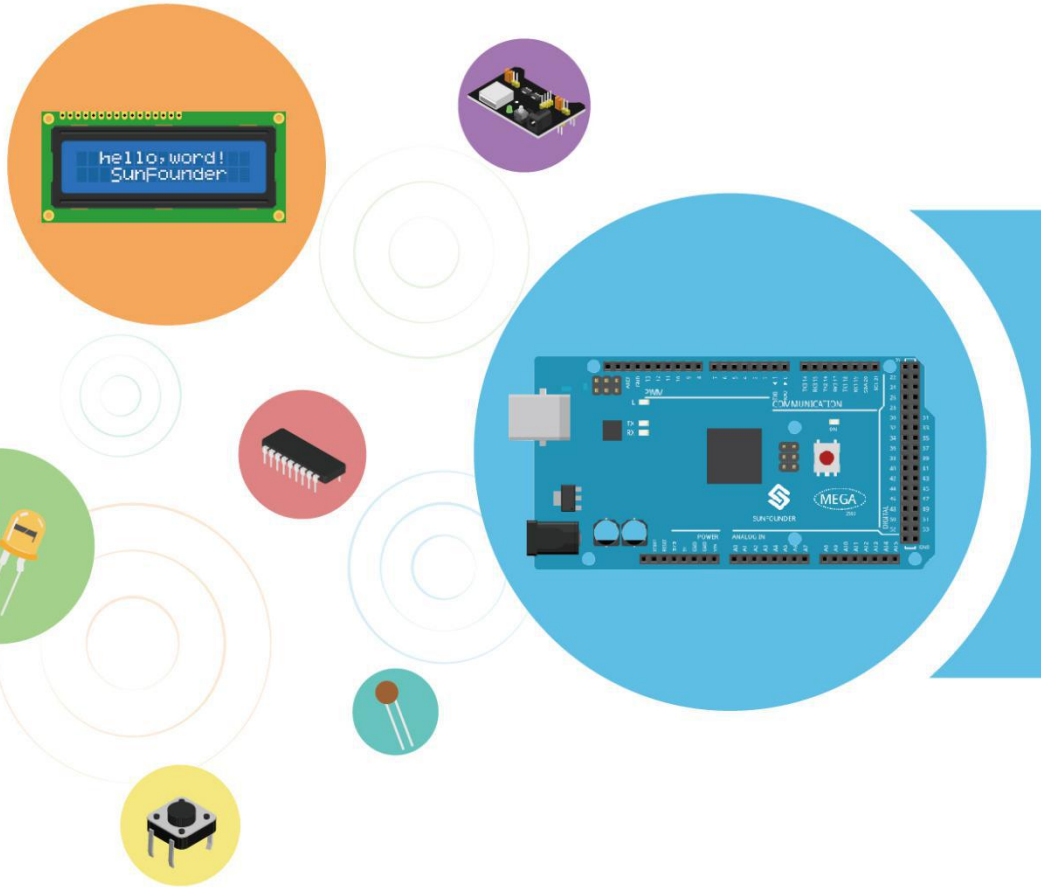
SunFounder
STEAM EDUCATION



SunFounder Mega Kit

Preface

About SunFounder

SunFounder is a company focused on STEM education with products like opensource robots, aircraft models, and smart devices distributed globally.In SunFounder, we strive to help elementary and middle school students as well as hobbyists, through STEM education, strengthen their hands-on practices and problem-solving abilities. In this way, we hope to disseminate knowledge and provide skill training in a full-of-joy way, thus fostering your interest in programming and making, and exposing you to a fascinating world of science and engineering. To embrace the future of artificial intelligence, it is urgent and meaningful to learn abundant STEM knowledge.

About the Mega Kit

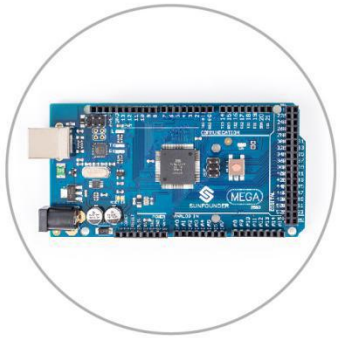
With this kit, we will walk you through the know-how of using the SunFounder board in a hands-on way. Starting with the basics of electronics, you'll learn through building several creative projects. Including a selection of the most common and useful electronic components, this kit will help you "control" the physical world.

Free Support

If you have any **TECHNICAL questions**, add a topic under **FORUM** section on our website and we'll reply as soon as possible.

For **NON-TECH questions** like order and shipment issues, please **send an email to service@sunfounder.com**.
You're also welcomed to share your projects on FORUM

Components list



Mega 2560 Controller Board

1 PCS

5v Relay

1 PCS

Remote Control

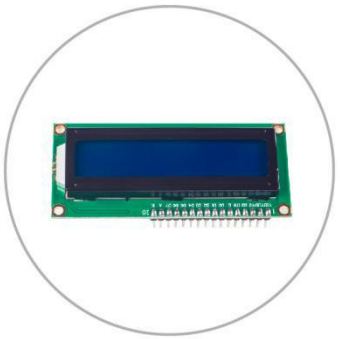
1 PCS

Servo Motor (SG90)

1 PCS

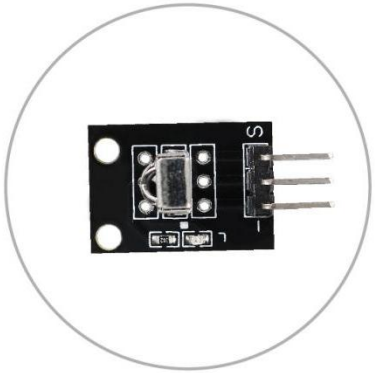
USB Cable

1 PCS



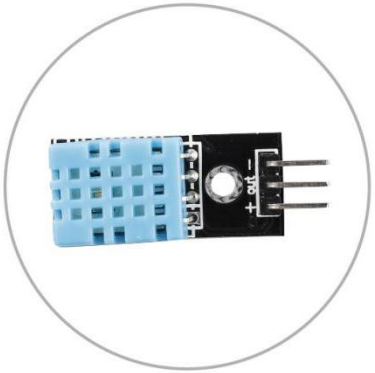
16*2 LCD Character Display

1 PCS



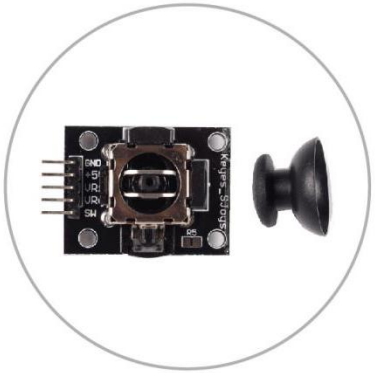
Infrared Receiver

1 PCS



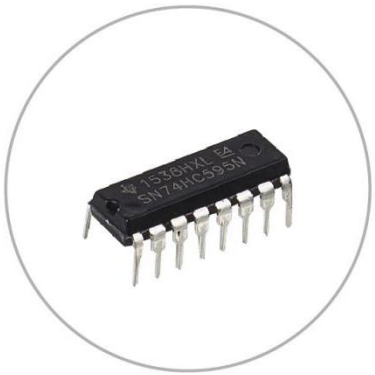
Humiture Sensor

1 PCS



Joystick PS2

1 PCS



74HC595

1 PCS

L293d

1 PCS

Button (small)

5 PCS

power supply module

1 PCS

Stepper Motor Driver

1 PCS

Active Buzzer

1 PCS

4-digit 7-segment Display

1 PCS

One-digit Segment Display

1 PCS

Stepper Motor

1 PCS

Jumper Wire

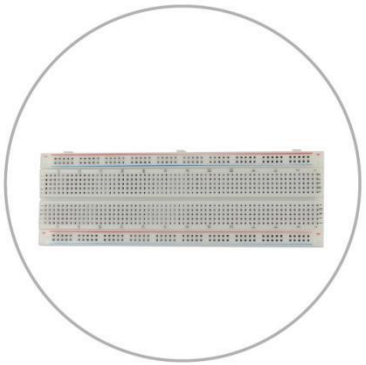
65 PCS

9V Battery Clip Connector

1 PCS

Dupont Wire

10 PCS



Breadboard

1 PCS

LED (Red)

5 PCS

LED (Green)

5 PCS

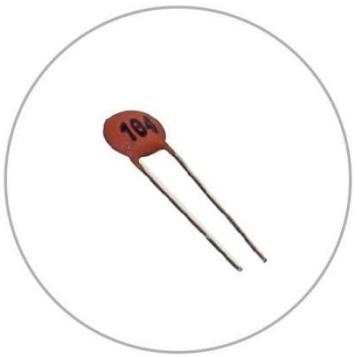
LED (Yellow)
5 PCS

LED (White)
5 PCS

LED(Blue)
5 PCS

Tilt Switch
1 PCS

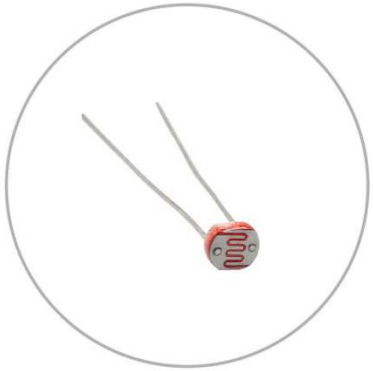
Thermistor
1 PCS



Capacitor Ceramic 100nF
4 PCS

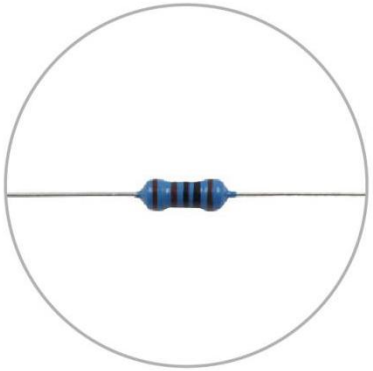
RGB LED

1 PCS



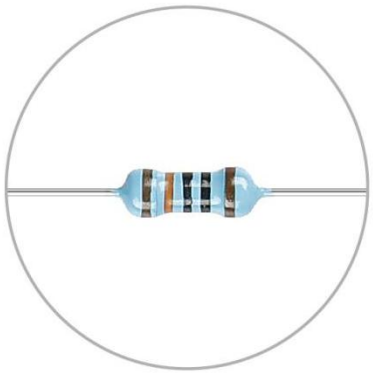
Photoresistor

1 PCS



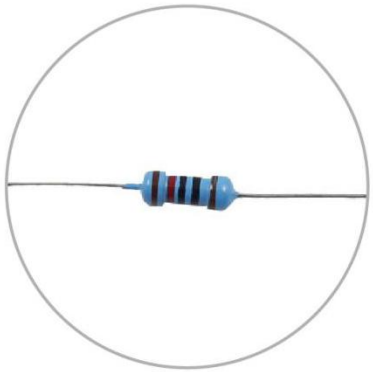
Resistor (1kΩ)

10 PCS



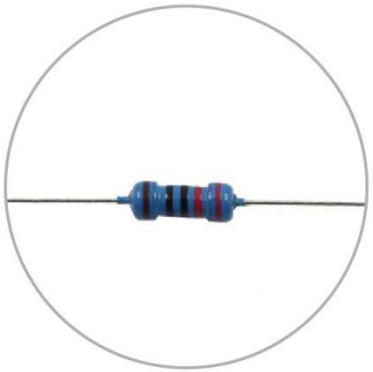
Resistor (100kΩ)

10 PCS



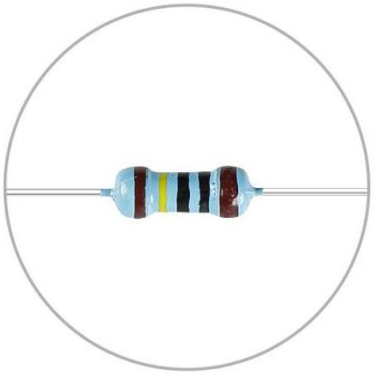
Resistor (10kΩ)

10 PCS



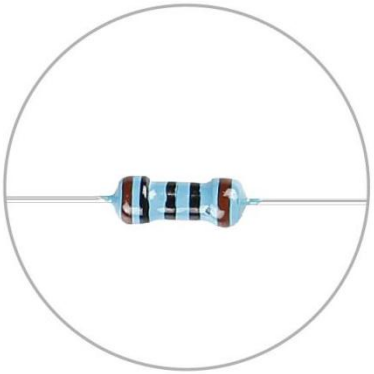
Resistor (220Ω)

10 PCS



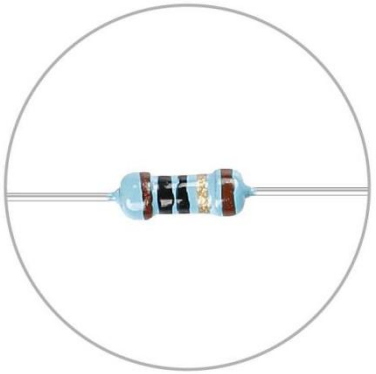
Resistor (1MΩ)

10 PCS



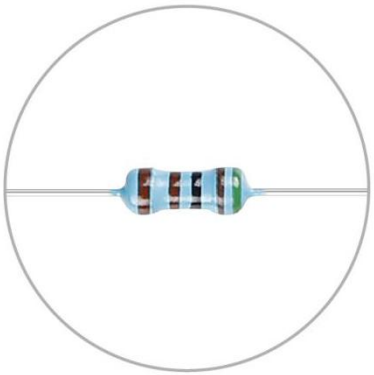
Resistor (100Ω)

10 PCS



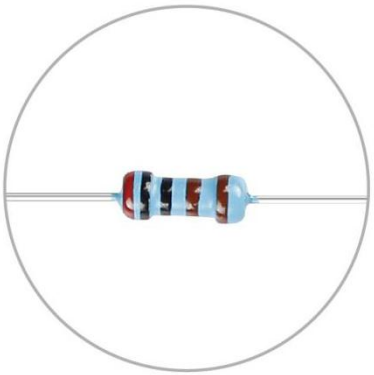
Resistor (10Ω)

10 PCS



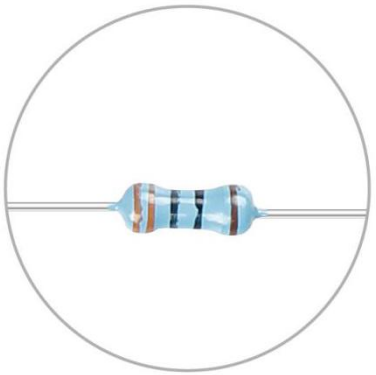
Resistor (5.1kΩ)

10 PCS



Resistor (2kΩ)

10 PCS

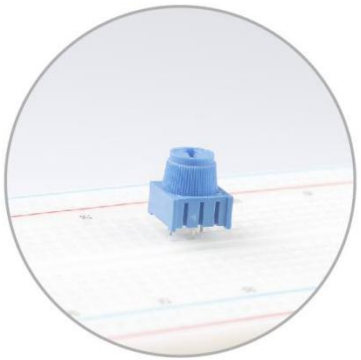


Resistor (330Ω)

10 PCS

1N4007

1 PCS

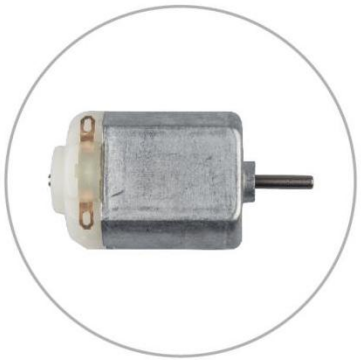


Potentiometer

1 PCS

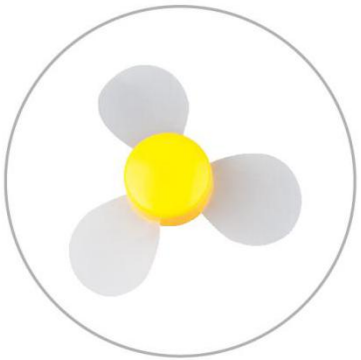
NPN Transistor (S8050)

2 PCS



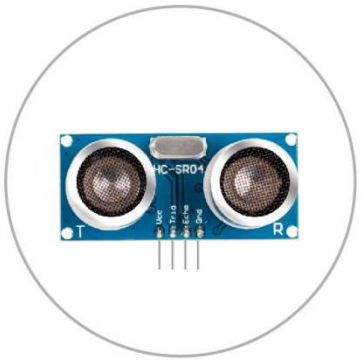
Small Motor

1 PCS



Fan

1 PCS



Ultrasonic Sensor Module

1 PCS

Contents

Lesson 1 Install and introduce Arduino IDE

Description

Arduino is an open source platform with simple software and hardware. You can pick it up in short time even if you are a beginner. It provides an integrated development environment (IDE) for code compiling, compatible with multiple control boards. So you can just download the Arduino IDE, upload the sketches (i.e. the code files) to the board, and then you can see relative experimental phenomena. For more information, refer to http://www.arduino.cc.

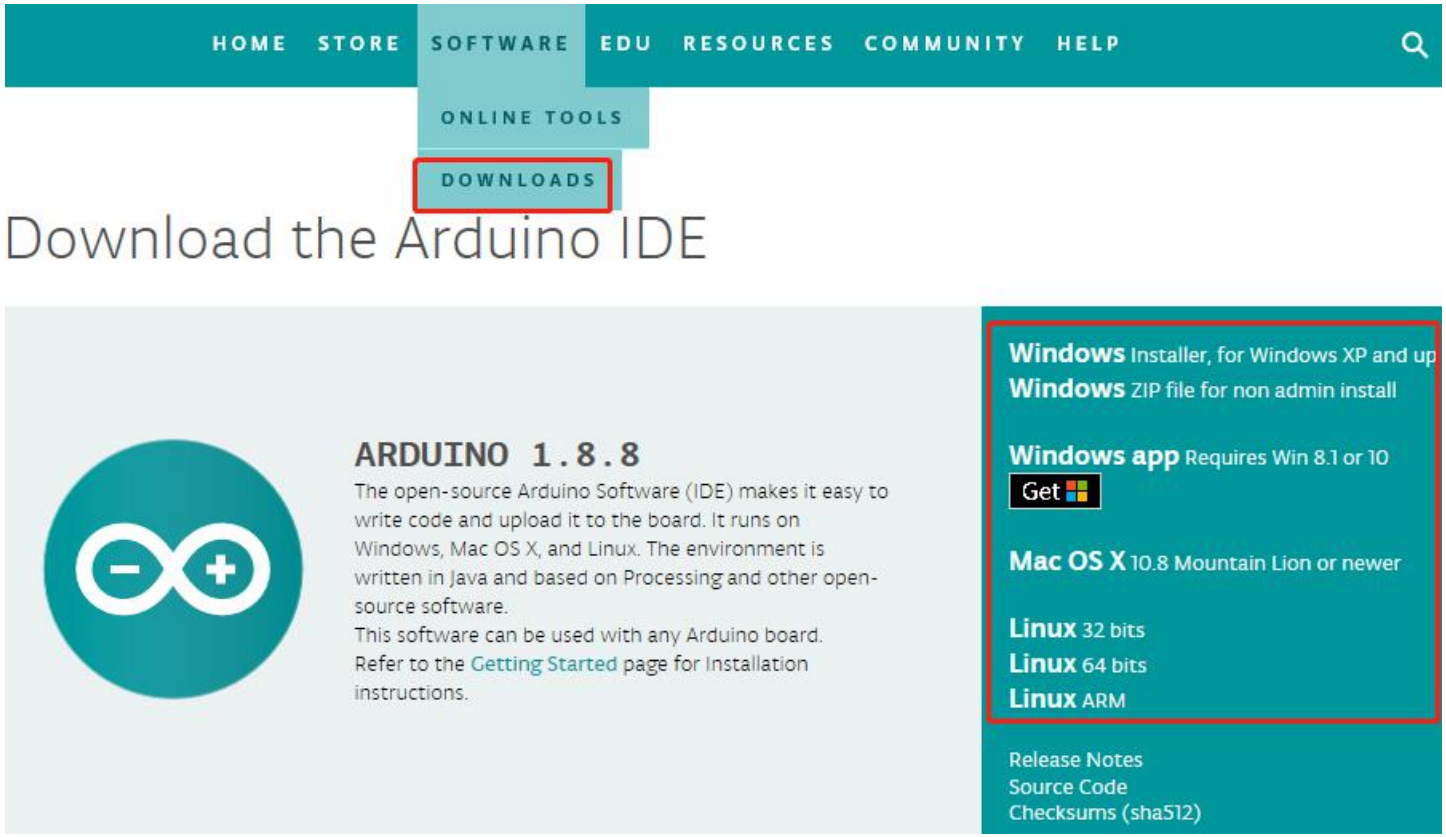
Install Arduino IDE

Here are the installation steps on the windows system.

For other systems, please refer to: **Install Arduinio IDE in different and FAQ.pdf**

The code in this kit is written based on Arduino, so you need to install the IDE first. Skip it if you have done this.

Now go to arduino.cc and click SOFTWARE -> DOWNLOADs. on the page, check the software list on the right side.



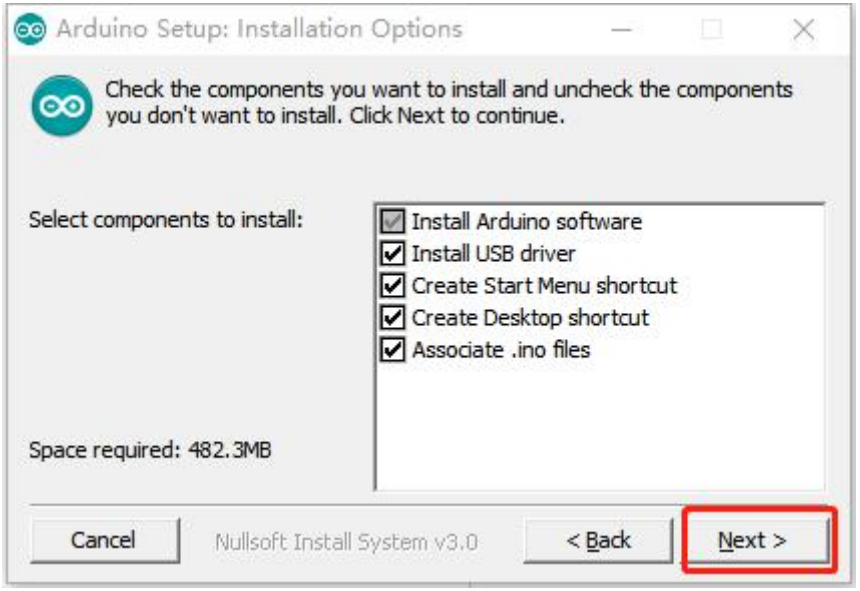
Find the one that suits your operation system and click to download. There are two versions of Arduino for Windows: Installer or ZIP file. You're recommended to download the former.

For Installer File

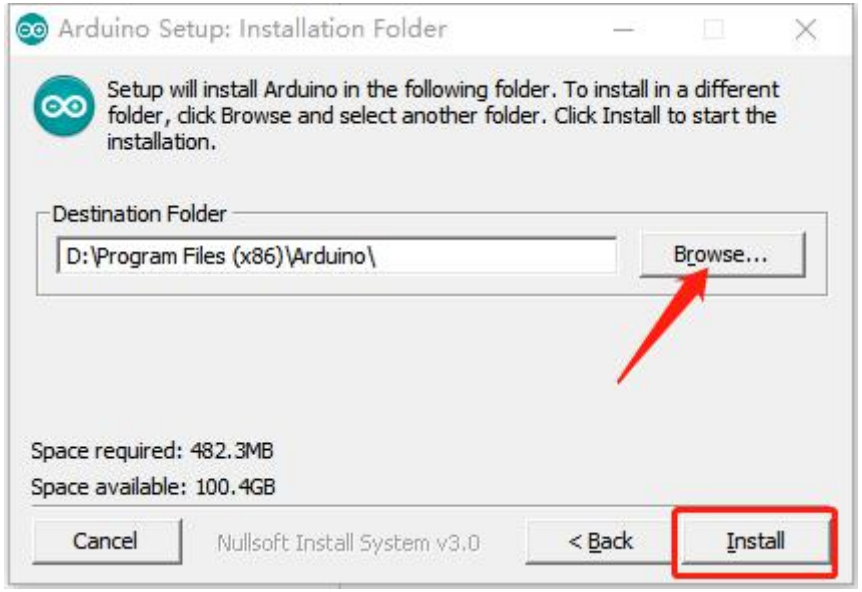
Step 1: Find the .exe file just downloaded.

Step 2: Double click the file and a window will pop up as below. Click **I Agree**.

Step 3: Click **Next.**



Step 4: Select the path to install. By default, it's set in the C disk. You can click **Browse** and choose other paths. Click **OK**. Then click Install.



Step 5: Meanwhile, it will prompts install the needed drivers, please select the 'Always trust software from "Arduino LLC" '. After the installation is done, click **Close**.

Note:

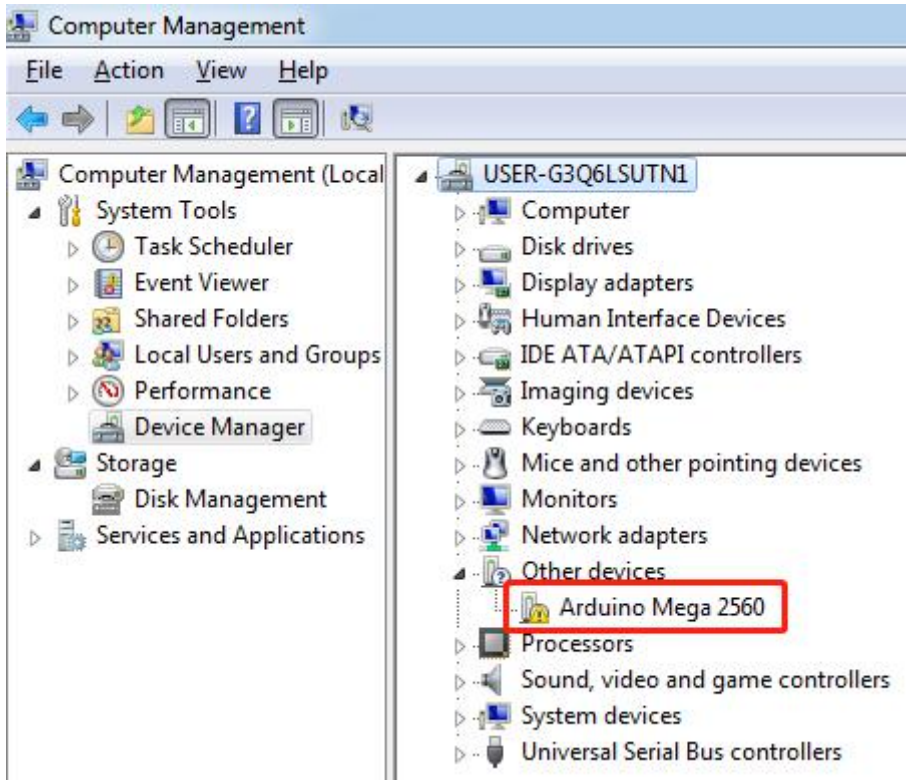
The new IDE may prompt errors when you're compiling code under Windows XP. So if your computer is running on XP, you're suggested to install Arduino 1.0.5 or 1.0.6. Also you can upgrade your computer.

For ZIP File

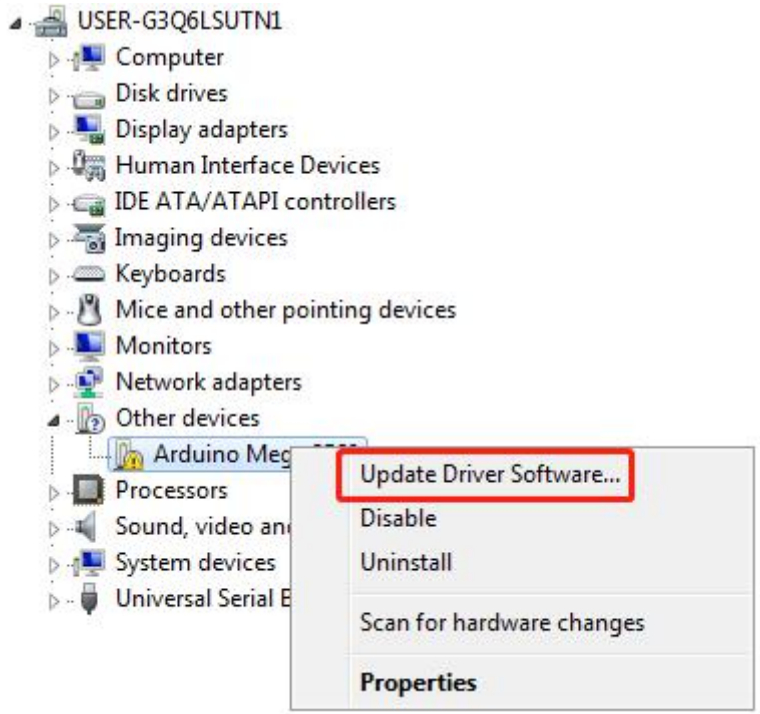
If you download the zip file before, when you connect the MCU to the computer, it may not be recognized. Then you need to install the driver manually. Take the following steps.

Step1: Plug in the board to the computer with a 5V USB cable. After a while, a prompt message of failed installation will appear.

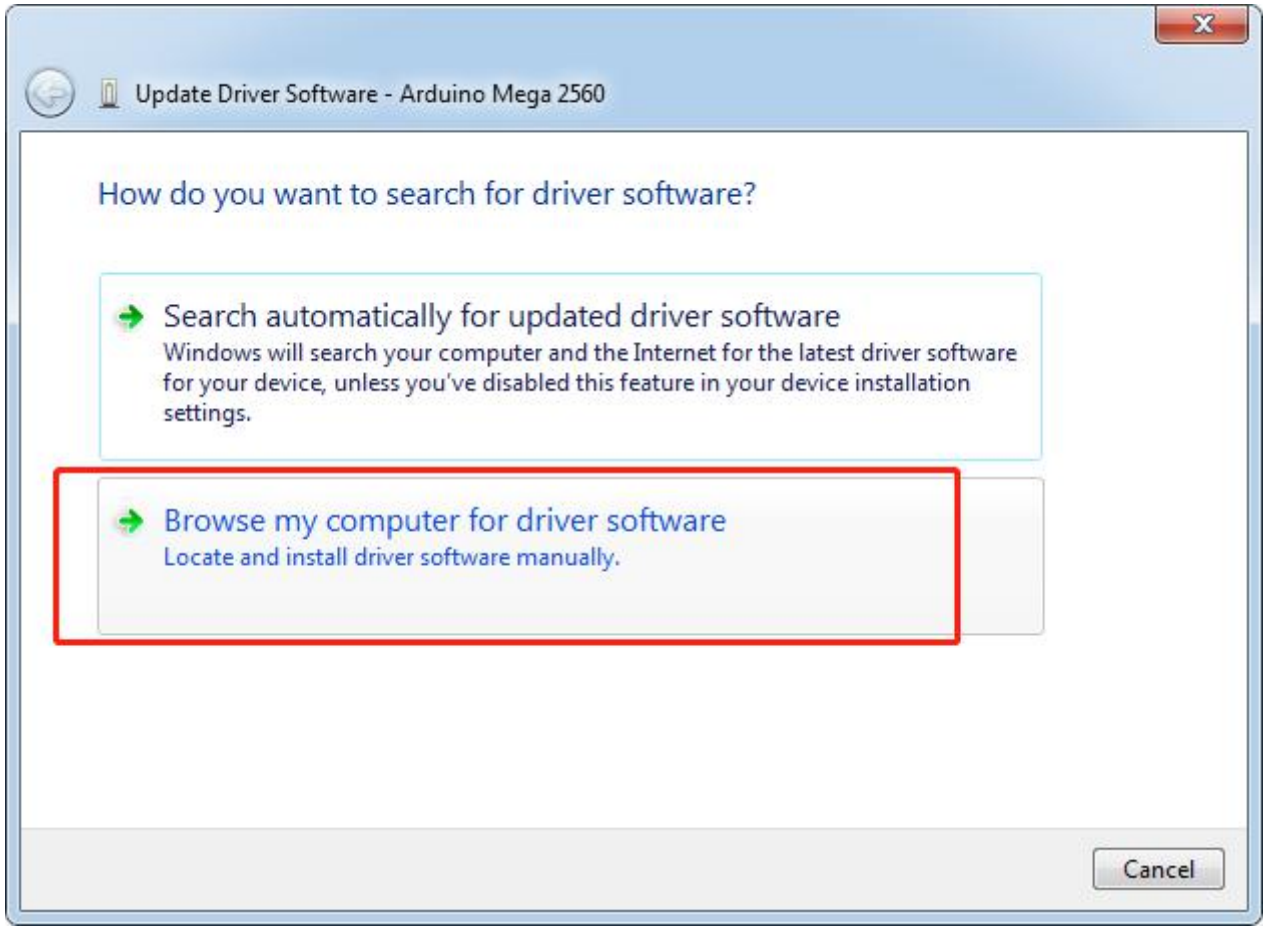
Step2: Go to the **Device Manager**. You will find under other devices, Arduino Mega 2560 with an exclamation mark appear, which means the computer did not recognize the board.



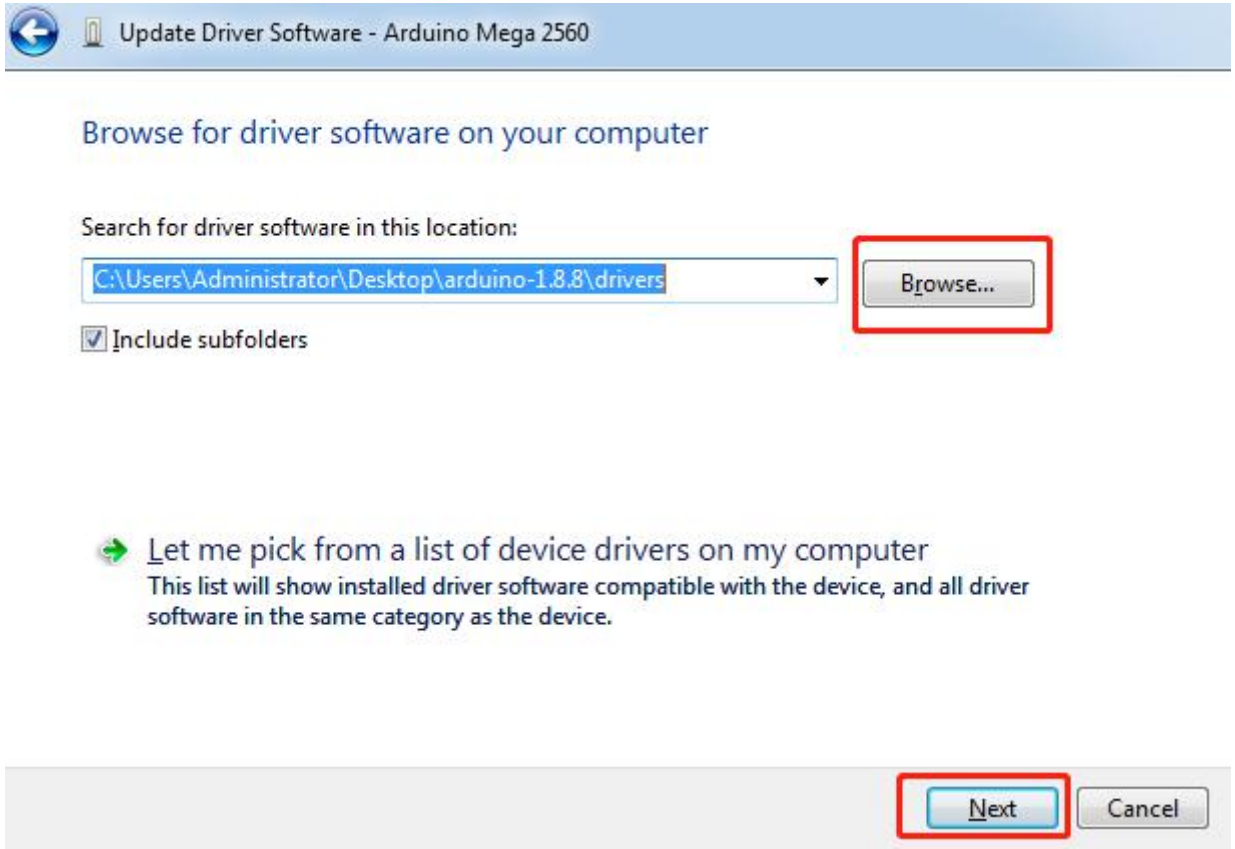
Step3: Right click on **Arduino Mega 2560** and select **Update Driver Software**.



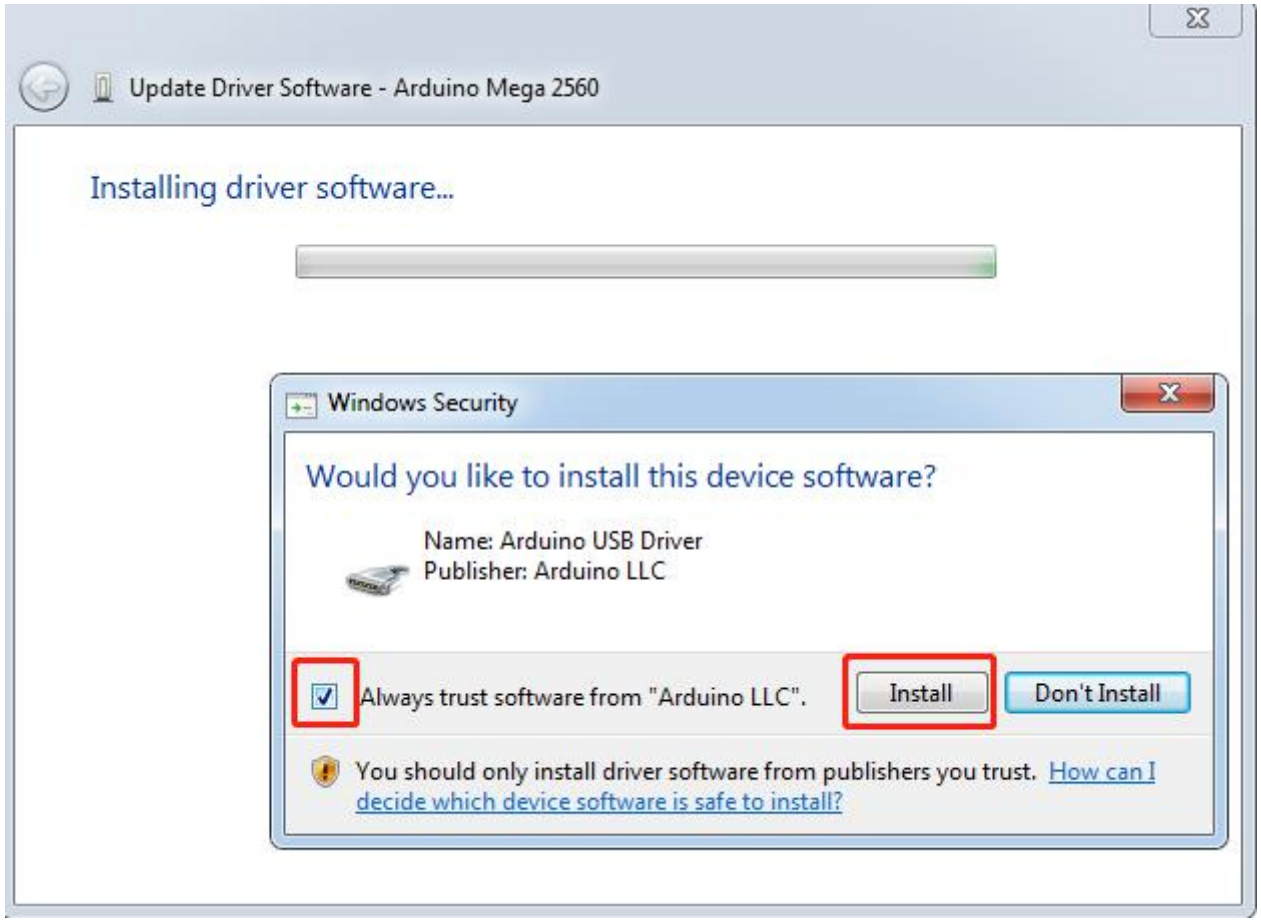
Step4: Choose the second option, **Browse my computer for Driver software**.



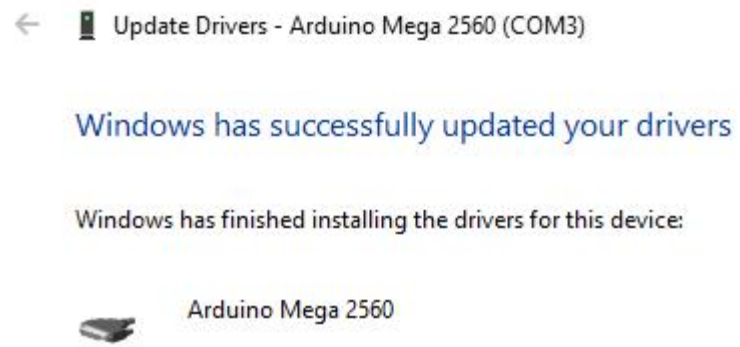
Step5: A window pops up then. Click **Browse**. Then go to the folder where you just extracted the file. Go to the *drivers* folder and click **OK** -> **Next**.



Step6: Select ‘Always trust software from “Arduino LLC” ‘ then click Install.



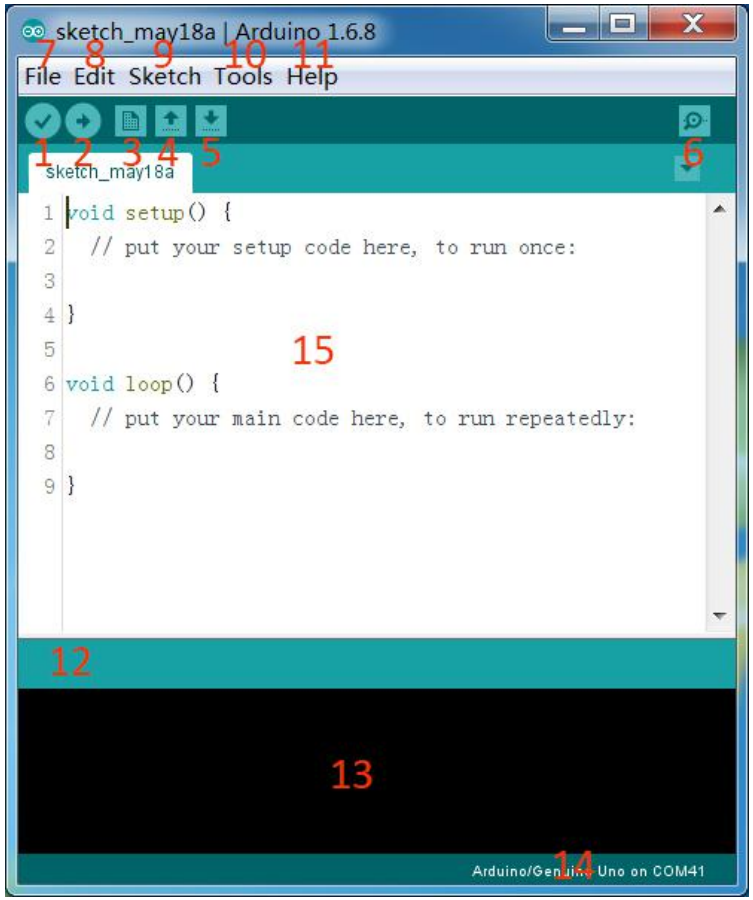
It may need a sec. Then the system prompts you the driver has been installed successfully. So the computer can recognize the board now. Click **Close**.



Open the Arduino Software (IDE)

Double-click the Arduino icon (arduino.exe) created by the installation process

Then the Arduino IDE will appear. Let's check details of the software.



1. **Verify**: Compile your code. Any syntax problem will be prompted with errors.

2. **Upload**: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.

3. **New**: Create a new code editing window.

4. **Open**: Open an .ino sketch.

5. **Save**: Save the sketch.

6. Serial Monitor: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.

7. **File**: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.

8. **Edit**: Click the menu. On the drop-down list, there are some editing operations like Cut, Copy, Paste, Find, and so on, with their corresponding shortcuts.

9. **Sketch**: Includes operations like Verify, Upload, Add files, etc. More important function is Include Library – where you can add libraries.

10. **Tool**: Includes some tools – the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.

11. **Help**: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.

12. In this message area, no matter when you compile or upload, the summary message will always appear.

13. Detailed messages during compile and upload. For example, the file used lies in which path, the details of error prompts.

14. **Board and Port**: Here you can preview the board and port selected for code upload. You can select them again by **Tools** -> **Board** / **Port** if any is incorrect.

15. The editing area of the IDE. You can write code here.

Lesson 2 Add libraries

What is Library?

A library, gathering some function definitions and header files, usually contains two files: .h (header file, including function statement, Macro definition, constructor definition, etc.) and .cpp (execution file, with function implementation, variable definition, and so on). When you need to use a function in some library, you just need to add a header file (e.g. #include <dht.h>), and then call that function. This can make your code more concise. If you don't want to use the library, you can also write that function definition directly. Though as a result, the code will be long and inconvenient to read.

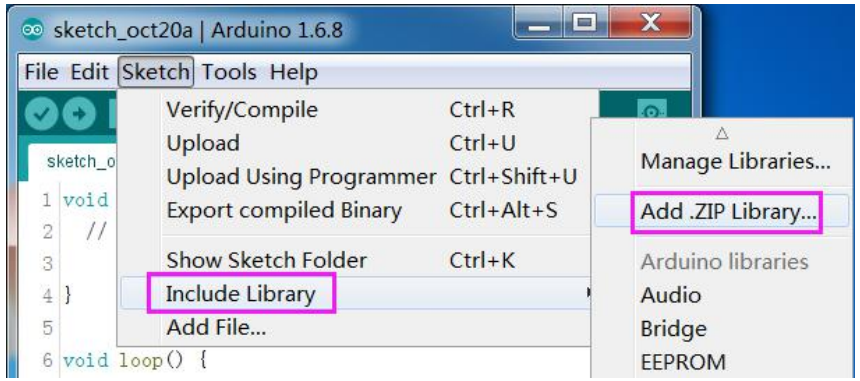
Add libraries

Some libraries are already built in the Arduino IDE, when some others may need to be added. So now let's see how to add one. There are 3 methods for that.

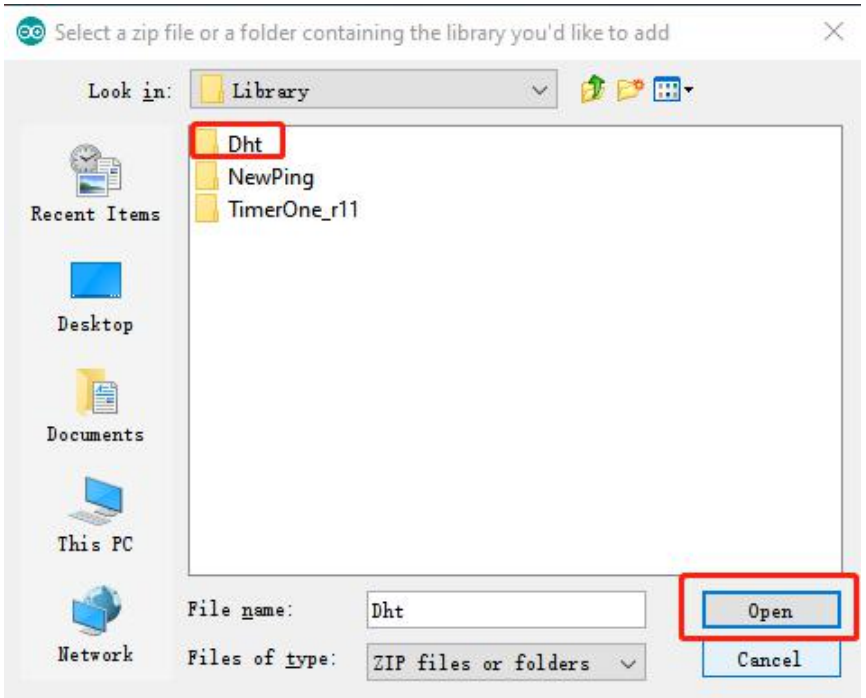
Method 1

Directly import the library in Arduino IDE (take Dht as an example below). The advantage of this method is easy to understand and operate, but on the other hand, only one library can be imported at a time. So it is inconvenient when you need to add quite a lot of libraries.

Step 1: Select **Sketch** -> **Include Library** -> **Add ZIP Library**.



Step 2: Find *SunFounder Mega Kit\Library*, Click **Open**.

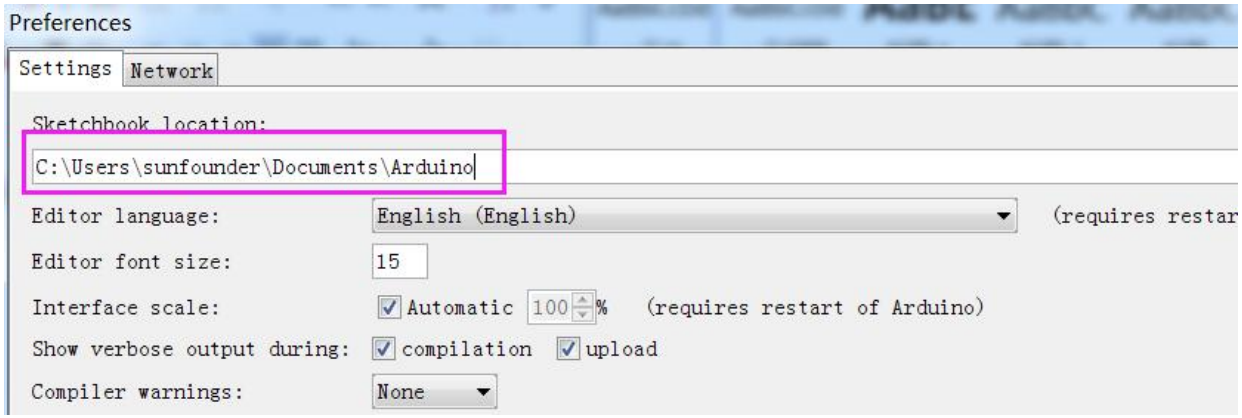


Step 3: When you see “Library added to your libraries. Check "Include library" menu”, it means you have added the library successfully. Please use the same method to add other libraries then.

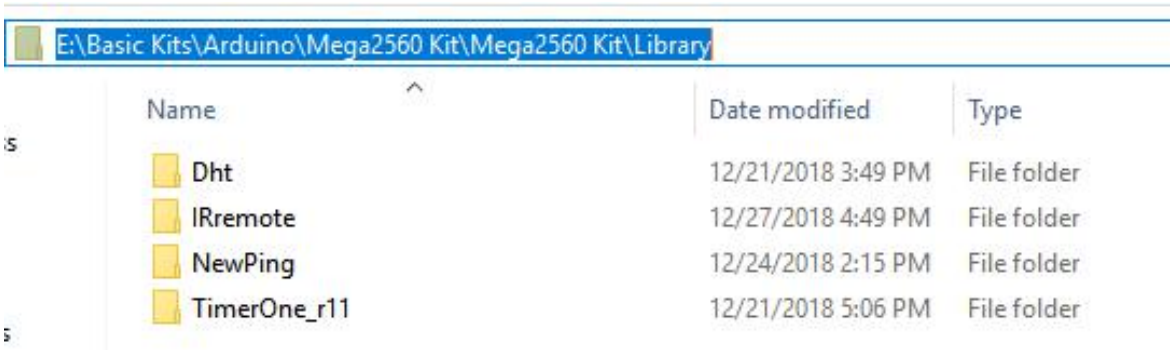
Method 2

Directly copy the library to libraries/Arduino path. This method can copy all libraries and add them at a time, but the drawback is that it is difficult to find libraries/Arduino.

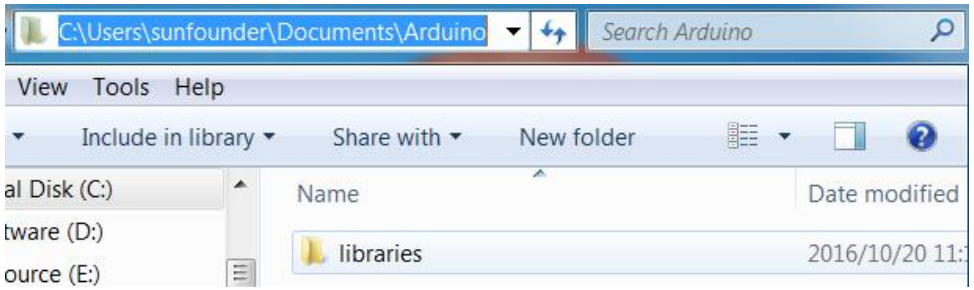
Step 1: Click **File** -> **Preferences** and on the pop-up window you can see the path of the libraries folder in the text box as shown below.



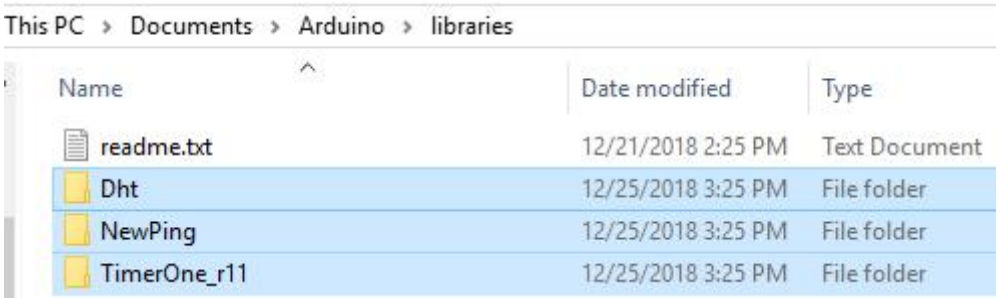
Step 3: Copy all Libraries in the *Library* folder.



Step 4: Go to the path above and you will see there is a *libraries* folder, click to open it.



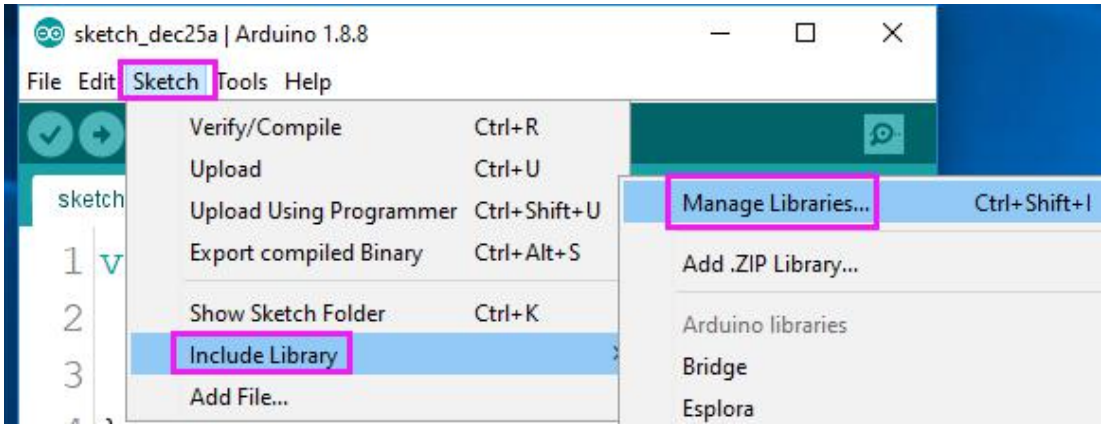
Step 5: Paste all the libraries copied before to the folder. Then you can see them in libraries folder.



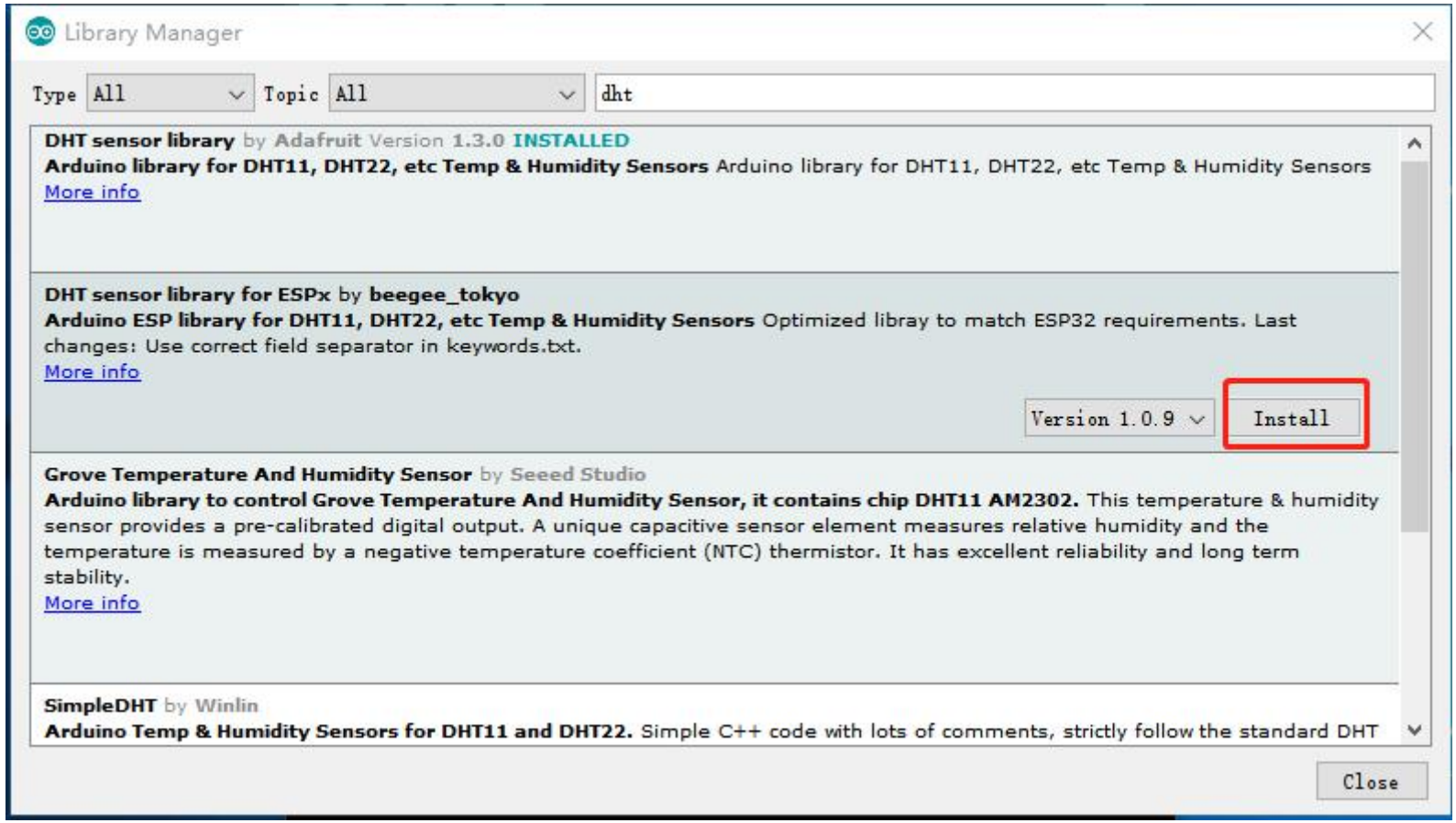
Method 3

Using the Library Manager to install a new library into the Arduino IDE.

Step1: Click **Sketch** -> **Include Library** ->**Manage Libraries**



Step2: Input a library name to search it, such as dht, you can choose the latest version to install.



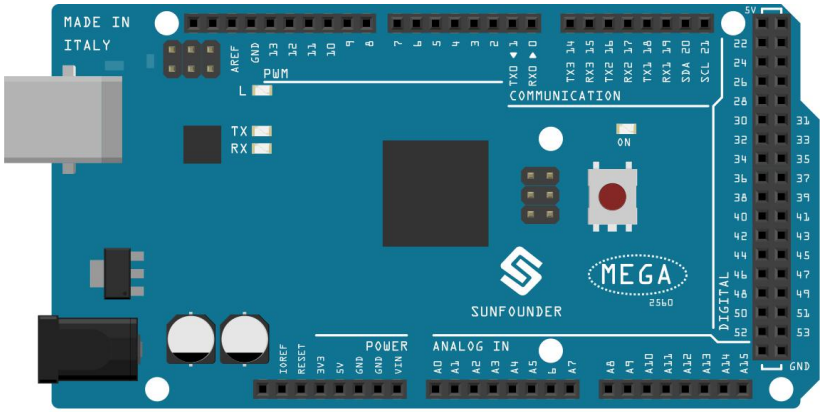
Once it has finished, you can close the library manager.

Lesson 3 Blinking LED

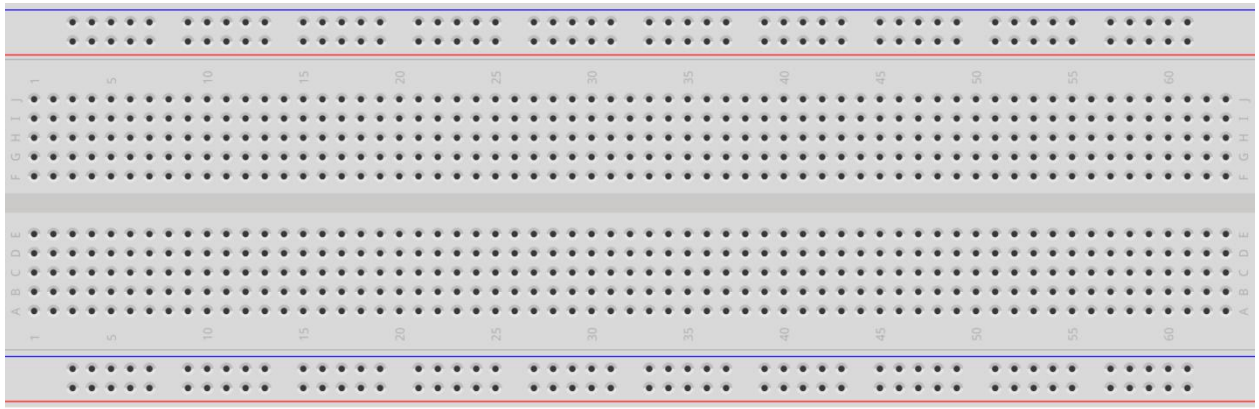
Introduction

You should've learnt how to install Arduino IDE and add libraries before. Now you can start with a simple experiment to learn the basic operation and code in the IDE.

Components

1 * Mega 2560 Board	1 * Resistor (220Ω)	1 * LED
		

1 * Breadboard



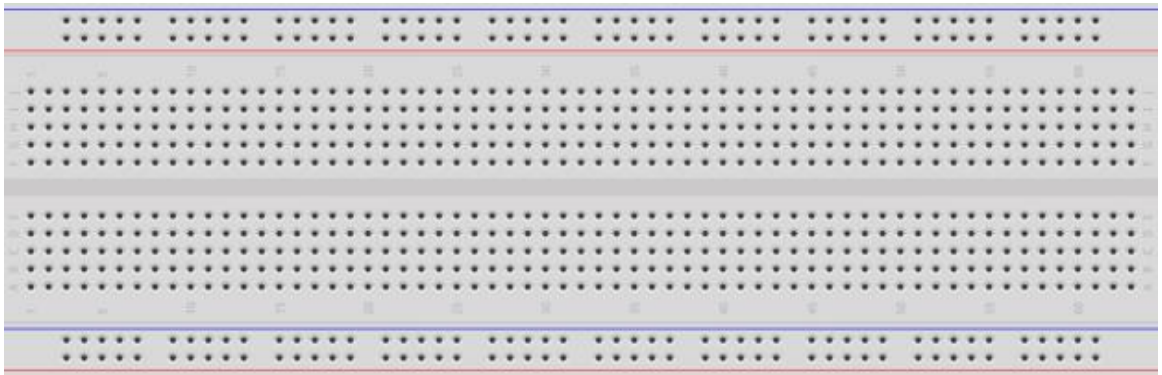
1 * USB cable

Several jumper wires

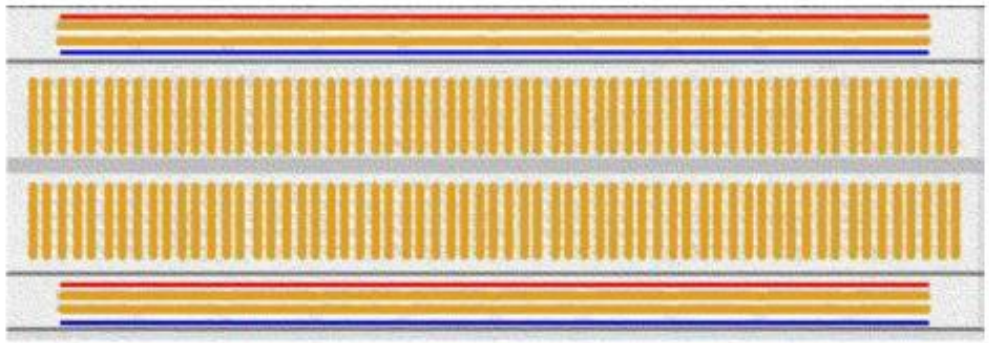
Component Introduction

Breadboard

A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finalizing any circuit design. And it has many holes into which components like ICs and resistors as well as jumper wires mentioned above can be inserted. The breadboard allows you to easily plug in and remove components. So if there are going to be many changes or if you just want to make a circuit quickly, it will be much quicker than soldering up your circuit. Therefore in lots of experiments, it is often used as a hub to connect two or more devices.



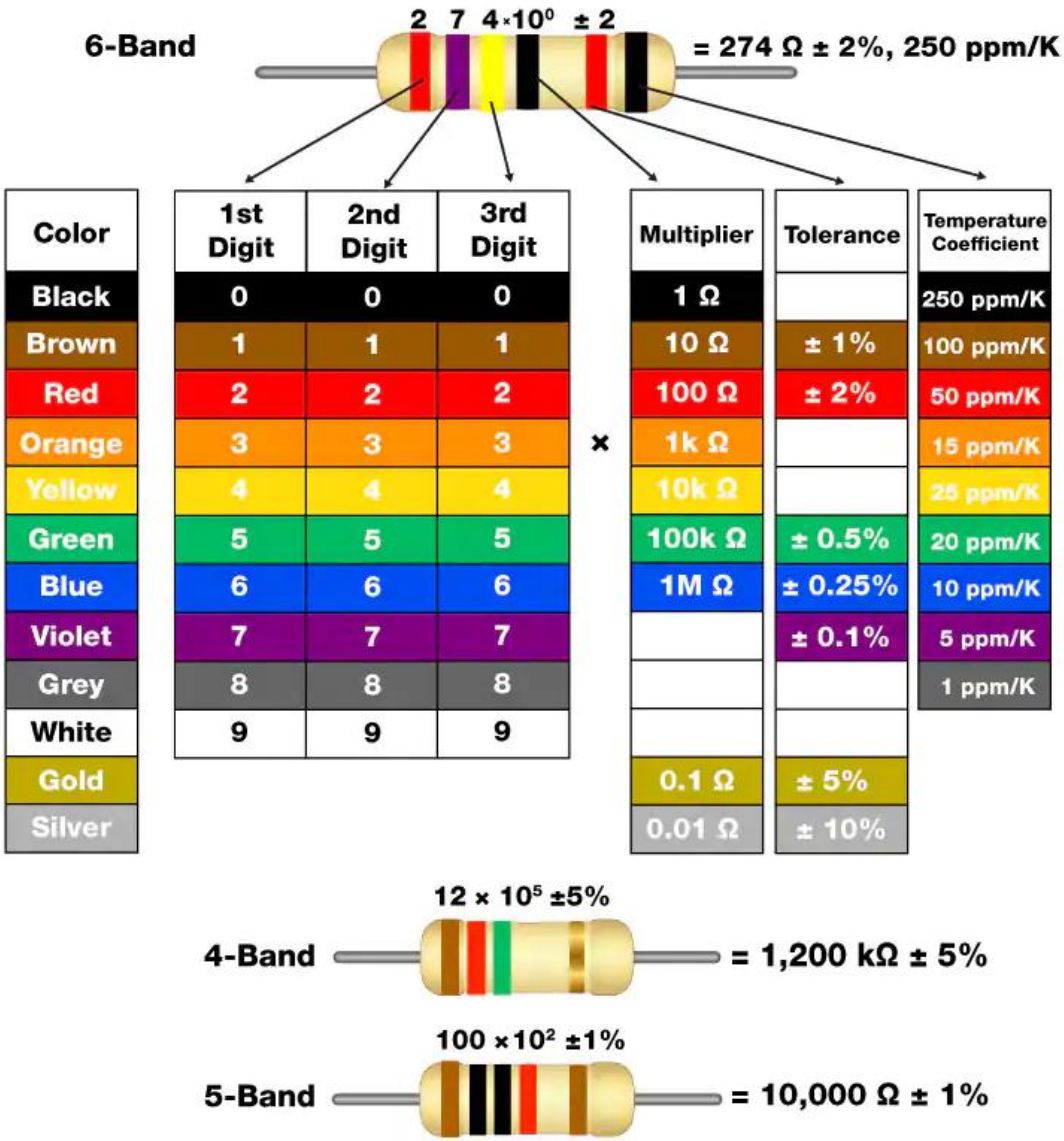
This is the internal structure of a full+ breadboard. Although there are holes on the breadboard, internally some of them are connected with metal strips. Those holes are to insert pins of devices or wires. There are four long metal strips on the long sides; the blue and red lines are marked just for clear observation. But you can take the blue line as the GND and red one as VCC for convenience. Every five holes in the middle are vertically connected with metal trips internally which don't connect with each other. You can connect them horizontally with wires or components. A groove is made in the middle on the breadboard for IC chips.



Resistor

Resistor is an electronic element that can limit the branch current. A fixed resistor is one whose resistance cannot be changed, when that of a potentiometer or variable resistor can be adjusted.

The resistors in this kit are fixed ones. It is essential in the circuit to protect the connected components. The following pictures show a real 220Ω resistor and two generally used circuit symbols for resistor. Ω is the unit of resistance and the larger includes KΩ, MΩ, etc. Their relationship can be shown as follows: 1 MΩ=1000 KΩ, 1 KΩ = 1000 Ω, which means 1 MΩ = 1000,000 Ω = 10^6 Ω. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.



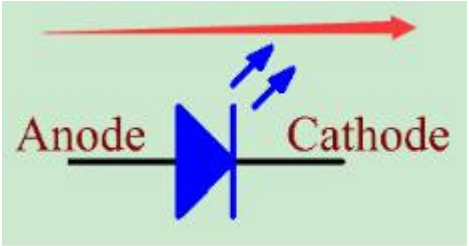
The resistance can be marked directly, in color code, and by character. The resistors offered in this kit are marked by different colors. Namely, the bands on the resistor indicate the resistance.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster. If you are not sure about the value, use the multimeter.

As shown in the card, each color stands for a number.

LED

Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode which is usually known as light-emitting diode (LED).



Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative. Thus the LED will light up.

An LED has two pins. The longer one is the anode, and shorter one, the cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$$R = (V_{supply} - V_D)/I$$

R stands for the resistance value of the current limiting resistor, Vsupply for voltage supply, VD for voltage drop and I for the working current of the LED.

If we provide 5 voltage for the red LED, the minimum resistance of the current limiting resistor should be: (5V-1.8v)/20mA = 160Ω. Therefore, you need a 160Ω or larger resistor to protect the LED. You are recommended to use the 220Ω resistor offered in the kit.

Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Among others, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jump wires are fitted by inserting their "end connectors" into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The "end connectors" are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female. The reason we call it Male-to-Female is because it has the outstanding tip in one end as well as a sunk female end. Male-to-Male means both side are male and Female-to-Female means both ends are female.

Male-to-Female

Male-to-Male

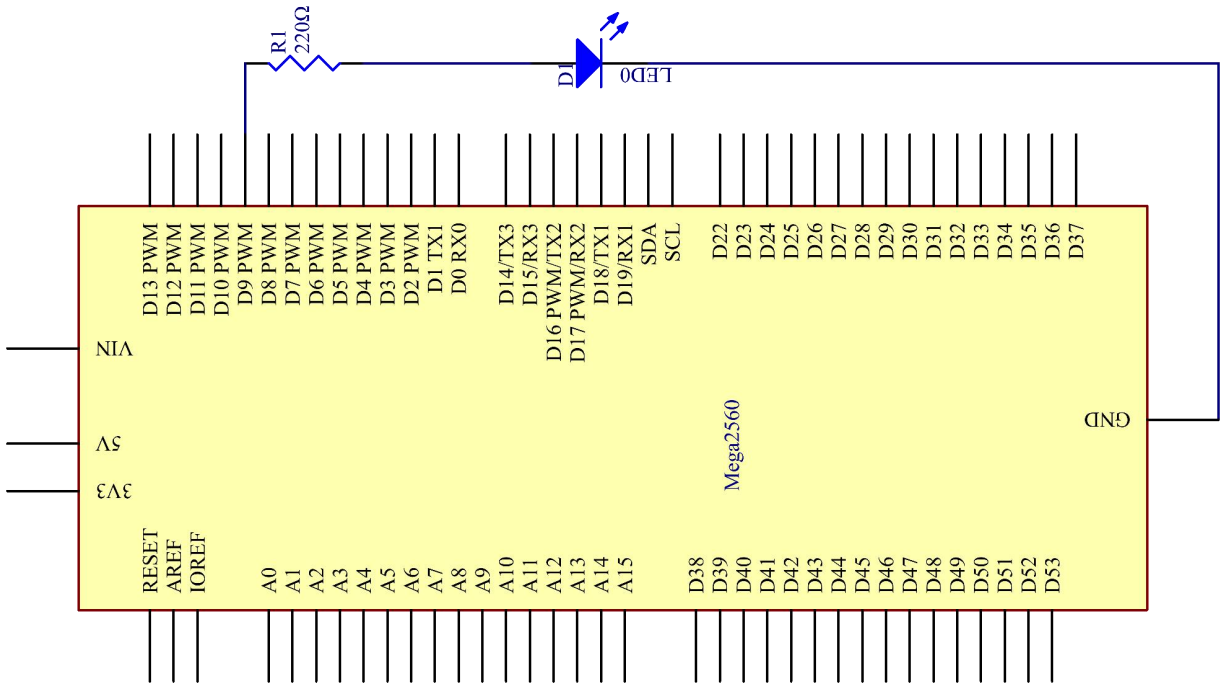
Female-to-Female

More than one type of them may be used in a project. The color of the jump wires is different but it doesn't mean their function is different accordingly; it's just designed so to better identify the connection between each circuit.

Principle:

Connect one end of the 220ohm resistor to pin 9 of the Mega 2560 and the other end to the anode (the long pin) of the LED, and the cathode (the short pin) of the LED to GND. When the pin 9 outputs high level, the current gets through the current limiting resistor to the anode of the LED. And since the cathode of the LED is connected to GND, the LED will light up. When pin 9 outputs low level, the LED goes out.

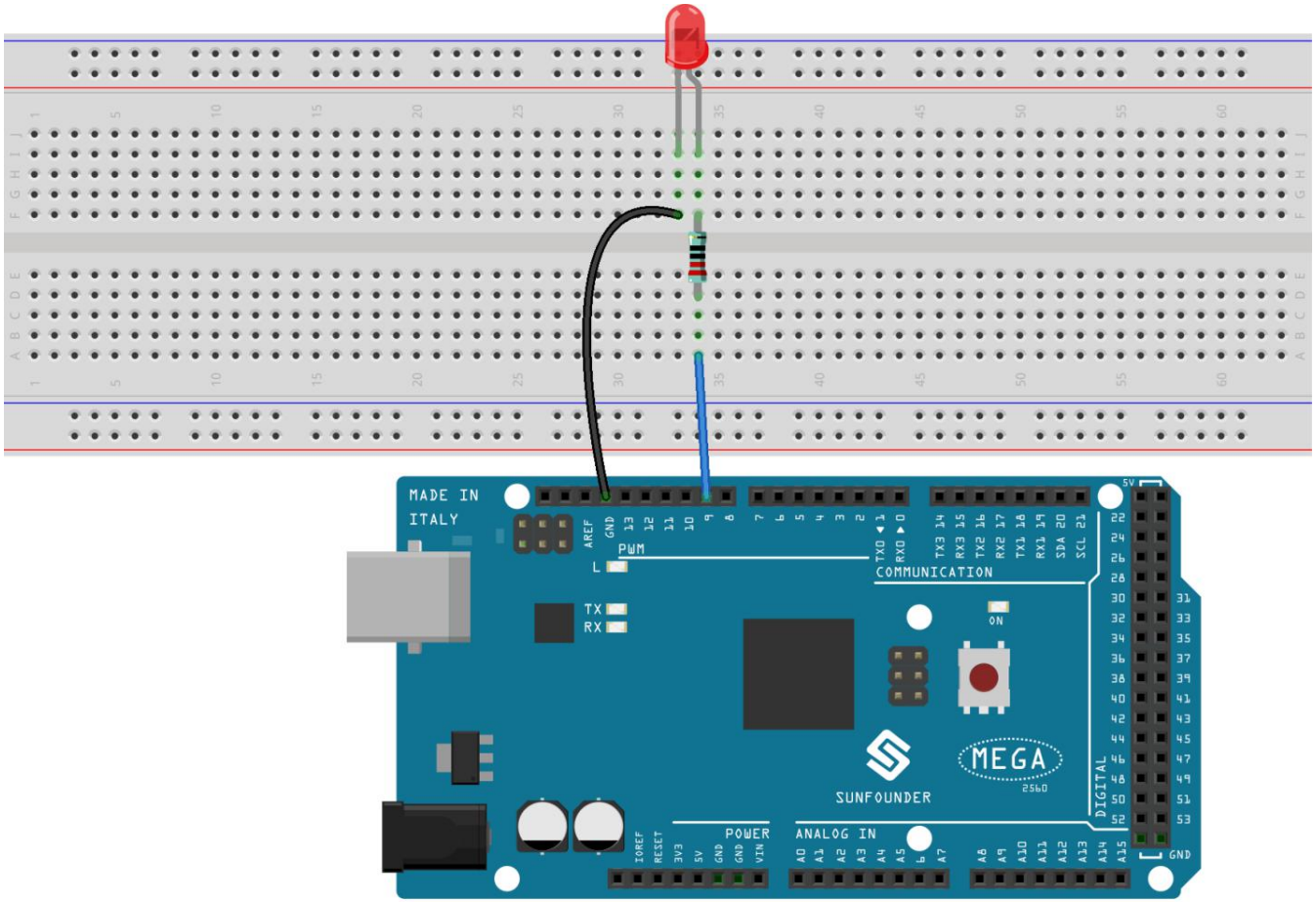
The schematic diagram：



Experimental Procedures

Step 1: Build the circuit (the pin with a curve is the anode of the LED).

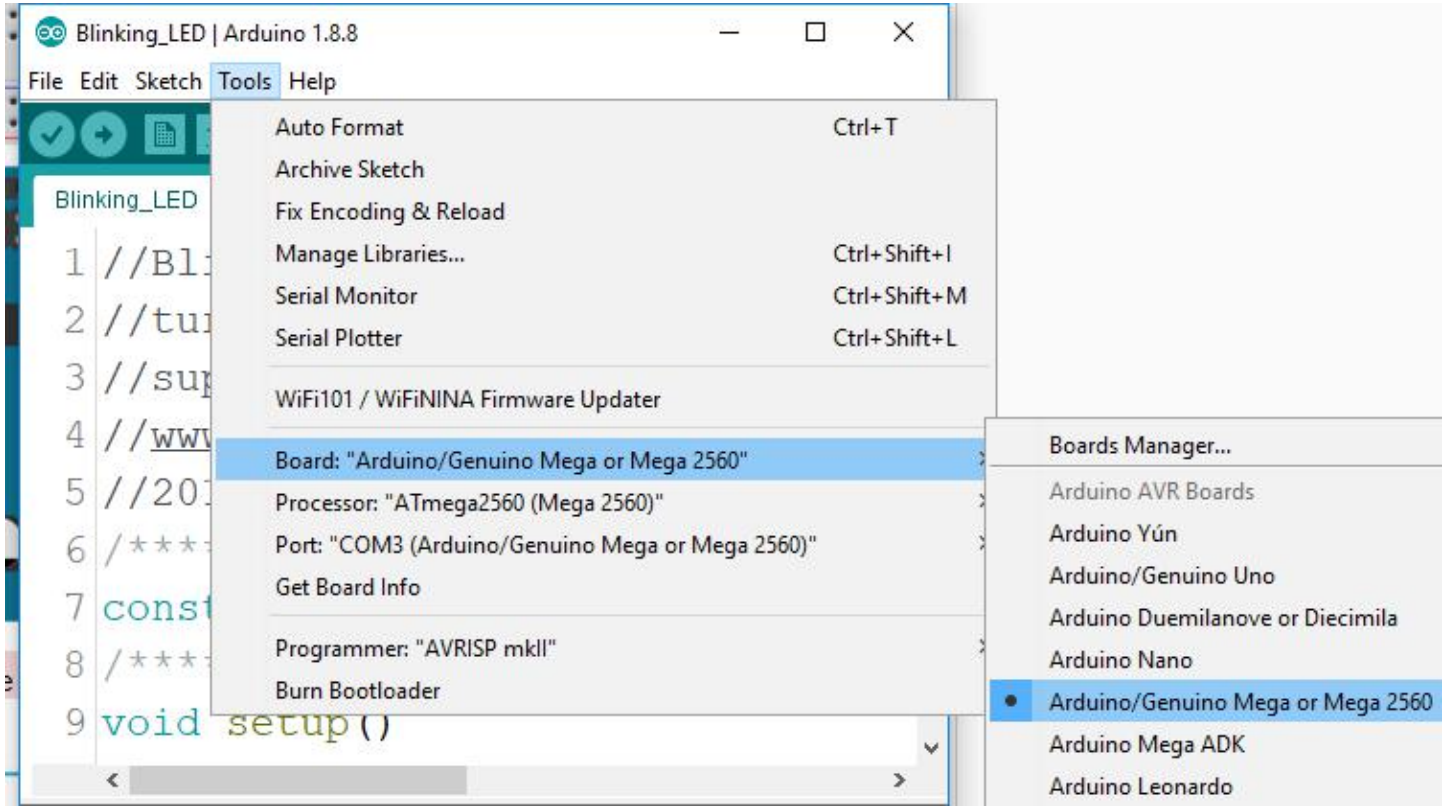
Then plug the board into the computer with a 5V USB cable.



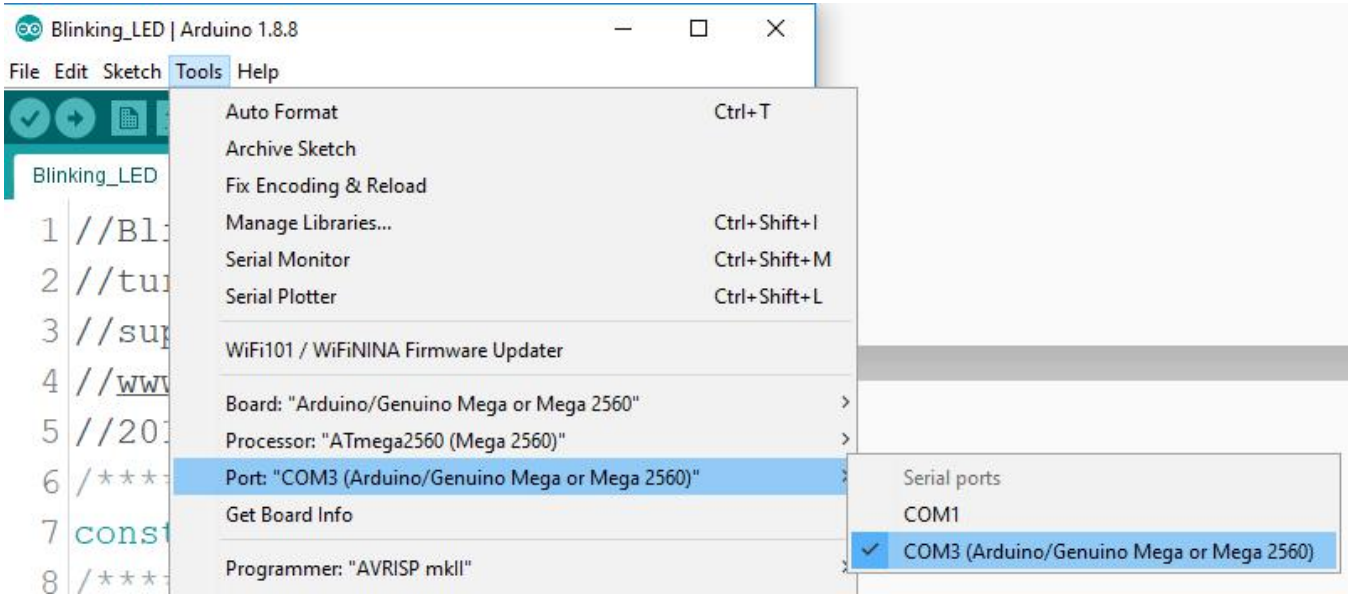
Step 2: Open the Lesson_3_Blinking_LED.ino code file in the path of *SunFounder Mega Kit\Code\Lesson_3_Blinking_LED*

Step 3: Select the Board and Port

Before uploading the code, you need to select the **Board** and **Port**. Click **Tools ->Board** and select **Arduino/Genuino Mega or Mega 2560**.

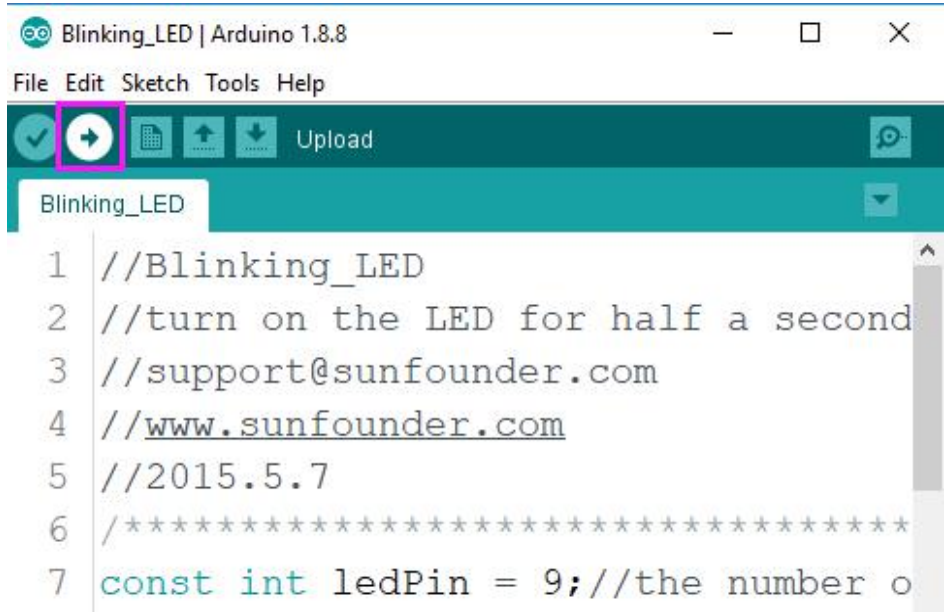


Then select **Tools** ->**Port**. Your port should be different from mine.



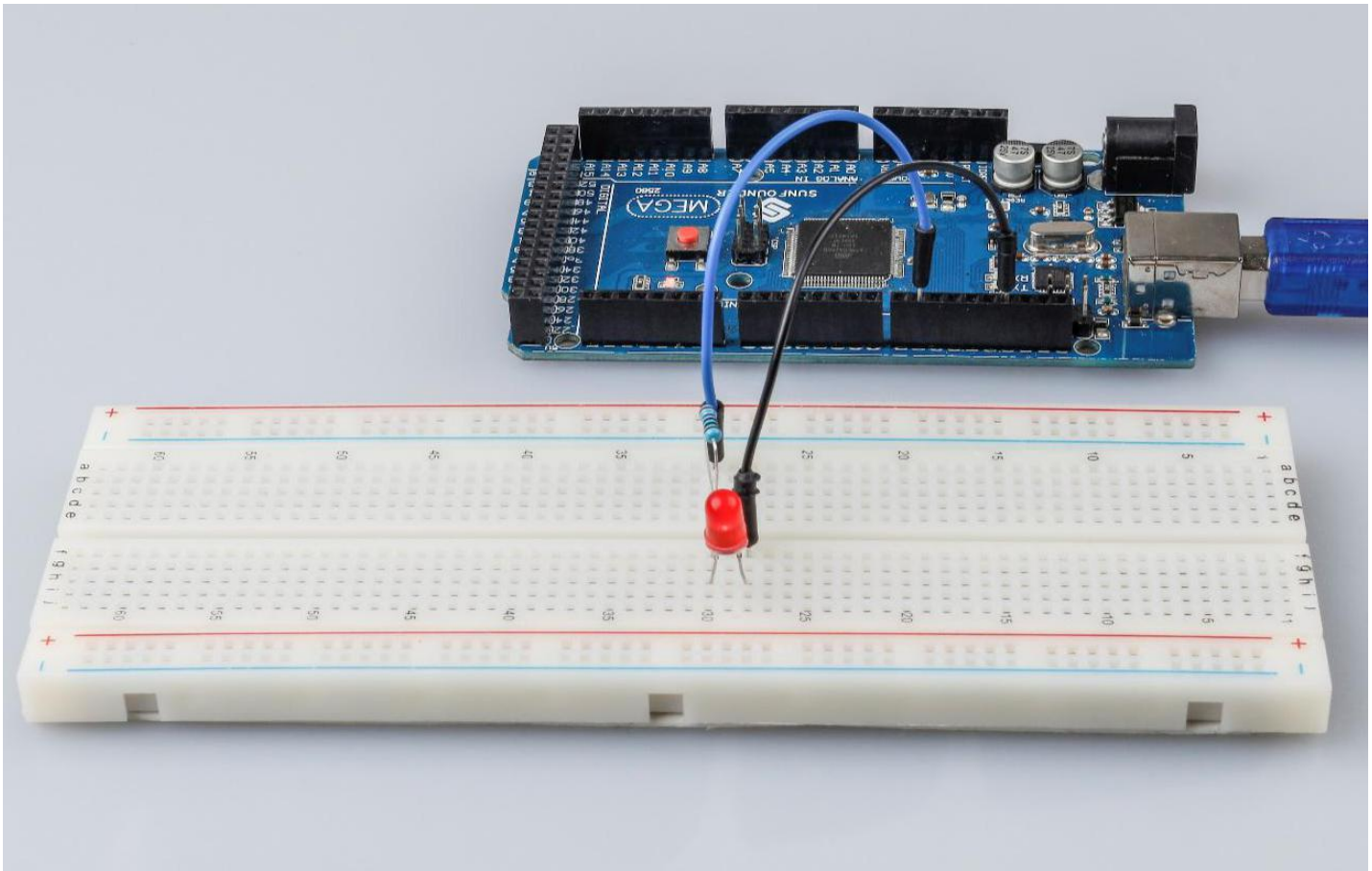
Step 4: Upload the sketch to the SunFounder Mega2560 board

Click the **Upload** icon to upload the code to the control board.



If "Done uploading" appears at the bottom of the window, it means the sketch has been successfully uploaded.

You should now see the LED blinking.



Code Analysis

Code Analysis 3-1 Define variables

```
const int ledPin = 9;//the number of the LED pin
```

You should define every variable before using in case of making mistakes. This line defines a constant variable *ledPin* for the pin 9. In the following code, *ledPin* stands for pin 9. You can also directly use pin 9 instead.

Code Analysis 3-2 setup() function

A typical Arduino program consists of two subprograms: *setup()* for initialization and loop() which contains the main body of the program.

The *setup()* function is usually used to initialize the digital pins and set them as input or output as well as the baud rate of the serial communication.

The *loop()* function contains what the MCU will run circularly. It will not stop unless something happens like power outages.

```
void setup()
{
  pinMode(ledPin,OUTPUT);
}
```

The setup() function here sets the *ledPin* as OUTPUT.

pinMode(Pin): Configures the specified pin to behave either as an input or an output.

The void before the setup means that this function will not return a value. Even when no pins need to be initialized, you still need this function. Otherwise there will be errors in compiling.

You can write:

```
void setup() {
  // put your setup code here, to run once:

}
```

Code Analysis 3-3 loop function

```
void loop()
{
  digitalWrite(ledPin,HIGH);//turn the LED on
  delay(500);               //wait for half a second
  digitalWrite(ledPin,LOW); //turn the LED off
  delay(500);               //wait for half a second
}
```

This program is to set *ledPin* as HIGH to turn on the LED, with a delay of 500ms. Set *ledPin* as LOW to turn the LED off and also delay 500ms. The MCU will run this program repeatedly and you will see that the LED brightens for 500ms and then dims for 500ms. This on/off alternation will not stop until the control board runs out of energy.

digitWrite(Pin): Write a HIGH or a LOW value to a digital pin. When this pin has been set as output in *pinModel()*, its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

Experiment Summary

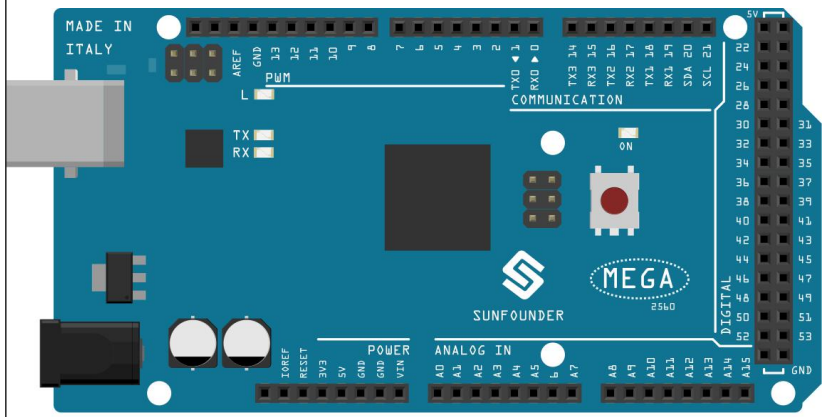
Through this experiment, you have learned how to turn on an LED. You can also change the blinking frequency of the LED by changing the *num* value in the delay function *delay (num)*. For example, change it to ***delay (250)*** and you will find that the LED blinks more quickly.

Lesson 4 Flowing LED Lights

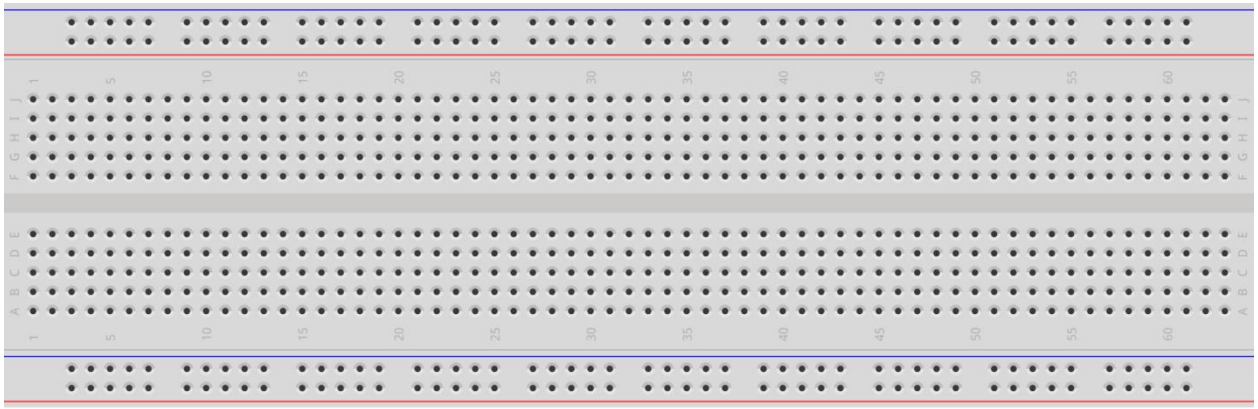
Introduction

In this lesson, we will conduct a simple yet interesting experiment – using LEDs to create flowing LED lights. As the name suggests, these eight LEDs in a row successively light up and dim one after another, just like flowing water.

Components

1 * Mega 2560 Board	8 * Resistor (220Ω)	8 * LED
		

1 * Breadboard



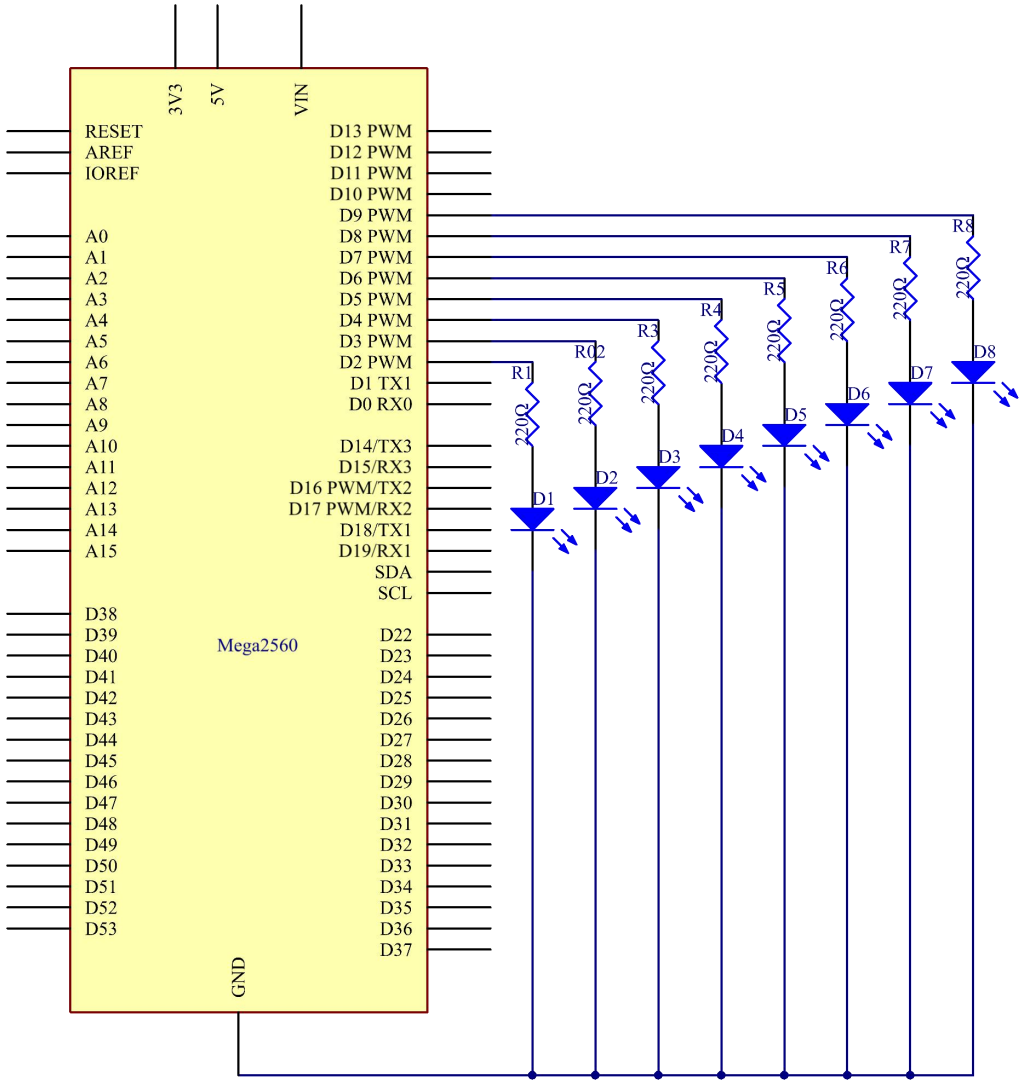
1 * USB cable

Several jumper wires

Experimental Principle

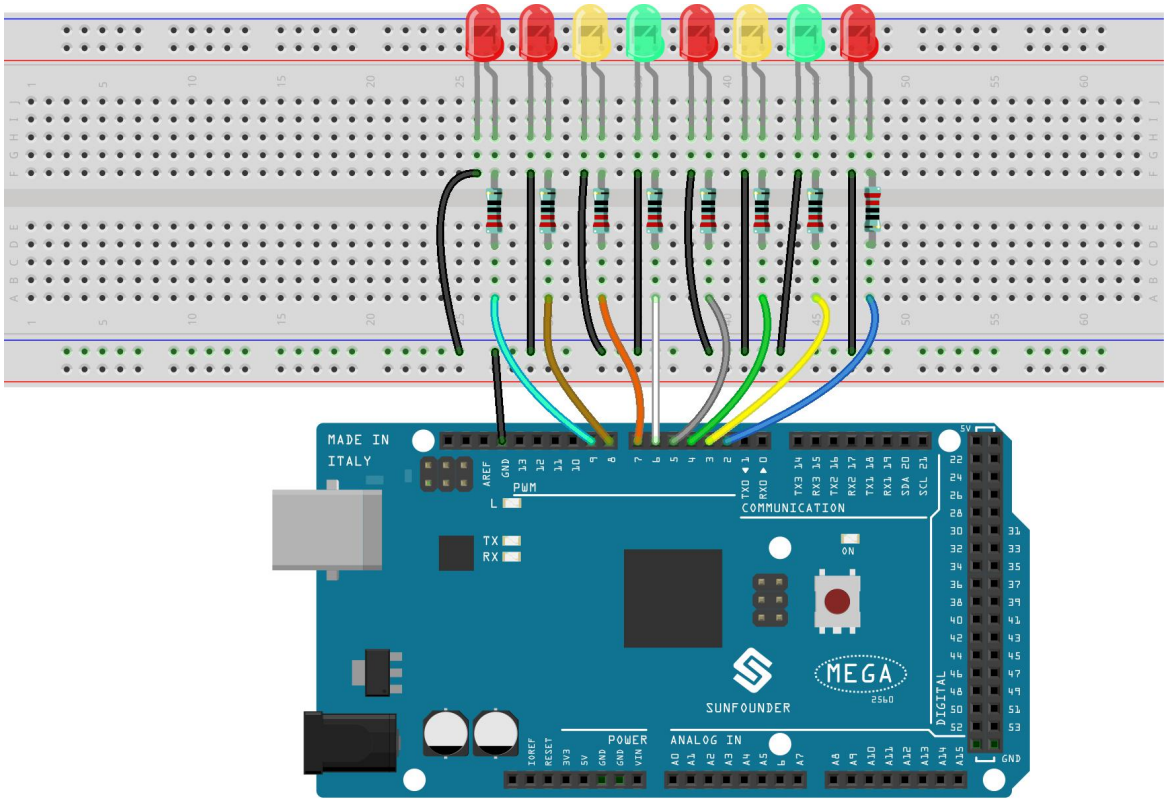
The principle of this experiment is simply to turn on eight LEDs in turn. The eight LEDs are connected to pin 2-pin 9 respectively. Set them as High level and the corresponding LED at the pins will light up. Control the time of each LED brightening and you will see flowing LED lights.

The schematic diagram


3V3
5V
VIN
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D38
D39
D40
D41
D42
D43
D44
D45
D46
D47
D48
D49
D50
D51
D52
D53
GND
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D3 PWM
D2 PWM
D1 TX1
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
SCL
D22
D23
D24
D25
D26
D27
D28
D29
D30
D31
D32
D33
D34
D35
D36
D37
Mega2560
R1
R2
R3
R4
R5
R6
R7
R8
220Ω
D1
D2
D3
D4
D5
D6
D7
D8

Experimental Procedures

Step 1: Build the circuit

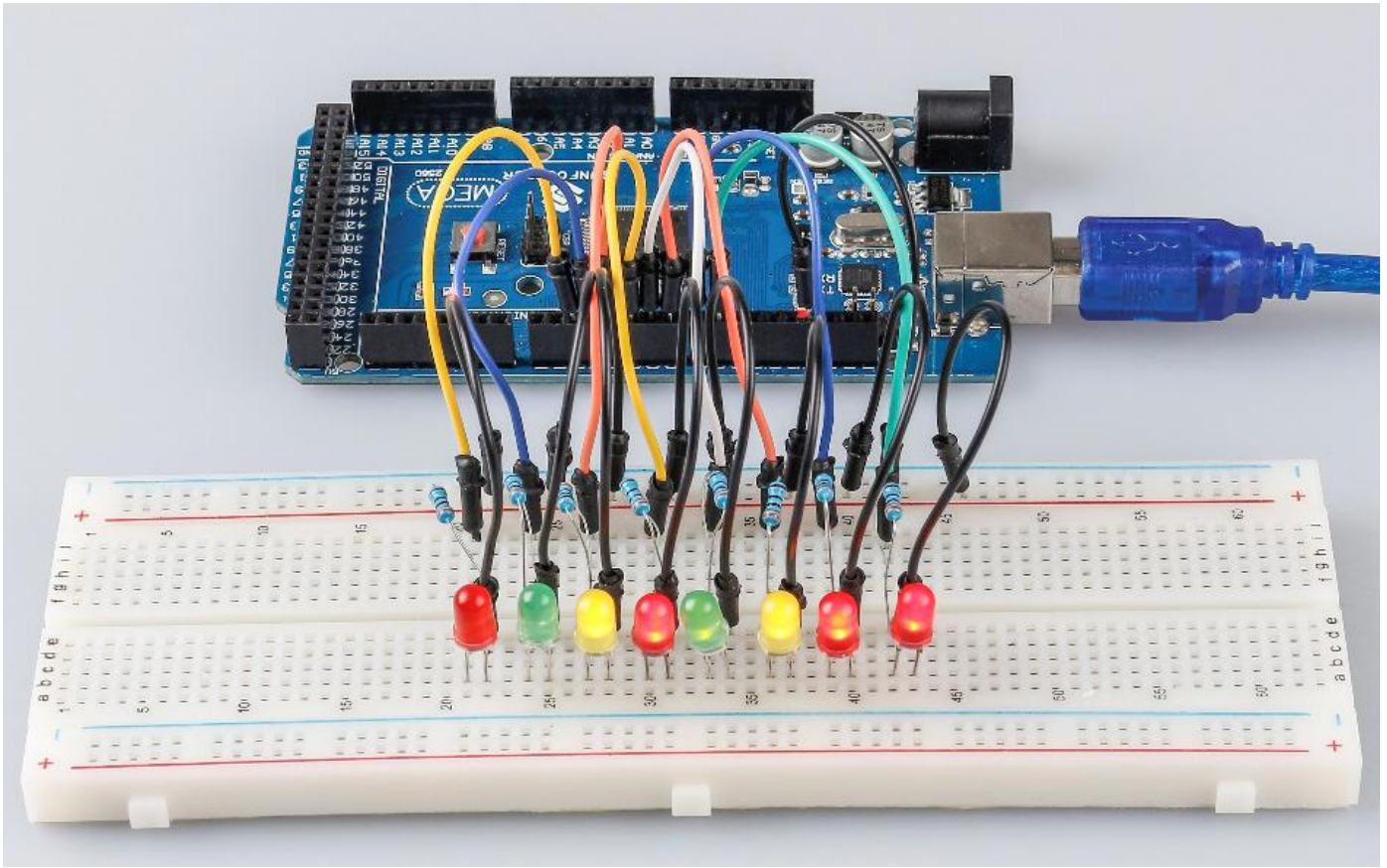


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you should see eight LEDs brighten one by one from the LED connected to pin 2 to that to pin 9, and then dim in turn from the LED at pin 9 to the one at pin 2. After that, the LEDs will light up from the LED at pin 9 to that at pin 2 and dim from the LED at pin 2 to that at pin 9. This whole process will repeat until the circuit is power off.



Code Analysis

Code Analysis 4-1 for() statement

```
for (int i = 2; i <= 9; i++)
```
//8 LEDs are connect to pin2-pin9, When i=2, which accords with the condition i<=9, then run the code in the curly braces, set the pin2 to OUTPUT. After that run i++(here in i = i + 1, the two "i"s are not the same, but $i_{now} = i_{before} + 1$). Use the for() statement to set pin 2-pin 9 as output respectively.

```
{
  pinMode(i, OUTPUT); //initialize a as an output
}
```

for (initialization; condition; increment) { //statement(s); }: The for statement is used to repeat a block of statements enclosed in curly braces.The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Code Analysis 4-2 Set flowing led lights

Use the for() statement to set pin2-pin9 to a high level inturn.

```
for (int a = 2; a <= 9; a++)
{
  digitalWrite(a, HIGH); //turn this led on
  delay(100);//wait for 100 ms
}
```

Then let the 8 LEDs go out from pin9 to pin2 in turn

```
for (int a = 9; a >= 2; a--)
{
  digitalWrite(a, LOW); //turn this led off
  delay(100);//wait for 100 ms
}
```

Finally, use the same way to turn on the 8 LEDs from pin9 to pin2 in turn and let them go out in turn.

```
for (int a = 9; a >= 2; a--)
{
  digitalWrite(a, HIGH); //turn this led on
  delay(100);//wait for 100 ms
}
for (int a = 2; a <= 9; a++)
{
  digitalWrite(a, LOW); //turn this led off
  delay(100);//wait for 100 ms
}
```

Experiment Summary

Through this experiment, you have learned how to use for() statement which is a very useful statement when you want to short the code.

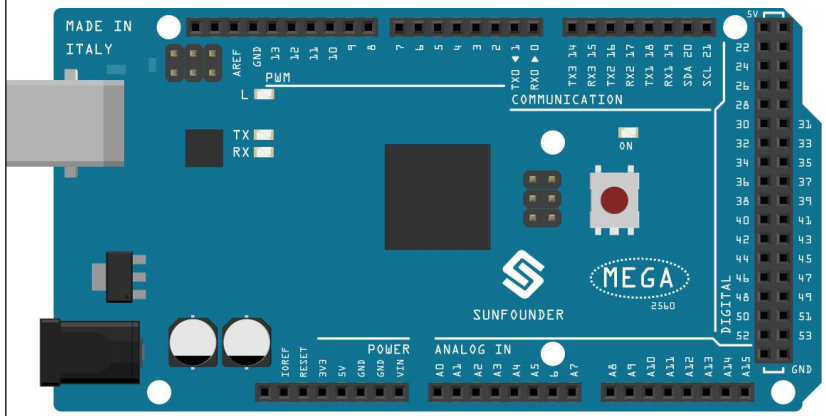
Lesson 5 Controlling LED by Button

Introduction

In this experiment, we will learn how to turn on/off an LED by using an I/O port and a button. The "I/O port" refers to the INPUT and OUTPUT port. Here the INPUT port of the Mega 2560 board is used to read the output of an external device. Since the board itself has an LED (connected to Pin 13), you can use this LED to do this experiment for convenience.

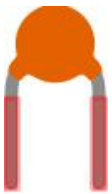
Components

1 * Mega 2560 Board



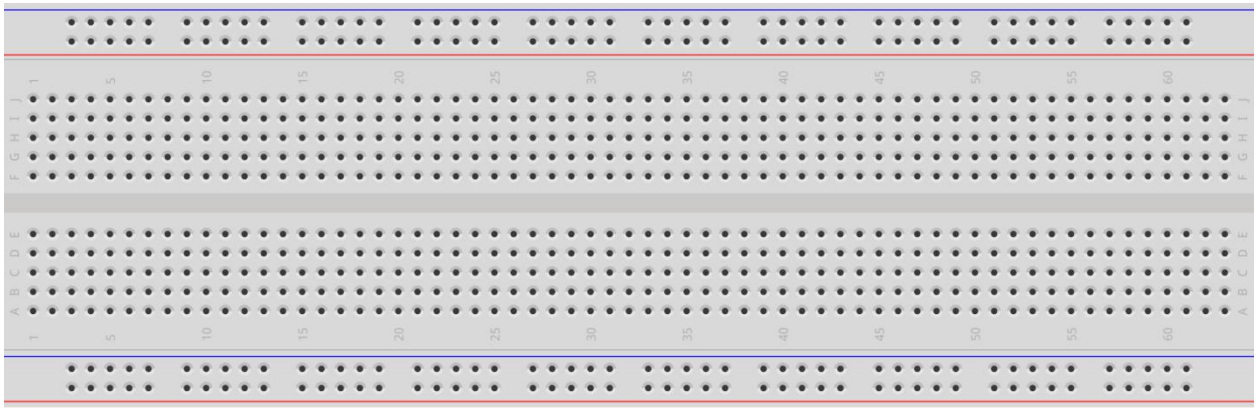
1 * Resistor (10kΩ)

1 * 104 Capacitor



1 * Button

1 * Breadboard



1 * USB cable

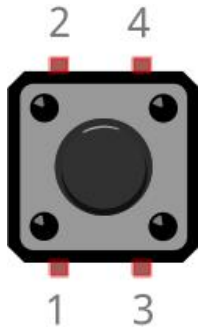
Several jumper wires

Experimental Principle

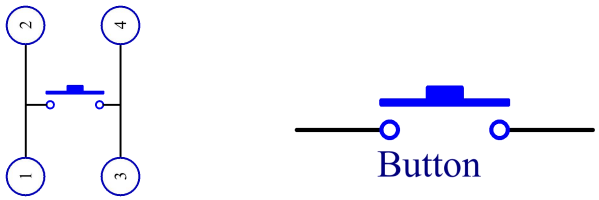
Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or break circuits. Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures.

Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.



The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.

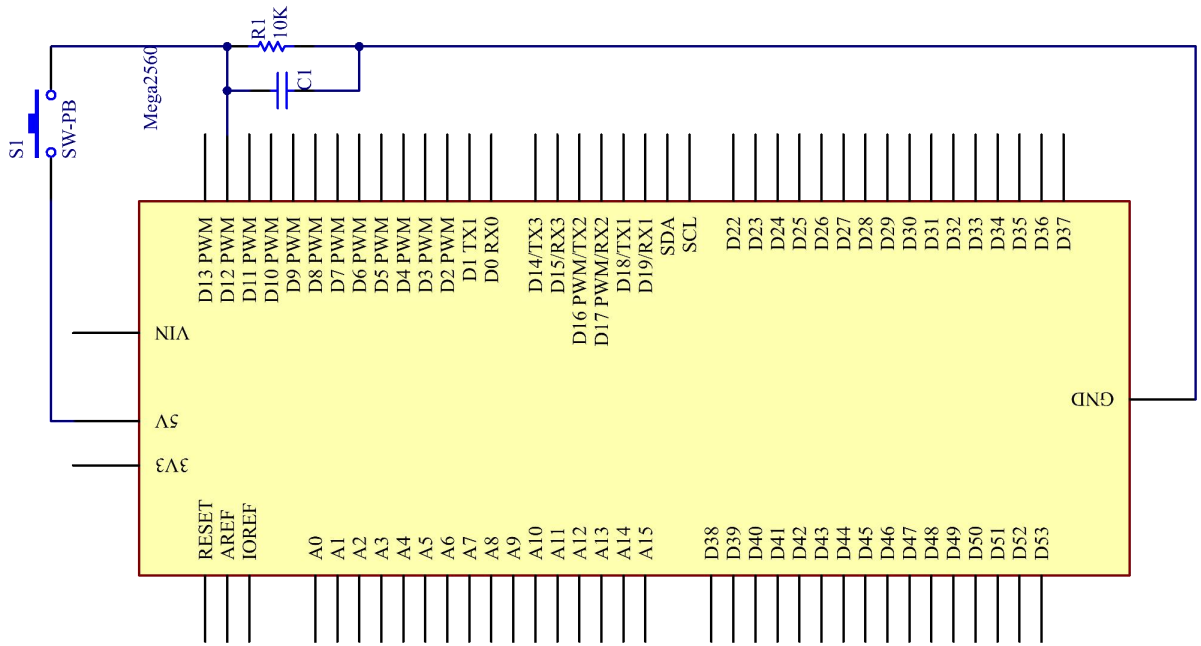


Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.

Principle:

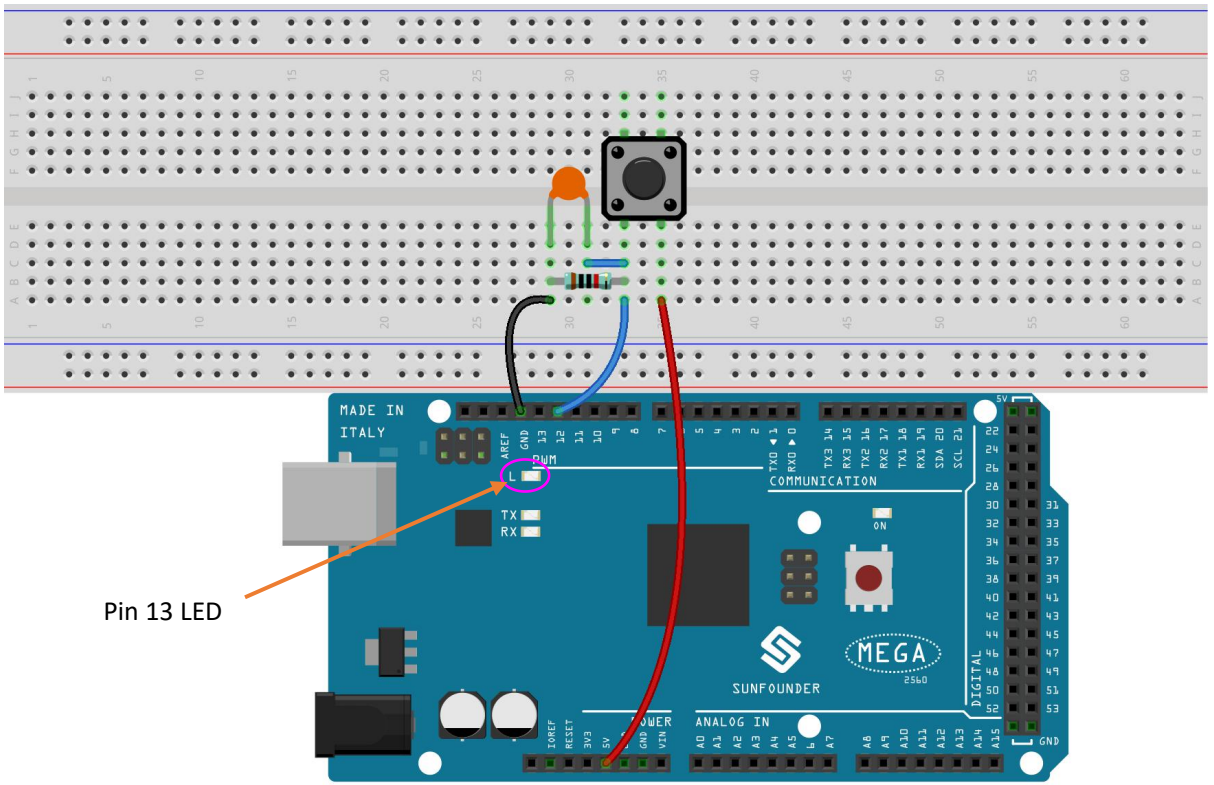
Connect one end of the buttons to pin 12 which connects with a pull-down resistor and a 0.1uF (104) capacitor (to eliminate jitter and output a stable level when the button is working). Connect the other end of the resistor to GND and one of the pins at the other end of the button to 5V. When the button is pressed, pin 12 is 5V (HIGH). Set the pin 12 as High level by programming and pin 13 (integrated with an LED) as High at the same time. Then release the button (pin 12 changes to LOW) and pin 13 is Low. So we will see the LED lights up and goes out alternately as the button is pressed and released.

The schematic diagram：



Experimental Procedures

Step 1: Build the circuit

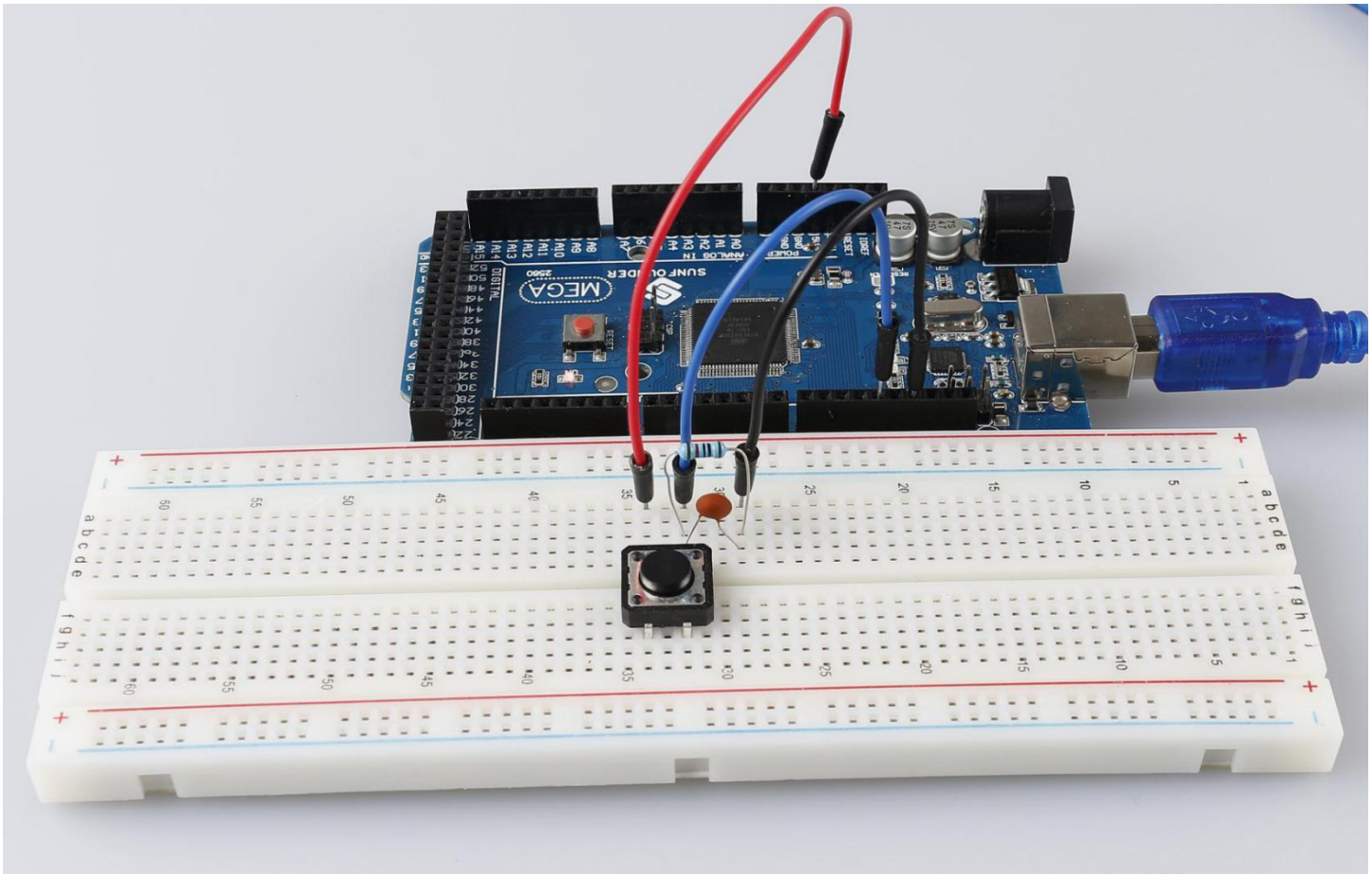


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, press the button, and the LED on the mega2560 board will light up.



Code Analysis

Code Analysis 5-1 Define variables

```
const int buttonPin = 12; //the button connect to pin 12
const int ledPin = 13;//the led connect to pin13
int buttonState = 0;         // variable for reading the pushbutton status
```

Connect the button to pin 12. LED has been connected to pin 13. Define a variable *buttonState* to restore the state of the button.

Code Analysis 5-2 Set the input and output status of the pins

```
pinMode(buttonPin, INPUT); //initialize thebuttonPin as input
pinMode(ledPin, OUTPUT); //initialize the led pin as output
```

We need to know the status of the button in this experiment, so here set the *buttonPin* as INPUT; to set HIGH/LOW of the LED, we set *LedPin* as OUTPUT.

Code Analysis 5-3 Read the status of the button

```
buttonState = digitalRead(buttonPin);
```

buttonPin(Pin12) is a digital pin; here is to read the value of the button and store it in *buttonState*.

digitalRead (Pin): Reads the value from a specified digital pin, either HIGH or LOW.

Code Analysis 5-4 Turn on the LED when the button is pressed

```
if (buttonState == HIGH )
{
  digitalWrite(ledPin, HIGH); //turn the led on
}
else
{
  digitalWrite(ledPin, LOW); //turn the led off
}
```

In this part, when the ***buttonState*** is High level, write *ledPin* as High and the LED will be turned on. As one end of the button has been connected to 5V and the other end to pin 12, when the button is pressed, pin 12 is 5V (HIGH). And then determine with the *if*(conditional); if the conditional is true, then the LED will light up.

Else means that when the if(conditional) is determined as false, run the code in *else*.

Experiment Summary

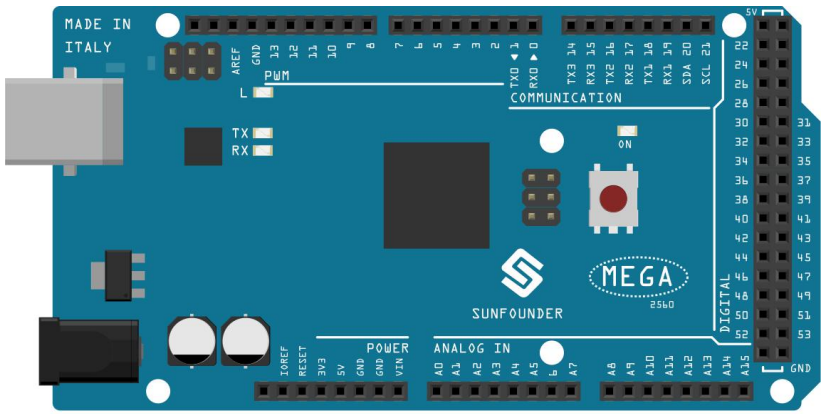
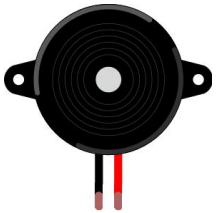
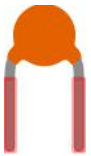
You can also change the code to: when the button is pressed, if (buttonState=HIGH). The LED goes out (digitalWrite(ledPin, LOW)). When the button is released (the else), the LED lights up ((digitalWrite(ledPin, HIGH)). You only need to replace the code in **if** with those in **else**.

Lesson 6 Doorbell

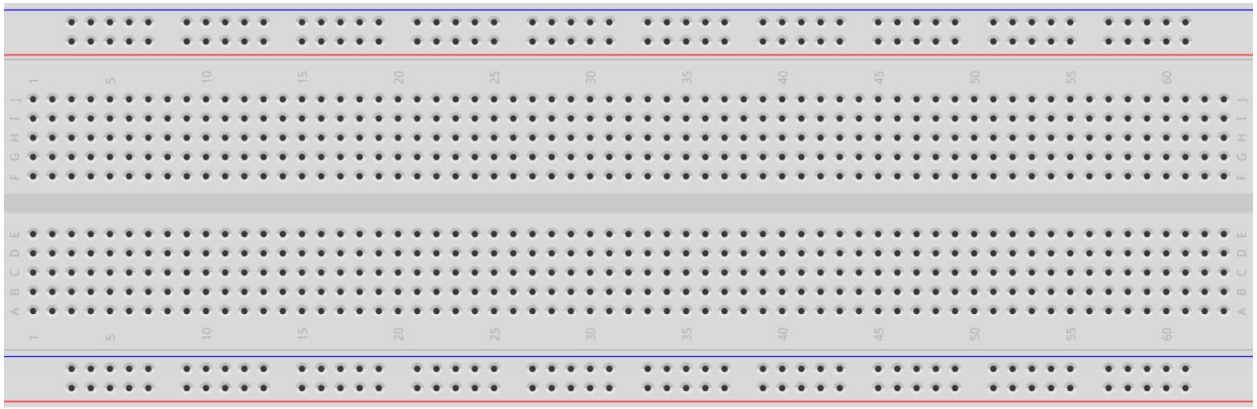
Introduction

A buzzer is a great tool in your experiments whenever you want to make some sounds. In this lesson, we will learn how to drive an active buzzer to build a simple doorbell.

Components

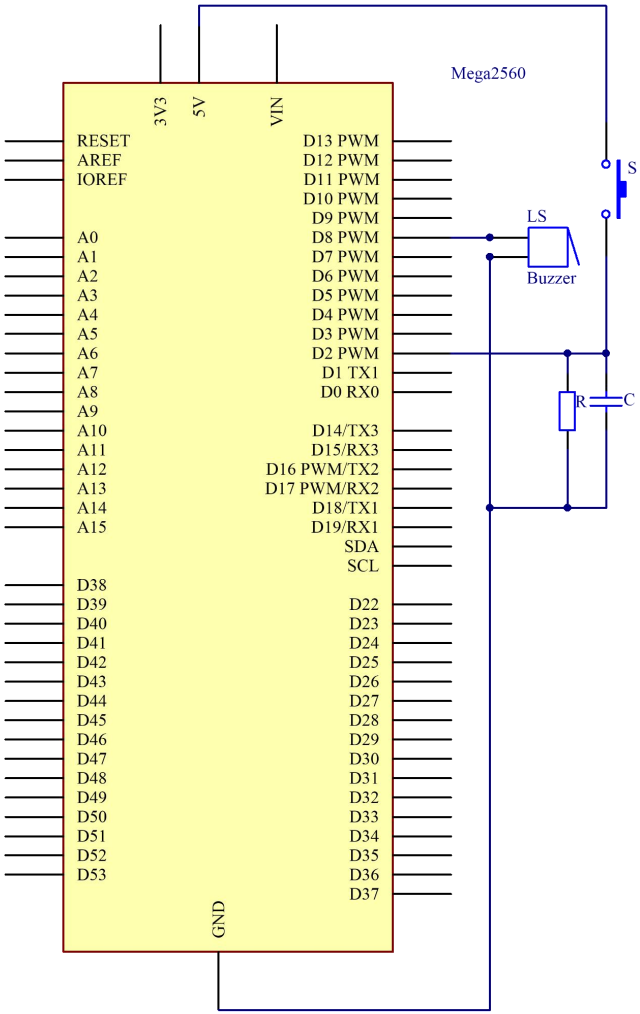
1 * Mega 2560 Board 	1*Buzzer(Active) 	1 * Button 
1 * 104 Capacitor 	1 * Resistor (10kΩ) 	

1 * Breadboard



1 * USB cable

Several jumper wires



Experimental Principle

As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive ones (see the following picture). Turn the pins of two buzzers face up, and the one with a green circuit board is a passive buzzer, while the other enclosed with a black tape is an active one.

The difference between an active buzzer and a passive buzzer:

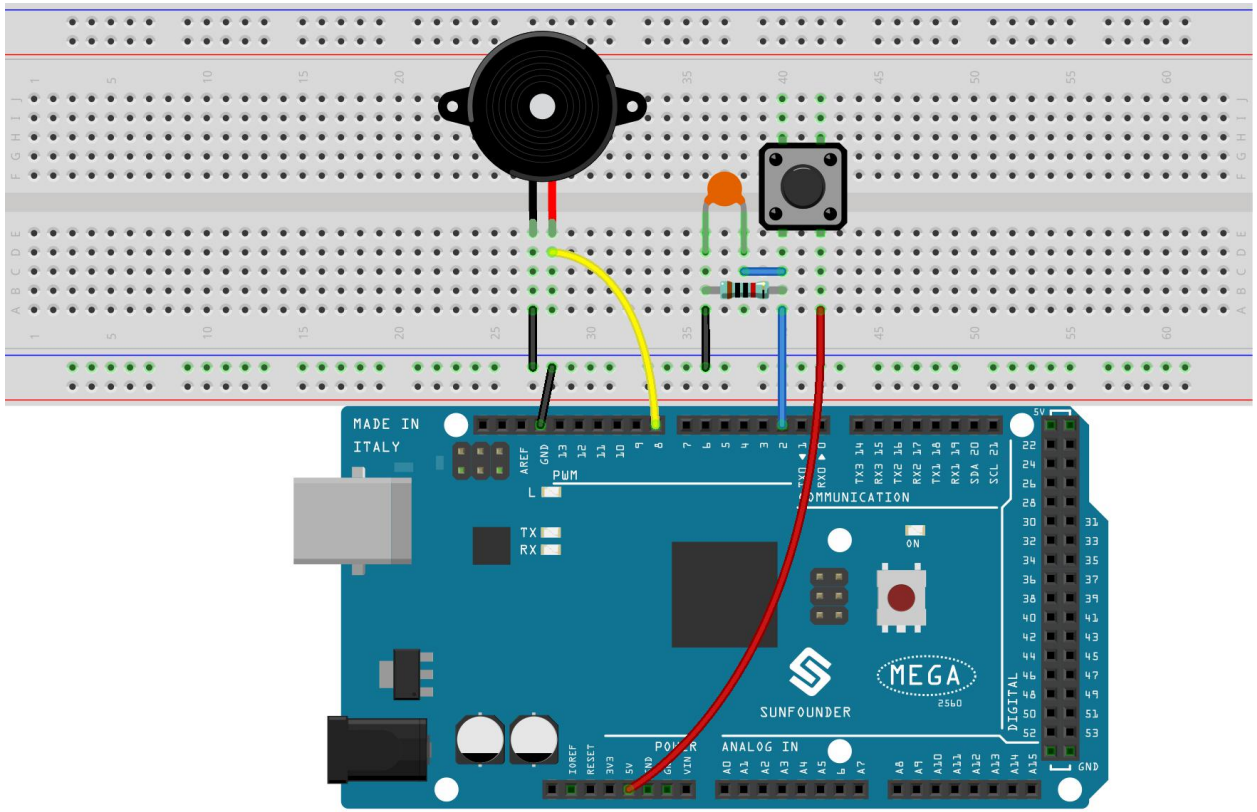
An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not tweet if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

In this experiment, we use an active buzzer.

The schematic diagram：

Experimental Procedures

Step 1: Build the circuit (Long pins of buzzer is the Anode and the short pin is Cathode).

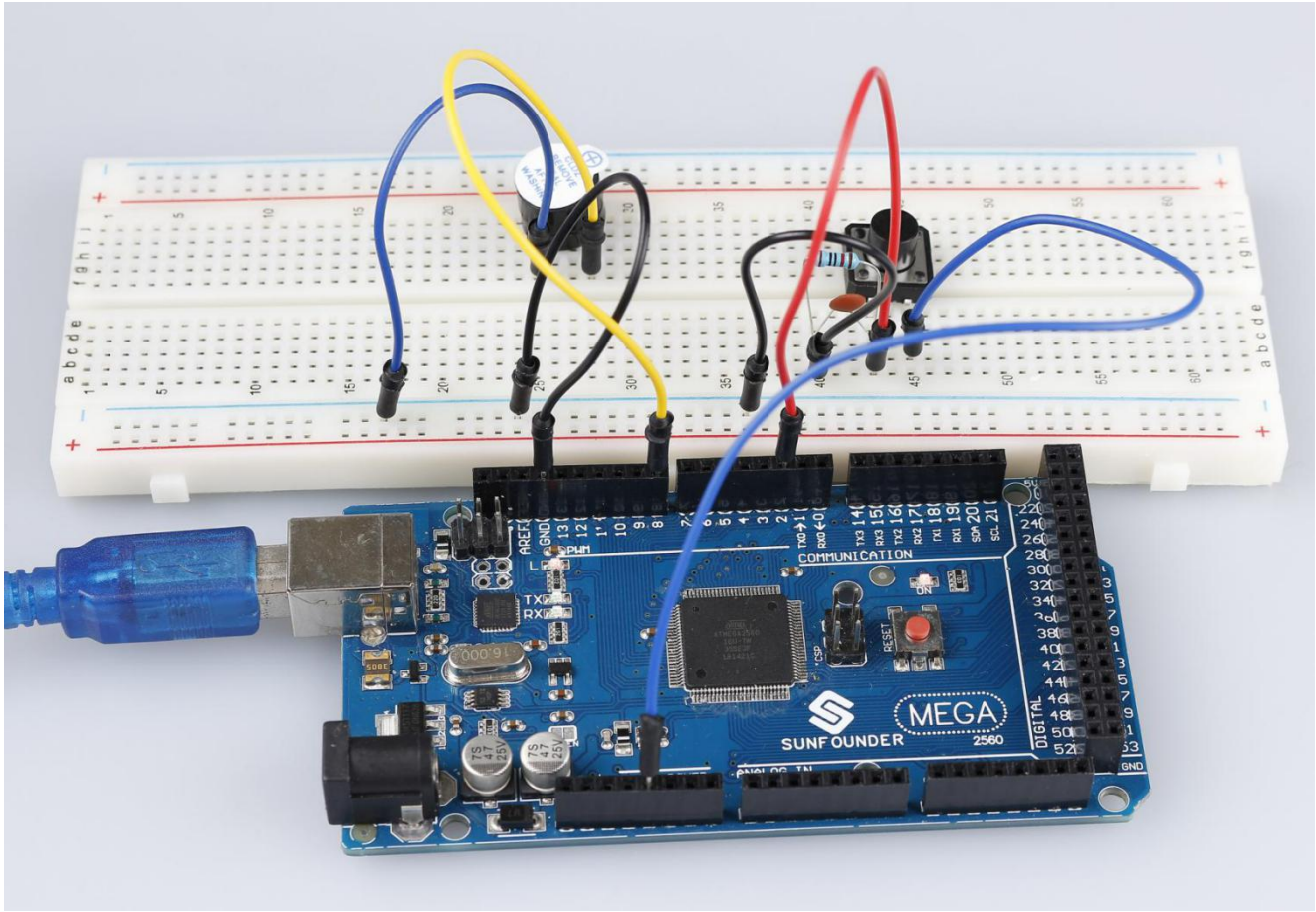


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you should hear the buzzer beep.



Code Analysis

Code Analysis 6-1 Define variables

```
const int buttonPin = 2; //the button connect to pin2
const int buzzerPin = 8;//the led connect to pin8
/**********************************/
int buttonState = 0;         // variable for reading the pushbutton status
```

Connect the button to pin 2 and buzzer to pin 8. Define a variable *buttonState* to restore the state of the button.

Code Analysis 6-2 Set the input and output status of the pins

```
pinMode(buttonPin, INPUT); //initialize the buttonPin as input
pinMode(buzzerPin, OUTPUT); //initialize the buzzerpin as output
```

We need to know the status of the button in this experiment, so here set the *buttonPin* as INPUT; to set HIGH/LOW of the buzzer, we set *buzzerPin* as OUTPUT.

Code Analysis 6-3 Read the status of the button

```
buttonState = digitalRead(buttonPin);
```

buttonPin(Pin2) is a digital pin; here is to read the value of the button and store it in *buttonState*.

digitalRead (Pin): Reads the value from a specified digital pin, either HIGH or LOW.

Code Analysis 6-4 Turn on the LED when the button is pressed

```
if (buttonState == HIGH )
{
  for (i = 0; i < 50; i++)
  {
    digitalWrite(buzzerPin, HIGH);
    delay(3);//wait for 1ms
    digitalWrite(buzzerPin, LOW);
    delay(3);//wait for 1ms
  }
   for (i = 0; i < 80; i++)
  {
    digitalWrite(buzzerPin, HIGH);
    delay(5);//wait for 1ms
    digitalWrite(buzzerPin, LOW);
    delay(5);//wait for 1ms
  }
}
```

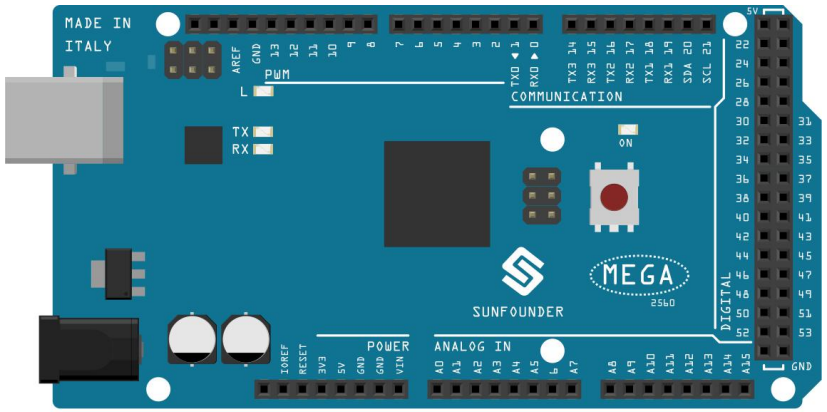
In this part, when the ***buttonState*** is High level, then let the buzzer beeping in different frequency which can simulate the doorbell.

Lesson 7 Tilt Switch

Introduction

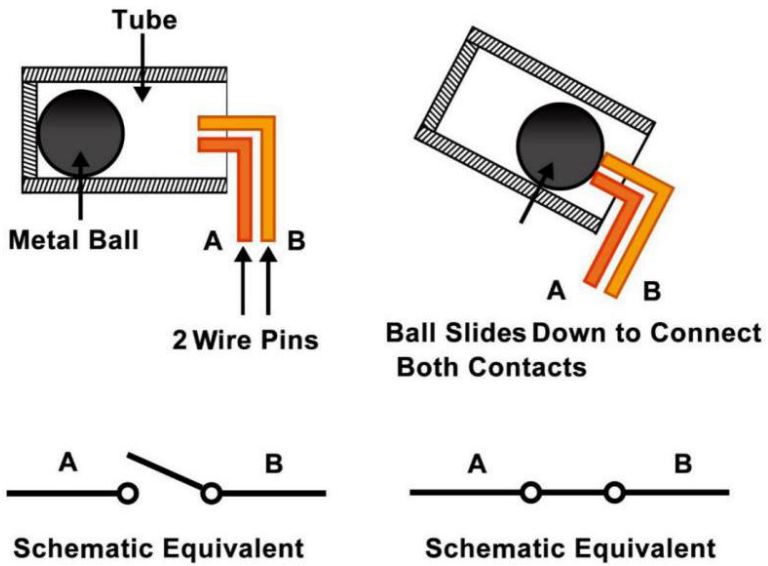
The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

Components

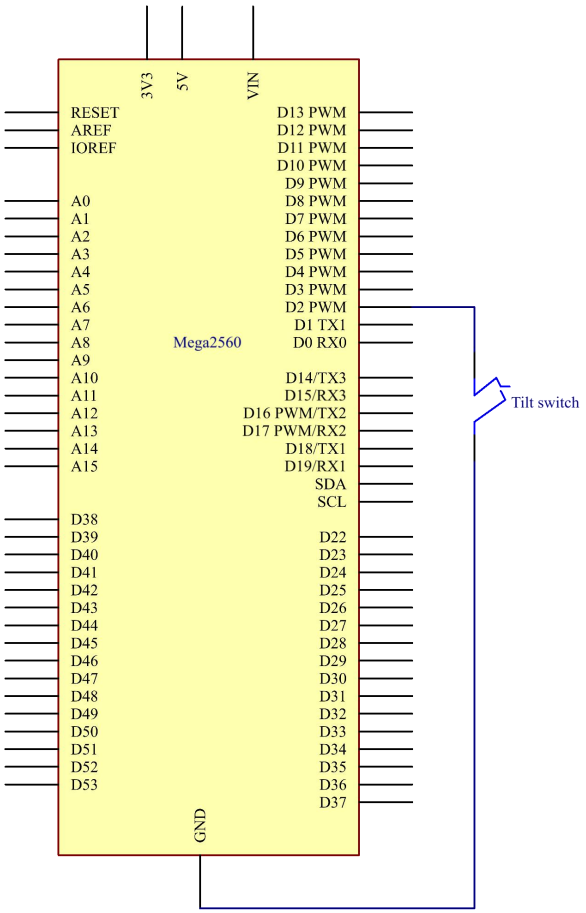
1 * Mega 2560 Board 	1*Tilt Switch 
1 * USB cable 	2 * Dupont wires 

Experimental Principle

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.

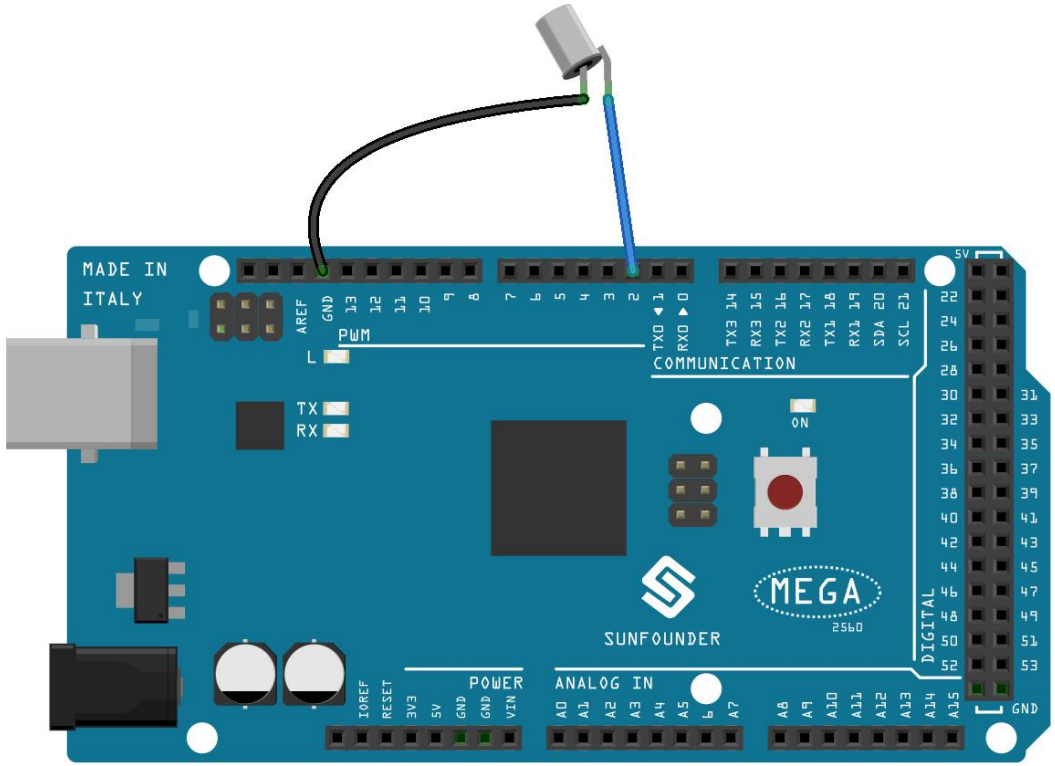


The schematic diagram:



Experimental Procedures

Step 1: Build the circuit

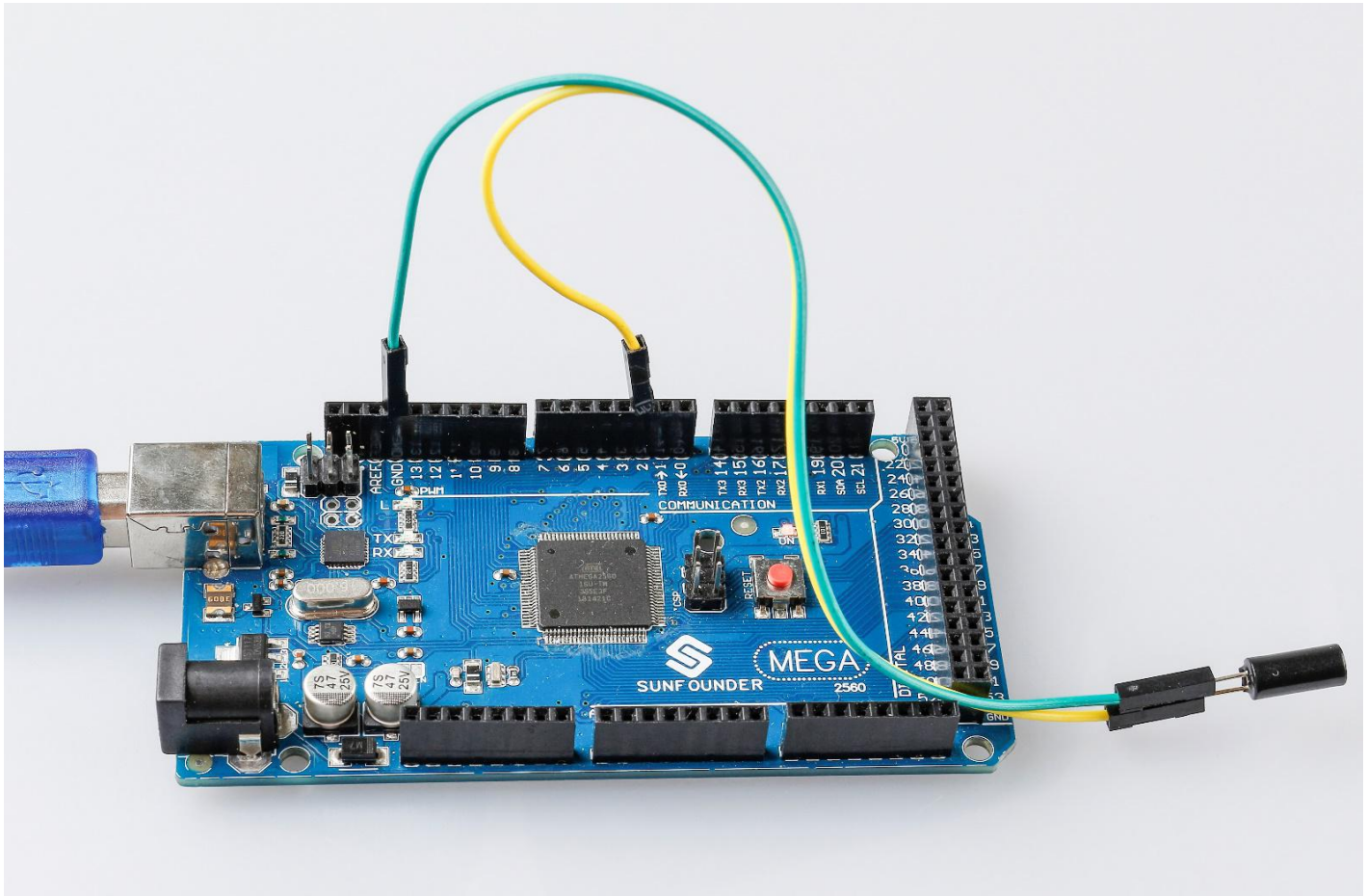


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, tilt the switch, and the LED attached to pin 13 on Mega 2560 board will light up.


COMMUNICATION
MEGA
SUNFOUNDER
2560

Code Analysis

Code Analysis 7-1 Whole Code

```
const int ledPin = 13;//the led attach to

void setup()
{
  pinMode(ledPin,OUTPUT);//initialize the ledPin as an output
  pinMode(2,INPUT);//set pin2 as INPUT
  digitalWrite(2, HIGH);//set pin2 as HIGH
}
/*****************************************/
void loop()
{
  int digitalVal = digitalRead(2);//Read the value of pin2
  if(HIGH == digitalVal)//if tilt switch is not breakover
  {
    digitalWrite(ledPin,LOW);//turn the led off
  }
  else ////if tilt switch breakover
  {
    digitalWrite(ledPin,HIGH);//turn the led on
  }
}
```

The whole code are very simple, one pin of the tilt switch is connected to pin2, another pin is connected to GND, when tilt the switch, the two pins of the switch will be connected to GND, then let the LED on the pin13 lights up.

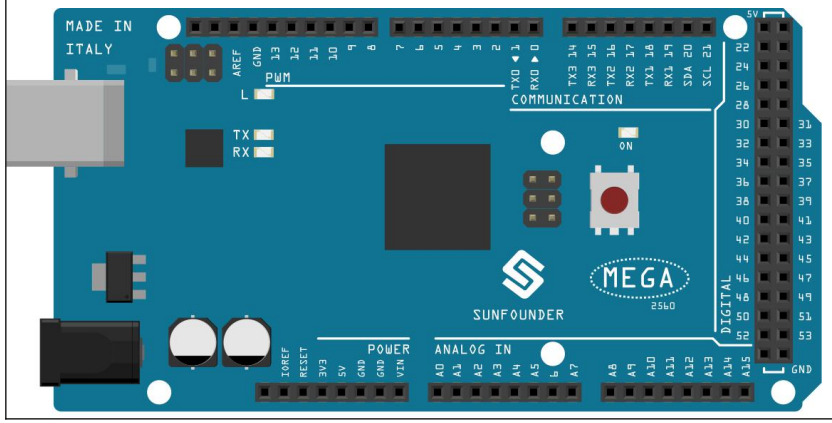
Lesson 8 Relay

Introduction

As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

Components

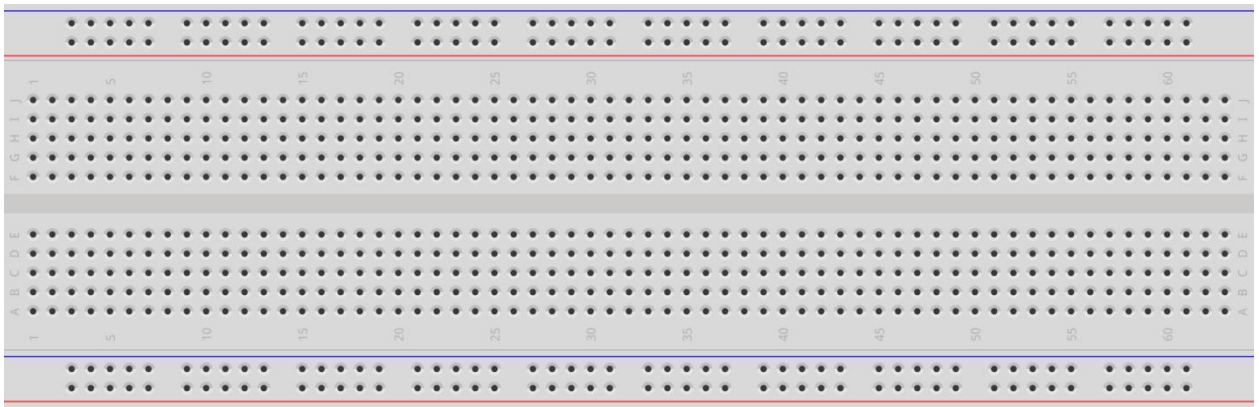
1 * Mega 2560 Board



1 * Resistor (220Ω)

1*Resistor(1kΩ)

1 * LED

1 * NPN Transistor 	1 * Diode1N4007 (Rectifier) 	1* relay 
1 * Breadboard 		
1 * USB cable 	Several jumper wires 	

Experimental Principle

Relay

There are 5 parts in every relay:

1. **Electromagnet** – It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

2. **Armature** – The movable magnetic strip is known as armature. When current flows through them, the coil is it energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

3. **Spring** – When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

4. Set of electrical **contacts** – There are two contact points:

. Normally open - connected when the relay is activated, and disconnected when it is inactive.

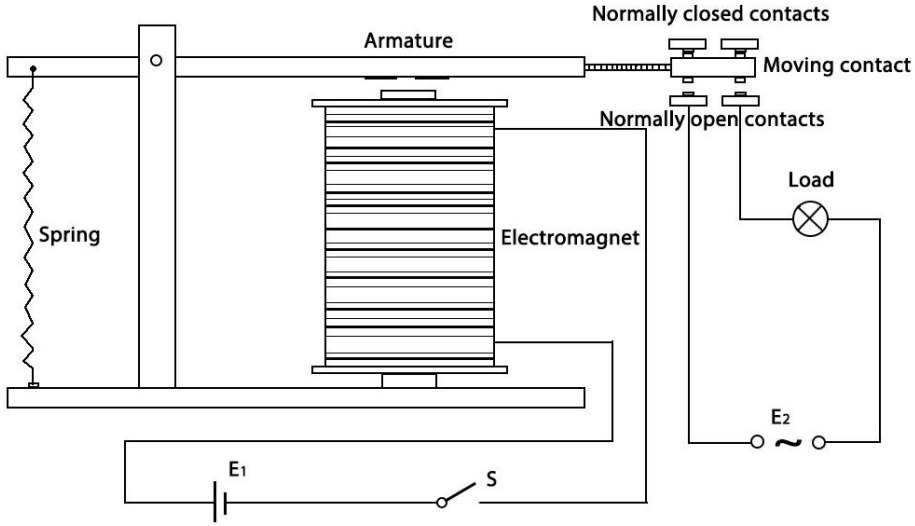
. Normally close – not connected when the relay is activated, and connected when it is inactive.

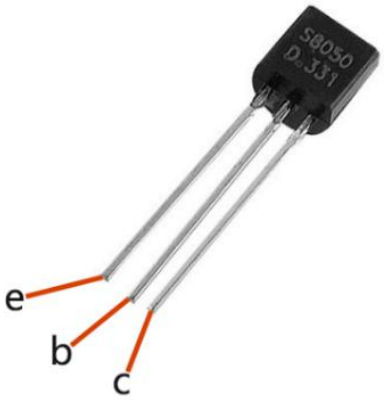
5. Molded frame – Relays are covered with plastic for protection.

Working of Relay

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized.

Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.



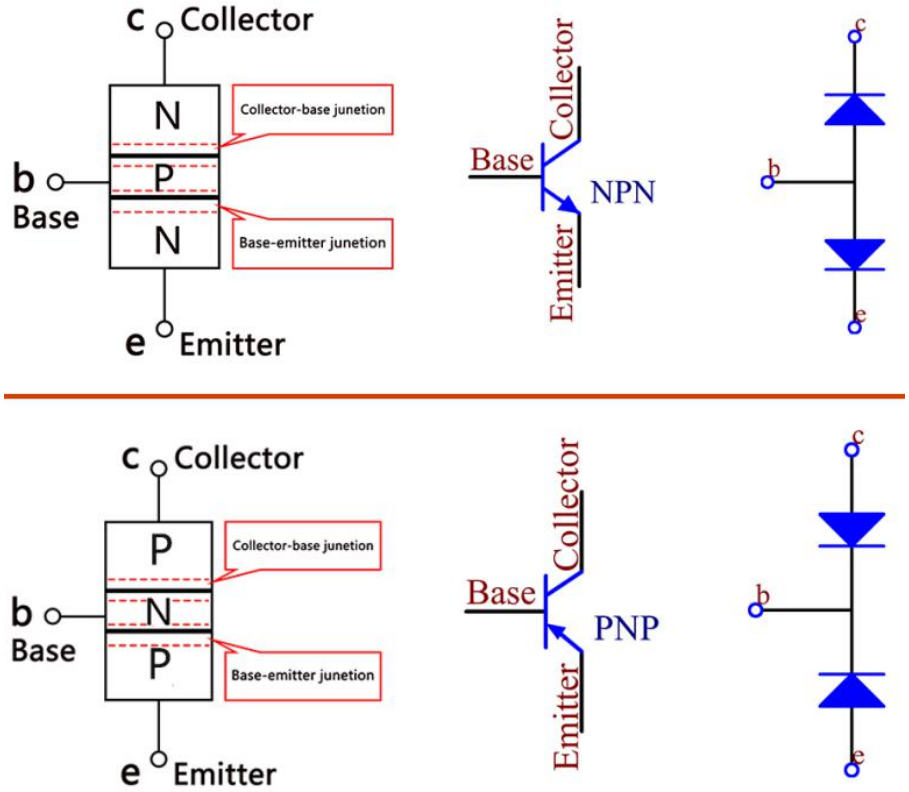


Transistor

Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch. A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier.

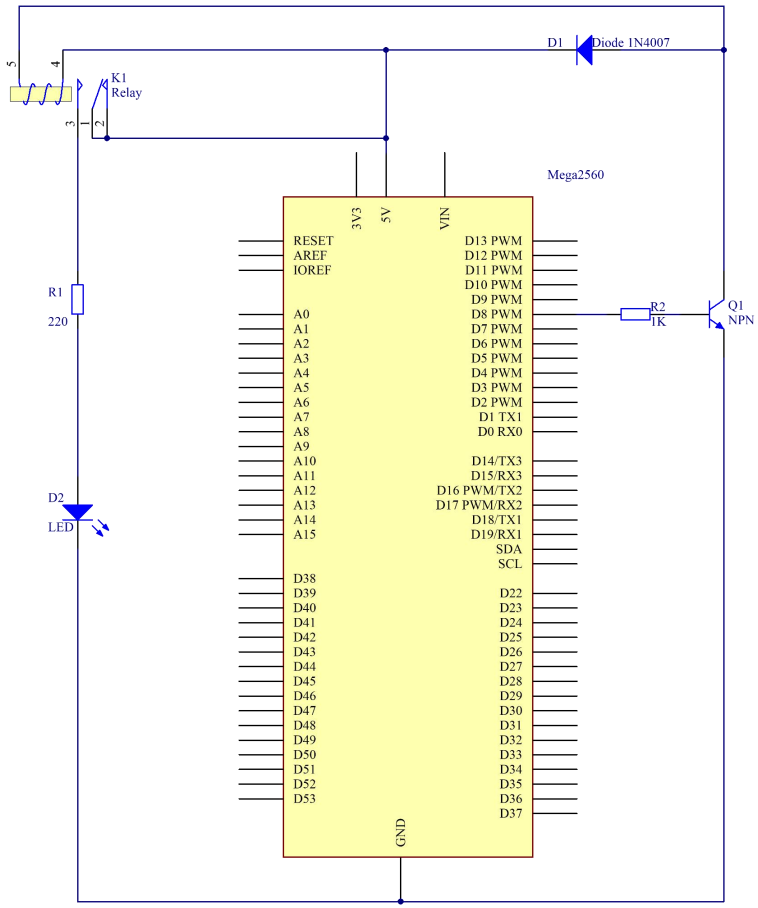
From these three regions, three poles are generated respectively, which are base (b),

emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction. Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.



When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

The schematic diagram:



Principle: Connect a 1K resistor (for current limiting when the transistor is energized) to pin 8 of the SunFounder Mega2560 board, then to an NPN transistor whose collector is connected to the coil of a relay and emitter to GND; connect the normally open contact of the relay to an LED and then GND. Therefore, when a High level signal is given to pin 8, the transistor is energized, thus making the coil of the relay conductive. Then its normally open contact is closed, and the LED will light up. When pin 8 is given a Low level, the LED will stay dim.

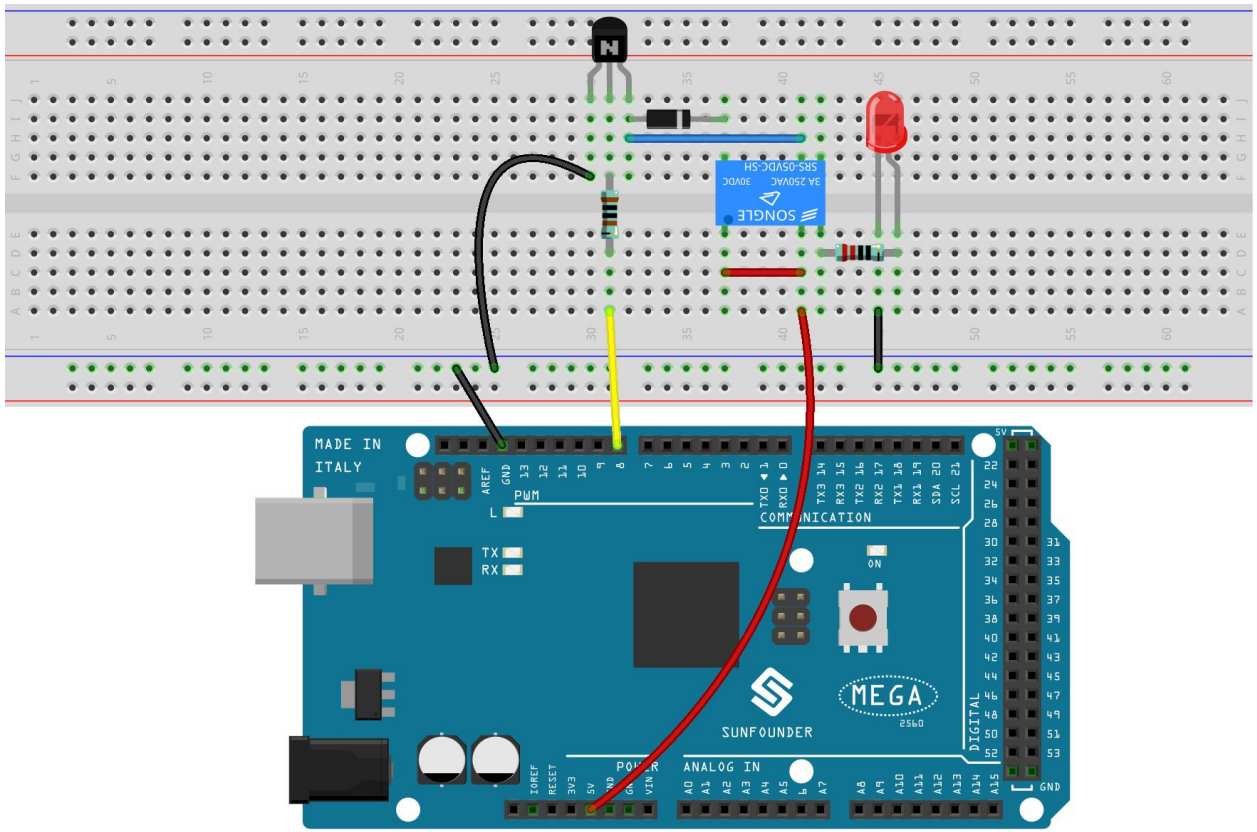
Function of the freewheeling diode: When the voltage input changes from High (5V) to Low (0V), the transistor changes from saturation (three working conditions: amplification, saturation, and cut-off) to cut-off, the current in the coil suddenly has no way to flow through. At this moment, without the freewheeling diode, a counter-electromotive force (EMF) will be generated at the ends of the coil, with positive at the bottom and negative at the top, a voltage higher than 100V. This voltage plus that from the power at the transistor are big enough to burn it. Therefore, the freewheeling diode is extremely important in discharging this counter-EMF in the

direction of the arrow in the figure above, so the voltage of the transistor to GND is no higher than +5V (+0.7V).

In this experiment, when the relay closes, the LED will light up; when the relay opens, the LED will go out.

Experimental Procedures

Step 1: Build the circuit

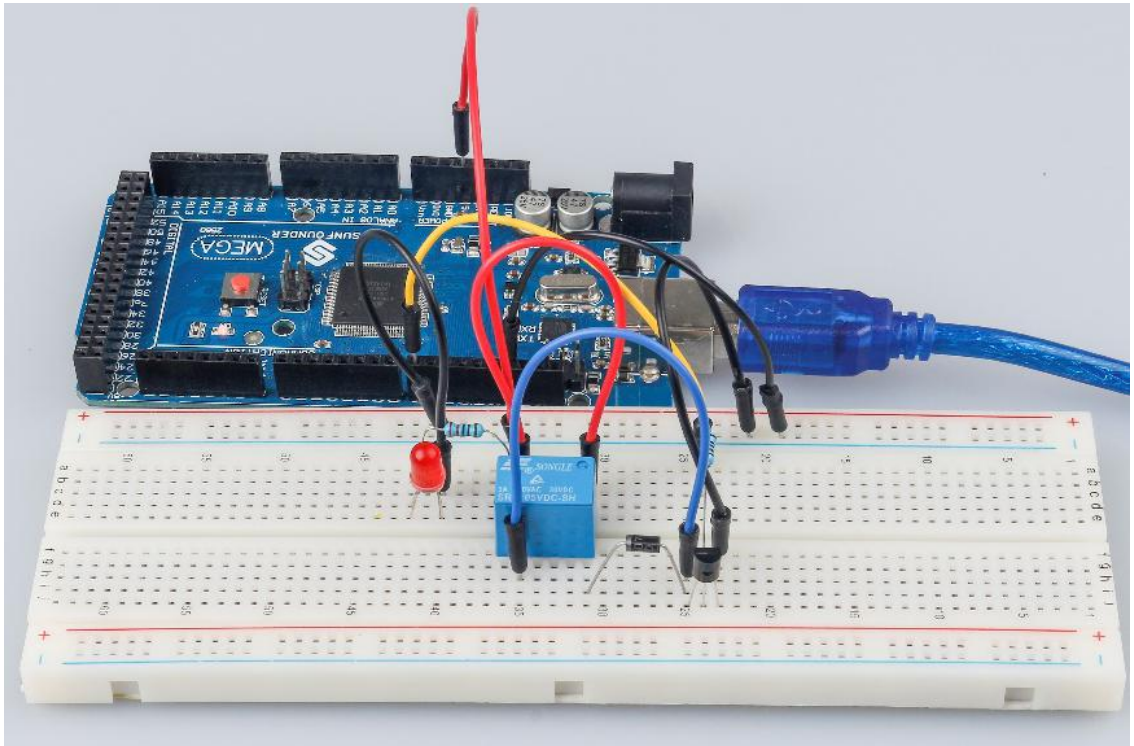


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, send a High level signal, and the relay will close and the LED will light up; send a low one, and it will open and the LED will go out. In addition, you can hear a tick-tock caused by breaking the normally close contact and closing the normally open one.



Code Analysis

```
void loop()
{
  digitalWrite(relayPin, HIGH); //drive relay closure conduction
  delay(1000); //wait for a second
  digitalWrite(relayPin, LOW); //drive the relay is closed off
  delay(1000); //wait for a second
}
```

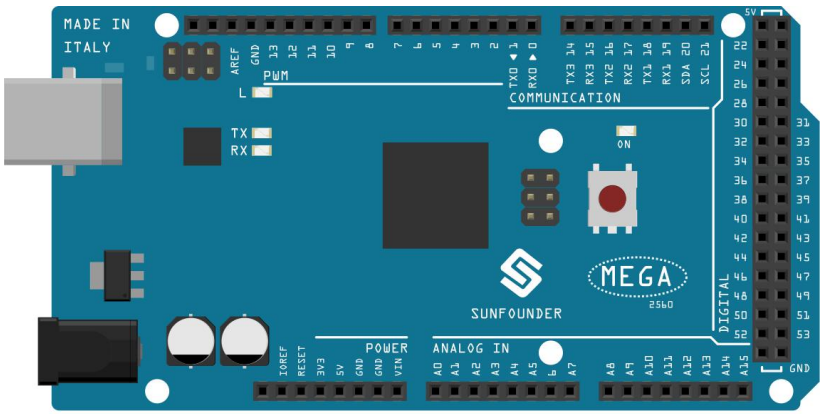
The code in this experiment is simple. First, set *relayPin* as High level and the LED connected to the relay will light up. Then set *relayPin* as Low level, and the LED goes out.

Lesson 9 RGB LED

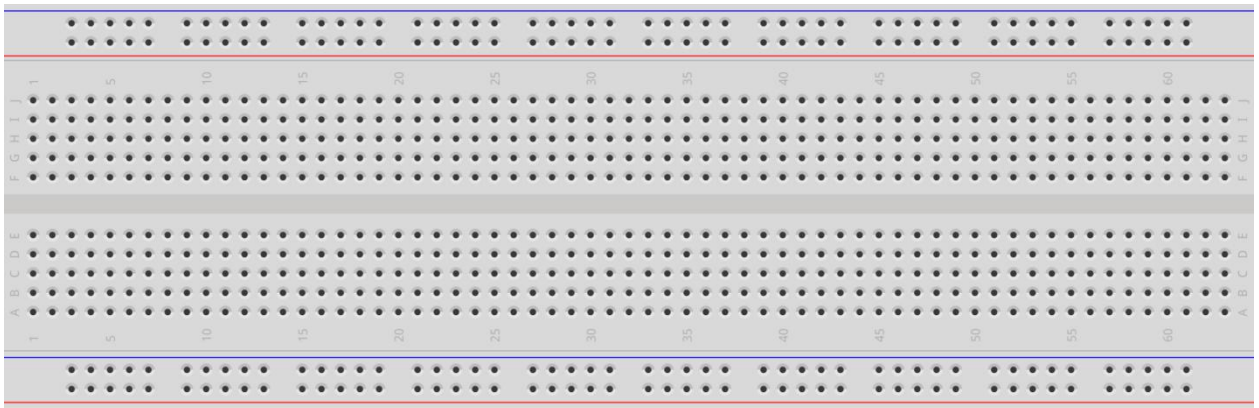
Introduction

Previously we've used the digital pin to control an LED brighten and dim. In this lesson, we will use PWM to control an RGB LED to flash various kinds of color. When different PWM values are set to the R, G, and B pins of the LED, its brightness will be different. When the three different colors are mixed, we can see that the RGB LED flashes different colors.

Components

1 * Mega 2560 Board	3 * Resistor (220Ω)	1 * RGB LED
		

1 * Breadboard



1 * USB cable

Several jumper wires

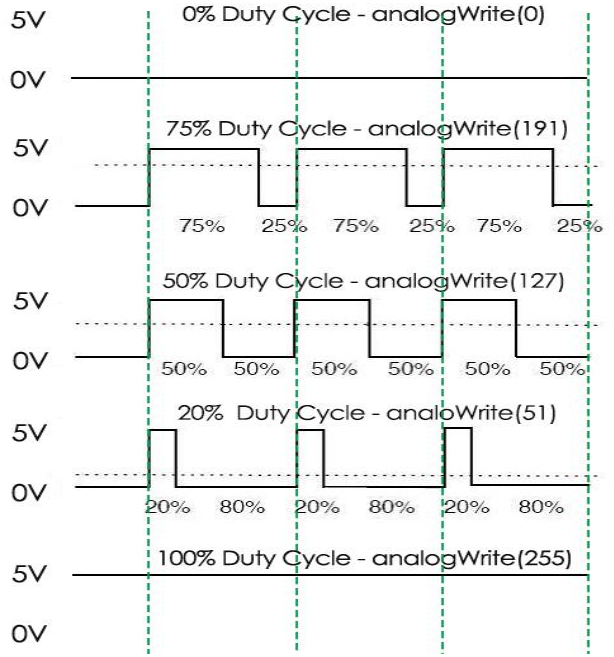
Experimental Principle

PWM

Pulse width modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you change, or modulate, that width. If you repeat this on-off pattern fast enough with some device, an LED for example, it would be like this: the signal is a steady voltage between 0 and 5V controlling the brightness of the LED. (See the PWM description on the

official website of Arduino).

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each.

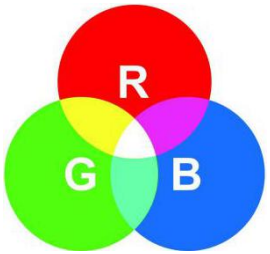


A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

You will find that the smaller the PWM value is, the smaller the value will be after being converted into voltage. Then the LED becomes dimmer accordingly. Therefore, we can control the brightness of the LED by controlling the PWM value.

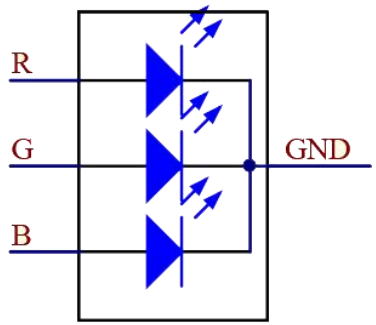
RGB LED

RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216 different colors.



RGB LEDs can be categorized into common anode and common cathode ones. In this experiment, the latter is used. The common cathode, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

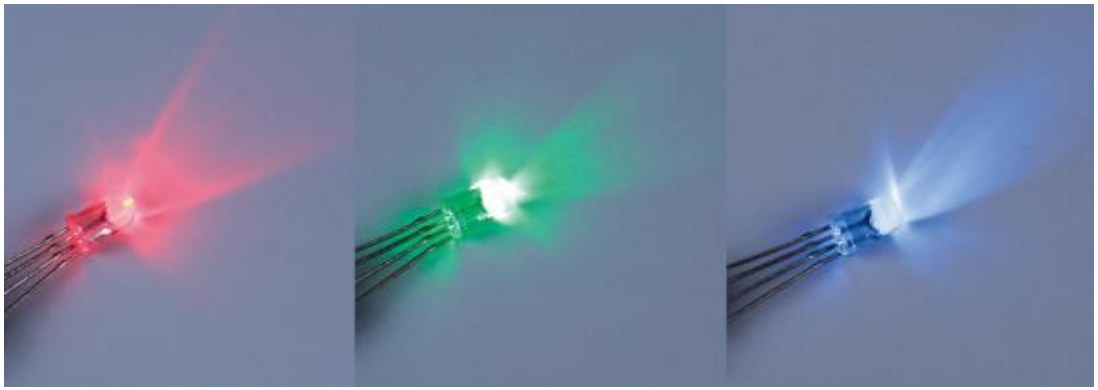
RGB LED



Circuit Symbol

An RGB LED has 4 pins: the longest one is GND; the others are Red, Green and Blue. Touch its plastic shell and you

will find a cut. The pin closest to the cut is the first pin, marked as Red, then GND, Green and Blue in turn.

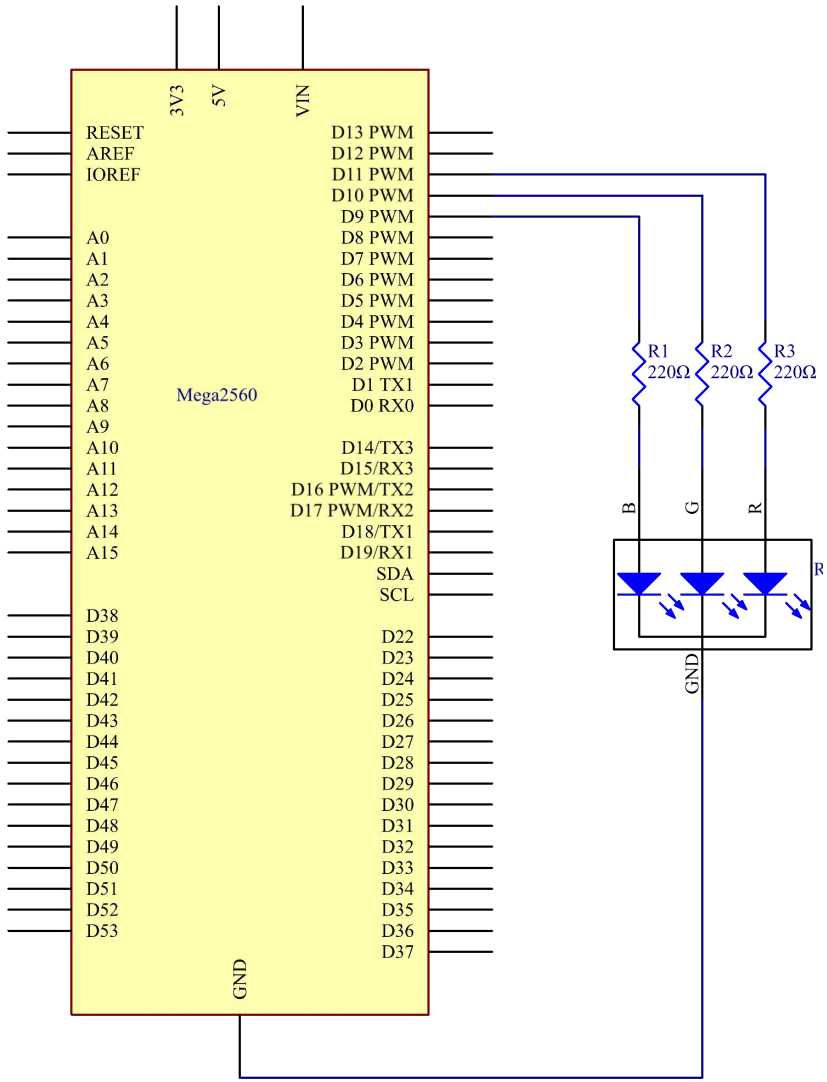


Or you can distinguish them in another way. As GND is the longest one and can be defined directly, you just need to confirm the other three pins. You can test it by giving them a small voltage. The forward voltage drop from the three pins to the GND are respectively 1.8V (red), 2.5V (blue), and 2.3V (green). Thus, when you connect the same current limiting resistor with the three pins and supply them with the same voltage, the red one is the brightest, and then comes the green and the blue one. Therefore, you may need to add a current limiting resistor with different resistances to the three pins for these colors.

Principle:

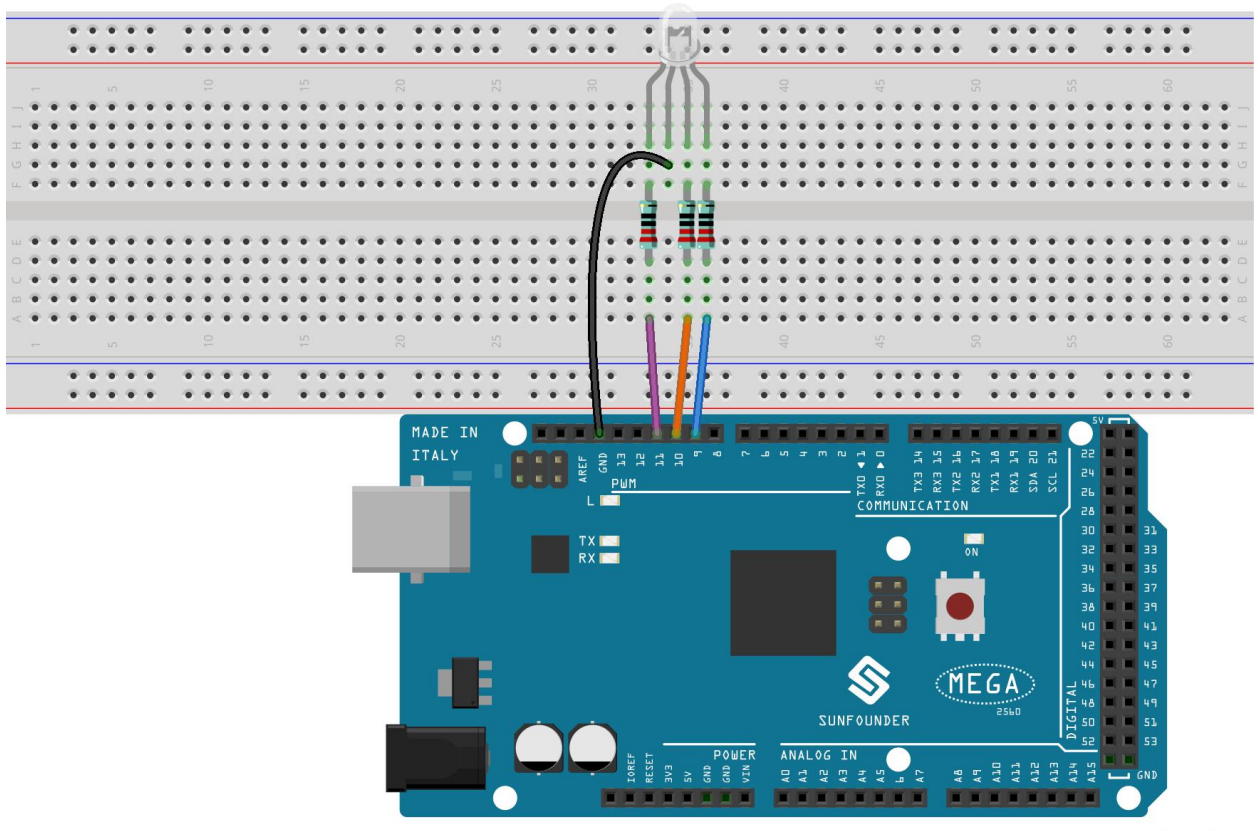
On the Mega2560 board, 2 to 13 and 44 to 46. Provide 8-bit PWM output with the analogWrite() function. You can connect any of these pins.Here we input a value between 0 and 255 to the three pins of the RGB LED to make it display different colors. After connecting the pins of R, G, and B to a current limiting resistor, connect them to the pin 9, pin 10, and pin 11 respectively. The longest pin (GND) of the LED connects to the GND of the Mega 2560. When the three pins are given different PWM values, the RGB LED will display different colors.

The schematic diagram:


3V3
5V
VIN
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D38
D39
D40
D41
D42
D43
D44
D45
D46
D47
D48
D49
D50
D51
D52
D53
GND
Mega2560
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D3 PWM
D2 PWM
D1 TX1
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
SCL
D22
D23
D24
D25
D26
D27
D28
D29
D30
D31
D32
D33
D34
D35
D36
D37
R1
220Ω
R2
220Ω
R3
220Ω
B
G
R
R
GND

Experimental Procedures

Step 1: Build the circuit

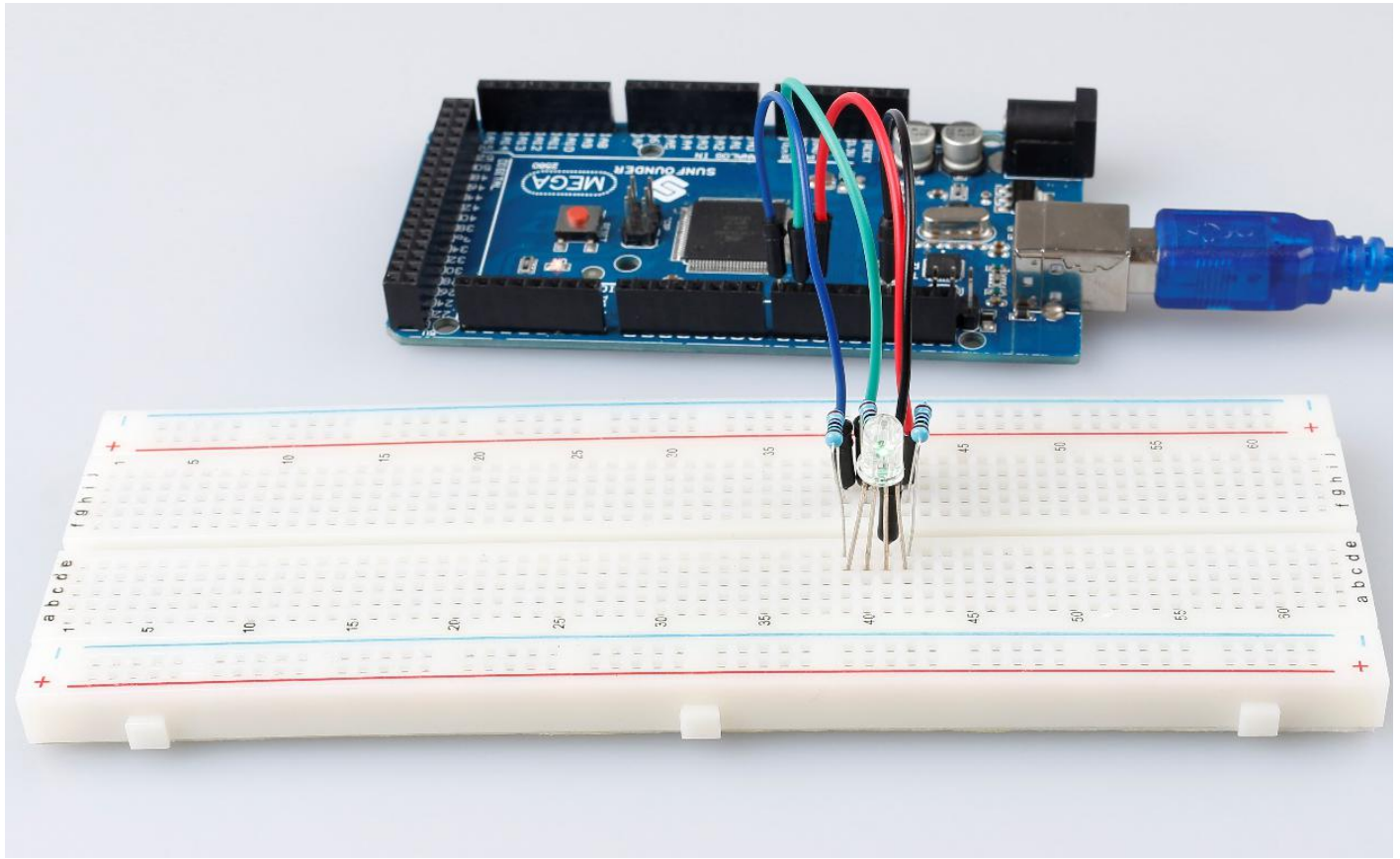


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Here you should see the RGB LED flash circularly red, green, and blue first, then red, orange, yellow, green, blue, indigo, and purple.



Code Analysis

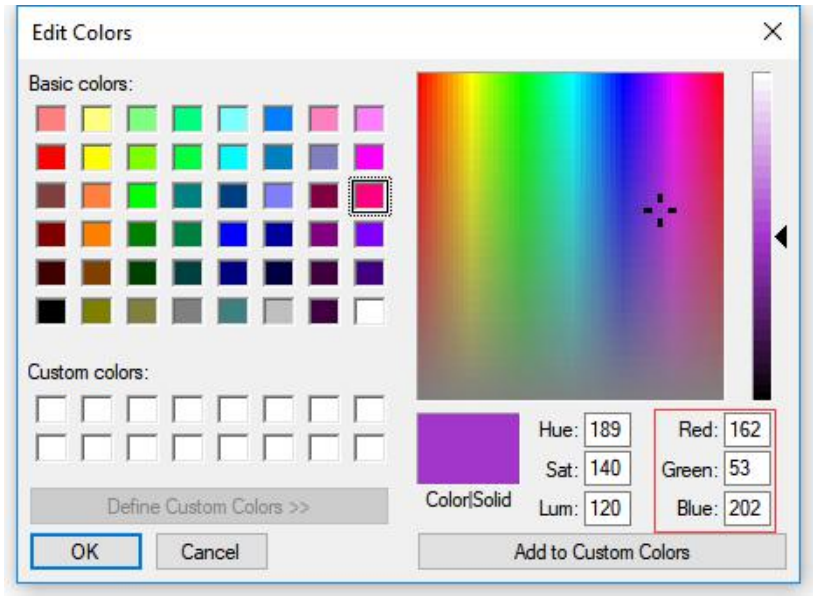
Code Analysis 9-1 Set the color

Here use the *color()* function to set the color of the RGB LED. In the code, it is set to flash 7 different colors.

You can use the paint tool on your computer to get the RGB value.

1) Open the paint tool on your computer and click to Edit colors.

2) Select one color, then you can see the RGB value of this color. Fill them in the code.



```
color(255, 0, 0); // turn the RGB LED red
delay(1000); // delay for 1 second
color(0,255, 0); // turn the RGB LED green
delay(1000); // delay for 1 second
color(0, 0, 255); // turn the RGB LED blue
delay(1000); // delay for 1 second
// Example blended colors:
color(255,0,252); // turn the RGB LED red
delay(1000); // delay for 1 second
color(237,109,0); // turn the RGB LED orange
delay(1000); // delay for 1 second
color(255,215,0); // turn the RGB LED yellow
delay(1000); // delay for 1 second
color(34,139,34); // turn the RGB LED green
delay(1000); // delay for 1 second
color(0,112,255); // turn the RGB LED blue
delay(1000); // delay for 1 second
color(0,46,90); // turn the RGB LED  indigo
delay(1000); // delay for 1 second
color(128,0,128); // turn the RGB LED purple
delay(1000); // delay for 1 second
```

Code Analysis 9-2 color()function

```
void color (unsigned char red, unsigned char green, unsigned char blue)
{
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}
```

Define three unsigned char variables, i.e., red, green and blue. Write their values to *redPin*, *greenPin* and *bluePin*. For example, color(128,0,128) is to write 128 to *redPin*, 0 to *greenPin* and 128 to *bluePin*. Then the result is the LED flashing purple.

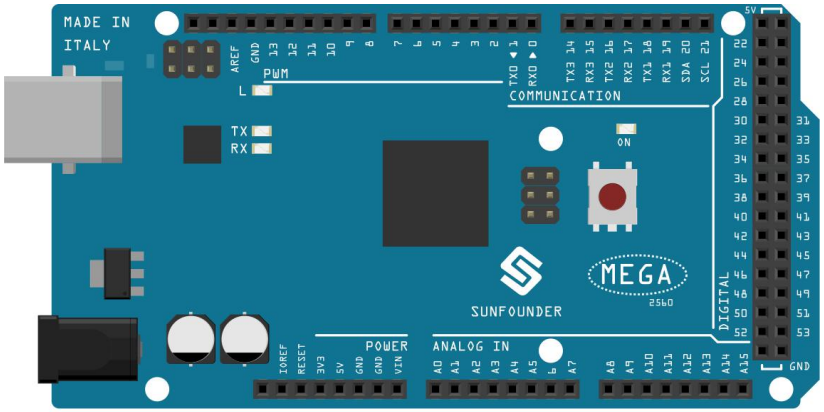
analogWrite(): Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *pinMode()* to set the pin as output before calling *analogWrite()*.

Lesson 10 Controlling an LED by Potentiometer

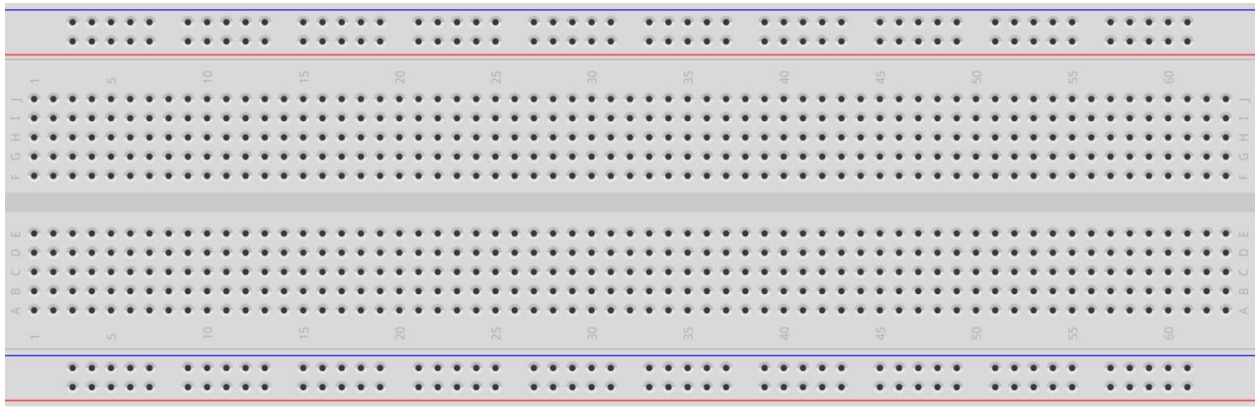
Introduction

In this lesson, let's see how to change the luminance of an LED by a potentiometer, and receive the data of the potentiometer in Serial Monitor to see its value change.

Components

1 * Mega 2560 Board 	1 * Resistor (220Ω) 	1 * LED 
	1* Potentiometer 	

1 * Breadboard



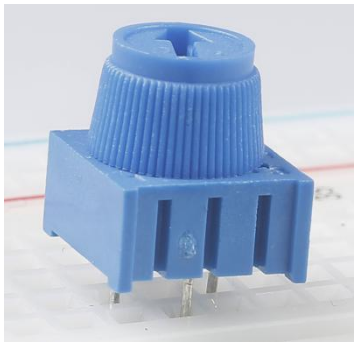
1 * USB cable

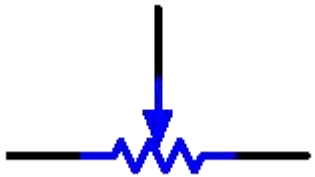
Several jumper wires

Experimental Principle

Potentiometer

Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation. Potentiometer usually consists of resistor and movable brush. When the brush is moving along the resistor, there is a certain resistance or voltage output depending on the displacement.





The functions of the potentiometer in the circuit are as follows:

1. Serving as a voltage divider

   Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

2. Serving as a rheostat

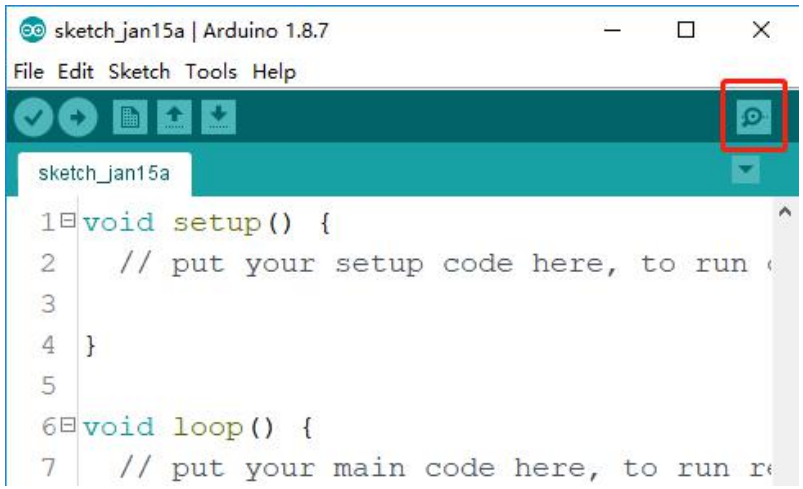
When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

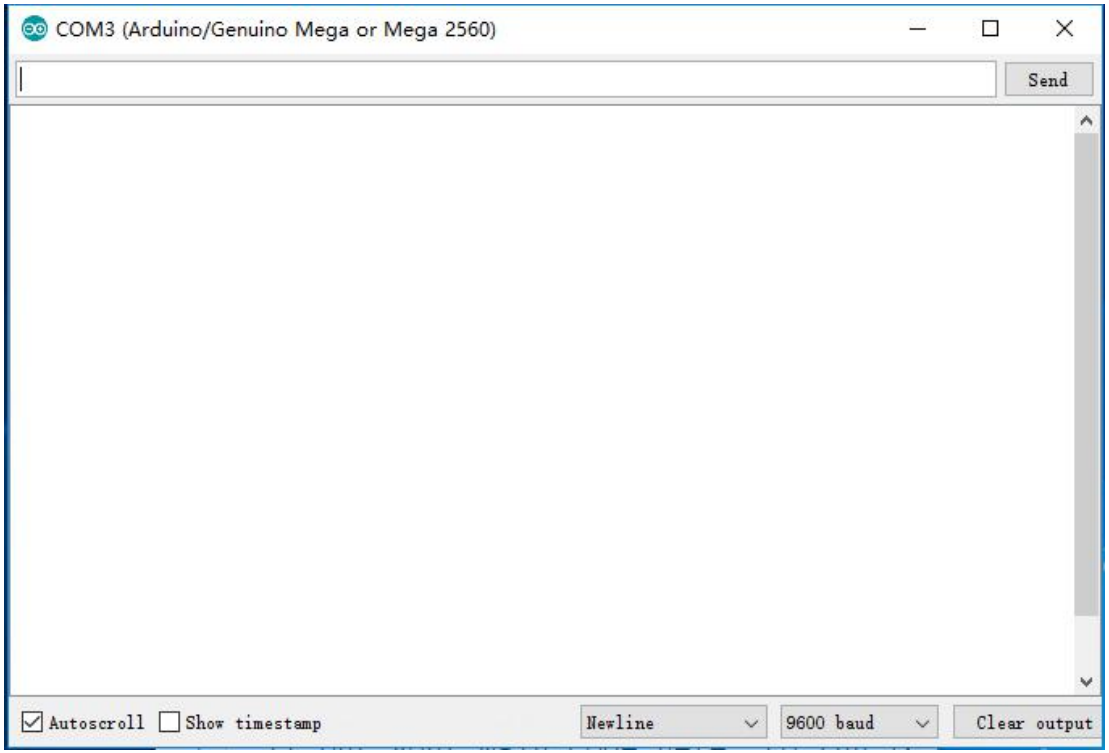
Serial Monitor

Serial Monitor is used for communication between the Mega 2560 board and a computer or other devices. It is a built-in software in the Arduino environment and you can click the button on the upper right corner to open it. You can send and receive data via the serial port on the control board and control the board by input from the keyboard.



Here, the Serial Monitor serves as a transfer station for communication between your computer and the Mega 2560 board. First, the computer transfers data to the Serial Monitor, and then the data is read by the Mega 2560 board.

Finally, the Mega 2560 will perform related operations. Click the icon at the top right corner and a window will pop up as shown below:

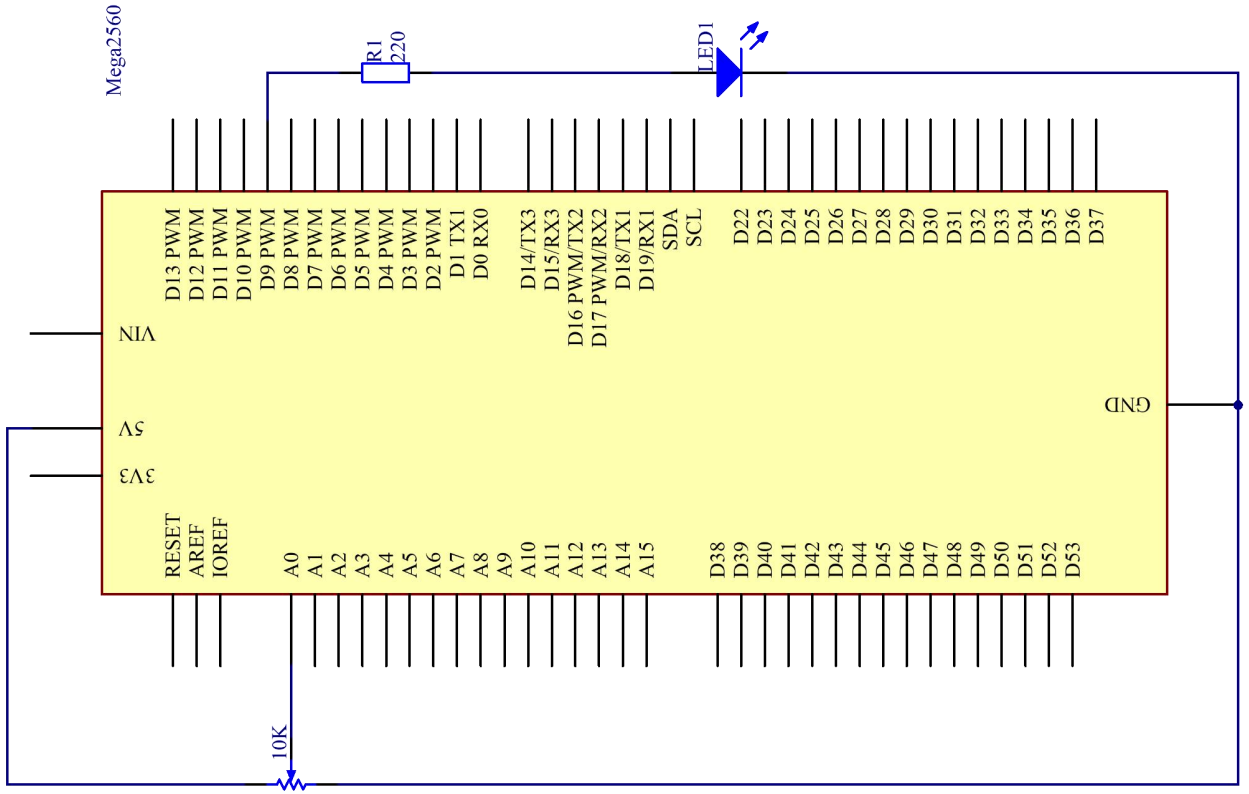


Analog V.S. Digital

A linear potentiometer is an analog electronic component. So what's the difference between an analog value and a digital one? Simply put, digital means on/off, high/low level with just two states, i.e. either 0 or 1. But the data state of analog signals is linear, for example, from 1 to 1000; the signal value changes over time instead of indicating an exact number. Analog signals include those of light intensity, humidity, temperature, and so on.

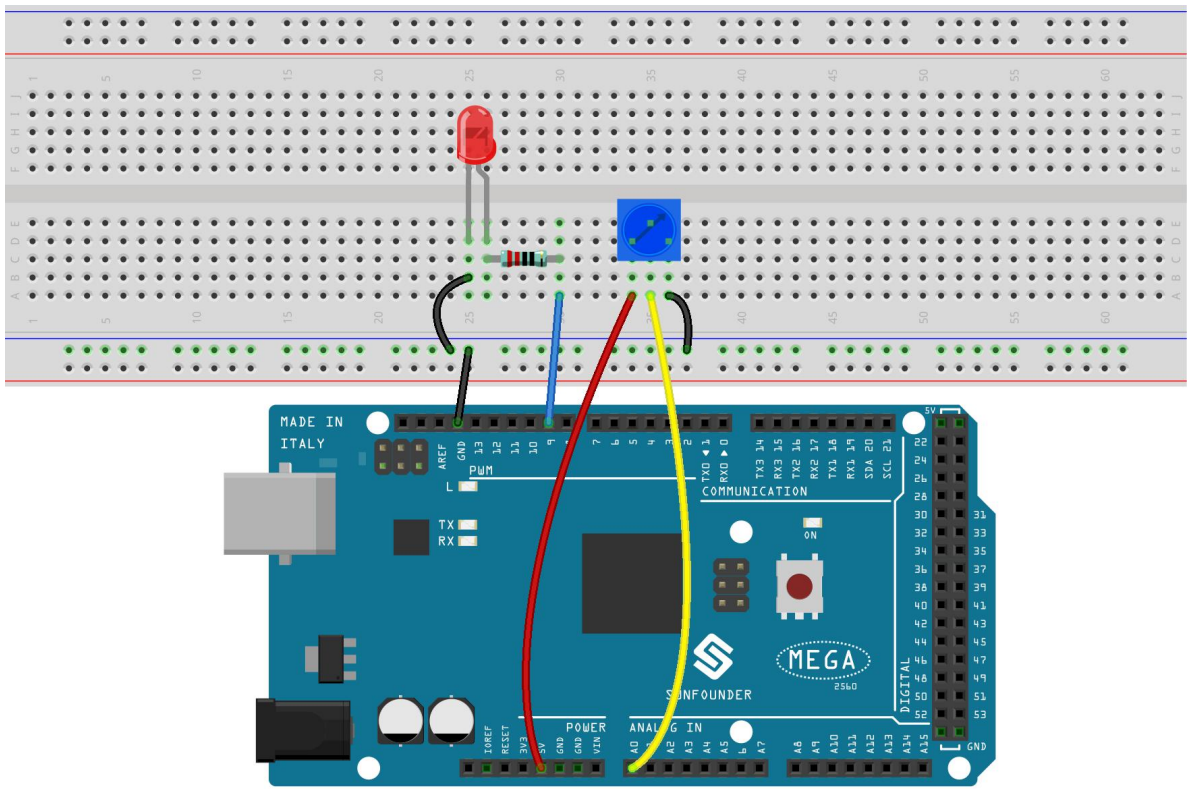
Principle: In this experiment, the potentiometer is used as voltage divider, meaning connecting devices to all of its three pins. Connect the middle pin of the potentiometer to pin A0 and the other two pins to 5V and GND respectively. Therefore, the voltage of the potentiometer is 0-5V. Spin the knob of the potentiometer, and the voltage at pin A0 will change. Then convert that voltage into a digital value (0-1024) with the AD converter in the control board. Through programming, we can use the converted digital value to control the brightness of the LED on the control board.

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit



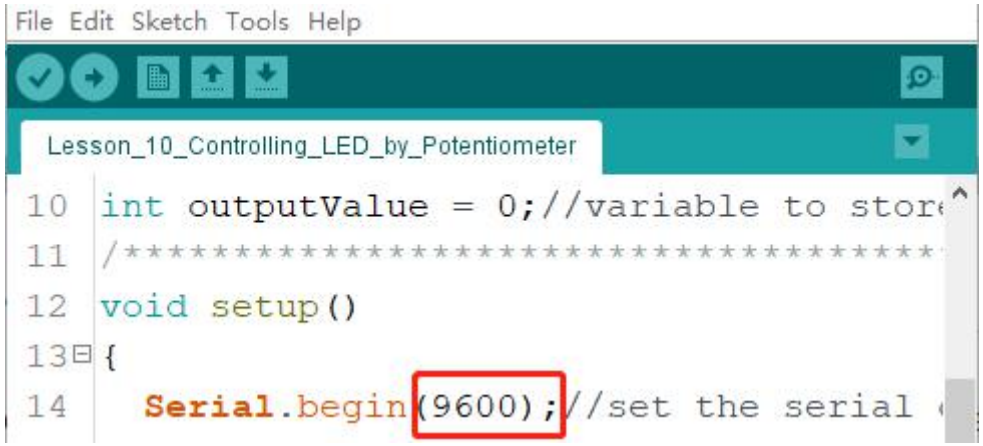
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

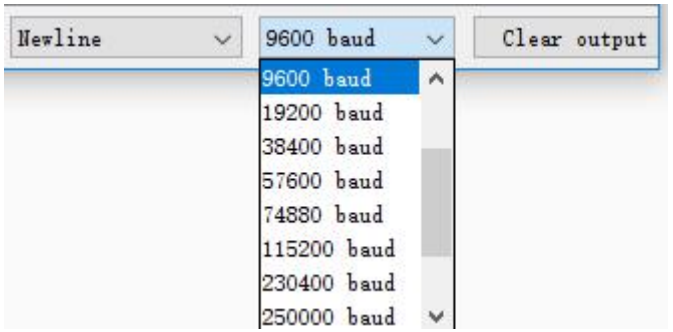
Step5: Open the Serial Monitor.

Find the Serial.begin() code to see what baud rate is set, here is 9600. Then click the top right corner icon to open the Serial Monitor.



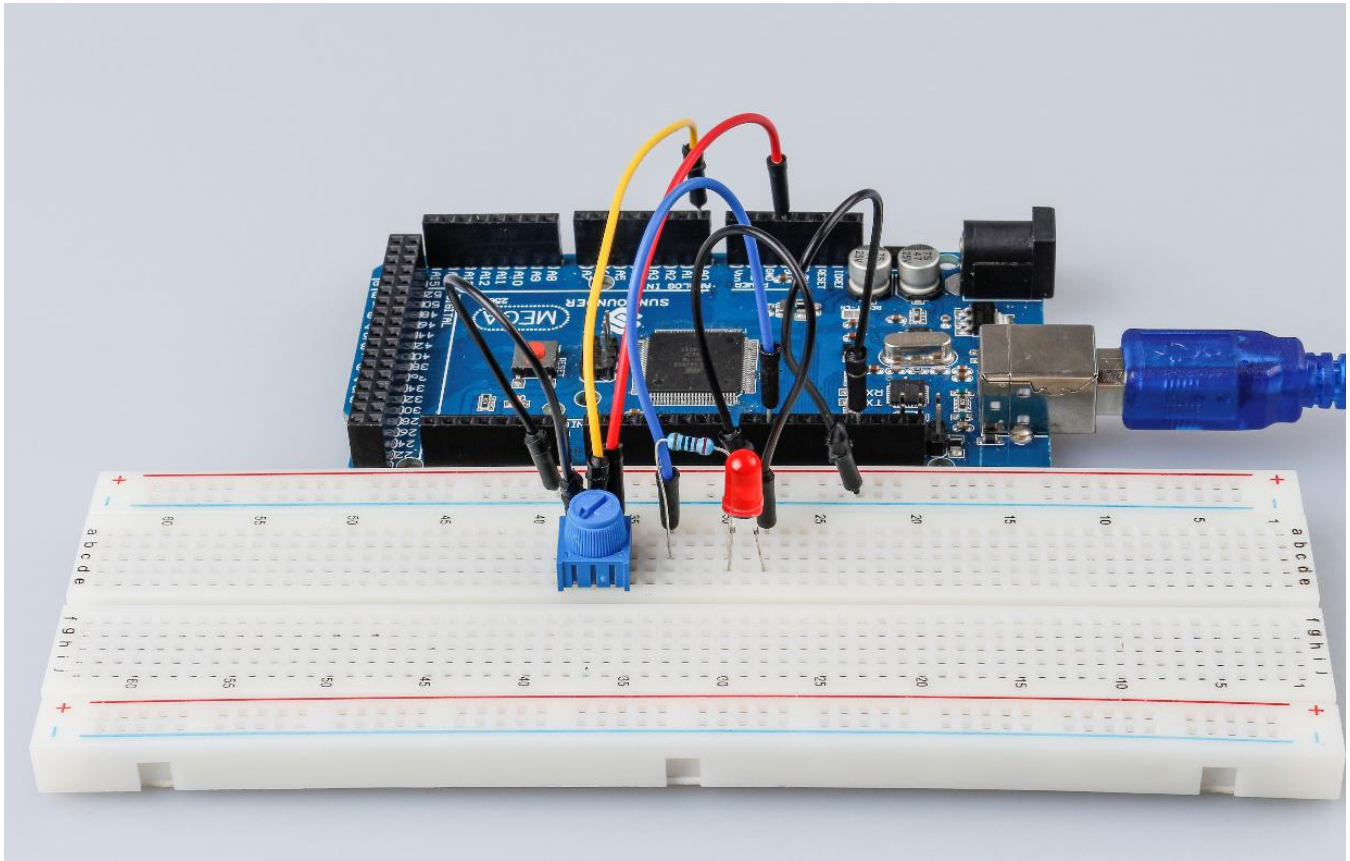
Step6: Set the baud rate to 9600.

The default baud rate for serial monitors is 9600, and if the code is also set to 9600, there is no need to change the baud rate bar.



Spin the shaft of the potentiometer and you should see the luminance of the LED change.

If you want to check the corresponding value changes, open the Serial Monitor and the data in the window will change with your spinning of the potentiometer knob.



Code Analysis

Code Analysis 10-1 Read the value from A0

```
inputValue = analogRead(analogPin);//read the value from the potentiometer
```

This line is to store the values A0 has read in the *inputValue* which has been defined before.

analog Read() reads the value from the specified analog pin. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

Code Analysis 10-2 Print values on Serial Monitor

```
Serial.print("Input: "); //print "Input"
Serial.println(inputValue); //print inputValue
```

Serial.print(): Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

Serial.print(): Commandant takes the same forms as Serial.print(), but it is followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').

Code Analysis 10-3 Map the values

```
outputValue = map(inputValue, 0, 1023, 0, 255);
```

map(value, Fromm, from High, to Low, thigh) re-maps a number from one range to another. That is, a **value** of ***Fromm*** would get mapped to one of ***to Low***, and a value of ***from High*** to one of ***thigh***, values in-between to values in-between, etc.

As the range of *led Pin* (pin 9) is 0-255, we need to map 0-1023 with 0-255.

Display the output value in Serial Monitor in the same way. If you are not so clear about the *map()* functions, you can observe the data in the Serial Monitor and analyze it.

```
Serial.print("Output: "); //print "Output"
Serial.println(outputValue); //print outputValue
```

Code Analysis 10-4 Write the value of the potentiometer to LED

```
analogWrite(ledPin, outputValue); //turn the led on depend on the output value
```

Write the output value to *led Pin* and you will see that the luminance of LED changes with your spinning of the potentiometer knob.

analog Write(): Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *incommode()* to set the pin as output before calling *analog Write()*.

Experiment Summary

This experiment can also be changed to others as you like. For example, use the potentiometer to control the time interval for the LED blinking. It is to use the value read from the potentiometer for delaying, as shown below. Have a try!

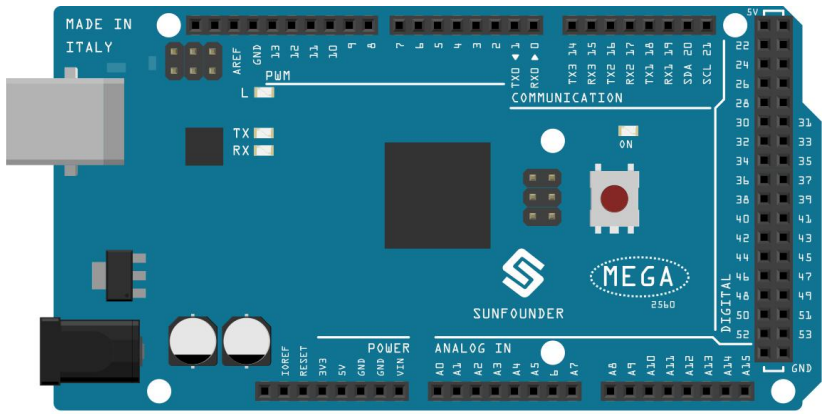
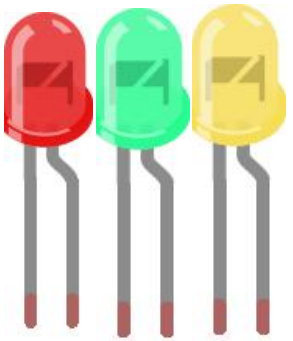
```
inputValue = analogRead(analogPin);//read the value from the sensor
digitalWrite(ledPin,HIGH);
delay(inputValue);
digitalWrite(ledPin,LOW);
delay(inputValue);
```

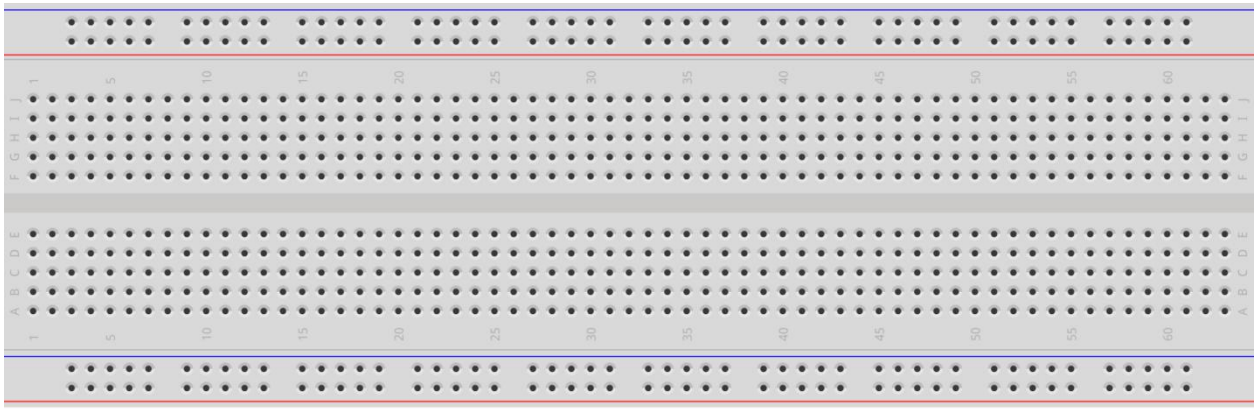
Lesson 11 Photo resistor

Introduction

In this lesson, you will learn to how to measure light intensity using a photo resistor. The resistance of a photo resistor changes with incident light intensity. If the light intensity gets higher, the resistance decreases; if it gets lower, the resistance increases.

Components

1 * Mega 2560 Board 	8 * Resistor (220Ω) 	8 * LED 
	1 * Resistor (10KΩ) 	

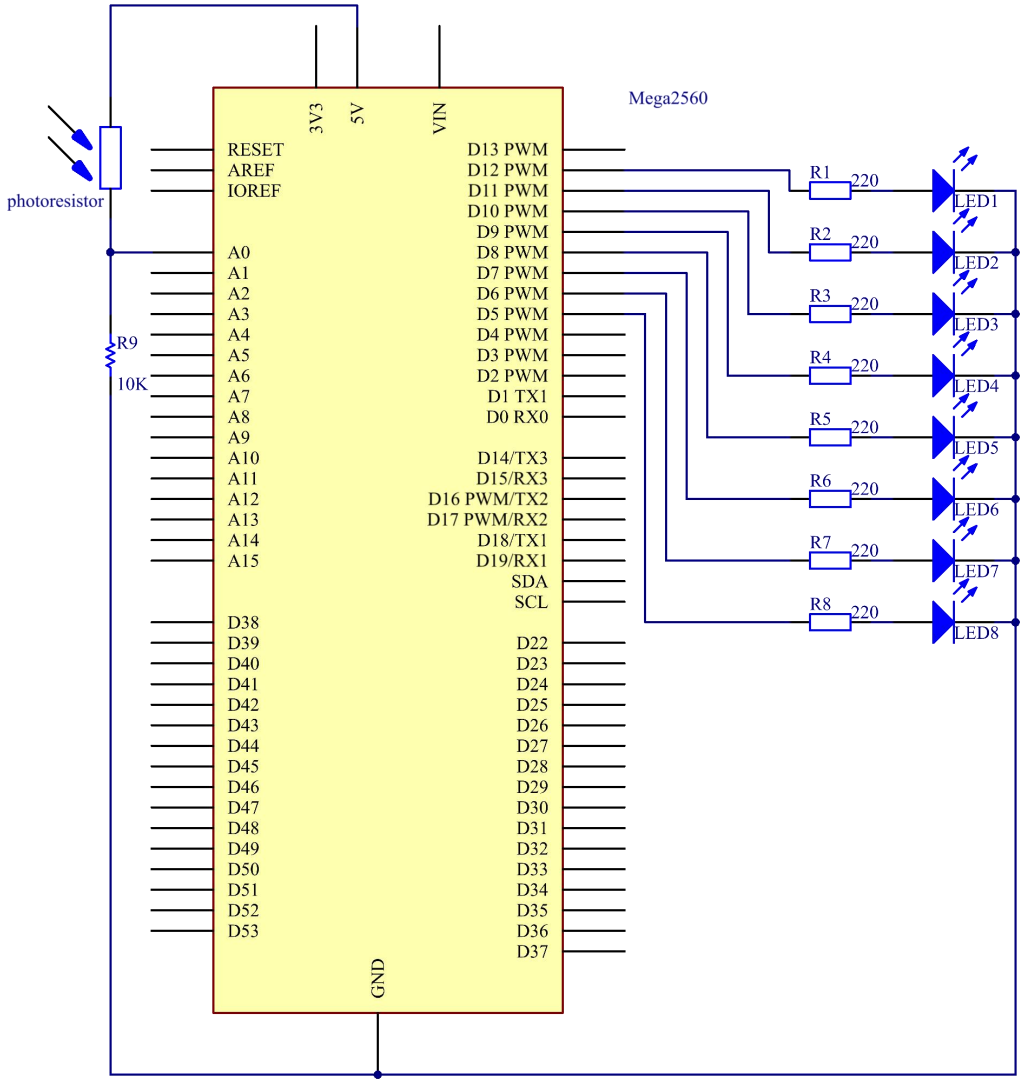
1 * Photo resistor 	1 * Breadboard 
1 * USB cable 	Several jumper wires 

Experimental Principle

A photo resistor or photocell is a light-controlled variable resistor. The resistance of a photo resistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity. A photo resistor can be applied in light-sensitive detector circuits, and light- and darkness-activated switching circuits.

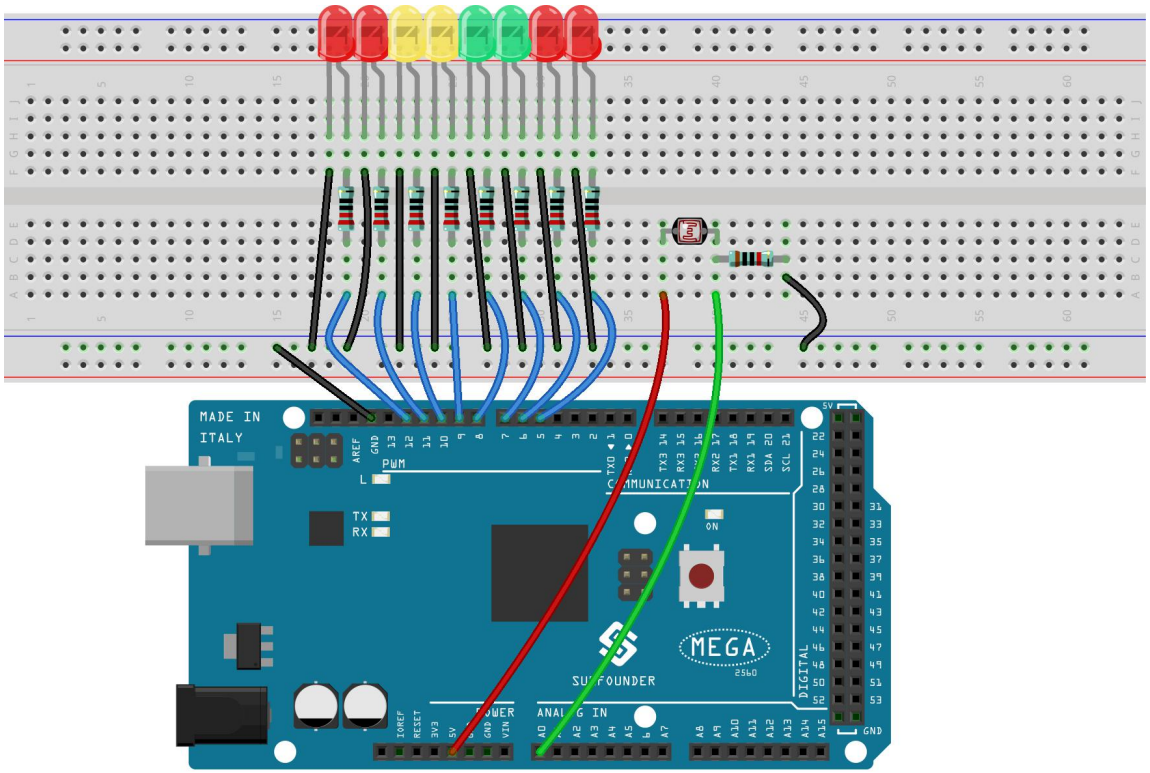
In this experiment, we will use 8 Eds to show the light intensity. The higher the light intensity is, the more Eds will light up. When the light intensity is high enough, all the Eds will be on. When there is no light, all the Eds will go out.

The schematic diagram:


Mega2560
photoresistor
R9
10K
3V3
5V
VIN
GND
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D38
D39
D40
D41
D42
D43
D44
D45
D46
D47
D48
D49
D50
D51
D52
D53
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D3 PWM
D2 PWM
D1 TX1
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
SCL
D22
D23
D24
D25
D26
D27
D28
D29
D30
D31
D32
D33
D34
D35
D36
D37
R1 220
R2 220
R3 220
R4 220
R5 220
R6 220
R7 220
R8 220
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8

Experimental Procedures

Step 1: Build the circuit

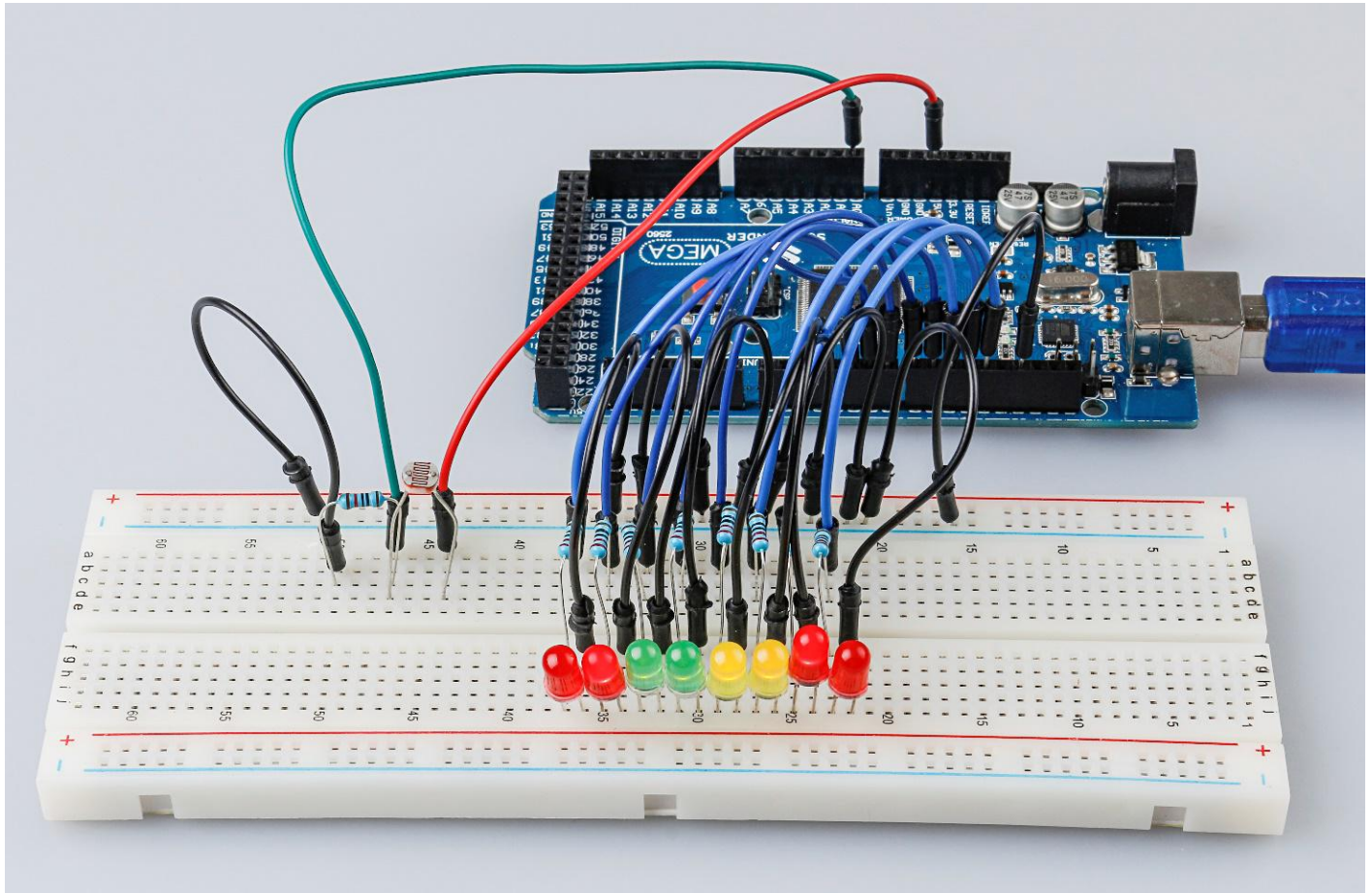


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, shine some light on the photo resistor, and you will see several Eds light up. Shine more light and you will see more Eds light up. When you place it in a dark environment, all the Eds will go out.



Code Analysis

Code Analysis 11-1 Set the variables

```
const int NbrLEDs = 8;//8 leds
const int ledPins[] = {5, 6, 7, 8, 9, 10, 11, 12};
const int photocellPin = A0;  //photoresistor atta
int sensorValue = 0;          // value read from the
int ledLevel = 0;             // sensor value conver
```

The 8 Eds are connected to pin5-pin12, in this code, use a array to store the pins, ledPins[0] is equal to 5, ledPins[1] to 6 and so on.

Code Analysis 11-2 Set 8 pins to OUTPUT

```
for (int led = 0; led < NbrLEDs; led++)
{
  pinMode(ledPins[led], OUTPUT);// make
}
```

Using the for() statement set the 8 pins to OUTPUT. The variable led is added from 0 to 8, and the pinMode() function sets pin5 to pin12 to OUTPUT in turn.

Code Analysis 11-3 Read the analog value of the photoresistor

```
sensorValue = analogRead(photocellPin);
```

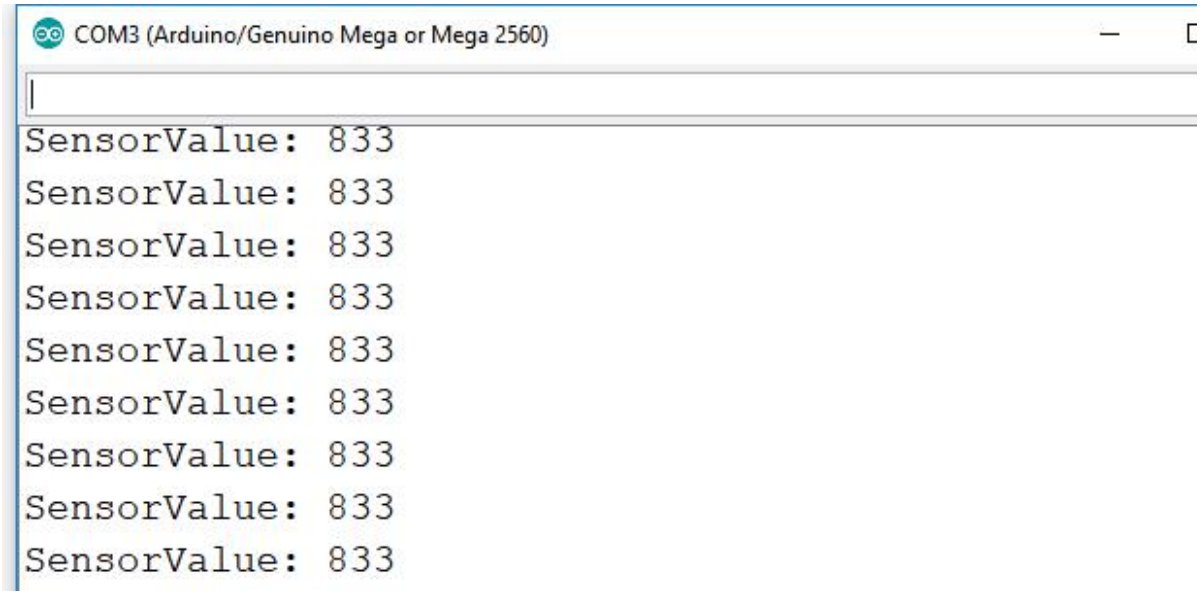
Read the analog value of the **photocellPin(A0**) and store to the variable **sensorValue.**
analogRead(): Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit

analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.

```
Serial.print("SensorValue: ");
Serial.println(sensorValue); /
```

Use the Serial.print()function to print the analog value of the photoresistor. You can see them on the Serial Monitor.

Serial.print(): Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.
Serial.println(): This command takes the same forms as Serial.print(), but it is followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').



Code Analysis 11-4 Map the analog value to 8 LEDs

```
ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs);
Serial.print("ledLevel: ");
Serial.println(ledLevel);
```

The map() command is used to map 0-1023 to 0-NbrLEDs(8), (1023-0)/(8-0)=127.875

0-127.875	128-255.75	256-383.625	384-511.5	512-639.375	640-767.25	768-895.125	896-1023
0	1	2	3	4	5	6	7

If sensorValue is 560, then the ledLevel is 4.

map(value, fromLow, fromHigh, toLow, toHigh) re-maps a number from one range to another. That is, a value of *fromLow* would get mapped to one of *toLow*, and a value of *fromHigh* to one of *toHigh*, values in-between to values in-between, etc.

Code Analysis 11-5 Light up the LEDs

```
for (int led = 0; led < NbrLEDs; led++)
{
  if (led <= ledLevel ) //When led is s
  {
    digitalWrite(ledPins[led], HIGH);
  }
  else
  {
    digitalWrite(ledPins[led], LOW);
  }
}
```

Light up the corresponding LEDs. Such as, when the ledLevel is 4, then light up the ledPins[0] to ledPins[4] and go out the ledPins[5] to ledPins[7].

Lesson 12 Servo

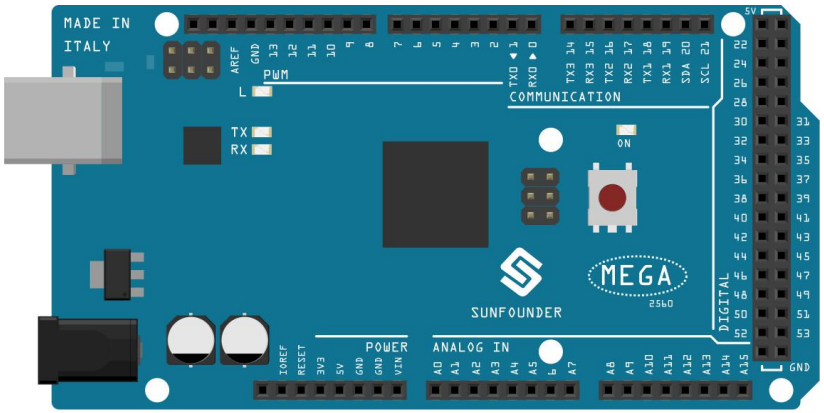
Introduction

Servo is a type of geared motor that can only rotate 180 degrees. It is controlled by sending electrical pulses from your board. These pulses tell the servo what position it should move to.

A servo has three wires: the brown wire is GND, the red one is VCC, and the orange one is signal line.

Components

1 * Mega 2560 Board

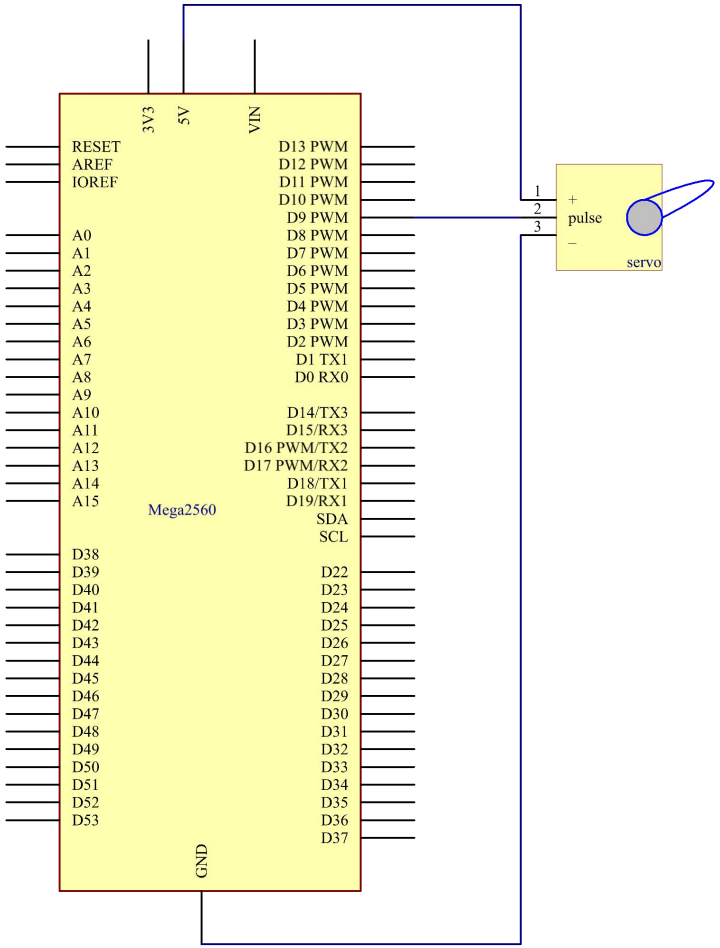


1 * Servo

Several jumper wires(M to F)

1 * USB Cable

Experimental Principle



Servo

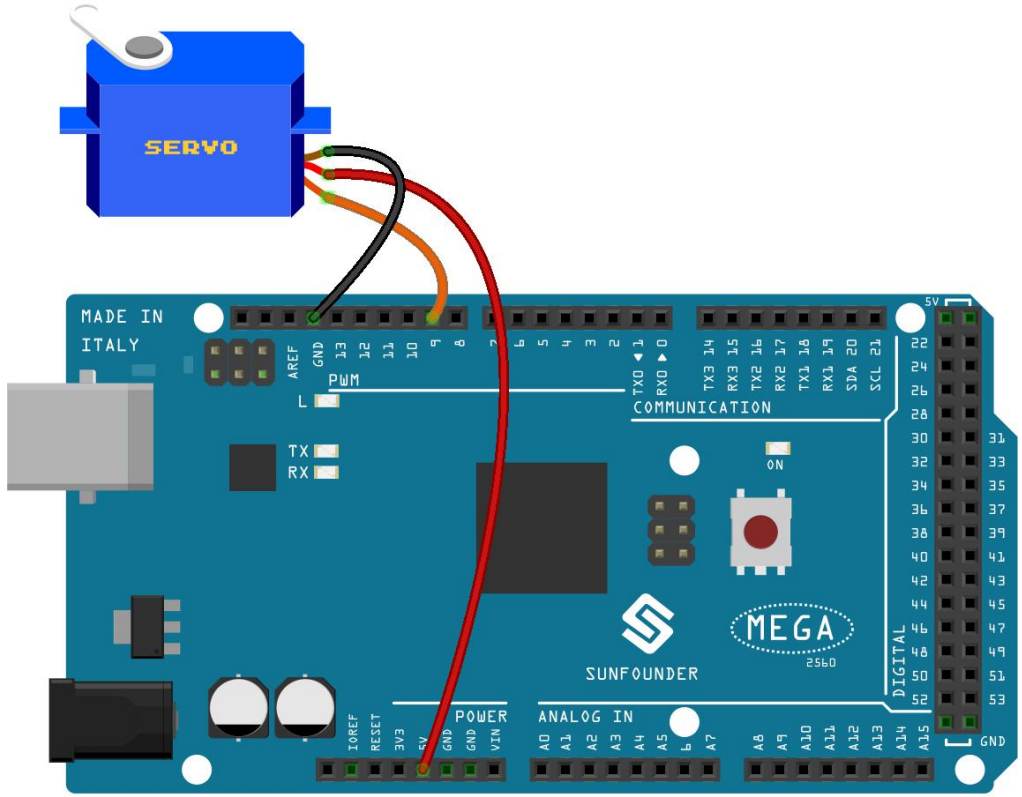
A servo is generally composed of the following parts: case, shaft, gear train, adjustable potentiometer, DC motor, and control circuit board.

It works like this: The Mega 2560 board sends out PWM signals to the servo, and then the control circuit in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear chain and then motivates the shaft after deceleration. The shaft and adjustable potentiometer of the servo are connected together. When the shaft rotates, it drives the pot, so the pot outputs a voltage signal to the circuit board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.

The schematic diagram:

Experimental Procedures

Step 1: Build the circuit (Brown to GND, Red to VCC, Orange to pin 9 of the control board)

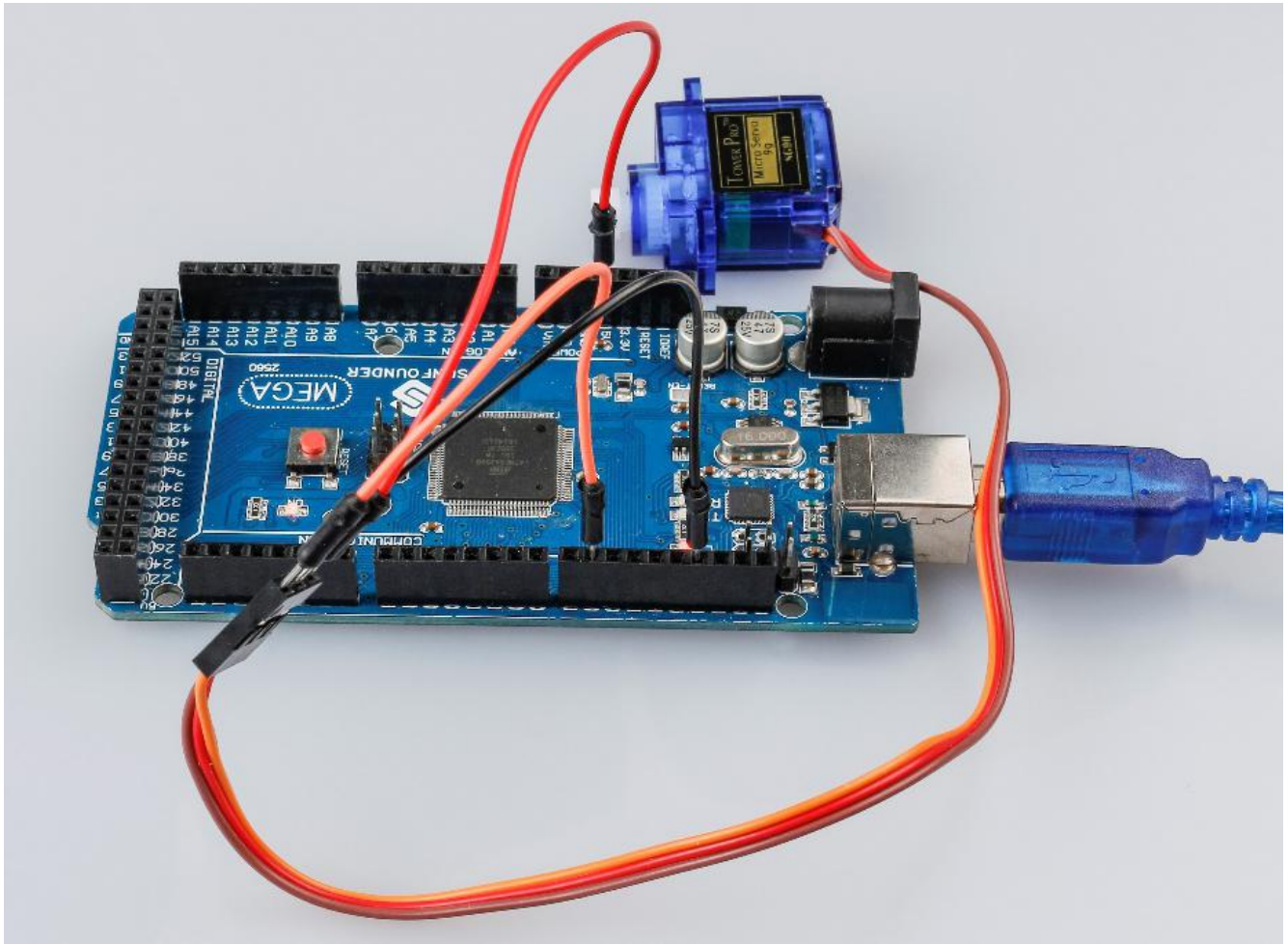


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you can see the rocker arm of the servo rotate and stop at 90 degrees (15 degrees each time). And then it rotates in the opposite direction.



Code Analysis

Code Analysis 12-1 Include a library

```
#include <Servo.h>
/*************************************************/
Servo myservo;//create servo object to control a servo
```

With the *Servo.h* file included, you can call the functions in this file later. Servo is a built-in library in the Arduino IDE. You can find the Servo folder under the installation path *C:\Program Files\Arduino\libraries*.

Code Analysis 12-2 Initialize the servo

```
myservo.attach(9);
myservo.write(0);/
```

myservo.attach(): Attach the Servo variable to a pin. Initialize the servo attach to pin9.

myservo.write(): Writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. Here let the servo stay in the 0 angle firstly.

Code Analysis 12-3 Servo rotate

```
for (int i = 0; i <= 180; i++)
{
  myservo.write(i); //write the i angle to the servo
  delay(15); //delay 15ms
}
for (int i = 180; i >= 0; i--)
{
  myservo.write(i); //write the i angle to the servo
  delay(15); //delay 15ms
}
```

.
.
.

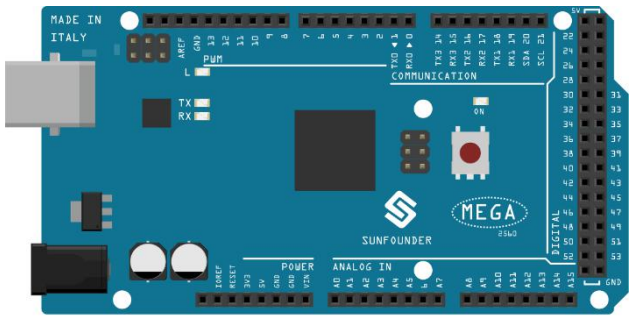
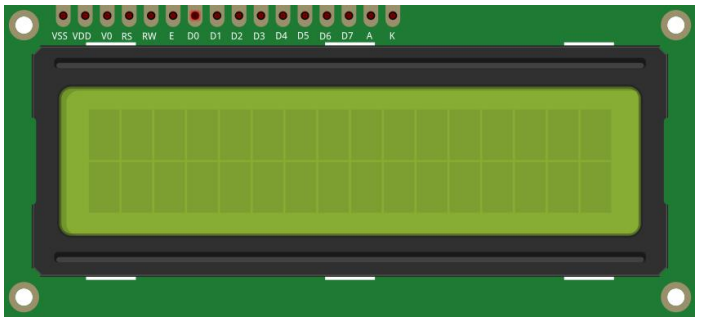
Use 2 for() statement to write 0 - 180 to the servo, so that you can see the servo rotate from 0 to 180 angle,then turn back to 0.

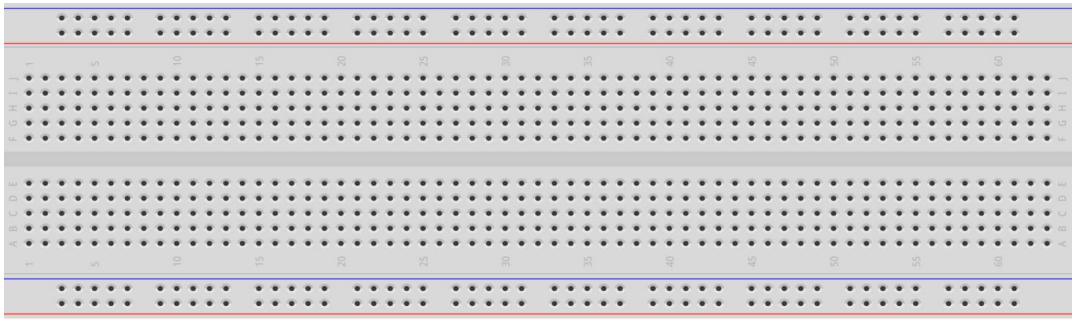
Lesson 13 LCD1602

Introduction

In this lesson, we will learn how to use an LCD1602 to display characters and strings. LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters and so on. It's composed of 5x7 or 5x11 dot matrix positions; each position can display one character. There's a dot pitch between two characters and a space between lines, thus separating characters and lines. The number 1602 means on the display, 2 rows can be showed and 16 characters in each. Now let's check more details!

Components

1 * Mega 2560 Board 	1 * LCD1602 

1 * Potentiometer (10kΩ) 	1 * Breadboard 	
	1 * USB cable 	Several jumper wires 

Experimental Principle

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the Mega 2560 board are almost completely occupied. If you want to connect more sensors, there will be no ports available. Therefore, the four-port connection is used here for better application.

Pins of LCD1602 and their functions

VSS: connected to ground

VDD: connected to a +5V power supply

VO: to adjust the contrast

RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

R/W: A Read/Write pin to select between reading and writing mode

E: An enabling pin that reads the information when High level (1) is received. The instructions are run when the signal changes from High level to Low level.

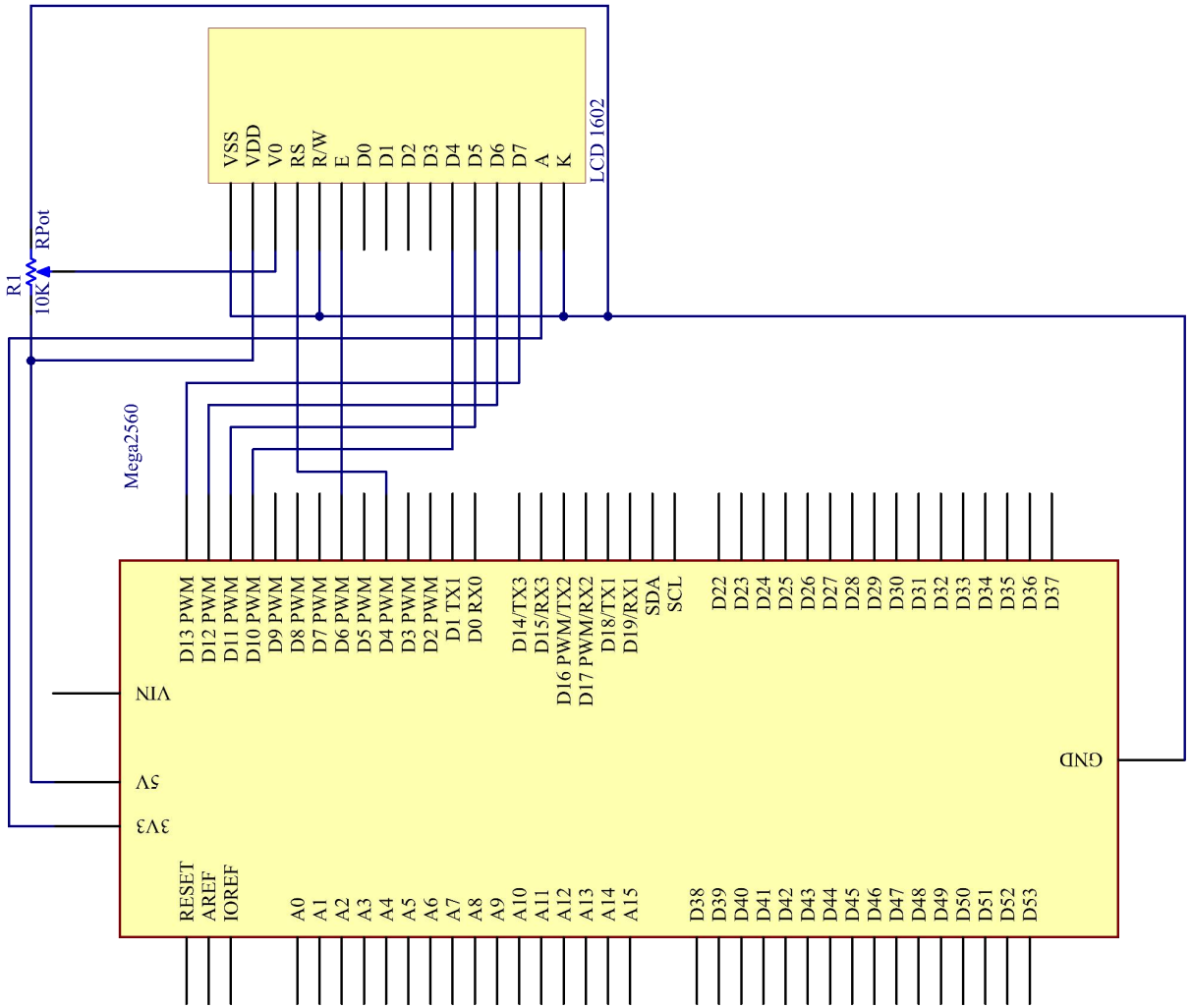
D0-D7: to read and write data

A and K: Pins that control the LCD backlight. Connect K to GND and A to 3.3v. Open the backlight and you will see clear characters in a comparatively dark environment.

Principle:

Connect K to GND and A to 3.3 V, and then the backlight of the LCD1602 will be turned on. Connect VSS to GND and the LCD1602 to the power source. Connect VO to the middle pin of the potentiometer – with it you can adjust the contrast of the screen display. Connect RS to D4 and R/W pin to GND, which means then you can write characters to the LCD1602. Connect E to pin6 and the characters displayed on the LCD1602 are controlled by D4-D7. For programming, it is optimized by calling function libraries.

The schematic diagram:



Experimental Procedures

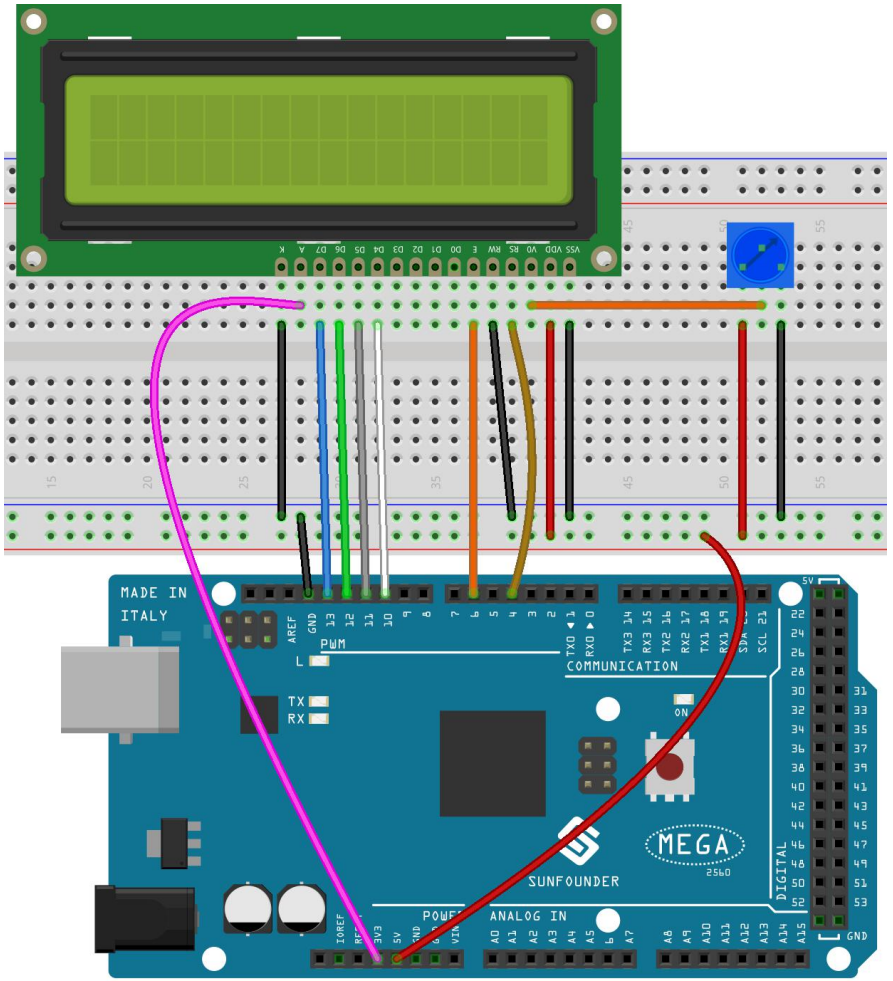
Step 1: Build the circuit (make sure the pins are connected correctly. Otherwise, characters will not be displayed properly):

Step 2: Open the code file.

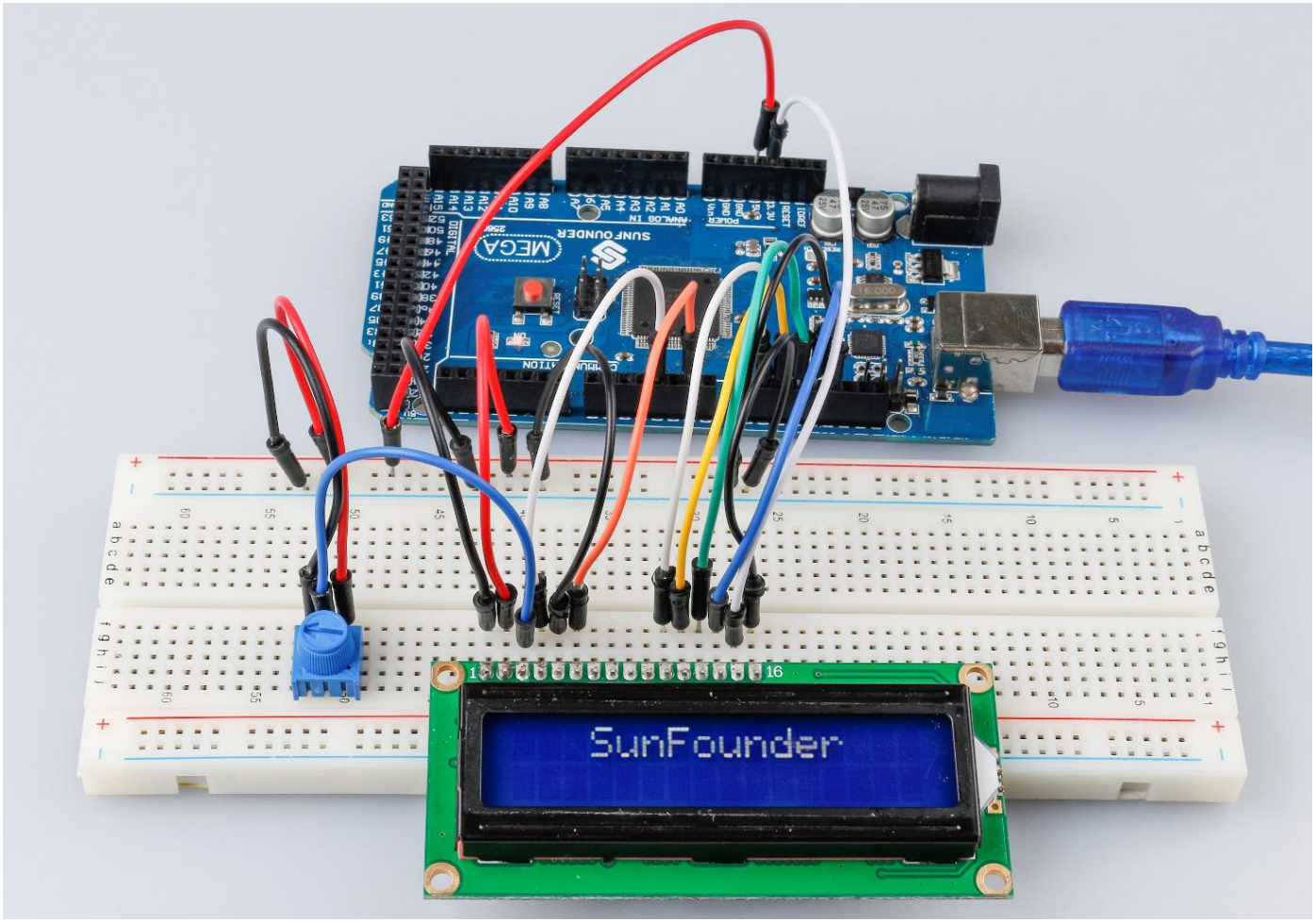
Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Note: you may need to adjust the potentiometer on the LCD1602 until it can display clearly.



You should now see the characters "**SunFounder**" and "**hello, world**" rolling on the LCD.



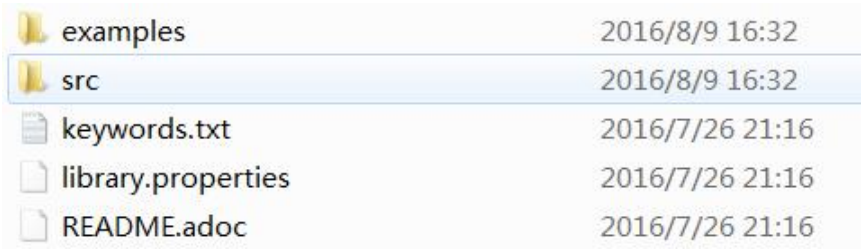
Code Analysis

Code Analysis 13-1 Include a library

```
#include <LiquidCrystal.h>// include the library code
```

With the *LiquidCrystal.h* file included, you can call the functions in this file later.

LiquidCrystal is a built-in library in the Arduino IDE. You can find the LiquidCrystal folder under the installation path *C:\Program Files\Arduino\libraries*.



There is an example in the *examples* folder. The src folder contains the major part of the library: *LiquidCrystal.cpp* (execution file, with function implementation, variable definition, etc.) and LiquidCrystal.h (header file, including function statement, Macro definition, struct definition, etc.). If you want to explore how a function is implemented, you can look up in the file *LiquidCrystal.cpp*.

Code Analysis 13-2 Displayed characters

```
char array1[]=" SunFounder          "; //the string to print on the LCD
char array2[]="hello, world!        "; //the string to print on the LCD
```

These are two character type arrays: *arry1[]* and *array2[]*. The contents in the quotation marks ”xxx” are their elements, including 26 characters in total (spaces counted). *array1[0]* stands for the first element in the array, which is a space, and *array1[2]* means the second element *S* and so on. So *array1[25]* is the last element (here it's also a space).

Code Analysis 13-3 Define the pins of LCD1602

```
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);
```

Define a variable *lcd* of LiquidCrystal type. Here use *lcd* to represent *LiquidCrystal* in the following code.

The basic format of the *LiquidCrysral()* function is: LiquidCrystal (rs, enable, d4, d5, d6, d7). You can check the *LiquidCrystal.cpp* file for details.

So this line defines that pin RS is connected to pin 4, the enable pin to pin 6, and d4-d7 to pin10-13 respectively.

Code Analysis 13-4 Initialize the LCD

```
lcd.begin(16, 2);  // set up the LCD's number of columns and rows:
```

begin(col,row) is to set the display of LCD. Here set as 16 x 2.

Code Analysis 13-5 Set the cursor position of LCD

```
lcd.setCursor(15,0);  // set the cursor to column 15, line 0
```

setCursor(col,row) sets the position of the cursor which is where the characters start to show. Here set it as 15col, 0 row.

Code Analysis 13-6 LCD displays the elements inside array1[] and array2[]

```
for ( int positionCounter1 = 0; positionCounter1 < 26; positionCounter1++)
    {
      lcd.scrollDisplayLeft();  //Scrolls the contents of the display one space to the left.
      lcd.print(array1[positionCounter1]);  // Print a message to the LCD.
      delay(tim);  //wait for 250 microseconds
    }
```

When *positionCounter1*=0, which accords with *positionCounter1*<26, *positionCounter1* adds 1. Move one bit to the left through *lcd.scrollDisplayLeft()*. Make the LCD display array1[0] by *lcd.print*(array1[positionCounter1]) and delay for *tim* ms (250 ms). After 26 loops, all the elements in *array1[]* have been displayed.

```
lcd.clear();  //Clears the LCD screen.
```

Clear the screen with *lcd.clear()* so it won't influence the display next time.

```
lcd.setCursor(15,1);  // set the cursor to column 15, line 1
```

Set the cursor at Col. 15 Line 1, where the characters will start to show.

```
for (int positionCounter2 = 0; positionCounter2 < 26; positionCounter2++)
    {
      lcd.scrollDisplayLeft();  //Scrolls the contents of the display one space to the left.
      lcd.print(array2[positionCounter2]);  // Print a message to the LCD.
      delay(tim);  //wait for 250 microseconds
    }
```

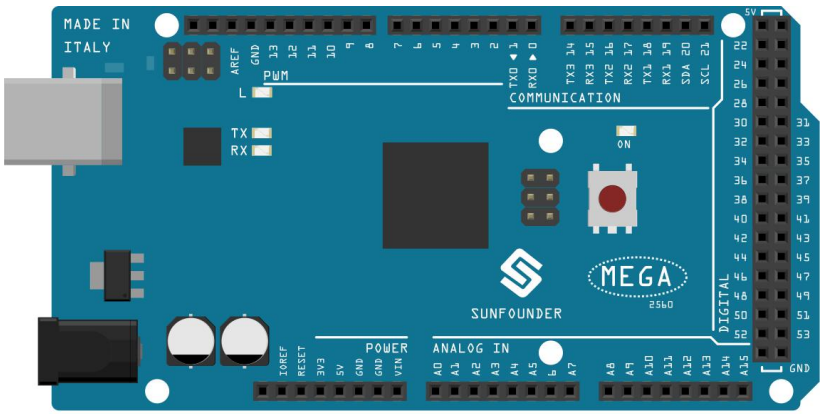
Similarly, the code is to display the elements in *array2[]* on the LCD. Therefore, you will see “SunFounder” scroll in the top line of the LCD, move left until it disappears. And then in the bottom line, “hello, world ! " appears, scrolls to the left until it disappears.

Lesson 14 Thermistor

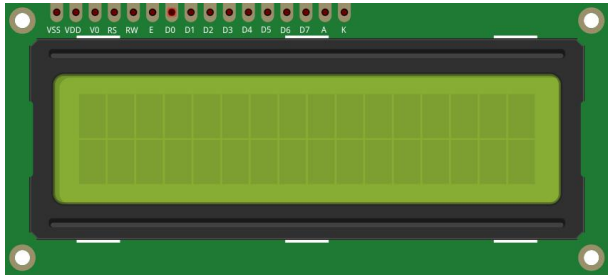
Introduction

We've learnt many devices so far. To make more things, you need to have a good command of more knowledge. Today we're going to meet a thermistor. It is similar to photoresistor in being able to change their resistance based on the outer change. Different from photoresistor, resistance of thermistor varies significantly with temperature in the outer environment.

Components

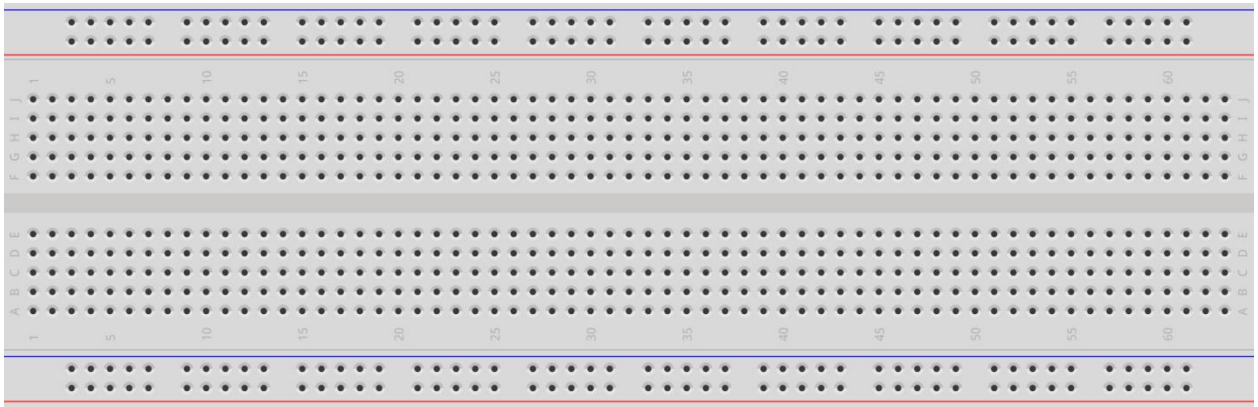
1 * Mega 2560 Board	1 * Thermistor	Potentiometer (10KΩ)
		

1 * LCD1602



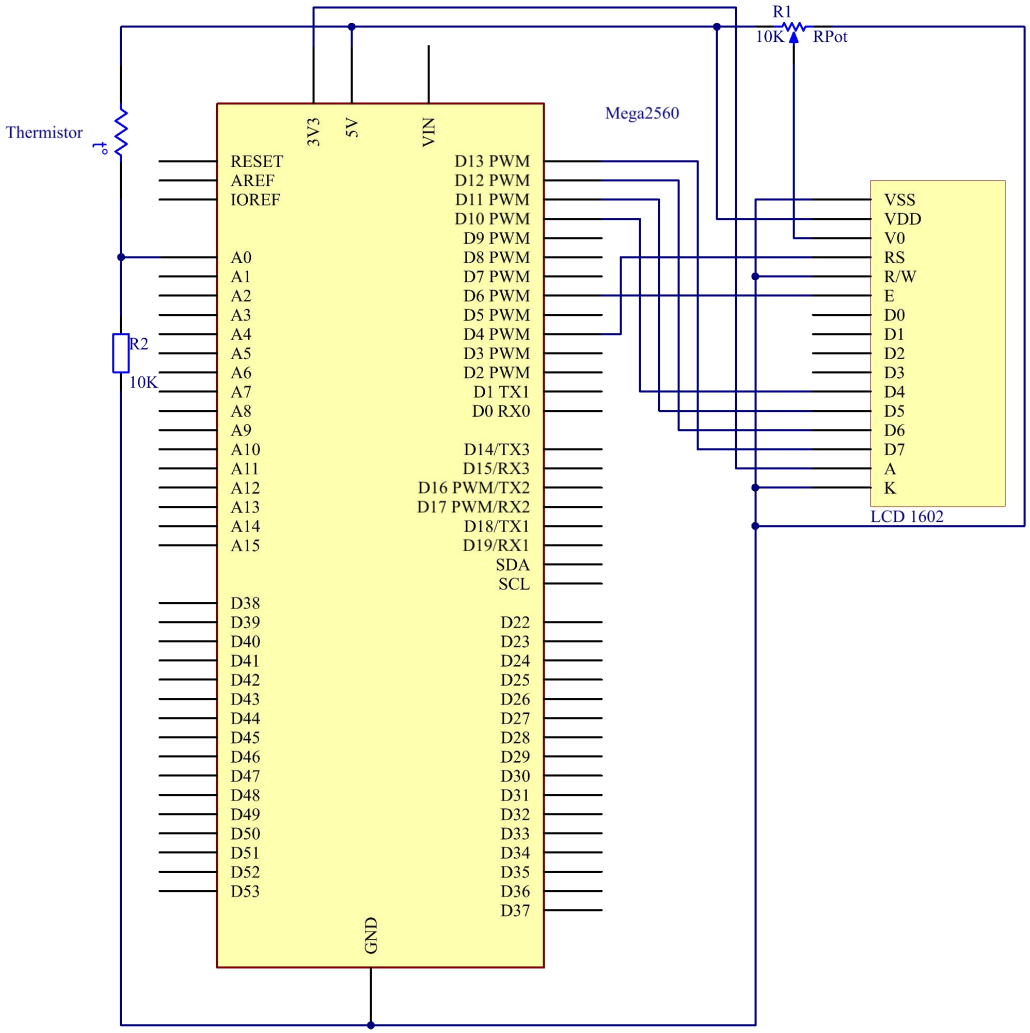
1 * Resister (10KΩ)

1 * Breadboard



1 * USB cable

Several jumper wires



Experimental Principle

Thermistor is a sensitive element, it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also NTC and PTC. Its resistance varies significantly with temperature. The resistance of PTC thermistor increases with higher temperature when that of NTC, decreases. In this experiment we use an NTC one.

The schematic diagram:

The principle is that the resistance of the NTC thermistor changes with the temperature difference in the outer environment. It detects the real-time temperature of the environment. When the temperature gets higher, the resistance of the thermistor decreases and the voltage of pin A0 increases accordingly. The voltage data then is converted to digital quantities by the A/D adapter. The temperature in Celsius and Fahrenheit then is output via programming and then displayed on LCD1602.

In this experiment a thermistor and a 10k pull-up resistor are used. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature change:

$$R_T = R_N \exp^{B(1/TK - 1/TN)}$$

R_T: resistance of the NTC thermistor when the temperature is T_K.

R_N: resistance of the NTC thermistor under the rated temperature which is T_N.

T_K is a Kelvin temperature and the unit is K.

T_N is a rated Kelvin temperature; the unit is K, also.

And, beta, here is the material constant of NTC thermistor, also called heat sensitivity index.

exp is short for exponential, an exponential with the base number e, which is a natural number and equals 2.7 approximately.

Note that this relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

Since T_K =T+273, T is Celsius temperature, the relation between resistance and temperature change can be

transformed into this:

$$R = R_o \exp^{B[1/(T+273) - 1/(To+273)]}$$

B, short for beta, is a constant. Here it is 4090. R_o is 10k ohms and T_o is 25 degrees Celsius. The data can be found in the datasheet of thermistor. Again, the above relation can be transformed into one to evaluate temperature:

$$T = B/[\ \ln(R/10) + (B/298)\ ] - 273$$ (So ln here means natural logarithm, a logarithm to the base e)

If we use a resistor with fixed resistance as 10k ohms, we can calculate the voltage of the analog input pin A0 with this formula:

$$V = 10k \times 5/(R+10K)$$

So, this relation can be formed:

$$R = (5 \times 10k\ /V) - 10k$$

The voltage of A0 is transformed via A/D adaptor into a digital number a.

$$a = V \times (1024/5)$$

$$V = a/205$$

Then replace V in the relation $R = (5 \times 10k\ /V) - 10k$ with the expression, and we can get this: R=1025 x 10k/a-10k.

Finally replace R in the formula here T= B/[ln(R/ 10) + (B/ 298)] – 273, which is formed just now. Then we at last get the relation for temperature as this:

$$T = B/[\ \ln\{[\ 1025 \times 10/a) - 10]/10\} + (B/298)] - 273$$

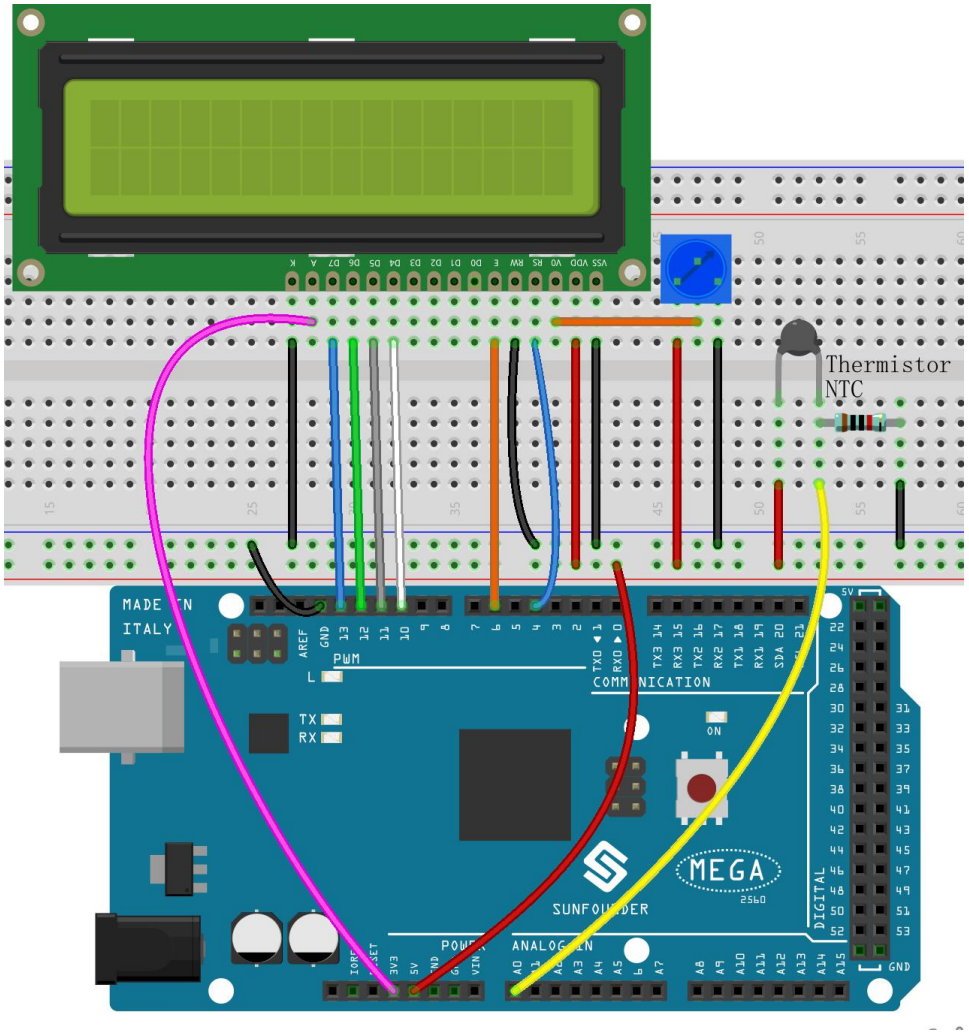
Experimental Procedures

Step 1: Build the circuit

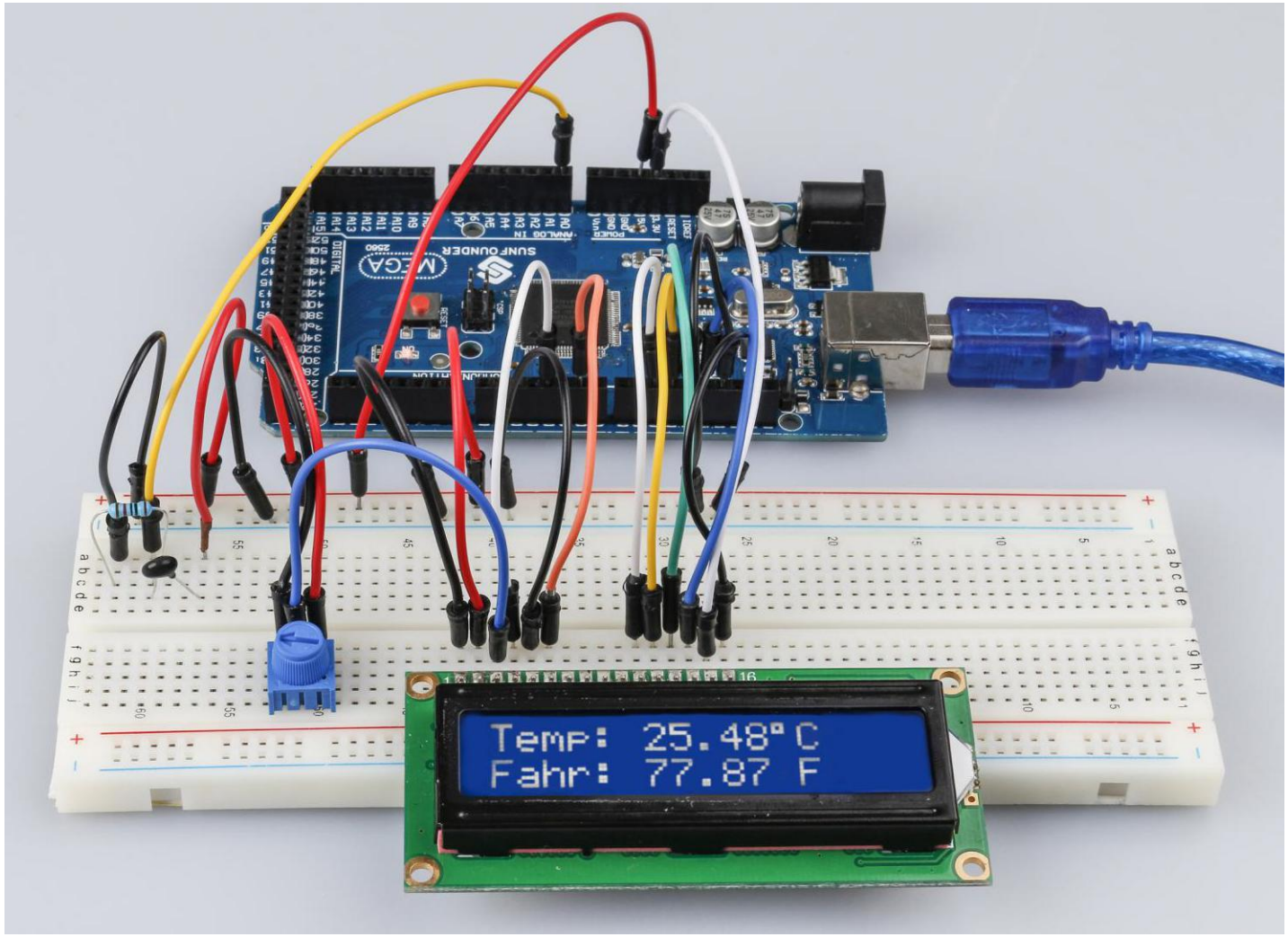
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, you can see the current temperature displayed both in Celsius and Fahrenheit degrees on the LCD1602.



Code Analysis

Code Analysis 14-1 Set the variables

```
#define analogPin  A0 //the thermistor attach to
#define beta 3950 //the beta of the thermistor
#define resistance 10 //the value of the pull-up resistor
```

Define the beta coefficient as 4090, which is described in the datasheet of thermistor.

Code Analysis 14-2 Get the temperature

```
long a = analogRead(analogPin);
```
//Read the resistance value of the thermistor to a via the signal from the analog pin. Here use a long type to make the value of a to be a long integer.

```
float tempC = beta / (log((1025.0 * 10 / a - 10) / 10) + beta / 298.0) - 273.0;
```
//The formula here is to calculate the temperature in Celsius, which we deduced previously.

```
float tempF = 1.8 * tempC + 32.0;
```
//define the temperature in Fahrenheit. As we know Fahrenheit equals to 1.8 * Celsius + 32.

Code Analysis 14-3 Display the temperature on LCD1602

```
  lcd.setCursor(0, 0); // set the cursor to column 0, line 0
  lcd.print("Temp: ");// Print a message of "Temp: "to the LCD.
  lcd.print(tempC);
  lcd.print(char(223));//print the unit" ° "
  lcd.print("C");
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1); // set the cursor to column 0, line 1
  lcd.print("Fahr: ");
  lcd.print(tempF);// Print a Fahrenheit temperature to the LCD.
  lcd.print(" F"); // Print the unit of the Fahrenheit temperature to the LCD.
  delay(200); //wait for 100 milliseconds
```

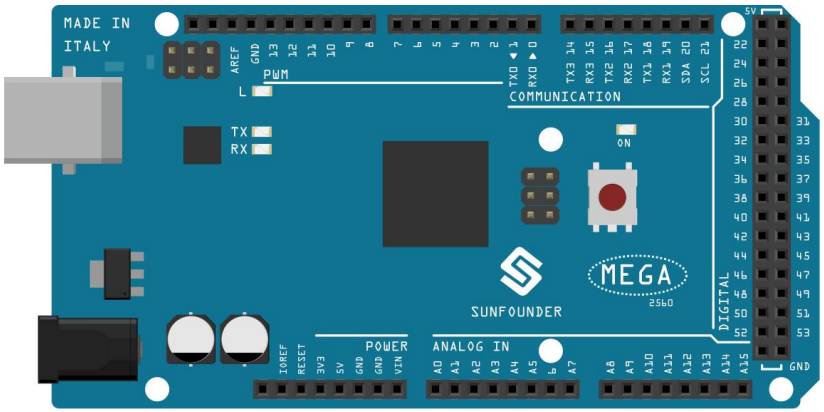
Lesson 15 Ultrasonic

Introduction

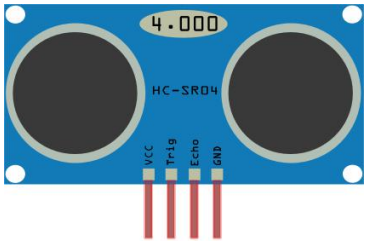
When you are reversing, you will see the distance between the car and the surrounding obstacles to avoid collision. The device for detecting the distance is an ultrasonic sensor. In this experiment, you will learn how the ultrasonic wave detects the distance.

Components

1 * Mega 2560 Board

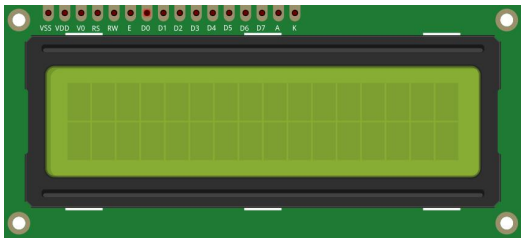


1 * Ultrasonic sensor

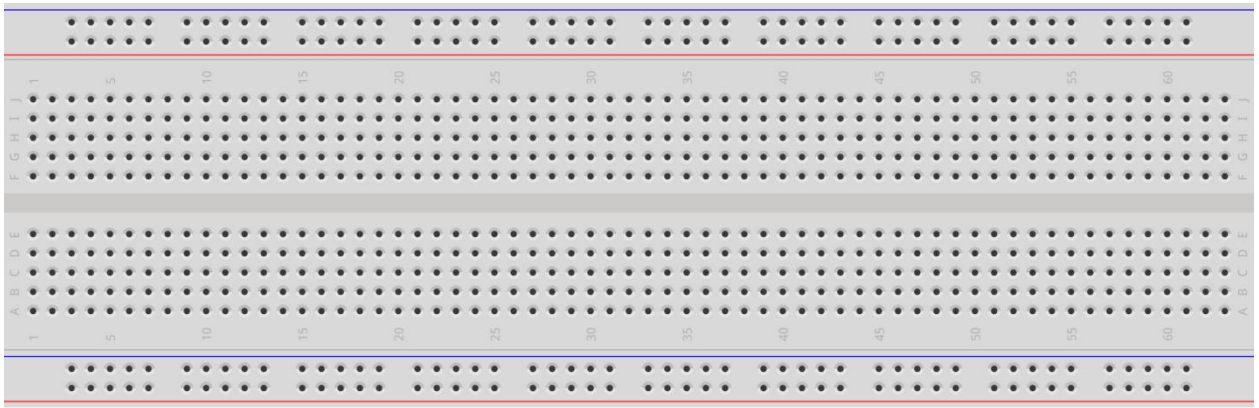


1 * Potentiometer

1 * LCD1602



1 * Breadboard



1 * USB cable

Several jumper wires

Experimental Principle

The ultrasonic distance sensor is really useful and widely applicable in our daily life. It has two probes. One is to send ultrasonic waves and the other is to receive the waves and transform the time of sending and receiving into a distance, thus detecting the distance between the device and an obstacle. In practice it is really convenient and functional.

The ultrasonic ranging module HC-SR04 provides 2cm-700cm non-contact measurement function, and the ranging accuracy can reach 3mm. Stable signal can be ensured within 5m, and signal gradually fades beyond 5m till disappearing at 7m position.

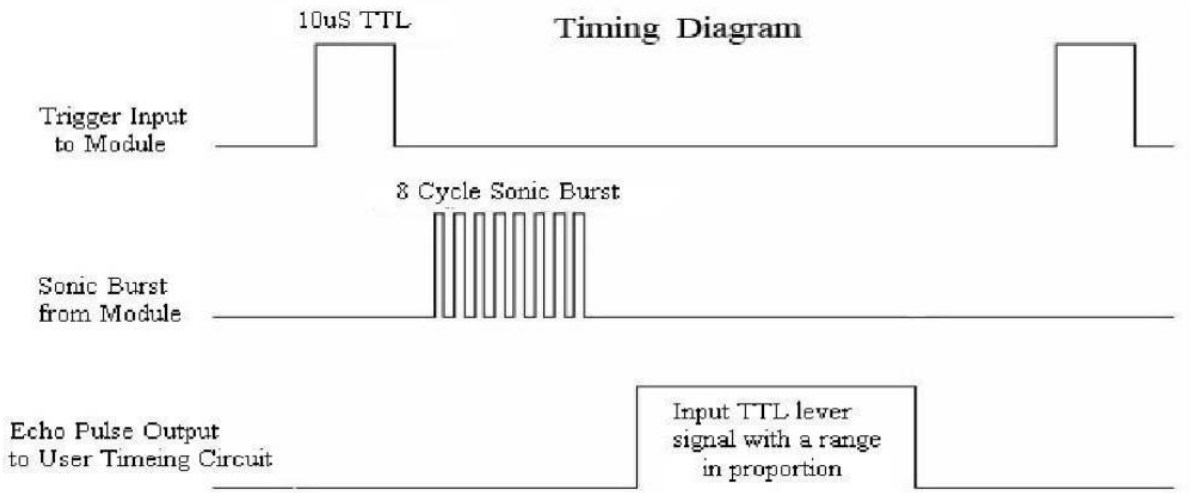
The module includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

1) Using IO trigger for at least 10us high level signal;
2) The module automatically sends eight 40 kHz square waves and detect whether there is a pulse signal sent back.
3) If there's a signal sent back, output a high level through pin ECHO; the time duration is the time from sending the ultrasonic to the returning.

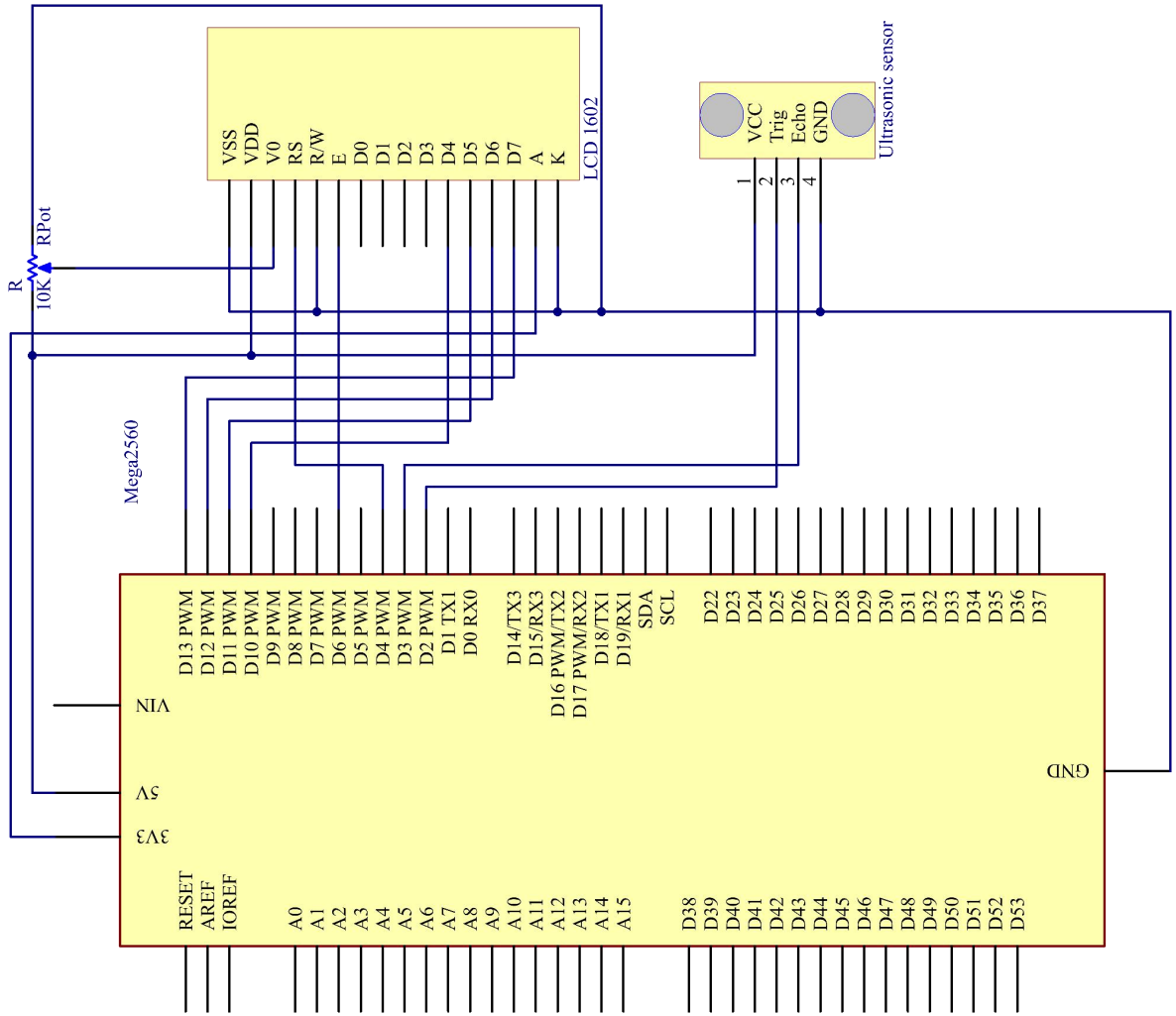
Thus, test distance = (high level time × velocity of sound (340M/S) / 2.

The timing diagram is as shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8-cycle burst of ultrasound at 40 kHz and raise its echo. The echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Thus,

uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; You're recommended to use over 60ms measurement cycle, in order to prevent conflicts between trigger signal and echo signal.

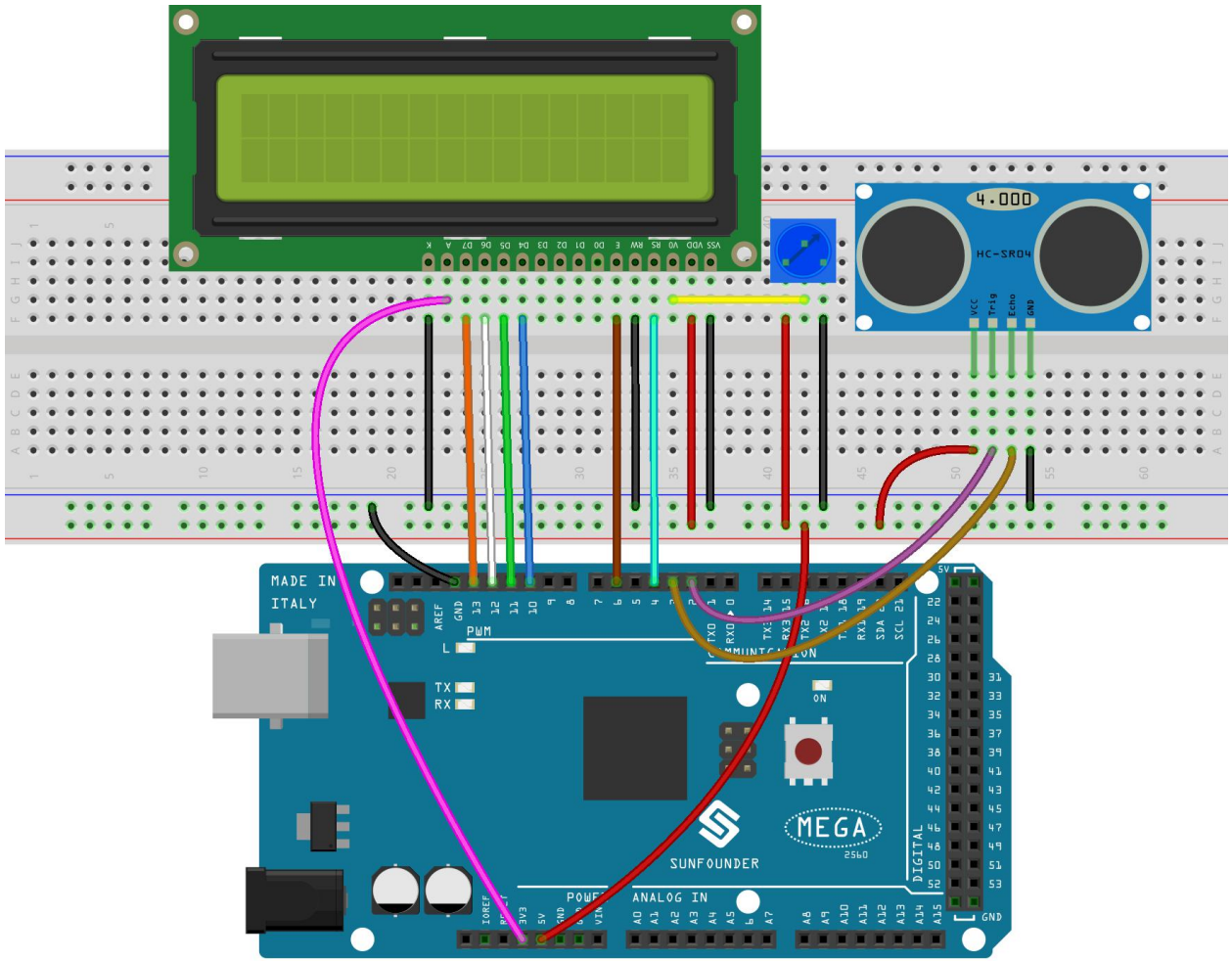


The schematic diagram:



Experimental Procedures

Step 1: Build the circuit

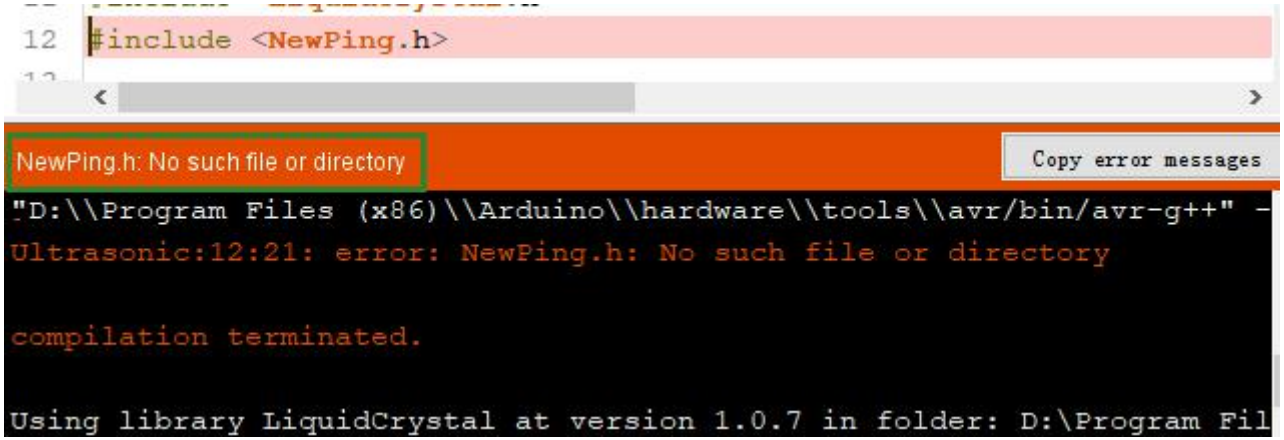


Step 2: Open the code file.

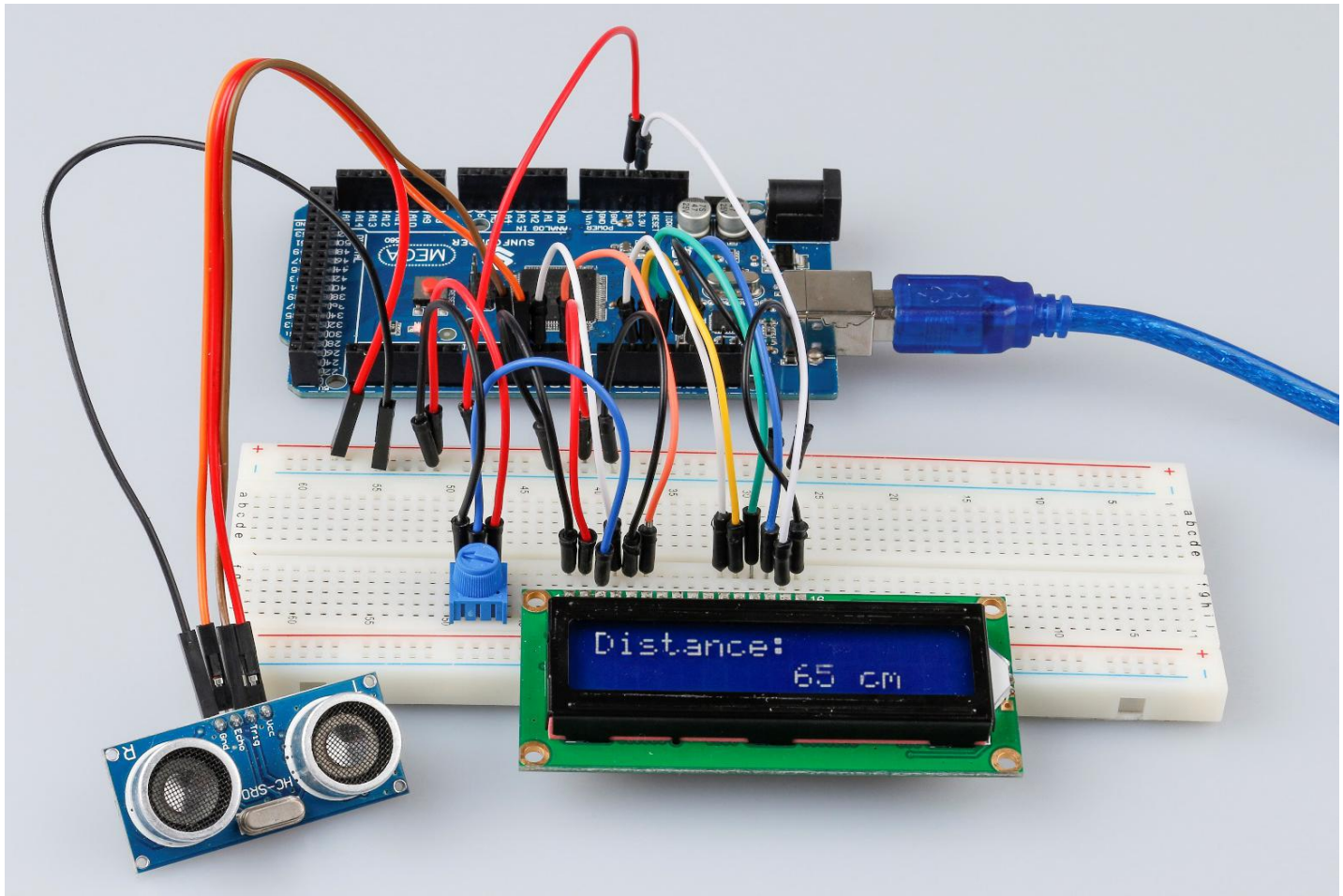
Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Note：If you receive the following error, it is because you didn't add a library named NewPing, please refer to Lesson 2 Add libraries to add it.



Now, if you use a piece of paper to approach or keep it far away from the sensor. You will see the value displayed on the LCD changes, which indicates the distance between the paper and the ultrasonic sensor.


Distance:
65 cm

Code Analysis

Code Analysis 15-1 Initialize the ultrasonic sensor and LCD1602

```
#include <LiquidCrystal.h>  // use #include  to define the header file.
#include <NewPing.h>    // use #include  to define the header file.

LiquidCrystal lcd(4, 6, 10, 11, 12, 13);//lcd(RS,E,D4,D5,D6,D7)

#define TRIGGER_PIN  2  //  trig pin on the ultrasonic sensor attach to pin2 .
#define ECHO_PIN     3  //  echo pin on the ultrasonic sensor attach to pin3.
#define MAX_DISTANCE 400 // Maximum distance we want to ping for (in centimeters). Maximum
sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum
distance.
```

Create a NewPing variable sonar. The basic format of NewPing is: NewPing (uint8_t trigger_pin, uint8_t echo_pin, int max_cm_distance). Here uint8 comes up again. As we mentioned previously in lesson 8 of the RFID series, uint means an unsigned integer and 8 means 8 bits. So a value in the uint8 format here means an unsigned-char type value.

Code Analysis 15-2 Convert the time to distance

```
unsigned int uS = sonar.ping(); // Send ping, get ping time in microseconds (uS).
```

ping() here is to calculate the time from pulse sending to receiving. Define a vairal uS and assign the time to it. Its unit should be microsecond (us).

```
 int distance = uS / US_ROUNDTRIP_CM;
```

uS / US_ROUNDTRIP_CM is a formula to convert the time between ping sending and receiving into a distance. The unit is cm.

Code Analysis 15-3 Display the distance on the LCE1602

```
  lcd.setCursor(0, 0);//Place the cursor at Line 1, Column 1. From here the characters are
to be displayed
  lcd.print("Distance:");////Print Distance: on the LCD
  lcd.setCursor(0, 1);//Set the cursor at Line 1, Column 0
  lcd.print("           ");//Here is to leave some spaces after the characters so as to clear
the previous characters that may still remain.
  lcd.setCursor(9, 1);//Set the cursor at Line 1, Column 9.
  lcd.print(distance);// print on the LCD the value of the distance converted from the time
between ping sending and receiving.
  lcd.setCursor(12, 1);//Set the cursor at Line 1, Column 12.
  lcd.print("cm");//print the unit "cm"
```

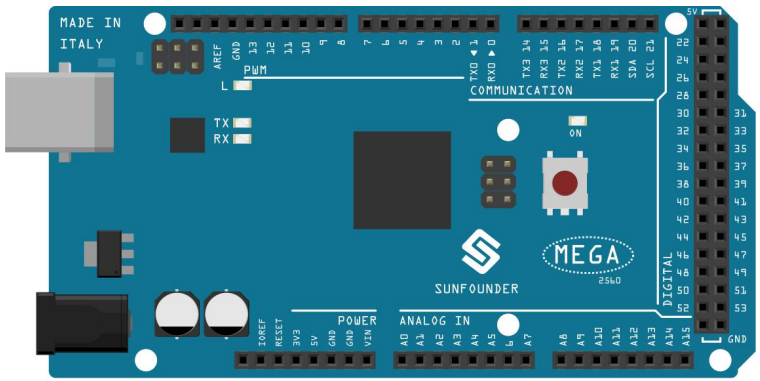
Lesson 16 Infrared-Receiver

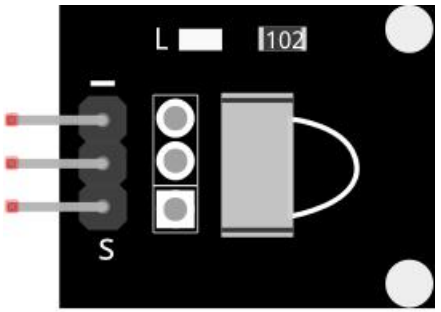
Introduction

An infrared-receiver is a component that receives infrared signals and can independently receive infrared ray and output signals compatible with TTL level. It's similar with a normal plastic-packaged transistor in size and it is suitable for all kinds of infrared remote control and infrared transmission.

Components

1 * Mega 2560 Board	1 * Infrared-Receiver module





1 * Remote controller

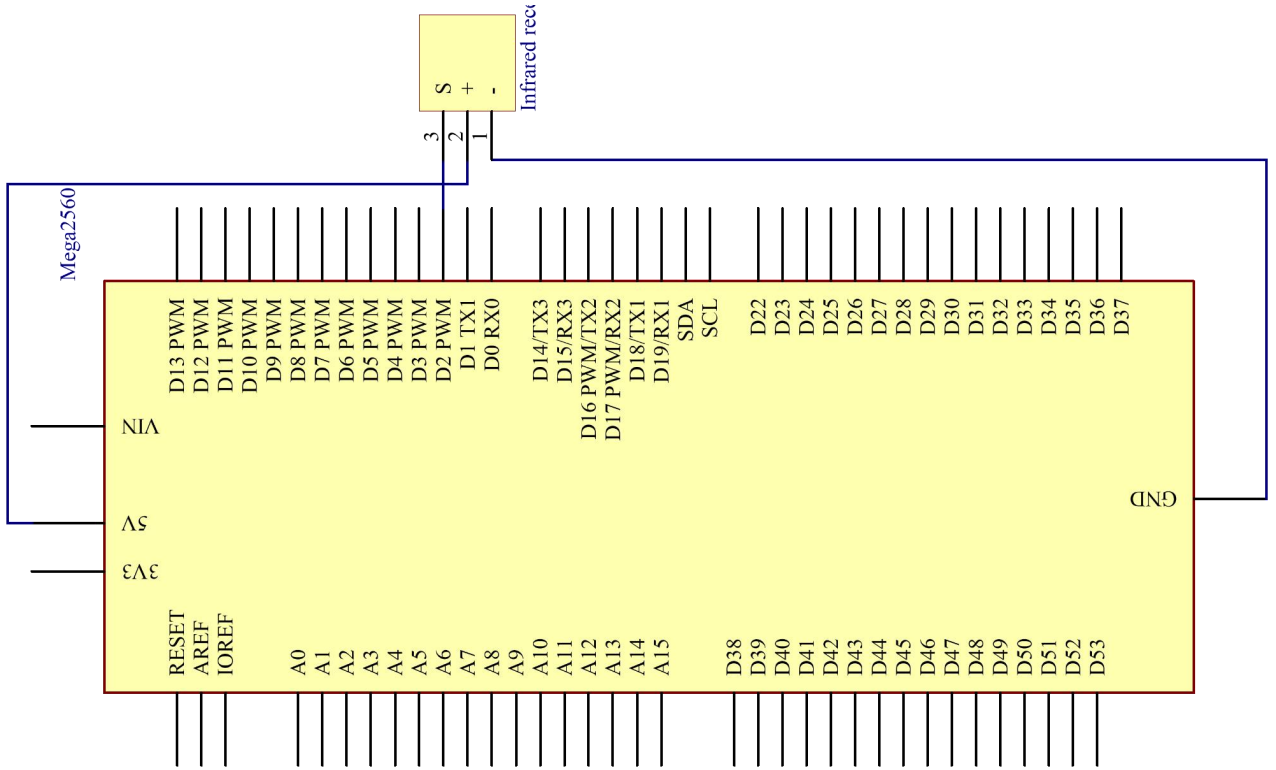
1 * USB cable

Several jumper wires

Experimental Principle

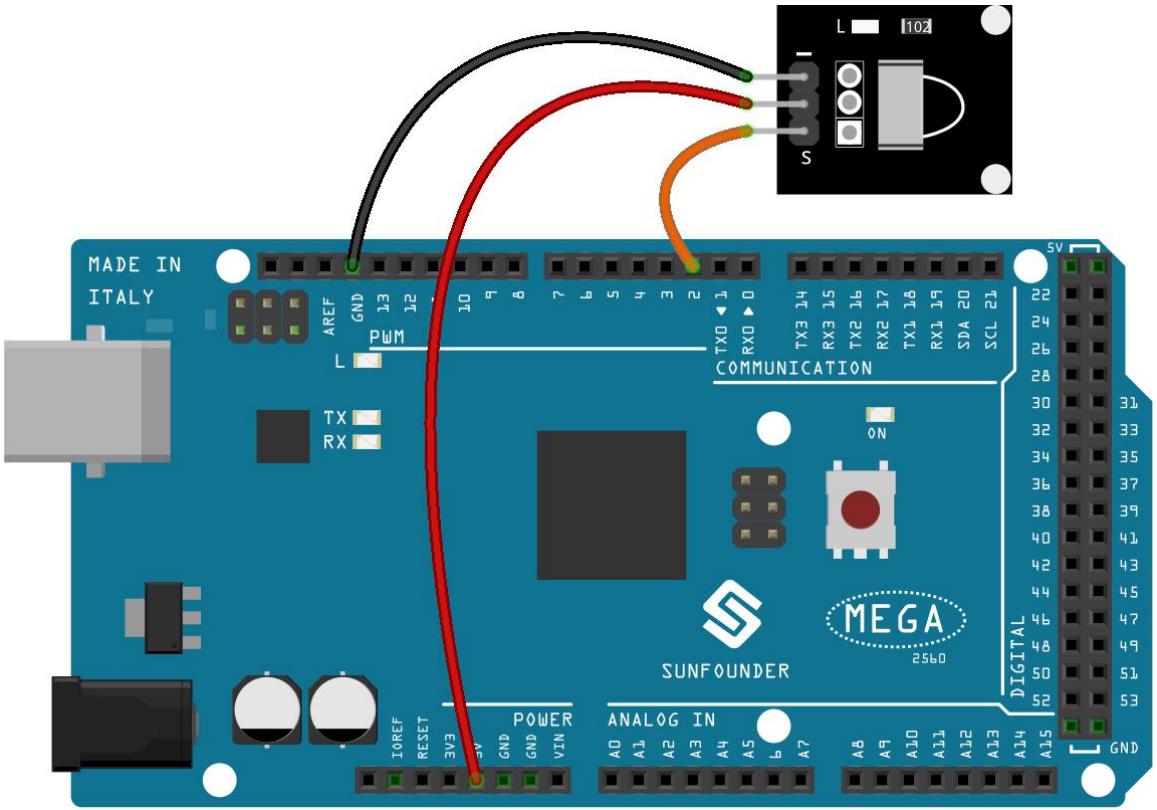
Control a certain key (for example, Power key) via a remote controller by programming. When you press the key, infrared rays will be emitted from the remote controller and received by the infrared receiver, and the LED on the Mega 2560 board will light up.

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit



Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

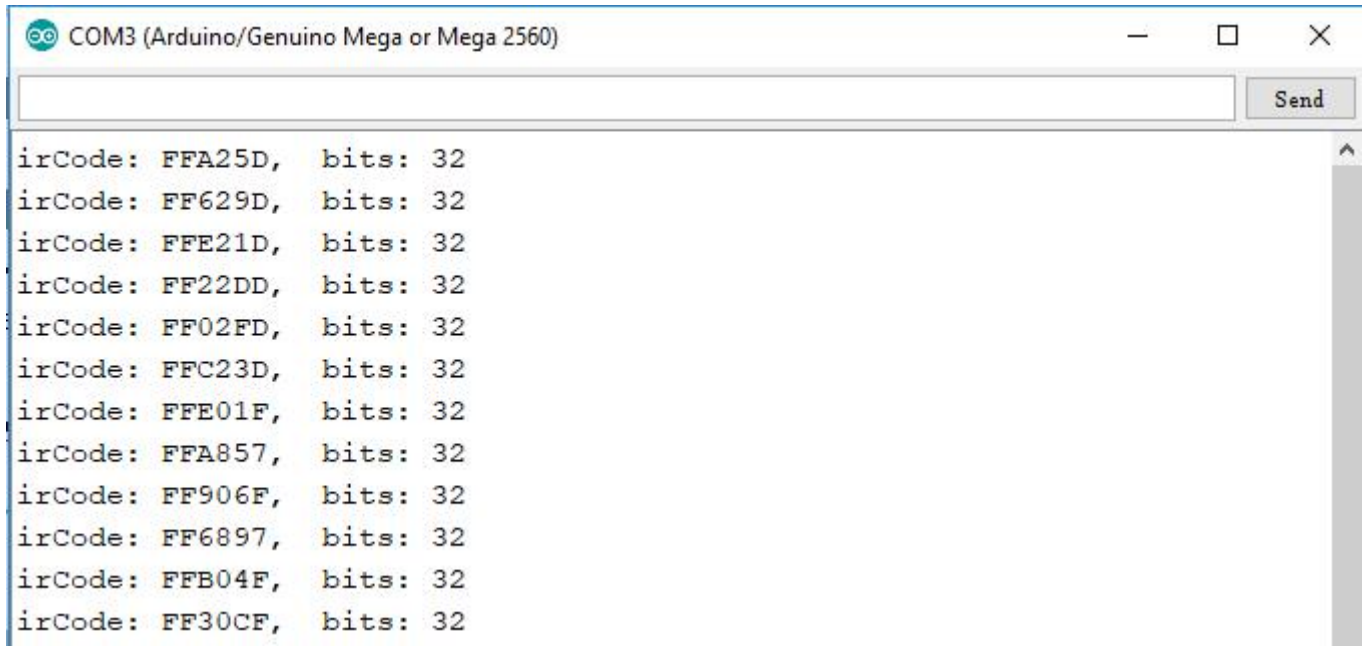
Step 4: Upload the sketch to the board.

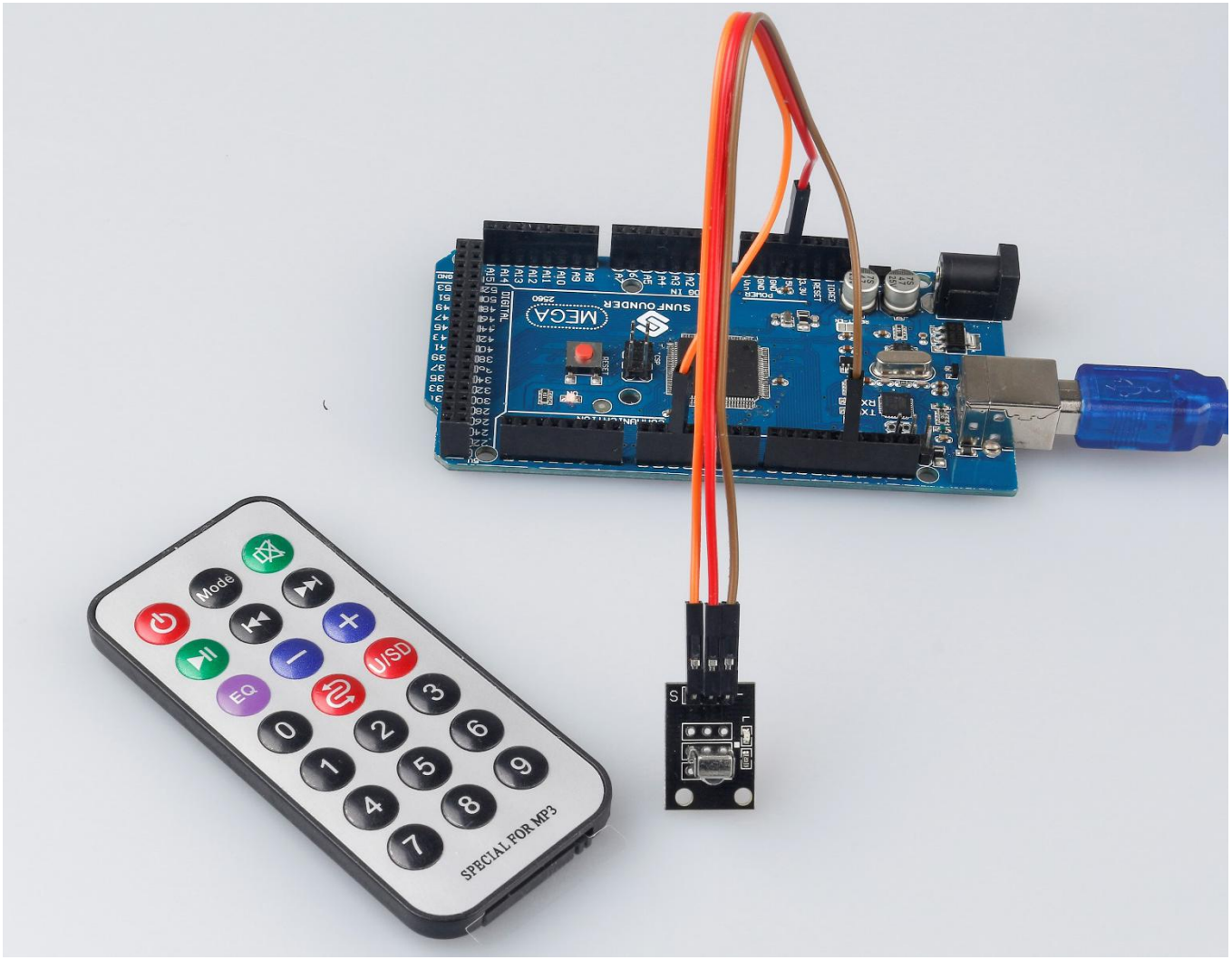
Now, press Power on the remote control and the LED attached to pin 13 on the Mega 2560 board will light up. If you press other keys, the LED will go out.

Note:

1) There is a transparent plastic piece at the back of the remote control to cut off the power and pull it out before you use the remote control.

2) Please gently press the button on the remote to avoid invalid data FFFFFFFF.




Mode
EQ
U/SD
SPECIAL FOR MP3
MEGA
SUNFOUNDER

Code Analysis

Code Analysis 16-1 Initialize the infrared-receiver

```
#include <IRremote.h>
const int irReceiverPin = 2;  //the infrared-receiver attact to pin2
const int ledPin = 13; //built-in LED
IRrecv irrecv(irReceiverPin); //Initialize the infrared-receiver
decode_results results;  //The decoding result is placed in the result of the decode results structure.
```

Code Analysis 16-2 Enable infrared-receiver

```
irrecv.enableIRIn();  //Restart the receiver
```

Code Analysis 16-3 Receive and print the data

```
  if (irrecv.decode(&results)) {   //If receive a data
```

decode(&results): Decodes the received IR message, returns 0 if no data ready, 1 if data ready. Results of decoding are stored in results

```
    Serial.print("irCode: ");   //print "irCode: " on the serial monitor
    Serial.print(results.value, HEX); //print the signal on serial monitor in hexadecimal
    Serial.print(",  bits: ");
    Serial.println(results.bits); // Print the data bits
    irrecv.resume();  //Receive next data
  }
```

```
  delay(600);
```

Code Analysis 16-4 If the power button is pressed

```
  if(results.value == 0xFFA25D)  // if the power button on the remote control is pressed
```

The 0xFFA25D is the code of the power button on the remote control, if you want to define other button, you can read the code of every button from the serial monitor.


```
{
    digitalWrite(ledPin,HIGH);  //Turn on the LED
  }
  else
  {
    digitalWrite(ledPin,LOW);  //else turn of the LED
```

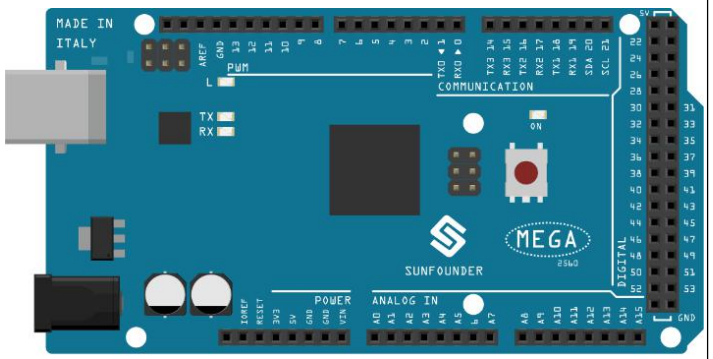
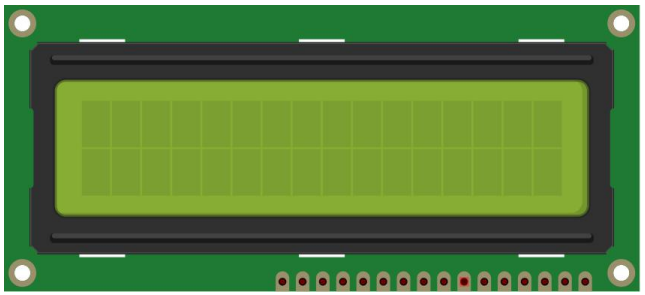
Lesson 17 Humiture Sensor

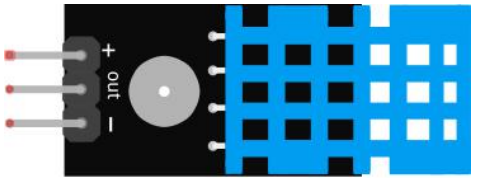
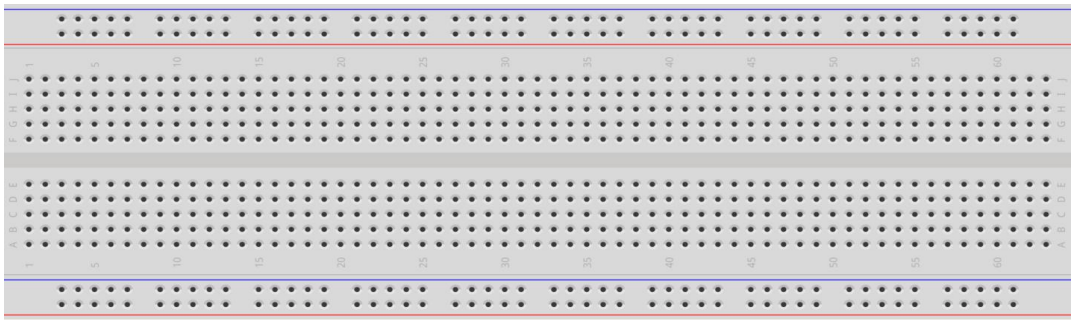
Introduction

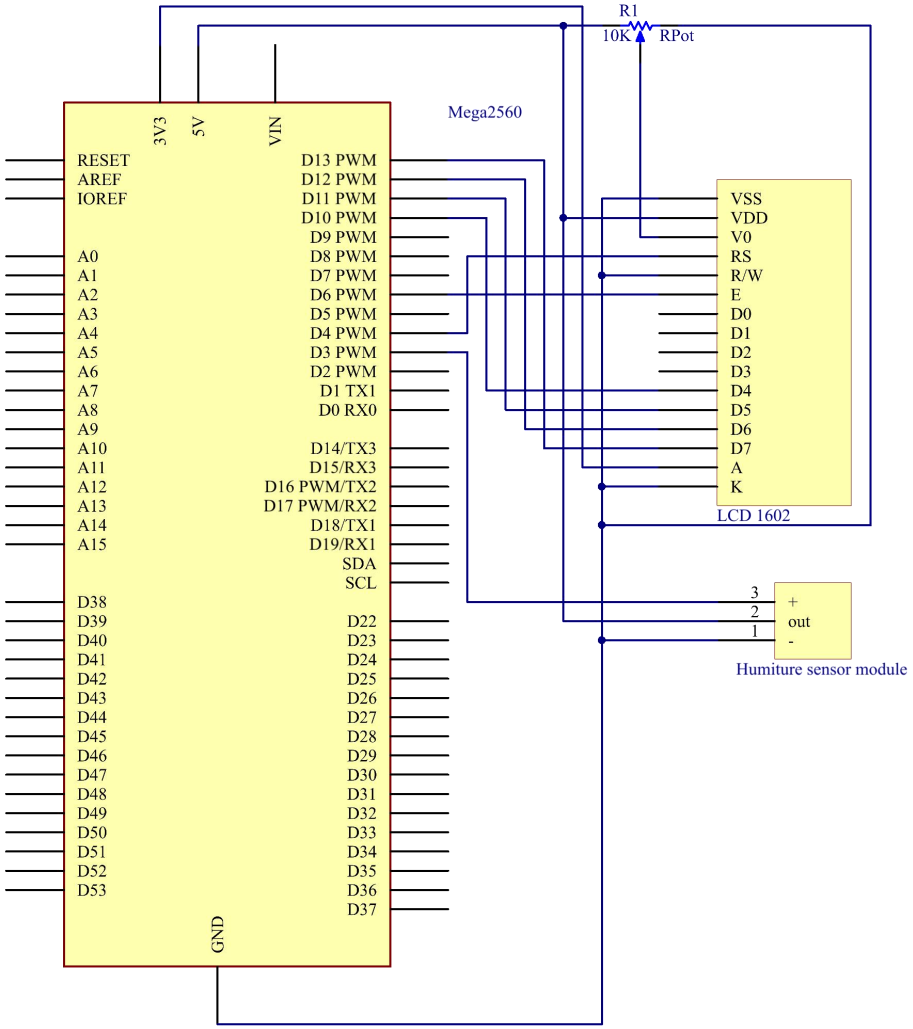
The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature and humidity sensing technology are applied to ensure that the product has high reliability and excellent long-term stability.

The sensor includes a resistive sense of wet component and an NTC temperature measurement device, and is connected with a high-performance 8-bit microcontroller.

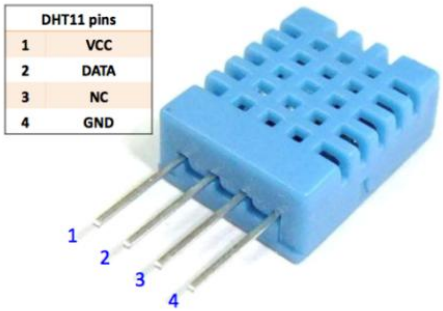
Components

1 * Mega 2560 Board 	1 * LCD1602 

1 * Potentiometer (10kΩ) 	1 * Humiture Sensor Module 
1 * Breadboard 	
1 * USB cable 	Several jumper wires 



Experimental Principle

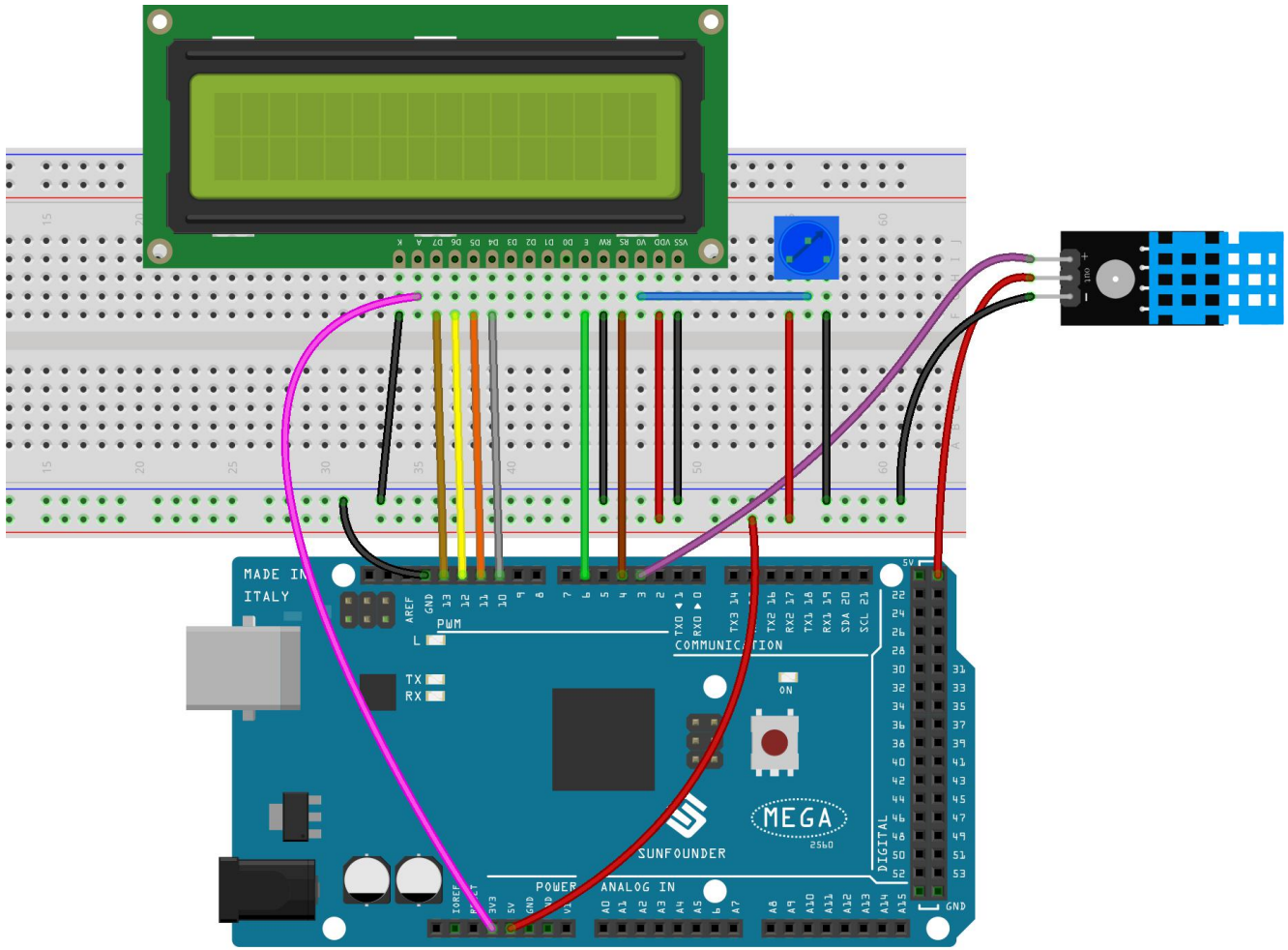


Only three pins are available for use: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humiture data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum). For more information, please refer to DHT11 datasheet.

The schematic diagram:

Experimental Procedures

Step 1: Build the circuit

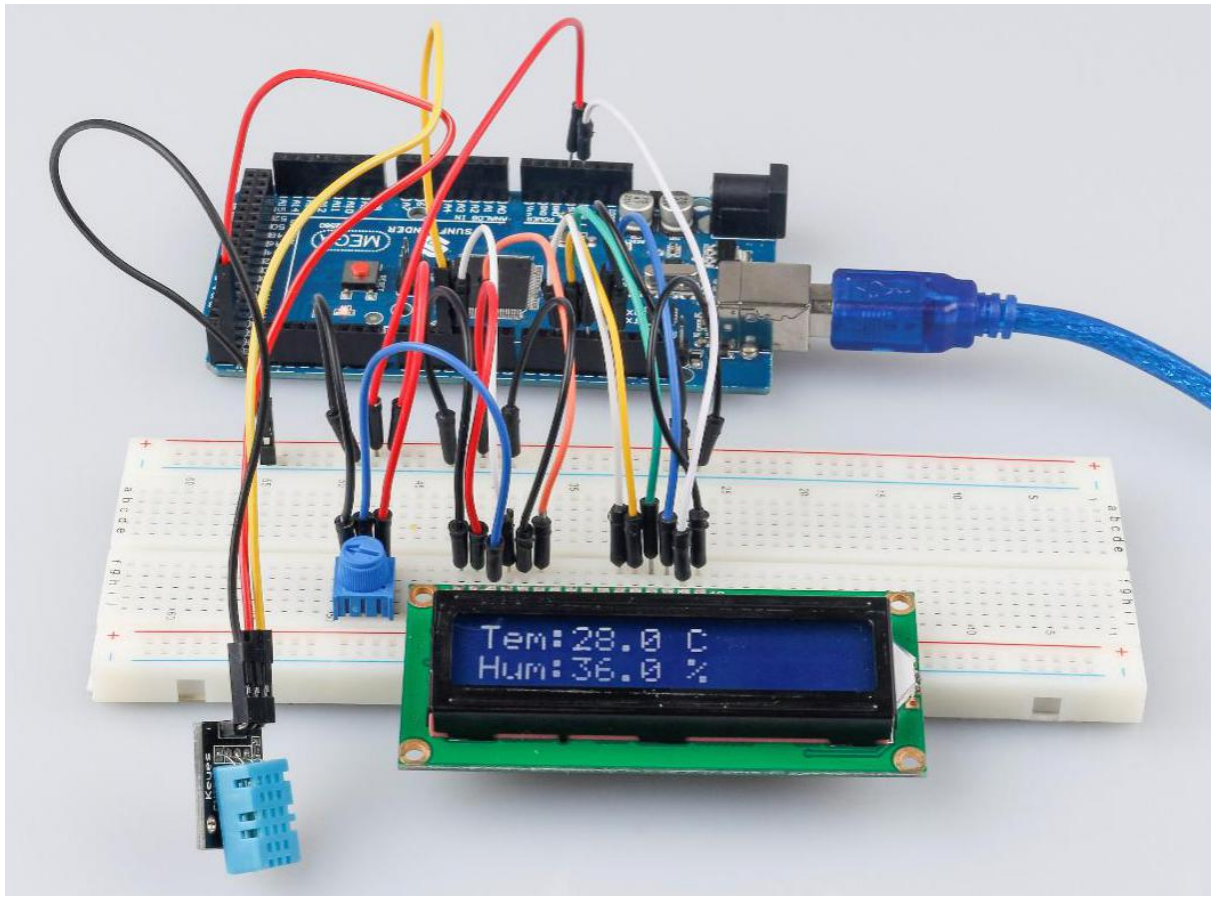


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you can see the value of the current humidity and temperature displayed on the LCD.



Code Analysis

Code Analysis 17-1 Initialize the humiture and LCD1602

```
#include <dht.h>   //Include the head file dht.h

#include <LiquidCrystal.h>  //

LiquidCrystal lcd(4, 6, 10, 11, 12, 13);  // initialize the LCD1602

dht DHT;

#define DHT11_PIN 3  //the humiture sensor attact to pin3
```

Code Analysis 17-2 Read the value of humiture

```
int chk = DHT.read11(DHT11_PIN);

switch (chk)

  {

    case DHTLIB_OK:

       Serial.println("OK,\t");

       break;

    case DHTLIB_ERROR_CHECKSUM:
```

```
        Serial.println("Checksum error,\t");
        break;
    case DHTLIB_ERROR_TIMEOUT:
        Serial.println("Time out error,\t");
        break;
    default:
        Serial.println("Unknown error,\t");
        break;
}
```

Use the read11() function to read the value of the temperature and humidity sensor. If OK is displayed on the Serial Monitor, the humiture sensor is working properly.

read11(): Return values:

// DHTLIB_OK: Indicate the humiture sensor is work well.

// DHTLIB_ERROR_CHECKSUM

// DHTLIB_ERROR_TIMEOUT

Code Analysis 17-3 Display on the LCD1602

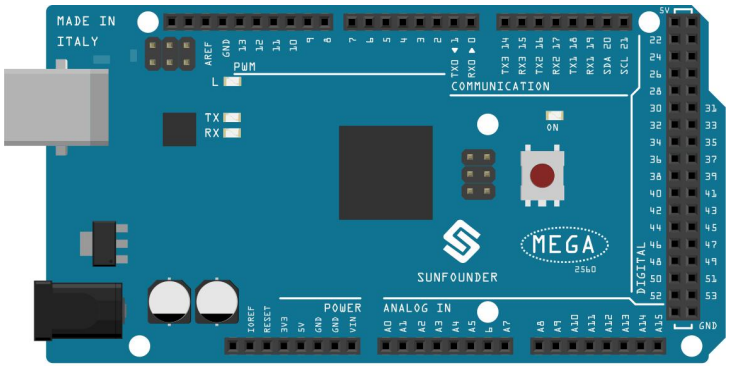
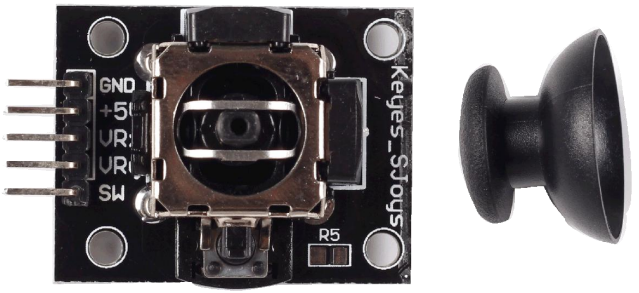
```
lcd.setCursor(0, 0);
 lcd.print("Tem:");
 lcd.print(DHT.temperature,1); //print the temperature on lcd, keep one decimal point
 lcd.print(" C");
 lcd.setCursor(0, 1);
 lcd.print("Hum:");
 lcd.print(DHT.humidity,1); //print the humidity on lcd
 lcd.print(" %");
 delay(200); //wait a while
```

Lesson 18 Joystick PS2

Introduction

A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. Joysticks are often used to control video games and robots. A Joystick PS2 is used here.

Components

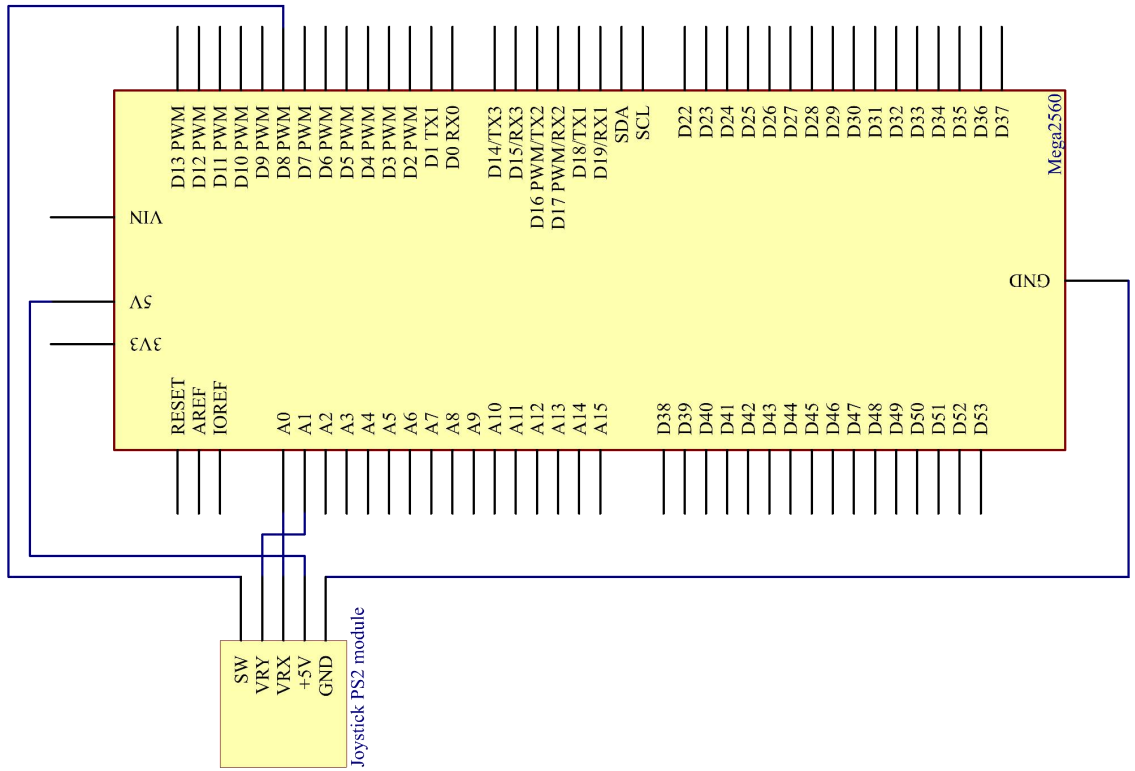
1 * Mega 2560 Board 	1 * Joystick PS2 module 
1 * USB cable 	Several jumper wires 

Experimental Principle

This module has two analog outputs (corresponding to X, Y biaxial offsets) and one digital output representing whether it is pressed on Z axis. The module integrates power indicator and can display operation condition.

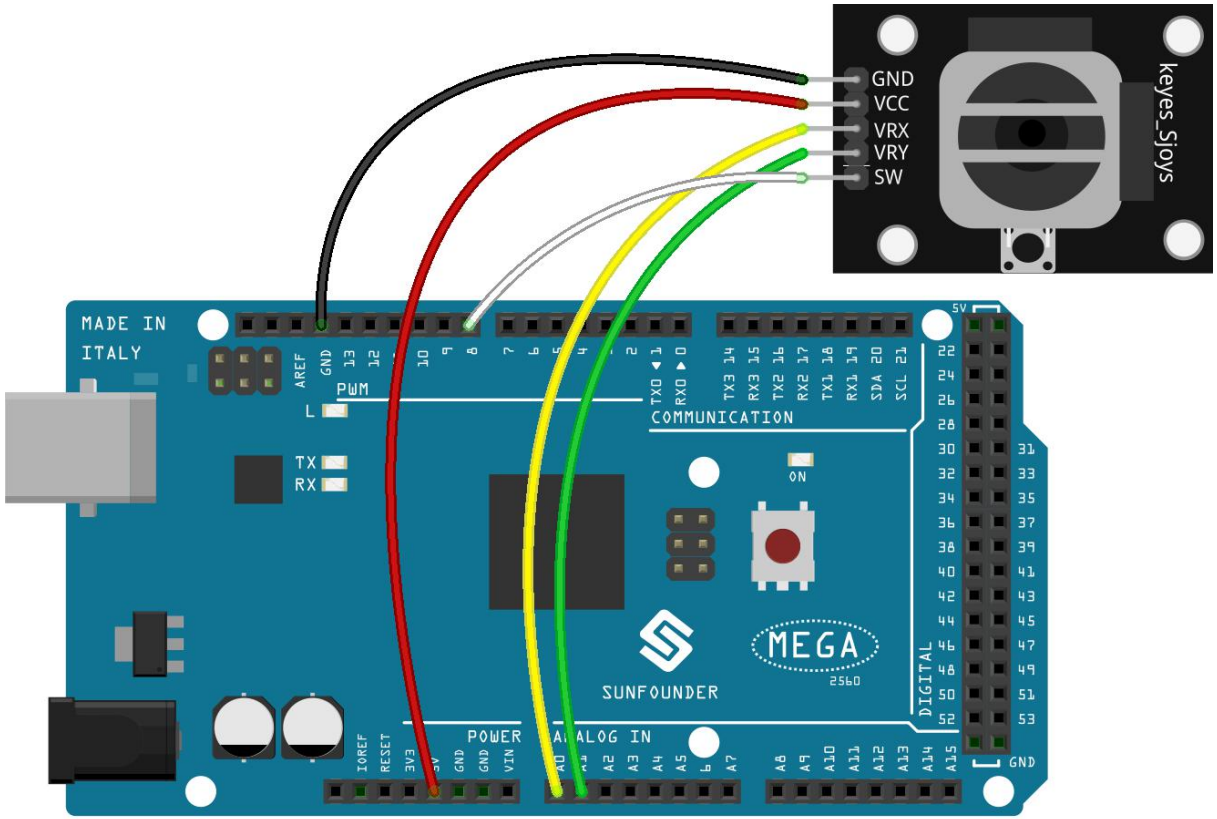
In this experiment, we use the Mega 2560 board to detect the moving direction of the Joystick knob and pressing of the button.

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit

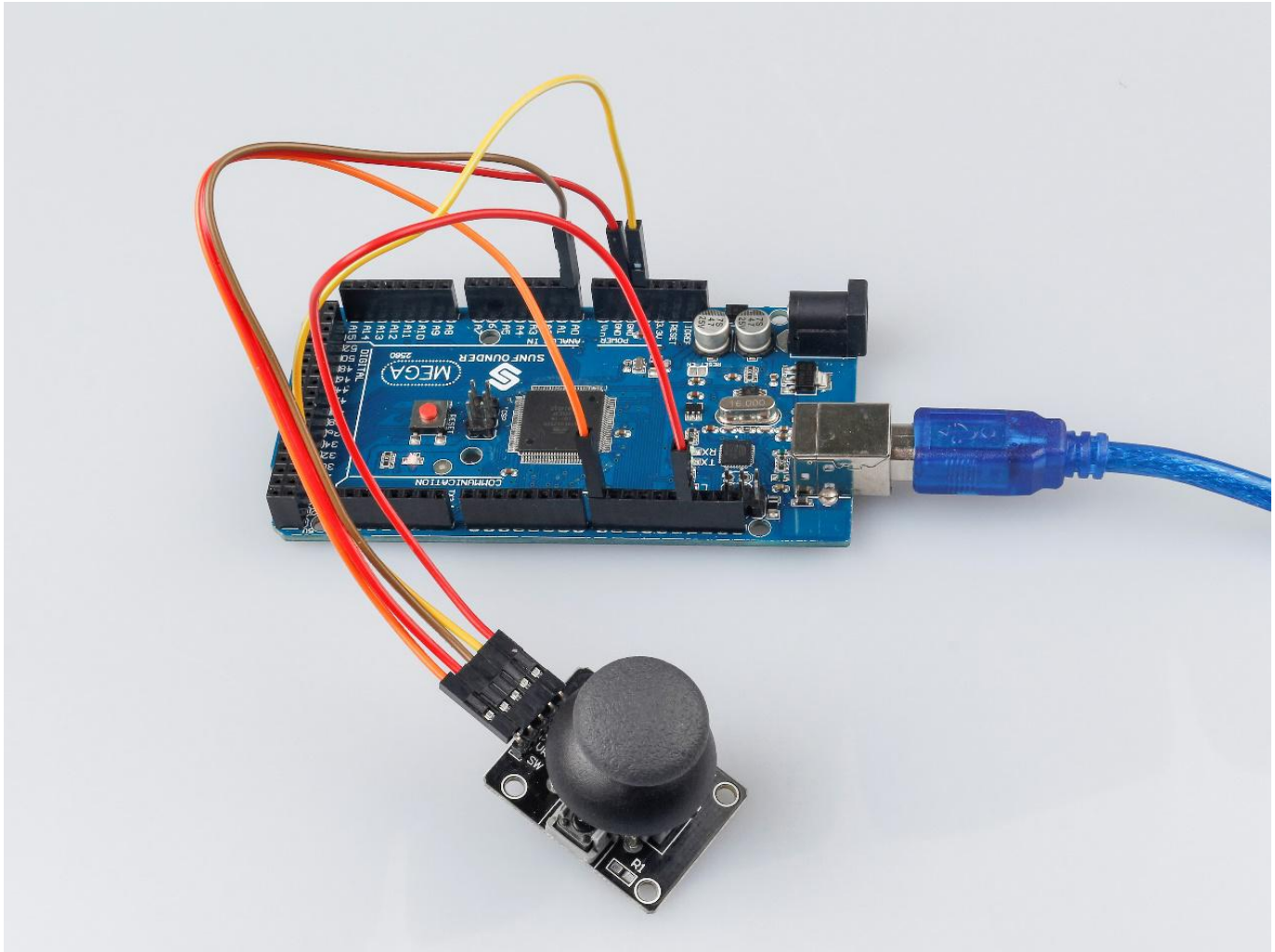


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, push the rocker and the coordinates of X and Y axes displayed on Serial Monitor will change accordingly; press the button, and the coordinate of Z=0 will also be displayed.



Code Analysis

The code is use the Serial monitor to print the value of the VRX,VRY and SW pins of the joystick ps2.

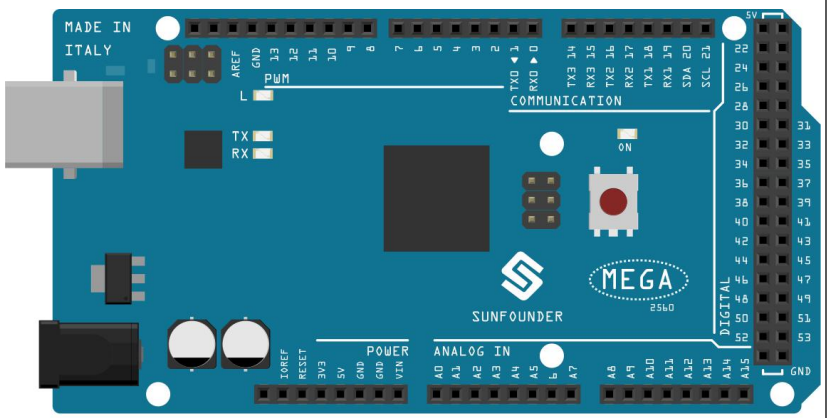
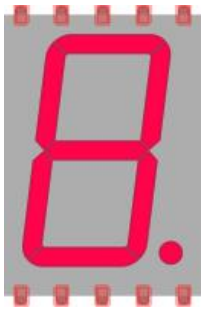
```
const int xPin = A0;  //the VRX attach to
const int yPin = A1;  //the VRY attach to
const int swPin = 8;  //the SW attach to
void setup()
{
  pinMode(swPin, INPUT);  //set the SW pin to INPUT
  digitalWrite(swPin, HIGH);   //And initial value is HIGH
  Serial.begin(9600);
}
void loop()
{
  Serial.print("X: ");
  Serial.print(analogRead(xPin), DEC);  // print the value of VRX in DEC
  Serial.print("|Y: ");
  Serial.print(analogRead(yPin), DEC);  // print the value of VRX in DEC
  Serial.print("|Z: ");
  Serial.println(digitalRead(swPin));   // print the value of SW
  delay(500);}
```

Lesson 19 7-Segment Display

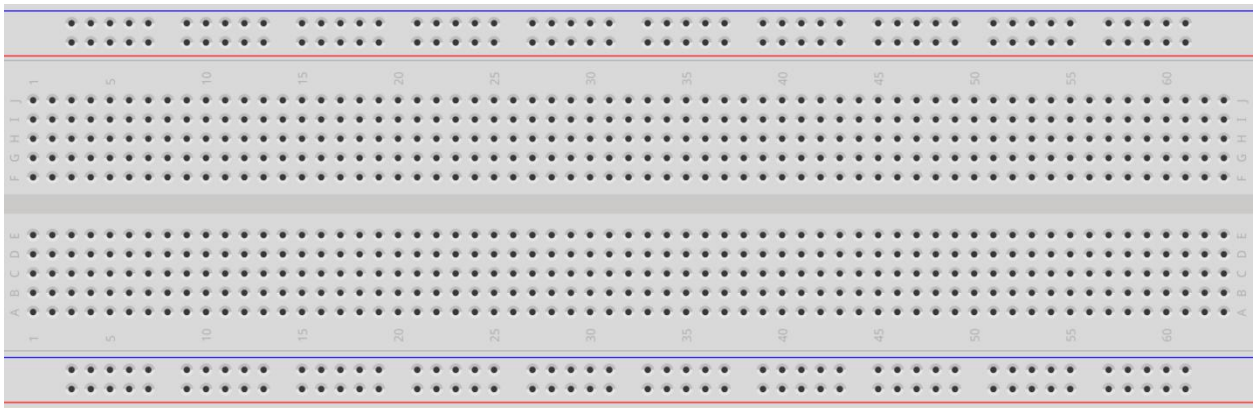
Introduction

A 7-segment display is a device that can display numerals and letters. It's made up of seven LEDs connected in parallel. Different letters/numbers can be shown by connecting pins on the display to the power source and enabling the related pins, thus turning on the corresponding LED segments. In this lesson let's learn how to display specific characters on it.

Components

1 * Mega 2560 Board	8 * Resistor (220Ω)	1 * 7-segment display **Common Cathode**
		

1 * Breadboard



1 * USB cable

Several jumper wires

Principle

7-Segment Display

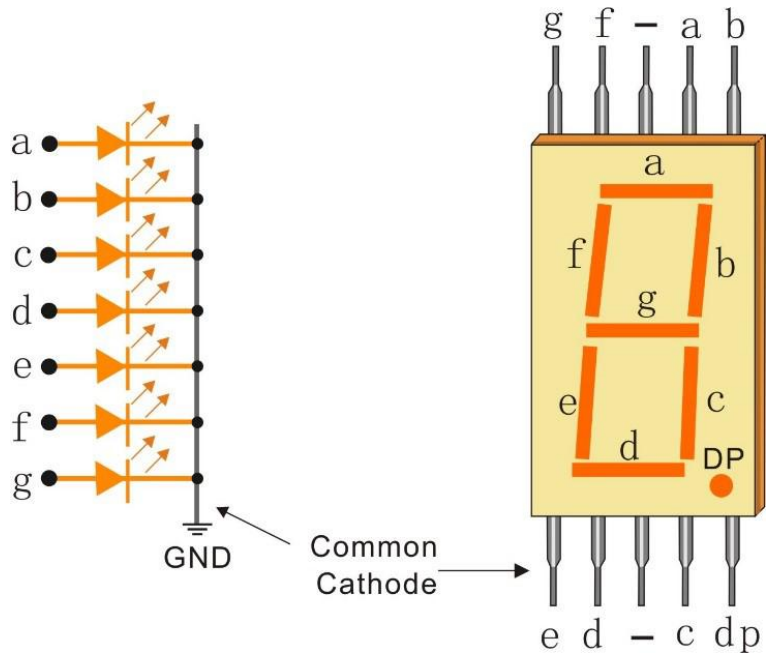
A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral (both decimal and hexadecimal) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from "a" through to "g" representing each individual LED. The other LED pins are connected together forming a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

The common pin of the display generally tells its type. There are two types of pin connection: a pin of connected cathodes and one of connected anodes, indicating Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected.

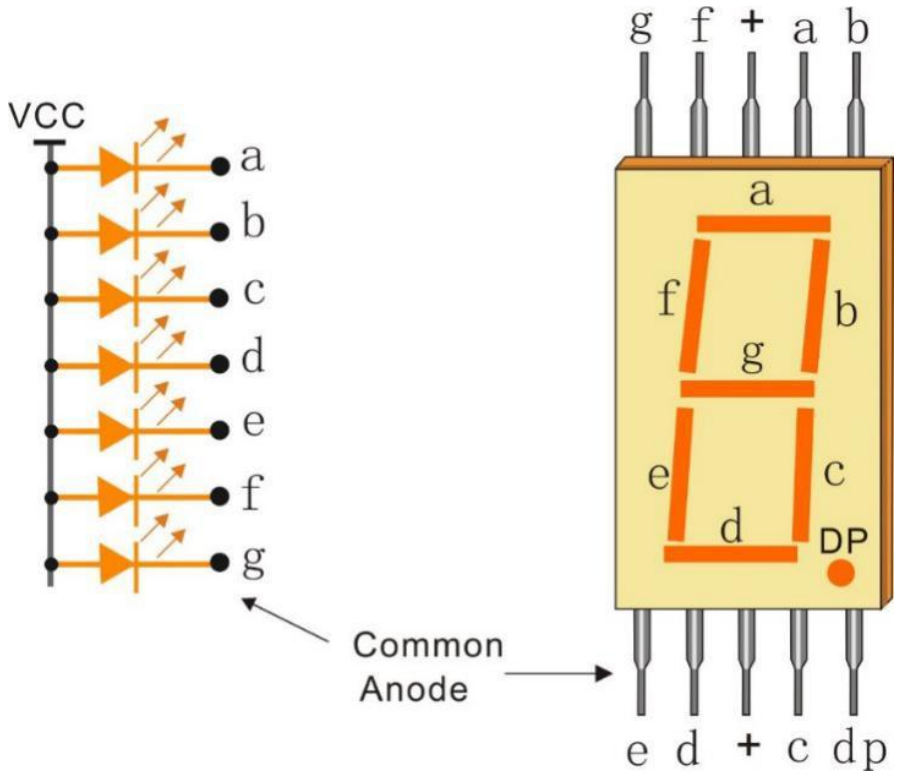
Common Cathode 7-Segment Display

In a common cathode display, the cathodes of all the LED segments are connected to the logic "0" or ground. Then an individual segment (a-g) is energized by a "HIGH", or logic "1" signal via a current limiting resistor to forward bias the anode of the segment.



Common Anode 7-Segment Display

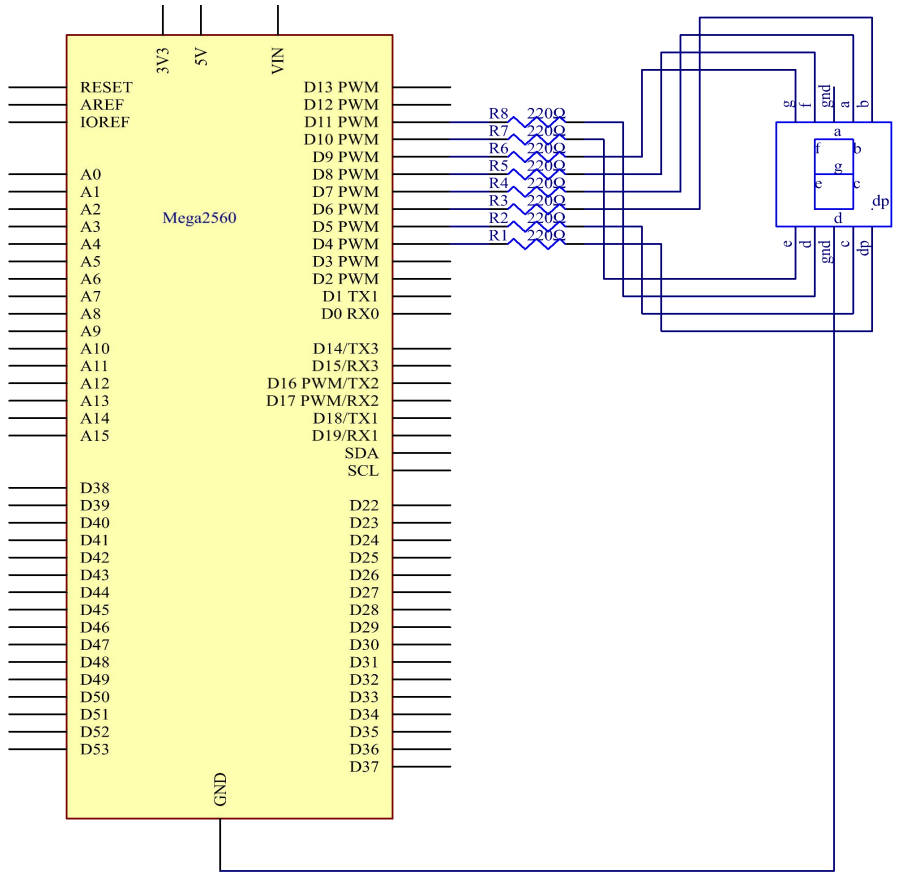
In a common anode display, the anodes of all the LED segments are connected to the logic "1". Then an individual segment (a-g) is energized by a ground, logic "0" or "LOW" signal via a current limiting resistor to the cathode of the segment.



Principle:

In this experiment, connect each of pin a-g of the 7-Segment Display to one 220ohm current limiting resistor respectively and then to pin 4–11. GND connects to GND. By programming, we can set one or several of pin4-11 as High level to light up the corresponding LED(s).

The schematic diagram:

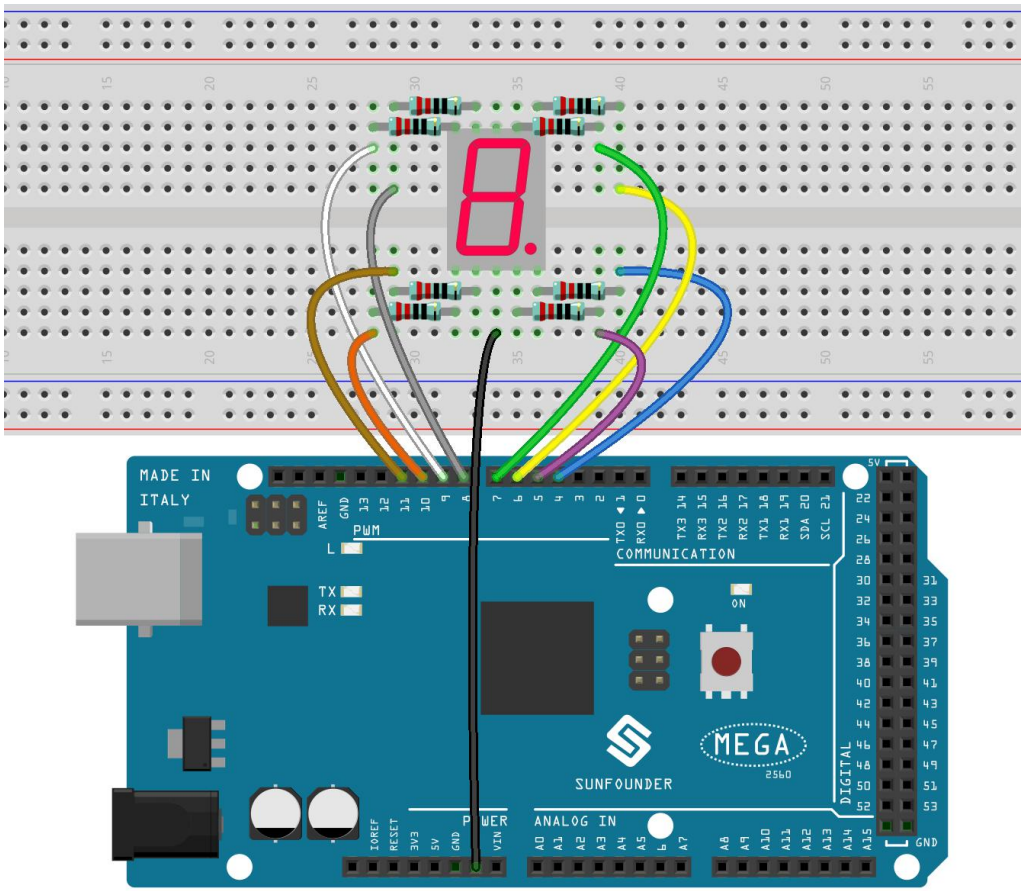


Experimental Procedures

Step 1: Build the circuit (here a common cathode 7-segment display is used)

The wiring between the 7-segment display and the Mega2560 board :

7-Segment	Mega2560 Board
a	7
b	6
c	5
d	11
e	10
f	8
g	9
dp	4
“ - “	GND

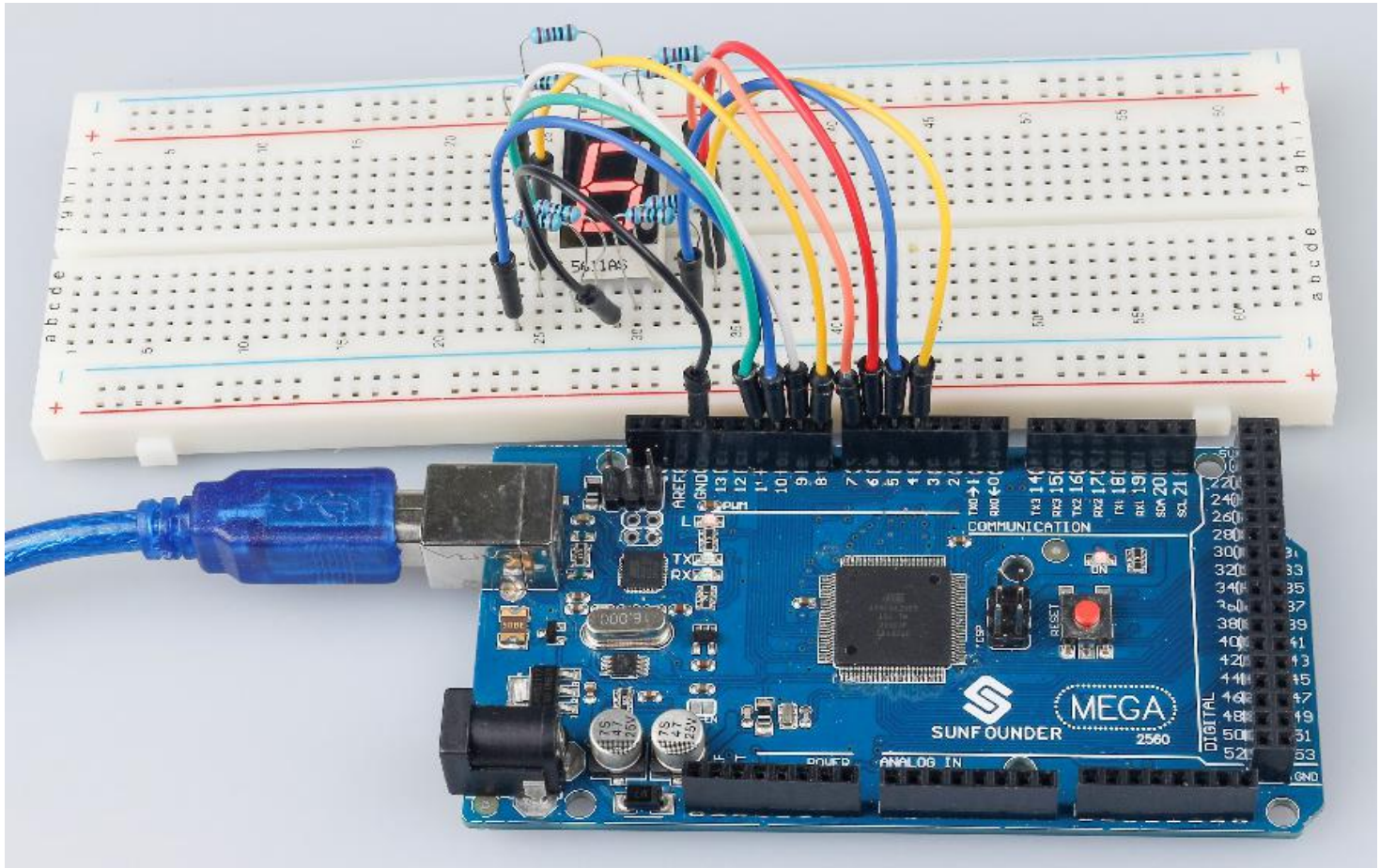


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

You should now see the 7-segment display from 0 to 9 and then A to F, back and forth.



Code Analysis

The code may be a little long for this experiment. But the syntax is simple. Let's take a look.

Code Analysis 19-1 Call the function in loop()

```
digital_1();//diaplay 1 to the 7-segment
delay(1000);//wait for a second
digital_2();//diaplay 2 to the 7-segment
delay(1000); //wait for a second
digital_3();//diaplay 3 to the 7-segment
delay(1000); //wait for a second
digital_4();//diaplay 4 to the 7-segment
```

…

Calling these functions into the loop() is to let the 7-Segment display 0-F. The functions are shown below. Take *digital_2()* for example:

Code Analysis 19-2 Detailed analysis of digital_2()

```
void digital_2(void) //diaplay 2 to the 7-segment
{
  digitalWrite(b,HIGH);
  digitalWrite(a,HIGH);
  for(int j = 9;j <= 11;j++)
    digitalWrite(j,HIGH);
  digitalWrite(c,LOW);
  digitalWrite(f,LOW);
}
```

First we need to know how it looks like when display the numeral **2** on the 7-Segment display. It's actually the segments a, b, d, e and g are power on, which generates the display of **2**. In programming, pins connected to these segments are set High level when c and f are Low level. Here we use a *for()* statement to set these pins as High level respectively (the braces after *for()* are deleted as there is only one line). Connect pin dp to pin 4; it's already defined as LOW in *setup()*.

After running this part, the 7-segment will display **2**. Similarly, the display of other characters are the same. Since the letters b and d in upper case, namely **B** and **D**, would look the same with **8** and **0** on the display, they are displayed in lower case instead.

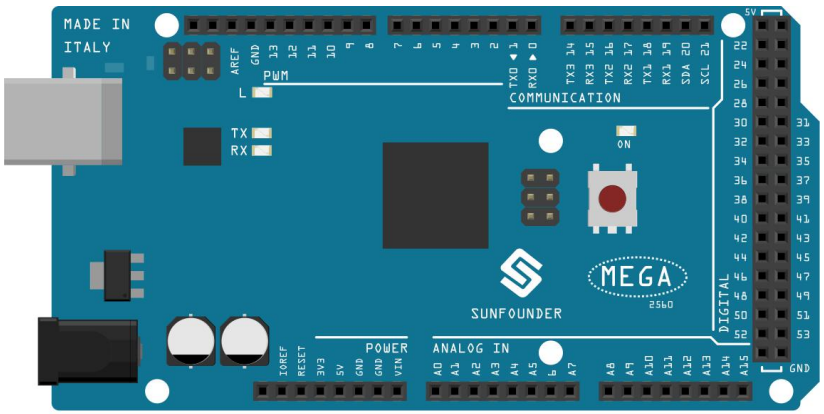
Lesson 20 74HC595

Introduction

Generally, there are two ways to drive a single 7-segment display. One way is to connect its 8 pins directly to eight ports on the Mega 2560 board, which we have done previously. Or you can connect the 74HC595 to three ports of the Mega 2560board and then the 7-segment display to the 74HC595. In this experiment, we will use the latter. By this way, we can save five ports – considering the Mega 2560 board's limited ports, this is very important. Now let's get started!

Components

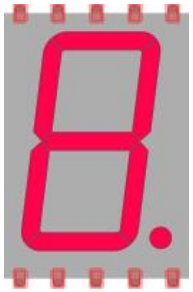
1 * Mega 2560 Board



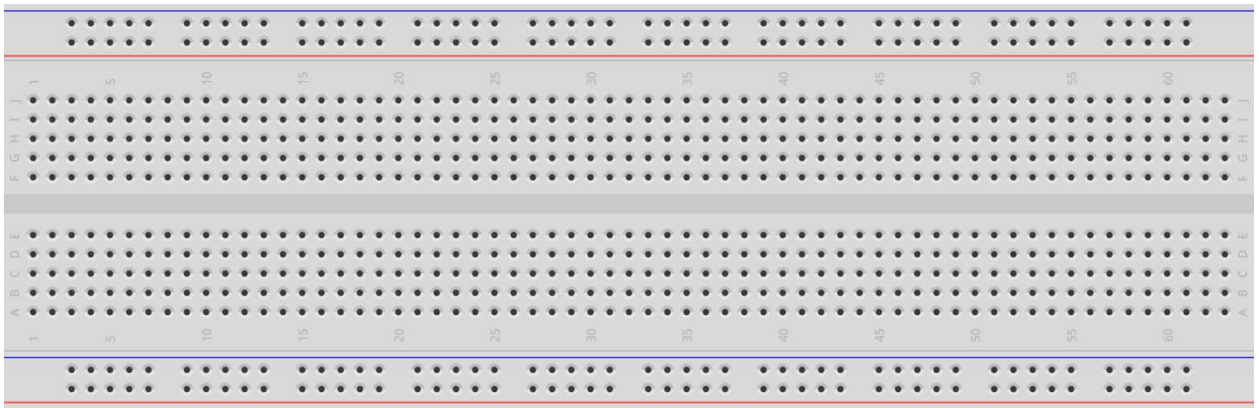
8 * Resistor (220Ω)

1 * 74HC595

1 * 7-segment display



1 * Breadboard



1 * USB cable

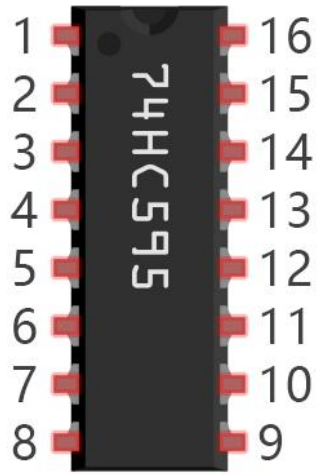
Several jumper wires

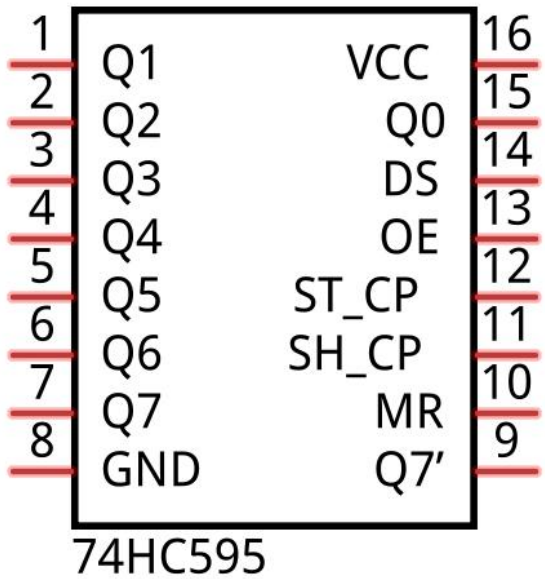
Experimental Principle

74HC595

The 74HC595 consists of an 8−bit shift register and a storage register with three−state parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU.

When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp. If the two clocks are connected together, the shift register is always one pulse earlier than the memory register. There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register. The memory register outputs a Bus with a parallel 8-bit and in three states. When OE is enabled (low level), the data in memory register is output to the bus.





Pins of 74HC595 and their functions:

Q0-Q7: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.

Q7': Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series

MR: Reset pin, active at low level; here it is directly connected to 5V.

SHcp: Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

STcp: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

OE: Output enable pin, active at low level. Here connected to GND.

DS: Serial data input pin

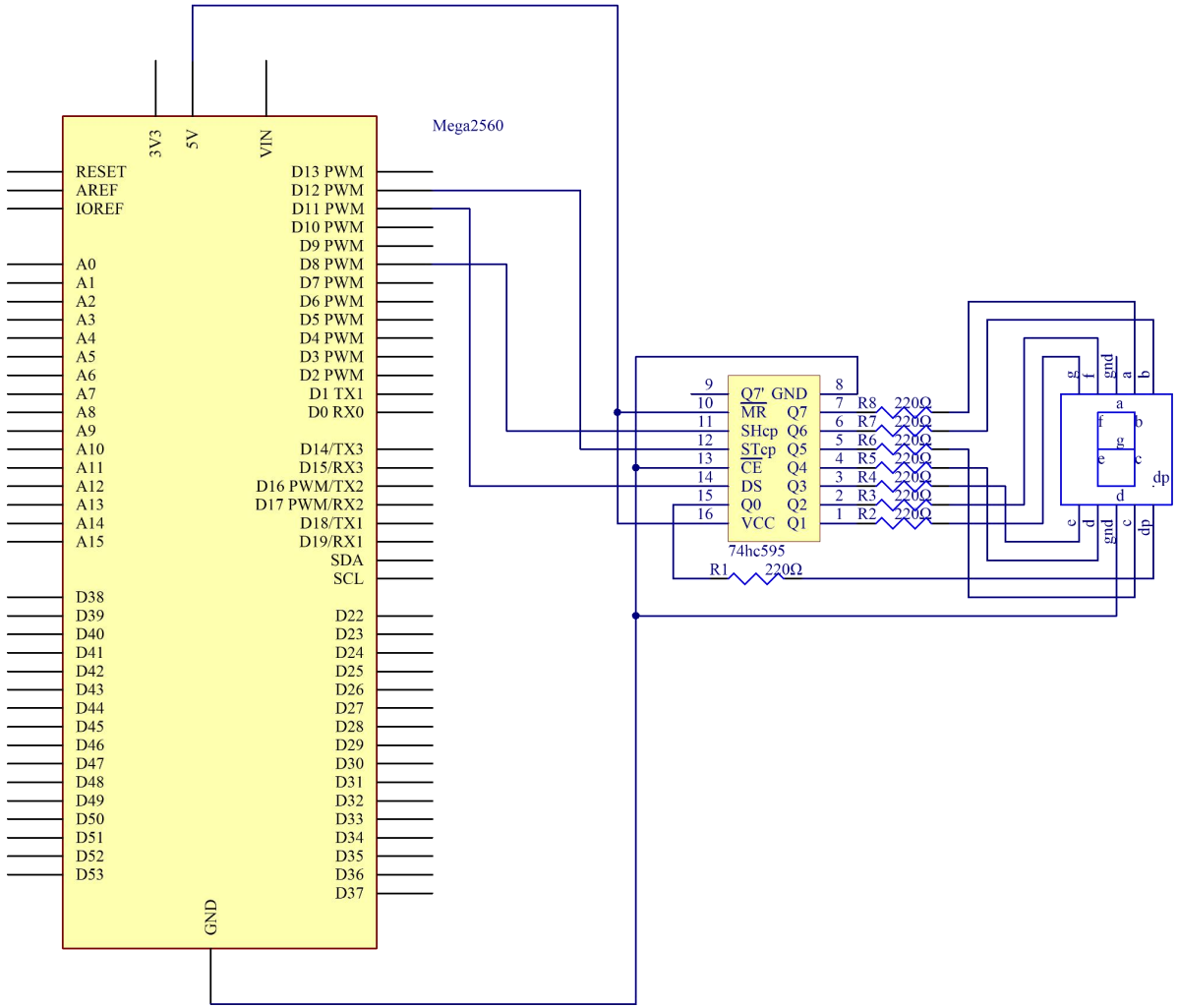
VCC: Positive supply voltage

GND: Ground

Principle:

In the experiment MR (pin10) is connected to 5V (HIGH Level) and OE (pin 13) to GND (LOW Level). Therefore, the data is input into the rising edge of SHcp and enters the memory register through the rising edge. We use the shiftout() function to output a 8-bit data to the shift register through DS. In the rising edge of the SHcp, the data in the shift register moves successively one bit in one time, i.e. data in Q1 moves to Q2, and so forth. In the rising edge of STcp, data in the shift register moves into the memory register. All data will be moved to the memory register after 8 times. Then the data in the memory register is output to the bus (Q0-Q7). So the 16 characters are displayed in the 7-segment in turn.

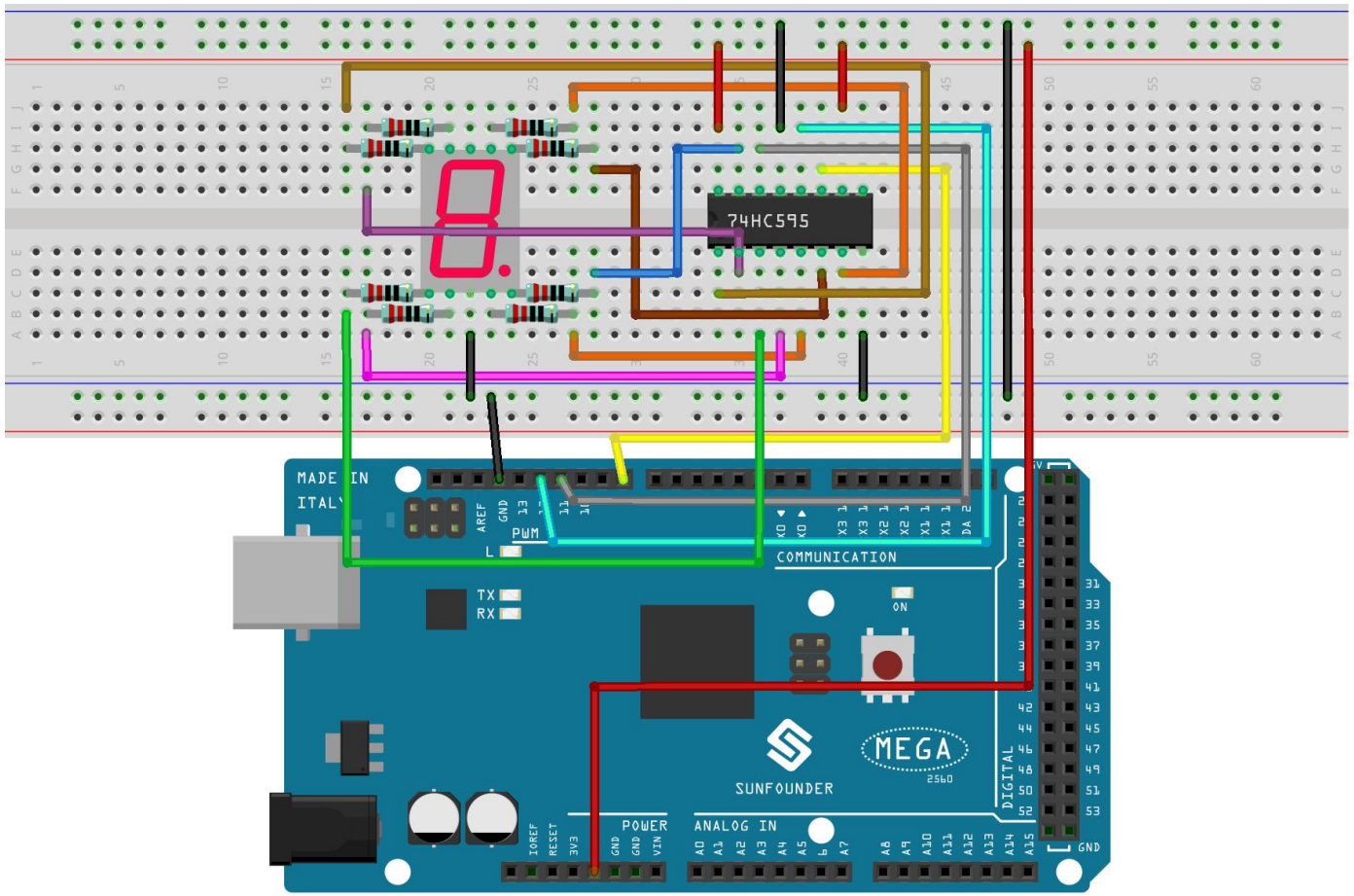
The schematic diagram:



Experimental Procedures

Step 1: Build the circuit (pay attention to the direction of the chip by the concave on it)

7-Segment Display	74HC595	Mega2560 Kit
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	5V
	DS	11
	CE	GND
	ST	12
	SH	8
	MR	5V
	Q7’	N/C
	GND	GND
-		GND

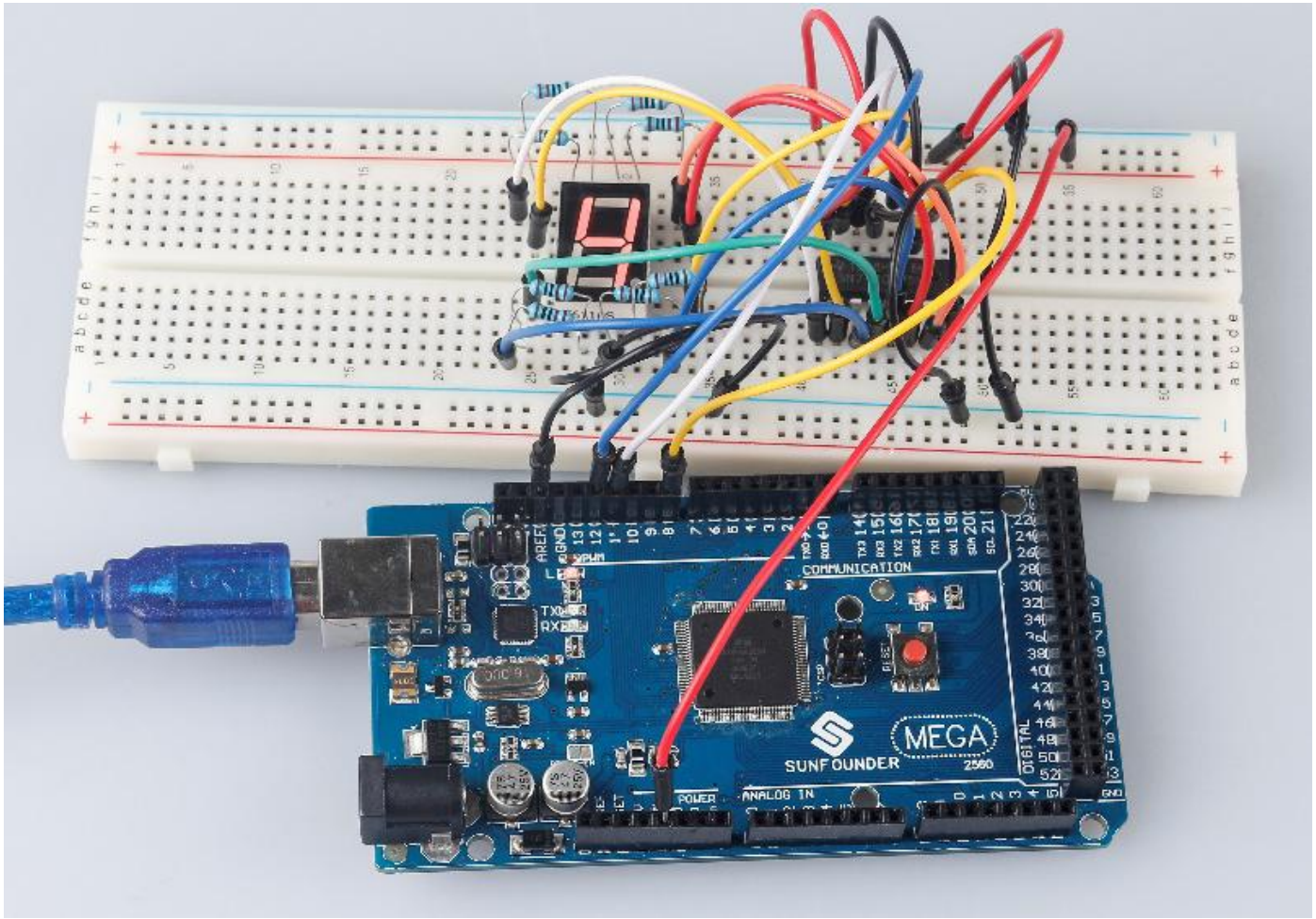


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

You should now see the 7-segment display from 0 to 9 and A to F.



Code Analysis

Code Analysis 20-1 Set the array elements

```
int datArray[16] = {252, 96, 218, 242, 102, 182, 190, 224, 254, 246, 238, 62, 156, 122, 158, 142};
```

This array stores the data of the 16 characters from 0 to F. 252 stands for 0, which you can calculate by yourself. To display 0, the segment g (the middle one) of the 7-segment display must be low level (dim).

Since the segment g is connected to Q1 of the 74HC595, set both Q1 and DP (the dot) as low level and leave the rest pins as high level. Therefore, the values of Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 are 1 1 1 1 1 1 0 0.

Change the binary numbers into decimal ones: $1x2^7+1x2^6+1x2^5+1x2^4+1x2^3+1x2^2+0x2^1+1x2^0=252$.

So that's the value for the number **0** to be displayed. You can calculate other characters similarly.

Code Analysis 20-2 Display 0-F in the 7-segment display

```
for(int num = 0; num < 16; num++)
  {
    digitalWrite(STcp,LOW); //ground ST_CP and hold low for as long as you are transmitting
    shiftOut(DS,SHcp,MSBFIRST,datArray[num]);
    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(STcp,HIGH); //pull the ST_CPST_CP to save the data
    delay(1000); //wait for a second
  }
```

Set *STcp* as low level first and then high level. It will generate a rising edge pulse of STcp.

shiftOut() is used to shift out a byte of data one bit at a time, which means to shift a byte of data in *dataArray[num]* to

the shifting register with the DS pin. *MSBFIRST* means to move from high bits.

After *digitalWrite(STcp,HIGH)* is run, the STcp will be at the rising edge. At this time, the data in the shift register will be moved to the memory register.

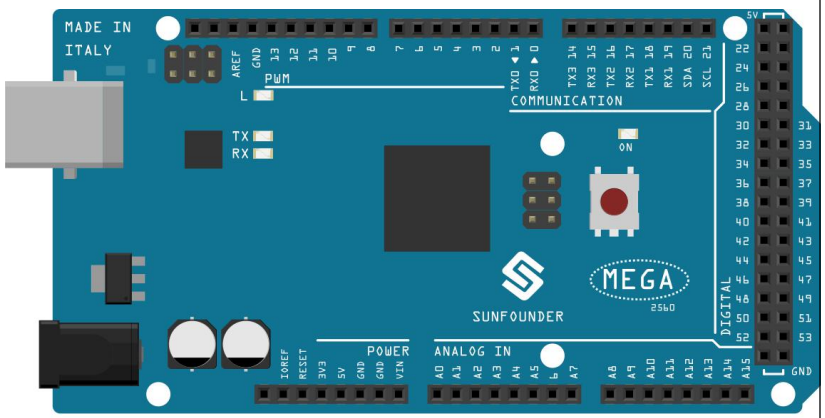
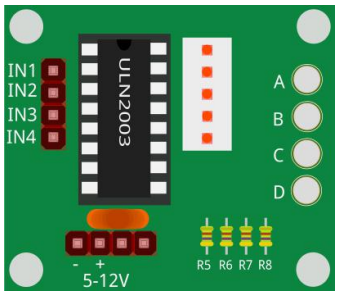
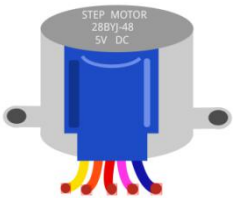
A byte of data will be transferred into the memory register after 8 times. Then the data of memory register is output to the bus (Q0-Q7). You will see a character is displayed on the 7-segment. Then delay for 1000ms. After that line, go back to *for()*. The loop repeats until all the characters are displayed in the 7-segment display one by one after 16 times.

Lesson 21 Stepper Motor

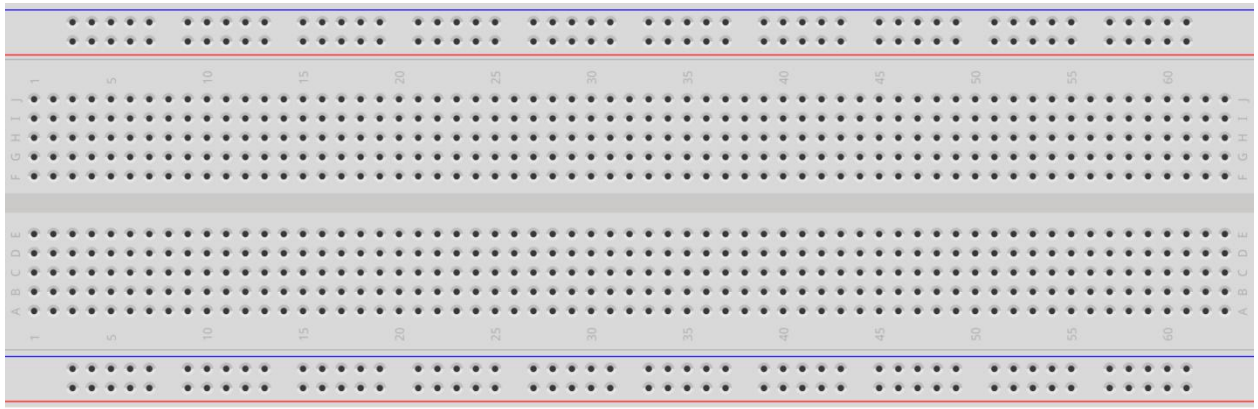
Introduction

Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely moving it forward or backward in small "steps".

Components

1 * Mega 2560 Board 	1 * Stepper Motor Driver 	1 * Stepper Motor  1 * Potentiometer 

1 * Breadboard



1 * USB cable

Several jumper wires

Experimental Principle

There are two types of steppers, unipolars and bipolars, and it is very important to know which type you are working with. In this experiment, we will use a unipolar stepper.

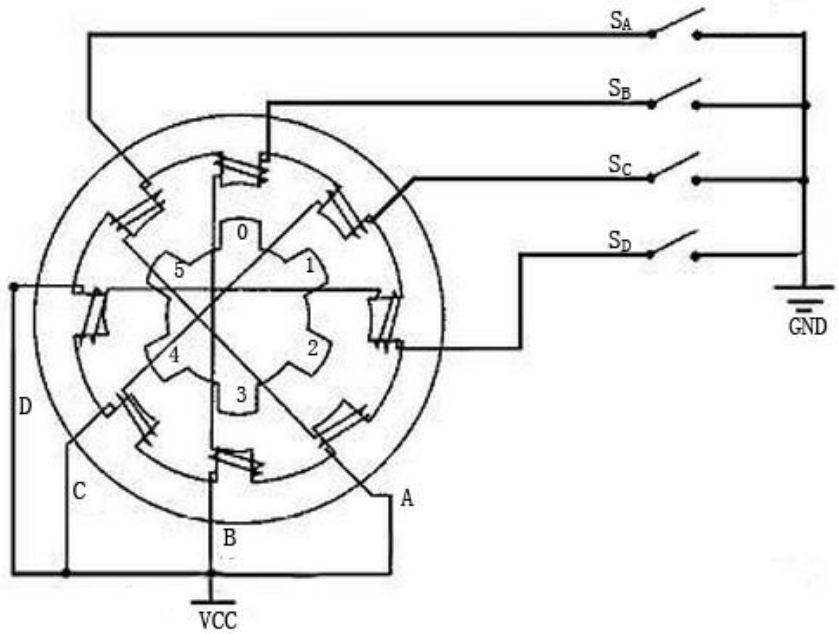
The stepper motor is a four-phase one, which uses a unipolarity DC power supply. As long as you electrify all phase windings of the motor by an appropriate timing sequence, you can make it rotate step by step. The schematic diagram of a four-phase reactive stepper motor:

In the figure, in the middle of the motor is a rotor – a gear-shaped permanent magnet. Around the rotor, 0 to 5 are teeth. Then more outside, there are 8 magnetic poles, with each two opposite ones connected by coil winding. So they form four pairs from A to D, which is called a phase. It has four lead wires to be connected with switches SA, SB, SC, and SD. Therefore, the four phases are in parallel in the circuit, and the two magnetic poles in one phase are in series.

Here's how a 4-phase stepper motor works:

At the beginning, switch SB is power on, switch SA, SC, and SD is power off, and B-phase magnetic poles align with tooth 0 and 3 of the rotor. At the same time, tooth 1 and 4 generate staggered teeth with C- and D-phase poles. Tooth

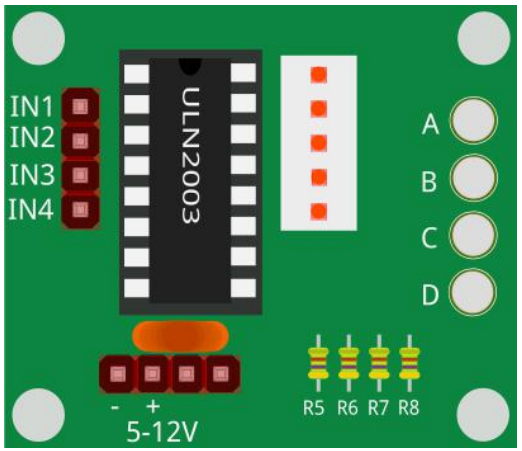
2 and 5 generate staggered teeth with D- and A-phase poles. When switch SC is power on, switch SB, SA, and SD is power off, the rotor rotates under magnetic field of C-phase winding and that between tooth 1 and 4. Then tooth 1 and 4 align with the magnetic poles of C-phase winding. While tooth 0 and 3 generate staggered teeth with A- and B-phase poles, and tooth 2 and 5 generate staggered teeth with the magnetic poles of A- and D-phase poles. The similar situation goes on and on. Energize the A, B, C and D phases in turn, and the rotor will rotate in the order of A, B, C and D.



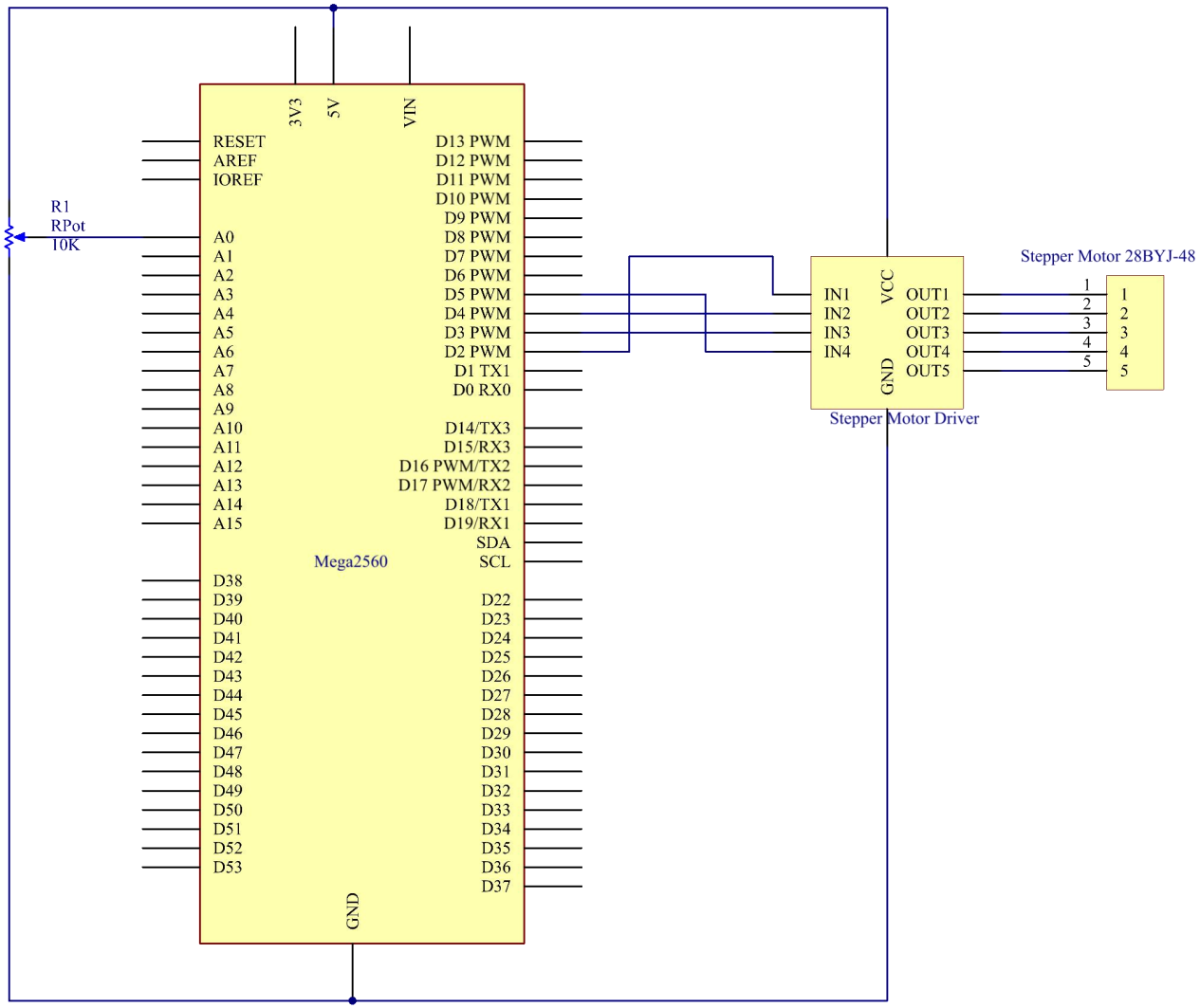
The four-phase stepper motor has three operating modes: single four-step, double four-step, and eight-step. The step angle for the single four-step and double four-step are the same, but the driving torque for the single four-step is smaller. The step angle of the eight-step is half that of the single four-step and double four-step. Thus, the eight-step

operating mode can keep high driving torque and improve control accuracy. In this experiment, we let the stepper motor work in the eight-step mode.

To apply the motor in the circuit, a driver board needs to be used. Stepper Motor Driver-ULN2003 is a 7-channel inverter circuit. That is, when the input end is at high level, the output end of ULN2003 is at low level, and vice versa. If we supply high level to IN1, and low level to IN2, IN3 and IN4, then the output end OUT1 is at low level, and all the other output ends are at high level. So D1 lights up, switch SA is power on, and the stepper motor rotates one step. The similar case repeats on and on. Therefore, just give the stepper motor a specific timing sequence, it will rotate step by step. The ULN2003 here is used to provide particular timing sequences for the stepper motor.



The schematic diagram of the Stepper Motor Driver:

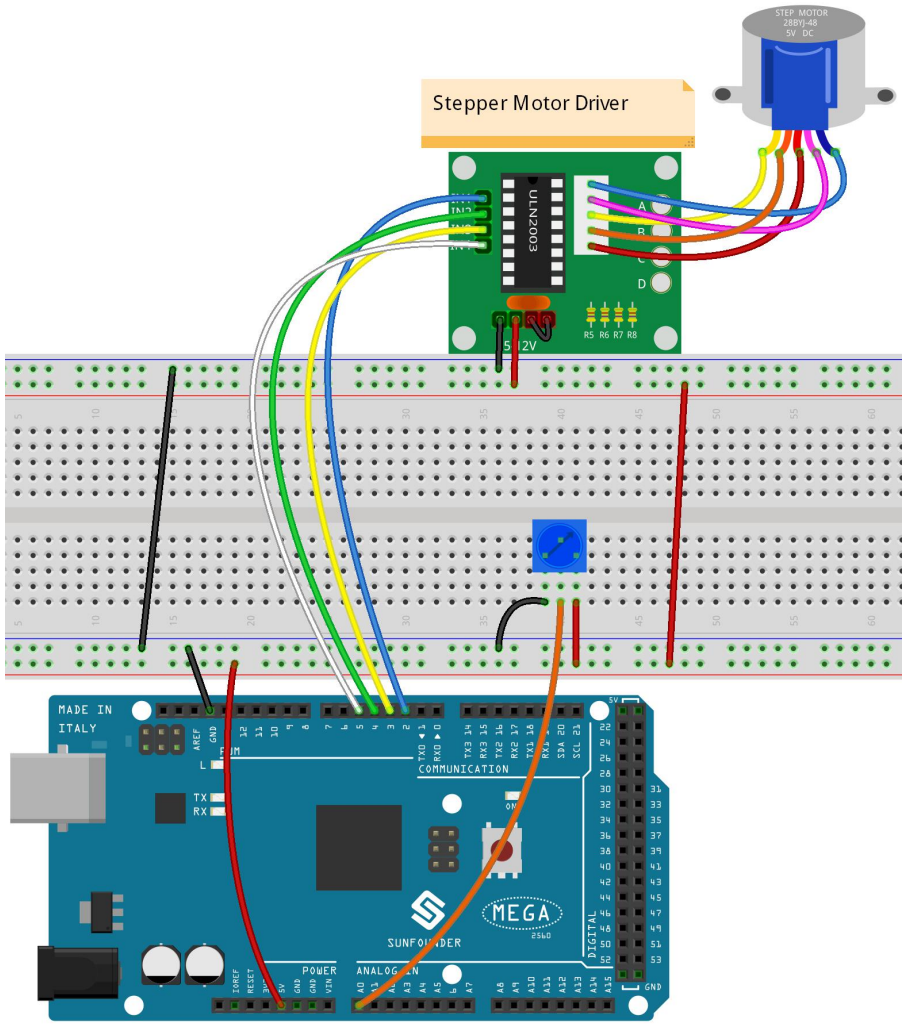


Experimental Procedures

Step 1: Build the circuit

The wiring between Stepper Motor Driver board and Mega 2560 board:

Stepper Motor Driver	Mega 2560
IN1	2
IN2	4
IN3	3
IN4	5
GND	GND
VCC	5v

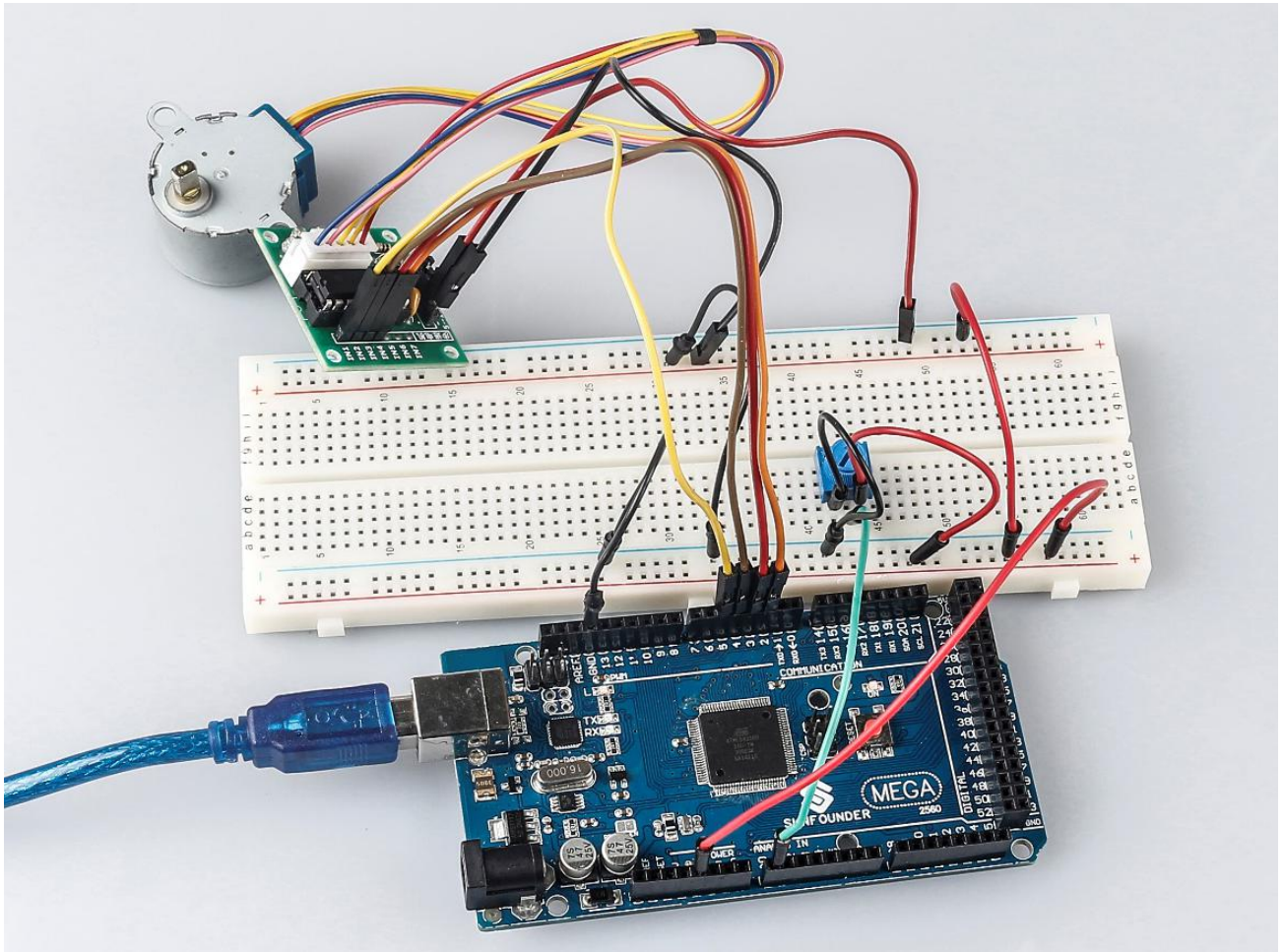


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you should see the rocker arm of the stepper motor spin clockwise and counterclockwise alternately.



Code Analysis

Code Analysis 21-1 Initialize the stepper

```
#include <Stepper.h>  //include a head file
//the steps of a circle
#define STEPS 100

//set steps and the connection with MCU
Stepper stepper(STEPS, 2, 3, 4, 5);

//available to store previous value
int previous = 0;
```

Include a head file Stepper.h, set the steps to 100 and then initialize the stepper with a function stepper().

Stepper(steps, pin1, pin2, pin3, pin4): This function creates a new instance of the Stepper class that represents a particular stepper motor attached to your Arduino board.

steps: The number of steps in one revolution of your motor. If your motor gives the number of degrees per step, divide that number into 360 to get the number of steps (e.g. 360 / 3.6 gives 100 steps). (*int*).

Code Analysis 21-2 setSpeed() function

```
//speed of 180 per minute
stepper.setSpeed(180);  //set the motor speed in rotations per minute(RPMs)
```

setSpeed(rpms): Sets the motor speed in rotations per minute (RPMs). This function doesn't make the motor turn, just sets the speed at which it will when you call step().

Parameters

rpms: the speed at which the motor should turn in rotations per minute - a positive number (long)

Code Analysis 21-3 setSpeed() function

```
void loop()

{ //get analog value

  int val = analogRead(0); //Read the value of the potentiometer

  //current reading minus the reading of history

  stepper.step(val - previous);  //Turn the motor in val-previous steps

  //store as prevous value

  previous = val;  //the value of potentiometer assignment to variable previous

}
```

step(steps): Turns the motor a specific number of steps, at a speed determined by the most recent call to setSpeed(). This function is blocking; that is, it will wait until the motor has finished moving to pass control to the next line in your sketch. For example, if you set the speed to, say, 1 RPM and called step(100) on a 100-step motor, this function would take a full minute to run. For better control, keep the speed high and only go a few steps with each call to step().

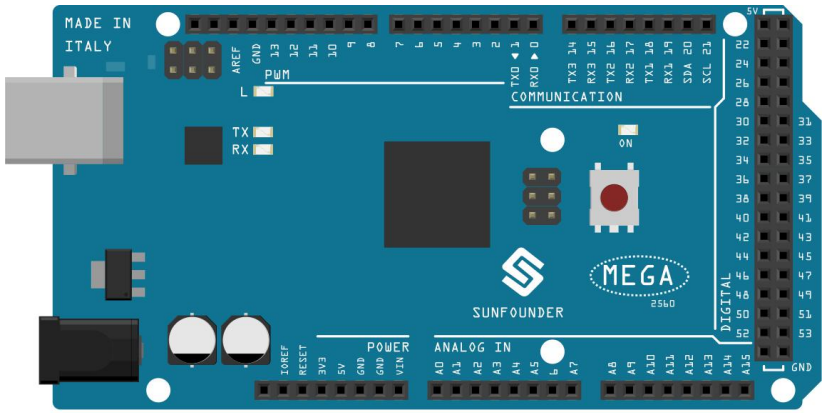
steps: the number of steps to turn the motor - positive to turn one direction, negative to turn the other (int).

Lesson 22 Simple Creation-Stopwatch

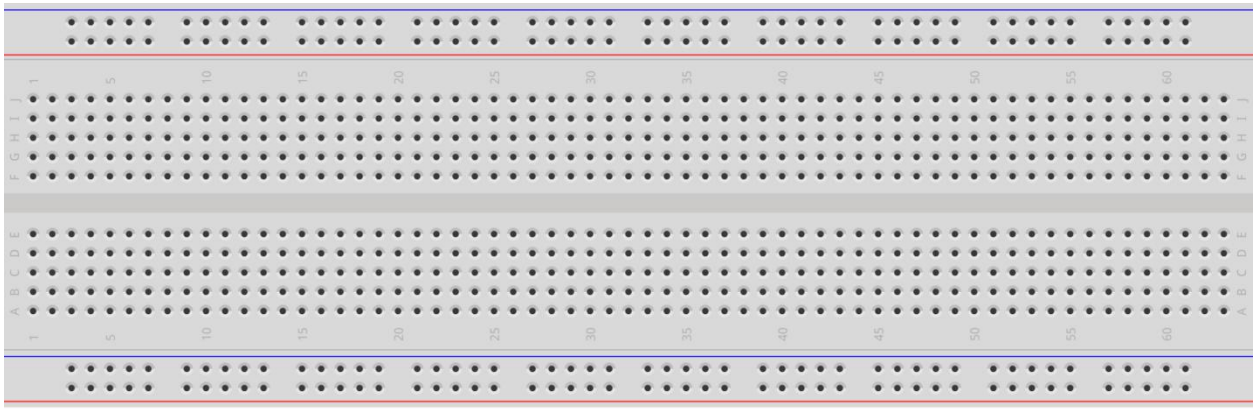
Introduction

In this lesson, we will use a 4-digit 7-segment display to make a stopwatch.

Components

1 * Mega 2560 Board	8 * Resistor (220Ω)	1 * 4-Digit 7-Segment Display
		

1 * Breadboard



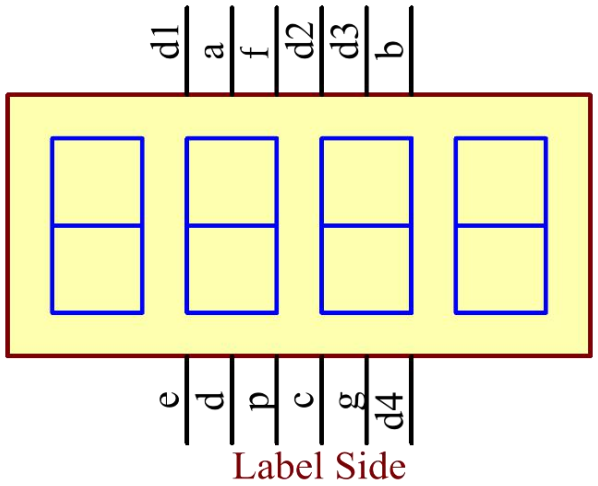
1 * USB cable

Several jumper wires

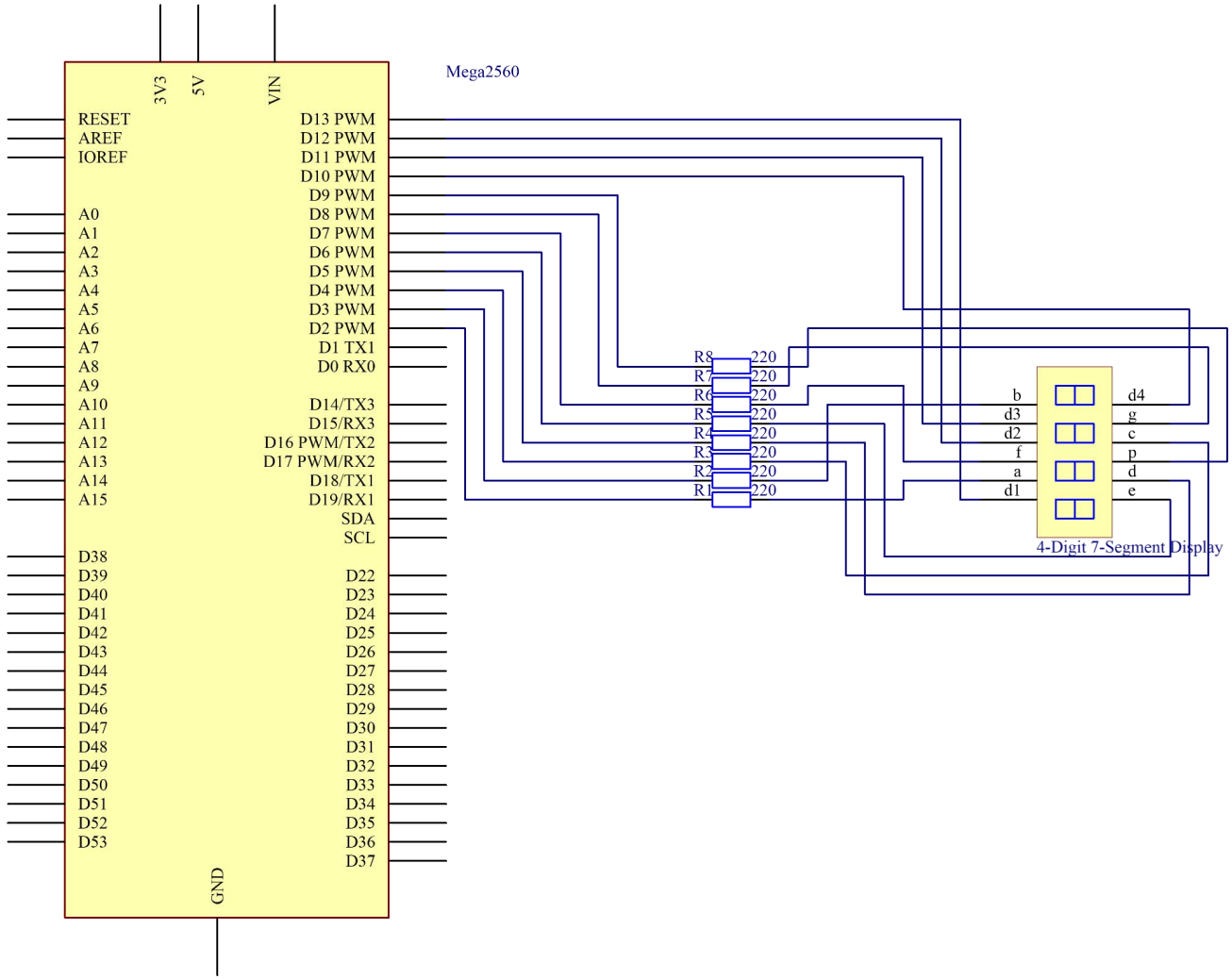
Experimental Principle

When a 7-segment display is used, if it is a common anode display, connect the anode pin to power source; if it is a common cathode one, connect the cathode pin to GND. When a 4-digit 7-segment display is used, the common anode or common cathode pin is to control the digit displayed. There is only one digit working. However, based on the principle of Persistence of Vision, we can see four 7-segment displays all displaying numbers. This is because the electronic scanning speed is too fast for us to notice interval.

The schematic diagram of the 4-digit 7-segment display is as shown below:



The schematic diagram:

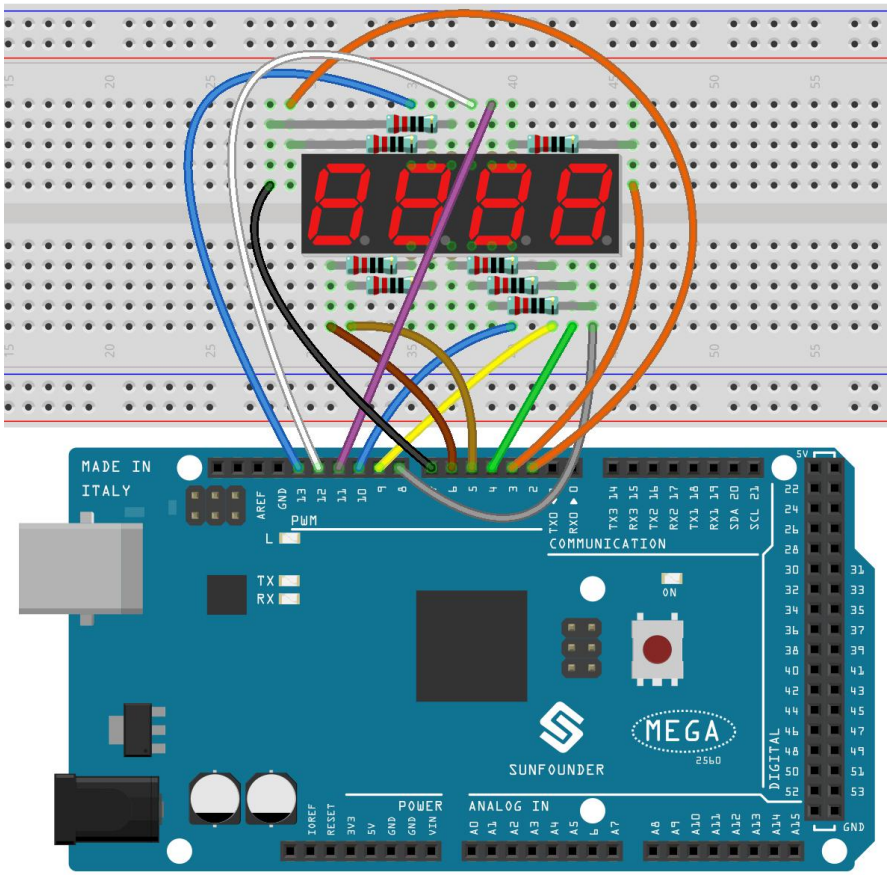


Experimental Procedures

Step 1: Build the circuit

The wiring between the 4-digit 7-segment display and the Mega 2560 Board board is as shown below:

4-Digit 7-Segment Display	Mega 2560 Board
a	2
b	3
c	4
d	5
e	6
f	7
g	8
p	9
D1	13
D2	12
D3	11
D4	10

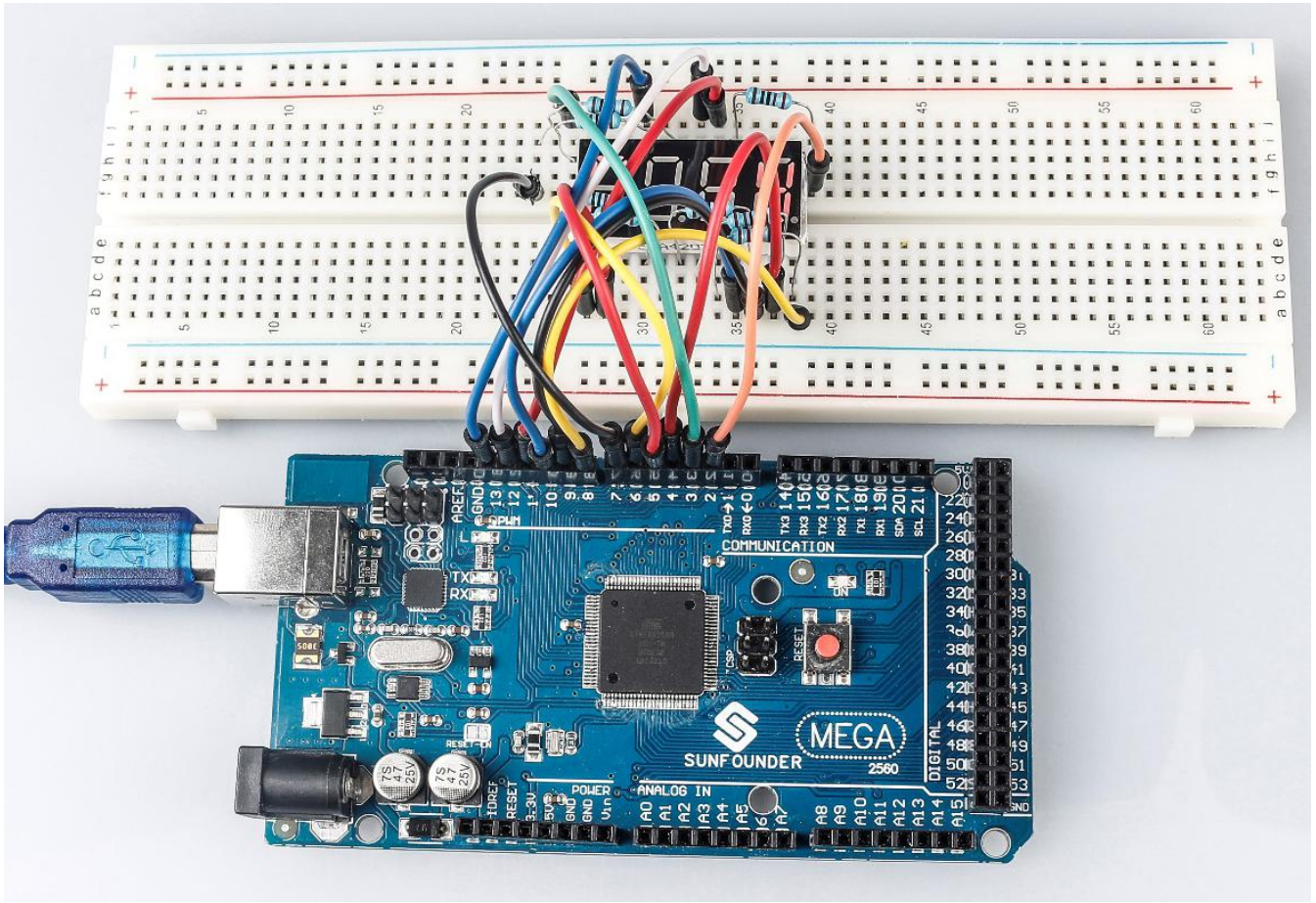


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you can see the number increases by one per second on the 4-digit 7-segment display.



Code Analysis

That's all for the code. It is long enough, so let me sum it up.

Setup: Set all the pins of the LED display as output. Set Timer1 as 0.1 second. Run the following functions. So add() will be called when it's 0.1 second; but before 0.1 second passes, add() is not called yet. Then use a loop() function. The 4 LEDs are displayed as 0000. Wait for a while. 0.1 second later, indicating count=10, call the function add(). then n++=1; because 1<10000, it will not restore to 0. Run loop() and the LEDs will be displayed as 0001. 0.1 second later, n increases by 1, n++=2, and the display will become 0002, and then 0003, and on and on, till 9999. n increases by 1 every second, and the number displayed increases accordingly, until n=10000 and n is 0 again. Then the counting starts from 0.

Code Analysis 22-1 Initialize the timer

```
Timer1.initialize(100000); // set a timer of length 100000 microseconds (or 0.1 sec - or
10Hz => the led will blink 5 times, 5 cycles of on-and-off, per second)
Timer1.attachInterrupt( add ); // attach the service routine here
```

The sentence attachInterrupt(ISR) is to attach an ISR function to call when there is an interrupt. ISR stands for interrupt service routine. Here we use an add routine.

Code Analysis 22-2 Loop function

```
void loop()
{
  clearLEDs();//clear the 7-segment display screen
  pickDigit(0);//Light up 7-segment display d1
  pickNumber((n/1000));// get the value of thousand
  delay(del);//delay 5ms

  clearLEDs();//clear the  7-segment display screen
```

```
  pickDigit(1);//Light up 7-segment display d2
  pickNumber((n%1000)/100);// get the value of hundred
  delay(del);//delay 5ms

  clearLEDs();//clear the  7-segment display screen
  pickDigit(2);//Light up 7-segment display d3
  pickNumber(n%100/10);//get the value of ten
  delay(del);//delay 5ms

  clearLEDs();//clear the 7-segment display screen
  pickDigit(3);//Light up 7-segment display d4
  pickNumber(n%10);//Get the value of single digit
  delay(del);//delay 5ms
}
```

The loop function is used to let four segment display to display the single digit, ten, one hundred and thousand of a value.

Such as n=1345, (1345/1000)=1, (1345%1000)/100)=3, ((1345%100)/10)=4, (n%10)=5.

Code Analysis 22-3 pickDigit(int x) function

```
void pickDigit(int x)  //light up a 7-segment display
{
  //The 7-segment LED display is a common-cathode one. So also use digitalWrite to set d1
as high and the LED will go out
  digitalWrite(d1, HIGH);
  digitalWrite(d2, HIGH);
  digitalWrite(d3, HIGH);
```

```
  digitalWrite(d4, HIGH);

  switch(x)
  {
    case 0:
    digitalWrite(d1, LOW);//Light d1 up
    break;
    case 1:
    digitalWrite(d2, LOW); //Light d2 up
    break;
    case 2:
    digitalWrite(d3, LOW); //Light d3 up
    break;
  default:
    digitalWrite(d4, LOW); //Light d4 up
    break;
  }
}
```

The 4 digital 7 segment is a common cathode one, set all the d1,d2,d3,d4 to HIGH to let them go out.
If x is equals to 0, then run case0 let d1 to LOW level to light first 7 segment up.

switch...case: Like if statements, switch case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.
The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or

the end of the switch statement is reached.

Code Analysis 22-4 pickNumber(int x) function

```
switch(x)
  {
    default:
    zero();
    break;
    case 1:
    one();
    break;
    case 2:
    two();
    break;
    case 3:
   three();
    .
    .
    .
```

The function is to control the LED to display numbers. Call zero(), one() until the nine() function to display 0-9 numbers.

Use zero() as an example:

The function void zero is to control the high/low level of LED. Use digitalWrite to set a to f as high, g as low. Based on the pin diagram just mentioned, when a to f is high and g is low, the number 0 will be displayed.

```
void zero() //the  7-segment led display 0
{
  digitalWrite(a, HIGH);
```

```
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, HIGH);
  digitalWrite(g, LOW);
}
```

Code Analysis 23-5 clearLEDs() function

```
void clearLEDs() //clear the    7-segment display screen
{
  digitalWrite(a, LOW);
  digitalWrite(b, LOW);
  digitalWrite(c, LOW);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);

}
```

Write all pins a-p to LOW level, let the 7-segment digital display go out.

Code Analysis 22-6 add() function

```
void add()
{
    // Toggle LED
    count ++;        //The original value of count is 0. count++=1; keep the counting till
10, because one LED can display a maximum of 9.
```

```
if(count == 10)     // If count=10, which is 1 second, the following statement will be run.
      {
        count = 0;   //which means count from 0
        n ++;     //then n++=1
  if(n == 10000)   //When n=10000,
        {
          n = 0;    //n restores to 0.
        }
      }
  }
```

Lesson 23 Simple Creation-Answer Machine

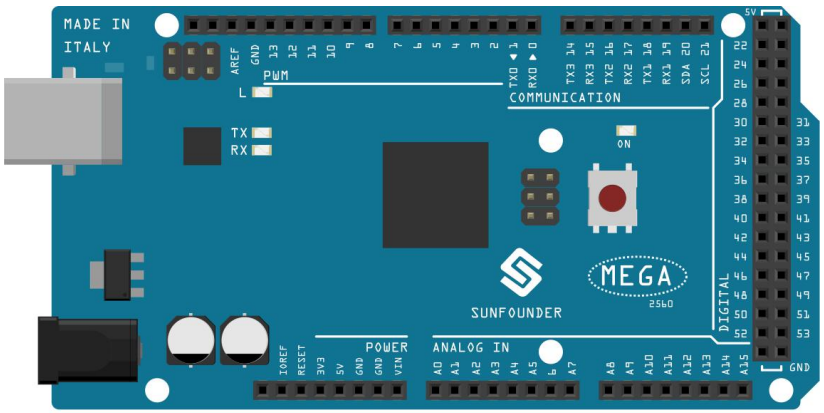
Introduction

In quiz shows, especially entertainment activities (e.g. competitive answering activities), organizers often apply a buzzer system in order to accurately, fairly and visually determine the seat number of a responder.

Now the system can illustrate the accuracy and equity of the judgment by data, which improves the entertainment. At the same time, it is more fair and just. In this lesson, we will use some buttons, buzzers, and LEDs to make a quiz buzzer system.

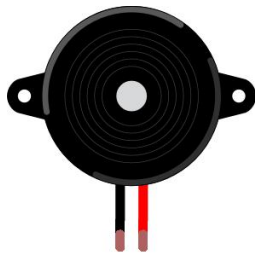
Components

1 * Mega 2560 Board

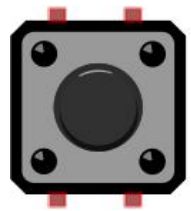


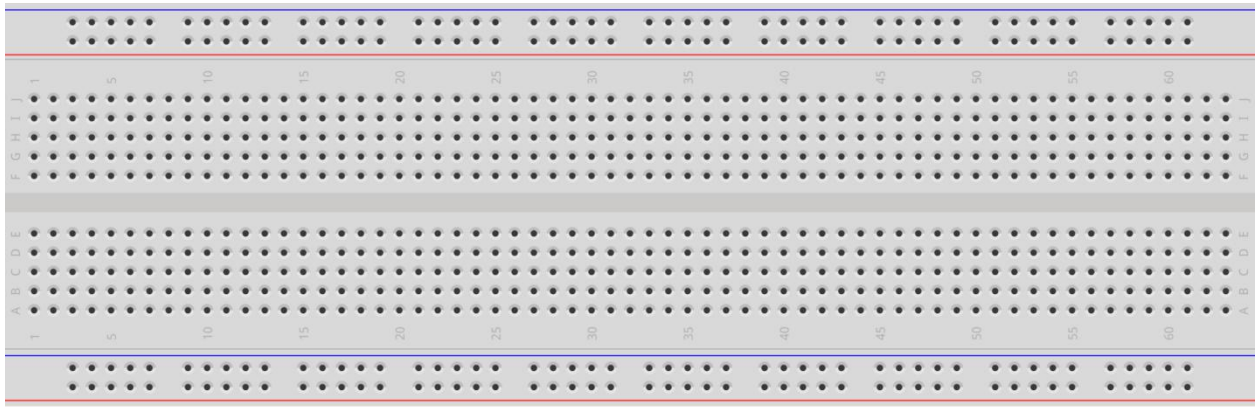
4 * Resistor (220Ω)

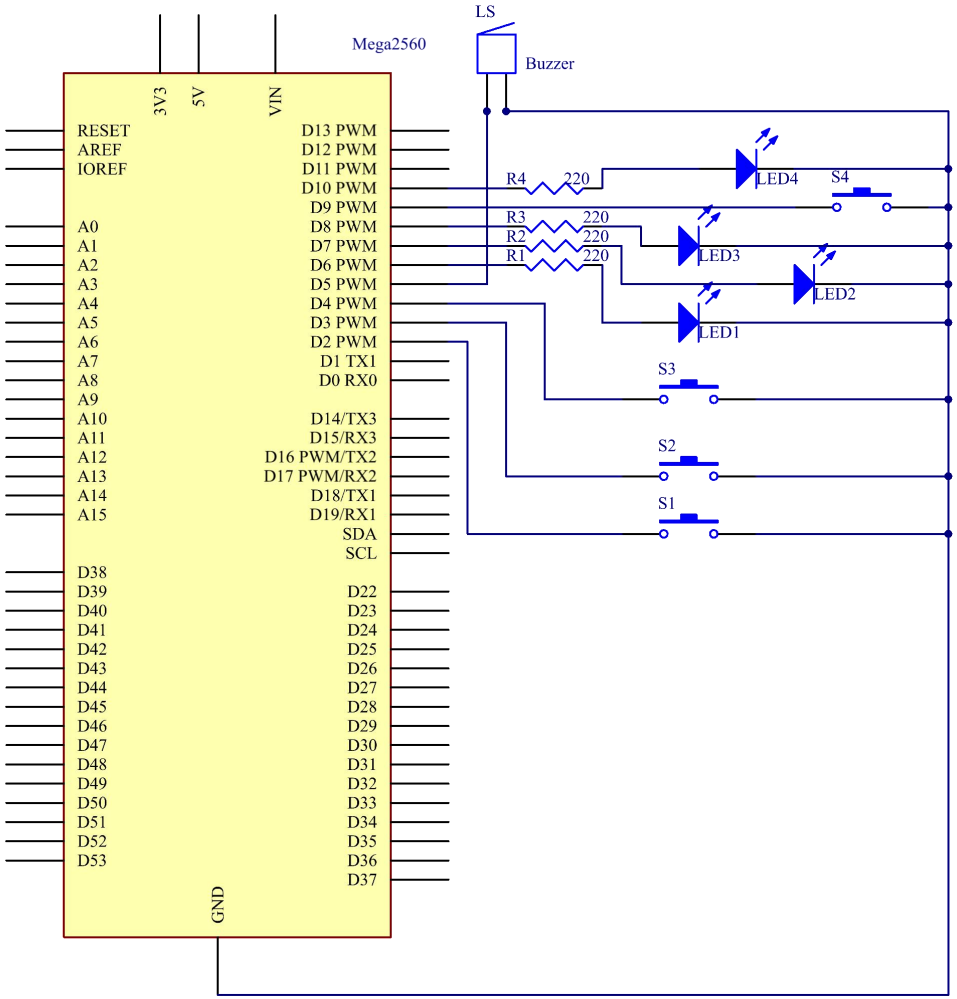
1 * Active Buzzer



4 * Button



1 * Red LED 	1 * Yellow LED 	1 * Green LED 	1 * Blue LED 
1 * Breadboard 			
1 * USB cable 		Several jumper wires 	



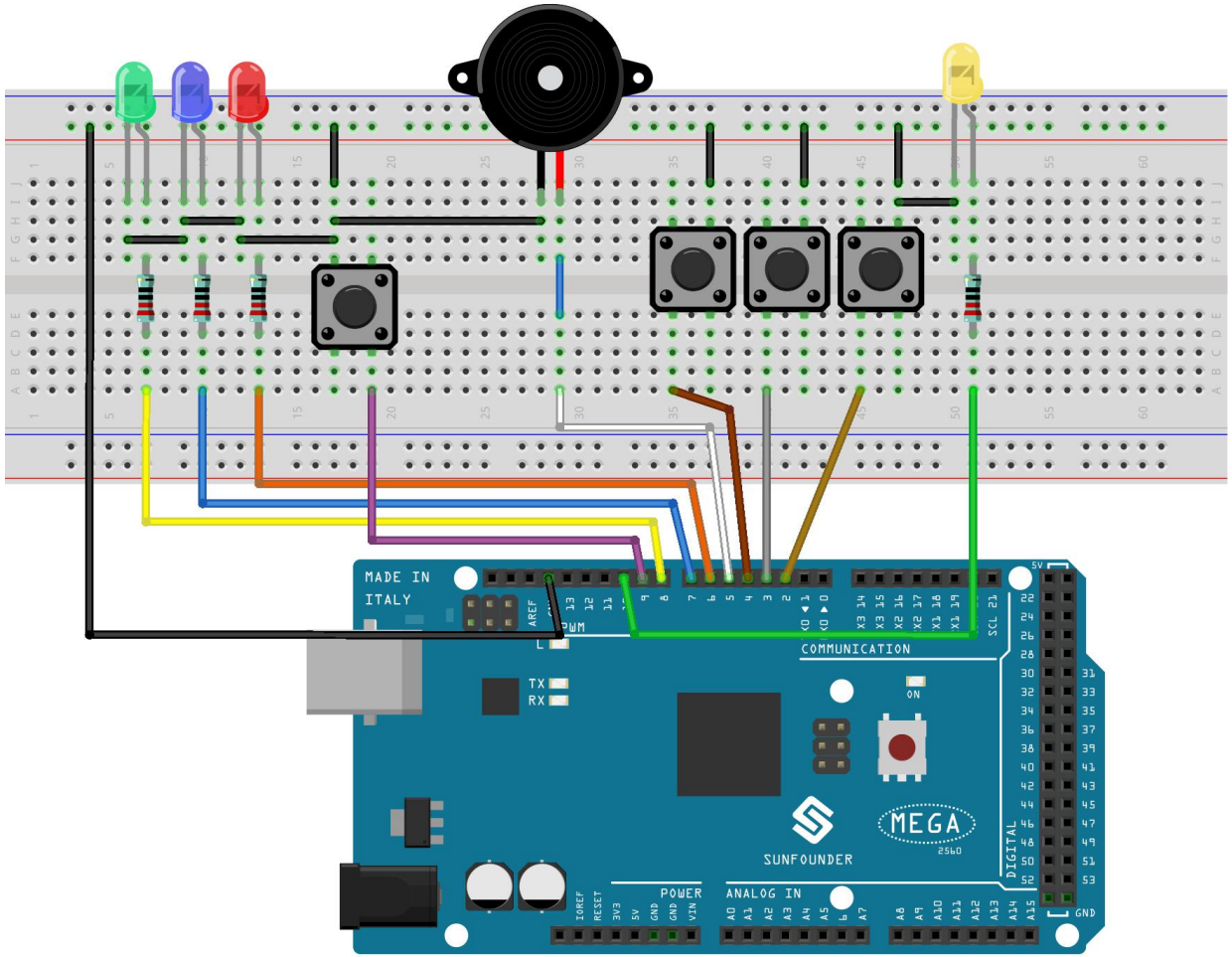
Experimental Principle

Button 1, 2 and 3 are answer buttons, and button 4 is the reset button. If button 1 is pressed first, the buzzer will beep, the corresponding LED will light up and all the other LEDs will go out. If you want to start another round, press button 4 to reset.

The schematic diagram：

Experimental Procedures

Step 1: Build the circuit

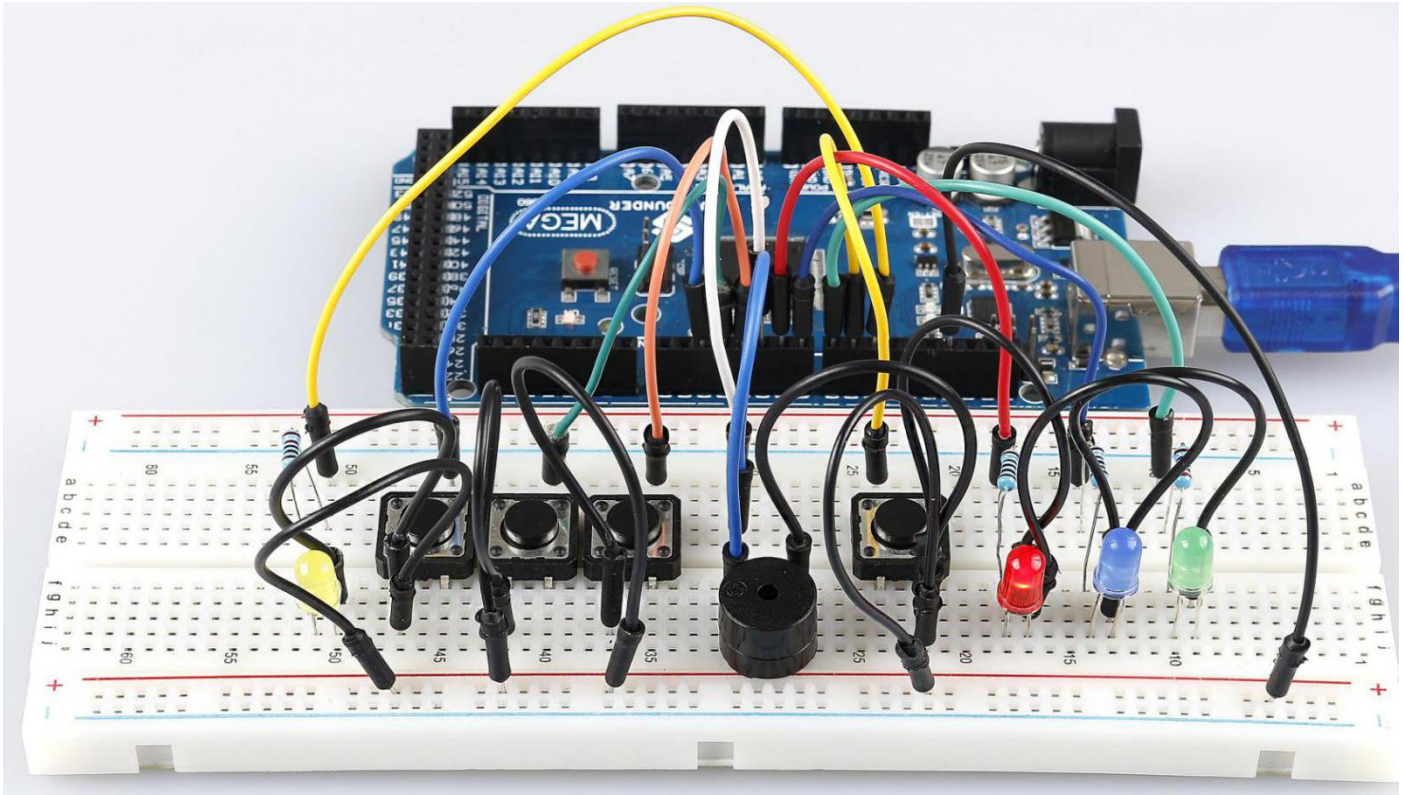


Step 2: Open the code file.

Step 3: Select the Board and Port.

Step 4: Upload the sketch to the board.

Now, first press button 4 to start. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.



Code Analysis

The code for this experiment may be a bit long. But the syntax is simple. Let's see.

Workflow: Read the state of button 4, if button 4 is pressed, the LED on pin 10 is illuminated while reading the state of the remaining buttons. If one of the buttons is detected to be pressed, the buzzer beeps and lights the corresponding LED until button 4 is pressed again.

Code Analysis 23-1 loop() function

```
b4State = digitalRead(button4);
  //when button4 pressed
  if(b4State == 0)
  {
    if(b4State == 0)  //confirm that the button4 is pressed. One pin of the button is connected to pin 9, the other pin is connected to GND, and when the button is pressed, pin 9 is pulled low.
    {
      flag = 1;  //if so,flag is 1
      digitalWrite(LED4, HIGH);  //turn the host LED on
      delay(200);
    }
  }
  if(1 == flag)
```

```
{
  //read the state of the state of buttons
  b1State = digitalRead(button1);
  b2State = digitalRead(button2);
  b3State = digitalRead(button3);
  //If the button1 press the first
  if(b1State == 0)
  {
    flag = 0;
    digitalWrite(LED4, LOW);
    Alarm();  //buzzer sound
    digitalWrite(LED1,HIGH);  //turn the LED1 on only
    digitalWrite(LED2,LOW);
    digitalWrite(LED3,LOW);
    while(digitalRead(button4));  //detect the button4,if pressed,out of the while loop
  }
  .
  .
  .
```

Code Analysis 23-2 Alarm() function

```
void Alarm()
{
  for(int i=0;i<100;i++){
    digitalWrite(buzzerPin,HIGH); //the buzzer sound
    delay(2);
    digitalWrite(buzzerPin,LOW);  //without sound
    delay(2);                     //when delay time changed,the frequency changed
  }
}
```

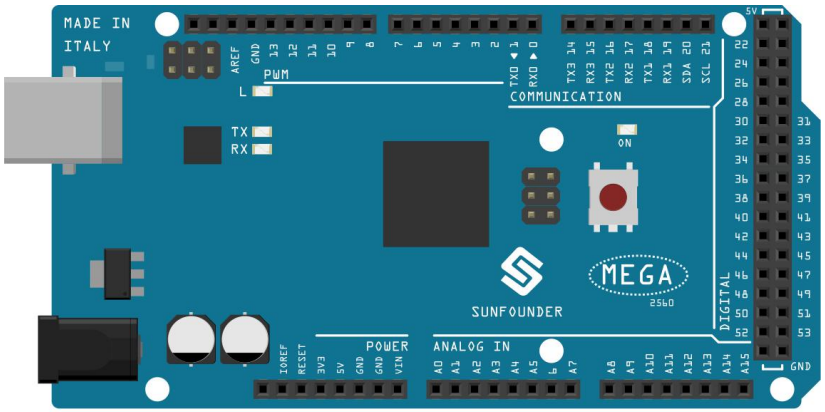
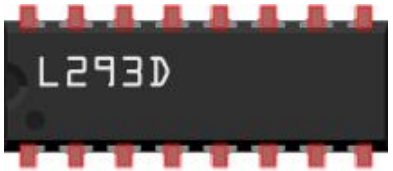
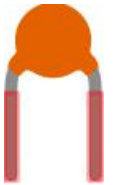
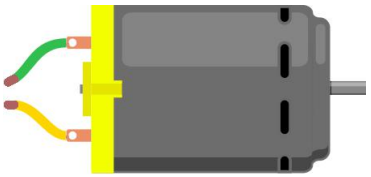
The alarm() function is to set the buzzer to beep.

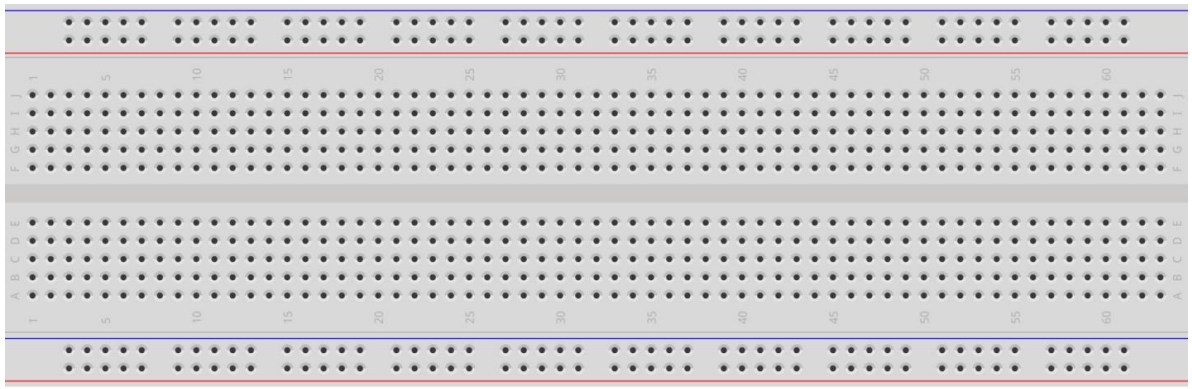
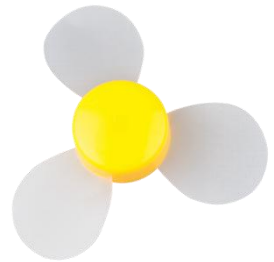
Lesson 24 Simple Creation-Small Fan

Introduction

In this experiment, we will learn how to control the direction and speed of a small-sized DC motor by a driver chip L293D. Making simple experiments, we will just make the motor rotate left and right, and accelerate or decelerate automatically.

Components

1 * Mega 2560 Board 	1 * L293D 	1 * 104 ceramic capacitor 
	1 * Small-sized DC motor 	1 * Button 

1 * Breadboard	1 * Fan
	

1 * USB cable	Several jumper wires
	

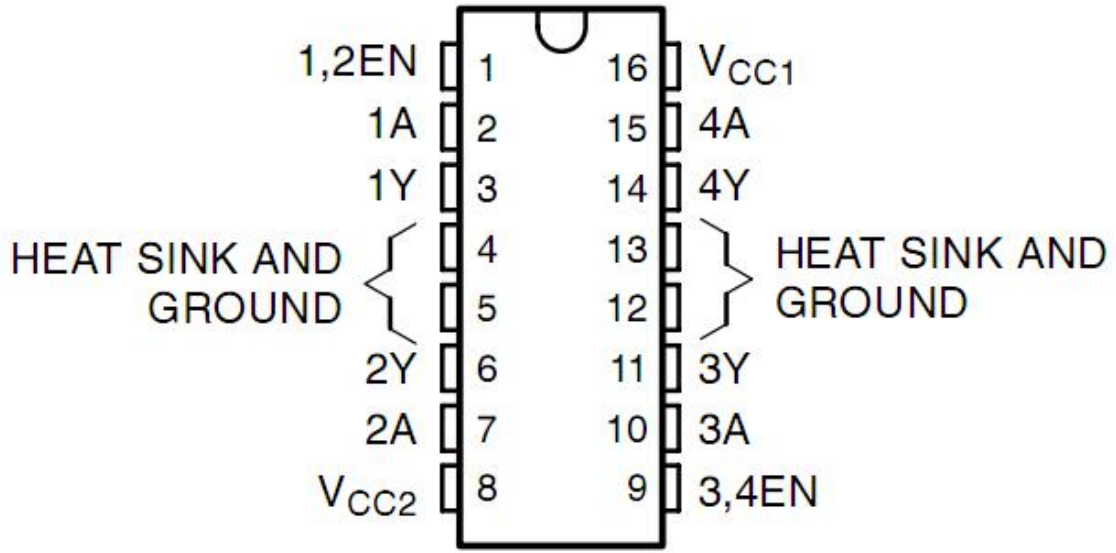
Experimental Principle

The maximum current of an Arduino I/O port is 20mA but the drive current of a motor is at least 70mA. Therefore, we cannot directly use the I/O port to drive the current; instead, we can use an L293D to drive the motor.

L293D

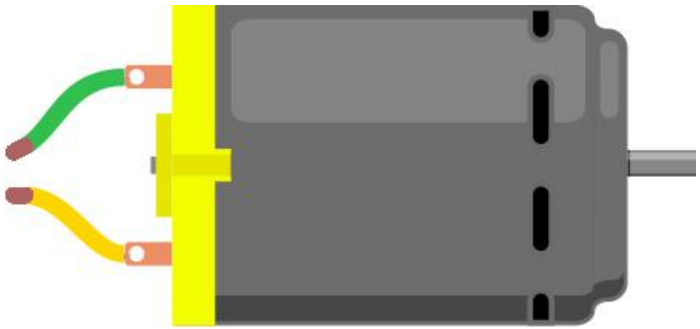
L293D is designed to provide bidirectional drive currents of up to 600mA at voltages from 4.5V to 36V. It's used to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

See the figure of pins below. L293D has two pins (Vcc1 and Vcc2) for power supply. Vcc2 is used to supply power for the motor, while Vcc1, for the chip. Since a small-sized DC motor is used here, connect both pins to +5V. If you use a higher power motor, you need to connect Vcc2 to an external power supply.



Pin EN is an enabling pin and works with High level. A stands for input and Y for output. When pin EN is High level, if A is High, Y outputs High level; if A is Low, Y outputs Low level. When pin EN is Low level, the L293D does not work. It just needs to drive one motor in this experiment, so here use one side of the L293D.

DC Motor



This is a 5V DC motor. Give the two terminals of the copper sheet one high and one low level, and the motor will rotate. For convenient purposes, you can weld the pins to it.

Size: 25*20*15MM

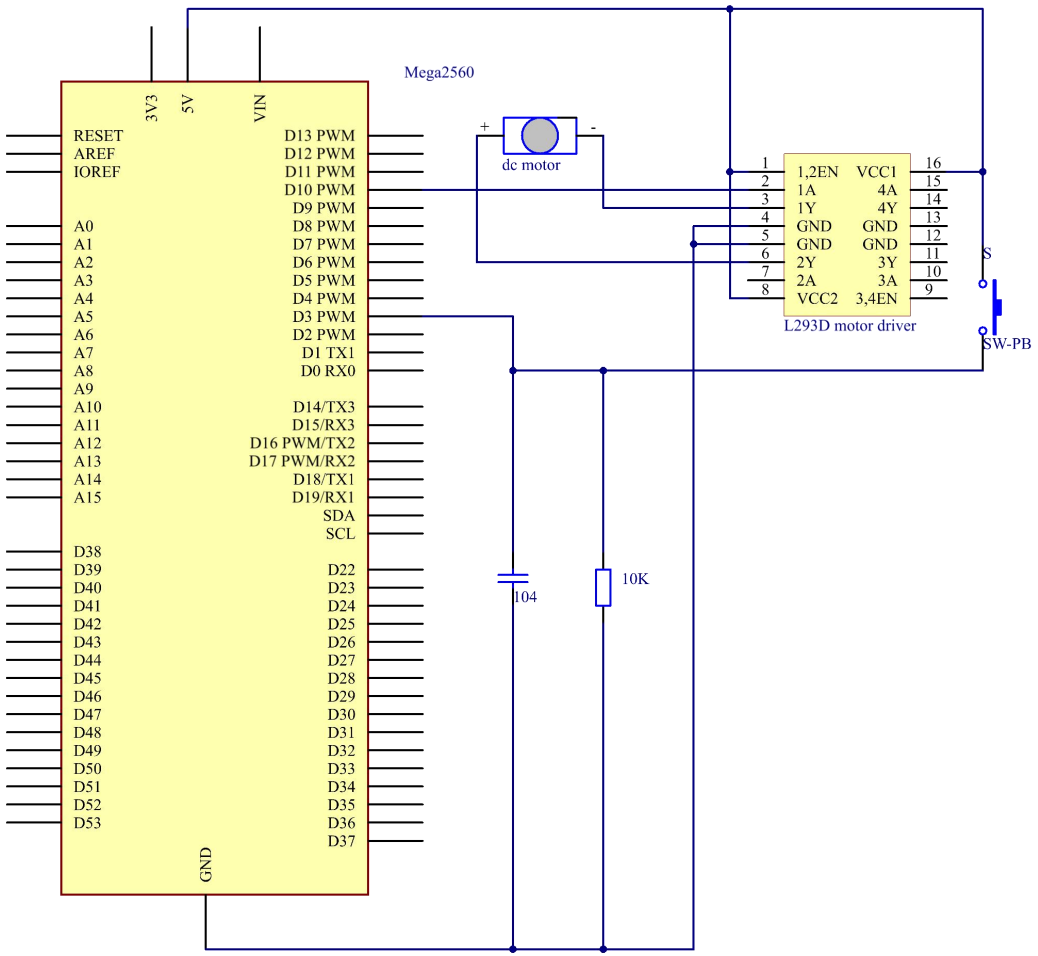
Free-run speed (3V): 13000RPM

Stall current (3V): 800 mA

Shaft diameter: 2 mm

Operation Voltage: 1-6V

Free-run current (3V): 70 mA



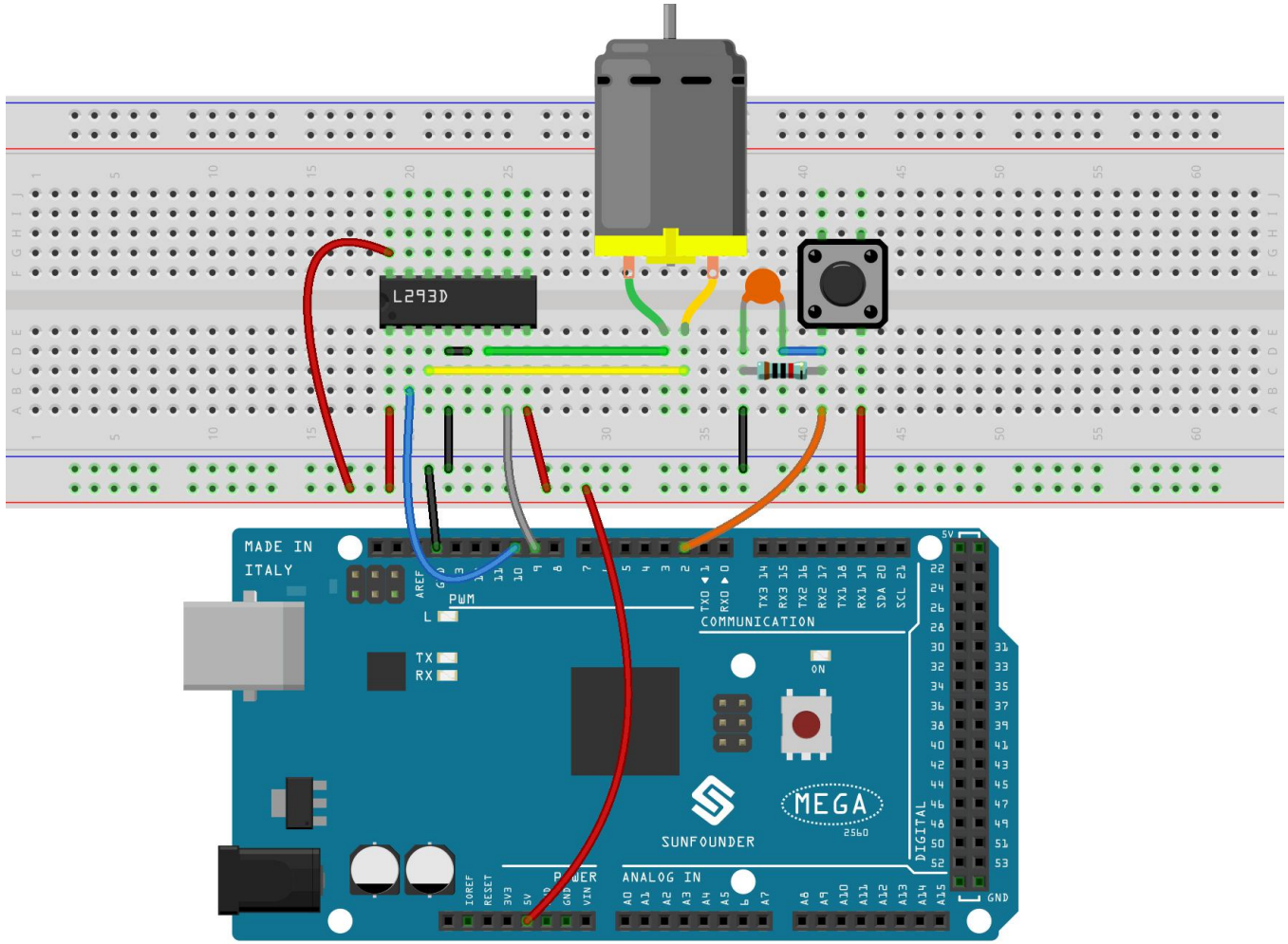
Principle:

The Enable pin 1,2EN of the L293D are connected to 5V already, so L293D is always in the working state. Connect pin 1A and 2A to pin 9 and 10 of the control board respectively. The two pins of the motor are connected to pin 1Y and 2Y respectively. When pin 10 is set as High level and pin 9 as Low, the motor will start to rotate towards one direction. When the pin 10 is Low and pin 9 is High, it rotates in the opposite direction.

The schematic diagram:

Experimental Procedures

Step 1: Build the circuit

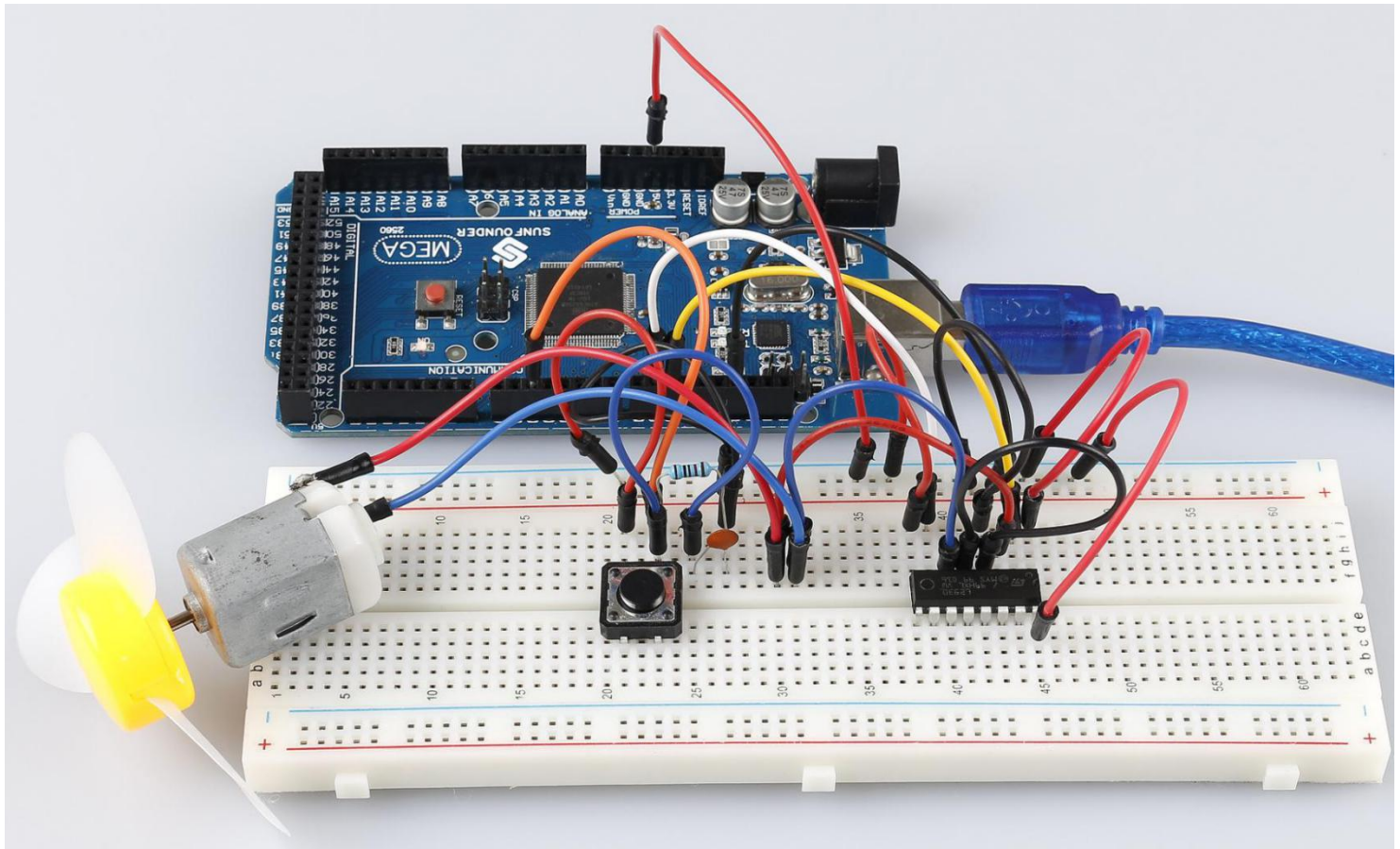


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

The blade of the DC motor will begin rotating left and right, in a speed that varies accordingly.



Code Analysis

Code Analysis 24-1 Workflow of the Small Fan

```
void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState)// If the button state is different from last time
  {
    lastDebounceTime = millis();// reset the debouncing timer
  }
  if ((millis() - lastDebounceTime) > debounceDelay)  // Determine whether the button has been pressed for over 50ms to prevent signal generated due to accidental touch.
{
if (reading != buttonState)// If it's over 50ms and reading does not equal to buttonState, it indicates the button state has changed.
  {
      buttonState = reading; // Store the state of button in buttonState
      if (buttonState == HIGH)// If buttonState is high level, it means the button has been pressed.
    {
```

```
        digitalWrite(ledPin, HIGH); //turn on the LED
        stat = stat + 1;
        if(stat >= 4)// When stat>=4, set it as 0.
        {
          stat = 0;
        }
      }
```

```
    else     //else, turn off the LED. When you press the button, the LED will light up and it goes out when you release the button.
      digitalWrite(ledPin, LOW);
  }
}
```

```
  // The rotational speed is different when the button is pressed at different times.
  switch(stat)
  {
  case 1:
    clockwise(rank1);// When stat=1, set the rotate speed of the motor as rank1=150
    break;
  case 2:
    clockwise(rank2);// When stat=2, set the rotate speed of the motor as rank1=200
    break;
  case 3:
```

```
    clockwise(rank3);// When stat=3, set the rotate speed of the motor as rank1=250
    break;
  default:
    clockwise(0);
  }
  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonState = reading;
```

Code Analysis 24-2 clockwise() function

```
void clockwise(int Speed)
{
  analogWrite(motorIn1,Speed);  //set the speed of motor
  analogWrite(motorIn2,0);  //stop the motorIn2 pin of motor
}
```

This function is to set the rotational speed of the *motor*: write *Speed* to pin 9 and 0 to pin 10. The motor rotates towards a certain direction and the speed is the value of *Speed*.

Experiment Summary

In this experiment, you can also control the motor to rotate or not. Just connect pin 1, 2EN of the L293D to an I/O port of the control board. Set 1, 2EN as High level, and the motor will start rotating; set it as Low level, it will stop the rotating.

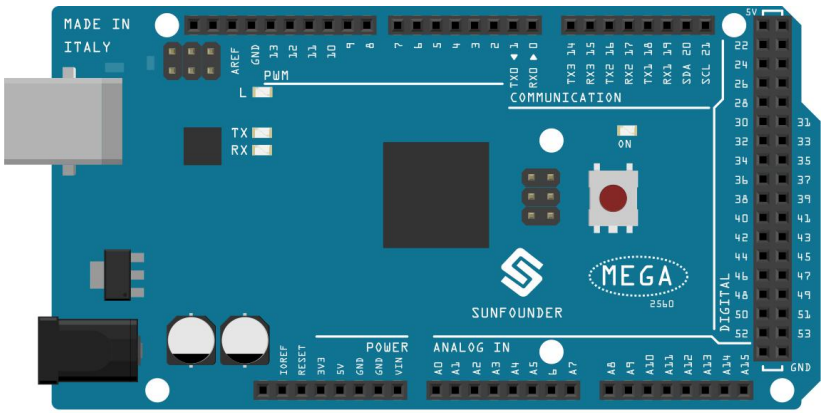
Lesson 25 Simple Creation - Digital Dice

Introduction

In previous experiments, we learned how to use a 7-segment display and control LEDs by a button. In this lesson, we will use a 7-segment display and a button together to create a simple digital dice.

Components

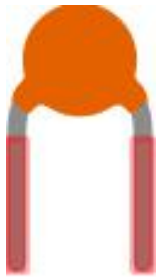
1 * Mega 2560 Board



1 * Resistor 10kΩ)

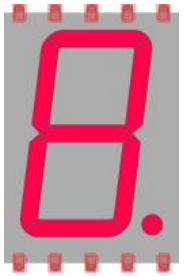
8 * Resistor (220Ω)

1 * 104 ceramic capacitor



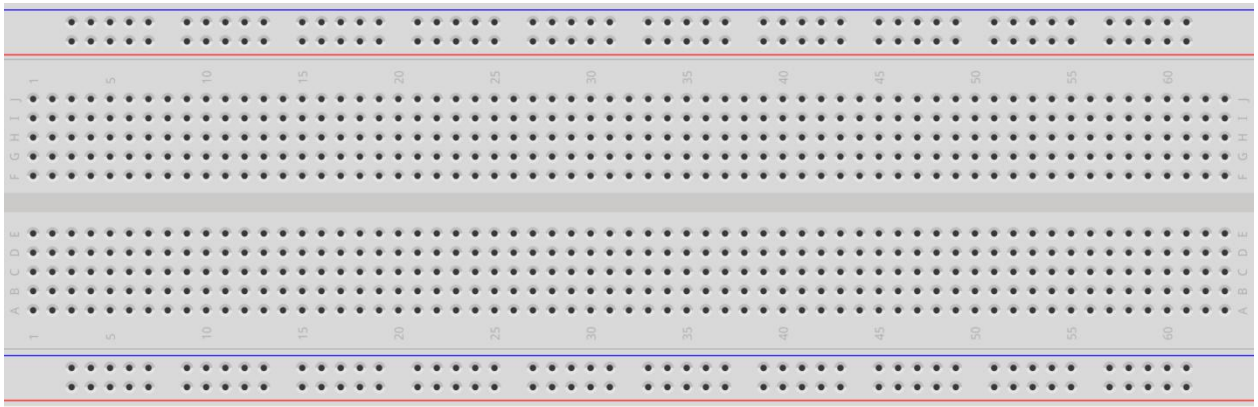
1 * 74HC595

1 * 7-segment display



1 * Button

1 * Breadboard



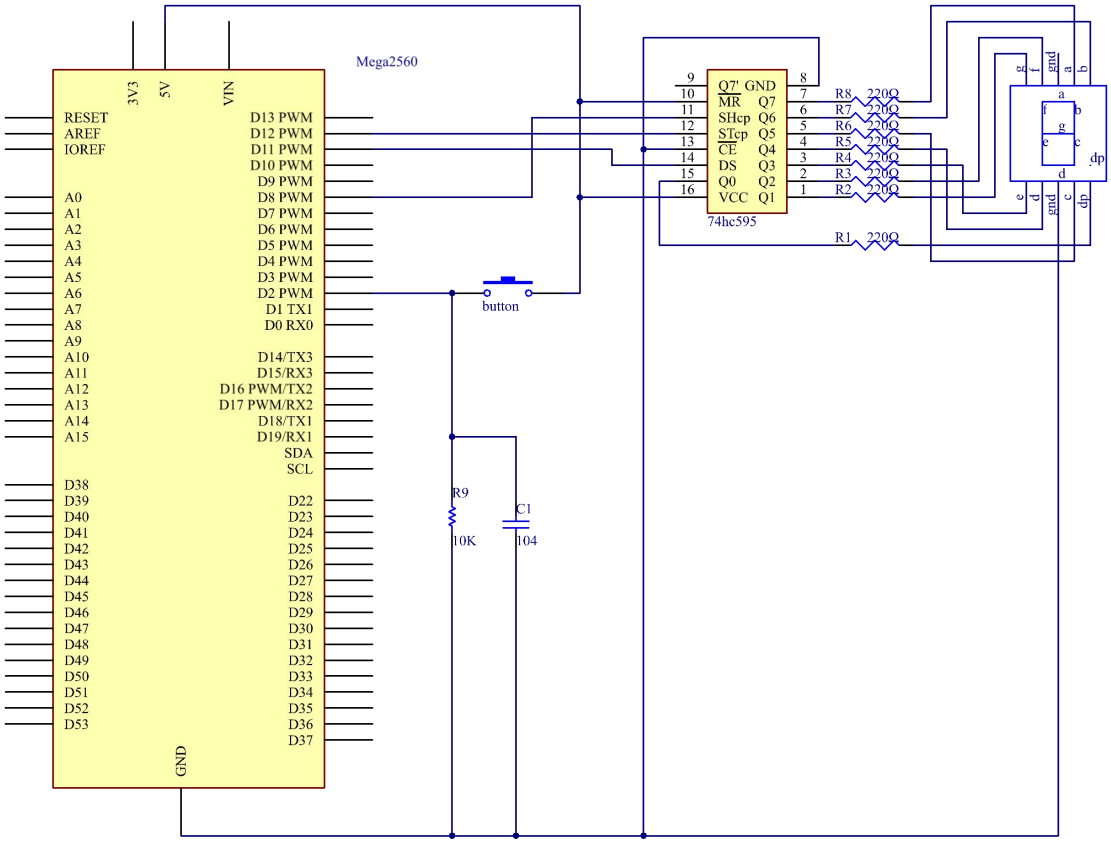
1 * USB cable

Several jumper wires

Experimental Principle

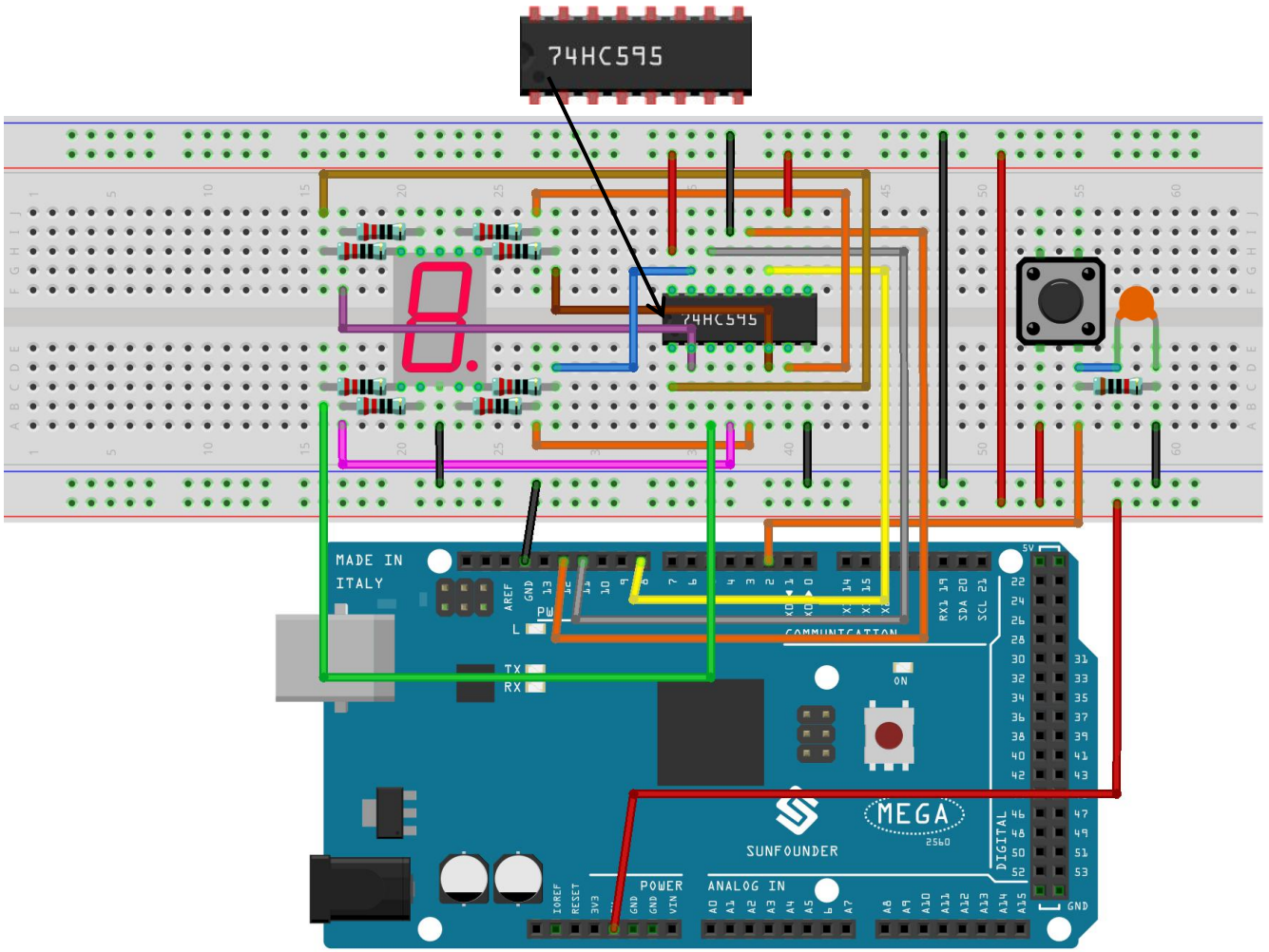
The idea behind a digital dice is very simple: a 7-segment display circularly jumps from 1 to 7 rapidly. When the button is pressed, the jumping will slow down until it stops on a number. When the button is pressed again, the process will repeat.

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit

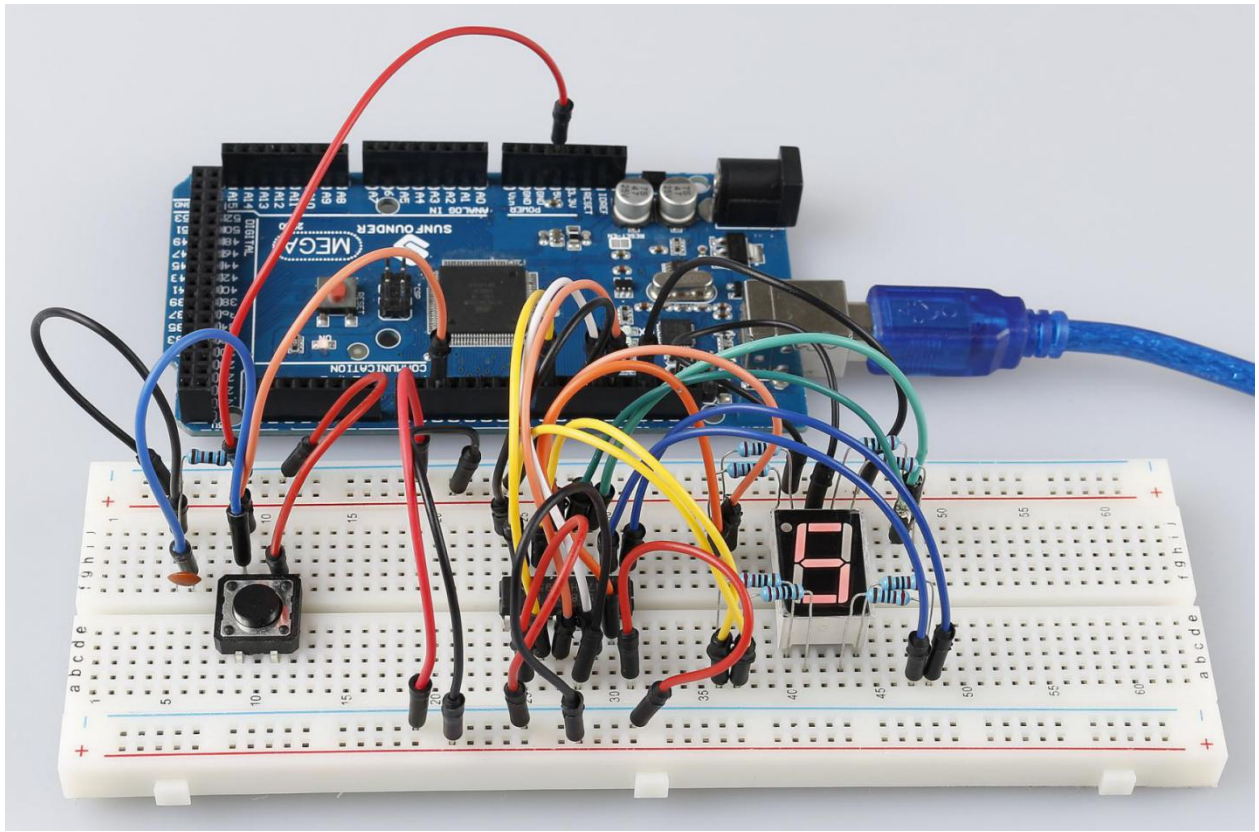


Step 2: Open the code file.

Step 3: Select the Board and Port.

Step 4: Upload the sketch to the board.

You can now see the 7-segment display jump between numbers from 1 to 6. Press the button, and the jumping will slow down until it stops three seconds later. Press the button again, and the process will repeat.



Code Analysis

Code Analysis 25-1 The initial random number comes from A0

```
randomSeed(analogRead(0));
```

The initial random number is generated from A0 and the range for the random numbers is 0-1023.

Code Analysis 25-2 Digital Dice

```
void loop()
{
  int stat = digitalRead(keyIn);  //store value read from keyIn
 if(stat == HIGH)  // check if the pushbutton is pressed  If yes, the corresponding pin is high level
 {
   num ++;  //num adds 1
  if(num > 1)  //If num > 1, clear the value. This is to prevent repeated pressing. So just count it as once no matter how many times you press.
  {
    num = 0;
  }
}
Serial.println(num);  // print the num on serial monitor
  if(num == 1)  //when pushbutton is pressed
  {
```

```
randNumber = random(1,7); //Generate a random number in 1-7
showNum(randNumber);  //show the randNumber on 7-segment
delay(1000);  //wait for 1 second
while(!digitalRead(keyIn));  //When not press button,program stop here  Make it keep displaying the last random number.
 int stat = digitalRead(keyIn);   // Read the state of the button again.
 if(stat == HIGH) // check if the pushbutton is pressed. If yes, run the code below
 {
   num ++; //num+1=2
  pinMode(ledPin,OUTPUT);//initialize the digital pin as an output
   digitalWrite(ledPin,HIGH); //turn on the led
   delay(100);
   digitalWrite(ledPin,LOW); //turn off the led
   delay(100);
       if(num >= 1)  // clear the num
    {
      num = 0;
    }
  }
}
```

```
  //show random numbers at 100 microseconds intervals
   // If the button has not been pressed
  randNumber = random(1,7);
  showNum(randNumber);
  delay(100);
}
```

Code Analysis 25-3 showNum() function

```
void showNum(int num)
{
  digitalWrite(latchPin,LOW); //ground latchPin and hold low for transmitting
  shiftOut(dataPin,clockPin,MSBFIRST,datArray[num]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin,HIGH); //pull the latchPin to save the data
}
```

This function is to display the number in *dataArray[]* on the 7-segment display.

Copyright Notice