FREENOVE

Free your innovation

Freenove is an open-source electronics platform.
www.freenove.com

Warning

When you purchase or use this product, please note the following:

- This product contains small parts. Swallowing or improper operation them can cause serious infections and death. Seek immediate medical attention when the accident happened.
- Do not allow children under 3 years old to play with or near this product. Please place this product in where children under 3 years of age cannot reach.
- Do not allow children lack of ability of safe to use this product alone without parental care.
- Never use this product and its parts near any AC electrical outlet or other circuits to avoid the potential risk of electric shock.
- Never use this product near any liquid and fire.
- Keep conductive materials away from this product.
- Never store or use this product in any extreme environments such as extreme hot or cold, high humidity and etc.
- Remember to turn off circuits when not in use this product or when left.
- Do not touch any moving and rotating parts of this product while they are operating.
- Some parts of this product may become warm to touch when used in certain circuit designs. This is normal. Improper operation may cause excessively overheating.
- Using this product not in accordance with the specification may cause damage to the product.

About

Freenove is an open-source electronics platform. Freenove is committed to helping customer quickly realize the creative idea and product prototypes, making it easy to get started for enthusiasts of programing and electronics and launching innovative open source products. Our services include:

- Electronic components and modules
- Learning kits for Arduino
- Learning kits for Raspberry Pi
- Learning kits for Technology
- Robot kits
- Auxiliary tools for creations

Our code and circuit are open source. You can obtain the details and the latest information through visiting the following web sites:
http://www.freenove.com
https://github.com/freenove

Your comments and suggestions are warmly welcomed, please send them to the following email address:
support@freenove.com

References

You can download the sketches and references used in this product in the following websites:
http://www.freenove.com
https://github.com/freenove
If you have any difficulties, you can send email to technical support for help.

The references for this product is named Freenove RFID Starter Kit for Arduino, which includes the following folders and files:

- Datasheet Datasheet of electronic components and modules
- Libraries Library files of Arduino Software
- Sketches Code of Arduino Software
- Readme.txt Instructions

Support

Freenove provides free and quick technical support, including but not limited to:

- Quality problems of products
- Problems in using products
- Questions for learning and technology
- Opinions and suggestions
- Ideas and thoughts

Please send email to:
support@freenove.com
On working day, we usually reply to you within 24 hours.

Copyright

Freenove reserves all rights to this book. No copies or plagiarizations are allowed for the purpose of commercial use.
The code and circuit involved in this product are released as Creative Commons Attribution ShareAlike 3.0. This means you can use them on your own derived works, in part or completely, as long as you also adopt the same license. Freenove brand and Freenove logo are copyright of Freenove Creative Technology Co., Ltd and cannot be used without formal permission.

Contents

Preface 1
Arduino Board 1
Arduino Software 4
First Use 7
Chapter 1 LED Blink 11
Project 1.1 Control LED by Manual Button 11
Project 1.2 Control LED by Arduino 18
Chapter 2 Two LEDs Blink 24
Project 2.1 Two LEDs Blink 24
Chapter 3 LED bar graph 31
Project 3.1 LED bar graph Display 31
Chapter 4 LED Blink Smoothly 38
Project 4.1 LEDs Emit Different Brightness 38
Project 4.2 LED Blink Smoothly 43
Chapter 5 Control LED Through Push Button 45
Project 5.1 Control LED Through Push Button 45
Project 5.2 Change LED State by Push Button 49
Chapter 6 Serial 51
Project 6.1 Send data through Serial 51
Project 6.2 Receive Data through Serial Port 56
Project 6.3 Application of Serial 60
Chapter 7 ADC 63
Project 7.1 ADC 63
Project 7.2 Control LED by Potentiometer 68
Project 7.3 Control LED through Photoresistor 71
Chapter 8 RGB LED 75
Project 8.1 Control RGB LED through Potentiometer 75
Project 8.2 Colorful LED 79
Chapter 9 Buzzer 82
Project 9.1 Active Buzzer 82
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 by L293D 101
Chapter 12 Servo 107
Project 12.1 Servo Sweep 107
Project 12.2 Control Servo by Potentiometer 111
Chapter 13 LED Matrix 114
Project 13.1 74HC595 114

Project 13.2 LED Matrix 120
Chapter 14 LCD1602 129
Project 14.1 Display the string on LCD1602 129
Project 14.2 LCD1602 Clock 134
Chapter 15 Ultrasonic Ranging 144
Project 15.1 Ultrasonic Ranging 144
Chapter 16 RFID 152
Project 16.1 Read UID 152
Project 16.2 Read and write 160
What's next? 164
Appendix 165
ASCII table 165
Resistor color code 166

Preface

If you want to create somethings fun or make your great ideas real, using this product will be a good start. Maybe you have heard of Arduino before. If not, it doesn't matter. You can easily create dozens of interesting projects, and gradually realize the fun of using Arduino to complete creative works by reference to this book.

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects. Arduino has a lot of fans in the world. They contributed a lot of high quality open source code and circuit, so we can use these free code and the circuit to realize our own creativity more efficiently.

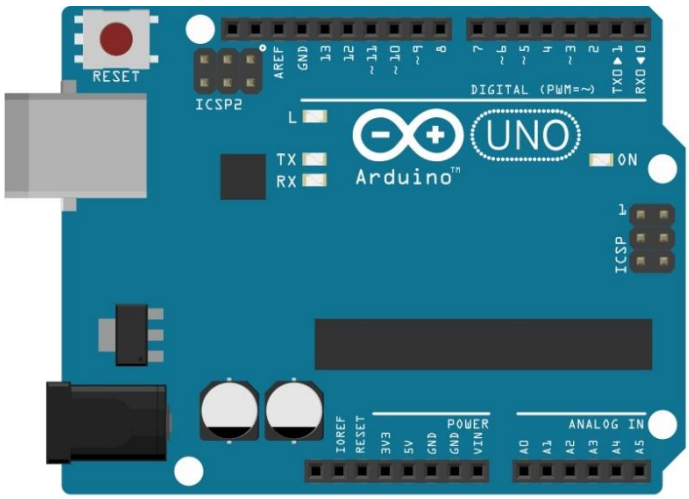
Usually, an Arduino project consists of code and circuit. If you never dealt with code and circuit before, don't worry. Because this book will gradually introduce C programming language and basic knowledge of electronic circuit from easy to difficult. And this product contains all electronic components and modules needed to complete these projects. You can also ask us for fast and free technical support at any time. It is especially suitable for beginners.

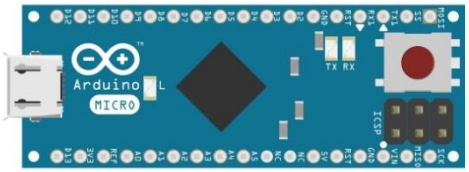
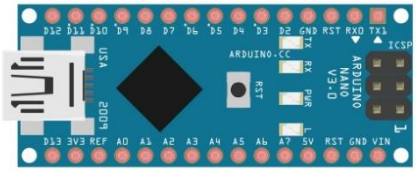
Arduino Board

Arduino Board is a circuit board, which integrates micro controller, input, output interface and etc. Arduino Board can use the sensor to sense the environment and receive user's operation to control LED, motor rotation, etc. We just need to assembly circuit and write the code.

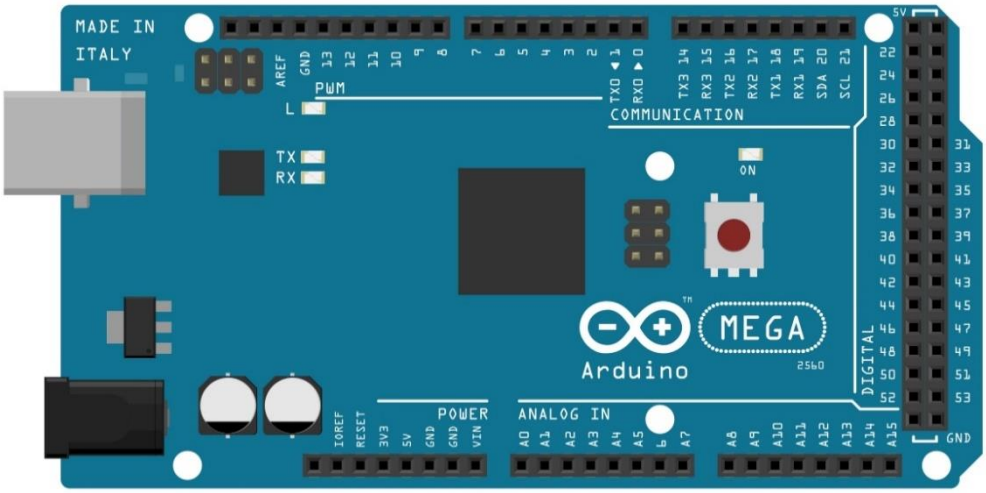
Currently, Arduino Board has several models, and the code between boards of different types is universal (some boards may not be completely compatible because of the differences in hardware). Popular boards include:

UNO

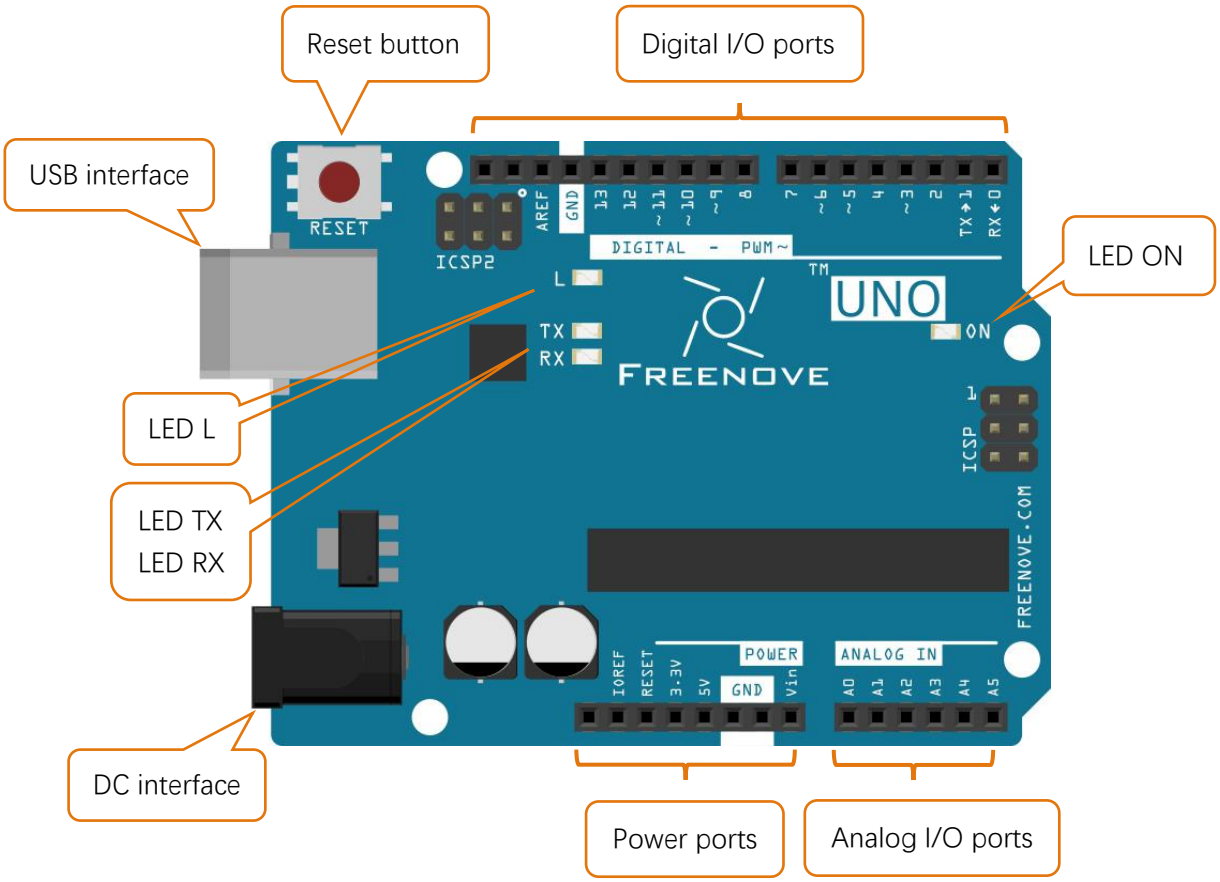


MICRO	NANO
	

MEGA



The board used in this book is Freenove UNO, and it is fully compatible to Arduino UNO. Diagram of Freenove UNO board is shown below:



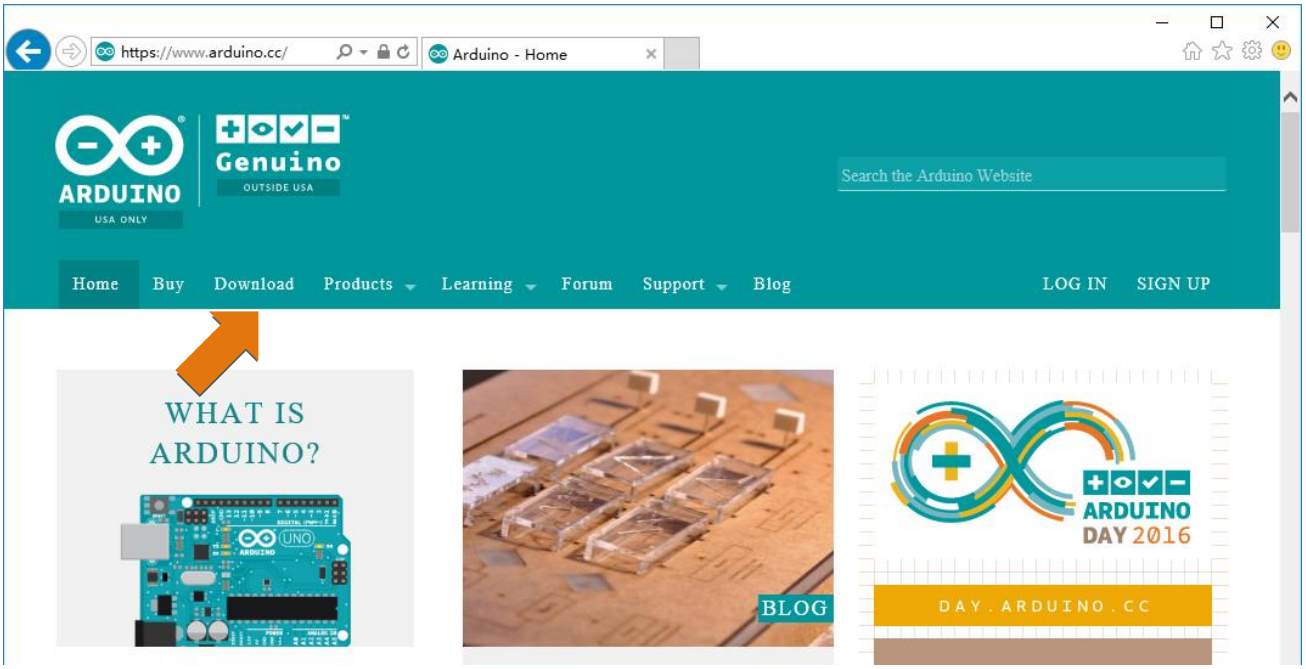
- Digital I/O ports is used to connect to other components or modules, to receive an input signal, or to send a control signal. Usually, we name it by adding a "D" in front of the number, such as D13.
- USB interface is used to provide power, upload code or communicate with PC.
- LED L is connected to digital I/O port 13 (D13).
- 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.

Freenove UNO is the most suitable board to complete the experiments of this book. You can also choose to use Arduino UNO, Arduino MICRO, Arduino NANO, Arduino MEGA (or other boards compatible to them).

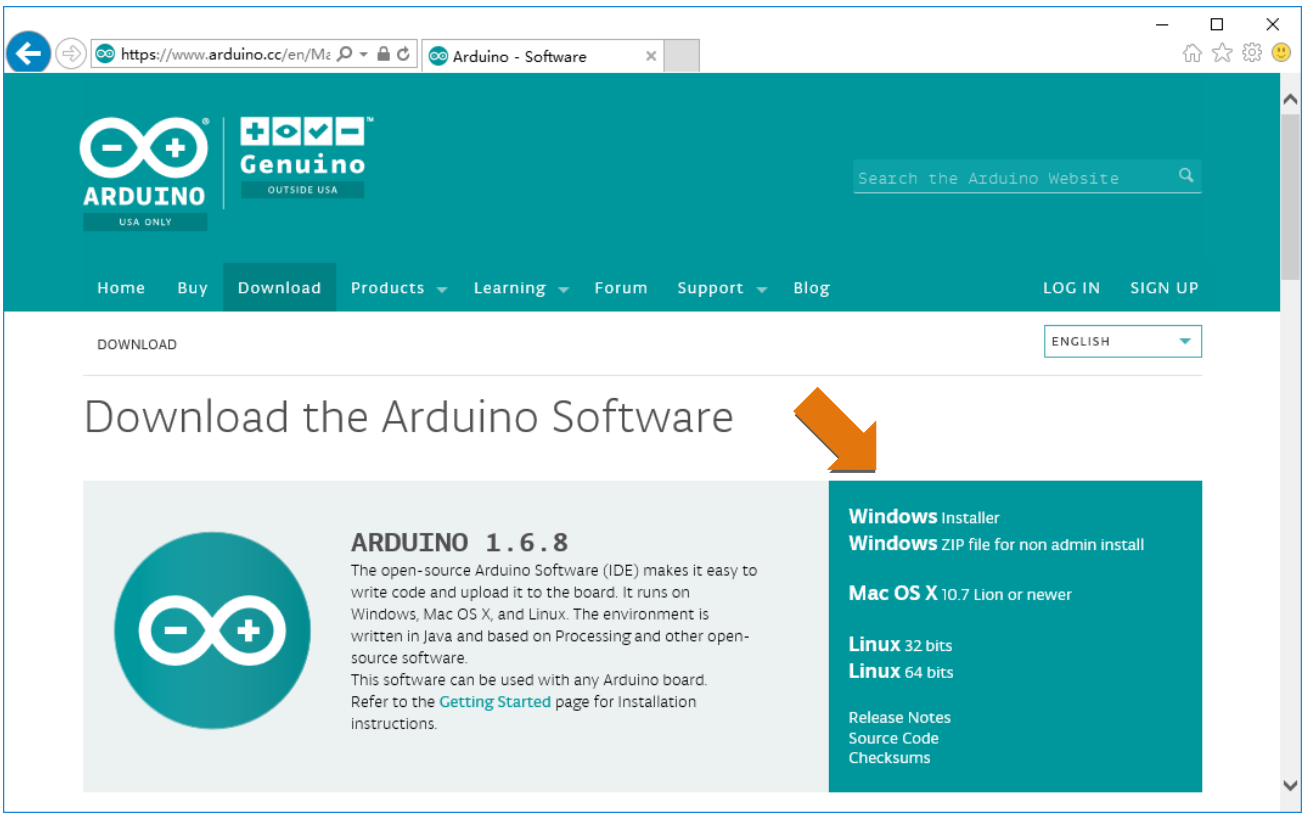
Arduino Software

Arduino Software (IDE) is used to write and upload the code for Arduino Board.

First, install Arduino Software (IDE): visit https://www.arduino.cc, click "Download" to enter the download page.

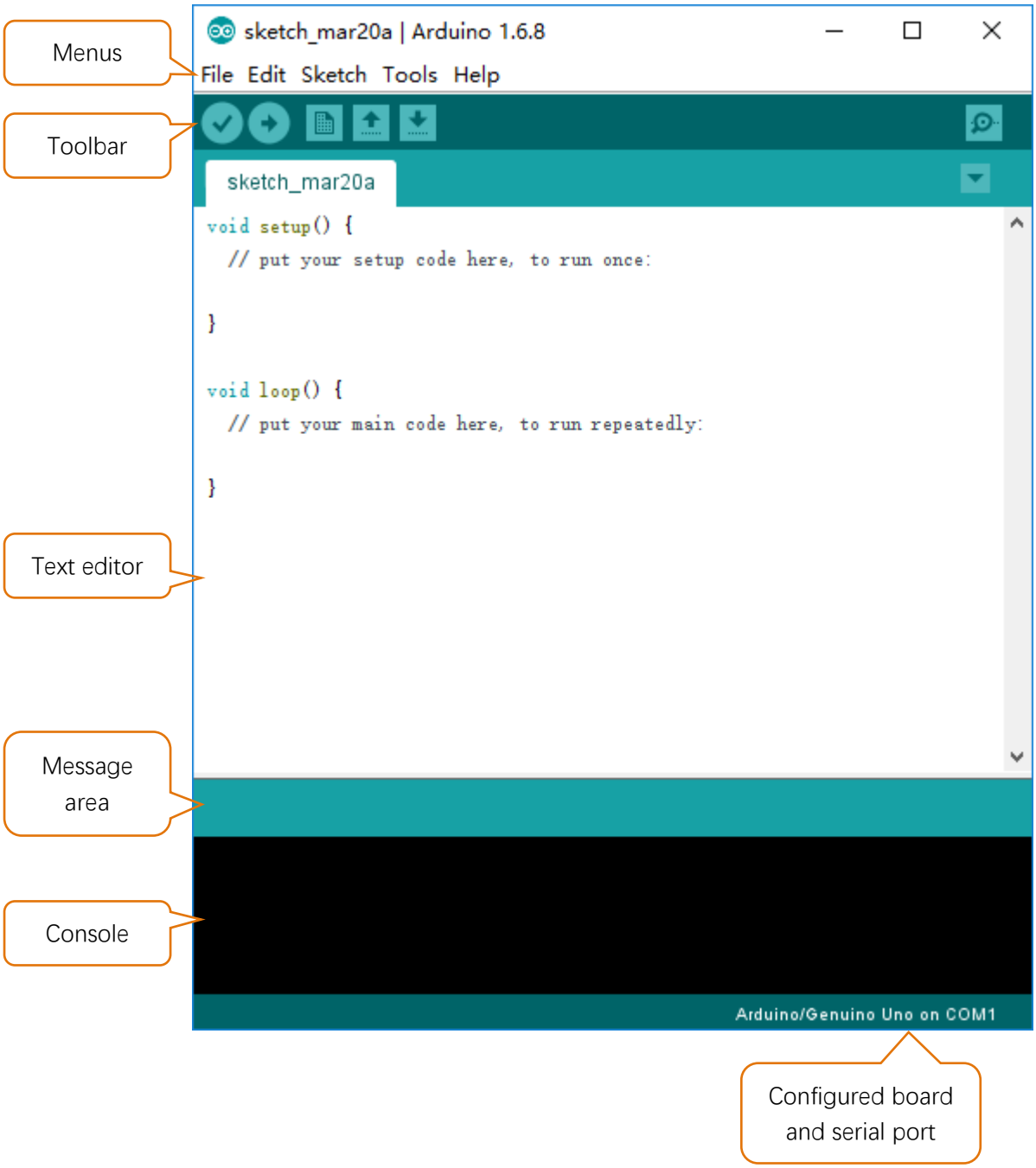


Download corresponding installation program according to your operating system. If you are a Windows user, please select the "Windows Installer" to download to install the driver correctly.



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

The interface of Arduino Software is as follows:



Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the 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 Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Verify
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.

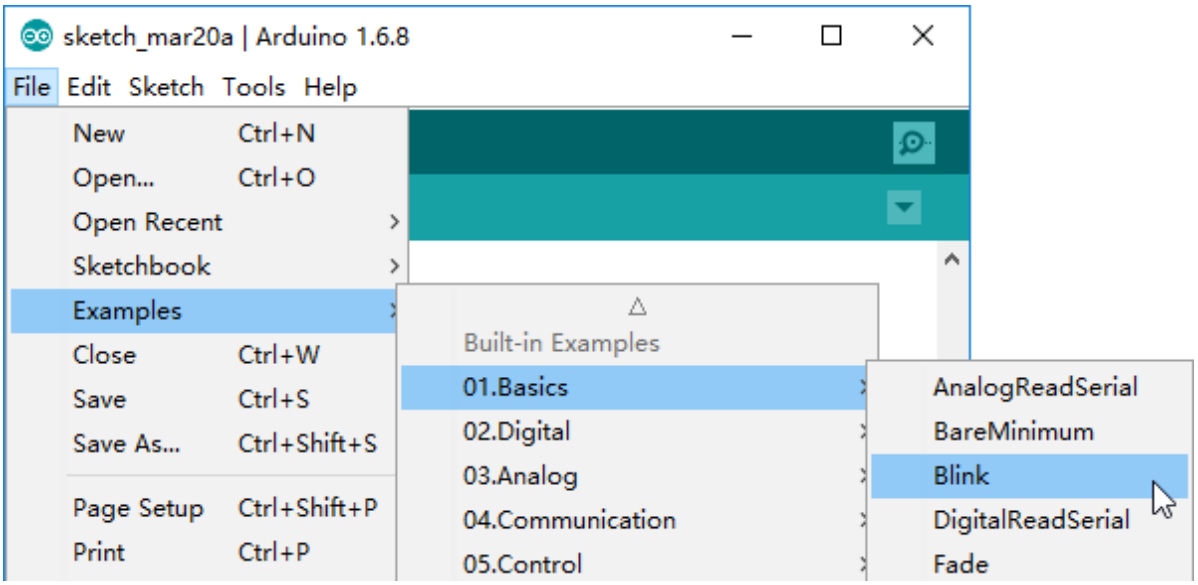
Save
Saves your sketch.

Serial Monitor
Opens the serial monitor.

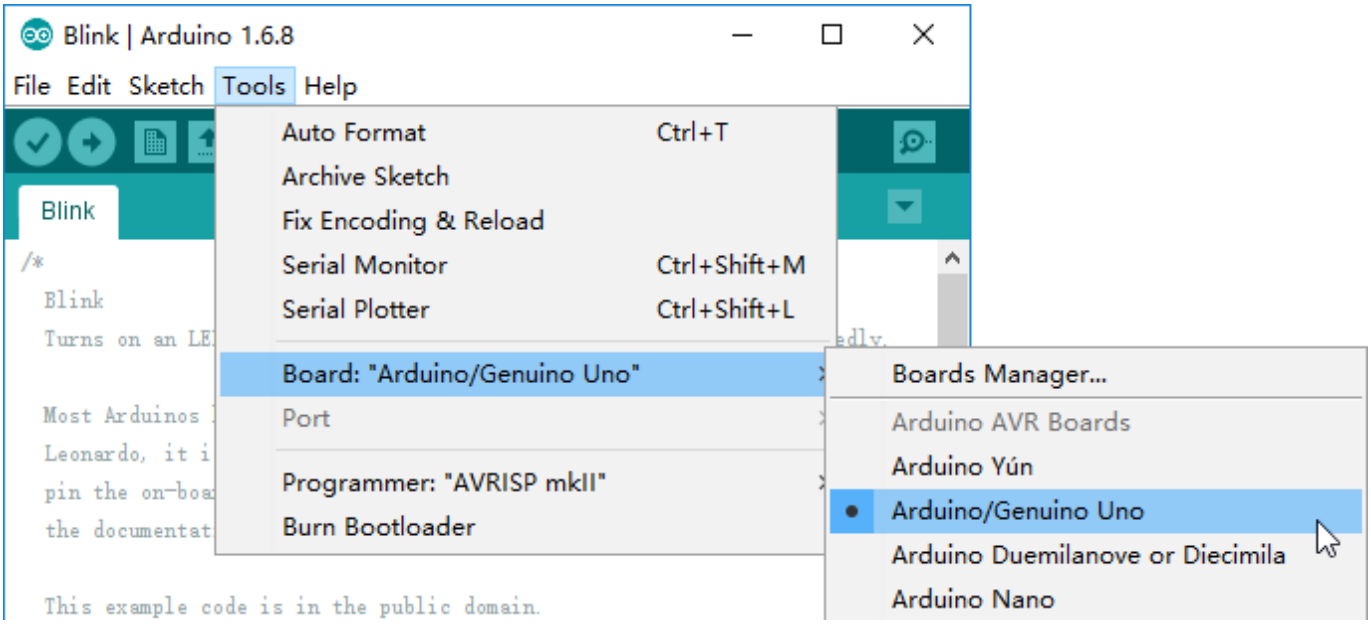
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

First Use

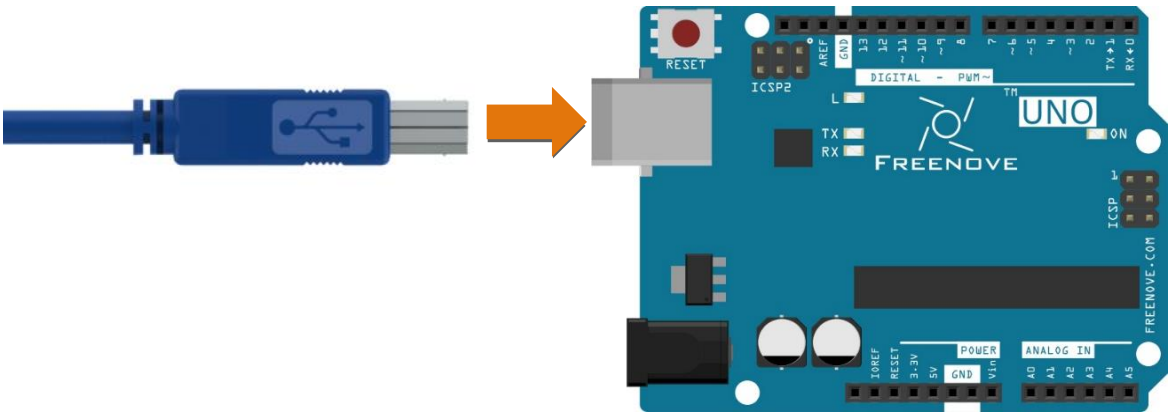
Open the example sketch "Blink" with Arduino Software.



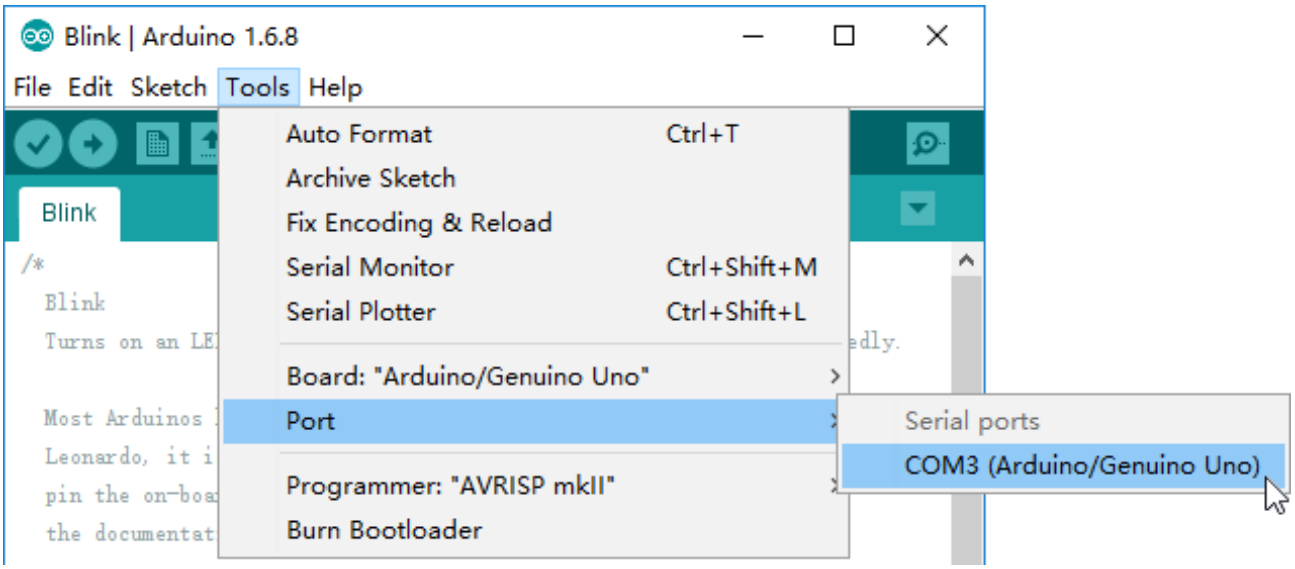
Select board "Arduino/Genuino Uno".



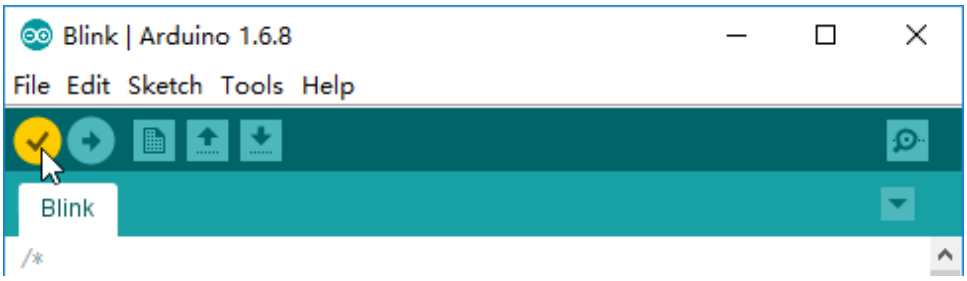
Connect Freenove UNO to computer with USB cable.



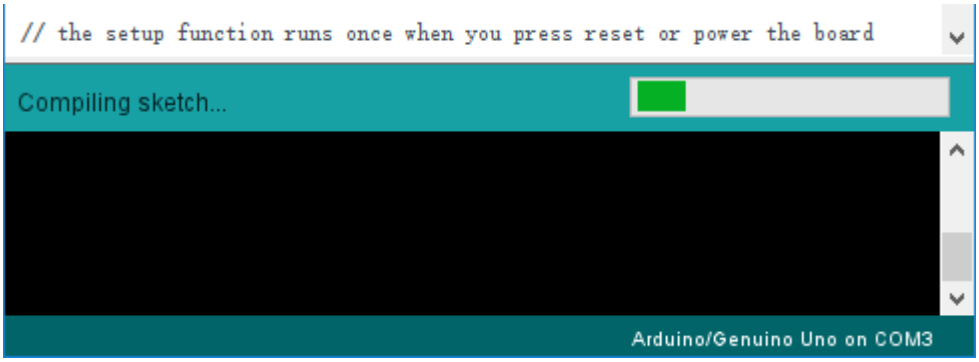
Select the serial port. Your serial number may be different from the following figure. If it is not detected immediately, please wait for a while, then click "Tools" to check again.



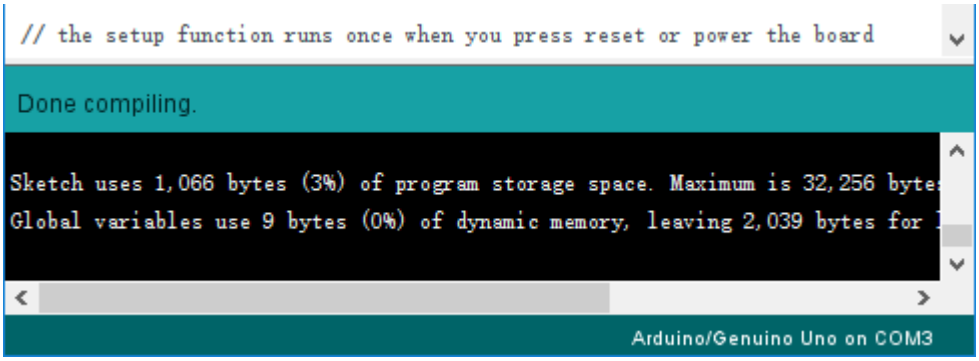
Click "Verify" button.



The following figure shows the code is being compiled.



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



Usually, when we write code, if it has a syntax error, the interface will prompt the error message. Then the compiling can't be completed.

Click "Upload" button.

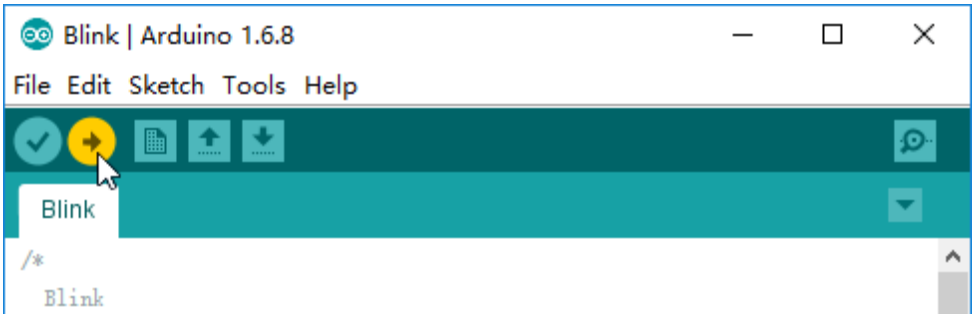
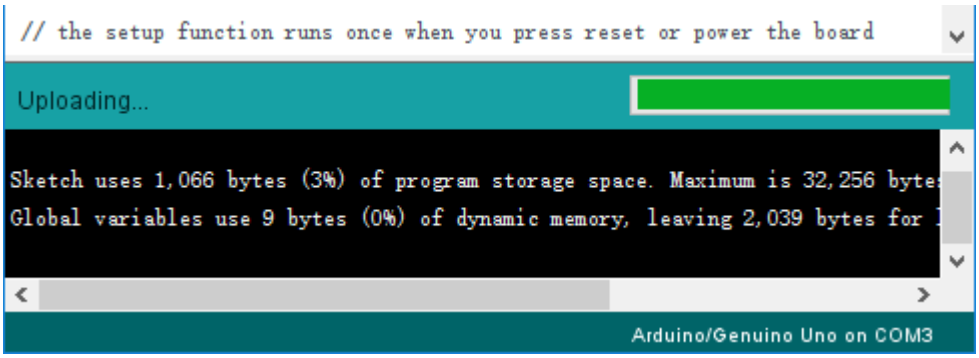
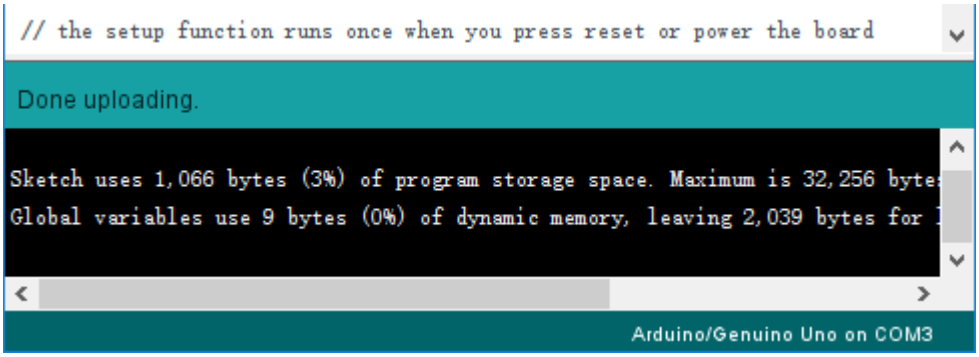


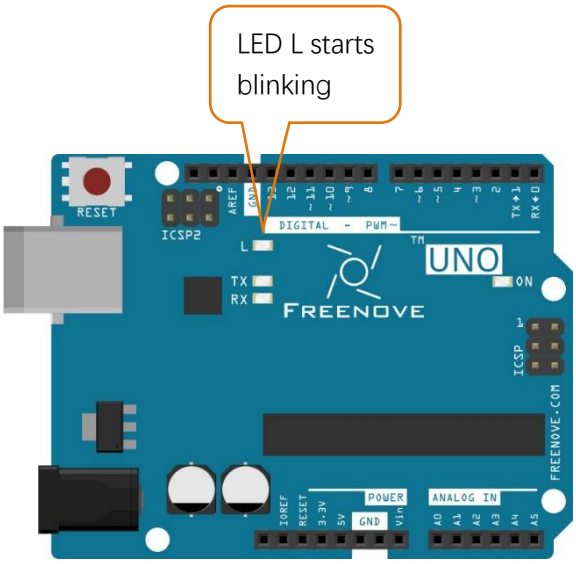
Figure below shows code are uploading.



Wait a moment, then the uploading is completed.



After that, we will see the LED marked with "L" on Freenove UNO starts 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 the Arduino programming and the building of electronic circuit.

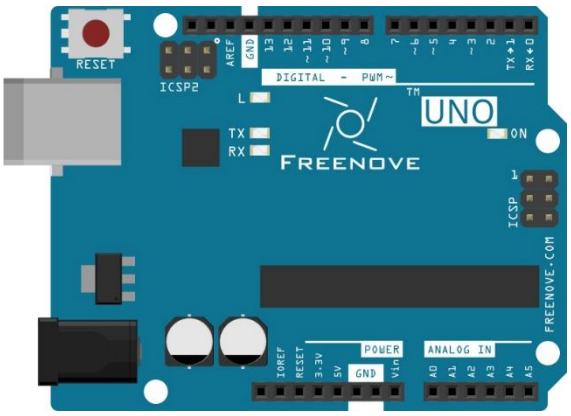
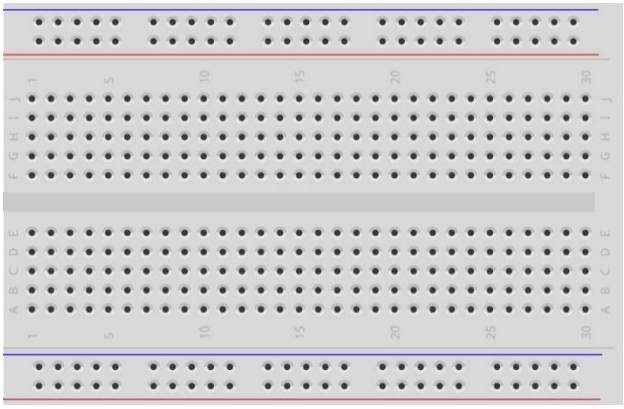
Chapter 1 LED Blink

We have previously tried to make the LED marked with "L" blink on Freenove UNO board. Now let us use electronic components and code to reproduce the phenomenon, and try to understand the principle among them.

Project 1.1 Control LED by Manual Button

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

Component list

Freenove UNO x1 	Breadboard x1 		
USB cable x1	LED x1	Resistor 220Ω x1	Push button x1
Jumper M/M x2			

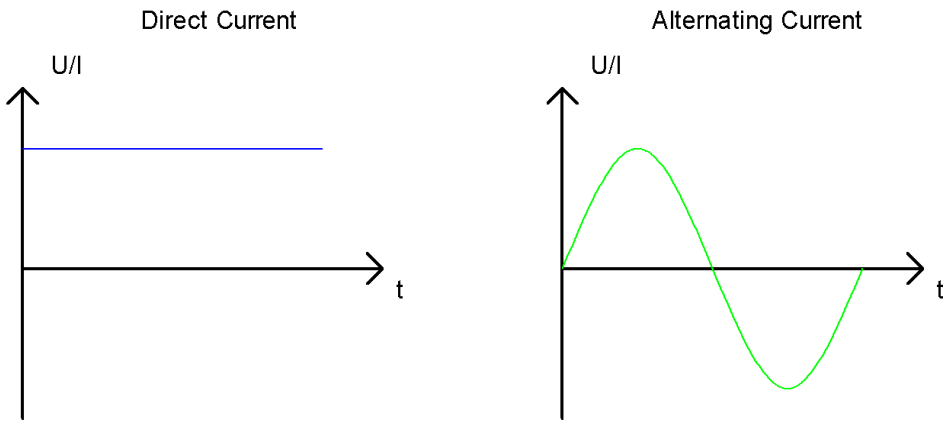
Circuit knowledge

Power supply

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

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

Voltage and current of AC power supply evolves periodically with time. Its basic form is sinusoidal voltage(current). It is suitable for long-distance transmission of electric energy. And it is used to supply power to homes.



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

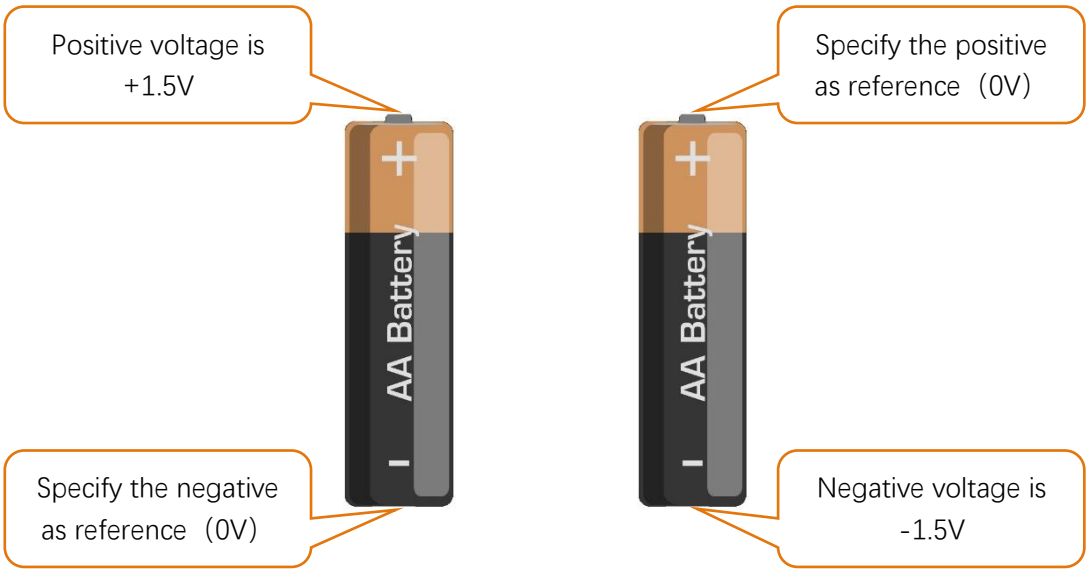
Battery or battery pack can be represented by the following symbols:

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

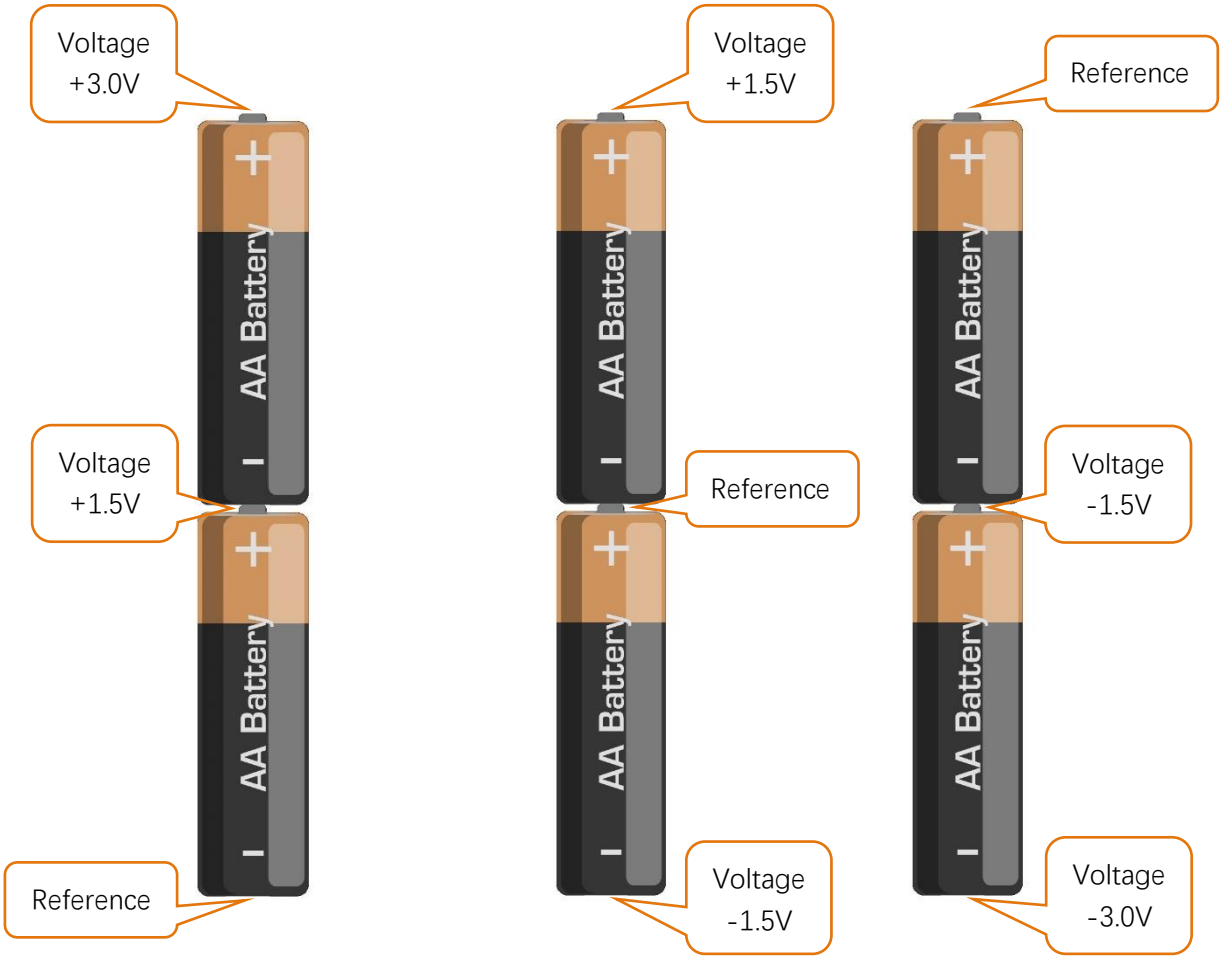
Voltage

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

Voltage is relative. As to a dry battery marked with "1.5V", it's 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:



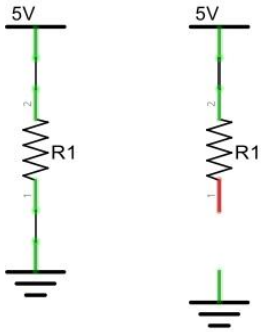
In practical circuits, we usually specify negative as reference voltage (0V), which is called "Ground". The positive is usually called "VCC". The positive and negative of power supply is usually represented by two following symbols:

Current

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

Closed loop consisting of electronic components is necessary for current.

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

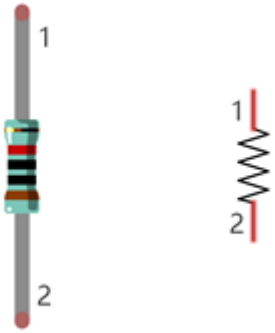


Resistor

The unit of resistance(R) is ohm(Ω). 1MΩ=1000kΩ, 1kΩ=1000Ω.

Resistor is an electrical component that limits or regulates the flow of current in an electronic circuit.

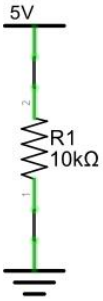
The left is the appearance of resistor. and the right is the symbol of resistor represented in circuit.



Color rings attached to the resistor is used to indicate its resistance. For more details of resistor color code, please refer to the appendix of this book.

With the same voltage there will be less current with more resistance. And the links among current, voltage and resistance can be expressed by the formula below: I=U/R.

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



Do not connect the two poles of power supply with low resistance, which will make the current too high to damage electronic components.

Component knowledge

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

Jumper

Jumper is a kind of wire. It is designed to connect the components together with its two terminals by inserting it onto Breadboard or Freenove UNO.

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

Jumper M/M

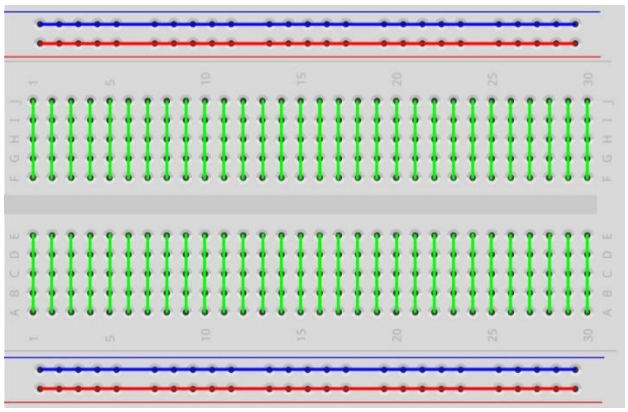
Jumper F/F

Jumper F/M

Breadboard

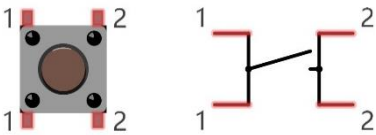
There are many small holes on breadboard to connect Jumper.

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



Push button

Push button has 4 pins. Two pins on the left is connected, and the right is similar as the left, which is shown in the below:

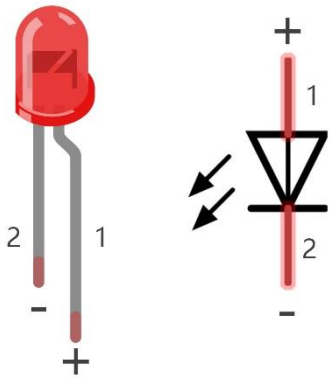


When the push button is pressed, the circuit is turned on.

LED

LED is a kind of diode. LED will shine only if the long pin of LED is connected to the positive electrode and the short pin is connected to negative electrode.

This is also the features of the common diode. Diode works only if the voltage of its positive electrode is higher than its negative electrode.

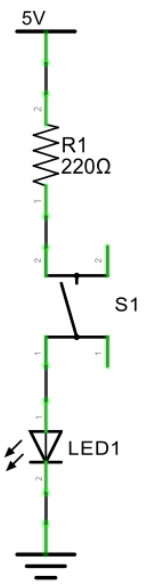
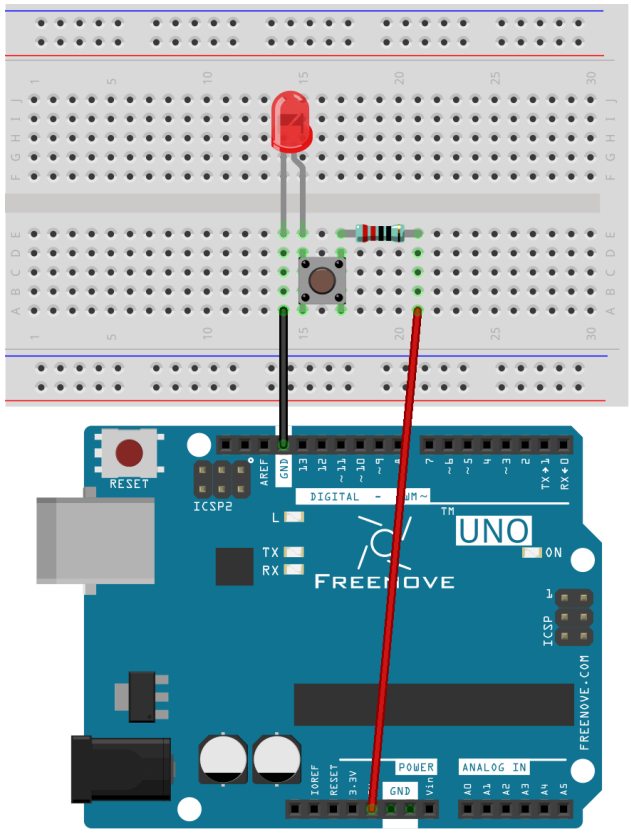


The LED can not be directly connected to power supply, which can damage component. A resistor with certain resistance must be connected in series in the circuit of LED.

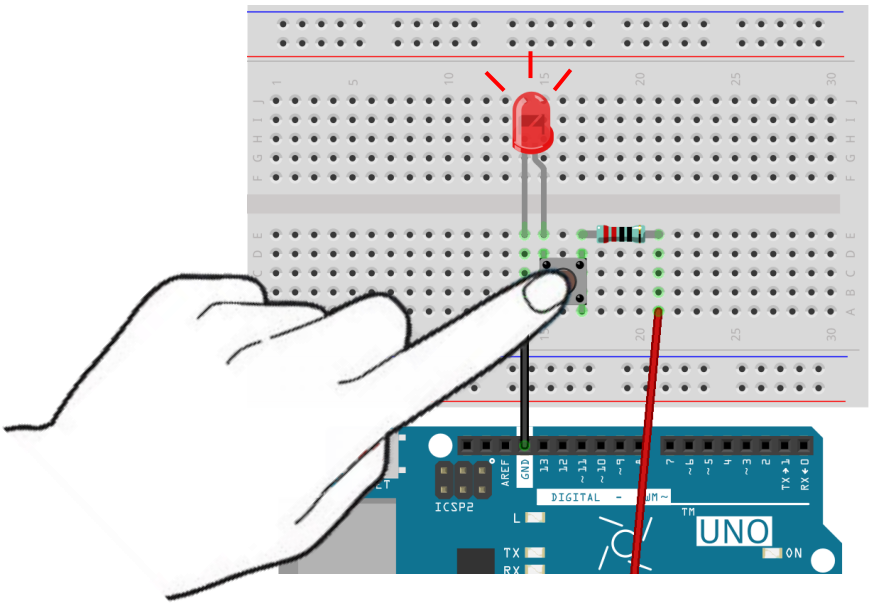
Circuit

In this experiment, the LED is controlled by push button, and Freenove UNO 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 there is no mistakes. Finally, connect Freenove UNO board to computer with USB cable to avoid that touch on wires may cause short-circuit fault.

Schematic diagram	Hardware connection
	

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



Project 1.2 Control LED by Arduino

Now, try using Arduino to make LED blink through programing.

Component list

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

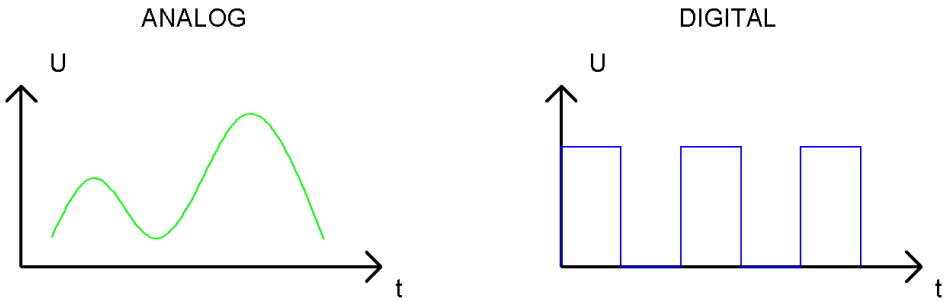
Circuit knowledge

Analog signal and Digital signal

The analog signal is a continuous signal in time and value. On the contrary, digital signal is a discrete signal in time and value.

Most signals in life are analog signals, for example, the temperature in one day is continuously changing, and will not appear a sudden change directly from 0℃ to 10℃, while the digital signal is a jump change, which can be directly from 1 to 0.

Their difference can be illustrated by the following figure.



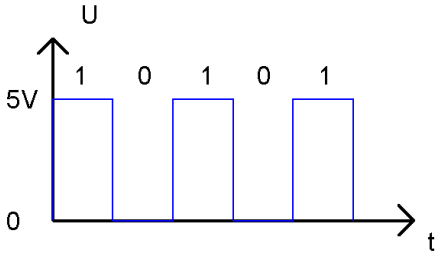
In practical application, the frequently used digital signal is binary signal, that is 0 and 1. The binary signal only has two forms (0 or 1), so it has strong stability. And digital signal and analog signal can be converted to each other.

Low level and high level

In circuit, the form of binary (0 and 1) is present 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 Freenove UNO is 0V and high level is 5V, as shown below. When IO port on Freenove UNO 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 won't affect the running of code.

There are two ways to use comments of sketches.

1. Symbol "//"

"//" will comment out the content behind it in current line:

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

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

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

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

"/*" and "*/" will comment out the content between them:

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

"/*" and "*/" can comment out multiple lines.:

```
/*
   this is a comment line.
   this is a comment line.
*/
```

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 date types, please visit the website: https://www.arduino.cc-Learning-Reference-Data Types.

Constant

Constant is a kind of data that can not 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 use, such as:

```
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:

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

"=" is used to passes 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:

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

Function

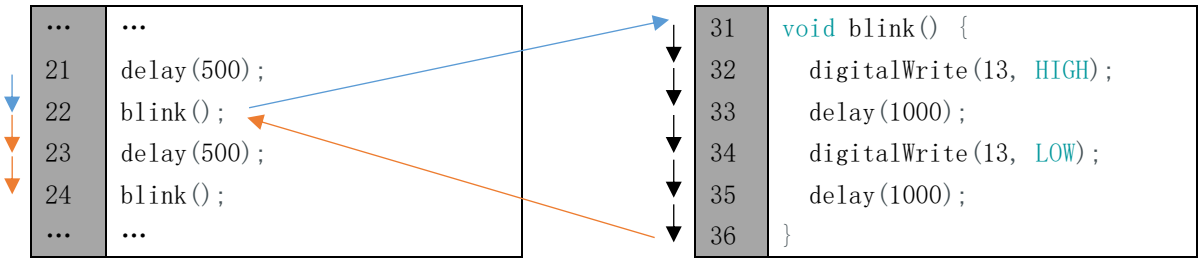
Function is a collection of statements with a sequence of order, which perform a defined task. Let's define a function void blink() as follows:

```
void blink() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

"void" indicates that the function does not return a value (the chapter 4 will detail the return value of functions);
"()" its inside is parameters of a function (the chapter 2 will detail the parameters of the function). 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.

```
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.

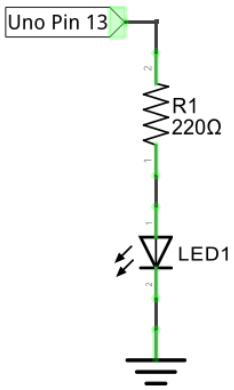
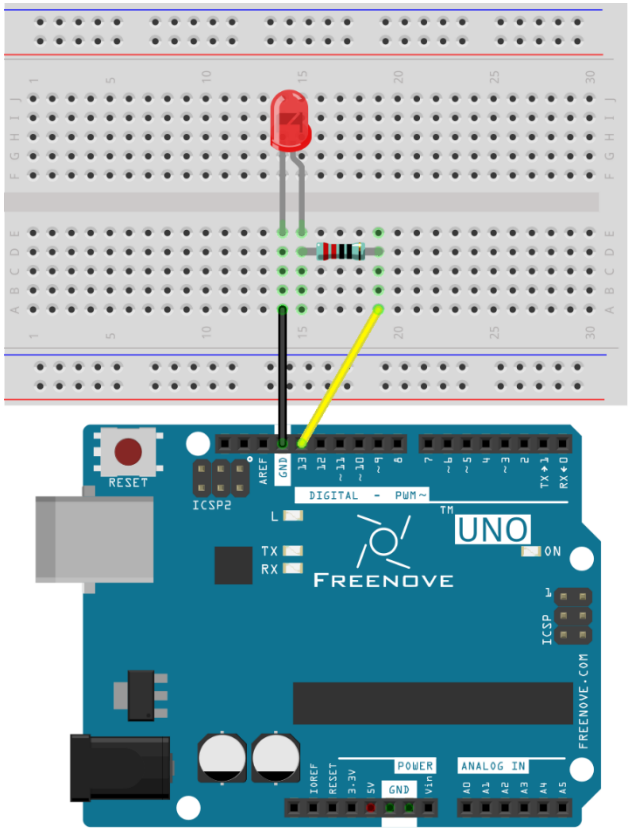


Some functions have one or more parameters. When you call such functions, you need to write parameters inside "()":

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

Circuit

Now, let 's use IO port of UNO Freenove to provide power for the LED. D13 pin of Freenove UNO is the digital pin. It can output high level or low level. In this way, Freenove UNO can control the state of LED.

Schematic diagram	Hardware connection
	

Sketch

Sketch 1.2.1

In order to make the LED blink, we need to make the 13 pin (D13) of Freenove UNO output high and low level alternately.

We highly recommend you type the code in person instead of copying and pasting, by this method, you can develop your coding skills and get more knowledge.

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

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

After Arduino board is reset, the setup() function will be executed firstly, 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 execute from the beginning. Switching on the power, finishing uploading the code and pressing the reset button will trigger reset operation.

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

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

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

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

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

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

Wait for 1000ms, that is 1s. Delay () function is used to make control board wait for a moment before executing the next statement. The parameter indicates the number of milliseconds to wait for.

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

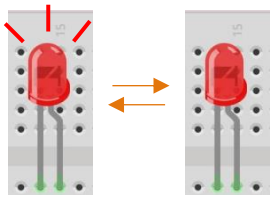
Then set the 13 pint to output low level, and LED light off. One second later, the execution of loop () function will be completed.

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

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

The function called above is standard function of the Arduino, which has been defined in the Arduino Software. Those function 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-Learning-Reference-Functions.

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



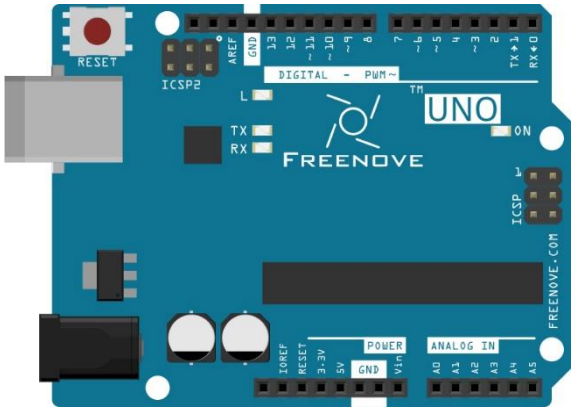
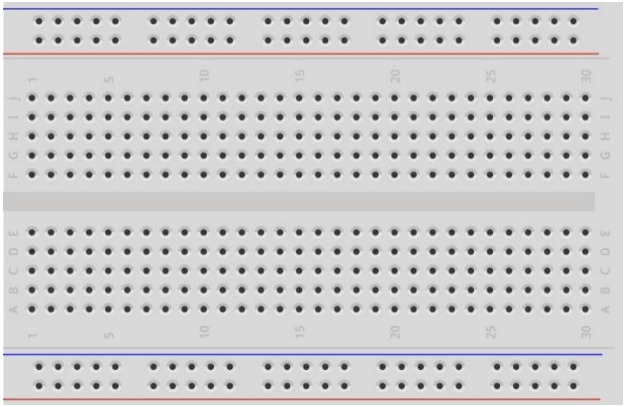
Chapter 2 Two LEDs Blink

In last chapter, we have already written code to make 1 LED blink on Freenove UNO 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 trough Freenove UNO board.

Component list

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

Code knowledge

In the last chapter, we have simple understanding of programming. Now let's learn more about the basic programming knowledge.

Parameters of function

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

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

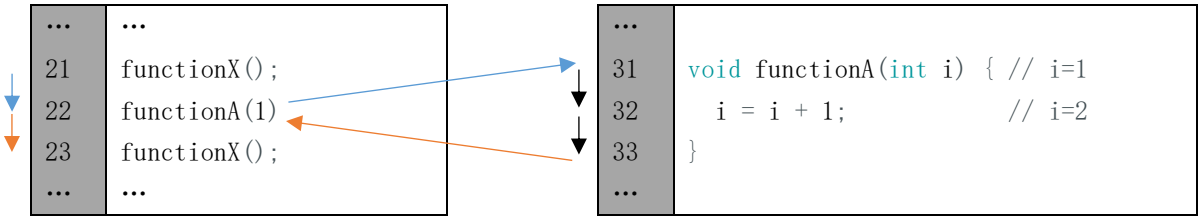
Next, we will define a function with a parameter as below:

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

"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:

```
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:

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

Boolean data type

Date 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.

```
boolean isTrue;
isTrue = true;            // after the execution, "isTrue" is assigned to true.
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:

```
Boolean isTrue;
isTrue = true && false;   // after the execution, "isTrue"is assigned to false.
isTrue = true || false;   // after the execution, "isTrue"is assigned to true.
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:

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

There are orther 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.

```
int i = 0;
if (i == 0)    // true
  i = 1;       // execution
functionX();
...
```

```
int i = 0;
if (i != 0)    // false
  i = 1;       // no execution
functionX();
...
```

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

```
int i = 0, j = 0;
if (i == 0) { // true
  i = 1;       // execution
  j = 1;       // execution
}
functionX();
...
```

```
int i = 0, j = 0;
if (i != 0) { // false
  i = 1;       // no execution
  j = 1;       // no execution
}
functionX();
...
```

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

```
int i = 0;
if (i == 0)    // true
  i = 1;       // execution
else
  i = 1;       // no execution
functionX();
...
```

```
int i = 0;
if (i != 0)    // false
  i = 1;       // no execution
else
  i = 1;       // execution
functionX();
...
```

In addition, it can judge multiple condition.

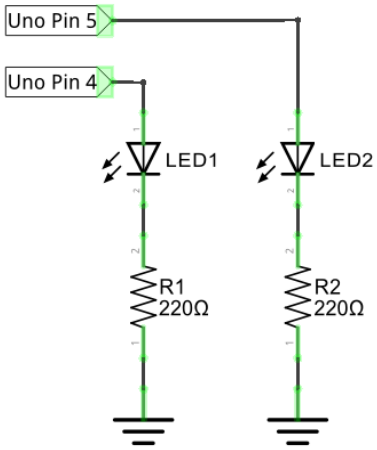
```
int i = 1;
if (i == 0)       // false
  i = 1;          // no execution
else if (i == 1)// true
  i = 2;          // execution
else if (i == 2)
  i = 3;          // no execution
else
  i = 4;          // no execution
functionX();
...
```

```
int i = 3;
if (i == 0)       // false
  i = 1;          // no execution
else if (i == 1)// false
  i = 2;          // no execution
else if (i == 2)// false
  i = 3;          // no execution
else
  i = 4;          // execution
functionX();
...
```

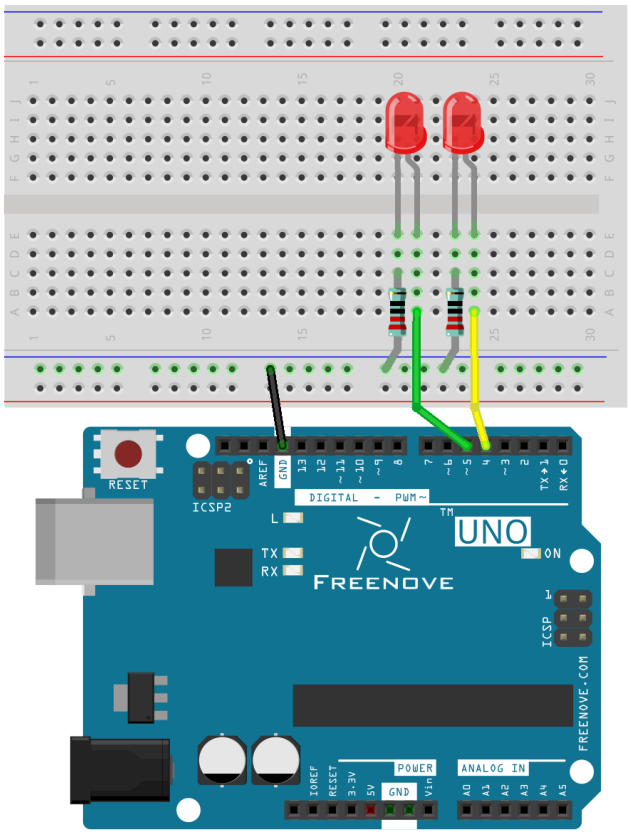
Circuit

Use D4 and D5 pin of Freenove UNO to drive these two LEDs respectively.

Schematic diagram



Hardware connection



Sketch

In order to show the difference between use of function and no use of 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.

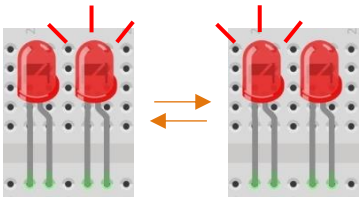
```
1  // set pin numbers:
2  int led1Pin = 4;            // the number of the LED1 pin
3  int led2Pin = 5;            // the number of the LED2 pin
4
5  void setup() {
6    // initialize the LED pin as an output:
7    pinMode(led1Pin, OUTPUT);
8    pinMode(led2Pin, OUTPUT);
9  }
10
11 void loop() {
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 with the last section. The difference is the amount of LEDs are two. And two LED blink alternatively.

Variable scope

In the 2,3 rows of code above, we define two variables to store the pin number. These two variables defined outside of the function are called "Global variable", which can be called by all other functions. Variables defined within 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 variable names represents local variable in the current 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 code.

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

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

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

```
// set pin numbers:
int led1Pin = 4;            // the number of the LED1 pin
int led2Pin = 5;            // the number of the LED2 pin

void setup() {
  // initialize the LED pin as an output:
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
}

void loop() {
  setLed(HIGH, LOW);      // set LED1 on, and LED2 off.
  setLed(LOW, HIGH);      // set LED1 off, and LED2 on.
}

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

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.

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

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

```
  setLed(HIGH, LOW);      // set LED1 on, and LED2 off.
  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 is defined in Arduino Software 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 enhance the readability of code and simplify code, such as INPUT, OUTPUT.

Sketch 2.1.3

In the last section of code, we used a function that integrates two similar paragraph of code. And We control two LEDs by using two parameters. As the two LEDs are always blinking alternatively, so let's 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.

```
// set pin numbers:
int led1Pin = 4;            // the number of the LED1 pin
int led2Pin = 5;            // the number of the LED2 pin

void setup() {
  // initialize the LED pin as an output:
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
}

void loop() {
  setLed1(HIGH);      // set LED1 on, and LED2 off.
  setLed1(LOW);       // set LED1 off, and LED2 on.
}

void setLed1(int led1) {
  digitalWrite(led1Pin, led1);   // the state of LED1 is determined by variable led1.

  if (led1 == HIGH)              // the state of LED2 is determined by variable led1.
    digitalWrite(led2Pin, LOW);  // if LED1 is turned on, LED2 will be turned off.
  else
    digitalWrite(led2Pin, HIGH); // if LED1 is turned off, LED2 will be turned on.

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

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, then two LEDs blink alternatively.

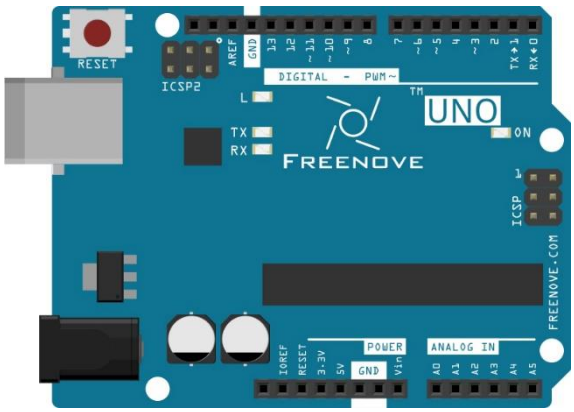
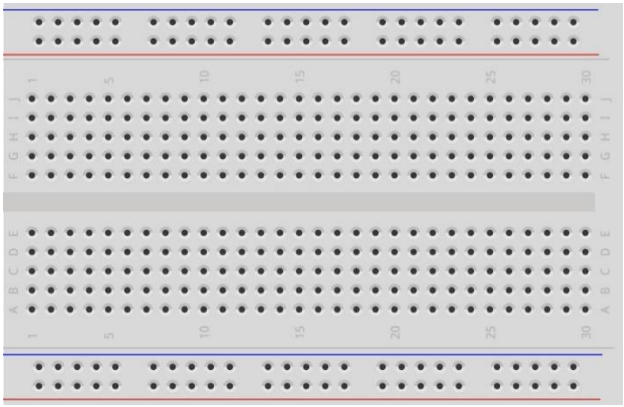
Chapter 3 LED bar graph

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

Project 3.1 LED bar graph Display

Let us use Freenove UNO to control a bar graph LED consist of 10 LEDs.

Component list

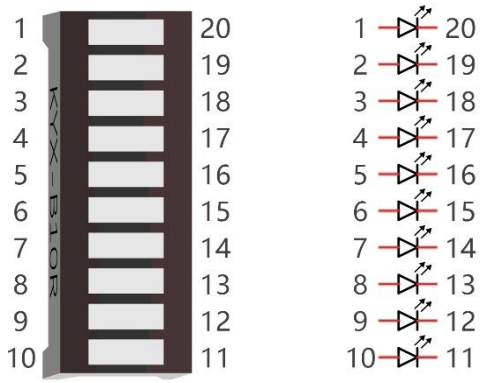
Freenove UNO x1 	Breadboard x1 	
USB cable x1	LED bar graph x1	Resistor 220Ω x10
Jumper M/M x11		

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 consist of 10 LEDs. There are two rows of pins at its bottom. 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 can not work in the circuit, it was 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 below:

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

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

```
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:

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

We can reference elements of an array as below:

```
int i, j;
i = a[0];
j = a[1];
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:

```
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.

```
while(expression)
  functionX();
```

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

```
while(expression){
  functionX();
  functionY();
}
```

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

```
int i = 0;
while (i < 2)
  i = i + 1;
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 go on the next execution:

```
do {
  functionX();
} while (expression);
```

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

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

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

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

Expression 1 is generally used to initialize variables; expression 2 is 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:

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

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

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

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

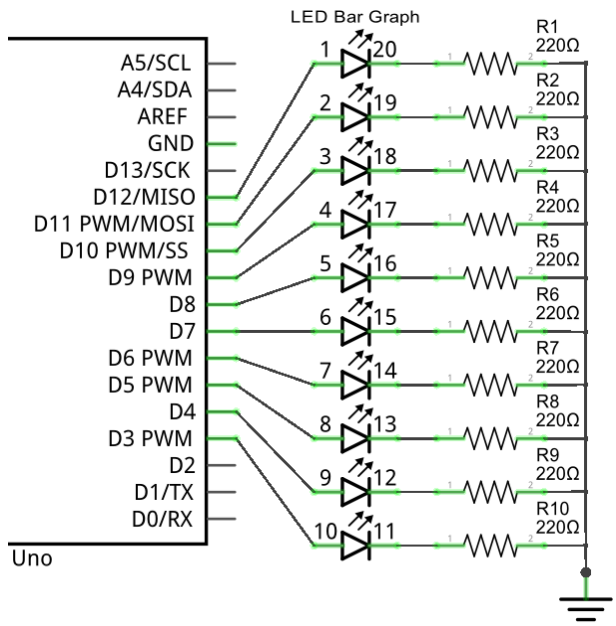
Operator ++, --

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

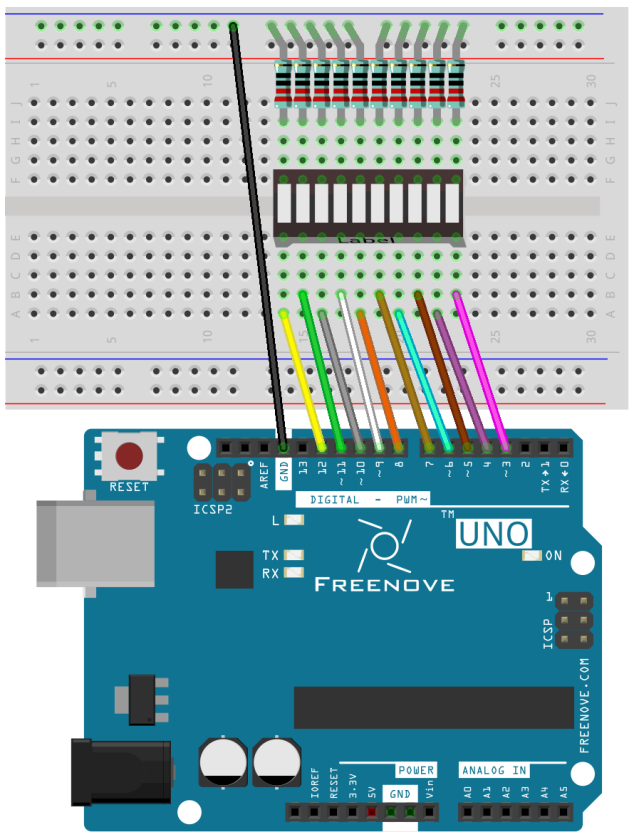
Circuit

Let us use D3, D4, D5, D6, D7, D8, D9, D10, D11, D12 pin of Freenove UNO board to drive LED bar graph.

Schematic diagram



Hardware connection



Sketch

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

Sketch 3.1.1

First, write a sketch which can achieve the LED light water.

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

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

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

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

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

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

```
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 with common variable. But read-only variables can be assigned only 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.

```
// an array of pin numbers to which LEDs are attached
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().

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

Define a function to open a certain LED on the LED bar graph and close the other LEDs.

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

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

```
void loop() {
  // make the "i"th LED of LED bar graph on and the others off in turn
  for (int i = 0; i < ledCount; i++) {
    barGraphDisplay(i);
  }
}
```

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

Sketch 3.1.2

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

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

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

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

void loop() {
  // makes the "i"th LED of LED bar graph bright in turn
  for (int i = 0; i < ledCount; i++) {
    barGraphDisplay(i);
  }
  // makes the "i"th LED of LED bar graph bright in reverse order
  for (int i = ledCount; i > 0; i--) {
    barGraphDisplay(i - 1);
  }
}

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

We have modified the code inside the function loop() and make the LED bar graph light in order, and then light in reverse order cyclically.

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

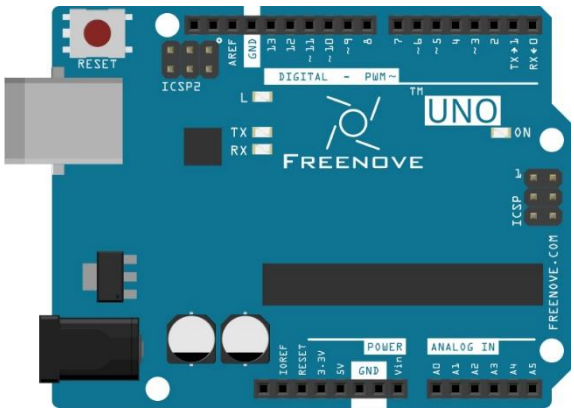
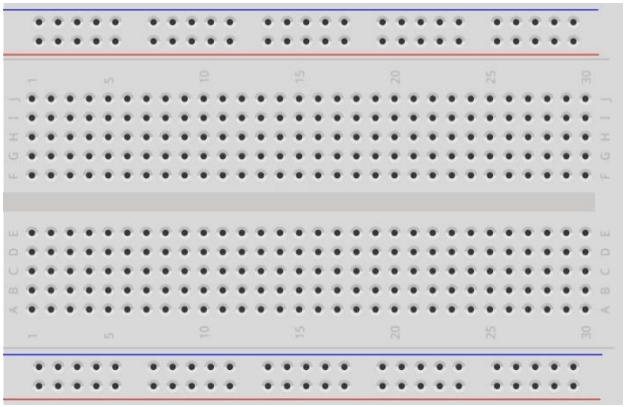
Chapter 4 LED Blink Smoothly

In the previous chapter, we have used Sketch on the Freenove UNO board to control up to 10 LED 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 Freenove UNO board to make 4 LED emit different brightness of light.

Component list

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

Circuit knowledge

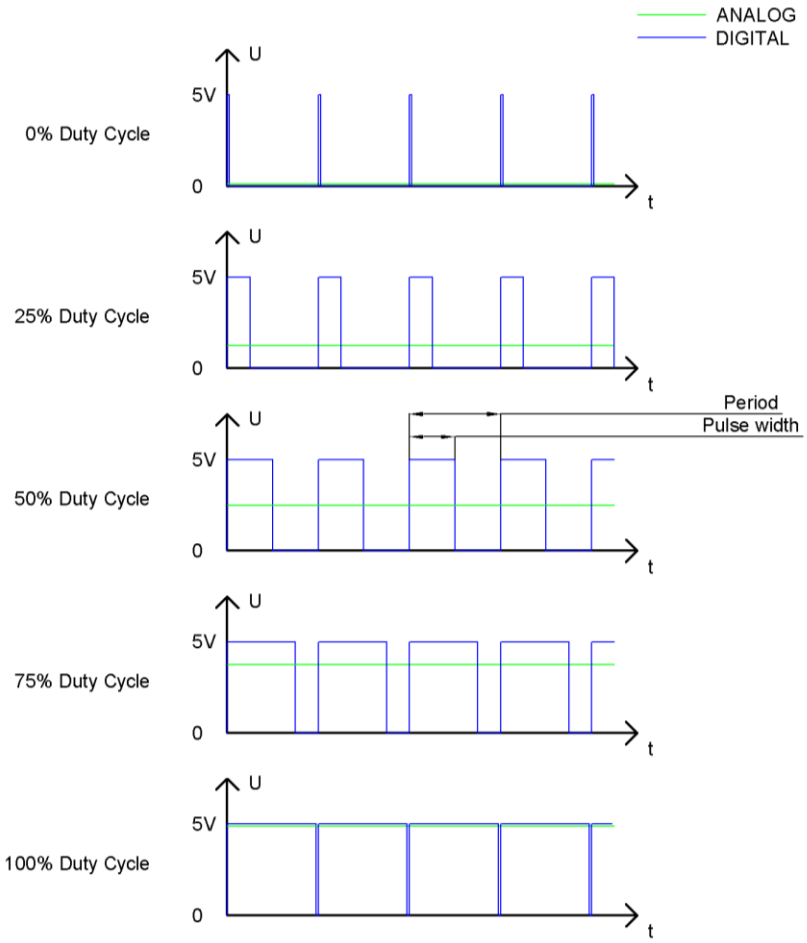
At first, let us learn the knowledge how to use the circuit to make LED emit different brightness of light,

PWM

PWM, namely Pulse Width Modulation, is a very effective technique for using digital signals to control analog circuits. The common processors can not directly output analog signals. PWM technology make it very convenient to achieve this purpose.

PWM technology uses digital pins to send certain frequency of square waves, that is, the output of high level and low level last for a period alternately. The total time for each set of high level and low level is generally fixed, which is called period (the reciprocal of the period is frequency). Output high time is generally called pulse width, and the percentage of pulse width is called duty cycle.

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



The larger PWM duty ratio, the lager the output power. So we can use PWM to control the brightness of LED, 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:

```
int sum(int i, int j) {
  int k = i + j;
  return k;
}
```

"int" declare the type of return value of the function sum(int i, int j). If the tape 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 return statement in this function.

A function with return value is called as follows:

```
int a = 1, b = 2, c = 0;
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:

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

It is equivalent to the following code:

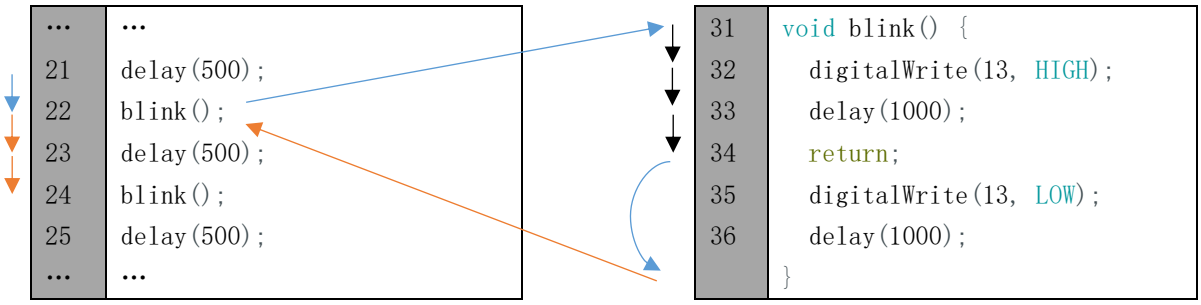
```
delay(300);
```

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 return value, and there is no data behind the return keyword:

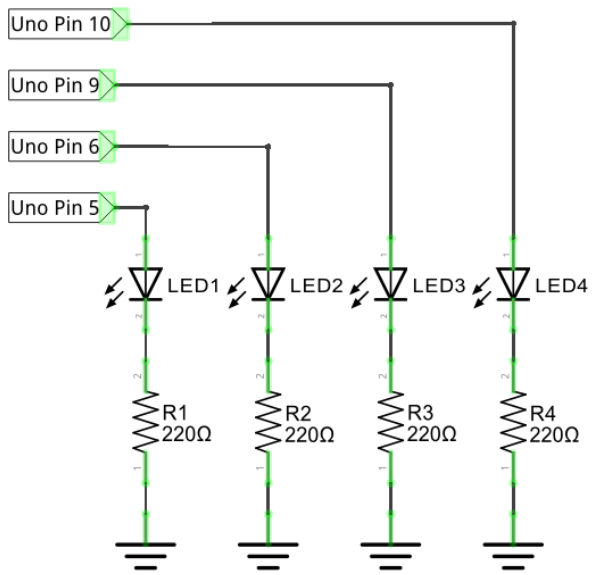
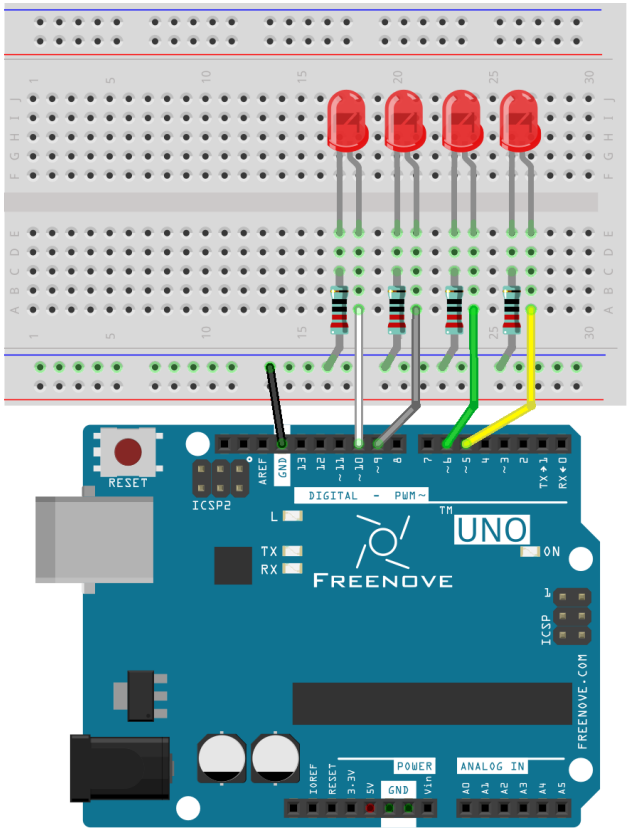
```
return;
```

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



Circuit

Use D5, D6, D9, D10 pin on Freenove UNO board to drive 4 LEDs.

Schematic diagram	Hardware connection
	

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 LED emit different brightness of the light.

```
// set pin numbers:
int ledPin1 = 5,            // the number of the LED1 pin
    ledPin2 = 6,            // the number of the LED2 pin
    ledPin3 = 9,            // the number of the LED3 pin
    ledPin4 = 10;           // the number of the LED4 pin

```

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
}

void loop()
{
  // set the ports output PWM waves with different duty cycle
  analogWrite(ledPin1, map(2, 0, 100, 0, 255));
  analogWrite(ledPin2, map(10, 0, 100, 0, 255));
  analogWrite(ledPin3, map(50, 0, 100, 0, 255));
  analogWrite(ledPin4, map(100, 0, 100, 0, 255));
}
```

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 0-255 range, and then output the PWM wave with duty cycle of 2%,

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

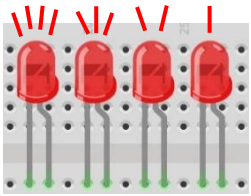
analogWrite(pin, value)

Arduino Software provides the function analogWrite (pin, value) which can make ports directly output PWM waves. Only the digital pin marked "~" symbol on The UNO board can use this function to output PWM waves. In the function analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255, which represent the duty ratio 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 percent in the range of toLow-toHigh is equal to the percent of "value" accounting for in range of fromLow-fromHigh. Such as 1 is in the range 0-1 in the maximum and the maximum value in the scope of 0-2 is 2, that is, the result value map (1, 0, 1, 0, 2) is 2.

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



Project 4.2 LED Blink Smoothly

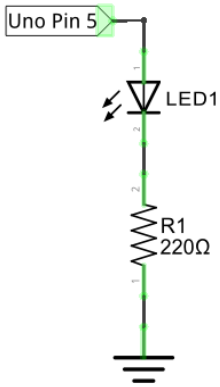
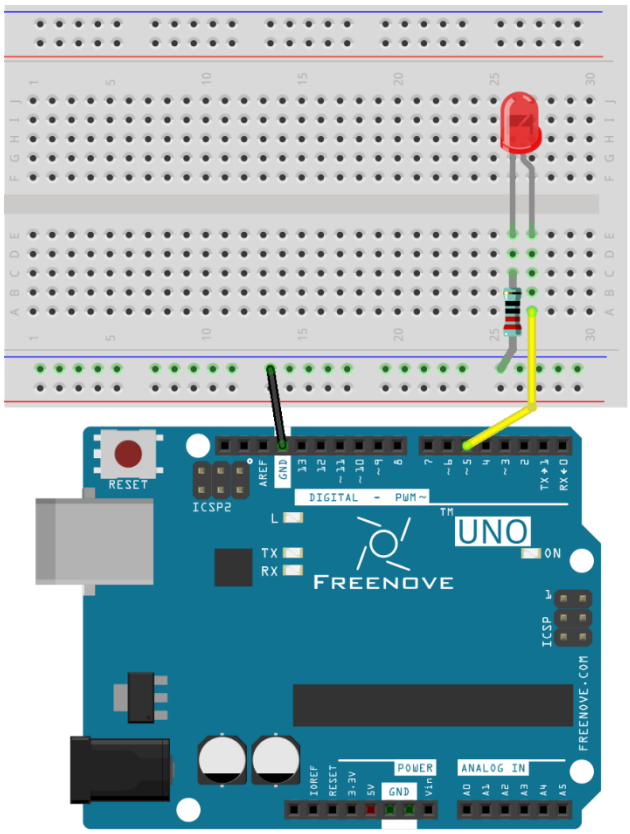
We will learn how to make a LED blink smoothly, that is, breathing light.

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 D6, D9, D10 pin on Freenove UNO board in circuit of the last section.

Schematic diagram	Hardware connection
	

Sketch

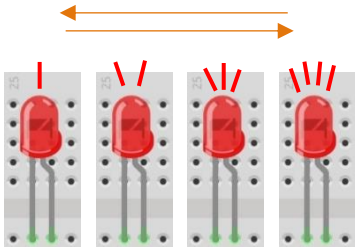
Sketch 4.2.1

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

```
// set pin numbers:
int ledPin = 5;             // the number of the LED pin

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // call breath() cyclically
  breath(ledPin, 6);
  delay(500);
}

void breath(int ledPin, int delayMs) {
  for (int i = 0; i <= 255; i++) {  // "i" change from 0 to 255
    analogWrite(ledPin, i);         // corresponding duty cycle change from 0%-100%
    delay(delayMs);                 // adjust the rate of change of brightness
  }
  for (int i = 255; i >= 0; i--) {  // "i" change from 255 to 0
    analogWrite(ledPin, i);         // corresponding duty cycle change from 0%-100%
    delay(delayMs);                 // adjust the rate of change in brightness
  }
}
```

Through two cycles, 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 rate of change in the "for" cycle, and you can try to modify the parameters to modify the rate of change in 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.



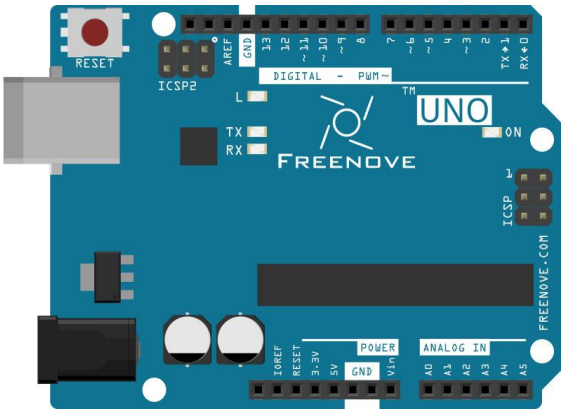
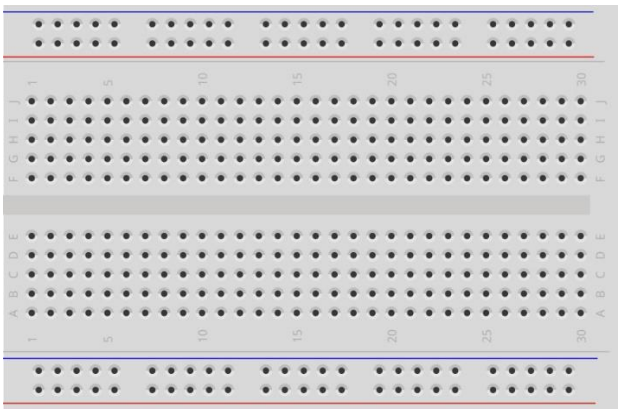
Chapter 5 Control LED Through Push Button

From previous chapter, we have use Freenove UNO to output signals to make 10 LEDs flash, and make one LED emit different brightness. Now, let's learn how to get the input signal.

Project 5.1 Control LED Through Push Button

We will use the Freenove UNO to get the status of the push button, and show that through LED.

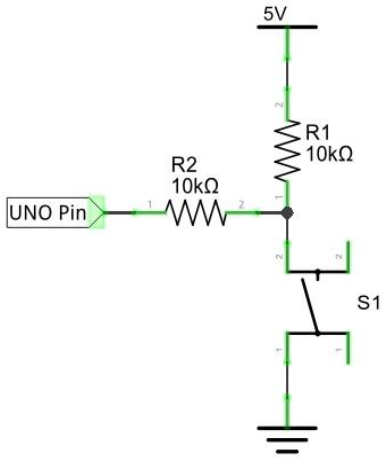
Component list

Freenove UNO x1 	Breadboard x1 			
USB cable x1	LED x1	Resistor 220Ω x1	Resistor 10kΩ x2	Push button x1
Jumper M/M x4				

Circuit knowledge

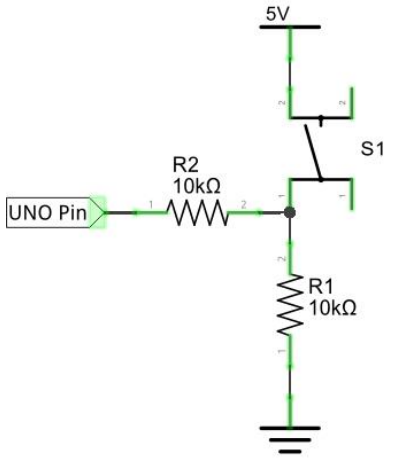
Connection of Push Button

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



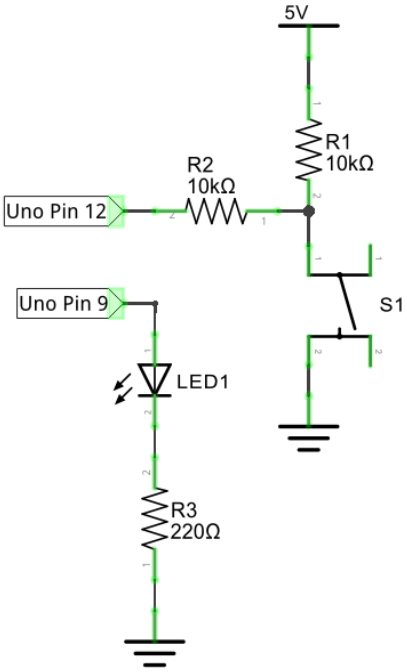
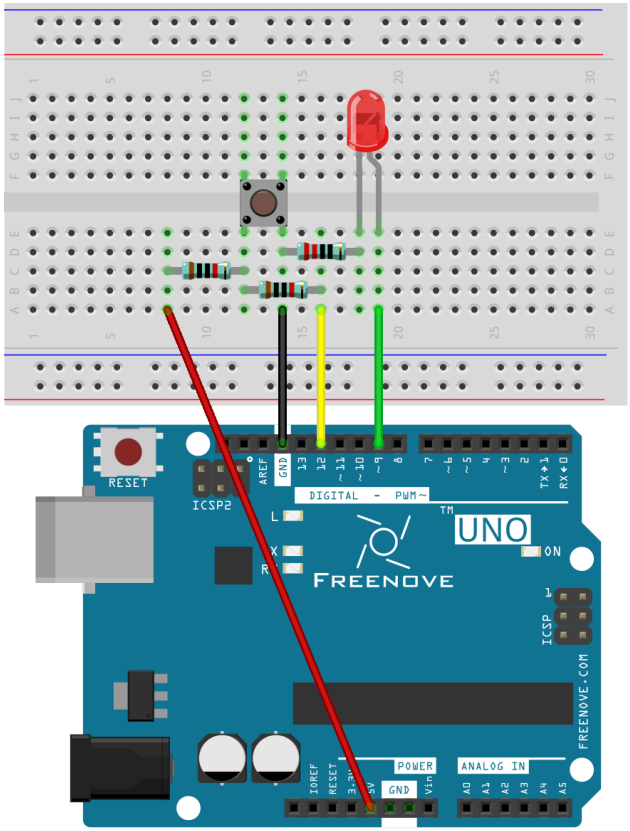
In the above circuit diagram, when the button is not pressed, 5V (high level) will be detected by UNO port; and 0V (low level) when the button is pressed. The role of Resistor R2 here is to prevent the port from being set to output high level by accident, which could be connected directly to the cathode and cause a short circuit when the button is pressed.

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



Circuit

Use D12 of Freenove UNO to detect the status of push button, and D9 to drive LED.

Schematic diagram	Hardware Connection
	

Sketch

Sketch 5.1.1

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

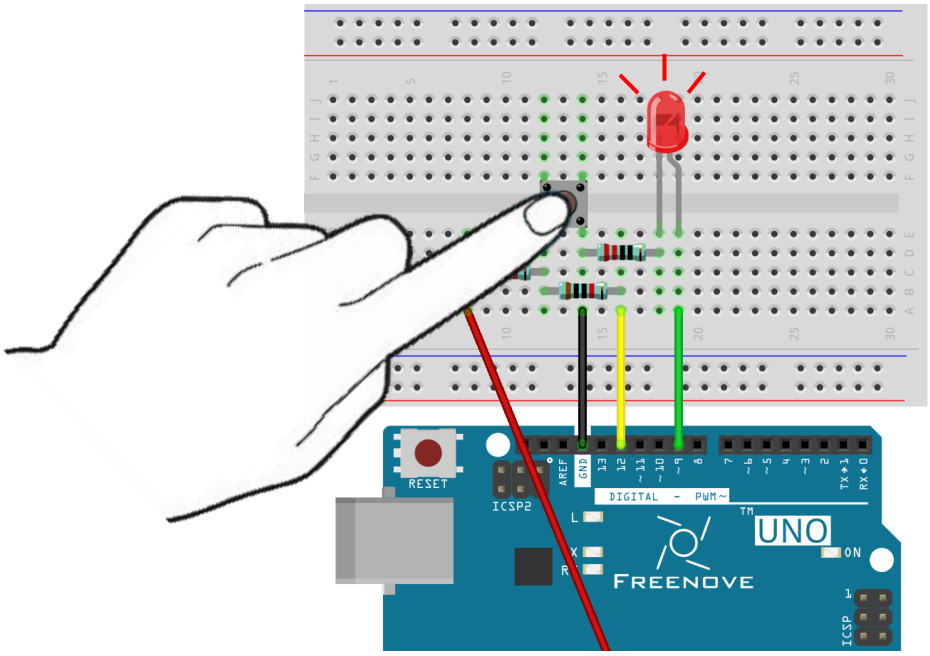
```
int buttonPin = 12; // the number of the push button pin
int ledPin = 9;     // the number of the LED pin

void setup() {
  pinMode(buttonPin, INPUT);  // set push button pin into input mode
  pinMode(ledPin, OUTPUT);    // set LED pin into output mode
}

void loop() {
  if (digitalRead(buttonPin) == HIGH) // if the button is not pressed
    digitalWrite(ledPin, LOW);        // switch off LED
  else                                // if the button is pressed
    digitalWrite(ledPin, HIGH);       // switch on LED
}
```

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

digitalRead(pin)

Arduino Software 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, then LED lights up; loosen the button, then LED lights off.



Project 5.2 Change LED State by Push Button

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

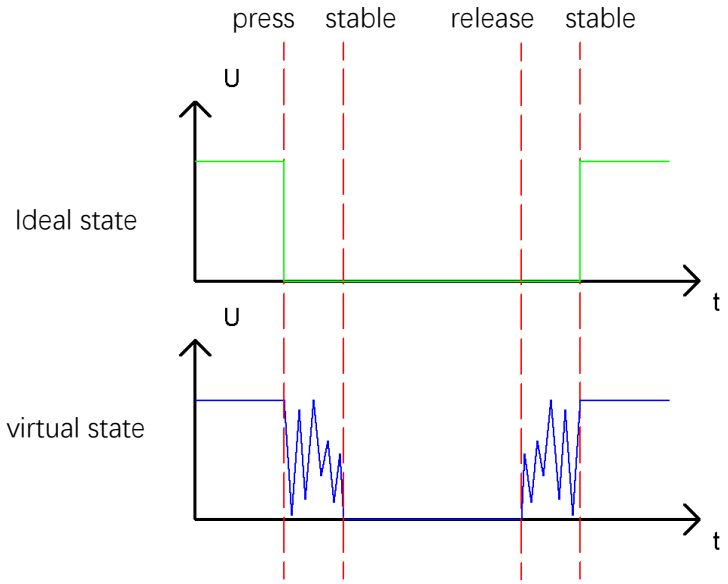
Component List

Same with the last section.

Circuit knowledge

Debounce for push button

When push button is pressed, the potential won't transfer from one state to another state ideally. In fact, it will occur a continuous bounce process before it becomes final stable state.



If you detect the state of push button, there may be repeated "press" and "release" detected during one process of pressing. So it is necessary to eliminate the influence caused by the bounce. Here is the most direct and effective way: when the changed signals are received by Freenove UNO for the first time, the program does not operate immediately, just waits for a certain time to skip the bounce process and confirm the final state of push button.

Circuit

Same with the last section.

Sketch

Sketch 5.2.1

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

```
int buttonPin = 12; // the number of the push button pin
int ledPin = 9;     // the number of the LED pin
boolean isLighting = false; // define a variable to save the state of LED

void setup() {
  pinMode(buttonPin, INPUT);  // set push button pin into input mode
  pinMode(ledPin, OUTPUT);    // set LED pin into output mode
}

void loop() {
  if (digitalRead(buttonPin) == LOW) {        // if the button is pressed
    delay(10);    // delay for a certain time to skip the bounce
    if (digitalRead(buttonPin) == LOW) {      // confirm again if the button is pressed
      reverseLED();                           // reverse LED
      while (digitalRead(buttonPin) == LOW);  // wait for releasing
      delay(10);  // delay for a certain time to skip bounce when the button is released
    }
  }
}

void reverseLED() {
  if (isLighting) {             // if LED is lighting,
    digitalWrite(ledPin, LOW);  // switch off LED
    isLighting = false;         // store the state of LED
  }
  else {                        // if LED is off,
    digitalWrite(ledPin, HIGH); // switch LED
    isLighting = true;          // store the state of LED
  }
}
```

We use a function reverseLED () to change the state of the LED.

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

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

Sketch 5.2.2

Beginners can upload Sketch_5.2.2 to try not dealing with the push button bounce and see what happens.

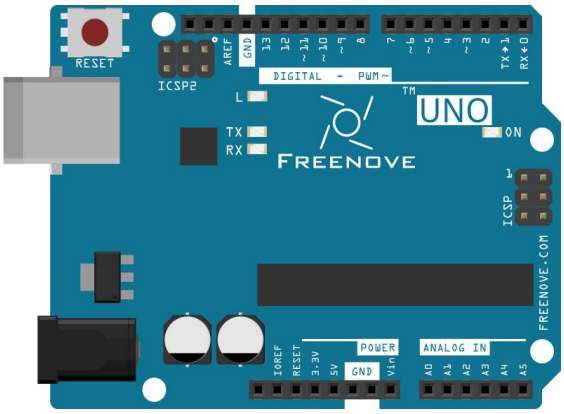
Chapter 6 Serial

Earlier, we already try to output signals to LED, and get the input signal of push button. Now, we can try a more advanced means of communication, serial communication.

Project 6.1 Send data through Serial

We will use the serial port on Freenove UNO to send data to computer.

Component list

Freenove UNO x1 	USB cable x1

Code knowledge

Bit and Byte

As mentioned earlier, the computer uses a binary signal. A binary signal is called 1 bit, and 8 bits organize in order is called 1 byte. Byte is the most basic unit of storage and communications for processors. 1 byte can represent a 2^8=256 numbers, that is, 0-255. For example:

As to binary number 10010110, "0" usually present the lowest 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 decimal number divided by 2 to convert it to binary number. Quotient continuously obtained divided by 2 until quotient is zero. Arrange all remainders from right to left in a line. Then we complete the conversion. For example:

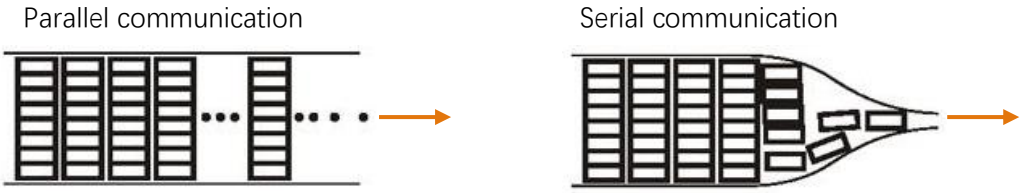
		Remainder	Sequence
2	150	0	0
2	75	1	1
2	37	1	2
2	18	0	3
2	9	1	4
2	4	0	5
2	2	0	6
2	1	1	7
	0		

The result is 10010110.

Circuit knowledge

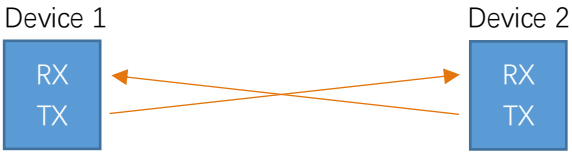
Serial and parallel communication

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



Serial communication

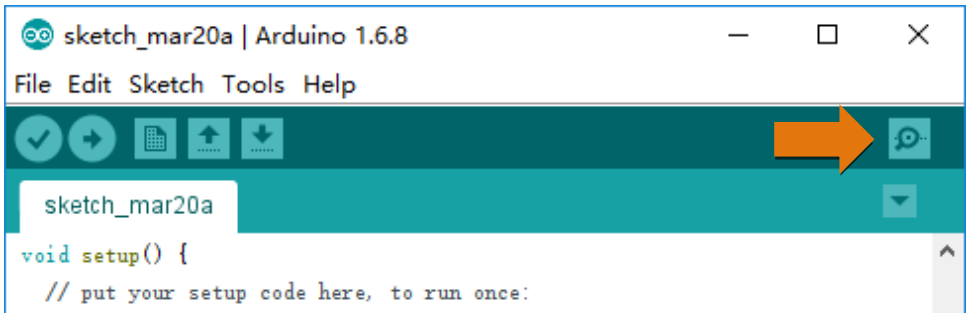
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART). And it is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and another for receiving data (RX line). The serial communication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



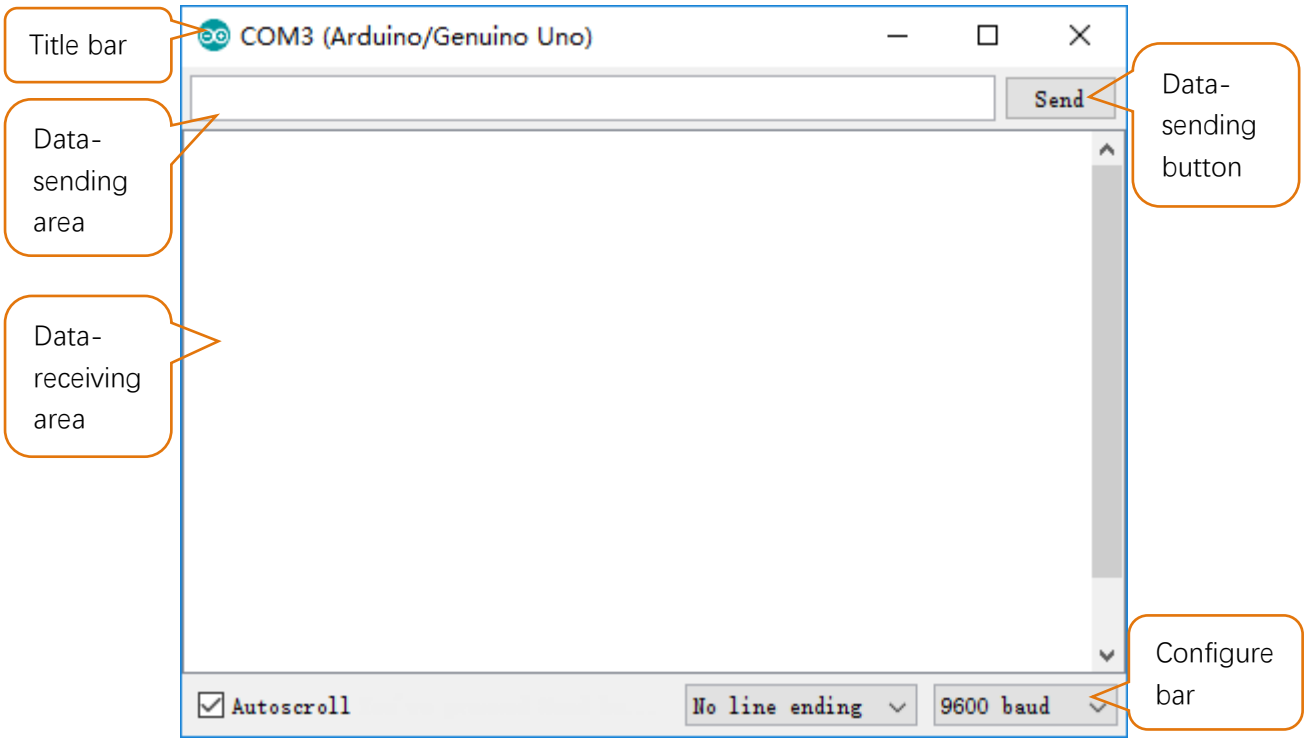
Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used is 9600 and 115200.

Serial port on Arduino

Freenove UNO has integrated USB to serial transfer, could communicates with computer when USB cable get connected to it. Arduino Software also uploads code for Freenove UNO through the serial connection. Computer identifies serial devices connected to your computer as COMx. We can use the Serial Monitor window of Arduino Software to communicate with Freenove UNO, connect Freenove UNO to computer through the USB cable, choose the right device, and then click the Serial Monitor icon to open the Serial Monitor window.

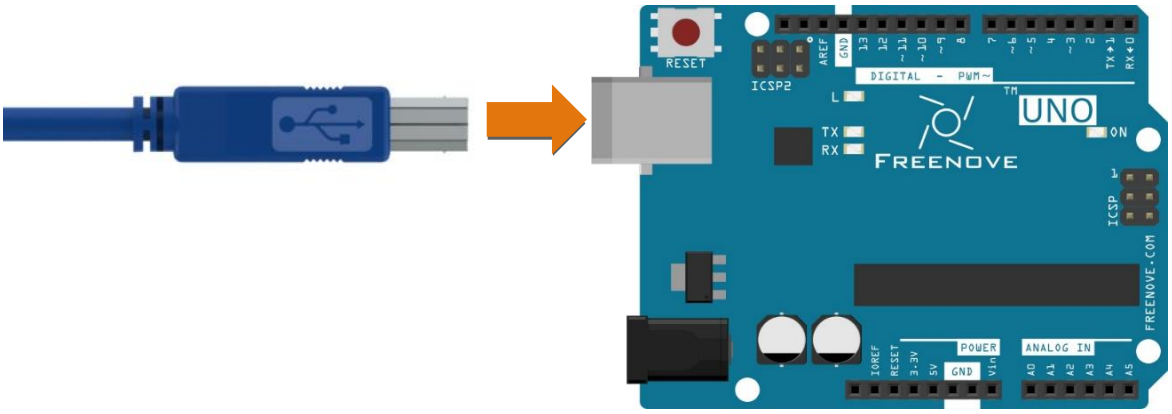


Interface of Serial Monitor window is as follows. If you can't open it, make sure Freenove UNO had been connected to the computer, and choose the right serial port in the menu bar "Tools".



Circuit

Connect Freenove UNO to the computer with USB cable.



Sketch

Sketch 6.1.1

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

```
int counter = 0;  // define a variable as a data sending to serial port

```

```
void setup() {
  Serial.begin(9600);                  // initialize the serial port, set the baud rate to 9600
  Serial.println("UNO is ready!");     // print the string "UNO is ready!"
}

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

setup() function initializes the serial port and send the string "UNO is ready!".

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

Serial class

Class is a C++ language concept. Arduino Software supports C++ language, which is a language extension. We don't explain specifically the concept here, only describes how to use it. If you are interested, you can learn that 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:

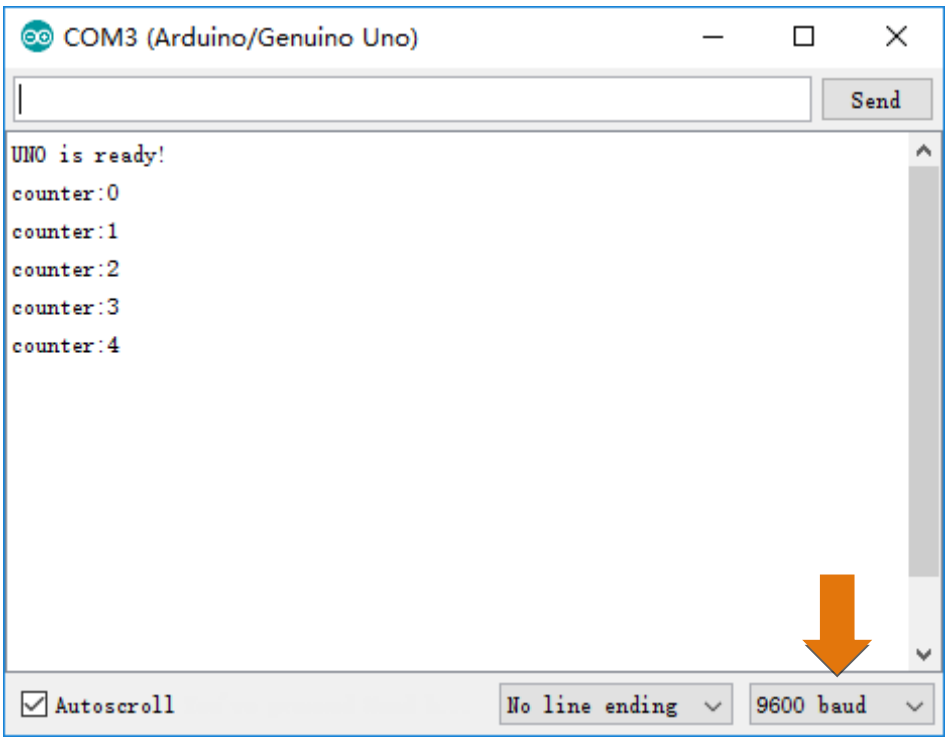
Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

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, then you'll see data sent from UNO.

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



Project 6.2 Receive Data through Serial Port

From last section, we use Serial port on Freenove UNO to send data to a computer, now we will use that to receive data from computer.

Component list

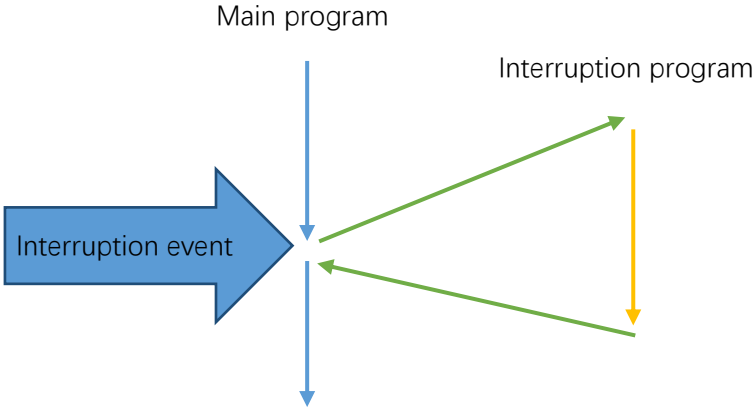
Same with last section.

Code knowledge

Interruption

Interruption is controller's response to an event. The event causing an interruption is interruption source. We'll illustrate this interruption concept. If you're watching TV, while water is 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 interruption can work as warning device for kettle, and before the water is boiling, you can focus on watching TV until a beep warning comes out, then go to handle it.

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



Circuit

Same with last section.

Sketch

Sketch 6.2.1

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

```
char inChar;      // define a variable to store characters received from serial port

void setup() {
  Serial.begin(9600);                 // initialize serial port, set baud rate to 9600
  Serial.println("UNO is ready!");    // print the string "UNO is ready!"
}

void loop() {
  if (Serial.available()) {         // judge whether data has been received
    inChar = Serial.read();         // read one character
    Serial.print("UNO received:");  // print the string "UNO received:"
    Serial.println(inChar);         // print the received character
  }
}
```

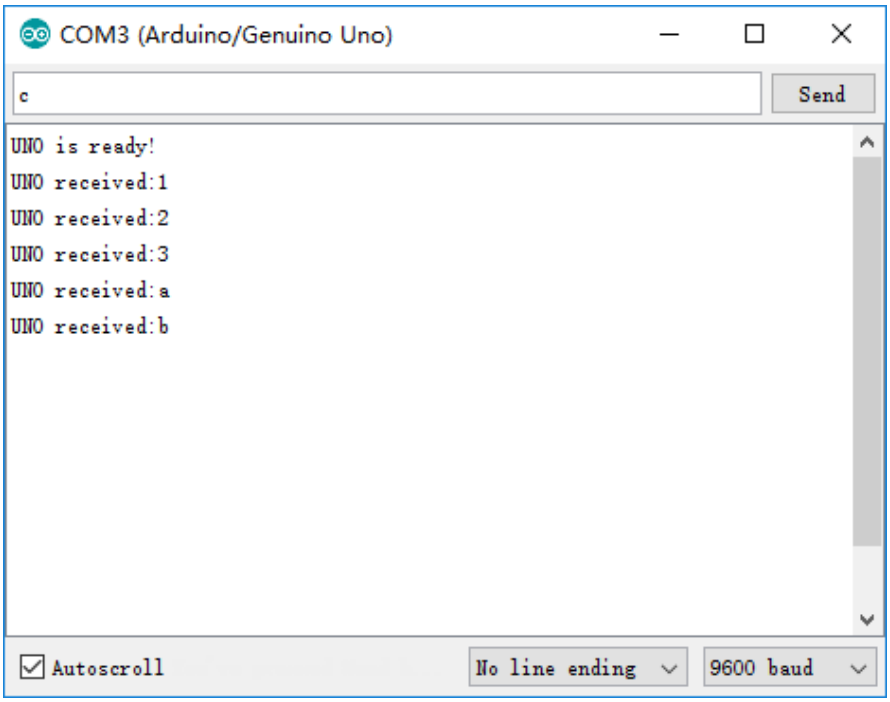
In the setup() function, we initialize the serial port and send the string "UNO is ready!". Then, the loop() function will continuously detect whether there are data needs to be read. if so, read the character and send it back.

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 sending area, click Send button, then you'll see information returned from UNO.



char type

char type variable can represent a character, but char type 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.

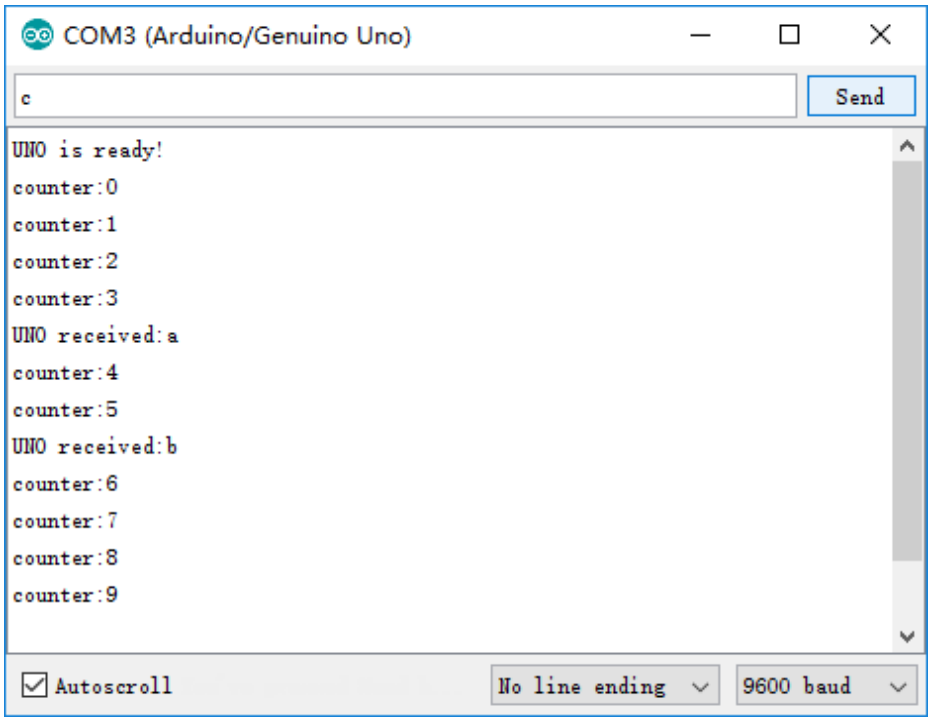
Sketch 6.2.2

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

```
char inChar;      // define a variable to store character received from serial port
int counter = 0;  // define a variable as the data sent to Serial port

void setup() {
  Serial.begin(9600);                 // initialize serial port and set baud rate to 9600
  Serial.println("UNO is ready!");    // print the string "UNO is ready!"
}

void loop() {
  // Print value of variable counter to serial
  Serial.print("counter:");    // print the string "counter:"
  Serial.println(counter);     // print the value of variable "counter"
  delay(1000);                 // wait 1000ms to avoid cycling too fast
  counter++;                   // variable "counter" increases 1
}

void serialEvent() {
  if (Serial.available()) {          // judge whether the data has been received
    inChar = Serial.read();          // read one character
    Serial.print("UNO received:");   // print the string "UNO received:"
    Serial.println(inChar);          // print the received character
  }
}
```

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

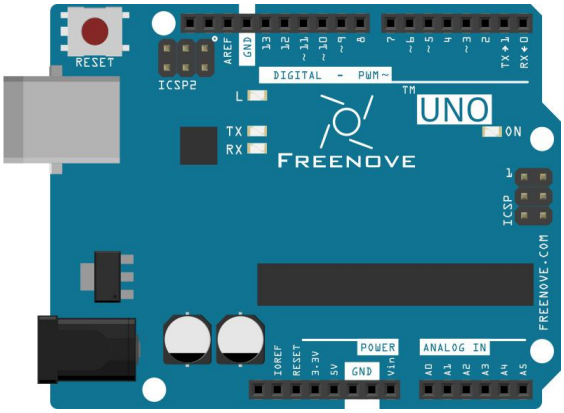
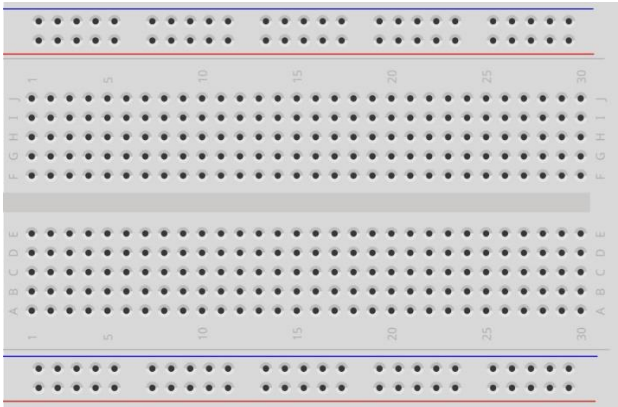
Verify and upload the code, open the Serial Monitor, then you'll see the number constantly sent from UNO. Fill character in the sending area, and click the Send button, then you'll see the string returned from UNO.



Project 6.3 Application of Serial

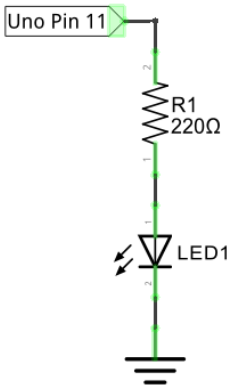
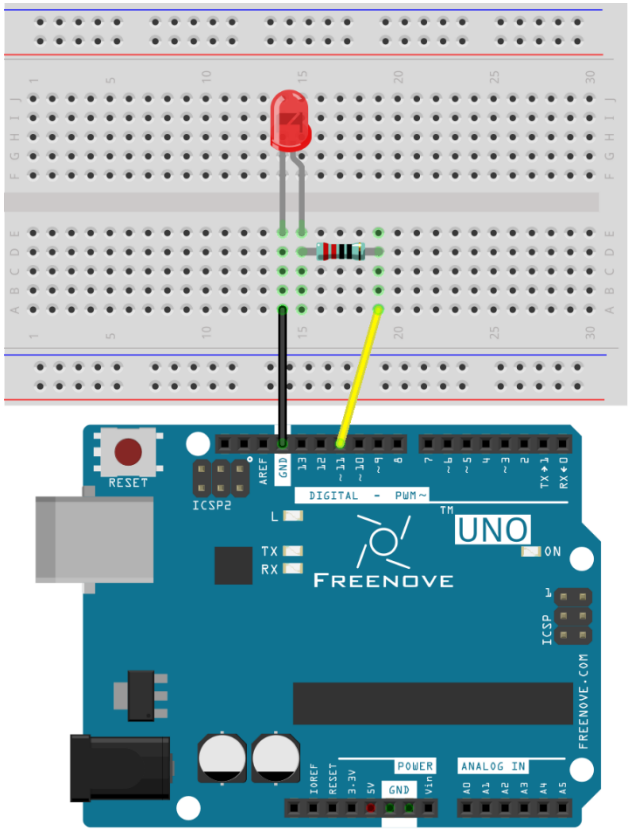
We will use the serial port on Freenove UNO to control one LED.

Component list

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

Circuit

Here we will use D11 pin of Freenove UNO to output PWM to drive 1 LED.

Schematic diagram	Hardware connection
	

Sketch

Sketch 6.3.1

Code is basically the same with Sketch 6.2.1. But after receiving the data, Freenove UNO will convert it into PWM duty cycle from output port.

```
int inInt;          // define a variable to store the data received from serial
int counter = 0;    // define a variable as the data sending to serial
int ledPin = 11;    // the number of the LED pin

void setup() {
  pinMode(ledPin, OUTPUT);              // initialize the LED pin as an output
  Serial.begin(9600);                   // initialize serial port, set baud rate to 9600
  Serial.println("UNO is ready!");      // print the string "UNO is ready!"
}

```

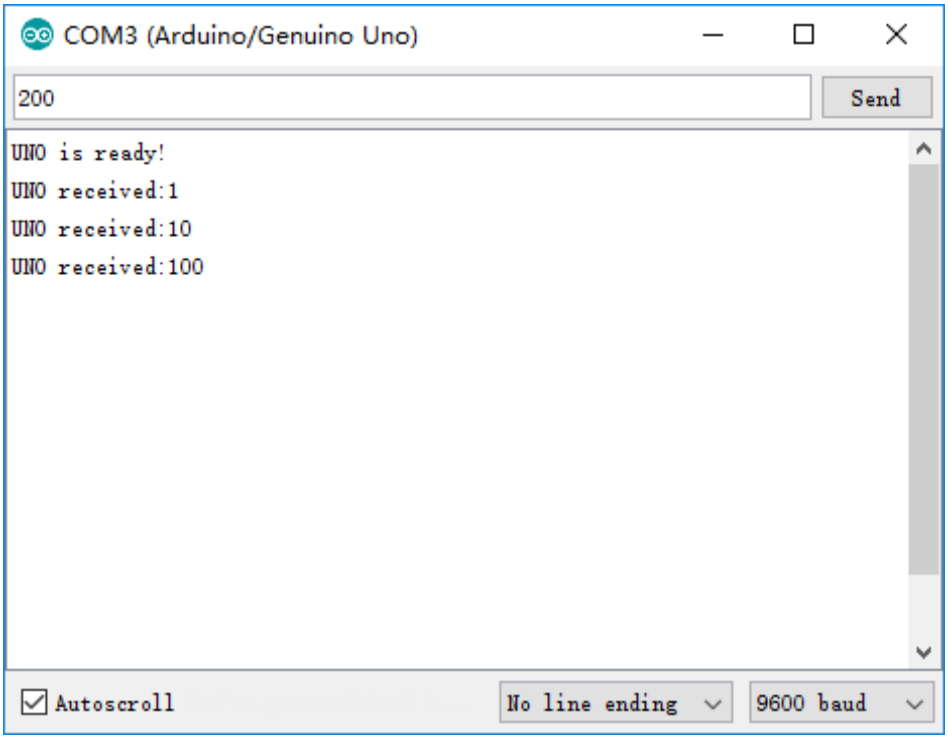
```
void loop() {
  if (Serial.available()) {          // judge whether the data has been received
    inInt = Serial.parseInt();       // read an integer
    Serial.print("UNO received:");   // print the string "UNO received:"
    Serial.println(inInt);           // print the received character
   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

When serial receives data, it converts the data into PWM duty cycle of output port, and then causes LED to 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 0-255 into the sending area and click the Send button. Then you'll see information returned from UNO, meanwhile, LED can emit light with different brightness according to the number you send.



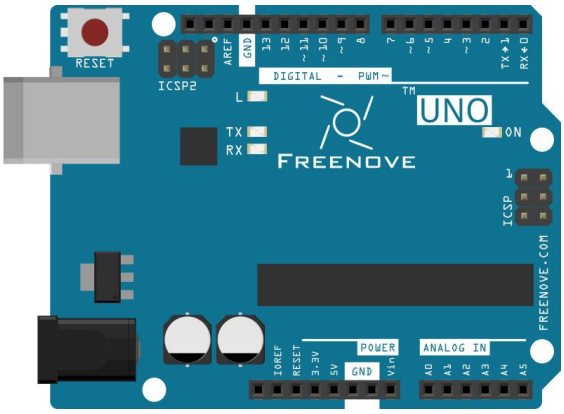
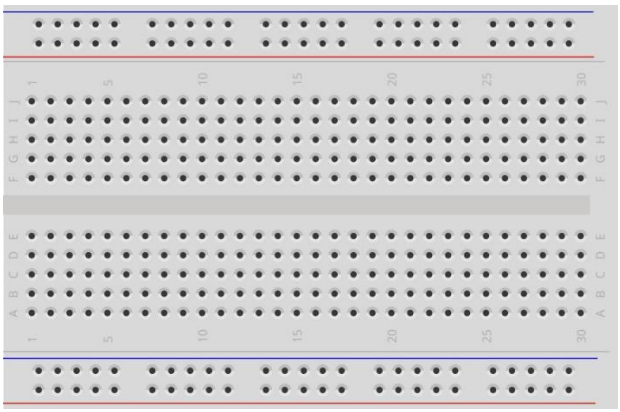
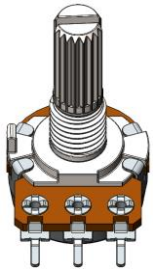
Chapter 7 ADC

Before, we have learned the digital ports of Freenove UNO, and tried the output and input signals. Now, let's learn how to use analog ports.

Project 7.1 ADC

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

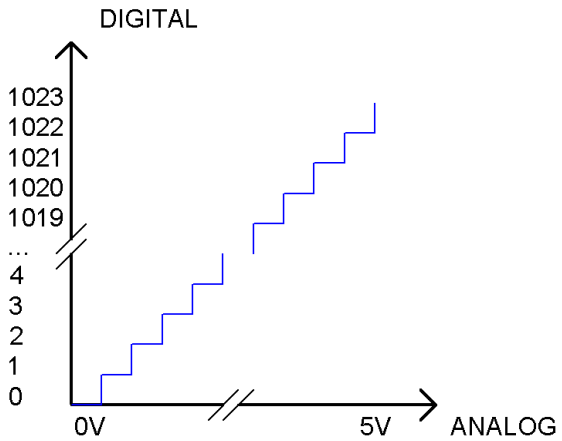
Component list

Freenove UNO x1 	Breadboard x1 
USB cable x1	Rotary potentiometer x1 
Jumper M/M x3	

Circuit knowledge

ADC

ADC, Analog-to-Digital Converter, is a device used to convert analog to digital. Freenove UNO has a 10 bits ADC, that means the resolution is 2^10=1024, and the range (here is 5V) will be divided equally into 1024 parts. The analog of each section corresponds to one ADC value. So the more bits ADC has, the denser the partition of analog will be, also the higher precision of the conversion will be.

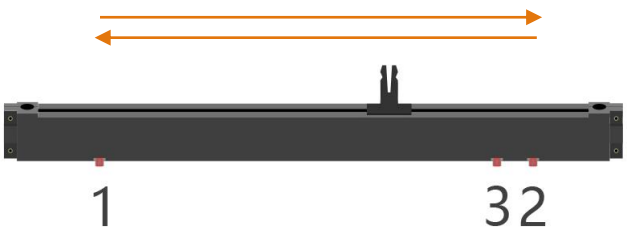


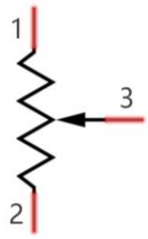
Subsection 1: the analog in rang of 0V-5/1024V corresponds to digital 0;
Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;
...
The following analog will be divided accordingly.

Component knowledge

Potentiometer

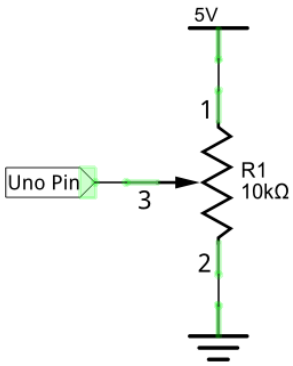
Potentiometer is a resistive component with three terminal parts and its resistance can be adjusted in accordance with according to a certain change rule. Potentiometer is often made up by resistance and removable brush. When the brush moves along the resistor body, there will be resistance or voltage that has a certain relationship with displacement on the output side (3). Figure shown below is the linear sliding potentiometer and its symbol.





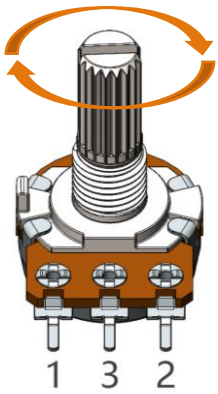
Pin 1 and pin 2 of the potentiometer is connected to two ends of a resistor body respectively, and pins 3 is connected to a brush. When the brush moves from pin 1 to pin 2, resistance between pin 1 and pin 3 will increase up to max resistance of the resistor body linearly, and resistance between pin 2 and pin 3 will decrease down to 0 linearly.

In the circuit, the two sides of resistor body are often connected to the positive and negative electrodes of a power respectively. When you slide the brush of pin 3, you can get a certain voltage in the range of negative voltage to positive voltage of the power supply.



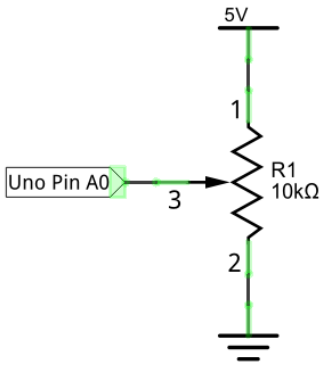
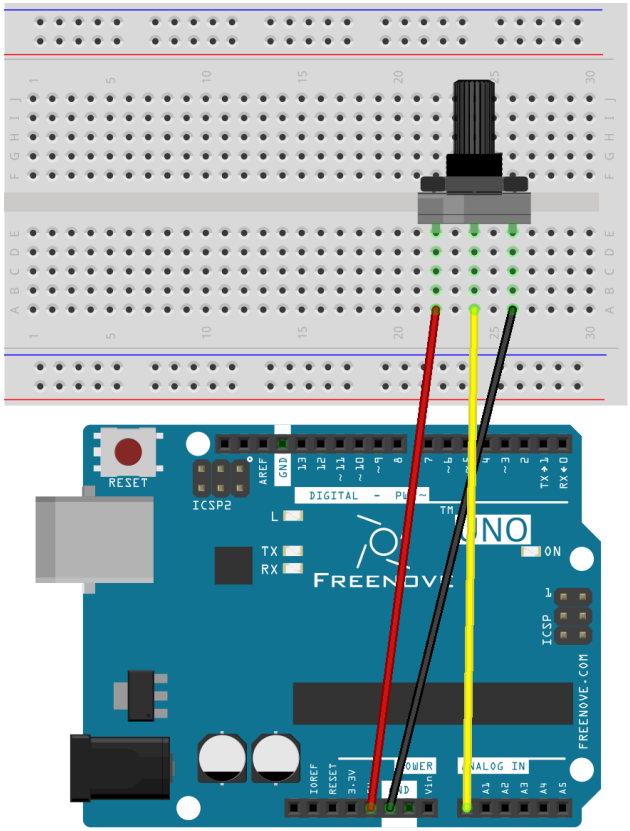
Rotary potentiometer

Rotary potentiometer and linear potentiometer have similar function; the only difference is: the resistance is adjusted through rotating the potentiometer.



Circuit

Use A0 port on Freenove UNO to detect the voltage of rotary potentiometer.

Schematic diagram	Hardware connection
	

Sketch

Sketch 7.1.1

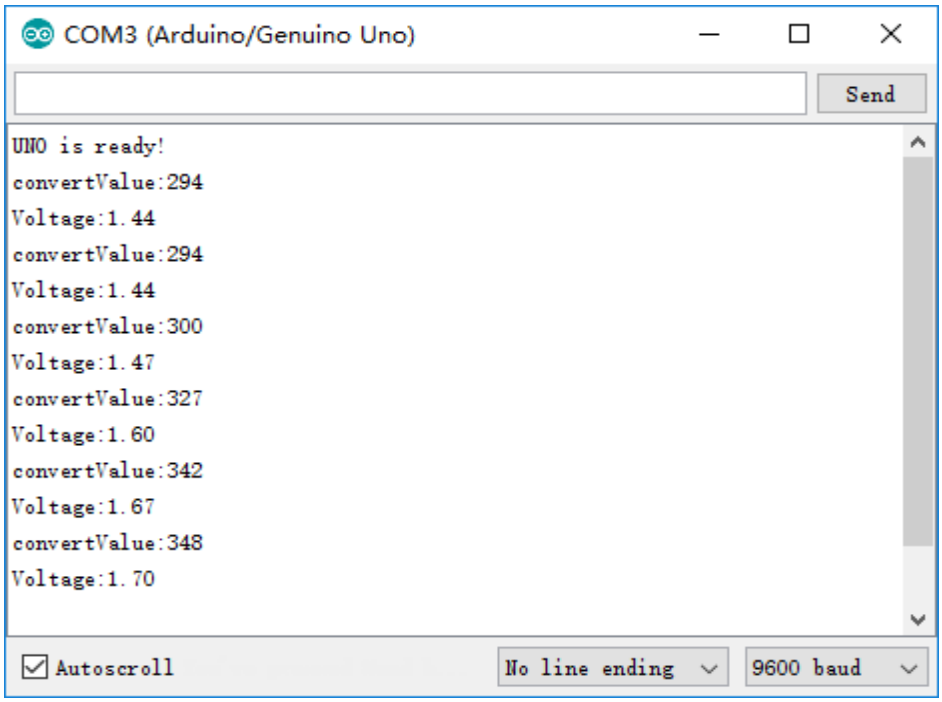
Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino Software through serial port.

```
int adcValue;     // Define a variable to save ADC value
float voltage;    // Define a variable to save the calculated voltage value

void setup() {
  Serial.begin(9600);      // Initialize the serial port and set the baud rate to 9600
  Serial.println("UNO is ready!");    // Print the string "UNO is ready!"
}

void loop() {
  adcValue = analogRead(A0);          // Convert analog of A0 port to digital
  voltage = adcValue * (5.0 / 1023.0);// Calculate voltage according to digital
  // Send the result to computer through serial
  Serial.print("convertValue:");
  Serial.println(adcValue);
  Serial.print("Voltage:");
  Serial.println(voltage);
  delay(500);
}
```

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

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

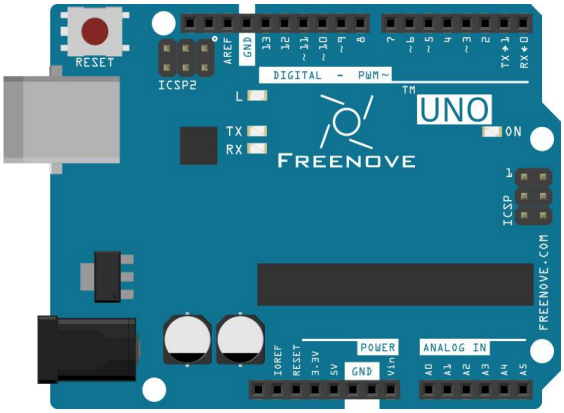
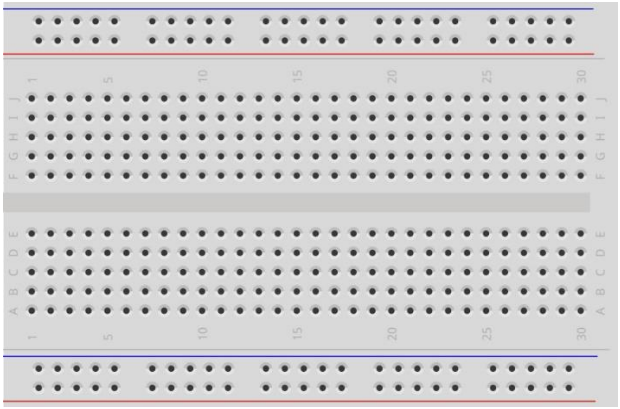
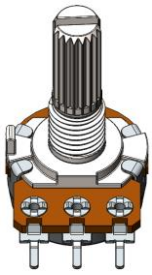
Turn the rotary potentiometer shaft, and you can see the voltage change.



Project 7.2 Control LED by Potentiometer

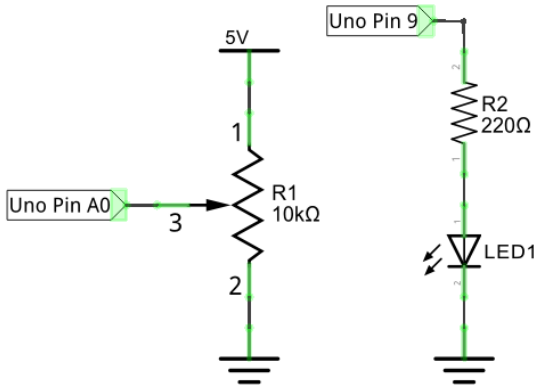
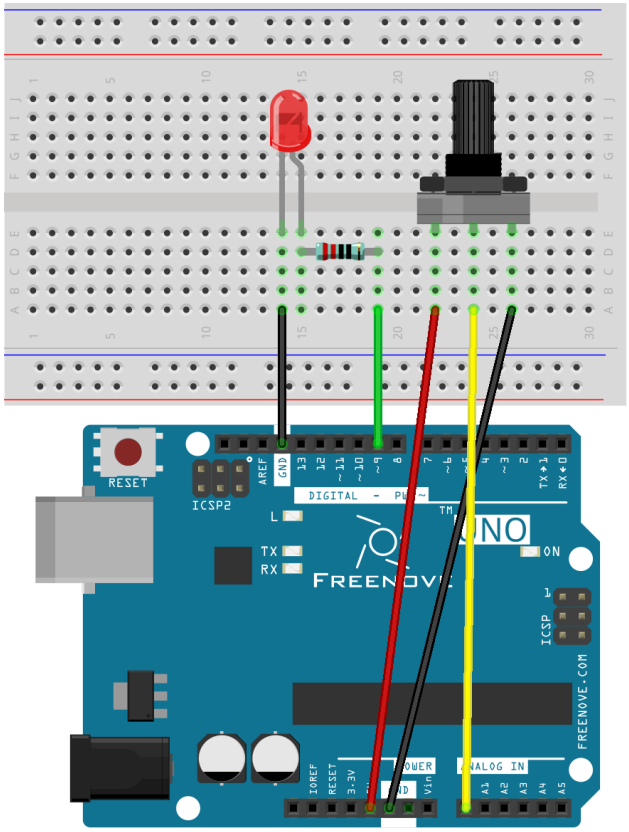
In the last section, we have finished reading ADC value and convert it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

Component list

Freenove UNO x1 	Breadboard x1 	
USB cable x1	Rotary potentiometer x1 	LED x1	Resistor 220Ω x1
Jumper M/M x5		

Circuit

Use A0 on Freenove UNO to detect the voltage of rotary potentiometer, and use D9 to control one LED.

Schematic diagram	Hardware connection
	

Sketch

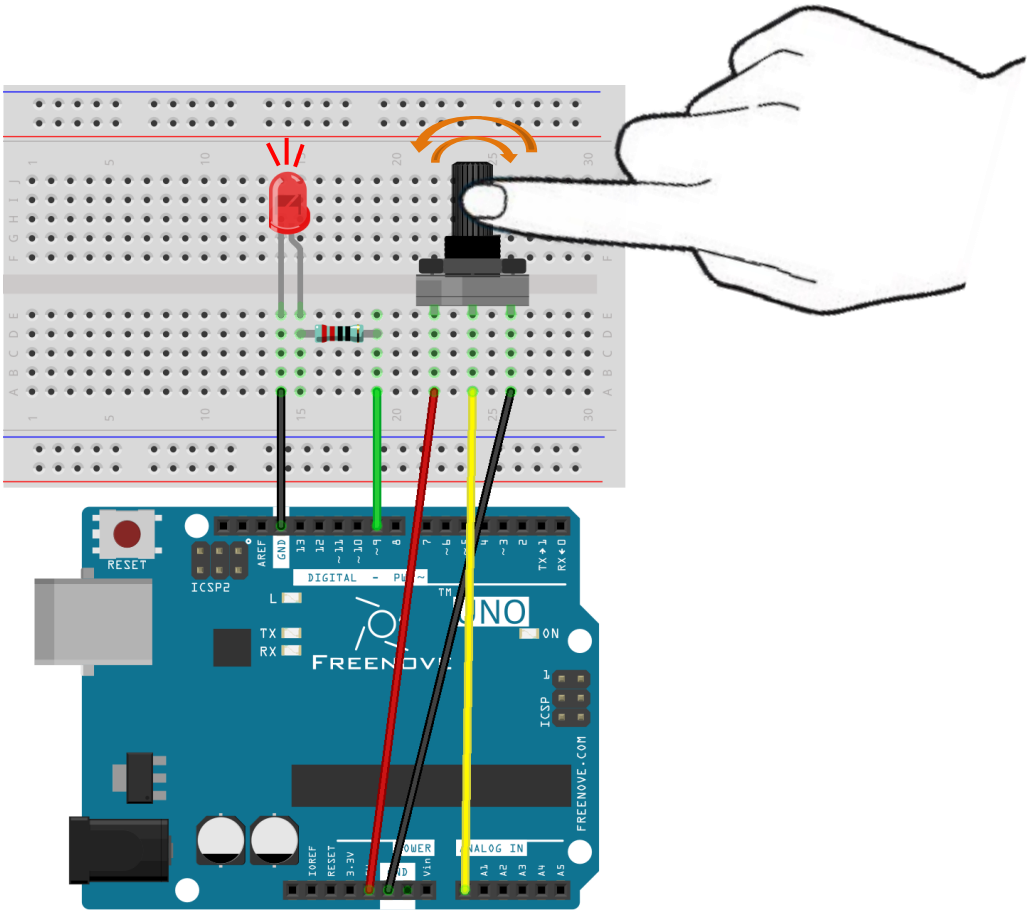
Sketch 7.2.1

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

```
int adcValue;      // Define a variable to save the ADC value
int ledPin = 9;    // Use D9 on Freenove UNO to control the LED

void setup() {
  pinMode(ledPin, OUTPUT);              // Initialize the LED pin as an output
}

void loop() {
  adcValue = analogRead(A0);            // Convert the analog of A0 port to digital
  // Map analog to the 0-255 range, and works as PWM duty cycle of ledPin port
  analogWrite(ledPin, map(adcValue, 0, 1023, 0, 255));
}
```

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

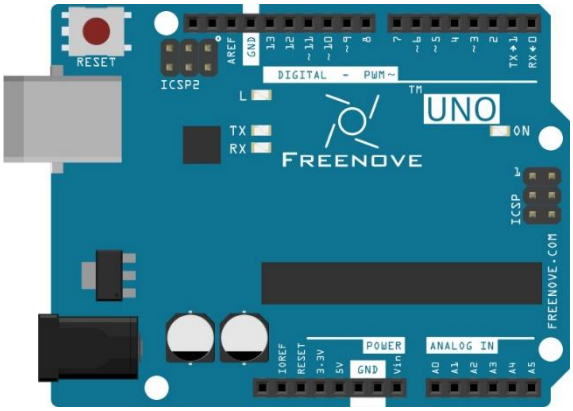
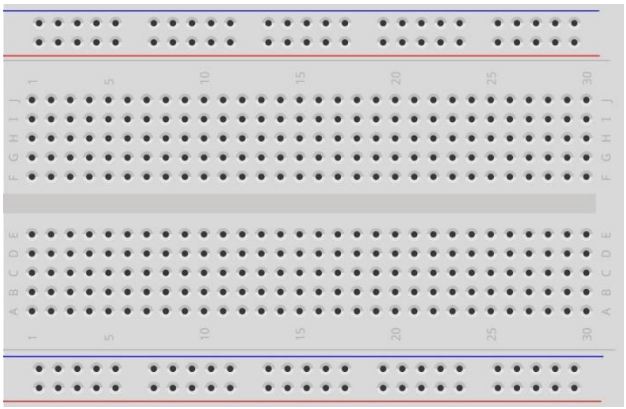
Verify and upload the code, turn the rotary potentiometer shaft, then you will see the LED brightness changes.



Project 7.3 Control LED through Photoresistor

In the last 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.

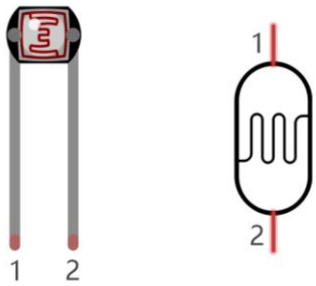
Component list

Freenove UNO x1 	Breadboard x1 			
USB cable x1	Photoresistor x1	LED x1	Resistor 10kΩ x1	Resistor 220Ω x1
Jumper M/M x5				

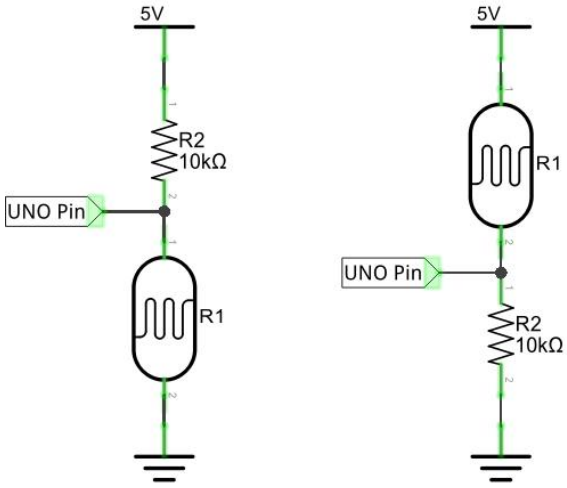
Component knowledge

Photoresistor

Photoresistor is a light sensitive resistor. When the strength that light casts onto the photoresistor surface is not the same, resistance of photoresistor will change. With this feature, we can use photoresistor to detect light intensity. Photoresistor and symbol are as follows.



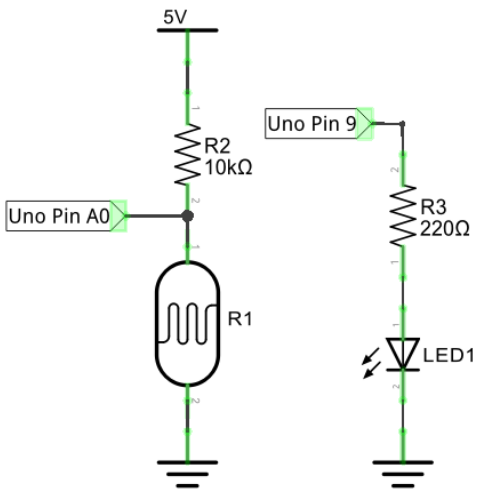
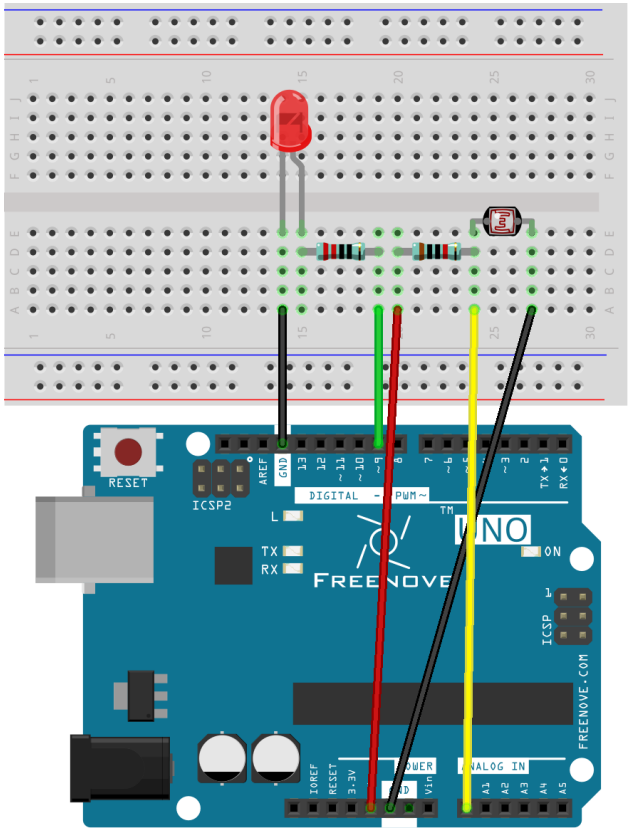
The circuit below is often used to detect the change of photoresistor resistance:



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

Circuit

Use A0 port on Freenove UNO to detect the voltage of photoresistor, and use D9 to control one LED.

Schematic diagram	Hardware connection
	

Sketch

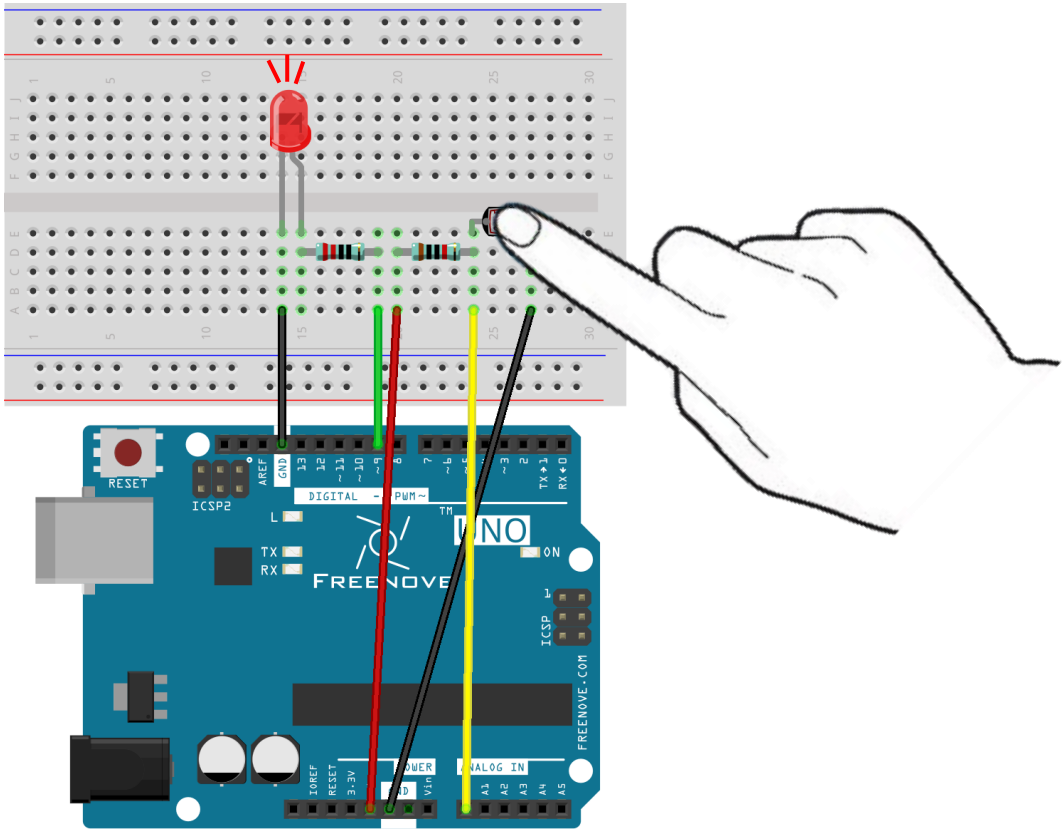
Sketch 7.3.1

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

```
int convertValue; // Define a variable to save the ADC value
int ledPin = 9;   // The number of the LED pin

void setup() {
  pinMode(ledPin, OUTPUT);  // Set ledPin into output mode
}

void loop() {
  convertValue = analogRead(A0);  // Read analog voltage value of A0 port, and save
  // Map analog to the 0-255 range, and works as ledPin duty cycle setting
  analogWrite(ledPin, map(convertValue, 0, 1023, 0, 255));
}
```

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

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



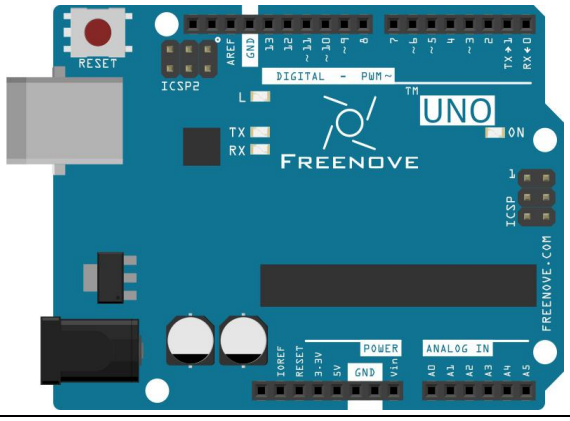
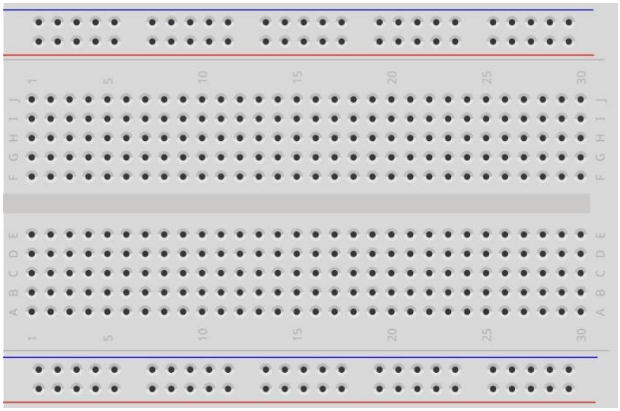
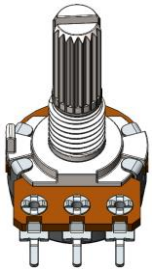
Chapter 8 RGB LED

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

Project 8.1 Control RGB LED through Potentiometer

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

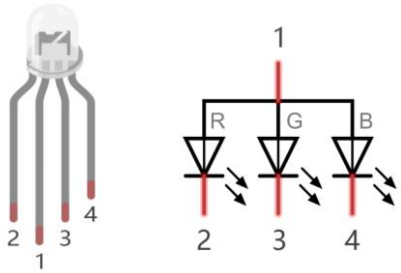
Component list

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

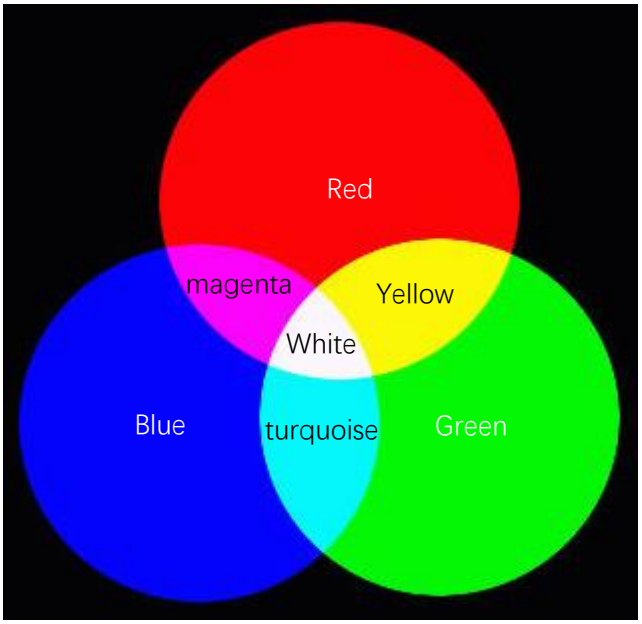
Component knowledge

RGB LED

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light, and it has 4 pins. The long pin (1) is the common port, that is, 3 LED 's cathode or anode. The RGB LED with common anode and its symbol are as follows. By controlling these 3 LED to emit light with different brightness, we can make RGB LED emit various colors of light.



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

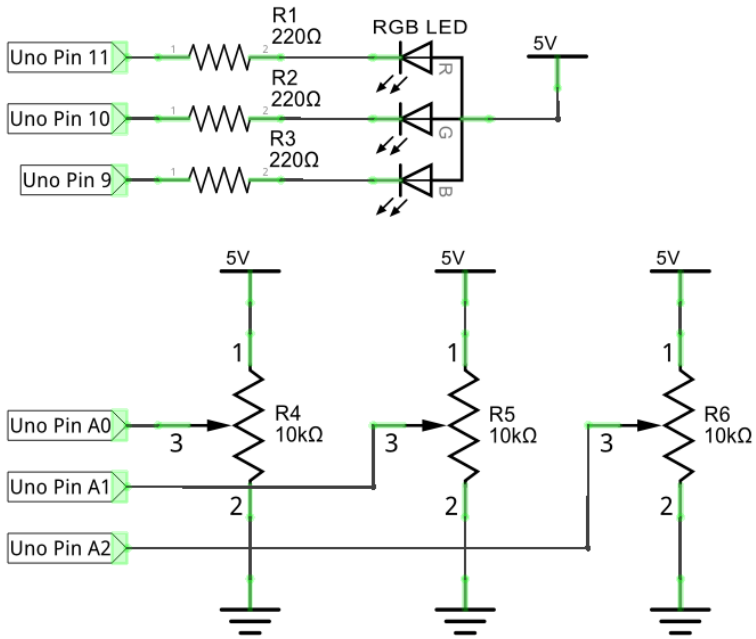


We know from previous section that, UNO controls LED to emit a total 256(0-255) different brightness by PWM. So, through the combination of three different colors of LED, RGB LED can emit 256^3=16777216 Colors, 16Million colors.

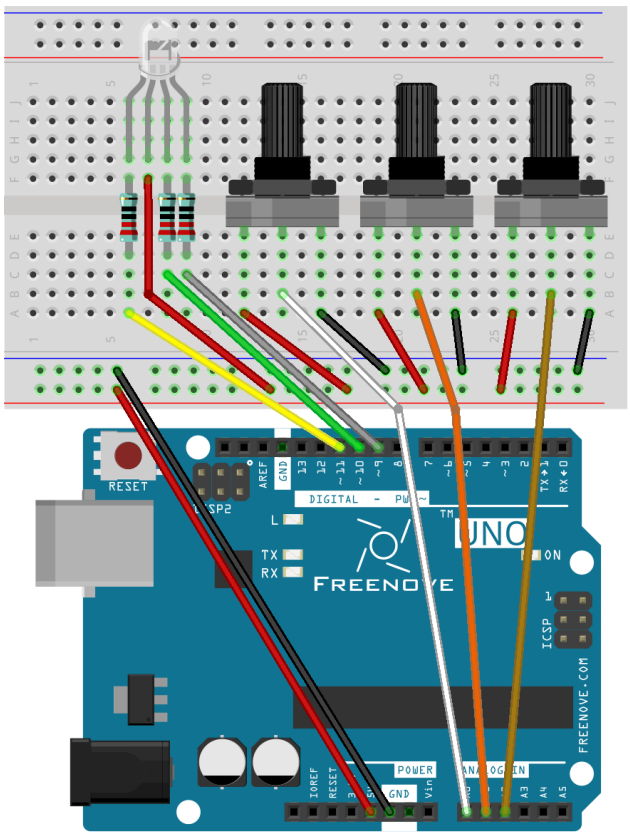
Circuit

Use A0, A1, A2 port of Freenove UNO to detect the voltage of rotary potentiometer, and control RGB LED by D9, D10, D11.

Schematic diagram



Hardware connection



Sketch

Sketch 8.1.1

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

```
// set pin numbers:
int ledPinR = 11;   // the number of the LED R pin
int ledPinG = 10;   // the number of the LED G pin
int ledPinB = 9;    // the number of the LED B pin

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPinR, OUTPUT);
  pinMode(ledPinG, OUTPUT);
  pinMode(ledPinB, OUTPUT);
}

void loop() {
  int adcValue;     // Define a variable to save the ADC value
  // Convert analog of A0 port into digital, and work as PWM duty cycle of ledPinR port
  adcValue = analogRead(A0);
  analogWrite(ledPinR, map(adcValue, 0, 1023, 0, 255));
  // Convert analog of A1 port into digital, and work as PWM duty cycle of ledPinG port
  adcValue = analogRead(A1);
  analogWrite(ledPinG, map(adcValue, 0, 1023, 0, 255));
  // Convert analog of A2 port into digital, and work as PWM duty cycle of ledPinB port
  adcValue = analogRead(A2);
  analogWrite(ledPinB, map(adcValue, 0, 1023, 0, 255));
}
```

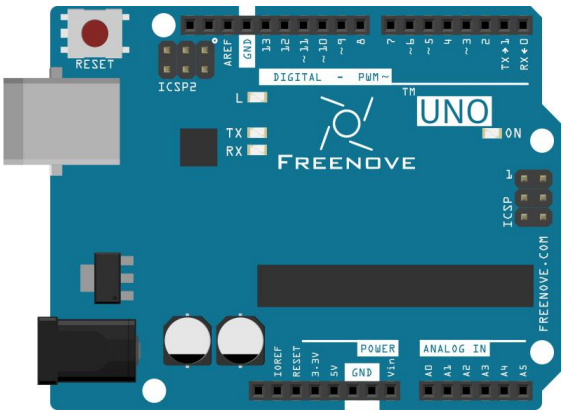
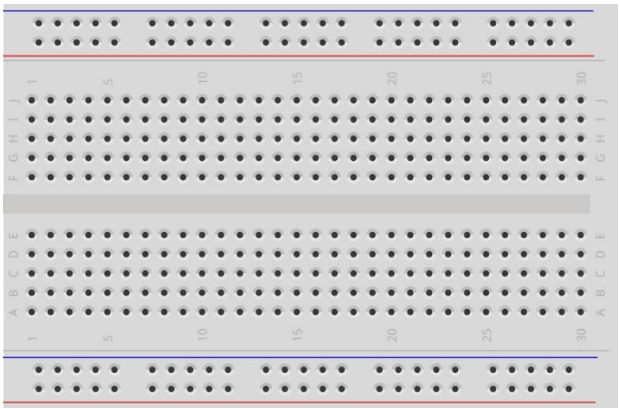
In the code, we get the voltage of three rotary potentiometers, and convert that into PWM duty cycle to control these three LED of the RGB LED to emit light with different brightness.

Verify and upload the code, turn the three rotary potentiometer shaft, then you can see the LED light changes in color and brightness.

Project 8.2 Colorful LED

From 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 colorful light automatically.

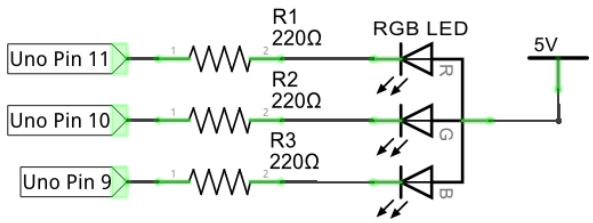
Component list

Freenove UNO x1 	Breadboard x1 	
USB cable x1	RGB LED x1	Resistor 220Ω x3
Jumper M/M x4		

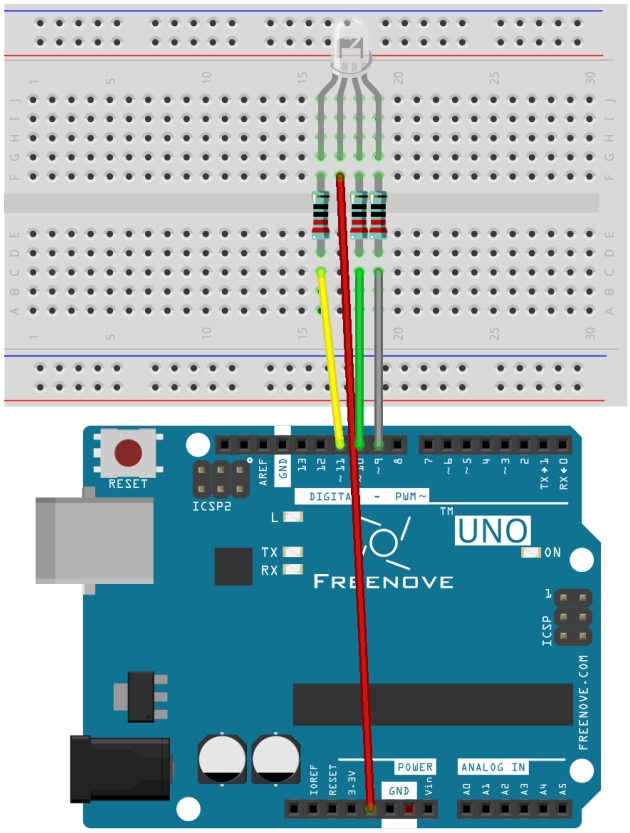
Circuit

Use D9, D10, D11 of Freenove UNO to control RGB LED.

Schematic diagram



Hardware connection



Sketch

Sketch 8.2.1

Now, write code to generate three random number, and convert that into PWM duty cycle to control these three LED of RGB LED.

```
// set pin numbers:
int ledPinR = 11;   // the number of the LED R pin
int ledPinG = 10;   // the number of the LED G pin
int ledPinB = 9;    // the number of the LED B pin

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPinR, OUTPUT);
  pinMode(ledPinG, OUTPUT);
  pinMode(ledPinB, OUTPUT);
}

void loop() {
  // Generate three random numbers between 0-255 as the output PWM duty cycle of ledPin
  rgbLedDisplay(random(256), random(256), random(256));
  delay(500);
}

void rgbLedDisplay(int red, int green, int blue) {
  // Set three ledPin to output the PWM duty cycle
  analogWrite(ledPinR, constrain(red, 0, 255));
  analogWrite(ledPinG, constrain(green, 0, 255));
  analogWrite(ledPinB, constrain(blue, 0, 255));
}
```

In the code, we create three random number, and convert that into PWM duty cycle, controlling these three LED 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, then RGB LED starts flashing with different colors and brightness.

Chapter 9 Buzzer

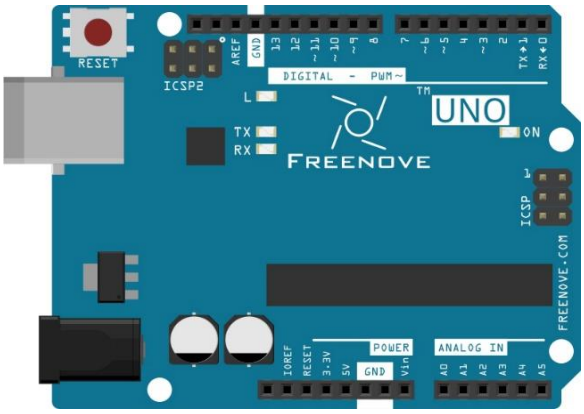
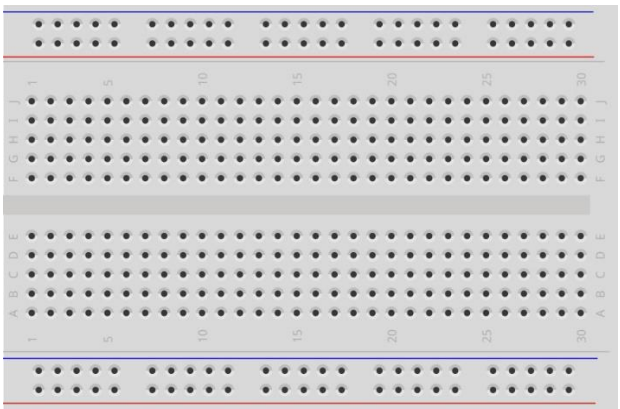
Before, we have used Freenove UNO and basic electronic components to carry out a series of interesting projects. Now, let's learn how to use some integrated electronic components and modules. These modules are usually integrated with a number of electronic components, so it has special features and uses.

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

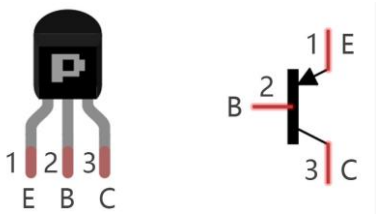
Freenove UNO x1 		Breadboard x1 		
USB cable x1		Jumper M/M x6		
NPN transistorx1	Active buzzer x1	Push button x1	Resistor 1kΩ x1	Resistor 10kΩ x2

Component knowledge

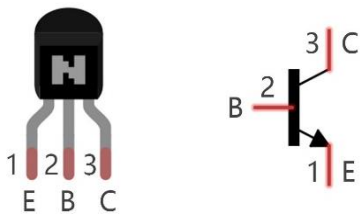
Transistor

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

PNP transistor



NPN transistor



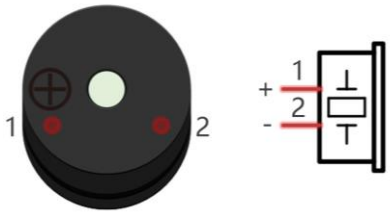
(The transistor marked 8550 is PNP, and 8050 is NPN.)

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

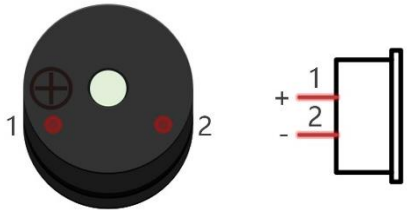
Buzzer

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

Active buzzer



Passive buzzer

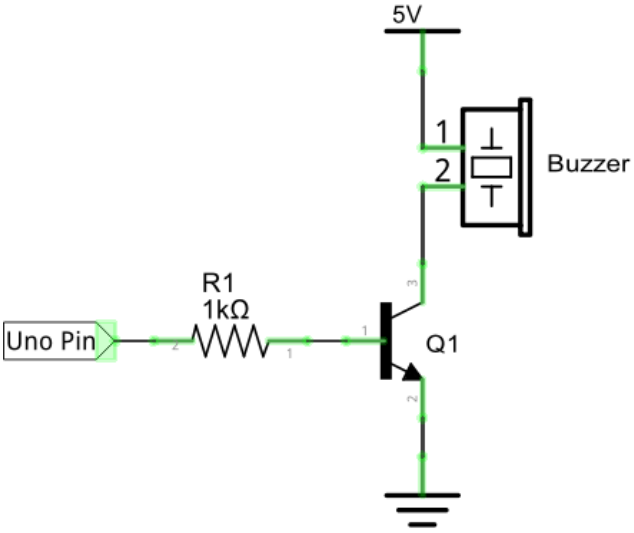


(A white label is attached to the active buzzer)

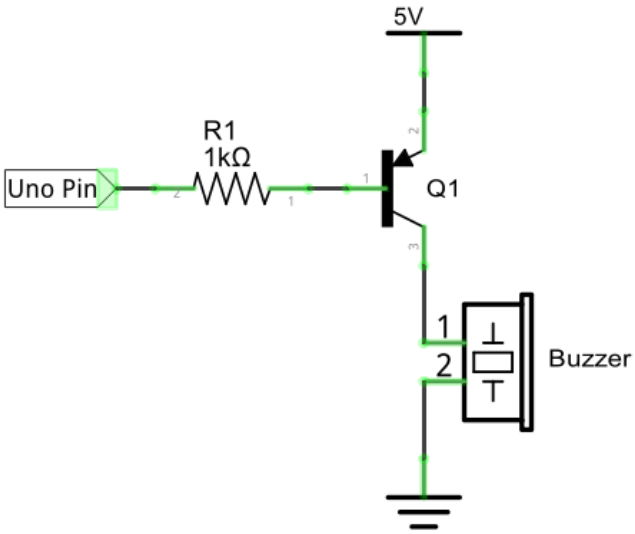
Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer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

The attached active buzzer is carrying a white label, and the passive buzzer is not labelled.
Buzzer needs larger current when it works. But generally, microcontroller port cannot provide enough current for that. In order to control buzzer by UNO, transistor can be used to drive a buzzer indirectly.

When use NPN transistor to drive buzzer, we often adopt the following method. If UNO pin outputs high level, current will flow through R1, the transistor gets conducted, and the buzzer make a sound. If UNO pin outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.



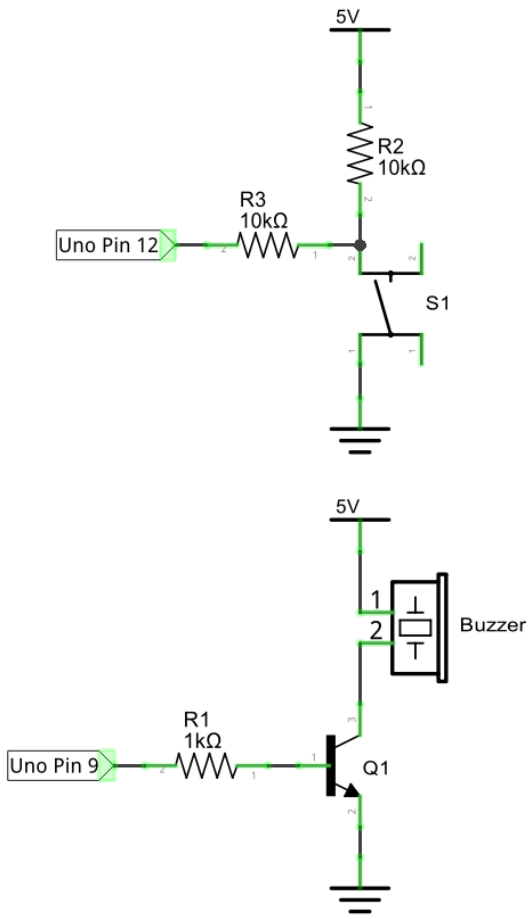
When use PNP transistor to drive buzzer, we often adopt the following method. If UNO pin outputs low level, current will flow through R1, the transistor gets conducted, buzzer make a sound. If UNO pin outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.



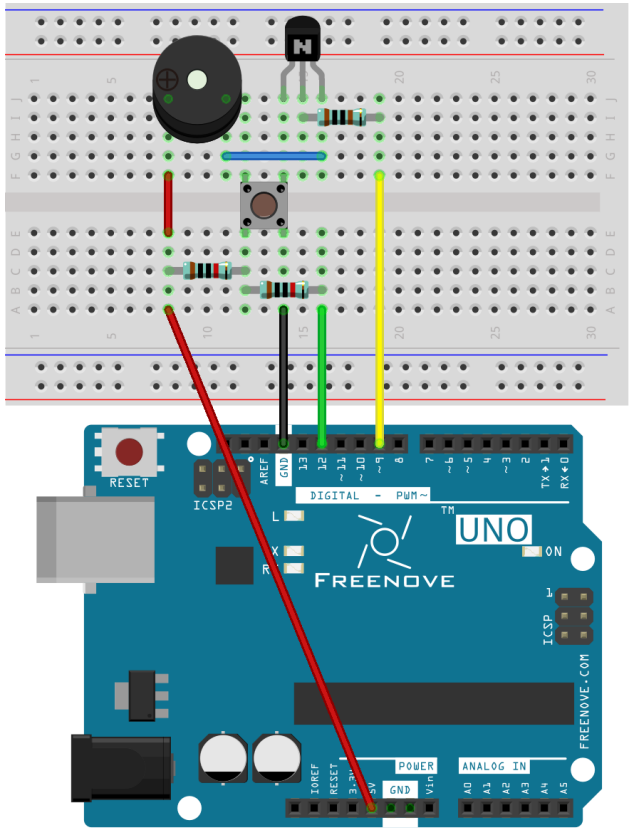
Circuit

Use D12 port of Freenove UNO to detect the state of push button, and D9 to drive active buzzer.

Schematic diagram



Hardware connections



Sketch

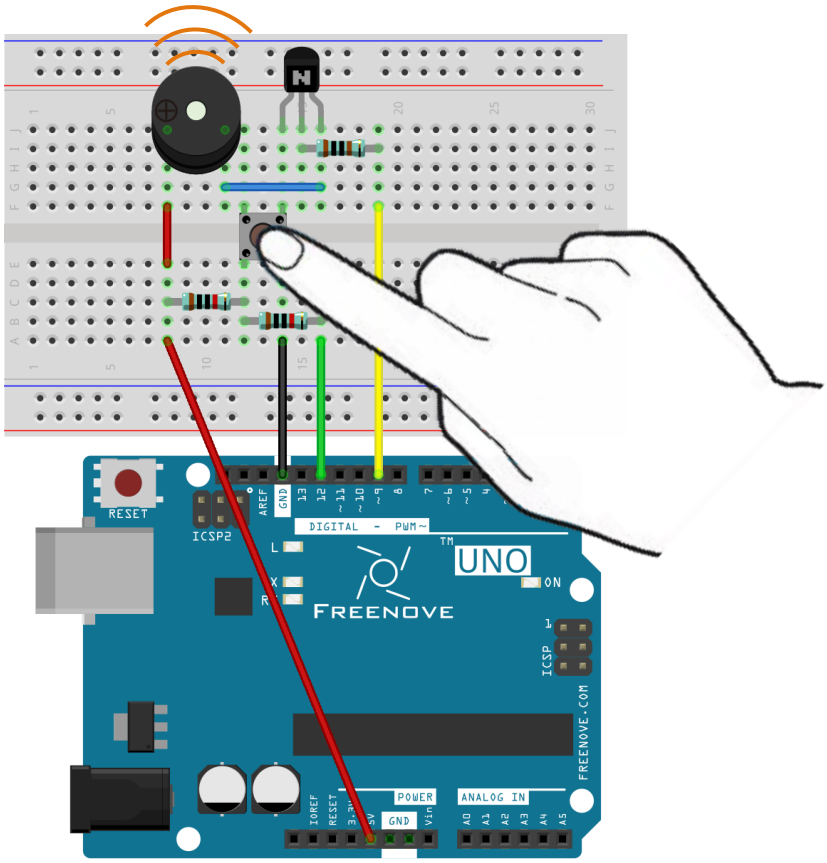
Sketch 9.1.1

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

```
int buttonPin = 12;   // the number of the push button pin
int buzzerPin = 9;    // the number of the buzzer pin

void setup() {
  pinMode(buttonPin, INPUT);  // Set push button pin into input mode
  pinMode(buzzerPin, OUTPUT); // Set Buzzer pin into output mode
}

void loop() {
 if (digitalRead(buttonPin) == HIGH)// If the pin is high level,the button is not pressed.
    digitalWrite(buzzerPin, LOW);     // Turn off Buzzer
  else                                // The button is pressed, if the pin is low level
    digitalWrite(buzzerPin, HIGH);    // Turn on Buzzer
}
```

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

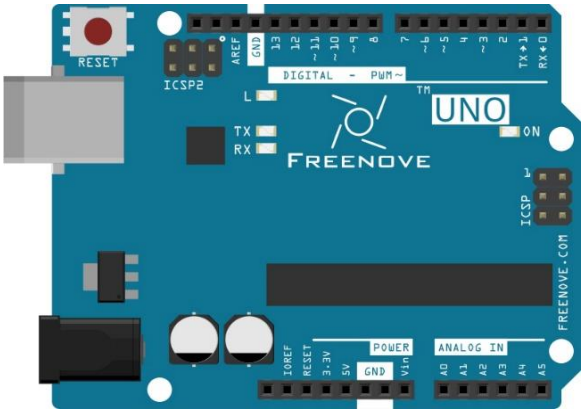
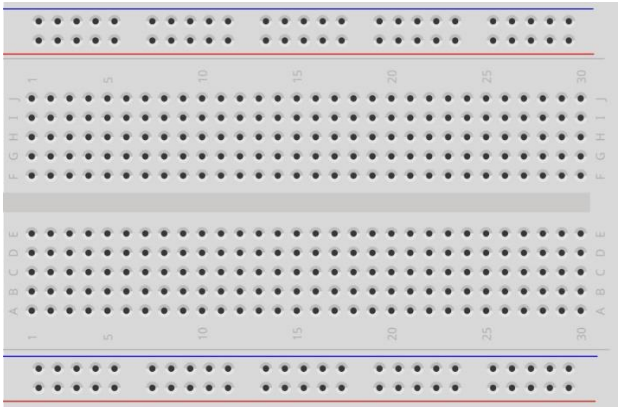
Verify and upload the code, press the push button, then the active buzzer will make a sound.



Project 9.2 Passive Buzzer

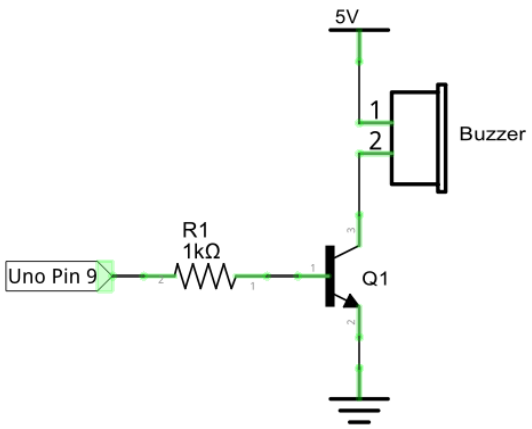
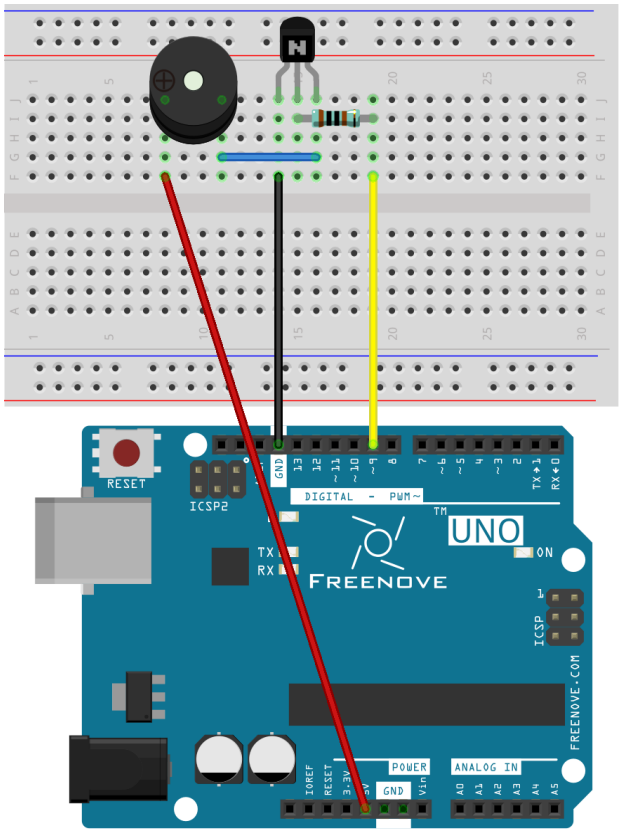
In the last section, we have finished using transistor to driven active buzzer to beep. Now, we will try to use passive buzzer to make a sound with different frequency.

Component list

Freenove UNO x1 	Breadboard x1 		
USB cable x1	NPN transistor x1	Passive buzzer x1	Resistor 1kΩ x1
Jumper M/M x4			

Circuit

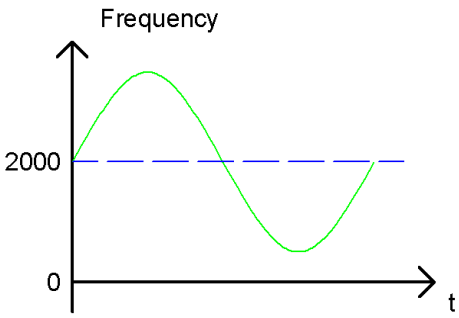
Use D9 port of Freenove UNO to drive a passive buzzer.

Schematic diagram	Hardware connection
	

Sketch

Sketch 9.2.1

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



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

```
int buzzerPin = 9;     // the number of the buzzer pin
float sinVal;          // Define a variable to save sine value
int toneVal;           // Define a variable to save sound frequency

void setup() {
  pinMode(buzzerPin, OUTPUT); // Set Buzzer pin to output mode
}

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

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

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

The parameter of sin() function is radian, so we should convert unit of π from angle to radian first before using it.

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.

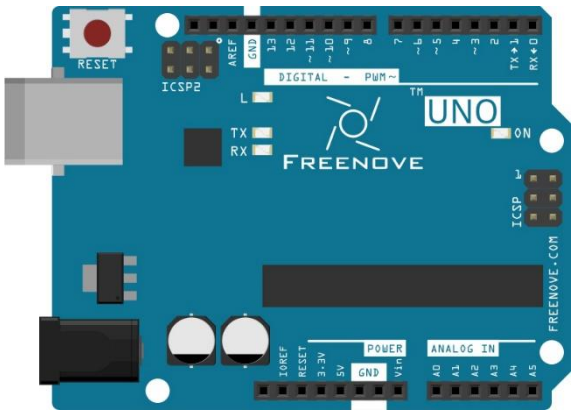
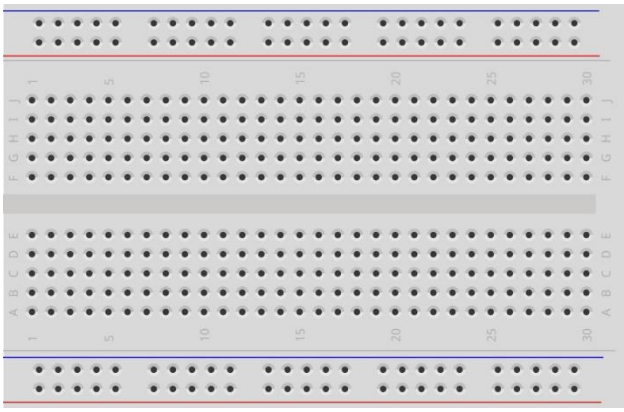
Chapter 10 Temperature Sensor

Before, we have used Freenove UNO 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.

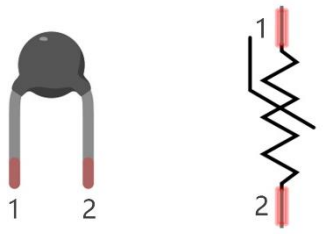
Component list

Freenove UNO 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 the temperature changes, resistance of thermistor will change. With this feature, we can use thermistor to detect temperature intensity. Thermistor and symbol are as follows.



The relationship between resistance value and temperature of thermistor is:

Rt=R*EXP[B*(1/T2-1/T1)]

Where:

Rt is the thermistor resistance under T2 temperature;

R is in the nominal resistance of thermistor under T1 temperature;

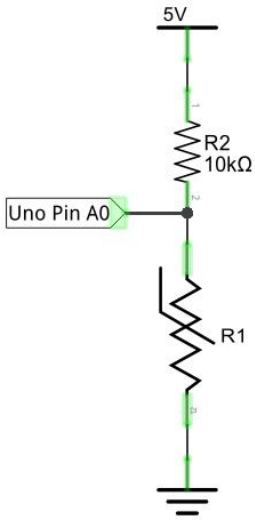
EXP[n] is nth power of E;

B is for thermal index;

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

Parameters of the thermistor we use is: B=3950, R=10k, T1=25.

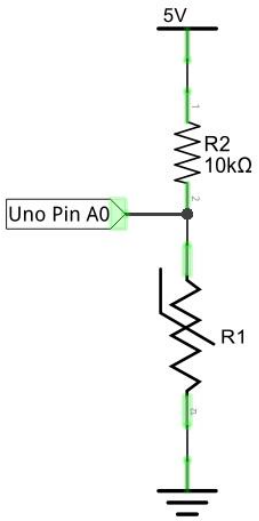
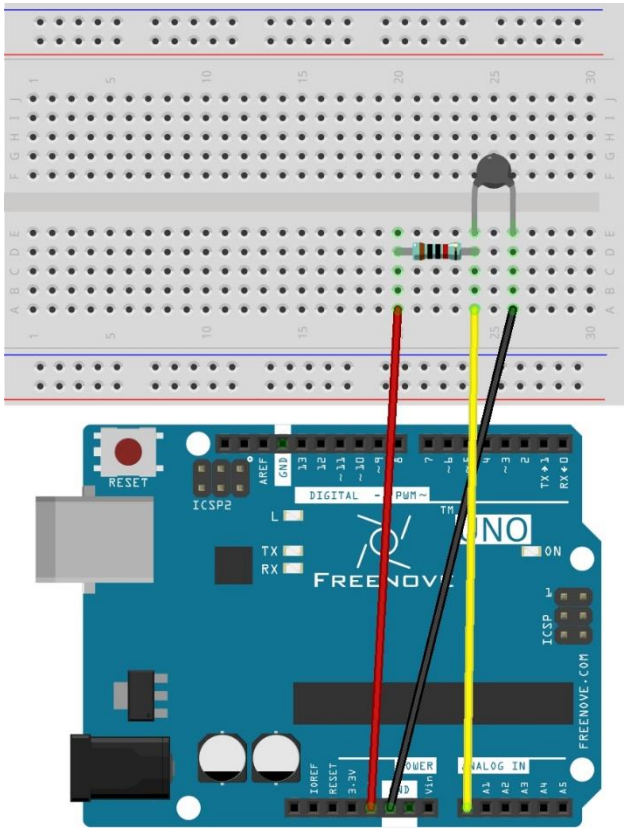
The circuit connection method of the thermistor is similar to photoresistor, like the following method:



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

Circuit

Use A0 port on Freenove UNO to detect the voltage of thermistor.

Schematic diagram	Hardware connection
	

Sketch

Sketch 10.1.1

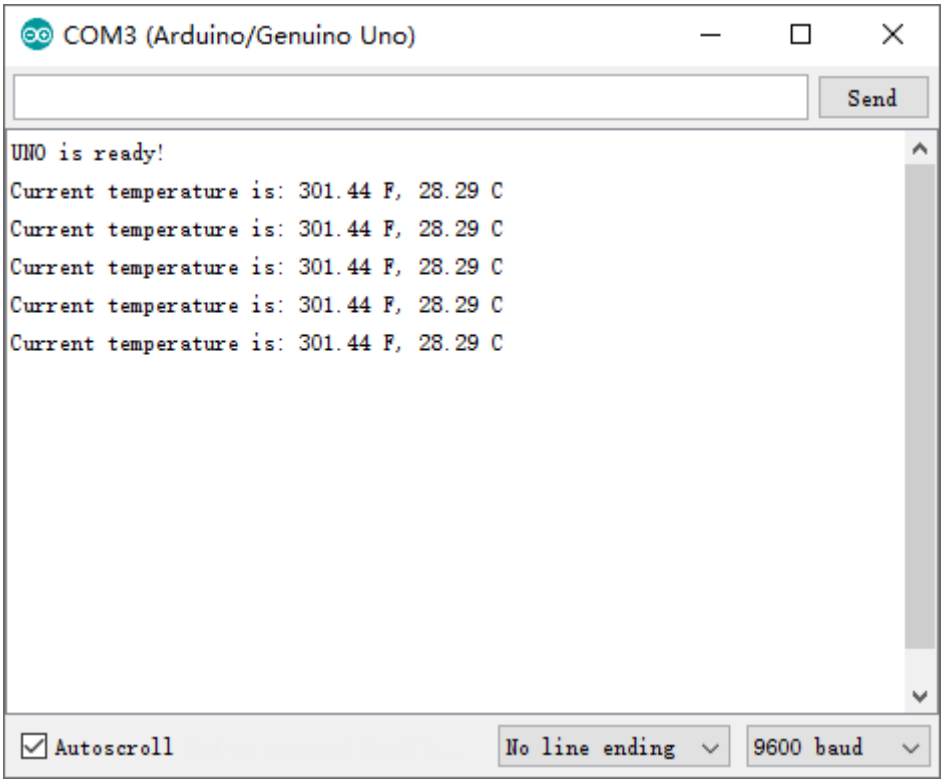
Now, write the code to detect the voltage value of thermistor, calculate the temperature value, and send it to Serial Monitor window.

```
void setup() {
  Serial.begin(9600);                 // Initialize the serial port, set the baud rate into 9600
  Serial.println("UNO is ready!");    // Print the string "UNO is ready!"
}

void loop() {
  // Convert analog value of A0 port into digital value
  int adcVal = analogRead(A0);
```

```
  // Calculate voltage
  float v = adcVal * 5.0 / 1024;
  // Calculate resistance value of thermistor
  float Rt = 10 * v / (5 - v);
  // Calculate temperature (Kelvin)
  float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));
  // Calculate temperature (Celsius)
  float tempC = tempK - 273.15;

  // Send the result to computer through serial port
  Serial.print("Current temperature is: ");
  Serial.print(tempK);
  Serial.print(" K, ");
  Serial.print(tempC);
  Serial.println(" C");
  delay(500);
}
```

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

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



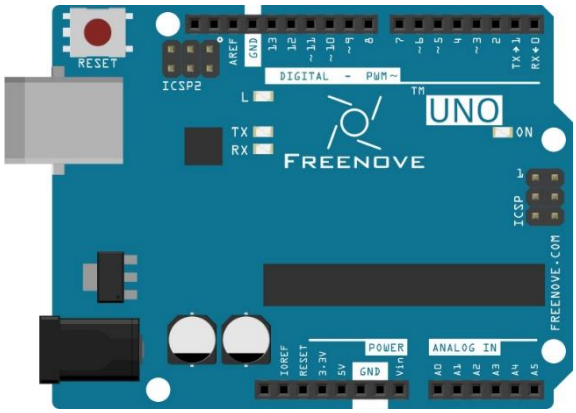
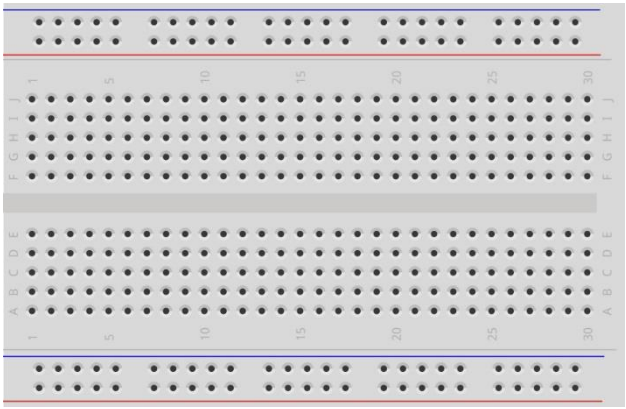
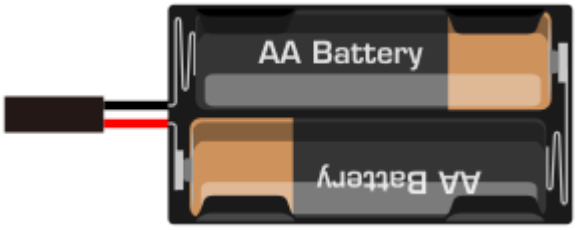
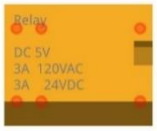
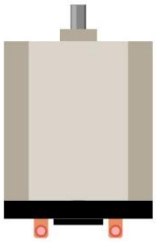
Chapter 11 Motor

Before, we have done a series of interesting projects with Freenove UNO and basic electronic components. Now, let's study some movable electronic modules. In this chapter, we will learn to control the motor.

Project 11.1 Control Motor by Relay

First, use relay to control the Motor.

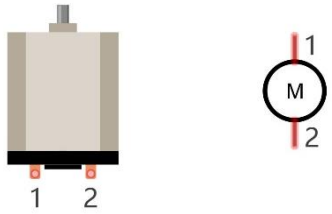
Component list

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

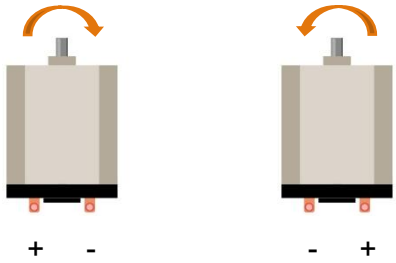
Component knowledge

Motor

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



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



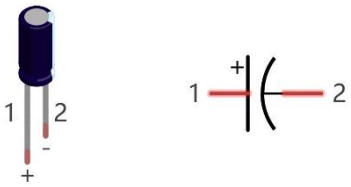
Capacitor

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

Capacitor is an energy storage device, with a certain capacitance. When capacitor voltage increases, the capacitor will be charged. And capacitor will discharge when the voltage drops. So capacitor voltage of both ends is not transient. According to this characteristic, capacitor is often used to stabilize the voltage of power supply, and filters 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.

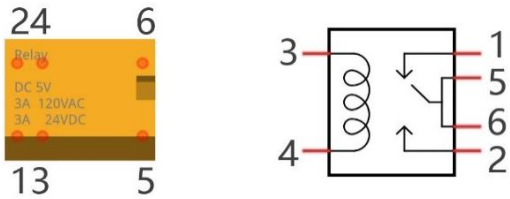
For polar capacitor, it usually has larger capacitance, and an electrolytic polar capacitor of that is shown below:



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

Relay is an electrical switch, which consists of coils and contacts. When the coil is energized, it will form the electromagnet, attracting contacts to close. Coils and contacts are independent of each other, so other independent circuit can be controlled by relay, through using a small current to control the circuit with large current. Diagram below is a relay with control voltage of 5V.



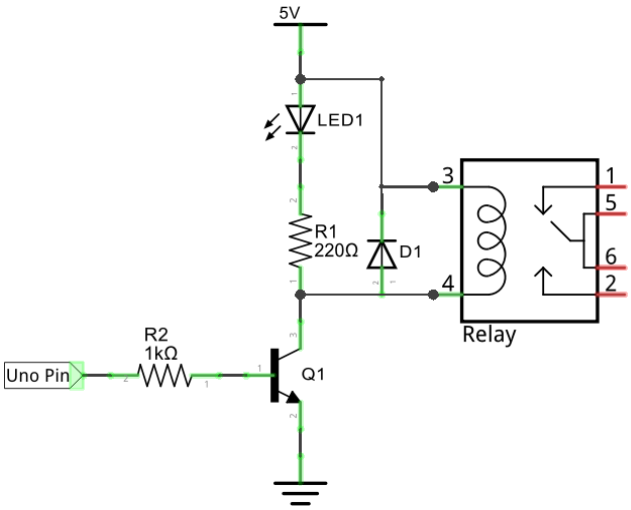
Pin 5 and pin 6 are connected to each other inside. When the coil pin3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

Inductor

The unit of inductance(L) is the henry (H). 1H=1000mH, 1mH=1000μH.

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

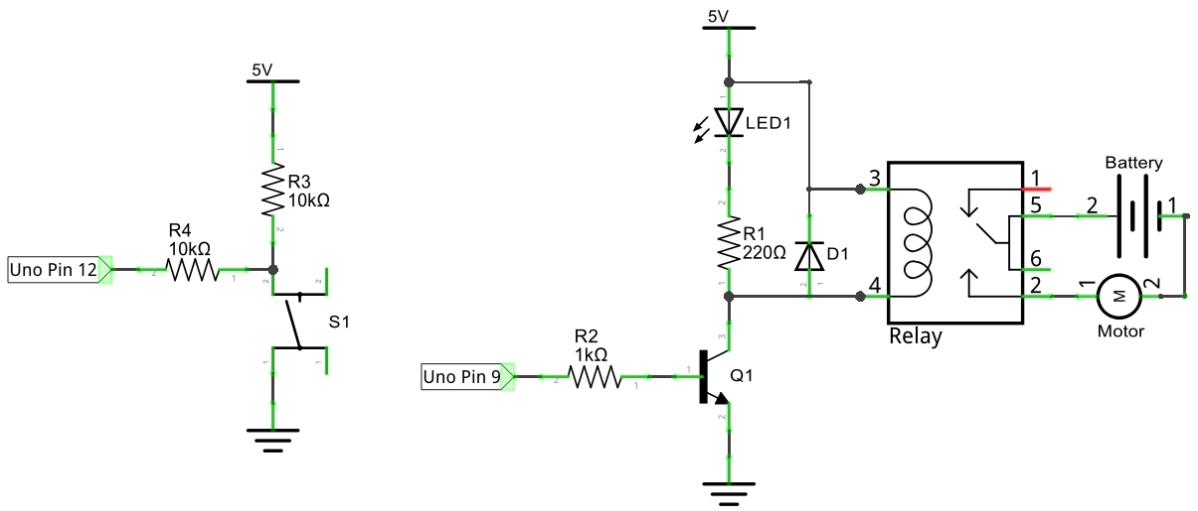
The reference circuit for relay is as follows. The coil of relay can be equivalent to inductor, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to both ends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



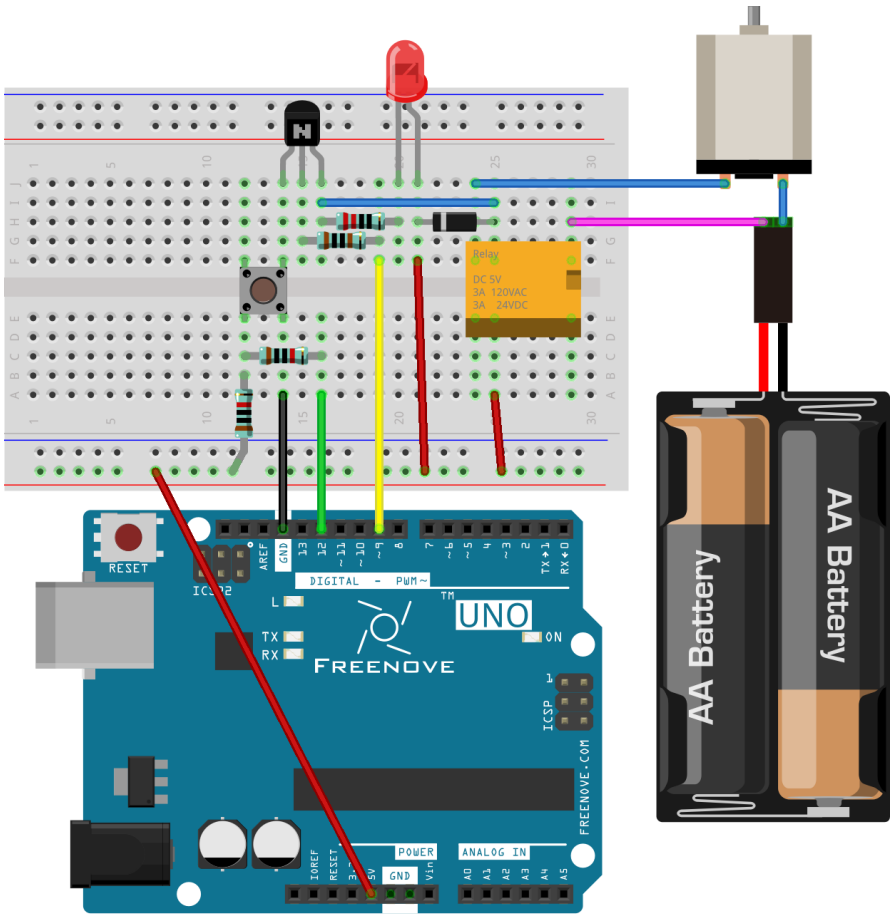
Circuit

Use D12 of Freenove UNO to detect the state of push button, and D9 to control the relay. As motor running needs larger power, we will use two AA batteries to supply power for the motor alone.

Schematic diagram



Hardware connections



Sketch

Sketch 11.1.1

Now, write code to detect the state of push button. 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.

```
int relayPin = 9;      // the number of the relay pin
int buttonPin = 12;    // the number of the push button pin

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

void setup() {
  pinMode(buttonPin, INPUT);   // Set push button pin into input mode
  pinMode(relayPin, OUTPUT);   // Set relay pin into output mode
  digitalWrite(relayPin, relayState);  // Set the initial state of relay into "off"
  Serial.begin(9600);                  // Initialize serial port,and set baud rate to 9600
}

void loop() {
  int nowButtonState = digitalRead(buttonPin);  // Read current state of button pin
  // If button pin state has changed, record the time point
  if (nowButtonState != lastButtonState) {
    lastChangeTime = millis();
  }
  // If button state changes, and stays stable for a while, then it should have skipped the bounce area
  if (millis() - lastChangeTime > 10) {
    if (buttonState != nowButtonState) {  // Confirm button state has changed
      buttonState = nowButtonState;
      if (buttonState == LOW) {      // Low level indicates the button is pressed
        relayState = !relayState;            // Reverse relay state
        digitalWrite(relayPin, relayState);  // Update relay state
        Serial.println("Button is Pressed!");
      }
      else {                         // High level indicates the button is released
        Serial.println("Button is Released!");
      }
    }
  }
  lastButtonState = nowButtonState;  // Save the state of last button
}
```

In this code, we used a new method to detect button state. In the loop() function, the level state of button pin is detected constantly. When the level is changed, record this time. If the level has not changed after a while, it will be considered 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 button and relay.

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

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

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

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

```
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.

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

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

```
  if (millis() - lastChangeTime > 10) {
    ...
  }
```

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

```
    if (buttonState != nowButtonState) {  // Confirm button state has changed
      buttonState = nowButtonState;
      ...
    }
```

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.

```
      if (buttonState == LOW) {      // Low level indicates the button is pressed
        relayState = !relayState;            // Reverse relay state
        digitalWrite(relayPin, relayState); // Update relay state
        Serial.println("Button is Pressed!");
      }
      else {                         // High level indicates the button is released
        Serial.println("Button is Released!");
      }
```

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 Arduino 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 by L293D

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

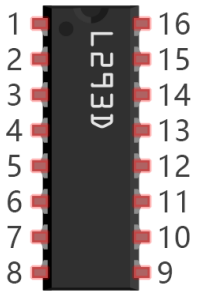
Component list

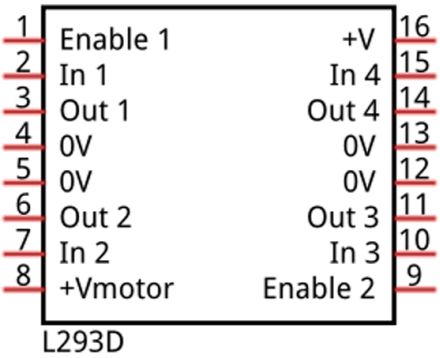
Freenove UNO x1	Breadboard x1		
USB cable x1	Jumper M/M x10		
AA Battery holder x1	Motor x1	L293D x1	Rotary potentiometer x1

Component knowledge

L293D

L293D is a chip integrated with 4-channel motor drive. You can drive a unidirectional motor with 4 ports or a bi-directional motor with 2 port or a stepper motor.



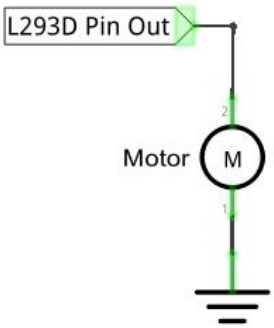


Port description of L293D module is as follows:

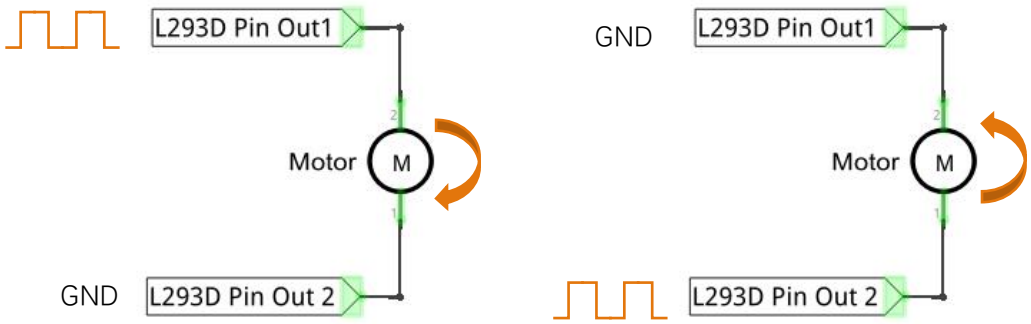
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4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



Following connection uses two channels: one channel outputs PWM wave, and another channel connects GND, so you can control the speed of motor. When these two channel signals are exchanged, the current direction of the motor can be reversed, and the motor will rotate in reverse direction. This can not only control the speed of motor, but also can control the steering of motor.

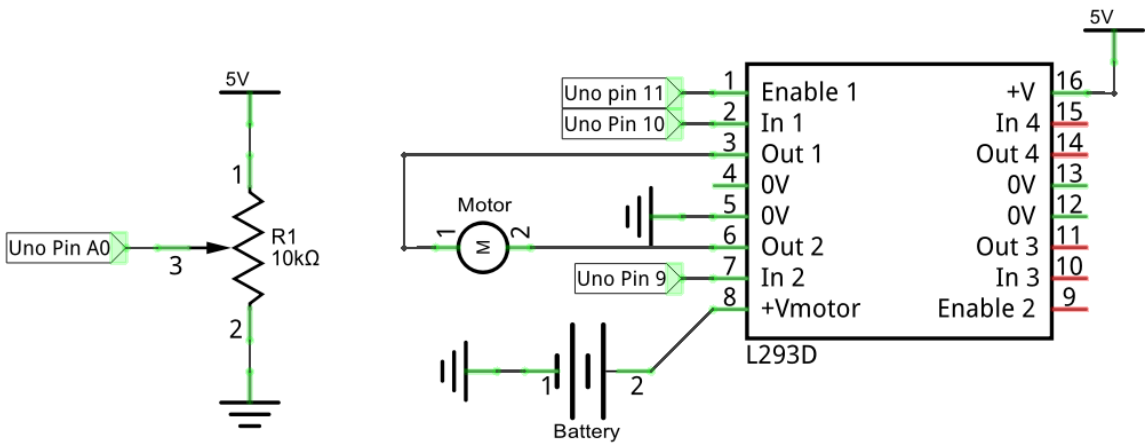


In actual use, motor is usually connected to the channel 1 and 2, output different level to in1 and in2 to control the rotation direction of the motor, and output PWM wave to Enable1 port to control the motor rotation speed. Or, get motor connected to the channel 3 and 4, output different level to in3 and in4 to control the motor's rotation direction, and output PWM wave to Enable2 pin to control the motor rotation speed.

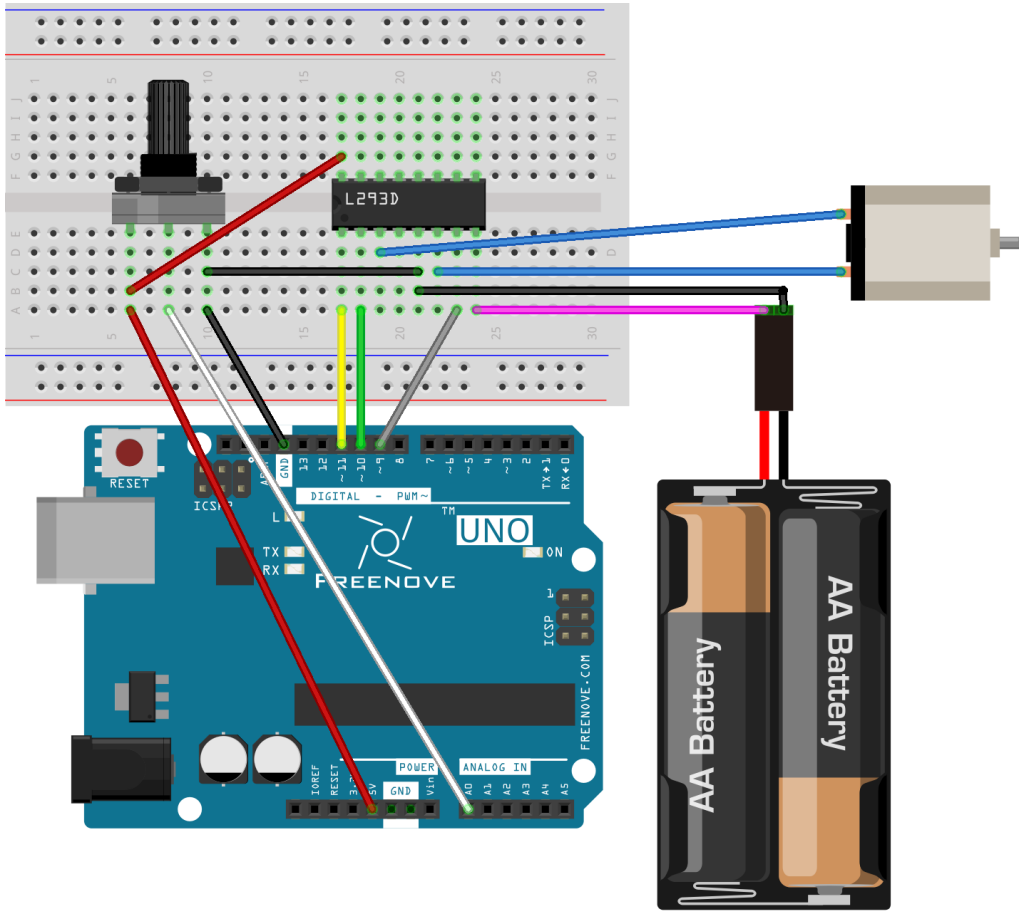
Circuit

Use A0 of Freenove UNO to detect the voltage of rotary potentiometer; D9, D10 to control the motor's rotation direction, D11 to output PWM wave to control motor speed.

Schematic diagram



Hardware connection



Sketch

Sketch 11.2.1

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 potentiometer clockwise or counterclockwise, the rotation direction of the motor is different.

```
int in1Pin = 10;        // Define L293D channel 1 pin
int in2Pin = 9;         // Define L293D channel 2 pin
int enable1Pin = 11;    // Define L293D enable 1 pin

boolean rotationDir;    // Define a variable to save the motor's rotation direction, true and false are represented by positive rotation and reverse rotation.
int rotationSpeed;      // Define a variable to save the motor rotation speed

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

void loop() {
  int potenVal = analogRead(A0);   // Convert the voltage of rotary potentiometer into digital
  // Compare the number with value 512, if more than 512, clockwise rotates, otherwise, counter clockwise rotates
  rotationSpeed = potenVal - 512;
  if (potenVal > 512)
    rotationDir = true;
  else
    rotationDir = false;
  // Calculate the motor speed, the far number of deviation from the middle value 512, the faster the control speed will be
  rotationSpeed = abs(potenVal - 512);
  // Control the steering and speed of the motor
  driveMotor(rotationDir,map(rotationSpeed, 0, 512, 0, 255));
}

void driveMotor(boolean dir, int spd) {
  // Control motor rotation direction
  if (rotationDir) {
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  }
  else {
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  }
  // Control motor rotation speed
  analogWrite(enable1Pin, constrain(spd, 0, 255));
}
```

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

```
void driveMotor(boolean dir, int spd) {
  // Control motor rotation direction
  if (rotationDir) {
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  }
  else {
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  }
  // Control motor rotation speed
  analogWrite(enable1Pin, constrain(spd, 0, 255));
}
```

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

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

abs(x)

Computes the absolute value of a number.

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

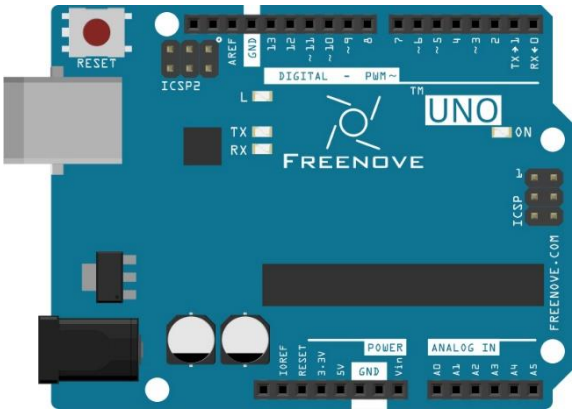
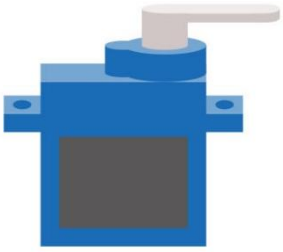
Chapter 12 Servo

Before, we have used Freenove UNO 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.

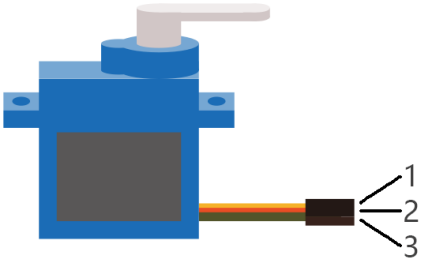
Component list

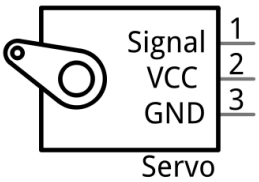
Freenove UNO x1 	USB cable x1
	Jumper M/M x3
	Servo x1 

Component knowledge

Servo

Servo is an auto-control system, consisting of DC motor, reduction gear, sensor and control circuit. Usually, it can rotate in the range of 180 degrees. Servo can output larger torque and is widely used in model airplane, robot and so on. It has three lines, including two for electric power line positive (2-VCC, red), negative (3-GND, brown), and the signal line (1-Signal, orange).





We use 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:

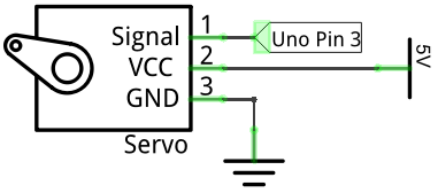
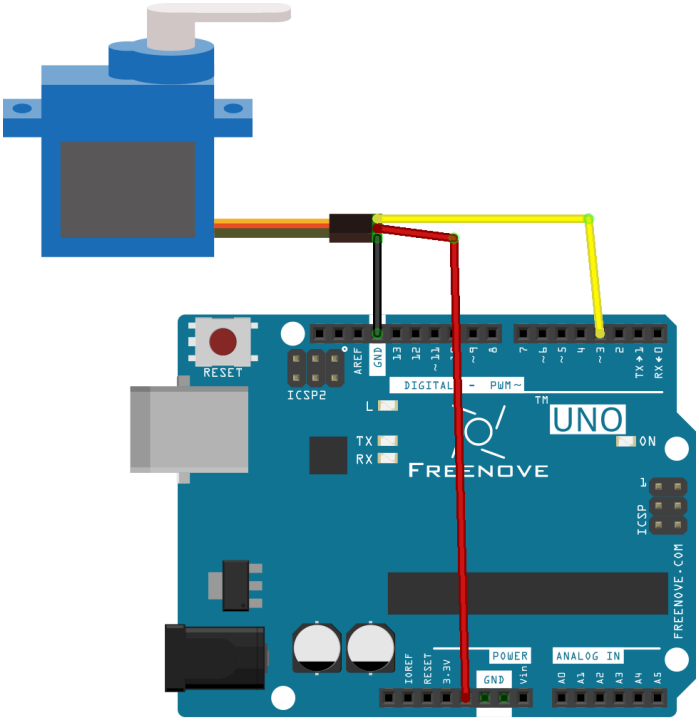
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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

Circuit

Use D3 of Freenove UNO to drive the servo.

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 12.1.1

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

```
#include <Servo.h>

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

int pos = 0;                // variable to store the servo position
int servoPin = 3;           // define the pin of servo signal line

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

```

```
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:

```
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 Arduino control board can define 12 objects of Servo type, namely, it 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:

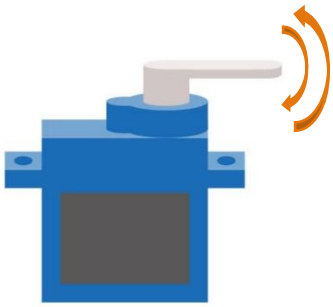
```
9    myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
```

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.

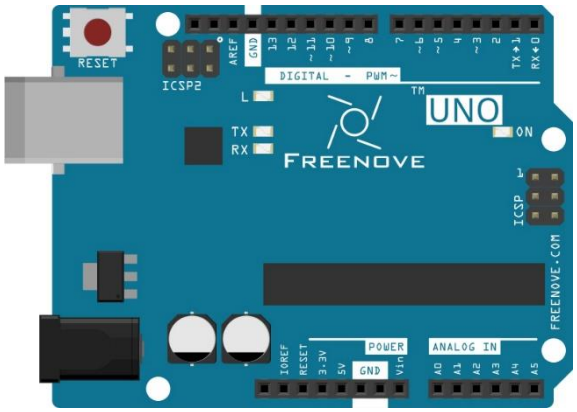
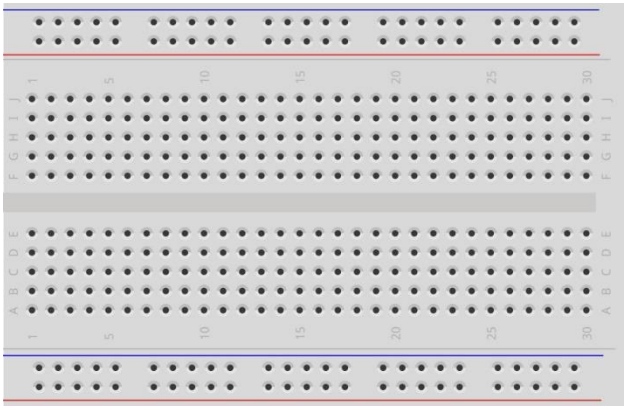
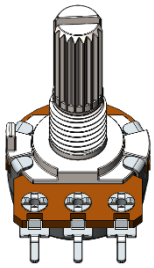
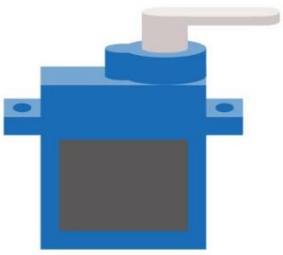
Verify and upload the code, the servo starts to sweep continuously.



Project 12.2 Control Servo by Potentiometer

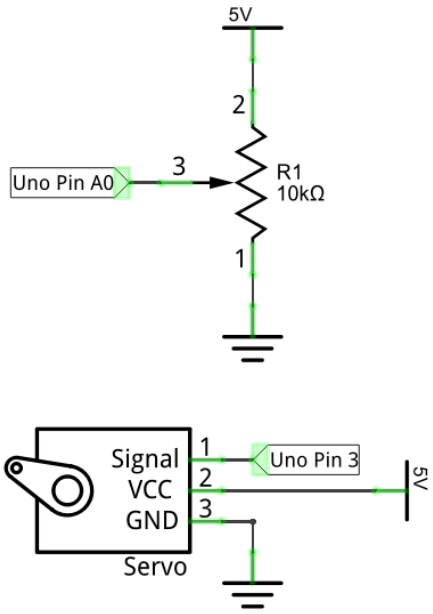
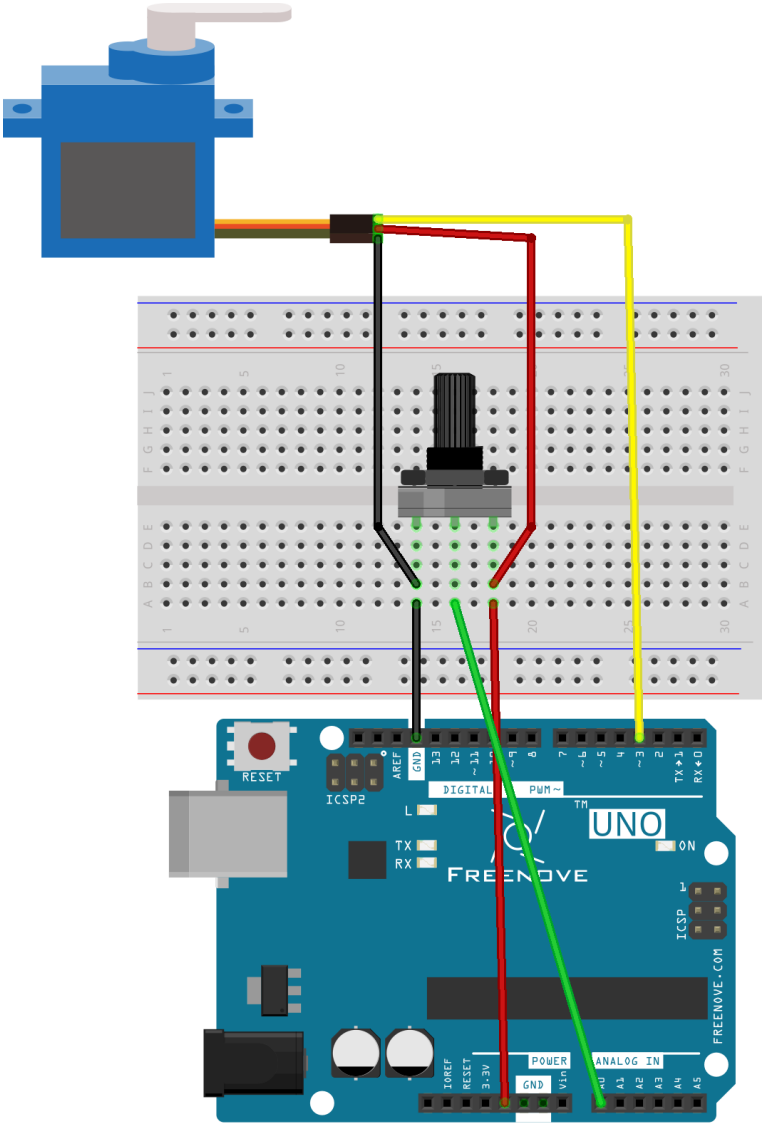
In the last section, we've made the servo sweep continuously. Now, we will use potentiometer to control the servo angle.

Component list

Freenove UNO x1 	Breadboard x1 	
USB cable x1	Rotary potentiometer x1 	Servo x1 
Jumper M/M x6		

Circuit

Use A0 of Freenove UNO to detect the voltage of rotary potentiometer, and D3 to drive the servo.

Schematic diagram	Hardware connection
	

Sketch

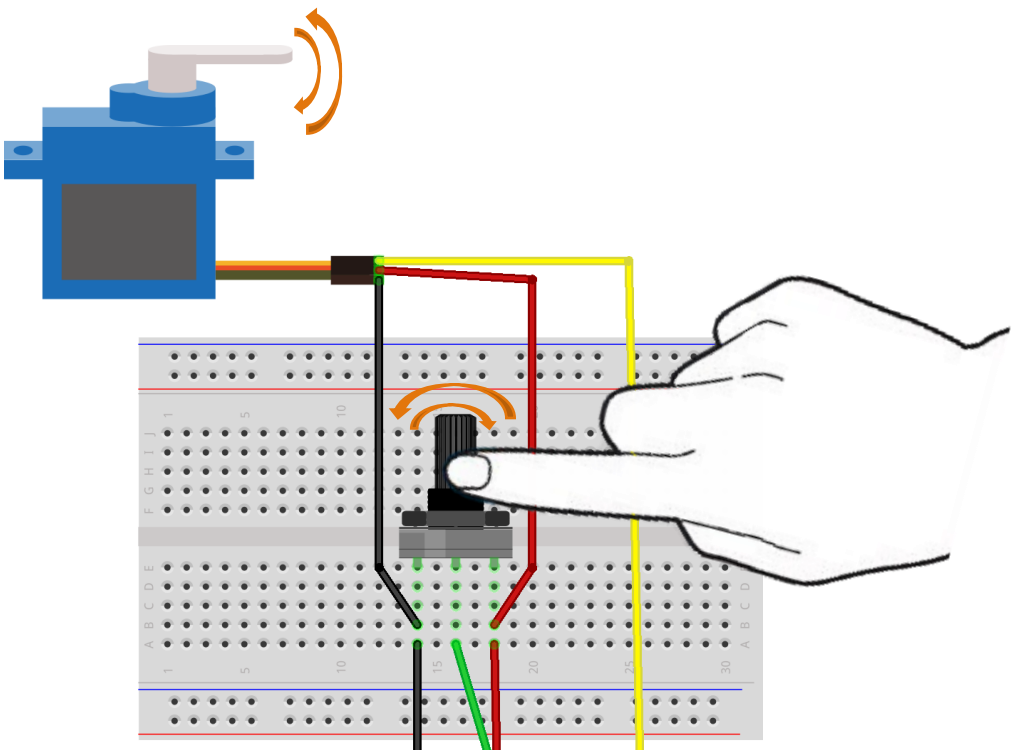
Sketch 12.2.1

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

```
#include <Servo.h>

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

int servoPin = 3;           // define the pin of servo signal line
int potPin = 0;             // analog pin used to connect the potentiometer
int potVal;                 // variable to read the potValue from the analog pin

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

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

In the code, we obtain the ADC value of A0 pin, and map it to the servo angle.

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



Chapter 13 LED Matrix

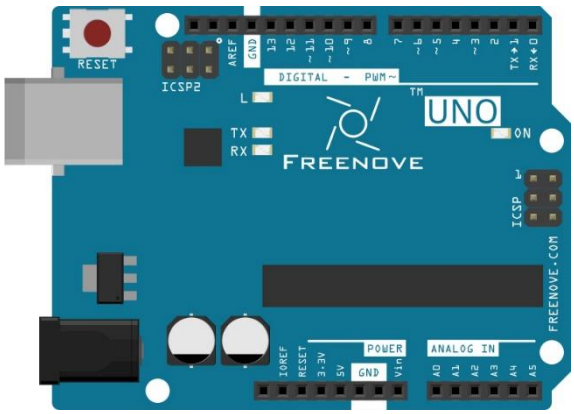
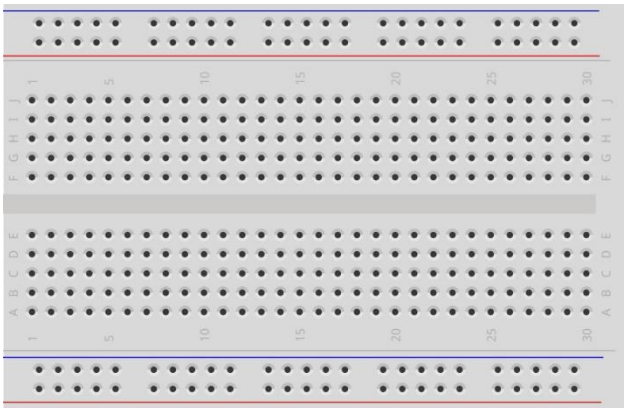
In the previous chapter, we have learned how to use some modules and sensors and shows some information on the computer through serial port. Now let us learn some modules which can output images and text.

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

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

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:

		Remainder	Sequence
16	18	····· 2	0
16	1	····· 1	1
	0		

The result is of the conversion 0x12.

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

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

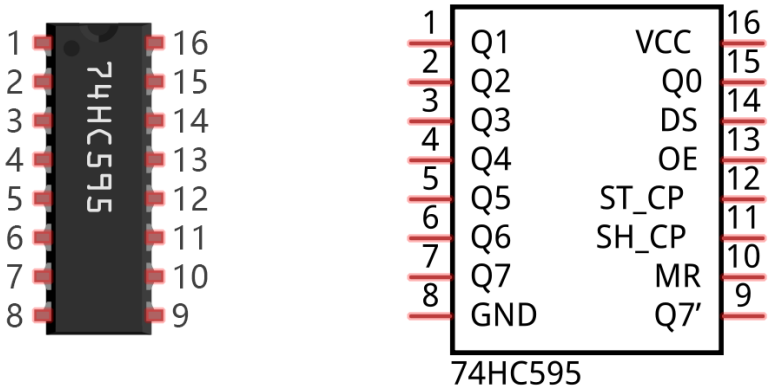
4 of bit binary	1000	1001	1010	1011	1100	1101	1110	1111
1 figure of hexadecimal	8	9	A	B	C	D	E	F

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

Component knowledge

74HC595

74HC595 chip is used to convert serial data into parallel data. 74HC595 can convert the serial data of one byte to 8 bits, and send its corresponding level to the corresponding 8 ports. With this feature, 74HC595 can be used to expand the IO port of Arduino board. At least 3 ports on the Arduino board are need to control 8 ports of 74HC595.



The ports of 74HC595 are described as follows:

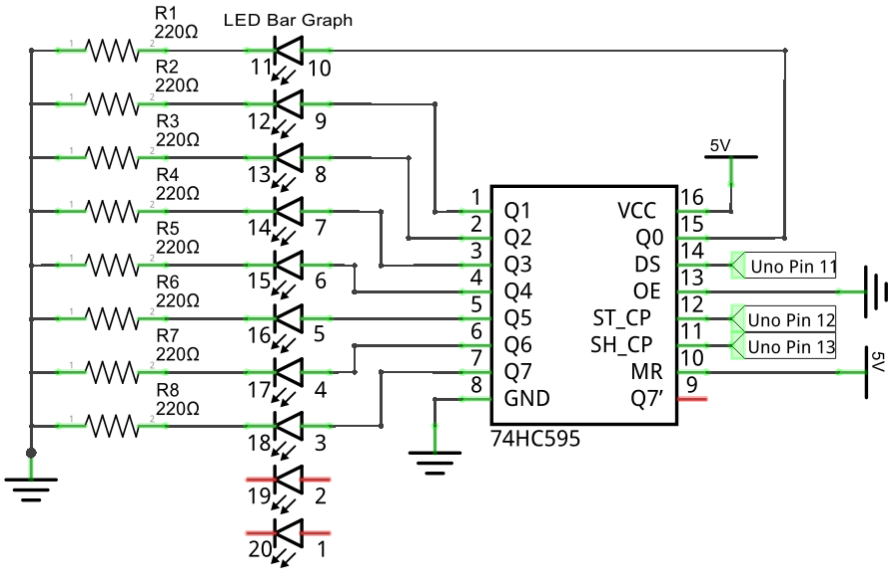
Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin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

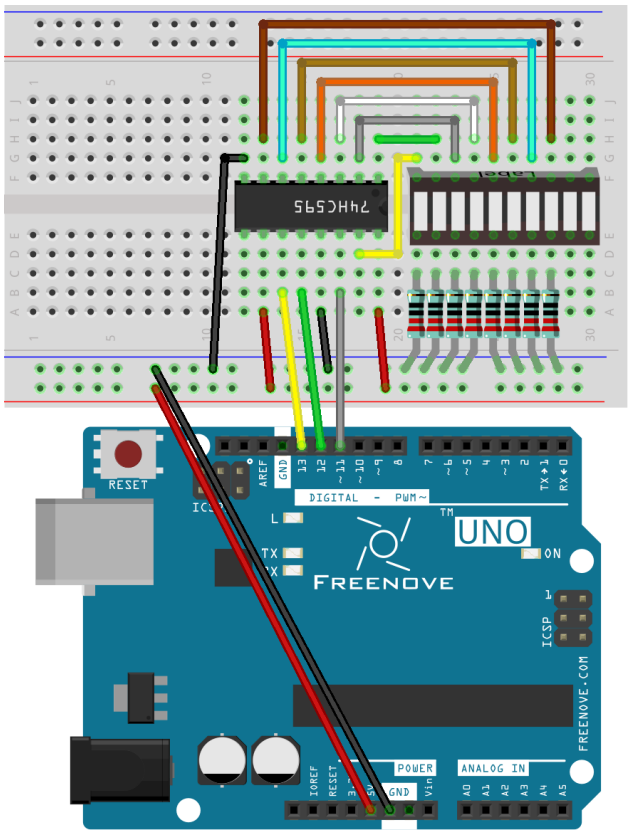
Circuit

Use D11, D12, D13 pin on Freenove UNO to control the 74HC595, and connect it to the 8 LEDs of LED bar graph.

Schematic diagram



Hardware connection



Sketch

Sketch 13.1.1

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

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)

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

void loop() {
  // Define a one-byte variable to use the 8 bits to represent the state of 8 LEDs of LED bar graph.
  // This variable is assigned to 0x01, that is binary 00000001, which indicates only one LED light on.
  byte x = 0x01;
  for (int j = 0; j < 8; j++) {
    // Output low level to latchPin
    digitalWrite(latchPin, LOW);
    // Send serial data to 74HC595
    shiftOut(dataPin, clockPin, LSBFIRST, x);
    // Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
    digitalWrite(latchPin, HIGH);
    // make the variable move one bit to left once, then the bright LED move one step to the left once.
    x <<= 1;
    delay(100);
  }
}
```

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.

```
  byte x = 0x01;
```

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

```
    // Output low level to latchPin
    digitalWrite(latchPin, LOW);
    // Send serial data to 74HC595
    shiftOut(dataPin, clockPin, LSBFIRST, x);
    // Output high level to latchPin, and 74HC595 will update the data to the parallel output port.
    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.

```
    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;

← ← ← ← ← ← ←

← | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ← | 0 |

The result of x is 2 (binary 00000010)。

0	0	0	0	0	0	1	0

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

byte x = 1 >> 1;

→ → → → → → →

0	→	0	0	0	0	0	0	0	1	→

The result of x is 0 (00000000)。

0	0	0	0	0	0	0	0

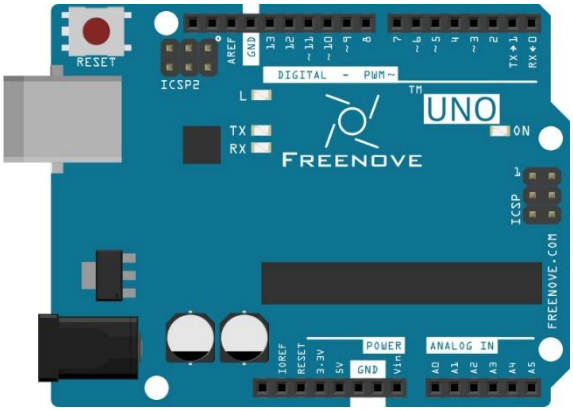
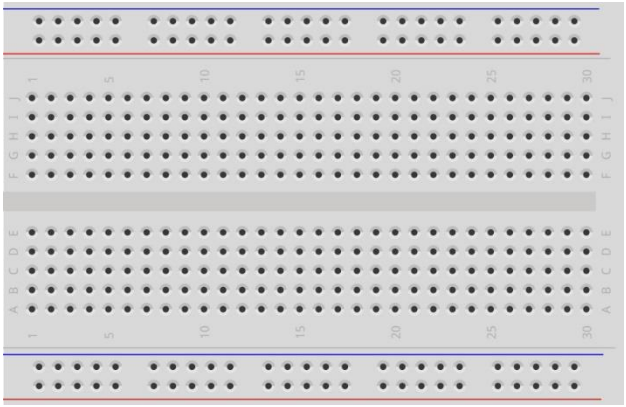
X <<= 1 is equivalent to x = x << 1 and x >>= 1 is equivalent to x = x >> 1

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

Project 13.2 LED Matrix

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

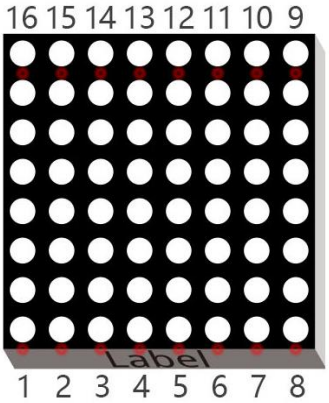
Component list

Freenove UNO x1 	Breadboard x1 		
USB cable x1	LED matrix x1	74HC595 x1	R220 x8
Jumper M/M x33			

Component knowledge

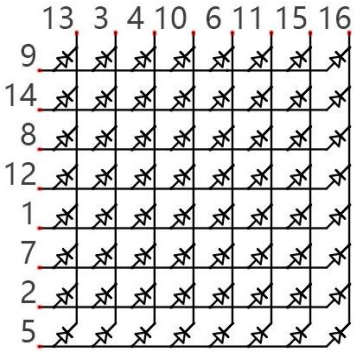
LED matrix

LED matrix is a rectangular display module that consists of several LEDs. The following is an 8*8 monochrome LED matrix with 64 LEDs (8 rows and 8 columns).

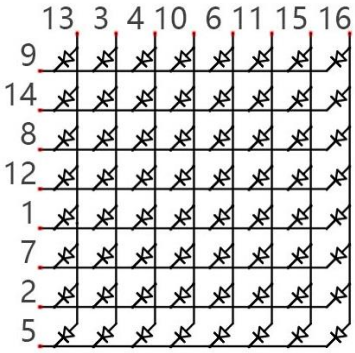


In order to facilitate the operation and save the ports, positive pole of LEDs in each row and negative pole of LEDs in each column are respectively connected together inside LED matrix module, which is called Common Anode. There is another form. Negative pole of LEDs in each row and positive pole of LEDs in each column are respectively connected together, which is called Common Cathode.

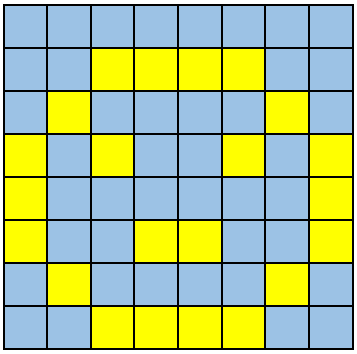
Connection mode of common anode



Connection mode of common cathode



Let us learn how connection mode of common anode works. Choose 16 ports on UNO board to connect to the 16 ports of LED Matrix. Configured one port in columns for low level, which make the column of the port selected. Then configure the eight ports in row to display content in the selected column. Delay for a moment. And then select the next column and outputs the corresponding content. This kind of operation to 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

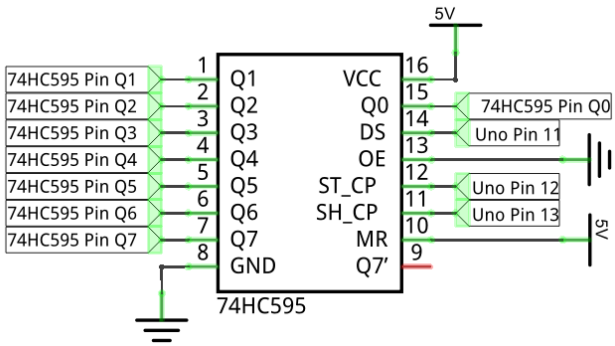
Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

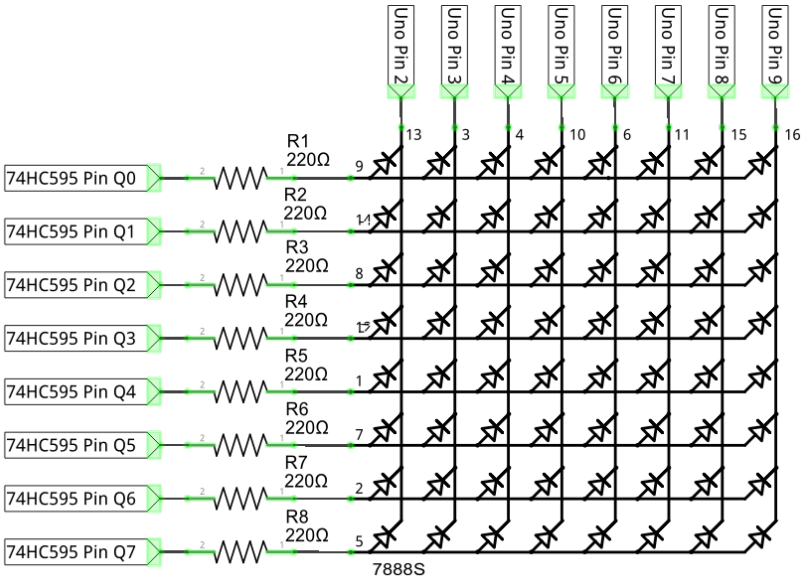
Scanning rows is another display way of dot matrix. It can save the controller ports, but it still needs 16 ports. So we use one 74HC595 to save some ports. You can use more 74HC595 to save ports where there are more ports needed.

Circuit

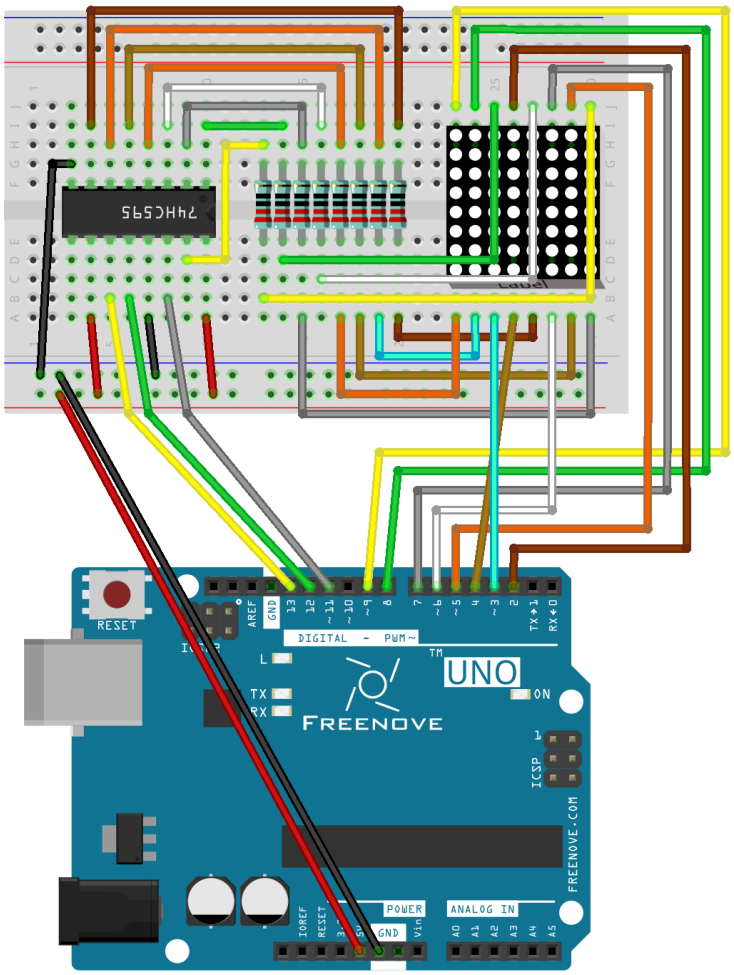
Use D11, D12, D13 pin on Freenove UNO board to control the 74HC595. And connect 74HC595 to the 8 anode pin of Matrix LED, in the meanwhile, connect 8 digitals port on UNO board are to the 8 cathode pins of LED Matrix.

Schematic diagram





Hardware connection



Sketch

Sketch 13.2.1

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.

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
int LEDPin[] = {2, 3, 4, 5, 6, 7, 8, 9};    // column pin (cathode) of LED Matrix

// Define the pattern data for a smiling face
const int smilingFace[] = {
  0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C
};
// Define the data of numbers and letters, and save them in flash area
const int data[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
  0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
  0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
  0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
  0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
  0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
  0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
  0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
  0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
  0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
  0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
  0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
  0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00  // "F"
};

void setup() {
  // set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (int i = 0; i < 8; i++) {
    pinMode(LEDPin[i], OUTPUT);
  }
```

```
}

void loop() {
  // Define a one-byte variable (8 bits) which is used to represent the selected state of 8 column.
  int cols;
  // Display the static smiling pattern
  for (int j = 0; j < 500; j++ ) {  // repeat 500 times
    cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the first column is selected.
    for (int i = 0; i < 8; i++) {   // display 8 column data by scanning
      matrixColsVal(cols);          // select this column
      matrixRowsVal(smilingFace[i]);// display the data in this column
      delay(1);                     // display them for a period of time
      matrixRowsVal(0x00);          // clear the data of this column
      cols <<= 1;                   // shift"cols" 1 bit left to select the next column
    }
  }
  // Display the dynamic patterns of numbers and letters
  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)
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixColsVal(cols);            // select this column
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
        delay(1);                       // display them for a period of time
        matrixRowsVal(0x00);            // close the data of this column
        cols <<= 1;                 // shift"cols" 1 bit left to select the next column
      }
    }
  }
}

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

```

```
void matrixColsVal(byte value) {
  byte cols = 0x01;
  // Output the column data to the corresponding port.
  for (int i = 0; i < 8; i++) {
    digitalWrite(LEDPin[i], ((value & cols) == cols) ? LOW : HIGH);
    cols <<= 1;
  }
}
```

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

```
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.

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

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.

```
const int data[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
  0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
  0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
  0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
  0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
  0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
  0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
  0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
  0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
  0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
  0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
  0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
  0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
  0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
  0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00  // "F"
};
```

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 UNO port to select the column.

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

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

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

? : operator

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

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

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

Bitwise logical operation

There are many bitwise logical operators such as and (&), or (|), xor (^), 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:

```
  1(00000001)
& 2(00000010)
  0(00000000)
```

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

byte a = ~15;

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

```
~  15(00001111)
  240(11110000)
```

In the loop () function, firstly, show the static smile pattern. Selected 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.

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

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.

```
  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)
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixColsVal(cols);            // select this column
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
        delay(1);                       // display them for a period of time
        matrixRowsVal(0x00);            // close the data of this column
        cols <<= 1;                 // shift "cols" 1 bit left to select the next column
      }
    }
  }
}
```

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.

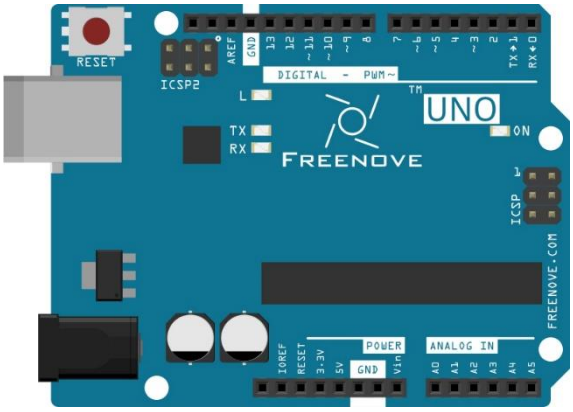
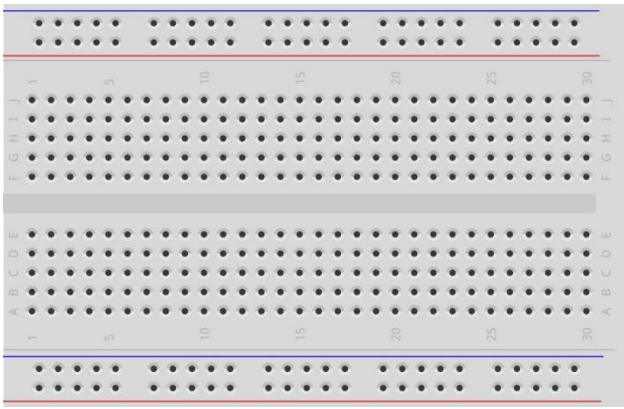
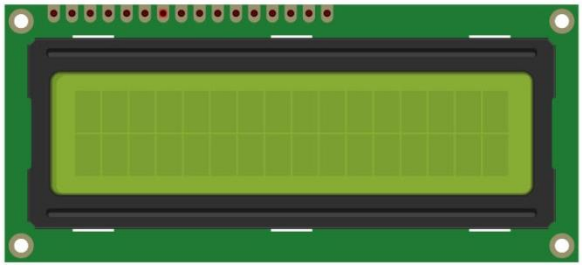
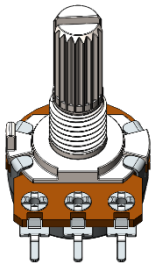
Chapter 14 LCD1602

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.

Project 14.1 Display the string on LCD1602

Firstly, use LCD1602 to display some strings.

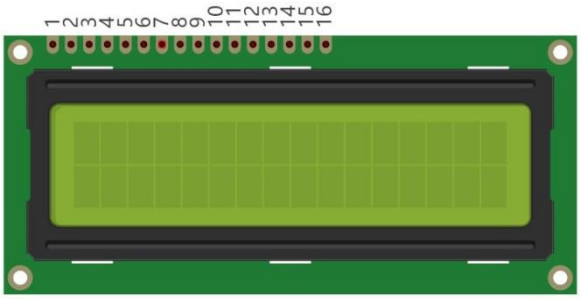
Component list

Freenove UNO x1 	Breadboard x1 
USB cable x1	Jumper M/M x16
LCD1602 x1 	Rotary potentiometer x1 

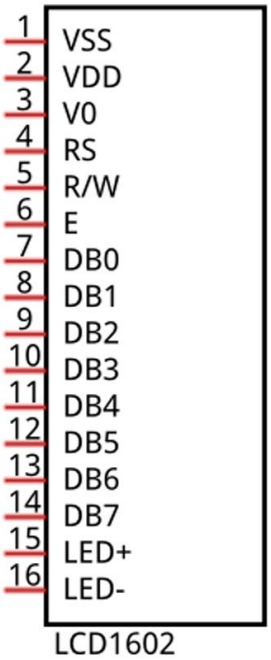
Component knowledge

LCD1602

LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. As shown in the following is a monochrome LCD1602 display screen:



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



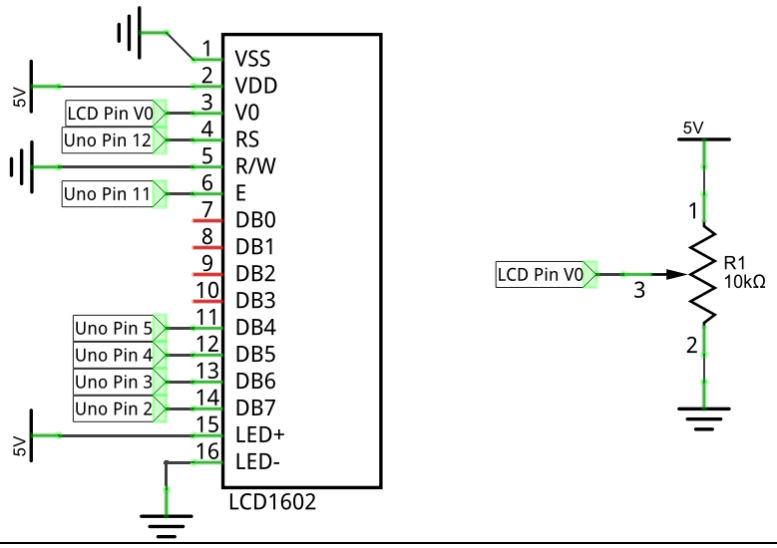
Pin name	Pin number	Description
VSS	1	Negative electrode of power supply
VDD	2	Positive electrode of power supply, the voltage is 5V
V0	3	Contrast and adjust the display effect
RS	4	Data/Command selection
RW	5	Read/Write selection
E	6	Enable pin
D0-D7	7-14	Data pin
LED+	15	Positive electrode of backlight LED, the voltage is 5V
LED-	16	Negative electrode of backlight LED

For more details, please refer to the datasheet.

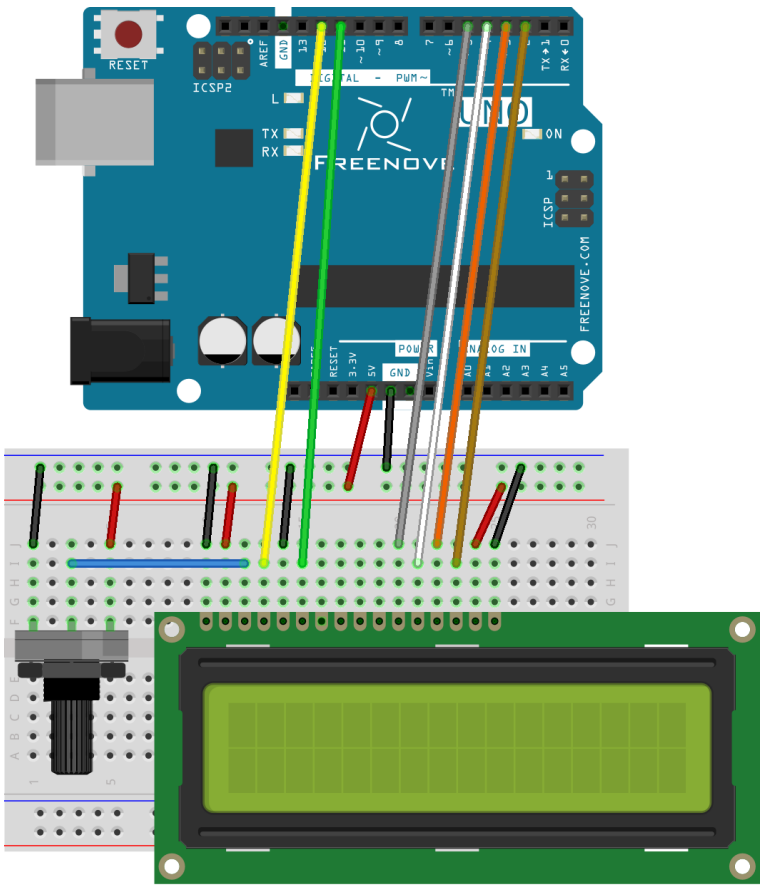
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 14.1.1

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

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD
  lcd.print("hello, world!");
}

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

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:

```
8    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:

```
lcd.setCursor(0, 1);
```

If we don't set the coordinate, the string will be printed behind the last 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);

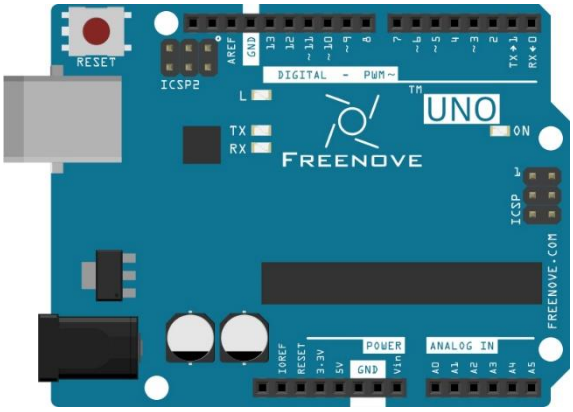
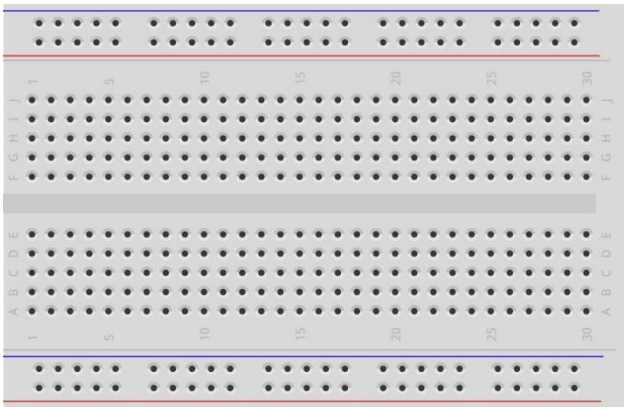
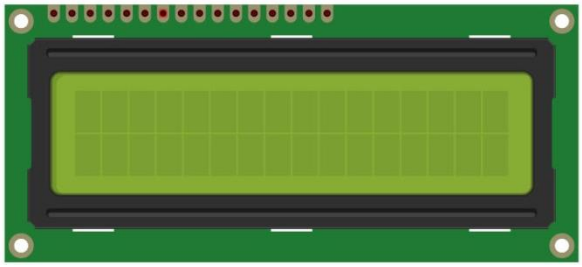
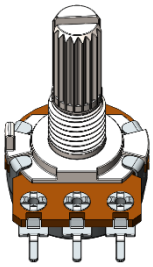
lcd.print(data): print the number and letters in the coordinate we set before. If we haven't set the coordinate, the string will be printed behind the last printed character.

Verify and upload the code, then you will see the LCD1602 display the string and changing number.
If the display of character is not normal, shift the rotary potentiometer to adjust the contrast.

Project 14.2 LCD1602 Clock

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

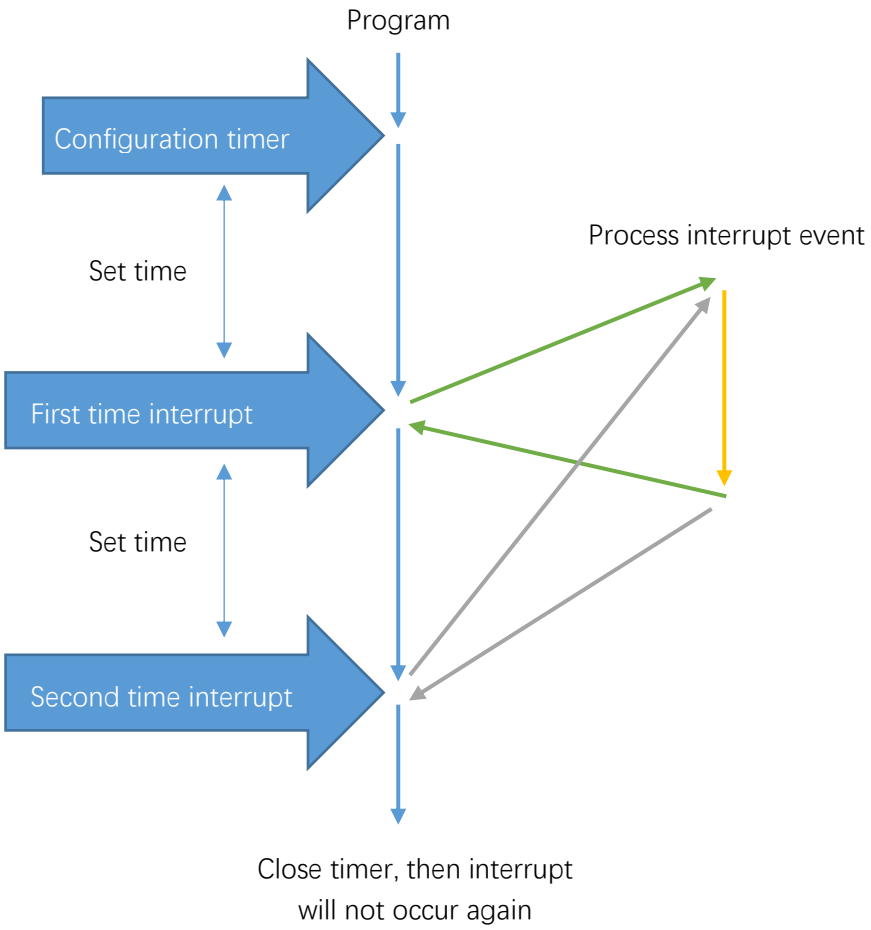
Component list

Freenove UNO x1 	Breadboard x1 		
USB cable x1	Jumper M/M x18		
LCD1602 x1 	Rotary potentiometer x1 	Thermistor x1	Resistor 10kΩ x1

Code knowledge

Timer

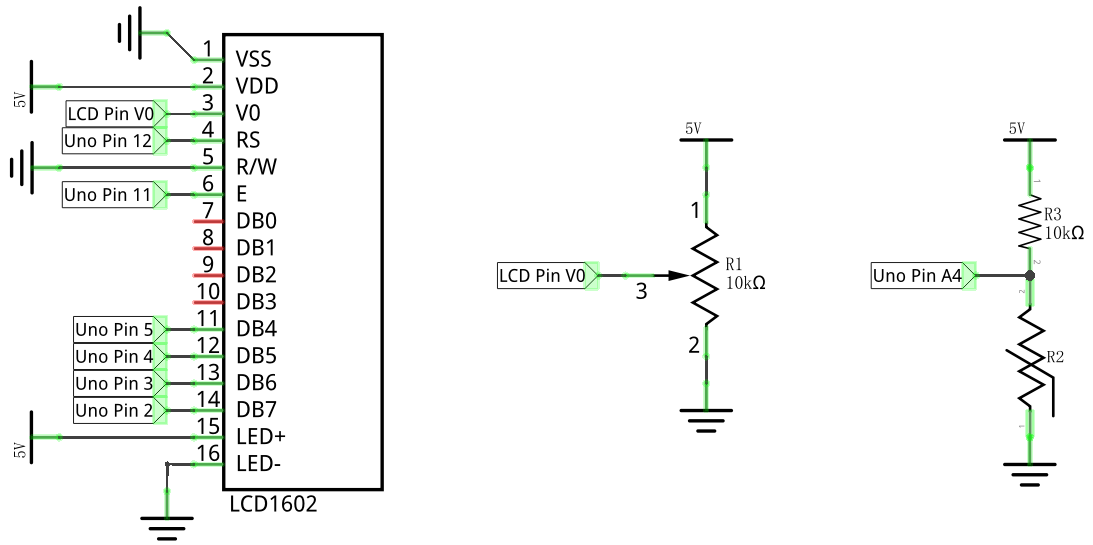
Timer can be set to produce an interrupt after a period of time. When the timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted place to go on. If you don't close the timer,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



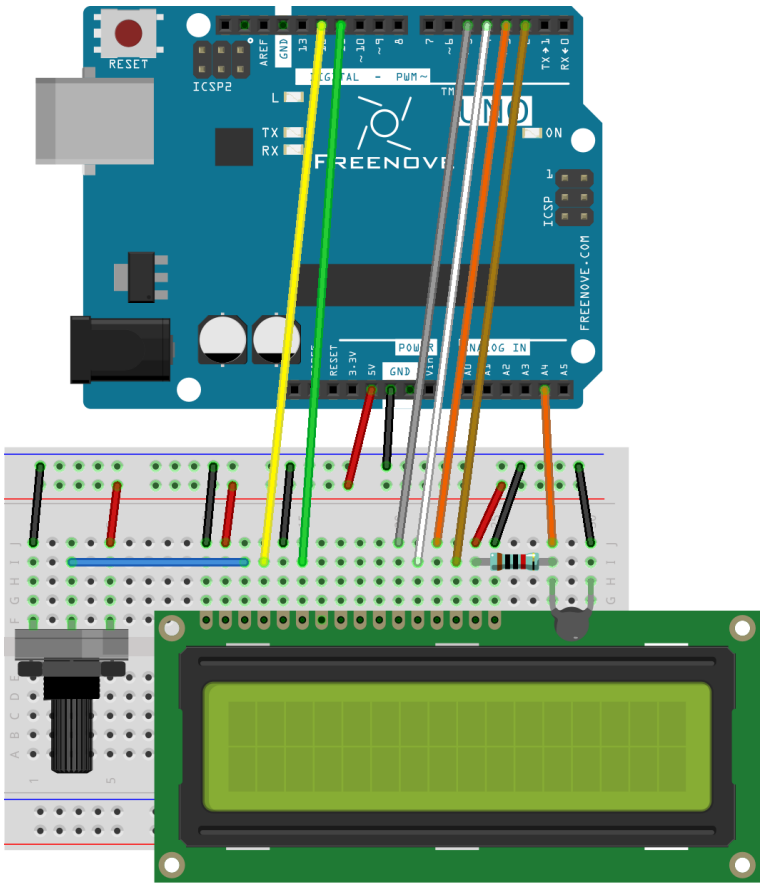
Circuit

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

Schematic diagram



Hardware connection

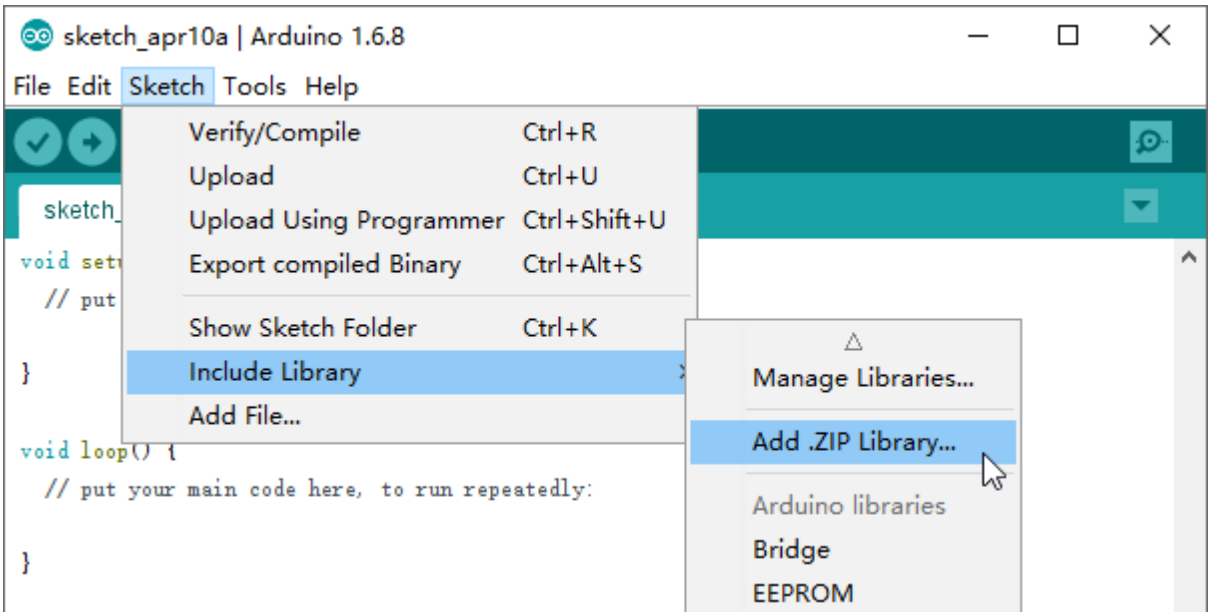


Sketch

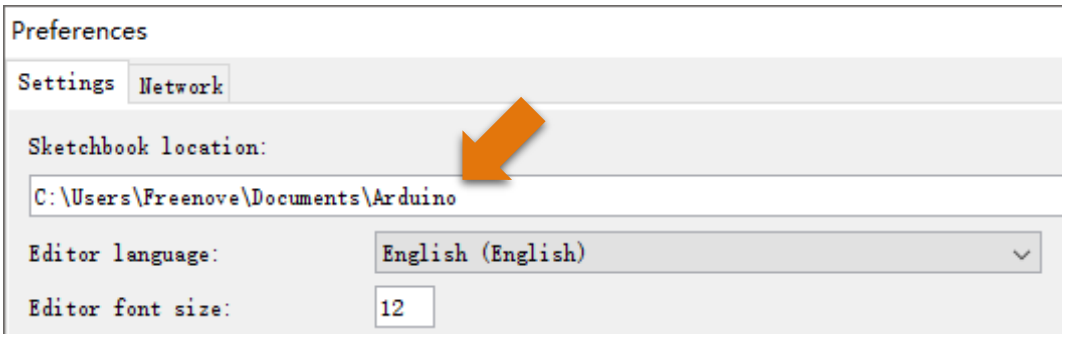
Sketch 14.2.1

Before writing the code, we need to add the library file FlexiTimer2.zip first. The FlexiTimer2 library can help manipulate the timer.

Library is a collection of code. It will make it very simple to implement complex functions.
Some libraries are built in the Arduino software. To add a new library, click "Add.ZIP Library..." and then choose the ZIP library file. All the ZIP library files we need are in Libraries folder.



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.



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

```
#include <LiquidCrystal.h>    // Contains LiquidCrystal Library
#include <FlexiTimer2.h>      // Contains FlexiTimer2 Library

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int tempPin = 4;                // define the pin of temperature sensor
float tempVal;                  // define a variable to store temperature value
int hour, minute, second;       // define variables stored record time

void setup() {
  lcd.begin(16, 2);     // set up the LCD's number of columns and rows
  startingAnimation();  // display a dynamic start screen
  FlexiTimer2::set(1000, timerInt);  // configure the timer and interrupt function
  FlexiTimer2::start();              // start timer
  Serial.begin(9600);                // initialize serial port with baud rate 9600
  Serial.println("UNO is ready!");   // print the string "UNO is ready!"
  Serial.println("Input hour,minute,second to set time.");
}

void loop() {
  // Get temperature
  tempVal = getTemp();
  if (second >= 60) {       // when seconds is equal to 60, minutes plus 1
    second = 0;
    minute++;
    if (minute >= 60) {     // when minutes is equal to 60, hours plus 1
      minute = 0;
      hour++;
      if (hour >= 24) {     // when hours is equal to 24, hours turn to zero
        hour = 0;
      }
    }
  }
  lcdDisplay();             // display temperature and time information on LCD
  delay(200);
}

void startingAnimation() {
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayRight();
  }
```

```
  lcd.print("starting...");
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayLeft();
    delay(300);
  }
  lcd.clear();
}

// the timer interrupt function of FlexiTimer2 is executed every 1s
void timerInt() {
  second++;       // second plus 1
}

// serial port interrupt function
void serialEvent() {
  int inInt[3];  // define an array to save the received serial data
  while (Serial.available()) {
    for (int i = 0; i < 3; i++) {
      inInt[i] = Serial.parseInt();  // receive 3 integer data
    }
    // print the received data for confirmation
    Serial.print("Your input is: ");
    Serial.print(inInt[0]);
    Serial.print(", ");
    Serial.print(inInt[1]);
    Serial.print(", ");
    Serial.println(inInt[2]);
    // use received data to adjust time
    hour = inInt[0];
    minute = inInt[1];
    second = inInt[2];
    // print the modified time
    Serial.print("Time now is: ");
    Serial.print(hour / 10);
    Serial.print(hour % 10);
    Serial.print(':');
    Serial.print(minute / 10);
    Serial.print(minute % 10);
    Serial.print(':');
    Serial.print(second / 10);
    Serial.println(second % 10);
  }
}

```

```
// function used by LCD1602 to display time and temperature
void lcdDisplay() {
  lcd.setCursor(0, 0);  // set the cursor to (0,0) (first column,first row).
  lcd.print("TEMP: ");  // display temperature information
  lcd.print(tempVal);
  lcd.print("C");
  lcd.setCursor(0, 1);  // set the cursor to (0,1) (first column,second row)
  lcd.print("TIME: ");  // display time information
  lcd.print(hour / 10);
  lcd.print(hour % 10);
  lcd.print(':');
  lcd.print(minute / 10);
  lcd.print(minute % 10);
  lcd.print(':');
  lcd.print(second / 10);
  lcd.print(second % 10);
}
```

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

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

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 parameter to FlexiTimer2::set () function.

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

After every interrupt, the second plus 1.

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

:: 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.

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

In the loop function, we need to control the second, minute, hour. When the second increases to 60, the minute adds 1, and reset the second to zero; when the minute increases to 60, the hour adds 1, and reset the minute to zero; when the hour increases to 24, reset it to zero.

```
  if (second >= 60) {       // when the second increases to 60, the minute adds 1
    second = 0;
    minute++;
    if (minute >= 60) {     //when the minute increases to 60, the hour adds 1
      minute = 0;
      hour++;
      if (hour >= 24) {     // when the hour increases to 24, turn it to zero
        hour = 0;
      }
    }
  }
```

We define a function lcdDisplay () to refresh the information on LCD display. In this function, use two bit to display the hour, minute, second on the LCD. Such as hour/ 10 is the units, hour% 10 is the tens.

```
void lcdDisplay() {
  lcd.setCursor(0, 0);  // set the cursor to (0,0) (first column,first row).
  lcd.print("TEMP: ");  // display temperature information
  lcd.print(tempVal);
  lcd.print("C");
  lcd.setCursor(0, 1);  // set the cursor to (0,1) (first column,second row)
  lcd.print("TIME: ");  // display time information
  lcd.print(hour / 10);
  lcd.print(hour % 10);
  lcd.print(':');
  lcd.print(minute / 10);
  lcd.print(minute % 10);
  lcd.print(':');
  lcd.print(second / 10);
  lcd.print(second % 10);
}
```

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

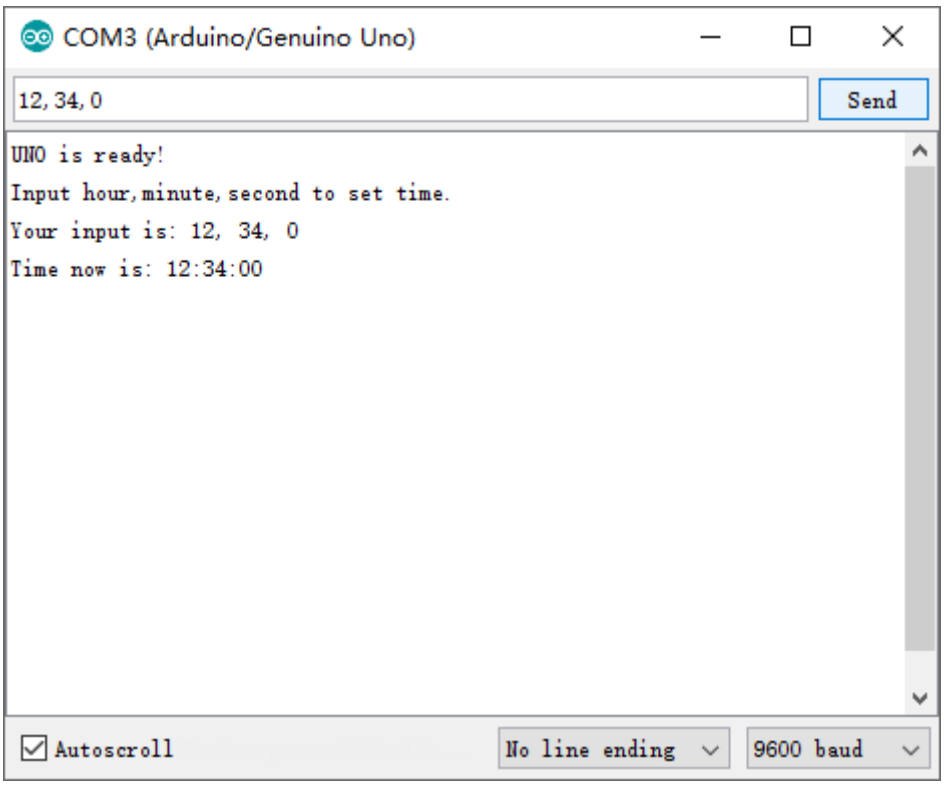
```
void serialEvent() {
  int inInt[3];  // define an array to save the received serial data
  while (Serial.available()) {
    for (int i = 0; i < 3; i++) {
      inInt[i] = Serial.parseInt();  // receive 3 integer data
    }
    // print the received data for confirmation
    Serial.print("Your input is: ");
    Serial.print(inInt[0]);
    Serial.print(", ");
```

```
    Serial.print(inInt[1]);
    Serial.print(", ");
    Serial.println(inInt[2]);
    // use the received data to adjust time
    hour = inInt[0];
    minute = inInt[1];
    second = inInt[2];
    // print the modified time
    Serial.print("Time now is: ");
    Serial.print(hour / 10);
    Serial.print(hour % 10);
    Serial.print(':');
    Serial.print(minute / 10);
    Serial.print(minute % 10);
    Serial.print(':');
    Serial.print(second / 10);
    Serial.println(second % 10);
  }
}
```

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

```
void startingAnimation() {
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayRight();
  }
  lcd.print("starting...");
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayLeft();
    delay(300);
  }
  lcd.clear();
}
```

Verify and upload the code. The LCD screen will display a scrolling string first, and then displays the temperature and time. We can open Monitor Serial and enter time in the sending area, then click the Send button to set the time.



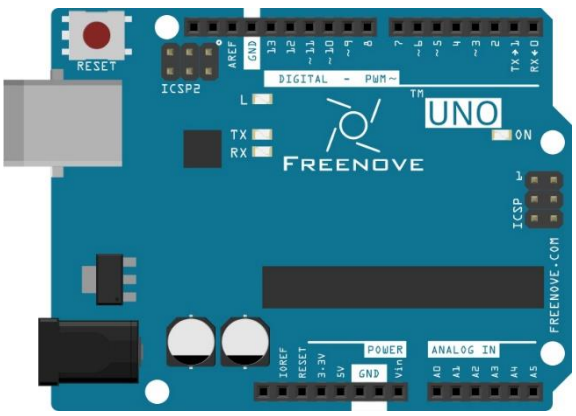
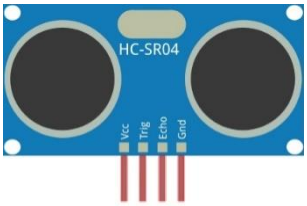
Chapter 15 Ultrasonic Ranging

In this chapter, we learn a module which 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.

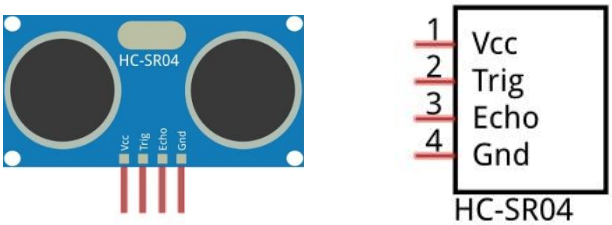
Component List

Freenove UNO x1 	USB cable x1
	Jumper F/M x4
	Ultrasonic ranging module x1 

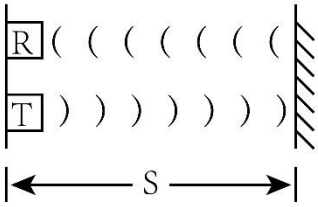
Component Knowledge

Ultrasonic ranging module

Ultrasonic ranging module use the principle that ultrasonic will reflect when it encounters obstacles. Ultrasonic module integrates a transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into sound waves (mechanical energy) and the function of the receiver is opposite.



Start counting the time when ultrasonic is transmitted. And when ultrasonic encounters an obstacle, it will reflect back. The counting will end after ultrasonic is received, and the time difference is the total time of ultrasonic from transmitting to receiving. Because the speed of sound in air is constant, and is about v=340m/s. So we can calculate the distance between the module and the obstacle: S=vt/2.



Pin description:

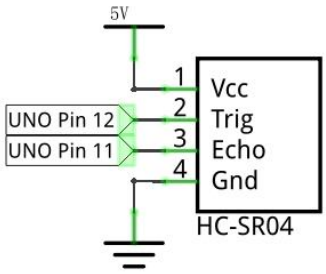
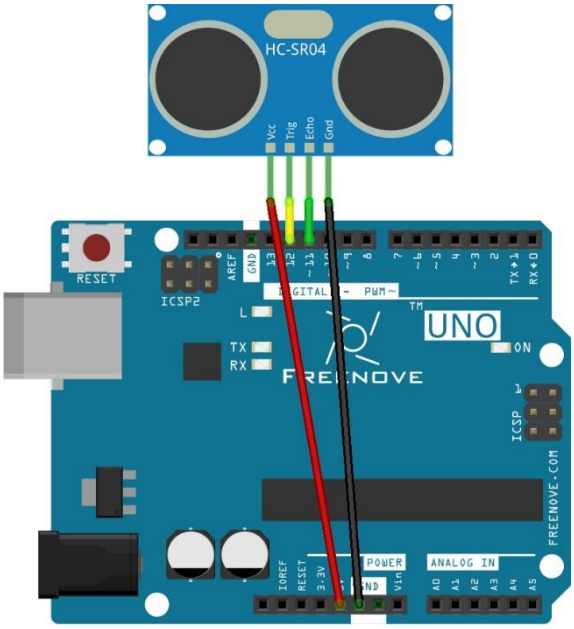
Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Triger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

Instructions for use:

Output a high-level pulse in Trig pin lasting for least 10us. Then the module begins to transmit ultrasonic. At the same time, the Echo pin will be pulled up. When the module receives the returned ultrasonic, the Echo pin will be pulled down. The duration of high level in Echo pin is the total time of the ultrasonic from transmitting to receiving, s=vt/2, which is used to operate the HC-SR04 in Arduino.

Circuit

The connection of Freenove UNO board and HC-SR04 is shown below.

Schematic diagram 
Hardware connection 

Sketch

Sketch 15.1.1

First, we use the HC-SR04 communication protocol to operate the module, get the range of time, and calculate the distance.

```
#define trigPin 12 // define TrigPin
#define echoPin 11 // define EchoPin.
#define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
// define the timeOut according to the maximum range. timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DISTANCE * 60;
int soundVelocity = 340; // define sound speed=340m/s

void setup() {
  pinMode(trigPin,OUTPUT);// set trigPin to output mode
  pinMode(echoPin,INPUT); // set echoPin to input mode
  Serial.begin(9600);     // Open serial monitor at 9600 baud to see ping results.
}

void loop() {
  delay(100); // Wait 100ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.
  Serial.print("Ping: ");
  Serial.print(getSonar()); // Send ping, get distance in cm and print result (0 = outside set distance range)
  Serial.println("cm");
}

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

First, define the pins and the maximum measurement distance.

```
#define trigPin 12 // define TrigPin
#define echoPin 11 // define EchoPin.
#define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
```

If the module does not return high level, we can not wait forever. So we need to calculate the lasting time over maximum distance, that is, time Out. timOut= 2*MAX_DISTANCE/100/340*1000000. The constant part behind is approximately equal to 58.8.

```
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 HC_SR04. Make trigPin output a high level lasting for at least 10μS, according to the communication protocol.

```
  digitalWrite(trigPin, HIGH); // make trigPin output high level lasting for 10μs to triger HC_SR04,
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

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.

```
  pingTime = pulseIn(echoPin, HIGH, timeOut); // Wait HC-SR04 returning to the high level and measure out this waitting time
```

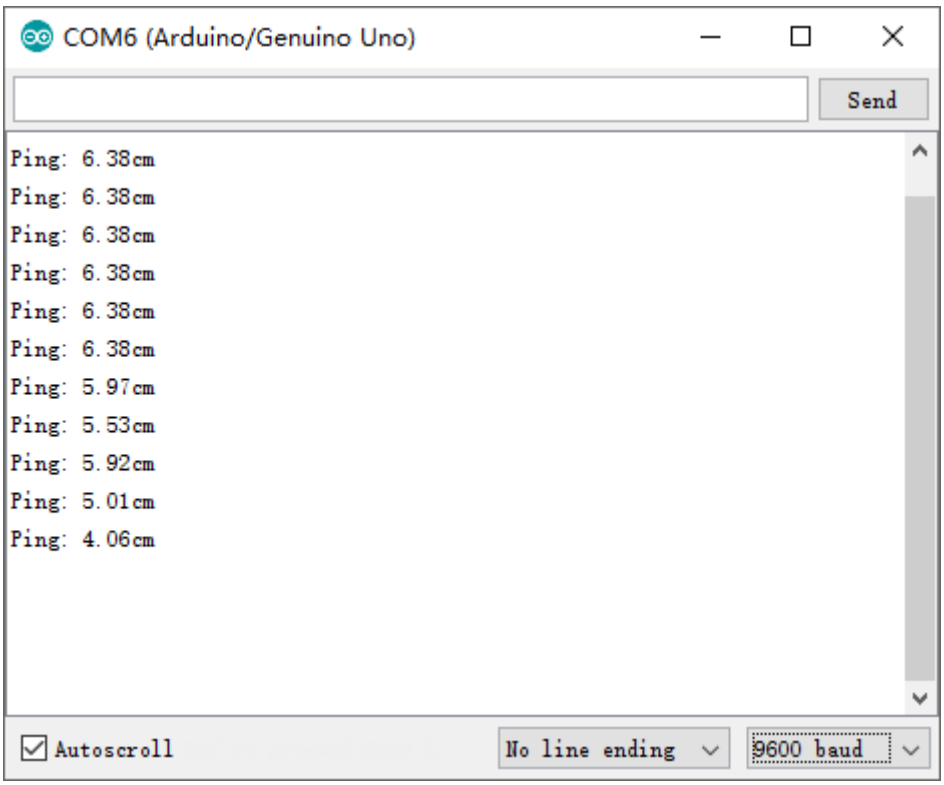
Calculate the distance according to the time and return the value.

```
  distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance according to the time
  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 UNO. 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 15.1.2

We can also use the library function to directly obtain the measuring distance. We need import the file NewPing.zip before using the library.

```
#include <NewPing.h>
#define trigPin 12 // define TrigPin
#define echoPin 11 // define EchoPin.
#define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.

NewPing sonar(trigPin, echoPin, MAX_DISTANCE); // NewPing setup of pins and maximum distance.

void setup() {
  Serial.begin(9600); // Open serial monitor at 9600 bauds to see ping results.
}

void loop() {
  delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.
  Serial.print("Ping: ");
  Serial.print(sonar.ping_cm()); // Send ping, get distance in cm and print result (0 = outside set distance range)
  Serial.println("cm");
}
```

First, include the header file of library, and then define the HC SR04 pin and the maximum measurement distance. And, write these parameters when we define the NewPing class objects.

```
#include <NewPing.h>
#define trigPin 12 // define TrigPin
#define echoPin 11 // define EchoPin.
#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:

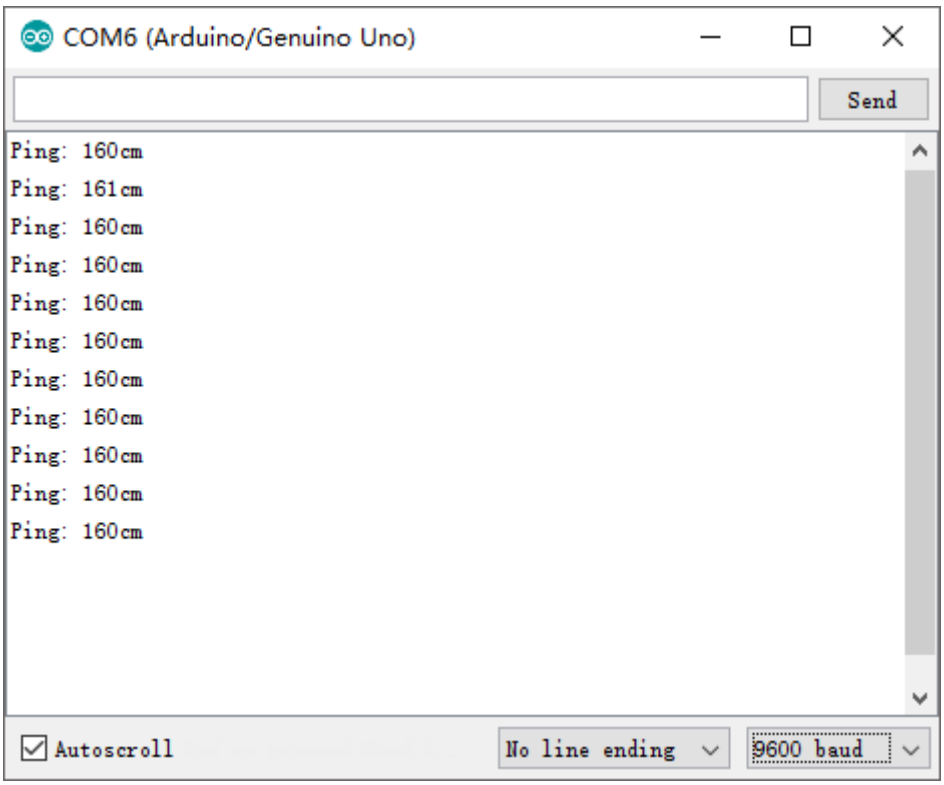
sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

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 uploaded the code to UNO. Open the serial port monitoring window. When you make ultrasonic probe toward a plane of an object, you can observe the distance changes.



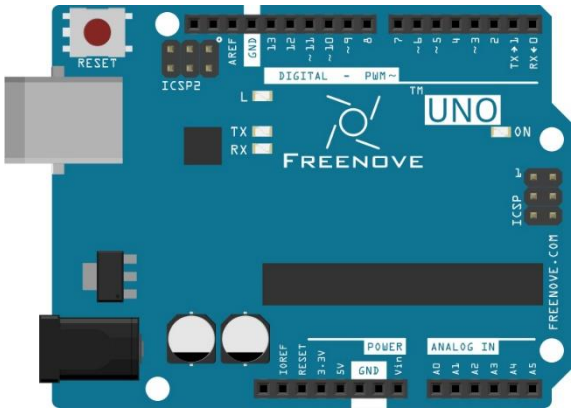
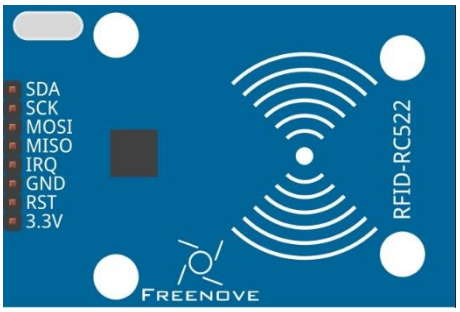
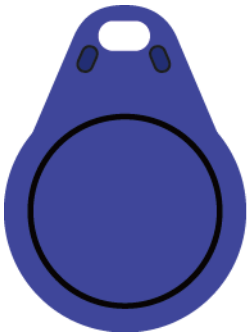
Chapter 16 RFID

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

Project 16.1 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

Freenove UNO 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.

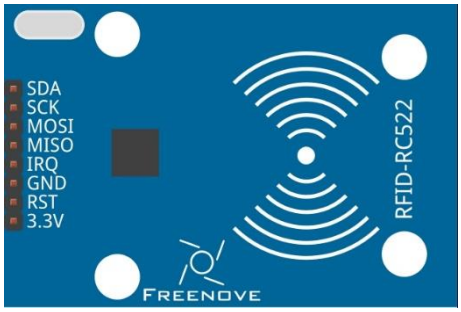
Component Knowledge

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.

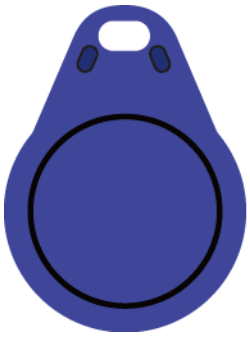
This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.



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*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.

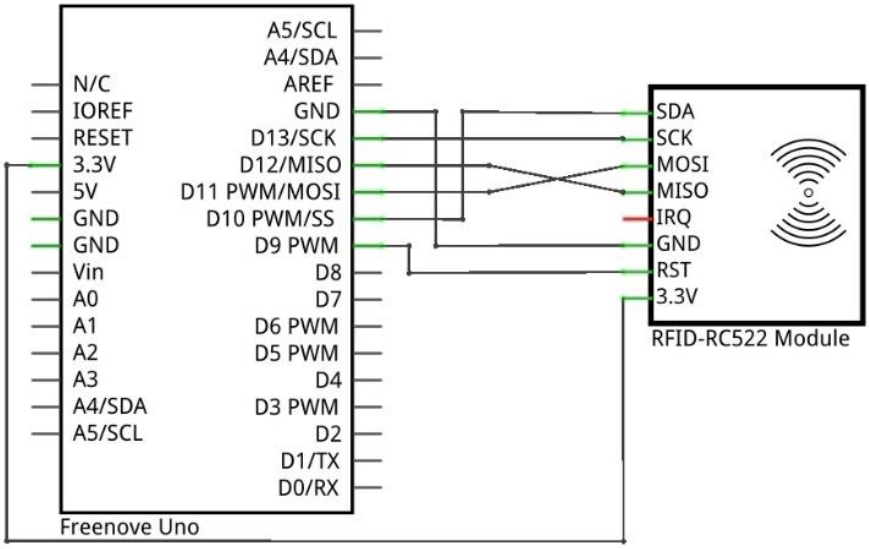
For more details about how to set data blocks and control blocks, please refer to Datasheet.

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.**

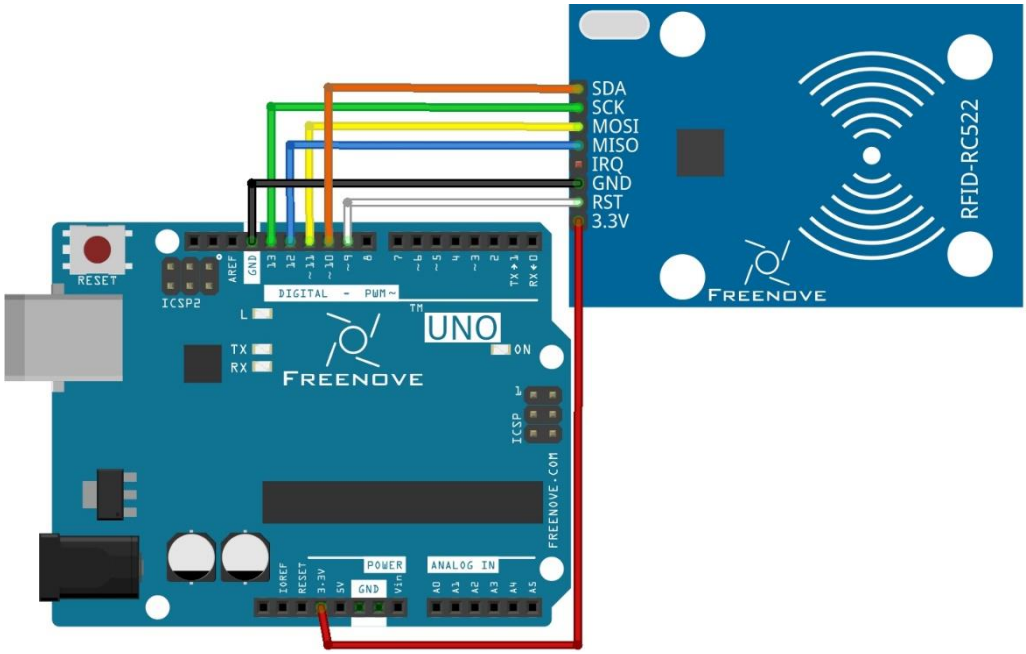
Circuit

The connection of Freenove UNO board and RFID module is shown below.

Schematic diagram



Hardware connection



Sketch

Sketch 16.1.1

Before writing the code, we need import the library RFID.zip first.

This sketch will read the unique ID number (UID) of the card, recognize the type of the card and display the information through serial port.

```
#include <SPI.h>
#include <RFID.h>

//D10:pin of card reader SDA.  D9:pin of card reader RST
RFID rfid(10, 9);
unsigned char status;
unsigned char str[MAX_LEN];  //MAX_LEN is 16: size of the array

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  rfid.init(); //initialization
  Serial.println("Please put the card to the induction area...");
}

void loop()
{
  //Search card, return card types
  if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
    Serial.println("Find the card!");
    // Show card type
    ShowCardType(str);
    //Anti-collision detection, read card serial number
    if (rfid.anticoll(str) == MI_OK) {
      Serial.print("The card's number is  : ");
      //Display card serial number
      for (int i = 0; i < 4; i++) {
        Serial.print(0x0F & (str[i] >> 4), HEX);
        Serial.print(0x0F & str[i], HEX);
      }
      Serial.println("");
    }
    //card selection (lock card to prevent redundant read, removing the line will make the sketch read cards continually)
    rfid.selectTag(str);
  }
```

```
  rfid.halt();  // command the card to enter sleeping state
}
void ShowCardType(unsigned char * type)
{
  Serial.print("Card type: ");
  if (type[0] == 0x04 && type[1] == 0x00)
    Serial.println("MFOne-S50");
  else if (type[0] == 0x02 && type[1] == 0x00)
    Serial.println("MFOne-S70");
  else if (type[0] == 0x44 && type[1] == 0x00)
    Serial.println("MF-UltraLight");
  else if (type[0] == 0x08 && type[1] == 0x00)
    Serial.println("MF-Pro");
  else if (type[0] == 0x44 && type[1] == 0x03)
    Serial.println("MF Desire");
  else
    Serial.println("Unknown");
}
```

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.

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

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

```
  Serial.begin(9600);
  SPI.begin();
  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.

```
  if (rfid.findCard(PICC_REQIDL, str) == MI_OK) {
```

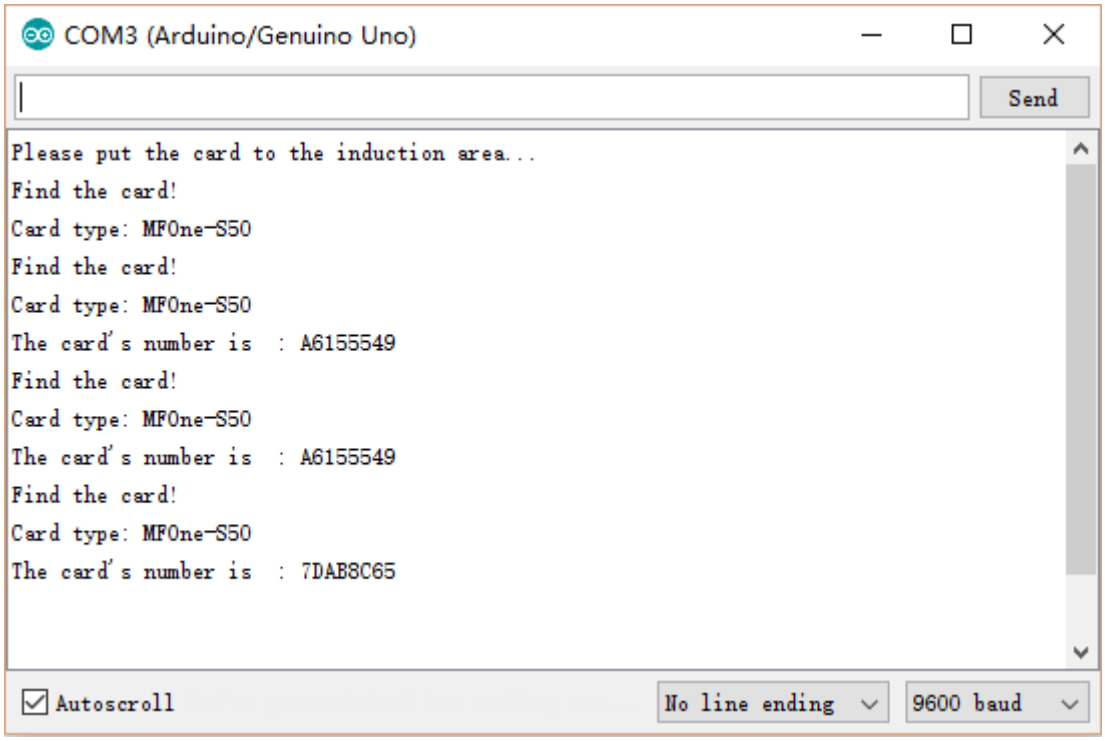
After entering if statement, call the sub function ShowCardType(). Then determine the type of the card according to the content of STR and print the type out through the serial port.

```
    ShowCardType(str);
```

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

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

After verifying and uploading the code to UNO, 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 16.2.1

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.

```
#include <SPI.h>
#include <RFID.h>

// D10:pin of card reader SDA.  D9: pin of card reader RST
RFID rfid(10, 9);

// 4-byte card serial number, the fifth byte is check byte
unsigned char serNum[5];
unsigned char status;
unsigned char str[MAX_LEN];
unsigned char blockAddr;        //Select the operation block address: 0 to 63

// Write card data you want(within 16 bytes)
unsigned char writeDate[16] = "WelcomeFreenove";

// The A password of original sector: 16 sectors; the length of password in each sector
is 6 bytes.
unsigned char sectorKeyA[16][16] = {
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
```

```
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
  {  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  } ,
};

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  rfid.init();
  Serial.println("Please put the card to the induction area...");
}

void loop()
{
  //find the card
  rfid.findCard(PICC_REQIDL, str);
  //Anti-collision detection, read serial number of card
  if (rfid.anticoll(str) == MI_OK) {
    Serial.print("The card's number is  : ");
    //print the card serial number
    for (int i = 0; i < 4; i++) {
      Serial.print(0x0F & (str[i] >> 4), HEX);
      Serial.print(0x0F & str[i], HEX);
    }
    Serial.println("");
    memcpy(rfid.serNum, str, 5);
  }
  //select card and return card capacity (lock the card to prevent multiple read and
written)
  rfid.selectTag(rfid.serNum);
  //first, read the data of data block 4
  readCard(4);
  //write data(within 16 bytes) to data block
  writeCard(4);
  //then read the data of data block again
  readCard(4);
```

```

  rfid.halt();
}

//write the card
void writeCard(int blockAddr) {
  if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) == MI_OK) //authenticate
  {
    //write data
    //status = rfid.write((blockAddr/4 + 3*(blockAddr/4+1)), sectorKeyA[0]);
    Serial.print("set the new card password, and can modify the data of the Sector: ");
    Serial.println(blockAddr / 4, DEC);
    // select block of the sector to write data
    if (rfid.write(blockAddr, writeDate) == MI_OK) {
      Serial.println("Write card OK!");
    }
  }
}

//read the card
void readCard(int blockAddr) {
  if ( rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) == MI_OK) // authenticate
  {
    // select a block of the sector to read its data
    Serial.print("Read from the blockAddr of the card : ");
    Serial.println(blockAddr, DEC);
    if ( rfid.read(blockAddr, str) == MI_OK) {
      Serial.print("The data is (char type display): ");
      Serial.println((char *)str);
      Serial.print("The data is (HEX type display): ");
      for (int i = 0; i < sizeof(str); i++) {
        Serial.print(str[i], HEX);
        Serial.print(" ");
      }
      Serial.println();
    }
  }
}
```

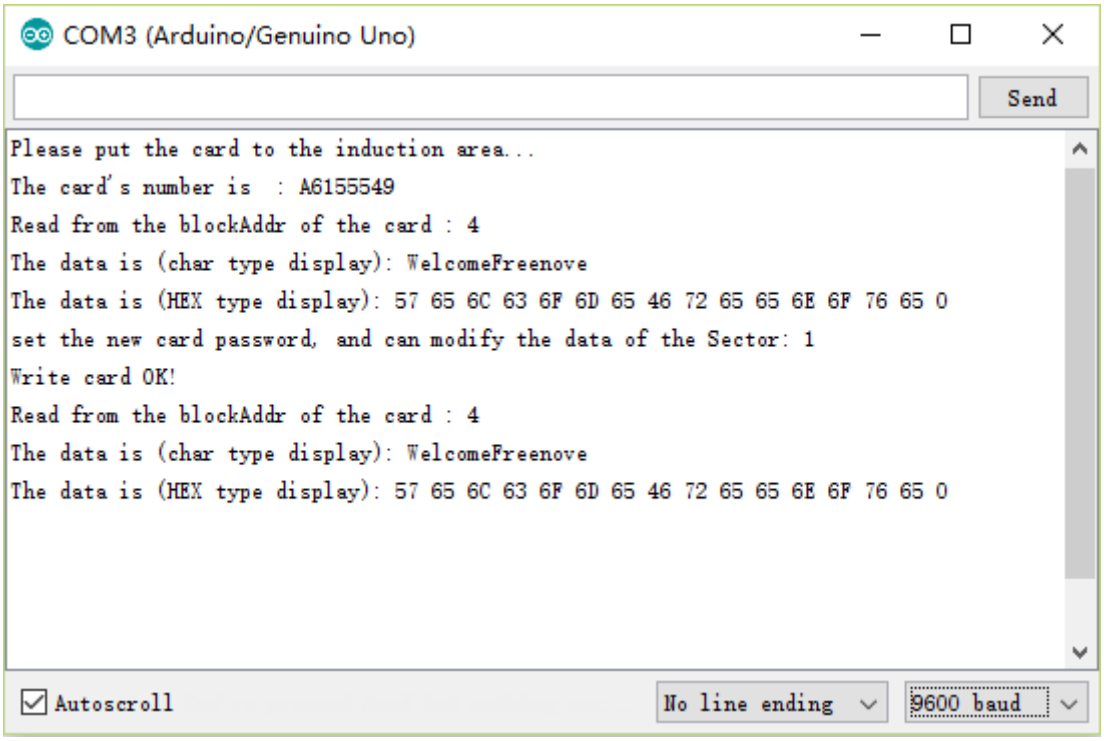
In the sub function of writeCard () and readCard (), we must first verify the password A, and then use the corresponding sub function to read and write. Here we do reading and writing operations to data block 0 (absolute NO.4) of the first sector.

```
  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.

```
  //first, read the data of data block 4
  readCard(4);
  //write data(within 16 bytes) to data block
  writeCard(4);
  //then read the data of data block again
  readCard(4);
```

After verifying and uploading the code to UNO, 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?

Thanks for your reading.

This book is all over here. If you find any mistakes, missions or you have other ideas and questions about contents of this book or the kit and etc., please feel free to contact us, and we will check and correct it as soon as possible.

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and other interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

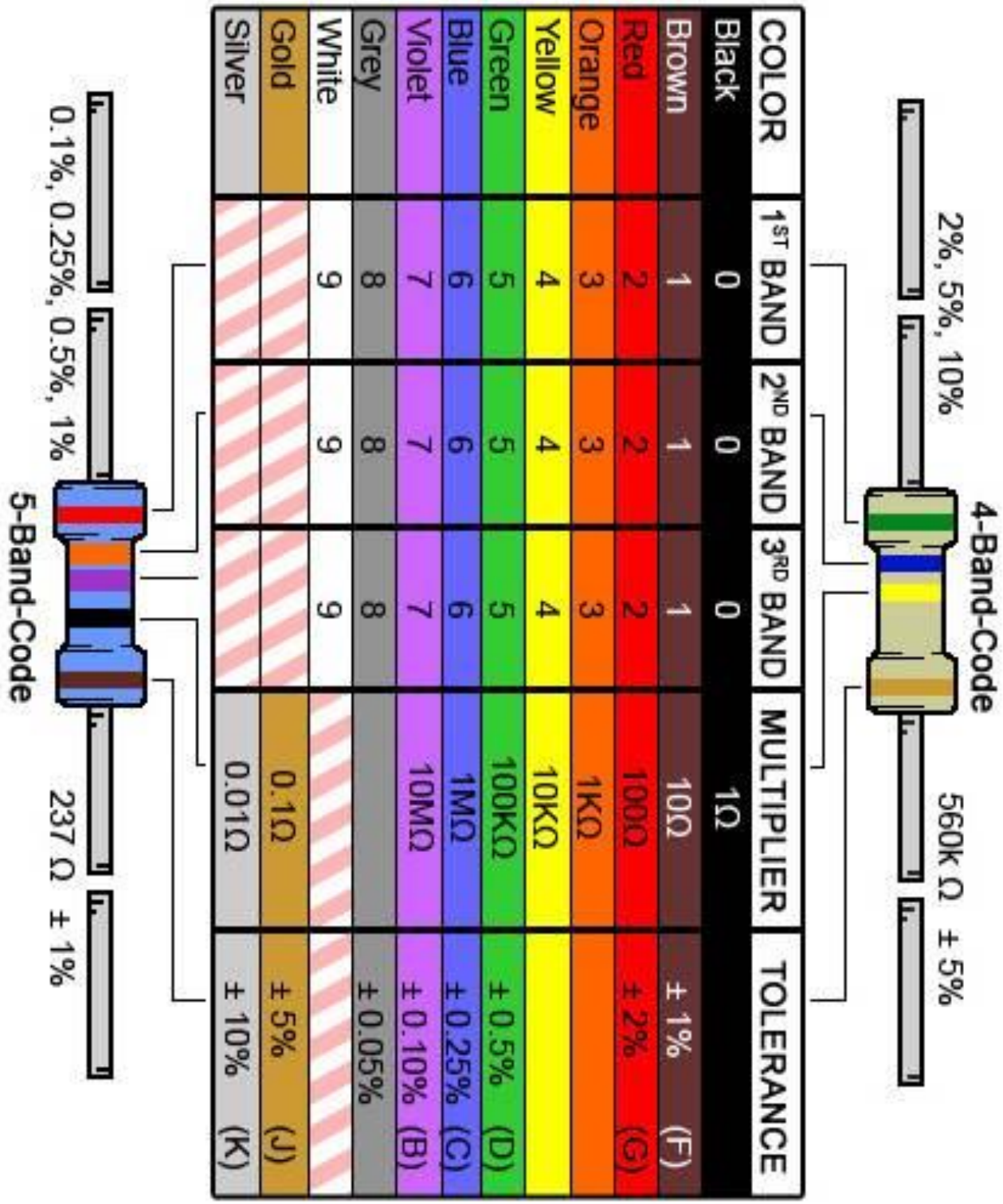
Thank you again for choosing Freenove products.

Appendix

ASCII table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	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	0	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



COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)