Welcome

Thank you for choosing Freenove products!

Getting Started

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.

Get Support

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

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

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi® and micro: bit®
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website: http://www.freenove.com

Copyright

Contents

Welcome i
Contents I
Preface 1
Control Board 1
Projects Board 3
Programming Software 4
First Use 7
Chapter 1 LED Blink 11
Project 1.1 LED Blink 11
Chapter 2 Flowing LED 19
Project 2.1 Flowing LED Display 19
Chapter 3 Control LED with Push Button Switch 25
Project 3.1 Control LED with Button 25
Project 3.2 Change LED State by Button 29
Chapter 4 Serial 31
Project 4.1 Send Data through Serial 31
Project 4.2 Receive Data through Serial Port 36
Project 4.3 Application of Serial 40
Chapter 5 ADC 45
Project 5.1 ADC 45
Project 5.2 Control LED by Potentiometer 50
Project 5.3 Control LED by Photoresistor 53
Chapter 6 RGB LED 57
Project 6.1 Control RGB LED through Potentiometer 57
Project 6.2 Multicolored LED 61
Chapter 7 LEDPixel 64
Project 7.1 LEDPixel 64
Project 7.2 Rainbow Light 71
Chapter 8 Buzzer 74
Project 8.1 Active Buzzer 74
Project 8.2 Passive Buzzer 80
Chapter 9 Relay 83
Project 9.1 Control Relay 83
Chapter 10 Motor 90
Project 10.1 Control Motor with L293D 90
Chapter 11 Servo 96
Project 11.1 Servo Sweep 96
Project 11.2 Control Servo with Potentiometer 101
Chapter 12 Temperature Sensor 104
Project 12.1 Detect the Temperature 104
Chapter 13 Joystick 108

Project 13.1 Joystick 108
Chapter 14 Acceleration sensor 113
Project 14.1 Acceleration Detection 113
Chapter 15 LED Bar 120
Project 15.1 LED Bar 120
Chapter 16 4-digit 7-segment Display 126
Project 16.1 4-digit 7-segment Display 126
Project 16.2 4-digit 7-segment Display 131
Chapter 17 LED Matrix 135
Project 17.1 LED Matrix 135
Chapter 18 I2C LCD1602 143
Project 18.1 Display the String on I2C LCD1602 143
Project 18.2 I2C LCD1602 Clock 148
Chapter 19 Stepper Motor 157
Project 19.1 Drive Stepper Motor 157
Chapter 20 Matrix Keypad 163
Project 20.1 Get Input Characters 163
Chapter 21 Infrared Remote 168
Project 21.1 Infrared Remote Control 168
Project 21.2 Control LED through Infrared Remote 173
Chapter 22 Infrared Motion Sensor 177
Project 22.1 Infrared Motion Sensor 177
Chapter 23 Ultrasonic Ranging 182
Project 23 Ultrasonic Ranging 182
Chapter 24 RFID 189
Project 24.1 Read UID 189
Project 24.2 Read and write 197
What's Next? 201
Appendix 202
ASCII Table 202

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.

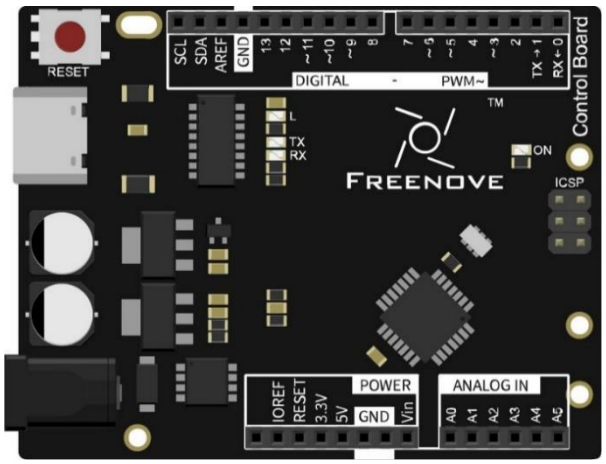
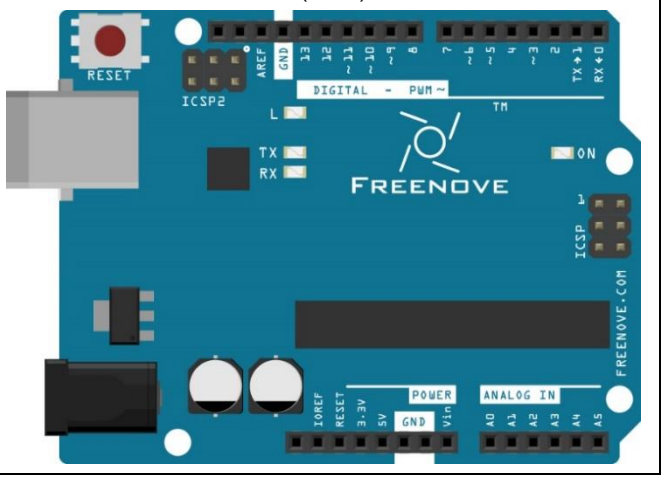
Support email: support@freenove.com

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.

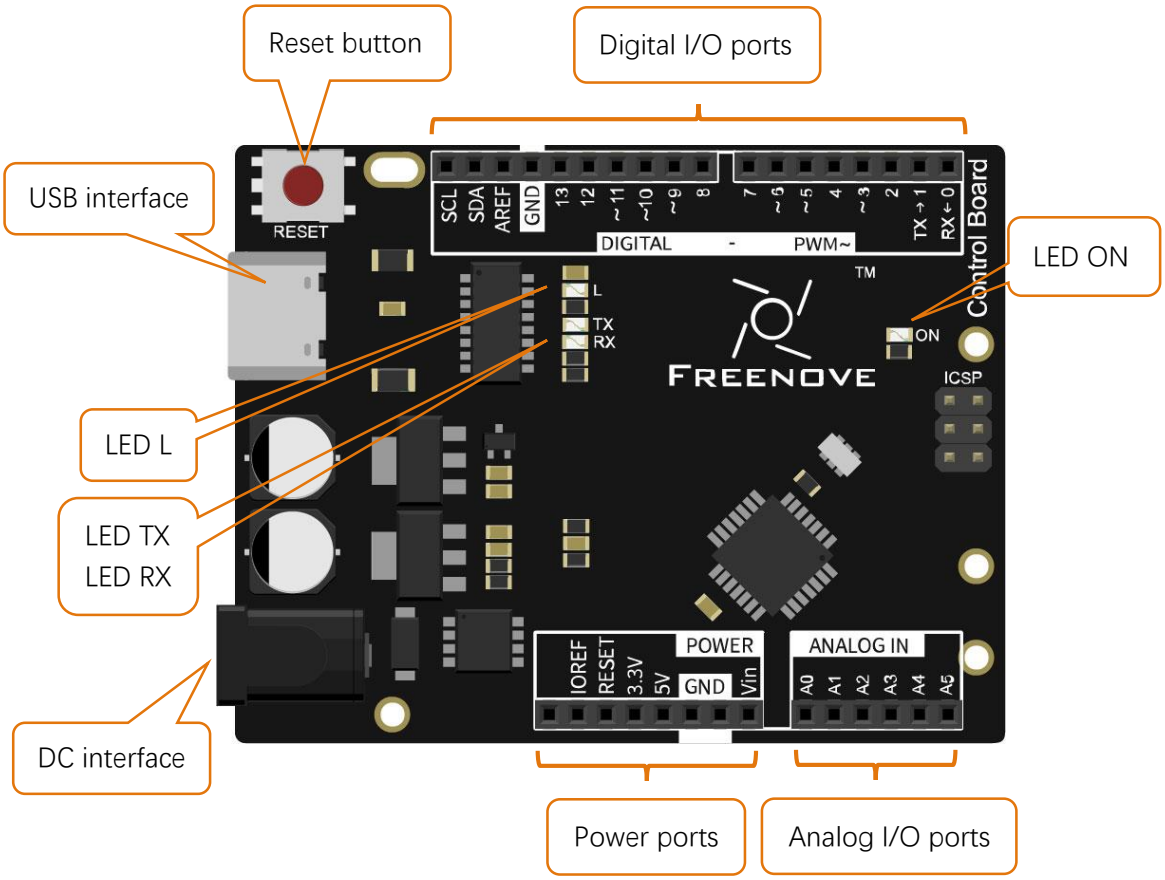
There are multiple versions of Freenove control board. Your purchase may be one of the following:

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.

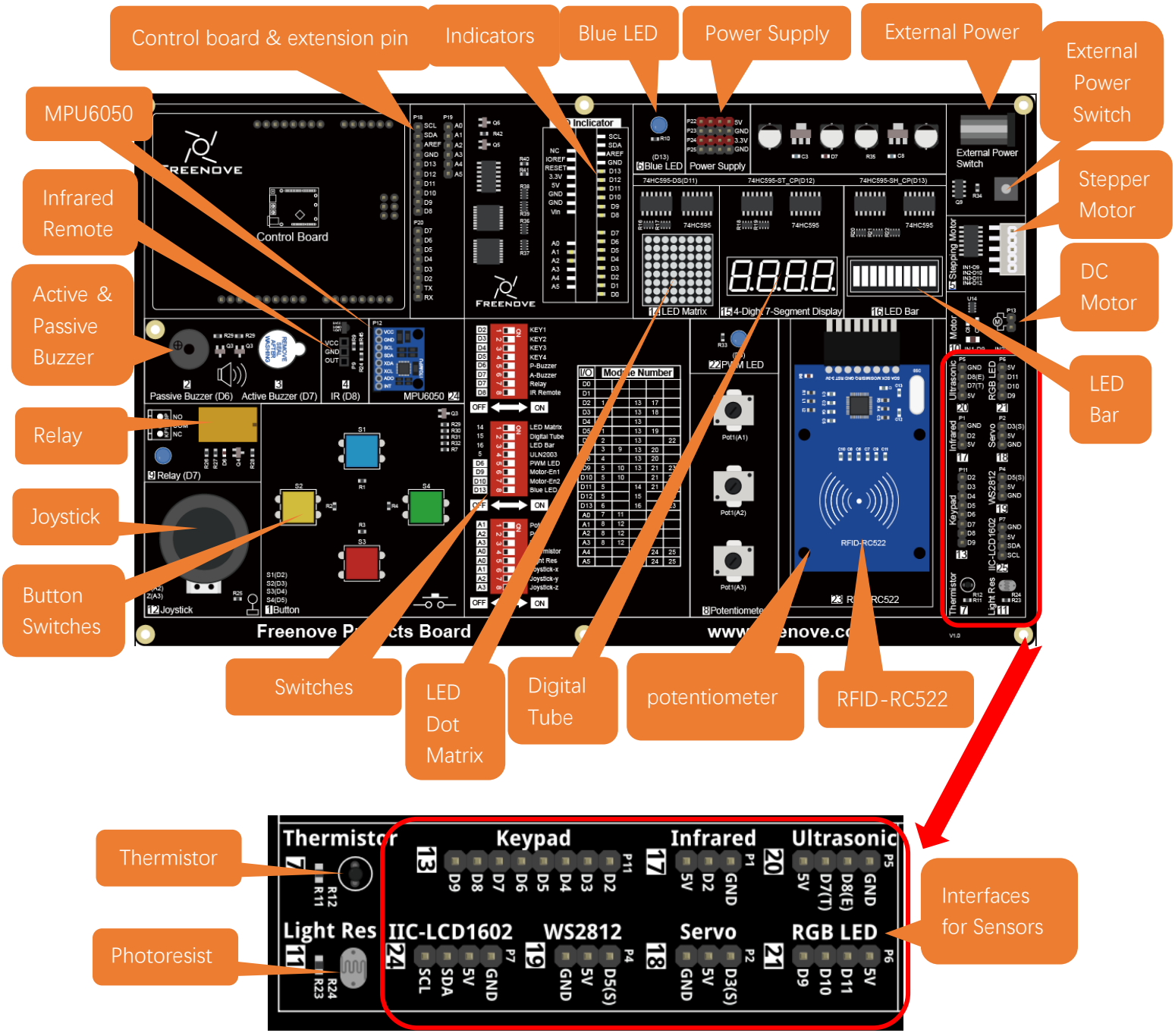
Diagram of the Freenove control board is shown below:



- 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.

Projects Board

Features

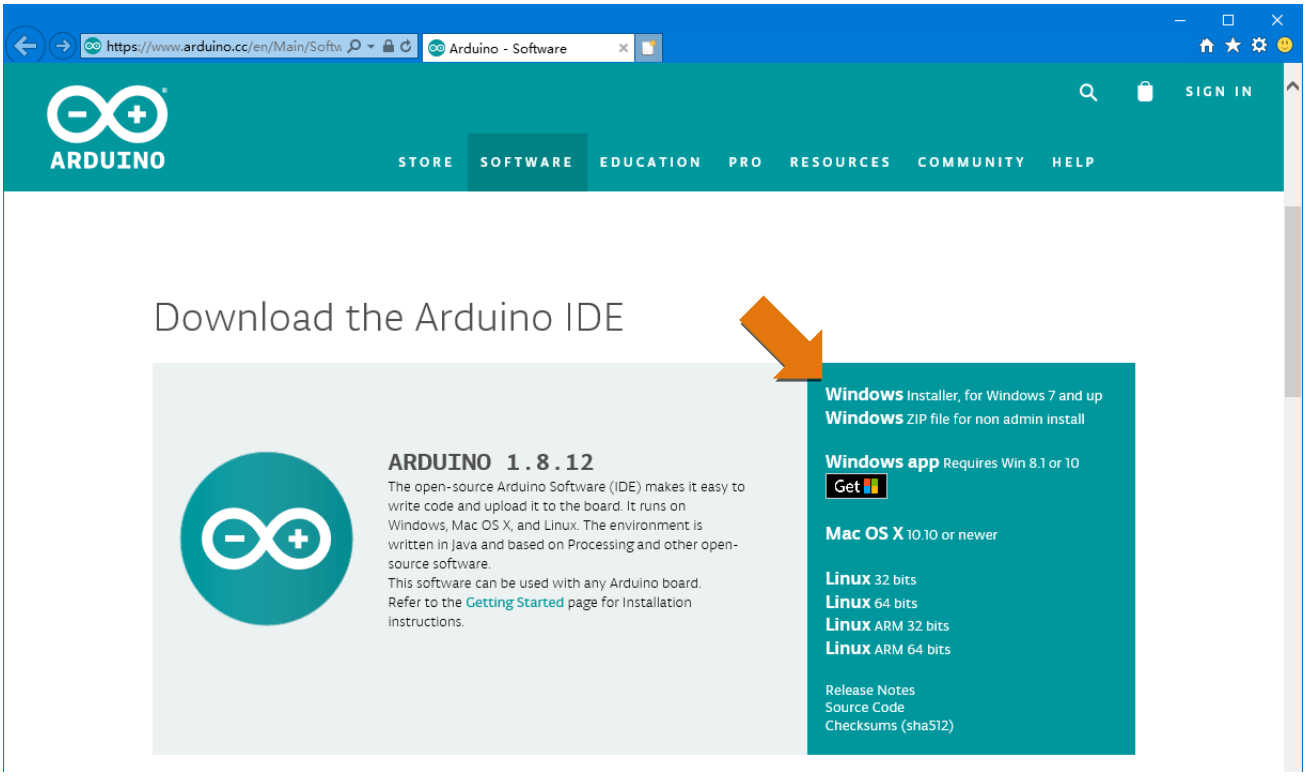


Programming Software

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

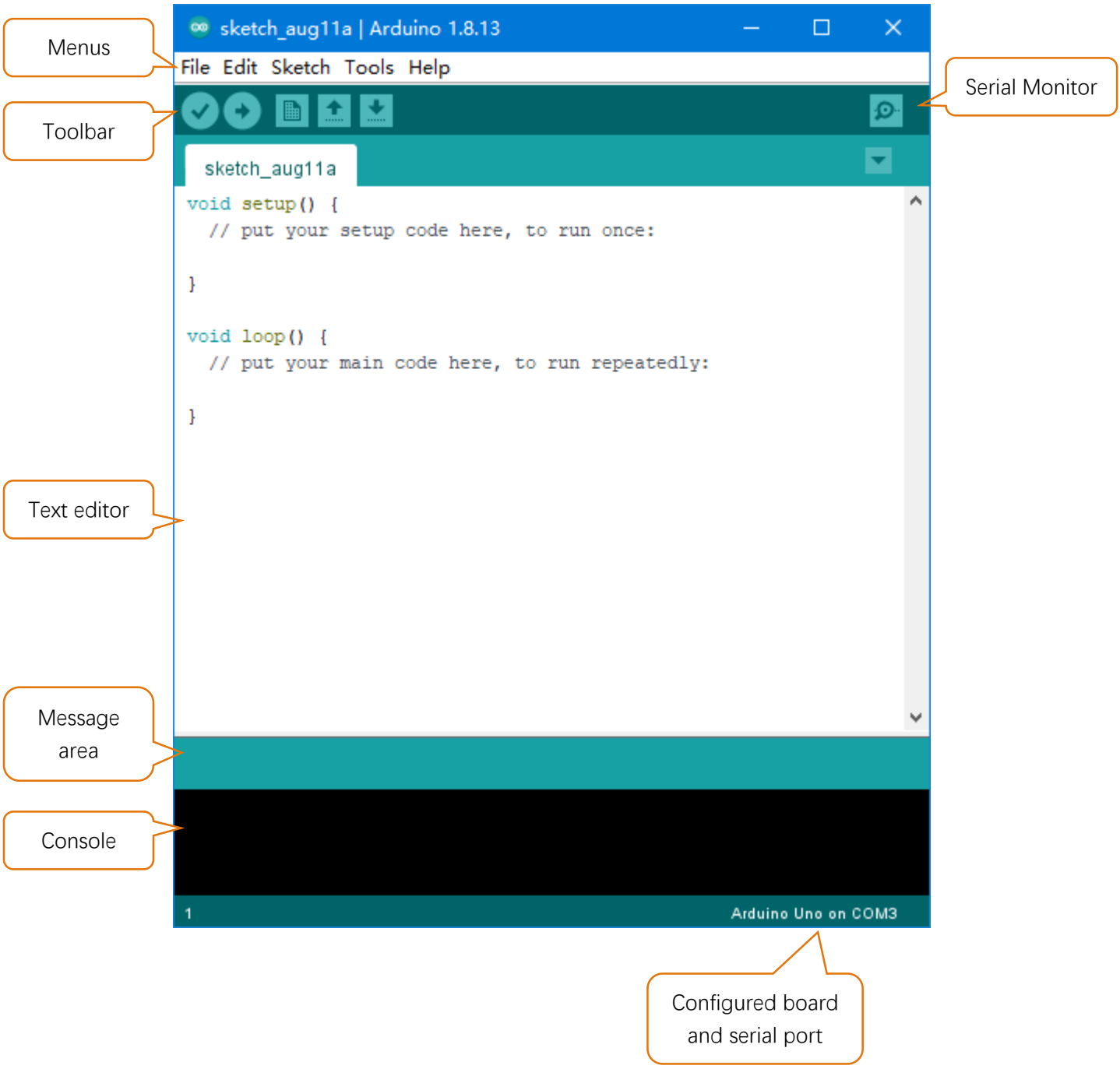
Arduino IDE uses C/C++ programming language. Don't worry even if you have never used it, because this document contains programming knowledge and detailed explanation of the code.

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



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

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



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

Verify
Checks your code for errors compiling it.

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

New
Creates a new sketch.

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

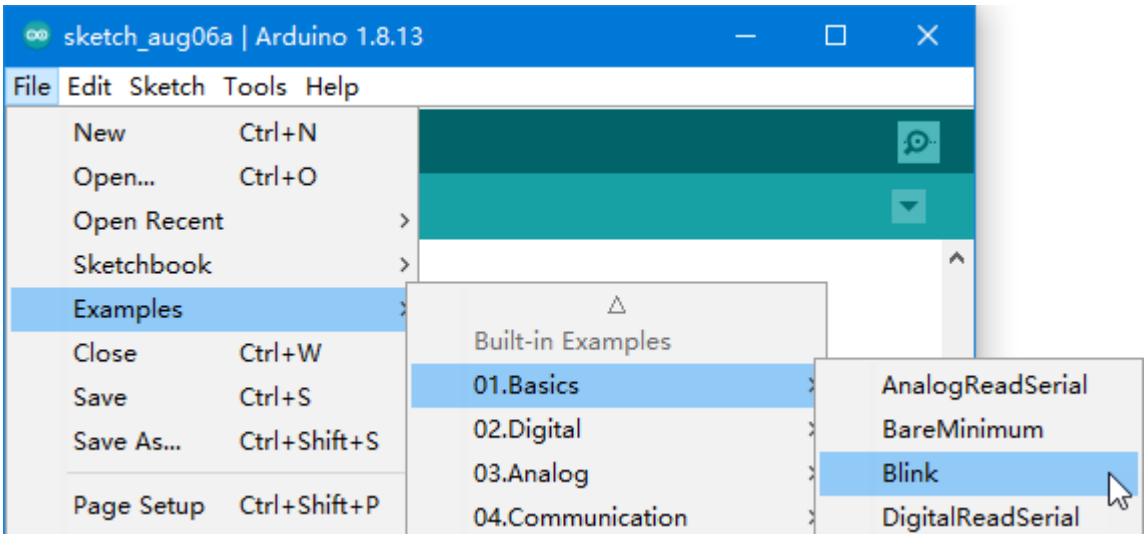
Save
Saves your sketch.

Serial Monitor
Opens the serial monitor.

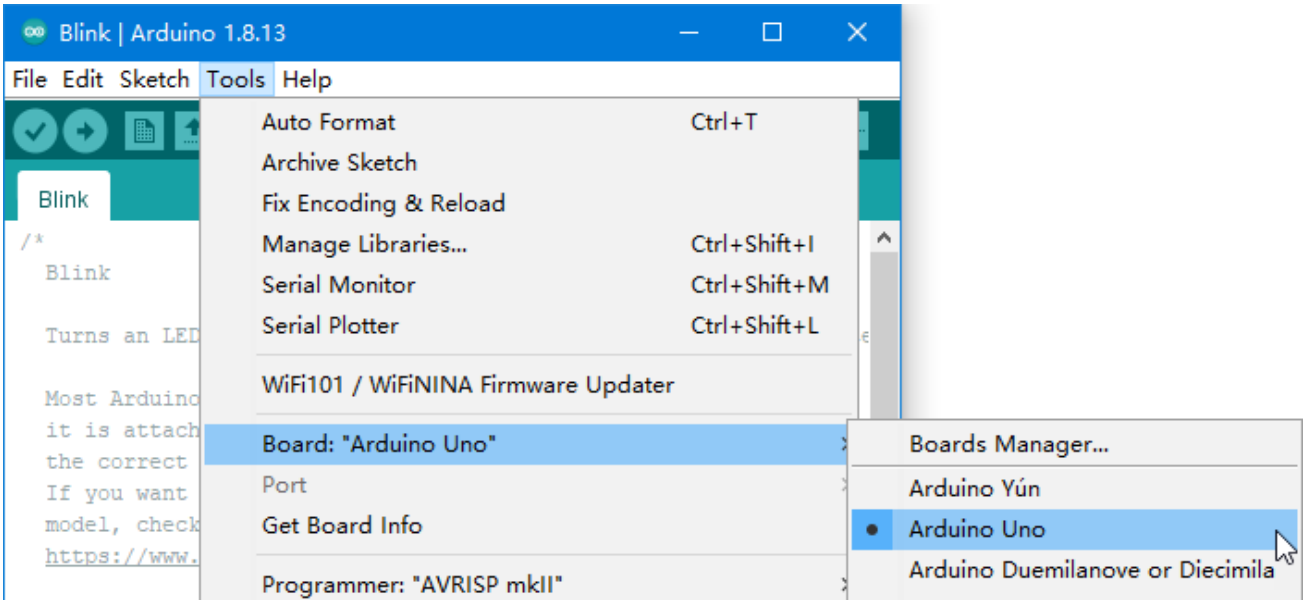
Additional commands are found within five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

First Use

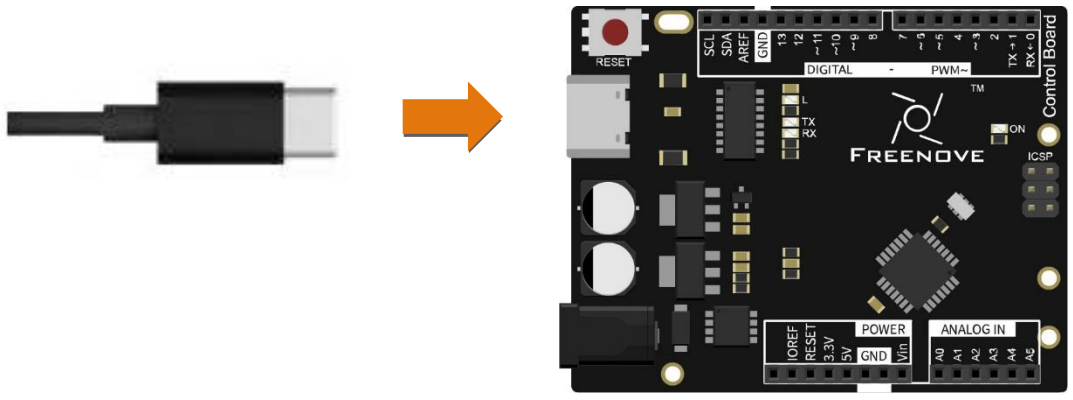
Open the example sketch "Blink".



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



Connect control board to your computer with USB cable.



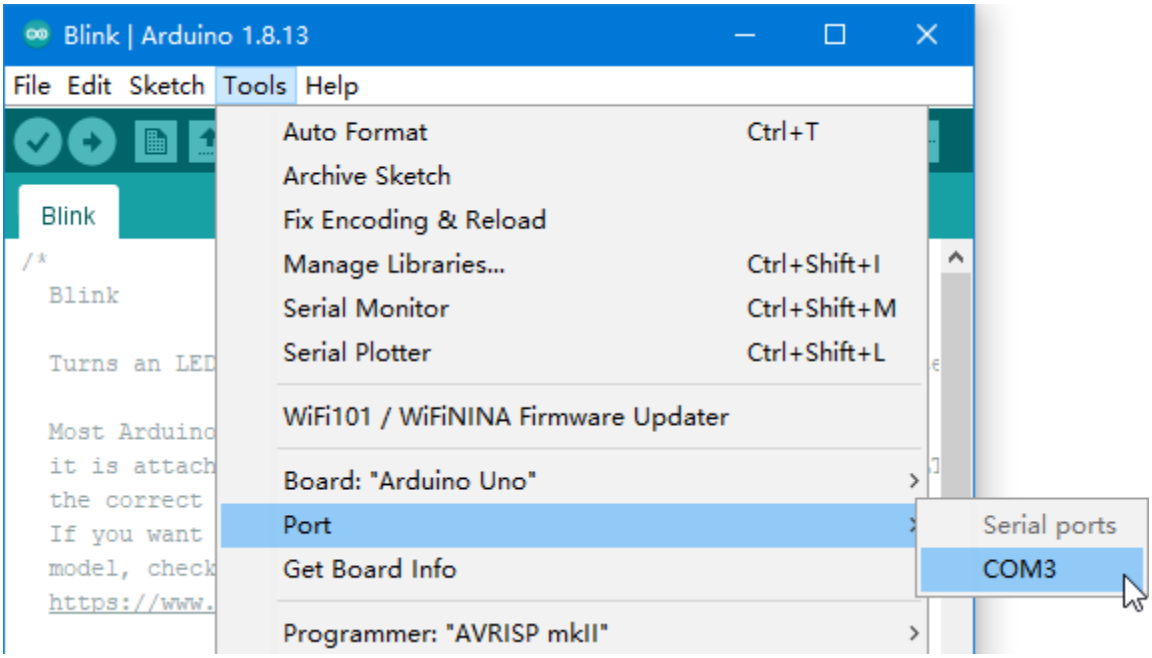
Select the port.

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

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

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

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



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

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

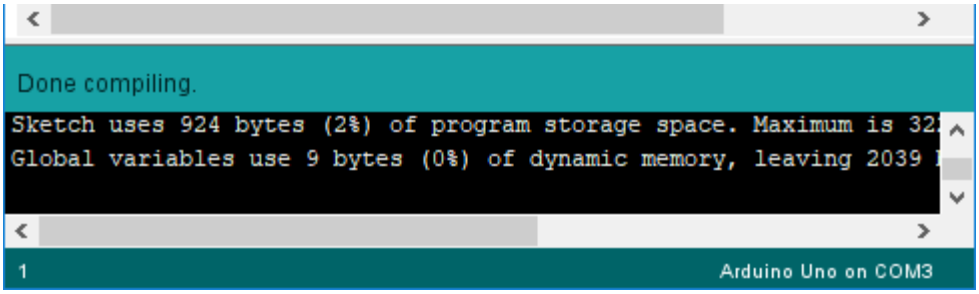
- For blue board, reinstall the latest version of Arduino IDE. During installation, agree to install the driver.
- For black board, see “InstallDriver.pdf” in “Drivers” folder (in the folder contains this Tutorial.pdf).

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

Click "Verify" button.

The following figure shows the code being compiled.

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



Click "Upload" button.

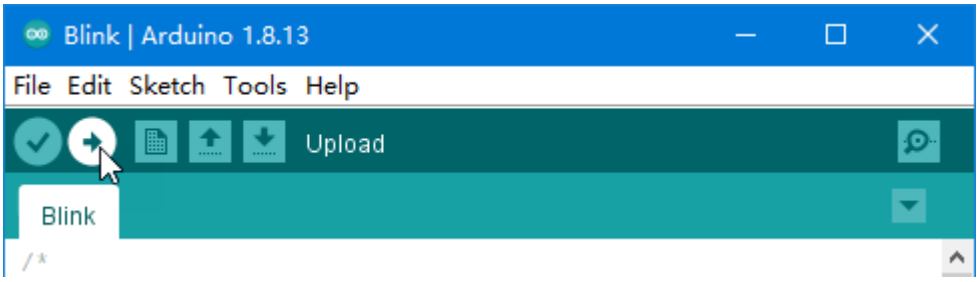
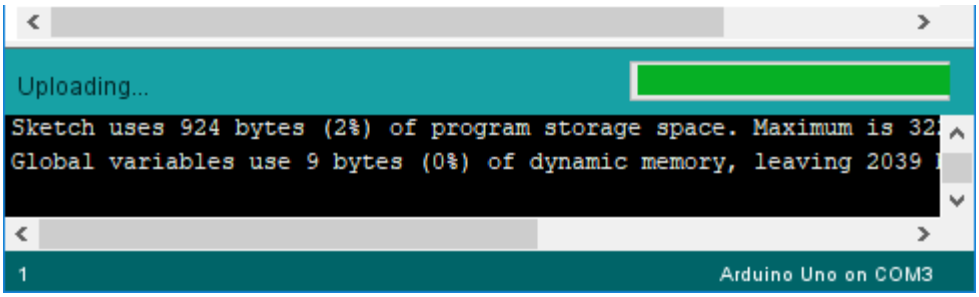
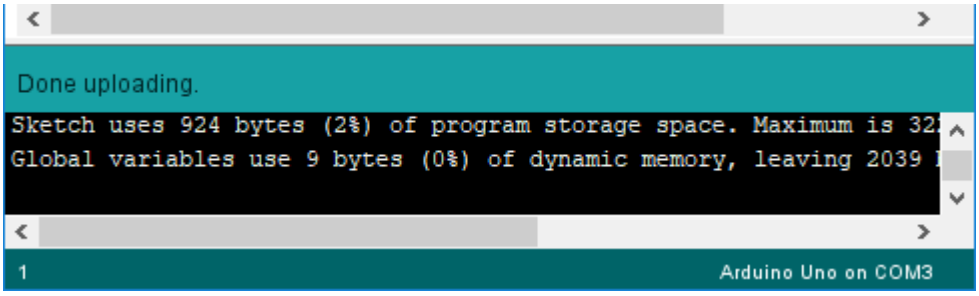


Figure below shows code uploading.

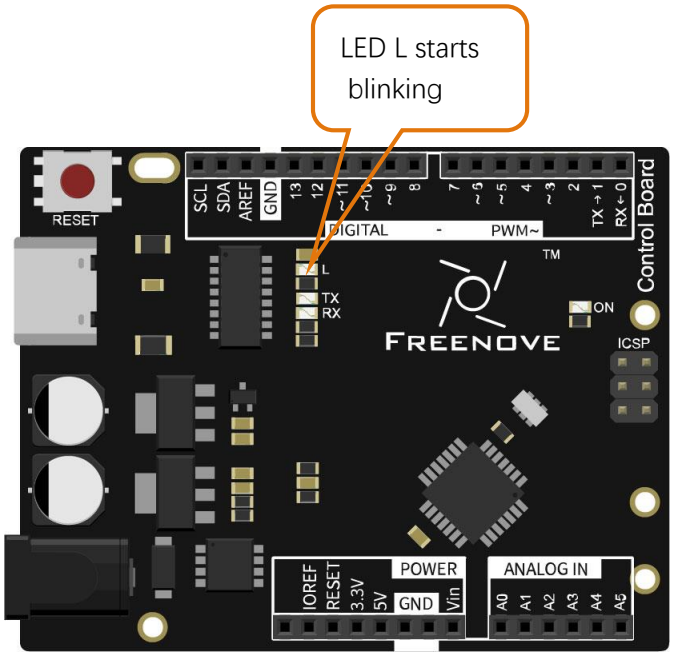


Wait a moment, and then the uploading is completed.



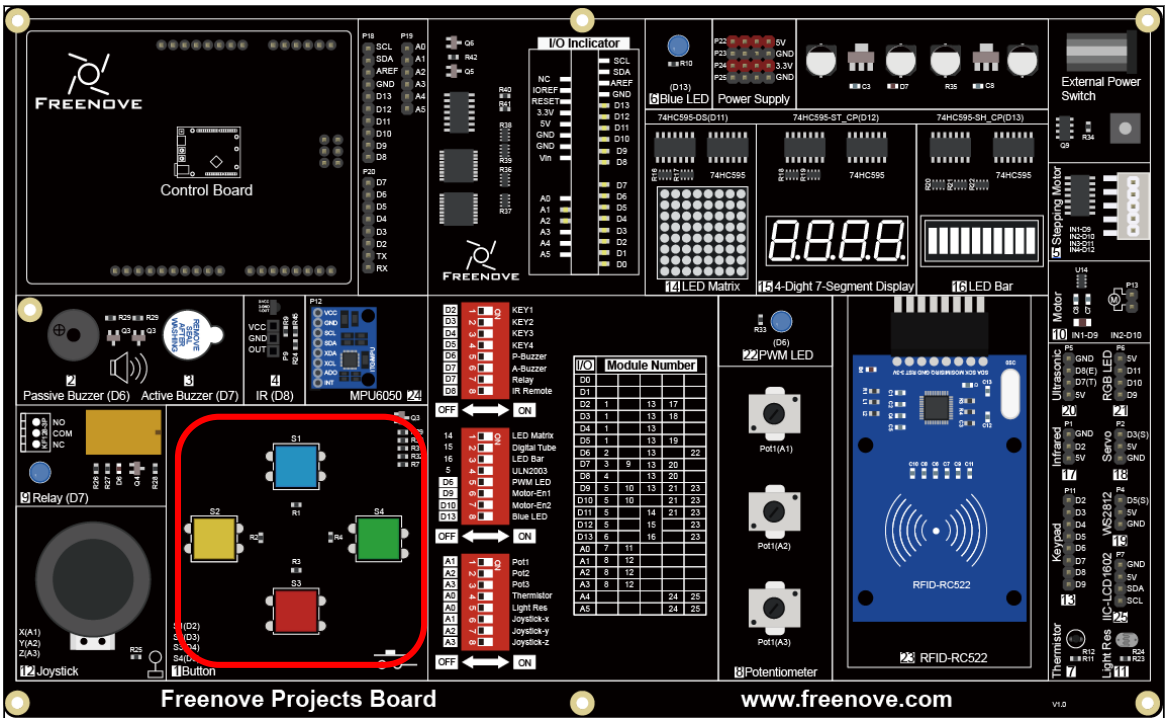
Having problems? Contact us for help! Send mail to: support@freenove.com

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, to help you learn programming.

Before use, you can install the button cap in the material package on the board.



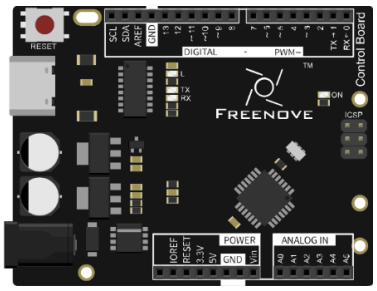
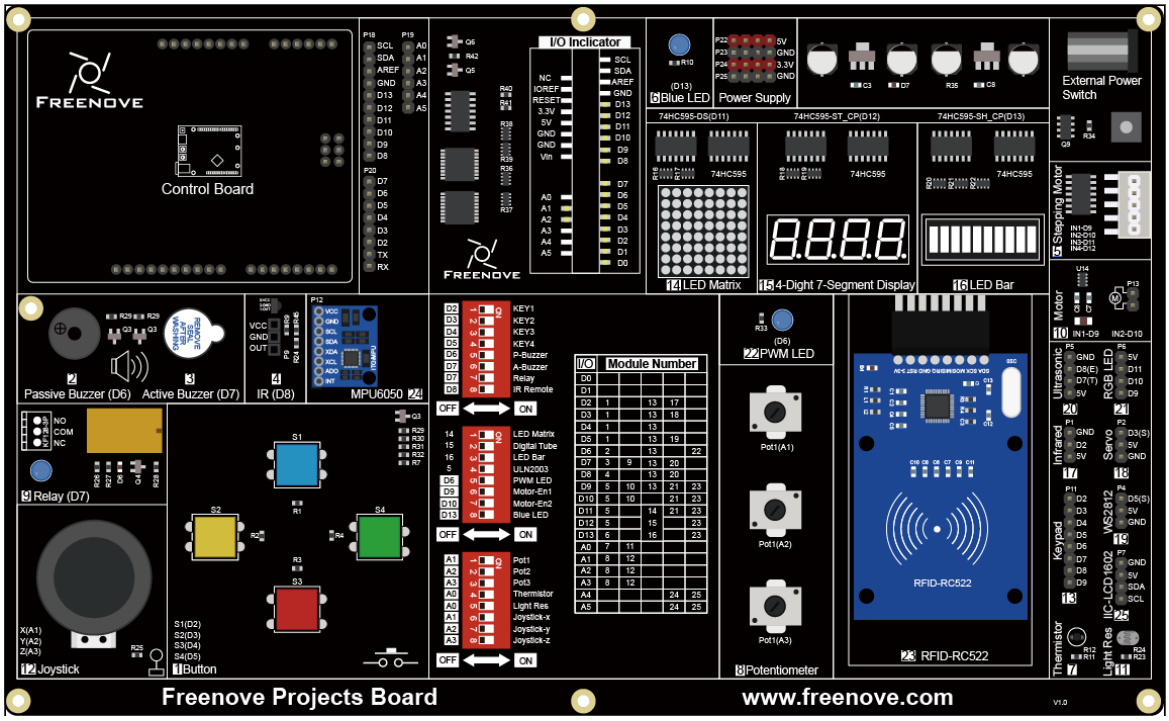
Chapter 1 LED Blink

Earlier we tried to make the LED marked "L" blink. Now let us use Freenove Projects Board to reproduce this phenomenon and try to understand its principle.

Project 1.1 LED Blink

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 	USB cable x1 
Freenove Projects Board 	

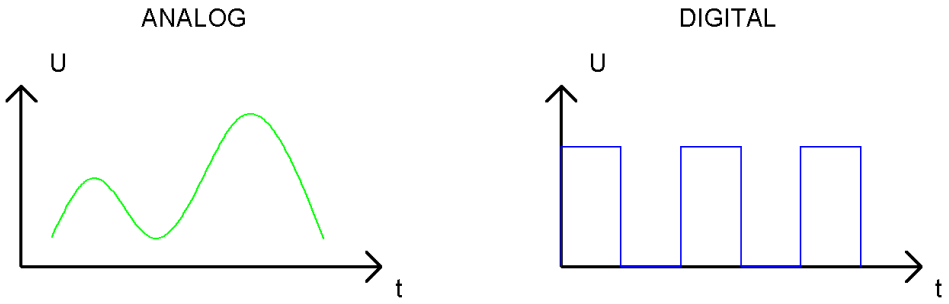
Component Knowledge

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

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.

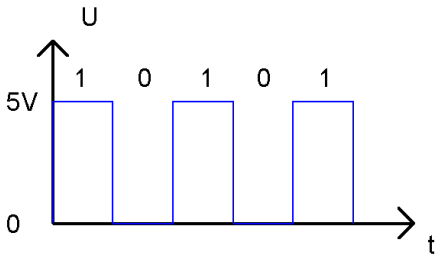


In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

Low level and high level

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

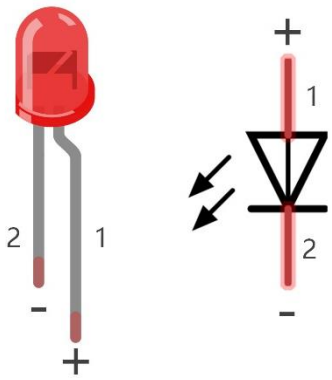
Low level is generally equal to ground voltage(0V), while high level is to the operating voltage of components. The low level of the control board is 0V and high level is 5V, as shown below. When IO port on control board outputs high level, components of small power can be directly lit, like LED.



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) when its longer pin (+) 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 common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



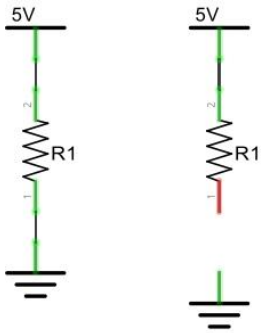
Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

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.

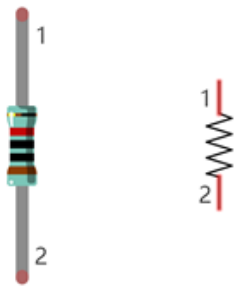
In the figures below: the left one is a loop circuit, so current flows through the circuit. The right one is not a loop circuit, so there is no current.



Resistor

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

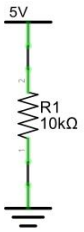
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and on the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

Code Knowledge

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

Comments

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

1. Symbol "//"

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

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

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

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

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

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

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

Or

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

Data type

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

Constant

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

Variable

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

```
1  int i;
```

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

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

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

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

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

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

Function

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

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

"void" indicates that the function does not return a value (Chapter 4 will detail the return value of functions);

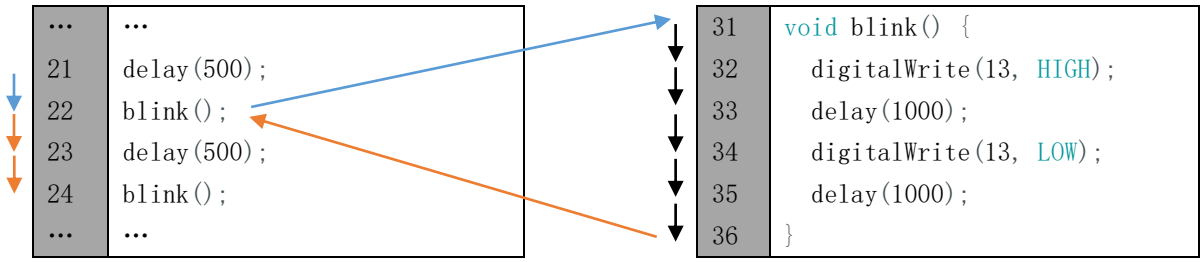
"()" its inside is parameters of a function (Chapter 2 will detail the parameters of the functions). No content inside it indicates that this function has no parameters;

"{}" contains the entire code of the function.

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

```
1  blink();
```

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



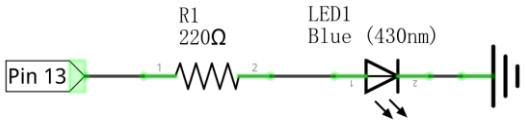
Some functions have one or more parameters. When you call such functions, you need to write parameters inside "()":

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

Circuit

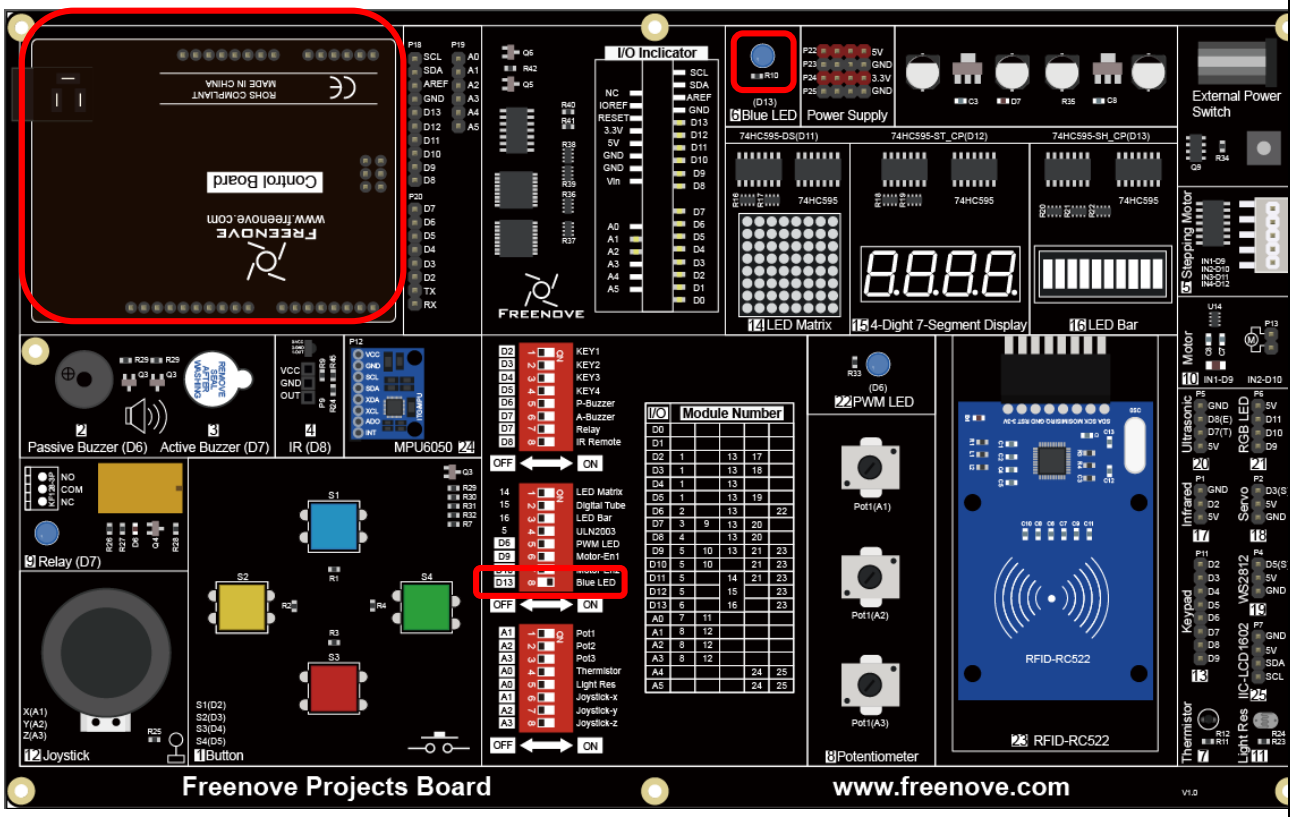
Now, we will use IO port of control board to provide power for the LED. Pin 13 of the control board is the digital pin. It can output high level or low level. In this way, control board can control the state of LED.

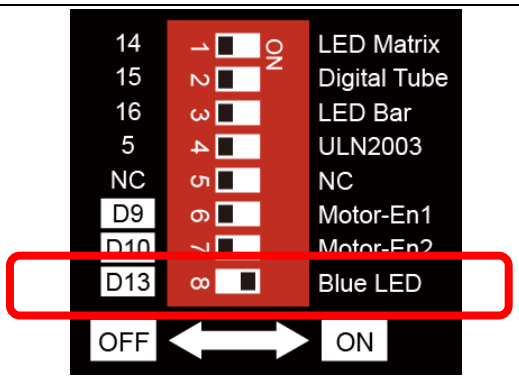
Schematic diagram



Hardware connection

Insert the Control Board upside down to the Freenove Projects Board, and then turn the corresponding switch to the right (On)





For other functions that are not being used, it is recommended to turn their DIP switches to the left.

Sketch

Blink

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 learn more.

```
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 }
```

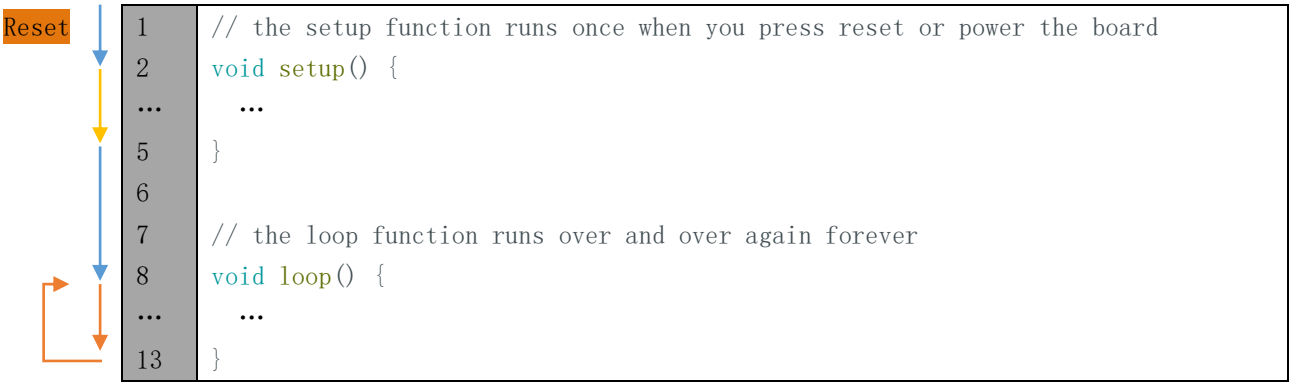
The code usually contains two basic functions: void setup() and void loop().

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
```

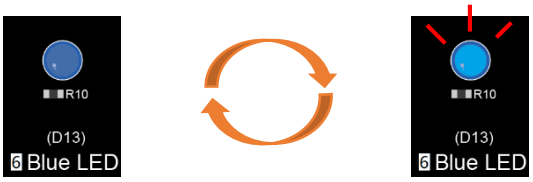
Then set the pin 13 to output low level, and LED lights off. One second later, the execution of loop () function will be completed.

```
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 Flowing LED

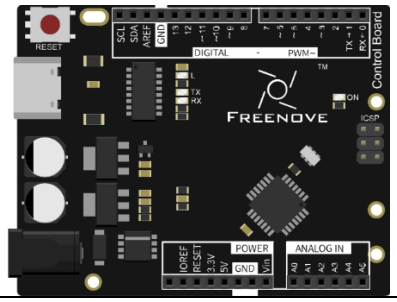
We have learned previously how to control 1 LED through Sketch on the control board and learned some basic knowledge of programming. Now let us try to control 14 LEDs and learn how to simplify the code.

Project 2.1 Flowing LED Display

Let us use control board to control 14 LEDs.

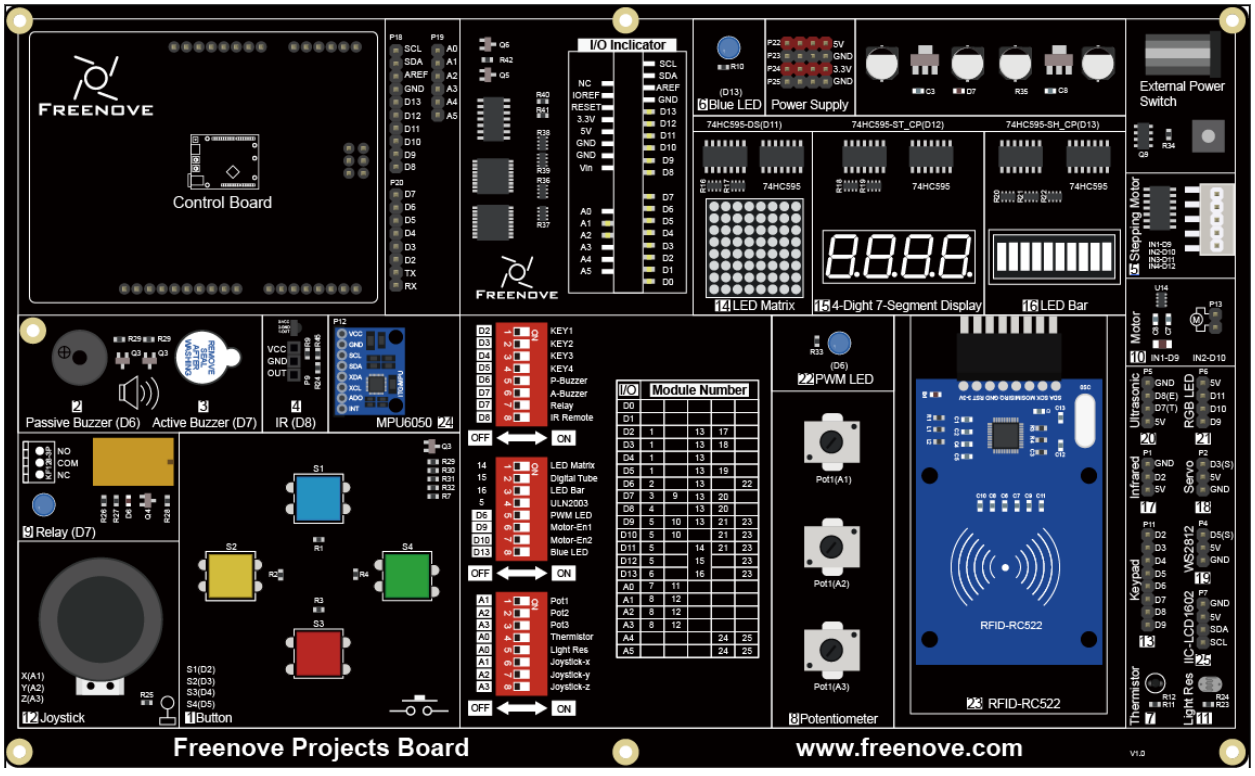
Component List

Control board x1



USB cable x1

Freenove Projects Board



Code Knowledge

This section will introduce new code knowledge.

Array

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

```
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.

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

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

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

We can reference elements of an array as below:

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

Among them, "[]" is the array index, with a[0] as the first elements in the array.
For example, now we define an array b[] below:

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

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

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

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

Loop

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

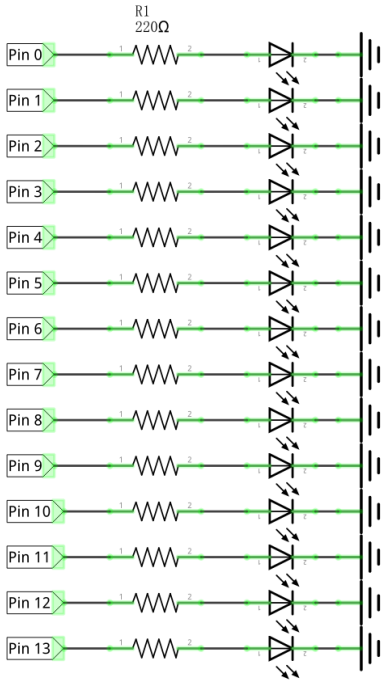
Operator ++, --

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

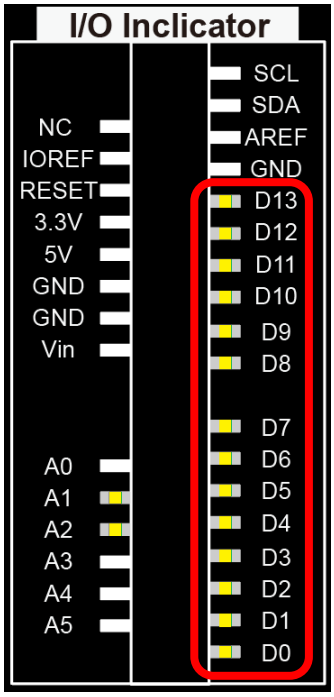
Circuit

Let us use pins 0-13 of the control board to drive LEDs.

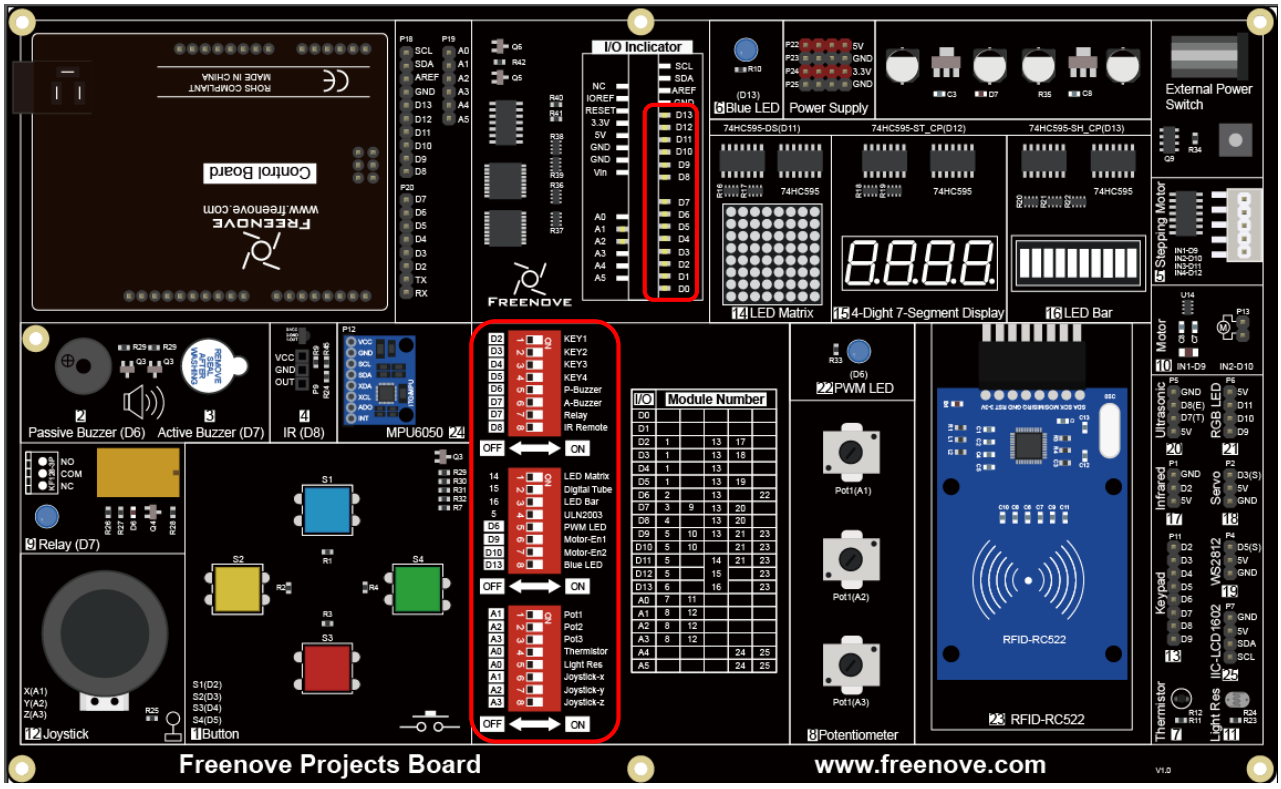
Schematic diagram



Hardware connection



Hardware connection



Turn all the DIP switches to the left.

Sketch

Now let us complete the sketch to control LEDs.

Flowing_LED_Display

First, write a sketch to achieve the "movements" of flowing water.

```
const int ledCount = 14;    // the number of LEDs

// an array of pin numbers to which LEDs are attached
int ledPins[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };

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 will light up in turn
  for (int i = 0; i < ledCount; i++) {
    barGraphDisplay(i);
  }
  for (int i = ledCount-1; i >=0; i--) {
    barGraphDisplay(i);
  }
}

void barGraphDisplay(int ledOn) {
  // make the "ledOn"th 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 = 14;    // the number of LEDs
```

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 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[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
```

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.

```
23 void barGraphDisplay(int ledOn) {
24   // make the "ledOn"th LED on and the others off
25   for (int i = 0; i < ledCount; i++) {
26     if (i == ledOn)
27       digitalWrite(ledPins[i], HIGH);
28     else
29       digitalWrite(ledPins[i], LOW);
30   }
31   delay(100);
32 }
```

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

```
13 void loop() {
14   // the ith LED will light up in turn
15   for (int i = 0; i < ledCount; i++) {
16     barGraphDisplay(i);
17   }
18   for (int i = ledCount-1; i >=0; i--) {
19     barGraphDisplay(i);
20   }
21 }
```

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

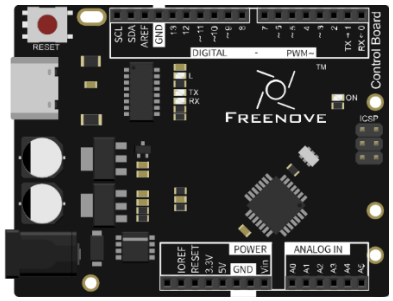
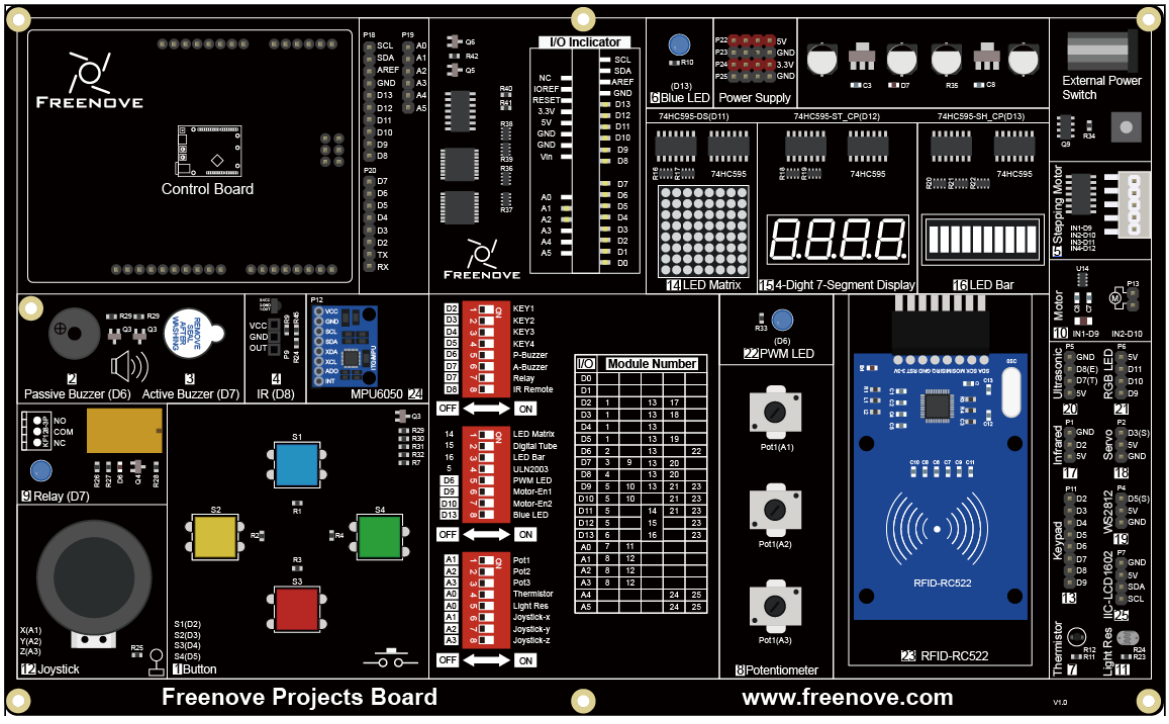
Chapter 3 Control LED with Push Button Switch

In the previous chapter, we have learned to control LED blinking. Now, let us learn how to control the LED through the buttons.

Project 3.1 Control LED with Button

We will use the Freenove Projects Board to get the status of the push button switch, and show it through LED.

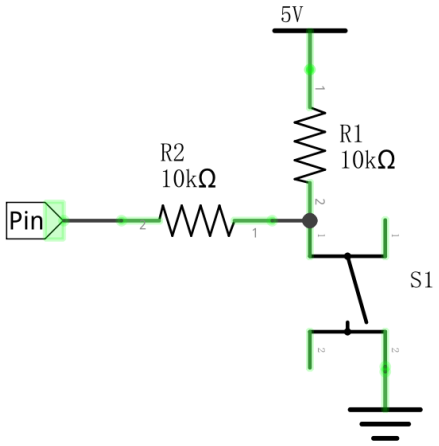
Component List

Control board x1 	USB cable x1 
Freenove Projects Board 	

Circuit Knowledge

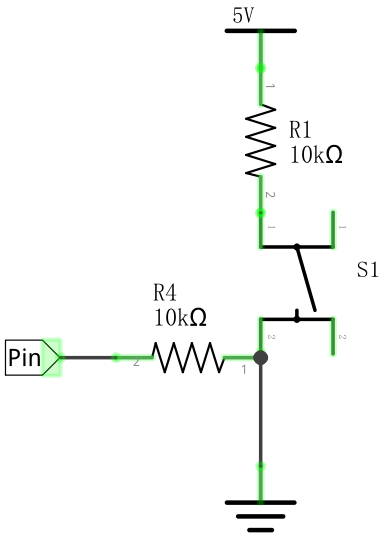
Connection of Push Button Switch

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.

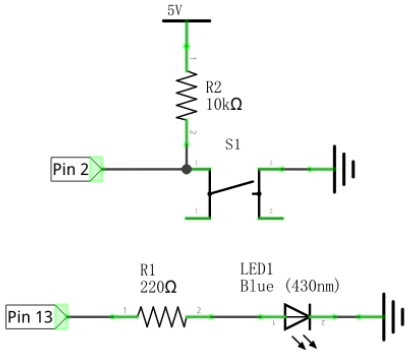
The following diagram shows another connection, in which the level detected by the control board port is opposite to the above diagram, whenever the button is pressed or not.



Circuit

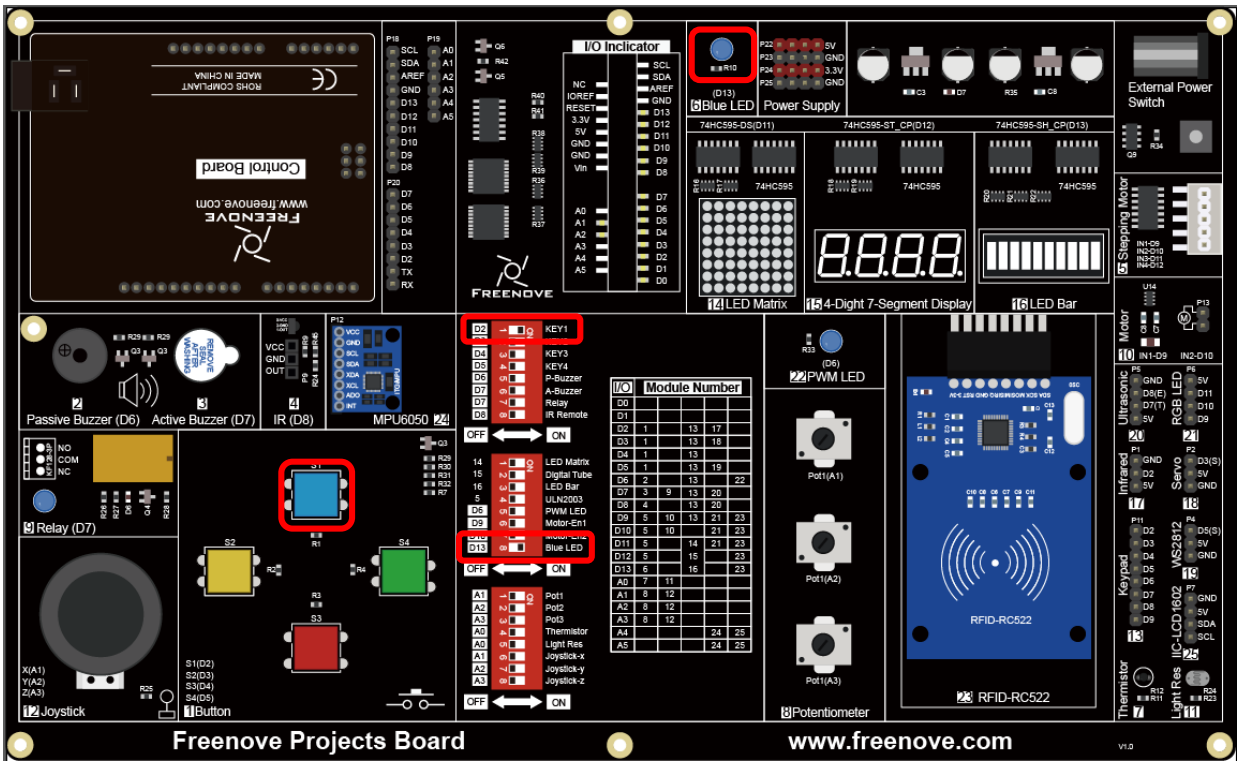
Use pin 2 of control board to detect the status of push button, and pin 13 to drive LED.

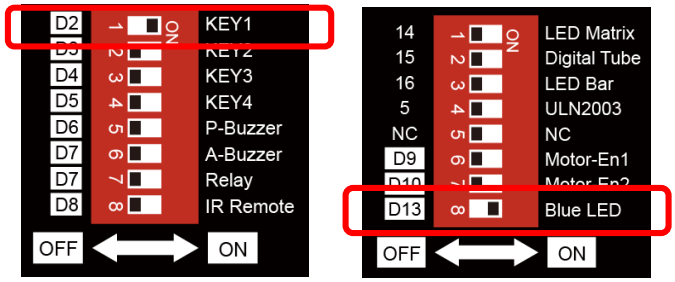
Schematic diagram



Hardware connection

Insert the Control Board to Freenove Projects Board, and then turn the corresponding switches to the right(ON).





Sketch

Control_LED_by_Button

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

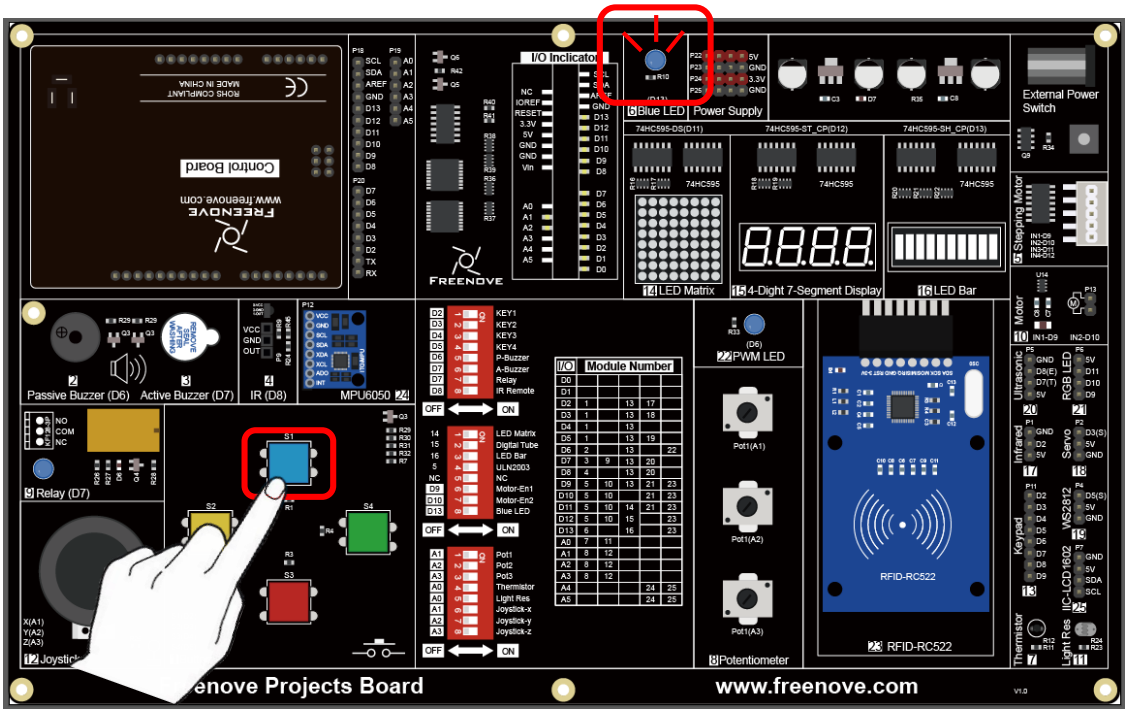
```
int buttonPin = 2; // the number of the push button pin
int ledPin = 13;     // the number of the LED pin

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 3.2 Change LED State by Button

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.

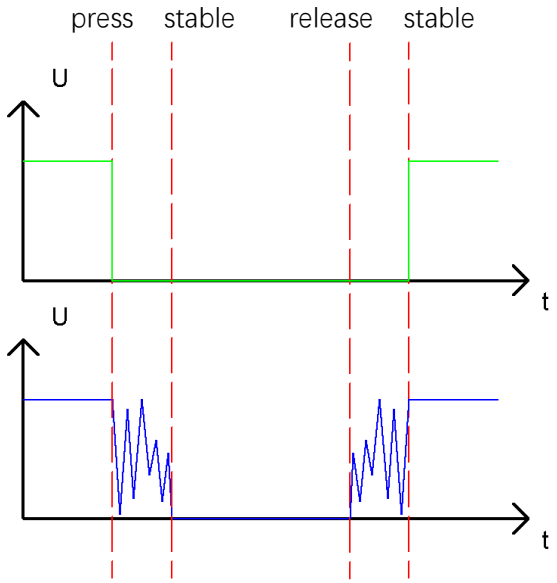
Component List

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

Change_LED_State_by_Button

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 = 2;    // the number of the push button pin
int ledPin = 13;      // 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 4 Serial

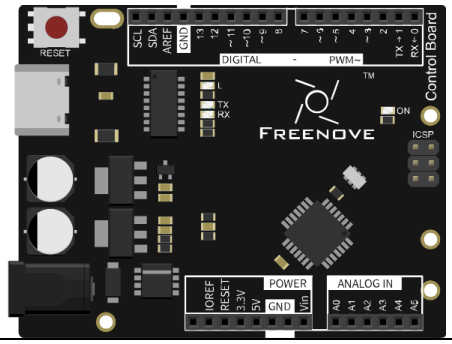
Earlier, we have already tried to output signals to LED, and get the input signal of push button switch. Now, we can try serial communication, a more advanced means of communication.

Project 4.1 Send Data through Serial

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

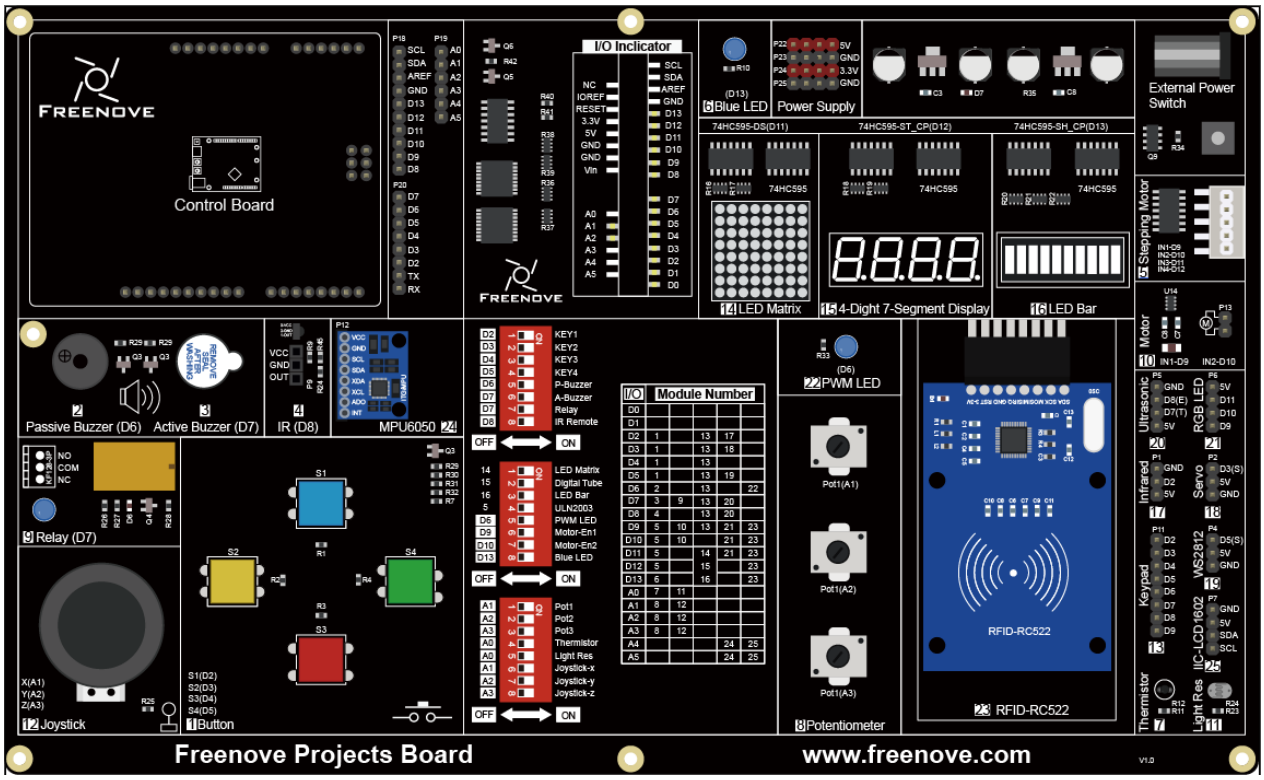
Component List

Control board x1



USB cable x1

Freenove Projects Board



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 converting a binary number to decimal number, first, multiply the nth number of it by n power of 2, and then sum up 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:

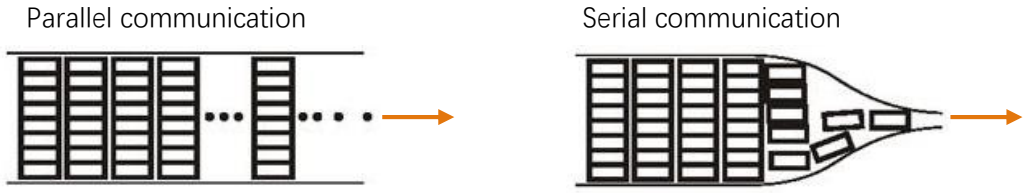
		Remainder	Sequence
2	150	0	0
2	75	1	1
2	37	1	2
2	18	0	3
2	9	1	4
2	4	0	5
2	2	0	6
2	1	1	7
	0		

The result is 10010110.

Circuit Knowledge

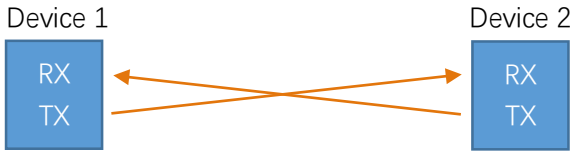
Serial and parallel communication

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:

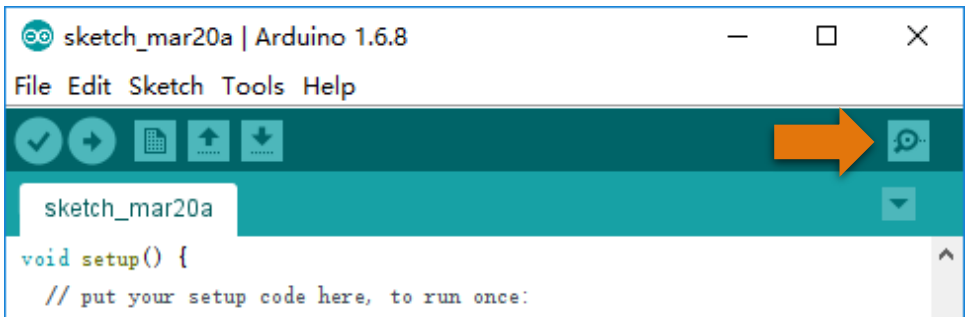


Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

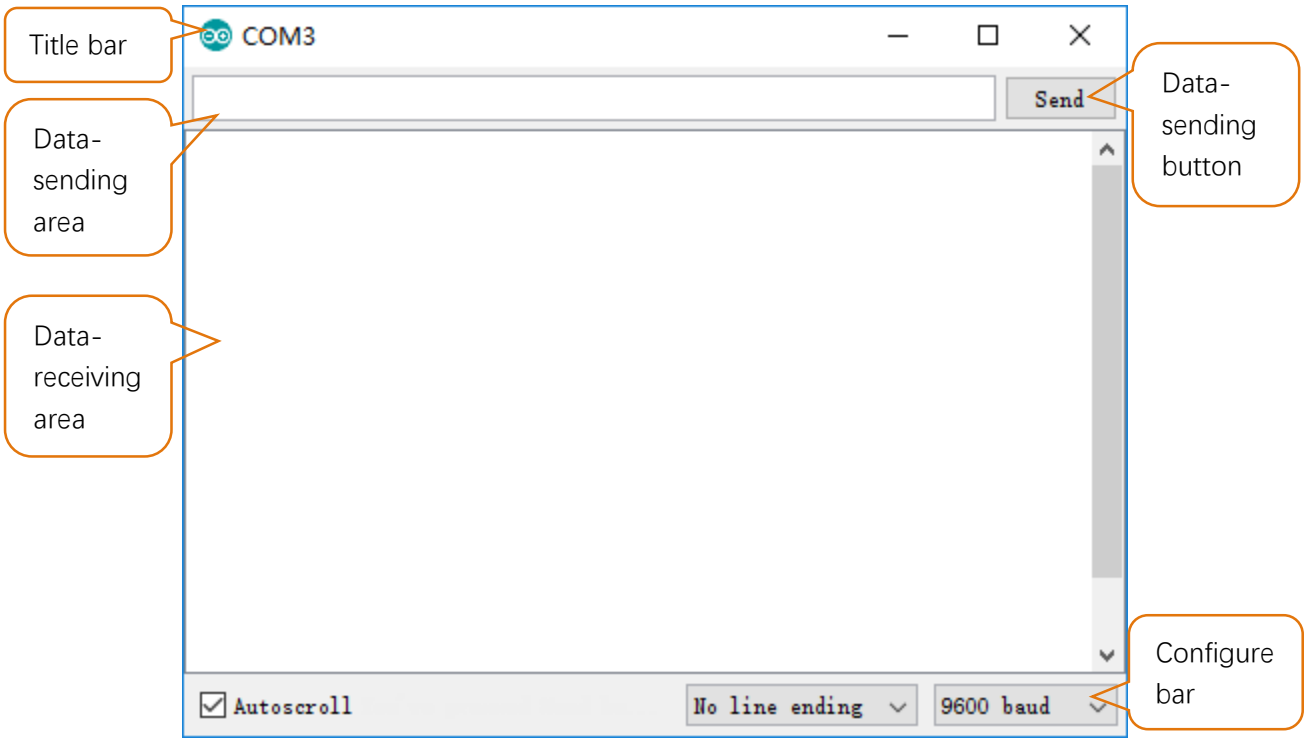
Serial port on Control board

Our control board has integrated USB to serial transfer, so it can communicate with computer when USB cable is 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.

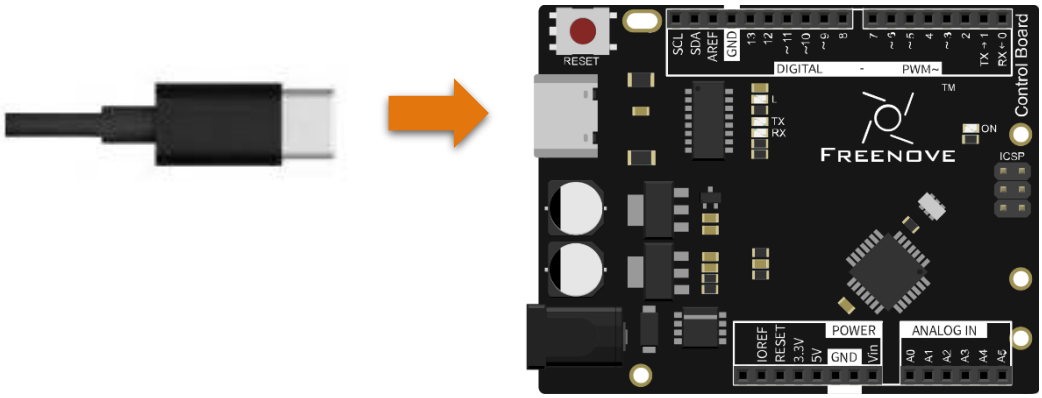


Interface of Serial Monitor window is as follows. If you can't open it, make sure control board had been connected to the computer, and choose the correct serial port in the menu bar "Tools".



Circuit

Connect control board to the computer with USB cable.



Sketch

Send_data_through_Serial

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:

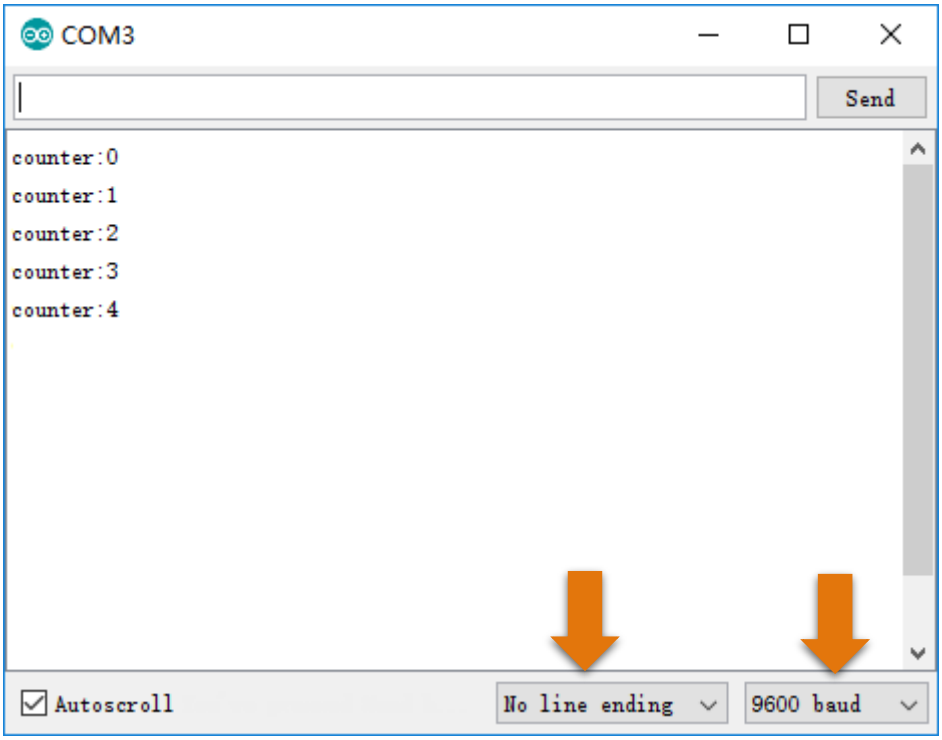
Serial.begin(speed): Initialize serial port, the parameter is the serial port baud rate;

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 4.2 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

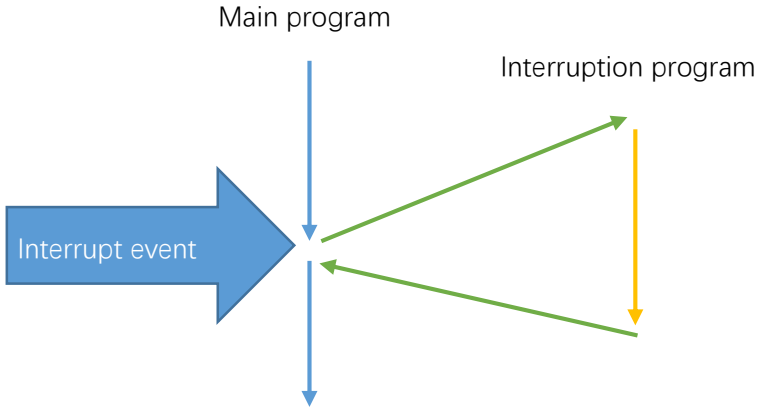
Same with the previous section.

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

Receive_Data_through_Serial_Port

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

```
String str;      // define a variable to store strings received from serial port

void setup() {
  Serial.begin(9600);                 // initialize serial port, set baud rate to 9600
}

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

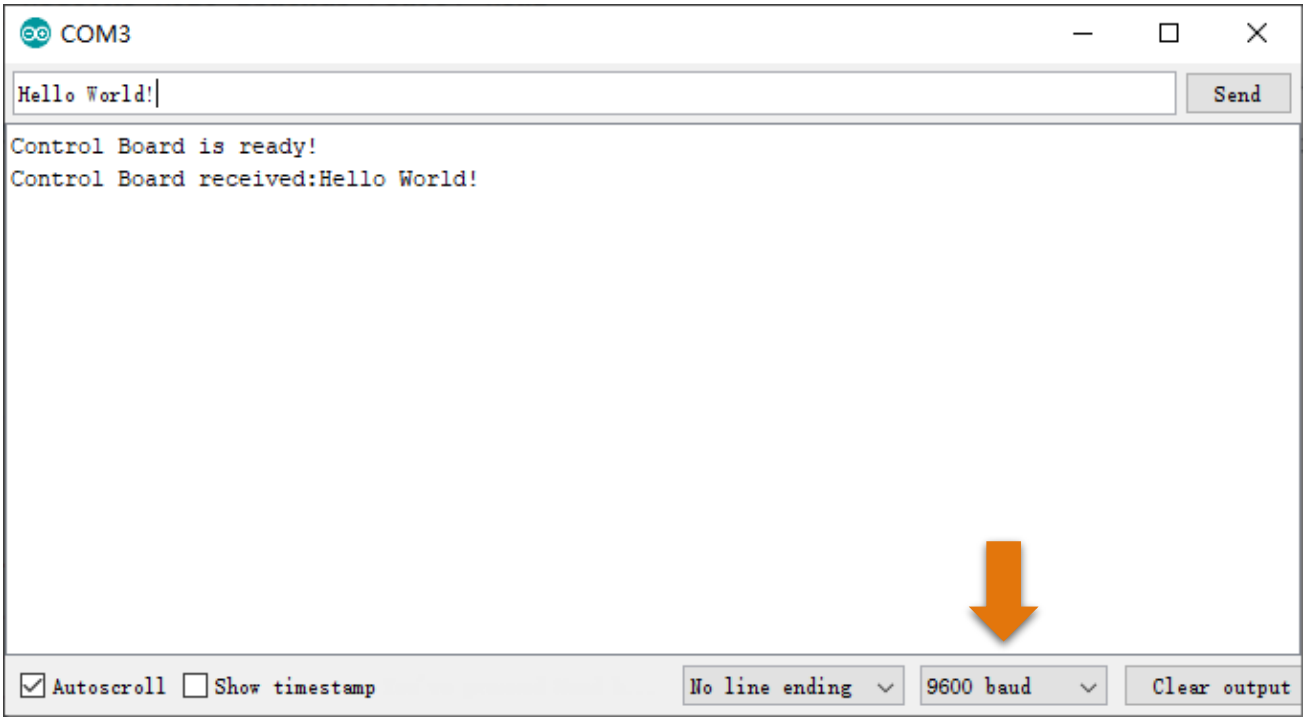
In the setup() function, we initialize the serial port. Then, the loop() function will continuously detect whether there are data to read. If so, it will read the character and send it back.

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.

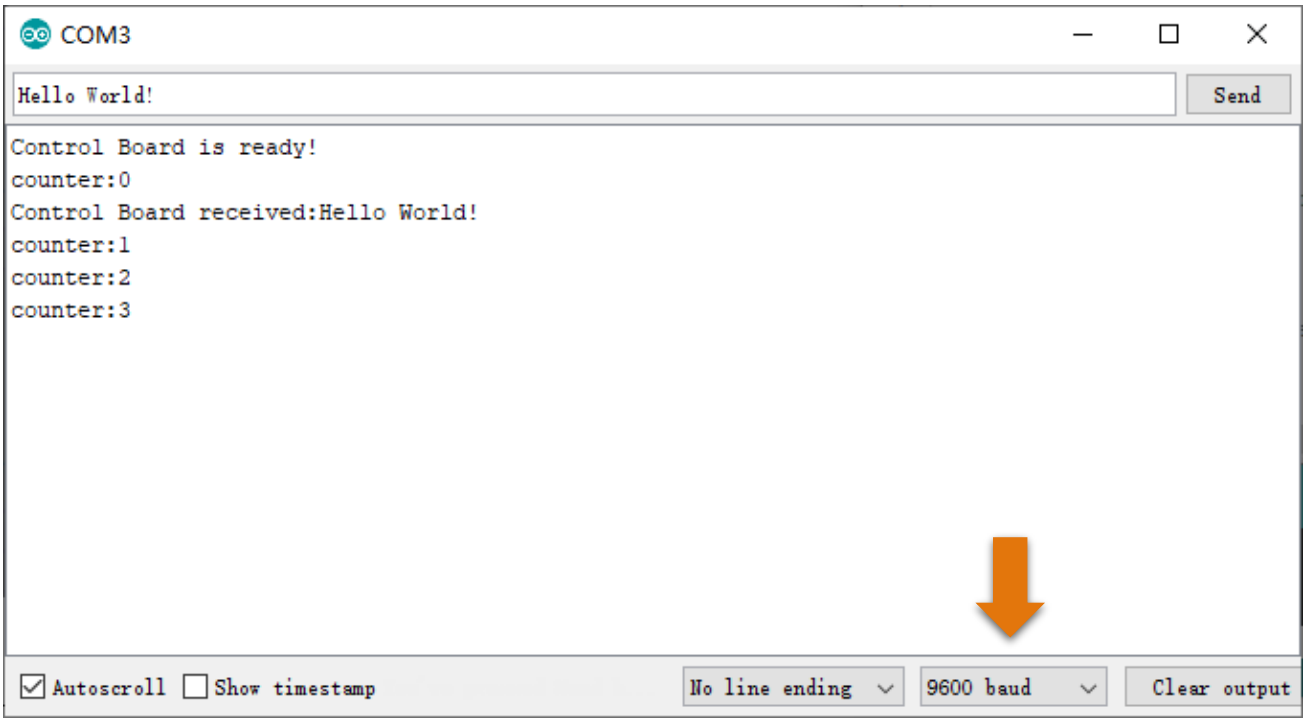
Receive_Data_through_Serial_Port

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  String str;       // 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     str = Serial.readString();            // read one character
19     Serial.print("Control Board received:");  // print the string "Control Board received:"
20     Serial.println(str);            // 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.

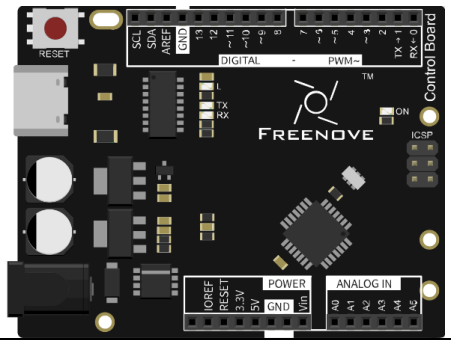


Project 4.3 Application of Serial

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

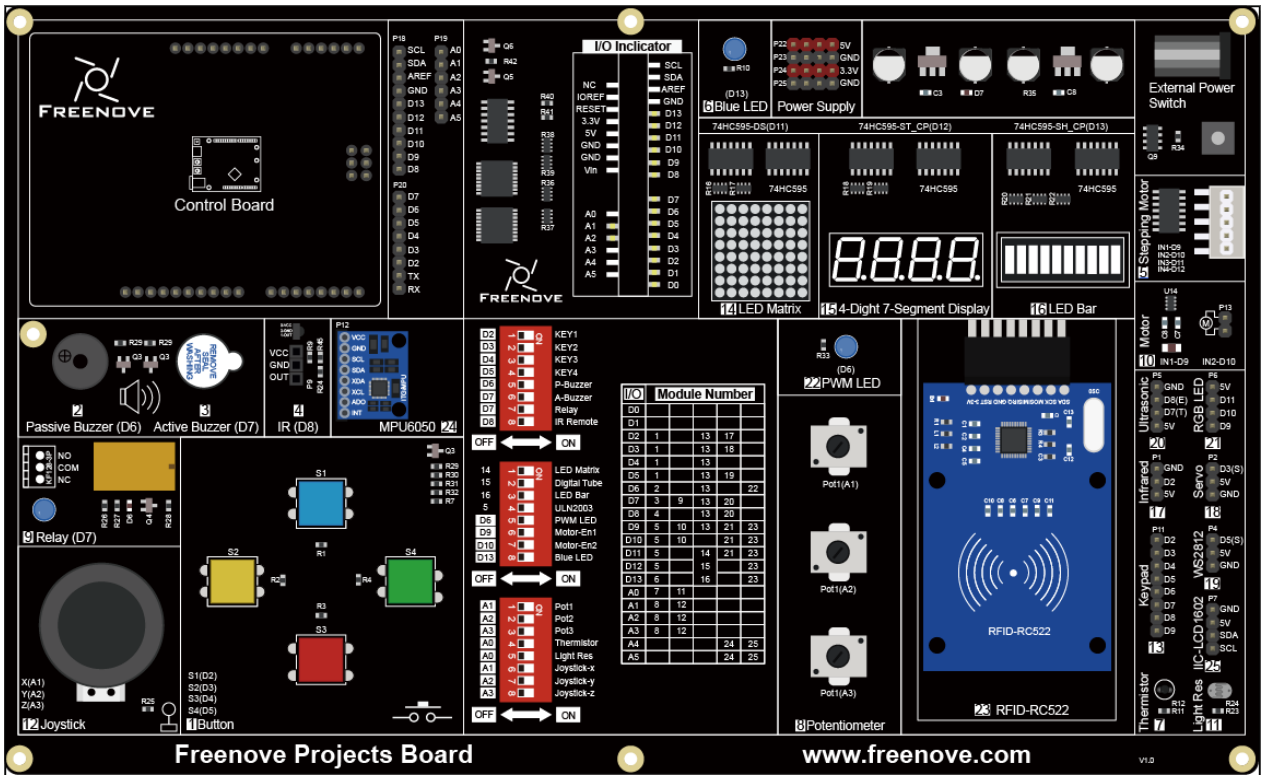
Component List

Control board x1



USB cable x1

Freenove Projects Board



Circuit Knowledge

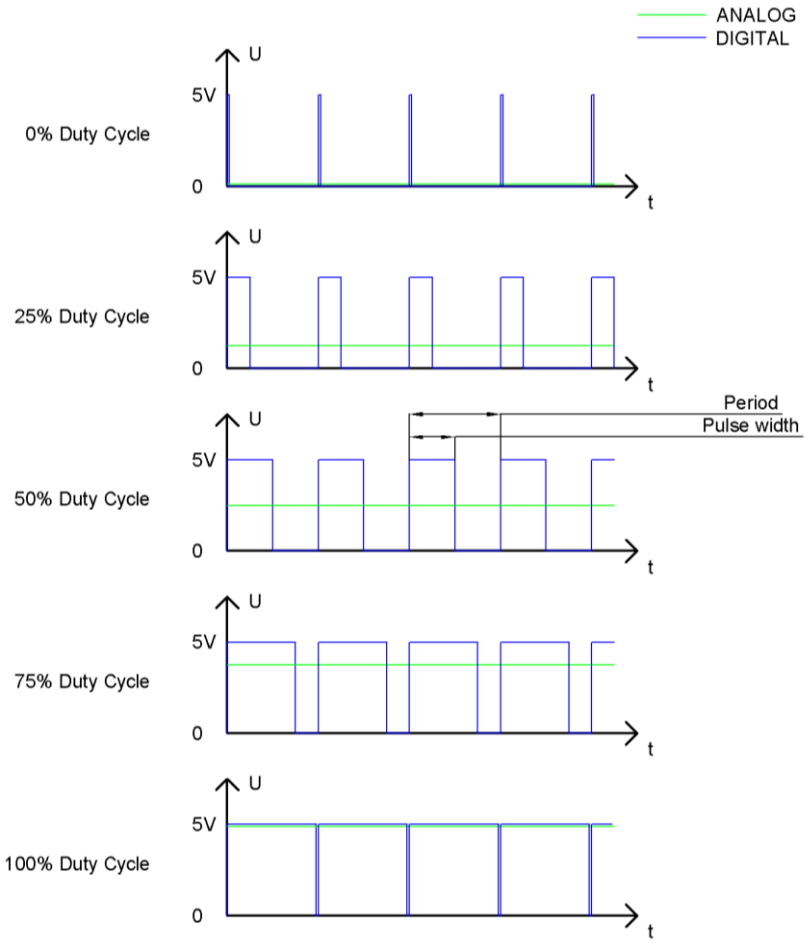
At first, let us learn how to use the circuit to make LED emit different brightness of light,

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%:

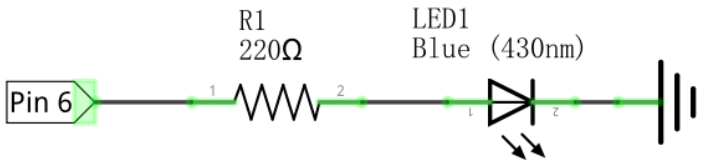


The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

Circuit

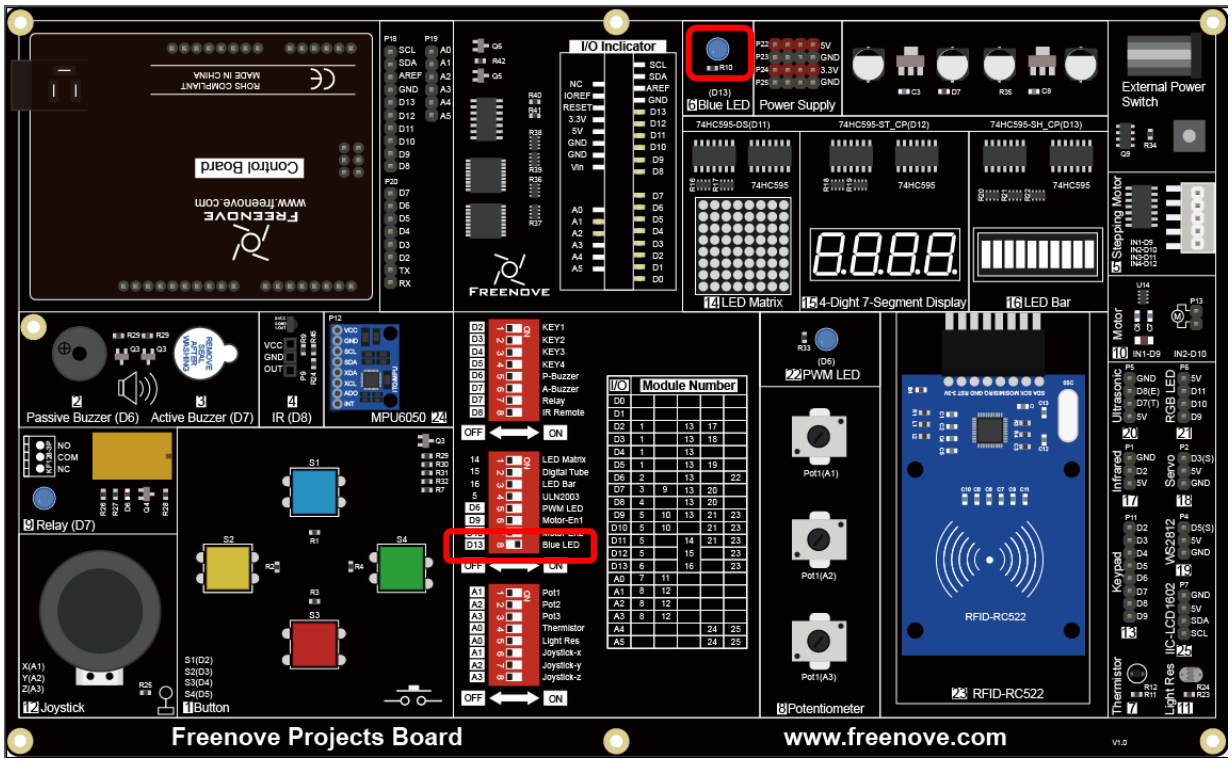
Here we will use pin 6 on the control board to drive 1 LED.

Schematic diagram



Hardware connection

Insert the Control Board to Freenove Projects Board, and then turn the corresponding switches to the right(ON).



Sketch

Application_of_Serial

```
1  int inInt;        // define a variable to store the data received from serial
2  int counter = 0;  // define a variable as the data sending to serial
3  int ledPin = 6;   // the number of the LED pin
4
5  void setup() {
6    pinMode(ledPin, OUTPUT);             // initialize the LED pin as an output
```

```
7    Serial.begin(9600);                    // initialize serial port, set baud rate to 9600
8    Serial.println("Control Board is ready!");     // print the string "Control Board is ready!"
9  }
10
11 void loop() {
12   if (Serial.available()) {          // judge whether the data has been received
13     inInt = Serial.parseInt();       // read an integer
14     Serial.print("Control Board received:");  // print the string "Control Board received:"
15     Serial.println(inInt);           // print the received character
16     counter = constrain(inInt,0,255);
17     digitalWrite(ledPin, counter);
18   }
19 }
```

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

Serial Class

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

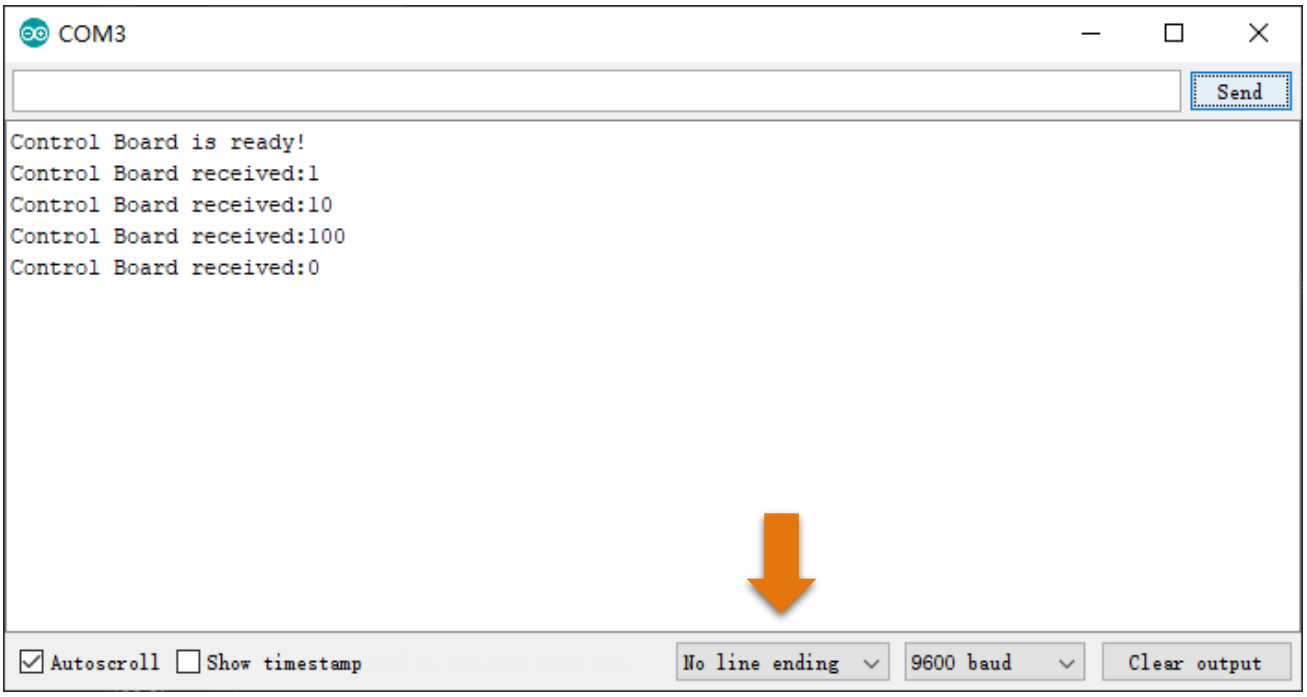
constrain(x, a, b)

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

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.

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.


COM3
Send
Control Board is ready!
Control Board received:1
Control Board received:10
Control Board received:100
Control Board received:0
Autoscroll
Show timestamp
No line ending
9600 baud
Clear output

Chapter 5 ADC

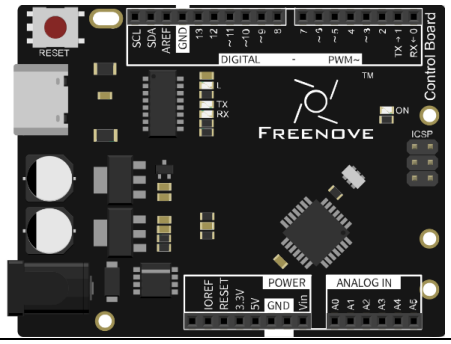
Earlier, we have learned the digital ports of control board, and tried the output and input signals. Now, let us learn how to use analog ports.

Project 5.1 ADC

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

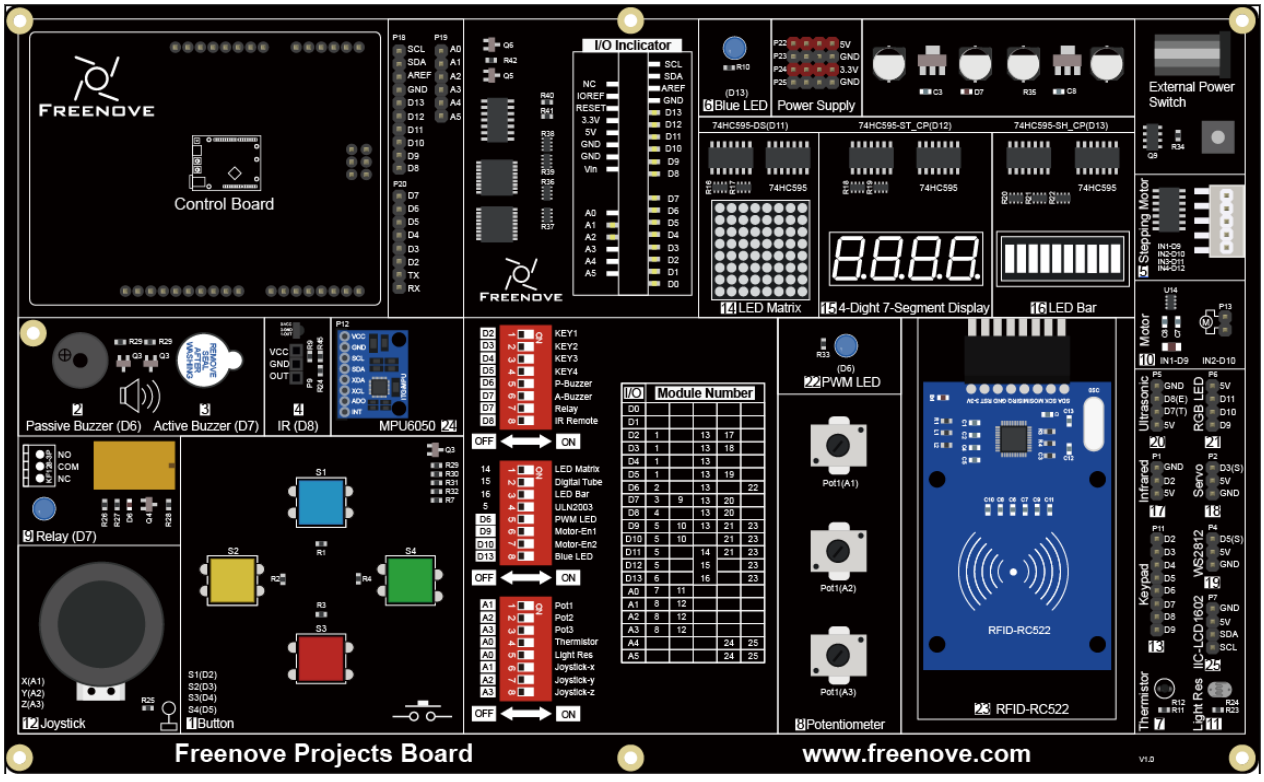
Component List

Control board x1



USB cable x1

Freenove Projects Board

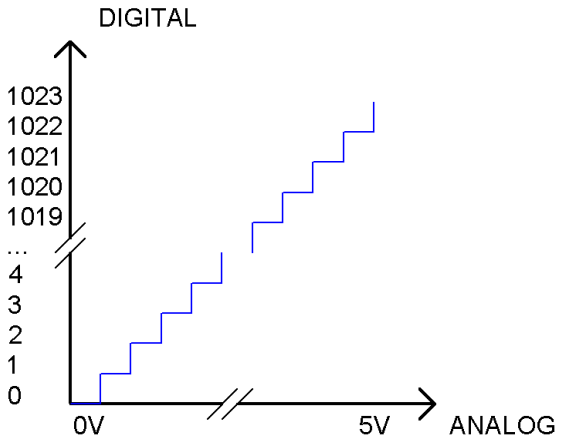


Circuit Knowledge

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;

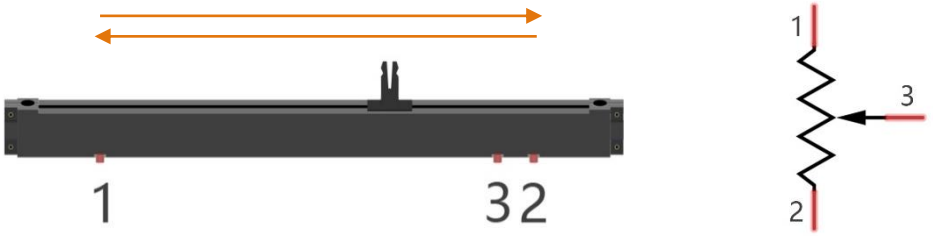
Subsection 2: the analog in rang of 5 /1024V-2*5/1024V corresponds to digital 1;

The following analog signal will be divided accordingly.

Component Knowledge

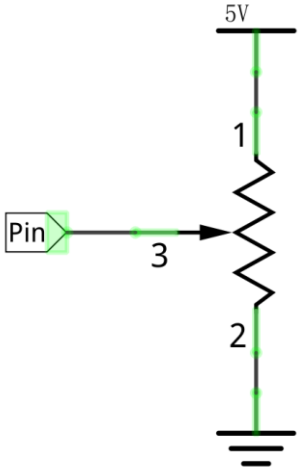
Potentiometer

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.



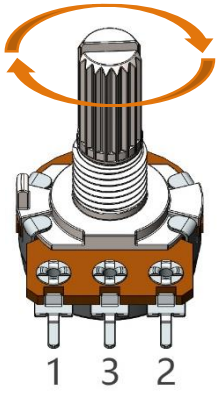
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly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

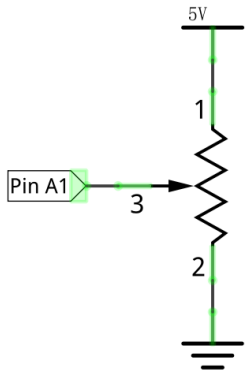
Rotary potentiometer and linear potentiometer have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



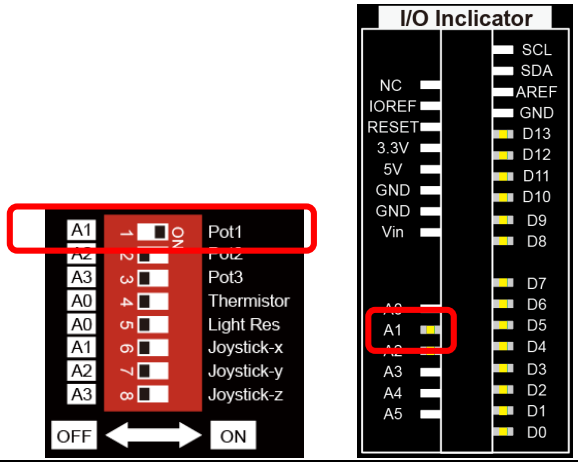
Circuit

Use pin A1 on the control board to detect the voltage of rotary potentiometer.

Schematic diagram

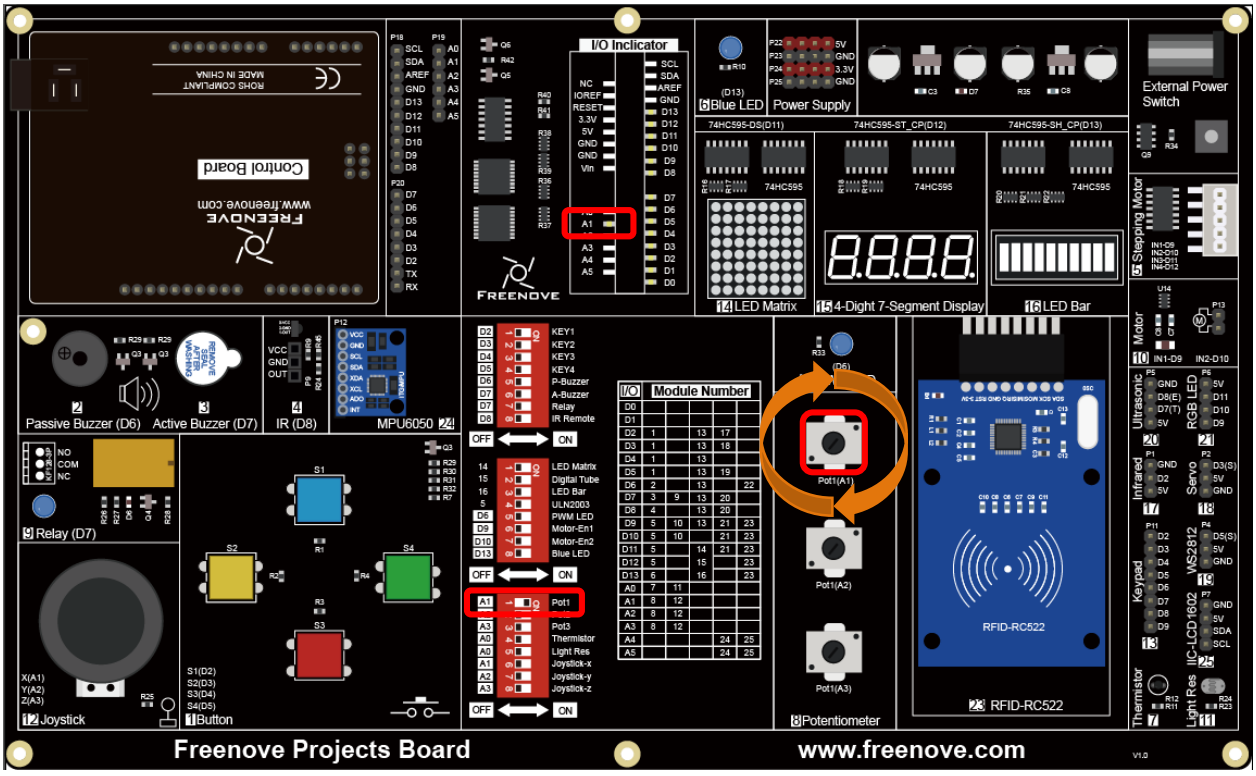


Hardware connection



Hardware connection

Insert the Control Board to Freenove Projects Board, and then turn the corresponding switch to the right(ON).



Sketch

ADC

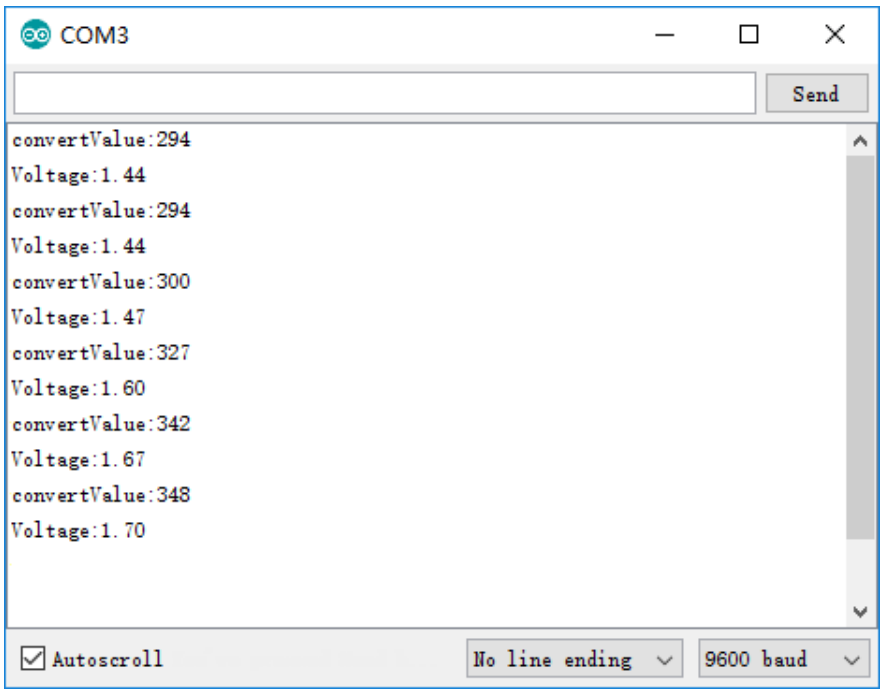
Now, write code to detect the voltage of rotary potentiometer, and send the data to Serial Monitor window of Arduino IDE through serial port.

```
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(A1);          // Convert analog of pin A1 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 A1, 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.

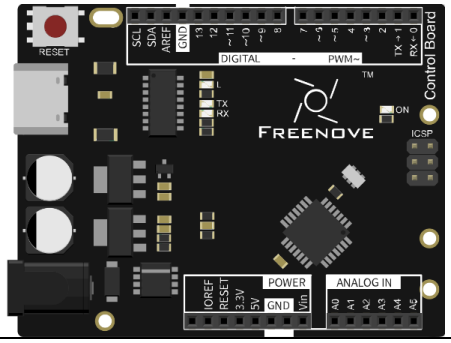


Project 5.2 Control LED by Potentiometer

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.

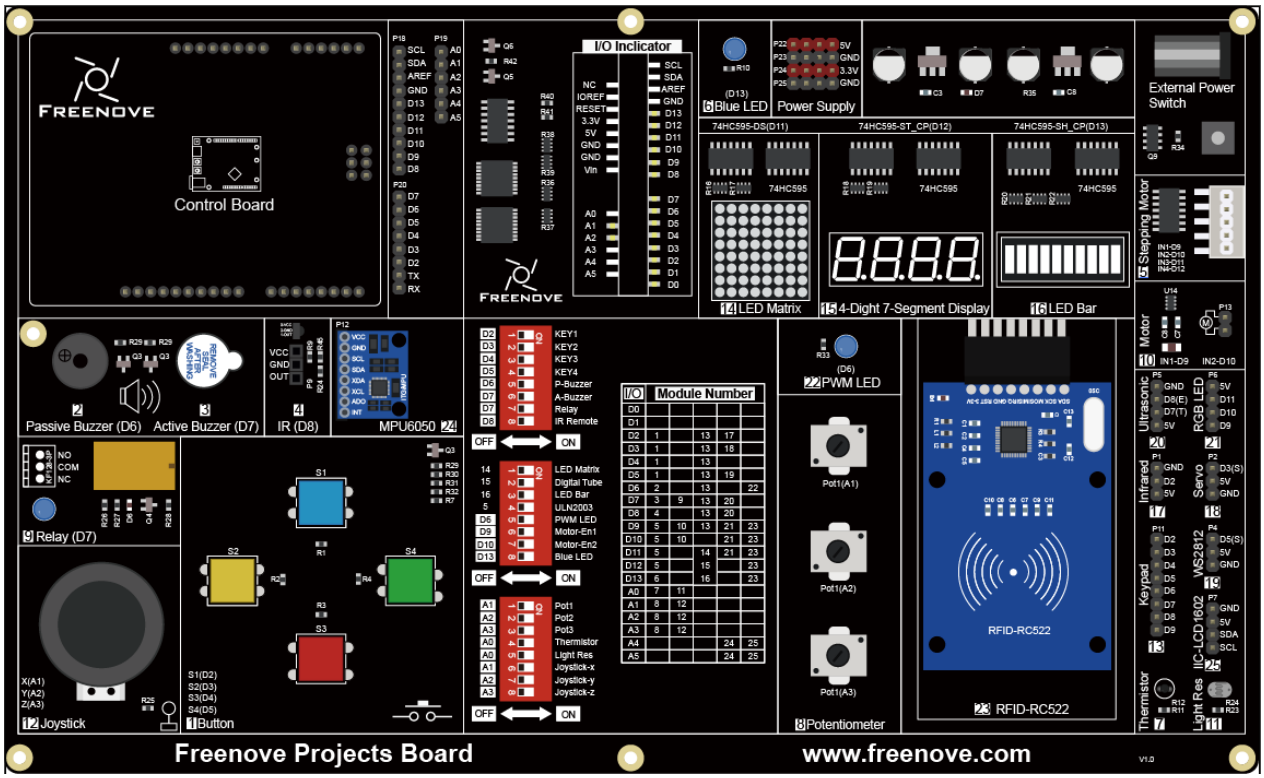
Component List

Control board x1



USB cable x1

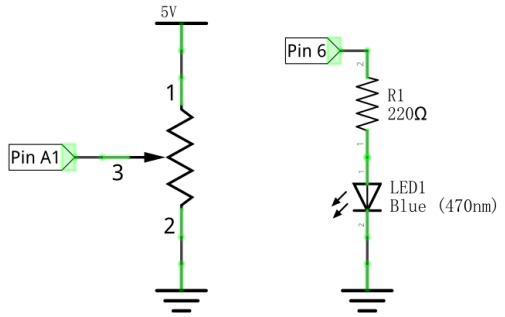
Freenove Projects Board



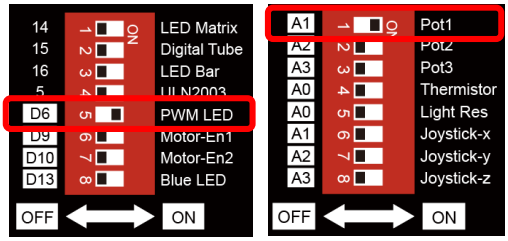
Circuit

Use pin A1 on control board to detect the voltage of rotary potentiometer, and use pin 6 to control one LED.

Schematic diagram

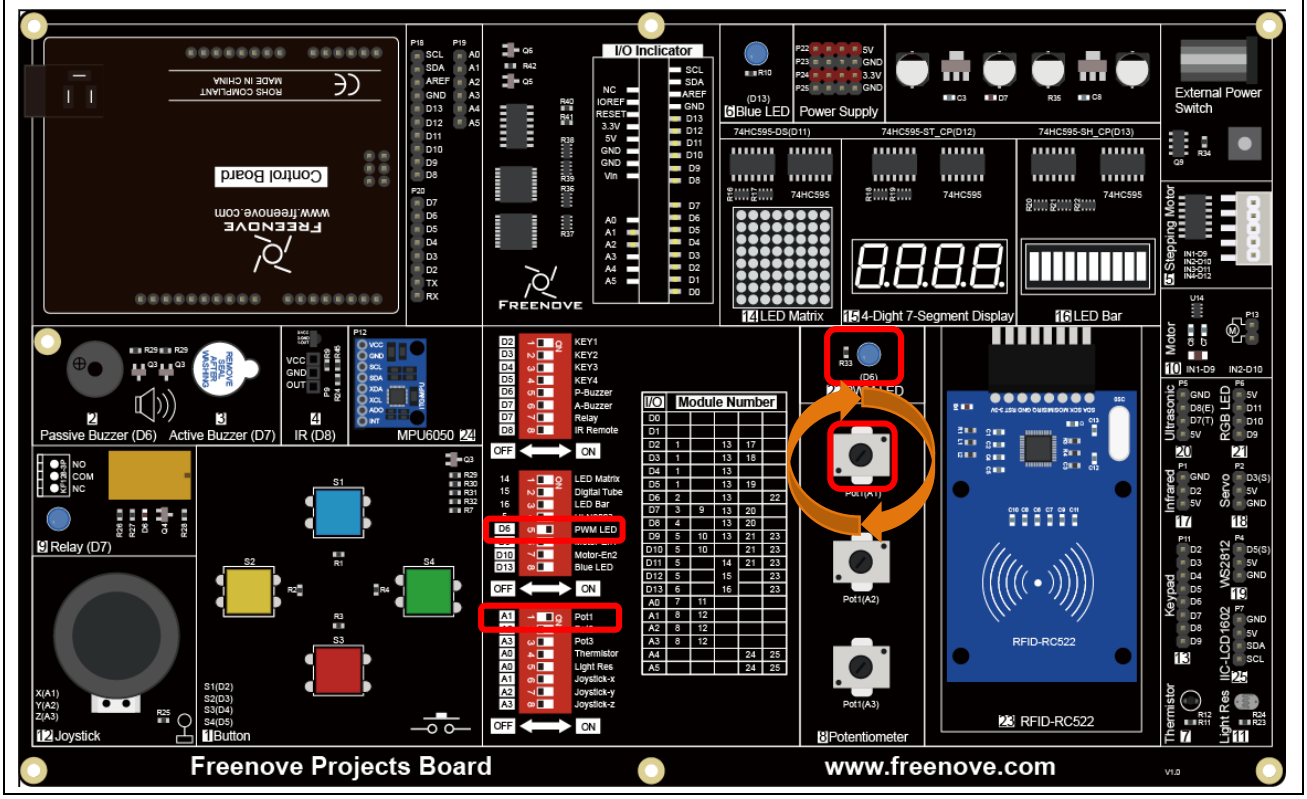


Hardware connection



Hardware connection

Insert the Control Board to Freenove Projects Board, and then turn the corresponding switches to the right(ON).



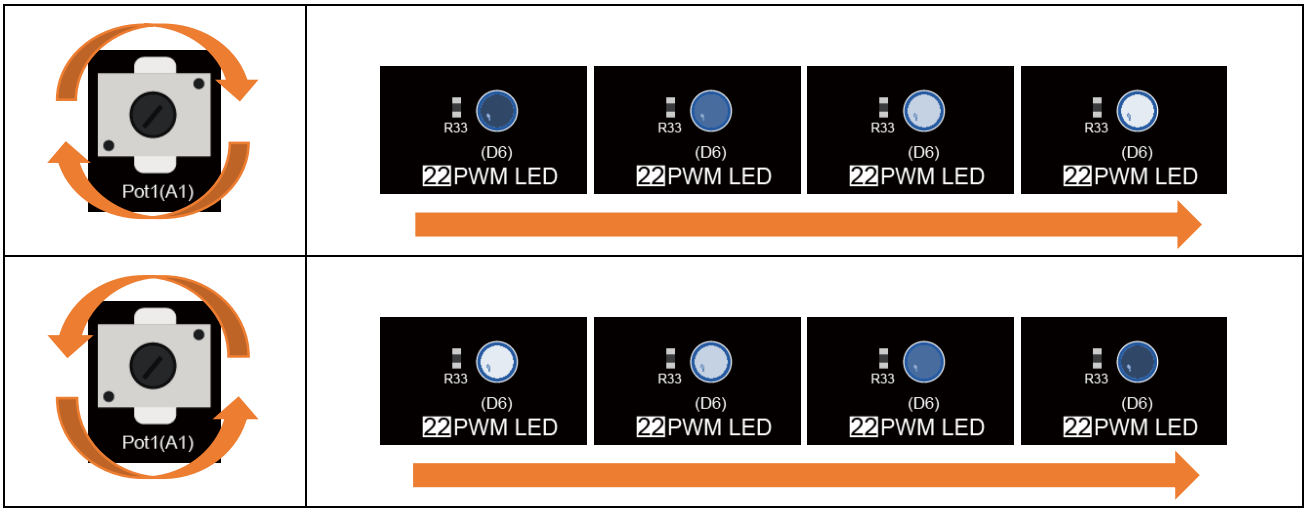
Sketch

Control_LED_by_Potentiometer

```
int adcValue;     // Define a variable to save the ADC value
int ledPin = 6;   // Use pin 6 on control board to control the LED

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

void loop() {
  adcValue = analogRead(A1);            // Convert the analog of A1 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 A1 and map it to PWM duty cycle of LED pin port. With different LED brightness, we can see the changes of voltage easily.

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



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

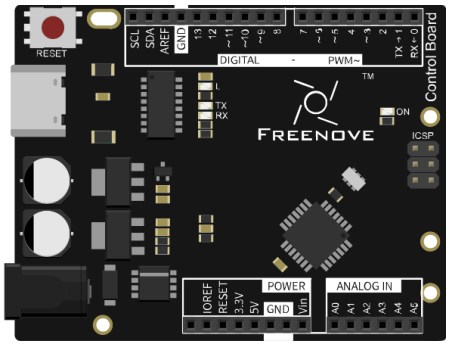
This function is used to remap a value, which will return a new value whose percentage in the range of toLow-toHigh is equal to the percentage of "value" in the range of fromLow-fromHigh. For example, 1 is the maximum in the range of 0-1 and the maximum value in the scope of 0-2 is 2, that is, the result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

Project 5.3 Control LED by Photoresistor

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

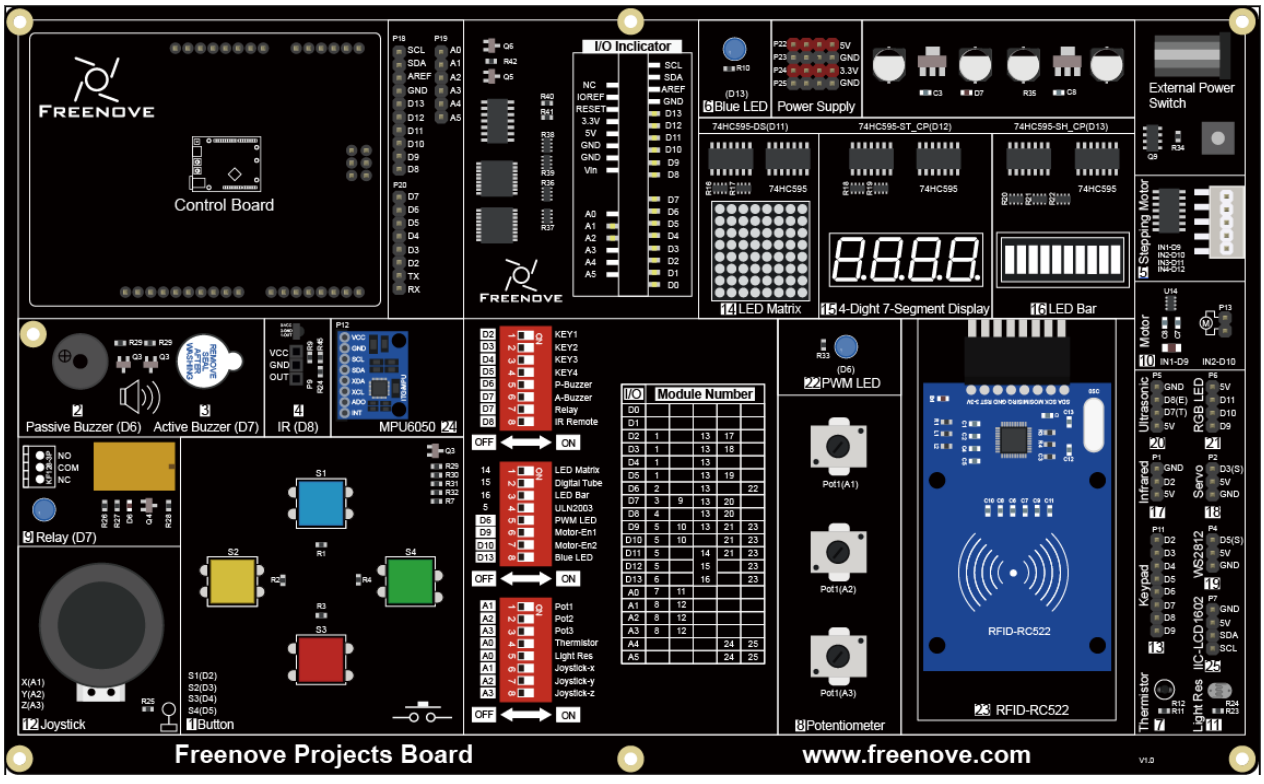
Component List

Control board x1



USB cable x1

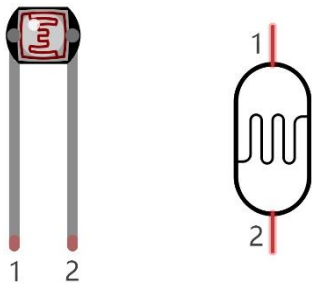
Freenove Projects Board



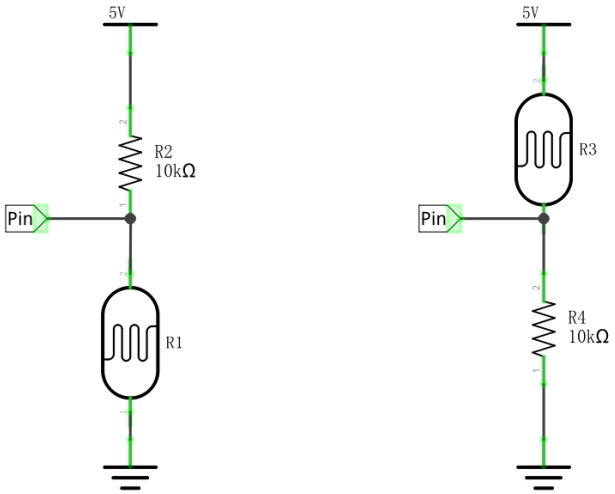
Component Knowledge

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:

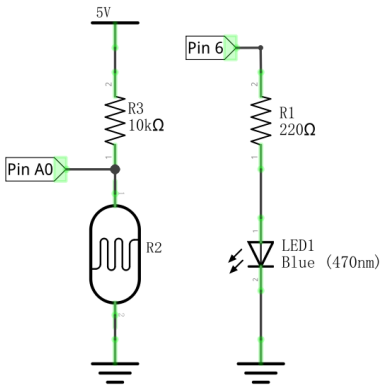


In the above circuit, when a photoresistor's resistance value changes due to a change in light intensity, the voltage between the photoresistor and resistor R1 will also change, so the intensity of the light can be obtained by measuring the voltage.

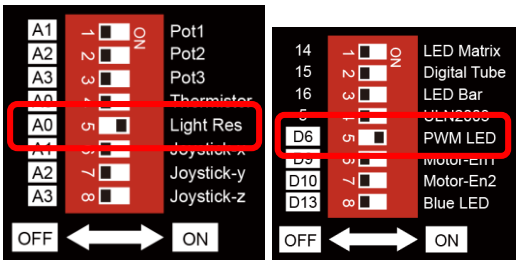
Circuit

Use pin A0 on control board to detect the voltage of photoresistor, and use pin 6 to control one LED.

Schematic diagram

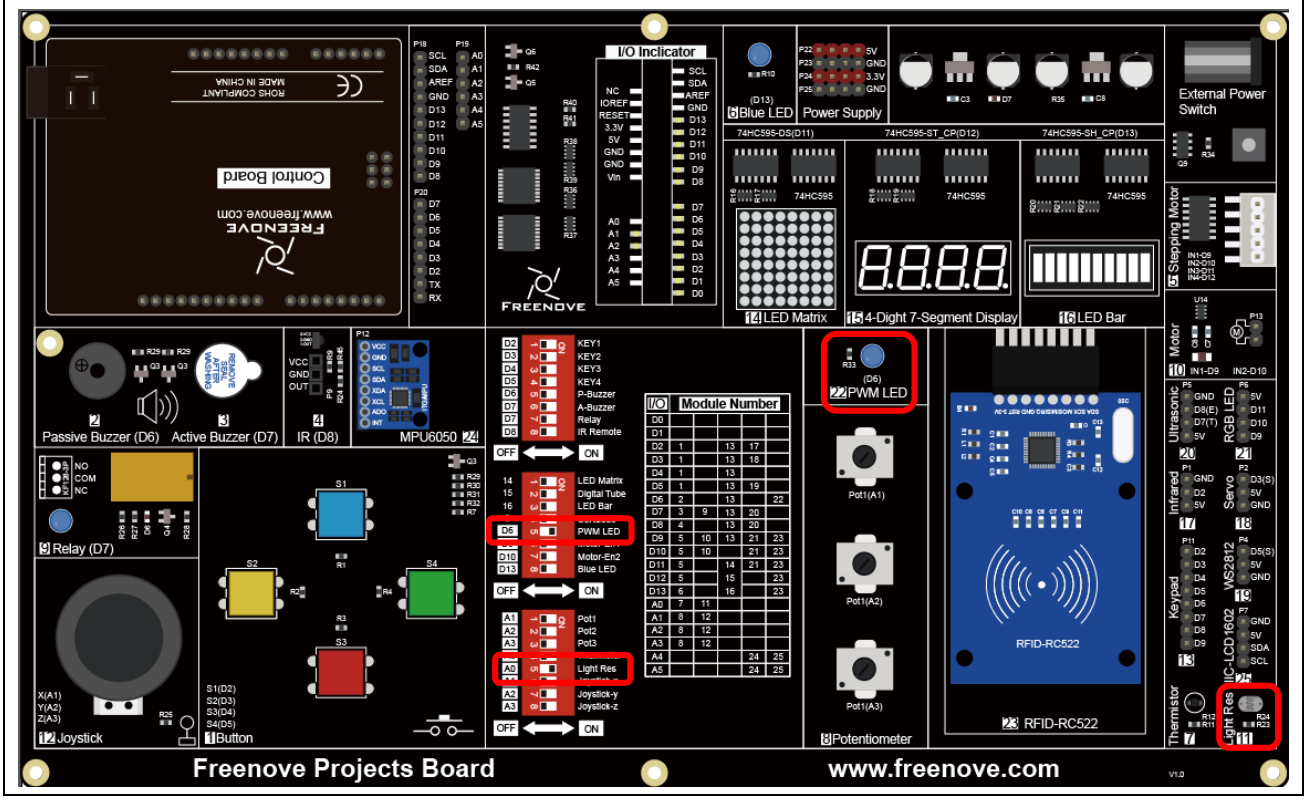


Hardware connection



Hardware connection

Insert the Control Board to Freenove Projects Board, and then turn the corresponding switches to the right(ON).



Sketch

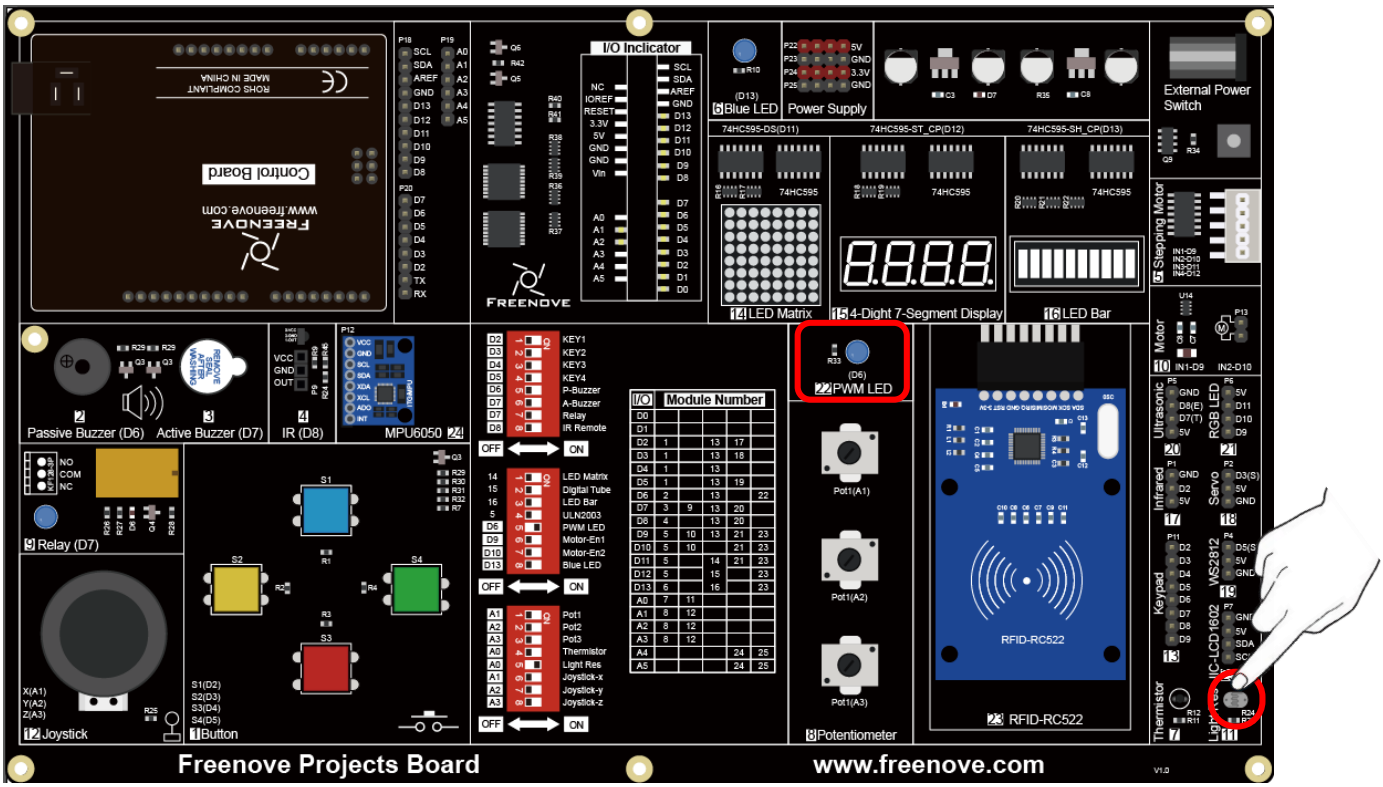
Control_LED_through_Photoresistor

Now, write code to detect the voltage of the 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 = 6;   // 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's brightness changes.



Chapter 6 RGB LED

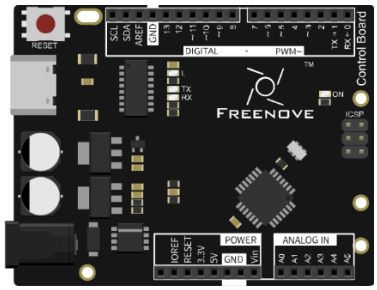
Earlier, we have learned to use the analog port and ADC of the control board. Now, we'll use ADC to control RGB LED.

Project 6.1 Control RGB LED through Potentiometer

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

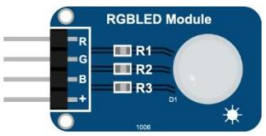
Component List

Control board x1



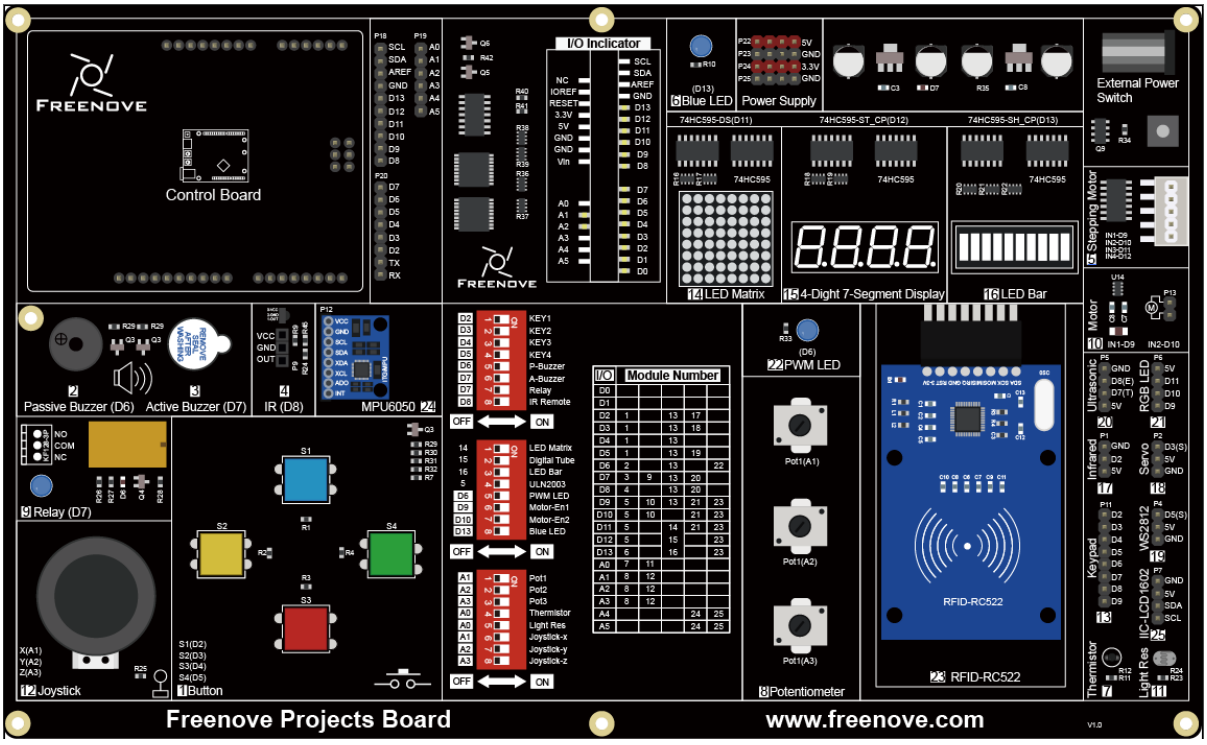
USB cable x1

RGB LED x1



Jumper Wire x4

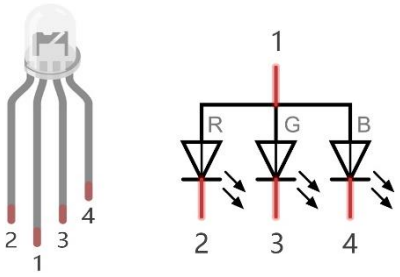
Freenove Projects Board



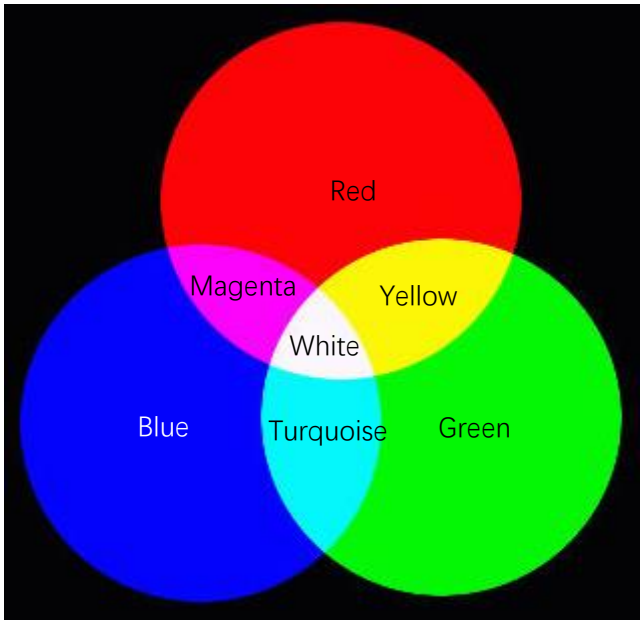
Component Knowledge

RGB LED

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

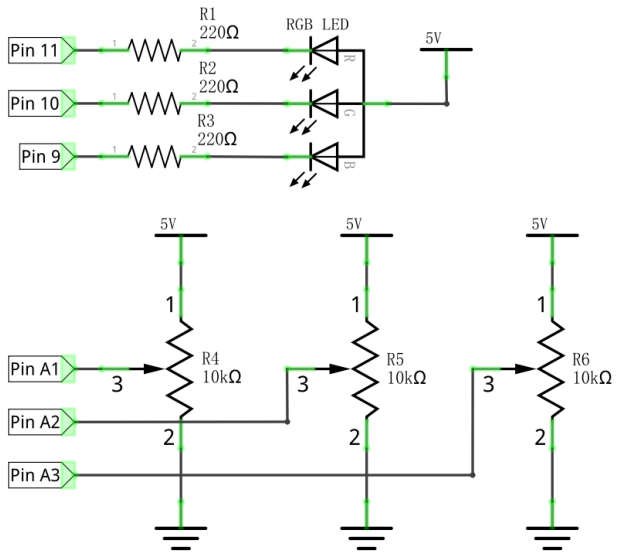


We know from the previous section that, control board controls LED to emit a total of 256(0-255) different brightness via PWM. Therefore, through different combinations of RGB light brightness, we can create 256^3=16777216(16Million) colors.

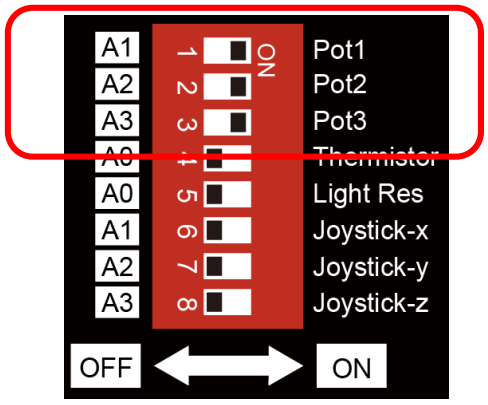
Circuit

Use pin A1, A2, A3 ports of the control board to detect the voltage of rotary potentiometer, and control RGB LED by pin 9, 10, 11.

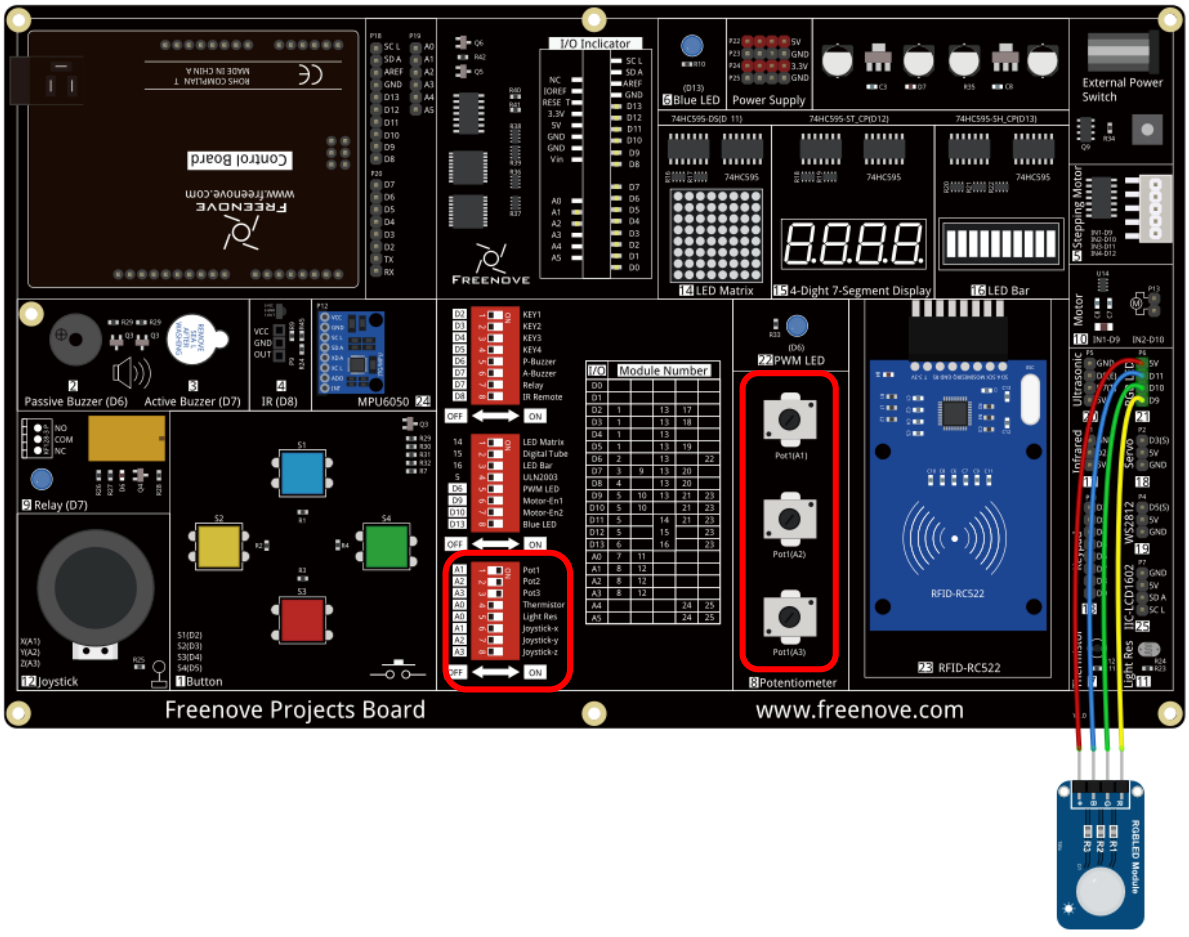
Schematic diagram



Hardware connection



Hardware connection



Sketch

Control_RGB_LED_through_Potentiometer

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 A1 port into digital, and work as PWM duty cycle of ledPinR port
  adcValue = analogRead(A1);
  analogWrite(ledPinR, map(adcValue, 0, 1023, 0, 255));
  // Convert analog of A2 port into digital, and work as PWM duty cycle of ledPinG port
  adcValue = analogRead(A2);
  analogWrite(ledPinG, map(adcValue, 0, 1023, 0, 255));
  // Convert analog of A3 port into digital, and work as PWM duty cycle of ledPinB port
  adcValue = analogRead(A3);
  analogWrite(ledPinB, map(adcValue, 0, 1023, 0, 255));
}
```

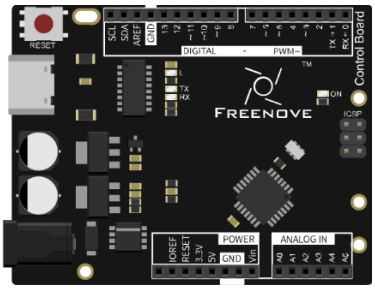
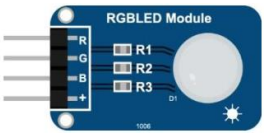
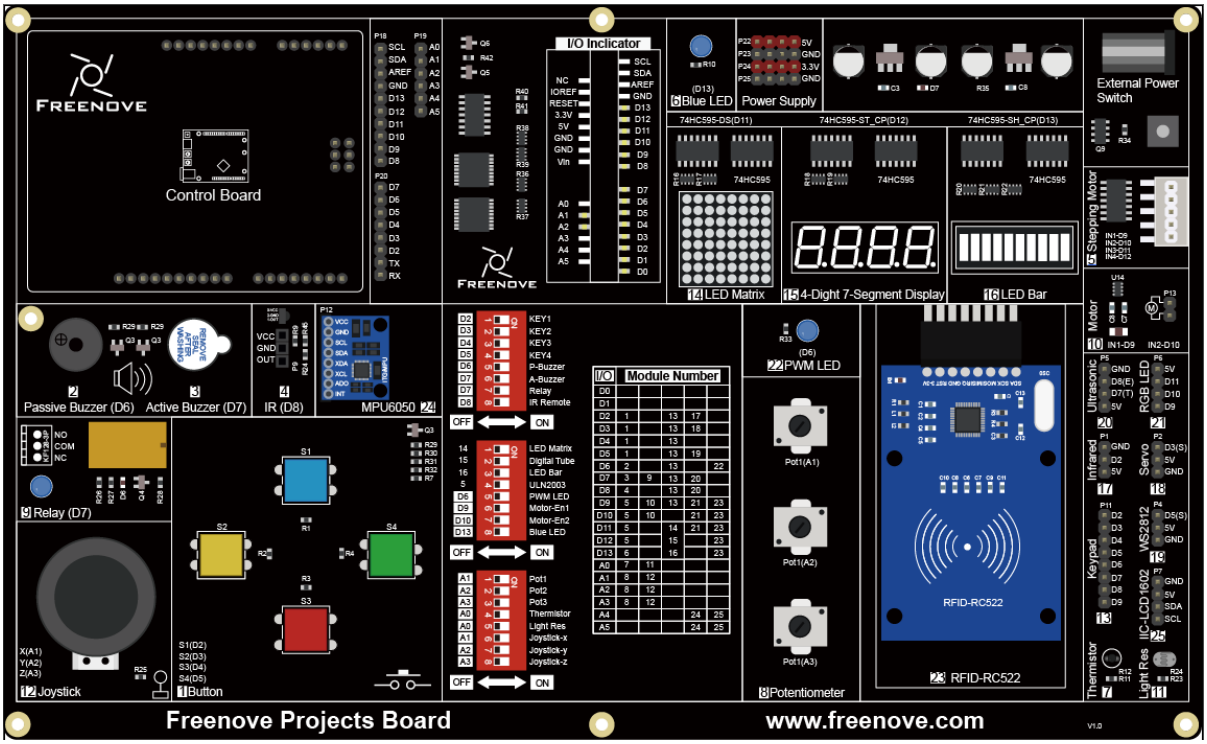
In the code, we get the voltages of three rotary potentiometers, and convert them into PWM duty cycle to control the three LEDs of the RGB LED to emit light with different brightness.

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

Project 6.2 Multicolored LED

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

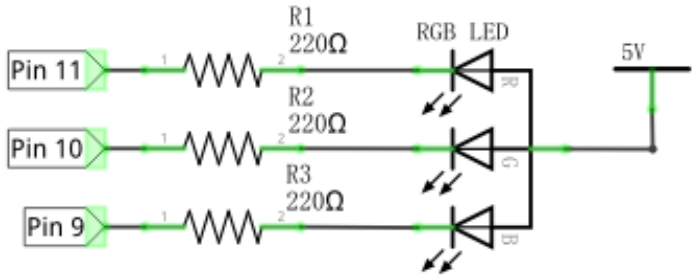
Component List

Control board x1 	USB cable x1 	RGB LED x1 
Jumper Wire x4 		
Freenove Projects Board 		

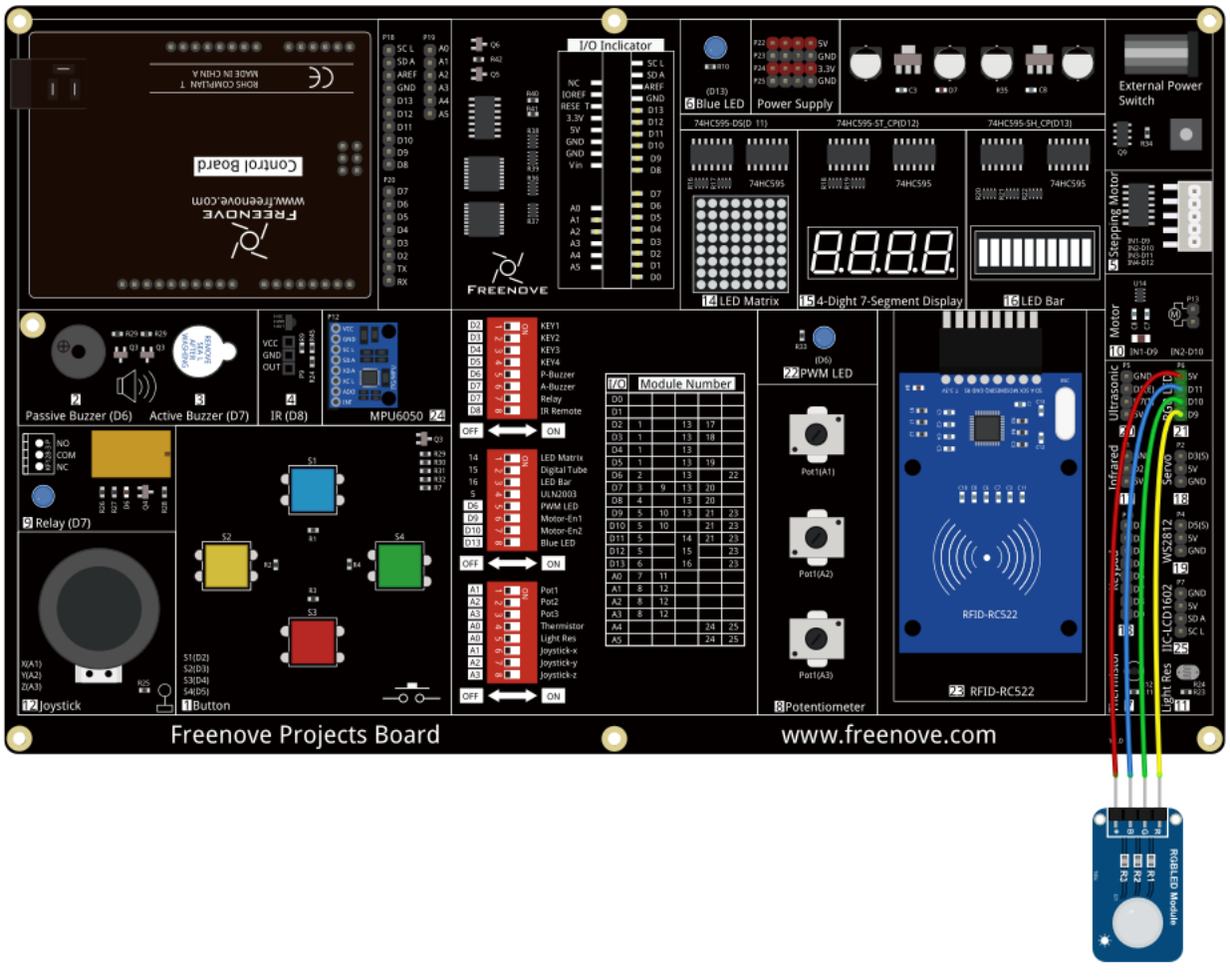
Circuit

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

Schematic diagram



Hardware connection



Sketch

Colorful_LED

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.

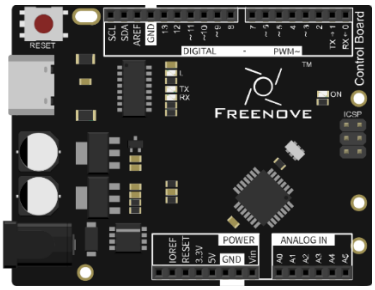
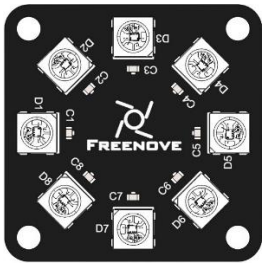
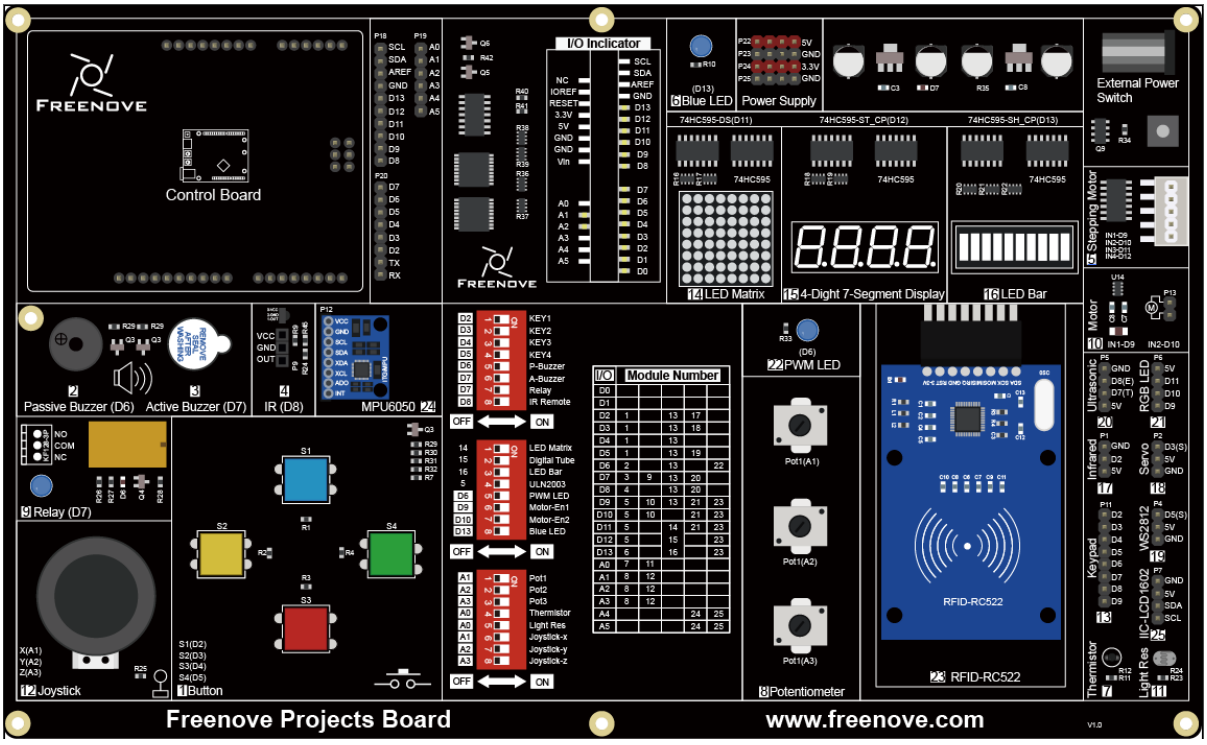
Chapter 7 LEDPixel

This chapter will help you learn to use a more convenient WS2812 LED, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 7.1 LEDPixel

Learn the basic usage of LEDPixel and make it blink red, green, blue and white lights.

Component List

Control board x1 	USB cable x1 	LEDPixel x1 
Jumper Wire x3 		
Freenove Projects Board 		

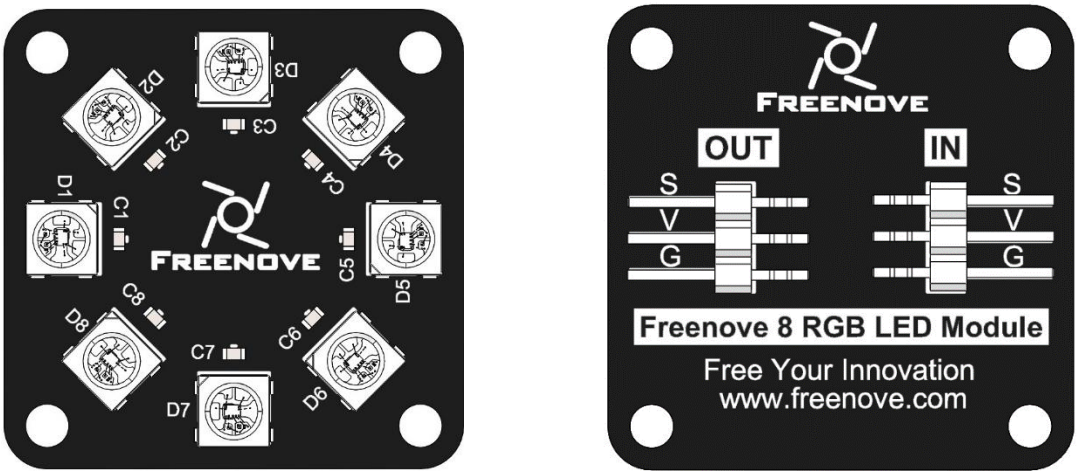
Related knowledge

Freenove 8 RGB LED Module

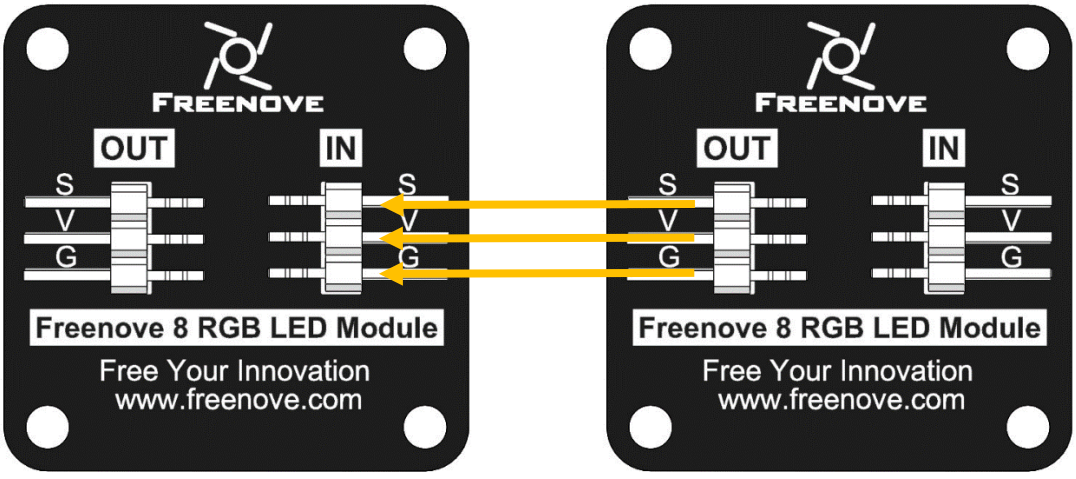
The Freenove 8 RGB LED Module is as below.

It consists of 8 WS2812, each of which requires only one pin to control and supports cascading. Each WS2812 has integrated 3 LEDs, red, green and blue respectively, and each of them supports 256-level brightness adjustment, which means that each WS2812 can emit 2^24=16,777,216 different colors.

You can use only one data pin to control eight LEDs on the module. As shown below:



And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In this way, you can use one data pin to control 8, 16, 32 ··· LEDs.

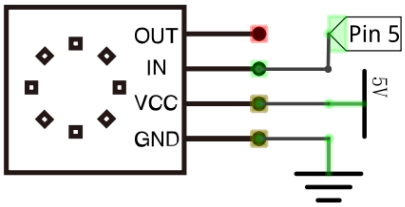


Pin description:

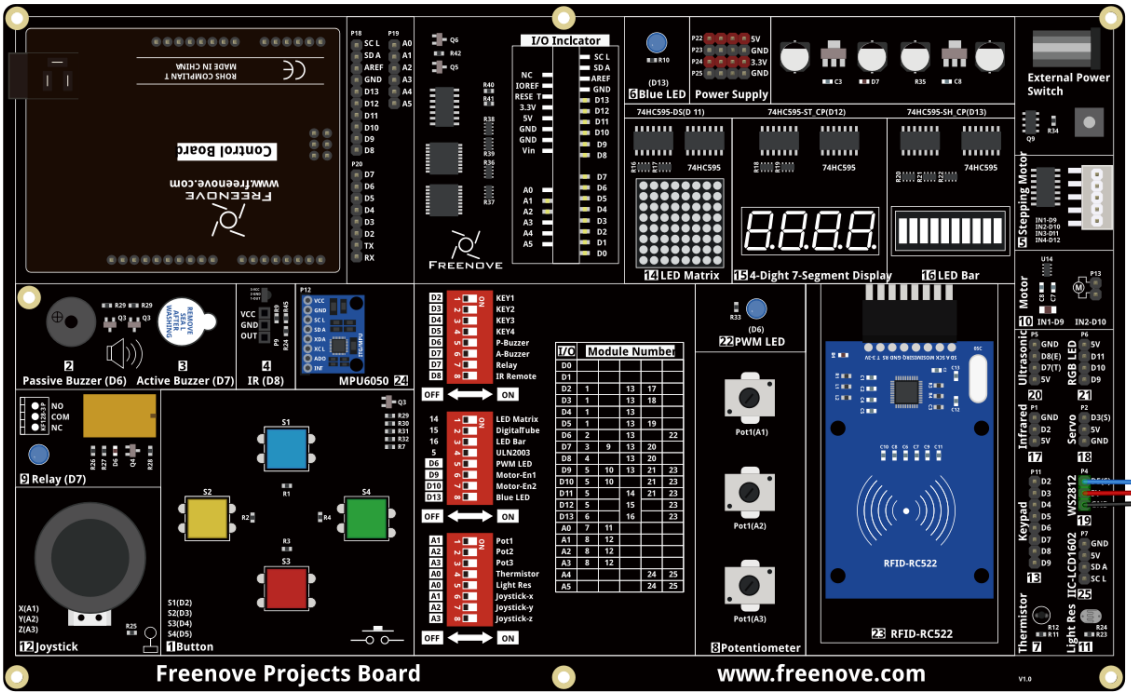
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

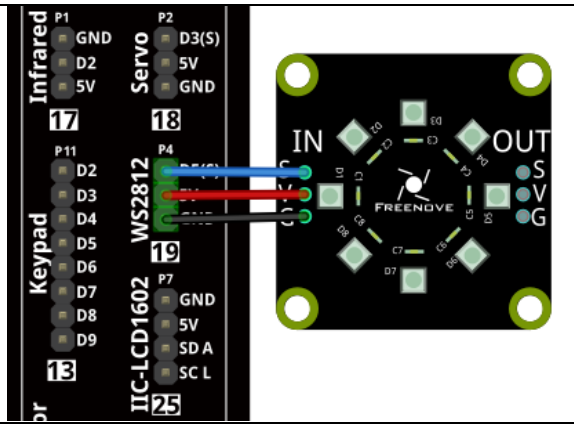
Circuit

Schematic diagram



Hardware connection.





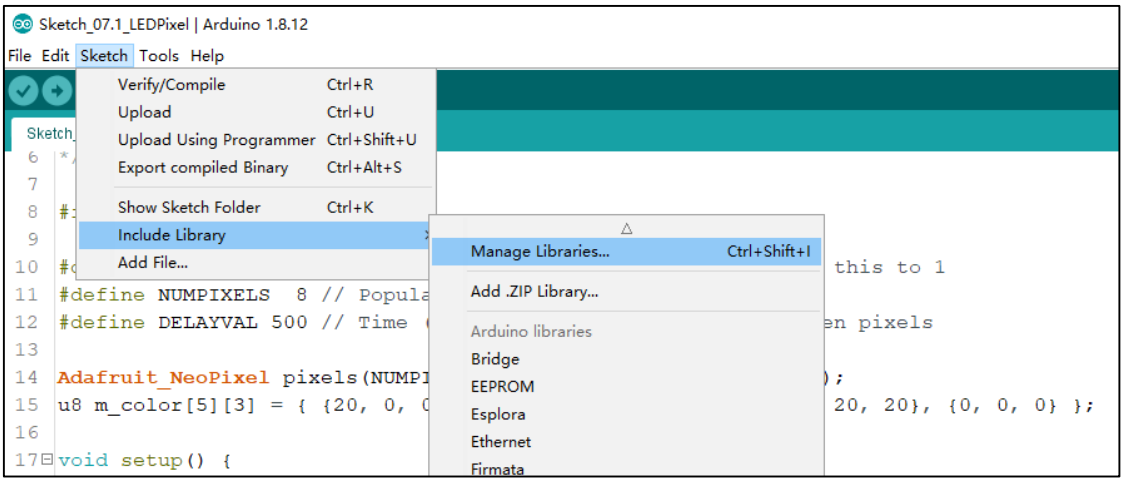
Board	LEDPixel
D5	S
5V	V
GND	G

Sketch

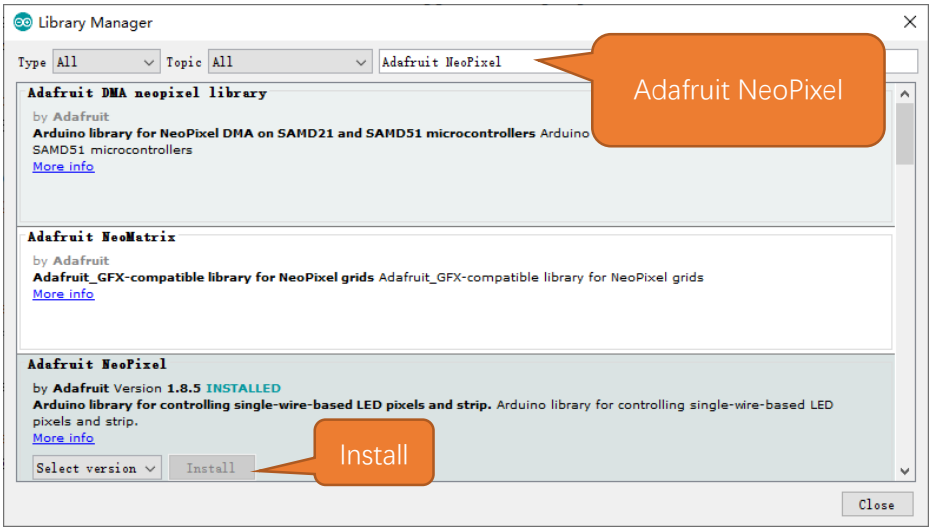
This code uses a library named " Adafruit NeoPixel", if you have not installed it, please do so first.
Library is an important feature of the open source world, and we know that Arduino is an open source platform that everyone can contribute to. Libraries are generally licensed under the LGPL, which means you can use them for free to apply to your creations.

How to install the library

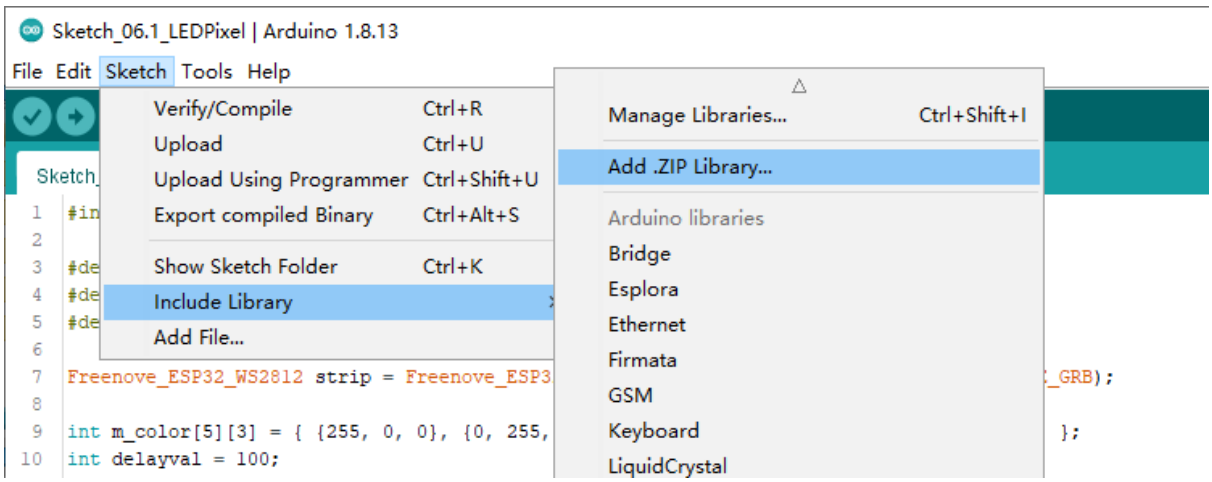
There are two ways to add libraries.
The first way, open the Arduino IDE, click Tools →Manage Libraries.



In the pop-up window, Library Manager, search for the name of the Library, “Adafruit NeoPixel”, and then click Install.



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

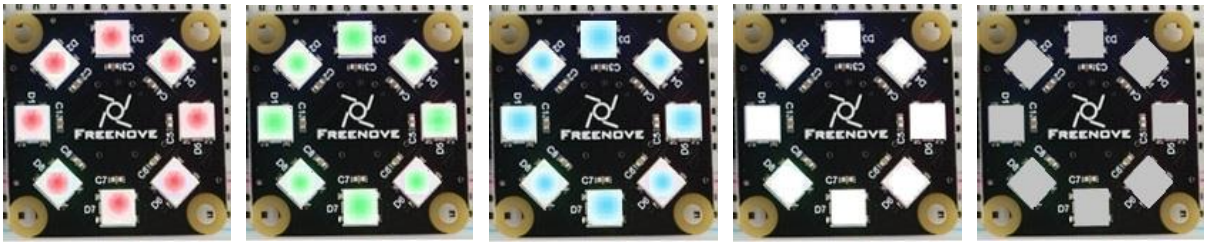


LEDPixel

The following is the program code:

```
#include <Adafruit_NeoPixel.h>

#define PIN        5 // On Trinket or Gemma, suggest changing this to 1
#define NUMPIXELS  8 // Popular LEDPixel ring size
#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
u8 m_color[5][3] = { {20, 0, 0}, {0, 20, 0}, {0, 0, 20}, {20, 20, 20}, {0, 0, 0} };

void setup() {
  pixels.begin(); // INITIALIZE LEDPixel strip object (REQUIRED)
}

void loop() {
  pixels.clear(); // Set all pixel colors to 'off'
  for(int j=0;j<5;j++){
    for(int i=0; i<NUMPIXELS; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(m_color[j][0], m_color[j][1], m_color[j][2]));
    }
    pixels.show();   // Send the updated pixel colors to the hardware.
    delay(DELAYVAL); // Pause before next pass through loop
  }
}
```

Download the code to Control Board, and LEDPixel begins to light up in red, green, blue, white and black.



To use some libraries, first you need to include the library's header file.

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

Define the pins connected to the module and the number of LEDs on it,

```
3  #define PIN        5 // On Trinket or Gemma, suggest changing this to 1
4  #define NUMPIXELS  8 // Popular LEDPixel ring size
```

Define a variable to set the time interval for each LED to light up. The smaller the value, the shorter the time interval between each two LEDs.

```
5  #define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
```

Use the above parameters to create a LEDPixel object strip.

```
7  Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
```

Define the color values to be used, as red, green, blue, white, and black.

```
8  u8 m_color[5][3] = { {20, 0, 0}, {0, 20, 0}, {0, 0, 20}, {20, 20, 20}, {0, 0, 0} };
```

Initialize pixels() in setup().

```
11   pixels.begin(); // INITIALIZE LEDPixel strip object (REQUIRED)
```

In the loop() function, before assigning a new color to LEDPixel, call the clear() function to clear the original color.

```
15   pixels.clear(); // Set all pixel colors to 'off'
```

Call setPixelColor() function to set the color for each LED on LEDPixels. LEDPixel won't emit lights until the show() function is called.

Use for loop to set color data for each LED, and then call show() function to make LEDPixel emit colors.

```
17      for(int i=0; i<NUMPIXELS; i++) { // For each pixel...
18        pixels.setPixelColor(i, pixels.Color(m_color[j][0], m_color[j][1], m_color[j][2]));
19      }
20      pixels.show();   // Send the updated pixel colors to the hardware.
```

Reference

```
Adafruit_NeoPixel(uint16_t n, uint16_t pin=6,neoPixelType type=NEO_GRB + NEO_KHZ800);
```

Every time you use LEDPixel, you need to create an LEDPixel object first.
n: the number of LEDs.
pin: The pin number of the Control Board to drive the LEDPixel data in.
type: The setting of the LED's built-in driver protocol type and driving speed.
Driver Protocol:
NEO_RGB: The order of loading colors for the LEDPixel module is red, green, and blue.
NEO_RBG: The order of loading colors for the LEDPixel module is red, blue, and green.
NEO_GRB: The order of loading colors for the LEDPixel module is green, red, and blue.
NEO_GBR: The order of loading colors for the LEDPixel module is green, blue, and red.
NEO_BRG: The order of loading colors for the LEDPixel module is blue, red, and green.
NEO_BGR: The order of loading colors for the LEDPixel module is blue, green, and red.

```
void begin(void);
```

Initialize the LEDPixel object.

```
void setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b);
void setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b, uint8_t w);
void setPixelColor(uint16_t n, uint32_t c);
```

Set the color of LED with sequence number n.

```
void show(void);
```

Send the color data to the LED and display the color immediately.

```
void setBrightness(uint8_t);
```

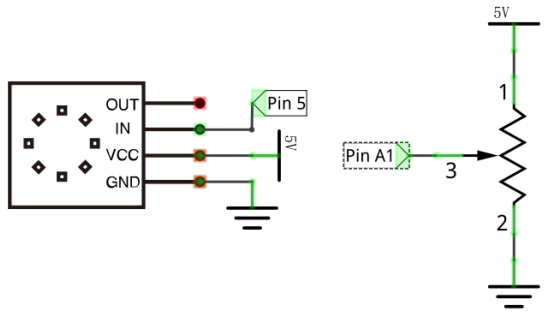
Set the brightness of the LED.

Project 7.2 Rainbow Light

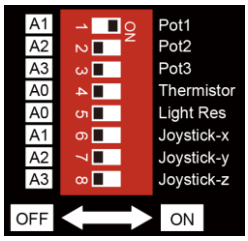
In the previous project, we have mastered the use of LEDPixel. This project will make a slightly complicated rainbow light. The component list and the circuit are exactly the same as the project LEDPixel.

Circuit

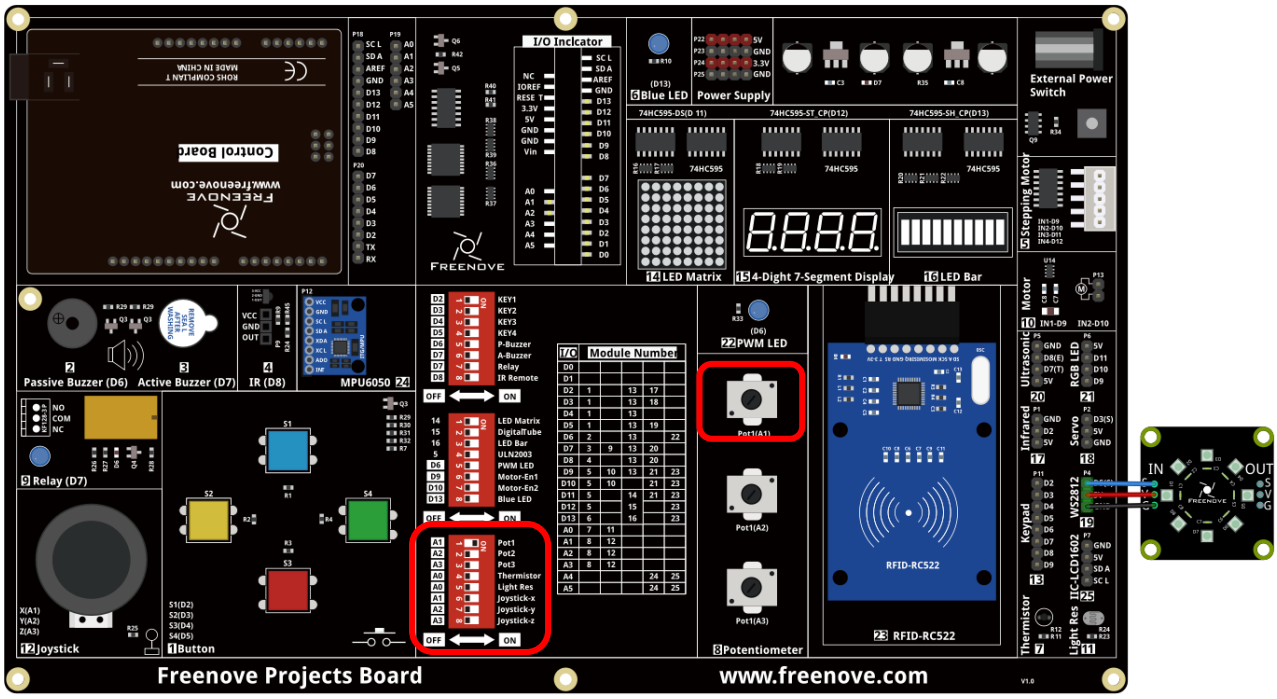
Schematic diagram

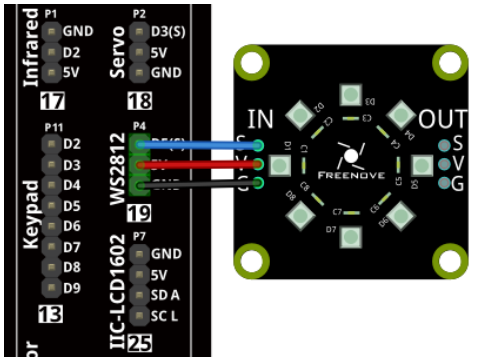


Hardware connection.



Hardware connection.





Board	LEDPixel
D5	S
5V	V
GND	G

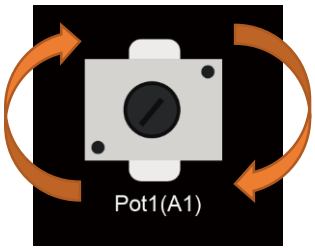
Sketch

Continue to use the following color model to equalize the color distribution of the 8 LEDs and gradually change.

RainbowLight

```
#include <Adafruit_NeoPixel.h>

#define PIN 5
#define NUMBER 8
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMBER, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
  strip.setBrightness(50);
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  int analogValue = analogRead(A1);
  int colorPos = map(analogValue, 0, 1023, 0, 255);
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i,Wheel(colorPos + i * 255 / NUMBER));
  }
  strip.show();
  delay(10);
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if (WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

Upload code to the control board. Rotate the potentiometer and the color of LEDPixel will change accordingly.



With the color model, write the Wheel() function to convert the color model into the corresponding color data.

```
25  uint32_t Wheel(byte WheelPos) {
26    WheelPos = 255 - WheelPos;
27    if (WheelPos < 85) {
28      return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
29    }
30    if (WheelPos < 170) {
31      WheelPos -= 85;
32      return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
33    }
34    WheelPos -= 170;
35    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
36  }
```

The range of the color model is 0-255. Get the ADC value of the potentiometer and map it to the color model range, to get the offset value in the color model corresponding to the rotation of the potentiometer.

```
14    int analogValue = analogRead(A1);
15    int colorPos = map(analogValue, 0, 1023, 0, 255);
```

Take NUMBER points evenly from the color model, and add the offset value of the potentiomter to each point to get the color point of each LED in actual display. Call the Wheel function to convert the color points into the corresponding color data and set the Neopixel..

```
16    for (int i = 0; i < strip.numPixels(); i++) {
17      strip.setPixelColor(i,Wheel(colorPos + i * 255 / NUMBER));
18    }
```

Chapter 8 Buzzer

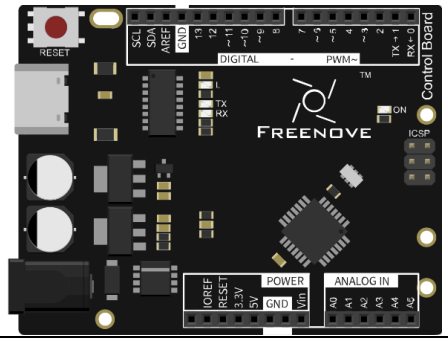
Earlier, we have used control board and basic electronic components to carry out a series of interesting projects. Now, let's learn how to use some integrated electronic components and modules. These modules are usually integrated with a number of electronic components, so they have special features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

Project 8.1 Active Buzzer

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

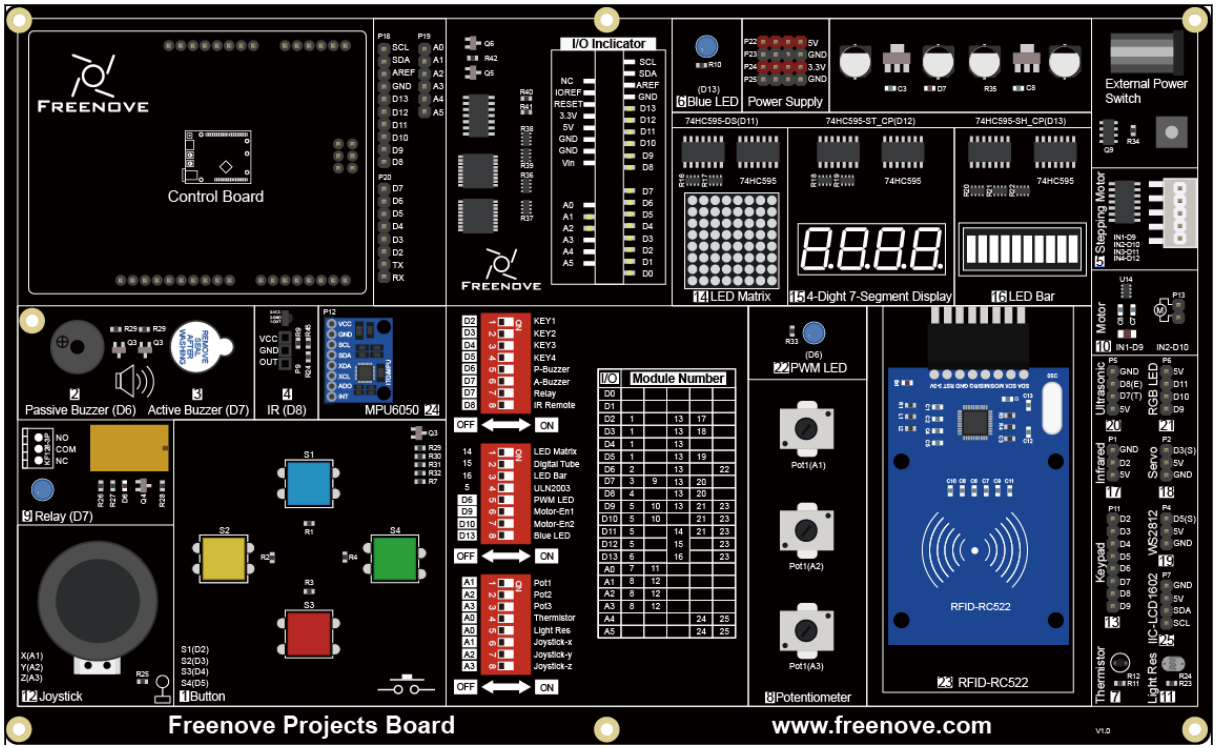
Component List

Control board x1



USB cable x1

Freenove Projects Board

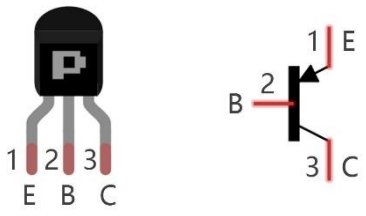


Component knowledge

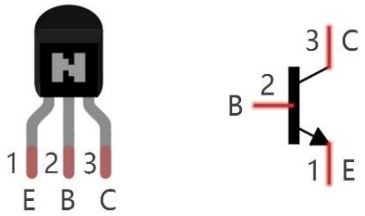
Transistor

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



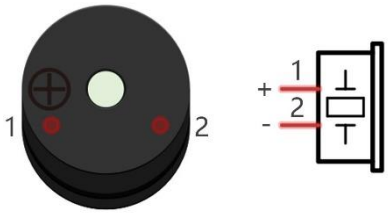
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

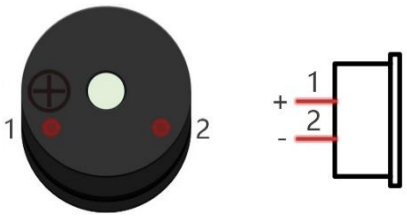
Buzzer

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?

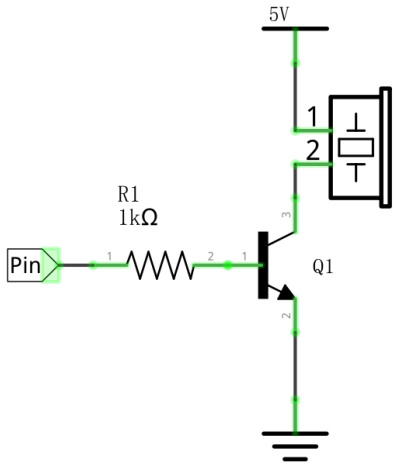
1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this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

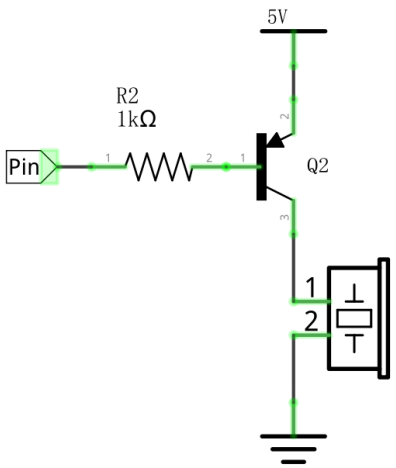
Passive buzzer bottom

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).



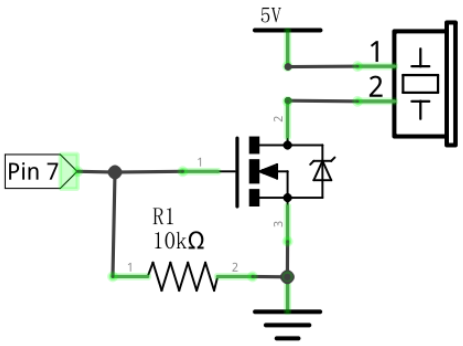
When we use a PNP transistor to drive a buzzer, we often use the following method. If control board pin outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If control board pin outputs high level, no current flows through R1, the transistor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



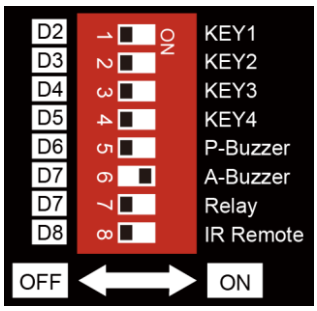
Circuit

Drive the active buzzer with pin7.

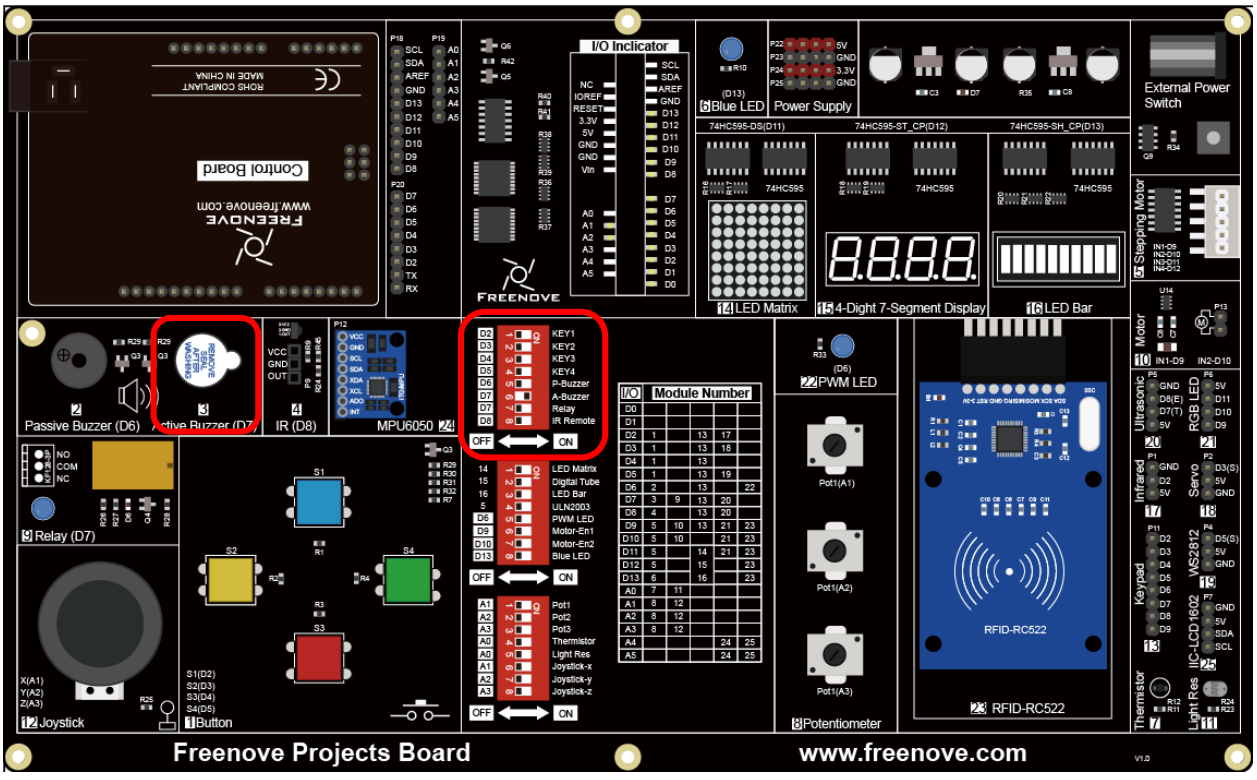
Schematic diagram



Hardware connection



Hardware connection



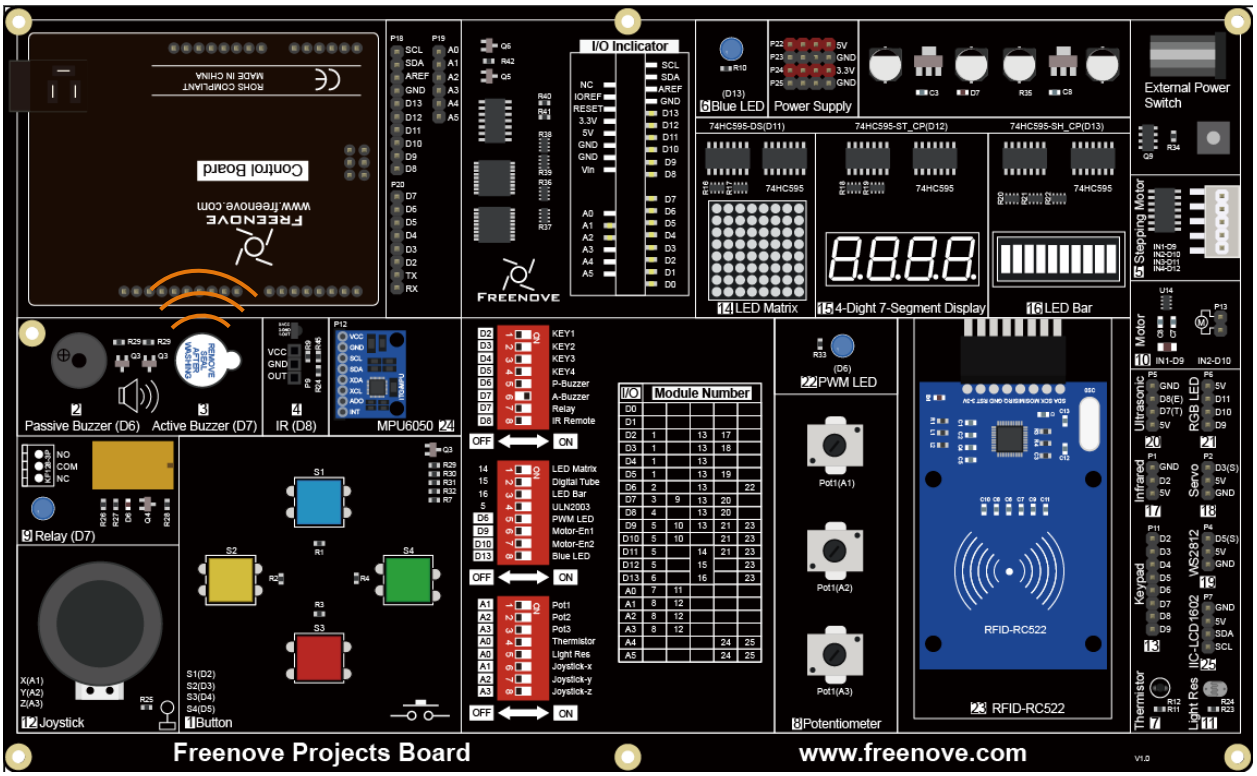
Sketch

Active_Buzzer

The usage of active buzzer is similar to that of the LED. The pin on Freenove Projects Board outputs high or low level to control the active buzzer's ON or OFF.

```
int buzzerPin = 7;    // the number of the buzzer pin

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

void loop() {
    digitalWrite(buzzerPin, LOW);     // Turn off Buzzer
    delay(500);
    digitalWrite(buzzerPin, HIGH);    // Turn on Buzzer
    delay(500);
}
```

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

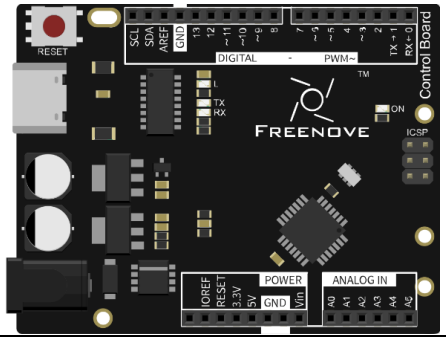


Project 8.2 Passive Buzzer

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.

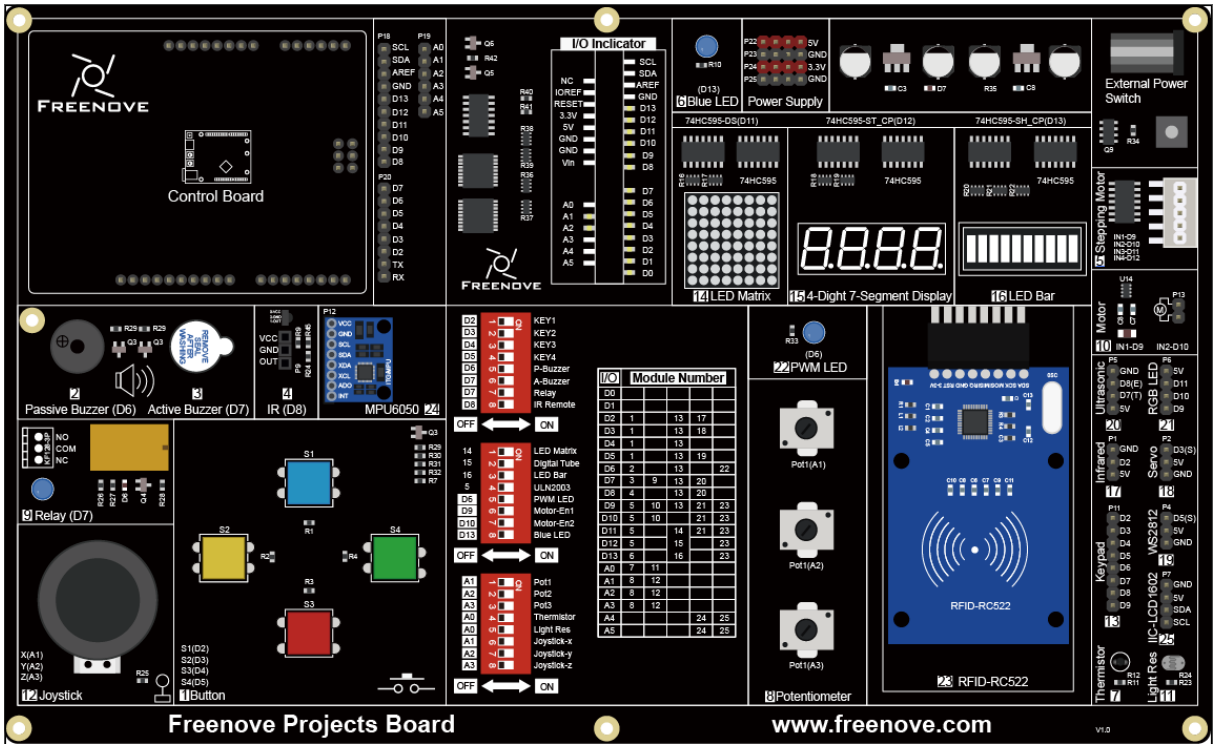
Component List

Control board x1



USB cable x1

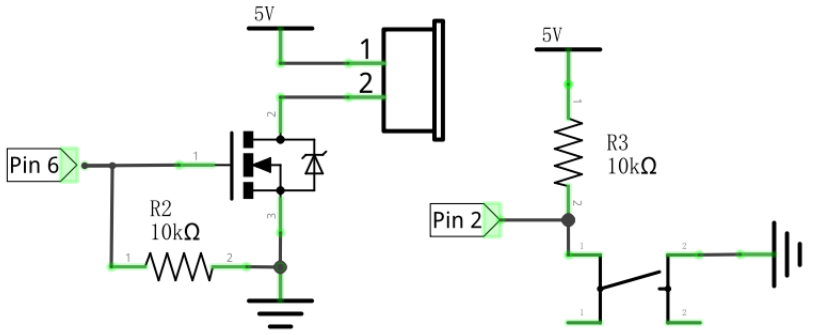
Freenove Projects Board



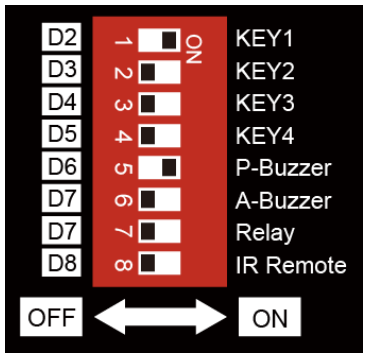
Circuit

Use pin 6 port of control board to drive a passive buzzer.

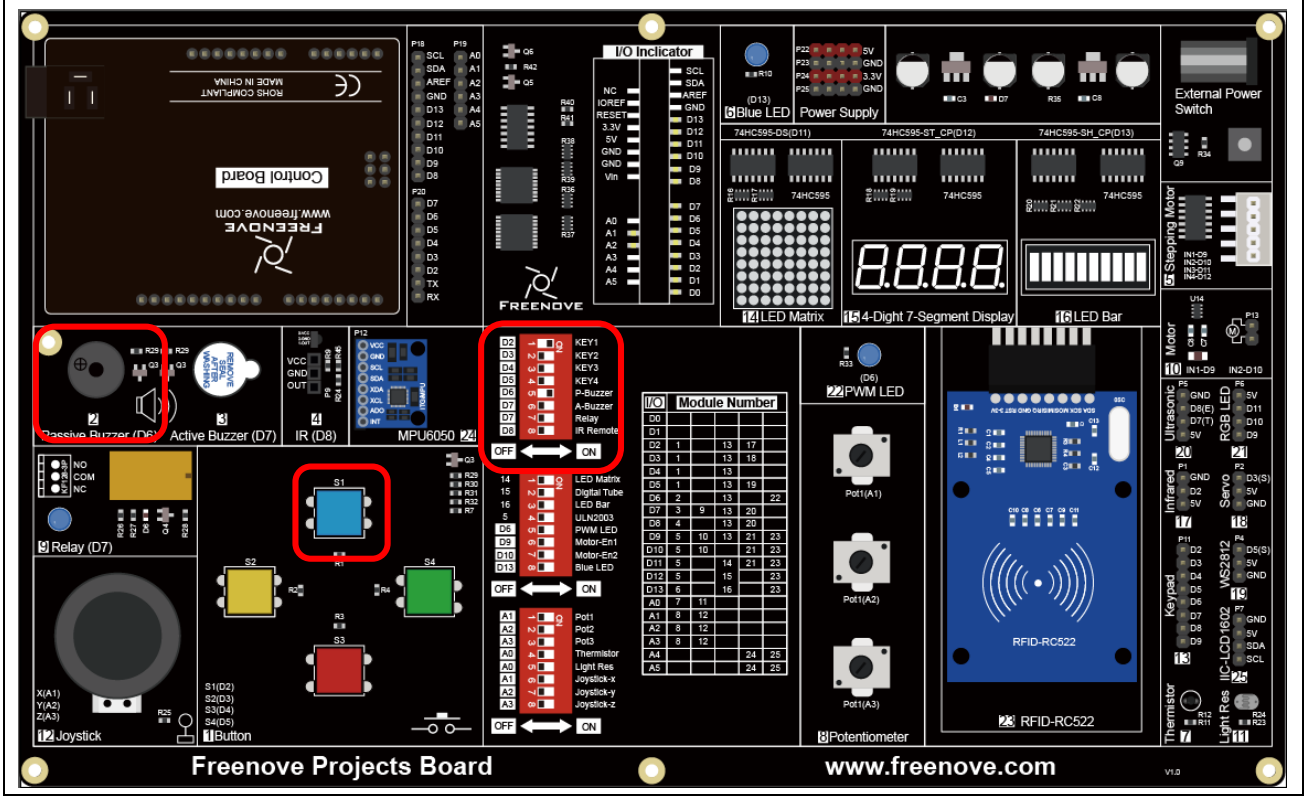
Schematic diagram



Hardware connection



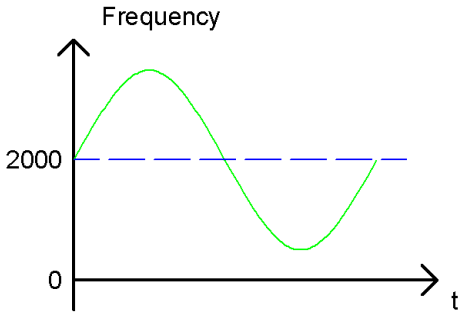
Hardware connection



Sketch

Passive_Buzzer

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:



When the key is pressed, the passive buzzer will make sounds in different frequence; When leased, the buzzer stops sounding.

```
int buzzerPin = 6;     // the number of the buzzer pin
int keyPin = 2;        // the number of the key1 pin
float sinVal;          // Define a variable to save sine value
int toneVal;           // Define a variable to save sound frequency

void setup() {
  pinMode(buzzerPin, OUTPUT); // Set Buzzer pin to output mode
  pinMode(keyPin, INPUT);     // Set Key1 pin to input mode
}

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

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

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

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

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

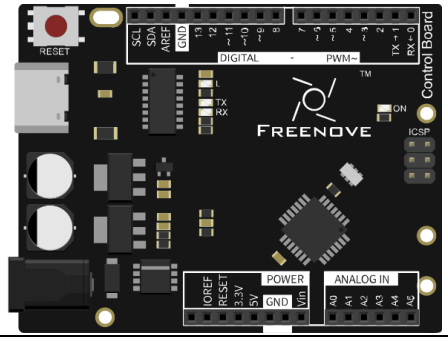
Chapter 9 Relay

In this chapter, we will learn how to control a relay.

Project 9.1 Control Relay

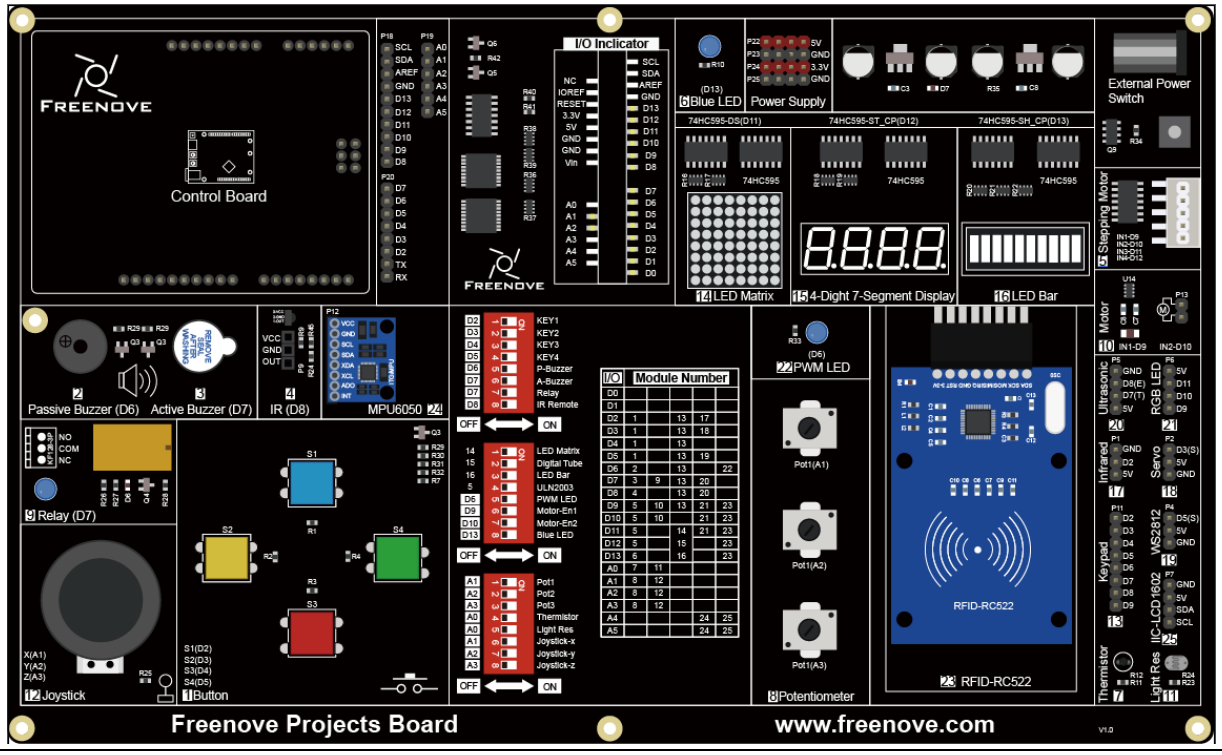
Component List

Control board x1



USB cable x1

Freenove Projects Board



Component Knowledge

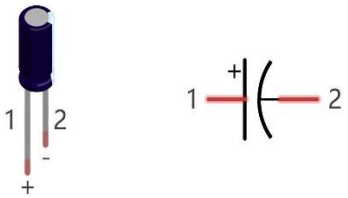
Capacitor

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

There are 2 kinds of capacitors: non-polar capacitors and polar capacitors. Generally, a 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:

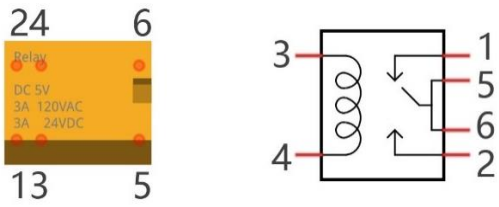


When the motor rotates, it will generate noise. As the contact of coil connects and disconnects the electrode constantly, it will cause the supply voltage unstable. Thus, a small capacitor is often connected to motor to reduce the impact on power supply from motor.

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:



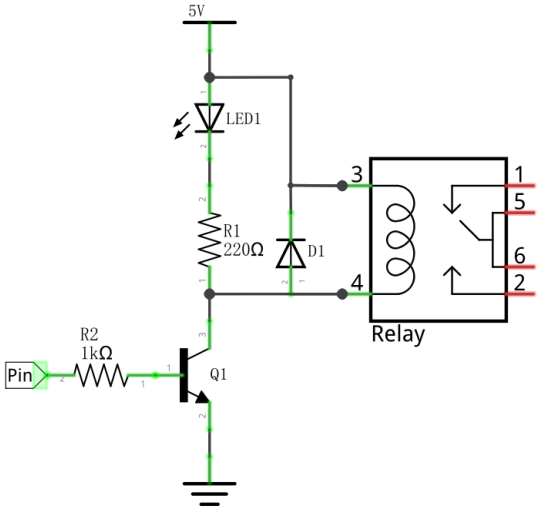
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

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.

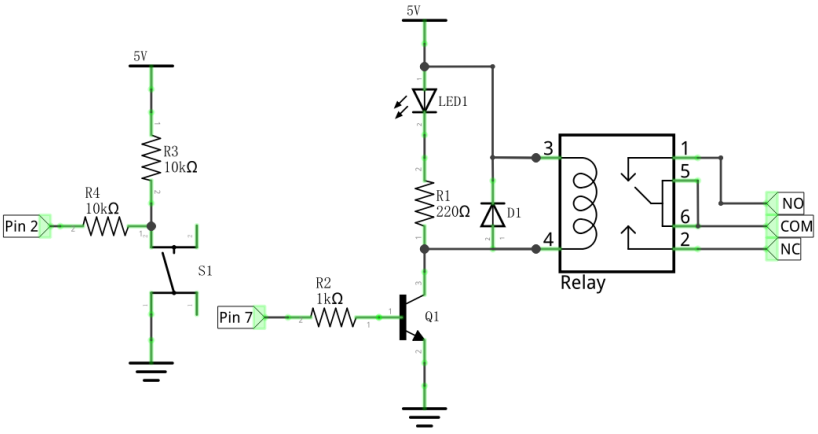
The circuit for a relay is as follows: The coil of relay can be equivalent to an Inductor, when a transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay’s coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes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



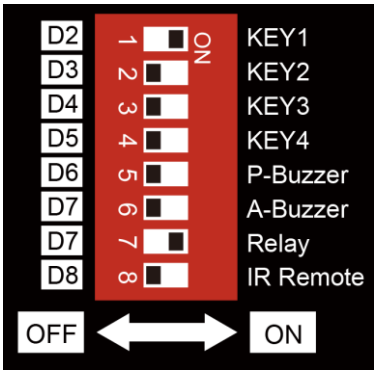
Circuit

Use pin 2 of control board to detect the state of push button switch, and pin 7 to control the relay. As the running of motor needs larger power, we will use two AA batteries to supply power for the motor alone.

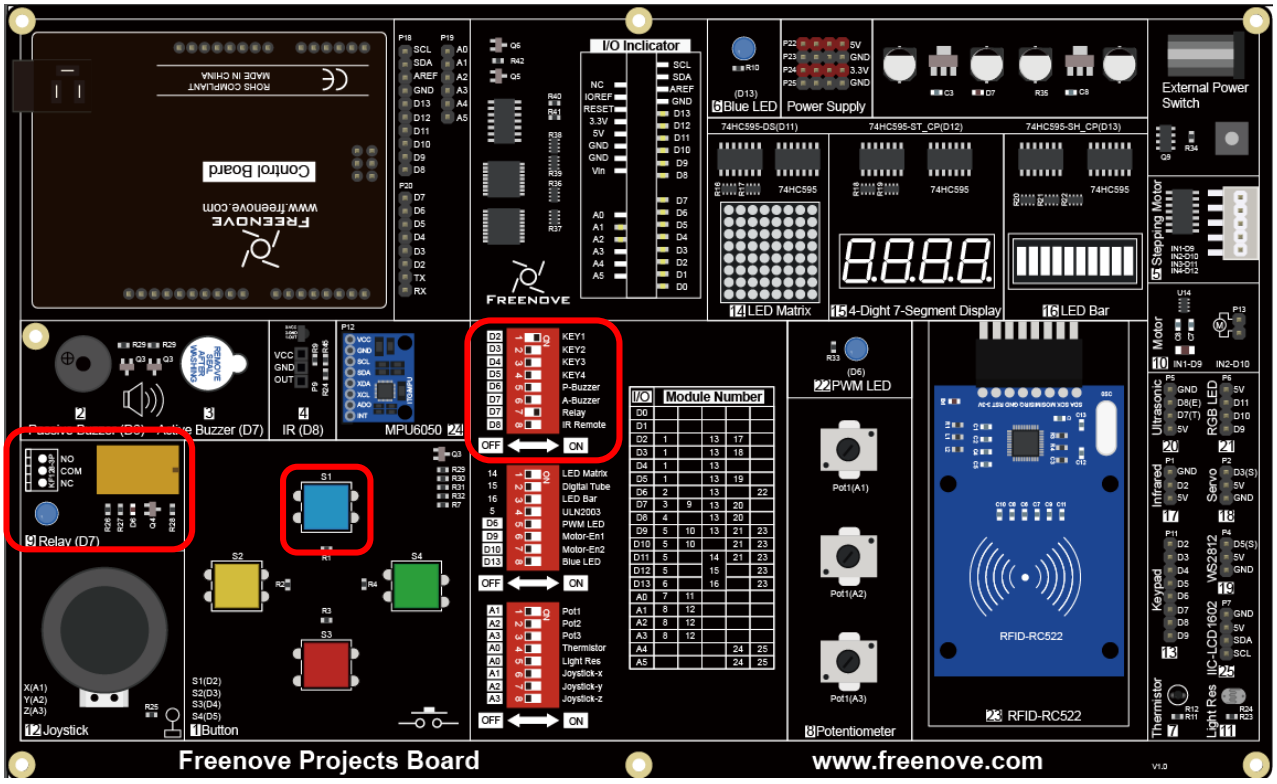
Schematic diagram



Hardware connection



Hardware connection



Sketch

Control_Relay

Now, write code to detect the state of push button switch. Each time you press the button, the switching status of relay will change.

```
int relayPin = 7;      // the number of the relay pin
int buttonPin = 2;     // the number of the push button pin

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

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

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

In this code, we used a new method to detect 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.

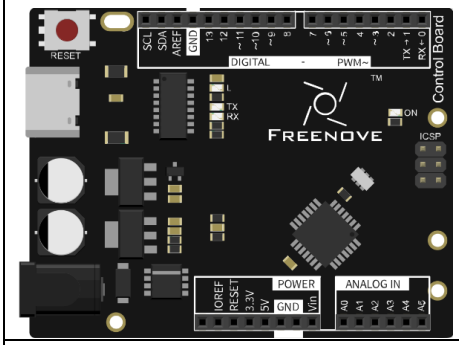
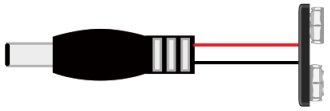
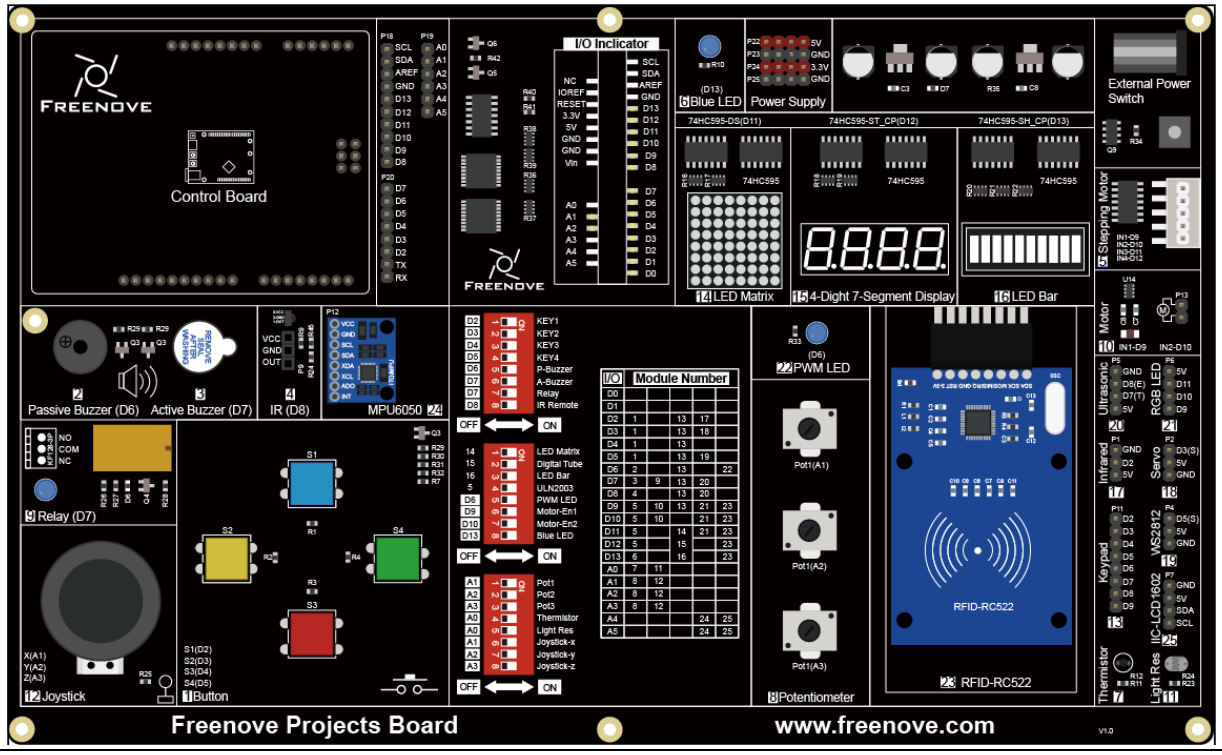
Verify and upload the code, every time you press the push button, the state of relay changes once.

Chapter 10 Motor

Project 10.1 Control Motor

Now, we will use dedicated chip DRV8837 to control the motor.

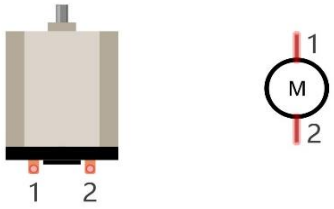
Component List

Control board x1 	USB cable x1 	Motor x1 
Jumper Wire x2 	9V Battery Connector (Batteries NOT Included) x1 	
Freenove Projects Board 		

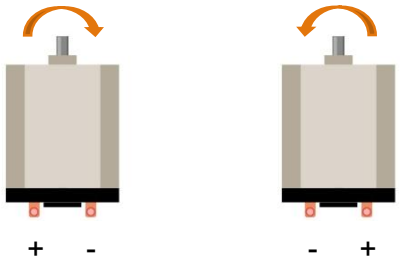
Component Knowledge

DC Motor

A 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.

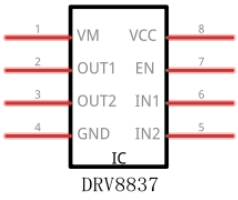


When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.



DRV8837

DRV8837 is a 2-channel motor driven IC chip. You can use it to drive a unidirectional DC motor with 2 ports or a bidirectional DC motor with 1 port.

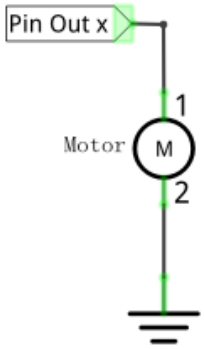


Port description of DRV8837 module is as follows:

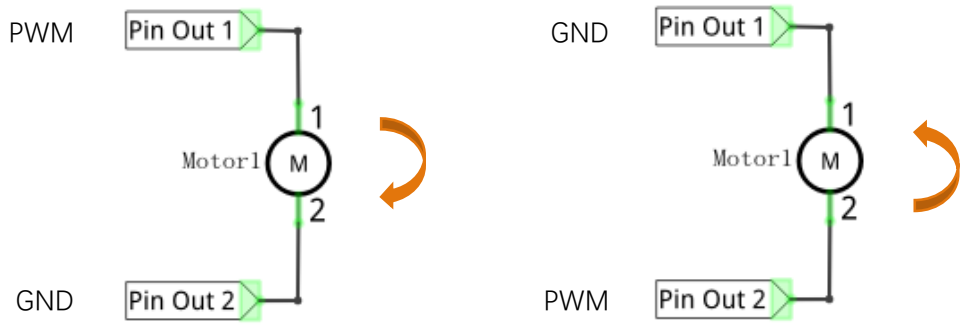
Pin name	Pin number	Description
VM	1	Power supply for motor
Out x	2,3	Output control chip: Determine the output state according to the In x input signal
GND	4	Negative pole
IN x	5,6	Input control signal
EN	5	Chip enable signal, valid at high level
VCC	6	3.3V or 5V logic power supply

For more details, please see datasheet.

When using DRV8837 to drive DC motor, there are usually two kinds of connection.
The following connection option uses one channel of the DRV8837, which can control motor speed through the PWM, However the motor then can only rotate in one direction.



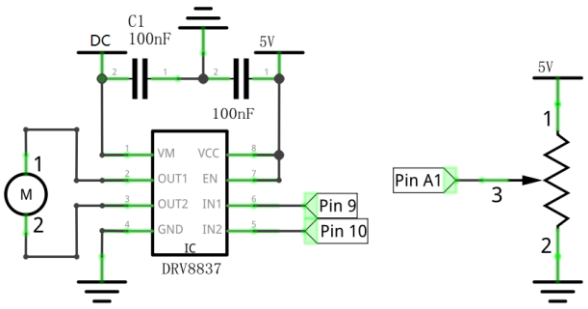
The following connection uses two channels of the DRV8837: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, not only controls the speed of motor, but also can control the direction of the motor.



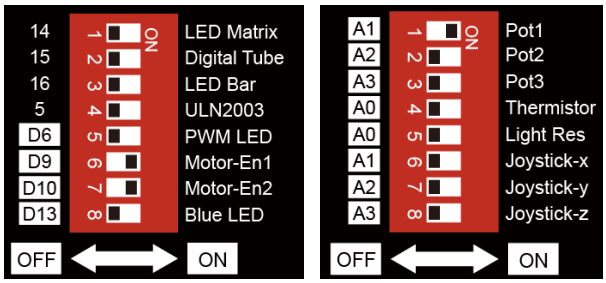
Circuit

Use pin A1 of the control board to detect the voltage of rotary potentiometer; pin 9 and pin 10 to control the motor's rotation direction and speed.

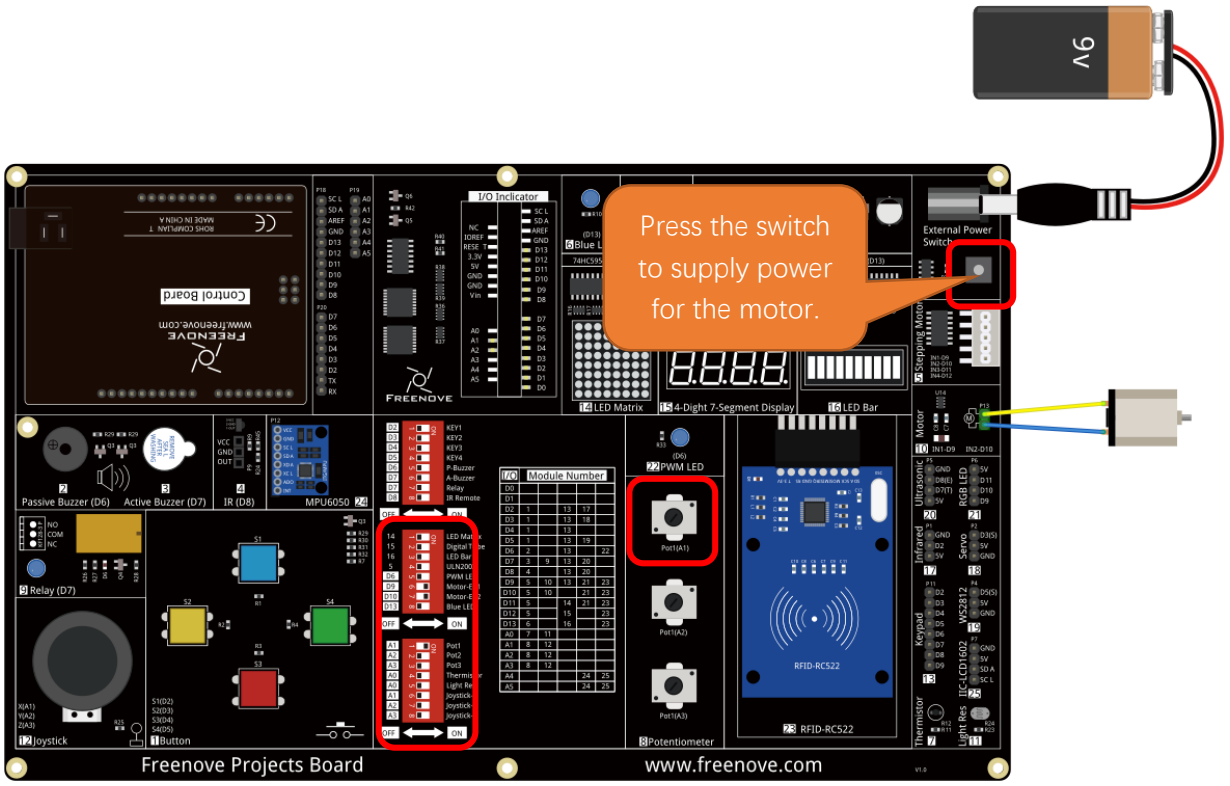
Schematic diagram



Hardware connection



Hardware connection



Sketch

Control_Motor

Now, write the code to control speed and rotation direction of motor through rotary potentiometer. When the potentiometer stays in the middle position, the motor speed is the minimum; When it deviates from the middle 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 = 9;        // Define Motor pin1
2  int IN2Pin = 10;       // Define Motor pin2
3
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 }
12
13 void loop() {
14   int potenVal = analogRead(A1);  // Convert the voltage of rotary potentiometer into digital
15
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   delay(10);
27 }
28
29 void driveMotor(boolean dir, int spd) {
30   // Control motor rotation direction
31   if (dir){
32     analogWrite(IN1Pin,spd);
33     analogWrite(IN2Pin,0);
```

```
34    }
35    else{
36      analogWrite(IN1Pin,0);
37      analogWrite(IN2Pin,spd);
38    }
39  }
```

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

```
29  void driveMotor(boolean dir, int spd) {
30    // Control motor rotation direction
31    if (dir){
32      analogWrite(IN1Pin,spd);
33      analogWrite(IN2Pin,0);
34    }
35    else{
36      analogWrite(IN1Pin,0);
37      analogWrite(IN2Pin,spd);
38    }
39  }
```

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

```
13  void loop() {
14    int potenVal = analogRead(A1);  // Convert the voltage of rotary potentiometer into digital
15
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    delay(10);
27  }
```

abs(x)
Calculates the absolute value of a number.

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

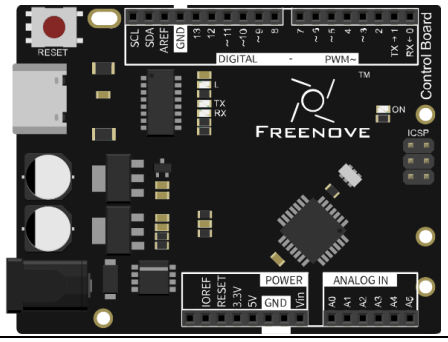
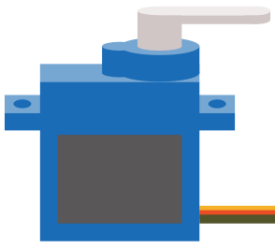
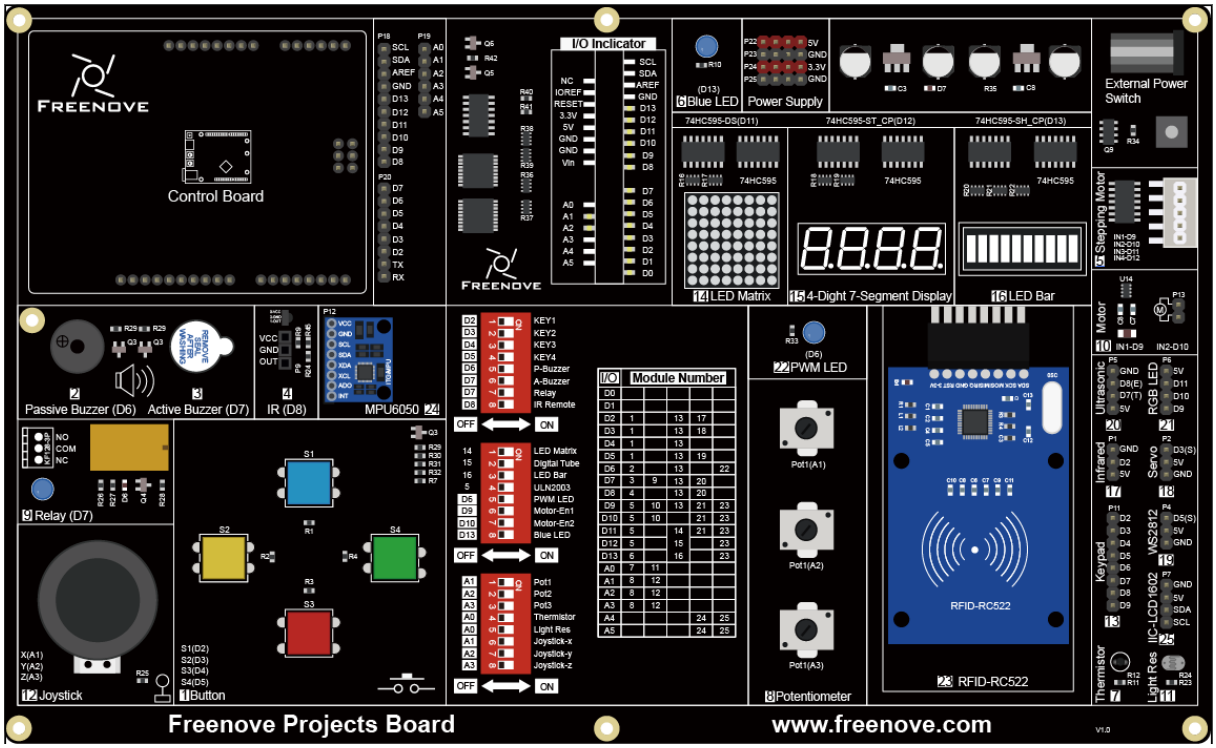
Chapter 11 Servo

Earlier, we have used control board and L293D module to control the motor speed and direction. 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.

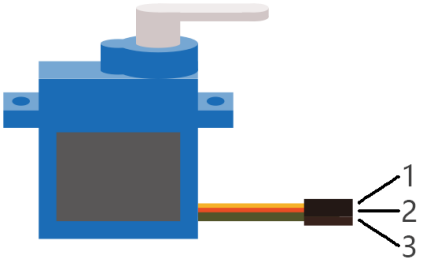
Component List

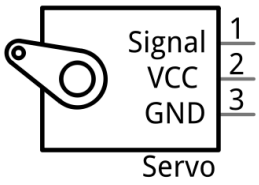
Control board x1 	USB cable x1 	Servo x1 
Freenove Projects Board 		

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 degrees 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

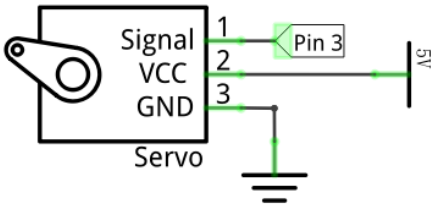
When you change the servo signal, the servo will rotate to the designated position.

Circuit

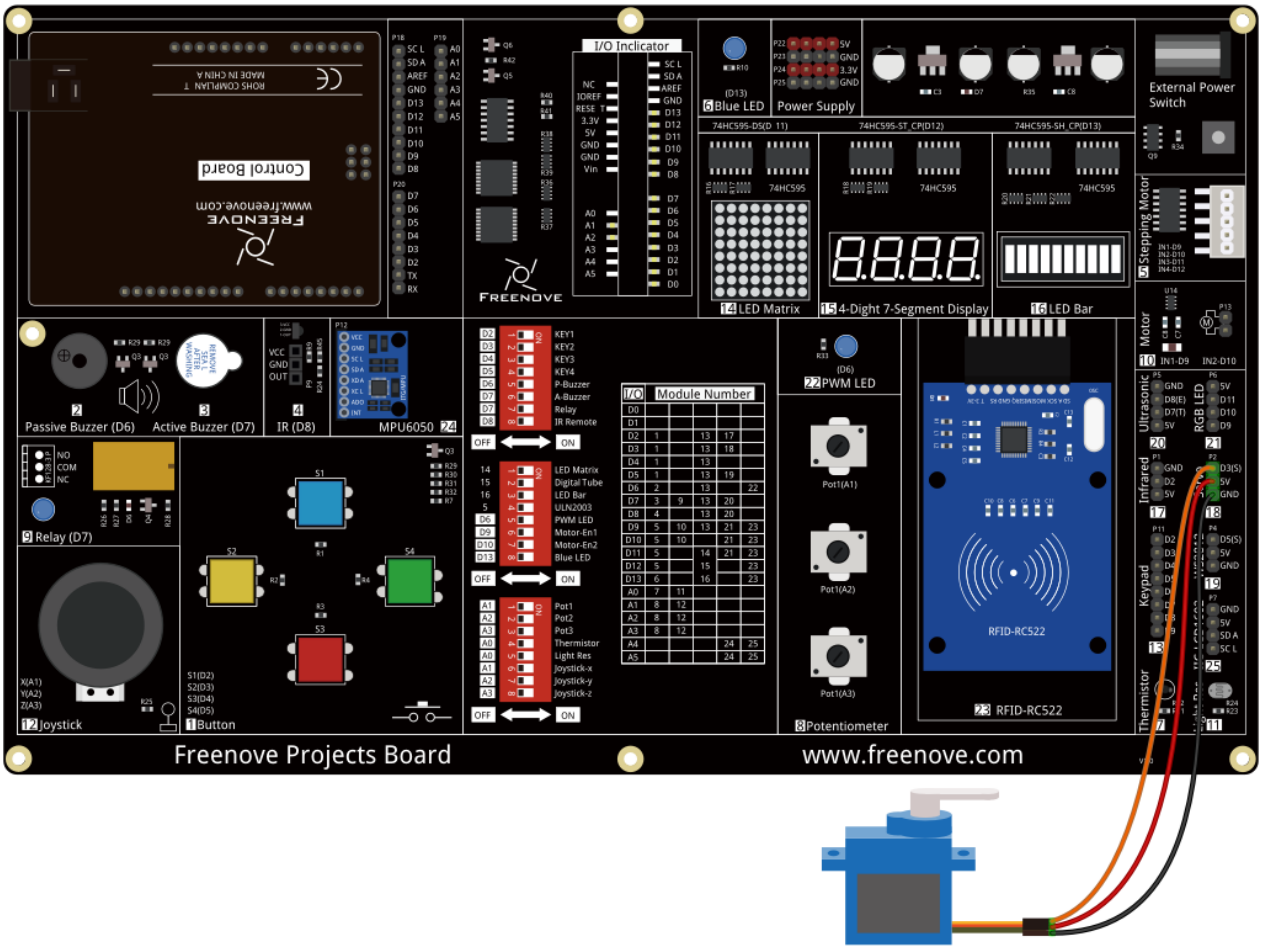
Use pin 3 of the control board to drive the servo.

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

Servo_Sweep

Now, write the code to control servo, making it sweep in the movement 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
}

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

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

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

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

```
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, an object of the Servo type must be defined, 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 defining the Servo object, you can reference functions, such as initializing the servo:

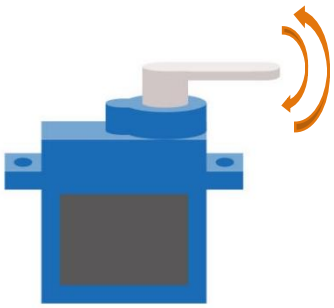
```
9    myservo.attach(servoPin); // attaches the servo on servoPin to the servo object
```

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.

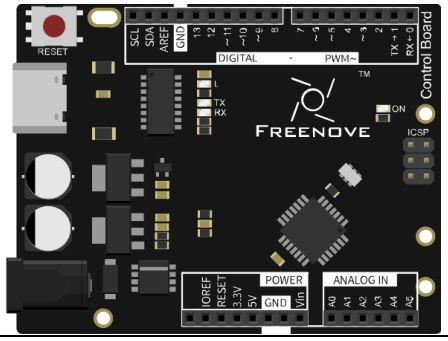
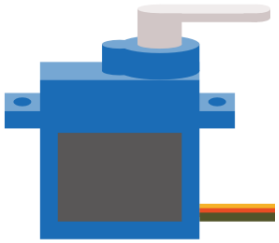
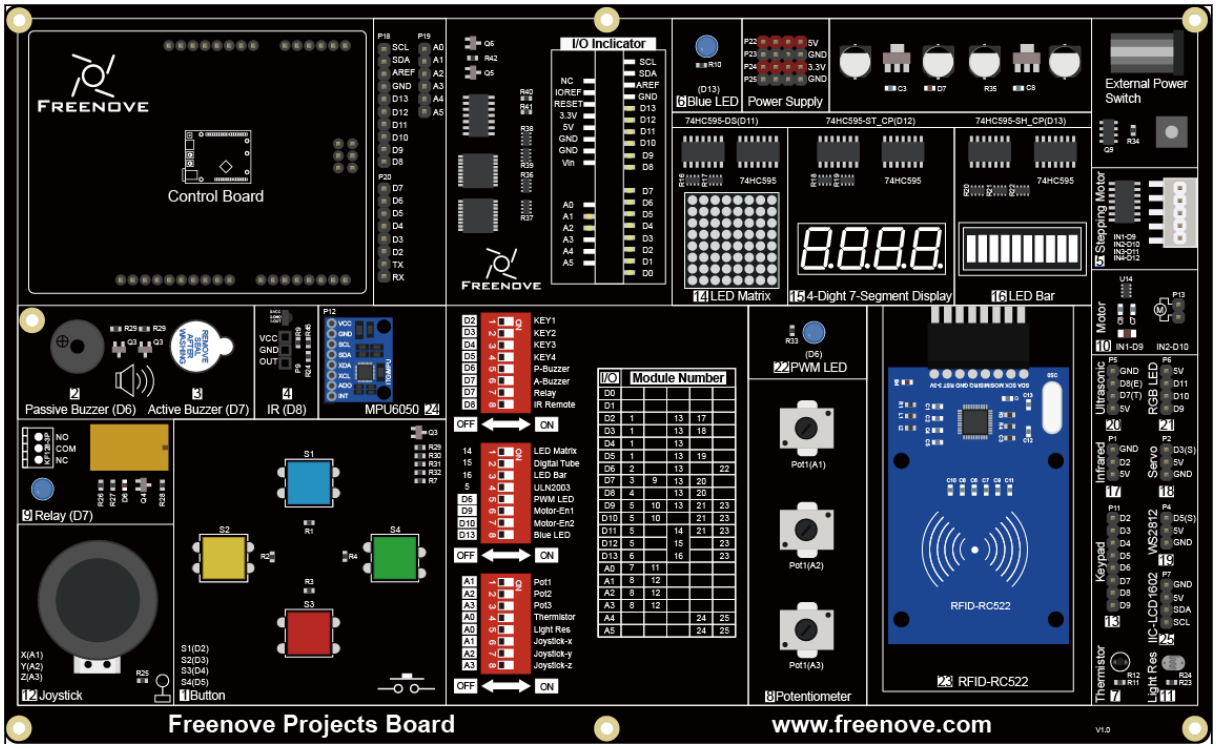
Verify and upload the code, the servo starts to sweep continuously.



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.

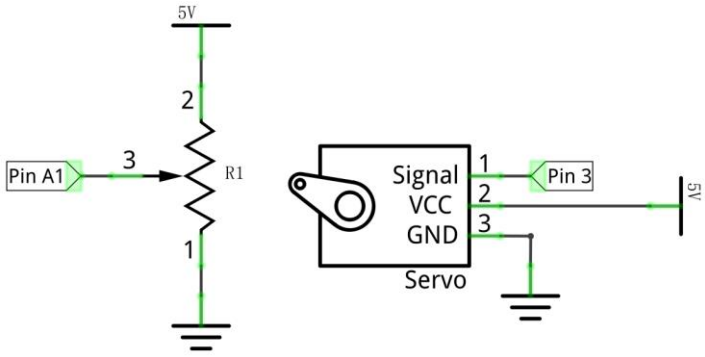
Component List

Control board x1 	USB cable x1 	Servo x1 
Freenove Projects Board 		

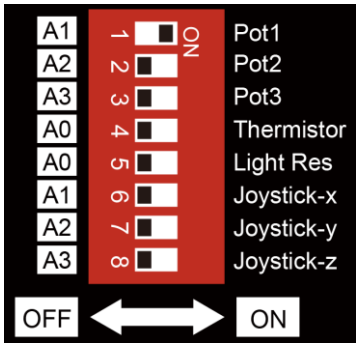
Circuit

Use pin A1 of the control board to detect the voltage of rotary potentiometer, and pin 3 to drive the servo.

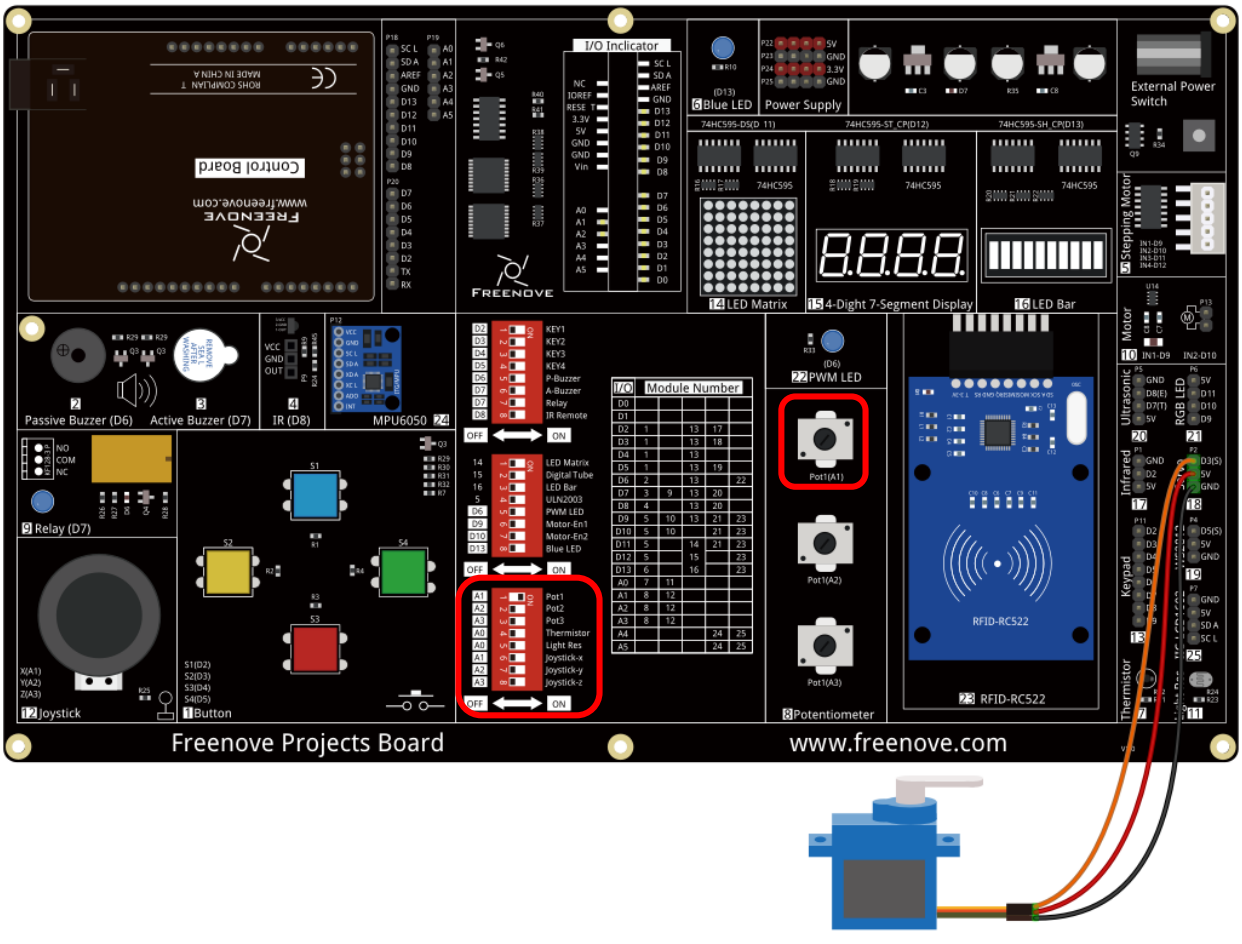
Schematic diagram



Hardware connection



Hardware connection



Sketch

Control_Servo_by_Potentiometer

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

```
#include <Servo.h>

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

int servoPin = 3;           // define the pin of servo signal line
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
  Serial.begin(115200);
}

void loop() {
  potVal = analogRead(A1);              // reads the potValue of the potentiometer
  Serial.println(potVal);
  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 A1, 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

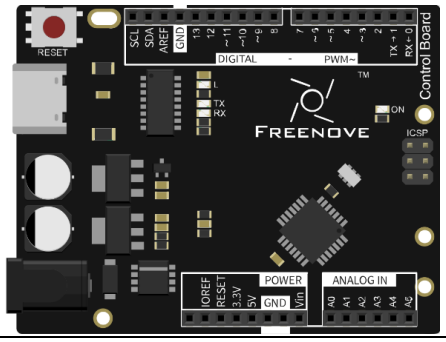
Earlier, we have used control board and photoresistor to detect the intensity of light. Now, we will learn to use the temperature sensor.

Project 12.1 Detect the Temperature

We will use a thermistor to detect the ambient temperature.

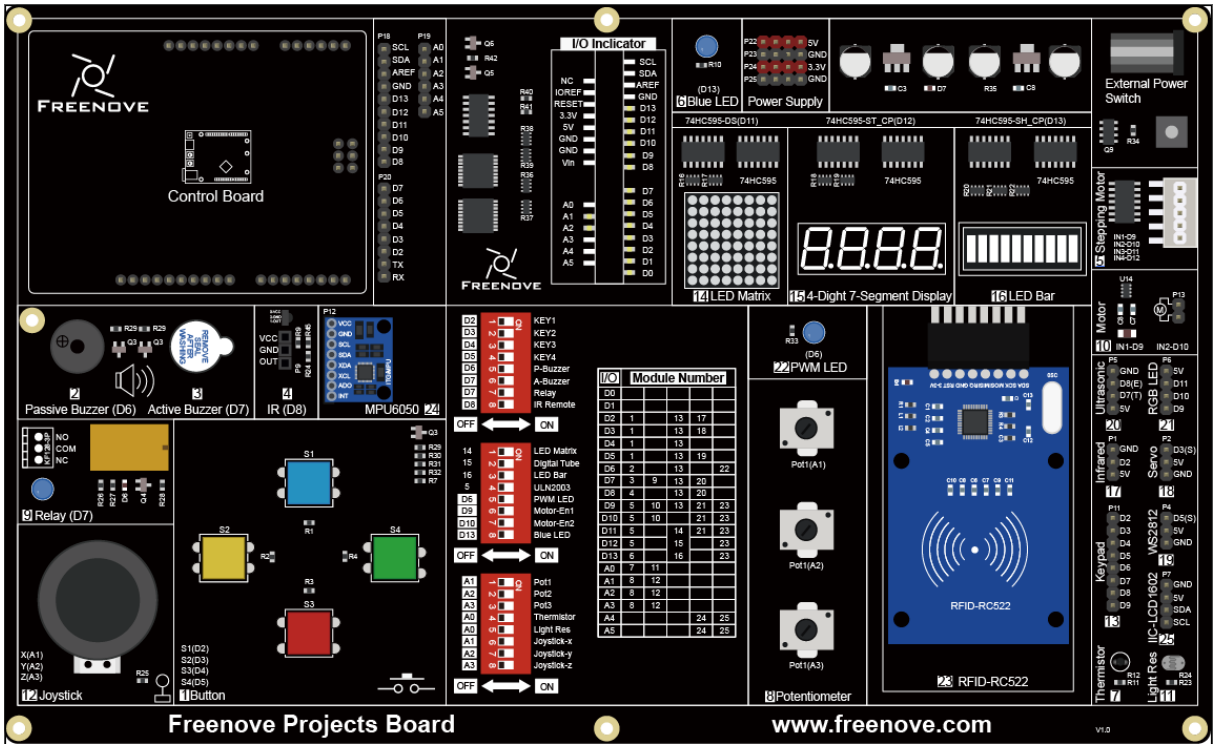
Component List

Control board x1



USB cable x1

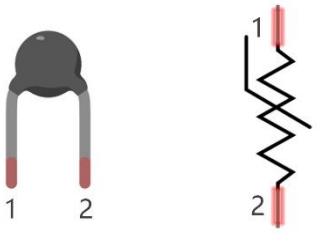
Freenove Projects Board



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;

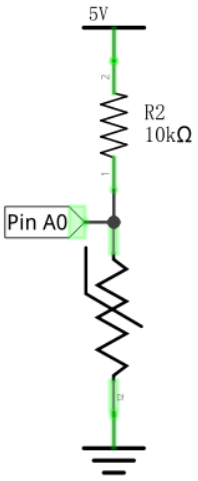
EXP[n] is nth power of e;

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:

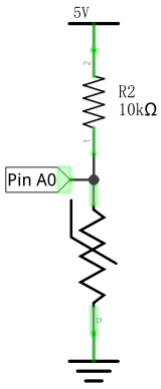


We can use the value measured by the analog pin of control board to obtain resistance value of the thermistor, and then we can use the formula to obtain the temperature value.

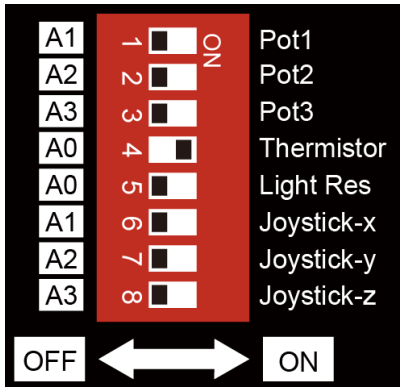
Circuit

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

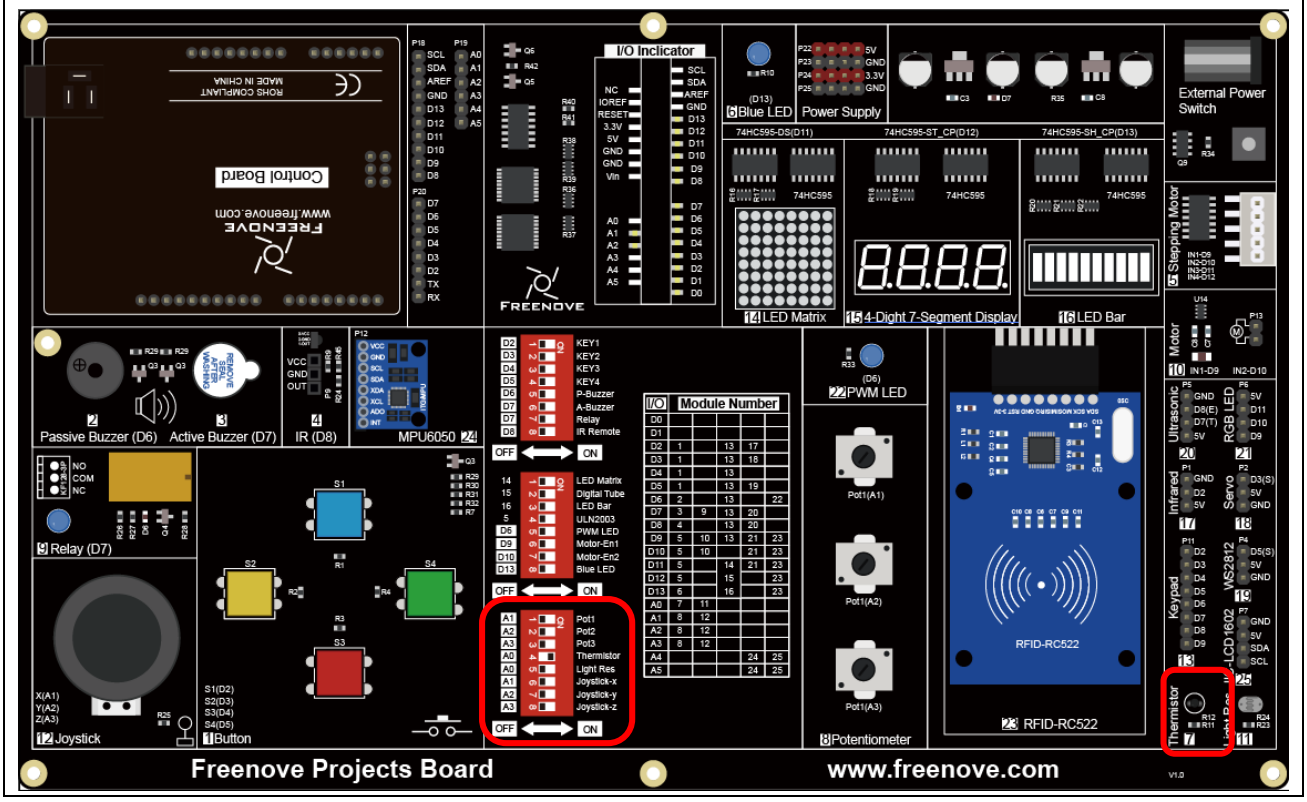
Schematic diagram



Hardware connection



Hardware connection



Sketch

Detect_the_temperature

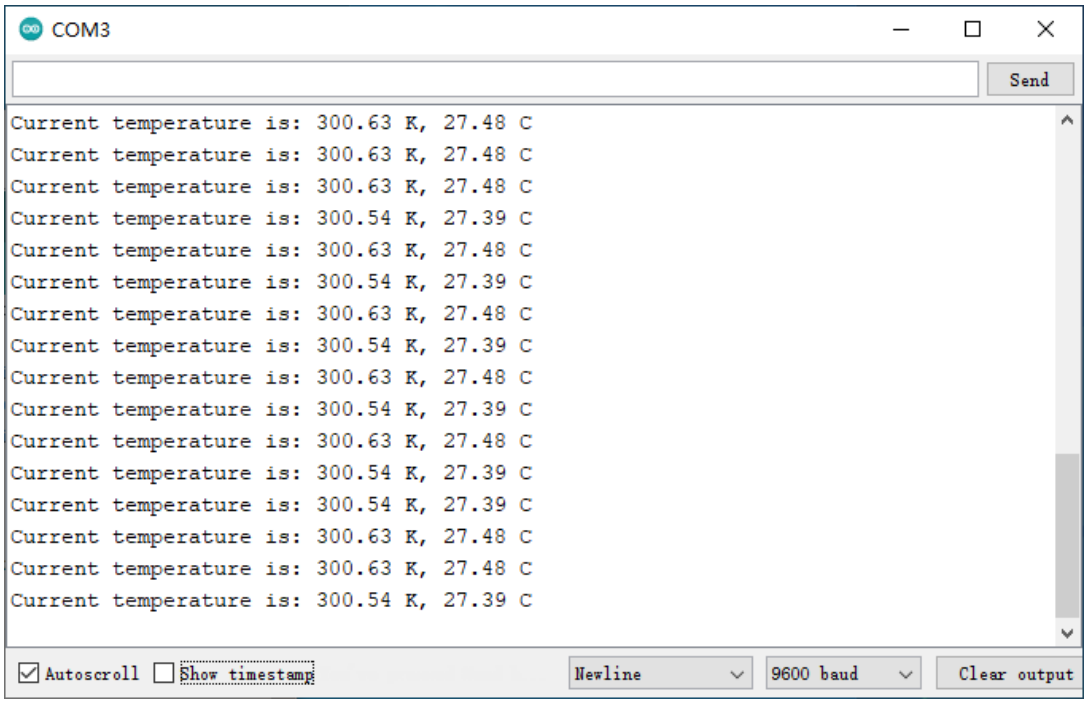
Now, write the code to detect the voltage value of thermistor, calculate the temperature value, and send it to Serial Monitor.

```
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

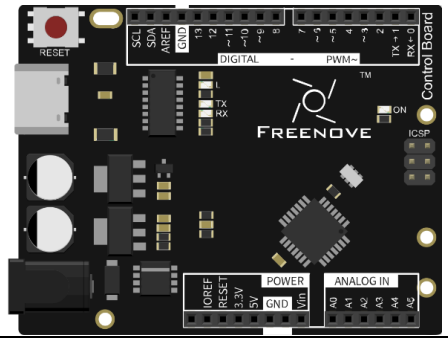
In the previous chapter, we have learned how to use rotary potentiometer. Now, let us learn a new electronic module Joystick that works on the same principle as rotary potentiometer.

Project 13.1 Joystick

We will use the serial port to get Joystick data.

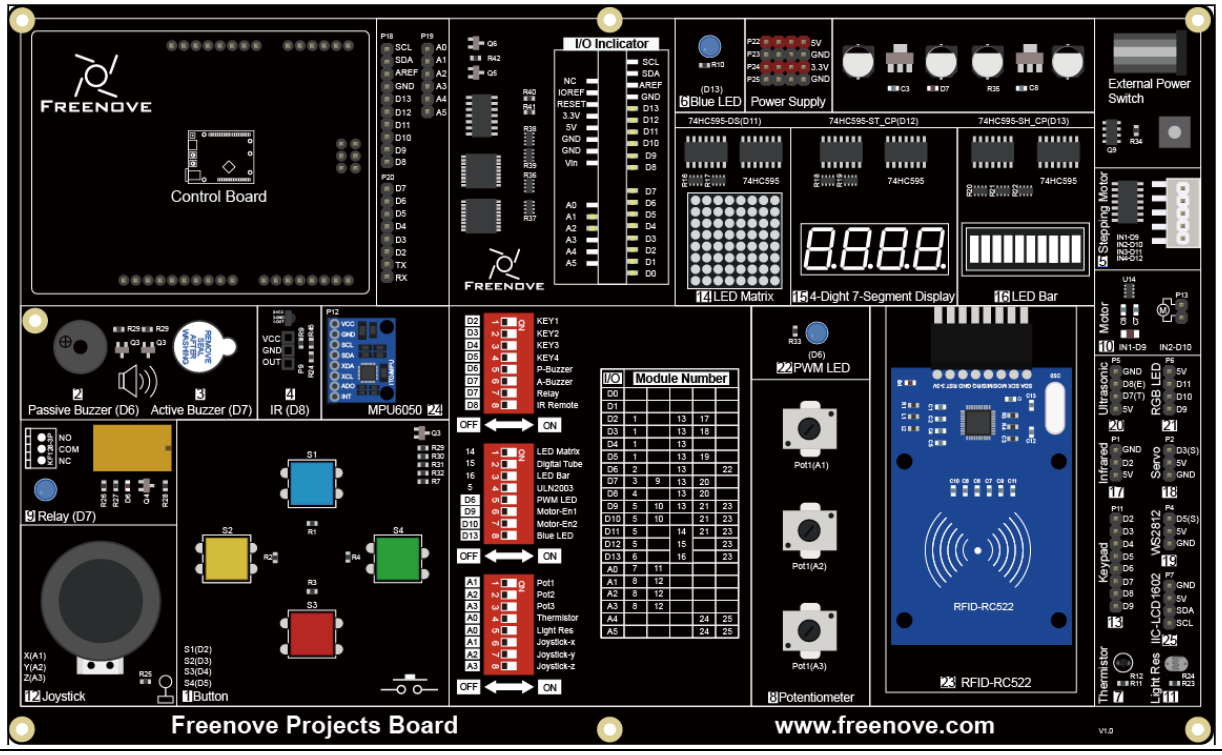
Component List

Control board x1



USB cable x1

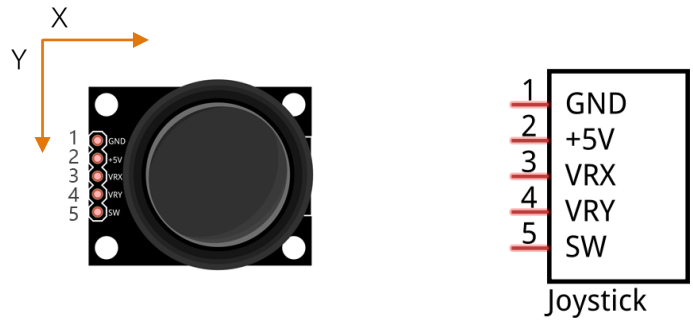
Freenove Projects Board



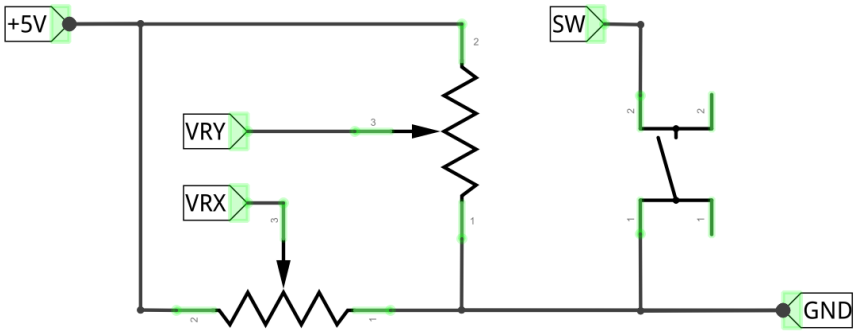
Component Knowledge

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)**.



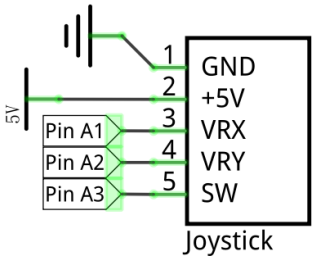
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses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



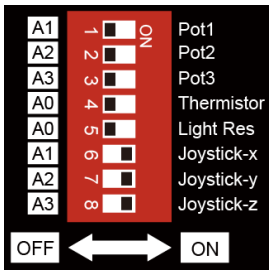
Circuit

Use pin A1 and pin A2 on control board to detect the voltage value of two rotary potentiometers inside Joystick, and use pin A3 port to detect the vertical button.

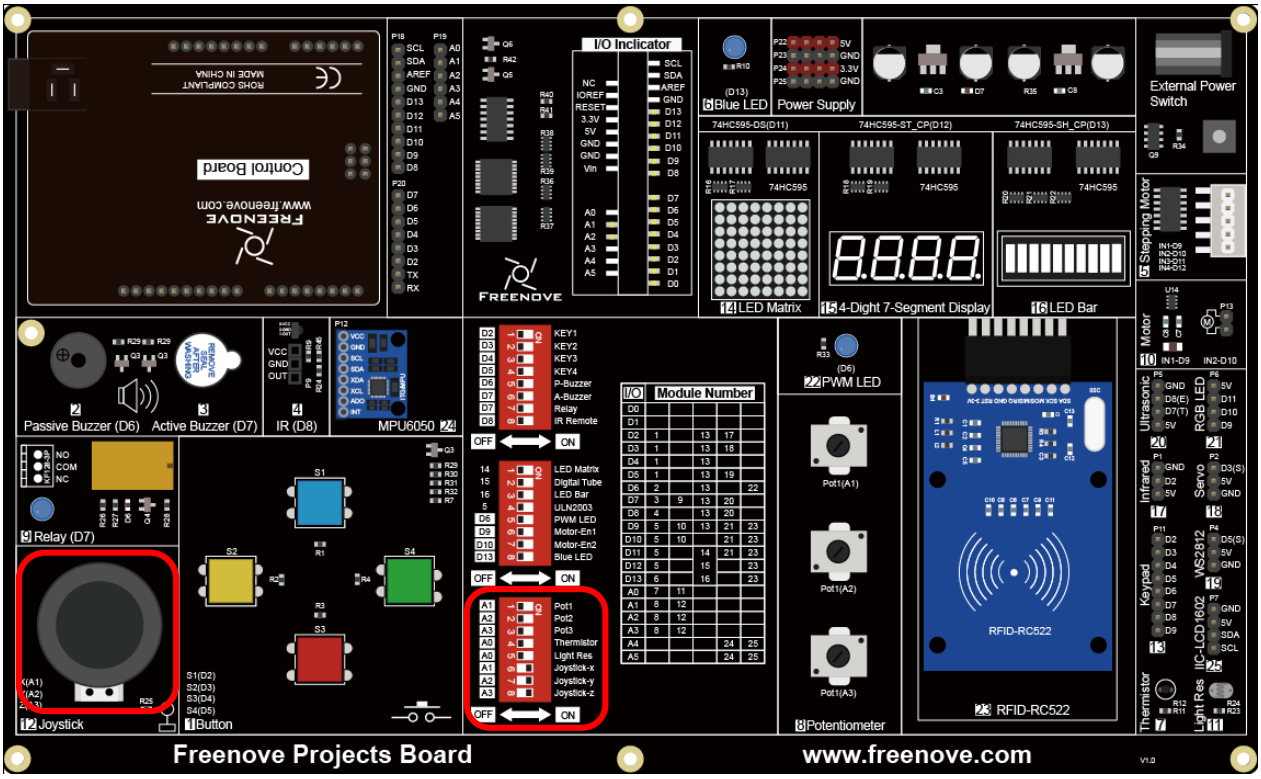
Schematic diagram



Hardware connection



Hardware connection



Sketch

Joystick

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.

```
int xVal, yVal, zVal;       // define 3 variables to store the values of 3 direction

void setup() {
  pinMode(A3, INPUT);       // initialize the port to input
  Serial.begin(9600);       // initialize the serial port with baud rate 9600
}

void loop() {
  // read analog value in XY axis
  xVal = analogRead(A1);
  yVal = analogRead(A2);
  // read digital value of switch in Z axis
  zVal = digitalRead(A3);
  //print the data read above
  Serial.print("X : ");
  Serial.print(xVal);
  Serial.print("   \t Y : ");
  Serial.print(yVal);
  Serial.print("   \t Z : ");
  Serial.println(zVal);
  delay(200);
}
```

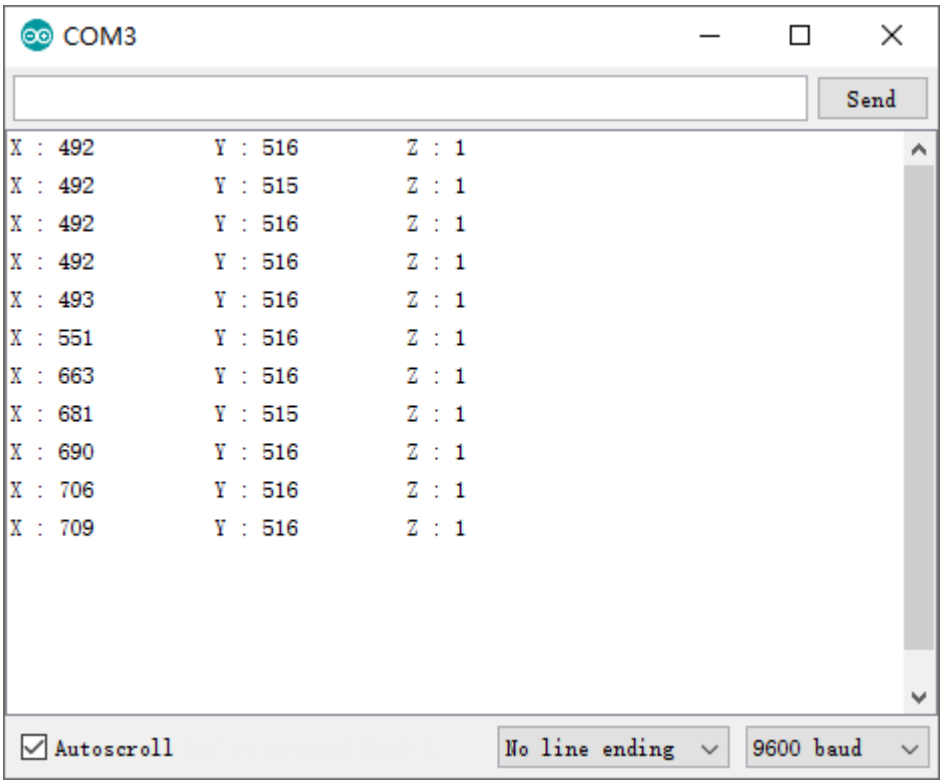
In the code, we get the ADC value of pin A1, A2 and the state of button, and then send the data to serial port.

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.


COM3
Send
X : 492 Y : 516 Z : 1
X : 492 Y : 515 Z : 1
X : 492 Y : 516 Z : 1
X : 492 Y : 516 Z : 1
X : 493 Y : 516 Z : 1
X : 551 Y : 516 Z : 1
X : 663 Y : 516 Z : 1
X : 681 Y : 515 Z : 1
X : 690 Y : 516 Z : 1
X : 706 Y : 516 Z : 1
X : 709 Y : 516 Z : 1
Autoscroll
No line ending
9600 baud

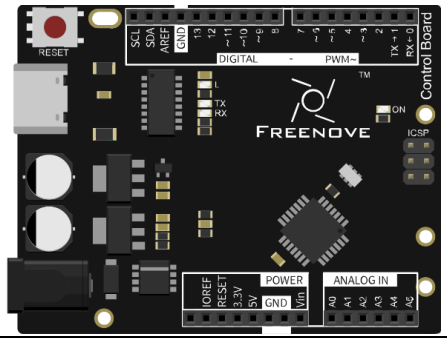
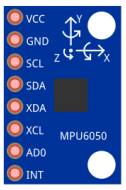
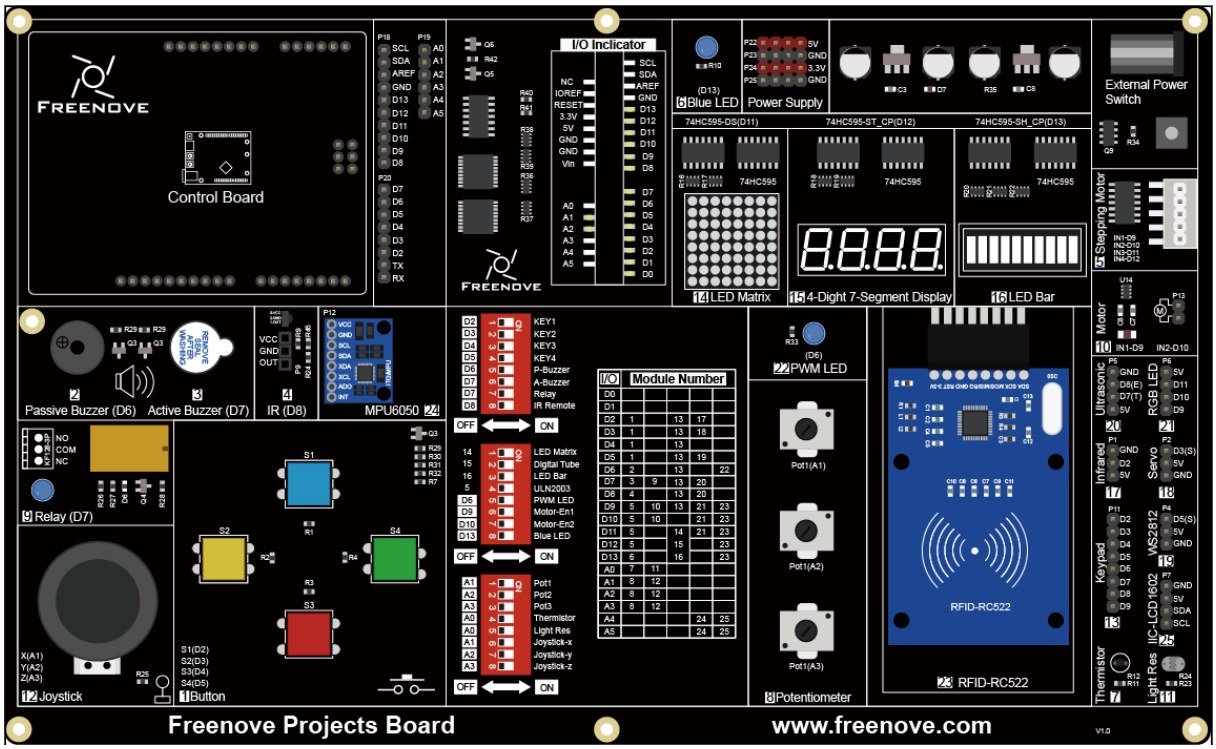
Chapter 14 Acceleration sensor

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 	USB cable x1 	MPU6050 x1 
Freenove Projects Board 		

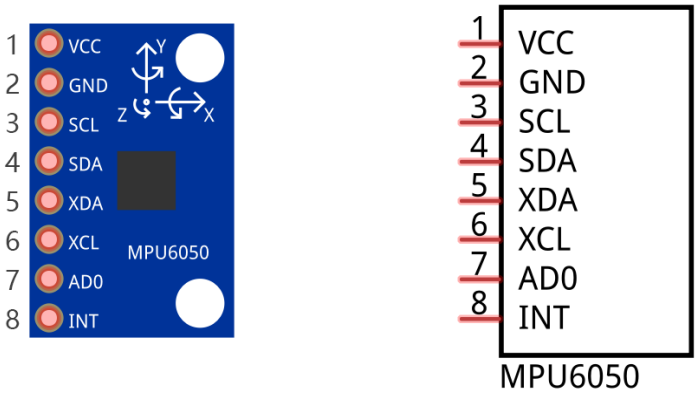
Component Knowledge

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 a voltage of 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

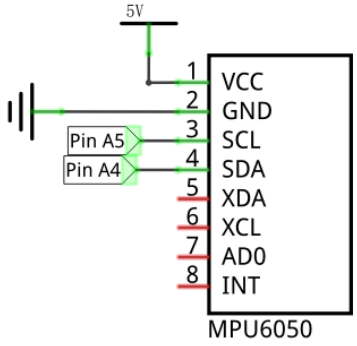
For more detail, please refer to datasheet.

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.

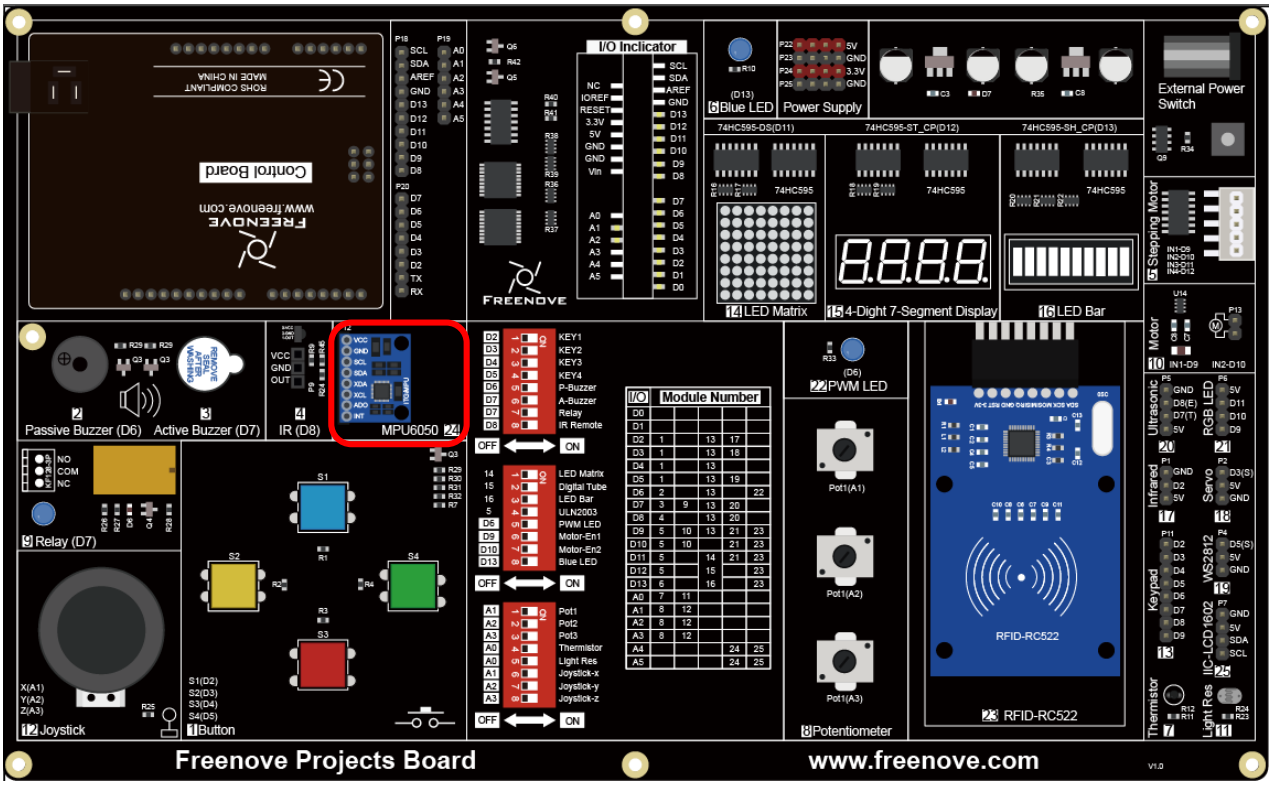
Circuit

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

Schematic diagram



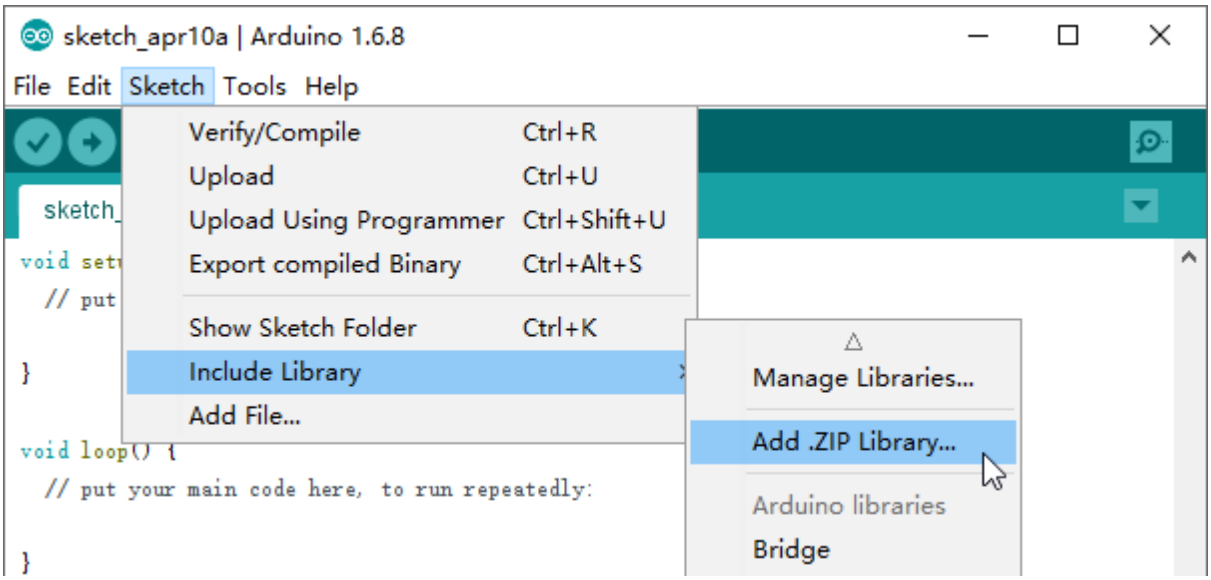
Hardware connection



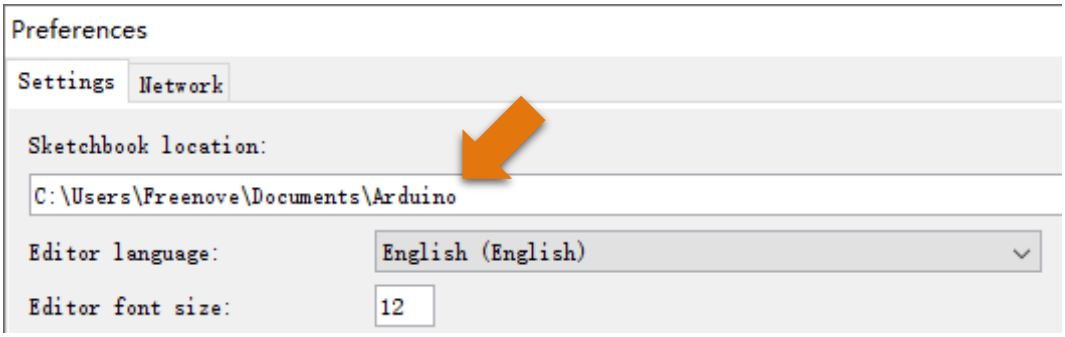
Sketch

Acceleration_Detection

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
```

```
bool blinkState = false;    //Define a variable to save the state of LED

void setup() {
  Serial.begin(9600);       // initialize the serial port, and baud rate is set to 9600
  Serial.println("Initializing I2C devices...");
  Wire.begin();             // initialize I2C
  accelgyro.initialize();   // initialize MPU6050
  Serial.println("Testing device connections...");
  // verify connection
  if (accelgyro.testConnection()) {
    Serial.println("MPU6050 connection successful");
  }
  else {
    Serial.println("MPU6050 connection failed");
    while (1);
  }
  // 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);
  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");
  // initialize LED port
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // read raw accel/gyro measurements from device
  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  // display tab-separated accel/gyro x/y/z values
  Serial.print("a/g:\t");
  Serial.print(ax); Serial.print("\t");
  Serial.print(ay); Serial.print("\t");
  Serial.print(az); Serial.print("\t");
  Serial.print(gx); Serial.print("\t\t");
  Serial.print(gy); Serial.print("\t\t");
  Serial.println(gz);
  // converted acceleration unit to g and the gyroscope unit to dps (degree per second) according to the sensitivity
  Serial.print("a/g:\t");
  Serial.print((float)ax / 16384); Serial.print("g\t");
```

```
52    Serial.print((float)ay / 16384); Serial.print("g\t");
53    Serial.print((float)az / 16384); Serial.print("g\t");
54    Serial.print((float)gx / 131); Serial.print("d/s  \t");
55    Serial.print((float)gy / 131); Serial.print("d/s  \t");
56    Serial.print((float)gz / 131); Serial.print("d/s  \n");
57    delay(300);
58    // blink LED to indicate activity
59    blinkState = !blinkState;
60    digitalWrite(LED_PIN, blinkState);
61  }
```

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

```
2   #include "Wire.h"
3   #include "I2Cdev.h"
4   #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.

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

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

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

Then do a test to confirm whether MPU6050 is connected to the I2C bus and print the related information in serial port.

```
19    if (accelgyro.testConnection()) {
20      Serial.println("MPU6050 connection successful");
21    }
22    else {
23      Serial.println("MPU6050 connection failed");
24      while (1);
25    }
```

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.

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

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

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

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

```
40    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 a 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.

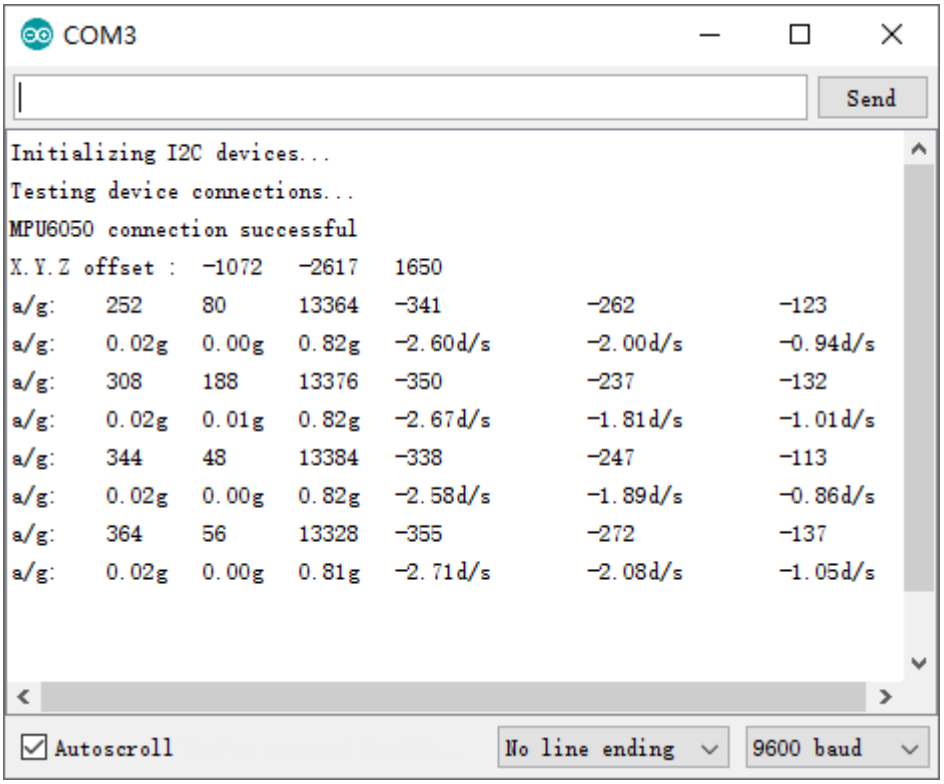
If the parameters of a function are defined as pointer type, then when the function is called, the variable as function parameter will be passed to the function body and participate in its operation, and the value will be changed. It is equivalent to that the function indirectly return more value.

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, and you can see the value of MPU6050 in both 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 **Acceleration_Detection**, which only send three direction acceleration values to the serial port, so it will be relatively easier to observe the change of numbers, when you rotate and move this module.

Chapter 15 LED Bar

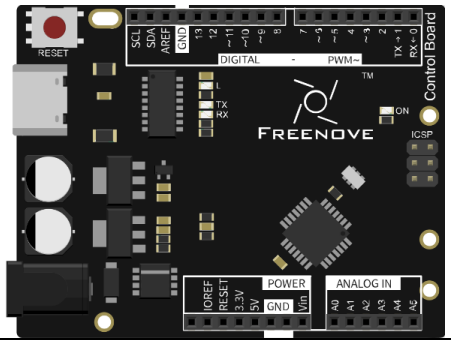
In the previous chapter, we have learned how to use some modules and sensors and display some information on the computer through serial port. Now let us learn some modules which can output images and text. In this chapter, we will learn to drive the LED Bar with 74HC595 chip.

Project 15.1 LED Bar

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

Component List

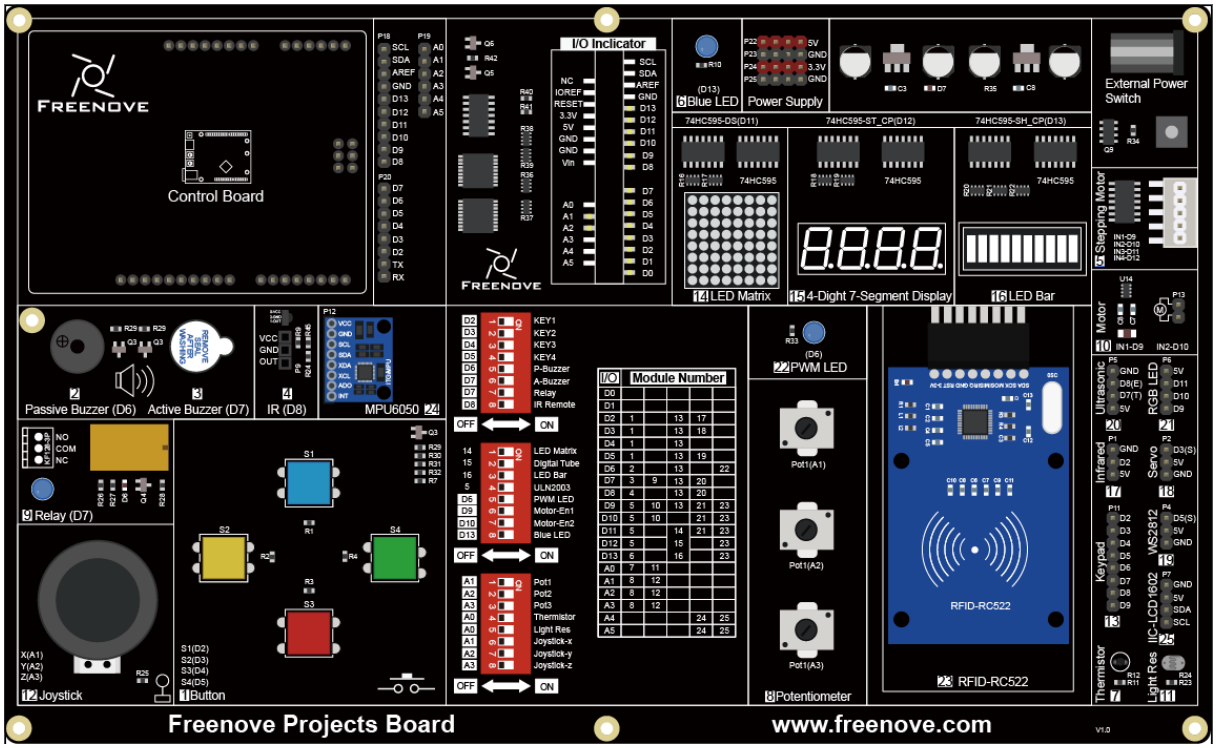
Control board x1



USB cable x1

LED Bar x1

Freenove Projects Board



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. In the code, you need to add the prefix 0x for hexadecimal numbers, such as 0x01.

One Hexadecimal bit can present one number between 0-15. To facilitate writing, numbers greater than 9 are written into letters 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, for example, the sixteen digit 0x12:

Sequence	1	0
Number	1	2

When converting a hexadecimal number to decimal number, first, multiply the nth number of it by n power of 16, and then sum up all multiplicative results. Take 0x12 as an example:

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

When a decimal number is converted to hexadecimal number, the decimal number is divided by 16, so we will get quotient and remainder, and then the quotient obtained will be continuously divided by 16 until quotient is zero. Arrange all remainders from right to left in a line and wecomplete 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 when 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

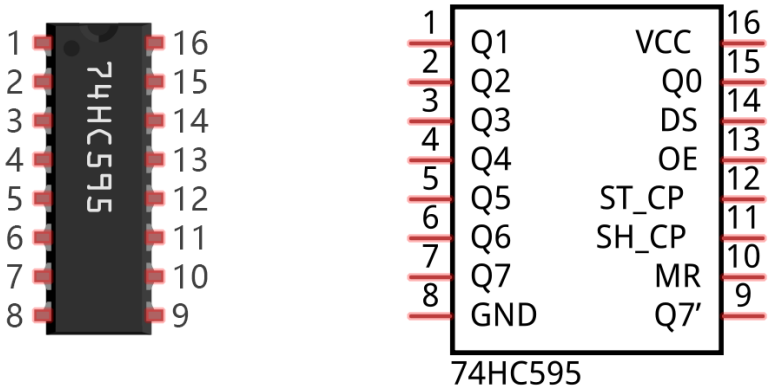
4 bit binary	1000	1001	1010	1011	1100	1101	1110	1111
1 figure of hexadecimal	8	9	A	B	C	D	E	F

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



The ports of 74HC595 are described as follows:

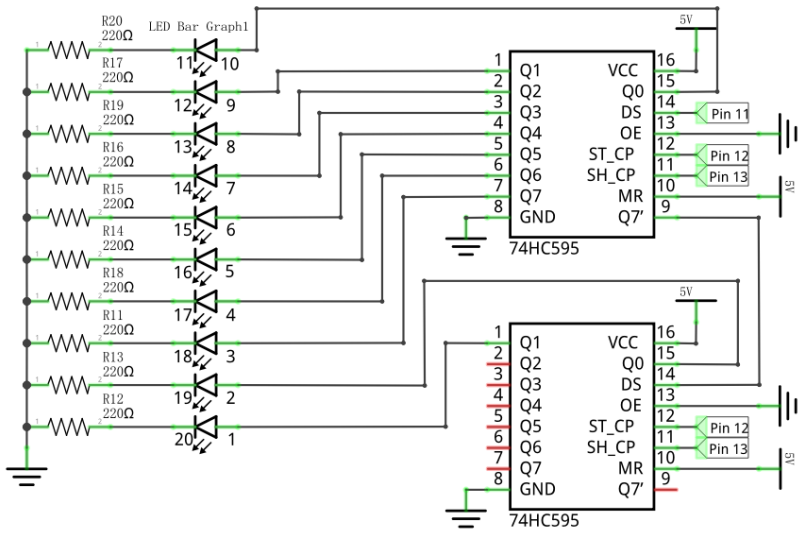
Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin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

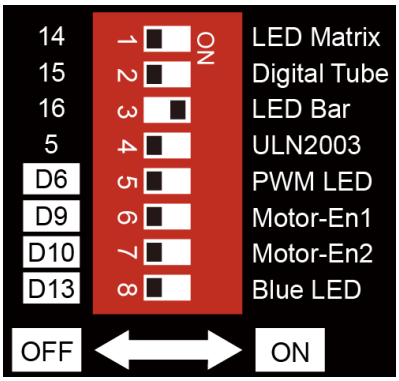
Circuit

Use pin 11, 12, 13 on the control board to control the 74HC595, and connect it to the 10 LEDs of LED bar graph.

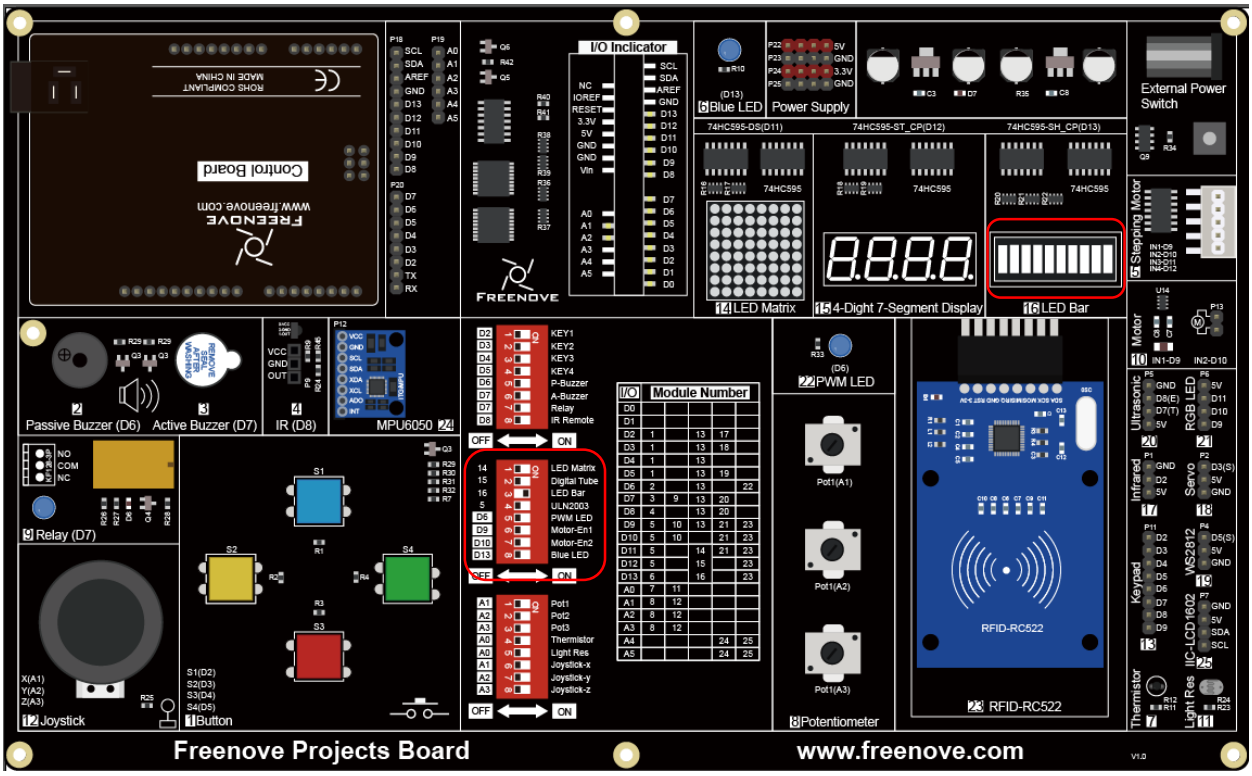
Schematic diagram



Hardware connection



Hardware connection



Sketch

74HC595_LEDBAR

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() {
  int val=0x01;
  for(int i=0;i<10;i++)
  {
    LedBar_MSBFIRST(val);
    delay(100);
    val<<=1;
  }
  val=0x200;
  for(int i=0;i<10;i++)
  {
    LedBar_MSBFIRST(val);
    delay(100);
    val>>=1;
  }
}

void LedBar_MSBFIRST(int value)
{
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, value>>8);//Send high 8 bits of data, high bits first
  shiftOut(dataPin, clockPin, MSBFIRST, value);//Send low 8 bits of data, high bits first
  digitalWrite(latchPin, HIGH);
}
```

In the code, we configure three pins to control 74HC595. And define a variable, through the variable bit to control the state of 10 LEDs. When the corresponding bit is 1, the LED lights up. If the variable is assigned a value of 0x01, which is the binary number 00000001, only one LED light will be on.

```
14    int val=0x01;
```

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

```
30  void LedBar_MSBFIRST(int value)
31  {
32    digitalWrite(latchPin, LOW);
33    shiftOut(dataPin, clockPin, MSBFIRST, value>>8);//Send high 8 bits of data, high bits first
34    shiftOut(dataPin, clockPin, MSBFIRST, value);//Send low 8 bits of data, high bits first
35    digitalWrite(latchPin, HIGH);
36  }
```

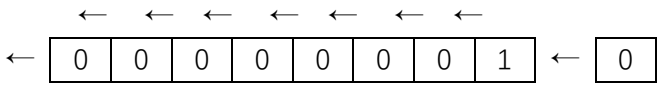
The val will be shift 1 bit to left in each cycle, which makes the light-up LED of the 8 LEDs move one bit, that is, the current ON LED will be OFF, and its left one will light up.

```
24      val <<= 1;
```

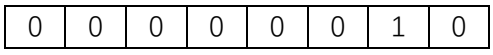
<< operator

"<<" 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 to the right (low). For example, shift binary 00000001 by 1 bit to left:

byte x = 1 << 1;

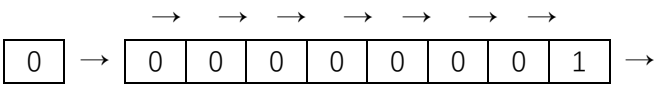


The result of x is 2 (binary 00000010)。

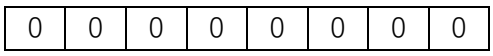


There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

byte x = 1 >> 1;

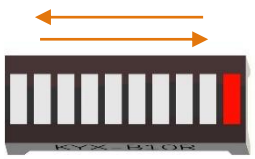


The result of x is 0 (00000000)。



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.



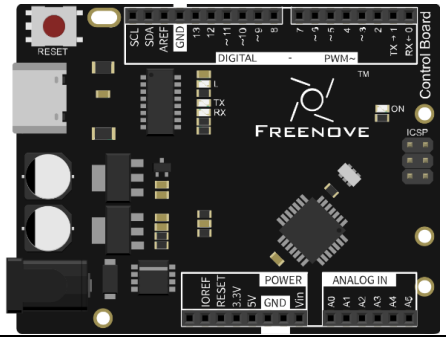
Chapter 16 4-digit 7-segment Display

Project 16.1 4-digit 7-segment Display

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

Component List

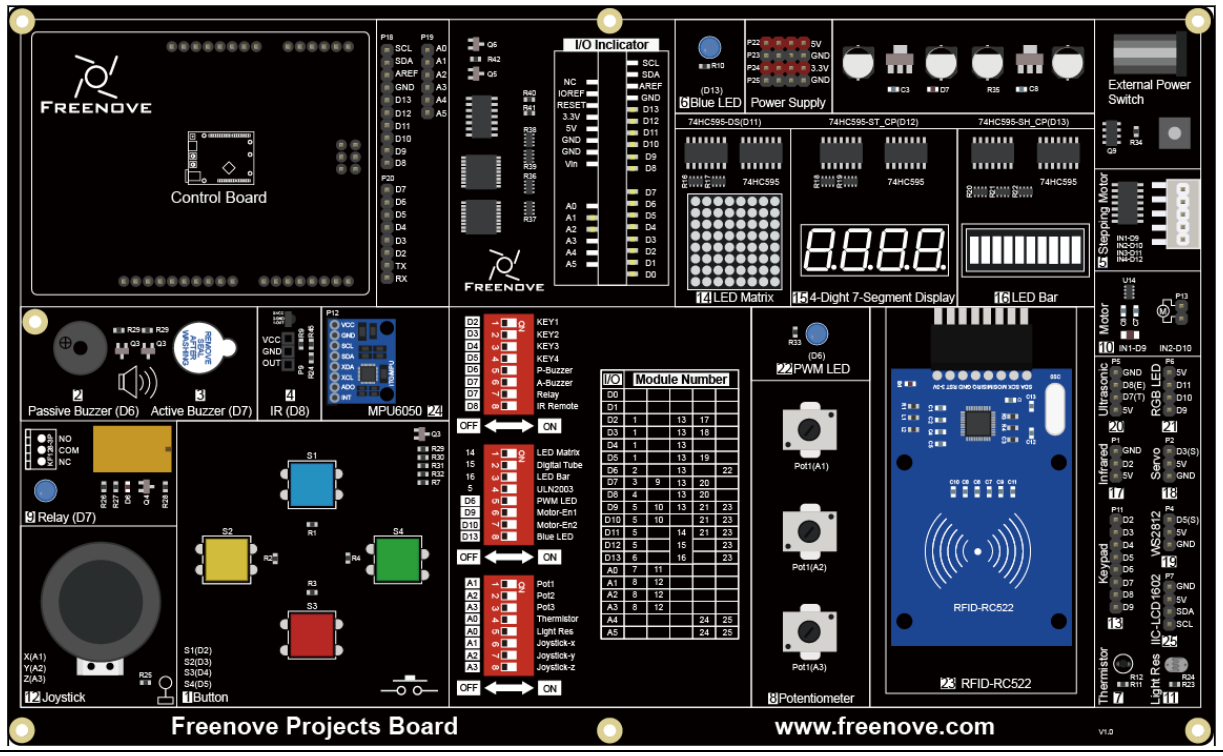
Control board x1



USB cable x1

Digital Tube x1

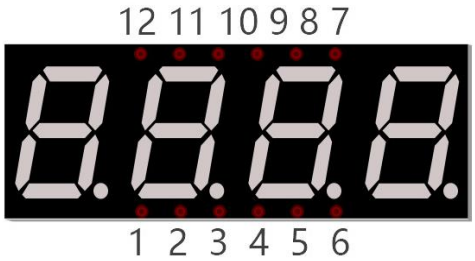
Freenove Projects Board



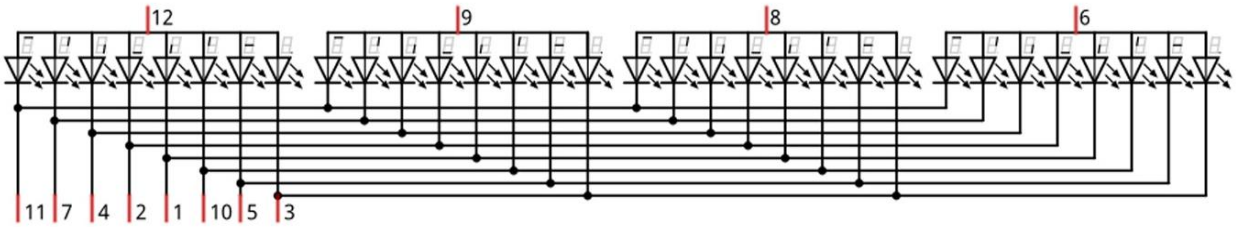
Component Knowledge

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 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 the eight LED cathode pins of each 1-digit 7-segment display are connected together.



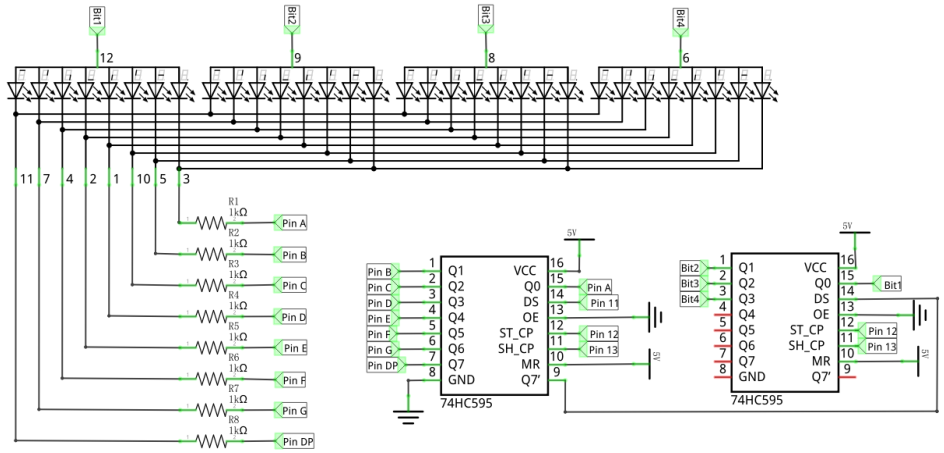
Display method of the 4-Digit 7-segment display is similar to that of the 1-Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and 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.

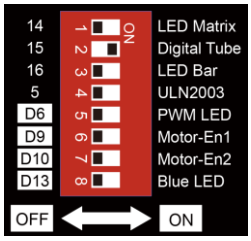
Circuit

Control 74HC595 with pins 11, 12, 13 of the control board, and connect the 4-digit 7-segment display to the board.

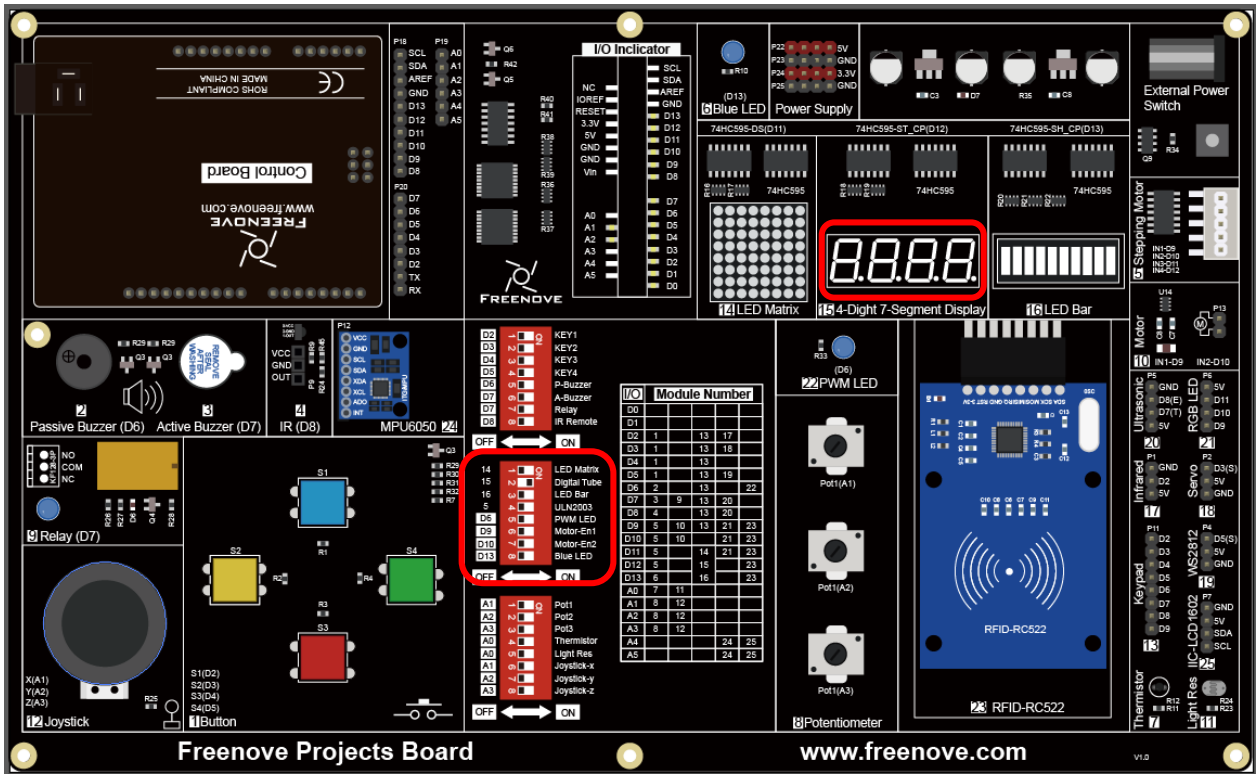
Schematic diagram



Hardware connection



Hardware connection



Sketch

Digital_Tube

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)

byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,//0-7
              0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};//8-F

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

void loop() {
  for(int j=0;j<100;j++){
    for (int i = 0; i < 4; i++){
      DigitalTube_MSBFIRST(i, num[i]);
      delay(5);
    }
  }
}

void DigitalTube_MSBFIRST(int number, byte value){
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, 0x01 << number);
  shiftOut(dataPin, clockPin, MSBFIRST, value);
  digitalWrite(latchPin, HIGH);
}
```

The 4-digit 7-segment display is a common-anode digital tube, whose code values from 0 to F are as follows:

```
5  byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,//0-7
6                0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};//8-F
```

Write a function to send the display bit data of the digital tube first, and then send its display content. The sending process is as follows

```
24 void DigitalTube_MSBFIRST(int number, byte value){
25   digitalWrite(latchPin, LOW);
26   shiftOut(dataPin, clockPin, MSBFIRST, 0x01 << number);
27   shiftOut(dataPin, clockPin, MSBFIRST, value);
28   digitalWrite(latchPin, HIGH);
29 }
```

First, display "0" on the first digital tube for 5 milliseconds, and then display "1" on the second one, for 5 milliseconds, and then display "2" on the third one for 5 milliseconds, and finally, display "3" on the fourth digital tube for 5 milliseconds. Repeat this process 100 times.

When the digital tube is displaying, although the four number characters are displayed in turn separately, this process is so fast that it is unperceivable to the naked eye, so what we observer is that all the four number characters are displaying at the same time. Based on this, we can make it display any number we want.

If it is difficult for you to understand, you can modify delay(5) to delay(500) to slow down the process by 100 times.

```
16   for(int j=0;j<100;j++){
17     for (int i = 0; i < 4; i++){
18       DigitalTube_MSBFIRST(i, num[i]);
19       delay(5);
20     }
21   }
```

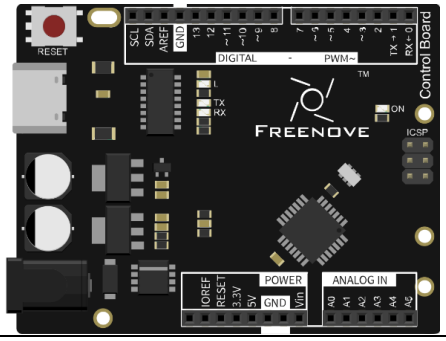
Verify and upload the code, and then you will see number 0123 shown on 4 digit 7-segment display.

Project 16.2 4-digit 7-segment Display

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

Component List

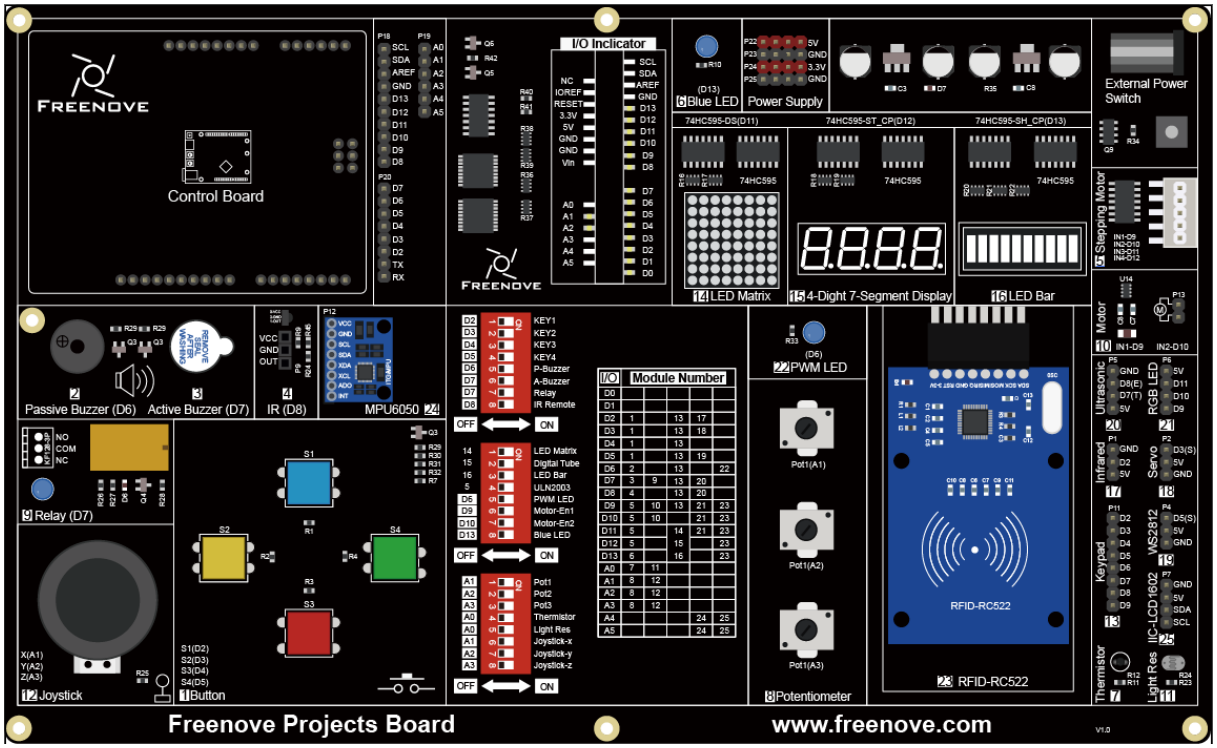
Control board x1



USB cable x1

Digital Tube x1

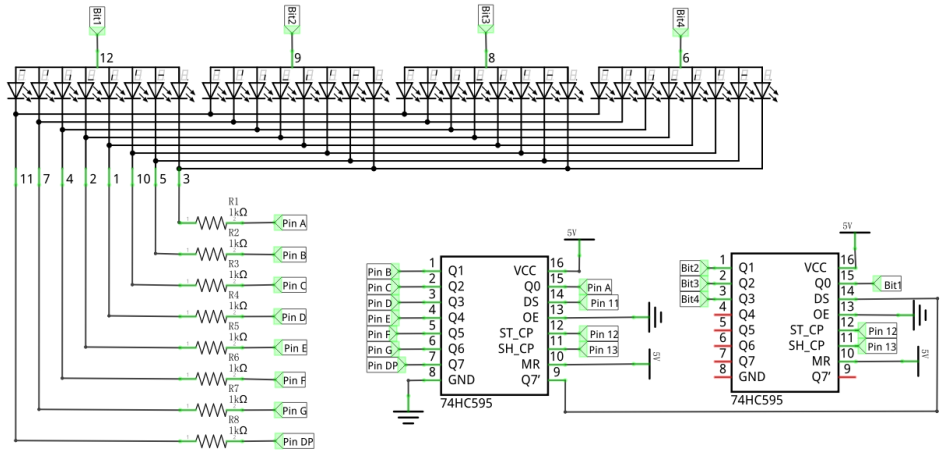
Freenove Projects Board



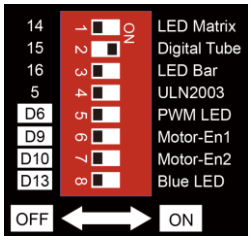
Circuit

Control 74HC595 with pins 11, 12, 13 of the control board, and connect the 4-digit 7-segment display to the board.

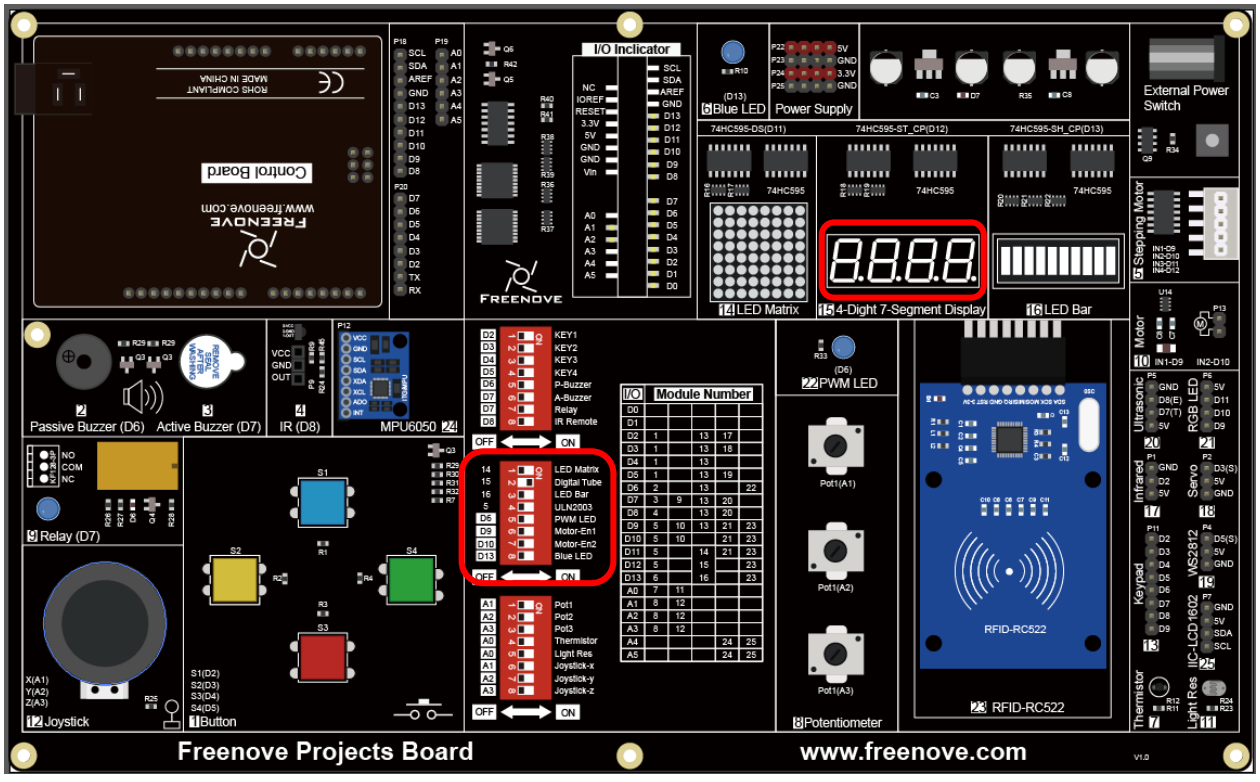
Schematic diagram



Hardware connection



Hardware connection



Sketch

Digital_Tube

Now, we use the digital tube to display counting.

```
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 countValue = 0;         // Digital tube display value

byte num[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,//0-7
              0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e
             };//8-F

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

void loop() {
  for (int j = 0; j < 250; j++) {
    ShowCount(countValue);
  }
  countValue++;
}

void ShowCount(int value) {
  DigitalTube_MSBFIRST(0, num[value % 10000 / 1000]); delay(1);
  DigitalTube_MSBFIRST(1, num[value % 1000 / 100]);   delay(1);
  DigitalTube_MSBFIRST(2, num[value % 100 / 10]);     delay(1);
  DigitalTube_MSBFIRST(3, num[value % 10]);           delay(1);
}

void DigitalTube_MSBFIRST(int number, byte value) {
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, 0x01 << number);
  shiftOut(dataPin, clockPin, MSBFIRST, value);
  digitalWrite(latchPin, HIGH);
}
```

Compile a function that converts the number to be displayed into the display data for each digital tube.

```
24  void ShowCount(int value) {
25    DigitalTube_MSBFIRST(0, num[value % 10000 / 1000]); delay(1);
26    DigitalTube_MSBFIRST(1, num[value % 1000 / 100]);   delay(1);
27    DigitalTube_MSBFIRST(2, num[value % 100 / 10]);     delay(1);
28    DigitalTube_MSBFIRST(3, num[value % 10]);           delay(1);
29  }
```

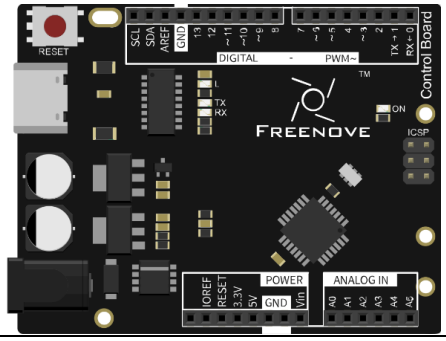
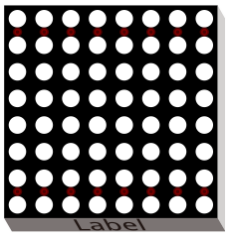
Compile and upload the code, and you can see the number on digital tube increases by one every approximately 1s

Chapter 17 LED Matrix

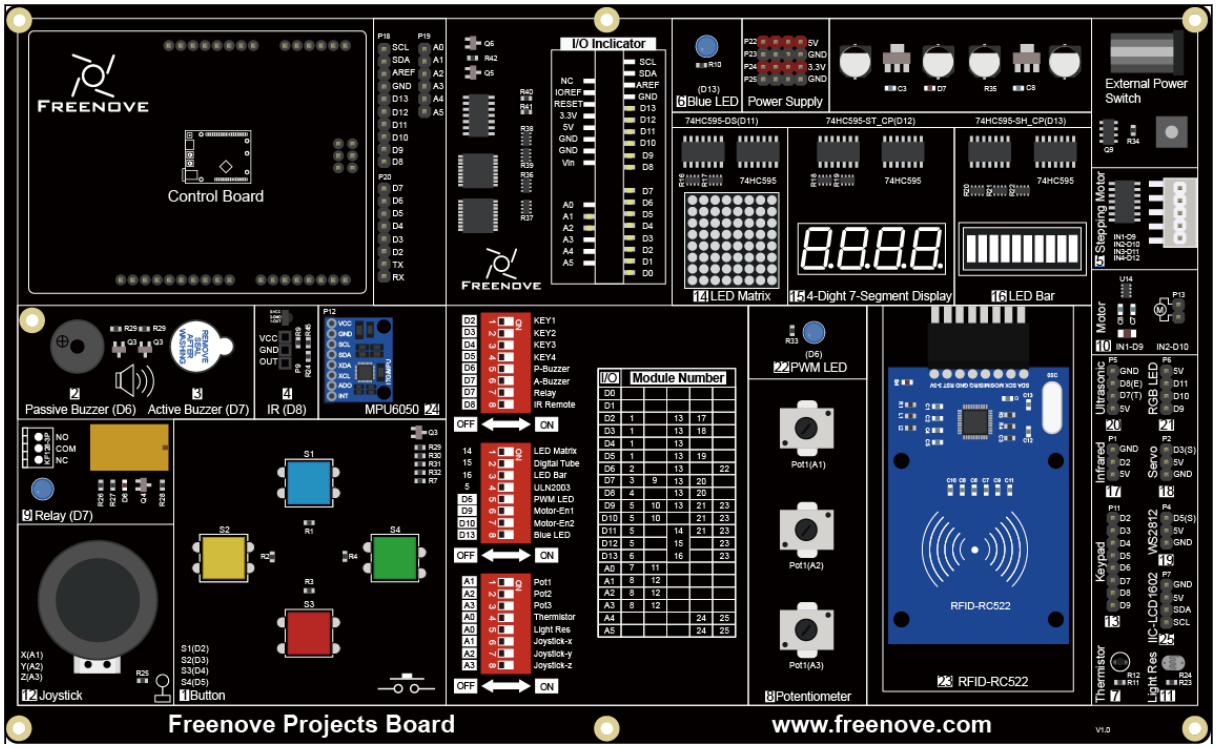
Project 17.1 LED Matrix

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

Component List

Control board x1 	USB cable x1 	LED Matrix 

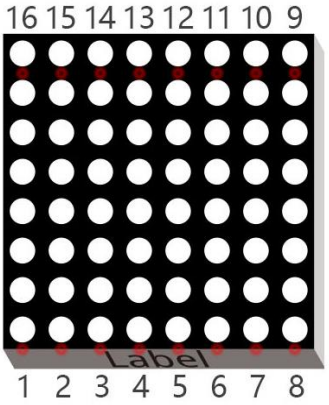
Freenove Projects Board



Component Knowledge

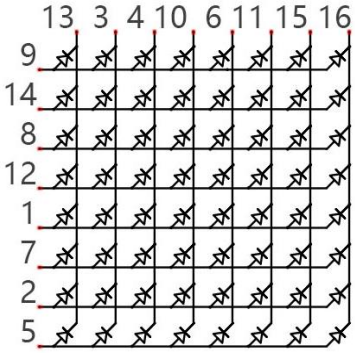
LED matrix

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

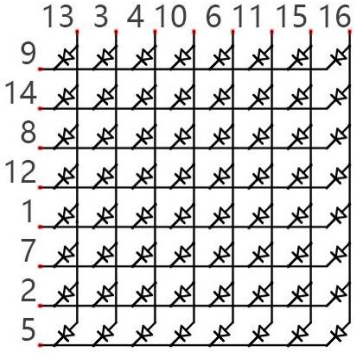


In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another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

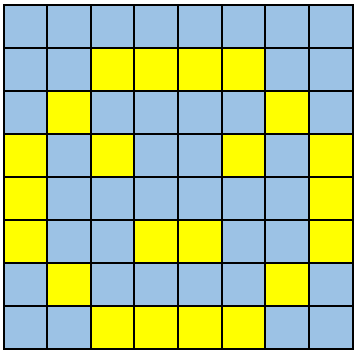
Connection mode of common anode



Connection mode of common cathode



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



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

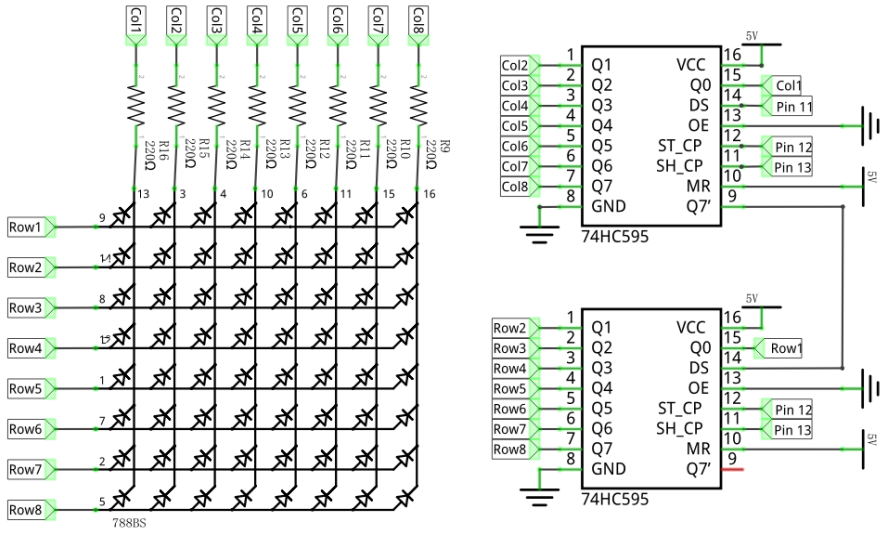
Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

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

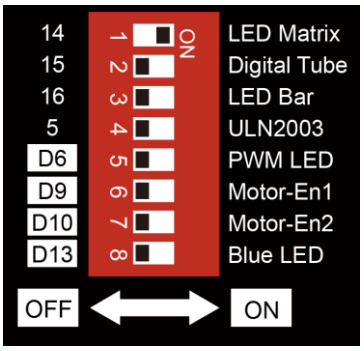
Circuit

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

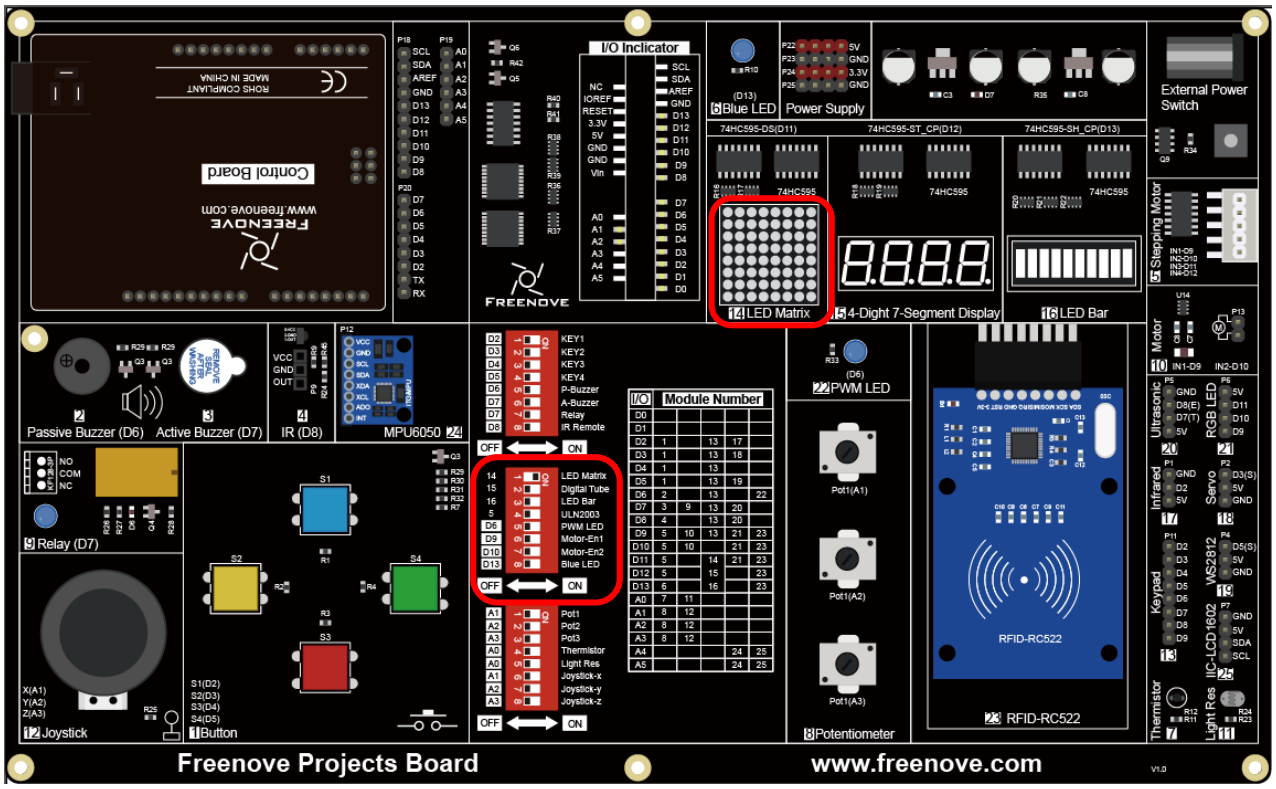
Schematic diagram



Hardware connection



Hardware connection



Sketch

LED_Matrix

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

```
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)

// 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);
}

void loop() {
```

```
39    // Define a one-byte variable (8 bits) which is used to represent the selected state of 8
      column.
40    int cols;
41    // Display the static smiling pattern
42    for (int j = 0; j < 500; j++ ) {  // repeat 500 times
43      cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the first
      column is selected.
44      for (int i = 0; i < 8; i++) {   // display 8 column data by scaning
45        matrix(smilingFace[i],cols);
46        delay(1);                     // display them for a period of time
47        cols <<= 1;                   // shift"cols" 1 bit left to select the next column
48      }
49    }
50    // Display the dynamic patterns of numbers and letters
51    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)
52      for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
53        cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
54        for (int j = i; j < 8 + i; j++) { // display image of each frame
55          matrix(pgm_read_word_near(data + j),cols);
56          delay(1);                       // display them for a period of time
57          cols <<= 1;                     // shift"cols" 1 bit left to select the next column
58        }
59      }
60    }
61  }
62
63  void matrix(byte row_value,byte col_value) {
64    digitalWrite(latchPin, LOW);
65    shiftOut(dataPin, clockPin, LSBFIRST, row_value);
66    shiftOut(dataPin, clockPin, MSBFIRST, ~col_value);
67    digitalWrite(latchPin, HIGH);
68  }
```

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

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

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 a function to send the data of the LEDs that need to be lit for each row first, and then send the data of those for each column. Details are as follows.

```
63  void matrix(byte row_value,byte col_value) {
64    digitalWrite(latchPin, LOW);
65    shiftOut(dataPin, clockPin, LSBFIRST, row_value);
66    shiftOut(dataPin, clockPin, MSBFIRST, ~col_value);
67    digitalWrite(latchPin, HIGH);
68  }
```

Bitwise logical operator

There are many bitwise logical operators such as and (&), or (|), xor (^), negate (~). The result of exclusive or (^) is true only when its arguments differ (one is true, the other is false).

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

```
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 result. Repeat the process for 500 times, and then we can see a static smile pattern.

```
42    for (int j = 0; j < 500; j++ ) {  // repeat 500 times
43      cols = 0x01; // Assign 0x01(binary 00000001) to the variable, which represents the first
      column is selected.
44      for (int i = 0; i < 8; i++) {   // display 8 column data by scaning
45        matrix(smilingFace[i],cols);
46        delay(1);                     // display them for a period of time
47        cols <<= 1;                   // shift"cols" 1 bit left to select the next column
48      }
49    }
```

Then display the dynamic pattern of numbers and letters. We have defined space, 0-9, A-F, a 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 128 frames of image from the first frame (1-8) to the last frame (128-136 columns). 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.

```
51    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)
52      for (int k = 0; k < 10; k++) {      // repeat image of each frame 10 times.
53        cols = 0x01;      // Assign binary 00000001. Means the first column is selected.
54        for (int j = i; j < 8 + i; j++) { // display image of each frame
55          matrix(pgm_read_word_near(data + j),cols);
56          delay(1);                       // display them for a period of time
57          cols <<= 1;                     // shift"cols" 1 bit left to select the next column
58        }
59      }
60    }
```

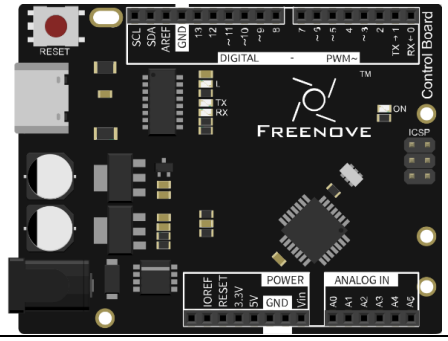
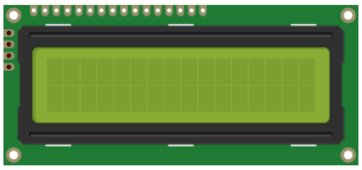
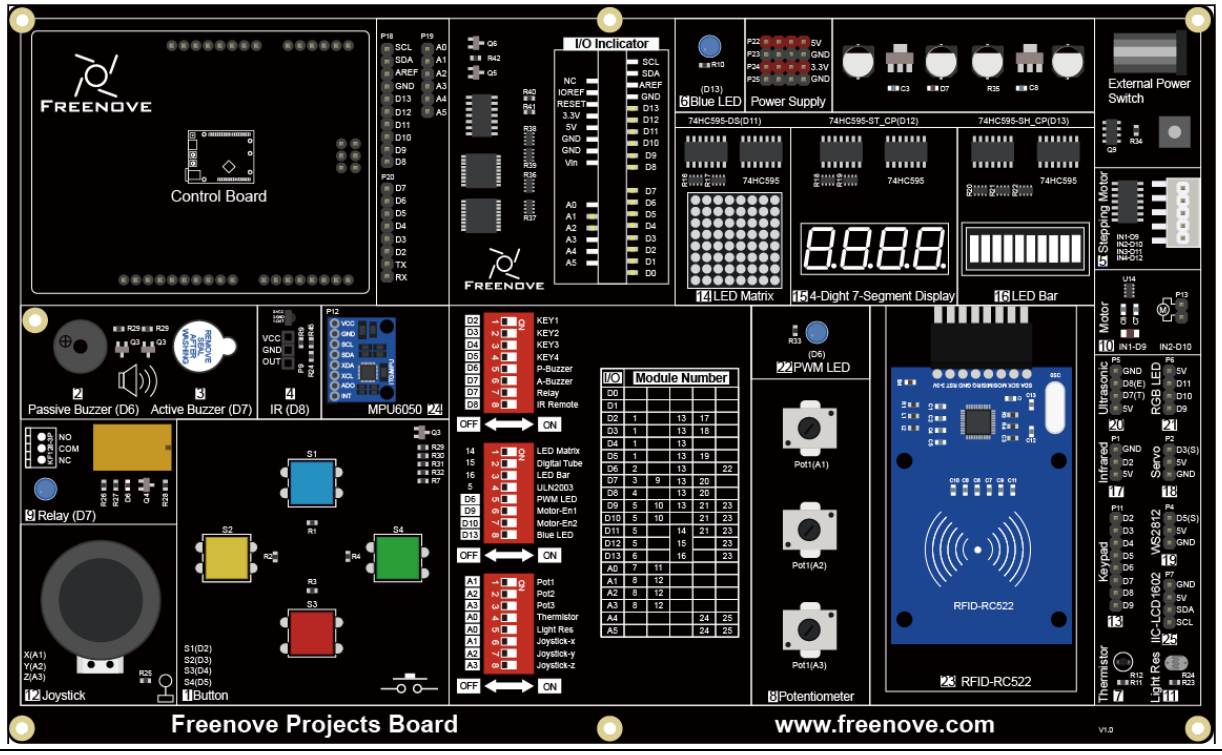
Verify and upload the code, and LED Matrix begins to display the static smile pattern. A few seconds later, LED Matrix will display the scrolling number 0-9 and the letter A-F.

Chapter 18 I2C LCD1602

Project 18.1 Display the String on I2C LCD1602

Firstly, use I2C LCD1602 to display some strings.

Component List

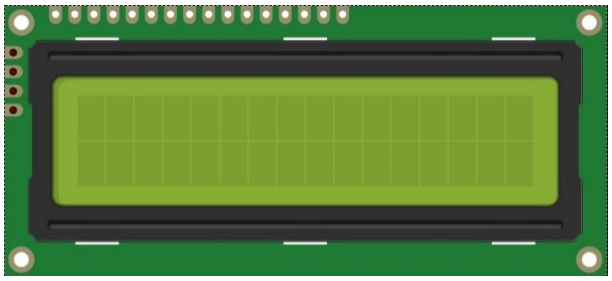
Control board x1 	USB cable x1 	I2C-LCD1602 x1 
Jumper Wire x4 		
Freenove Projects Board 		

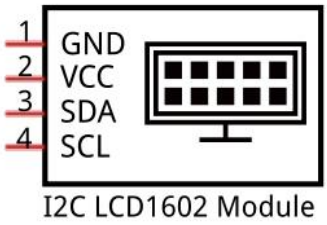
Component Knowledge

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





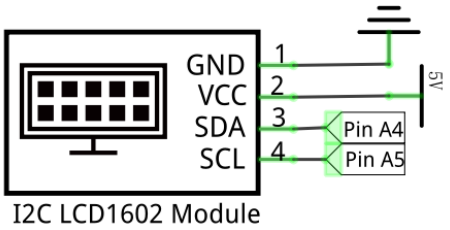
I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique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

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

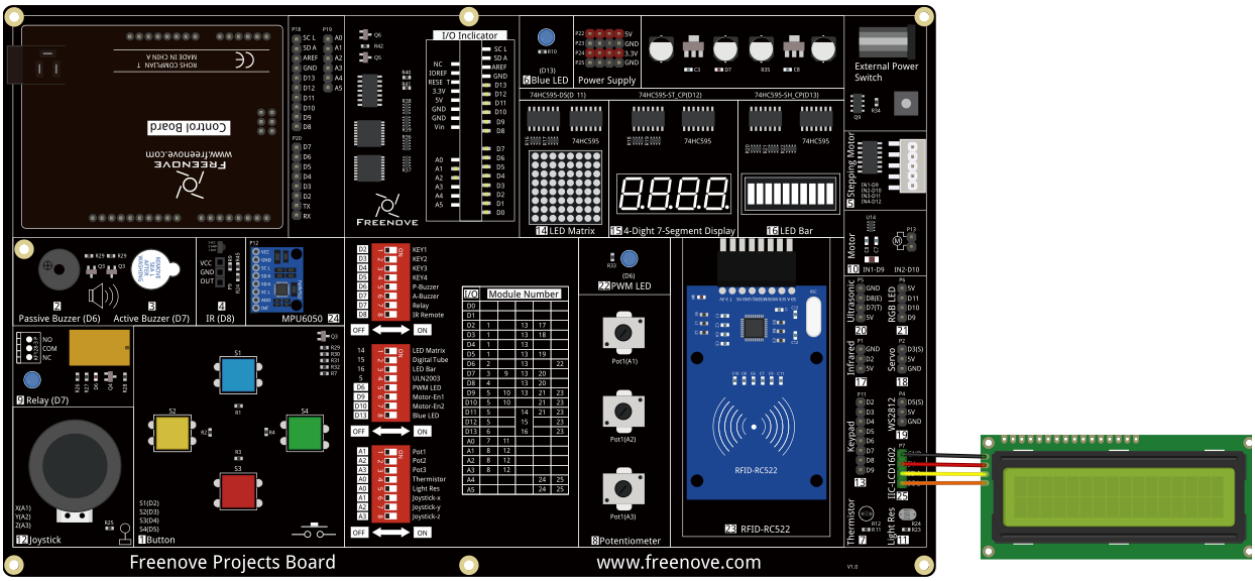
Circuit

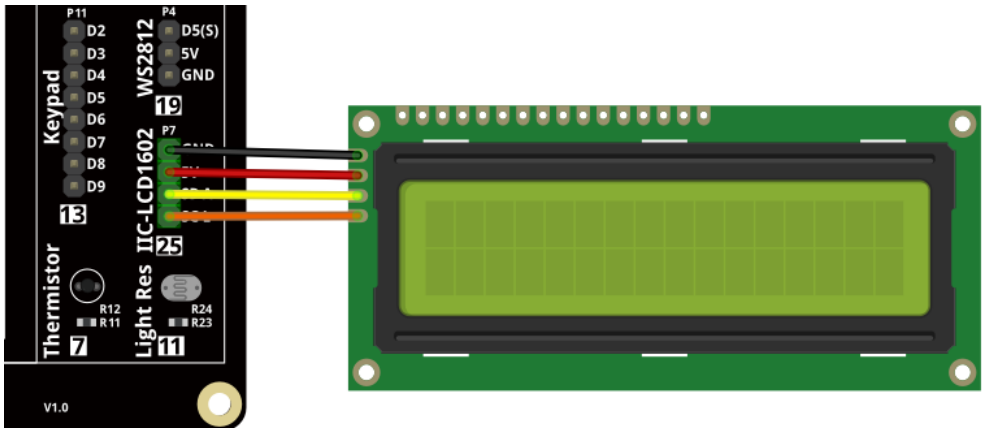
The connection of control board and I2C LCD1602 is shown below.

Schematic diagram



Hardware connection





Sketch

Display_the_string_on_LCD1602

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() {
  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);
}
```

The following is 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.

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

And then print a string:

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

Print a changing number in the loop () function:

```
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);
}
```

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

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

LiquidCrystal_I2C Class

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

LiquidCrystal_I2C lcd(0x27, 16, 2);

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

The functions used in the LiquidCrystal_I2C class are as follows:

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

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

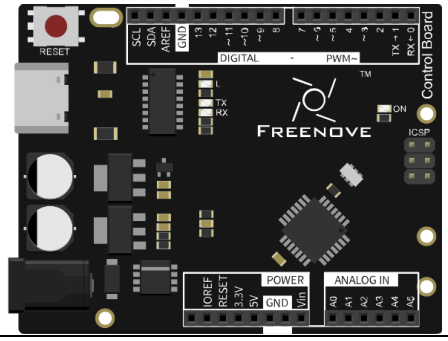
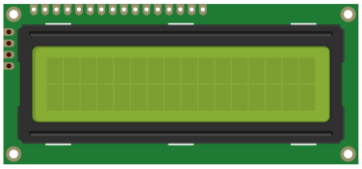
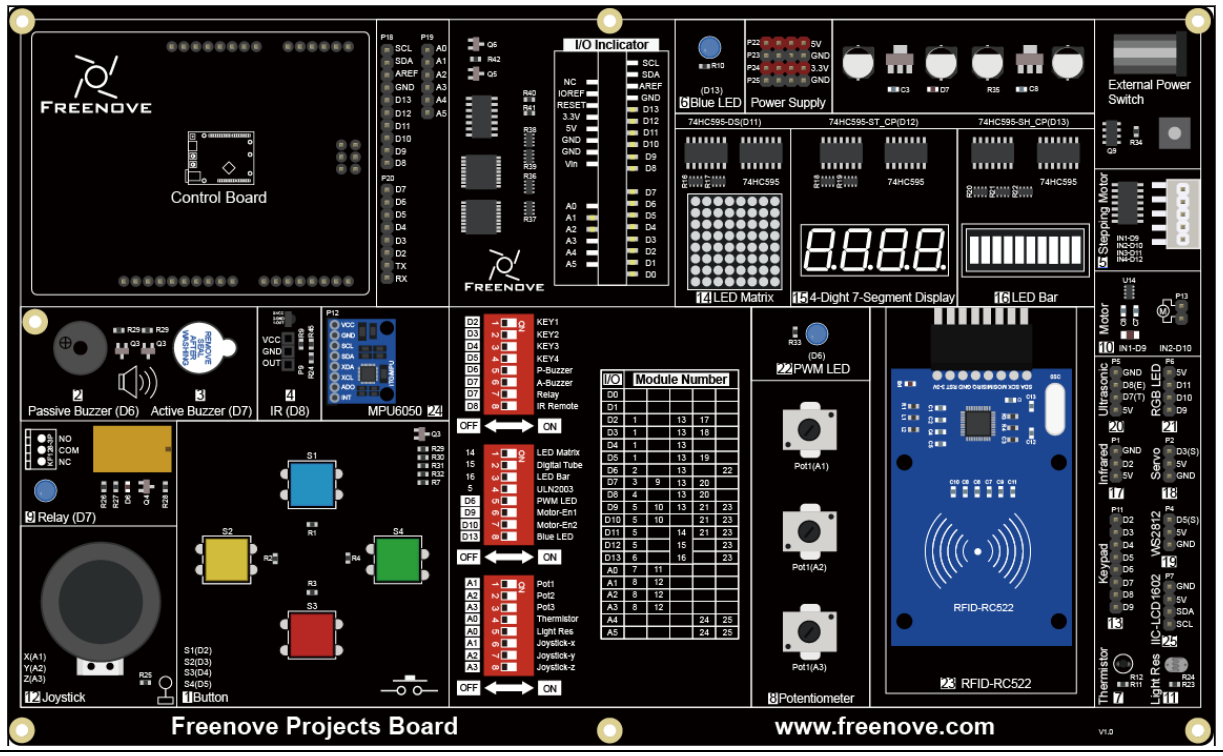
Verify and upload the code, then observe the LCD screen. If the display is not clear or there is no display, adjust the potentiometer on the back of I2C module to adjust the screen contrast until the character is clearly displayed on the LCD.

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

Project 18.2 I2C LCD1602 Clock

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

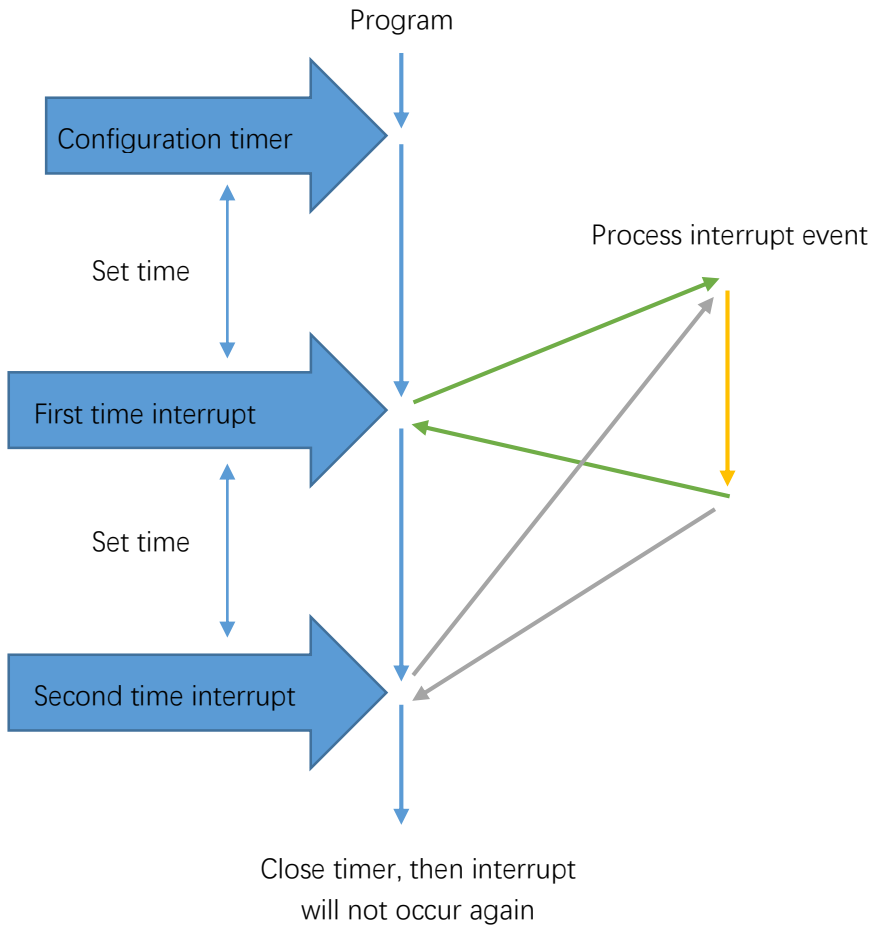
Component List

Control board x1 	USB cable x1 	IIC-LCD1602 x1 
Jumper Wire x4 		
Freenove Projects Board 		

Code Knowledge

Timer

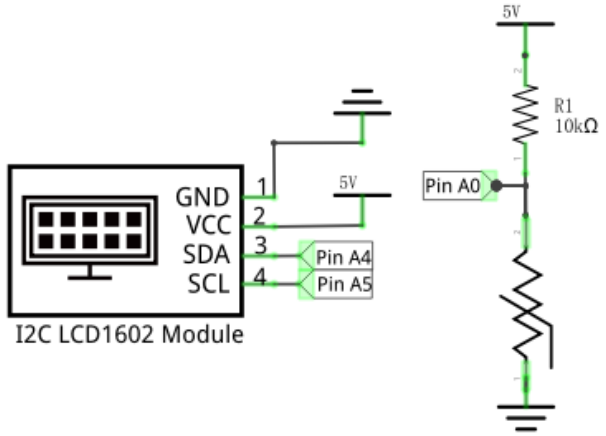
A Timer can be set to produce an interrupt after a period of time. When a timer interrupt occurs, the processor will jump to the interrupt function to process the interrupt event. After completing the processing, it will return to the interrupted location to continue the program. If you don't close the timer, interrupt will occur at the intervals you set.



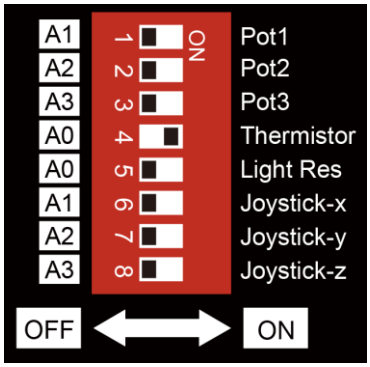
Circuit

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

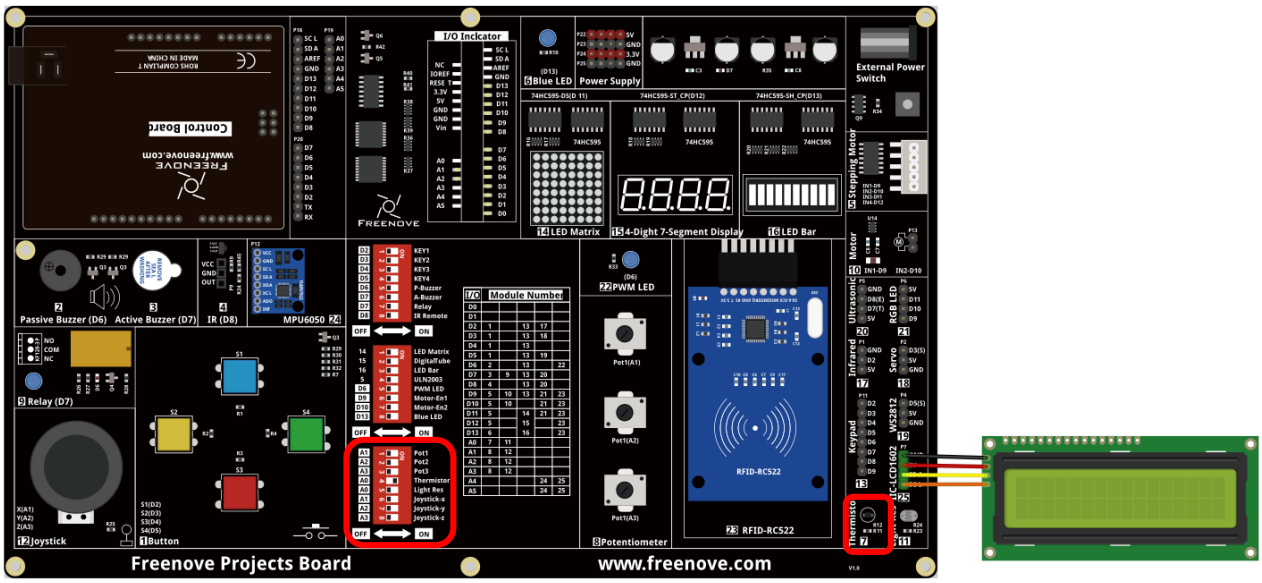
Schematic diagram

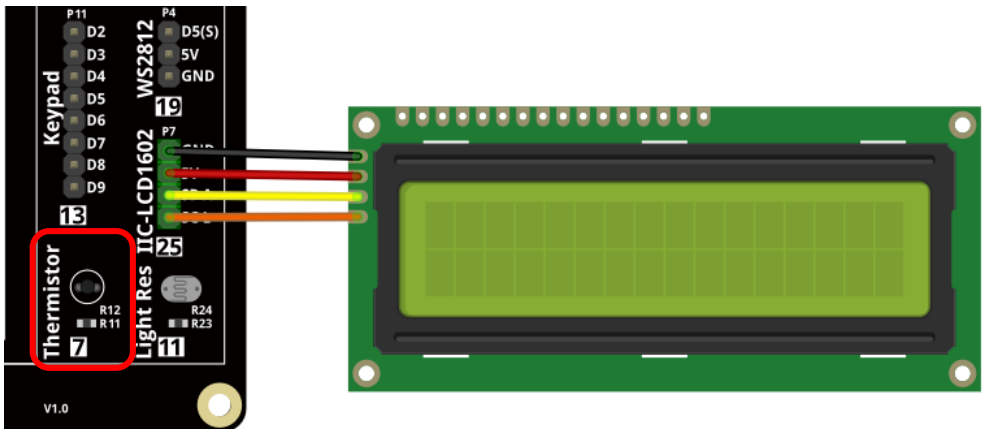


Hardware connection



Hardware connection





Sketch

LCD1602_Clock

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

```
}

void startingAnimation() {
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayRight();
  }
  lcd.print("starting...");
  for (int i = 0; i < 16; i++) {
    lcd.scrollDisplayLeft();
    delay(300);
  }
  lcd.clear();
}
// the timer interrupt function of FlexiTimer2 is executed every 1s
void timerInt() {
  second++;       // second plus 1
}
// serial port interrupt function
void serialEvent() {
  int inInt[3];  // define an array to save the received serial data
  while (Serial.available()) {
    for (int i = 0; i < 3; i++) {
      inInt[i] = Serial.parseInt();  // receive 3 integer data
    }
    // print the received data for confirmation
    Serial.print("Your input is: ");
    Serial.print(inInt[0]);
    Serial.print(", ");
    Serial.print(inInt[1]);
    Serial.print(", ");
    Serial.println(inInt[2]);
    // use received data to adjust time
    hour = inInt[0];
    minute = inInt[1];
    second = inInt[2];
    // print the modified time
    Serial.print("Time now is: ");
    Serial.print(hour / 10);
    Serial.print(hour % 10);
    Serial.print(':');
    Serial.print(minute / 10);
    Serial.print(minute % 10);
    Serial.print(':');
    Serial.print(second / 10);
```

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

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

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

Defines a timer with a cycle of 1000 millisecond (1 second). After each interruption, the number of seconds is increased by 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 each interruption, the number of seconds is increased by 1.

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

:: operator

"::" is a scope operator. The function behind "::" is within the scope of that in front of "::". If we want to call the function defined in the FlexiTimer2 scope externally, we need to use the operator, which can be global scope operator, class scope operator or namespace scope operator. Here we use a namespace scope operator. Because functions of FlexiTimer2 library is defined in the namespace of FlexiTimer2, 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 increases by 1, and the second is reset to zero; when the minute increases to 60, the hour adds by 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, the result of hour/ 10 is the tens digit, hour% 10 is the ones digit.

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

```
99  }
```

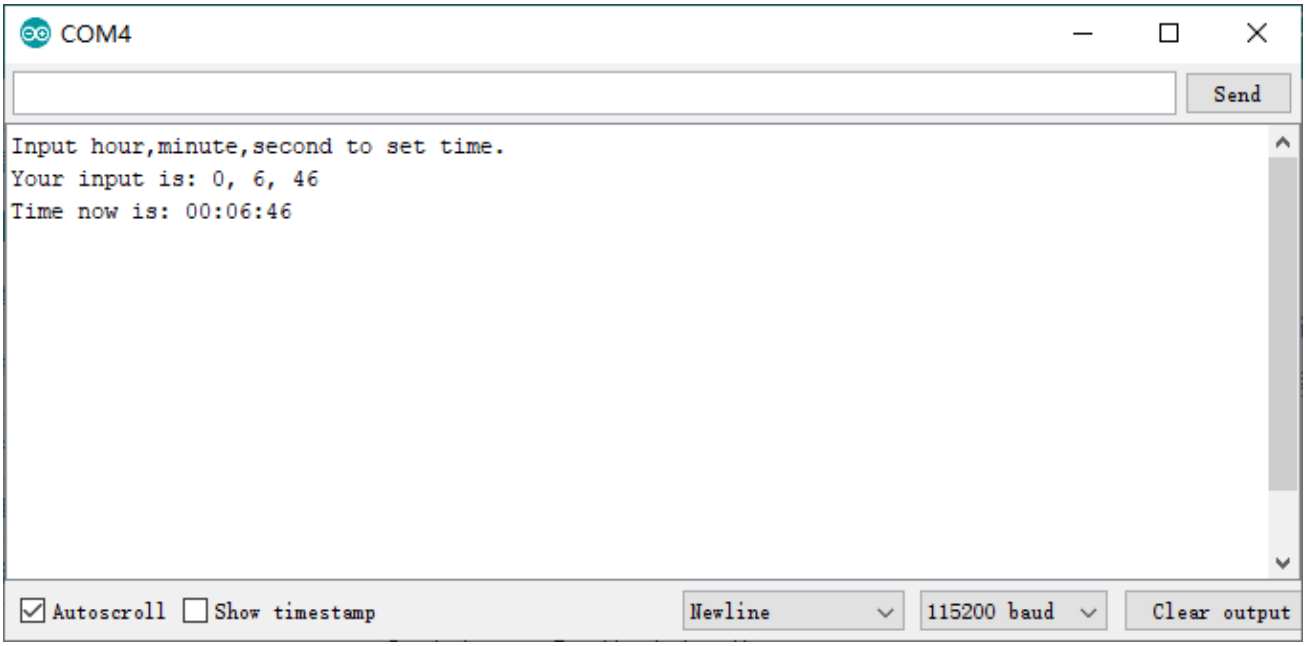
Serial port interrupt function is used to receive the data sent by computer to adjust the time, and return the data for confirmation.

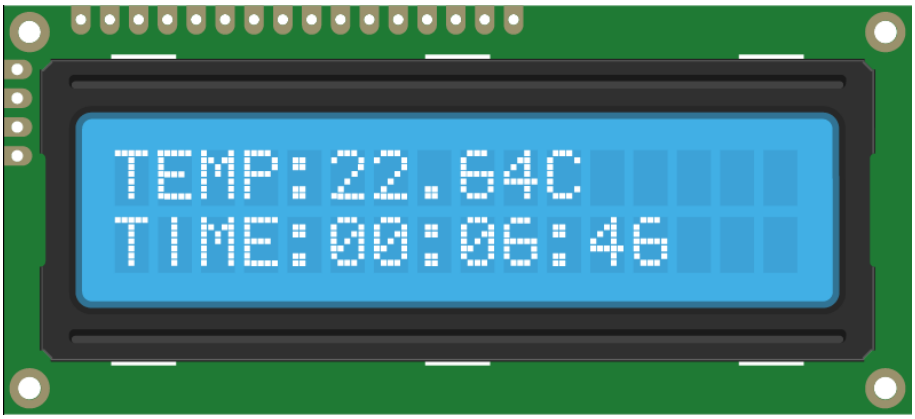
```
54  void serialEvent() {
55    int inInt[3];  // define an array to save the received serial data
56    while (Serial.available()) {
57      for (int i = 0; i < 3; i++) {
58        inInt[i] = Serial.parseInt();  // receive 3 integer data
59      }
60      // print the received data for confirmation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

We also define a function that displays a scrolling string as the control board 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 19 Stepper Motor

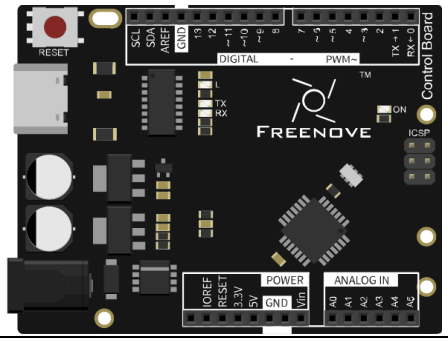
A Stepper motor is a kind of motor that can rotate a certain angle at a time, with which we can achieve mechanical movement at a higher accuracy more easily.

Project 19.1 Drive Stepper Motor

Now, try to drive a stepper motor.

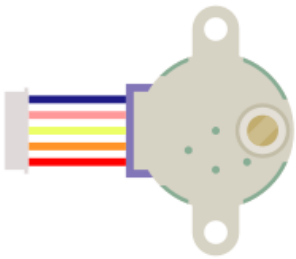
Component List

Control board x1

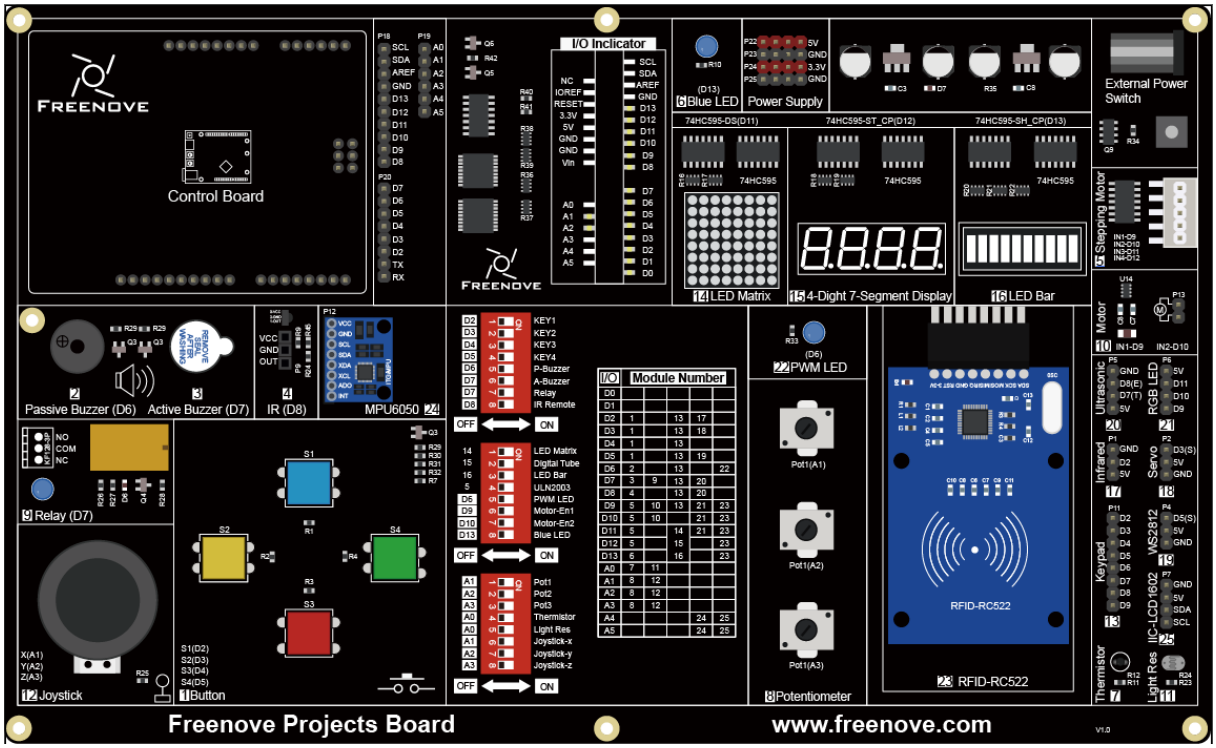


USB cable x1

Stepping Motor x1



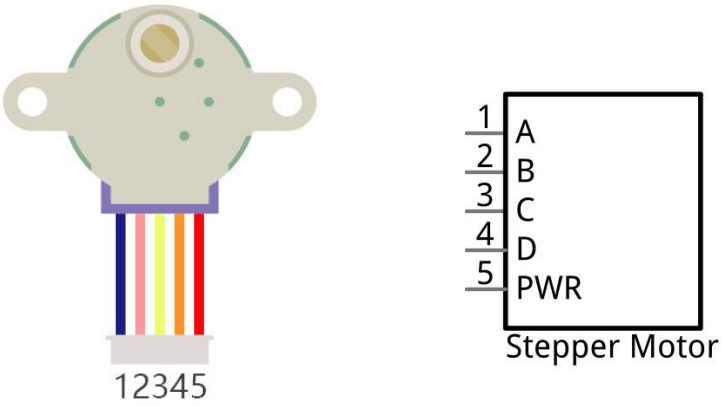
Freenove Projects Board



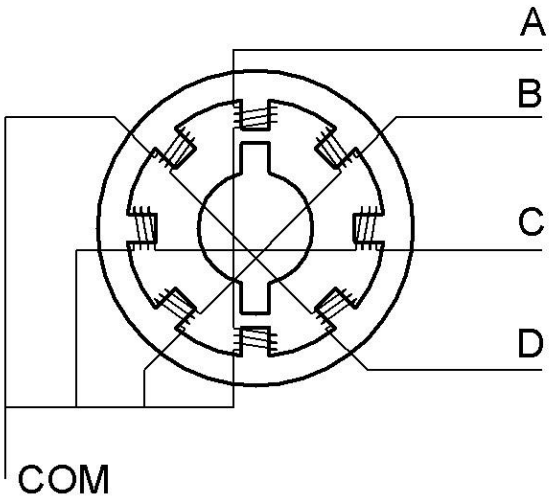
Component Knowledge

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:

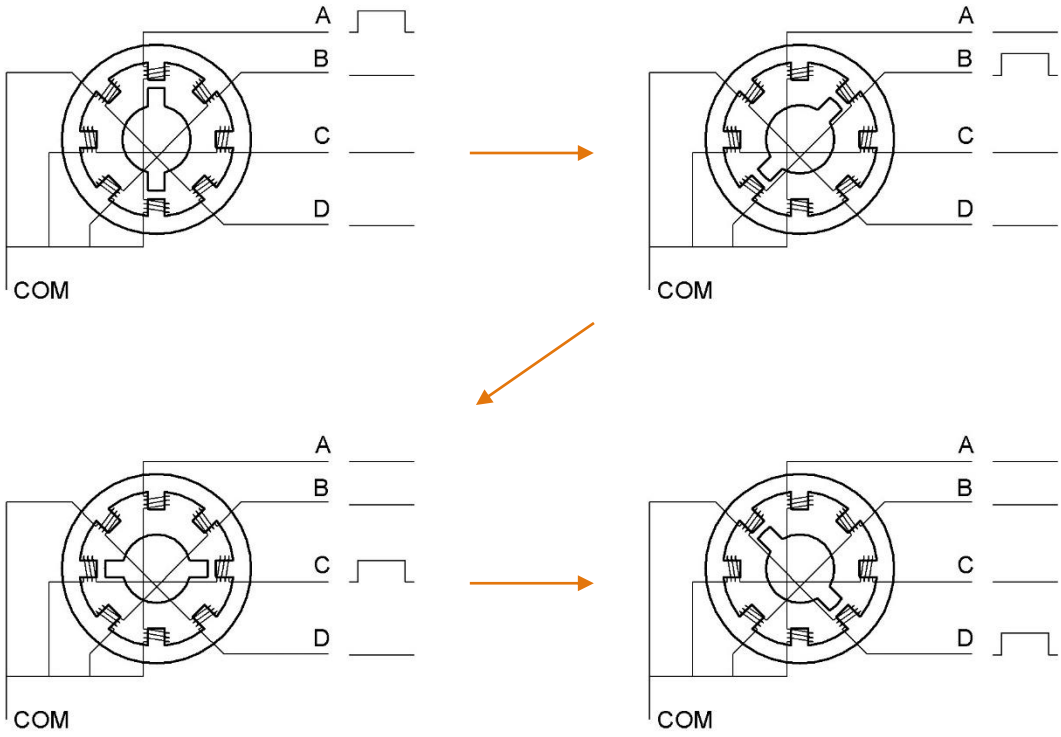


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



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:



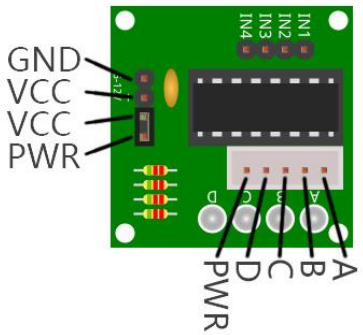
In the sequence above, the Stepper Motor rotates by a certain angle at once, which is called a “step”. By controlling the number of rotational steps, you can then control the Stepper Motor’s rotation angle. By defining the time between two steps, you can control the Stepper Motor’s rotation speed.

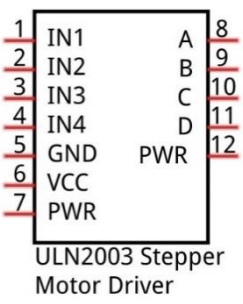
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. If you are interested in it, you can learn this 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.

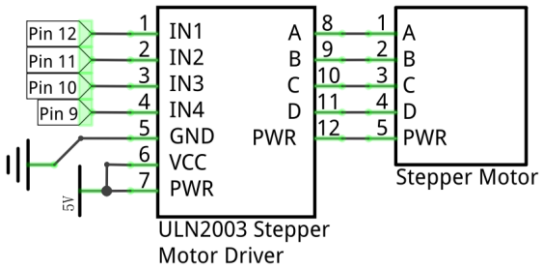




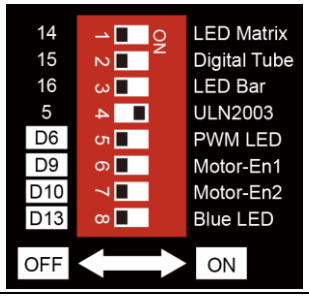
Circuit

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

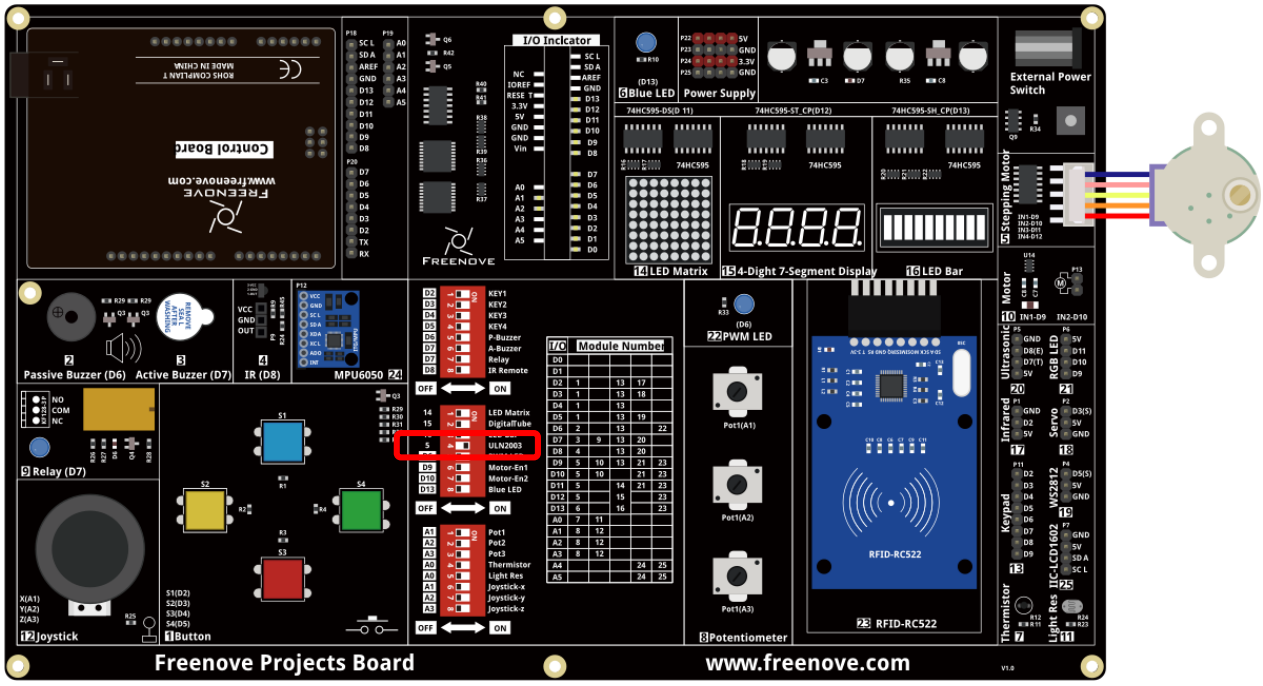
Schematic diagram



Hardware connection



Hardware connection



Sketch

Drive_Stepper_Motor

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

```
// Connect the port of the stepper motor driver
int outPorts[] = {12, 11, 10, 9};

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(outPorts[i], OUTPUT); // set pins to 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;
  }
  for (int i = 0; i < 4; i++) {// Output singal to each port
    digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
  }
}
```

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.

```
23  void moveOneStep(bool dir) {
…   …
36  }
```

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.

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

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

```
33    for (int i = 0; i < 4; i++) { // Output signal to each port
34      digitalWrite(outPorts[i], (out & (0x01 << i)) ? HIGH : LOW);
35    }
```

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

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

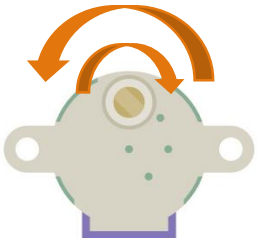
? : operator

"? :" 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.

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 20 Matrix Keypad

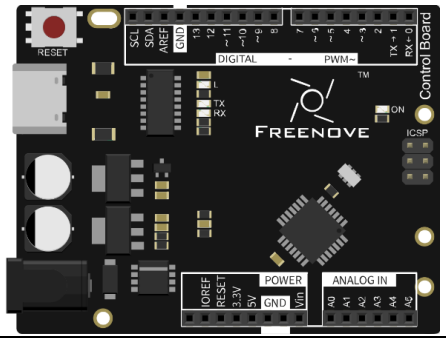
We've already learned and used a push button switch before, now let us try to use a keypad, a device integrated with a number of Push Button Switches as Keys for the purposes of Input.

Project 20.1 Get Input Characters

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

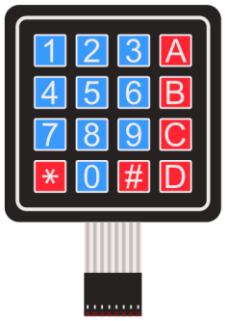
Component List

Control board x1

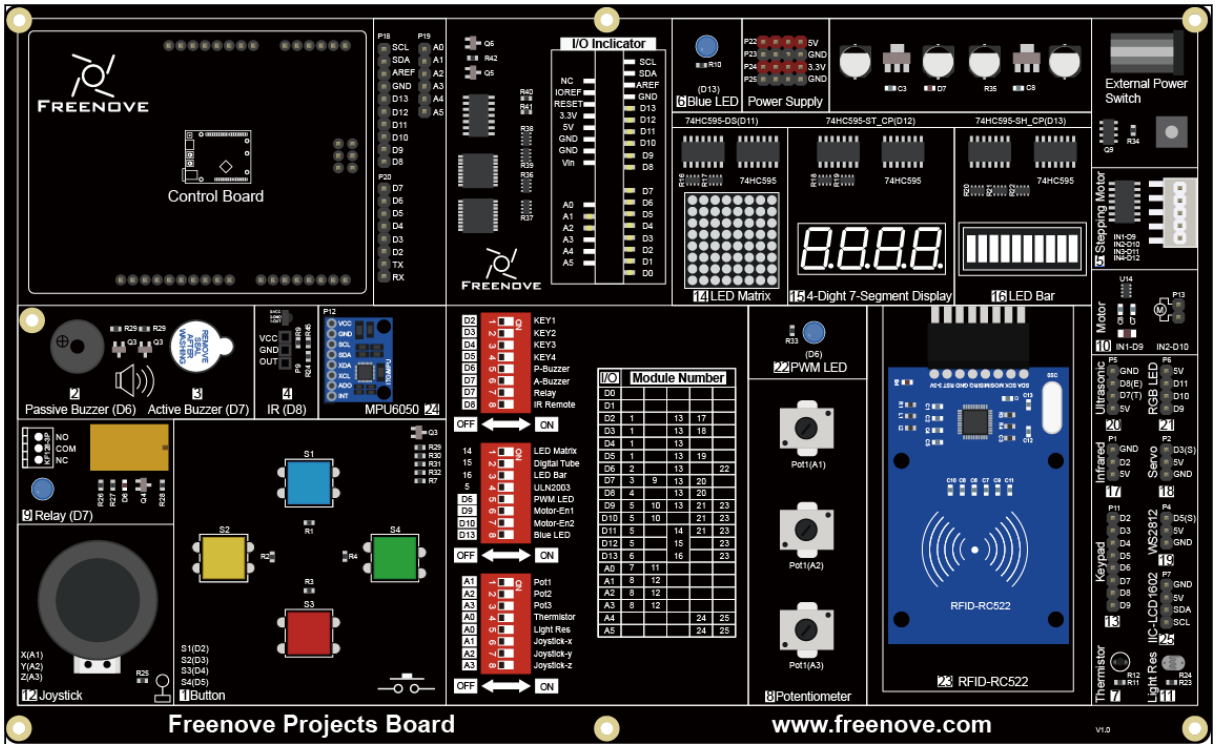


USB cable x1

Keypad x1



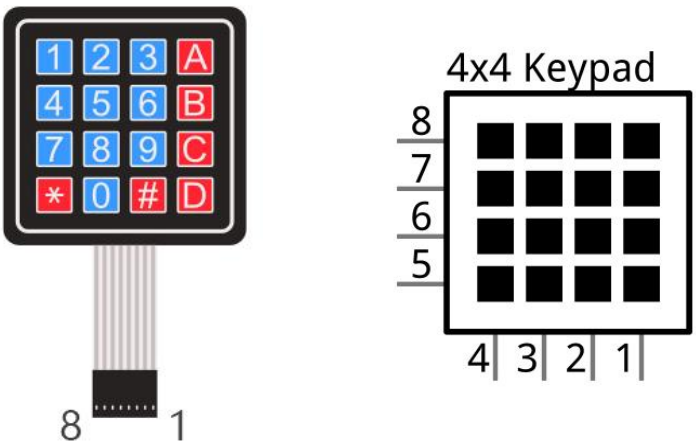
Freenove Projects Board



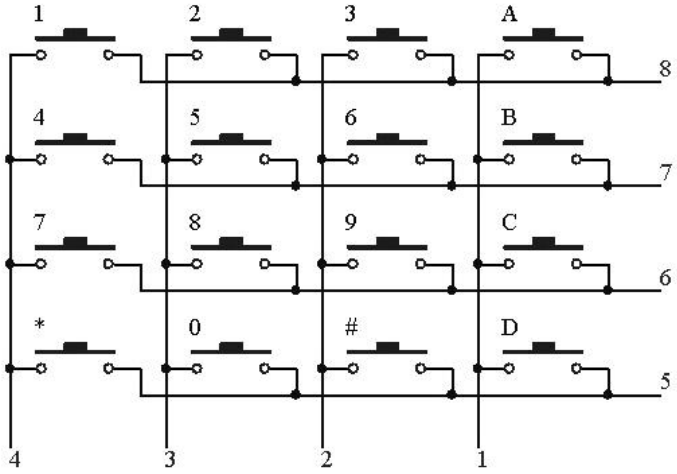
Component Knowledge

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.

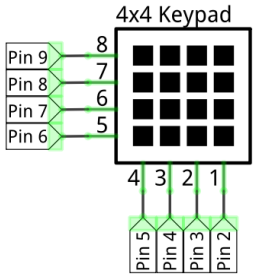


The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key 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.

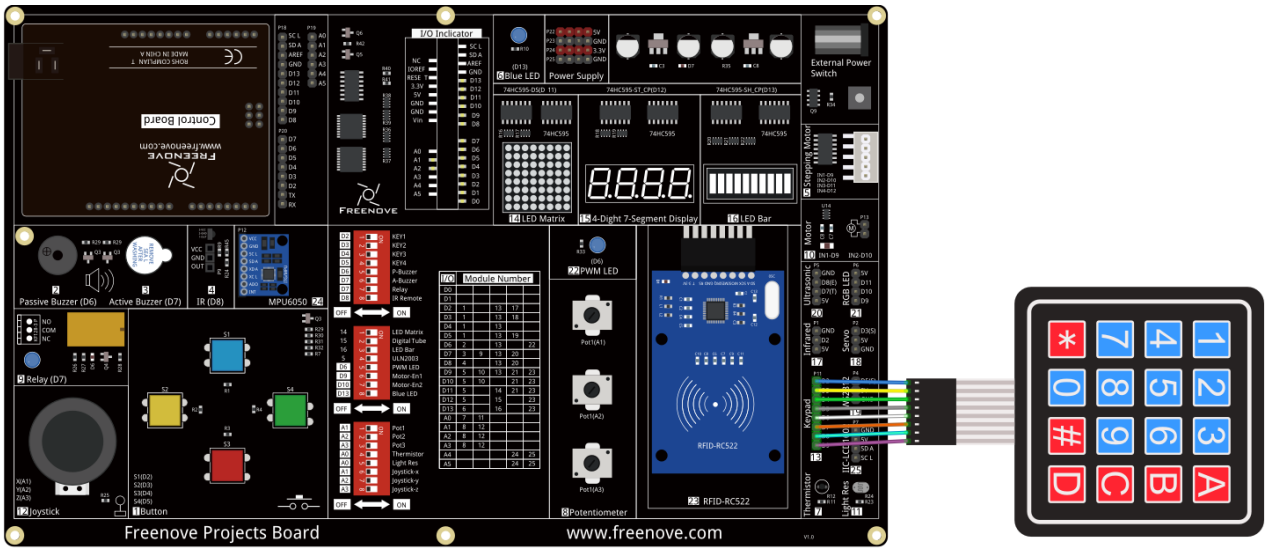
Circuit

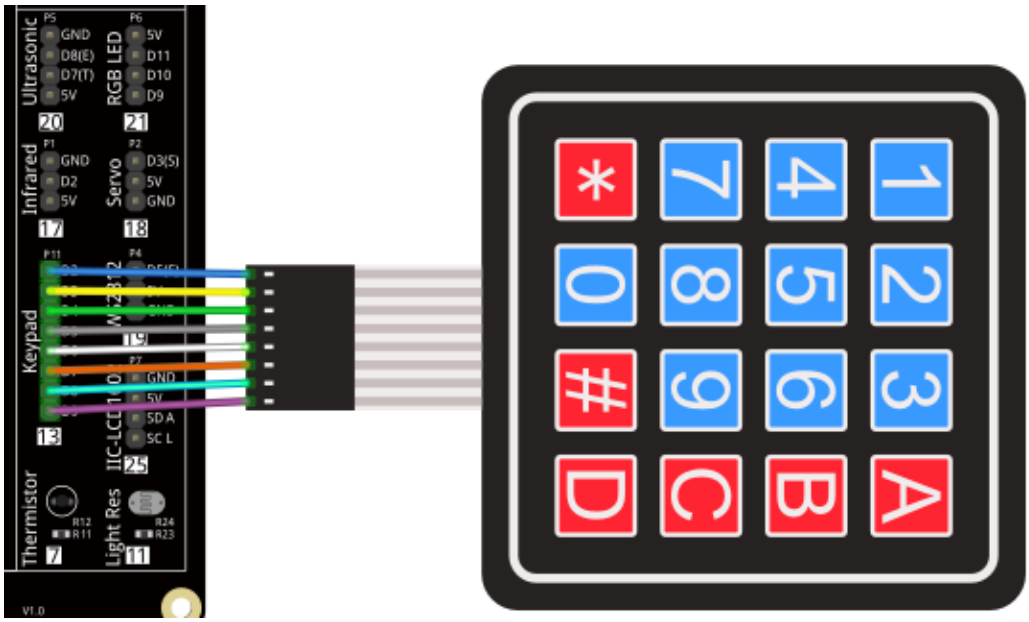
Use pin 9-2 on control board to connect 4x4 keypad.

Schematic diagram



Hardware connection





Sketch

Get_Input_Characters

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] = {2, 3, 4, 5}; // connect to the row pinouts of the keypad
byte colPins[4] = {6, 7, 8, 9};   // 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  };
```

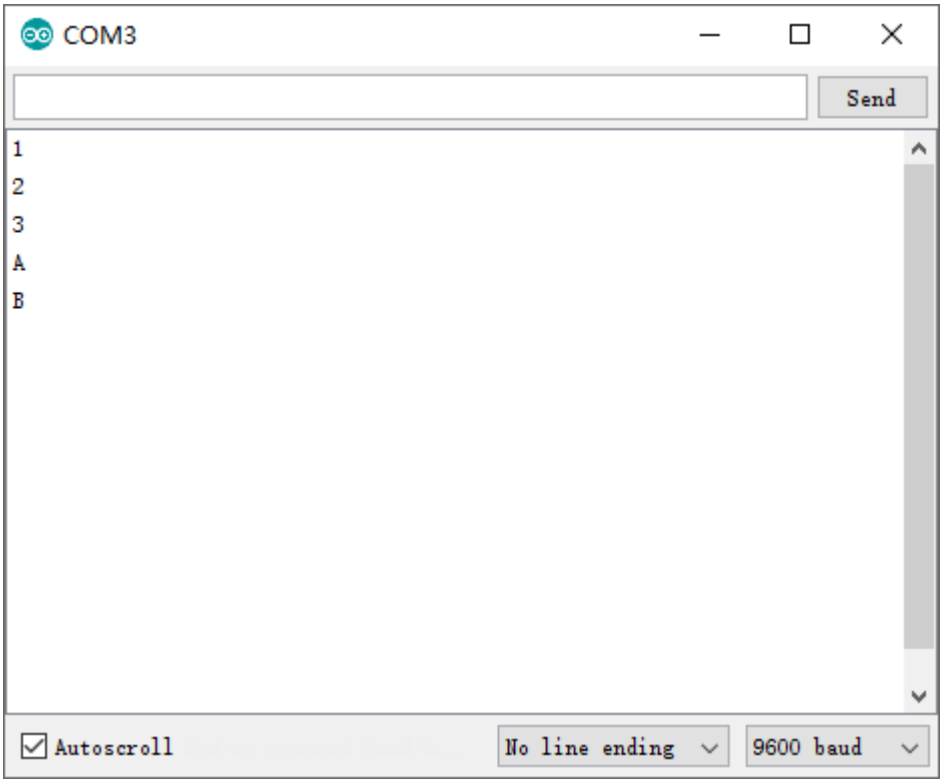
These two arrays record the row and column's connection pins of keypad.

```
11 byte rowPins[4] = {2, 3, 4, 5}; // connect to the row pinouts of the keypad
12 byte colPins[4] = {6, 7, 8, 9};   // connect to the column pinouts of the keypad
```

Send the input obtained 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.



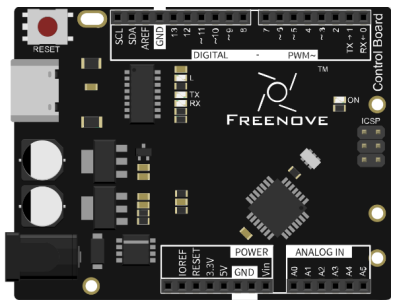
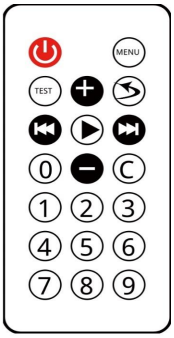
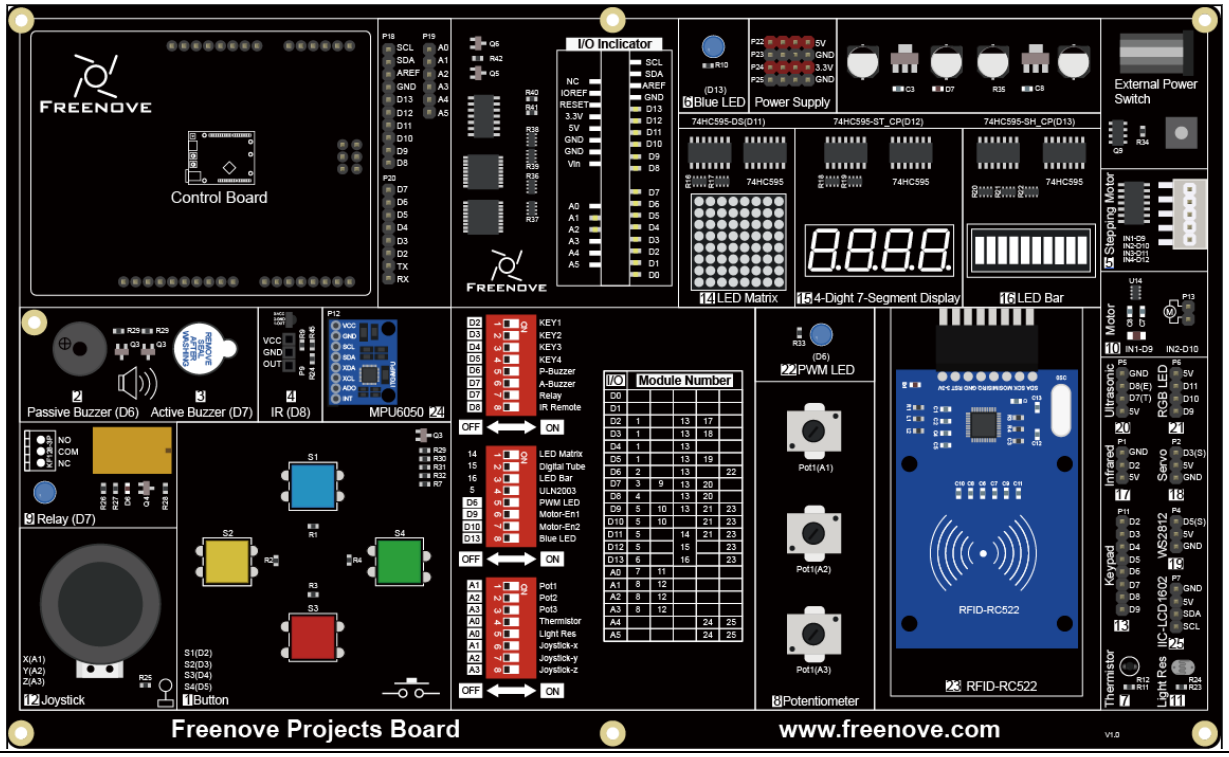
Chapter 21 Infrared Remote

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.

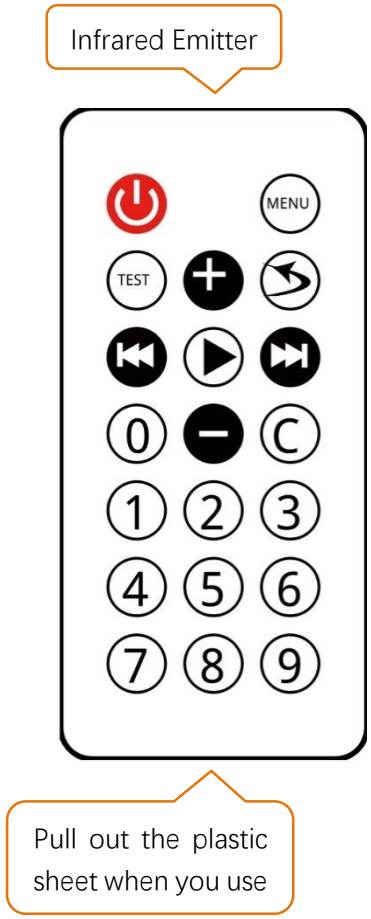
Component List

Control board x1	USB cable x1	Infrared receiver x1	Infrared remote control x1
			
Freenove Projects Board			
			

Component Knowledge

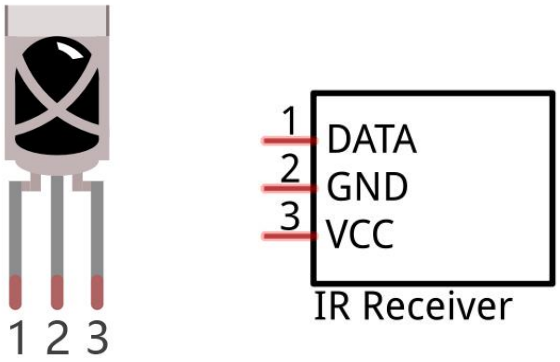
Infrared remote

An Infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emitting tube, which is located on the front of the remote control, send infrared signals 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 conditioner that are far from you. The remote control we use is shown below:



Infrared receiver

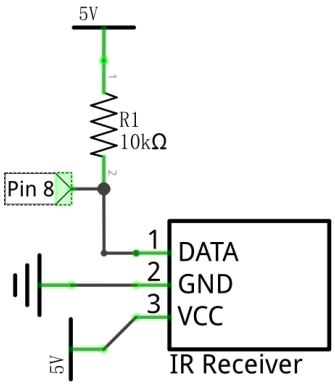
Infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.



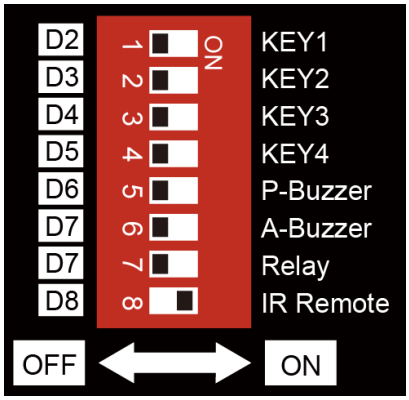
Circuit

Use pin 8 on the control board to connect IR receiver.

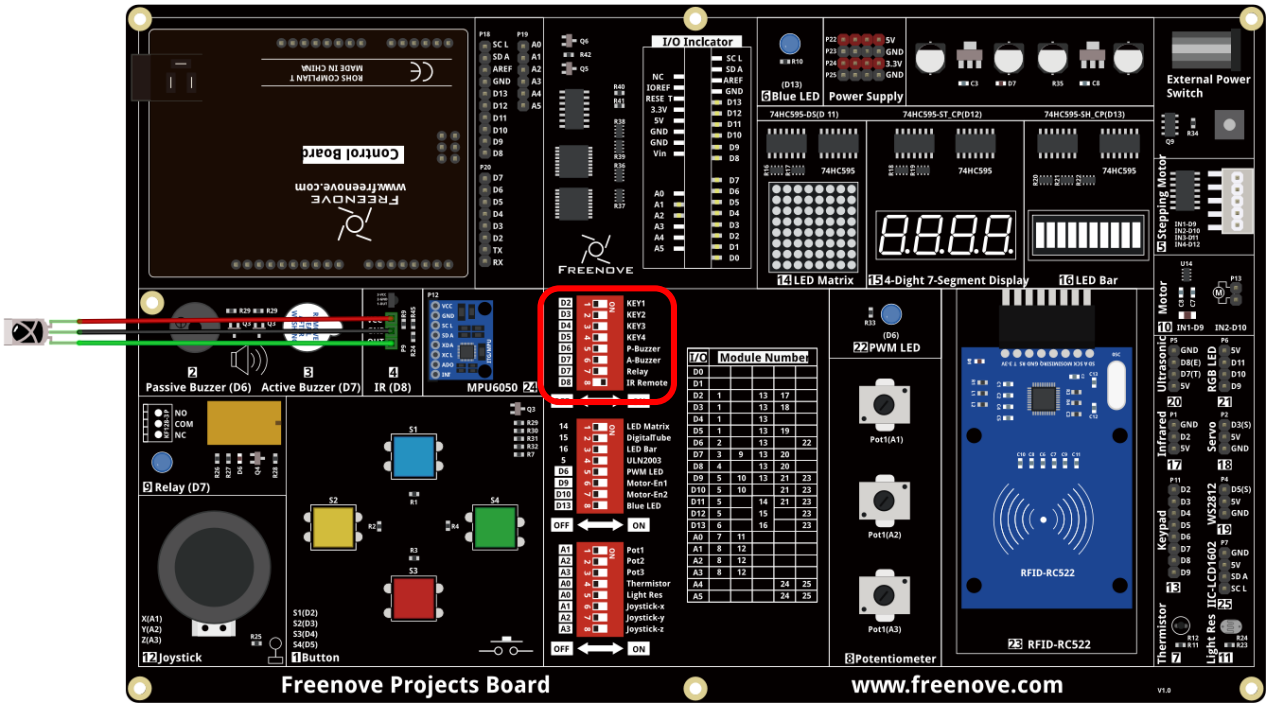
Schematic diagram



Hardware connection



Hardware connection



Connect the infrared receiver to the IR interface on the control board.

Note: the head of the receiver should orient to the right.

Sketch

Infrared_Remote_Control

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.

```
1  #include <IRremote.h>
2
3  int RECV_PIN = 8;         // Infrared receiving pin
4  IRrecv irrecv(RECV_PIN);  // Create a class object used to receive class
5  decode_results results;   // Create a decoding results class object
6
7  void setup()
8  {
9    Serial.begin(9600); // Initialize the serial port and set the baud rate to 9600
10   irrecv.enableIRIn(); // Start the receiver
11 }
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
```

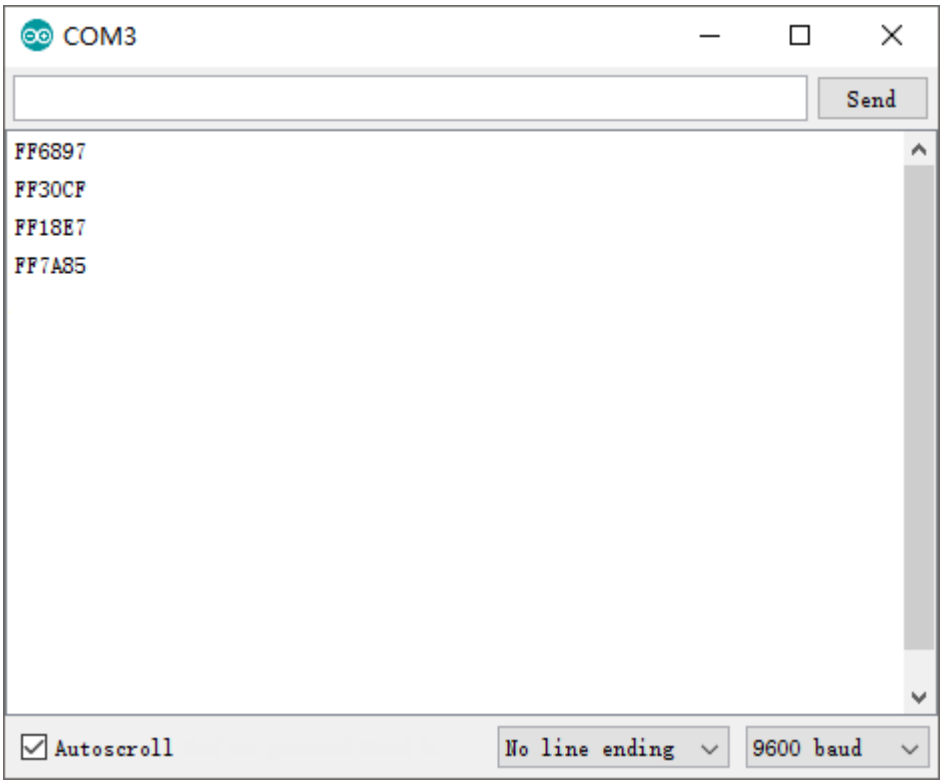
Start the signal receiving in the setup() function

```
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 }
```

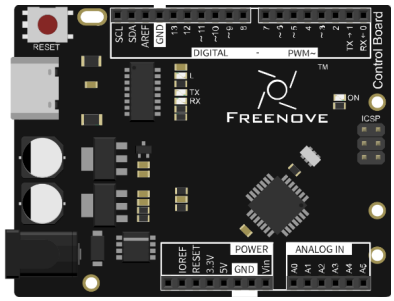
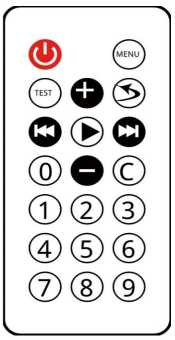
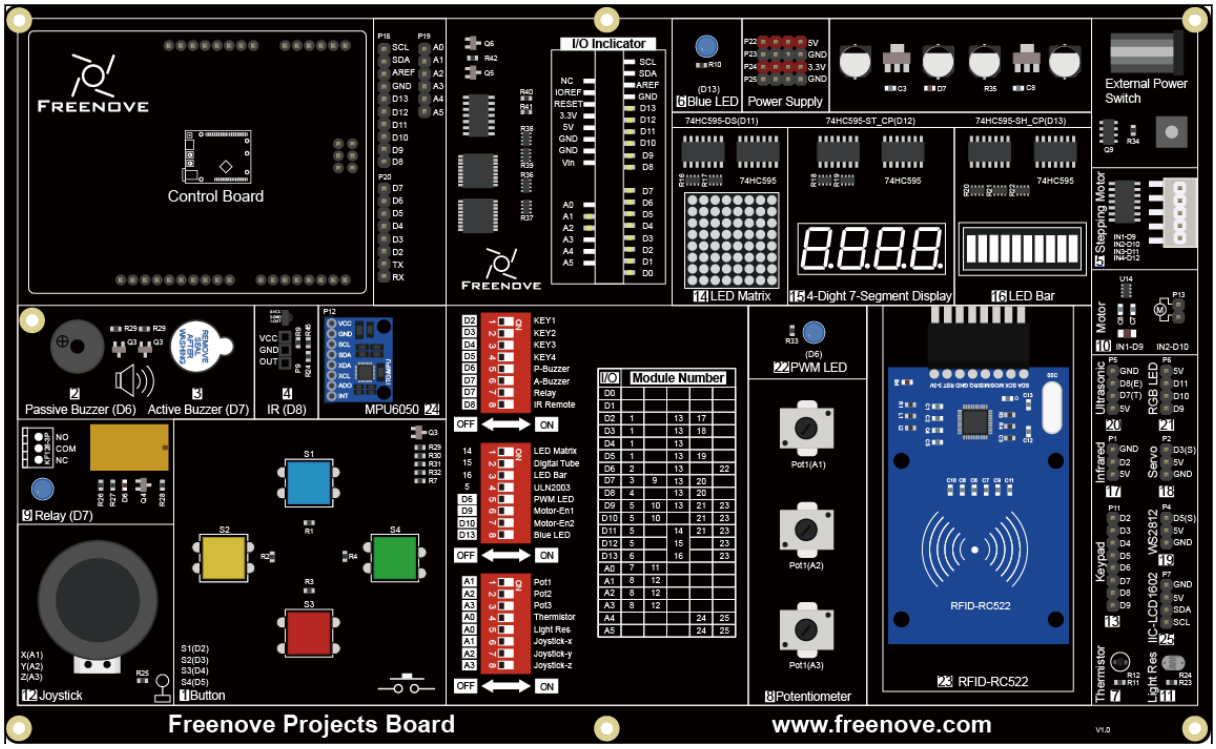
Verify and upload the code, open the Serial Monitor, and press the IR control button, and then you can see the corresponding code being printed out.



Project 21.2 Control LED through Infrared Remote

Now, let us try to control anLED through infrared remote.

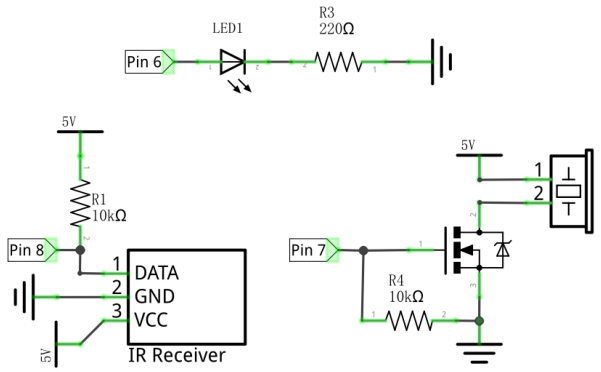
Component List

<table>
<tr>
<td>Control board x1
  </td>
<td>USB cable x1
  </td>
<td>Infrared receiver x1
  </td>
<td>Infrared remote control x1
  </td>
</tr>
<tr>
<td colspan="4">Freenove Projects Board
  </td>
</tr>
</table>

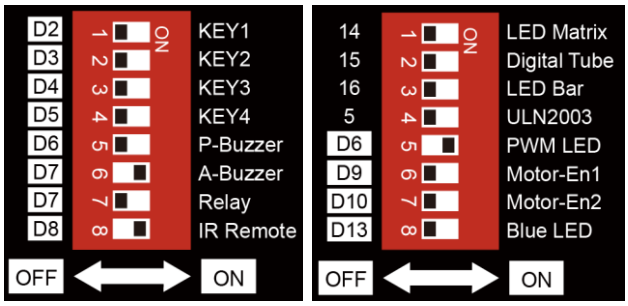
Circuit

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.

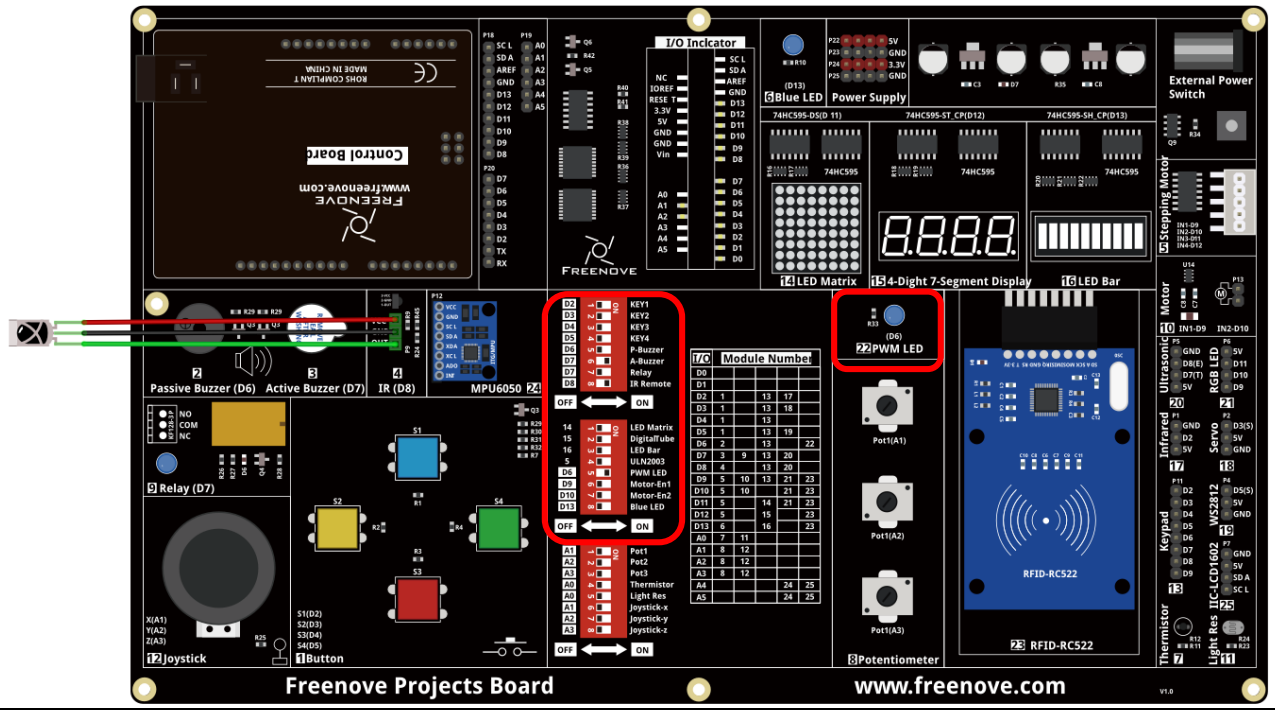
Schematic diagram



Hardware connection



Hardware connection



Sketch

Control_LED_through_Infrared_Remote

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 = 8;          // 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 = 6;            // the number of the LED pin
int buzzerPin = 7;         // 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  }
```

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

We define a function 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.

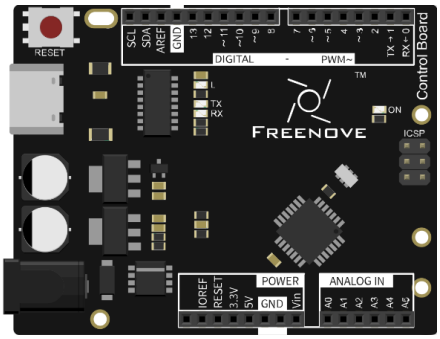
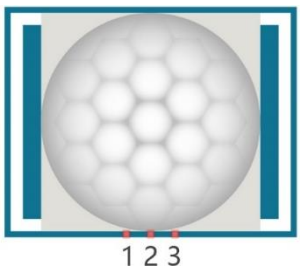
Chapter 22 Infrared Motion Sensor

This Infrared MotionSensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals. Now let us try to use it.

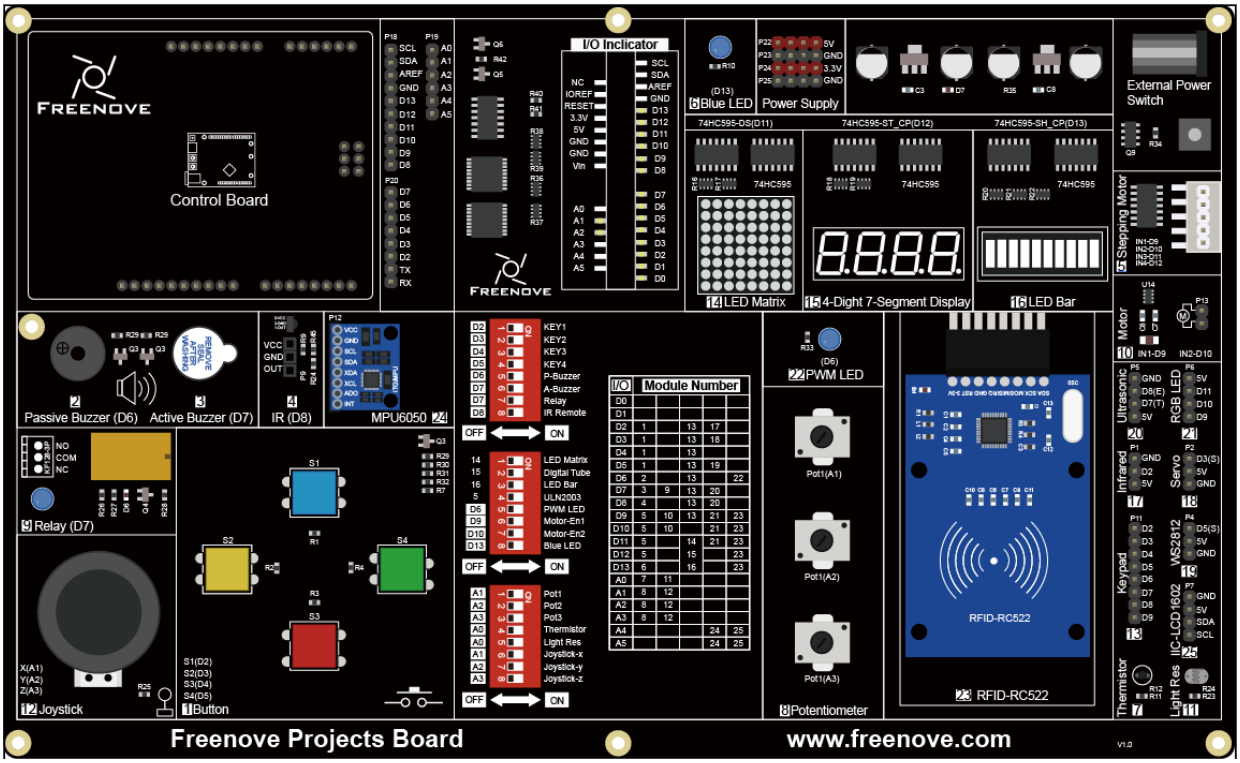
Project 22.1 Infrared Motion Sensor

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

Component List

Control board x1 	USB cable x1 
Jumper Wire x4 	Infrared Motion Sensor x1 

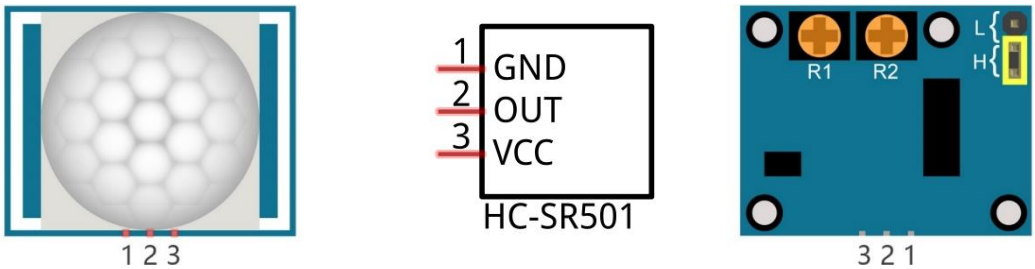
Freenove Projects Board



Component Knowledge

Infrared Motion Sensor

Infrared Motion Sensor is an integrated sensor that can detect the infrared spectrum (heat signatures) emitted by living humans and animals. It can detect whether someone is moving within the detecting range by detecting the infrared infrared spectrum (heat signatures) 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.

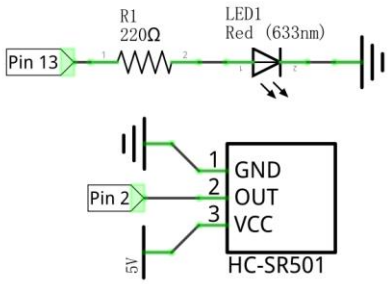
4. Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level.
5. Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, 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.

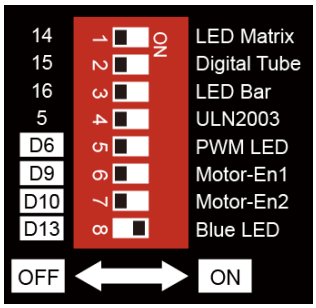
Circuit

Use pin 2 on the control board to connect out-pin of HC-SR501 infrared motion sensor.

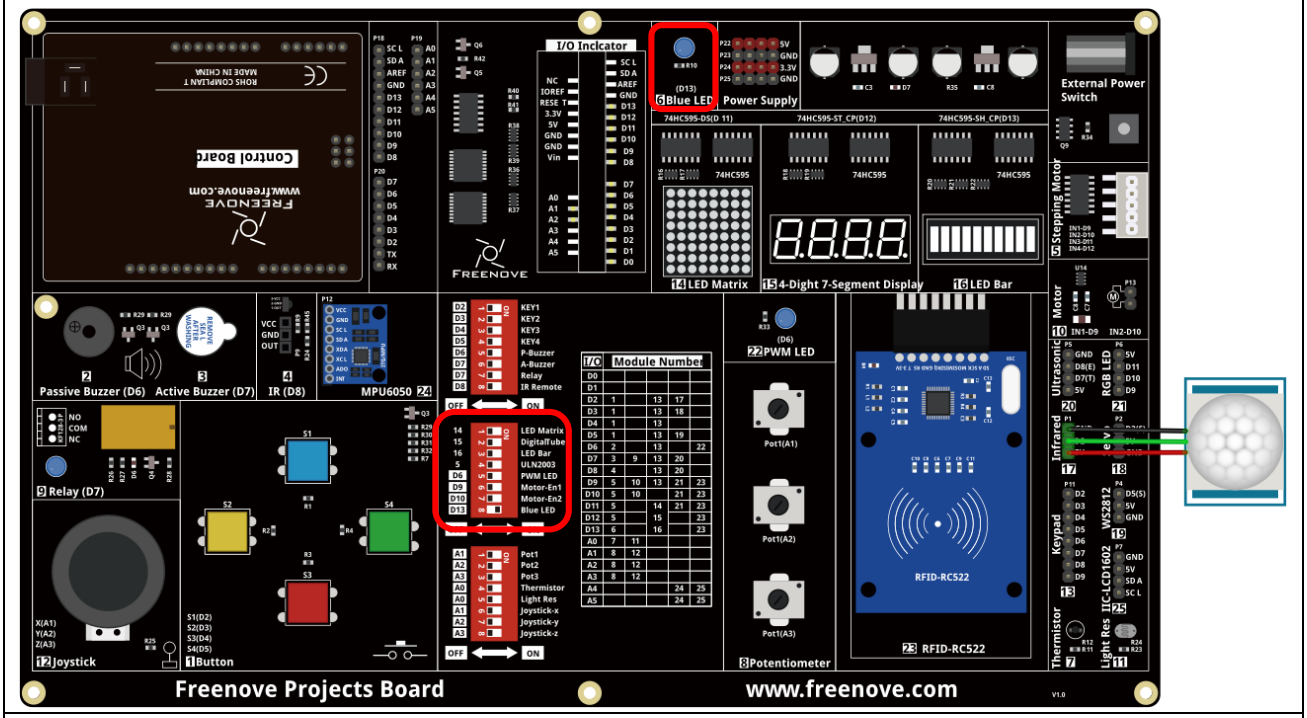
Schematic diagram

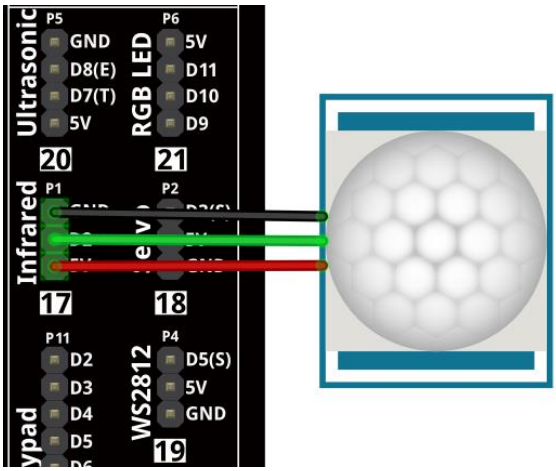


Hardware connection



Hardware connection





Sketch

Infrared_Motion_Sensor

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

```
1  int sensorPin = 2; // the number of the infrared motion sensor pin
2  int ledPin = 13;    // the number of the LED pin
3
4  void setup() {
5    pinMode(sensorPin, INPUT);  // initialize the sensor pin as input
6    pinMode(ledPin, OUTPUT);    // initialize the LED pin as output
7  }
8
9  void loop() {
10    // Turn on or off LED according to Infrared Motion Sensor
11   digitalWrite(ledPin, digitalRead(sensorPin));
12   delay(100);              // wait for a second
13 }
```

This code is relatively simple. We have obtained the sensor's output signal, and control an LED accordingly.

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

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

You can rotate the potentiometer on the sensor to adjust the detection effect, or use different modes by 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.

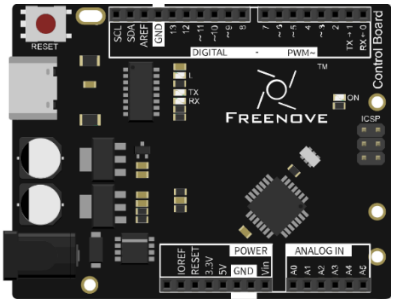
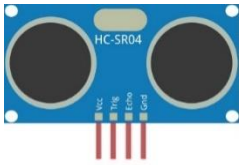
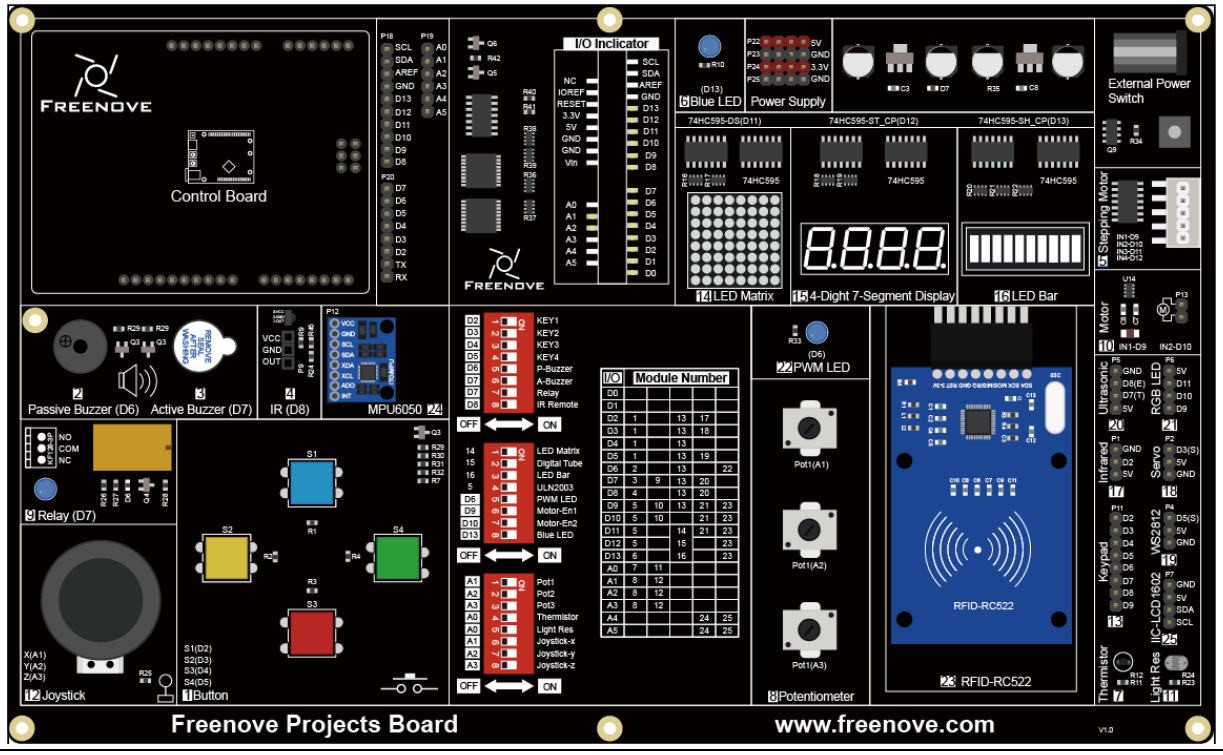
Chapter 23 Ultrasonic Ranging

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

Project 23 Ultrasonic Ranging

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

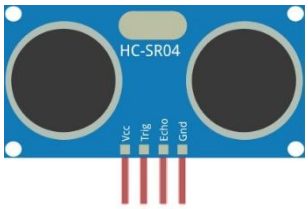
Component List

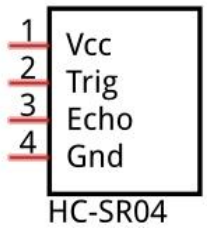
Control board x1 	USB cable x1 	Ultrasonic module x1 
Jumper Wire x4 		
Freenove Projects Board 		

Component Knowledge

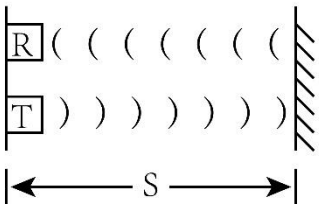
Ultrasonic ranging module

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





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



Pin description:

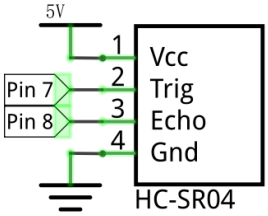
Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Triger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

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.

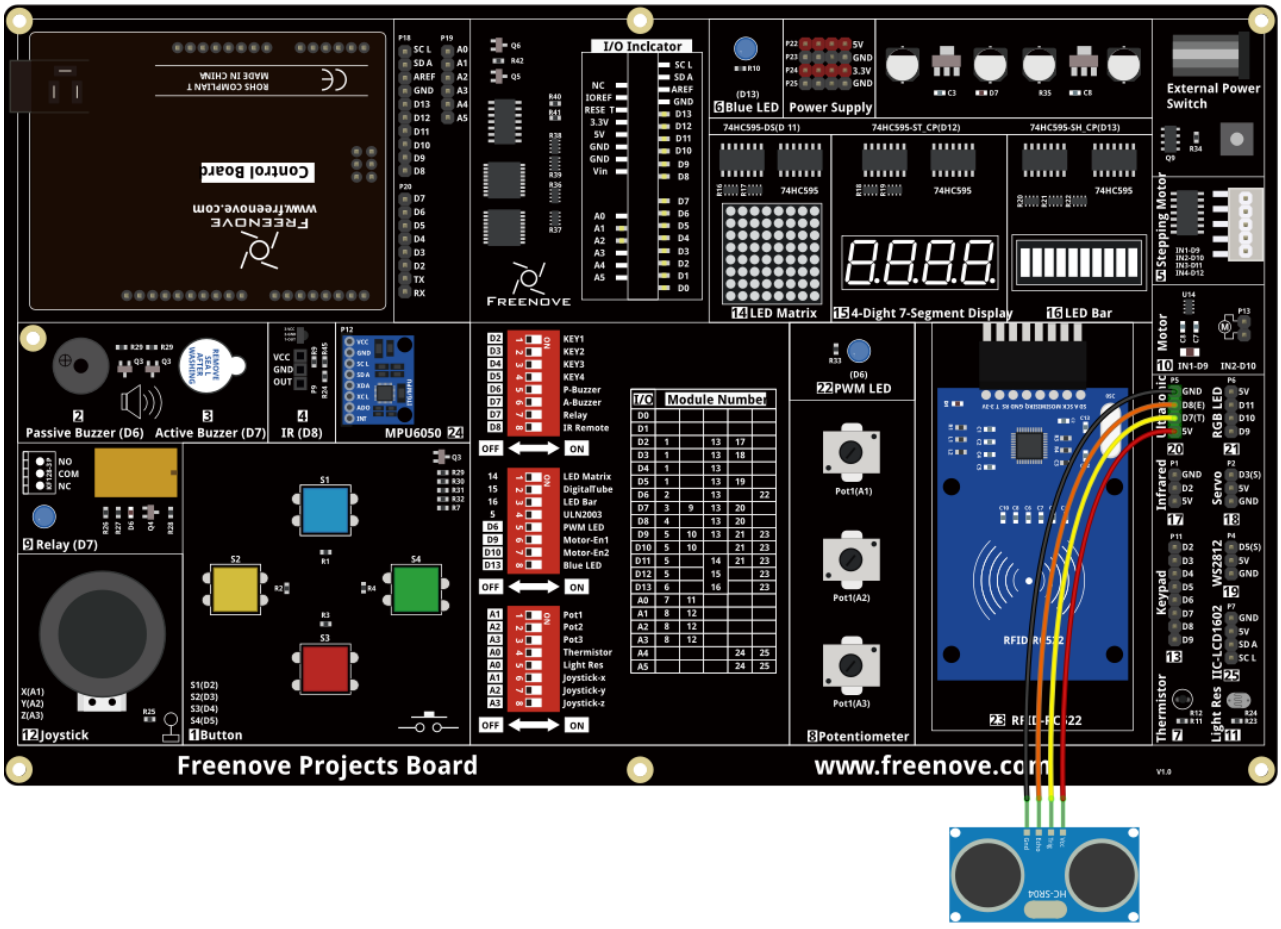
Circuit

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

Schematic diagram



Hardware connection



Sketch

Ultrasonic_Ranging

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

```
#define trigPin 7 // define TrigPin
#define echoPin 8 // 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 7 // define TrigPin
#define echoPin 8 // 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. timeOut= 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.

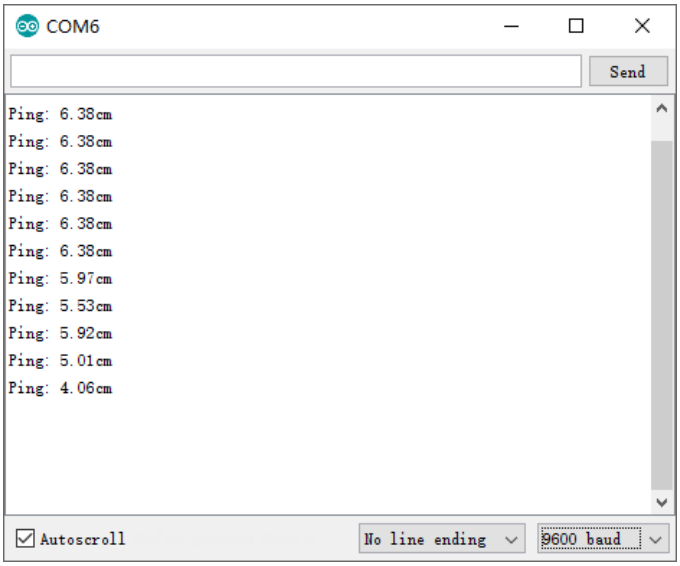
```
  distance = (float)pingTime * soundVelocity / 2 / 10000; // calculate the distance according to the time
  return distance; // return the distance value
```

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.



Ultrasonic_Ranging

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 7 // define TrigPin
3  #define echoPin 8 // 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 7 // define TrigPin
3  #define echoPin 8 // 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:

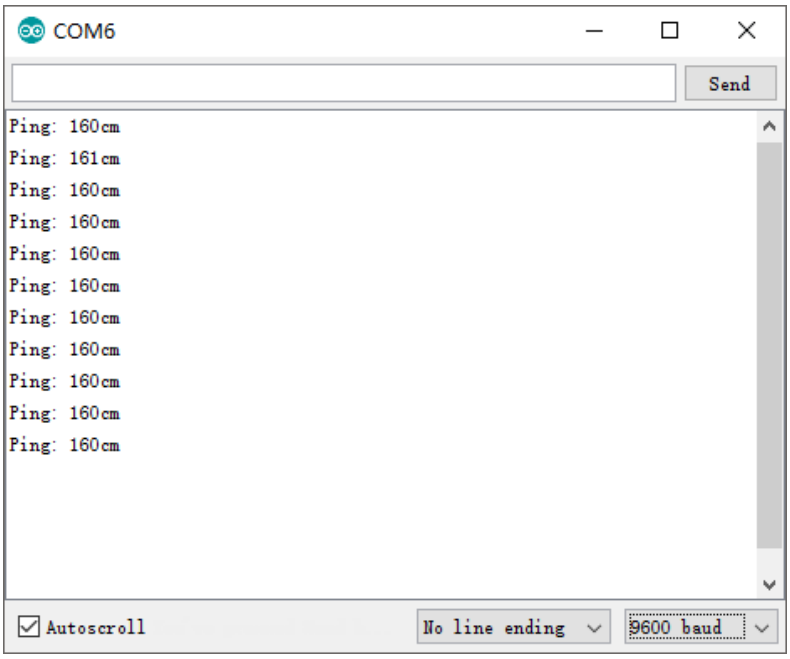
sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

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 the ultrasonic probe aims at a flat object, the distance between the two can be measured.



Chapter 24 RFID

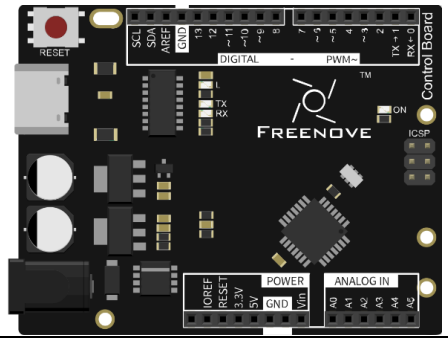
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 24.1 Read UID

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

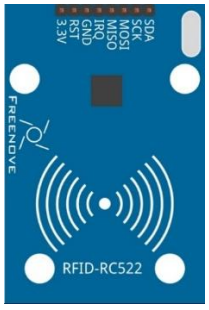
Component List

Control board x1

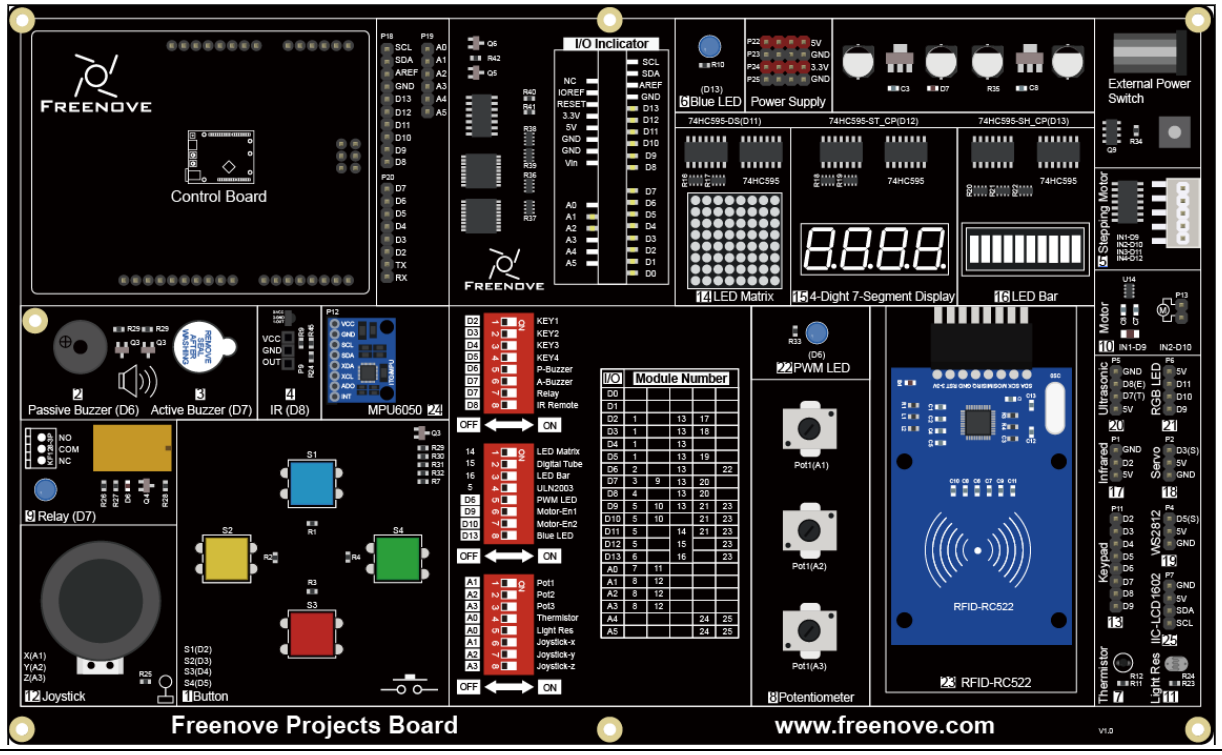


USB cable x1

RFID-RC522 x1



Freenove Projects Board



Circuit knowledge

RFID

RFID (Radio Frequency Identification) is a wireless communication technology. A complete RFID system is generally composed of the responder and reader. Generally, we use tags as responders, and each tag has a unique code, which is attached to the object to identify the target object. The reader is a device for reading (or writing) tag information.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products. And Passive RFID products are the earliest, the most mature and most widely used products in the market among others. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of these belong to close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

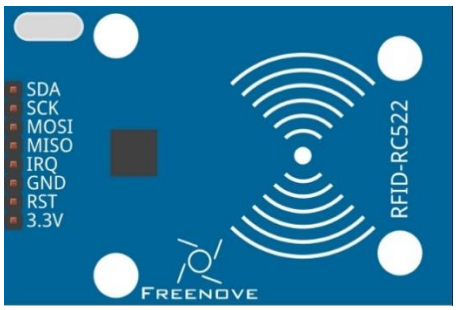
Component Knowledge

MFRC522 RFID Module

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

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.

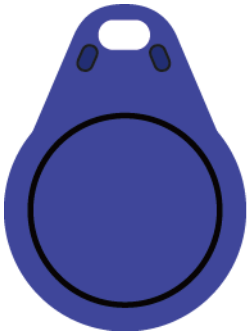
This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.



Mifare1 S50 Card

Mifare1 S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for this kit are shown below.

Mifare1 S50 Standard card



Mifare1 S50 Non-standard card

The Mifare1 S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63). And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 0	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control which are put in the last block of each sector, and the block is also known as sector trailer, that is Block 3 in each sector. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the vendor code, which has been solidified and can't be changed, and the card serial number is stored here. In addition to the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:
(1) used as general data storage and can be operated for reading and writing.
(2) used as data value, and can be operated for initializing the value, adding value, subtracting and reading the value.
The sector trailer block in each sector is the control block, including a 6-byte password A, 4-byte access control and 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally A0A1A2A3A4A5 for password A, B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

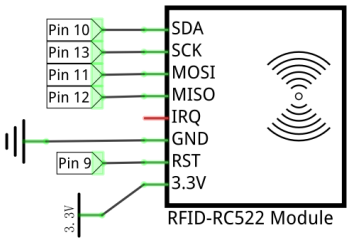
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read. If you choose to verify password A and then you forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

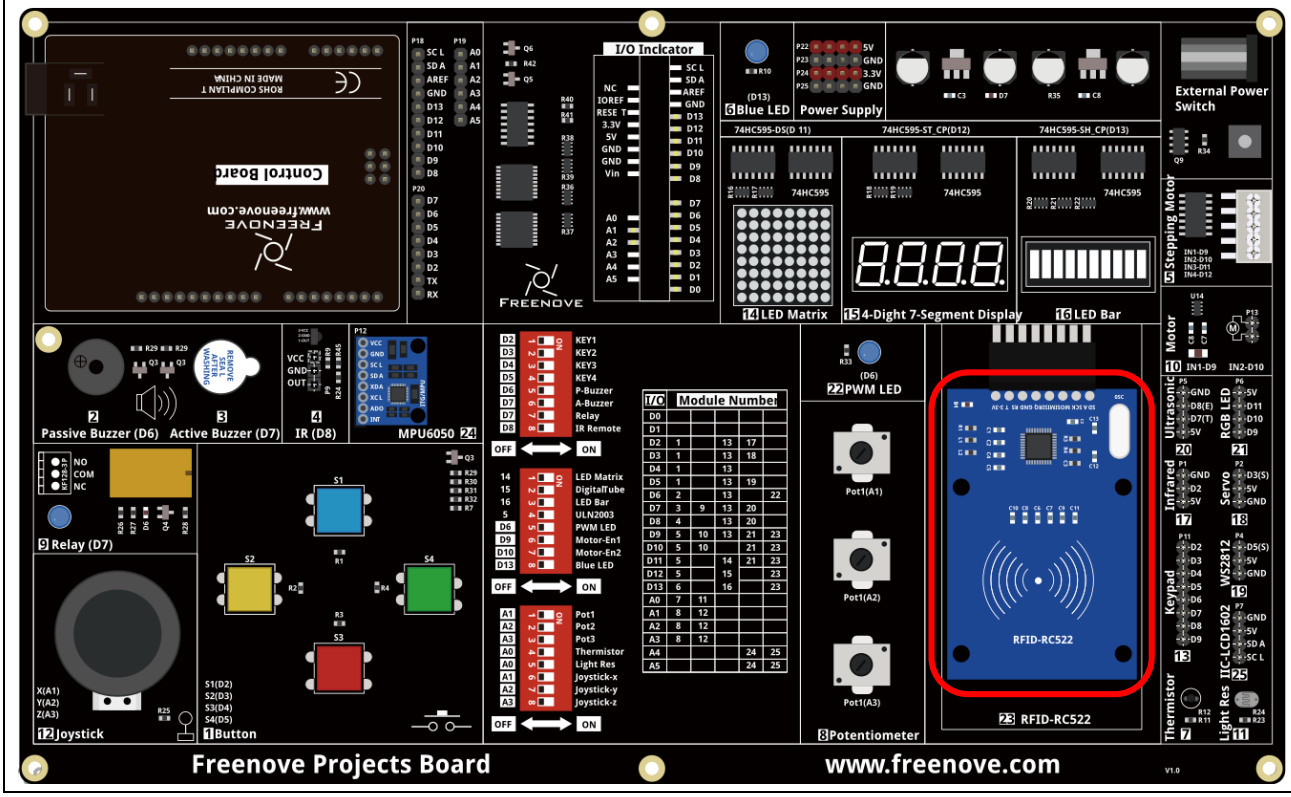
Circuit

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

Schematic diagram



Hardware connection



Sketch

RFID_Read_UID

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

Click “Add .ZIP Library...” and then find **RFID.zip** in libraries folder (this folder is in the folder unzipped form the ZIP file we provided). This library makes it easy to operate the RFID module.

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

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

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

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

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

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

After including the RFID library, we need to construct an RFID class object before using the function in RFID library. Its constructor needs to be written to two pins, respectively to the SDA pin and the RST pin.

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

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

```
11    Serial.begin(9600);
12    SPI.begin();
13    rfid.init(); //initialization
```

In loop(), use findCard() waiting for the card approaching. Once it detects the card, this function will return MI_OK and save the card type data in parameter str and then enter the if statement.

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

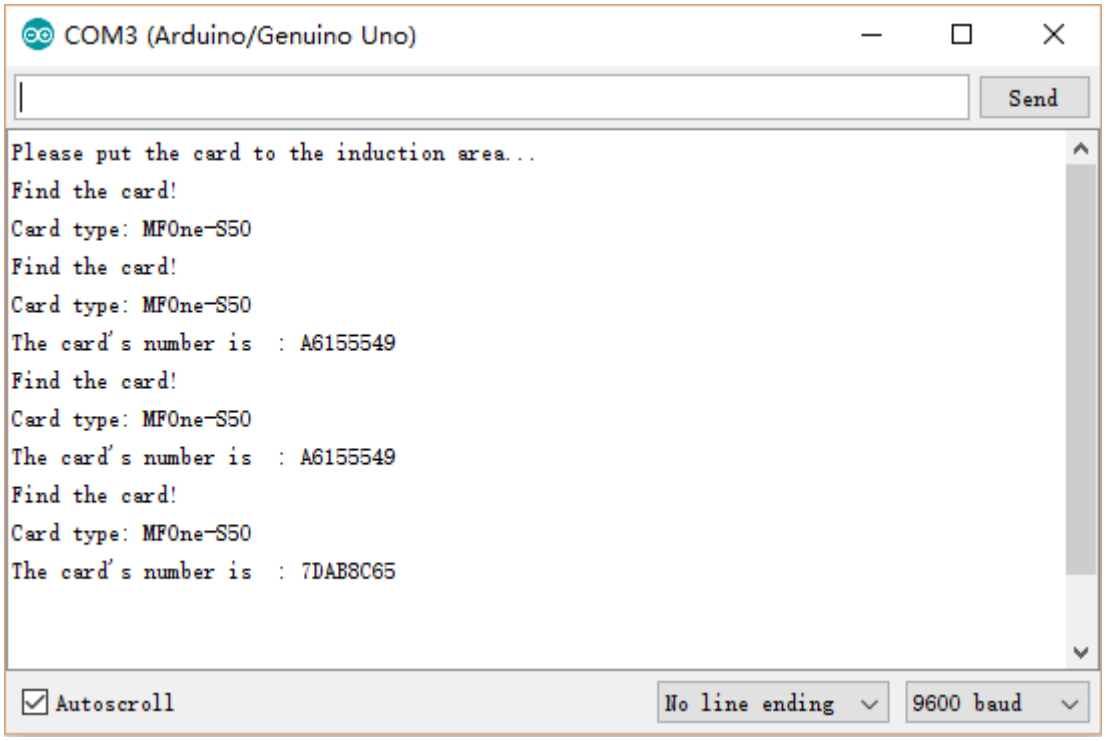
After entering if statement, call the sub function ShowCardType(). Then determine the type of the card according to the content of STR and print the type out through the serial port.

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

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

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

After verifying and uploading the code to control board, open the serial port monitor and place a card on the sensing area of RFID module. Then serial port monitoring window will display the displacement ID number and the type of the card. If the induction time is too short, it may lead to incomplete-information display.



Project 24.2 Read and write

In this project, we will read and write a card.

Component list

Same with last section.

Circuit

Same with last section.

Sketch

RFID_Read_And_Write

In this sketch, first read the data in particular location of the S50 M1 Card, then write data in that position and read it. Display these data through the serial port.

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

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

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

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

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

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

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

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

```
67
68    rfid.halt();
69  }
70
71  //write the card
72  void writeCard(int blockAddr) {
73    if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
    MI_OK) //authenticate
74    {
75      //write data
76      //status = rfid.write((blockAddr/4 + 3*(blockAddr/4+1)), sectorKeyA[0]);
77      Serial.print("set the new card password, and can modify the data of the Sector: ");
78      Serial.println(blockAddr / 4, DEC);
79      // select block of the sector to write data
80      if (rfid.write(blockAddr, writeDate) == MI_OK) {
81        Serial.println("Write card OK!");
82      }
83    }
84  }
85
86  //read the card
87  void readCard(int blockAddr) {
88    if ( rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
    MI_OK) // authenticate
89    {
90      // select a block of the sector to read its data
91      Serial.print("Read from the blockAddr of the card : ");
92      Serial.println(blockAddr, DEC);
93      if ( rfid.read(blockAddr, str) == MI_OK) {
94        Serial.print("The data is (char type display): ");
95        Serial.println((char *)str);
96        Serial.print("The data is (HEX type display): ");
97        for (int i = 0; i < sizeof(str); i++) {
98          Serial.print(str[i], HEX);
99          Serial.print(" ");
100       }
101       Serial.println();
102     }
103   }
104 }
```

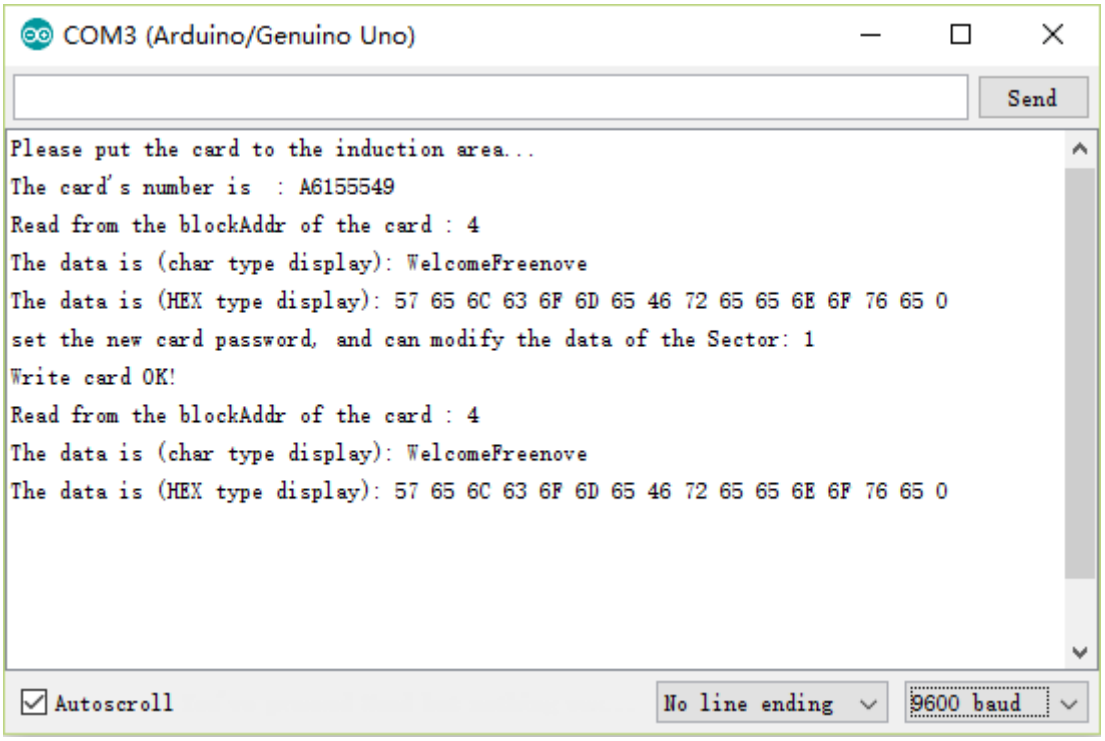
In the sub function of writeCard () and readCard (), we must first verify the password A, and then use the corresponding sub function to read and write. Here we read and wirte data block 0 (absolute NO.4) of the first sector.

```
73    if (rfid.auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr / 4], rfid.serNum) ==
      MI_OK) //authenticate
```

In loop (), compare the contents of the data block NO.4 after written with the original contents.

```
61    //first, read the data of data block 4
62    readCard(4);
63    //write data(within 16 bytes) to data block
64    writeCard(4);
65    //then read the data of data block again
66    readCard(4);
```

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



What's Next?

THANK YOU for participating in this learning experience!

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

support@freenove.com

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

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

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

http://www.freenove.com/

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

Thank you again for choosing Freenove products.

Appendix

ASCII Table

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