

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
Programming Software	3
First Use	6
Chapter 1 LED Blink	10
Project 1.1 Control LED with Manual Button	10
Project 1.2 Control LED with Control Board	17
Chapter 2 Two LEDs Blink	23
Project 2.1 Two LEDs Blink	23
Chapter 3 LED Bar Graph	30
Project 3.1 LED Bar Graph Display	30
Chapter 4 LED Blink Smoothly	37
Project 4.1 LEDs Emit Different Brightness	37
Project 4.2 LED Blinking Smoothly	42
Chapter 5 Control LED with Push Button Switch	44
Project 5.1 Control LED with Push Button Switch	44
Project 5.2 Change LED State with Push Button Switch	48
Chapter 6 Serial	50
Project 6.1 Send Data through Serial	50
Project 6.2 Receive Data through Serial Port	55
Project 6.3 Application of Serial	59
Chapter 7 ADC	62
Project 7.1 ADC	62
Project 7.2 Control LED by Potentiometer	67
Project 7.3 Control LED by Potentiometer	70
Chapter 8 RGB LED	74
Project 8.1 Control RGB LED through Potentiometer	74
Project 8.2 Multicolored LED	78
Chapter 9 Buzzer	81
Project 9.1 Active Buzzer	81
Project 9.2 Passive Buzzer	87
Chapter 10 Temperature Sensor	90
Project 10.1 Detect the Temperature	90
Chapter 11 Motor	94
Project 11.1 Control Motor by Relay	94
Project 11.2 Control Motor with L293D	101
Chapter 12 Servo	107
Project 12.1 Servo Sweep	107
Project 12.2 Control Servo with Potentiometer	111

Chapter 13 LED Matrix	114
Project 13.1 74HC595.....	114
Project 13.2 LED Matrix.....	120
Chapter 14 LCD1602.....	129
Chapter 14A LCD1602.....	130
Project 14A.1 Display the String on LCD1602.....	130
Project 14A.2 LCD1602 Clock.....	135
Chapter 14B I2C LCD1602.....	144
Project 14B.1 Display the String on I2C LCD1602.....	144
Project 14B.2 I2C LCD1602 Clock.....	148
Chapter 15 Ultrasonic Ranging.....	157
Project 15.1 Ultrasonic Ranging.....	157
Chapter 16 RFID.....	166
Project 16 Read UID.....	166
Project 16.2 Read and write	174
What's Next?.....	178
Appendix	179
ASCII Table	179
Resistor Color Code	180

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.

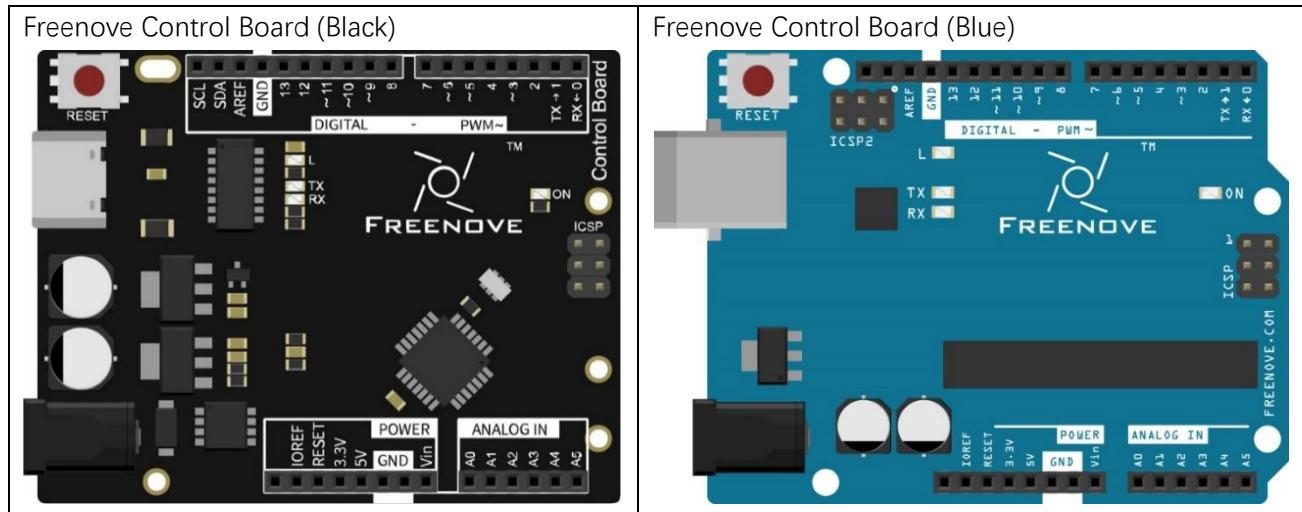
Support email: support@freenove.com

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.

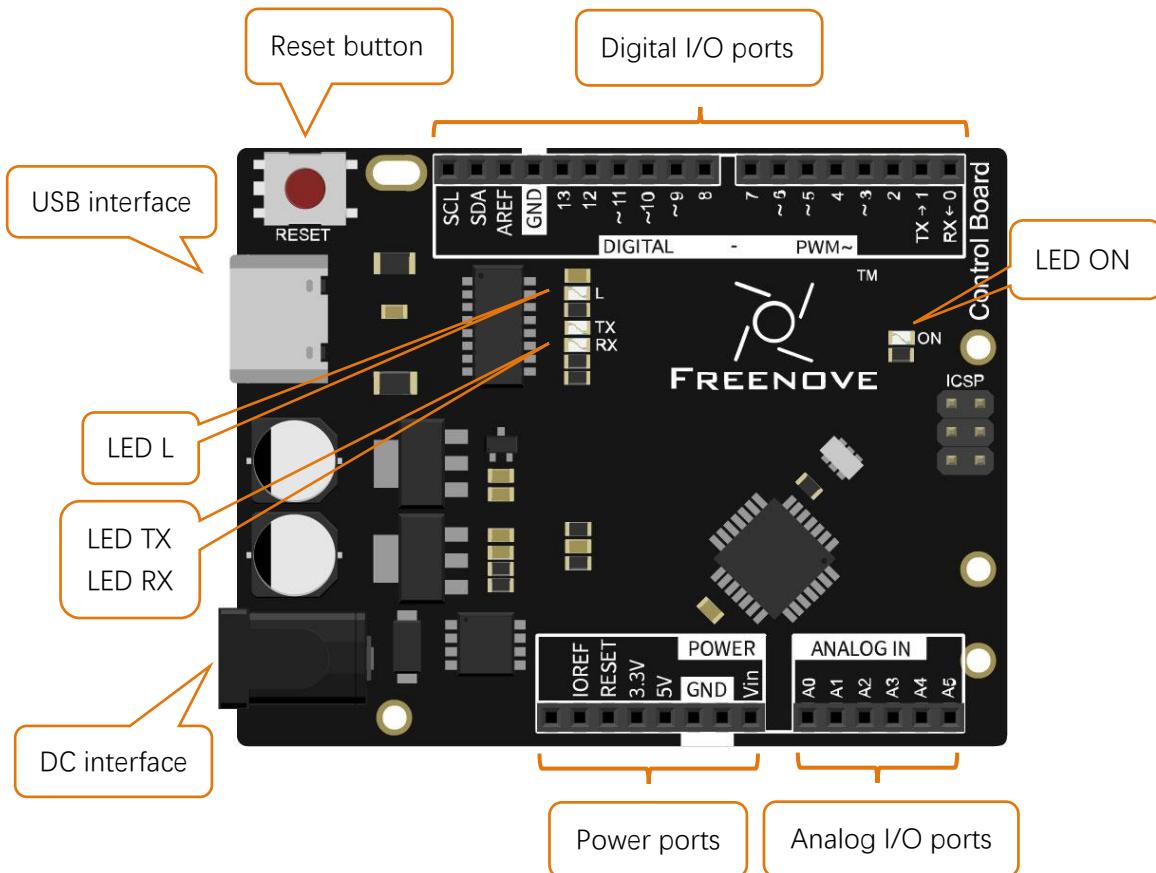
There are multiple versions of Freenove control board. Your purchase may be one of the following:



Note: Although the colors and shapes of these control boards are somewhat different, their ports and functions are the same. They can be replaced with each other, and there is no difference in their usages.

Note: Only the black control board is used to display the hardware connection in this document. The hardware connection of the blue control board is the same.

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.

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/Main/Software>. Select and download a corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer".

The screenshot shows the Arduino Software download page. On the left, there's a large circular Arduino logo. In the center, the text "ARDUINO 1.8.12" is displayed above a detailed description of the software. To the right, there's a sidebar with download links for different operating systems. An orange arrow points from the text "Download the Arduino IDE" to the "Windows Installer" link in the sidebar.

Download the Arduino IDE

ARDUINO 1.8.12

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions.

Windows Installer, for Windows 7 and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
[Get](#)

Mac OS X 10.10 or newer

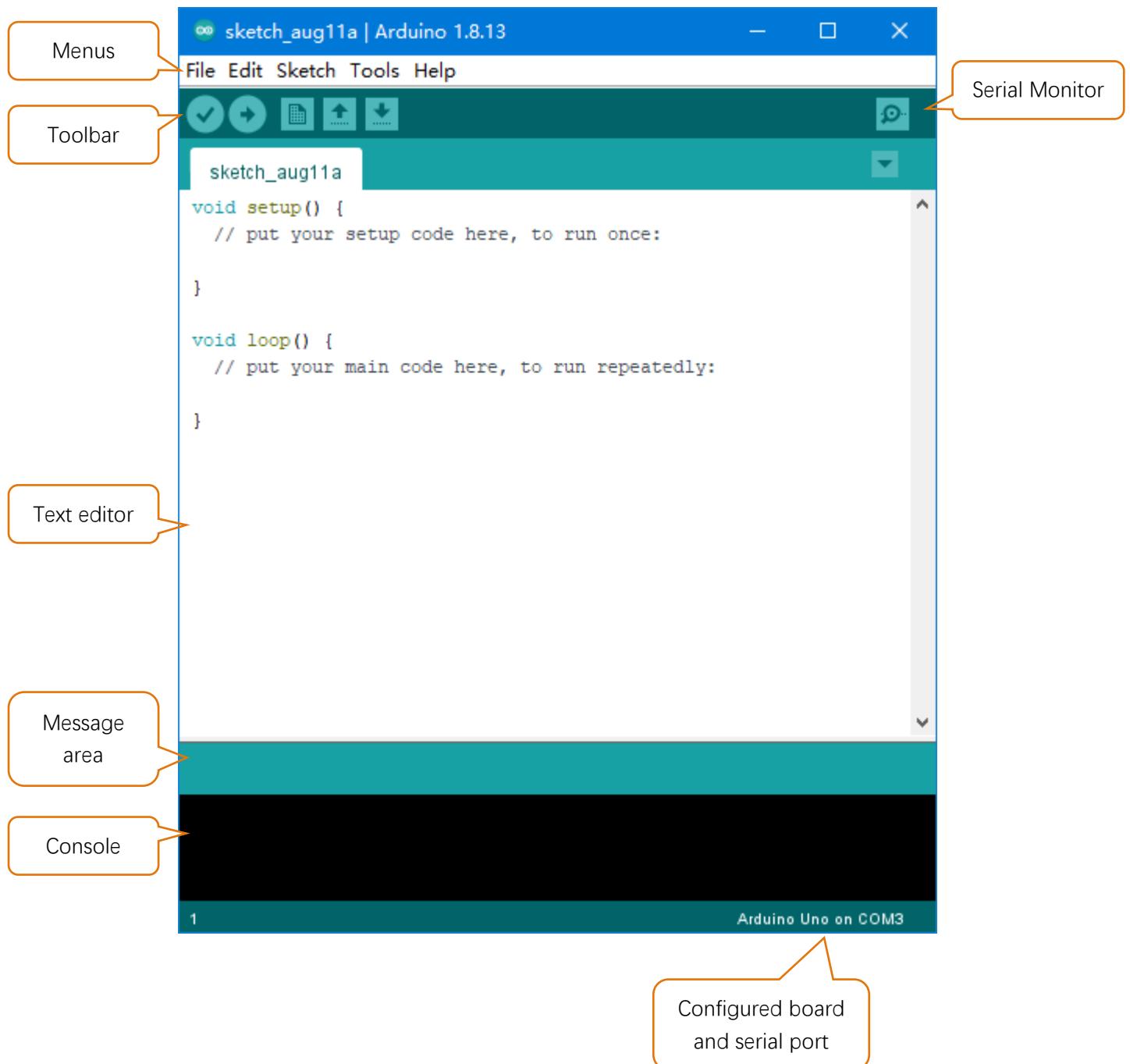
Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

[Release Notes](#)
[Source Code](#)
[Checksums \(sha512\)](#)

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.



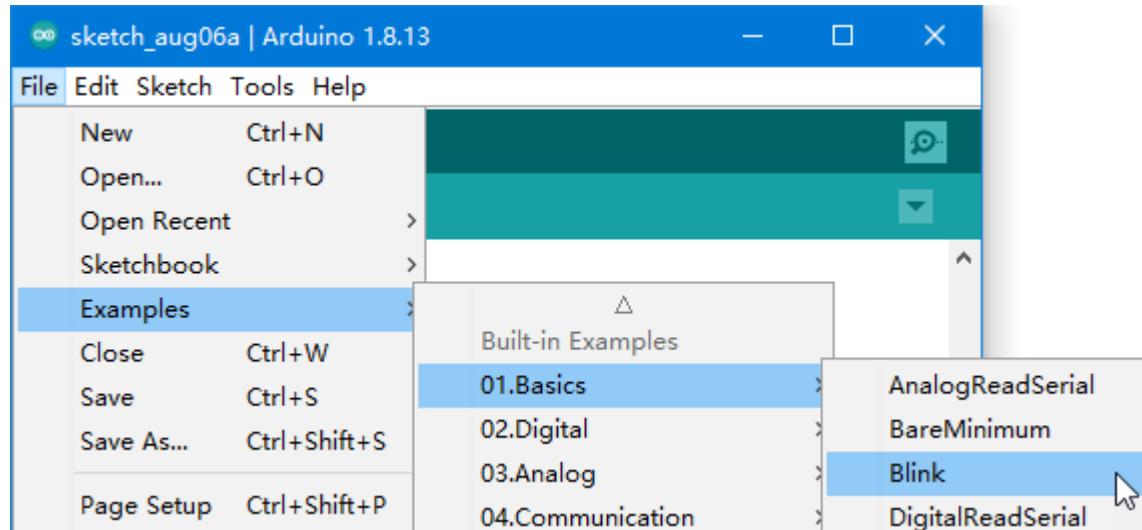
Serial Monitor

Opens the serial monitor.

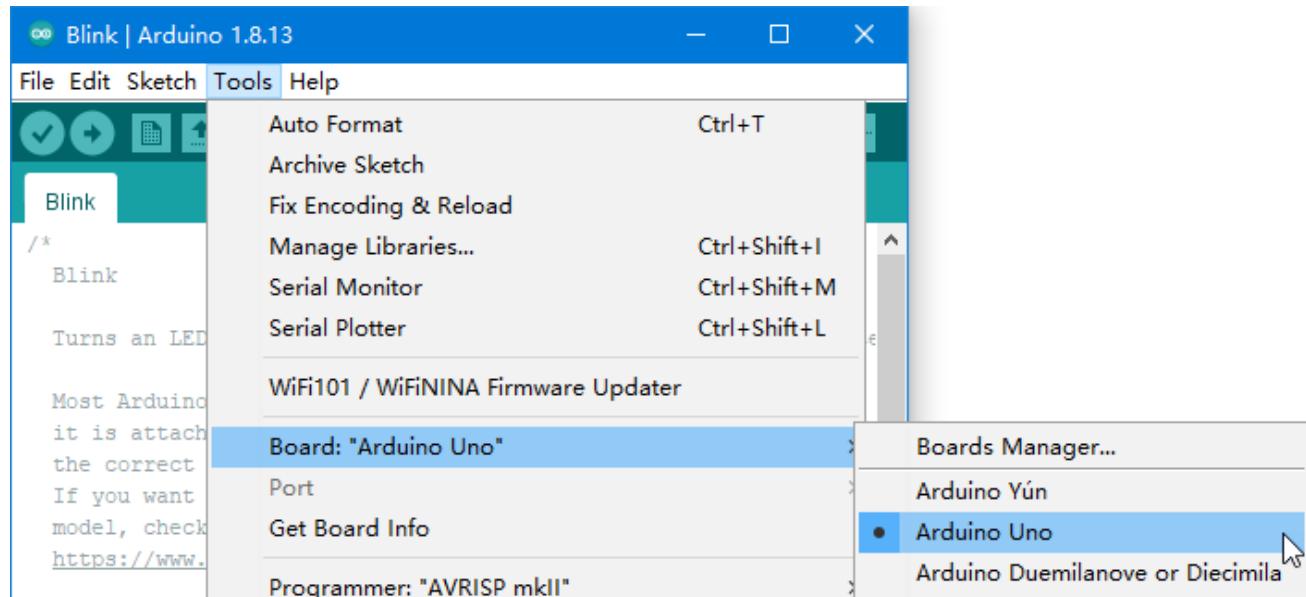
Additional commands are found within five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

First Use

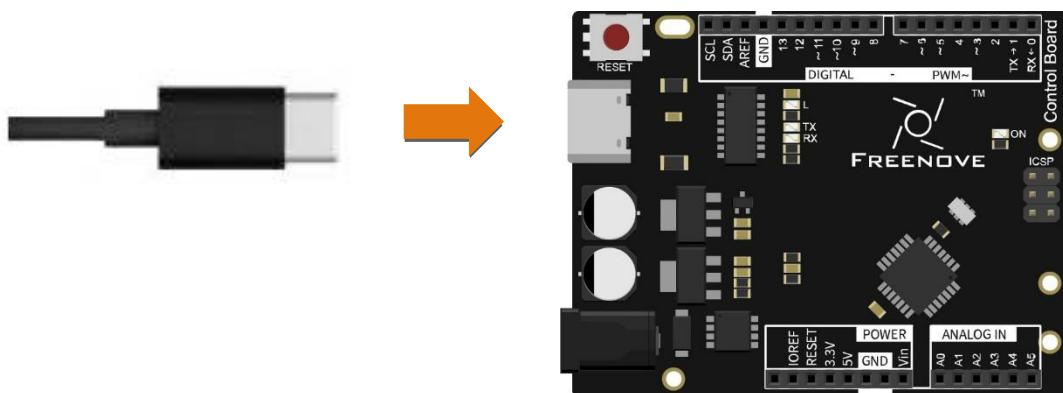
Open the example sketch "Blink".



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



Connect control board to your computer with USB cable.



Need help? Contact support@freenove.com

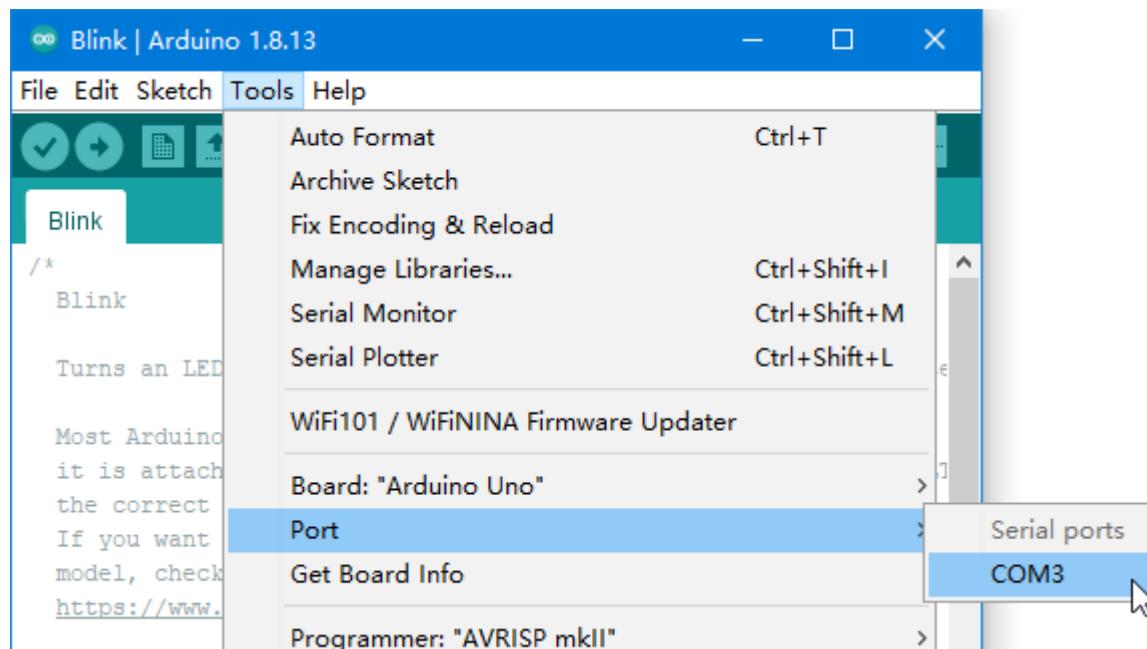
Select the port.

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

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

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

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



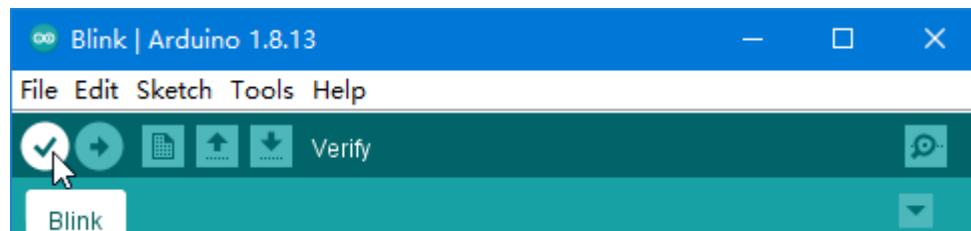
Note: If there is more than one port and you cannot decide which one to choose, disconnect the USB cable and check the port. Then connect the USB cable and check the port again. The new one is the correct port.

If there is no COM port for control board, you may need to install a driver to your computer.

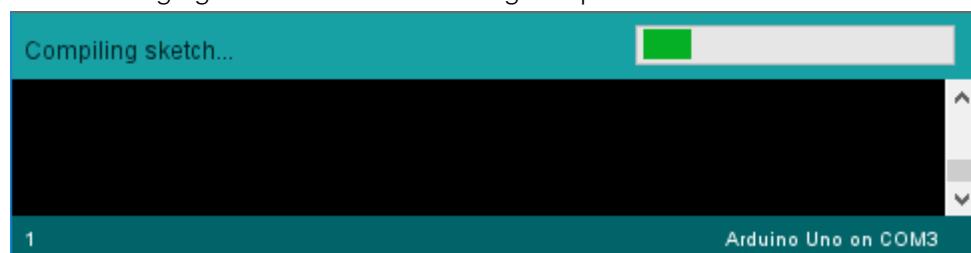
- For blue board, reinstall the latest version of Arduino IDE. During installation, agree to install the driver.
- For black board, see "InstallDriver.pdf" in "Drivers" folder (in the folder contains this Tutorial.pdf).

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

Click "Verify" button.

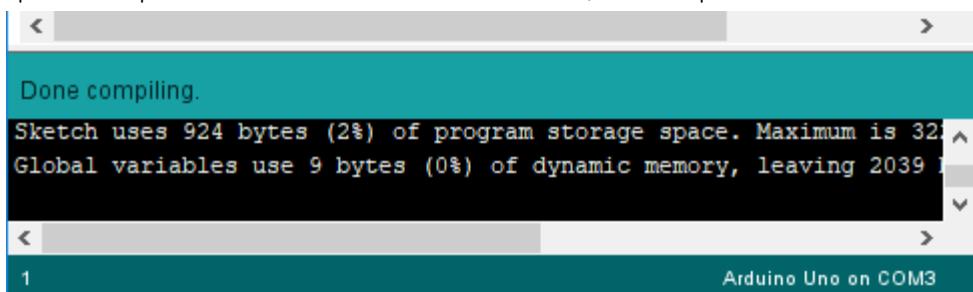


The following figure shows the code being compiled.



Need help? Contact support@freenove.com

Wait a moment for the compiling to be completed. Figure below shows the code size and percentage of space occupation. If there is an error in the code, the compilation will fail and the details are shown here.



The screenshot shows the Arduino IDE interface. The title bar says "Blink | Arduino 1.8.13". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has icons for file operations and an "Upload" button, which is highlighted with a cursor. The main text area displays the following message:
Done compiling.
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.
The status bar at the bottom right shows "Arduino Uno on COM3".

Click "Upload" button.

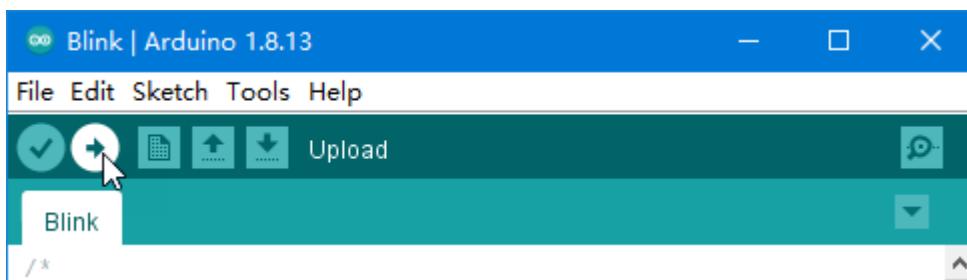
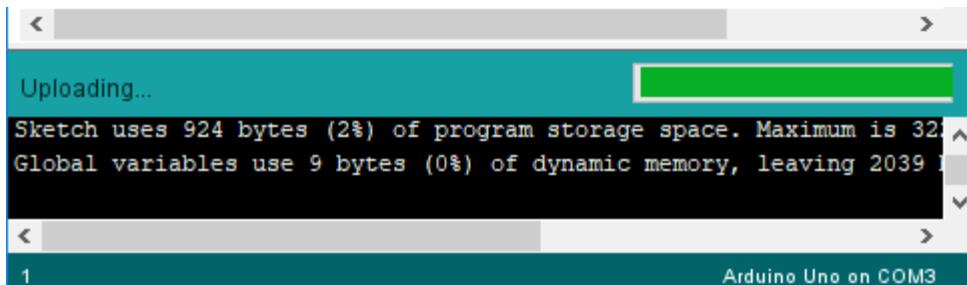
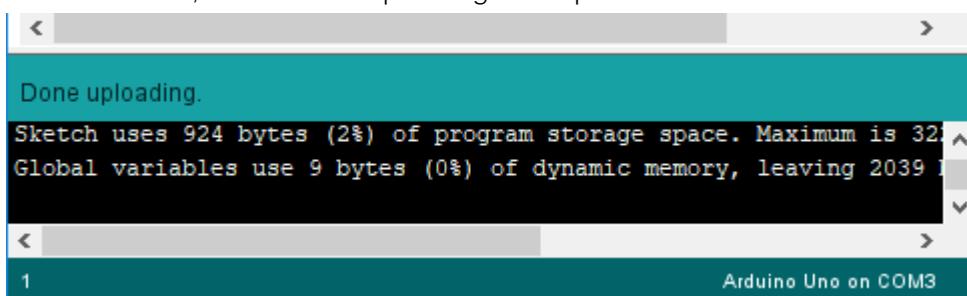


Figure below shows code are uploading.



The screenshot shows the Arduino IDE interface during the upload process. The title bar says "Blink | Arduino 1.8.13". The main text area displays:
Uploading...
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.
The status bar at the bottom right shows "Arduino Uno on COM3".

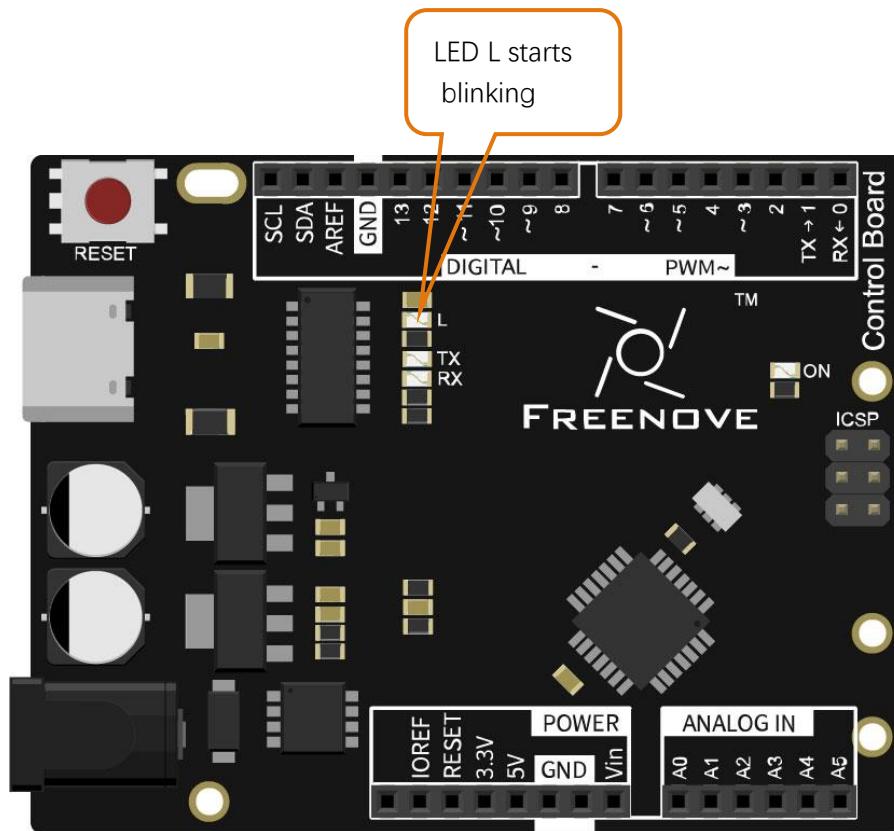
Wait a moment, and then the uploading is completed.



The screenshot shows the Arduino IDE interface after the upload is completed. The title bar says "Blink | Arduino 1.8.13". The main text area displays:
Done uploading.
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.
The status bar at the bottom right shows "Arduino Uno on COM3".

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.

Chapter 1 LED Blink

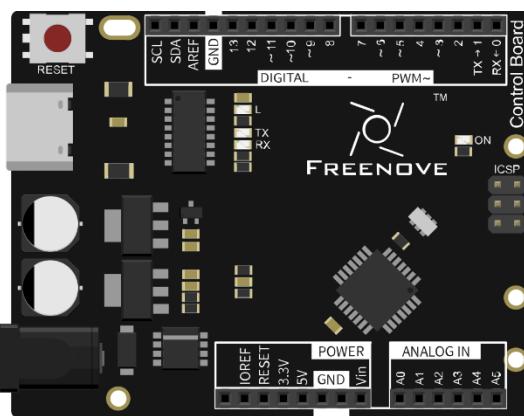
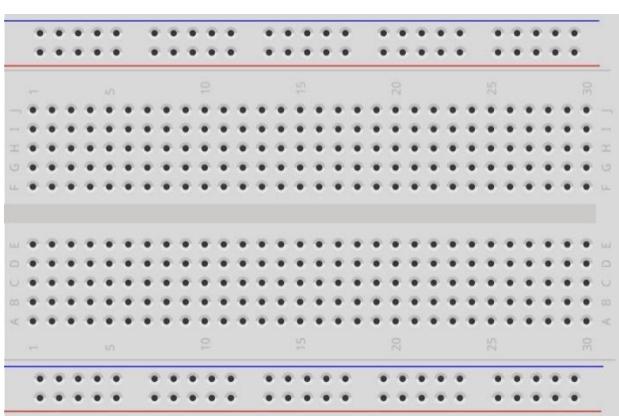
We have previously tried to make the LED marked with "L" blink on the control board. Now let us use electronic components and codes to reproduce the phenomenon, and try to understand their principle.

Project 1.1 Control LED with Manual Button

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

Component List

Note: The control board you received may be black or blue. They are the same in use.
Only the black control board is used to display the hardware connection in this document.

Control board x1	Breadboard x1	
		
USB cable x1	LED x1	Resistor 220Ω x1
		
Jumper M/M x2		Push button x1
		

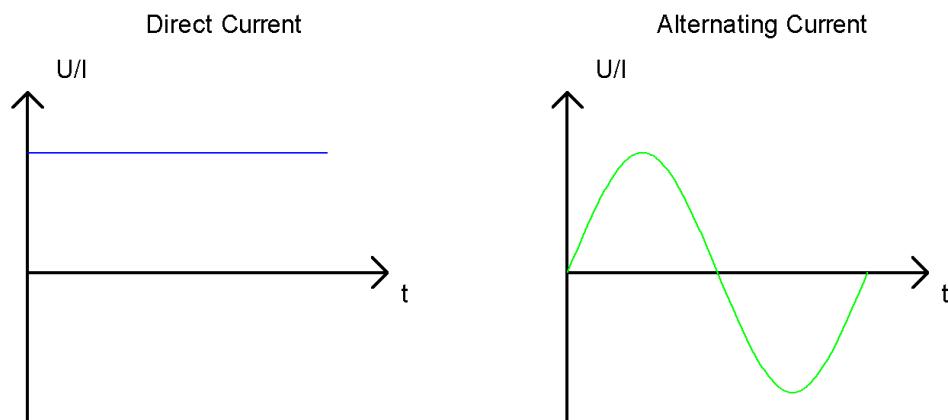
Circuit Knowledge

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:

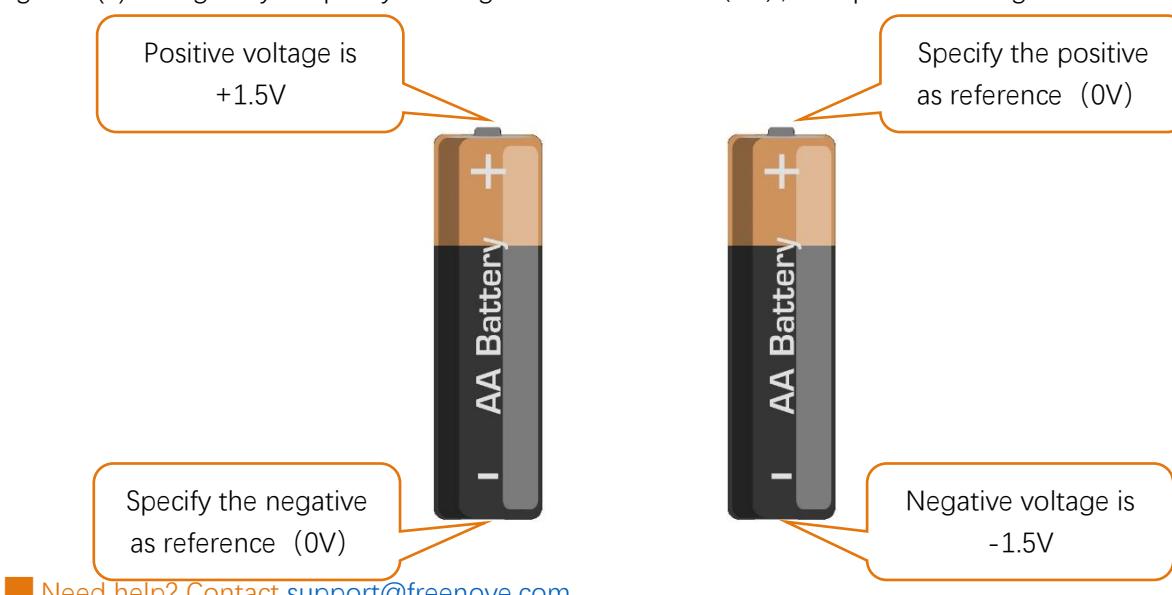


The positive and negative poles of the power supply must not be directly connected, otherwise it may scald you and cause damage to the battery.

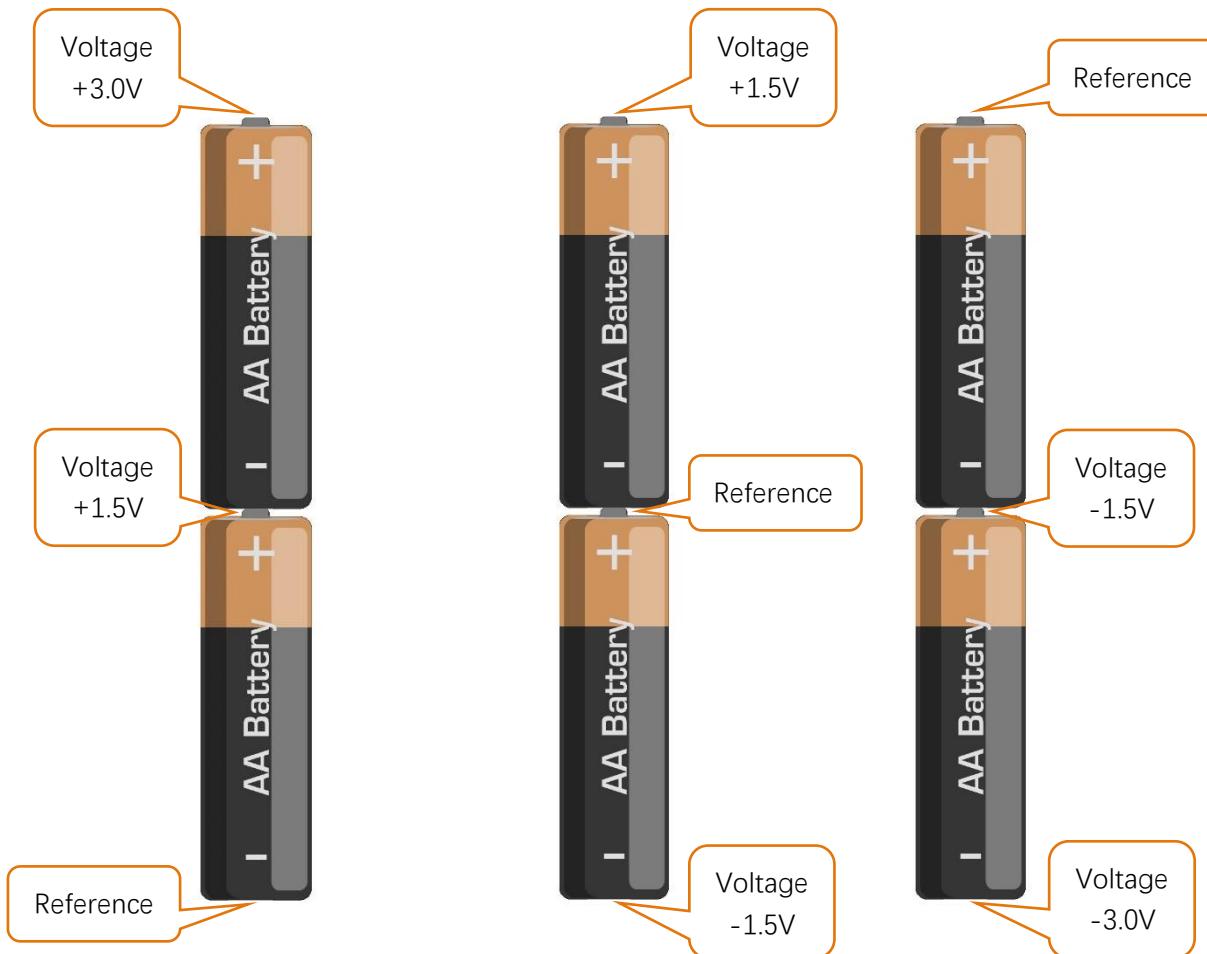
Voltage

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

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:

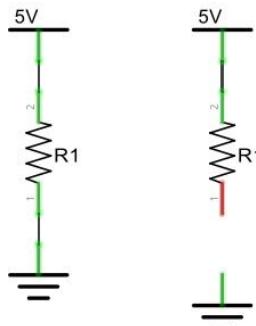


Current

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.

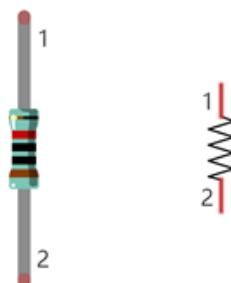


Resistor

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.

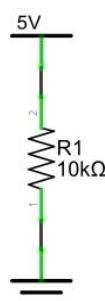
On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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



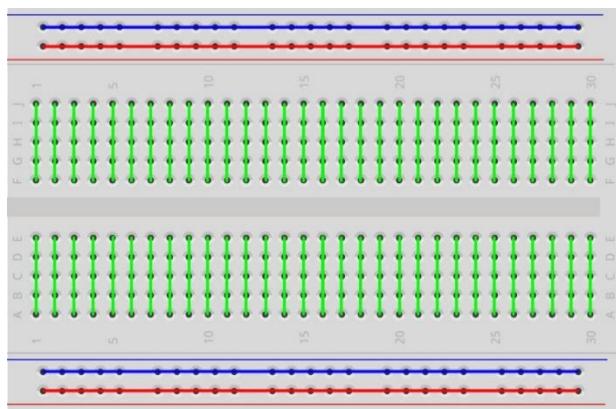
Jumper F/M



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



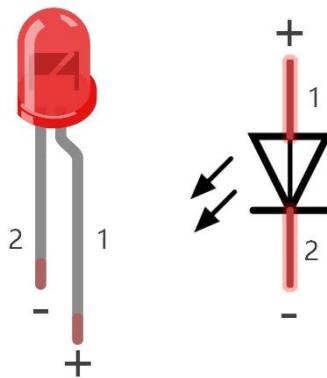
and right sides are the same per the illustration:

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.

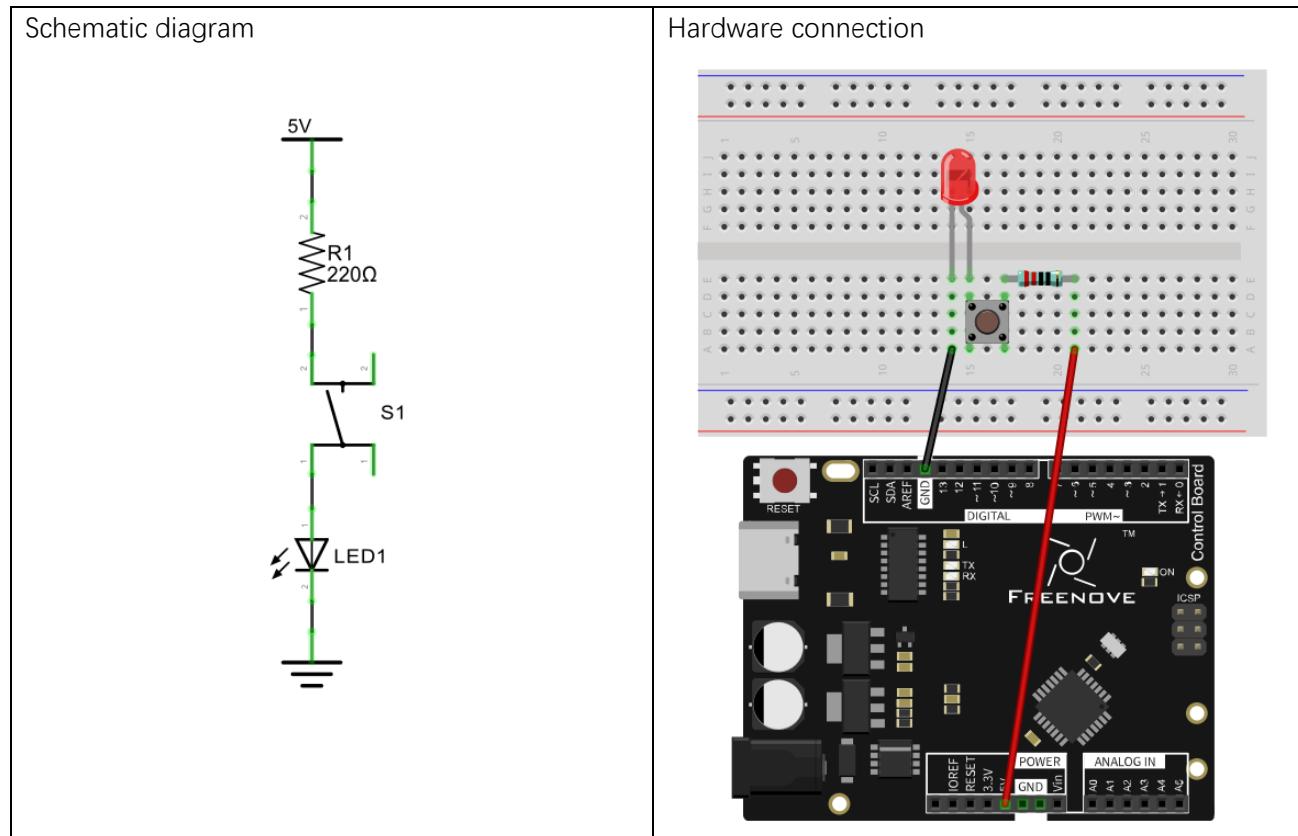
Circuit

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.

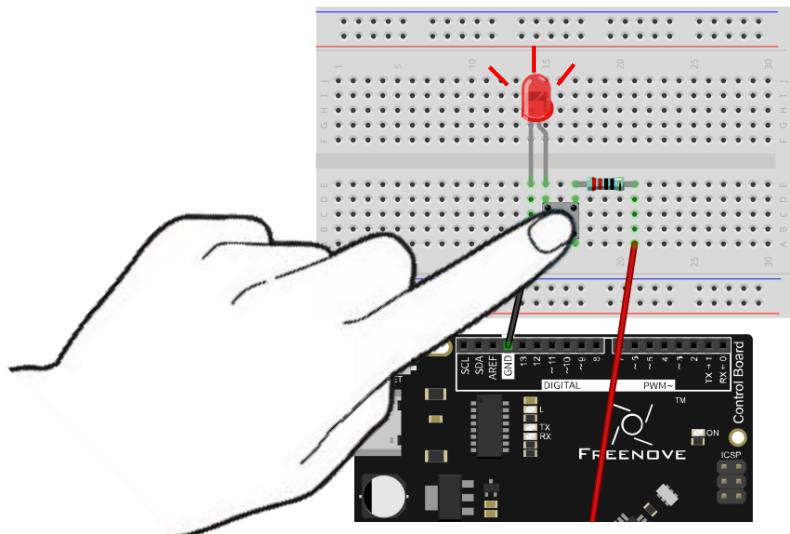
Firstly, connect components with jumpers according to "hardware connection". Secondly, check the connection to confirm that there are no mistakes. Finally, connect the control board to computer with USB cable to avoid short circuit caused by contacting the wires.

Note: The control board you received may be black or blue. They are the same in use.

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



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

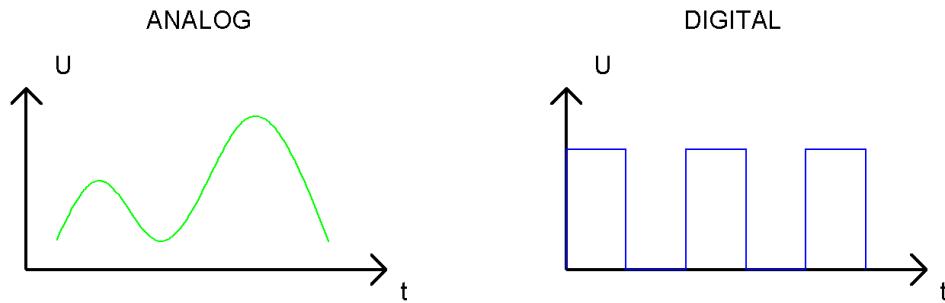
Components are basically the same with those in last section. Push button switch is no more needed.

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.



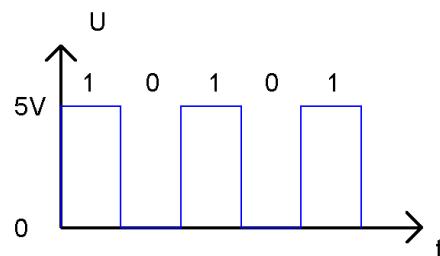
In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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 }

```

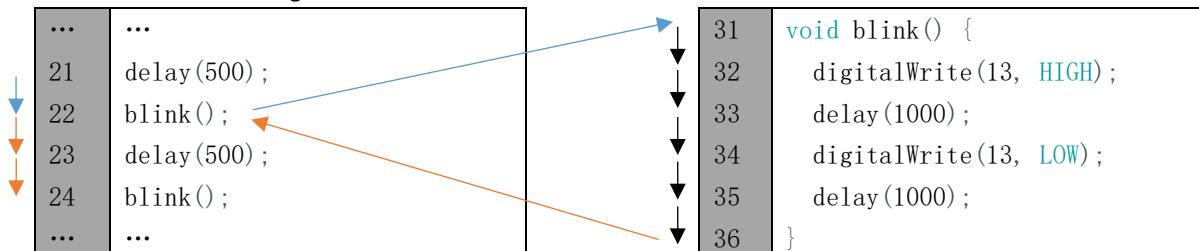
"void" indicates that the function does not return a value (Chapter 4 will detail the return value of functions); "()" its inside is parameters of a function (Chapter 2 will detail the parameters of the functions). No content inside it indicates that this function has no parameters;

"{}" 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

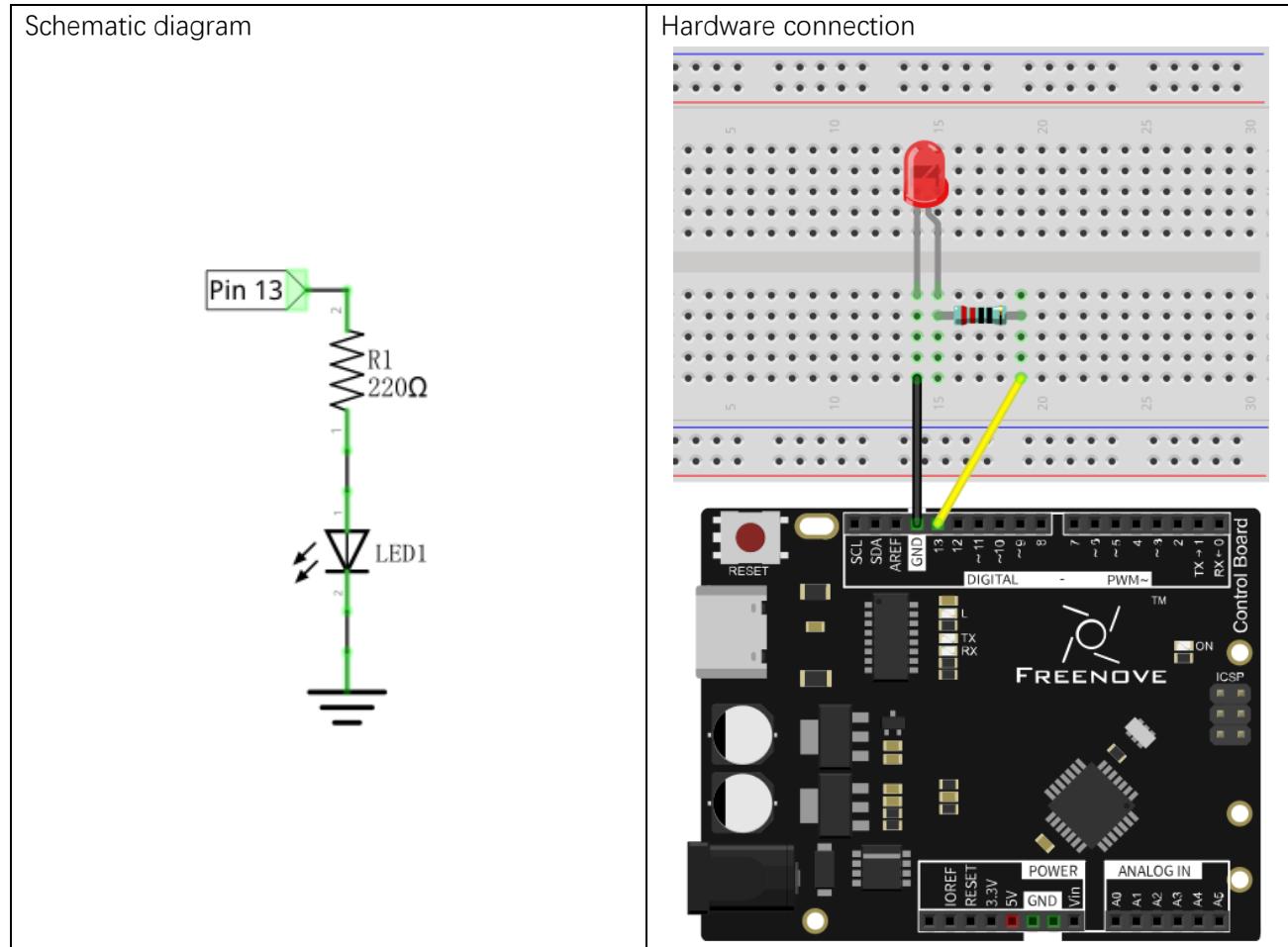
```

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.

Note: The control board you received may be black or blue. They are the same in use.

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



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 }
```

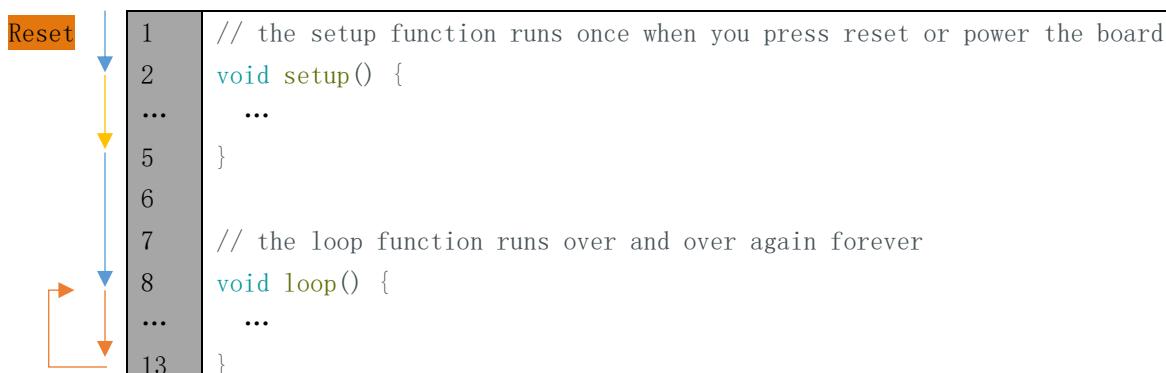
The code usually contains two basic functions: void setup() and void loop().

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.

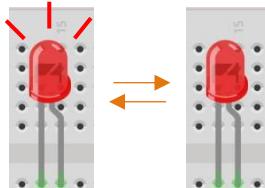
```
11 digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
12 delay(1000); // wait for a second
```

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.



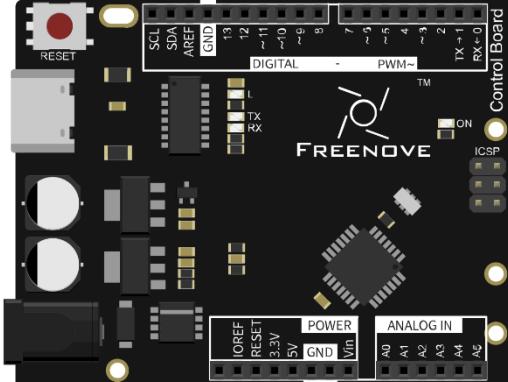
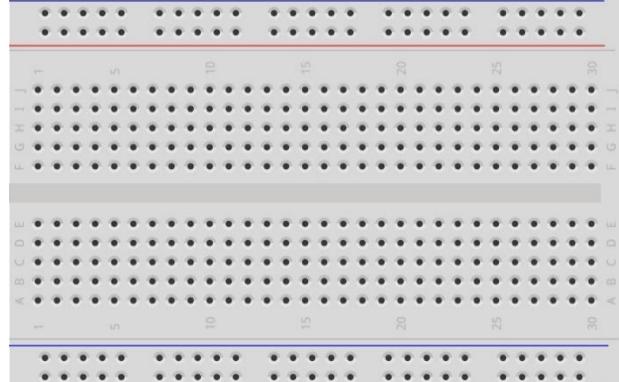
Chapter 2 Two LEDs Blink

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	Breadboard x1
	
USB cable x1	LED x2
	
Jumper M/M x3	Resistor 220Ω x2
	

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
```

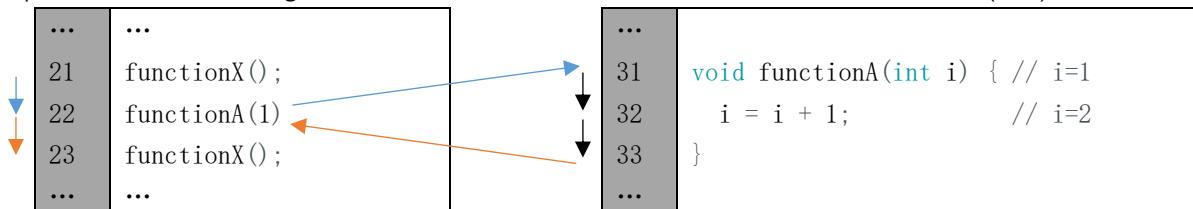
Next, we will define a function with a parameter as below:

```
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:

```

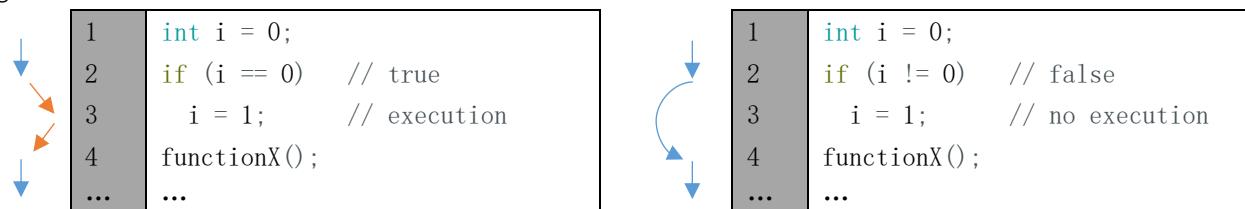
1 boolean isTrue;
2 isTrue = 1 < 2;           // after the execution, "isTrue" is true.
3 isTrue = 1 > 2;           // after the execution, "isTrue" is false.

```

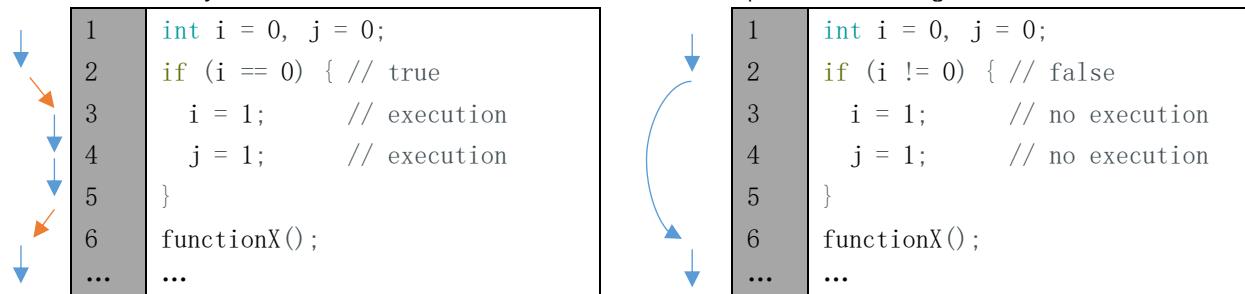
There are other relational operators such as "==" (equal to), ">=" (greater than or equal to), "<=" (less than or equal to) and "!=" (not equal to).

Conditional statement

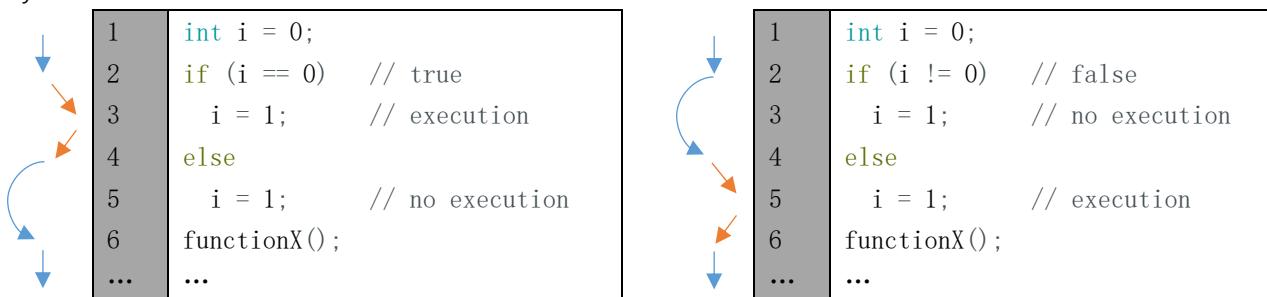
Conditional statements are used to decide whether or not to execute the program based on the result of judgment statement.



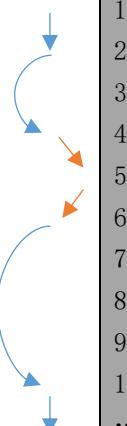
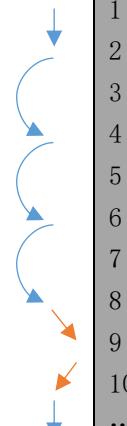
When there are many statements needed to be executed, we can put them into "{}":



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



In addition, it can judge multiple conditions.

 <pre> 1 int i = 1; 2 if (i == 0) // false 3 i = 1; // no execution 4 else if (i == 1)// true 5 i = 2; // execution 6 else if (i == 2) 7 i = 3; // no execution 8 else 9 i = 4; // no execution 10 functionX(); ... </pre>	 <pre> 1 int i = 3; 2 if (i == 0) // false 3 i = 1; // no execution 4 else if (i == 1)// false 5 i = 2; // no execution 6 else if (i == 2)// false 7 i = 3; // no execution 8 else 9 i = 4; // execution 10 functionX(); ... </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Circuit

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

Schematic diagram	Hardware connection

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.

```

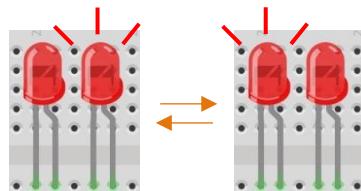
1 // set pin numbers:
2 int led1Pin = 4;           // the number of the LED1 pin
3 int led2Pin = 5;           // the number of the LED2 pin
4
5 void setup() {
6     // initialize the LED pin as an output:
7     pinMode(led1Pin, OUTPUT);
8     pinMode(led2Pin, OUTPUT);
9 }
10
11 void loop() {
12     digitalWrite(led1Pin, 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 = 4; // the number of the LED1 pin
3 int led2Pin = 5; // 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 = 4;           // the number of the LED1 pin  
3 int led2Pin = 5;           // 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.

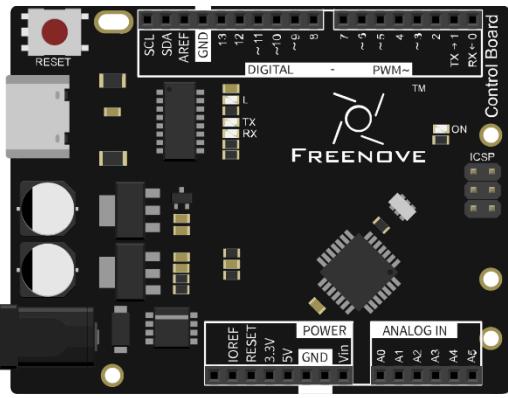
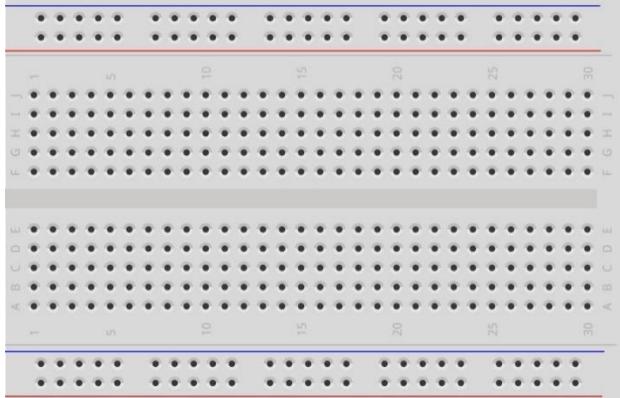
Chapter 3 LED Bar Graph

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

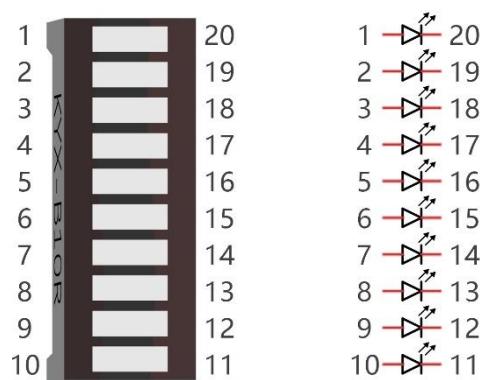
Control board x1	Breadboard x1	
		
USB cable x1	LED bar graph x1	Resistor 220Ω x10
		
Jumper M/M x11		
		

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 $i=2$, $j=2$;

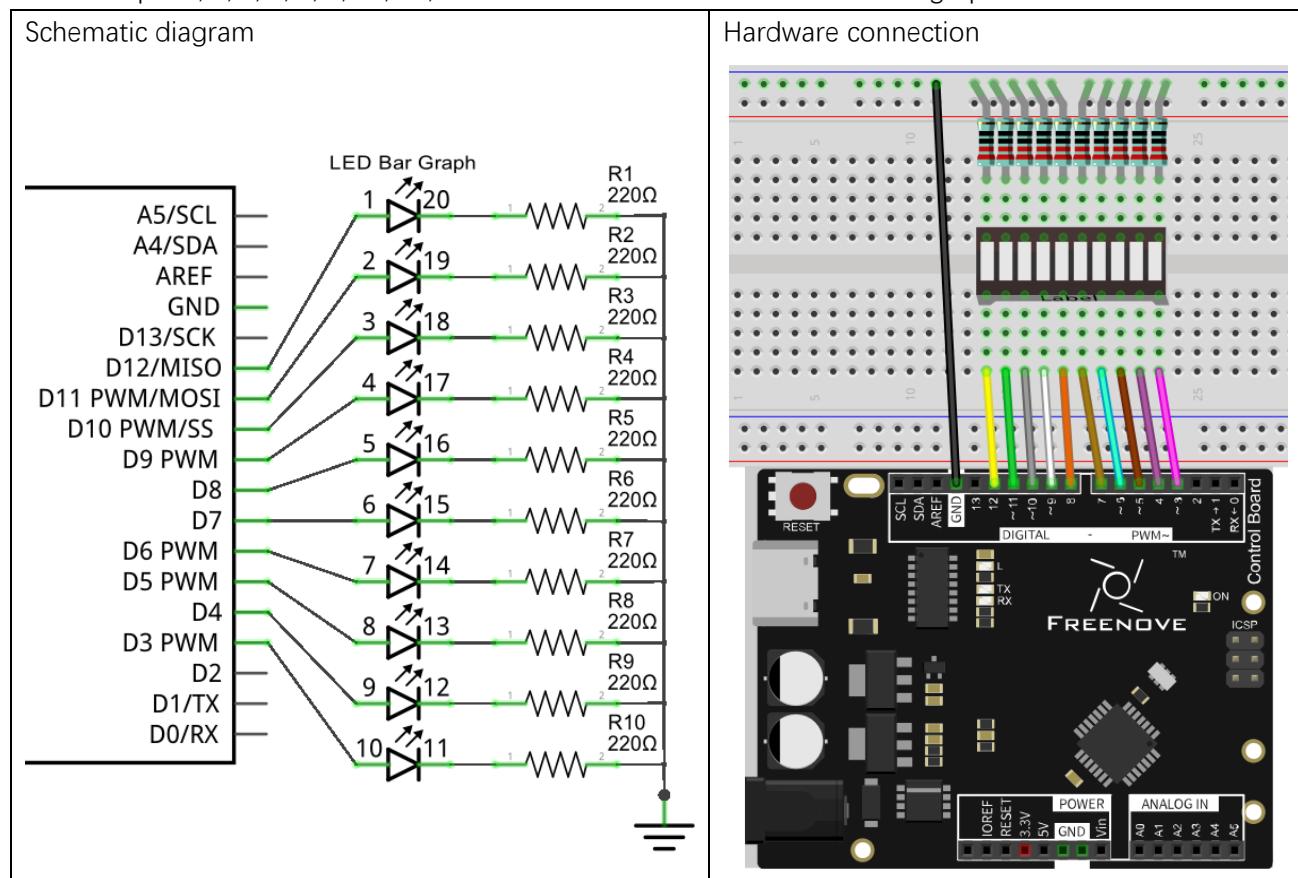
Third time: $i<2$ is tenable, $i=2$ is not tenable. The execution of loop statements is completed. Statement $i=5$ will be executed next.

Operator `++`, `--`

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

Circuit

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



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 bar graph LED 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 }
```

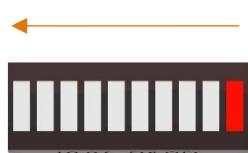
Define a function to turn ON a certain LED on the LED bar graph and turn OFF the other LEDs.

```
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     // make the "i"th LED of LED bar graph on and the others off 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.



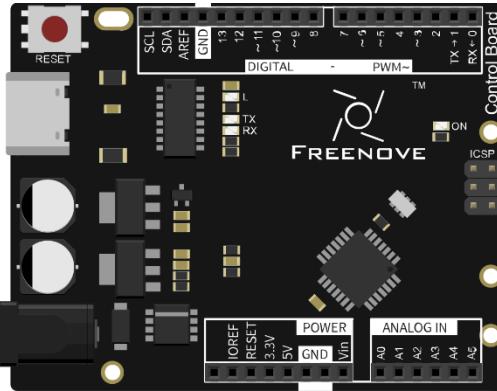
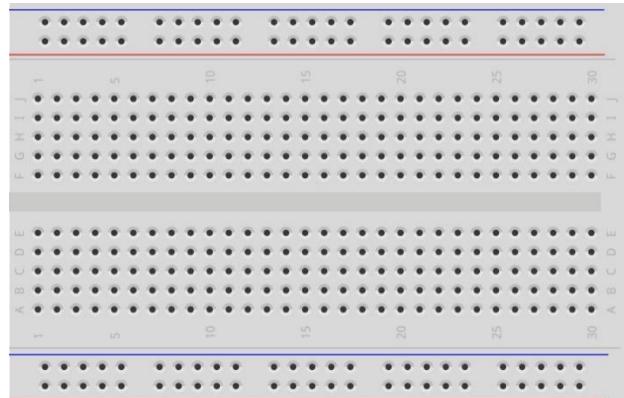
Chapter 4 LED Blink Smoothly

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	Breadboard x1
	
USB cable x1	LED x4
	
Jumper M/M x5	Resistor 220Ω x4
	

Circuit Knowledge

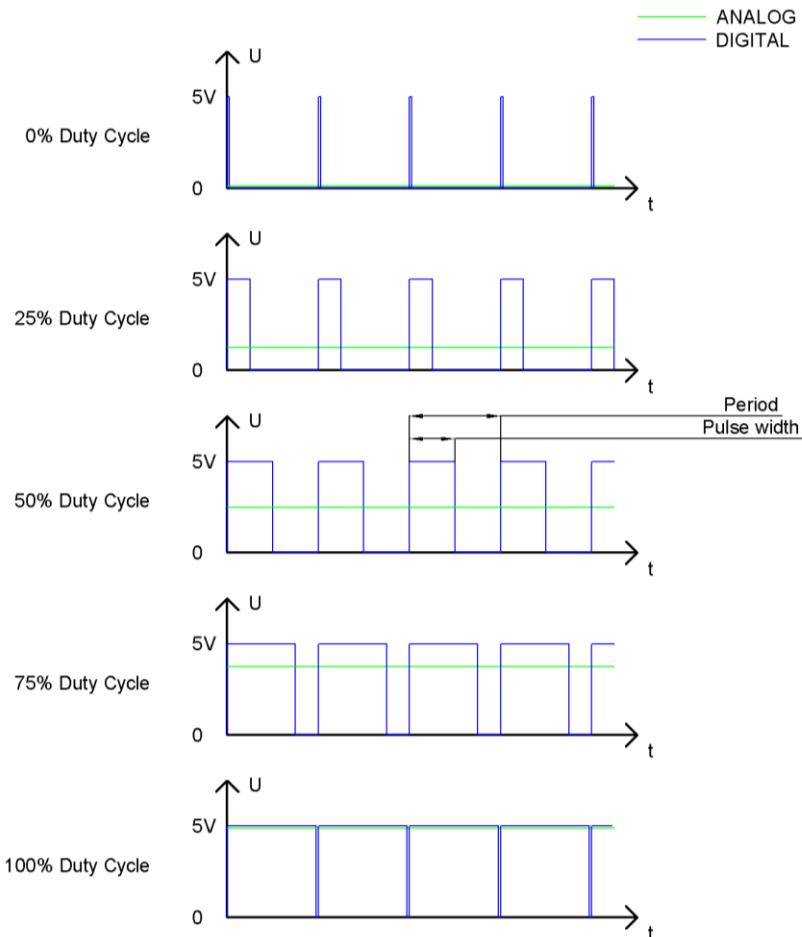
At first, let us learn how to use the circuit to make LED emit different brightness of light,

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:

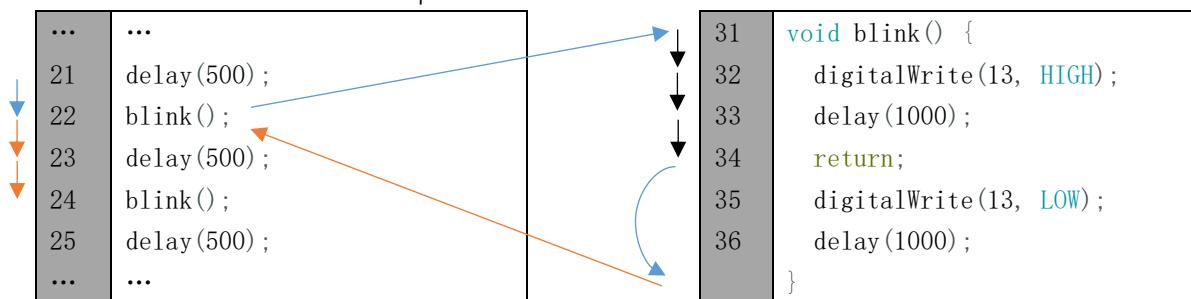
```
1 delay(300);
```

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:

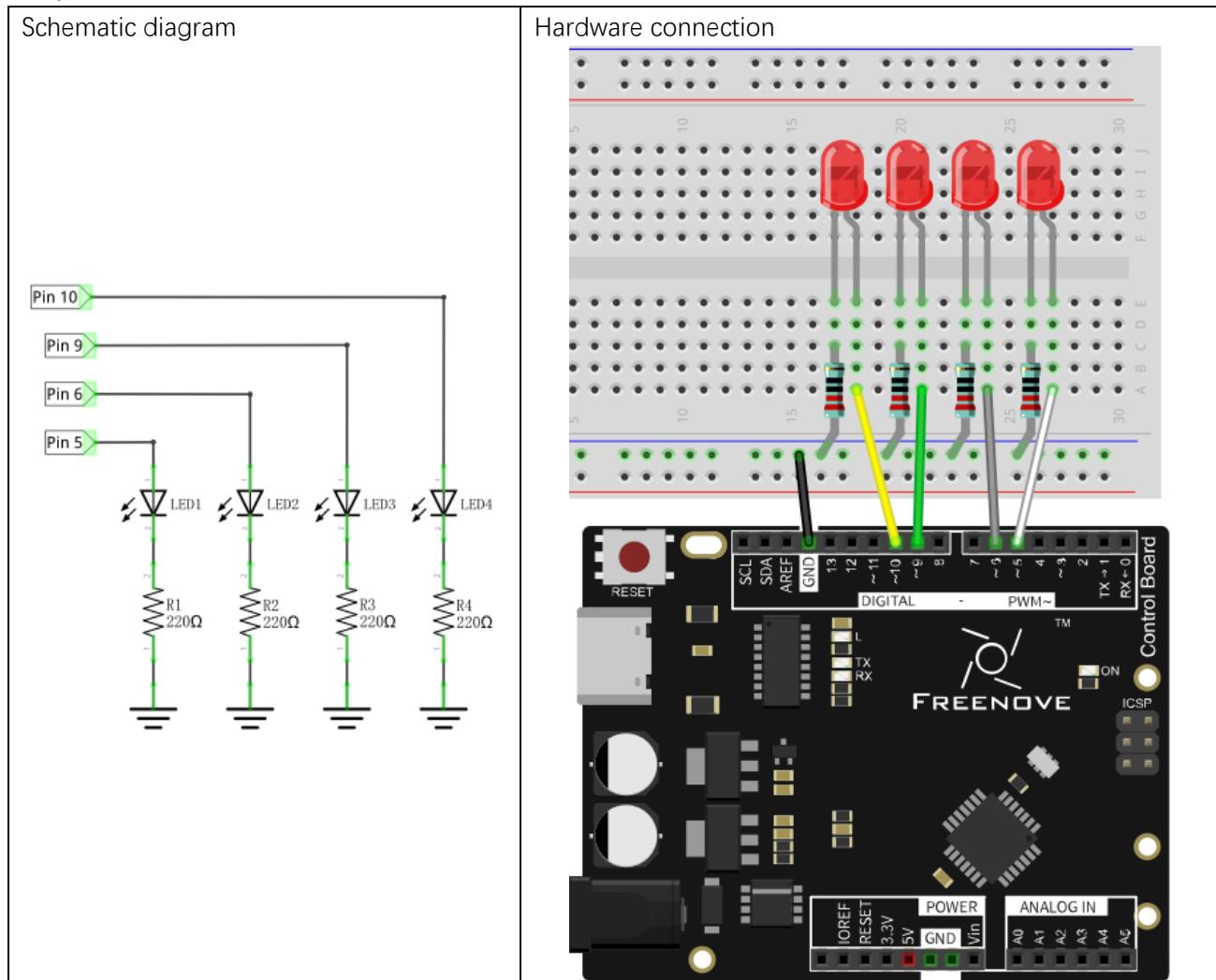
```
1 return;
```

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.



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 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 analogWrite(ledPin1, map(2, 0, 100, 0, 255));
```

analogWrite(pin, value)

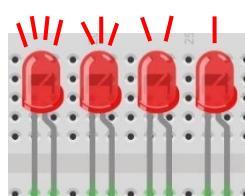
Arduino IDE provides the function, `analogWrite(pin, value)`, which can make ports directly output PWM waves. Only the digital pin marked with "˜" symbol on the control board can use this function to output PWM waves. In the function called `analogWrite(pin, value)`, the parameter "pin" specifies the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

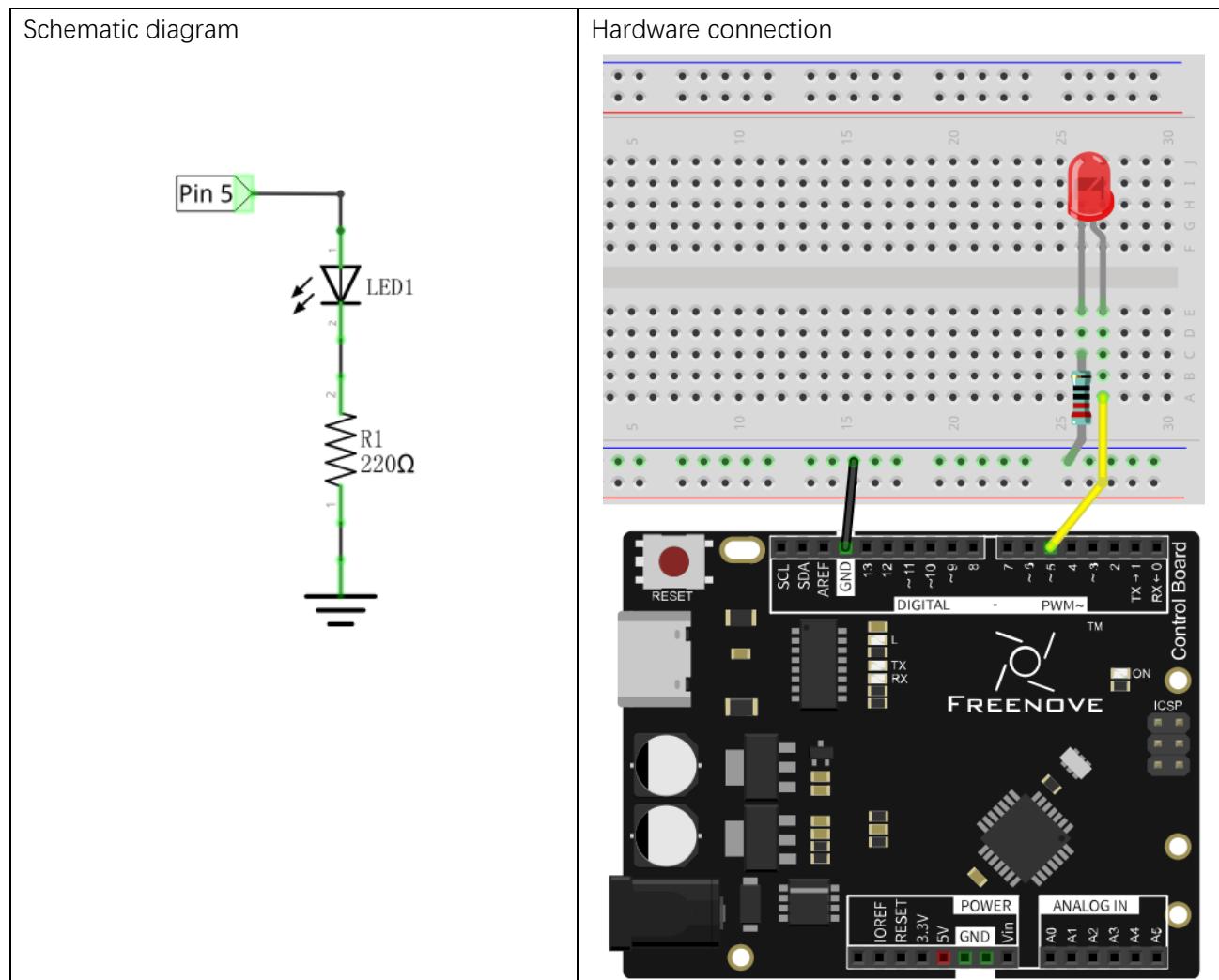
We will learn how to make a LED blink smoothly, that is, breathing light.

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.



Sketch

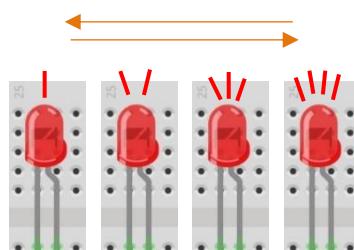
Sketch 4.2.1

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.



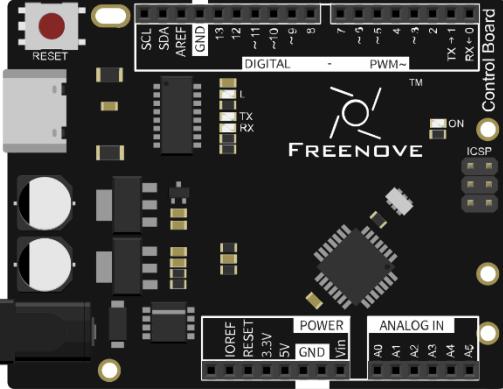
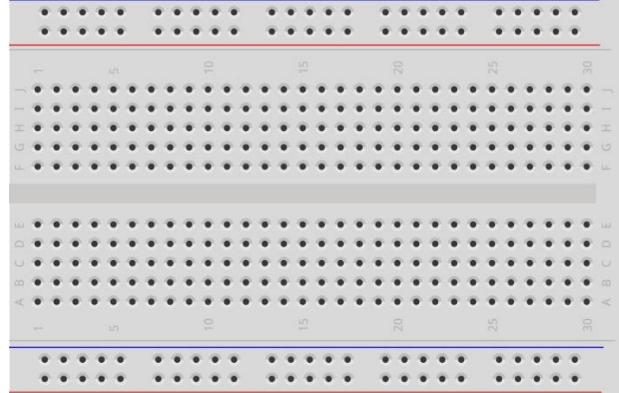
Chapter 5 Control LED with Push Button Switch

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.

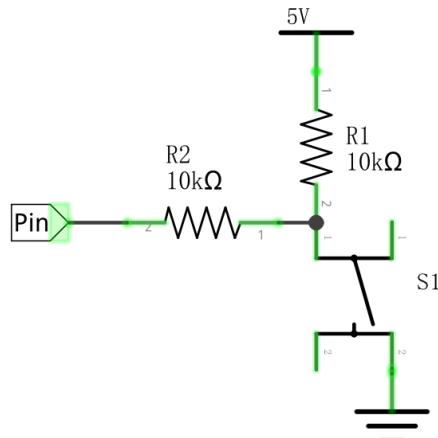
Component List

Control board x1	Breadboard x1
	
USB cable x1	LED x1 Resistor 220Ω x1 Resistor 10kΩ x2 Push button Switch x1
	   
Jumper M/M x4	

Circuit Knowledge

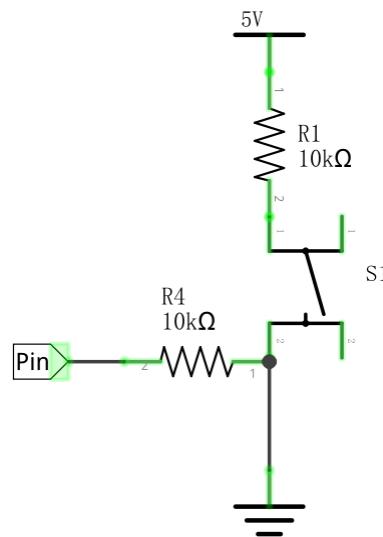
Connection of Push Button Switch

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

Sketch

Sketch 5.1.1

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

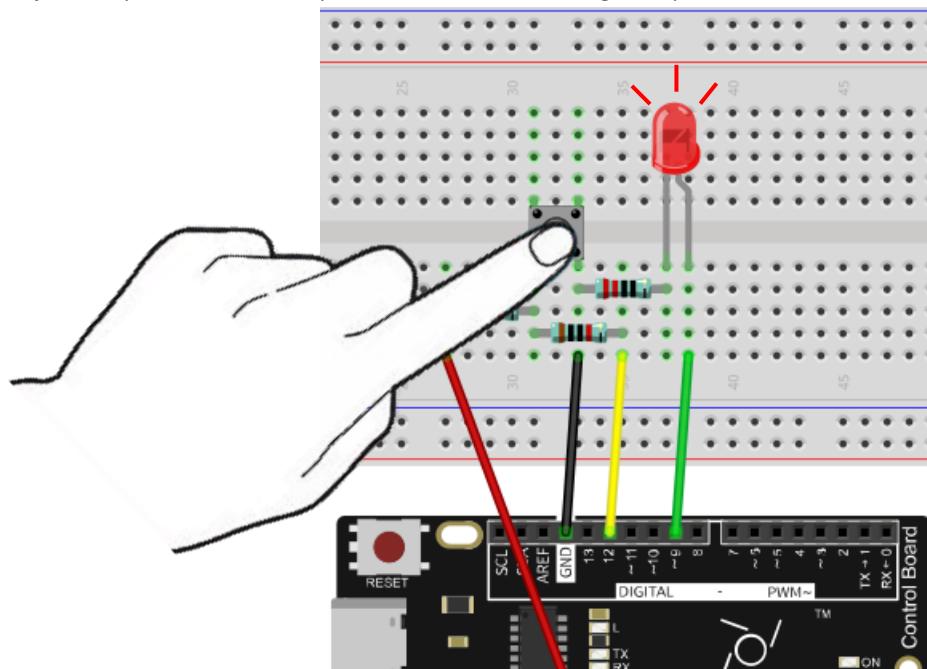
```
1 int buttonPin = 12; // the number of the push button pin
2 int ledPin = 9;      // the number of the LED pin
3
4 void setup() {
5     pinMode(buttonPin, INPUT); // set push button pin into input mode
6     pinMode(ledPin, OUTPUT); // set LED pin into output mode
7 }
8
9 void loop() {
10    if (digitalRead(buttonPin) == 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.

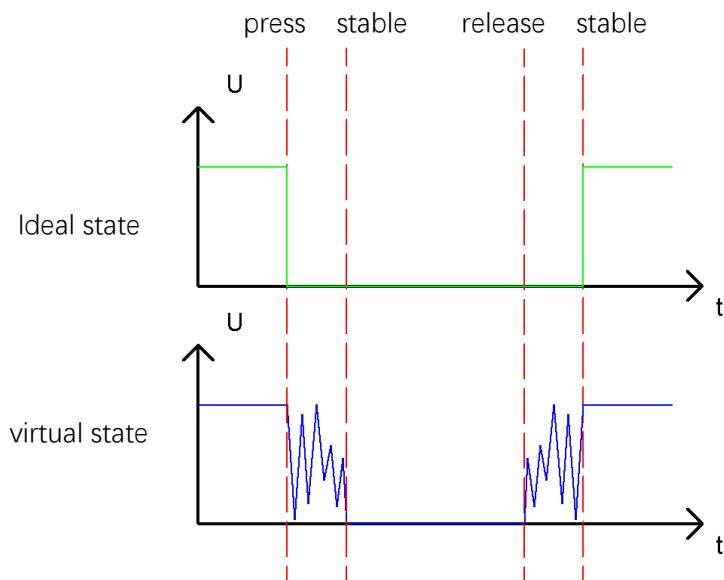
Component List

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.

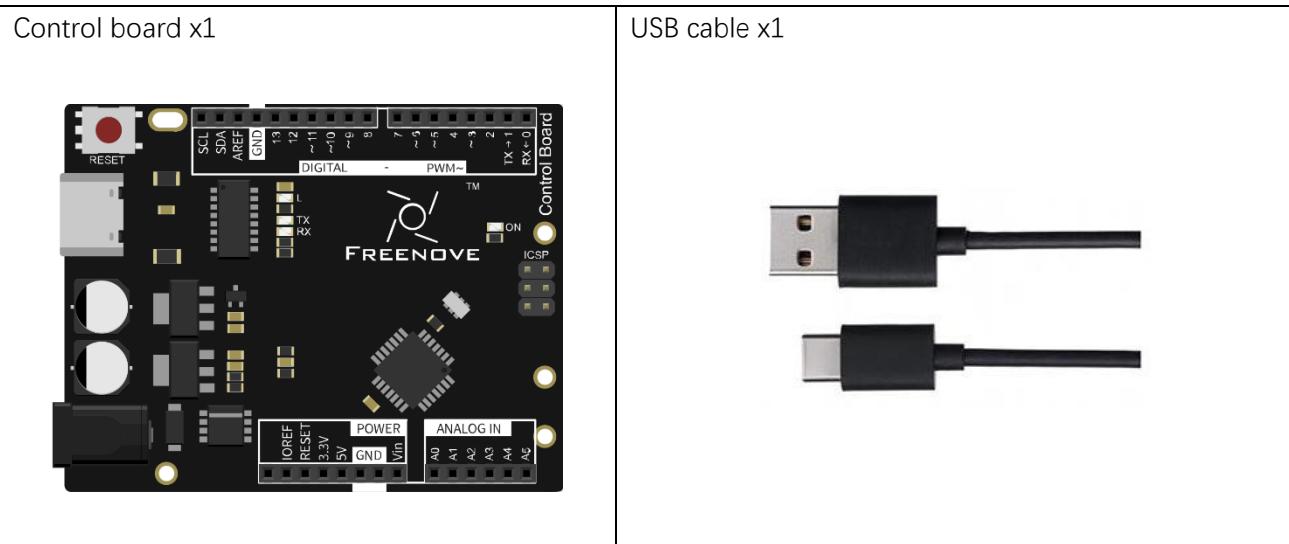
Chapter 6 Serial

Earlier, we have already tried to output signals to LED, and get the input signal of push button switch. Now, we can try serial communication, 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



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:

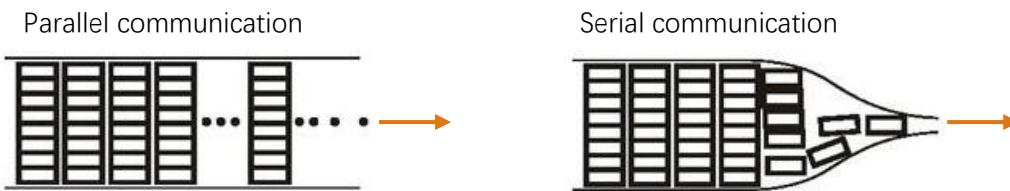
	Remainder	Sequence
2	150 0
2	75 1
2	37 1
2	18 0
2	9 1
2	4 0
2	2 0
2	1 1
	0	

The result is 10010110.

Circuit Knowledge

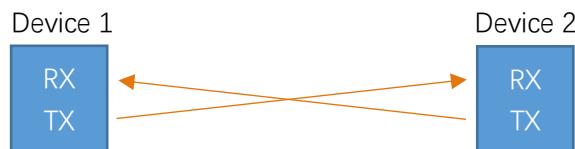
Serial and parallel communication

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:

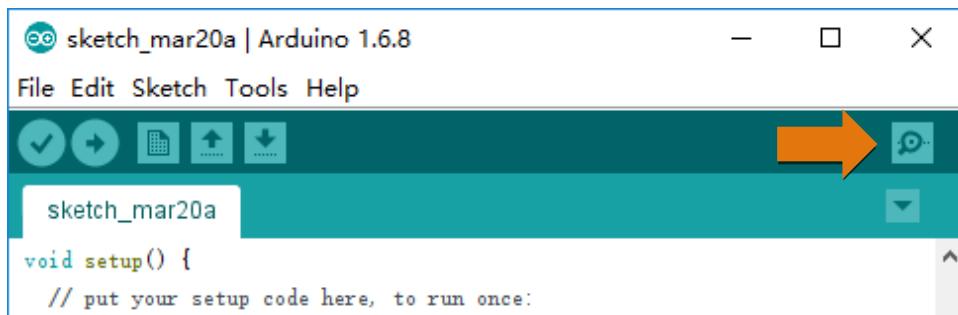


Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

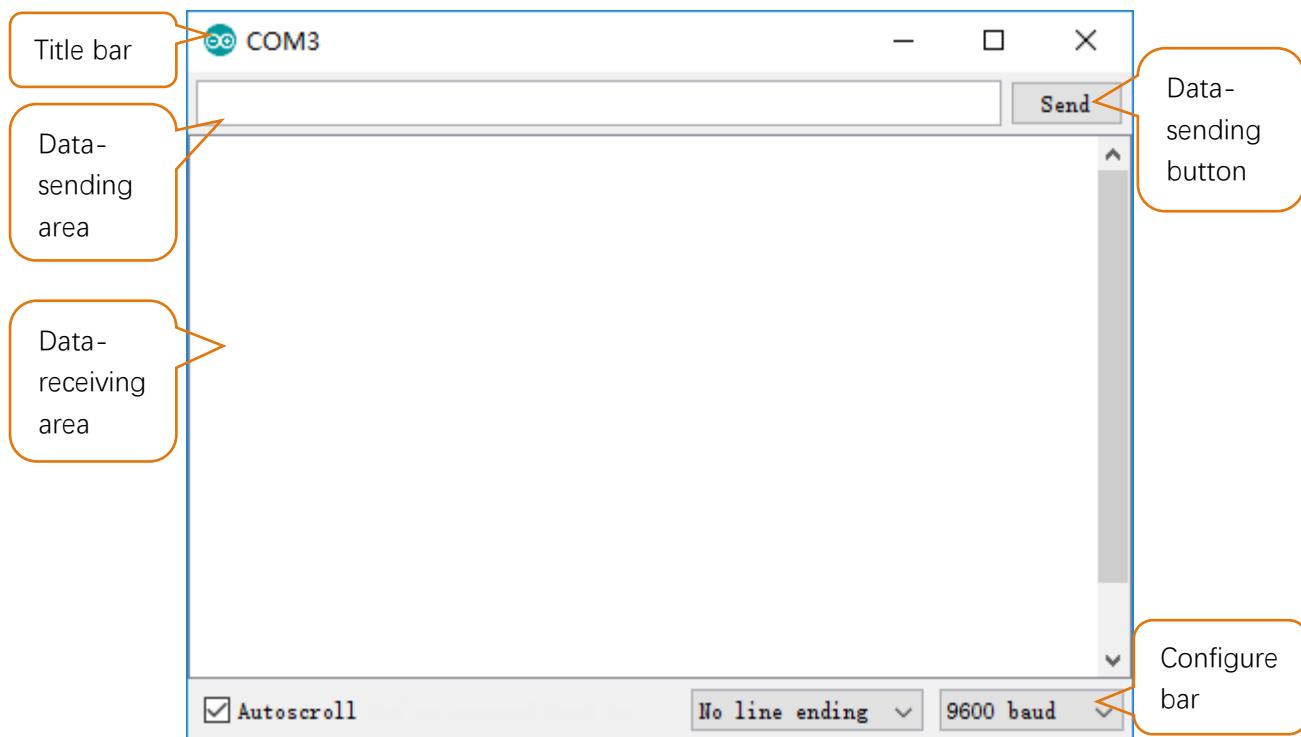
Serial port on Control board

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.

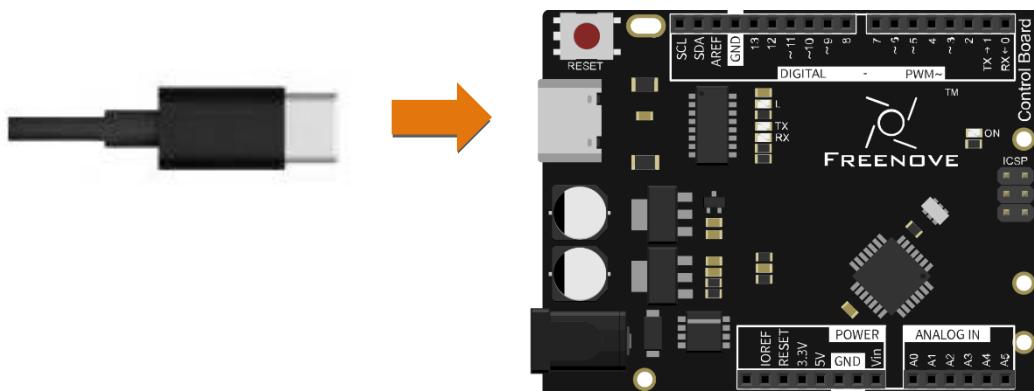


Interface of Serial Monitor window is as follows. If you can't open it, make sure control board had been connected to the computer, and choose the correct serial port in the menu bar "Tools".



Circuit

Connect control board to the computer with USB cable.



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 }
```

```

6
7 void loop() {
8     // print variable counter value to serial
9     Serial.print("counter:");
10    Serial.println(counter);
11    delay(500);
12    counter++;
13 }

```

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:

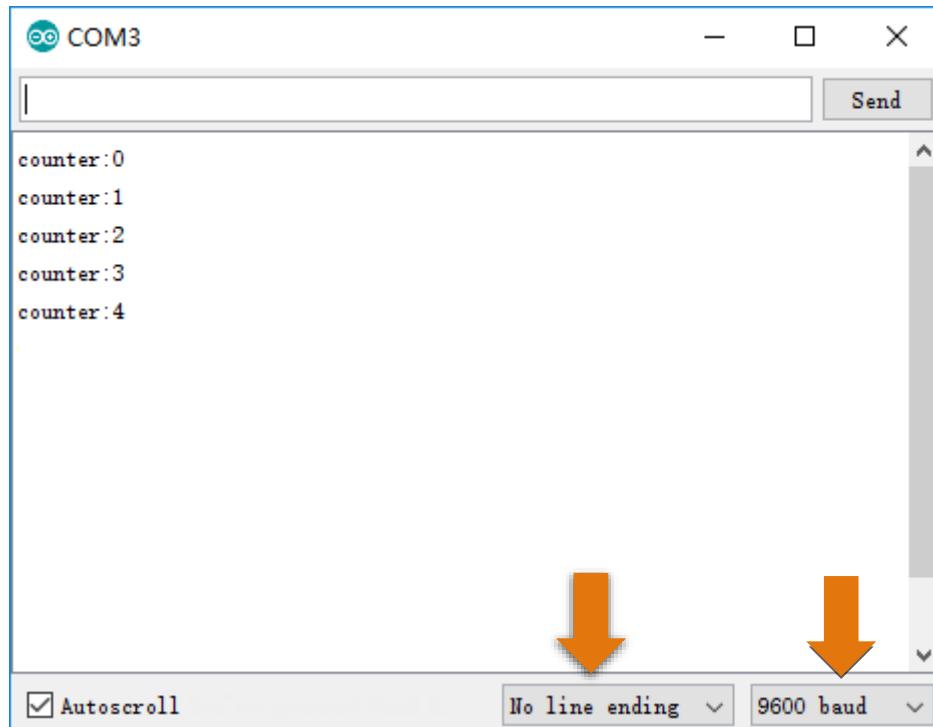
Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

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.



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

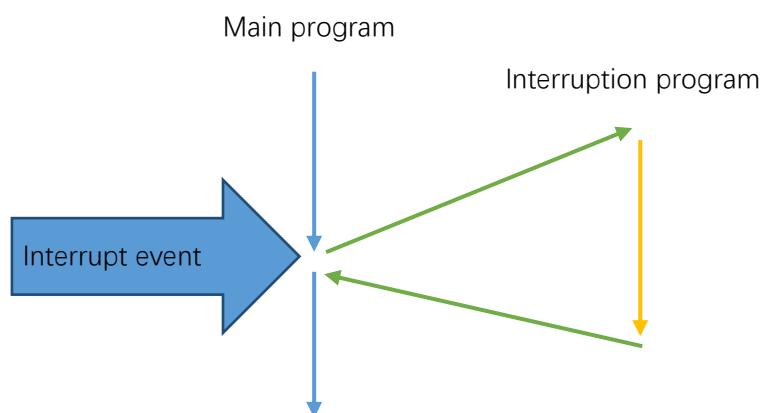
Same with the previous section.

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 }
```

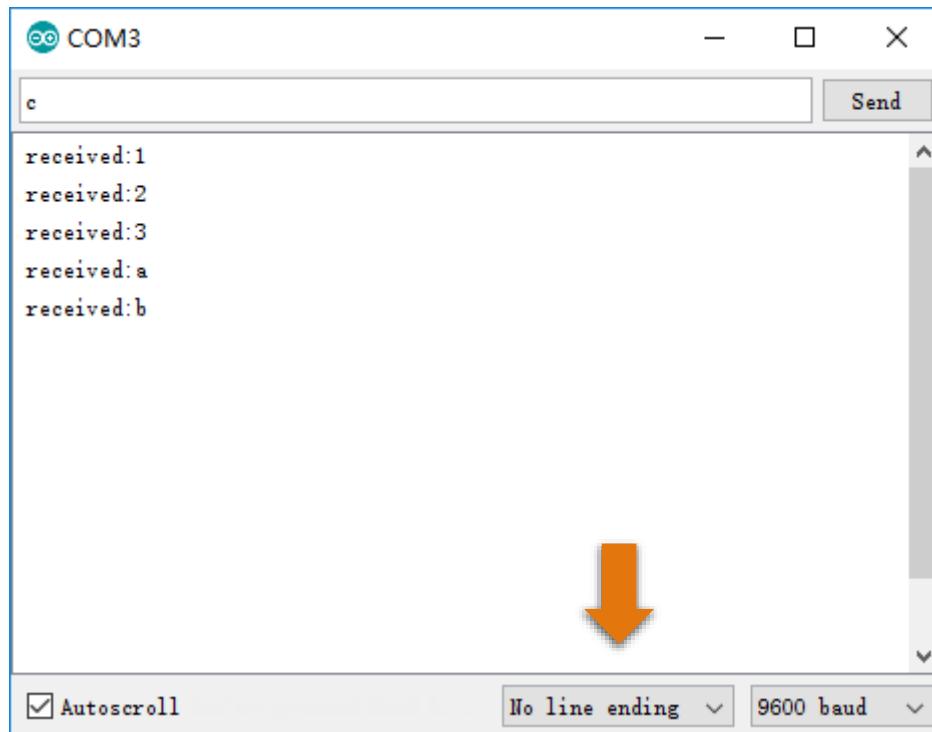
In the setup() function, we initialize the serial port. Then, the loop() function will continuously detect whether there are data to read. If so, it will read the character and send it back.

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.



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 use 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.

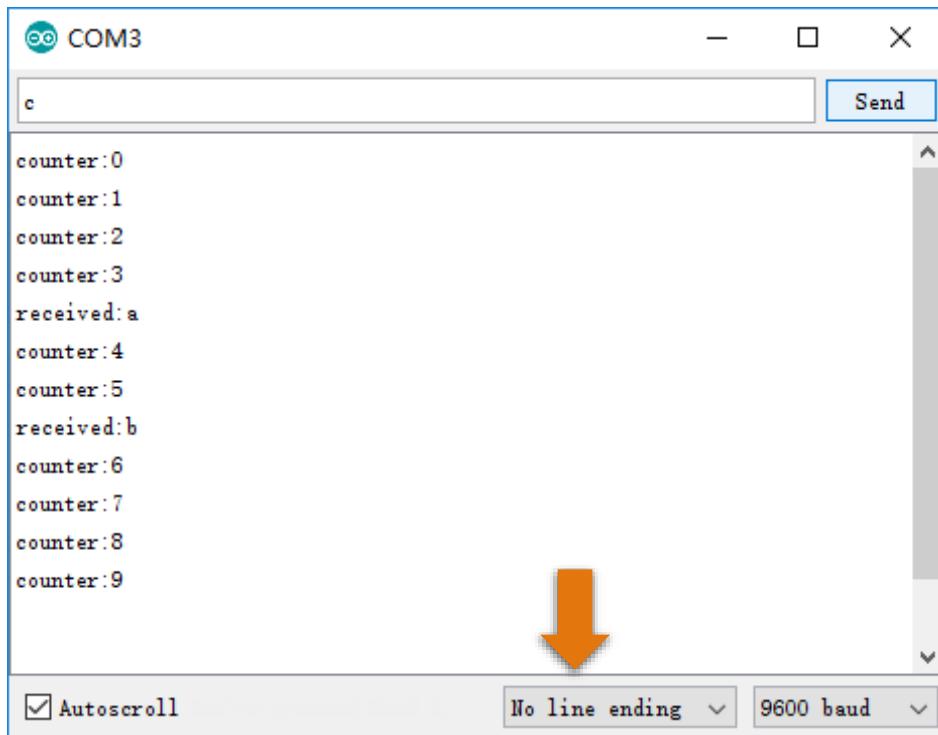
Sketch 6.2.2

When serial port receives data, it can trigger an interrupt event, and enters into the interrupt handling function. Now we use an interrupt to receive information from Serial Monitor window, and send it back. To illustrate that the interrupt does not influence the program's running, we will constantly send changing number in loop() function.

```
1  char inChar;      // define a variable to store character received from serial port
2  int counter = 0;  // define a variable as the data sent to Serial port
3
4  void setup() {
5      Serial.begin(9600);          // initialize serial port and set baud rate to 9600
6  }
7
8  void loop() {
9      // Print value of variable counter to serial
10     Serial.print("counter:");
11     Serial.println(counter);
12     delay(1000);               // wait 1000ms to avoid cycling too fast
13     counter++;                // variable "counter" increases 1
14 }
15
16 void serialEvent() {
17     if (Serial.available()) {    // judge whether the data has been received
18         inChar = Serial.read();  // read one character
19         Serial.print("received:");
20         Serial.println(inChar);  // print the received character
21     }
22 }
```

void serialEvent () function here is the serial port interrupt function. When serial receives data, processor will jump to this function, and return to where the interrupt occurs to proceed after execution. So loop () function's running is not affected.

Verify and upload the code, open the Serial Monitor, then you'll see the number constantly sent from control board. Fill in characters in the sending area, and click the Send button, then you'll see the string returned from control board.

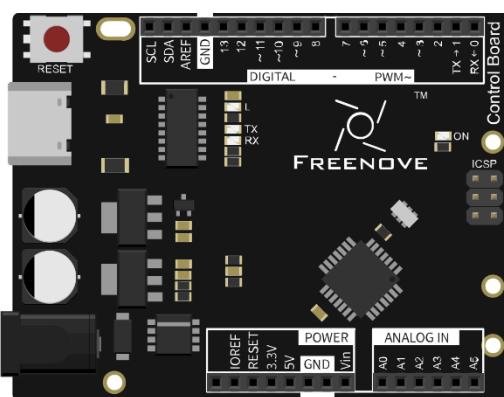


Project 6.3 Application of Serial

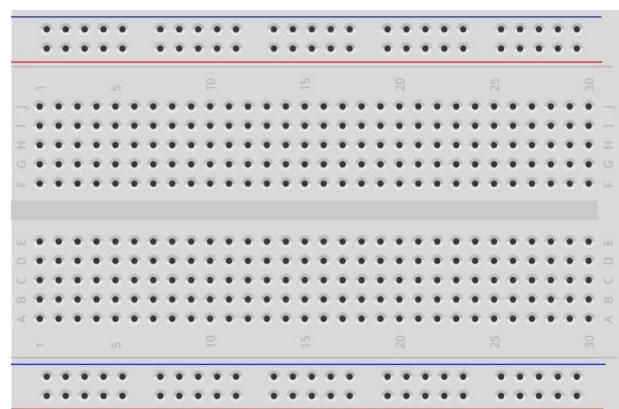
We will use the serial port on control board to control one LED.

Component List

Control board x1



Breadboard x1



USB cable x1



Jumper M/M x2



LED x1

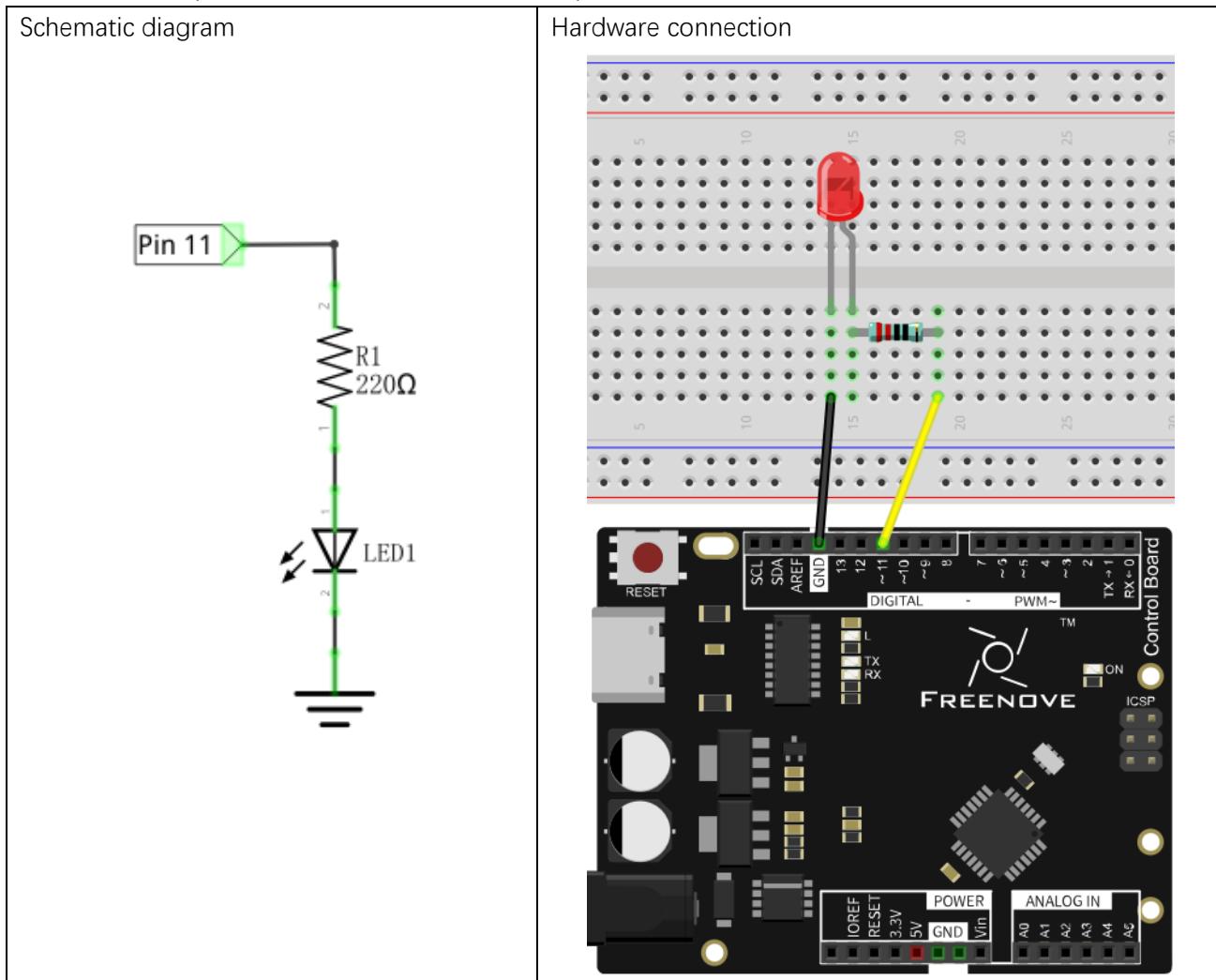


Resistor 220Ω x1



Circuit

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



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 }
```

```
9
10 void loop() {
11     if (Serial.available()) {           // judge whether the data has been received
12         inInt = Serial.parseInt();      // read an integer
13         Serial.print("received:");     // print the string " received:"
14         Serial.println(inInt);        // print the received character
15         // convert the received integer into PWM duty cycle of ledPin port
16         analogWrite(ledPin, constrain(inInt, 0, 255));
17     }
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.

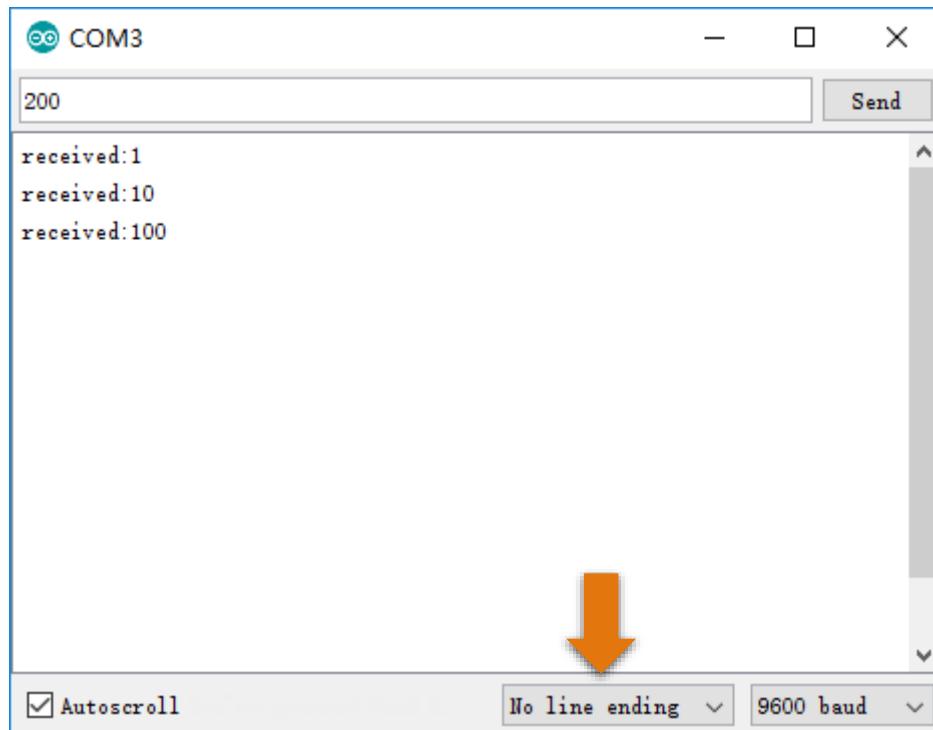
Serial Class

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.



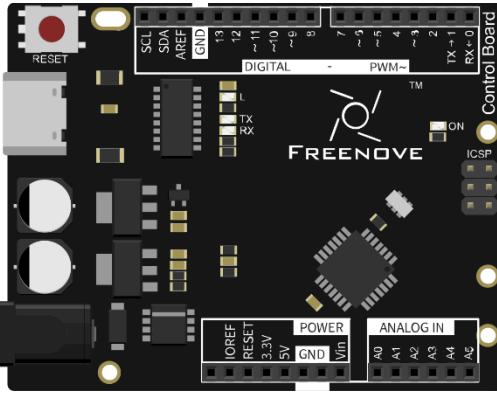
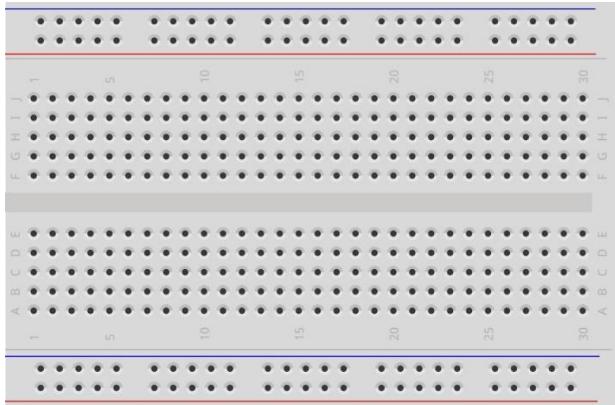
Chapter 7 ADC

Earlier, we have learned the digital ports of control board, and tried the output and input signals. Now, let us learn how to use analog ports.

Project 7.1 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

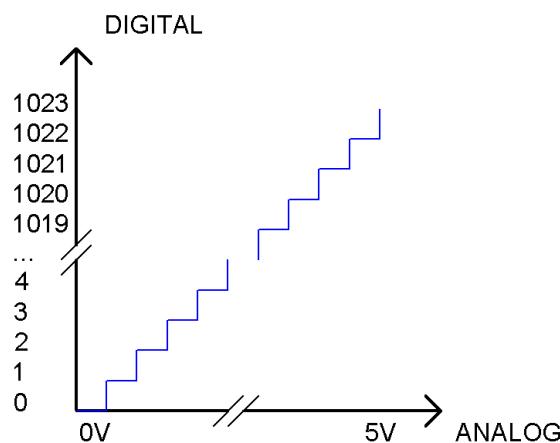
Control board x1	Breadboard x1
	
USB cable x1	Rotary potentiometer x1
	
Jumper M/M x3	
	

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 range of our ADC module is 10 bits, that means the resolution is $2^{10}=1024$, so that its range (at 5V) will be divided equally into 1024 parts.

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;

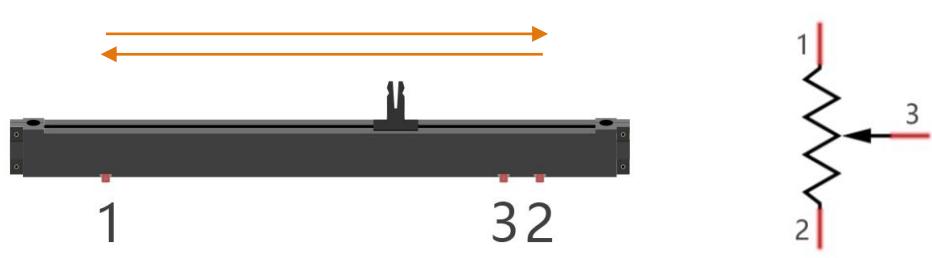
Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;

The following analog signal will be divided accordingly.

Component Knowledge

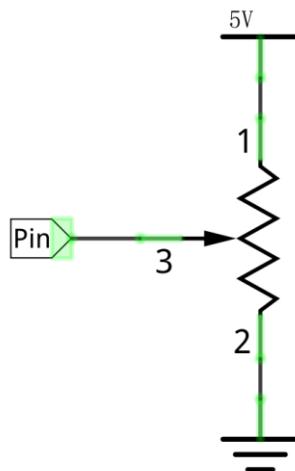
Potentiometer

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.



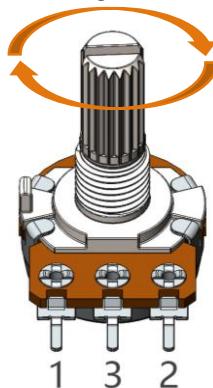
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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



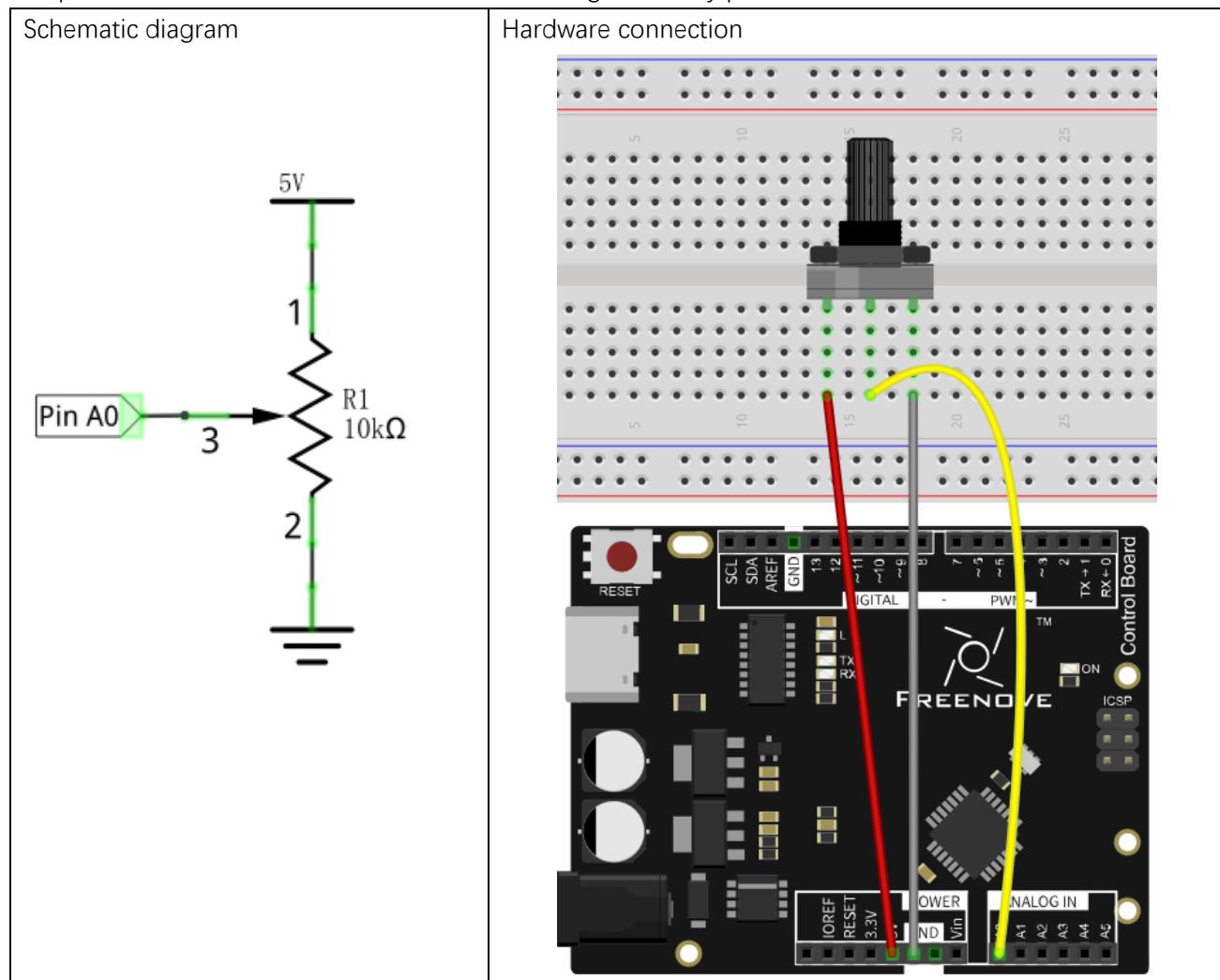
Rotary potentiometer

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.



Sketch

Sketch 7.1.1

Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino IDE through serial port.

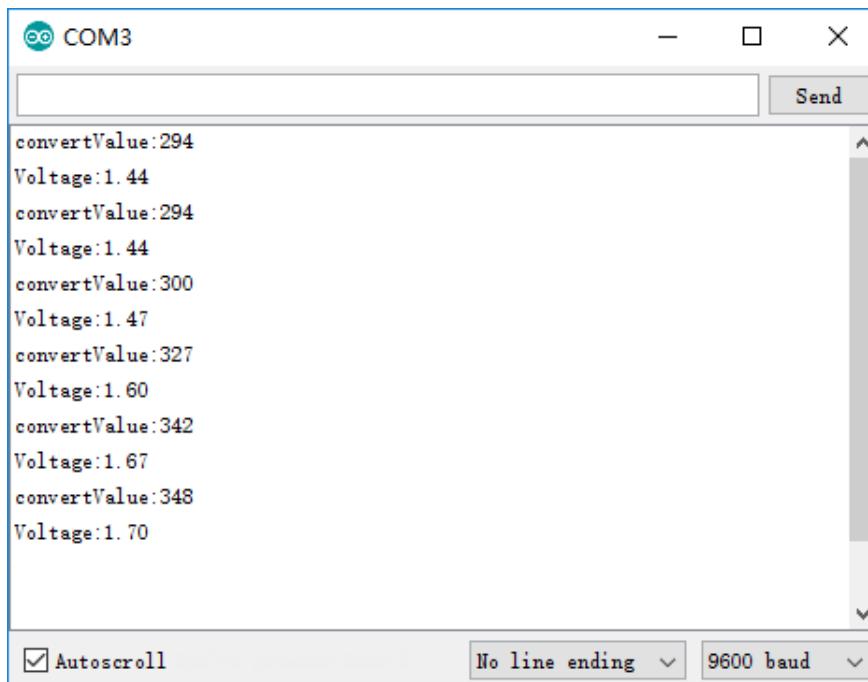
```

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 }
7
8 void loop() {
9     adcValue = analogRead(A0);          // Convert analog of pin A0 to digital
10    voltage = adcValue * (5.0 / 1023.0); // Calculate voltage according to digital
11    // Send the result to computer through serial
12    Serial.print("convertValue:");
13    Serial.println(adcValue);
14    Serial.print("Voltage:");
15    Serial.println(voltage);
16    delay(500);
17 }
```

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.

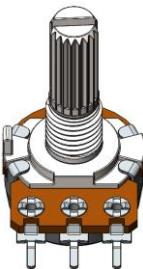
Turn the rotary potentiometer shaft, and you can see the voltage change.



Project 7.2 Control LED by Potentiometer

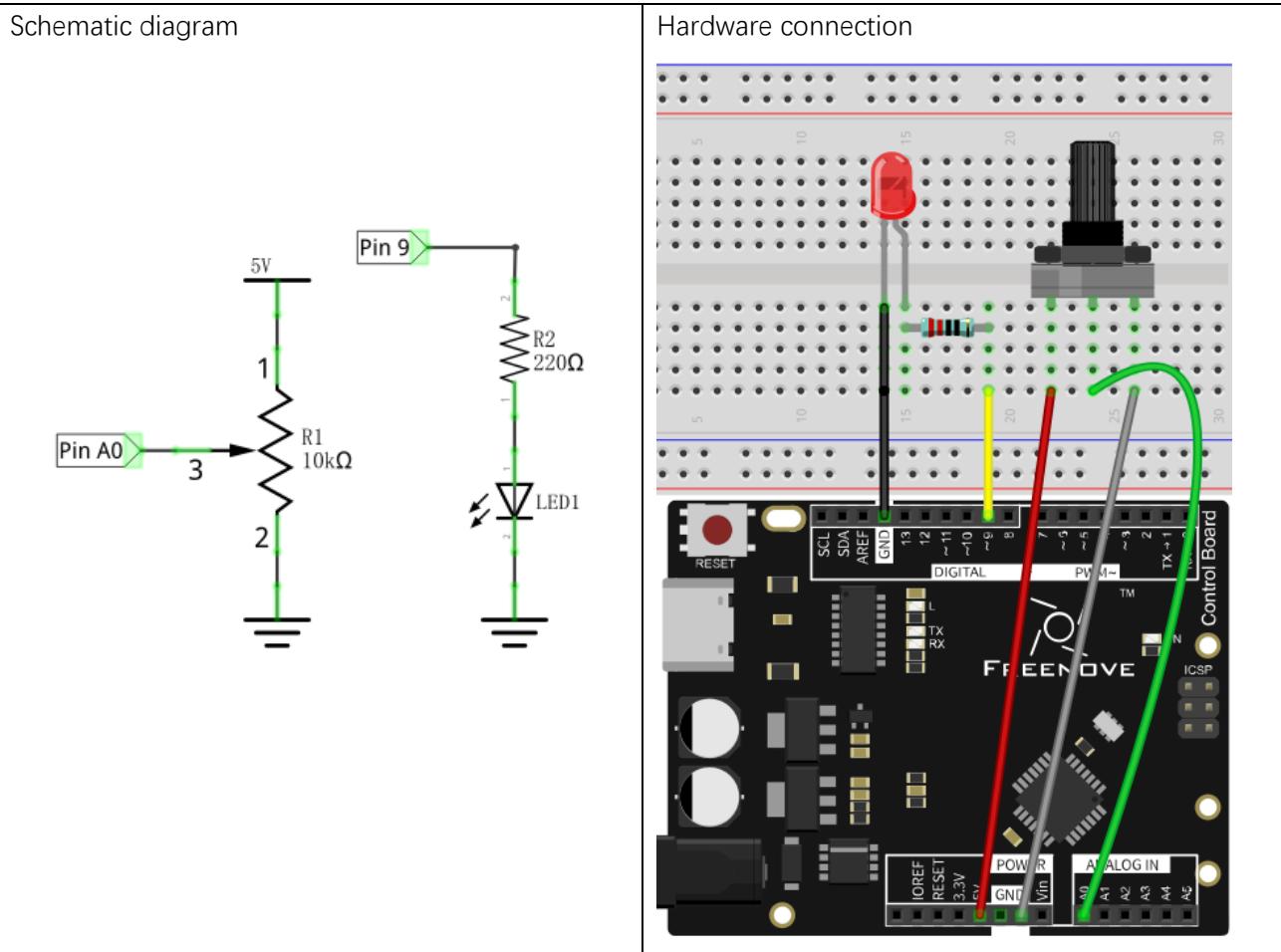
In the previous section, we have finished reading ADC value and converting it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

Component List

Control board x1	Breadboard x1	
		
USB cable x1	Rotary potentiometer x1	LED x1
		
Jumper M/M x5		Resistor 220Ω x1
		

Circuit

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



Sketch

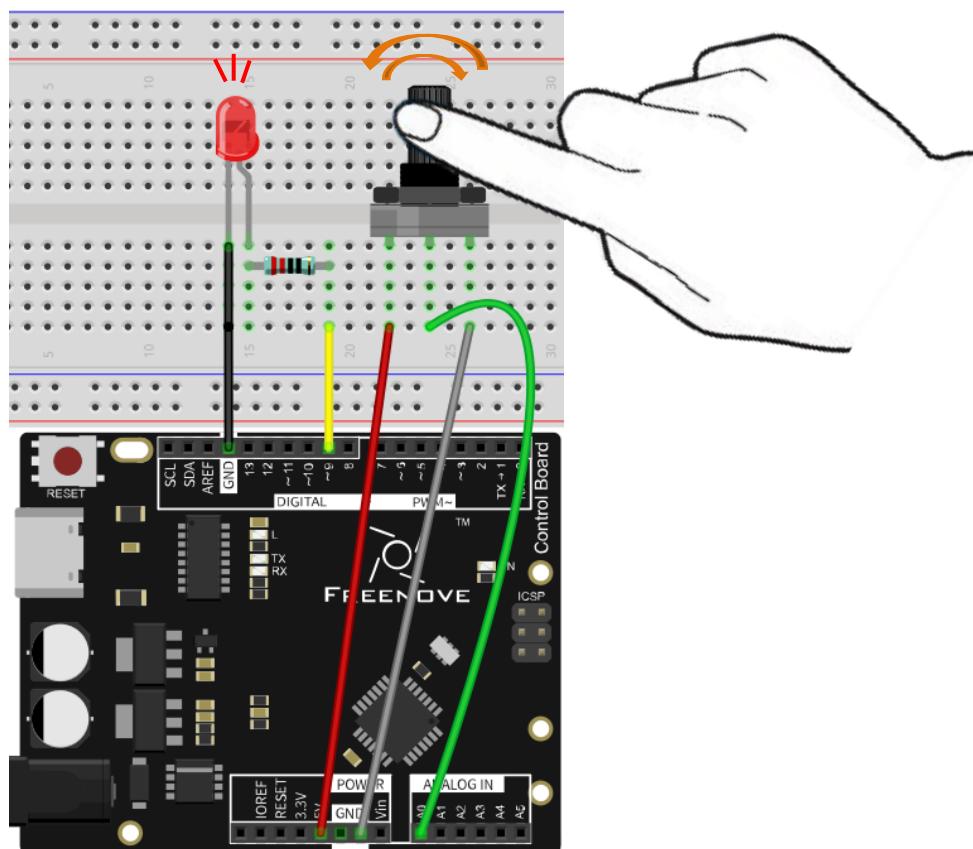
Sketch 7.2.1

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 pin 9 on control board 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.

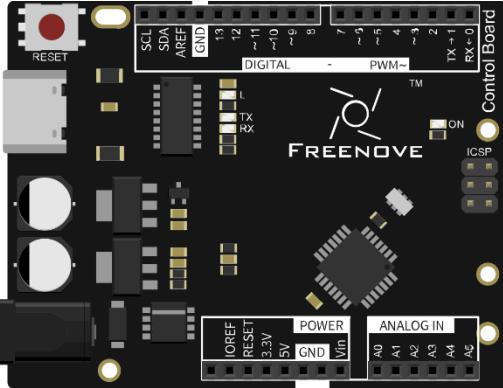
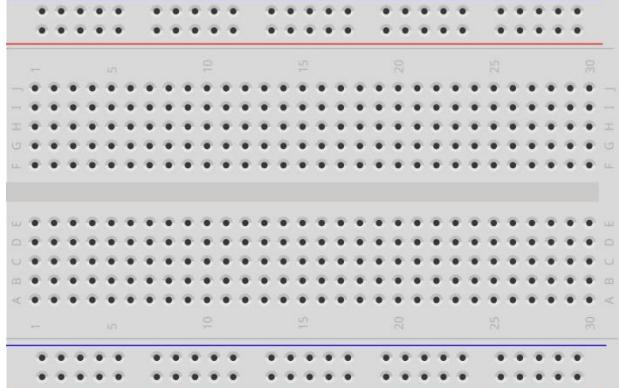
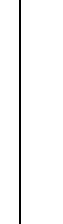
Verify and upload the code, turn the rotary potentiometer shaft, you will see the LED brightness change.



Project 7.3 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.

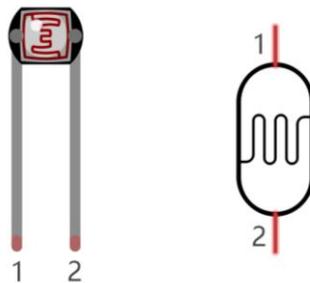
Component List

Control board x1	Breadboard x1			
				
USB cable x1	Photoresistor x1	LED x1	Resistor 10kΩ x1	Resistor 220Ω x1
				
Jumper M/M x5				
				

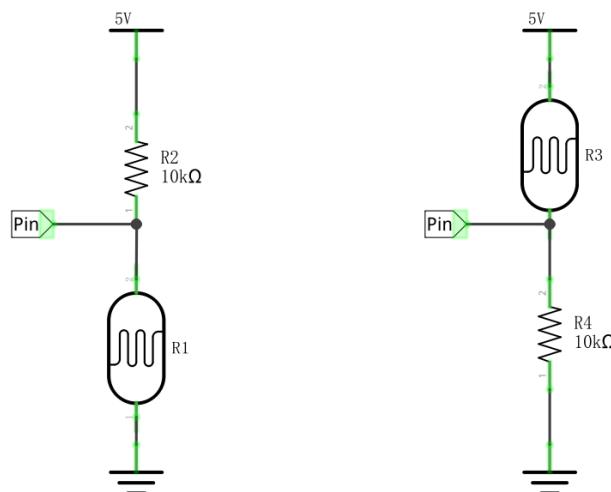
Component Knowledge

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.



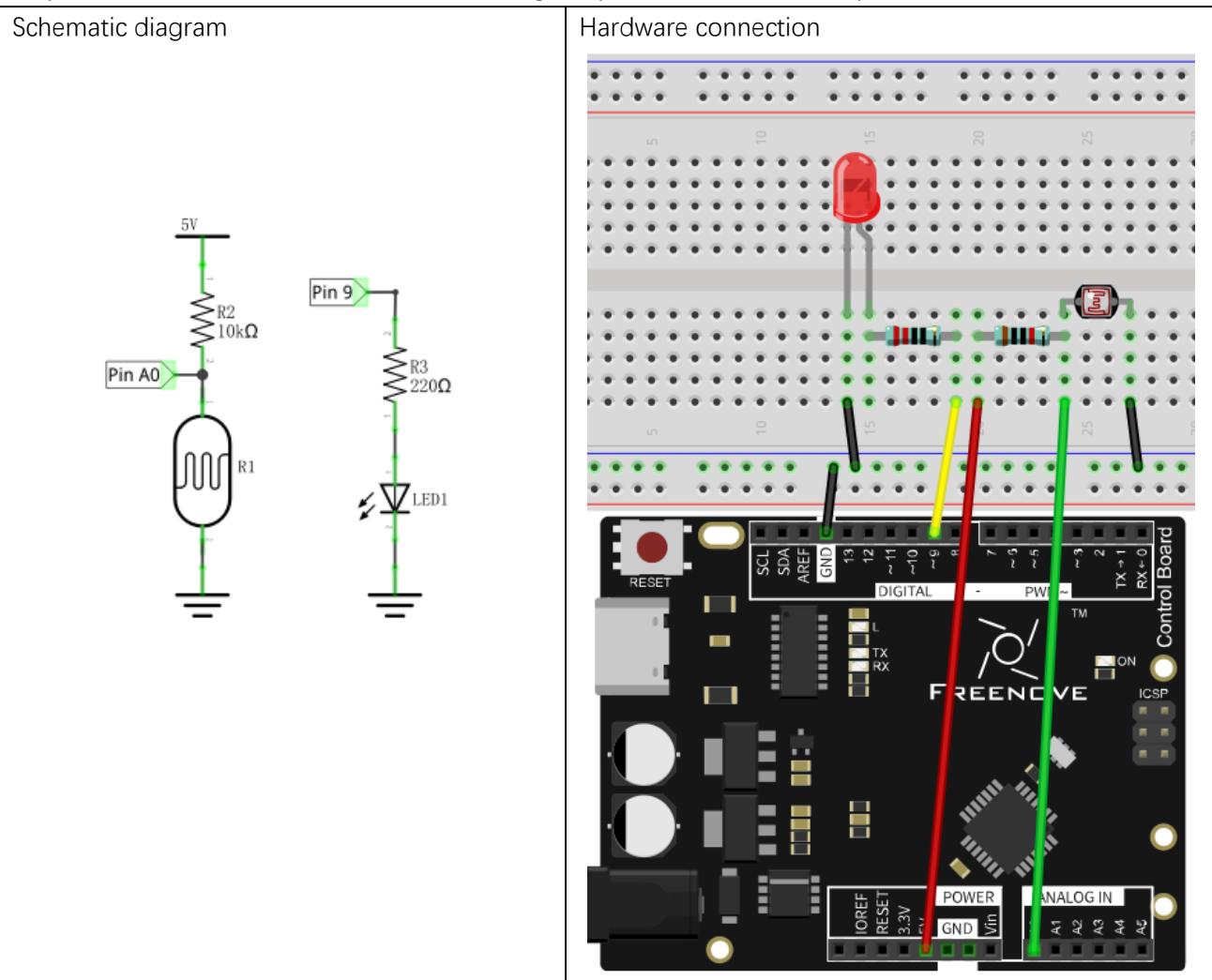
The circuit below is often used to detect the change of photoresistor's resistance value:



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.



Sketch

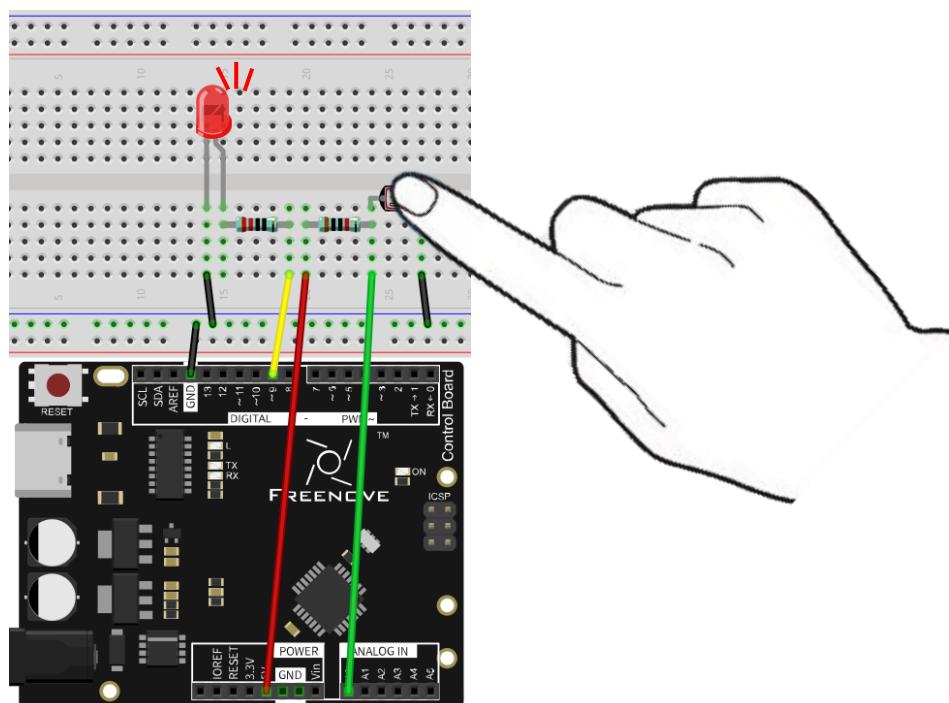
Sketch 7.3.1

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 pin A0, and save
10    // Map analog to the 0~255 range, and works as ledPin duty cycle setting
11    analogWrite(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.



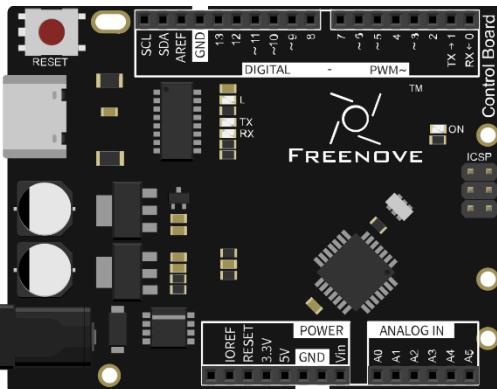
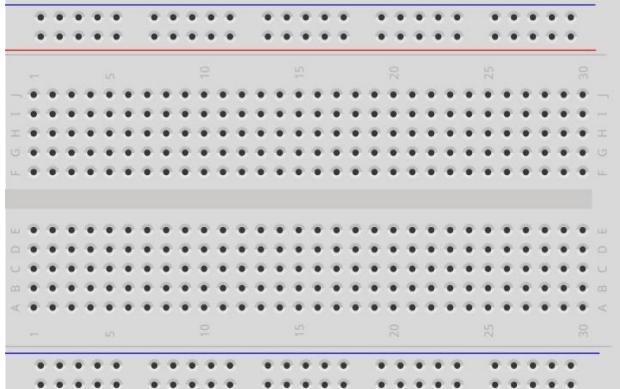
Chapter 8 RGB LED

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 8.1 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.

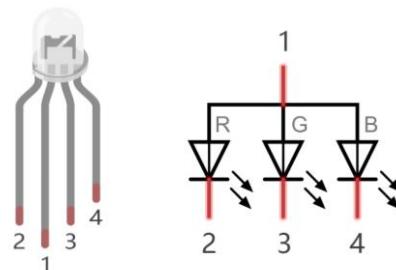
Component List

Control board x1	Breadboard x1	
		
USB cable x1	Rotary potentiometer x3	
		
Jumper M/M x15	RGB LED x1	Resistor 220Ω x3
		

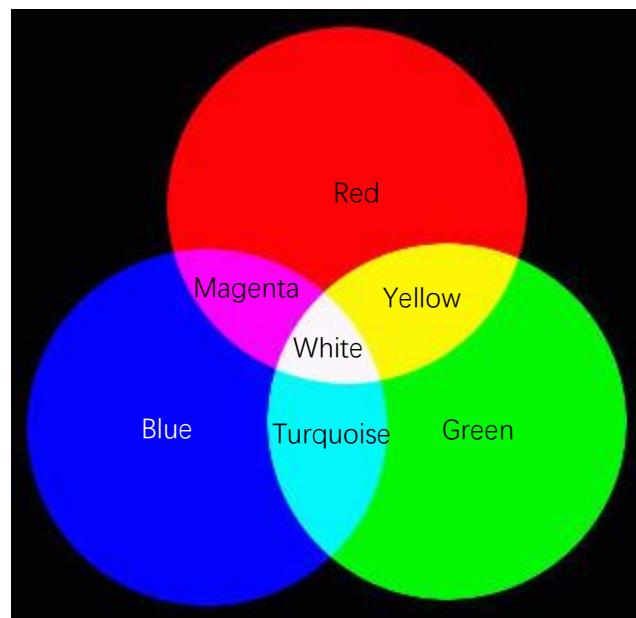
Component Knowledge

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.

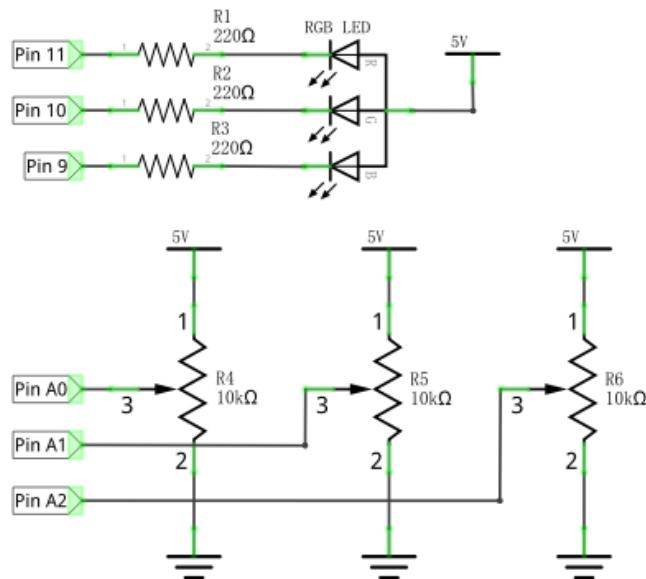


We know from the previous section that, control board controls LED to emit a total of 256(0-255) different brightness via PWM. So, through different combinations of RGB light brightness, we can create $256^3=16777216$ (16Million) colors.

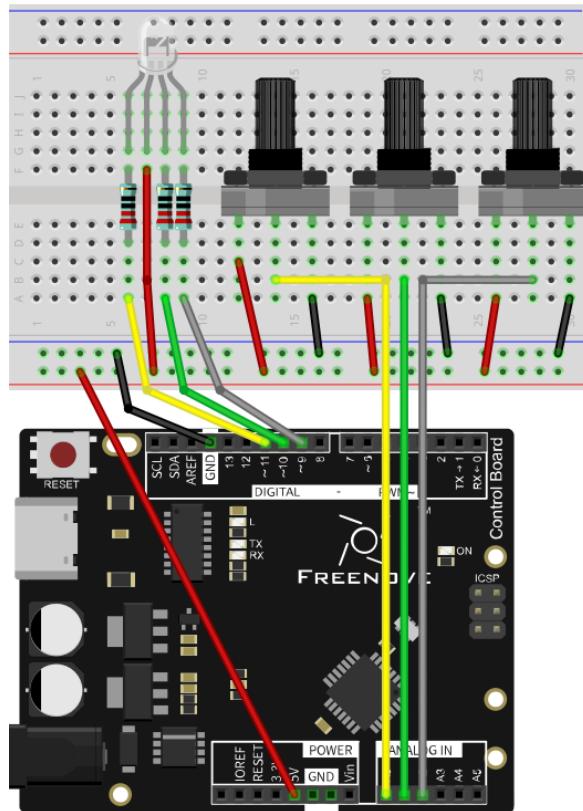
Circuit

Use pin A0, A1, A2 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



Sketch

Sketch 8.1.1

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 A0 port into digital, and work as PWM duty cycle of ledPinR port  
16     adcValue = analogRead(A0);  
17     analogWrite(ledPinR, map(adcValue, 0, 1023, 0, 255));  
18     // Convert analog of A1 port into digital, and work as PWM duty cycle of ledPinG port  
19     adcValue = analogRead(A1);  
20     analogWrite(ledPinG, map(adcValue, 0, 1023, 0, 255));  
21     // Convert analog of A2 port into digital, and work as PWM duty cycle of ledPinB port  
22     adcValue = analogRead(A2);  
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.

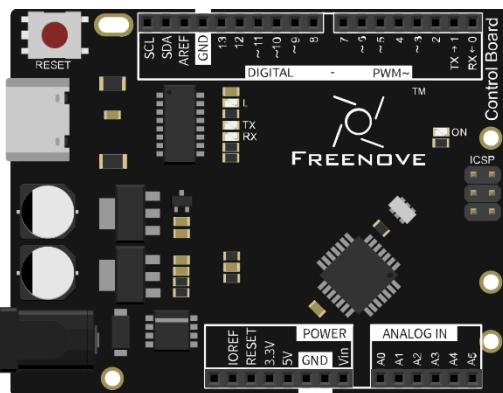
Verify and upload the code, rotate the three rotary potentiometer shaft, and you can see the LED light change in its color and brightness.

Project 8.2 Multicolored LED

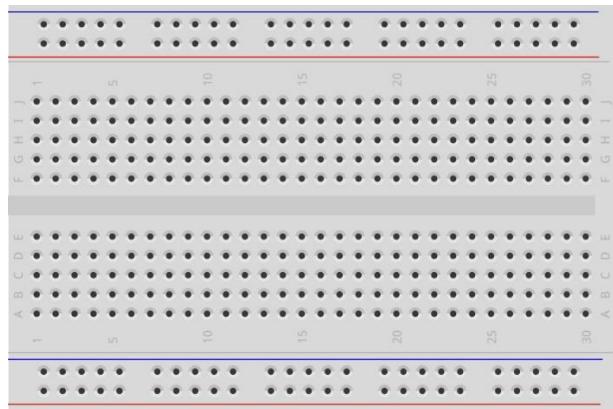
In the previous section, we have finished controlling the RGB LED to emit light with different color and brightness through three potentiometers. Now, we will try to make RGB LED emit multicolored lights automatically.

Component List

Control board x1



Breadboard x1



USB cable x1



Jumper M/M x4



RGB LED x1



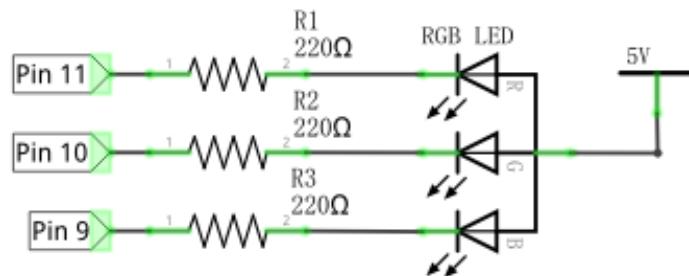
Resistor 220Ω x3



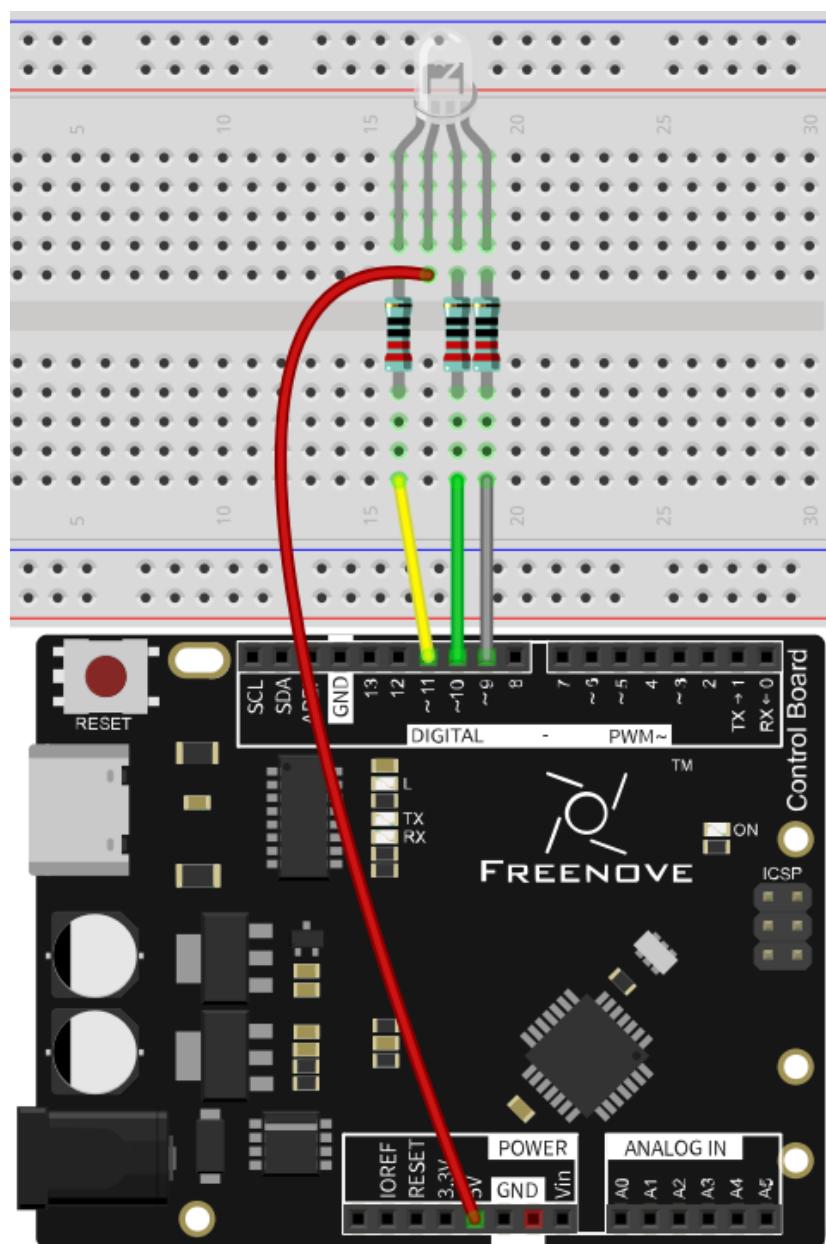
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 8.2.1

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

```
1 // 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     // Generate three random numbers between 0-255 as the output PWM duty cycle of ledPin  
15     rgbLedDisplay(random(256), random(256), random(256));  
16     delay(500);  
17 }  
18  
19 void rgbLedDisplay(int red, int green, int blue) {  
20     // Set three ledPin to output the PWM duty cycle  
21     analogWrite(ledPinR, constrain(red, 0, 255));  
22     analogWrite(ledPinG, constrain(green, 0, 255));  
23     analogWrite(ledPinB, constrain(blue, 0, 255));  
24 }
```

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.

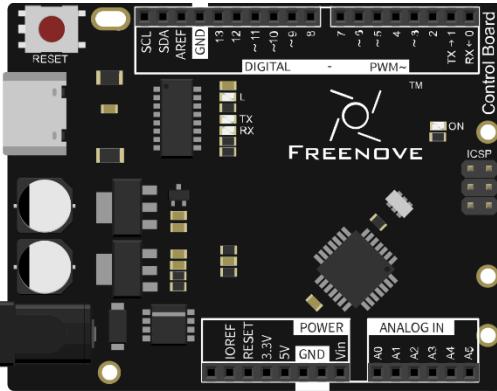
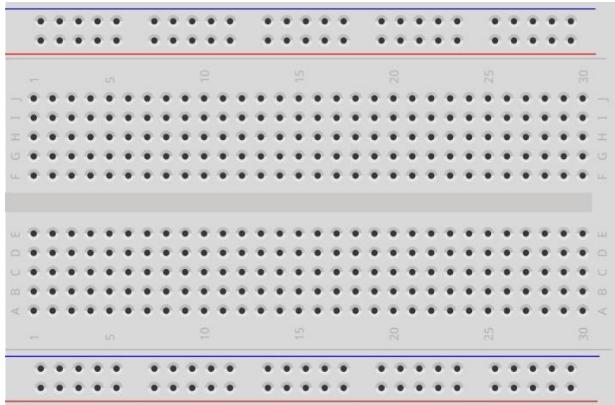
Chapter 9 Buzzer

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 9.1 Active Buzzer

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

Component List

Control board x1	Breadboard 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,

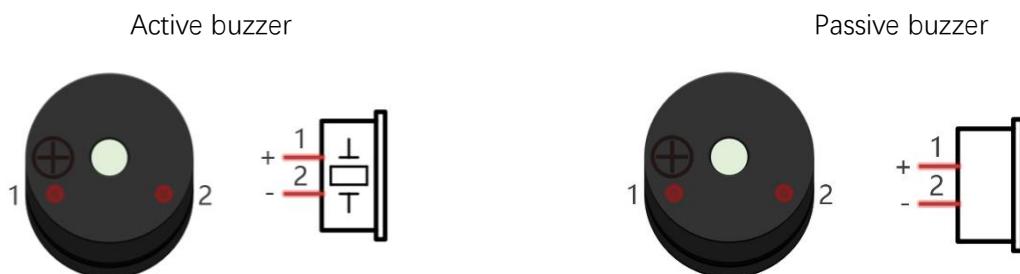


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.



(A white label is attached to the active buzzer)

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).



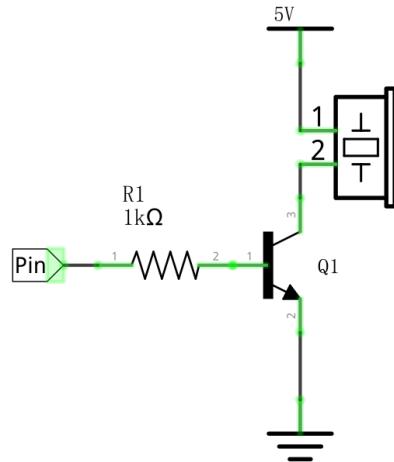
Active buzzer bottom



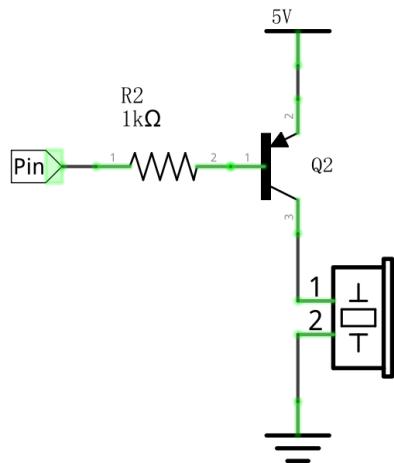
Passive buzzer bottom

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).

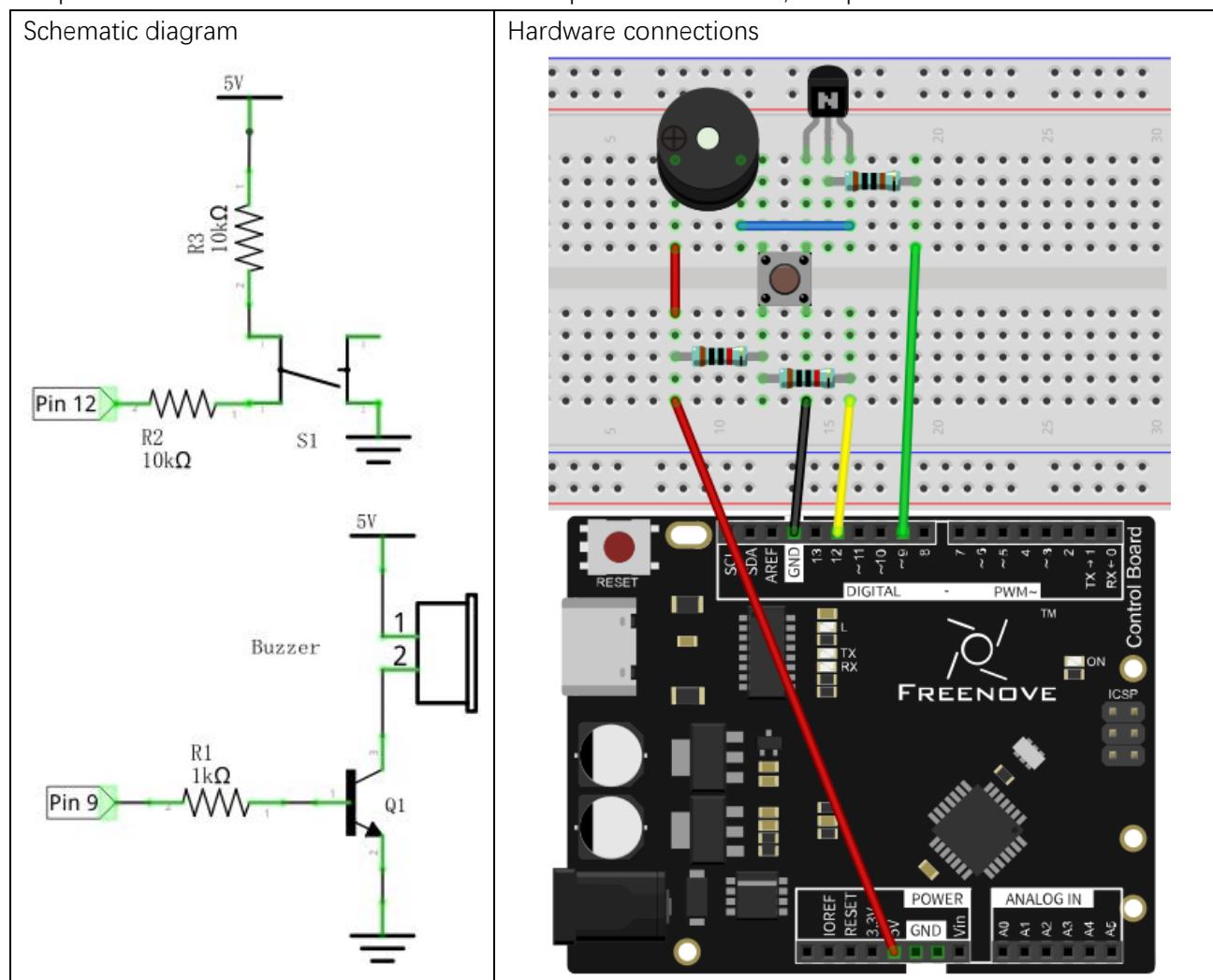


When we use a PNP transistor to drive a buzzer, we often use the following method. If control board pin outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If control board pin outputs high level, no current flows through R1, the transistor 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.



Sketch

Sketch 9.1.1

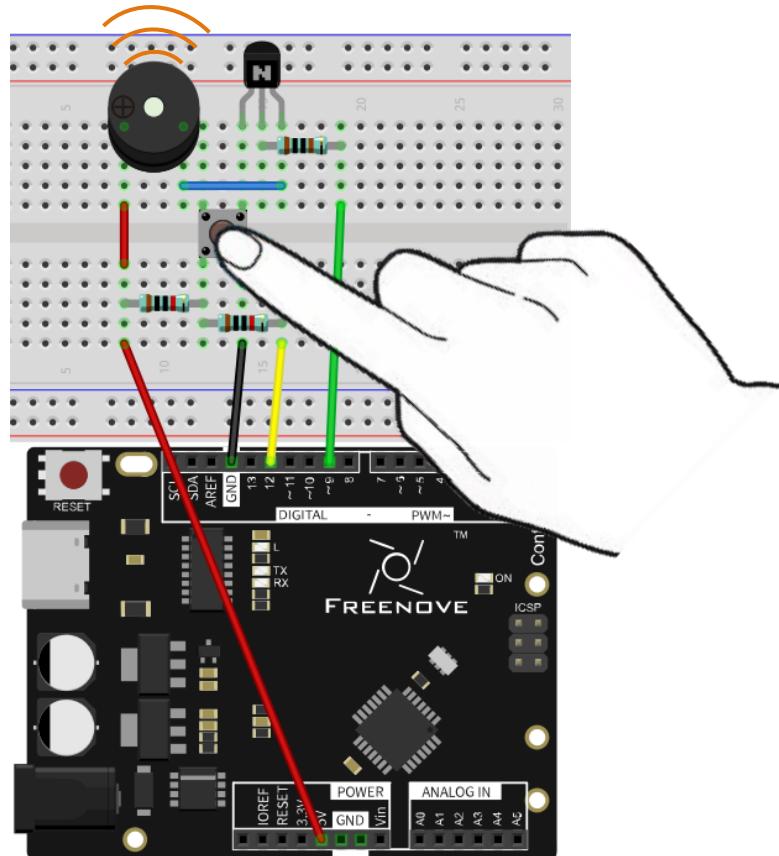
Now, write code to detect the state of push button, and drive active buzzer to make a sound when it is pressed.

```

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.

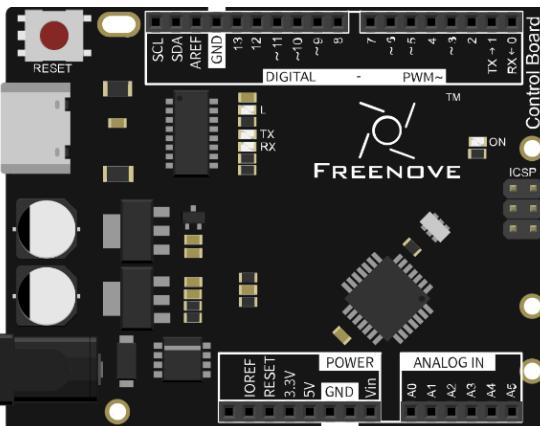
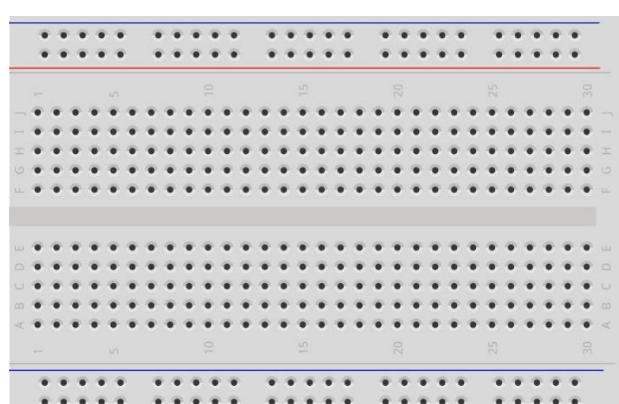
Verify and upload the code, press the push button, and the active buzzer will make a sound.



Project 9.2 Passive Buzzer

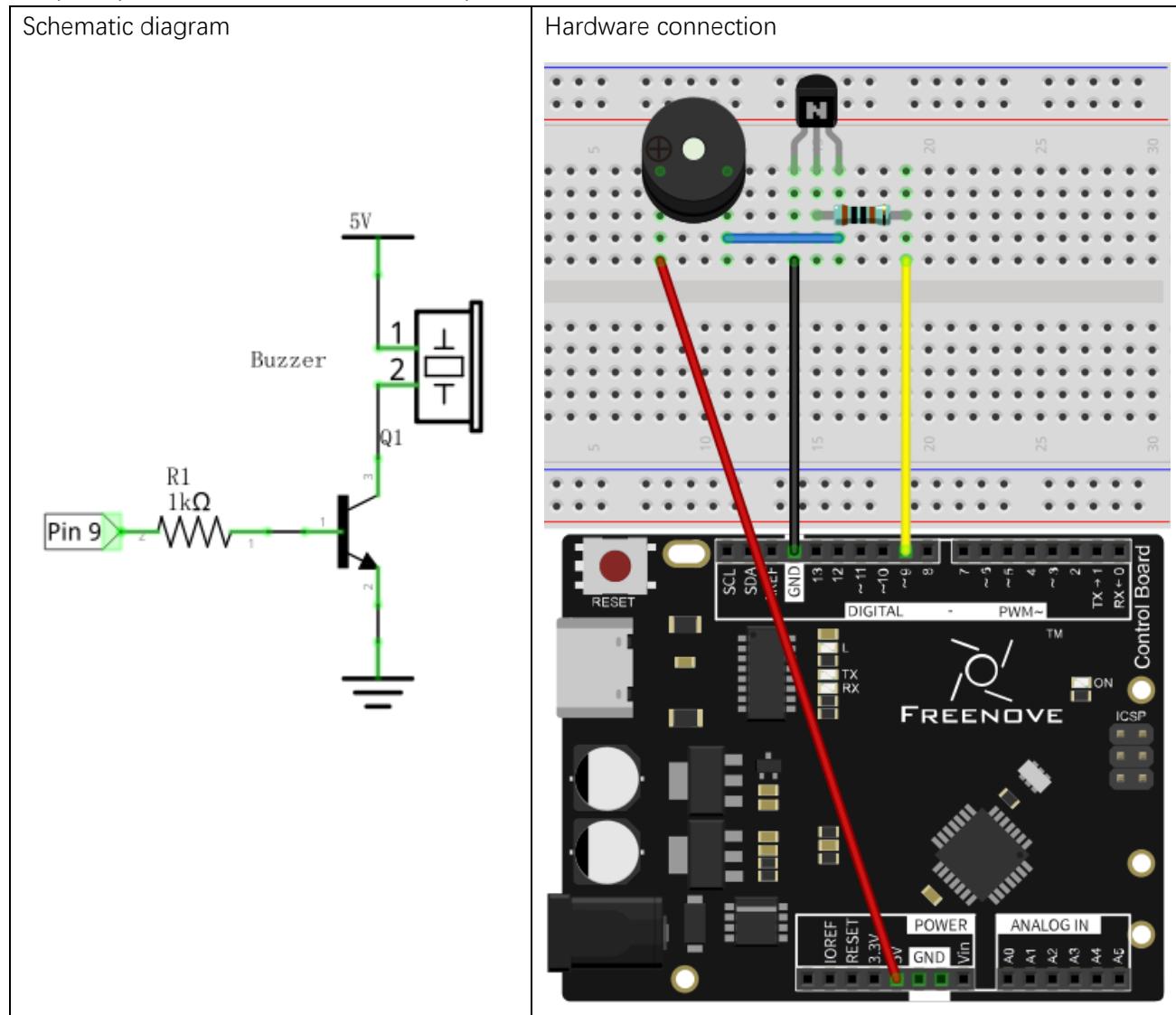
In the previous section, we have finished using transistor to drive an active buzzer to beep. Now, we will try to use a passive buzzer to make a sound with different frequency.

Component List

Control board x1	Breadboard x1
 A black Freenove Control Board (Arduino Uno compatible) with various pins, connectors, and components. It features a red LED, two white push buttons, and several jumper wires. Labels include SCL, SDA, AREF, GND, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, TX, RX, PWM, IOREF, 3.3V, POWER, GND, Vin, A0, A1, A2, A3, A4, A5, and ANALOG IN.	 A standard breadboard with four vertical columns of 30 rows each, labeled A through J along the bottom and 1 through 30 along the left side. Red and blue rails run horizontally across the top and bottom respectively.
USB cable x1	NPN transistor x1
 Two black USB cables, one with a standard A-type connector and one with a micro-B type.	 A small black NPN transistor component with three pins: base, collector, and emitter.
Jumper M/M x4	Passive buzzer x1
 A single black jumper cable with two male pins.	 A black circular passive buzzer component with three terminals: a central green terminal and two red terminals.
	Resistor 1kΩ x1
	 A cylindrical resistor component with a brown band indicating 1k ohms.

Circuit

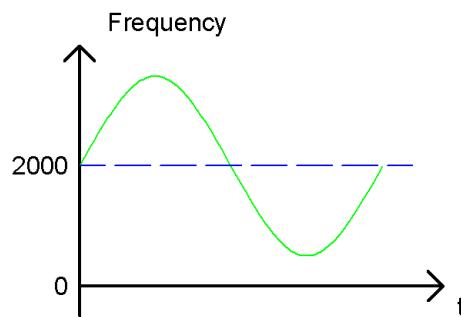
Use pin 9 port of control board to drive a passive buzzer.



Sketch

Sketch 9.2.1

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.

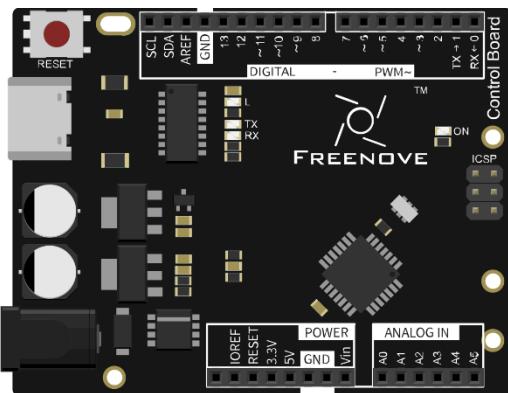
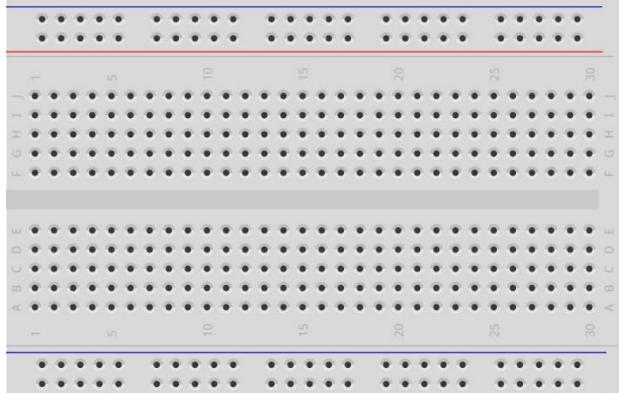
Chapter 10 Temperature Sensor

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

Project 10.1 Detect the Temperature

We will use a thermistor to detect the ambient temperature.

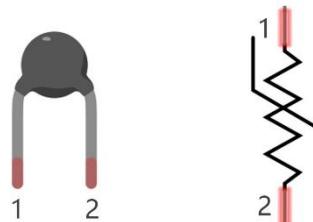
Component List

Control board x1	Breadboard 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;

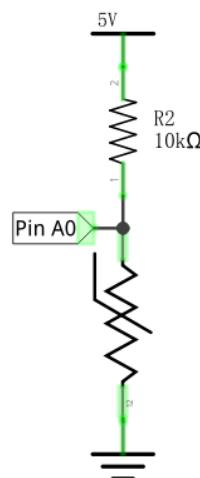
$\exp[n]$ is nth power of e;

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: $B=3950$, $R=10k$, $T_1=25$.

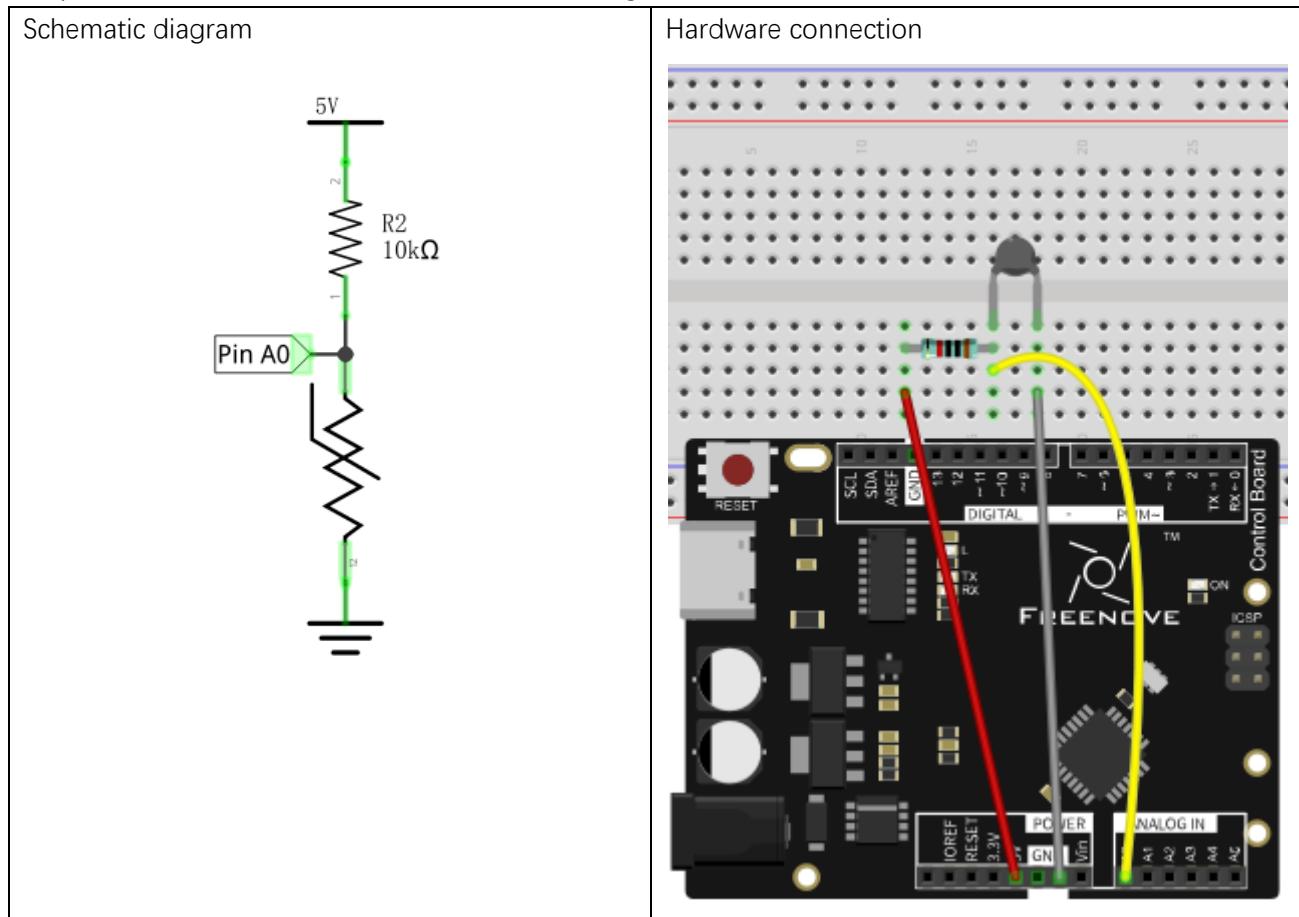
The circuit connection method of the thermistor is similar to photoresistor, as the following:



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.



Sketch

Sketch 12.1.1

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

```

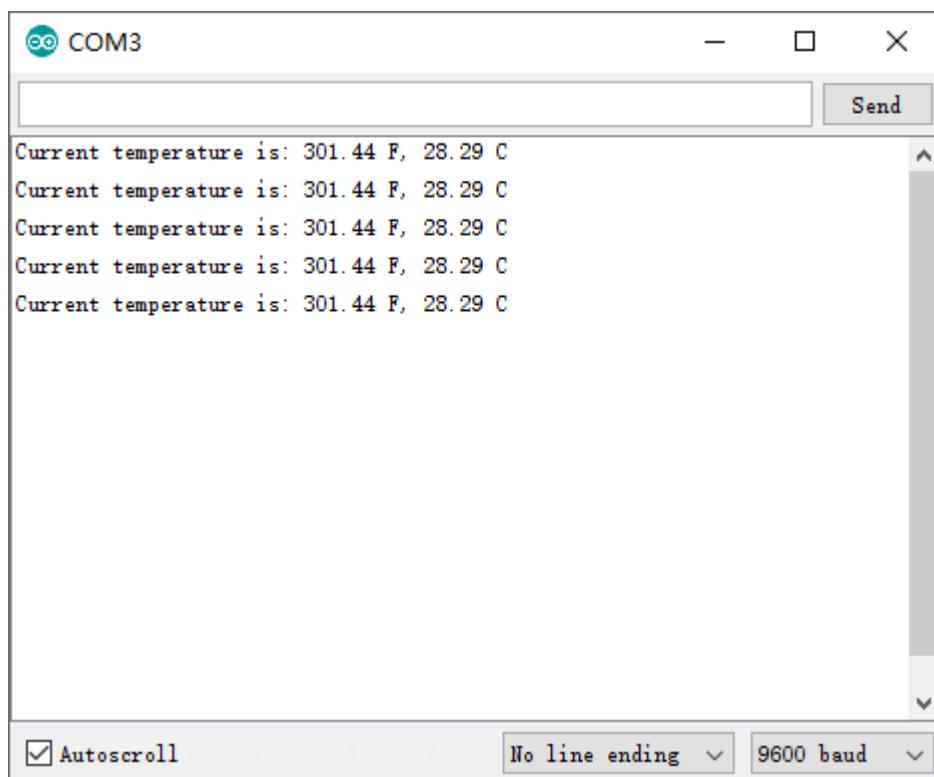
1 void setup() {
2     Serial.begin(9600);           // Initialize the serial port, set the baud rate
3     // into 9600
4 }
5 void loop() {
6     // Convert analog value of A0 port into digital value
7     int adcVal = analogRead(A0);
8     // Calculate voltage
9     float v = adcVal * 5.0 / 1024;
10    // 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.



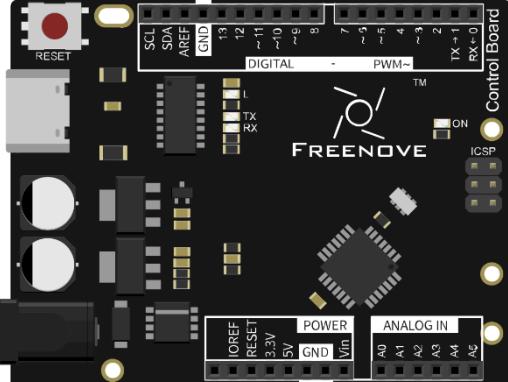
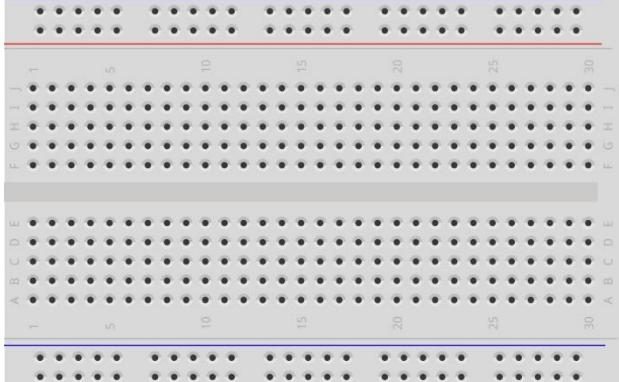
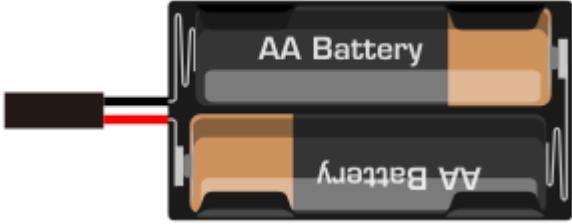
Chapter 11 Motor

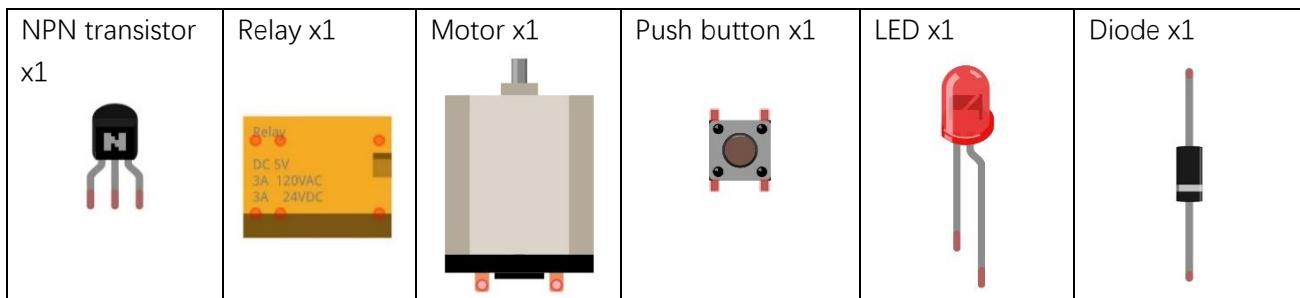
Earlier, we have done a series of interesting projects with control board and basic electronic components. Now, let us study some movable electronic modules. In this chapter, we will learn to control a motor.

Project 11.1 Control Motor by Relay

First, use relay to control a Motor.

Component List

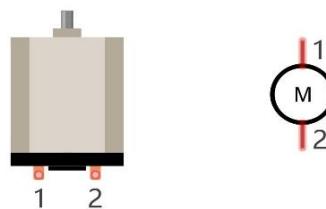
Control board x1	Breadboard x1		
			
USB cable x1	Jumper M/M x8		
			
AA Battery holder x1 (Need AA battery x2)	Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1
			



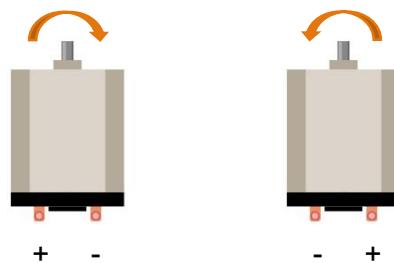
Component Knowledge

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

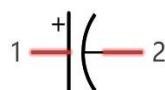
The unit of capacitance(C) is farad (F). $1F = 1000000\mu F$, $1\mu F = 1000000pF$.

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:

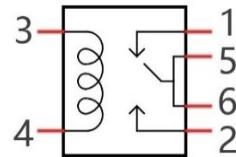


When the motor rotates, it will generate noise. As the contact of coil connects and disconnects the electrode constantly, it will cause the supply voltage unstable. Thus, a small capacitor is often connected to motor to reduce the impact on power supply from motor.

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.

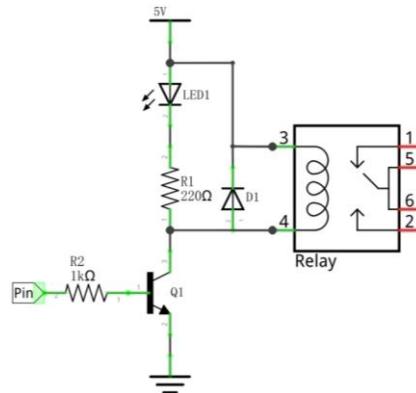
Inductor

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.



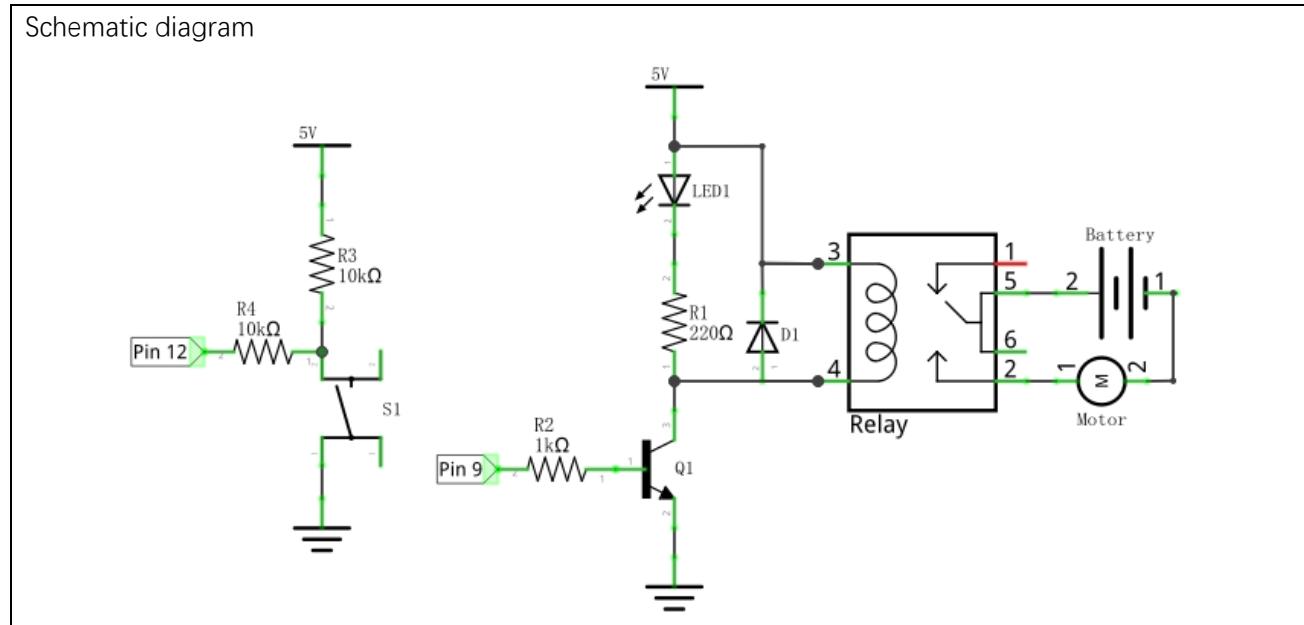
The circuit for a relay is as follows: The coil of relay can be equivalent to an Inductor, when a transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes 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.



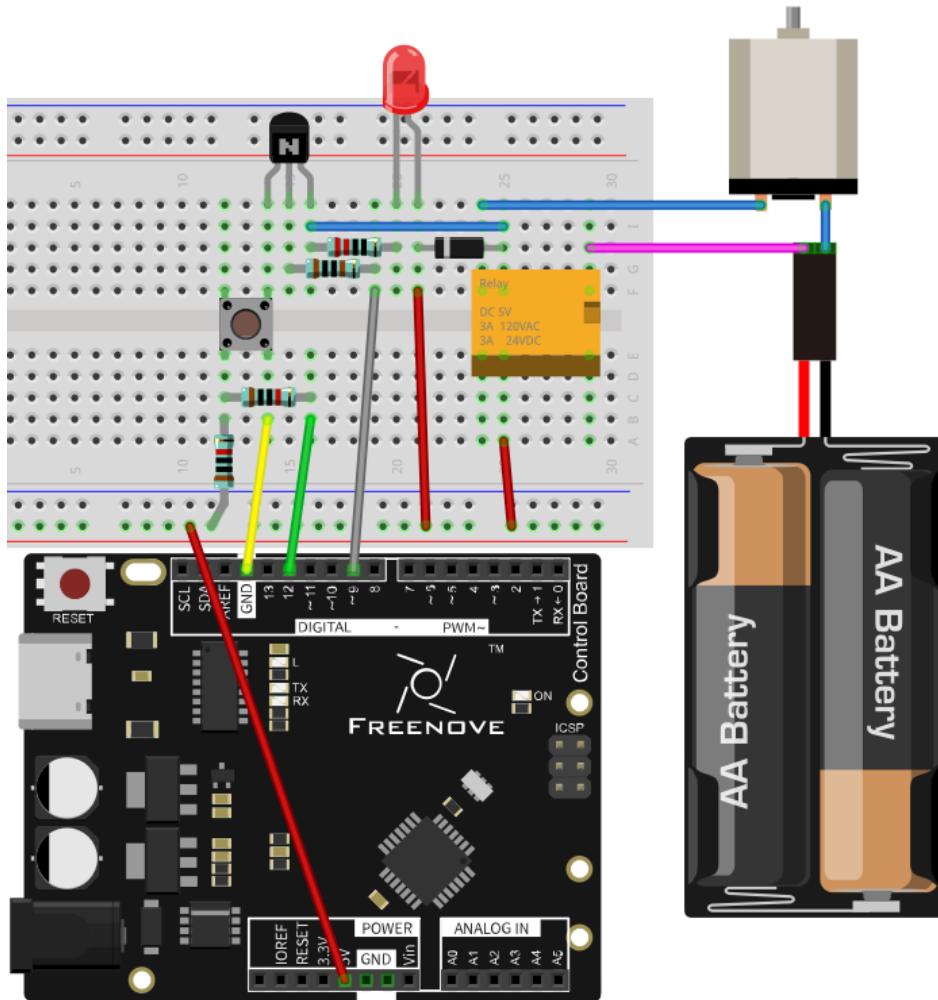
Circuit

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 connections



Sketch

Sketch 10.1.1

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   }
...
36   lastButtonState = nowButtonState; // Save the state of last button
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.

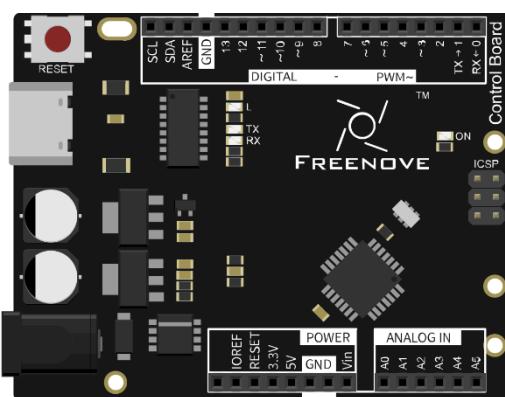
Verify and upload the code, every time you press the push button, the state of relay and motor changes once.

Project 11.2 Control Motor with L293D

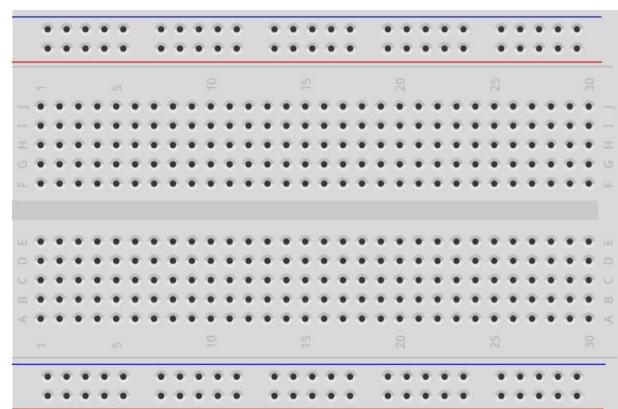
Now, we will use dedicated chip L293D to control the motor.

Component List

Control board x1



Breadboard x1



USB cable x1

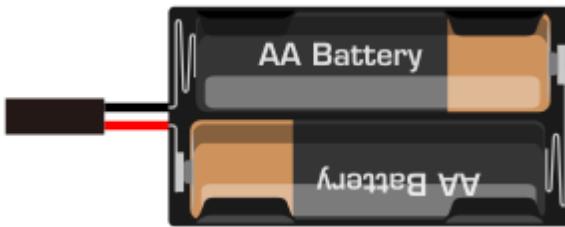


Jumper M/M x10

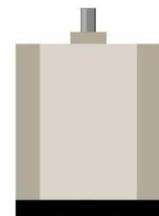
Jumper F/M x2



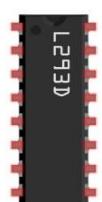
AA Battery holder x1



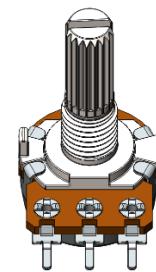
Motor x1



L293D x1



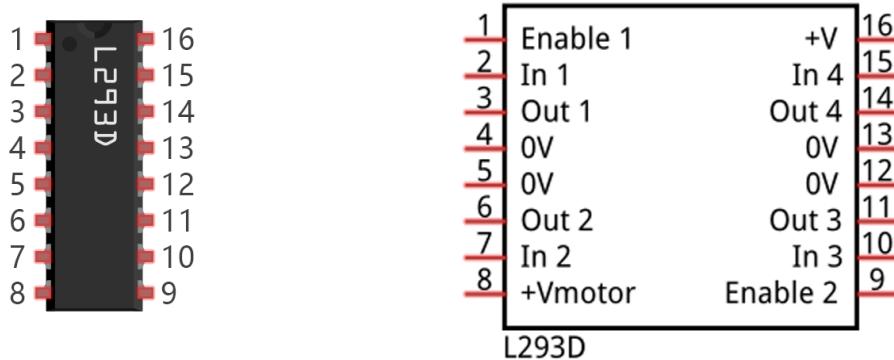
Rotary potentiometer x1



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).



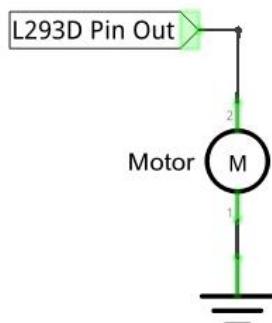
Port description of L293D module is as follows:

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

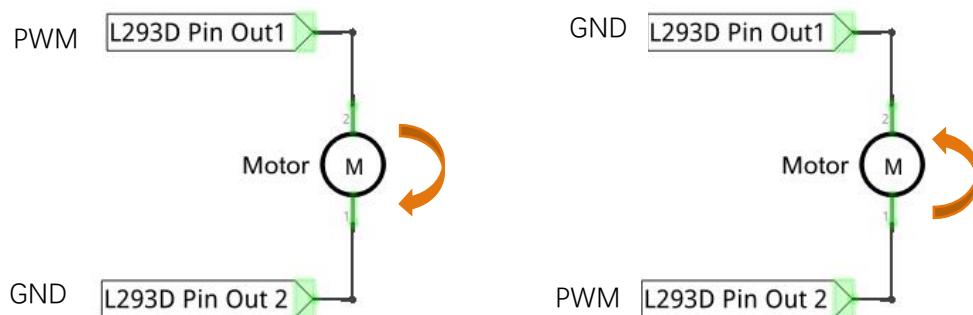
For more details, please see datasheet.

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 L293D: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, not only controls the speed of motor, but also can control the speed of the motor.

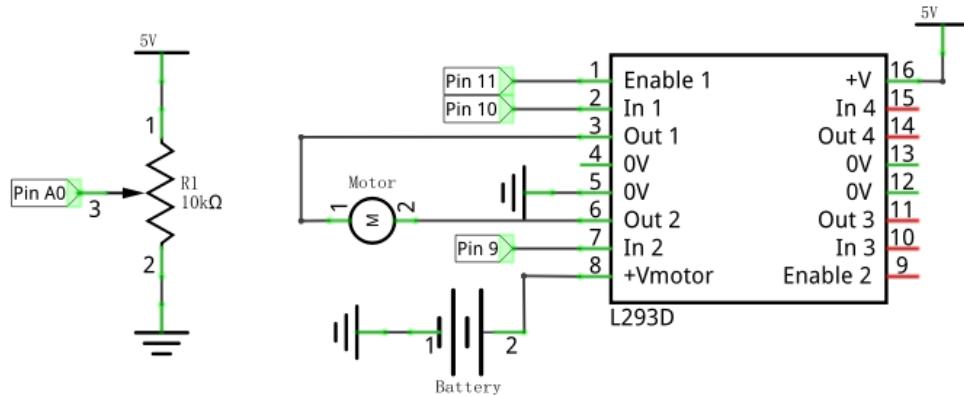


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

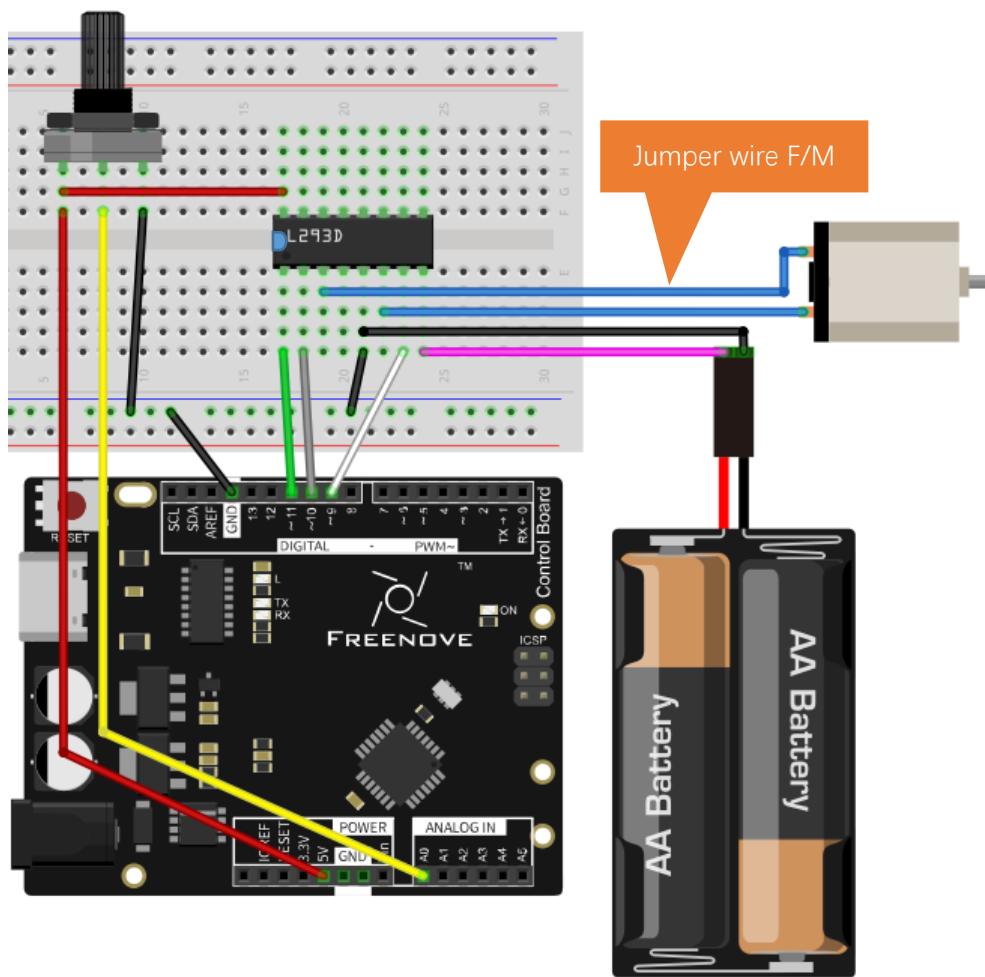
Circuit

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



Sketch

Sketch 10.2.1

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.

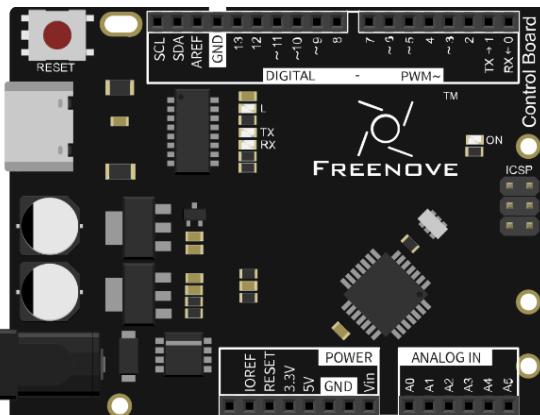
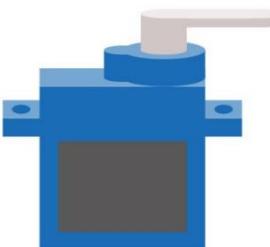
Chapter 12 Servo

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 12.1 Servo Sweep

First, let's get the servo to rotate.

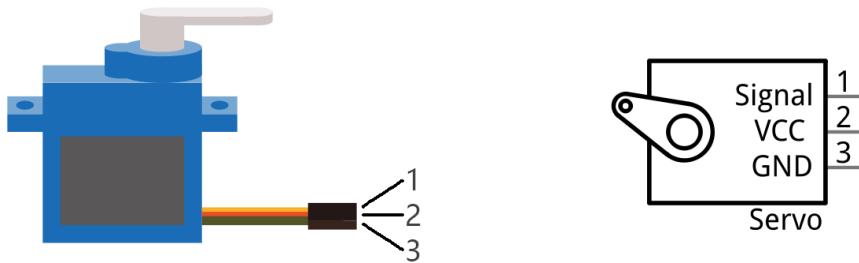
Component List

Control board x1	USB cable x1
 A black Freenove Control Board (Arduino Uno compatible) with various pins labeled: SCL, SDA, AREF, GND, 13, 12, ~11, ~10, ~9, 8, DIGITAL, PWM~, TX, RX, TM, IOREF, 3.3V, POWER, GND, V _{in} , ANALOG IN, A0, A1, A2, A3, A4, A5. It also features a red pushbutton labeled 'RESET' and a yellow LED labeled 'ON'.	 A standard black USB cable with a Type-A male connector on one end and a Type-B male connector on the other.
Jumper M/M x3	 A green jumper wire with two black pins.
Servo x1	 A blue servo motor with a white plastic horn and a grey metal gear.

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:

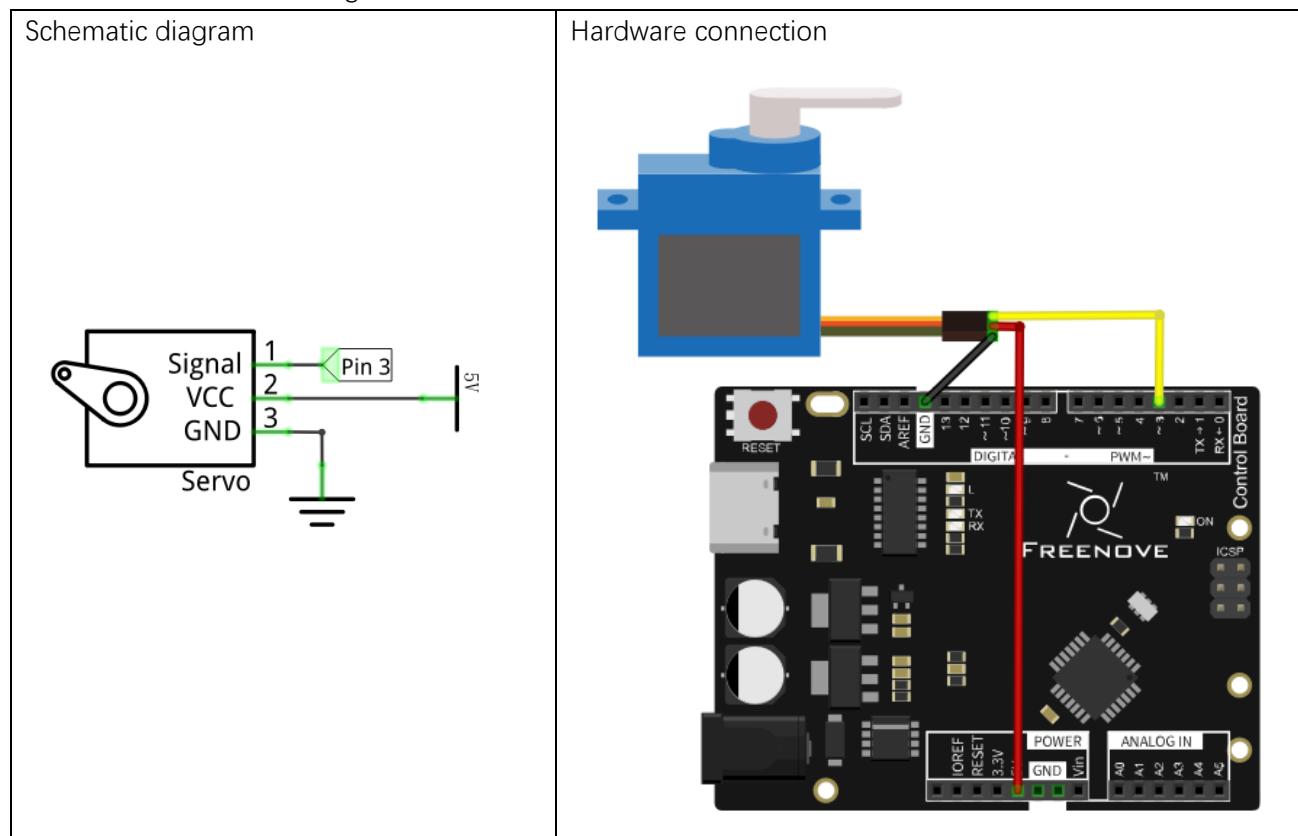
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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.



Sketch

Sketch 11.1.1

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.

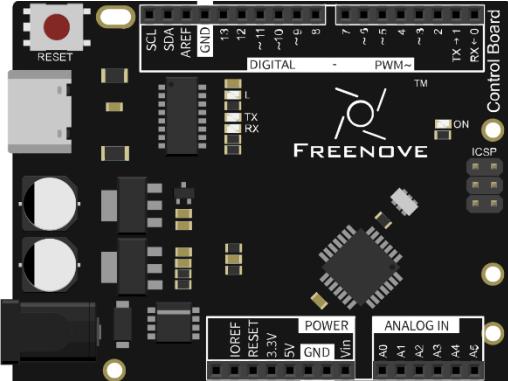
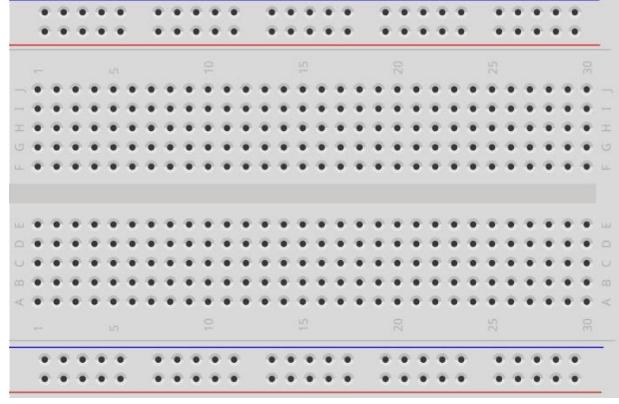
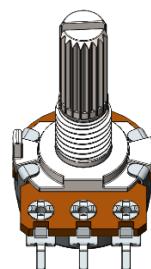
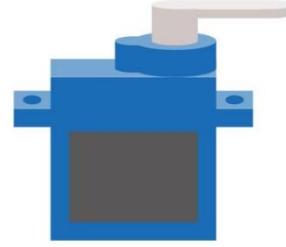
Verify and upload the code, the servo starts to sweep continuously.



Project 12.2 Control Servo with Potentiometer

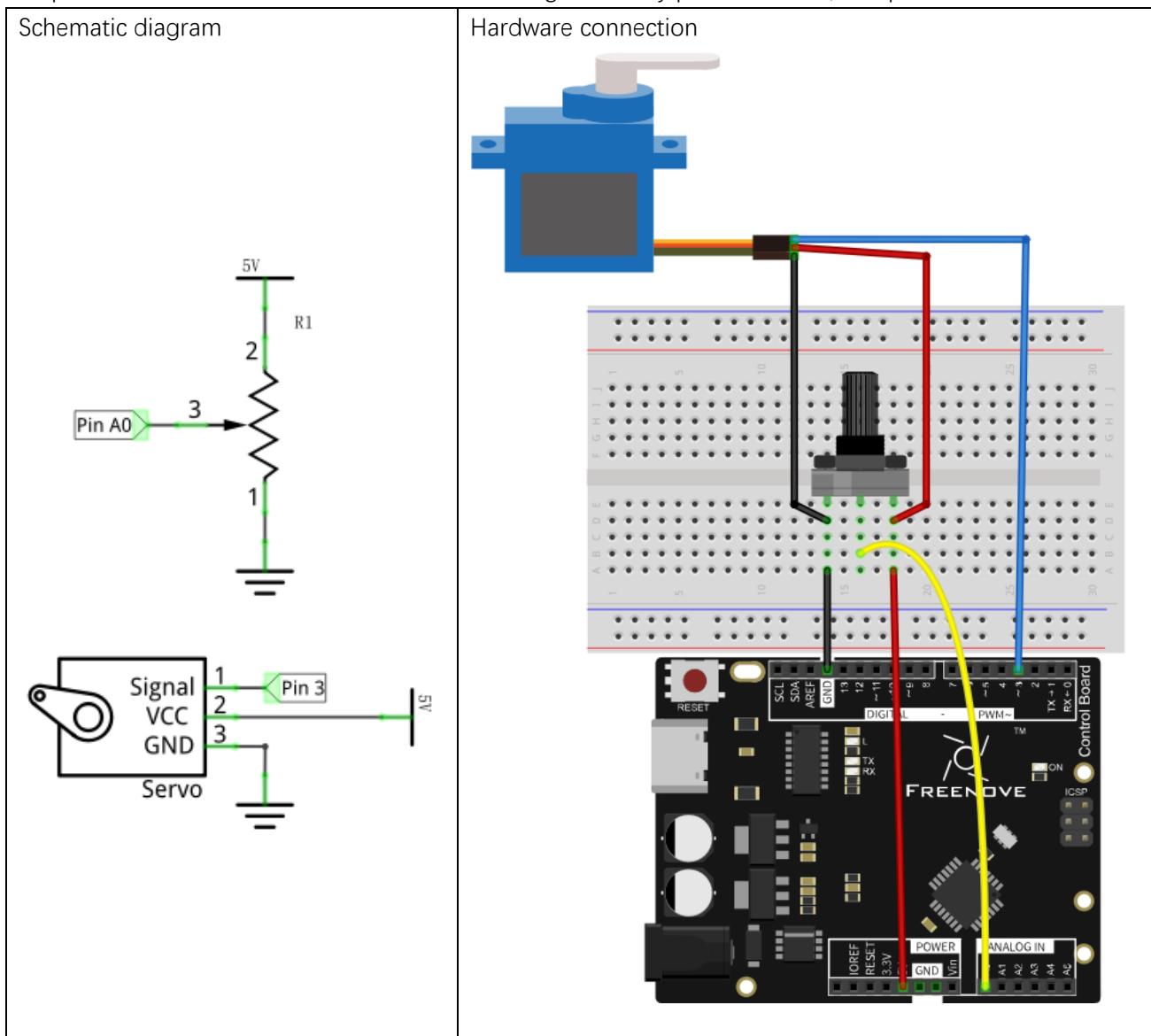
In the previous section, we've made the servo sweep continuously. Now, we will use a potentiometer to control the servo's angle.

Component List

Control board x1	Breadboard 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.



Sketch

Sketch 11.2.1

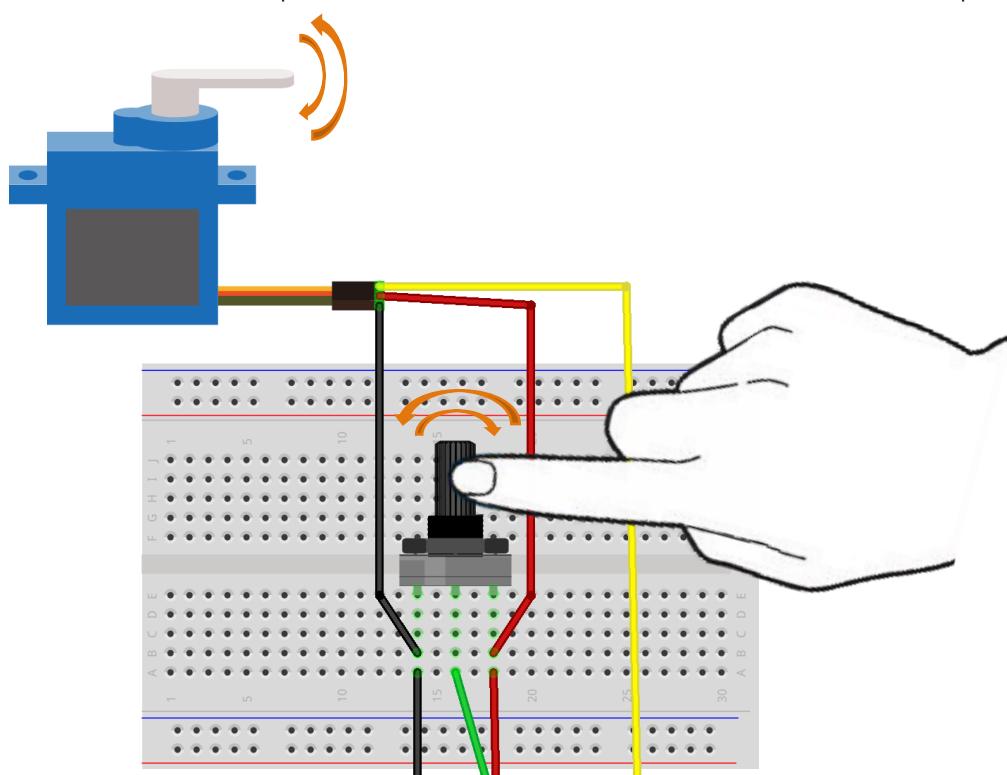
Now, write the code to detect the voltage of rotary potentiometer, and control servo to rotate to a different angle according to that.

```

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 13 LED Matrix

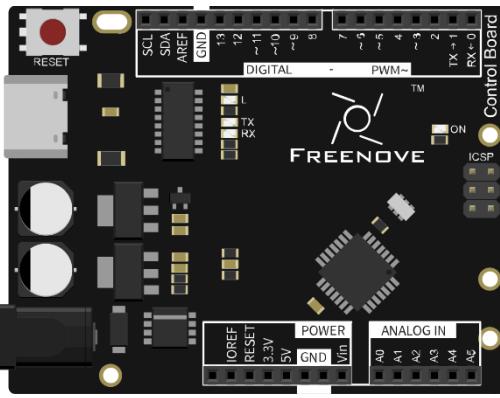
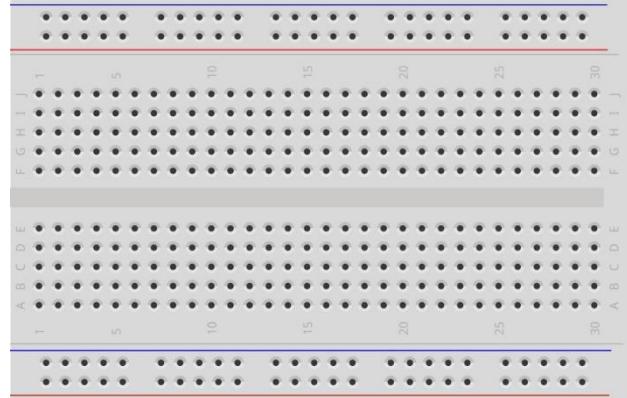
In the previous chapter, we have learned how to use some modules and sensors and shows some information on the computer through serial port. Now let us learn some modules which can output images and text.

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

Project 13.1 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	Breadboard x1
	
USB cable x1	74HC595 x1
	
Jumper M/M x17	LED bar graph x1
	
	Resistor 220Ω x8
	

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
	1	1	1

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

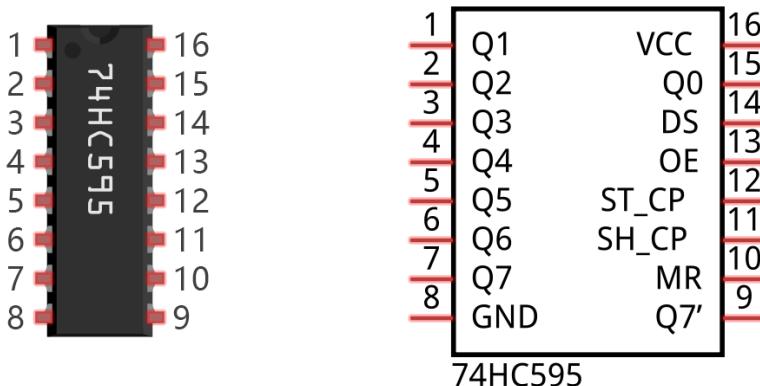
4 bit binary	1000	1001	1010	1011	1100	1101	1110	1111
1 figure of hexadecimal	8	9	A	B	C	D	E	F

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:

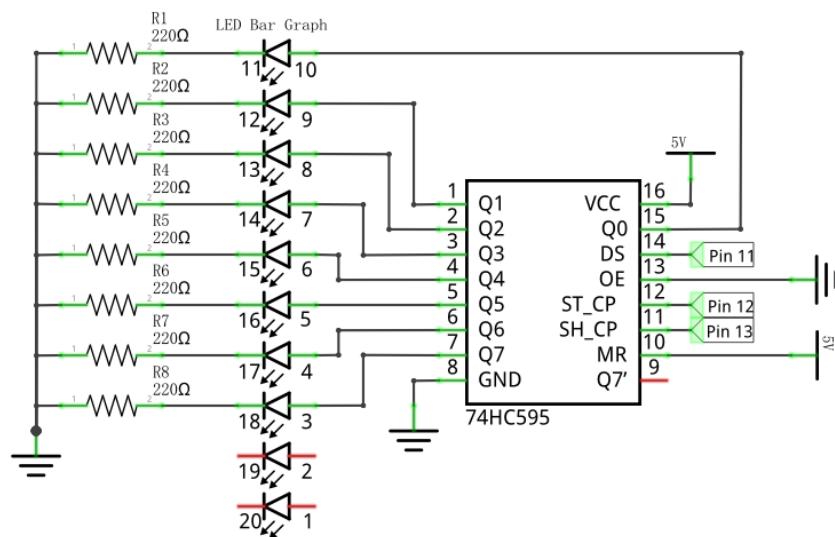
Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin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

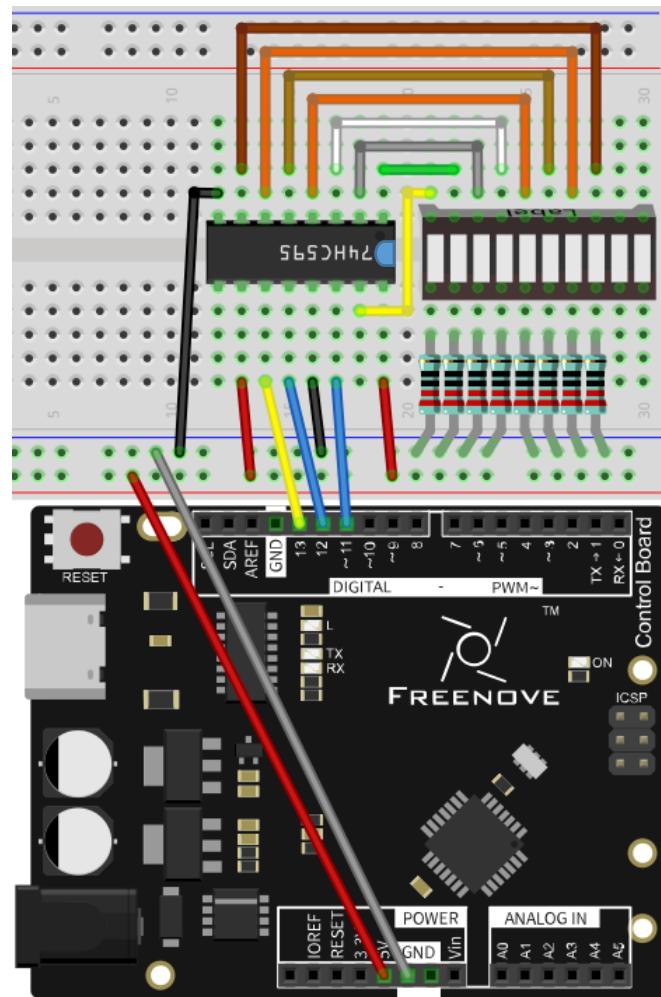
Circuit

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



Sketch

Sketch 15.1.1

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);

```

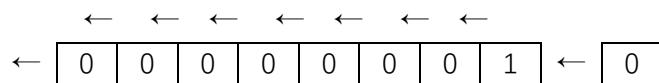
The x will be shift 1 bit to left in each cycle, which makes the bright LED of the 8 LEDs move one bit.

```
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;
```

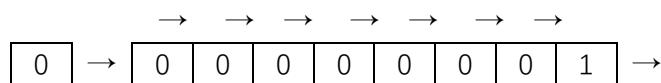


The result of x is 2 (binary 00000010).



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

```
byte x = 1 >> 1;
```

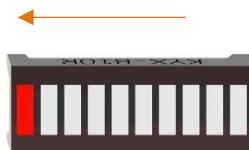


The result of x is 0 (00000000).



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

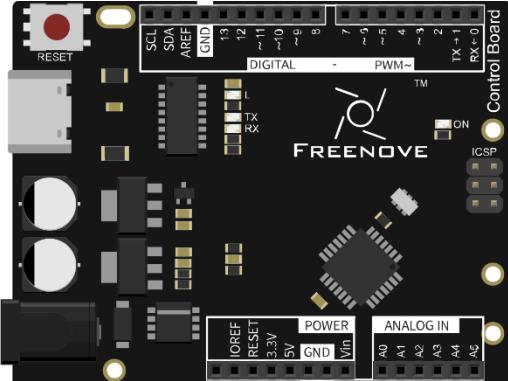
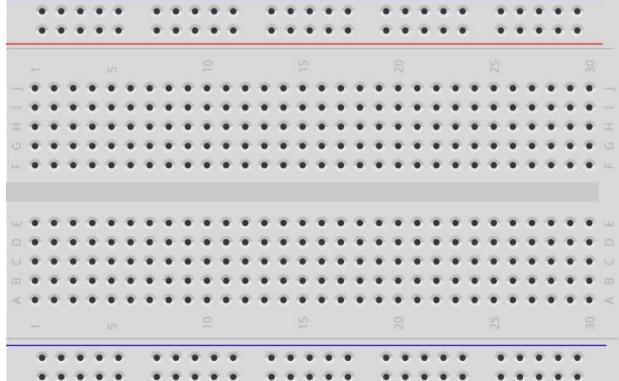
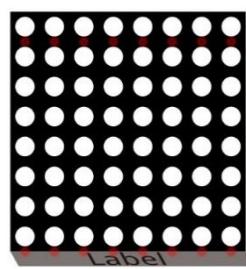
Verify and upload the code, and then you will see the LED bar graph with the effect of flowing water.



Project 13.2 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.

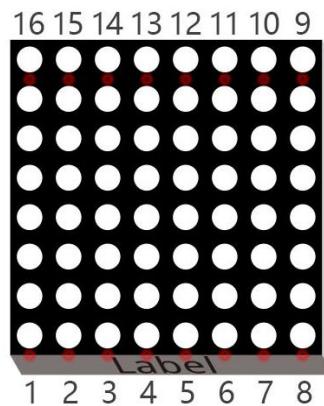
Component List

Control board x1	Breadboard x1		
			
USB cable x1	LED matrix x1	74HC595 x1	R220 x8
			
Jumper M/M x33			
			

Component Knowledge

LED matrix

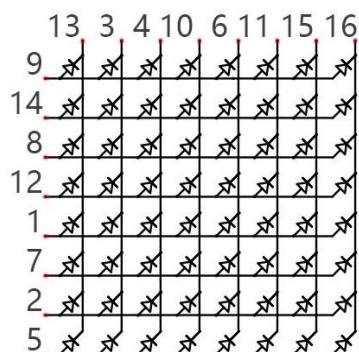
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



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.

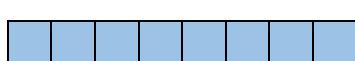
Connection mode of common anode



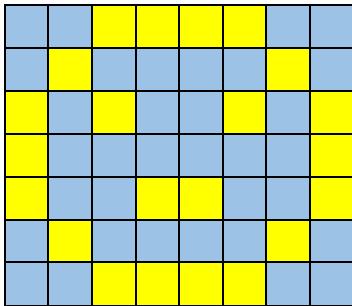
Connection mode of common cathode



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

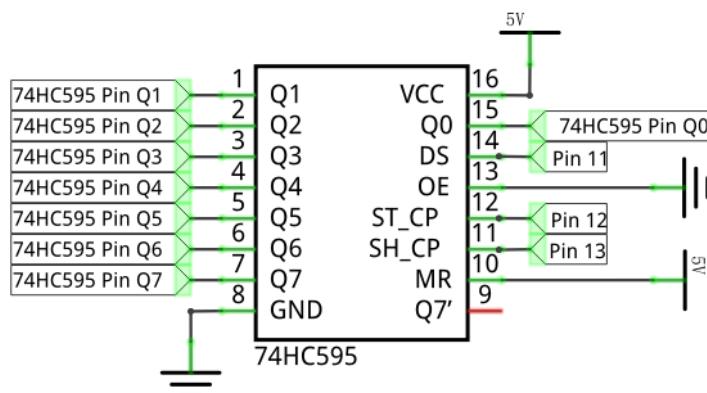
Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

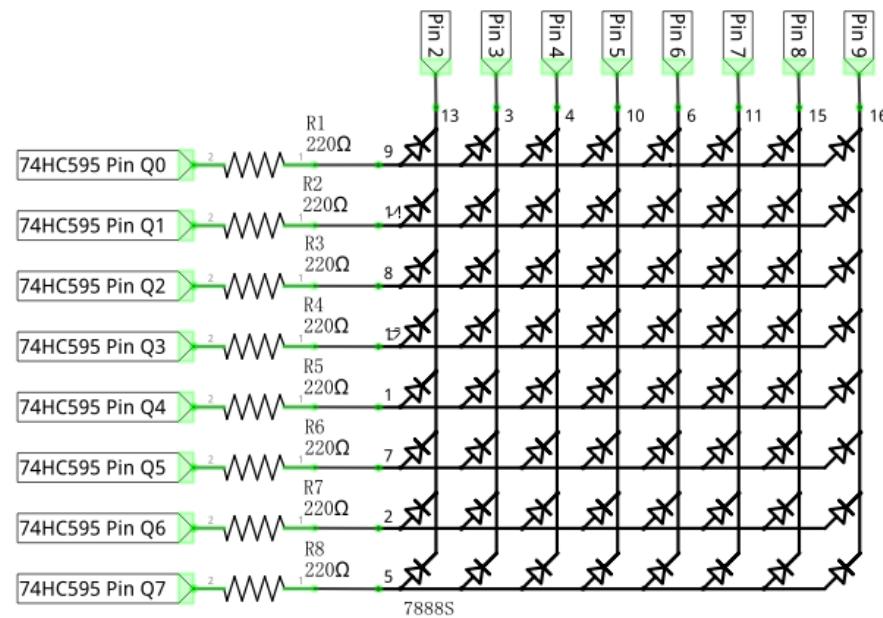
Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit.

Circuit

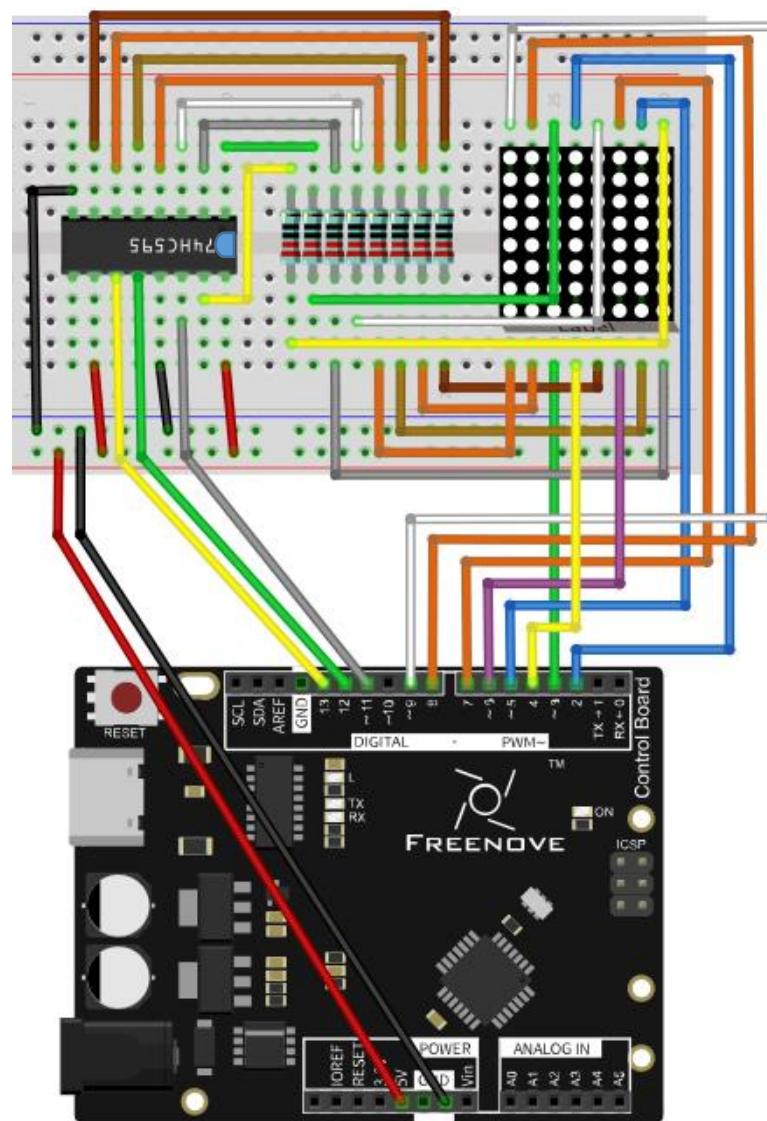
Use pin 11, 12, 13 on control board to control the 74HC595. And connect 74HC595 to the 8 anode pins of LED Matrix, in the meanwhile, connect 8 digital port on control board to the 8 cathode pins of LED Matrix.

Schematic diagram





Hardware connection



Sketch

Sketch 15.2.1

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, 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     }
57     // Display the dynamic patterns of numbers and letters
58     for (int i = 0; i < 128; i++) { // "space, 0-9, A-F"16 letters , each letter hold 8
59     // columns, total 136 columns. Firstly, display space ,then we need shift 128 times (136-8)
60     for (int k = 0; k < 10; k++) { // repeat image of each frame 10 times.
61         cols = 0x01; // Assign binary 00000001. Means the first column is selected.
62         for (int j = i; j < 8 + i; j++) { // display image of each frame
63             matrixColsVal(cols); // select this column
64             matrixRowsVal(pgm_read_word_near(data + j)); // display the data in this column
65             delay(1); // display them for a period of time
66             matrixRowsVal(0x00); // close the data of this column
67             cols <<= 1; // shift "cols" 1 bit left to select the next column
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
76     // output
77     digitalWrite(latchPin, HIGH);
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 int LEDPin[] = {2, 3, 4, 5, 6, 7, 8, 9}; // Column pins (cathode) of LED Matrix
```

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

```

7 const int smilingFace[] = {
8     0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C
9 };
```

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 const int data[] PROGMEM = {
12     0x00, 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 };
```

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);
78 }
```

?: 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 (^), negate (~). The result of exclusive or (^) is true only when two corresponding bit is not equal.

&, | and ^ 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:

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

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

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

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

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

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
        column is selected
47     for (int i = 0; i < 8; i++) { // display the data in 8 column data by scanning
48       matrixColsVal(cols); // select this column
49       matrixRowsVal(smilingFace[i]); // display data in this column
50       delay(1); // display the data for a period of time
51       matrixRowsVal(0x00); // close the data of this column
52       cols <= 1; // shift "cols" 1 bit left to select the next column
53     }
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
      columns, total 136 columns. Firstly, display space ,then we need shift 128 times (136-8)
57     for (int k = 0; k < 10; k++) { // repeat image of each frame 10 times.
58       cols = 0x01; // Assign binary 00000001. Means the first column is selected.
59       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 14 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:

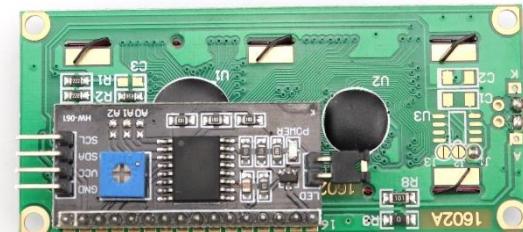
LCD1602

Please choose "Chapter 16A LCD1602"



I2C LCD1602

Please choose "Chapter 16B I2C LCD1602"



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.

Note: Please select the corresponding chapter according to your model.

Chapter 14A LCD1602

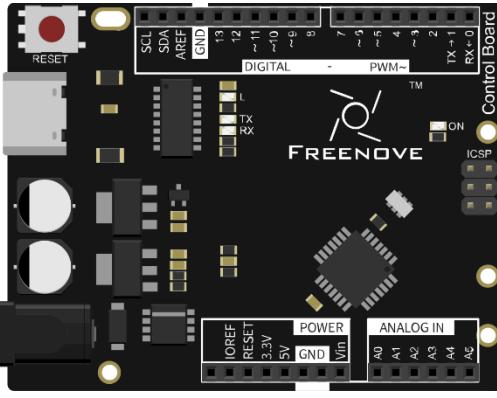
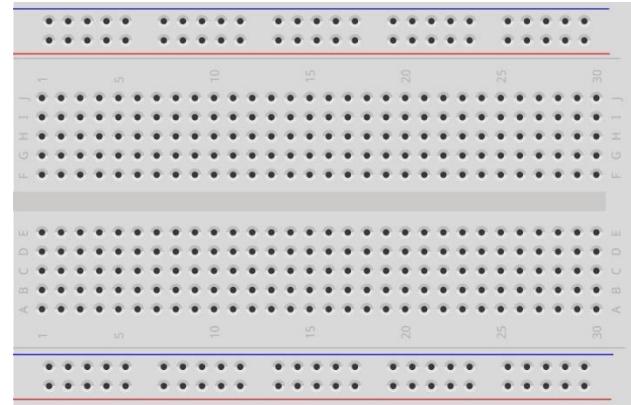
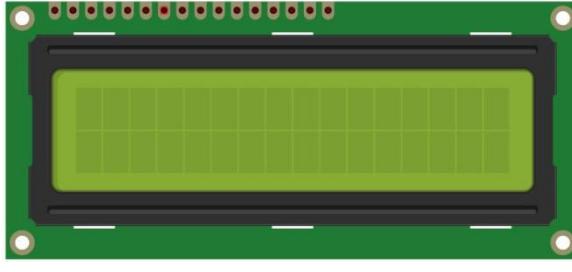
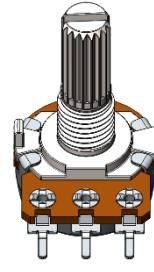
Note: Please select the appropriate chapter according to your LCD model.

This chapter only applies to LCD1602. Please select another chapter for I2C LCD1602.

Project 14A.1 Display the String on LCD1602

Firstly, use LCD1602 to display some strings.

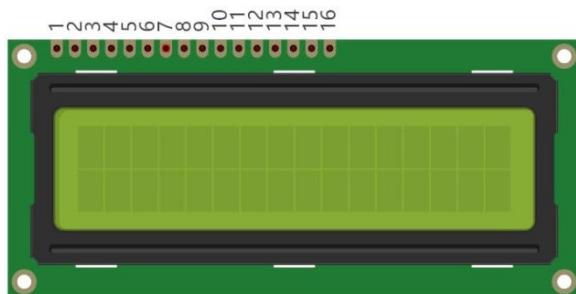
Component List

Control board x1	Breadboard x1
	
USB cable x1	Jumper M/M x16
	
LCD1602 x1	Rotary potentiometer x1
	

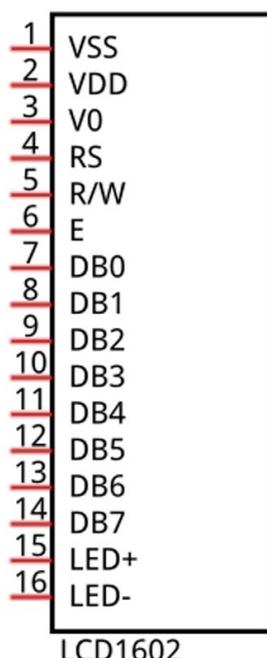
Component Knowledge

LCD1602

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



Circuit symbol and pin descriptions of LCD1602 are shown as follows:



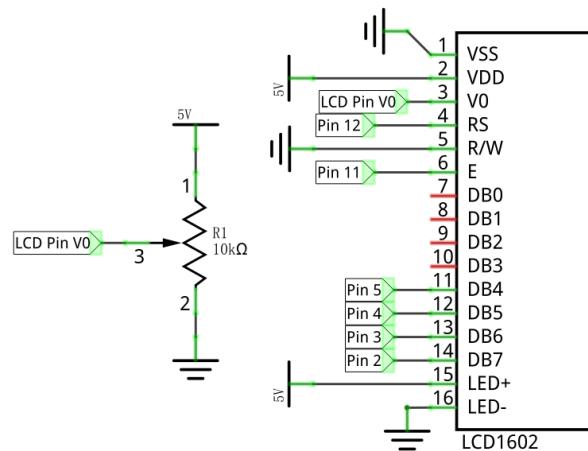
Pin name	Pin number	Description
VSS	1	Negative electrode of power supply
VDD	2	Positive electrode of power supply, the voltage is 5V
VO	3	Contrast and adjust the display effect
RS	4	Data/Command selection
RW	5	Read/Write selection
E	6	Enable pin
D0-D7	7-14	Data pin
LED+	15	Positive electrode of backlight LED, the voltage is 5V
LED-	16	Negative electrode of backlight LED

For more details, please refer to the datasheet.

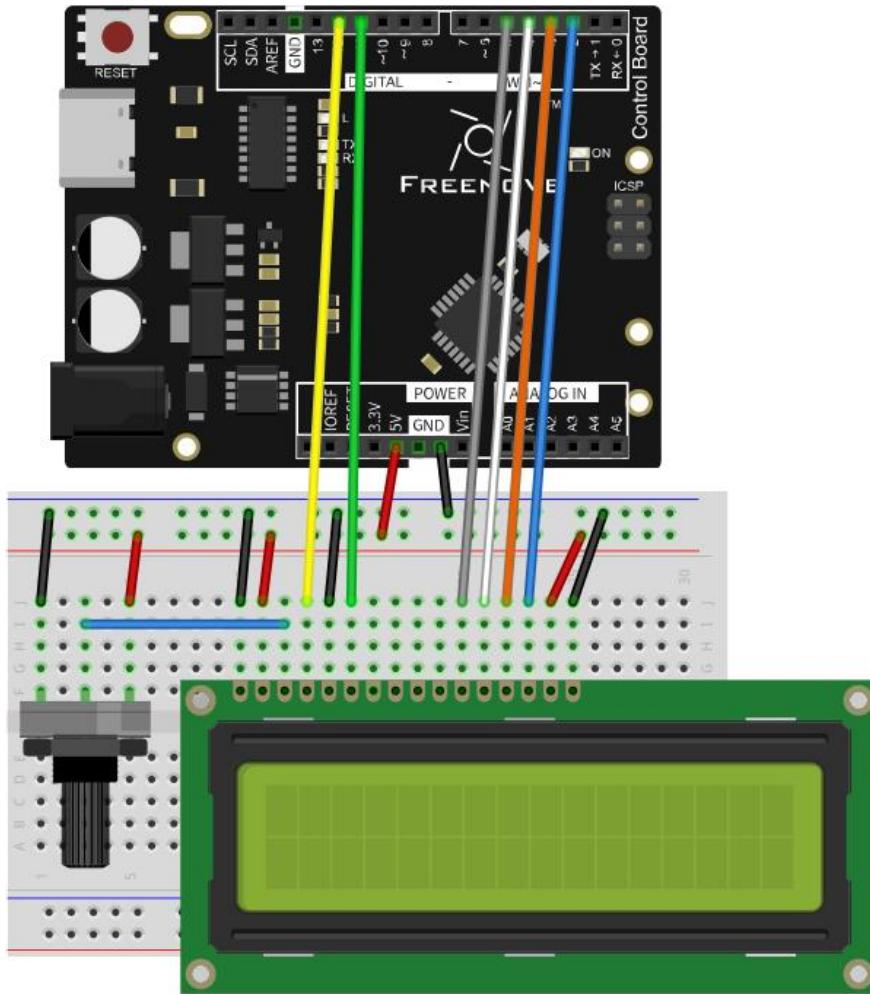
Circuit

The connection of control board and LCD1602 is shown below. And use a rotary potentiometer to adjust the contrast of LCD1602.

Schematic diagram



Hardware connection



Sketch

Sketch 16.1.1

Now write code to make LCD1602 display a string and changing numbers.

```
1 #include <LiquidCrystal.h>
2
3 // initialize the library with the numbers of the interface pins
4 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
5
6 void setup() {
7     // set up the LCD's number of columns and rows:
8     lcd.begin(16, 2);
9     // Print a message to the LCD
10    lcd.print("hello, world!");
11 }
12
13 void loop() {
14     // set the cursor to column 0, line 1
15     // (note: line 1 is the second row, since counting begins with 0):
16     lcd.setCursor(0, 1);
17     // print the number of seconds since reset:
18     lcd.print("Counter:");
19     lcd.print(millis() / 1000);
20 }
```

Use LiquidCrystal library to control the LCD:

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

LiquidCrystal library provides a specific class used to control LCD1602. We instantiate a LiquidCrystal object, and we can input some parameters which is related to pins of the LCD1602:

```
4 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Initialize the LCD1602 in the setup () function, and the parameters are the number of columns and rows of characters which LCD can display:

```
8 lcd.begin(16, 2);
```

And then print a string:

```
10 lcd.print("hello, world!");
```

Print a changing number in the loop () function:

```
13 void loop() {
14     // set the cursor to column 0, line 1
15     // (note: line 1 is the second row, since counting begins with 0):
16     lcd.setCursor(0, 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);
```

If we don't set the coordinate, the string will be printed behind the previous printed character.

LiquidCrystal class

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

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

when instantiate the object, we need to write the pins connected to the LCD. There are many different ways for instantiation LiquidCrystal class. The way we use is shown below:

```
LiquidCrystal(rs, enable, d4, d5, d6, d7);
```

The common function of LiquidCrystal is shown below:

`lcd.begin(cols, rows)`: Initialize the LCD, and the parameters are the number of columns and rows of characters which LCD can display;

`lcd.setCursor(col, row)`: set the coordinate of the printed character, and the parameters is the row and the column (starting from 0, the number 0 represents first line or column);

`lcd.print(data)`: print the number and letters in the coordinate we set before. If we haven't set the coordinate, the string will be printed behind the last printed character.

Verify and upload the code, then you will see the LCD1602 display the string and changing numbers.

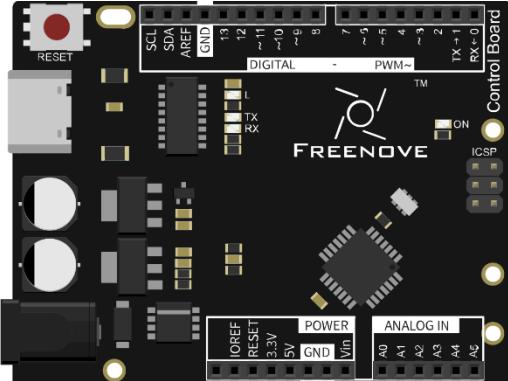
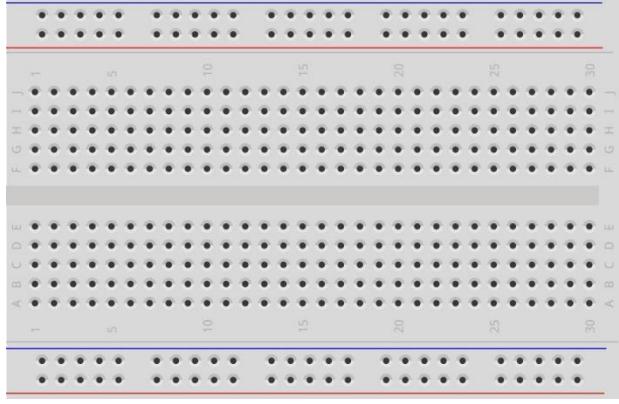
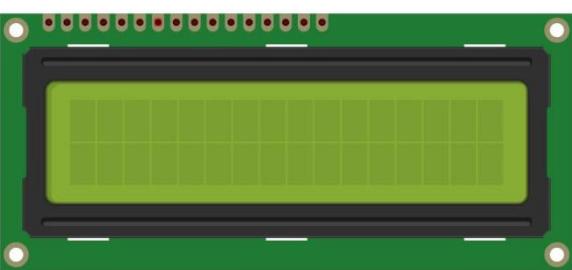
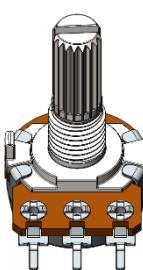
If the display of characters is not normal, shift the rotary potentiometer to adjust the contrast.



Project 14A.2 LCD1602 Clock

In the previous chapter, we have used LCD1602 to display some strings, and now let us use LCD1206 to make a clock.

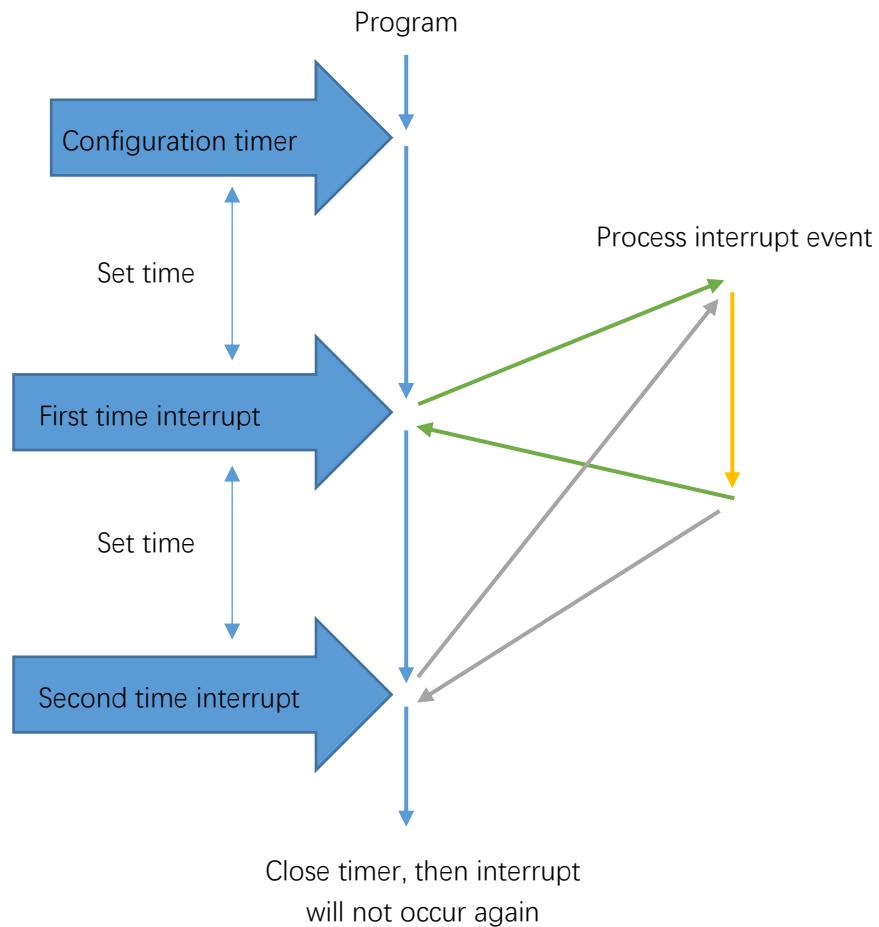
Component List

Control board x1	Breadboard x1		
			
USB cable x1	Jumper M/M x18		
			
LCD1602 x1	Rotary potentiometer x1	Thermistor x1	Resistor 10kΩ x1
			

Code Knowledge

Timer

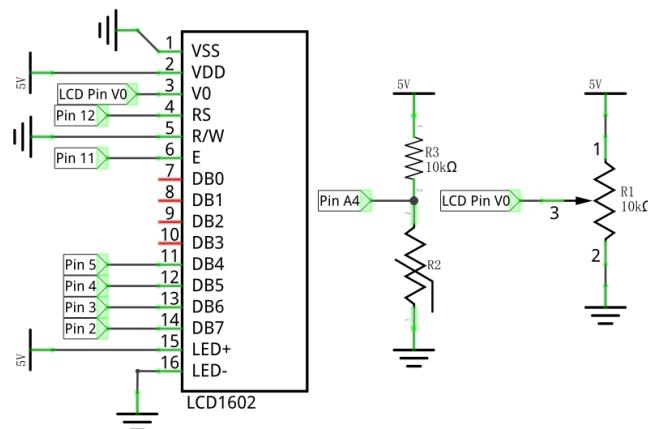
A timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted location to go on. If you don't close the timer,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



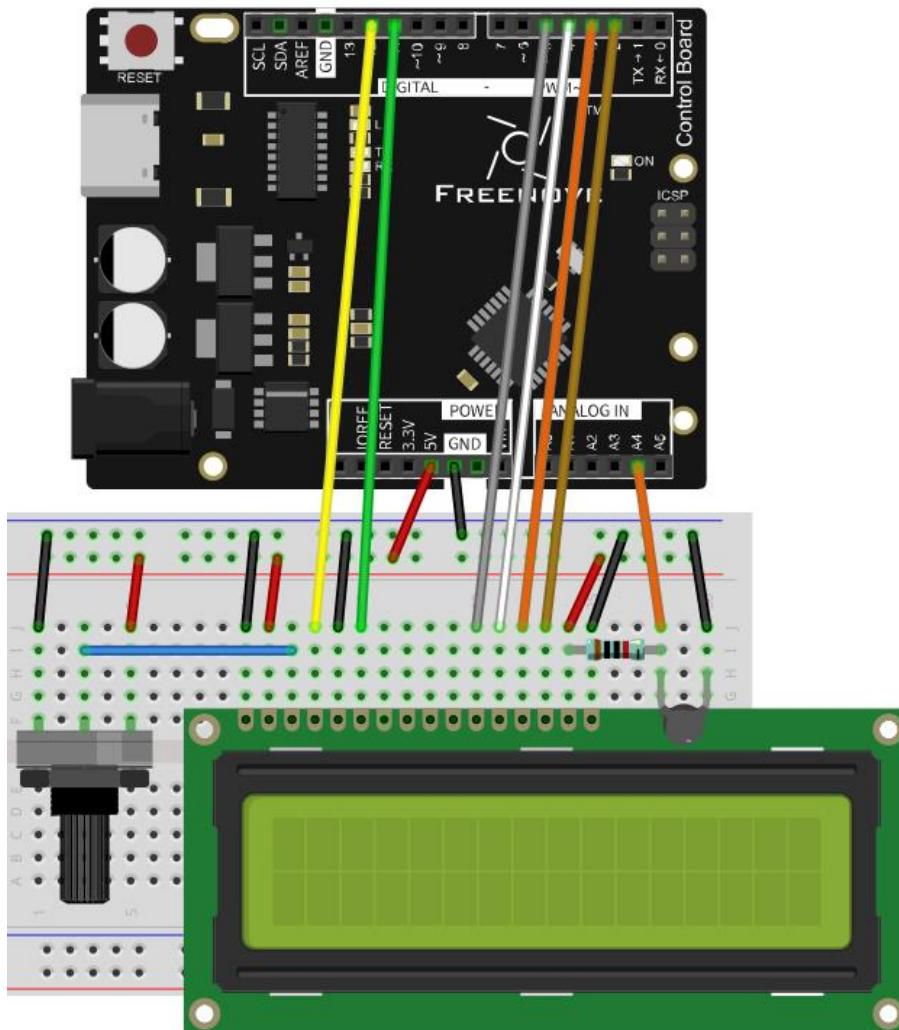
Circuit

The connection between control board and LCD1602 is shown below, and a rotary potentiometer is used to adjust the contrast of LCD1602. Pin A4 port is used to detect the voltage of thermistor.

Schematic diagram



Hardware connection



Sketch

Sketch 16.2.1

Before writing code, we need to import the library needed.

Click "Add .ZIP Library..." and then find FlexiTimer2.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library can help manipulate the timer.

Now write code to make LCD1602 display the time and temperature, and the time can be modified through the serial port.

```
1 #include <LiquidCrystal.h>      // Contains LiquidCrystal Library
2 #include <FlexiTimer2.h>        // Contains FlexiTimer2 Library
3
4 // initialize the library with the numbers of the interface pins
5 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
6
7 int tempPin = 4;                // define the pin of temperature sensor
8 float tempVal;                 // define a variable to store temperature value
9 int hour, minute, second;       // define variables stored record time
10
11 void setup() {
12     lcd.begin(16, 2);          // set up the LCD's number of columns and rows
13     startingAnimation();       // display a dynamic start screen
14     FlexiTimer2::set(1000, timerInt); // configure the timer and interrupt function
15     FlexiTimer2::start();       // start timer
16     Serial.begin(115200);      // initialize serial port with baud rate 115200
17     Serial.println("Input hour, minute, second to set time.");
18 }
19
20 void loop() {
21     // Get temperature
22     tempVal = getTemp();
23     if (second >= 60) {         // when seconds is equal to 60, minutes plus 1
24         second = 0;
25         minute++;
26     if (minute >= 60) {        // when minutes is equal to 60, hours plus 1
27         minute = 0;
28         hour++;
29         if (hour >= 24) {       // when hours is equal to 24, hours turn to zero
30             hour = 0;
31         }
32     }
33 }
34 lcdDisplay();                  // display temperature and time information on LCD
35 delay(200);
```

```
36 }
37
38 void startingAnimation() {
39     for (int i = 0; i < 16; i++) {
40         lcd.scrollDisplayRight();
41     }
42     lcd.print("starting... ");
43     for (int i = 0; i < 16; i++) {
44         lcd.scrollDisplayLeft();
45         delay(300);
46     }
47     lcd.clear();
48 }
49
50 // the timer interrupt function of FlexiTimer2 is executed every 1s
51 void timerInt() {
52     second++;      // second plus 1
53 }
54
55 // serial port interrupt function
56 void serialEvent() {
57     int inInt[3]; // define an array to save the received serial data
58     while (Serial.available()) {
59         for (int i = 0; i < 3; i++) {
60             inInt[i] = Serial.parseInt(); // receive 3 integer data
61         }
62         // print the received data for confirmation
63         Serial.print("Your input is: ");
64         Serial.print(inInt[0]);
65         Serial.print(", ");
66         Serial.print(inInt[1]);
67         Serial.print(", ");
68         Serial.println(inInt[2]);
69         // use received data to adjust time
70         hour = inInt[0];
71         minute = inInt[1];
72         second = inInt[2];
73         // print the modified time
74         Serial.print("Time now is: ");
75         Serial.print(hour / 10);
76         Serial.print(hour % 10);
77         Serial.print(':');
78         Serial.print(minute / 10);
79         Serial.print(minute % 10);
```

```
80     Serial.print(':' );
81     Serial.print(second / 10);
82     Serial.println(second % 10);
83 }
84 }
85
86 // function used by LCD1602 to display time and temperature
87 void lcdDisplay() {
88     lcd.setCursor(0, 0); // set the cursor to (0,0) (first column, first row).
89     lcd.print("TEMP: "); // display temperature information
90     lcd.print(tempVal);
91     lcd.print("C");
92     lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second row)
93     lcd.print("TIME: "); // display time information
94     lcd.print(hour / 10);
95     lcd.print(hour % 10);
96     lcd.print(':' );
97     lcd.print(minute / 10);
98     lcd.print(minute % 10);
99     lcd.print(':' );
100    lcd.print(second / 10);
101    lcd.print(second % 10);
102 }
103
104 // function used to get temperature
105 float getTemp() {
106     // Convert analog value of tempPin into digital value
107     int adcVal = analogRead(tempPin);
108     // Calculate voltage
109     float v = adcVal * 5.0 / 1024;
110     // Calculate resistance value of thermistor
111     float Rt = 10 * v / (5 - v);
112     // Calculate temperature (Kelvin)
113     float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));
114     // Calculate temperature (Celsius)
115     return tempK - 273.15;
116 }
```

In the code, we define 3 variables to represent time: second, minute, hour.

9	int hour, minute, second; // define variables to store hour, minute, seconds
---	------------------------------------------------------------------------------

Define 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 parameter to FlexiTimer2::set () function.

```
14   FlexiTimer2::set(1000, timerInt); // configure timer and timer interrupt function
15   FlexiTimer2::start();           // start timer
```

After every interrupt, the second plus 1.

```
51 void timerInt() {
52     sec++; // second plus 1
53 }
```

:: operator

"::" is scope operator. The function behind "::" belongs to the scope of the front. If we want to call the function defined in the FlexiTimer2 scope outside, we need to use the "::". It can be global scope operator, class scope operator and namespace scope operator. It is a namespace scope operator here. Because functions of FlexiTimer2 library is defined in the namespace of FlexiTimer2, so we can find them in FlexiTimer2 library file.

In the loop () function, the information on the LCD display will be refreshed at set intervals.

```
20 void loop() {
...
34     lcdDisplay();           // display temperature and time information on LCD
35     delay(200);
36 }
```

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.

```
23 if (second >= 60) {           // when the second increases to 60, the minute adds 1
24     second = 0;
25     minute++;
26     if (minute >= 60) {       //when the minute increases to 60, the hour adds 1
27         minute = 0;
28         hour++;
29         if (hour >= 24) {     // when the hour increases to 24, turn it to zero
30             hour = 0;
31         }
32     }
33 }
```

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.

```
87 void lcdDisplay() {
88     lcd.setCursor(0, 0); // set the cursor to (0, 0) (first column, first row).
89     lcd.print("TEMP: "); // display temperature information
90     lcd.print(tempVal);
91     lcd.print("C");
```

```
92 lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second row)
93 lcd.print("TIME: "); // display time information
94 lcd.print(hour / 10);
95 lcd.print(hour % 10);
96 lcd.print(' : ');
97 lcd.print(minute / 10);
98 lcd.print(minute % 10);
99 lcd.print(' : ');
100 lcd.print(second / 10);
101 lcd.print(second % 10);
102 }
```

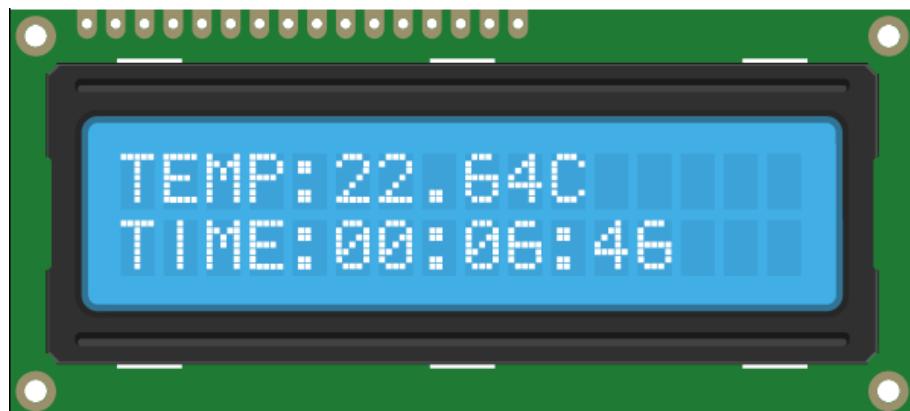
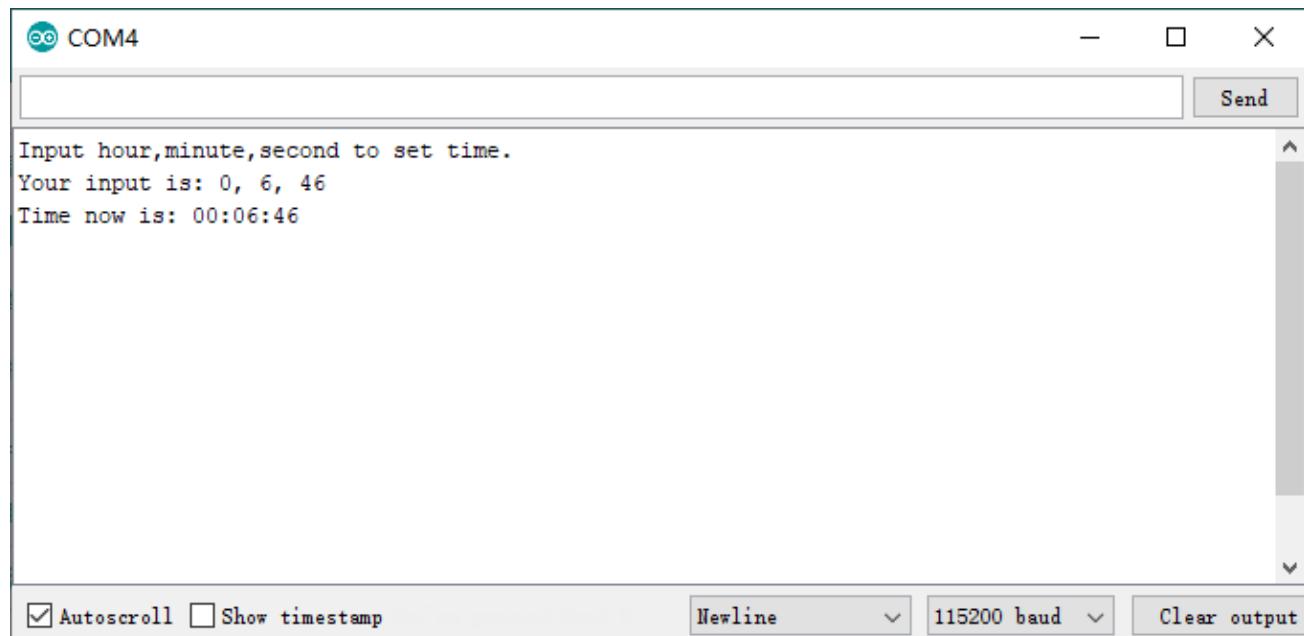
Serial port interrupt function is used to receive the data sent by computer to adjust the time, and return the data for confirmation.

```
56 void serialEvent() {
57     int inInt[3]; // define an array to save the received serial data
58     while (Serial.available()) {
59         for (int i = 0; i < 3; i++) {
60             inInt[i] = Serial.parseInt(); // receive 3 integer data
61         }
62         // print the received data for confirmation
63         Serial.print("Your input is: ");
64         Serial.print(inInt[0]);
65         Serial.print(", ");
66         Serial.print(inInt[1]);
67         Serial.print(", ");
68         Serial.println(inInt[2]);
69         // use the received data to adjust time
70         hour = inInt[0];
71         minute = inInt[1];
72         second = inInt[2];
73         // print the modified time
74         Serial.print("Time now is: ");
75         Serial.print(hour / 10);
76         Serial.print(hour % 10);
77         Serial.print(' : ');
78         Serial.print(minute / 10);
79         Serial.print(minute % 10);
80         Serial.print(' : ');
81         Serial.print(second / 10);
82         Serial.println(second % 10);
83     }
84 }
```

We also define a function that displays a scrolling string when the control board has been just started.

```
38 void startingAnimation() {  
39     for (int i = 0; i < 16; i++) {  
40         lcd.scrollDisplayRight();  
41     }  
42     lcd.print("starting...");  
43     for (int i = 0; i < 16; i++) {  
44         lcd.scrollDisplayLeft();  
45         delay(300);  
46     }  
47     lcd.clear();  
48 }
```

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 14B I2C LCD1602

Note: Please select the appropriate chapter according to your LCD model.

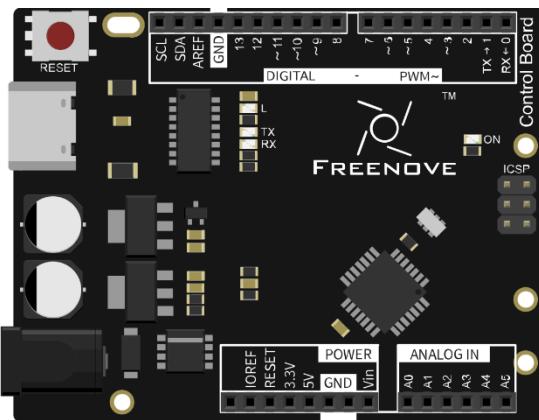
This chapter only applies to I2C LCD1602. Please select another chapter for LCD1602.

Project 14B.1 Display the String on I2C LCD1602

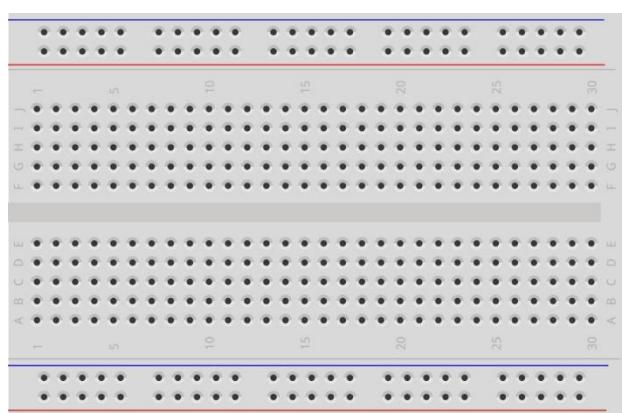
Firstly, use I2C LCD1602 to display some strings.

Component List

Control board x1



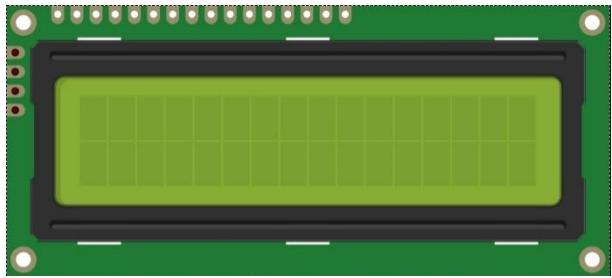
Breadboard x1



USB cable x1



I2C LCD1602 Module x1



Jumper M/F x4

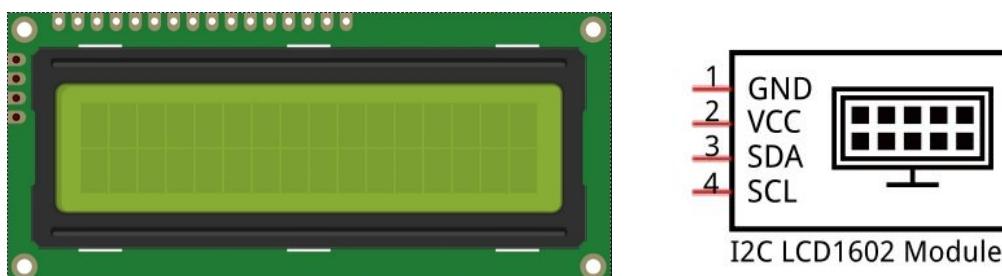


Component Knowledge

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.



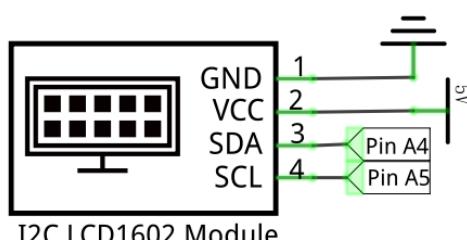
I2C (Inter-Integrated Circuit) is a two-wire serial communication, which is used to connect micro controller with its peripheral equipments. Device that use I2C communication are all connected to the serial data (SDA) line and a serial clock (SCL) line (called I2C bus). Each device among them has a unique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

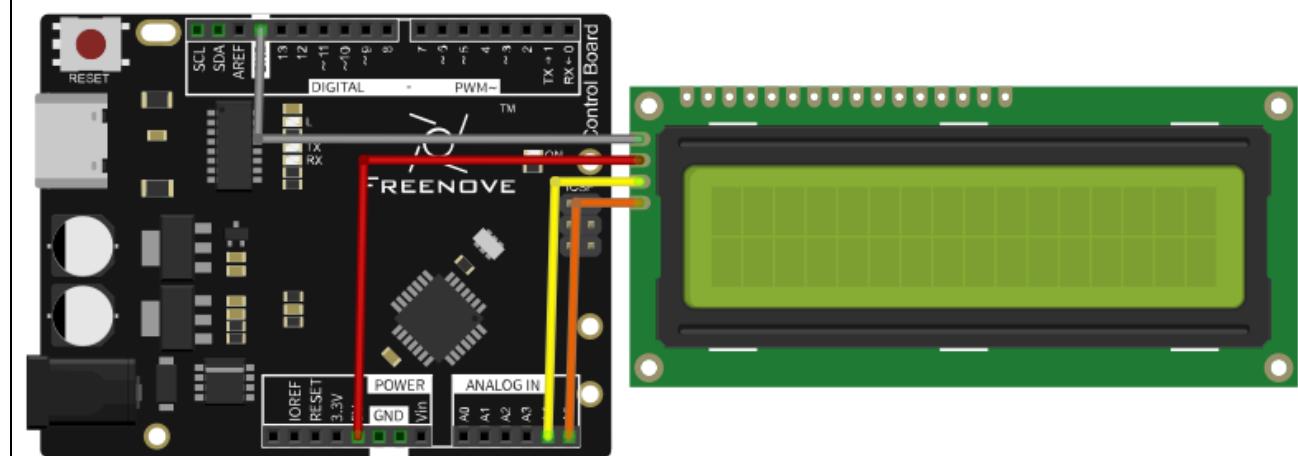
Circuit

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

Schematic diagram



Circuit connection



Sketch

Sketch 16.1.1

Before writing code, we need to import the library needed.

Click “Add .ZIP Library...” and then find LiquidCrystal_I2C.zip in libraries folder (this folder is in the folder unzipped from 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     lcd.init();           // initialize the lcd
8     lcd.backlight();      // Turn on backlight
9     lcd.print("hello, world!"); // Print a message to the LCD
10 }
11 void loop() {
12     // (note: line 1 is the second row, since counting begins with 0):
13     lcd.setCursor(0, 1); // set the cursor to column 0, line 1
14     // print the number of seconds since reset:
15     lcd.print("Counter:");
16     lcd.print(millis() / 1000);
17 }
```

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.

```
7 lcd.init();           // initialize the lcd
8 lcd.backlight();      // Turn on backlight
```

And then print a string:

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

Print a changing number in the loop () function:

```

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

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

```
13     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.

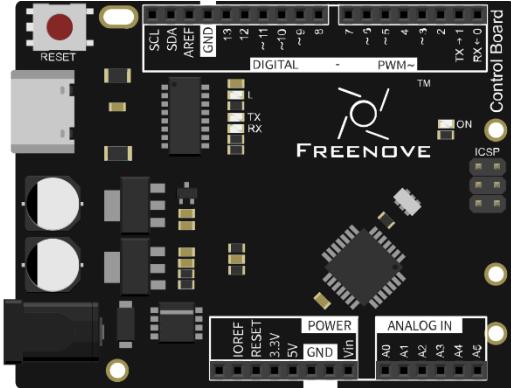
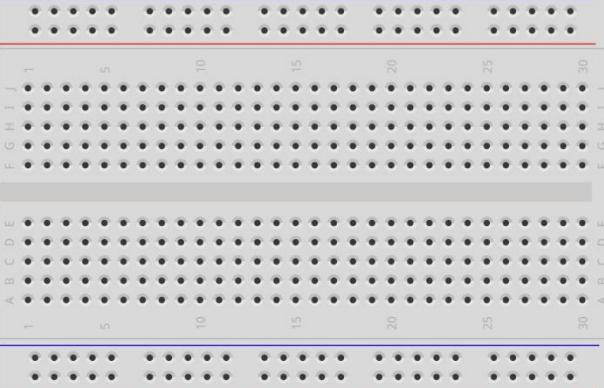
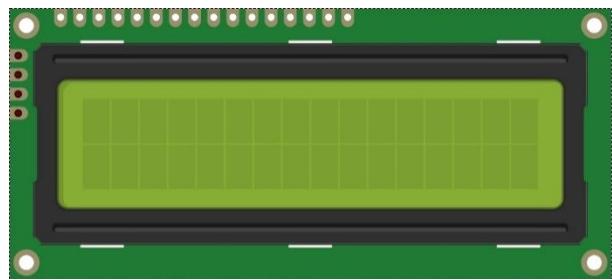


You can use the I2C LCD1602 to replace the serial port as a mobile screen when you print the data latter.

Project 14B.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.

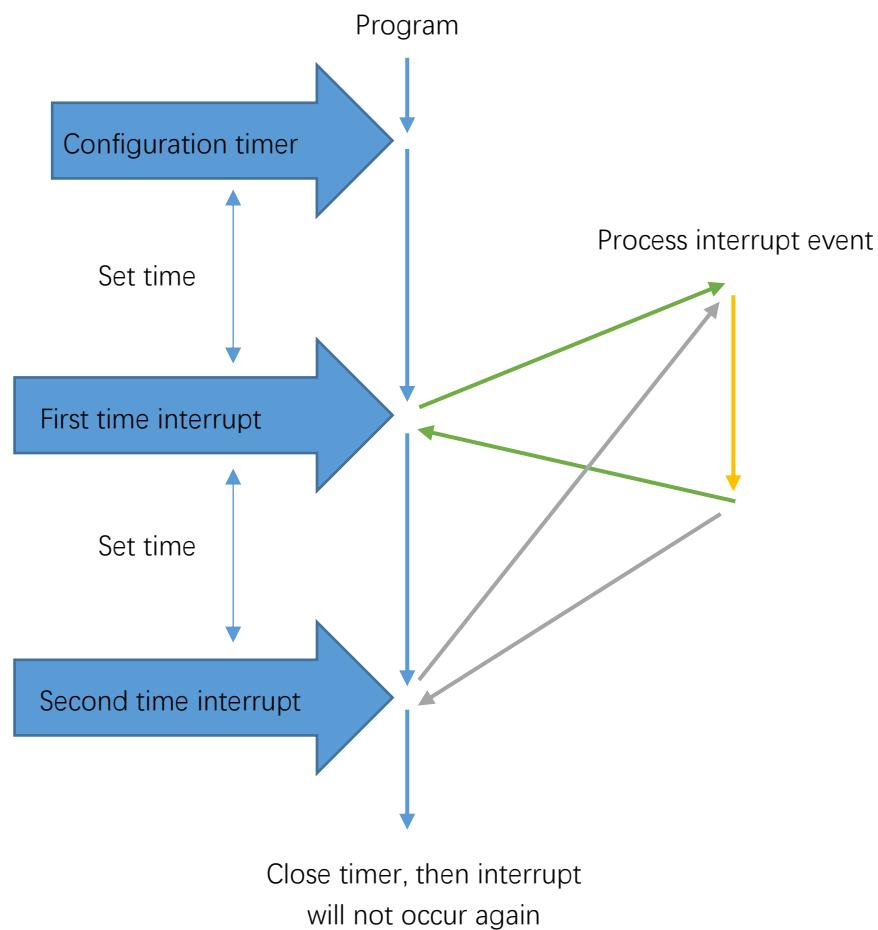
Component List

Control board x1	Breadboard x1	
		
USB cable x1	Jumper M/M x18	
		
I2C LCD1602 Module x1	Thermistor x1	Resistor 10kΩ x1
		

Code Knowledge

Timer

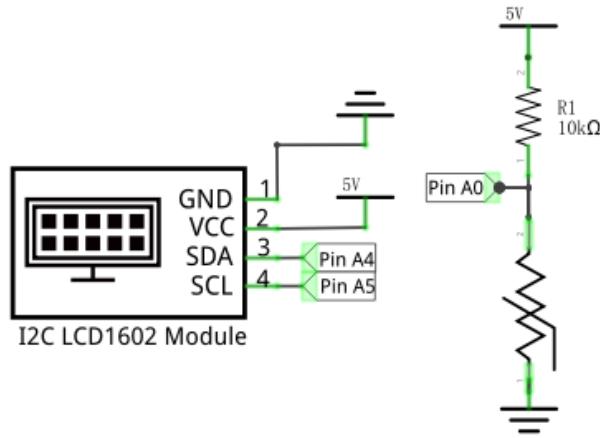
A Timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted location to go on. If you don't close the timer,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



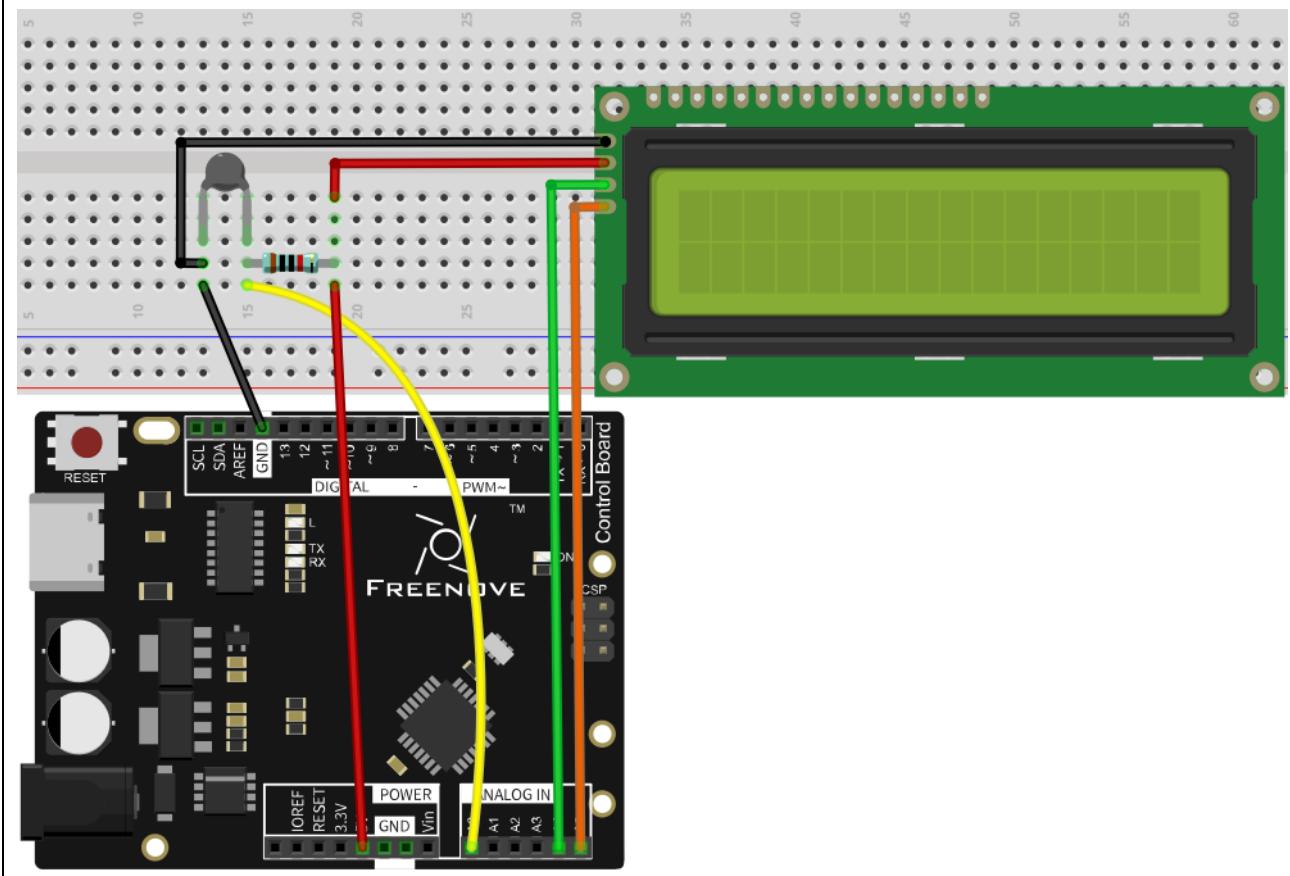
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 16.2.1

Before writing code, we need to import the library needed.

Click "Add .ZIP Library..." and then find FlexiTimer2.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library can help manipulate the timer.

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 <FlexiTimer2.h>      // Contains FlexiTimer2 Library
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 value
8 int hour, minute, second; // define variables stored record time
9
10 void setup() {
11     lcd.init();           // initialize the lcd
12     lcd.backlight();      // Turn on backlight
13     startingAnimation();  // display a dynamic start screen
14     FlexiTimer2::set(1000, timerInt); // configure the timer and interrupt function
15     FlexiTimer2::start();   // start timer
16     Serial.begin(115200);  // initialize serial port with baud rate 9600
17     Serial.println("Input hour, minute, second to set time.");
18 }
19
20 void loop() {
21     // Get temperature
22     tempVal = getTemp();
23     if (second >= 60) {    // when seconds is equal to 60, minutes plus 1
24         second = 0;
25         minute++;
26         if (minute >= 60) { // when minutes is equal to 60, hours plus 1
27             minute = 0;
28             hour++;
29             if (hour >= 24) { // when hours is equal to 24, hours turn to zero
30                 hour = 0;
31             }
32         }
33     }
34     lcdDisplay();          // display temperature and time information on LCD
35     delay(200);
```

```
36  }
37
38 void startingAnimation() {
39     for (int i = 0; i < 16; i++) {
40         lcd.scrollDisplayRight();
41     }
42     lcd.print("starting... ");
43     for (int i = 0; i < 16; i++) {
44         lcd.scrollDisplayLeft();
45         delay(300);
46     }
47     lcd.clear();
48 }
49 // the timer interrupt function of FlexiTimer2 is executed every 1s
50 void timerInt() {
51     second++;      // second plus 1
52 }
53 // serial port interrupt function
54 void serialEvent() {
55     int inInt[3]; // define an array to save the received serial data
56     while (Serial.available()) {
57         for (int i = 0; i < 3; i++) {
58             inInt[i] = Serial.parseInt(); // receive 3 integer data
59         }
60         // print the received data for confirmation
61         Serial.print("Your input is: ");
62         Serial.print(inInt[0]);
63         Serial.print(", ");
64         Serial.print(inInt[1]);
65         Serial.print(", ");
66         Serial.println(inInt[2]);
67         // use received data to adjust time
68         hour = inInt[0];
69         minute = inInt[1];
70         second = inInt[2];
71         // print the modified time
72         Serial.print("Time now is: ");
73         Serial.print(hour / 10);
74         Serial.print(hour % 10);
75         Serial.print(':');
76         Serial.print(minute / 10);
77         Serial.print(minute % 10);
78         Serial.print(':');
79         Serial.print(second / 10);
```

```

80     Serial.println(second % 10);
81 }
82 }
83 // function used by LCD1602 to display time and temperature
84 void lcdDisplay() {
85     lcd.setCursor(0, 0); // set the cursor to (0,0) (first column, first row).
86     lcd.print("TEMP: "); // display temperature information
87     lcd.print(tempVal);
88     lcd.print("C");
89     lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second row)
90     lcd.print("TIME: "); // display time information
91     lcd.print(hour / 10);
92     lcd.print(hour % 10);
93     lcd.print(':' );
94     lcd.print(minute / 10);
95     lcd.print(minute % 10);
96     lcd.print(':' );
97     lcd.print(second / 10);
98     lcd.print(second % 10);
99 }
100 // function used to get temperature
101 float getTemp() {
102     // Convert analog value of tempPin into digital value
103     int adcVal = analogRead(tempPin);
104     // Calculate voltage
105     float v = adcVal * 5.0 / 1024;
106     // Calculate resistance value of thermistor
107     float Rt = 10 * v / (5 - v);
108     // Calculate temperature (Kelvin)
109     float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));
110     // Calculate temperature (Celsius)
111     return tempK - 273.15;
112 }
```

In the code, we define 3 variables to represent time: second, minute, hour.

8	<code>int hour, minute, second; // define variables to store hour, minute, seconds</code>
---	-------------------------------------------------------------------------------------------

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 FlexiTimer2::set () function.

14	<code>FlexiTimer2::set(1000, timerInt); // configure timer and timer interrupt function</code>
15	<code>FlexiTimer2::start(); // start timer</code>

After every interrupt, the second plus 1.

50	<code>void timerInt() {</code>
51	<code> sec++; // second plus 1</code>
52	<code>}</code>

:: operator

"::" is scope operator. The function behind "::" belongs to the scope of the front. If we want to call the function defined in the FlexiTimer2 scope outside, we need to use the "::". It can be global scope operator, class scope operator and namespace scope operator. It is a namespace scope operator here. Because functions of FlexiTimer2 library is defined in the namespace of FlexiTimer2, so we can find them in FlexiTimer2 library file.

In the loop () function, the information on the LCD display will be refreshed at set intervals.

```
20 void loop() {
...
34     lcdDisplay();           // display temperature and time information on LCD
35     delay(200);
36 }
```

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.

```
23 if (second >= 60) {           // when the second increases to 60, the minute adds 1
24     second = 0;
25     minute++;
26     if (minute >= 60) {        //when the minute increases to 60, the hour adds 1
27         minute = 0;
28         hour++;
29         if (hour >= 24) {      // when the hour increases to 24, turn it to zero
30             hour = 0;
31         }
32     }
33 }
```

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.

```
84 void lcdDisplay() {
85     lcd.setCursor(0, 0); // set the cursor to (0,0) (first column, first row).
86     lcd.print("TEMP: "); // display temperature information
87     lcd.print(tempVal);
88     lcd.print("C");
89     lcd.setCursor(0, 1); // set the cursor to (0,1) (first column, second row)
90     lcd.print("TIME: "); // display time information
91     lcd.print(hour / 10);
92     lcd.print(hour % 10);
93     lcd.print(':' );
94     lcd.print(minute / 10);
95     lcd.print(minute % 10);
96     lcd.print(':' );
97     lcd.print(second / 10);
98     lcd.print(second % 10);
99 }
```

Serial port interrupt function is used to receive the data sent by computer to adjust the time, and return the data for confirmation.

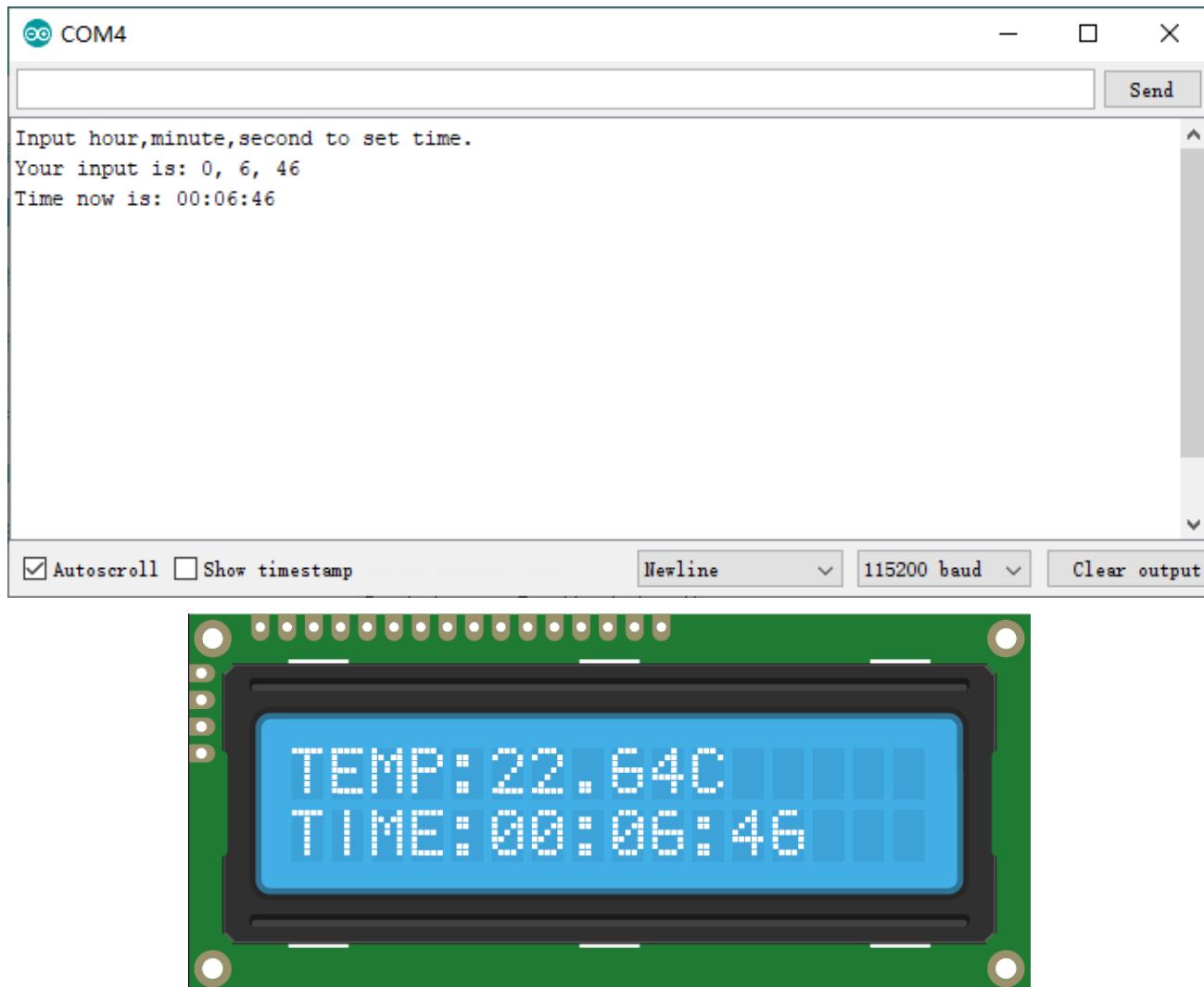
```
54 void serialEvent() {  
55     int inInt[3]; // define an array to save the received serial data  
56     while (Serial.available()) {  
57         for (int i = 0; i < 3; i++) {  
58             inInt[i] = Serial.parseInt(); // receive 3 integer data  
59         }  
60         // print the received data for confirmation  
61         Serial.print("Your input is: ");  
62         Serial.print(inInt[0]);  
63         Serial.print(", ");  
64         Serial.print(inInt[1]);  
65         Serial.print(", ");  
66         Serial.println(inInt[2]);  
67         // use the received data to adjust time  
68         hour = inInt[0];  
69         minute = inInt[1];  
70         second = inInt[2];  
71         // print the modified time  
72         Serial.print("Time now is: ");  
73         Serial.print(hour / 10);  
74         Serial.print(hour % 10);  
75         Serial.print(':' );  
76         Serial.print(minute / 10);  
77         Serial.print(minute % 10);  
78         Serial.print(':' );  
79         Serial.print(second / 10);  
80         Serial.println(second % 10);  
81     }  
82 }
```

We also define a function that displays a scrolling string when the control board has been just started.

```
38 void startingAnimation() {  
39     for (int i = 0; i < 16; i++) {  
40         lcd.scrollDisplayRight();  
41     }  
42     lcd.print("starting...");  
43     for (int i = 0; i < 16; i++) {  
44         lcd.scrollDisplayLeft();  
45         delay(300);  
46     }  
47     lcd.clear();  
48 }
```

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.



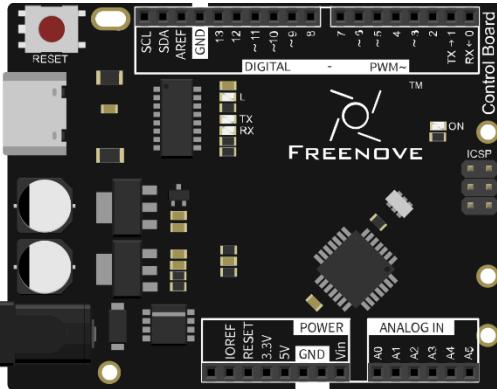
Chapter 15 Ultrasonic Ranging

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

Project 15.1 Ultrasonic Ranging

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

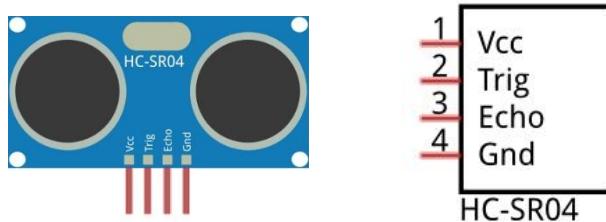
Component List

Control board x1	USB cable x1
 The Freenove Control Board is a breadboard-friendly expansion board for the Arduino Uno. It features a central ATmega328P microcontroller, various pins, and connectors. Key labels include 'RESET', 'IREF', '3.3V', '5V', 'POWER', 'GND', 'Vm', 'ANALOG IN' (A0-A5), and digital pins labeled 'DIGITAL' (pins 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0). It also includes 'TX', 'RX', 'PWM', and 'ICSP' headers.	 A standard black USB cable with A and B type connectors.

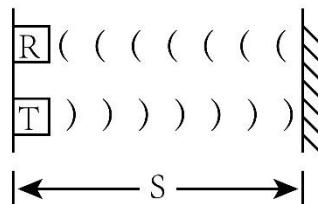
Component Knowledge

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:

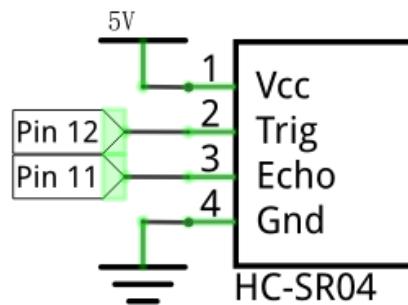
Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Trigger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

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.

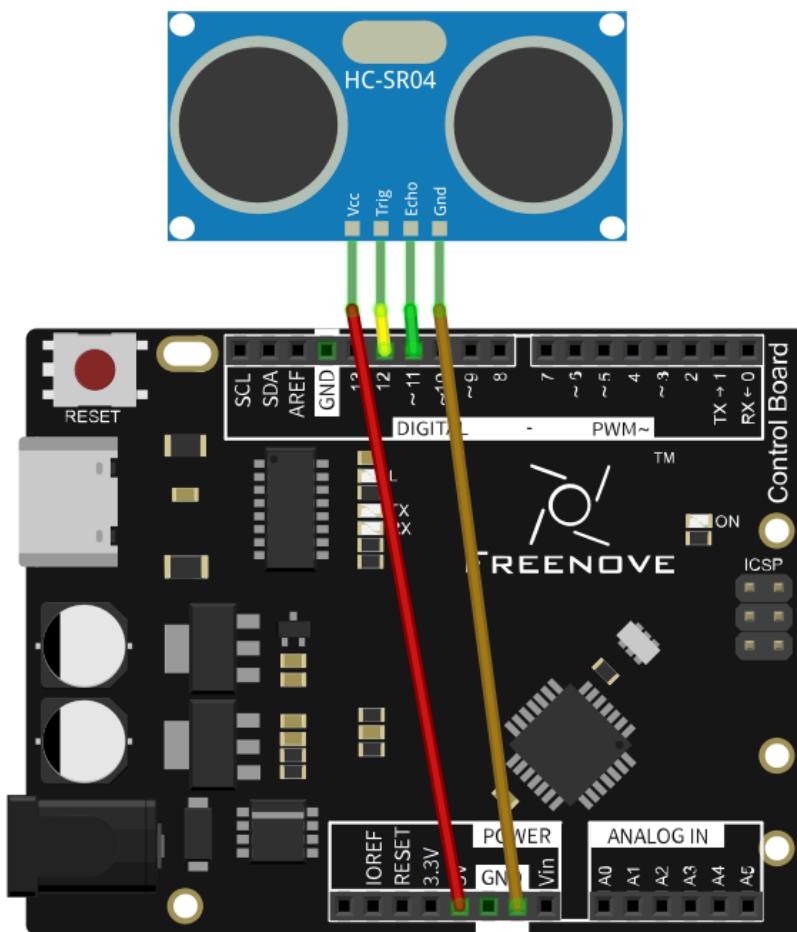
Circuit

The connection of the control board and HC-SR04 is shown below.

Schematic diagram



Hardware connection



Sketch

Sketch 14.1.1

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 10 µs 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.

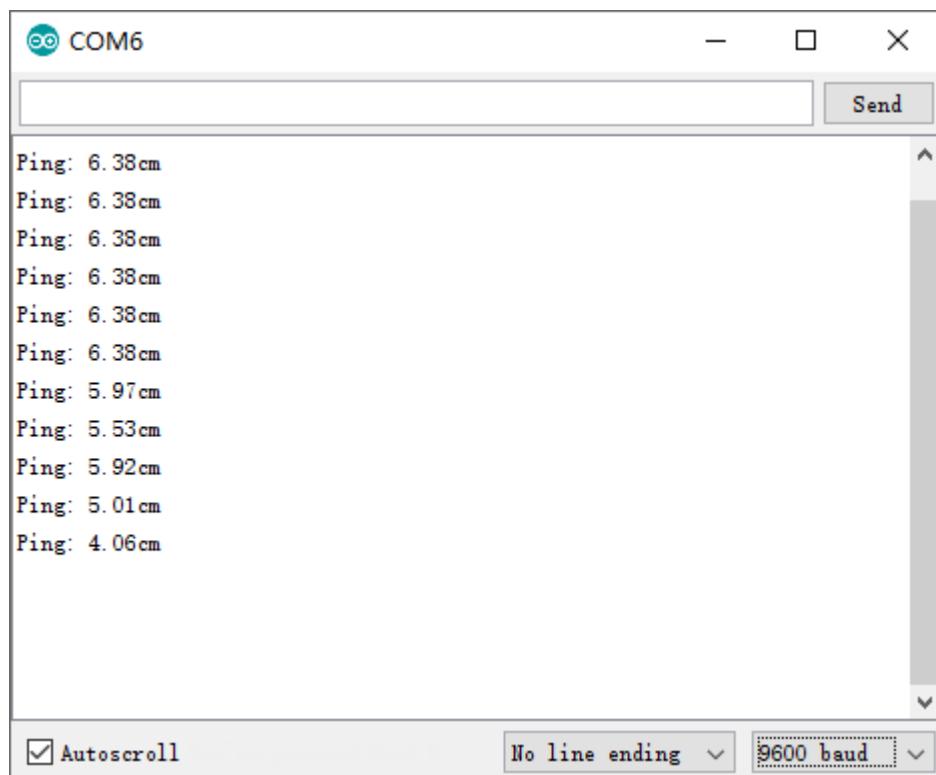
```
28 distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance  
according to the time  
29 return distance; // return the distance value
```

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).

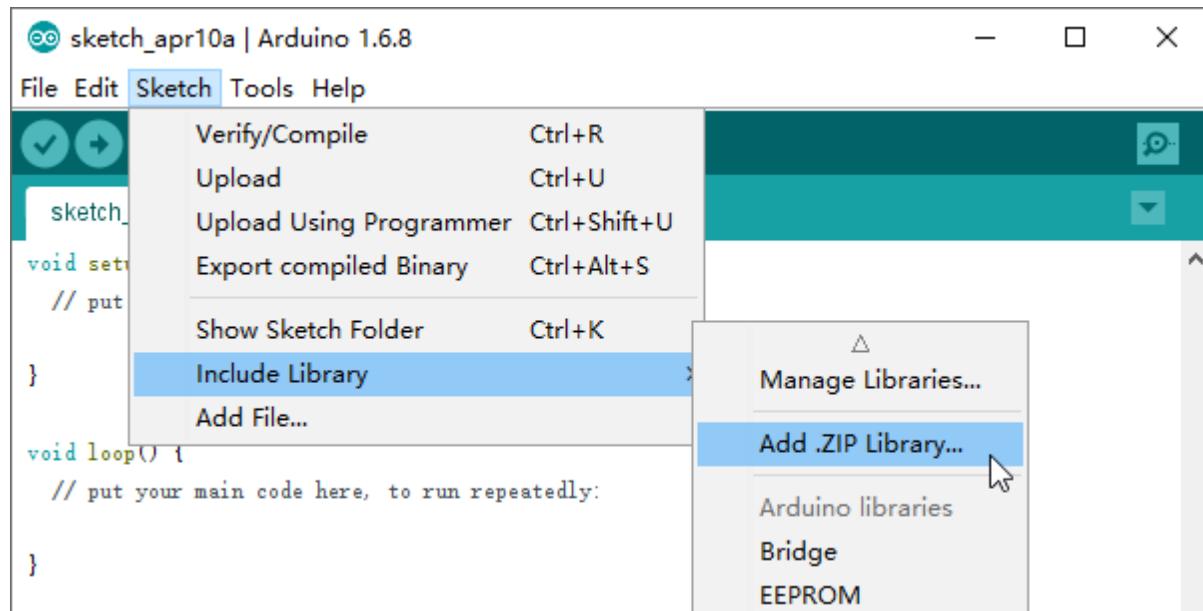
Verify and upload the code to the control board, open the serial port monitoring window, turn the HC-SR04 probe towards the object plane, and observe the data in the serial port monitoring window.



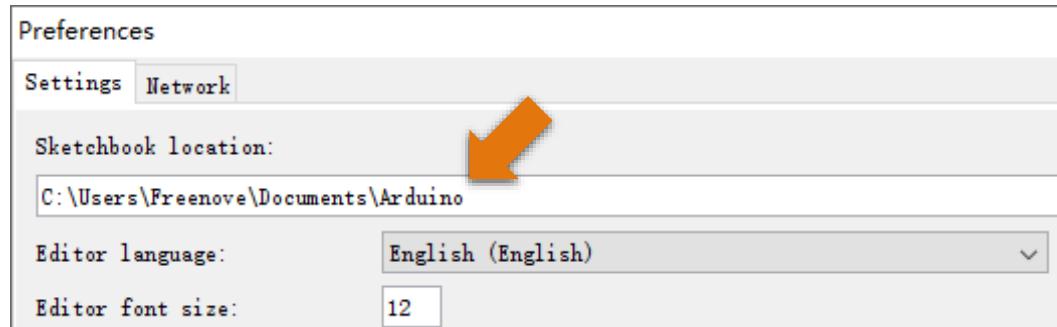
Sketch 14.1.2

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

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 make it easy to obtain the measuring distance.



When a library is added, you can find it in the libraries folder under Sketchbook location in the File -> Preferences window. You can view its source code to understand their specific usage.



```

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:

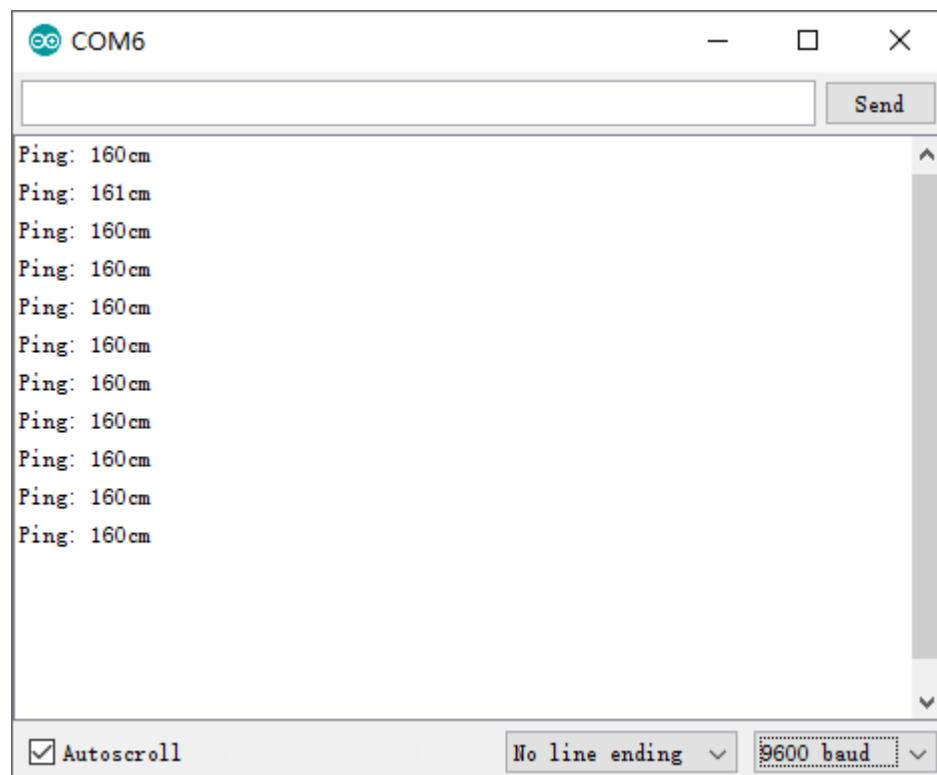
sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

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.



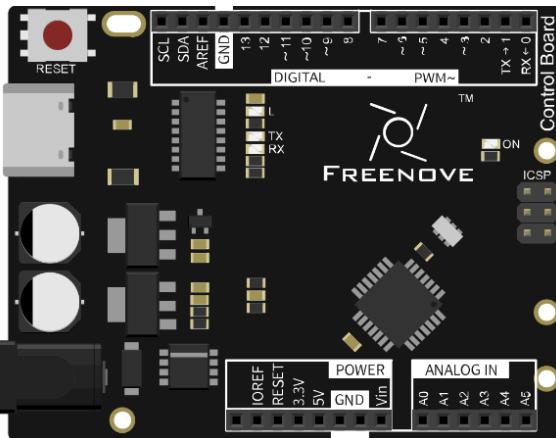
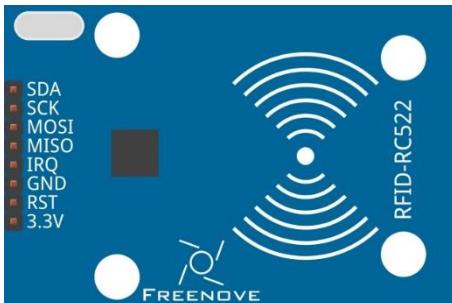
Chapter 16 RFID

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

Project 16 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	RFID Module (RC522) x1
	
Mifare1 S50 Standard card x1	Mifare1 S50 Non-standard card x1
USB able x1	Jumper M/F x7

Circuit 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.

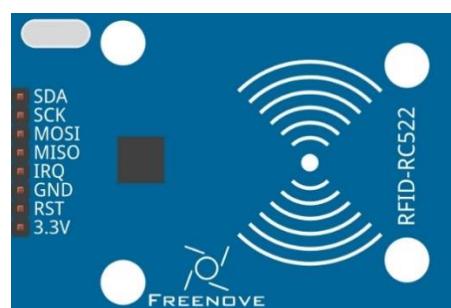
Component Knowledge

MFRC522 RFID Module

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

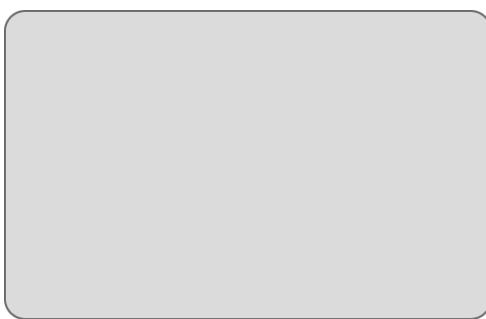
The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 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.

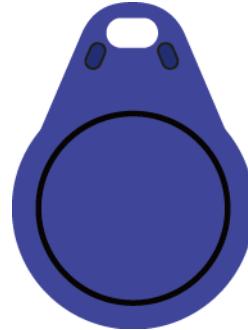


Mifare1 S50 Card

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 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 0	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.

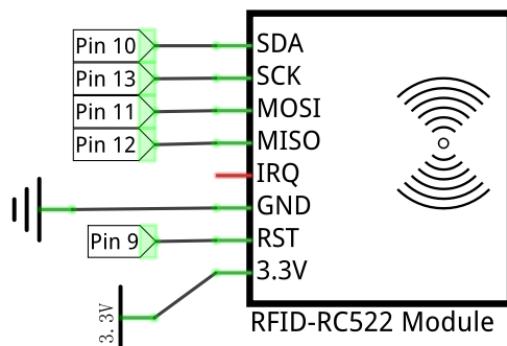
For more details about how to set data blocks and control blocks, please refer to Datasheet.

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.**

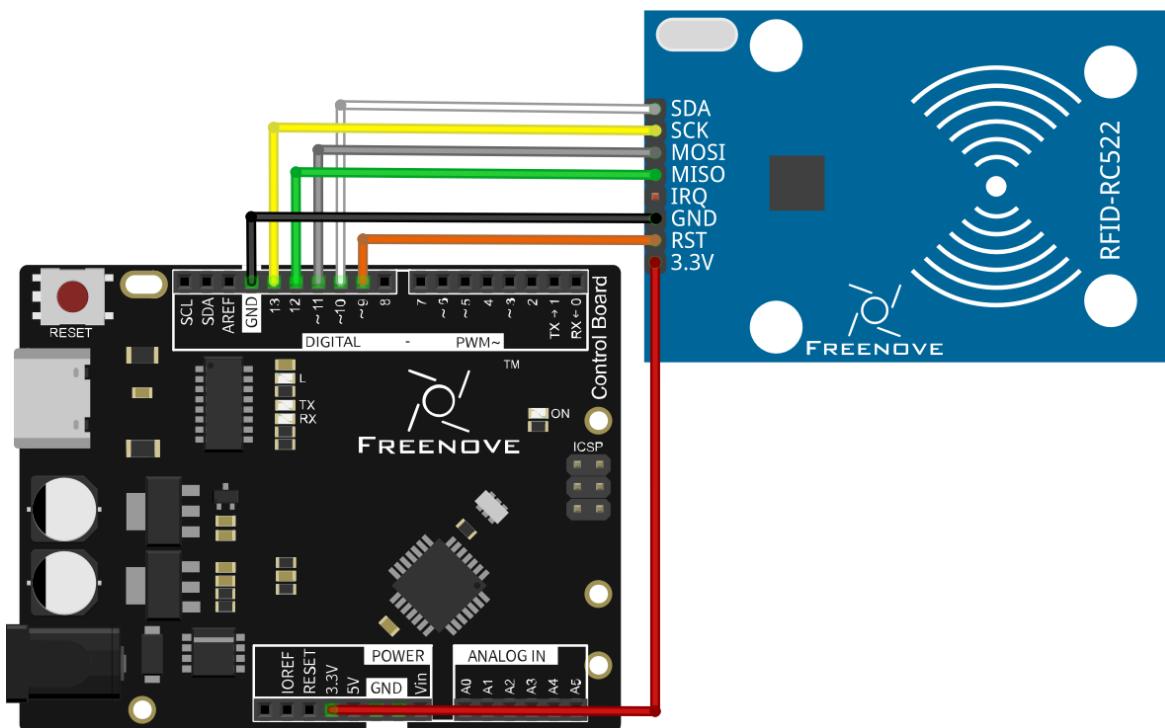
Circuit

The connection of control board and RFID module is shown below.

Schematic diagram



Hardware connection



Sketch

Sketch 23.1.1

Before writing code, we need to import the library needed.

Click “Add .ZIP Library...” and then find RFID.zip in libraries folder (this folder is in the folder unzipped 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 //D10:pin of card reader SDA. D9:pin of card reader RST
5 RFID rfid(10, 9);
6 unsigned char status;
7 unsigned char str[MAX_LEN]; //MAX_LEN is 16: size of the array
8
9 void setup()
10 {
11     Serial.begin(9600);
12     SPI.begin();
13     rfid.init(); //initialization
14     Serial.println("Please put the card to the induction area...");
15 }
16
17 void loop()
18 {
19     //Search card, return card types
20     if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
21         Serial.println("Find the card!");
22         // Show card type
23         ShowCardType(str);
24         //Anti-collision detection, read card serial number
25         if (rfid.anticoll(str) == MI_OK) {
26             Serial.print("The card's number is : ");
27             //Display card serial number
28             for (int i = 0; i < 4; i++) {
29                 Serial.print(0x0F & (str[i] >> 4), HEX);
30                 Serial.print(0x0F & str[i], HEX);
31             }
32             Serial.println("");
33         }
34         //card selection (lock card to prevent redundant read, removing the line will make
the sketch read cards continually)
```

```

35     rfid.selectTag(str);
36 }
37 rfid.halt(); // command the card to enter sleeping state
38 }
39 void ShowCardType(unsigned char * type)
40 {
41     Serial.print("Card type: ");
42     if (type[0] == 0x04 && type[1] == 0x00)
43         Serial.println("MFOne-S50");
44     else if (type[0] == 0x02 && type[1] == 0x00)
45         Serial.println("MFOne-S70");
46     else if (type[0] == 0x44 && type[1] == 0x00)
47         Serial.println("MF-UltraLight");
48     else if (type[0] == 0x08 && type[1] == 0x00)
49         Serial.println("MF-Pro");
50     else if (type[0] == 0x44 && type[1] == 0x03)
51         Serial.println("MF Desire");
52     else
53         Serial.println("Unknown");
54 }
```

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.

```
5   RFID rfid(10, 9);
```

In setup, initialize the serial port, SPI and RFID.

```

11   Serial.begin(9600);
12   SPI.begin();
13   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. Then enter the if statement.

```
20   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.

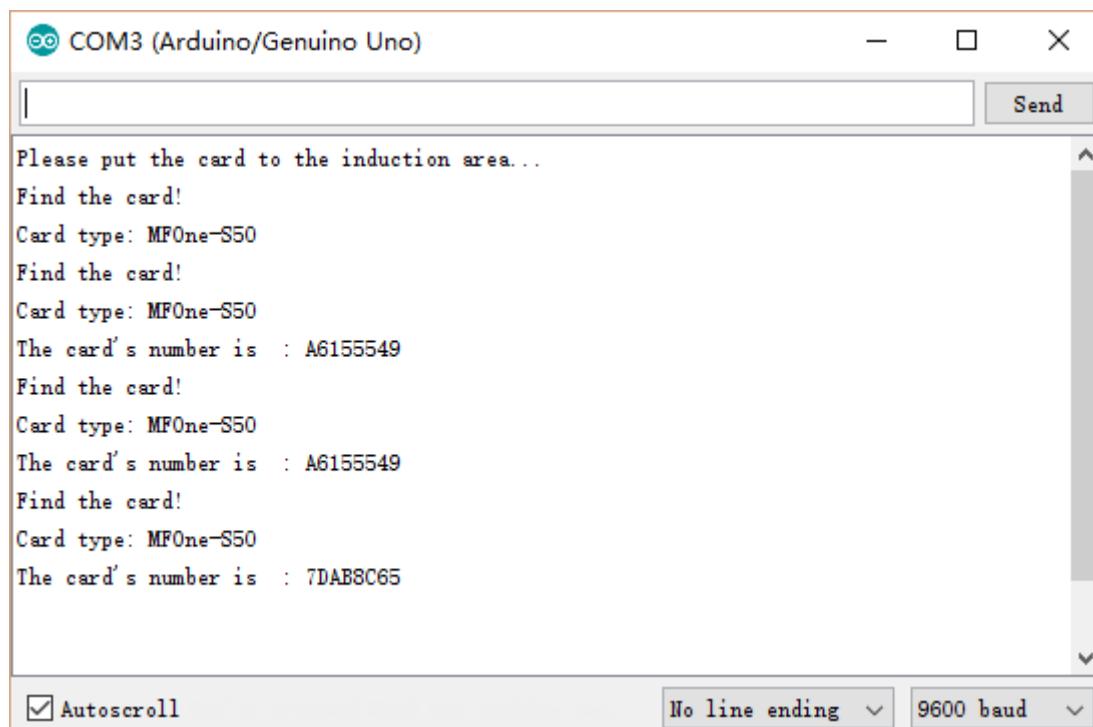
```
23   ShowCardType(str);
```

Then use the.anticoll() to read UID of the card and use serial port to print it out.

```

25   if (rfid.anticoll(str) == MI_OK) {
26       Serial.print("The card's number is : ");
27       //Display card serial number
28       for (int i = 0; i < 4; i++) {
29           Serial.print(0x0F & (str[i] >> 4), HEX);
30           Serial.print(0x0F & str[i], HEX);
31       }
32       Serial.println("");
33 }
```

After verifying and uploading the code to control board, open the serial port monitor and make a card approach the sensing area of RFID module. Then serial port monitoring window 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.



Project 16.2 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 23.2.1

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 // D10:pin of card reader SDA. D9: 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(9600);  
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 }
```

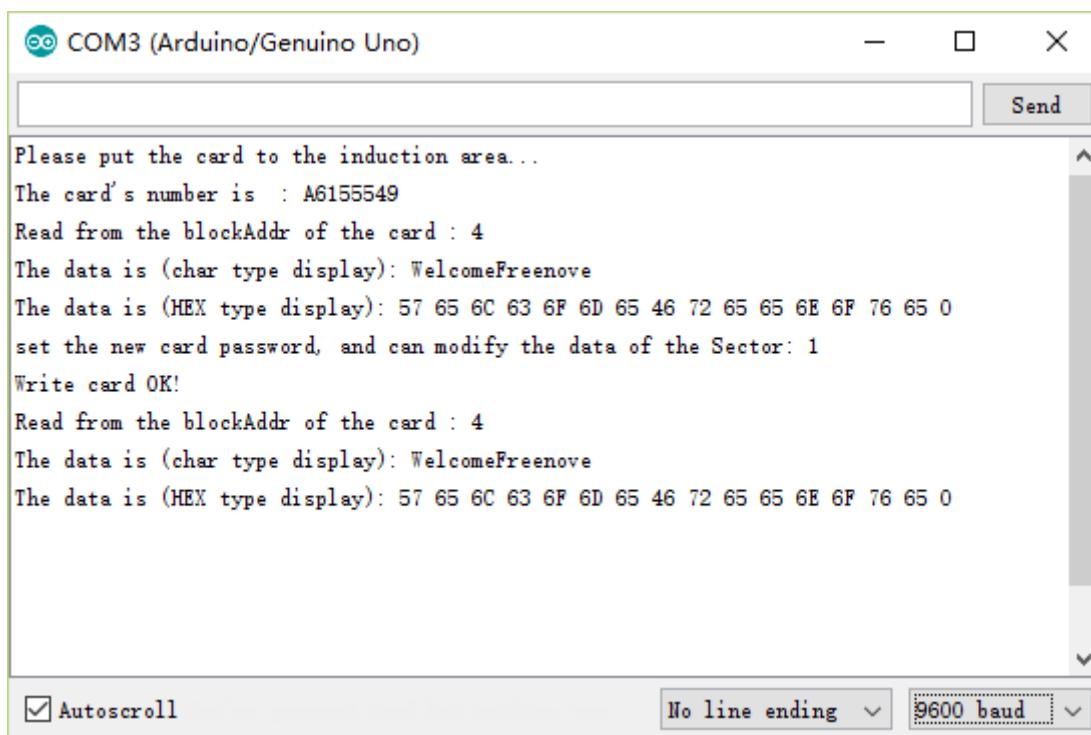
In the sub function of writeCard () and readCard (), we must first verify the password A, and then use the corresponding sub function to read and write. Here we do reading and writing operations to data block 0 (absolute NO.4) of the first sector.

```
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 to control board, open the serial port monitor and make a card approach the sensing area of RFID module, then the serial port monitoring window will display displacement ID numbers of the card, the type of this card and the contents (before and after writing operation) of data block. If the induction time is too short, it may lead to incomplete information display.



What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this tutorial. If you find errors, omissions or you have suggestions and/or questions about 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 colored 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 colored bands. The first four bands represent the resistance value (e.g., 237), and the fifth band represents the tolerance (e.g., ±1%).

Below the resistors, a table maps colors to their corresponding values and tolerances:

COLOR	1 ST BAND	2 ND BAND	3 RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	± 1% (F)
Brown	1	1	1	10Ω	± 2% (G)
Red	2	2	2	100Ω	
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 row:

- Row 1: 0.1%, 0.25%, 0.5%, 1% (corresponding to the first four bands of a 5-band resistor).
- Row 2: 237 Ω ± 1% (corresponding to the first four bands of a 5-band resistor).