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.

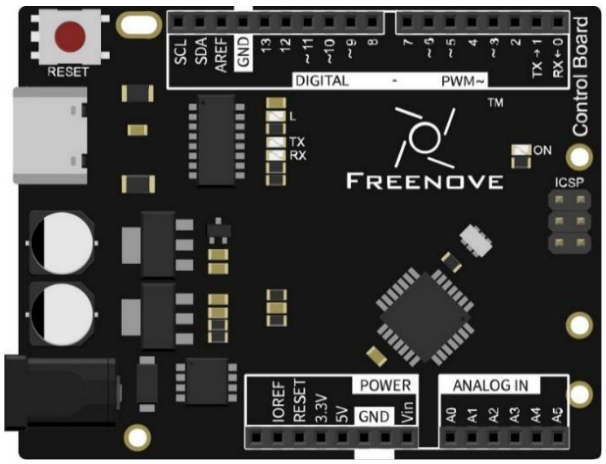
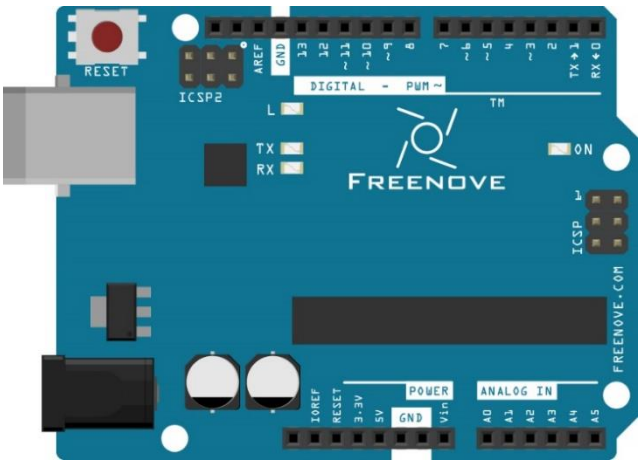
Support email: support@freenove.com

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.

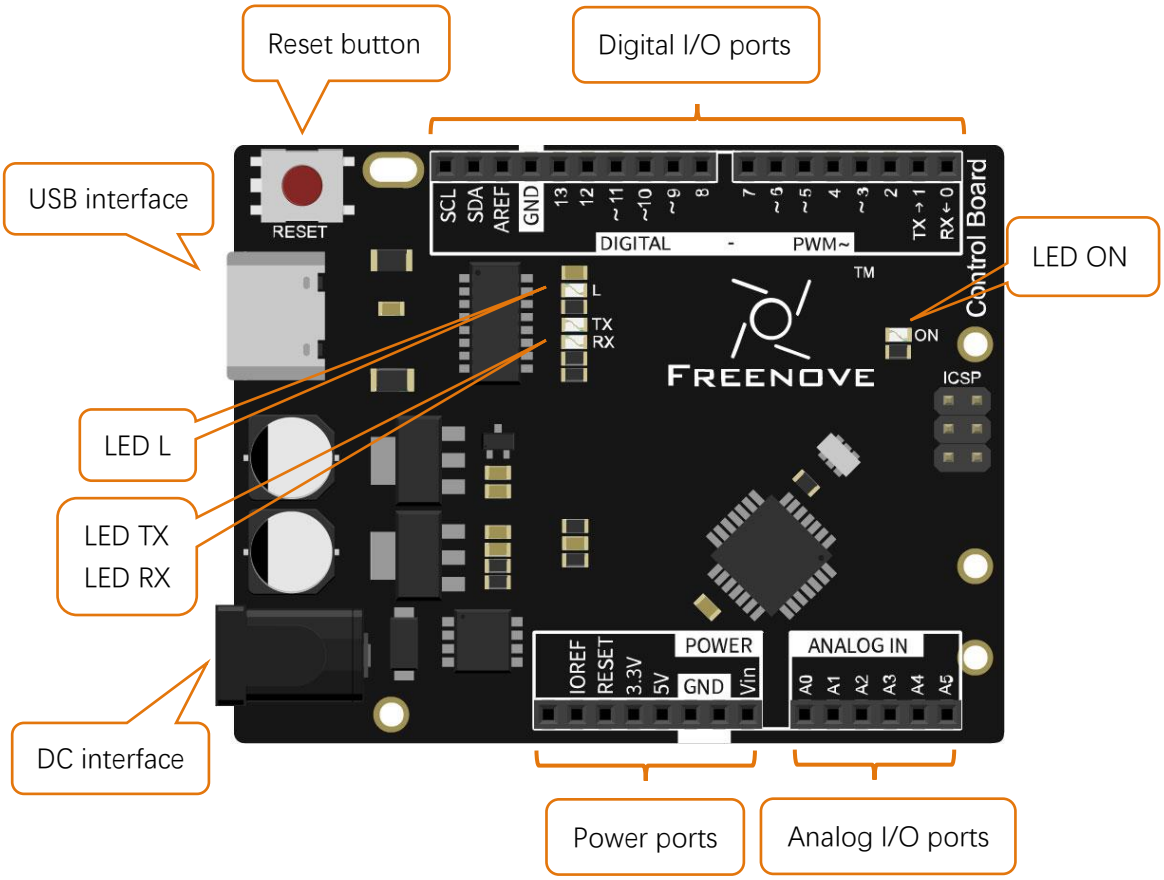
There are multiple versions of Freenove control board. Your purchase may be one of the following:

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:

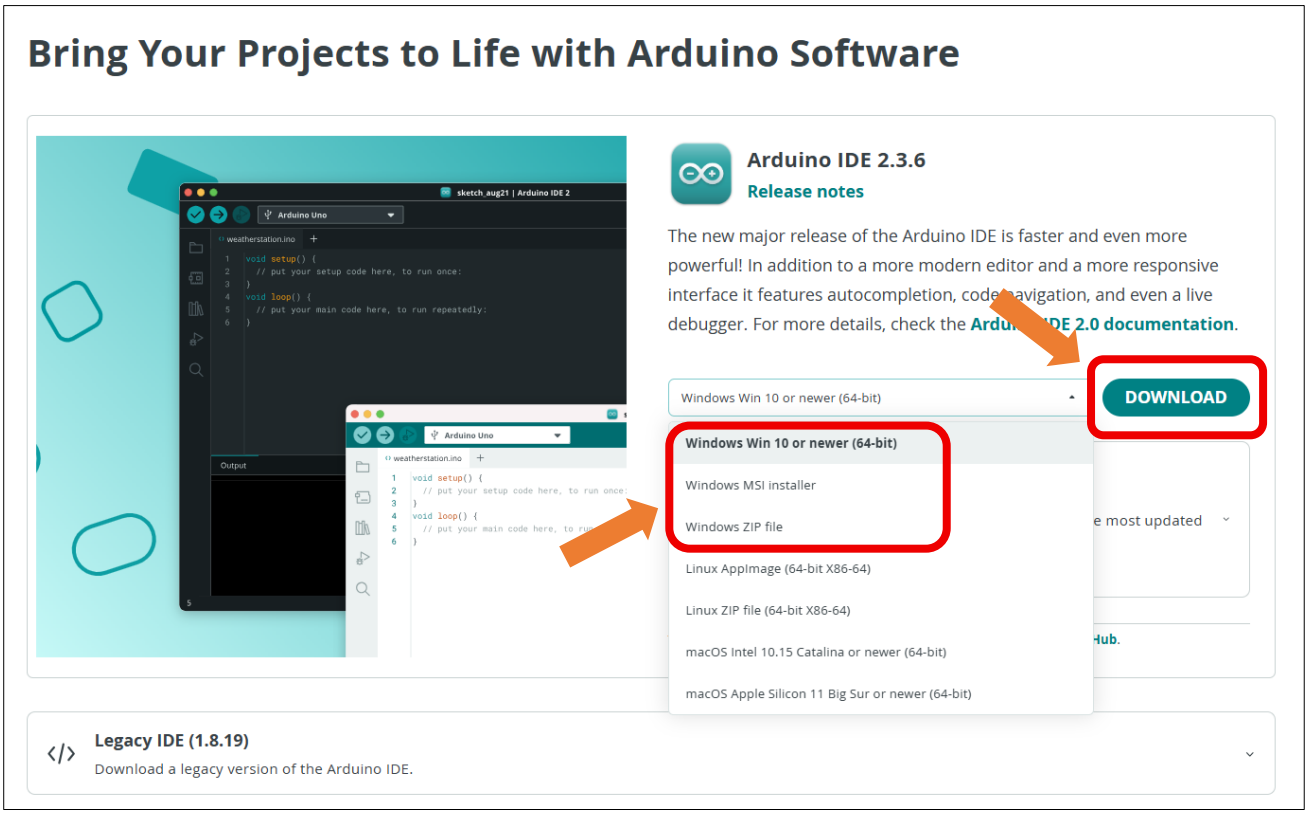


- Digital I/O ports is used to connect to other components or modules, to receive an input signal, or to send a control signal. Usually, we name it by adding a "D" in front of the number, such as D13 (pin 13).
- USB interface is used to provide power, upload code or communicate with PC.
- LED L is connected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

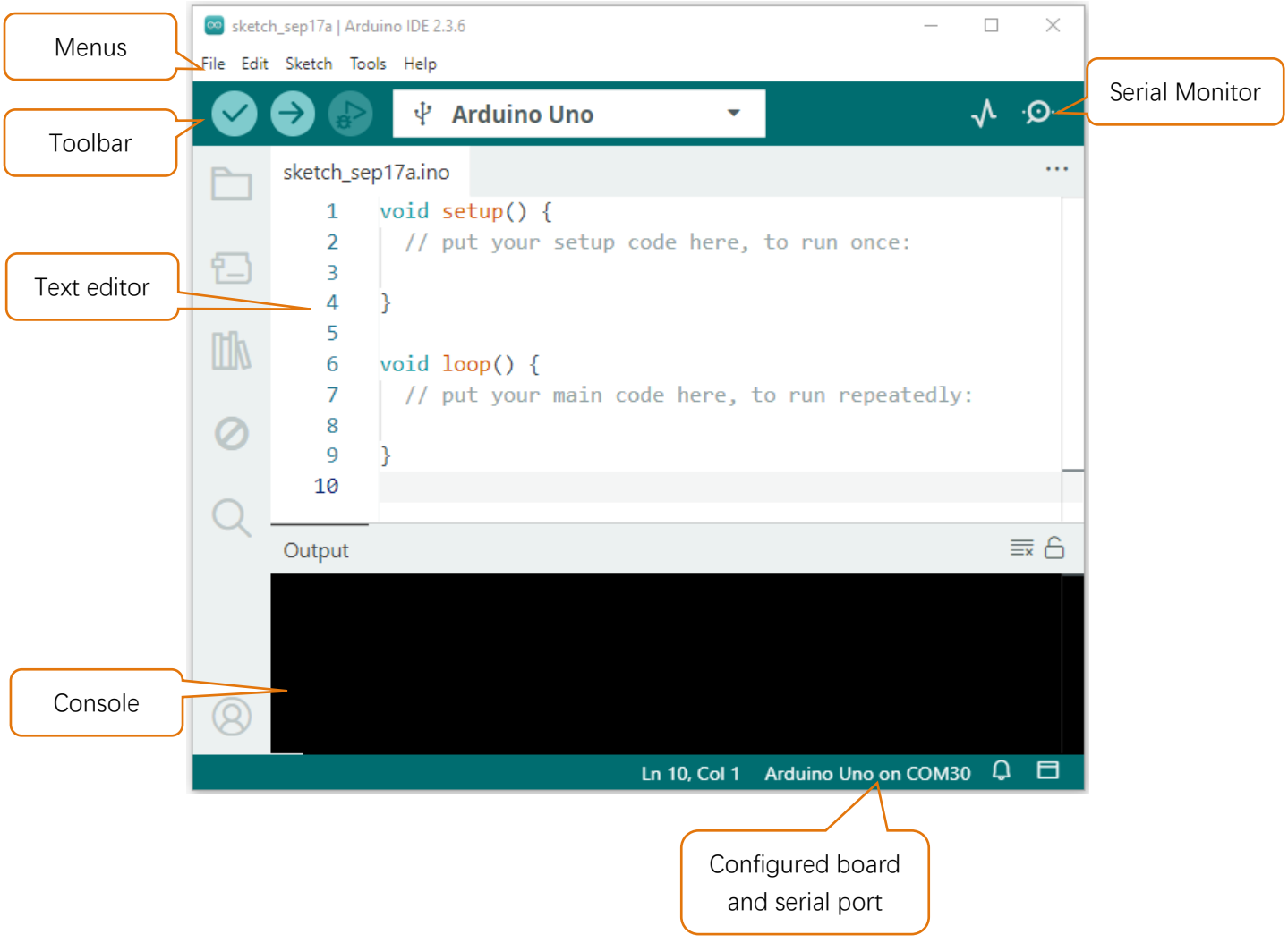
Arduino Software (IDE) is used to write and upload the code for Arduino Board.
First, install Arduino Software (IDE): visit https://www.arduino.cc/en/software/

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



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:

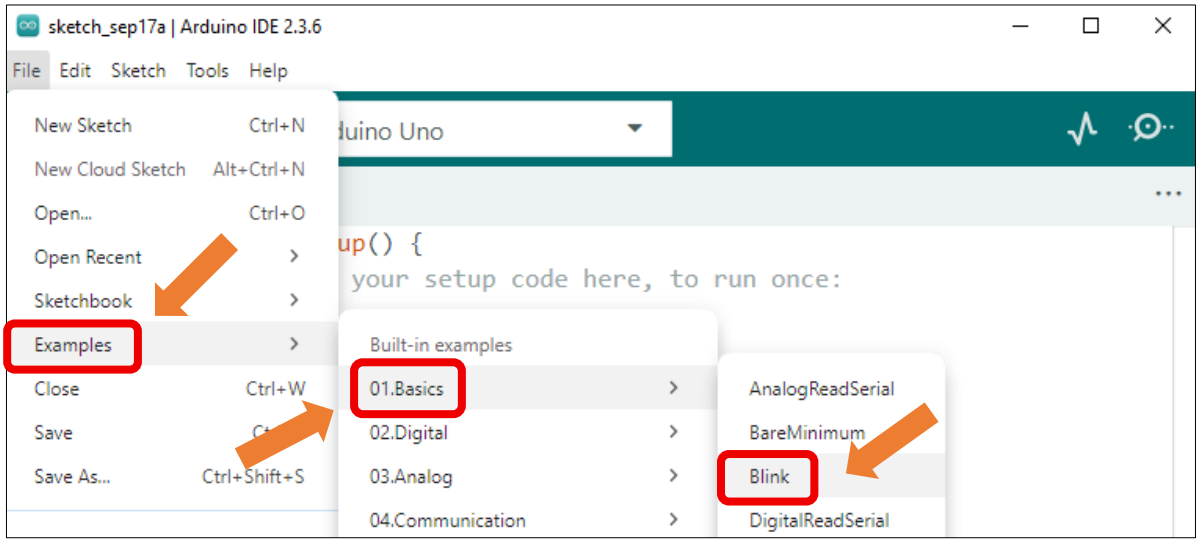


Programs written with Arduino IDE are called sketches. These sketches are written in a text editor and are saved with the file extension.ino. The editor has features for cutting/pasting and for searching/replacing text. The console displays text output by the Arduino IDE, including complete error messages 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, open the serial monitor, and access the serial plotter.

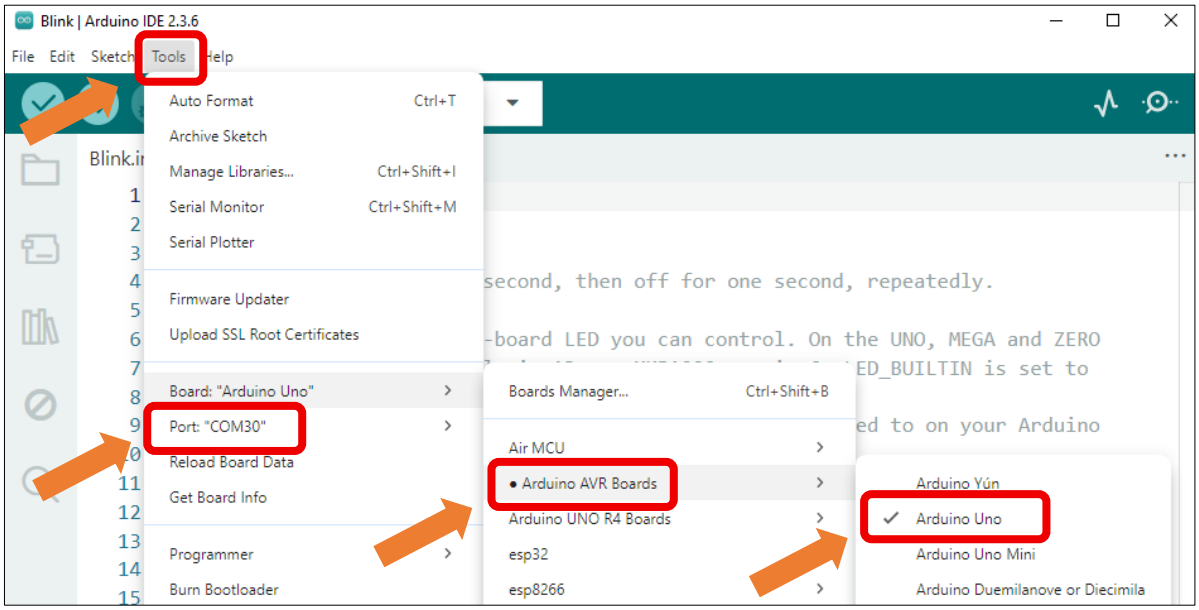
	Verify Checks your code for errors compiling it.
	Upload Compiles your code and uploads it to the configured board.
	Debug Troubleshoot code errors and monitor program running status.
	Serial Plotter Real-time plotting of serial port data charts.
	Serial Monitor Used for debugging and communication between devices and computers.

First Use

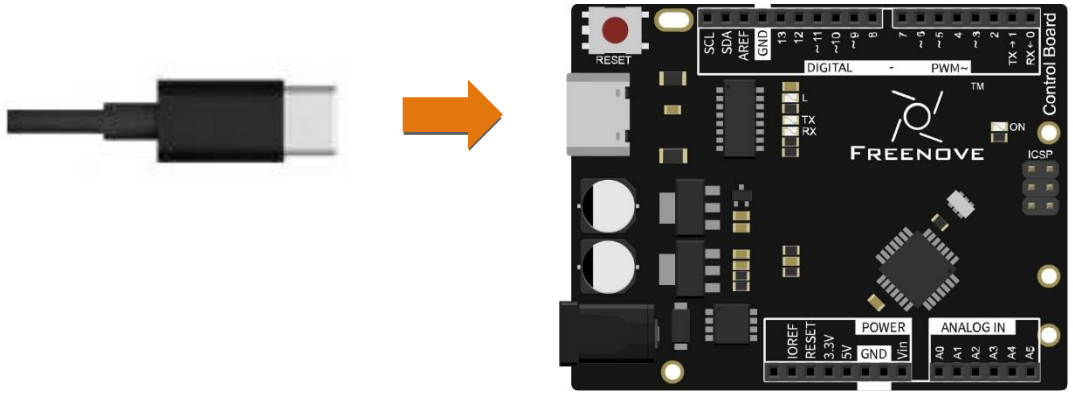
Open the example sketch "Blink".



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



Connect control board to your computer with USB cable.



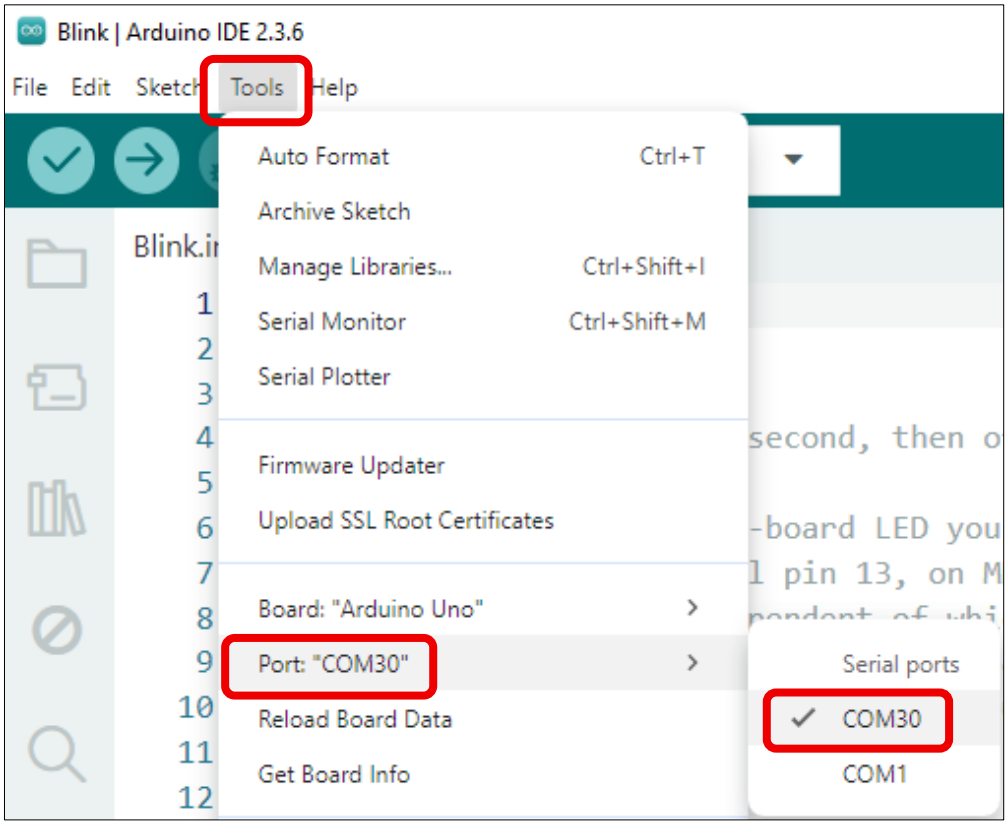
Select the port.

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.

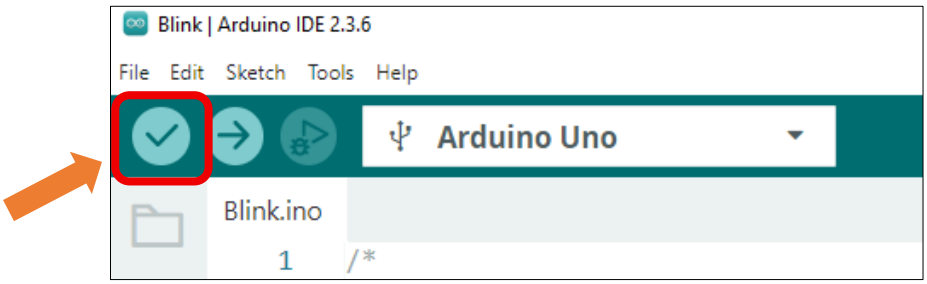


Note: If there is more than one port and you cannot decide which one to choose, disconnect the USB cable and check the port. Then connect the USB cable and check the port again. The new one is the correct port. If there is no COM port for control board, you may need to install a driver to your conputer.

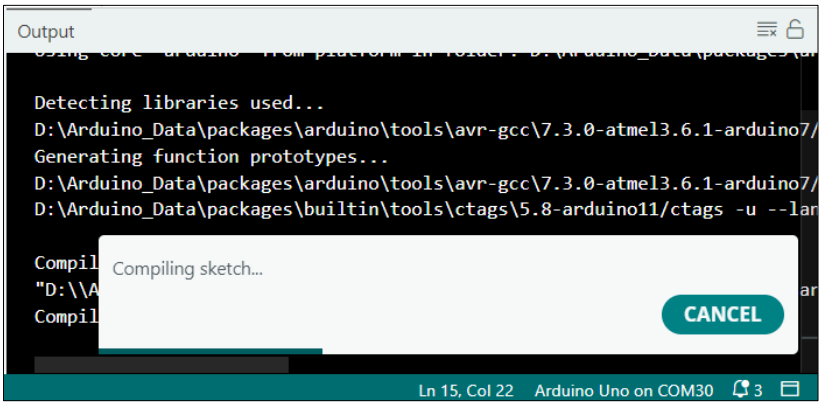
- For blue board, reinstall the latest version of Arduino IDE. During installation, agree to install the driver.
- For black board, see “InstallDriver.pdf” in “Drivers” folder (in the folder contains this Tutorial.pdf).

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

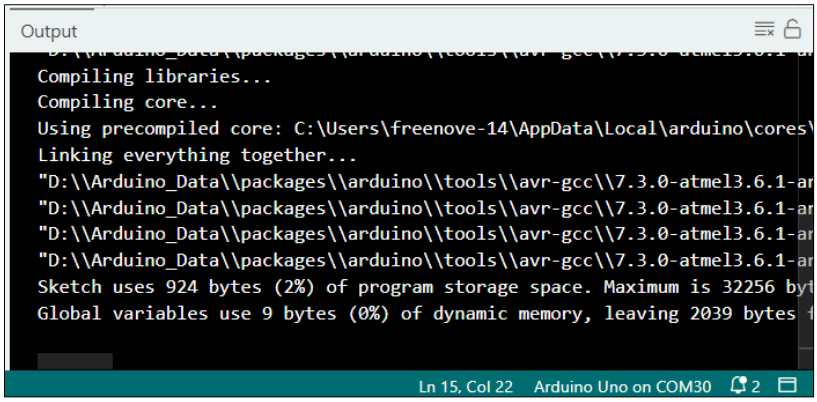
Click "Verify" button.



The following figure shows the code being compiled.

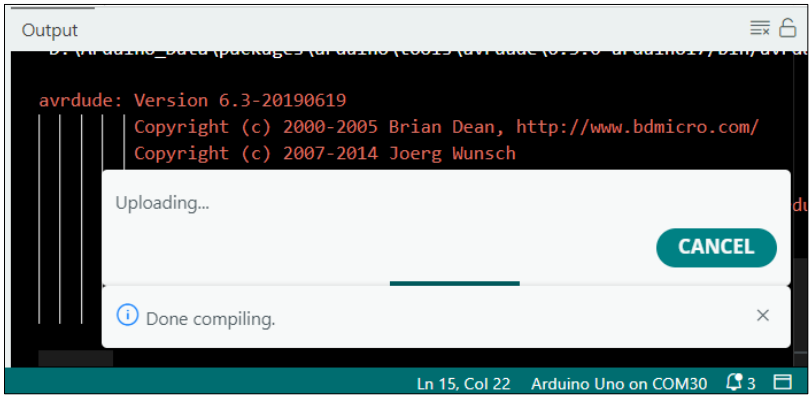


Wait a moment for the compiling to be completed. Figure below shows the code size and percentage of space occupation. If there is an error in the code, the compilation will fail and the details are shown here.

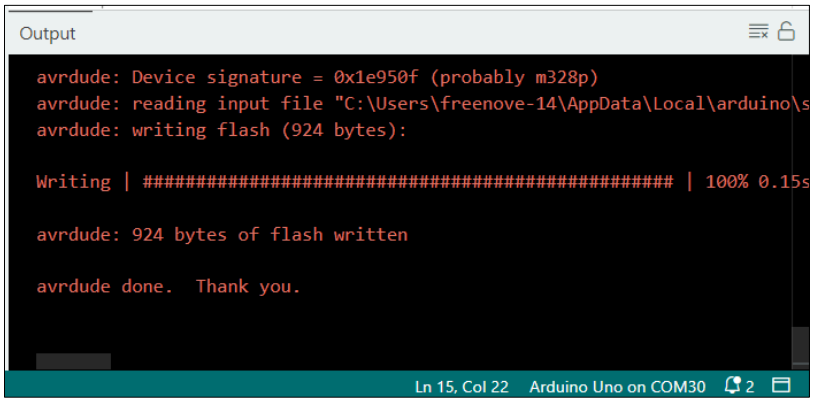


Click "Upload" button.

Figure below shows code are uploading.

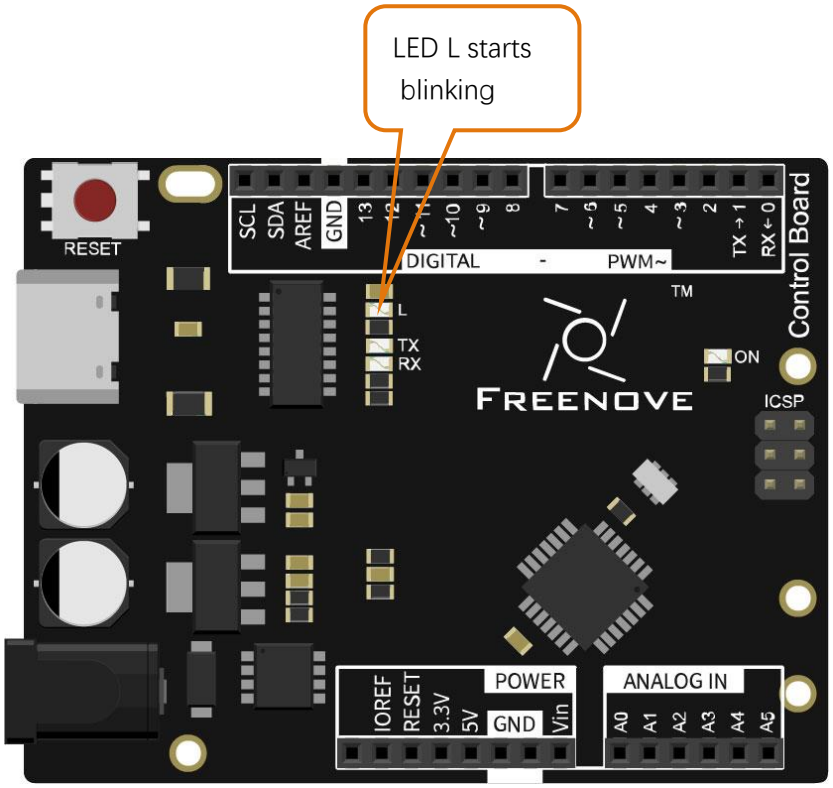


Wait a moment, and then the uploading is completed.



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

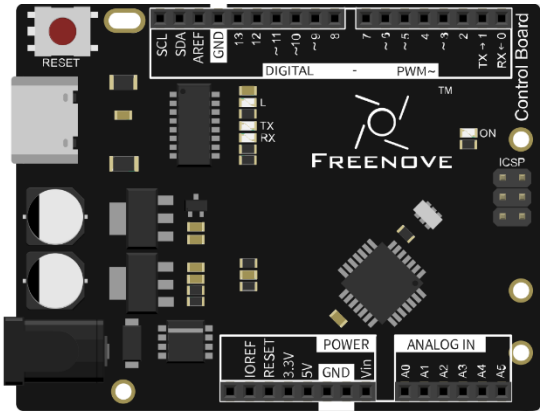
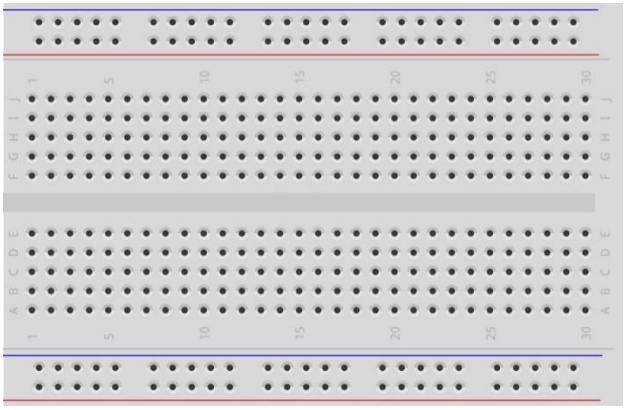
We have previously tried to make the LED marked with "L" blink on the control board. Now let us use electronic components and codes to reproduce the phenomenon, and try to understand their principle.

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.

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

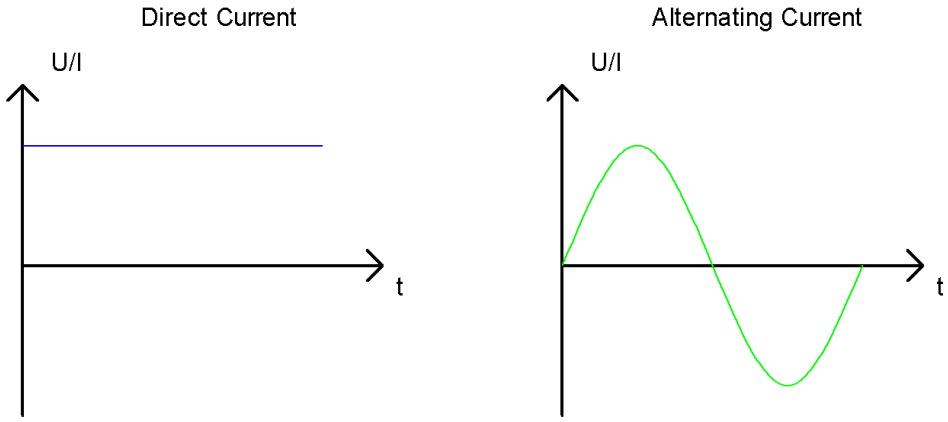
Circuit Knowledge

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.

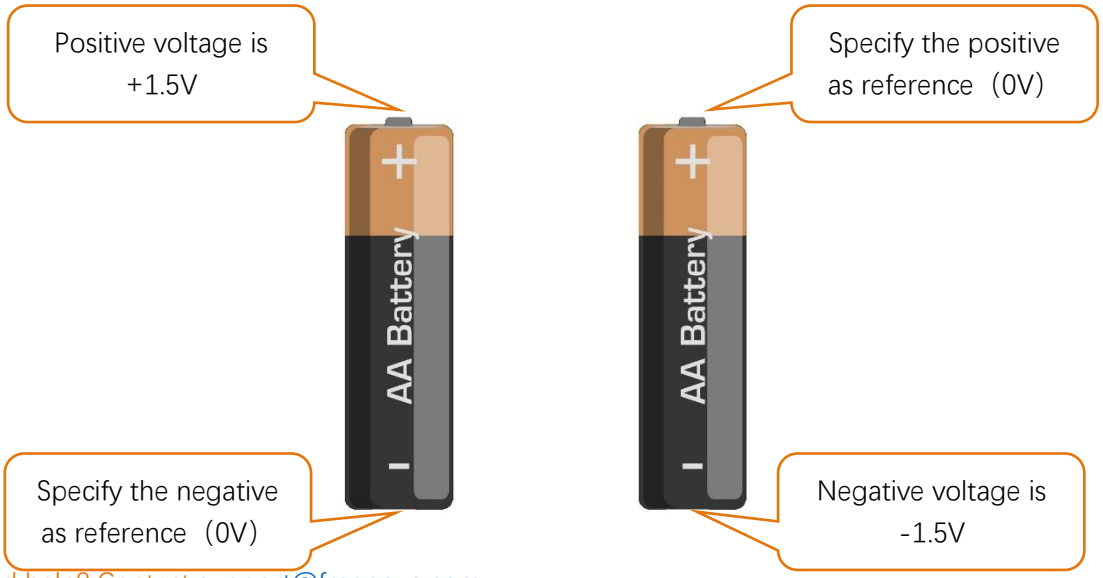
Battery or battery pack can be represented by the following symbols:

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.

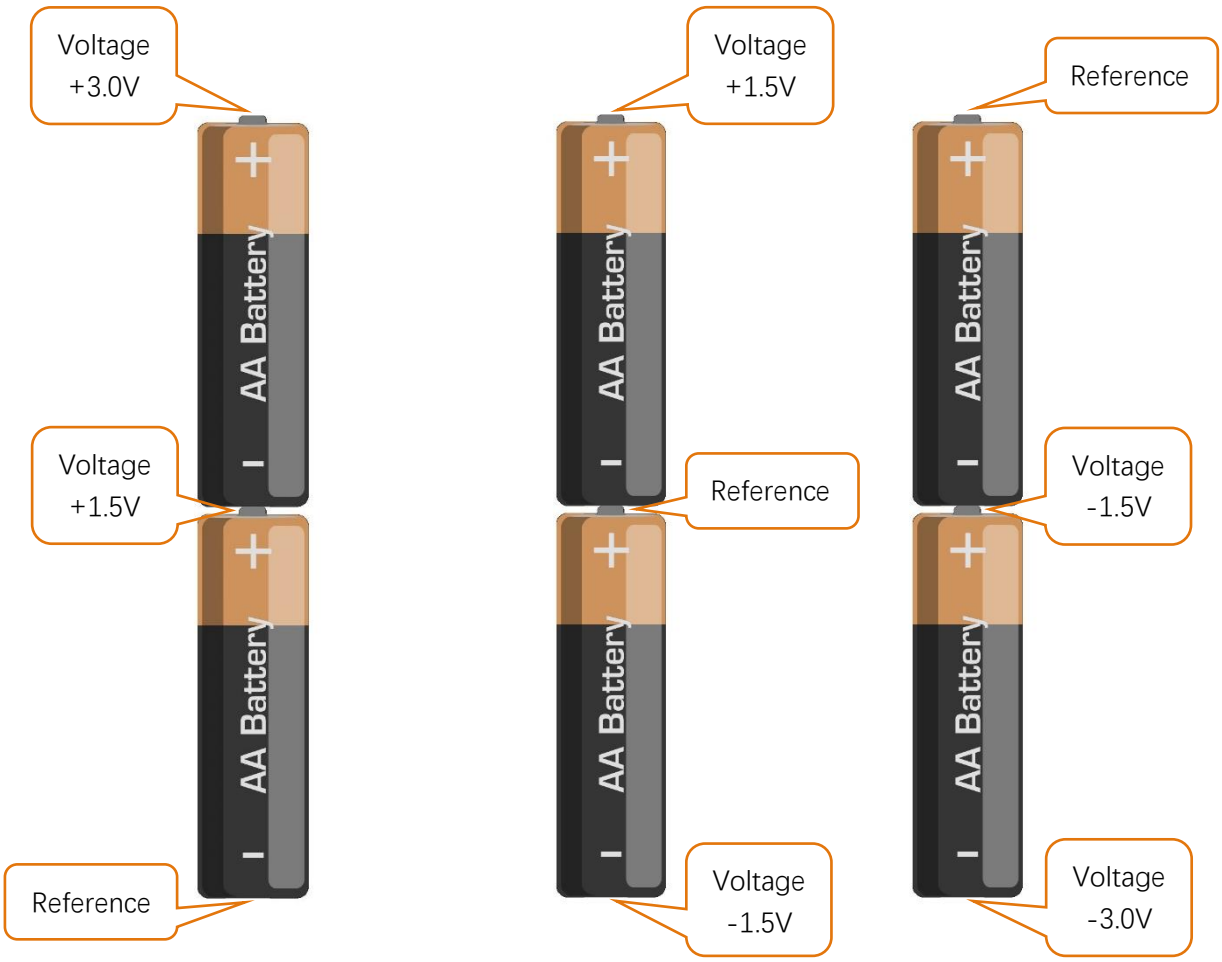
Voltage

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:

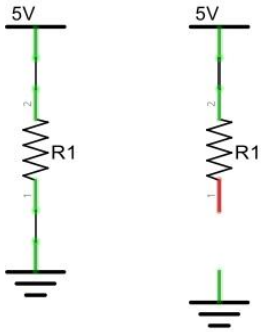
Positive(VCC) Negative(Ground, GND)

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.

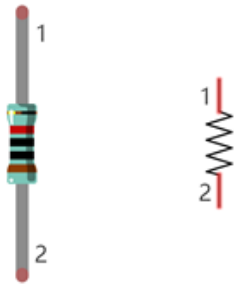
In the figures below: the left one is a loop circuit, so current flows through the circuit. The right one is not a loop circuit, so there is no current.



Resistor

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

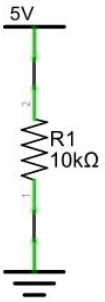
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

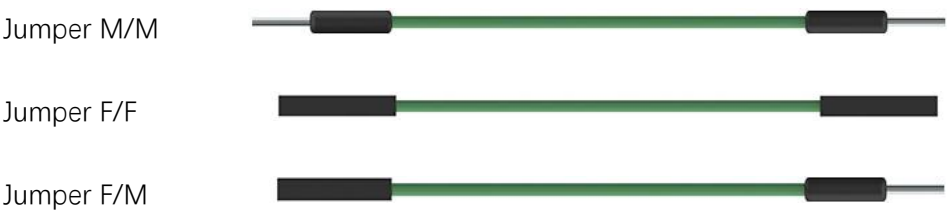
Component Knowledge

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.

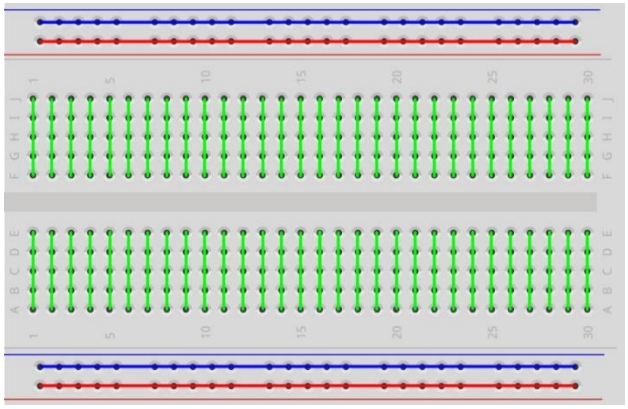
Jumpers have male end (pin) and female end (slot), so jumpers can be divided into the following 3 types.



Breadboard

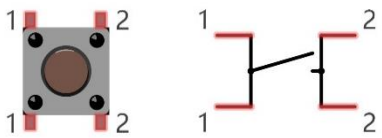
There are many small holes on breadboard to connect Jumpers.

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:

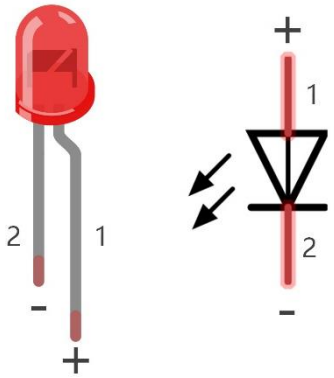


When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

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.



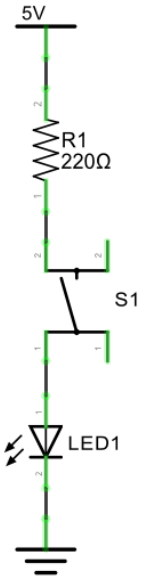
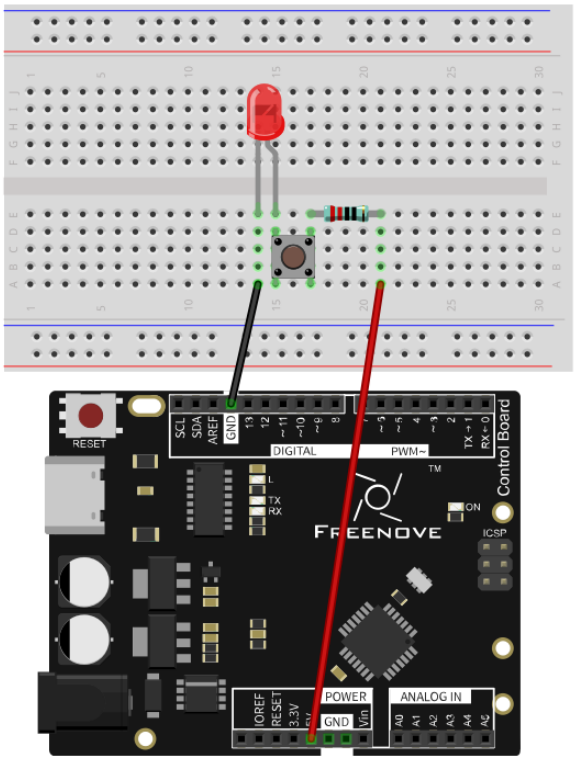
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Circuit

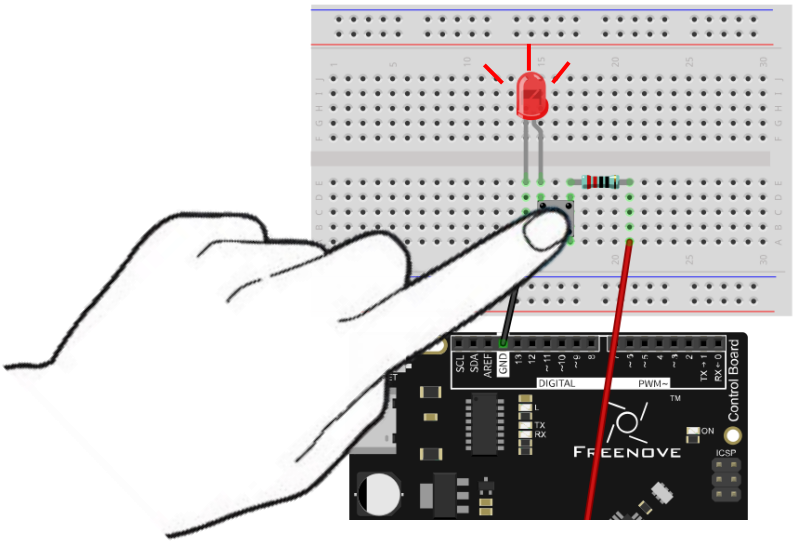
In this project, the LED is controlled by a push button switch, and the control board here only plays the role of power supply in the circuit.

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

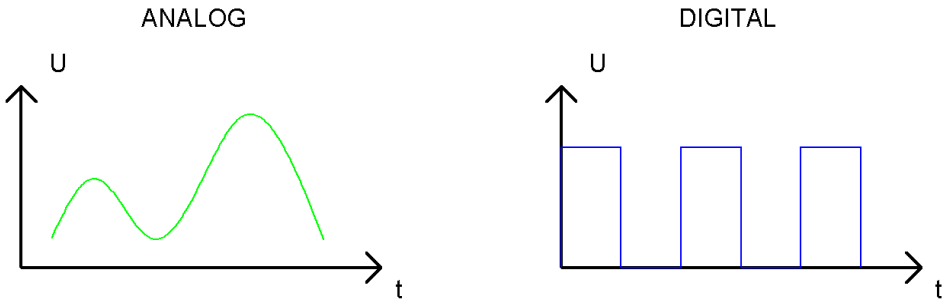
Components are basically the same with those in last section. Push button switch is no more needed.

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.



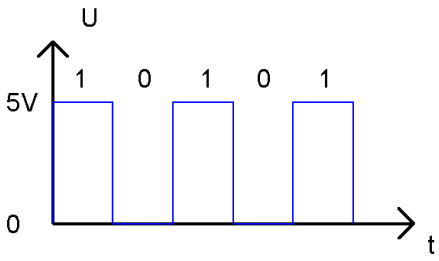
In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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

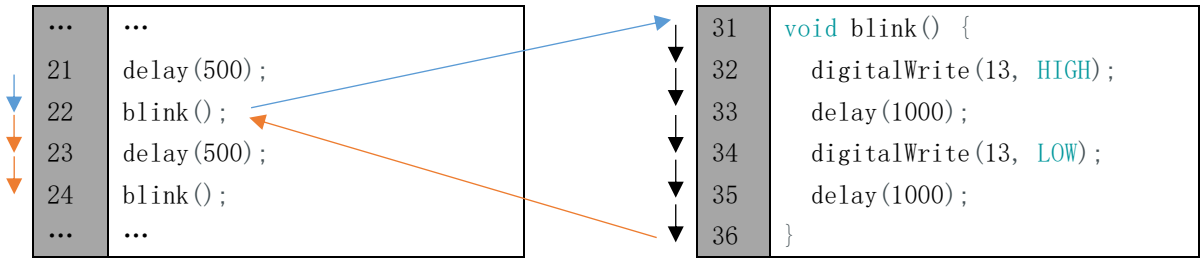
A function is a collection of statements with a sequence of order, which performs a defined task. Let's define a function void blink() as follows:

```
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.



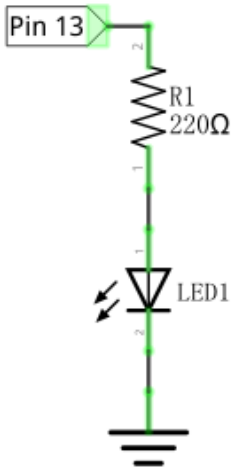
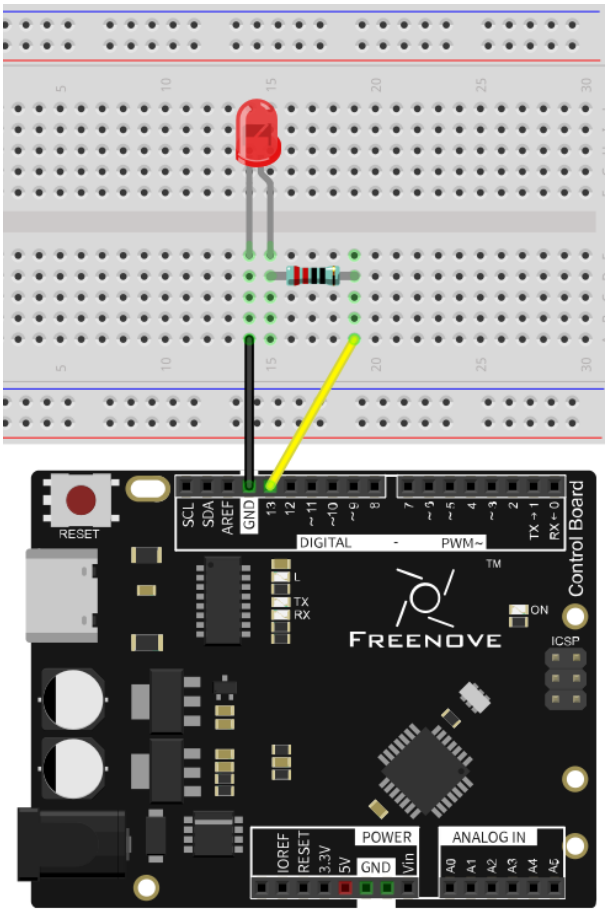
Some functions have one or more parameters. When you call such functions, you need to write parameters inside "()":

```
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 }
```

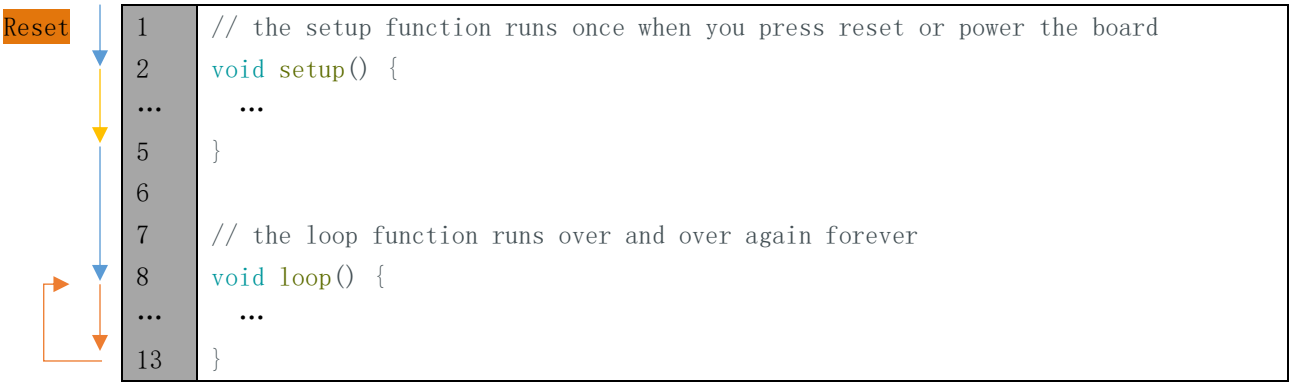
The code usually contains two basic functions: void setup() and void loop().

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
```

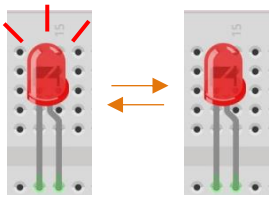
Then set the 13 pint to output low level, and LED light off. One second later, the execution of loop () function will be completed.

```
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.



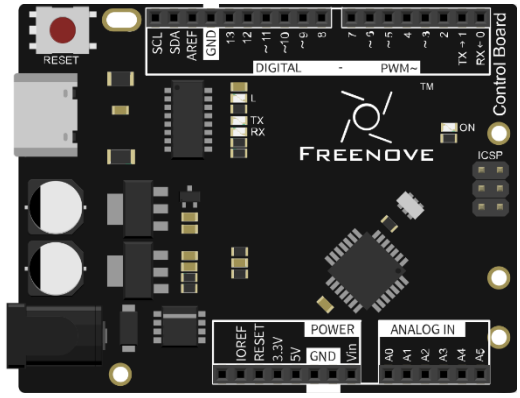
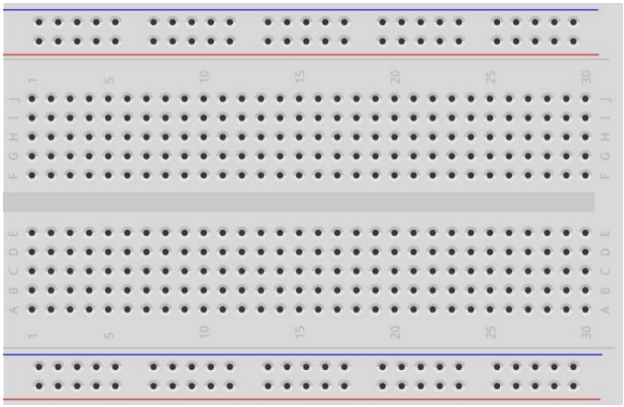
Chapter 2 Two LEDs Blink

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
```

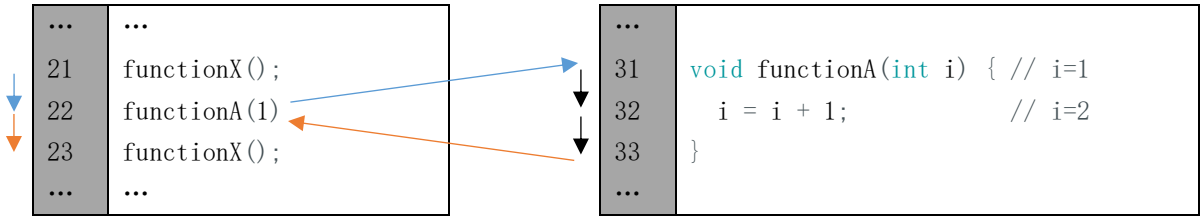
Next, we will define a function with a parameter as below:

```
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.

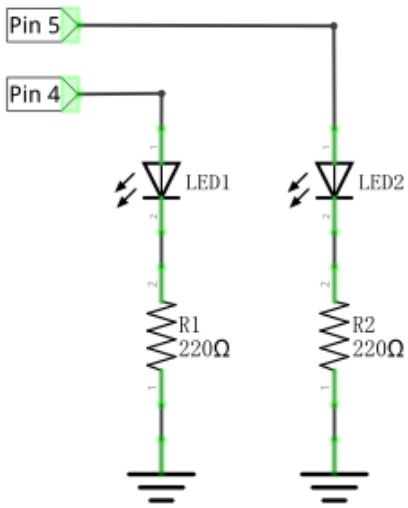
```
1   int i = 1;
2   if (i == 0)       // false
3     i = 1;          // no execution
4   else if (i == 1)// true
5     i = 2;          // execution
6   else if (i == 2)
7     i = 3;          // no execution
8   else
9     i = 4;          // no execution
10  functionX();
…   …
```

```
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();
…   …
```

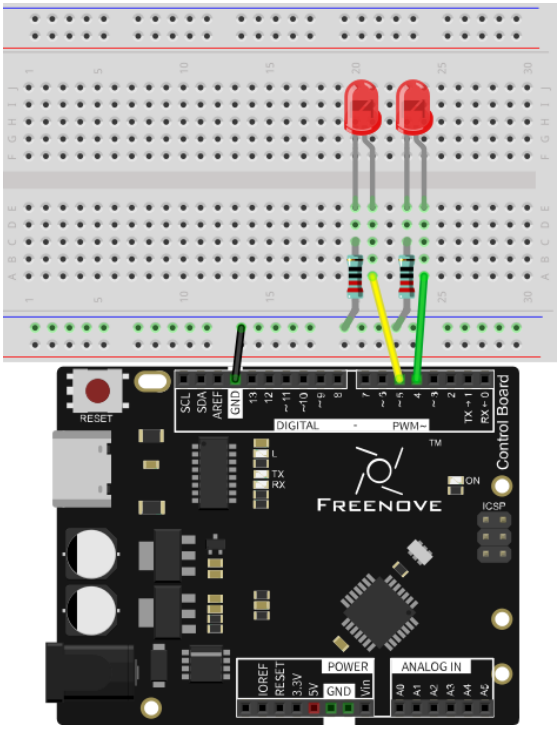
Circuit

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.

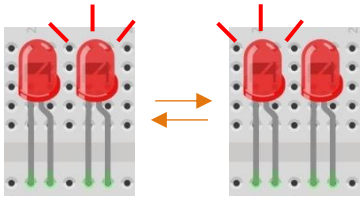
```
// set pin numbers:
int led1Pin = 4;            // the number of the LED1 pin
int led2Pin = 5;            // the number of the LED2 pin

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.

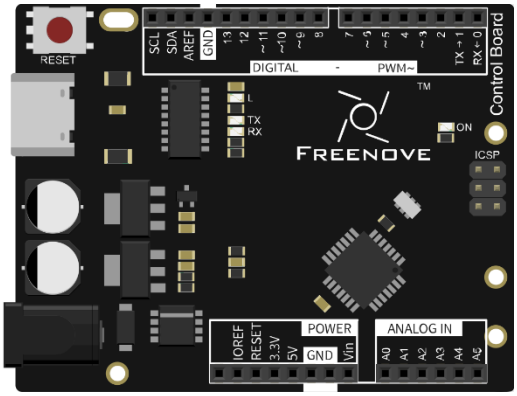
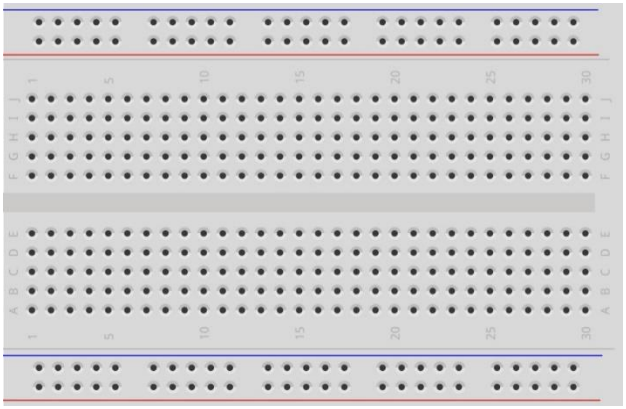
Chapter 3 LED Bar Graph

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

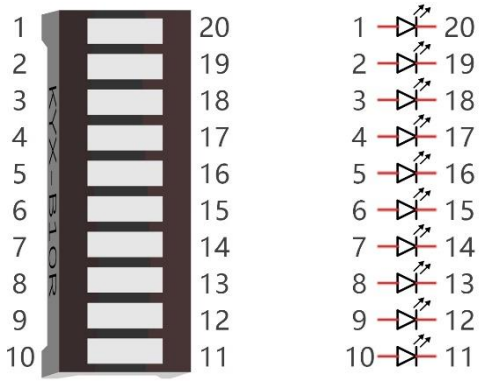
Control board x1 	Breadboard x1 	
USB cable x1 	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

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.

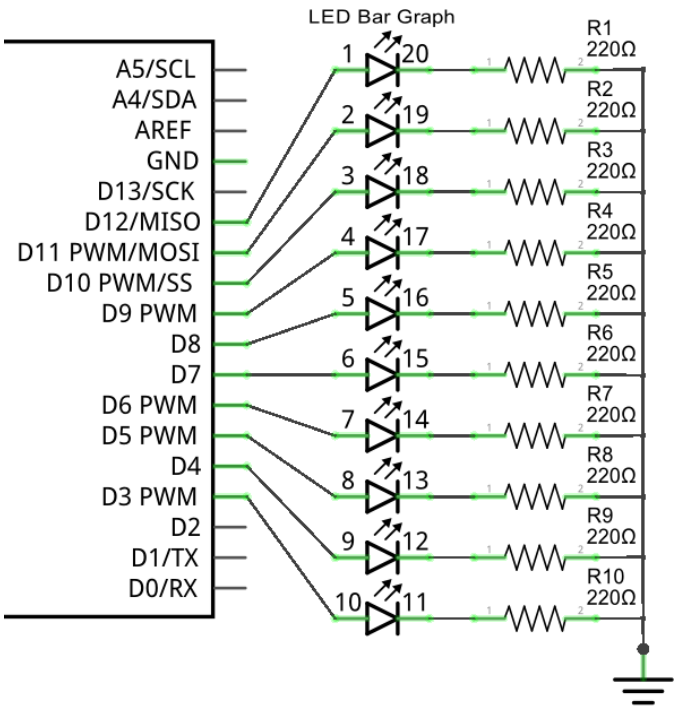
Operator ++, --

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

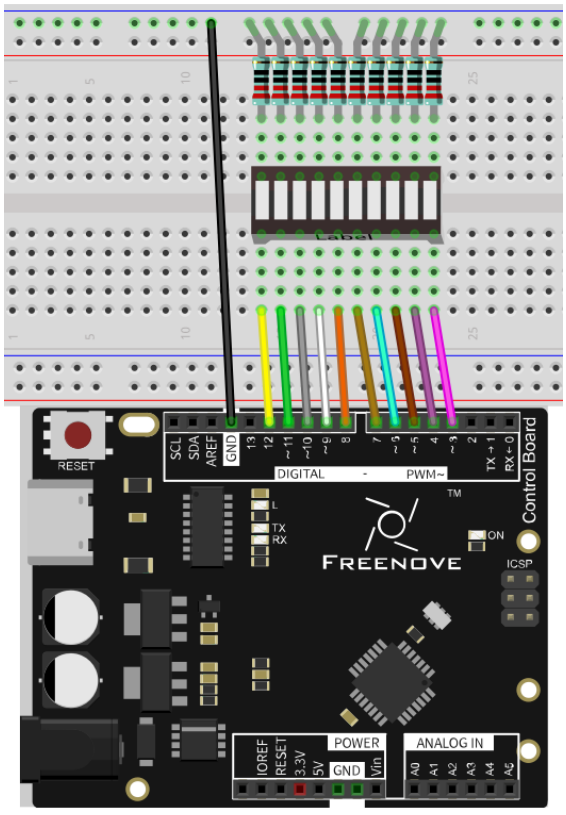
Circuit

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.

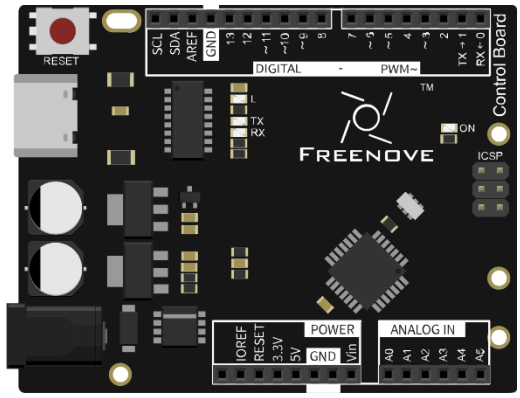
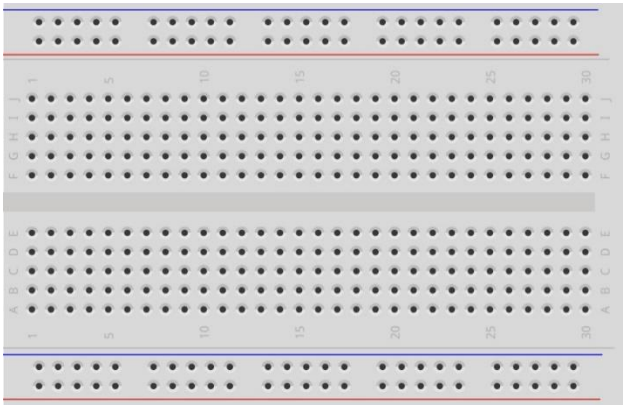
Chapter 4 LED Blink Smoothly

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

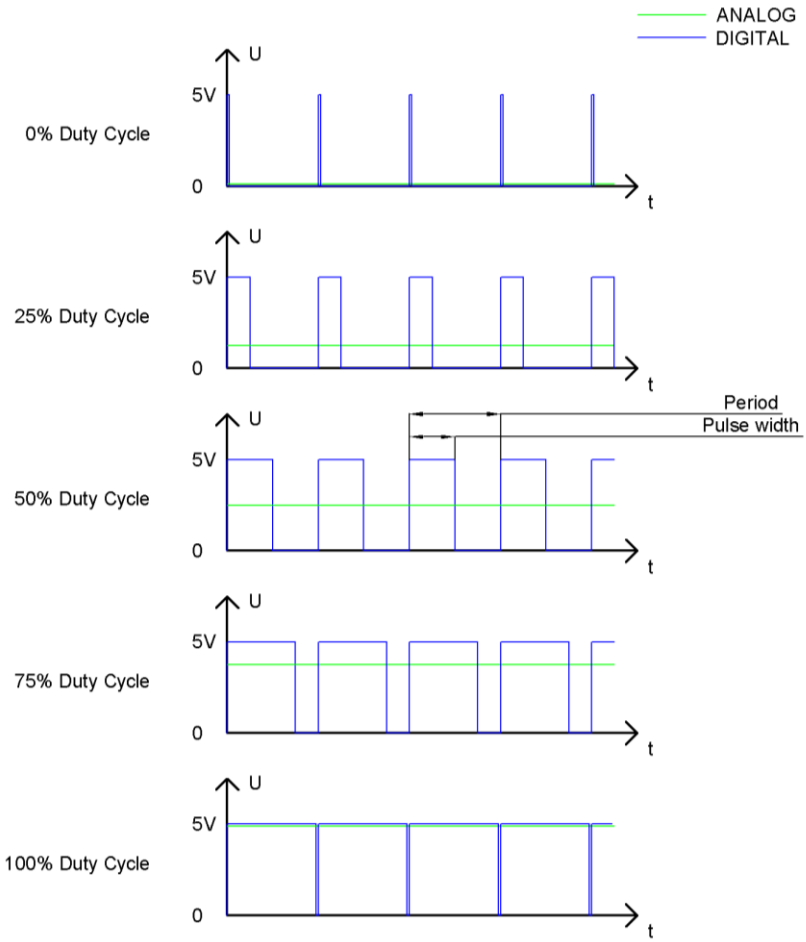
At first, let us learn how to use the circuit to make LED emit different brightness of light,

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:

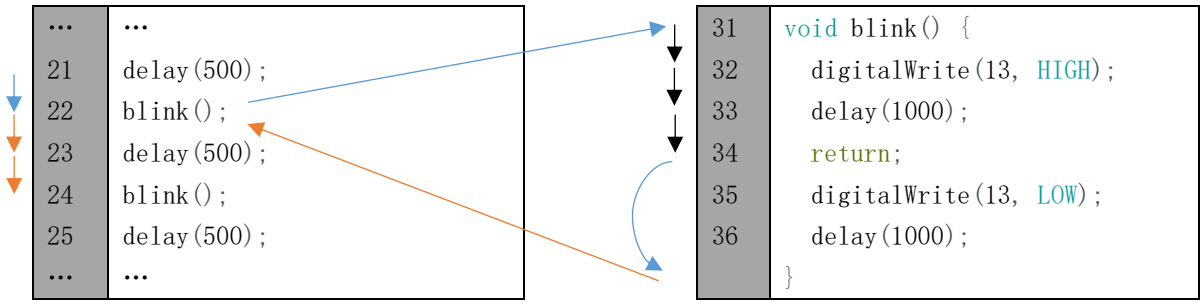
```
1  delay(300);
```

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:

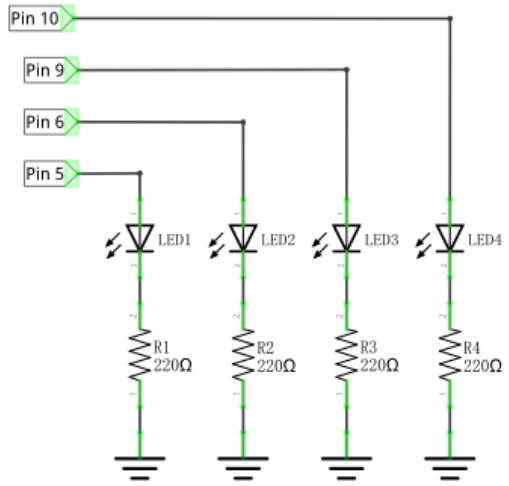
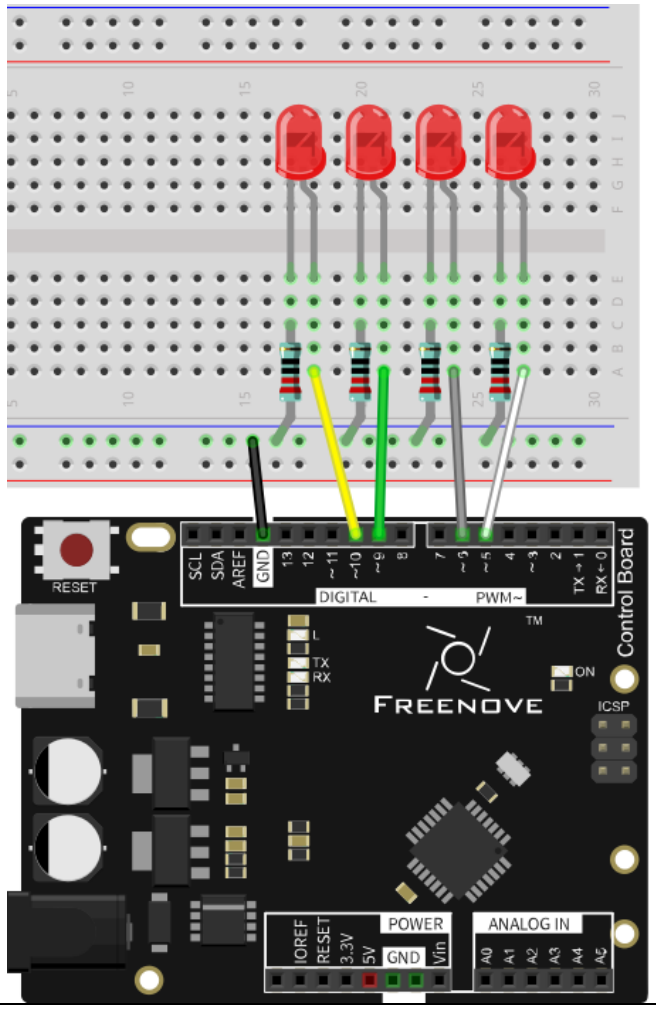
```
1  return;
```

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));
```

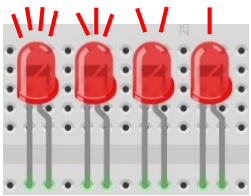
analogWrite(pin, value)

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

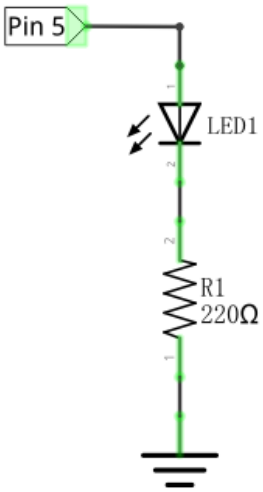
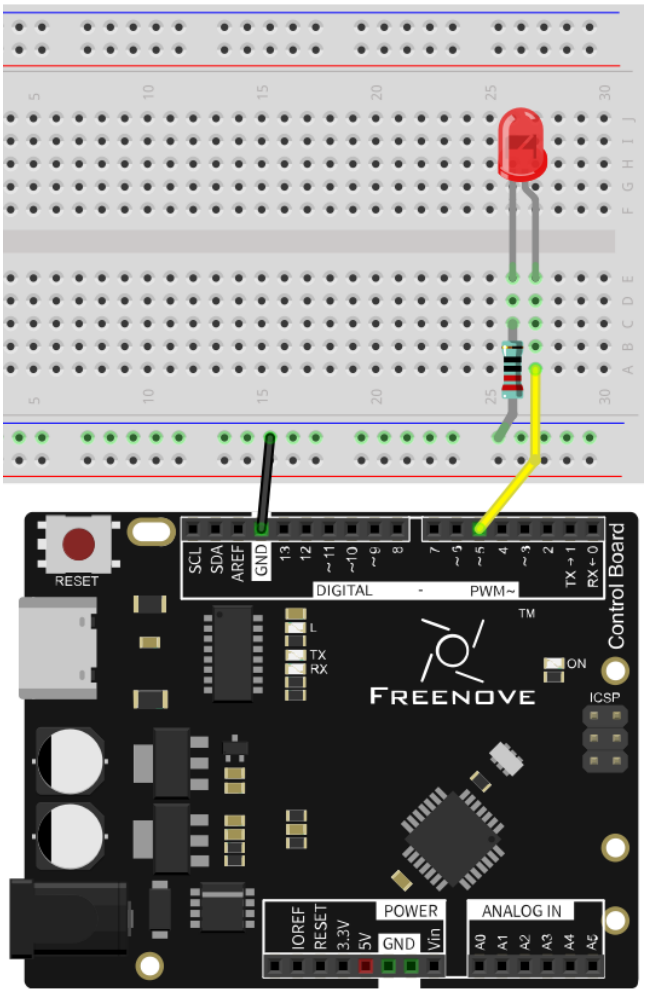
We will learn how to make a LED blink smoothly, that is, breathing light.

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

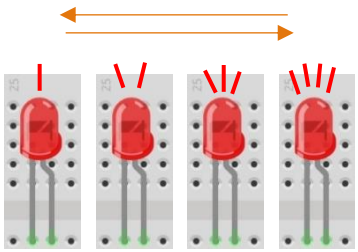
Sketch 4.2.1

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.



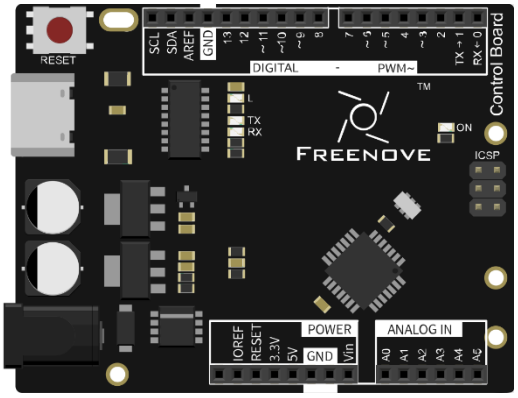
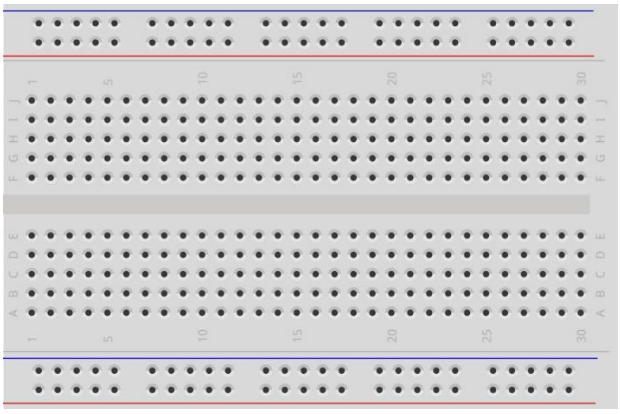
Chapter 5 Control LED with Push Button Switch

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.

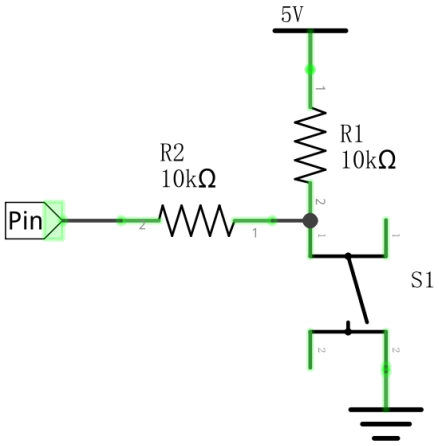
Component List

<table>
<tr><td>Control board x1
  </td><td colspan="4">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">Resistor
10kΩ x2
  </td><td rowspan="2">Push button
Switch x1
  </td></tr>
<tr><td>Jumper M/M x4
  </td></tr>
</table>

Circuit Knowledge

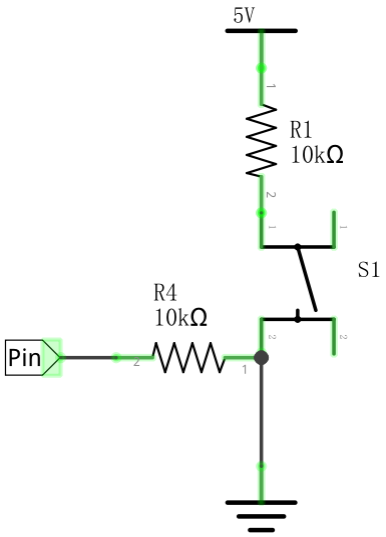
Connection of Push Button Switch

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:



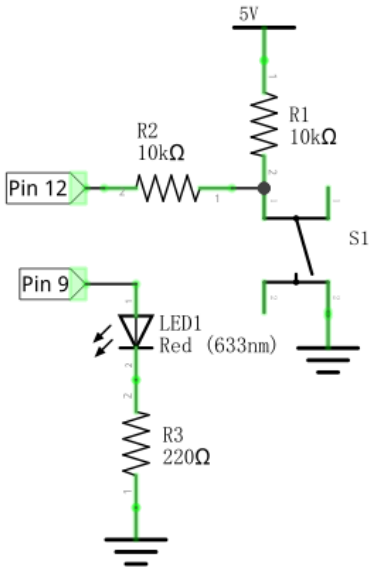
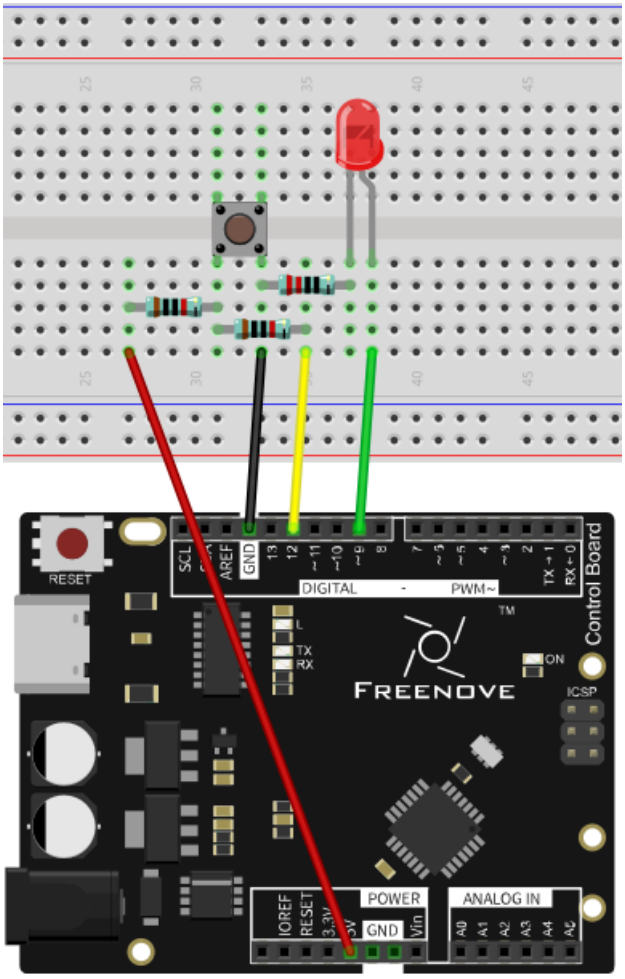
In the above circuit diagram, when the button is not pressed, 5V (high level) will be detected by control board port; and 0V (low level) when the button is pressed. The role of Resistor R2 here is to prevent the port from being set to output high level by accident. Without R2, the port could be connected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

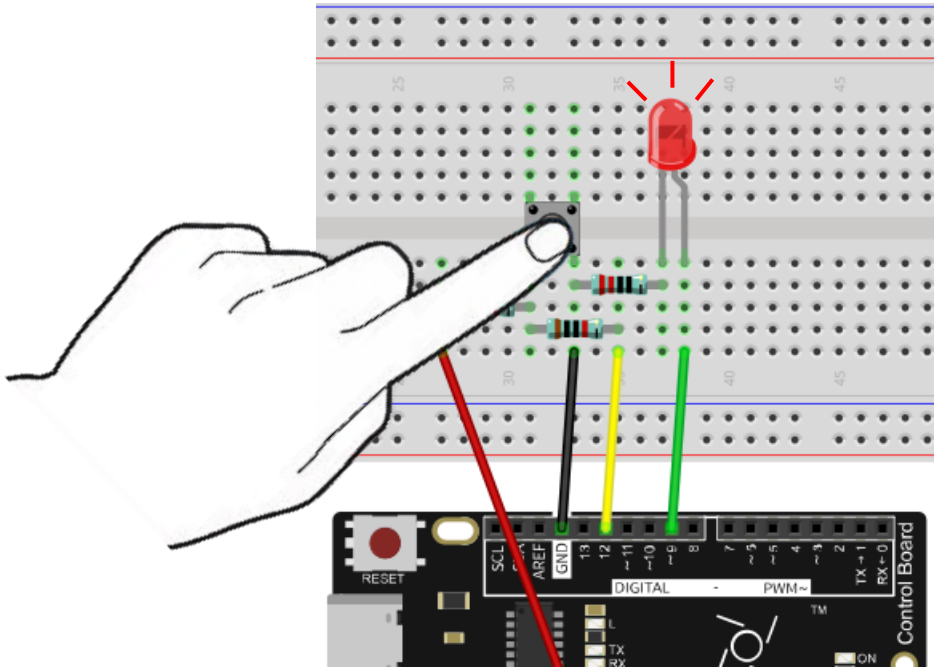
```
int buttonPin = 12; // the number of the push button pin
int ledPin = 9;     // the number of the LED pin

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.

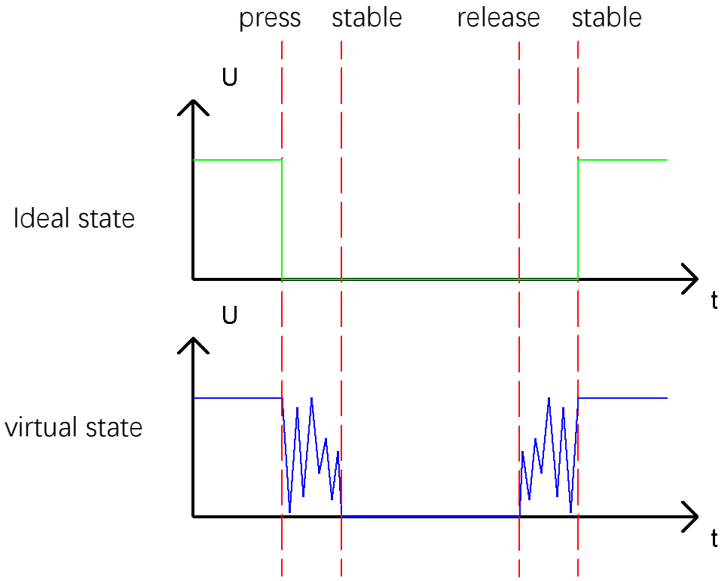
Component List

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.

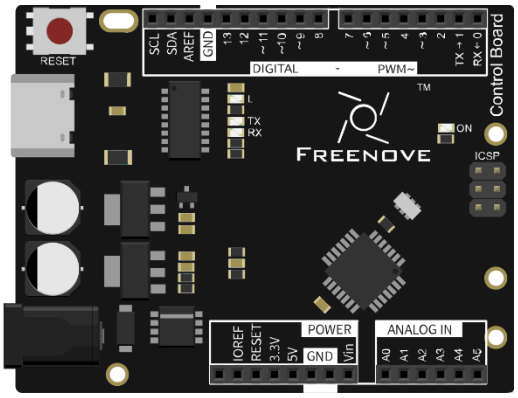
Chapter 6 Serial

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:

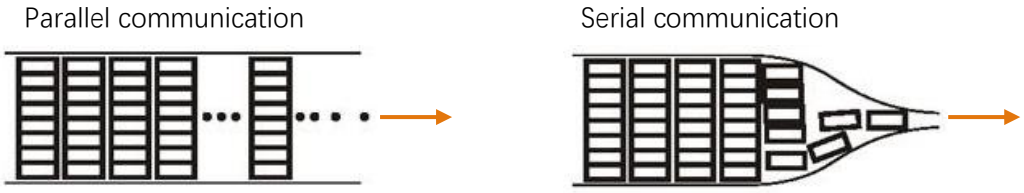
		Remainder	Sequence
2	150	0	0
2	75	1	1
2	37	1	2
2	18	0	3
2	9	1	4
2	4	0	5
2	2	0	6
2	1	1	7
	0		

The result is 10010110.

Circuit Knowledge

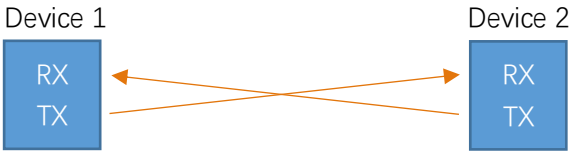
Serial and parallel communication

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:

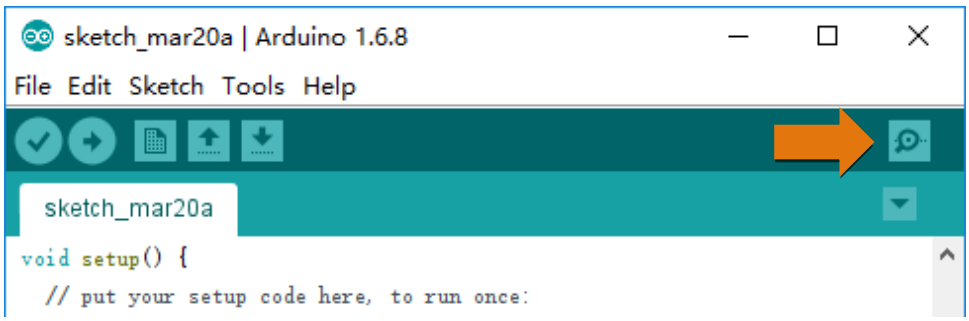


Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

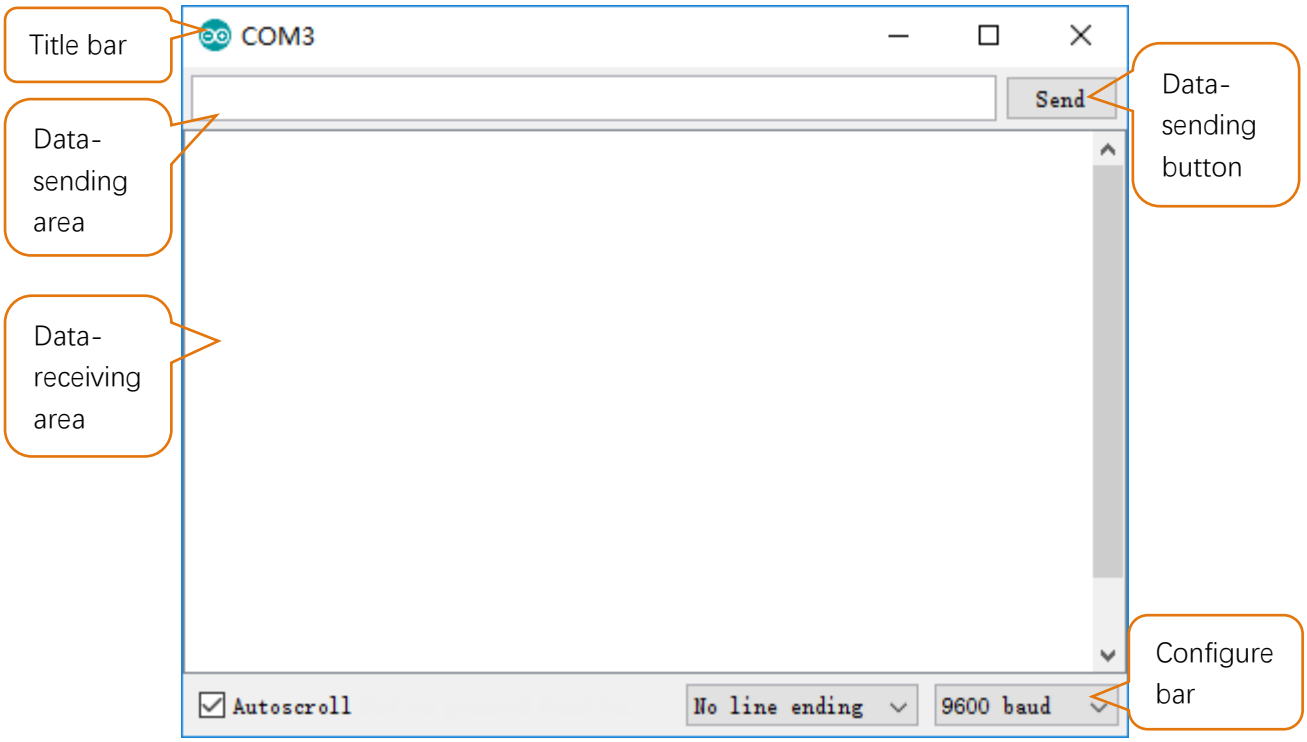
Serial port on Control board

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.

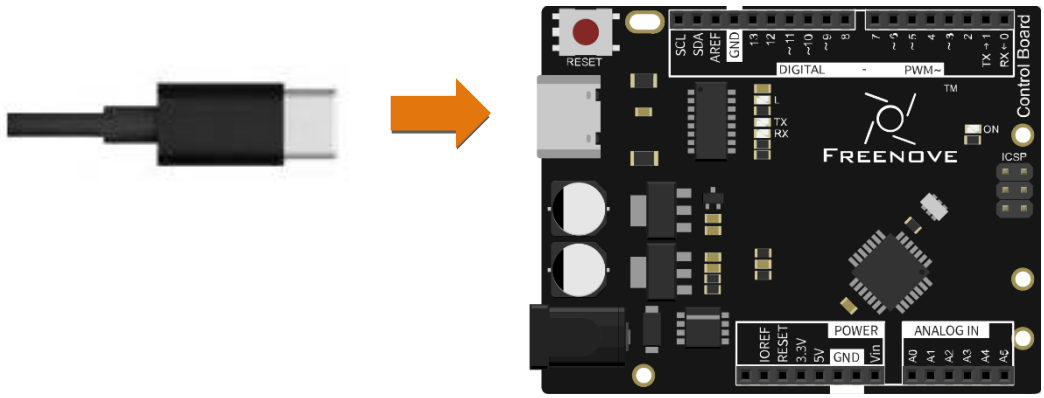


Interface of Serial Monitor window is as follows. If you can't open it, make sure control board had been connected to the computer, and choose the correct serial port in the menu bar "Tools".



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:

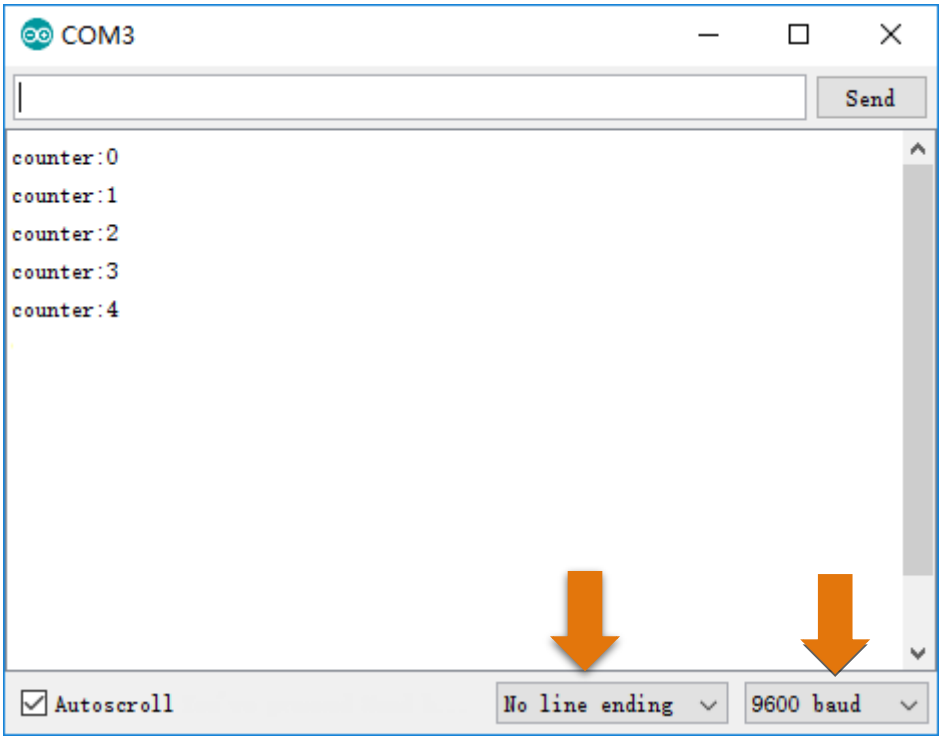
Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

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

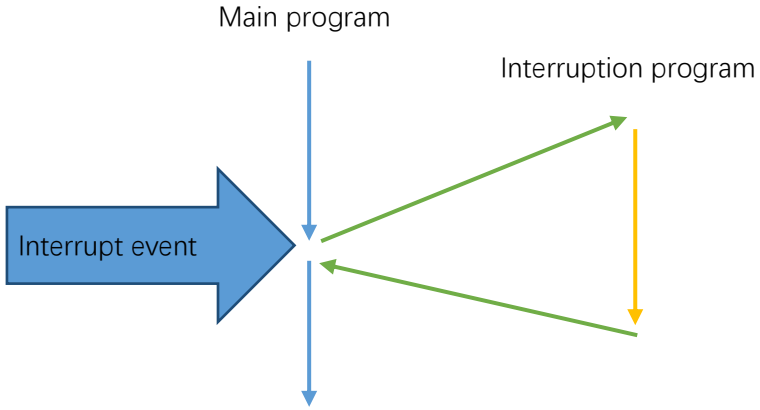
Same with the previous section.

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.

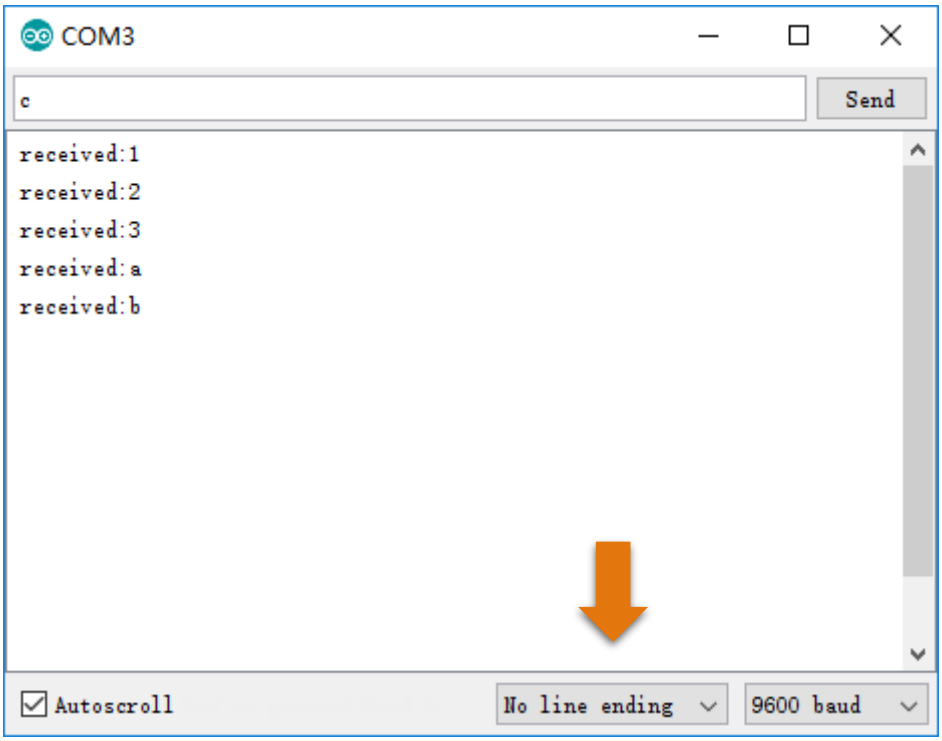
```
1  char inChar;      // define a variable to store characters received from serial port
2
3  void setup() {
4    Serial.begin(9600);              // initialize serial port, set baud rate to 9600
5  }
6
7  void loop() {
8    if (Serial.available()) {        // judge whether data has been received
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.

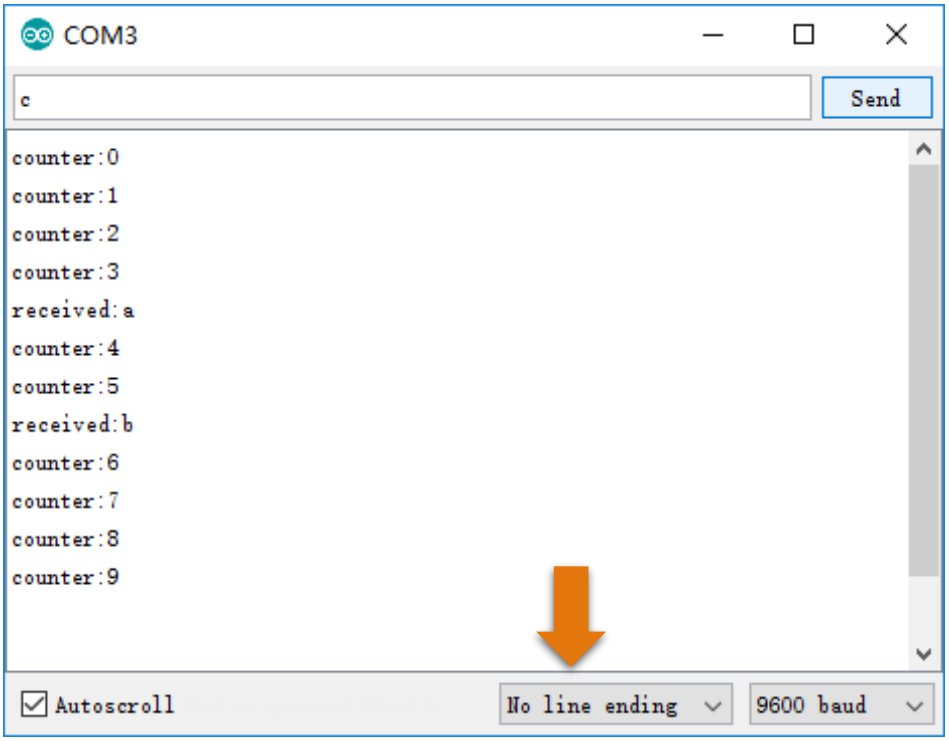
Sketch 6.2.2

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.

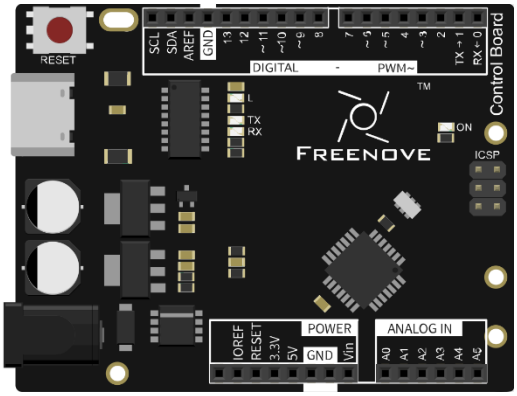
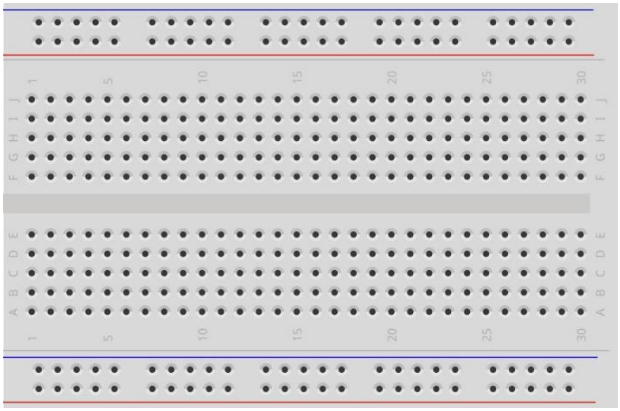
Verify and upload the code, open the Serial Monitor, then you'll see the number constantly sent from control board. Fill in characters in the sending area, and click the Send button, then you'll see the string returned from control board.



Project 6.3 Application of Serial

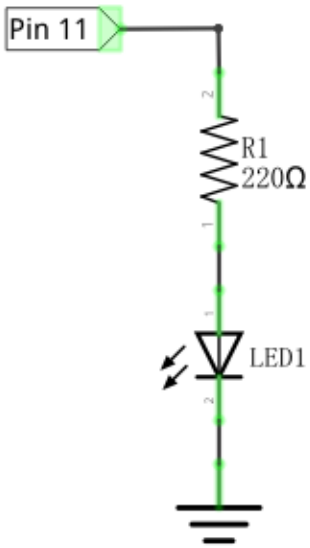
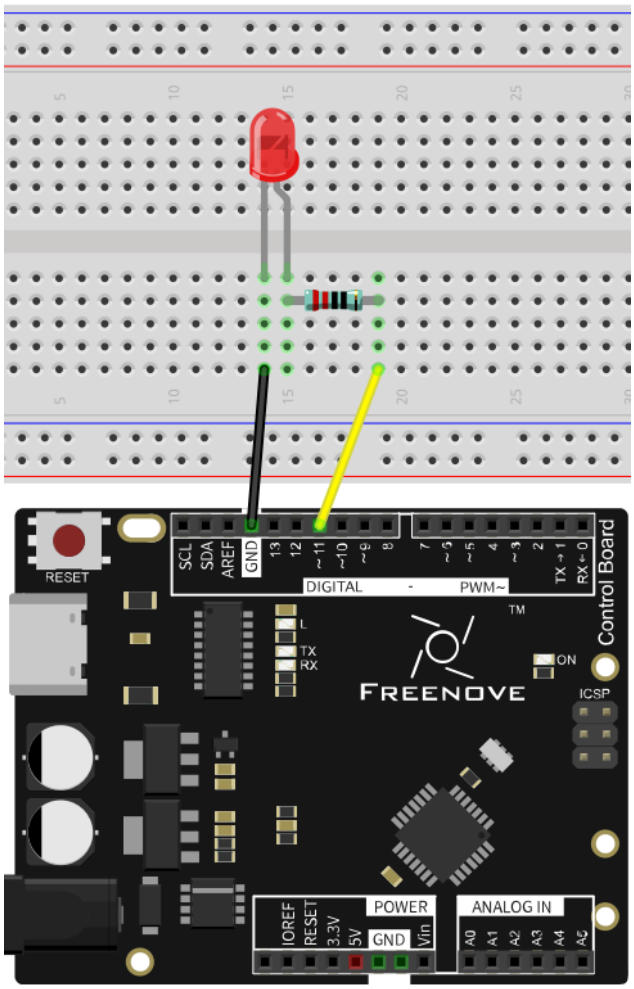
We will use the serial port on control board to control one LED.

Component List

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

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.

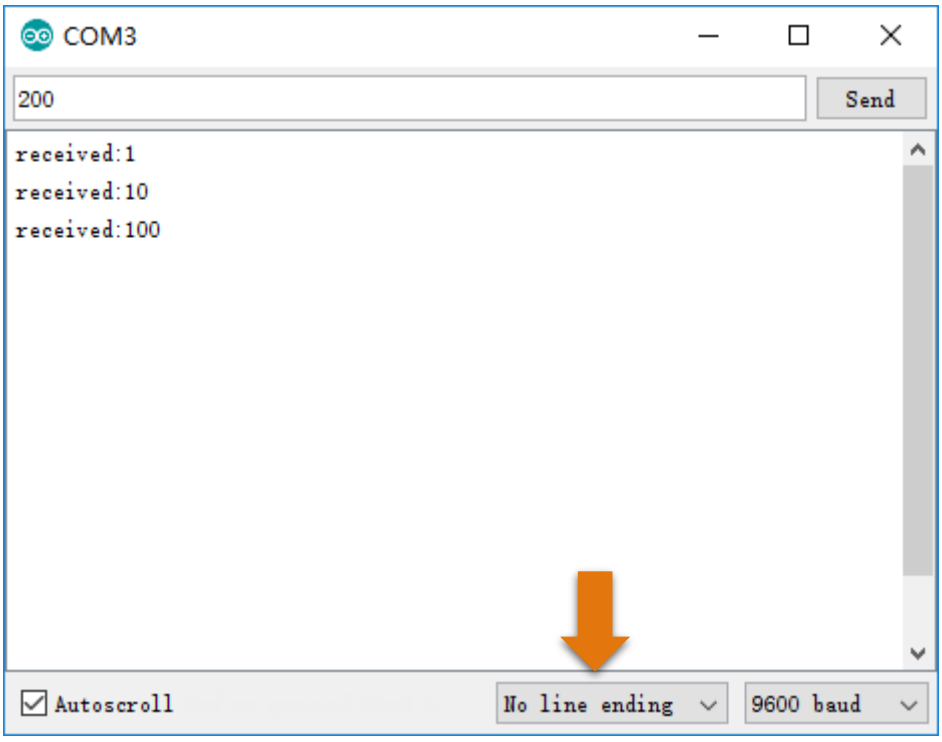
Serial Class

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.



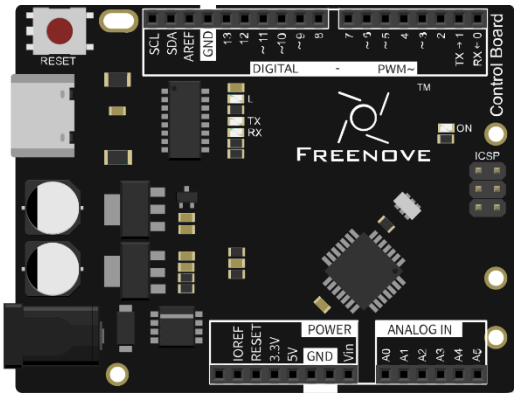
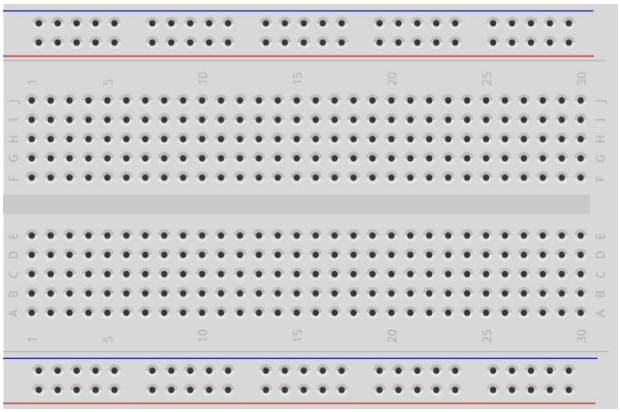
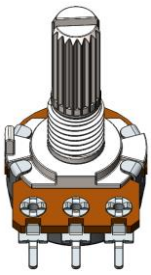
Chapter 7 ADC

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

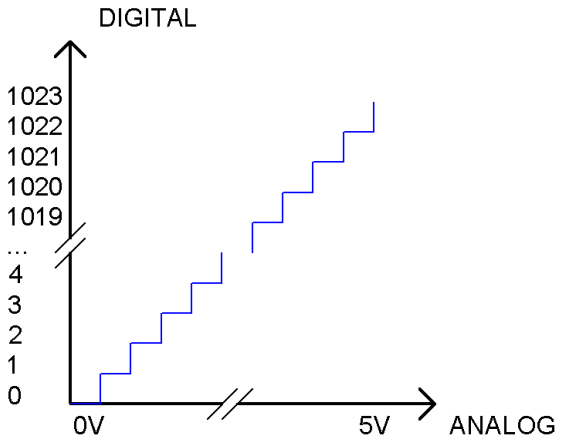
Control board x1 	Breadboard x1 
USB cable x1 	Rotary potentiometer x1 
Jumper M/M x3 	

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;

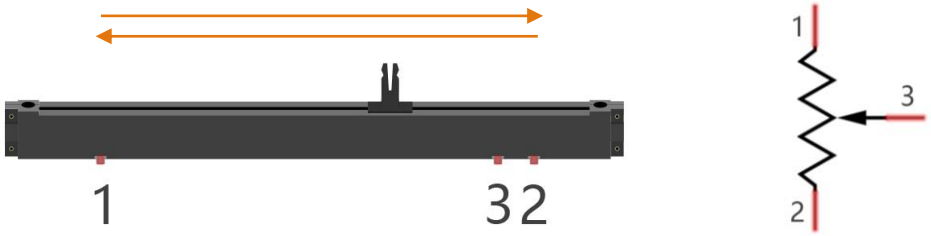
Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;

The following analog signal will be divided accordingly.

Component Knowledge

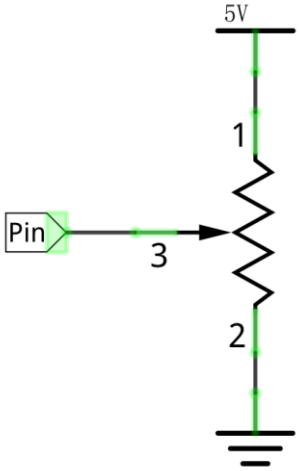
Potentiometer

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.



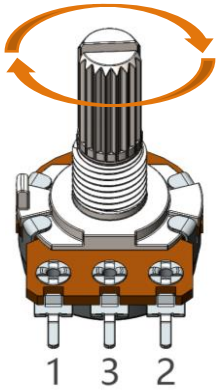
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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



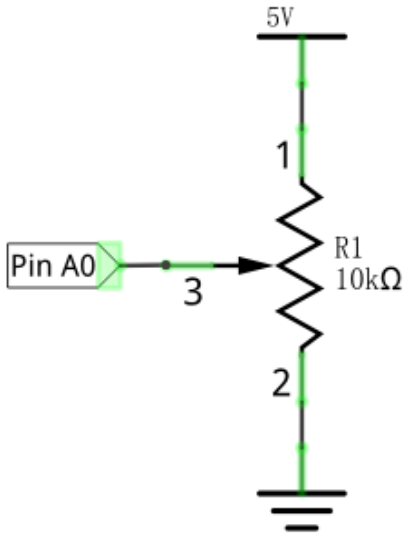
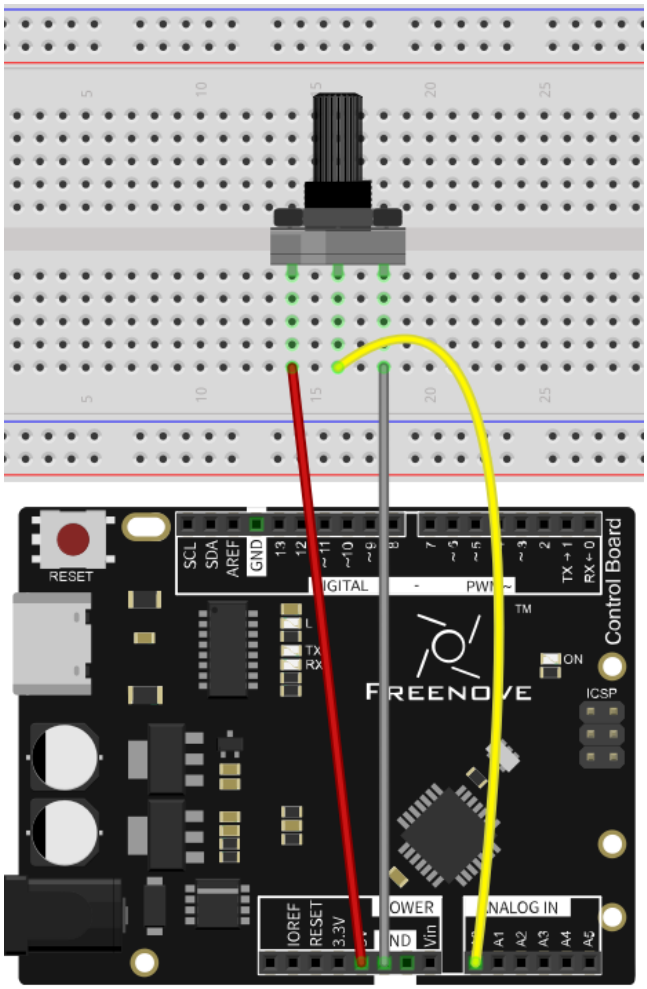
Rotary potentiometer

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

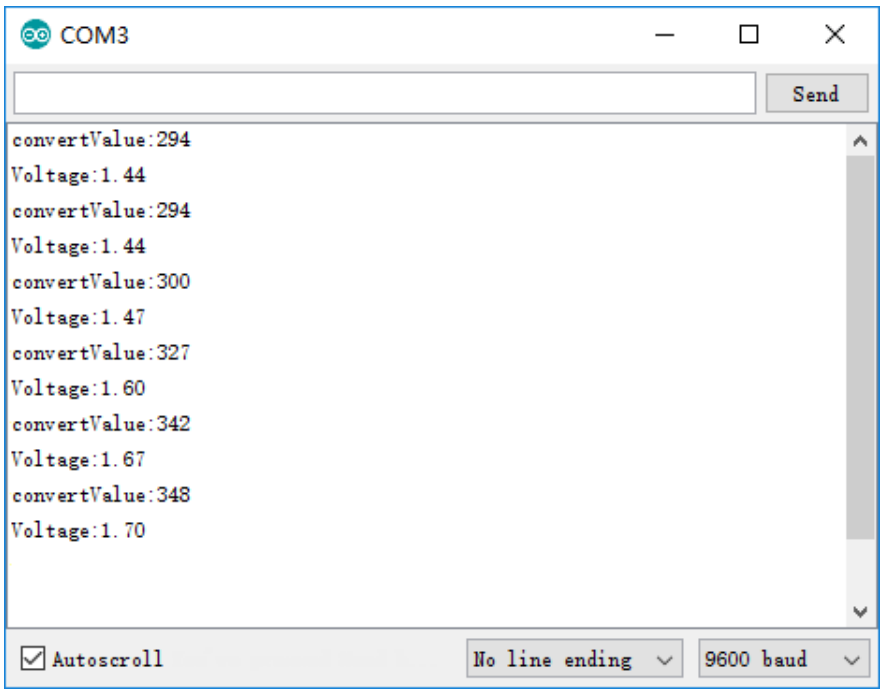
Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino IDE through serial port.

```
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.

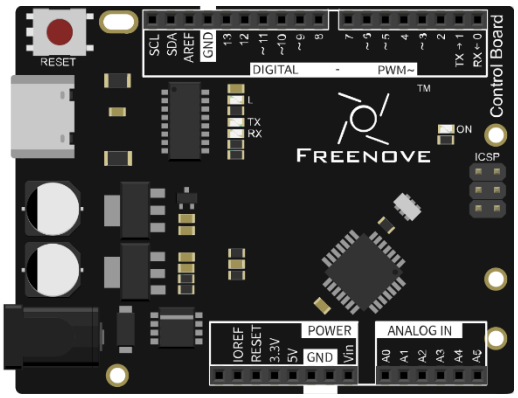
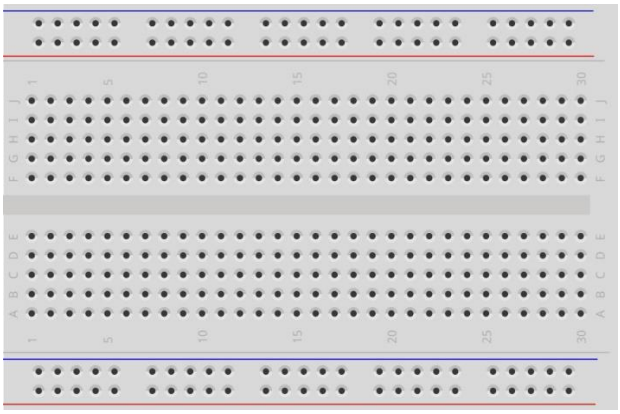
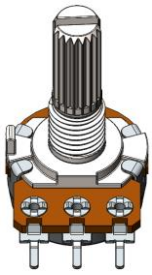
Turn the rotary potentiometer shaft, and you can see the voltage change.



Project 7.2 Control LED by Potentiometer

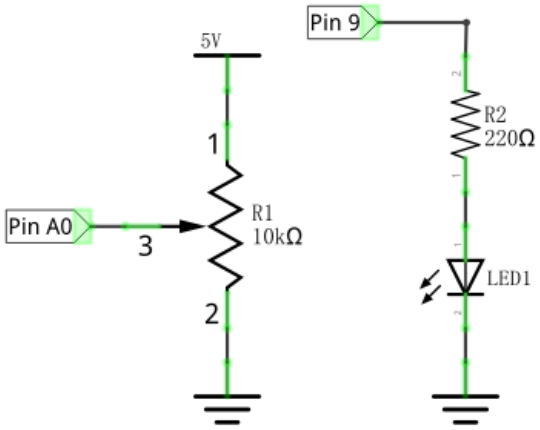
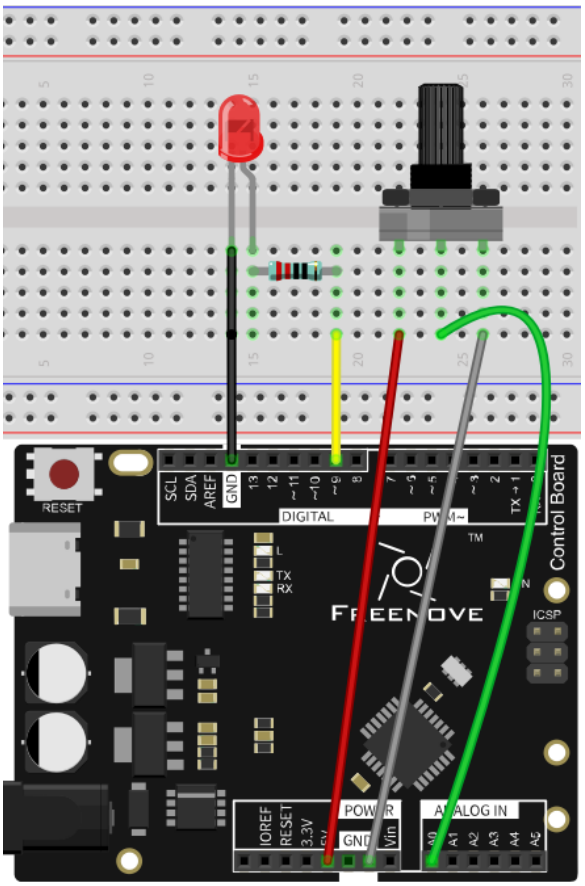
In the previous section, we have finished reading ADC value and converting it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

Component List

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

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

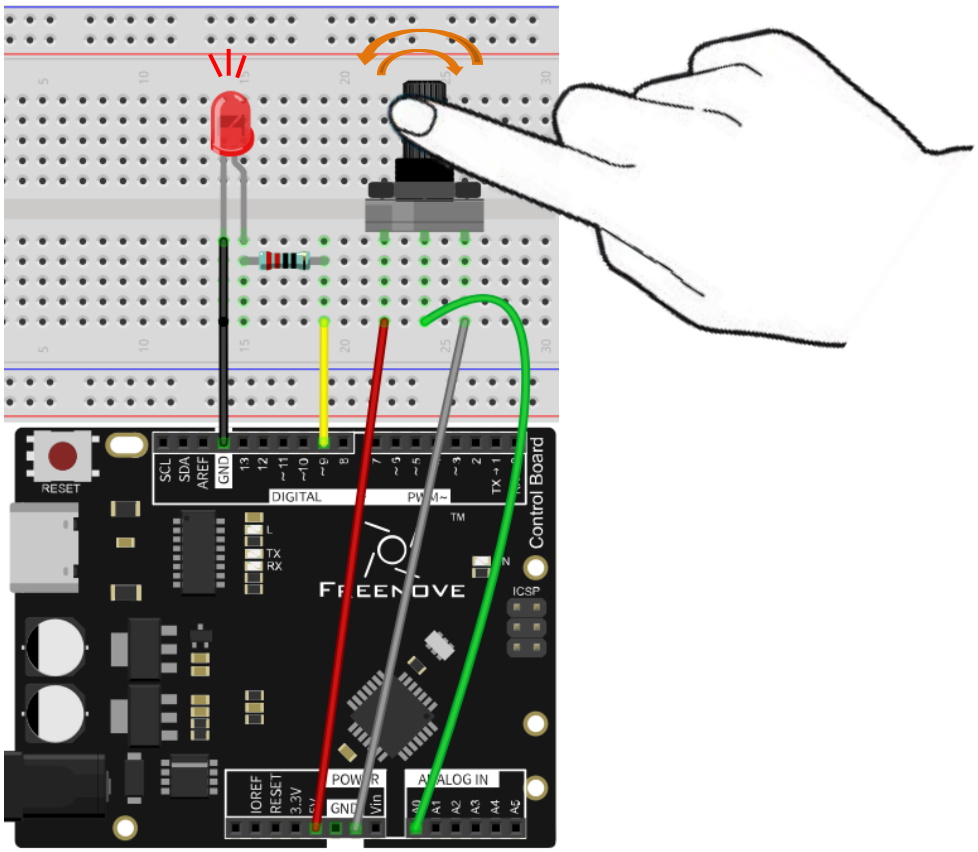
Sketch 7.2.1

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.

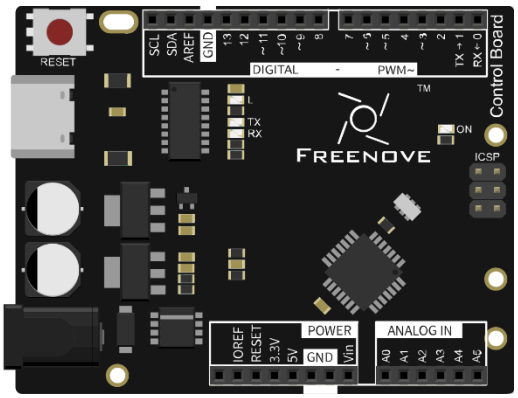
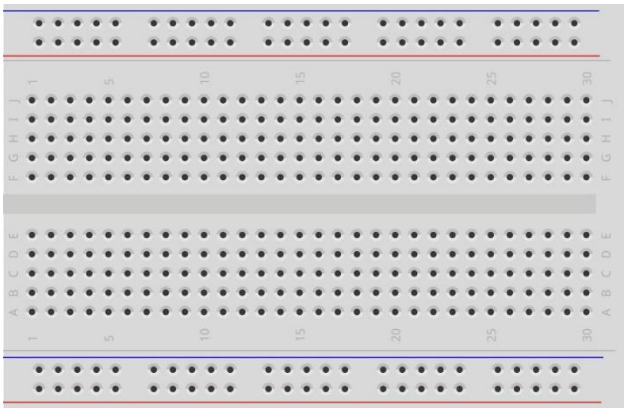
Verify and upload the code, turn the rotary potentiometer shaft, you will see the LED brightness change.



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.

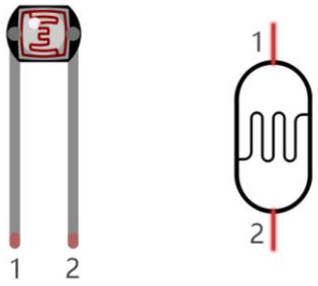
Component List

<table>
<tr><td colspan="3">Control board x1
  </td><td colspan="3">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="3"></td></tr>
</table>

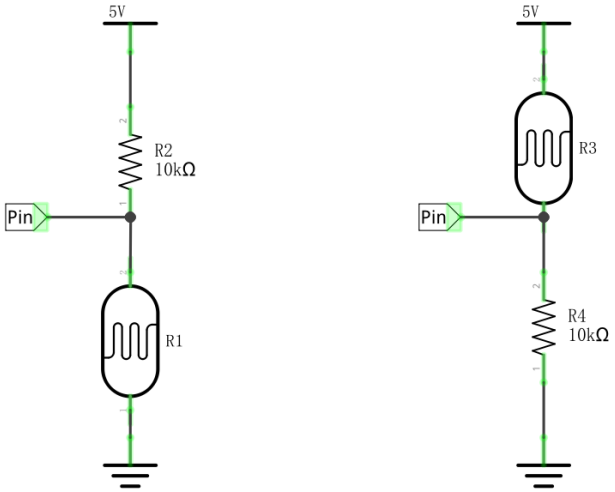
Component Knowledge

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.



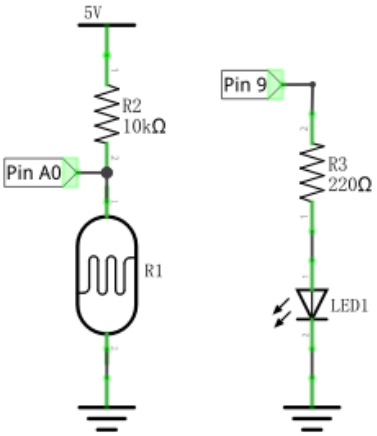
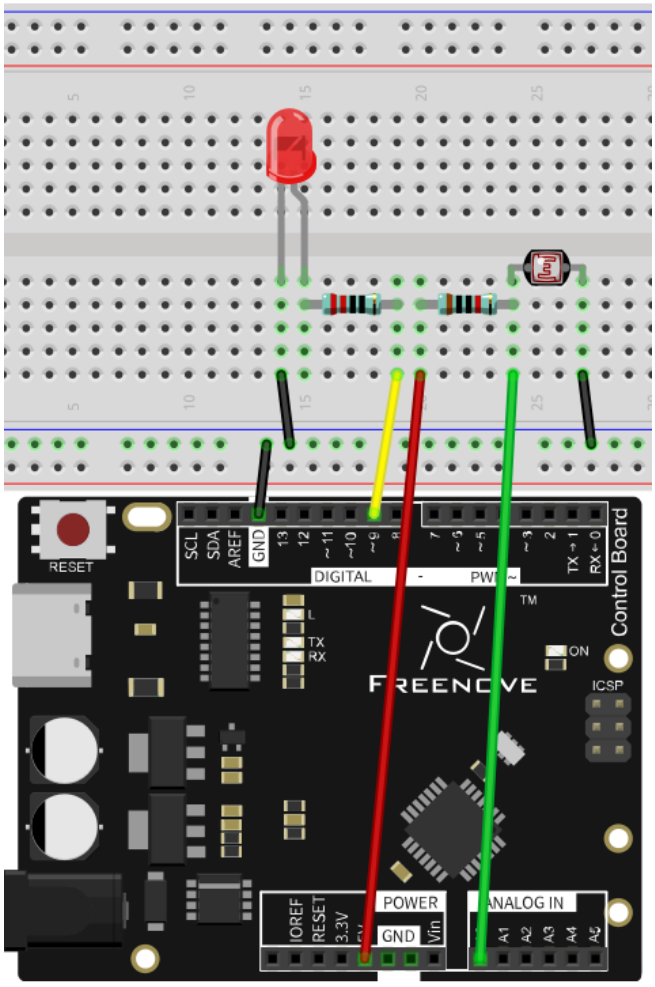
The circuit below is often used to detect the change of photoresistor's resistance value:



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

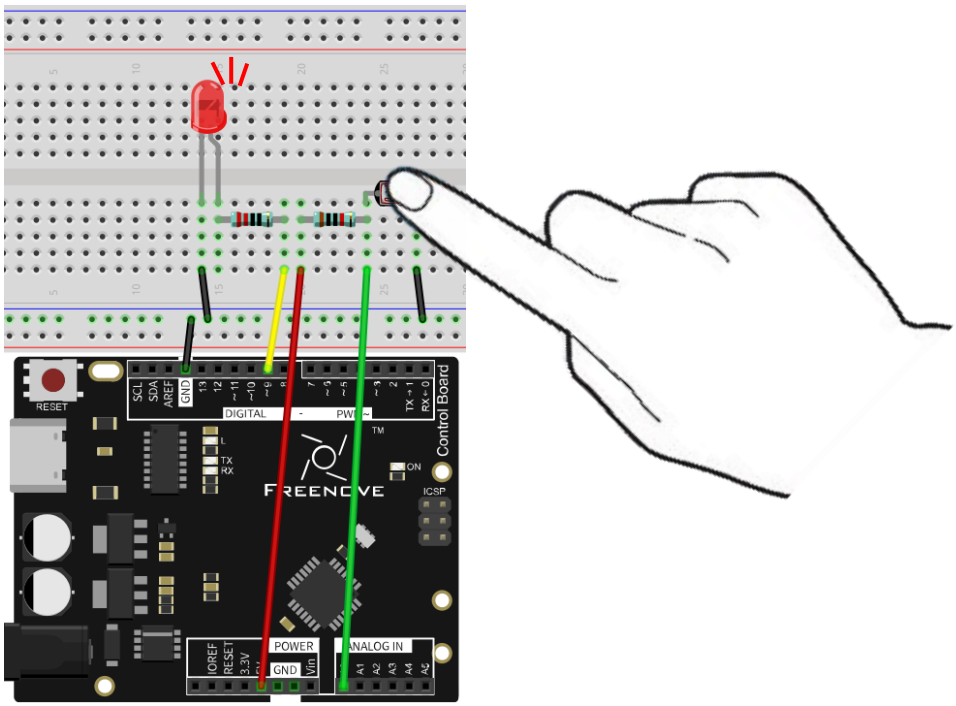
Sketch 7.3.1

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.



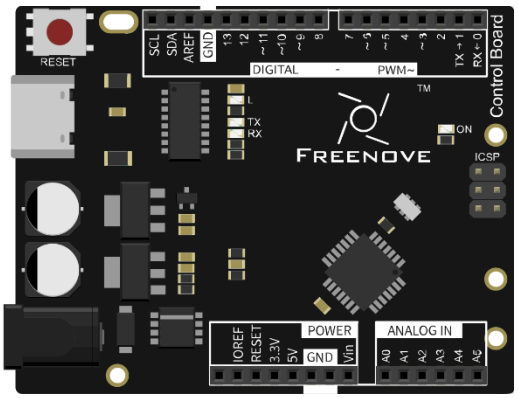
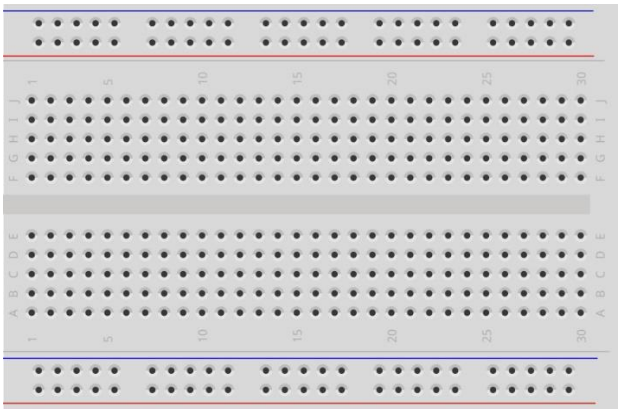
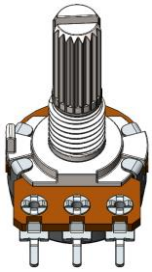
Chapter 8 RGB LED

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.

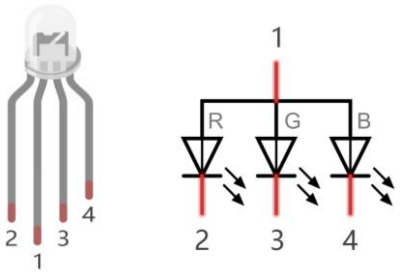
Component List

<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>

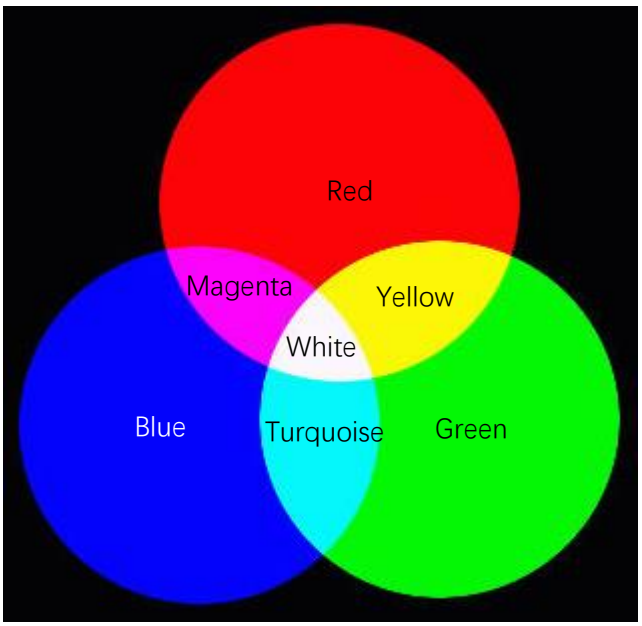
Component Knowledge

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 Cathode (+) or positive lead, the other 3 are the Anodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Anodes (2, 3 & 4) of the RGB LED



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

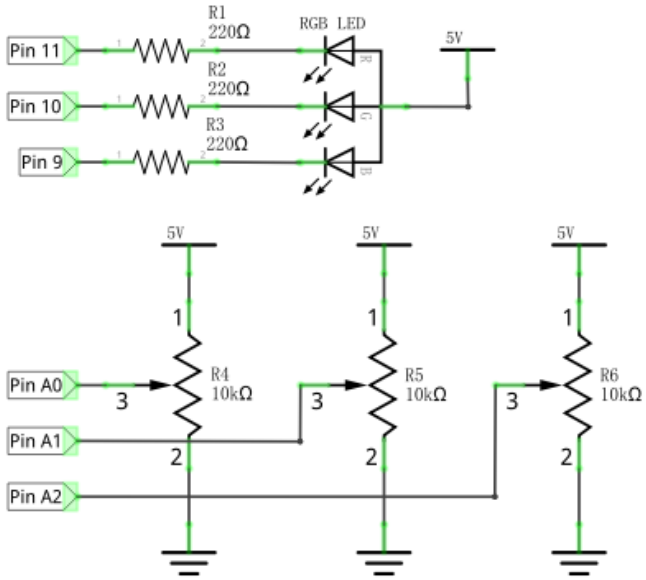


We know from the previous section that, control board controls LED to emit a total of 256(0-255) different brightness via PWM. So, through different combinations of RGB light brightness, we can create 256^3=16777216(16Million) colors.

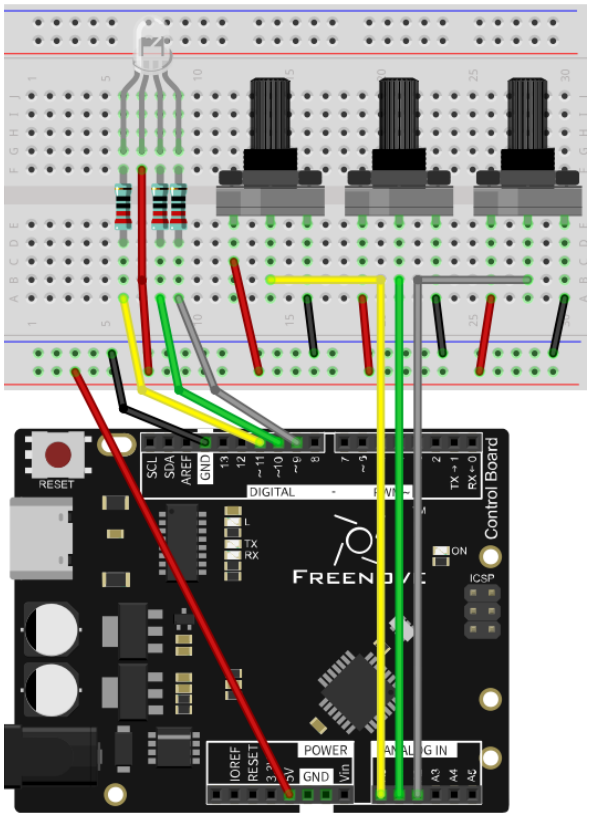
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 8.1.1

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

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

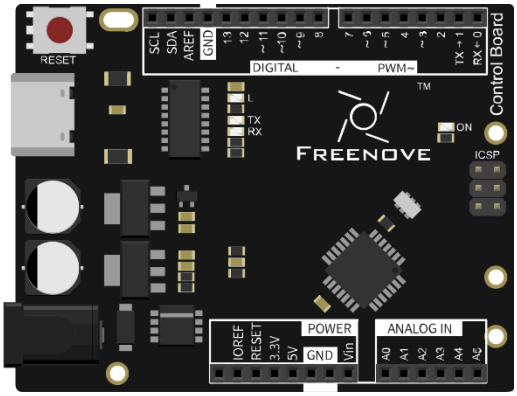
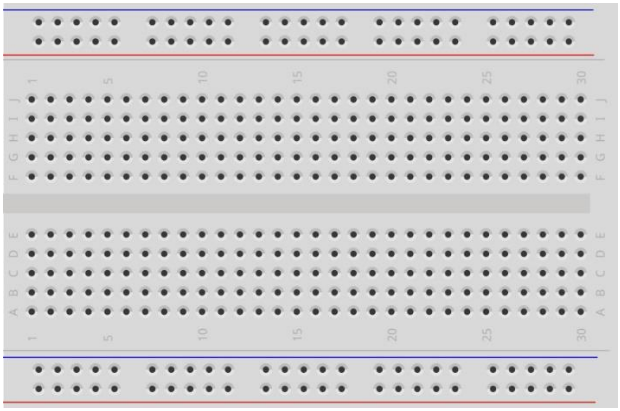
In the code, we get the voltages of three rotary potentiometers, and convert them into PWM duty cycle to control the three LEDs of the RGB LED to emit light with different brightness.

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.

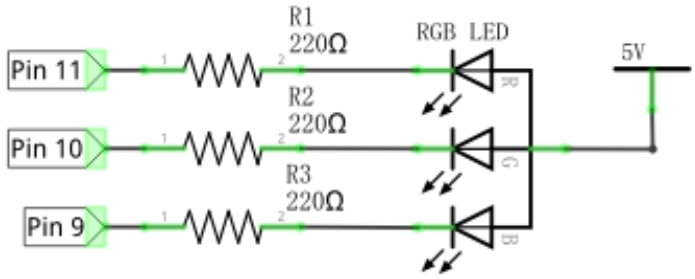
Component List

<table>
<tr><td>Control board x1
  </td><td>Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>RGB LED x1
  </td><td>Resistor 220Ω x3
  </td></tr>
<tr><td>Jumper M/M x4
  </td><td></td><td></td></tr>
</table>

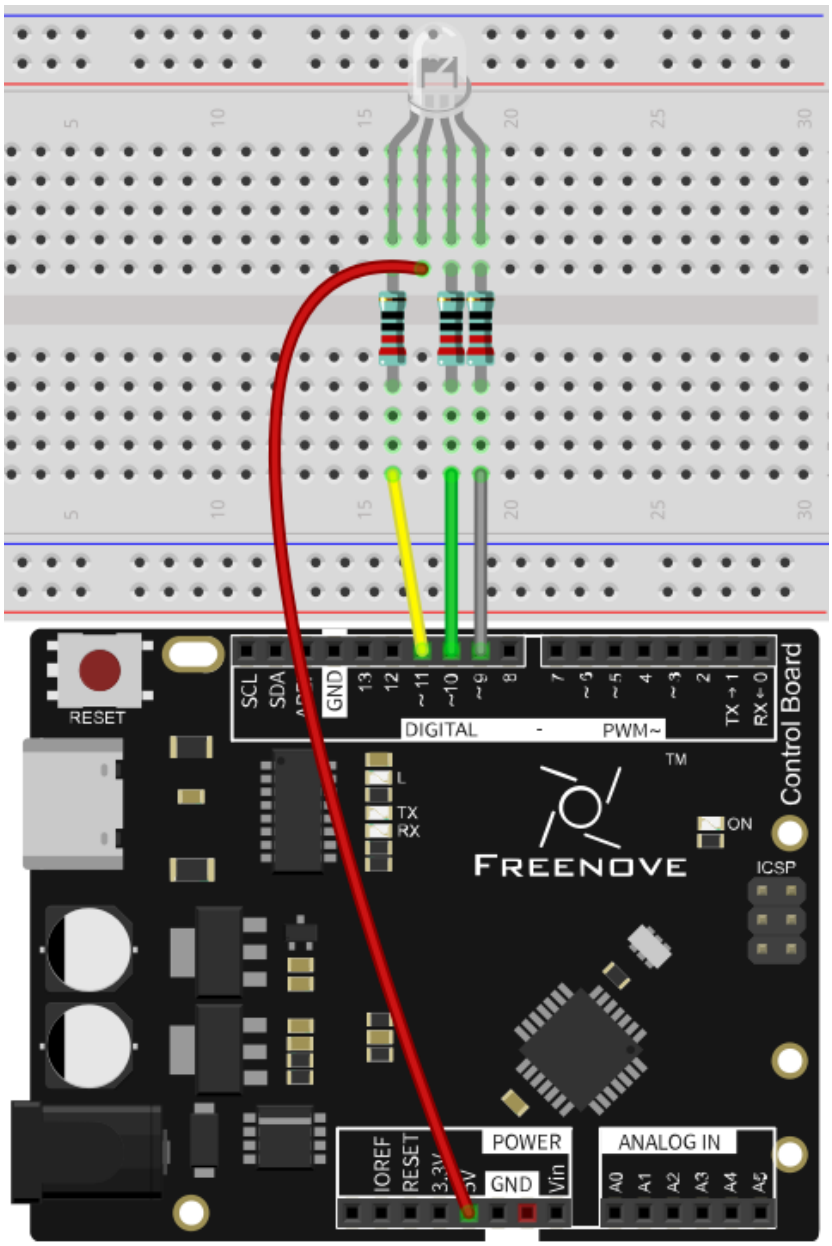
Circuit

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.

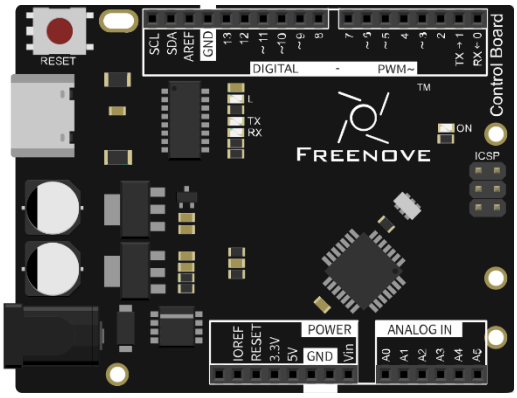
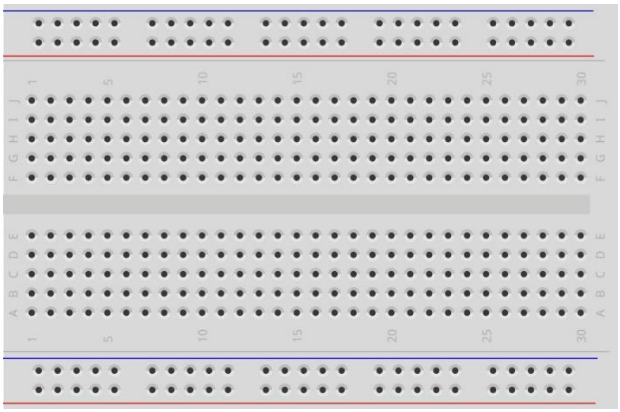
Chapter 9 Buzzer

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

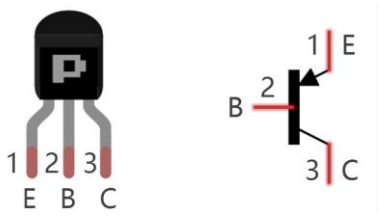
<table>
<tr><td colspan="2">Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td colspan="2">USB cable x1
  </td><td colspan="3">Jumper M/M x6
</td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor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

Component knowledge

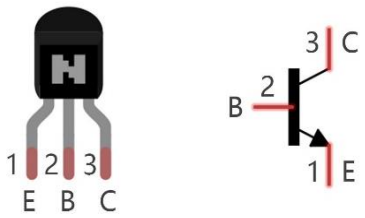
Transistor

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



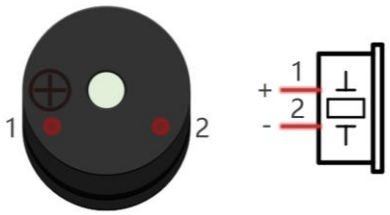
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

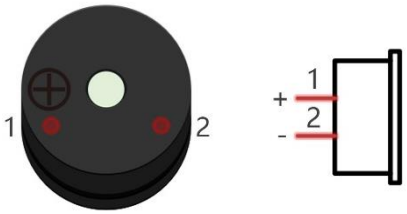
Buzzer

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?

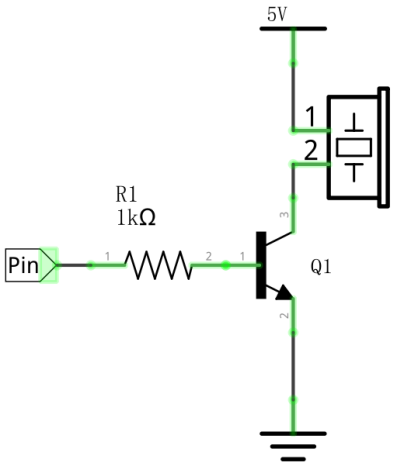
1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this 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

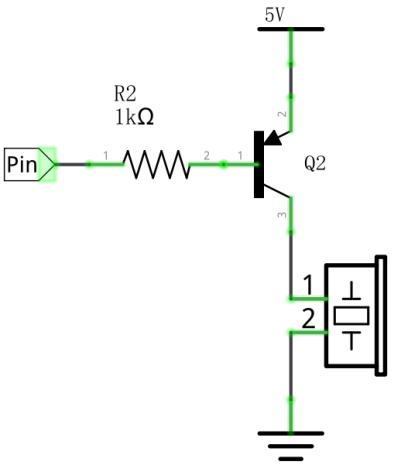
Passive buzzer bottom

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).

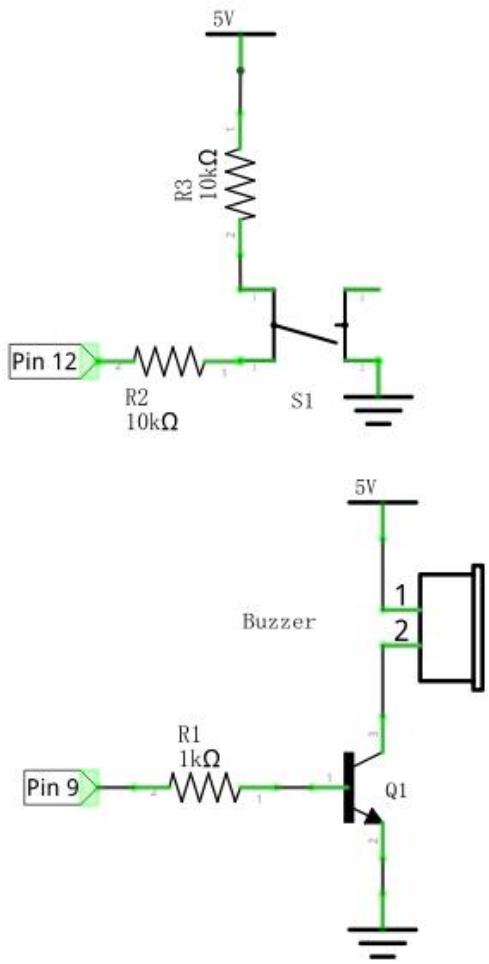
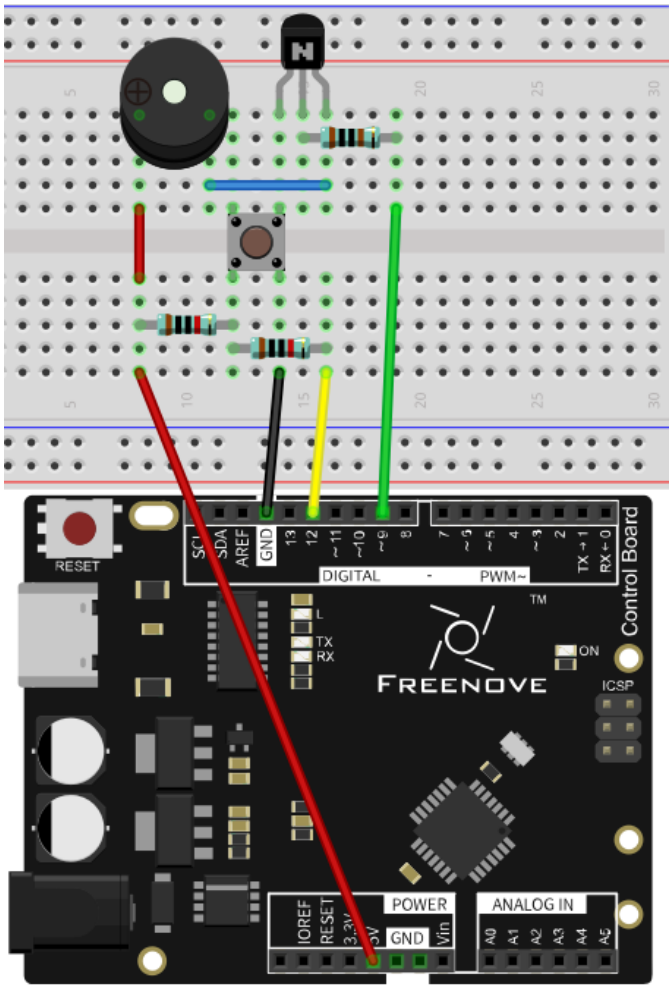


When we use a PNP transistor to drive a buzzer, we often use the following method. If control board pin outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If control board pin outputs high level, no current flows through R1, the transistor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

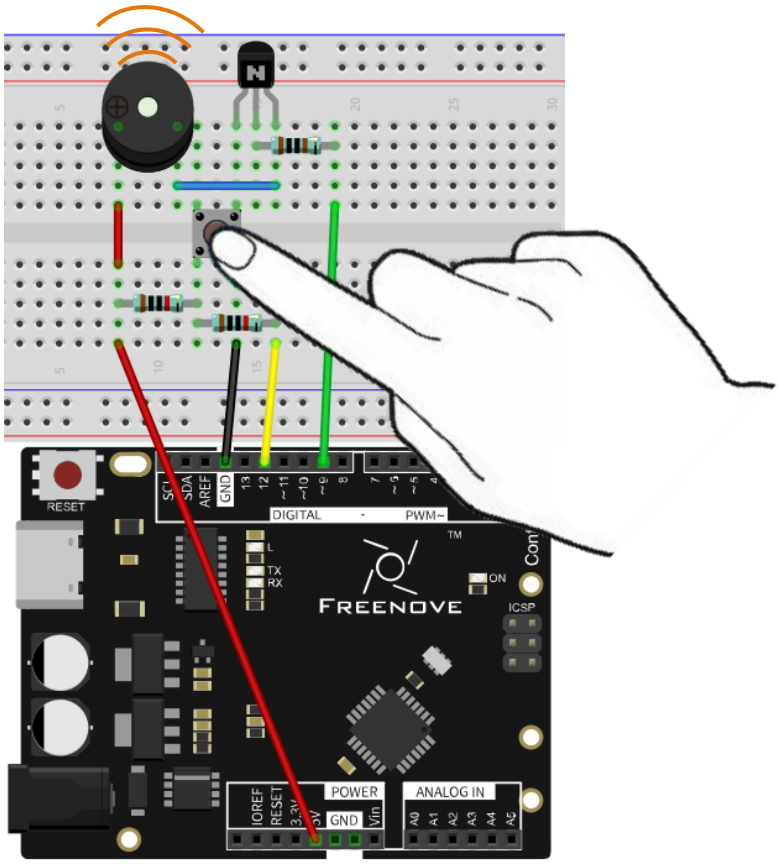
Sketch 9.1.1

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.

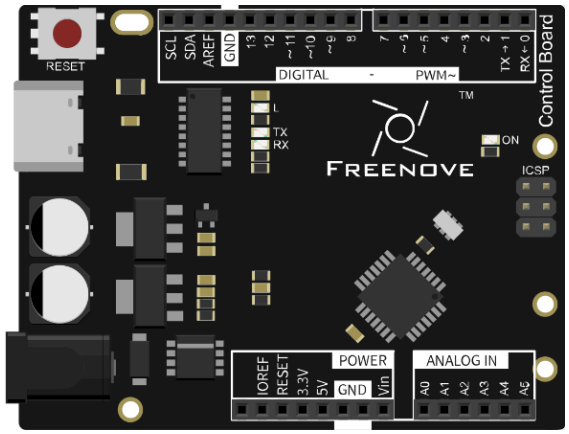
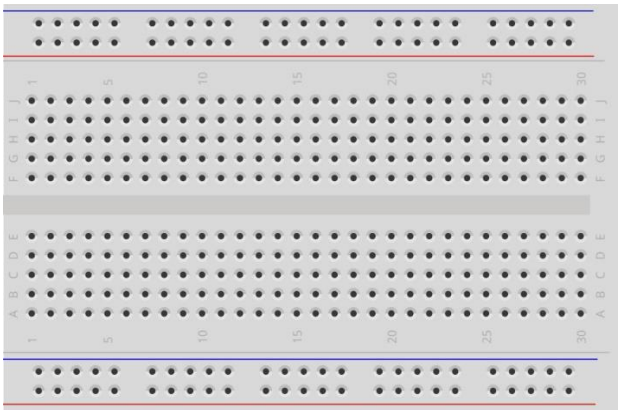
Verify and upload the code, press the push button, and the active buzzer will make a sound.



Project 9.2 Passive Buzzer

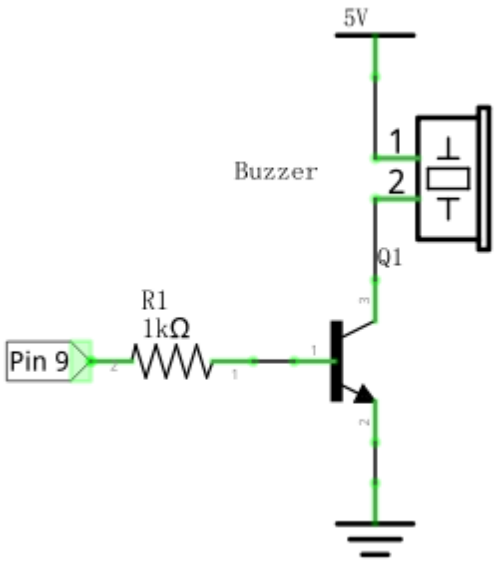
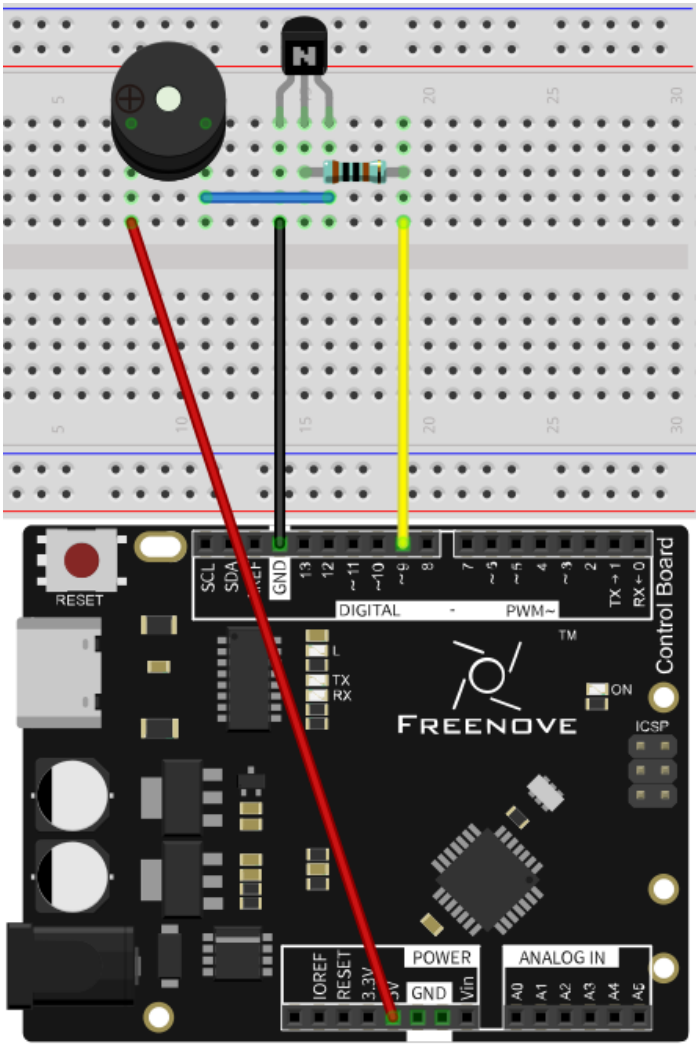
In the previous section, we have finished using transistor to drive an active buzzer to beep. Now, we will try to use a passive buzzer to make a sound with different frequency.

Component List

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

Circuit

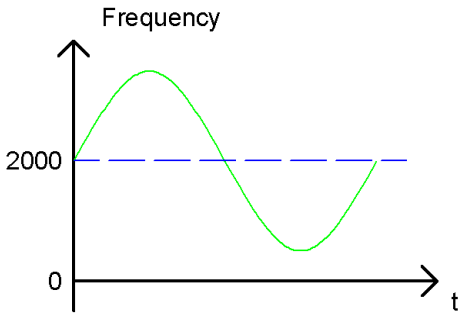
Use pin 9 port of control board to drive a passive buzzer.

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.

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

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

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

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

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

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

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

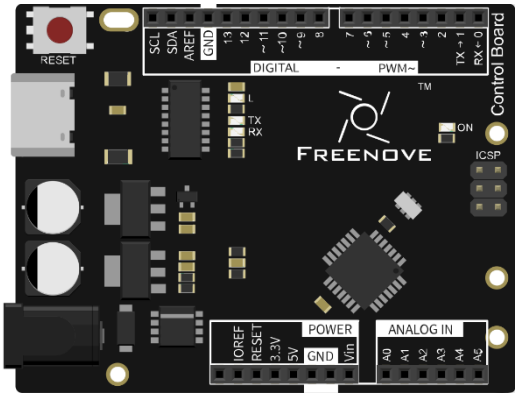
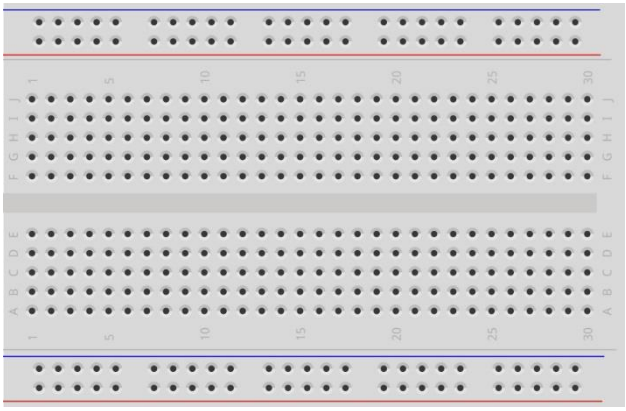
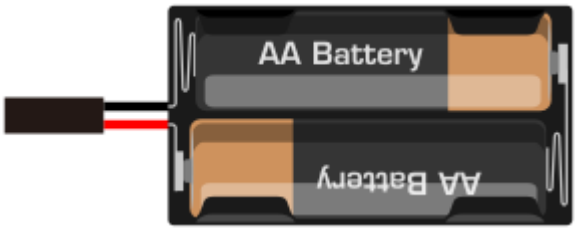
Chapter 10 Motor

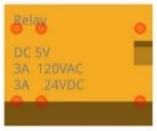
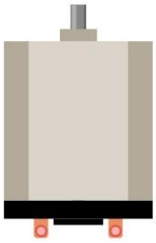
Earlier, we have done a series of interesting projects with control board and basic electronic components. Now, let us study some movable electronic modules. In this chapter, we will learn to control a motor.

Project 10.1 Control Motor by Relay

First, use relay to control a Motor.

Component List

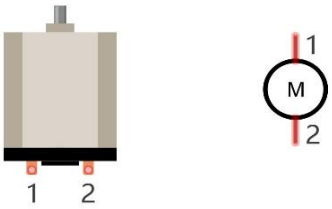
Control board x1 	Breadboard x1 		
USB cable x1 	Jumper M/M x8 		
AA Battery holder x1 (Need AA battery x2) 	Resistor 10kΩ x2 	Resistor 1kΩ x1 	Resistor 220Ω x1 

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

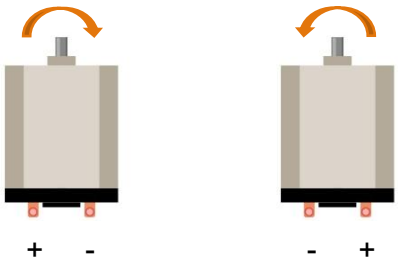
Component Knowledge

DC Motor

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



When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.



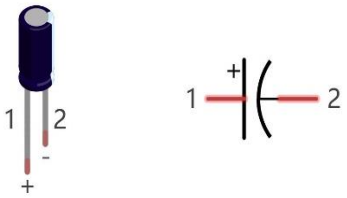
Capacitor

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

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

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

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

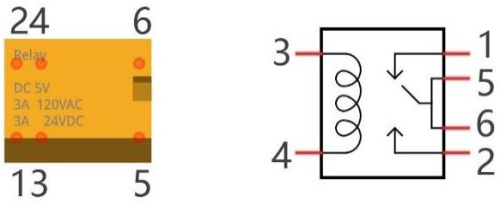


When the motor rotates, it will generate noise. As the contact of coil connects and disconnects the electrode constantly, it will cause the supply voltage unstable. Thus, a small capacitor is often connected to motor to reduce the impact on power supply from motor.

Relay

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

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



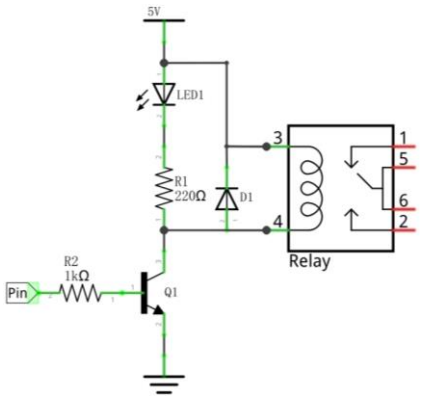
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

Inductor

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

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

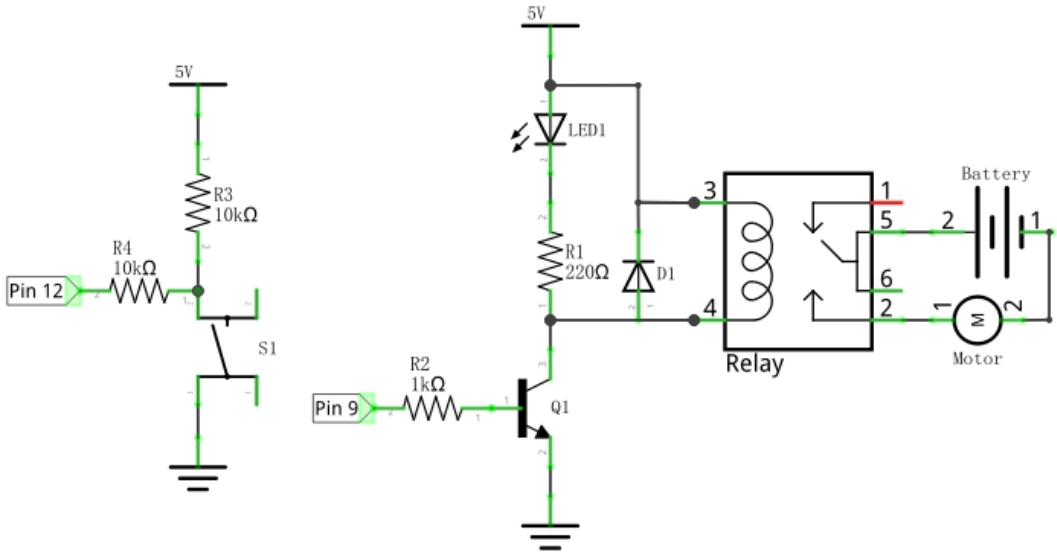
The circuit for a relay is as follows: The coil of relay can be equivalent to an Inductor, when a transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes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



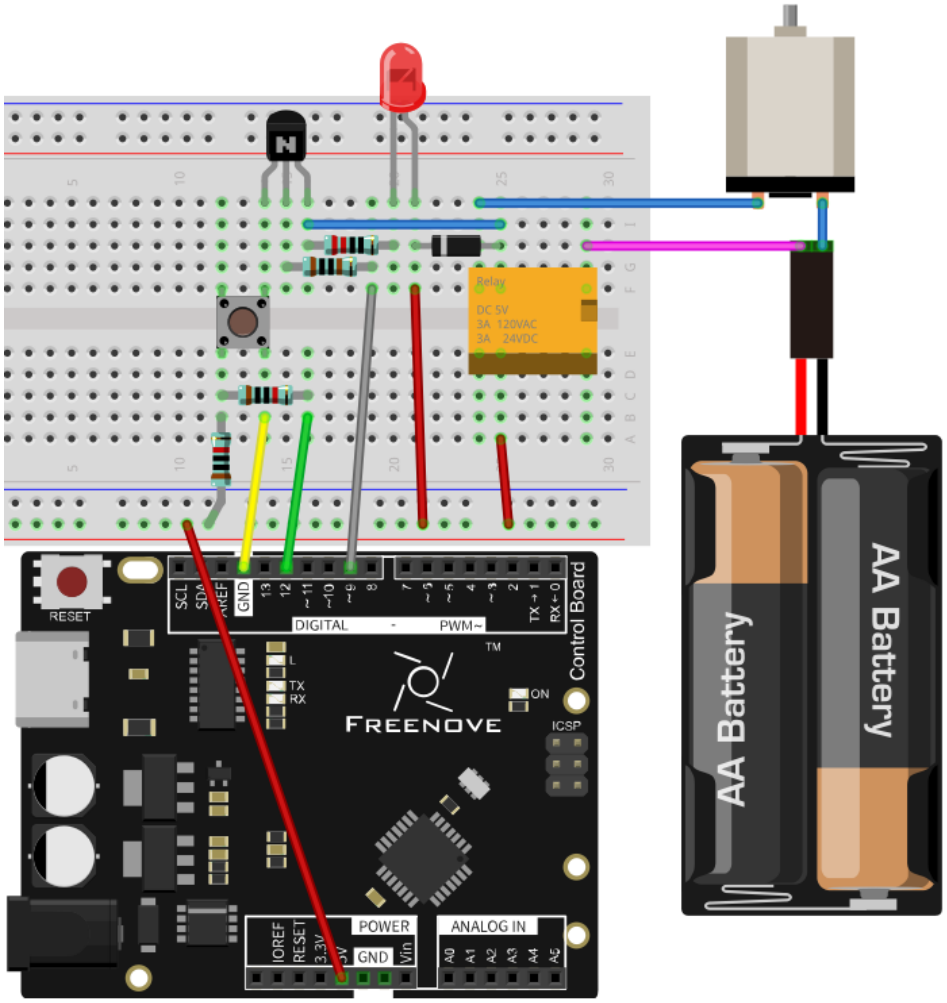
Circuit

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

Schematic diagram



Hardware connections



Sketch

Sketch 10.1.1

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

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

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

void setup() {
  pinMode(buttonPin, INPUT);  // Set push button pin into input mode
```

```
11    pinMode(relayPin, OUTPUT);   // Set relay pin into output mode
12    digitalWrite(relayPin, relayState); // Set the initial state of relay into "off"
13    Serial.begin(9600);                 // Initialize serial port,and set baud rate to 9600
14  }
15
16  void loop() {
17    int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
18    // If button pin state has changed, record the time point
19    if (nowButtonState != lastButtonState) {
20      lastChangeTime = millis();
21    }
22    // If button state changes, and stays stable for a while, then it should have skipped the bounce area
23    if (millis() - lastChangeTime > 10) {
24      if (buttonState != nowButtonState) {  // Confirm button state has changed
25        buttonState = nowButtonState;
26        if (buttonState == LOW) {      // Low level indicates the button is pressed
27          relayState = !relayState;            // Reverse relay state
28          digitalWrite(relayPin, relayState); // Update relay state
29          Serial.println("Button is Pressed!");
30        }
31        else {                         // High level indicates the button is released
32          Serial.println("Button is Released!");
33        }
34      }
35    }
36    lastButtonState = nowButtonState; // Save the state of last button
37  }
```

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

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

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

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

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

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

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

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

```
16  void loop() {
17    int nowButtonState = digitalRead(buttonPin); // Read current state of button pin
18    // If the state of button pin has changed, record the time point
```

```
19    if (nowButtonState != lastButtonState) {
20      lastChangeTime = millis();
21    }
...   ...
36    lastButtonState = nowButtonState; // Save the state of last button
37  }
```

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

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

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

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

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

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

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

millis()

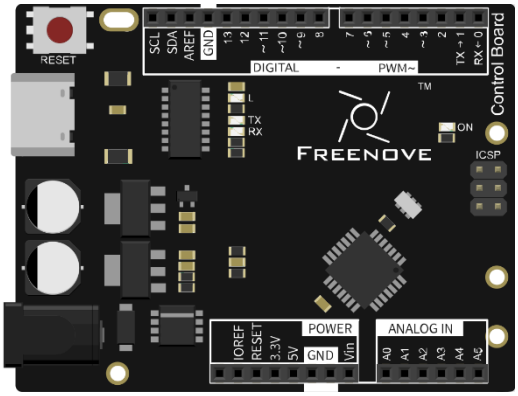
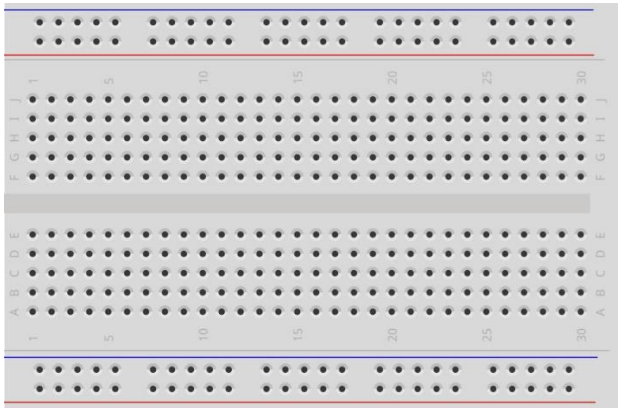
Returns the number of milliseconds since the control board began running the current program.

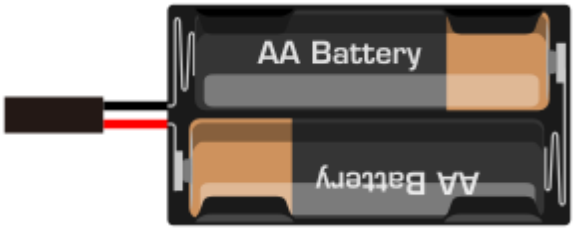
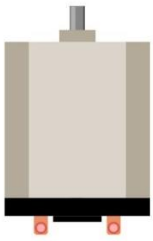
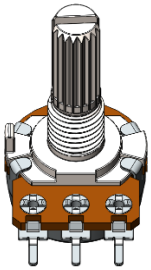
Verify and upload the code, every time you press the push button, the state of relay and motor changes once.

Project 10.2 Control Motor with L293D

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

Component List

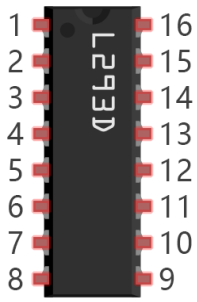
<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td colspan="3">Jumper M/M x10

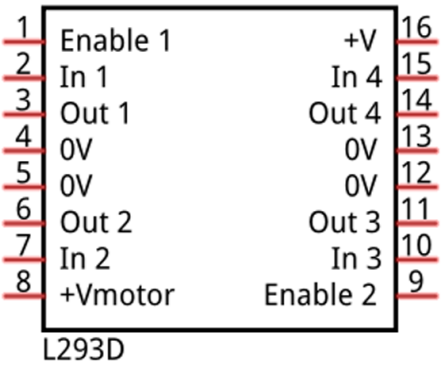
Jumper F/M x2
  </td></tr>
<tr><td>AA Battery holder x1
  </td><td>Motor x1
  </td><td>L293D x1
  </td><td>Rotary potentiometer x1
  </td></tr>
</table>

Component Knowledge

L293D

L293D is an IC chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a unidirectional DC motor with 4 ports or a bi-directional DC motor with 2 ports or a stepper motor (stepper motors are covered later in this Tutorial).



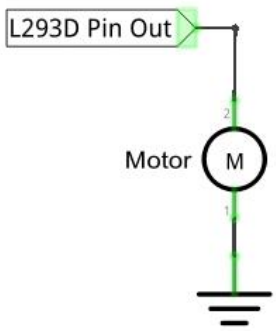


Port description of L293D module is as follows:

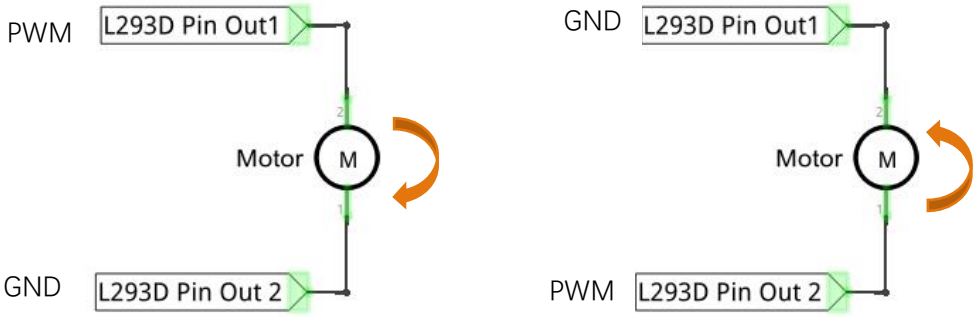
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 4.5~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

For more details, please see datasheet.

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



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

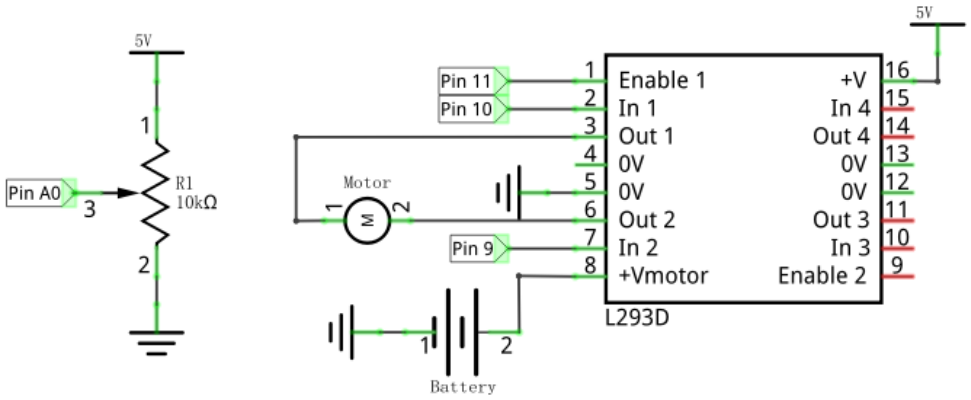


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting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

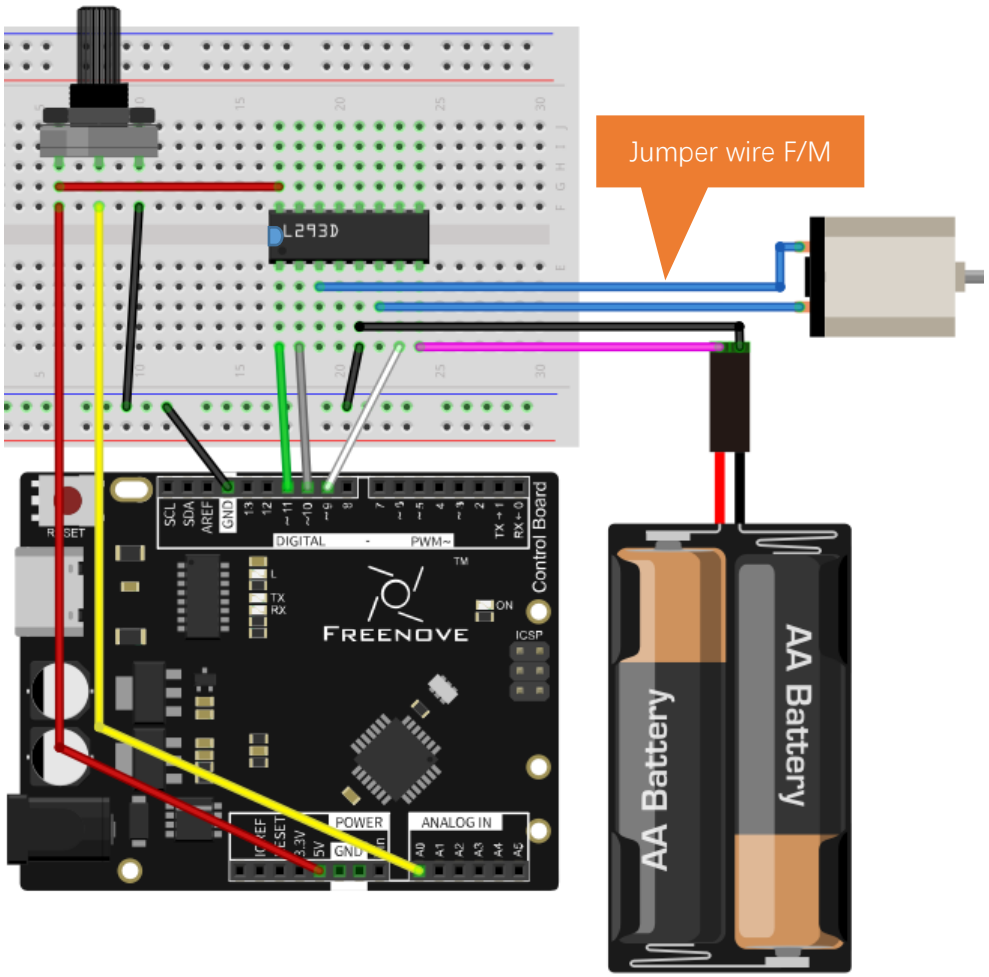
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 10.2.1

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

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

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

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

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

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

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

abs(x)

Computes the absolute value of a number.

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

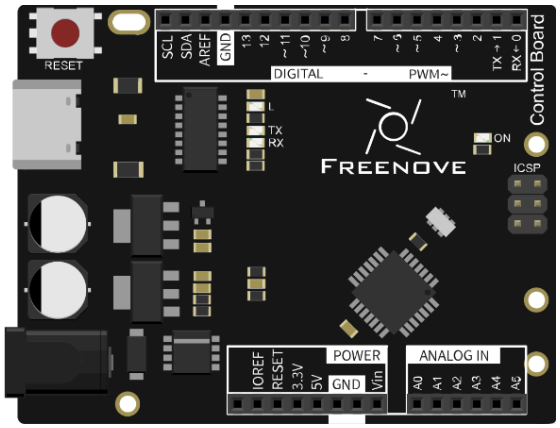
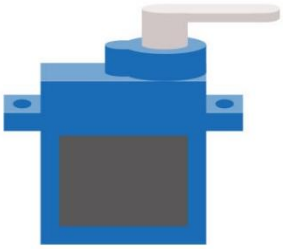
Chapter 11 Servo

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

Project 11.1 Servo Sweep

First, let's get the servo to rotate.

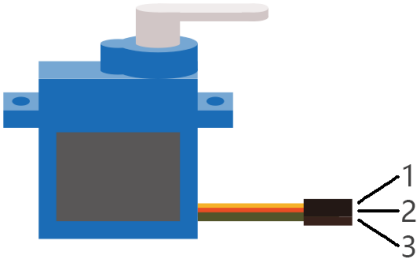
Component List

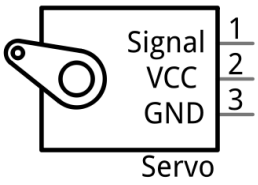
Control board x1 	USB cable x1 
	Jumper M/M x3 
	Servo x1 

Component Knowledge

Servo

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





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

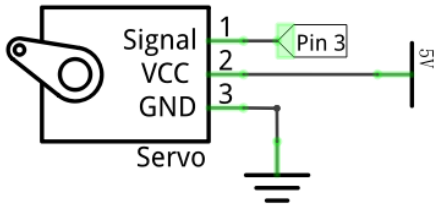
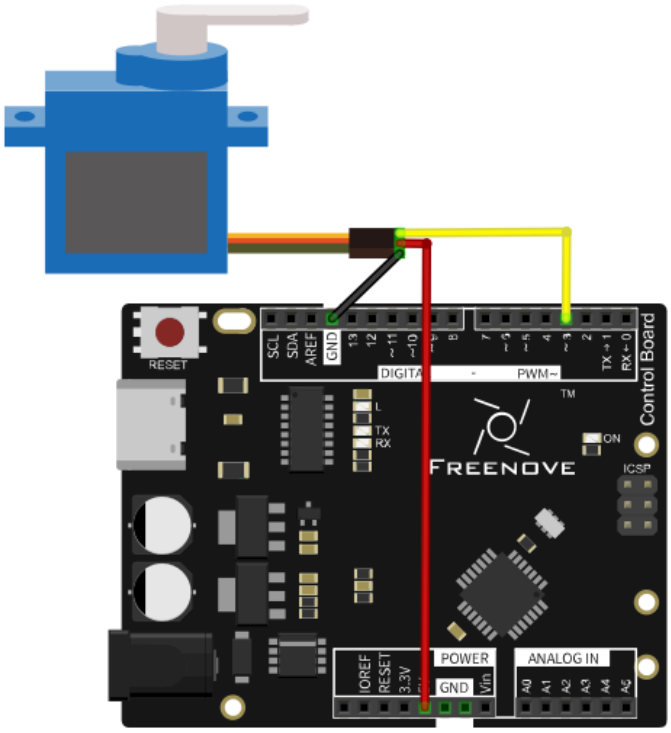
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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

Circuit

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

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 11.1.1

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

```
#include <Servo.h>

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

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

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

```

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

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

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

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

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

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

Servo Class

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

Servo myservo;

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

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

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

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

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

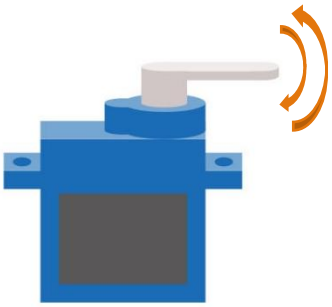
```
9    myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
```

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

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

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

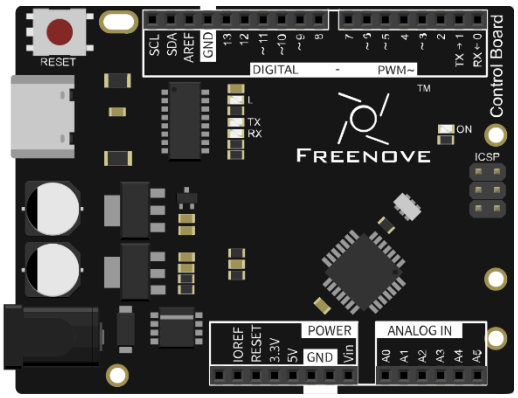
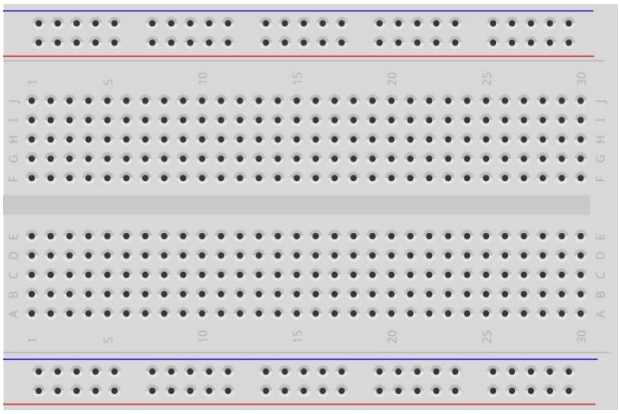
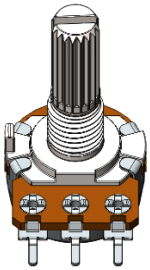
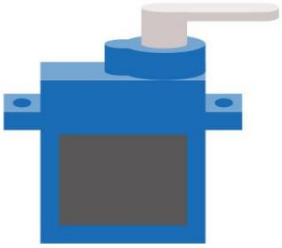
Verify and upload the code, the servo starts to sweep continuously.



Project 11.2 Control Servo with Potentiometer

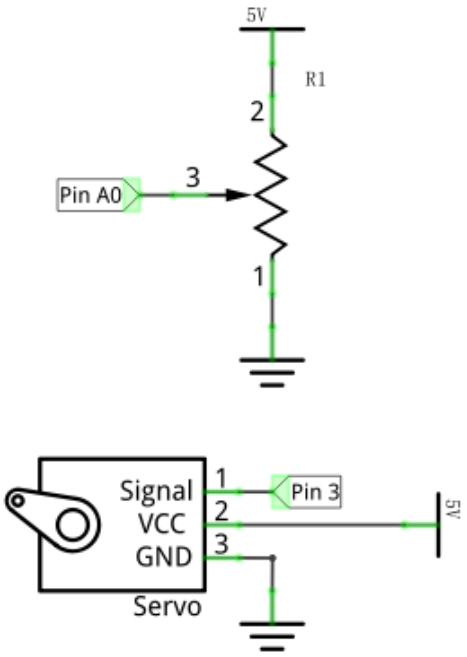
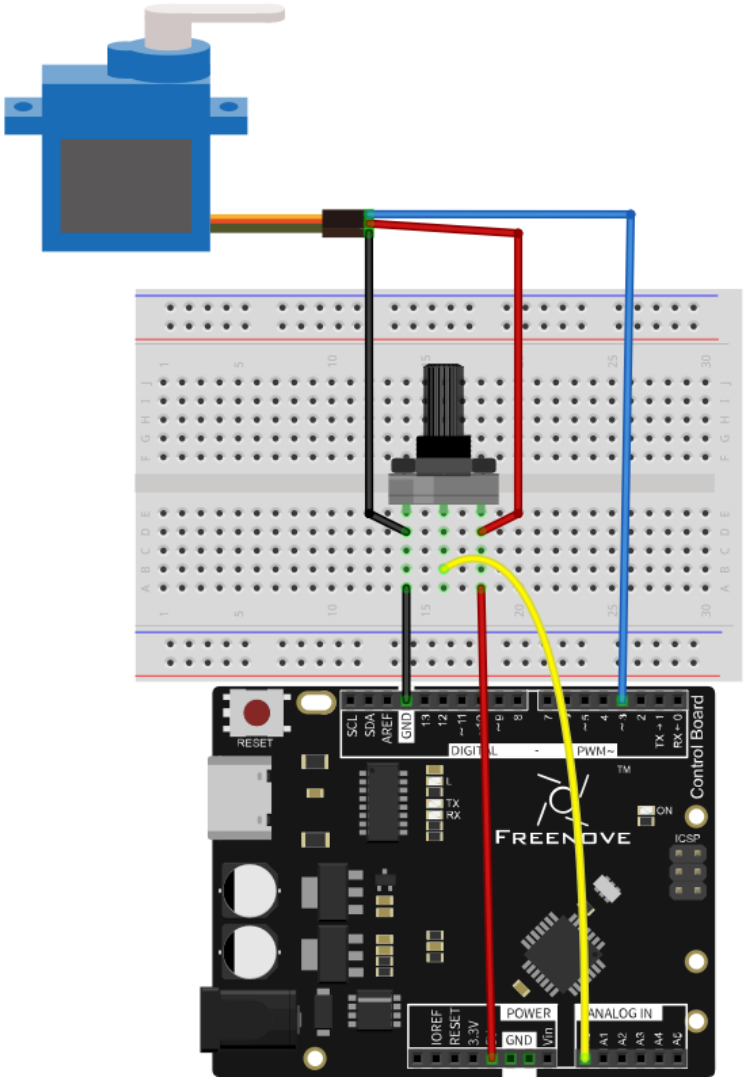
In the previous section, we've made the servo sweep continuously. Now, we will use a potentiometer to control the servo's angle.

Component List

Control board x1 	Breadboard x1 	
USB cable x1 	Rotary potentiometer x1 	Servo x1 
Jumper M/M x6 		

Circuit

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

Schematic diagram	Hardware connection
	

Sketch

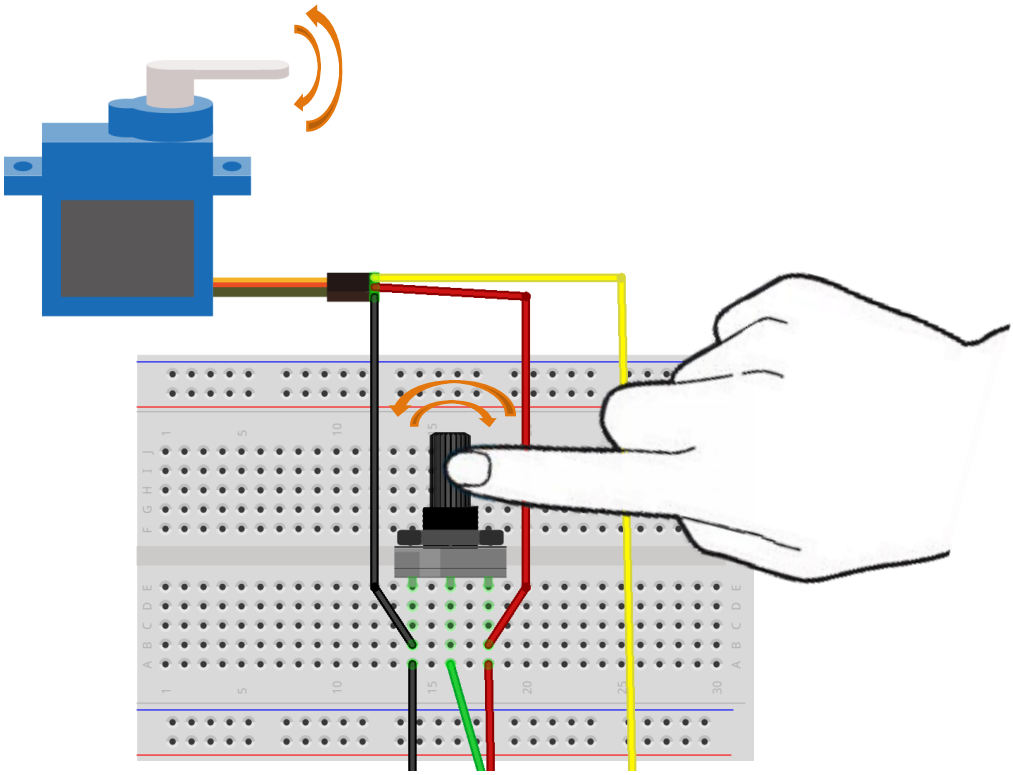
Sketch 11.2.1

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

```
#include <Servo.h>

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

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

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

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

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

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



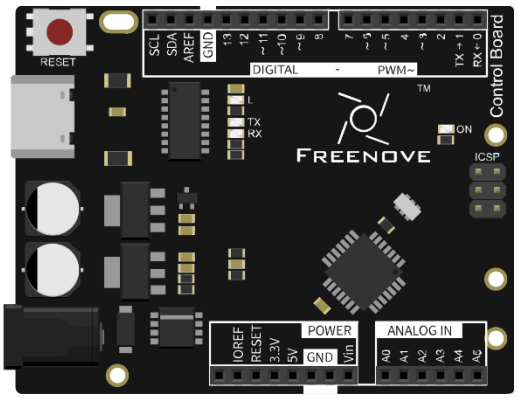
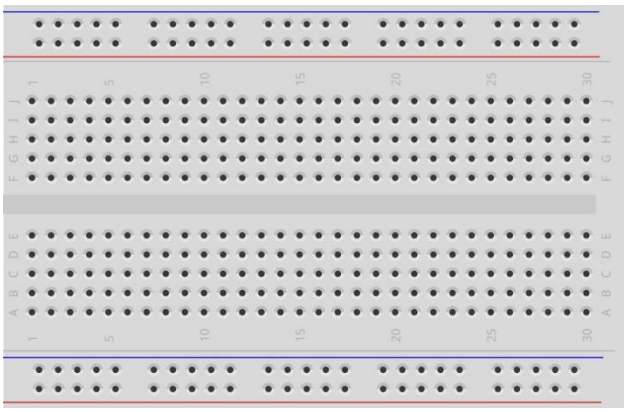
Chapter 12 Temperature Sensor

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

Project 12.1 Detect the Temperature

We will use a thermistor to detect the ambient temperature.

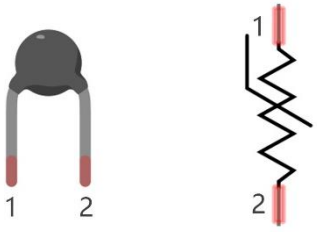
Component List

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;

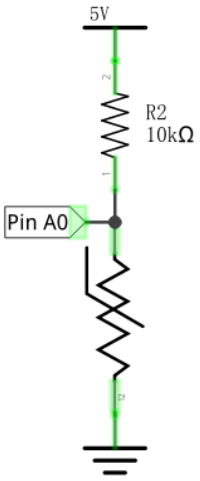
EXP[n] is nth power of e;

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.

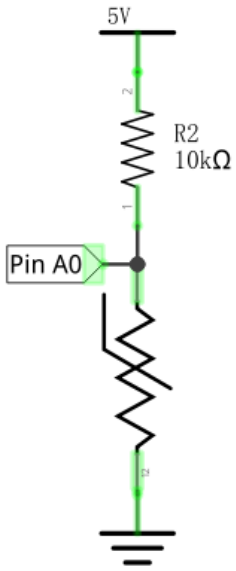
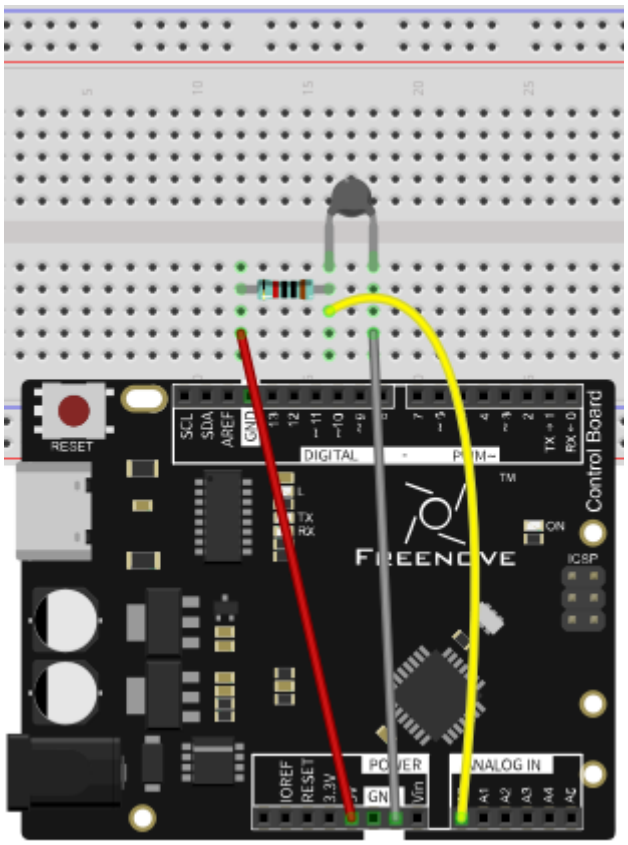
The circuit connection method of the thermistor is similar to photoresistor, as the following:



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

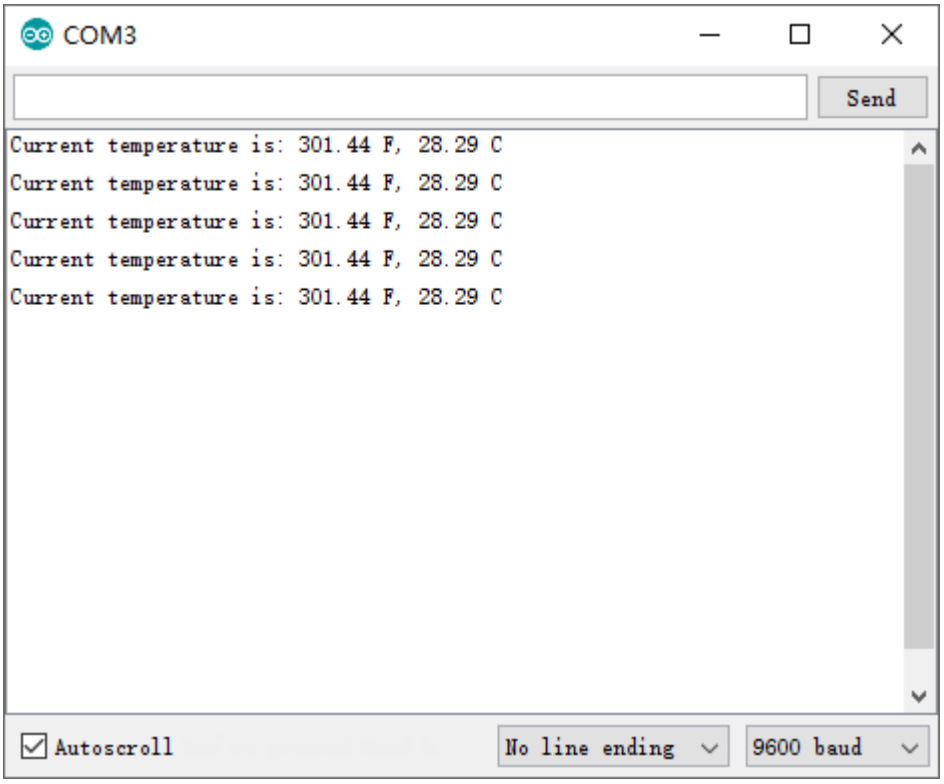
Now, write the code to detect the voltage value of thermistor, calculate the temperature value, and send it to Serial Monitor.

```
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.



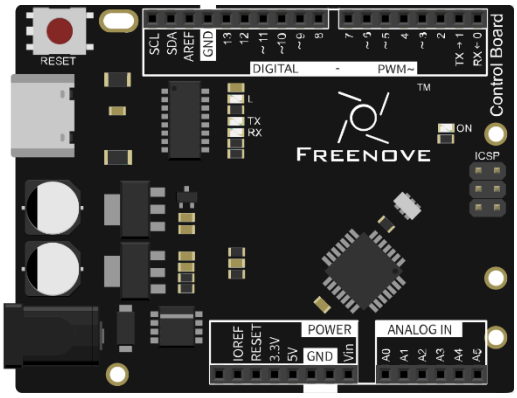
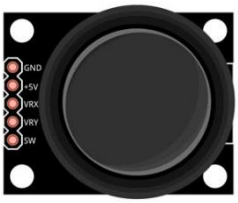
Chapter 13 Joystick

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

Project 13.1 Joystick

We will use the serial port to get Joystick data.

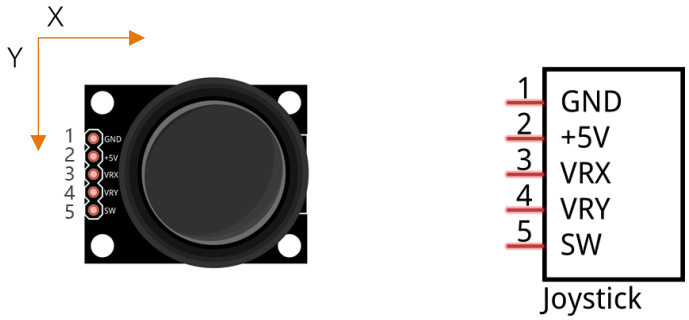
Component List

Control board x1 	Joystick x1 
USB cable x1 	Jumper F/M x5 

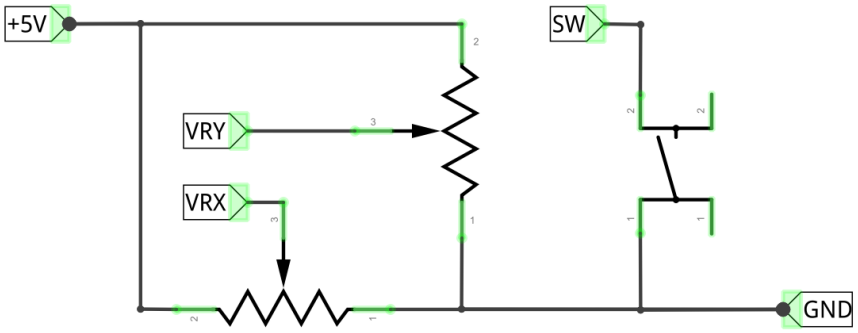
Component Knowledge

Joystick

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

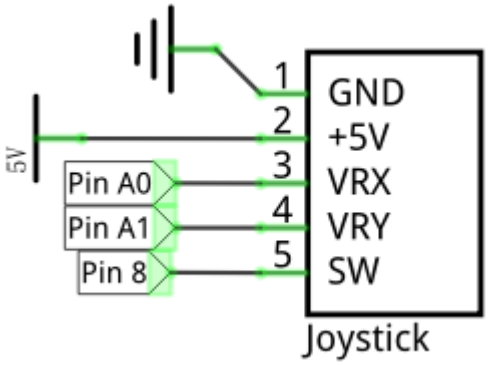
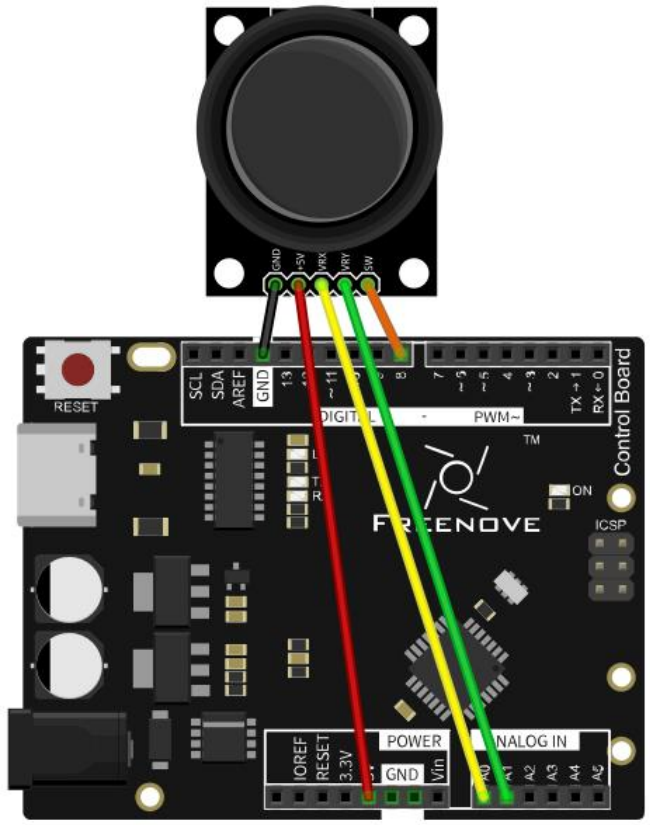


This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



Circuit

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

Schematic diagram	Hardware connection
	

Sketch

Sketch 13.1.1

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

```
1  int xAxisPin = 0;           // define X pin of Joystick
2  int yAxisPin = 1;           // define Y pin of Joystick
3  int zAxisPin = 8;           // define Z pin of Joystick
4  int xVal, yVal, zVal;       // define 3 variables to store the values of 3 direction
5
6  void setup() {
7    pinMode(zAxisPin, INPUT_PULLUP);    // initialize the port to pull-up input
8    Serial.begin(9600);         // initialize the serial port with baud rate 9600
9  }
10
```

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

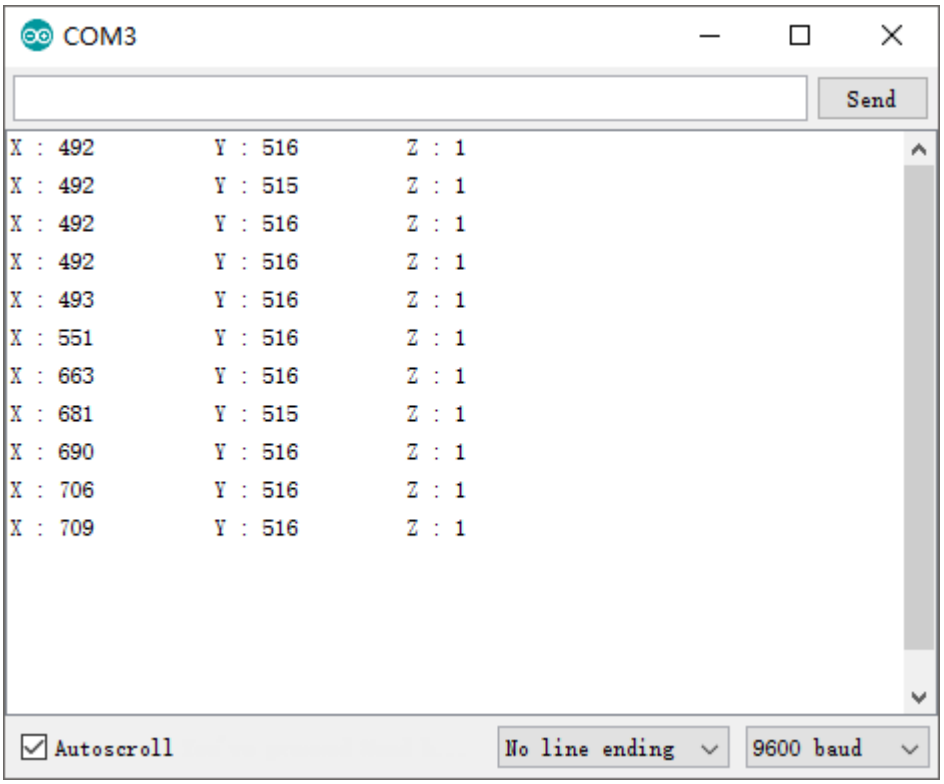
In the code, we get the ADC value of pin A0, A1 and the state of button, and then sent the data to serial port.

INPUT_PULLUP

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

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

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



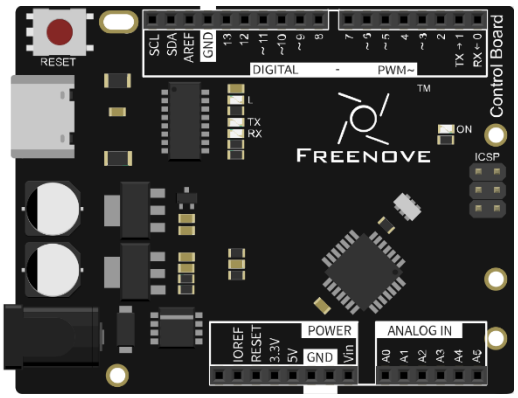
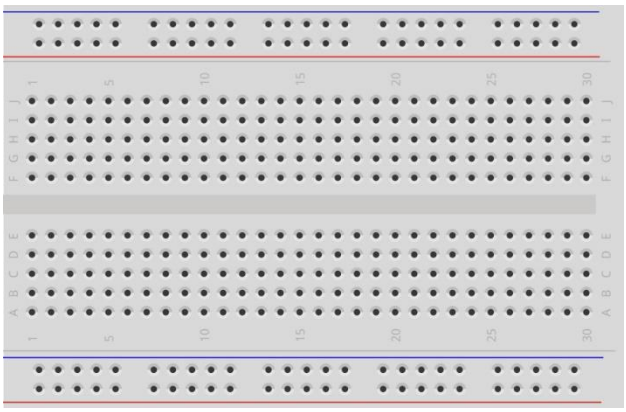
Chapter 14 Acceleration sensor

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

Project 14.1 Acceleration Detection

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

Component List

Control board x1 	Breadboard x1 
USB cable x1 	MPU6050 module x1 
Jumper M/M x4 	

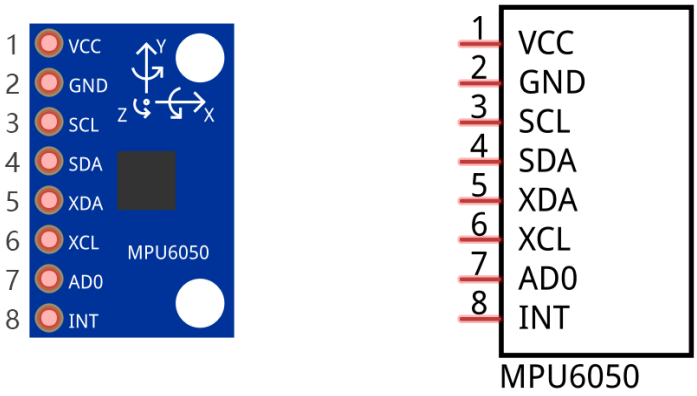
Component Knowledge

I2C communication

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

MPU6050

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



The port description of the MPU6050 module is as follows:

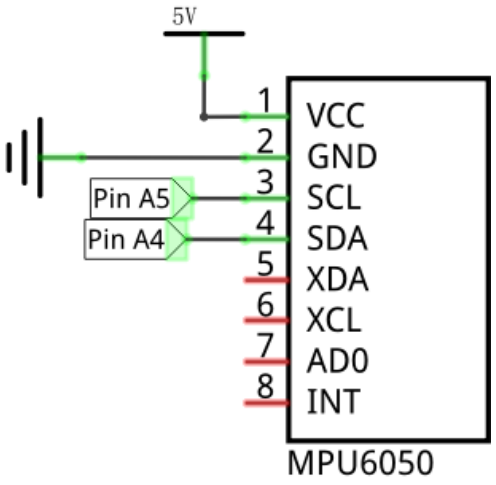
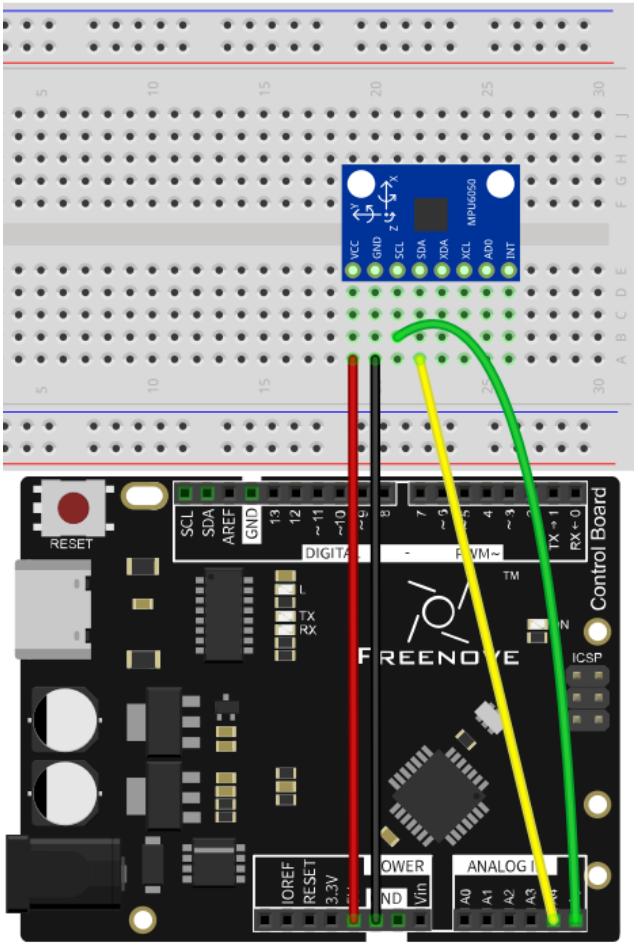
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication data pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to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

For more detail, please refer to datasheet.

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

Circuit

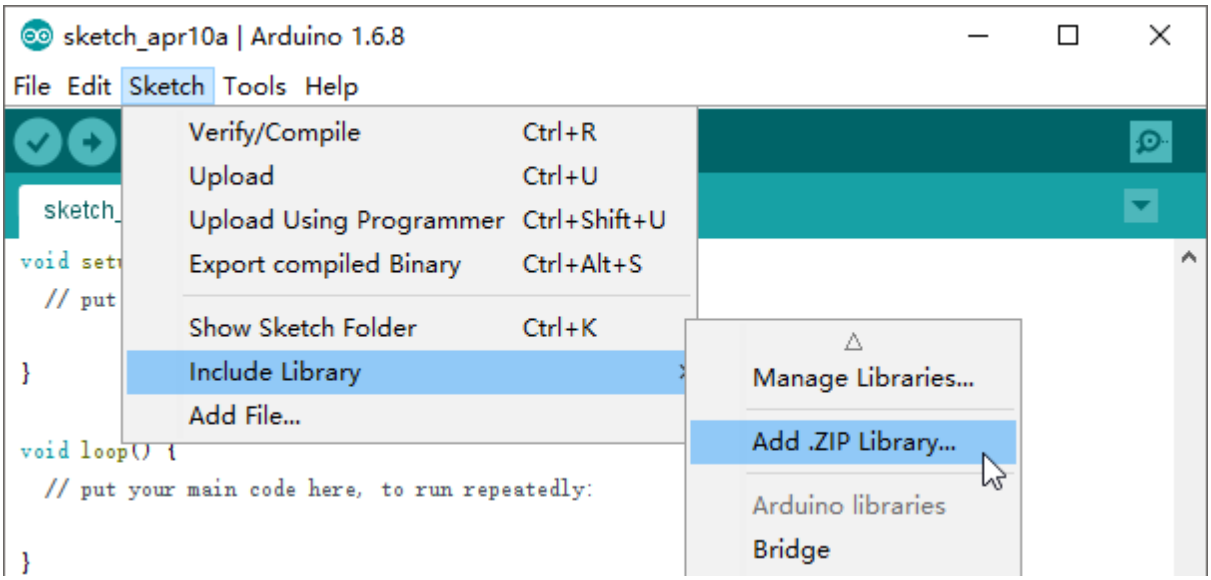
Use pin A4/SDA, pin A5/SCL port on the control board to communicate with MPU6050 module.

Schematic diagram	Hardware connection
	

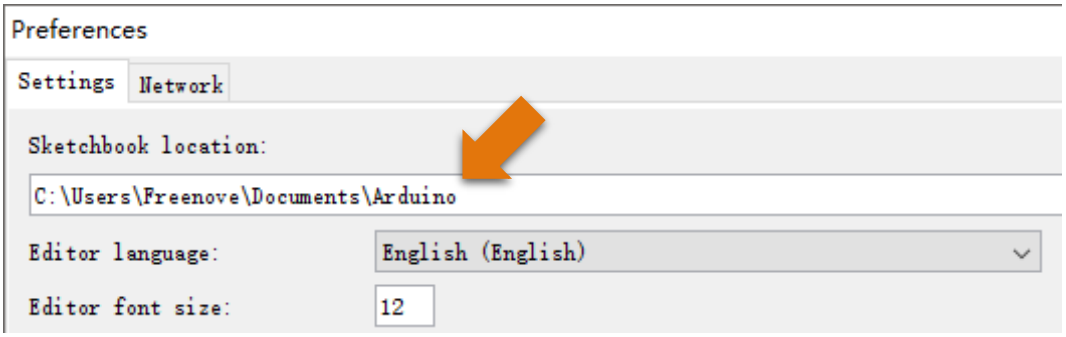
Sketch

Sketch 14.1.1

Library is a collection of code. We can use code provided by libraries to make programming simple.
Click "Add .ZIP Library..." and then find I2Cdev.zip and MPU6050.zip in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). These libraries make it easy to use MPU6050 module.



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



Now write sketch to communicate with the MPU6050 module and send the captured data to Serial Monitor window.

```
1  // Reference the library to be used by MPU6050
2  #include "Wire.h"
3  #include "I2Cdev.h"
4  #include "MPU6050.h"
5
6  MPU6050 accelgyro;          // Construct a MPU6050 object using the default address
7  int16_t ax, ay, az;         // define acceleration values of 3 axes
8  int16_t gx, gy, gz;         // Define variables to save the values in 3 axes of gyroscope
9  #define LED_PIN 13          // the number of the LED pin
10 bool blinkState = false;    // Define a variable to save the state of LED
11
```

```
12  void setup() {
13    Serial.begin(9600);       // initialize the serial port, and baud rate is set to 9600
14    Serial.println("Initializing I2C devices...");
15    Wire.begin();             // initialize I2C
16    accelgyro.initialize();   // initialize MPU6050
17    Serial.println("Testing device connections...");
18    // when you need to calibrate the gravity acceleration, you can set the offset here and
      eliminate the note
19    // accelgyro.setXAccelOffset(-1200);
20    // accelgyro.setYAccelOffset(-2500);
21    // accelgyro.setZAccelOffset(1988);
22    Serial.print("X.Y.Z offset :\t");
23    Serial.print(accelgyro.getXAccelOffset()); Serial.print("\t");
24    Serial.print(accelgyro.getYAccelOffset()); Serial.print("\t");
25    Serial.print(accelgyro.getZAccelOffset()); Serial.print("\n");
26    // initialize LED port
27    pinMode(LED_PIN, OUTPUT);
28  }
29
30  void loop() {
31    // read raw accel/gyro measurements from device
32    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
33    // display tab-separated accel/gyro x/y/z values
34    Serial.print("a/g:\t");
35    Serial.print(ax); Serial.print("\t");
36    Serial.print(ay); Serial.print("\t");
37    Serial.print(az); Serial.print("\t");
38    Serial.print(gx); Serial.print("\t\t");
39    Serial.print(gy); Serial.print("\t\t");
40    Serial.println(gz);
41    // converted acceleration unit to g and the gyroscope unit to dps (degree per second)
      according to the sensitivity
42    Serial.print("a/g:\t");
43    Serial.print((float)ax / 16384); Serial.print("g\t");
44    Serial.print((float)ay / 16384); Serial.print("g\t");
45    Serial.print((float)az / 16384); Serial.print("g\t");
46    Serial.print((float)gx / 131); Serial.print("d/s  \t");
47    Serial.print((float)gy / 131); Serial.print("d/s  \t");
48    Serial.print((float)gz / 131); Serial.print("d/s  \n");
49    delay(300);
50    // blink LED to indicate activity
51    blinkState = !blinkState;
52    digitalWrite(LED_PIN, blinkState);
53  }
```

We reference the libraries designed for the I2C bus and the MPU6050 to manipulate the MPU6050.

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
```

MPU6050 library provides MPU6050 class to manipulate the MPU6050, and it is necessary to instantiate an object of class before using it.

```
MPU6050 accelgyro;          // Construct a MPU6050 object using the default address
```

First initialize the I2C bus, and then initialize the MPU6050.

```
  Wire.begin();             // initialize I2C
  accelgyro.initialize();   // initialize MPU6050
```

If you want to make the results more close to the actual situation, you can adjust the offset of MPU6050 before using it. You can refer to the MPU6050 library files for more details about setting offset. If there are no strict requirements, this step can also be ignored.

```
  // when you need to calibrate the gravity acceleration, you can set the offset here and eliminate the note
  // accelgyro.setXAccelOffset(-1200);
  // accelgyro.setYAccelOffset(-2500);
  // accelgyro.setZAccelOffset(1988);
```

We can also read out the value of the offset which is already set through following code:

```
  Serial.print("X.Y.Z offset :\t");
  Serial.print(accelgyro.getXAccelOffset()); Serial.print("\t");
  Serial.print(accelgyro.getYAccelOffset()); Serial.print("\t");
  Serial.print(accelgyro.getZAccelOffset()); Serial.print("\n");
```

Read the values of 3 accelerations and 3 angular accelerations in the loop () function,

```
  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

Then, convert the data and send them to the serial port. For the conversion from the raw data unit of MPU6050 to the standard unit, please refer to datasheet.

& Operator

The function of the operator "&" is to get the address. We know that the parameter of the function is used to pass the value to the function body. When a function is called, the value of variables that works as parameters does not change.

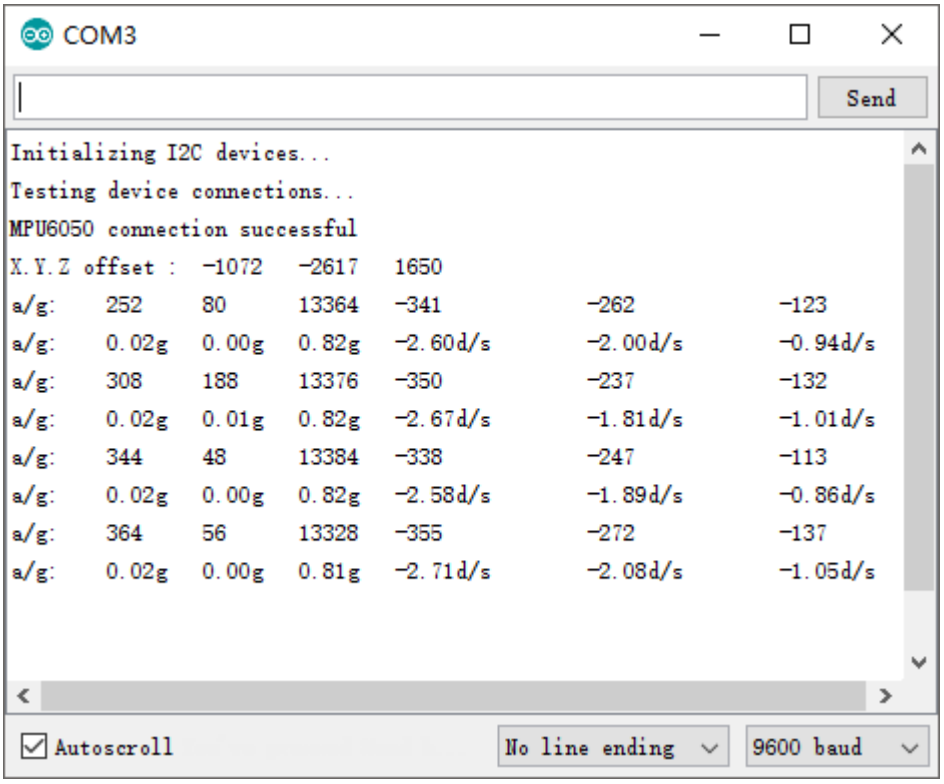
If the parameters of function are defined as pointer type, then when the function is called, the variable as function parameter will be passed to the function body and participate in the operation of the function body. And the value will be changed. It is equivalent to that the function indirectly return more value.

A pointer type variable points to an address. When we define it, we need to add "*" in front of it, for example:

```
int *a;
```

When the function's parameter is pointer type, and the common variable works as parameter of the function, the & operator need to be added in front of the parameter.

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



Data sent by this code may be too much for the users not familiar with acceleration. You can choose to upload sketch 14.1.2, which only send three direction acceleration values to the serial port. And it will be relatively easy to observe change of numbers, when you rotate and move this module,

Chapter 15 LED Matrix

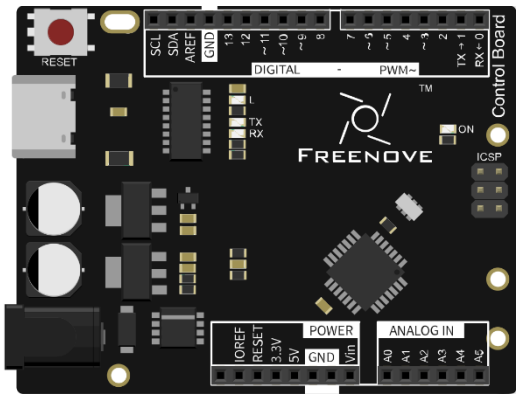
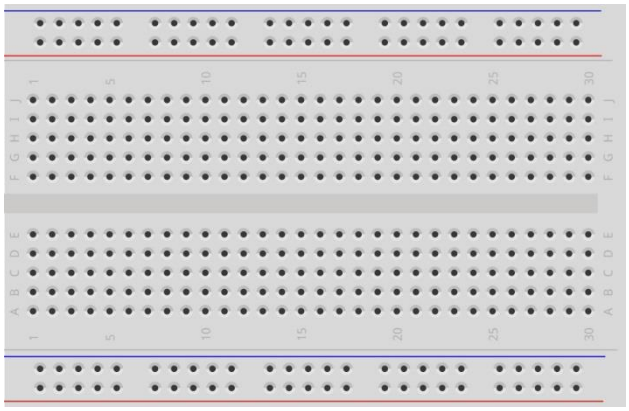
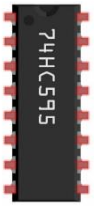
In the previous chapter, we have learned how to use some modules and sensors and shows some information on the computer through serial port. Now let us learn some modules which can output images and text.

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

Project 15.1 74HC595

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

Component List

<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td rowspan="2">USB cable x1
  </td><td>74HC595 x1
  </td><td>LED bar graph x1
  </td><td>Resistor 220Ω x8
  </td></tr>
<tr><td colspan="3">Jumper M/M x17
  </td></tr>
</table>

Code Knowledge

Hexadecimal

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

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

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

Sequence	1	0
Number	1	2

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

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

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

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

The result is of the conversion 0x12.

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

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

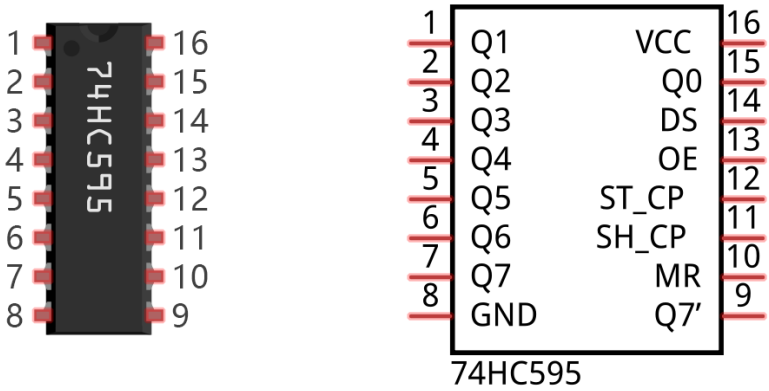
4 bit binary	1000	1001	1010	1011	1100	1101	1110	1111
1 figure of hexadecimal	8	9	A	B	C	D	E	F

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

Component Knowledge

74HC595

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



The ports of 74HC595 are described as follows:

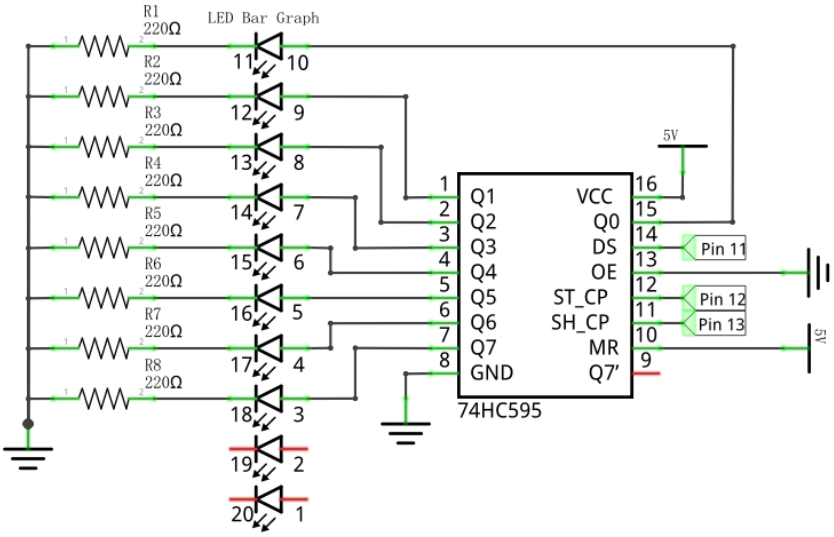
Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin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

For more detail, please refer to the datasheet.

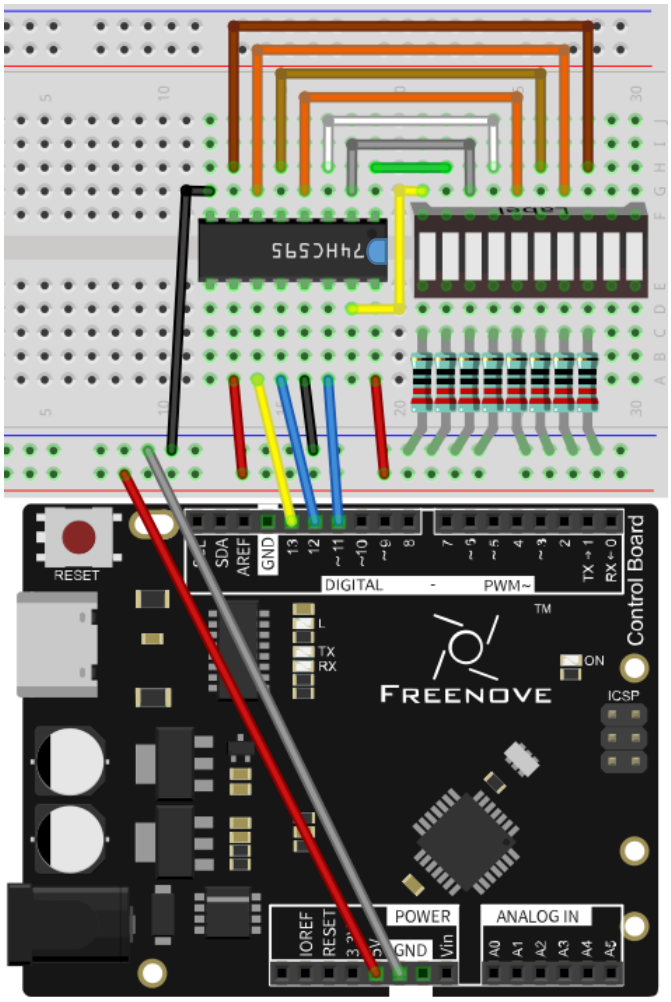
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 15.1.1

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

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

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

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

In the code, we configure three pins to control the 74HC595. And define a one-byte variable to control the state of 8 LEDs through the 8 bits of the variable. The LED lights on when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED on.

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

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

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

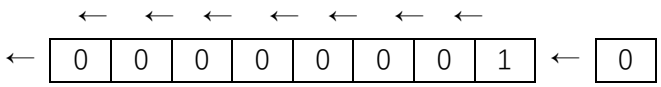
The x will be shift 1 bit to left in each cycle, which makes the bright LED of the 8 LEDs move one bit.

```
24      x <<= 1;
```

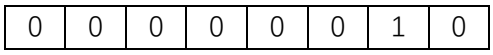
<< operator

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

byte x = 1 << 1;

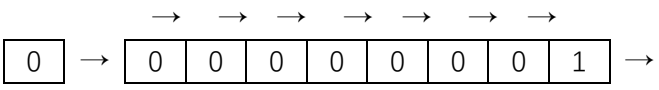


The result of x is 2 (binary 00000010)。

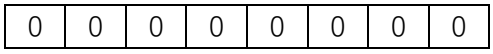


There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

byte x = 1 >> 1;



The result of x is 0 (00000000)。



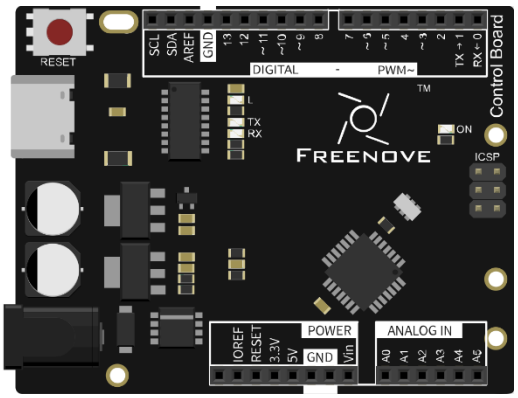
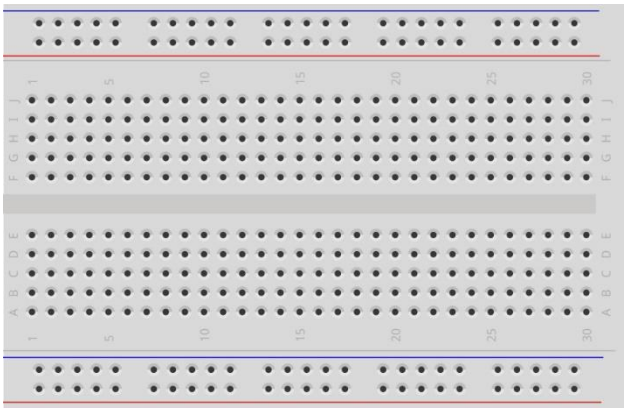
X <<= 1 is equivalent to x = x << 1 and x >>= 1 is equivalent to x = x >> 1

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

Project 15.2 LED Matrix

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

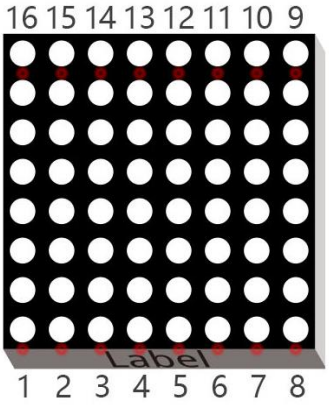
Component List

<table>
<tr><td>Control board x1
  </td><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>LED matrix x1
  </td><td>74HC595 x1
  </td><td>R220 x8
  </td></tr>
<tr><td>Jumper M/M x33
  </td><td colspan="3"></td></tr>
</table>

Component Knowledge

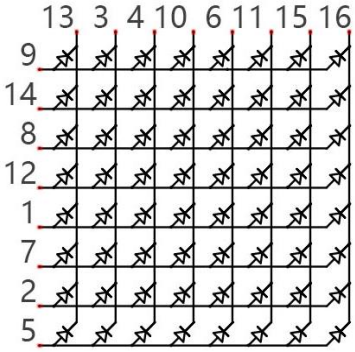
LED matrix

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

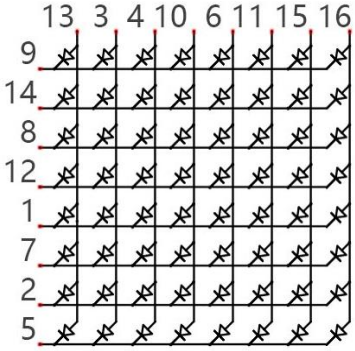


In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

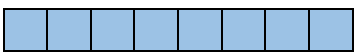
Connection mode of common anode

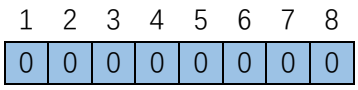


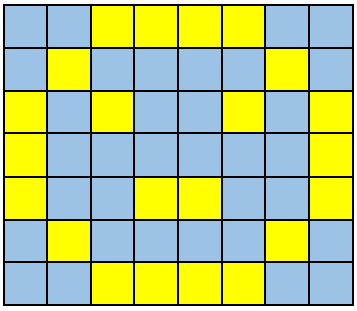
Connection mode of common cathode



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







0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

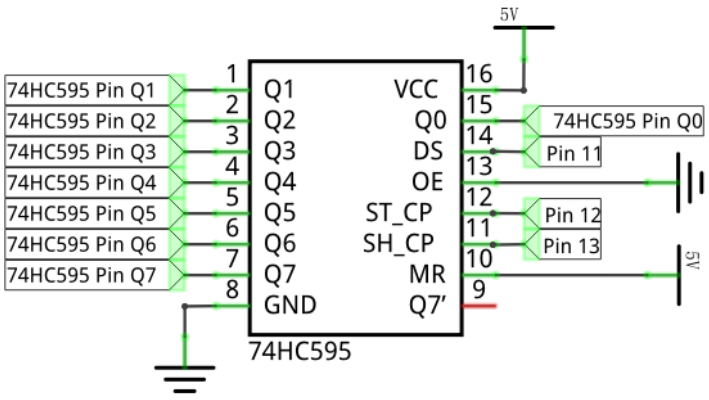
Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

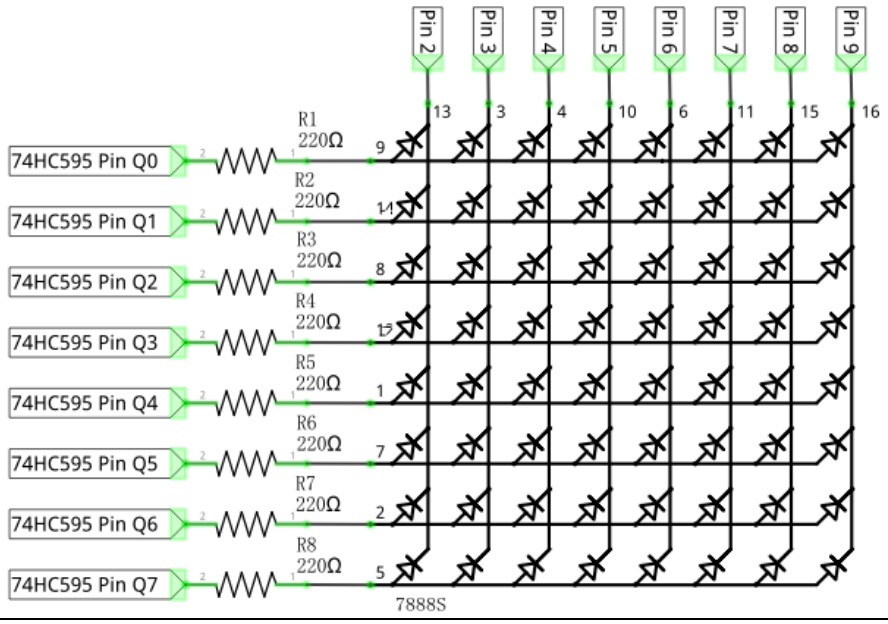
Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit.

Circuit

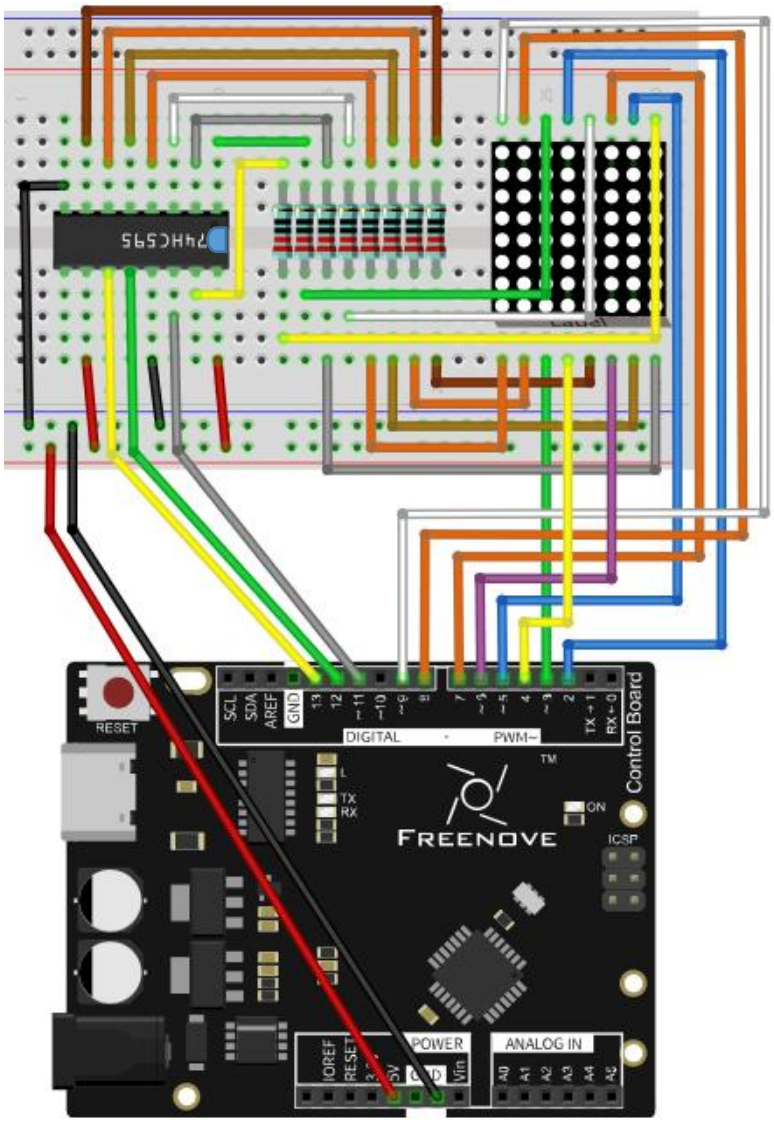
Use pin 11, 12, 13 on control board to control the 74HC595. And connect 74HC595 to the 8 anode pins of LED Matrix, in the meanwhile, connect 8 digitals port on control board to the 8 cathode pins of LED Matrix.

Schematic diagram





Hardware connection



Sketch

Sketch 15.2.1

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

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

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

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

```
}

void loop() {
  // Define a one-byte variable (8 bits) which is used to represent the selected state of 8 columns.
  int cols;
  // Display the static smiling pattern
  for (int j = 0; j < 500; j++ ) {  // repeat 500 times
    cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the first column is selected.
    for (int i = 0; i < 8; i++) {   // display 8 column data by scanning
      matrixColsVal(cols);          // select this column
      matrixRowsVal(smilingFace[i]);// display the data in this column
      delay(1);                     // display them for a period of time
      matrixRowsVal(0x00);          // clear the data of this column
      cols <<= 1;                   // shift "cols" 1 bit left to select the next column
    }
  }
  // Display the dynamic patterns of numbers and letters
  for (int i = 0; i < 128; i++) { // "space, 0-9, A-F"16 letters ,each letter hold 8 columns, total 136 columns. Firstly, display space ,then we need shift 128 times (136-8)
    for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
      cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
      for (int j = i; j < 8 + i; j++) { // display image of each frame
        matrixColsVal(cols);            // select this column
        matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
        delay(1);                       // display them for a period of time
        matrixRowsVal(0x00);            // close the data of this column
        cols <<= 1;                 // shift "cols" 1 bit left to select the next column
      }
    }
  }
}

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

```

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

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

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

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

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

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

```
11  const int data[] PROGMEM = {
12    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
13    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
14    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
15    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
16    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
17    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
18    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
19    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
20    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
21    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
22    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
23    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
24    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
25    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
26    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
27    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
28    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00  // "F"
29  };
```

PROGMEM keyword

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

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

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

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

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

? : operator

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

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

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

Bitwise logical operation

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

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

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

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

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

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

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

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

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

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

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

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

```
56    for (int i = 0; i < 128; i++) { // "space, 0-9, A-F"16 letters ,each letter hold 8
      columns, total 136 columns. Firstly, display space ,then we need shift 128 times (136-8)
57      for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
58        cols = 0x01;       // Assign binary 00000001. Means the first column is selected.
59        for (int j = i; j < 8 + i; j++) { // display image of each frame
60          matrixColsVal(cols);            // select this column
61          matrixRowsVal(pgm_read_word_near(data + j));// display the data in this column
62          delay(1);                       // display them for a period of time
63          matrixRowsVal(0x00);            // close the data of this column
64          cols <<= 1;                   // shift"cols" 1 bit left to select the next column
65        }
66      }
67    }
68  }
```

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

Chapter 16 I2C LCD1602

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

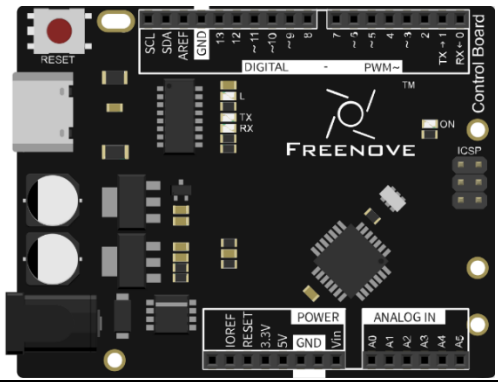
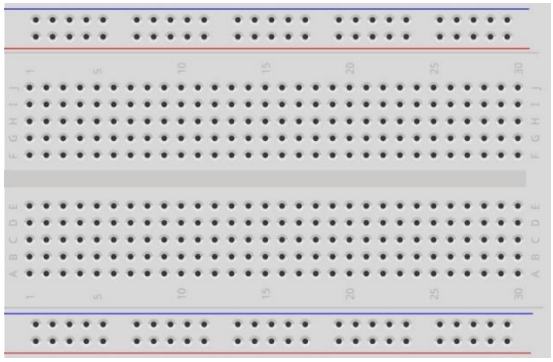
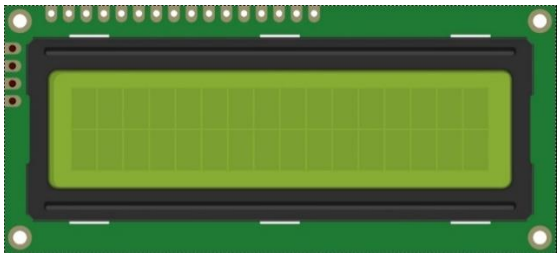
There are multiple versions of LCD1602. Your purchase may be one of the following:

I2C LCD1602	I2C LCD1602
	

Project 16.1 Display the String on I2C LCD1602

Firstly, use I2C LCD1602 to display some strings.

Component List

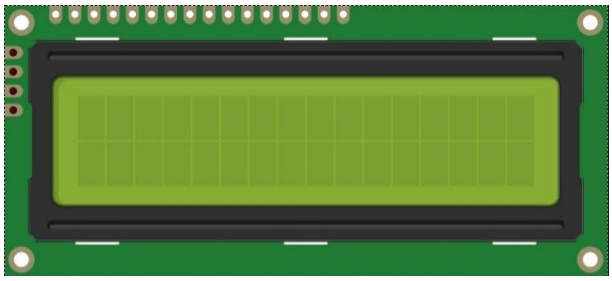
Control board x1 	Breadboard x1 
USB cable x1 	I2C LCD1602 Module x1 
Jumper M/F x4 	

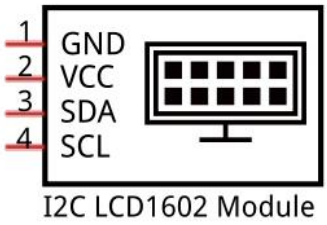
Component Knowledge

I2C LCD1602

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

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





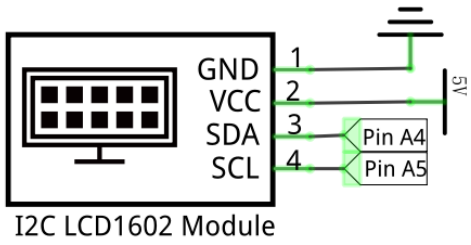
I2C (Inter-Integrated Circuit) is a two-wire serial communication, which is used to connect micro controller with its peripheral equipments. Device that use I2C communication are all connected to the serial data (SDA) line and a serial clock (SCL) line (called I2C bus). Each device among them has a unique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

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

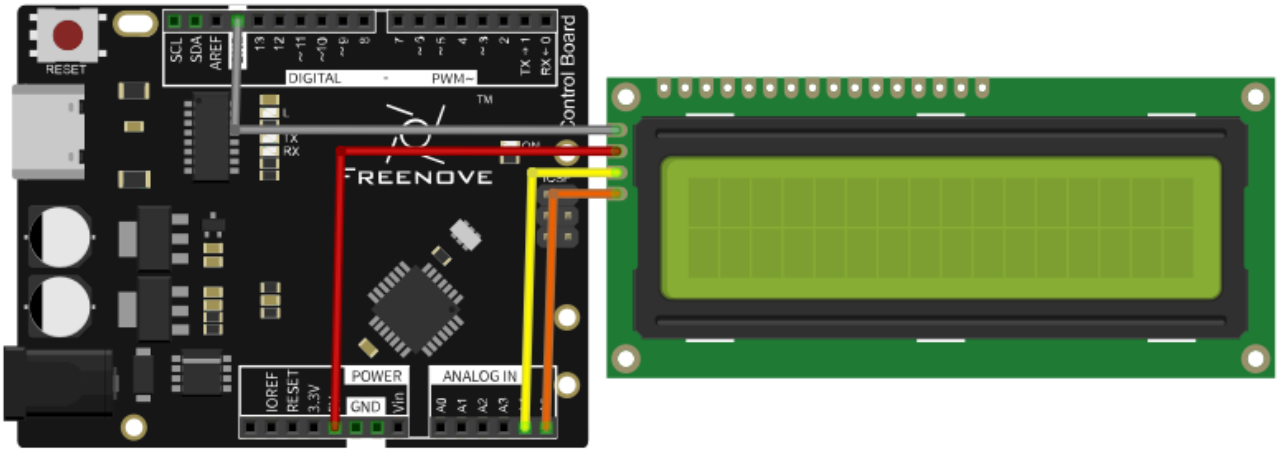
Circuit

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

Schematic diagram



Circuit connection



Sketch

Sketch 16.1.1

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

Click "Add .ZIP Library..." and then find LiquidCrystal_I2C.zip in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library can facilitate our operation of I2C LCD1602.

Now let's start to write code to use LCD1602 to display static characters and dynamic variables.

```
#include <LiquidCrystal_I2C.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                  // initialize the lcd
  lcd.backlight();             // Turn on backlight
  lcd.print("hello, world!");// Print a message to the LCD
}
void loop() {
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);// set the cursor to column 0, line 1
  // print the number of seconds since reset:
  lcd.print("Counter:");
  lcd.print(millis() / 1000);
}
bool i2CAddrTest(uint8_t addr) {
  Wire.begin();
  Wire.beginTransmission(addr);
  if (Wire.endTransmission() == 0) {
    return true;
  }
  return false;
}
```

Following are the LiquidCrystal_I2C library used for controlling LCD:

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

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

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

First, initialize the LCD and turn on LCD backlight.

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

And then print a string:

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

Print a changing number in the loop () function:

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

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

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

LiquidCrystal_I2C Class

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

LiquidCrystal_I2C lcd(0x27, 16, 2);

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

The functions used in the LiquidCrystal_I2C class are as follows:

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

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

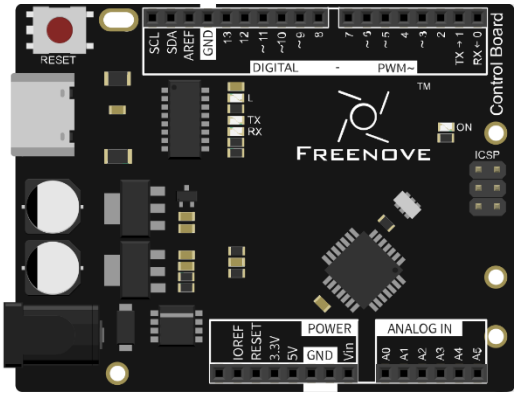
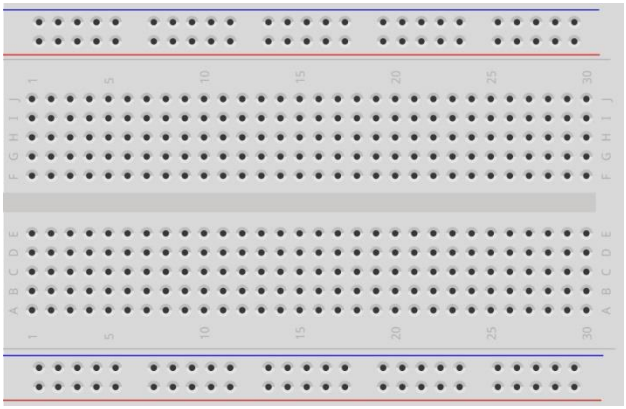
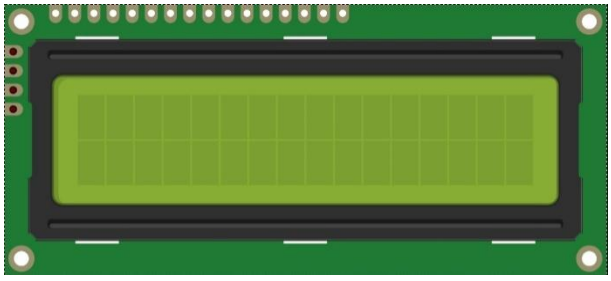
Verify and upload the code, then observe the LCD screen. If the display is not clear or there is no display, adjust the potentiometer on the back of I2C module to adjust the screen contrast until the character is clearly displayed on the LCD.

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

Project 16.2 I2C LCD1602 Clock

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

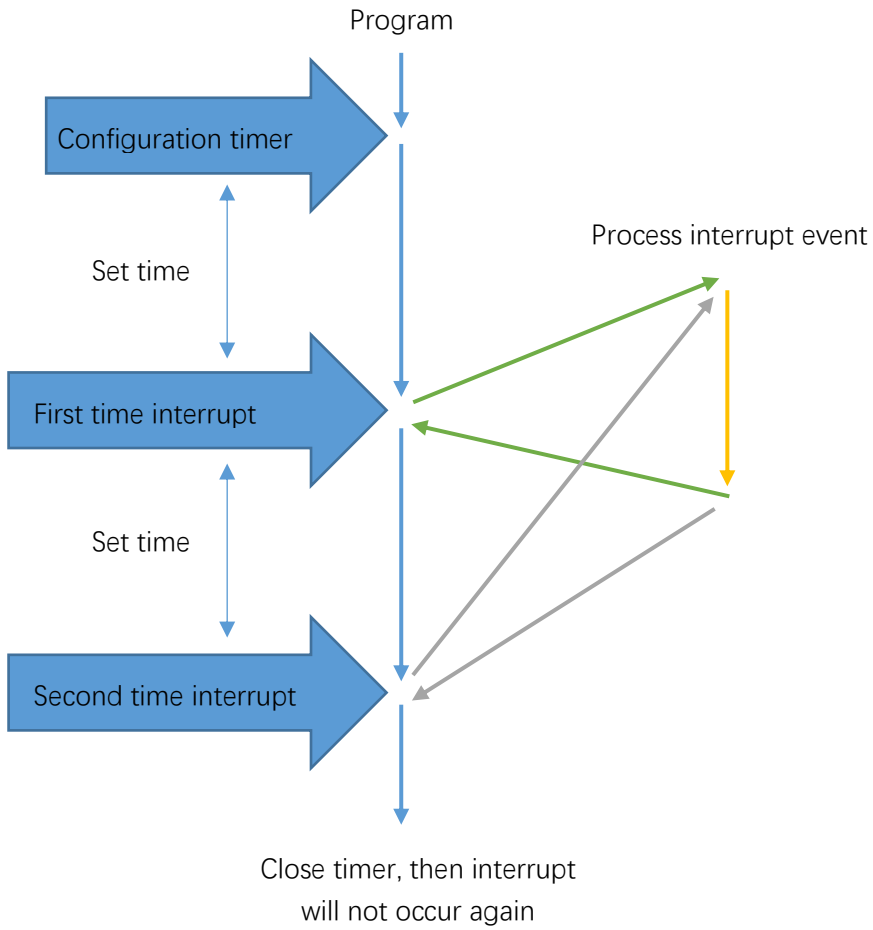
Component List

<table>
<tr><td>Control board x1
  </td><td>Breadboard x1
  </td></tr>
<tr><td>USB cable x1
  </td><td>Jumper M/M x18
  </td></tr>
<tr><td>I2C LCD1602 Module x1
  </td><td>Thermistor x1
  </td><td>Resistor 10kΩ x1
  </td></tr>
</table>

Code Knowledge

Timer

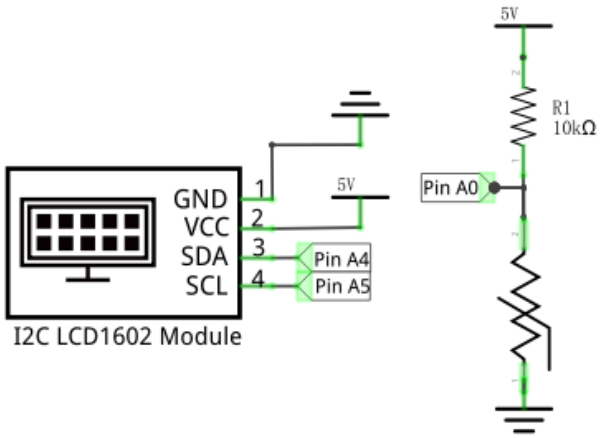
A Timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. And after completion the processing, execution will return to the interrupted location to go on. If you don't close the timer,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



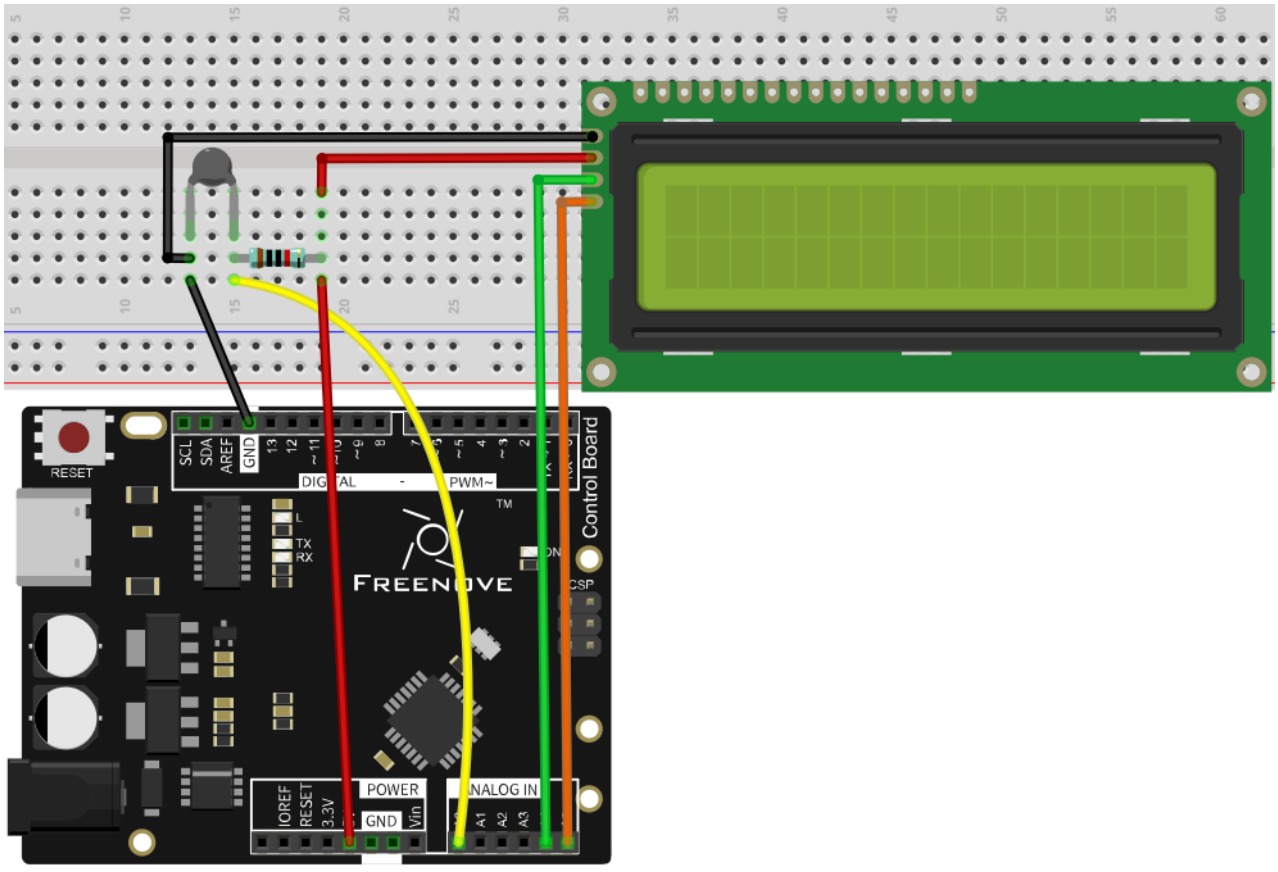
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 16.2.1

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

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

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

```
#include <LiquidCrystal_I2C.h>
#include <FlexiTimer2.h>      // Contains FlexiTimer2 Library

// initialize the library with the numbers of the interface pins
LiquidCrystal_I2C lcd(0x27, 16, 2);
int tempPin = 0;                // define the pin of temperature sensor
float tempVal;                  // define a variable to store temperature value
int hour, minute, second;       // define variables stored record time

void setup() {
  if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
  }
  lcd.init();                  // initialize the lcd
  lcd.backlight();             // Turn on backlight
  startingAnimation();         // display a dynamic start screen
  FlexiTimer2::set(1000, timerInt);  // configure the timer and interrupt function
  FlexiTimer2::start();              // start timer
  Serial.begin(115200);                // initialize serial port with baud rate 9600
  Serial.println("Input hour,minute,second to set time.");
}

void loop() {
  // Get temperature
  tempVal = getTemp();
  if (second >= 60) {       // when seconds is equal to 60, minutes plus 1
    second = 0;
    minute++;
    if (minute >= 60) {     // when minutes is equal to 60, hours plus 1
      minute = 0;
      hour++;
      if (hour >= 24) {     // when hours is equal to 24, hours turn to zero
        hour = 0;
      }
    }
```

```
36    }
37    lcdDisplay();              // display temperature and time information on LCD
38    delay(200);
39  }
40
41  void startingAnimation() {
42    for (int i = 0; i < 16; i++) {
43      lcd.scrollDisplayRight();
44    }
45    lcd.print("starting...");
46    for (int i = 0; i < 16; i++) {
47      lcd.scrollDisplayLeft();
48      delay(300);
49    }
50    lcd.clear();
51  }
52  // the timer interrupt function of FlexiTimer2 is executed every 1s
53  void timerInt() {
54    second++;       // second plus 1
55  }
56  // serial port interrupt function
57  void serialEvent() {
58    int inInt[3];  // define an array to save the received serial data
59    while (Serial.available()) {
60      for (int i = 0; i < 3; i++) {
61        inInt[i] = Serial.parseInt();  // receive 3 integer data
62      }
63      // print the received data for confirmation
64      Serial.print("Your input is: ");
65      Serial.print(inInt[0]);
66      Serial.print(", ");
67      Serial.print(inInt[1]);
68      Serial.print(", ");
69      Serial.println(inInt[2]);
70      // use received data to adjust time
71      hour = inInt[0];
72      minute = inInt[1];
73      second = inInt[2];
74      // print the modified time
75      Serial.print("Time now is: ");
76      Serial.print(hour / 10);
77      Serial.print(hour % 10);
78      Serial.print(':');
79      Serial.print(minute / 10);
```

```
80      Serial.print(minute % 10);
81      Serial.print(':');
82      Serial.print(second / 10);
83      Serial.println(second % 10);
84    }
85  }
86  // function used by LCD1602 to display time and temperature
87  void lcdDisplay() {
88    lcd.setCursor(0, 0);  // set the cursor to (0,0) (first column,first row).
89    lcd.print("TEMP: ");  // display temperature information
90    lcd.print(tempVal);
91    lcd.print("C");
92    lcd.setCursor(0, 1);  // set the cursor to (0,1) (first column,second row)
93    lcd.print("TIME: ");  // display time information
94    lcd.print(hour / 10);
95    lcd.print(hour % 10);
96    lcd.print(':');
97    lcd.print(minute / 10);
98    lcd.print(minute % 10);
99    lcd.print(':');
100   lcd.print(second / 10);
101   lcd.print(second % 10);
102 }
103 // function used to get temperature
104 float getTemp() {
105   // Convert analog value of tempPin into digital value
106   int adcVal = analogRead(tempPin);
107   // Calculate voltage
108   float v = adcVal * 5.0 / 1024;
109   // Calculate resistance value of thermistor
110   float Rt = 10 * v / (5 - v);
111   // Calculate temperature (Kelvin)
112   float tempK = 1 / (log(Rt / 10) / 3950 + 1 / (273.15 + 25));
113   // Calculate temperature (Celsius)
114   return tempK - 273.15;
115 }
116 bool i2CAddrTest(uint8_t addr) {
117   Wire.begin();
118   Wire.beginTransmission(addr);
119   if (Wire.endTransmission() == 0) {
120     return true;
121   }
122   return false;
123 }
```

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

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

Defines a timer with cycle of 1000 millisecond (1 second). And each interrupt makes the second plus 1. When setting the timer, you need to define a function and pass the function name that works as a parameter to FlexiTimer2::set () function.

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

After every interrupt, the second plus 1.

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

:: operator

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

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

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

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

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

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

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

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

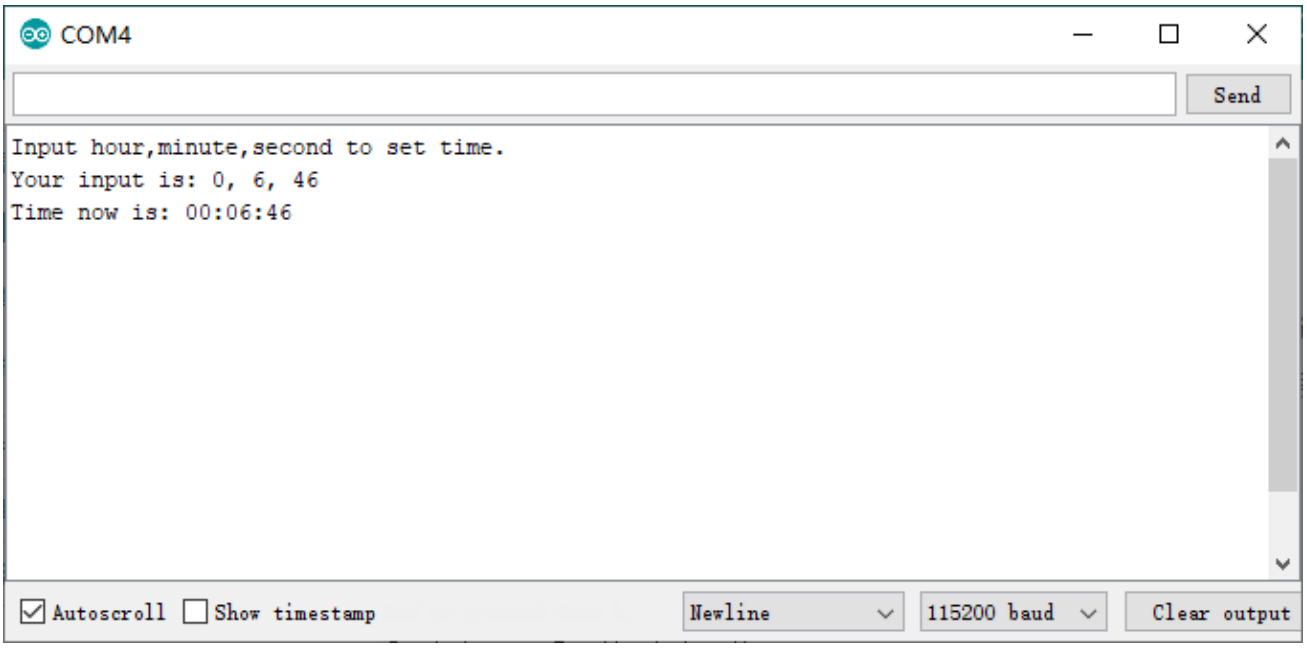
Serial port interrupt function is used to receive the data sent by computer to adjust the time, and return the data for confirmation.

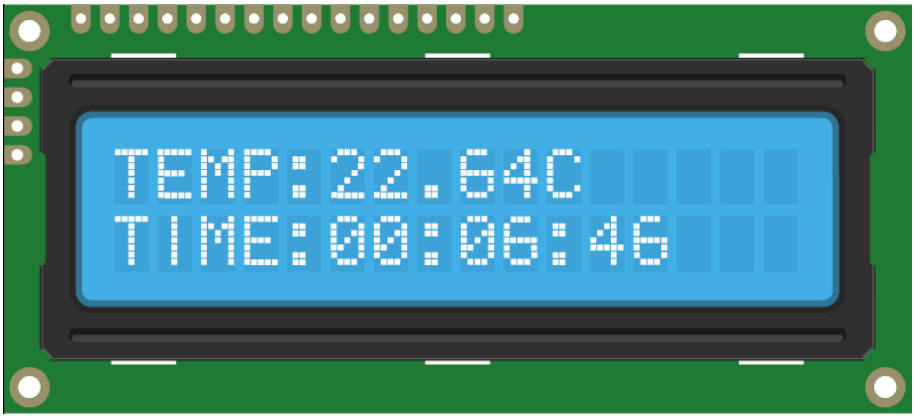
```
void serialEvent() {
  int inInt[3];  // define an array to save the received serial data
  while (Serial.available()) {
    for (int i = 0; i < 3; i++) {
      inInt[i] = Serial.parseInt();  // receive 3 integer data
    }
    // print the received data for confirmation
    Serial.print("Your input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

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

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

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





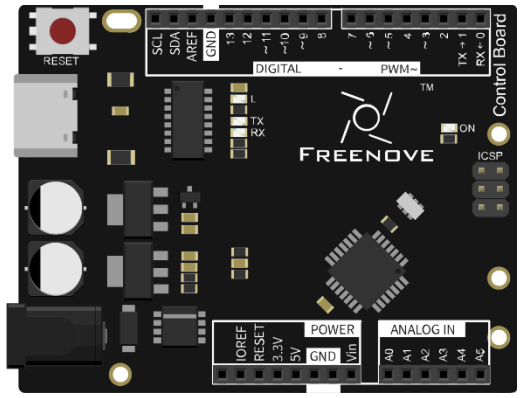
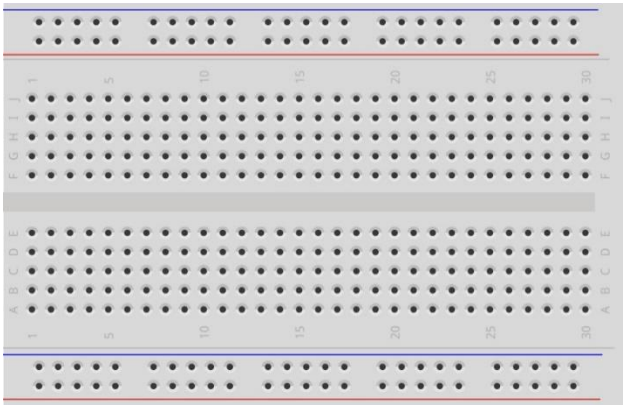
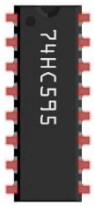
Chapter 17 Digital Display

Digital display is a kind of device that can display one or several digits. We will learn how to use it in this chapter.

Project 17.1 1-digit 7-segment Display

Frist, try to use the digit display that can display 1-digit number.

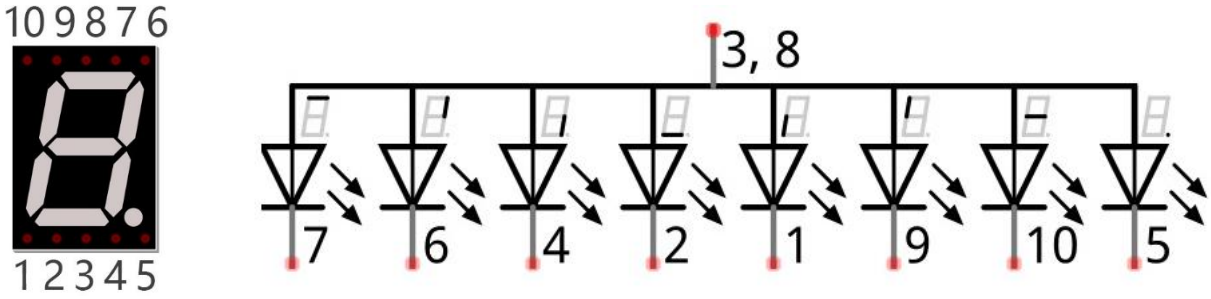
Component List

Control board x1 	Breadboard x1 		
USB cable x1 	74HC595 x1 	Resistor 220Ω x8 	1-digit 7-segment display x1 
Jumper M/M x18 			

Component Knowledge

1-digit 7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. It can display numbers of 0~9 by lighting some of its LED segment. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a “0”: we need to turn ON LED segments 7, 6, 4, 2, 1 and 9, and turn OFF LED segments 5 and 10.

If we use a byte to show the state of the LEDs that connected to pin 5, 10, 9, 1, 2, 4, 6, 7, we can make 0 represent the state of ON and 1 for OFF. Then the number 0 can be expressed as a binary number 11000000, namely hex 0xc0.

The numbers and letters that can be display are shown below:

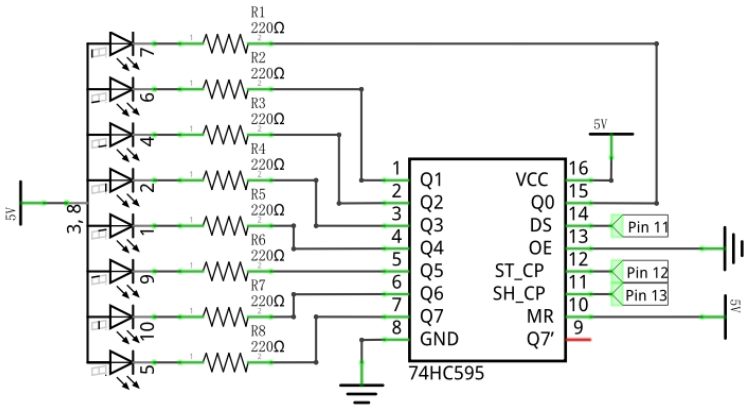
Number/Letter	Binary number	Hexadecimal number
0	11000000	0xc0
1	11111001	0xf9
2	10100100	0xa4
3	10110000	0xb0
4	10011001	0x99
5	10010010	0x92
6	10000010	0x82
7	11111000	0xf8
8	10000000	0x80
9	10010000	0x90
A	10001000	0x88
b	10000011	0x83
C	11000110	0xc6
d	10100001	0xa1

E	10000110	0x86
F	10001110	0x8e

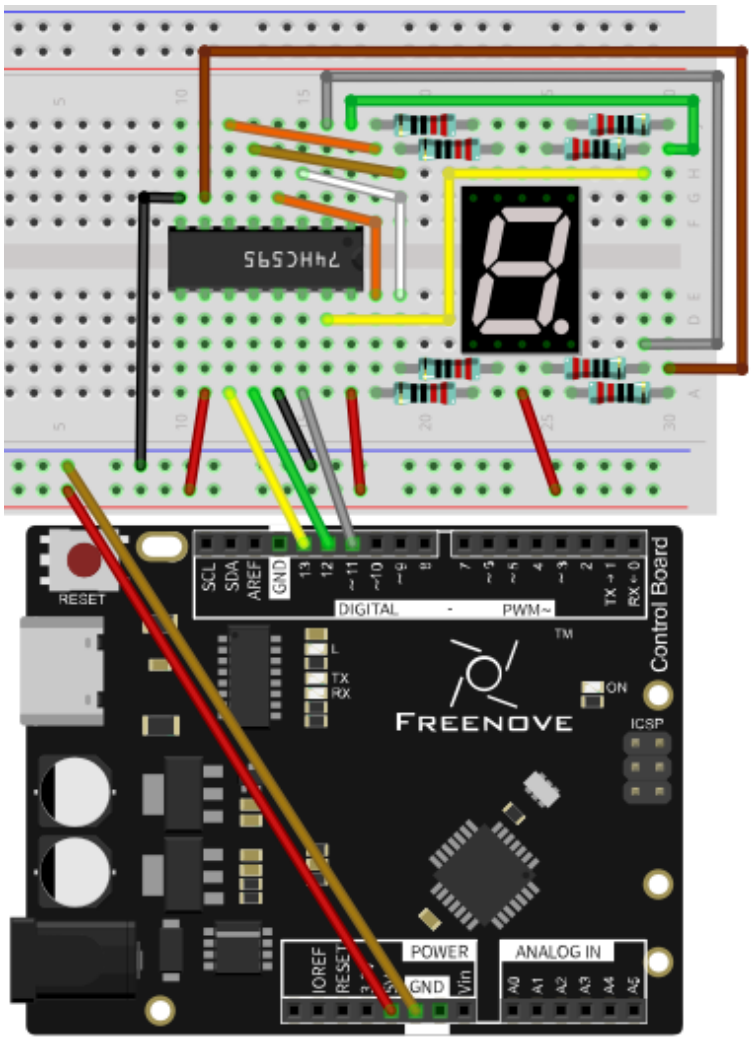
Circuit

Use pin 11, 12, 13 on control board to control the 74HC595, and connect it to a1-digit 7-segment display.

Schematic diagram



Hardware connection



Sketch

Sketch 17.1.1

Now write code to control the 1-digit 7-segment display through 74HC595.

```
1  int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
2  int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
3  int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
4
5  // Define the encoding of characters 0-F of the common-anode 7-segment Display
6  byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83,
   0xc6, 0xa1, 0x86, 0x8e};
7
8  void setup() {
9    // set pins to output
10   pinMode(latchPin, OUTPUT);
11   pinMode(clockPin, OUTPUT);
12   pinMode(dataPin, OUTPUT);
13 }
14
15 void loop() {
16   // Cycling display 0-F
17   for (int i = 0; i <= 0x0f; i++) {
18     // Output low level to latchPin
19     digitalWrite(latchPin, LOW);
20     // Send serial data to 74HC595
21     shiftOut(dataPin, clockPin, MSBFIRST, num[i]);
22     // Output high level to latchPin, and 74HC595 will update the data to the parallel
   output port.
23     digitalWrite(latchPin, HIGH);
24     delay(1000);
25   }
26 }
```

We define an array to save and display the encoding of character 0-F in this code.

```
6  byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83,
   0xc6, 0xa1, 0x86, 0x8e};
```

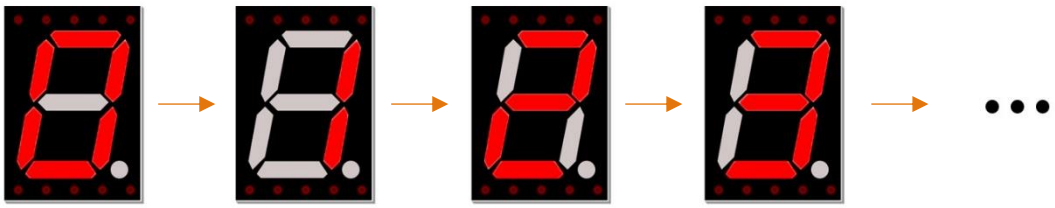
Initialize the pin connected to 74HC595 in setup() function.

```
8  void setup() {
9    // set pins to output
10   pinMode(latchPin, OUTPUT);
11   pinMode(clockPin, OUTPUT);
12   pinMode(dataPin, OUTPUT);
13 }
```

Then in loop() function, send the encoding of 0-F to 74HC595 circularly.

```
17    for (int i = 0; i <= 0x0f; i++) {
18      // Output low level to latchPin
19      digitalWrite(latchPin, LOW);
20      // Send serial data to 74HC595
21      shiftOut(dataPin, clockPin, MSBFIRST, num[i]);
22      // Output high level to latchPin, and 74HC595 will update the data to the parallel
23  output port.
        digitalWrite(latchPin, HIGH);
24      delay(1000);
25    }
```

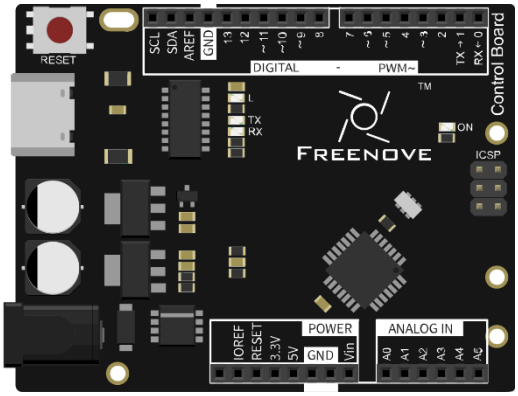
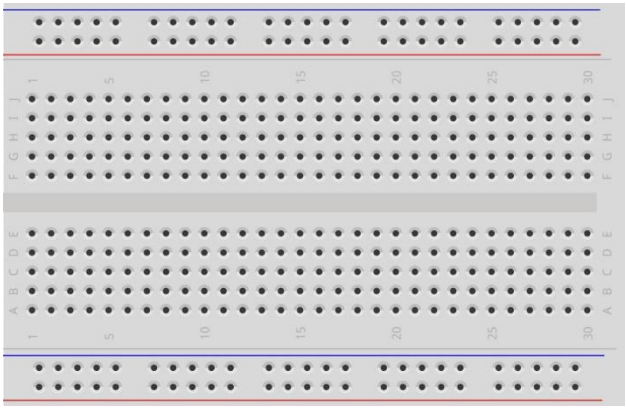
Verify and upload the code, then you will see the 1-digit 7-segment display show 0-F in a circular manner.



Project 17.2 4-digit 7-segment Display

Now, try to use digit display that can display 4-digit numbers.

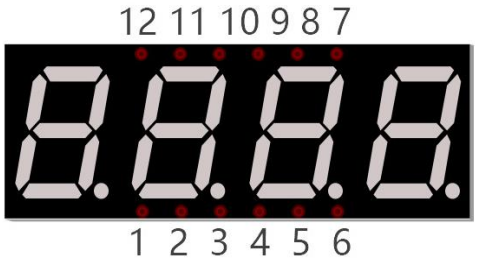
Component List

Control board x1 	Breadboard x1 	
USB cable x1 	Jumper M/M x21 	
4-digit 7-segment display x1 	74HC595 x1 	Resistor 220Ω x8 

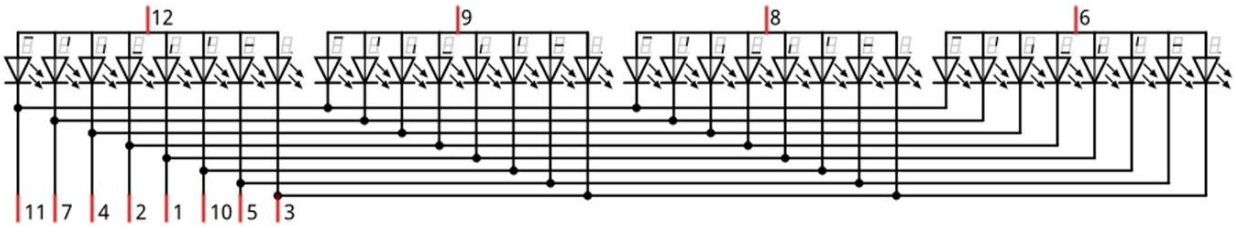
Component Knowledge

4-digit 7-segment display

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



The internal electronic circuit is shown below, and all eight LED cathode pins of each 1-digit 7-segment display are connected together.



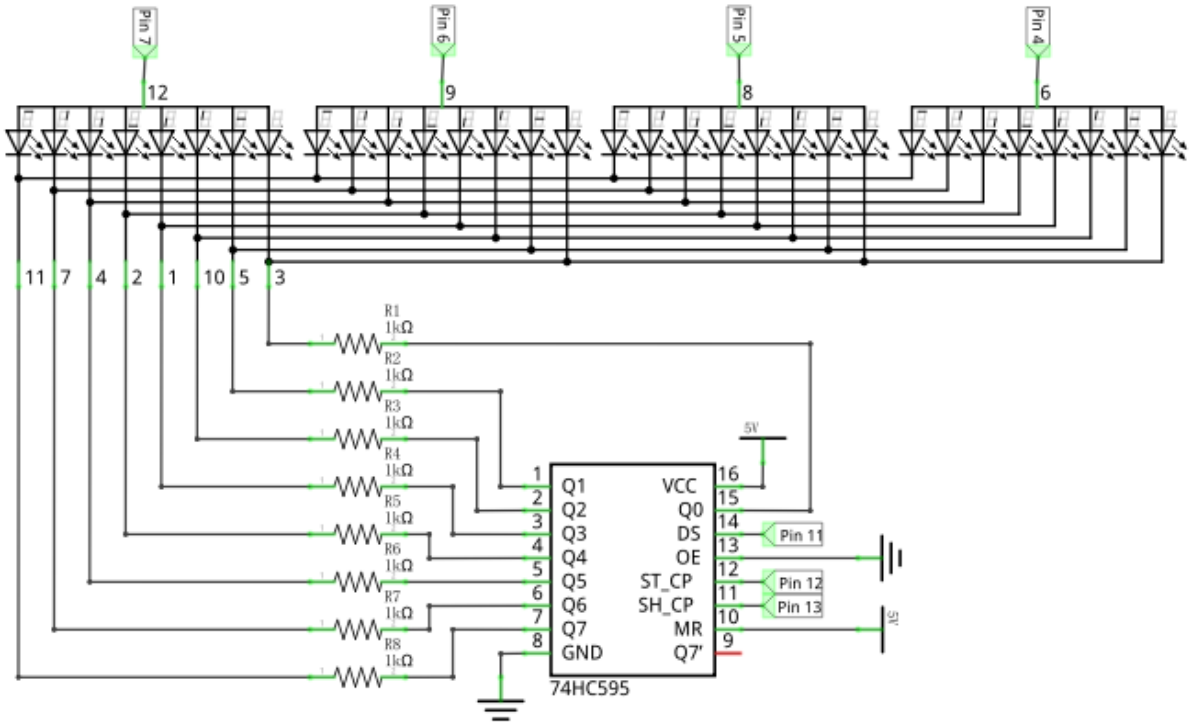
Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to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

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

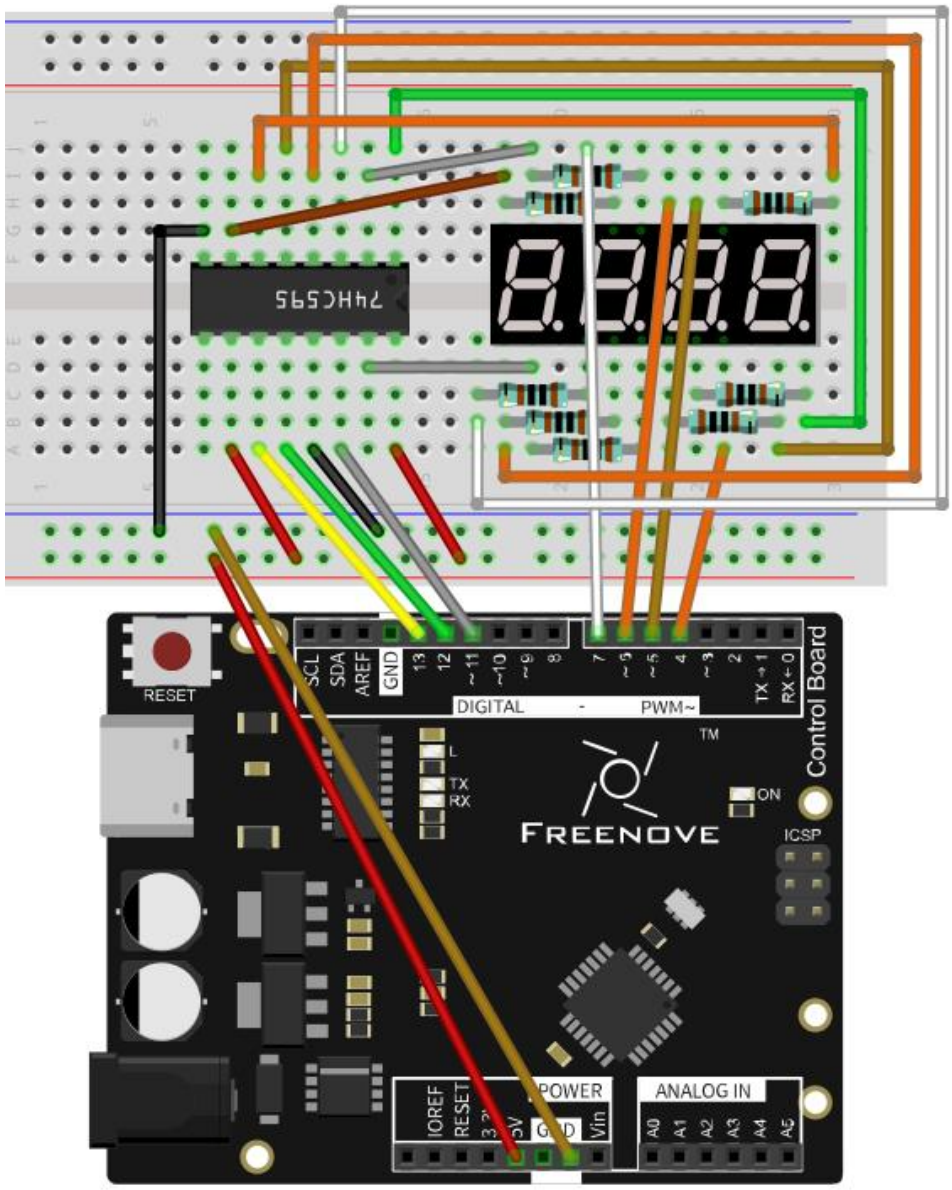
Circuit

Use pin 11, 12, 13 on control board to control the 74HC595, and connect it to the 4-digit 7-segment display. Use pin 7, 6, 5, 4 to control the 4 common ports.

Schematic diagram



Hardware connection



Sketch

Sketch 17.2.1

Now, write code to control 4-digit 7-segment display to display 4 numbers.

```
int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {7, 6, 5, 4};// Common pin (anode) of 4 digit 7-segment display

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

```
8
9   void setup() {
10    // set pins to output
11    pinMode(latchPin, OUTPUT);
12    pinMode(clockPin, OUTPUT);
13    pinMode(dataPin, OUTPUT);
14    for (int i = 0; i < 4; i++) {
15      pinMode(comPin[i], OUTPUT);
16    }
17  }
18
19  void loop() {
20    for (int i = 0; i < 4; i++) {
21      // Select a single 7-segment display
22      chooseCommon(i);
23      // Send data to 74HC595
24      writeData(num[i]);
25      delay(5);
26      // Clear the display content
27      writeData(0xff);
28    }
29  }
30
31  void chooseCommon(byte com) {
32    // Close all single 7-segment display
33    for (int i = 0; i < 4; i++) {
34      digitalWrite(comPin[i], LOW);
35    }
36    // Open the selected single 7-segment display
37    digitalWrite(comPin[com], HIGH);
38  }
39
40  void writeData(int value) {
41    // Make latchPin output low level
42    digitalWrite(latchPin, LOW);
43    // Send serial data to 74HC595
44    shiftOut(dataPin, clockPin, LSBFIRST, value);
45    // Make latchPin output high level, then 74HC595 will update the data to parallel
    output
46    digitalWrite(latchPin, HIGH);
47  }
```

Besides the similarity with the previous section, the difference is that this code is to output content to the four 7-segment display continuously. Write a function to select a common port.

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

Write a function to send data to 74HC595.

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

First select a common port and then output the content, which will be displayed by the 7-segment display connected to common port, to 74HC595 when operating. Clear the display content after a period of time to avoid ghosting phenomenon.

```
21      // Select a single 7-segment display
22      chooseCommon(i);
23      // Send data to 74HC595
24      writeData(num[i]);
25      delay(5);
26      // Clear the display content
27      writeData(0xff);
```

Use cycle command in loop() function to output content to the four 7-segment display.

```
20    for (int i = 0; i < 4; i++) {
...   ...
28    }
```

Repeat this operation continuously.

Verify and upload the code, and then you will see number 0123 shown by 4 digit 7-segment display.

Sketch 17.2.2

Now write code to control 4-digit 7-segment display to display dynamic numbers.

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

int latchPin = 12;          // Pin connected to ST_CP of 74HC595 (Pin12)
int clockPin = 13;          // Pin connected to SH_CP of 74HC595 (Pin11)
int dataPin = 11;           // Pin connected to DS of 74HC595 (Pin14)
int comPin[] = {7, 6, 5, 4};// Common pin (anode) of 4 digit 7-segment display

int second = 0;             // Define variables stored record time

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

void setup() {
  // Set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (int i = 0; i < 4; i++) {
    pinMode(comPin[i], OUTPUT);
  }
  FlexiTimer2::set(1000, timerInt);  // configure the timer and interrupt function
  FlexiTimer2::start();              // start timer
}

void loop() {
  // Calculate the seconds of each digit number
  byte bit[4];
  bit[0] = second % 10;
  bit[1] = second / 10 % 10;
  bit[2] = second / 100 % 10;
  bit[3] = second / 1000 % 10;
  // Show seconds
  for (int i = 0; i < 4; i++) {
    // Select a single 7-segment display
    chooseCommon(i);
    // Send data to 74HC595
    writeData(num[bit[3 - i]]);
    delay(5);
    // Clear the display content
    writeData(0xff);
  }
```

```
}

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

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

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

Frist set a timer with a cycle of 1s

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

In the timer interrupt function, make the variable second plus 1

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

In function loop(), The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively.

```
  byte bit[4];
  bit[0] = second % 10;
  bit[1] = second / 10 % 10;
  bit[2] = second / 100 % 10;
  bit[3] = second / 1000 % 10;
```

Then display the value of each bit.

```
    writeData(num[bit[3 - i]]);
```

Verify and upload the code, then you will see the dymaic number shown by 4 digit 7-segment display.

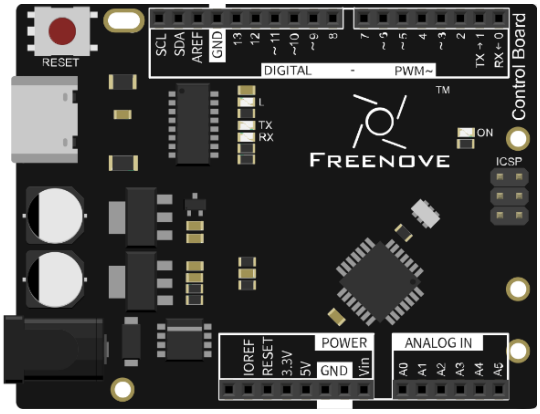
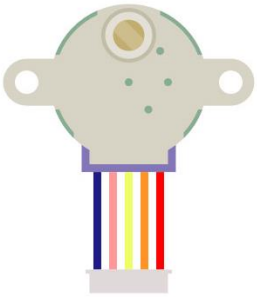
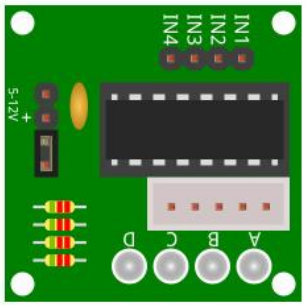
Chapter 18 Stepper Motor

A Stepper motor is a kind of motor that can rotate a certain angle at a time, with which we can achieve mechanical movement at a higher accuracy more easily.

Project 18.1 Drive Stepper Motor

Now, try to drive a stepper motor.

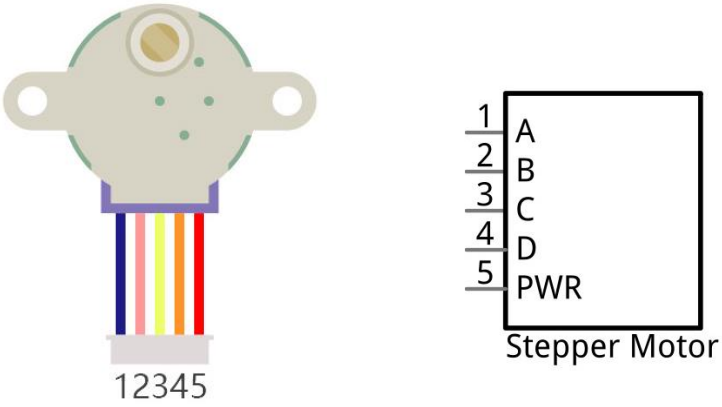
Component List

Control board x1 	Stepper motor x1 	ULN2003 stepper motor driver x1 
USB cable x1 	Jumper F/M x6 	

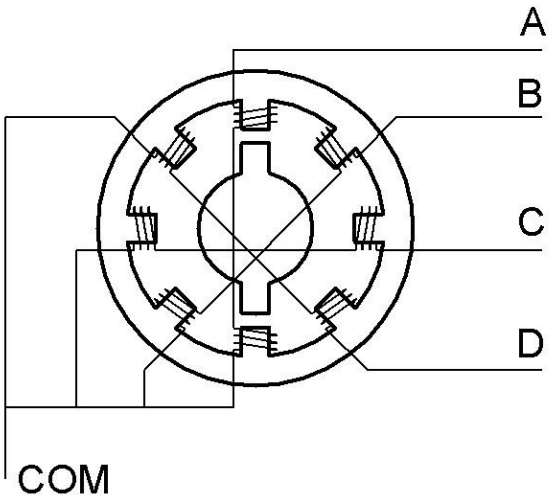
Component Knowledge

Stepper Motor

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

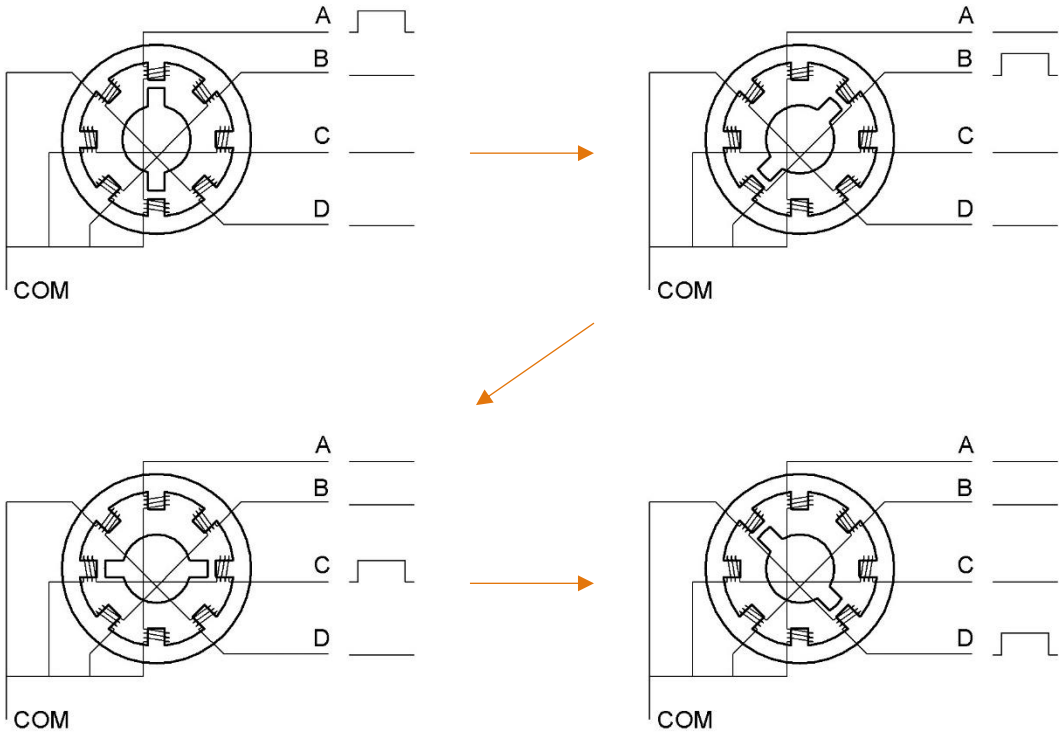


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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

A common drive sequence is as follows:



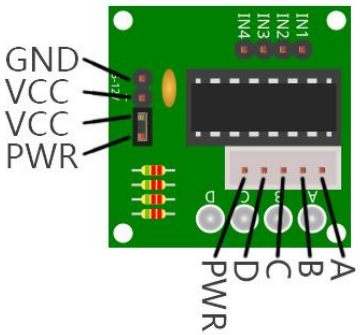
In the sequence above, the Stepper Motor rotates once at a certain angle, which is called a "step". By controlling the number of rotational steps, you can then control the Stepper Motor's rotation angle. By defining the time between two steps, you can control the Stepper Motor's rotation speed.

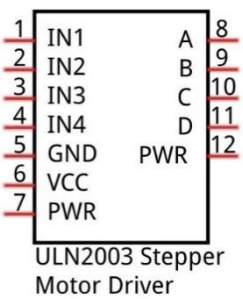
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. We can use this way to improve the stepper motor's stability and reduce noise at the same time. If you are interested in it, you can learn this method yourself by searching related knowledge.

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

ULN2003 stepper motor driver

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

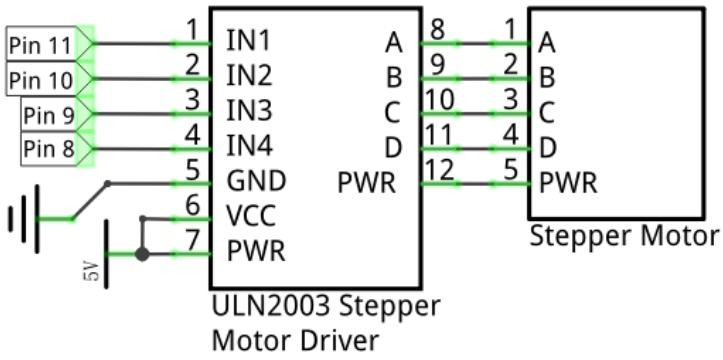




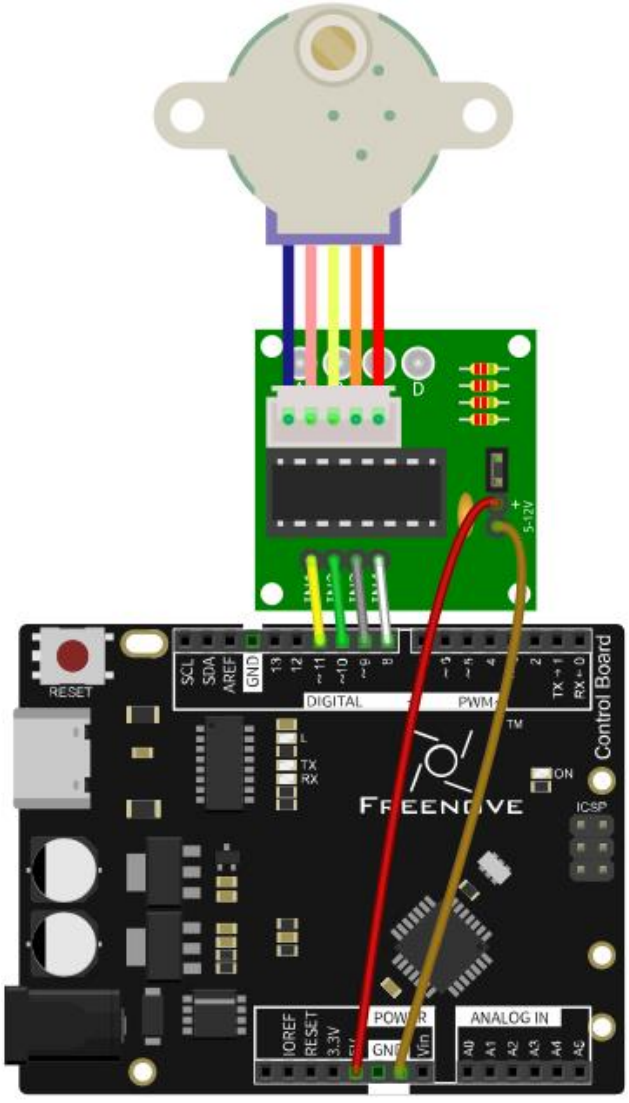
Circuit

Use pin 11, 10, 9, 8 on the control board to control the ULN2003 stepper motor driver, and connect it to the stepper motor.

Schematic diagram



Hardware connection



Sketch

Sketch 18.1.1

Now write code to control the stepper motor through ULN2003 stepper motor driver.

```
// Connect the port of the stepper motor driver
int outPorts[] = {11, 10, 9, 8};

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

void loop()
{
  // Rotate a full turn
  moveSteps(true, 32 * 64, 2);
  delay(1000);
  // Rotate a full turn towards another direction
  moveSteps(false, 32 * 64, 2);
  delay(1000);
}

void moveSteps(bool dir, int steps, byte ms) {
  for (int i = 0; i < steps; i++) {
    moveOneStep(dir); // Rotate a step
    delay(ms);        // Control the speed
  }
}

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

```
40      digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
41    }
42  }
```

In the code, we define a function to make the motor rotate for a step. And the parameter determines the rotation direction of the stepper motor.

```
28  void moveOneStep(bool dir) {
...   ...
42  }
```

A variable is defined in this function and we use four low bits to show the state of 4 ports. These ports are connected in order, so the variable can be assigned to 0x01 and we can use the shifting method to change the bit of the connected port.

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

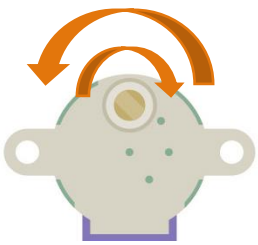
Then change the state of the port according to the above variables.

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

We define a function to control the step motor to rotate several steps and control the direction and speed through parameters. Call it directly in the loop () function.

```
21  void moveSteps(bool dir, int steps, byte ms) {
22    for (int i = 0; i < steps; i++) {
23      moveOneStep(dir); // Rotate one step
24      delay(ms);        // Contorl the speed
25    }
26  }
```

Verify and upload the code, and you will see the step motor rotate a full turn, and then repeat this process in a reverse direction.



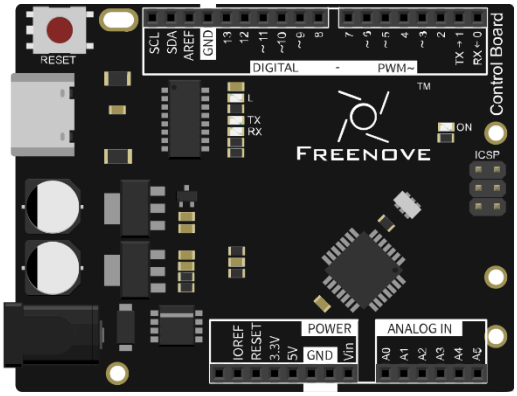
Chapter 19 Matrix Keypad

We've already learned and used a push button switch before, now let us try to use a keypad, a device integrated with a number of Push Button Switches as Keys for the purposes of Input.

Project 19.1 Get Input Characters

First, try to understand the keypad, and get the input characters.

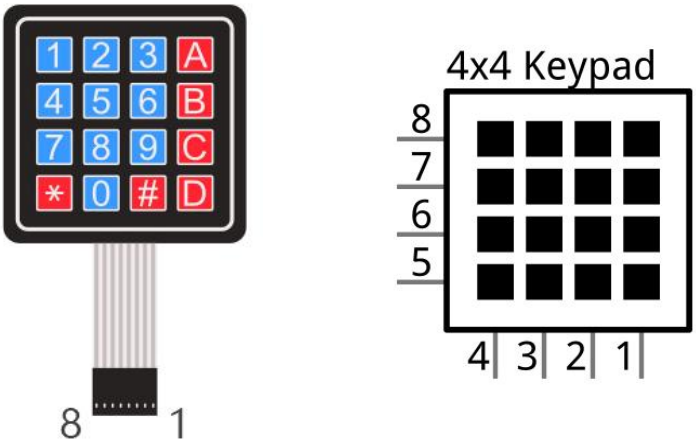
Component List

Control board x1 	4x4 keypad x1 
USB cable x1 	
Jumper M/M x8 	

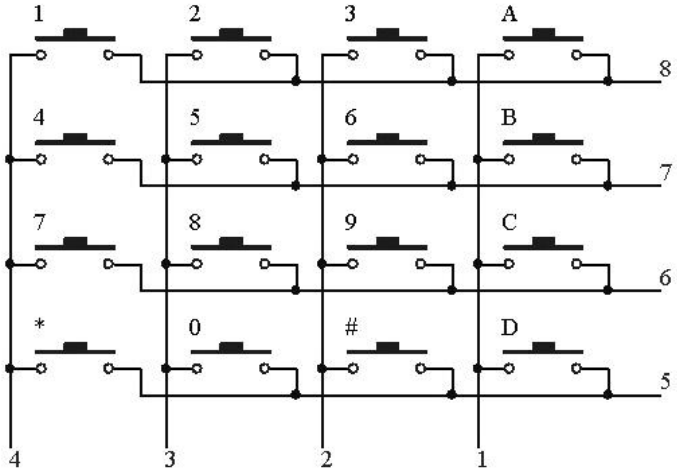
Component Knowledge

4x4 keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad Matrix integrates 16 keys (think of this as 16 Push Button Switches in one module):



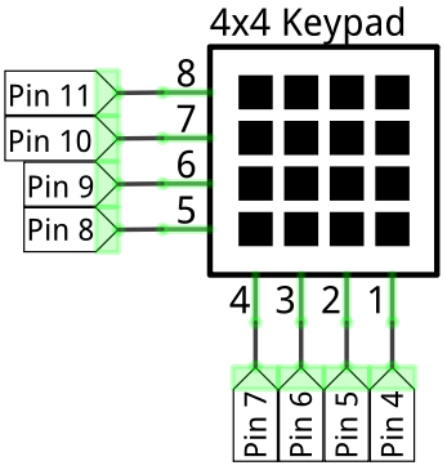
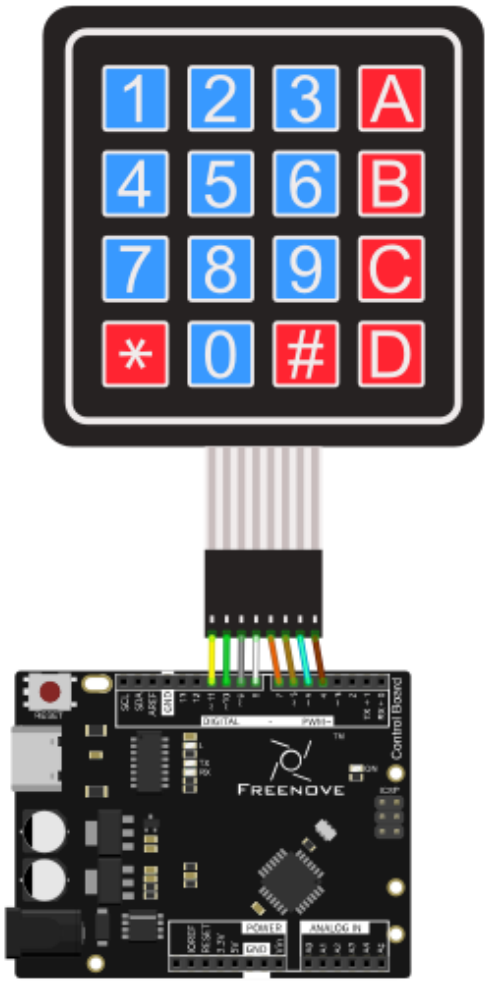
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



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

Circuit

Use pin 11-4 on control board to connect 4x4 keypad.

Schematic diagram	Hardware connection
	

Sketch

Sketch 19.1.1

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

Click “Add .ZIP Library...” and then find Keypad.zip in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library can facilitate our operation of keypad.

Now write the code to obtain the keypad characters, and send them to the serial port.

```
#include <Keypad.h>

// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
byte colPins[4] = {7, 6, 5, 4};   // connect to the column pinouts of the keypad

// initialize an instance of class Keypad
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);

void setup() {
  Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
}

void loop() {
  // Get the character input
  char keyPressed = myKeypad.getKey();
  // If there is a character input, sent it to the serial port
  if (keyPressed) {
    Serial.println(keyPressed);
  }
}
```

In the code, we use a Keypad class provided by the Keypad library to operate the keypad. The following code is to instantiate a keypad object, and the last two parameters represent the number of the row and column of the keypad.

```
6  Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
```

The two-dimensional arrays record the keypad characters, and these characters can be returned when you press the keyboard.

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

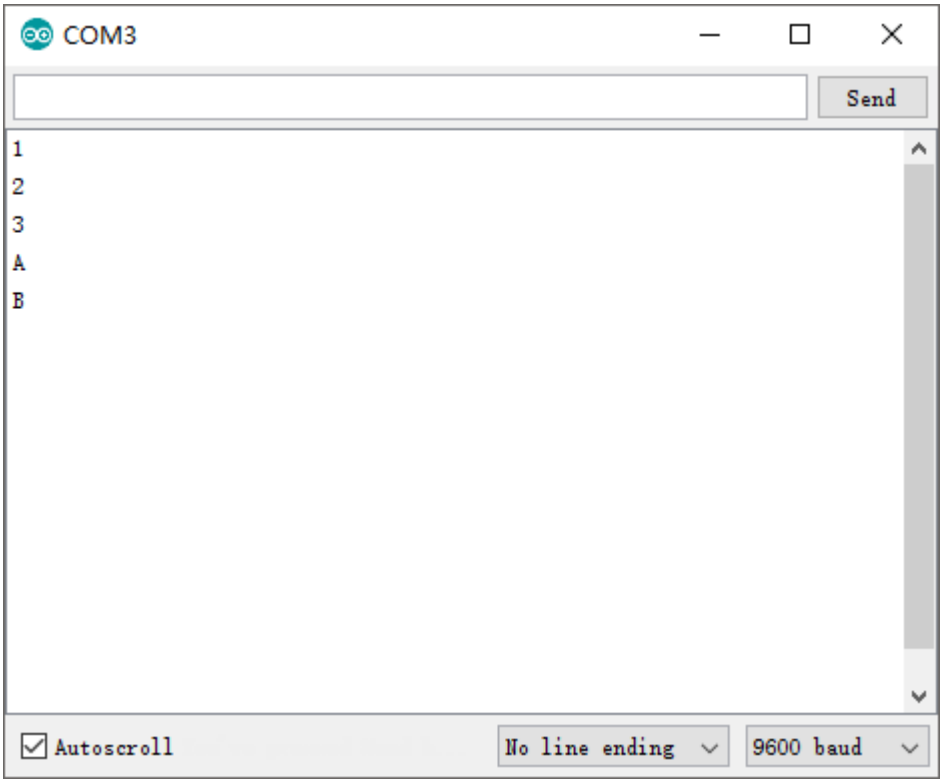
These two arrays record the row and column's connection pins of keypad.

```
11 byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
12 byte colPins[4] = {7, 6, 5, 4};   // connect to the column pinouts of the keypad
```

Send the input that get from the keyboard to the computer via the serial port in function loop().

```
21 void loop() {
22   // Get the character input
23   char keyPressed = myKeypad.getKey();
24   // If there is a character input, sent it to the serial port
25   if (keyPressed) {
26     Serial.println(keyPressed);
27   }
28 }
```

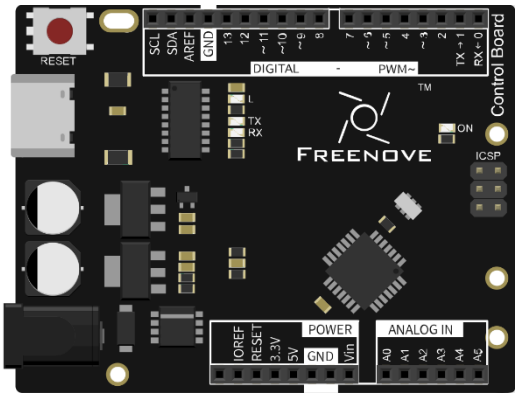
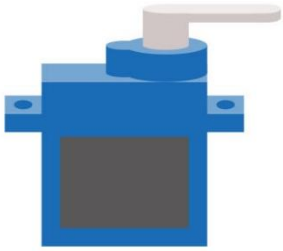
Verify and upload the code, open the Serial Monitor and press the keypad, and then you will see the characters you press being printed out.



Project 19.2 Combination Lock

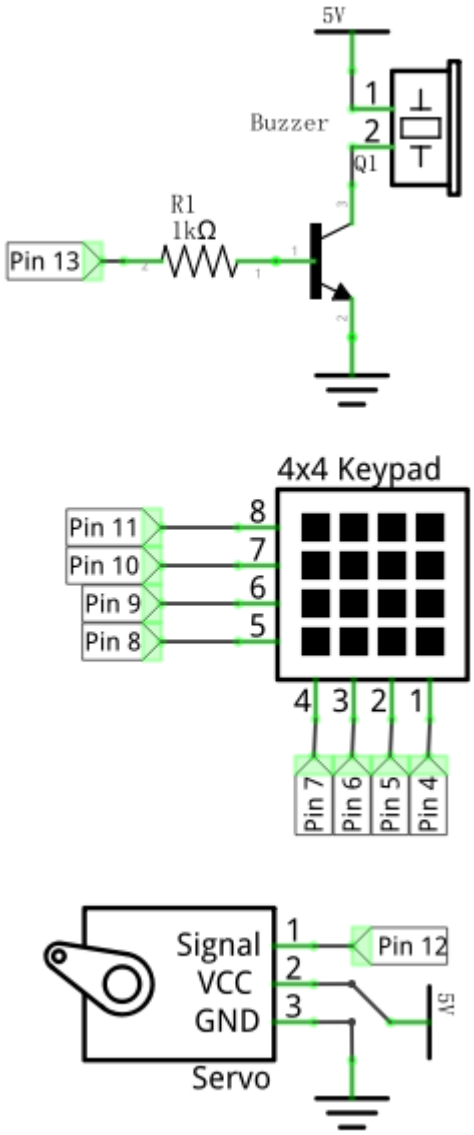
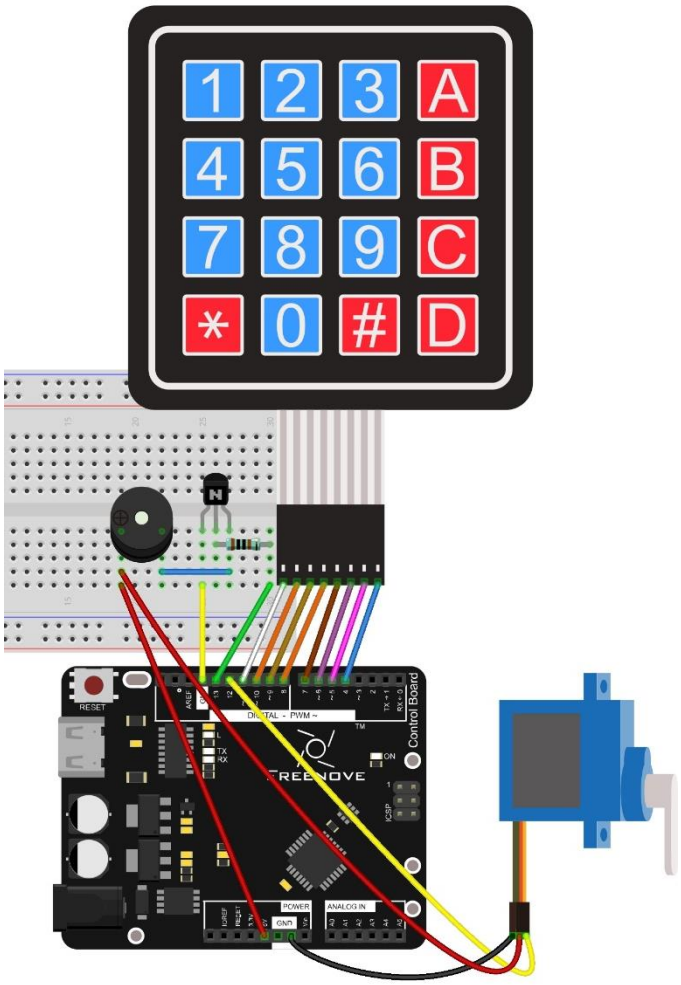
Now, let us try to use keypad to make a combination lock.

Component List

<table>
<tr><td>Control board x1
  </td><td rowspan="3">4x4 keypad x1
  </td></tr>
<tr><td>USB cable x1
  </td></tr>
<tr><td>Jumper M/M x15
  </td></tr>
<tr><td>NPN transistor x1
  </td><td>Active buzzer x1
  </td><td>Resistor 1kΩ x1
  </td><td>Servo x1
  </td></tr>
</table>

Circuit

Use pin 11-4 on the control board to connect 4x4 keypad, pin 13 to drive buzzer and pin 12 to drive servo. Servos can be used to drive the mechanical switch and turn on the switch when the password is correct.

Schematic diagram	Hardware connection
	

Sketch

Sketch 19.2.1

Now write the code to obtain the keypad characters, and compare them with the preset password. If the input is correct, the servo moves, otherwise the buzzer makes an input error alarm.

```
1  #include <Keypad.h>
2  #include <Servo.h>
3
4  // define the symbols on the buttons of the keypad
```

```
5   char keys[4][4] = {
6     {'1', '2', '3', 'A'},
7     {'4', '5', '6', 'B'},
8     {'7', '8', '9', 'C'},
9     {'*', '0', '#', 'D'}
10  };
11
12  byte rowPins[4] = {11, 10, 9, 8}; // connect to the row pinouts of the keypad
13  byte colPins[4] = {7, 6, 5, 4};   // connect to the column pinouts of the keypad
14
15  // initialize an instance of class NewKeypad
16  Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);
17
18  Servo myservo;      // Create servo object to control a servo
19  int servoPin = 12;  // Define the servo pin
20  int buzzerPin = 13; // Define the buzzer pin
21
22  char passWord[] = {'1', '2', '3', '4'}; // Save the correct password
23
24  void setup() {
25    myservo.attach(servoPin);   // Attaches the servo on servoPin to the servo object
26    myservo.write(45);          // Let the steering gear move to the start position
27    pinMode(buzzerPin, OUTPUT);
28  }
29
30  void loop() {
31    static char keyIn[4];     // Save the input character
32    static byte keyInNum = 0; // Save the number of input characters
33    char keyPressed = myKeypad.getKey();  // Get the character input
34    // Handle the input characters
35    if (keyPressed) {
36      // Make a prompt tone each time press the key
37      digitalWrite(buzzerPin, HIGH);
38      delay(100);
39      digitalWrite(buzzerPin, LOW);
40      // Save the input characters
41      keyIn[keyInNum++] = keyPressed;
42      // Judge the correctness after input
43      if (keyInNum == 4) {
44        bool isRight = true;            // Save password is correct or not
45        for (int i = 0; i < 4; i++) {   // Judge each character of the password is correct
    or not
46          if (keyIn[i] != passWord[i])
47
```

```
            isRight = false;              // Mark wrong password if there is any wrong
48  character
49        }
50        if (isRight) {                  // If the input password is right
51          myservo.write(135);           // Open the switch
52          delay(2000);                  // Delay a period of time
53          myservo.write(45);            // Close the switch
54        }
55        else {                          // If the input password is wrong
56          digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
57          delay(500);
58          digitalWrite(buzzerPin, LOW);
59        }
60        keyInNum = 0; // Reset the number of the input characters to 0
61      }
62    }
  }
```

Based on the last section, this code adds a comparison function, the corresponding action will be different whenever the password is right or wrong.

First we define a correct password with four characters.

```
22  char passWord[] = {'1', '2', '3', '4'}; // save the right password
```

Make a prompt sound every time when a key is pressed and save the pressed characters.

```
35    if (keyPressed) {
36      // Make a prompt tone each time press the key.
37      digitalWrite(buzzerPin, HIGH);
38      delay(100);
39      digitalWrite(buzzerPin, LOW);
40      // Save the input characters
41      keyIn[keyInNum++] = keyPressed;
...   ...
61    }
```

Compare with the preset password after 4 characters are input and adopt the corresponding operation.

```
43      if (keyInNum == 4) {
44        bool isRight = true;              // Sav password is correct or not
45        for (int i = 0; i < 4; i++) {     // Judge each character of the password is correct
    or not
46          if (keyIn[i] != passWord[i])
47            isRight = false;              // Mark wrong passageword if there is any wrong
    character.
48        }
49        if (isRight) {                    // If the input password is right
50          myservo.write(135);             // Open the switch
51          delay(2000);                    // Delay a period of time
52          myservo.write(45);              // Close the switch
53        }
54        else {                            // If the input password is wrong
55          digitalWrite(buzzerPin, HIGH);// Make a wrong password prompt tone
56          delay(500);
57          digitalWrite(buzzerPin, LOW);
58        }
59        keyInNum = 0; // Reset the number of the input characters to 0.
60      }
```

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

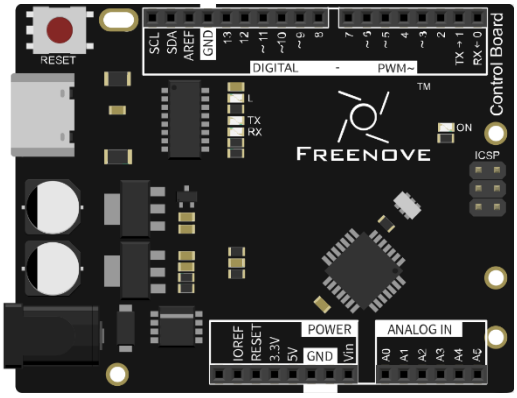
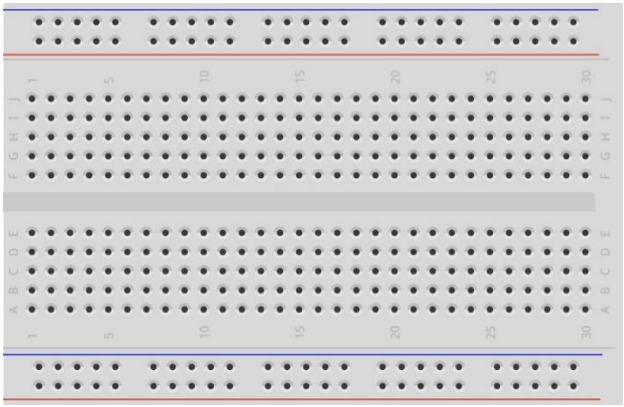
Chapter 20 Vibration Switch

Vibration switch is a component that can detect vibration. We will use this sensor later.

Project 20.1 Detect Vibration

Now let us try to use vibration switch to detect vibration.

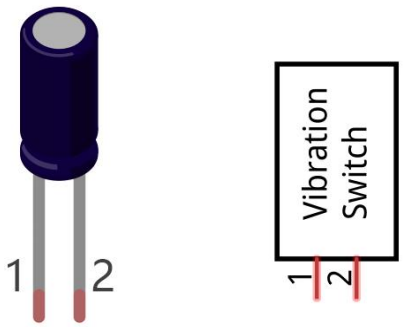
Component List

Control board x1 		Breadboard x1 	
USB cable x1 		Jumper F/M x2  Jumper M/M x4 	
NPN transistorx1 	Active buzzer x1 	Resistor 1kΩ x1 	Vibration switch x1 

Component Knowledge

Vibration Switch

Vibration switches is a kind of sensor that can detect vibration. When the vibration amplitude is greater than the critical value, two pins of the vibration switch will be switched on.



The internal circuit of the Vibration switch is as below, one of the pins is connected with a metal bar and the other is connected with spring. The spring will swing when a vibration occurs. When the vibration is strong enough, the spring will contact with the metal bar to make the two pins connected with each other.

Circuit Knowledge

Digital pins with interrupts

Some of the control board digital pins can be configured to interrupt mode, which can trigger interrupt event and execute interrupt function.

Pin 2 and 3 of the control board can be configured to interrupt mode. Conditions that trigger interrupt can be configured to:

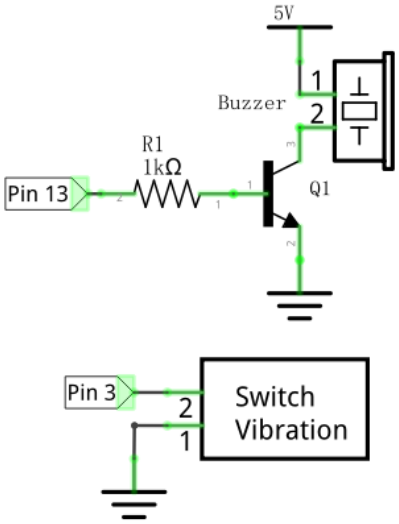
Condition	Function
LOW	to trigger the interrupt whenever the pin is low
CHANGE	to trigger the interrupt whenever the pin changes value
RISING	to trigger when the pin goes from low to high
FALLING	to trigger for when the pin goes from high to low

Interrupts are useful for making things happen automatically in microcontroller programs, and they can help solve timing problems. Ideal tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

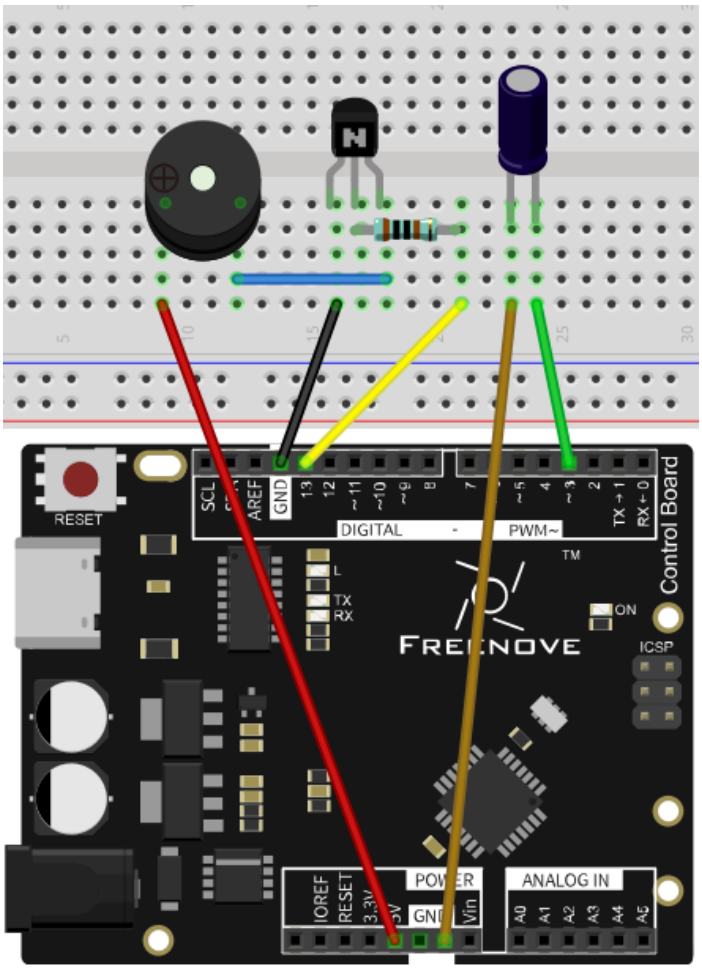
Circuit

Use pin 3 on the control board to connect vibration switch, and pin 13 to drive buzzer.

Schematic diagram



Hardware connection



Sketch

Sketch 20.1.1

Now write code to detect whether the vibration switch is conducted and whether it makes a buzzer sound when conducted.

```
int buzzerPin = 13; // Define the buzzer pin
int switchPin = 3;  // Define the vibration switch pin, which can cause interrupt

bool isVibrate = false;

void setup() {
  pinMode(buzzerPin, OUTPUT);        // Set the buzzer pin to output mode
  pinMode(switchPin, INPUT_PULLUP); // Set the vibration switch pin to pull up input mode
  // Set interrupt, if vibration switch pin change from high level to low level, vibrate function will be called
  attachInterrupt(digitalPinToInterrupt(switchPin), vibrate, FALLING);
}

void loop() {
  if (isVibrate) {  // If the interrupt function is triggered
    isVibrate = false;  // Marked no trigger again
    digitalWrite(buzzerPin, HIGH);  // Open the buzzer
    delay(50);          // Delay for a period of time
  }
  else              // Else close the buzzer
    digitalWrite(buzzerPin, LOW);
}

void vibrate() {
  isVibrate = true; // Marked as the trigger
}
```

This code configures the pin that is connected to vibration switch to be triggered by falling edge, that is, FALLING mode.

```
  attachInterrupt(digitalPinToInterrupt(switchPin), vibrate, FALLING);
```

attachInterrupt(interrupt, ISR, mode);

This function is used to configure the digital pin's interrupt mode. Each parameter of the function is as follows:

interrupt: the number of the interrupt (int). Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number.

ISR: the ISR to be called when an interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered.

And we configure the pins that connected with vibration switch into pull up input mode. This way can ensure a high level of vibration switch when not connected, and low level when connected. So it can cause interrupt.

```
8    pinMode(switchPin, INPUT_PULLUP); // Set the vibration switch pin to pull up input mode
```

The following is the interrupt function, and it will be executed when the interrupt is triggered.

```
23  void vibrate() {
24    isVibrate = true; // Marked as the trigger
25  }
```

Interrupt function should keep short, so we use a variable "isVibrate" to record whether the interrupt is triggered, and dispose it in the loop() function. And the buzzer will be connected if the interrupt is triggered.

```
14    if (isVibrate) {  // If the interrupt function is triggered.
15      isVibrate = false;  // Marked no trigger again
16
17      digitalWrite(buzzerPin, HIGH);  // Open the buzzer
18      delay(50);          // Delay for a period of time
19    }
20    else              // Else close the buzzer
        digitalWrite(buzzerPin, LOW);
```

Verify and upload the code and tap on the vibration switch, and then the buzzer will make a sound.

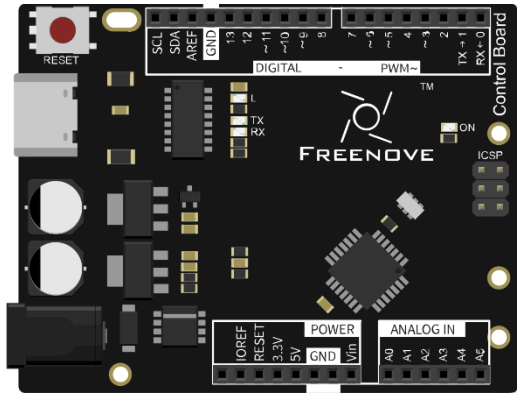
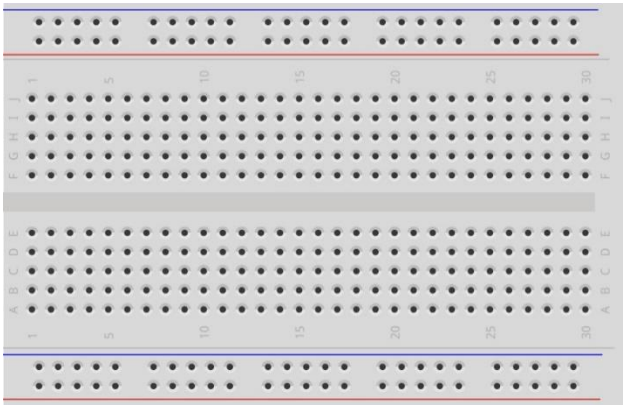
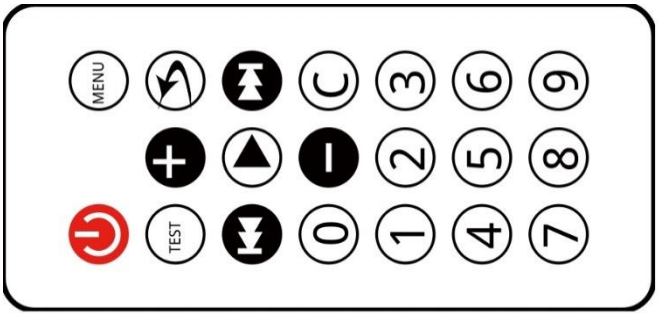
Chapter 21 Infrared Remote

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

Project 21.1 Infrared Remote Control

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

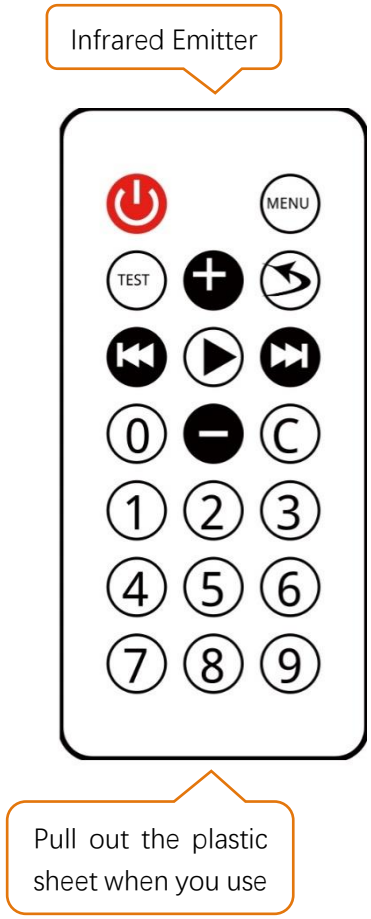
Component List

Control board x1 	Breadboard x1 	
USB cable x1 	Jumper M/M x4 	
Infrared remote x1 (May need CR2025 battery x1, please check the holder) 	Infrared receiver x1 	Resistor 10kΩ x1 

Component Knowledge

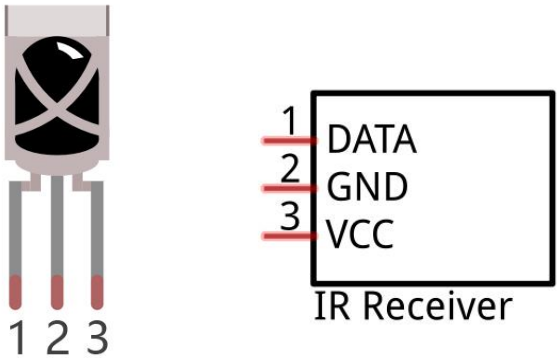
Infrared remote

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



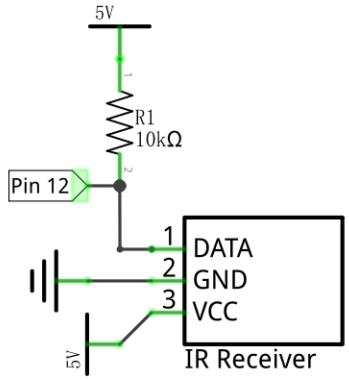
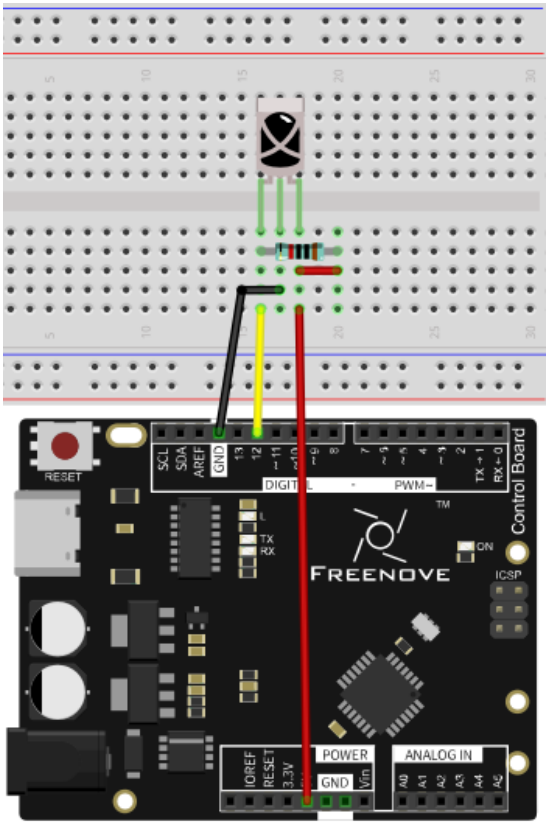
Infrared receiver

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



Circuit

Use pin 12 on the control board to connect IR receiver.

Schematic diagram	Hardware connection
	

Sketch

Sketch 21.1.1

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

Click “Add .ZIP Library...” and then find IRremote.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library makes it easy to control IR receiver.

Now, write code to get the command sent from IR remote control, and send it to the serial port.

```
#include <IRremote.h>

int RECV_PIN = 12;        // Infrared receiving pin
IRrecv irrecv(RECV_PIN);  // Create a class object used to receive class
decode_results results;   // Create a decoding results class object

void setup()
{
  Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
  irrecv.enableIRIn(); // Start the receiver
}
```

```
12
13  void loop() {
14    if (irrecv.decode(&results)) {  // Waiting for decoding
15      Serial.println(results.value, HEX); // Print out the decoded results
16      irrecv.resume(); // Receive the next value
17    }
18    delay(100);
19  }
```

We use the IRrecv class provided by the IRremote library to control IR receiver in this code. As shown below, instantiate one IRrecv object, and the parameter represents the pin connected to the IR receiver.

```
4   IRrecv irrecv(RECV_PIN); // Create a class object used to receive class
```

decode_results class provided by the IRremote library is used to save the results of IR control decoding.

```
5   decode_results results;  // Create a decoding results class object
```

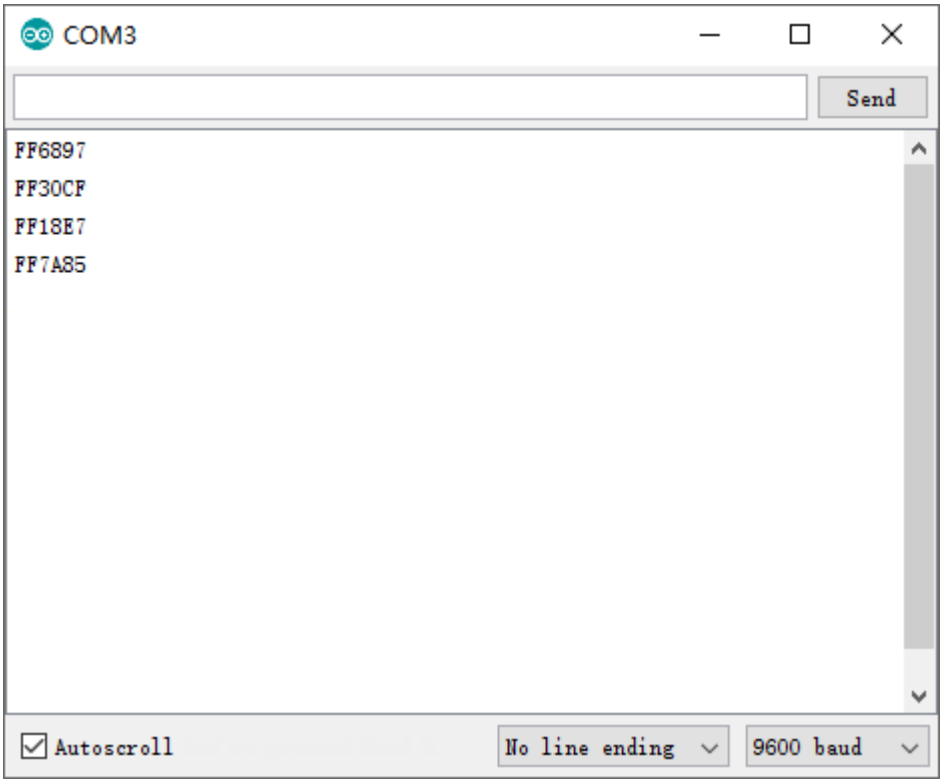
Start the signal receiving in the setup() function

```
10    irrecv.enableIRIn();   // Start the receiver
```

In the loop() function, decode the received signal, and sent it to computer through the serial port.

```
13  void loop() {
14    if (irrecv.decode(&results)) {  // Waiting for decoding
15      Serial.println(results.value, HEX); // Print out the decoded results
16      irrecv.resume(); // Receive the next value
17    }
18    delay(100);
19  }
```

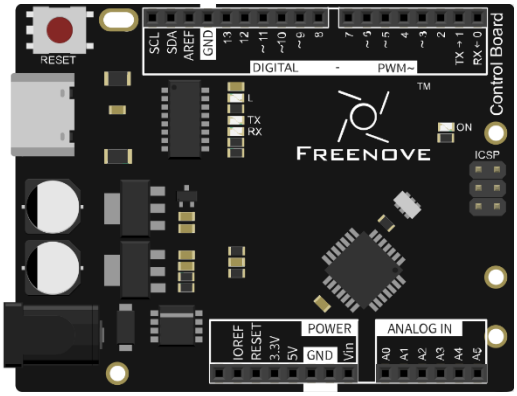
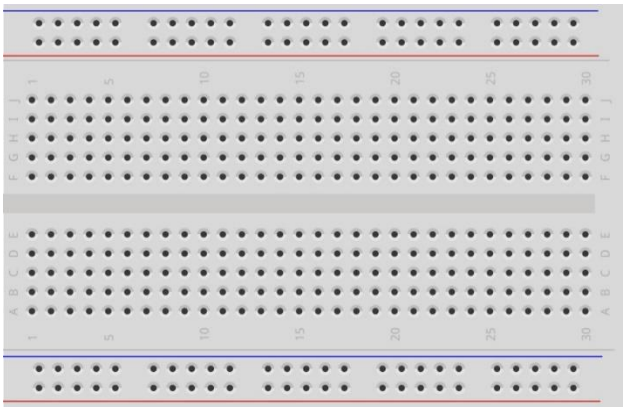
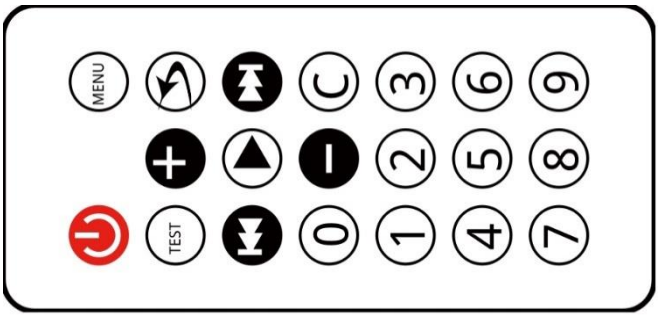
Verify and upload the code, open the Serial Monitor, and press the IR control button, and then you can see the corresponding code being printed out.



Project 21.2 Control LED through Infrared Remote

Now, let us try to control a LED through infrared remote.

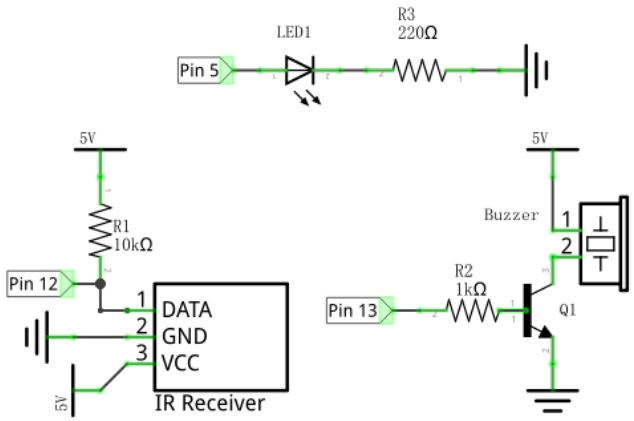
Component List

Control board x1 			Breadboard x1 			
USB cable x1 			Infrared remote x1 			
Jumper M/M x1 						
NPN transistor x1 	Active buzzer x1 	Infrared receiver x1 	LED x1 	Resistor 220Ω x1 	Resistor 1kΩ x1 	Resistor 10kΩ x1 

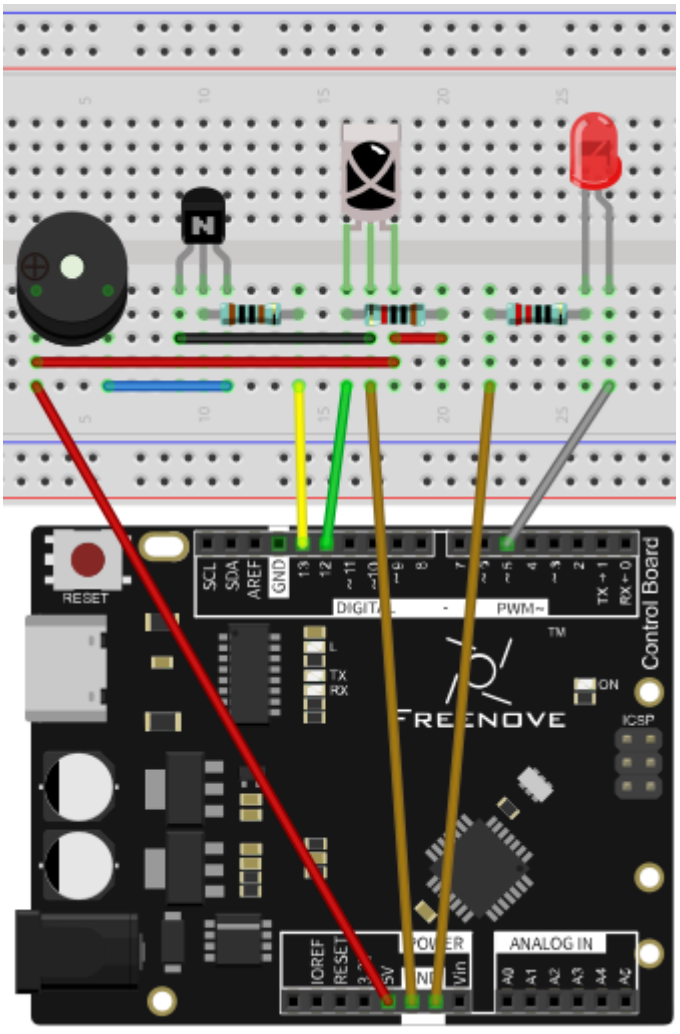
Circuit

Connect pin 12 on the control board to IR receiver to simulate a desk lamp. And drive buzzer through pin 13, drive LED through pin 5.

Schematic diagram



Hardware connection



Sketch

Sketch 21.2.1

Now, write code to get the commands sent from IR remote, and control the LED light on/off or emit light with different brightness, and control the buzzer to generate a confirmation sound when it receives the command.

```
#include <IRremote.h>

int RECV_PIN = 12;        // Infrared receiving pin
IRrecv irrecv(RECV_PIN);  // Create a class object used to receive class
decode_results results;   // Create a decoding results class object

int ledPin = 5;           // the number of the LED pin
int buzzerPin = 13;       // the number of the buzzer pin

void setup()
{
  Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
  irrecv.enableIRIn(); // Start the receiver
  pinMode(ledPin, OUTPUT);          // set LED pin into output mode
  pinMode(buzzerPin, OUTPUT);       // set buzzer pin into output mode
}

void loop() {
  if (irrecv.decode(&results)) {  // Waiting for decoding
    Serial.println(results.value, HEX); // Print out the decoded results
    handleControl(results.value); // Handle the commands from remote control
    irrecv.resume(); // Receive the next value
  }
}

void handleControl(unsigned long value) {
  // Make a sound when it receives commands
  digitalWrite(buzzerPin, HIGH);
  delay(100);
  digitalWrite(buzzerPin, LOW);
  // Handle the commands
  switch (value) {
    case 0xFF6897:              // Receive the number '0'
      analogWrite(ledPin, 0);   // Turn off LED
      break;
    case 0xFF30CF:              // Receive the number '1'
      analogWrite(ledPin, 7);   // Dimmest brightness
      break;
```

```
39      case 0xFF18E7:              // Receive the number '2'
40        analogWrite(ledPin, 63);  // Medium brightness
41        break;
42      case 0xFF7A85:              // Receive the number '3'
43        analogWrite(ledPin, 255); // Strongest brightness
44        break;
45    }
46  }
47
```

Based on the last section, we add some new functions: control LED and buzzer.

We define a function that is used to handle the received commands. When this function is executed, make the buzzer beep first, then output PWM signals with different duty cycle to the pin connected to LED according to the receiving commands

```
26  void handleControl(unsigned long value) {
27    // Make a sound when it rereives commands
28    digitalWrite(buzzerPin, HIGH);
29    delay(100);
30    digitalWrite(buzzerPin, LOW);
31    // Handle the commands
32    switch (value) {
33      case 0xFF6897:              // Receive the number '0'
34        analogWrite(ledPin, 0);   // Turn off LED.
35        break;
36      case 0xFF30CF:              // Receive the number '1'
37        analogWrite(ledPin, 7);   // Dimmest brightness.
38        break;
39      case 0xFF18E7:              // Receive the number '2'
40        analogWrite(ledPin, 63);  // Medium brightness.
41        break;
42      case 0xFF7A85:              // Receive the number '3'
43        analogWrite(ledPin, 255); // Strongest brightnss.
44        break;
45    }
46  }
```

Each time when the command is received, the function above will be called in the loop() function.

```
18  void loop() {
19    if (irrecv.decode(&results)) {  // Waiting for decoding
20      Serial.println(results.value, HEX); // Print out the decoded results
21      handleControl(results.value); // Handle the commands from remote control
22      irrecv.resume(); // Receive the next value
23    }
24  }
```

Verify and upload the code, press the button '0', '1', '2', '3' on IR remote, and then you can see LED emit light with different brightness, and the buzzer will start beeping when it receives the signal.

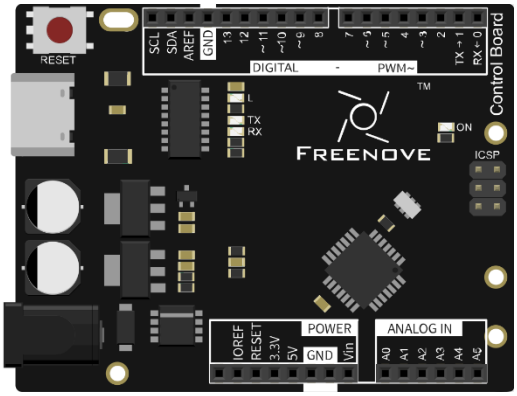
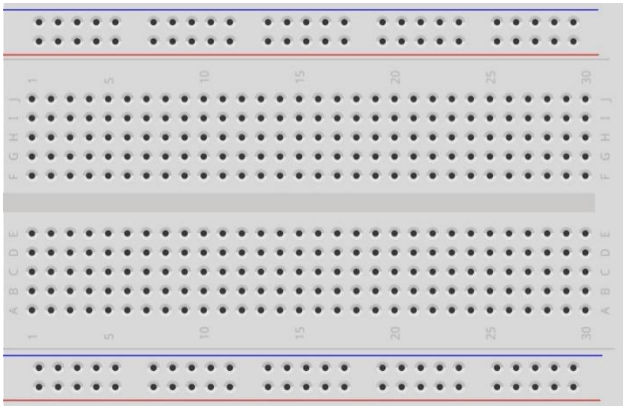
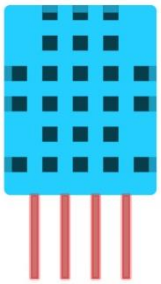
Chapter 22 Temperature & Humidity Sensor

In this chapter, we will learn how to use a sensor that can detect temperature and humidity.

Project 22.1 Temperature & Humidity Sensor

Now, let's try to get the temperature and humidity value of the current environment.

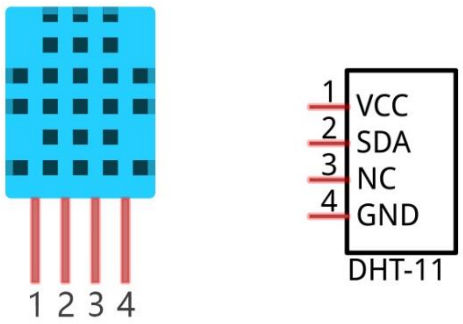
Component List

Control board x1 	Breadboard x1 	
USB cable x1 	Resistor 10kΩ x1 	DHT11 x1 
Jumper M/M x4 		

Component Knowledge

DHT11

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

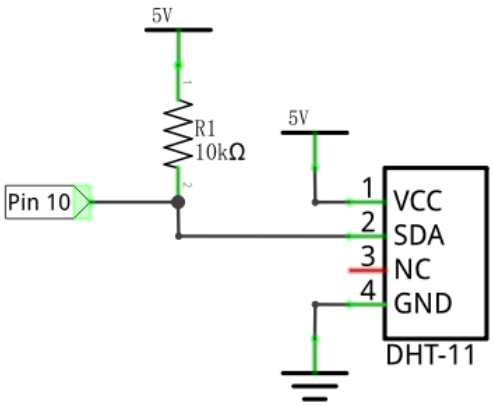


DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

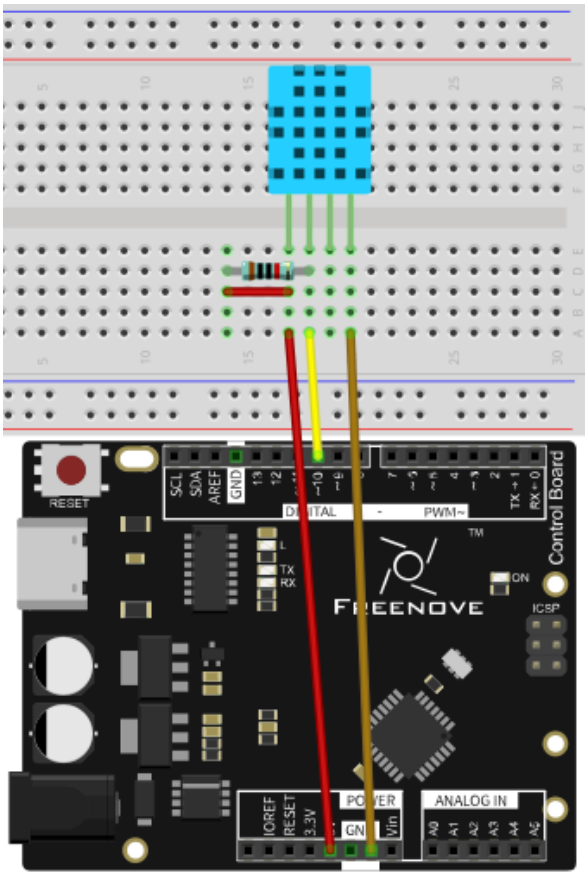
Circuit

Use pin 10 on control board to connect DHT11.

Schematic diagram



Hardware connection



Sketch

Sketch 22.1.1

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

Click "Add .ZIP Library..." and then find DHT.zip in libraries folder (this folder is in the folder unzipped from the ZIP file we provided). This library can make it convenient for us to control DHT11.

Now, write code to get the temperature and humidity data measured by DHT11, and sent it to the serial port.

```
#include <dht.h>

dht DHT;          // create dht object
int dhtPin = 10;  // the number of the DHT11 sensor pin

void setup() {
  Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
}
void loop() {
  // read DHT11 and judge the state according to the return value
  int chk = DHT.read11(dhtPin);
  switch (chk)
```

```
13    {
14      case DHTLIB_OK: // When read data successfully, print temperature and humidity data
15        Serial.print("Humidity: ");
16        Serial.print(DHT.humidity);
17        Serial.print("%, Temperature: ");
18        Serial.print(DHT.temperature);
19        Serial.println("C");
20        break;
21      case DHTLIB_ERROR_CHECKSUM: // Checksum error
22        Serial.println("Checksum error");
23        break;
24      case DHTLIB_ERROR_TIMEOUT:  // Time out error
25        Serial.println("Time out error");
26        break;
27      default:                    // Unknown error
28        Serial.println("Unknown error");
29        break;
30    }
31    delay(1000);
32  }
```

In the code, we use the dht class provided by the DHT library to control DHT11. The following is a DHT object. As shown below, instantiate one dht object.

```
3   dht DHT;          // create dht object
```

Read DHT11 data and send the result to the serial port in the loop() function.

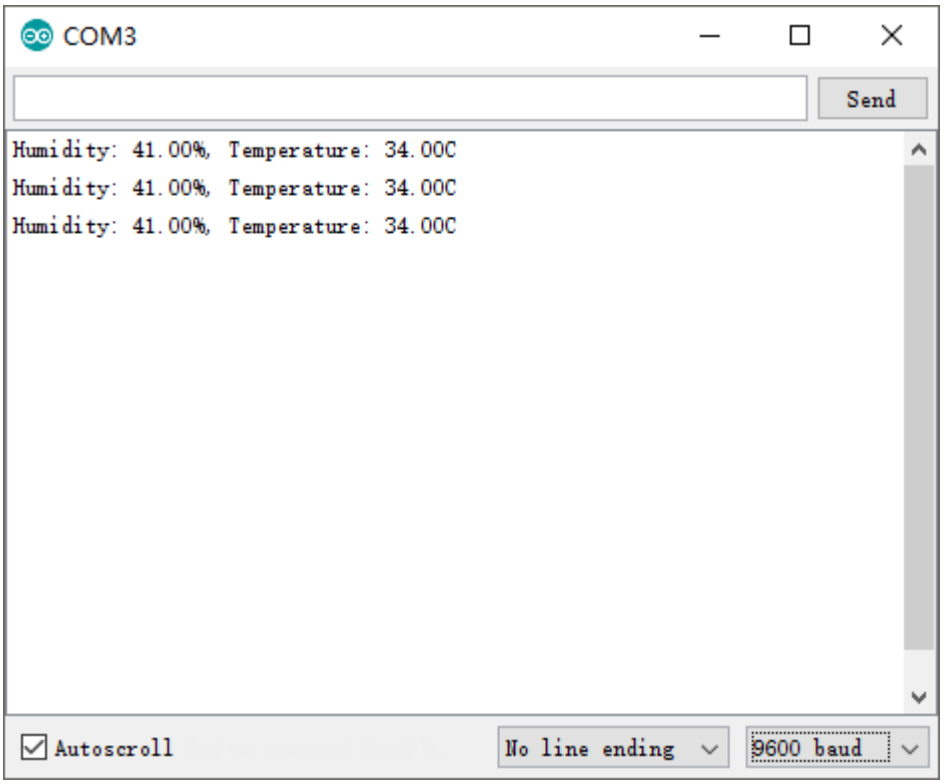
```
11    // read DHT11 and judge the state according to the return value
12    int chk = DHT.read11(dhtPin);
13    switch (chk)
14    {
15      case DHTLIB_OK: // When read data successfully print temperature and humidity data
16        Serial.print("Humidity: ");
17        Serial.print(DHT.humidity);
18        Serial.print("%, Temperature: ");
19        Serial.print(DHT.temperature);
20        Serial.println("C");
21        break;
...   ...
31    }
```

Send the failure reason when fail to read.

```
22      case DHTLIB_ERROR_CHECKSUM: // Checksum error
23        Serial.println("Checksum error");
24        break;
25      case DHTLIB_ERROR_TIMEOUT:  // Time out error
26        Serial.println("Time out error");
27        break;
```

```
28      default:                        // Unknown error
29        Serial.println("Unknown error");
30        break;
```

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



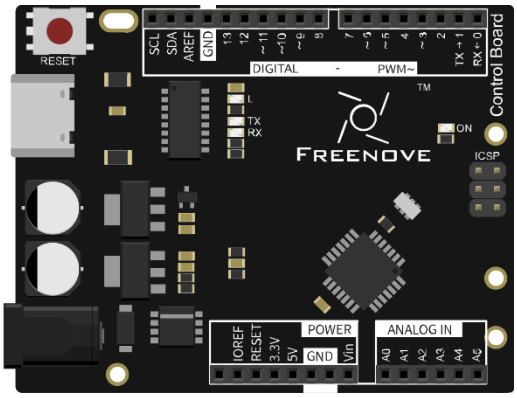
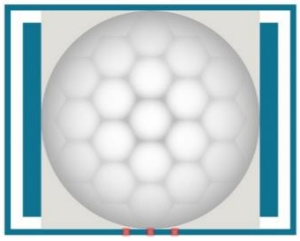
Chapter 23 Infrared Motion Sensor

Infrared Motion Sensor is an integrated sensor that can sense the motion of the human body. Now let us try to use it.

Project 23.1 Infrared Motion Sensor

Now, we'll use Infrared Motion Sensor to detect human motion.

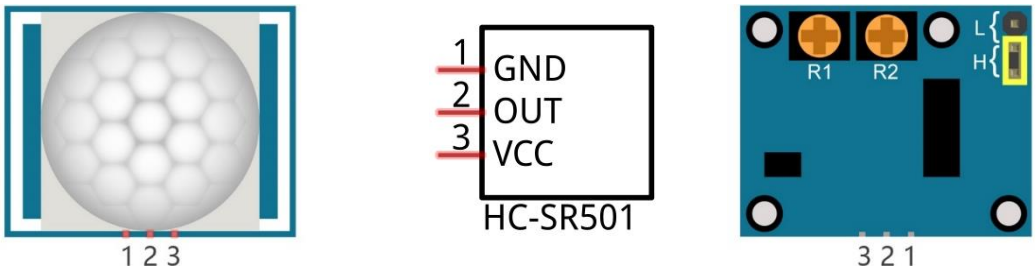
Component List

Control board x1 	USB cable x1 
	Jumper F/M x3 
	Infrared motion sensor x1 

Component Knowledge

Infrared Motion Sensor

Infrared Motion Sensor is an integrated sensor that can sense the motion of a human body. It can detect whether someone is moving in the sensing range by detecting the infrared light emitted by human body. Here is the HC-SR501 infrared motion sensor and its back:



Description:

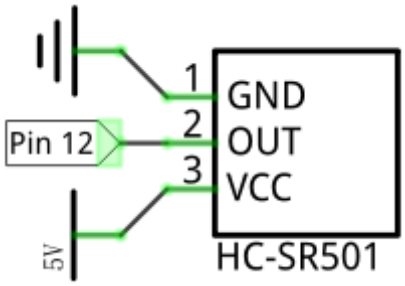
1. Working voltage: 5v-20v(DC), static current: 65uA.
2. Automatic trigger. When a living body enters into the active area of sensor, the module will output high level (3.3V: Though the high level of control board is 5v, 3.3v is identified as high level here). When the body leaves the sensor's active detection area, it will output high level lasting for time period T, then output low level(0V). Delay time T can be adjusted by the potentiometer R1.
3. According to the position of Fresnel lenses dome, you can choose non-repeatable trigger modes or repeatable modes.
   L: non-repeatable trigger mode. The module output high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.
   H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves during the period of high level output. After this, it starts to time and output low level after delaying T time.
4. Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level.
5. Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.
6. One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitively. When a body moves close to or moves away from the sensor's dome in a vertical direction (perpendicular to the dome), the sensor cannot detect well (please take note of this deficiency). Actually this makes sense when you consider that this sensor is usually placed on a celling as part of a security product. Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

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

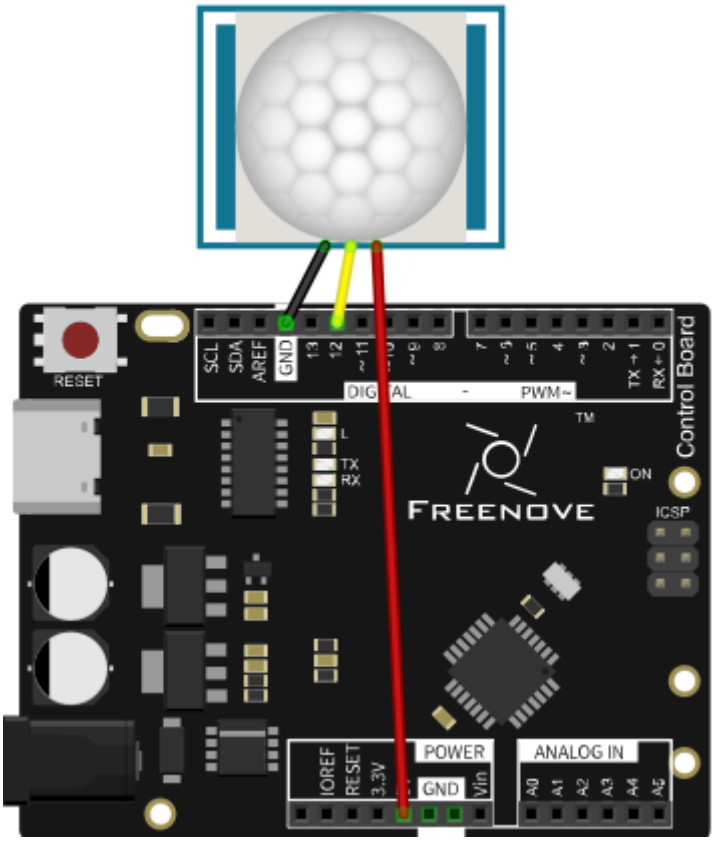
Circuit

Use pin 12 on the control board to connect out-pin of HC-SR501 infrared motion sensor.

Schematic diagram



Hardware connection



Sketch

Sketch 23.1.1

Now, write code to get the results measured by the HC-SR501 infrared motion sensor, and display that through LED.

```
int sensorPin = 12; // the number of the infrared motion sensor pin
int ledPin = 13;    // the number of the LED pin

void setup() {
  pinMode(sensorPin, INPUT);  // initialize the sensor pin as input
  pinMode(ledPin, OUTPUT);    // initialize the LED pin as output
}

void loop() {
   // Turn on or off LED according to Infrared Motion Sensor
  digitalWrite(ledPin, digitalRead(sensorPin));
  delay(1000);              // wait for a second
}
```

This code is relatively simple. We have obtained the sensor's output signal, and control a LED according to it.

```
  digitalWrite(ledPin, digitalRead(sensorPin));
```

Verify and upload the code, put the sensor on a stationary table and wait for about a minute. And then try to get close to or away from the sensor, and observe whether the LED will be turned on or turned off automatically.

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

Apart from that, you can use this sensor to control some other modules to achieve different functions by re-editing the code, such as the induction lamp, induction door.

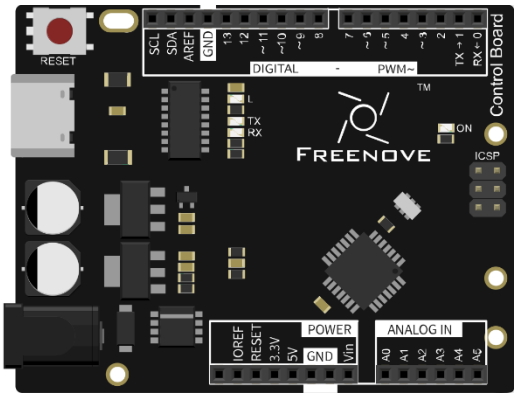
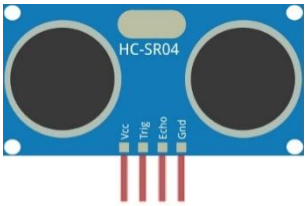
Chapter 24 Ultrasonic Ranging

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

Project 24.1 Ultrasonic Ranging

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

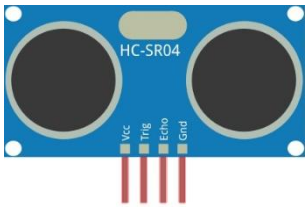
Component List

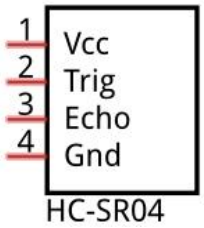
Control board x1 	USB cable x1 
	Jumper F/M x4 
	Ultrasonic ranging module x1 

Component Knowledge

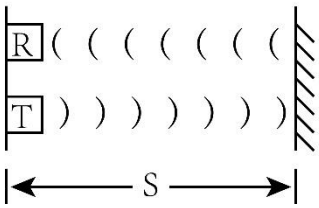
Ultrasonic ranging module

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





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



Pin description:

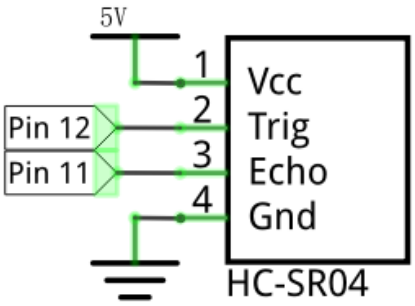
Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Triger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

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

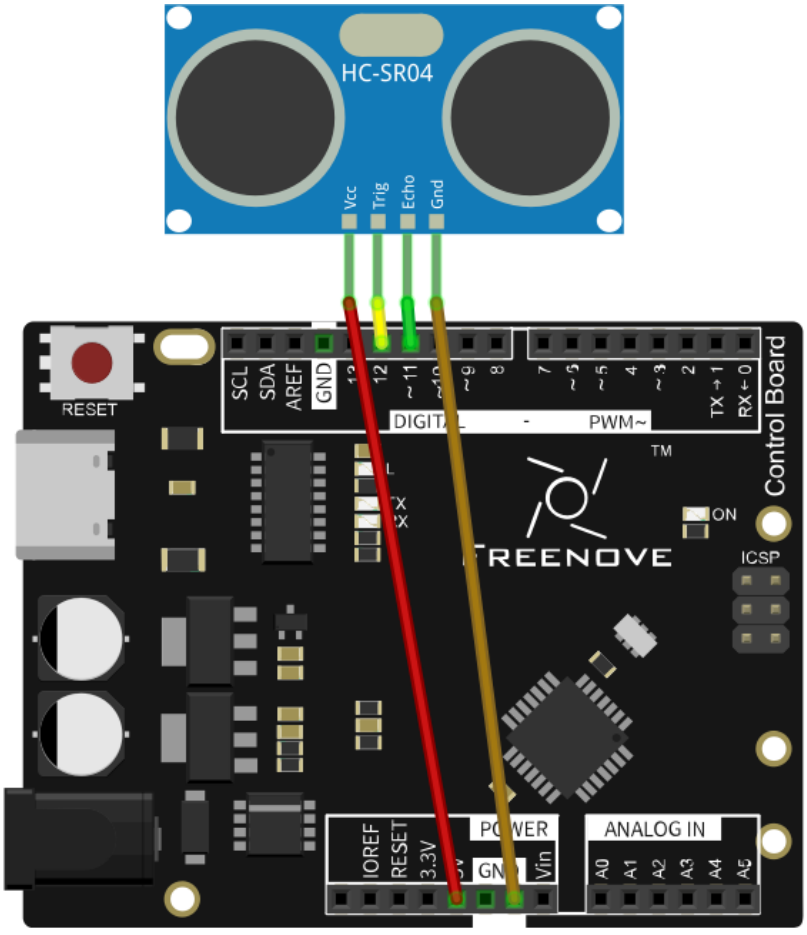
Circuit

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

Schematic diagram



Hardware connection



Sketch

Sketch 24.1.1

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

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

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

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

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

First, define the pins and the maximum measurement distance.

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

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

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

Then, in the setup (), set the pin to input or output, and set the serial port. In the loop(), we continue to use serial to print the value of subfunction getSonar (), which is used to return the measured distance of the HC_SR04. Make trigPin output a high level lasting for at least 10μs, according to the communication protocol.

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

And then the echoPin of HC_SR04 will output a pulse. Time of the pulse is the total time of ultrasonic from transmitting to receiving. We use the pulseIn () function to return the time, and set the timeout.

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

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

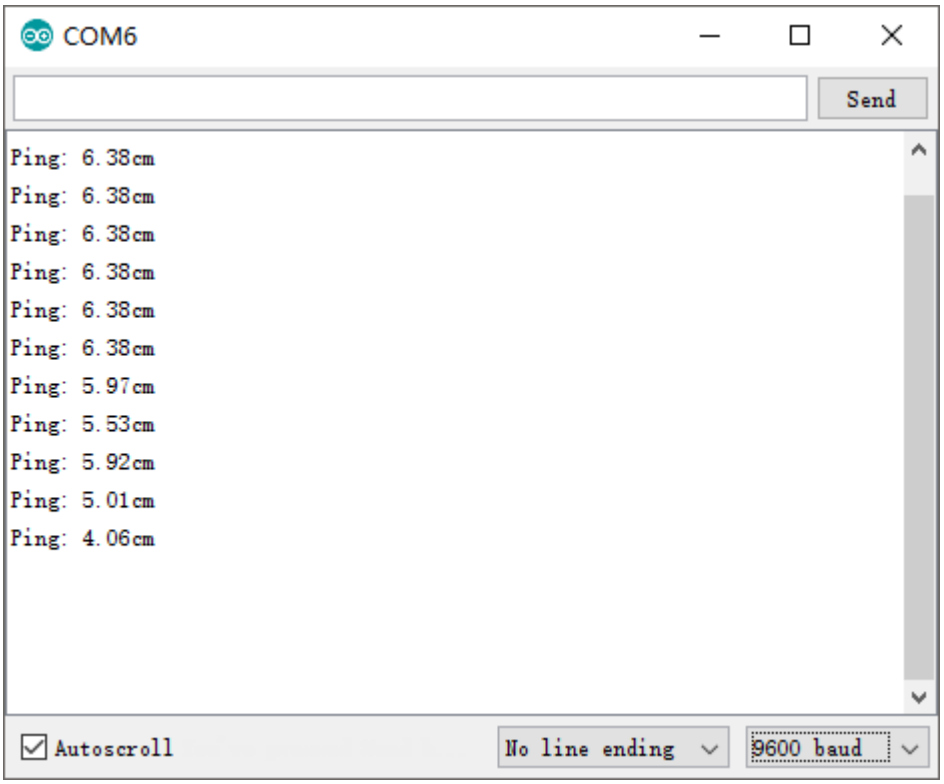
```
  distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance according to the time
  return distance; // return the distance value
```

Finally, the code above will be called in the loop ().

pulseIn(pin, value) / pulseIn(pin, value, timeout)

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

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


COM6
Send
Ping: 6.38cm
Ping: 6.38cm
Ping: 6.38cm
Ping: 6.38cm
Ping: 6.38cm
Ping: 6.38cm
Ping: 5.97cm
Ping: 5.53cm
Ping: 5.92cm
Ping: 5.01cm
Ping: 4.06cm
Autoscroll
No line ending
9600 baud

Sketch 24.1.2

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

Click "Add .ZIP Library..." and then find NewPing.zip in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library makes it easy to obtain the measuring distance.

```
1  #include <NewPing.h>
2  #define trigPin 12 // define TrigPin
3  #define echoPin 11 // define EchoPin.
4  #define MAX_DISTANCE 200 // Maximum sensor distance is rated at 400-500cm.
5
6  NewPing sonar(trigPin, echoPin, MAX_DISTANCE); // NewPing setup of pins and maximum
   distance.
7
8  void setup() {
9    Serial.begin(9600); // Open serial monitor at 9600 bauds to see ping results.
10 }
11
12 void loop() {
13   delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest
   delay between pings.
14   Serial.print("Ping: ");
15   Serial.print(sonar.ping_cm()); // Send ping, get distance in cm and print result (0 =
   outside set distance range)
16   Serial.println("cm");
17 }
```

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

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

And then, in the loop (), use sonar.ping_cm () to obtain the ultrasonic module detection distance with unit of centimeter. And print the distance out. When the distance exceeds range of 2cm~200cm, the printed data is zero.

NewPing Class

NewPing class can be used for SR04, SRF05, SRF06 and other sensors. An object that needs to be instantiated when the class is used. The three parameters of the constructor function are: trigger pin, echo pin and maximum measurement distance.

NewPing sonar(trigger_pin, echo_pin [, max_cm_distance])

Some member function:

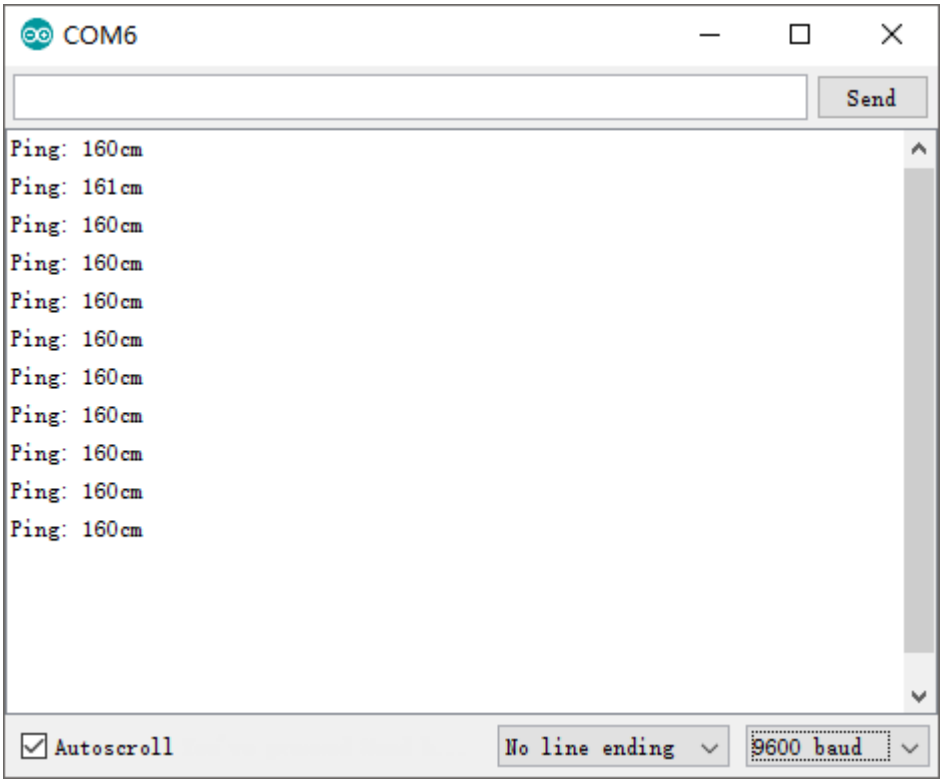
sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

sonar.ping_in() - Send a ping and get the distance in whole inches.

sonar.ping_cm() - Send a ping and get the distance in whole centimeters.

For more details, please refer to the NewPing.h in the NewPing library.

Verify and upload the code to control board and open the Serial Monitor. When you make ultrasonic probe toward a plane of an object, you can observe the distance between them.



Chapter 25 Soldering Circuit Board

From previous chapters, we have learned about electronic circuits and components and have built a variety of circuits using a Breadboard device, which is not designed to be used permanently. We now will take a further step to make permanent projects using a Perfboard (a type of Prototype Circuit Board). Note: Perfboard is a stiff, thin sheet of insulated material with holes bored on a grid. The grid is usually a squared off shape with a spacing of 0.1 inches. Square copper pads cover these holes to make soldering electronic components easier.

To finish this chapter, you need to prepare the necessary soldering equipment, including an electric soldering iron (or soldering pencil) and solder. We have already prepared the Perfboard for you.

CAUTION: Please use extreme caution and attention to safety when you operate soldering tools used in these projects.

Project 25.1 Solder a Buzzer

You should be familiar with the Buzzer from our previous project. We will solder a permanent circuit that when a Push Button Switch is pressed a Buzzer sounds

Note: This circuit **does not** require programming and will work when it is powered ON. When the button is not pressed and the Buzzer is not in use, there is no power consumption.

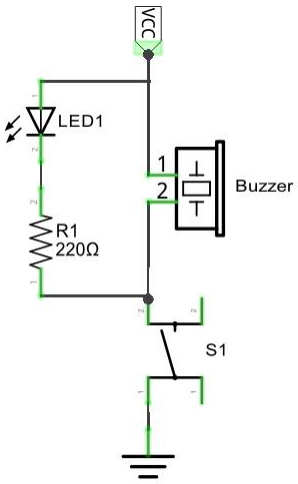
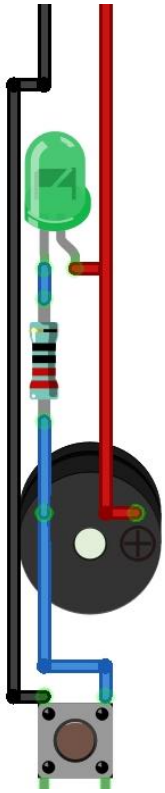
You can install it on your bicycle, your bedroom door or any other place where you want a Buzzer.

Component List

Pin header x2	LED x1	Resistor 220Ω x1	Active buzzer x1	Push button x1
				

Circuit

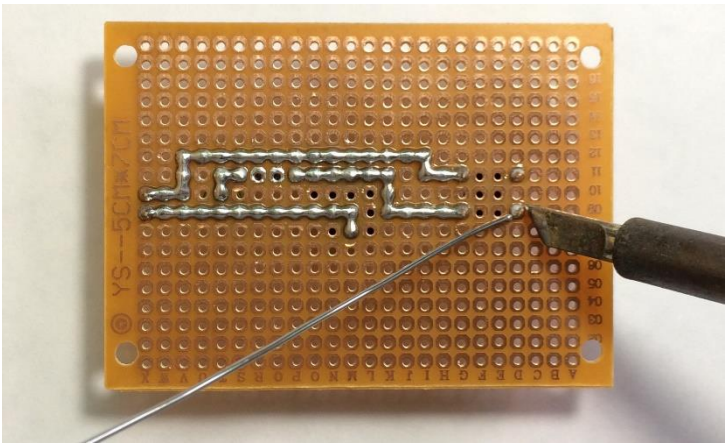
We will solder the following circuit on the Perfboard.

Schematic diagram	Hardware connection
	

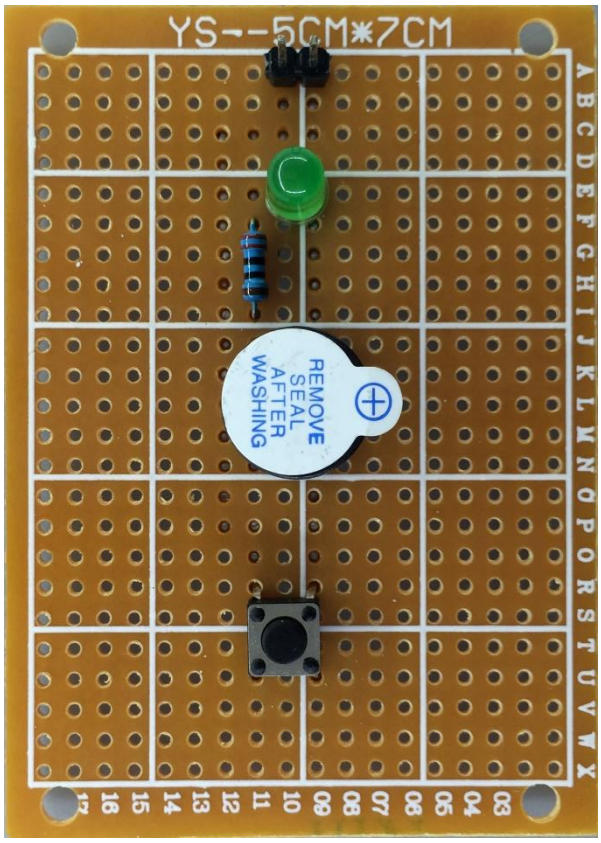
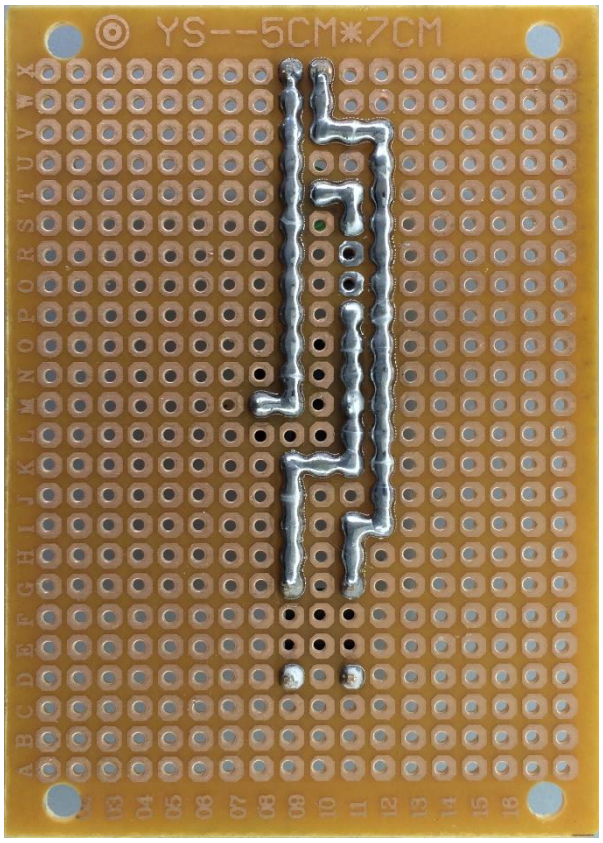
Note: If you are new to soldering electronic components on any type of circuit board we strongly recommend that you watch some instructional How-To videos by doing an Internet search and practice your soldering technique before attempting to solder the following projects. Some components can be damaged by exposure to excessive heat for prolonged times and there are various techniques you can learn that will help with making neater solder joints.

Solder the Circuit

Insert the components in the Perfboard following the Hardware Connection image as a general visual guide. Insert the pins of the components (all from the same side) so that you have only the components on one side of the Perfboard and the pins on the other. Then from the side with the pins carefully solder the circuit on the backside without having excess solder shorting out any portions of the circuit.

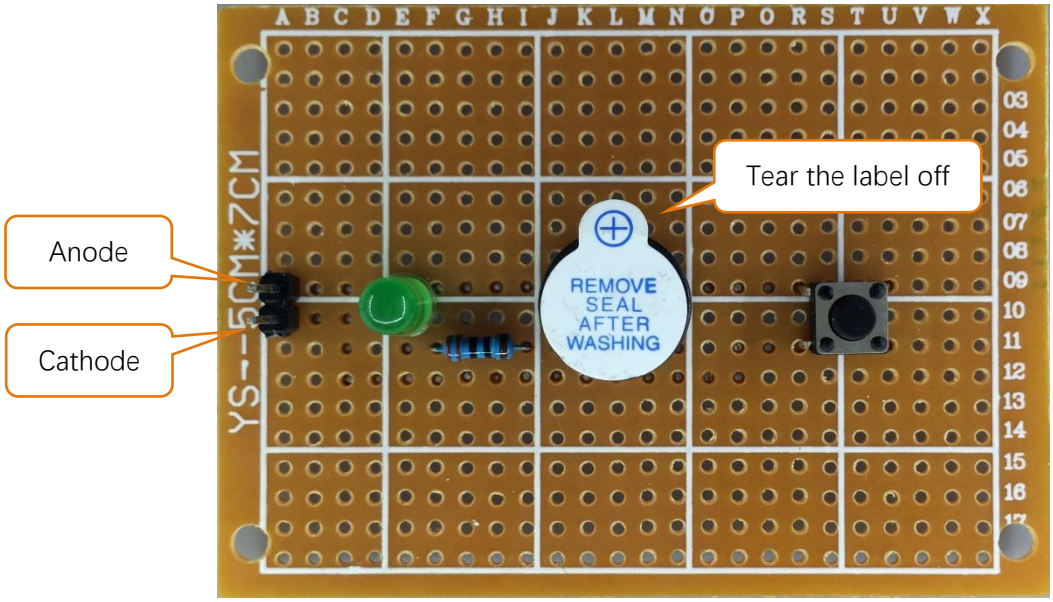


Here is a diagram after soldering from both sides of the Perfboard:

Front	Back
	

Test Circuit

Connect the circuit board to power supply (3~5V). You can use control board or battery box as the power supply.

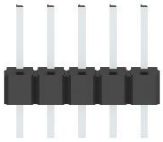
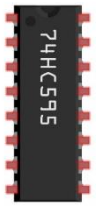


Press the push button switch after connecting the power, and then the buzzer will sound.

Project 25.2 Solder a Flowing Water Light

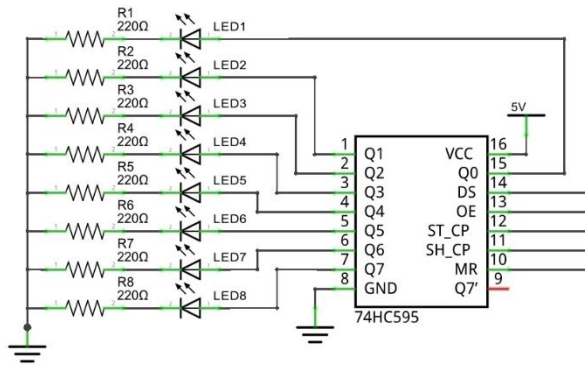
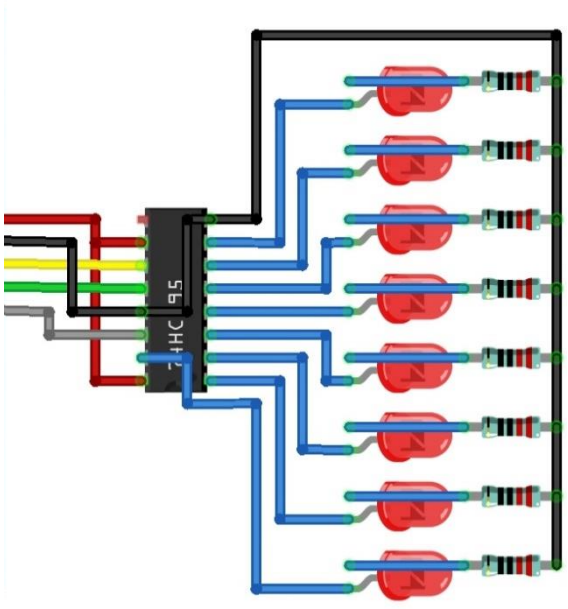
You should be familiar with the Flowing Water Light from our previous project. We will solder a permanent circuit using improved code to make a more interesting Flowing Water Light.

Component List

Pin header x5	Resistor 220Ω x8	LED x8	74HC595 x1
			

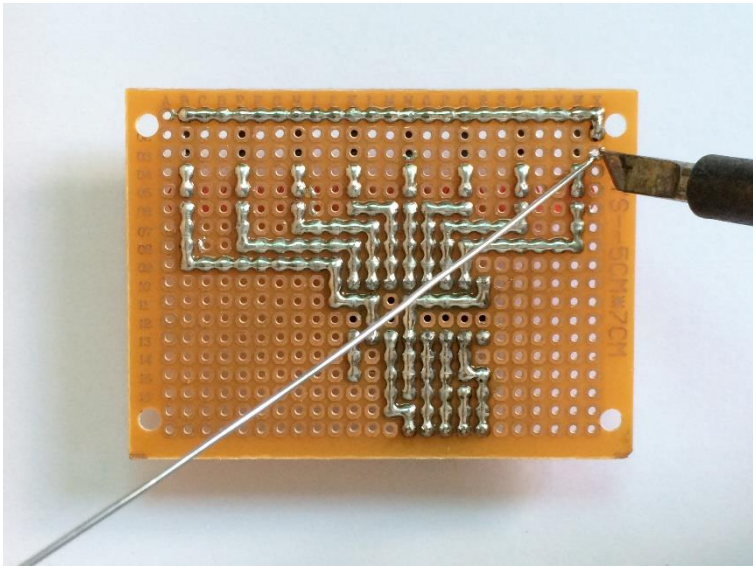
Circuit

Solder the following circuit on the Perfboard.

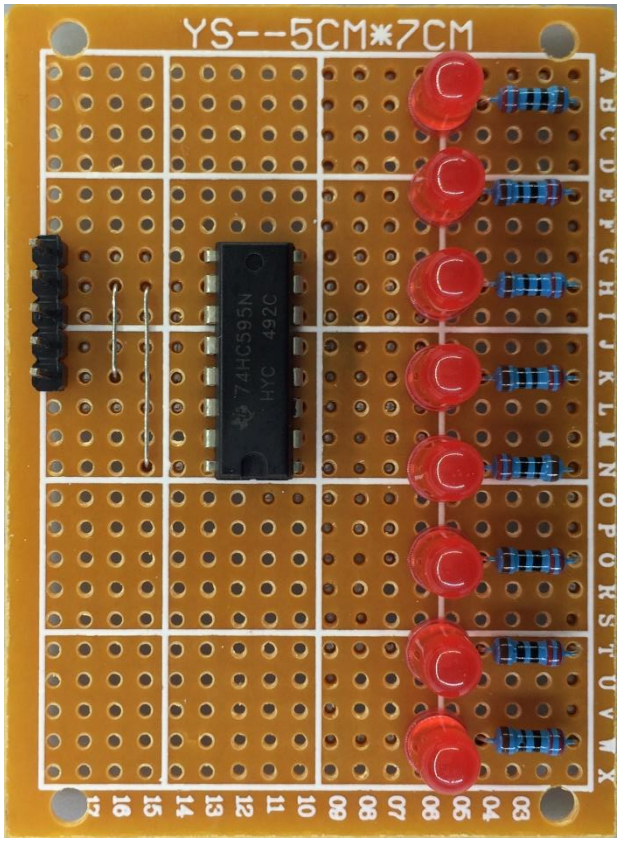
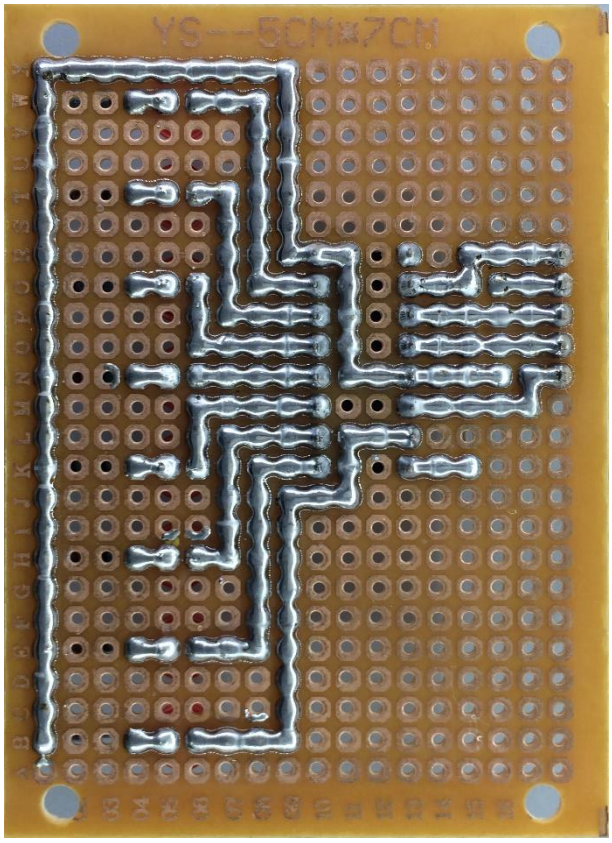
Schematic diagram	Hardware connection
	

Solder the Circuit

Insert the components in the Perfboard, and solder the circuit on the back per earlier instructions.

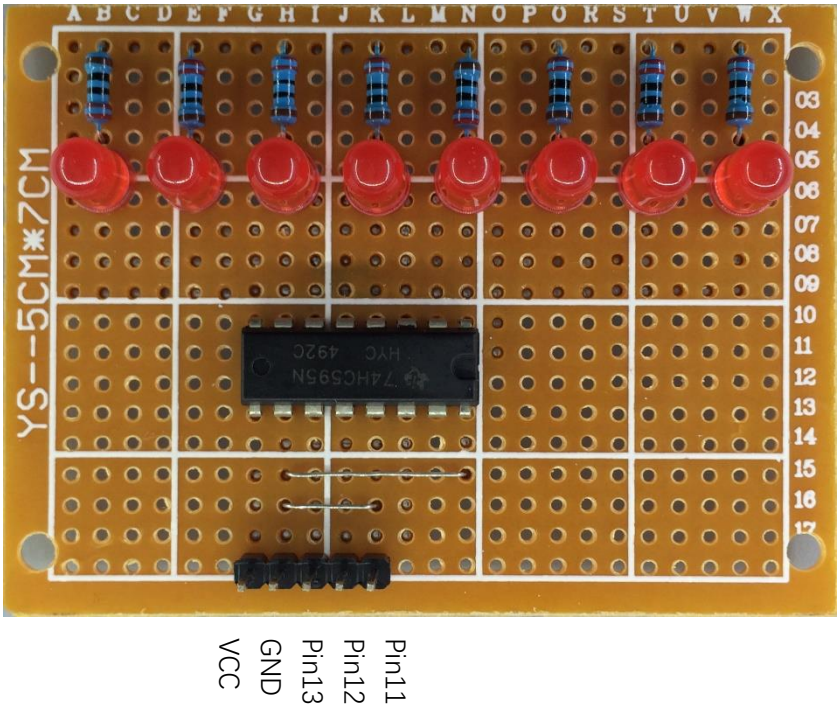


Here is a diagram after soldering from both sides of the Perfboard:

Front	Back
	

Connect the Circuit

Connect the board to control board with jumper wire in the following way.



Sketch

Sketch 25.2.1

Now, let's write code to make a dropping-rain effect on our board.

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

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

void loop() {
  // Define an array to save the pulse width of LED. Output the signal to the 8 adjacent
LEDs in order, and then it produces the dropping-rain effect
```

```
14    const byte pulse[] = {0, 0, 0, 0, 0, 0, 0, 0, 64, 48 ,32, 16, 8, 4, 2, 1, 0, 0, 0, 0,
      0, 0, 0, 0};
15    // Define a variable to select 8 contiguous data in the array sequentially
16    static byte offset;
17    // Define a variable to control the speed
18    static unsigned int counter;
19    if (counter++ % 8 == 0)               // Reduce the self-increasing speed of offset
20      offset < 15 ? offset++ : offset = 0;// offset increases
21    // Out put PWM wave
22    for (int i = 0; i < 64; i++) {  // The cycle of PWM is 64 cycles
23      byte data = 0;                // Define a variable to represent the output state of
      this loop
24      for (int j = 0; j < 8; j++)   // Calculate the output state of this loop
25      {
26        if (i < pulse[j + offset])  // Calculate the LED state according to the pulse width
27        {
28          data |= 0x01 << j;        // Represent the LED state with the corresponding bit
      of a variable
29        }
30      }
31      // Send the state of LED to 74HC595
32      writeData(data);
33    }
34  }
35
36  void writeData(int value) {
37    // Make latchPin output low level
38    digitalWrite(latchPin, LOW);
39    // Send serial data to 74HC595
40    shiftOut(dataPin, clockPin, MSBFIRST, value);
41    // Make latchPin output high level, then 74HC595 will update the data to parallel
      output
42    digitalWrite(latchPin, HIGH);
43  }
```

First we define an array to modulate different PWM pulse widths for LEDs, in doing so different LEDs can emit varied brightness. Starting from the array index 0, take an array of 8 adjacent numbers as the LED duty cycle and output it one at a time. Increasing the starting index number in turn, then it will create a flowing effect.

```
14    const byte pulse[] = {0, 0, 0, 0, 0, 0, 0, 0, 64, 48 ,32, 16, 8, 4, 2, 1, 0, 0, 0, 0,
      0, 0, 0, 0};
```

Define a variable to select 8 adjacent numbers in an array in turn.

```
16    static byte offset;
```

Define a variable to control the speed of the raindrops.

```
18    static unsigned int counter;
```

Reduce the auto-increment speed of the variable offset with the following code.

```
19    if (counter++ % 8 == 0)                // Reduce the self-increasing speed of offset
20      offset < 15 ? offset++ : offset = 0;// offset increases
```

We use software to output PWM waveform. Define the cycle of PWM to be 64 cycles and determine the pulse width of LED (that is brightness) according to the selected eight numbers of the array in each cycle.

```
22    for (int i = 0; i < 64; i++) {  // The cycle of PWM is 64 cycles
23      byte data = 0;                // Define a variable to represent the output state of
      this loop.
24      for (int j = 0; j < 8; j++)   // Calculate the output state of this loop
25      {
26        if (i < pulse[j + offset])  // Calculate the LED state according to the pulse width
27        {
28          data |= 0x01 << j;        // Represent the LED state with the corresponding bit
      of variable
29        }
30      }
```

Due to the change of the variable offset, the LED will output the brightness that the eight adjacent numbers represents in the array, and form the dropping-rain effect.

Verify and upload the code, and then you will see the dropping-rain effect that LED forms.

Other Components

This kit also includes other common components that can help your ideas come true. Now we will introduce components not mentioned in the previous section.

Component Knowledge

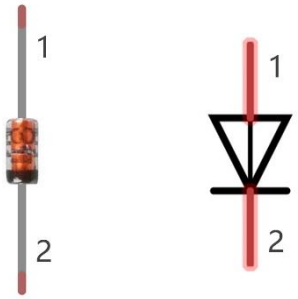
Toggle switch

Like push button switch, toggle switch is also a kind of switching devices. The difference is that toggle switch is suitable for long-time open or close circuits.

When the lever is moved to the left, pin 1, 2 get conducted, and pin 2, 3 are disconnected from each other;
When the lever is moved to the right, pin 2,3 get conducted, and pin 1,2 are disconnected from each other;

Switch diode

There are several types of diodes. We have used 1N4001 before, which is a common rectifier diode and commonly used in ac rectifier.
Here is 1N4148, which is a kind of high-speed switching diodes and is characterized by a relatively rapid switching.



For the switching diode, the changing time from conduction to cut off or from cut off to conduction is shorter than general diode and it is mainly used in electronic computer, pulse and switching circuits.

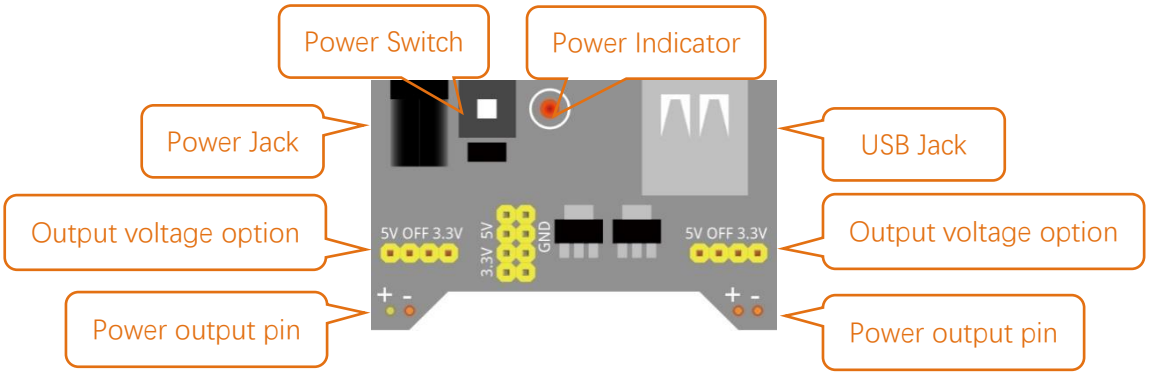
9V battery cable

A 9V battery cable can connect a 9 V battery, which can supply power for control board.

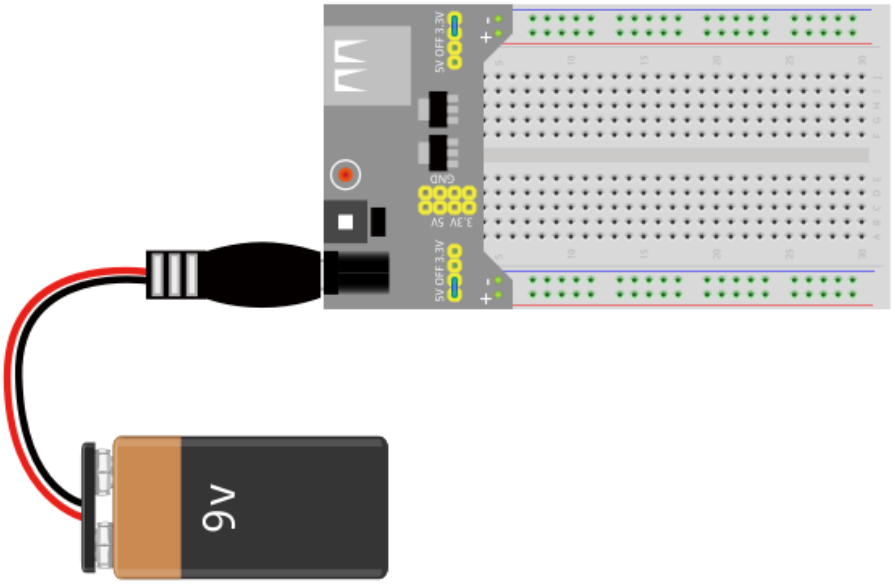
The installation of 9V battery cable is as follows:

Power supply module for breadboard

The following is the power supply module for breadboard. This module can provide the breadboard with two-channel power supply separately, and each can be configured to 3.3 V or 5 V separately through a jumper.



We can build a circuit conveniently by using this module. You only need to provide power supply for this module, and then insert it on the breadboard.



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.

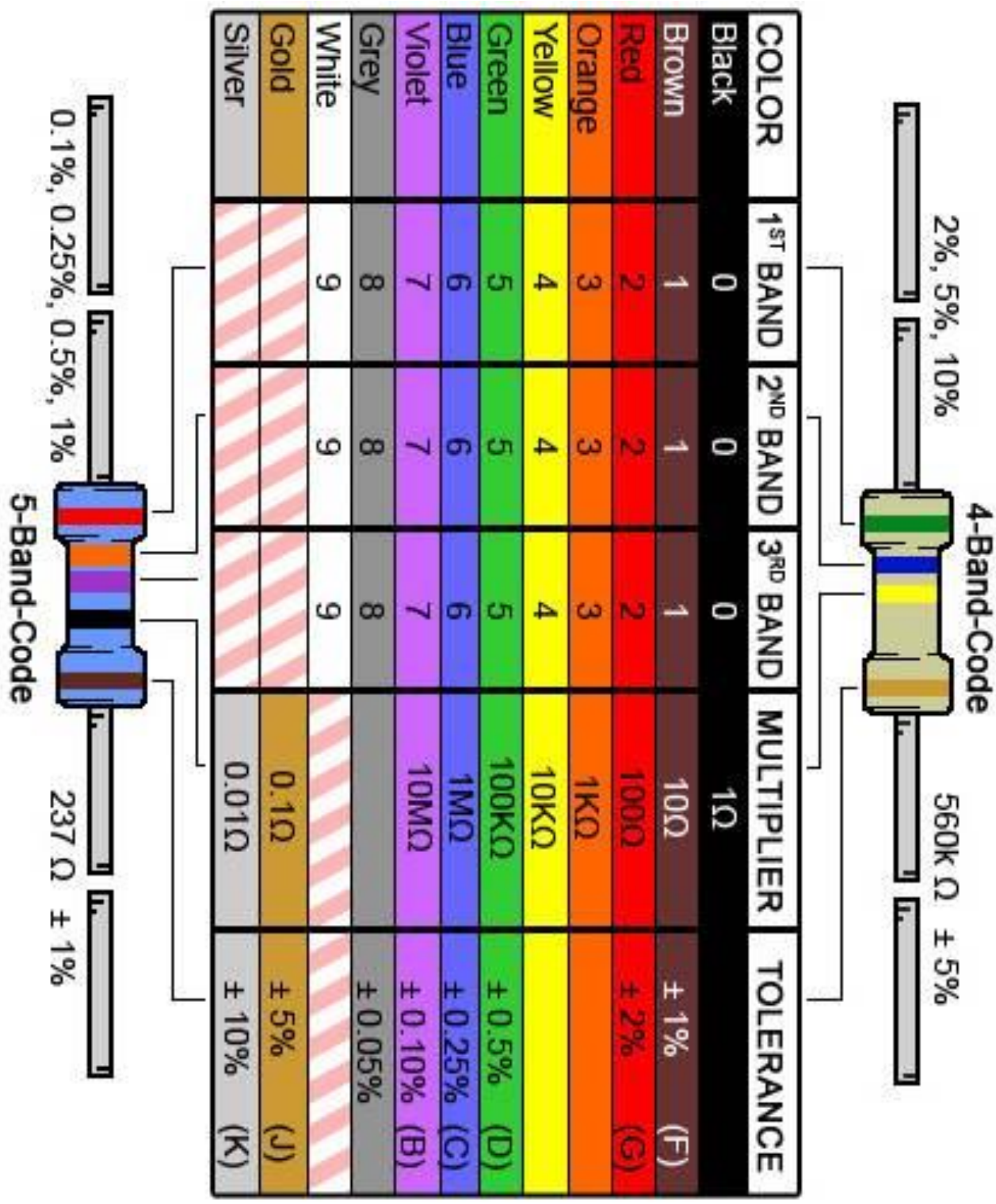
Thank you again for choosing Freenove products.

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)