPUT);
```

Then, in the loop () function, set pin 13 of the control board to output high level to make LED light up.

```
9 digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
```

Wait for 1000ms, which 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.

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

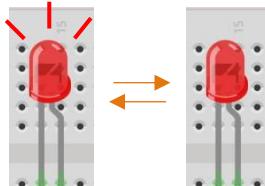
Then set the 13 pint to output low level, and LED light off. One second later, the execution of loop () function will be completed.

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

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

The functions called above are standard functions of the Arduino IDE, which have been defined in the Arduino IDE, and they can be called directly. We will introduce more common standard functions in later chapters. For more standard functions and the specific use method, please visit <https://www.arduino.cc>-Resources-Reference-Functions.

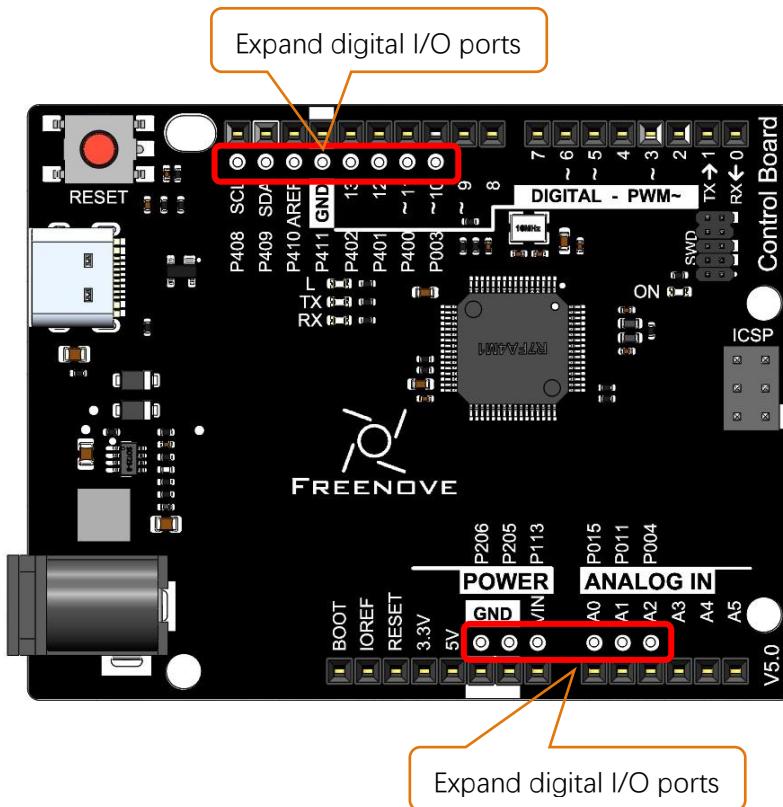
Verify and upload the code, then the LED starts blinking.



How to Use the Expanding GPIO Pins

In this section, we will learn to use the expanding GPIO pins. If you are working on the board with Bluetooth and WiFi functions, you can skip this section.

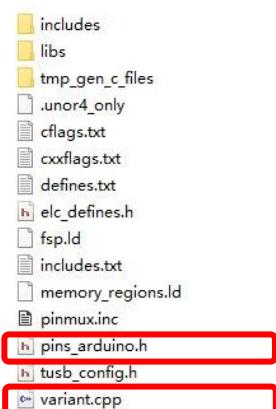
The board used in this section is as shown below:



Before using the expanding GPIO pins, please complete the following configuration first.

Copy "pins_arduino.h" and "variant.cpp" under the file path "/Libraries/Expanding_GPIO_Pins" to the file path "C:\Users\Freenove\AppData\Local\Arduino15\packages\arduino\hardware\renesas_arduino\1.1.0\variants\MINIMA".

```
1 > Freenove > AppData > Local > Arduino15 > packages > arduino > hardware > renesas_arduino > 1.1.0 > variants > MINIMA
```



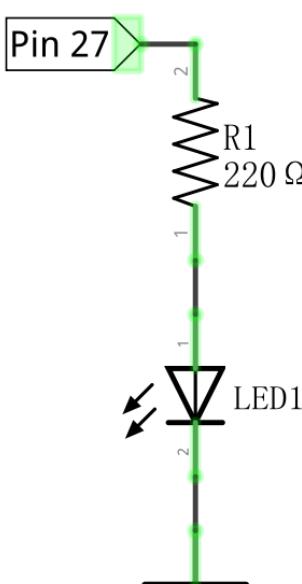
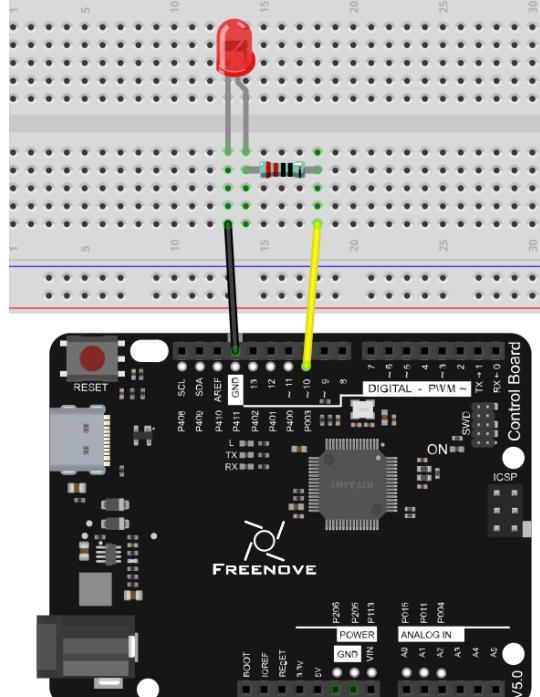
The path may vary due to different versions of board package installed.

The configuration is completed once the files are copied to the above path. The definition of the expanding GPIOs is as shown in the table below.

GPIOs on the control board	Definition of Arduino GPIO
P003	D27
P400	D28
P401	D29
P402	D30
P411	D31
P410	D32
P409	D33
P408	D34
P004	D35
P011	D36
P015	D37
P113	D38
P205	D39
P206	D40

Now, let's try to use the expanding GPIO pins to light up an LED. Open the Blinking sketch, modify the control pin according to the pin definition shown in the above table. Here we take P003 on the control board as an example, so we modify the control pin of the LED to 27.

Build the circuit as below, and upload the sketch to the board.

Schematic diagram 	Hardware connection 
---	--

```
1 // the setup function runs once when you press reset or power the board
2 void setup() {
3     // initialize digital pin 13 as an output
4     pinMode(27, OUTPUT);
5 }
6
7 // the loop function runs over and over again forever
8 void loop() {
9     digitalWrite(27, HIGH);    // turn the LED on (HIGH is the voltage level)
10    delay(1000);             // wait for a second
11    digitalWrite(27, LOW);    // turn the LED off by making the voltage LOW
12    delay(1000);             // wait for a second
13 }
```

In this way, you can use more GPIOs to design more interesting projects. It is worth noting that all the expansion GPIOs are uniformly defined as ordinary digital I/O interfaces. We do not solder male or female headers to them. When you need to use these pins, you can solder headers yourself.

Chapter 2 Two LEDs Blink

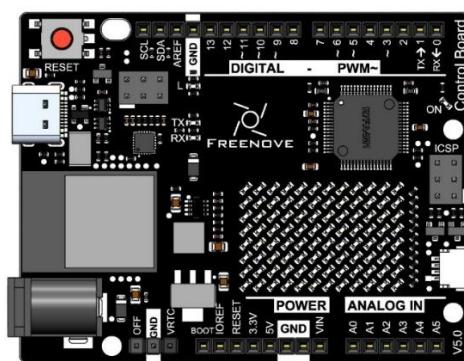
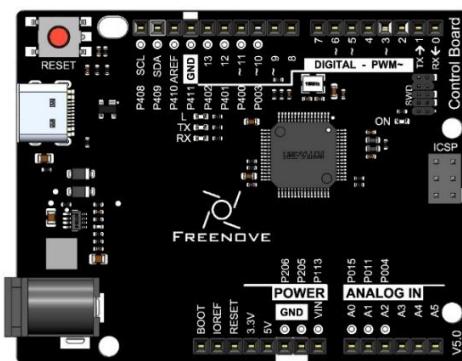
In last chapter, we have already written code to make 1 LED blink on the control board. And now, we will try to make 2 LEDs blink for further programming study.

Project 2.1 Two LEDs Blink

Now, try to make two LEDs blink on control board.

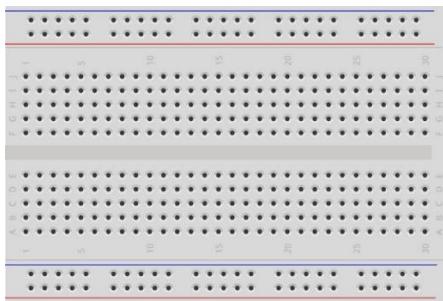
Component List

Control board x1

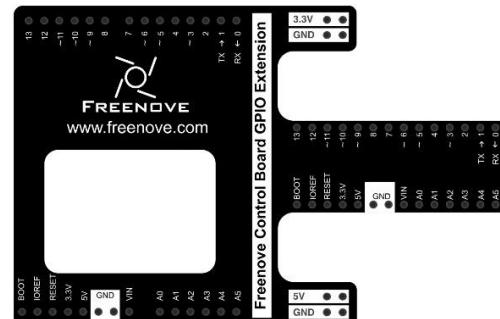


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



LED x2



Resistor 220Ω x2



Jumper M/M x4



Code Knowledge

In the last chapter, we have taken a brief look at programming. Now let us learn more about the basic programming knowledge.

Parameters of function

In the last chapter, we have used a function with a parameter, such as:

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

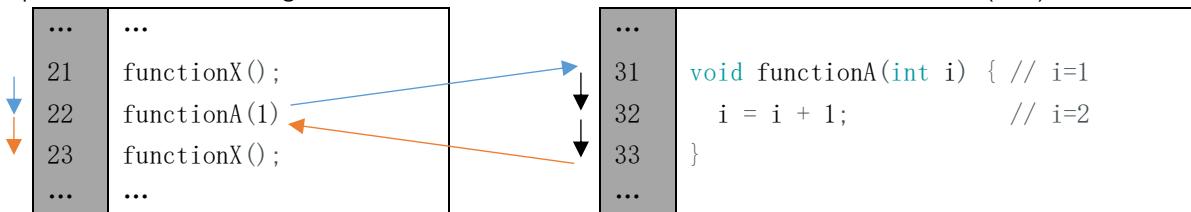
Next, we will define a function with a parameter as below:

```
1 void functionA(int i) {
2     i = i + 1;
3 }
```

"i" is the parameter of this function. "int" is the type of i. When calling this function, it is necessary to enter the parameter of int type:

```
1 functionA(1);
```

The input parameter will be assigned to "i" and involved in the calculation of the functionA(int i):



A function can define more than one parameter and the type of the parameters can be different:

```
1 void functionB(int i, char j) {
2     char k = 'a';
3     i = i + 1;
4     k = j;
5 }
```

Boolean data type

Data of Boolean type can only be assigned to "true" or "false".

"true" generally represents a certain relationship which is tenable and correct, and "false" is the opposite.

```
1 boolean isTrue;
2 isTrue = true; // after the execution, "isTrue" is assigned to true.
3 isTrue = false; // after the execution, "isTrue" is assigned to false.
```

In the code, the number 0 can be considered to be false, and nonzero numbers can be considered true.

Logical operator

The logic operators have "&&" (and), "||" (or), "!" (non), and the calculation object of them are boolean type. The result of logic operation is as follows:

&&	true	false
true	true	false
false	false	false

	true	false
true	true	true
false	true	false

!	
true	false
false	true

For example:

```

1 boolean isTrue;
2 isTrue = true && false; // after the execution, "isTrue" is assigned to false.
3 isTrue = true || false; // after the execution, "isTrue" is assigned to true.
4 isTrue = !true;        // after the execution, "isTrue" is assigned to false.

```

Relation operator

Relational operator is used to judge whether the relationship of the two amount is tenable and correct. If the relationship is tenable, the result is true. Otherwise, the result is false.

For example, the results of "1<2" is true and the result of "1>2" is false:

```

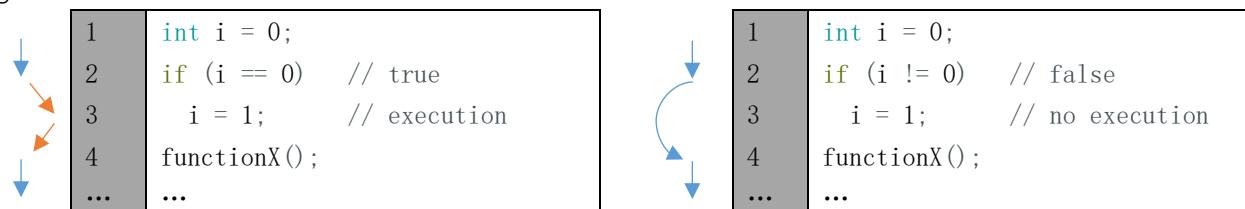
1 boolean isTrue;
2 isTrue = 1 < 2;           // after the execution, "isTrue" is true.
3 isTrue = 1 > 2;           // after the execution, "isTrue" is false.

```

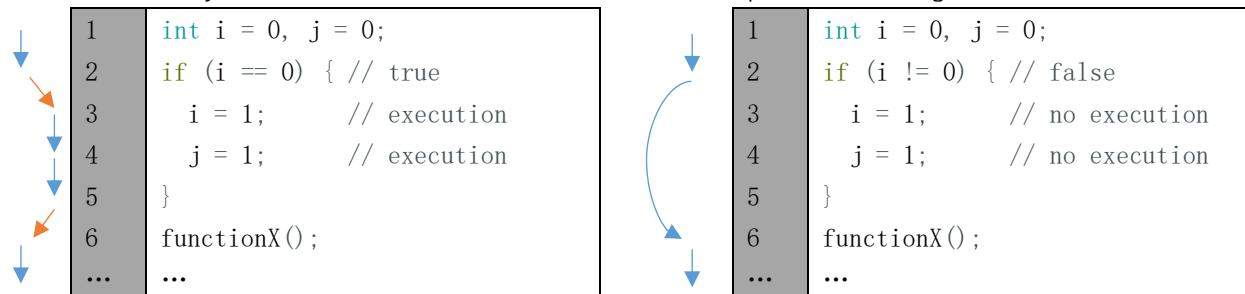
There are other relational operators such as "==" (equal to), ">=" (greater than or equal to), "<=" (less than or equal to) and "!=" (not equal to).

Conditional statement

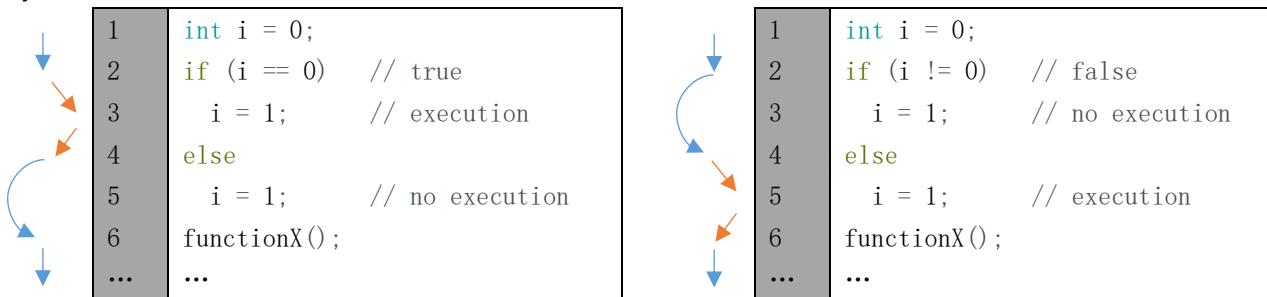
Conditional statements are used to decide whether or not to execute the program based on the result of judgment statement.



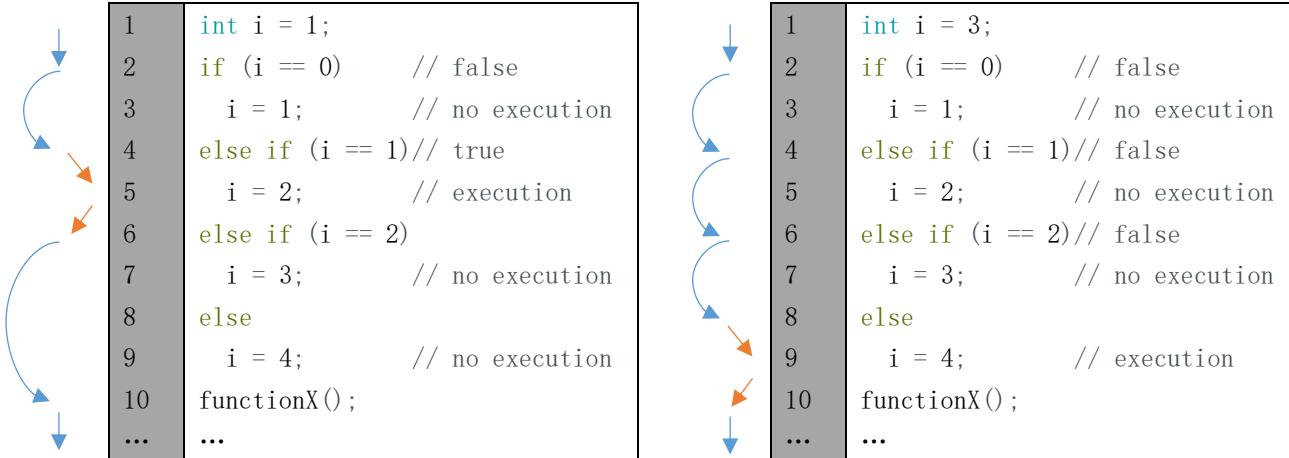
When there are many statements needed to be executed, we can put them into "{}":



Only the section of code in which conditions are tenable will be executed:



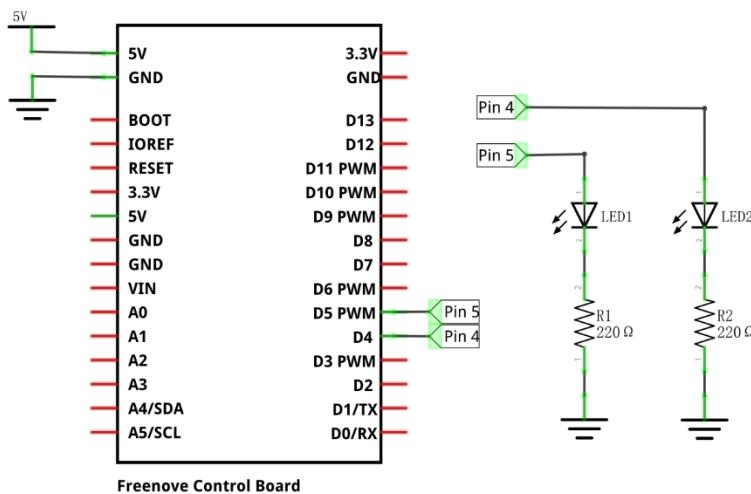
In addition, it can judge multiple conditions.



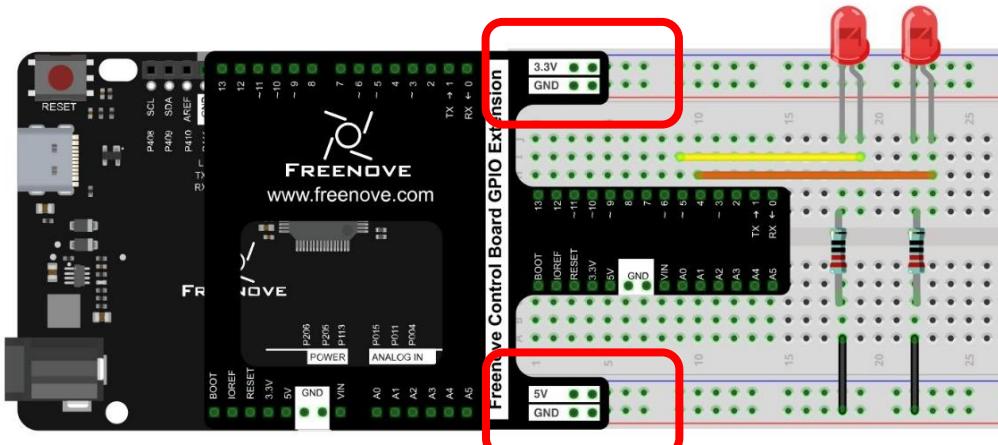
Circuit

Use pin 4 and pin 5 of the control board to drive these two LEDs respectively.

Schematic diagram



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



Sketch

In order to show the difference between using function and not using function, we will write two different sketches to make two LEDs blink.

Sketch 2.1.1

At first, use sketch without function to make two LEDs blink alternatively.

```

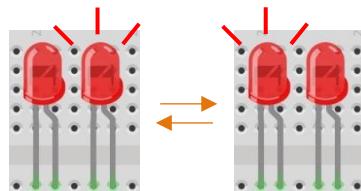
1 // set pin numbers:
2 int led1Pin = 5;           // the number of the LED1 pin
3 int led2Pin = 4;           // the number of the LED2 pin
4
5 void setup() {
6     // initialize the LED pin as an output:
7     pinMode(led1Pin, OUTPUT);
8     pinMode(led2Pin, OUTPUT);
9 }
10
11 void loop() {
12     digitalWrite(led1Pin, HIGH);    // turn the LED1 on
13     digitalWrite(led2Pin, LOW);    // turn the LED2 off
14     delay(1000);                 // wait for a second
15
16     digitalWrite(led1Pin, LOW);    // turn the LED1 off
17     digitalWrite(led2Pin, HIGH);   // turn the LED2 on
18     delay(1000);                 // wait for a second
19 }
```

This section of code is similar to the previous section. The difference is that the amount of LEDs is two, and the two LEDs blink alternatively.

Variable scope

In the 2nd and 3rd rows of the code above, we define two variables to store the pin number. These two variables defined outside the function are called "Global variable", which can be called by all other functions. Variables defined inside a function is called "local variable", which can be called only by the current function. When local variables and global variables have same names, the global variable is inaccessible within the function.

Verify and upload the code, then you will see the two LEDs blink alternatively.



Sketch 2.1.2

In the last sketch, we can see that the following two sections of the code are similar, so we will use one function to replace them to simplify the code.

```
12  digitalWrite(led1Pin, HIGH); // turn the LED1 on
13  digitalWrite(led2Pin, LOW); // turn the LED2 off
14  delay(1000); // wait for a second
```

```
16  digitalWrite(led1Pin, LOW); // turn the LED1 off
17  digitalWrite(led2Pin, HIGH); // turn the LED2 on
18  delay(1000); // wait for a second
```

Now, we will use a function to improve the above code.

```
1 // set pin numbers:
2 int led1Pin = 5; // the number of the LED1 pin
3 int led2Pin = 4; // the number of the LED2 pin
4
5 void setup() {
6     // initialize the LED pin as an output:
7     pinMode(led1Pin, OUTPUT);
8     pinMode(led2Pin, OUTPUT);
9 }
10
11 void loop() {
12     setLed(HIGH, LOW); // set LED1 on, and LED2 off.
13     setLed(LOW, HIGH); // set LED1 off, and LED2 on.
14 }
15
16 void setLed(int led1, int led2) {
17     digitalWrite(led1Pin, led1); // the state of LED1 is determined by variable led1.
18     digitalWrite(led2Pin, led2); // the state of LED2 is determined by variable led2.
19     delay(1000); // wait for a second
20 }
```

In the sketch above, we integrate the 2 LED statements into one function, void setLed(int led1, int led2), and control two LEDs through the parameters led1 and led2.

```
16 void setLed(int led1, int led2) {
17     digitalWrite(led1Pin, led1); // the state of LED1 is determined by variable led1.
18     digitalWrite(led2Pin, led2); // the state of LED2 is determined by variable led2.
19     delay(1000); // wait for a second
20 }
```

When the function above is called, we will control the two LEDs by using different parameters as below.

```
12 setLed(HIGH, LOW); // set LED1 on, and LED2 off.
13 setLed(LOW, HIGH); // set LED1 off, and LED2 on.
```

Verify and upload the code, then you will see the two LEDs blink alternatively.

HIGH and LOW

The macro is an identifier that represents a number, a statement, or a piece of code. HIGH and LOW are two macros. HIGH and LOW are defined in Arduino IDE as below:

```
#define HIGH 1  
#define LOW 0
```

In the code, a macro is used as the content defined by itself. For example, setLed (HIGH, LOW) is equivalent to setLed (1, 0).

Using macros can simplify the code and enhance its readability, such as INPUT, OUTPUT.

Sketch 2.1.3

In the previous section of code, we used a function that integrates two similar paragraphs of code. And we control two LEDs to blink alternatively by using two parameters. Now, let us try to use one parameter to control these two LEDs, which is achieved by conditional statements.

Now, we'll use conditional statement to improve the code above.

```
1 // set pin numbers:  
2 int led1Pin = 5;           // the number of the LED1 pin  
3 int led2Pin = 4;           // the number of the LED2 pin  
4  
5 void setup() {  
6     // initialize the LED pin as an output:  
7     pinMode(led1Pin, OUTPUT);  
8     pinMode(led2Pin, OUTPUT);  
9 }  
10  
11 void loop() {  
12     setLed1(HIGH);      // set LED1 on, and LED2 off.  
13     setLed1(LOW);       // set LED1 off, and LED2 on.  
14 }  
15  
16 void setLed1(int led1) {  
17     digitalWrite(led1Pin, led1); // the state of LED1 is determined by variable led1.  
18  
19     if (led1 == HIGH)        // the state of LED2 is determined by variable led1.  
20         digitalWrite(led2Pin, LOW); // if LED1 is turned on, LED2 will be turned off.  
21     else  
22         digitalWrite(led2Pin, HIGH); // if LED1 is turned off, LED2 will be turned on.  
23  
24     delay(1000);            // wait for a second  
25 }
```

Here, we rewrite the function so that we only need to set the state of LED1, and the state of LED2 can be set automatically.

Verify and upload the code, and then two LEDs blink alternatively.

Chapter 3 LED Bar Graph

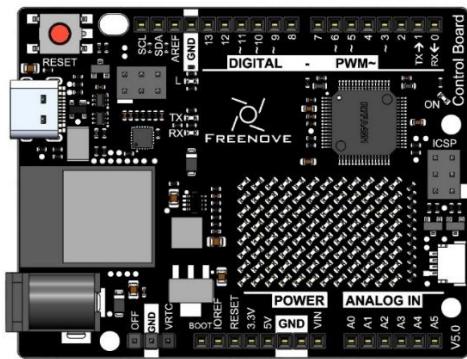
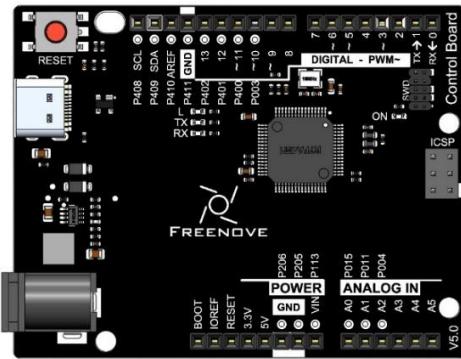
We have learned previously how to control 1 or 2 LEDs through Sketch on the control board and learned some basic knowledge of programming. Now let us try to control 10 LEDs and learn how to simplify the code.

Project 3.1 LED Bar Graph Display

Let us use control board to control a bar graph LED consisting of 10 LEDs.

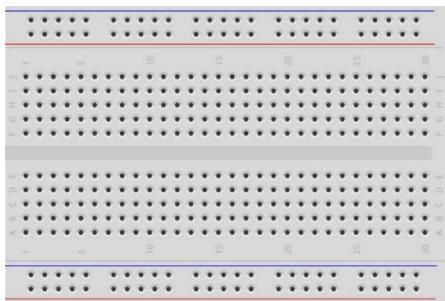
Component List

Control board x1

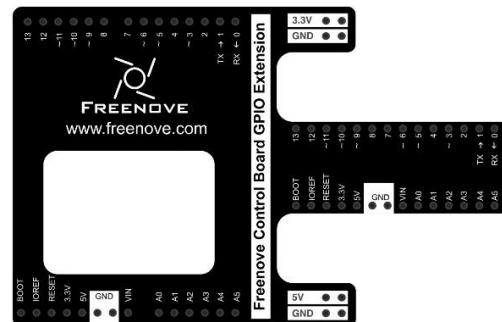


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



LED bar graph x1



Resistor 220Ω x10



Jumper M/M x10

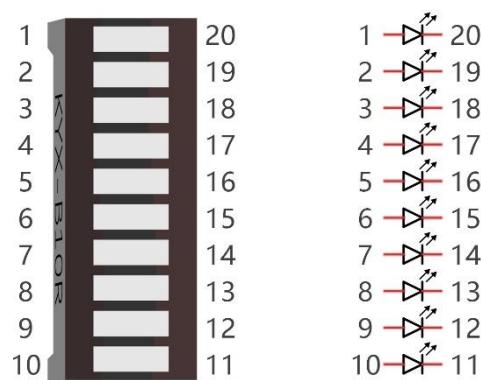


Component Knowledge

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

LED bar graph

LED bar graph is a component integration consisting of 10 LEDs. At the bottom of the LED bar graph, there are two rows of pins, corresponding to positive and negative pole separately. If the LED bar graph cannot work in the circuit, it is probably because the connection between positive and negative pole is wrong. Please try to reverse the LED bar graph connection.



Code Knowledge

This section will use new code knowledge.

Array

Array is used to record a set of variables. An array is defined as below:

```
1 int a[10];
```

"int" is the type of the array and "10" represents the amount of elements of the array. This array can store 10 int types of elements as below.

```
1 int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Or there is another form that the number of elements is the size of the array:

```
1 int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

We can reference elements of an array as below:

```
1 int i, j;
2 i = a[0];
3 j = a[1];
4 a[0] = 0;
```

Among them, "[]" is the array index, with a[0] as the first elements in the array.

For example, now we define an array b[] below:

```
1 int b[] = {5, 6, 7, 8};
```

The value of each element in array b[] is as follows:

b[0]	b[1]	b[2]	b[3]
5	6	7	8

This is just the use of one-dimensional array. And there are two-dimensional arrays, three-dimensional arrays, and multi-dimensional arrays. Readers interested of this part can develop your own learning.

Loop

The loop statement is used to perform repetitive work such as the initialization to all the elements of an array.

```
1 while(expression)
2     functionX();
```

When there is more than one statement to be executed, the form is as follows:

```
1 while(expression) {
2     functionX();
3     functionY();
4 }
```

The first step of the execution is judging the expression inside "()". If the result is false, the statements inside "{}" will not be executed; if result is true, the statements will be executed.

```
1 int i = 0;
2 while (i < 2)
3     i = i + 1;
4 i = 5;
```

First time: i<2, i=0 is tenable, execute i=i+1, then i=1;

Second time: i<2, i=1 is tenable, execute i=i+1, then i=2;

Third time: i<2, i=2 is not tenable, execution of loop statements is completed. Statement i=5 will be executed next.

"do while" and "while" is similar. The difference is that the loop statements of "do while" is executed before judging expression. The result of the judgment will decide whether or not to go on the next execution:

```
1 do {
2     functionX();
3 } while (expression);
```

"for" is another loop statement, and its form is as follows:

```
1 for (expression1; expression2; expression 3)
2     functionX();
```

When there is more than one statement to be executed, the form is as follows:

```
1 for (expression 1; expression 2; expression 3) {
2     functionX();
3     functionY();
4 }
```

Expression 1 is generally used to initialize variables; expression 2 is a judgement which is used to decide whether or not to execute loop statements; the expression 3 is generally used to change the value of variables.

For example:

```
1 int i = 0, j = 0;
2 for (i = 0; i < 2; i++)
3     j++;
4 i = 5;
```

First time: $i=0$, $i<2$ is tenable, execute $j++$, and execute $i++$, then $i=1$, $j=1$;

Second time: $i=1$, $i<2$ is tenable, execute $j++$, and execute $i++$, then $i=2$, $j=2$;

Third time: $i<2$ is tenable, $i=2$ is not tenable. The execution of loop statements is completed. Statement $i=5$ will be executed next.

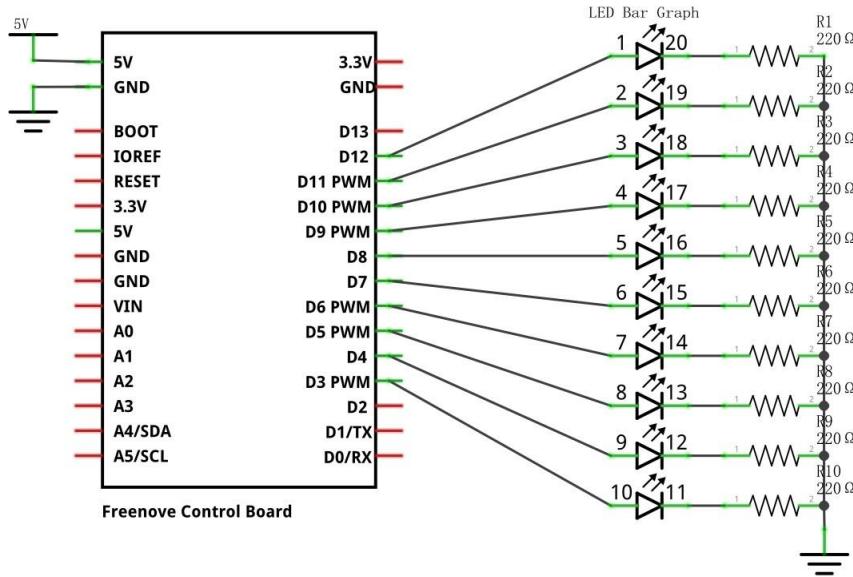
Operator $++$, $--$

" $i++$ " is equivalent to " $i=i+1$ ". And " $i--$ " equivalent to " $i=i-1$ ".

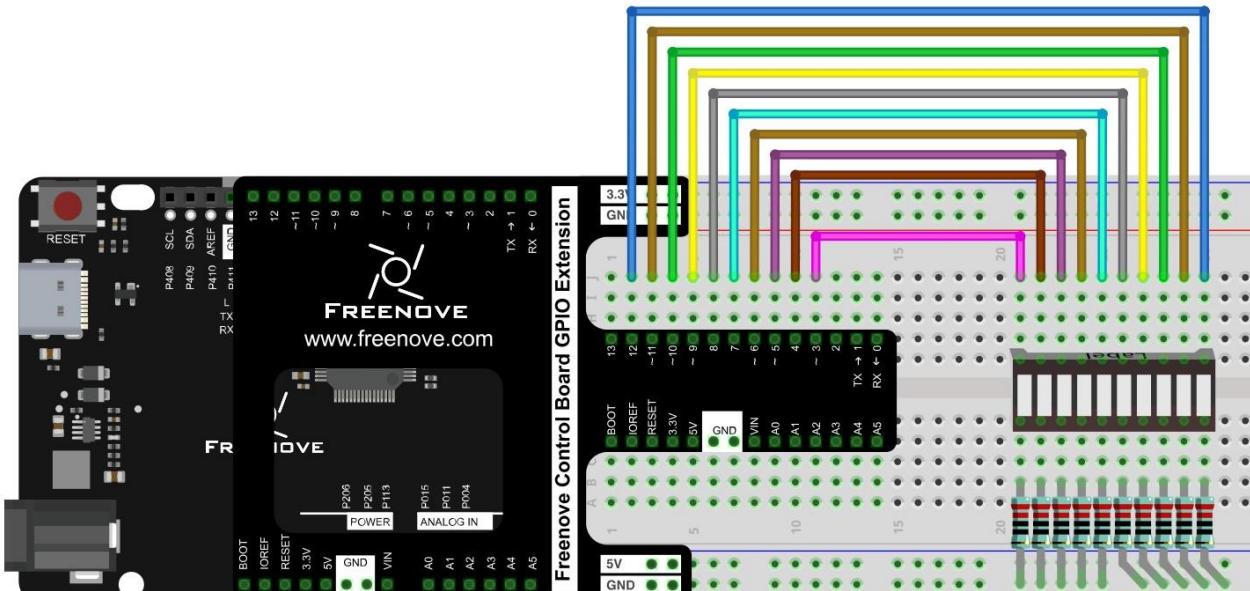
Circuit

Let us use pin 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 of the control board to drive LED bar graph.

Schematic diagram



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



Sketch

Now let us complete the sketch to control LED bar graph.

Sketch 3.1.1

First, write a sketch to achieve the LED light water.

```
1 const int ledCount = 10;      // the number of LEDs in the bar graph
2
3 // an array of pin numbers to which LEDs are attached
4 int ledPins[] = { 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
5
6 void setup() {
7     // loop over the pin array and set them all to output:
8     for (int i = 0; i < ledCount; i++) {
9         pinMode(ledPins[i], OUTPUT);
10    }
11 }
12
13 void loop() {
14     // the ith LED of LED bar graph will light up in turn
15     for (int i = 0; i < ledCount; i++) {
16         barGraphDisplay(i);
17     }
18 }
19
20 void barGraphDisplay(int ledOn) {
21     // make the "ledOn"th LED of LED bar graph on and the others off
22     for (int i = 0; i < ledCount; i++) {
23         if (i == ledOn)
24             digitalWrite(ledPins[i], HIGH);
25         else
26             digitalWrite(ledPins[i], LOW);
27     }
28     delay(100);
29 }
```

Firstly, let us define a read-only variable to record the number of LEDs as the number of times in the loop.

```
1 const int ledCount = 10;      // the number of LEDs in the bar graph
```

Read-only variable

"const" keyword is used to define read-only variables, which is called in the same way as other variables. But read-only variables can only be assigned once.

Then we define an array used to store the number of pins connected to LED bar graph. So it is convenient to

manipulate arrays to modify the pin number.

```
3 // an array of pin numbers to which LEDs are attached
4 int ledPins[] = { 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

Use loop statement to set the pins to output mode in function setup().

```
6 void setup() {
7     // loop over the pin array and set them all to output:
8     for (int i = 0; i < ledCount; i++) {
9         pinMode(ledPins[i], OUTPUT);
10    }
11 }
```

Define a function to turn ON a certain LED on the LED bar graph and turn OFF the other LEDs.

```
20 void barGraphDisplay(int ledOn) {
21     // make the "ledOn"th LED of LED bar graph on and the others off
22     for (int i = 0; i < ledCount; i++) {
23         if (i == ledOn)
24             digitalWrite(ledPins[i], HIGH);
25         else
26             digitalWrite(ledPins[i], LOW);
27     }
28     delay(100);
29 }
```

Finally, when the above function is called cyclically, there will be a formation of flowing water lamp effect in LED bar graph.

```
13 void loop() {
14     // the ith LED of LED bar graph will light up in turn
15     for (int i = 0; i < ledCount; i++) {
16         barGraphDisplay(i);
17     }
18 }
```

Verify and upload the code, then you will see the LED bar graph flashing like flowing water.



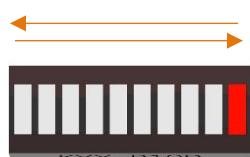
Sketch 3.1.2

Then modify the code to create a reciprocating LED light water.

```
1 const int ledCount = 10;      // the number of LEDs in the bar graph
2
3 // an array of pin numbers to which LEDs are attached
4 int ledPins[] = { 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
5
6 void setup() {
7     // loop over the pin array and set them all to output:
8     for (int i = 0; i < ledCount; i++) {
9         pinMode(ledPins[i], OUTPUT);
10    }
11 }
12
13 void loop() {
14     // makes the "i"th LED of LED bar graph bright in turn
15     for (int i = 0; i < ledCount; i++) {
16         barGraphDisplay(i);
17     }
18     // makes the "i"th LED of LED bar graph bright in reverse order
19     for (int i = ledCount; i > 0; i--) {
20         barGraphDisplay(i - 1);
21     }
22 }
23
24 void barGraphDisplay(int ledOn) {
25     // make the "ledOn"th LED of LED bar graph on and the others off
26     for (int i = 0; i < ledCount; i++) {
27         if (i == ledOn)
28             digitalWrite(ledPins[i], HIGH);
29         else
30             digitalWrite(ledPins[i], LOW);
31     }
32     delay(100);
33 }
```

We have modified the code inside the function loop() to make the LED bar graph light up in one direction, and then in a reverse direction repeatedly.

Verify and upload the code, then you will see a reciprocating LED water light on LED bar graph.



Chapter 4 LED Blink Smoothly

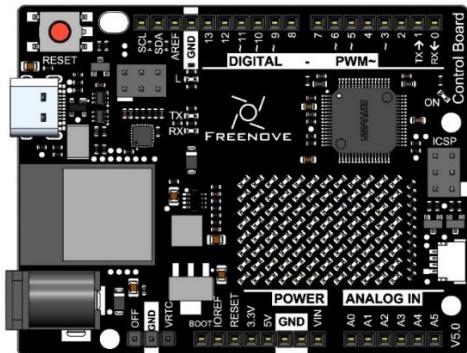
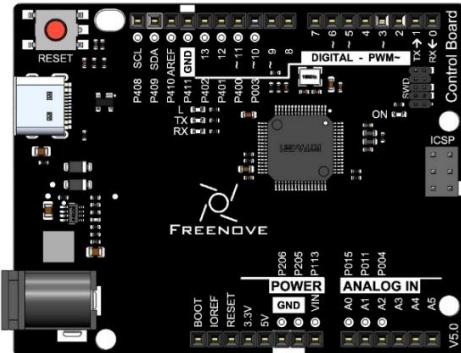
In the previous chapter, we have used Sketch to control up to 10 LEDs on the control board to blink and learned some basic knowledge of programming. Now, let us try to make LED emit different brightness of light.

Project 4.1 LEDs Emit Different Brightness

Now, let us use control board to make 4 LED emit different brightness of light.

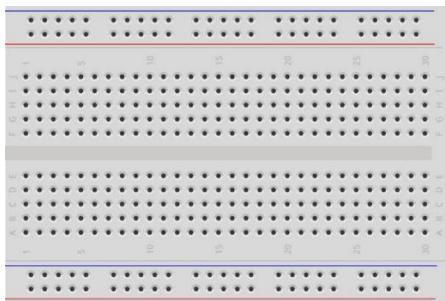
Component List

Control board x1

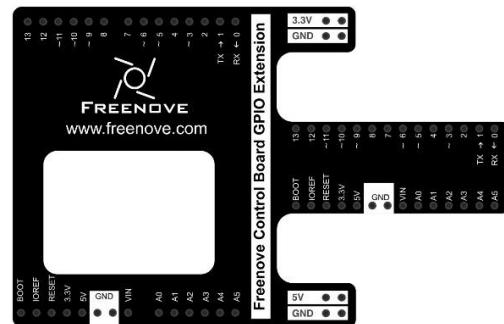


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



LED x4



Resistor 220Ω x4



Jumper M/M x8



Circuit Knowledge

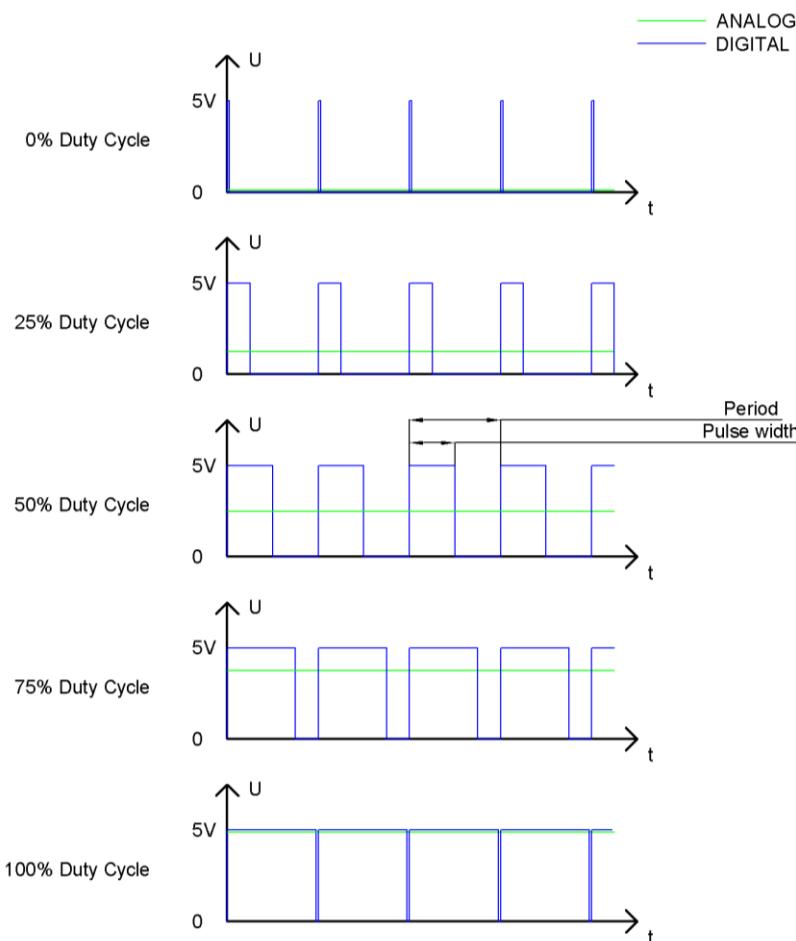
At first, let us learn how to use the circuit to make LED emit different brightness of light,

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Digital 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.

Code Knowledge

We will use new code knowledge in this section.

Return value of function

We have learned and used the function without return value, now we will learn how to use the function with return value. A function with return value is shown as follow:

```

1 int sum(int i, int j) {
2     int k = i + j;
3     return k;
4 }
```

"int" declares the type of return value of the function sum(int i, int j). If the type of the return value is void, the function does not return a value.

One function can only return one value. It is necessary to use the return statement to return the value of function.

When the return statement is executed, the function will return immediately regardless of code behind the return statement in this function.

A function with return value is called as follows:

```

1 int a = 1, b = 2, c = 0;
2 c = sum(1, 2);           // after the execution the value of c is 3
```

A function with a return value can also be used as a parameter of functions, for example:

```
1 delay(sum(100, 200));
```

It is equivalent to the following code:

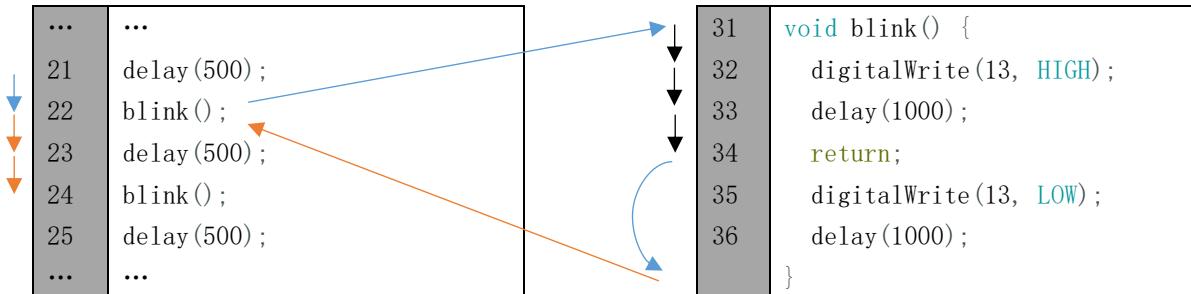
```
1 delay(300);
```

return

We have learned the role of the return statement in a function with a return value. It can also be used in functions without a return value, and there is no data behind the return keyword:

```
1 return;
```

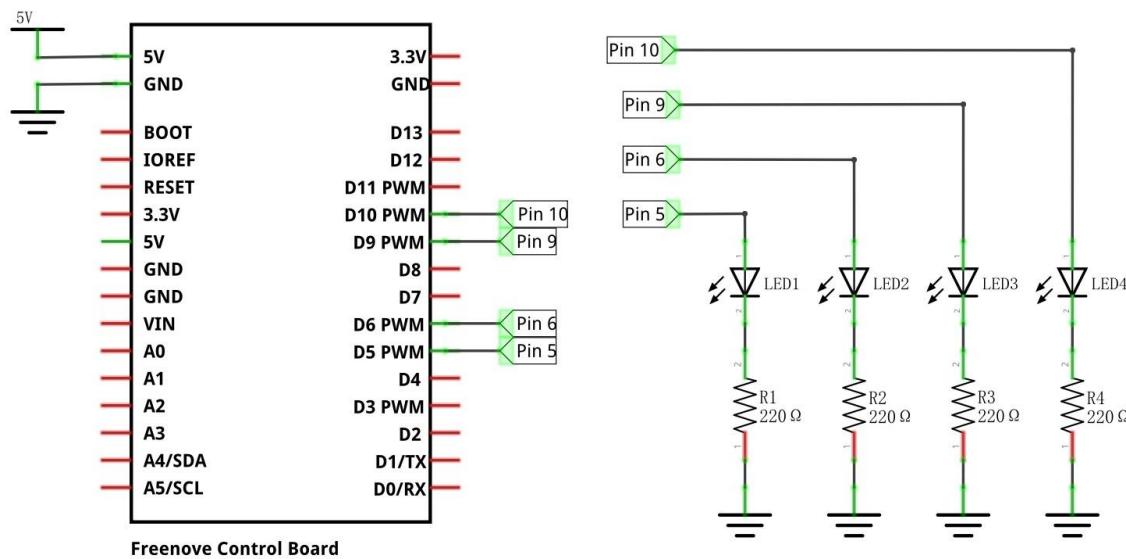
In this case, when the return statement is executed, the function will immediately end its execution rather than return to the end of the function. For example:



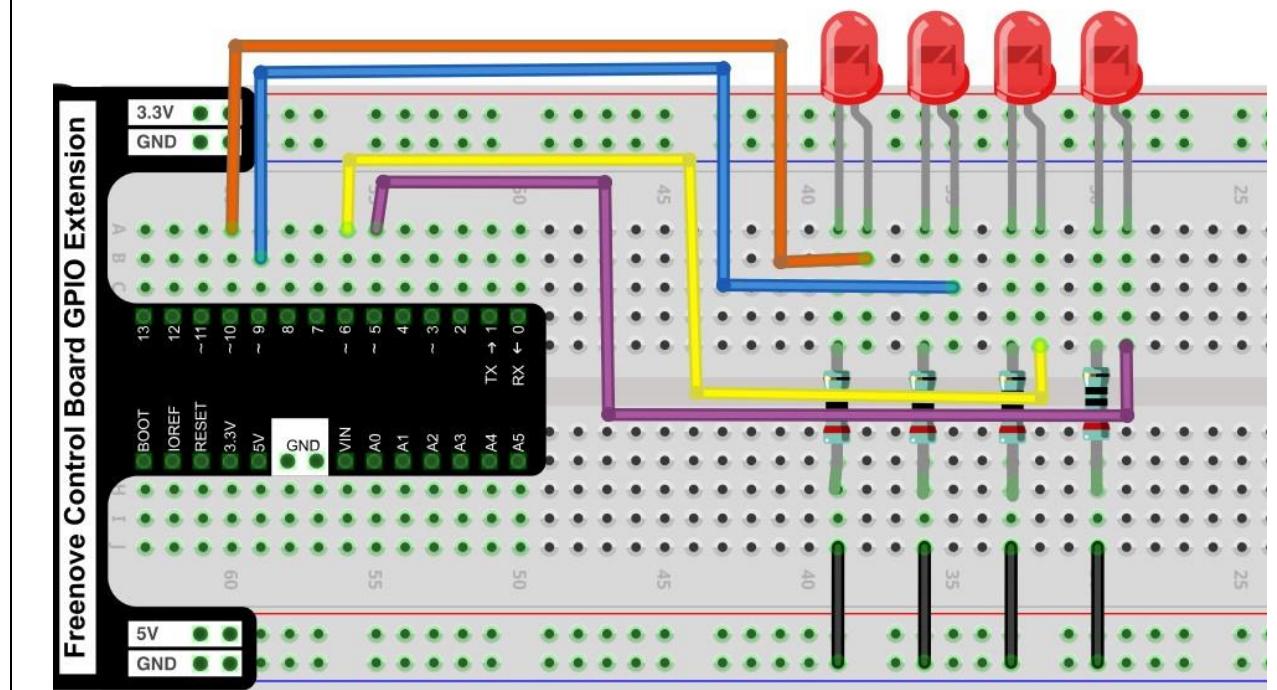
Circuit

Use pin 5, 6, 9, 10 on the control board to drive 4 LEDs.

Schematic diagram



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



Sketch

Sketch 4.1.1

Now let us use sketch to make 4 LEDs emit different brightness of light. We will transmit signal to make the 4

Need help? Contact support@freenove.com

ports connected to LEDs output the PWM waves with duty cycle of 2%, 10%, 50%, and 100% to let the LEDs emit different brightness of the light.

```

1 // set pin numbers:
2 int ledPin1 = 5,           // the number of the LED1 pin
3     ledPin2 = 6,           // the number of the LED2 pin
4     ledPin3 = 9,           // the number of the LED3 pin
5     ledPin4 = 10;          // the number of the LED4 pin
6
7 void setup() {
8     // initialize the LED pin as an output:
9     pinMode(ledPin1, OUTPUT);
10    pinMode(ledPin2, OUTPUT);
11    pinMode(ledPin3, OUTPUT);
12    pinMode(ledPin4, OUTPUT);
13 }
14
15 void loop()
16 {
17     // set the ports output PWM waves with different duty cycle
18     analogWrite(ledPin1, map(2, 0, 100, 0, 255));
19     analogWrite(ledPin2, map(10, 0, 100, 0, 255));
20     analogWrite(ledPin3, map(50, 0, 100, 0, 255));
21     analogWrite(ledPin4, map(100, 0, 100, 0, 255));
22 }
```

After the initialization of the 4 ports, we set the ports to output PWM waves with different duty cycle. Take ledPin1 as an example, firstly map 2% to the range of 0-255, and then output the PWM wave with duty cycle of 2%,

1	<code>analogWrite(ledPin1, map(2, 0, 100, 0, 255));</code>
---	--

analogWrite(pin, value)

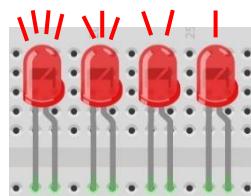
Arduino IDE provides the function, `analogWrite(pin, value)`, which can make ports directly output PWM waves. Only the digital pin marked with "˜" symbol on the control board can use this function to output PWM waves. In the function called `analogWrite(pin, value)`, the parameter "pin" specifies the 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.

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.

Verify and upload the code, and you will see the 4 LEDs emit light with different brightness.



Project 4.2 LED Blinking Smoothly

We will learn how to make a LED blink smoothly, that is, breathing light.

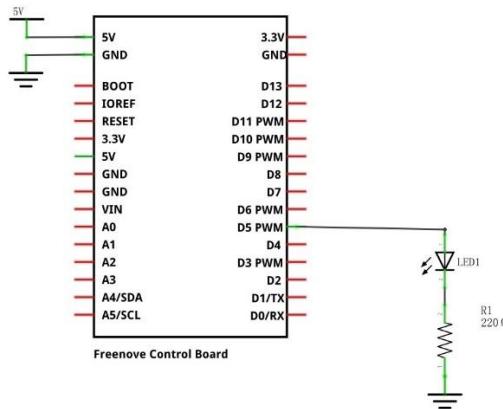
Component List

The Component list is basically the same as those in last section. And we need to get rid of a few LEDs and resistors.

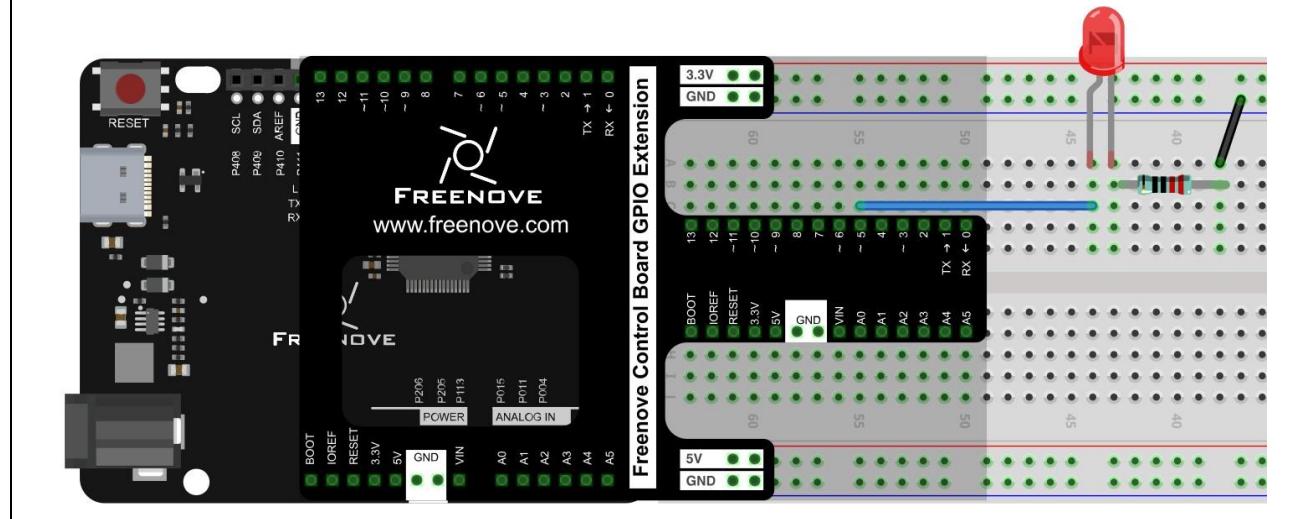
Circuit

Remove some LEDs and resistors connected to pin 6, 9, 10 on the control board in the circuit of the previous section.

Schematic diagram



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



Sketch

Sketch 4.2.1

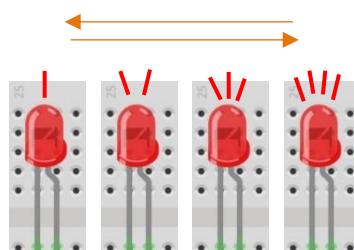
Now complete the sketch to make brightness of LED change from dark to bright, and then from bright to dark. That is to make the duty cycle of the PWM wave change from 0%-100%, and then from 100%-0% cyclically.

```

1 // set pin numbers:
2 int ledPin = 5;           // the number of the LED pin
3
4 void setup() {
5     // initialize the LED pin as an output:
6     pinMode(ledPin, OUTPUT);
7 }
8
9 void loop() {
10    // call breath() cyclically
11    breath(ledPin, 6);
12    delay(500);
13 }
14
15 void breath(int ledPin, int delayMs) {
16     for (int i = 0; i <= 255; i++) { // "i" change from 0 to 255
17         analogWrite(ledPin, i);      // corresponding duty cycle change from 0%-100%
18         delay(delayMs);          // adjust the rate of change of brightness
19     }
20     for (int i = 255; i >= 0; i--) { // "i" change from 255 to 0
21         analogWrite(ledPin, i);      // corresponding duty cycle change from 0%-100%
22         delay(delayMs);          // adjust the rate of change in brightness
23     }
24 }
```

Through two “for” loops, the duty cycle of the PWM wave changes from 0% to 100%, and then from 100% to 0% cyclically. `delay(ms)` function is used to control the change rate in the “for” loop, and you can try to modify the parameters to modify the change rate of brightness.

Verify and upload the code, then you will see that the brightness of the LED changes from dark to light, and from the light to dark cyclically.



Chapter 5 Control LED with Push Button Switch

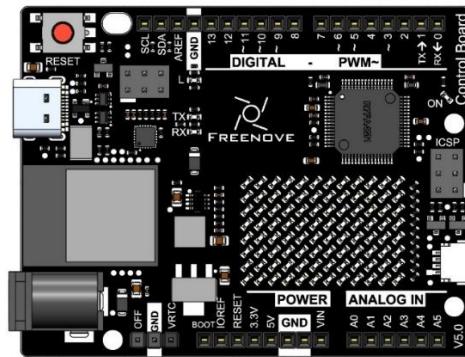
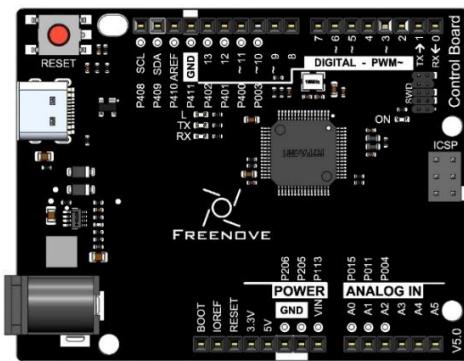
In the previous chapter, we have used the control board to output signals to make 10 LEDs flash, and make one LED emit different brightness. Now, let's learn how to get an input signal.

Project 5.1 Control LED with Push Button Switch

We will use the control board to get the status of the push button switch, and show it through LED.

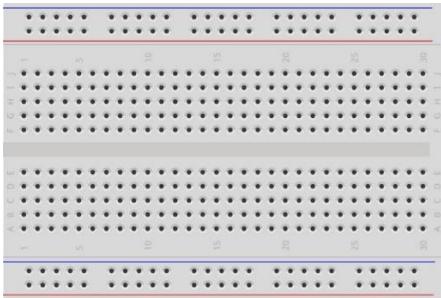
Component List

Control board x1

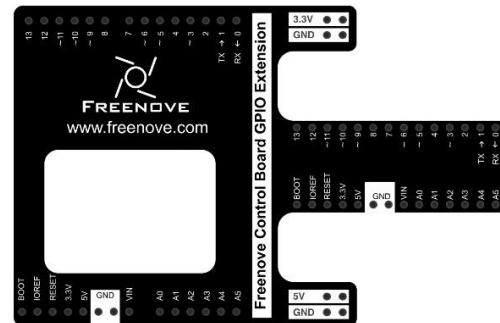


or

Breadboard x1



GPIO Extension Board x1



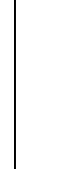
USB cable x1



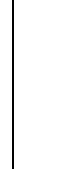
LED x1



Resistor 220Ω x1



Resistor 10kΩ x2



Push button Switch x1



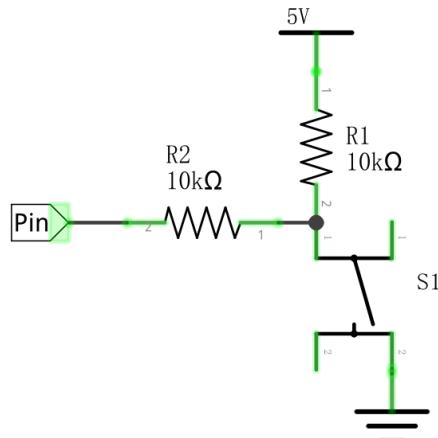
Jumper M/M x4



Circuit Knowledge

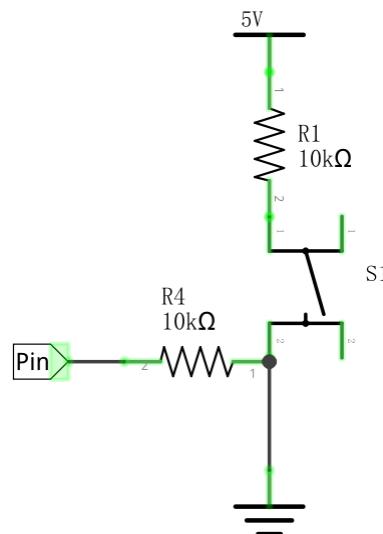
Connection of Push Button Switch

In Chapter 1, we connect push button switch directly to power up the circuit to control the LED to turn on or off. In digital circuits, we need to use the push button switch as an input signal. The recommended connection is as follows:



In the above circuit diagram, when the button is not pressed, 5V (high level) will be detected by control board port; and 0V (low level) when the button is pressed. The role of Resistor R2 here is to prevent the port from being set to output high level by accident. Without R2, the port could be connected directly to the cathode and cause a short circuit when the button is pressed.

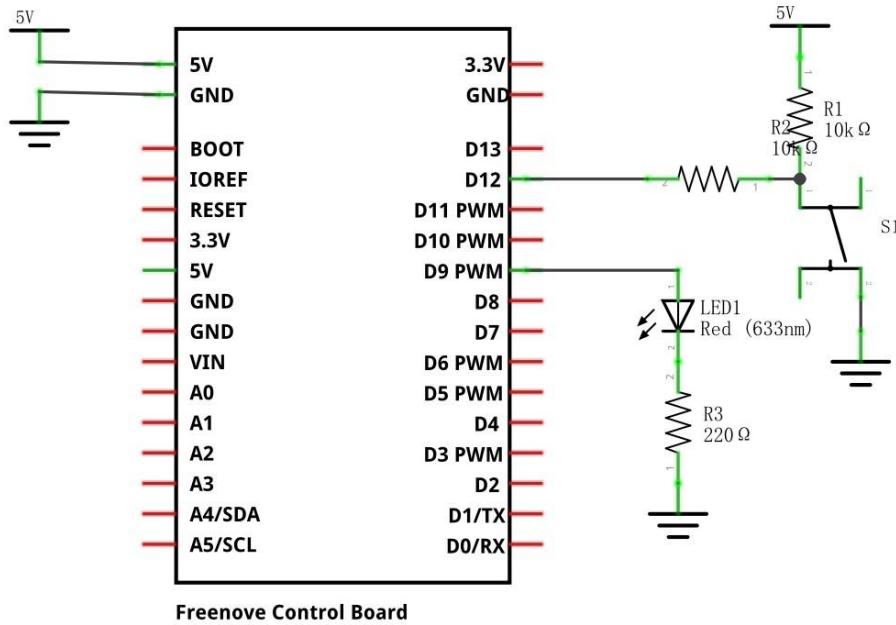
The following diagram shows another connection, in which the level detected by the control board port is opposite to the above diagram, whenever the button is pressed or not.



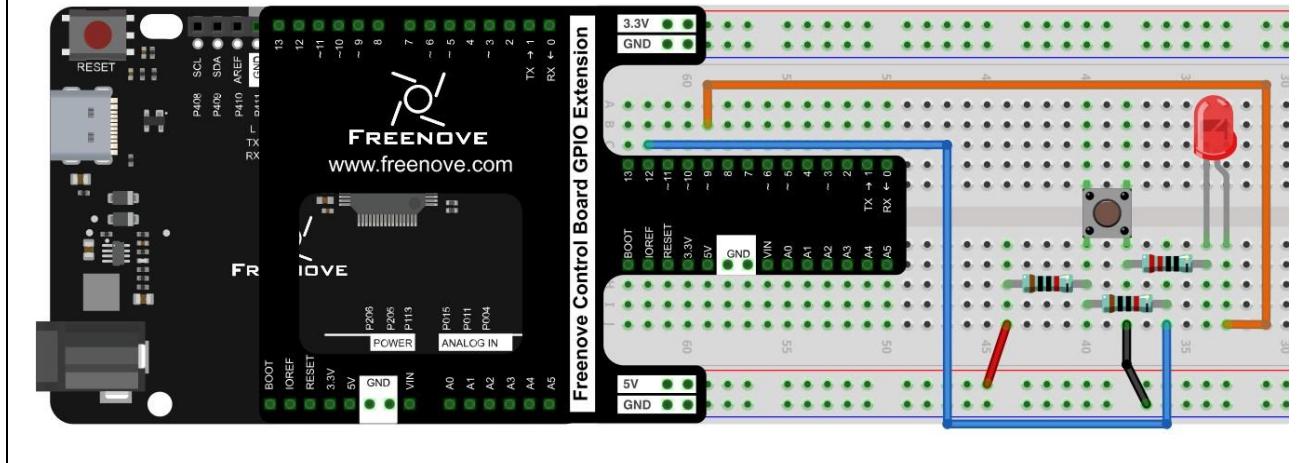
Circuit

Use pin 12 of control board to detect the status of push button, and pin 9 to drive LED.

Schematic diagram



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



Sketch

Sketch 5.1.1

Now, write code to detect the state of push button, and show it through LED.

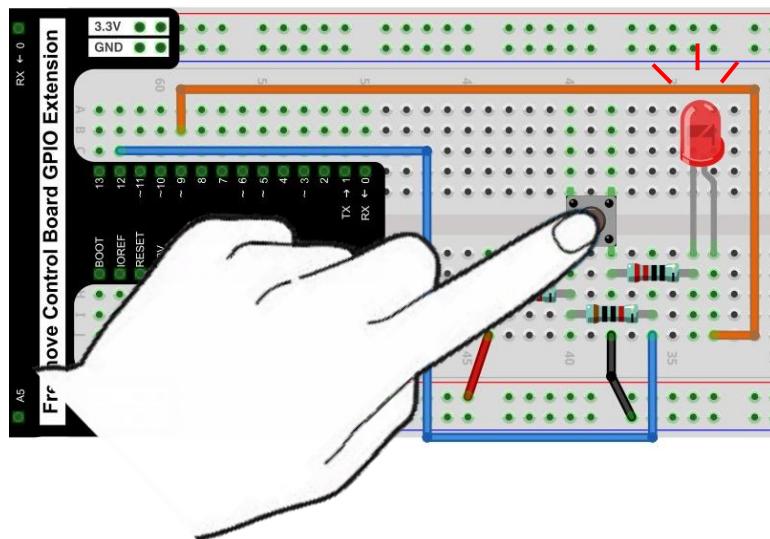
```
1 int buttonPin = 12; // the number of the push button pin
2 int ledPin = 9; // the number of the LED pin
3
4 void setup() {
5     pinMode(buttonPin, INPUT); // set push button pin into input mode
6     pinMode(ledPin, OUTPUT); // set LED pin into output mode
7 }
8
9 void loop() {
10    if (digitalRead(buttonPin) == HIGH) // if the button is not pressed
11        digitalWrite(ledPin, LOW); // switch off LED
12    else // if the button is pressed
13        digitalWrite(ledPin, HIGH); // switch on LED
14 }
```

After the port is initialized, the LED will be turned on or off in accordance with the state of the pin connected to push button switch.

digitalRead(pin)

Arduino IDE provides a function `digitalRead(pin)` to obtain the state of the port pin. The return value is HIGH or LOW, that is, high level or low level.

Verify and upload the code, press the button, LED lights up; release the button, LED lights off.



Project 5.2 Change LED State with Push Button Switch

In the previous section, we have finished the experiment that LED lights ON when push button switch is pressed, and lights OFF as soon as it's released. Now, let's try something new: each time you press the button down, the state of LED will be changed.

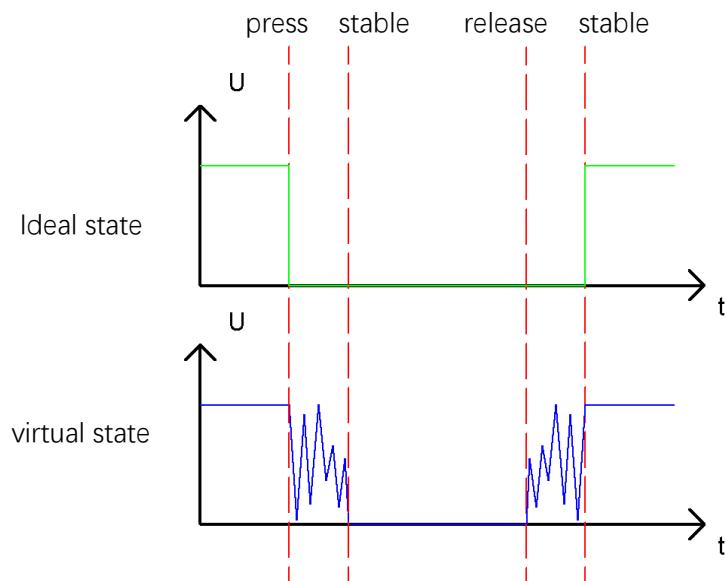
Component List

Same with the previous section.

Circuit Knowledge

Debounce a push button switch

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 stabilizes in a new 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).

Circuit

Same with the previous section.

Sketch

Sketch 5.2.1

Now, write a code to detect the state of the push button switch. Every time you pressed it, the state of LED will be changed.

```
1 int buttonPin = 12; // the number of the push button pin
2 int ledPin = 9; // the number of the LED pin
3 boolean isLighting = false; // define a variable to save the state of LED
4
5 void setup() {
6     pinMode(buttonPin, INPUT); // set push button pin into input mode
7     pinMode(ledPin, OUTPUT); // set LED pin into output mode
8 }
9
10 void loop() {
11     if (digitalRead(buttonPin) == LOW) { // if the button is pressed
12         delay(10); // delay for a certain time to skip the bounce
13         if (digitalRead(buttonPin) == LOW) { // confirm again if the button is pressed
14             reverseLED(); // reverse LED
15             while (digitalRead(buttonPin) == LOW); // wait for releasing
16             delay(10); // delay for a certain time to skip bounce when the button is released
17         }
18     }
19 }
20
21 void reverseLED() {
22     if (isLighting) { // if LED is lighting,
23         digitalWrite(ledPin, LOW); // switch off LED
24         isLighting = false; // store the state of LED
25     }
26     else { // if LED is off,
27         digitalWrite(ledPin, HIGH); // switch LED
28         isLighting = true; // store the state of LED
29     }
30 }
```

Verify and upload the code, then each time you press the button, LED changes its state accordingly.

When judging the push button switch state, if it is detected as "pressed down", wait for a certain time to detect again to eliminate the effect of bounce. When the state is stable, released push button switch, and wait for a certain time to eliminate the effect of bounce after it is released.

Chapter 6 Serial

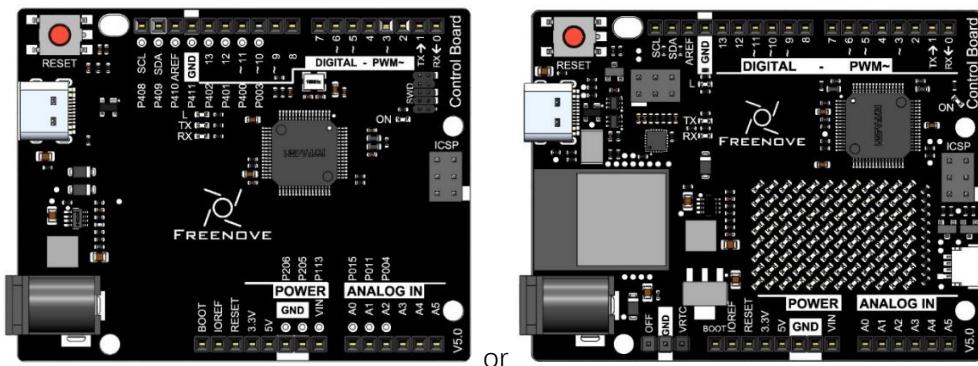
In this chapter, we will get into serial communication, which is a more advanced means of communication.

Project 6.1 Send Data through Serial

We will use the serial port on control board to send data to computer.

Component List

Control board x1



or

USB cable x1



Code Knowledge

Bit and Byte

As mentioned earlier, computers use a binary signal. A binary signal is called 1 bit, and 8 bits organized in order is called 1 byte. Byte is the basic unit of information in computer storage and processing. 1 byte can represent $2^8=256$ numbers, that is, 0-255. For example:

As to binary number 10010110, "0" usually presents the lowest value in code.

Sequence	7	6	5	4	3	2	1	0
Number	1	0	0	1	0	1	1	0

When a binary number need to be converted to decimal number, first, the nth number of it need be multiplied by n power of 2, then sum all multiplicative results. Take 10010110 as an example:

$$1*2^7+0*2^6+0*2^5+1*2^4+0*2^3+1*2^2+1*2^1+0*2^0=150$$

We can make a decimal number divided by 2 to convert it to binary number. Get the integer quotient for the next iteration and get the remainder for the binary digit. Repeat the steps until the quotient is equal to zero. Arrange all remainders from right to left in a line. Then we complete the conversion. For example:

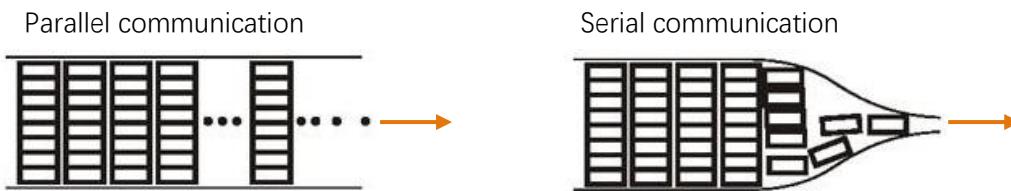
	Remainder	Sequence
2 150 0	0
2 75 1	1
2 37 1	2
2 18 0	3
2 9 1	4
2 4 0	5
2 2 0	6
2 1 1	7
	0	

The result is 10010110.

Circuit Knowledge

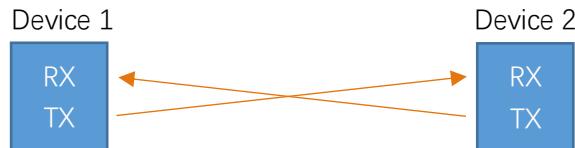
Serial and parallel communication

Serial communication uses one data cable to transfer data one bit by another in turn, while parallel communication means that the data is transmitted simultaneously on multiple cables. Serial communication takes only a few cables to exchange information between systems, which is especially suitable for computers to computer, long distance communication between computers and peripherals. Parallel communication is faster, but it requires more cables and higher cost, so it is not appropriate for long distance communication.



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 of two devices use is as follows:

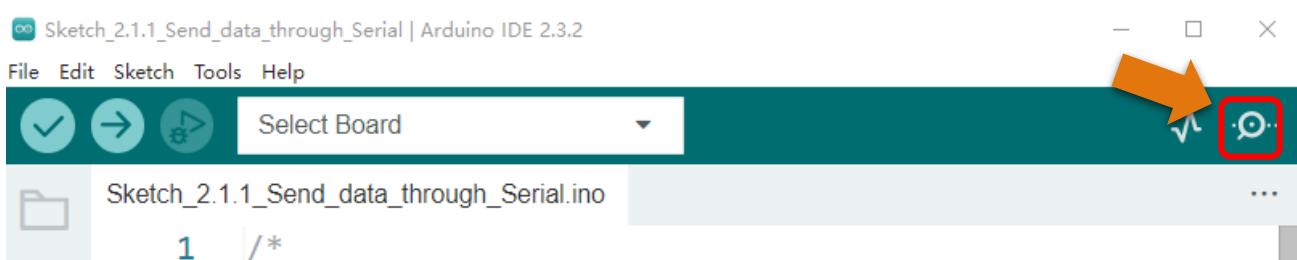


Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

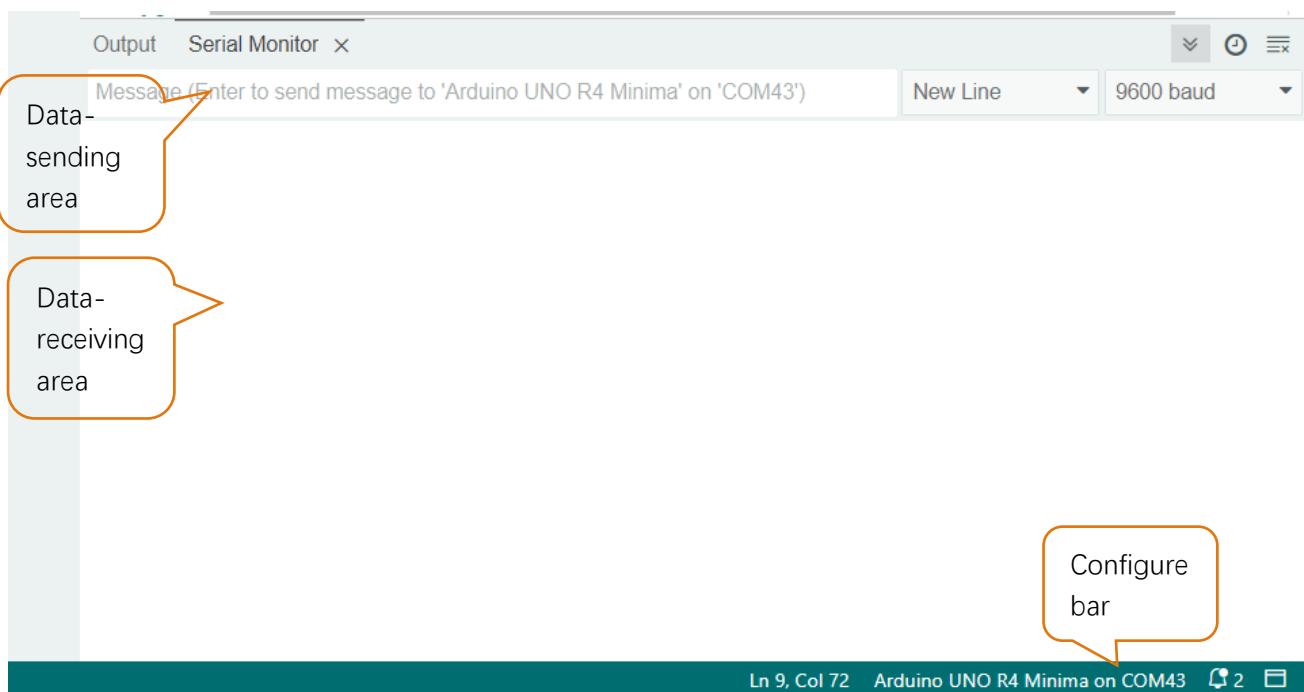
Serial port on Control board

Control board has integrated USB to serial transfer, so it can communicate with computer when USB cable get connected to it. Arduino IDE also uploads code to control board through the serial connection.

Computer identifies serial devices connected to your computer as COMx. We can use the Serial Monitor window of Arduino IDE to communicate with control board, connect control board to computer through the USB cable, choose the correct device, and then click the Serial Monitor icon to open the Serial Monitor window.

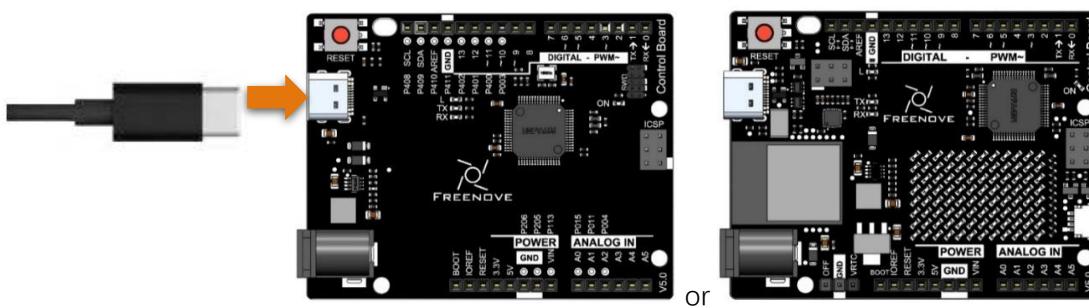


Interface of Serial Monitor window is as follows. If you can't open it, make sure control board had been connected to the computer, and choose the correct serial port in the menu bar "Tools".



Circuit

Connect control board to the computer with USB cable.



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

Sketch

Sketch 6.1.1

Now, write code to send some texts to the Serial Monitor window

```

1 int counter = 0; // define a variable as a data sending to serial port
2
3 void setup() {
4     Serial.begin(9600);           // initialize the serial port, set the baud rate to 9600
5     Serial.println("UNO is ready!"); // print the string "UNO is ready!"
6 }
```

```

7
8 void loop() {
9   // print variable counter value to serial
10  Serial.print("counter:"); // print the string "counter:"
11  Serial.println(counter); // print the variable counter value
12  delay(500); // wait 500ms to avoid cycling too fast
13  counter++; // variable counter increases 1
14 }
```

setup() function initializes the serial port.

And then continuously sends variable counter values in the loop () function.

Serial class

Class is a C++ language concept. Arduino IDE supports C++ language, which is a language extension. We don't explain specifically the concept here, but only describe how to use it. If you are interested in it, you can learn by yourself. Serial is a class name, which contains variables and functions. You can use the "." operational character to visit class variables and functions, such as:

Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

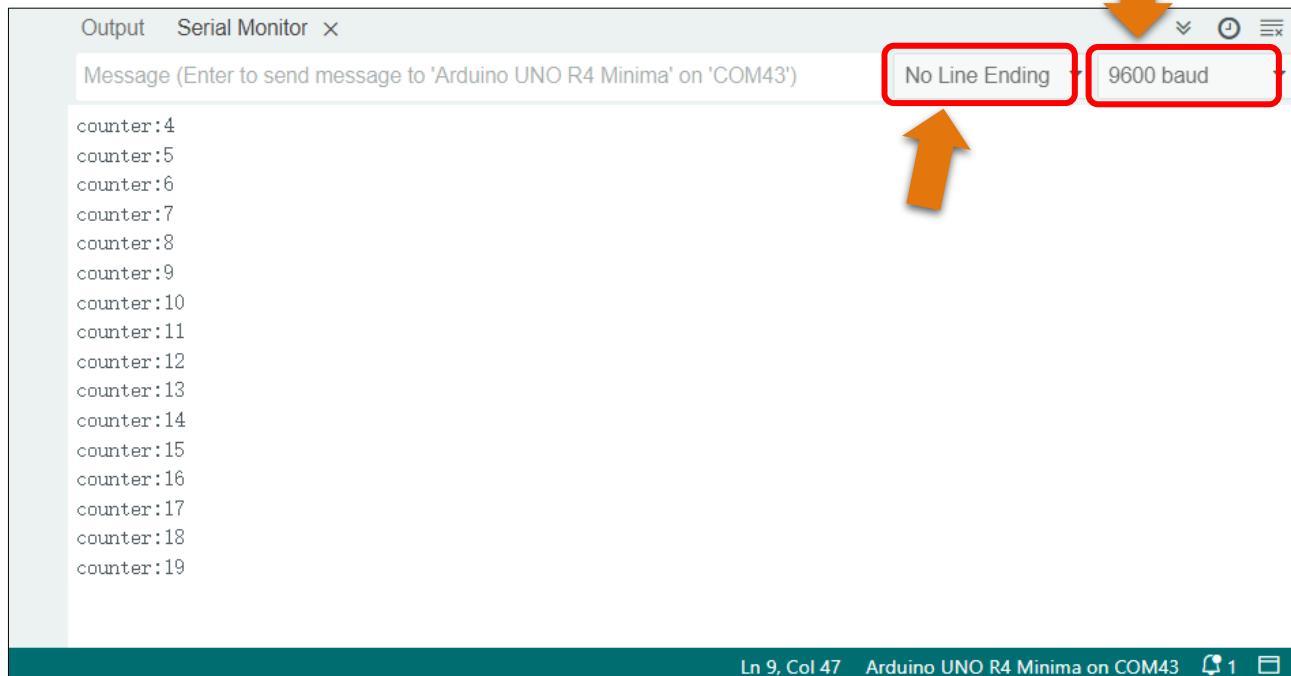
Serial.print(val): Send string, the parameter here is what you want to send;

Serial.println(val): Send newline behind string.

Verify and upload the code, open the Serial Monitor, and then you'll see data sent from control board.

If it is not displayed correctly, check whether the configuration of the Serial Monitor in the lower right corner of the window is correct.

Please note: You need to select "**No Line Ending**"



Project 6.2 Receive Data through Serial Port

In the previous section, we used Serial port on control board to send data to a computer, now we will use it to receive data from computer.

Component List

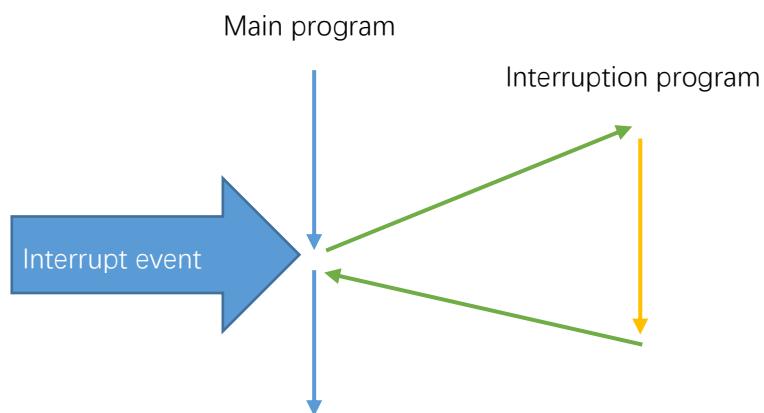
Same with the previous section.

Code Knowledge

Interrupt

An interrupt is a controller's response to an event. The event causing an interrupt is an interrupt source. We'll illustrate the interruption concept. For example, suppose you're watching TV while there is water in your kitchen heating, then you have to check whether the water is boiling or not from time to time, so you can't concentrate on watching TV. But if you have an interrupt, things will be different. Interrupt can work as a warning device for your kettle, which will beep when the water is about to boil. So before the water is boiling, you can focus on watching TV until a beep warning comes out.

Advantages of interrupt here: Processor won't need to check whether the event has happened every now and then, but when an event occurs, it informs the controller immediately. When an interrupt occurs, the processor will jump to the interrupt function to handle interrupt events, then return to where the interrupt occurs after finishing it and go on this program.



Circuit

Same with the previous section.

Sketch

Sketch 6.2.1

Now, write code to receive the characters from Serial Monitor window, and send it back.

```

1  char inChar;      // define a variable to store characters received from serial port
2
3  void setup() {
4      Serial.begin(9600);          // initialize serial port, set baud rate to 9600
5  }
6
7  void loop() {
8      if (Serial.available()) {    // judge whether data has been received
9          inChar = Serial.read(); // read one character
10         Serial.print("received:");
11         Serial.println(inChar); // print the received character
12     }
13 }
```

In the setup() function, we initialize the serial port. Then, the loop() function will continuously detect whether there are data to read. If so, it will read the character and send it back.

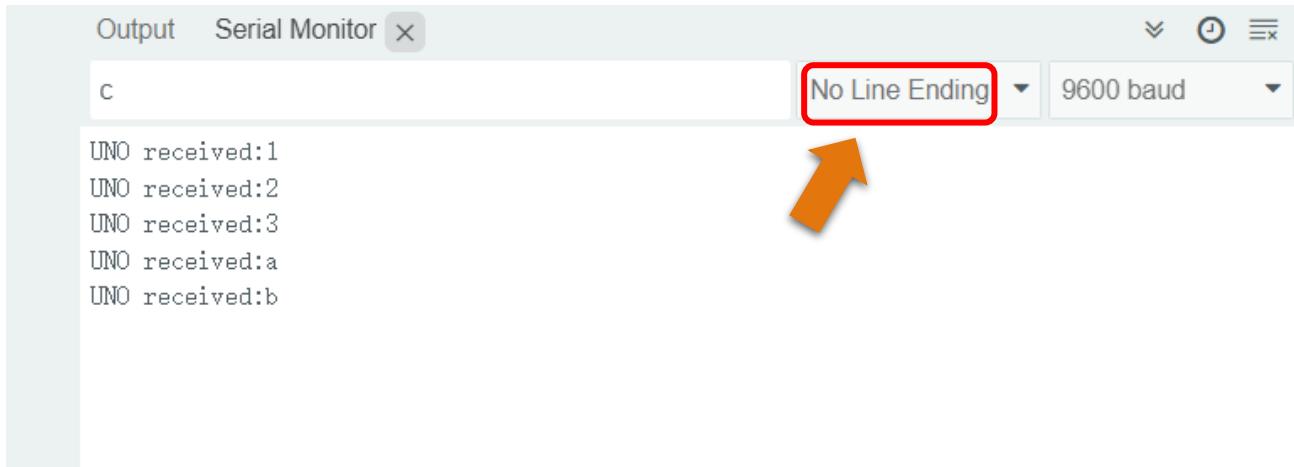
Serial Class

Serial.available(): return bytes of data that need to be read by serial port;

Serial.read(): return 1 byte of data that need to be read by serial port.

Verify and upload the code, open the Serial Monitor, write character in the sending area, click Send button, then you'll see information returned from control board.

Please note: You need to select "**No Line Ending**"



char type

char type variable can represent a character, but it cannot store characters directly. It stores numbers to replace characters. char type occupies 1-byte store area, and uses a value 0-127 to correspond to 128 characters. The corresponding relation between number and character is ruled by ASCII table. For more details of ASCII table, please refer to the appendix of this book.

Example: Define char aChar = 'a', bChar = '0', then the decimal value of aChar is 97, bChar will be 48.

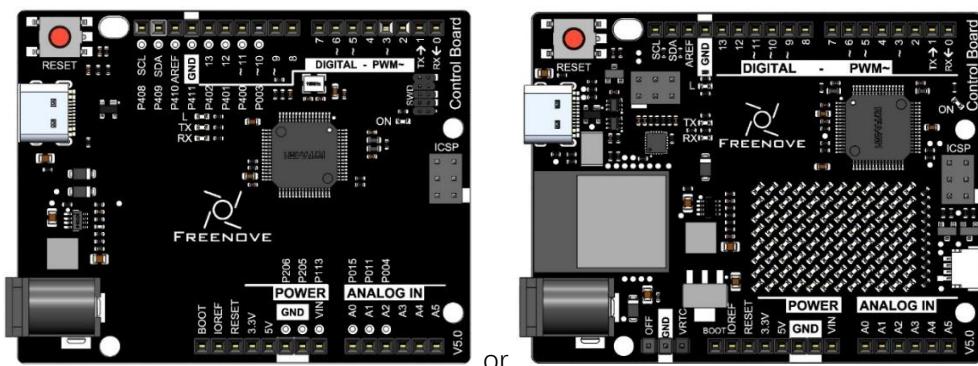
Project 6.3 Application of Serial

We will use the serial port on control board to control one LED.

Component List

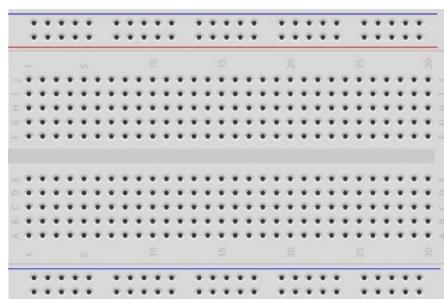
If the kit you bought does not include the following components, you can use only the control board and USB cable to finish this project.

Control board x1



or

Breadboard x1



USB cable x1



Jumper M/M x2



LED x1



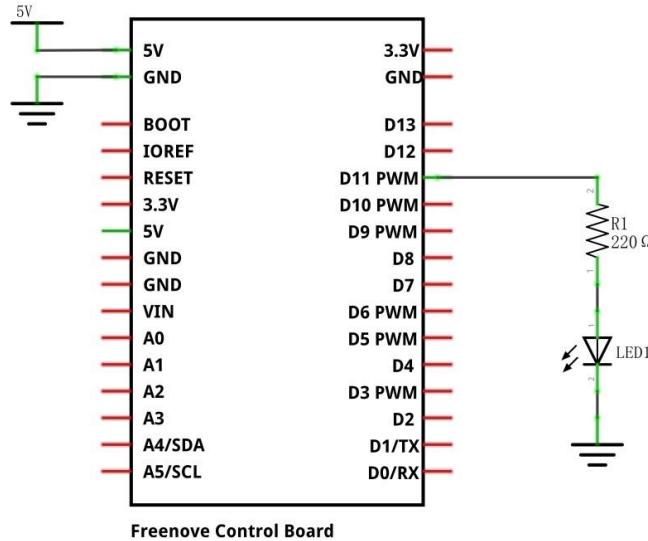
Resistor 220Ω x1



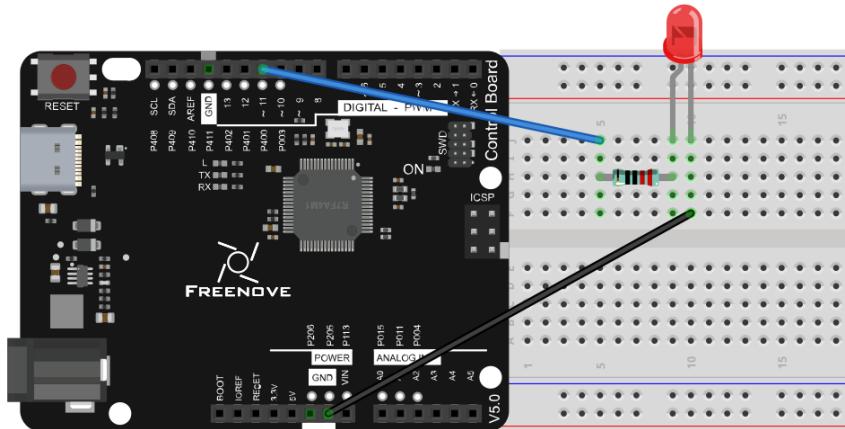
Circuit

Here we will use pin 11 of the control board to output PWM to drive 1 LED.

Schematic diagram



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



If you only use the control board for this project, the LED being controlled is the onboard LED. You only need to change the LED pin to 13.

Sketch

Sketch 6.3.1

Code is basically the same with Sketch 6.2.1. But after receiving the data, control board will convert it into PWM duty cycle of output port.

```

1 int inInt;           // define a variable to store the data received from serial
2 int counter = 0;    // define a variable as the data sending to serial
3 int ledPin = 11;    // the number of the LED pin
4
5 void setup() {
6     pinMode(ledPin, OUTPUT);          // initialize the LED pin as an output
7     Serial.begin(9600);             // initialize serial port, set baud rate to 9600
8     Serial.println("UNO is ready!");   // print the string "UNO is ready!"
9 }
10
11 void loop() {
12     if (Serial.available()) {        // judge whether the data has been received
13         inInt = Serial.parseInt();    // read an integer
14         Serial.print("UNO received:"); // print the string "UNO received:"
15         Serial.println(inInt);       // print the received character
16         // convert the received integer into PWM duty cycle of ledPin port
17         analogWrite(ledPin, constrain(inInt, 0, 255));
18     }
19 }
```

When serial receives data, it converts the data into PWM duty cycle of output port to make LED emit light with corresponding brightness.

Serial Class

`Serial.parseInt()`: Receive an int type number as the return value.

constrain(x, a, b)

Limit x between a and b, if $x < a$, return a; if $x > b$, return b.

Verify and upload the code, open the Serial Monitor, and put a number in the range of 0-255 into the sending area and click the Send button. Then you'll see information returned from control board, meanwhile, LED can emit light with different brightness according to the number you send.

Please note: You need to select "**No Line Ending**"



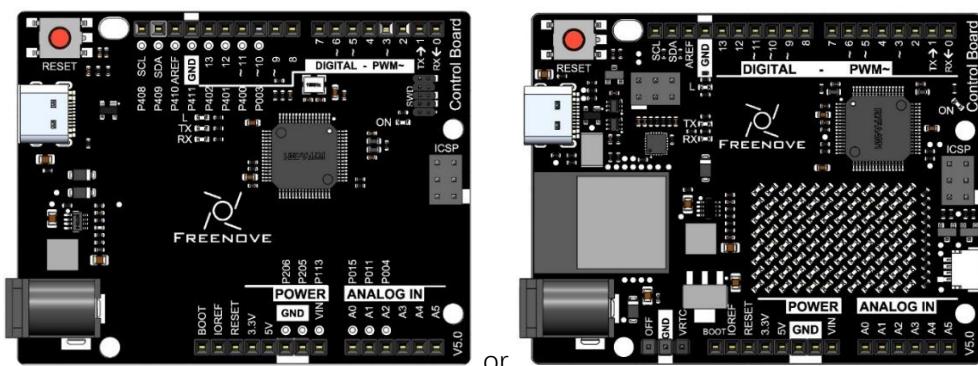
Chapter 7 Timer

In this chapter, we will use the timer of the board, trigger the timer at intervals, thus allowing the serial port to print messages.

Project 7.1 Serial print using timer

Component List

Control board x1



or

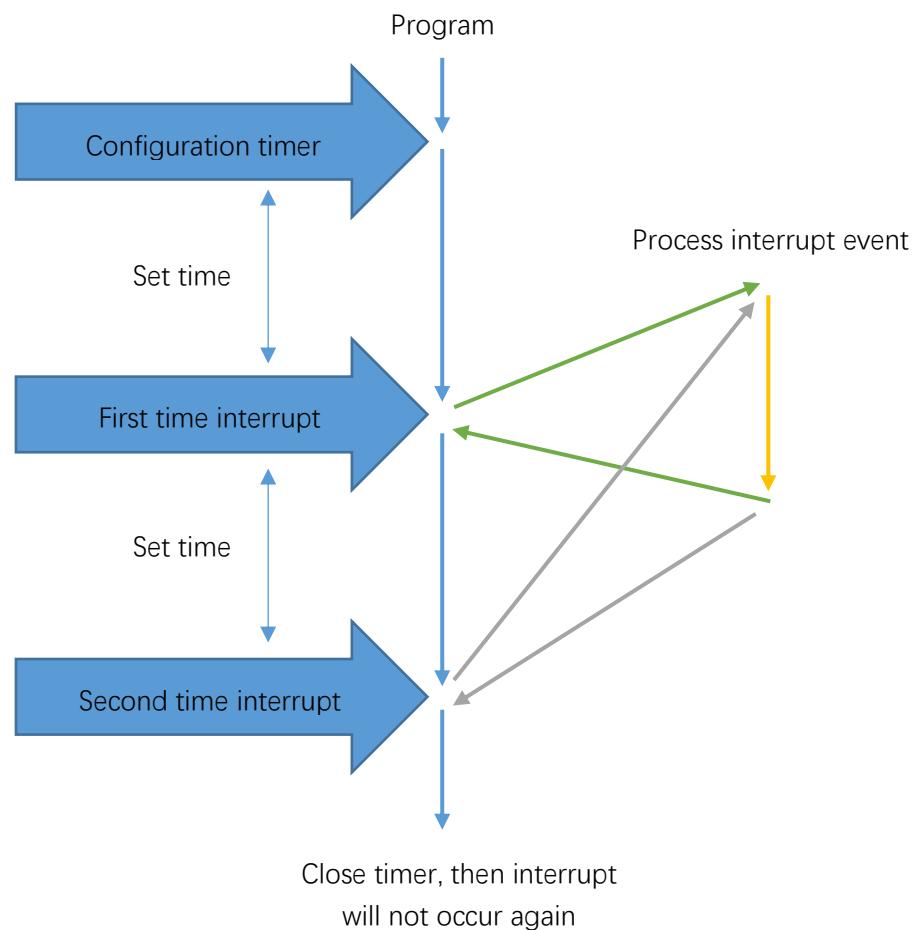
USB cable x1



Code Knowledge

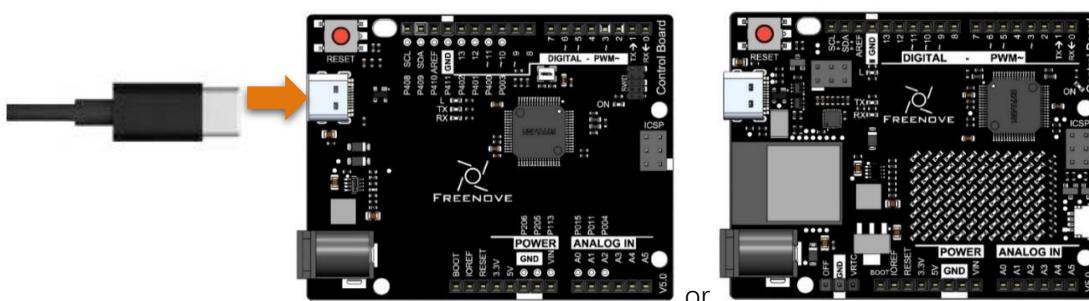
Timer

A Timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted location to go on. If you don't close the timer, interrupt will occur at the intervals you set.



Circuit

Connect control board to the computer with USB cable.



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

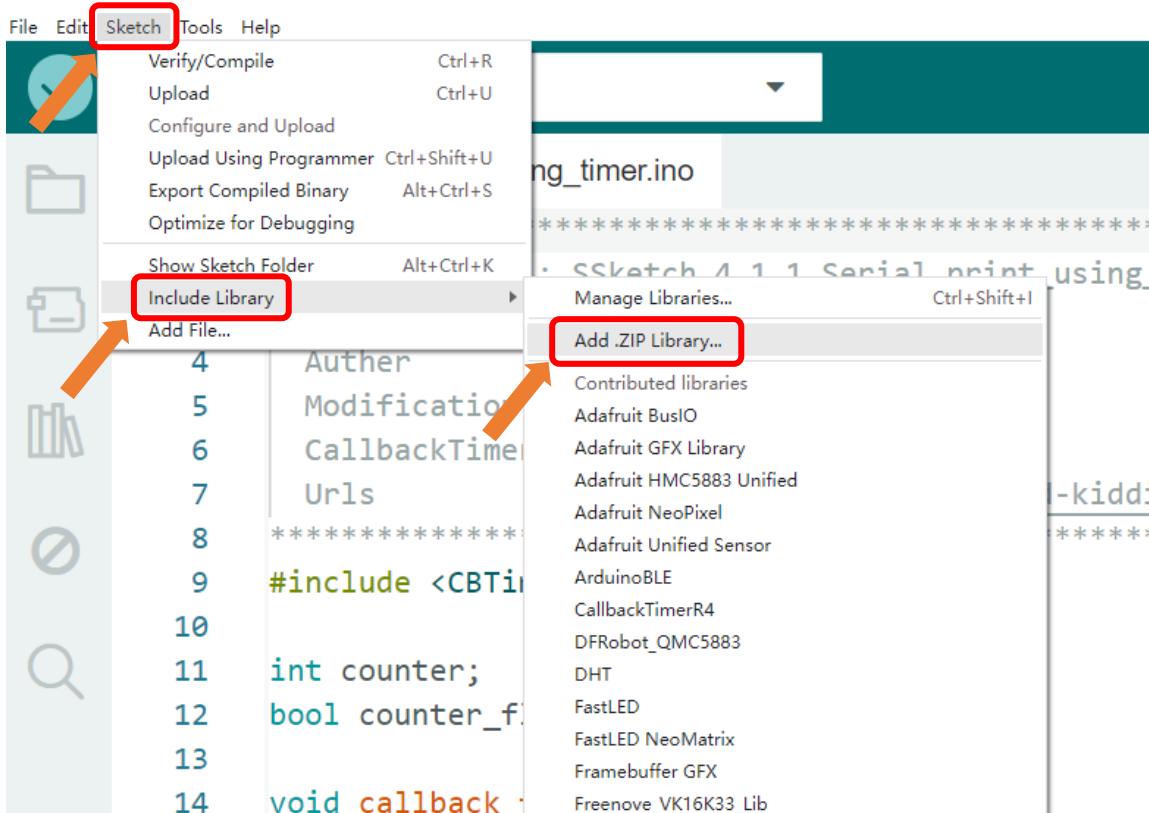
Sketch

This code uses a library named "CallbackTimerR4 ", 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.

How to install the library

open Arduino IDE, click Sketch→Include Library→Add .ZIP Library, In the pop-up window, find the file named ".Libraries/ **CallbackTimerR4-main.zip**" which locates in this directory, and click OPEN.



Sketch 7.1.1

```

1 #include <CBTimer.h>      // Contains CallbackTimer4 Library
2
3 int counter;
4 bool counter_flag;
5
6 void callback_func(void) {
7     counter_flag = true;
8     counter += 1;
9 }
10
11 void setup() {
12     // put your setup code here, to run once:

```

```
13 Serial.begin(9600);           // initialize serial port, set baud rate to 9600
14 static CBTimer timer;
15 timer.begin(1000, callback_func);
16 }
17
18 void loop() {
19 // put your main code here, to run repeatedly:
20 if(counter_flag == true)
21 {
22     counter_flag = false;
23     Serial.print ("counter:");
24     Serial.println(counter);
25 }
26 }
```

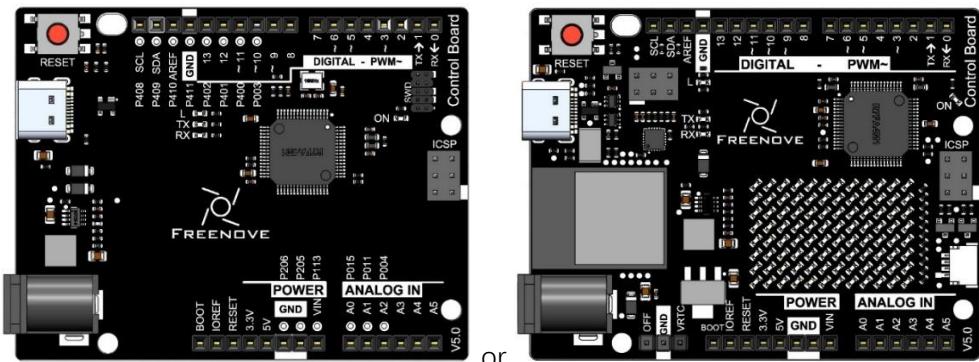
Verify and upload the code, open the Serial Monitor, and then you'll see data sent from control board. If it is not displayed correctly, check whether the configuration of the Serial Monitor in the lower right corner of the window is correct.



Project 7.2 Using timer to implement LED blinking

Component List

Control board x1

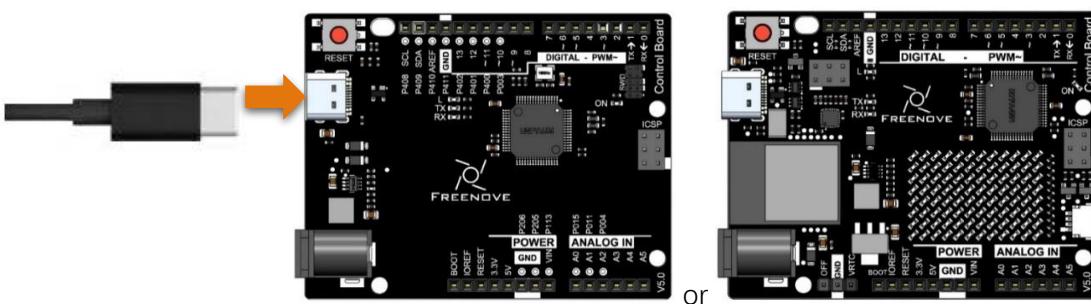


USB cable x1



Circuit

Connect control board to the computer with USB cable.



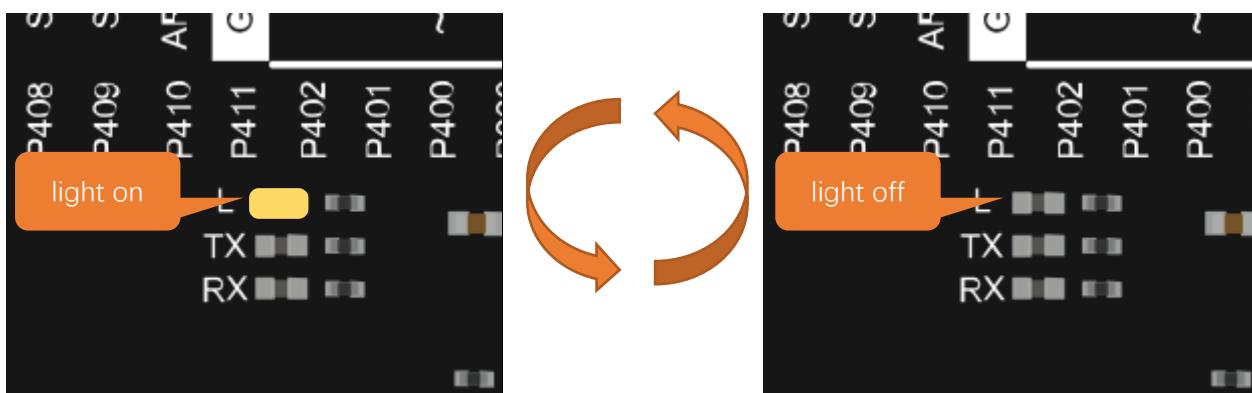
If you need any support, please feel free to contact us via: support@freenove.com

Sketch

Sketch 7.2.1

```
1 #include <CBTimer.h> // Contains CallbackTimerR4 Library
2
3 const int ledpin = 13;
4
5 void callback_func(void) {
6     int ledstate = digitalRead(ledpin);
7     digitalWrite(ledpin, !ledstate);
8 }
9
10 void setup() {
11     Serial.begin(9600); // initialize serial port, set baud rate to 9600
12     pinMode(ledpin, OUTPUT);
13     static CBTimer timer;
14     timer.begin(1000, callback_func);
15 }
16
17 void loop() {
18     // put your main code here, to run repeatedly:
19 }
```

Verify and upload the code, then you will see the LED light starts to flash with a period of 1 second



Chapter 8 ADC

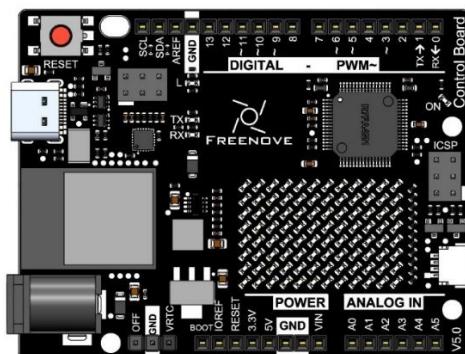
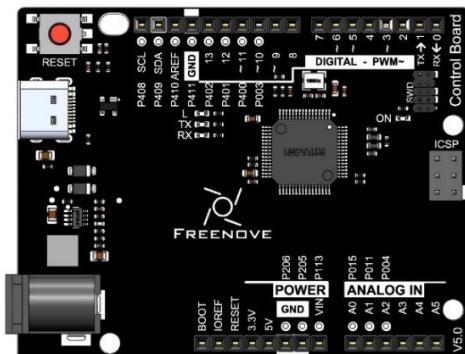
Previously we have learned about the digital ports of the control board and tried to output and input signals. Now, let's learn how to use the analog-to-digital converter (ADC) on the control board. By default, the resolution of the control board ADC is set to 10 bits, which can be updated to 12 bits (0-4096) and 14 bits (0-16383) to improve the accuracy of analog readings.

Project 8.1 ADC

ADC is used to convert analog signals into digital signals. Control chip on the control board has integrated this function. Now let us try to use this function to convert analog signals into digital signals.

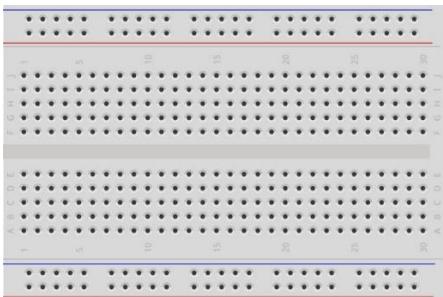
Component List

Control board x1

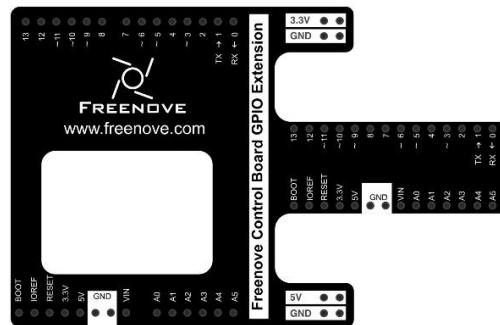


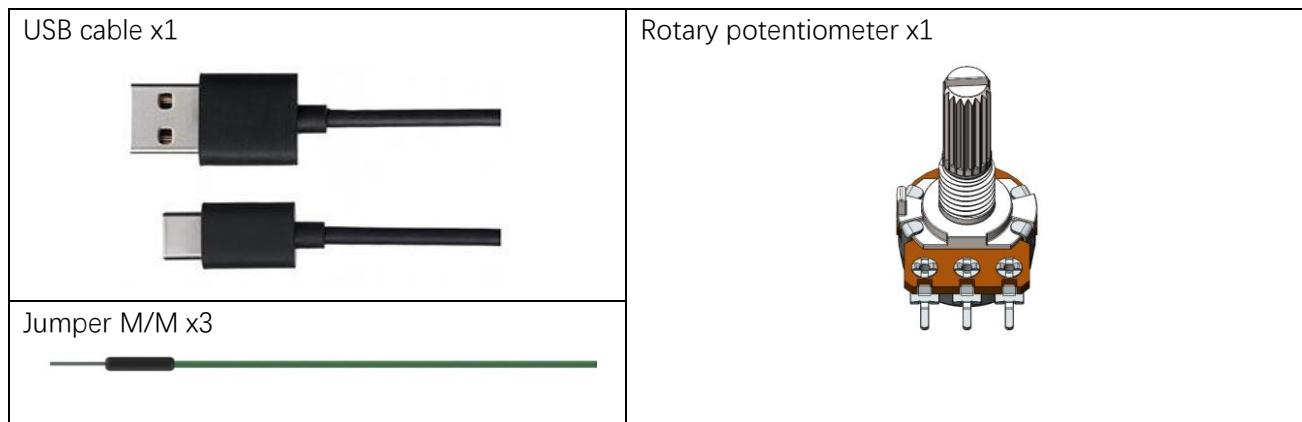
or

Breadboard x1



GPIO Extension Board x1

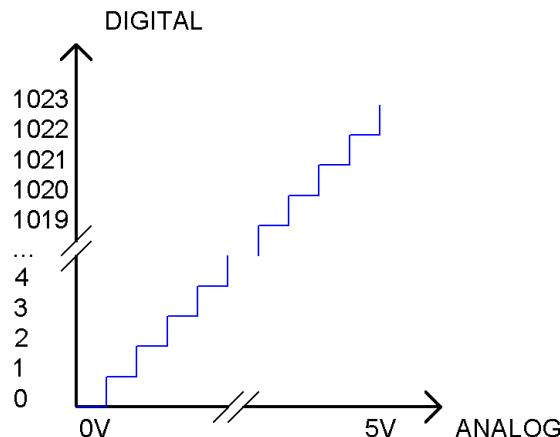




Circuit Knowledge

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 onboard ADC module is set to 10 bits by default, which means the resolution is $2^{10}=1024$, so its range (at 5V) will be evenly divided into 1024 parts. By default, the resolution of the onboard ADC is set to 10 bits, which can be updated to 12 bits (0-4096) and 14 bits (0-16383) resolution to improve the accuracy of the analog readings. Any analog value can be mapped to one digital value using the resolution of the converter. 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-5/1024V corresponds to digital 0;

Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;

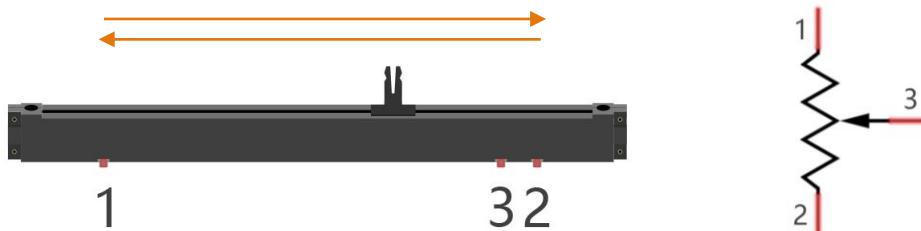
The following analog signal will be divided accordingly.

Component Knowledge

Potentiometer

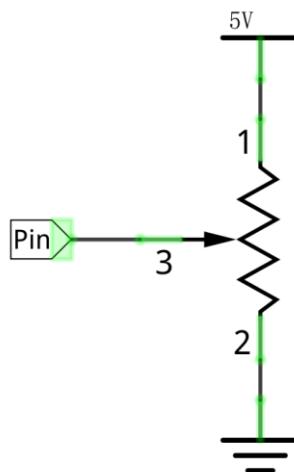
Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A

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.



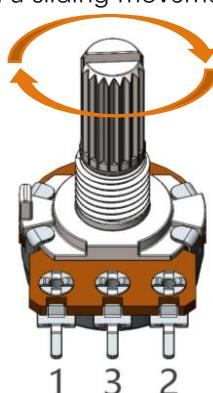
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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

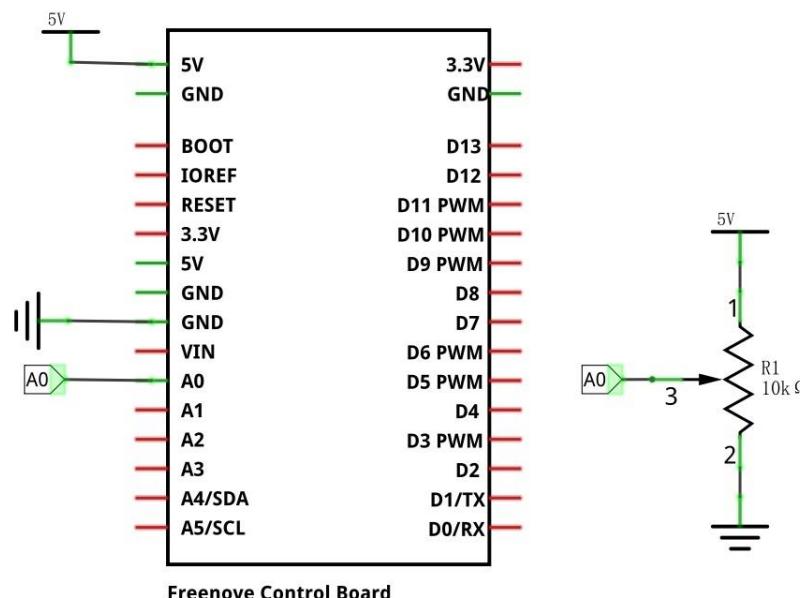
Rotary potentiometer and linear potentiometer have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



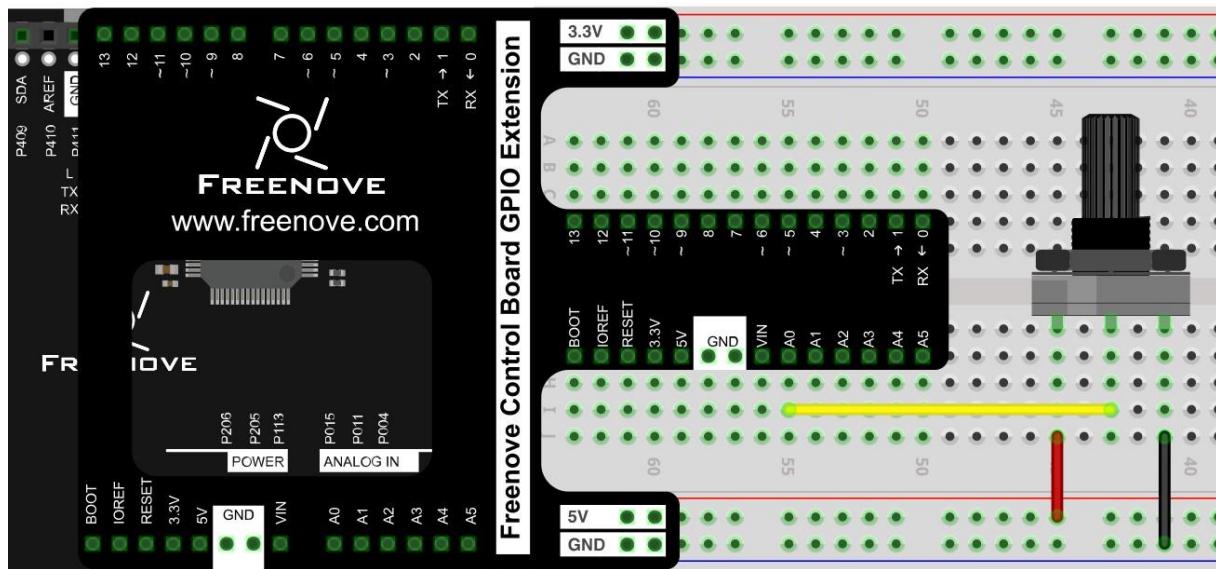
Circuit

Use pin A0 on the control board to detect the voltage of rotary potentiometer.

Schematic diagram



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



Sketch

Sketch 8.1.1

Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino IDE through serial port. The code here demonstrates how to set the ADC to 14 bits precision.

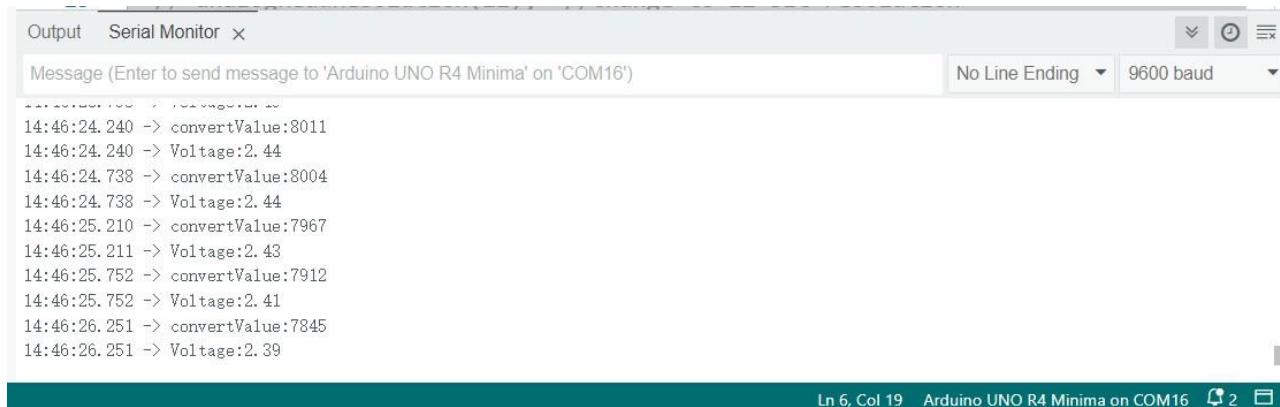
```

1 int adcValue; // Define a variable to save ADC value
2 float voltage; // Define a variable to save the calculated voltage value
3
4 void setup() {
5     Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
6     // analogReadResolution(10); //change to 10-bit resolution
7     // analogReadResolution(12); //change to 12-bit resolution
8     analogReadResolution(14); //change to 14-bit resolution
9 }
10
11 void loop() {
12     // int reading = analogRead(A0); // returns a value between 0-1023
13     // voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital
14     // int reading = analogRead(A0); // returns a value between 0-4095
15     // voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital
16     int reading = analogRead(A0); // returns a value between 0-16383
17     voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital
18     Serial.print("convertValue:");
19     Serial.println(reading);
20     Serial.print("Voltage:");
21     Serial.println(voltage);
22     delay(500);
23 }
```

From the code, we get the ADC value of pin A0, then convert it into voltage and sent to the serial port.

Verify and upload the code, open the Serial Monitor, and then you will see the original ADC value and converted voltage sent from control board.

Turn the rotary potentiometer shaft, and you can see the voltage change.



If you want to change the resolution of the ADC, you only need to modify the parameter of the function `analogReadResolution(int bits)`; among which, bits is the bit number of the ADC. If the parameter is not set, it will be 10 bits by default. As this project applies 10-bit ADC, you do not need to set the parameter for this function.

```
1 // analogReadResolution(10); //change to 10-bit resolution  
2 // analogReadResolution(12); //change to 12-bit resolution  
3 analogReadResolution(14); //change to 14-bit resolution
```

Read the ADC value and calculate the voltage based on the set ADC bit number.

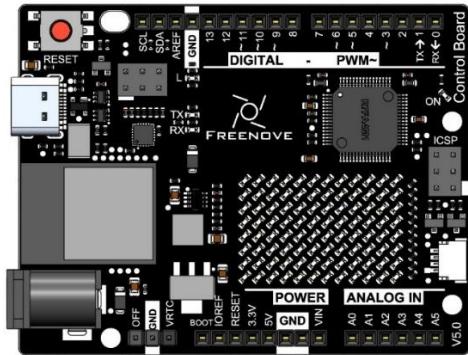
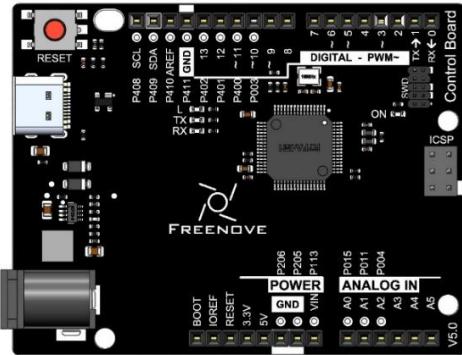
```
// int reading = analogRead(A0); // returns a value between 0-1023  
// voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital  
// int reading = analogRead(A0); // returns a value between 0-4095  
// voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital  
int reading = analogRead(A0); // returns a value between 0-16383  
voltage = reading * (5.0 / 16383.0); // Calculate voltage according to digital
```

Project 8.2 Control LED by Potentiometer

In the previous section, we have finished reading ADC value and converting it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

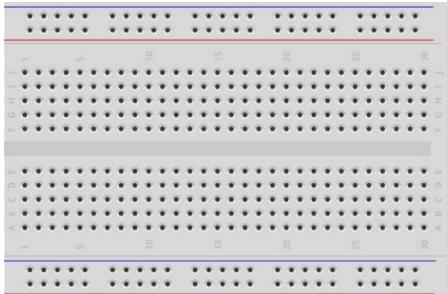
Component List

Control board x1

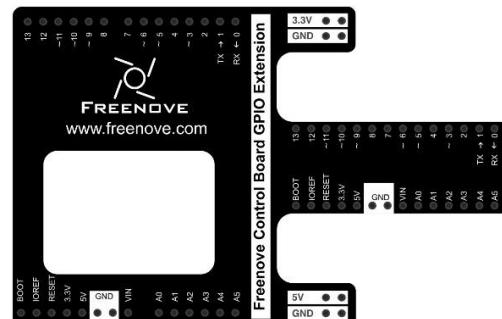


or

Breadboard x1



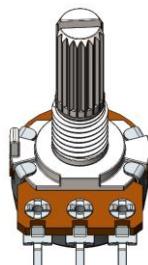
GPIO Extension Board x1



USB cable x1



Rotary potentiometer x1



LED x1



Resistor 220Ω x1



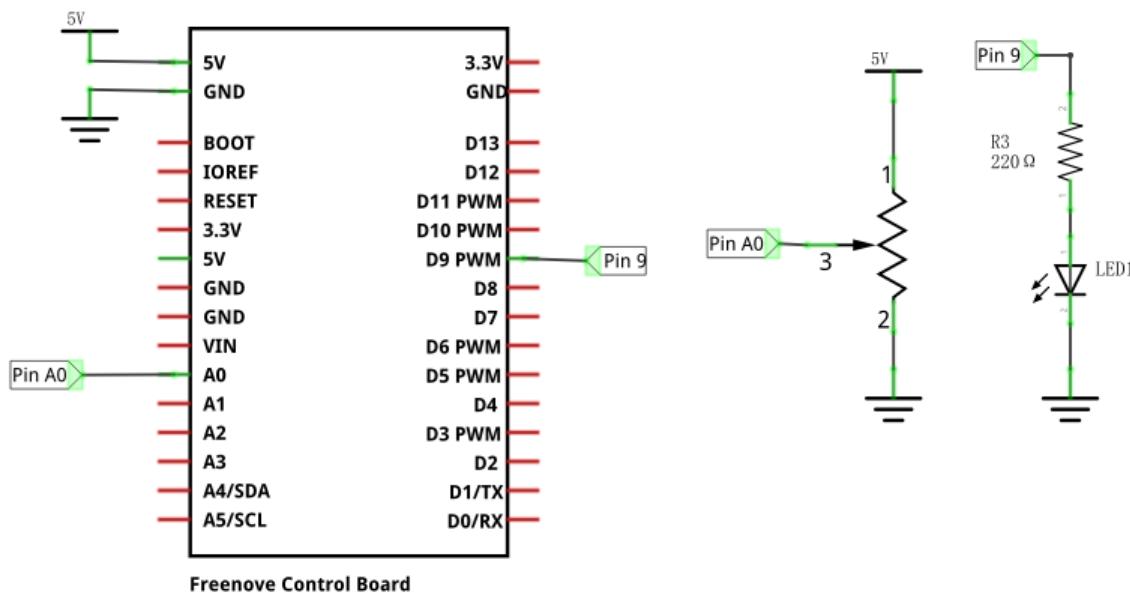
Jumper M/M x5



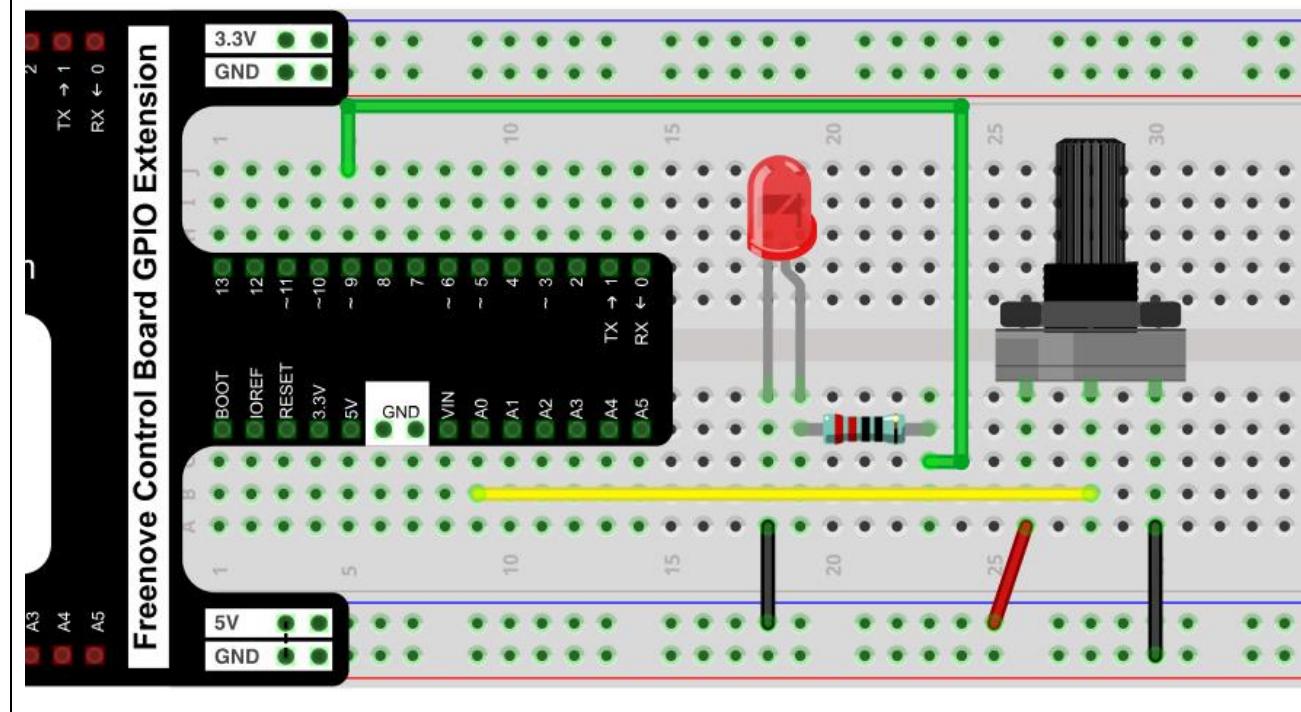
Circuit

Use pin A0 on control board to detect the voltage of rotary potentiometer, and use pin 9 to control one LED.

Schematic diagram



Hardware connection



Sketch

Sketch 8.2.1

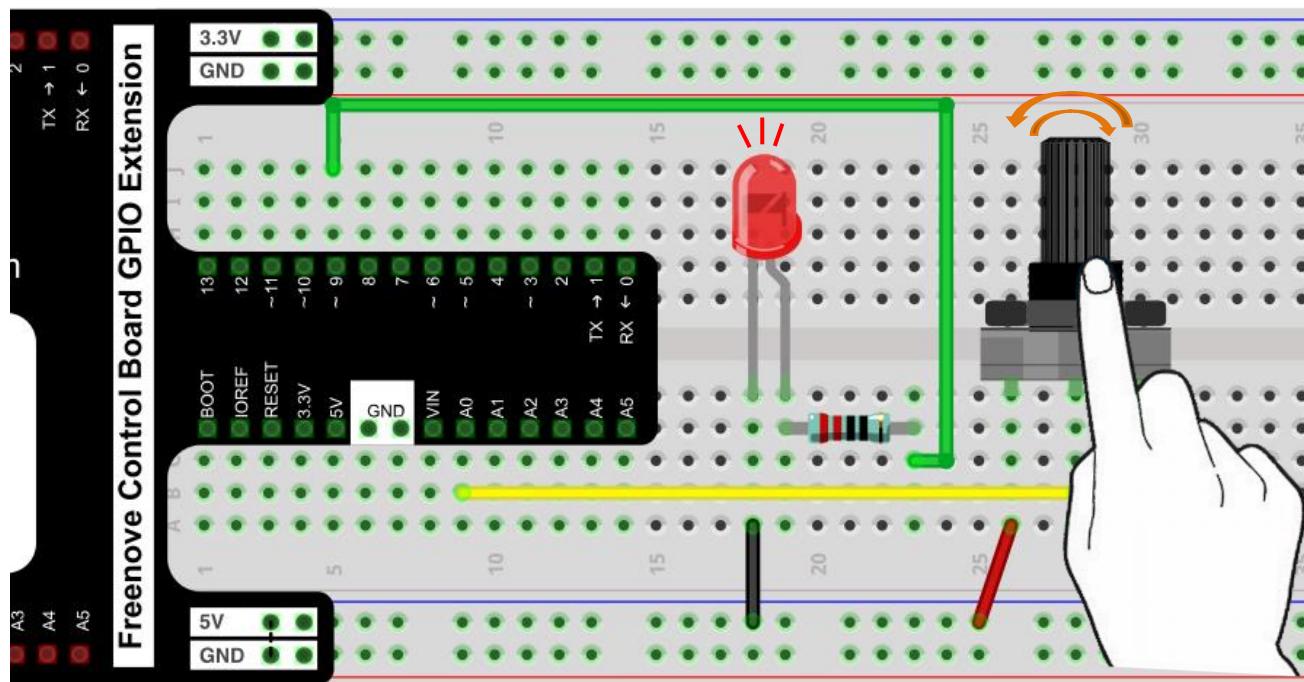
Now, write the code to detect the voltage of rotary potentiometer, and control LED to emit light with different brightness according to that.

```

1 int adcValue; // Define a variable to save the ADC value
2 int ledPin = 9; // Use D9 on Freenove UNO to control the LED
3
4 void setup() {
5     pinMode(ledPin, OUTPUT); // Initialize the LED pin as an output
6 }
7
8 void loop() {
9     adcValue = analogRead(A0); // Convert the analog of A0 port to digital
10    // Map analog to the 0-255 range, and works as PWM duty cycle of ledPin port
11    analogWrite(ledPin, map(adcValue, 0, 1023, 0, 255));
12 }
```

In the code, we get the ADC value of pin A0 and map it to PWM duty cycle of LED pin port. According to different LED brightness, we can see the changes of voltage easily.

Verify and upload the code, turn the rotary potentiometer shaft, you will see the LED brightness change.

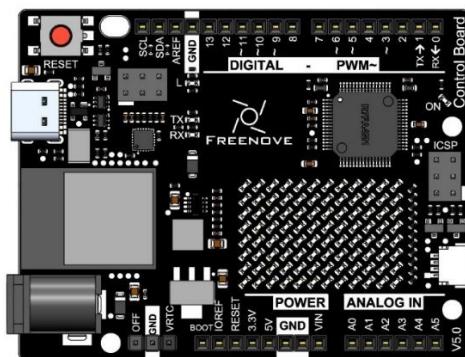
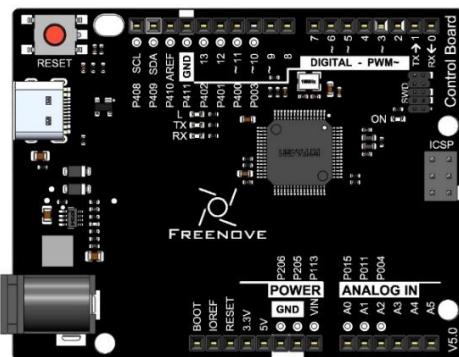


Project 8.3 Control LED by Potentiometer

In the previous section, we have finished reading ADC value and converted it into LED brightness. There are many components, especially the sensor whose output is analog. Now, we will try to use photoresistor to measure the brightness of light.

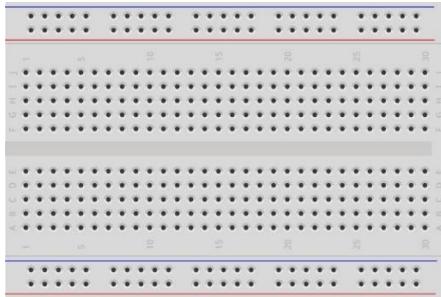
Component List

Control board x1

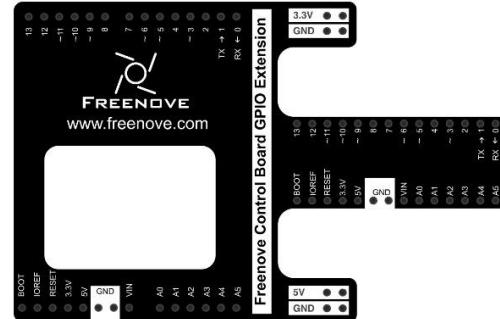


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Photoresistor x1



LED x1



Resistor 10kΩ x1



Resistor 220Ω x1



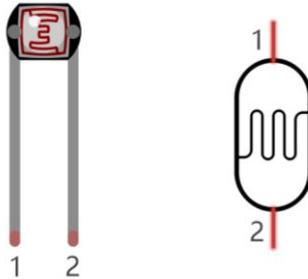
Jumper M/M x5



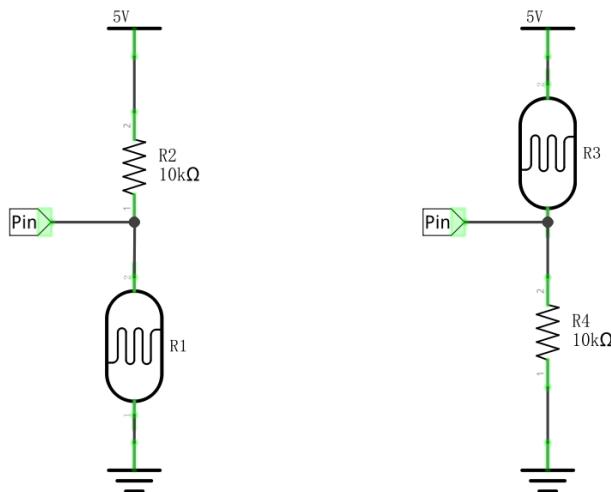
Component Knowledge

Photoresistor

A 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. A 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 often used to detect the change of photoresistor's resistance value:

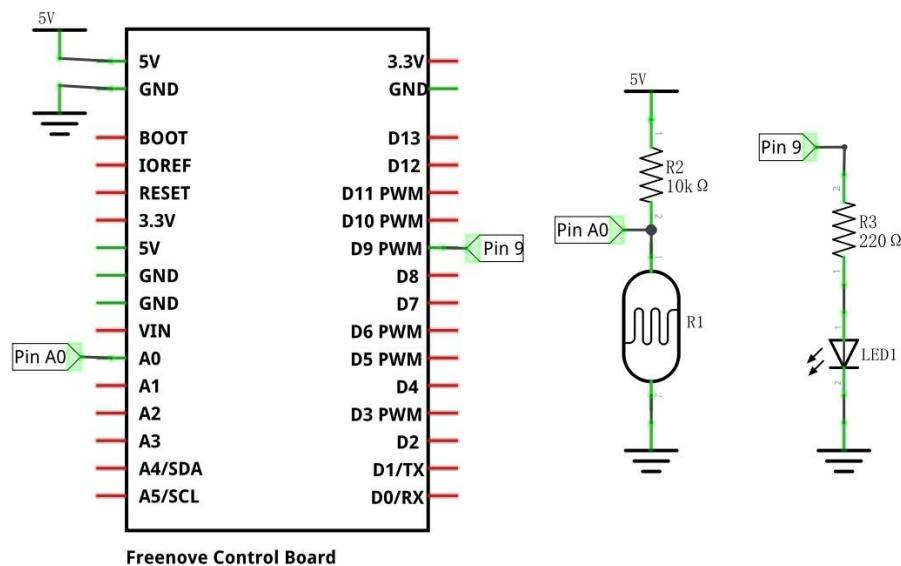


In the above circuit, when a photoresistor's resistance value changes due to a change in light intensity, the voltage between the photoresistor and resistor R1 will also change, so the intensity of the light can be obtained by measuring the voltage.

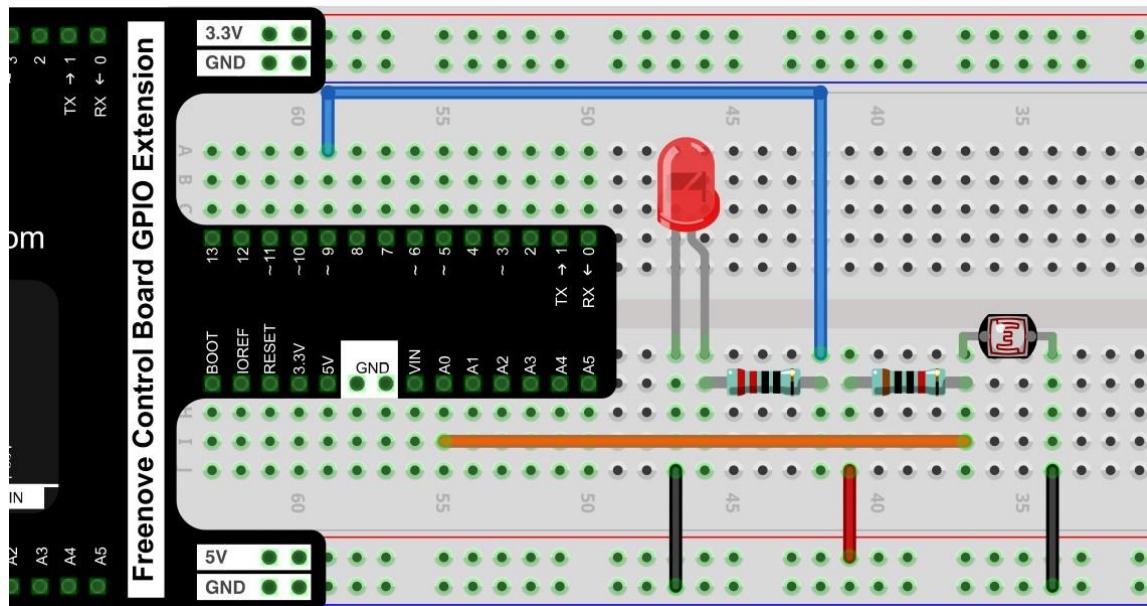
Circuit

Use pin A0 on control board to detect the voltage of photoresistor, and use pin 9 to control one LED.

Schematic diagram



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



Sketch

Sketch 7.3.1

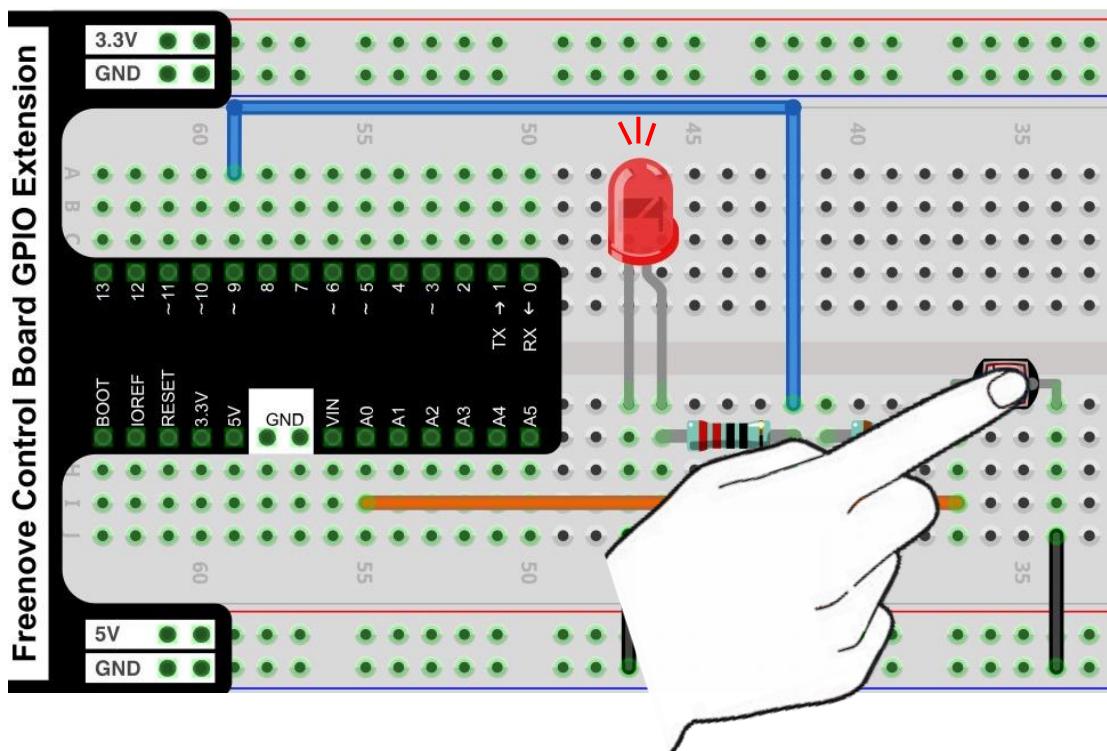
Now, write code to detect the voltage of rotary potentiometer, and control LED to emit light with different brightness according to that.

```

1 int convertValue; // Define a variable to save the ADC value
2 int ledPin = 9; // The number of the LED pin
3
4 void setup() {
5     pinMode(ledPin, OUTPUT); // Set ledPin into output mode
6 }
7
8 void loop() {
9     convertValue = analogRead(A0); // Read analog voltage value of A0 port, and save
10    // Map analog to the 0-255 range, and works as ledPin duty cycle setting
11    digitalWrite(ledPin, map(convertValue, 0, 1023, 0, 255));
12 }
```

In the code, we get the ADC value of pin A0, map it to PWM duty cycle of LED pin port. According to the brightness of LED, we can see the changes of voltage easily.

Verify and upload the code, cover photoresistor with your hand, then you can see the LED brightness change.



Chapter 9 RGB LED

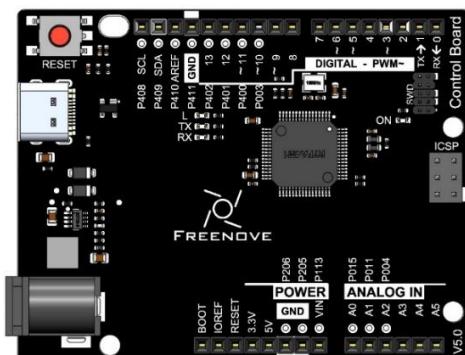
Earlier, we have learned to apply the analog port and ADC of the control board. Now, we'll use ADC to control RGB LED.

Project 9.1 Control RGB LED through Potentiometer

RGB LED has three different-color LEDs inside, and we will use 3 potentiometers to control these 3 LEDs to emit light with different brightness, and observe what will happen.

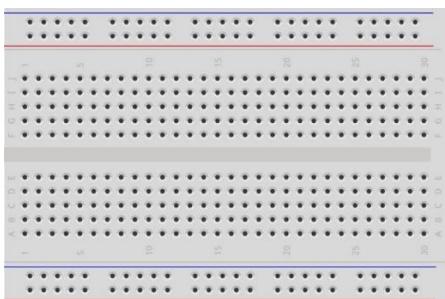
Component List

Control board x1

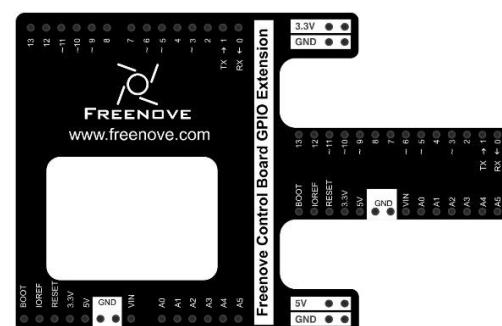


The image shows a Freenove Uno R3 microcontroller board. It features a central ATmega328P microchip. Various pins are labeled: digital pins 0-13, PWM pins 3-9, analog input pins A0-A5, power pins 5V, GND, and 3.3V, and ground pins GND and AGND. On the left, there's a USB port, a red pushbutton labeled 'RESET', and a 16x2 character LCD module. On the right, there's a 16x2 character LCD module, a 74HC595 shift register, and several other components like resistors and capacitors. The board is densely populated with surface-mount technology.

Breadboard x1



GPIO Extension Board x1



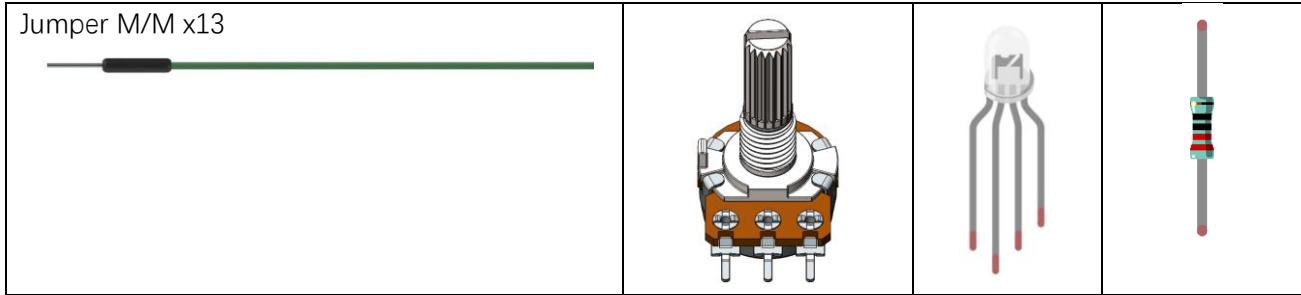
USB cable x1



Rotary potentiometer
x3

RGB LED x1

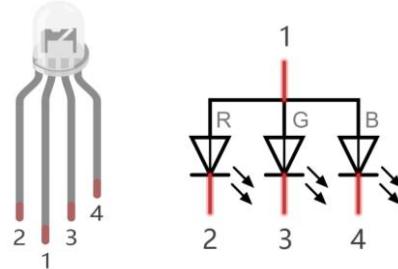
Resistor 220Ω



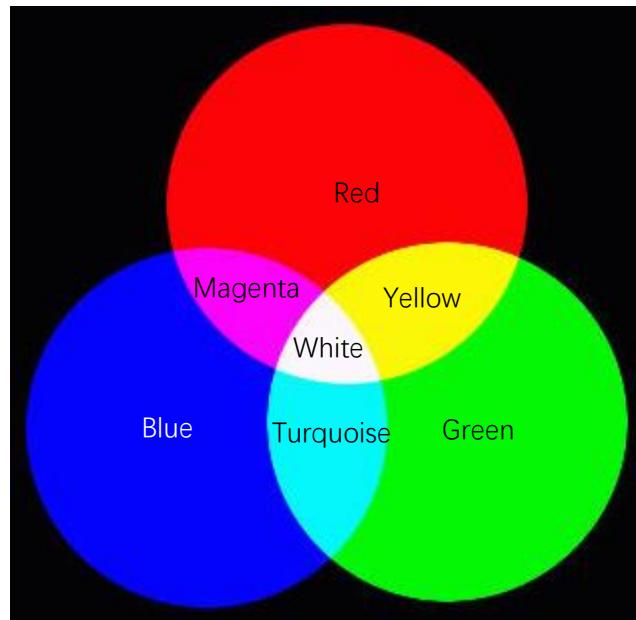
Component Knowledge

RGB LED

A RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Cathode (+) or positive lead, the other 3 are the Anodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Anodes (2, 3 & 4) of the RGB LED



Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon.

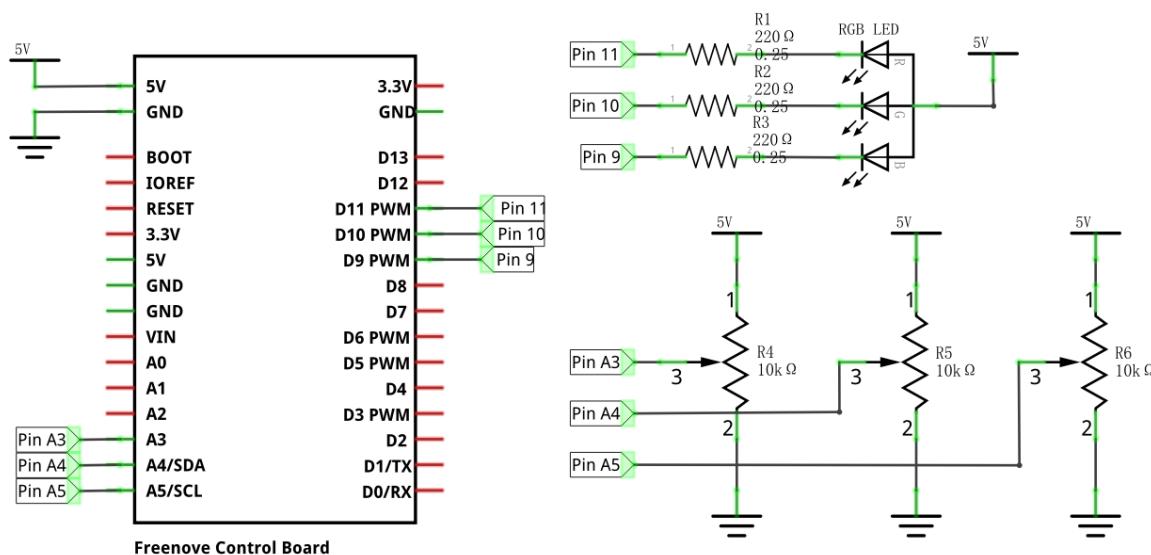


We know from the previous section that, control board controls LED to emit a total of 256(0-255) different brightness via PWM. So, through different combinations of RGB light brightness, we can create $256^3=16777216$ (16Million) colors.

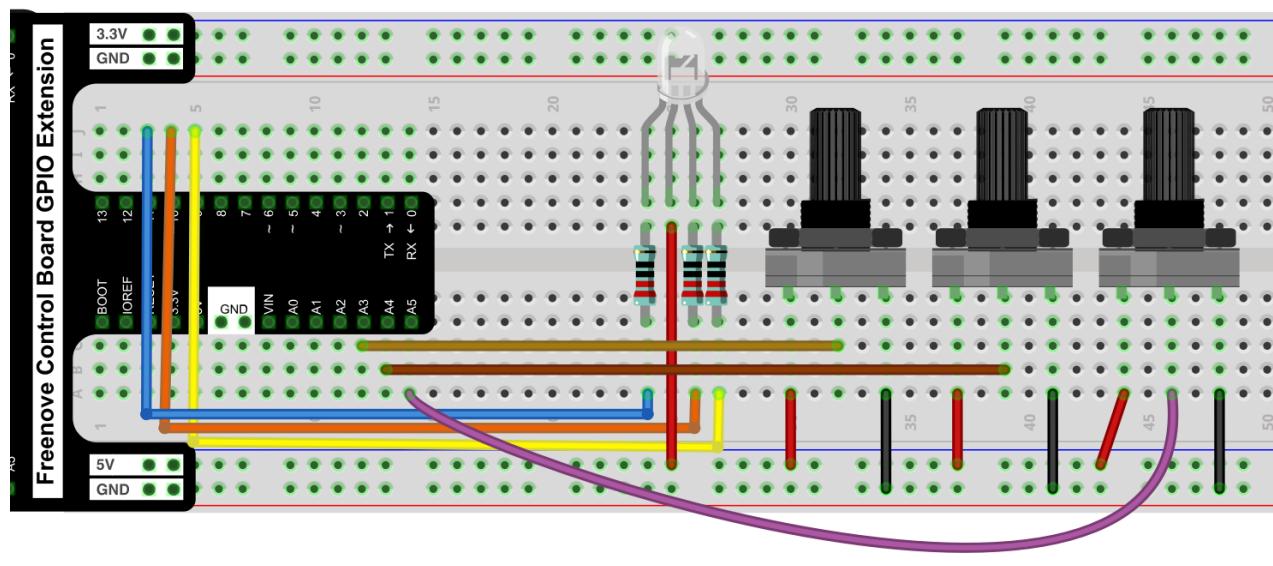
Circuit

Use pin A3, A4, A5 ports of the control board to detect the voltage of rotary potentiometer, and control RGB LED by pin 9, 10, 11.

Schematic diagram



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



Sketch

Sketch 9.1.1

Now, write code to detect the voltages of these three rotary potentiometers, and convert them into PWM duty cycle to control 3 LEDs inside the RGB LED.

```
1 // set pin numbers:  
2 int ledPinR = 11; // the number of the LED R pin  
3 int ledPinG = 10; // the number of the LED G pin  
4 int ledPinB = 9; // the number of the LED B pin  
5  
6 void setup() {  
7     // initialize the LED pin as an output:  
8     pinMode(ledPinR, OUTPUT);  
9     pinMode(ledPinG, OUTPUT);  
10    pinMode(ledPinB, OUTPUT);  
11}  
12  
13 void loop() {  
14     int adcValue; // Define a variable to save the ADC value  
15     // Convert analog of A3 port into digital, and work as PWM duty cycle of ledPinR port  
16     adcValue = analogRead(A3);  
17     analogWrite(ledPinR, map(adcValue, 0, 1023, 0, 255));  
18     // Convert analog of A4 port into digital, and work as PWM duty cycle of ledPinG port  
19     adcValue = analogRead(A4);  
20     analogWrite(ledPinG, map(adcValue, 0, 1023, 0, 255));  
21     // Convert analog of A5 port into digital, and work as PWM duty cycle of ledPinB port  
22     adcValue = analogRead(A5);  
23     analogWrite(ledPinB, map(adcValue, 0, 1023, 0, 255));  
24 }
```

In the code, we get the voltages of three rotary potentiometers, and convert them into PWM duty cycle to control the three LEDs of the RGB LED to emit light with different brightness.

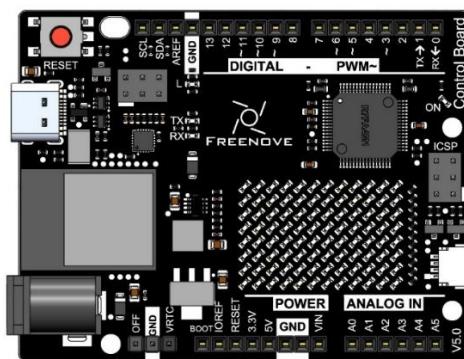
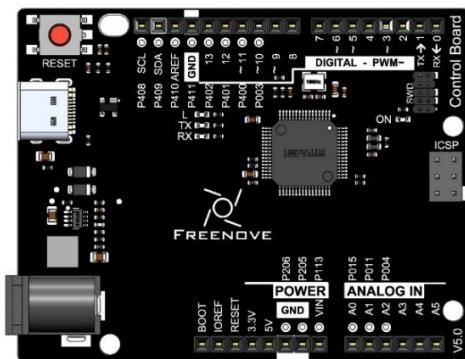
Verify and upload the code, rotate the three rotary potentiometer shaft, and you can see the LED light change in its color and brightness.

Project 9.2 Multicolored LED

In the previous section, we have finished controlling the RGB LED to emit light with different color and brightness through three potentiometers. Now, we will try to make RGB LED emit multicolored lights automatically.

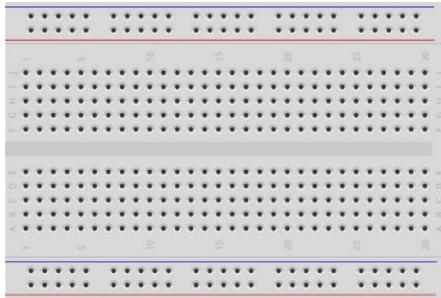
Component List

Control board x1

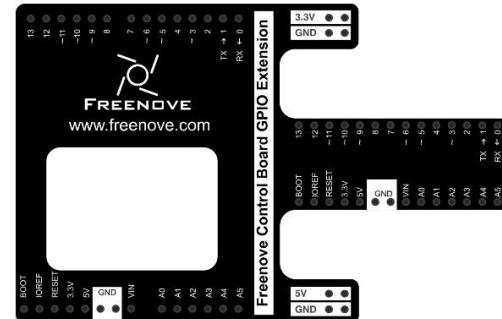


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



RGB LED x1



Resistor 220Ω x3



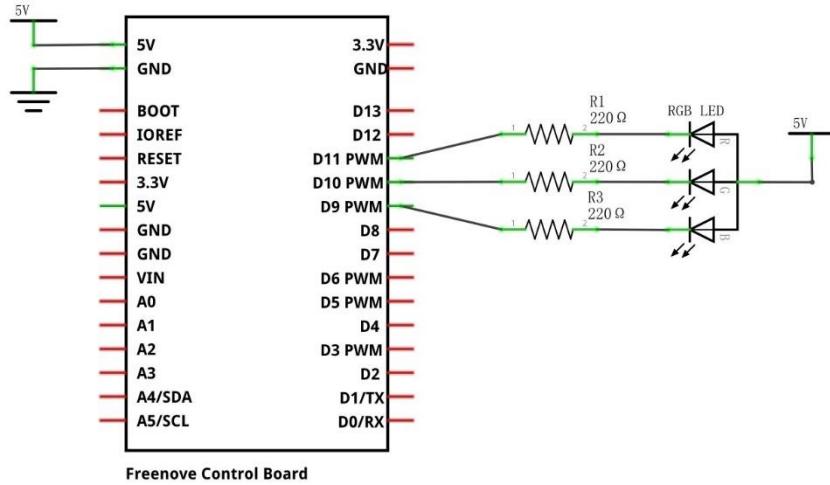
Jumper M/M x4



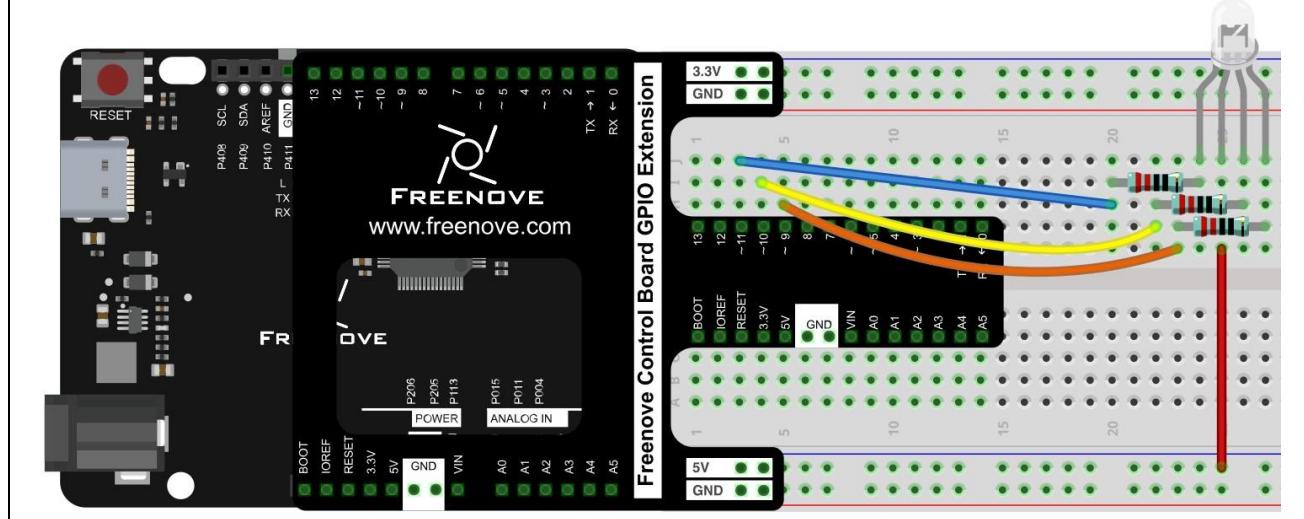
Circuit

Use pin 9, 10, 11 of the control board to control RGB LED.

Schematic diagram



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



Sketch

Sketch 9.2.1

Now, write code to generate three random numbers, and convert them into PWM duty cycle to control the three LEDs of RGB LED.

```
1 int ledPinR = 11; // the number of the LED R pin
2 int ledPinG = 10; // the number of the LED G pin
3 int ledPinB = 9; // the number of the LED B pin
4
5 void setup() {
6     // initialize the LED pin as an output:
7     pinMode(ledPinR, OUTPUT);
8     pinMode(ledPing, OUTPUT);
9     pinMode(ledPinB, OUTPUT);
10 }
11
12 void loop() {
13     // Generate three random numbers between 0-255 as the output PWM duty cycle of ledPin
14     rgbLedDisplay(random(256), random(256), random(256));
15     delay(500);
16 }
17
18 void rgbLedDisplay(int red, int green, int blue) {
19     // Set three ledPin to output the PWM duty cycle
20     analogWrite(ledPinR, constrain(red, 0, 255));
21     analogWrite(ledPinG, constrain(green, 0, 255));
22     analogWrite(ledPinB, constrain(blue, 0, 255));
23 }
```

In the code, we create three random numbers, and convert them into PWM duty cycle to control the three LEDs of RGB LED to emit light with different brightness. At regular intervals, a new random number will be created, so RGB LED will start flashing light with different colors and brightness.

random(min, max)

random (min, max) function is used to generate random number, and it will return a random value in the range (min, Max-1).

You can also use random (max) function, the function set the minimum value into 0 by default, and returns a random value in the range (0, Max-1).

Verify and upload the code, and RGB LED starts flashing with different colors and brightness.

Chapter 10 Buzzer

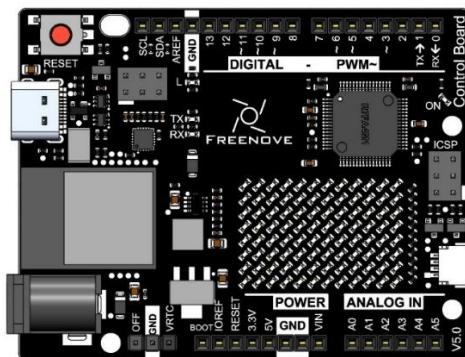
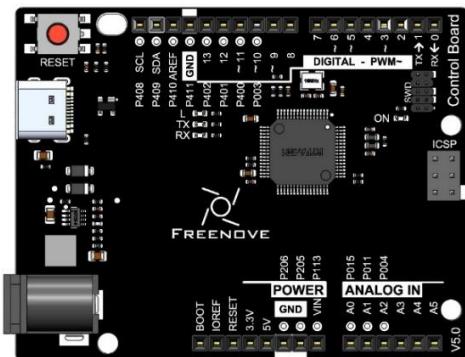
Earlier, we have used control board and basic electronic components to carry out a series of interesting projects. Now, let's learn how to use some integrated electronic components and modules. These modules are usually integrated with a number of electronic components, so they have special features and usages. In this chapter, we'll use a sounding module, buzzer. It has two types: active and passive buzzer.

Project 10.1 Active Buzzer

First, let's study some knowledge about active buzzer.

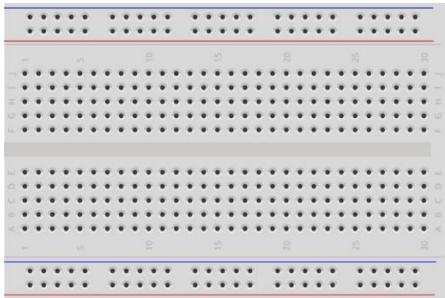
Component List

Control board x1

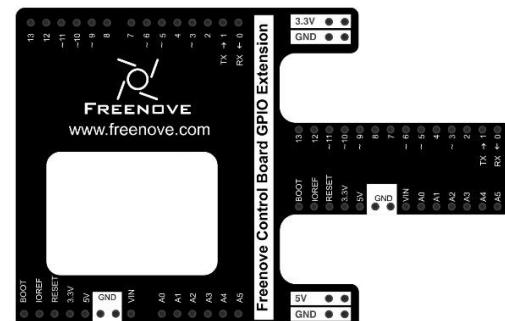


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Jumper M/M x6



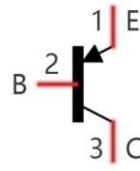
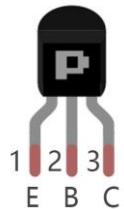
NPN transistor x1	Active buzzer x1	Push button x1	Resistor 1kΩ x1	Resistor 10kΩ x2
				

Component knowledge

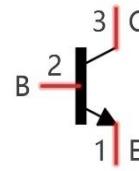
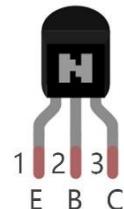
Transistor

Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current (think of a transistor as an electronic "amplifying or switching device"). Transistors can be used to amplify weak signals, or to work as a switch. Transistors have three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", then "ce" will have a several-fold current increase(transistor magnification), in this configuration the transistor acts as an amplifier. When current produced by "be" exceeds a certain value, "ce" will limit the current output. at this point the transistor is working in its saturation region and acts like a switch. Transistors are available as two types as shown below: PNP and NPN,

PNP transistor



NPN transistor

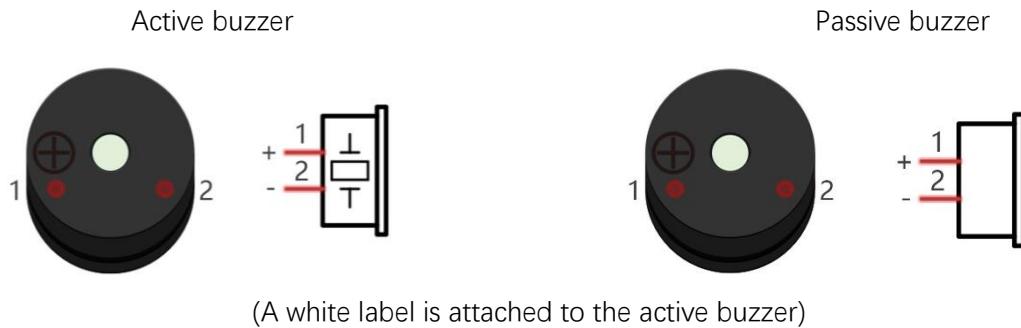


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Thanks to the transistors' characteristics, they are often used as switches in digital circuits. As micro-controllers output current capacity is very weak, we will use a transistor to amplify its current in order to drive components requiring higher current.

Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.



Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

How to identify active and passive buzzer?

1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
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, view 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).

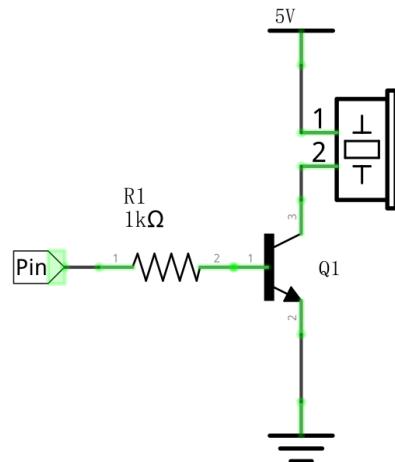


Active buzzer bottom

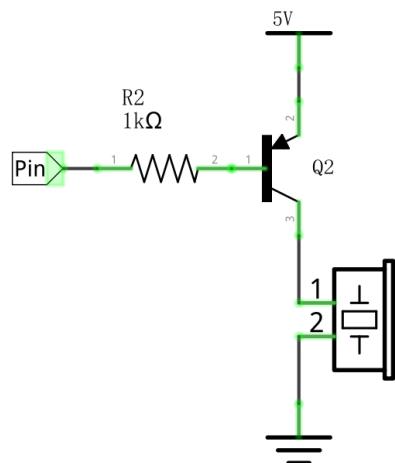
Passive buzzer bottom

Buzzers need relatively larger current when they work. But generally, microcontroller port can't provide enough current for them. In order to control buzzer by control board, transistor can be used to drive a buzzer indirectly.

When we use a NPN transistor to drive a buzzer, we often use the following method. If control board pin outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If control board pin outputs low level, no current will flow through R1, the transistor will not conduct current and buzzer will remain silent (no sounds).



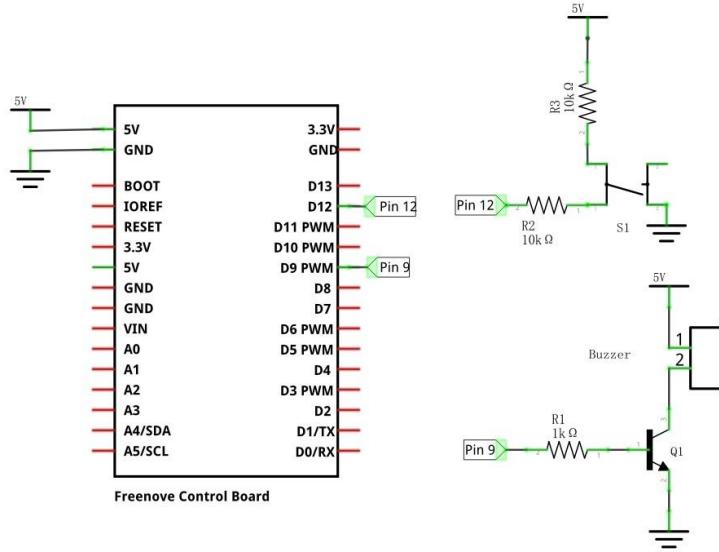
When we use a PNP transistor to drive a buzzer, we often use the following method. If control board pin outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If control board pin outputs high level, no current flows through R1, the transistor will not conduct current and buzzer will remain silent (no sounds). Below are the circuit schematics for both a NPN and PNP transistor to power a buzzer.



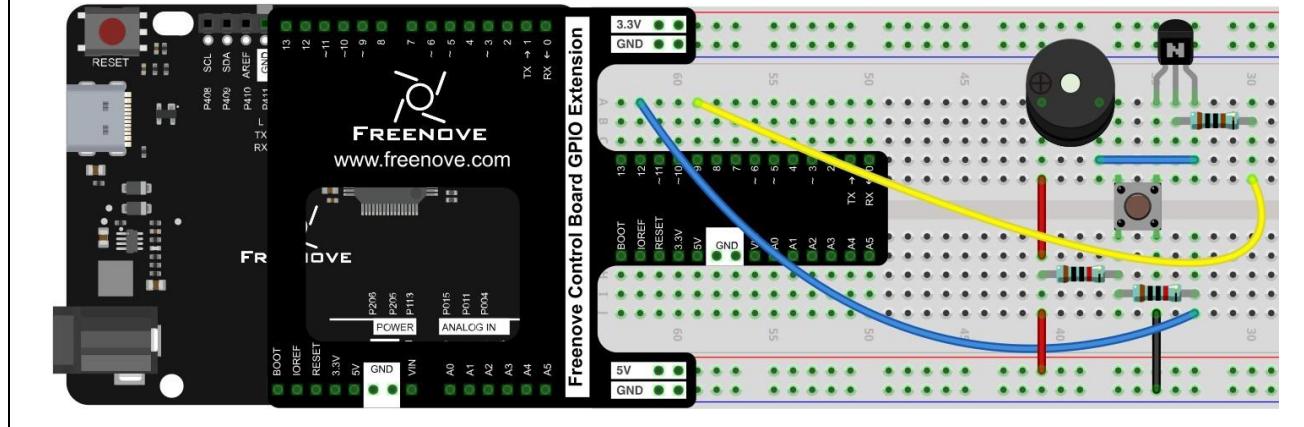
Circuit

Use pin 12 of control board to detect the state of push button switch, and pin 9 to drive active buzzer.

Schematic diagram



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



Sketch

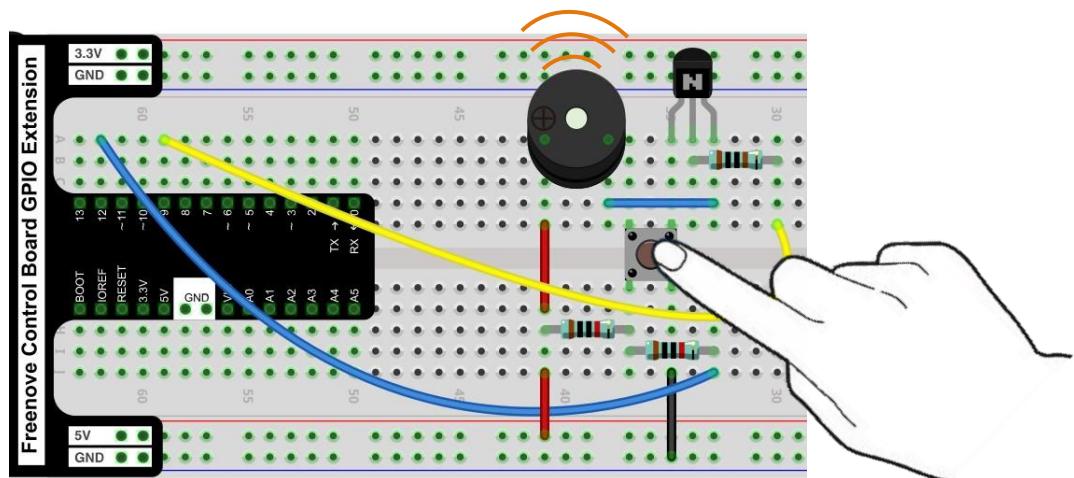
Sketch 10.1.1

Now, write code to detect the state of push button, and drive active buzzer to make a sound when it is pressed.

```
1 int buttonPin = 12; // the number of the push button pin
2 int buzzerPin = 9; // the number of the buzzer pin
3
4 void setup() {
5     pinMode(buttonPin, INPUT); // Set push button pin into input mode
6     pinMode(buzzerPin, OUTPUT); // Set Buzzer pin into output mode
7 }
8
9 void loop() {
10    if (digitalRead(buttonPin) == HIGH)// If the pin is high level, the button is not pressed.
11        digitalWrite(buzzerPin, LOW); // Turn off Buzzer
12    else // The button is pressed, if the pin is low level
13        digitalWrite(buzzerPin, HIGH); // Turn on Buzzer
14 }
```

In the code, we check the state of push button switch. When it is pressed, the output high level controls transistor to get conducted, and drives active buzzer to make a sound.

Verify and upload the code, press the push button, and the active buzzer will make a sound.

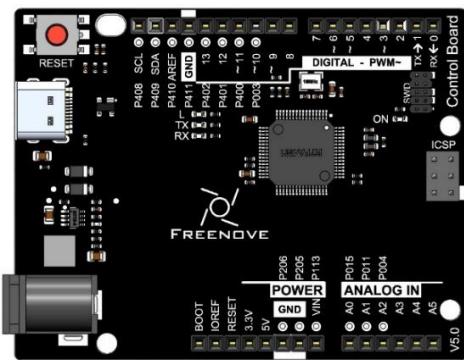


Project 10.2 Passive Buzzer

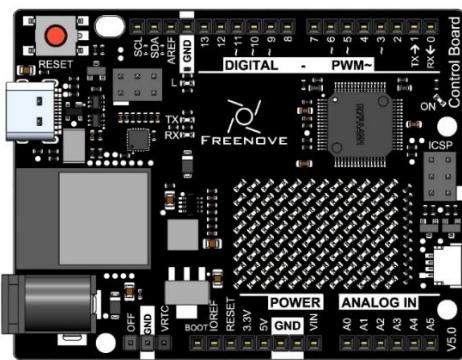
In the previous section, we have finished using transistor to drive an active buzzer to beep. Now, we will try to use a passive buzzer to make a sound with different frequency.

Component List

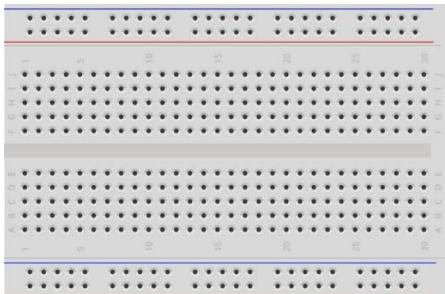
Control board x1



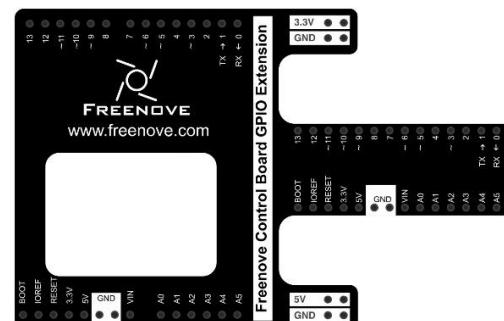
or



Breadboard x1



GPIO Extension Board x1



USB cable x1



NPN transistor
x1



Passive buzzer
x1



Resistor 1k Ω x1



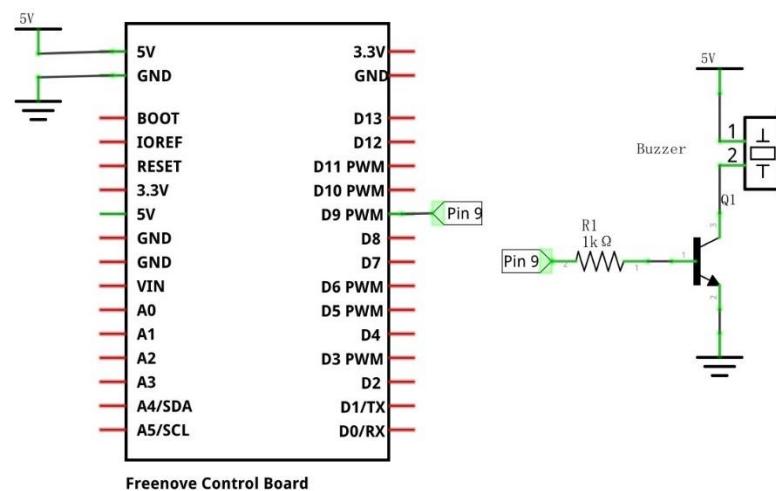
Jumper M/M x4



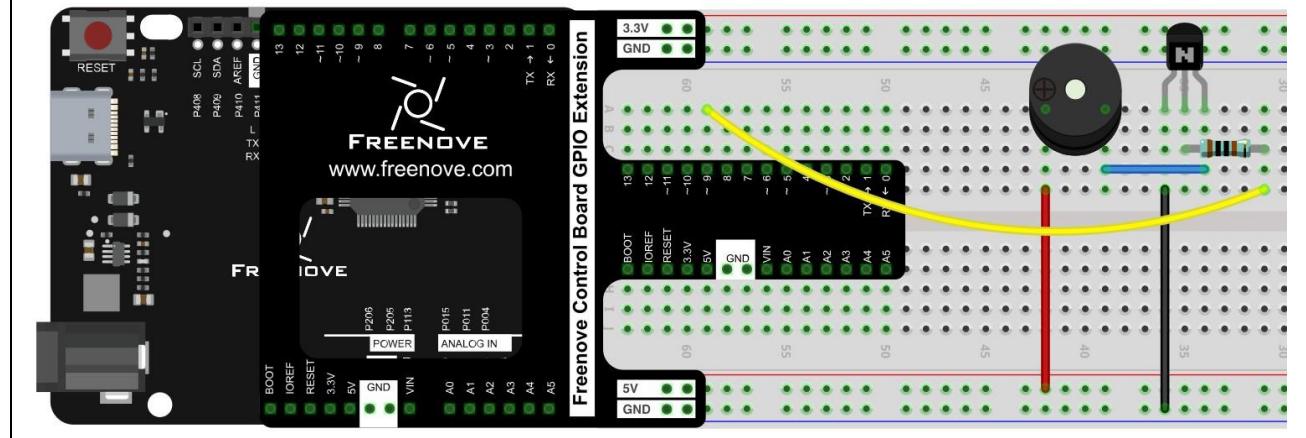
Circuit

Use pin 9 port of control board to drive a passive buzzer.

Schematic diagram



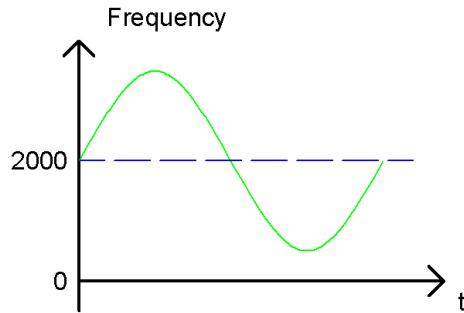
Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Sketch

Sketch 10.2.1

Now, write code to drive a passive buzzer to make a warning sound. The frequency of the passive buzzer conforms roughly to the following sine curve over time:



Output PWM waves with different frequency to the port, which is connected to the transistor, to drive buzzer to make a sound with different frequency.

```

1 int buzzerPin = 9;      // the number of the buzzer pin
2 float sinVal;          // Define a variable to save sine value
3 int toneVal;           // Define a variable to save sound frequency
4
5 void setup() {
6     pinMode(buzzerPin, OUTPUT); // Set Buzzer pin to output mode
7 }
8
9 void loop() {
10    for (int x = 0; x < 360; x++) {        // X from 0 degree->360 degree
11        sinVal = sin(x * (PI / 180));       // Calculate the sine of x
12        toneVal = 2000 + sinVal * 500;        // Calculate sound frequency according to the sine of x
13        tone(buzzerPin, toneVal);           // Output sound frequency to buzzerPin
14        delay(1);
15    }
16 }
```

In the code, use one loop to control the sound frequency, varying according to sine curve in the range of 2000 ± 500 .

```

10   for (int x = 0; x < 360; x++) {        // X from 0 degree->360 degree
11       sinVal = sin(x * (PI / 180));       // Calculate the sine of x
12       toneVal = 2000 + sinVal * 500;        // Calculate sound frequency according to the sine of x
13       tone(buzzerPin, toneVal);           // Output sound frequency to buzzerPin
14       delay(1);
15 }
```

The parameter of `sin()` function is radian, so we need convert unit of π from angle to radian first .

tone(pin, frequency)

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin.

Verify and upload the code, passive buzzer starts making a warning sound.

You can try using PNP transistor to complete the project of this chapter again.

Chapter 11 DAC

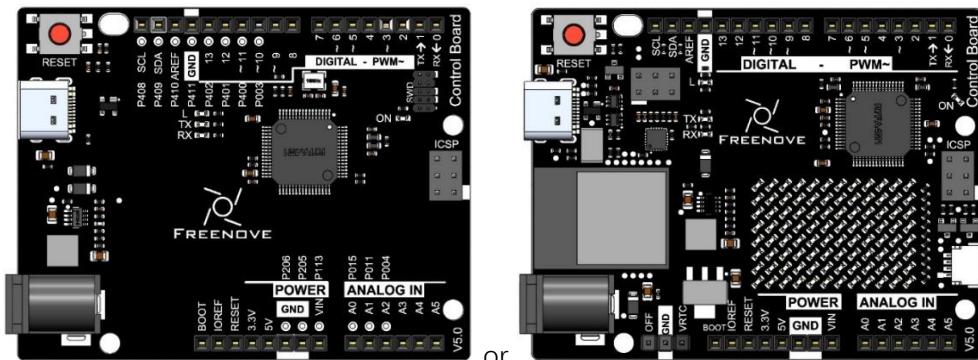
In this chapter, we learn about DAC (Digital to Analog Converter). The Control board (V5) has a built-in DAC (Digital to Analog Converter) which is used to convert digital signals into analog signals.

Project 11.1 DAC

In this project, let's learn the DAC function of the control board and use it to control a passive buzzer to play music.

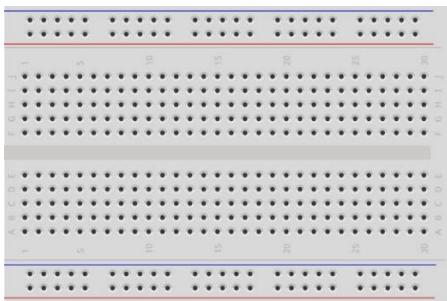
Component List

Control board x1

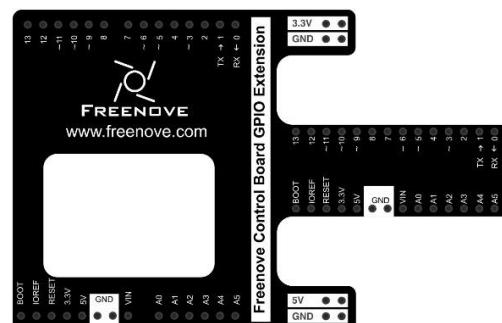


or

Breadboard x1



GPIO Extension Board x1



USB cable x1	NPN transistor x1	Speaker or Passive buzzer x1	Resistor 1kΩ x1
Jumper M/M x6			

If you don't have a speaker in your kit, use a passive buzzer instead.

Component Knowledge

DAC

Digital-to-Analog Converters (DACs) are capable of converting digital data into analog signals, making them suitable for a variety of applications. DACs, as the name suggests, are modules that convert digital codes into corresponding analog voltage outputs, functioning in a manner that is opposite to that of ADCs (Analog-to-Digital Converters).

Analog Output vs PWM

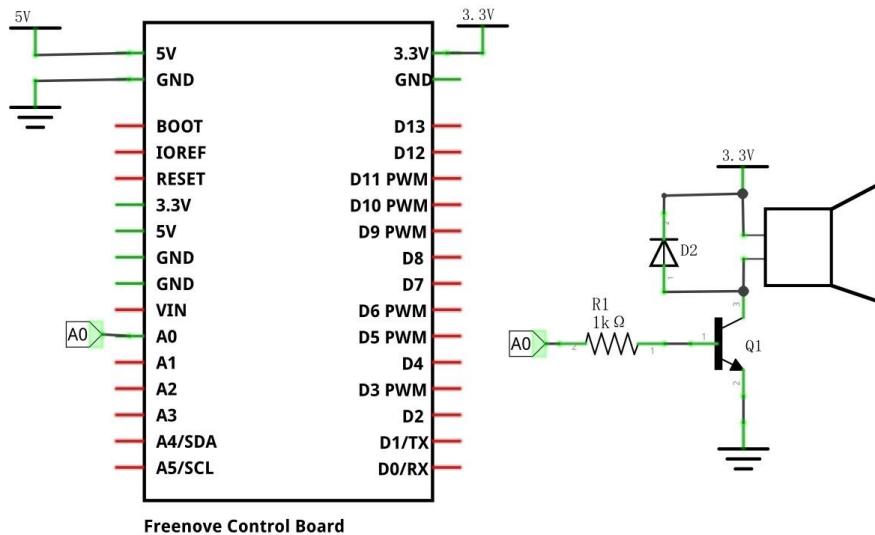
For many use cases when analog output is required, using PWM (Pulse Width Modulation) instead of genuine analog output will yield essentially the same results. A digital output pin can only either be fully on (HIGH) or fully off (LOW), but by turning it on and off very quickly with precise timings, the average voltage can be controlled and emulate an analog output. This method is called PWM.

For example when dimming an LED, you can freely use a PWM enabled digital pin as an analog output pin and the LED would dim just the same as if you'd be using a DAC output.

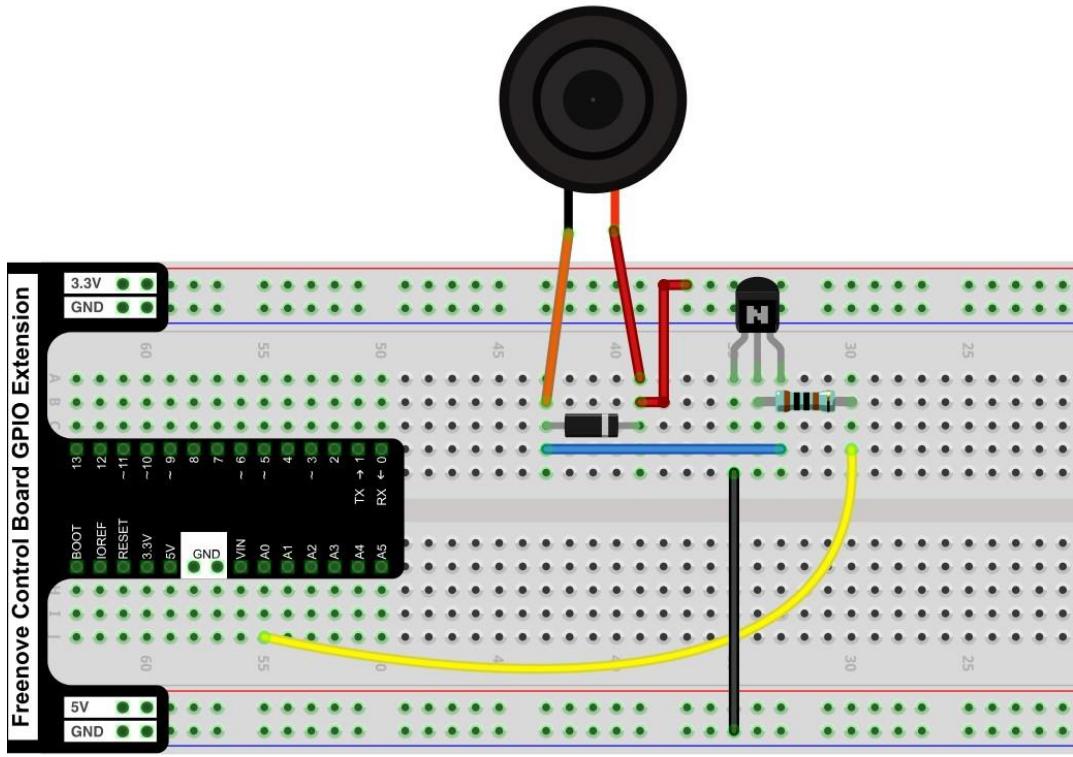
However this will not always be the case, and for many uses you will need to use a genuine analog output to get your desired results. One such case is in audio purposes, where a PWM output simply will not give the same quality of sound as a genuine analog output, and requires some fiddling to work in the first place.

Circuit

Schematic diagram



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



In addition, you can also refer to the above circuit to replace the speaker with a passive buzzer, the experimental effect is basically the same.

Sketch

Sketch 11.1.1

Upload the sketch to the board and you will hear the loudspeaker make sounds.

```
1 #include "analogWave.h"
2 #include "pitches.h"
3
4 // Melody and rhythm defined as an array
5 // Each note is followed by its duration (4 = quarter note, 8 = eighth note, etc.)
6 int melody[] = {
7     // Super Mario Bros theme
8     // Score available at https://musescore.com/user/2123/scores/2145
9     // Theme by Koji Kondo
10    NOTE_E5, 8, NOTE_E5, 8, REST, 8, NOTE_E5, 8, REST, 8, NOTE_C5, 8, NOTE_E5, 8, //1
11    NOTE_G5, 4, REST, 4, NOTE_G4, 8, REST, 4,
12    NOTE_C5, -4, NOTE_G4, 8, REST, 4, NOTE_E4, -4, // 3
13    NOTE_A4, 4, NOTE_B4, 4, NOTE_AS4, 8, NOTE_A4, 4,
14    NOTE_G4, -8, NOTE_E5, -8, NOTE_G5, -8, NOTE_A5, 4, NOTE_F5, 8, NOTE_G5, 8,
15    REST, 8, NOTE_E5, 4, NOTE_C5, 8, NOTE_D5, 8, NOTE_B4, -4,
16 };
17
18 analogWave wave(DAC); // Initialize the analogWave object for DAC(A0)
19
20 int noteCounter = 0; // Index to keep track of which note is being played
21 int tempo = 200; // Set the tempo in BPM (Beats Per Minute)
22 int wholenote = (60000 * 4) / tempo; // Calculate the duration of a whole note in ms
23 int divider = 0, noteDuration = 0; // Variables to hold note duration
24
25 void setup() {
26     wave.sine(10); // Initialize the sine wave generator with a frequency of 10 Hz
27 }
28
29 void loop() {
30     // Calculate the duration of the current note
31     divider = melody[noteCounter + 1];
32     if (divider > 0) {
33         // For regular notes
34         noteDuration = wholenote / divider;
35     } else if (divider < 0) {
36         // For dotted notes (duration increased by 50%)
37         noteDuration = wholenote / abs(divider);
38         noteDuration *= 1.5; // Increase the duration by 50% for dotted notes
39     }
```

```
40
41 // Play the note
42 wave.freq(melody[noteCounter]);
43 delay(noteDuration * 0.85); // Play the note for 85% of its duration
44 wave.stop();
45
46 // Pause between notes
47 delay(noteDuration * 0.15); // Pause for 15% of the note duration
48
49 // Increment the note counter by 2 (because each note is followed by its duration)
50 noteCounter += 2;
51
52 // Reset the counter when reaching the end of the melody
53 int totalNotes = sizeof(melody) / sizeof(melody[0]);
54 noteCounter = noteCounter % totalNotes;
55
56 // Add a delay between repetitions of the melody
57 if (noteCounter == 0) {
58     delay(1000);
59 }
60 }
```

If you want to know more about the DAC Settings of the control panel, please refer to the connection:
<https://docs.arduino.cc/tutorials/uno-r4-minima/dac/>

You can also use the above circuit to play an interesting audio clip by uploading Sketch_10.1.2_DAC_Jacques.ino to your development board. If you don't have a speaker, you can replace it with a passive buzzer.

Chapter 12 RTC

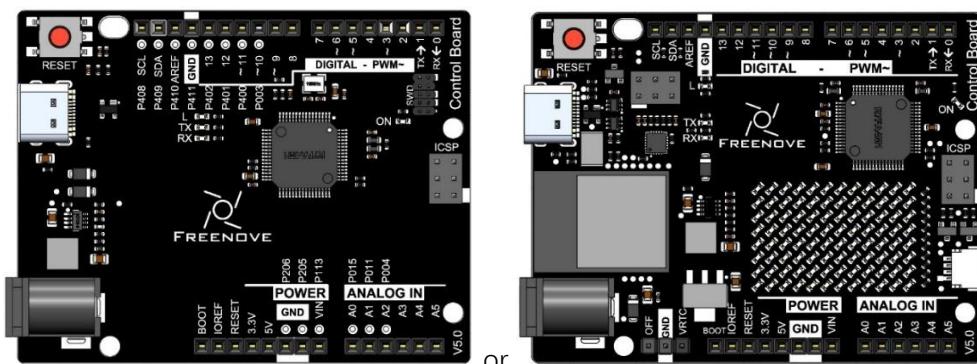
In this chapter, we explore how to access the Real-Time Clock (RTC) on the Control board (V5). The RTC is embedded in the microcontroller (RA4M1) of the Control board (V5).

Project 12.1 RTC

With this project, let's delve into controlling the RTC function of the control board, applying it to flash the indicator every two seconds.

Component List

Control board x1



or

USB cable x1



Component Knowledge

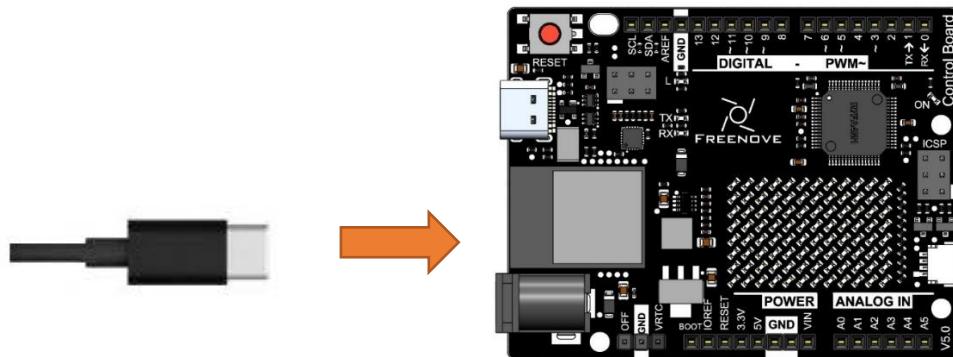
RTC

The Real-Time Clock (RTC) is integrated into the microcontroller (RA4M1) of the control board, serving as an independent timekeeping device that continues to operate even when the primary source of power is off or unavailable. This is achieved thanks to a backup power source, such as a battery. Such feature significantly enhances the RTC's utility across a spectrum of applications, from automating tasks in home systems to timestamping data entries in logging applications.

It should be highlighted that the control board (WiFi) is equipped with a VRTC pin that sustains the RTC's functionality when the board is not powered. To engage this functionality, a voltage between 1.6 and 3.6 V must be supplied to the VRTC pin. It is crucial to stay within this voltage range to avoid damaging the board. It is also important to recognize that this advanced feature is exclusive to the control board with WiFi function and is not available in the standard models.

Circuit

Connect the board to the computer using the USB cable.



Sketch

Sketch 12.1.1

Upload the sketch to the board and you will see the onboard LED flash every two seconds.

```
1 #include "RTC.h"
2
3 volatile bool irqFlag = false; // Flag to indicate a periodic callback.
4 bool ledState = false; // Current LED state: true for ON and false for OFF
5 const int led = LED_BUILTIN; // Pin number for the built-in LED
6
7 void setup() {
8     pinMode(led, OUTPUT); // Configure the LED pin as output
9
10    Serial.begin(9600);
11
12    // Initialize the RTC
13    RTC.begin();
14
15    // Set an arbitrary time to the RTC (required for RTC.setPeriodicCallback to work)
16    RTCTime mytime(13, Month::MAY, 2024, 05, 13, 00, DayOfWeek::MONDAY,
17    SaveLight::SAVING_TIME_ACTIVE);
18
19    RTC.setTime(mytime);
20
21    // Set the periodic callback to trigger every 2 seconds
```

```

20   if (!RTC.setPeriodicCallback(periodicCallback, Period::ONCE_EVERY_2_SEC)) {
21     Serial.println("ERROR: periodic callback not set");
22   }
23 }
24
25 void loop() {
26   // Check if the periodic callback has been triggered
27   if (irqFlag) {
28     Serial.println("Timed CallBack"); // Output a debugging message
29     ledState = !ledState;           // Toggle the LED state
30     digitalWrite(led, ledState);    // Update the LED
31     irqFlag = false;              // Reset the flag
32   }
33 }
34
35 // This is the callback function to be passed to RTC.setPeriodicCallback()
36 void periodicCallback() {
37   // Set the flag to true to indicate the callback has been triggered
38   irqFlag = true;
39 }
```

Initialize RTC.

13	RTC.begin();
----	--------------

Configure the time for RTC.

15	RTCTime mytime(13, Month::MAY, 2024, 05, 13, 00, DayOfWeek::MONDAY,
16	SaveLight::SAVING_TIME_ACTIVE);
17	RTC.setTime(mytime);

A periodic interrupt allows you to set a recurring callback. Set the periodcallback to trigger once every two seconds.

20	if (!RTC.setPeriodicCallback(periodicCallback, Period::ONCE_EVERY_2_SEC)) {
21	Serial.println("ERROR: periodic callback not set");
22	}

Change the status of the LED every two seconds.

27	if (irqFlag) {
28	Serial.println("Timed CallBack"); // Output a debugging message
29	ledState = !ledState; // Toggle the LED state
30	digitalWrite(led, ledState); // Update the LED
31	irqFlag = false; // Reset the flag
32	}

Chapter 13 Motor

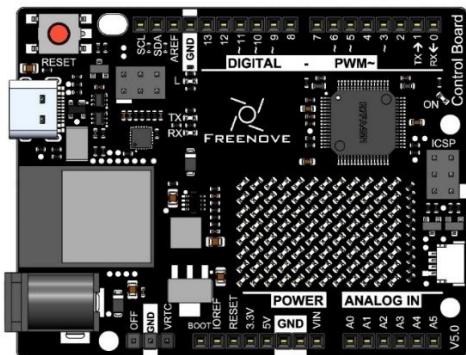
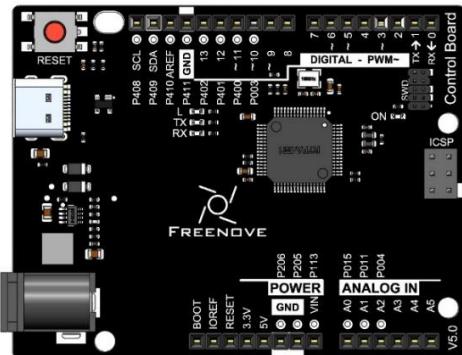
Earlier, we have done a series of interesting projects with control board and basic electronic components. Now, let us study some movable electronic modules. In this chapter, we will learn to control a motor.

Project 13.1 Control Motor by Relay

First, use relay to control a Motor.

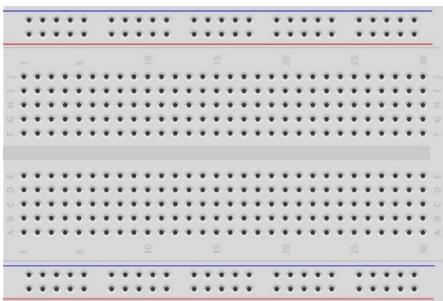
Component List

Control board x1

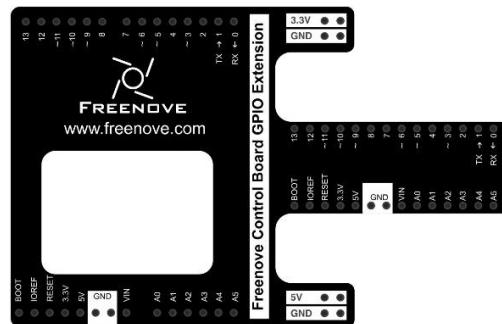


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Jumper M/M x9



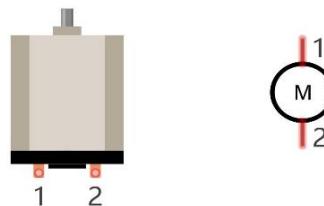
AA Battery holder x1 (Need AA battery x2)	Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1
			

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

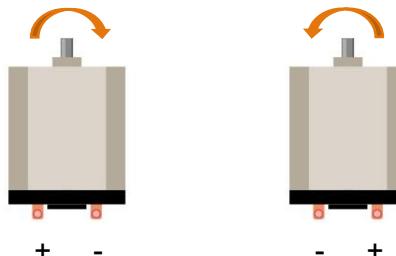
Component Knowledge

DC Motor

DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair of permanent magnets), and it has terminals to connect to the power if it is made up of electromagnet coils. Most Hobby DC Motors only use Permanent Magnets for the Stator Field. The Rotor is usually the shaft of motor with 3 or more electromagnets connected to a commutator where the brushes (via the terminals 1 & 2 below) supply electrical power, which can drive other mechanical devices. The diagram below shows a small DC Motor with two terminal pins.



When a 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. This is important to note.



Capacitor

The unit of capacitance(C) is farad (F). $1F = 1000000\mu F$, $1\mu F = 1000000pF$.

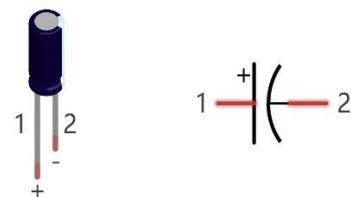
A Capacitor is an energy storage device, with a certain capacitance. When capacitor's voltage increases, the

capacitor will be charged. And capacitor will be discharged when its voltage drops. So capacitor's voltage of both ends is not transient. According to this characteristic, capacitor is often used to stabilize the voltage of power supply, and filter the signal. Capacitor with large capacity can filter out low frequency signals, and small-capacity capacitor can filter out high frequency signals.

The capacitor has a non-polar capacitor and a polar capacitor. Generally, non-polar capacitor has small capacitance, and a ceramic non-polar capacitor is shown below.



For polar capacitor, it usually has larger capacitance, and an electrolytic polar capacitor of that is shown below:

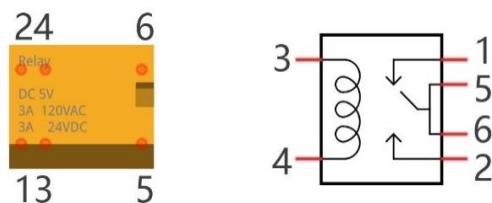


When the motor rotates, it will generate noise. As the contact of coil connects and disconnects the electrode constantly, it will cause the supply voltage unstable. Thus, a small capacitor is often connected to motor to reduce the impact on power supply from motor.

Relay

Relays are a type of Switch that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit using an electromagnet to initiate the Switch action. When the electromagnet is energized (powered), it will attract internal contacts completing a circuit, which act as a Switch. Many times Relays are used to allow a low powered circuit (and a small low amperage switch) to safely turn ON a larger more powerful circuit. They are commonly found in automobiles, especially from the ignition to the starter motor.

The following is the image and circuit symbol diagram of the 5V relay used in this project:



Pin 5 and pin 6 are internally connected to each other. When the coil pin 3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

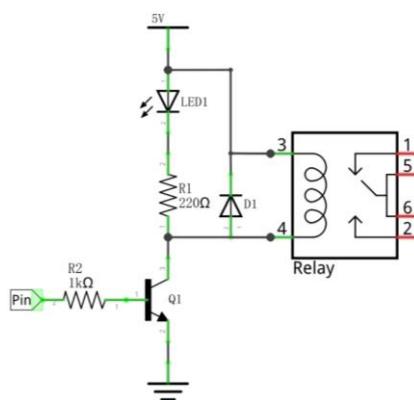
Inductor

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: $1\text{H}=1000\text{mH}$, $1\text{mH}=1000\mu\text{H}$.

An inductor is a passive device that stores energy in its magnetic field and returns energy to the circuit whenever required. An inductor is formed by a Cylindrical Core with many turns of conducting wire (usually copper wire). Inductors will hinder the changing current passing through it. When the current passing through the inductor increases, it will attempt to hinder the increasing movement of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing movement of current. So the current passing through an inductor is not transient.



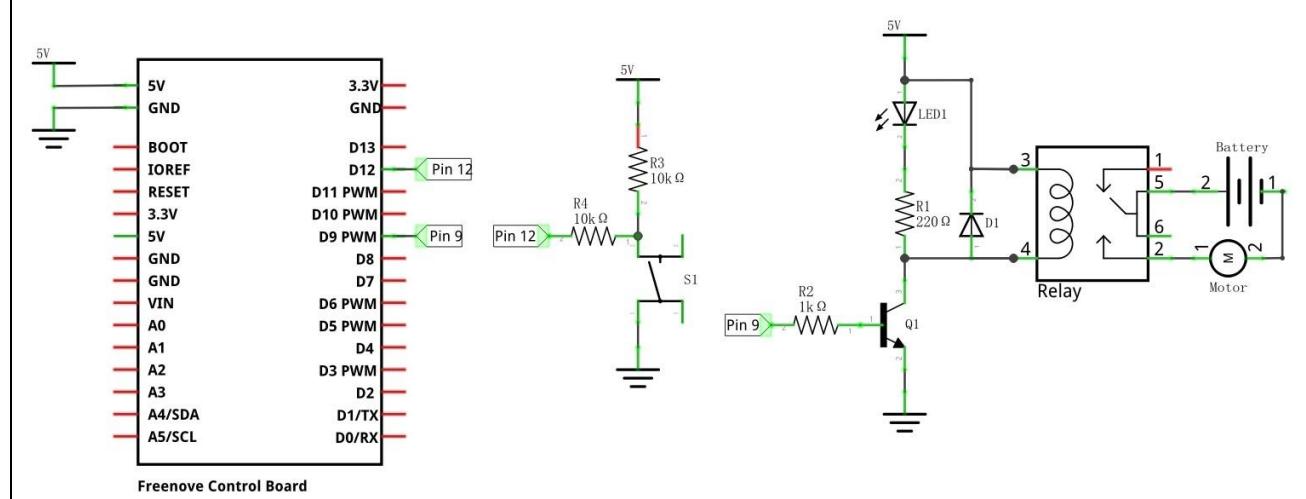
The circuit for a relay is as follows: The coil of relay can be equivalent to an Inductor, when a transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes in parallel are placed on both ends of the relay coil pins in opposite polar direction. Having the current pass through the diodes will avoid any adverse effect on the power supply.



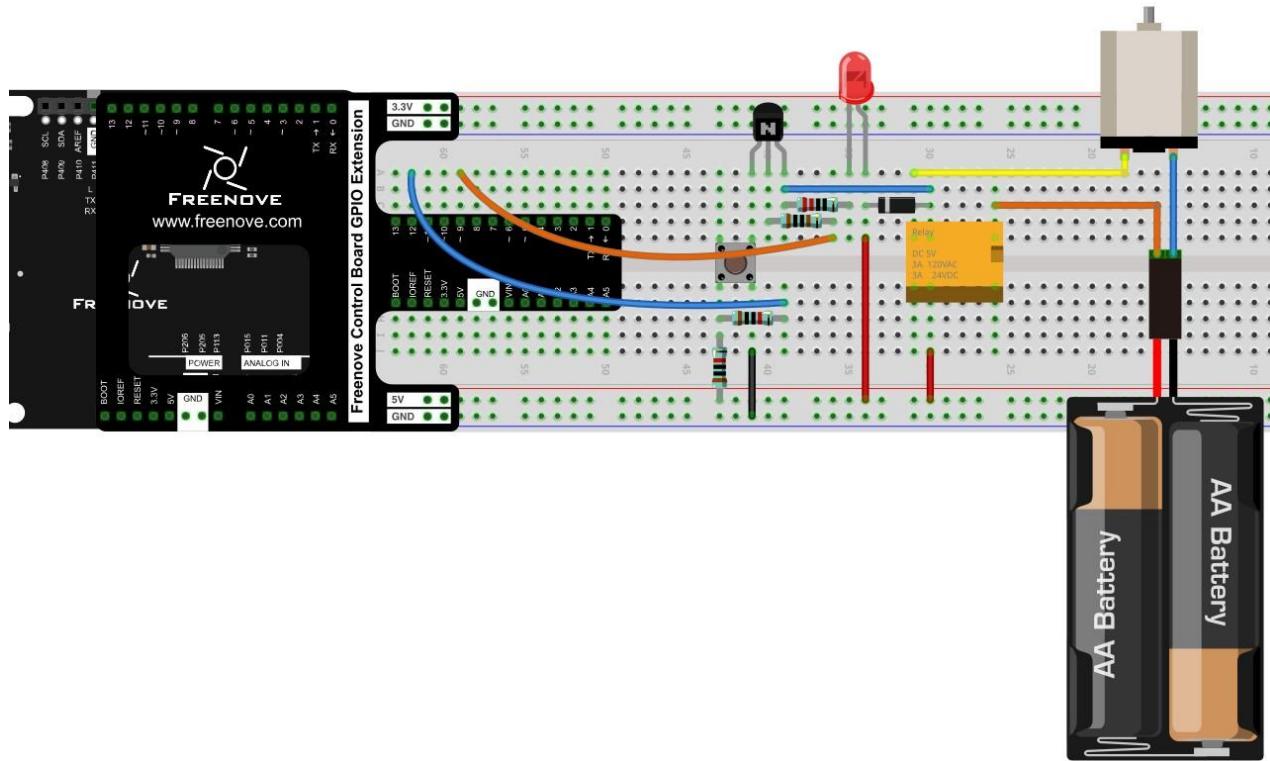
Circuit

Use pin 12 of control board to detect the state of push button switch, and pin 9 to control the relay. As the running of motor needs larger power, we will use two AA batteries to supply power for the motor alone.

Schematic diagram



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



The DC electric power here can also be powered using 5V or 3.3V on the control board.

Sketch

Sketch 13.1.1

Now, write code to detect the state of push button switch. Each time you press the button, the switching status of relay will change. So we control the motor to rotate or stop in this way.

```

1 int relayPin = 9;      // the number of the relay pin
2 int buttonPin = 12;    // the number of the push button pin
3
4 int buttonState = HIGH; // Record button state, and initial the state to high level
5 int relayState = LOW;  // Record relay state, and initial the state to low level
6 int lastButtonState = HIGH; // Record the button state of last detection
7 long lastChangeTime = 0; // Record the time point for button state change
8
9 void setup() {
10   pinMode(buttonPin, INPUT); // Set push button pin into input mode
11   pinMode(relayPin, OUTPUT); // Set relay pin into output mode
12   digitalWrite(relayPin, relayState); // Set the initial state of relay into "off"
13   Serial.begin(9600);           // Initialize serial port, and set baud rate to 9600
14 }
15

```

```

16 void loop() {
17     int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
18     // If button pin state has changed, record the time point
19     if (nowButtonState != lastButtonState) {
20         lastChangeTime = millis();
21     }
22     // If button state changes, and stays stable for a while, then it should have skipped the bounce area
23     if (millis() - lastChangeTime > 10) {
24         if (buttonState != nowButtonState) { // Confirm button state has changed
25             buttonState = nowButtonState;
26             if (buttonState == LOW) { // Low level indicates the button is pressed
27                 relayState = !relayState; // Reverse relay state
28                 digitalWrite(relayPin, relayState); // Update relay state
29                 Serial.println("Button is Pressed!");
30             }
31             else { // High level indicates the button is released
32                 Serial.println("Button is Released!");
33             }
34         }
35     }
36     lastButtonState = nowButtonState; // Save the state of last button
37 }
```

In this code, we used a new method to detect the button's state. In the loop() function, the level state of button pin is detected constantly. When the level is changed, record its time point. If the level has not changed after a while, it will be considered that the bounce area has already been skipped. Then, judge whether the button is pressed or released according to button pin state.

First, define two variables to record the state of the button and the relay.

```

4 int buttonState = HIGH; // Record button state, initial the state into high level
5 int relayState = LOW; // Record relay state, initial the state into low level
```

Define a variable to record button pin's state of last detection.

```
6 int lastButtonState = HIGH; // Record the button state of last detection
```

Define a variable to record the time of the last button pin change.

```
7 long lastChangeTime = 0; // Record the time point for button state change
```

In the loop() function, the detected pin state of button will be compared with the last detected state. If it changes, record this time point.

```

16 void loop() {
17     int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
18     // If the state of button pin has changed, record the time point
19     if (nowButtonState != lastButtonState) {
20         lastChangeTime = millis();
21     }
22     ...
23     lastButtonState = nowButtonState; // Save the state of last button
24 }
```

	37 } }
--	--------

If the level stays unchanged over a period of time, it is considered that the bounce area has already been skipped.

	23 if (millis() - lastChangeTime > 10) { 35 }
--	--

After the pin state stays stable, the changed state of button is confirmed, then it will be recorded for the next comparison.

	24 if (buttonState != nowButtonState) { // Confirm button state has changed 25 buttonState = nowButtonState; 34 }
--	--

Judge whether the button is pressed or released according to button pin level, print button information to serial port, and reverse relay when the button is pressed.

	26 if (buttonState == LOW) { // Low level indicates the button is pressed 27 relayState = !relayState; // Reverse relay state 28 digitalWrite(relayPin, relayState); // Update relay state 29 Serial.println("Button is Pressed!"); 30 } 31 else { // High level indicates the button is released 32 Serial.println("Button is Released!"); 33 }
--	---

This button detecting method does not put program into the state of delay waiting, you can increase the efficiency of code execution.

	millis()
--	----------

Returns the number of milliseconds since the control board began running the current program.

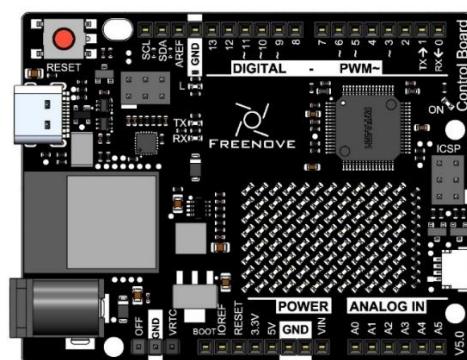
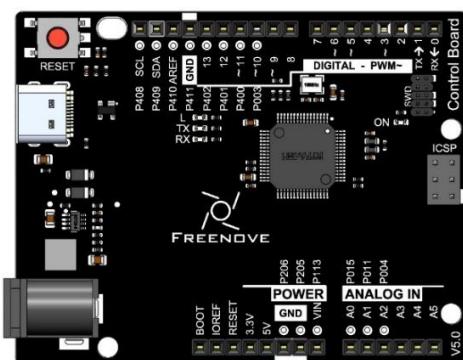
Verify and upload the code, every time you press the push button, the state of relay and motor changes once.

Project 13.2 Control Motor with L293D

Now, we will use dedicated chip L293D to control the motor.

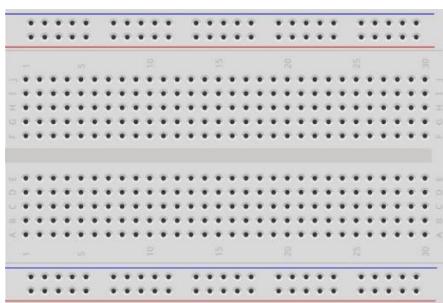
Component List

Control board x1

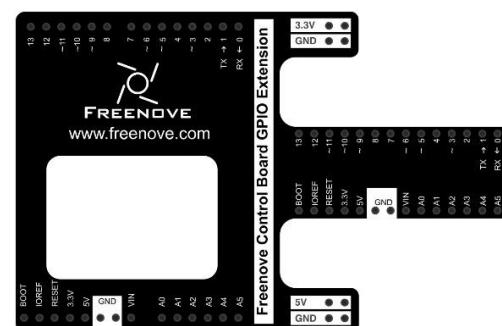


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Jumper M/M x10

Jumper F/M x2



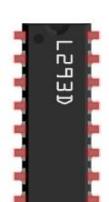
AA Battery holder x1



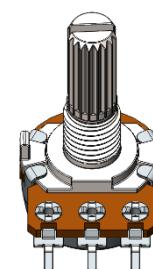
Motor x1



L293D x1



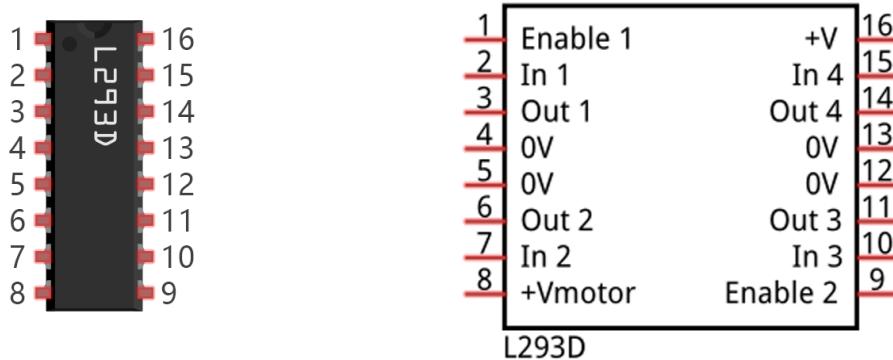
Rotary potentiometer x1



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).



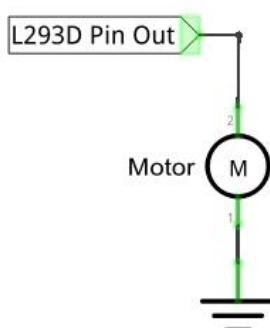
Port description of L293D module is as follows:

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 4.5~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

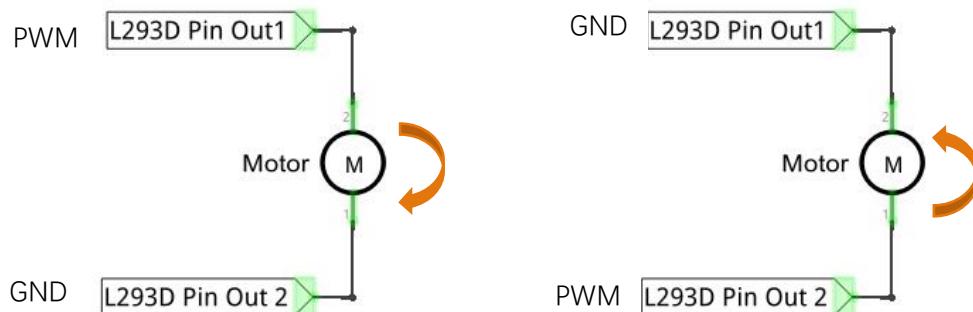
For more details, please see datasheet.

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

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



The following connection uses two channels of the 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 controls the speed of motor, but also can control the steering of the motor.

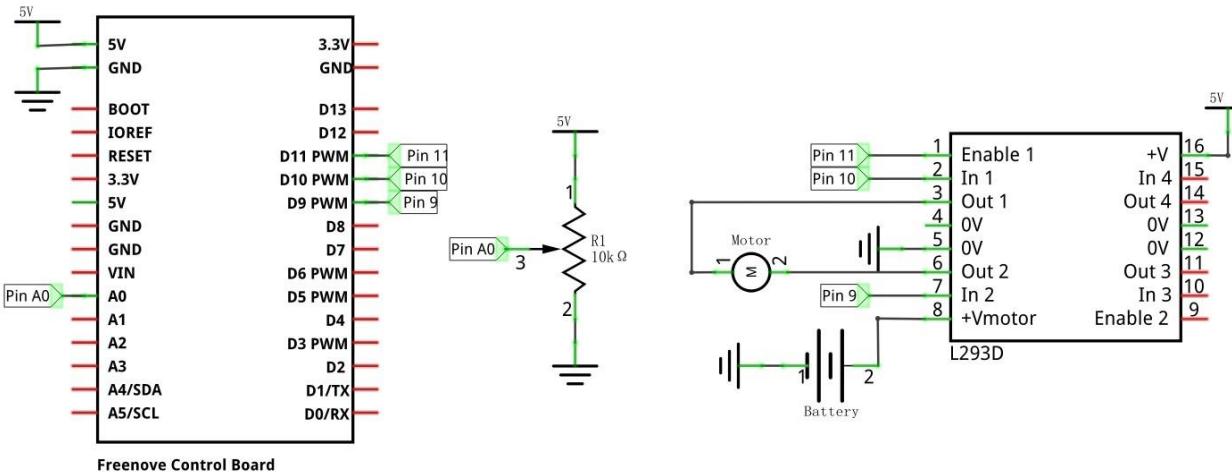


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

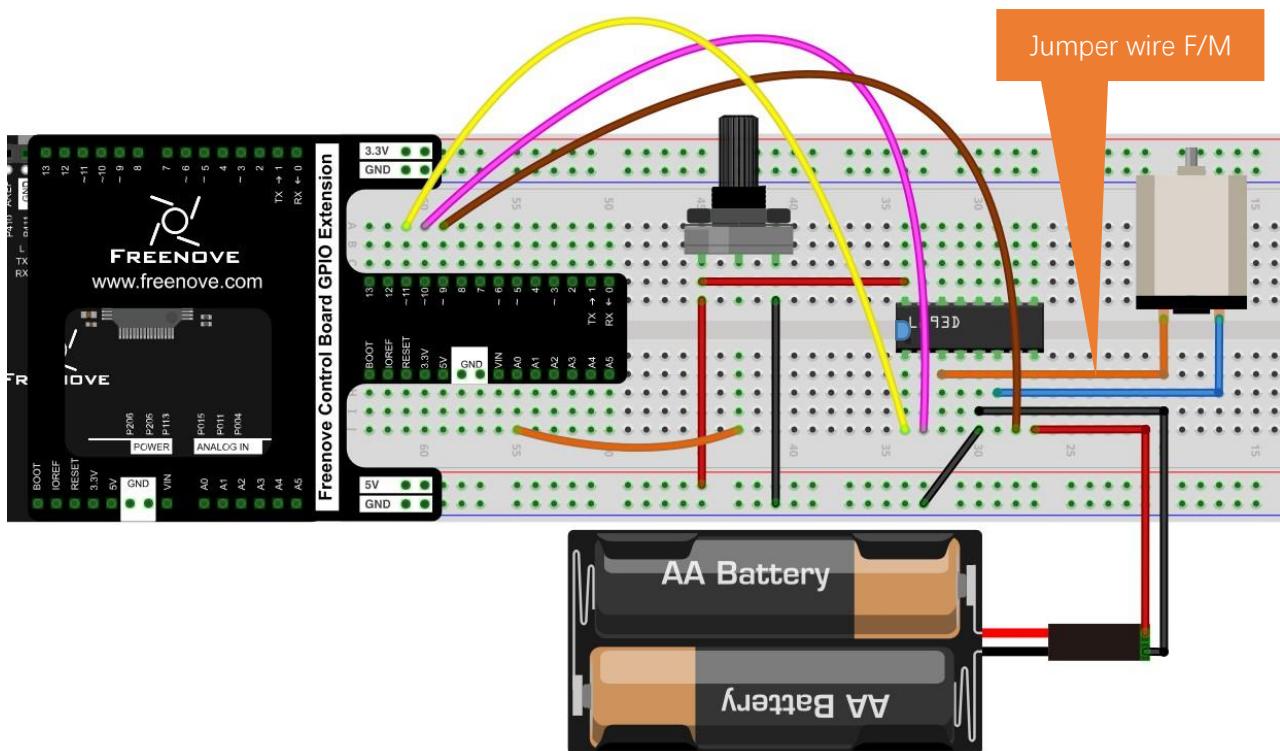
Circuit

Use pin A0 of the control board to detect the voltage of rotary potentiometer; pin 9 and pin 10 to control the motor's rotation direction and pin 11 to output PWM wave to control motor speed.

Schematic diagram



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



The DC electric power here can also be powered using 5V or 3.3V on the control board.

Sketch

Sketch 13.2.1

Now, write the code to control speed and rotation direction of motor through rotary potentiometer. When the potentiometer stays in the middle position, motor speed will be minimum, and when deviates intermediate position, the speed will increase. Also, if the potentiometer deviates from the middle position of potentiometer clockwise or counterclockwise, the rotation direction of the motor is different.

```
1 int in1Pin = 10;      // Define L293D channel 1 pin
2 int in2Pin = 9;       // Define L293D channel 2 pin
3 int enable1Pin = 11;  // Define L293D enable 1 pin
4 boolean rotationDir; // Define a variable to save the motor's rotation direction, true and false are
                       // represented by positive rotation and reverse rotation.
5 int rotationSpeed;   // Define a variable to save the motor rotation speed
6
7 void setup() {
8     // Initialize the pin into an output mode:
9     pinMode(in1Pin, OUTPUT);
10    pinMode(in2Pin, OUTPUT);
11    pinMode(enable1Pin, OUTPUT);
12 }
13
14 void loop() {
15     int potenVal = analogRead(A0); // Convert the voltage of rotary potentiometer into digital
16     // Compare the number with value 512, if more than 512, clockwise rotates, otherwise, counter
17     // clockwise rotates
18     rotationSpeed = potenVal - 512;
19     if (potenVal > 512)
20         rotationDir = true;
21     else
22         rotationDir = false;
23     // Calculate the motor speed, the far number of deviation from the middle value 512, the faster the
24     // control speed will be
25     rotationSpeed = abs(potenVal - 512);
26     // Control the steering and speed of the motor
27     driveMotor(rotationDir, map(rotationSpeed, 0, 512, 0, 255));
28 }
29
30 void driveMotor(boolean dir, int spd) {
31     // Control motor rotation direction
32     if (rotationDir) {
33         digitalWrite(in1Pin, HIGH);
34         digitalWrite(in2Pin, LOW);
35     }
```

```

34     else {
35         digitalWrite(in1Pin, LOW);
36         digitalWrite(in2Pin, HIGH);
37     }
38     // Control motor rotation speed
39     analogWrite(enable1Pin, constrain(spd, 0, 255));
40 }
```

In the code, we write a function to control the motor, and control the speed and steering through two parameters.

```

28 void driveMotor(boolean dir, int spd) {
29     // Control motor rotation direction
30     if (rotationDir) {
31         digitalWrite(in1Pin, HIGH);
32         digitalWrite(in2Pin, LOW);
33     }
34     else {
35         digitalWrite(in1Pin, LOW);
36         digitalWrite(in2Pin, HIGH);
37     }
38     // Control motor rotation speed
39     analogWrite(enable1Pin, constrain(spd, 0, 255));
40 }
```

In the loop () function, detect the digital value of rotary potentiometer, and convert it into the motor speed and steering through calculation.

```

14 void loop() {
15     int potenVal = analogRead(A0); // Convert the voltage of rotary potentiometer into digital
16     // Compare the digital number with middle value 512, if more than 512, clockwise rotates, otherwise,
17     counter clockwise rotates
18     rotationSpeed = potenVal - 512;
19     if (potenVal > 512)
20         rotationDir = true;
21     else
22         rotationDir = false;
23     // Calculate the motor speed, the far number of deviation from the middle value 512, the faster the
24     control speed will be
25     rotationSpeed = abs(potenVal - 512);
26     // Control the steering and speed of the motor
27     driveMotor(rotationDir, map(rotationSpeed, 0, 512, 0, 255));
28 }
```

abs(x)

Computes the absolute value of a number.

Verify and upload the code, turn the shaft of rotary potentiometer, and then you can see the change of the motor speed and direction.

Chapter 14 Servo

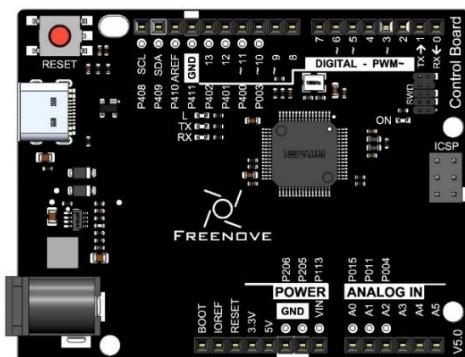
Earlier, we have used control board and L293D module to control the motor speed and steering. Now, we will use another motor, servo, which can rotate to a certain angle.

Project 14.1 Servo Sweep

First, let's get the servo to rotate.

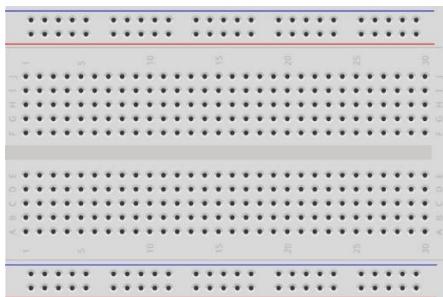
Component List

Control board x1

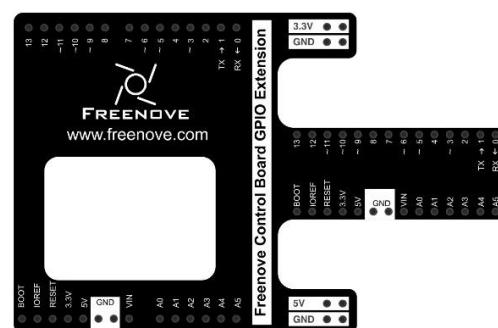


The image shows the front side of a Freenove Uno R3 microcontroller board. It features an ATmega328P microcontroller chip at the center. Various pins are labeled: GND, AREF, AREF+, SCL, SDA, I2C, TXD, RXD, and several digital and analog pins (e.g., D0-D13, A0-A5). A USB port is located on the left. A large, semi-transparent watermark with the text "FREENOVE" and a stylized logo is centered on the board.

Breadboard x1



GPIO Extension Board x1



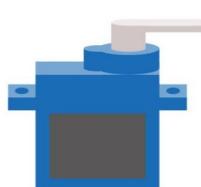
USB cable x1



Jumper M/M x3



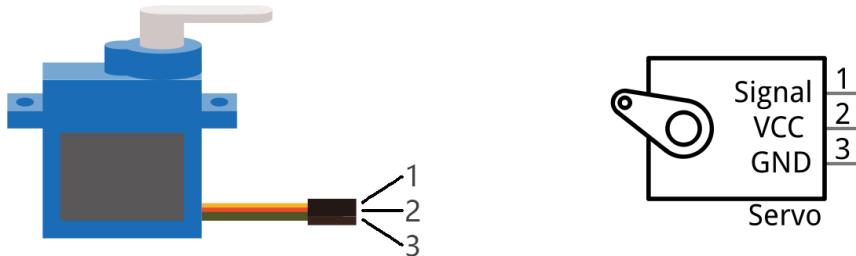
Servo x1



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 lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

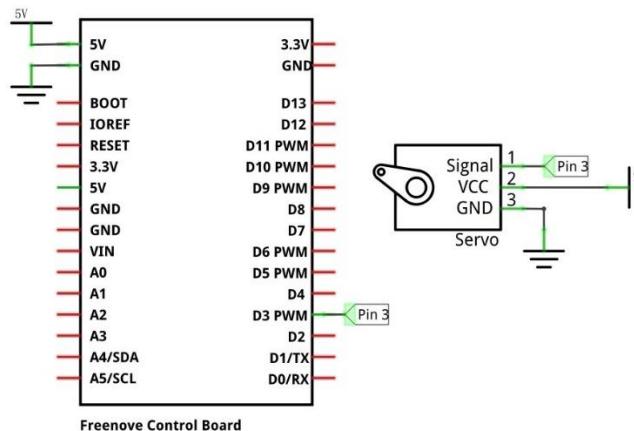
When you change the servo signal, the servo will rotate to the designated position.

Circuit

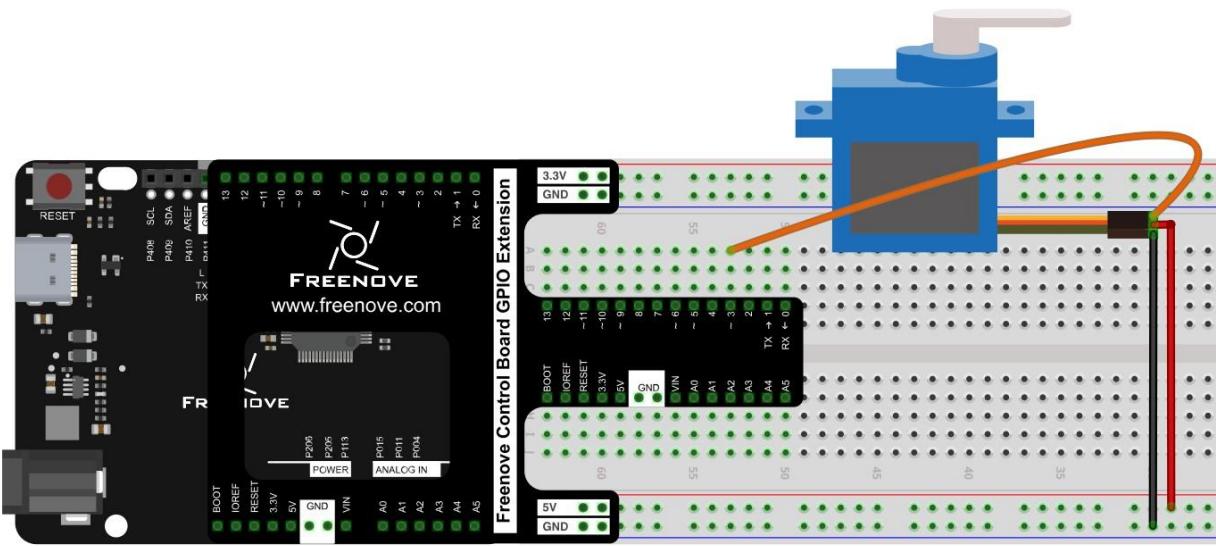
Use pin 3 of the control board to drive the servo.

Pay attention to the color of servo lead wire: VCC (red), GND (brown), and signal line (orange). The wrong connection can cause damage to servo.

Schematic diagram



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



Sketch

Sketch 14.1.1

Now, write the code to control servo, making it sweep in the motion range continuously.

```

1 #include <Servo.h>
2
3 Servo myservo; // create servo object to control a servo
4
5 int pos = 0; // variable to store the servo position
6 int servoPin = 3; // define the pin of servo signal line

```

```

7
8 void setup() {
9     myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
10 }
11
12 void loop() {
13     for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
14         // in steps of 1 degree
15         myservo.write(pos);           // tell servo to go to position in variable "pos"
16         delay(15);                 // waits 15ms for the servo to reach the position
17     }
18     for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
19         myservo.write(pos);           // tell servo to go to position in variable "pos"
20         delay(15);                 // waits 15ms for the servo to reach the position
21     }
22 }
```

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

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

Servo library provides the Servo class that controls it. Different from previous Serial class, the Servo class must be instantiated before you use it:

```
3 Servo myservo;           // create servo object to control a servo
```

The code above defines an object of Servo type, myservo.

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.
```

After the Servo object is defined, it can refer to the function, such as initializing the servo:

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

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

```
15 myservo.write(pos);           // tell servo to go to position in variable "pos"
```

In the loop () function, we use the loop to control the servo to rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, then repeat the cycle all the time.

Verify and upload the code, the servo starts to sweep continuously.

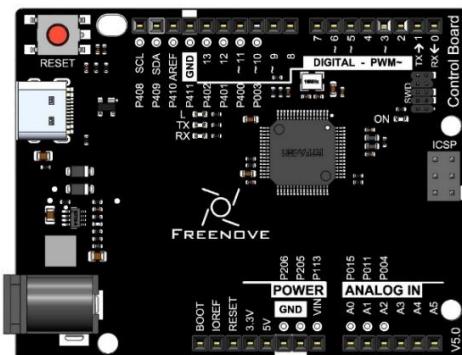


Project 14.2 Control Servo with Potentiometer

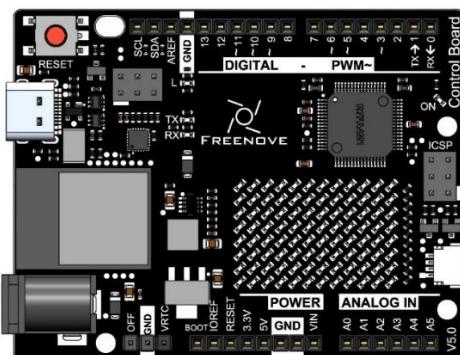
In the previous section, we've made the servo sweep continuously. Now, we will use a potentiometer to control the servo's angle.

Component List

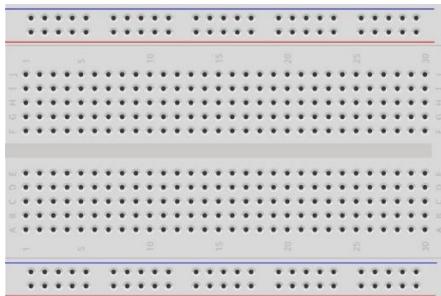
Control board x1



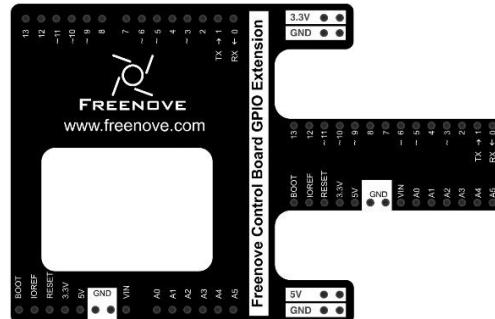
or



Breadboard x1



GPIO Extension Board x1



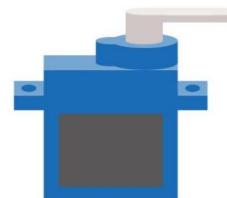
USB cable x1



Rotary potentiometer x1



Servo x1



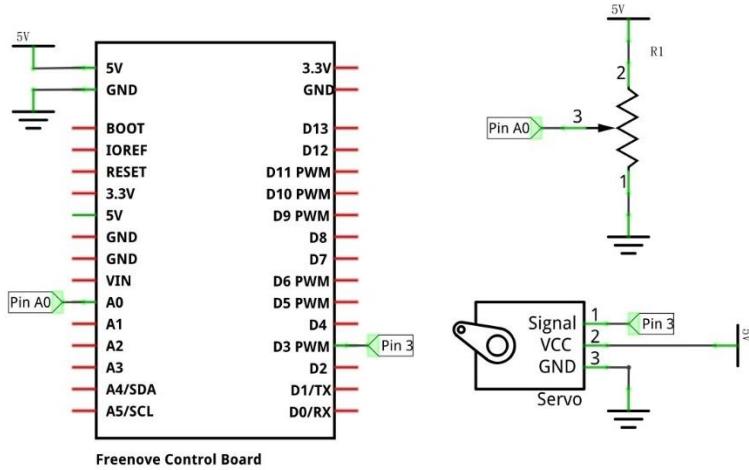
Jumper M/M x6



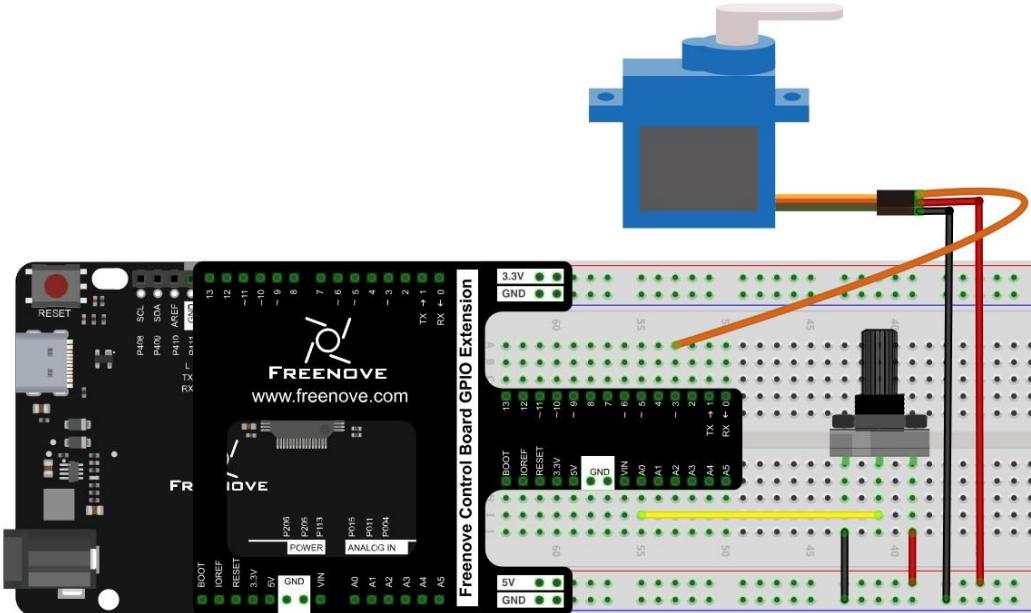
Circuit

Use pin A0 of the control board to detect the voltage of rotary potentiometer, and pin 3 to drive the servo.

Schematic diagram



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



Sketch

Sketch 14.2.1

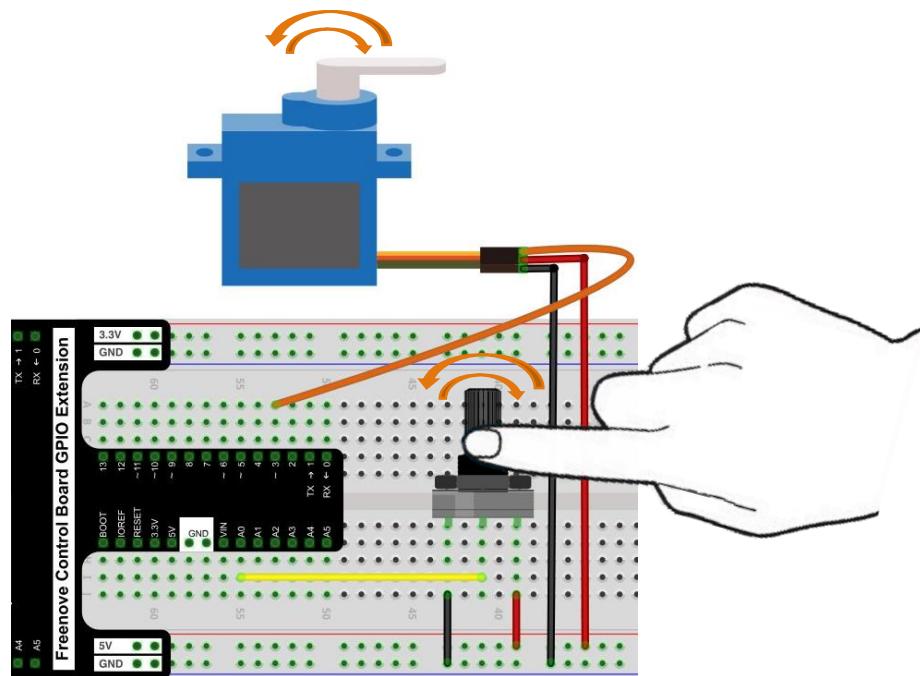
Now, write the code to detect the voltage of rotary potentiometer, and control servo to rotate to a different angle according to that.

```

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

In the code, we obtain the ADC value of pin A0, 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 15 Temperature Sensor

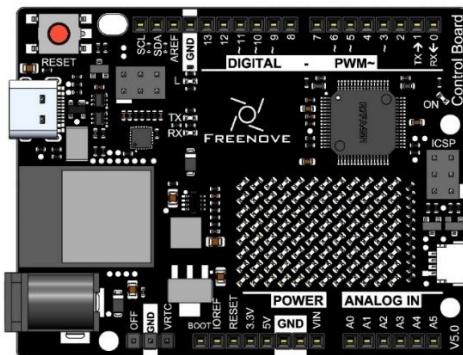
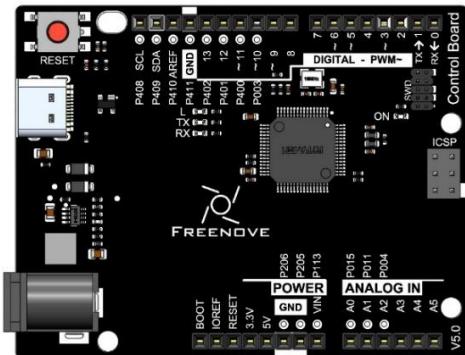
Earlier, we have used control board and photoresistor to detect the intensity of light. Now, we will learn to use the temperature sensor.

Project 15.1 Detect the Temperature

We will use a thermistor to detect the ambient temperature.

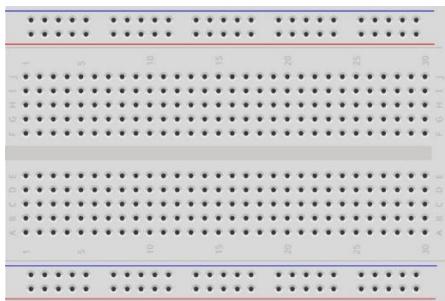
Component List

Control board x1

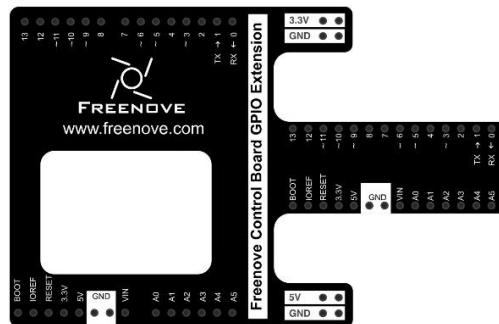


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Thermistor x1



Resistor 10kΩ x1



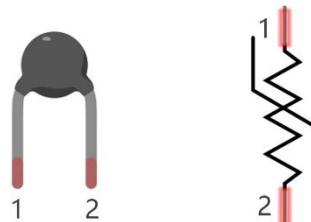
Jumper M/M x3



Component Knowledge

Thermistor

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 thermistor is:

$$R_t = R \cdot \exp[B \cdot (1/T_2 - 1/T_1)]$$

Where:

R_t is the thermistor resistance under T_2 temperature;

R is in the nominal resistance of thermistor under T_1 temperature;

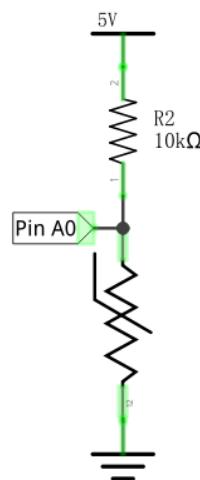
$\exp[n]$ is nth power of e;

B is for thermal index;

T_1, T_2 is Kelvin temperature (absolute temperature). Kelvin temperature = 273.15 + celsius temperature.

Parameters of the thermistor we use is: $B=3950$, $R=10k$, $T_1=25$.

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

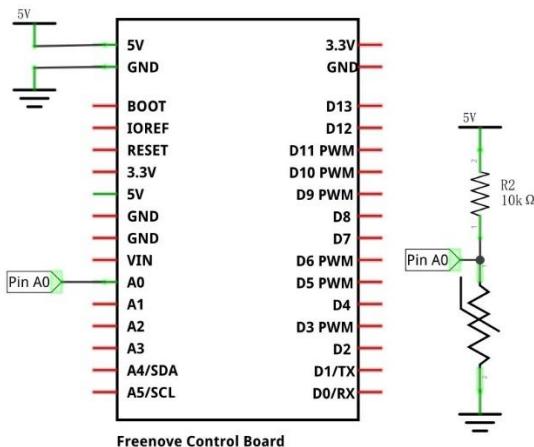


We can use the value measured by the analog pin of control board to obtain resistance value of the thermistor, and then we can use the formula to obtain the temperature value.

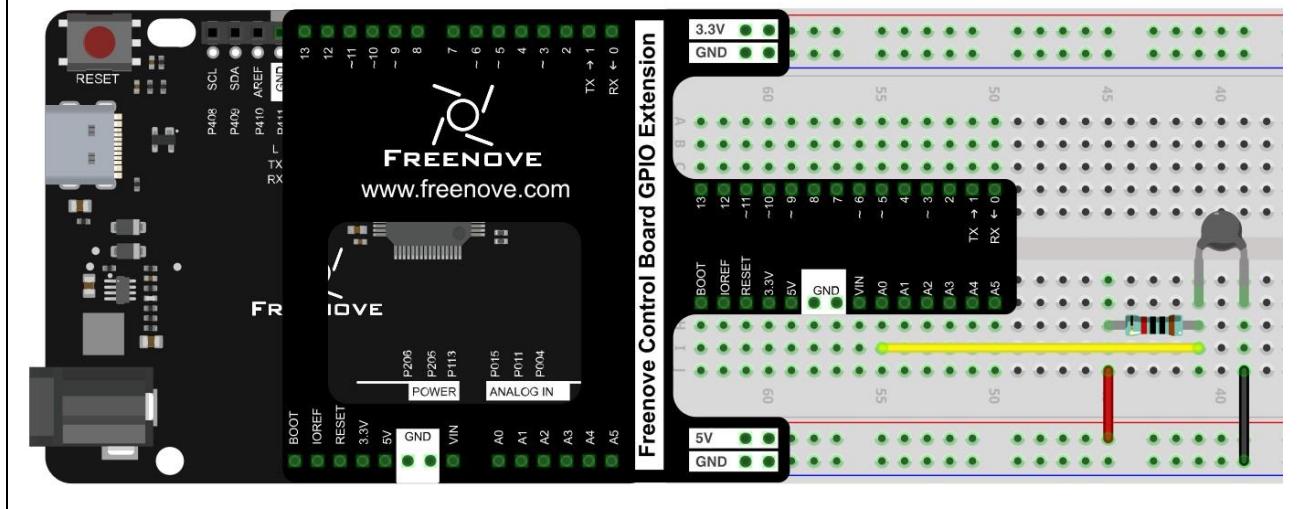
Circuit

Use pin A0 on the control board to detect the voltage of thermistor.

Schematic diagram



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



Sketch

Sketch 15.1.1

Now, write the code to detect the voltage value of thermistor, calculate the temperature value, and send it to Serial Monitor.

```

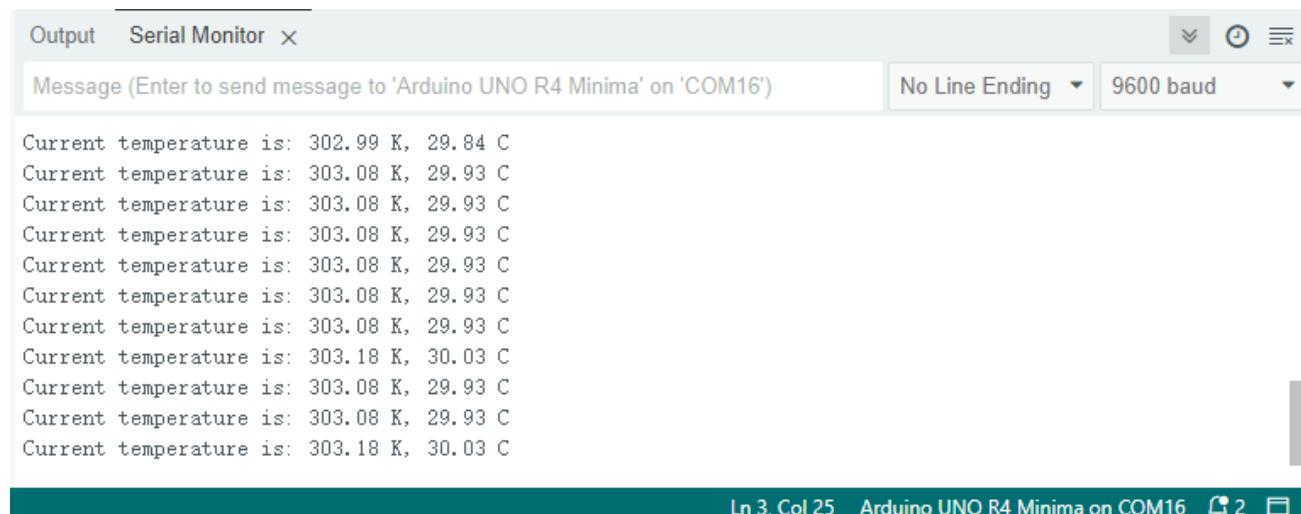
1 void setup() {
2     Serial.begin(9600);           // Initialize the serial port, set the baud rate
3     // into 9600
4
5 void loop() {
6     // Convert analog value of A0 port into digital value

```

```
7 int adcVal = analogRead(A0);  
8 // Calculate voltage  
9 float v = adcVal * 5.0 / 1024;  
10 // Calculate resistance value of thermistor  
11 float Rt = 10 * v / (5 - v);  
12 // Calculate temperature (Kelvin)  
13 float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));  
14 // Calculate temperature (Celsius)  
15 float tempC = tempK - 273.15;  
16  
17 // Send the result to computer through serial port  
18 Serial.print("Current temperature is: ");  
19 Serial.print(tempK);  
20 Serial.print(" K, ");  
21 Serial.print(tempC);  
22 Serial.println(" C");  
23 delay(500);  
24 }
```

In the code, we obtain the ADC value of pin A0, and convert it into temperature value, and then send it to the serial port.

Verify and upload the code, open the Serial Monitor, and then you will see the temperature value sent from control board.



Chapter 16 Joystick

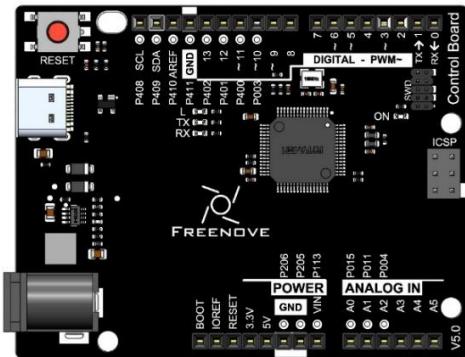
In the previous chapter, we have learned how to use rotary potentiometer. Now, let us learn a new electronic module Joystick that works on the same principle as rotary potentiometer.

Project 16.1 Joystick

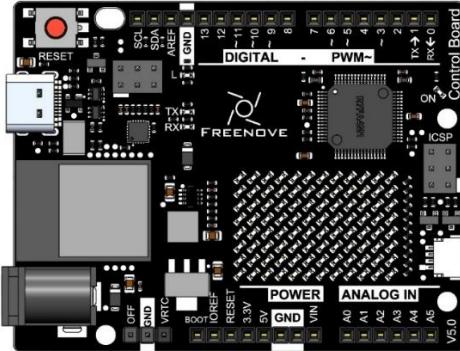
We will use the serial port to get Joystick data.

Component List

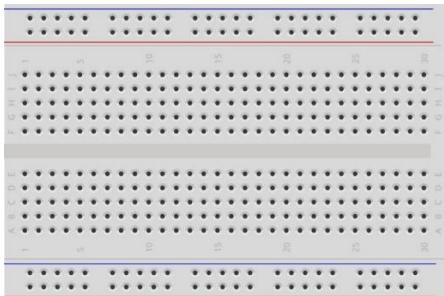
Control board x1



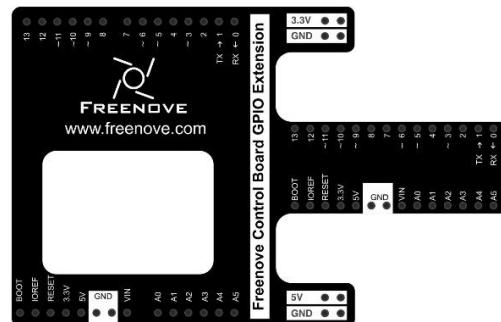
or



Breadboard x1



GPIO Extension Board x1



USB cable x1



Joystick x1



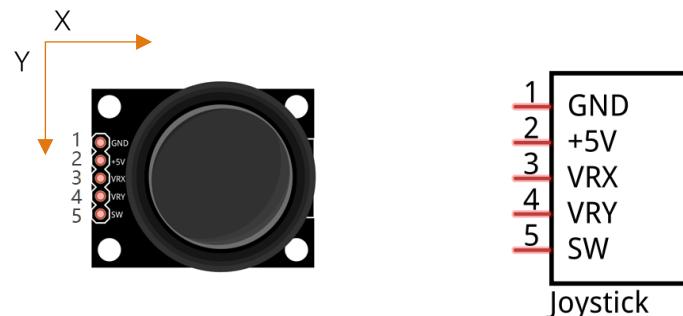
Jumper F/M x5



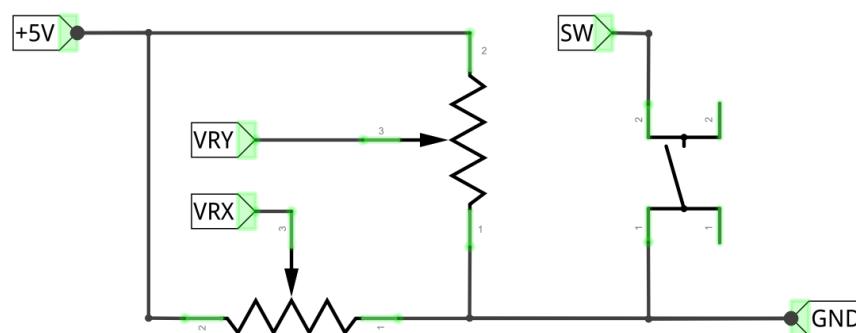
Component Knowledge

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)**.



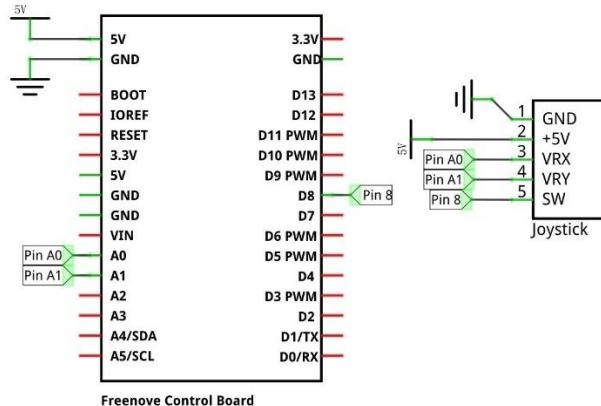
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



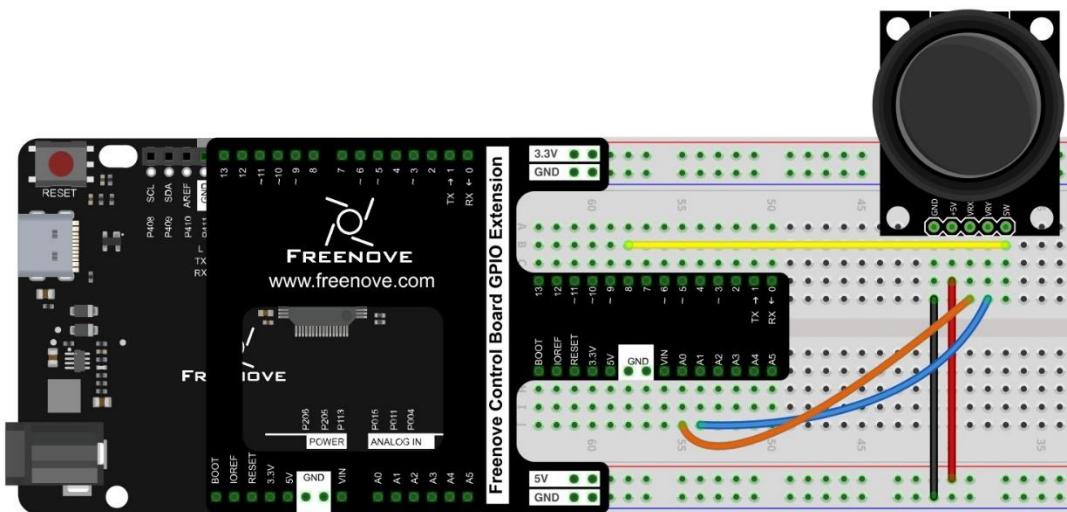
Circuit

Use pin A0 and pin A1 on control board to detect the voltage value of two rotary potentiometers inside Joystick, and use pin 8 port to detect the vertical button.

Schematic diagram



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



Sketch

Sketch 16.1.1

Now write the sketch to detect the voltage value of these two rotary potentiometers and the state of the button in vertical direction, then sent the data to Serial Monitor window.

```

1 int xAxisPin = 0;           // define X pin of Joystick
2 int yAxisPin = 1;           // define Y pin of Joystick
3 int zAxisPin = 8;           // define Z pin of Joystick
4 int xVal, yVal, zVal;       // define 3 variables to store the values of 3 direction
5
6 void setup() {
7     pinMode(zAxisPin, INPUT_PULLUP); // initialize the port to pull-up input

```

```

8   Serial.begin(9600);           // initialize the serial port with baud rate 9600
9 }
10
11 void loop() {
12   // read analog value in XY axis
13   xVal = analogRead(xAxisPin);
14   yVal = analogRead(yAxisPin);
15   // read digital value of switch in Z axis
16   zVal = digitalRead(zAxisPin);
17   //print the data read above
18   Serial.print("X : ");
19   Serial.print(xVal);
20   Serial.print("\t Y : ");
21   Serial.print(yVal);
22   Serial.print("\t Z : ");
23   Serial.println(zVal);
24   delay(200);
25 }
```

In the code, we get the ADC value of pin A0, A1 and the state of button, and then sent the data to serial port.

INPUT_PULLUP

Set the port to INPUT_PULLUP mode, which is equivalent to configuring the port to INPUT mode, then connect a resistor with high resistance value to VCC behind the port.

Push button of joystick is left hanging when it is not pressed (connected to no circuits with certain voltage value). The results of push button port read by control board are not fixed. So we can set this port to INPUT_PULLUP mode. Then when the push button is not pressed, the state of the port is high. But if it is pressed, the state turns into low level.

Verify and upload the code, open the Serial Monitor, and you can see the Joystick state value sent by control board. Shift and press the rocker of joystick with your finger, and you can see the change of value.



Chapter 17 Acceleration sensor

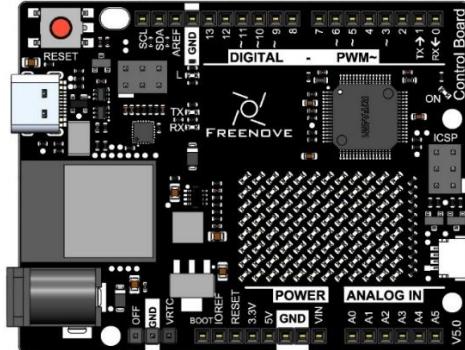
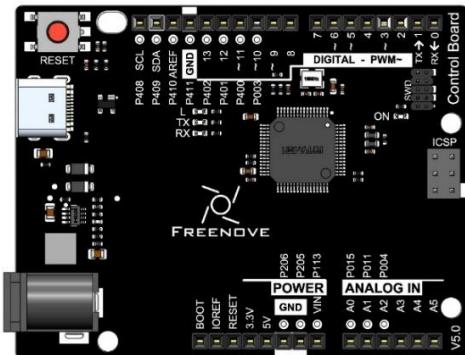
In the previous chapter, we have learned sensors that are used to detect light or temperature. Now we will learn a sensor that can detect acceleration.

Project 17.1 Acceleration Detection

We will use serial port to get the data of MPU6050 module.

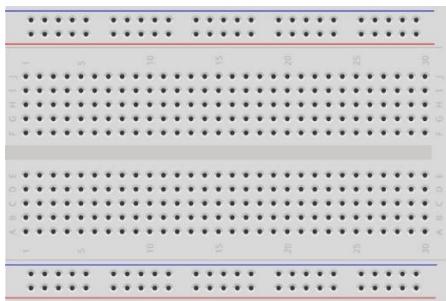
Component List

Control board x1

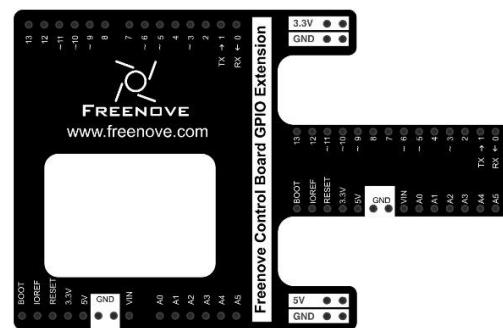


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



MPU6050 module x1



Jumper M/M x4



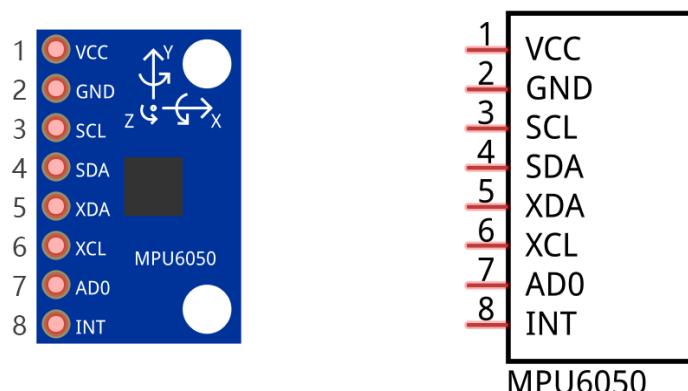
Component Knowledge

I2C communication

I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used to the connection of micro controller and its 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.

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.



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 data 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

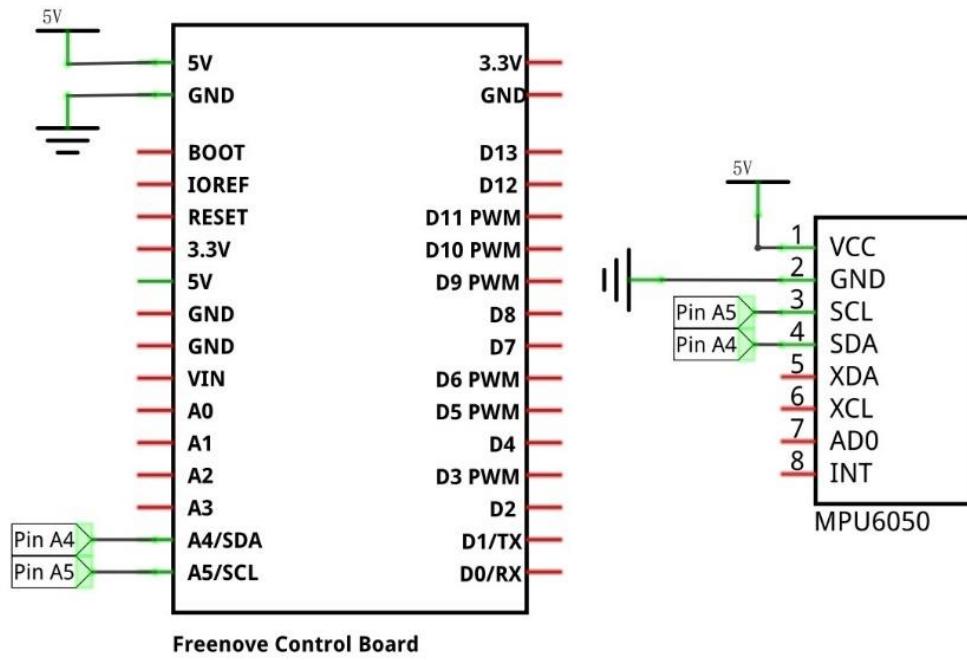
For more detail, please refer to datasheet.

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.

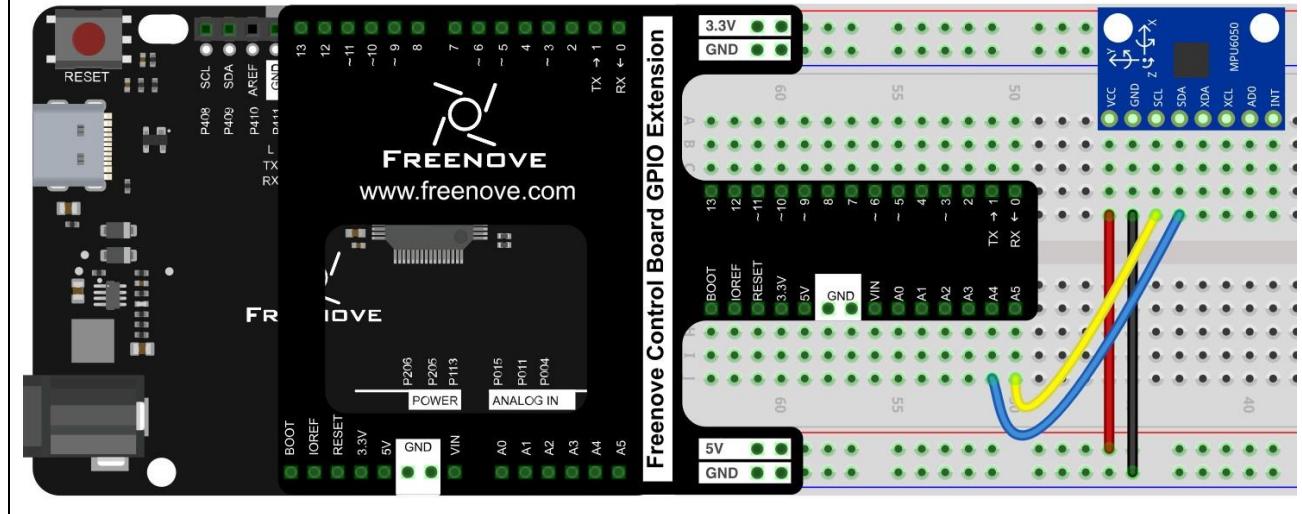
Circuit

Use pin A4/SDA, pin A5/SCL port on the control board to communicate with MPU6050 module.

Schematic diagram



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

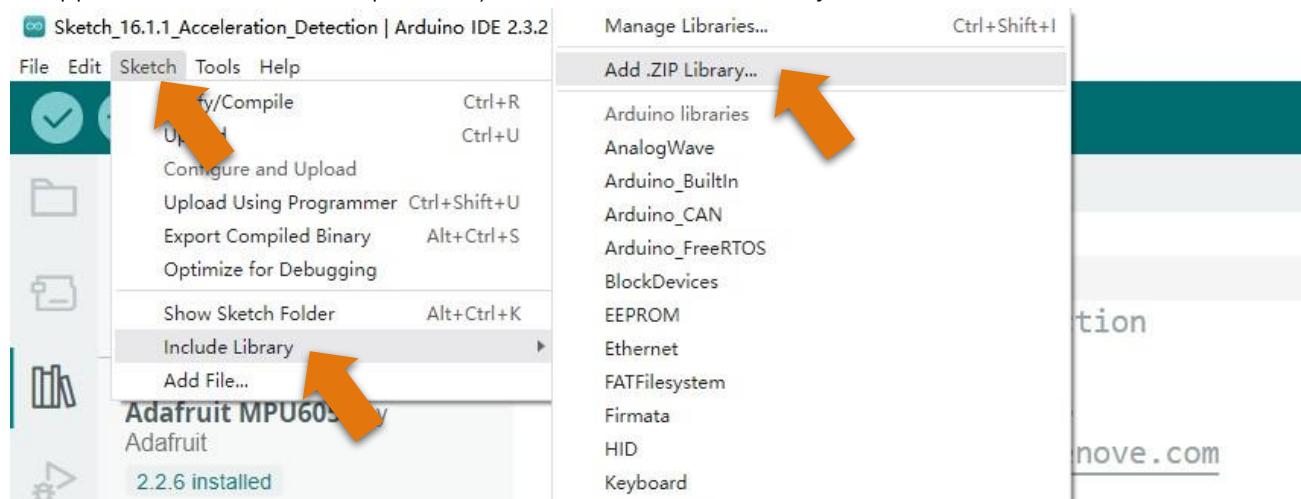


Sketch

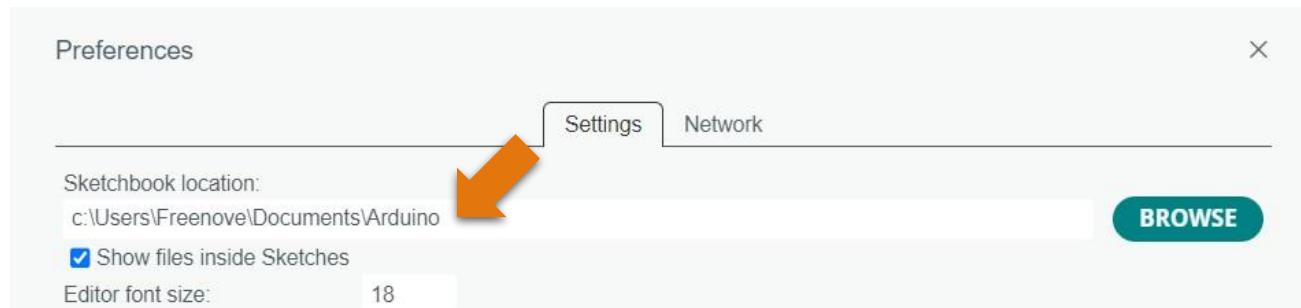
Sketch 17.1.1

Library is a collection of code. We can use code provided by libraries to make programming simple.

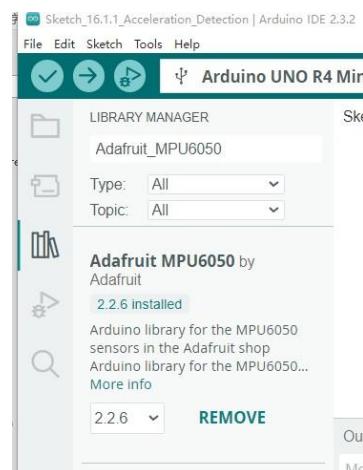
Click "Add .ZIP Library..." and then find Adafruit_MPU6050.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). These libraries make it easy to use MPU6050 module.



When these libraries are added, you can locate them in the libraries under Sketchbook location in the File-Preferences window. You can view the source code of these library files to understand their specific usage.



Or, you can search Adafruit_MPU6050 on library manager to install.



Now write sketch to communicate with the MPU6050 module and send the collected data to Serial Monitor

window.

```
1 // Reference the library to be used by Adafruit_MPU6050
2 #include <Adafruit_MPU6050.h>
3 #include <Adafruit_Sensor.h>
4 #include <Wire.h>
5
6 // Create an object for the MPU6050 sensor
7 Adafruit_MPU6050 mpu;
8
9 void setup() {
10 // Initialize the serial communication
11 Serial.begin(9600);
12 // Check if the MPU6050 sensor is detected
13 if (!mpu.begin()) {
14     Serial.println("Failed to find MPU6050 chip");
15     while (1) {
16         delay(10);
17     }
18 }
19 Serial.println("MPU6050 connection successful!");
20
21 // set accelerometer range to +-8G
22 mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
23 // set gyro range to +- 500 deg/s
24 mpu.setGyroRange(MPU6050_RANGE_500_DEG);
25 // set filter bandwidth to 21 Hz
26 mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
27 // Add a delay for stability
28 delay(100);
29 }
30
31 void loop() {
32 // Get new sensor events with the readings
33 sensors_event_t a, g, temp;
34 mpu.getEvent(&a, &g, &temp);
35
36 // Print out the acceleration readings in m/s^2
37 Serial.print("Acceleration: X:");
38 Serial.print(a.acceleration.x);
39 Serial.print(", Y:");
40 Serial.print(a.acceleration.y);
41 Serial.print(", Z:");
42 Serial.print(a.acceleration.z);
43 Serial.println(" (m/s^2)");
```

```

44 // Print out the rotation readings in rad/s
45 Serial.print("Rotation:      X:");
46 Serial.print(g.gyro.x);
47 Serial.print(", Y:");
48 Serial.print(g.gyro.y);
49 Serial.print(", Z:");
50 Serial.print(g.gyro.z);
51 Serial.println(" (rad/s)");
52
53
54 // Print out the temperature reading in degrees Celsius
55 Serial.print("Temperature:    ");
56 Serial.print(temp.temperature);
57 Serial.println(" (degC)");
58
59 // Add a blank line for readability
60 Serial.println("");
61
62 // Add a delay to avoid flooding the serial monitor
63 delay(250);
64 }

```

Include the necessary libraries.

```

2 #include <Adafruit_MPU6050.h>
3 #include <Adafruit_Sensor.h>
4 #include <Wire.h>

```

Initialize the sensor object. The Adafruit_MPU6050 library provides the Adafruit_MPU6050 class to operate the MPU6050. Before using it, you need to instantiate an object of the class.

```

7 Adafruit_MPU6050 mpu;

```

Initialize serial communication and call the function to initialize the MPU6050 sensor.

```

11 Serial.begin(9600);
12 // Check if the MPU6050 sensor is detected
13 if (!mpu.begin()) {
14     Serial.println("Failed to find MPU6050 chip");
15     while (1) {
16         delay(10);
17     }
18 }

```

After successfully connecting to the MPU6050, set the range for the accelerometer and gyroscope, and configure the filter bandwidth.

```

21 // set accelerometer range to +-8G
22 mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
23 // set gyro range to +- 500 deg/s
24 mpu.setGyroRange(MPU6050_RANGE_500_DEG);
25 // set filter bandwidth to 21 Hz

```

```

26   mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
27   // Add a delay for stability

```

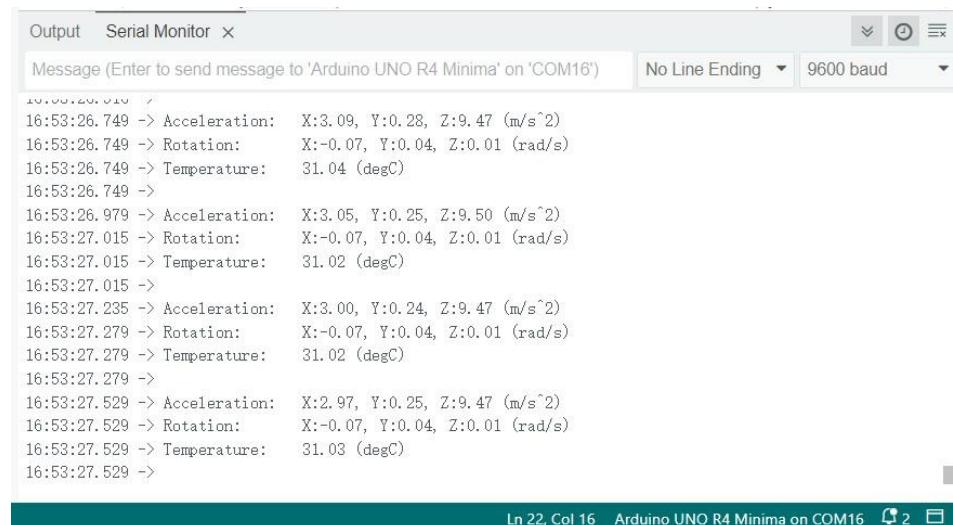
Read acceleration, gyroscope, and temperature data from the MPU6050 and print them to the serial monitor.

```

31 void loop() {
32   // Get new sensor events with the readings
33   sensors_event_t a, g, temp;
34   mpu.getEvent(&a, &g, &temp);
35
36   // Print out the acceleration readings in m/s^2
37   Serial.print("Acceleration: X:");
38   Serial.print(a.acceleration.x);
39   Serial.print(", Y:");
40   Serial.print(a.acceleration.y);
41   Serial.print(", Z:");
42   Serial.print(a.acceleration.z);
43   Serial.println(" (m/s^2)");
44
45   // Print out the rotation readings in rad/s
46   Serial.print("Rotation:      X:");
47   Serial.print(g.gyro.x);
48   Serial.print(", Y:");
49   Serial.print(g.gyro.y);
50   Serial.print(", Z:");
51   Serial.print(g.gyro.z);
52   Serial.println(" (rad/s)");
53
54   // Print out the temperature reading in degrees Celsius
55   Serial.print("Temperature:   ");
56   Serial.print(temp.temperature);
57   Serial.println(" (degC)");
58
59   // Add a blank line for readability
60   Serial.println("");
61
62   // Add a delay to avoid flooding the serial monitor
63   delay(250);
64 }

```

Verify and upload the code, open the Serial Monitor, then you can see the value of MPU6050 in original state and converted state, which is sent from control board. Rotate and move the MPU6050 module, and then you can see the change of values.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor x". The main area displays a continuous stream of data messages. At the top of the message list, there is a placeholder "Message (Enter to send message to 'Arduino UNO R4 Minima' on 'COM16')". Below this, the data is presented in a tabular format with three columns: timestamp, message type, and value. The data includes Acceleration (X, Y, Z), Rotation (X, Y, Z), and Temperature (in degrees Celsius). The baud rate is set to 9600, and the line ending is set to "No Line Ending". The status bar at the bottom indicates "Ln 22, Col 16" and "Arduino UNO R4 Minima on COM16".

Time	Message	Value
16:53:26.749	-> Acceleration:	X:3.09, Y:0.28, Z:9.47 (m/s^2)
16:53:26.749	-> Rotation:	X:-0.07, Y:0.04, Z:0.01 (rad/s)
16:53:26.749	-> Temperature:	31.04 (degC)
16:53:26.749	->	
16:53:26.979	-> Acceleration:	X:3.05, Y:0.25, Z:9.50 (m/s^2)
16:53:27.015	-> Rotation:	X:-0.07, Y:0.04, Z:0.01 (rad/s)
16:53:27.015	-> Temperature:	31.02 (degC)
16:53:27.015	->	
16:53:27.235	-> Acceleration:	X:3.00, Y:0.24, Z:9.47 (m/s^2)
16:53:27.279	-> Rotation:	X:-0.07, Y:0.04, Z:0.01 (rad/s)
16:53:27.279	-> Temperature:	31.02 (degC)
16:53:27.279	->	
16:53:27.529	-> Acceleration:	X:2.97, Y:0.25, Z:9.47 (m/s^2)
16:53:27.529	-> Rotation:	X:-0.07, Y:0.04, Z:0.01 (rad/s)
16:53:27.529	-> Temperature:	31.03 (degC)
16:53:27.529	->	

Data sent by this code may be too much for the users not familiar with acceleration. You can choose to upload sketch 17.1.2, which only send three direction acceleration values to the serial port. And it will be relatively easy to observe change of numbers, when you rotate and move this module,

Chapter 18 LED Matrix

In the previous chapter, we have learned how to use some modules and sensors and shows some information on the computer through serial port. Now let us learn some modules which can output images and text.

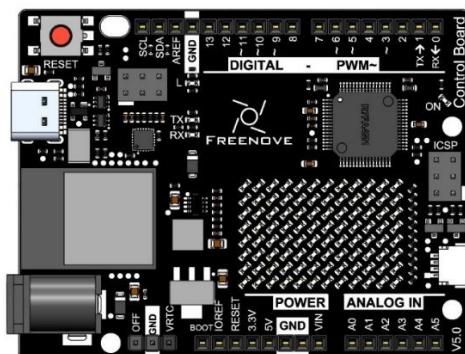
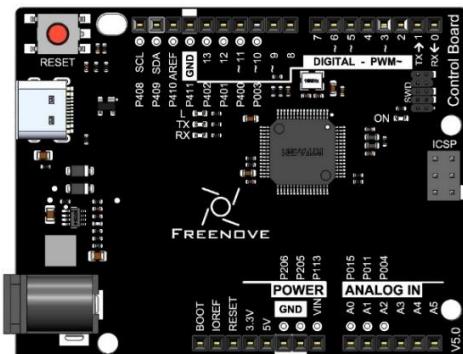
In this chapter, we will learn how to use the LED matrix to output characters and images.

Project 18.1 74HC595

Firstly, let us learn how to use the 74HC595 chip, which is very helpful for us to control the LED matrix.

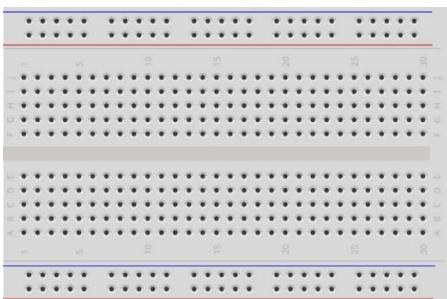
Component List

Control board x1

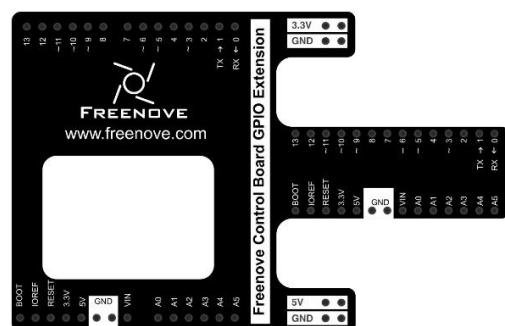


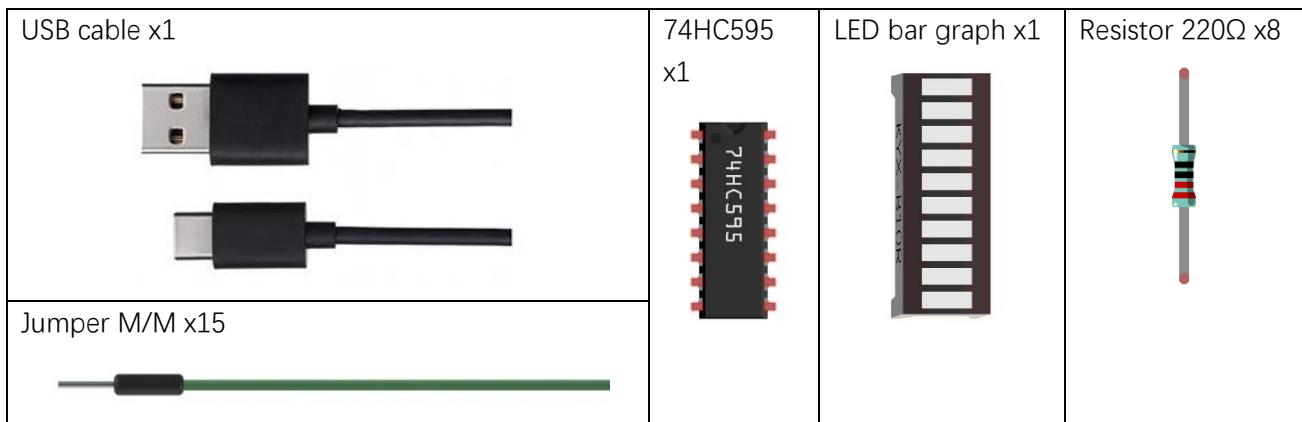
or

Breadboard x1



GPIO Extension Board x1





Code Knowledge

Hexadecimal

The conversion between binary and decimal system has been mentioned before. When you write the code, the number is decimal by default. Hexadecimal numbers need to add the 0x prefix in the code, such as 0x01. One Hexadecimal bit can present one number between 0-15. In order to facilitate writing, the numbers greater than 9 are written into the letter A-F (case-insensitive) such as 0x2A. The corresponding relationship is as follows:

Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Represent	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Conversion between hexadecimal and decimal system is similar to the conversion between hexadecimal and binary such as the sixteen digit 0x12:

Sequence	1	0
Number	1	2

When a hexadecimal number need to be converted to decimal number, first, the nth number of it need be multiplied by n power of 16, then sum all multiplicative results. Take 0x12 as an example:

$$1 * 16^1 + 2 * 16^0 = 18$$

When decimal number is converted to hexadecimal number, decimal number is divided by 16. Then we will get quotient and remainder, and quotient obtained will be continuously divided by 16 until quotient is zero. Arrange all remainders from right to left in a line. Then we complete the conversion. For example:

16	18	2	0
16	1	1	1
			0	

The result is of the conversion 0x12.

When you write code, sometimes it is convenient to use hexadecimal, especially involving bit operation, because 1 hexadecimal number can be expressed by 4 binary number ($2^4=16$). The corresponding relationship between 4 bit binary numbers and 1 hexadecimal number is shown as follows:

4 bit binary	0000	0001	0010	0011	0100	0101	0110	0111
1 figure of hexadecimal	0	1	2	3	4	5	6	7

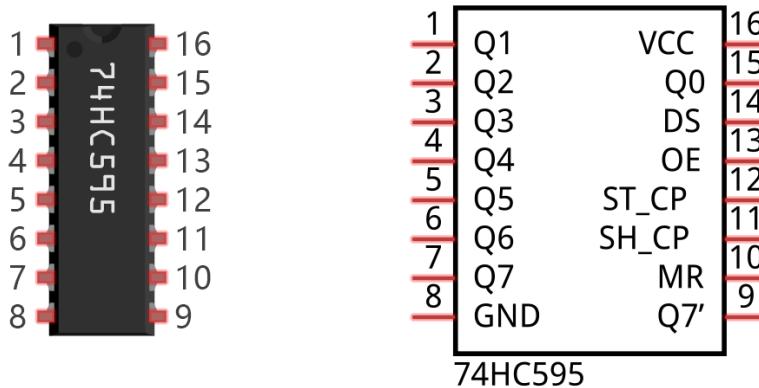
4 bit binary	1000	1001	1010	1011	1100	1101	1110	1111
1 figure of hexadecimal	8	9	A	B	C	D	E	F

For example, binary 00010010 is corresponding to hexadecimal 0x12.

Component 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 the control board. At least 3 ports on the RPI board are required to control the 8 ports of the 74HC595 chip.



The ports of 74HC595 are described as follows:

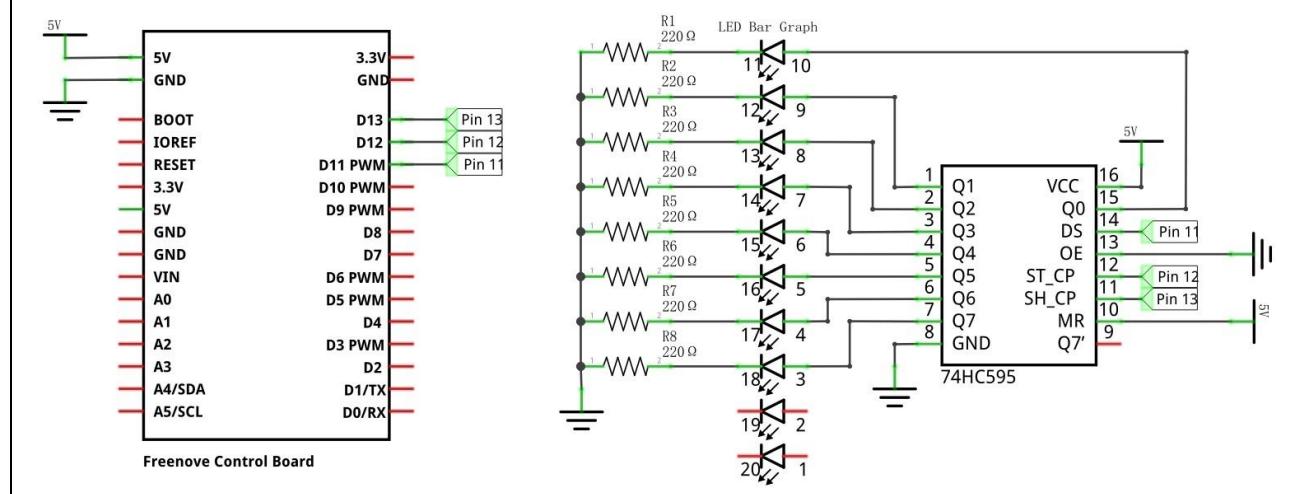
Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin 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.

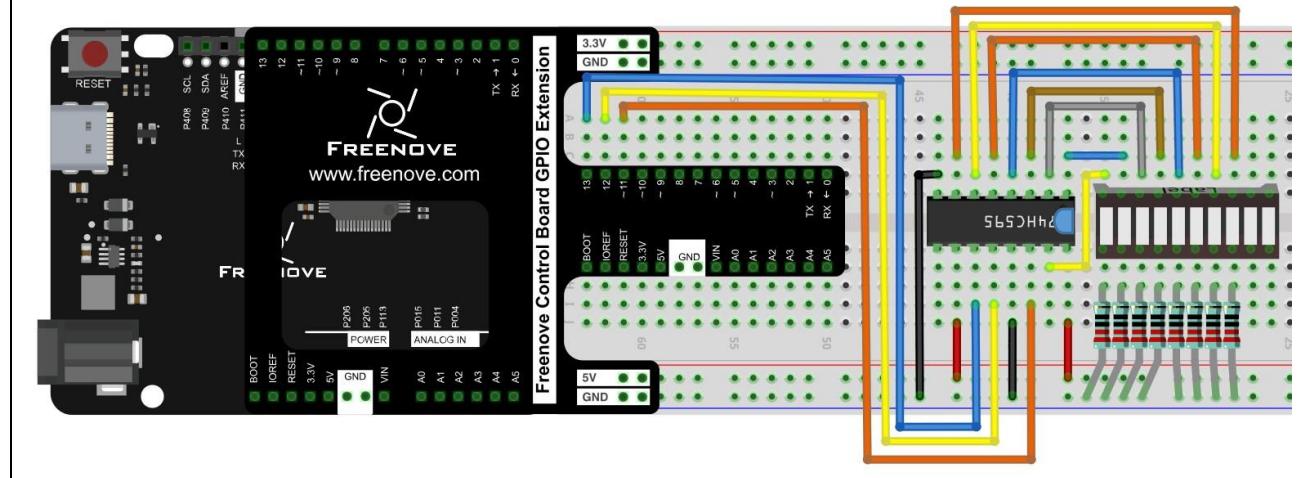
Circuit

Use pin 11, 12, 13 on the control board to control the 74HC595, and connect it to the 8 LEDs of LED bar graph.

Schematic diagram



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



Sketch

Sketch 18.1.1

Now write code to control the 8 LEDs of LED bar graph through 74HC595.

```

1 int latchPin = 12;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 13;           // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 11;            // Pin connected to DS of 74HC595 (Pin14)
4
5 void setup() {
6     // set pins to output
7     pinMode(latchPin, OUTPUT);
8     pinMode(clockPin, OUTPUT);
9     pinMode(dataPin, OUTPUT);

```

```

10    }
11
12 void loop() {
13     // Define a one-byte variable to use the 8 bits to represent the state of 8 LEDs of LED
14     // bar graph.
15     // This variable is assigned to 0x01, that is binary 00000001, which indicates only one
16     // LED light on.
17     byte x = 0x01;
18     for (int j = 0; j < 8; j++) {
19         // Output low level to latchPin
20         digitalWrite(latchPin, LOW);
21         // Send serial data to 74HC595
22         shiftOut(dataPin, clockPin, LSBFIRST, x);
23         // Output high level to latchPin, and 74HC595 will update the data to the parallel
24         // output port.
25         digitalWrite(latchPin, HIGH);
26         // make the variable move one bit to left once, then the bright LED move one step to
27         // the left once.
28         x <<= 1;
29         delay(100);
30     }
31 }
```

In the code, we configure three pins to control the 74HC595. And define a one-byte variable to control the state of 8 LEDs through the 8 bits of the variable. The LED lights 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.

15	byte x = 0x01;
----	----------------

In each loop, the x is sent to 74HC595. The sending process is as follows:

17	// Output low level to latchPin
18	digitalWrite(latchPin, LOW);
19	// Send serial data to 74HC595
20	shiftOut(dataPin, clockPin, LSBFIRST, x);
21	// Output high level to latchPin, and 74HC595 will update the data to the parallel
22	output port.
23	digitalWrite(latchPin, HIGH);

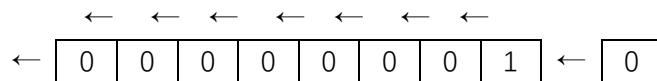
The x will be shift 1 bit to left in each cycle, which makes the bright LED of the 8 LEDs move one bit.

24	x <<= 1;
----	----------

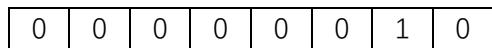
<< operator

"<<" 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;

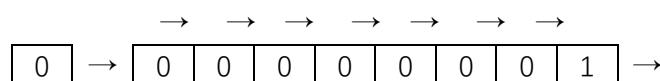


The result of x is 2 (binary 00000010).

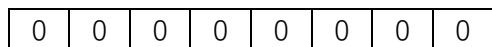


There is another similar operator " >> ". For example, shift binary 00000001 by 1 bit to right:

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

Verify and upload the code, and then you will see the LED bar graph with the effect of flowing water.

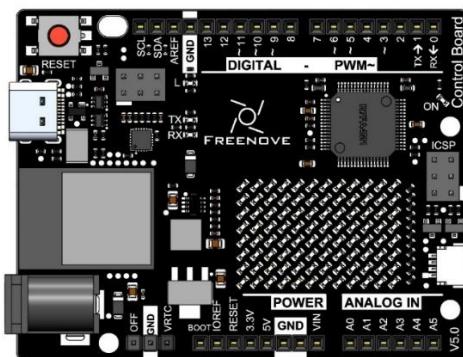
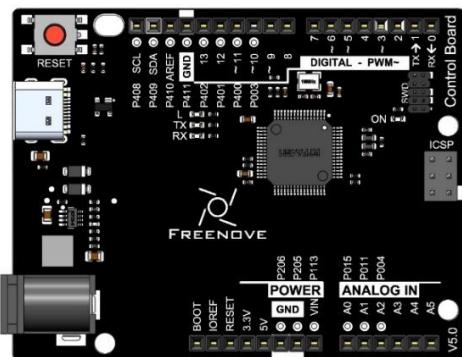


Project 18.2 LED Matrix

In the previous section, we have used 74HC595 to control 8 LEDs of the LED bar graph. Now let's use 74HC595 to control LED matrix.

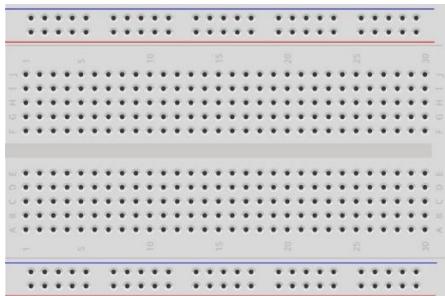
Component List

Control board x1

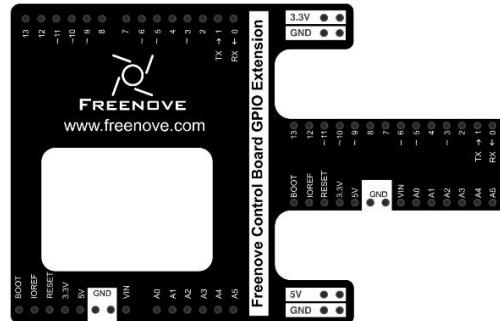


or

Breadboard x1



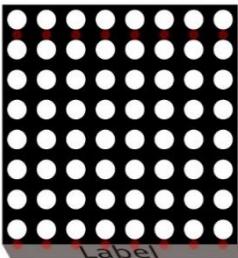
GPIO Extension Board x1



USB cable x1



LED matrix x1



74HC595 x1



R220 x8



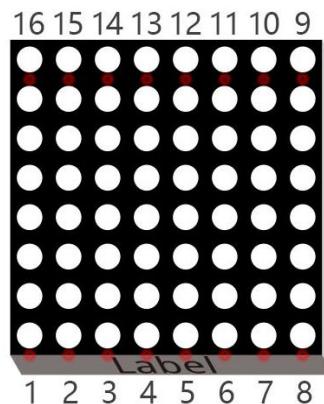
Jumper M/M x31



Component Knowledge

LED matrix

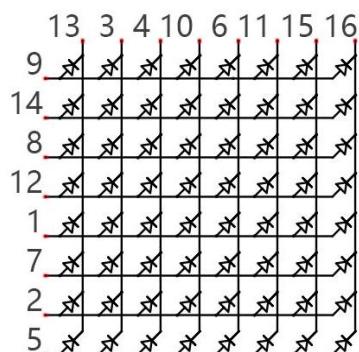
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



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.

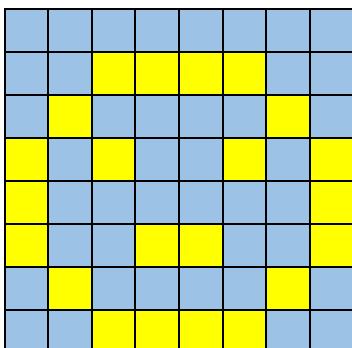
Connection mode of common anode



Connection mode of common cathode



Here is how a Common Anode LED Matrix works. First, choose 16 ports on RPI board 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

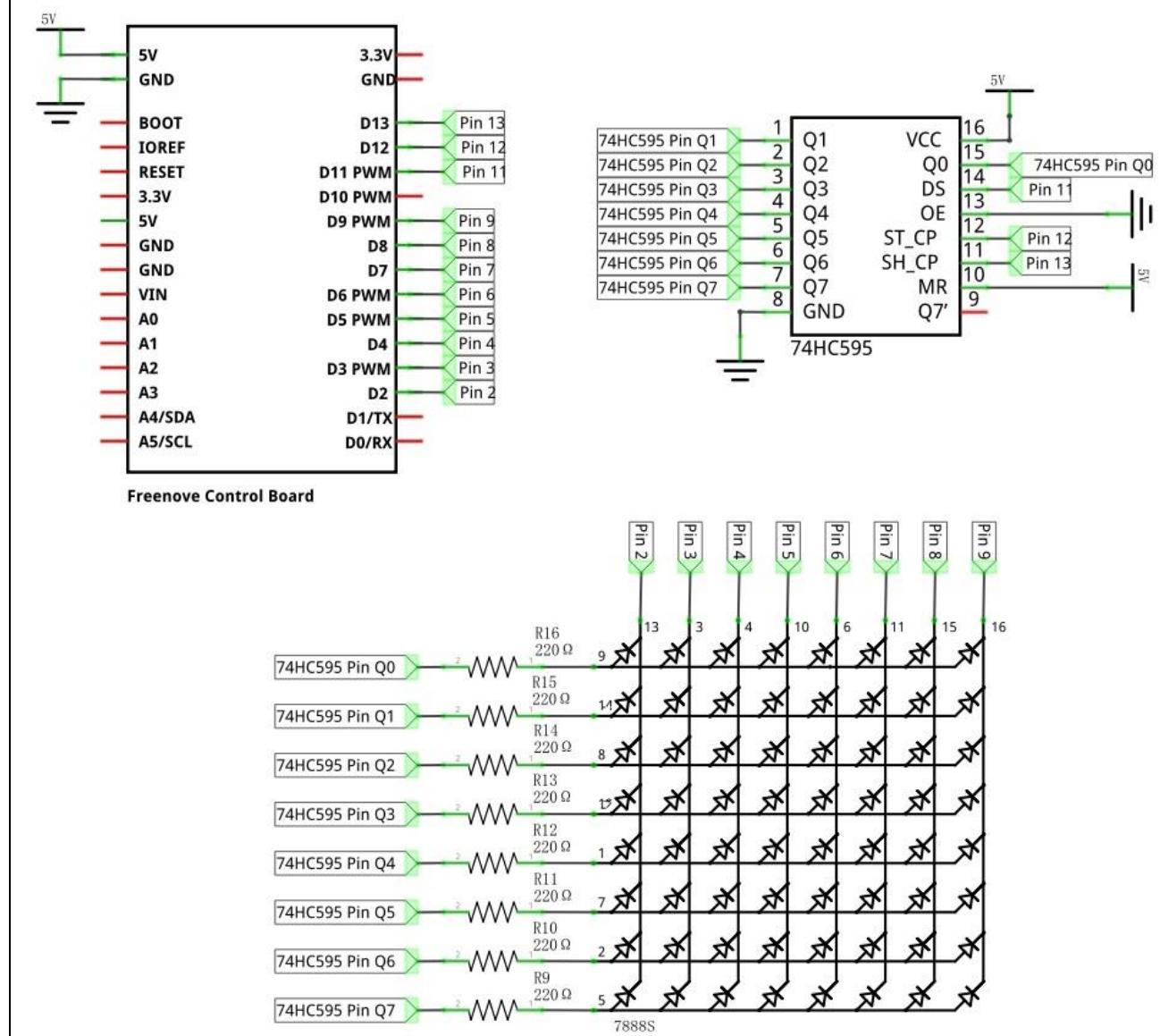
Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit.

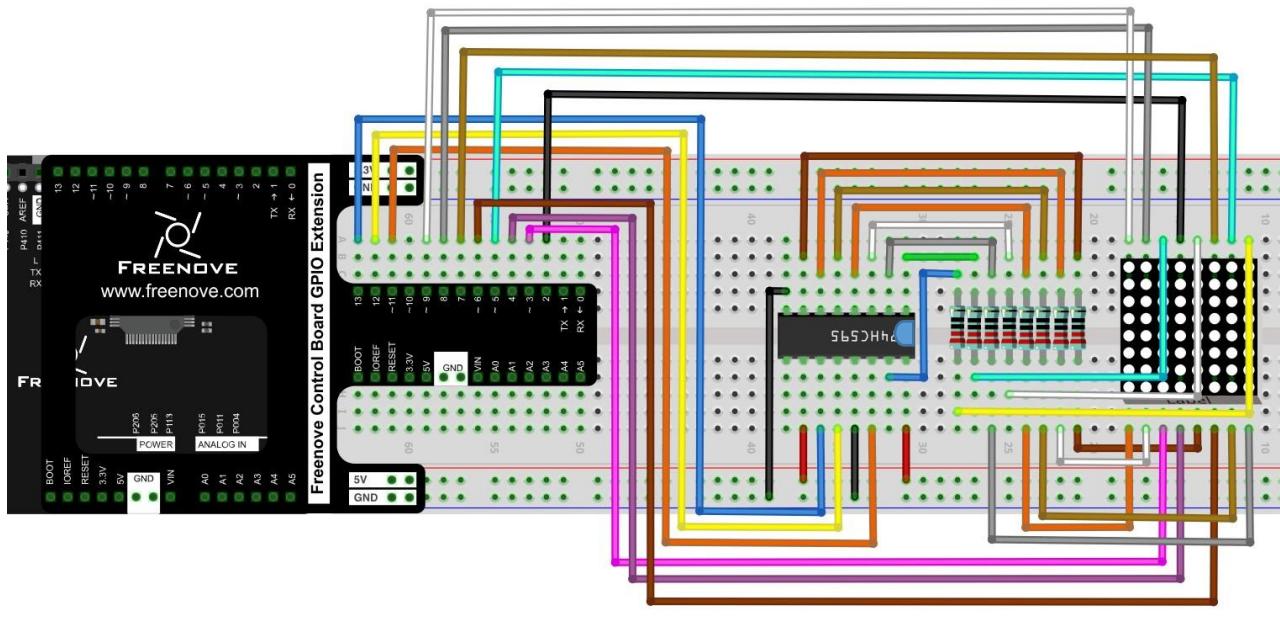
Circuit

Use pin 11, 12, 13 on control board to control the 74HC595. And connect 74HC595 to the 8 anode pins of LED Matrix, in the meanwhile, connect 8 digital port on control board to the 8 cathode pins of LED Matrix.

Schematic diagram



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



Sketch

Sketch 17.2.1

Now write the code to drive LED dot matrix to display static and dynamic images, in fact, the dynamic image is formed by continuous static image.

```

1 int latchPin = 12;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 13;           // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 11;            // Pin connected to DS of 74HC595 (Pin14)
4 int LEDPin[] = {2, 3, 4, 5, 6, 7, 8, 9};    // column pin (cathode) of LED Matrix
5
6 // Define the pattern data for a smiling face
7 const int smilingFace[] = {
8     0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C
9 };
10 // Define the data of numbers and letters, and save them in flash area
11 const int data[] PROGMEM = {
12     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
13     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
14     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
15     0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
16     0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
17     0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
18     0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"

```

```
19   0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
20   0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
21   0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
22   0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
23   0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
24   0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
25   0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
26   0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
27   0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
28   0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00 // "F"
29 }
30
31 void setup() {
32   // set pins to output
33   pinMode(latchPin, OUTPUT);
34   pinMode(clockPin, OUTPUT);
35   pinMode(dataPin, OUTPUT);
36   for (int i = 0; i < 8; i++) {
37     pinMode(LEDPin[i], OUTPUT);
38   }
39 }
40
41 void loop() {
42   // Define a one-byte variable (8 bits) which is used to represent the selected state of
43   // 8 columns.
44   int cols;
45   // Display the static smiling pattern
46   for (int j = 0; j < 500; j++) { // repeat 500 times
47     cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the
48     // first column is selected.
49     for (int i = 0; i < 8; i++) { // display 8 column data by scanning
50       matrixColsVal(cols); // select this column
51       matrixRowsVal(smilingFace[i]); // display the data in this column
52       delay(1); // display them for a period of time
53       matrixRowsVal(0x00); // clear the data of this column
54       cols <= 1; // shift "cols" 1 bit left to select the next column
55     }
56   // Display the dynamic patterns of numbers and letters
57   for (int i = 0; i < 128; i++) { // "space, 0-9, A-F"16 letters , each letter hold 8
58     // columns, total 136 columns. Firstly, display space , then we need shift 128 times (136-8)
59     for (int k = 0; k < 10; k++) { // repeat image of each frame 10 times.
60       cols = 0x01; // Assign binary 00000001. Means the first column is selected.
61       for (int j = i; j < 8 + i; j++) { // display image of each frame
62 }
```

```

60     matrixColsVal(cols);           // select this column
61     matrixRowsVal(pgm_read_word_near(data + j)); // display the data in this column
62     delay(1);                     // display them for a period of time
63     matrixRowsVal(0x00);          // close the data of this column
64     cols <= 1;                   // shift "cols" 1 bit left to select the next column
65   }
66 }
67 }
68 }
69
70 void matrixRowsVal(int value) {
71   // make latchPin output low level
72   digitalWrite(latchPin, LOW);
73   // Send serial data to 74HC595
74   shiftOut(dataPin, clockPin, LSBFIRST, value);
75   // make latchPin output high level, then 74HC595 will update the data to parallel
    output
76   digitalWrite(latchPin, HIGH);
77 }
78
79 void matrixColsVal(byte value) {
80   byte cols = 0x01;
81   // Output the column data to the corresponding port.
82   for (int i = 0; i < 8; i++) {
83     digitalWrite(LEDPin[i], ((value & cols) == cols) ? LOW : HIGH);
84     cols <= 1;
85   }
86 }
```

In the code, use an array to define the column pins of LED Matrix.

4	<code>int LEDPin[] = {2, 3, 4, 5, 6, 7, 8, 9}; // Column pins (cathode) of LED Matrix</code>
---	--

Use another array to define 8 column data of a smiling face.

7	<code>const int smilingFace[] = {</code>
8	<code> 0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C</code>
9	<code>};</code>

Use another array to define some numbers and letters, and every eight elements of the array represent a dot matrix pattern data of a number or a letter.

11	<code>const int data[] PROGMEM = {</code>
12	<code> 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // "</code>
13	<code> 0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"</code>
14	<code> 0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"</code>
15	<code> 0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"</code>
16	<code> 0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"</code>
17	<code> 0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"</code>

```

18   0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
19   0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
20   0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
21   0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
22   0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
23   0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
24   0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
25   0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
26   0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
27   0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
28   0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00 // "F"
29 } ;

```

PROGMEM keyword

Microprocessors generally have two storage areas, namely ROM and RAM. ROM is used to store code. And these stored data will not change with the execution of code until the code are uploaded. RAM is used to store data, for example, the variables we defined are stored here. The stored data will change in real time with the execution of the code. Generally, capacity of RAM is small. So we can use PROGMEM keyword to save the data that don't change in ROM.

Define two functions, one of them uses control board port to select the column.

```

79 void matrixColsVal(byte value) {
80     byte cols = 0x01;
81     // output the column data to the corresponding port.
82     for (int i = 0; i < 8; i++) {
83         digitalWrite(LEDPin[i], ((value & cols) == cols) ? LOW : HIGH);
84         cols <= 1;
85     }
86 }

```

Another one uses 74HC595 to write data of the row.

```

70 void matrixRowsVal(int value) {
71     // ouput low level to latchPin
72     digitalWrite(latchPin, LOW);
73     // send serial data to 74HC595
74     shiftOut(dataPin, clockPin, LSBFIRST, value);
75     // output high level to latchPin, then 74HC595 will update the data to parallel output
76     // port
77     digitalWrite(latchPin, HIGH);
}

```

? : operator

"? :" operator is similar to conditional statements. When the expression in front of "?" is tenable, the statement in front of ":" will be executed. When the expression is not tenable, the statement behind ":" will be executed. For example:

```
int a = (1 > 0) ? 2: 3;
```

Because $1 > 0$ is tenable, so "a" will be assigned to 2.

Bitwise logical operation

There are many bitwise logical operators such as and ($\&$), or ($|$), xor (\wedge), negate (\sim). The result of exclusive or (\wedge) is true only when two corresponding bit is not equal.

$\&$, $|$ and \wedge is used to operate the corresponding bit of two numbers. Such as:

```
byte a = 1 & 2;
```

"a" will be assigned to 0. The calculation procedure is as follows:

$$\begin{array}{r} 1(00000001) \\ \& 2(00000010) \\ \hline 0(00000000) \end{array}$$

$$\begin{array}{r} 1(00000001) \\ \& 2(00000010) \\ \hline 0(00000000) \end{array}$$

Negate (\sim) is used to negate a number, for example:

```
byte a = ~15;
```

"a" will be assigned to 240. The calculation procedure is as follows:

$$\begin{array}{r} \sim 15(00001111) \\ \hline 240(11110000) \end{array}$$

$$\begin{array}{r} \sim 15(00001111) \\ \hline 240(11110000) \end{array}$$

In the loop () function, firstly, show the static smile pattern. Select one of the 8 columns circularly in turn to display each the result. Repeat 500 times the process, then we can see a static smile pattern.

```
45  for (int j = 0; j < 500; j++) { //repeat 10 times
46      cols = 0x01; // variable cols are assigned to 0x01(binary 00000001), then the first
47      // column is selected
48      for (int i = 0; i < 8; i++) { // display the data in 8 column data by scanning
49          matrixColsVal(cols); // select this column
50          matrixRowsVal(smilingFace[i]); // display data in this column
51          delay(1); // display the data for a period of time
52          matrixRowsVal(0x00); // close the data of this column
53          cols <= 1; // shift "cols" 1 bit left to select the next column
54      }
}
```

Then display the dynamic pattern of the numbers and letters. We have defined space, 0-9, A-F, total of 16 characters (136 columns) in an array, among which 8 adjacent rows of data form one frame. Shift one column once. There are for 128 frames of image from the first frame (1-8) to the last frame (128-136 column). Each frame image is displayed 10 times, then display the next frame. Repeat the process above, then we can see the pattern of scrolling numbers and letters.

```
56  for (int i = 0; i < 128; i++) { // "space, 0-9, A-F"16 letters , each letter hold 8
57  // columns, total 136 columns. Firstly, display space ,then we need shift 128 times (136-8)
58  for (int k = 0; k < 10; k++) { // repeat image of each frame 10 times.
59  cols = 0x01; // Assign binary 00000001. Means the first column is selected.
    for (int j = i; j < 8 + i; j++) { // display image of each frame
```

```
60     matrixColsVal(cols);           // select this column
61     matrixRowsVal(pgm_read_word_near(data + j)); // display the data in this column
62     delay(1);                      // display them for a period of time
63     matrixRowsVal(0x00);           // close the data of this column
64     cols <<= 1;                  // shift "cols" 1 bit left to select the next column
65   }
66 }
67 }
68 }
```

Verify and upload the code, then LED Matrix begins to display the static smile pattern. A few seconds later, LED Matrix will display the scrolling number 0-9 and the letter A-F.

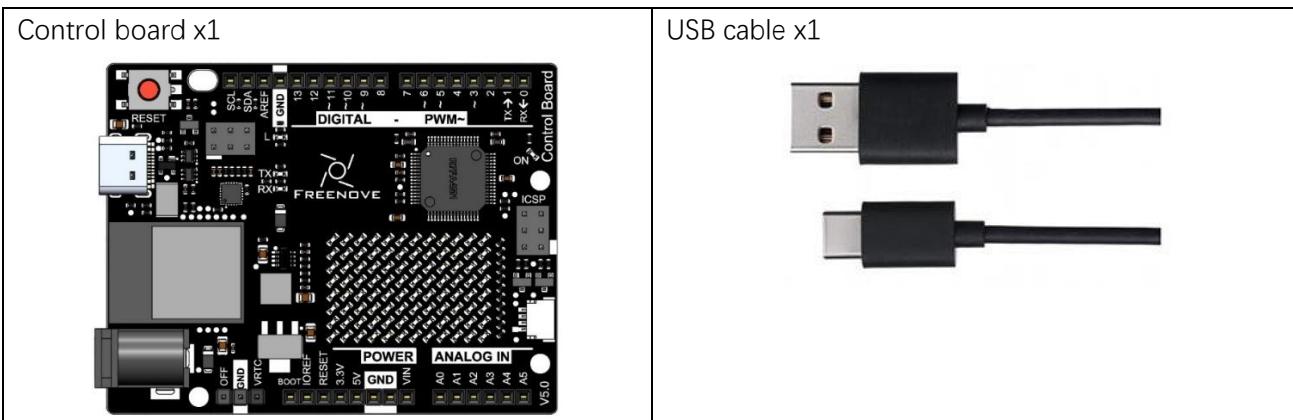
Chapter 19 Onboard LED Matrix (WiFi Board)

In this chapter, we utilize the LED matrix on the control board to display interesting patterns. The control board features a built-in 12x8 LED matrix that can be programmed to display graphics, animations, serve as an interface, and even play games.

Project 19.1 LED Matrix

In this section, we will use the LED matrix to display static graphics.

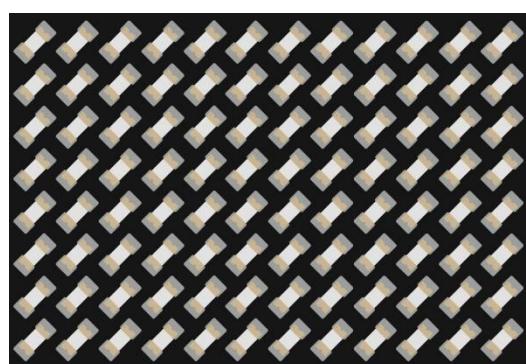
Component List



Component Knowledge

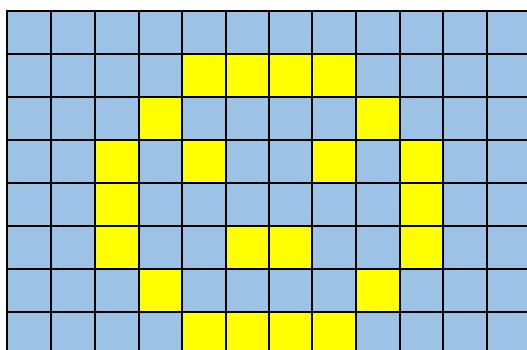
LED matrix

The LED matrix on the control board is a rectangular display module composed of a uniform grid of LEDs. Below is an 8x12 monochrome LED matrix, which includes 96 LEDs (8 rows by 12 columns).



You can call the LED matrix library to display any content you wish. With the LED matrix library, you can quickly display any graphics. For example, to display a smiling face, simply assign a value of 1 to the positions of the

LEDs that need to be lit. Conversely, assigning a value of 0 will turn off the corresponding LEDs, allowing you to set up a variety of interesting patterns.



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

To control the onboard 12x8 LED matrix, you will need a memory space of at least 96 bits in size, as shown below:

```

1 byte frame[8][12] = {
2 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
3 { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0 },
4 { 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 },
5 { 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 },
6 { 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0 },
7 { 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0 },
8 { 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 },
9 { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0 }};
```

The setup method mentioned above is easy to understand because you can visualize the image in the array pattern, and it is straightforward to edit while running. The elements in the array above form a smiley face, which is the image you see on the screen.

If you need to locate a single pixel, select its address and change the value. Remember to start counting from 0. Thus, the following lines will address the third pixel from the left and the second pixel from the top, and then activate it:

```

1 frame[2][1] = 1;
2
3 matrix.renderBitmap(frame, 8, 12);
```

However, this method consumes more memory than required. Even though each LED only needs one bit to store its state, 8 bits (one byte) are used. There is a more memory-efficient approach for storing frames that employs an array of 32-bit integers.

Unsigned long variables store 32 bits, and dividing 96 by 32 gives us 3, indicating that 96 LEDs can be split into three segments. Thus, an array of unsigned long integers is an efficient way to store all the bits necessary for the LED matrix.

The following is the data of converting the binary values of each row of the LED matrix to hexadecimal from left to right:

Column	Binary	Hexadecimal
1	0000 0000 0000	0x000
2	0000 1111 0000	0x0F0
3	0001 0000 1000	0x108
4	0010 1001 0100	0x294
5	0010 0000 0100	0x204
6	0010 0110 0100	0x264
7	0001 0000 1000	0x108
8	0000 1111 0000	0x0F0

The above 8 sets of data are converted into 3 groups as follows:

1	0000 0000 0000 0000 1111 0000 0001 0000
2	
3	1000 0010 1001 0100 0010 1001 0100 0010
4	
5	0110 0100 0001 0000 1000 0000 1111 0000

1	0x0000F010
2	
3	0x82942042
4	
5	0x641080F0

From this we can conclude that the 32-bit integer array of a smiley face is as follows.

1	unsigned long frame[] = {
2	0x0000F010,
3	0x82942042,
4	0x641080F0
5	};

Uploading the above data to the control board will display the corresponding pattern. Additionally, if you have multiple different frames, you can load and display them as below:

1	const uint32_t happy[] = {
2	0x19819,
3	0x80000001,
4	0x81f8000
5	};
6	const uint32_t heart[] = {
7	0x3184a444,
8	0x44042081,
9	0x100a0040
10	};
11	matrix.loadFrame(happy);
12	delay(500);
13	

```
14     matrix.loadFrame(heart);
15     delay(500);
```

Call the display function and you can see the LED display the above contents.

Sketch

Sketch 19.1.1

Upload the sketch to the control board and you should see the entire matrix lights up, turn off after a second, and then display the static expression resembling a smiley face.

The following is the program code:

```
1 #include "Arduino_LED_Matrix.h" // Include the LED_Matrix library
2
3 ArduinoLEDMatrix matrix; // Create an instance of the ArduinoLEDMatrix class
4
5 byte frame[8][12] = {
6     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
7     { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0 },
8     { 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 },
9     { 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 },
10    { 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
11    { 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0 },
12    { 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 },
13    { 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0 }
14 };
151
16 unsigned long frame2[] = {
17     0x0000F010,
18     0x82942042,
19     0x641080F0
20 };
21
22 const uint32_t fullOn[] = {
23     0xffffffff,
24     0xffffffff,
25     0xffffffff
26 };
27 const uint32_t fullOff[] = {
28     0x00000000,
29     0x00000000,
30     0x00000000
```

```

31  };
32
33 void setup() {
34   matrix.begin(); // Initialize the LED matrix
35   matrix.loadFrame(fullOn);
36   delay(250);
37   matrix.loadFrame(fullOff);
38   delay(250);
39 }
40
41 void loop() {
42   //matrix.loadFrame(frame2);
43   matrix.renderBitmap(frame, 8, 12);
44   delay(250);
45 }
```

First, include the library “Arduino_LED_Matrix.h” at the beginning of the sketch, as shown below.

```
1 #include "Arduino_LED_Matrix.h" // Include the LED_Matrix library
```

Create an object for the LED matrix in the sketch.

```
3 ArduinoLEDMatrix matrix; // Create an instance of the ArduinoLEDMatrix class
```

Activate the LED matrix by adding the line “matrix.begin();” under void setup() as shown below.

```
34 matrix.begin(); // Initialize the LED matrix
```

After the program initializes, it first lights up the entire LED matrix and then turns off the LED matrix, and subsequently continuously displays the smiling face graphic.

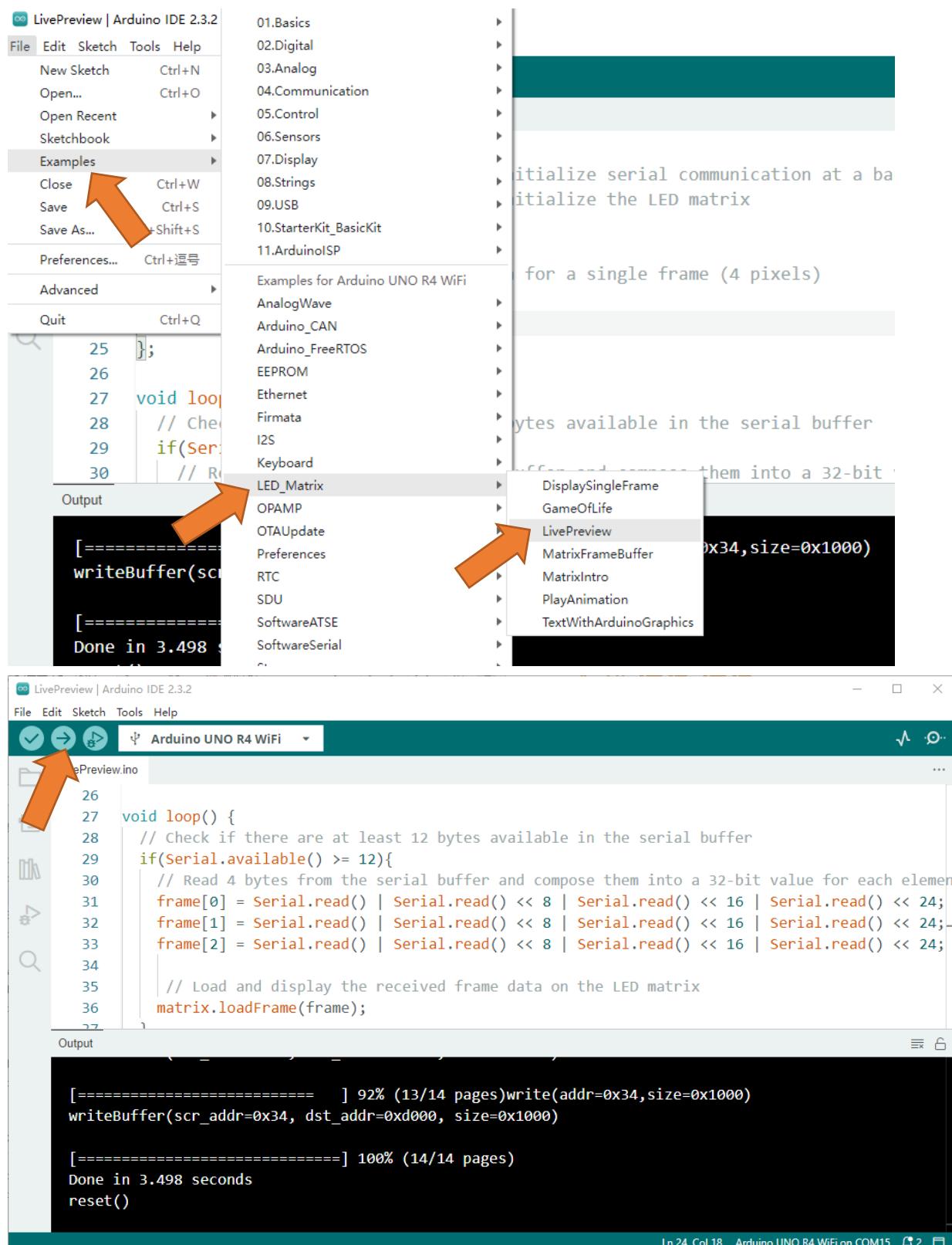
```

34 matrix.begin(); // Initialize the LED matrix
35 matrix.loadFrame(fullOn);
36 delay(250);
37 matrix.loadFrame(fullOff);
38 delay(250);
...
43 matrix.renderBitmap(frame, 8, 12);
44 delay(250);
```

Arduino has also created an online tool called LED Matrix Editor to simplify frame creation. Each frame can be edited graphically and the duration specified. Once this process is complete, simply name the file and download it for use in your project.

To customize more graphics, you can design via this link: [LED matrix editor \(Arduino cc\)](#)

When using the above tools to customize the graphics, the following examples should be uploaded to the control board first. After uploading, the corresponding device can be connected to the real-time transmission of the graphics to the control board for display.

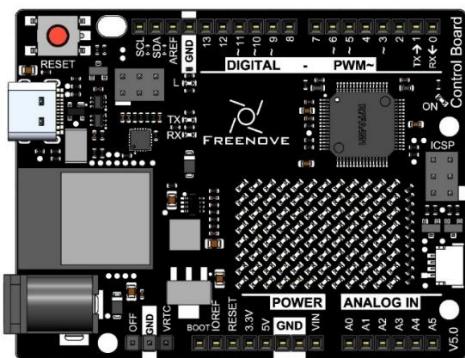


Project 19.2 LED Matrix

In this segment, we'll harness the onboard LED matrix to showcase dynamic visuals by scrolling the message "**Hello World!**" across the matrix.

Component List

Control board x1



USB cable x1



Sketch

Sketch 19.2.1

When you upload the sketch to the control board, you will observe the onboard LED matrix displaying dynamic scenes. The text "**Hello World!**" will be presented in a scrolling manner.

The following is the program code:

```

1 #include "ArduinoGraphics.h"
2 #include "Arduino_LED_Matrix.h"
3
4 ArduinoLEDMatrix matrix;
5
6 void setup() {
7   Serial.begin(115200);
8   matrix.begin();
9 }
10
11 void loop() {
12
13   // Make it scroll!
14   matrix.beginDraw();
15 }
```

```
16 matrix.stroke(0xFFFFFFFF);
17 matrix.textScrollSpeed(50);
18
19 // add the text
20 const char text[] = "Hello World!";
21 matrix.setFont(Font_5x7);
22 matrix.beginText(0, 1, 0xFFFFFFFF);
23 matrix.println(text);
24 matrix.endText(SCROLL_LEFT);
25
26 matrix.endDraw();
27 }
```

First, include the library “Arduino_LED_Matrix.h” at the beginning of the sketch, as shown below.

```
1 #include "ArduinoGraphics.h"
2 #include "Arduino_LED_Matrix.h"
```

Create an object for the LED matrix in the sketch.

```
4 ArduinoLEDMatrix matrix; // Create an instance of the ArduinoLEDMatrix class
```

Activate the LED matrix by adding the line “matrix.begin();” under void setup() as shown below.

```
8 matrix.begin(); // Initialize the LED matrix
```

In the main function, print “Hello World!” on the LED matrix.

```
13 // Make it scroll!
14 matrix.beginDraw();
15
16 matrix.stroke(0xFFFFFFFF);
17 matrix.textScrollSpeed(50);
18
19 // add the text
20 const char text[] = "Hello World!";
21 matrix.setFont(Font_5x7);
22 matrix.beginText(0, 1, 0xFFFFFFFF);
23 matrix.println(text);
24 matrix.endText(SCROLL_LEFT);
25
26 matrix.endDraw();
```

For more examples please refer to:

[Using the Arduino UNO R4 WiFi LED Matrix | Arduino Documentation](#)

Project 19.3 Play the game with LED matrix

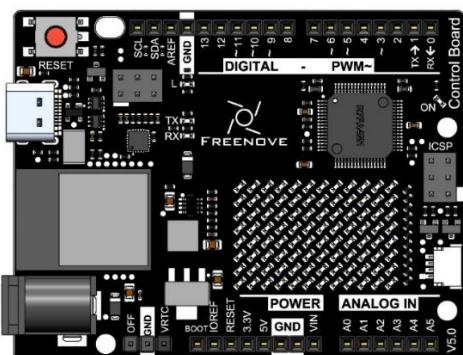
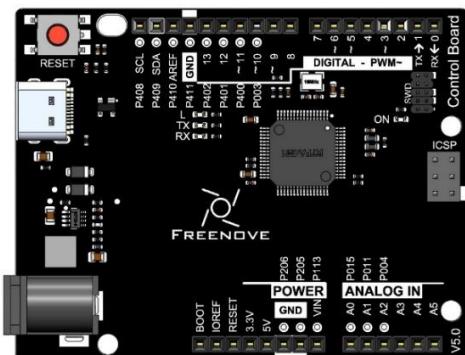
In this section, we play the game using the LED matrix.

Project 19.3.1 LED Matrix Bounce Game

Play a bounce game with an LED matrix.

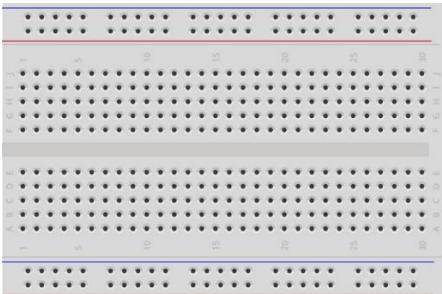
Component List

Control board x1

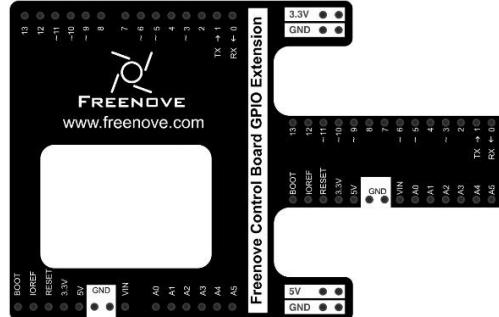


or

Breadboard x1



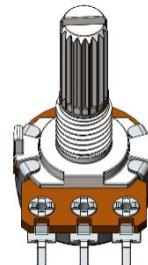
GPIO Extension Board x1



USB cable x1



Rotary potentiometer x1



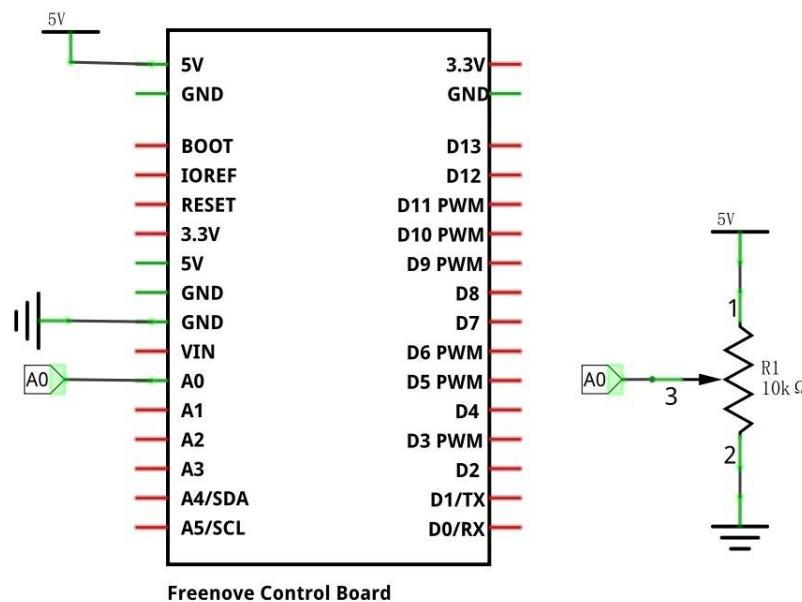
Jumper M/M x3



Circuit

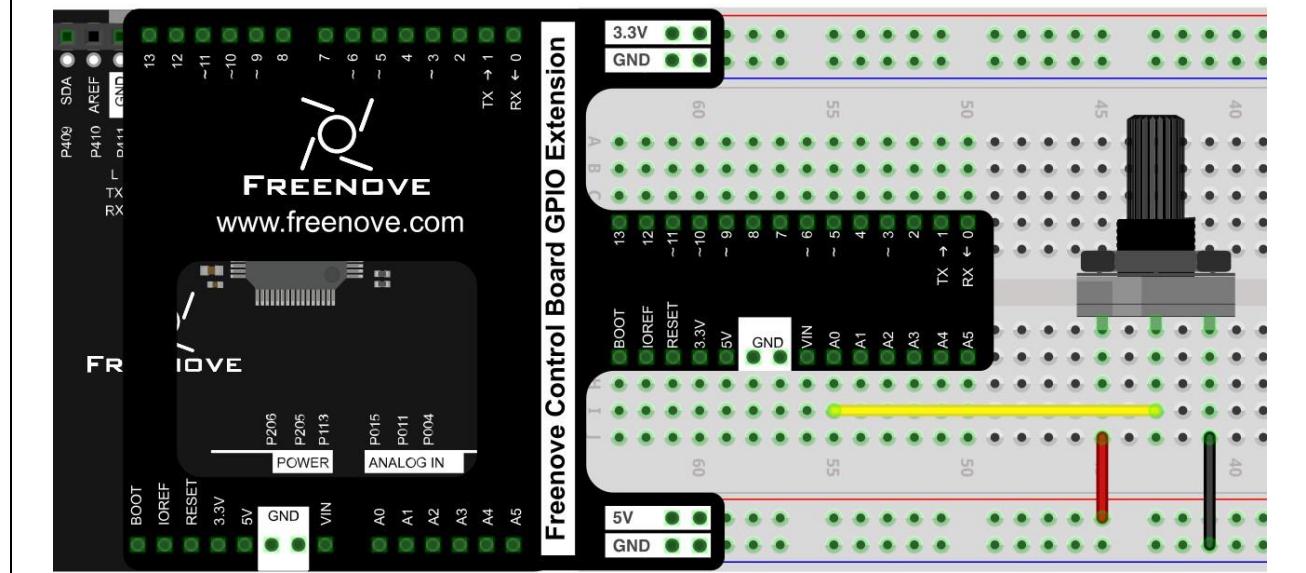
The voltage of the rotary potentiometer is detected with the A0 pin on the control board to control the movement of the bat.

Schematic diagram



Freenove Control Board

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



Sketch

Sketch 19.3.1

This is a simple ball bounce game, using a potentiometer to control the movement of the bat, when your bat fails to catch the ball, the game will end and the led matrix will show "OUT", and then start the game again.

The following is the program code:

```
1 #include "Arduino_LED_Matrix.h"
2
3 ArduinoLEDMatrix matrix;
4
5 void setup() {
6     Serial.begin(115200);
7     matrix.begin();
8 }
9
10 uint8_t frame[8][12] = {
11     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
12     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
13     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
14     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
15     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
16     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
17     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
18     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
19 };
20
21 uint8_t game_over[8][12] = {
22     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
23     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
24     { 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1 },
25     { 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0 },
26     { 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0 },
27     { 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0 },
28     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
29     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
30 };
31
32 int y = 4;           // starting y position
33 int x = 4;           // starting x position
34 int b = 1;           // starting y direction
35 int a = 1;           // starting x direction
```

```
36 int anticipate = 5; // variable used to anticipate whether ball hits bat
37 int speed = 300; // initial delay which decreases each time ball hit
38
39
40 void loop() {
41     int batcenter = (analogRead(A0) / 204);
42     frame[batcenter][11] = 1;
43     frame[batcenter + 1][11] = 1;
44     frame[batcenter + 2][11] = 1;
45     frame[y][x] = 1;
46     matrix.renderBitmap(frame, 8, 12);
47     delay(speed);
48     frame[batcenter][11] = 0;
49     frame[batcenter + 1][11] = 0;
50     frame[batcenter + 2][11] = 0;
51     frame[y][x] = 0;
52     matrix.renderBitmap(frame, 8, 12);
53     delay(1);
54
55     if (y == 7) {
56         b = -b;
57     }
58     if (y == 0) {
59         b = -b;
60     }
61     if (x == 10) { // when ball reaches line in front of batcenter, checks if batcenter hit
62         anticipate = (y + b);
63         speed = (speed - 20); // reduces delay each time batcenter hit
64         if ((anticipate == batcenter) || (anticipate == (batcenter + 1)) || (anticipate == (batcenter + 2))) {
65             a = -a;
66         } else { // game over grid
67             matrix.renderBitmap(game_over, 8, 12);
68             delay(2000);
69             x = 4;
70             y = 4;
71             a = -1;
72             b = -1;
73             speed = 300;
74         }
75     }
76 }
77 if (x == 0) {
78     a = -a;
79 }
```

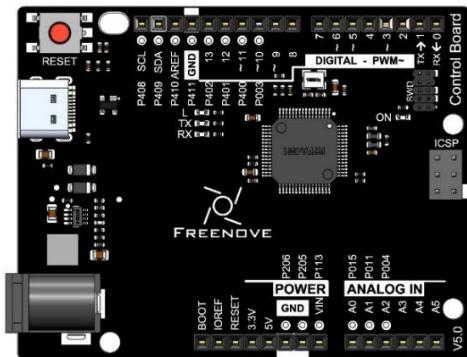
```
80     y = (y + b); // adjusts ball position  
81     x = (x + a);  
82 }  
83
```

Project 19.3.2 LED Matrix Snake Game

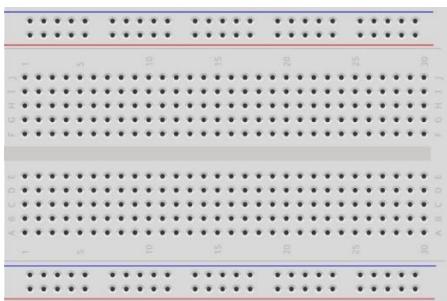
Play a snake game with an LED matrix.

Component List

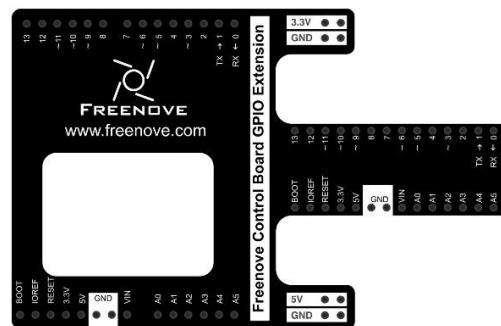
Control board x1



Breadboard x1



GPIO Extension Board x1



USB cable x1



Joystick x1



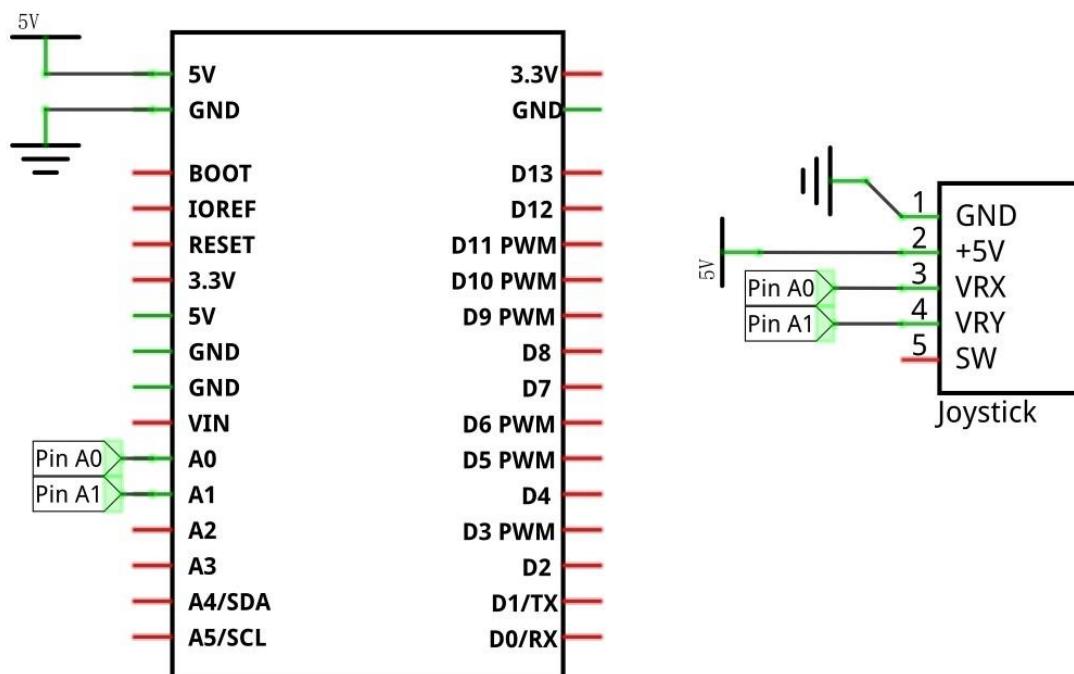
Jumper F/M x5



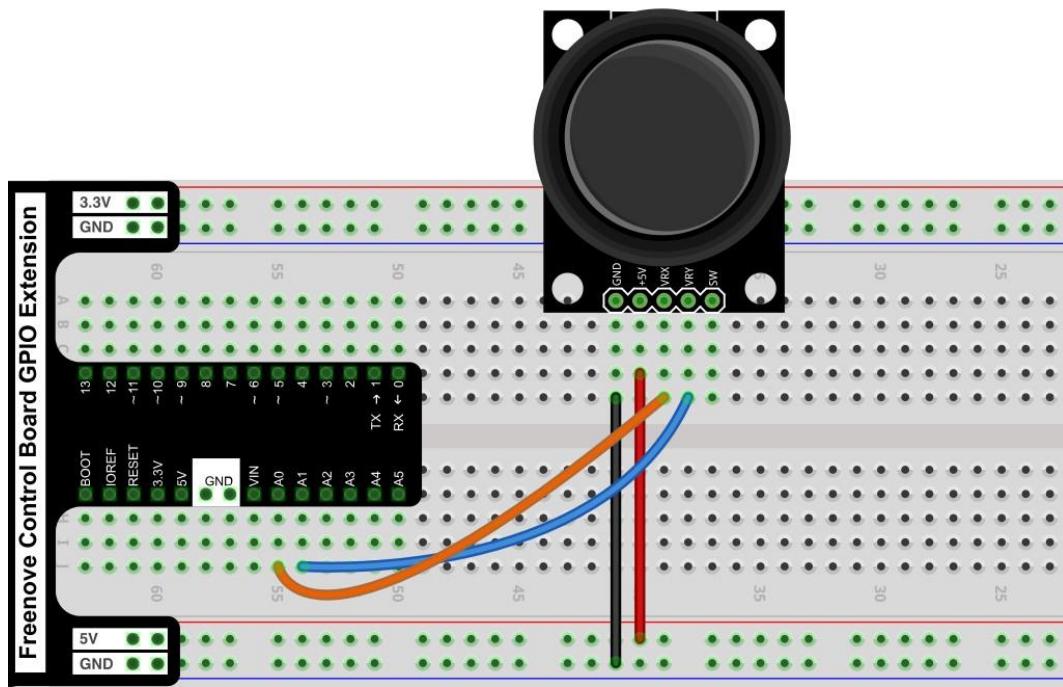
Circuit

Use pin A0 and pin A1 on control board to detect the voltage value of two rotary potentiometers inside Joystick.

Schematic diagram



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



Sketch

Sketch 19.3.2

This is a simple snake game, using a Joystick to control the direction of movement of the snake head, when the snake head contacts the body of the snake, the game will end, the led matrix will show "OUT", and then start the game again.

The following is the program code:

```
1 #include "Arduino_LED_Matrix.h"
2
3 ArduinoLEDMatrix matrix;
4
5 void setup() {
6     matrix.begin();
7 }
8
9 #define LEFT 0
10 #define RIGHT 1
11 #define UP 2
12 #define DOWN 3
13
14 // Global variables
15 byte frame[8][12];
16 byte dir = RIGHT;
17 byte snake_len = 2;
18 byte col_idx[96];
19 byte row_idx[96];
20 byte food_col = random(2, 12);
21 byte food_row = random(1, 8);
22 byte speed = 500;
23 byte food_hit = 1;
24 byte blank_frame[8][12];
25
26 uint8_t gameover[8][12] = {
27     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
28     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
29     { 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1 },
30     { 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0 },
31     { 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0 },
32     { 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0 },
33     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
34     { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
35 };
```

```
36
37 int get_dir() {
38     // Get values from ADC (joy stick)
39     int x = analogRead(A0);
40     int y = analogRead(A1);
41
42     if((x >= 0) && (x<= 300) && dir != RIGHT) {
43         return LEFT;
44     }
45     else if( (x >=723) && (x <= 1023) && dir != LEFT) {
46         return RIGHT;
47     }
48     else if((y >= 0) && (y <= 300) && dir != DOWN) {
49         return UP;
50     }
51     else if((y >= 723) && (y <= 1023) && dir != UP) {
52         return DOWN;
53     }
54     else return dir;
55 }
56
57 void display_frame() {
58     // Clear the frame first - Turn off all LEDs
59     for(int i=0; i<8; i++) {
60         for(int j=0; j<12; j++) {
61             frame[i][j] = 0;
62         }
63     }
64
65     // Turn on LEDs for snake
66     for(int i=0; i<snake_len; i++) {
67         // Turn on LED for Body
68         frame[row_idx[i]][col_idx[i]] = 1;
69     }
70
71     // Turn on LEDs for food
72     frame[food_row][food_col] = 1;
73
74     // Display frame value in the LED matrix
75     matrix.renderBitmap(frame, 8, 12);
76 }
77
78 void game_over() {
79     // Pause Snake movement and blink for 5 times
```

```
80  for(int i=0; i<5; i++) {
81      // Display current frame (paused movement)
82      matrix.renderBitmap(frame, 8, 12);
83      delay(100);
84      // Display blank frame (to make LEDs blink)
85      matrix.renderBitmap(blank_frame, 8, 12);
86      delay(100);
87      matrix.renderBitmap(gameover, 8, 12);
88      delay(2000);
89  }
90  // Reset the length of snake
91  snake_len = 2;
92 }
93
94 void chk_snake_body_hit() {
95     for(int i=1; i<snake_len; i++) {
96         // Check if head hit the body
97         if((row_idx[i] == row_idx[0]) && (col_idx[i] == col_idx[0])){
98             game_over();
99         }
100    }
101 }
102
103 void chk_food_hit() {
104     // Logic to detect food is hit
105     if(col_idx[0] == food_col && row_idx[0] == food_row) {
106         // Increment snake length if food is hit
107         snake_len = snake_len + 1;
108         // Set food_hit to 1 to enable generation of new food coordinates
109         food_hit = 1;
110     }
111 }
112
113 void get_snake_position() {
114     // Logic to Calculate the next frame
115     // body
116     for(int i=snake_len-1; i>0; i--) {
117         col_idx[i] = col_idx[i-1];
118         row_idx[i] = row_idx[i-1];
119     }
120     // head
121     if (dir == RIGHT) {
122         // Wrap around to left
123         if(col_idx[0] == 11) {
```

```
124     col_idx[0] = 0;
125 }
126 else {
127     col_idx[0] = col_idx[0] + 1;
128 }
129 }
130 if(dir == LEFT) {
131     // Wrap around to right
132     if(col_idx[0] == 0){
133         col_idx[0] = 11;
134     }
135     else{
136         col_idx[0] = col_idx[0] - 1;
137     }
138 }
139 if (dir == DOWN) {
140     // Wrap around to top
141     if(row_idx[0] == 7){
142         row_idx[0] = 0;
143     }
144     else{
145         row_idx[0] = row_idx[0] + 1;
146     }
147 }
148 if(dir == UP) {
149     // Wrap around to bottom
150     if(row_idx[0] == 0){
151         row_idx[0] = 7;
152     }
153     else{
154         row_idx[0] = row_idx[0] - 1;
155     }
156 }
157 }

158 void gen_food() {
159     // Generate coordinates for food
160     if(food_hit == 1){
161         food_col = random(0, 12);
162         food_row = random(0, 8);
163         // Set food_hit to 0 so it will not randomize food coordinates unless hit
164         food_hit = 0;
165     }
166 }
```

```
168
169 void loop() {
170     // Get the snake direction
171     dir = get_dir();
172     // Display food and snake in the LED Matrix
173     display_frame();
174     // Check if snake body is hit by its head
175     chk_snake_body_hit();
176     // Check if the food is hit
177     chk_food_hit();
178     // Get the snake positions for the next frame
179     get_snake_position();
180     // Generate random coordinates for food
181     gen_food();
182     // Speed of the animation/snake movement
183     delay(speed);
184 }
```

This experiment comes from the Internet. If you want to know more about the experiment, you can visit the following link:

<https://www.eeplayground.com/posts/snake-game-arduino/>

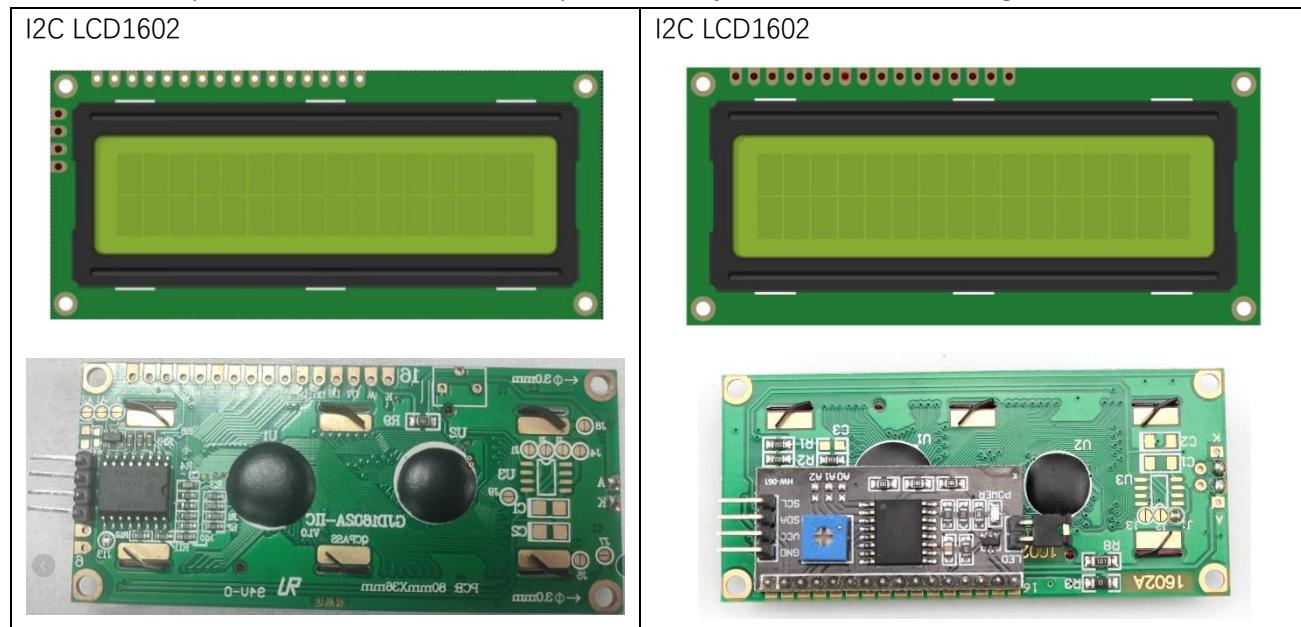
In addition, you can also refer to the following link. This experiment is a more multifunctional snake game experiment, adding scoring and other functions.

<https://github.com/siphyshu/snake-R4>

Chapter 20 I2C LCD1602

In the previous chapter, we have used the LED matrix to display images and characters. Now, let us use a screen module LCD1602 with a higher resolution to display more content.

There are multiple versions of LCD1602. Your purchase may be one of the following:



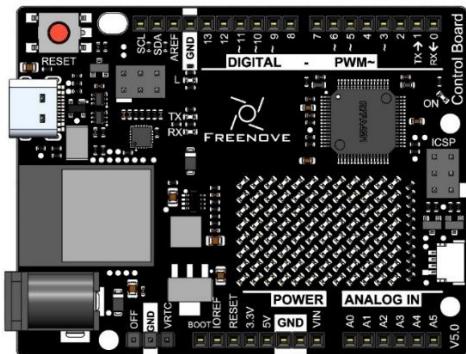
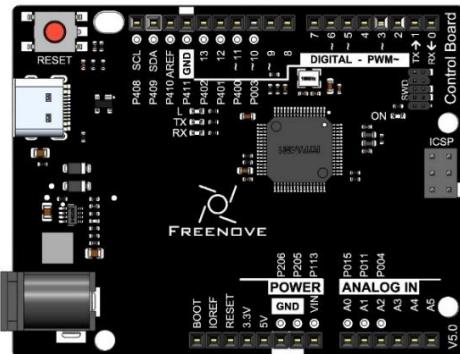
There is a conversion module on the back of the I2C LCD1602 module. Convert parallel interface to I2C serial interface. Although they have different interfaces, their functions are the same.

Project 20.1 Display the String on I2C LCD1602

In this chapter, we will learn about the LCD1602 Display Screen. Firstly, use I2C LCD1602 to display some strings.

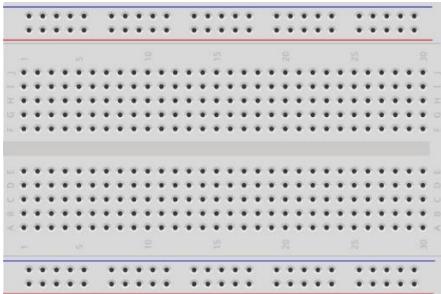
Component List

Control board x1

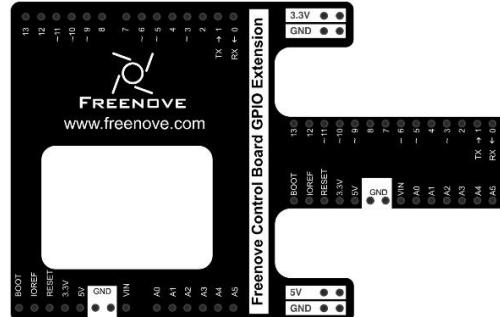


or

Breadboard x1



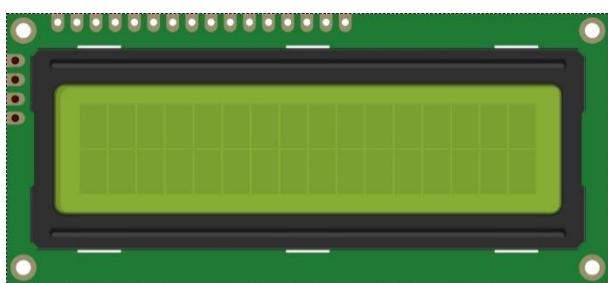
GPIO Extension Board x1



USB cable x1



I2C LCD1602 Module x1



Jumper M/F x4

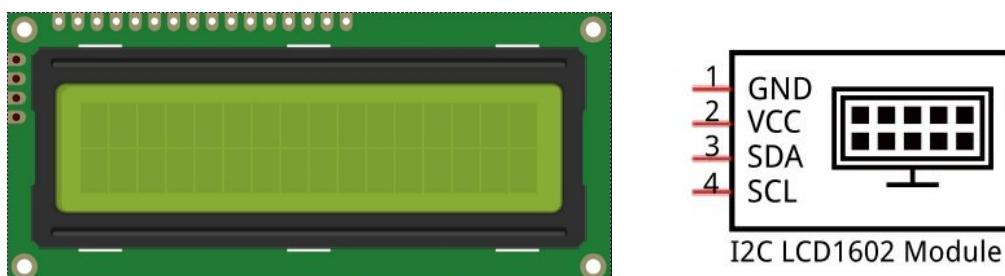


Component Knowledge

I2C LCD1602

LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on.

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



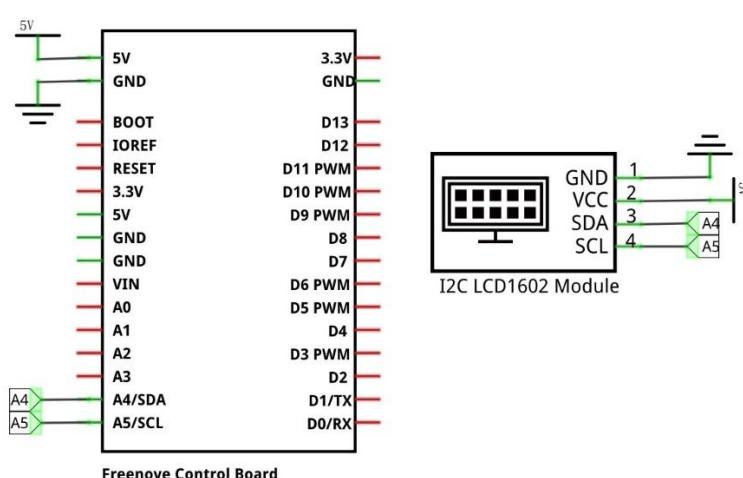
I2C (Inter-Integrated Circuit) is a two-wire serial communication, which is used to connect micro controller with its peripheral equipments. Device that use I2C communication are all connected to the serial data (SDA) line and a serial clock (SCL) line (called I2C bus). Each device among them has a unique address and can be a transmitter or receiver. Additionally, they can communicate with devices connected to the Bus.

Next, let's try to use the LCD1602 I2C module to display characters.

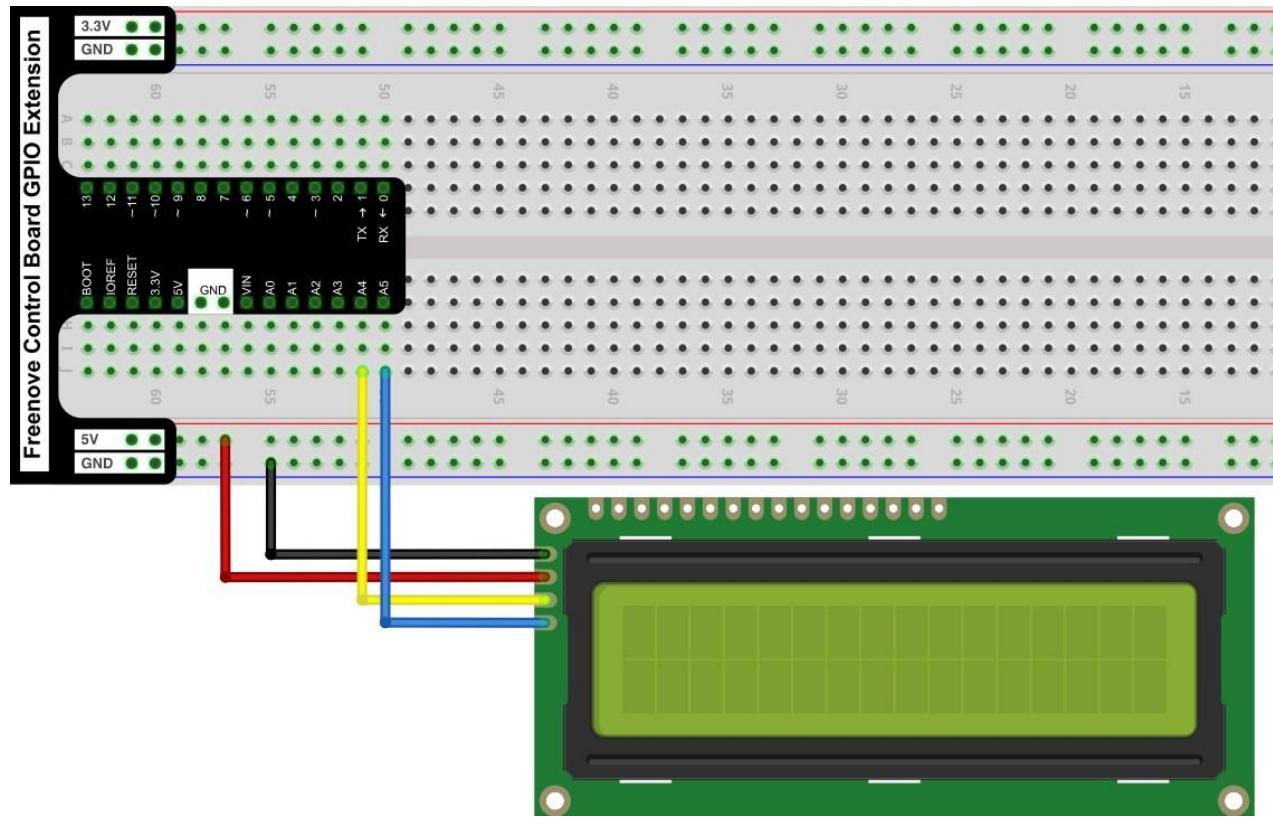
Circuit

The connection of control board and I2C LCD1602 is shown below.

Schematic diagram



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



Sketch

Sketch 20.1.1

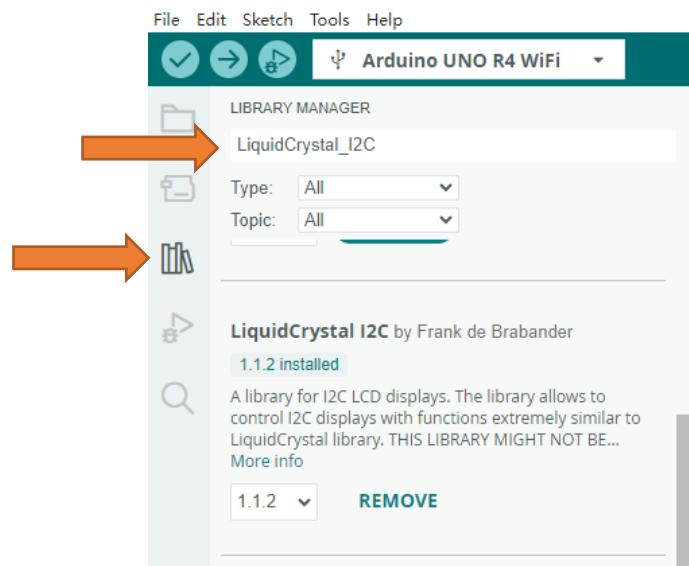
This code uses a library named "LiquidCrystal_I2C", if you have not installed it, please do so first.

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.

How to install the library

There are two ways to add libraries.

you can search "LiquidCrystal_I2C" in library manager to install.



The second way, Click "Add .ZIP Library..." and then find LiquidCrystal_I2C.zip in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library can facilitate our operation of I2C LCD1602. Now let's start to write code to use LCD1602 to display static characters and dynamic variables.

```

1 #include <LiquidCrystal_I2C.h>
2
3 // initialize the library with the numbers of the interface pins
4 LiquidCrystal_I2C lcd(0x27, 16, 2);
5
6 void setup() {
7     if (!i2CAddrTest(0x27)) {
8         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
9     }
10    lcd.init();           // initialize the lcd
11    lcd.backlight();      // Turn on backlight
12    lcd.print("hello, world!"); // Print a message to the LCD
13 }
14 void loop() {
15     // (note: line 1 is the second row, since counting begins with 0):
16     lcd.setCursor(0, 1); // set the cursor to column 0, line 1
17     // print the number of seconds since reset:
18     lcd.print("Counter:");
19     lcd.print(millis() / 1000);
20 }
21 bool i2CAddrTest(uint8_t addr) {
22     Wire.begin();
23     Wire.beginTransmission(addr);
24     if (Wire.endTransmission() == 0) {
25         return true;
26     }
27     return false;

```

```
28 }
```

Following are the LiquidCrystal_I2C library used for controlling LCD:

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

LiquidCrystal_I2C library provides LiquidCrystal_I2C class that controls LCD1602. When we instantiate a LiquidCrystal_I2C object, we can input some parameters. And these parameters are the row/column numbers of the I2C addresses and screen that connect to LCD1602:

```
4 LiquidCrystal_I2C lcd(0x27, 16, 2); // set the LCD address to 0x27 for a 16 chars and 2 line display
```

First, initialize the LCD and turn on LCD backlight.

```
10 lcd.init(); // initialize the lcd  
11 lcd.backlight(); // Turn on backlight
```

And then print a string:

```
12 lcd.print("hello, world!"); // Print a message to the LCD
```

Print a changing number in the loop () function:

```
14 void loop() {  
15     // (note: line 1 is the second row, since counting begins with 0):  
16     lcd.setCursor(0, 1); // set the cursor to column 0, line 1  
17     // print the number of seconds since reset:  
18     lcd.print("Counter:");  
19     lcd.print(millis() / 1000);  
20 }
```

Before printing characters, we need to set the coordinate of the printed character, that is, in which line and which column:

```
16     lcd.setCursor(0, 1); // set the cursor to column 0, line 1  
LiquidCrystal_I2C Class
```

LiquidCrystal_I2C class can control common LCD screen. First, we need instantiate an object of LiquidCrystal_I2C type, for example:

LiquidCrystal_I2C lcd(0x27, 16, 2);

When an object is instantiated, a constructed function of the class is called a constructor. In the constructor function, we need to fill in the I2C address of the LCD module, as well as the number of columns and rows of the LCD module. The number of columns and rows can also be set in the lcd.begin () .

The functions used in the LiquidCrystal_I2C class are as follows:

lcd.setCursor (col, row): set the coordinates of the to-be-printed character. The parameters are the numbers of columns and rows of the characters (start from 0, the number 0 represents first row or first line).

lcd.print (data): print characters. Characters will be printed on the coordinates set before. If you do not set the coordinates, the string will be printed behind the last printed character.

Verify and upload the code, then observe the LCD screen. If the display is not clear or there is no display, adjust the potentiometer on the back of I2C module to adjust the screen contrast until the character is clearly displayed on the LCD.



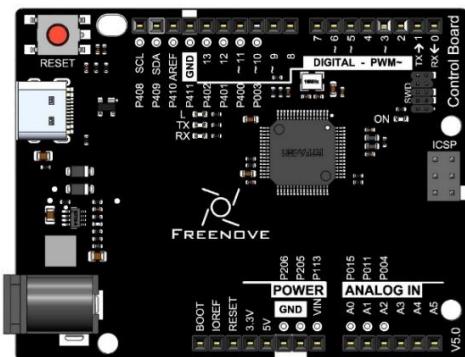
You can use the I2C LCD1602 to replace the serial port as a mobile screen when you print the data latter.

Project 20.2 I2C LCD1602 Clock

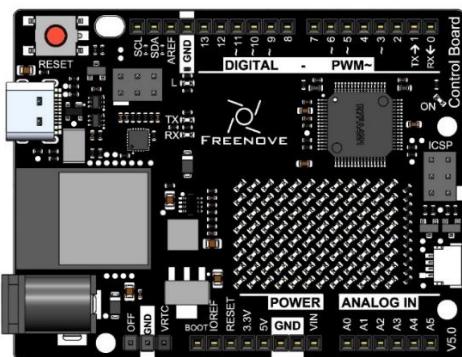
In the previous chapter, we have used I2C LCD1602 to display some strings, and now let us use I2C LCD1602 to display the temperature sensor value.

Component List

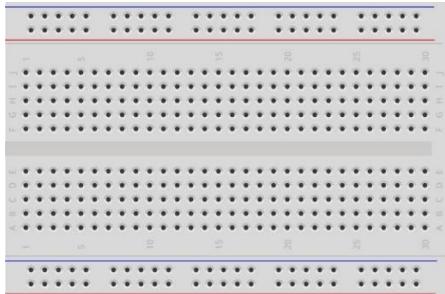
Control board x1



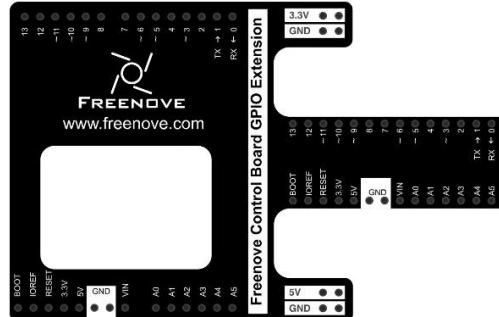
or



Breadboard x1



GPIO Extension Board x1



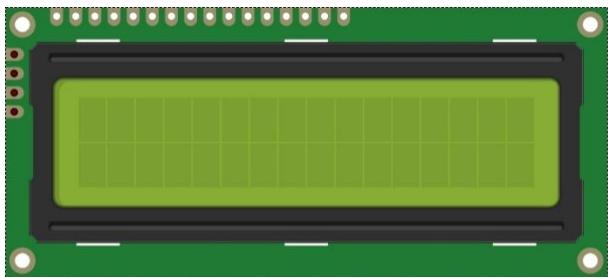
USB cable x1



Jumper M/M x7



I2C LCD1602 Module x1



Thermistor x1



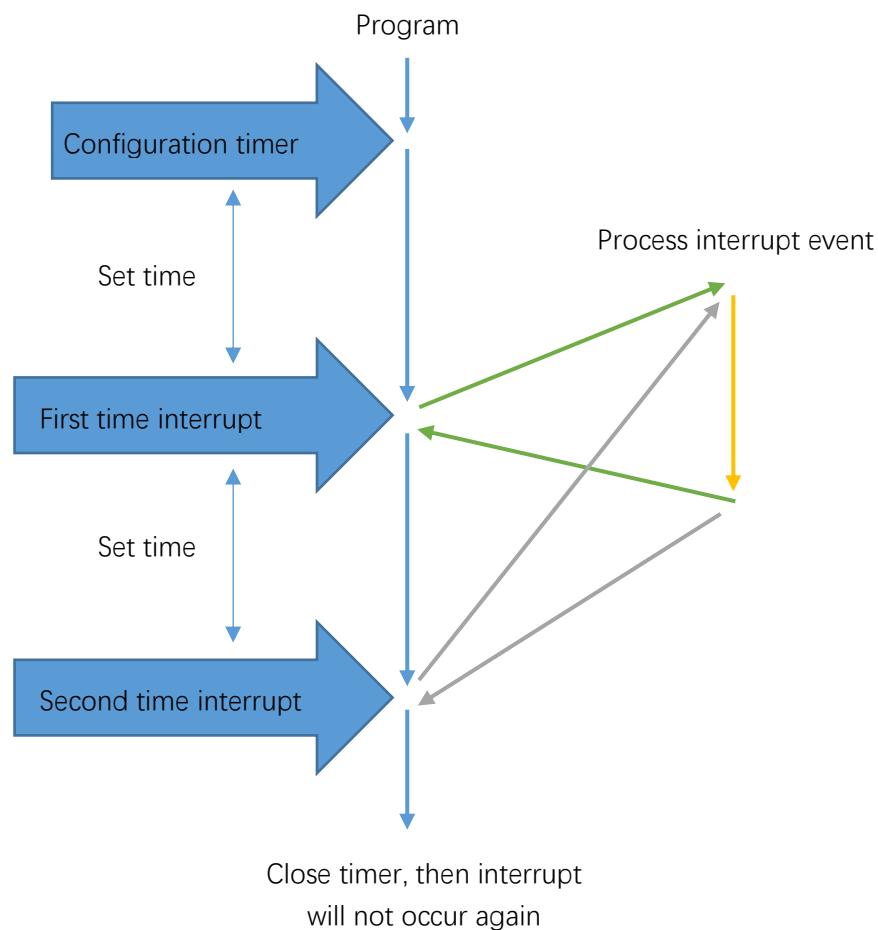
Resistor 10k Ω x1



Code Knowledge

Timer

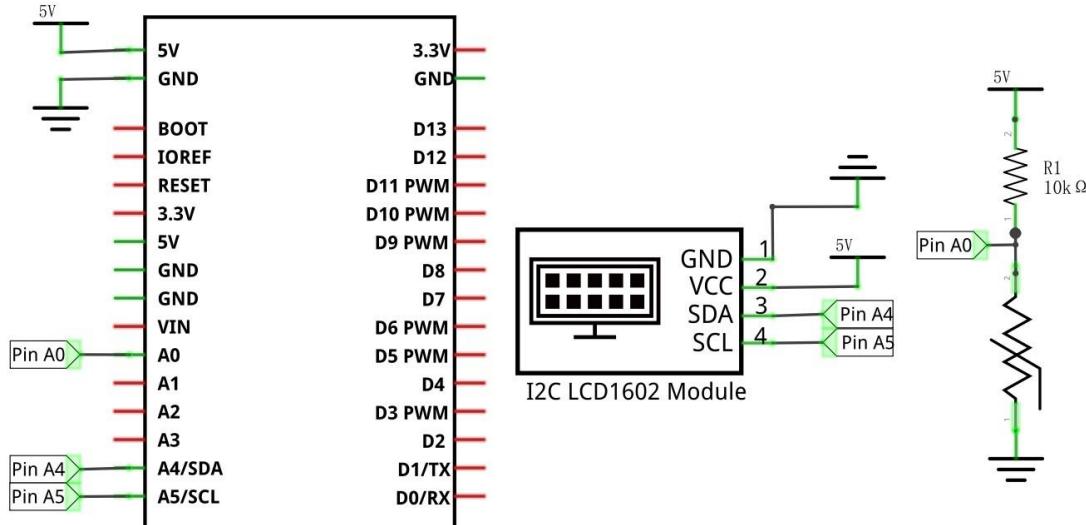
A Timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted location to go on. If you don't close the timer, interrupt will occur at the intervals you set.



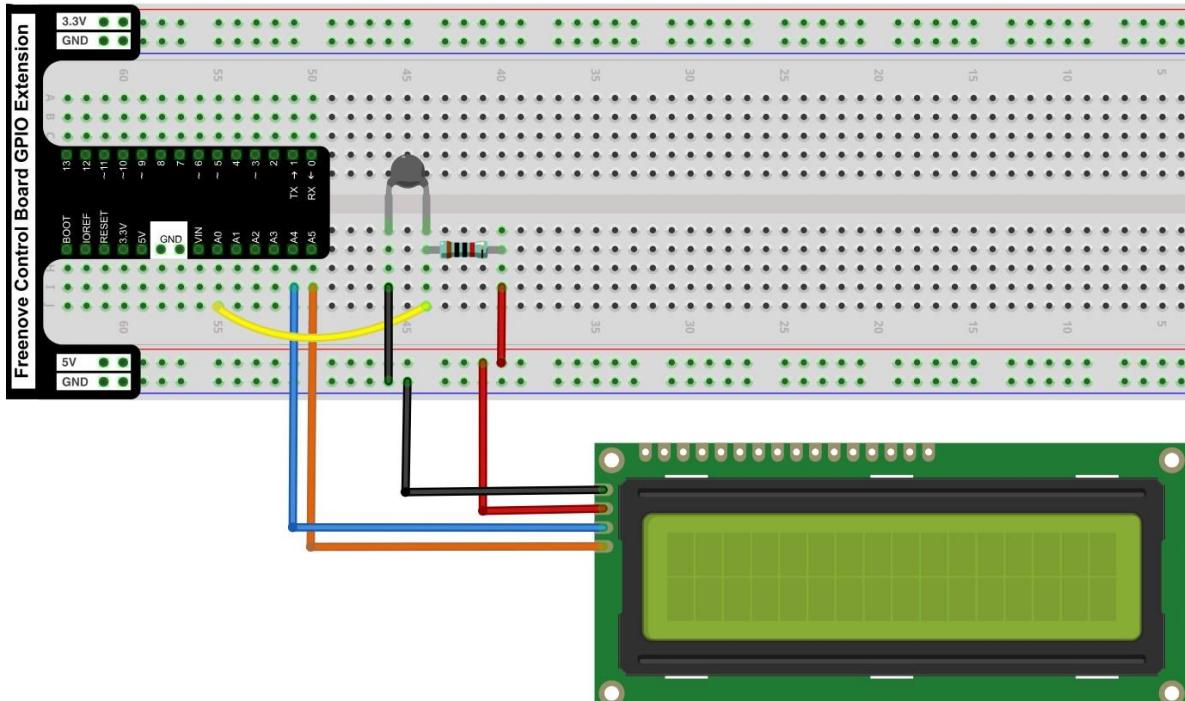
Circuit

The connection is shown below. Pin A0 is used to detect the voltage of thermistor.

Schematic diagram



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



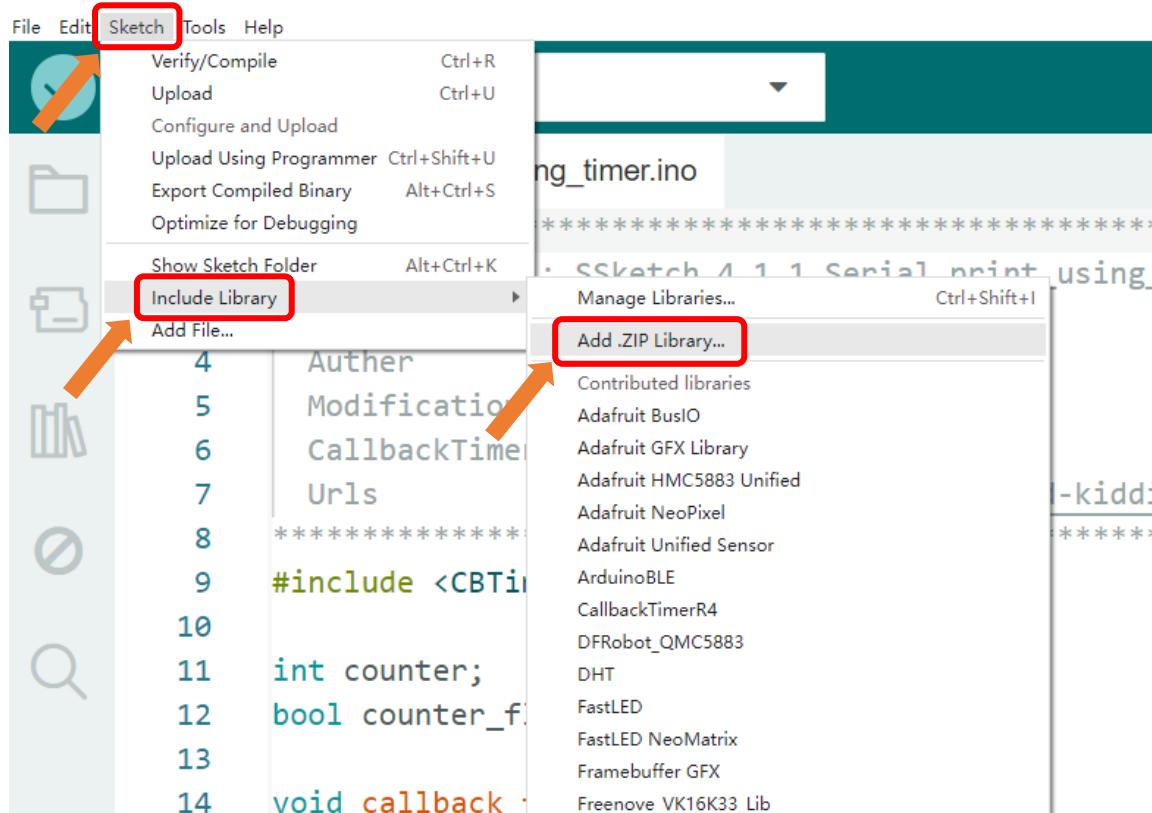
Sketch

This code uses a library named "CallbackTimerR4 ", 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.

How to install the library

open Arduino IDE, click Sketch→Include Library→Add .ZIP Library, In the pop-up window, find the file named ".Libraries/ **CallbackTimerR4-main.zip**" which locates in this directory, and click OPEN.



Sketch 20.2.1

The CBTimer library is used here to manipulate timers.

Now write code to make LCD1602 display the time and temperature, and the time can be modified through the serial port.

```

1 #include <LiquidCrystal_I2C.h>
2 #include "CBTimer.h"
3
4 // initialize the library with the numbers of the interface pins
5 LiquidCrystal_I2C lcd(0x27, 16, 2);
6 int tempPin = 0; // define the pin of temperature sensor
7 float tempVal; // define a variable to store temperature
8 value
9 int hour, minute, second; // define variables stored record time

```

```
10
11 void setup() {
12     if (!i2CAddrTest(0x27)) {
13         lcd = LiquidCrystal_I2C(0x3F, 16, 2);
14     }
15     lcd.init(); // initialize the lcd
16     lcd.backlight(); // Turn on backlight
17     startingAnimation(); // display a dynamic start screen
18     Serial.begin(115200); // initialize serial port with baud
19     rate 9600
20     Serial.println("Input hour,minute,second to set time.");
21     static CBTimer timer;
22     timer.begin(1000, timerInt);
23 }
24
25 void loop() {
26     // Get temperature
27     tempVal = getTemp();
28     if (second >= 60) { // when seconds is equal to 60, minutes plus 1
29         second = 0;
30         minute++;
31         if (minute >= 60) { // when minutes is equal to 60, hours plus 1
32             minute = 0;
33             hour++;
34             if (hour >= 24) { // when hours is equal to 24, hours turn to
35             zero
36                 hour = 0;
37             }
38         }
39     }
40     lcdDisplay(); // display temperature and time information on
41     LCD
42     delay(200);
43     serial_Event();
44 }
45
46
47 void startingAnimation() {
48     for (int i = 0; i < 16; i++) {
49         lcd.scrollDisplayRight();
50     }
51     lcd.print("starting...");
52     for (int i = 0; i < 16; i++) {
53         lcd.scrollDisplayLeft();
```

```
54     delay(300);
55 }
56 lcd.clear();
57 }
58 // the timer interrupt function of FlexiTimer2 is executed every 1s
59 void timerInt() {
60     second++;           // second plus 1
61 }
62 // serial port interrupt function
63 void serial_Event() {
64     int inInt[3]; // define an array to save the received serial data
65     while (Serial.available()) {
66         for (int i = 0; i < 3; i++) {
67             inInt[i] = Serial.parseInt(); // receive 3 integer data
68         }
69         // print the received data for confirmation
70         Serial.print("Your input is: ");
71         Serial.print(inInt[0]);
72         Serial.print(", ");
73         Serial.print(inInt[1]);
74         Serial.print(", ");
75         Serial.println(inInt[2]);
76         // use received data to adjust time
77         hour = inInt[0];
78         minute = inInt[1];
79         second = inInt[2];
80         // print the modified time
81         Serial.print("Time now is: ");
82         Serial.print(hour / 10);
83         Serial.print(hour % 10);
84         Serial.print(':');
85         Serial.print(minute / 10);
86         Serial.print(minute % 10);
87         Serial.print(':');
88         Serial.print(second / 10);
89         Serial.println(second % 10);
90     }
91 }
92 // function used by LCD1602 to display time and temperature
93 void lcdDisplay() {
94     lcd.setCursor(0, 0); // set the cursor to (0,0) (first column,first
95     row).
96     lcd.print("TEMP: "); // display temperature information
97     lcd.print(tempVal);
```

```

98     lcd.print("C");
99     lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second
100    row)
101    lcd.print("TIME: "); // display time information
102    lcd.print(hour / 10);
103    lcd.print(hour % 10);
104    lcd.print(':');
105    lcd.print(minute / 10);
106    lcd.print(minute % 10);
107    lcd.print(':');
108    lcd.print(second / 10);
109    lcd.print(second % 10);
110 }
111 // function used to get temperature
112 float getTemp() {
113     // Convert analog value of tempPin into digital value
114     int adcVal = analogRead(tempPin);
115     // Calculate voltage
116     float v = adcVal * 5.0 / 1024;
117     // Calculate resistance value of thermistor
118     float Rt = 10 * v / (5 - v);
119     // Calculate temperature (Kelvin)
120     float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));
121     // Calculate temperature (Celsius)
122     return tempK - 273.15;
123 }
124
125 bool i2CAaddrTest(uint8_t addr) {
126     Wire.begin();
127     Wire.beginTransmission(addr);
128     if (Wire.endTransmission() == 0) {
129         return true;
130     }
131     return false;
132 }
```

In the code, we define 3 variables to represent time: second, minute, hour.

```
int hour, minute, second; // define variables stored record time
```

Defines a timer with cycle of 1000 millisecond (1 second). And each interrupt makes the second plus 1. When setting the timer, you need to define a function and pass the function name that works as a parameter to `timer.begin()` function.

In the `timer.begin()` function, there are two parameters to pass. The first parameter is: time, and the second parameter is the function to be executed after the first parameter time.

```
static CBTimer timer;
timer.begin(1000, timerInt);
```

After every interrupt, the second plus 1.

```
void timerInt() {
    second++; // second plus 1
}
```

In the loop () function, the information on the LCD display will be refreshed at set intervals.

```
void loop() {
    ...
    lcdDisplay(); // display temperature and time information on LCD
    delay(200);
    ..serial_Event();
}
```

In the loop function, we need to control the second, minute, hour. When the second increases to 60, the minute adds 1, and the second is reset to zero; when the minute increases to 60, the hour adds 1, and the minute is reset to zero; when the hour increases to 24, reset it to zero.

```
if (second >= 60) { // when the second increases to 60, the minute adds 1
    second = 0;
    minute++;
    if (minute >= 60) { //when the minute increases to 60, the hour adds 1
        minute = 0;
        hour++;
        if (hour >= 24) { // when the hour increases to 24, turn it to zero
            hour = 0;
        }
    }
}
```

We define a function lcdDisplay () to refresh the information on LCD display. In this function, use two bit to display the hour, minute, second on the LCD. For example, hour/ 10 is the unit, hour% 10 is the tens.

```
void lcdDisplay() {
    lcd.setCursor(0, 0); // set the cursor to (0,0) (first column, first row).
    lcd.print("TEMP: "); // display temperature information
    lcd.print(tempVal);
    lcd.print("C");
    lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second row)
    lcd.print("TIME: "); // display time information
    lcd.print(hour / 10);
    lcd.print(hour % 10);
    lcd.print(':');
    lcd.print(minute / 10);
    lcd.print(minute % 10);
    lcd.print(':');
    lcd.print(second / 10);
    lcd.print(second % 10);
}
```

The serial_Event function is used to receive the data sent by the computer to adjust the time and return the

data for confirmation.

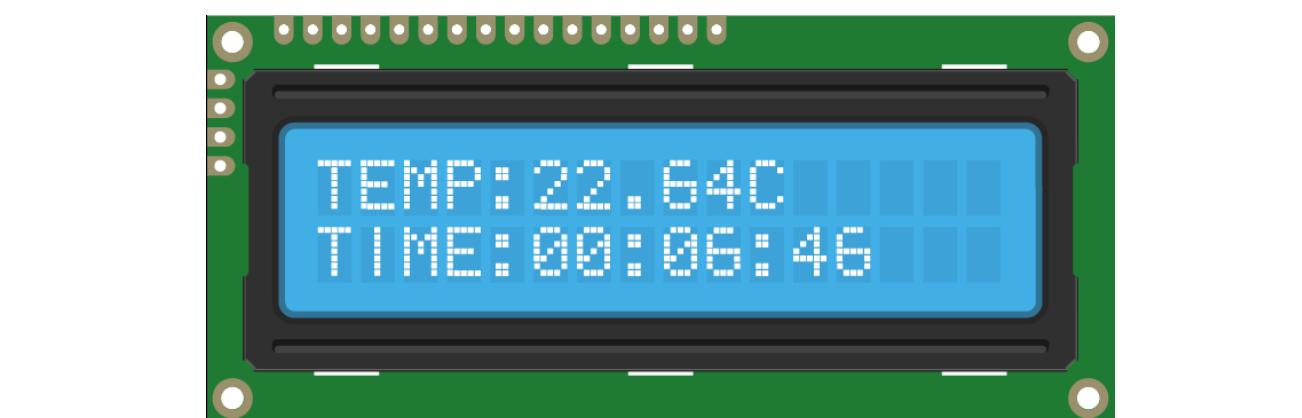
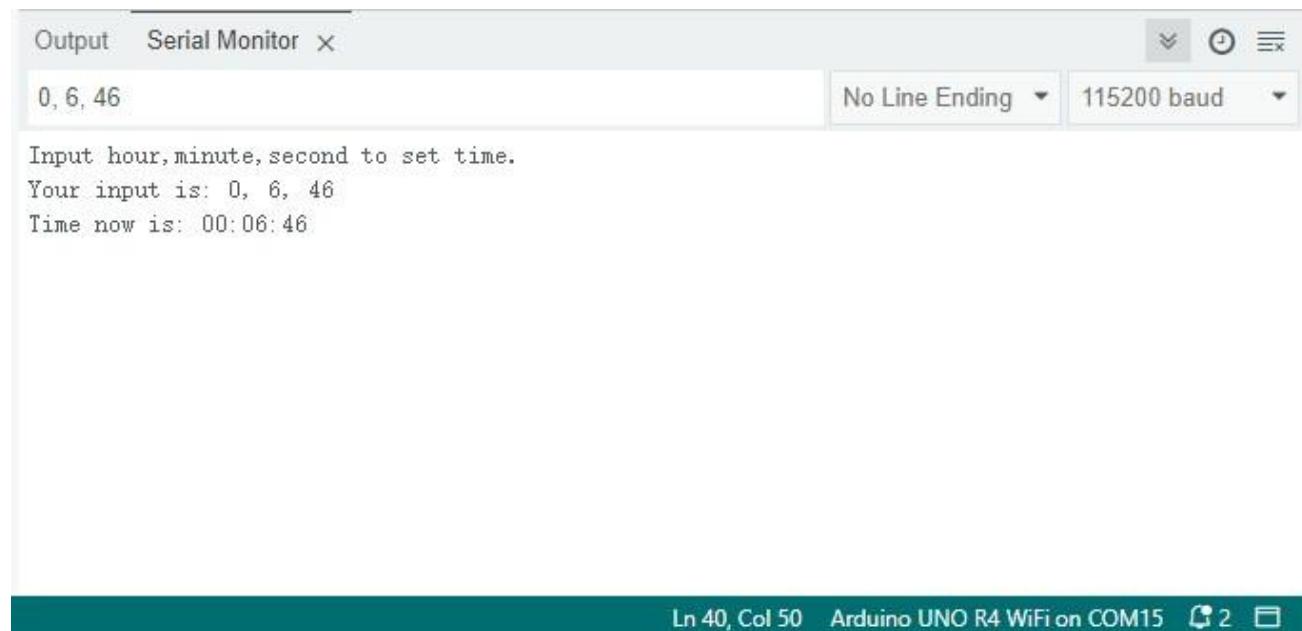
```
void serial_Event() {  
    int inInt[3]; // define an array to save the received serial data  
    while (Serial.available()) {  
        for (int i = 0; i < 3; i++) {  
            inInt[i] = Serial.parseInt(); // receive 3 integer data  
        }  
        // print the received data for confirmation  
        Serial.print("Your input is: ");  
        Serial.print(inInt[0]);  
        Serial.print(", ");  
        Serial.print(inInt[1]);  
        Serial.print(", ");  
        Serial.println(inInt[2]);  
        // use the received data to adjust time  
        hour = inInt[0];  
        minute = inInt[1];  
        second = inInt[2];  
        // print the modified time  
        Serial.print("Time now is: ");  
        Serial.print(hour / 10);  
        Serial.print(hour % 10);  
        Serial.print(':');  
        Serial.print(minute / 10);  
        Serial.print(minute % 10);  
        Serial.print(':');  
        Serial.print(second / 10);  
        Serial.println(second % 10);  
    }  
}
```

We also define a function that displays a scrolling string when the control board has been just started.

```
void startingAnimation() {  
    for (int i = 0; i < 16; i++) {  
        lcd.scrollDisplayRight();  
    }  
    lcd.print("starting...");  
    for (int i = 0; i < 16; i++) {  
        lcd.scrollDisplayLeft();  
        delay(300);  
    }  
    lcd.clear();  
}
```

Verify and upload the code. The LCD screen will display a scrolling string first, and then displays the

temperature and time. We can open Serial Monitor and enter time in the sending area, then click the Send button to set the time.



Chapter 21 Digital Display

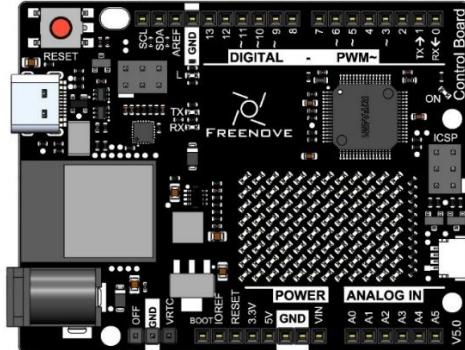
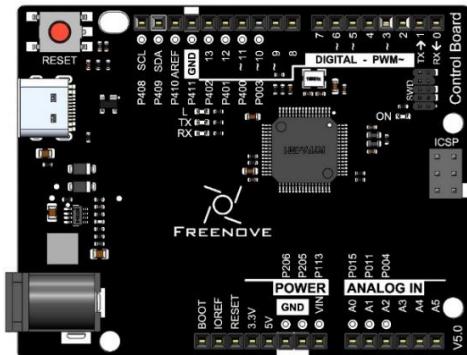
Digital display is a kind of device that can display one or several digits. We will learn how to use it in this chapter.

Project 21.1 1-digit 7-segment Display

First, try to use the digit display that can display 1-digit number.

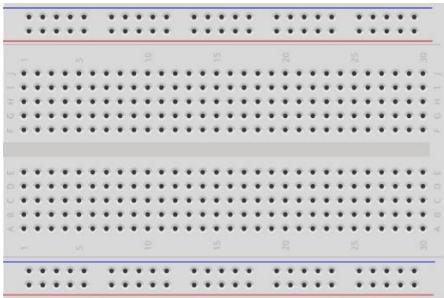
Component List

Control board x1

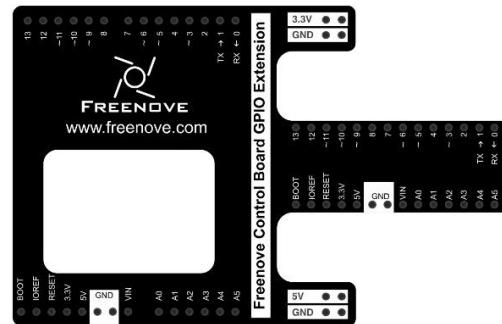


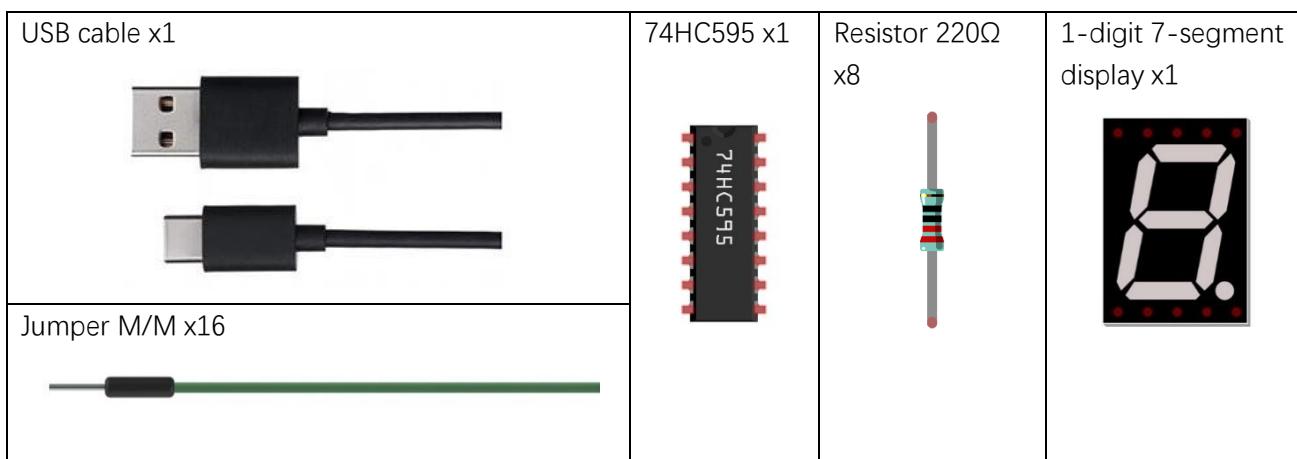
or

Breadboard x1



GPIO Extension Board x1

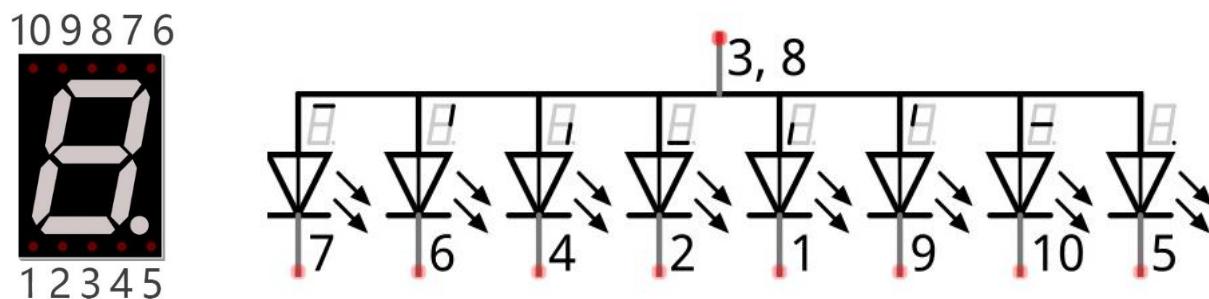




Component Knowledge

1-digit 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. It can display numbers of 0~9 by lighting some of its LED segment. 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 7, 6, 4, 2, 1 and 9, and turn OFF LED segments 5 and 10.



If we use a byte to show the state of the LEDs that connected to pin 5, 10, 9, 1, 2, 4, 6, 7, we can make 0 represent the state of ON and 1 for OFF. Then the number 0 can be expressed as a binary number 11000000, namely hex 0xc0.

The numbers and letters that can be display are shown below:

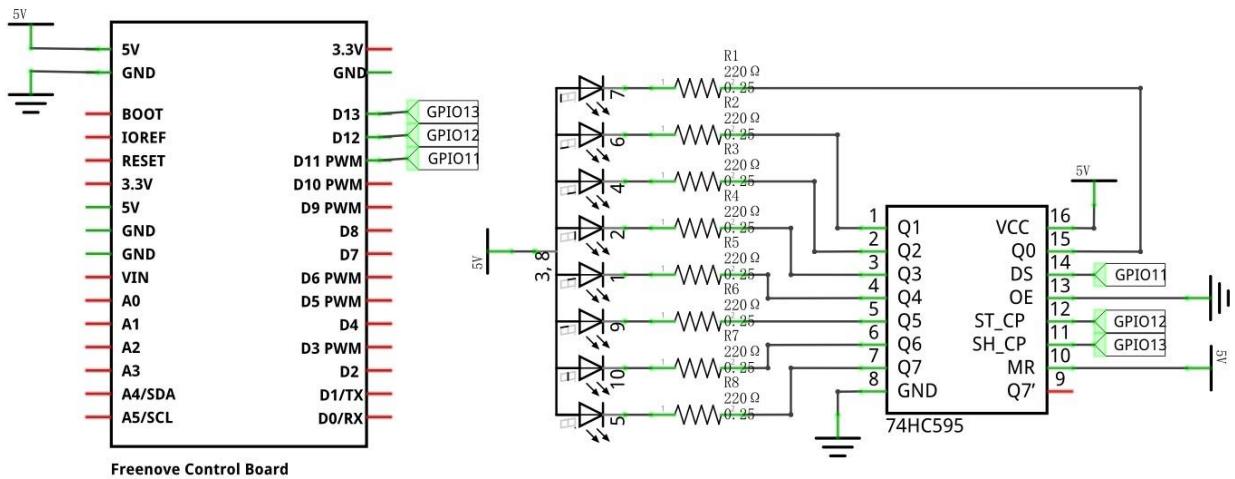
Number/Letter	Binary number	Hexadecimal number
0	11000000	0xc0
1	11111001	0xf9

2	10100100	0xa4
3	10110000	0xb0
4	10011001	0x99
5	10010010	0x92
6	10000010	0x82
7	11111000	0xf8
8	10000000	0x80
9	10010000	0x90
A	10001000	0x88
b	10000011	0x83
C	11000110	0xc6
d	10100001	0xa1
E	10000110	0x86
F	10001110	0x8e

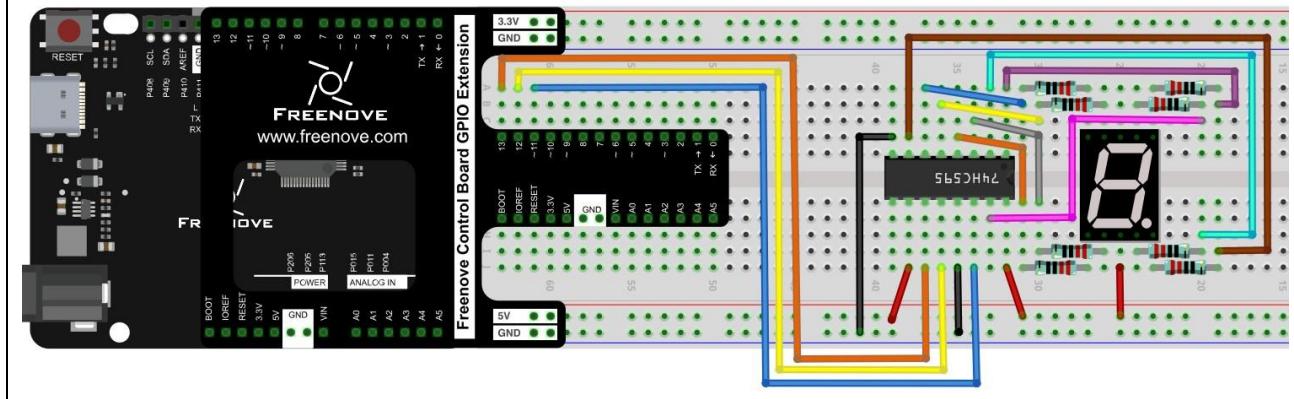
Circuit

Use pin 11, 12, 13 on control board to control the 74HC595, and connect it to a 1-digit 7-segment display.

Schematic diagram



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



Sketch

Sketch 21.1.1

Now write code to control the 1-digit 7-segment display through 74HC595.

```
1 int latchPin = 12;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 13;           // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 11;            // Pin connected to DS of 74HC595 (Pin14)
4
5 // Define the encoding of characters 0-F of the common-anode 7-segment Display
6 byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83,
7 0xc6, 0xa1, 0x86, 0x8e};
8
9 void setup() {
10    // set pins to output
11    pinMode(latchPin, OUTPUT);
12    pinMode(clockPin, OUTPUT);
13    pinMode(dataPin, OUTPUT);
14 }
15
16 void loop() {
17    // Cycling display 0-F
18    for (int i = 0; i <= 0x0f; i++) {
19        // Output low level to latchPin
20        digitalWrite(latchPin, LOW);
21        // Send serial data to 74HC595
22        shiftOut(dataPin, clockPin, MSBFIRST, num[i]);
23        // Output high level to latchPin, and 74HC595 will update the data to the parallel
24        // output port.
25        digitalWrite(latchPin, HIGH);
26        delay(1000);
27    }
28 }
```

We define an array to save and display the encoding of character 0-F in this code.

```
6 byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83,
7 0xc6, 0xa1, 0x86, 0x8e};
```

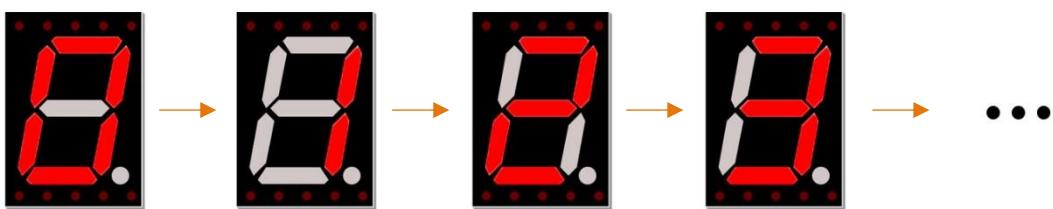
Initialize the pin connected to 74HC595 in setup() function.

```
8 void setup() {
9    // set pins to output
10   pinMode(latchPin, OUTPUT);
11   pinMode(clockPin, OUTPUT);
12   pinMode(dataPin, OUTPUT);
13 }
```

Then in loop() function, send the encoding of 0-F to 74HC595 circularly.

```
17  for (int i = 0; i <= 0x0f; i++) {  
18      // Output low level to latchPin  
19      digitalWrite(latchPin, LOW);  
20      // Send serial data to 74HC595  
21      shiftOut(dataPin, clockPin, MSBFIRST, num[i]);  
22      // Output high level to latchPin, and 74HC595 will update the data to the parallel  
23      // output port.  
24      digitalWrite(latchPin, HIGH);  
25      delay(1000);  
}
```

Verify and upload the code, then you will see the 1-digit 7-segment display show 0-F in a circular manner.

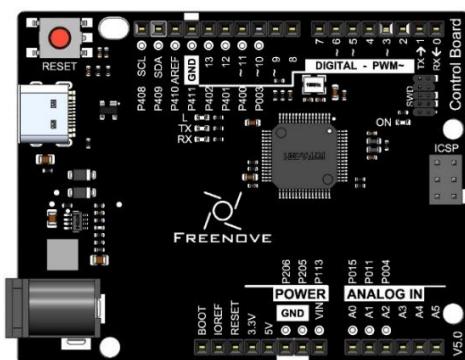


Project 21.2 4-digit 7-segment Display

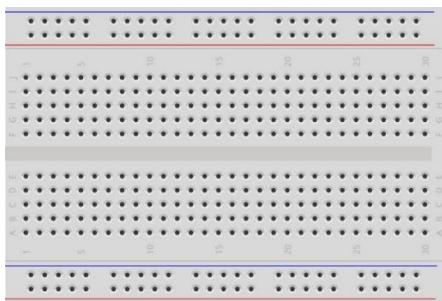
Now, try to use digit display that can display 4-digit numbers.

Component List

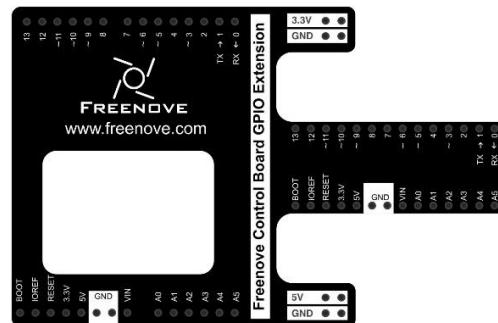
Control board x1



Breadboard x1



GPIO Extension Board x1



USB cable x1



Jumper M/M x19



4-digit 7-segment display x1



74HC595 x1



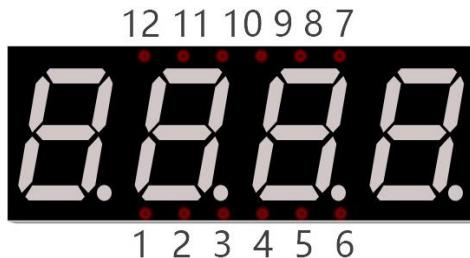
Resistor 1k Ω x 8



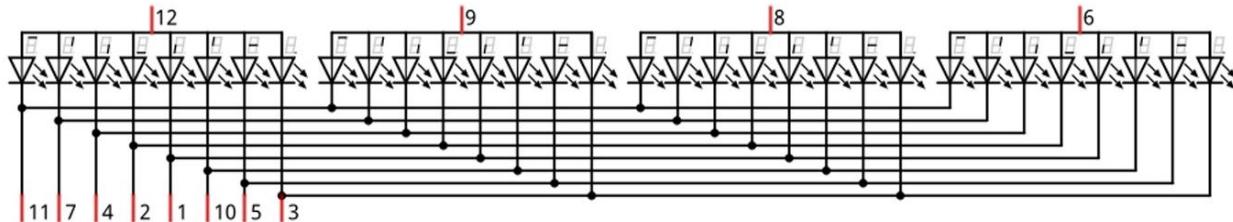
Component Knowledge

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 of 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 eight LED cathode pins of each 1-digit 7-segment display are connected together.



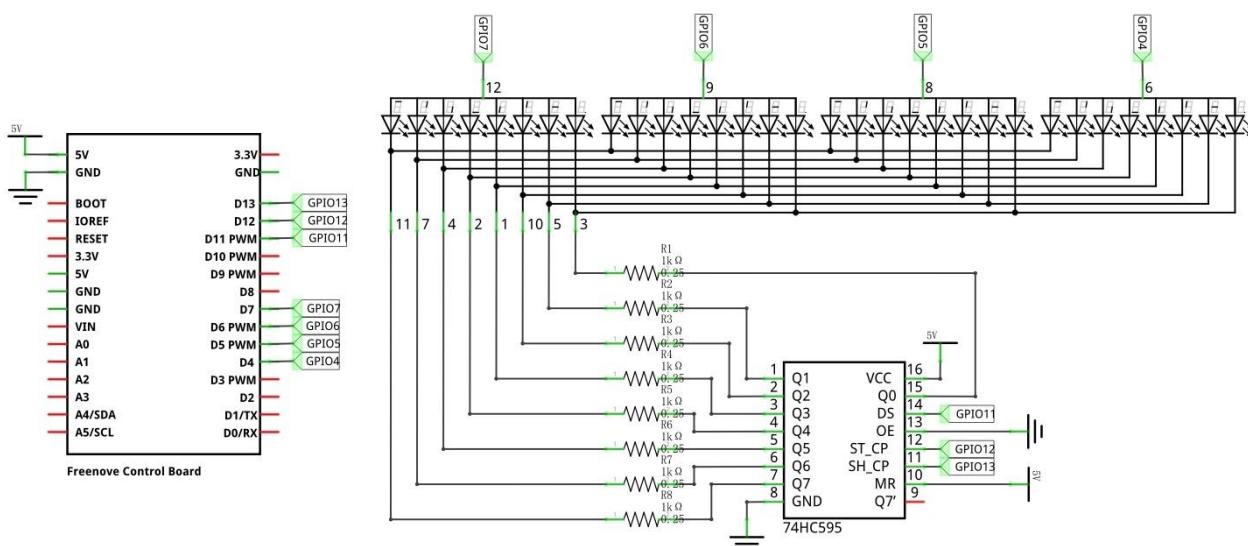
Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to 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 unperceivable 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.

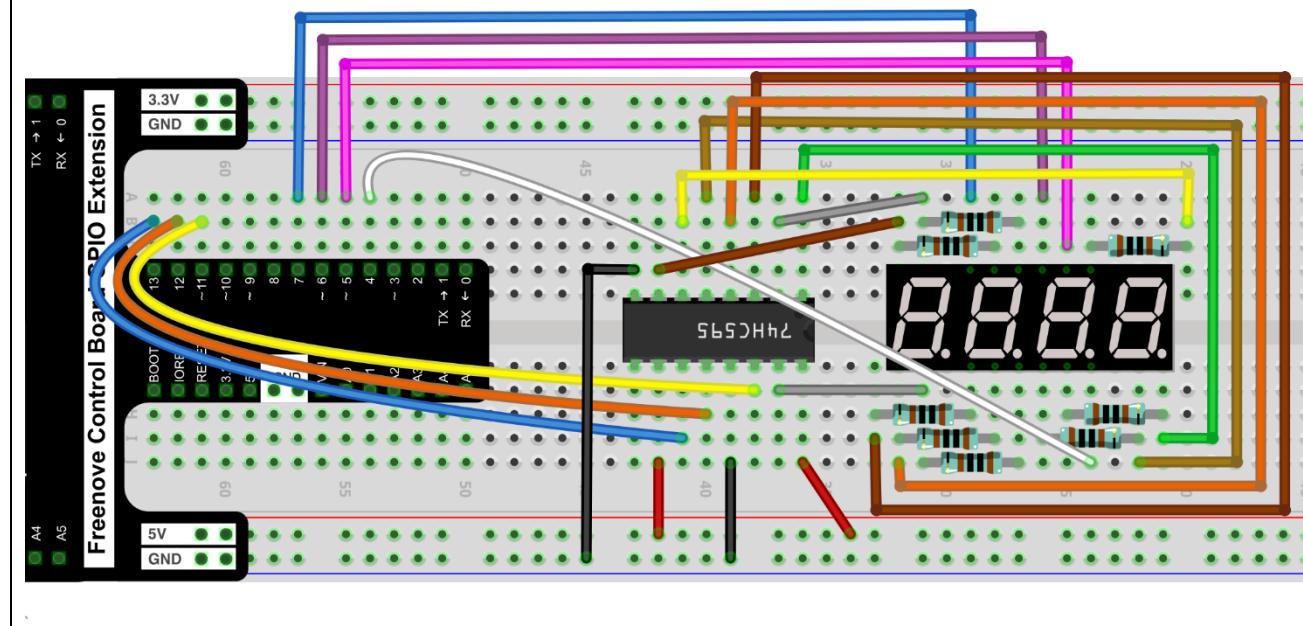
Circuit

Use pin 11, 12, 13 on control board to control the 74HC595, and connect it to the 4-digit 7-segment display. Use pin 7, 6, 5, 4 to control the 4 common ports.

Schematic diagram



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



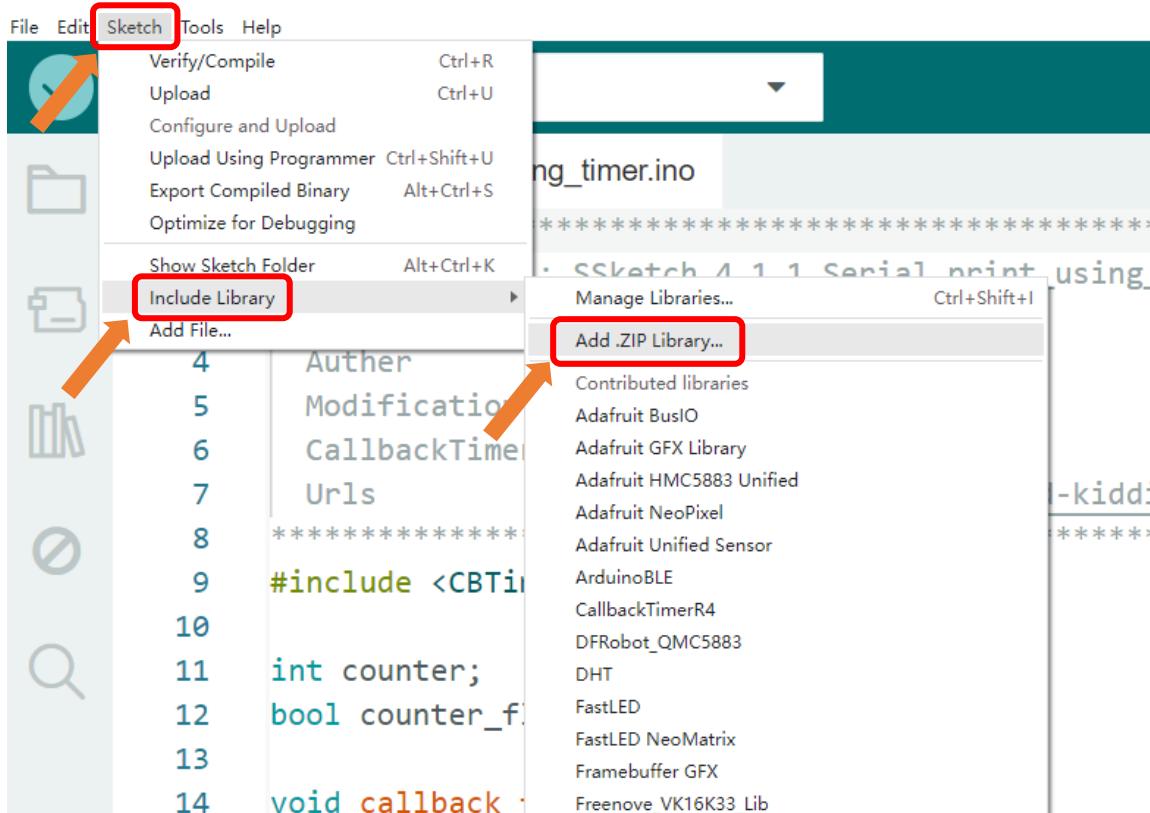
Sketch

This code uses a library named "CallbackTimerR4", 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.

How to install the library

open Arduino IDE, click Sketch → Include Library → Add .ZIP Library, In the pop-up window, find the file named "./Libraries/ **CallbackTimerR4-main.zip**" which locates in this directory, and click OPEN.



Sketch 21.2.1

Now, write code to control 4-digit 7-segment display to display 4 numbers.

```

1 int latchPin = 12;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
4 int comPin[] = {7, 6, 5, 4};// 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, 0x80, 0x90, 0x88, 0x83,
8 0xc6, 0xa1, 0x86, 0x8e};
9
10 void setup() {
    // set pins to output
  
```

```
11  pinMode(latchPin, OUTPUT);
12  pinMode(clockPin, OUTPUT);
13  pinMode(dataPin, OUTPUT);
14  for (int i = 0; i < 4; i++) {
15      pinMode(comPin[i], OUTPUT);
16  }
17 }
18
19 void loop() {
20     for (int i = 0; i < 4; i++) {
21         // Select a single 7-segment display
22         chooseCommon(i);
23         // Send data to 74HC595
24         writeData(num[i]);
25         delay(5);
26         // Clear the display content
27         writeData(0xff);
28     }
29 }
30
31 void chooseCommon(byte com) {
32     // Close all single 7-segment display
33     for (int i = 0; i < 4; i++) {
34         digitalWrite(comPin[i], LOW);
35     }
36     // Open the selected single 7-segment display
37     digitalWrite(comPin[com], HIGH);
38 }
39
40 void writeData(int value) {
41     // Make latchPin output low level
42     digitalWrite(latchPin, LOW);
43     // Send serial data to 74HC595
44     shiftOut(dataPin, clockPin, LSBFIRST, value);
45     // Make latchPin output high level, then 74HC595 will update the data to parallel
46     // output
47     digitalWrite(latchPin, HIGH);
}
```

Besides the similarity with the previous section, the difference is that this code is to output content to the four 7-segment display continuously. Write a function to select a common port.

```

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

Write a function to send data to 74HC595.

```

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

First select a common port and then output the content, which will be displayed by the 7-segment display connected to common port, to 74HC595 when operating. Clear the display content after a period of time to avoid ghosting phenomenon.

```

21     // Select a single 7-segment display
22     chooseCommon(i);
23     // Send data to 74HC595
24     writeData(num[i]);
25     delay(5);
26     // Clear the display content
27     writeData(0xff);
```

Use cycle command in loop() function to output content to the four 7-segment display.

```

20     for (int i = 0; i < 4; i++) {
...
28 }
```

Repeat this operation continuously.

Verify and upload the code, and then you will see number 0123 shown by 4 digit 7-segment display.



Sketch 21.2.2

Now write code to control 4-digit 7-segment display to display dynamic numbers.

```
1 #include <CBTimer.h>      // Contains CallbackTimerR4 Library
2
3 int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
4 int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
5 int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
6 int comPin[] = {7, 6, 5, 4};// Common pin (anode) of 4 digit 7-segment display
7
8 int second = 0;            // Define variables stored record time
9
10 // Define the encoding of characters 0-F for the common-anode 7-Segment Display
11 byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83,
12   0xc6, 0xa1, 0x86, 0x8e} ;
13
14 void setup() {
15   // Set pins to output
16   pinMode(latchPin, OUTPUT);
17   pinMode(clockPin, OUTPUT);
18   pinMode(dataPin, OUTPUT);
19   for (int i = 0; i < 4; i++) {
20     pinMode(comPin[i], OUTPUT);
21   }
22   static CBTimer timer;
23   timer.begin(1000, callback_func);
24 }
25
26 void loop() {
27   // Calculate the seconds of each digit number
28   byte bit[4];
29   bit[0] = second % 10;
30   bit[1] = second / 10 % 10;
31   bit[2] = second / 100 % 10;
32   bit[3] = second / 1000 % 10;
33   // Show seconds
34   for (int i = 0; i < 4; i++) {
35     // Select a single 7-segment display
36     chooseCommon(i);
37     // Send data to 74HC595
38     writeData(num[bit[3 - i]]);
39     delay(5);
40     // Clear the display content
41     writeData(0xff);
42 }
```

```

42 }
43
44 // the timer interrupt function of FlexiTimer2 is executed every 1s
45 void timerInt() {
46     second++;      // second plus 1
47 }
48
49 void chooseCommon(byte com) {
50     // Close all single 7-segment display
51     for (int i = 0; i < 4; i++) {
52         digitalWrite(comPin[i], LOW);
53     }
54     // Open the selected single 7-segment display
55     digitalWrite(comPin[com], HIGH);
56 }
57
58 void writeData(int value) {
59     // Make latchPin output low level
60     digitalWrite(latchPin, LOW);
61     // Send serial data to 74HC595
62     shiftOut(dataPin, clockPin, LSBFIRST, value);
63     // Make latchPin output high level, then 74HC595 will update the data to parallel
64     // output
65     digitalWrite(latchPin, HIGH);
}

```

Frist set a timer with a cycle of 1s

21	FlexiTimer2::set(1000, timerInt); // configure the timer and interrupt function
22	FlexiTimer2::start(); // start timer

In the timer interrupt function, make the variable second plus 1

45	void timerInt() {
46	second++; // second plus 1
47	}

In function loop(), The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively.

27	byte bit[4];
28	bit[0] = second % 10;
29	bit[1] = second / 10 % 10;
30	bit[2] = second / 100 % 10;
31	bit[3] = second / 1000 % 10;

Then display the value of each bit.

37	writeData(num[bit[3 - i]]);
----	-----------------------------

Verify and upload the code, then you will see the dymaic number shown by 4 digit 7-segment display.

Chapter 22 Stepper Motor

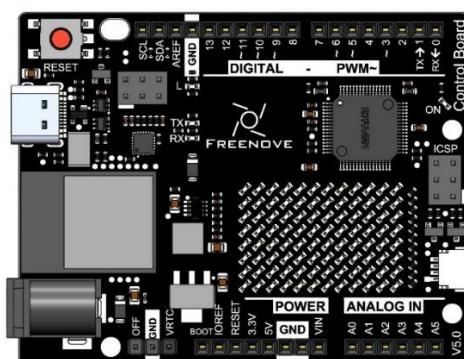
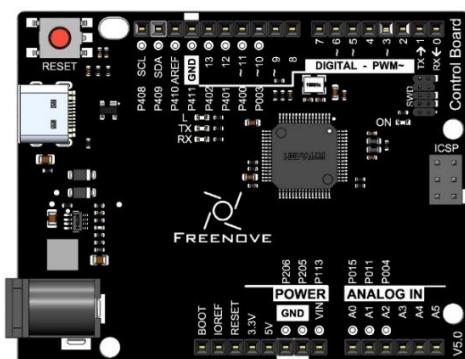
A Stepper motor is a kind of motor that can rotate a certain angle at a time, with which we can achieve mechanical movement at a higher accuracy more easily.

Project 22.1 Drive Stepper Motor

Now, try to drive a stepper motor.

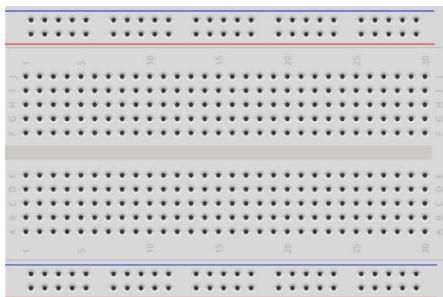
Component List

Control board x1

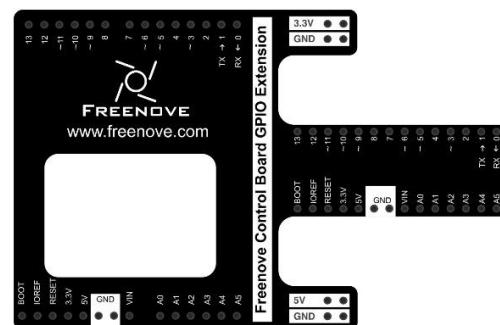


or

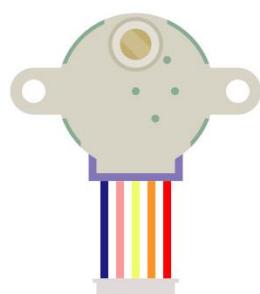
Breadboard x1



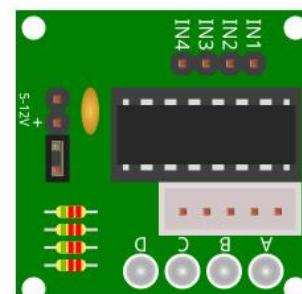
GPIO Extension Board x1

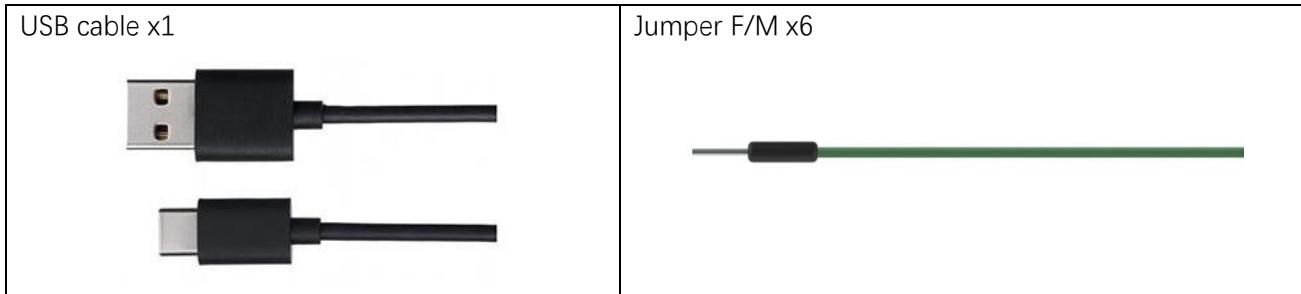


Stepper motor x1



ULN2003 stepper motor driver x1

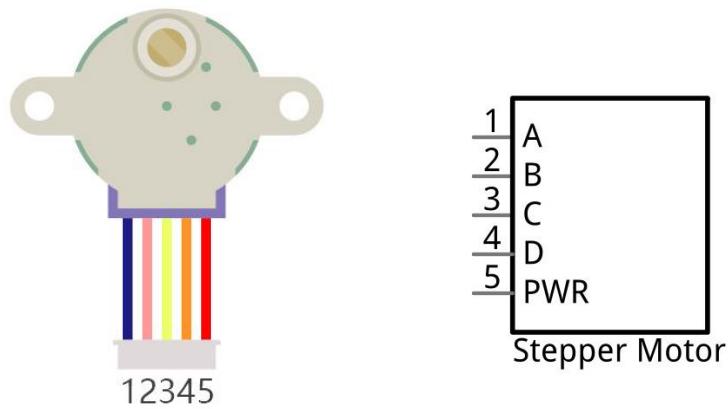




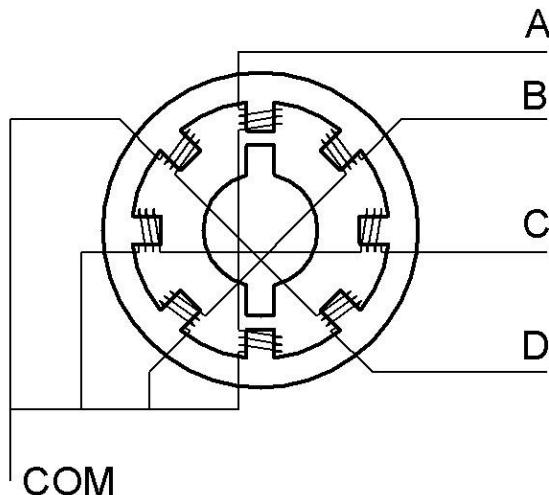
Component Knowledge

Stepper Motor

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 depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:



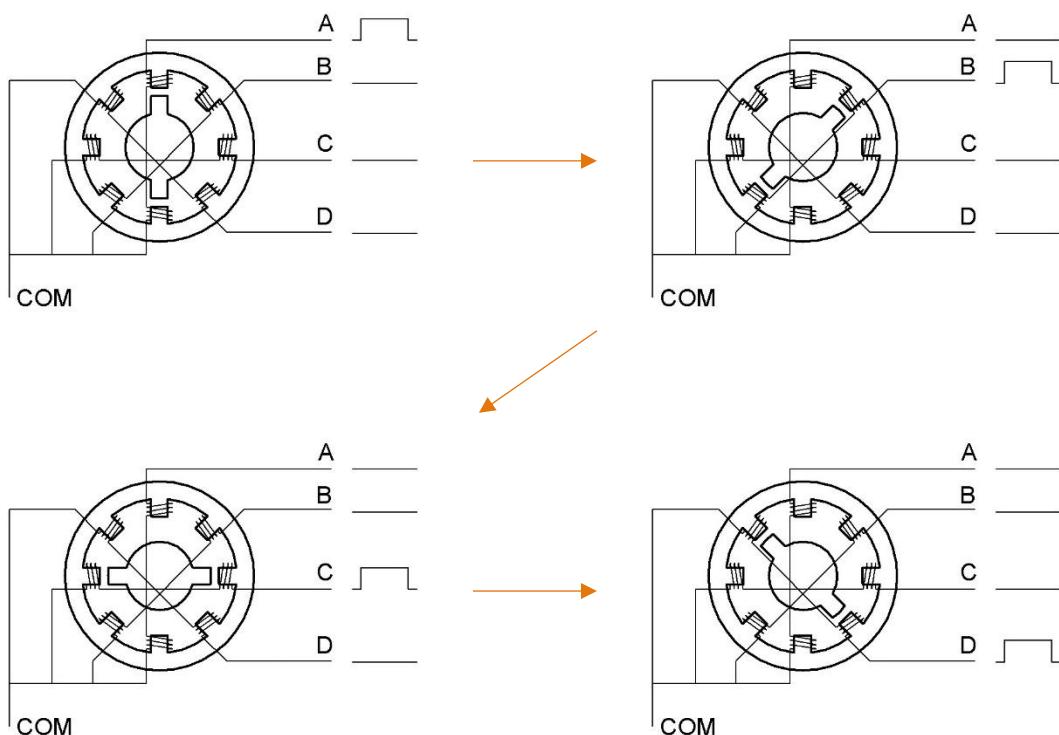
The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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 drive sequence is as follows:



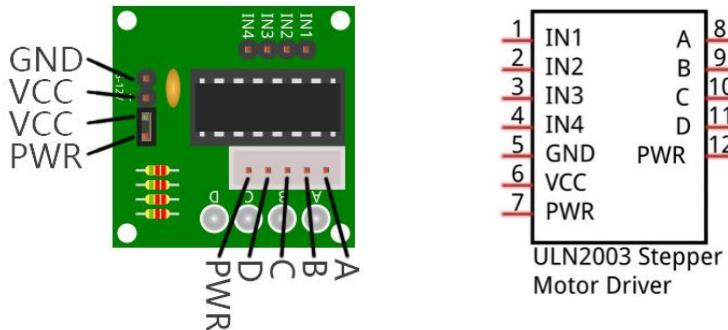
In the sequence above, the Stepper Motor rotates once at a certain angle, which is called a "step". By controlling the number of rotational steps, you can then control the Stepper Motor's rotation angle. By defining the time between two steps, you can control the Stepper Motor's rotation speed.

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. We can use this way to improve the stepper motor's stability and reduce noise at the same time. If you are interested in it, you can learn this method yourself by searching related knowledge.

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

ULN2003 stepper motor driver

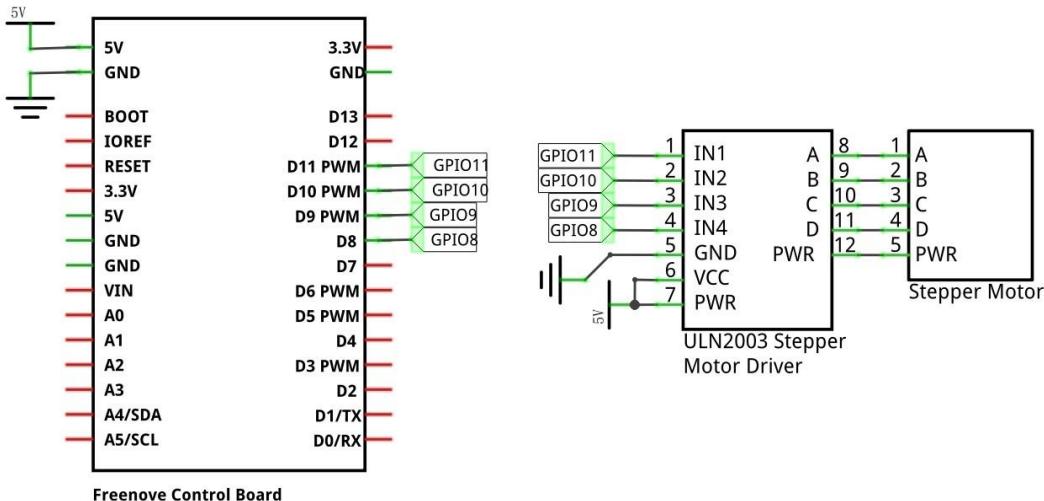
A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals.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



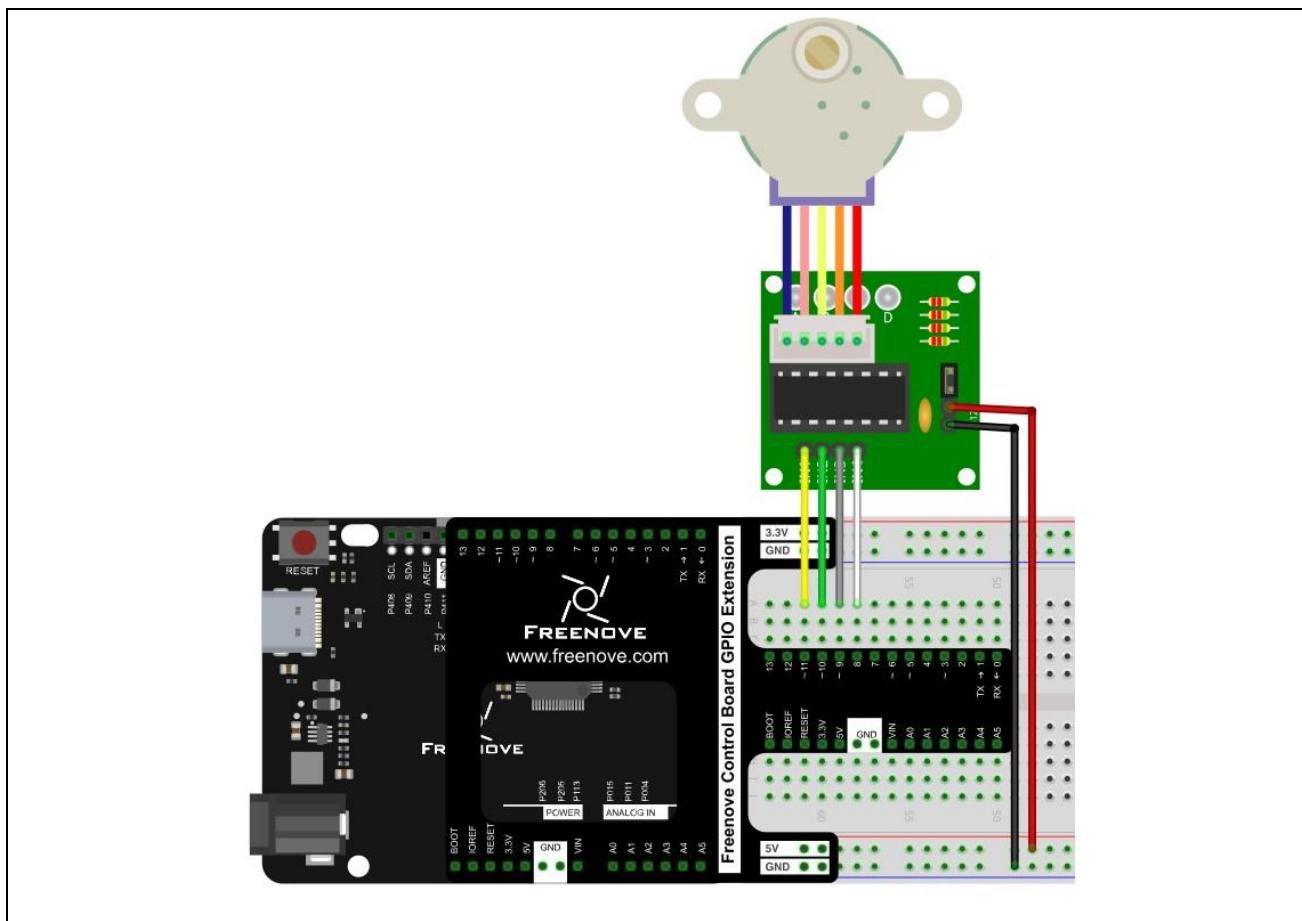
Circuit

Use pin 11, 10, 9, 8 on the control board to control the ULN2003 stepper motor driver, and connect it to the stepper motor.

Schematic diagram



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



Sketch

Sketch 22.1.1

Now write code to control the stepper motor through ULN2003 stepper motor driver.

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

```

40   digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
41 }
42 }
```

In the code, we define a function to make the motor rotate for a step. And the parameter determines the rotation direction of the stepper motor.

```

28 void moveOneStep(bool dir) {
...
42 }
```

A variable is defined in this function and we use four low bits to show the state of 4 ports. These ports are connected in order, so the variable can be assigned to 0x01 and we can use the shifting method to change the bit of the connected port.

```

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 }
```

Then change the state of the port according to the above variables.

```

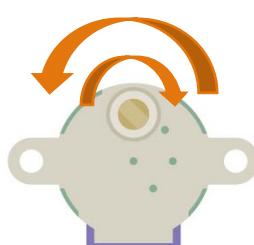
38 // Output signal to each port
39 for (int i = 0; i < 4; i++) {
40     digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
41 }
```

We define a function to control the step motor to rotate several steps and control the direction and speed through parameters. Call it directly in the loop () function.

```

21 void moveSteps(bool dir, int steps, byte ms) {
22     for (int i = 0; i < steps; i++) {
23         moveOneStep(dir); // Rotate one step
24         delay(ms); // Control the speed
25     }
26 }
```

Verify and upload the code, and you will see the step motor rotate a full turn, and then repeat this process in a reverse direction.



Chapter 23 Matrix Keypad

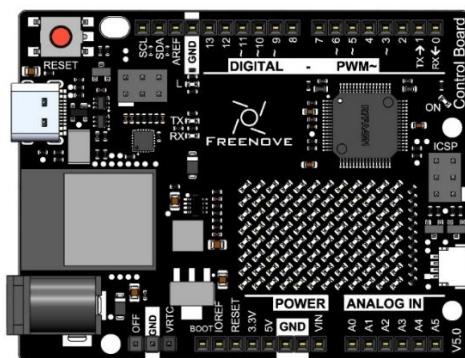
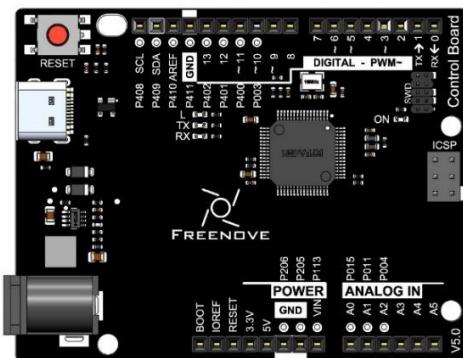
We've already learned and used a push button switch before, now let us try to use a keypad, a device integrated with a number of Push Button Switches as Keys for the purposes of Input.

Project 23.1 Get Input Characters

First, try to understand the keypad, and get the input characters.

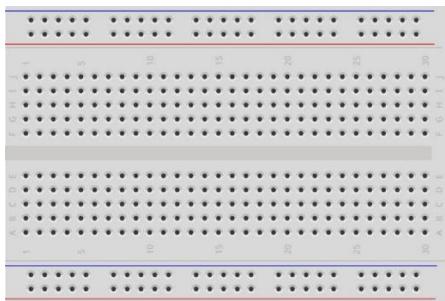
Component List

Control board x1

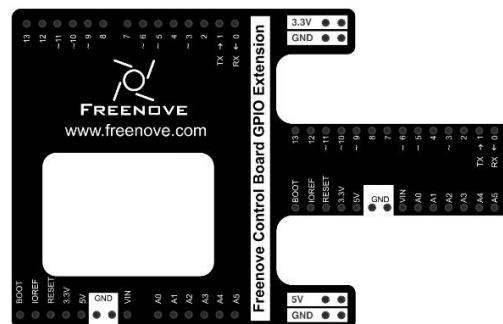


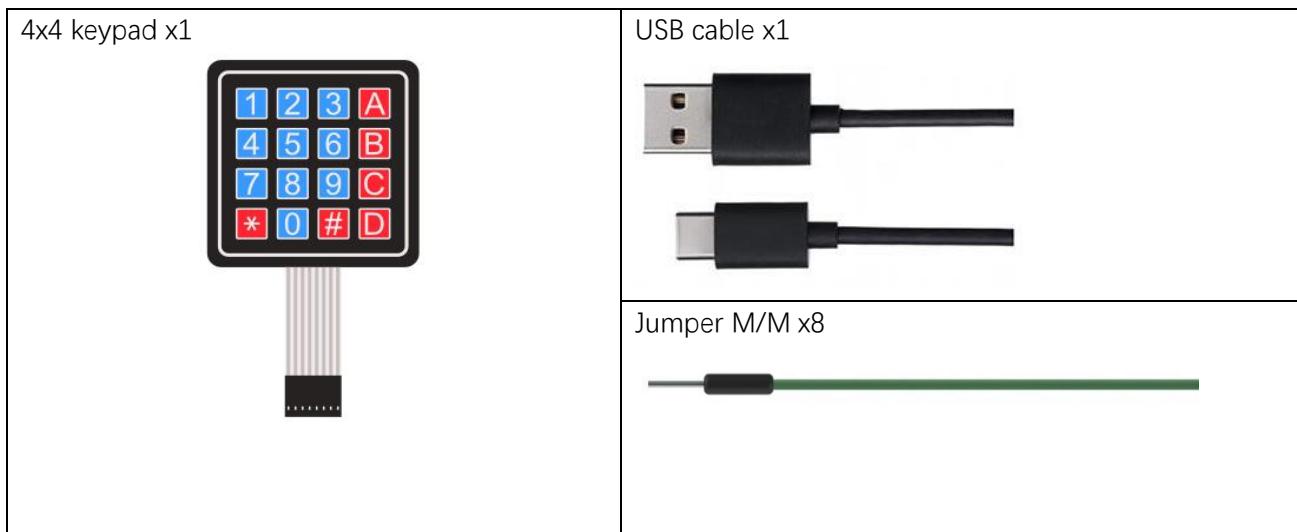
or

Breadboard x1



GPIO Extension Board x1

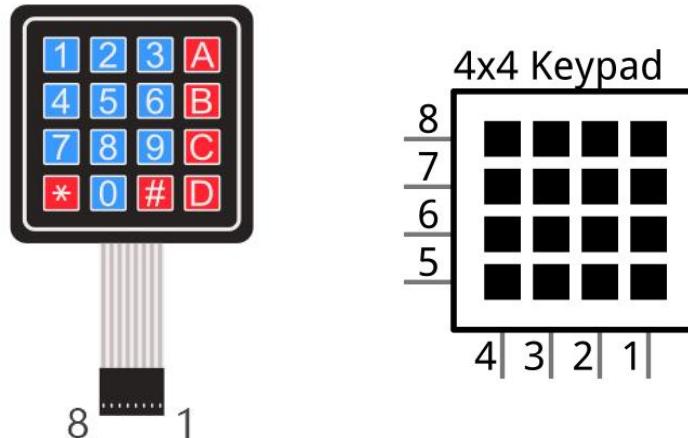




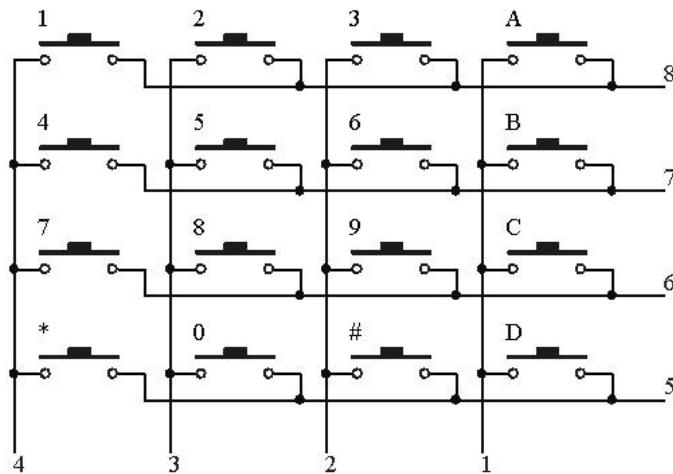
Component Knowledge

4x4 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 (think of this as 16 Push Button Switches in one module):



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.

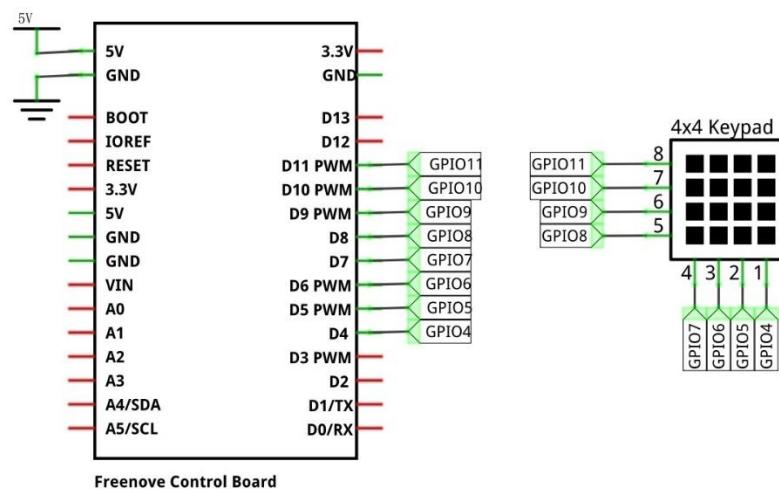


The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key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

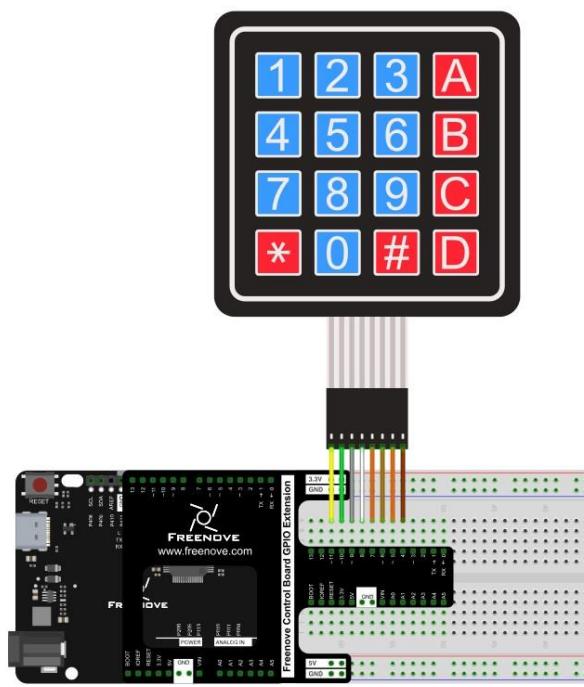
Circuit

Use pin 11-4 on control board to connect 4x4 keypad.

Schematic diagram



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



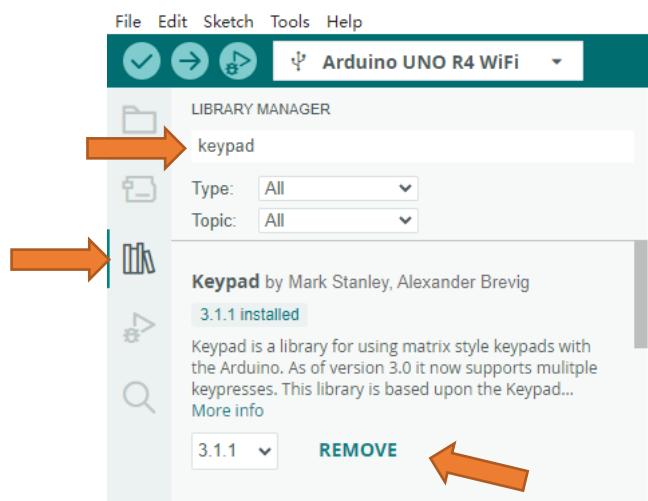
Sketch

Sketch 23.1.1

Before writing code, we need to import the library needed.

Click “Add .ZIP Library...” and then find Keypad.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library can facilitate our operation of keypad.

You can also type "Keypad" into the library search bar to install the library.



Now write the code to obtain the keypad characters, and send them to the serial port.

```

1 #include <Keypad.h>
2
3 // define the symbols on the buttons of the keypad
4 char keys[4][4] = {
5     {'1', '2', '3', 'A'},
6     {'4', '5', '6', 'B'},
7     {'7', '8', '9', 'C'},
8     {'*', '0', '#', 'D'}
9 };
10
11 byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
12 byte colPins[4] = {7, 6, 5, 4}; // connect to the column pinouts of the keypad
13
14 // initialize an instance of class Keypad
15 Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
16
17 void setup() {
18     Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
19 }
```

```

20
21 void loop() {
22     // Get the character input
23     char keyPressed = myKeypad.getKey();
24     // If there is a character input, sent it to the serial port
25     if (keyPressed) {
26         Serial.println(keyPressed);
27     }
28 }
```

In the code, we use a Keypad class provided by the Keypad library to operate the keypad. The following code is to instantiate a keypad object, and the last two parameters represent the number of the row and column of the keypad.

```
6 Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
```

The two-dimensional arrays record the keypad characters, and these characters can be returned when you press the keyboard.

```

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

These two arrays record the row and column's connection pins of keypad.

```
11 byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
12 byte colPins[4] = {7, 6, 5, 4}; // connect to the column pinouts of the keypad
```

Send the input that get from the keyboard to the computer via the serial port in function loop().

```

21 void loop() {
22     // Get the character input
23     char keyPressed = myKeypad.getKey();
24     // If there is a character input, sent it to the serial port
25     if (keyPressed) {
26         Serial.println(keyPressed);
27     }
28 }
```

Verify and upload the code, open the Serial Monitor and press the keypad, and then you will see the characters you press being printed out.

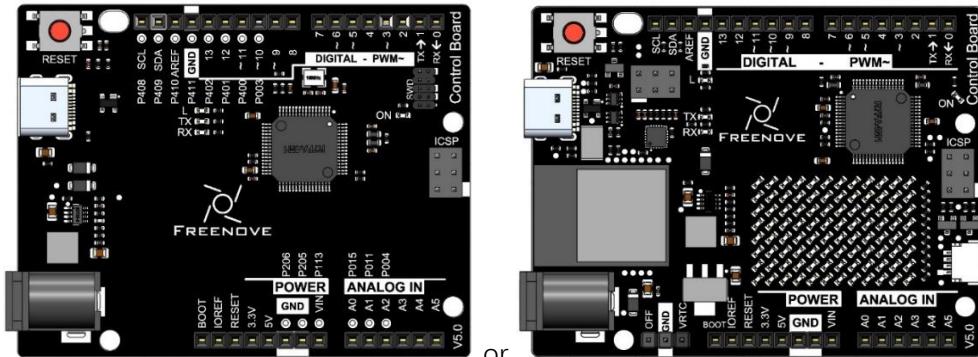
```
1
2
3
A
B
```

Project 23.2 Combination Lock

Now, let us try to use keypad to make a combination lock.

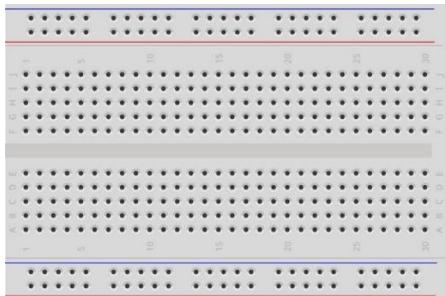
Component List

Control board x1

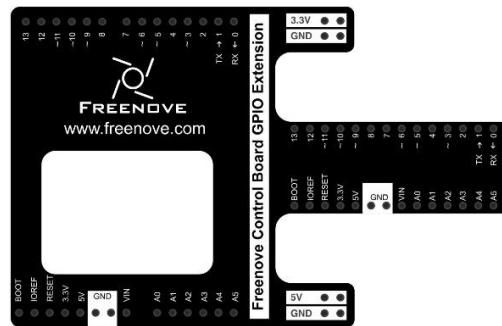


or

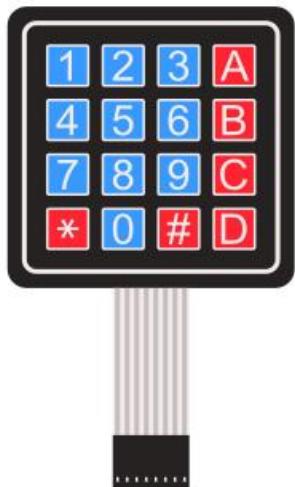
Breadboard x1



GPIO Extension Board x1



4x4 keypad x1

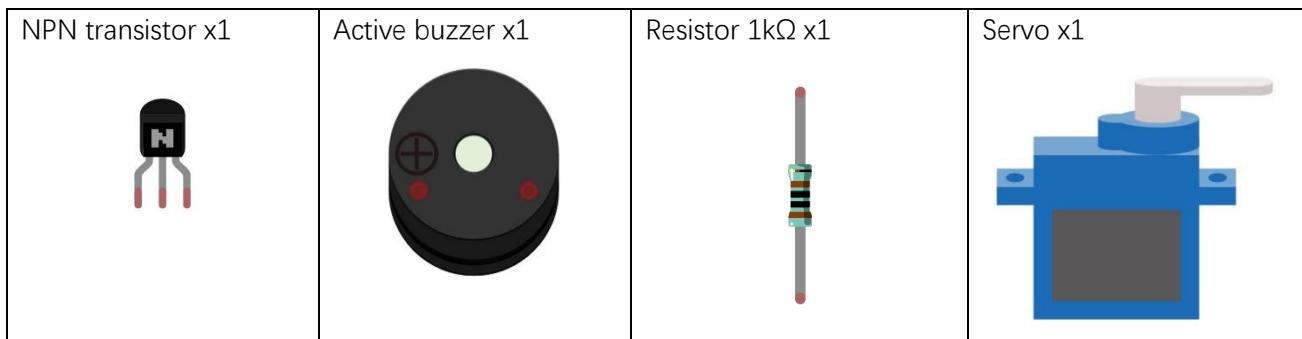


USB cable x1



Jumper M/M x15

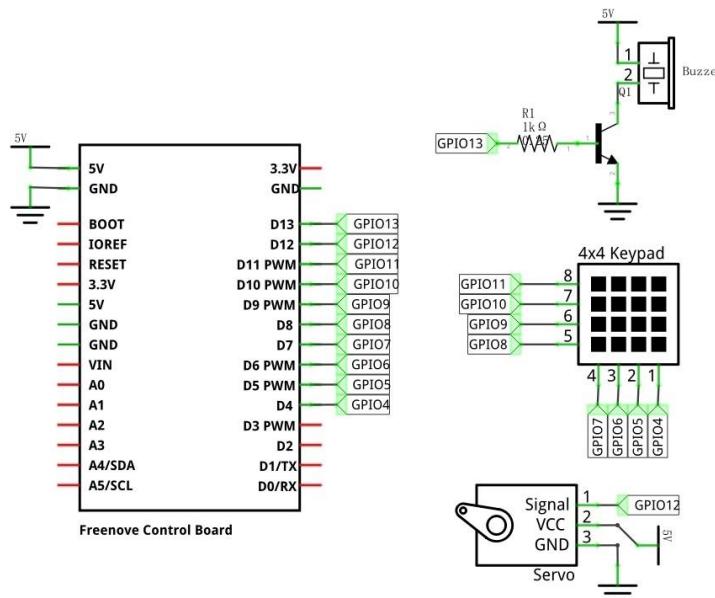




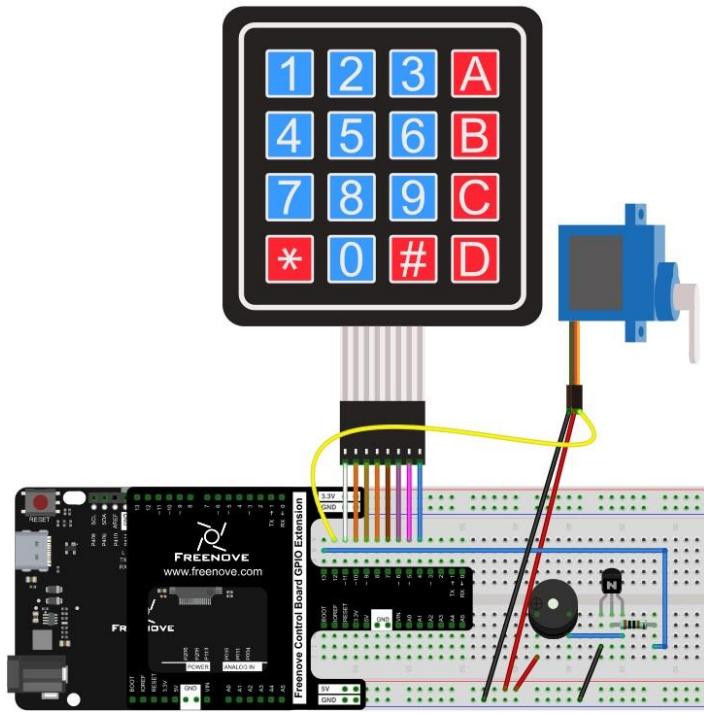
Circuit

Use pin 11-4 on the control board to connect 4x4 keypad, pin 13 to drive buzzer and pin 12 to drive servo. Servos can be used to drive the mechanical switch and turn on the switch when the password is correct.

Schematic diagram



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



Sketch

Sketch 23.2.1

Now write the code to obtain the keypad characters, and compare them with the preset password. If the input is correct, the servo moves, otherwise the buzzer makes an input error alarm.

```

1 #include <Keypad.h>
2 #include <Servo.h>
3
4 // define the symbols on the buttons of the keypad
5 char keys[4][4] = {
6     {'1', '2', '3', 'A'},
7     {'4', '5', '6', 'B'},
8     {'7', '8', '9', 'C'},
9     {'*', '0', '#', 'D'}
10 };
11
12 byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
13 byte colPins[4] = {7, 6, 5, 4}; // connect to the column pinouts of the keypad
14
15 // initialize an instance of class NewKeypad
16 Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
17
18 Servo myservo; // Create servo object to control a servo

```

```
19 int servoPin = 12; // Define the servo pin
20 int buzzerPin = 13; // Define the buzzer pin
21
22 char passWord[] = {'1', '2', '3', '4'}; // Save the correct password
23
24 void setup() {
25     myservo.attach(servoPin); // Attaches the servo on servoPin to the servo object
26     myservo.write(45); // Let the steering gear move to the start position
27     pinMode(buzzerPin, OUTPUT);
28 }
29
30 void loop() {
31     static char keyIn[4]; // Save the input character
32     static byte keyInNum = 0; // Save the number of input characters
33     char keyPressed = myKeypad.getKey(); // Get the character input
34     // Handle the input characters
35     if (keyPressed) {
36         // Make a prompt tone each time press the key
37         digitalWrite(buzzerPin, HIGH);
38         delay(100);
39         digitalWrite(buzzerPin, LOW);
40         // Save the input characters
41         keyIn[keyInNum++] = keyPressed;
42         // Judge the correctness after input
43         if (keyInNum == 4) {
44             bool isRight = true; // Save password is correct or not
45             for (int i = 0; i < 4; i++) { // Judge each character of the password is correct
46                 // or not
47                 if (keyIn[i] != passWord[i])
48                     isRight = false; // Mark wrong password if there is any wrong
49                 character
50             }
51             if (isRight) { // If the input password is right
52                 myservo.write(135); // Open the switch
53                 delay(2000); // Delay a period of time
54                 myservo.write(45); // Close the switch
55             }
56             else { // If the input password is wrong
57                 digitalWrite(buzzerPin, HIGH); // Make a wrong password prompt tone
58                 delay(500);
59                 digitalWrite(buzzerPin, LOW);
60             }
61         }
62         keyInNum = 0; // Reset the number of the input characters to 0
63     }
64 }
```

```
61 }  
62 }
```

Based on the last section, this code adds a comparison function, the corresponding action will be different whenever the password is right or wrong.

First we define a correct password with four characters.

```
22 char passWord[] = {'1', '2', '3', '4'}; // save the right password
```

Make a prompt sound every time when a key is pressed and save the pressed characters.

```
35 if (keyPressed) {  
36     // Make a prompt tone each time press the key.  
37     digitalWrite(buzzerPin, HIGH);  
38     delay(100);  
39     digitalWrite(buzzerPin, LOW);  
40     // Save the input characters  
41     keyIn[keyInNum++] = keyPressed;  
...  
61 }
```

Compare with the preset password after 4 characters are input and adopt the corresponding operation.

```
43 if (keyInNum == 4) {  
44     bool isRight = true;           // Sav password is correct or not  
45     for (int i = 0; i < 4; i++) {  // Judge each character of the password is correct  
or not  
46         if (keyIn[i] != passWord[i])  
47             isRight = false;        // Mark wrong passageword if there is any wrong  
character.  
48     }  
49     if (isRight) {                // If the input password is right  
50         myservo.write(135);       // Open the switch  
51         delay(2000);            // Delay a period of time  
52         myservo.write(45);       // Close the switch  
53     }  
54     else {                      // If the input password is wrong  
55         digitalWrite(buzzerPin, HIGH); // Make a wrong password prompt tone  
56         delay(500);  
57         digitalWrite(buzzerPin, LOW);  
58     }  
59     keyInNum = 0; // Reset the number of the input characters to 0.  
60 }
```

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

Chapter 24 Vibration Switch

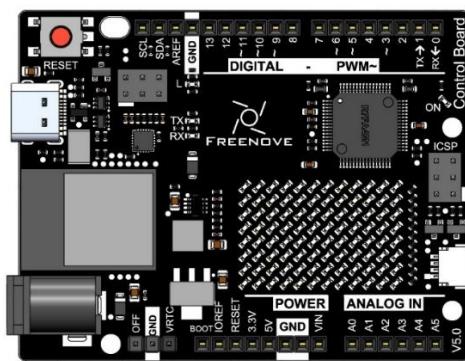
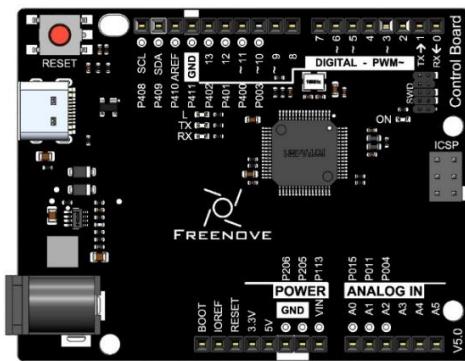
Vibration switch is a component that can detect vibration. We will use this sensor later.

Project 24.1 Detect Vibration

Now let us try to use vibration switch to detect vibration.

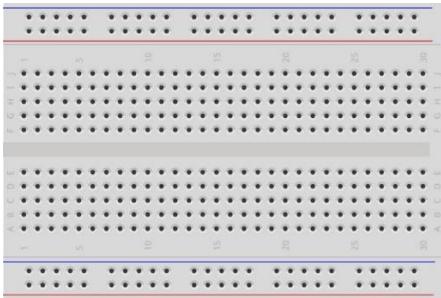
Component List

Control board x1

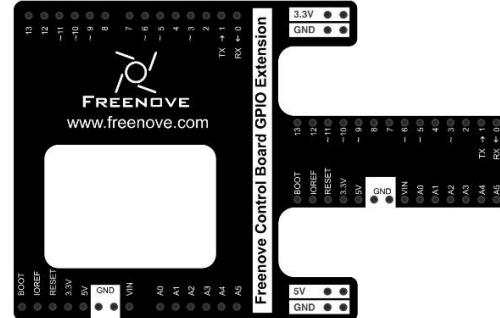


or

Breadboard x1



GPIO Extension Board x1

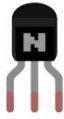


USB cable x1



Jumper M/M x6

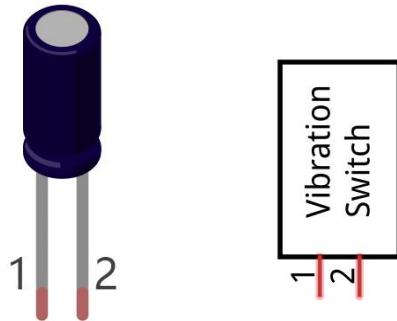


NPN transistor x1	Active buzzer x1	Resistor 1kΩ x1	Vibration switch x1
			

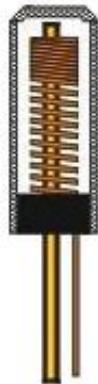
Component Knowledge

Vibration Switch

Vibration switches is a kind of sensor that can detect vibration. When the vibration amplitude is greater than the critical value, two pins of the vibration switch will be switched on.



The internal circuit of the Vibration switch is as below, one of the pins is connected with a metal bar and the other is connected with spring. The spring will swing when a vibration occurs. When the vibration is strong enough, the spring will contact with the metal bar to make the two pins connected with each other.



Circuit Knowledge

Digital pins with interrupts

Some of the control board digital pins can be configured to interrupt mode, which can trigger interrupt event and execute interrupt function.

Pin 2 and 3 of the control board can be configured to interrupt mode. Conditions that trigger interrupt can

be configured to:

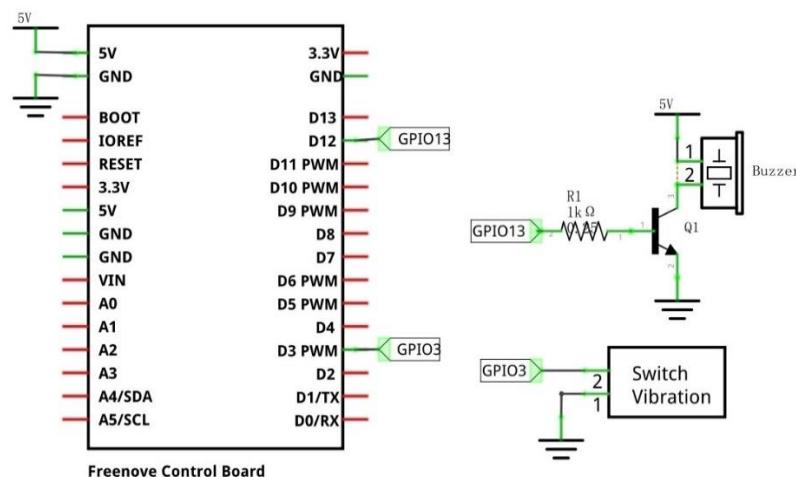
Condition	Function
LOW	to trigger the interrupt whenever the pin is low
CHANGE	to trigger the interrupt whenever the pin changes value
RISING	to trigger when the pin goes from low to high
FALLING	to trigger for when the pin goes from high to low

Interrupts are useful for making things happen automatically in microcontroller programs, and they can help solve timing problems. Ideal tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

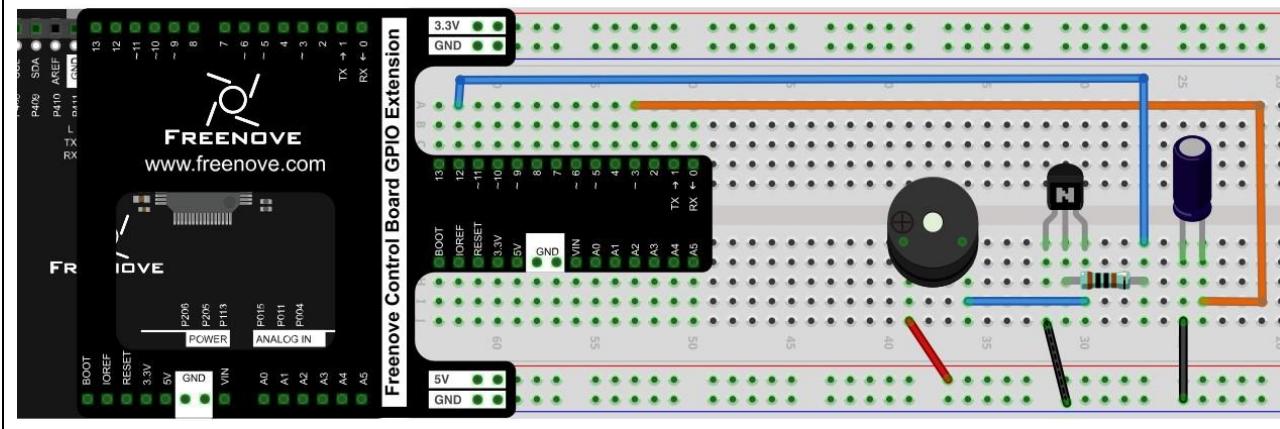
Circuit

Use pin 3 on the control board to connect vibration switch, and pin 13 to drive buzzer.

Schematic diagram



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



Sketch

Sketch 24.1.1

Now write code to detect whether the vibration switch is conducted and whether it makes a buzzer sound when conducted.

```

1 int buzzerPin = 13; // Define the buzzer pin
2 int switchPin = 3; // Define the vibration switch pin, which can cause interrupt
3
4 bool isVibrate = false;
5
6 void setup() {
7     pinMode(buzzerPin, OUTPUT); // Set the buzzer pin to output mode
8     pinMode(switchPin, INPUT_PULLUP); // Set the vibration switch pin to pull up input mode
9     // Set interrupt, if vibration switch pin change from high level to low level, vibrate
function will be called
10    attachInterrupt(digitalPinToInterrupt(switchPin), vibrate, FALLING);
11 }
12
13 void loop() {
14     if (isVibrate) { // If the interrupt function is triggered
15         isVibrate = false; // Marked no trigger again
16         digitalWrite(buzzerPin, HIGH); // Open the buzzer
17         delay(50); // Delay for a period of time
18     }
19     else // Else close the buzzer
20         digitalWrite(buzzerPin, LOW);
21 }
22
23 void vibrate() {
24     isVibrate = true; // Marked as the trigger
25 }
```

This code configures the pin that is connected to vibration switch to be triggered by falling edge, that is, FALLING mode.

10	attachInterrupt(digitalPinToInterrupt(switchPin), vibrate, FALLING);
----	--

attachInterrupt(interrupt, ISR, mode);
--

This function is used to configure the digital pin's interrupt mode. Each parameter of the function is as follows:

interrupt: the number of the interrupt (int). Normally you should use digitalPinToInterruption(pin) to translate the actual digital pin to the specific interrupt number.

ISR: the ISR to be called when an interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered.

And we configure the pins that connected with vibration switch into pull up input mode. This way can ensure a high level of vibration switch when not connected, and low level when connected. So it can cause interrupt.

```
8     pinMode(switchPin, INPUT_PULLUP); // Set the vibration switch pin to pull up input mode
```

The following is the interrupt function, and it will be executed when the interrupt is triggered.

```
23 void vibrate() {  
24     isVibrate = true; // Marked as the trigger  
25 }
```

Interrupt function should keep short, so we use a variable "isVibrate" to record whether the interrupt is triggered, and dispose it in the loop() function. And the buzzer will be connected if the interrupt is triggered.

```
14     if (isVibrate) { // If the interrupt function is triggered.  
15         isVibrate = false; // Marked no trigger again  
16  
17         digitalWrite(buzzerPin, HIGH); // Open the buzzer  
18         delay(50); // Delay for a period of time  
19     }  
20     else // Else close the buzzer  
21         digitalWrite(buzzerPin, LOW);
```

Verify and upload the code and tap on the vibration switch, and then the buzzer will make a sound.

Chapter 25 Infrared Remote

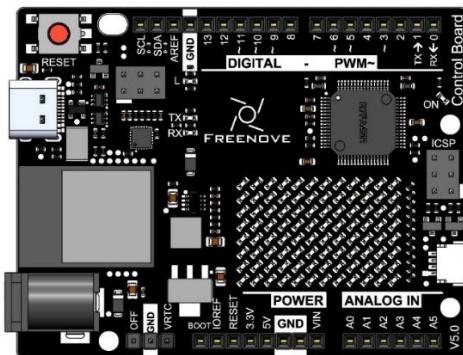
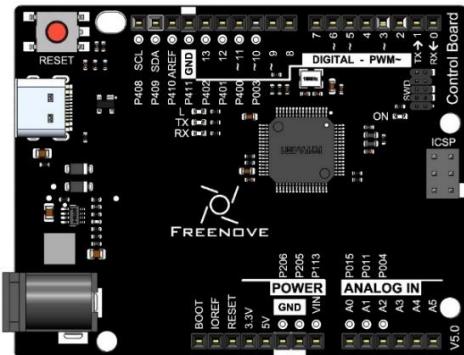
In this chapter, we'll learn how to use an infrared remote control, and control a LED.

Project 25.1 Infrared Remote Control

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

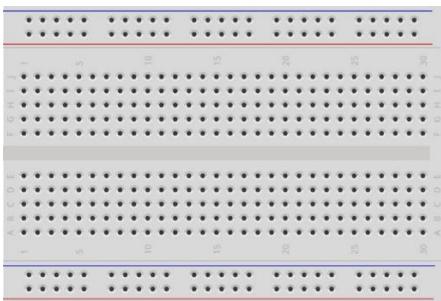
Component List

Control board x1

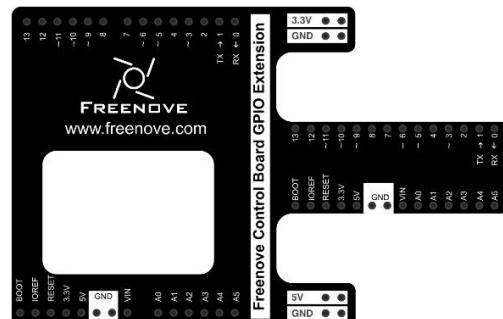


or

Breadboard x1



GPIO Extension Board x1

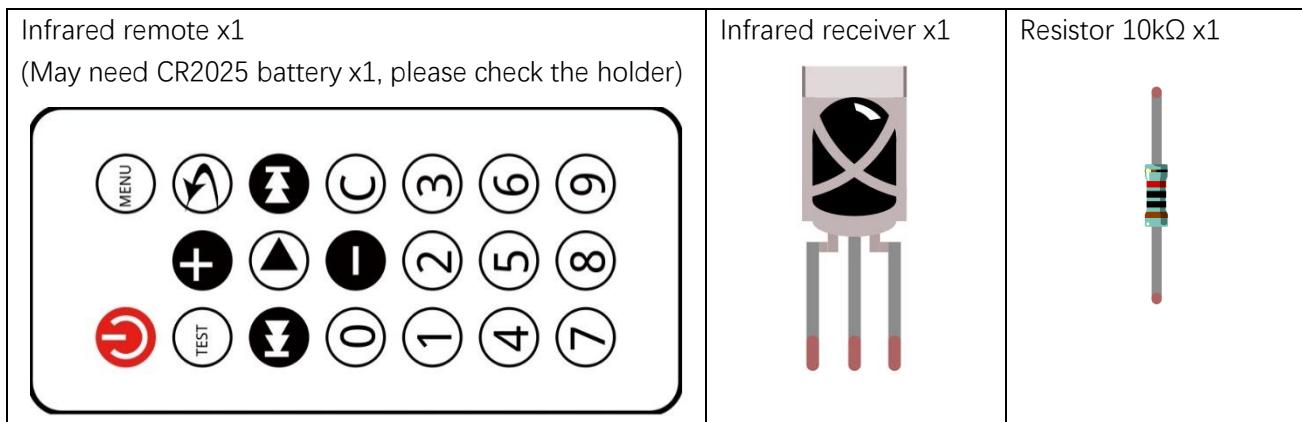


USB cable x1



Jumper M/M x4

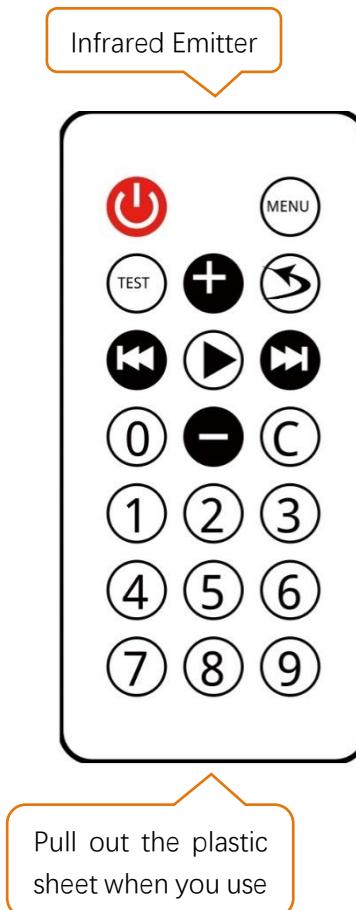




Component Knowledge

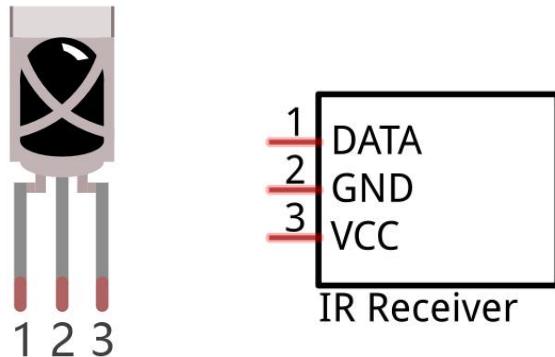
Infrared remote

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 with different encoding. Infrared remote control technology is widely used in household appliances, such as TV, air conditioning, etc. Thus, it makes it possible for you to switch TV programs and adjust the temperature of the air conditioning away from them. The remote control we use is shown below:



Infrared receiver

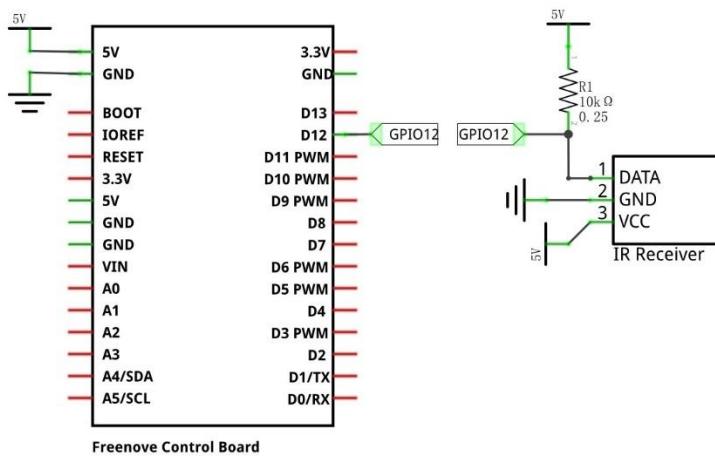
Infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.



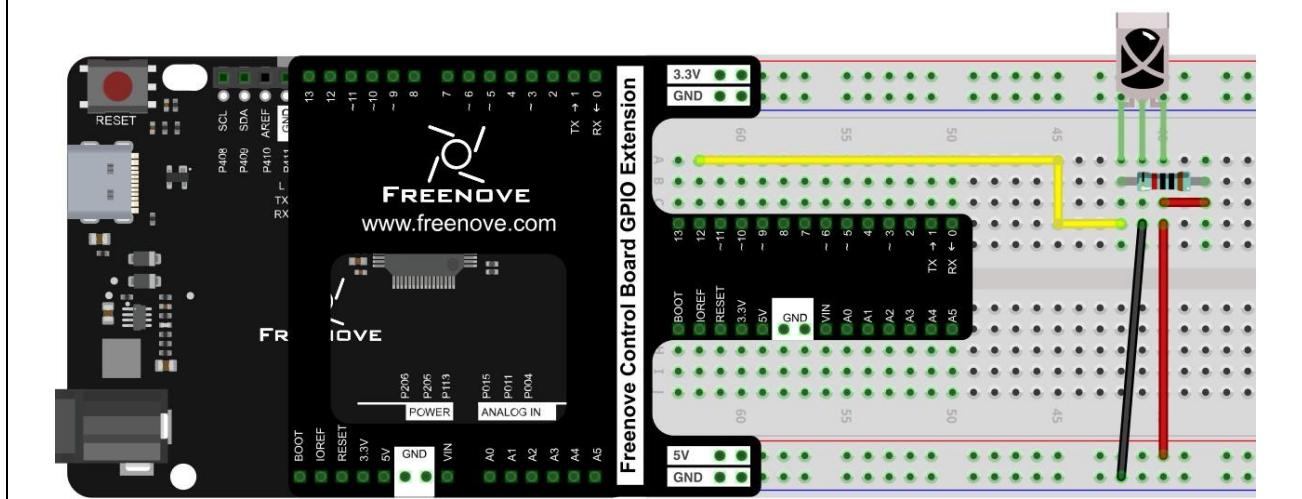
Circuit

Use pin 12 on the control board to connect IR receiver.

Schematic diagram



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



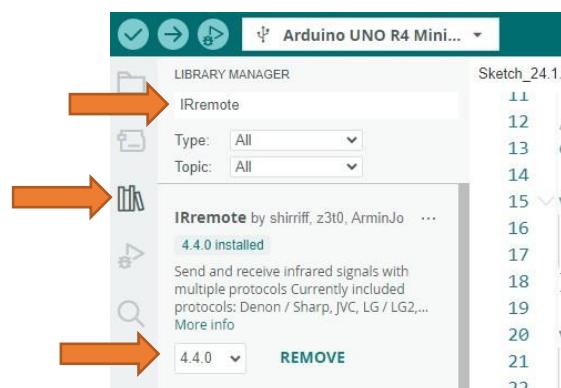
Sketch

Sketch 25.1.1

Before writing code, we need to import the library needed.

Click "Add .ZIP Library..." and then find IRremote.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library makes it easy to control IR receiver.

You can also type "IRremote" into the library search bar to install the library.



Now, write code to get the command sent from IR remote control, and send it to the serial port.

```

1 // Include the necessary libraries
2 #include <IRremote.hpp>
3
4 // Define the pin numbers for the IR receiver
5 const int IR_RECEIVE_PIN = 12;
6
7 void setup() {
8     Serial.begin(9600);                                // Start serial communication at
9     9600 baud rate
10    IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK); // Start the IR receiver
11 }
12
13 void loop() {
14     // Check if there is any incoming IR signal
15     if (IrReceiver.decode()) {
16         // Serial.println(IrReceiver.decodedIRData.command, HEX); // Print the command in
17         // hexadecimal format
18         // Serial.println(IrReceiver.decodedIRData.decodedRawData, HEX); // Map and print the
19         // decoded IR signal to corresponding key value
20         Serial.println(decodeKeyValue(IrReceiver.decodedIRData.command)); // Map and print the
21         // decoded IR signal to corresponding key value
22
23         IrReceiver.resume(); // Enable receiving of the next value
24     }
25 }
```

```
26 // Function to map received IR signals to corresponding keys
27 String decodeKeyValue(long result) {
28     // Each case corresponds to a specific IR command
29     switch (result) {
30         case 69:
31             return "POWER";
32         case 71:
33             return "MENU";
34         case 68: // Receive the number 'TEST'
35             return "TEST";
36         case 64: // Receive the number '+'
37             return "+";
38         case 67:
39             return "Back";
40         case 7: // Receive the number '|<<|
41             return "|<<";
42         case 21: // Receive the number '▶'
43             return "▶";
44         case 9: // Receive the number '>>|'
45             return ">>|";
46         case 22: // Receive the number '0'
47             return "0";
48         case 25: // Receive the number '-'
49             return "-";
50         case 13: // Receive the number 'C'
51             return "C";
52         case 12: // Receive the number '1'
53             return "1";
54         case 24: // Receive the number '2'
55             return "2";
56         case 94: // Receive the number '3'
57             return "3";
58         case 8: // Receive the number '4'
59             return "4";
60         case 28: // Receive the number '5'
61             return "5";
62         case 90: // Receive the number '6'
63             return "6";
64         case 66: // Receive the number '7'
65             return "7";
66         case 82: // Receive the number '8'
67             return "8";
68         case 74: // Receive the number '9'
```

```

70     return "9";
71     default:
72         break;
73     }
74 }
```

First, include header file. Each time you use the infrared sensor, you need to include the header file at the beginning of the program.

```
2 #include <IRremote.hpp>
```

Second, define an infrared receive pin and the infrared sensor is initialized.

```

4 // Define the pin numbers for the IR receiver
5 const int IR_RECEIVE_PIN = 12;
...
10 IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK); // Start the IR receiver
```

The function decodeKeyValue() is used to map the received IR signal to the corresponding key.

```

27 // Function to map received IR signals to corresponding keys
28 String decodeKeyValue(long result) {
29     // Each case corresponds to a specific IR command
30     switch (result) {
31         case 69:
32             return "POWER";
33         case 71:
34             return "MENU";
35         case 68: // Receive the number 'TEST'
36             return "TEST";
37         case 64: // Receive the number '+'
38             return "+";
39         case 67:
40             return "Back";
41         case 7: // Receive the number '|<<'
42             return "|<<";
43         case 21: // Receive the number '▶'
44             return "▶";
45         case 9: // Receive the number '|>>'
46             return ">>|";
47         case 22: // Receive the number '0'
48             return "0";
49         case 25: // Receive the number '-'
50             return "-";
51         case 13: // Receive the number 'C'
52             return "C";
53         case 12: // Receive the number '1'
54             return "1";
55         case 24: // Receive the number '2'
56             return "2";
```

```

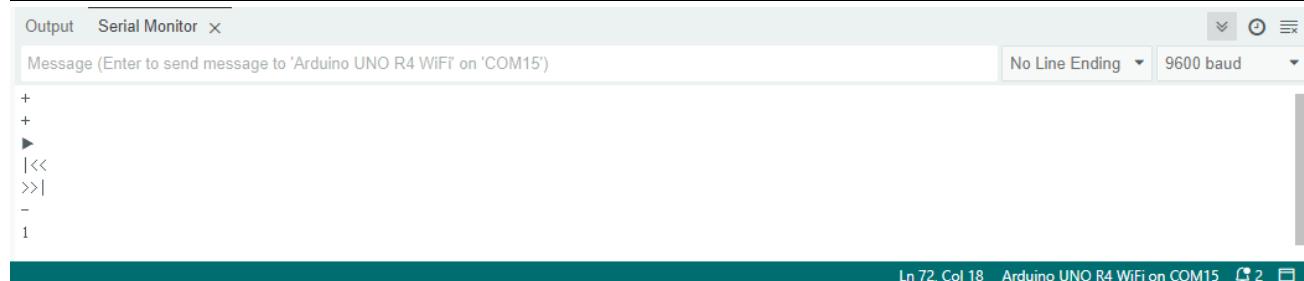
57     case 94: // Receive the number '3'
58         return "3";
59     case 8: // Receive the number '4'
60         return "4";
61     case 28: // Receive the number '5'
62         return "5";
63     case 90: // Receive the number '6'
64         return "6";
65     case 66: // Receive the number '7'
66         return "7";
67     case 82: // Receive the number '8'
68         return "8";
69     case 74: // Receive the number '9'
70         return "9";
71     default:
72         break;
73     }
74 }
```

Finally, **IrReceiver.decode()** is used to determine whether an infrared signal has been received, returning true/1 if an infrared signal has been received, or false/0 if no infrared signal has been received; If an infrared signal is received, the received infrared coded value is printed through the serial port.

Please note that **IrReceiver.resume()** must be applied to release the infrared receiver function each time data are received. Otherwise, the infrared receiver function can only be used once and data cannot be received next time.

```

15     if (IrReceiver.decode()) {
16         // Serial.println(IrReceiver.decodedIRData.command, HEX); // Print the command in
17         // hexadecimal format
18         // Serial.println(IrReceiver.decodedIRData.decodedRawData, HEX); // Map and print the
19         // decoded IR signal to corresponding key value
20         Serial.println(decodeKeyValue(IrReceiver.decodedIRData.command)); // Map and print the
21         // decoded IR signal to corresponding key value
22
23         IrReceiver.resume(); // Enable receiving of the next value
24     }
```

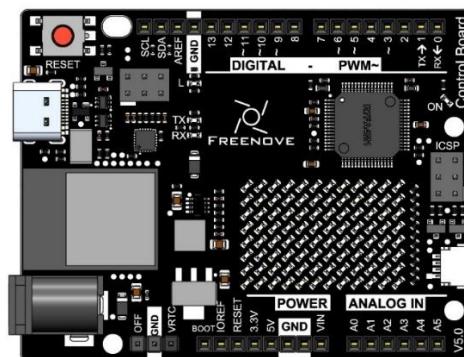
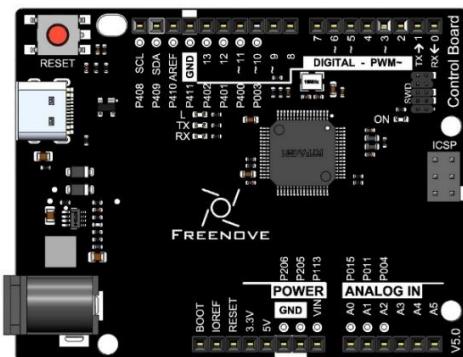


Project 25.2 Control LED through Infrared Remote

Now, let us try to control a LED through infrared remote.

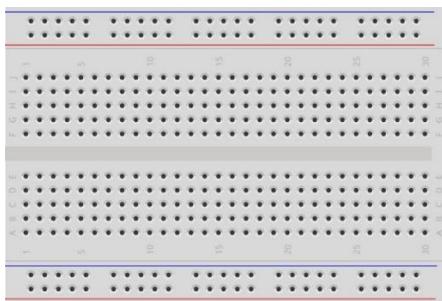
Component List

Control board x1

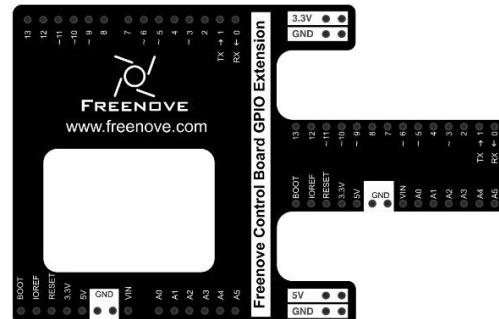


or

Breadboard x1



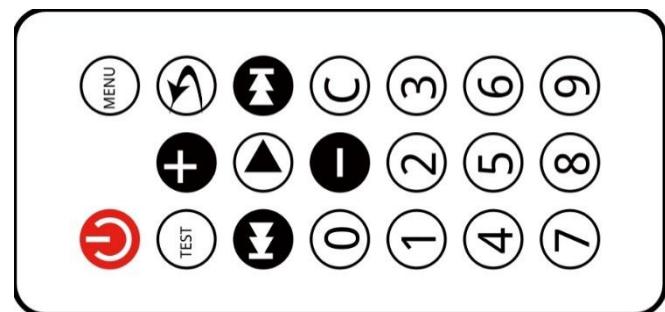
GPIO Extension Board x1



USB cable x1

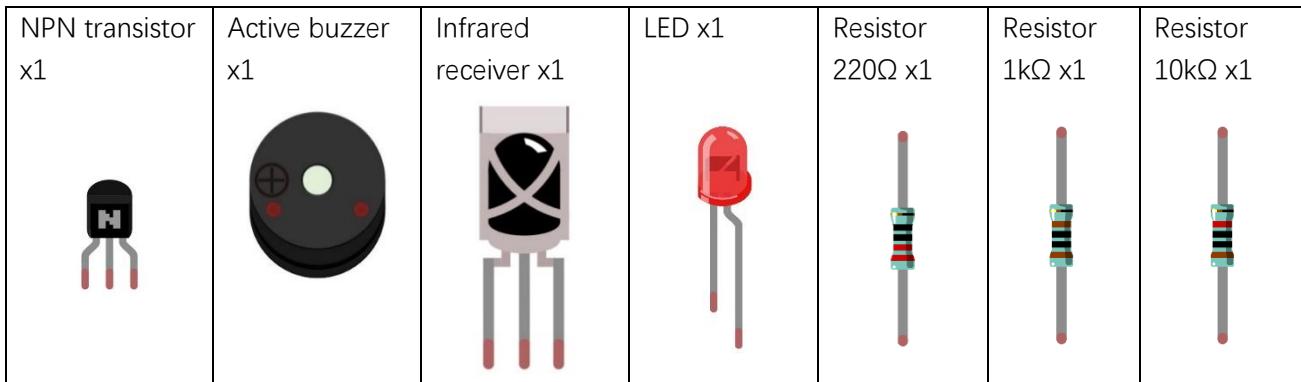


Infrared remote x1



Jumper M/M x11

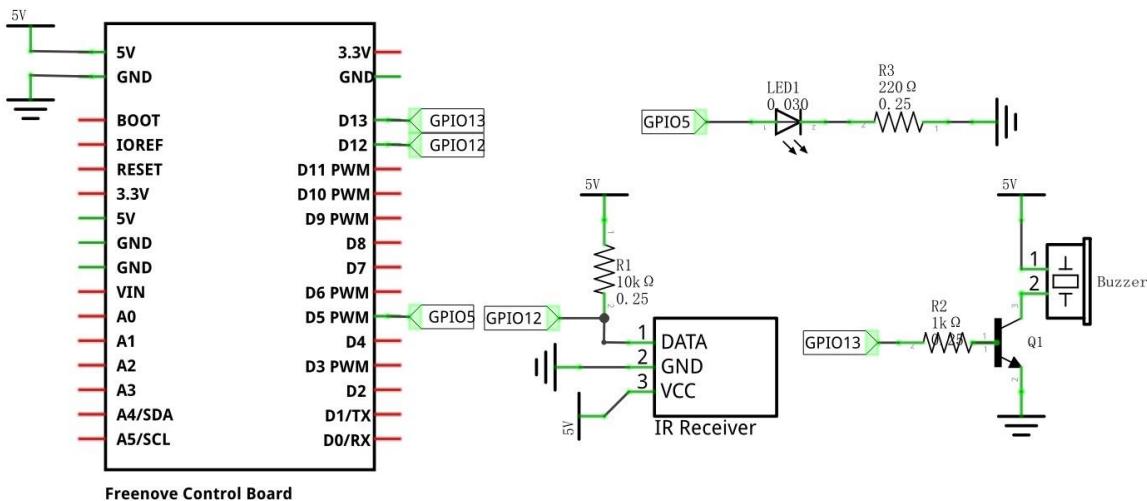




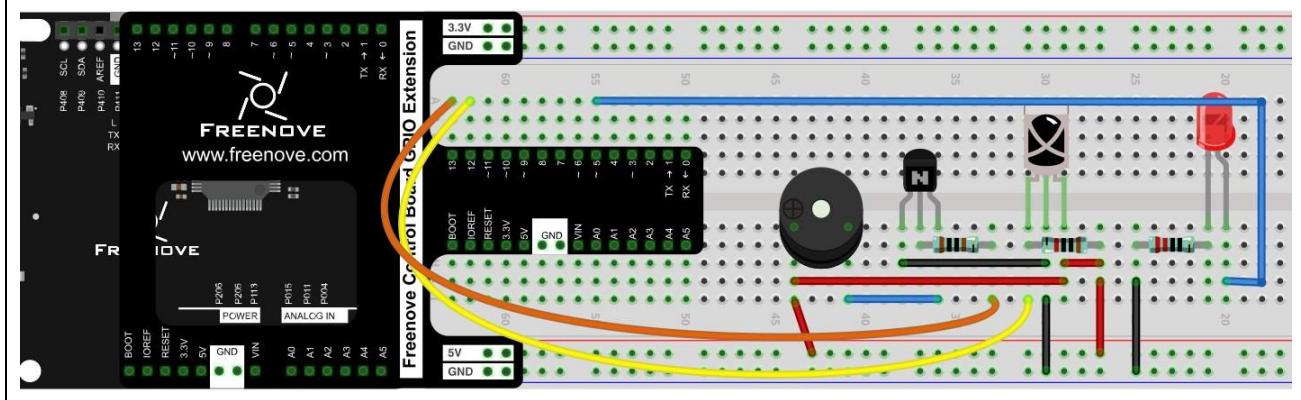
Circuit

Connect pin 12 on the control board to IR receiver to simulate a desk lamp. And drive buzzer through pin 13, drive LED through pin 5.

Schematic diagram



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



Sketch

Sketch 25.2.1

Now, write code to get the commands sent from IR remote, and control the LED light on/off or emit light with different brightness, and control the buzzer to generate a confirmation sound when it receives the command.

```
1 #include <IRremote.hpp>
2
3 // Define the pin numbers for the IR receiver
4 const int IR_RECEIVE_PIN = 12;
5
6 decode_results results; // Create a decoding results class object
7
8 int ledPin = 5;           // the number of the LED pin
9 int buzzerPin = 13;       // the number of the buzzer pin
10
11 void setup()
12 {
13     Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
14     Serial.println("UNO is ready!"); // Print the string "UNO is ready!"
15     IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK); // Start the IR receiver
16     pinMode(ledPin, OUTPUT); // set LED pin into output mode
17     pinMode(buzzerPin, OUTPUT); // set buzzer pin into output mode
18 }
19
20 void loop() {
21     if (IrReceiver.decode()) {
22         Serial.println(IrReceiver.decodedIRData.command); // Print out the decoded results
23         handleControl(IrReceiver.decodedIRData.command); // Handle the commands from remote
24         control
25         IrReceiver.resume(); // Enable receiving of the next value
26     }
27 }
28
29 void handleControl(unsigned long value) {
30     // Make a sound when it receives commands
31     digitalWrite(buzzerPin, HIGH);
32     delay(100);
33     digitalWrite(buzzerPin, LOW);
34     // Handle the commands
35     switch (value) {
36         case 22: // Receive the number '0'
37             analogWrite(ledPin, 0); // Turn off LED
38             break;
```

```

39   case 12: // Receive the number '1'
40     analogWrite(ledPin, 7); // Dimmest brightness
41     break;
42   case 24: // Receive the number '2'
43     analogWrite(ledPin, 63); // Medium brightness
44     break;
45   case 94: // Receive the number '3'
46     analogWrite(ledPin, 255); // Strongest brightness
47     break;
48 }
49 }
```

Based on the last section, we add some new functions: control LED and buzzer.

We define a function that is used to handle the received commands. When this function is executed, make the buzzer beep first, then output PWM signals with different duty cycle to the pin connected to LED according to the receiving commands

```

29 void handleControl(unsigned long value) {
30   // Make a sound when it receives commands
31   digitalWrite(buzzerPin, HIGH);
32   delay(100);
33   digitalWrite(buzzerPin, LOW);
34   // Handle the commands
35   switch (value) {
36     case 22: // Receive the number '0'
37       analogWrite(ledPin, 0); // Turn off LED
38       break;
39     case 12: // Receive the number '1'
40       analogWrite(ledPin, 7); // Dimmest brightness
41       break;
42     case 24: // Receive the number '2'
43       analogWrite(ledPin, 63); // Medium brightness
44       break;
45     case 94: // Receive the number '3'
46       analogWrite(ledPin, 255); // Strongest brightness
47       break;
48   }
49 }
```

Each time when the command is received, the function above will be called in the loop() function.

```

20 void loop() {
21   if (IrReceiver.decode()) {
22     Serial.println(IrReceiver.decodedIRData.command); // Print out the decoded results
23     handleControl(IrReceiver.decodedIRData.command); // Handle the commands from remote
24     control
25     IrReceiver.resume(); // Enable receiving of the next value
26   }
```

27 }

Verify and upload the code, press the button '0', '1', '2', '3' on IR remote, and then you can see LED emit light with different brightness, and the buzzer will start beeping when it receives the signal.

Chapter 26 Temperature & Humidity Sensor

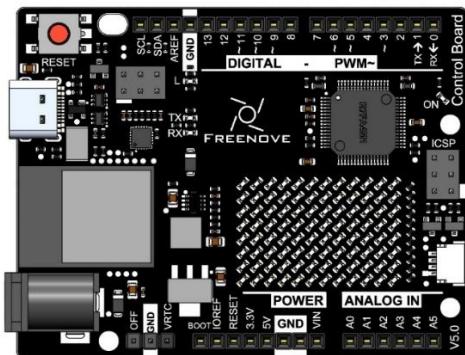
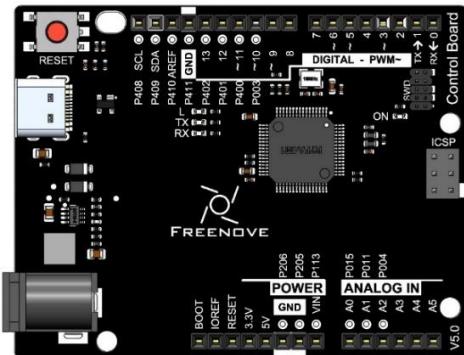
In this chapter, we will learn how to use a sensor that can detect temperature and humidity.

Project 26.1 Temperature & Humidity Sensor

Now, let's try to get the temperature and humidity value of the current environment.

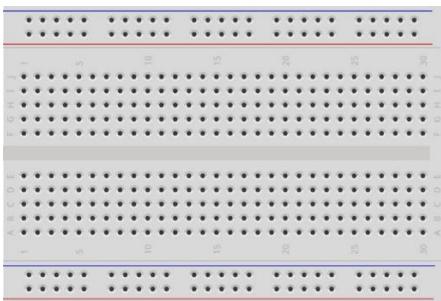
Component List

Control board x1

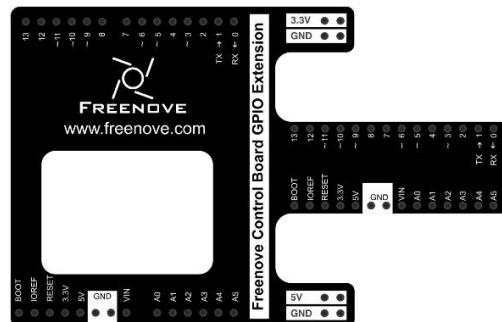


or

Breadboard x1



GPIO Extension Board x1



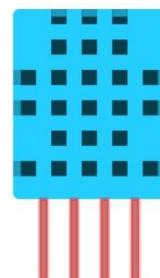
USB cable x1



Resistor 10kΩ x1



DHT11 x1



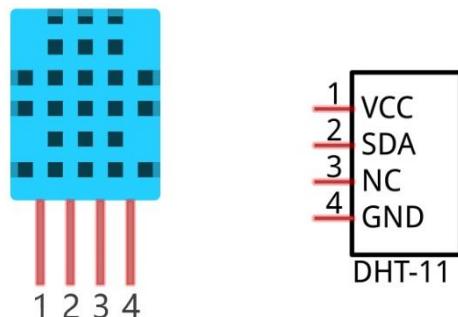
Jumper M/M x4



Component Knowledge

DHT11

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

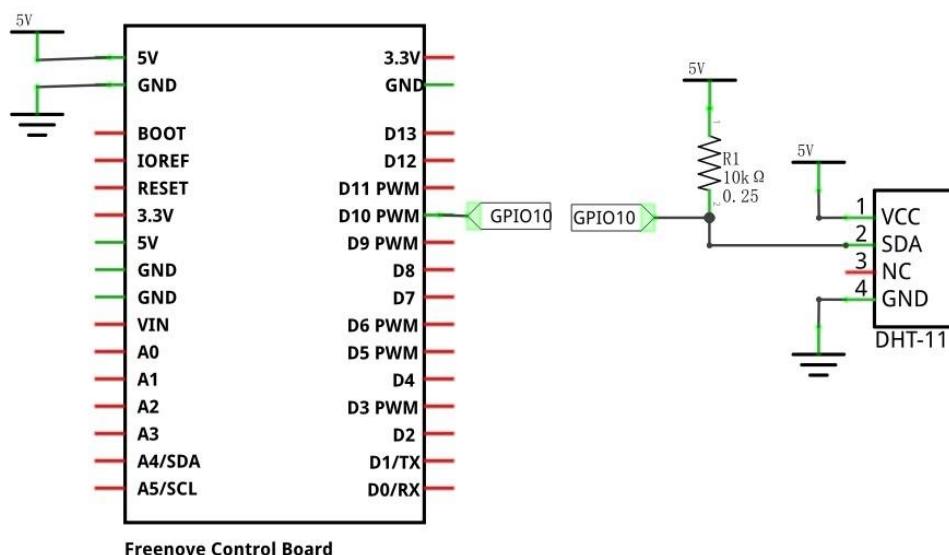


DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

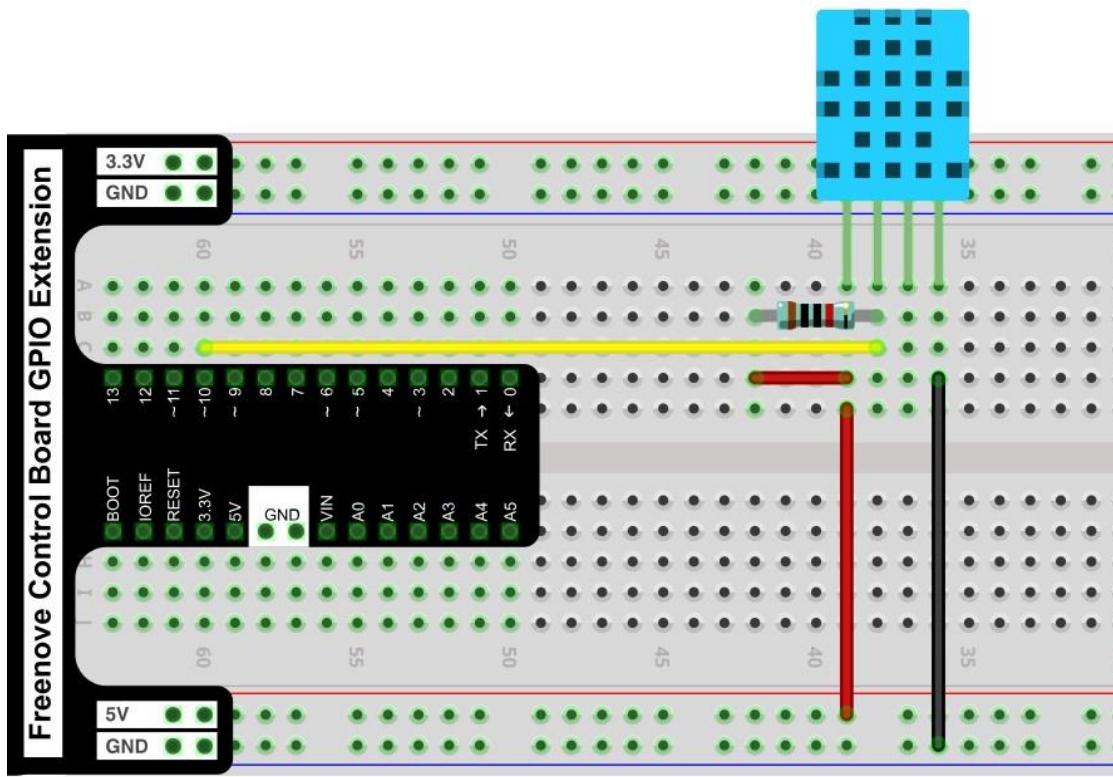
Circuit

Use pin 10 on control board to connect DHT11.

Schematic diagram



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



Sketch

Sketch 26.1.1

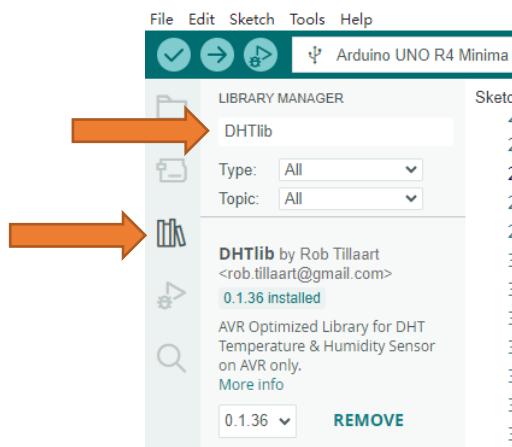
This code uses a library named "DHTlib", if you have not installed it, please do so first.

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.

How to install the library

There are two ways to add libraries.

you can search "DHTlib" in library manager to install.



The second way, Click “Add .ZIP Library...” and then find DHT.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library can make it convenient for us to control DHT11.

Now, write code to get the temperature and humidity data measured by DHT11, and sent it to the serial port.

```
1 #include <dht.h>
2
3 dht DHT;           // create dht object
4 int dhtPin = 10;   // the number of the DHT11 sensor pin
5
6 void setup() {
7     Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
8 }
9 void loop() {
10    // read DHT11 and judge the state according to the return value
11    int chk = DHT.read11(dhtPin);
12    switch (chk)
13    {
14        case DHTLIB_OK: // When read data successfully, print temperature and humidity data
15            Serial.print("Humidity: ");
16            Serial.print(DHT.humidity);
17            Serial.print("%, Temperature: ");
18            Serial.print(DHT.temperature);
19            Serial.println("C");
20            break;
21        case DHTLIB_ERROR_CHECKSUM: // Checksum error
22            Serial.println("Checksum error");
23            break;
24        case DHTLIB_ERROR_TIMEOUT: // Time out error
25            Serial.println("Time out error");
26            break;
27        default:                  // Unknown error
28            Serial.println("Unknown error");
29            break;
30    }
31    delay(1000);
32 }
```

In the code, we use the dht class provided by the DHT library to control DHT11. The following is a DHT object. As shown below, instantiate one dht object.

```
3 dht DHT;           // create dht object
```

Read DHT11 data and send the result to the serial port in the loop() function.

```
11 // read DHT11 and judge the state according to the return value
12 int chk = DHT.read11(dhtPin);
13 switch (chk)
14 {
15     case DHTLIB_OK: // When read data successfully print temperature and humidity data
```

```

16     Serial.print("Humidity: ");
17     Serial.print(DHT.humidity);
18     Serial.print("%, Temperature: ");
19     Serial.print(DHT.temperature);
20     Serial.println("C");
21     break;
22 ...
23 }

```

Send the failure reason when fail to read.

```

22     case DHTLIB_ERROR_CHECKSUM: // Checksum error
23         Serial.println("Checksum error");
24         break;
25     case DHTLIB_ERROR_TIMEOUT: // Time out error
26         Serial.println("Time out error");
27         break;
28     default:                  // Unknown error
29         Serial.println("Unknown error");
30         break;

```

Verify and upload the code, open the Serial Monitor, and then you will see the temperature and humidity data being sent from control board.



Chapter 27 Infrared Motion Sensor

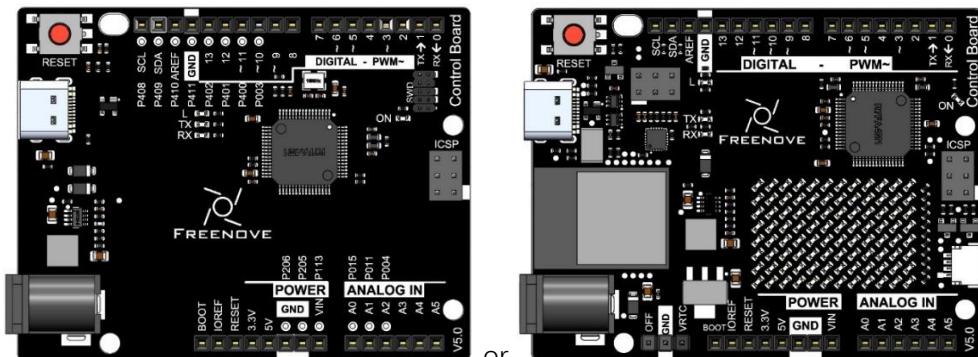
Infrared Motion Sensor is an integrated sensor that can sense the motion of the human body. Now let us try to use it.

Project 27.1 Infrared Motion Sensor

Now, we'll use Infrared Motion Sensor to detect human motion.

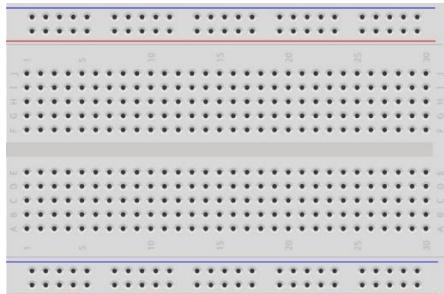
Component List

Control board x1

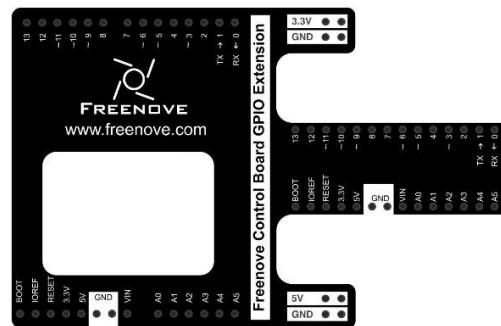


or

Breadboard x1



GPIO Extension Board x1



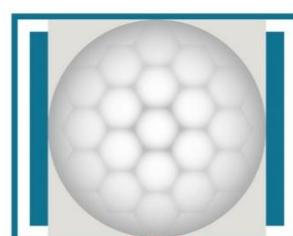
USB cable x1



Jumper F/M x5



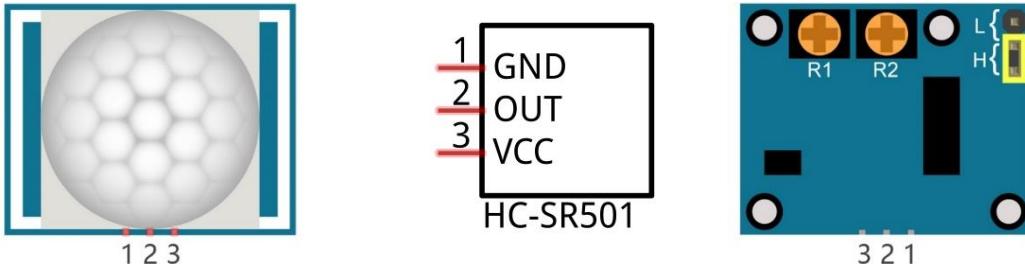
Infrared motion sensor x1



Component Knowledge

Infrared Motion Sensor

Infrared Motion Sensor is an integrated sensor that can sense the motion of a human body. It can detect whether someone is moving in the sensing range by detecting the infrared light emitted by human body. Here is the HC-SR501 infrared motion sensor and its back:



Description:

1. Working voltage: 5v-20v(DC), static current: 65uA.
2. Automatic trigger. When a living body enters into the active area of sensor, the module will output high level (3.3V: Though the high level of control board is 5v, 3.3v is identified as high level here). 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.
3. According to the position of Fresnel lenses dome, you can choose non-repeatable trigger modes or repeatable modes.

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.

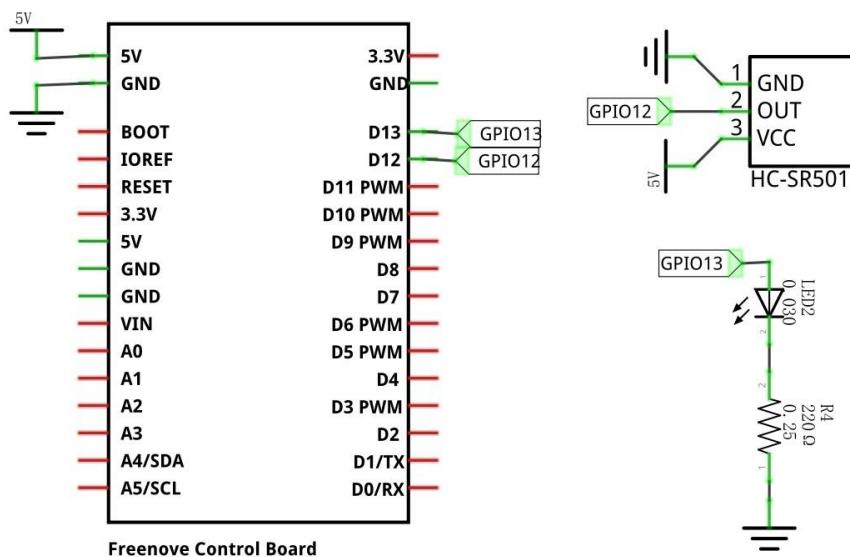
4. Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level.
5. Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.
6. 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 sensitively. When a body moves close to or moves away from the sensor's dome in a vertical direction (perpendicular to the dome), the sensor cannot detect well (please take note of this deficiency). Actually this makes sense when you consider that this sensor is usually placed on a ceiling as part of a security product. 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.

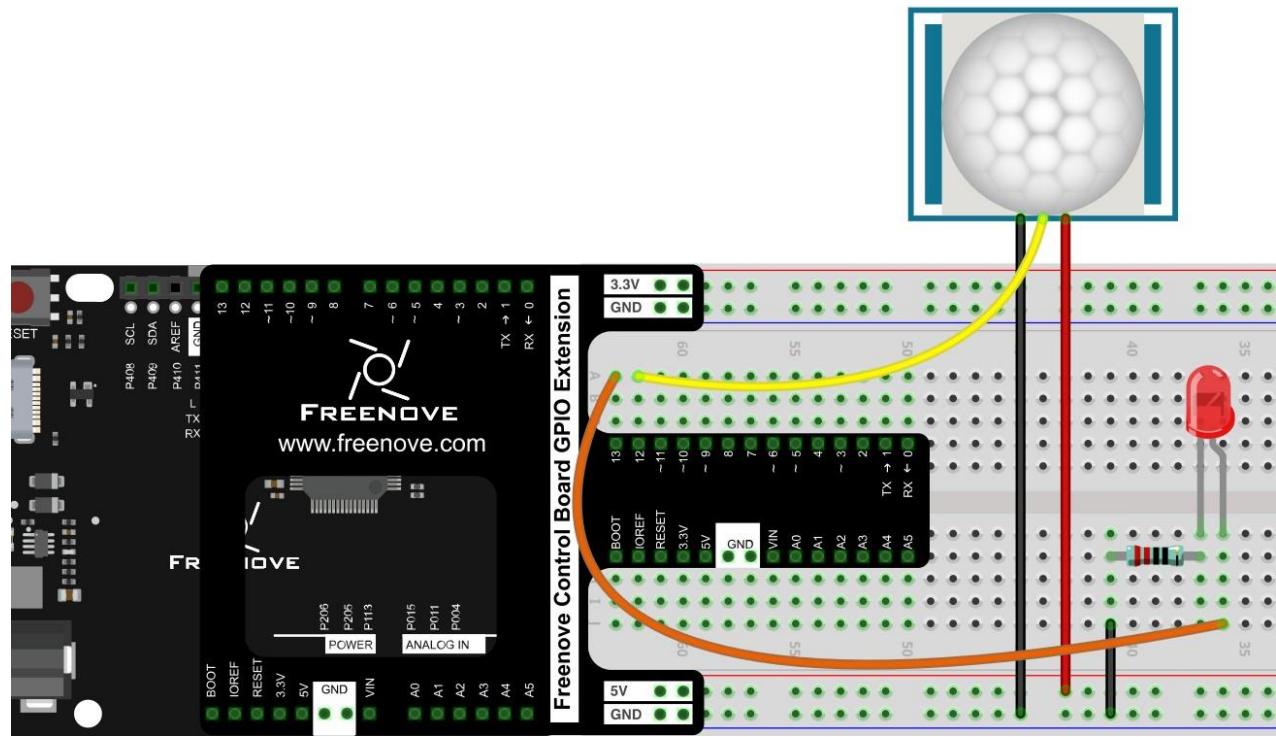
Circuit

Use pin 12 on the control board to connect out-pin of HC-SR501 infrared motion sensor.

Schematic diagram



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



Sketch

Sketch 27.1.1

Now, write code to get the results measured by the HC-SR501 infrared motion sensor, and display that through LED.

```
1 int sensorPin = 12; // the number of the infrared motion sensor pin
2 int ledPin = 13;    // the number of the LED pin
3
4 void setup() {
5     pinMode(sensorPin, INPUT); // initialize the sensor pin as input
6     pinMode(ledPin, OUTPUT);  // initialize the LED pin as output
7 }
8
9 void loop() {
10    // Turn on or off LED according to Infrared Motion Sensor
11    digitalWrite(ledPin, digitalRead(sensorPin));
12    delay(1000);           // wait for a second
13 }
```

This code is relatively simple. We have obtained the sensor's output signal, and control a LED according to it.

```
11    digitalWrite(ledPin, digitalRead(sensorPin));
```

Verify and upload the code, put the sensor on a stationary table and wait for about a minute. And then try to get close to or away from the sensor, and observe whether the LED will be turned on or turned off automatically.

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

Apart from that, you can use this sensor to control some other modules to achieve different functions by re-editing the code, such as the induction lamp, induction door.

Chapter 28 Ultrasonic Ranging

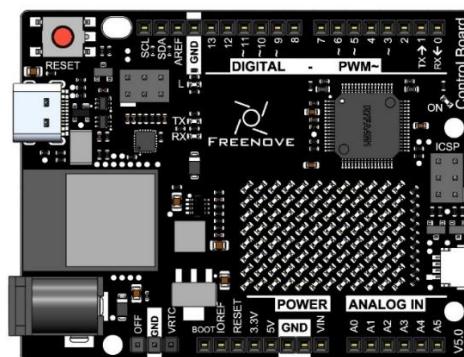
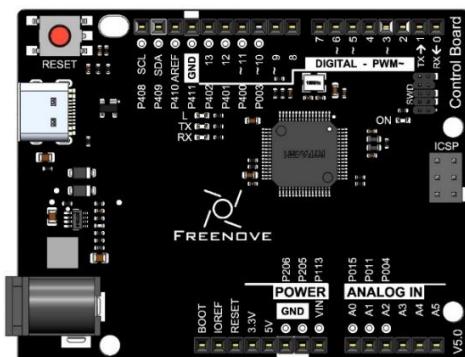
In this chapter, we learn a module that use ultrasonic to measure distance, HC-SR04 ultrasonic ranging module.

Project 28.1 Ultrasonic Ranging

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

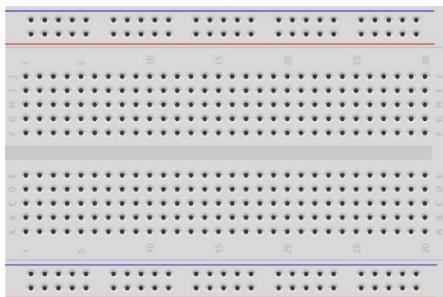
Component List

Control board x1

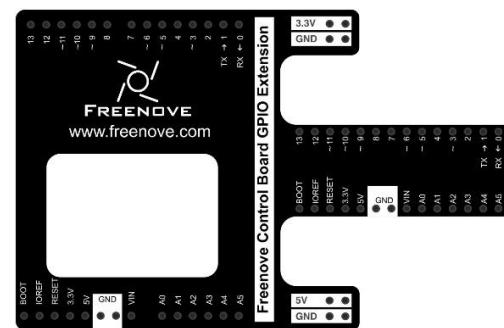


or

Breadboard x1



GPIO Extension Board x1



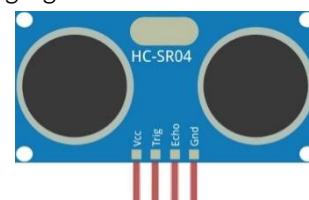
USB cable x1



Jumper F/M x4



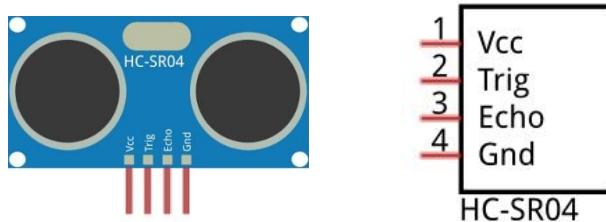
Ultrasonic ranging module x1



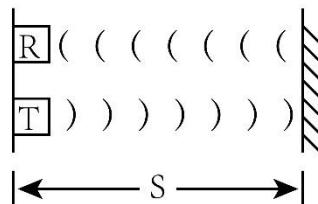
Component Knowledge

Ultrasonic ranging module

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:



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



Pin description:

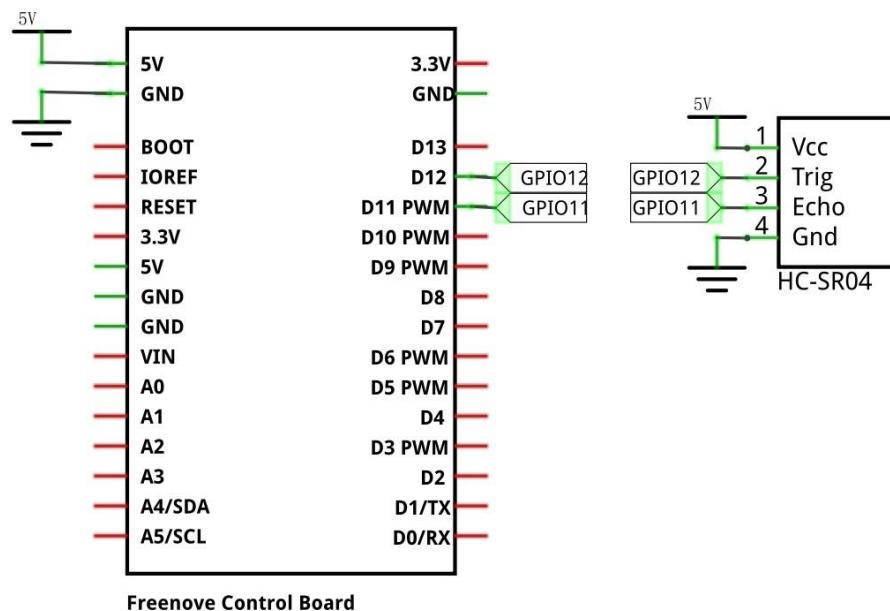
Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Trigger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

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, $s=vt/2$. This is done constantly.

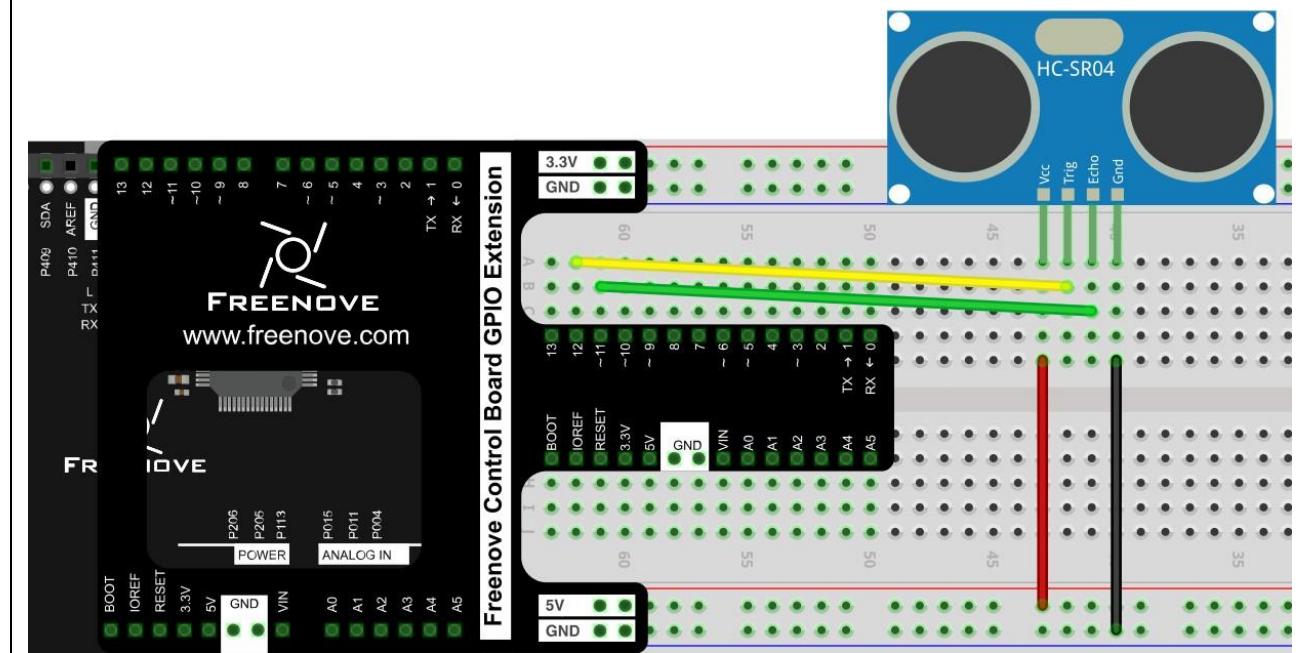
Circuit

The connection of the control board and HC-SR04 is shown below.

Schematic diagram



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



Sketch

Sketch 28.1.1

First, we use the HC-SR04 communication protocol to operate the module, get the range of time, and calculate the distance.

```

1 #define trigPin 12 // define TrigPin
2 #define echoPin 11 // define EchoPin.
3 #define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400~500cm.
4 // define the timeOut according to the maximum range. timeOut= 2*MAX_DISTANCE /100 /340
5 *1000000 = MAX_DISTANCE*58.8
6 float timeOut = MAX_DISTANCE * 60;
7 int soundVelocity = 340; // define sound speed=340m/s
8
9 void setup() {
10     pinMode(trigPin, OUTPUT); // set trigPin to output mode
11     pinMode(echoPin, INPUT); // set echoPin to input mode
12     Serial.begin(9600); // Open serial monitor at 9600 baud to see ping results.
13 }
14
15 void loop() {
16     delay(100); // Wait 100ms between pings (about 20 pings/sec). 29ms should be the
17     shortest delay between pings.
18     Serial.print("Ping: ");
19     Serial.print(getSonar()); // Send ping, get distance in cm and print result (0 =
20     outside set distance range)
21     Serial.println("cm");
22 }
23
24 float getSonar() {
25     unsigned long pingTime;
26     float distance;
27     digitalWrite(trigPin, HIGH); // make trigPin output high level lasting for 10µs to
28     trigger HC_SR04,
29     delayMicroseconds(10);
30     digitalWrite(trigPin, LOW);
31     pingTime = pulseIn(echoPin, HIGH, timeOut); // Wait HC-SR04 returning to the high level
32     and measure out this waiting time
33     distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance
34     according to the time
35     return distance; // return the distance value
36 }
```

First, define the pins and the maximum measurement distance.

```
1 #define trigPin 12 // define TrigPin
2 #define echoPin 11 // define EchoPin.
3 #define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
```

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, timeOut = 2 * MAX_DISTANCE / 100 / 340 * 1000000. The result of the constant part in this formula is approximately equal to 58.8.

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

Then, in the setup(), set the pin to input or output, and set the serial port. In the loop(), we continue to use serial to print the value of subfunction getSonar(), which is used to return the measured distance of the HC_SR04. Make trigPin output a high level lasting for at least 10 μ s, according to the communication protocol.

```
24 digitalWrite(trigPin, HIGH); // make trigPin output high level lasting for 10  $\mu$ s to
triger HC_SR04,
25 delayMicroseconds(10);
26 digitalWrite(trigPin, LOW);
```

And then the echoPin of HC_SR04 will output a pulse. Time of the pulse is the total time of ultrasonic from transmitting to receiving. We use the pulseIn() function to return the time, and set the timeout.

```
27 pingTime = pulseIn(echoPin, HIGH, timeOut); // Wait HC-SR04 returning to the high level
and measure out this waitting time
```

Calculate the distance according to the time and return the value.

```
28 distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance
according to the time
29 return distance; // return the distance value
```

Finally, the code above will be called in the loop().

pulseIn(pin, value) / pulseIn(pin, value, timeout)

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

Verify and upload the code to the control board, open the serial port monitoring window, turn the HC-SR04 probe towards the object plane, and observe the data in the serial port monitoring window.

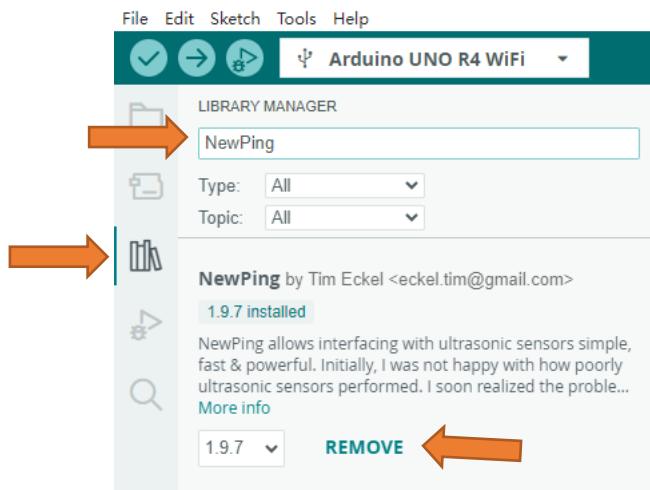


Sketch 28.1.2

Before writing code, we need to import the library needed.

Click “Add .ZIP Library...” and then find NewPing.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library makes it easy to obtain the measuring distance.

You can also type "NewPing" into the library search bar to install the library.



```

1 #include <NewPing.h>
2 #define trigPin 12 // define TrigPin
3 #define echoPin 11 // define EchoPin.
4 #define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
5
6 NewPing sonar(trigPin, echoPin, MAX_DISTANCE); // NewPing setup of pins and maximum
distance.
7
8 void setup() {
9     Serial.begin(9600); // Open serial monitor at 9600 bauds to see ping results.
10 }
11
12 void loop() {
13     delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest
delay between pings.
14     Serial.print("Ping: ");
15     Serial.print(sonar.ping_cm()); // Send ping, get distance in cm and print result (0 =
outside set distance range)
16     Serial.println("cm");
17 }
```

First, include the header file of library, and then define the HC SR04 pin and the maximum measurement distance. And then write these parameters when we define the NewPing class objects.

```

1 #include <NewPing.h>
2 #define trigPin 12 // define TrigPin
```

```
3 #define echoPin 11 // define EchoPin.  
4 #define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
```

And then, in the loop (), use sonar.ping_cm () to obtain the ultrasonic module detection distance with unit of centimeter. And print the distance out. When the distance exceeds range of 2cm~200cm, the printed data is zero.

NewPing Class

NewPing class can be used for SR04, SRF05, SRF06 and other sensors. An object that needs to be instantiated when the class is used. The three parameters of the constructor function are: trigger pin, echo pin and maximum measurement distance.

```
NewPing sonar(trigger_pin, echo_pin [, max_cm_distance])
```

Some member function:

sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

sonar.ping_in() - Send a ping and get the distance in whole inches.

sonar.ping_cm() - Send a ping and get the distance in whole centimeters.

For more details, please refer to the NewPing.h in the NewPing library.

Verify and upload the code to control board and open the Serial Monitor. When you make ultrasonic probe toward a plane of an object, you can observe the distance between them.



Chapter 29 LEDpixel

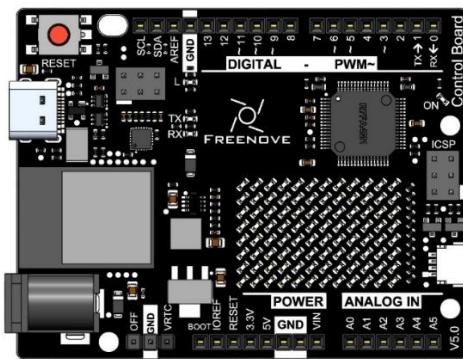
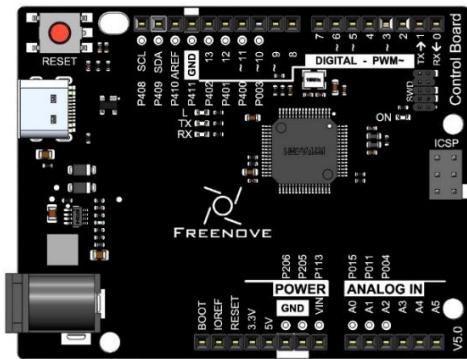
This chapter will help you learn to use a more convenient RGB LED lamp, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 29.1 LEDpixel

Learn the basic usage of LEDPixel and use it to flash red, green, blue and white.

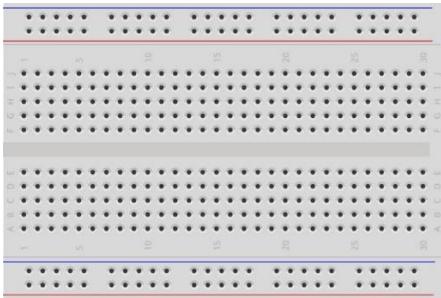
Component List

Control board x1

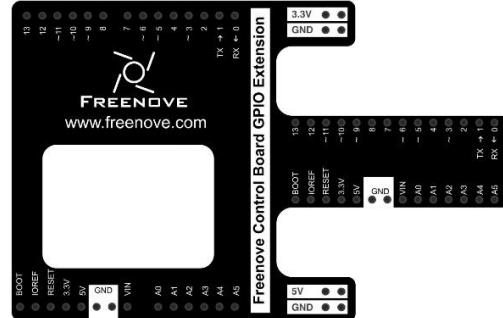


or

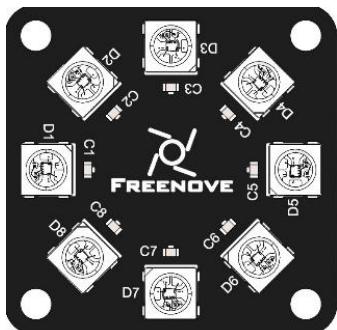
Breadboard x1



GPIO Extension Board x1



Freenove 8 RGB LED Module x1



Jumper x3



USB cable x1



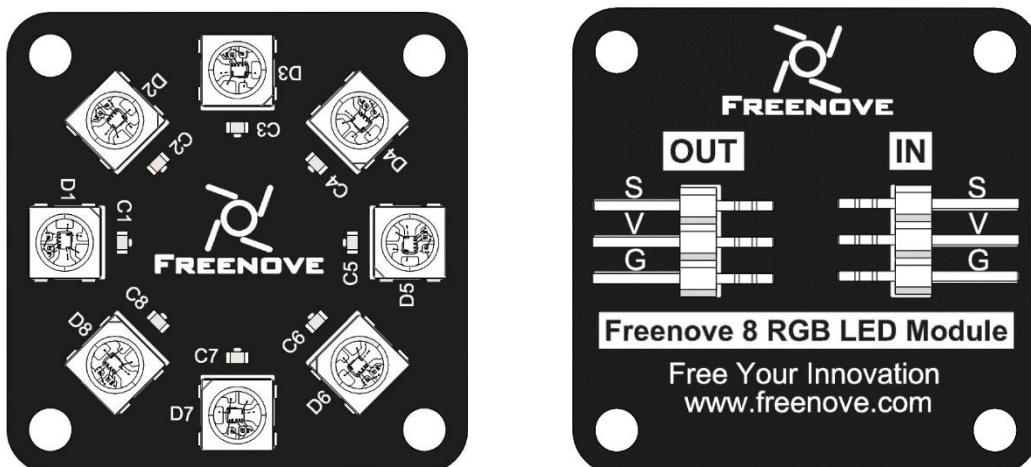
Component knowledge

Freenove 8 RGB LED Module

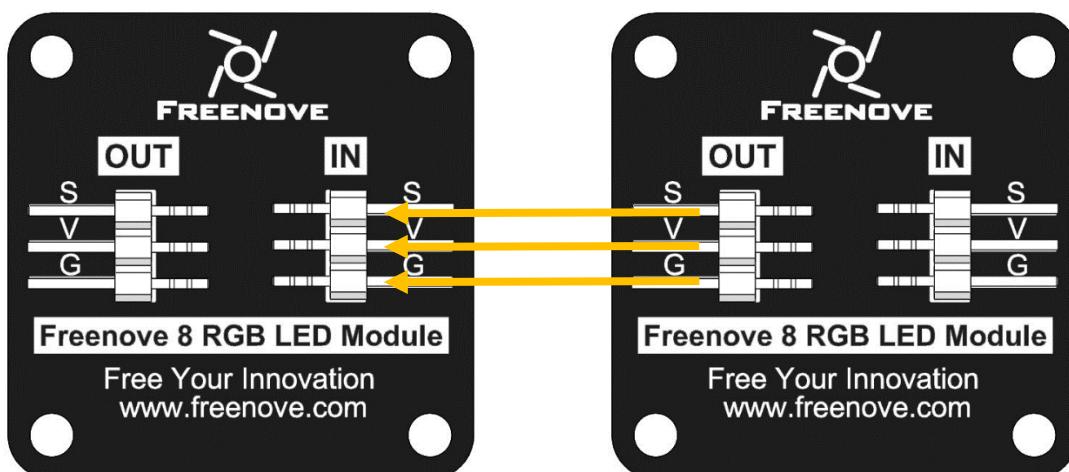
The Freenove 8 RGB LED Module is as below.

It consists of 8 WS2812, each of which requires only one pin to control and supports cascade. Each WS212 has integrated 3 LEDs, red, green and blue respectively, and each of them supports 256-level brightness adjustment, which means that each WS2812 can emit $2^{24}=16,777,216$ different colors.

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 such way, you can use one data pin to control 8, 16, 32 ... LEDs.

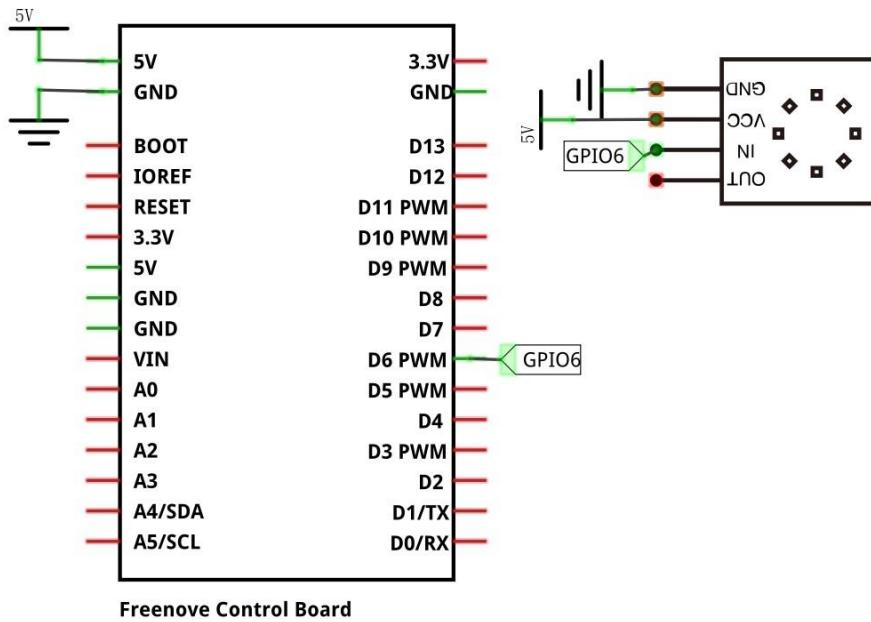


Pin description:

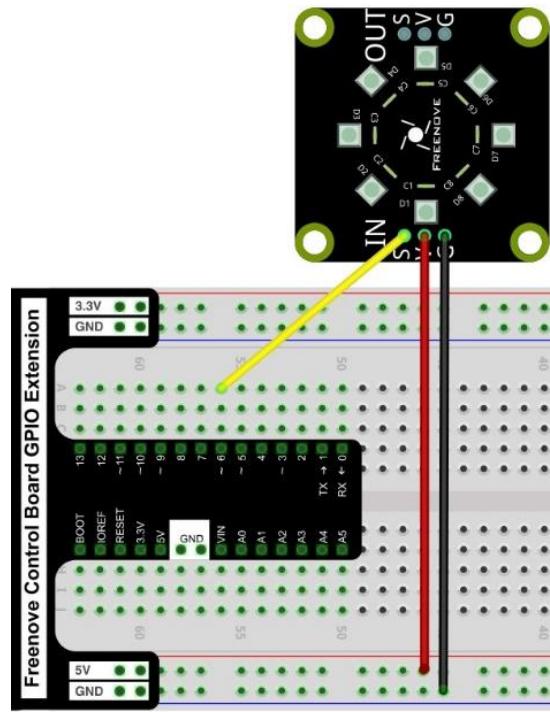
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



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



Sketch

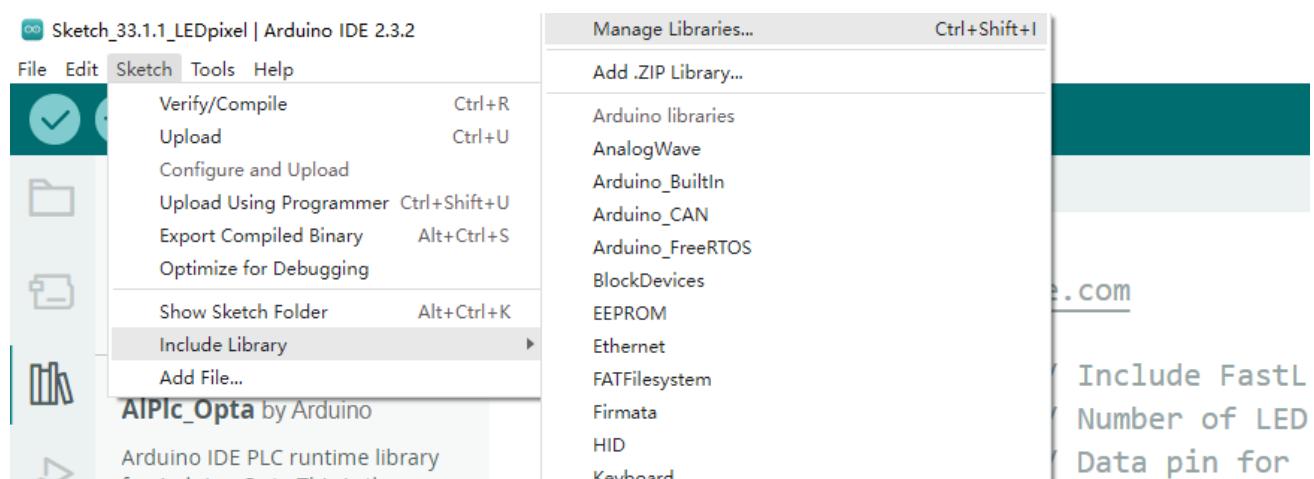
This code uses a library named "FastLED", if you have not installed it, please do so first.

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.

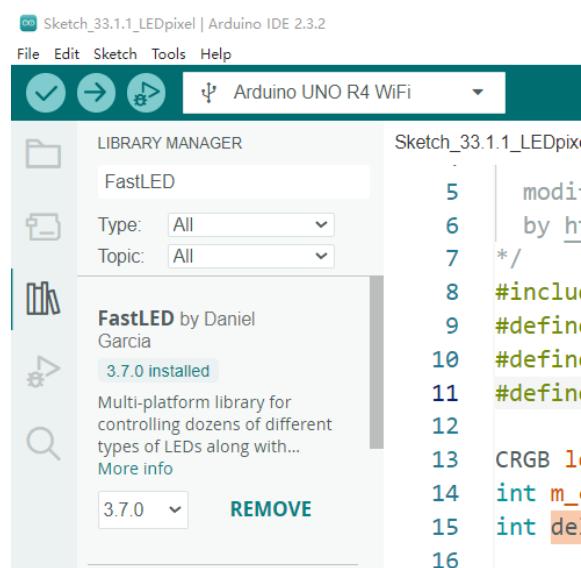
How to install the library

There are two ways to add libraries.

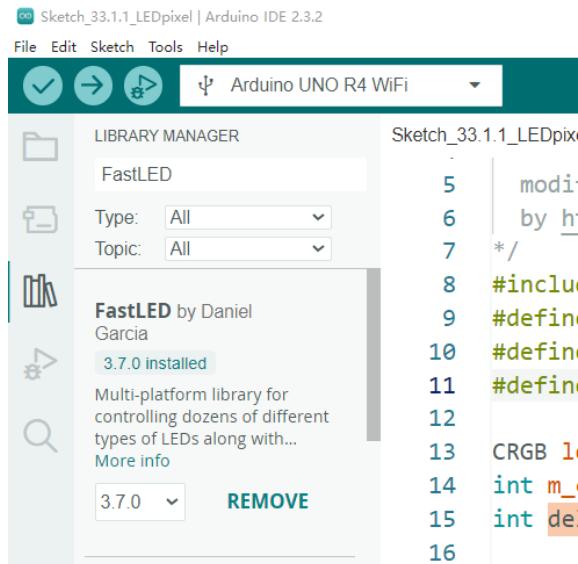
The first way, open the Arduino IDE, click Tools → Manager Libraries.



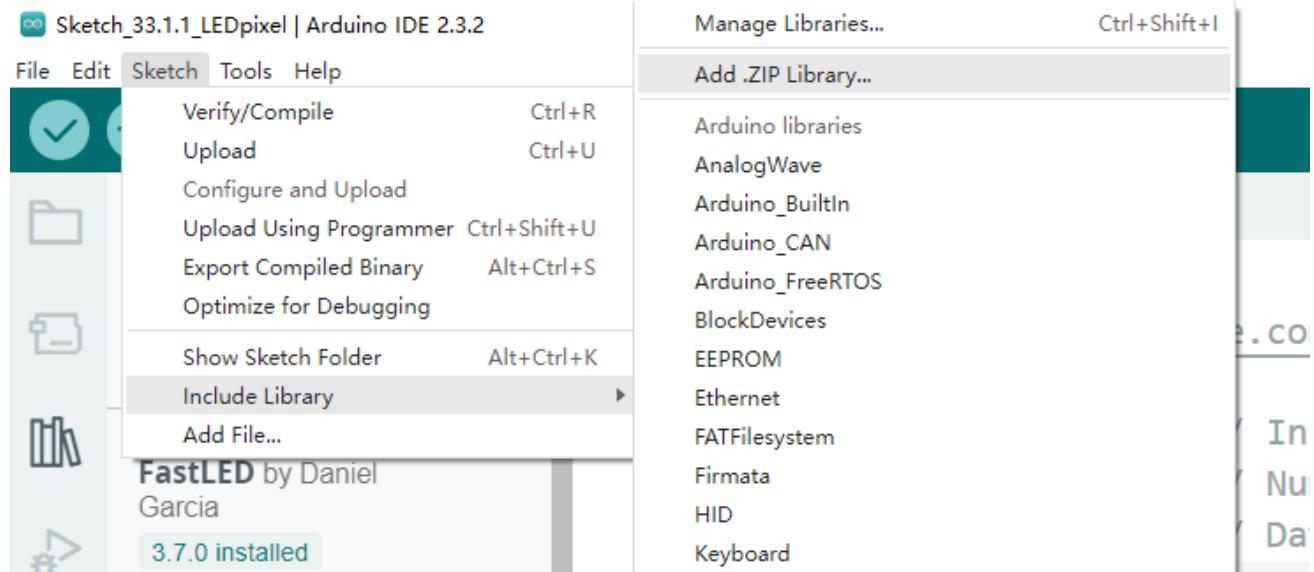
In the pop-up window, Library Manager, search for the name of the Library, "FastLED". Then click Install.



Or, you can search "FastLED" in library manager to install.

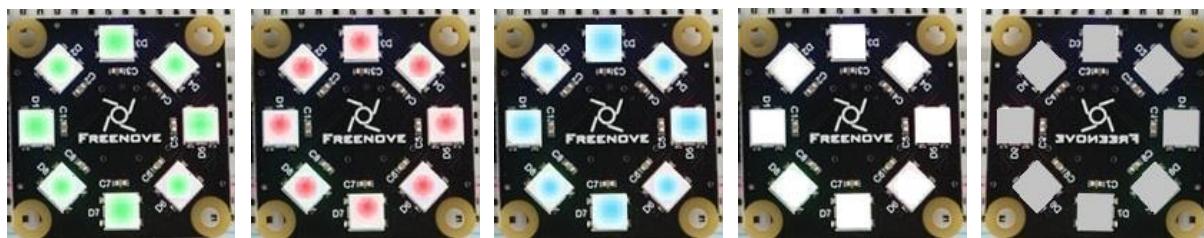


The second way, open Arduino IDE, click Sketch→Include Library→Add .ZIP Library, In the pop-up window, find the file named "./Libraries/ FastLED.Zip" which locates in this directory, and click OPEN.



Sketch 29.1.1

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



The following is the program code:

```

1 #include <FastLED.h> // Include FastLED library
2 #define NUM_LEDS 8 // Number of LEDs in the chain
3 #define DATA_PIN 6 // Data pin for LED control

```

```

4 #define BRIGHTNESS 64
5
6 CRGB leds[NUM_LEDS]; // Array to hold LED color data
7 int m_color[5][3] = { { 255, 0, 0 }, { 0, 255, 0 }, { 0, 0, 255 }, { 255, 255, 255 }, { 0, 0,
8 0 } };
9 int delayval = 100;
10
11 void setup() {
12   FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS); // Initialize LEDs
13   FastLED.setBrightness(BRIGHTNESS);
14 }
15
16 void loop() {
17   // Loop through each LED and set it to blue
18   for (int j = 0; j < 5; j++) {
19     for (int dot = 0; dot < NUM_LEDS; dot++) {
20       leds[dot] = CRGB(m_color[j][0], m_color[j][1], m_color[j][2]);
21       FastLED.show(); // Update LEDs
22       leds[dot] = CRGB::Black; // Clear the current LED
23       delay(delayval); // Wait for a short period before moving to the next LED
24     }
25     delay(500);
26   }
27 }
```

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

```
1 #include <FastLED.h> // Include FastLED library
```

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

```
#define NUM_LEDS 8 // Number of LEDs in the chain
#define DATA_PIN 6 // Data pin for LED control
```

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

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

Define a variable to set the time interval for each led to light up. The smaller the value is, the faster it will light up.

```
9 int delayval = 100;
```

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

```
12 FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS); // Initialize LEDs
13 FastLED.setBrightness(BRIGHTNESS);
```

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

```
18 for (int j = 0; j < 5; j++) {
19   for (int dot = 0; dot < NUM_LEDS; dot++) {
20     leds[dot] = CRGB(m_color[j][0], m_color[j][1], m_color[j][2]);
```

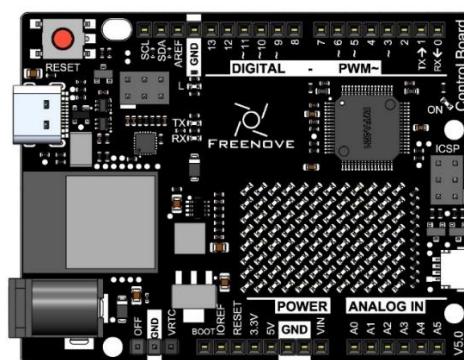
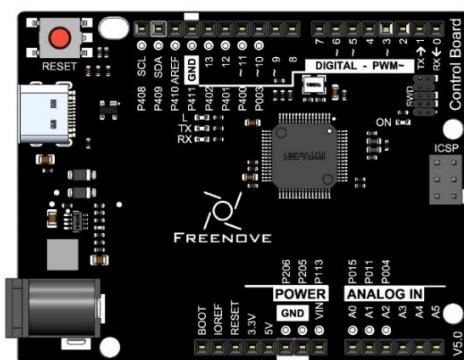
```
21     FastLED.show();           // Update LEDs
22     leds[dot] = CRGB::Black; // Clear the current LED
23     delay(delayval);        // Wait for a short period before moving to the next LED
24 }
25     delay(500);
26 }
```

Project 29.2 RainbowLight

In the previous project, we have mastered the use of LEDPixel. This project will realize a slightly complicated rainbow light. The component list and the circuit are exactly the same as the project fashionable light.

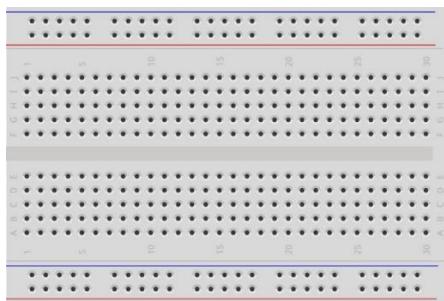
Component List

Control board x1

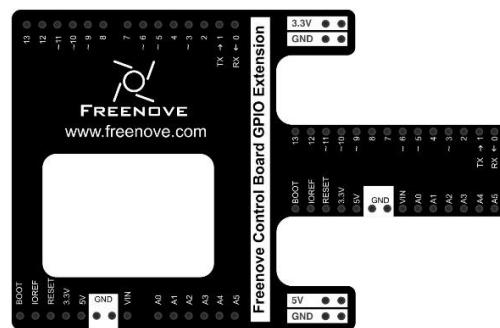


or

Breadboard x1



GPIO Extension Board x1



Freenove 8 RGB LED Module x1



Jumper x3

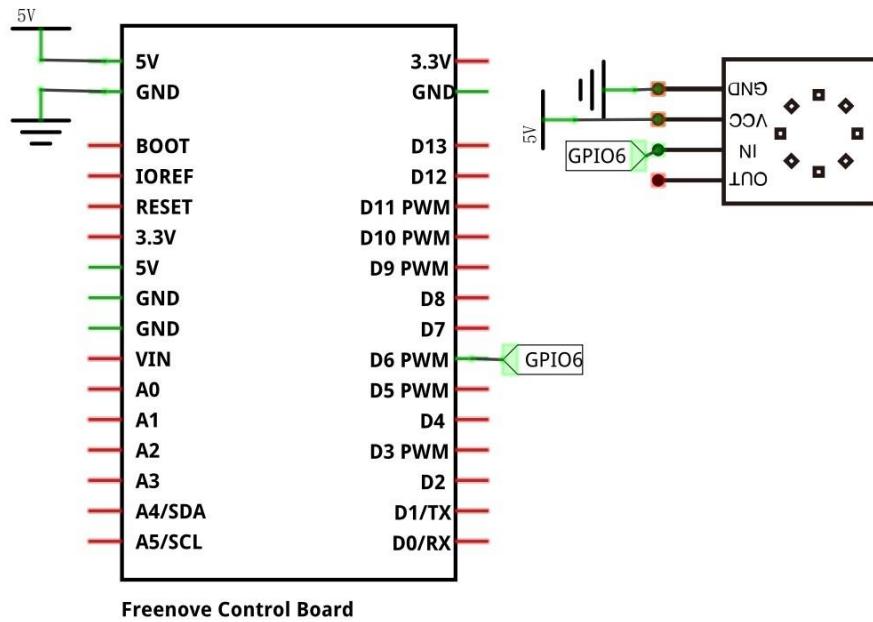


USB cable x1



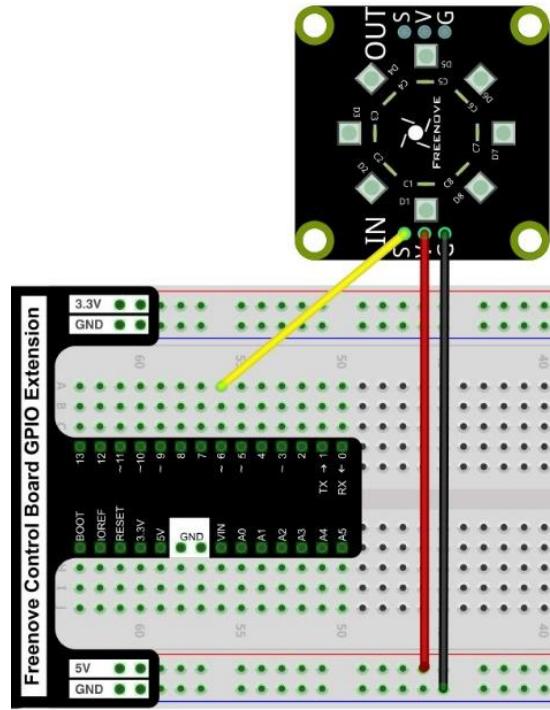
Circuit

Schematic diagram



Freenove Control Board

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



Sketch

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



Sketch 29.2.1

Download the code to ESP32-WROVER and RGB LED begins to light up in red, green, blue, white and black.



The following is the program code:

```
1 #include <FastLED.h> // Include FastLED library
2 #define NUM_LEDS 8 // Number of LEDs in the chain
3 #define DATA_PIN 6 // Data pin for LED control
4 #define BRIGHTNESS 64
5
6 int red = 0;
7 int green = 0;
8 int blue = 0;
9 CRGB leds[NUM_LEDS]; // Array to hold LED color data
10
11 void setup() {
12     FastLED.addLeds<NEOPixel, DATA_PIN>(leds, NUM_LEDS); // Initialize LEDs
13     FastLED.setBrightness(BRIGHTNESS);
14 }
15
16 void loop() {
17     for (int j = 0; j < 256 * 5; j++) {
18         for (int i = 0; i < 8; i++) {
19             Wheel(((i * 256 / 8) + j)%255);
20             leds[i] = CRGB(red, green, blue);
21         }
22         FastLED.show(); // Update LEDs
23         delay(10);
24     }
25 }
```

```

27 void Wheel(byte WheelPos) {
28     WheelPos = 255 - WheelPos;
29     if (WheelPos < 85) {
30         red = 255 - WheelPos * 3;
31         green = 0;
32         blue = WheelPos * 3;
33     }
34     else if (WheelPos < 170) {
35         WheelPos -= 85;
36         red = 0;
37         green = WheelPos * 3;
38         blue = 255 - WheelPos * 3;
39     }
40     else {
41         WheelPos -= 170;
42         red = WheelPos * 3;
43         green = 255 - WheelPos * 3;
44         blue = 0;
45     }
46 }
```

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 / LEDS_COUNT + j) & 255 will take color from the color model at equal intervals starting from i.

```

17 for (int j = 0; j < 256 * 5; j++) {
18     for (int i = 0; i < 8; i++) {
19         Wheel(((i * 256 / 8) + j)%255);
20         leds[i] = CRGB(red, green, blue);
21     }
22     FastLED.show();           // Update LEDs
23     delay(10);
24 }
```

Chapter 30 RFID

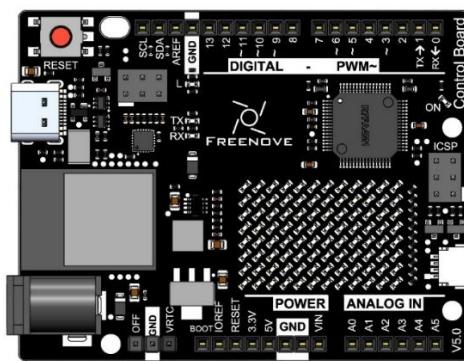
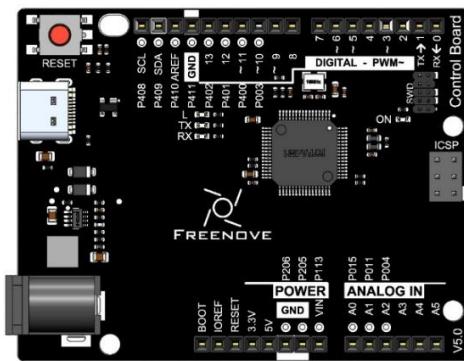
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 30.1 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.

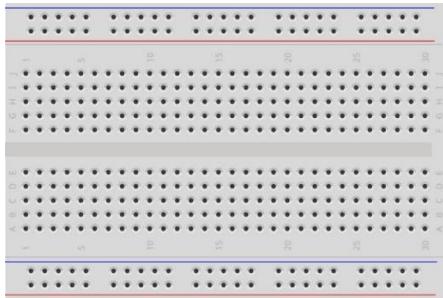
Component List

Control board x1

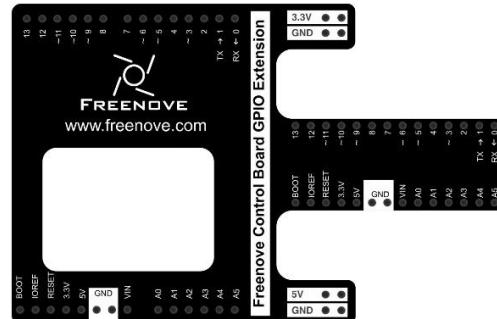


or

Breadboard x1



GPIO Extension Board x1

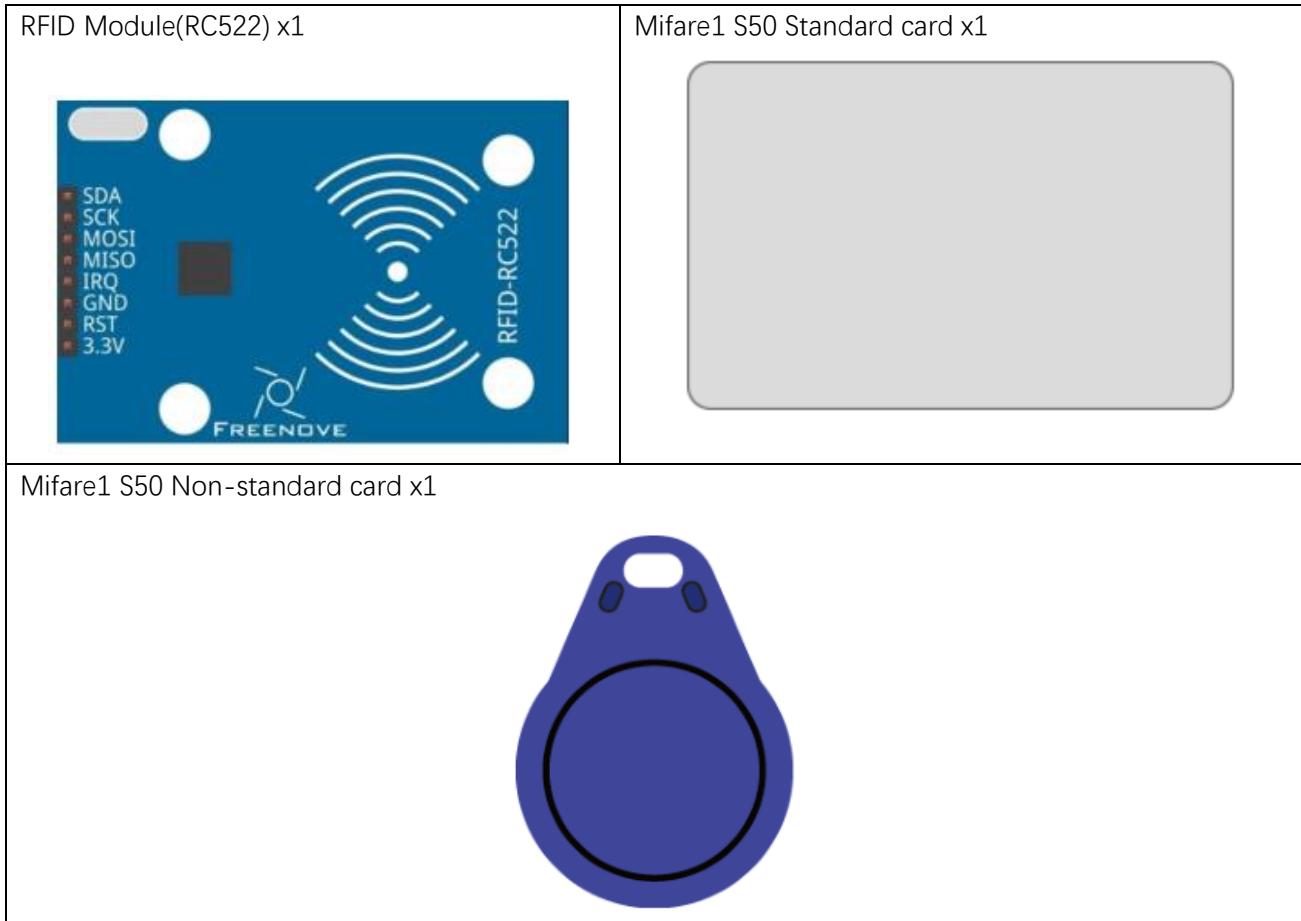


USB cable x1



Jumper





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.

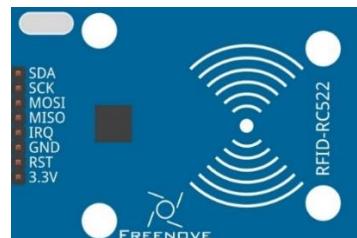
MFRC522 RFID Module

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

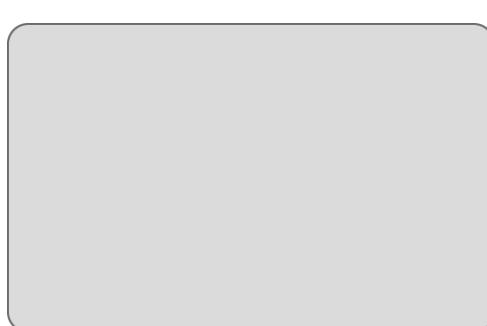
provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

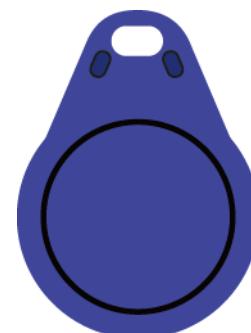


Mifare1 S50 Card

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), $64 \times 16 = 1024$. As is shown in the following table:

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

Each sector has a set of independent password and access control 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.

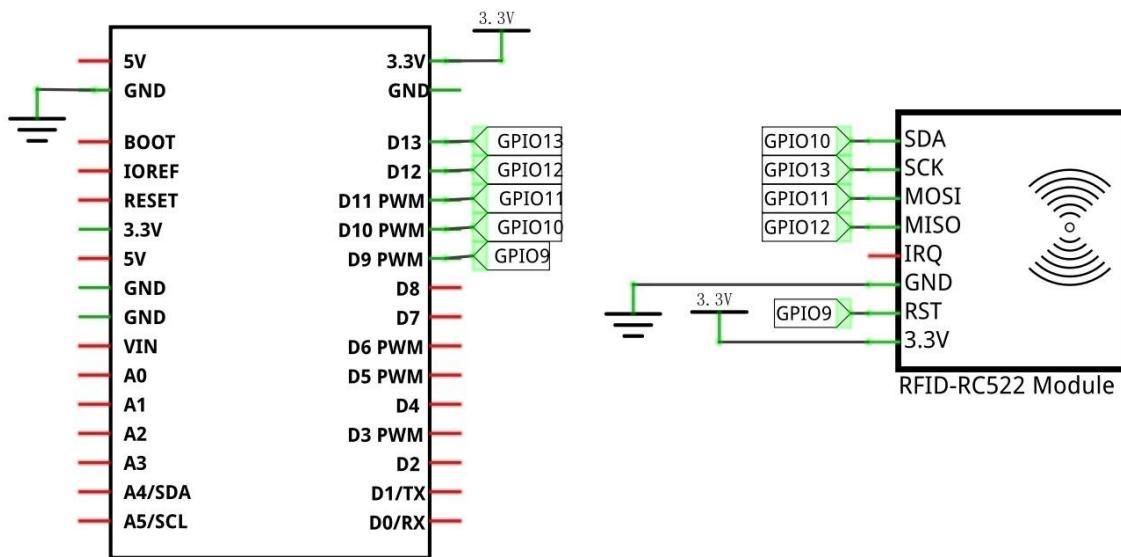
For more details about how to set data blocks and control blocks, please refer to Datasheet.

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.**

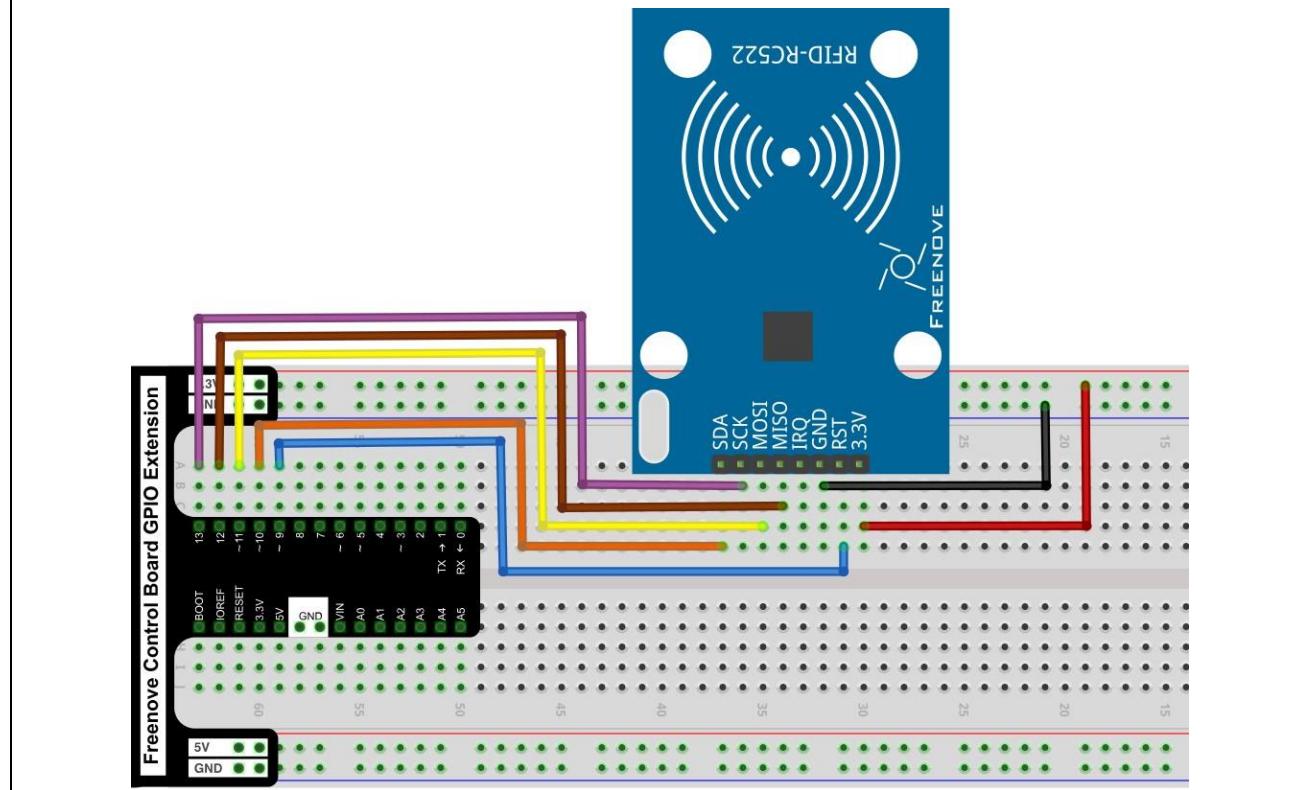
Circuit

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_30.1.1

Before writing code, we need to import the library needed.

Need help? Contact support@freenove.com

Click “Add .ZIP Library...” and then find **RFID.zip** in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library make it easy to operate RFID module.

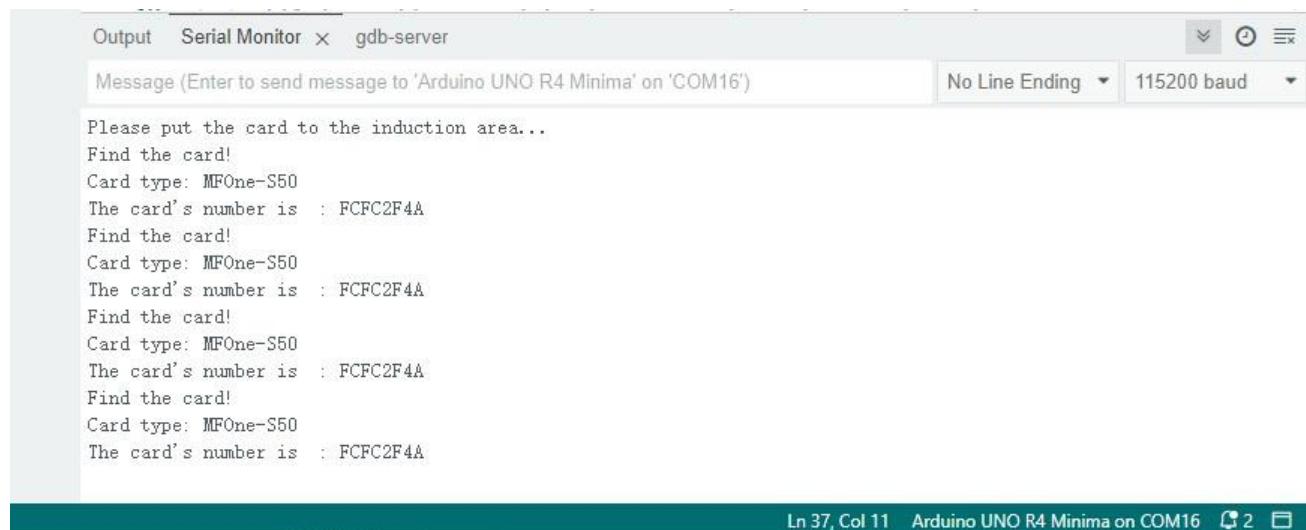
This sketch will read the unique ID number (UID) of the card, recognize the type of the card and display the information through serial port.

```

1 #include <SPI.h>
2 #include <RFID.h>
3
4 RFID rfid(10, 9);
5 unsigned char status;
6 unsigned char str[MAX_LEN]; //MAX_LEN is 16: size of the array
7
8 void setup()
9 {
10   Serial.begin(115200);
11   SPI.begin();
12   rfid.init(); //initialization
13   Serial.println("Please put the card to the induction area...");
14 }
15
16 void loop()
17 {
18   //Search card, return card types
19   if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
20     Serial.println("Find the card!");
21     // Show card type
22     ShowCardType(str);
23     //Anti-collision detection, read card serial number
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     }
33     //card selection (lock card to prevent redundant read, removing the line will make
34     //the sketch read cards continually)
35     rfid.selectTag(str);
36   }
37   rfid.halt(); // command the card to enter sleeping state
}
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(10, 9);
```

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 30.2 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 30.2.1

In this sketch, first read the data in particular location of the S50 M1 Card, then write data in that position and read it out. Display these data through the serial port.

```
1 #include <SPI.h>
2 #include <RFID.h>
3
4 //pin5:pin of card reader SDA. pin6:pin of card reader RST
5 RFID rfid(10, 9);
6
7 // 4-byte card serial number, the fifth byte is check byte
8 unsigned char serNum[5];
9 unsigned char status;
10 unsigned char str[MAX_LEN];
11 unsigned char blockAddr;           //Select the operation block address: 0 to 63
12
13 // Write card data you want(within 16 bytes)
14 unsigned char writeDate[16] = "WelcomeFreenove";
15
16 // The A password of original sector: 16 sectors; the length of password in each sector
17 // is 6 bytes.
17 unsigned char sectorKeyA[16][16] = {
18     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
19     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
20     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
21     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
22     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
23     { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,
```

```
24 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
25 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
26 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
27 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
28 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
29 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
30 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
31 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
32 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
33 { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF } ,  
34 } ;  
35  
36 void setup()  
37 {  
38     Serial.begin(115200);  
39     SPI.begin();  
40     rfid.init();  
41     Serial.println("Please put the card to the induction area...");  
42 }  
43  
44 void loop()  
45 {  
46     //find the card  
47     rfid.findCard(PICC_REQIDL, str);  
48     //Anti-collision detection, read serial number of card  
49     if (rfid.anticoll(str) == MI_OK) {  
50         Serial.print("The card's number is : ");  
51         //print the card serial number  
52         for (int i = 0; i < 4; i++) {  
53             Serial.print(0x0F & (str[i] >> 4), HEX);  
54             Serial.print(0x0F & str[i], HEX);  
55         }  
56         Serial.println("");  
57         memcpy(rfid.serNum, str, 5);  
58     }  
59     //select card and return card capacity (lock the card to prevent multiple read and  
written)  
60     rfid.selectTag(rfid.serNum);  
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);
```

```
67     rfid.halt();
68 }
69
70 //write the card
71 void writeCard(int blockAddr) {
72     if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) == MI_OK) //authenticate
73     {
74         //write data
75         //status = rfid.write((blockAddr/4 + 3*(blockAddr/4+1)), sectorKeyA[0]);
76         Serial.print("set the new card password, and can modify the data of the Sector: ");
77         Serial.println(blockAddr / 4, DEC);
78         // select block of the sector to write data
79         if (rfid.write(blockAddr, writeDate) == MI_OK) {
80             Serial.println("Write card OK!");
81         }
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 }
```

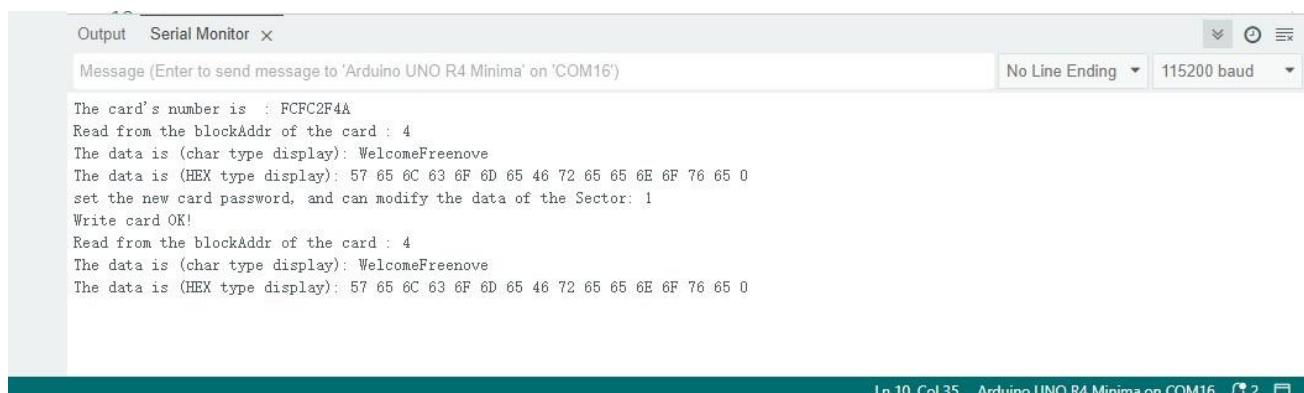
In the sub function of writeCard() and readCard(), we must first verify the password A, and then use the corresponding sub function to read and write. Here we do reading and writing operations to data block 0 (absolute NO.4) of the first sector.

```
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.



Chapter 31 WiFi Working Modes (WiFi Board)

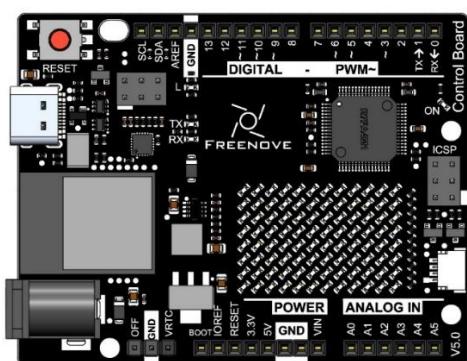
In this chapter, we'll focus on the WiFi infrastructure for control board.

Control board has 3 different WiFi operating modes: station mode, AP mode and AP+station mode. All WiFi programming projects must be configured with WiFi operating mode before using WiFi, otherwise WiFi cannot be used.

Project 31.1 Station mode

Component List

Control board x1



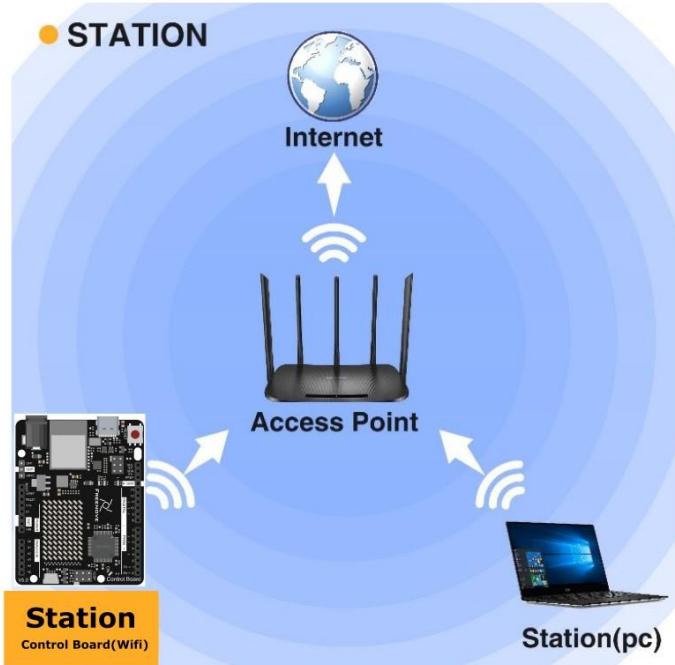
USB cable x1



Component knowledge

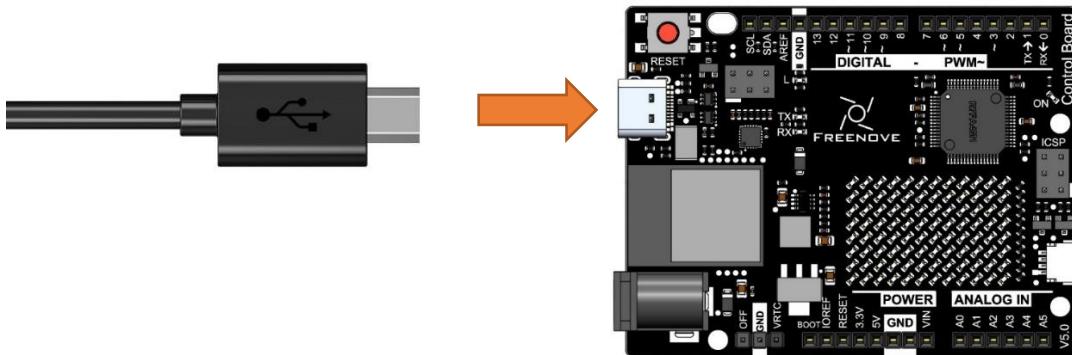
Station mode

When control board(wifi) selects Station mode, it acts as a WiFi client. It can connect to the router network and communicate with other devices on the router via WiFi connection. As shown below, the PC is connected to the router, and if control board(wifi) wants to communicate with the PC, it needs to be connected to the router.



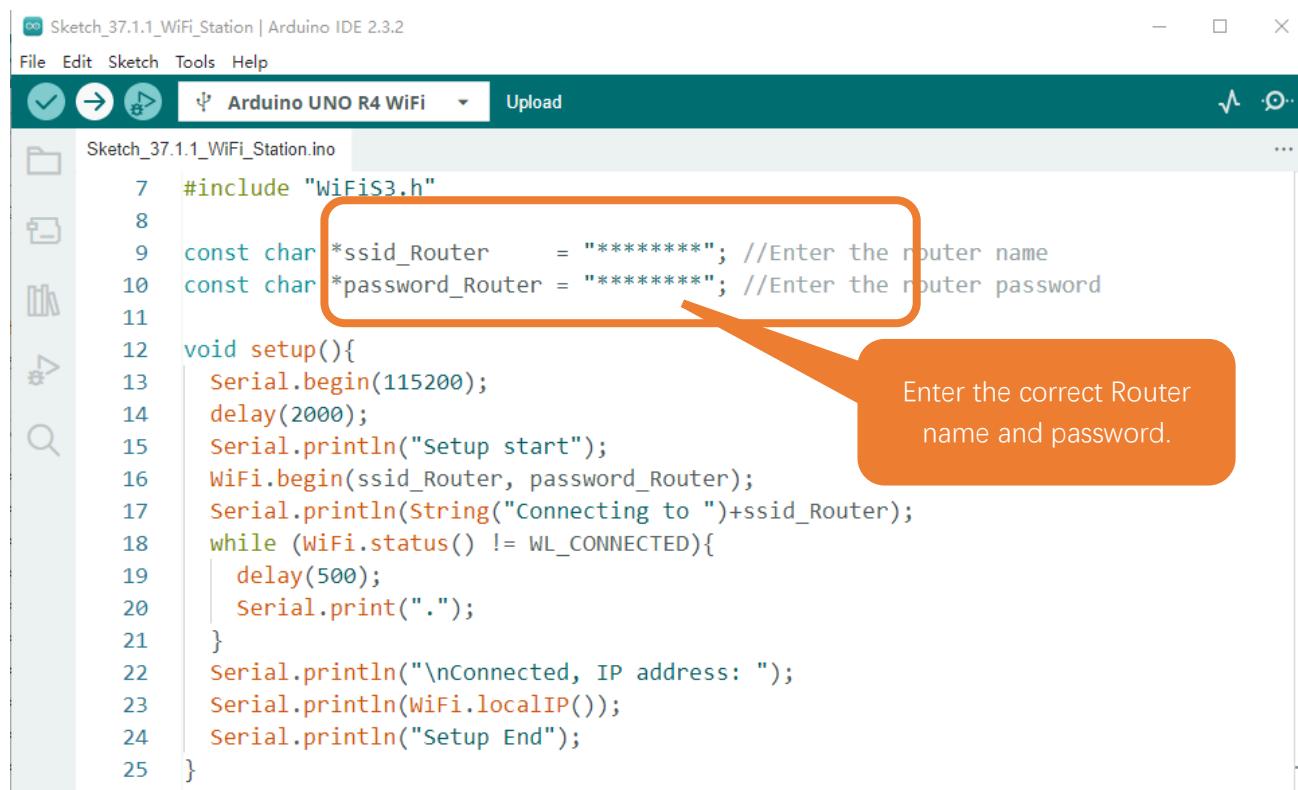
Circuit

Connect Freenove control board(wifi)to the computer using the USB cable.



Sketch

Sketch_31.1.1



```

Sketch_37.1.1_WiFi_Station | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_37.1.1_WiFi_Station.ino Upload
7 #include "WiFiS3.h"
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11
12 void setup(){
13     Serial.begin(115200);
14     delay(2000);
15     Serial.println("Setup start");
16     WiFi.begin(ssid_Router, password_Router);
17     Serial.println(String("Connecting to ") +ssid_Router);
18     while (WiFi.status() != WL_CONNECTED){
19         delay(500);
20         Serial.print(".");
21     }
22     Serial.println("\nConnected, IP address: ");
23     Serial.println(WiFi.localIP());
24     Serial.println("Setup End");
25 }

```

Enter the correct Router name and password.

Because the names and passwords of routers in various places are different, before the Sketch runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to control board, open serial monitor and set baud rate to 115200. And then it will display as follows:



Output Serial Monitor ×

Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM15') No Line Ending 115200 baud

```

Setup start
Connecting to FYI_2.4G

Connected, IP address:
192.168.1.94
Setup End

```

Ln 23, Col 7 Arduino UNO R4 WiFi on COM15 4 2

When control board successfully connects to "ssid_Router", serial monitor will print out the IP address assigned to control board by the router.

The following is the program code:

```
1 #include "WiFiS3.h"
```

```

2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5
6 void setup() {
7     Serial.begin(115200);
8     delay(2000);
9     Serial.println("Setup start");
10    WiFi.begin(ssid_Router, password_Router);
11    Serial.println(String("Connecting to ") + ssid_Router);
12    while (WiFi.status() != WL_CONNECTED) {
13        delay(500);
14        Serial.print(".");
15    }
16    Serial.println("\nConnected, IP address: ");
17    Serial.println(WiFi.localIP());
18    Serial.println("Setup End");
19 }
20
21 void loop() {
22 }
```

Include the WiFi Library header file of control board.

```
#include "WiFiS3.h"
```

Enter correct router name and password.

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

Set control board in Station mode and connect it to your router.

```
WiFi.begin(ssid_Router, password_Router);
```

Check whether control board has connected to router successfully every 0.5s.

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
```

Serial monitor prints out the IP address assigned to control board.

```
Serial.println(WiFi.localIP());
```

Reference

Class Station

Every time when using WiFi, you need to include header file "WiFiS3.h".

begin(ssid, password,channel, bssid, connect): control board is used as Station to connect hotspot.

ssid: WiFi hotspot name

password: WiFi hotspot password

channel: WiFi hotspot channel number; communicating through specified channel; optional parameter

bssid: mac address of WiFi hotspot, optional parameter

connect: boolean optional parameter, defaulting to true. If set as false, then control board won't connect

WiFi.

config(local_ip, gateway, subnet, dns1, dns2): set static local IP address.

local_ip: station fixed IP address.

subnet: subnet mask

dns1,dns2: optional parameter. define IP address of domain name server

status: obtain the connection status of WiFi

local IP(): obtian IP address in Station mode

disconnect(): disconnect wifi

setAutoConnect(boolean): set automatic connection Every time control board is power on, it will connect WiFi aitomatically.

setAutoReconnect(boolean): set automatic reconnection Every time control board disconnects WiFi, it will reconnect to WiFi automatically.

Project 31.2 AP mode

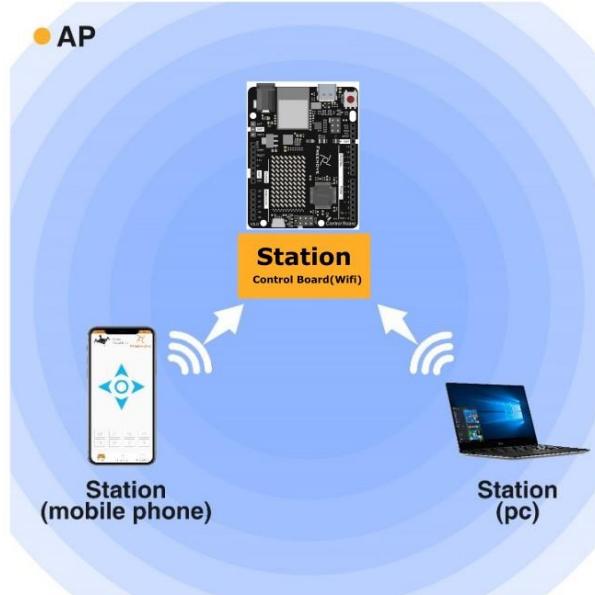
Component List & Circuit

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

Component knowledge

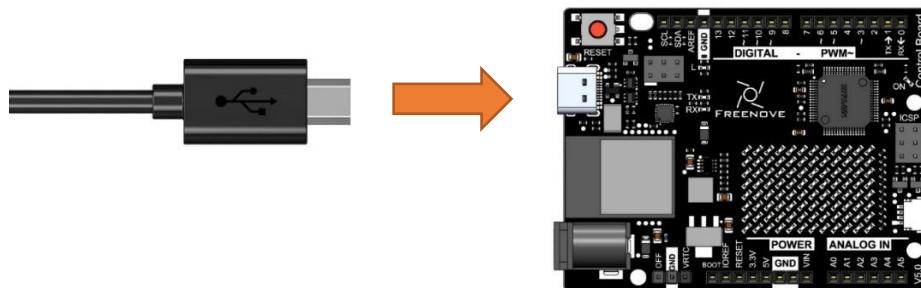
AP mode

When control board selects AP mode, it creates a hotspot network that is separate from the Internet and waits for other WiFi devices to connect. As shown in the figure below, Control board is used as a hotspot. If a mobile phone or PC wants to communicate with control board, it must be connected to the hotspot of control board. Only after a connection is established with control board can they communicate.



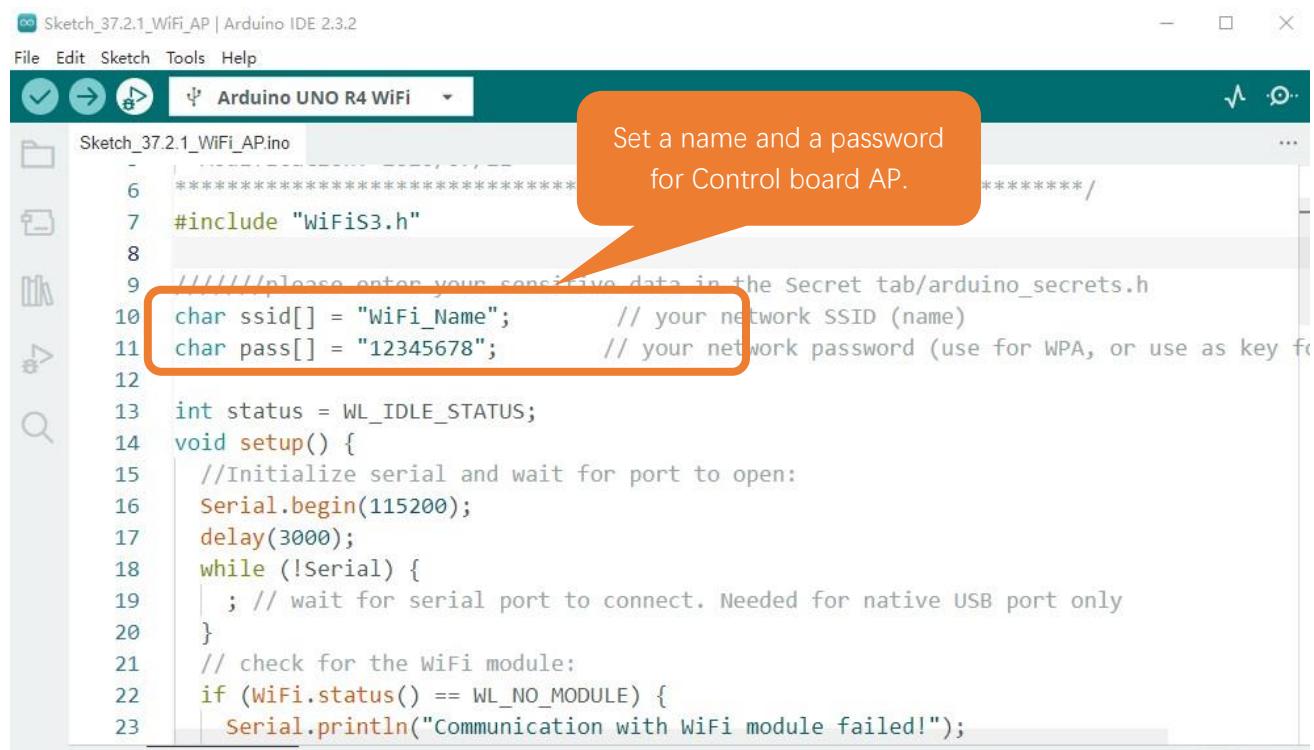
Circuit

Connect Freenove control board to the computer using the USB cable.



Sketch

Sketch_31.2.1



```
Sketch_31.2.1_WiFi_AP | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Arduino UNO R4 WiFi
Sketch_31.2.1_WiFi_APino
6 ****
7 #include "WiFi.h"
8
9 //please enter your sensitive data in the Secret tab/arduino_secrets.h
10 char ssid[] = "WiFi_Name";           // your network SSID (name)
11 char pass[] = "12345678";          // your network password (use for WPA, or use as key for
12
13 int status = WL_IDLE_STATUS;
14 void setup() {
15     //Initialize serial and wait for port to open:
16     Serial.begin(115200);
17     delay(3000);
18     while (!Serial) {
19         ; // wait for serial port to connect. Needed for native USB port only
20     }
21     // check for the WiFi module:
22     if (WiFi.status() == WL_NO_MODULE) {
23         Serial.println("Communication with WiFi module failed!");
```

Set a name and a password
for Control board AP.

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

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

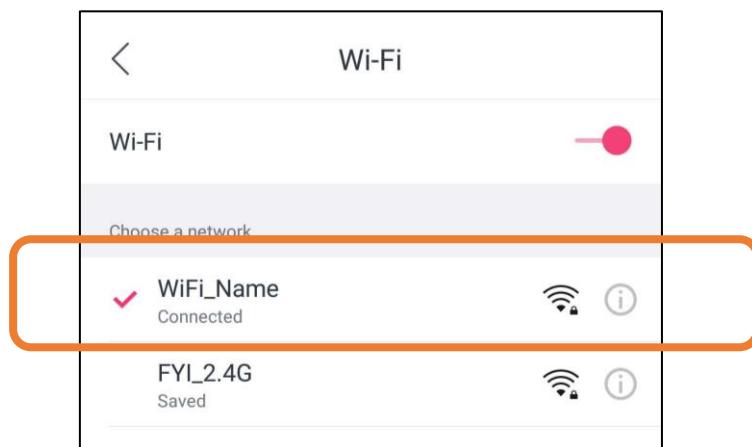


The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor". The message area displays the following text:

```
Creating access point named: WiFi_Name
SSID: WiFi_Name
IP Address: 192.48.56.2
To see this page in action, open a browser to http://192.48.56.2
```

The bottom status bar shows "Ln 56, Col 36" and "Arduino Uno R4 WiFi on COM15".

When observing the print information of the serial monitor, turn on the WiFi scanning function of your phone, and you can see the ssid_AP on control board, which is called "WiFi_Name" in this Sketch. You can enter the password "12345678" to connect it or change its AP name and password by modifying Sketch.



Sketch_37.2_AP_mode

The following is the program code:

```
1 #include "WiFIS3.h"
2
3 //+++++please enter your sensitive data in the Secret
4 tab/arduino_secrets.h
5 char ssid[] = "WiFi_Name";           // your network SSID (name)
6 char pass[] = "12345678";           // your network password (use for WPA, or
7 use as key for WEP)
8
9 int status = WL_IDLE_STATUS;
10 void setup() {
11     //Initialize serial and wait for port to open:
12     Serial.begin(9600);
13     while (!Serial) {
14         ; // wait for serial port to connect. Needed for native USB port only
15     }
16     // check for the WiFi module:
17     if (WiFi.status() == WL_NO_MODULE) {
18         Serial.println("Communication with WiFi module failed!");
19         // don't continue
20         while (true);
21     }
22
23     String fv = WiFi.firmwareVersion();
24     if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
25         Serial.println("Please upgrade the firmware");
26     }
27     // by default the local IP address will be 192.168.4.1
28     // you can override it with the following:
29     WiFi.config(IPAddress(192,48,56,2));
30     // print the network name (SSID);
31     Serial.print("Creating access point named: ");
32     Serial.println(ssid);
33
34     // Create open network. Change this line if you want to create an WEP
35     network:
36     status = WiFi.beginAP(ssid, pass);
37     if (status != WL_AP_LISTENING) {
38         Serial.println("Creating access point failed");
39         // don't continue
40         while (true);
41     }
42 }
```

```

43 // wait 5 seconds for connection:
44 delay(5000);
45 // you're connected now, so print out the status
46 printWiFiStatus();
47 }
48
49 void loop() {
50 }
51
52 void printWiFiStatus() {
53 // print the SSID of the network you're attached to:
54 Serial.print("SSID: ");
55 Serial.println(WiFi.SSID());
56
57 // print your WiFi shield's IP address:
58 IPAddress ip = WiFi.localIP();
59 Serial.print("IP Address: ");
60 Serial.println(ip);
61
62 // print where to go in a browser:
63 Serial.print("To see this page in action, open a browser to http://");
64 Serial.println(ip);
65 }
66 }
```

Include WiFi Library header file of control board.

```
1 #include "WiFi.h"
```

Enter correct AP name and password.

```

5 char ssid[] = "WiFi_Name";           // your network SSID (name)
6 char pass[] = "12345678";           // your network password (use for WPA, or
use as key for WEP)
```

Check whether the AP is turned on successfully. If yes, print out IP and MAC address of AP established by control board. If no, print out the failure prompt.

```

36     status = WiFi.beginAP(ssid, pass);
37     if (status != WL_AP_LISTENING) {
38         Serial.println("Creating access point failed");
39         // don't continue
40         while (true);
41     }
...
43     // wait 5 seconds for connection:
44     delay(5000);
45     // you're connected now, so print out the status
46     printWiFiStatus();
```

```
47 ...
52 void printWiFiStatus() {
53     // print the SSID of the network you're attached to:
54     Serial.print("SSID: ");
55     Serial.println(WiFi.SSID());
56
57     // print your WiFi shield's IP address:
58     IPAddress ip = WiFi.localIP();
59     Serial.print("IP Address: ");
60     Serial.println(ip);
61
62     // print where to go in a browser:
63     Serial.print("To see this page in action, open a browser to http://");
64     Serial.println(ip);
65
66 }
```

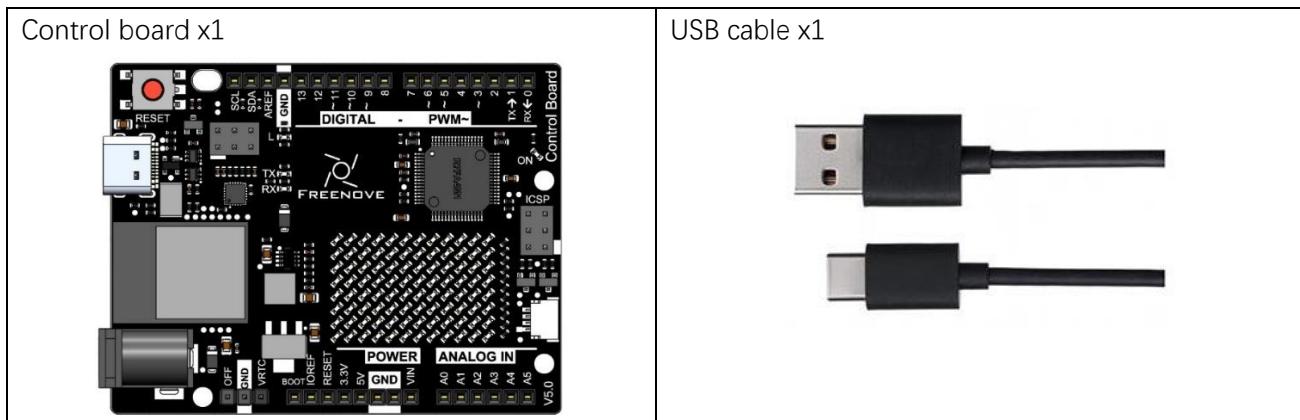
Chapter 32 TCP/IP (WiFi Board)

In this chapter, we will introduce how control board(wifi) implements network communications based on TCP/IP protocol. There are two roles in TCP/IP communication, namely Server and Client, which will be implemented respectively with two projects in this chapter.

Project 32.1 As Client

In this section, control board(wifi) is used as Client to connect Server on the same LAN and communicate with it.

Component List



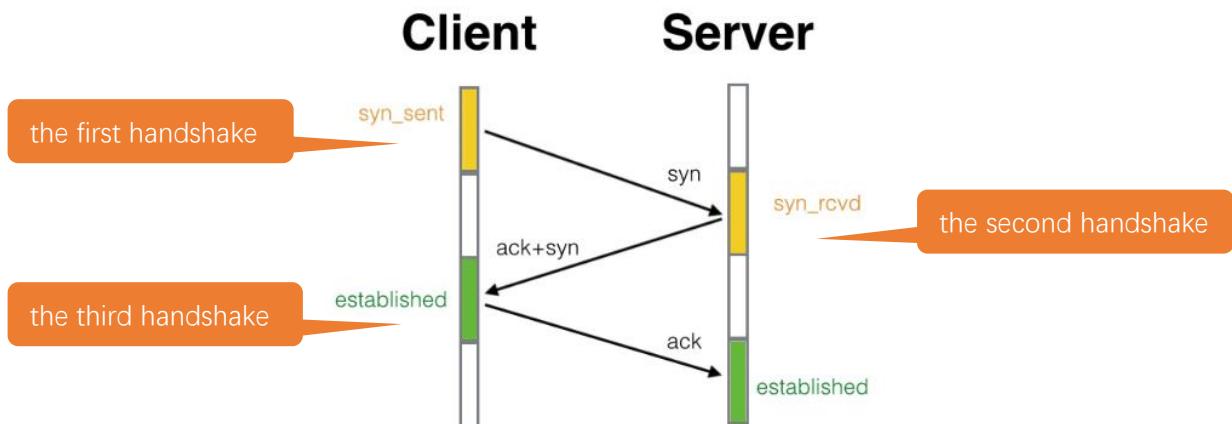
Component knowledge

TCP connection

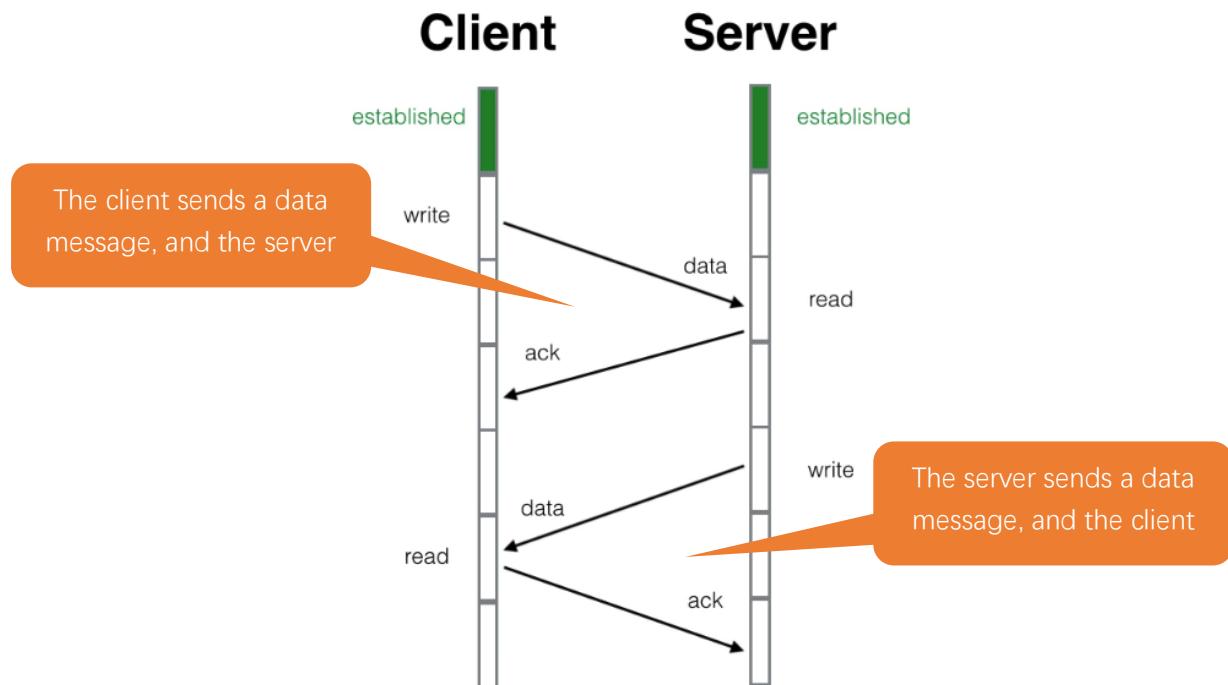
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request to the server, and each time such a request is proposed, a "three-times handshake" is required.

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

The third handshake, the client sends a confirmation message to the server again to confirm the connection.



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



Install Processing

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

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



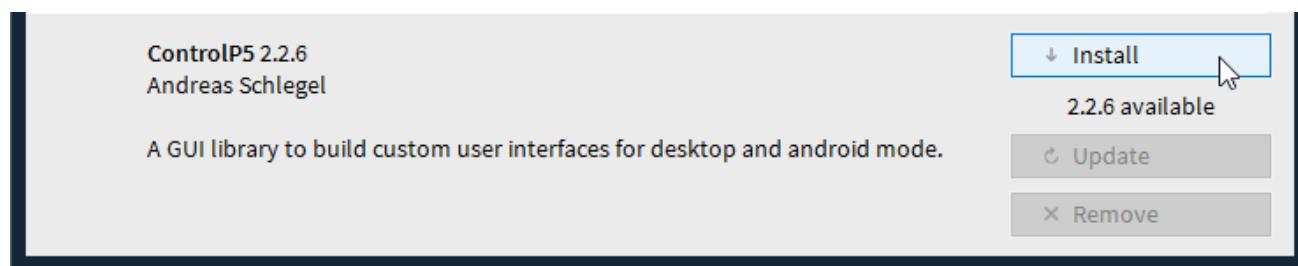
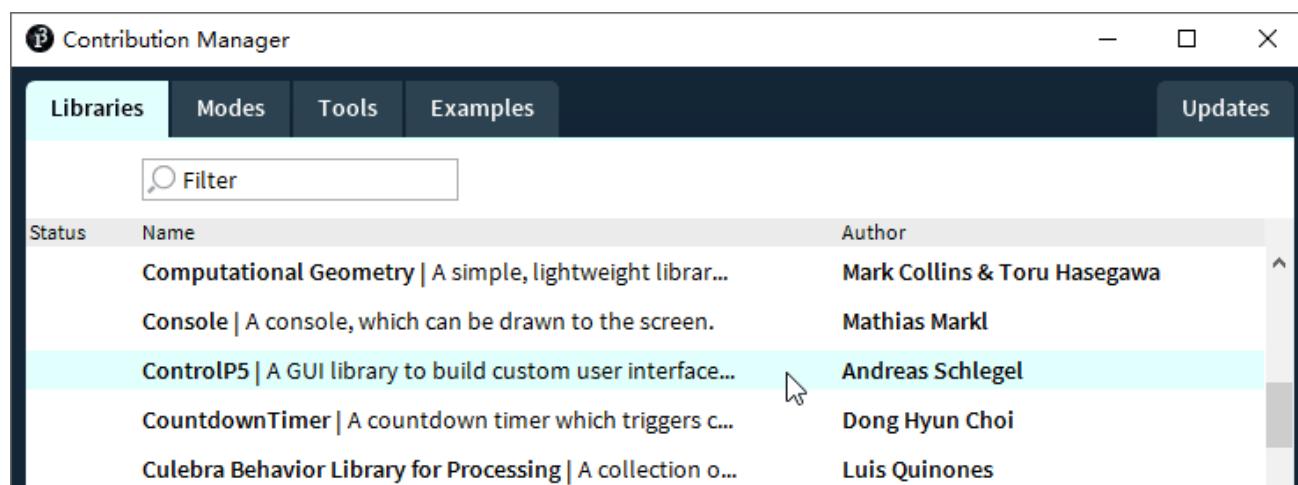
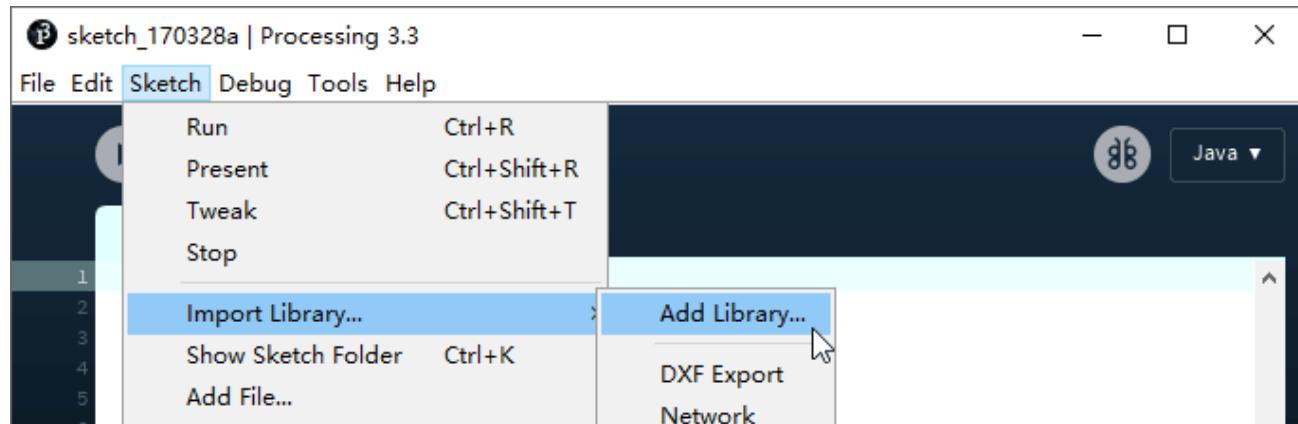
The screenshot shows the official Processing Foundation website. At the top, there's a navigation bar with links for "Processing", "p5.js", "Processing.py", "Processing for Android", "Processing for Pi", and "Processing Foundation". Below the navigation is a large "Processing" logo with a geometric background. To the right is a search bar. On the left side of the main content area, there's a sidebar with links: "Cover", "Download", "Donate", "Exhibition", "Reference", "Libraries", "Tools", "Environment", "Tutorials", "Examples", "Books", "Overview", and "People". The main content area has a heading "Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below." It shows a large "P" logo with a geometric pattern. Below the logo, it says "3.5.4 (17 January 2020)". It provides download links for "Windows 64-bit", "Windows 32-bit", "Linux 64-bit", and "Mac OS X". Under the "Tutorials" section, there are links to "» Github", "» Report Bugs", "» Wiki", "» Supported Platforms", and a note about "changes in 3.0".

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

 core	2020/1/17 12:16
 java	2020/1/17 12:17
 lib	2020/1/17 12:16
 modes	2020/1/17 12:16
 tools	2020/1/17 12:16
 processing.exe	2020/1/17 12:16
 processing-java.exe	2020/1/17 12:16
 revisions.txt	2020/1/17 12:16

Use Server mode for communication

Install ControlP5.



Open the “**Sketches\Sketches\Sketch_37.1_WiFiClient\sketchWiFi\sketchWiFi.pde**”, and click “Run”.

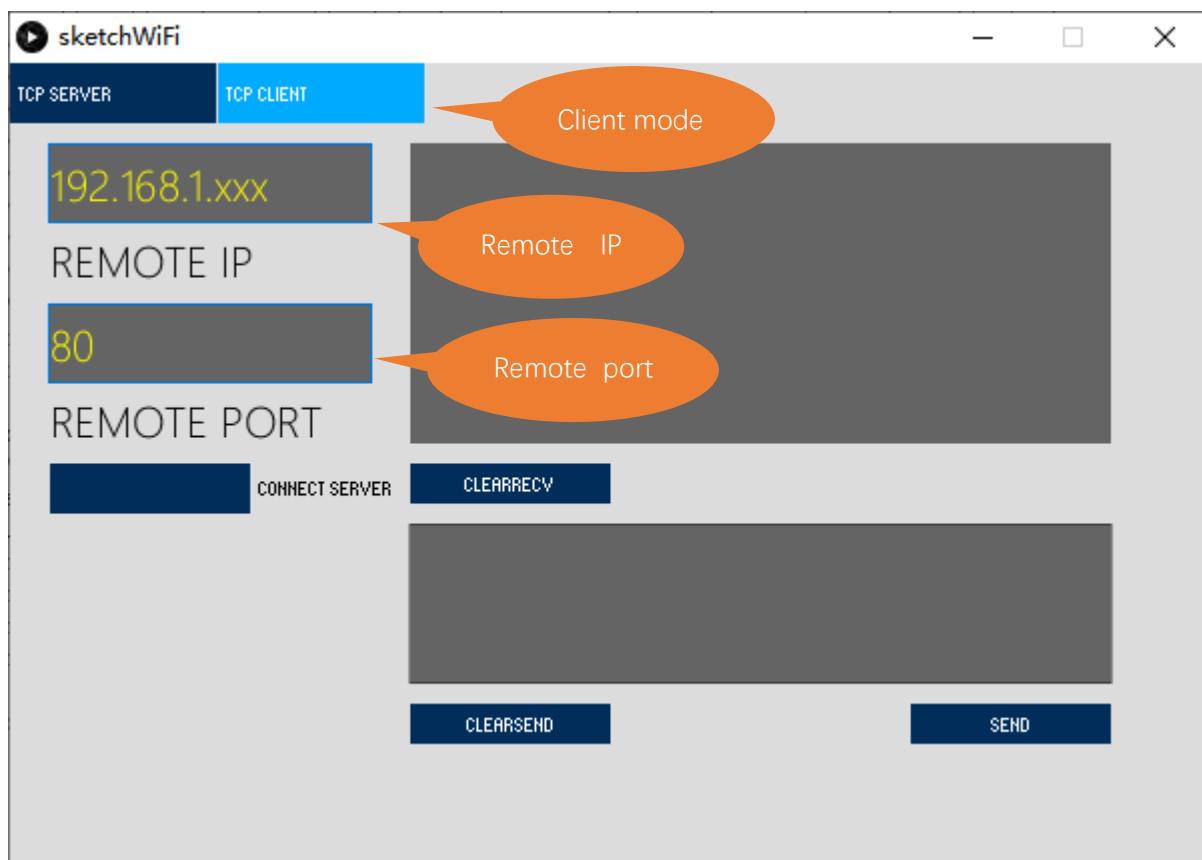


The new pop-up interface is as follows. If control board(wifi) is used as client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, control board(wifi) Sketch needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP32 serves as server, select TCP CLIENT mode for sketchWiFi.



When sketchWiFi selects TCP CLIENT mode, the LOCAL IP and LOCAL PORT of sketchWiFi need to be changed according to the IP address and port number printed by the serial monitor.

Mode selection: select **Server mode/Client mode**.

IP address: In server mode, this option does not need to be filled in, and the computer will automatically obtain the IP address.

In client mode, fill in the remote IP address to be connected.

Port number: In server mode, fill in a port number for client devices to make an access connection.

In client mode, fill in port number given by the Server devices to make an access connection.

Start button: In server mode, push the button, then the computer will serve as server and open a port number for client to make access connection. During this period, the computer will keep monitoring.

In client mode, before pushing the button, please make sure the server is on, remote IP address and remote port number is correct; push the button, and the computer will make access connection to the remote port number of the remote IP as a client.

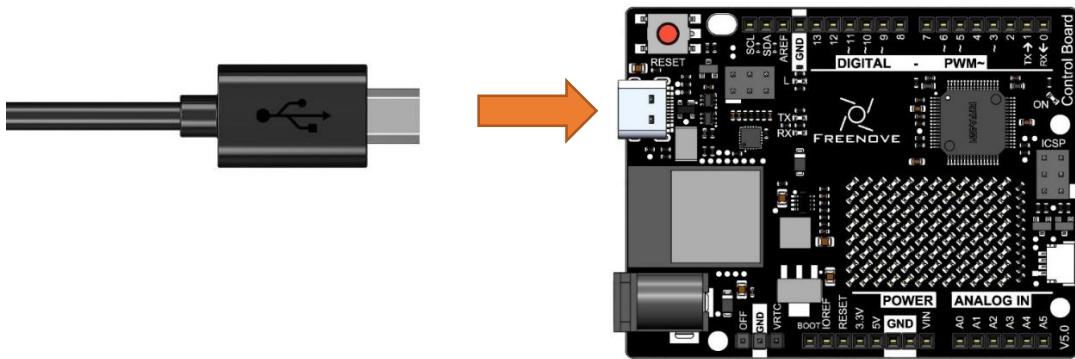
clear receive: clear out the content in the receiving text box

clear send: clear out the content in the sending text box

Sending button: push the sending button, the computer will send the content in the text box to others.

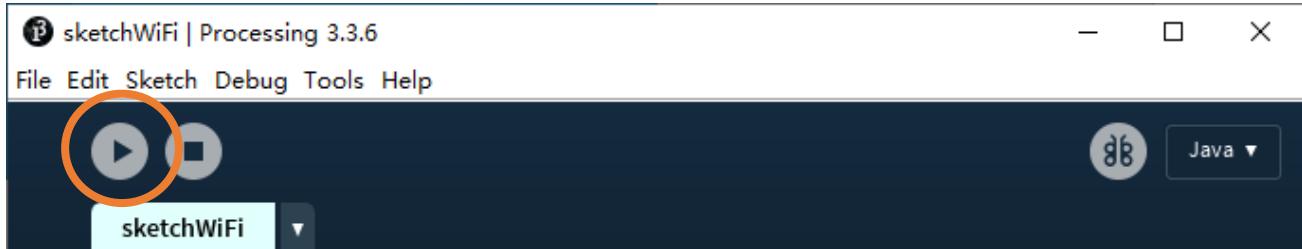
Circuit

Connect Freenove control board to the computer using USB cable.



Sketch

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



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



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

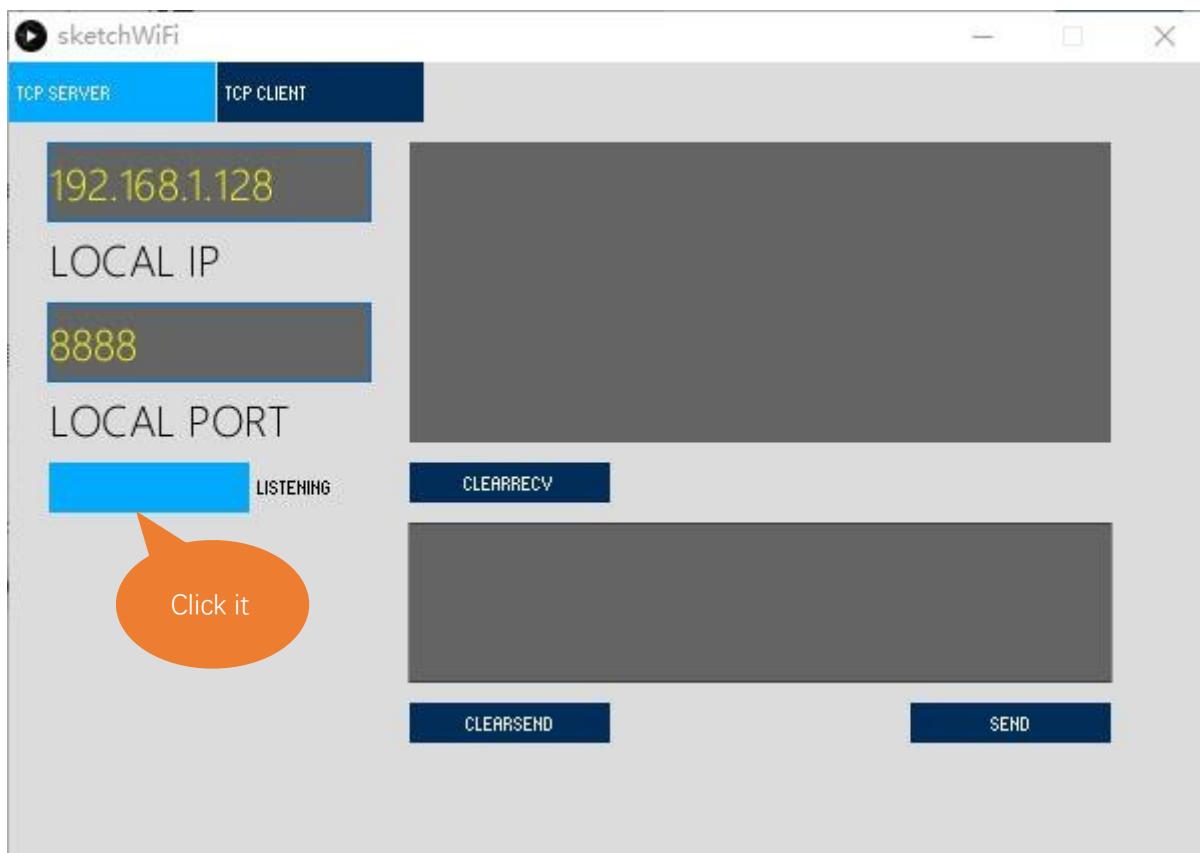
```

Sketch_38.1.1_WiFiClient | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Select Board
Sketch_38.1.1_WiFiClient.ino
7 #include "WiFiS3.h"
8
9 const char *ssid_Router    = "*****"; // Enter the Router name
10 const char *password_Router = "*****"; //Enter the Router password
11 #define REMOTE_IP          "192.168.1.128" //input the remote IP
12 #define REMOTE_PORT         8888           //input the remote port which is the remote server port
13 WiFiClient client;
14

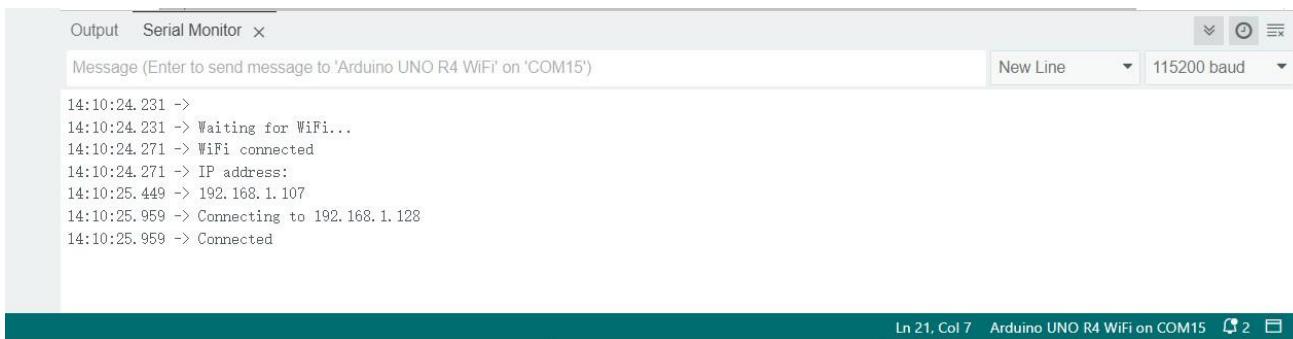
```

REMOTE_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE_IP is “192.168.1.128”. Generally, by default, the ports do not need to change its value.

Click LISTENING, turn on TCP SERVER's data listening function and wait for control board(wifi) to connect.



Compile and upload code to control board(wifi), open the serial monitor and set the baud rate to 115200. control board(wifi) connects router, obtains IP address and sends access request to server IP address on the same LAN till the connection is successful. When connect successfully, control board(wifi) can send messages to server.

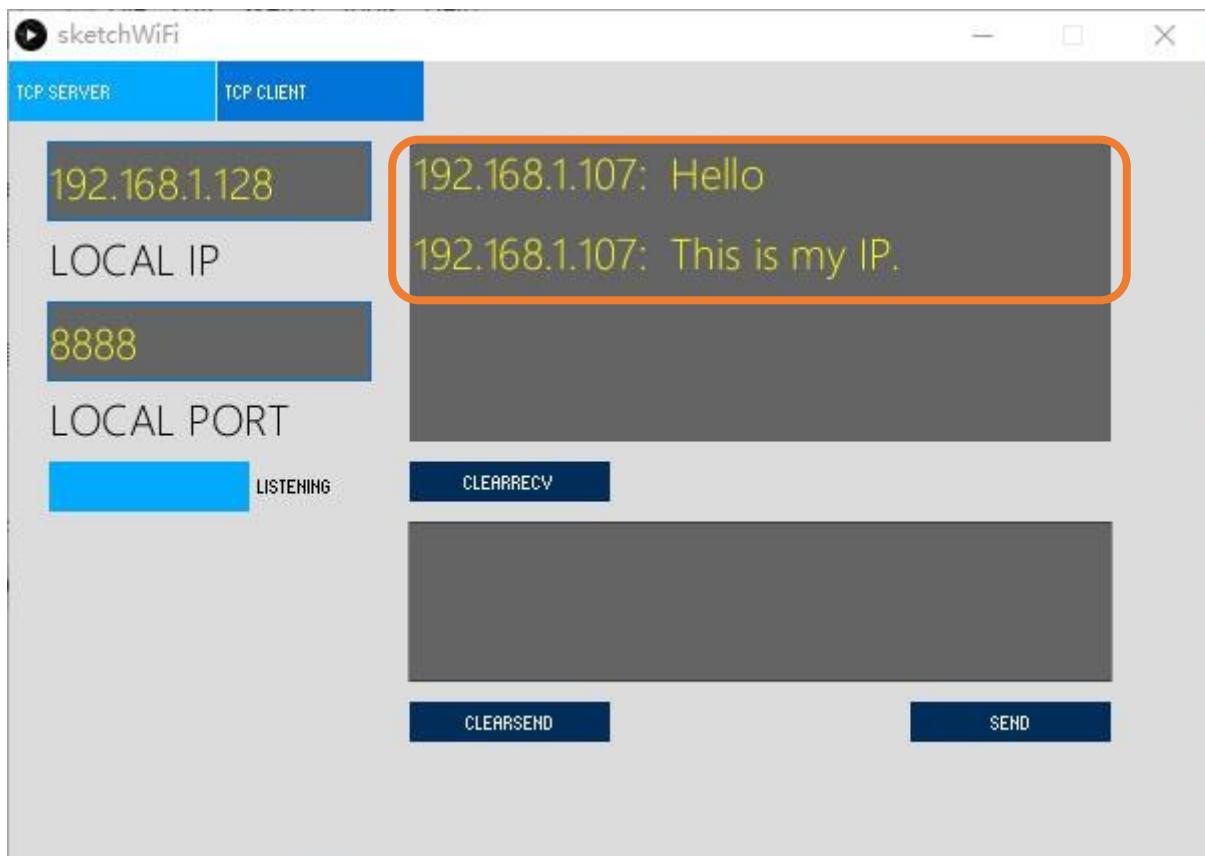


The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor". The message area displays the following log entries:

```
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM15')
14:10:24.231 ->
14:10:24.231 -> Waiting for WiFi...
14:10:24.271 -> WiFi connected
14:10:24.271 -> IP address:
14:10:25.449 -> 192.168.1.107
14:10:25.959 -> Connecting to 192.168.1.128
14:10:25.959 -> Connected
```

The status bar at the bottom right shows "Ln 21, Col 7" and "Arduino UNO R4 WiFi on COM15".

Control board(wifi) connects with TCP SERVER, and TCP SERVER receives messages from control board(wifi), as shown in the figure below.



Sketch_38.1_As_Client

The following is the program code:

```
1 #include "WiFIS3.h"
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5 #define    REMOTE_IP        "*****" //input the remote server which is you want to connect
6 #define    REMOTE_PORT       8888     //input the remote port which is the remote provide
7 WiFiClient client;
8
9 void setup() {
10   Serial.begin(115200);
11   delay(10);
12
13   WiFi.begin(ssid_Router, password_Router);
14   Serial.print("\nWaiting for WiFi... ");
15   while (WiFi.status() != WL_CONNECTED) {
16     Serial.print(".");
17     delay(500);
18   }
19   Serial.println("");
20   Serial.println("WiFi connected");
21   Serial.println("IP address: ");
22   Serial.println(WiFi.localIP());
23   delay(500);
24
25   Serial.print("Connecting to ");
26   Serial.println(REMOTE_IP);
27
28   while (!client.connect(REMOTE_IP, REMOTE_PORT)) {
29     Serial.println("Connection failed.");
30     Serial.println("Waiting a moment before retrying... ");
31   }
32   Serial.println("Connected");
33   client.print("Hello\n");
34   client.print("This is my IP. \n");
35
36 void loop() {
37   if (client.available() > 0) {
38     delay(20);
39     //read back one line from the server
40     String line = client.readString();
41     Serial.println(REMOTE_IP + String(":") + line);
```

```

42 }
43 if (Serial.available() > 0) {
44     delay(20);
45     String line = Serial.readString();
46     client.print(line);
47 }
48 if (client.connected () == 0) {
49     client.stop();
50     WiFi.disconnect();
51 }
52 }
```

Add WiFi function header file.

```
1 #include "WiFiS3.h"
```

Enter the actual router name, password, remote server IP address, and port number.

```

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

Apply for the method class of WiFiClient.

```
7 WiFiClient client;
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

```

13 WiFi.begin(ssid_Router, password_Router);
14 Serial.print("\nWaiting for WiFi... ");
15 while (WiFi.status() != WL_CONNECTED) {
16     Serial.print(".");
17     delay(500);
18 }
```

Send connection request to remote server until connect successfully. When connect successfully, print out the connecting prompt on the serial monitor and send messages to remote server.

```

28 while (!client.connect(REMOTE_IP, REMOTE_PORT)) {//Connect to Server
29     Serial.println("Connection failed.");
30     Serial.println("Waiting a moment before retrying... ");
31 }
32 Serial.println("Connected");
33 client.print("Hello\n");
```

When control board(wifi) receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```

37 if (client.available() > 0) {
38     delay(20);
39     //read back one line from the server
40     String line = client.readString();
41     Serial.println(REMOTE_IP + String(":") + line);
42 }
```

```
43 if (Serial.available() > 0) {  
44     delay(20);  
45     String line = Serial.readString();  
46     client.print(line);  
47 }
```

If the server is disconnected, turn off WiFi of control board.

```
48 if (client.connected () == 0) {  
49     client.stop();  
50     WiFi.disconnect();  
51 }
```

Reference

Class Client

Every time when using Client, you need to include header file "WiFiS3.h."

connect(ip, port, timeout)/connect(*host, port, timeout): establish a TCP connection.

ip, *host: ip address of target server

port: port number of target server

timeout: connection timeout

connected(): judge whether client is connecting. If return value is 1, then connect successfully; If return value is 0, then fail to connect.

stop(): stop tcp connection

print(): send data to server connecting to client

available(): return to the number of bytes readable in receive buffer, if no, return to 0 or -1.

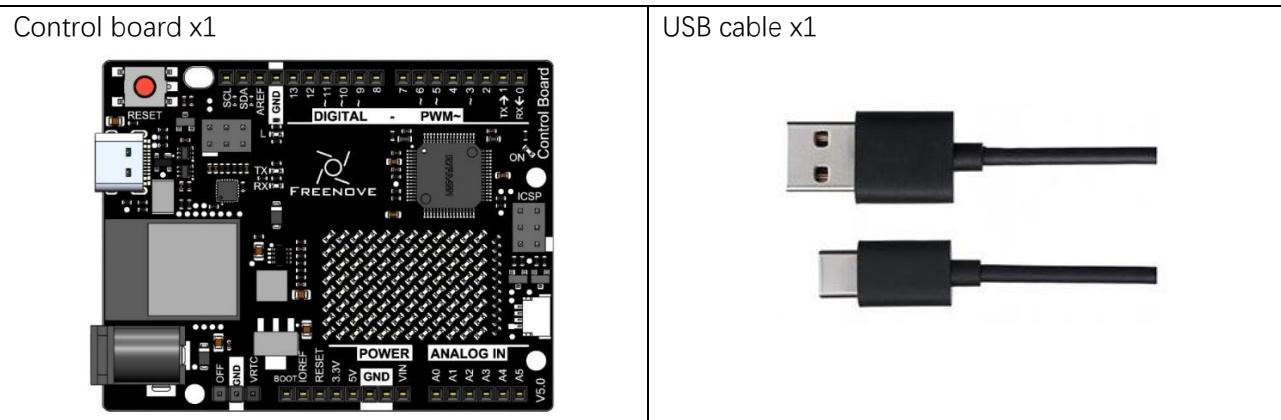
read(): read one byte of data in receive buffer

readString(): read string in receive buffer

Project 32.2 As Server

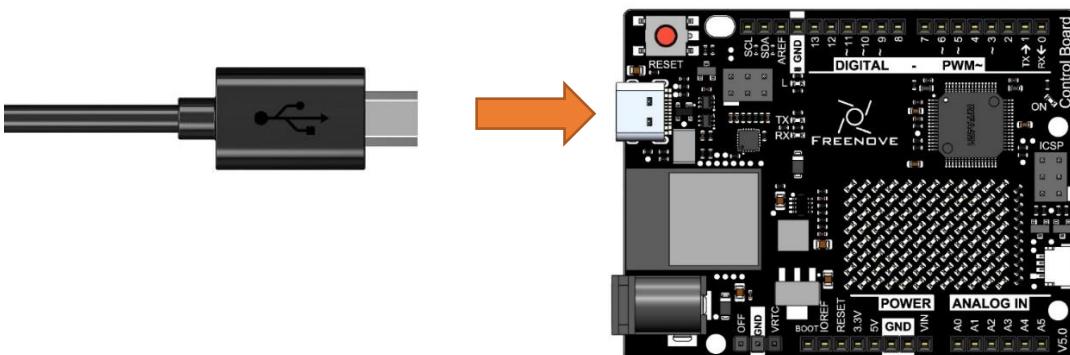
In this section, control board(wifi) is used as a server to wait for the connection and communication of client on the same LAN.

Component List



Circuit

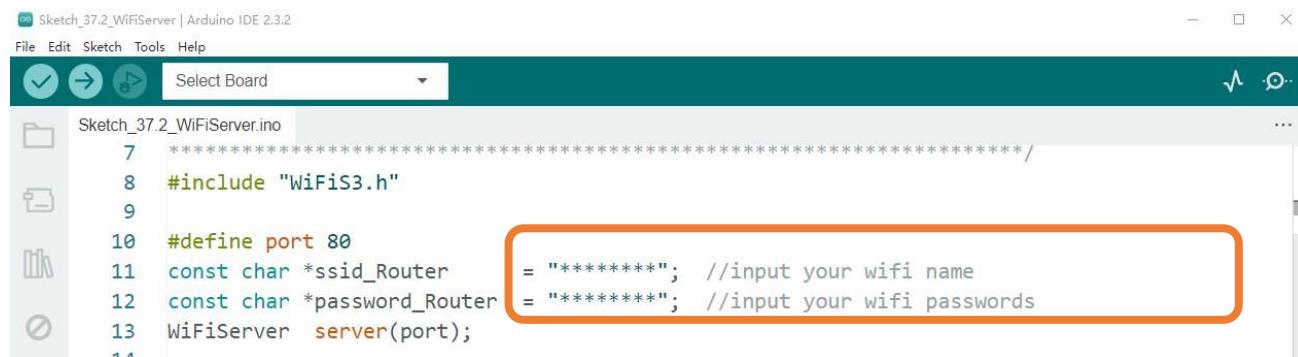
Connect Freenove control board(wifi) to the computer using a USB cable.



Sketch

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

Sketch_32.2.1

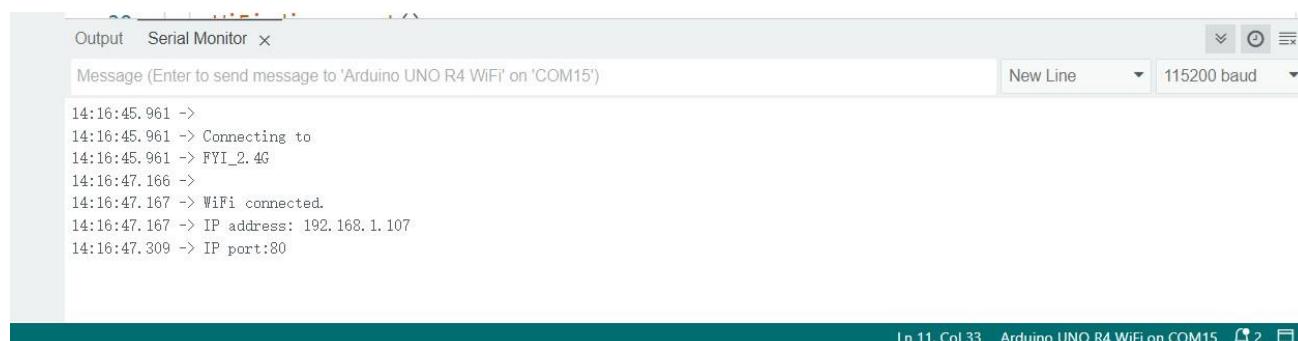


```

Sketch_37.2_WiFiServer | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Select Board
Sketch_37.2_WiFiServer.ino
7 *****
8 #include "WiFiS3.h"
9
10 #define port 80
11 const char *ssid_Router
12 const char *password_Router
13 WiFiServer server(port);
14

```

Compile and upload code to control board(wifi) , open the serial monitor and set the baud rate to 115200. Turn on server mode for control board(wifi), waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other. If the control board(wifi) fails to connect to router, press the reset button as shown below and wait for control board(wifi) to run again.



```

Output Serial Monitor x
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM15')
New Line 115200 baud
14:16:45.961 ->
14:16:45.961 -> Connecting to
14:16:45.961 -> FYI_2.4G
14:16:47.166 ->
14:16:47.167 -> WiFi connected.
14:16:47.167 -> IP address: 192.168.1.107
14:16:47.309 -> IP port:80

```

Ln 11, Col 33 Arduino UNO R4 WiFi on COM15 2

Serial Monitor



```

Output Serial Monitor x
Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM15')
New Line 115200 baud
14:16:45.961 ->
14:16:45.961 -> Connecting to
14:16:45.961 -> FYI_2.4G
14:16:47.166 ->
14:16:47.167 -> WiFi connected.
14:16:47.167 -> IP address: 192.168.1.107
14:16:47.309 -> IP port:80

```

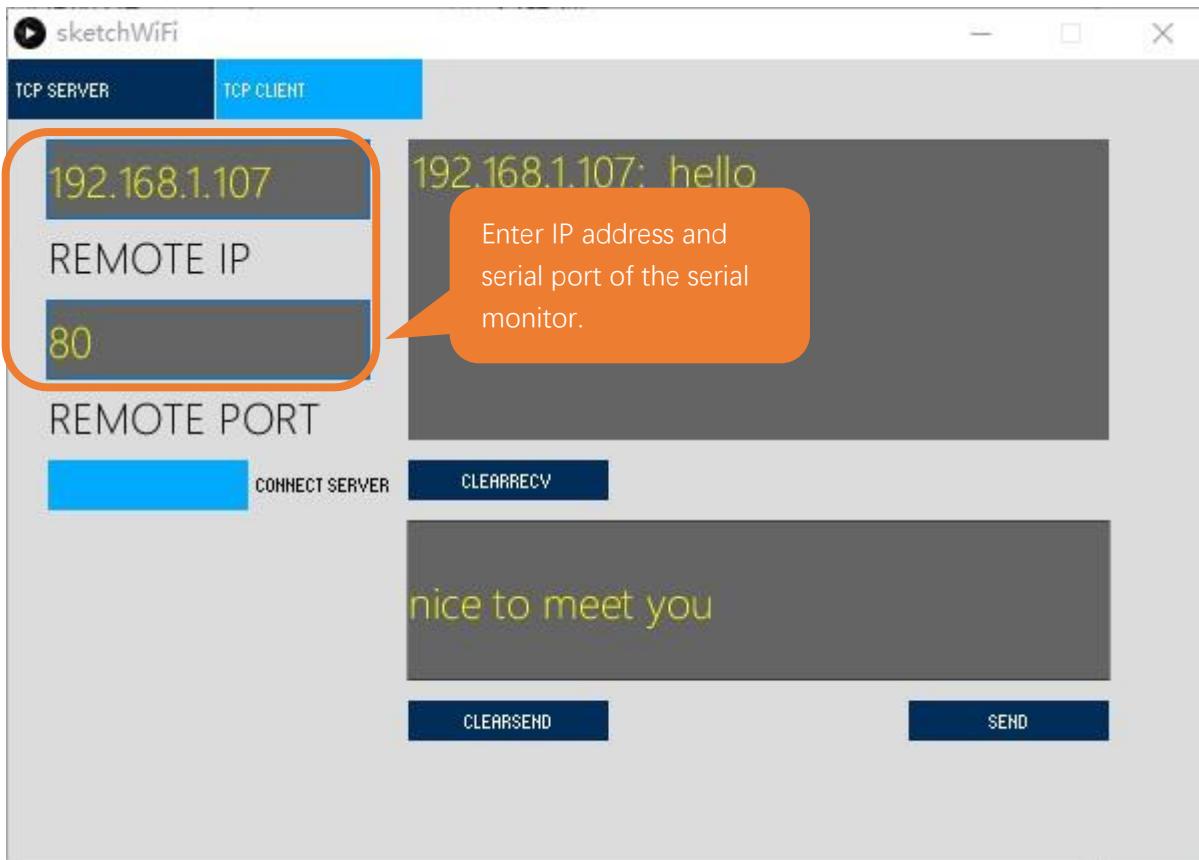
IP address and
serial port

Ln 11, Col 33 Arduino UNO R4 WiFi on COM15 2

Processing:

Open the “**Sketches\Sketch_31.2.1_WiFiServer\sketchWiFi\sketchWiFi.pde**”.

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



The following is the program code:

```

1 #include "WiFIS3.h"
2
3 #define port 80
4 const char *ssid_Router      = "*****"; //input your wifi name
5 const char *password_Router = "*****"; //input your wifi passwords
6 WiFiServer server(port);
7
8 void setup()
9 {
10    Serial.begin(115200);
11    Serial.printf("\nConnecting to ");
12    Serial.println(ssid_Router);
13    WiFi.disconnect();
14    WiFi.begin(ssid_Router, password_Router);
15    delay(1000);
16    while (WiFi.status() != WL_CONNECTED) {
17        delay(500);
18        Serial.print(".");
19    }
20    Serial.println("");
21    Serial.println("WiFi connected.");

```

```

22   Serial.print("IP address: ");
23   Serial.println(WiFi.localIP());
24   Serial.printf("IP port: %d\n", port);
25   server.begin(port);
26   WiFi.setAutoConnect(true);
27   WiFi.setAutoReconnect(true);
28 }
29
30 void loop() {
31   WiFiClient client = server.available();           // listen for incoming clients
32   if (client) {                                     // if you get a client
33     Serial.println("Client connected.");
34     while (client.connected()) {                   // loop while the client's connected
35       if (client.available()) {                   // if there's bytes to read from the
36         Serial.println(client.readStringUntil('\n')); // print it out the serial monitor
37         while(client.read()>0);                  // clear the wifi receive area cache
38       }
39       if(Serial.available()){                     // if there's bytes to read from the
39         Serial.println("Client Disconnected.");
40       }
41     }
42     client.stop();                                // stop the client connecting.
43   }
44 }
```

Apply for method class of WiFiServer.

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

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

13	WiFi.disconnect(); 14 WiFi.begin(ssid_Router, password_Router); 15 delay(1000); 16 while (WiFi.status() != WL_CONNECTED) { 17 delay(500); 18 Serial.print("."); 19 } 20 Serial.println(""); 21 Serial.println("WiFi connected.");
----	---

Print out the IP address and port number of control board(wifi).

22	Serial.print("IP address: "); 23 Serial.println(WiFi.localIP()); //print out IP address of ESP32
----	--

```
24   Serial.printf("IP port: %d\n", port);           //Print out ESP32's port number
```

Turn on server mode of control board(wifi), start automatic connection and turn on automatic reconnection.

```
25   server.begin();                                //Turn ON ESP32 as Server mode
26   WiFi.setAutoConnect(true);
27   WiFi.setAutoReconnect(true);
```

When control board(wifi) receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```
35   if (client.available()) {                      // if there's bytes to read from the
    client
36       Serial.println(client.readStringUntil('\n'));// print it out the serial monitor
37       while(client.read()>0);                  // clear the wifi receive area cache
38   }
39   if(Serial.available()){                        // if there's bytes to read from the
    serial monitor
40       client.print(Serial.readStringUntil('\n'));// print it out the client.
41       while(Serial.read()>0);                  // clear the wifi receive area cache
42   }
```

Reference

Class Server

Every time use Server functionality, we need to include header file "WiFiS3.h".

WiFiServer(uint16_t port=80, uint8_t max_clients=4): create a TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 80.

max_clients: maximum number of clients with default number as 4.

begin(port): start the TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 0.

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

nodelay: true stands for forbidden Nagle algorithm.

close(): close tcp connection.

stop(): stop tcp connection.

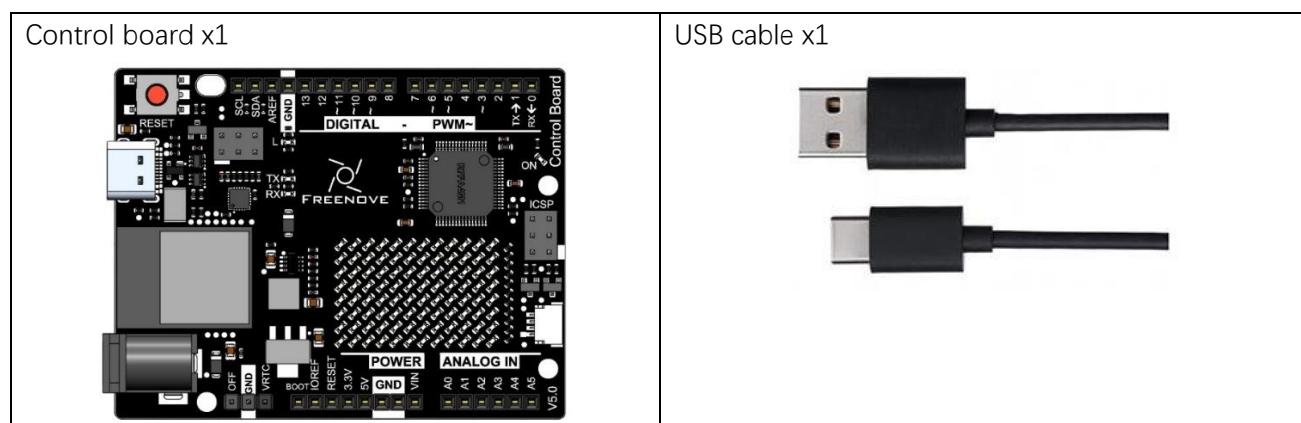
Chapter 33 Control LED with Web (WiFi Board)

In this chapter, we will use control board to make a simple smart home. We will learn how to control LED lights through web pages.

Project 33.1 Control the LED with Web

In this project, we need to build a Web Service and then use the control board to control the LED through the Web browser of the phone or PC. Through this example, you can remotely control the appliances in your home to achieve smart home.

Component List



Component knowledge

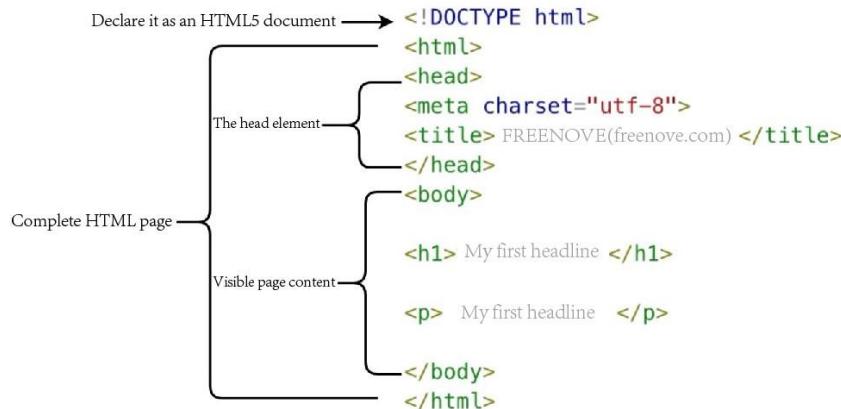
HTML

HyperText Markup Language (HTML) is a standard Markup Language for creating web pages. It includes a set of tags that unify documents on the network and connect disparate Internet resources into a logical whole. HTML text is descriptive text composed of HTML commands that describe text, graphics, animations, sounds, tables, links, etc. The extension of the HTML file is HTM or HTML. Hyper Text is a way to organize information. It uses hyperlinks to associate words and charts in Text with other information media. These related information media may be in the same Text, other files, or files located on a remote computer. This way of organizing information connects the information resources distributed in different places, which is convenient for people to search and retrieve information.

The nature of the Web is hypertext Markup Language (HTML), which can be combined with other Web technologies (e.g., scripting languages, common gateway interfaces, components, etc.) to create powerful Web pages. Thus, HYPERtext Markup Language (HTML) is the foundation of World Wide Web (Web) programming, that is, the World Wide Web is based on hypertext. Hypertext Markup Language is called hypertext Markup language because the text contains so-called "hyperlink" points.

You can build your own WEB site using HTML, which runs on the browser and is parsed by the browser.

Example analysis is shown in the figure below:



<!DOCTYPE html>: Declare it as an HTML5 document

<html>: Is the root element of an HTML page

<head>: Contains meta data for the document, such as < meta charset="utf-8" > Define the web page encoding format to UTF-8.

<title>: Notes the title of the document

<body>: Contains visible page content

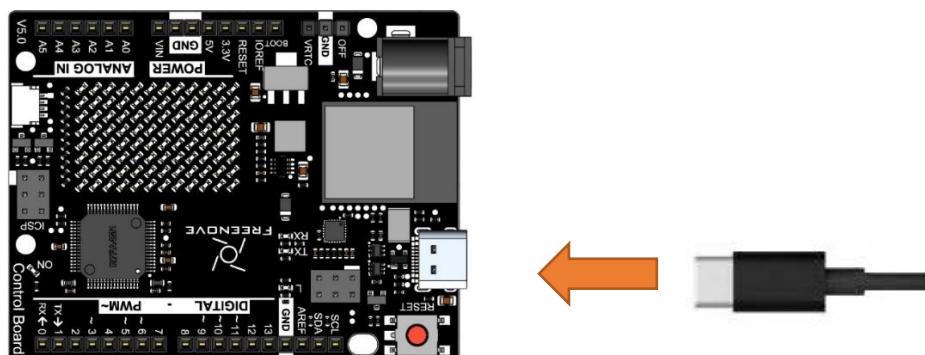
<h1>: Define a big heading

<p>: Define a paragraph

For more information, please visit: <https://developer.mozilla.org/en-US/docs/Web/HTML>

Circuit

Connect the board to the computer using the USB cable.



Sketch

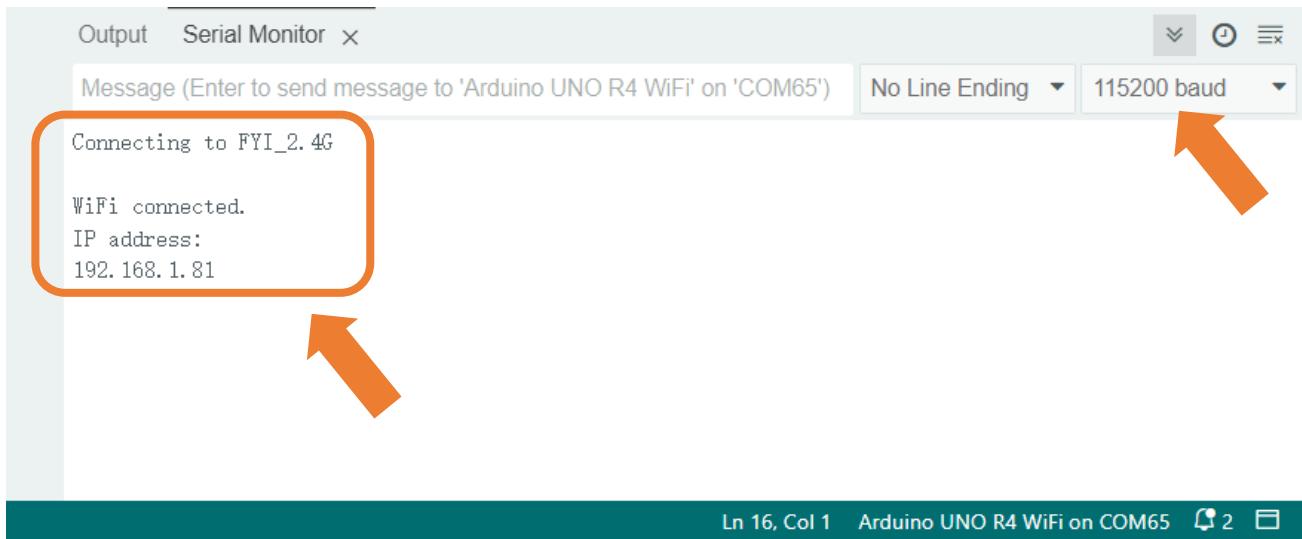
Sketch_33.1.1

The screenshot shows the Arduino IDE interface with the sketch `Sketch_33.1.1` open. The code is as follows:

```
11  *****/
12 #include "WiFiS3.h"
13
14 // Replace with your network credentials
15 char* ssid      = "*****";
16 char* password = "*****".
17
18 // Set web server port number to 80
19 WiFiServer server(80);
20 // Variable to store the HTTP request
21 String header;
22 // Auxiliar variables to store the current output state
23 String PIN_LEDState = "OFF";
24
25 // Current time
26 unsigned long currentTime = millis();
27 // Previous time
28 unsigned long previousTime = 0;
29 // Define timeout time in milliseconds (example: 2000ms = 2s)
30 const long timeoutTime = 2000;
31
```

A callout bubble with the text "Enter the correct Router name and password." points to the line `char* ssid = "*****";`.

Upload the code to the control board, open the serial monitor and set the board rate to 115200. After the board connects to WiFi successfully, the IP address will be printed out, as shown below.



When Control Board successfully connects to "ssid_Router", serial monitor will print out the IP address assigned to Control Board by the router. Access <http://192.168.1.26> in a computer browser on the LAN. As shown in the following figure:



You can click the corresponding button to control the LED on and off.

The following is the program code:

```

1 #include "WiFiS3.h"
2
3 // Replace with your network credentials
4 char* ssid      = "*****";
5 char* password = "*****";

```

```
6 // Set web server port number to 80
7 WiFiServer server(80);
8 // Variable to store the HTTP request
9 String header;
10 // Auxiliar variables to store the current output state
11 String PIN_LEDState = "OFF";
12
13 // Current time
14 unsigned long currentTime = millis();
15 // Previous time
16 unsigned long previousTime = 0;
17 // Define timeout time in milliseconds (example: 2000ms = 2s)
18 const long timeoutTime = 2000;
19
20
21 void setup() {
22   Serial.begin(115200);
23   // Initialize the output variables as outputs
24   pinMode(LED_BUILTIN, OUTPUT);
25   digitalWrite(LED_BUILTIN, LOW);
26
27   // Connect to Wi-Fi network with SSID and password
28   Serial.print("Connecting to ");
29   Serial.println(ssid);
30   WiFi.begin(ssid, password);
31   while (WiFi.status() != WL_CONNECTED) {
32     delay(500);
33     Serial.print(".");
34   }
35   // Print local IP address and start web server
36   Serial.println("");
37   Serial.println("WiFi connected.");
38   Serial.println("IP address: ");
39   Serial.println(WiFi.localIP());
40   server.begin();
41 }
42
43 void loop() {
44   WiFiClient client = server.available(); // Listen for incoming clients
45   if (client) { // If a new client connects,
46     Serial.println("New Client.");
47     String currentLine = "";
48     client
```

```
      currentTime = millis();
      previousTime = currentTime;
```

```
49  while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while
50    the client's connected
51    currentTime = millis();
52    if (client.available()) { // if there's bytes to read from the client,
53      char c = client.read(); // read a byte, then
54      Serial.write(c); // print it out the serial monitor
55      header += c;
56      if (c == '\n') { // if the byte is a newline character
57        // if the current line is blank, you got two newline characters in a row.
58        // that's the end of the client HTTP request, so send a response:
59        if (currentLine.length() == 0) {
60          // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
61          // and a content-type so the client knows what's coming, then a blank line:
62          client.println("HTTP/1.1 200 OK");
63          client.println("Content-type:text/html");
64          client.println("Connection: close");
65          client.println();
66          // turns the GPIOs on and off
67          if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
68            Serial.println("LED_BUILTIN ON");
69            PIN_LEDState = "ON";
70            digitalWrite(LED_BUILTIN, HIGH);
71          } else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
72            Serial.println("LED_BUILTIN OFF");
73            PIN_LEDState = "OFF";
74            digitalWrite(LED_BUILTIN, LOW);
75          }
76          // Display the HTML web page
77          client.println("<!DOCTYPE html><html>");
78          client.println("<head> <title>Control Board Web Server</title> <meta");
79          name="viewport" content="width=device-width, initial-scale=1\"");
80          // CSS to style the on/off buttons
81          // Feel free to change the background-color and font-size attributes to fit your
82          preferences
83          client.println("<style>html {font-family: Helvetica; display:inline-block; margin:");
84          0px auto; text-align: center;}");
85          client.println(" h1{color: #0F3376; padding: 2vh;} p{font-size: 1.5rem;}");
86          client.println(".button{background-color: #4286f4; display: inline-block; border:");
87          none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:");
88          30px; margin: 2px; cursor: pointer;}");
89          client.println(".button2{background-color: #4286f4;display: inline-block; border:");
90          none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:");
91          30px; margin: 2px; cursor: pointer;}</style></head>");
```

```

    // Web Page Heading
    client.println("<body><h1>Control Board Web Server</h1>");
    client.println("<p>GPIO state: " + PIN_LEDState + "</p>");
    client.println("<p><a href=\"/LED_BUILTIN/ON\"><button class=\"button button2\">ON</button></a></p>" );
    client.println("<p><a href=\"/LED_BUILTIN/OFF\"><button class=\"button button2\">OFF</button></a></p>" );
    client.println("</body></html>");
    // The HTTP response ends with another blank line
    client.println();
    // Break out of the while loop
    break;
} else { // if you got a newline, then clear currentLine
    currentLine = "";
}
} else if (c != '\r') { // if you got anything else but a carriage return character,
    currentLine += c;      // add it to the end of the currentLine
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}
}
// Replace with your network credentials
char* ssid      = "*****";
char* password = "*****";

```

Include the WiFi Library header file of Control Board.

```
1 #include "WiFi.h"
```

Enter correct router name and password.

```
3 // Replace with your network credentials
4 char* ssid      = "*****";
5 char* password = "*****";
```

Set Control Board in Station mode and connect it to your router.

```
30 WiFi.begin(ssid, password);
```

Check whether Control Board has connected to router successfully every 0.5s.

```
31 while (WiFi.status() != WL_CONNECTED) {
32     delay(500);
33     Serial.print(".");
}
```

34	}
----	---

Serial monitor prints out the IP address assigned to Control Board.

39	Serial.println(WiFi.localIP());
----	---------------------------------

Click the button on the web page to control the LED light on and off.

65	// turns the GPIOs on and off
66	if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
67	Serial.println("LED_BUILTIN ON");
68	PIN_LEDState = "ON";
69	digitalWrite(LED_BUILTIN, HIGH);
70	} else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
71	Serial.println("LED_BUILTIN OFF");
72	PIN_LEDState = "OFF";
73	digitalWrite(LED_BUILTIN, LOW);
74	}

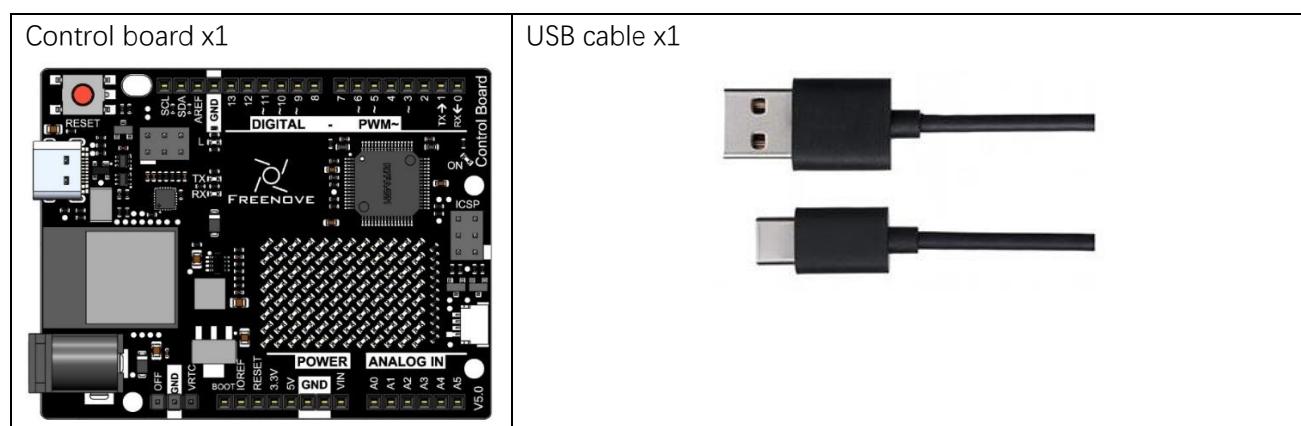
Chapter 34 Bluetooth (WiFi Board)

In this chapter, we engage with the onboard Bluetooth module. Bluetooth and Wi-Fi are common wireless communication technologies.

Project 34.1 Bluetooth Low Energy Data Passthrough

In this section, we first learn how to make simple data transmission between the control board (Bluetooth) and our phone.

Component List



Component knowledge

Control board's integrated Bluetooth function Bluetooth is a short-distance communication system, which can be divided into two types, namely Bluetooth Low Energy(BLE) and Classic Bluetooth. There are two modes for simple data transmission: master mode and slave mode.

Master mode

In this mode, works are done in the master device and it can connect with a slave device. And we can search and select slave devices nearby to connect with. When a device initiates connection request in master mode, it requires information of the other Bluetooth devices including their address and pairing passkey. After finishing pairing, it can connect with them directly.

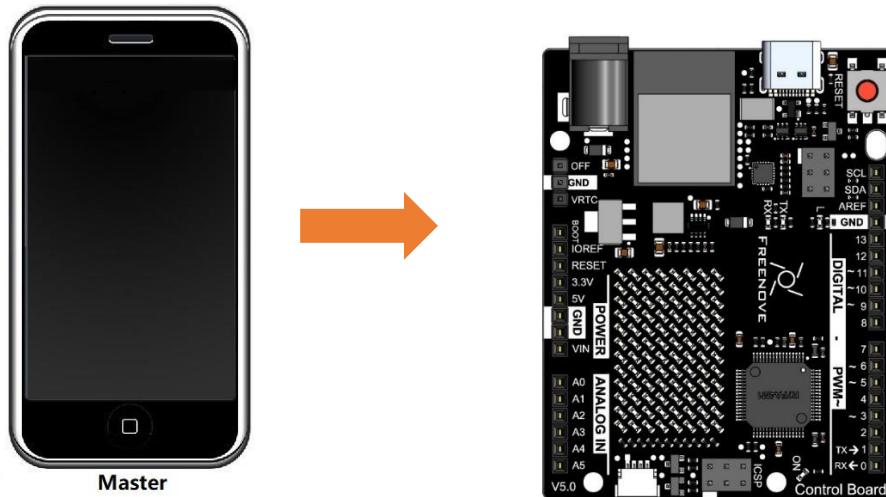
Slave mode

The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

Bluetooth devices can make data interaction with each other, as one is in master mode and the other in slave mode. When they are making data interaction, the Bluetooth device in master mode searches and selects devices nearby to connect to. When establishing connection, they can exchange data. When mobile phones exchange data with Control board, they are usually in master mode and Control board in slave mode.

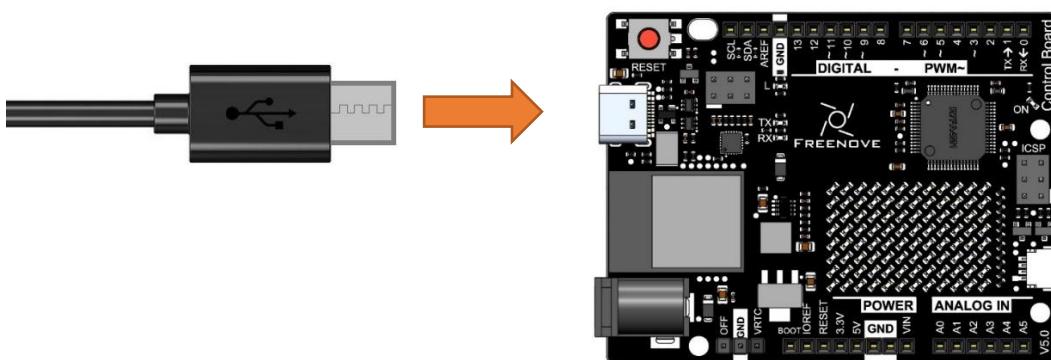
The control board is equipped with an ESP32 module, which provides Bluetooth and Wi-Fi functions, supporting speeds up to 2Mbps. The ESP32 module integrates a tracking antenna, which allows you to take advantage of the development board's connectivity without an external antenna.

However, It is worth noting that the Wi-Fi and Bluetooth modules share the same antenna, which means you cannot use Bluetooth and Wi-Fi at the same time.



Circuit

Connect Freenove Control board to the computer using the USB cable.

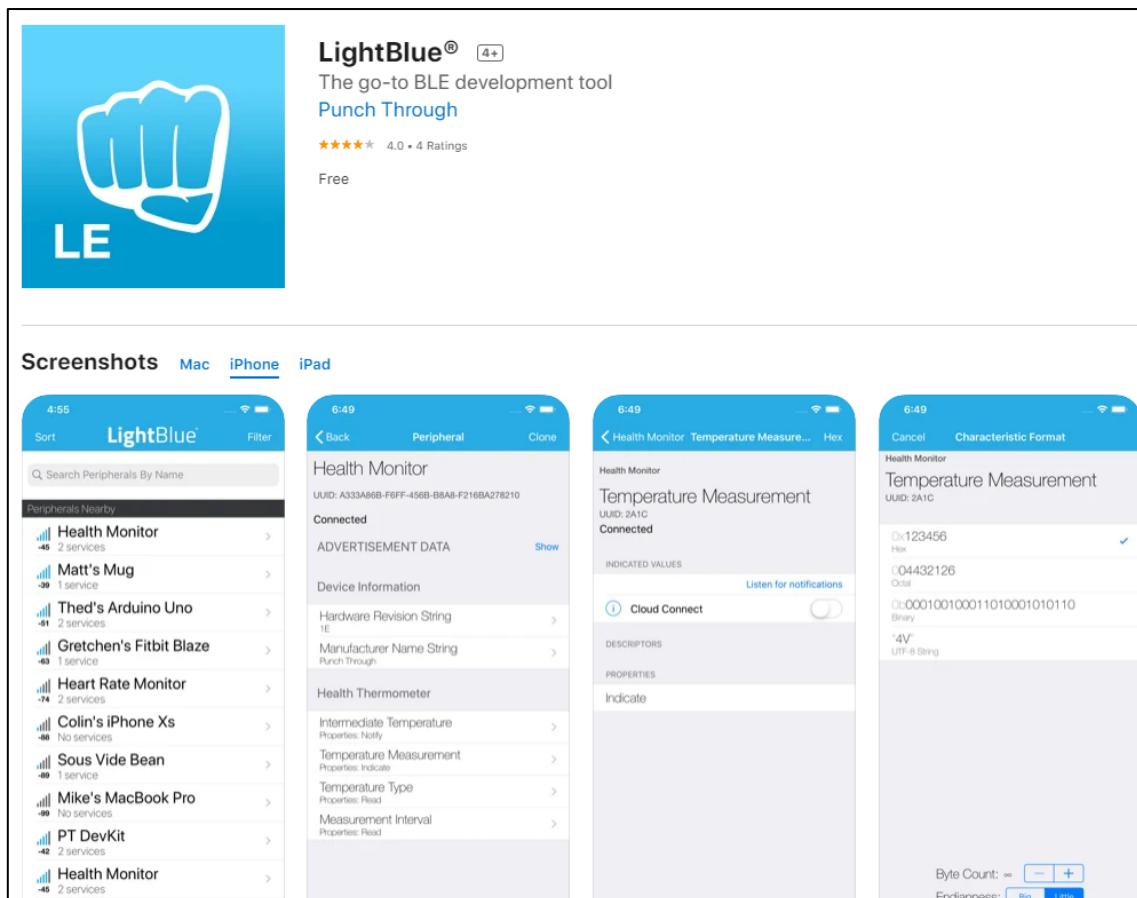


Sketch

Lightblue

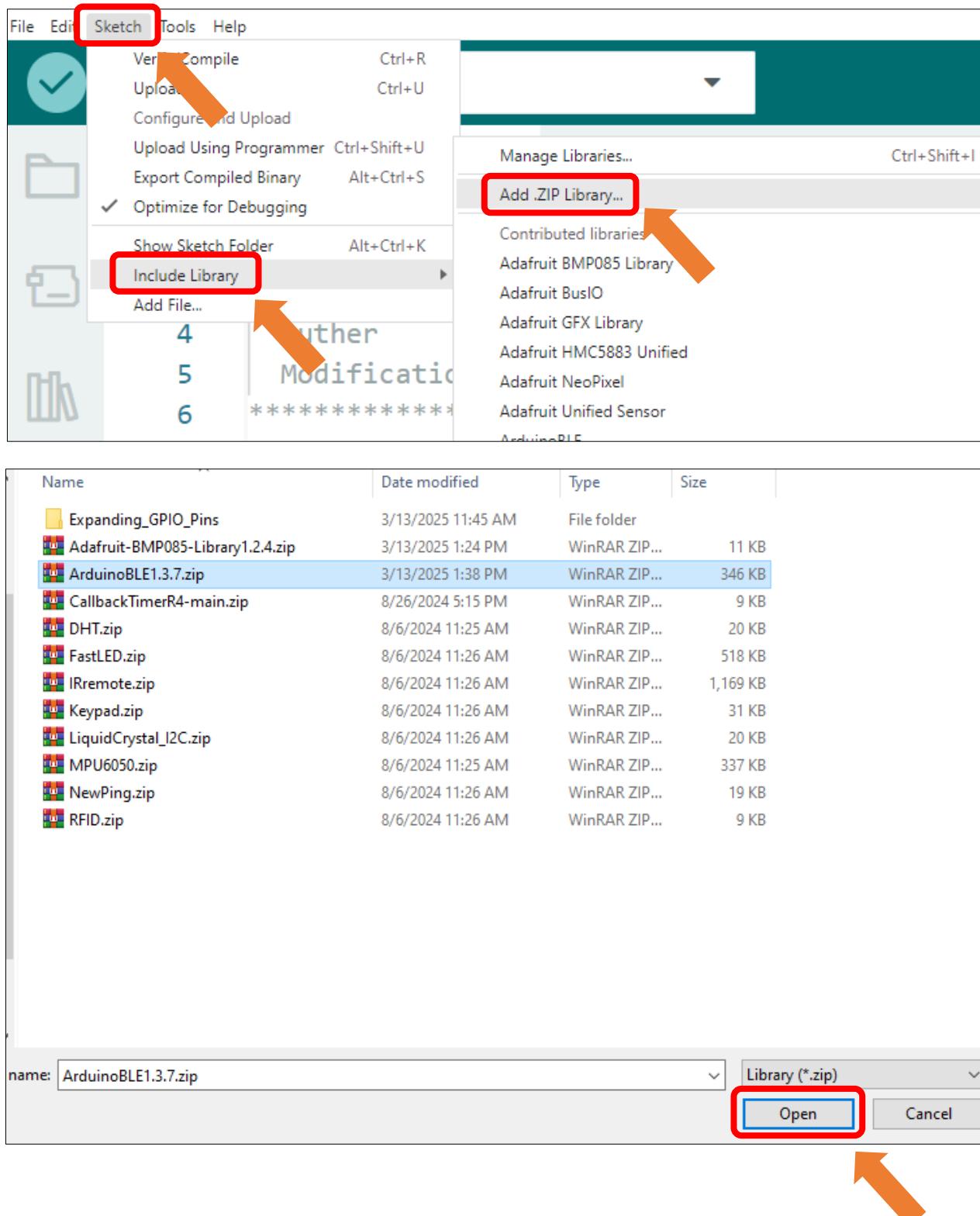
If you can't install Serial Bluetooth on your phone, try LightBlue. If you do not have this software installed on your phone, you can refer to this link:

<https://apps.apple.com/us/app/lightblue/id557428110#?platform=iphone.>



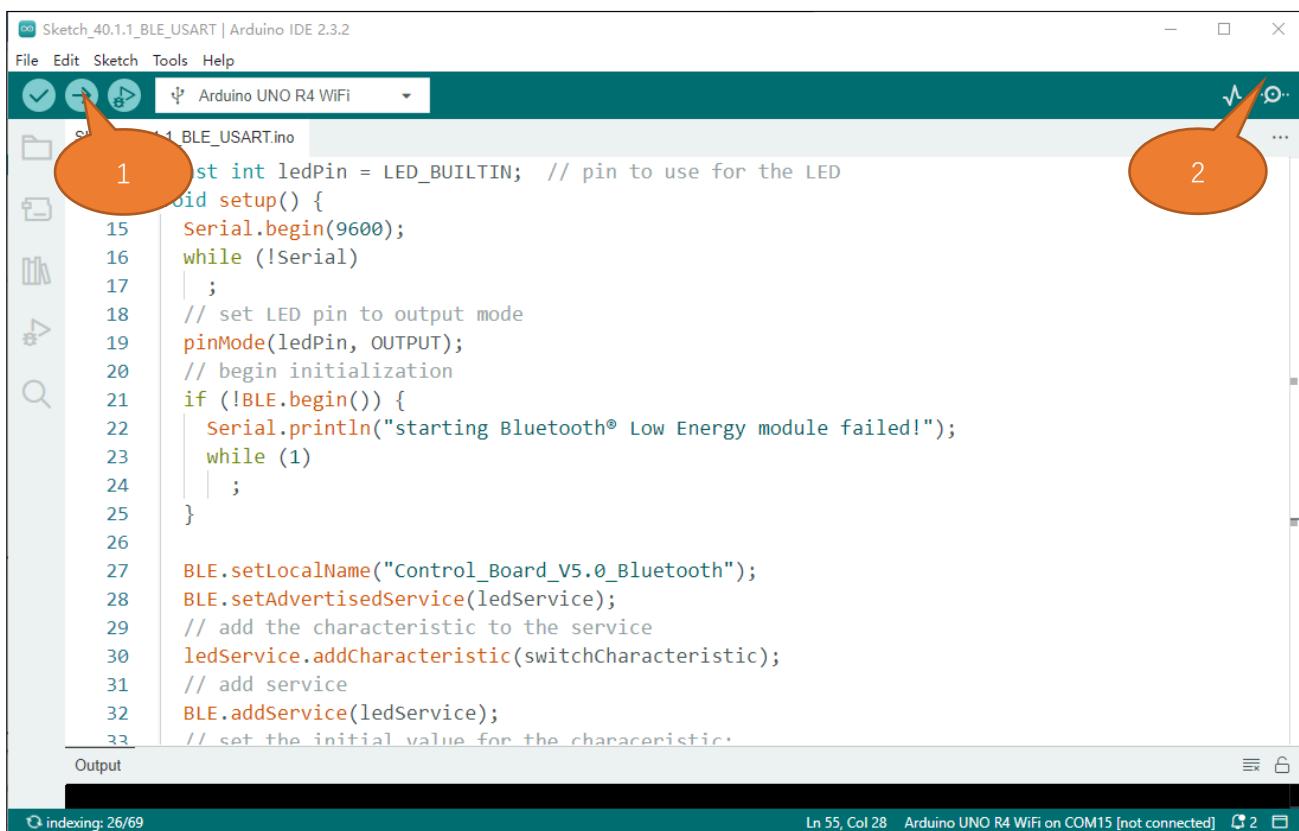
How to install the library

open Arduino IDE, click Sketch → Include Library → Add .ZIP Library, In the pop-up window, find the file named “./Libraries/ **ArduinoBLE1.3.7.zip**” which locates in this directory, and click OPEN.



Step1. Upload the code of Project 33.1 to control board.

Step2. Click on serial monitor.



```
Sketch_40.1.1_BLE_USART | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Arduino UNO R4 WiFi
Sketch_40.1.1_BLE_USART.ino
1 #include <BLE.h>
2
3 const int ledPin = LED_BUILTIN; // pin to use for the LED
4
5 void setup() {
6     Serial.begin(9600);
7     while (!Serial)
8     | ;
9     // set LED pin to output mode
10    pinMode(ledPin, OUTPUT);
11    // begin initialization
12    if (!BLE.begin()) {
13        Serial.println("starting Bluetooth® Low Energy module failed!");
14        while (1)
15        | ;
16    }
17
18    BLE.setLocalName("Control_Board_V5.0_Bluetooth");
19    BLE.setAdvertisedService(ledService);
20    // add the characteristic to the service
21    ledService.addCharacteristic(switchCharacteristic);
22    // add service
23    BLE.addService(ledService);
24    // set the initial value for the characteristic.
25
26
27
28
29
30
31
32
33
```

Output

indexing: 26/69 Ln 55, Col 28 Arduino UNO R4 WiFi on COM15 [not connected] 2

Step3. Set baud rate to 9600.



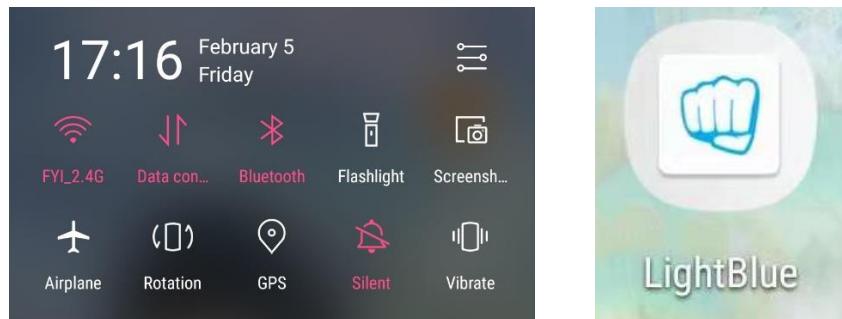
Output Serial Monitor x

Message (Enter to send message to 'Arduino UNO R4 WiFi' on 'COM15') New Line 9600 baud

15:47:23.293 ->Connected event, central: 76:b4:2c:f3:fad6
15:47:59.411 ->

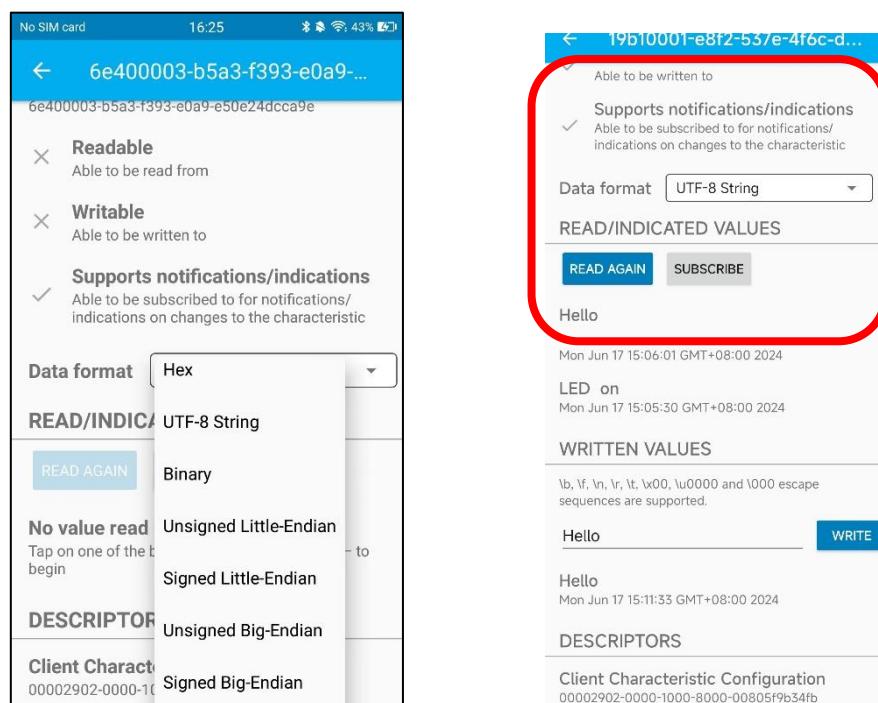
Ln 16, Col 6 Arduino UNO R4 WiFi on COM15 2

Turn ON Bluetooth on your phone, and open the Lightblue APP.



In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click Control_Board_V5.0_Bluetooth.

Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



Back to the serial monitor on your computer. You can type anything in the left border of Send, and then click Send.



The last received data will be printed when the READ AGAIN button on the app is tapped.

◀ 19b10001-e8f2-437e-4f6c-d...

✓ Able to be written to

✓ Supports notifications/indications

✓ Able to be subscribed to for notifications/indications on changes to the characteristic

Data format **UTF-8 String**

READ/INDICATED VALUES

READ AGAIN **SUBSCRIBE**

Hello

Mon Jun 17 15:06:01 GMT+08:00 2024

LED on

Mon Jun 17 15:05:30 GMT+08:00 2024

WRITTEN VALUES

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.

Hello **WRITE**

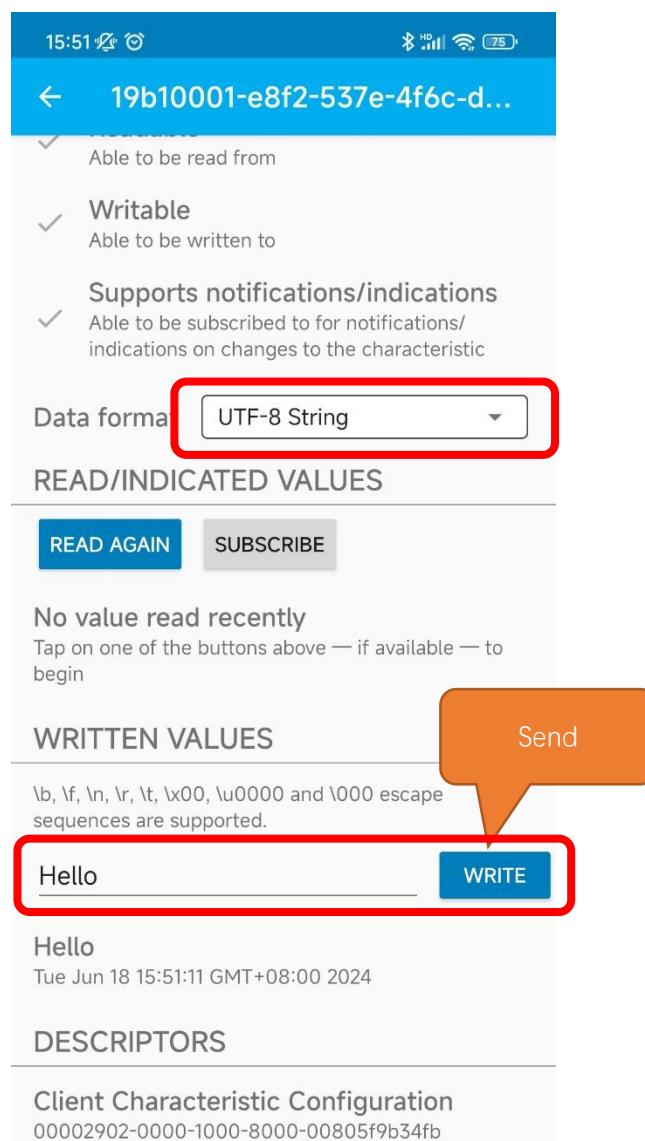
Hello

Mon Jun 17 15:11:33 GMT+08:00 2024

DESCRIPTORS

Client Characteristic Configuration
00002902-0000-1000-8000-00805f9b34fb

Similarly, you can select “Send” on your phone. Set Data format, and then enter anything in the sending box and click Write to send.



And the computer will receive the message from the mobile Bluetooth.



And now data can be transferred between your mobile phone and computer via control board.

The following is the program code:

```
1 #include <ArduinoBLE.h>
2 #include "String.h"
3
4 BLEService ledService("19B10000-E8F2-537E-4F6C-D104768A1214"); // Bluetooth® Low Energy LED
5 Service
6 // Bluetooth® Low Energy LED Switch Characteristic - custom 128-bit UUID, read and writable by
7 central
8 BLECharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | 
9 BLEWrite | BLENotify | BLEBroadcast, 20);
10
11 String inputString = ""; // a String to hold incoming data
12 bool stringComplete = false; // whether the string is complete
13 long lasttime = 0;
14
15 const int ledPin = LED_BUILTIN; // pin to use for the LED
16 void setup() {
17   Serial.begin(9600);
18   while (!Serial)
19     ;
20   // set LED pin to output mode
21   pinMode(ledPin, OUTPUT);
22   // begin initialization
23   if (!BLE.begin()) {
24     Serial.println("starting Bluetooth® Low Energy module failed!");
25     while (1)
26       ;
27   }
28
29   BLE.setLocalName("Control_Board_V5.0_Bluetooth");
30   BLE.setAdvertisedService(ledService);
31   // add the characteristic to the service
32   ledService.addCharacteristic(switchCharacteristic);
33   // add service
34   BLE.addService(ledService);
35   // set the initial value for the characteristic:
36   switchCharacteristic.writeValue("LED");
37
38   BLE.setEventHandler(BLEConnected, blePeripheralConnectHandler);
39   BLE.setEventHandler(BLEDisconnected, blePeripheralDisconnectHandler);
40
41   switchCharacteristic.setEventHandler(BLEWritten, switchCharacteristicWritten);
```

```
41 // start advertising
42 BLE.advertise();
43
44 Serial.println("Bluetooth® device active, waiting for connections..");
45 }
46
47 void loop() {
48 long nowtime = millis();
49 if (nowtime - lasttime > 1000) {
50 BLE.poll();
51 Serial.print(".");
52 if (Serial.available() > 0) {
53 String str = Serial.readString();
54 const char *newValue = str.c_str();
55 switchCharacteristic.writeValue(newValue);
56 }
57 lasttime = nowtime;
58 }
59 }
60
61 void blePeripheralConnectHandler(BLEDevice central) {
62 Serial.print("Connected event, central: ");
63 Serial.println(central.address());
64 }
65
66 void blePeripheralDisconnectHandler(BLEDevice central) {
67 Serial.print("Disconnected event, central: ");
68 Serial.println(central.address());
69 }
70
71 void switchCharacteristicWritten(BLEDevice central, BLECharacteristic characteristic) {
72 Serial.print("Characteristic event, written: ");
73 uint8_t characteristicValue[20];
74 int bytesRead = characteristic.readValue(characteristicValue, sizeof(characteristicValue));
75 Serial.print("Received bytes: ");
76 for (int i = 0; i < bytesRead; i++) {
77 Serial.print(characteristicValue[i], HEX);
78 Serial.print(" ");
79 }
80 Serial.println();
81 String receivedString = "";
82 for (int i = 0; i < bytesRead; i++) {
83 receivedString += (char)characteristicValue[i];
84 }
```

```

85   Serial.println("Value: " + receivedString);
86 }
```

Initialize the BLE function and name it.

```
27   BLE.setLocalName("Control_Board_V5.0_Bluetooth");
```

Write a Callback function for BLE server to manage connection of BLE.

```

61   void blePeripheralConnectHandler(BLEDevice central) {
62     Serial.print("Connected event, central: ");
63     Serial.println(central.address());
64   }
65
66   void blePeripheralDisconnectHandler(BLEDevice central) {
67     Serial.print("Disconnected event, central: ");
68     Serial.println(central.address());
69 }
```

When the mobile phone send data to control board via BLE Bluetooth, it will print them out with serial port;

When the serial port of control board receive data, it will send them to mobile via BLE Bluetooth.

```

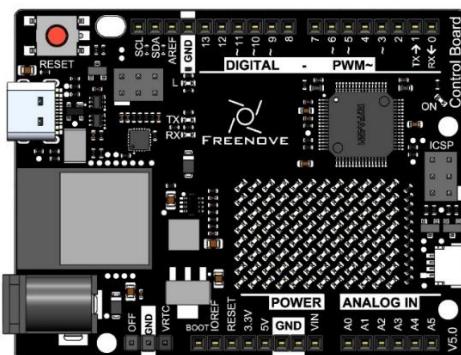
52   if (Serial.available() > 0) {
53     String str = Serial.readString();
54     const char *newValue = str.c_str();
55     switchCharacteristic.writeValue(newValue);
56   }
...
71   void switchCharacteristicWritten(BLEDevice central, BLECharacteristic characteristic) {
72     Serial.print("Characteristic event, written: ");
73     uint8_t characteristicValue[20];
74     int bytesRead = characteristic.readValue(characteristicValue, sizeof(characteristicValue));
75     Serial.print("Received bytes: ");
76     for (int i = 0; i < bytesRead; i++) {
77       Serial.print(characteristicValue[i], HEX);
78       Serial.print(" ");
79     }
80     Serial.println();
81     String receivedString = "";
82     for (int i = 0; i < bytesRead; i++) {
83       receivedString += (char)characteristicValue[i];
84     }
85     Serial.println("Value: " + receivedString);
86 }
```

Project 34.2 Control LED with Bluetooth

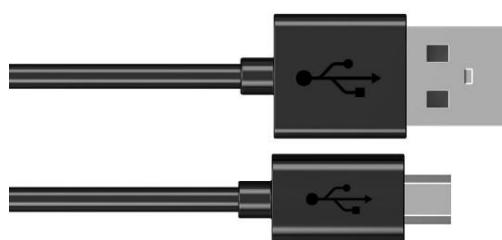
In this section, we will further learn how to use Bluetooth to control an LED.

Component List

Control board x1



Micro USB Wire x1



LED x1



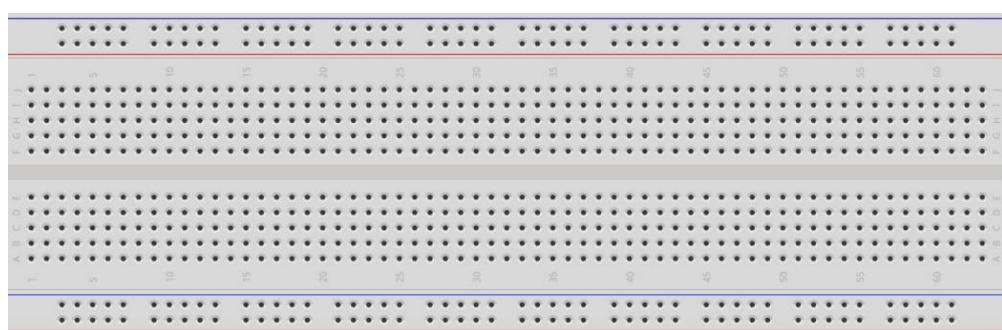
Resistor 220Ω x1



Jumper M/M x2



Breadboard x1

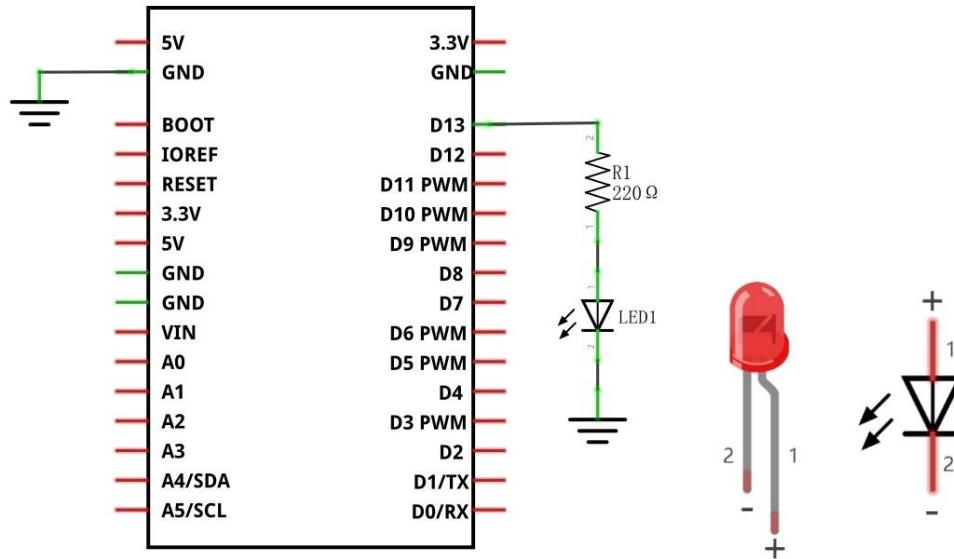


If you do not have the above components in your kit, you can use the control board alone to finish the project. The control pin of the LED in this project is the same as the one controlling the onboard LED.

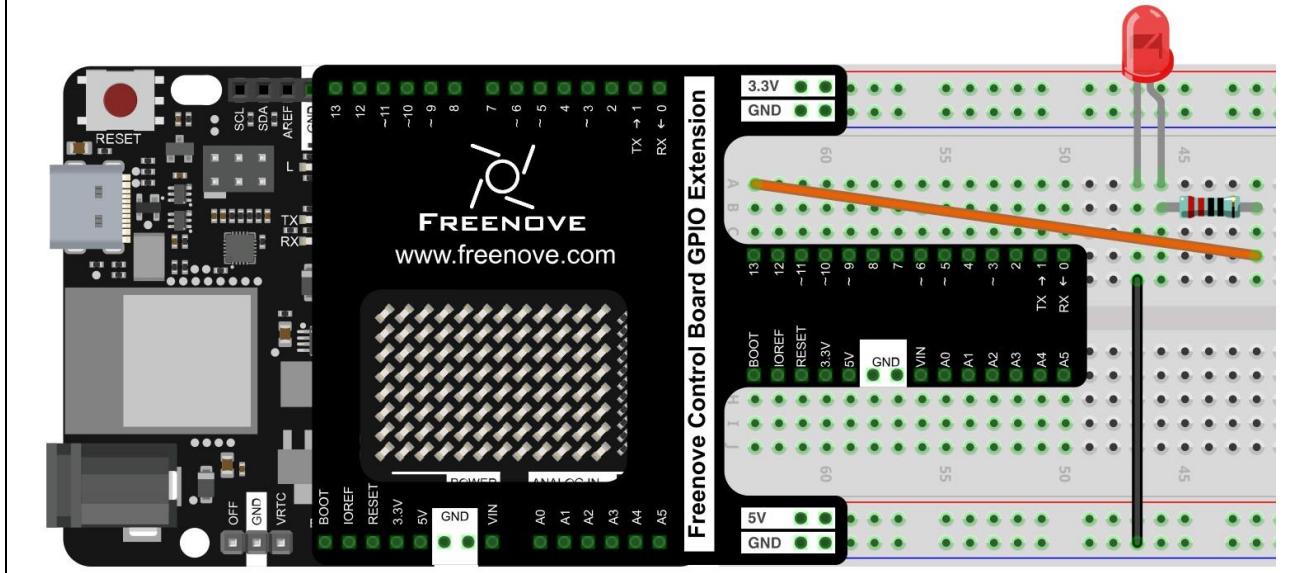
Circuit

Connect the control board to your computer with the USB cable.

Schematic diagram



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



Sketch

Sketch_34.2.1

The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The title bar indicates the sketch is named "Sketch_40.2.1_Control_LED_with_Bluetooth" and is using "Arduino IDE 2.3.2". The central workspace displays the code for "Sketch_40.2.1_Control_LED_with_Bluetooth.ino". The code uses the BLE library to handle Bluetooth connections and control LEDs. The Serial Monitor window at the bottom shows the following output:

```
[=====] 95% (21/22 pages) write(addr=0x34, size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x15000, size=0x1000)

[=====] 100% (22/22 pages)
Done in 5.499 seconds
reset()
```

The status bar at the bottom right shows "Ln 60, Col 1" and "Arduino Uno R4 WiFi on COM15".

Compile and upload code to Control board. The operation of the APP is the same as 40.1, you only need to change the sending content to "led_on" and "led_off" to operate LEDs on the Control board.

Data sent from mobile APP:

← 19b10001-e8f2-537e-4f6c-d...

Data format UTF-8 String ▾

READ/INDICATED VALUES

READ AGAIN SUBSCRIBE

No value read recently
Tap on one of the buttons above — if available — to begin

WRITTEN VALUES

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.

led_off WRITE

led_off
Tue Jun 18 16:21:54 GMT+08:00 2024

led_on
Tue Jun 18 16:21:48 GMT+08:00 2024

led_off
Tue Jun 18 16:21:44 GMT+08:00 2024

led_on
Tue Jun 18 16:21:39 GMT+08:00 2024

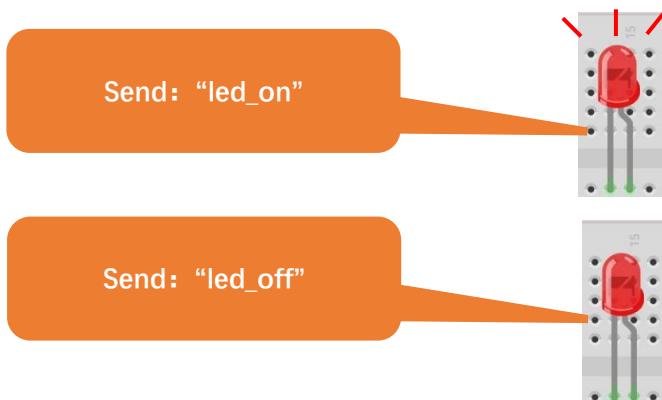
DESCRIPTORS

Client Characteristic Configuration
00002902-0000-1000-8000-00805f9b34fb

Display on the serial port of the computer:

```
Message (Ctrl + Enter to send message to 'ESP32S3 Dev Module' on 'COM3')  
New Line 115200 baud  
The device started, now you can pair it with Bluetooth!  
led_on  
led_off  
led_on  
led_off  
led_on  
led_off  
Ln 17, Col 17  UTF-8  ESP32S3 Dev Module on COM3  2  
```

The phenomenon of LED



Attention: If the sending content isn't "led-on" or "led-off", then the state of LED will not change. If the LED is on, when receiving irrelevant content, it keeps on; Correspondingly, if the LED is off, when receiving irrelevant content, it keeps off.

If you do not connect external LED circuit, you can check the status of the onboard LED.

With this example you can design more interesting projects.

The following is the program code:

```
1 #include <ArduinoBLE.h>
2 #include "String.h"
3
4 BLEService ledService("19B10000-E8F2-537E-4F6C-D104768A1214"); // Bluetooth® Low Energy LED
5 Service
6 // Bluetooth® Low Energy LED Switch Characteristic - custom 128-bit UUID, read and writable by
7 central
8 BLECharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | 
9 BLEWrite | BLENotify | BLEBroadcast, 20);
10
11 const int ledPin = LED_BUILTIN; // pin to use for the LED
12 void setup() {
13     Serial.begin(9600);
14     while (!Serial)
15         ;
16     // set LED pin to output mode
17     pinMode(ledPin, OUTPUT);
18     // begin initialization
19     if (!BLE.begin()) {
20         Serial.println("starting Bluetooth® Low Energy module failed!");
21         while (1)
22             ;
23     }
24
25     BLE.setLocalName("Control_Board_V5.0_Bluetooth");
26     BLE.setAdvertisedService(ledService);
27     // add the characteristic to the service
28     ledService.addCharacteristic(switchCharacteristic);
29     // add service
30     BLE.addService(ledService);
31     // set the initial value for the characteristic:
32     switchCharacteristic.writeValue("Led");
33
34     BLE.setEventHandler(BLEConnected, blePeripheralConnectHandler);
35     BLE.setEventHandler(BLEDDisconnected, blePeripheralDisconnectHandler);
36
37     switchCharacteristic.setEventHandler(BLEWritten, switchCharacteristicWritten);
38
39     // start advertising
40     BLE.advertise();
41
42     Serial.println("Bluetooth® device active, waiting for connections...");
43 }
```

```
44
45 void loop() {
46     BLEDevice central = BLE.central();
47     if (central) {
48         central.poll();
49         delay(1000);
50         Serial.print(".");
51     }
52 }
53
54 void blePeripheralConnectHandler(BLEDevice central) {
55     Serial.print("Connected event, central: ");
56     Serial.println(central.address());
57 }
58
59 void blePeripheralDisconnectHandler(BLEDevice central) {
60     Serial.print("Disconnected event, central: ");
61     Serial.println(central.address());
62 }
63
64 void switchCharacteristicWritten(BLEDevice central, BLECharacteristic characteristic) {
65     Serial.print("Characteristic event, written: ");
66
67     uint8_t characteristicValue[20];
68     int bytesRead = characteristic.readValue(characteristicValue, sizeof(characteristicValue));
69
70     Serial.print("Received bytes: ");
71     for (int i = 0; i < bytesRead; i++) {
72         Serial.print(characteristicValue[i], HEX);
73         Serial.print(" ");
74     }
75     Serial.println();
76
77     String receivedString = "";
78
79     for (int i = 0; i < bytesRead; i++) {
80         receivedString += (char)characteristicValue[i];
81     }
82     Serial.println("Value: " + receivedString);
83     if (receivedString == "led_on") {
84         Serial.println("LED on");
85         digitalWrite(ledPin, HIGH); // will turn the LED on
86     }
87     if (receivedString == "led_off") { // a 0 value
```

```
88     Serial.println(F("LED off"));
89     digitalWrite(ledPin, LOW); // will turn the LED off
90 }
91 }
```

Use character string to handle function header file.

```
2 #include "String.h"
```

Initialize the BLE Bluetooth and name it as "Control_Board_V5.0_Bluetooth"

```
25 BLE.setLocalName("Control_Board_V5.0_Bluetooth");
```

Compare the content in buffer array with "led_on" and "led_off" to see whether they are the same. If yes, execute the corresponding operation.

```
83 if (receivedString == "led_on") {
84     Serial.println("LED on");
85     digitalWrite(ledPin, HIGH); // will turn the LED on
86 }
87 if (receivedString == "led_off") { // a 0 value
88     Serial.println(F("LED off"));
89     digitalWrite(ledPin, LOW); // will turn the LED off
90 }
```

Chapter 35 USB HID

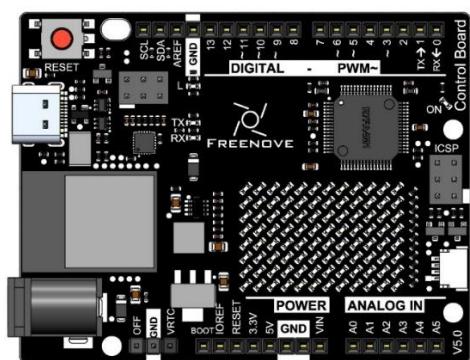
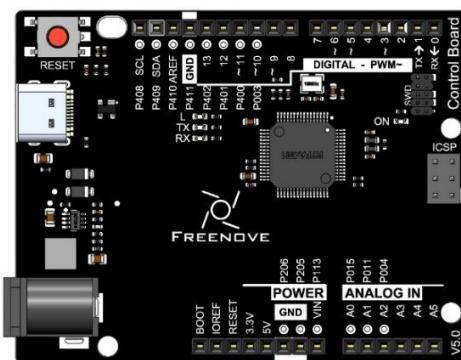
In this chapter, we will learn how to use a control board with keyboard and mouse APIs to emulate mouse and keyboard actions. This feature can be used to create game controllers, keyboard extensions, or other Human Interface Devices (HID).

Project 35.1 Mouse control

In this project, we learn how to emulate a mouse control.

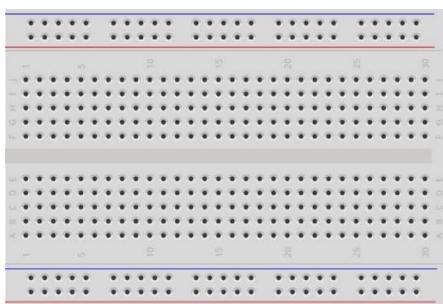
Component List

Control board x1

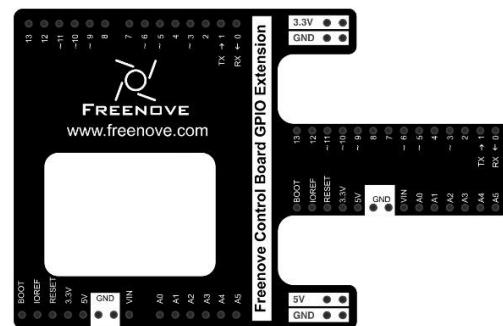


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Resistor 10kΩ x8



Push button Switch x4



Jumper M/M x12



Component knowledge

Human Interface Device (HID)

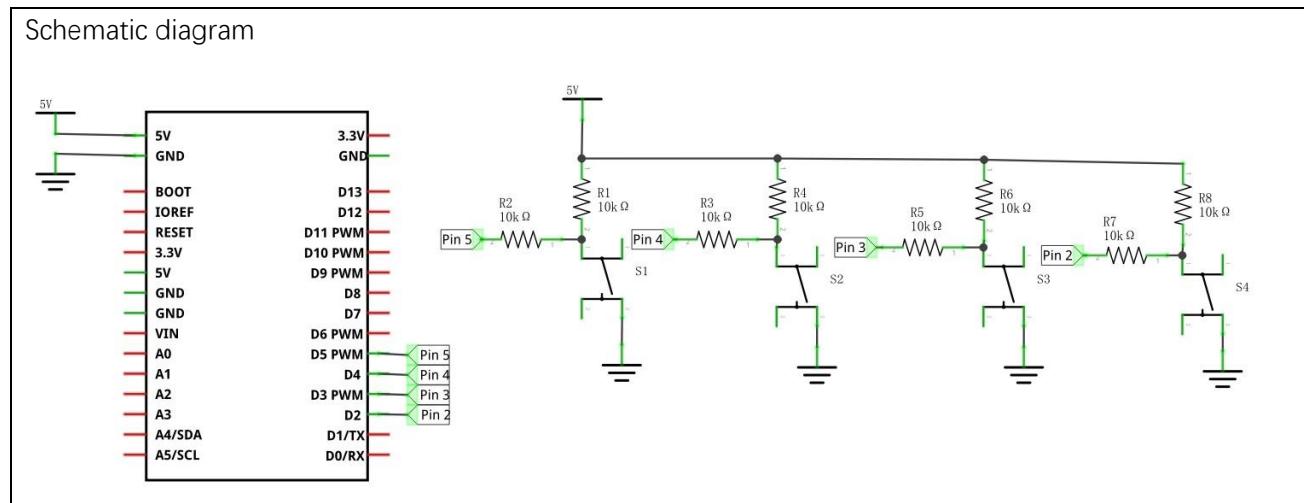
Human Interface Devices (HID) are devices designed for human use (such as keyboards, mice, game controllers, etc.), which often send data to the computer via USB. When you press a key on the keyboard, you send data to the computer, the computer reads the data and then activates the corresponding key.

HID, or Human-Interface Devices, is a category of computer devices designed for direct interaction with humans and are typically used for input purposes. This category includes devices such as keyboards, mice, and game controllers. With the control board (V5), you can simulate these devices, unlocking a multitude of possibilities for DIY projects.

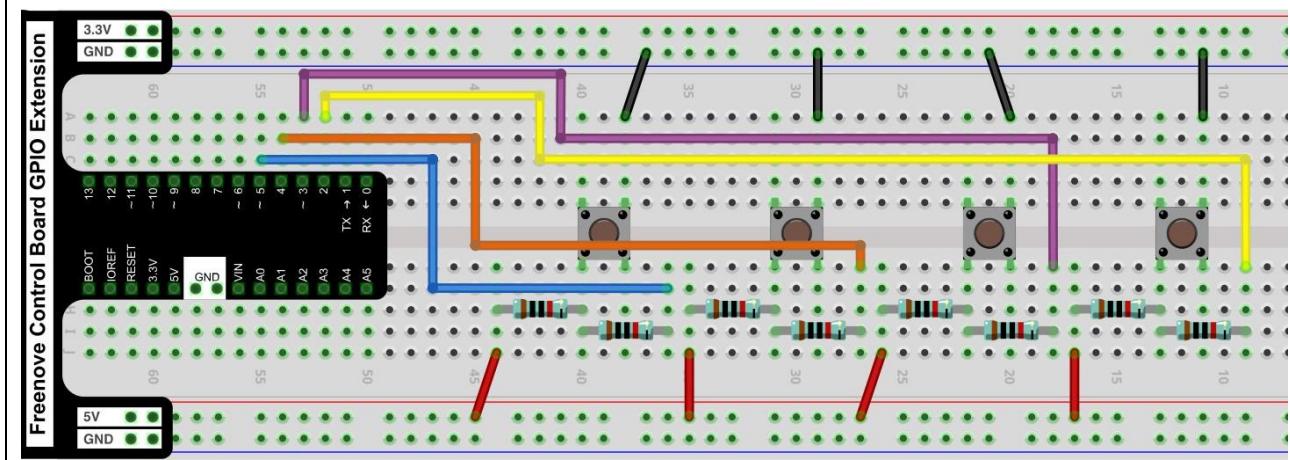
The control board (V5) has built-in support for HID, a feature found on most modern development boards but not on previous UNO versions.

The control board (V5) is more than just a powerful development board; it also has built-in support for Human-Interface Devices (HID). This allows you to use the board to simulate devices such as mice and keyboards, adding a new level of interactivity to your projects.

Circuit



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



Sketch

Sketch_35.1.1

Upload the code to the control board, and you can easily control the movement of the mouse with the buttons. Here, four buttons are set up to control the mouse for moving up, down, left, and right.

The following is the program code:

```

1 #include <Mouse.h> // Include the Mouse library for mouse control
2
3 const int upButton = 2; // Define the pin where the button is connected
4 const int downButton = 3; // Define the pin where the button is connected
5 const int leftButton = 4; // Define the pin where the button is connected
6 const int rightButton = 5; // Define the pin where the button is connected
7
8 void setup() {
9     pinMode(upButton, INPUT_PULLUP); // Set the button pin as an input
10    pinMode(downButton, INPUT_PULLUP); // Set the button pin as an input
11    pinMode(leftButton, INPUT_PULLUP); // Set the button pin as an input
12    pinMode(rightButton, INPUT_PULLUP); // Set the button pin as an input
13    Mouse.begin(); // Initialize mouse control
14    delay(1000); // Wait for 1 second (1000 milliseconds) for hardware
15    initialization
16 }
17
18 void loop() {
19     // Check if the button is pressed (HIGH)
20     if (digitalRead(rightButton) == LOW) {
21         //right

```

```

22     Mouse.move(10, 0); // Move the mouse 10 units to the right
23     delay(200);      // Wait for 200 milliseconds to slow down mouse movement
24 }
25 if (digitalRead(leftButton) == LOW) {
26     //Left
27     Mouse.move(-10, 0);
28     delay(200);
29 }
30 if (digitalRead(downButton) == LOW) {
31     // Down
32     Mouse.move(0, 10);
33     delay(200);
34 }
35 if (digitalRead(upButton) == LOW) {
36     // Up
37     Mouse.move(0, -10);
38     delay(200);
39 }
40 }
```

Include the mouse library for mouse control.

1	#include <Mouse.h> // Include the Mouse library for mouse control
---	---

Determine whether a button is press, and execute corresponding mouse movement.

```

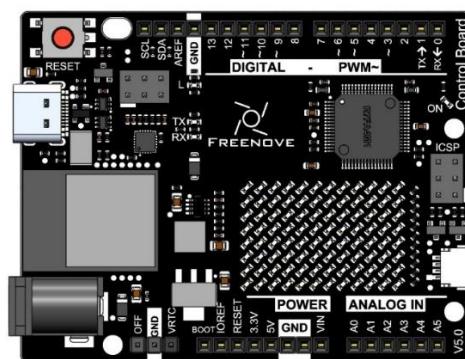
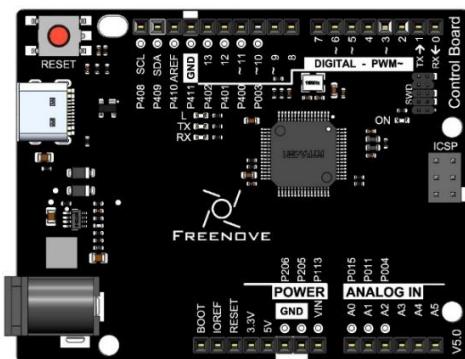
19 // Check if the button is pressed (HIGH)
20 if (digitalRead(rightButton) == LOW) {
21     //right
22     Mouse.move(10, 0); // Move the mouse 10 units to the right
23     delay(200);      // Wait for 200 milliseconds to slow down mouse movement
24 }
25 if (digitalRead(leftButton) == LOW) {
26     //Left
27     Mouse.move(-10, 0);
28     delay(200);
29 }
30 if (digitalRead(downButton) == LOW) {
31     // Down
32     Mouse.move(0, 10);
33     delay(200);
34 }
35 if (digitalRead(upButton) == LOW) {
36     // Up
37     Mouse.move(0, -10);
38     delay(200);
39 }
```

Project 35.2 Keypad Control

In this project, we will control keyboard input through buttons.

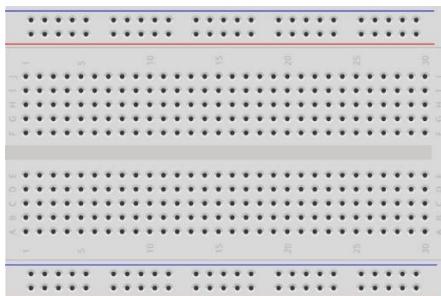
Component List

Control board x1

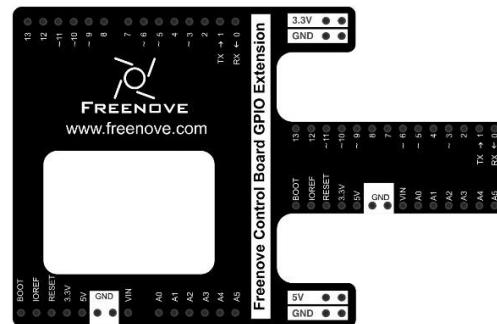


or

Breadboard x1



GPIO Extension Board x1



USB cable x1



Resistor 10kΩ x2



Push button Switch x4

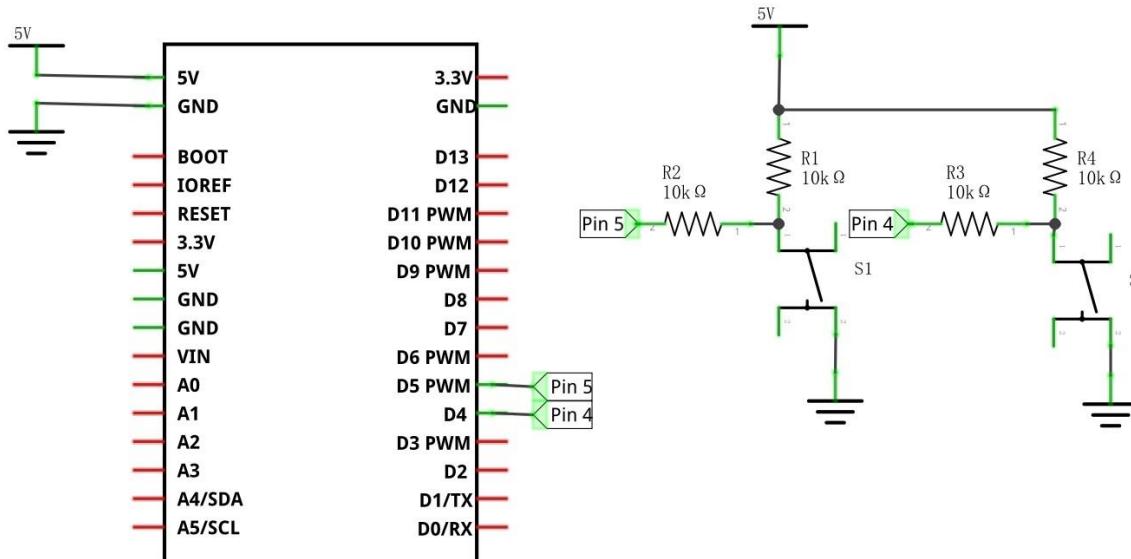


Jumper M/M x6

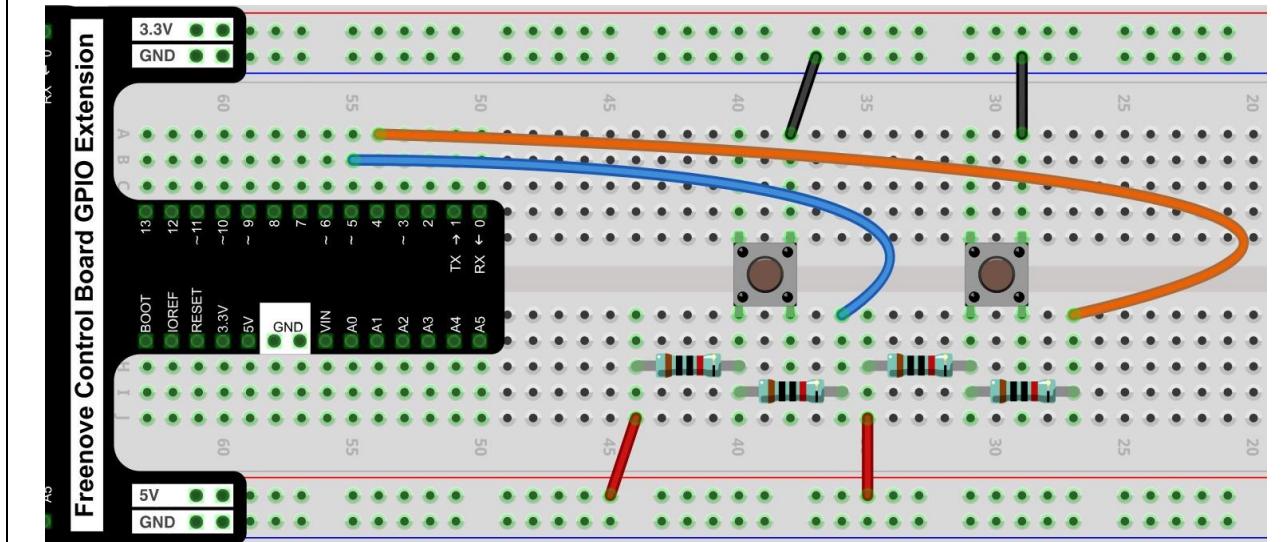


Circuit

Schematic diagram



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



Sketch

Sketch_35.2.1

Upload the code to the control board, and you can easily control keyboard input through buttons. Here, two buttons are set up: one for copying (simultaneously pressing the CTRL and C keys on the keyboard), and the other for pasting (simultaneously pressing the CTRL and V keys on the keyboard). We can use these two buttons to complete the copy and paste of text. For example, to copy the text "Freenove.com" from the file "test.txt," use the mouse to select the text, press the copy button, and then press the paste button to perform the text copy.

Need help? Contact support@freenove.com

freenove.com

freenove.com freenove.com

Select the texts to copy

Press the copy and paste buttons in turn

The following is the program code:

```

1 #include <Keyboard.h> // Include the Keyboard library to enable keyboard functionalities
2
3 const int copyButtonPin = 4; // Pin number for the copy button
4 const int pasteButtonPin = 5; // Pin number for the paste button
5
6 void setup() {
7     Keyboard.begin(); // Initialize the Keyboard library
8     pinMode(copyButtonPin, INPUT_PULLUP); // Set the copy button pin as input
9     pinMode(pasteButtonPin, INPUT_PULLUP); // Set the paste button pin as input
10    delay(1000); // Wait for 1 second to allow hardware initialization
11 }
12
13 void loop() {
14     // Check if the copy button is pressed
15     if (digitalRead(copyButtonPin) == LOW) {
16         Keyboard.press(KEY_LEFT_CTRL); // Press the Ctrl key
17         Keyboard.press('c'); // Press the 'c' key
18         Keyboard.releaseAll(); // Release all keys
19         delay(150); // Wait for 200 milliseconds
20     }
21     // Check if the paste button is pressed
22     else if (digitalRead(pasteButtonPin) == LOW) {
23         Keyboard.press(KEY_LEFT_CTRL); // Press the Ctrl key
24         Keyboard.press('v'); // Press the 'v' key
25         Keyboard.releaseAll(); // Release all keys
26         delay(150); // Wait for 200 milliseconds
27     }
28 }
```

Include the Keyboard library to enable keyboard functionalities

```
1 #include <Keyboard.h> // Include the Keyboard library to enable keyboard functionalities
```

Determine whether a button is pressed, and execute corresponding key input values.

```

15     if (digitalRead(copyButtonPin) == LOW) {
16         Keyboard.press(KEY_LEFT_CTRL); // Press the Ctrl key
17         Keyboard.press('c'); // Press the 'c' key
18         Keyboard.releaseAll(); // Release all keys
19         delay(150); // Wait for 200 milliseconds
```

```
20 }  
21 // Check if the paste button is pressed  
22 else if (digitalRead(pasteButtonPin) == LOW) {  
23     Keyboard.press(KEY_LEFT_CTRL); // Press the Ctrl key  
24     Keyboard.press('v'); // Press the 'v' key  
25     Keyboard.releaseAll(); // Release all keys  
26     delay(150); // Wait for 200 milliseconds  
27 }
```

For more examples of USB HID, please refer to:

<https://docs.arduino.cc/tutorials/uno-r4-minima/usb-hid/>

Chapter 36 Soldering Circuit Board

From previous chapters, we have learned about electronic circuits and components and have built a variety of circuits using a Breadboard device, which is not designed to be used permanently. We now will take a further step to make permanent projects using a Perfboard (a type of Prototype Circuit Board). Note: Perfboard is a stiff, thin sheet of insulated material with holes bored on a grid. The grid is usually a squared off shape with a spacing of 0.1 inches. Square copper pads cover these holes to make soldering electronic components easier.

To finish this chapter, you need to prepare the necessary soldering equipment, including an electric soldering iron (or soldering pencil) and solder. We have already prepared the Perfboard for you.

CAUTION: Please use extreme caution and attention to safety when you operate soldering tools used in these projects.

Project 36.1 Solder a Buzzer

You should be familiar with the Buzzer from our previous project. We will solder a permanent circuit that when a Push Button Switch is pressed a Buzzer sounds

Note: This circuit **does not** require programming and will work when it is powered ON. When the button is not pressed and the Buzzer is not in use, there is no power consumption.

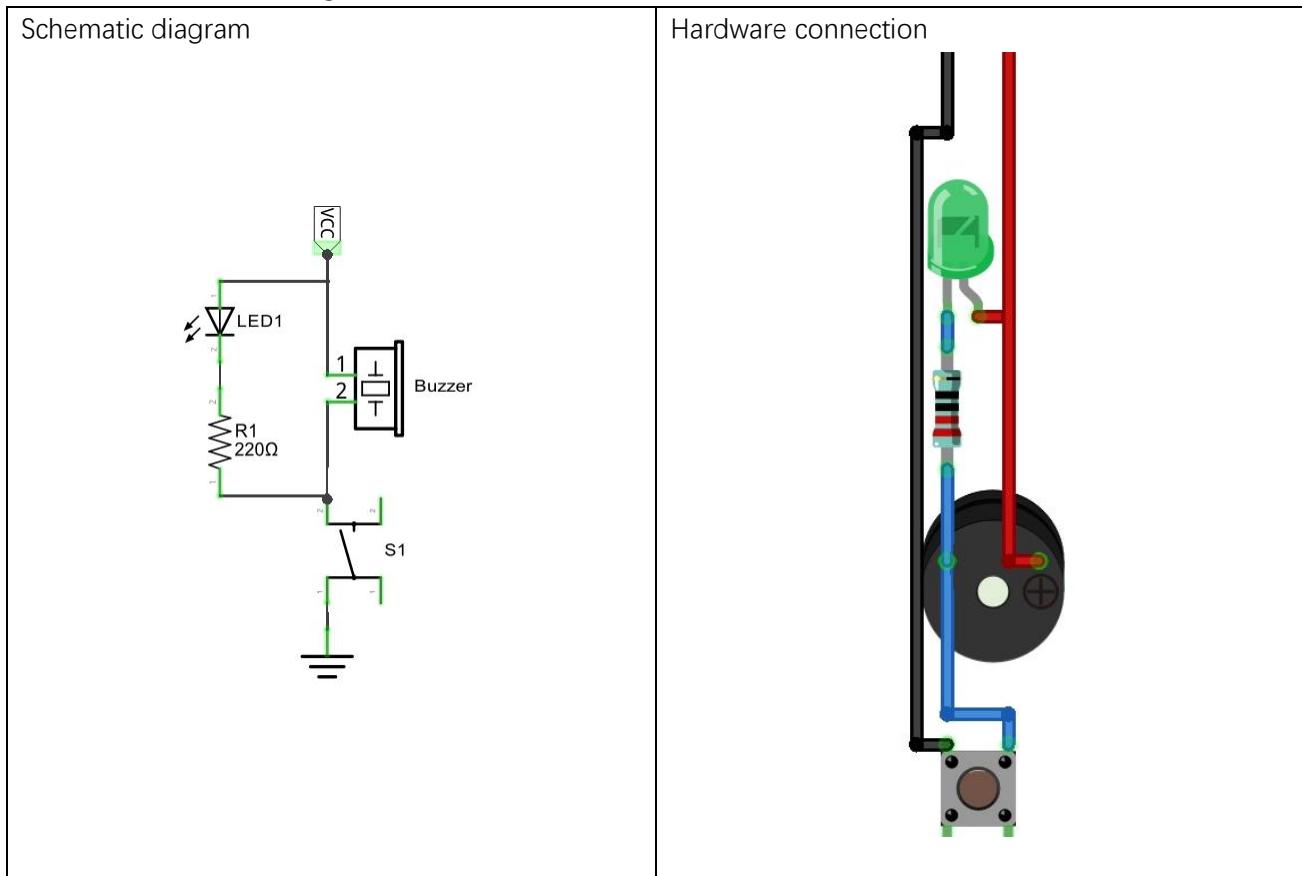
You can install it on your bicycle, your bedroom door or any other place where you want a Buzzer.

Component List

Pin header x2	LED x1	Resistor 220Ω x1	Active buzzer x1	Push button x1
				

Circuit

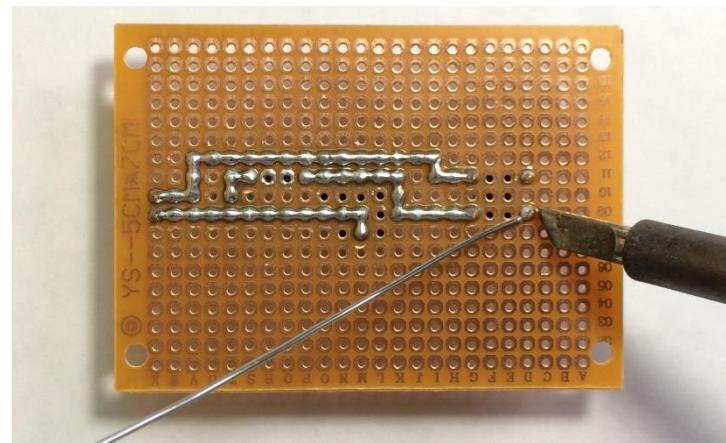
We will solder the following circuit on the Perfboard.



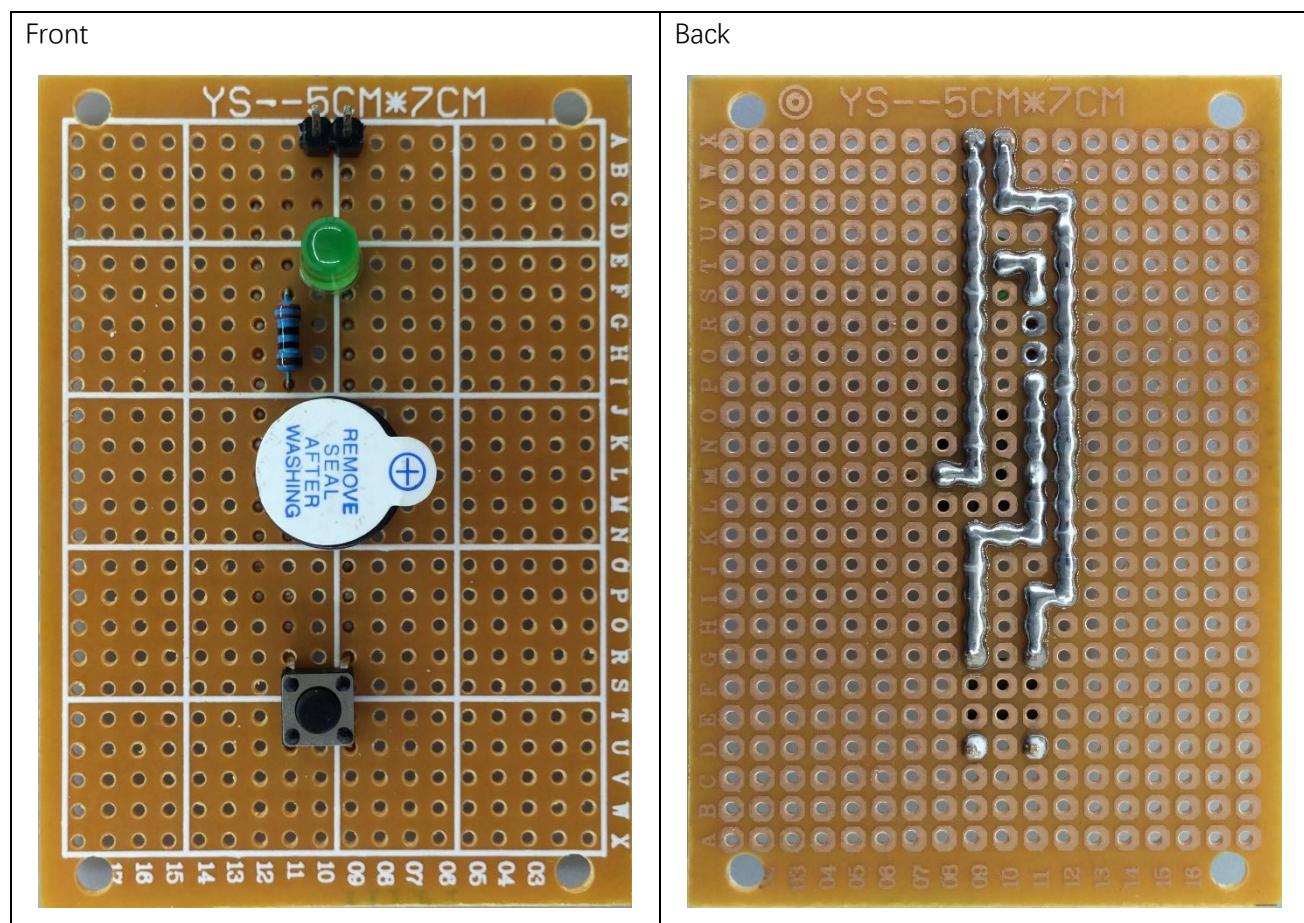
Note: If you are new to soldering electronic components on any type of circuit board we strongly recommend that you watch some instructional How-To videos by doing an Internet search and practice your soldering technique before attempting to solder the following projects. Some components can be damaged by exposure to excessive heat for prolonged times and there are various techniques you can learn that will help with making neater solder joints.

Solder the Circuit

Insert the components in the Perfboard following the Hardware Connection image as a general visual guide. Insert the pins of the components (all from the same side) so that you have only the components on one side of the Perfboard and the pins on the other. Then from the side with the pins carefully solder the circuit on the backside without having excess solder shorting out any portions of the circuit.

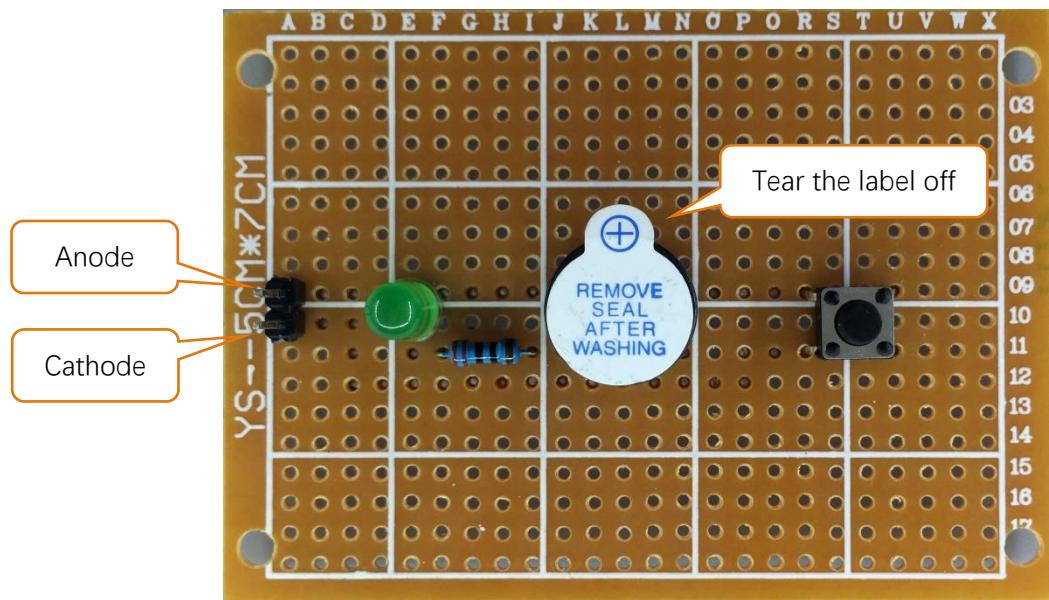


Here is a diagram after soldering from both sides of the Perfboard:



Test Circuit

Connect the circuit board to power supply (3~5V). You can use control board or battery box as the power supply.



Press the push button switch after connecting the power, and then the buzzer will sound.

Project 36.2 Solder a Flowing Water Light

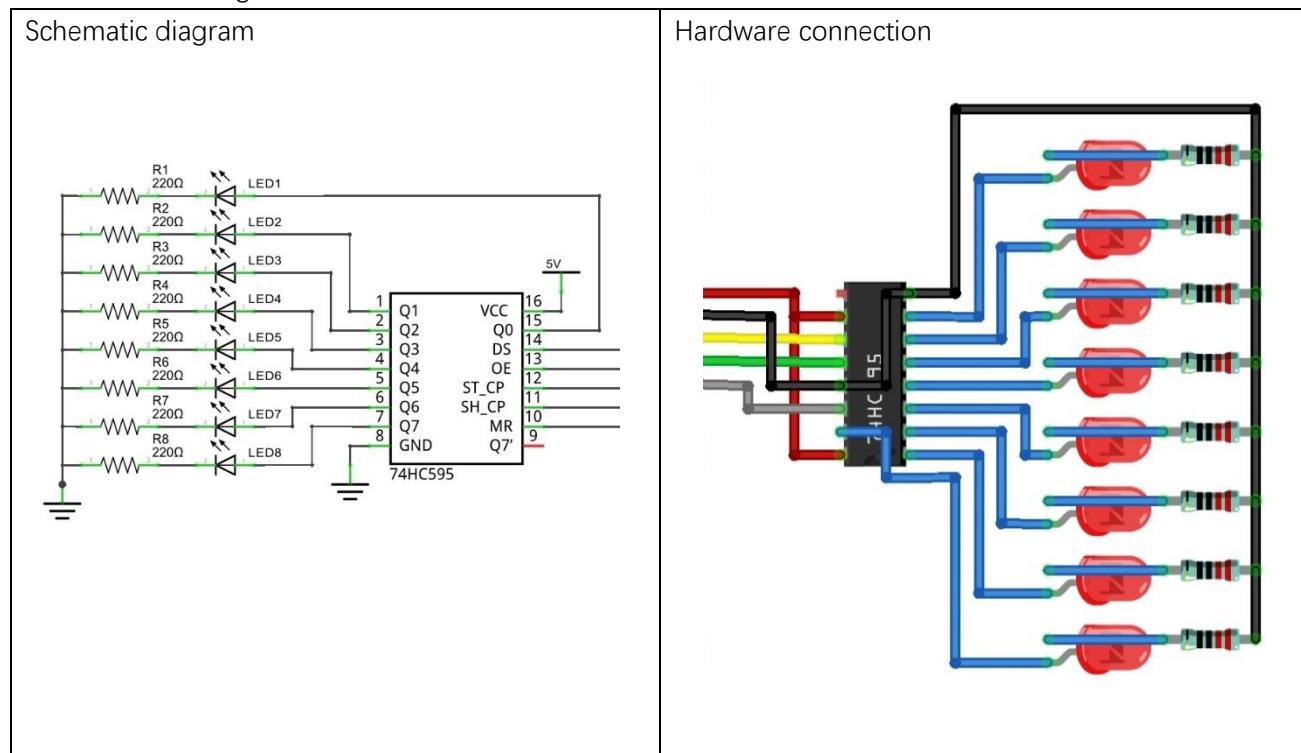
You should be familiar with the Flowing Water Light from our previous project. We will solder a permanent circuit using improved code to make a more interesting Flowing Water Light.

Component List

Pin header x5	Resistor 220Ω x8	LED x8	74HC595 x1

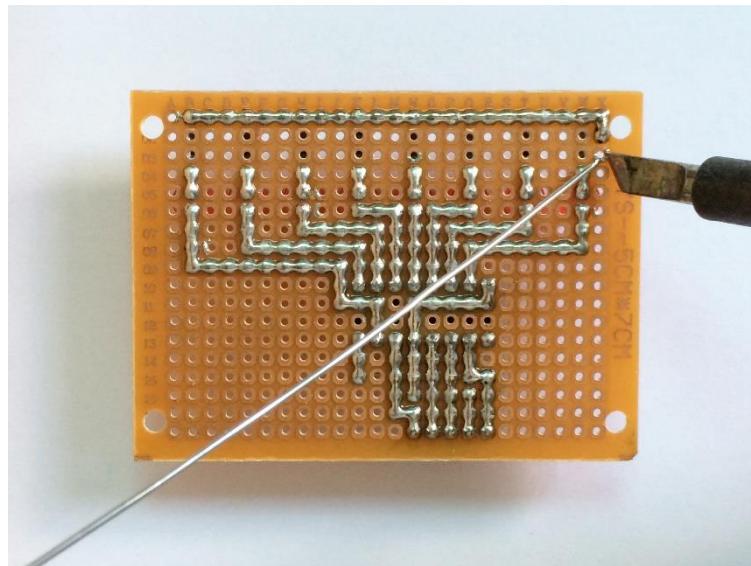
Circuit

Solder the following circuit on the Perfboard.

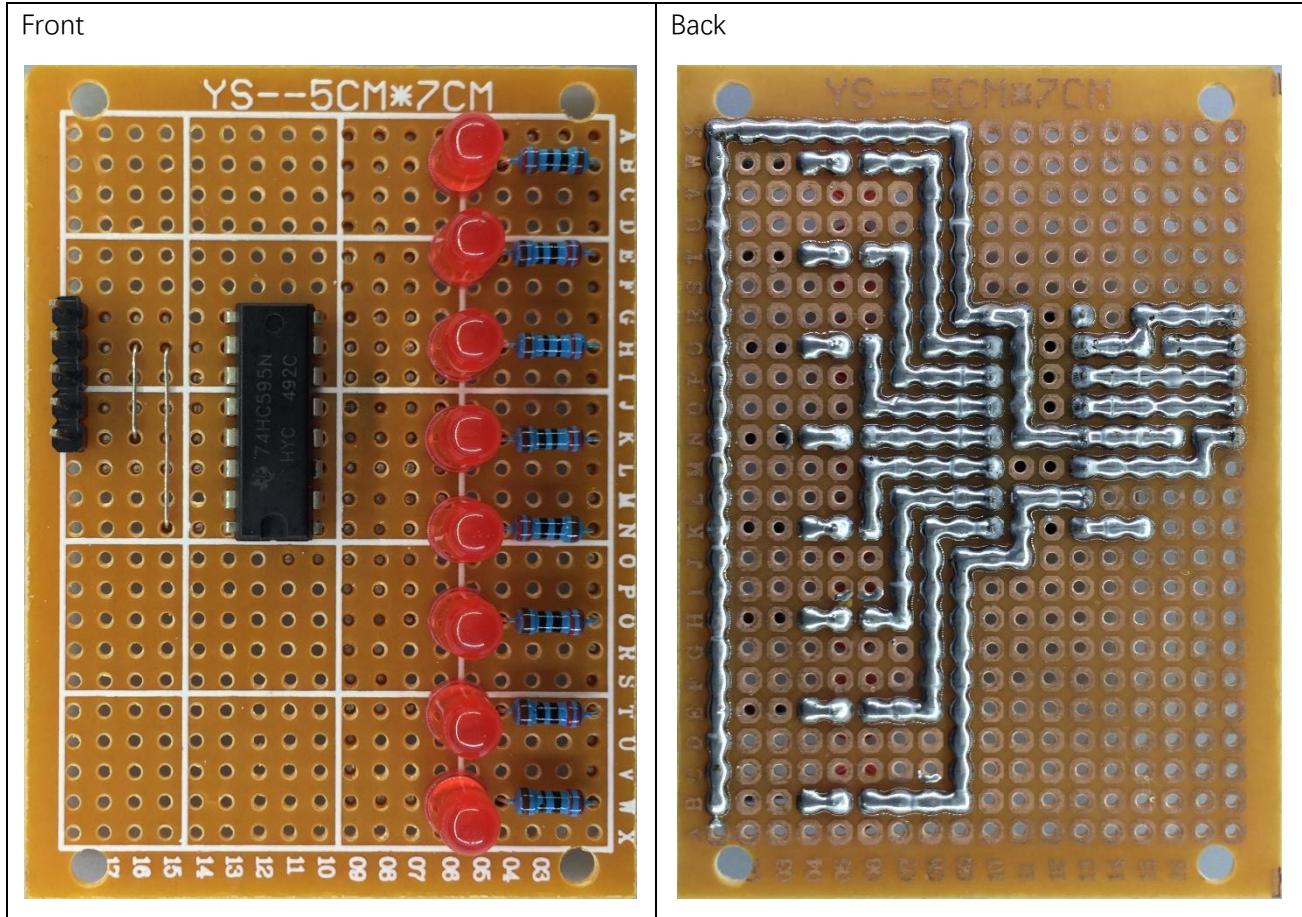


Solder the Circuit

Insert the components in the Perfboard, and solder the circuit on the back per earlier instructions.

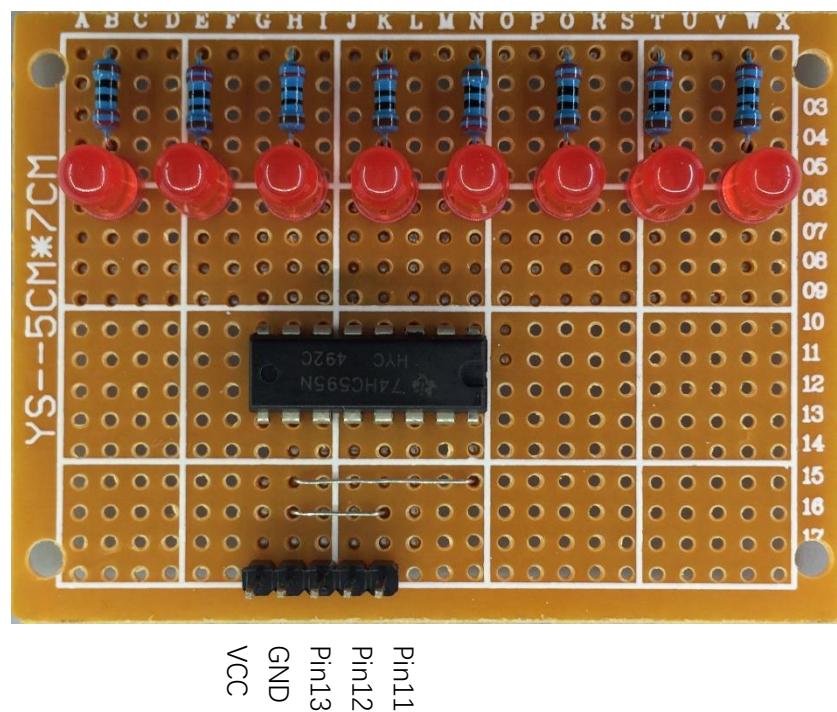


Here is a diagram after soldering from both sides of the Perfboard:



Connect the Circuit

Connect the board to control board with jumper wire in the following way.



Sketch

Sketch 36.2.1

Now, let's write code to make a dropping-rain effect on our board.

```

1 int latchPin = 12;           // Pin connected to ST_CP of 74HC595 (Pin12)
2 int clockPin = 13;           // Pin connected to SH_CP of 74HC595 (Pin11)
3 int dataPin = 11;            // Pin connected to DS of 74HC595 (Pin14)
4
5 void setup() {
6     // set pins to output
7     pinMode(latchPin, OUTPUT);
8     pinMode(clockPin, OUTPUT);
9     pinMode(dataPin, OUTPUT);
10 }
11
12 void loop() {
13     // Define an array to save the pulse width of LED. Output the signal to the 8 adjacent
14     // LEDs in order, and then it produces the dropping-rain effect

```

```

14   const byte pulse[] = {0, 0, 0, 0, 0, 0, 0, 0, 64, 48, 32, 16, 8, 4, 2, 1, 0, 0, 0, 0,
15   // Define a variable to select 8 contiguous data in the array sequentially
16   static byte offset;
17   // Define a variable to control the speed
18   static unsigned int counter;
19   if (counter++ % 8 == 0)           // Reduce the self-increasing speed of offset
20       offset < 15 ? offset++ : offset = 0;// offset increases
21   // Out put PWM wave
22   for (int i = 0; i < 64; i++) { // The cycle of PWM is 64 cycles
23       byte data = 0;           // Define a variable to represent the output state of
this loop
24       for (int j = 0; j < 8; j++) // Calculate the output state of this loop
25   {
26       if (i < pulse[j + offset]) // Calculate the LED state according to the pulse width
27   {
28           data |= 0x01 << j;    // Represent the LED state with the corresponding bit
of a variable
29       }
30   }
31   // Send the state of LED to 74HC595
32   writeData(data);
33   }
34   }
35
36 void writeData(int value) {
37   // Make latchPin output low level
38   digitalWrite(latchPin, LOW);
39   // Send serial data to 74HC595
40   shiftOut(dataPin, clockPin, MSBFIRST, value);
41   // Make latchPin output high level, then 74HC595 will update the data to parallel
output
42   digitalWrite(latchPin, HIGH);
43 }
```

First we define an array to modulate different PWM pulse widths for LEDs, in doing so different LEDs can emit varied brightness. Starting from the array index 0, take an array of 8 adjacent numbers as the LED duty cycle and output it one at a time. Increasing the starting index number in turn, then it will create a flowing effect.

14	const byte pulse[] = {0, 0, 0, 0, 0, 0, 0, 0, 64, 48, 32, 16, 8, 4, 2, 1, 0, 0, 0, 0,
----	---

Define a variable to select 8 adjacent numbers in an array in turn.

16	static byte offset;
----	---------------------

Define a variable to control the speed of the raindrops.

```
18 static unsigned int counter;
```

Reduce the auto-increment speed of the variable offset with the following code.

```
19 if (counter++ % 8 == 0)           // Reduce the self-increasing speed of offset  
20     offset < 15 ? offset++ : offset = 0; // offset increases
```

We use software to output PWM waveform. Define the cycle of PWM to be 64 cycles and determine the pulse width of LED (that is brightness) according to the selected eight numbers of the array in each cycle.

```
22 for (int i = 0; i < 64; i++) { // The cycle of PWM is 64 cycles  
23     byte data = 0;           // Define a variable to represent the output state of  
this loop.  
24     for (int j = 0; j < 8; j++) // Calculate the output state of this loop  
25     {  
26         if (i < pulse[j + offset]) // Calculate the LED state according to the pulse width  
27         {  
28             data |= 0x01 << j;      // Represent the LED state with the corresponding bit  
of variable  
29         }  
30     }
```

Due to the change of the variable offset, the LED will output the brightness that the eight adjacent numbers represents in the array, and form the dropping-rain effect.

Verify and upload the code, and then you will see the dropping-rain effect that LED forms.

Other Components

This kit also includes other common components that can help your ideas come true. Now we will introduce components not mentioned in the previous section.

Component Knowledge

Toggle switch

Like push button switch, toggle switch is also a kind of switching devices. The difference is that toggle switch is suitable for long-time open or close circuits.

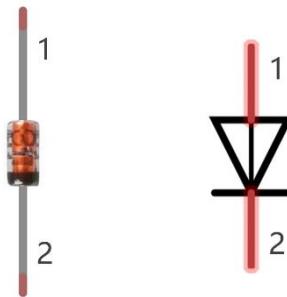


When the lever is moved to the left, pin 1, 2 get conducted, and pin 2, 3 are disconnected from each other;
When the lever is moved to the right, pin 2,3 get conducted, and pin 1,2 are disconnected from each other;

Switch diode

There are several types of diodes. We have used 1N4001 before, which is a common rectifier diode and commonly used in ac rectifier.

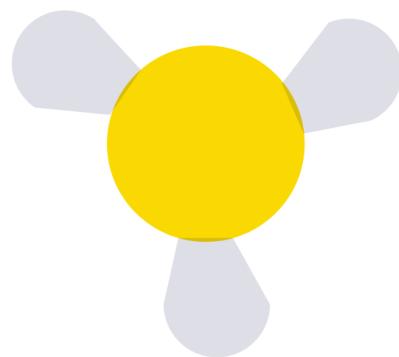
Here is 1N4148, which is a kind of high-speed switching diodes and is characterized by a relatively rapid switching.



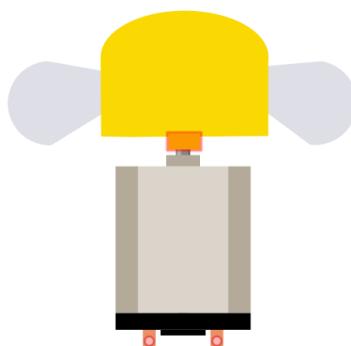
For the switching diode, the changing time from conduction to cut off or from cut off to conduction is shorter than general diode and it is mainly used in electronic computer, pulse and switching circuits.

Motor soft fan blade

This is the fan blade of the motor.



The installation of soft fan blades is as follows:

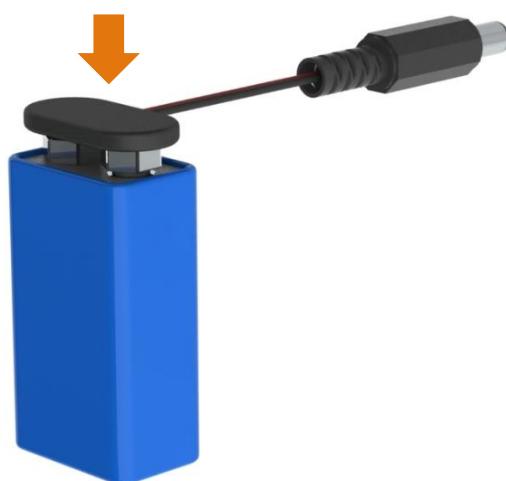


9V battery cable

A 9V battery cable can connect a 9 V battery, which can supply power for control board.



The installation of 9V battery cable is as follows:





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 this 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 are interesting in Processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, micro:bit, robots, smart cars and other interesting products, please visit our website:

<http://www.freenove.com/>

We will continue to launch fun, cost-effective, innovative and exciting products.

Thank you again for choosing Freenove products.



Appendix

ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	Ø	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	H	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	Ø	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	:	91	5B	[123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Resistor Color Code

The diagram illustrates two methods for resistor color coding:

- 4-Band-Code:** A resistor with four bands. The first three bands represent the resistance value (e.g., 560), and the fourth band represents the tolerance (e.g., ±5%).
- 5-Band-Code:** A resistor with five bands. The first four bands represent the resistance value (e.g., 237), and the fifth band represents the tolerance (e.g., ±1%).

Below the diagrams is a table mapping colors to their corresponding values for both the 4-band and 5-band codes.

COLOR	1 ST BAND	2 ND BAND	3 RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Annotations on the left side of the table indicate the color mapping for each band:

- 1ST BAND: Black, Brown, Red, Orange, Yellow, Green, Blue, Violet, Grey, White
- 2ND BAND: Black, Brown, Red, Orange, Yellow, Green, Blue, Violet, Grey, White
- 3RD BAND: Black, Brown, Red, Orange, Yellow, Green, Blue, Violet, Grey, White
- MULTIPLIER: Black, Brown, Red, Orange, Yellow, Green, Blue, Violet, Grey, White
- TOLERANCE: Gold, Silver

Annotations at the bottom left indicate the total resistance value and tolerance for the 5-band code:

- 0.1%, 0.25%, 0.5%, 1% (Tolerance)
- 237 Ω ± 1% (Total Resistance)