Welcome

Thank you for choosing Freenove products!

How to Start

When reading this, you should have downloaded the ZIP file for this product.
Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

! Unzip the ZIP file instead of opening the file in the ZIP file directly.
! Do not move, delete or rename files in the folder just unzipped.

Unpack

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

Get Support

Encounter problems? Don't worry! Refer to “TroubleShooting.pdf” or contact us.

When there are packaging damage, quality problems, questions encountering in use, etc., just send us an email. We will reply to you within one working day and provide a solution.

support@freenove.com

Attention

Pay attention to safety when using and storing this product:

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

About

Freenove provides open source electronic products and services.

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

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

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

http://www.freenove.com

Copyright

Contents

Preface

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

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

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

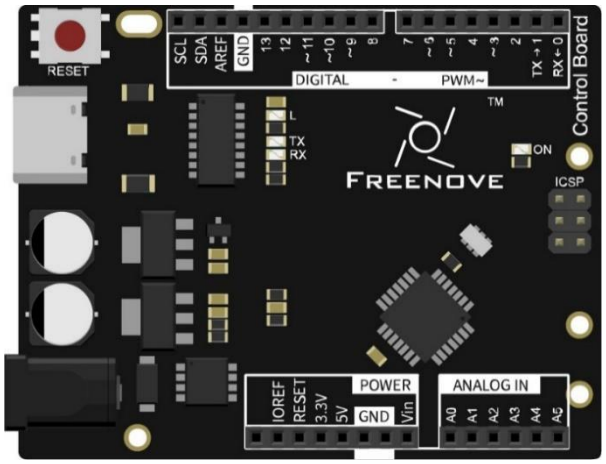
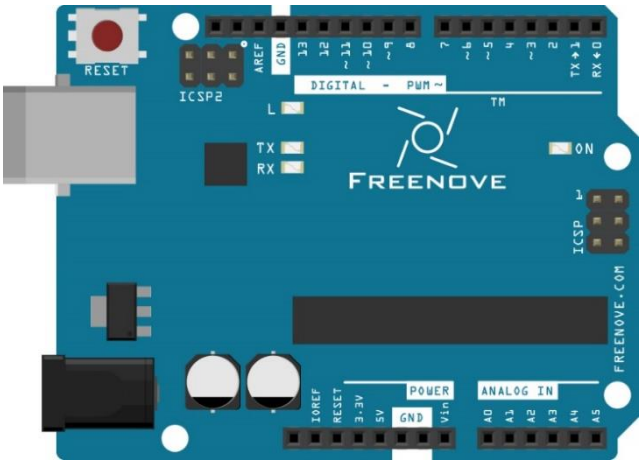
Support email: support@freenove.com

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

Control Board

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

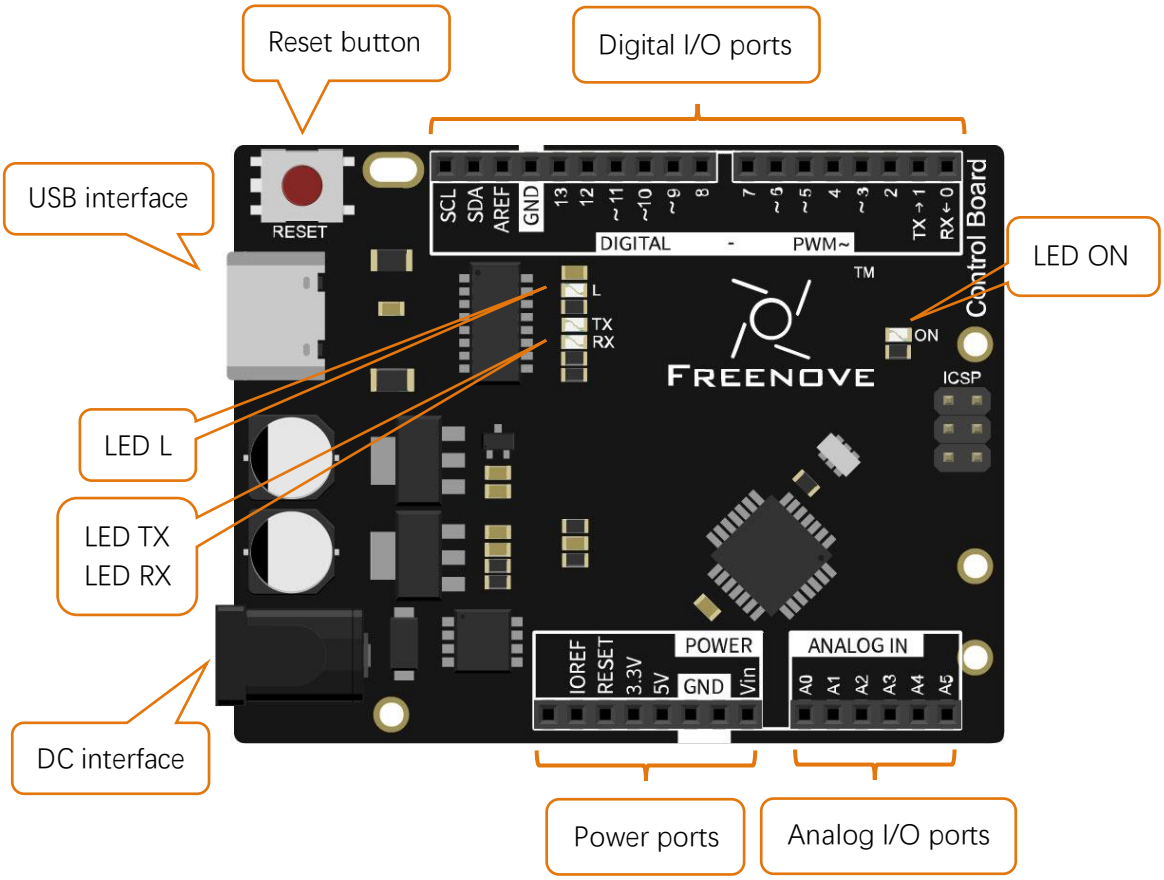
There are multiple versions of Freenove control board. Your purchase may be one of the following:

Freenove Control Board (Black)	Freenove Control Board (Blue)
	

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

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

Diagram of the Freenove control board is shown below:



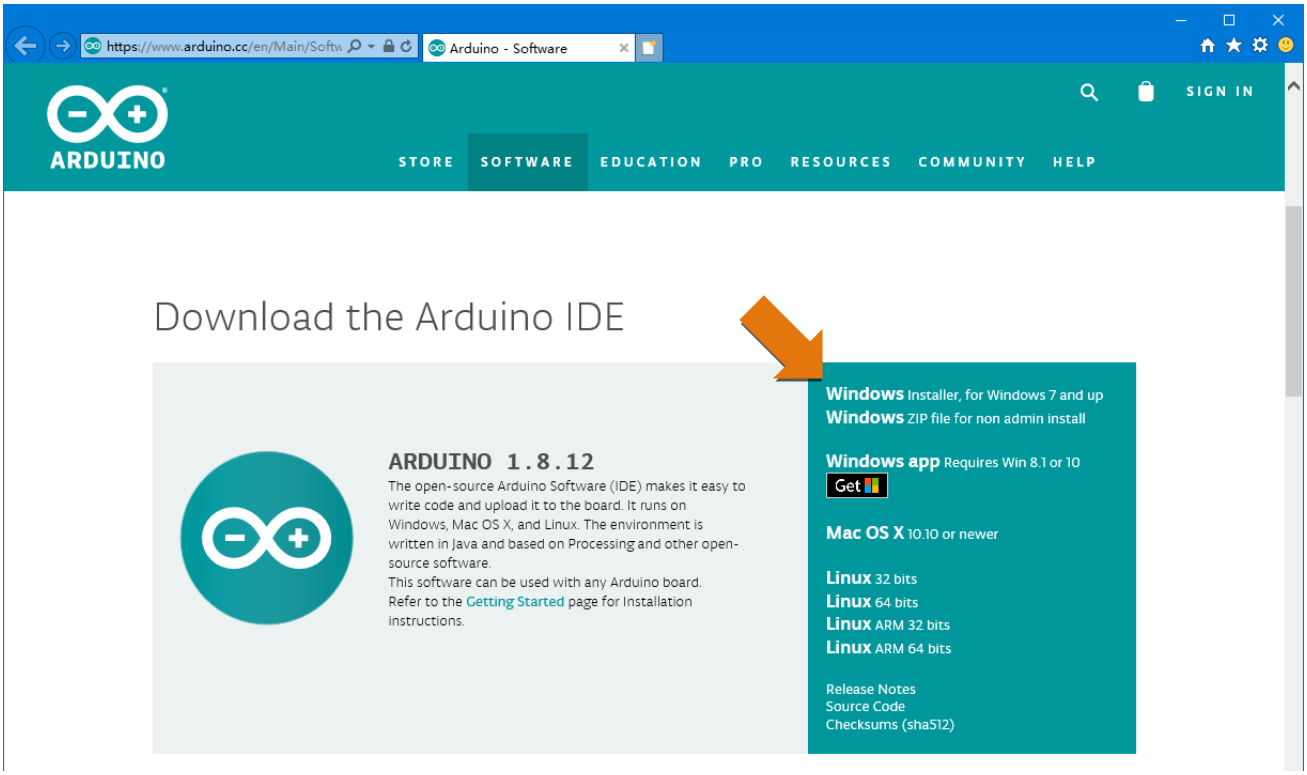
- Digital I/O ports is used to connect to other components or modules, to receive an input signal, or to send a control signal. Usually, we name it by adding a "D" in front of the number, such as D13 (pin 13).
- USB interface is used to provide power, upload code or communicate with PC.
- LED L is connected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

Programming Software

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

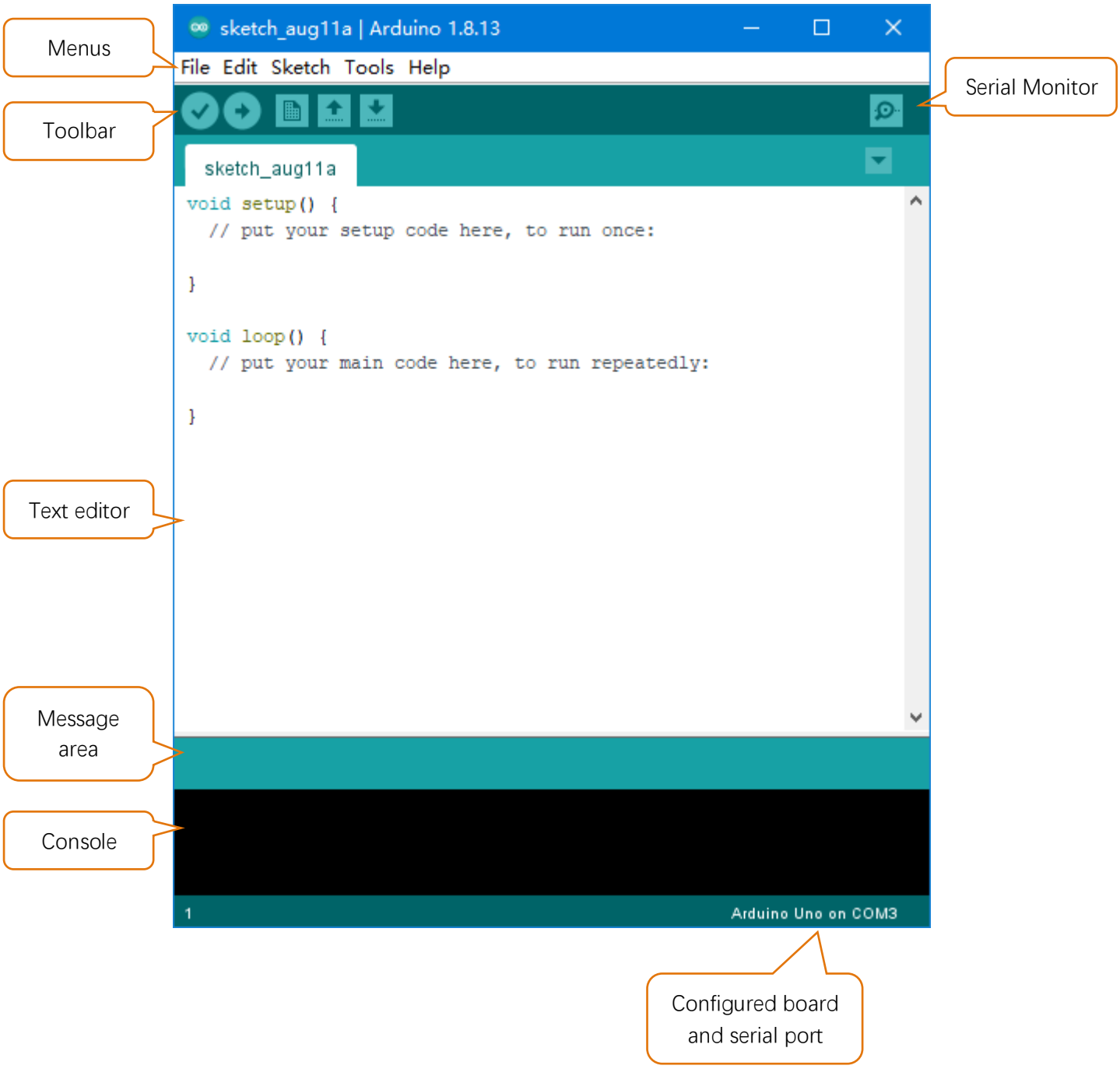
Arduino IDE uses C/C++ programming language. Don't worry if you have never used it, because this document contains programming knowledge and detailed explanation of the code.

First, install Arduino IDE. Visit https://www.arduino.cc/en/Main/Software. Select and download a corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer".



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

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



Programs written with Arduino IDE are called **sketches**. These sketches are written in a text editor and are saved with the file extension**.ino**. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino IDE, including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Verify
Checks your code for errors compiling it.

Upload
Compiles your code and uploads it to the configured board.

New
Creates a new sketch.

Open
Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

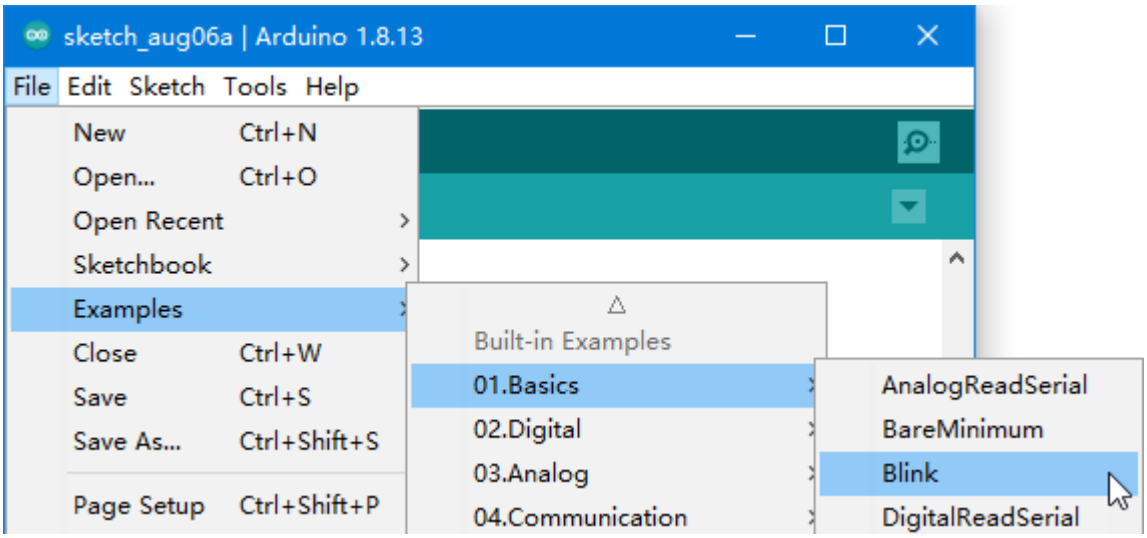
Save
Saves your sketch.

Serial Monitor
Opens the serial monitor.

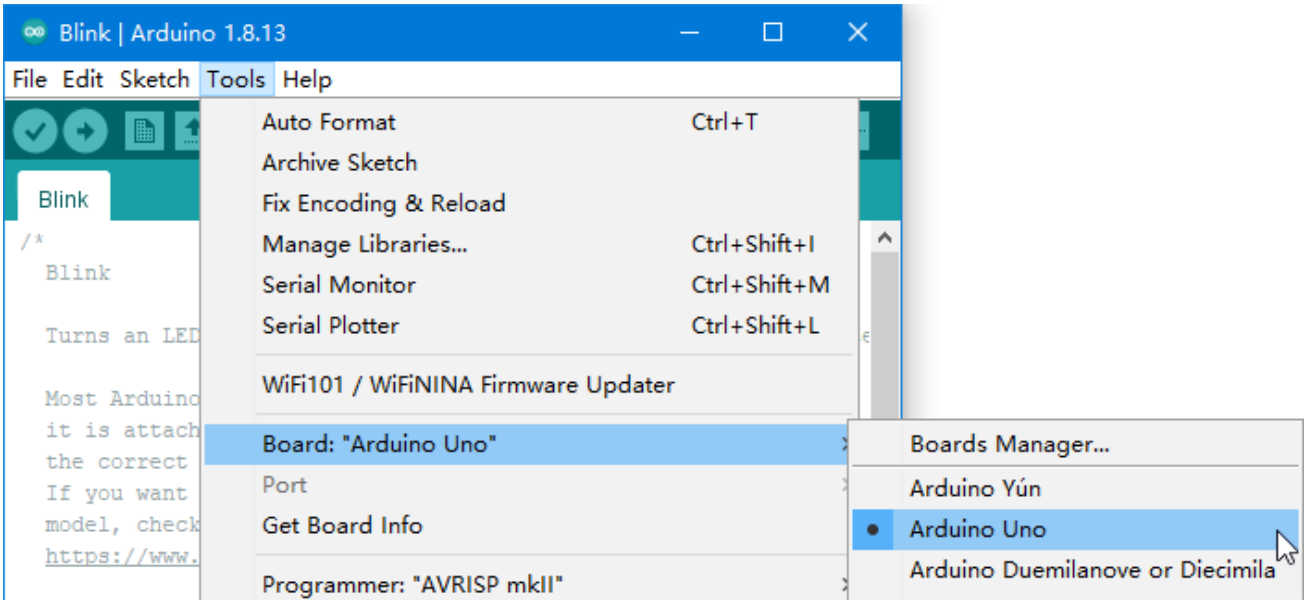
Additional commands are found within five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

First Use

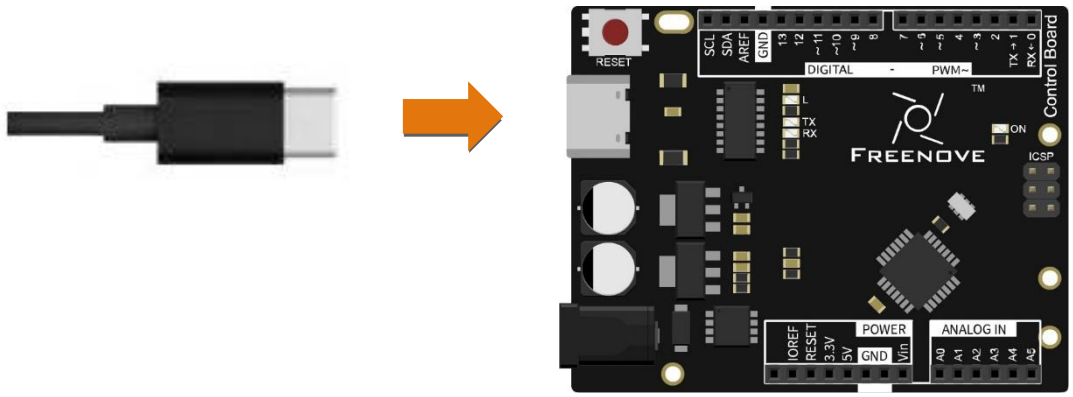
Open the example sketch "Blink".



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



Connect control board to your computer with USB cable.



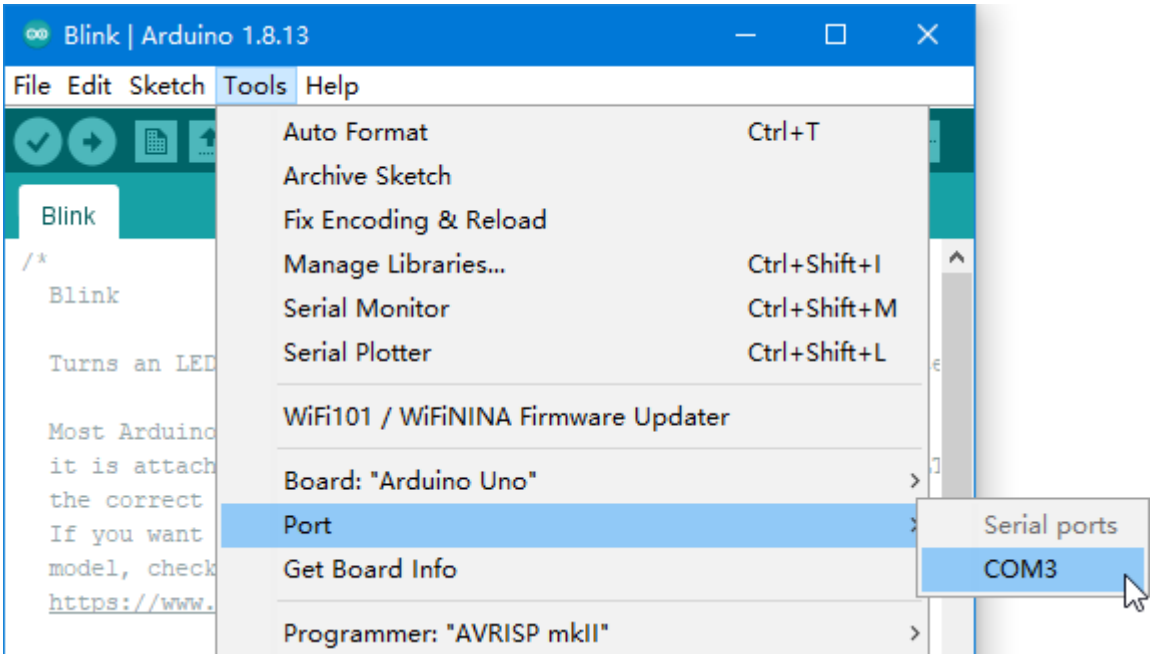
Select the port.

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

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

On Mac: It may be /dev/cu.usbserial-710, /dev/cu.usemodem7101 (Arduino Uno) or something like that.

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



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

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

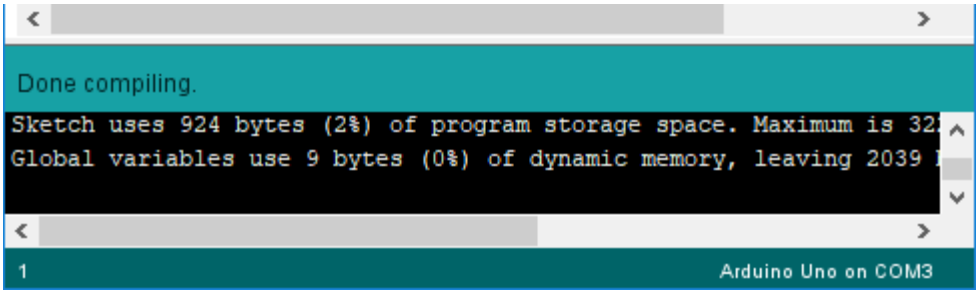
- For blue board, reinstall the latest version of Arduino IDE. During installation, agree to install the driver.
- For black board, see “InstallDriver.pdf” in “Drivers” folder (in the folder contains this Tutorial.pdf).

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

Click "Verify" button.

The following figure shows the code being compiled.

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



Click "Upload" button.

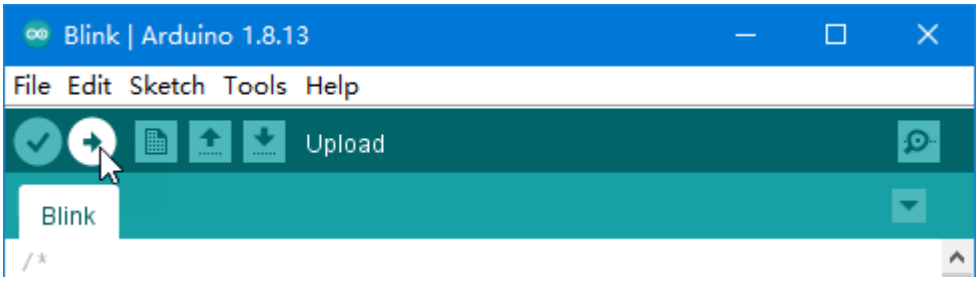
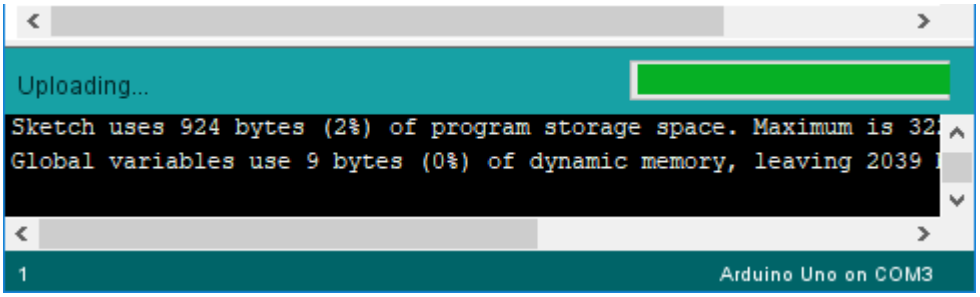
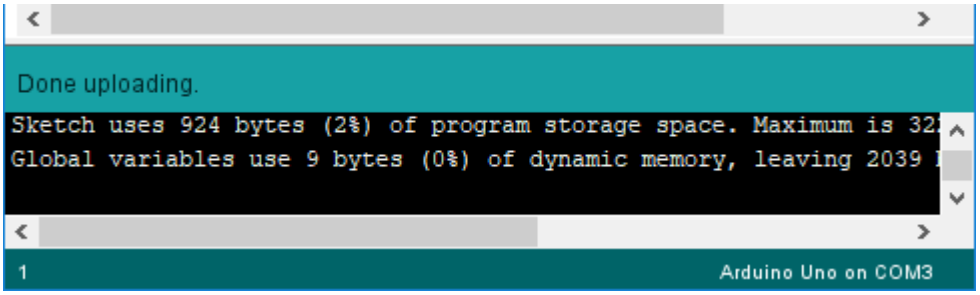


Figure below shows code are uploading.

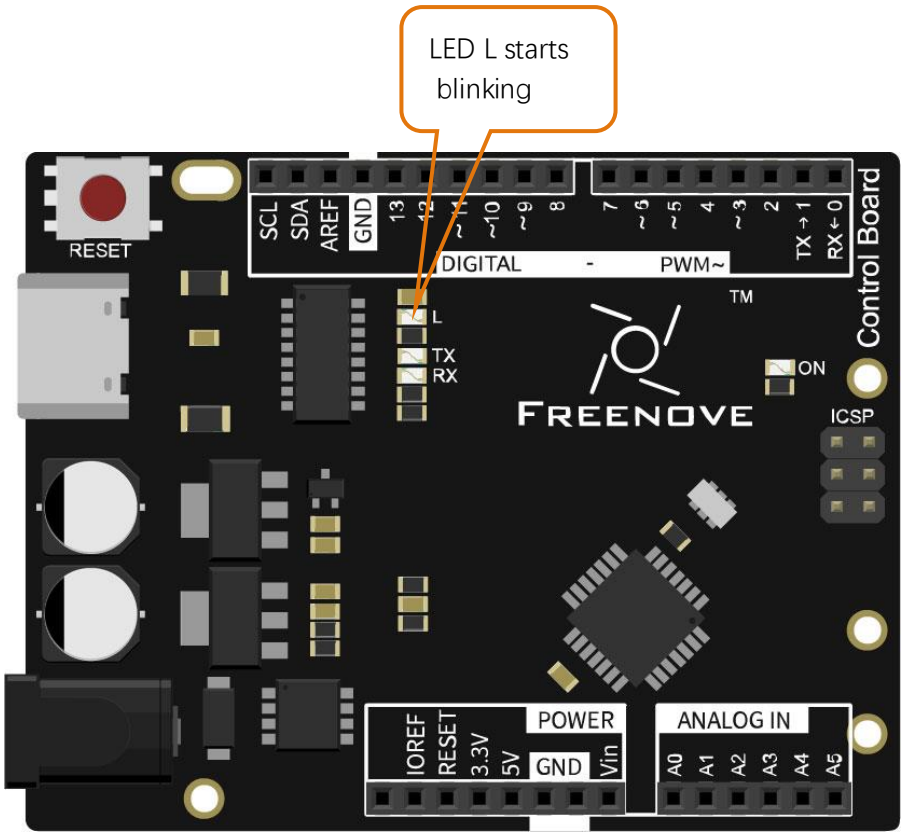


Wait a moment, and then the uploading is completed.



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

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



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

Chapter 1 LED Blink

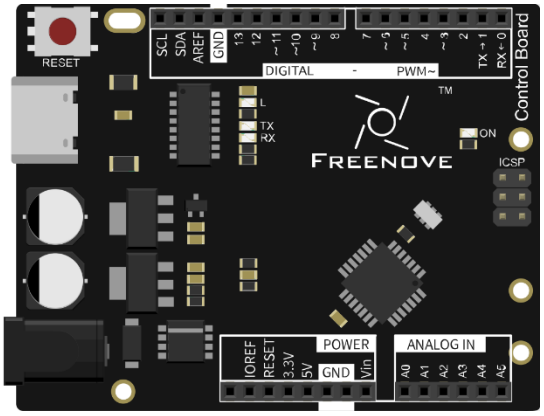
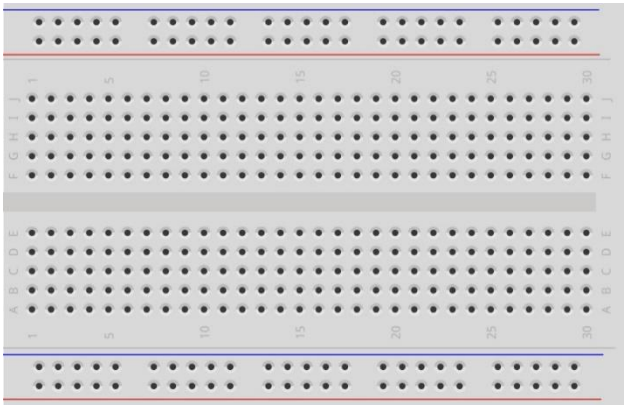
We have previously tried to make the LED marked with "L" blink on the control board. Now let us use electronic components and codes to reproduce the phenomenon, and try to understand their principle.

Project 1.1 Control LED with Manual Button

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

Component List

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

<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td rowspan="2">LED x1
  </td><td rowspan="2">Resistor 220Ω x1
  </td><td rowspan="2">Push button x1
  </td></tr>
<tr><td>Jumper M/M x2
  </td></tr>
</table>

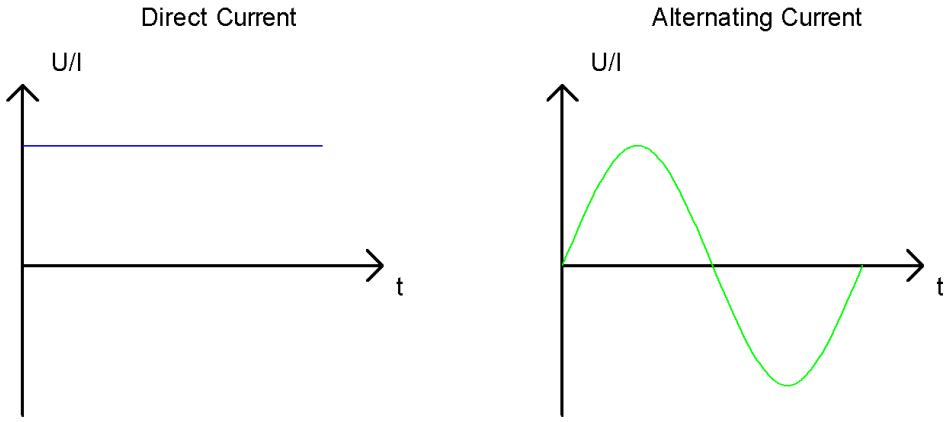
Circuit Knowledge

Power supply

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

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

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



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

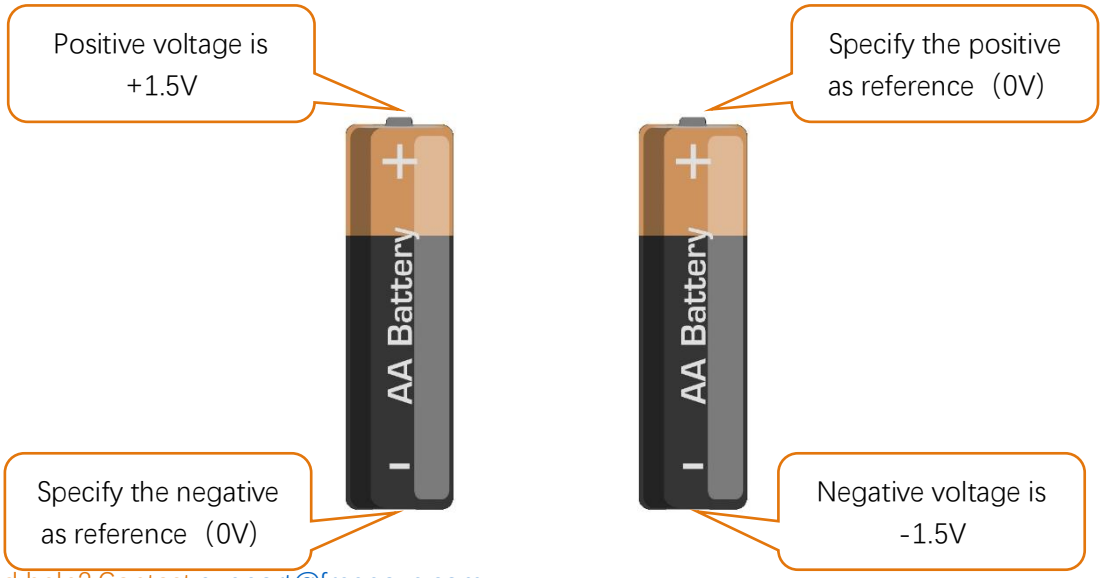
Battery or battery pack can be represented by the following symbols:

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

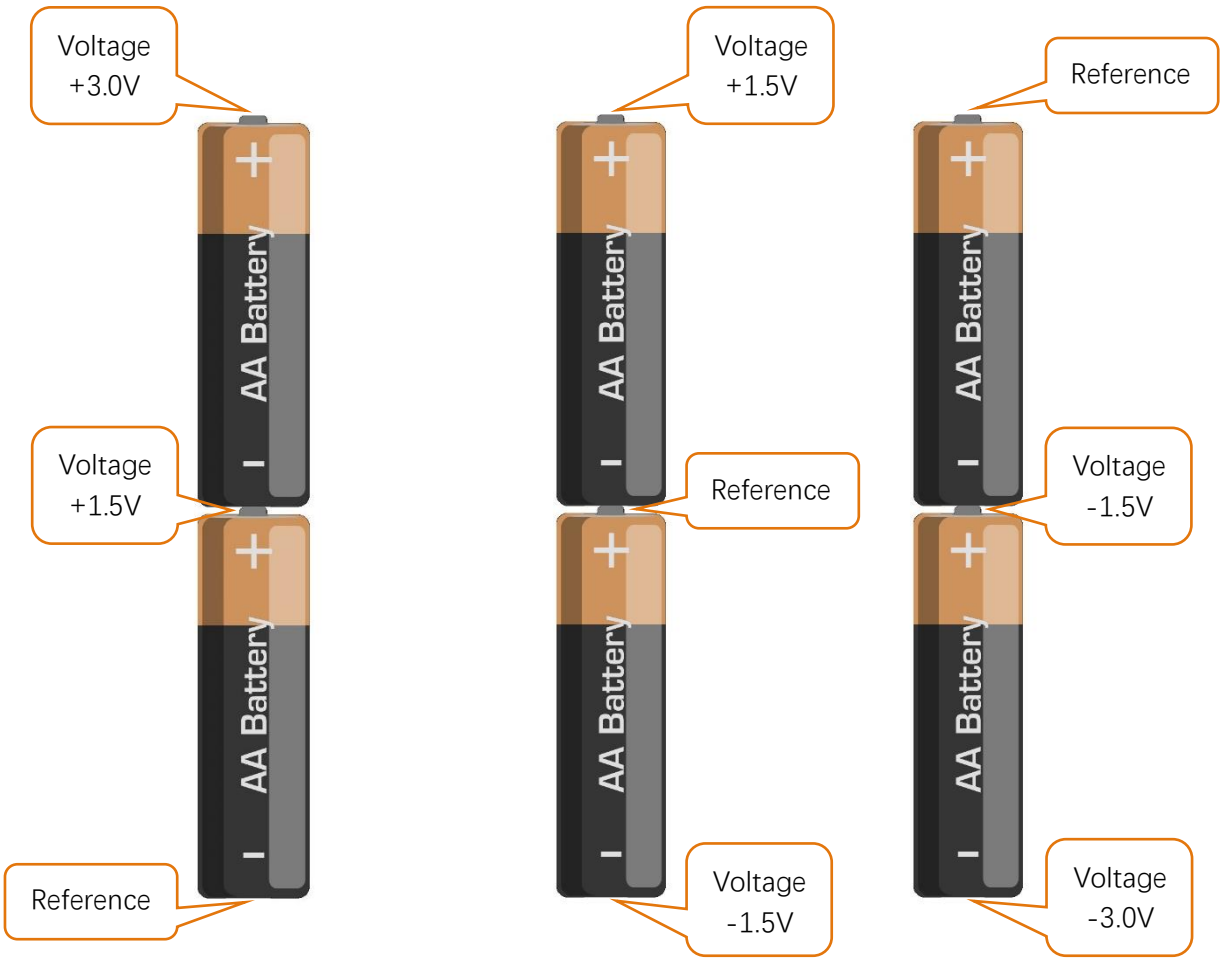
Voltage

The unit of voltage(U) is volt(V). 1kV=1000V, 1V=1000mV, 1mV=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 poles of power supply is usually represented by the following two symbols:

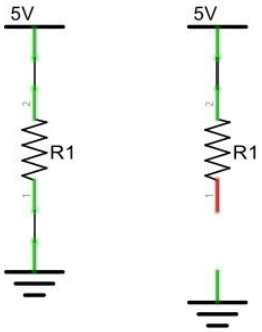
Positive(VCC) Negative(Ground, GND)

Current

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

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

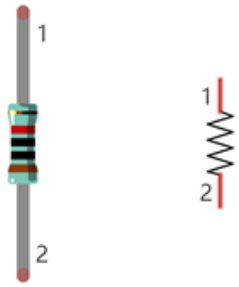
In the figures below: the left one is a loop circuit, so current flows through the circuit. The right one is not a loop circuit, so there is no current.



Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

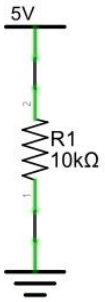
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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



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

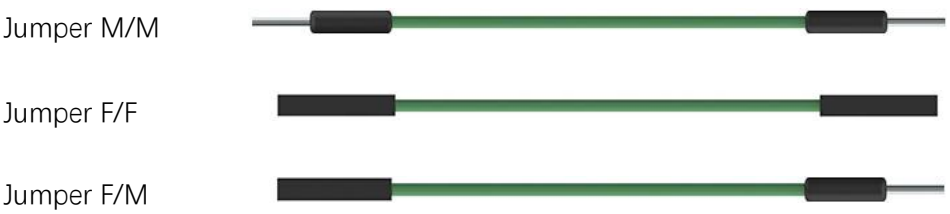
Component Knowledge

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

Jumper

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

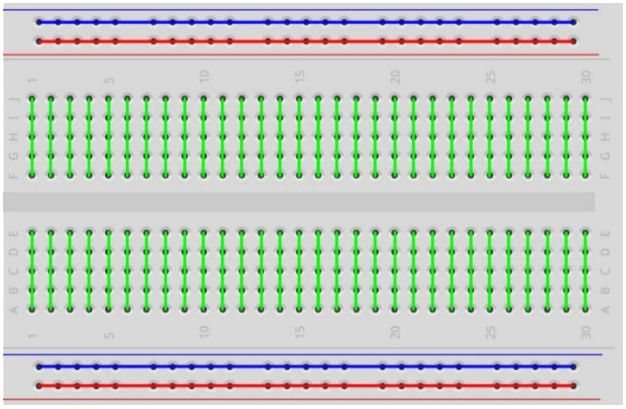
Jumpers have male end (pin) and female end (slot), so jumpers can be divided into the following 3 types.



Breadboard

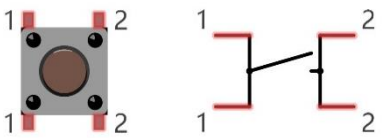
There are many small holes on breadboard to connect Jumpers.

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



Push Button Switch

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

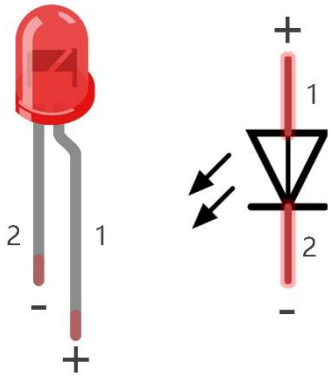


When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

LED

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

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



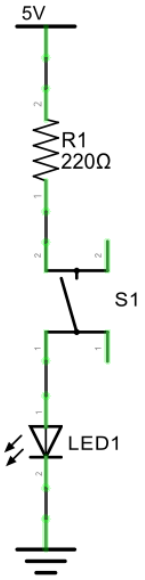
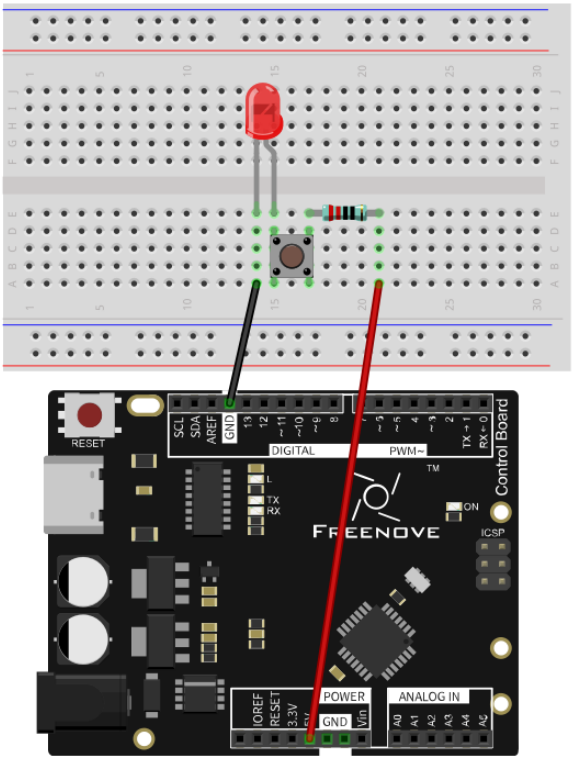
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Circuit

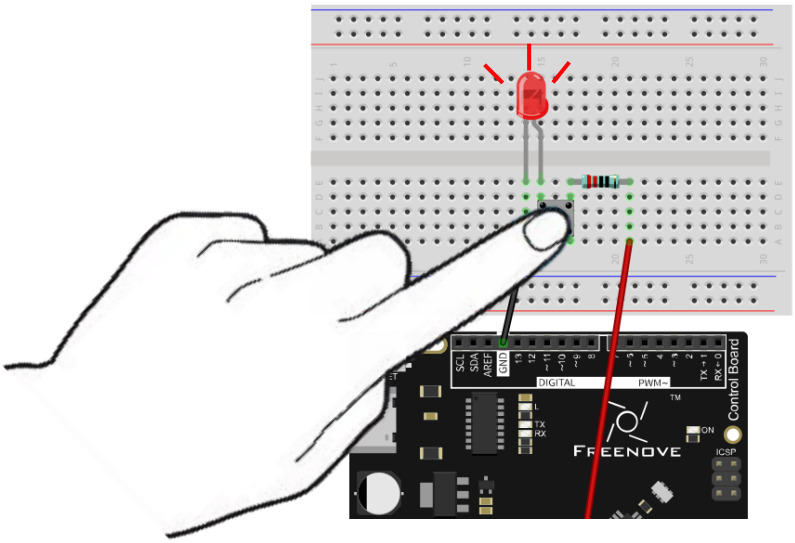
In this project, the LED is controlled by a push button switch, and the control board here only plays the role of power supply in the circuit.

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

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

Schematic diagram	Hardware connection
	

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



Project 1.2 Control LED with Control Board

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

Component List

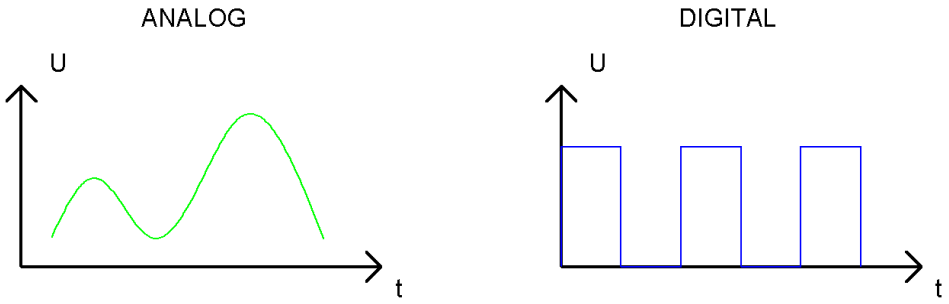
Components are basically the same with those in last section. Push button switch is no more needed.

Circuit Knowledge

Analog signal and Digital signal

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

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



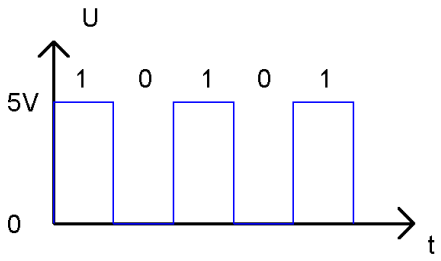
In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

Low level and high level

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

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

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



Code Knowledge

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

Comments

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

1. Symbol "//"

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

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

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

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

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

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

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

Or

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

Data type

When programming, we often use digital, characters and other data. C language has several basic data types as follows:
int: A number that does not have a fractional part, an integer, such as 0, 12, -1;
float: A number that has a fractional part, such as 0.1, -1.2;
char: It means character, such as 'a', '@', '0';
For more about date types, please visit the website: https://www.Arduino.cc-Resources-Reference-Data Types.

Constant

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

Variable

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

```
1  int i;
```

"int" indicates the type, ";" indicates the end of the statement. The statement is usually written in one single line; and these statements form the code.
After declaration of the variable, you can use it. The following is an assignment to a variable:

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

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

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

```
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

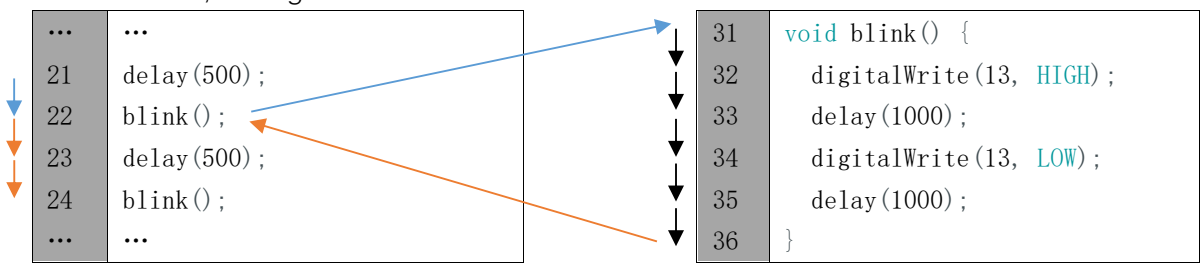
A function is a collection of statements with a sequence of order, which performs a defined task. Let's define a function void blink() as follows:

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

"void" indicates that the function does not return a value (Chapter 4 will detail the return value of functions);
"()" its inside is parameters of a function (Chapter 2 will detail the parameters of the functions). No content inside it indicates that this function has no parameters;
"{}" contains the entire code of the function.
After the function is defined, it is necessary to be called before it is executed. Let's call the function void blink(), as shown below.

```
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.



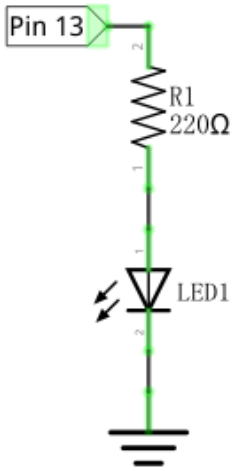
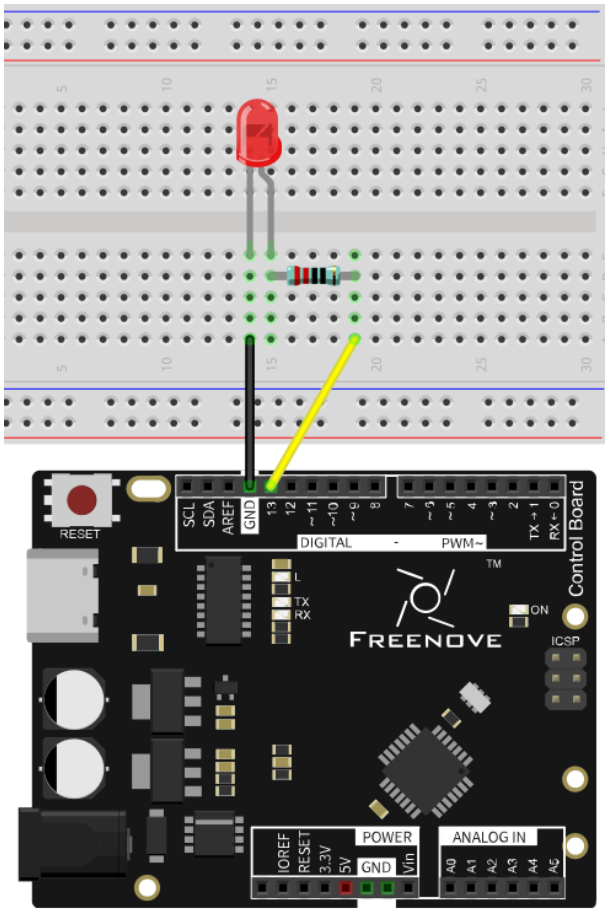
Some functions have one or more parameters. When you call such functions, you need to write parameters inside "()":

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

Circuit

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

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 1.2.1

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

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

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

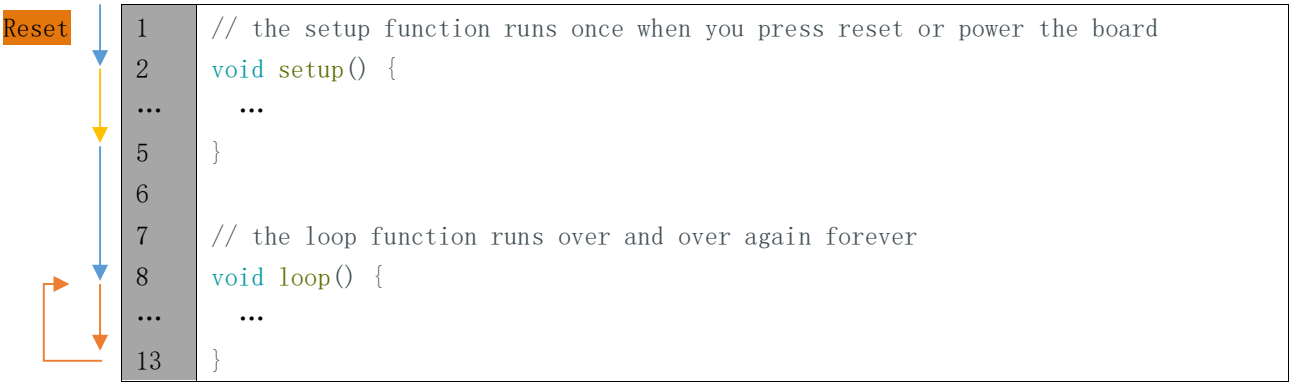
The code usually contains two basic functions: void setup() and void loop().

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

setup() function is generally used to write code to initialize the hardware. And loop() function is used to write code to achieve certain functions. loop() function is executed repeatedly. When the execution reaches the end of loop(), it will jump to the beginning of loop() to run again.

Reset

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



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

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

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

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

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

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

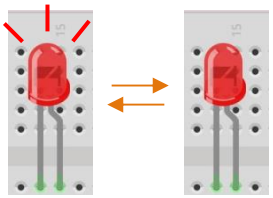
Then set the 13 pint to output low level, and LED light off. One second later, the execution of loop () function will be completed.

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

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

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

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



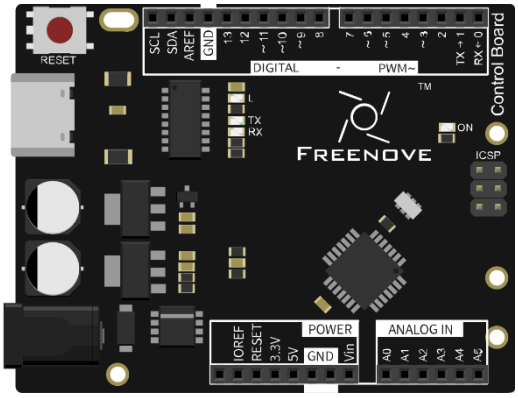
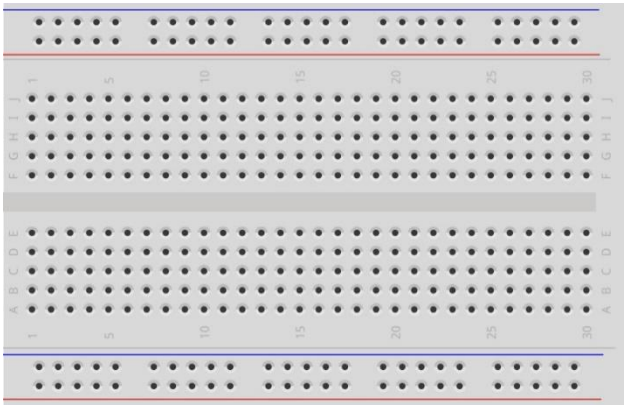
Chapter 2 Two LEDs Blink

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

Project 2.1 Two LEDs Blink

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

Component List

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

Code Knowledge

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

Parameters of function

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

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

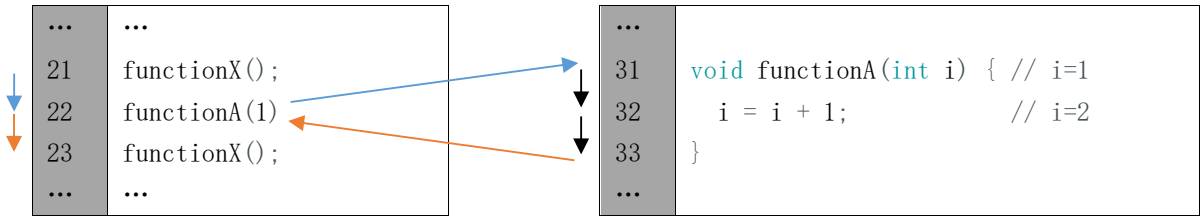
Next, we will define a function with a parameter as below:

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

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

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

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



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

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

Boolean data type

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

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

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

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

Logical operator

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

&&	true	false
true	true	false
false	false	false

||	true	false
true	true	true
false	true	false

!	
true	false
false	true

For example:

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

Relation operator

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

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

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

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

Conditional statement

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

```
1  int i = 0;
2  if (i == 0)    // true
3    i = 1;       // execution
4  functionX();
…  …
```

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

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

```
1  int i = 0,  j = 0;
2  if (i == 0) { // true
3    i = 1;       // execution
4    j = 1;       // execution
5  }
6  functionX();
…  …
```

```
1  int i = 0,  j = 0;
2  if (i != 0) { // false
3    i = 1;       // no execution
4    j = 1;       // no execution
5  }
6  functionX();
…  …
```

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

```
1  int i = 0;
2  if (i == 0)    // true
3    i = 1;       // execution
4  else
5    i = 1;       // no execution
6  functionX();
…  …
```

```
1  int i = 0;
2  if (i != 0)    // false
3    i = 1;       // no execution
4  else
5    i = 1;       // execution
6  functionX();
…  …
```

In addition, it can judge multiple conditions.

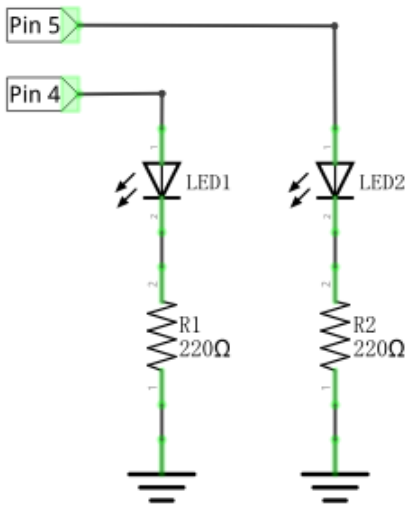
```
1   int i = 1;
2   if (i == 0)       // false
3     i = 1;          // no execution
4   else if (i == 1)// true
5     i = 2;          // execution
6   else if (i == 2)
7     i = 3;          // no execution
8   else
9     i = 4;          // no execution
10  functionX();
…   …
```

```
1   int i = 3;
2   if (i == 0)       // false
3     i = 1;          // no execution
4   else if (i == 1)// false
5     i = 2;          // no execution
6   else if (i == 2)// false
7     i = 3;          // no execution
8   else
9     i = 4;          // execution
10  functionX();
…   …
```

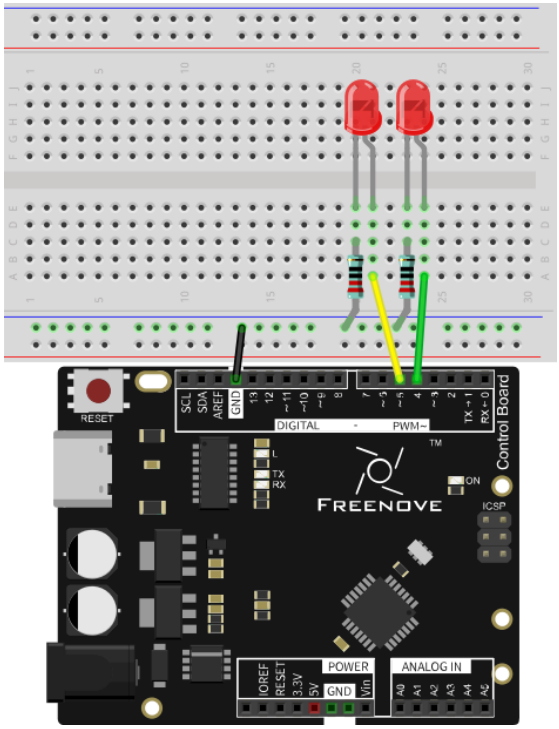
Circuit

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

Schematic diagram



Hardware connection



Sketch

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

Sketch 2.1.1

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

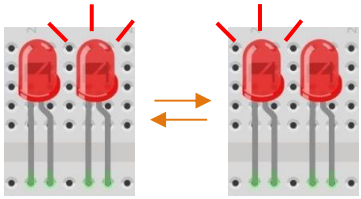
```
// set pin numbers:
int led1Pin = 4;            // the number of the LED1 pin
int led2Pin = 5;            // the number of the LED2 pin

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

void loop() {
  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
}
```

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

Variable scope

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

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



Sketch 2.1.2

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

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

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

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

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

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

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

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

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

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

HIGH and LOW

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

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

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

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

Sketch 2.1.3

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

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

```
// 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, and then two LEDs blink alternatively.

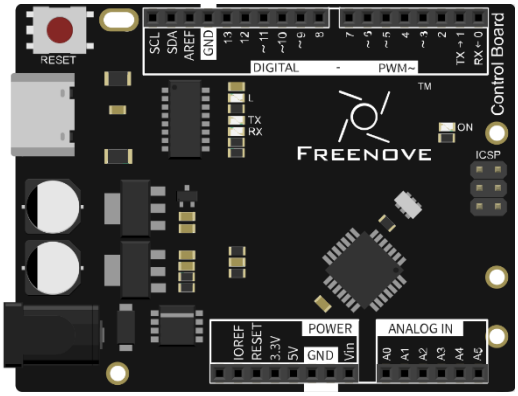
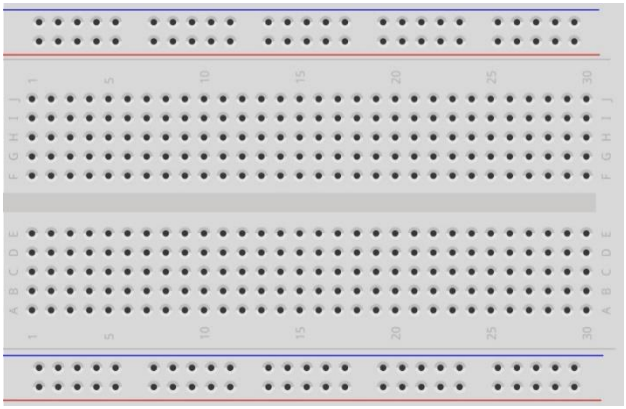
Chapter 3 LED Bar Graph

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

Project 3.1 LED Bar Graph Display

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

Component List

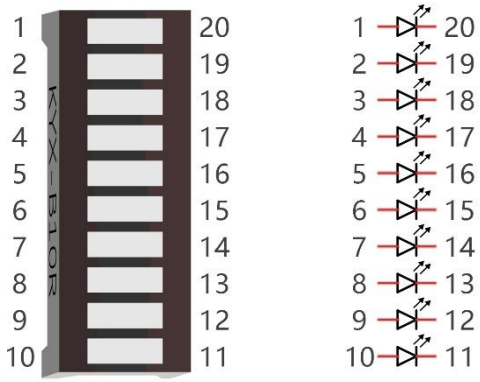
Control board x1 	Breadboard x1 	
USB cable x1 	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

Component Knowledge

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

LED bar graph

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



Code Knowledge

This section will use new code knowledge.

Array

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

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

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

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

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

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

We can reference elements of an array as below:

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

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

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

```
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 judging the expression inside "()". If the result is false, the statements inside "{}" will not be executed; if result is true, the statements will be executed.

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

For example:

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

First time: i=0, i<2 is tenable, execute j++, and execute i++, then i=1, j=1;
Second time: i=1, i<2 is tenable, execute j++, and execute i++, then i=2, j=2;
Third time: i<2 is tenable, i=2 is not tenable. The execution of loop statements is completed. Statement i=5 will be executed next.

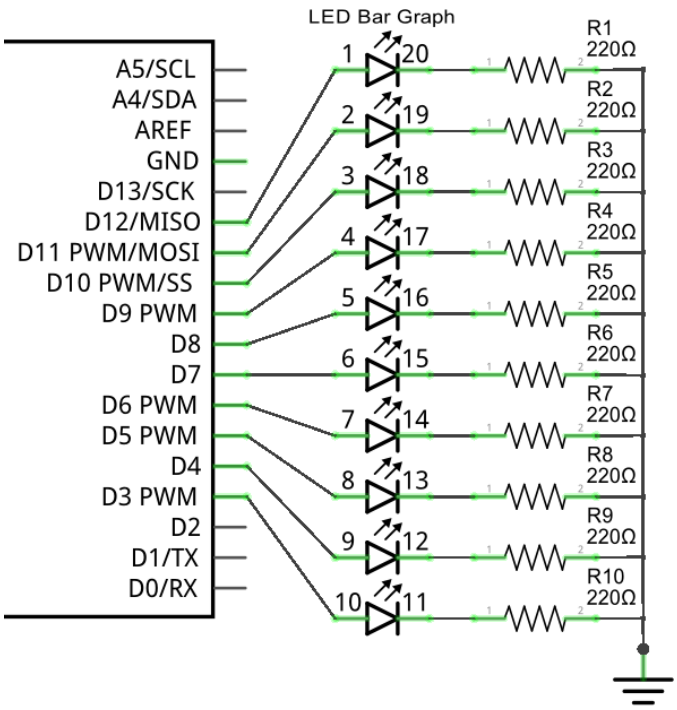
Operator ++, --

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

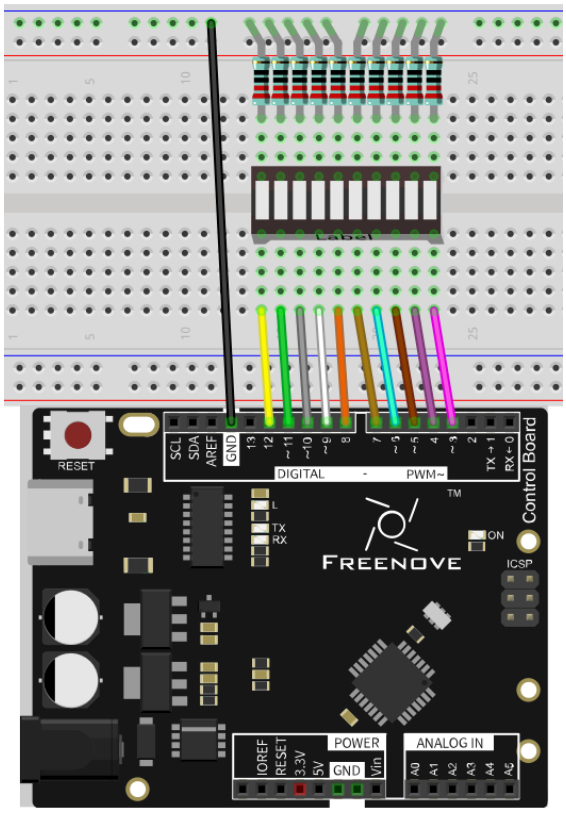
Circuit

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

Schematic diagram



Hardware connection



Sketch

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

Sketch 3.1.1

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

```
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 as other variables. But read-only variables can only be assigned once.

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

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

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

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

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

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

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

```
13 void loop() {
14   // make the "i"th LED of LED bar graph on and the others off in turn
15   for (int i = 0; i < ledCount; i++) {
16     barGraphDisplay(i);
17   }
18 }
```

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

Sketch 3.1.2

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

```
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() to make the LED bar graph light up in one direction, and then in a reverse direction repeatedly.

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

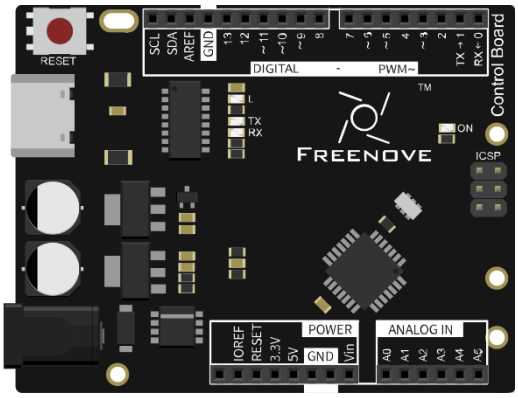
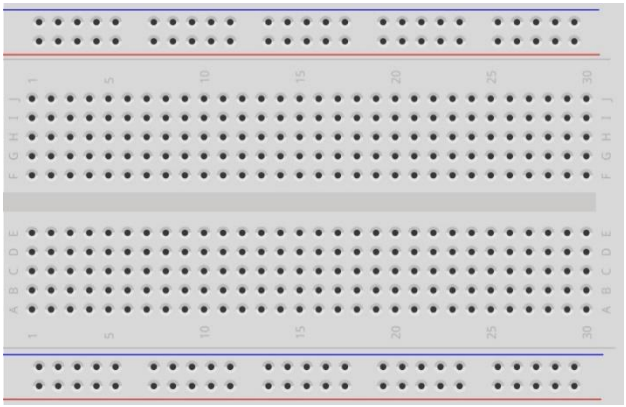
Chapter 4 LED Blink Smoothly

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

Project 4.1 LEDs Emit Different Brightness

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

Component List

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

Circuit Knowledge

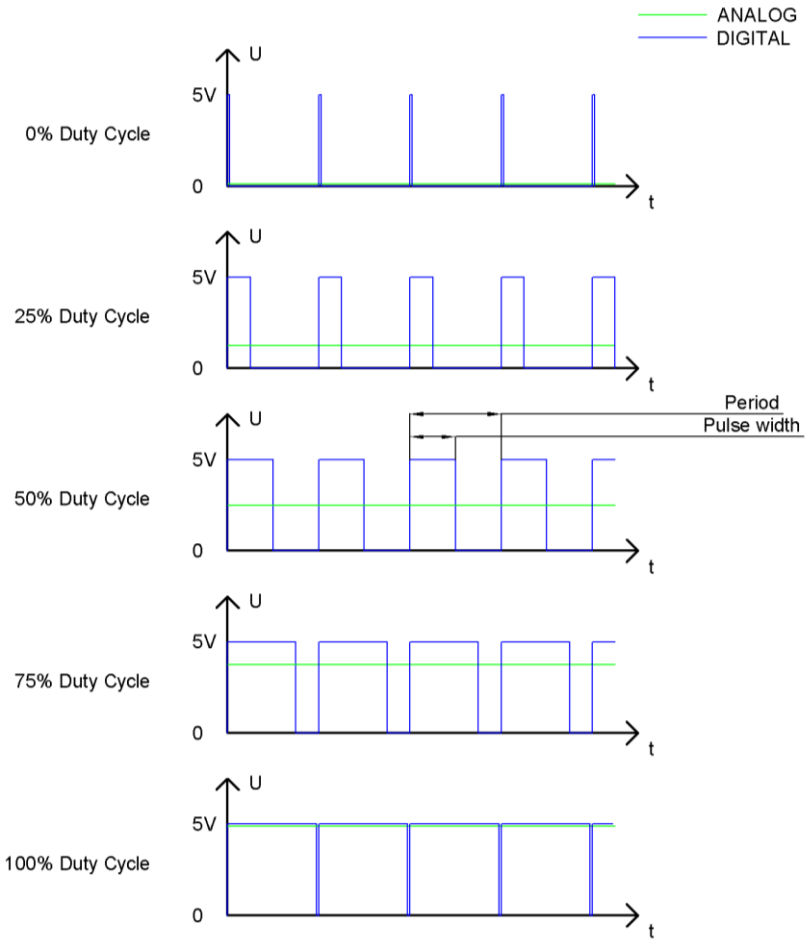
At first, let us learn how to use the circuit to make LED emit different brightness of light,

PWM

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

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

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



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

Code Knowledge

We will use new code knowledge in this section.

Return value of function

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

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

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

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

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

A function with return value is called as follows:

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

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

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

It is equivalent to the following code:

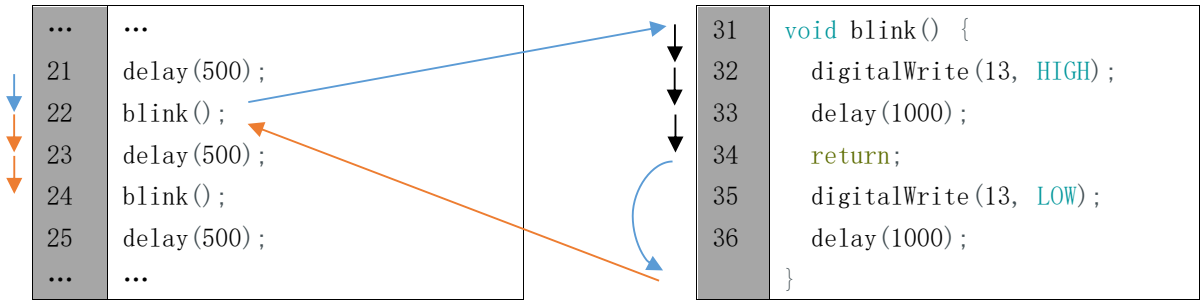
```
1  delay(300);
```

return

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

```
1  return;
```

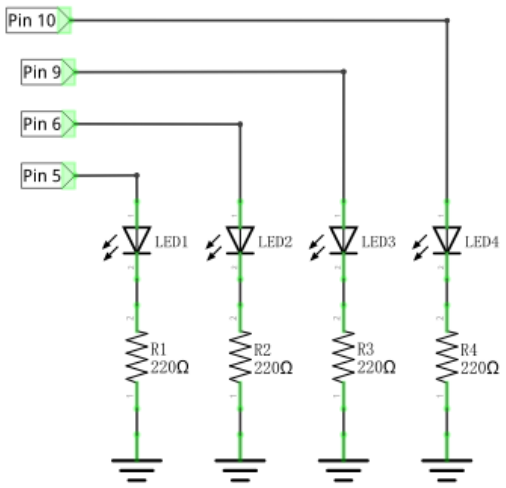
In this case, when the return statement is executed, the function will immediately end its execution rather than return to the end of the function. For example:



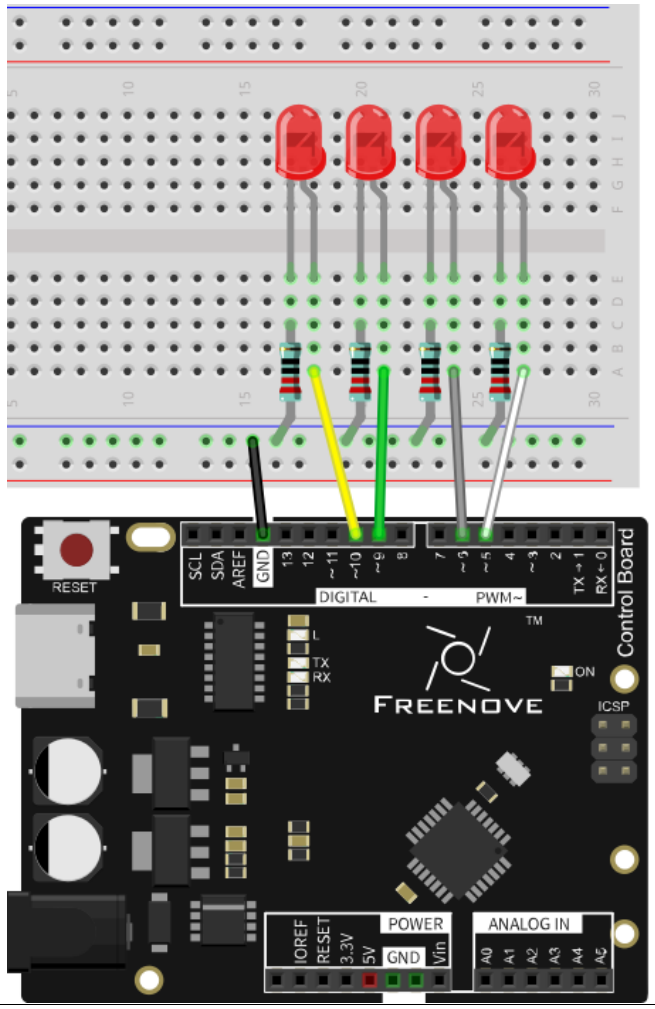
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 4.1.1

Now let us use sketch to make 4 LEDs emit different brightness of light. We will transmit signal to make the 4 ports connected to LEDs output the PWM waves with duty cycle of 2%, 10%, 50%, and 100% to let the LEDs emit different brightness of the light.

```
1  // set pin numbers:
2  int ledPin1 = 5,            // the number of the LED1 pin
3      ledPin2 = 6,            // the number of the LED2 pin
4      ledPin3 = 9,            // the number of the LED3 pin
5      ledPin4 = 10;           // the number of the LED4 pin
6
7  void setup() {
```

```
8     // initialize the LED pin as an output:
9     pinMode(ledPin1, OUTPUT);
10    pinMode(ledPin2, OUTPUT);
11    pinMode(ledPin3, OUTPUT);
12    pinMode(ledPin4, OUTPUT);
13  }
14
15  void loop()
16  {
17    // set the ports output PWM waves with different duty cycle
18    analogWrite(ledPin1, map(2, 0, 100, 0, 255));
19    analogWrite(ledPin2, map(10, 0, 100, 0, 255));
20    analogWrite(ledPin3, map(50, 0, 100, 0, 255));
21    analogWrite(ledPin4, map(100, 0, 100, 0, 255));
22  }
```

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

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

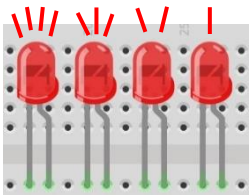
analogWrite(pin, value)

Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Only the digital pin marked with "~" symbol on the control board can use this function to output PWM waves. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255, which represents the duty cycle of 0%-100%.
In order to use this function, we need to set the port to output mode.

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

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

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



Project 4.2 LED Blinking Smoothly

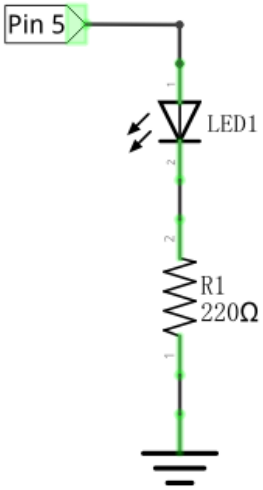
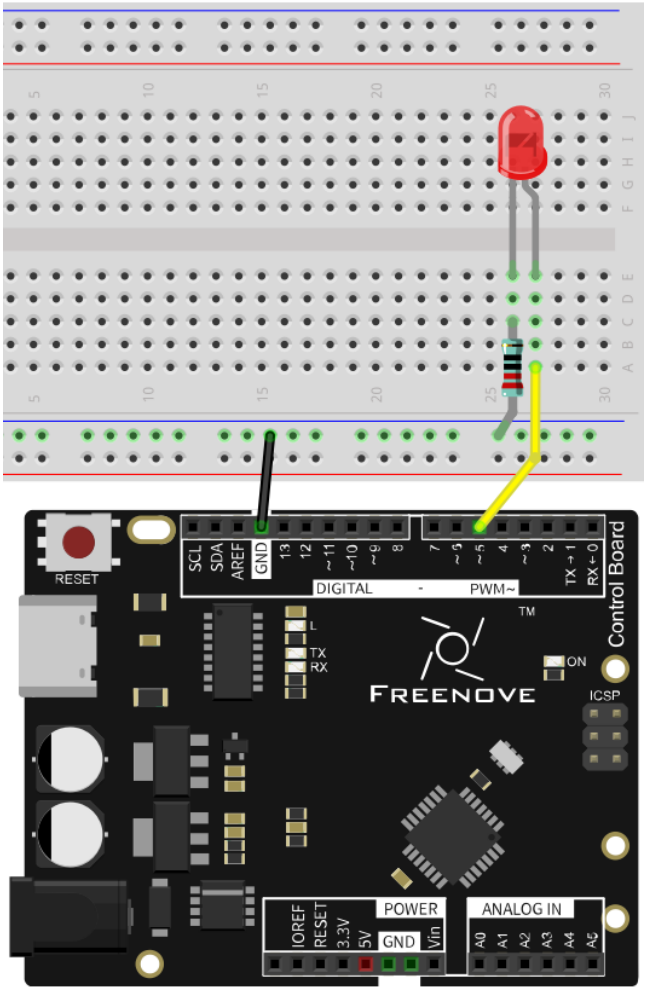
We will learn how to make a LED blink smoothly, that is, breathing light.

Component List

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

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

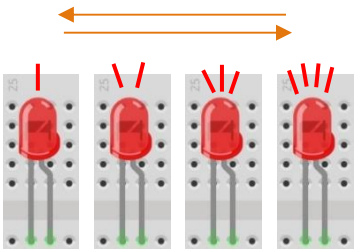
Sketch 4.2.1

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

```
// 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);         // correspongding duty cycle change from 0%-100%
    delay(delayMs);                 // adjust the rate of change in brightness
  }
}
```

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

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



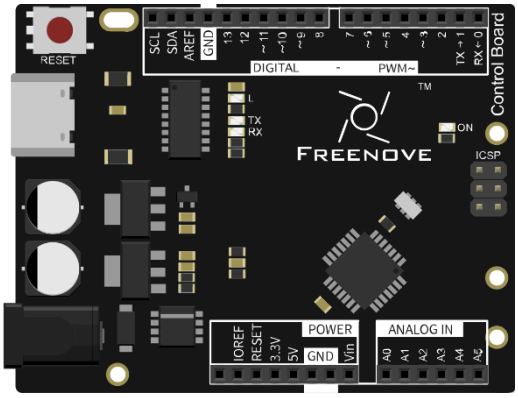
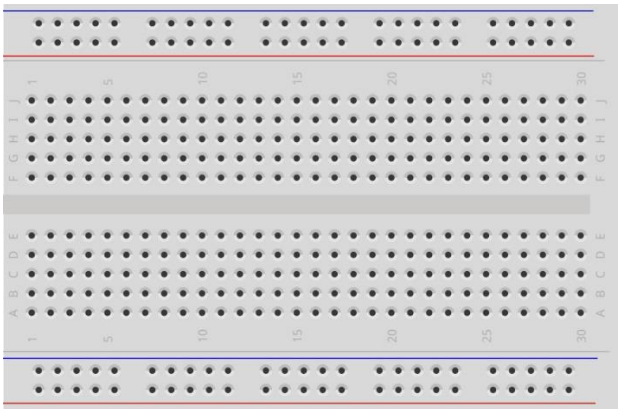
Chapter 5 Control LED with Push Button Switch

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

Project 5.1 Control LED with Push Button Switch

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

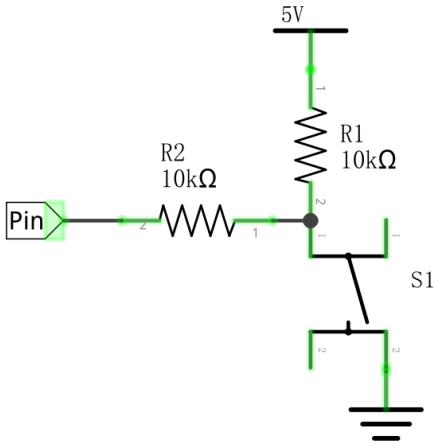
Component List

<table>
<tr><td colspan="2">Control board x1
  </td><td colspan="4">Breadboard x1
  </td></tr>
<tr><td colspan="2">USB cable x1
  </td><td>LED x1
  </td><td>Resistor
220Ω x1
  </td><td>Resistor
10kΩ x2
  </td><td>Push button
Switch x1
  </td></tr>
<tr><td colspan="2">Jumper M/M x4
  </td><td colspan="4"></td></tr>
</table>

Circuit Knowledge

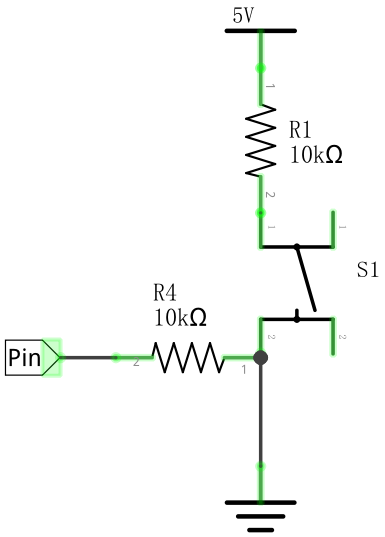
Connection of Push Button Switch

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



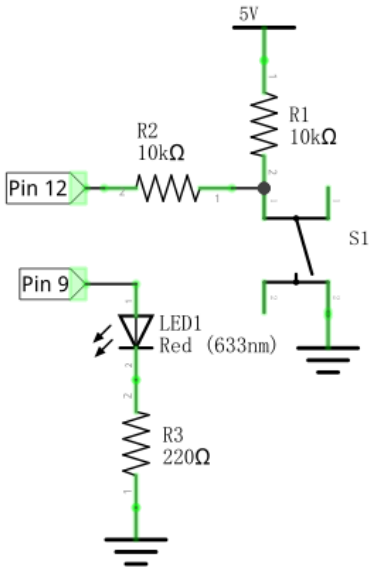
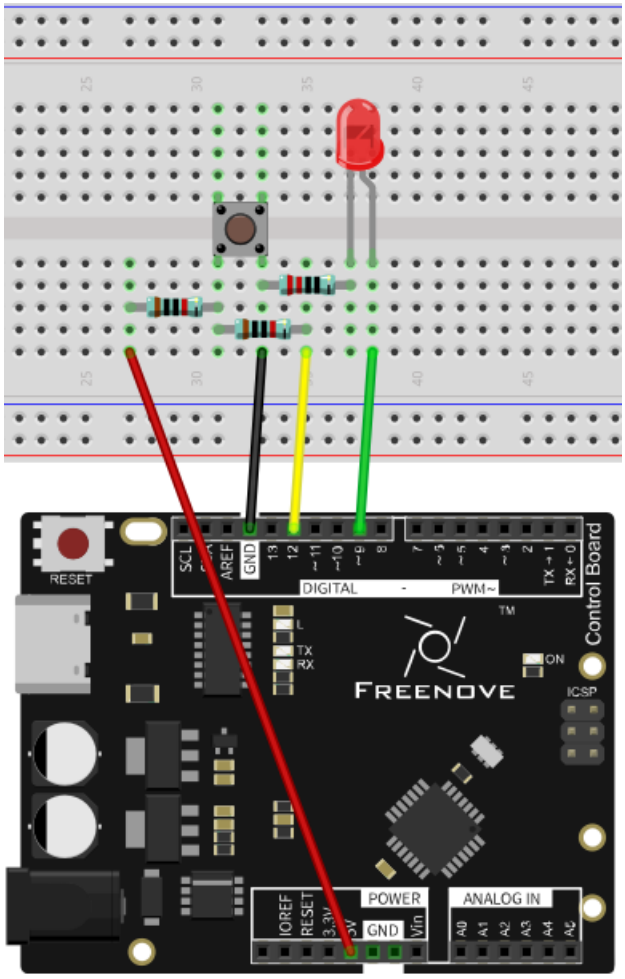
In the above circuit diagram, when the button is not pressed, 5V (high level) will be detected by control board port; and 0V (low level) when the button is pressed. The role of Resistor R2 here is to prevent the port from being set to output high level by accident. Without R2, the port could be connected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

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



Circuit

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

Schematic diagram	Hardware Connection
	

Sketch

Sketch 5.1.1

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

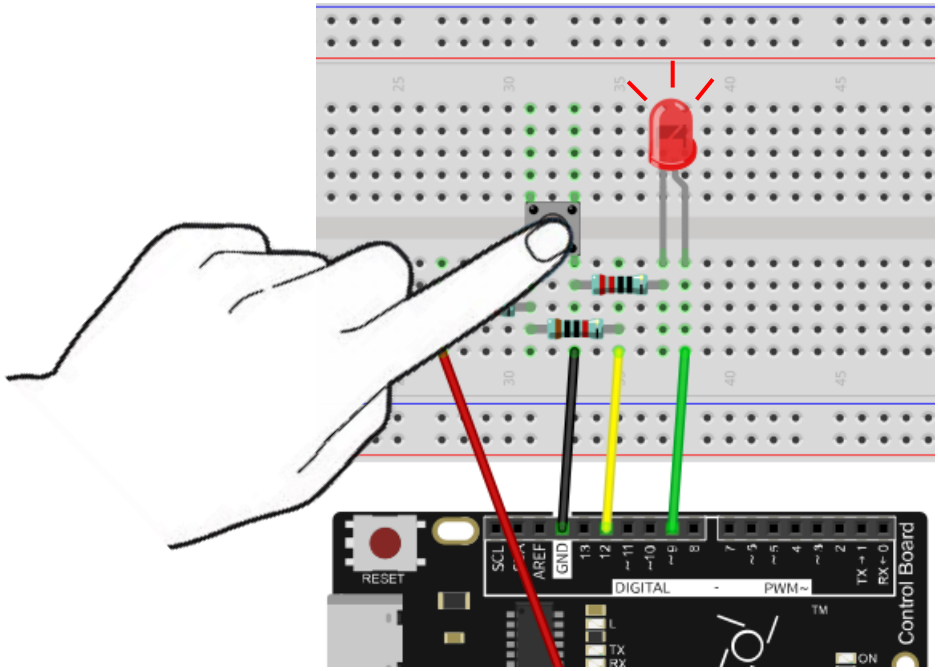
```
int buttonPin = 12; // the number of the push button pin
int ledPin = 9;     // the number of the LED pin

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 turned on or off in accordance with the state of the pin connected to push button switch.

digitalRead(pin)

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

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



Project 5.2 Change LED State with Push Button Switch

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

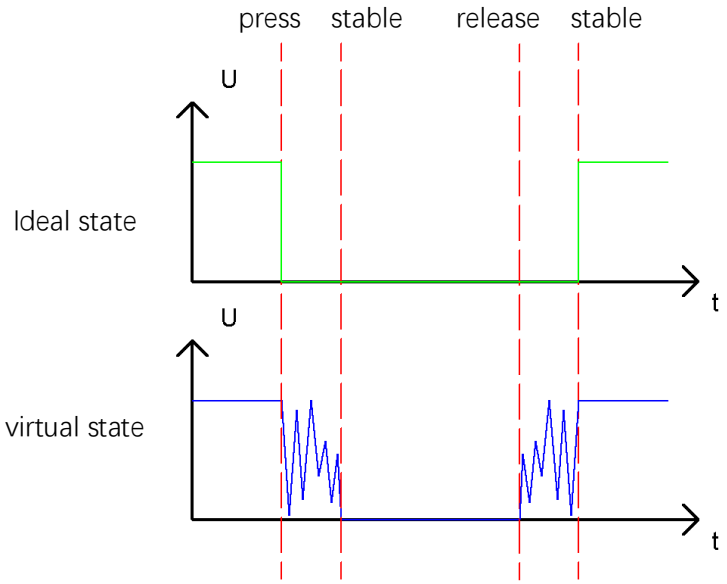
Component List

Same with the previous section.

Circuit Knowledge

Debounce a push button switch

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it stabilizes in a new state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.



Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

Circuit

Same with the previous section.

Sketch

Sketch 5.2.1

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

```
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
  }
}
```

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

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

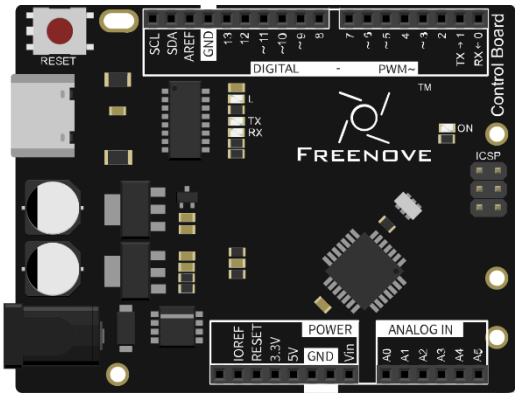
Chapter 6 Serial

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

Project 6.1 Send Data through Serial

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

Component List

Control board x1 	USB cable x1 

Code Knowledge

Bit and Byte

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

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

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

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

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

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

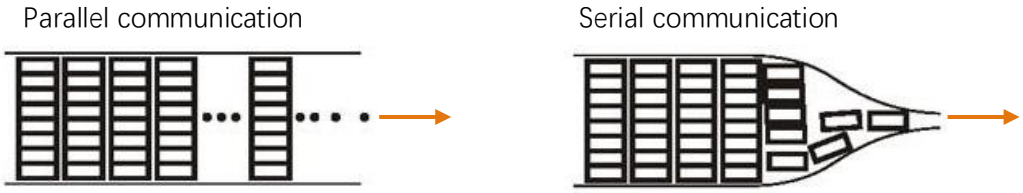
		Remainder	Sequence
2	150	0	0
2	75	1	1
2	37	1	2
2	18	0	3
2	9	1	4
2	4	0	5
2	2	0	6
2	1	1	7
	0		

The result is 10010110.

Circuit Knowledge

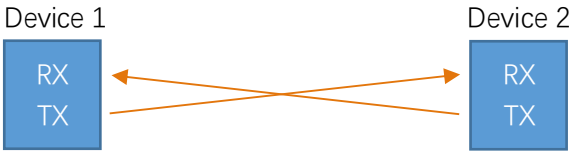
Serial and parallel communication

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



Serial communication

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

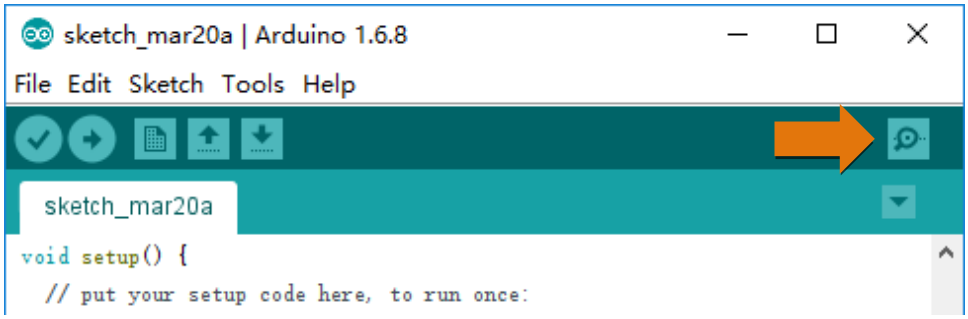


Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

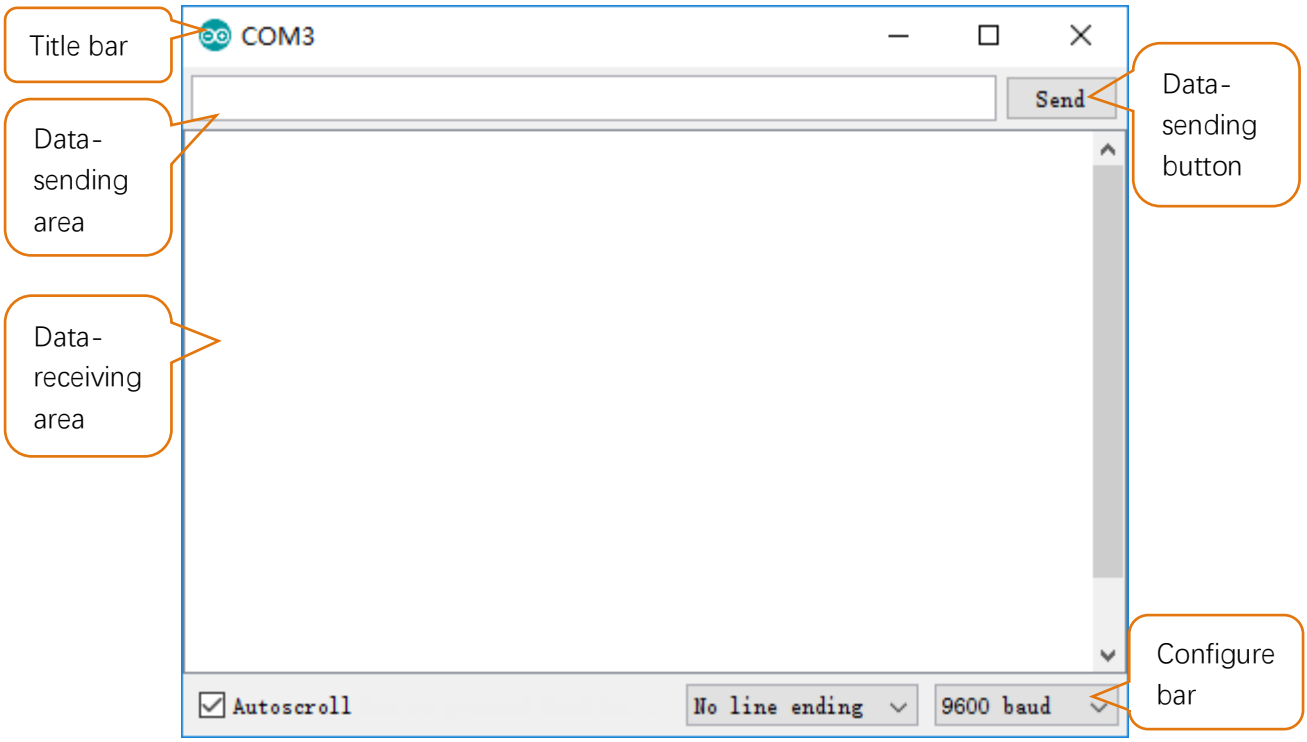
Serial port on Control board

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

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

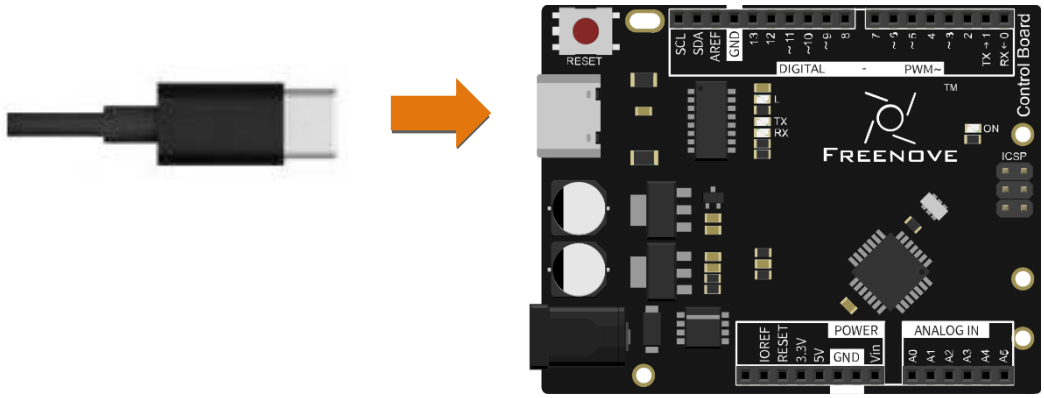


Interface of Serial Monitor window is as follows. If you can't open it, make sure control board had been connected to the computer, and choose the correct serial port in the menu bar "Tools".



Circuit

Connect control board to the computer with USB cable.



Sketch

Sketch 6.1.1

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

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

```
6
7   void loop() {
8     // print variable counter value to serial
9     Serial.print("counter:");     // print the string "counter:"
10    Serial.println(counter);     // print the variable counter value
11    delay(500);                  // wait 500ms to avoid cycling too fast
12    counter++;                   // variable counter increases 1
13  }
```

setup() function initializes the serial port.

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

Serial class

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

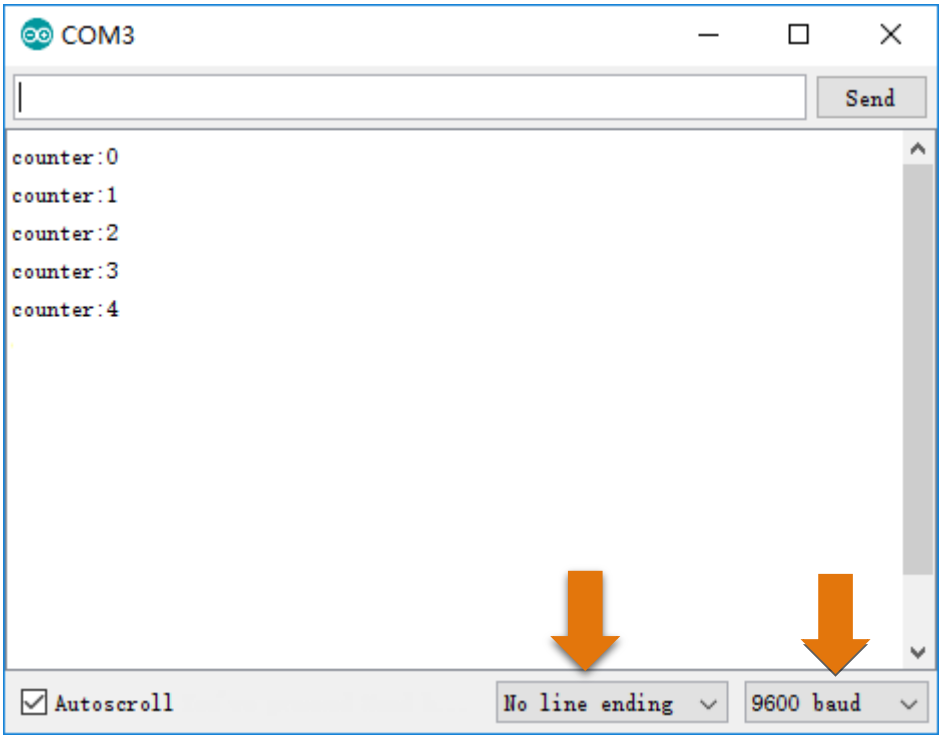
Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

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

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

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

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



Project 6.2 Receive Data through Serial Port

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

Component List

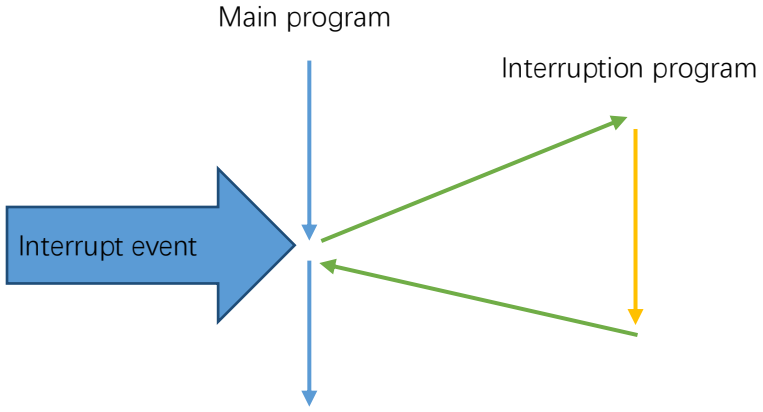
Same with the previous section.

Code Knowledge

Interrupt

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

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



Circuit

Same with the previous section.

Sketch

Sketch 6.2.1

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

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

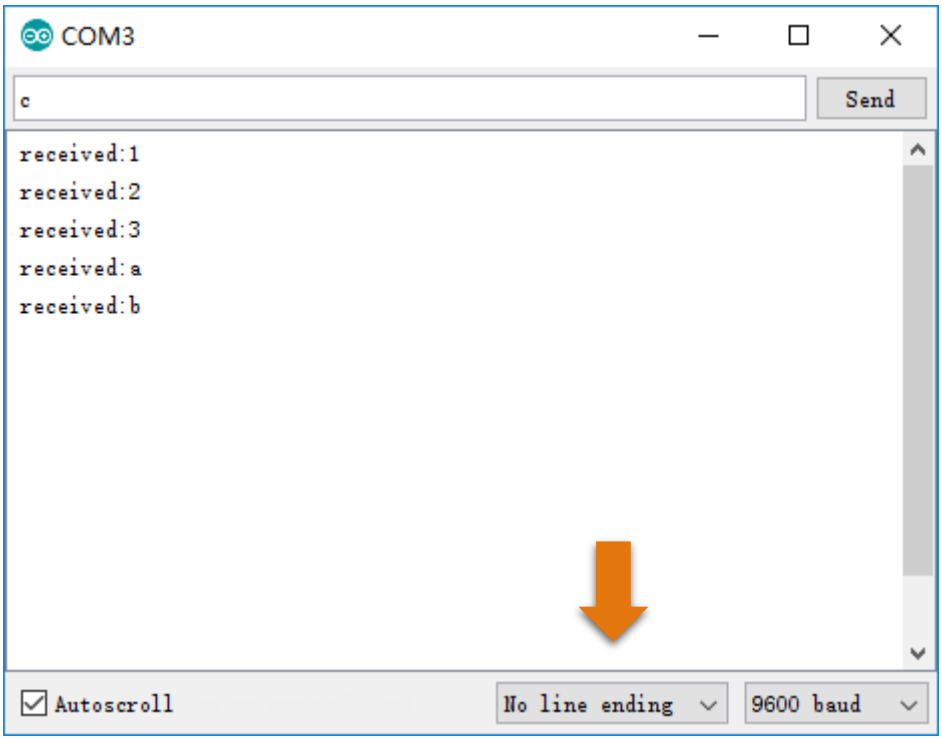
In the setup() function, we initialize the serial port. Then, the loop() function will continuously detect whether there are data to read. if so, it will read the character and send it back.

Serial Class

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

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

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



char type

char type variable can represent a character, but it cannot store characters directly. It stores numbers to replace characters. char type occupies 1-byte store area, and use a value 0-127 to correspond to 128 characters. The corresponding relation between number and character is ruled by ASCII table. For more details of ASCII table, please refer to the appendix of this book.
Example: Define char aChar = 'a', bChar = '0', then the decimal value of aChar is 97, bChar will be 48.

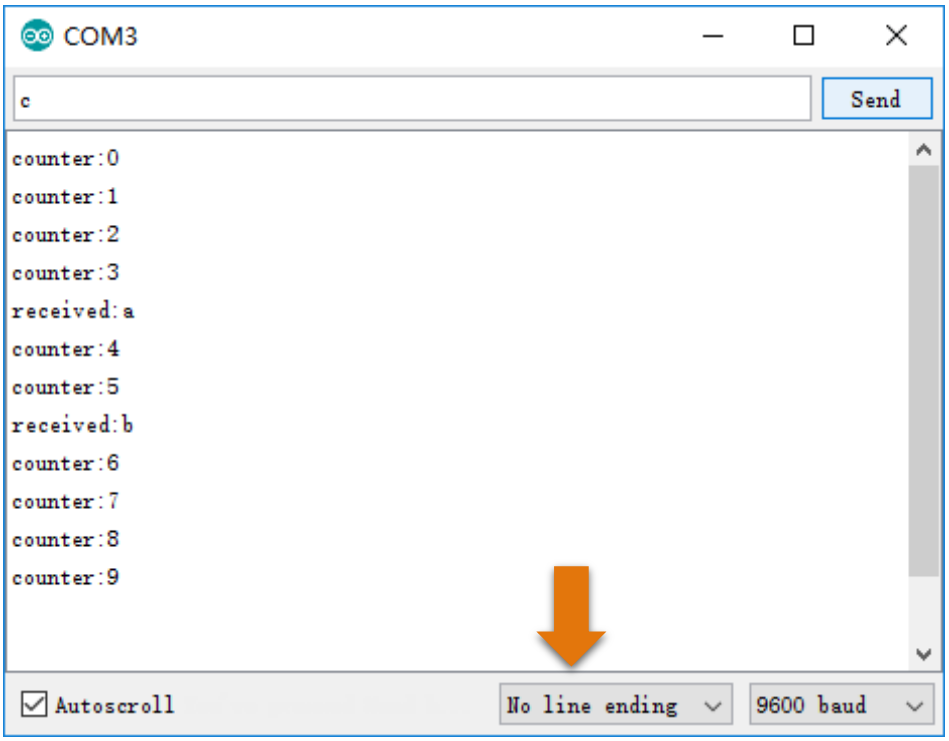
Sketch 6.2.2

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

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

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

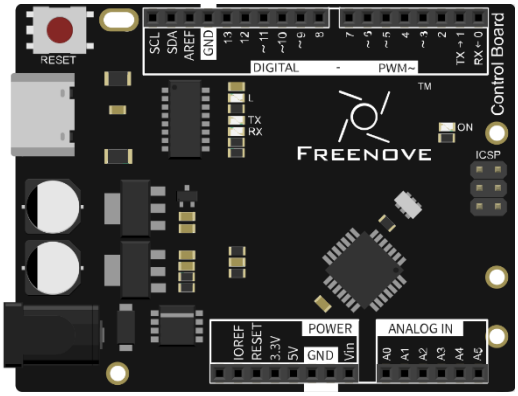
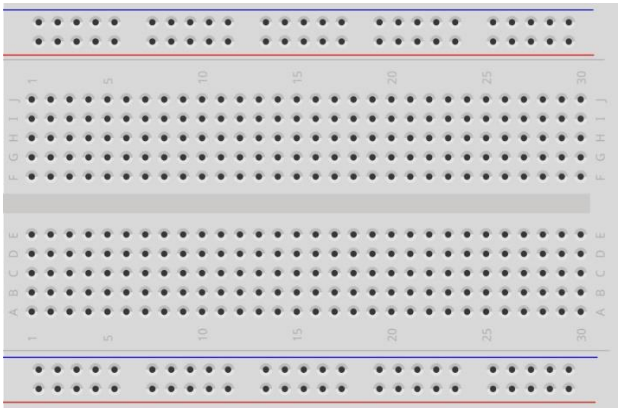
Verify and upload the code, open the Serial Monitor, then you'll see the number constantly sent from control board. Fill in characters in the sending area, and click the Send button, then you'll see the string returned from control board.



Project 6.3 Application of Serial

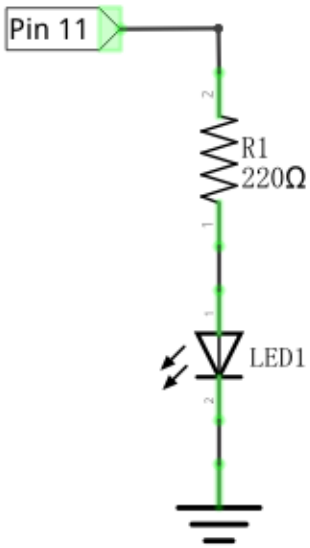
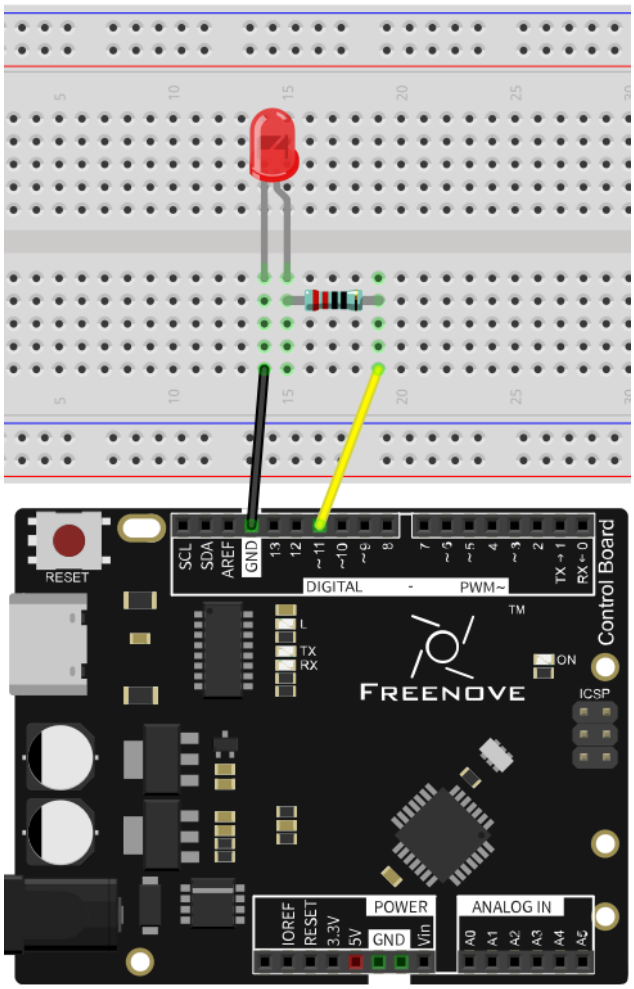
We will use the serial port on control board to control one LED.

Component List

<table>
<tr><td>Control board x1
  </td><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td></tr>
<tr><td>Jumper M/M x2
  </td><td colspan="2"></td></tr>
</table>

Circuit

Here we will use pin 11 of the control board 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, control board will convert it into PWM duty cycle of 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
}
```

```
9
10  void loop() {
11    if (Serial.available()) {          // judge whether the data has been received
12      inInt = Serial.parseInt();       // read an integer
13      Serial.print("received:");       // print the string " received:"
14      Serial.println(inInt);           // print the received character
15      // convert the received integer into PWM duty cycle of ledPin port
16      analogWrite(ledPin, constrain(inInt, 0, 255));
17    }
18  }
19
```

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

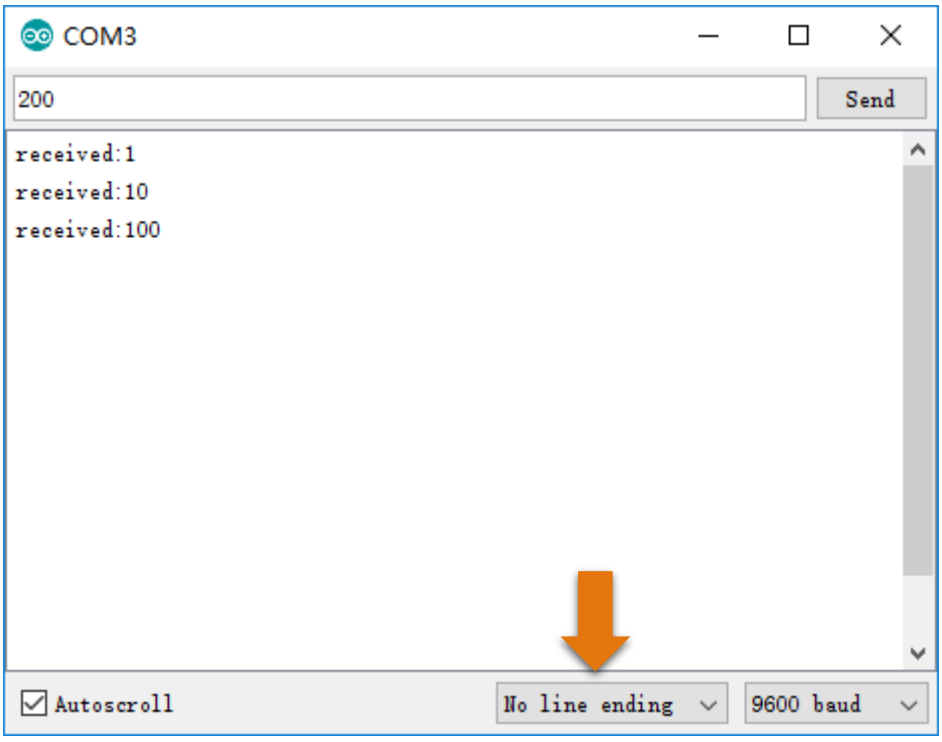
Serial Class

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

constrain(x, a, b)

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

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



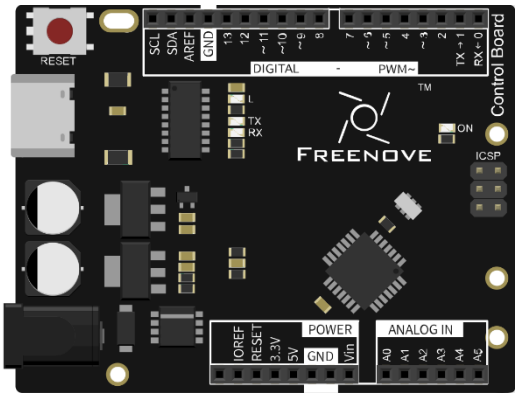
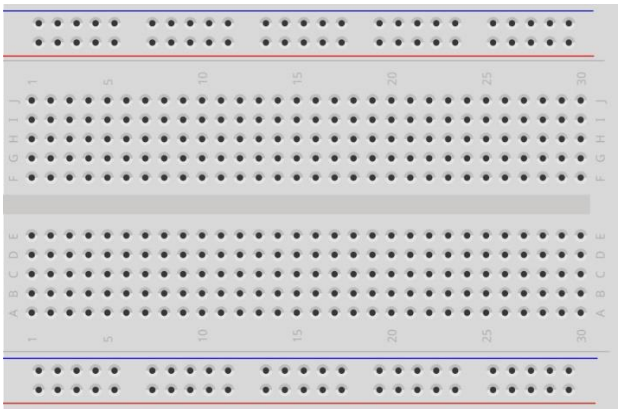
Chapter 7 ADC

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

Project 7.1 ADC

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

Component List

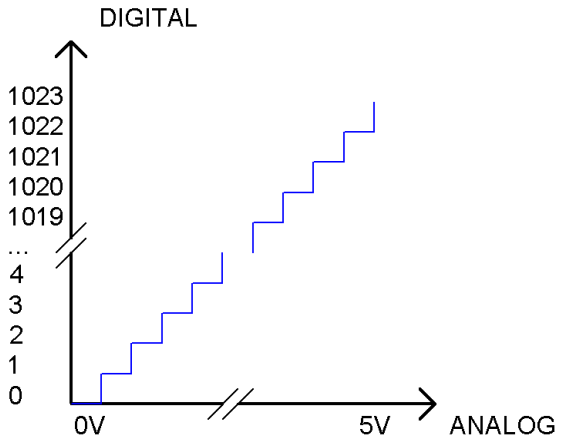
Control board x1 	Breadboard x1 
USB cable x1 	Rotary potentiometer x1 
Jumper M/M x3 	

Circuit Knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC module is 10 bits, that means the resolution is 2^10=1024, so that its range (at 5V) will be divided equally into 1024 parts.

Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in rang of 0V-5/1024V corresponds to digital 0;

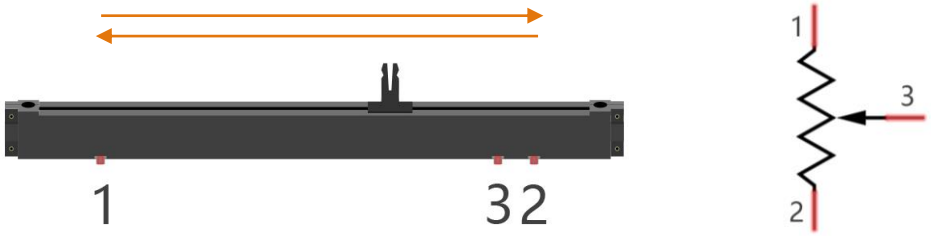
Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;

The following analog signal will be divided accordingly.

Component Knowledge

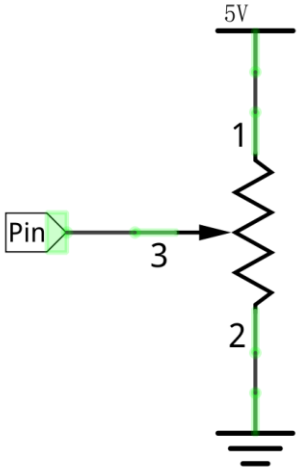
Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



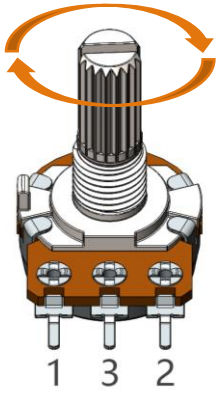
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



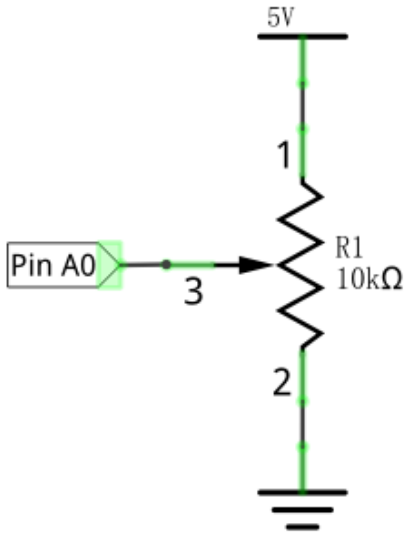
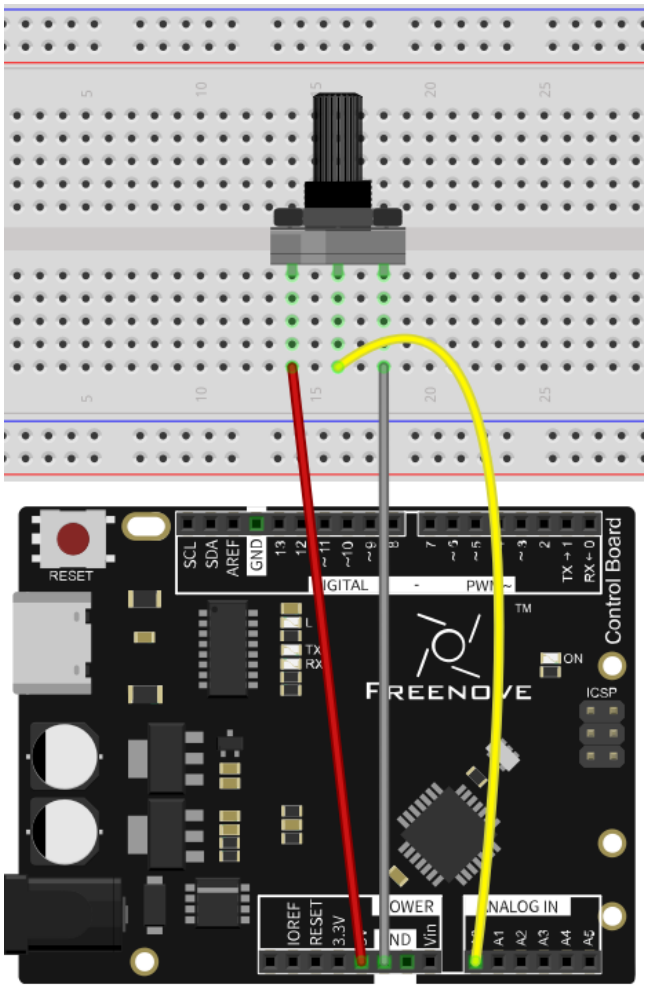
Rotary potentiometer

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



Circuit

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 7.1.1

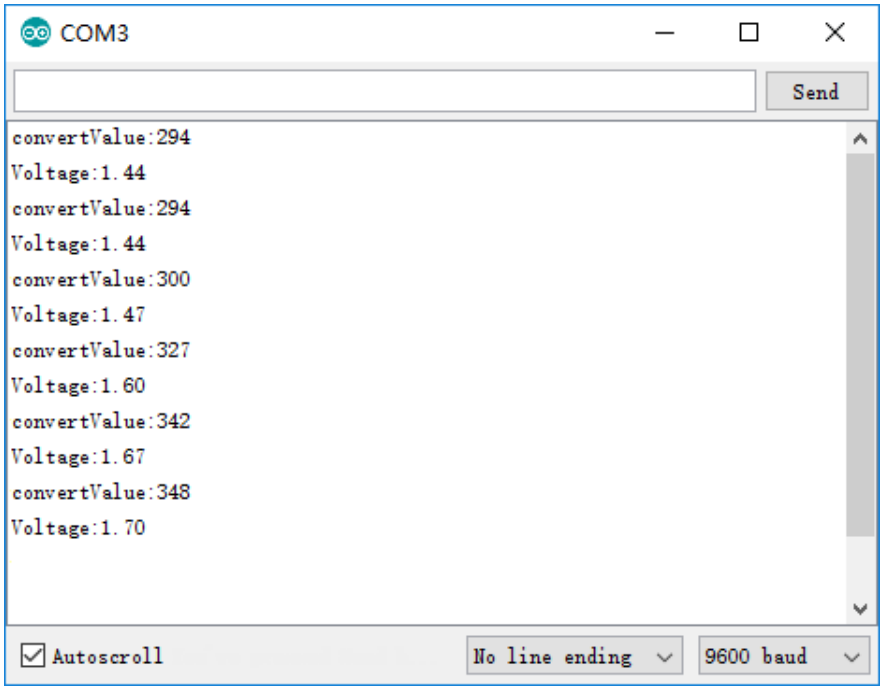
Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino IDE through serial port.

```
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
}

void loop() {
  adcValue = analogRead(A0);          // Convert analog of pin A0 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 pin A0, then convert it into voltage and sent to the serial port.

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

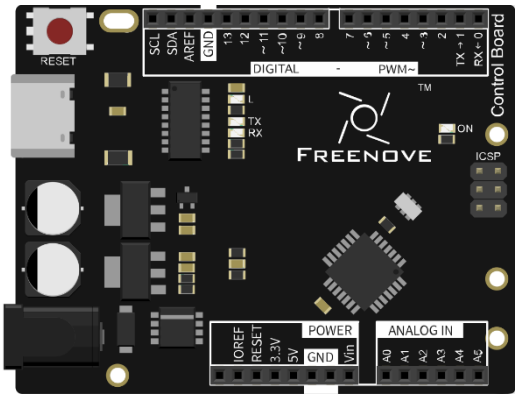
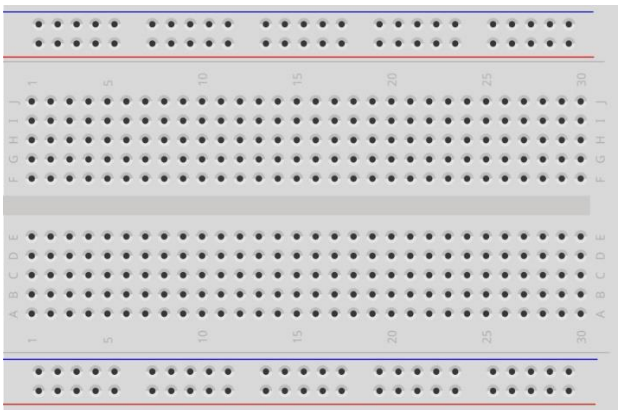
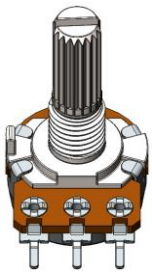
Turn the rotary potentiometer shaft, and you can see the voltage change.



Project 7.2 Control LED by Potentiometer

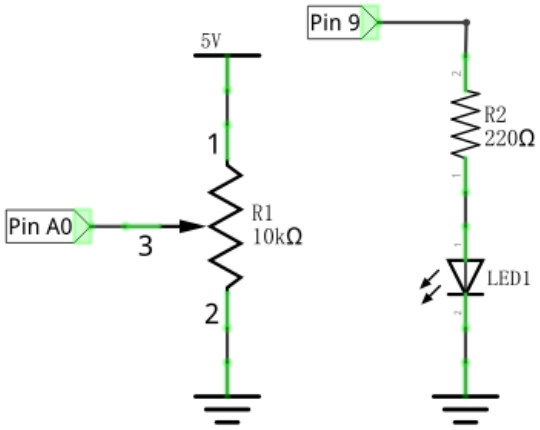
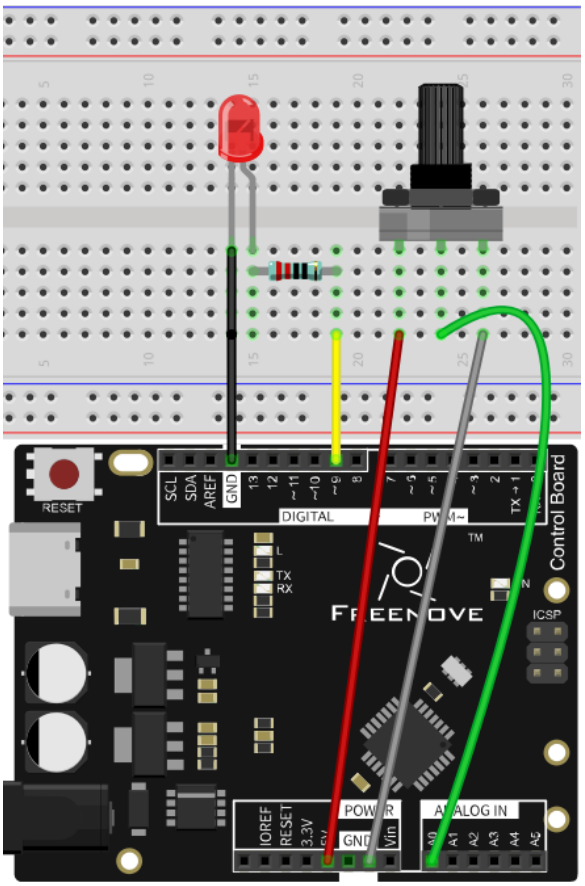
In the previous section, we have finished reading ADC value and converting it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

Component List

<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>Rotary potentiometer x1
  </td><td>LED x1
  </td><td>Resistor 220Ω x1
  </td></tr>
<tr><td>Jumper M/M x5
  </td><td colspan="3"></td></tr>
</table>

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

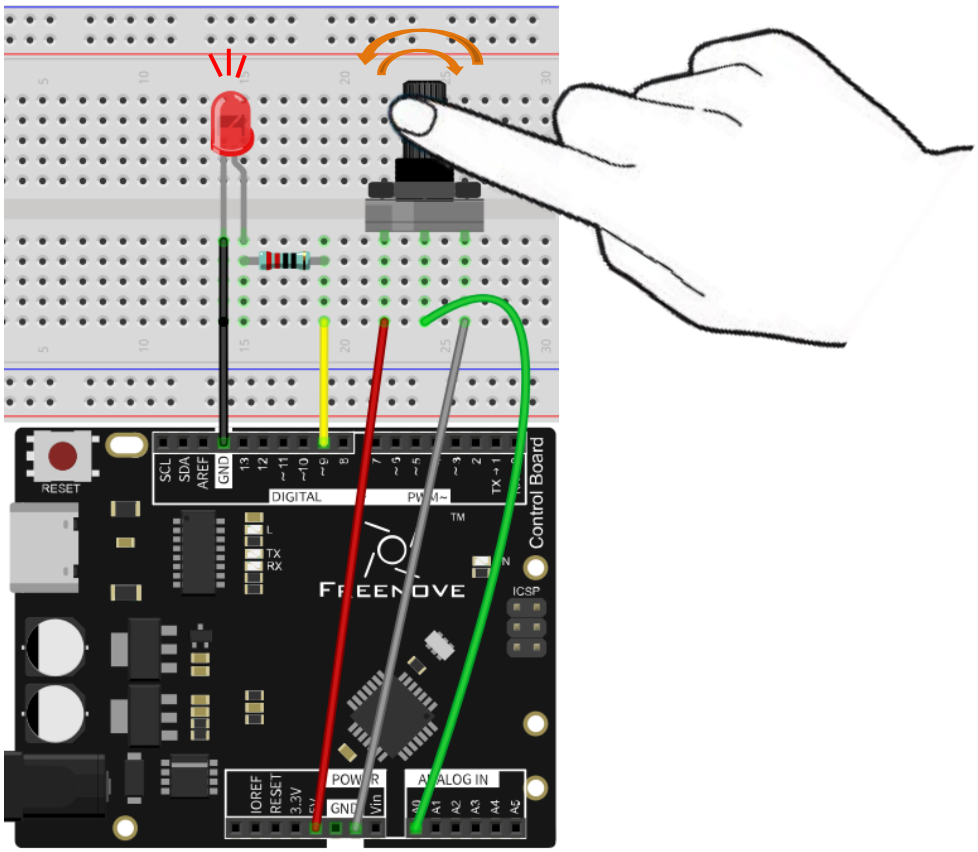
Sketch 7.2.1

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 pin 9 on control board 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 pin A0 and map it to PWM duty cycle of LED pin port. According to different LED brightness, we can see the changes of voltage easily.

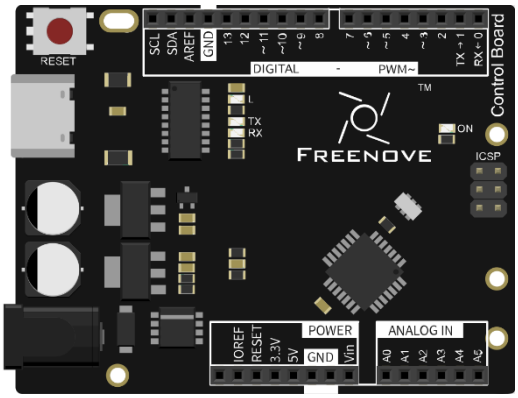
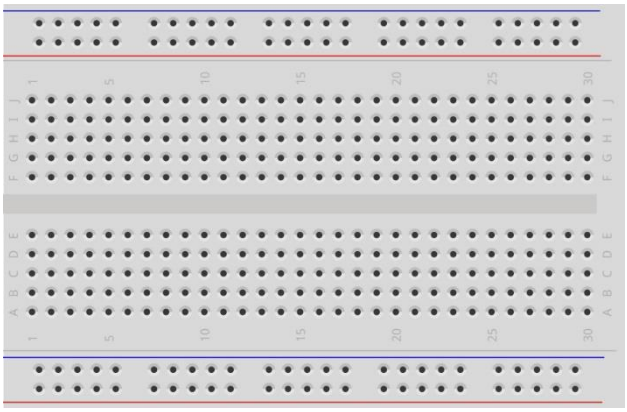
Verify and upload the code, turn the rotary potentiometer shaft, you will see the LED brightness change.



Project 7.3 Control LED by Potentiometer

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

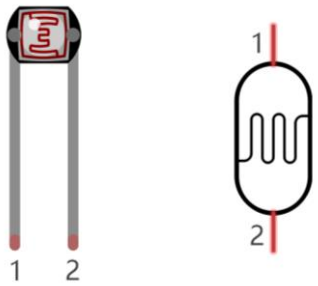
Component List

<table>
<tr><td colspan="2">Control board x1
  </td><td colspan="4">Breadboard x1
  </td></tr>
<tr><td colspan="2">USB cable x1
  </td><td>Photoresistor x1
  </td><td>LED x1
  </td><td>Resistor 10kΩ x1
  </td><td>Resistor 220Ω x1
  </td></tr>
<tr><td colspan="2">Jumper M/M x5
  </td><td colspan="4"></td></tr>
</table>

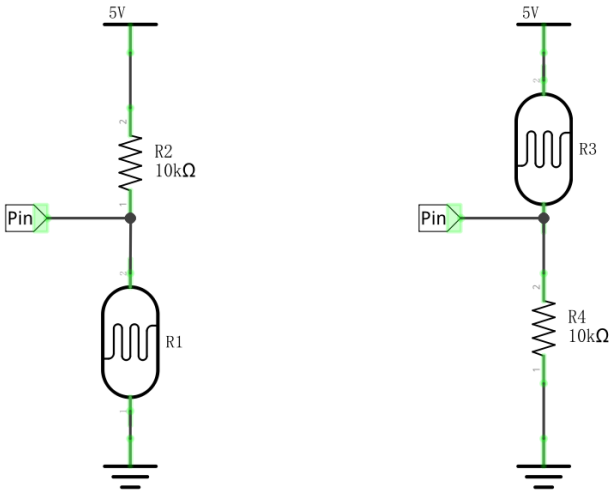
Component Knowledge

Photoresistor

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



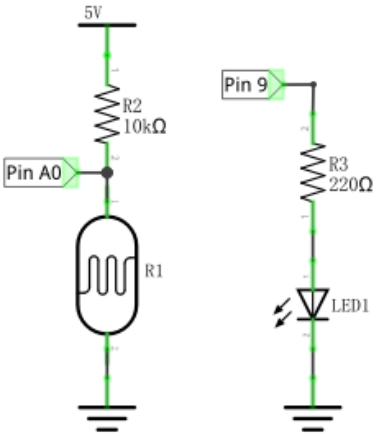
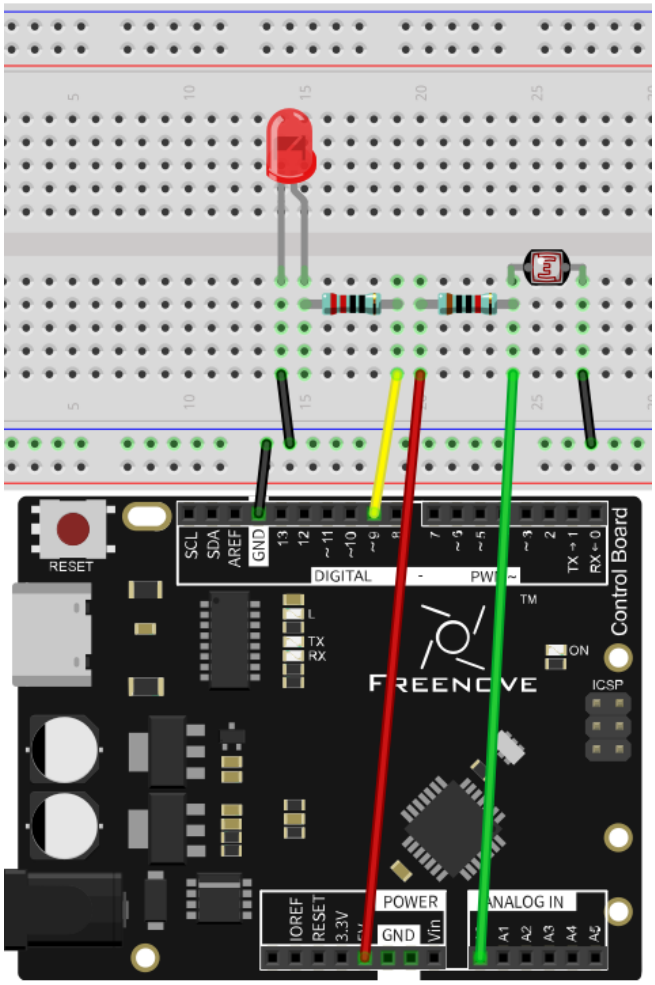
The circuit below is often used to detect the change of photoresistor's resistance value:



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

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

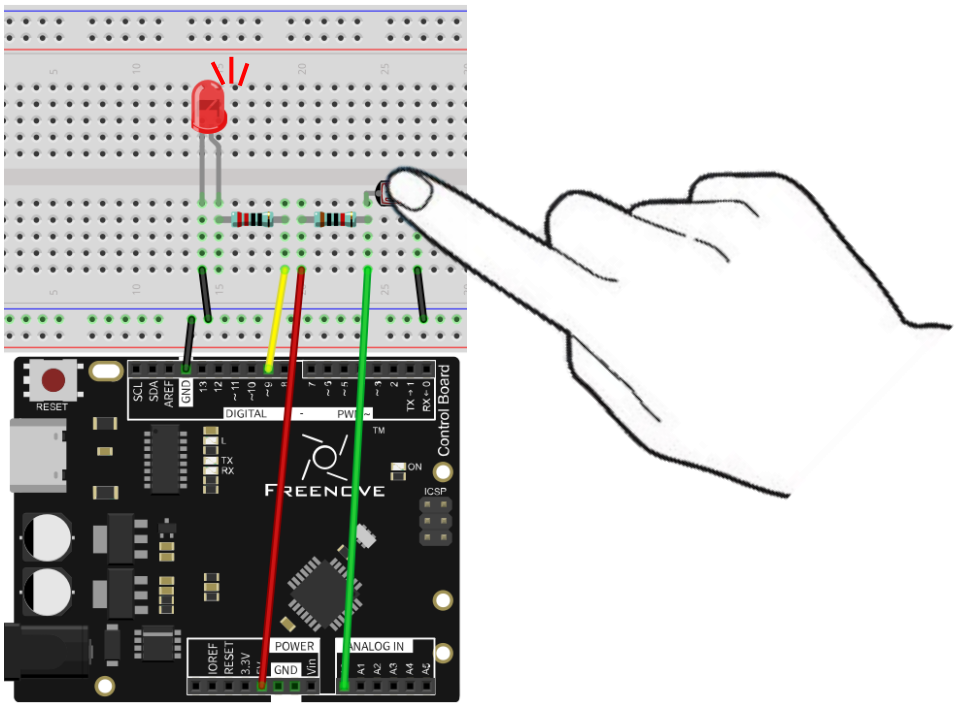
Sketch 7.3.1

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

```
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 pin A0, 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 pin A0, map it to PWM duty cycle of LED pin port. According to the brightness of LED, we can see the changes of voltage easily.

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



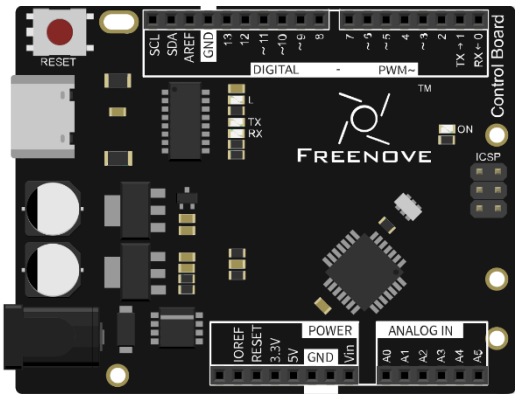
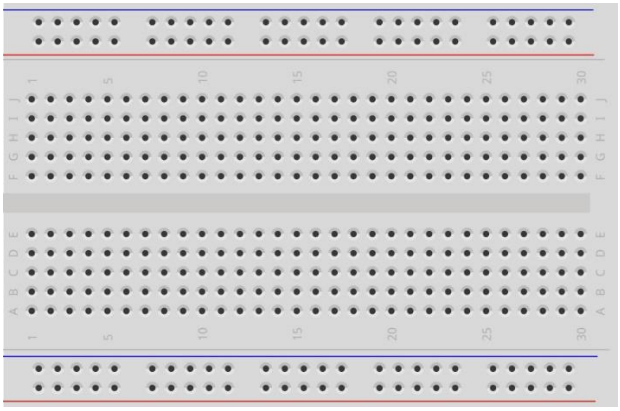
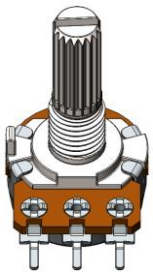
Chapter 8 RGB LED

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

Project 8.1 Control RGB LED through Potentiometer

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

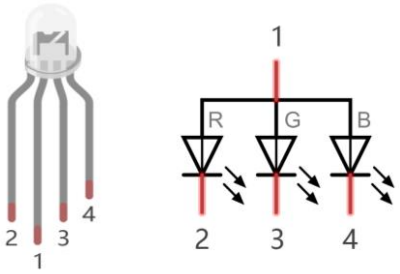
Component List

<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>Rotary potentiometer x3
  </td><td>RGB LED x1
  </td><td>Resistor 220Ω x3
  </td></tr>
<tr><td>Jumper M/M x15
  </td><td colspan="3"></td></tr>
</table>

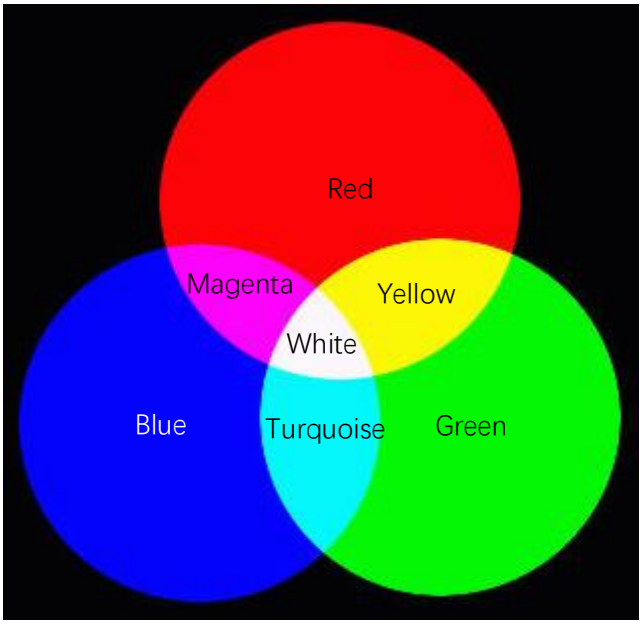
Component Knowledge

RGB LED

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



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

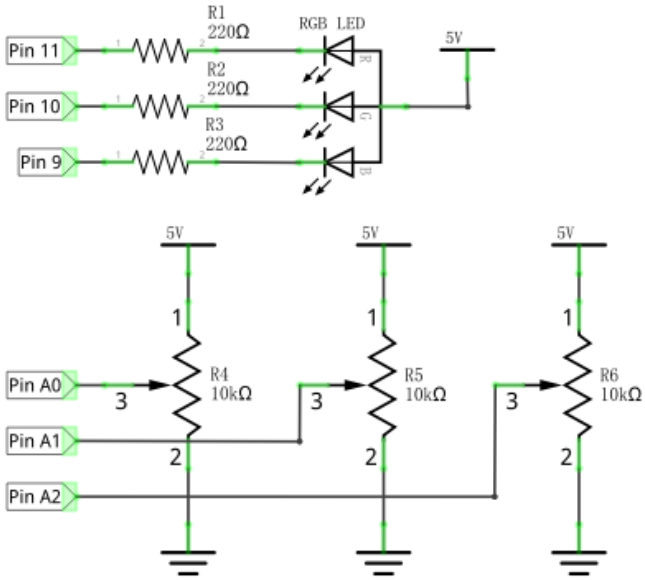


We know from the previous section that, control board controls LED to emit a total of 256(0-255) different brightness via PWM. So, through different combinations of RGB light brightness, we can create 256^3=16777216(16Million) colors.

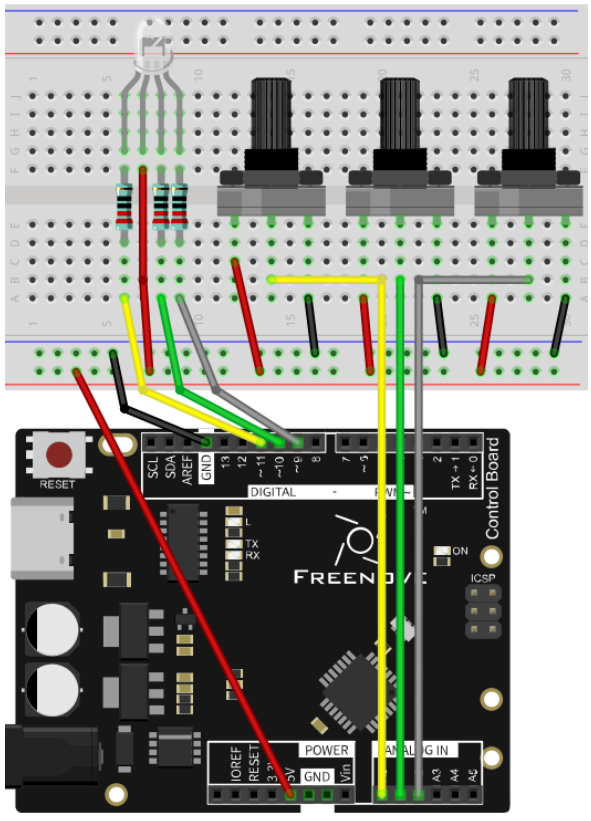
Circuit

Use pin A0, A1, A2 ports of the control board to detect the voltage of rotary potentiometer, and control RGB LED by pin 9, 10, 11.

Schematic diagram



Hardware connection



Sketch

Sketch 8.1.1

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

```
// 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));
}
```

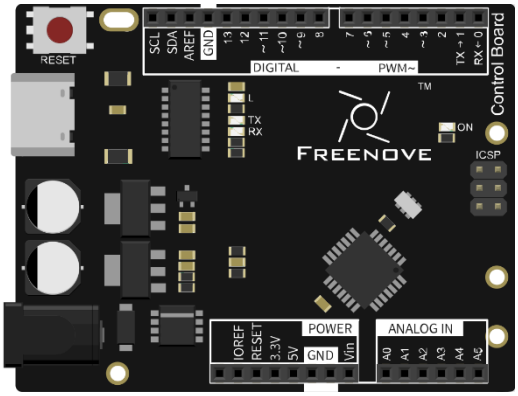
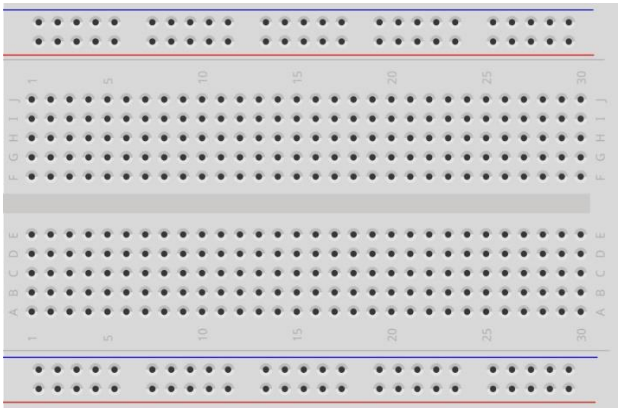
In the code, we get the voltages of three rotary potentiometers, and convert them into PWM duty cycle to control the three LEDs of the RGB LED to emit light with different brightness.

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

Project 8.2 Multicolored LED

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

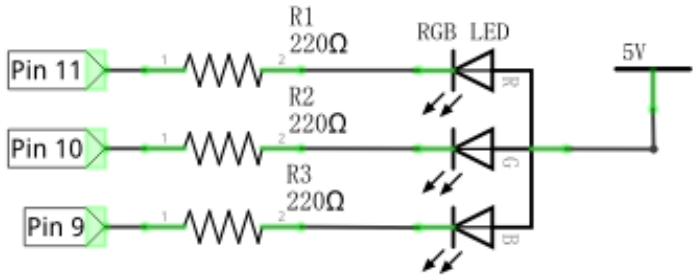
Component List

Control board x1 	Breadboard x1 	
USB cable x1 	RGB LED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

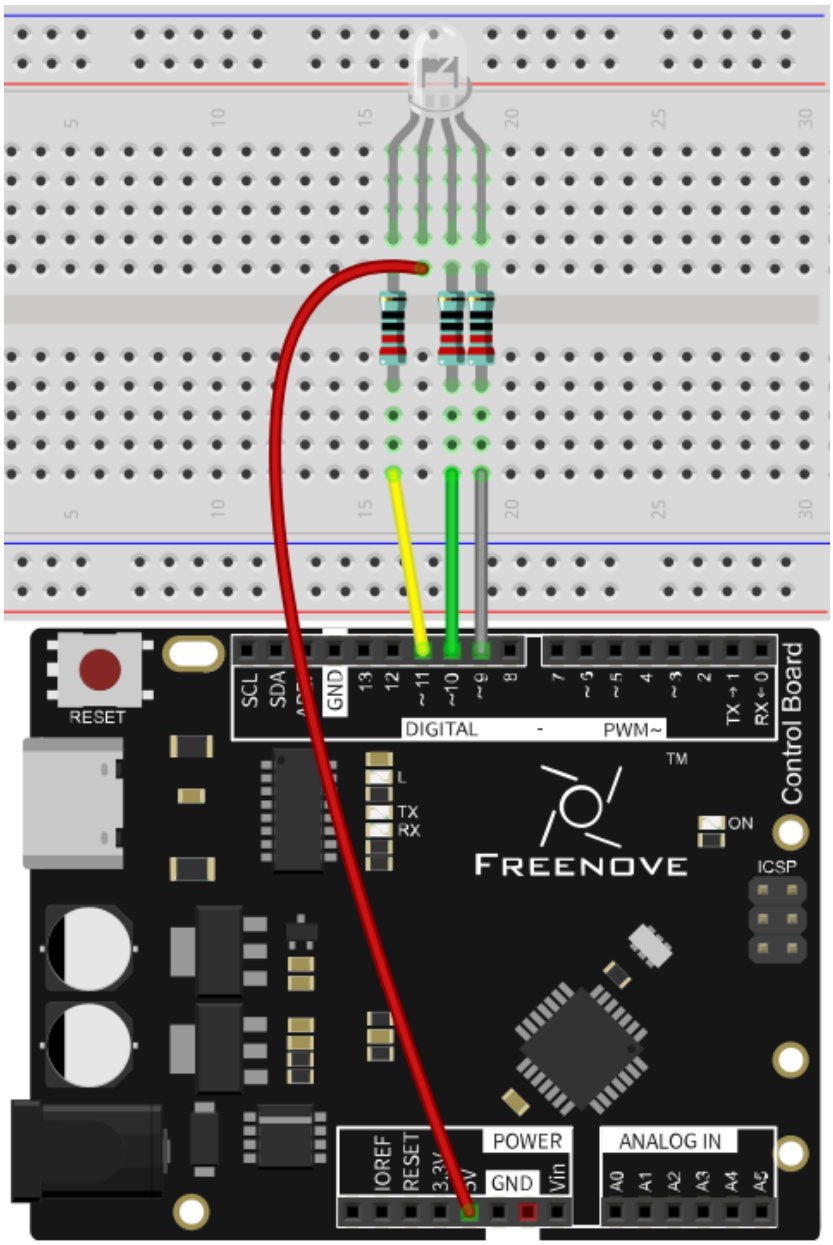
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 8.2.1

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

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

random(min, max)

random (min, max) function is used to generate random number, and it will return a random value in the range (min, Max-1).
You can also use random (max) function, the function set the minimum value into 0 by default, and returns a random value in the range (0, Max-1).

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

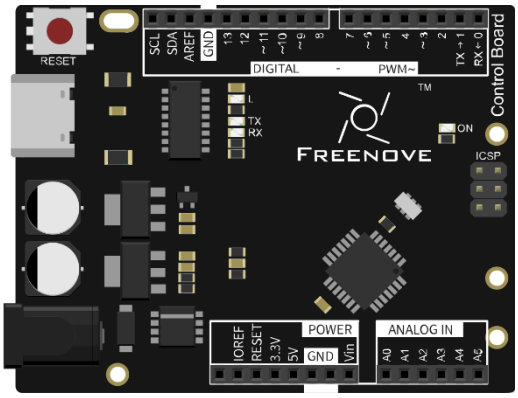
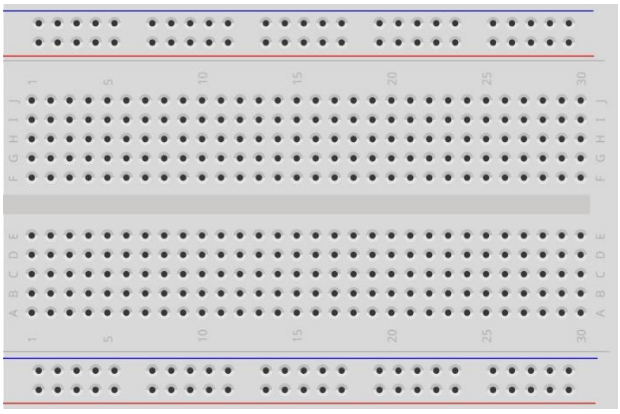
Chapter 9 Buzzer

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

Project 9.1 Active Buzzer

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

Component List

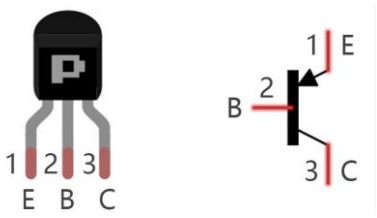
<table>
<tr><td colspan="2">Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td colspan="2">USB cable x1
  </td><td colspan="3">Jumper M/M x6
</td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

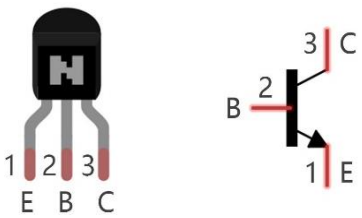
Transistor

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

PNP transistor



NPN transistor



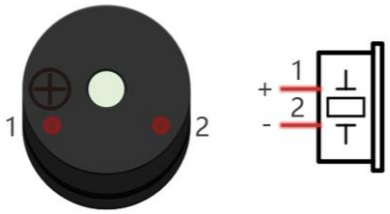
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

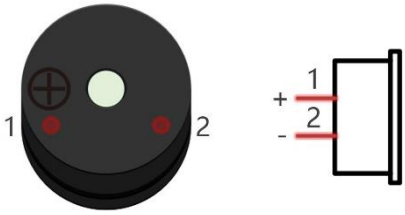
Buzzer

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

Active buzzer



Passive buzzer



(A white label is attached to the active buzzer)

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

How to identify active and passive buzzer?

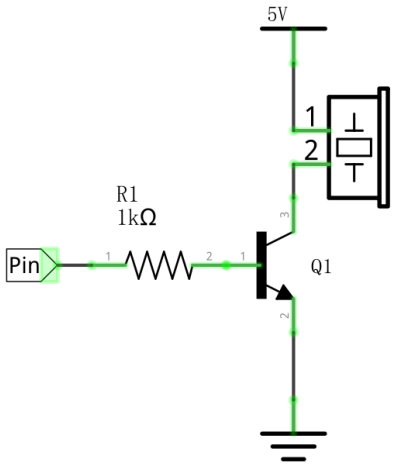
1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this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

Active buzzer bottom

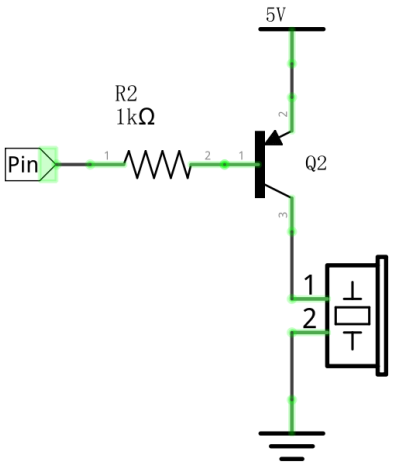
Passive buzzer bottom

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

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

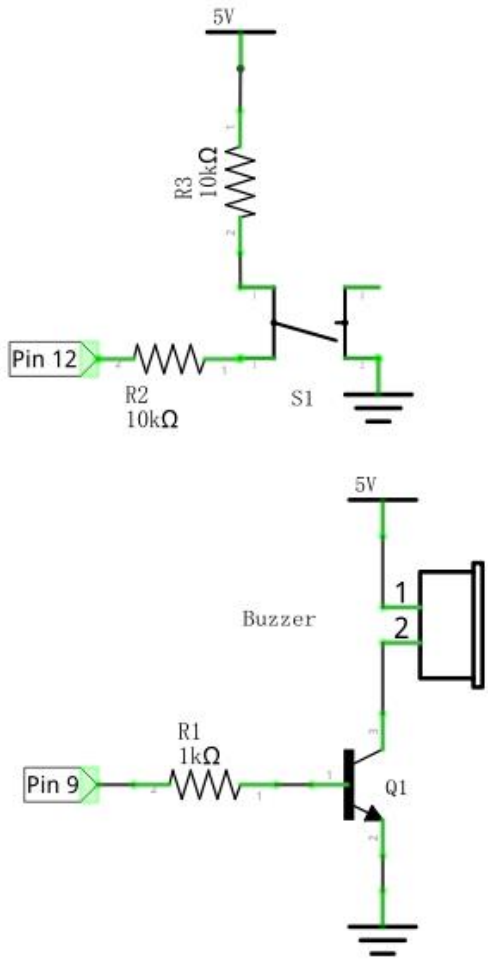
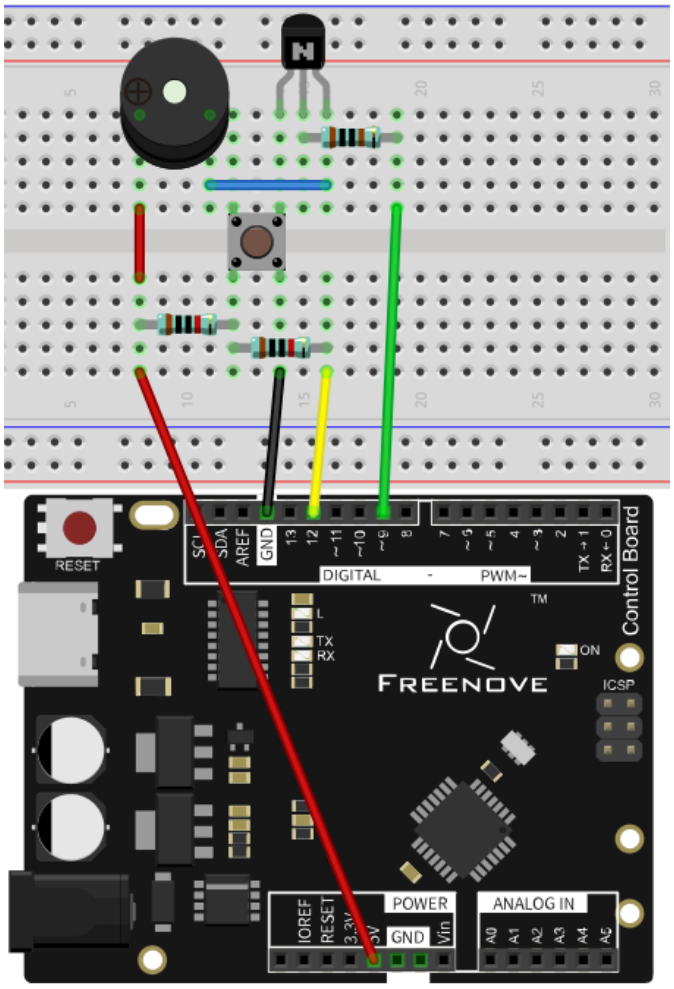


When we use a PNP transistor to drive a buzzer, we often use the following method. If control board pin outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If control board pin outputs high level, no current flows through R1, the transistor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



Circuit

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

Schematic diagram	Hardware connections
	

Sketch

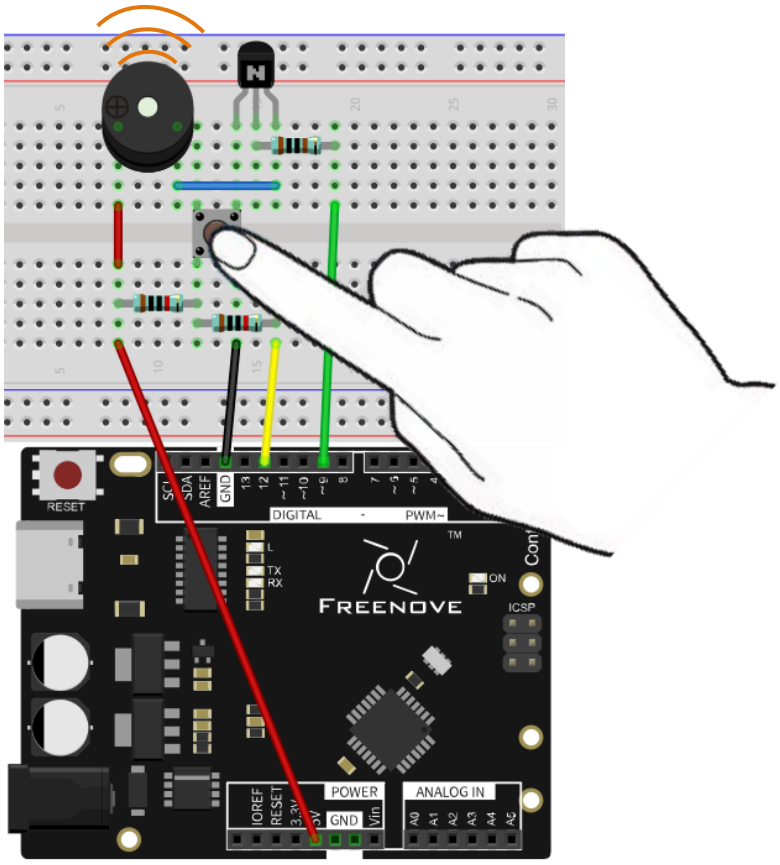
Sketch 9.1.1

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 switch. When it is pressed, the output high level controls transistor to get conducted, and drives active buzzer to make a sound.

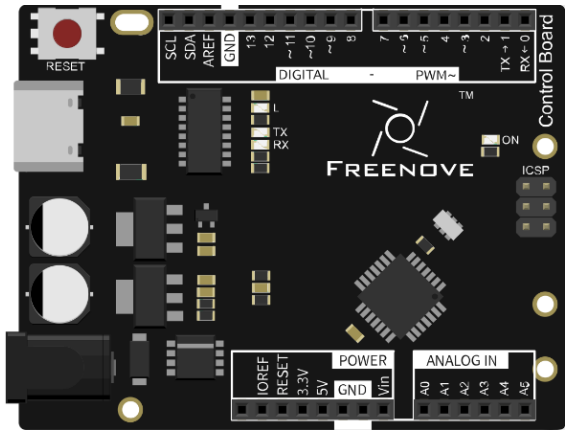
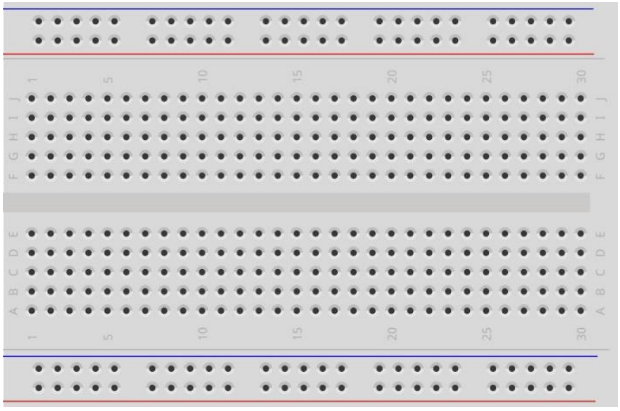
Verify and upload the code, press the push button, and the active buzzer will make a sound.



Project 9.2 Passive Buzzer

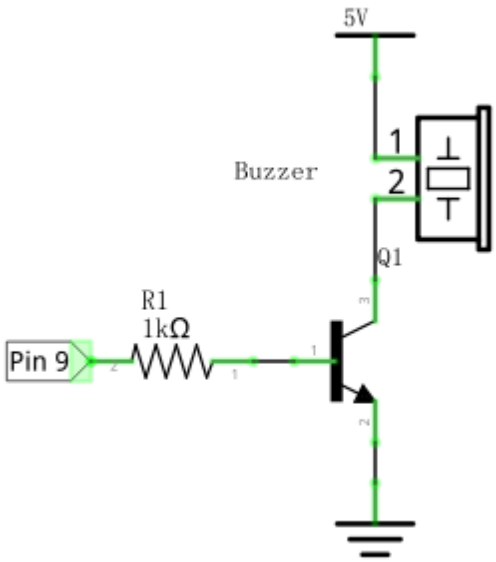
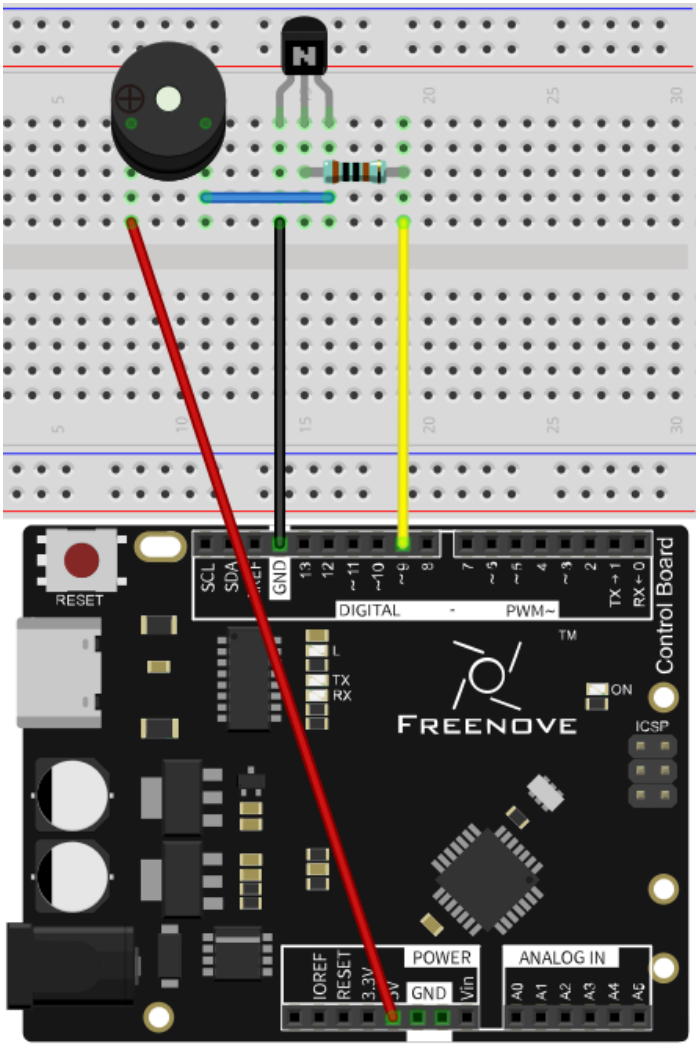
In the previous section, we have finished using transistor to drive an active buzzer to beep. Now, we will try to use a passive buzzer to make a sound with different frequency.

Component List

Control board x1 	Breadboard x1 		
USB cable x1 	NPN transistor x1 	Passive buzzer x1 	Resistor 1kΩ x1 
Jumper M/M x4 			

Circuit

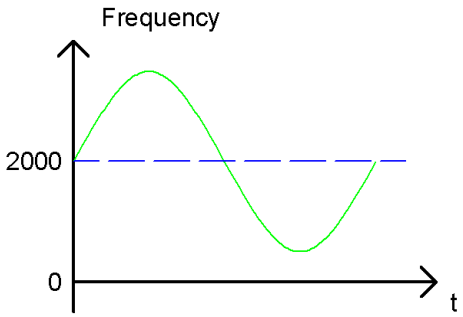
Use pin 9 port of control board to drive a passive buzzer.

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 the passive buzzer conforms roughly to the following sine curve over time:



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

```
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 of 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 need convert unit of π from angle to radian first .

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

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

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

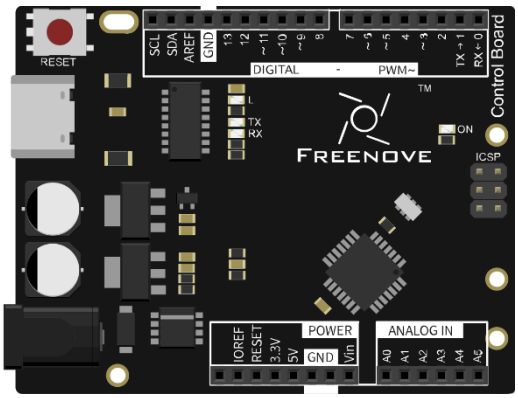
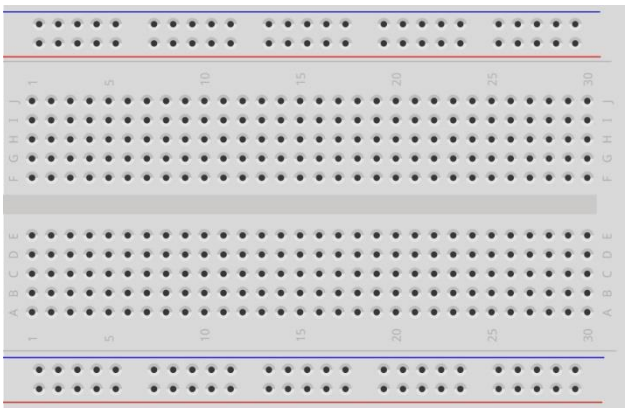
Chapter 10 Temperature Sensor

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

Project 10.1 Detect the Temperature

We will use a thermistor to detect the ambient temperature.

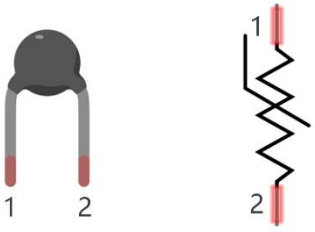
Component List

Control board x1 	Breadboard x1 	
USB cable x1 	Thermistor x1 	Resistor 10kΩ x1 
Jumper M/M x3 		

Component Knowledge

Thermistor

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



The relationship between resistance value and temperature of thermistor is:

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;

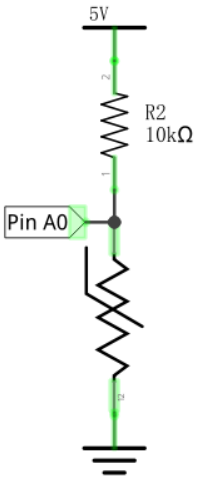
EXP[n] is nth power of e;

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.

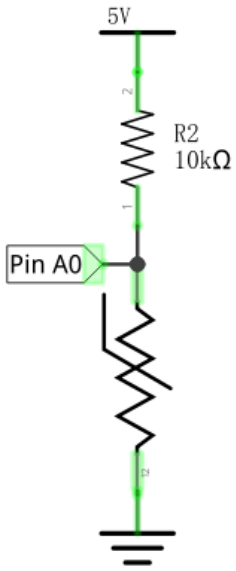
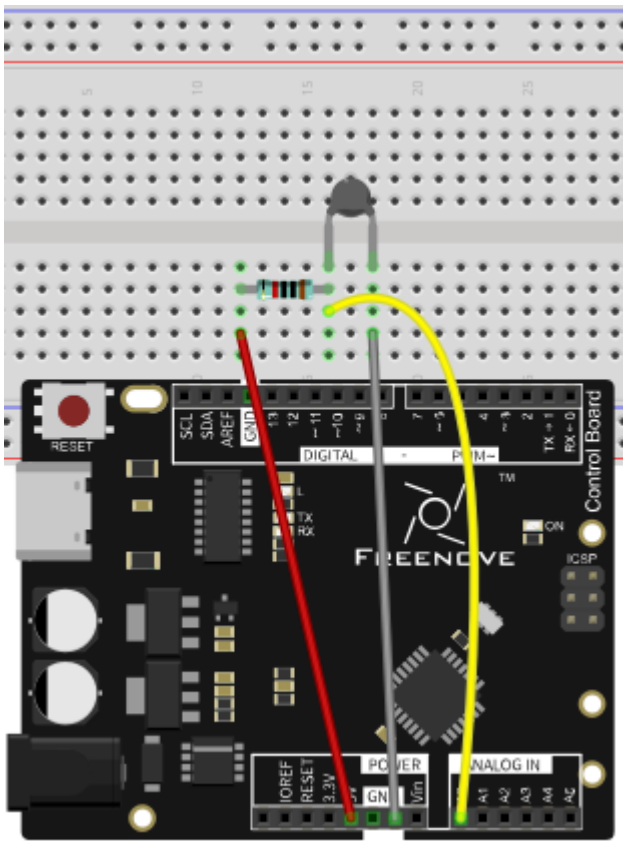
The circuit connection method of the thermistor is similar to photoresistor, as the following:



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

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 12.1.1

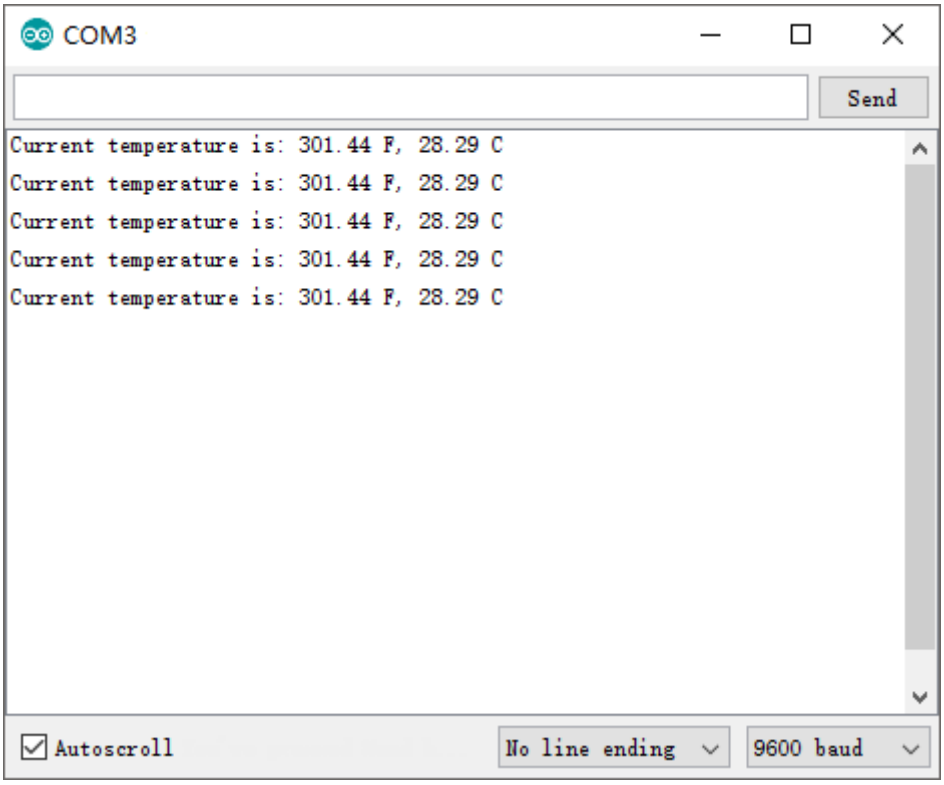
Now, write the code to detect the voltage value of thermistor, calculate the temperature value, and send it to Serial Monitor.

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

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

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

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



What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this tutorial. If you find errors, omissions or you have suggestions and/or questions about this tutorial or component contents of this kit, please feel free to contact us:

support@freenove.com

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in Processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, micro:bit, robots, smart cars and other interesting products, please visit our website:

http://www.freenove.com/

We will continue to launch fun, cost-effective, innovative and exciting products.

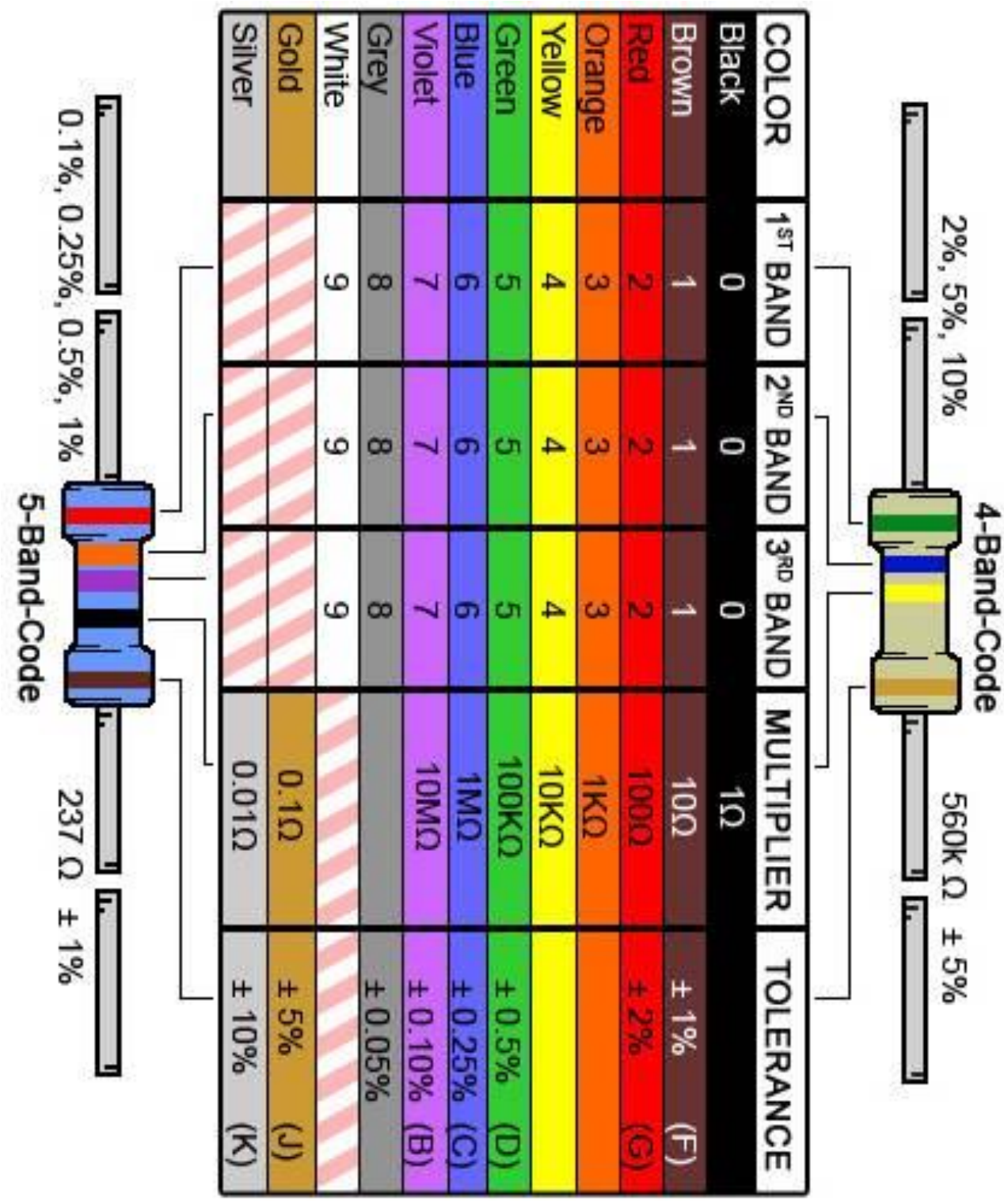
Thank you again for choosing Freenove products.

Appendix

ASCII Table

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