Preface

About SunFounder

SunFounder is a company focused on STEAM education with products like open source robots, development boards, STEAM kit, modules, tools and other smart devices distributed globally. In SunFounder, we strive to help elementary and middle school students as well as hobbyists, through STEAM 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 STEAM knowledge.

About this kit

This kit is suitable for SunFounder Uno, SunFounder Mega 2560, SunFounder Duemilanove and SunFounder Nano. All the code in this user guide is compatible with these boards.

With this kit, we will walk you through the know-how of using the Arduino 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.

Warning

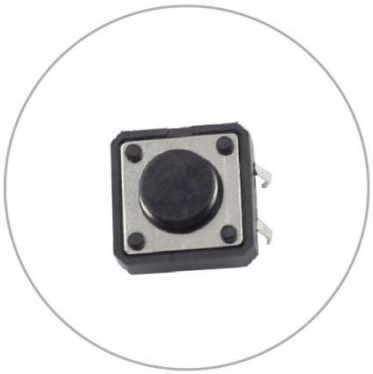
When you purchase or use this product, please note the following:

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

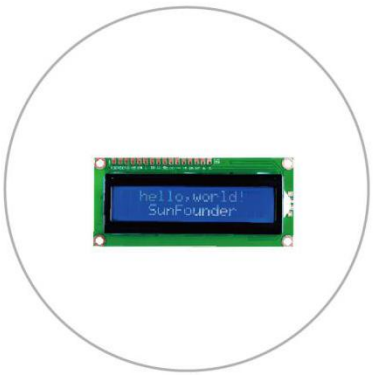
Contents

Components List

7-Segment
1 PCS

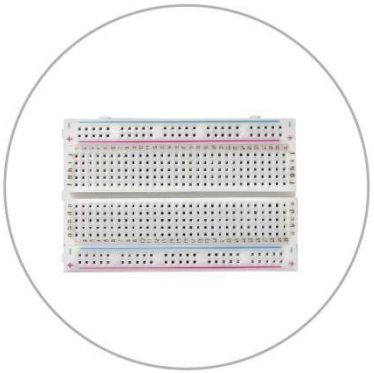


Button
10 PCS



LCD1602
1 PCS

Servo
1 PCS



Breadboard
1 PCS

1M USB cable
1 PCS

Passive buzzer

1 PCS

Relay

1 PCS

Slide Switch

5 PCS

74HC595

1 PCS

4N35

1 PCS

555 Timer

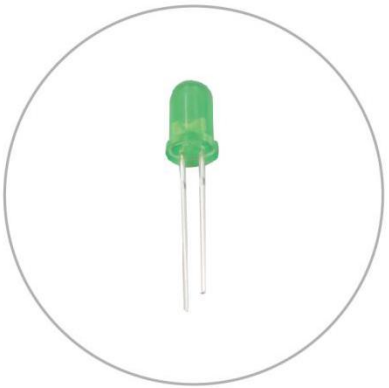
1 PCS

RGB LED

1 PCS

LED (Red)

10 PCS



LED (Green)

10 PCS

LED (Yellow)

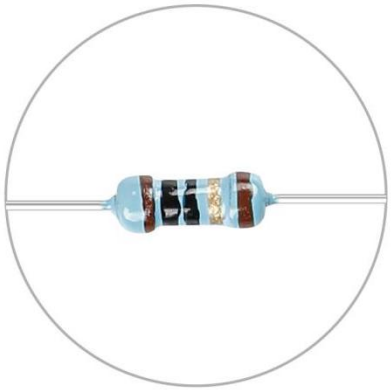
10 PCS

LED (White)

10 PCS

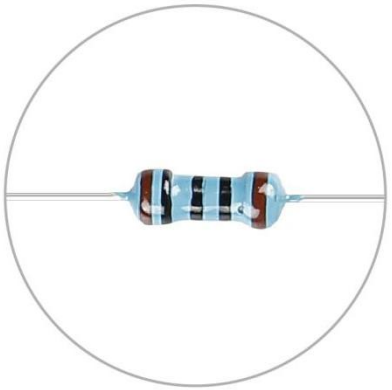
LED(Blue)

10 PCS



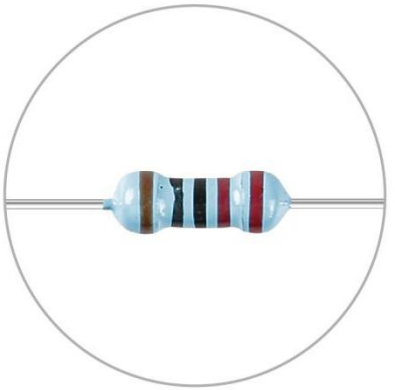
Resistor(10Ω)

10 PCS



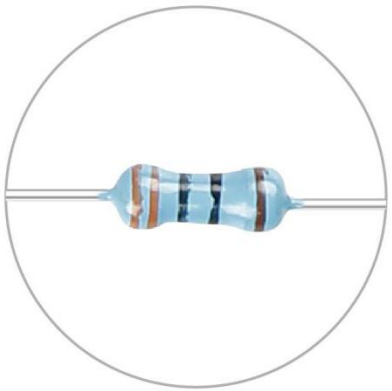
Resistor(100Ω)

10 PCS



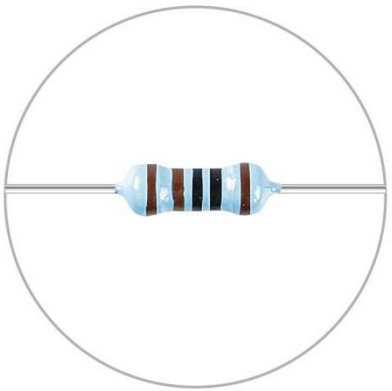
Resistor(220Ω)

10 PCS



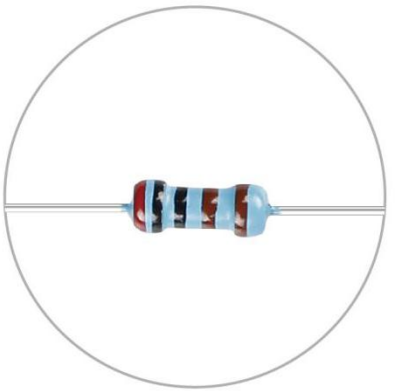
Resistor(330Ω)

10 PCS



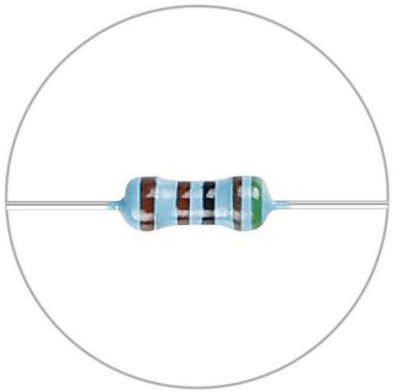
Resistor(1KΩ)

10 PCS



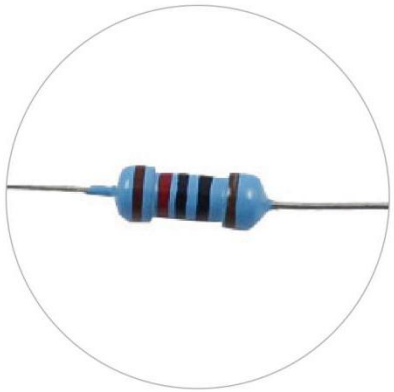
Resistor(2KΩ)

10 PCS



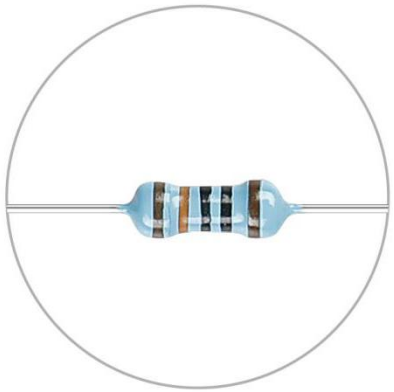
Resistor(5.1KΩ)

10 PCS



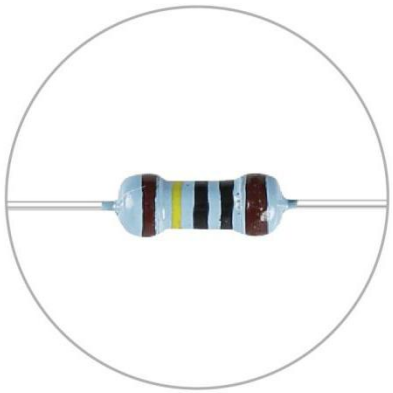
Resistor(10KΩ)

10 PCS



Resistor(100KΩ)

10 PCS



Resistor(1MΩ)

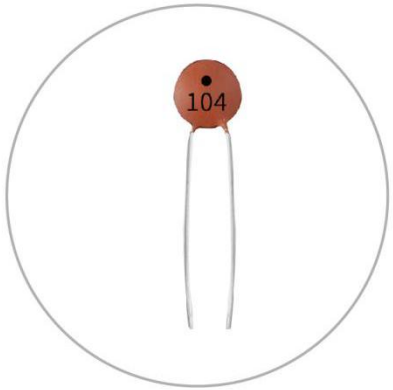
10 PCS

1N4007

5 PCS

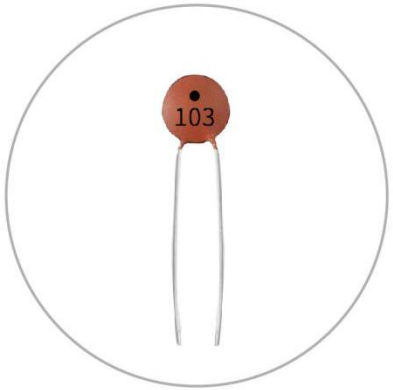
Zener diode

1 PCS



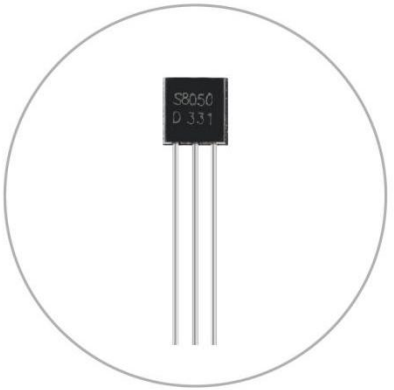
104 Capacitor

10 PCS



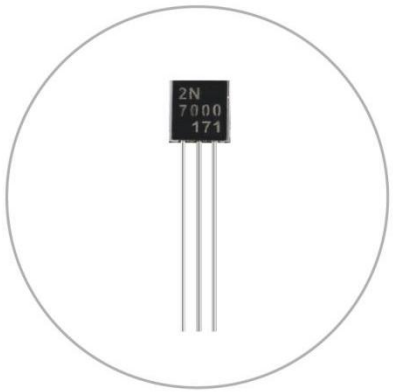
103 Capacitor

10 PCS



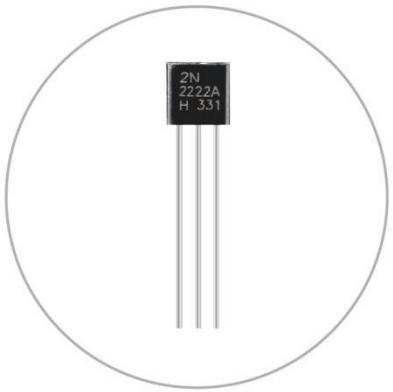
S8050 Transistor

2 PCS



2N7000 Transistor

1 PCS

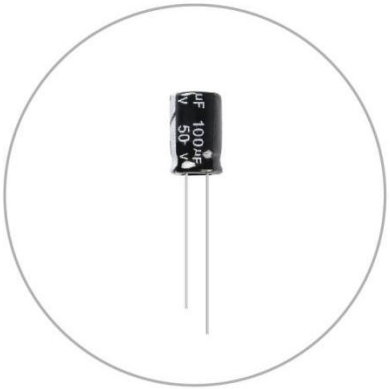


PN222A

5 PCS

Capacitor (10UF)

5 PCS



Capacitor (100UF)

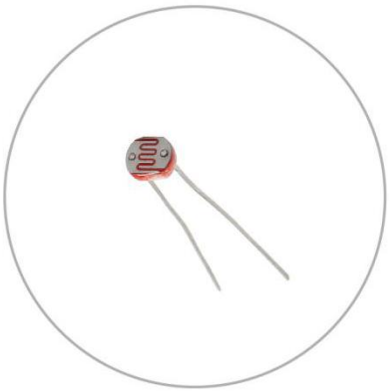
5 PCS

Thermistor

1 PCS

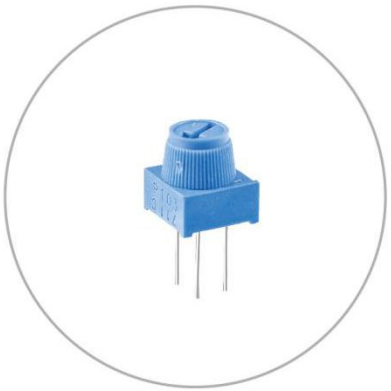
Tilt switch

1 PCS



Photoresistor

2 PCS

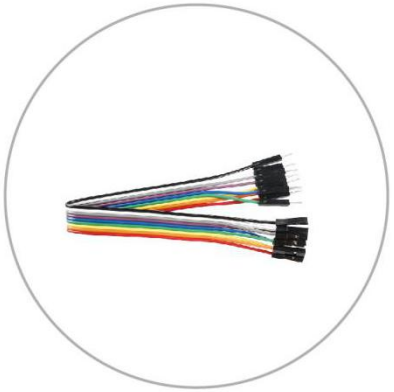


Potentiometer

2 PCS

Active buzzer

1 PCS

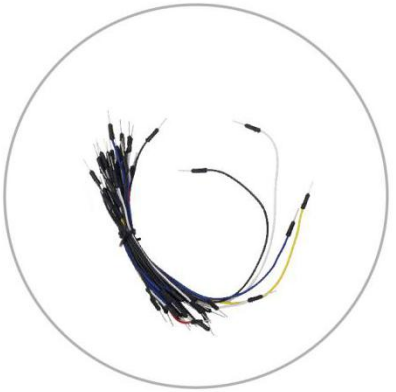


DuPont wires

20 PCS

Battery snap

1 PCS



Jump wires

65 PCS

Note:

After unpacking, please check that the number of components is correct and that all components are in good condition.

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.

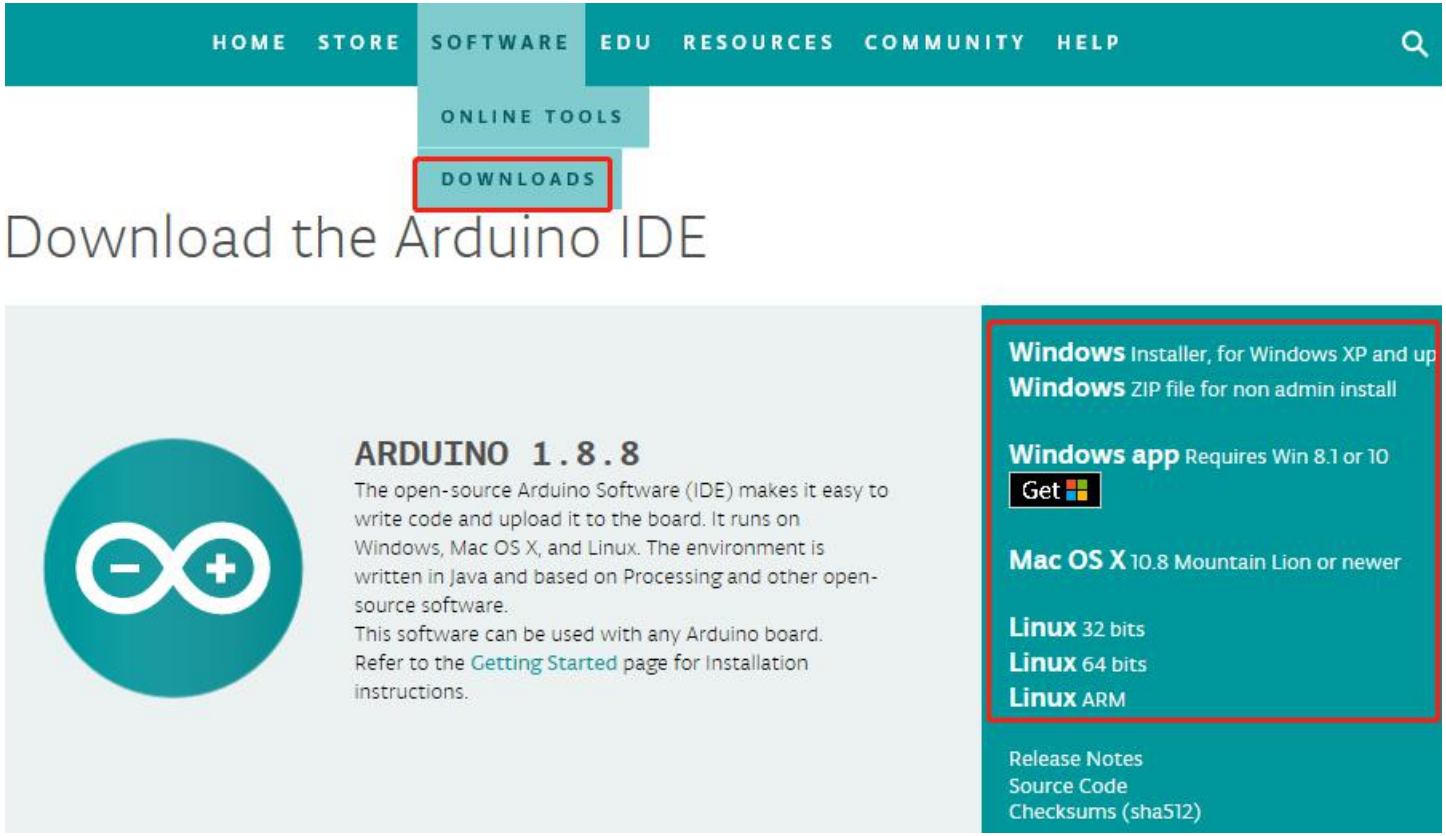
Install Arduino IDE

Here are the installation steps on the windows system.

For other systems, please refer to: **Install Arduinio IDE in different system 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.



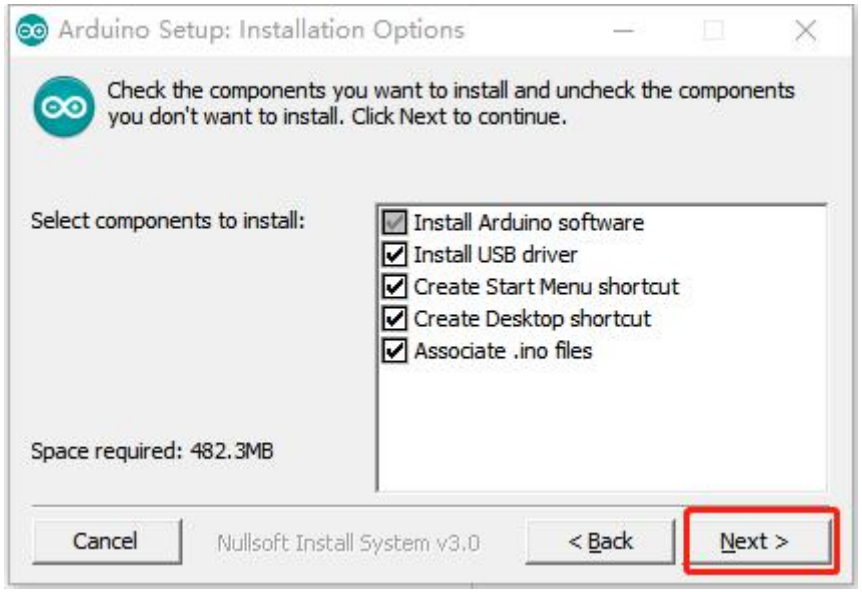
Find the one that suits your operation system and click to download. There are two versions of Arduino for Windows: Installer or ZIP file. You're recommended to download the former.

For Installer File

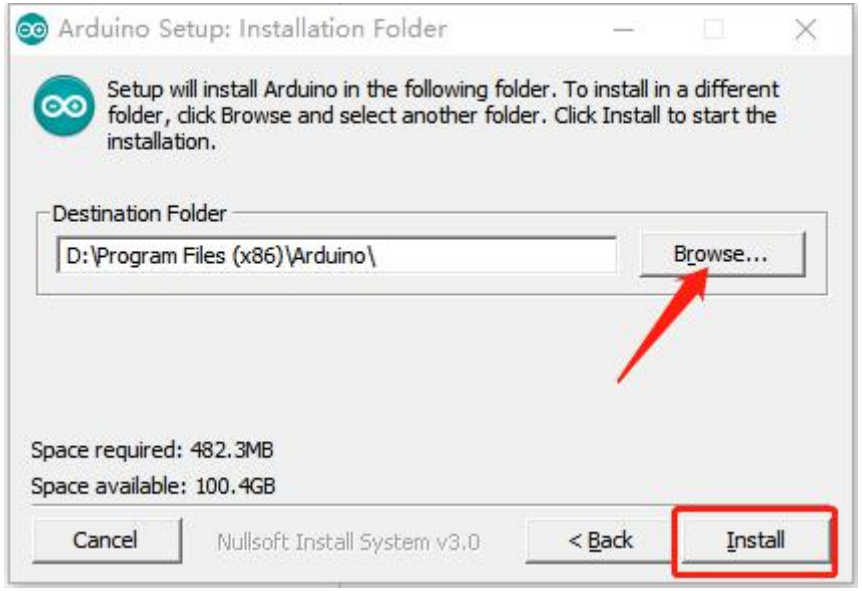
Step 1: Find the .exe file just downloaded.

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:

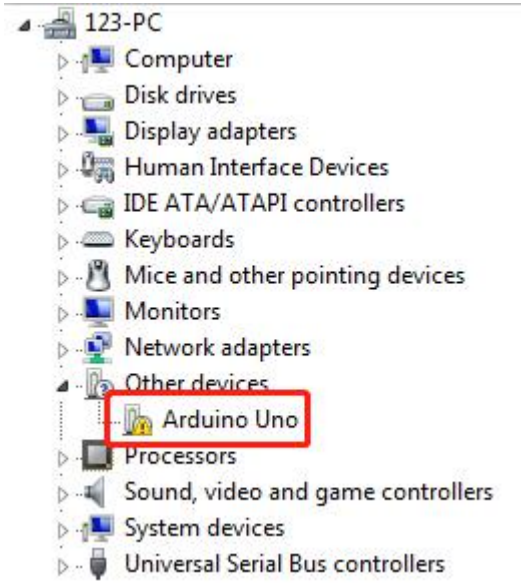
The new IDE may prompt errors when you're compiling code under Windows XP. So if your computer is running on XP, you're suggested to install Arduino 1.0.5 or 1.0.6. Also you can upgrade your computer.

For ZIP File

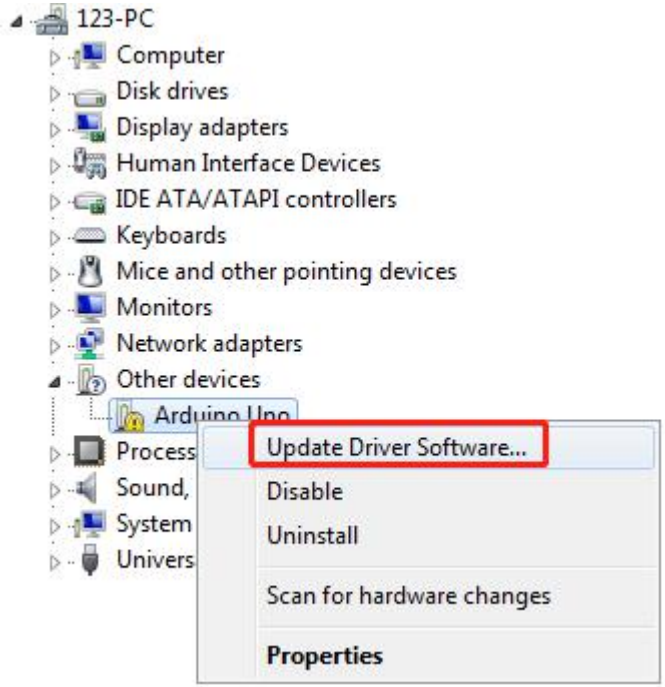
If you download the zip file before, when you connect the MCU to the computer, it may not be recognized. Then you need to install the driver manually. Take the following steps.

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

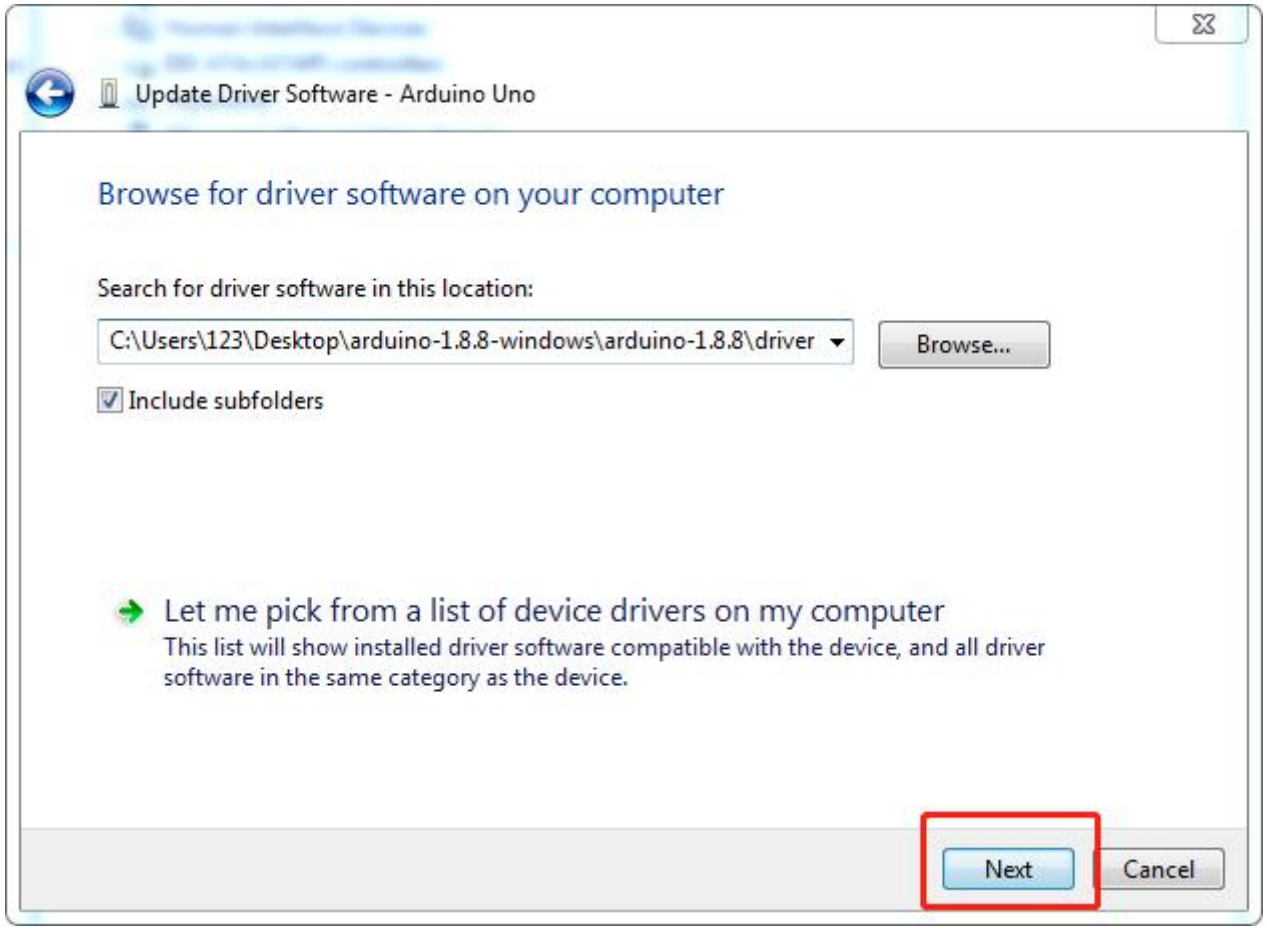
Step2: Go to the **Device Manager**. You will find under other devices, Arduino Uno with an exclamation mark appear, which means the computer did not recognize the board.



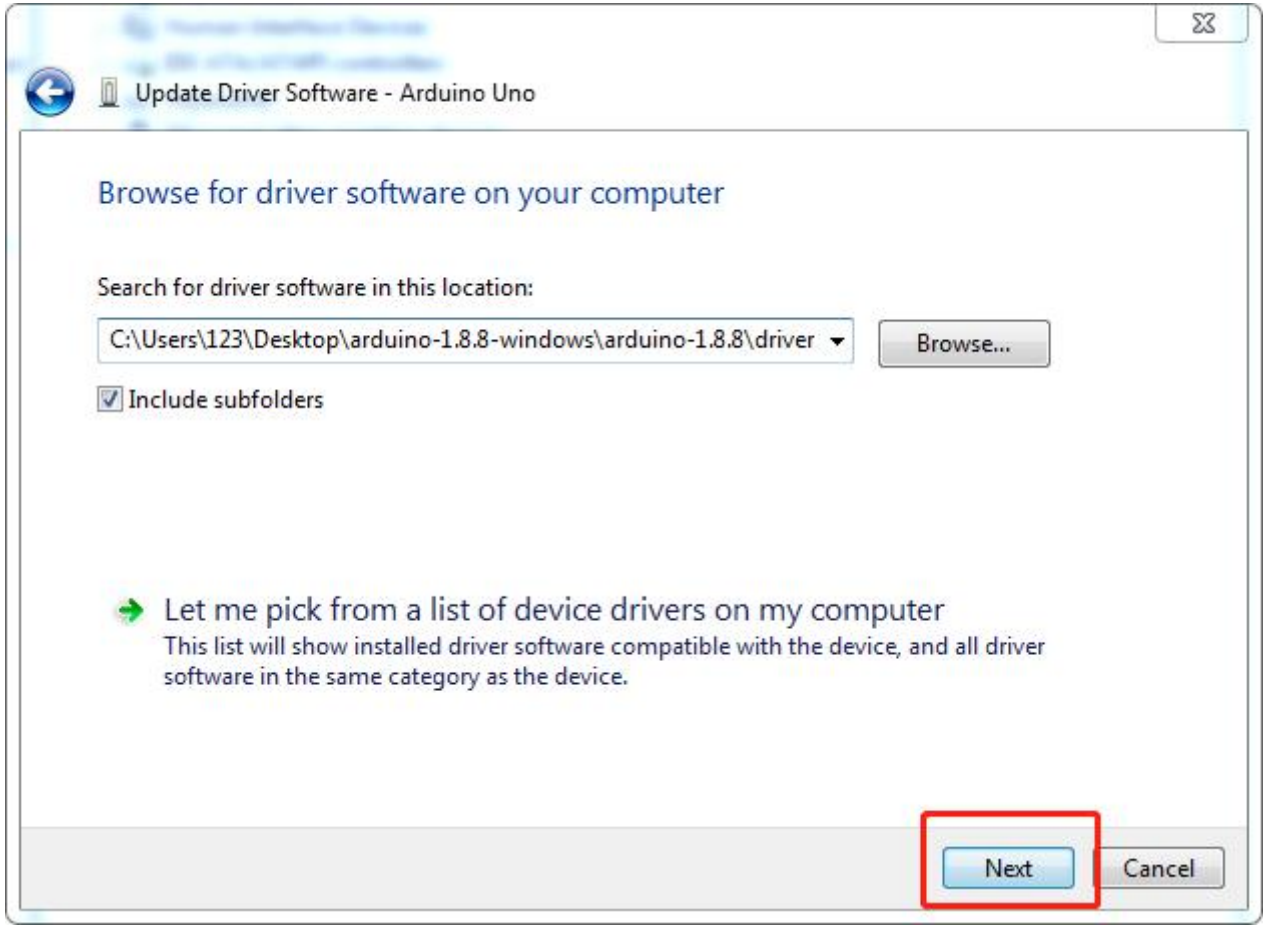
Step3: Right click on **Arduino Uno** and select **Update Driver Software**.



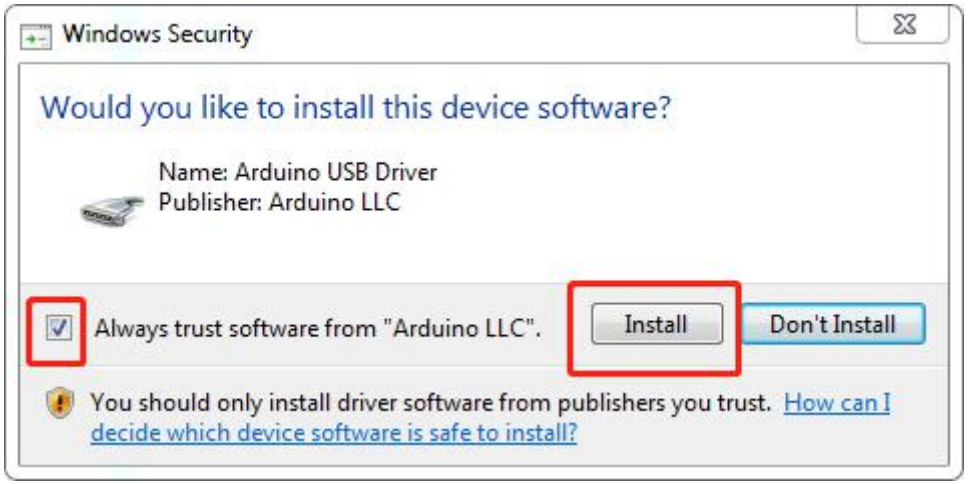
Step4: Choose the second option, **Browse my computer for Driver software**.



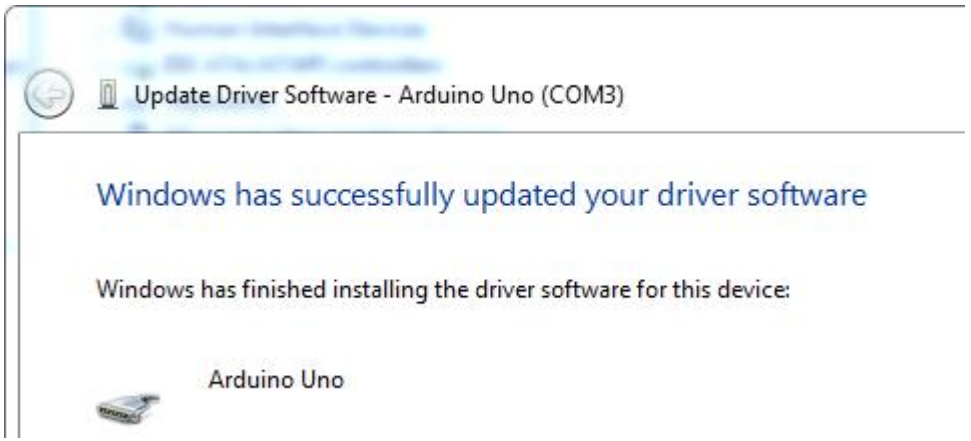
Step5: A window pops up then. Click **Browse**. Then go to the folder where you just extracted the file. Go to the *drivers* folder and click **OK** -> **Next**.



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



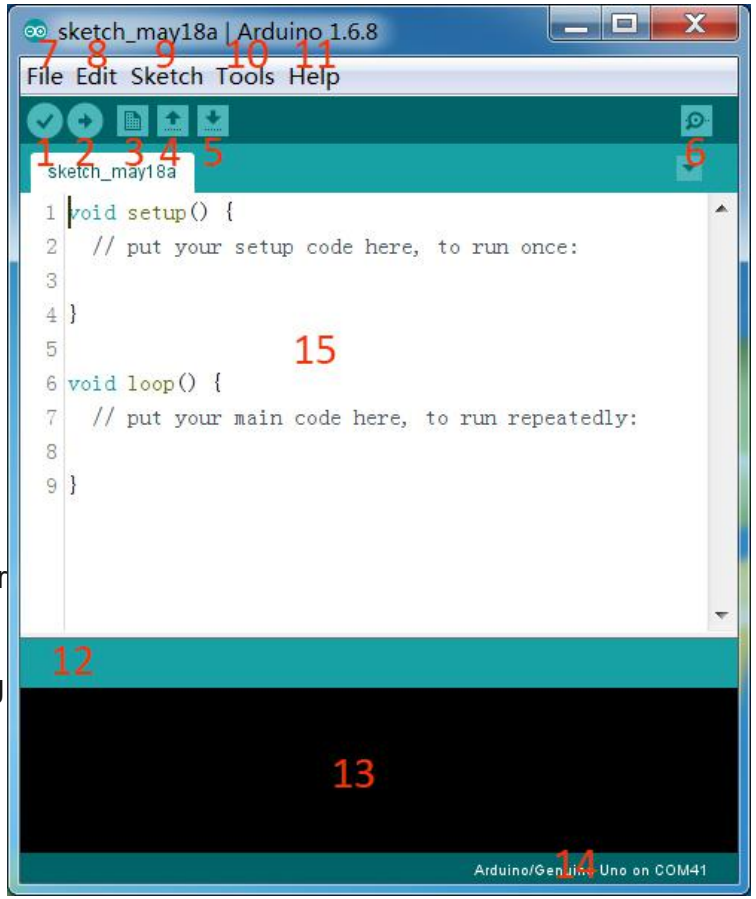
It may need a sec. Then the system prompts you the driver has been installed successfully. So the computer can recognize the board now. Click **Close**.



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.

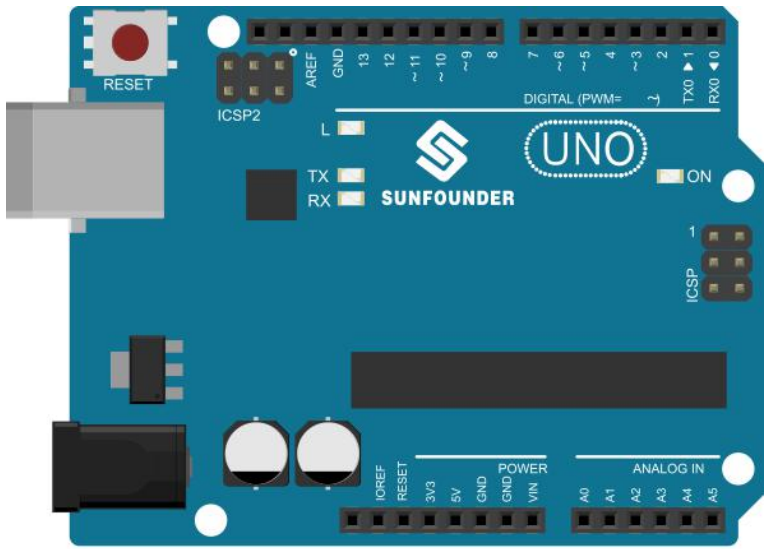
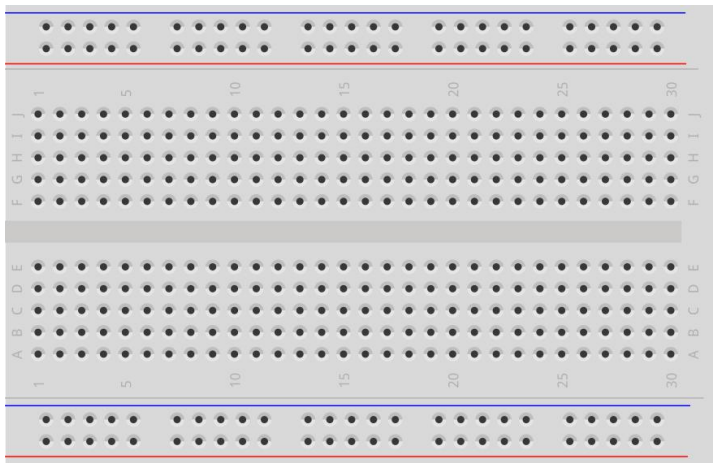
15. The editing area of the IDE. You can write code here.

Lesson 2 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 * Uno Board 	1 * Breadboard 

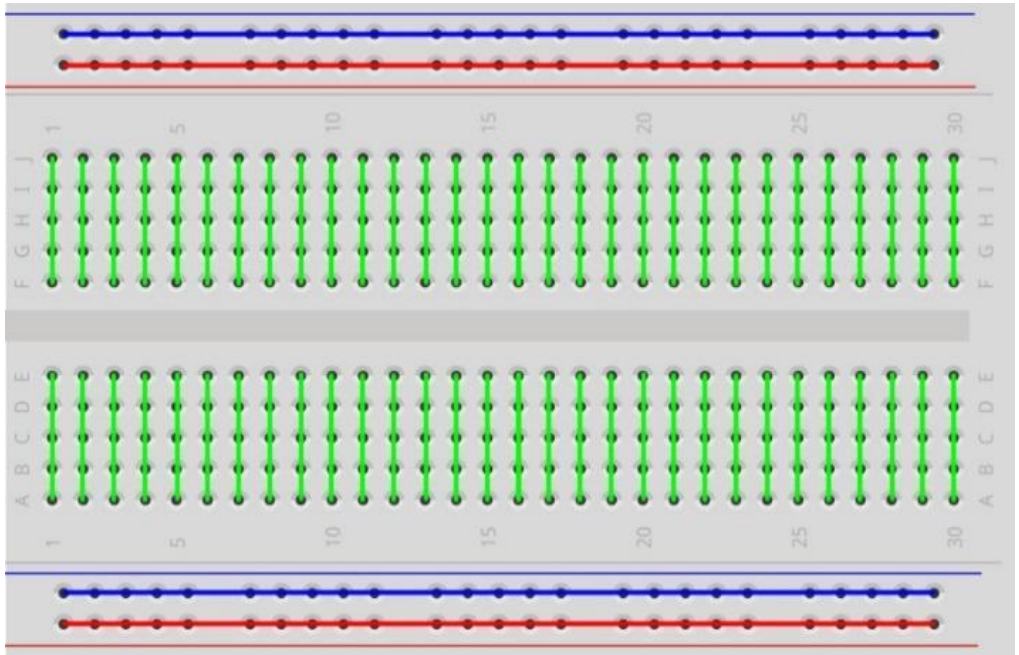
1 * Resistor (220Ω) 	1 * LED 
1 * USB Cable 	Several Jumper Wires 

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.

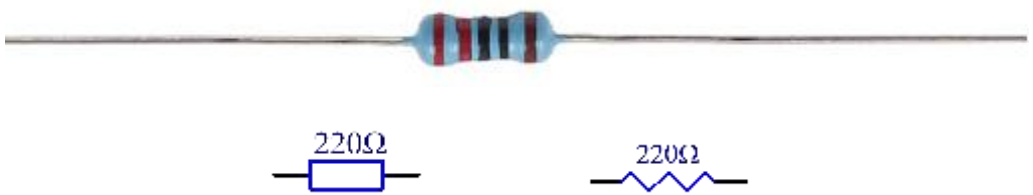
This is the internal structure of a full+ breadboard. Although there are holes on the breadboard, internally some of them are connected with metal strips.



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.

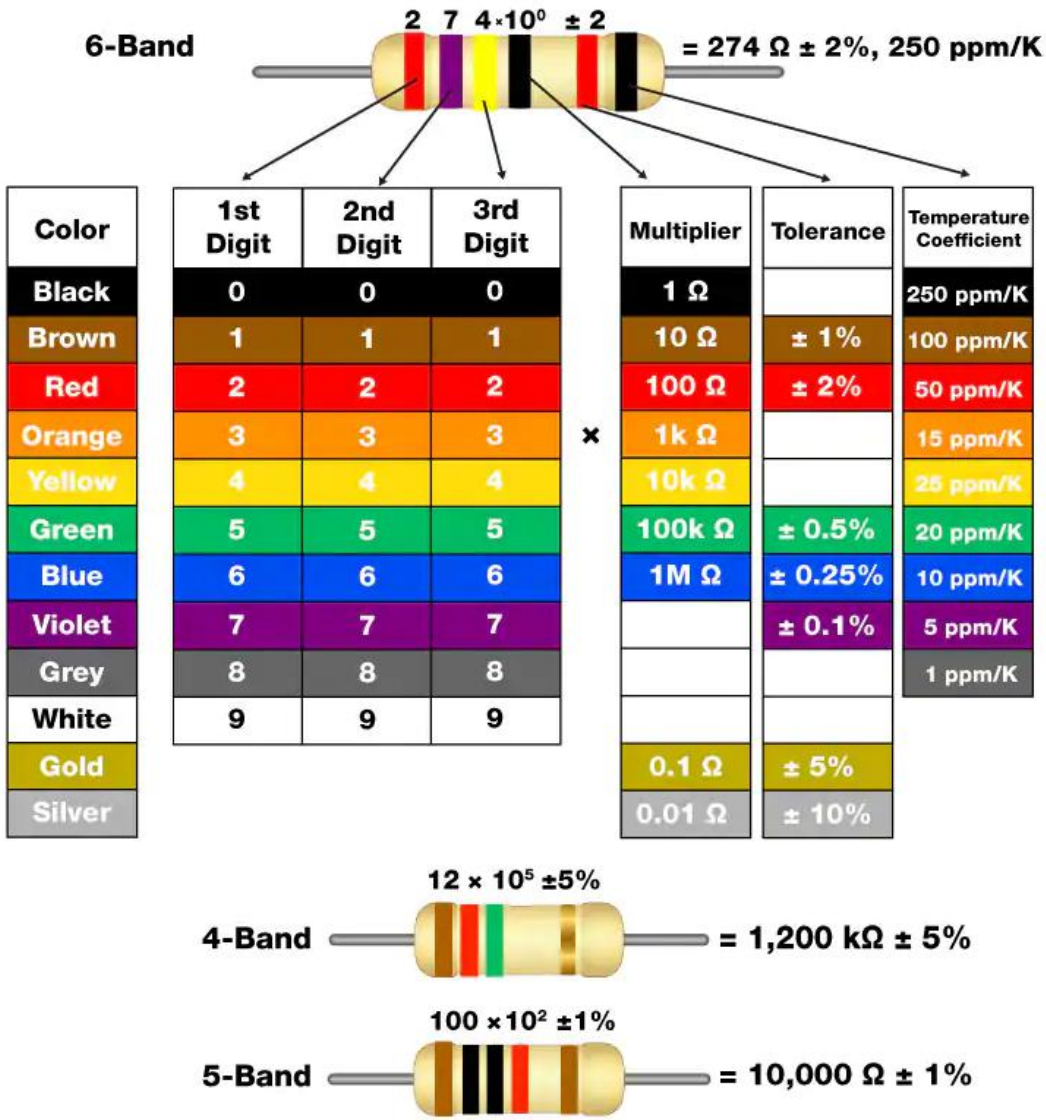
The resistors in this kit are fixed ones. It is essential in the circuit to protect the connected components. The following pictures show a real 220Ω resistor and two generally used circuit symbols for resistor. Ω is the unit of resistance and the larger includes KΩ, MΩ, etc. Their relationship can be 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.

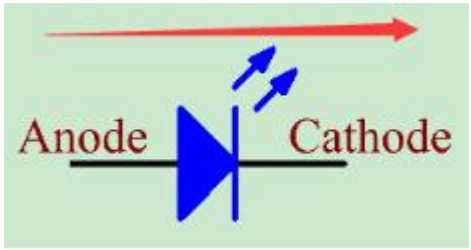
When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster. If you are not sure about the value, use the multimeter.

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


6-Band
2 7 4×10⁰ ± 2
= 274 Ω ± 2%, 250 ppm/K
Color
1st Digit
2nd Digit
3rd Digit
Multiplier
Tolerance
Temperature Coefficient
Black 0 0 0 1 Ω 250 ppm/K
Brown 1 1 1 10 Ω ± 1% 100 ppm/K
Red 2 2 2 100 Ω ± 2% 50 ppm/K
Orange 3 3 3 1k Ω 15 ppm/K
Yellow 4 4 4 10k Ω 25 ppm/K
Green 5 5 5 100k Ω ± 0.5% 20 ppm/K
Blue 6 6 6 1M Ω ± 0.25% 10 ppm/K
Violet 7 7 7 ± 0.1% 5 ppm/K
Grey 8 8 8 1 ppm/K
White 9 9 9
Gold 0.1 Ω ± 5%
Silver 0.01 Ω ± 10%
×
12 × 10⁵ ±5%
4-Band
= 1,200 kΩ ± 5%
100 ×10² ±1%
5-Band
= 10,000 Ω ± 1%

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.

Male-to-Female

Male-to-Male

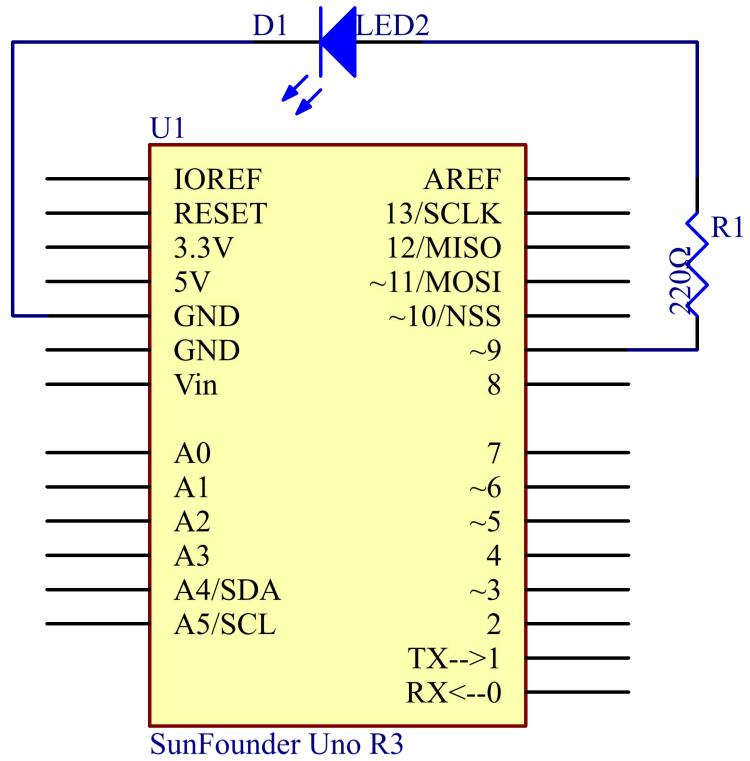
Female-to-Female

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 Uno 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.

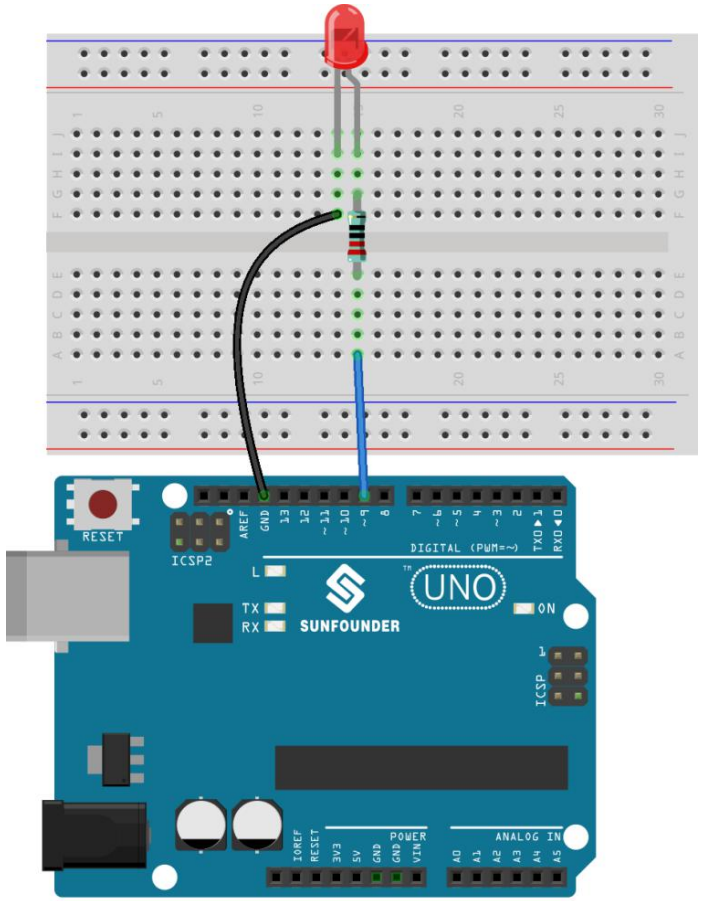
The schematic diagram:



Experimental Procedures

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

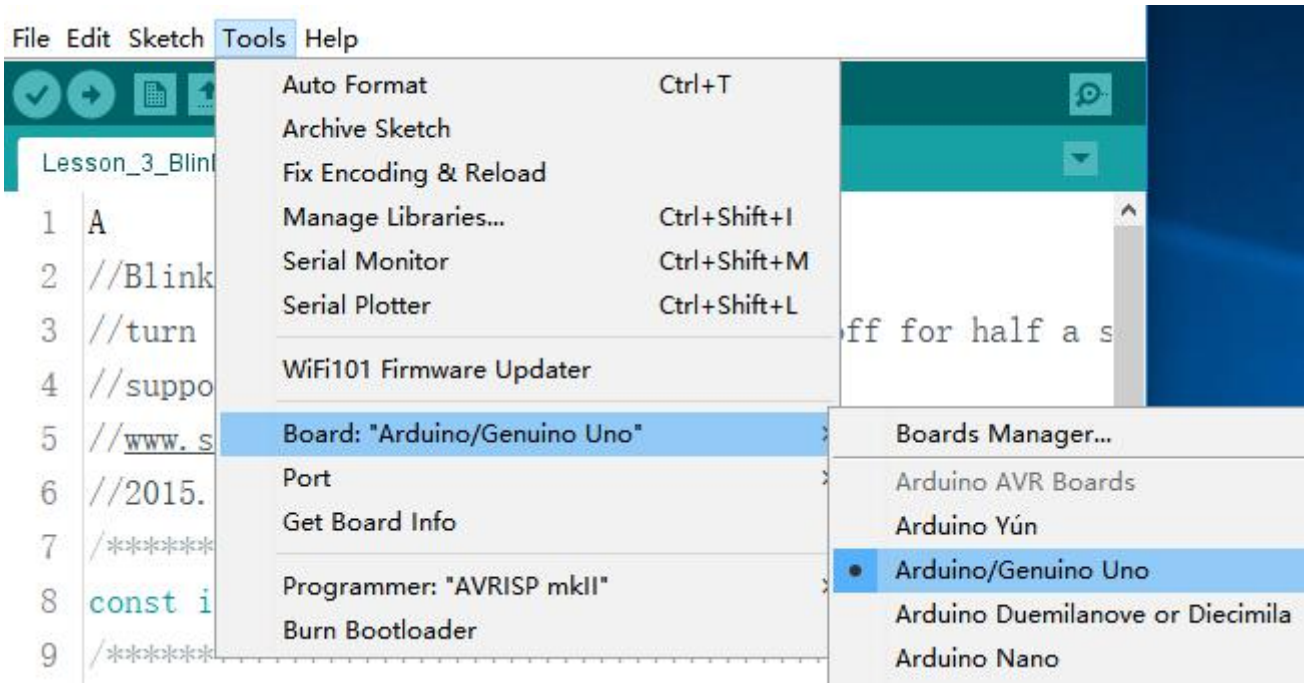
Then plug the board into the computer with a 5V USB cable.



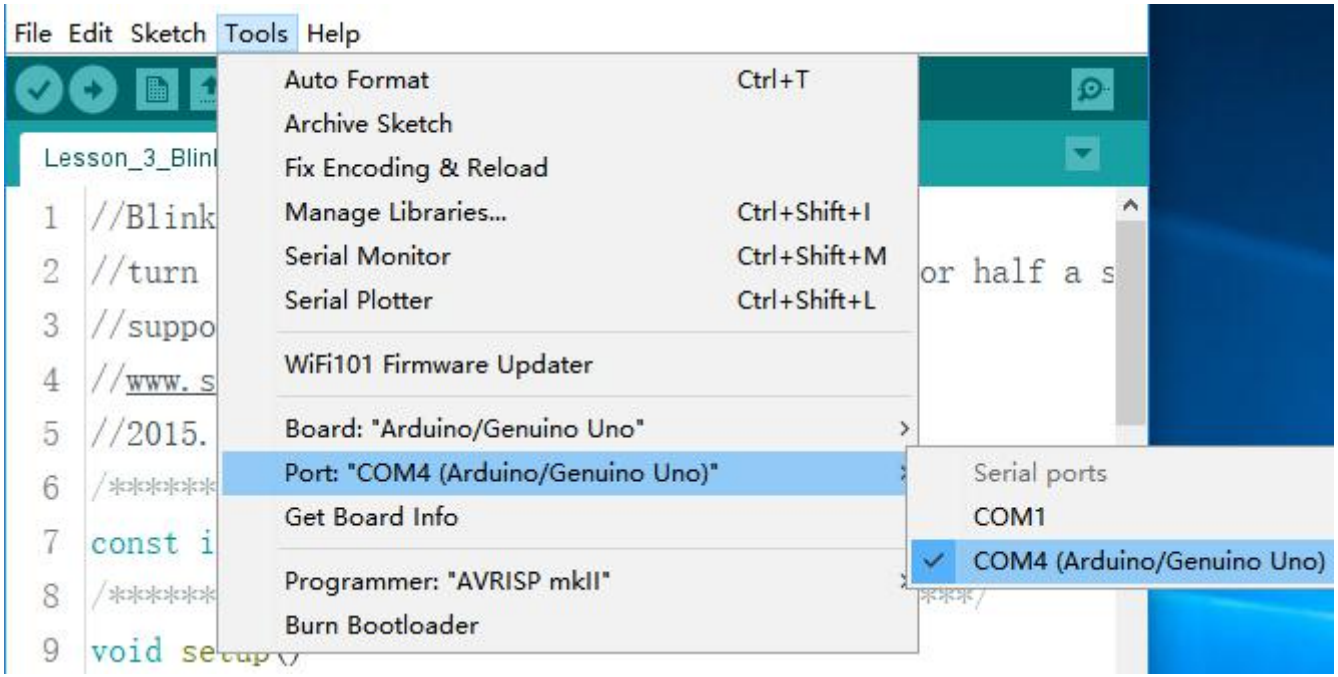
Step 2: Open the Lesson_2_Blinking_LED.ino code file in the path of *electronic-kit\for-Arduino\code\Lesson_2_Blinking_LED*

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 Uno**.

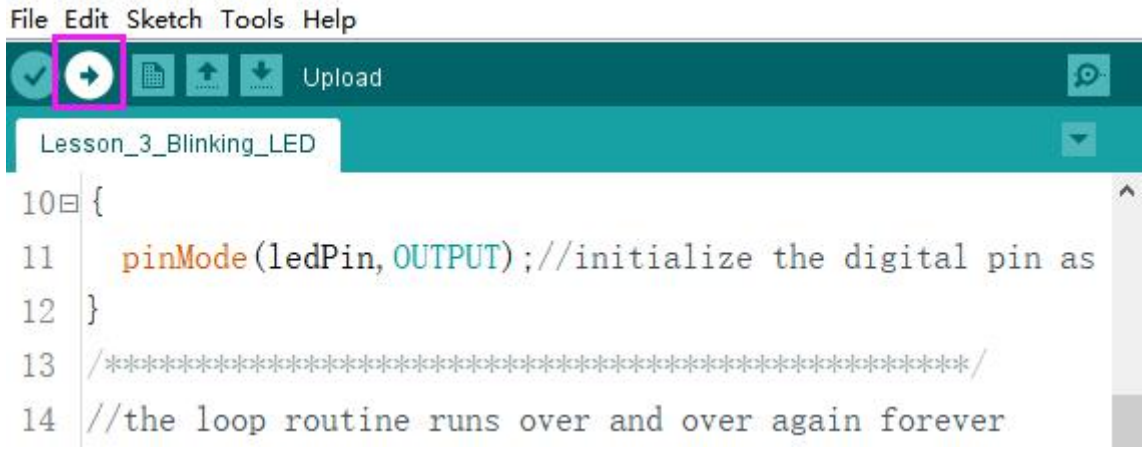


Then select **Tools** ->**Port**. Your port should be different from mine.

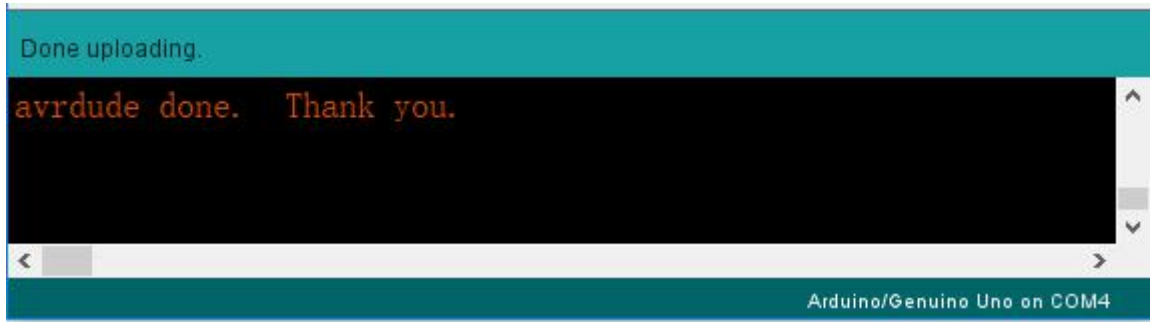


Step 4: Upload the sketch to the Uno board.

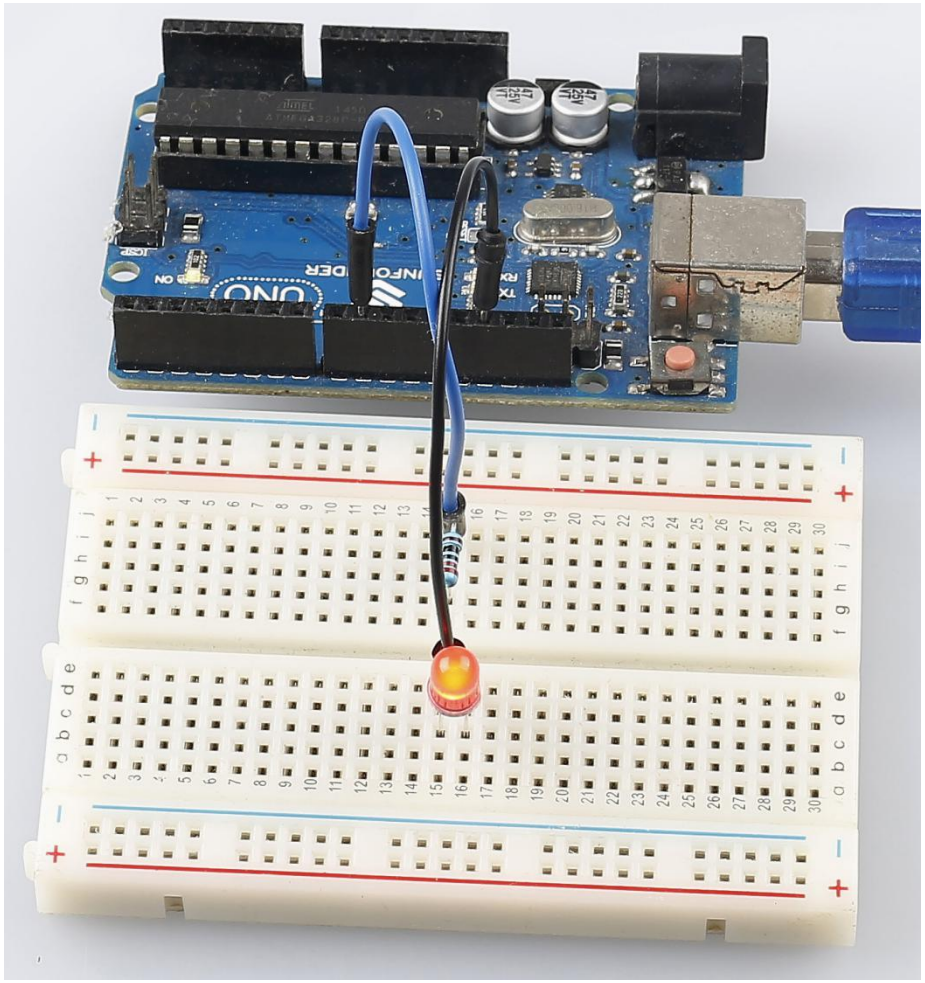
Click the **Upload** icon to upload the code to the control board.



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 2-1 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 2-2 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);//initialize the digital pin as an 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.

Code Analysis 2-3 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

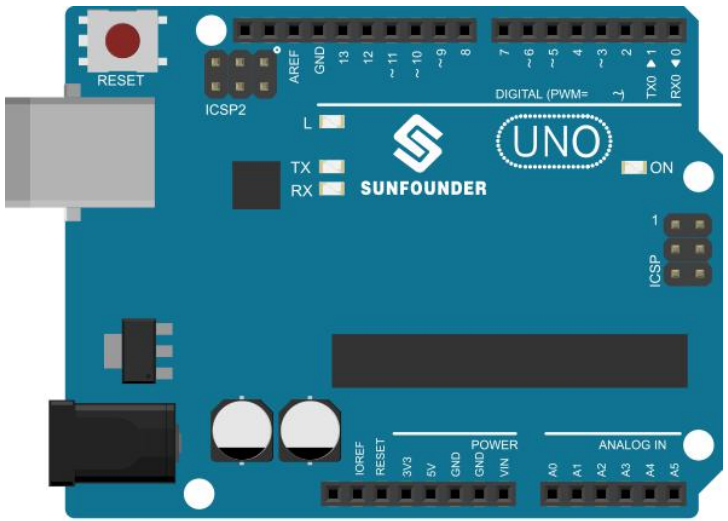
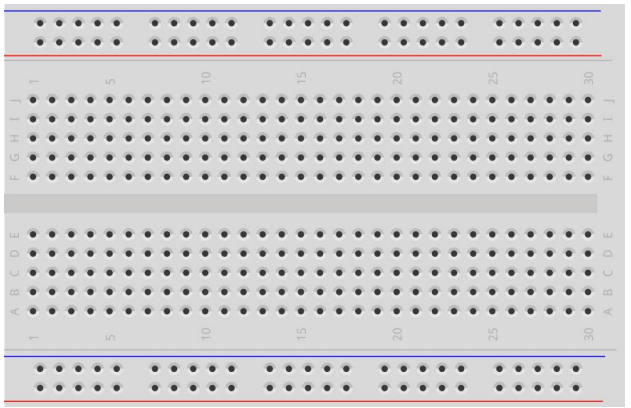
Through this experiment, you have learned how to turn on an LED. You can also change the blinking frequency of the LED by changing the *num* value in the delay function *delay (num)*. For example, change it to ***delay (250)*** and you will find that the LED blinks more quickly.

Lesson 3 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 Uno 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 * Uno Board 	1 * Breadboard 

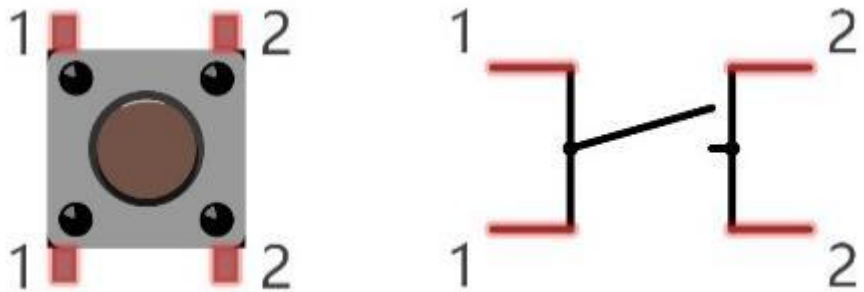
1 * Resistor (10kΩ) 	1 * Button 
1 * USB Cable 	Several Jumper Wires 

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.

Two pins on the left is connected, and the right is similar as the left, which is shown in the below:



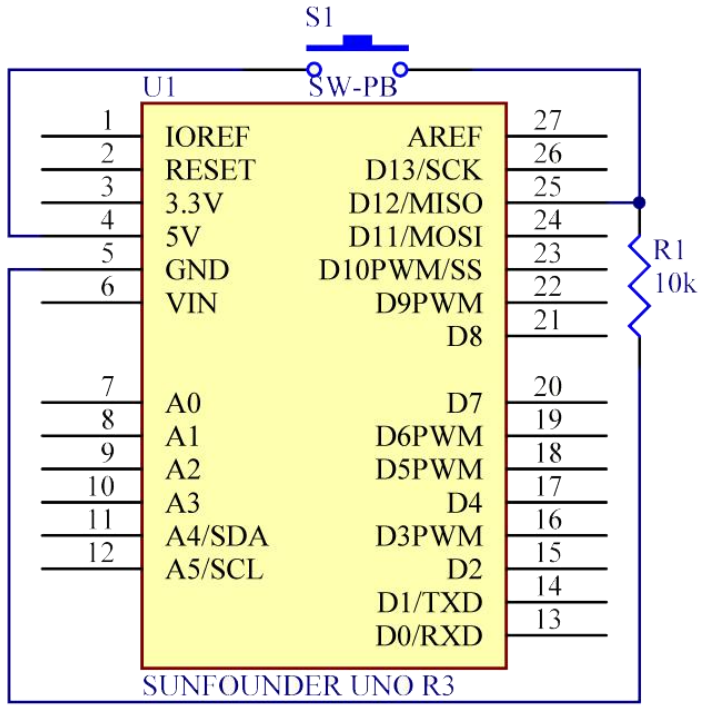
The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.

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 (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：



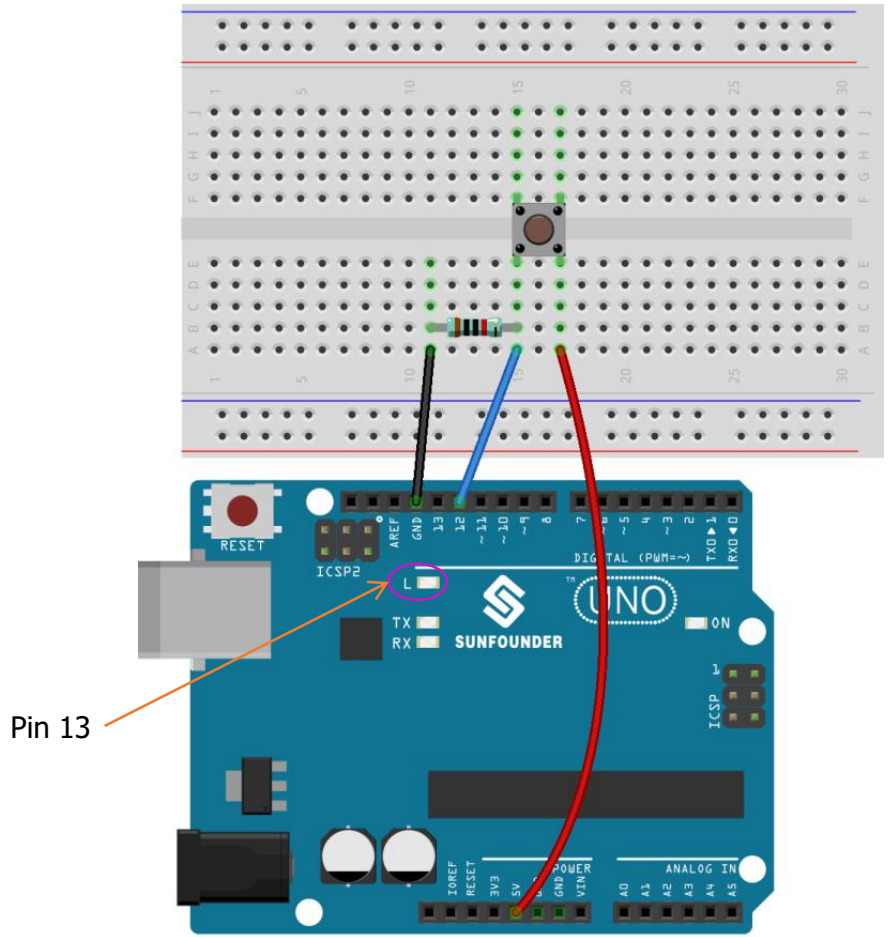
Experimental Procedures

Step 1: Build the circuit

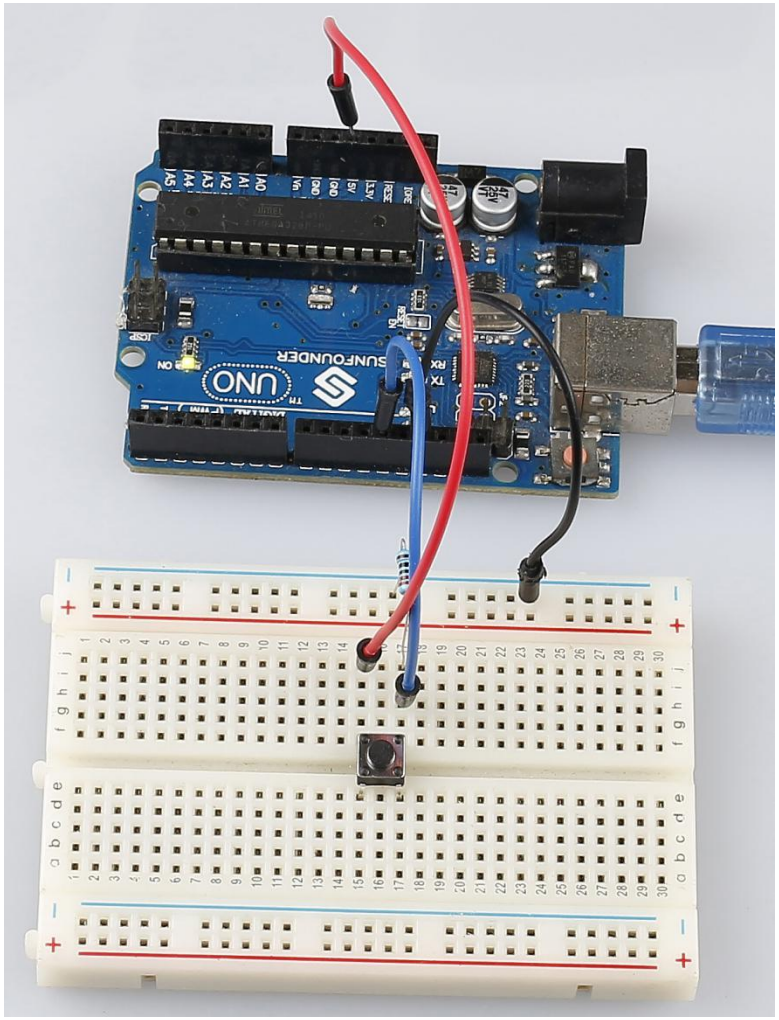
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.



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



Code Analysis

Code Analysis 3-1 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 3-2 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 3-3 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 3-4 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

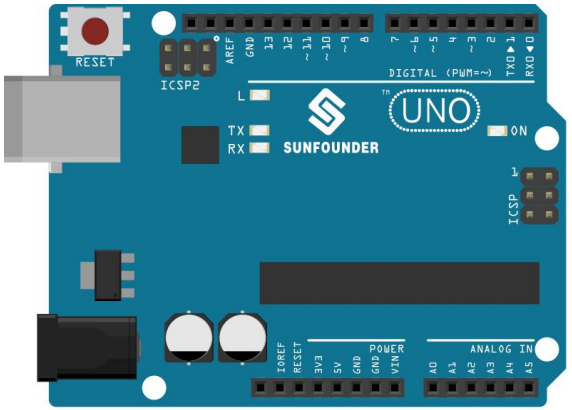
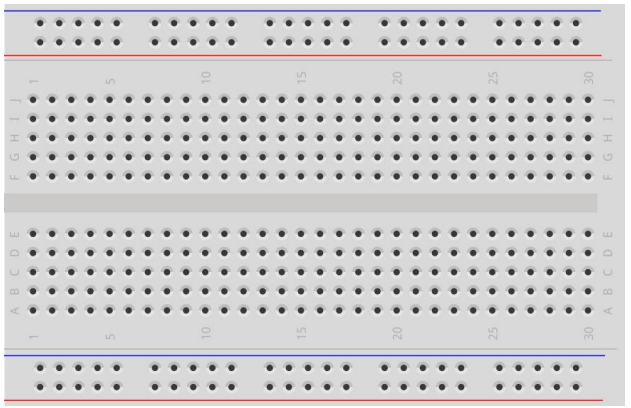
You can also change the code to: when the button is pressed, if (buttonState=HIGH). The LED goes out (digitalWrite(ledPin, LOW)). When the button is released (the else), the LED lights up ((digitalWrite(ledPin, HIGH)). You only need to replace the code in **if** with those in **else**.

Lesson 4 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

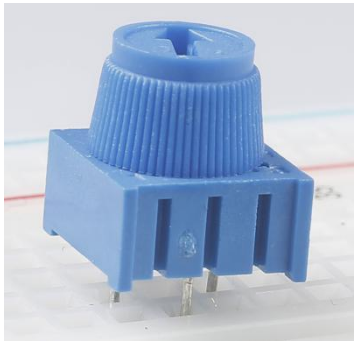
1 * Uno Board	1 * Breadboard
	

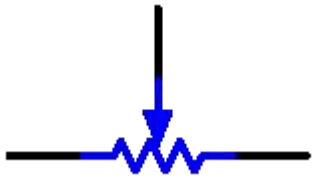
1 * Resistor (220Ω) 	1* Potentiometer 	1 * LED 
1 * USB Cable 	Several Jumper Wires 	

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

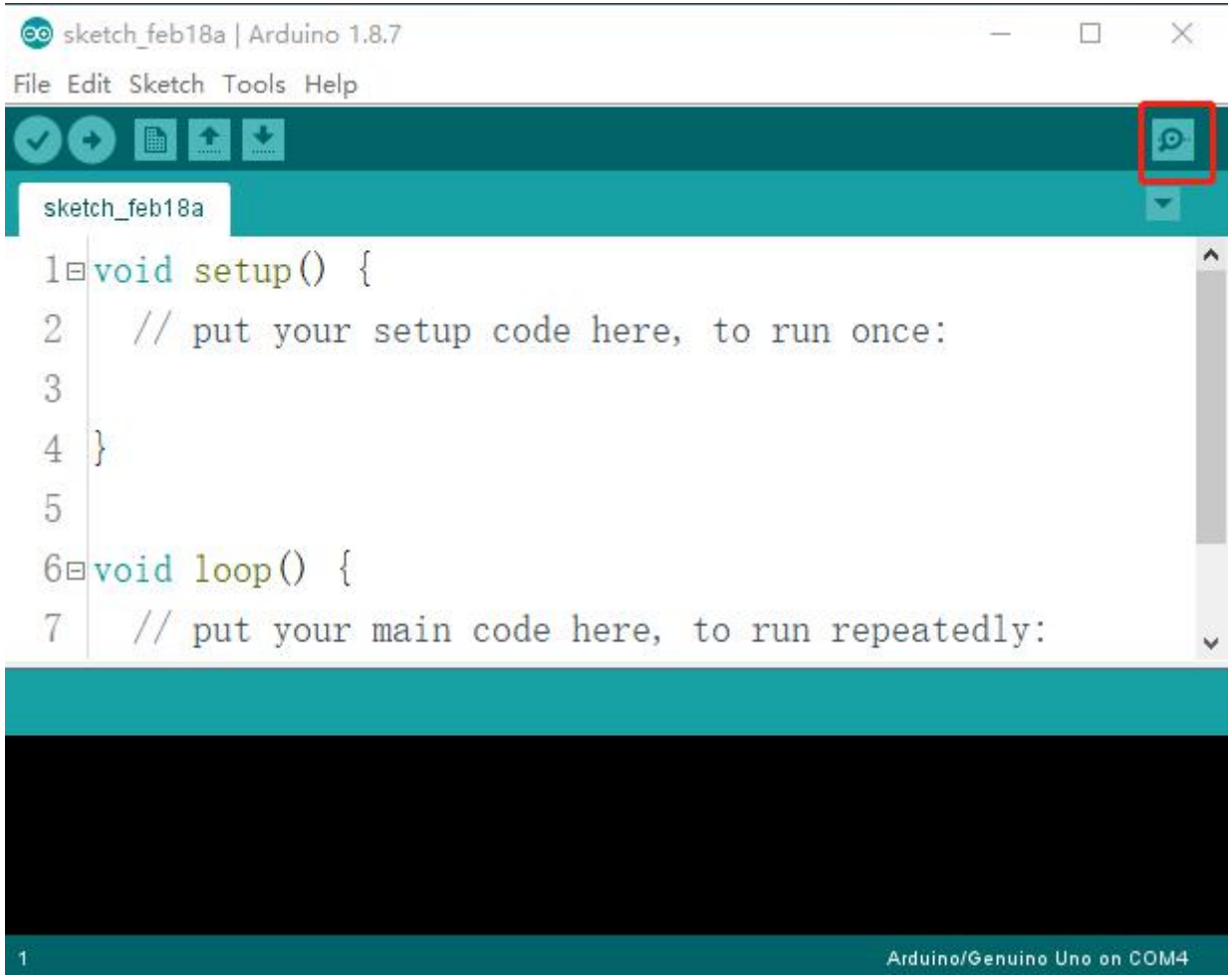
   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

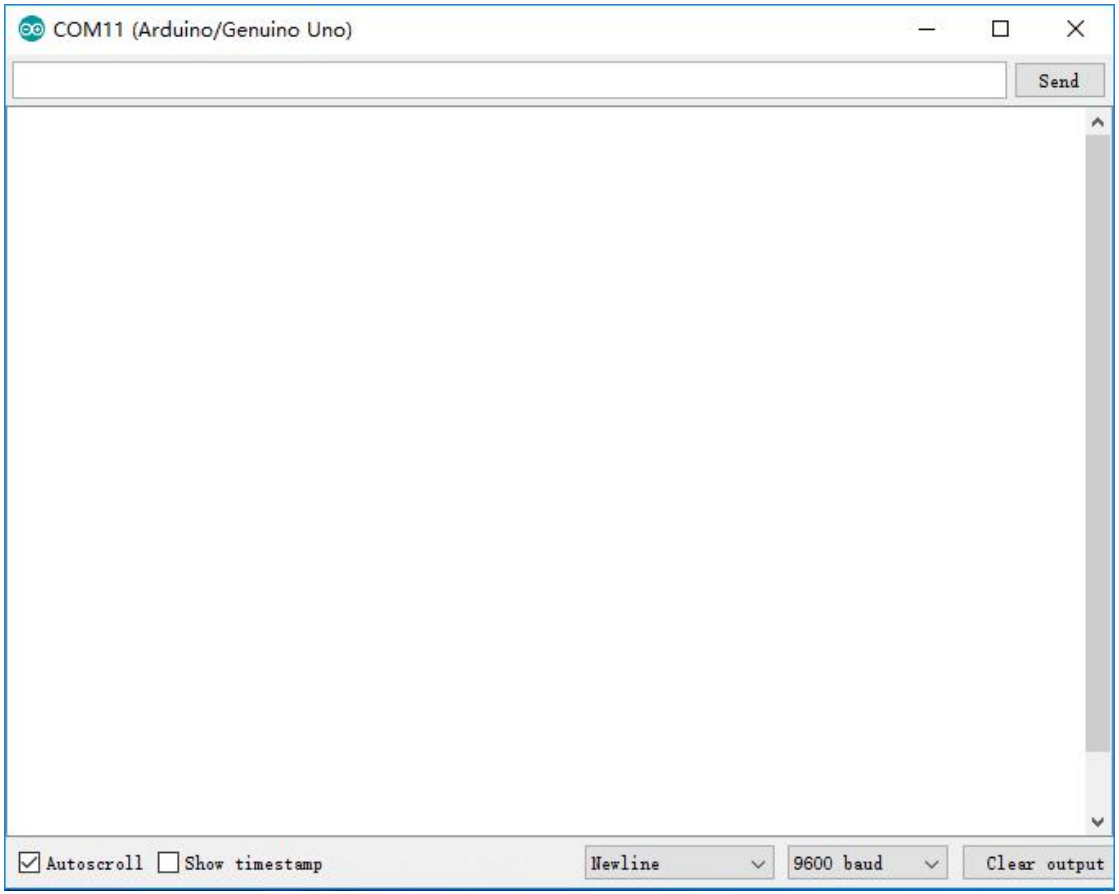
   When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

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.

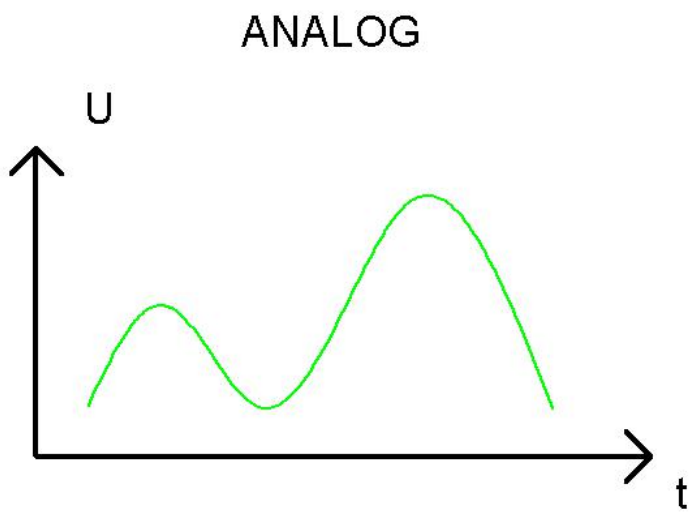


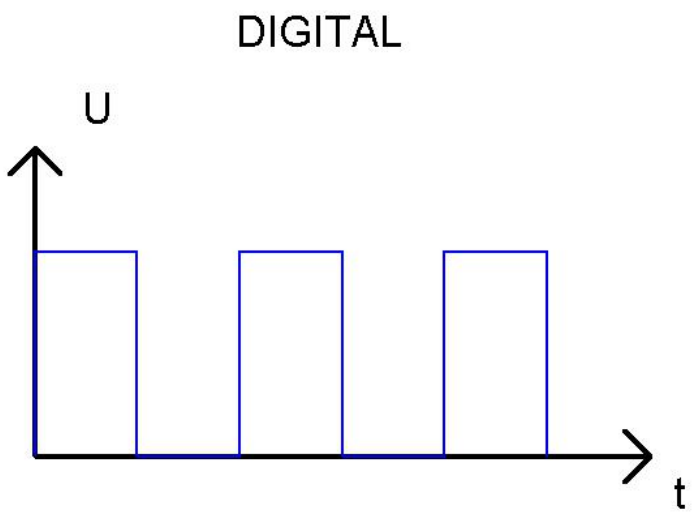
Here, the Serial Monitor serves as a transfer station for communication between your computer and the Uno board. First, the computer transfers data to the Serial Monitor, and then the data is read by the Uno board. Finally, the Uno will perform related operations. Click the icon at the top right corner and a window will pop up as shown below:



Analog V.S. Digital

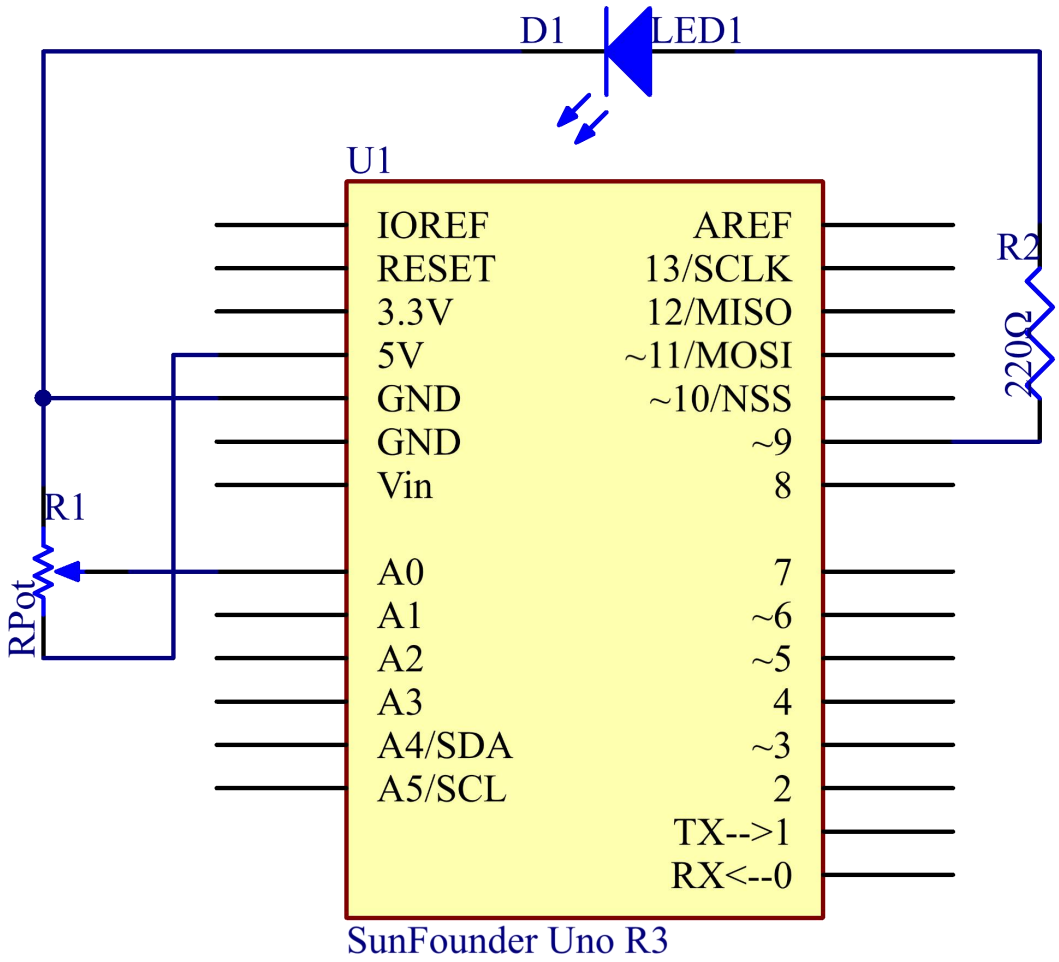
A linear potentiometer is an analog electronic component. So what's the difference between an analog value and a digital one? Simply put, digital means on/off, high/low level with just two states, i.e. either 0 or 1. But the data state of analog signals is linear, for example, from 1 to 1000; the signal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





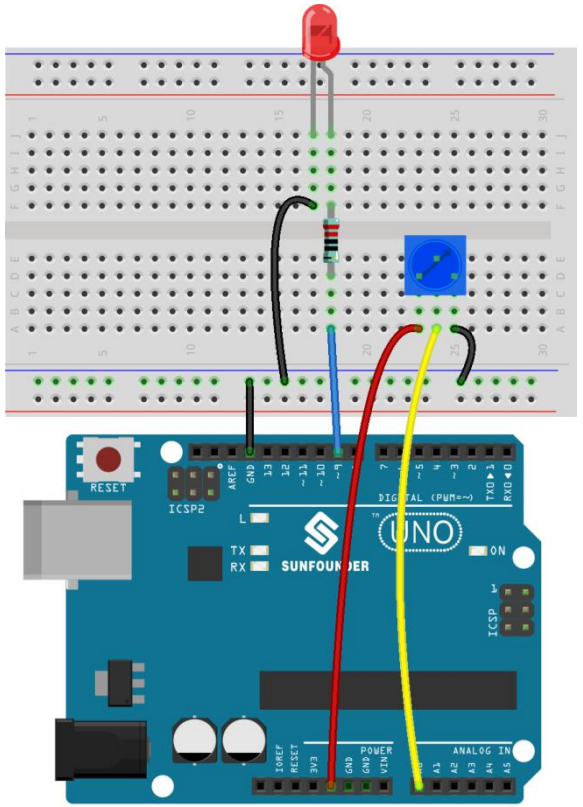
Principle: In this experiment, the potentiometer is used as voltage divider, meaning connecting devices to all of its three pins. Connect the middle pin of the potentiometer to pin A0 and the other two pins to 5V and GND respectively. Therefore, the voltage of the potentiometer is 0-5V. Spin the knob 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:


D1
LED1
U1
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4/SDA
A5/SCL
AREF
13/SCLK
12/MISO
~11/MOSI
~10/NSS
~9
8
7
~6
~5
4
~3
2
TX-->1
RX<--0
R1
RPot
R2
220Ω
SunFounder Uno R3

Experimental Procedures

Step 1: Build the circuit



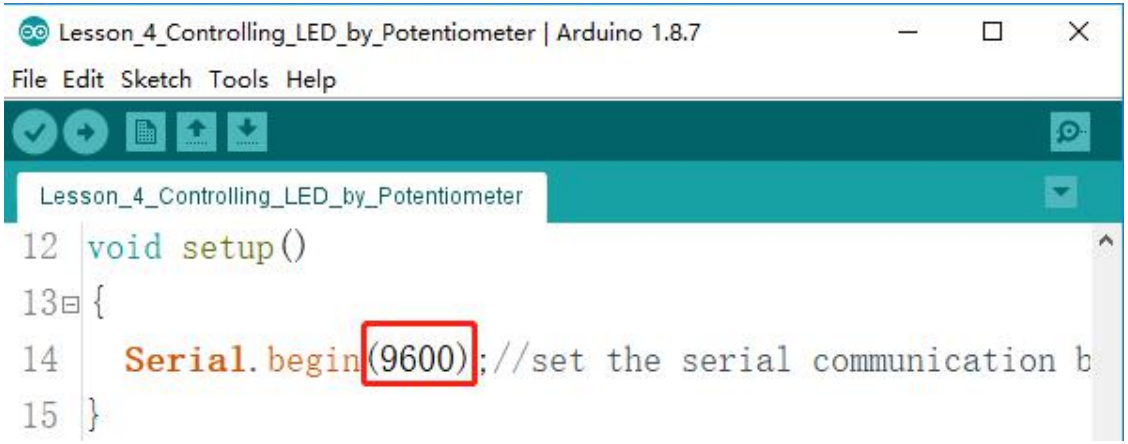
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.

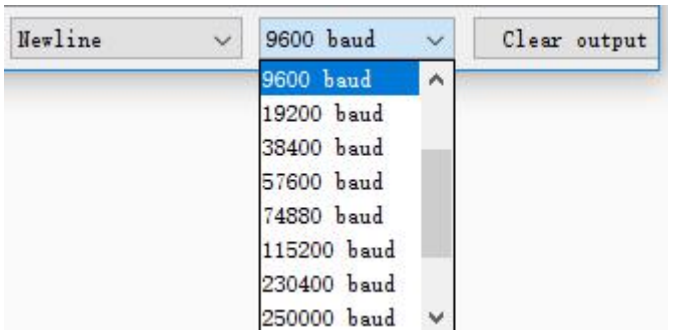
Step5: Open the Serial Monitor.

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.



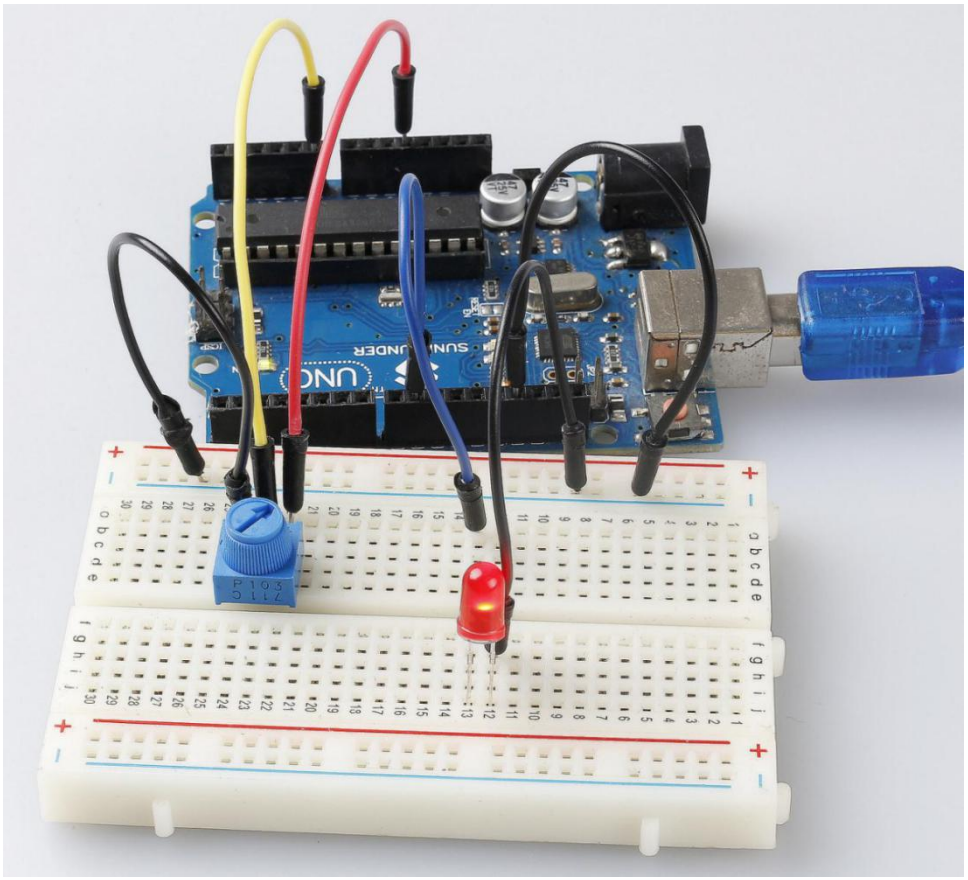
Step6: Set the baud rate to 9600.

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 4-1 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 4-2 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 4-3 Map the values

```
outputValue = map(inputValue, 0, 1023, 0, 255); //Convert from 0-1023 proportional to the number of a number of from 0 to 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 4-4 Write the value of the potentiometer to LED

```
analogWrite(ledPin, outputValue); //turn the LED on depending 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!

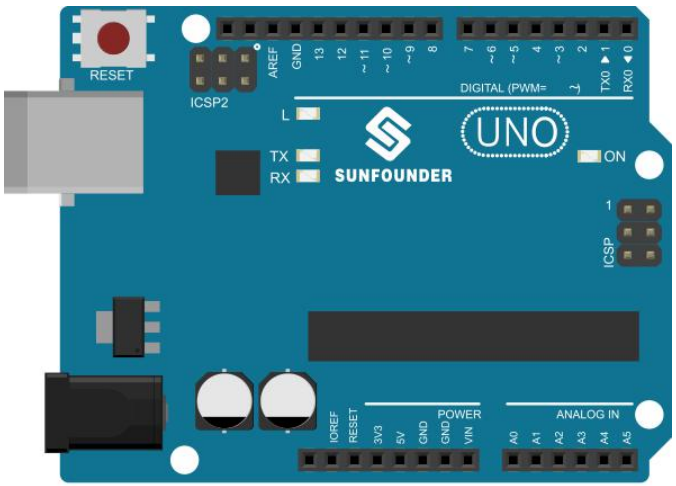
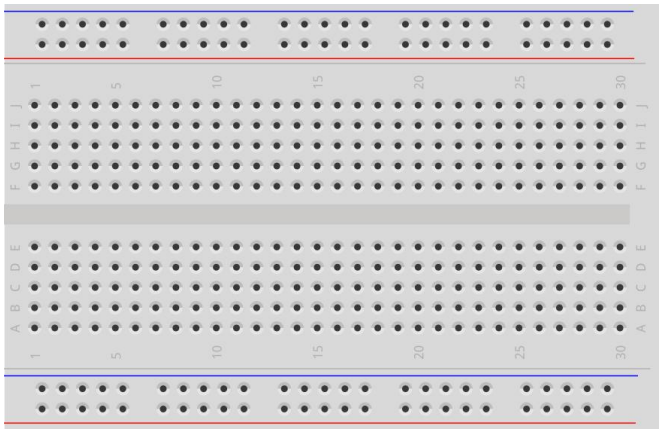
```
inputValue = analogRead(analogPin);//read the value from the sensor
digitalWrite(ledPin,HIGH);
delay(inputValue);
digitalWrite(ledPin,LOW);
delay(inputValue);
```

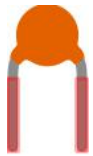
Lesson 5 Doorbell

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 * Uno Board 	1 * Breadboard 

1 * 104 Capacitor 	1*Buzzer(Active) 	1 * Resistor (10kΩ) 	1 * Button 
1 * USB Cable 		Several Jumper Wires 	

Experimental Principle

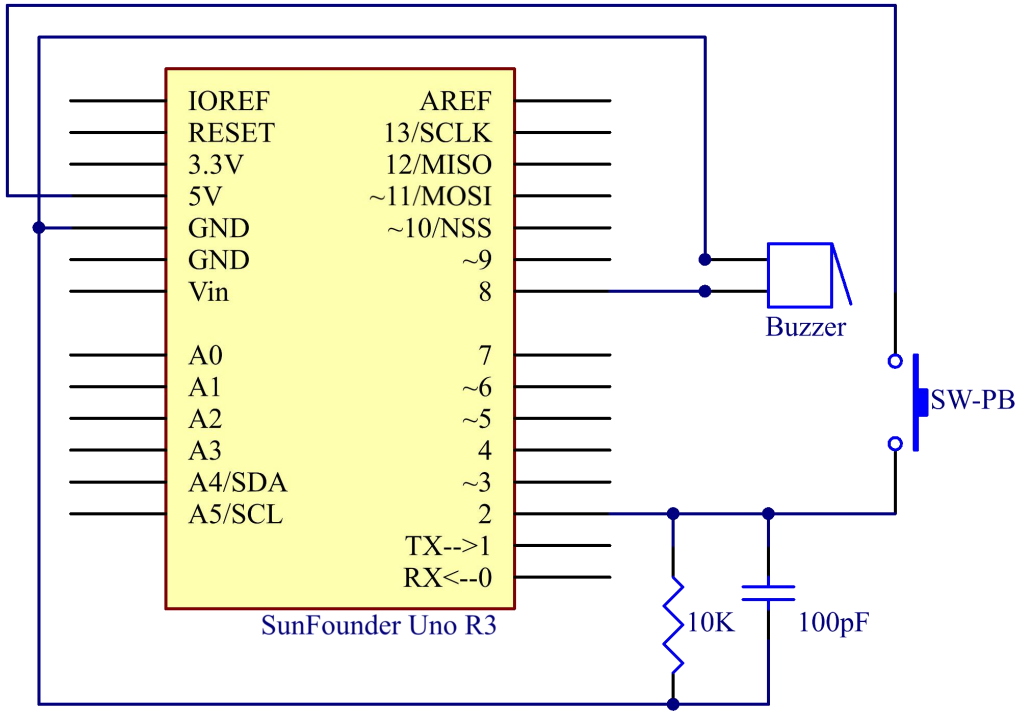
As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products for voice devices. Buzzers can be categorized 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：



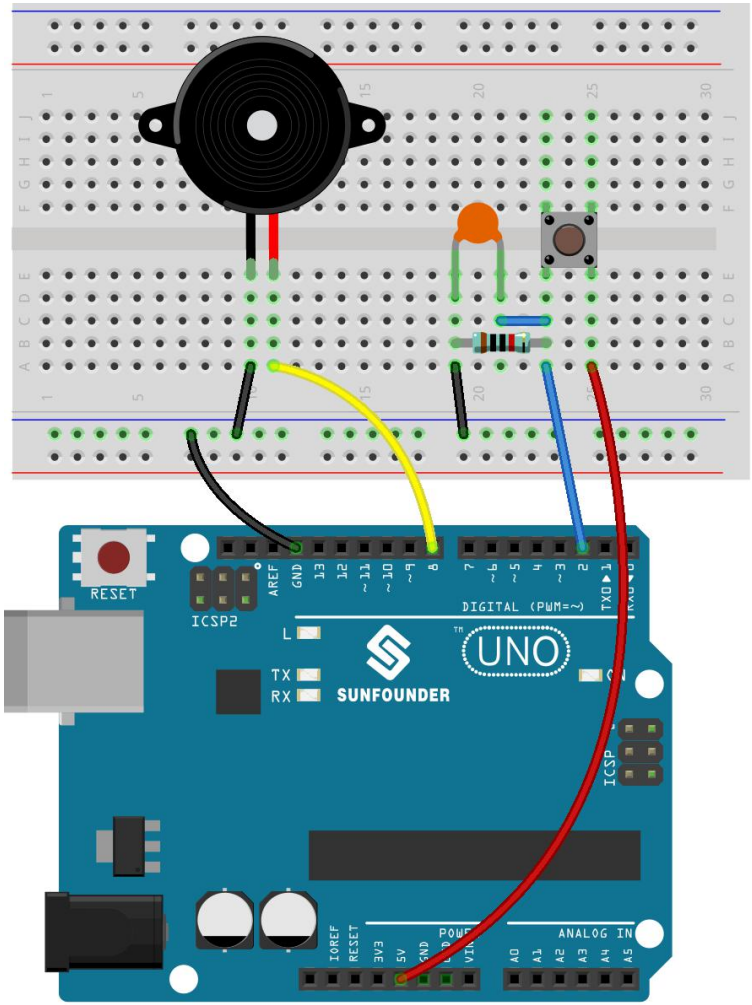
Experimental Procedures

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

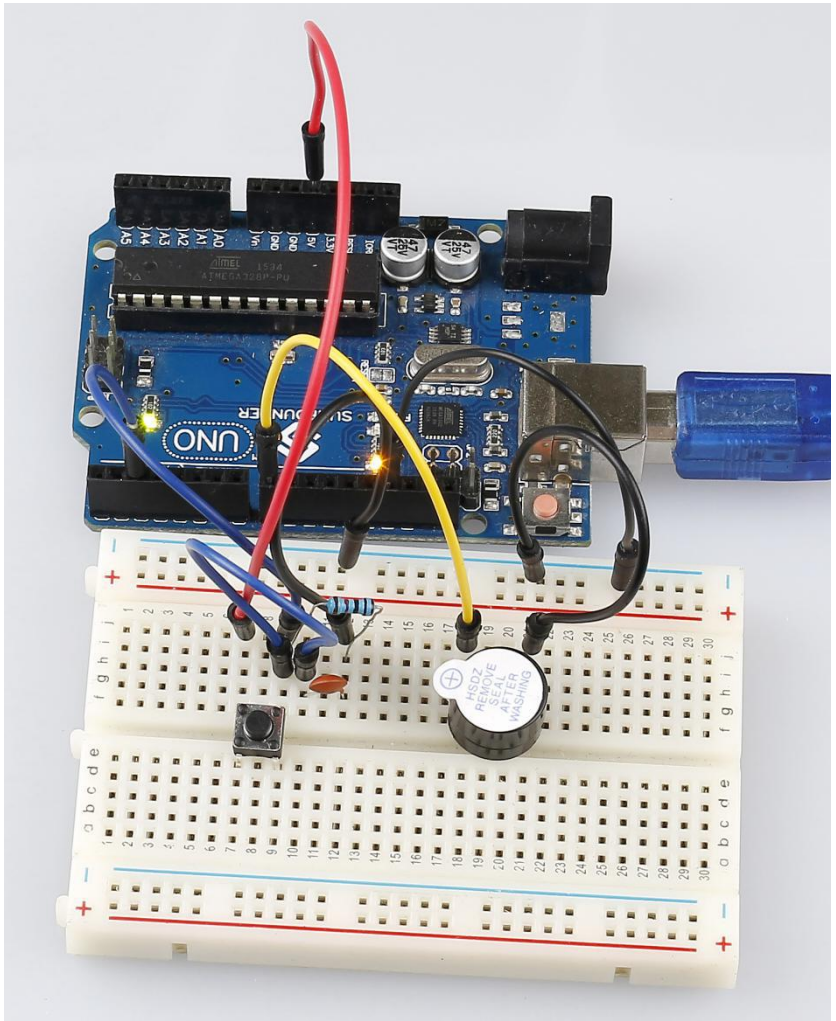
Step 2: Open the code file.

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 5-1 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 5-2 Set the input and output status of the pins

```
void setup()

{

  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 5-3 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 5-4 Turn on the LED when the button is pressed

```
if (buttonState == HIGH )  //When press the button, run the following code.
 {
   for (i = 0; i < 50; i++)  //When i=o, which accords with the condition i<=50, i++ equals to 1 (here in i = i + 1, the two "i"s are not the same, but i_now = i_before + 1). Run the code in the curly braces: let the buzzer beep for 3ms and stop for 3ms. Then repeat 50 times.
   { digitalWrite(buzzerPin, HIGH);  //Let the buzzer beep.
     delay(3);//wait for 3ms
     digitalWrite(buzzerPin, LOW);   //Stop the buzzer.
     delay(3);//wait for 3ms }
    for (i = 0; i < 80; i++)   //Let the buzzer beep for 5ms and stop for 5ms, repeat 80 times.
   { digitalWrite(buzzerPin, HIGH);
     delay(5);//wait for 5ms
     digitalWrite(buzzerPin, LOW);
     delay(5);//wait for 5ms
   }}
```

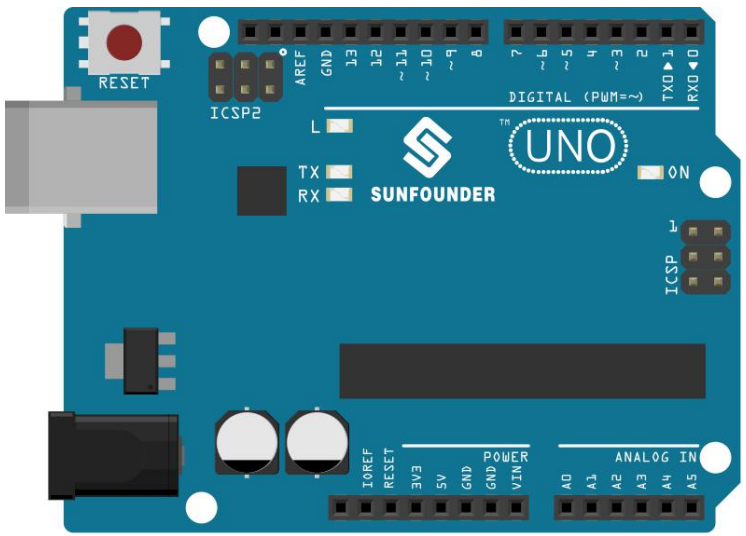
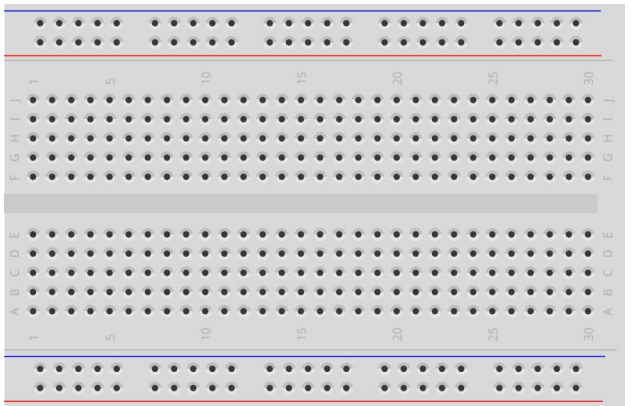
In this part, when the ***buttonState*** is High level, then let the buzzer beeping in different frequency which can simulate the doorbell.

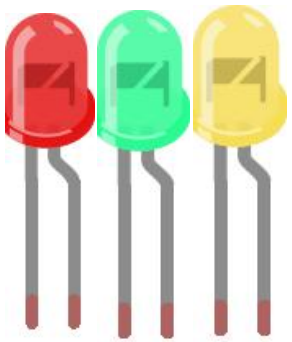
Lesson 6 Photoresistor

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 * Uno Board 	1 * Breadboard 

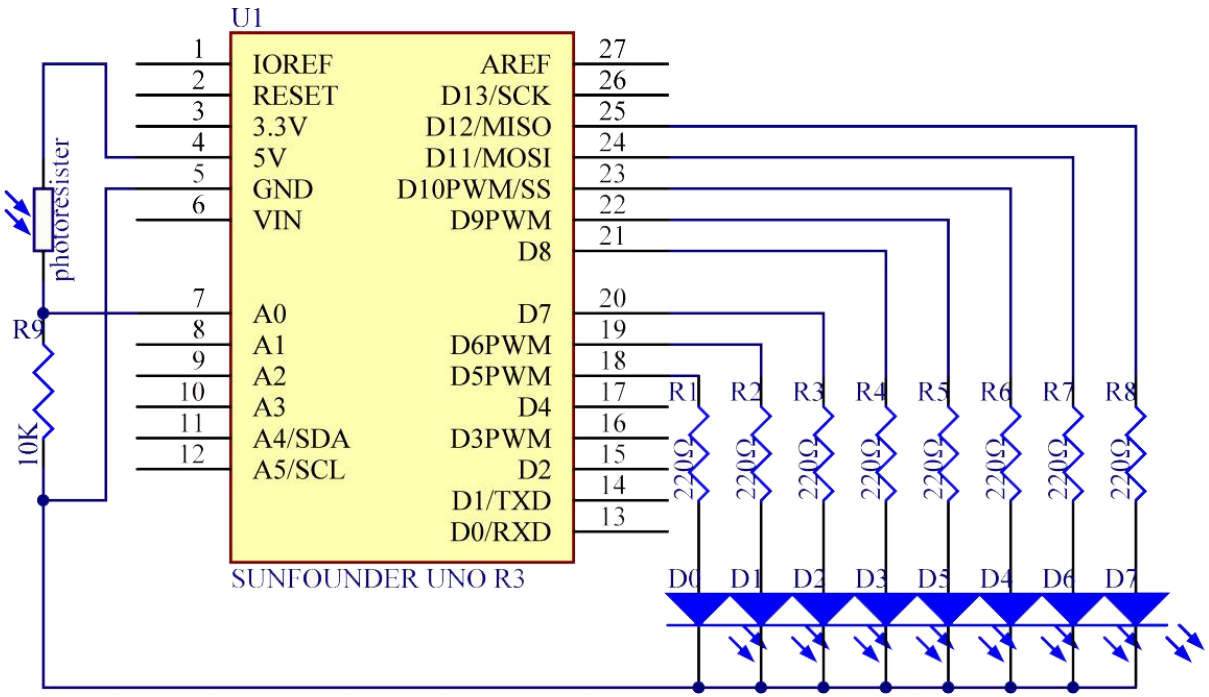
1 * Photo resistor 	8 * LED 	8 * Resistor (220Ω) 	1 * Resistor (10KΩ) 
1 * USB Cable 		Several Jumper Wires 	

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 LEDs to show the light intensity. The higher the light intensity is, the more LEDs will light up. When the light intensity is high enough, all the LEDs will be on. When there is no light, all the LEDs will go out.

The schematic diagram:



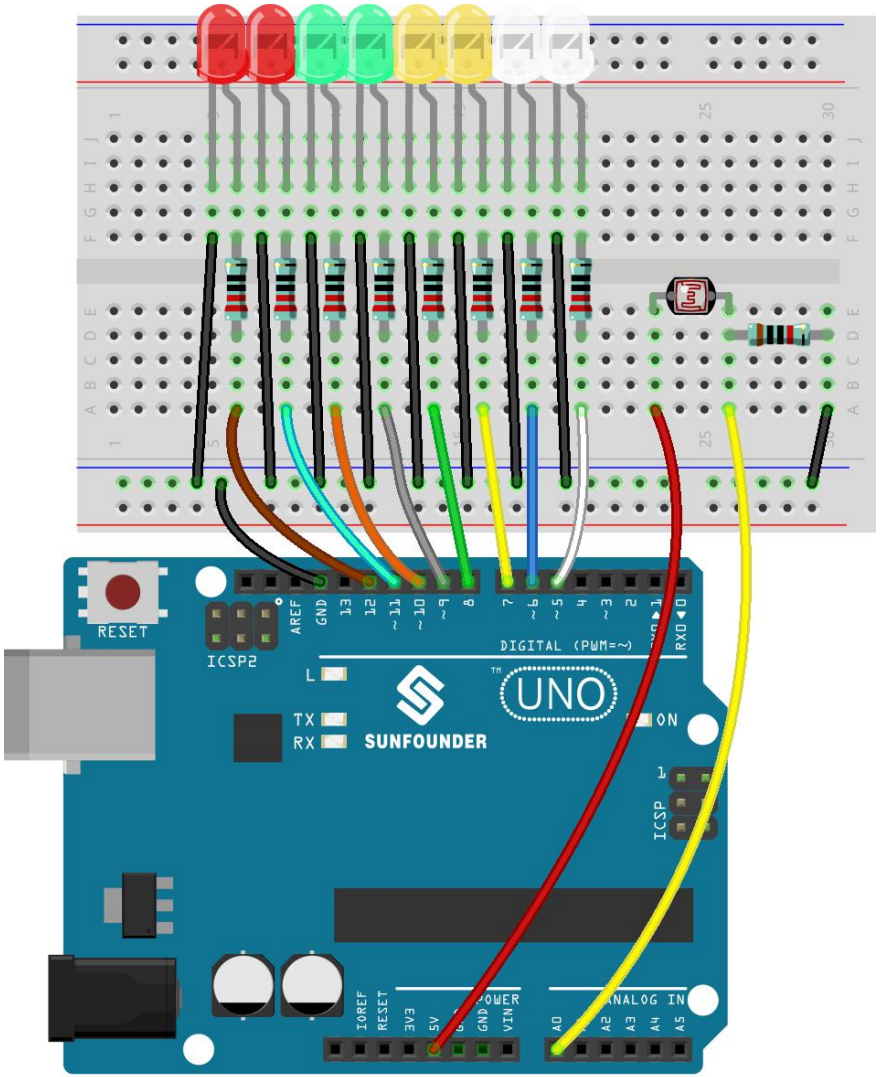
Experimental Procedures

Step 1: Build the circuit

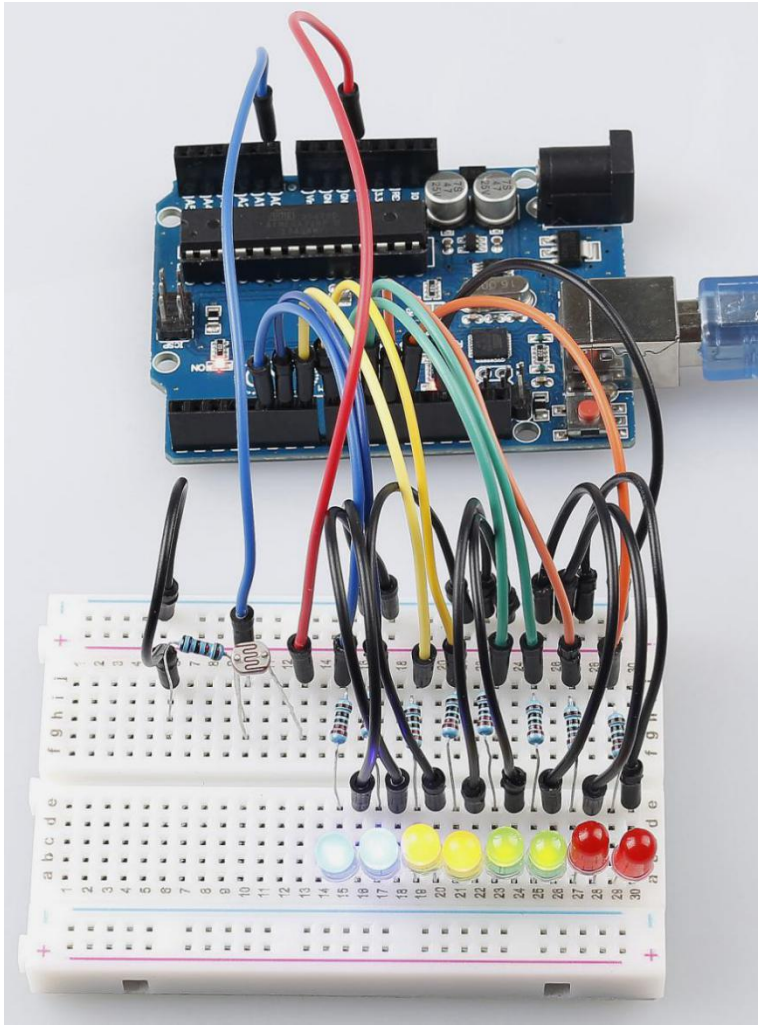
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.



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



Code Analysis

Code Analysis 6-1 Set the variables

```
const int NbrLEDs = 8;//8 leds

const int ledPins[] = {5, 6, 7, 8, 9, 10, 11, 12};//8 leds attach to pin 5-12 respectively

const int photocellPin = A0;  //photoresistor attach to A0

int sensorValue = 0;        // value read from the sensor

int ledLevel = 0;           // sensor value converted into LED 'bars'
```

The 8 LEDs 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 6-2 Set 8 pins to OUTPUT

```
for (int led = 0; led < NbrLEDs; led++)

  {

    pinMode(ledPins[led], OUTPUT);// make all the LED pins outputs

  }
```

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 6-3 Read the analog value of the photoresistor

```
sensorValue = analogRead(photocellPin); //read the value of A0
```

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); //Print the analog value of the photoresistor
```

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 6-4 Map the analog value to 8 LEDs

```
ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs);  // map to the number of LEDs
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	256-383.6	384-511.5	512-639.4	640-767.25	768-895.1	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 6-5 Light up the LEDs

```
for (int led = 0; led < NbrLEDs; led++)
  {
    if (led <= ledLevel ) //When led is smaller than ledLevel, run the following code.
    {
      digitalWrite(ledPins[led], HIGH);     // turn on pins less than the level
    }
    else
    {
      digitalWrite(ledPins[led], LOW);     // turn off pins higher than
    }
  }
```

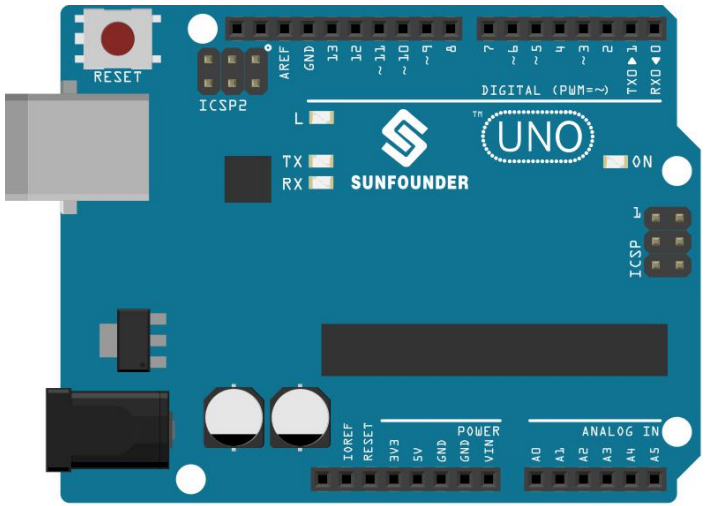
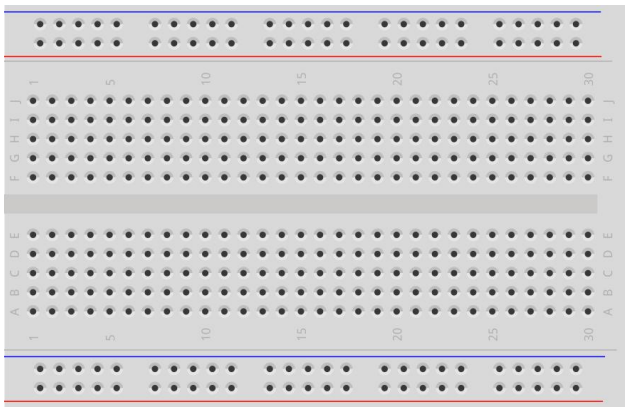
Light up the corresponding LEDs. Such as, when the ledLevel is 4, then light up the ledPins[0] to ledPins[4] and go out the ledPins[5] to ledPins[7].

Lesson 7 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 * Uno Board 	1 * Breadboard 

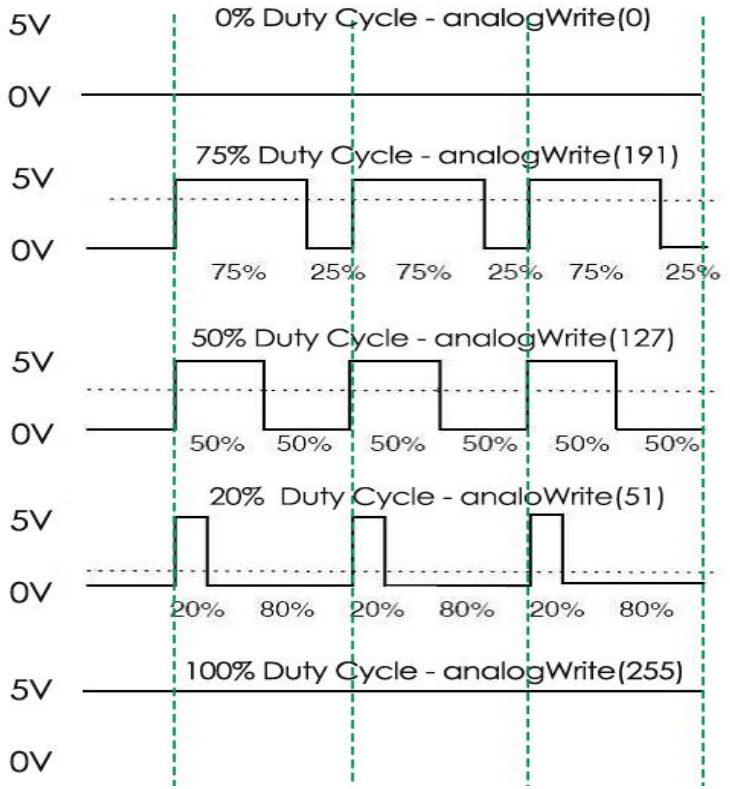
1 * RGB LED 	3 * Resistor (220Ω) 
1 * USB Cable 	Several Jumper Wires 

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.

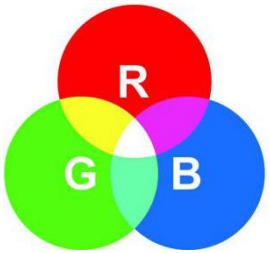


A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

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.

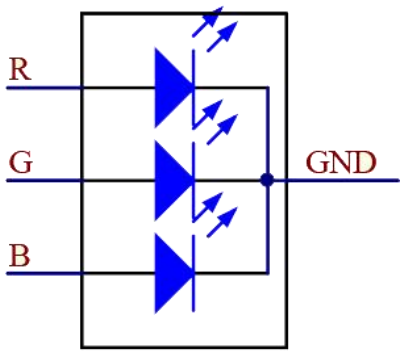
RGB LED

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.



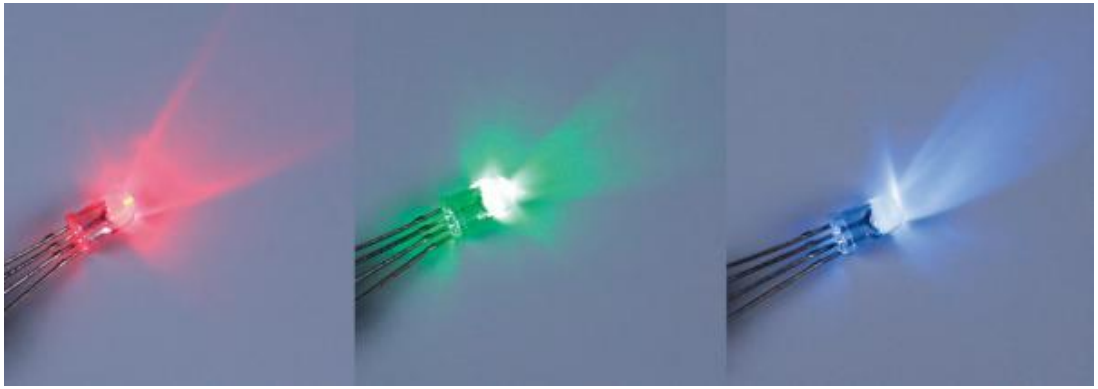
RGB LEDs can be categorized into common anode and common cathode ones. In this experiment, the latter is used. The common cathode, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

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.

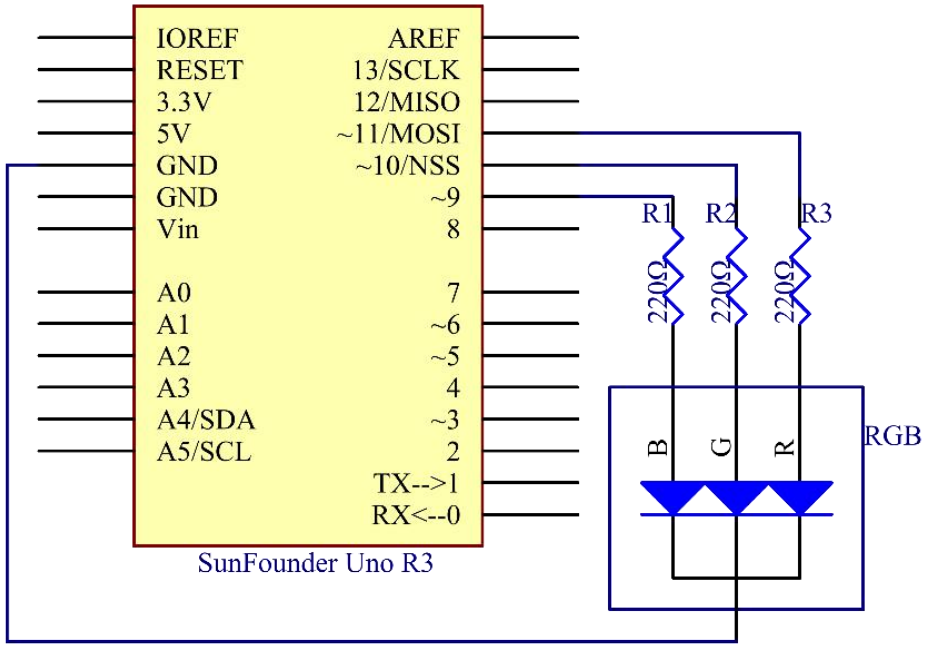


Or you can distinguish them in another way. As GND is the longest one and can be defined directly, you just need to confirm the other three pins. You can test it by giving them a small voltage. The forward voltage drop from the three pins to the GND are respectively 1.8V (red), 2.5V (blue), and 2.3V (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 Uno board, 3, 5, 6 and 9-11 is the PWM pins. 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 Uno. When the three pins are given different PWM values, the RGB LED will display different colors.

The schematic diagram:



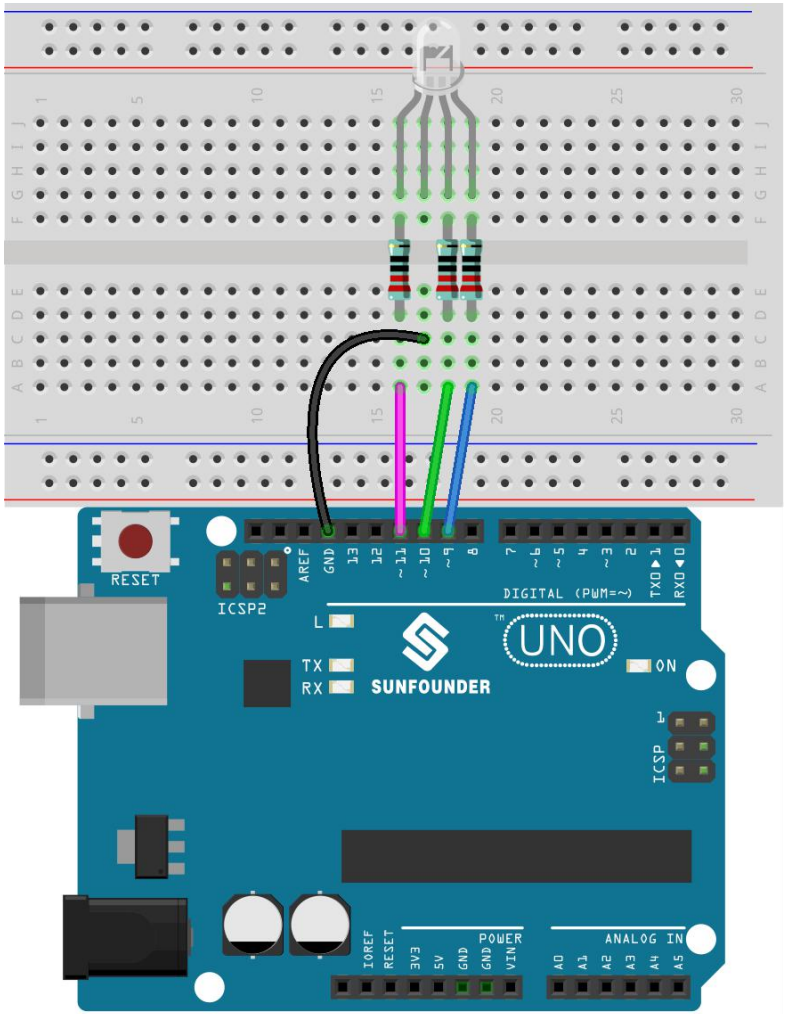
Experimental Procedures

Step 1: Build the circuit

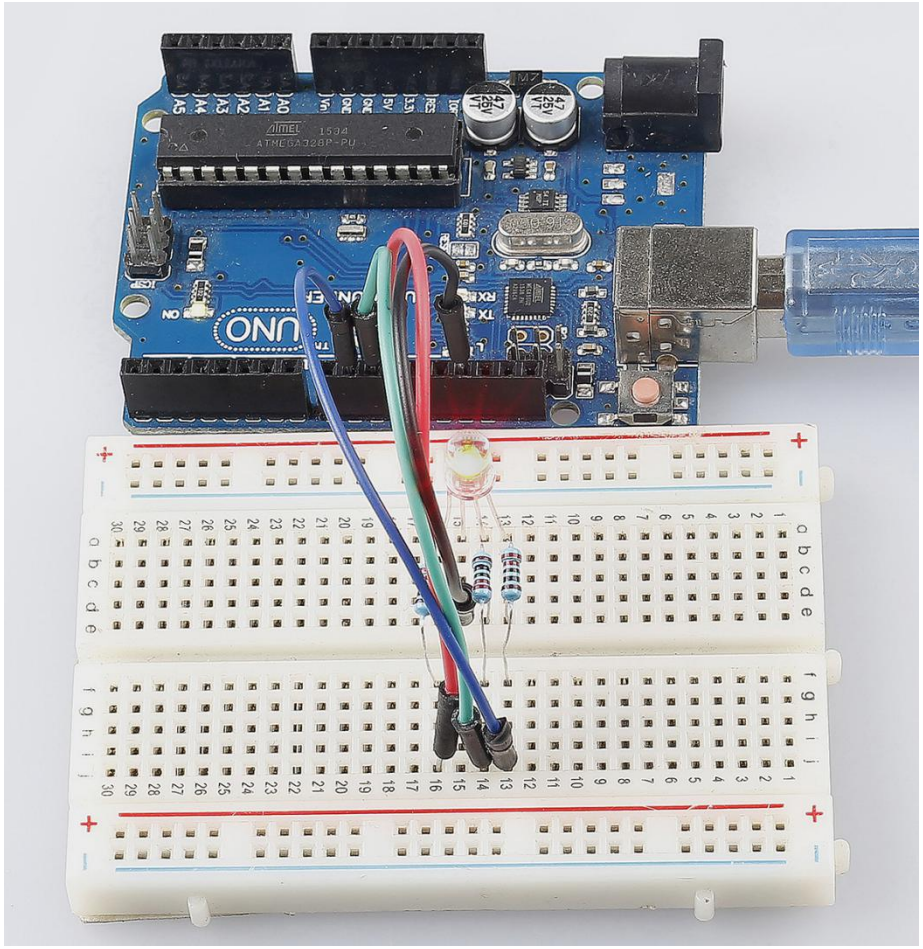
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.



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



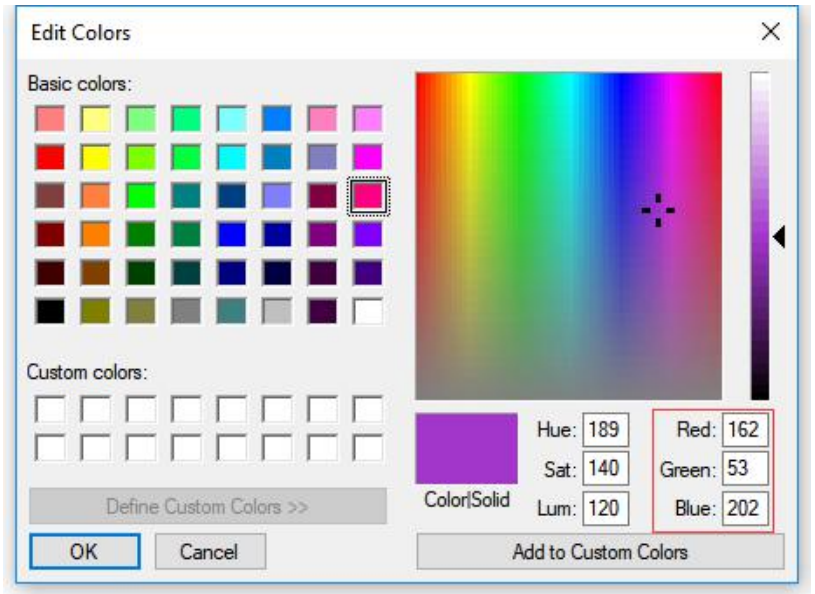
Code Analysis

Code Analysis 7-1 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.



```
void loop()  // run over and over again
{
  // Basic colors:
  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 7-2 color()function

```
void color (unsigned char red, unsigned char green, unsigned char blue)// the color generating function
{
  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.

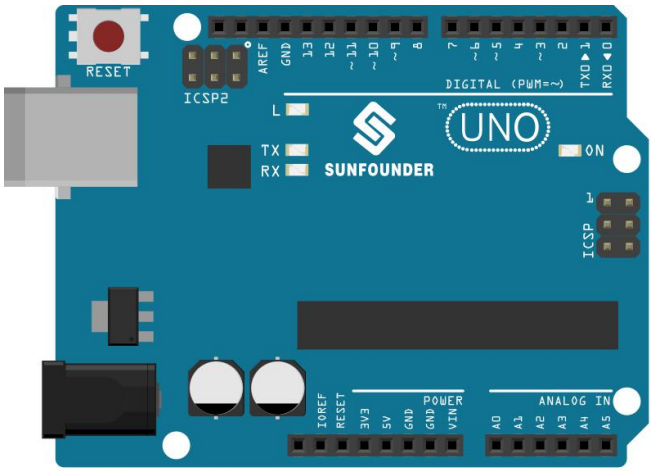
analogWrite(): Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *pinMode()* to set the pin as output before calling *analogWrite()*.

Lesson 8 Tilt Switch

Introduction

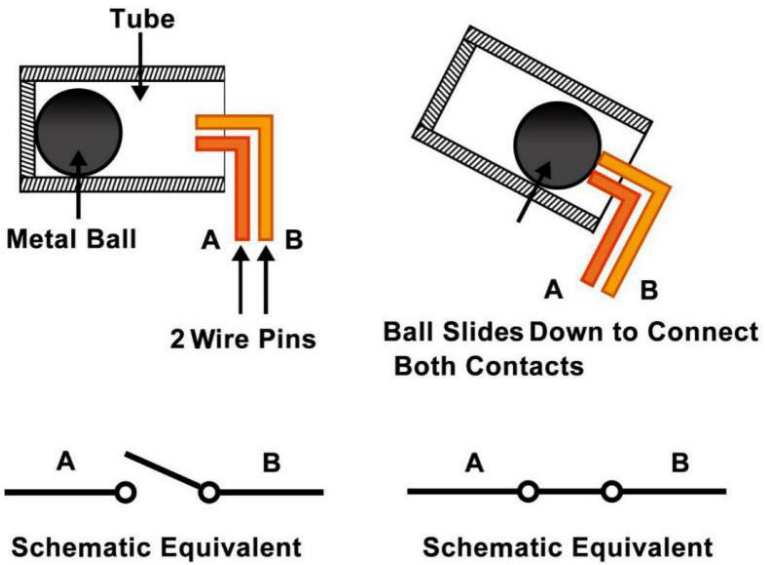
The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

Components

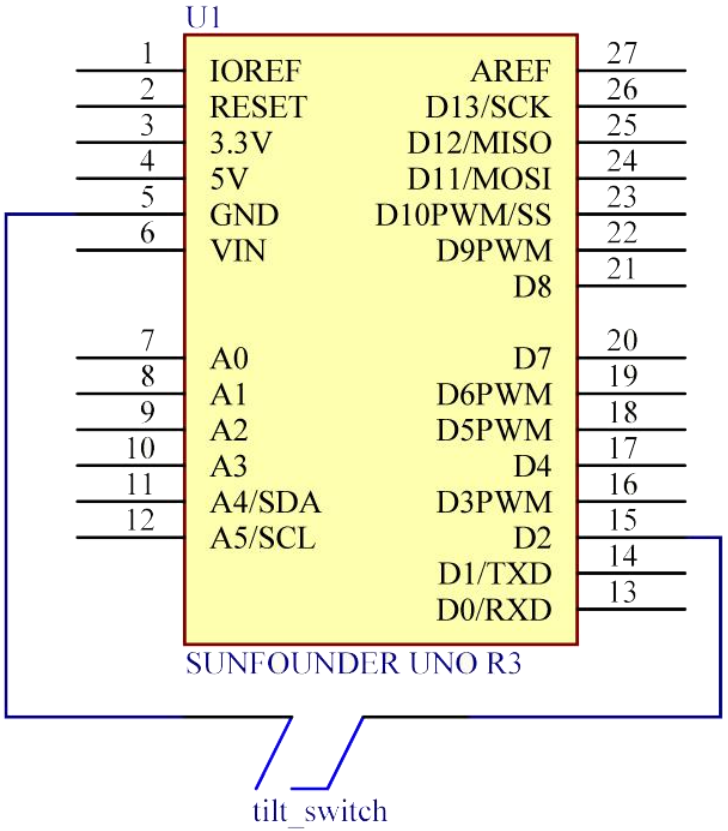
1 * Uno Board 	1*Tilt Switch 
1 * USB Cable 	2 * Dupont wires 

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.

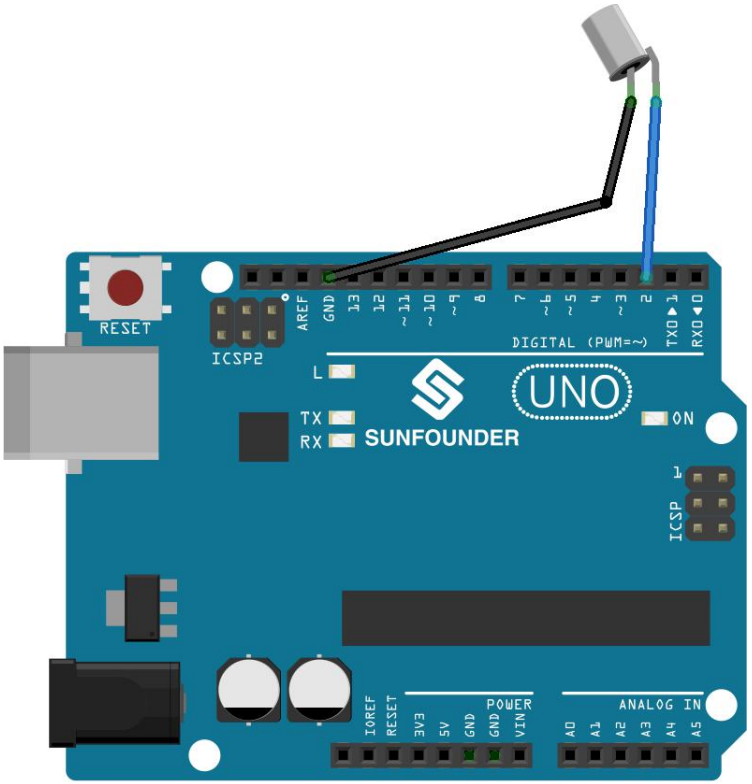


The schematic diagram:



Experimental Procedures

Step 1: Build the circuit

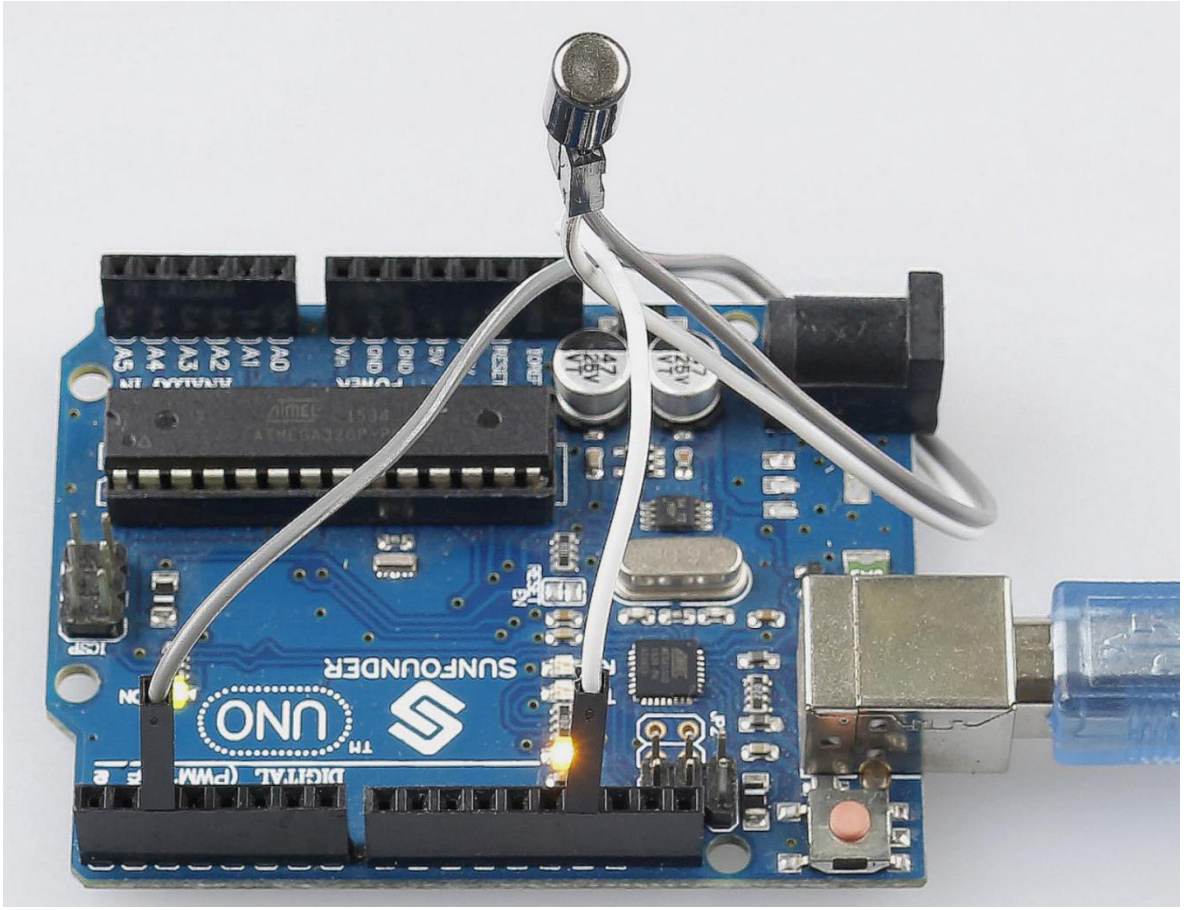


Step 2: Open the code file.

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 Uno board will light up.



Code Analysis

Code Analysis 8-1 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
  }
}
```

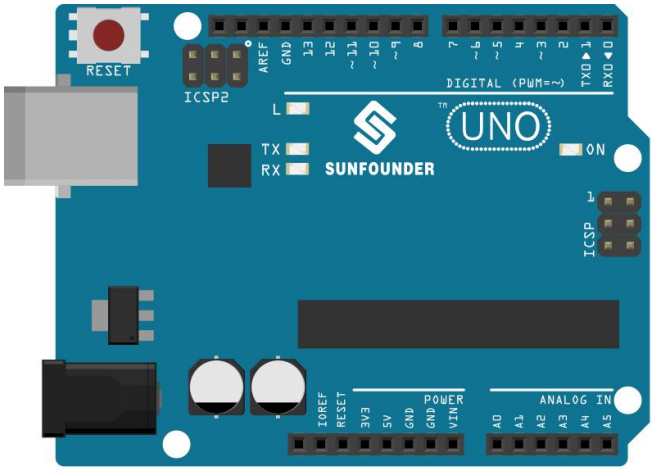
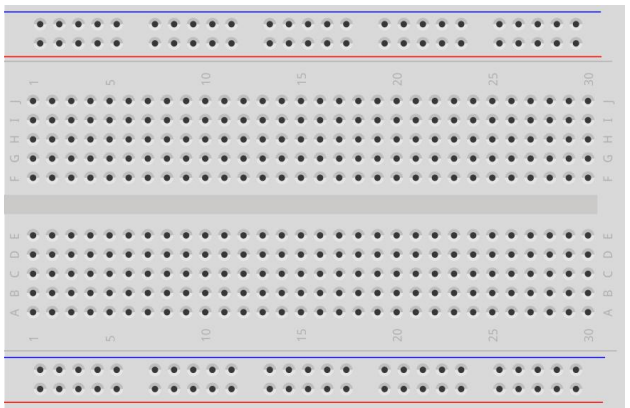
The whole code are very simple, one pin of the tilt switch is connected to pin2, another pin is connected to GND, when tilt the switch, the two pins of the switch will be connected to GND, then let the LED on the pin13 lights up.

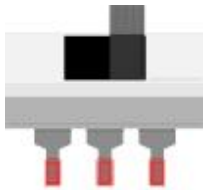
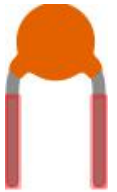
Lesson 9 Slide Switch

Introduction

In this lesson, we are going to use a slide switch to turn on/off an external LED. The slide switch is a device to connect or disconnect the circuit by sliding its handle. They are quite common in our surroundings. Now let's see how it works.

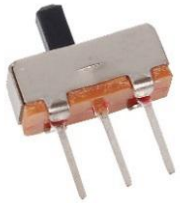
Components

1 * Uno Board	1 * Breadboard
	

<table>
<tr>
<td>1 * Resistor (220Ω)
 
1*Resistor(10kΩ)
  </td>
<td>1 * LED
  </td>
<td>1 *Slide Switch
  </td>
<td>1 * Capacitor 104
  </td>
</tr>
<tr>
<td colspan="2">1 * USB Cable
  </td>
<td colspan="2">Several Jumper Wires
  </td>
</tr>
</table>

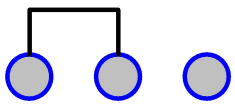
Experimental Principle

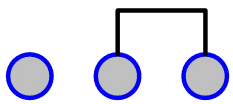
Slide Switch



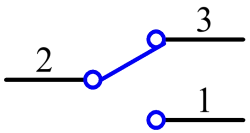
Just as its name suggests, slide switch is to connect or disconnect the circuit by sliding its switch handle so as to switch the circuit. The common types of slide switch include single pole double throw, single pole triple throw, double pole double throw, and double pole triple throw and so on. Generally, it is used in circuits with a low voltage and features flexibility and stabilization. Slide switches are commonly used in all kinds of instruments/meters equipment, electronic toys and other fields related.

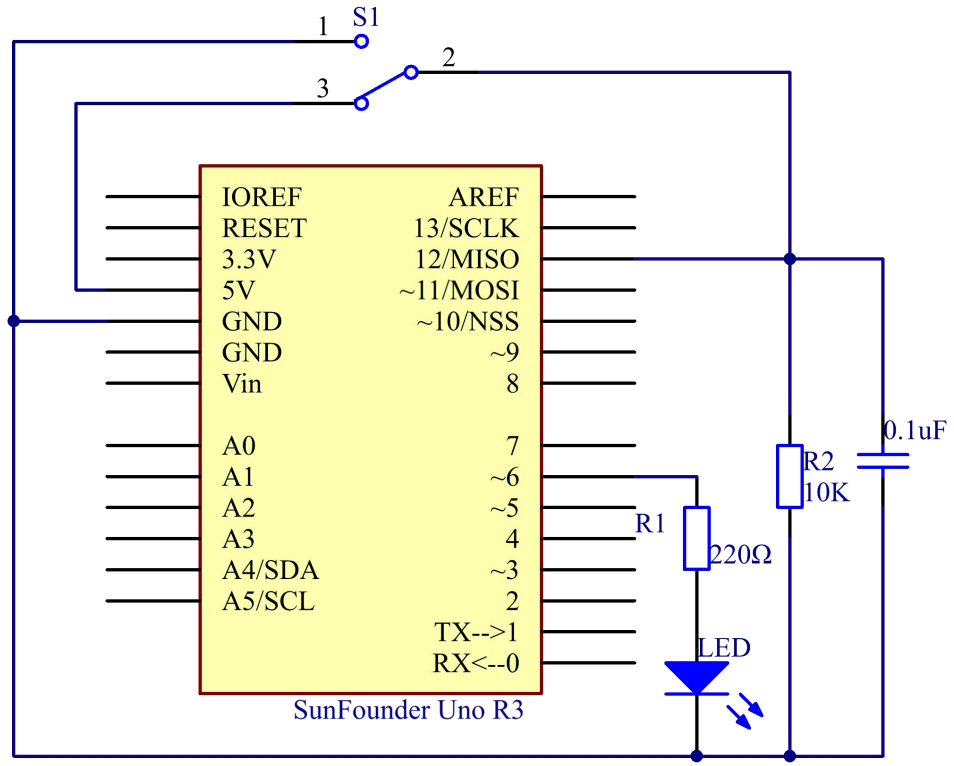
How it works: The middle pin is fixed. When the handle is pushed to the left, the left two pins are connected; push it to the right, the two pins on the right connect, thus switching circuits.





See the circuit symbol for slide switch and 2 is the middle pin.



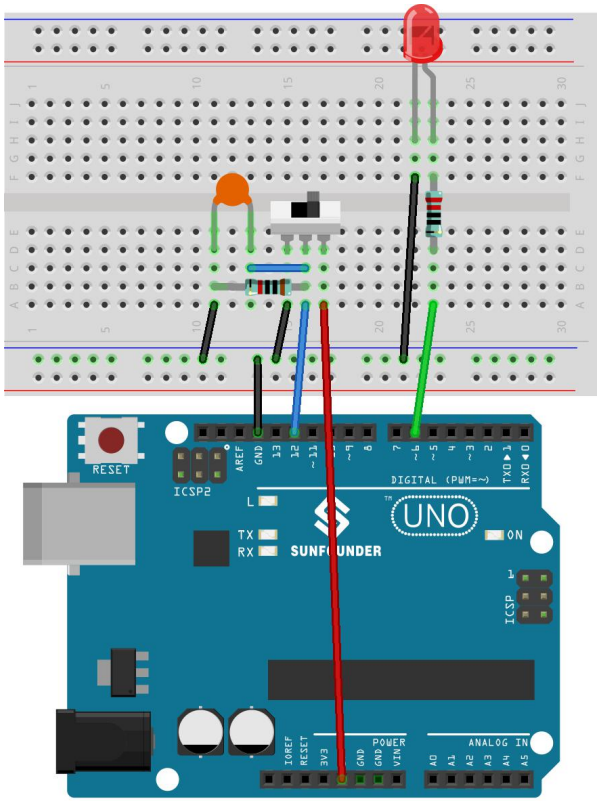


Principle:

Here we use a slide switch to control the on/off of an LED which is simple. Connect the middle pin of the switch to VCC. Connect one pin at one end to pin 12. After connecting a 10K resistor and a 104 capacitor, connect it to GND (to let the switch output stable level signal). Connect an LED to pin 6. Push the handle of the slide switch to the pin connected with pin 12 which is High level, we can light up the LED at pin 6 by programming.

Experimental Procedures

Step 1: Build the circuit

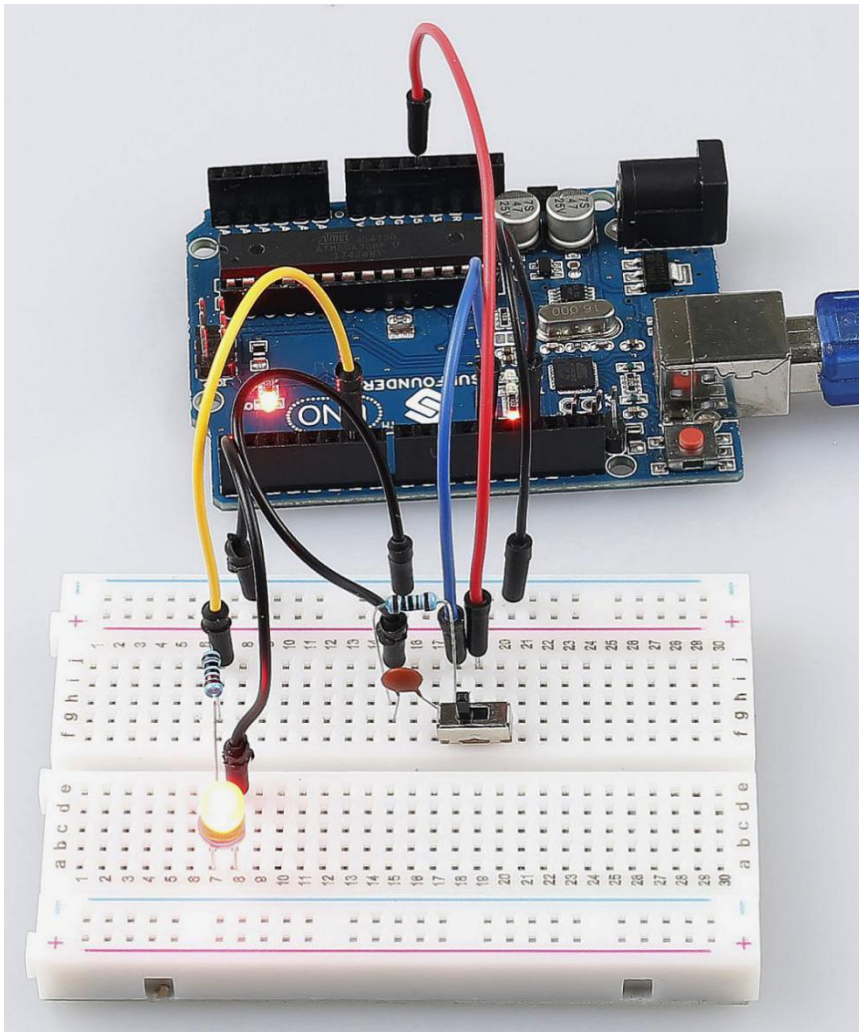


Step 2: Open the code file

Step 3: Select correct Board and Port

Step 4: Upload the sketch to the SunFounder Uno board

When you toggle the switch to pin12, the LED lights.



Code Analysis

Code Analysis 9-1 Read the switch state to turn on/off the LED

```
void loop()
{
  //read the state of the switch value
  switchState = digitalRead(switchPin);
  if (switchState == HIGH ) //if it is,the state is HIGH
  {
    digitalWrite(ledPin, HIGH); //turn the led on
  }
  else
  {
    digitalWrite(ledPin, LOW); //turn the led off
  }
}
```

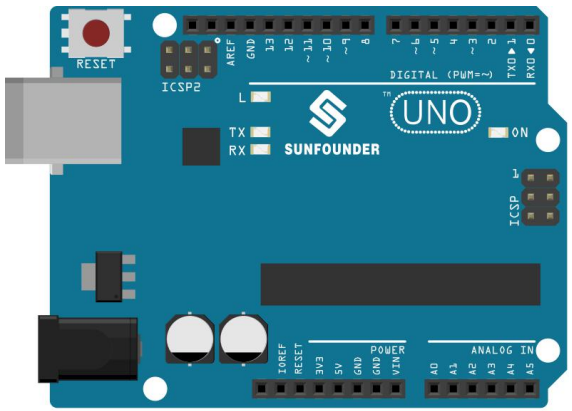
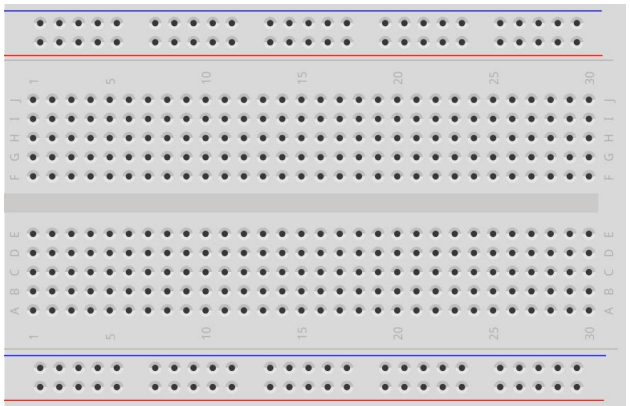
First, read the state of the *switchPin* and see whether you have moved the switch handle. If it has been pushed to pin 12, then the *switchState* is High level, so set *ledPin* as High level, which means to light up the LED; otherwise, to turn it off.

Lesson 10 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 * Uno Board 	1 * Breadboard 

1*Resistor(1kΩ) 	1 * Resistor (220Ω) 	1 * Diode1N4007 (Rectifier) 
1 * LED 	1 * NPN S8050 	1* Relay 
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.

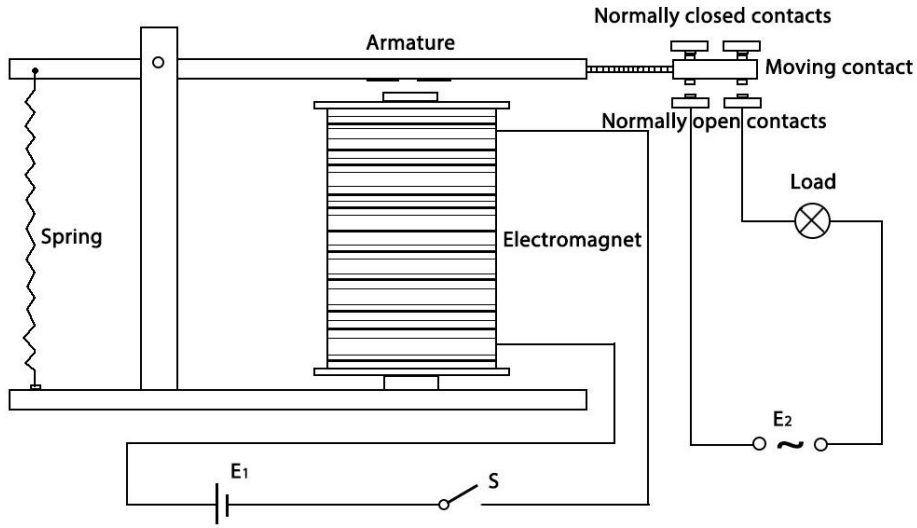
. Normally close – not connected when the relay is activated, and connected when it is inactive.

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

Working of Relay

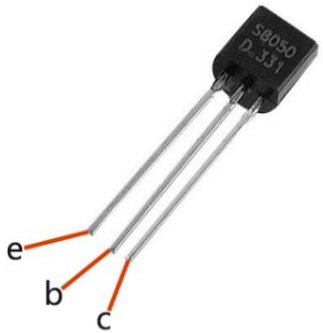
The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open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

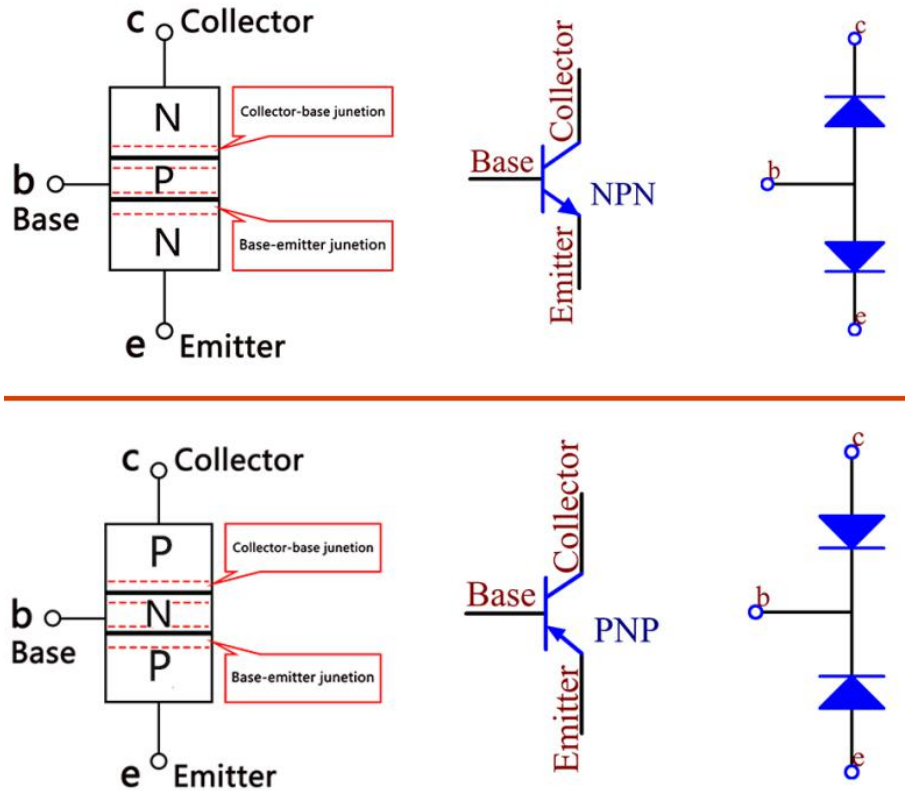
Transistor is a semiconductor device that amplifying weak signal to larger amplitude transistor is a three-layer structure They form the three regions internally. The two are both N-type or P-type ones – the emitter region, when the other one is the transistor to be an amplifier.



controls current by current. It functions by signal and is also used for non-contact switch. A composed of P-type and N-type semiconductors. thinner in the middle is the base region; the other smaller region with intense majority carriers is the collector region. This composition enables the

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.

Principle:

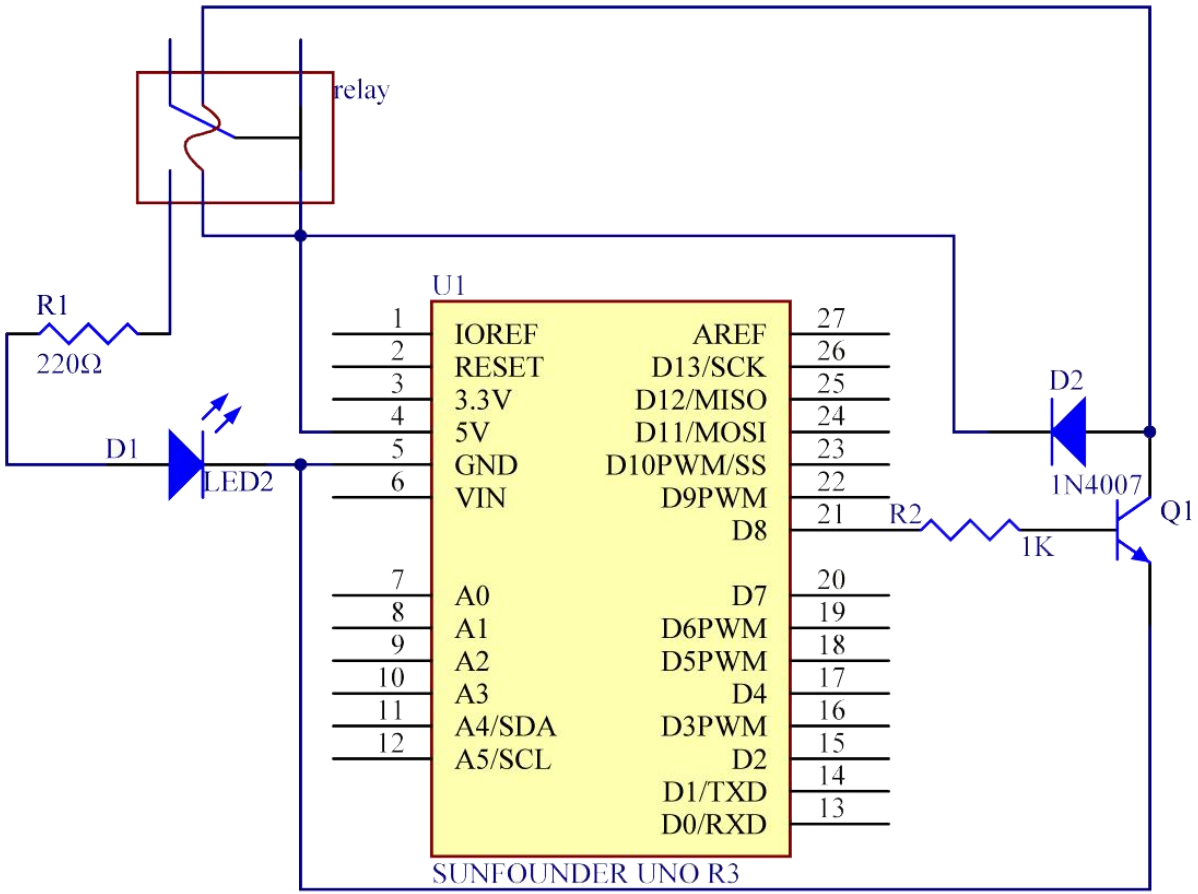
Connect a 1K resistor (for current limiting when the transistor is energized) to pin 8 of the SunFounder Uno board, then to an NPN transistor whose collector is connected to the coil of a relay and emitter to GND; connect the normally open contact of the relay to an LED and then GND. Therefore, when 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.

The schematic diagram:



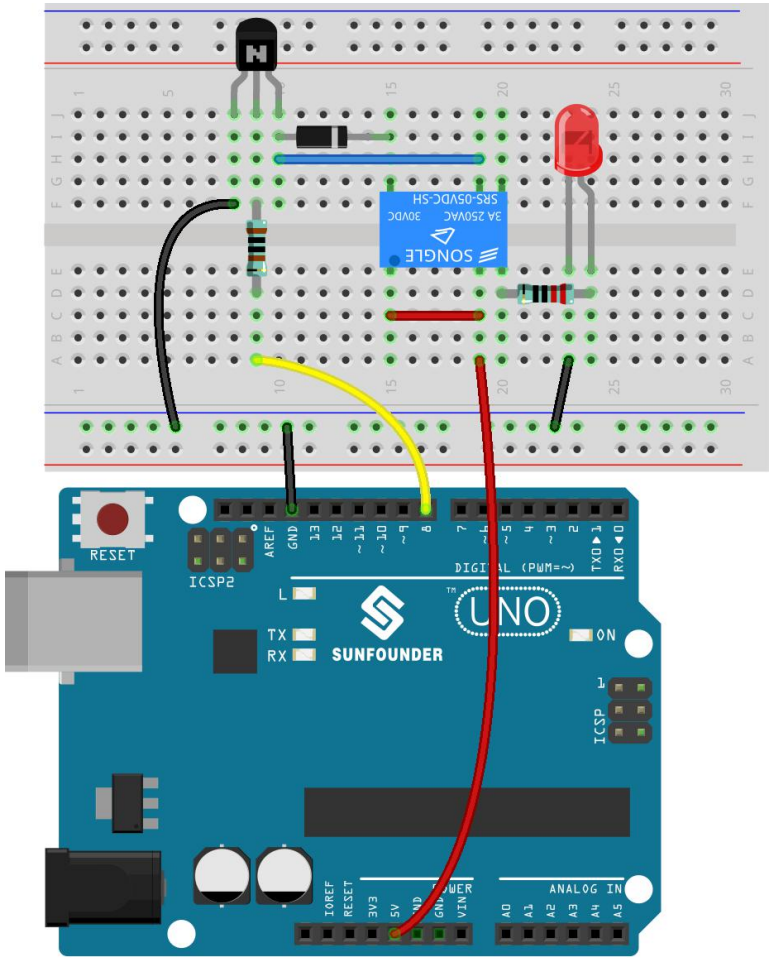
Experimental Procedures

Step 1: Build the circuit

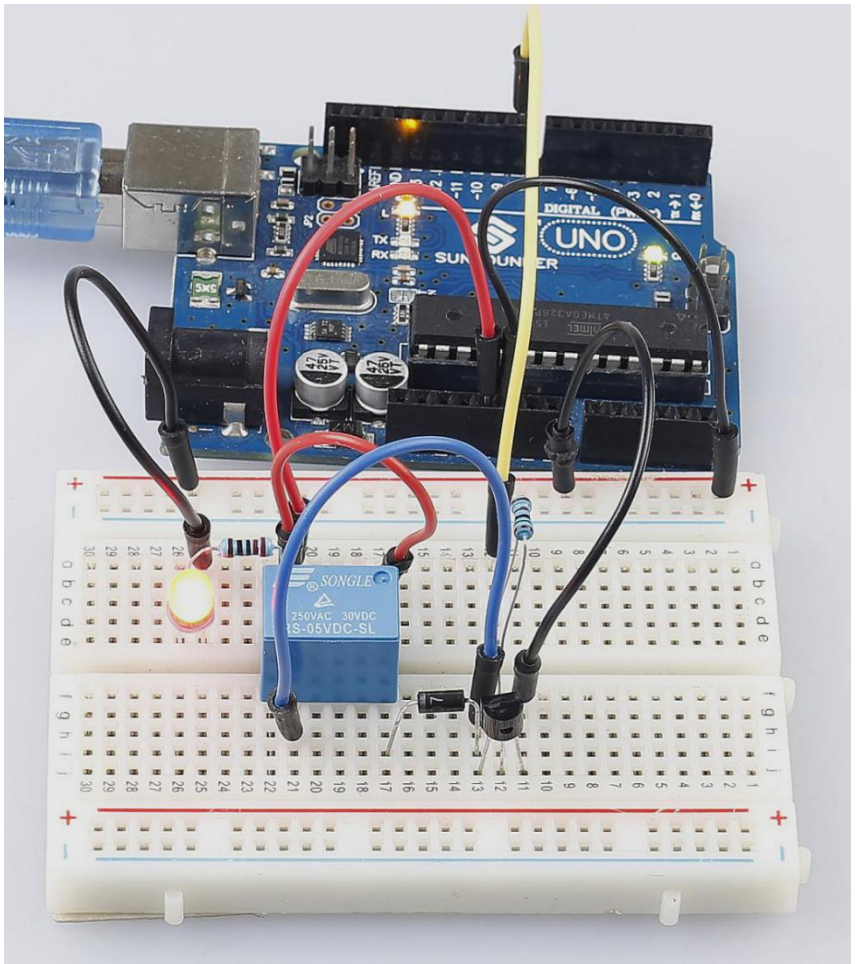
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.



Now, 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

}
```

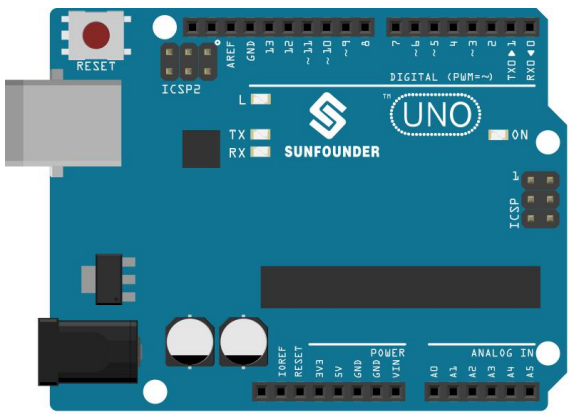
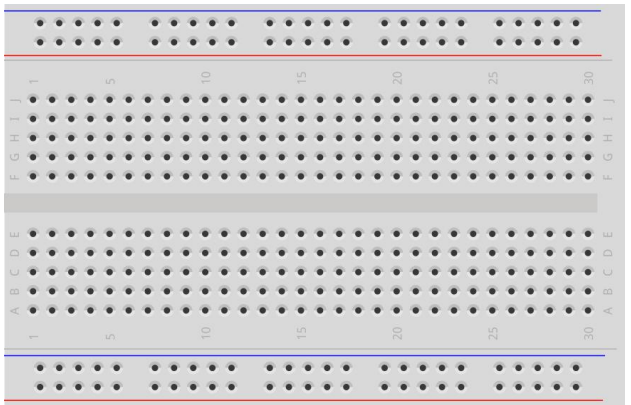
The code in this experiment is simple. First, set relayPin as HIGH level and the LED connected to therelay will light up. Then set relayPin as LOW level and the LED goes out.

Lesson 11 4N35

Introduction

The 4N35 is an optocoupler that consists of a gallium arsenide infrared LED and a silicon NPN phototransistor. When the input signal is applied to the LED in the input terminal, the LED lights up. After receiving the light signal, the light receiver then converts it into electrical signal and outputs the signal directly or after amplifying it into a standard digital level. Thus, the transition and transmission of electricity-light-electricity is completed. Since light is the media of the transmission, meaning the input terminal and the output one are isolated electrically, this process is also be known as electrical isolation.

Components

1 * Uno Board 	1 * Breadboard 

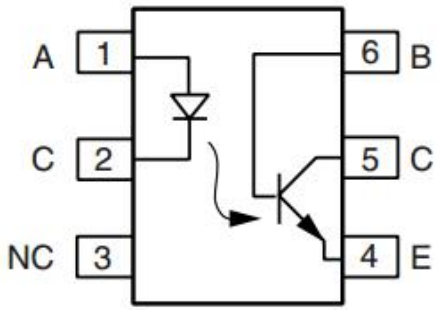
1*4N35 	1 * Resistor (220Ω) 	1 * LED 
1 * USB Cable 	Several Jumper Wires 	

Experimental Principle

4N35

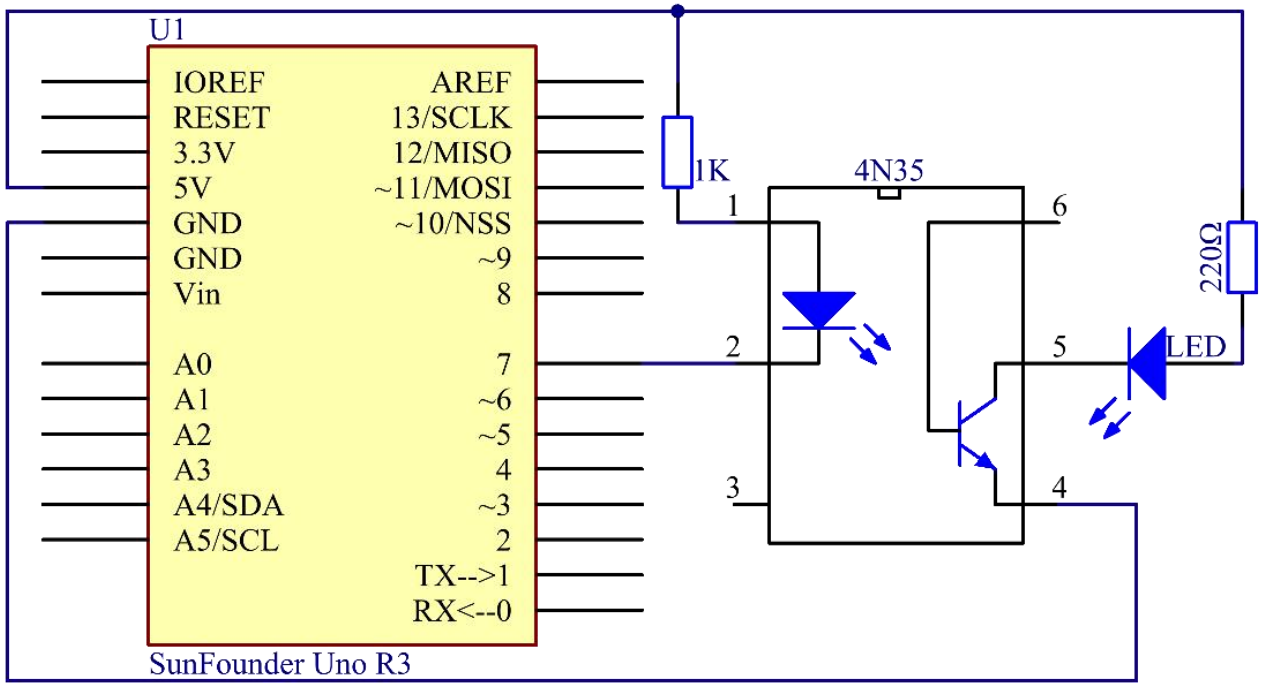
The 4N35 is an optocoupler for general purpose application. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor.

What an optocoupler does is to break the connection between signal source and signal receiver, so as to stop electrical interference. In other words, it is used to prevent interference from external electrical signals. 4N35 can be used in AV conversion audio circuits. Broadly it is widely used in electrical isolation for a general optocoupler.



See the internal structure of the 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays. This can be done to control the load connected to the phototransistor. Even when the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

The schematic diagram:



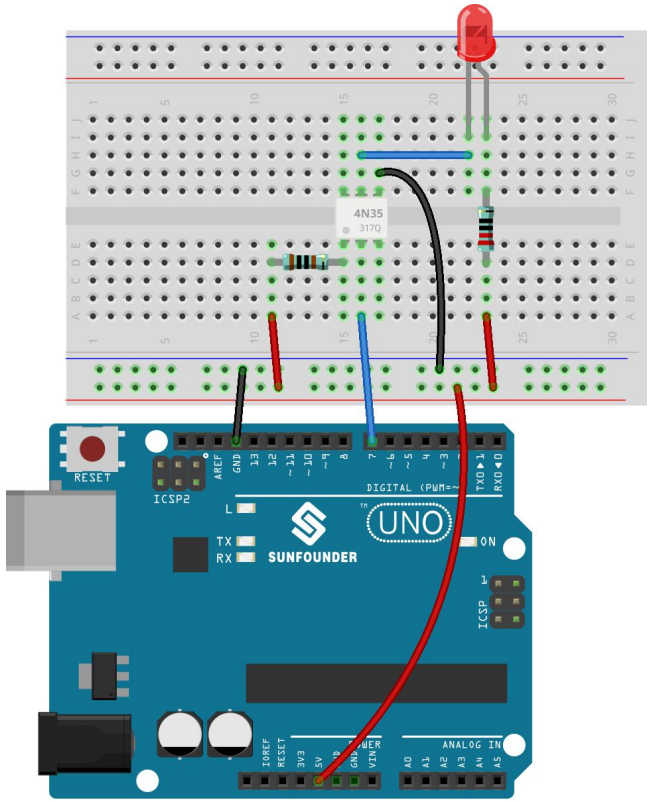
Principle:

In this experiment, use an LED as the load connected to the NPN phototransistor. Connect pin 2 of the 4N35 to pin 7 of the control board, and pin 1 to a 1K current limiting resistor and then to 5V. Connect pin 4 to GND of the Uno, and pin 5 to the cathode of the LED. Then hook the anode of the LED to 5V after connecting with a 220 Ohm resistor. When in program, a LOW level is given to pin 7, the infrared LED will emit infrared rays. Then the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus

turning on the LED. Also you can control the LED by circuits only – connect pin 2 to ground and it will brighten.

Experimental Procedures

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

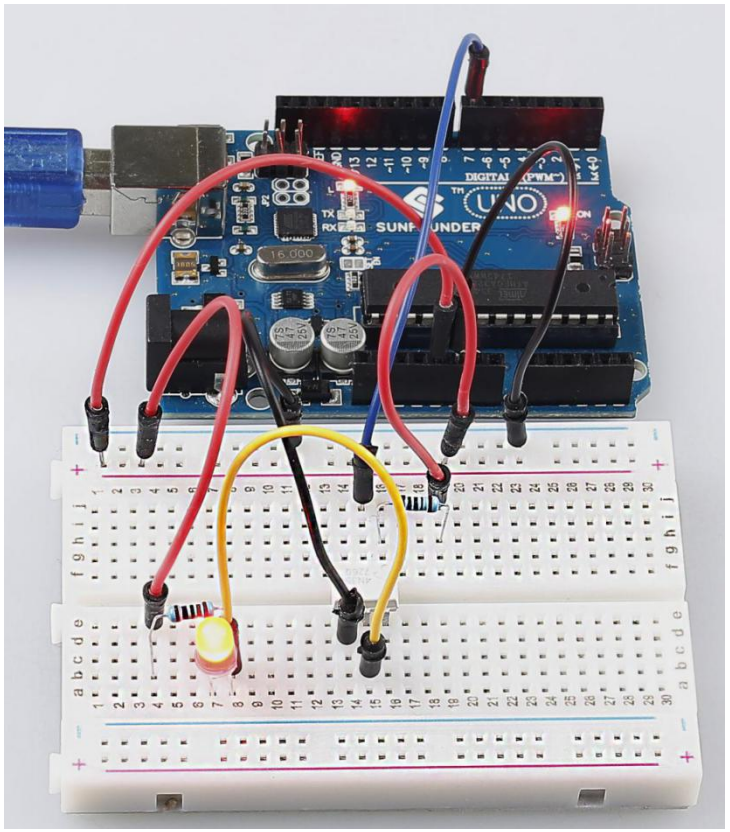


Step 2: Open the code file.

Step 3: Select correct Board and Port.

Step 4: Upload the sketch to the SunFounder Uno board.

You will see the LED blinks.



Exploration

4N35 is usually used for driving relay as well as motor circuits. As there is no direct connection between the input and output, even if a short circuit at the output end occurs, the control board will not be burnt. Have a try!

Code Analysis

```
void loop()

{

  digitalWrite(OptoPin, LOW);  //set the OptoPin as LOW level,then the led connected on the output of 4n35 will be light

  delay(500);  //delay 500ms

  digitalWrite(OptoPin, HIGH);  //turn off the led

  delay(500);  //delay 500ms

}
```

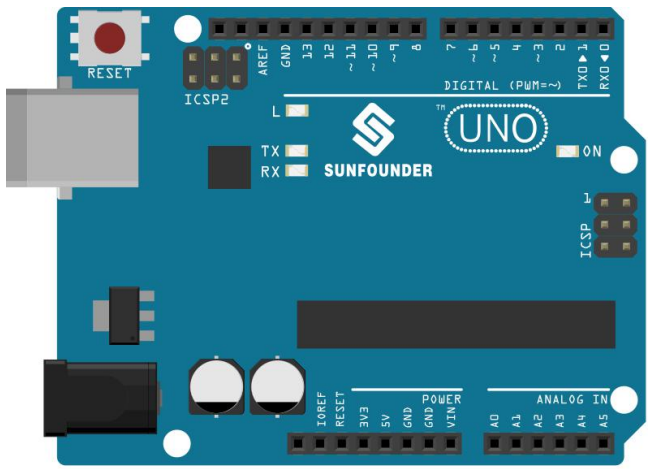
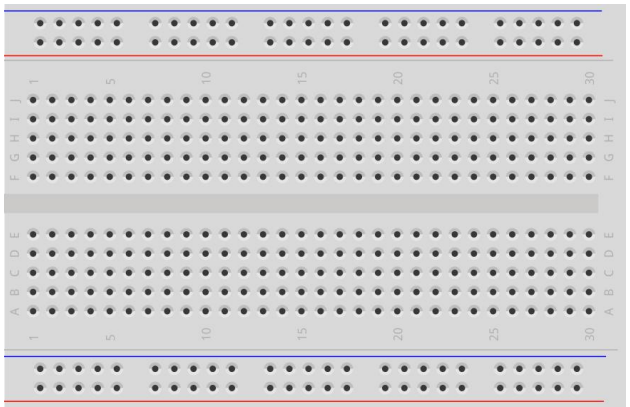
The code in this experiment is very easy to understand. Set pin 7 as Low level and the LED will light up; set it as High, and the LED goes out.

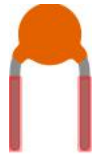
Lesson 12 NE555 Timer

Introduction

The NE555 Timer, a mixed circuit composed of analog and digital circuits, integrates analog and logical functions into an independent IC, thus tremendously expanding the applications of analog integrated circuits. It is widely used in various timers, pulse generators, and oscillators. In this experiment, the SunFounder Uno board is used to test the frequencies of square waves generated by the 555 oscillating circuit and show them on Serial Monitor.

Components

1 * Uno Board 	1 * Breadboard 

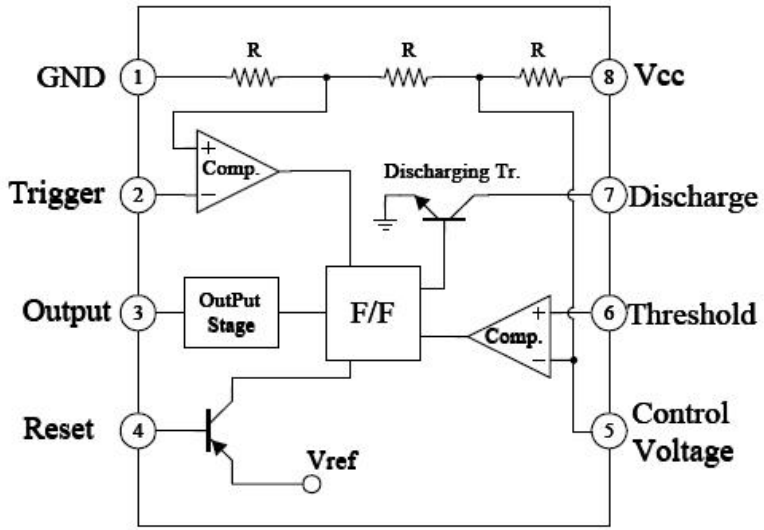
1 * Potentiometer (10KΩ) 	2 * 104 ceramic capacitor 	1 * NE555 	1*Resistor(10kΩ) 
1 * USB Cable 		Several Jumper Wires 	

Experimental Principle

555 IC

The 555 IC was originally used as a timer, hence the name 555 time base circuit. It is now widely used in various electronic products because of its reliability, convenience, and low price. The 555 is a complex hybrid circuit with dozens of components such as a divider, comparator, basic R-S trigger, discharge tube, and buffer.

Its pins and their functions:

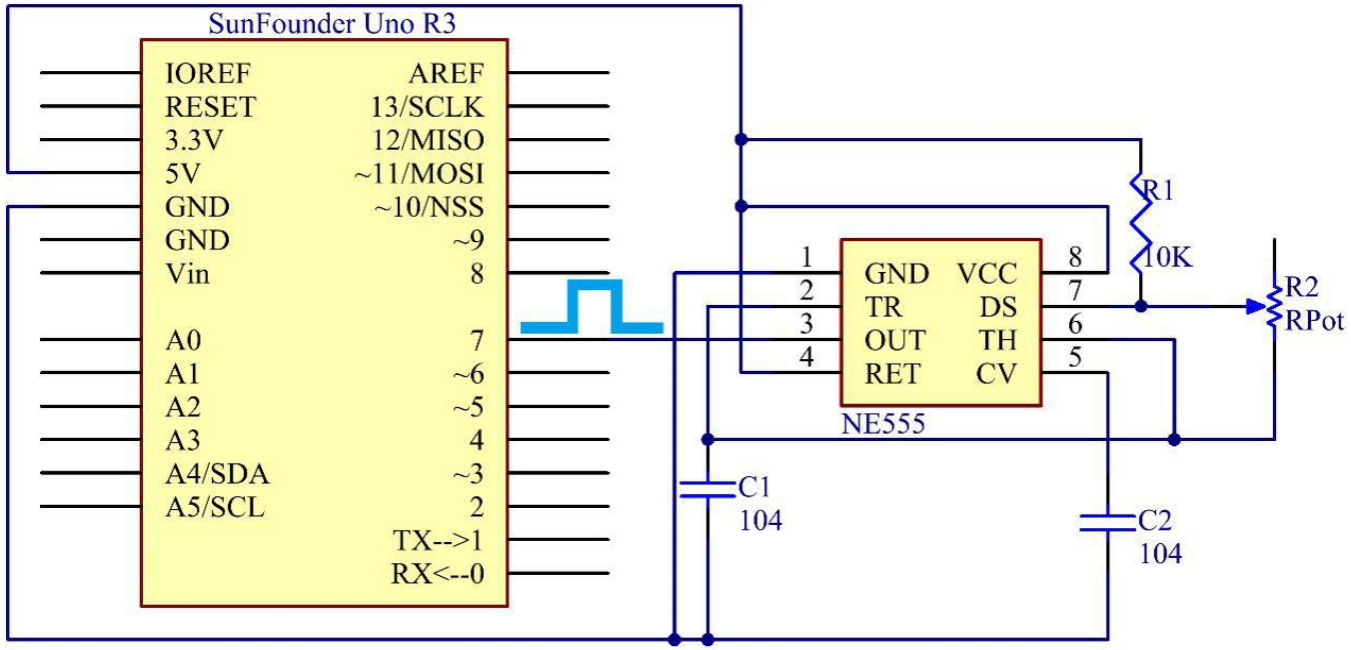


As shown in the picture, the pins are set dual in-line with the 8-pin package.

- Pin 1 (**GND**): the ground
- Pin 2 (**TRIGGER**): when the voltage at the pin reduces to 1/3 of the VCC (or the threshold defined by the control board), the output terminal sends out a High level

Pin 3 (**OUTPUT**): outputs High or Low, two states 0 and 1 decided by the input electrical level; maximum output current approx. 200mA at High

- Pin 4 (**RESET**): when a Low level is received at the pin, the timer will be reset and the output will return to Low level; usually connected to positive pole or neglected
- Pin 5 (**CONTROL VOLTAGE**): to control the threshold voltage of the chip (if it skips connection, by default, the threshold voltage is 1/3 VCC and 2/3 VCC)
- Pin 6 (**THRESHOLD)**: when the voltage at the pin increases to 2/3 VCC (or the threshold defined by the control board), the output terminal sends out a High level
- Pin 7 (**DISCHARGE)**: output synchronized with Pin 3, with the same logical level; but this pin does not output current, so pin 3 is the real High (or Low) when pin 7 is the virtual High (or Low); connected to the open collector (OC) inside to discharge the capacitor
- Pin 8 (**VCC**): positive terminal for the NE555 timer IC, ranging +4.5V to +16V
- The NE555 timer works under the monostable, astable and bistable modes. In this experiment, apply it under the astable mode, which means it works as an oscillator, as shown below:



Connect a resistor R1 between the VCC and the discharging pin DS, another resistor between pin DS and the trigger pin TR which is connected to the threshold pin TH and then to the capacitor C1. Connect the RET (pin 4) to GND, CV (pin 5) to another capacitor C2 and then to the ground.

Working process:

The oscillator starts to shake once the circuit is power on. Upon the energizing, since the voltage at C1 cannot change abruptly, which means pin 2 is Low level initially, set the timer to 1, so pin 3 is High level. The capacitor C1 charges via R1 and R2, in a time span:

$$Tc=0.693(R1+R2)$$

When the voltage at C1 reaches the threshold 2/3Vcc, the timer is reset and pin 3 is Low level. Then C1 discharges via R2 till 2/3Vcc, in a time span:

$$Td=0.693(R2)$$

Then the capacitor is recharged and the output voltage flips again:

$$\text{Duty cycle } D=Tc/(Tc+Td) \times 100\%$$

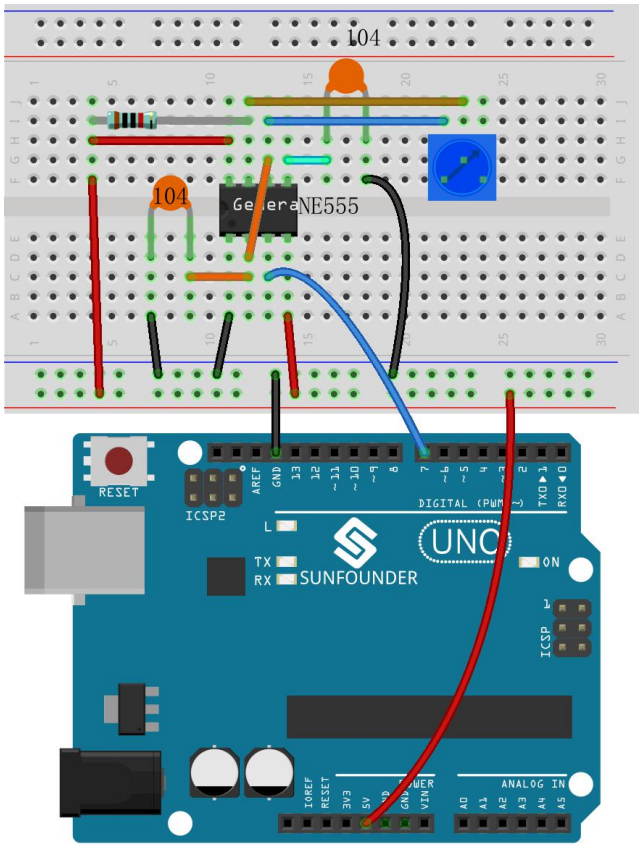
Since a potentiometer is used for resistor, we can output square wave signals with different duty cycles by adjusting its resistance. But R1 is a 10K resistor and R2 is 0k-10k, so the range of the ideal duty cycle is 66.7%-100%. If you want another else, you need to change the resistance of R1 and R2.

$$Dmin=(0.693(10K+0K))/(0.693(10K+0K)+0.693x0k) \times 100\%=100\%$$

$$Dmax=(0.693(10K+10K))/(0.693(10K+10K)+0.693x10k) \times 100\%=66.7\%$$

Experimental Procedures

Step 1: Build the circuit

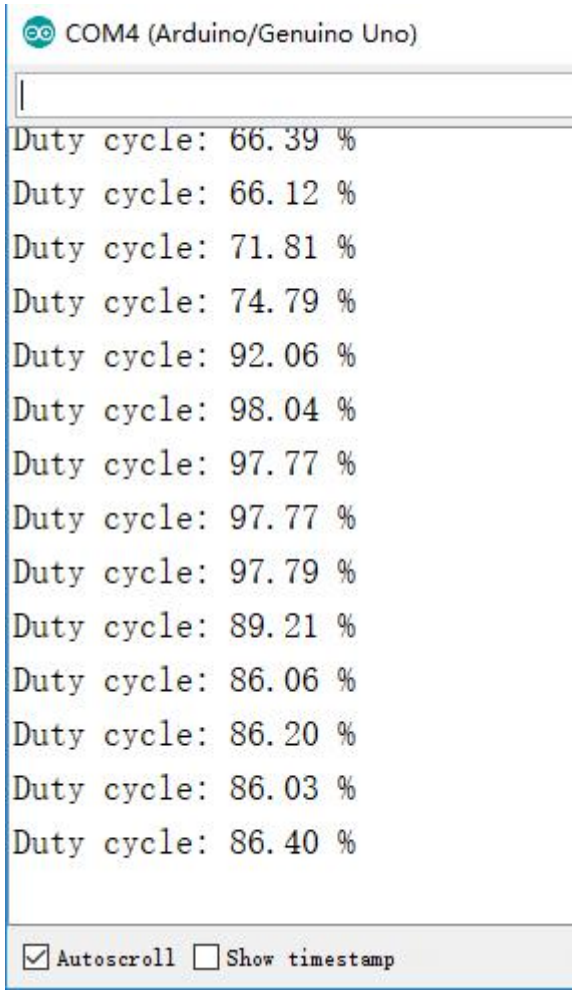


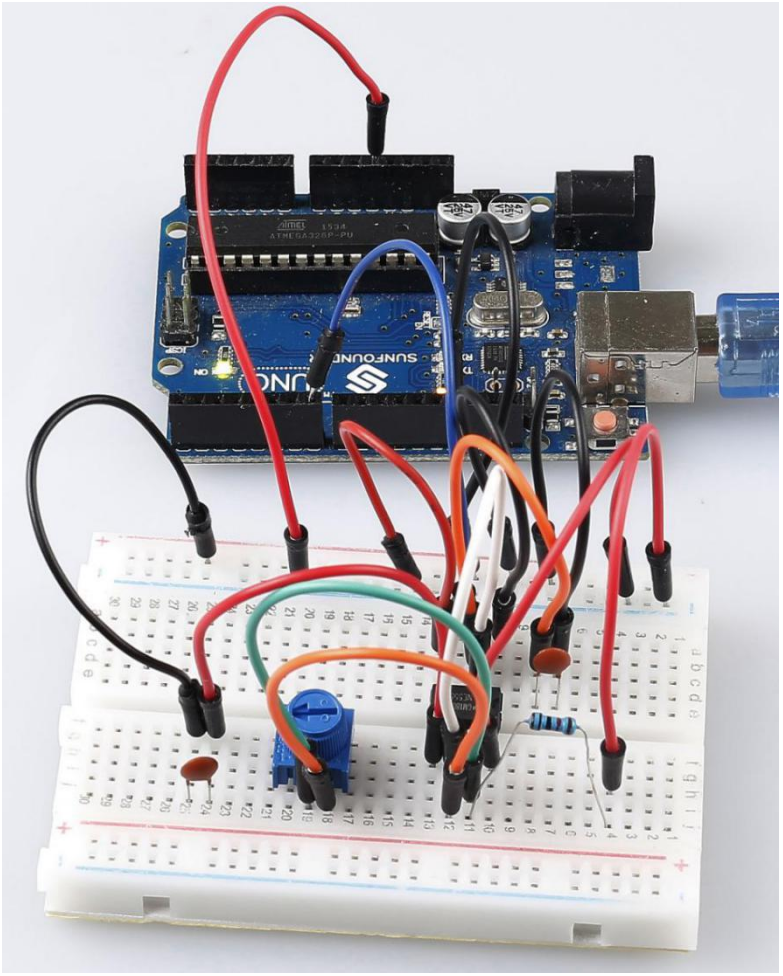
Step 2: Open the code file

Step 3: Select correct Board and Port

Step 4: Upload the sketch to the SunFounder Uno board

After uploading, open the Serial Monitor and you will see the following window.





Code Analysis

Code Analysis 12-1 Calculate the duty cycle

```
void loop()
{
  duration1 = pulseIn(ne555, HIGH);  //Reads a pulse on ne555
  duration2 = pulseIn(ne555, LOW);  //Reads a pulse on ne555
  dc = float (duration1) / (duration1 + duration2) * 100;
  Serial.print("Duty cycle: ");
  Serial.print(dc);  //print the length of the pulse on the serial monitor
  Serial.print(" %");
  Serial.println();  //print an blank on serial monitor
  delay(500);  //wait for 500 microseconds
}
```

Read a pulse waits for the ne555(pin 7) from HIGH to LOW firstly, then read a pulse waits for pin 7 from LOW to HIGH. so the range of the ideal duty cycle dc is `float` (duration1) / (duration1 + duration2) * 100;

You can rotate the potentiometer and read the duty cycle from the serial monitor.

pulseIn()

[Advanced I/O]

Description

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax

pulseIn(pin, value)

pulseIn(pin, value, timeout)

Parameters

pin: the number of the pin on which you want to read the pulse. (int)

value: type of pulse to read: either HIGH or LOW. (int)

timeout (optional): the number of microseconds to wait for the pulse to start; default is one second (unsigned long)

Returns

the length of the pulse (in microseconds) or 0 if no pulse started before the timeout (unsigned long)

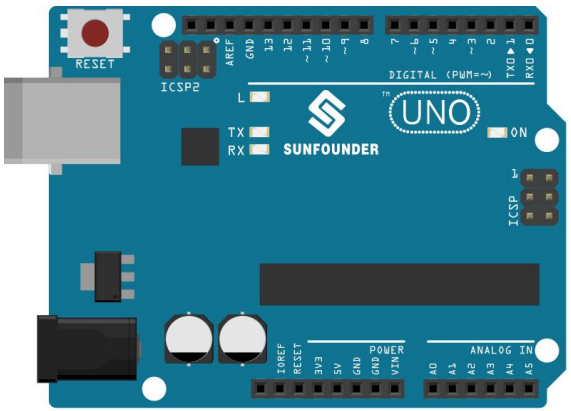
Lesson 13 Servo

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 * Uno Board 	1 * Servo 	1 * USB Cable 
	Several Jumper Wires(M to F) 	

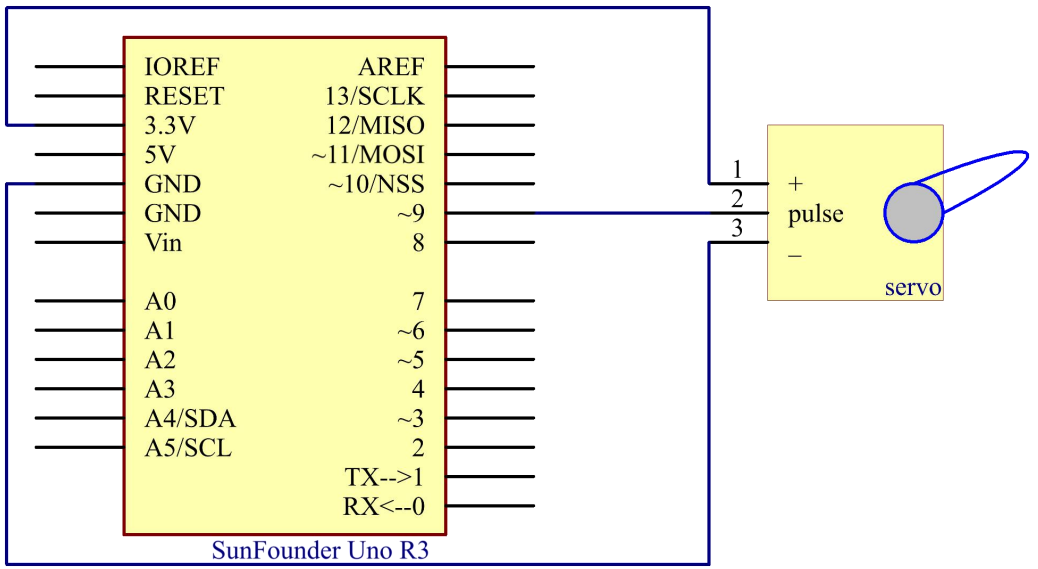
Experimental Principle

Servo

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

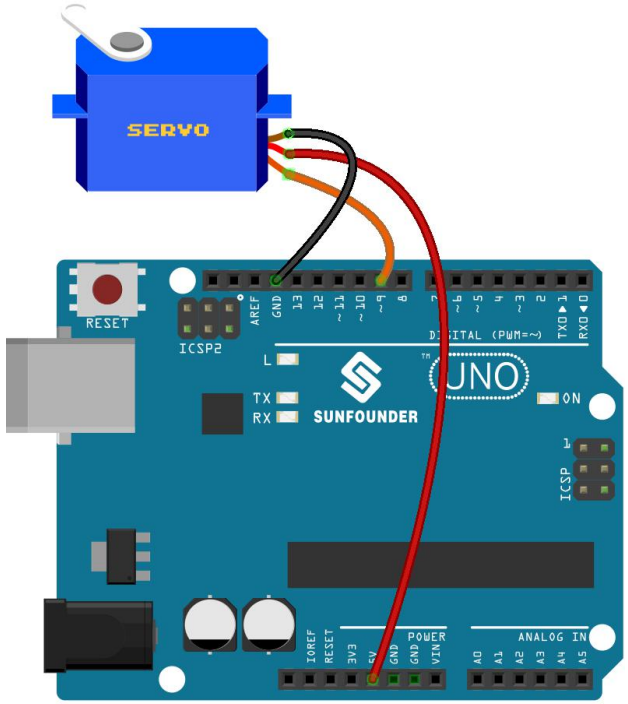
It works like this: The Uno board sends out PWM signals to the servo, and then the control circuit in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear chain and then motivates the shaft after deceleration. The shaft and 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)

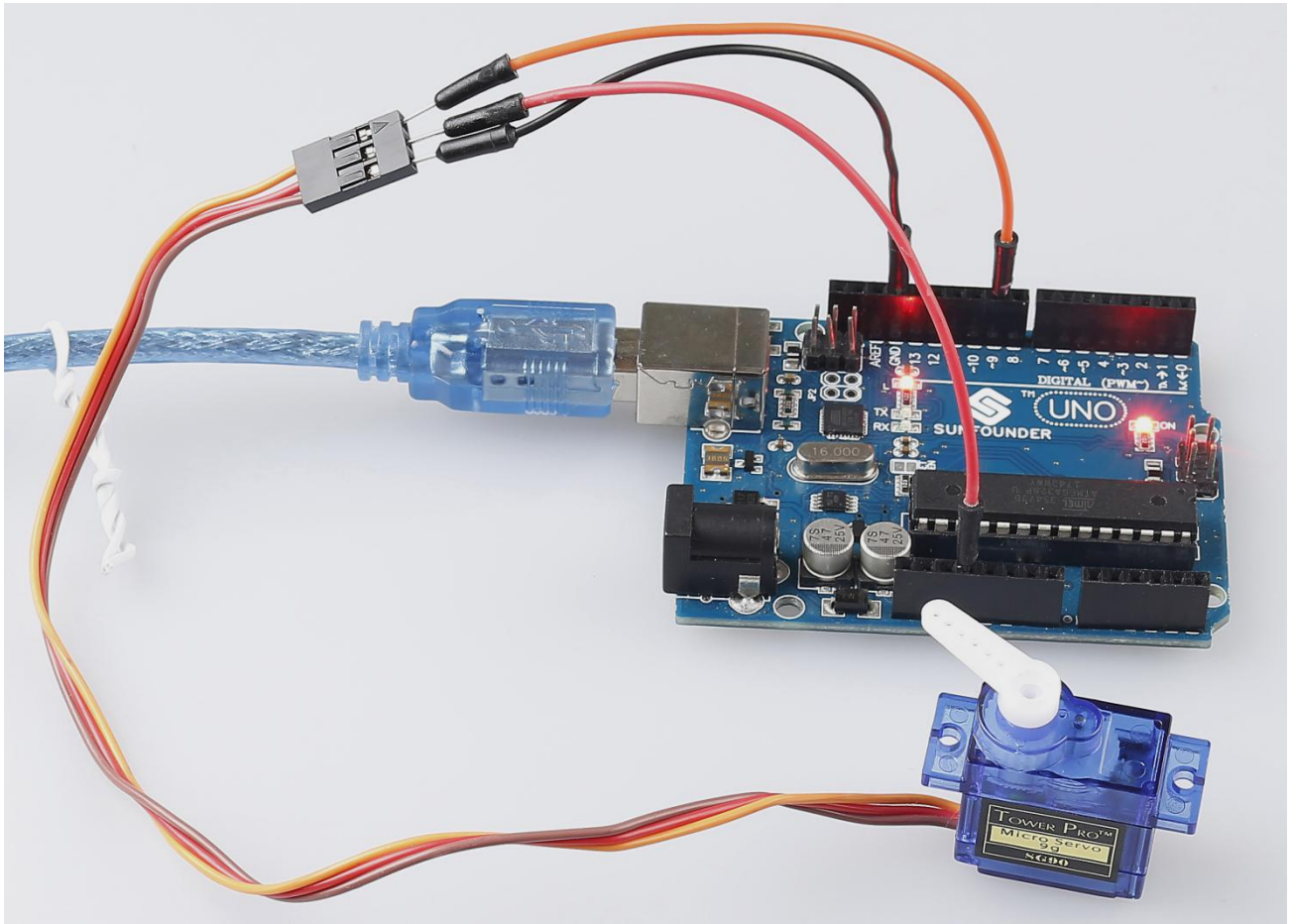


Step 2: Open the code file.

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 13-1 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 13-2 Initialize the servo

```
void setup()
{
  myservo.attach(9);//attachs the servo on pin 9 to servo object
  myservo.write(0);//back to 0 degrees
  delay(1000);//wait for a second
}
```

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 13-3 Servo rotate

```
void loop()
```

```
{
  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
  }
}
```

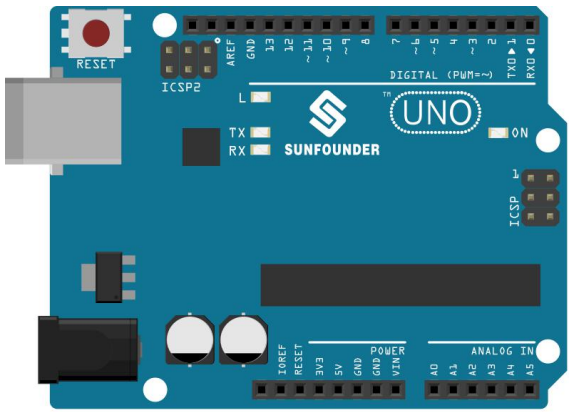
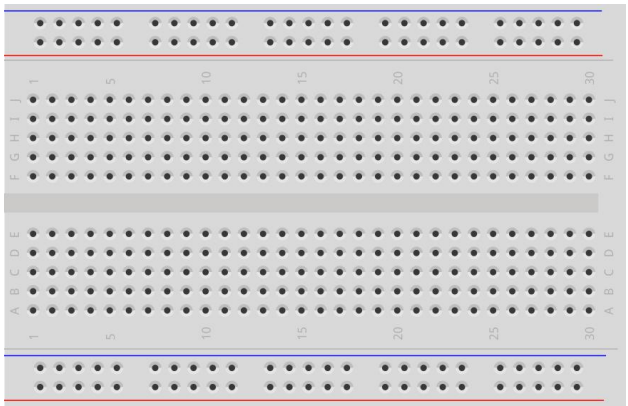
Use 2 for() statement to write 0 - 180 to the servo, so that you can see the servo rotate from 0 to 180 angle,then turn back to 0.

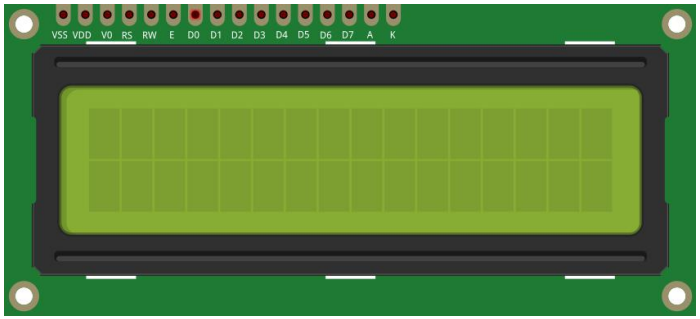
Lesson 14 LCD1602

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 * Uno Board 	1 * Breadboard 

1 * Potentiometer (10kΩ) 	1 * LCD1602 
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 Uno 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.

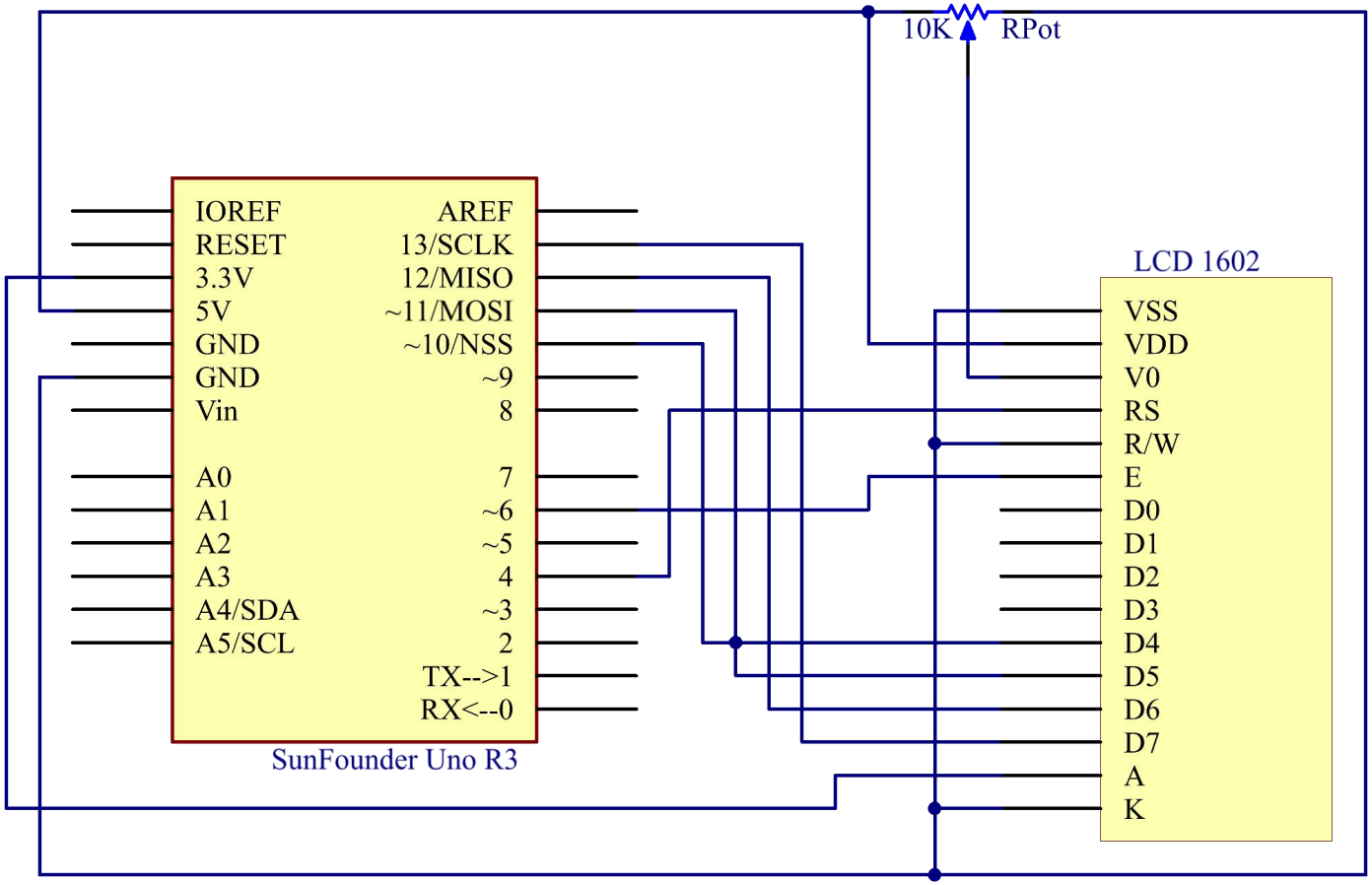
D0-D7: to read and write data

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

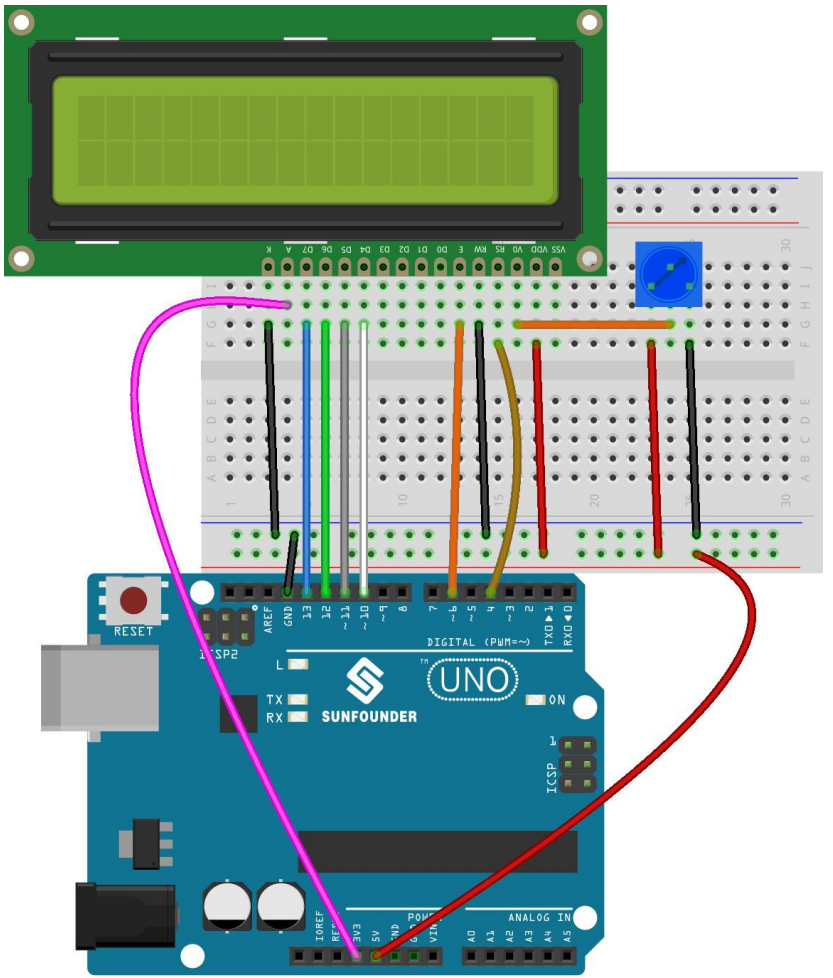
Step 1: Build the circuit (make sure the pins are connected correctly. Otherwise, characters will not be displayed properly):

Step 2: Open the code file.

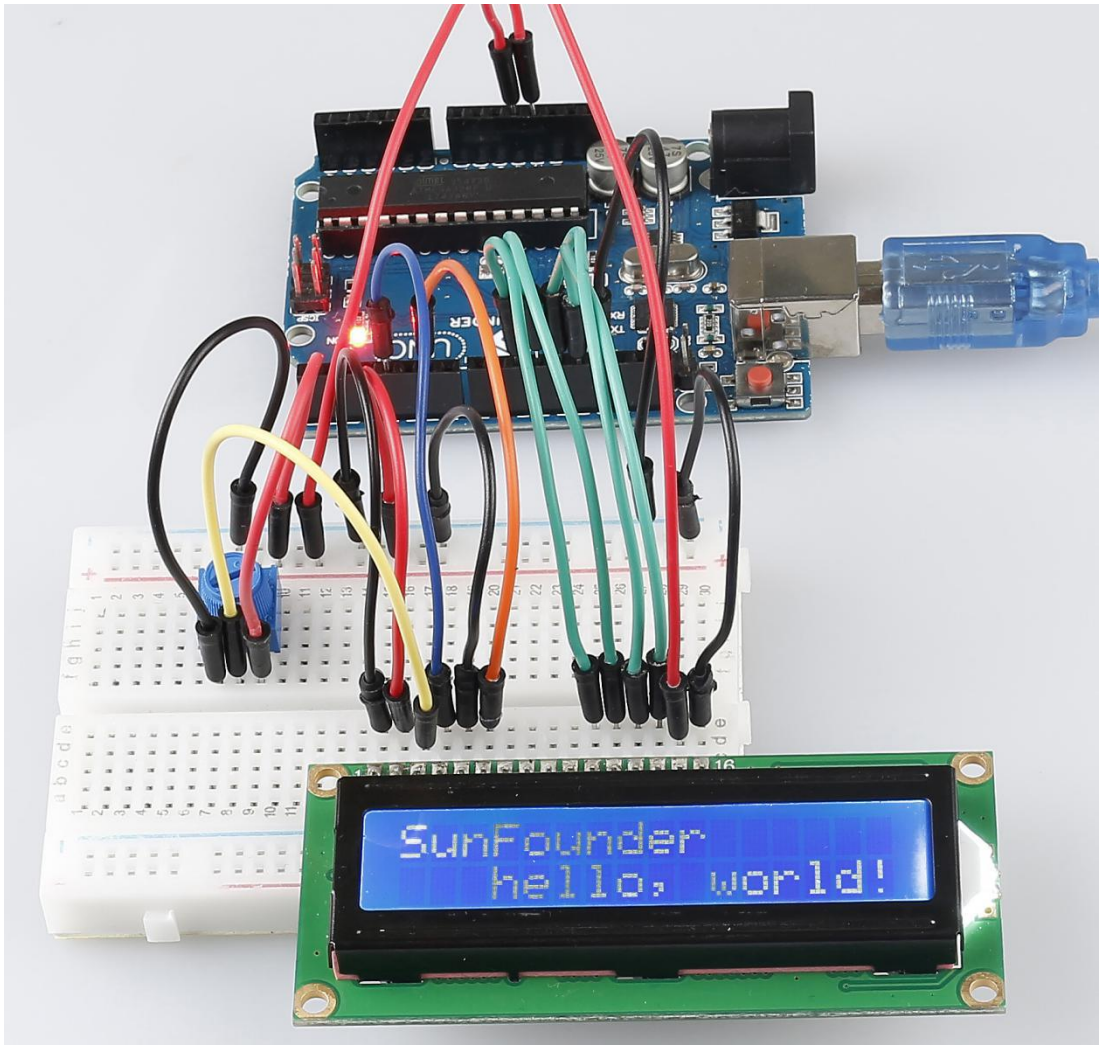
Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

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



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



Code Analysis

Code Analysis 14-1 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 14-2 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 14-3 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 14-4 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 14-5 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 14-6 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
    }
```

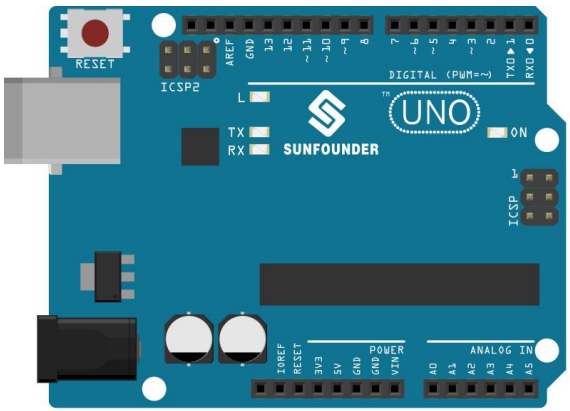
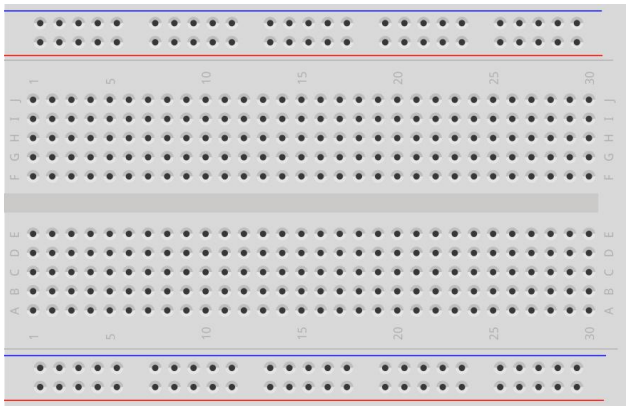
Similarly, the code is to display the elements in *array2[]* on the LCD. Therefore, you will see “SunFounder” scroll in the top line of the LCD, move left until it disappears. And then in the bottom line, “hello, world ! " appears, scrolls to the left until it disappears.

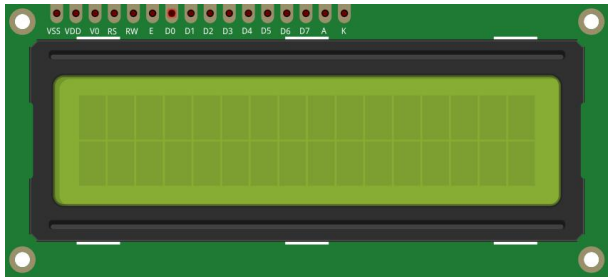
Lesson 15 Thermistor

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

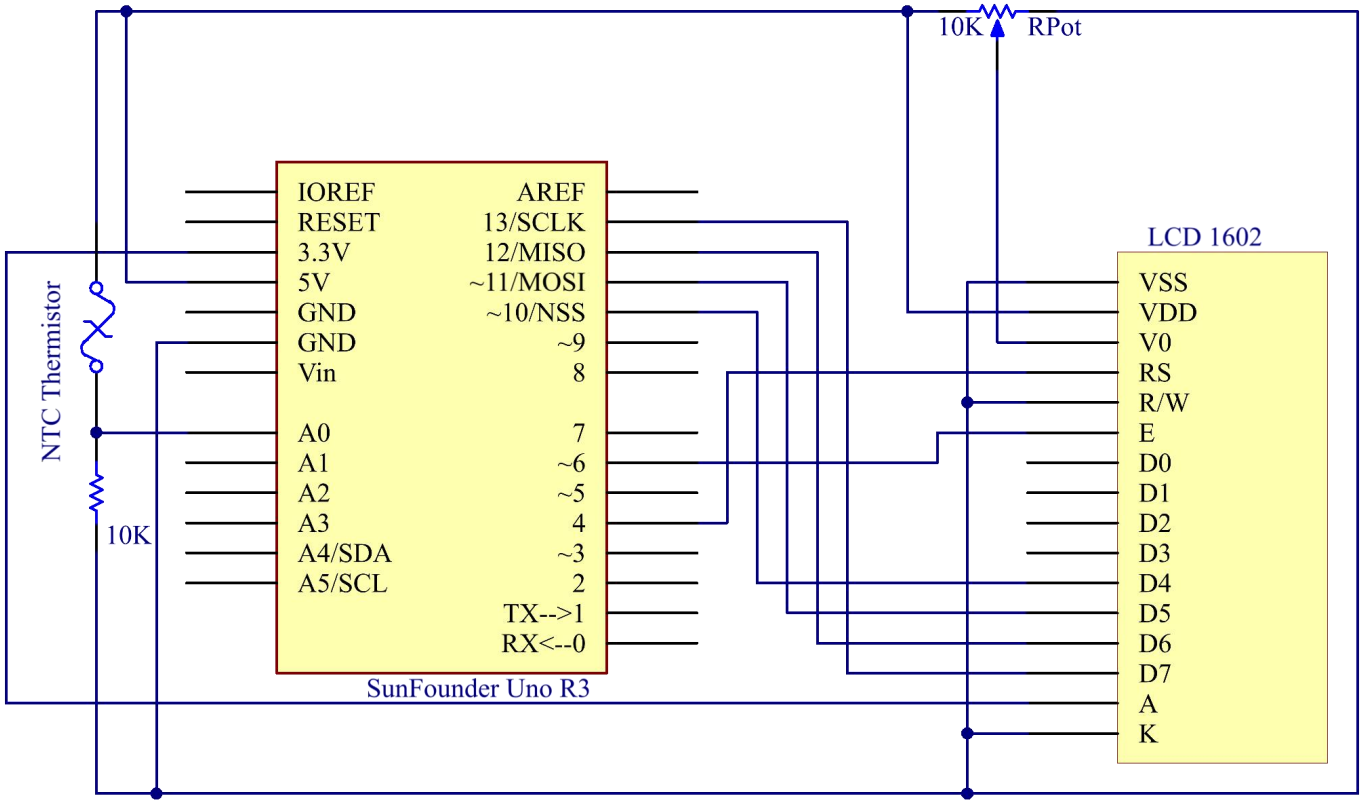
1 * Uno Board 	1 * Breadboard 

1 * LCD1602 	1 * Thermistor 
1 * Resister (10KΩ) 	1 * Potentiometer (10KΩ) 
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 x *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:

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

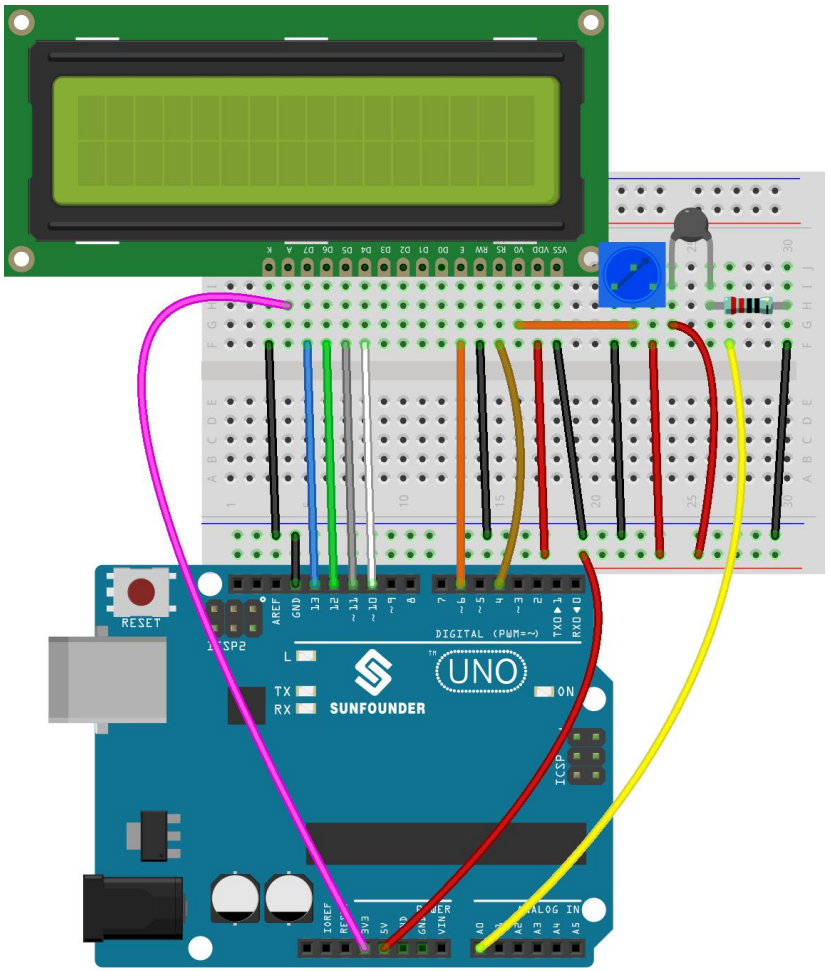
Experimental Procedures

Step 1: Build the circuit

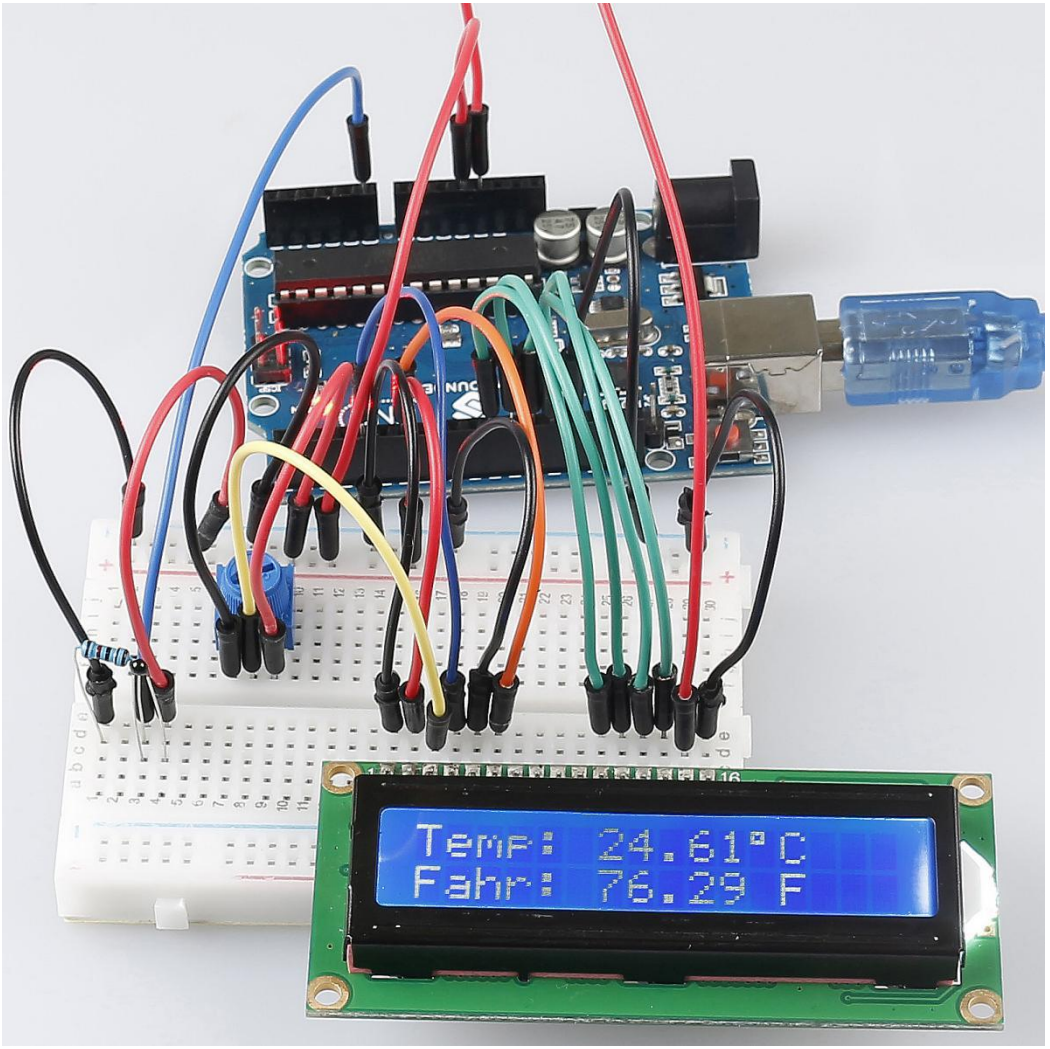
Step 2: Open the code file.

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 15-1 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 3950, which is described in the datasheet of thermistor.

Code Analysis 15-2 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 15-3 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);  //Print the tempC value on display.

  lcd.print(char(223));//print the unit" ° "

  lcd.print("C");
```

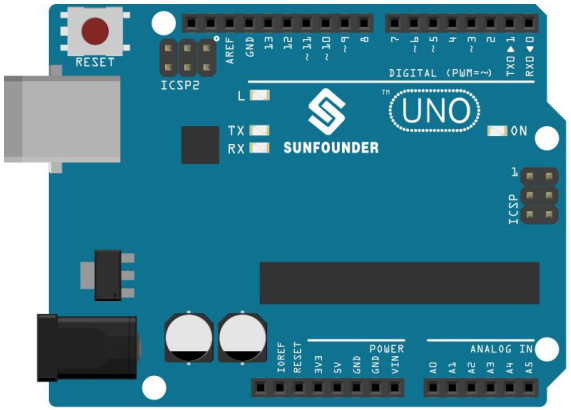
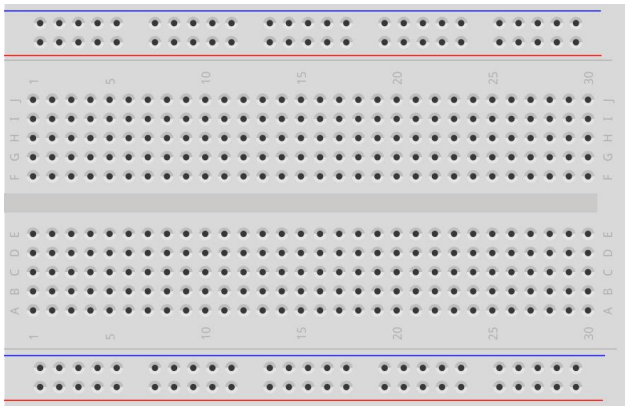
```
// (note: line 1 is the second row, since counting begins with 0):
lcd.setCursor(0, 1); // set the cursor to column 0, line 1
lcd.print("Fahr: ");
lcd.print(tempF);// Print a Fahrenheit temperature to the LCD.
lcd.print(" F"); // Print the unit of the Fahrenheit temperature to the LCD.
delay(200);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

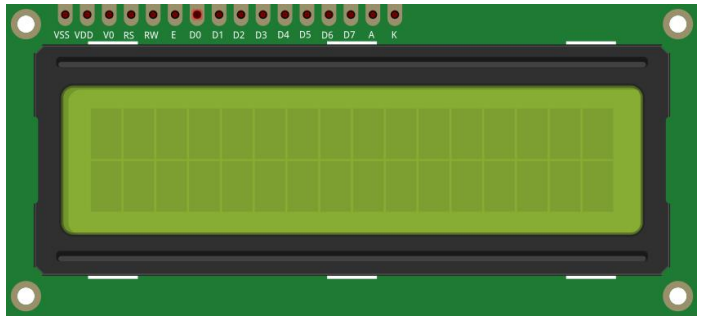
Lesson 16 Voltmeter

Introduction

In this lesson, we will use two potentiometers and an LCD1602 to make a DIY voltmeter.

Components

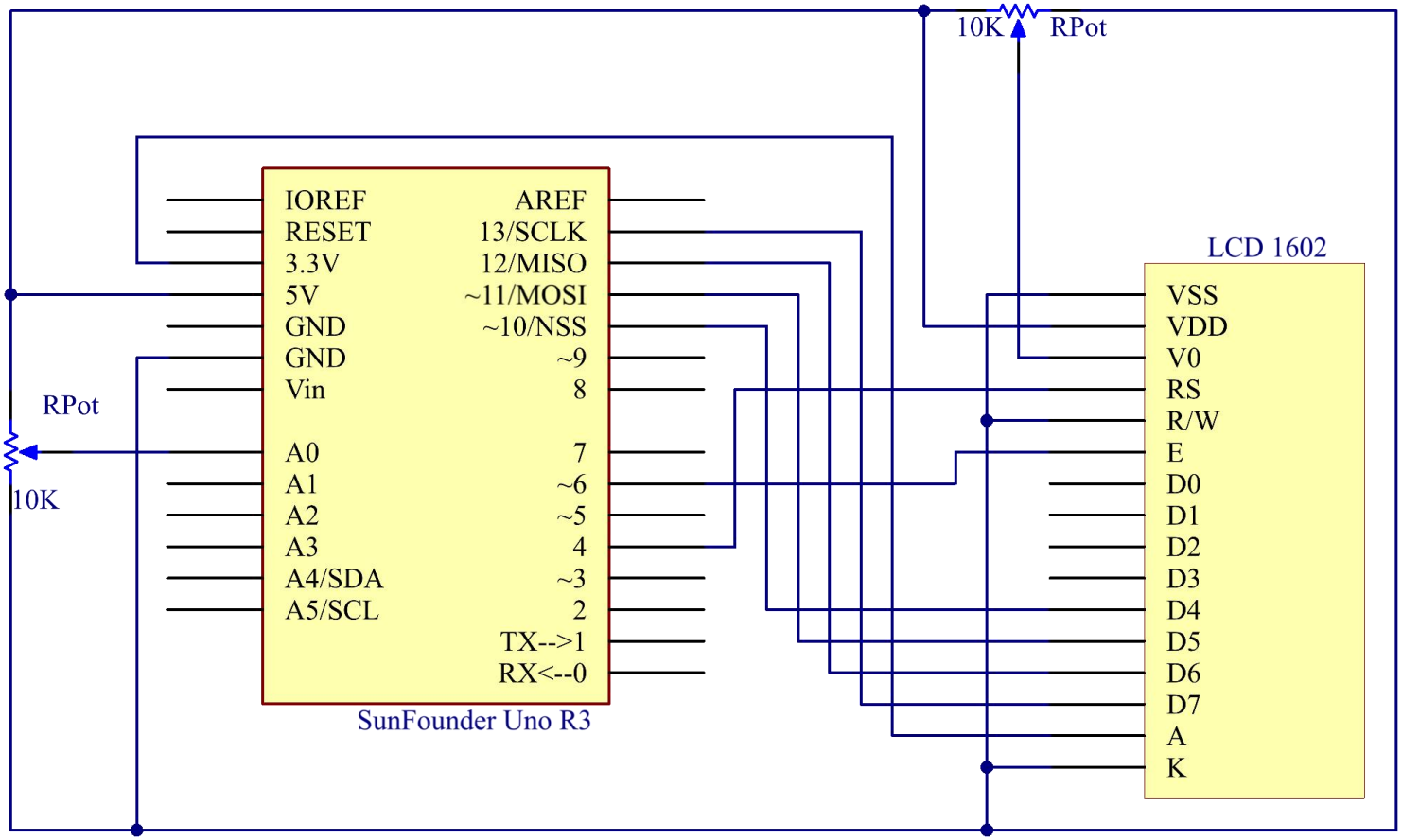
1 * Uno Board 	1 * Breadboard 
1 * USB Cable 	Several Jumper Wires 

2 * Potentiometer (10kΩ) 	1 * LCD1602 

Experimental Principle

Here one potentiometer is used to adjust the contrast of the LCD1602 and the other to divide voltage. When you adjust the potentiometer connected to pin A0 of the SunFounder Uno board, the resistance of the potentiometer will change and the voltage at pin A0 will change accordingly. This voltage change is converted into digital values by A/D converter on the SunFounder Uno board. We can see this change on the serial monitor. Then convert the digital values into voltage with the following formula: the voltage equals the digital value divides by 1024 and then multiplies by 5.0. Finally, display the voltage on the LCD1602.

The schematic diagram:


10K
RPot
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4/SDA
A5/SCL
AREF
13/SCLK
12/MISO
~11/MOSI
~10/NSS
~9
8
7
~6
~5
4
~3
2
TX-->1
RX<--0
SunFounder Uno R3
RPot
10K
LCD 1602
VSS
VDD
V0
RS
R/W
E
D0
D1
D2
D3
D4
D5
D6
D7
A
K

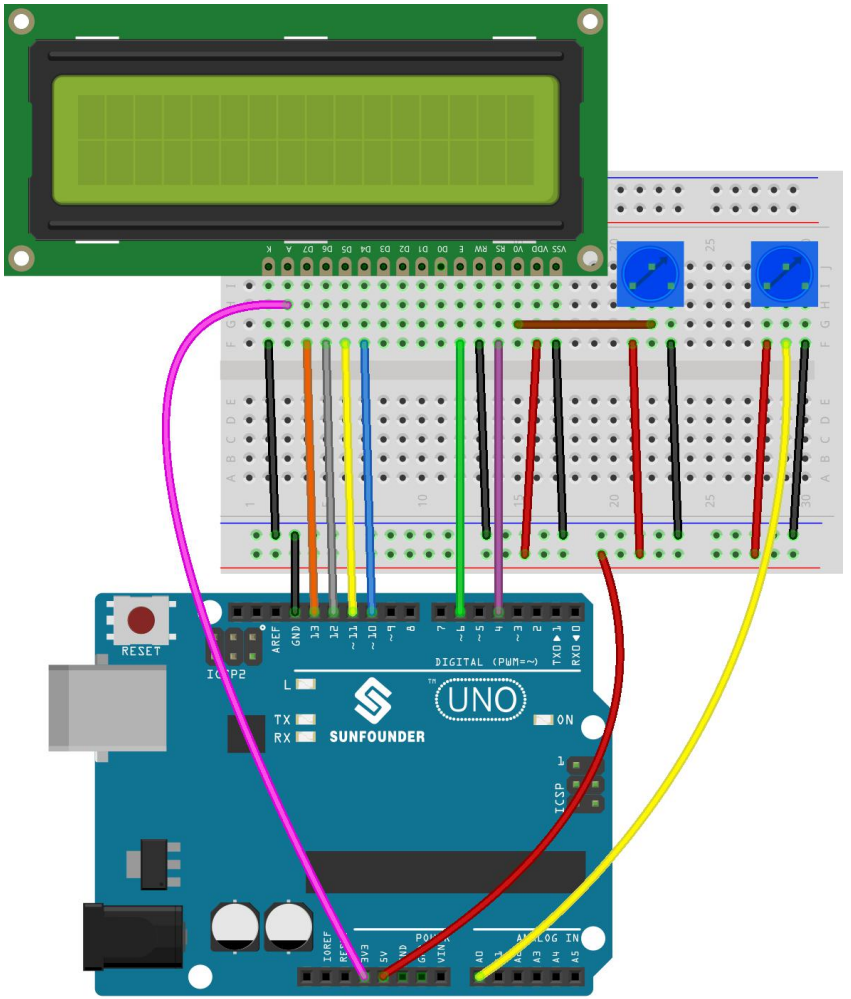
Experimental Procedures

Step 1: Build the circuit

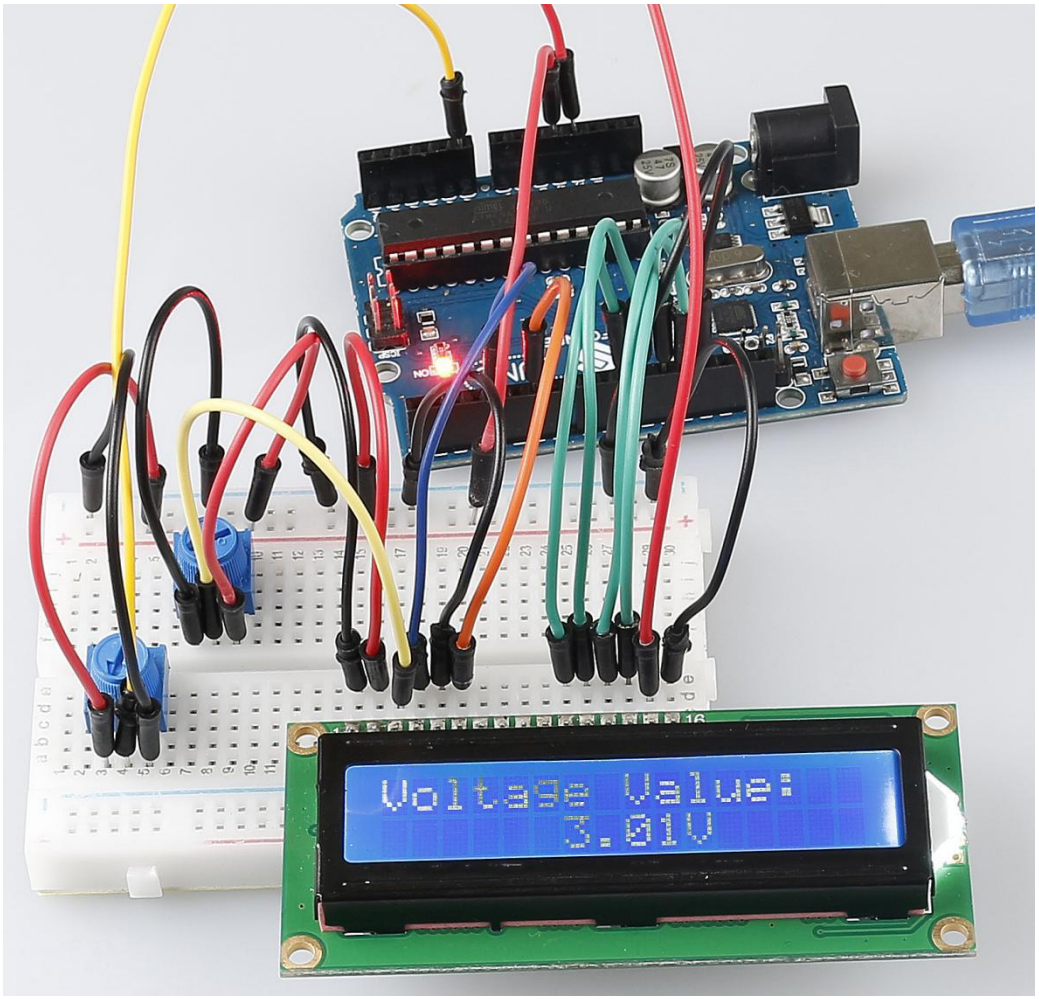
Step 2: Open the code file

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

Step 4: Upload the sketch to the SunFounder Uno board



Now, adjust the potentiometer connected to pin A0, and you will see the voltage displayed on the LCD1602 varies accordingly.



Code Analysis

Code Analysis 16-1 Define the pins of LCD1602 and potentiometer

```
#include <LiquidCrystal.h>

/*************************************************/

float analogIn = 0; //store the analog value of A0

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

float vol = 0; // store the voltage
```

Call the LiquidCrystal library and define the pins of LCD1602 connect to 4,6 and 10 to 13 of Uno board.

Assign the value of A0 to analogIn.

Code Analysis 16-2 Initialize the LCD1602 and serial monitor

```
void setup()

{

  Serial.begin(9600);//Initialize the serial monitor

  lcd.begin(16, 2);// set the position of the characters on the LCD as Line 2, Column 16

  lcd.print("Voltage Value:");//print "Voltage Value:"

}
```

Initialize the baud rate of serial monitor to 9600bps and set the position of the characters on the LCD as Line 2, Column 16. Print “Voltage Value: ” on the LCD1602.

Code Analysis 16-3 Read the analog of A0 and convert to voltage

```
void loop()

{

  analogIn = analogRead(A0);//Read the value of the potentiometer to val

  vol = analogIn/1024*5.0;// Convert the data to the corresponding voltage value in a math
way

  Serial.print(vol);//Print the number of val on the serial monitor

  Serial.println("V"); // print the unit as V, short for voltage on the serial monitor

  lcd.setCursor(6,1);//Place the cursor at Line 1, Column 6. From here the characters are
to be displayed

  lcd.print(vol);//Print the number of val on the LCD

  lcd.print("V");//Then print the unit as V, short for voltage on the LCD

  delay(200); //Wait for 200ms

}
```

The analog value of A0 is: Analog value=5/VA0 * 1024, so VA0= Analog value/1024 * 5, if you connect the potentiometer to 3.3v, then modify 5V to 3.3V.

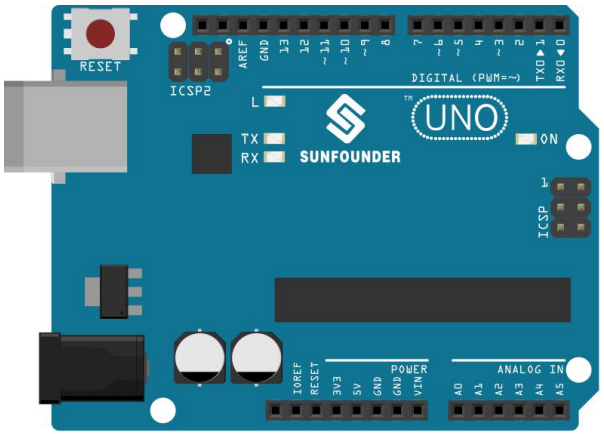
Print the voltage to serial monitor or the LCD1602.

Lesson 17 Automatically Tracking Light Source

Introduction

In this lesson, we will make some interesting creations – use a servo motor, a photoresistor and a pull-down resistor to assemble an automatically tracking light source system.

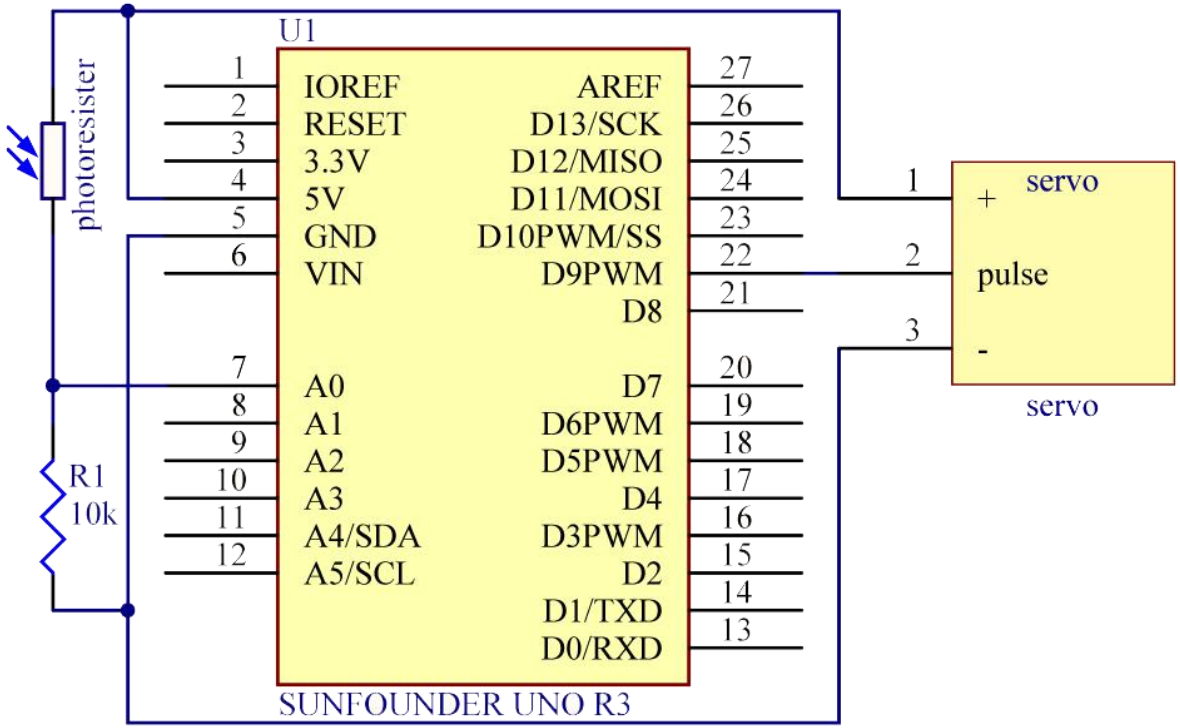
Components

1* Uno Board 	1 * Photo resistor 	1 * Servo 
	1 * Resistor (10KΩ) 	
1 * USB Cable 	Several Jumper Wires 	

Experimental Principle

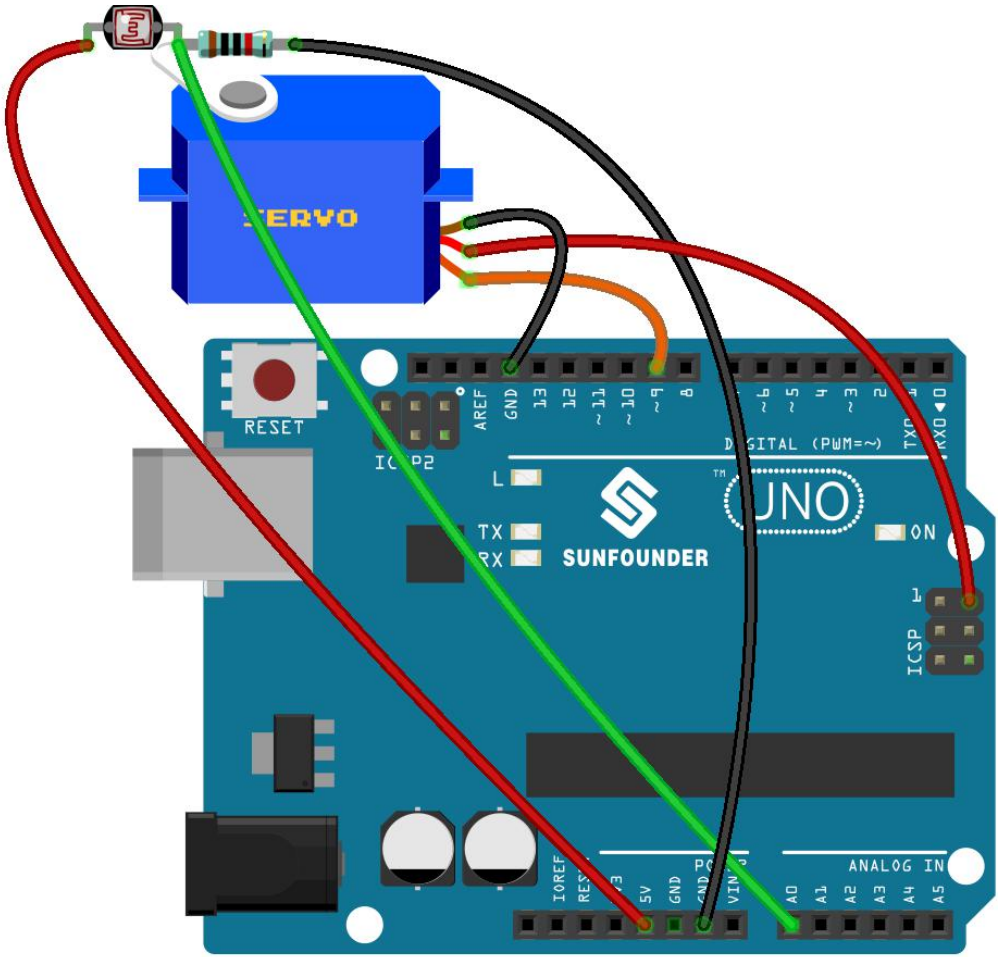
The rocker arm of the servo and the bundled photoresistor sway together to scan and "look" for light source within 180 degrees and record the location of light source when finding one. Then they stop swaying just at the direction of the light source.

The schematic diagram:



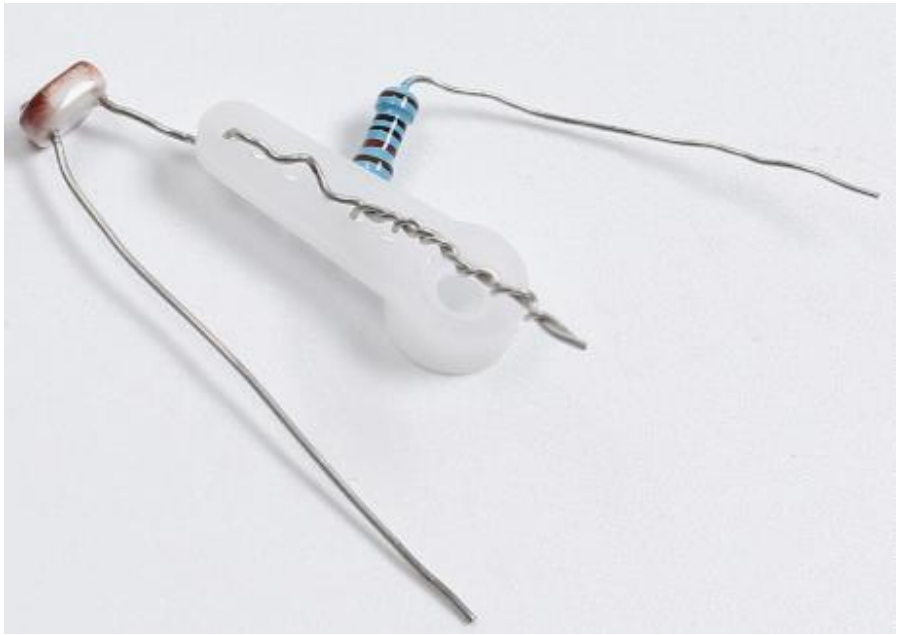
Experimental Procedures

Step 1: Build the circuit



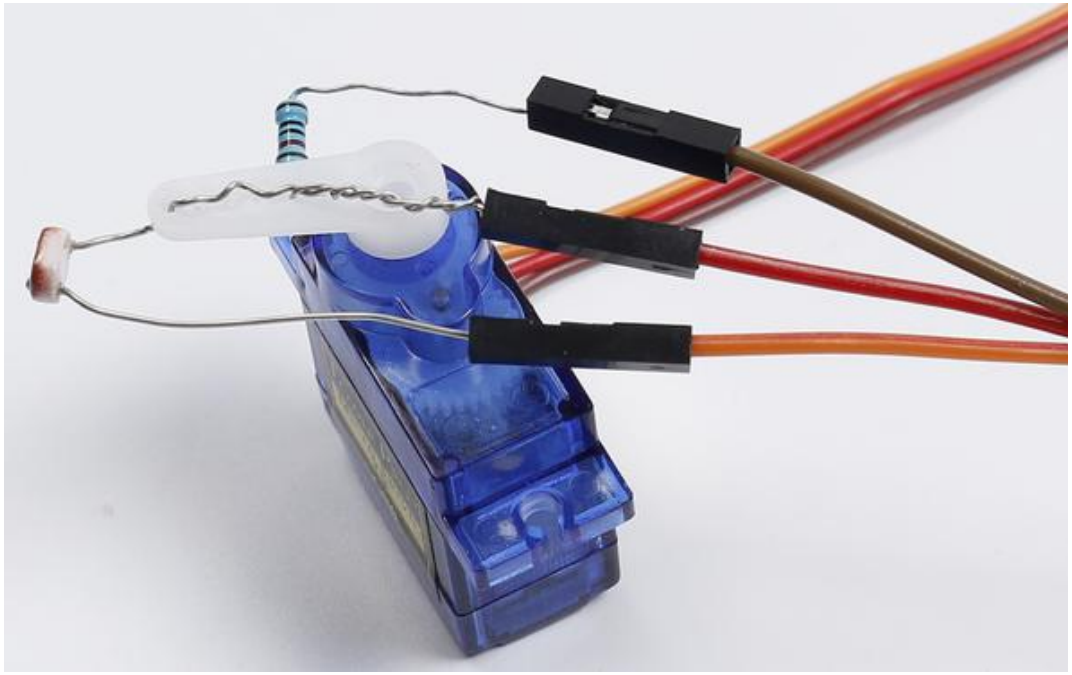
Note: you need to bind one end of the resistor and photoresistor to the rocker arm of the servo (cross the pin through the holes of the arm).

1) Insert one pin of the photoresistor and 10 resistor through the holes on the rocker arm. Pay attention here to tightly winding them because you need to make sure they are connected in the circuit.

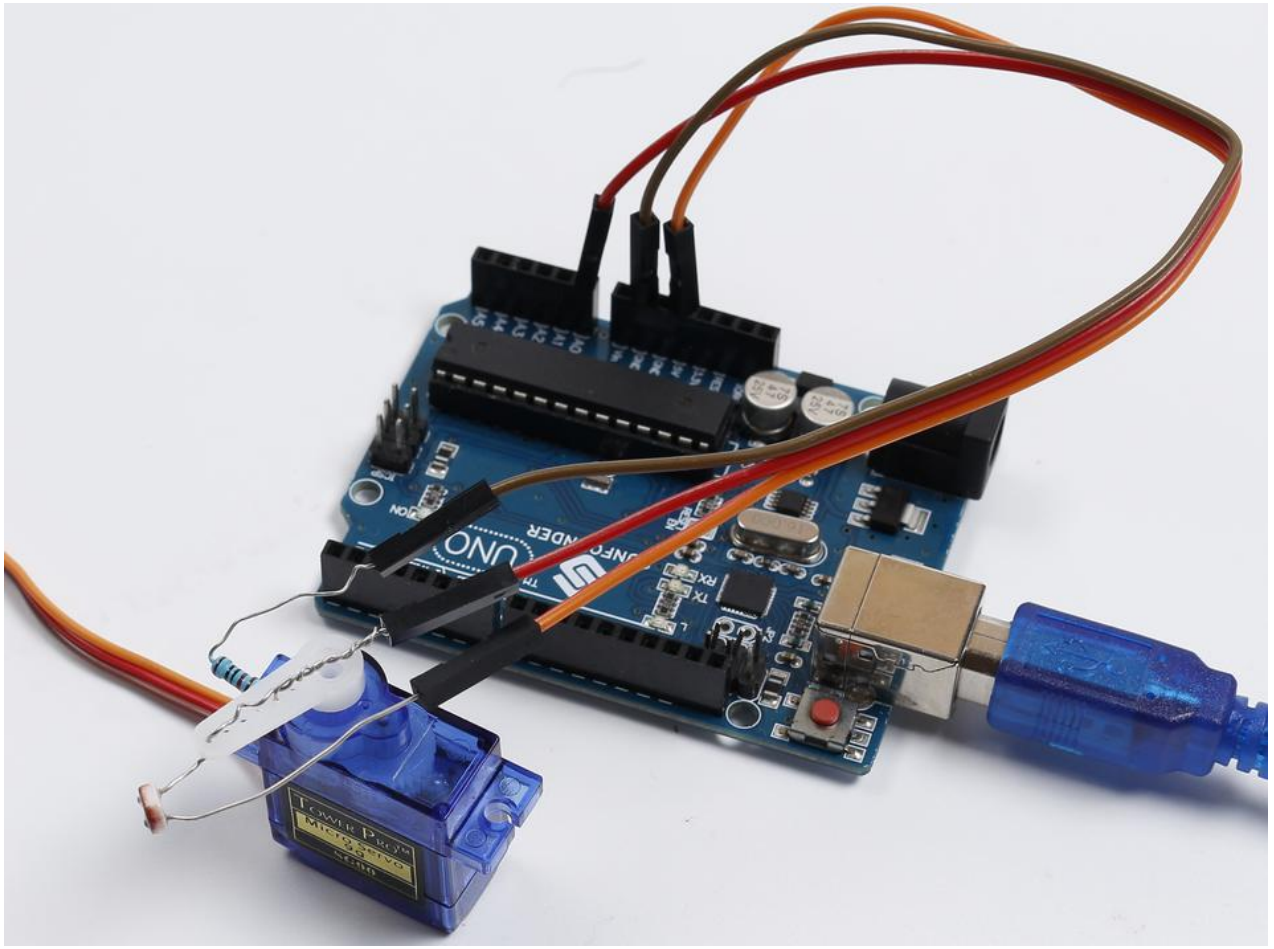


2) Plug in the rock arm to the servo and use 3 jumper wires to hook up the 3 pins.

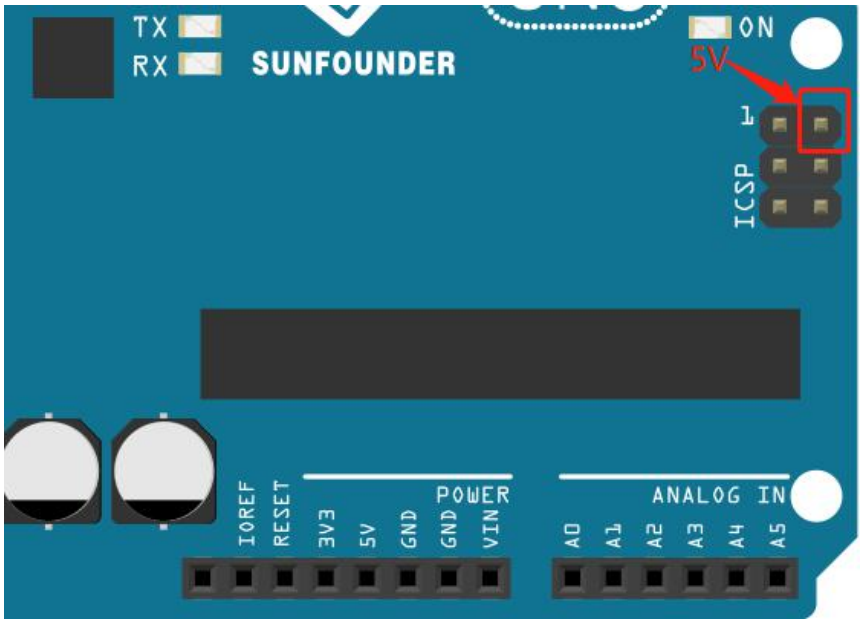
pay attention to plug the pin tightly in case of disentanglement.



3) Hook up the middle pin to pin A0 of the Uno board, another pin of the 10k resistor to GND, photoresistor to 5V.



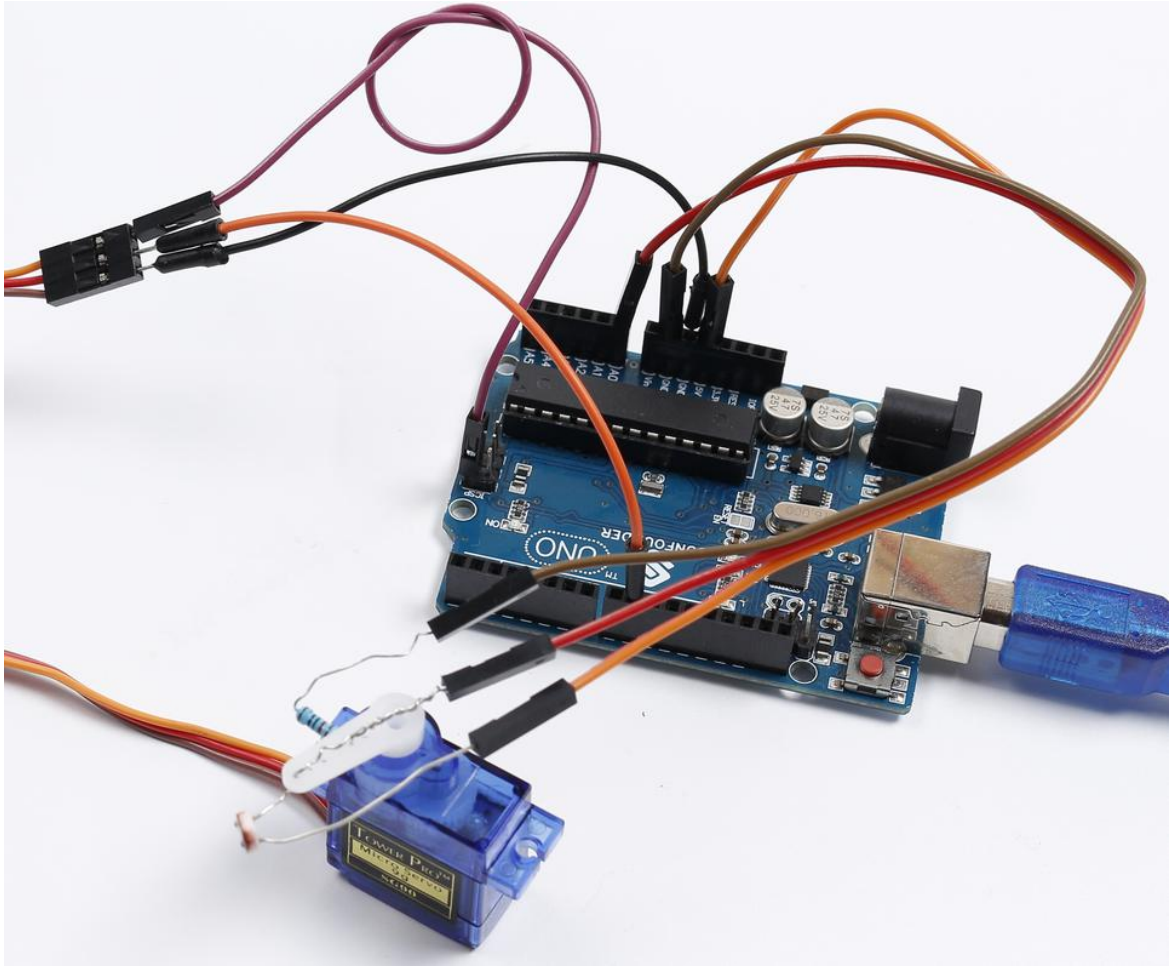
4) Connect the brown wire of servo to GND and red to 5v. Since the 5v usually used is occupied already, you need to connect the other 5v as the following picture shows.



5) Next connect the orange wire to pin 9 of the SunFounder Uno board. OK now the circuit is completed! Connect the Uno board to your computer with a USB cable.

Step 2: Open the code file
Step 3: Select the correct **Board** and **Port**
Step 4: Upload the sketch to the SunFounder Uno board

Now, shine a flashlight onto the photoresistor. Then you will see the rocker arm of the servo and the photoresistor rotate and finally stop at the direction of light source.



Code Analysis

Code Analysis 17-1 Initialize and define variables

```
#include <Servo.h>

const int photocellPin = A0;  //The photoresistor is connected to A0

/*********************************************/

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

int outputValue = 0;  //Save the value read from A0

int angle[] = {0,10, 20, 30, 40, 50, 60,70, 80, 90, 100,110,120,130,140,150,160,170,180};
//Define the angle of servo

int maxVal = 0;  //Record the maximum number

int  maxPos = 0;  //Record the angle of the servo when the read the maximum number of
photoresistor.
```

Define an integer array angle[], which contains 19 elements from 0 to 18, representing 0 to 180 which indicates the degree of servo rotation. For example, angle[0] means 0 degree, angle[1] is 10 degrees, and so forth.

Code Analysis 17-2 Servo rock arm stop at the direction of light source

```
void loop()

{
```

```
  for(int i = 0; i < 19; i ++)
  {
    myservo.write(angle[i]);//write the angle from the angle[i] array to servo. When i=0, angle[0]=0, i=1, angle[1]=10, and so on.
    outputValue = analogRead(photocellPin);//read the value of A0
    Serial.println(outputValue);//print it
    if(outputValue > maxVal)//if the current value of A0 is greater than previous
    {
      maxVal = outputValue;//write down the value
      maxPos =i;//write down the angle
    }   delay(200);  //delay 200ms }
  myservo.write(angle[ maxPos]);//write the angle to servo which A0 has greatest value
 delay(1000);  //delay 1s
}
```

Set the servo to rotate from 0 to 180, and the angle is defined in the angle [] array. Since the photoresistor is wound with the servo rock arm, the resistance of the photoresistor changes with different light intensities each time the servo is rotated, so the analog value of A0 is changed at the same time.

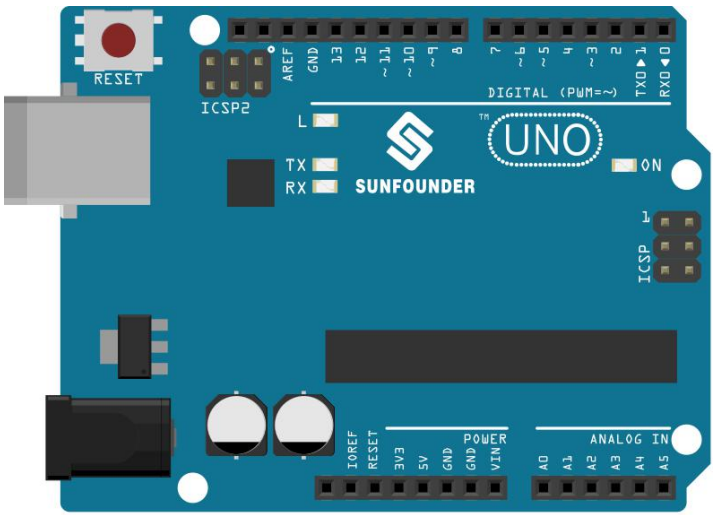
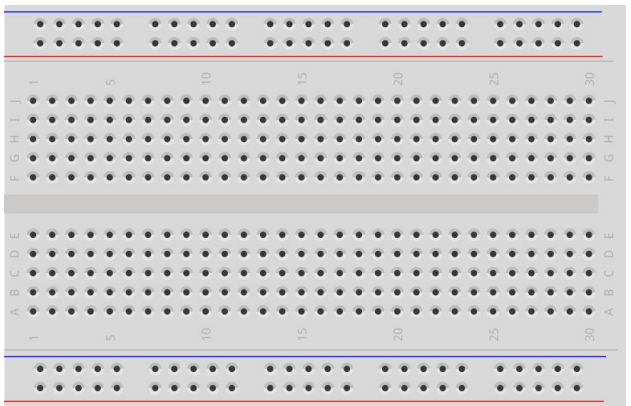
By comparing the value of A0 with the previously recorded maximum value, record the maximum A0 value and current angle of the servo. Finally let the servo turn to this angle.

Lesson 18 Light Alarm

Introduction

This experiment is a very interesting one – a DIY phototransistor. DIY phototransistors use the glow effect and photoelectric effect of LEDs. That is, LEDs will generate weak currents when some light is shined on it. And we use a transistor to amplify the currents generated, so the SunFounder Uno board can detect them.

Components

1 * Uno Board 	1 * Breadboard 

1 * Resistor (10KΩ) 	1 * LED 	1 * Buzzer(Passive) 	1 * NPN S8050 
1 * USB Cable 		Several Jumper Wires 	

Experimental Principle

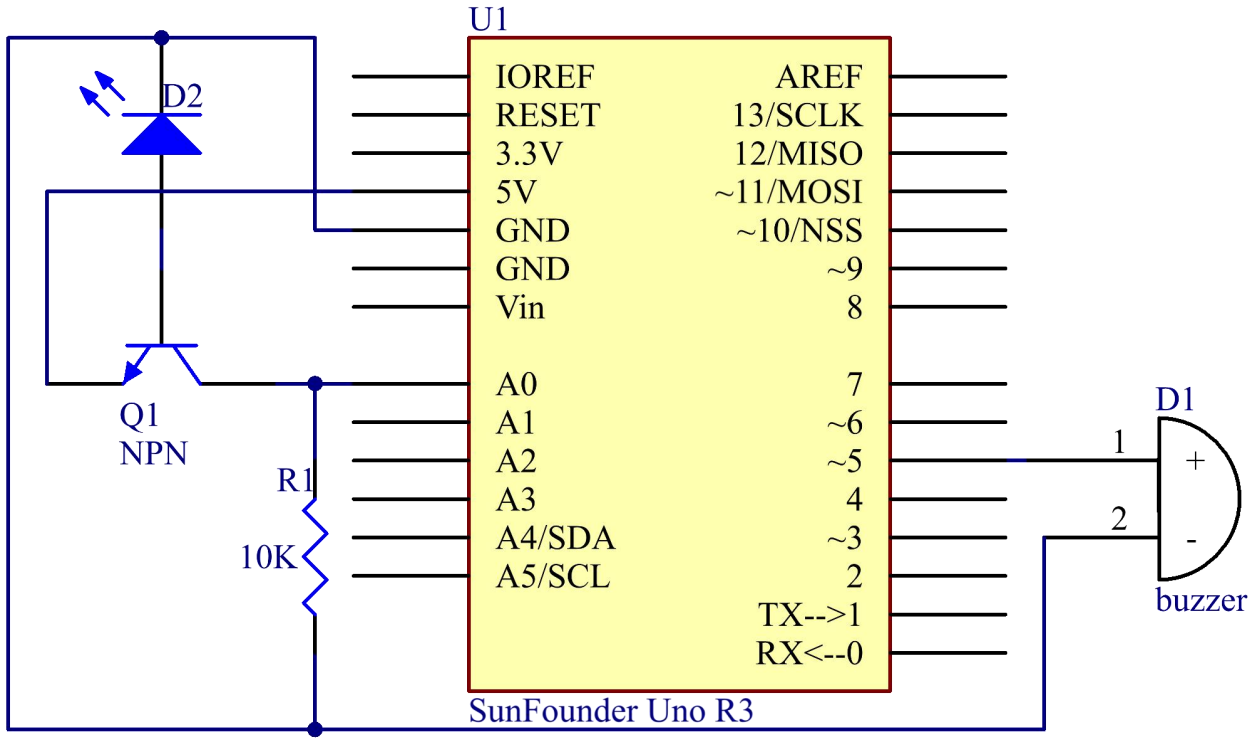
LEDs not only have a glow effect, but also a photoelectric effect. They will generate weak currents when exposed to light waves.

NPN consists of a layer of P-doped semiconductor (the "base") between two N-doped layers (see the picture above). A small current entering the base is amplified to produce a large collector and emitter current. That is, when there is a positive potential difference measured from the emitter of a NPN transistor to its base (i.e., when the base is high relative to the emitter) as well as positive potential difference measured from the base to the collector, the transistor becomes active. In this "on" state, current flows between the collector and emitter of the transistor.

A 10kΩ pull-down resistor is attached to the transistor output stage in order to avoid analog port suspending to interfere with signals and cause misjudgment.

When the LED exposed to light waves, the LED generates weak currents, then the NPN transistor becomes active. Then read the analog value of A0, when A0>0, then set pin 5(buzzer) to high level.

The schematic diagram:



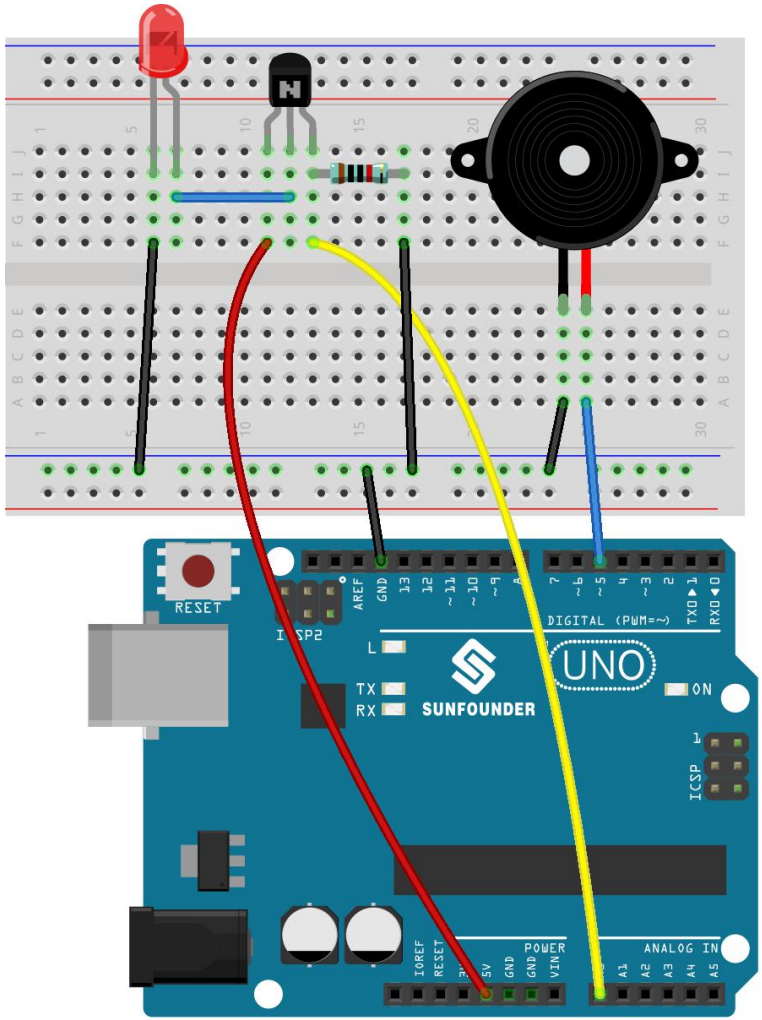
Experimental Procedures

Step 1: Build the circuit.

Step 2: Open the code file

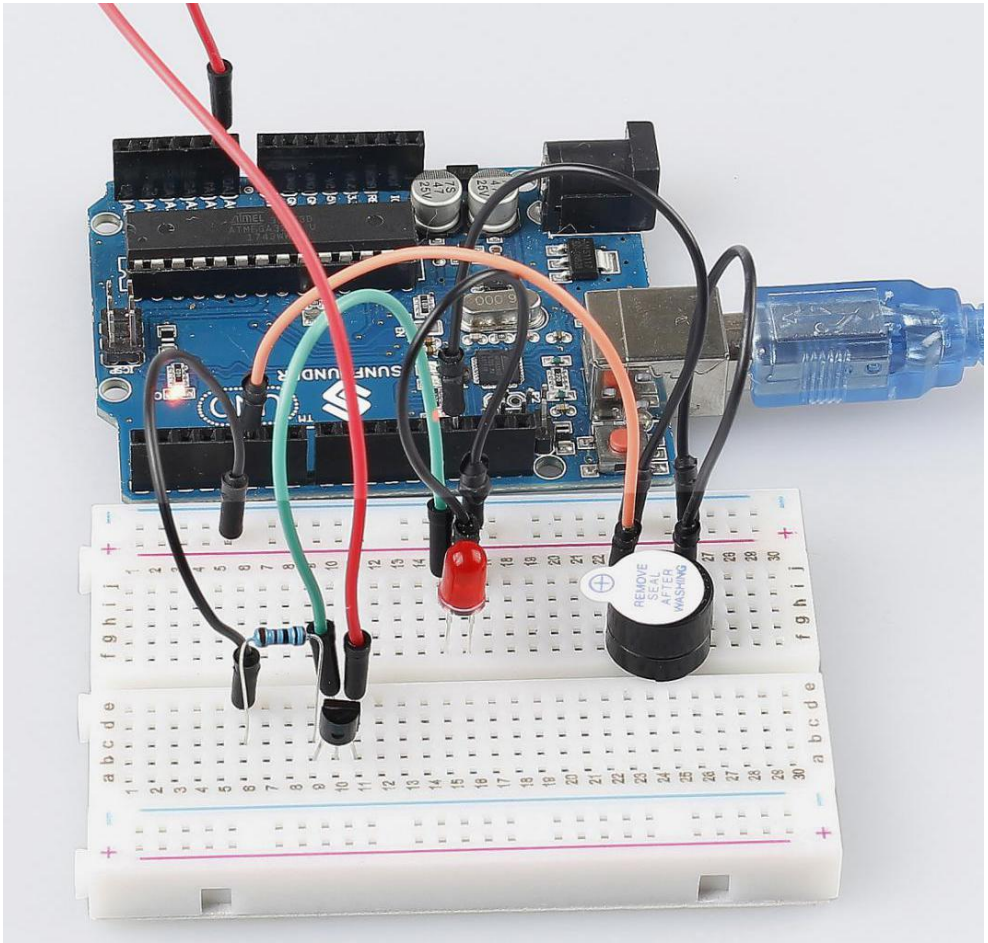
Step 3: Select the correct Board and Port

Step 4: Upload the sketch to the board



Now, you can hear the buzzer beep when shining a flashlight on the LED.

Note: You need to do this experiment in a dark environment, or the lights you give need to be much stronger than ambient light.



Code Analysis

Code Analysis 18-1 Whole Code

```
void setup()
{
  Serial.begin(9600);  // start serial port at 9600 bps:
}
void loop()
{
  int n=analogRead(A0);   //read the value from analog pin AO
  Serial.println(n);
  if(n>0)                //If there is a voltage
  {
    pinMode(5,OUTPUT);         //set the digital pin 5 as an output
    tone(5,10000);          //Generates a square wave of the frequency of 10000 Hz (and 50%
duty cycle) on pin 5
    pinMode(5,INPUT);           //set the pin 5 as an input
  }
}
```

tone()

Description

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the `tone()` function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

Syntax

`tone(pin, frequency)`

`tone(pin, frequency, duration)`

Parameters

`pin`: the pin on which to generate the tone

`frequency`: the frequency of the tone in hertz - `unsigned int`

`duration`: the duration of the tone in milliseconds (optional) - `unsigned long`

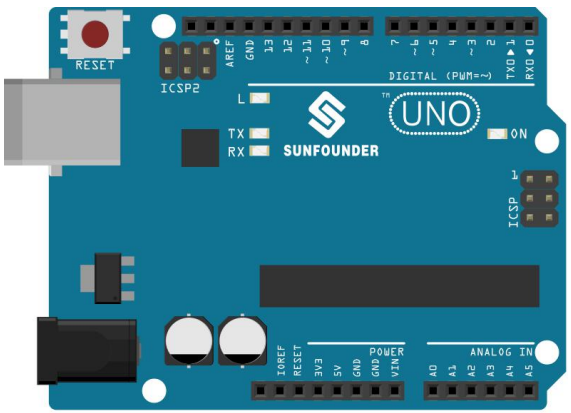
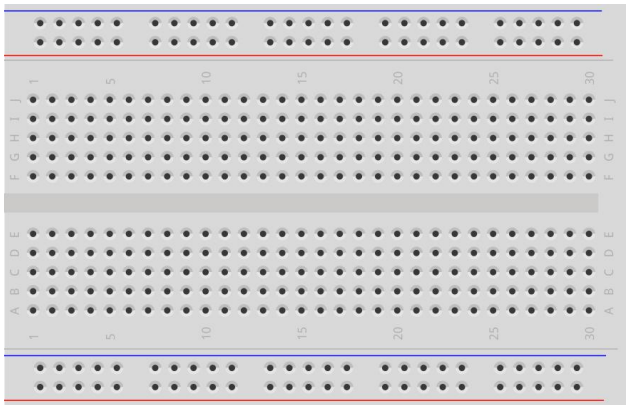
Lesson 19 Answer Machine

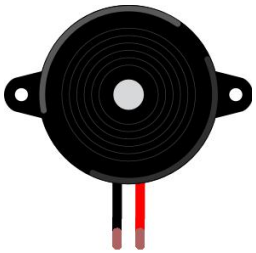
Introduction

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

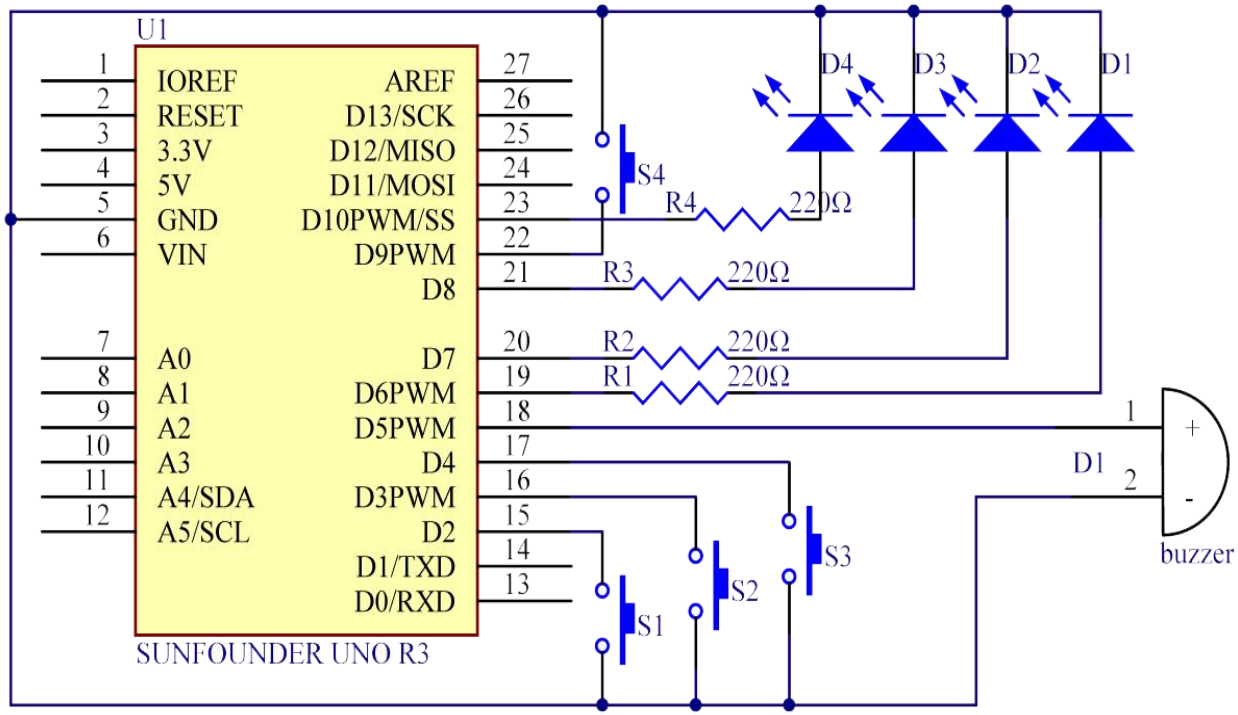
1 * Uno Board 	1 * Breadboard 

1 * Red LED 	1 * Yellow LED 	1 * Green LED 	1 * Blue LED 
1 * Active Buzzer 	4 * Button 	4 * Resistor (220Ω) 	
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:



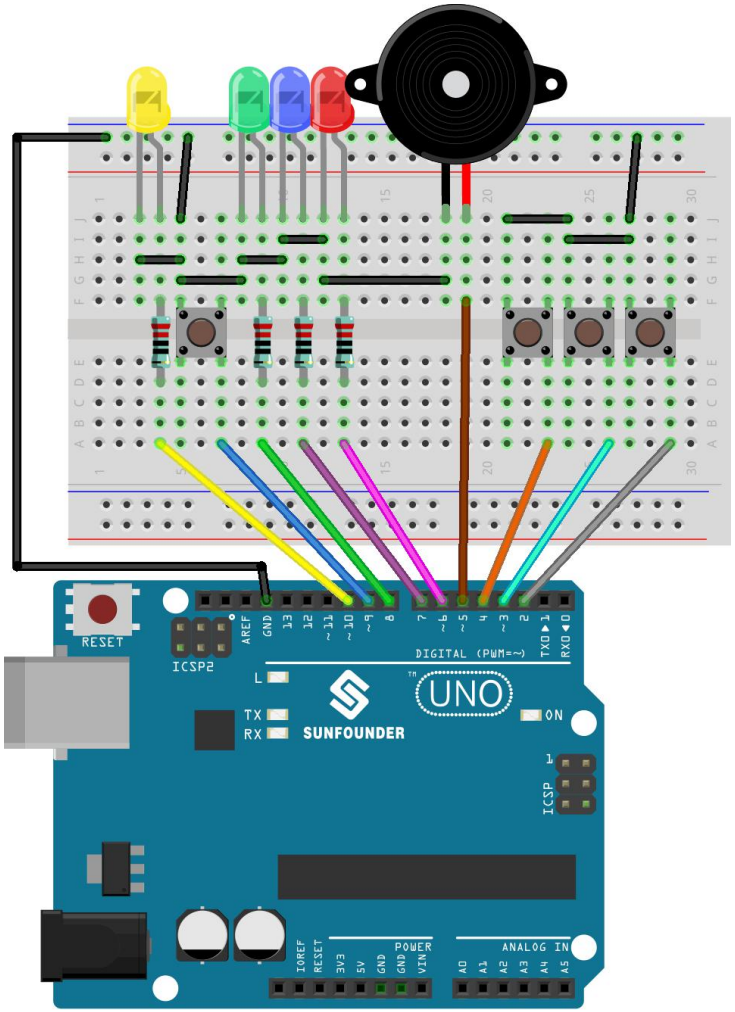
Experimental Procedures

Step 1: Build the circuit

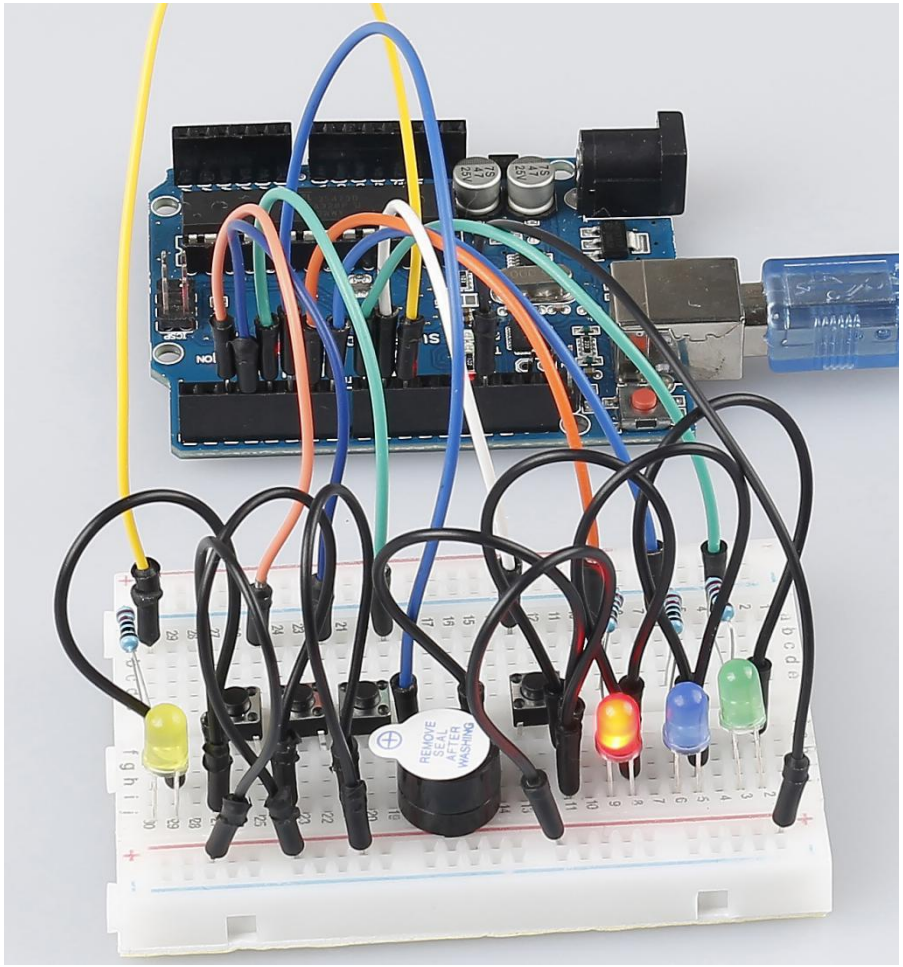
Step 2: Open the code file.

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 19-1 loop() function

```
b4State = digitalRead(button4); //read the value of button4 to see if it was pressed.
Serial.println(b4State);  //print it value.
  //when button4 pressed
  if(b4State == 0)  //if the button4 is pressed, the 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 reset LED on
      delay(200);  //delay 200ms
    }
  }
```

```
if(1 == flag)
{
  //read the state of other buttons
  b1State = digitalRead(button1);
  b2State = digitalRead(button2);
  b3State = digitalRead(button3);
  //If the button1 press the first
  if(b1State == 0)  //if button1 is pressed
  {
    flag = 0;  //flag equals to 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
  }
  ...
```

Use the same way to detect the button2 and button3, 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 19-2 Alarm() function

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

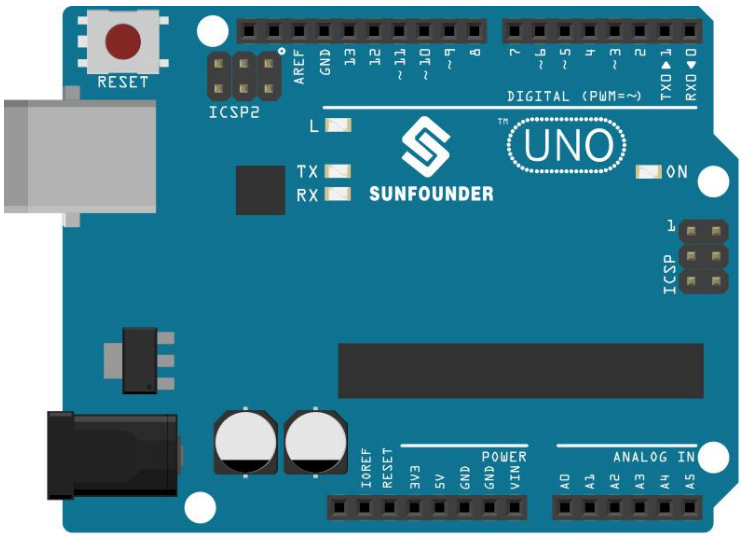
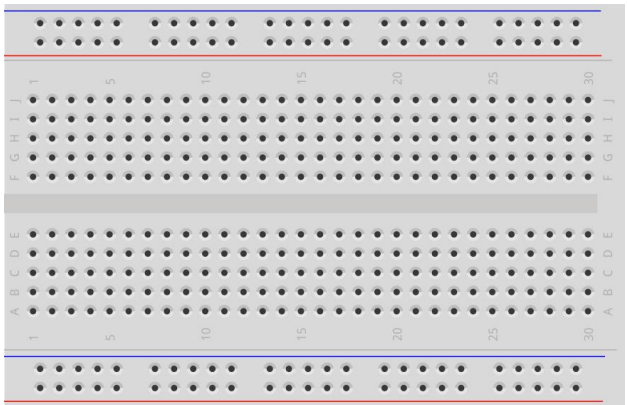
The alarm() function is to set the buzzer to beep. You can change the frequency and time of the buzzer sound.

Lesson 20 Controlling Voice by Light

Introduction

Previously we have learnt how to use a photoresistor. In this lesson, let's get further - control a buzzer to beep in different frequencies by the photoresistor.

Components

1 * Uno Board 	1 * Breadboard 

1 * Active Buzzer 	1 * Photoresistor 	1 * Resistor (10KΩ) 
1 * USB Cable 	Several Jumper Wires 	

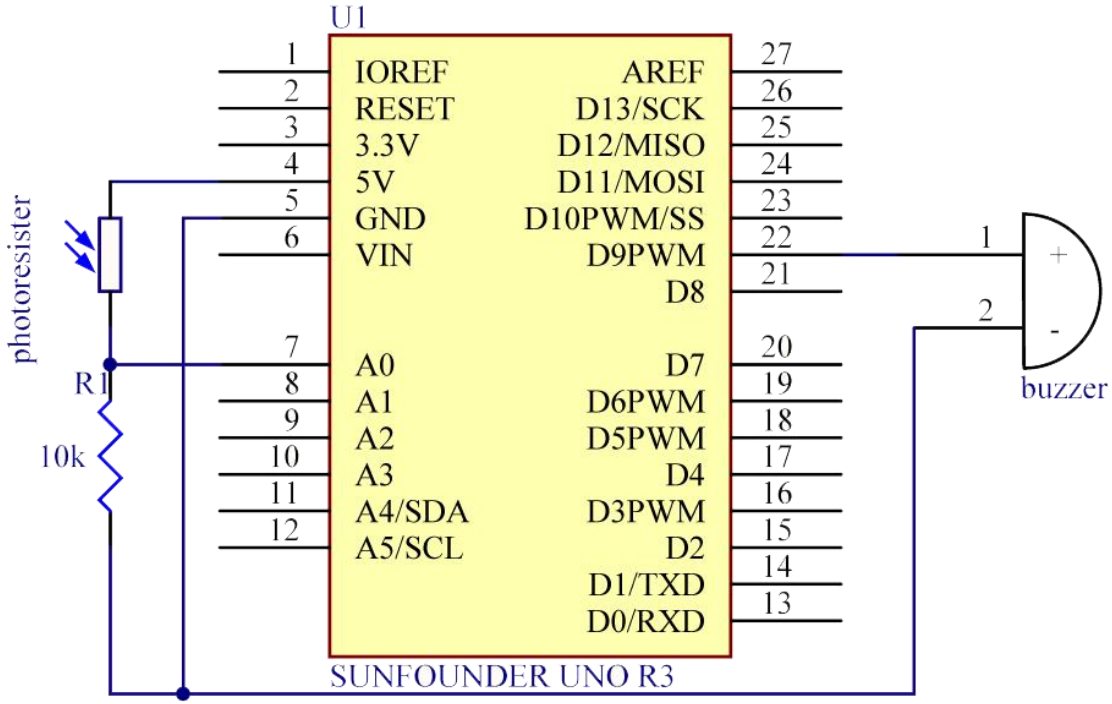
Experiment Principle

When you shine some light on the photoresistor, if the incident light gets stronger, the resistance of the photoresistor will decrease; if the incident light becomes weaker, the resistance will increase. We can apply this principle to change the voltage distribution in the circuit.

In this experiment, the output of the photoresistor is sent to pin A0 on the SunFounder Uno board and then processed by the ADC on the board to output a digital signal. We use this digital signal as the parameter of the delay() function in the sketch to make the buzzer beep.

When the incident light is strong, the output value gets greater, thus the buzzer will beep slowly; when incident light is weak, the output value is smaller, thus the buzzer will beep sharply.

The schematic diagram:



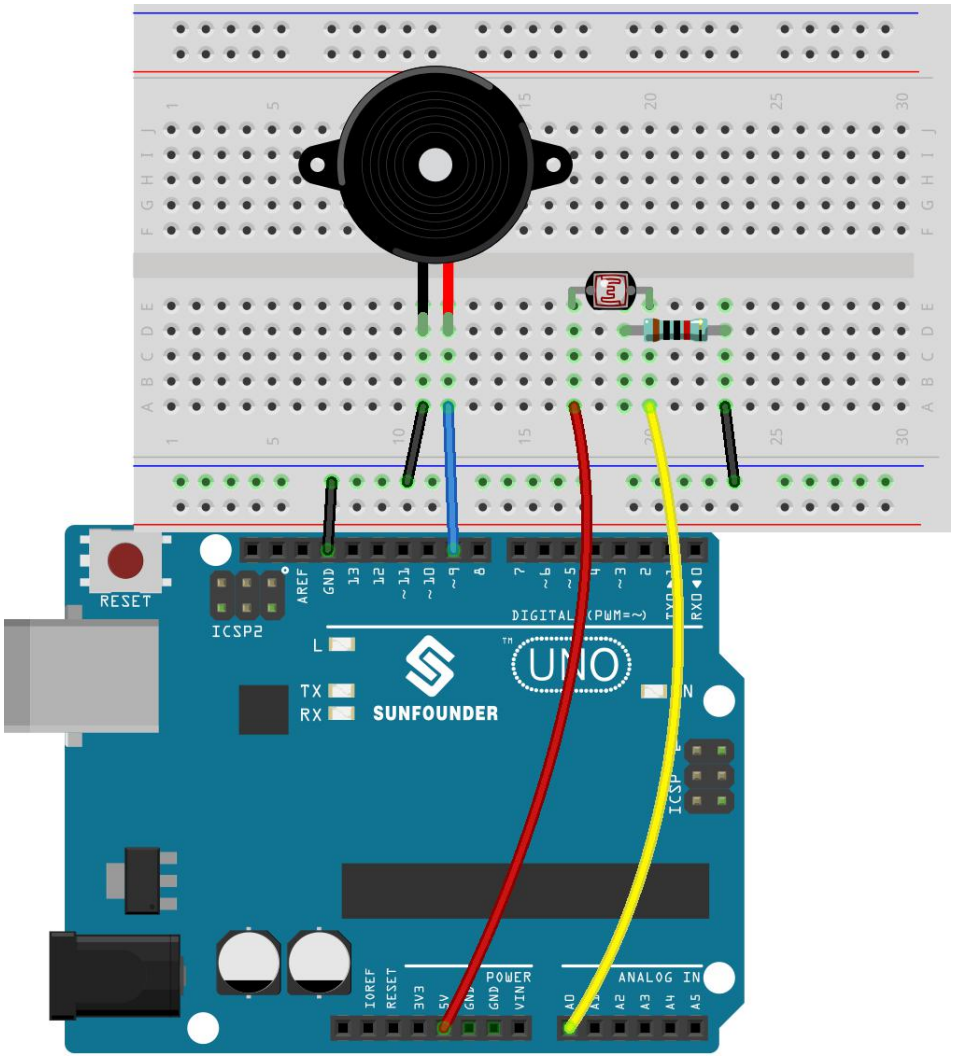
Experiment Procedures

Step 1: Build the circuit

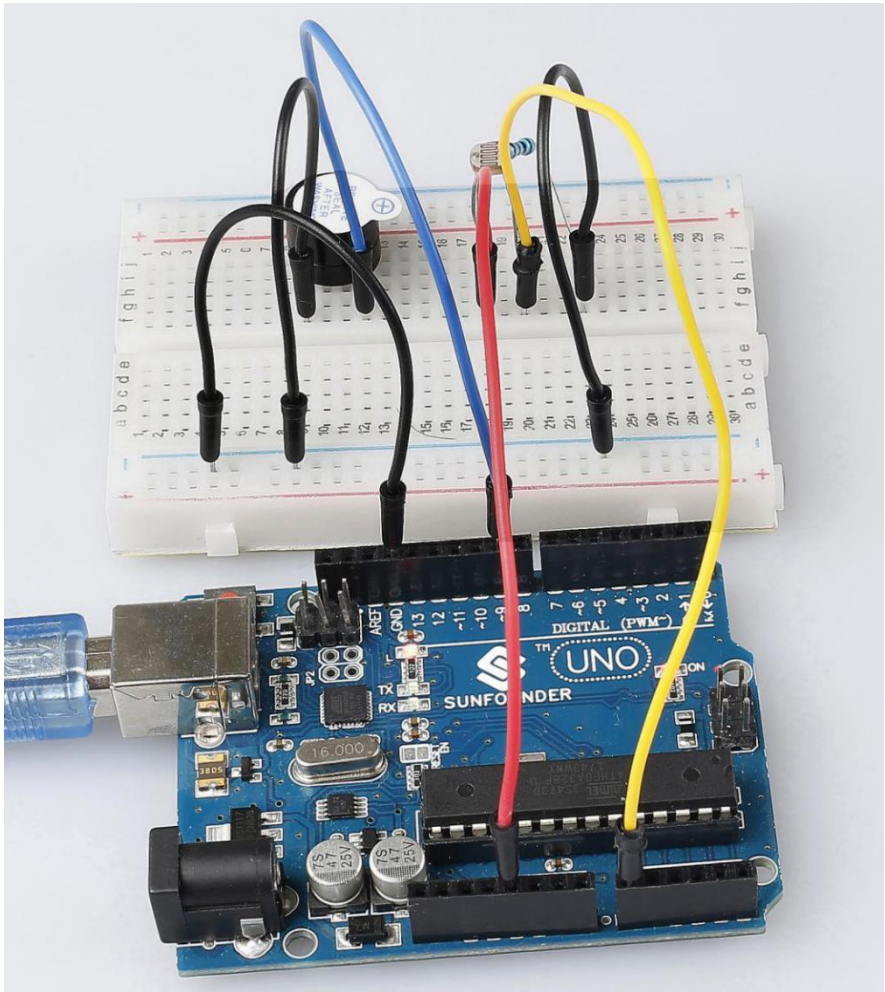
Step 2: Open the code file.

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

Step 4: Upload the sketch to the board.



Now, if you place the photoresistor in a dark environment, the buzzer will beep sharply; if you shine a flashlight on the photoresistor, the buzzer beeping will slow down.



Code Analysis

Code Analysis 20-1 Set the array elements

```
void loop()
{
  sensorValue = analogRead(photocellPin); //read the value of A0
  digitalWrite(buzzerPin, HIGH); //
  delay(sensorValue);  //wait for a while,and the delay time depend on the sensorValue
  digitalWrite(buzzerPin, LOW);
  delay(sensorValue);
}
```

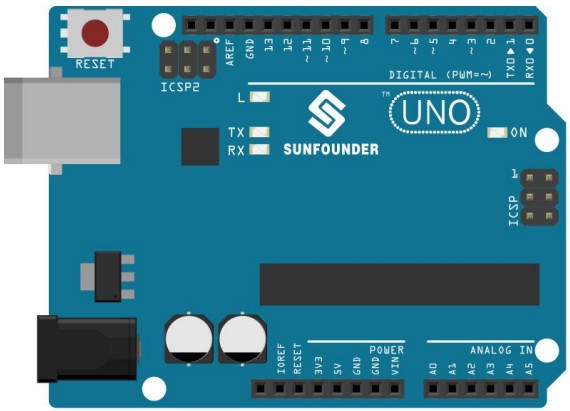
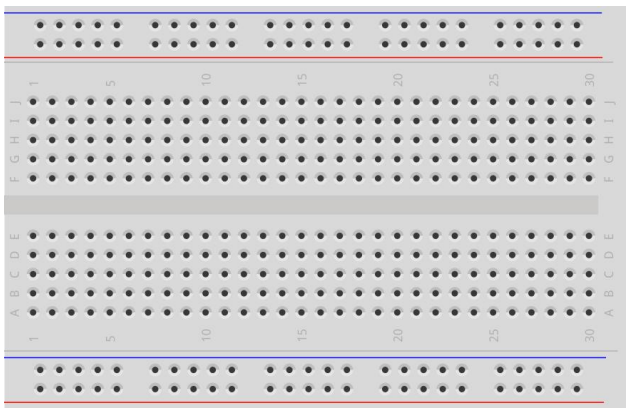
The value of the photoresistor is read, and when the incident light is strong, the output value becomes large. Then set the buzzer to high level to make it beep, delay the **sensorvalue** ms, then turn off the buzzer and also delay the **sensorvalue** ms. So you can see that if you put the photoresistor in a dark environment, the buzzer will make a sharp humming sound; if you illuminate the flashlight on the photoresistor, the buzzer will beep.

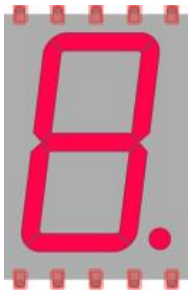
Lesson 21 74HC595

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 Uno board. Or you can connect the 74HC595 to three ports of the Uno board 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 Uno board's limited ports, this is very important. Now let's get started!

Components

1 * Uno Board 	1 * Breadboard 

1 * 7-segment display 	1 * 74HC595 	8 * Resistor (220Ω) 
1 * USB Cable 	Several Jumper Wires 	

Experimental 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.

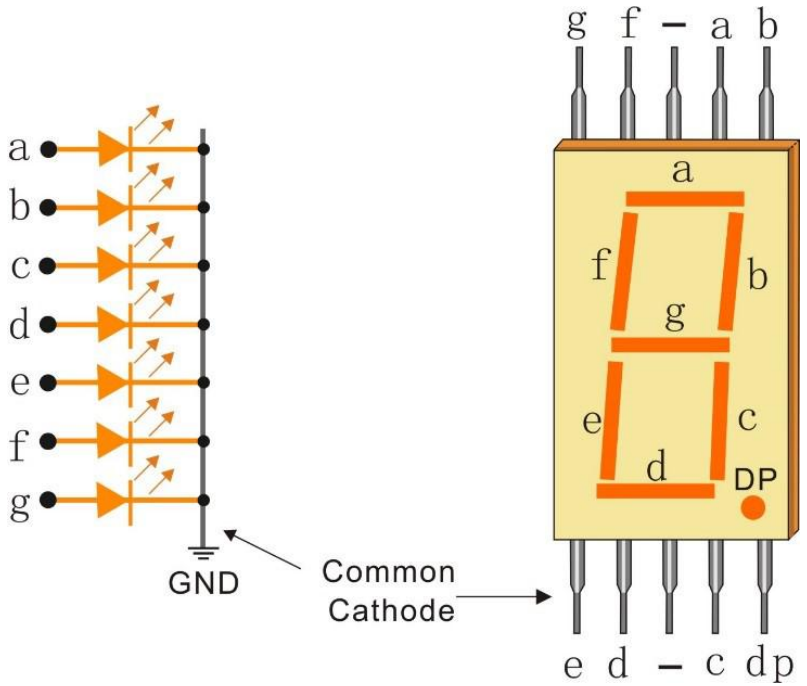
Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from "a" through to "g" representing each individual LED. The other LED pins are connected together forming a common pin. So by forward 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.

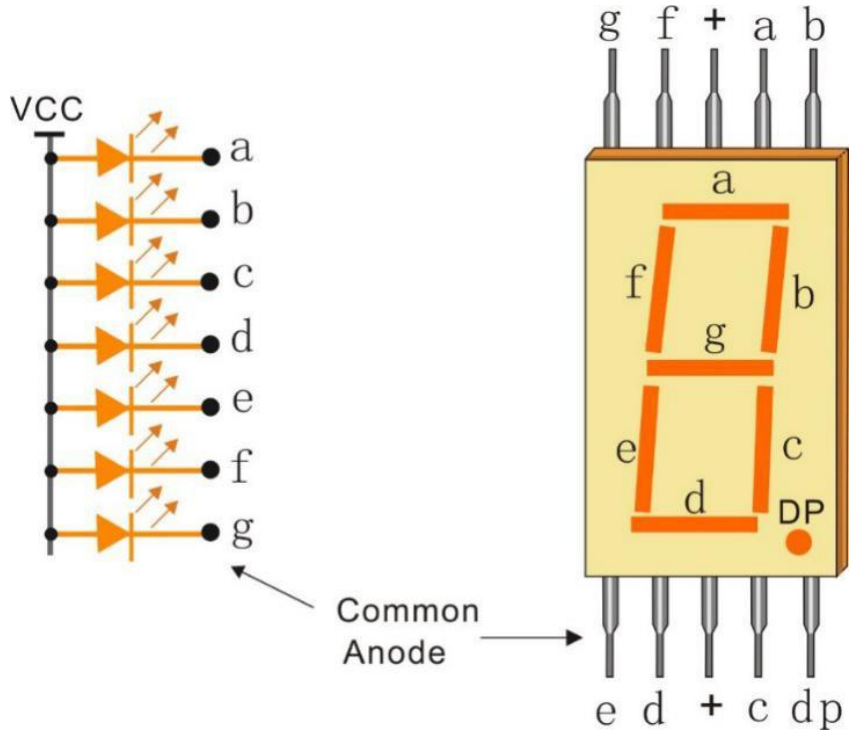
Common Cathode 7-Segment Display

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

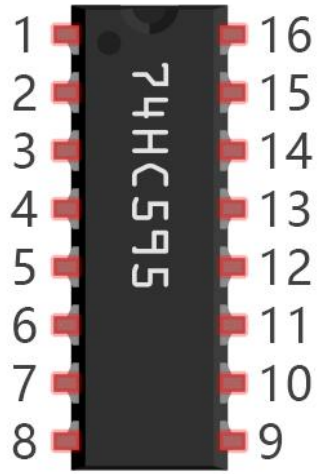
In a common anode display, the anodes of all the LED segments are connected to the logic "1". Then an individual segment (a-g) is energized by a ground, logic "0" or "LOW" signal via a current limiting resistor to the cathode of the segment.

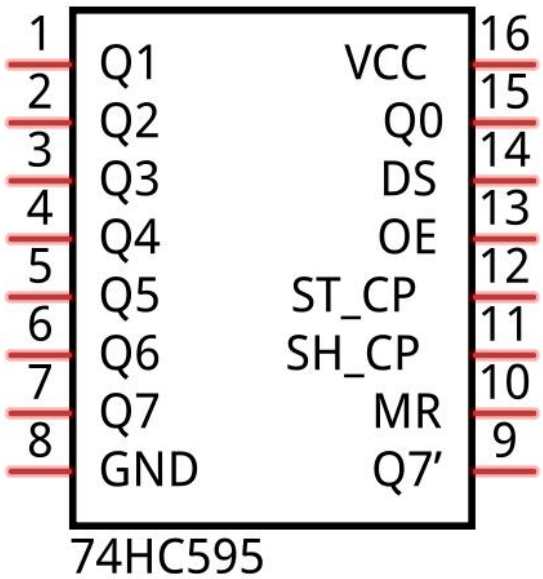


74HC595

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

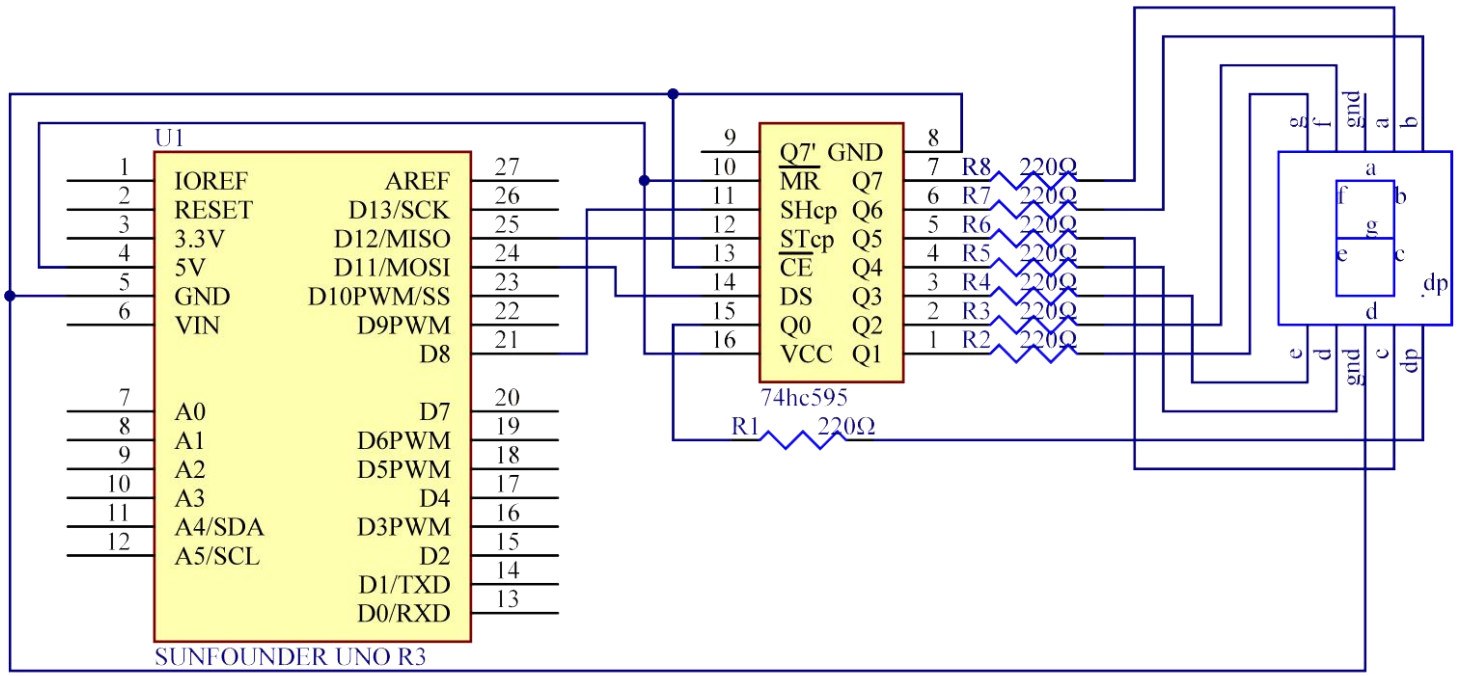
VCC: Positive supply voltage

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.

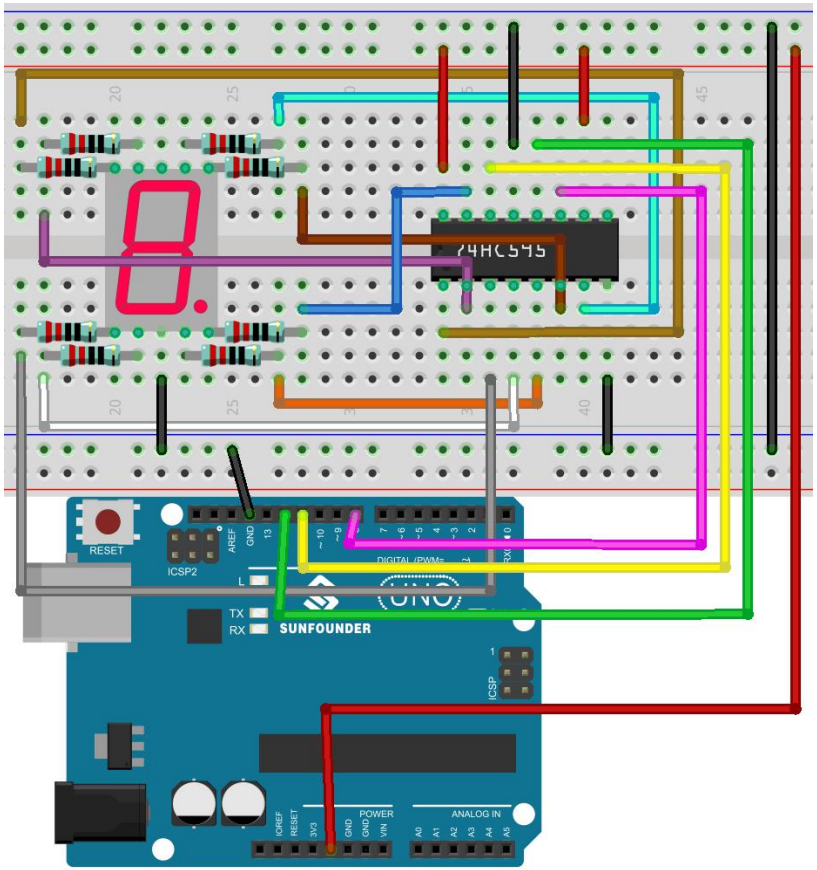
The schematic diagram:



Experimental Procedures

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

7-Segment	74HC595	Uno R3
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

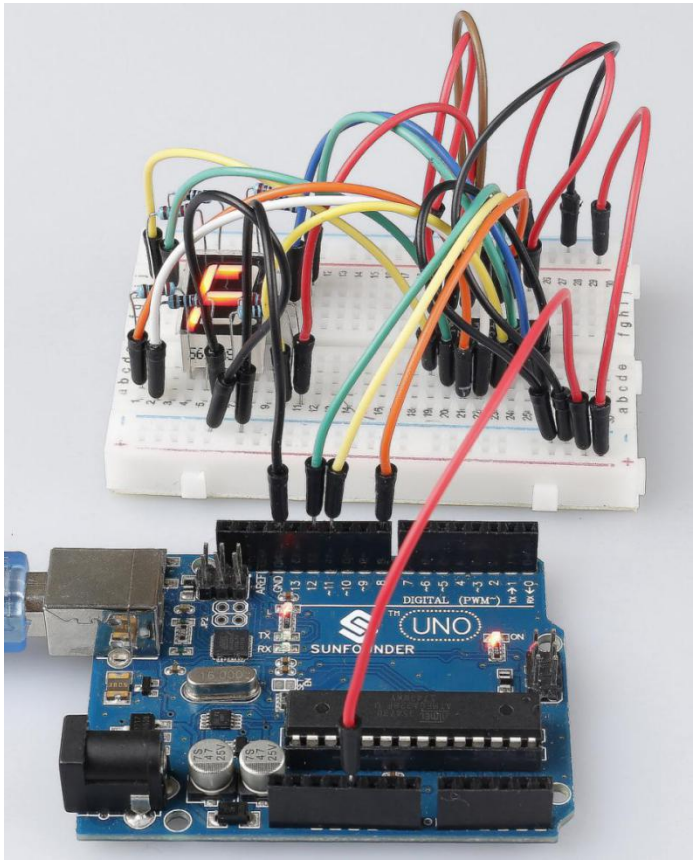


Step 2: Open the code file.

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 21-1 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. 218 stands for 2, which you can calculate by yourself. To display 2, the segment f and c of the 7-segment display must be low level (dim).

Since the segment f and c are connected to Q2 and Q5 of the 74HC595, set both Q0, Q2 and Q5 (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 0 1 1 0 1 0.

Change the binary numbers into decimal ones: $1x2^7+1x2^6+0x2^5+1x2^4+1x2^3+0x2^2+1x2^1+0x2^0=218$.

So that's the value for the number **2** to be displayed. You can calculate other characters similarly.

Code Analysis 21-2 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.

Copyright Notice