

---

# SunFounder R4 Basic Kit

[www.sunfounder.com](http://www.sunfounder.com)

Jan 29, 2024



# CONTENTS

<b>1</b>	<b>Components Introduction</b>	<b>3</b>
1.1	Arduino Uno R4 Minima . . . . .	3
1.2	Breadboard . . . . .	7
1.3	Jumper Wires . . . . .	7
1.4	Resistor . . . . .	8
1.5	Transistor . . . . .	11
1.6	Capacitor . . . . .	14
1.7	Diode . . . . .	16
1.8	74HC595 . . . . .	17
1.9	L293D . . . . .	18
1.10	LED . . . . .	19
1.11	RGB LED . . . . .	20
1.12	7-segment Display . . . . .	22
1.13	4-Digit 7-Segment Display . . . . .	24
1.14	LCD1602 . . . . .	26
1.15	Buzzer . . . . .	27
1.16	DC Motor . . . . .	28
1.17	Stepper Motor . . . . .	30
1.18	Servo . . . . .	32
1.19	Relay . . . . .	33
1.20	Power Supply Module . . . . .	35
1.21	Button . . . . .	36
1.22	Potentiometer . . . . .	37
1.23	Joystick Module . . . . .	39
1.24	Photoresistor . . . . .	41
1.25	Thermistor . . . . .	42
1.26	Tilt Switch . . . . .	43
1.27	IR Receiver Module . . . . .	44
1.28	Ultrasonic Module . . . . .	46
1.29	Humiture Sensor Module . . . . .	47
<b>2</b>	<b>Get Started with Arduino</b>	<b>49</b>
2.1	What is Arduino? . . . . .	49
2.2	What can Arduino do? . . . . .	49
2.3	How to build an Arduino Project . . . . .	50
2.4	How to add libraries? (Important) . . . . .	79
<b>3</b>	<b>Projects</b>	<b>81</b>
3.1	Lesson 1 Blinking LED . . . . .	81
3.2	Lesson 2 Flowing LED Lights . . . . .	86

3.3	Lesson 3 Controlling LED by Button . . . . .	90
3.4	Lesson 4 Doorbell . . . . .	94
3.5	Lesson 5 Tilt Switch . . . . .	99
3.6	Lesson 6 Relay . . . . .	102
3.7	Lesson 7 RGB LED . . . . .	106
3.8	Lesson 8 Controlling an LED by Potentiometer . . . . .	111
3.9	Lesson 9 Photoresistor . . . . .	118
3.10	Lesson 10 Servo . . . . .	123
3.11	Lesson 11 LCD1602 . . . . .	126
3.12	Lesson 12 Thermistor . . . . .	131
3.13	Lesson 13 Ultrasonic . . . . .	136
3.14	Lesson 14 Infrared-Receiver . . . . .	141
3.15	Lesson 15 Humiture Sensor . . . . .	144
3.16	Lesson 16 Joystick PS2 . . . . .	149
3.17	Lesson 17 7-Segment Display . . . . .	152
3.18	Lesson 18 74HC595 . . . . .	157
3.19	Lesson 19 Stepper Motor . . . . .	161
3.20	Lesson 20 Simple Creation-Stopwatch . . . . .	166
3.21	Lesson 21 Simple Creation-Answer Machine . . . . .	172
3.22	Lesson 22 Simple Creation-Small Fan . . . . .	176
3.23	Lesson 23 Simple Creation - Digital Dice . . . . .	181
<b>4</b>	<b>Thank You</b>	<b>187</b>
<b>5</b>	<b>Copyright Notice</b>	<b>189</b>



The Arduino Uno R4 Basic Kit is an all-inclusive package designed for enthusiasts and beginners alike to explore the world of Arduino. This kit provides everything you need to dive into the exciting world of electronics and programming with the Arduino Uno board.

With the Arduino Uno R4 Basic Kit, you can learn and experiment with a wide range of electronic components and modules. The kit includes various sensors, actuators, and displays, allowing you to build projects that interact with the physical world. Whether you want to create a temperature and humidity monitoring system, a motion-activated LED display, or a robotic arm, this kit provides the essential components to bring your ideas to life.

The kit comes with comprehensive documentation and tutorials that guide you through the process of setting up the Arduino Uno board, installing the necessary software, and understanding the basics of programming with Arduino. The included code examples and step-by-step instructions make it easy to learn and build exciting projects.

Additionally, the Arduino Uno R4 Basic Kit provides access to datasheets for all the components, Fritzing diagrams for visualizing the circuit connections, and libraries that enhance the functionality of the Arduino platform.

Whether you are a beginner or an experienced maker, the Arduino Uno R4 Basic Kit is an excellent choice to unleash your creativity, expand your knowledge in electronics and programming, and embark on countless exciting projects.

You can download the code package for the SunFounder Uno R4 Basic Kit from the following link:

- [SunFounder Uno R4 Basic Kit](#)
- Or check out the code at [.](#)

If you have any questions, please send an email to [service@sunfounder.com](mailto:service@sunfounder.com) and we will respond as soon as possible.



## **COMPONENTS INTRODUCTION**

Below is the introduction to each component, which contains the operating principle of the component and the corresponding projects.

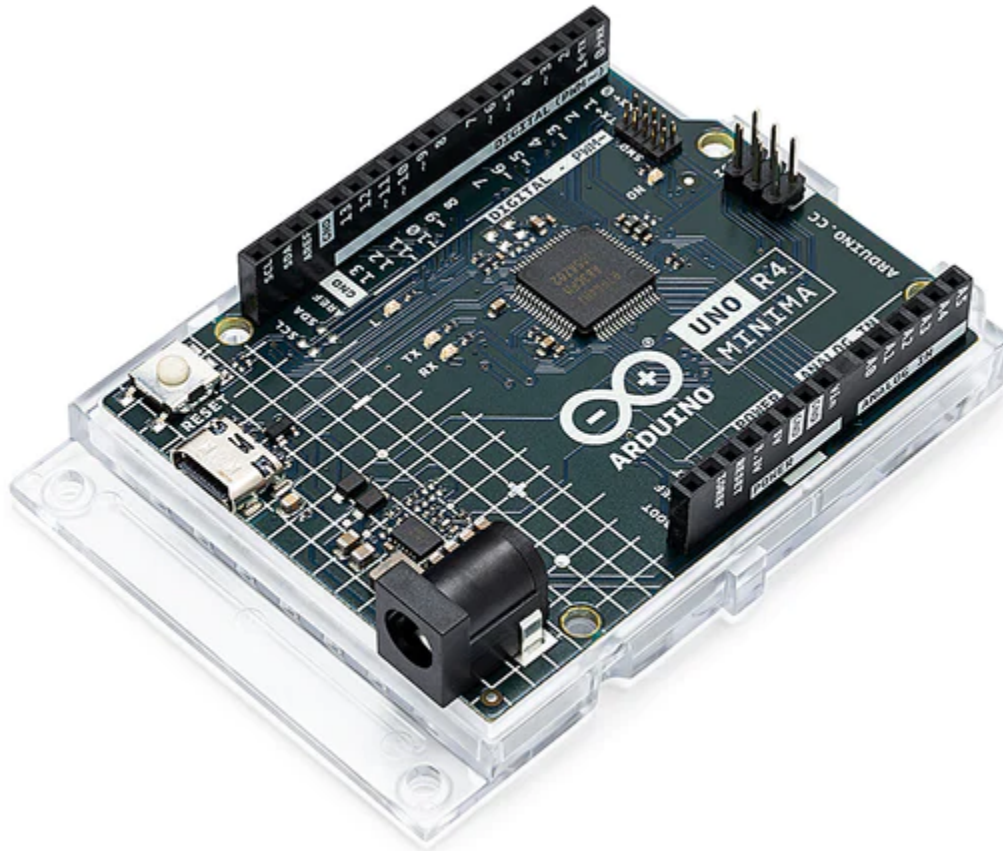
### **Basic**

## **1.1 Arduino Uno R4 Minima**

### **Overview**

Enhanced and improved, the Arduino UNO R4 Minima is armed with a powerful 32-bit microcontroller courtesy of Renesas. Brace yourself for increased processing power, expanded memory, and a whole new level of on-board peripherals. The best part? Compatibility with existing shields and accessories remains intact, and there's no need to make any changes to the standard form factor or 5 V operating voltage.

Joining the Arduino ecosystem, the UNO R4 is a trusty addition suitable for both beginners and seasoned electronics enthusiasts. Whether you're just starting out or looking to push the boundaries of your projects, this robust board delivers reliable performance every time.



Here's what the UNO R4 Minima brings to the table:

- **Hardware compatibility with UNO form factor:** The UNO R4 Minima maintains the same form factor, pinout, and 5 V operating voltage as its predecessor, the UNO R3. This ensures a seamless transition for existing shields and projects, leveraging the extensive and unique ecosystem already established for the Arduino UNO.
- **Expanded memory and faster clock:** Prepare for more precise calculations and the ability to handle complex projects with ease. The UNO R4 Minima boasts increased memory and a faster clock speed, empowering you to tackle demanding tasks effortlessly.
- **Extra on-board peripherals:** The UNO R4 Minima introduces a range of on-board peripherals, including a 12-bit DAC, CAN BUS, and OP AMP. These additional components provide you with expanded capabilities and flexibility in your designs.
- **Extended 24 V tolerance:** The UNO R4 Minima now supports a wider input voltage range, allowing power supplies up to 24 V. This enables seamless integration with motors, LED strips, and other actuators, simplifying your projects by utilizing a single power source.
- **SWD connector:** Debugging is a critical aspect of any project. Simply connect an external debugger to the UNO R4 Minima and effortlessly monitor the inner workings of your system. Stay in control and gain valuable insights.
- **HID support:** The UNO R4 Minima comes with built-in HID (Human Interface Device) support, enabling it to simulate a mouse or keyboard when connected to a computer via a USB cable. This convenient feature makes it a breeze to send keystrokes and mouse movements to a computer, enhancing usability and functionality.

#### Tech specs

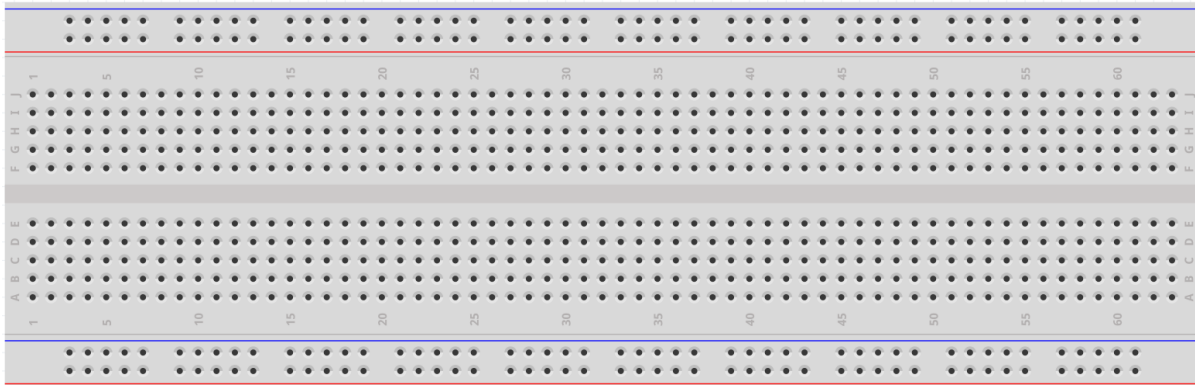
- **Board:**
  - Name: Arduino® UNO R4 Minima

- SKU: ABX00080
- Microcontroller: Renesas RA4M1 (Arm® Cortex®-M4)
- USB: USB-C® Programming Port
- **Pins:**
  - Digital I/O Pins: 14
  - Analog input pin: 6
  - DAC: 1
  - PWM pins: 6
- **Communication**
  - UART: Yes, 1x
  - I2C: Yes, 1x
  - SPI: Yes, 1x
  - CAN: Yes 1 CAN Bus
- **Power**
  - Circuit operating voltage: 5 V
  - Input voltage (VIN): 6-24 V
  - DC Current per I/O Pin: 8 mA
- Clock speed Main core 48 MHz
- Memory RA4M1 256 kB Flash, 32 kB RAM
- **Dimensions**
  - Width: 68.85 mm
  - Length: 53.34 mm

## **Pinout**



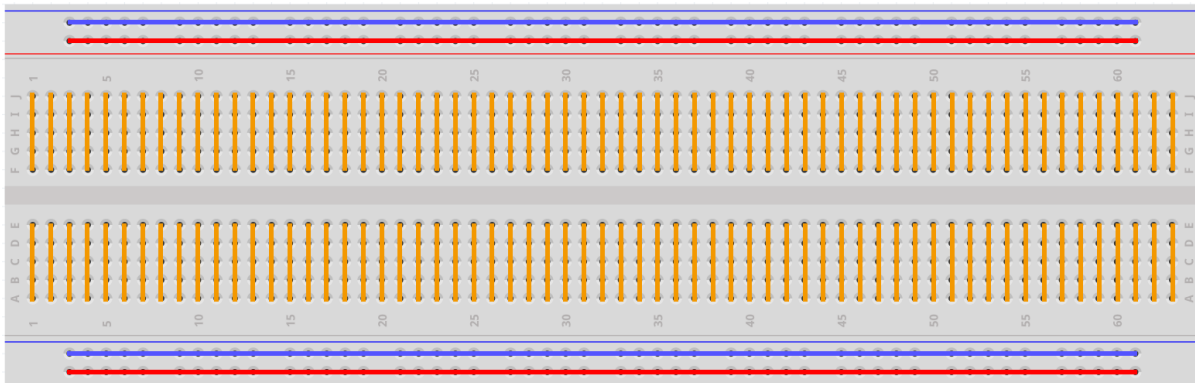
## 1.2 Breadboard



A breadboard is a construction base for prototyping of electronics. Originally the word referred to a literal bread board, a polished piece of wood used for slicing bread.[1] In the 1970s the solderless breadboard (a.k.a. plugboard, a terminal array board) became available and nowadays the term “breadboard” is commonly used to refer to these.

It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

The picture shows the internal structure of a breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



If you want to know more about breadboard, refer to: [How to Use a Breadboard - Science Buddies](#)

## 1.3 Jumper Wires

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

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

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

both side are male and Female-to-Female means both ends are female.



---

**Note:**

- 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.
- 

## 1.4 Resistor



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

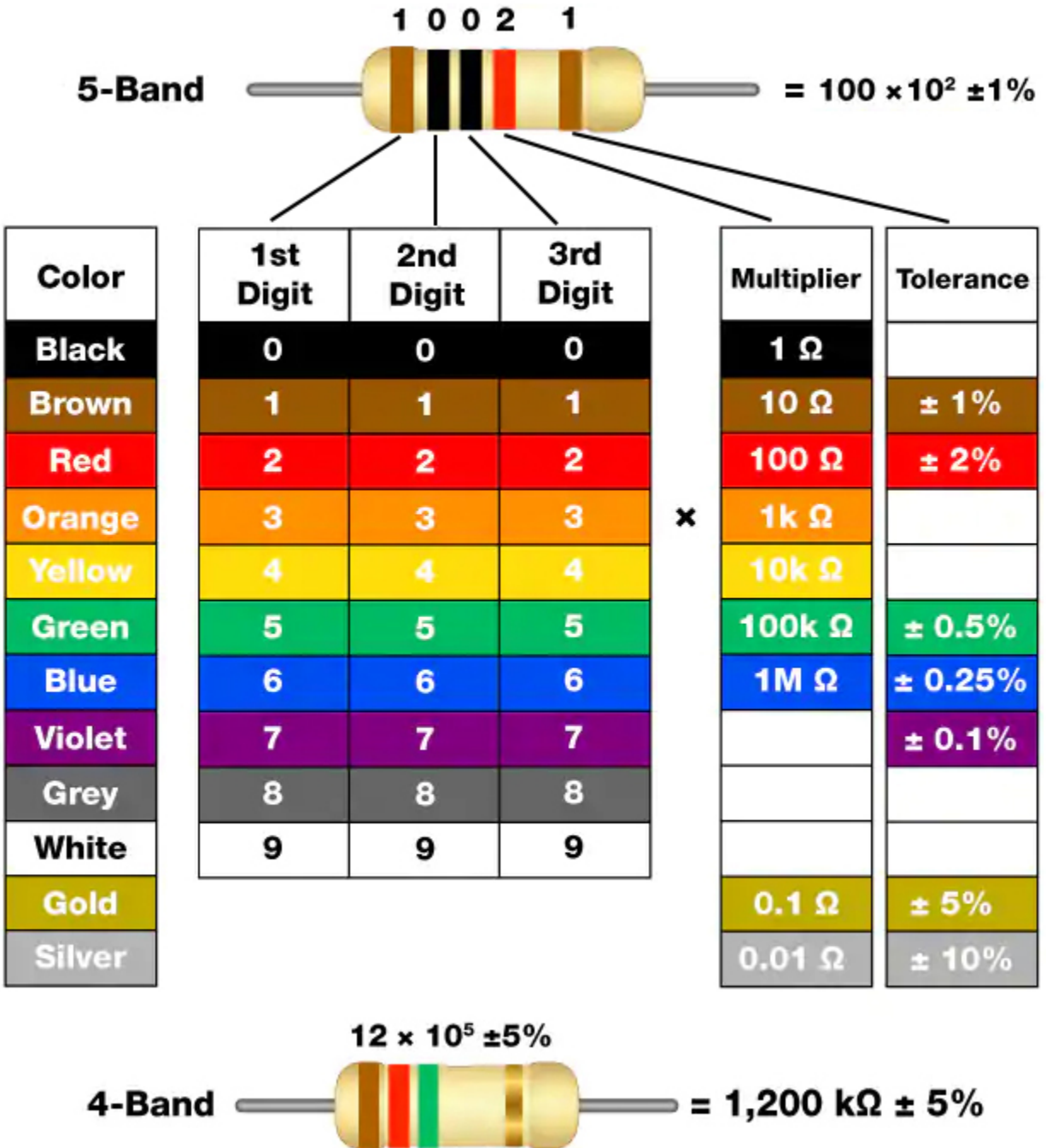
Two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.



is the unit of resistance and the larger units include K, M, etc. Their relationship can be shown as follows: 1 M=1000 K, 1 K = 1000 . Normally, the value of resistance is marked on it.

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.





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

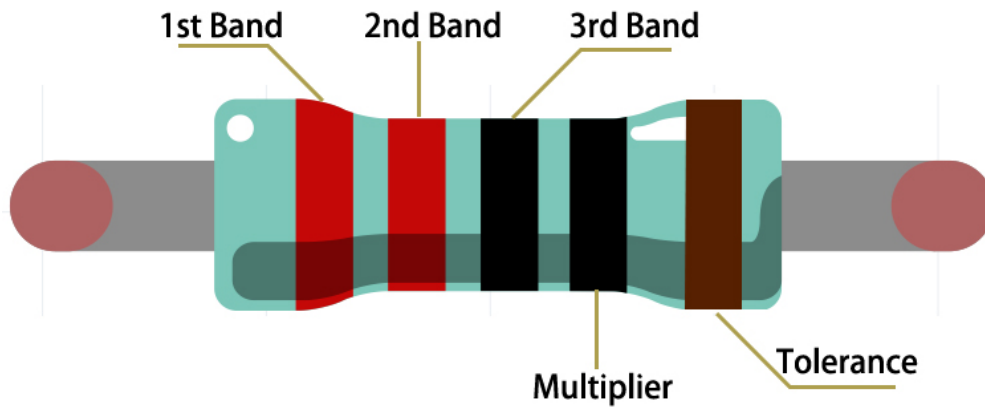
Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White	Gold	Silver
0	1	2	3	4	5	6	7	8	9	0.1	0.01

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands.

Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4th and 5th band will be comparatively larger.

Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.

Let's see how to read the resistance value of a 5-band resistor as shown below.



So for this resistor, the resistance should be read from left to right. The value should be in this format: 1st Band 2nd Band 3rd Band x  $10^{\text{Multiplier}}$  and the permissible error is  $\pm \text{Tolerance}\%$ . So the resistance value of this resistor is  $2(\text{red})\ 2(\text{red})\ 0(\text{black}) \times 10^0(\text{black}) = 220$ , and the permissible error is  $\pm 1\%$  (brown).

## Common resistor color band

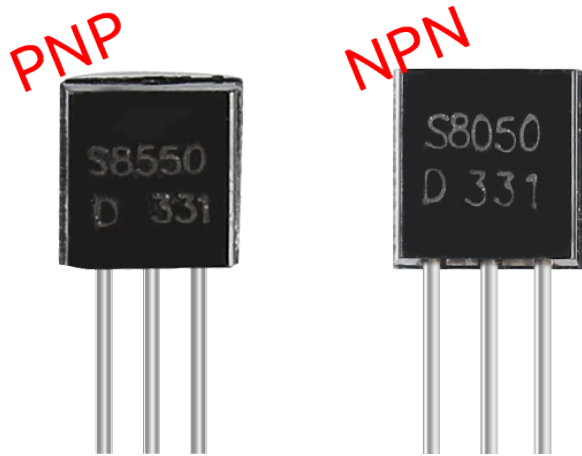
Resistor	Color Band
10	brown black black silver brown
100	brown black black black brown
220	red red black black brown
330	orange orange black black brown
1k	brown black black brown brown
2k	red black black brown brown
5.1k	green brown black brown brown
10k	brown black black red brown
100k	brown black black orange brown
1M	brown black black green brown

You can learn more about resistor from Wiki: [Resistor - Wikipedia](#).

## Example

- [Lesson 1 Blinking LED](#) (R4 Minima Board Project)
- [Lesson 3 Controlling LED by Button](#) (R4 Minima Board Project)

## 1.5 Transistor



Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch.

A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier. From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction.

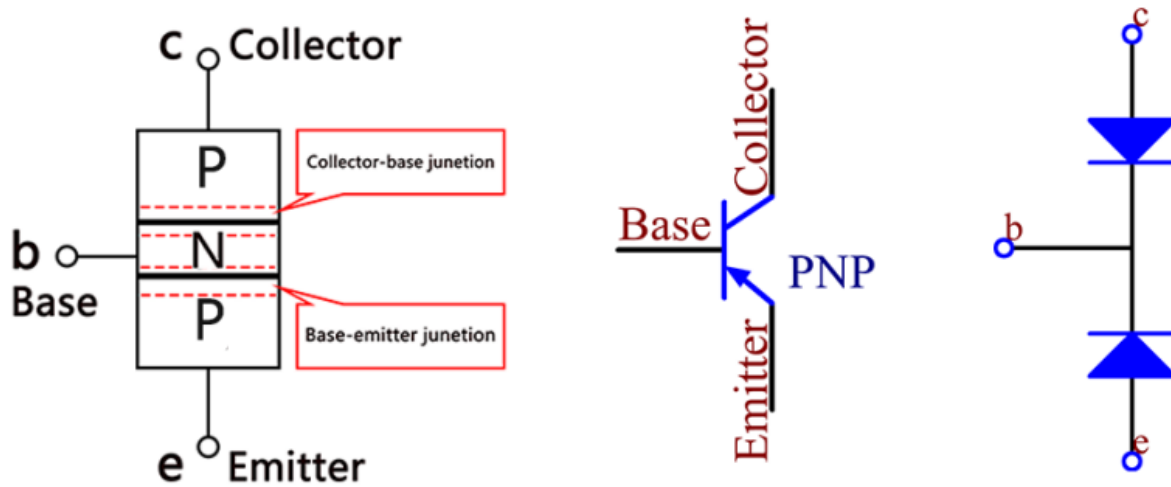
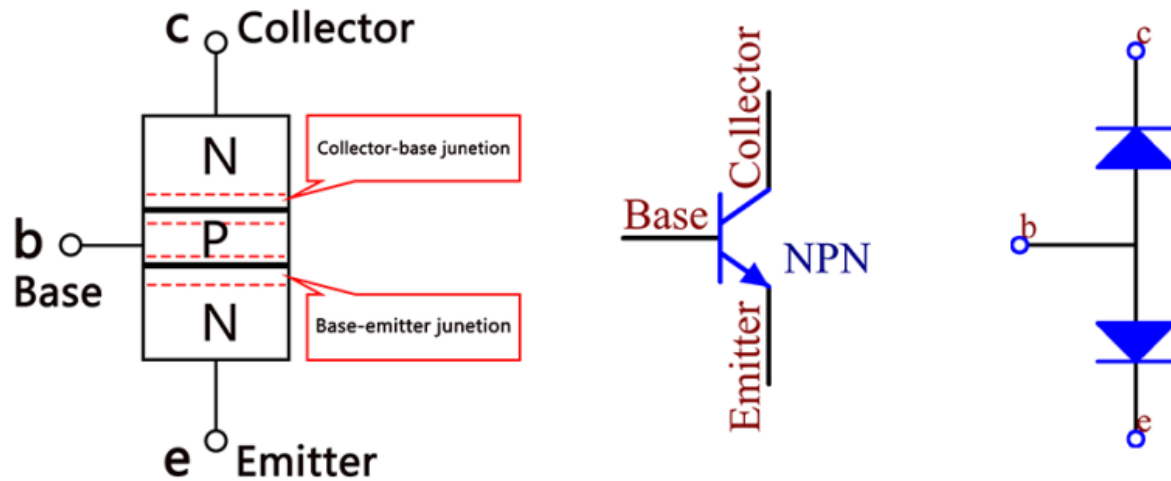
- [P-N junction - Wikipedia](#)

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.

---

**Note:** s8550 is PNP transistor and the s8050 is the NPN one, They look very similar, and we need to check carefully to see their labels.

---



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.

Put the label side facing us and the pins facing down. The pins from left to right are emitter(e), base(b), and collector(c).



- [S8050 Transistor Datasheet](#)
- [S8550 Transistor Datasheet](#)

## 1.6 Capacitor





Capacitor, refers to the amount of charge storage under a given potential difference, denoted as  $C$ , and the international unit is farad (F). Generally speaking, electric charges move under force in an electric field. When there is a medium between conductors, the movement of electric charges is hindered and the electric charges accumulate on the conductors, resulting in accumulation of electric charges.

The amount of stored electric charges is called capacitance. Because capacitors are one of the most widely used electronic components in electronic equipment, they are widely used in direct current isolation, coupling, bypass, filtering, tuning loops, energy conversion, and control circuits. Capacitors are divided into electrolytic capacitors, solid capacitors, etc.

According to material characteristics, capacitors can be divided into: aluminum electrolytic capacitors, film capacitors, tantalum capacitors, ceramic capacitors, super capacitors, etc.

- [Ceramic Capacitor - Wikipedia](#)
- [Electrolytic Capacitor - Wikipedia](#)

There are 103 or 104 label on the ceramic capacitors, which represent the capacitance value,  $103=10 \times 10^3 \text{pF}$ ,  $104=10 \times 10^4 \text{pF}$

#### Unit Conversion

$$1\text{F}=10^3\text{mF}=10^6\mu\text{F}=10^9\text{nF}=10^{12}\text{pF}$$

#### Example

- [Lesson 4 Doorbell](#) (R4 Minima Board Project)

## 1.7 Diode

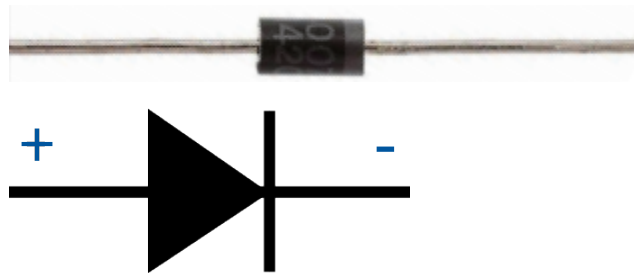
A diode is an electronic component with two electrodes. It allows current to flow in only one direction, which is often called the “Rectifying” function. Thus, a diode can be thought of as an electronic version of a check valve.

Because of its unidirectional conductivity, the diode is used in almost all electronic circuits of some complexity. It is one of the first semiconductor devices and has a wide range of applications.

According to its use classification, it can be divided into detector diodes, rectifier diodes, limiter diodes, voltage regulator diodes, etc.

Rectifier diodes and voltage regulator diodes are included in this kit.

### Rectifier Diode



A rectifier diode is a semiconductor diode, used to rectify AC (alternating current) to DC (direct current) using the rectifier bridge application. The alternative of rectifier diode through the Schottky barrier is mainly valued within digital electronics. This diode is capable to conduct the values of current which changes from mA to a few kA & voltages up to a few kV.

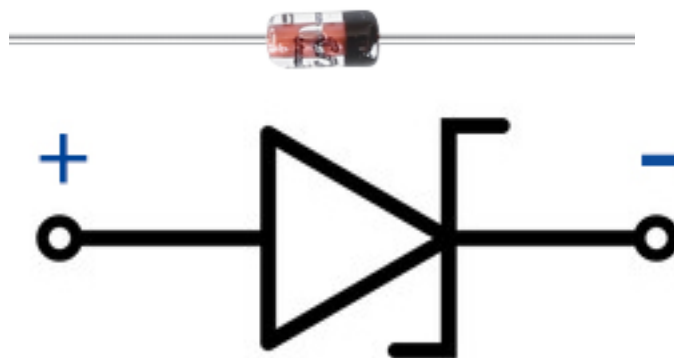
The designing of rectifier diodes can be done with Silicon material and they are capable of conducting high electric current values. These diodes are not famous but still used Ge or gallium arsenide-based semiconductor diodes. Ge diodes have less allowable reversed voltage as well as a lesser allowable junction temperature. The Ge diode has a benefit as compared to Si diode that is low threshold voltage value while operating in a forward-bias.

- [1N400x general-purpose diode - Wikipedia](#)

### Zener Diode

A Zener diode is a special type of diode designed to reliably allow current to flow “backwards” when a certain set reverse voltage, known as the Zener voltage, is reached.

This diode is a semiconductor device that has a very high resistance up to the critical reverse breakdown voltage. At this critical breakdown point, the reverse resistance is reduced to a very small value, and the current increases while the voltage remains constant in this low resistance region.



- [Zener diode - Wikipedia](#)



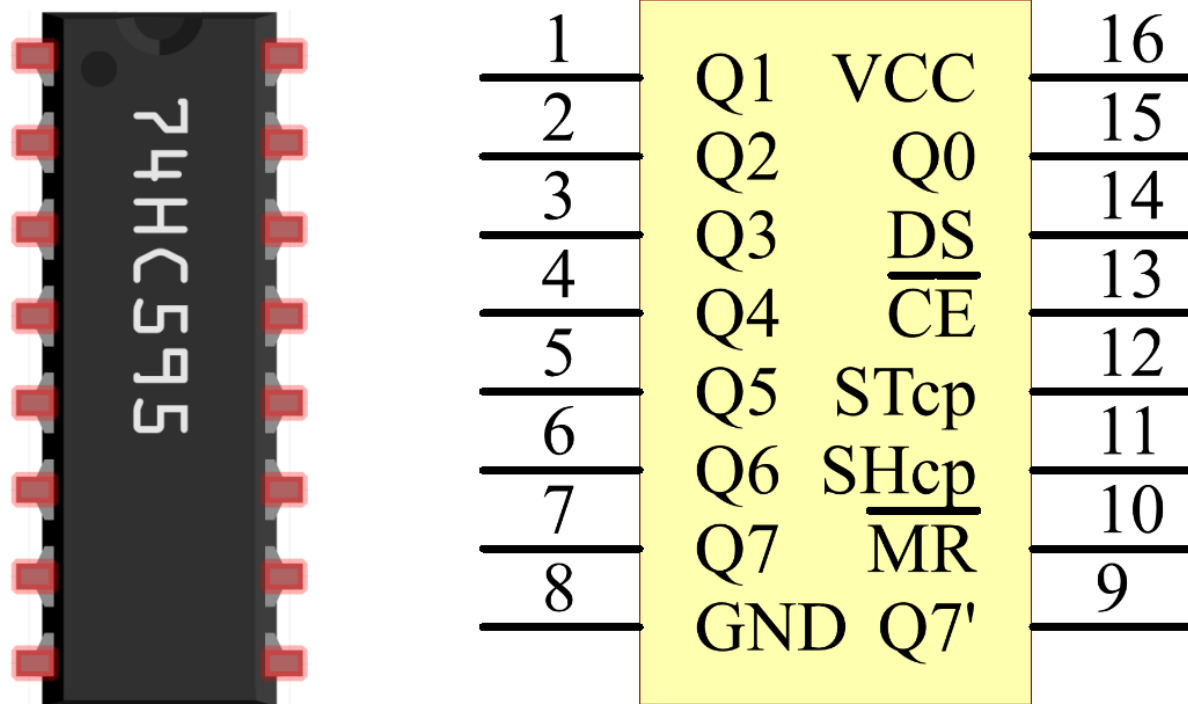
**Example**

- *Lesson 6 Relay* (R4 Minima Board Project)

**Chip****1.8 74HC595**

The 74HC595 consists of an 8bit shift register and a storage register with threestate 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.

- [74HC595 Datasheet](#)



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;
- **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.

- **STep**: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.
- **CE**: Output enable pin, active at low level.
- **DS**: Serial data input pin
- **VCC**: Positive supply voltage.
- **GND**: Ground.

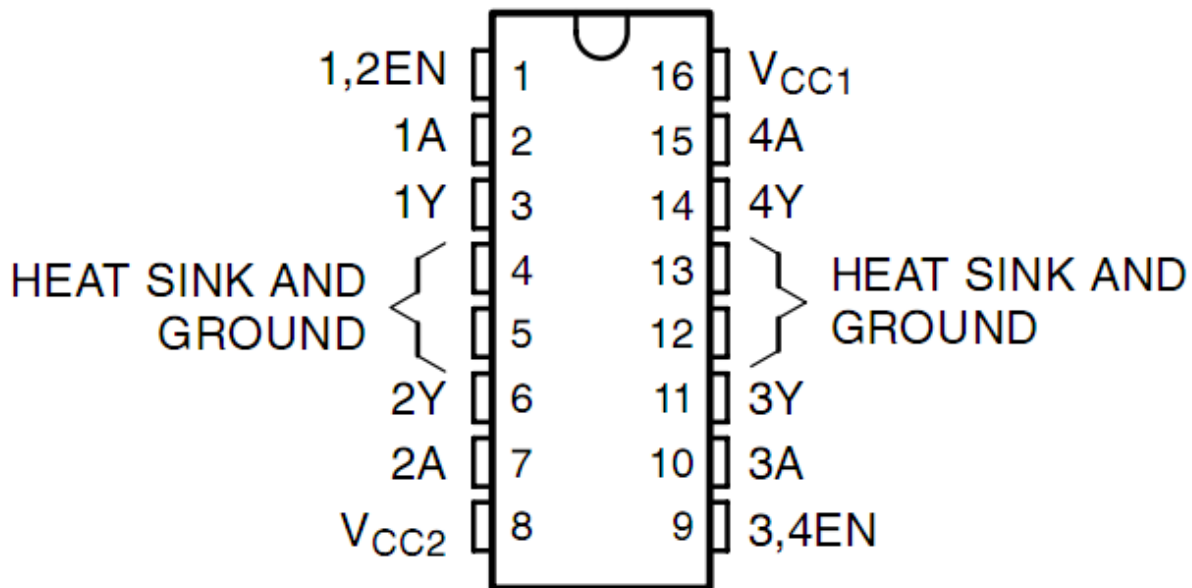
#### Example

- *Lesson 18 74HC595* (R4 Minima Board Project)
- *Lesson 23 Simple Creation - Digital Dice* (R4 Minima Board Project)

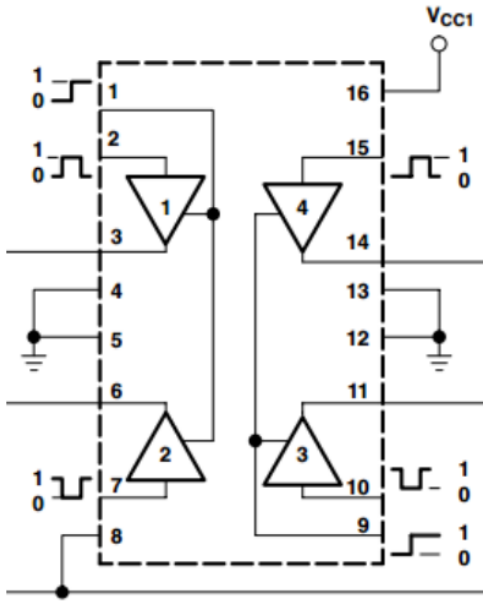
## 1.9 L293D

L293D is a 4-channel motor driver integrated by chip with high voltage and high current. It's designed to connect to standard DTL, TTL logic level, and drive inductive loads (such as relay coils, DC, Stepper Motors) and power switching transistors etc. DC Motors are devices that turn DC electrical energy into mechanical energy. They are widely used in electrical drive for their superior speed regulation performance.

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



The following is the internal structure of L293D. Pin EN is an enable pin and only works with high level; A stands for input and Y for output. You can see the relationship among them at the right bottom. When pin EN is High level, if A is High, Y outputs high level; if A is Low, Y outputs Low level. When pin EN is Low level, the L293D does not work.



INPUTS†		OUTPUT Y
A	EN	
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,  
Z = high impedance (off)

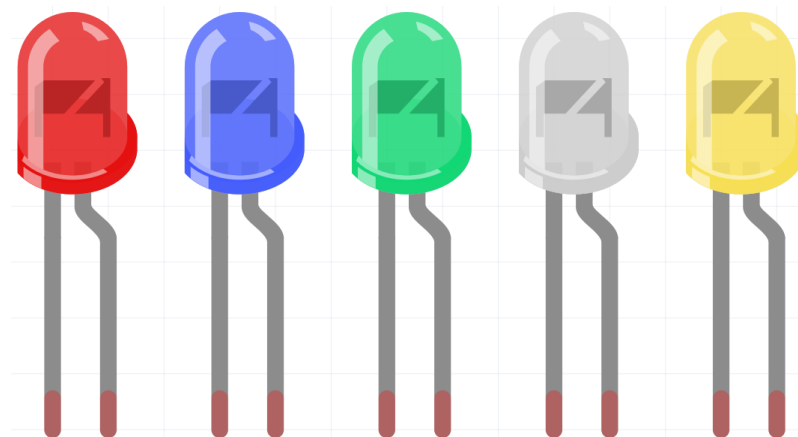
- [L293D Datasheet](#)

### Example

- *Lesson 22 Simple Creation-Small Fan* (R4 Minima Board Project)

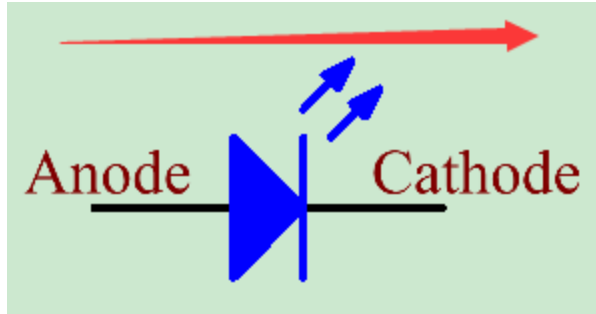
### Display

## 1.10 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:

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

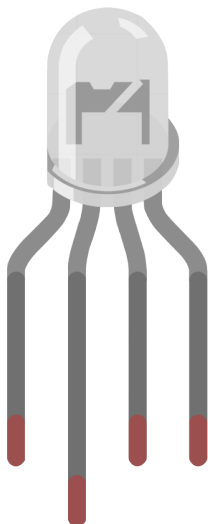
**R** stands for the resistance value of the current limiting resistor, **V<sub>supply</sub>** for voltage supply, **V<sub>D</sub>** for voltage drop and **I** for the working current of the LED.

Here is the detailed introduction for the LED: [LED - Wikipedia](#).

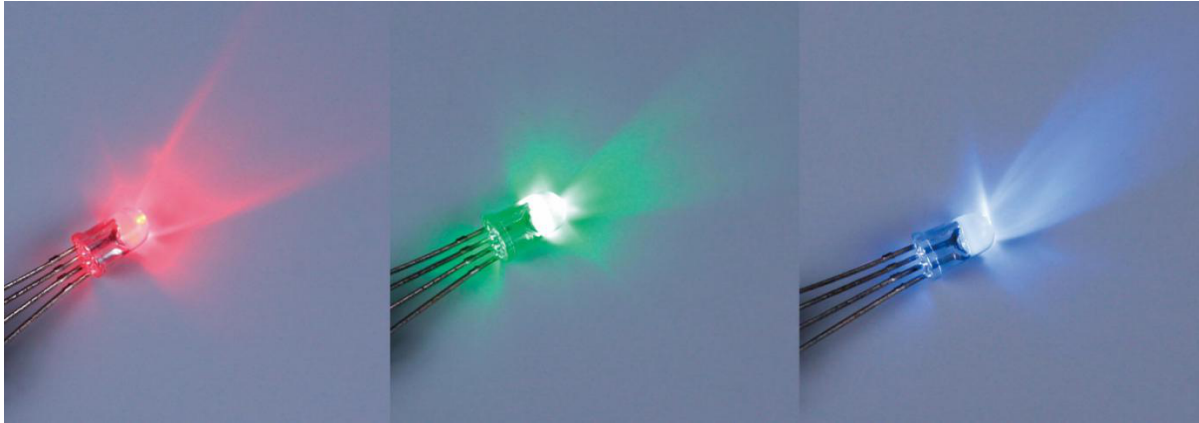
#### Example

- [Lesson 1 Blinking LED](#) (R4 Minima Board Project)
- [Lesson 8 Controlling an LED by Potentiometer](#) (R4 Minima Board Project)

## 1.11 RGB LED

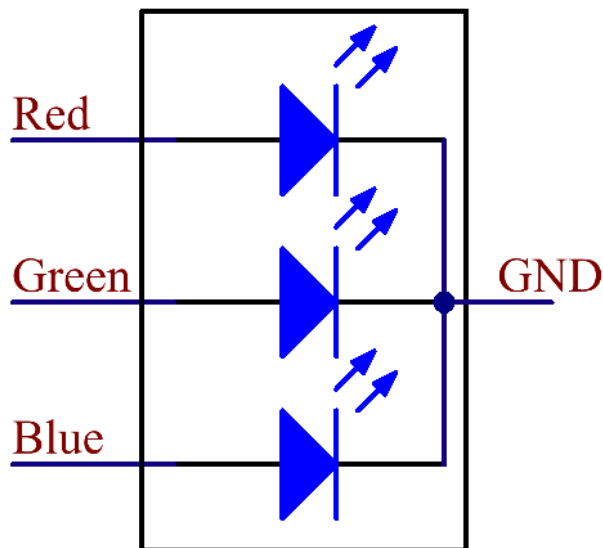


RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216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

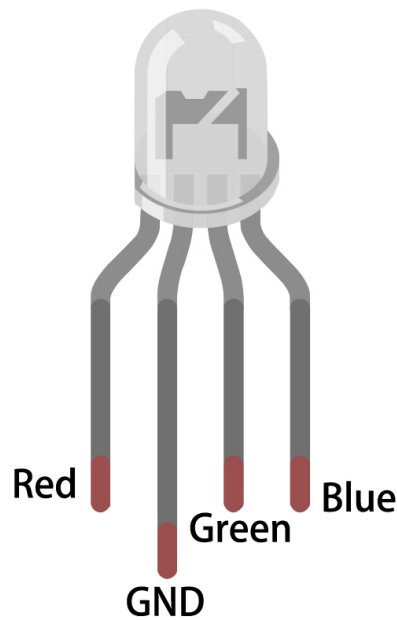


RGB LEDs can be categorized into common anode and common cathode ones. In this kit, the latter is used. The **common cathode**, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

Its circuit symbol is shown as figure.



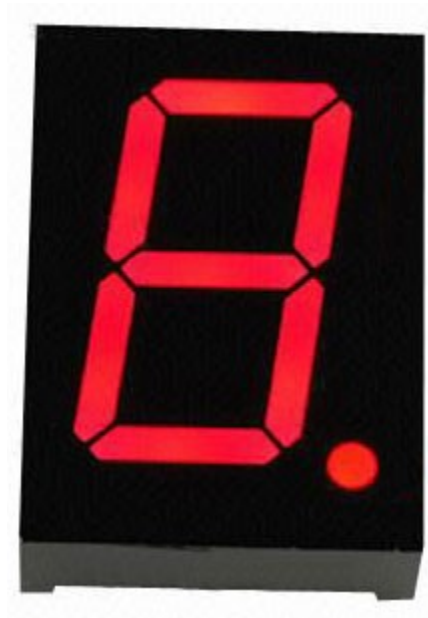
An RGB LED has 4 pins: the longest one is GND; the others are Red, Green and Blue. Touch its plastic shell and you will find a cut. The pin closest to the cut is the first pin, marked as Red, then GND, Green and Blue in turn.

**Example**

- *Lesson 7 RGB LED* (R4 Minima Board Project)

## 1.12 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.



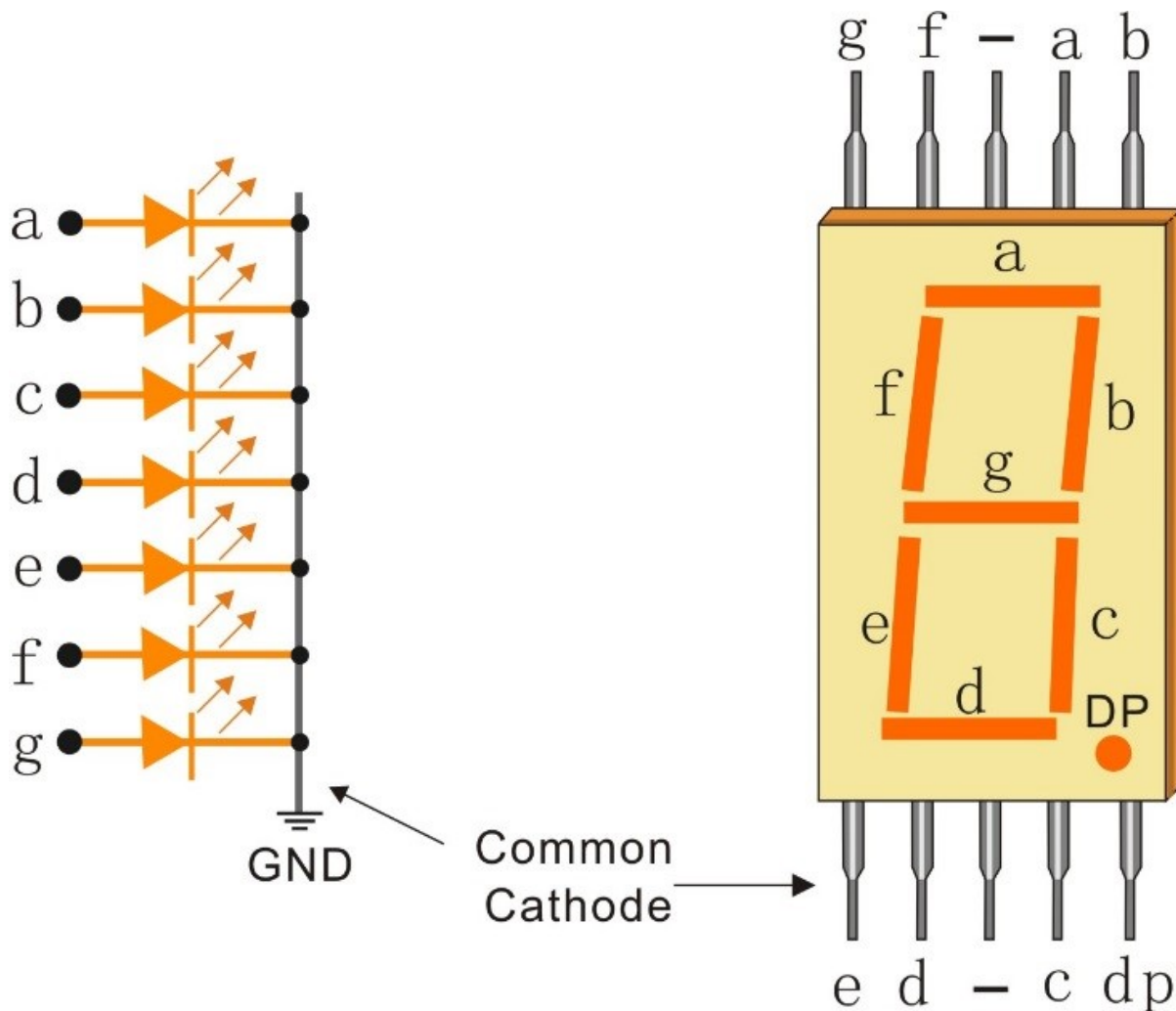
Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular

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.

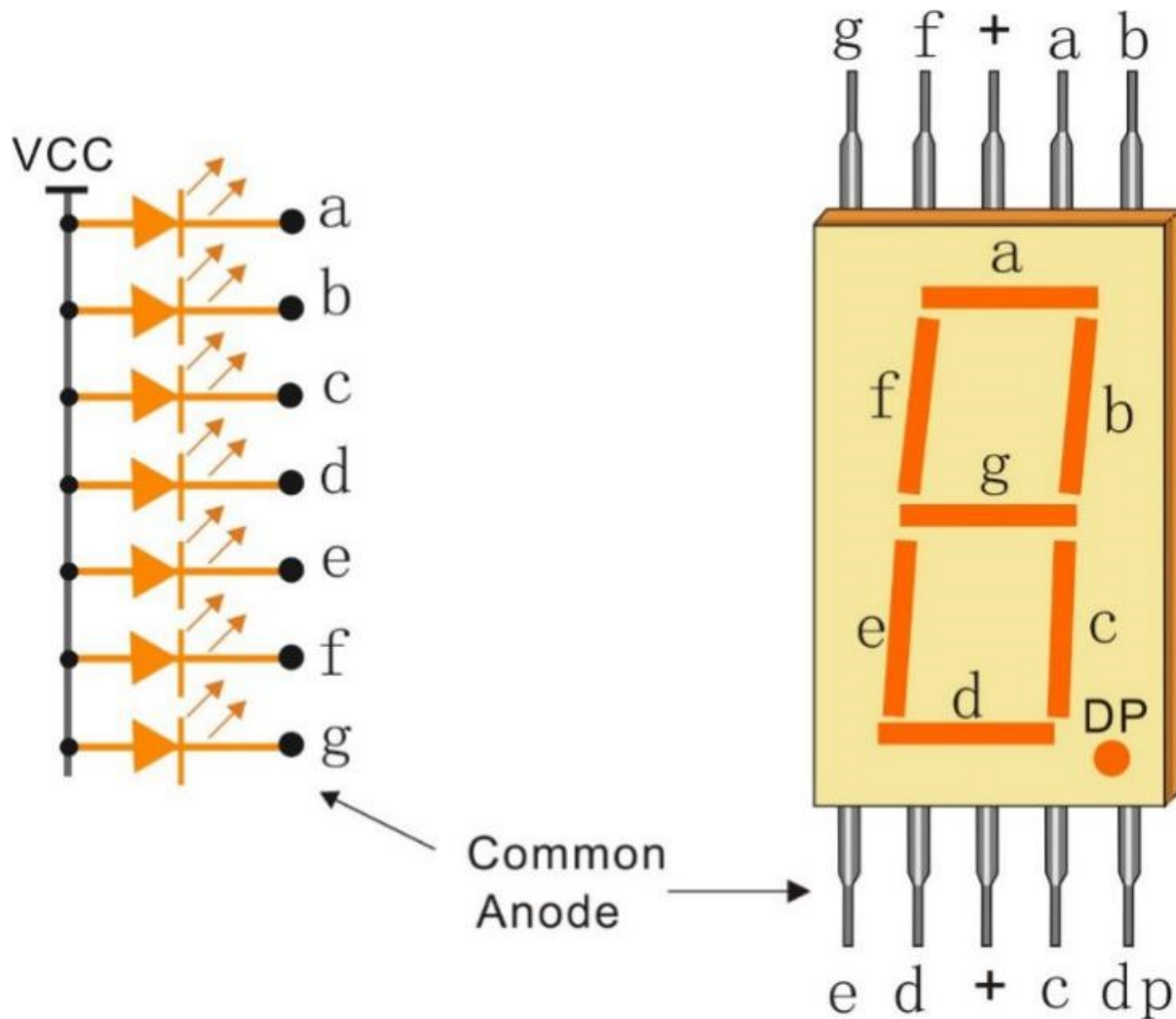
### Common Cathode 7-Segment Display

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-imgg) 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.

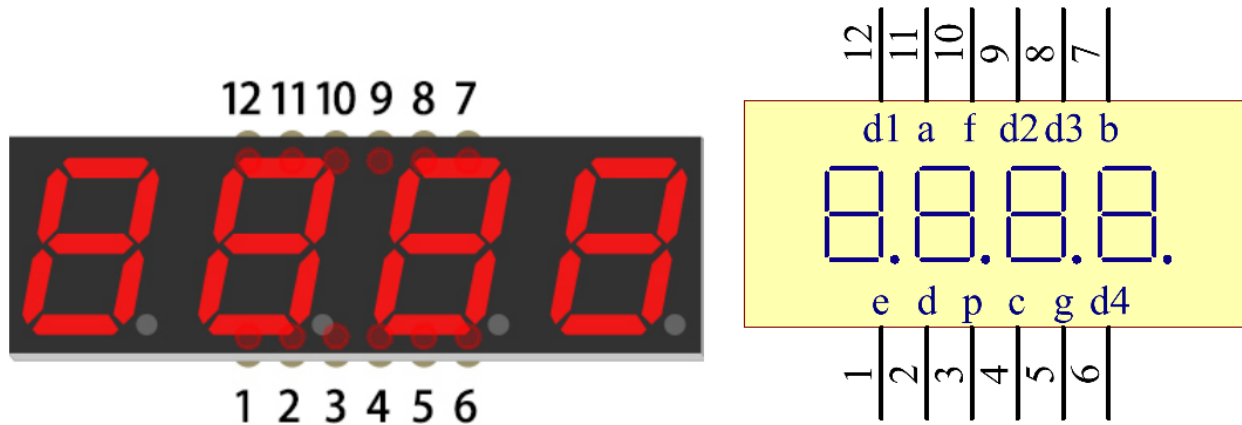
**Example**

- *Lesson 17 7-Segment Display* (R4 Minima Board Project)
- *Lesson 23 Simple Creation - Digital Dice* (R4 Minima Board Project)

## 1.13 4-Digit 7-Segment Display

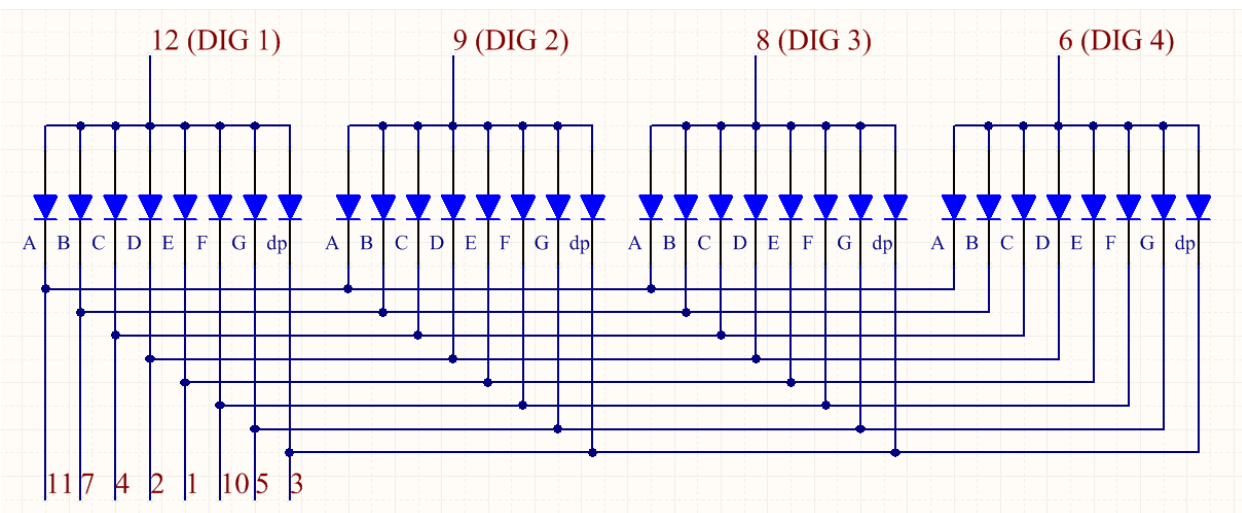
4-Digit 7-segment display consists of four 7-segment displays working together.





The 4-digit 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when “1234” is displayed on the display, “1” is displayed on the first 7-segment, and “234” is not displayed. After a period of time, the second 7-segment shows “2”, the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.



### Display Codes

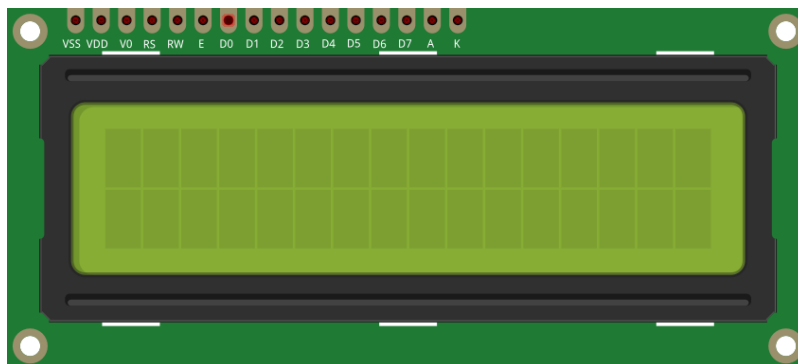
To help you get to know how 7-segment displays(Common Anode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 11000000 means that DP and G are set to 1, while others are set to 0. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Numbers	Common Anode		Numbers	Common Anode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA A	Hex Code
0	11000000	0xc0	A	10001000	0x88
1	11111001	0xf9	B	10000011	0x83
2	10100100	0xa4	C	11000110	0xc6
3	10110000	0xb0	D	10100001	0xa1
4	10011001	0x99	E	10000110	0x86
5	10010010	0x92	F	10001110	0x8e
6	10000010	0x82	.	01111111	0x7f
7	11111000	0xf8			
8	10000000	0x80			
9	10010000	0x90			

**Example**

- *Lesson 20 Simple Creation-Stopwatch* (R4 Minima Board Project)

## 1.14 LCD1602



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!

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be cate-

gorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the control 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.

### Pinouts

- **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.

### Example

- *Lesson 11 LCD1602* (R4 Minima Board Project)
- *Lesson 15 Humiture Sensor* (R4 Minima Board Project)

### Sound

## 1.15 Buzzer



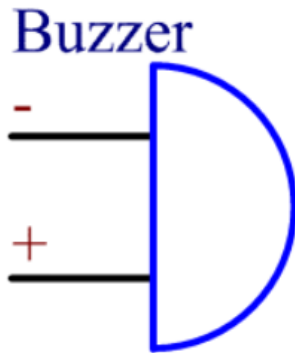
As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices.

Buzzers can be categorized as active and passive ones. Turn the buzzer so that its pins are facing up, and the buzzer with a green circuit board is a passive buzzer, while the one 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 beep 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.

The following is the electrical symbol of a buzzer. It has two pins with positive and negative poles. With a + in the surface represents the anode and the other is the cathode.



You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

[Buzzer - Wikipedia](#)

### Example

- *Lesson 4 Doorbell* (R4 Minima Board Project)
- *Lesson 21 Simple Creation-Answer Machine* (R4 Minima Board Project)

### Driver

## 1.16 DC Motor



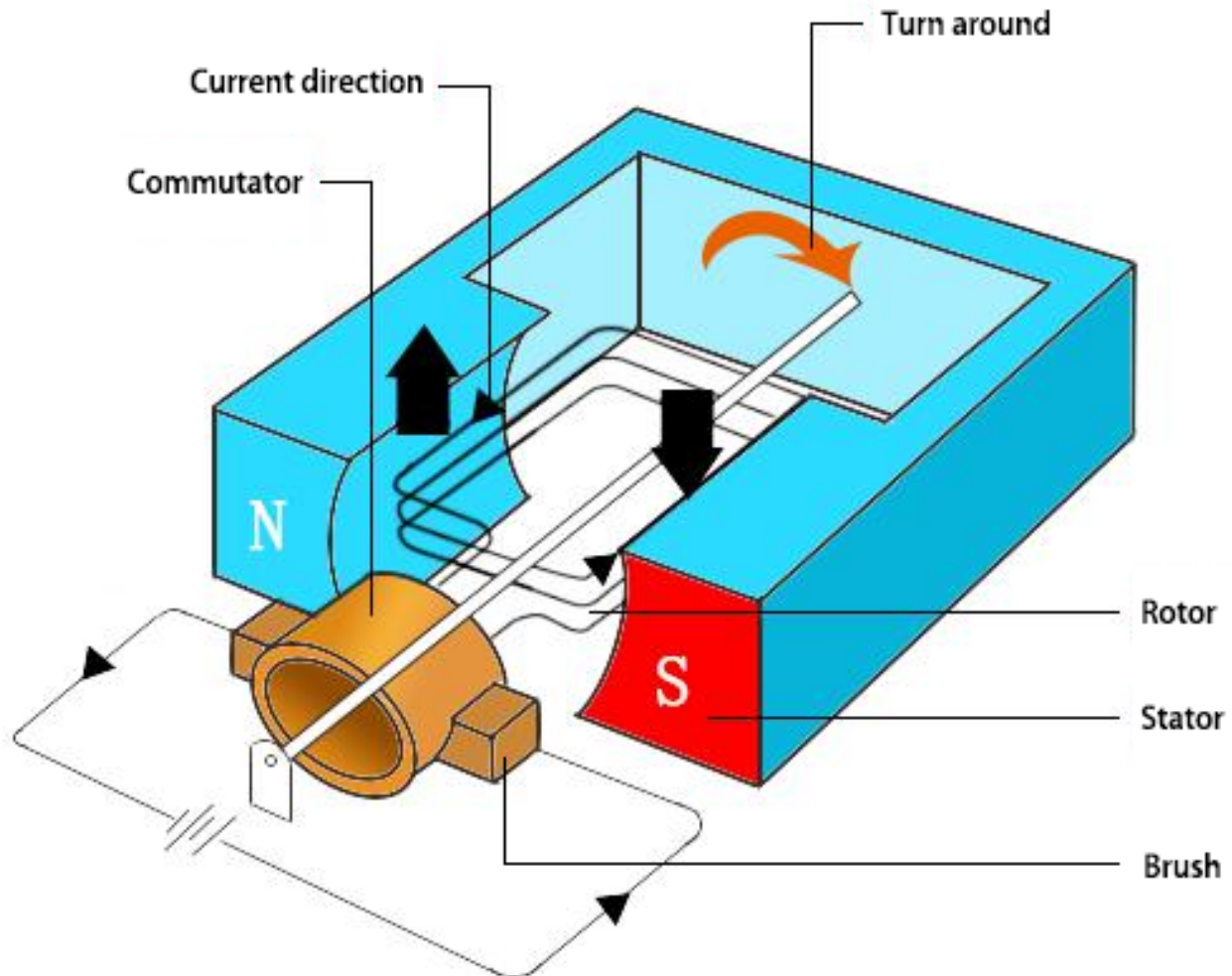
This is a 3V DC motor. When you give a high level and a low level to each of the 2 terminals, it will rotate.

- **Size:** 25\*20\*15MM
- **Operation Voltage:** 1-6V
- **Free-run Current** (3V): 70m
- **A Free-run Speed** (3V): 13000RPM

- **Stall Current** (3V): 800mA
- **Shaft Diameter:** 2mm

Direct current (DC) motor is a continuous actuator that converts electrical energy into mechanical energy. DC motors make rotary pumps, fans, compressors, impellers, and other devices work by producing continuous angular rotation.

A DC motor consists of two parts, the fixed part of the motor called the **stator** and the internal part of the motor called the **rotor** (or **armature** of a DC motor) that rotates to produce motion. The key to generating motion is to position the armature within the magnetic field of the permanent magnet (whose field extends from the north pole to the south pole). The interaction of the magnetic field and the moving charged particles (the current-carrying wire generates the magnetic field) produces the torque that rotates the armature.



Current flows from the positive terminal of the battery through the circuit, through the copper brushes to the commutator, and then to the armature. But because of the two gaps in the commutator, this flow reverses halfway through each complete rotation. This continuous reversal essentially converts the DC power from the battery to AC, allowing the armature to experience torque in the right direction at the right time to maintain rotation.

- [DC Motor - MagLab](#)

#### Example

- [Lesson 22 Simple Creation-Small Fan](#) (R4 Minima Board Project)

## 1.17 Stepper Motor

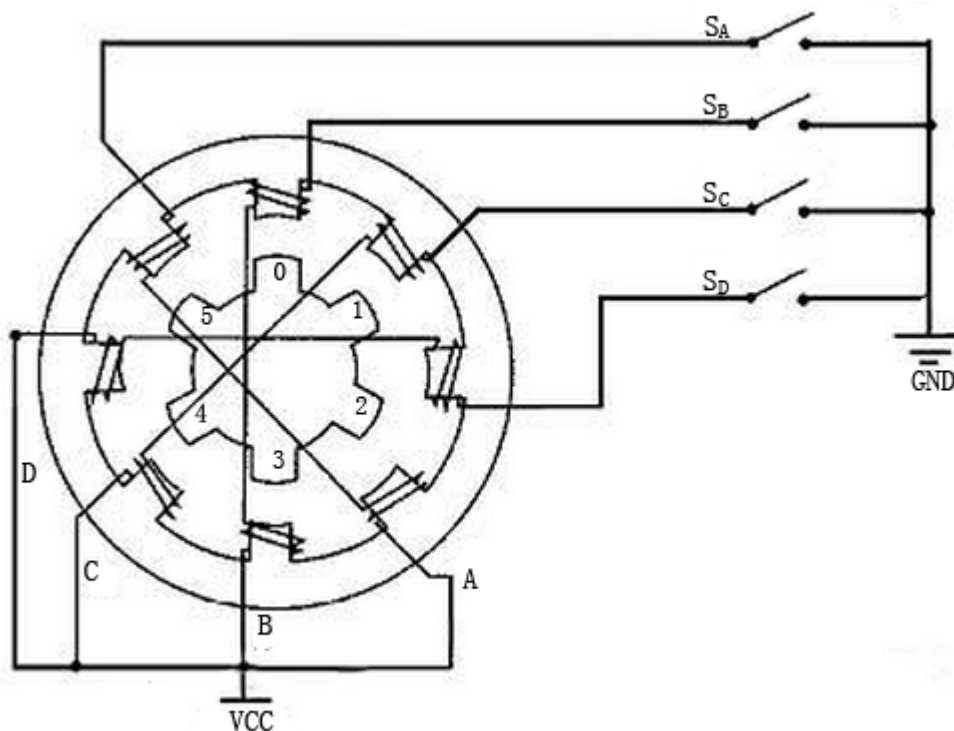


Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely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

### Principle

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

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



In the figure, in the middle of the motor is a rotor – a gear-shaped permanent magnet. Around the rotor, 0 to 5 are teeth. Then more outside, there are 8 magnetic poles, with each two opposite ones connected by coil winding. So they form

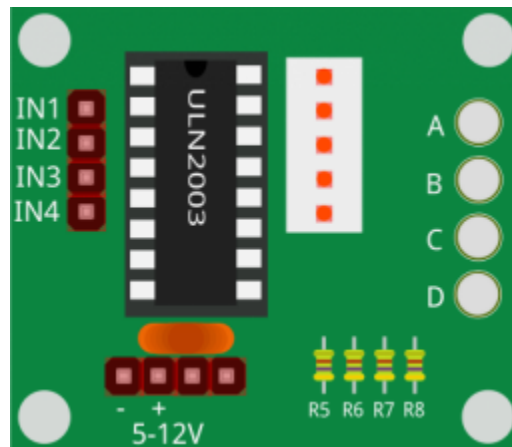
four pairs from A to D, which is called a phase. It has four lead wires to be connected with switches SA, SB, SC, and SD. Therefore, the four phases are in parallel in the circuit, and the two magnetic poles in one phase are in series.

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

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

The four-phase stepper motor has three operating modes: single four-step, double four-step, and eight-step. The step angle for the single four-step and double four-step are the same, but the driving torque for the single four-step is smaller. The step angle of the eight-step is half that of the single four-step and double four-step. Thus, the eight-step operating mode can keep high driving torque and improve control accuracy. In this experiment, we let the stepper motor work in the eight-step mode.

### ULN2003 Module



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

### Example

- [Lesson 19 Stepper Motor](#) (R4 Minima Board Project)

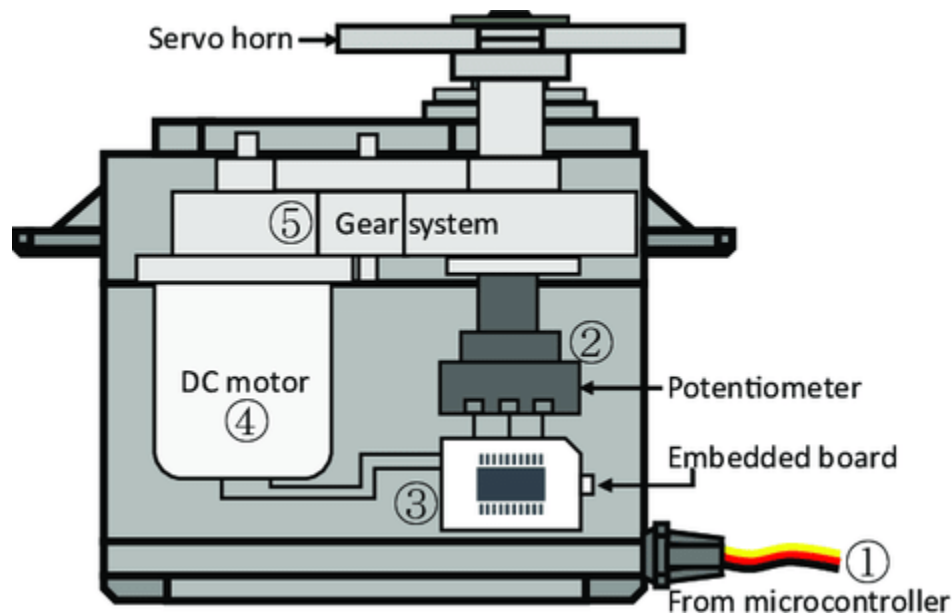


## 1.18 Servo



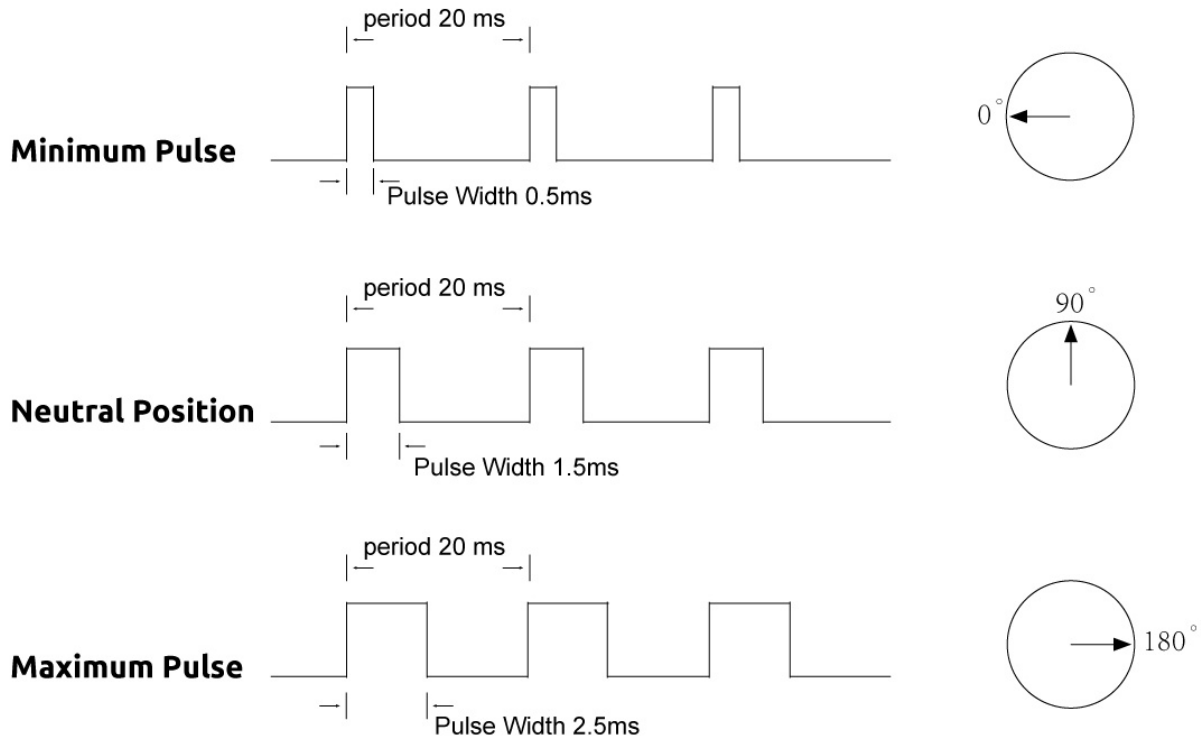
A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board 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 system and then motivates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded 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 angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.



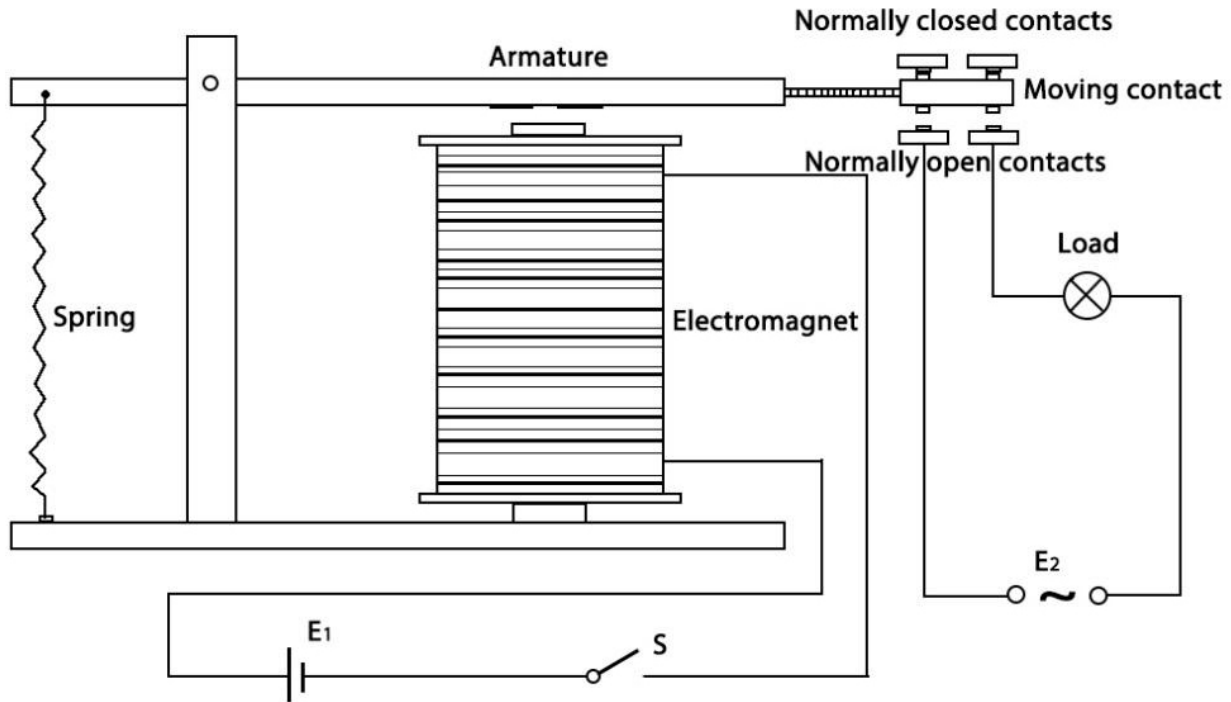
**Example**

- *Lesson 10 Servo* (R4 Minima Board Project)

**1.19 Relay**

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.

There are 5 parts in every relay:



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

**Armature** - The movable magnetic strip is known as armature. When current flows through them, the coil is 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).

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

**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.

**Molded frame** - Relays are covered with plastic for protection.

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.

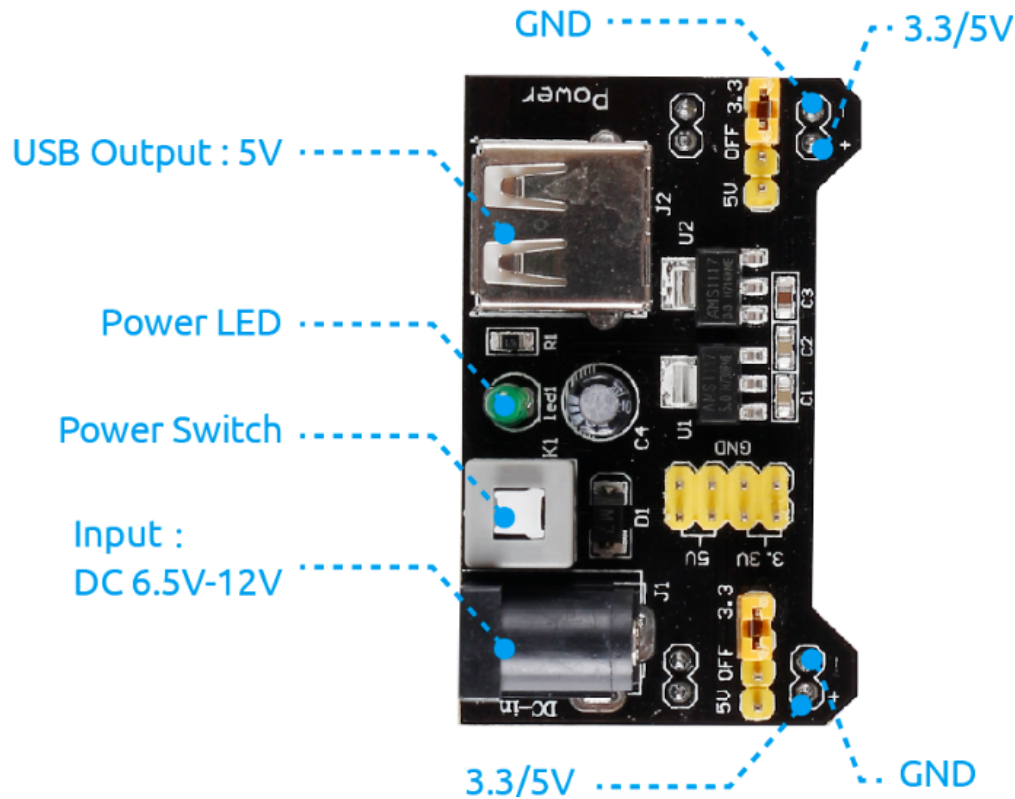
### Example

- [Lesson 6 Relay](#) (R4 Minima Board Project)

## 1.20 Power Supply Module

When we need a large current to drive a component, which will severely interfere with the normal work of main board. Therefore, we separately supply power for the component by this module to make it run safely and steadily.

You can just plug it in the breadboard to supply power. It provides a voltage of 3.3V and 5V, and you can connect either via a jumper cap included.

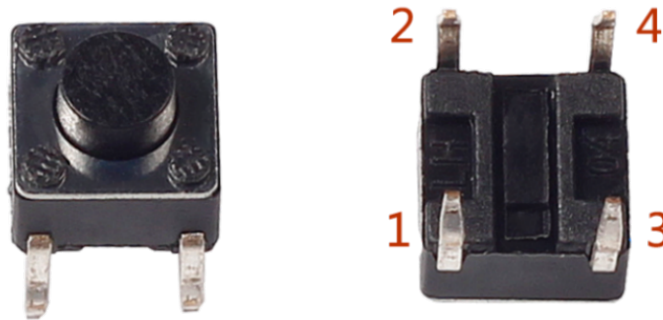


### Features and specifications

- Input voltage: 6.5 - 12V
- Two Independent Channel
- Output voltage: 5V, 3.3V (adjustable via jumpers. 0V, 3.3V, and 5V configuration)
- Output current: Maximum output current 700mA
- Onboard berg male header for GND, 5V, 3.3V output
- ON-OFF Switch available.
- USB (Type-A) input available.
- DC Barrel Jack input available.
- Onboard power LED
- Dimension: 53mm x 33mm (L x W)

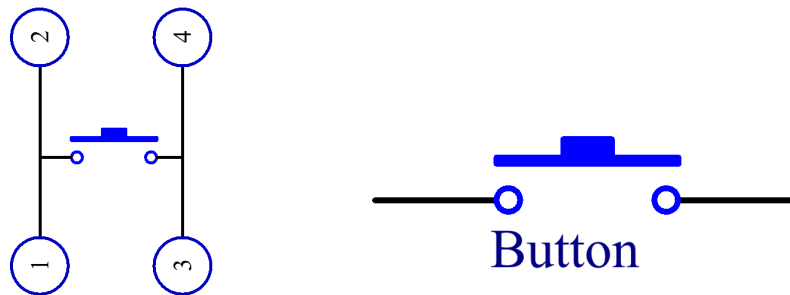
### Controller

## 1.21 Button

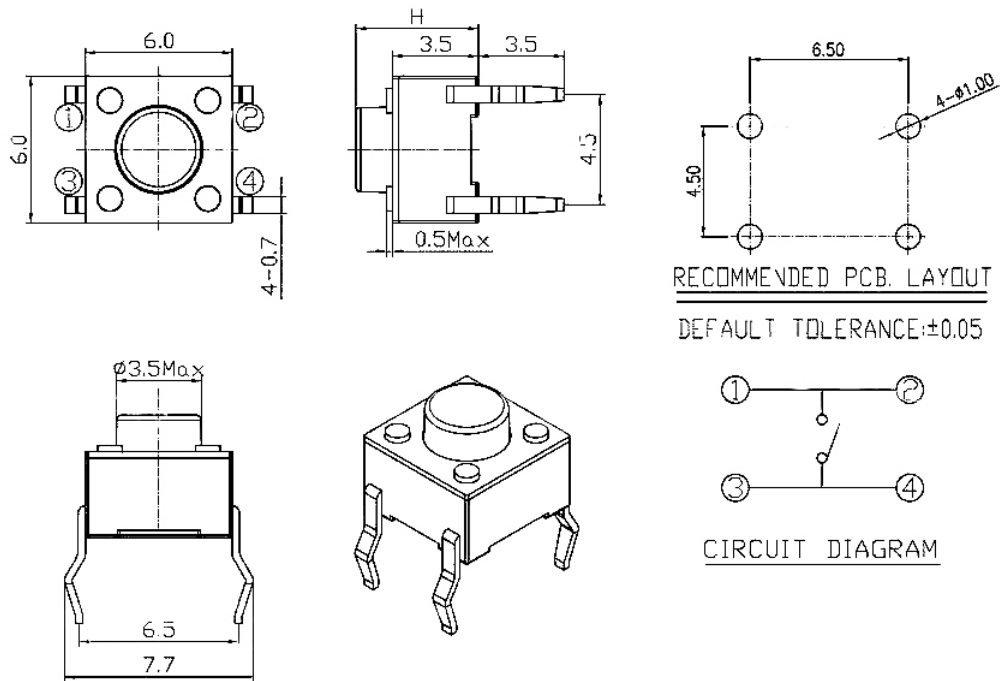


Buttons are a common component used to control electronic devices, they are usually used as switches to connect or break circuits.

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



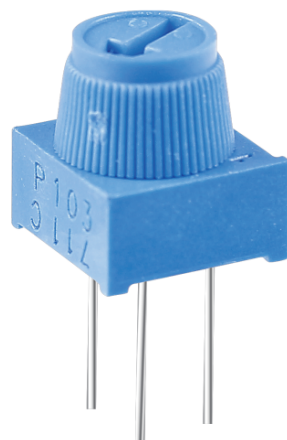
Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.



### Example

- *Lesson 3 Controlling LED by Button* (R4 Minima Board Project)
- *Lesson 21 Simple Creation-Answer Machine* (R4 Minima Board Project)

## 1.22 Potentiometer

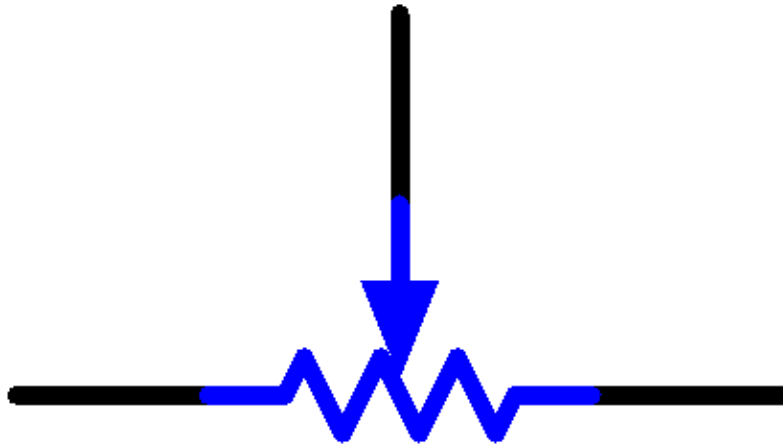


Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

Here is the circuit symbol of potentiometer.



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

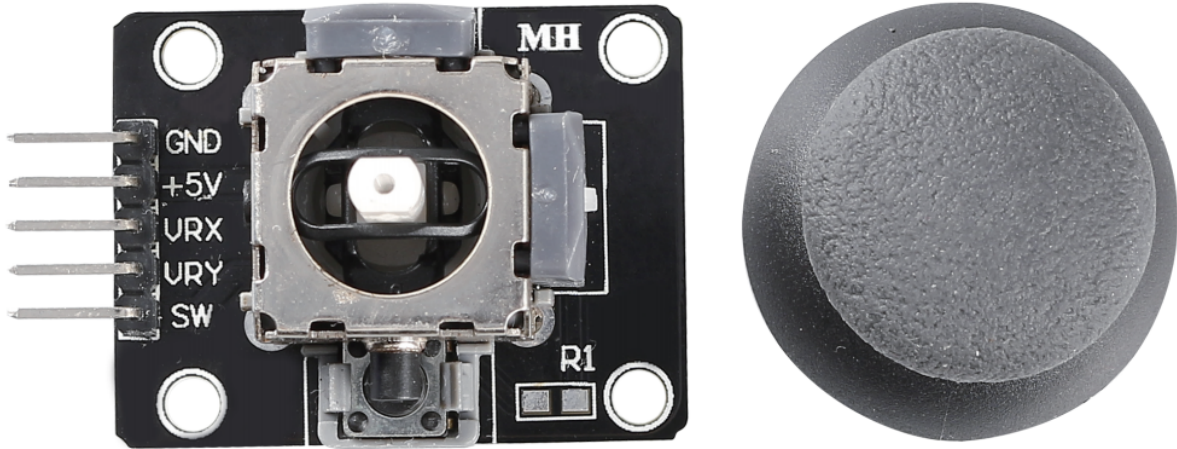
When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

If you want to know more about potentiometer, refer to: [Potentiometer - Wikipedia](#)

### Example

- *Lesson 8 Controlling an LED by Potentiometer* (R4 Minima Board Project)

## 1.23 Joystick Module

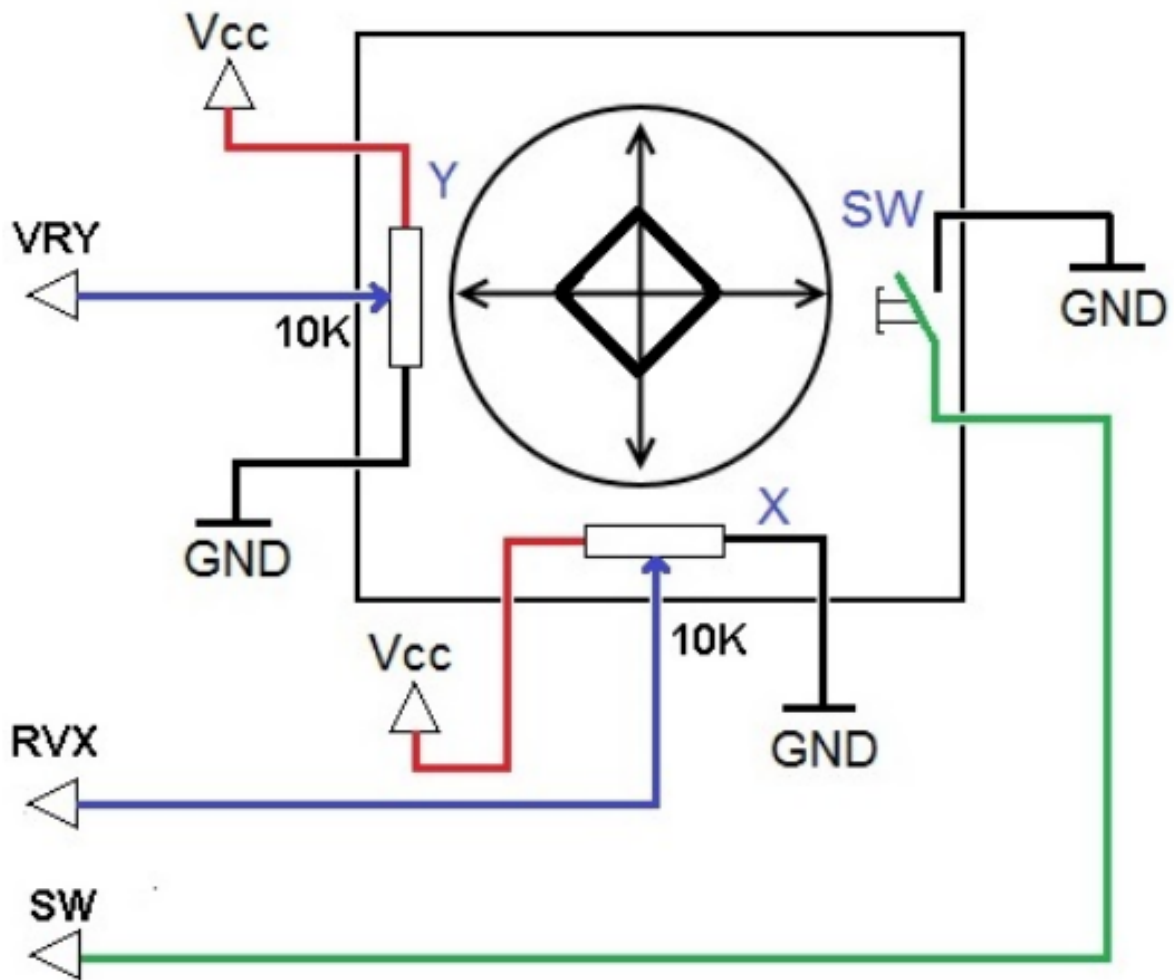


The basic idea of a joystick is to translate the movement of a stick into electronic information that a computer can process.

In order to communicate a full range of motion to the computer, a joystick needs to measure the stick's position on two axes – the X-axis (left to right) and the Y-axis (up and down). Just as in basic geometry, the X-Y coordinates pinpoint the stick's position exactly.

To determine the location of the stick, the joystick control system simply monitors the position of each shaft. The conventional analog joystick design does this with two potentiometers, or variable resistors.

The joystick also has a digital input that is actuated when the joystick is pressed down.



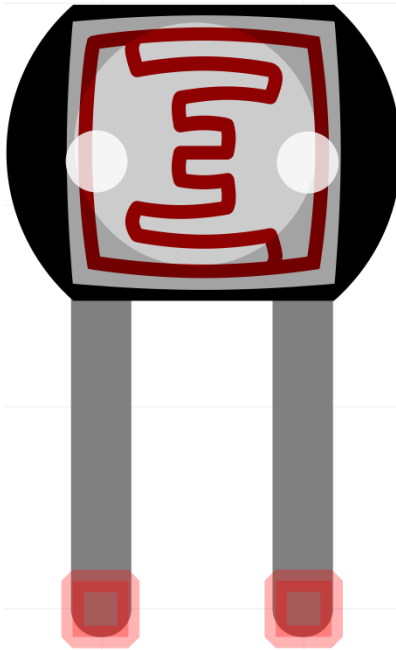
### Example

- *Lesson 16 Joystick PS2* (R4 Minima Board Project)

### Sensor



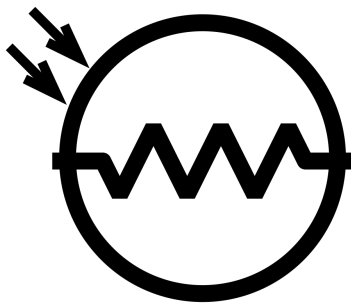
## 1.24 Photoresistor



A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.

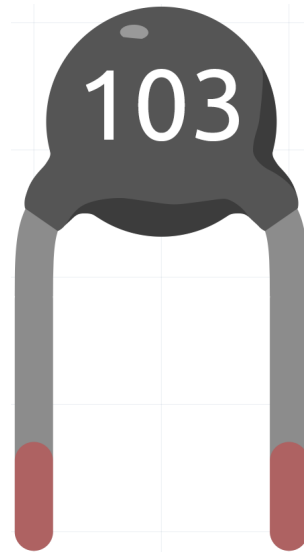


- [Photoresistor - Wikipedia](#)

### Example

- *Lesson 9 Photoresistor* (R4 Minima Board Project)

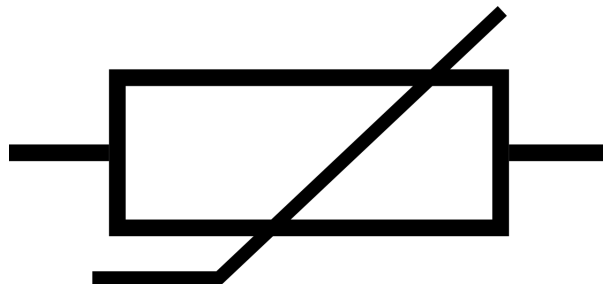
## 1.25 Thermistor



A thermistor is a type of resistor whose resistance is strongly dependent on temperature, more so than in standard resistors. The word is a combination of thermal and resistor. Thermistors are widely used as inrush current limiters, temperature sensors (negative temperature coefficient or NTC type typically), self-resetting overcurrent protectors, and self-regulating heating elements (positive temperature coefficient or PTC type typically).

- [Thermistor - Wikipedia](#)

Here is the electronic symbol of thermistor.



Thermistors are of two opposite fundamental types:

- With NTC thermistors, resistance decreases as temperature rises usually due to an increase in conduction electrons bumped up by thermal agitation from valency band. An NTC is commonly used as a temperature sensor, or in series with a circuit as an inrush current limiter.
- With PTC thermistors, resistance increases as temperature rises usually due to increased thermal lattice agitations particularly those of impurities and imperfections. PTC thermistors are commonly installed in series with a circuit, and used to protect against overcurrent conditions, as resettable fuses.

In this kit we use an NTC one. 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:

$$R_T = R_N * \exp(B(1/T_K - 1/T_N))$$

- **R<sub>T</sub>** is the resistance of the NTC thermistor when the temperature is T<sub>K</sub>.

- **RN** is the resistance of the NTC thermistor under the rated temperature  $T_N$ . Here, the numerical value of RN is 10k.
- **TK** is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- **TN** is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And **B(beta)**, the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- **exp** is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

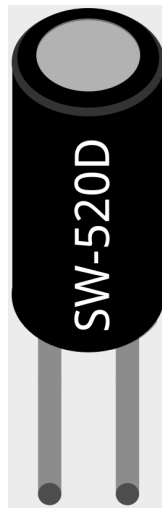
Convert this formula  $TK = 1 / (\ln(RT/RN) / B + 1 / TN)$  to get Kelvin temperature that minus 273.15 equals degree Celsius.

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

### Example

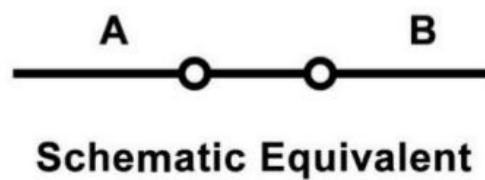
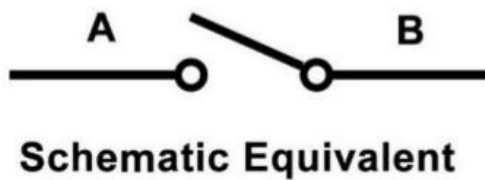
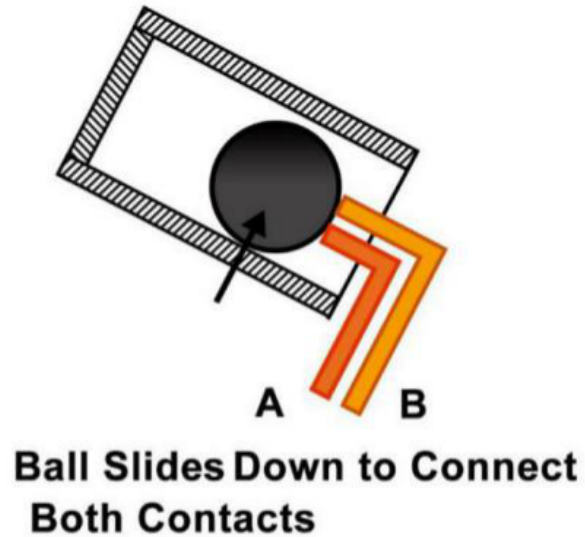
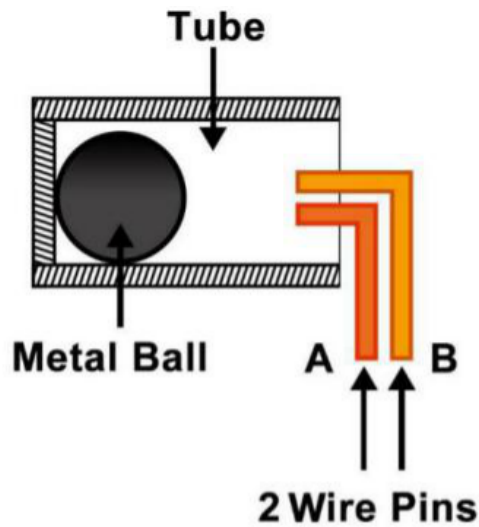
- *Lesson 12 Thermistor* (R4 Minima Board Project)

## 1.26 Tilt Switch



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

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.



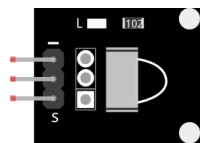
- [SW520D Tilt Switch Datasheet](#)

#### Example

- [Lesson 5 Tilt Switch](#) (R4 Minima Board Project)

## 1.27 IR Receiver Module

### IR Receiver Module



- S: Signal output
- +VCC
- -: GND

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

Infrared, or IR, communication is a popular, low-cost, easy-to-use wireless communication technology. Infrared light has a slightly longer wavelength than visible light, so it is imperceptible to the human eye - ideal for wireless communication. A common modulation scheme for infrared communication is 38KHz modulation.

- Adopted HX1838 IR Receiver Sensor, high sensitivity
- Can be used for remote control
- Power Supply: 5V
- Interface: Digital
- Modulate Frequency: 38Khz
- Pin Definitions: (1) Output (2) Vcc (3) GND
- Size: 23.5mm x 21.5mm

### Remote Control



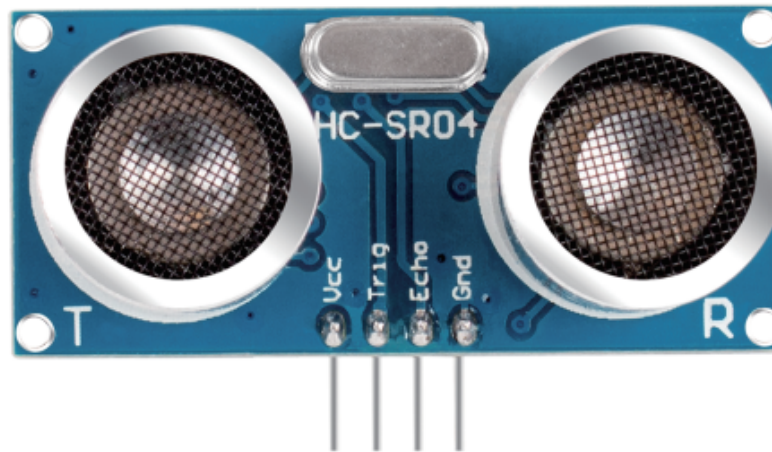
This is a Mini thin infrared wireless remote control with 21 function buttons and a transmitting distance of up to 8 meters, which is suitable for operating a wide range of devices in a kid's room.

- Size: 85x39x6mm
- Remote control range: 8-10m
- Battery: 3V button type lithium manganese battery
- Infrared carrier frequency: 38KHz
- Surface paste material: 0.125mm PET
- Effective life: more than 20,000 times

### Example

- [\*Lesson 14 Infrared-Receiver\*](#) (R4 Minima Board Project)

## 1.28 Ultrasonic Module



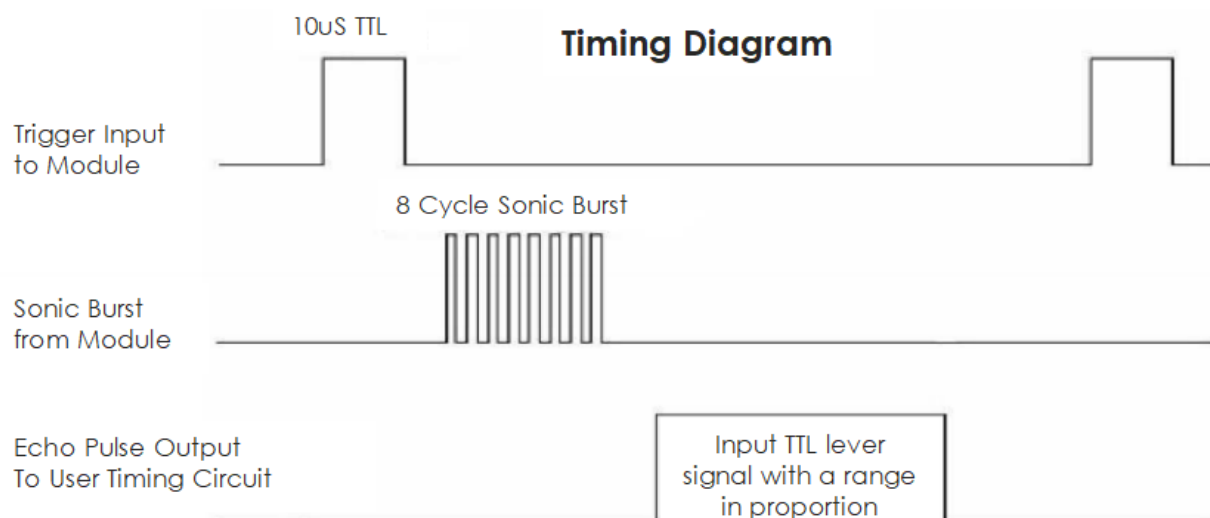
An ultrasonic sensor module is an instrument that measures the distance to an object using ultrasonic sound waves. It has two probes. One is to send ultrasonic waves and the other is to receive the waves and transform the time of sending and receiving into a distance, thus detecting the distance between the device and an obstacle. In practice it is really convenient and functional.

It provides 2cm - 400cm non-contact measurement function, and the ranging accuracy can reach to 3mm. It can ensure that the signal is stable within 5m, and the signal is gradually weakened after 5m, till the 7m position disappears.

The module includes ultrasonic transmitters, receiver and control circuit. The basic principles are as follows:

1. Use an IO flip-flop to process a high level signal of at least 10 $\mu$ s.
2. The module automatically sends eight 40khz and detects if there is a pulse signal return.
3. If the signal returns, passing the high level, the high output IO duration is the time from the transmission of the ultrasonic wave to the return of it. Here, test distance = (high time x sound speed (340 m / s) / 2.

The timing diagram is shown below.



You only need to supply a short 10 $\mu$ s pulse for the trigger input to start the ranging, and then the module will send

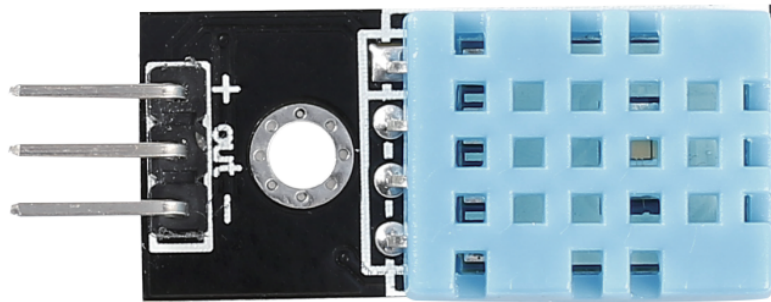
out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula:  $us / 58 = \text{centimeters}$  or  $us / 148 = \text{inch}$ ; or: the range = high level time \* velocity (340M/S) / 2; you are suggested to use measurement cycle over 60ms in order to prevent signal collisions of trigger signal and the echo signal.

### Example

- *Lesson 13 Ultrasonic* (R4 Minima Board Project)

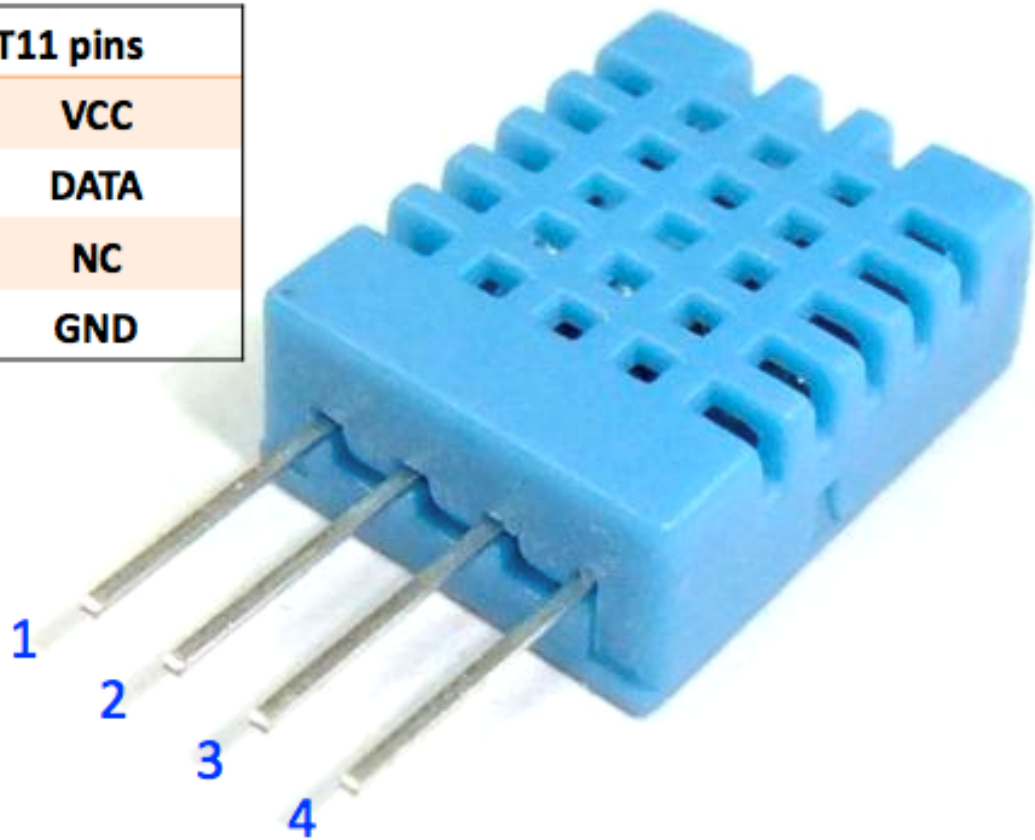
## 1.29 Humiture Sensor Module



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

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

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



- [DHT11 Datasheet](#)

#### Example

- *Lesson 15 Humiture Sensor* (R4 Minima Board Project)



## **GET STARTED WITH ARDUINO**

If you have no idea about Arduino. There are several words I would like to show you: electronics, design, programming, and even Maker. Some of you may think these words are quite far away from us, but in fact, they are not far at all. Because Arduino can take us into the world of programming and help us realize the dream of being a Maker. In this session we will learn:

### **2.1 What is Arduino?**

First of all, I will give you a brief introduction to Arduino.

Arduino is a convenient, flexible, and easy-to-use open-source electronic prototyping platform, including hardware Arduino boards of various models and software Arduino IDE. It is not only suitable for engineers for rapid prototyping, but also artists, designers, hobbyists, while it is almost a must-have tool for modern Makers.

Arduino is quite a large system. It has software, hardware, and a very huge online community of people who have never met each other but are able to work together because of a common hobby. Everyone in the Arduino family is using their wisdom, making with their hands, and sharing one great invention after another. And you can also be a part of it.

### **2.2 What can Arduino do?**

Speaking of which, you may have doubts about what Arduino can actually do. Suffice it to say, Arduino will solve all your problems.

Technically speaking, Arduino is a programmable logic controller. It is a development board that can be used to create many exciting and creative electronic creations: such as remote-controlled cars, robotic arms, bionic robots, smart homes, etc.

Arduino boards are straightforward, simple, and powerful, suitable for students, makers and even professional programmers.

To this day, electronics enthusiasts worldwide continue to develop creative electronic creations based on Arduino development boards.

## 2.3 How to build an Arduino Project

Embarking on a journey into the world of Arduino? You're in the right place! This guide will walk you step-by-step, starting from the very basics. Dive into the installation process of the Arduino IDE, familiarize yourself with its interface, and take your first steps in creating, saving, and uploading your projects. But that's just the tip of the iceberg! Delve deeper into the fundamentals of coding structures, unravel the mysteries of syntax, master the art of variables, and finally, light up your knowledge by setting up basic circuits. So, gear up and let's embark on this electrifying adventure into Arduino!

### 2.3.1 Download and Install Arduino IDE 2.0

The Arduino IDE, known as Arduino Integrated Development Environment, provides all the software support needed to complete an Arduino project. It is a programming software specifically designed for Arduino, provided by the Arduino team, that allows us to write programs and upload them to the Arduino board.

The Arduino IDE 2.0 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

In this tutorial, we will show how to download and install the Arduino IDE 2.0 on your Windows, Mac, or Linux computer.

#### Requirements

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: "Mojave" or newer, 64 bits

#### Download the Arduino IDE 2.0

1. Visit .
2. Download the IDE for your OS version.



## Arduino IDE 2.0.0

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

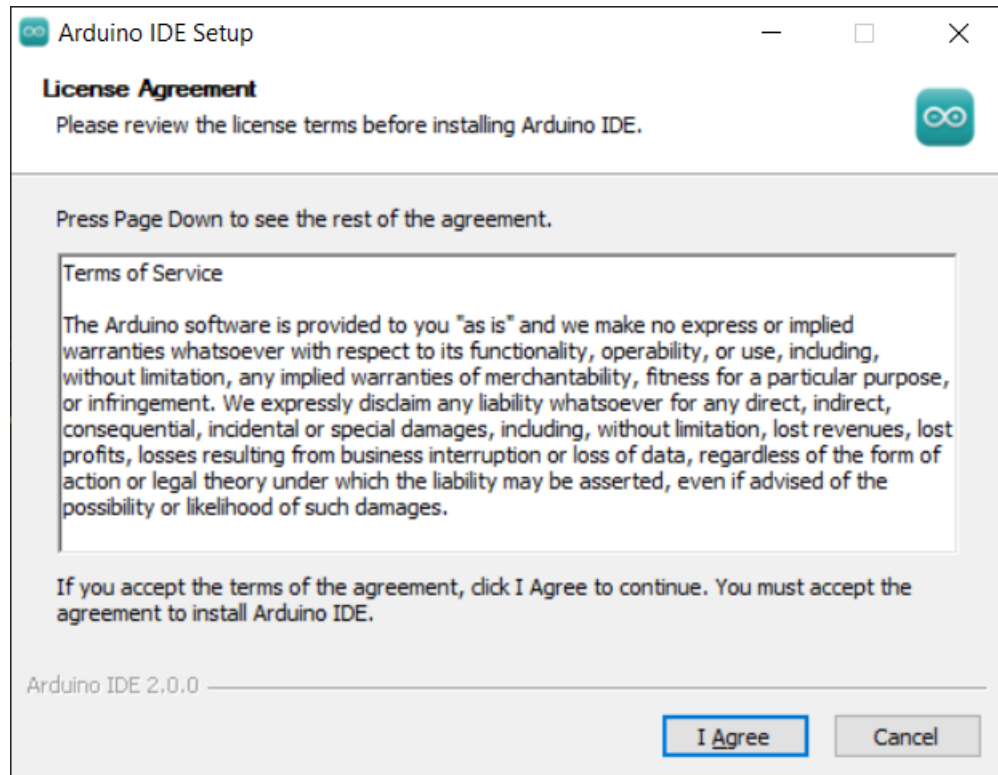
#### DOWNLOAD OPTIONS

<b>Windows</b>	Win 10 and newer, 64 bits
<b>Windows</b>	MSI installer
<b>Windows</b>	ZIP file
<b>Linux</b>	AppImage 64 bits (X86-64)
<b>Linux</b>	ZIP file 64 bits (X86-64)
<b>macOS</b>	10.14: "Mojave" or newer, 64 bits

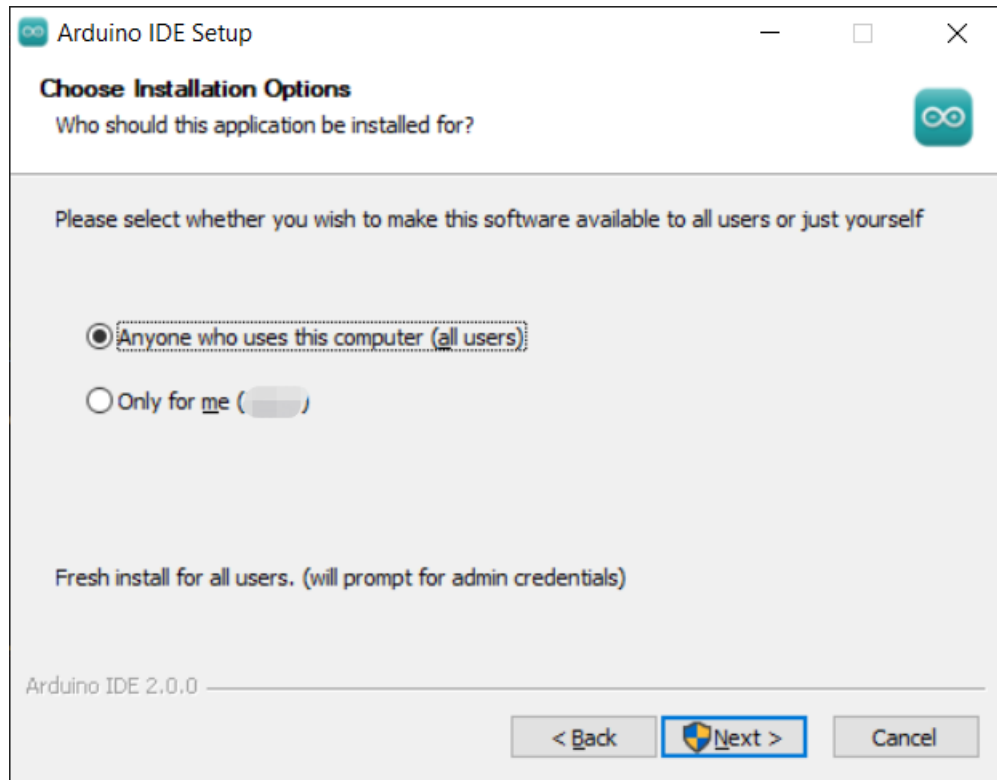
## Installation

### Windows

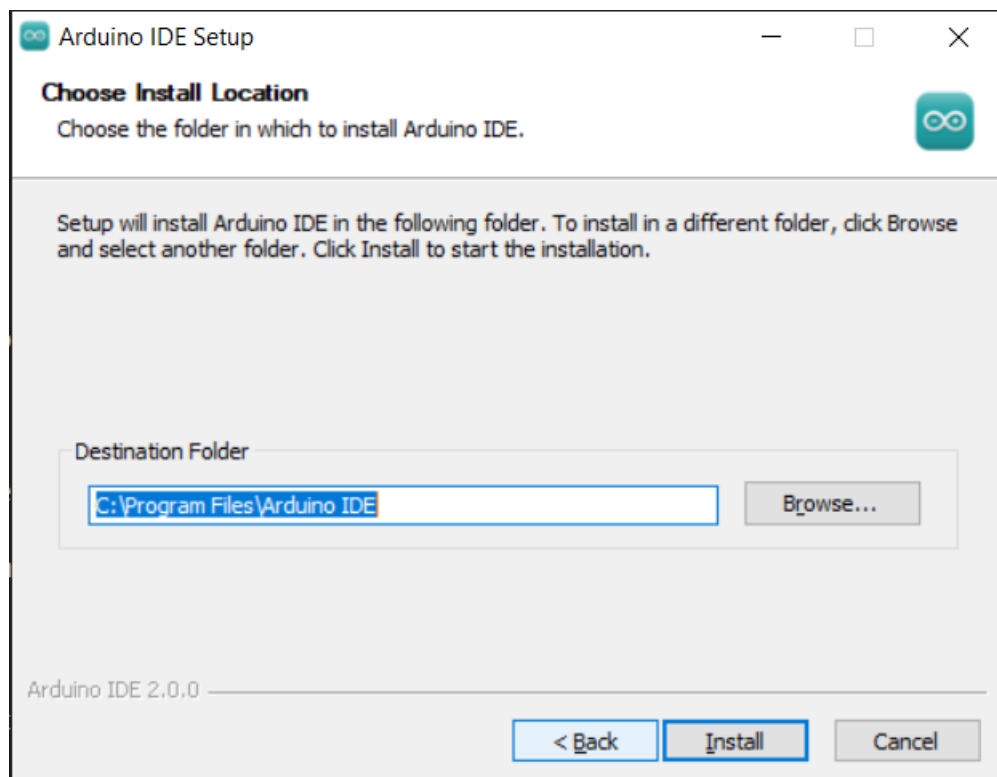
1. Double click the `arduino-ide_xxxx.exe` file to run the downloaded file.
2. Read the License Agreement and agree it.



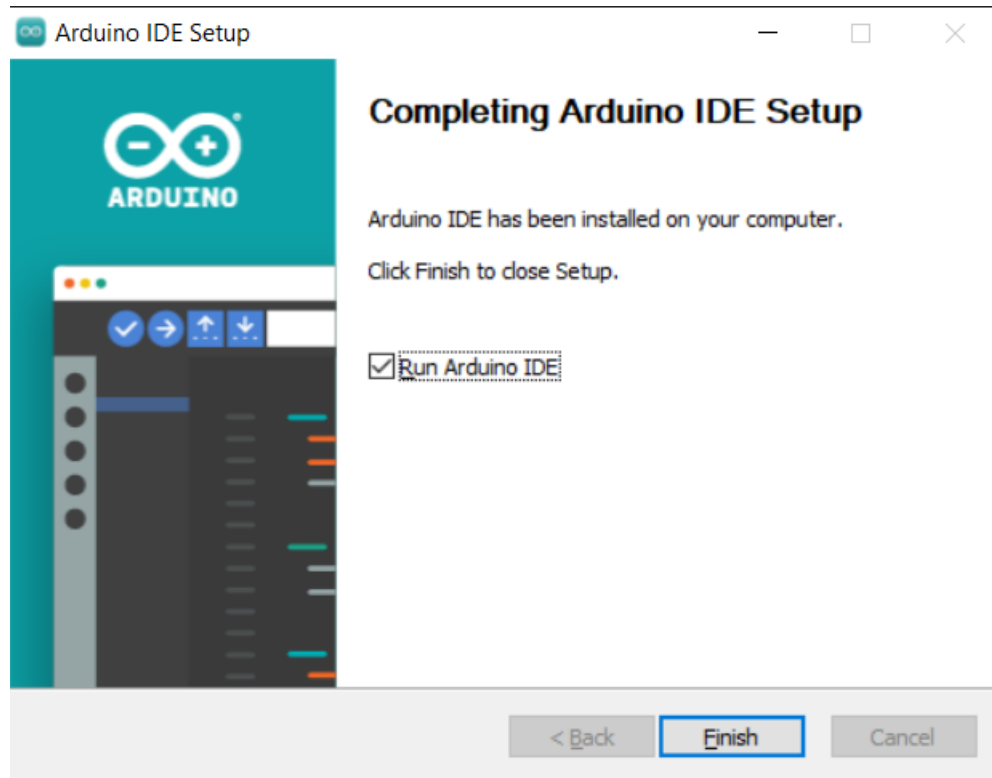
3. Choose installation options.



4. Choose install location. It is recommended that the software be installed on a drive other than the system drive.

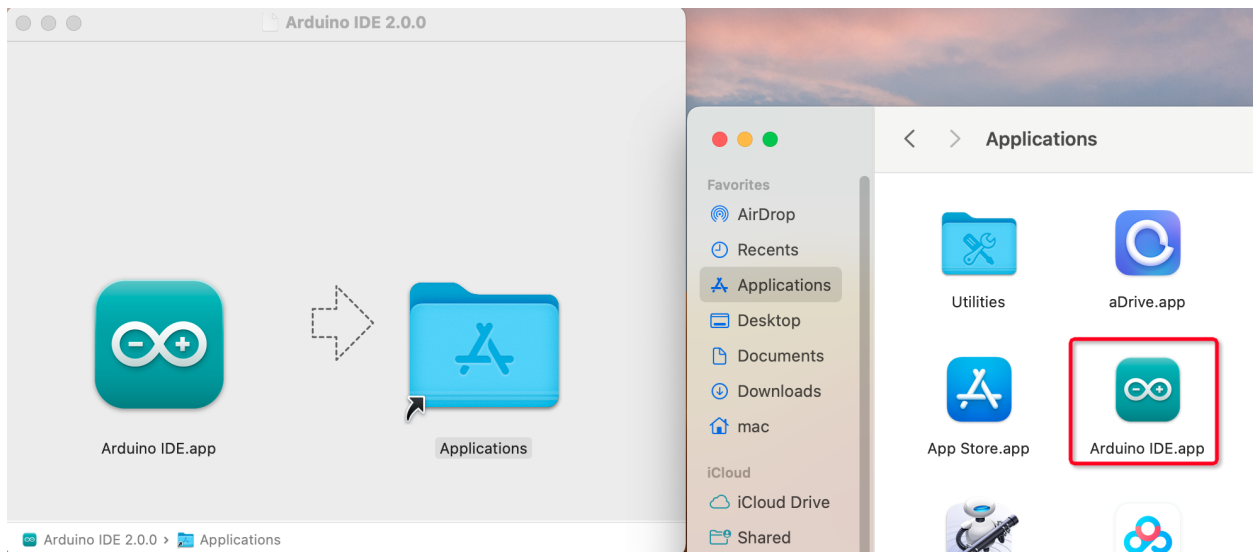


5. Then Finish.



## macOS

Double click on the downloaded `arduino_ide_xxxx.dmg` file and follow the instructions to copy the **Arduino IDE.app** to the **Applications** folder, you will see the Arduino IDE installed successfully after a few seconds.

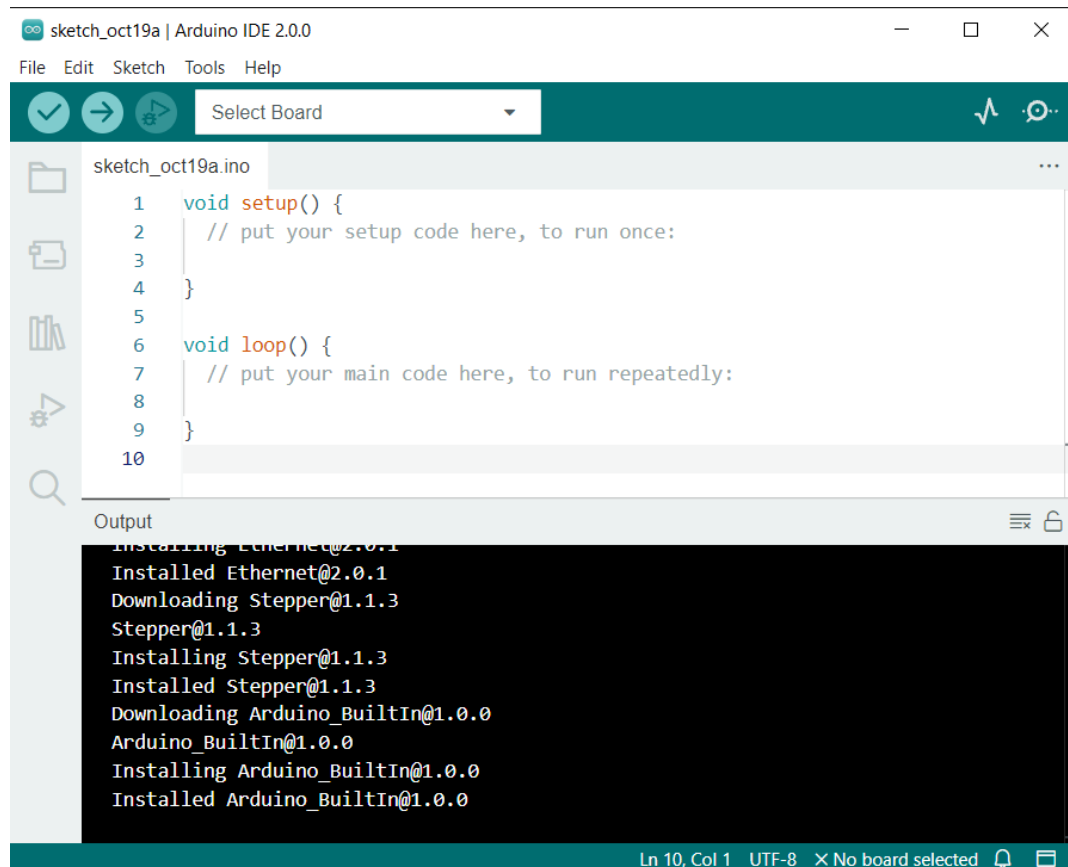


### Linux

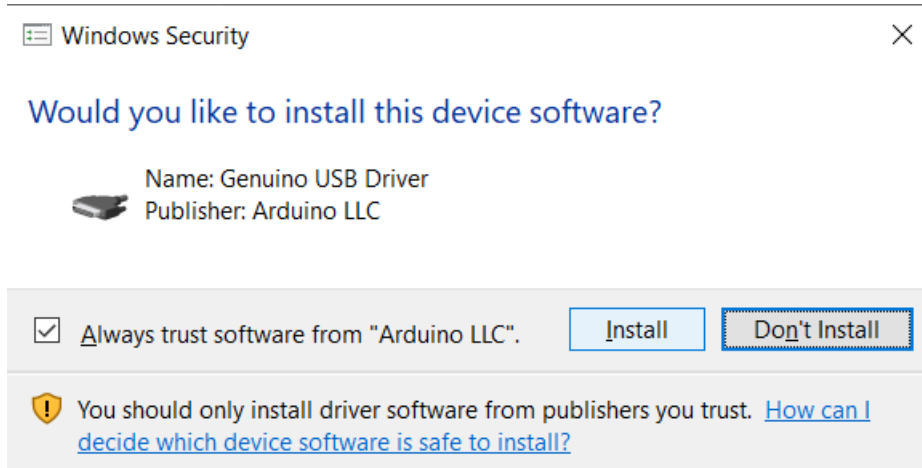
For the tutorial on installing the Arduino IDE 2.0 on a Linux system, please refer to: <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing#linux>

### Open the IDE

1. When you first open Arduino IDE 2.0, it automatically installs the Arduino AVR Boards, built-in libraries, and other required files.



2. In addition, your firewall or security center may pop up a few times asking you if you want to install some device driver. Please install all of them.



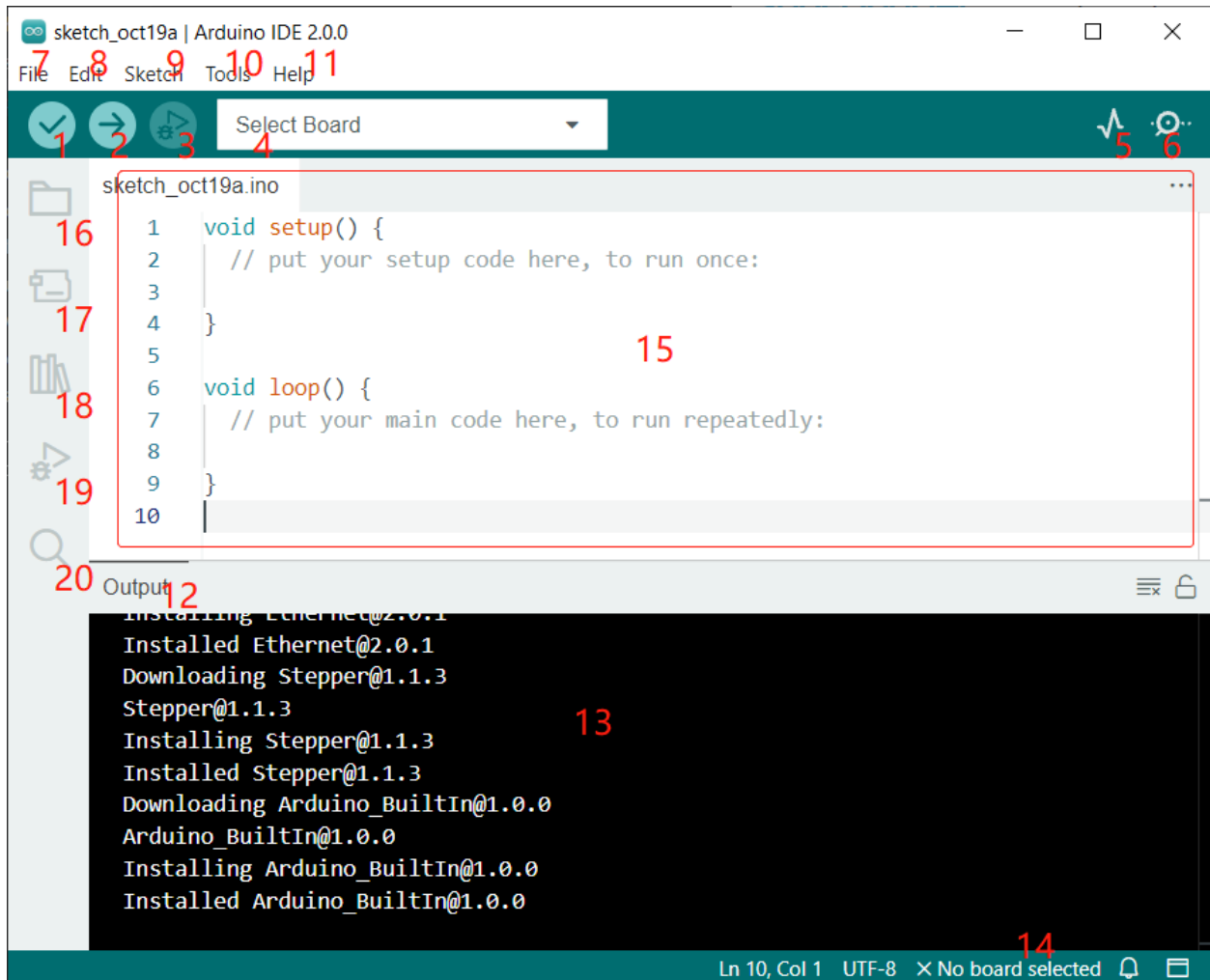
3. Now your Arduino IDE is ready!

---

**Note:** In the event that some installations didn't work due to network issues or other reasons, you can reopen the Arduino IDE and it will finish the rest of the installation. The Output window will not automatically open after all installations are complete unless you click Verify or Upload.

---

## 2.3.2 Introduce of Arduino IDE



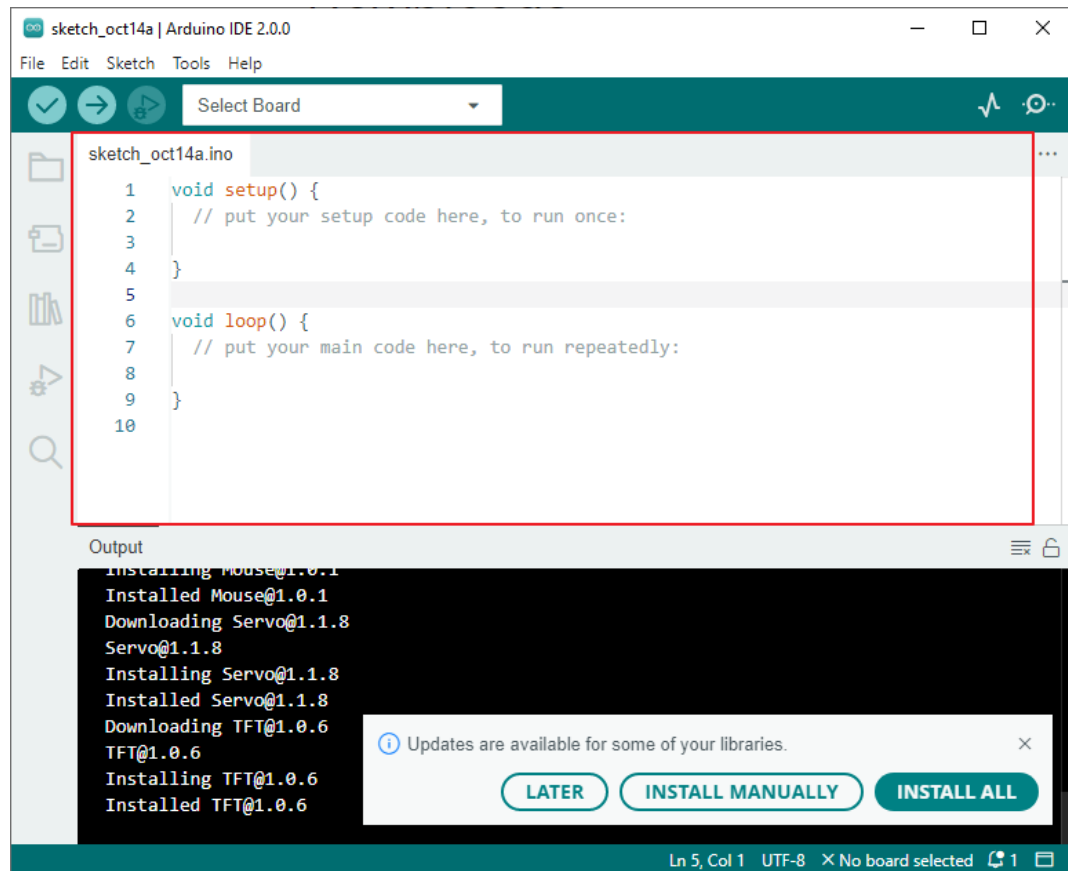
1. **Verify:** Compile your code. Any syntax problem will be prompted with errors.
2. **Upload:** Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **Debug:** For line-by-line error checking.
4. **Select Board:** Quick setup board and port.
5. **Serial Plotter:** Check the change of reading value.
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. **Output Bar:** Switch the output tab here.
13. **Output Window:** Print information.
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.
16. **Sketchbook:** For managing sketch files.
17. **Board Manager:** For managing board driver.
18. **Library Manager:** For managing your library files.
19. **Debug:** Help debugging code.
20. **Search:** Search the codes from your sketches.

### 2.3.3 How to create, open or Save the Sketch?

1. When you open the Arduino IDE for the first time or create a new sketch, you will see a page like this, where the Arduino IDE creates a new file for you, which is called a “sketch”.



These sketch files have a regular temporary name, from which you can tell the date the file was created. sketch\_oct14a.ino means October 14th first sketch, .ino is the file format of this sketch.

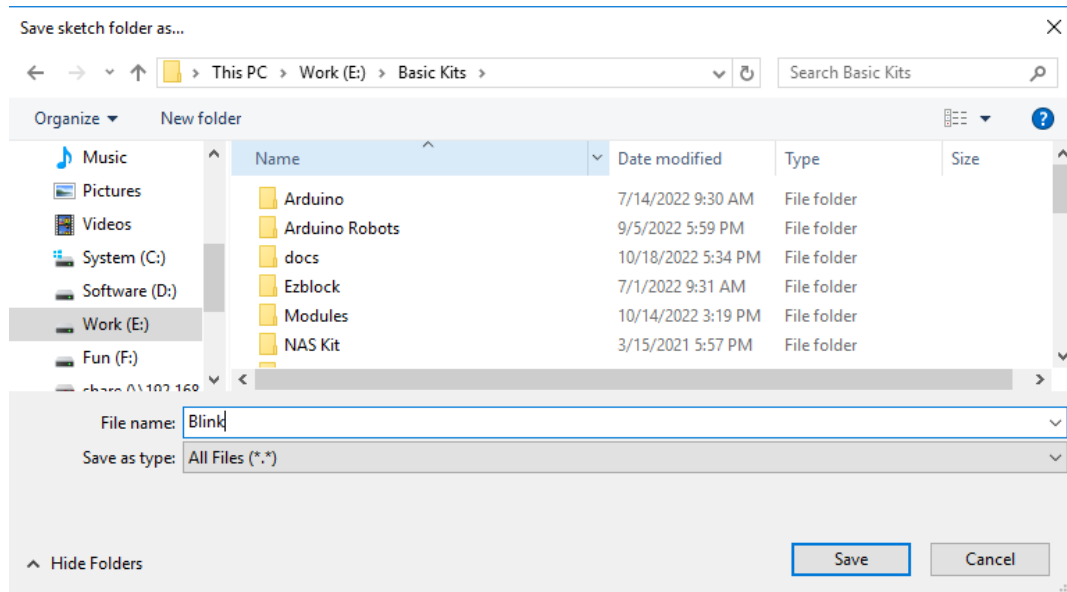
2. Now let's try to create a new sketch. Copy the following code into the Arduino IDE to replace the original code.



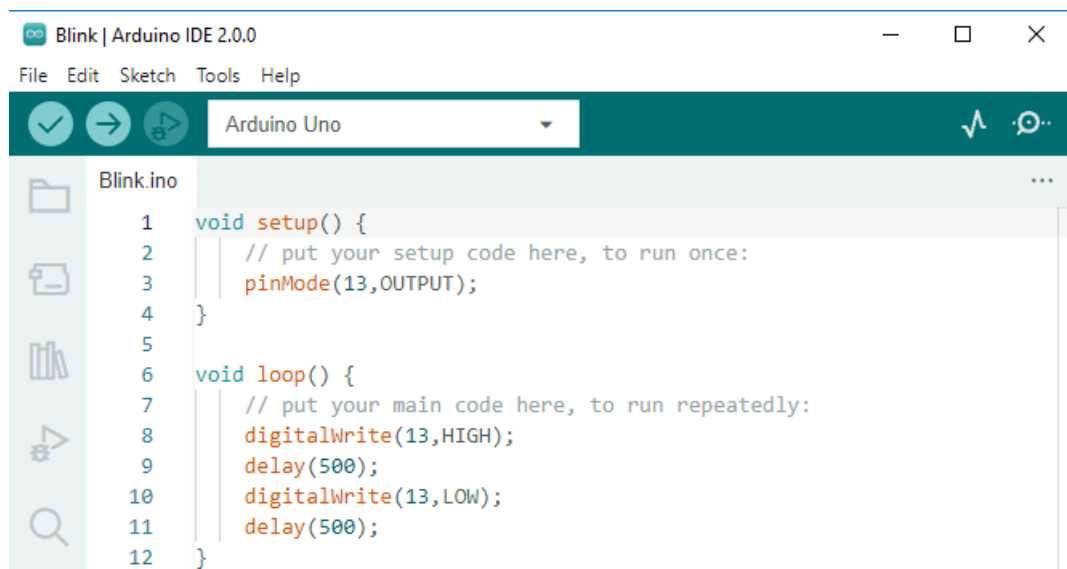
```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
}
```

3. Press Ctrl+S or click **File** -> **Save**. The Sketch is saved in: C:\Users\{your\_user}\Documents\Arduino by default, you can rename it or find a new path to save it.



4. After successful saving, you will see that the name in the Arduino IDE has been updated.



Please continue with the next section to learn how to upload this created sketch to your Arduino board.

## 2.3.4 How to upload Sketch to the Board?

In this section, you will learn how to upload the sketch created previously to the Arduino board, as well as learn about some considerations.

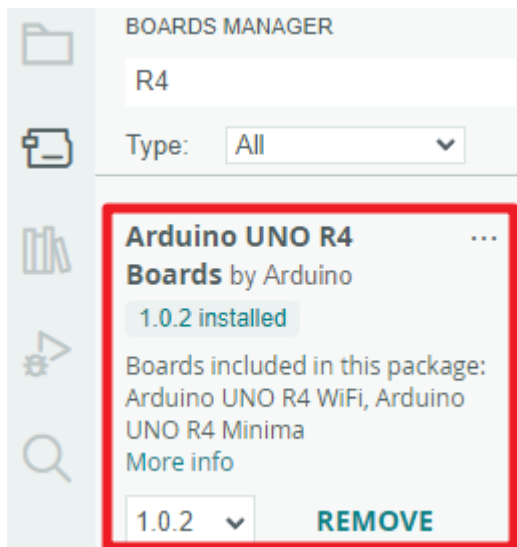
### 1. Choose Board and port

Arduino development boards usually come with a USB cable. You can use it to connect the board to your computer.

Select the correct **Board** and **Port** in the Arduino IDE. Normally, Arduino boards are recognized automatically by the computer and assigned a port, so you can select it here.



If your board is already plugged in, but not recognized, check if the **INSTALLED** logo appears in the **Arduino UNO R4 Boards** section of the **Boards Manager**, if not, please scroll down a bit and click on **INSTALL**.



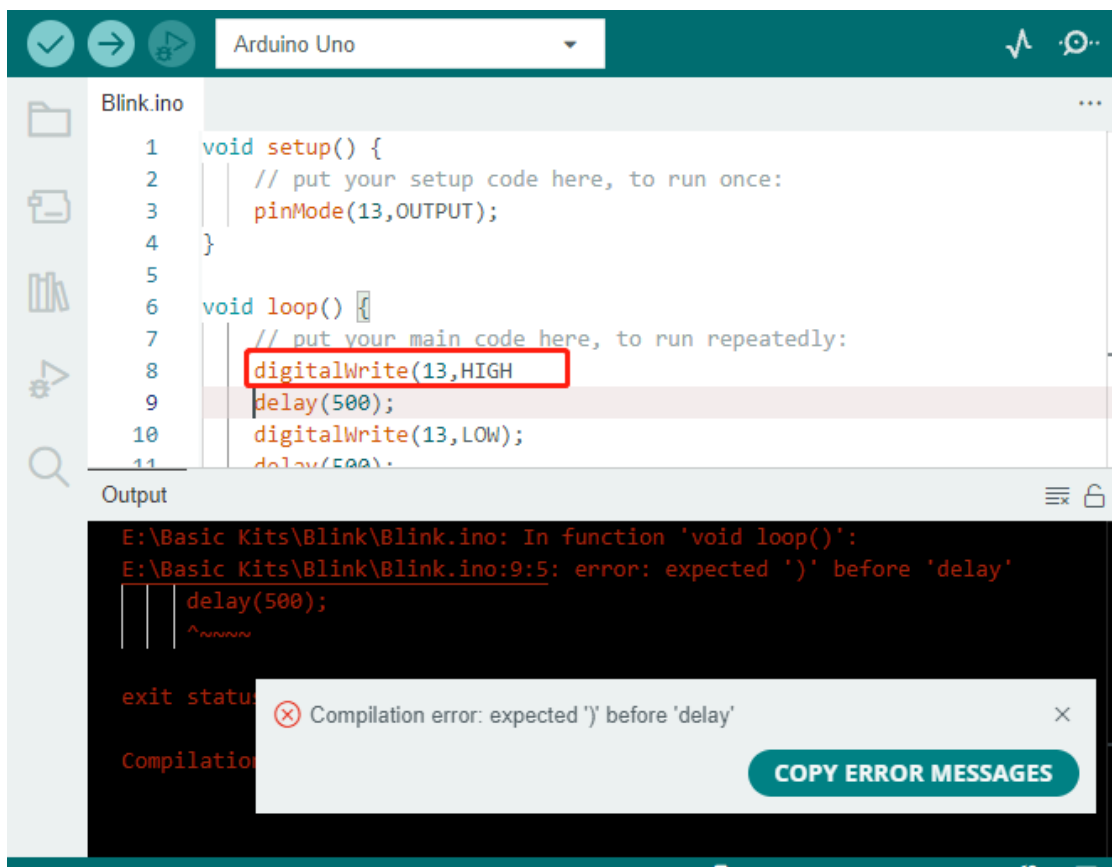
Reopening the Arduino IDE and re-plugging the Arduino board will fix most of the problems. You can also click **Tools** -> **Board** or **Port** to select them.

## 2. Verify the Sketch

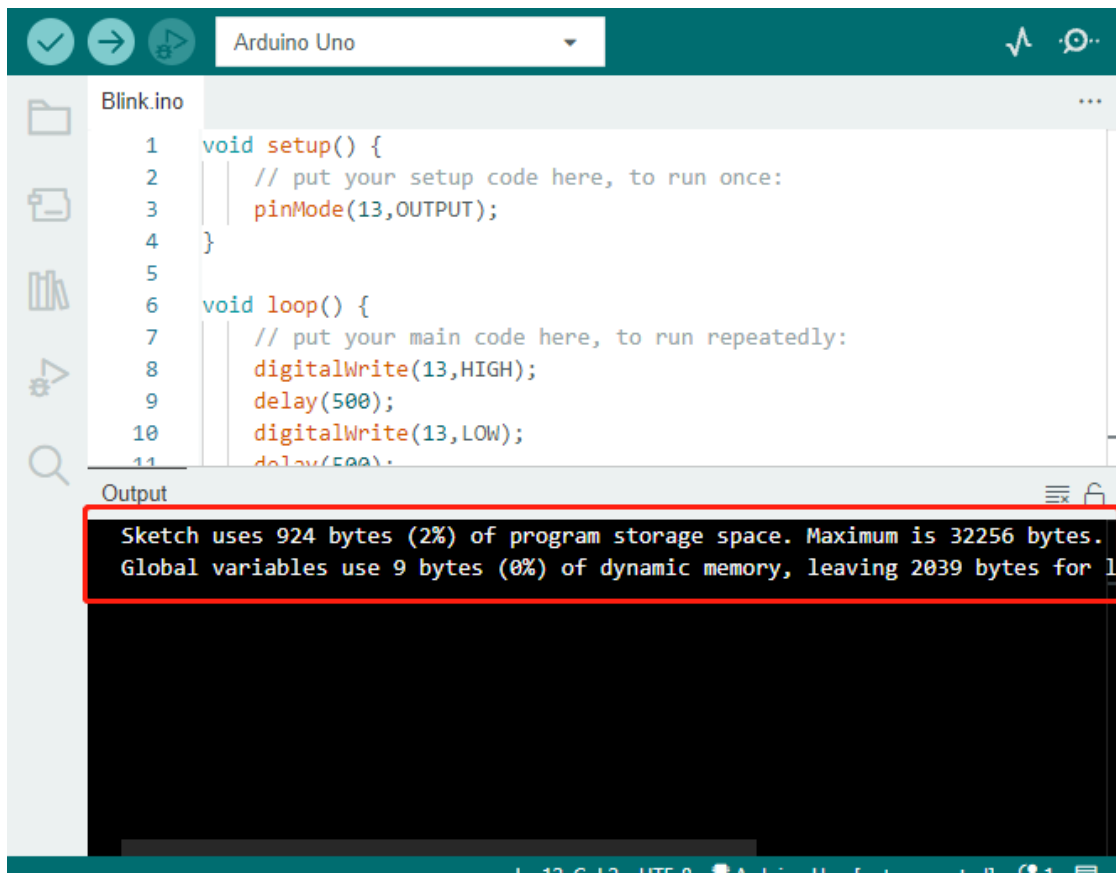
After clicking the Verify button, the sketch will be compiled to see if there are any errors.



You can use it to find mistakes if you delete some characters or type a few letters by mistake. From the message bar, you can see where and what type of errors occurred.

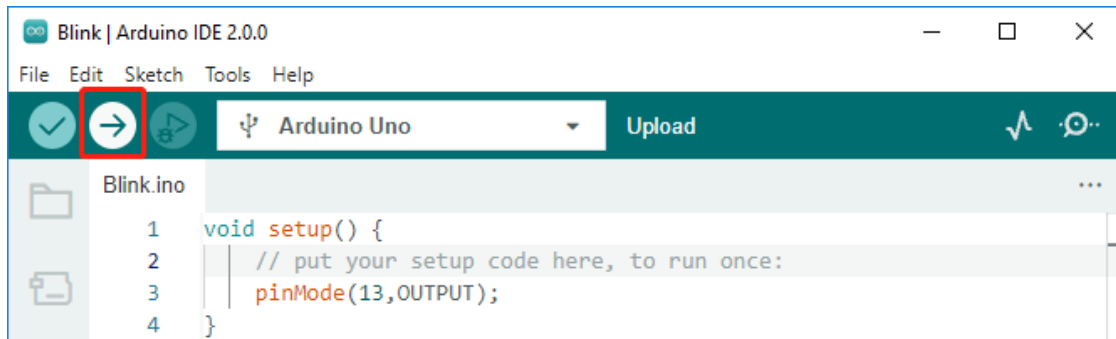


If there are no errors, you will see a message like the one below.

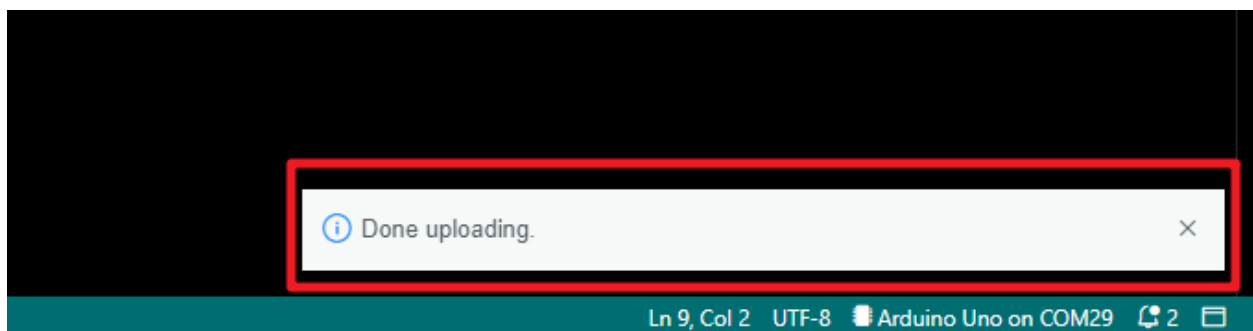


### 3. Upload sketch

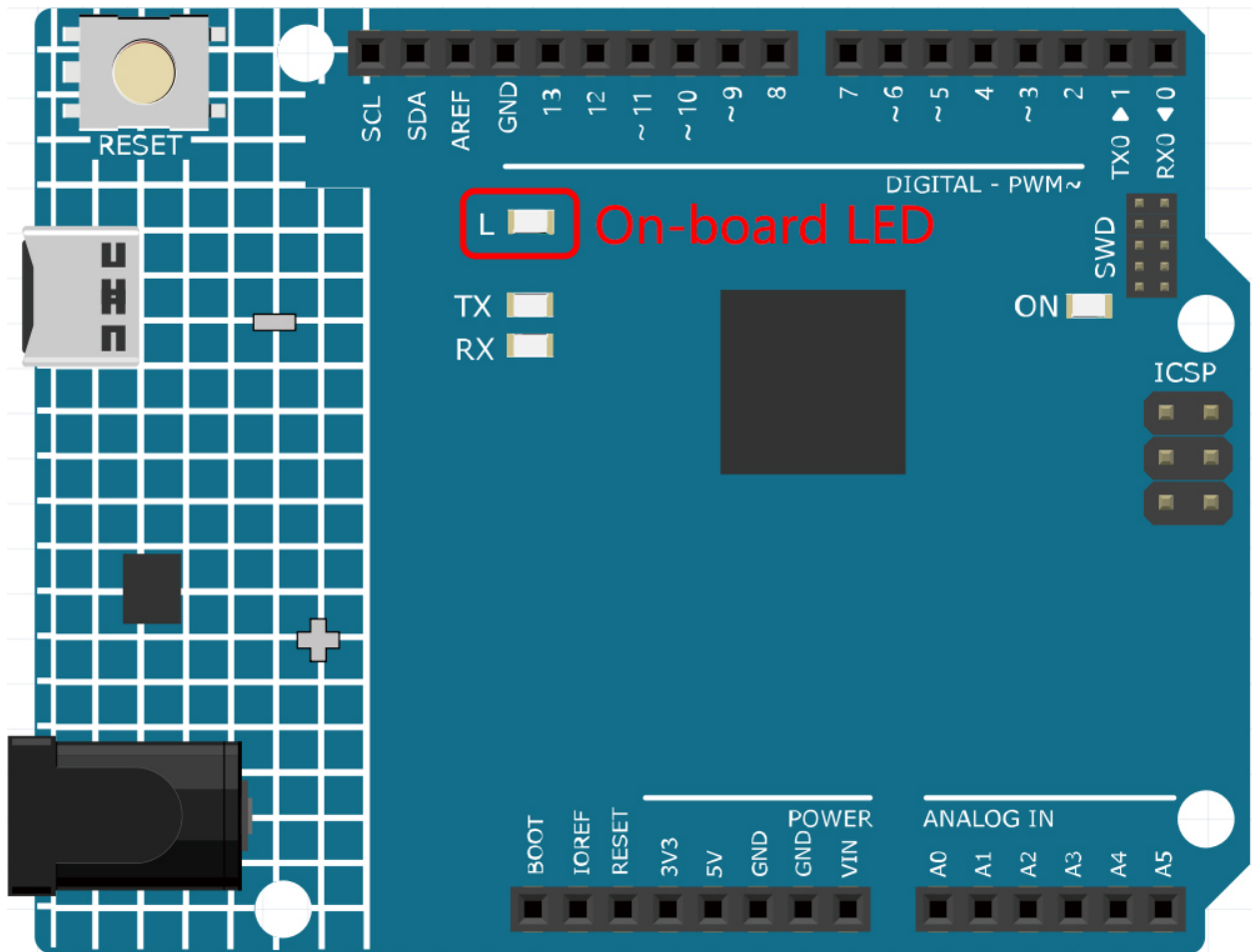
After completing the above steps, click the **Upload** button to upload this sketch to the board.



If successful, you will be able to see the following prompt.



At the same time, the on-board LED blink.



The Arduino board will automatically run the sketch after power is applied after the sketch is uploaded. The running program can be overwritten by uploading a new sketch.

### 2.3.5 Arduino Program Structure

Let's take a look at the new sketch file. Although it has a few lines of code itself, it is actually an “empty” sketch. Uploading this sketch to the development board will cause nothing to happen.

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

void loop() {
  // put your main code here, to run repeatedly:
}
```

If we remove `setup()` and `loop()` and make the sketch a real blank file, you will find that it does not pass the verification. They are the equivalent of the human skeleton, and they are indispensable.

During sketching, `setup()` is run first, and the code inside it (inside `{}`) is run after the board is powered up or reset

and only once. `loop()` is used to write the main feature, and the code inside it will run in a loop after `setup()` is executed.

To better understand `setup()` and `loop()`, let's use four sketches. Their purpose is to make the on-board LED of the Arduino blink. Please run each experiment in turn and record them specific effects.

- Sketch 1: Make the on-board LED blink continuously.

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(13,OUTPUT);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(13,HIGH);  
    delay(500);  
    digitalWrite(13,LOW);  
    delay(500);  
}
```

- Sketch 2: Make the on-board LED blink only once.

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(13,OUTPUT);  
    digitalWrite(13,HIGH);  
    delay(500);  
    digitalWrite(13,LOW);  
    delay(500);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
}
```

- Sketch 3: Make the on-board LED blink slowly once and then blink quickly.

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(13,OUTPUT);  
    digitalWrite(13,HIGH);  
    delay(1000);  
    digitalWrite(13,LOW);  
    delay(1000);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(13,HIGH);  
    delay(200);  
    digitalWrite(13,LOW);  
    delay(200);  
}
```

- Sketch 4: Report an error.



```

void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

digitalWrite(13,HIGH);
delay(1000);
digitalWrite(13,LOW);
delay(1000);

void loop() {
    // put your main code here, to run repeatedly:
}

```

With the help of these sketches, we can summarize several features of `setup-loop`.

- `loop()` will be run repeatedly after the board is powered up.
- `setup()` will run only once after the board is powered up.
- After the board is powered up, `setup()` will run first, followed by `loop()`.
- The code needs to be written within the `{}` scope of `setup()` or `loop()`, out of the framework will be an error.

**Note:** Statements such as `digitalWrite(13,HIGH)` are used to control the on-board LED, and we will talk about their usage in detail in later chapters.

## 2.3.6 Sketch Writing Rule

If you ask a friend to turn on the lights for you, you can say “Turn on the lights.”, or “Lights on, bro.”, you can use any tone of voice you want.

However, if you want the Arduino board to do something for you, you need to follow the Arduino program writing rules to type in the commands.

This chapter contains the basic rules of the Arduino language and will help you understand how to translate natural language into code.

Of course, this is a process that takes time to get familiar with, and it is also the most error-prone part of the process for newbies, so if you make mistakes often, it’s okay, just try a few more times.

### Semicolon ;

Just like writing a letter, where you write a period at the end of each sentence as the end, the Arduino language requires you to use `;` to tell the board the end of the command.

Take the familiar “onboard LED blinking” example. A healthy sketch should look like this.

Example:

```

void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

```

(continues on next page)

(continued from previous page)

```
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(13,HIGH);  
  delay(500);  
  digitalWrite(13,LOW);  
  delay(500);  
}
```

Next, let's take a look at the following two sketches and guess if they can be correctly recognized by Arduino before running them.

Sketch A:

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(13,HIGH)  
  delay(500)  
  digitalWrite(13,LOW)  
  delay(500)  
}
```

Sketch B:

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(13,  
HIGH); delay  
  (500  
  );  
  digitalWrite(13,  
  
  LOW);  
      delay(500)  
  ;  
}
```

The result is that **Sketch A** reports an error and **Sketch B** runs.

- The errors in **Sketch A** are missing ; and although it looks normal, the Arduino can't read it.
- **Sketch B**, looks anti-human, but in fact, indentation, line breaks and spaces in statements are things that do not exist in Arduino programs, so to the Arduino compiler, it looks the same as in the example.

However, please don't write your code as **Sketch B**, because it is usually natural people who write and view the code,

so don't get yourself into trouble.

### Curlybraces {}

{ } is the main component of the Arduino programming language, and they must appear in pairs. A better programming convention is to insert a structure that requires curly braces by typing the right curly brace directly after typing the left curly brace, and then moving the cursor between the curly braces to insert the statement.

### Comment //

Comment is the part of the sketch that the compiler ignores. They are usually used to tell others how the program works.

If we write two adjacent slashes in a line of code, the compiler will ignore anything up to the end of the line.

If we create a new sketch, it comes with two comments, and if we remove these two comments, the sketch will not be affected in any way.

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

}

void loop() {
    // put your main code here, to run repeatedly:

}
```

Comment is very useful in programming, and several common uses are listed below.

- Usage A: Tell yourself or others what this section of code does.

```
void setup() {
    pinMode(13,OUTPUT); //Set pin 13 to output mode, it controls the onboard LED
}

void loop() {
    digitalWrite(13,HIGH); // Activate the onboard LED by setting pin 13 high
    delay(500); // Status quo for 500 ms
    digitalWrite(13,LOW); // Turn off the onboard LED
    delay(500); // Status quo for 500 ms
}
```

- Usage B: Temporarily invalidate some statements (without deleting them) and uncomment them when you need to use them, so you don't have to rewrite them. This is very useful when debugging code and trying to locate program errors.

```
void setup() {
    pinMode(13,OUTPUT);
    // digitalWrite(13,HIGH);
    // delay(1000);
    // digitalWrite(13,LOW);
    // delay(1000);
}
```

(continues on next page)

(continued from previous page)

```
void loop() {  
    digitalWrite(13,HIGH);  
    delay(200);  
    digitalWrite(13,LOW);  
    delay(200);  
}
```

**Note:** Use the shortcut Ctrl+/ to help you quickly comment or uncomment your code.

### Comment `/**/`

Same as `//` for comments. This type of comment can be more than one line long, and once the compiler reads `/*`, it ignores anything that follows until it encounters `*/`.

Example 1:

```
/* Blink */  
  
void setup() {  
    pinMode(13,OUTPUT);  
}  
  
void loop() {  
    /*  
    The following code will blink the onboard LED  
    You can modify the number in delay() to change the blinking frequency  
    */  
    digitalWrite(13,HIGH);  
    delay(500);  
    digitalWrite(13,LOW);  
    delay(500);  
}
```

### `#define`

This is a useful C++ tool.

```
#define identifier token-string
```

The compiler automatically replaces `identifier` with `token-string` when it reads it, which is usually used for constant definitions.

As an example, here is a sketch that uses `define`, which improves the readability of the code.

```
#define ONBOARD_LED 13  
#define DELAY_TIME 500  
  
void setup() {  
    pinMode(ONBOARD_LED,OUTPUT);
```

(continues on next page)

(continued from previous page)

```

}

void loop() {
    digitalWrite(ONBOARD_LED,HIGH);
    delay(DELAY_TIME);
    digitalWrite(ONBOARD_LED,LOW);
    delay(DELAY_TIME);
}

```

To the compiler, it actually looks like this.

```

void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}

```

We can see that the identifier is replaced and does not exist inside the program. Therefore, there are several caveats when using it.

1. A token-string can only be modified manually and cannot be converted into other values by arithmetic in the program.
2. Avoid using symbols such as ;. For example.

```

#define ONBOARD_LED 13;

void setup() {
    pinMode(ONBOARD_LED,OUTPUT);
}

void loop() {
    digitalWrite(ONBOARD_LED,HIGH);
}

```

The compiler will recognize it as the following, which is what will be reported as an error.

```

void setup() {
    pinMode(13;,OUTPUT);
}

void loop() {
    digitalWrite(13;,HIGH);
}

```

**Note:** A naming convention for #define is to capitalize identifier to avoid confusion with variables.

### 2.3.7 Variable

The variable is one of the most powerful and critical tools in a program. It helps us to store and call data in our programs.

The following sketch file uses variables. It stores the pin numbers of the on-board LED in the variable `ledPin` and a number “500” in the variable `delayTime`.

```
int ledPin = 13;
int delayTime = 500;

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

void loop() {
  digitalWrite(ledPin,HIGH);
  delay(delayTime);
  digitalWrite(ledPin,LOW);
  delay(delayTime);
}
```

Wait, is this a duplicate of what `#define` does? The answer is NO.

- The role of `#define` is to simply and directly replace text, it is not considered by the compiler as part of the program.
- A variable, on the other hand, exists within the program and is used to store and call value. A variable can also modify its value within the program, something that a define cannot do.

The sketch file below self-adds to the variable and it will cause the on-board LED to blink longer after each blink.

```
int ledPin = 13;
int delayTime = 500;

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

void loop() {
  digitalWrite(ledPin,HIGH);
  delay(delayTime);
  digitalWrite(ledPin,LOW);
  delay(delayTime);
  delayTime = delayTime+200; //Each execution increments the value by 200
}
```

## Declare a variable

Declaring a variable means creating a variable.

To declare a variable, you need two things: the data type, and the variable name. The data type needs to be separated from the variable by a space, and the variable declaration needs to be terminated by a ;.

Let's use this variable as an example.

```
int delayTime;
```

### Data Type

Here `int` is a data type called integer type, which can be used to store integers from -32768 to 32766. It can also not be used to store decimals.

Variables can hold different kinds of data other than integers. The Arduino language (which, remember, is C++) has built-in support for a few of them (only the most frequently used and useful are listed here):

- `float`: Store a decimal number, for example 3.1415926.
- `byte`: Can hold numbers from 0 to 255.
- `boolean`: Holds only two possible values, `True` or `False`, even though it occupies a byte in memory.
- `char`: Holds a number from -127 to 127. Because it is marked as a `char` the compiler will try to match it to a character from the .
- `string`: Can stores a string of characters, e.g. Halloween.

### Variable Name

You can set the variable to any name you want, such as `i`, `apple`, `Bruce`, `R2D2`, `Sectumsempra`, but there are some basic rules to follow.

1. describe what it is used for. Here, I named the variable `delayTime`, so you can easily understand what it does. It works fine if I name the variable `barryAllen`, but it confuses the person looking at the code.
2. Use regular nomenclature. You can use CamelCase like I did, with the initial T in `delayTime` so that it is easy to see that the variable consists of two words. Also, you can use UnderScoreCase to write the variable as `delay_time`. It doesn't affect the program's running, but it would help the programmer to read the code if you use the nomenclature you prefer.
3. Don't use keywords. Similar to what happens when we type "`int`", the Arduino IDE will color it to remind you that it is a word with a special purpose and cannot be used as a variable name. Change the name of the variable if it is colored.
4. Special symbols are not allowed. For example, space, `#`, `$`, `/`, `+`, `%`, etc. The combination of English letters (case sensitive), underscores, and numbers (but numbers cannot be used as the first character of a variable name) is rich enough.

### Assign a value to a variable

Once we have declared the variable, it is time to store the data. We use the assignment operator (i.e. `=`) to put value into the variable.

We can assign values to the variable as soon as we declare it.

```
int delayTime = 500;
```

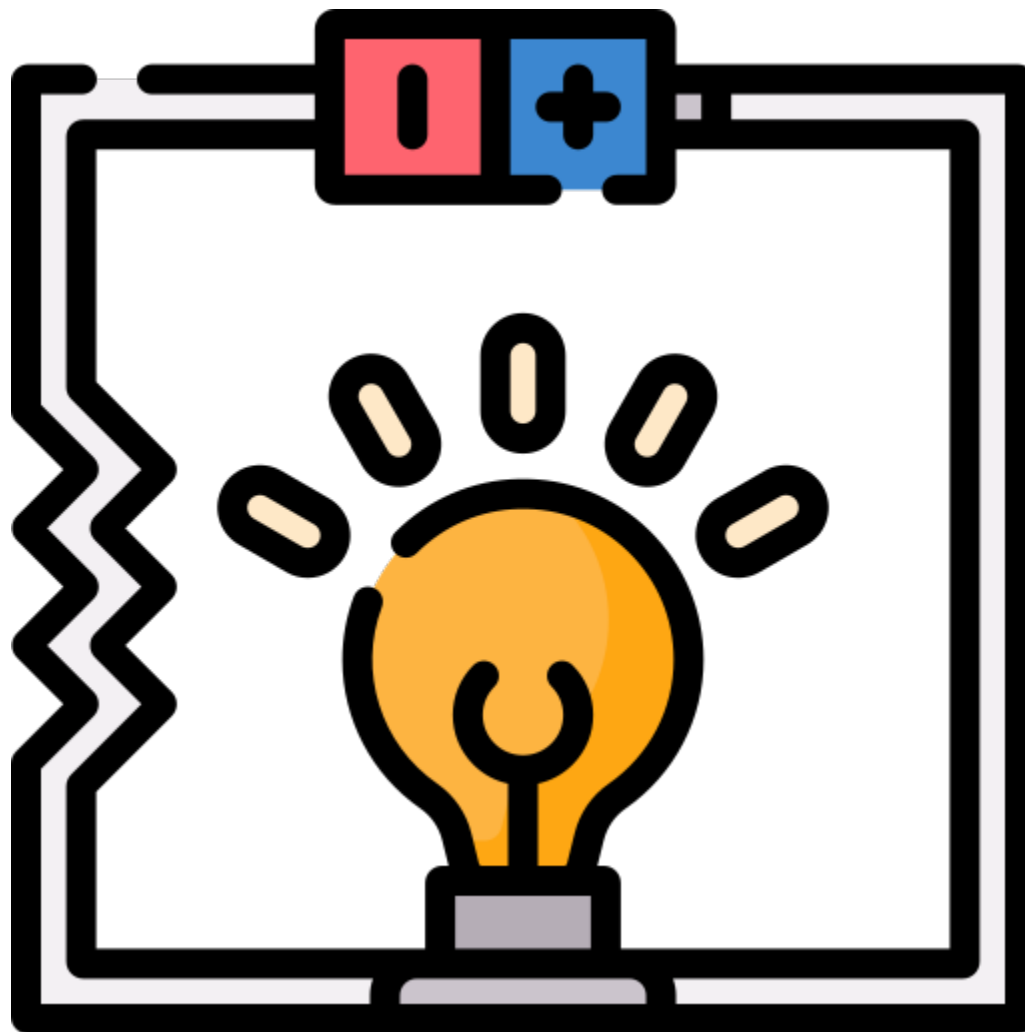
It is also possible to assign a new value to it at some time.

```
int delayTime; // no value
delayTime = 500; // value is 500
delayTime = delayTime +200; // value is 700
```

### 2.3.8 How to Build the Circuit

Many of the things you use every day are powered by electricity, like the lights in your house and the computer you're reading.

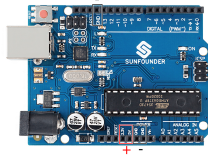
To use electricity, you must build an electrical circuit. Basically, a circuit is a path through which electricity flows, or an electronic circuit, and is made up of electrical devices and components (appliances) that are connected in a certain way, such as resistors, capacitors, power supplies, and switches.



A circuit is a closed path in which electrons move to create an electric current. To flow current, there must be a conducting path between the positive terminal of the power supply and the negative terminal, which is called a closed circuit (if it is broken, it is called an open circuit.) .

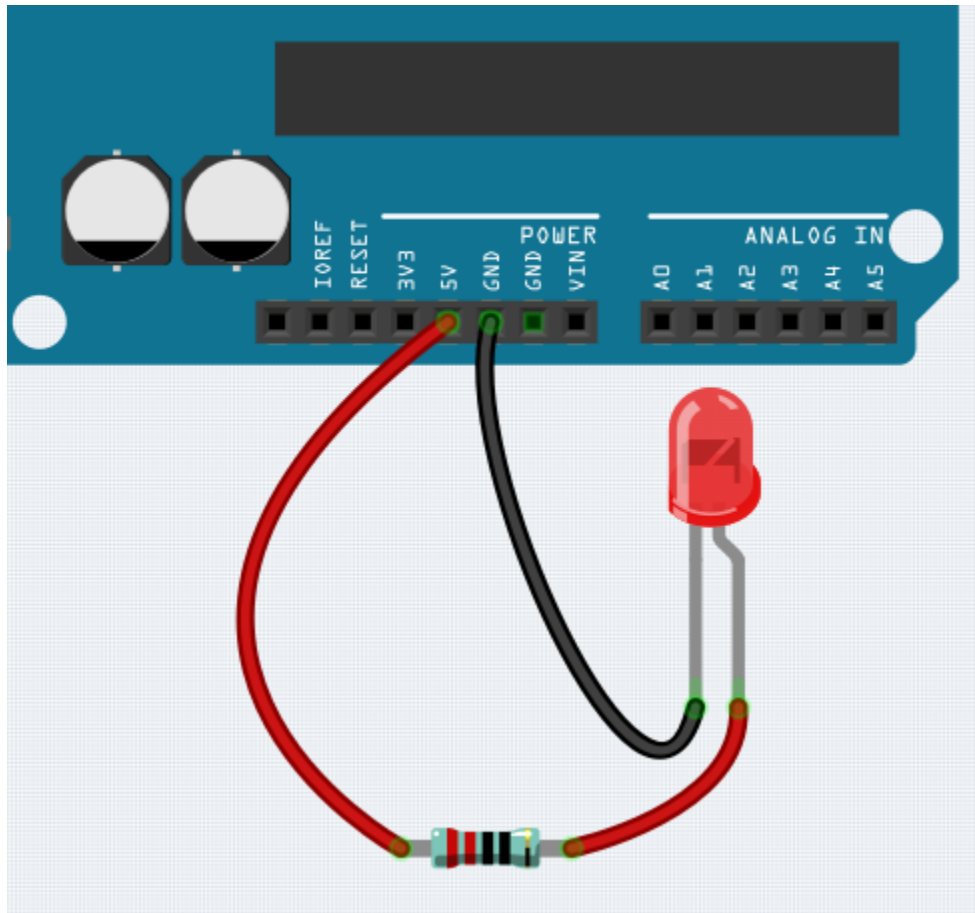
The Arduino Board has some power output pins (positive) and some ground pins (negative). You can use these pins as the positive and negative sides of the power supply by plugging the power source into the board.





With electricity, you can create works with light, sound, and motion. You can light up an LED by connecting the long pin to the positive terminal and the short pin to the negative terminal. The LED will break down very quickly if you do this, so you need to add a 220\* resistor inside the circuit to protect it.

The circuit they form is shown below.



You may have questions this time: how do I build this circuit? Hold the wires by hand, or tape the pins and wires?

In this situation, solderless breadboards will be your strongest allies.

### Hello, Breadboard!

A breadboard is a rectangular plastic plate with a bunch of small holes. These holes allow us to easily insert electronic components and build electronic circuits. Breadboards do not permanently fix electronic components, so we can easily repair a circuit and start over if something goes wrong.

---

**Note:** There is no need for special tools to use breadboards. However, many electronic components are very small, and a pair of tweezers can help us to pick up small parts better.

---

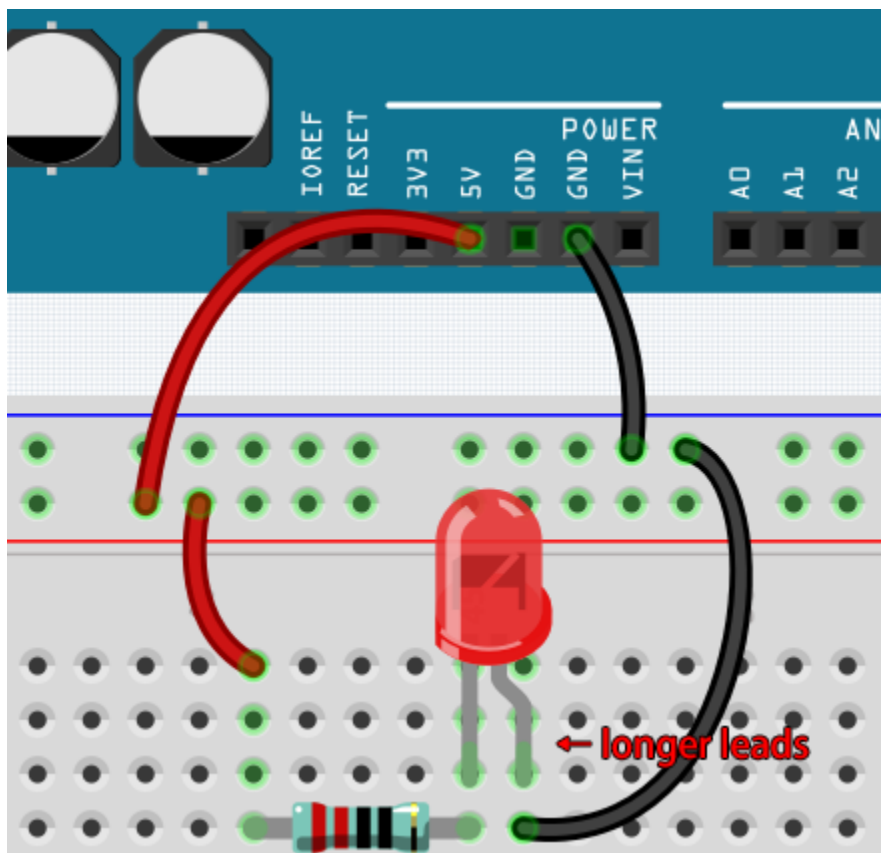
On the Internet, we can find a lot of information about breadboards.

- [How to Use a Breadboard - Science Buddies](#)
- [What is a BREADBOARD? - Makezine](#)

Here are some things you should know about breadboards.

1. Each half-row group (such as column A-E in row 1 or column F-J in row 3) is connected. Therefore, if an electrical signal flows in from A1, it can flow out from B1, C1, D1, E1, but not from F1 or A2.
2. In most cases, both sides of the breadboard are used as power buses, and the holes in each column (about 50 holes) are connected together. As a general rule, positive power supplies are connected to the holes near the red wire, and negative power supplies are connected to the holes near the blue wire.
3. In a circuit, current flows from the positive pole to the negative pole after passing through the load. In this case, a short circuit may occur.

Let us follow the direction of the current to build the circuit!



1. In this circuit, we use the 5V pin of the board to power the LED. Use a male-to-male (M2M) jumper wire to connect it to the red power bus.
2. To protect the LED, the current must pass through a 220 ohm resistor. Connect one end (either end) of the resistor to the red power bus, and the other end to the free row of the breadboard.

---

**Note:** The color ring of the 220 ohm resistor is red, red, black, black and brown.

---

3. If you pick up the LED, you will see that one of its leads is longer than the other. Connect the longer lead to the same row as the resistor, and the shorter lead to the other row.

---

**Note:** The longer lead is the anode, which represents the positive side of the circuit; the shorter lead is the cathode, which represents the negative side.

The anode needs to be connected to the GPIO pin through a resistor; the cathode needs to be connected to the GND pin.

---

4. Using a male-to-male (M2M) jumper wire, connect the LED short pin to the breadboard's negative power bus.
5. Connect the GND pin of board to the negative power bus using a jumper.

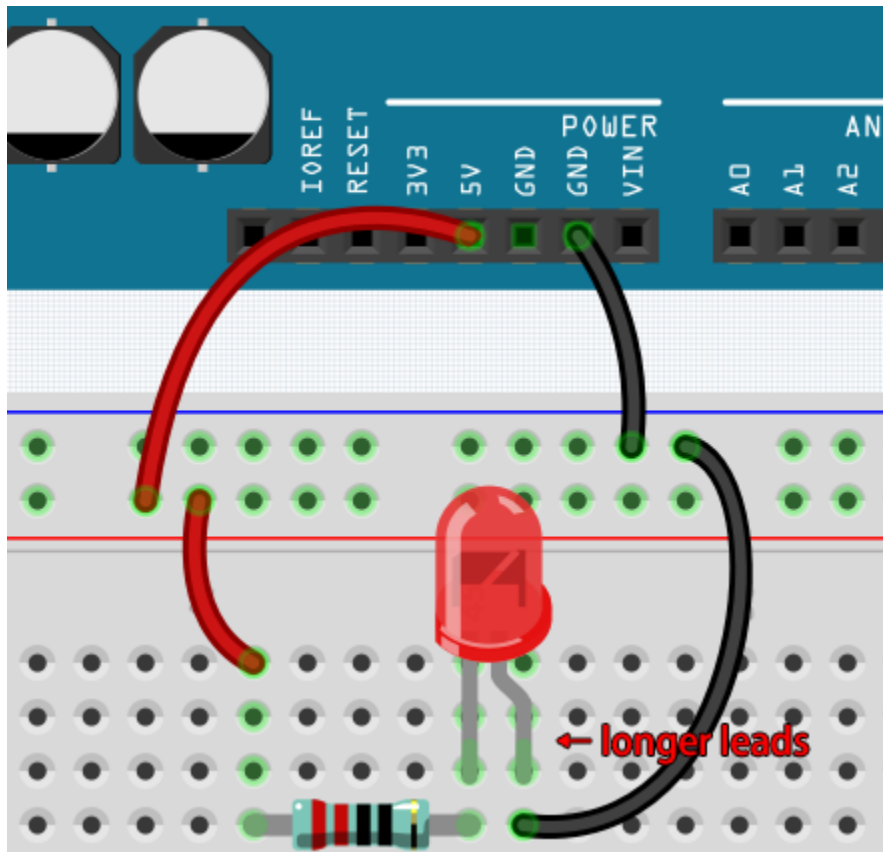
### **Beware of short circuits**

Short circuits can occur when two components that shouldn't be connected are "accidentally" connected. This kit includes resistors, transistors, capacitors, LEDs, etc. that have long metal pins that can bump into each other and cause a short. Some circuits are simply prevented from functioning properly when a short occurs. Occasionally, a short circuit can damage components permanently, especially between the power supply and the ground bus, causing the circuit to get very hot, melting the plastic on the breadboard and even burning the components!

Therefore, always make sure that the pins of all the electronics on the breadboard are not touching each other.

### **Direction of the circuit**

There is an orientation to circuits, and the orientation plays a significant role in certain electronic components. There are some devices with polarity, which means they must be connected correctly based on their positive and negative poles. Circuits built with the wrong orientation will not function properly.



If you reverse the LED in this simple circuit that we built earlier, you will find that it no longer works.

In contrast, some devices have no direction, such as the resistors in this circuit, so you can try inverting them without affecting the LEDs' normal operation.

Most components and modules with labels such as “+”, “-”, “GND”, “VCC” or have pins of different lengths must be connected to the circuit in a specific way.

### Protection of the circuit

Current is the rate at which electrons flow past a point in a complete electrical circuit. At its most basic, current = flow. An ampere (AM-pir), or amp, is the international unit used for measuring current. It expresses the quantity of electrons (sometimes called “electrical charge”) flowing past a point in a circuit over a given time.

The driving force (voltage) behind the flow of current is called voltage and is measured in volts (V).

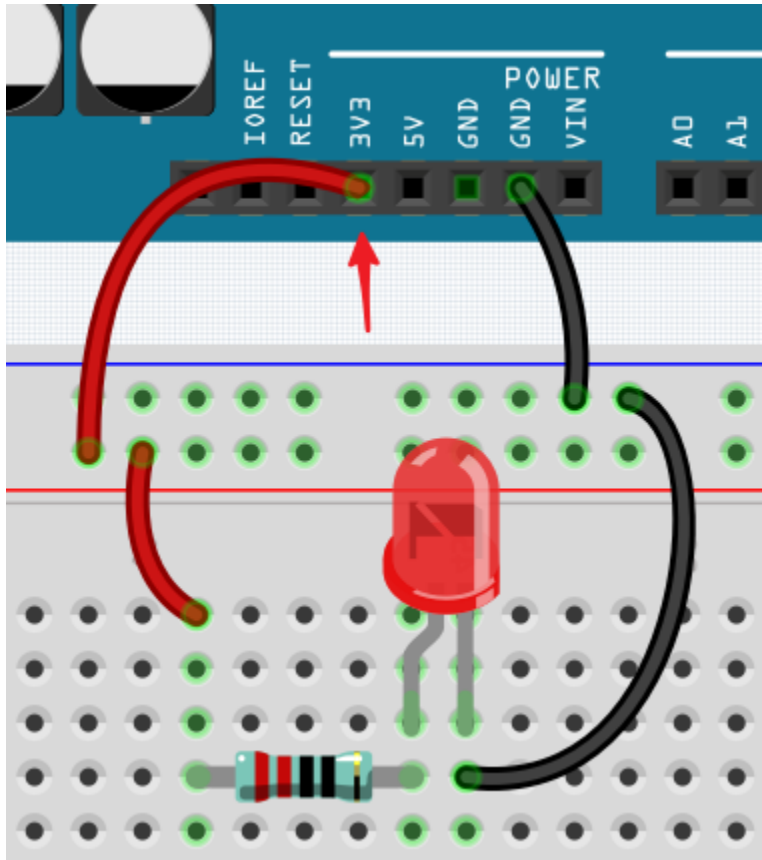
Resistance (R) is the property of the material that restricts the flow of current, and it is measured in ohms ( $\Omega$ ).

According to Ohm's law (as long as the temperature remains constant), current, voltage, and resistance are proportional. A circuit's current is proportional to its voltage and inversely proportional to its resistance.

Therefore, current (I) = voltage (V) / resistance (R).

- [Ohm's law - Wikipedia](#)

About Ohm's law we can do a simple experiment.



By changing the wire connecting 5V to 3.3V, the LED gets dimmer. If you change the resistor from 220ohm to 1000ohm (color ring: brown, black, black, brown and brown), you will notice that the LED becomes dimmer than before. The larger the resistor, the dimmer the LED.

---

**Note:** For an introduction to resistors and how to calculate resistance values, see [Resistor](#).

---

Most packaged modules only require access to the proper voltage (usually 3.3V or 5V), such as ultrasonic module.

However, in your self-built circuits, you need to be aware of the supply voltage and resistor usage for electrical devices.

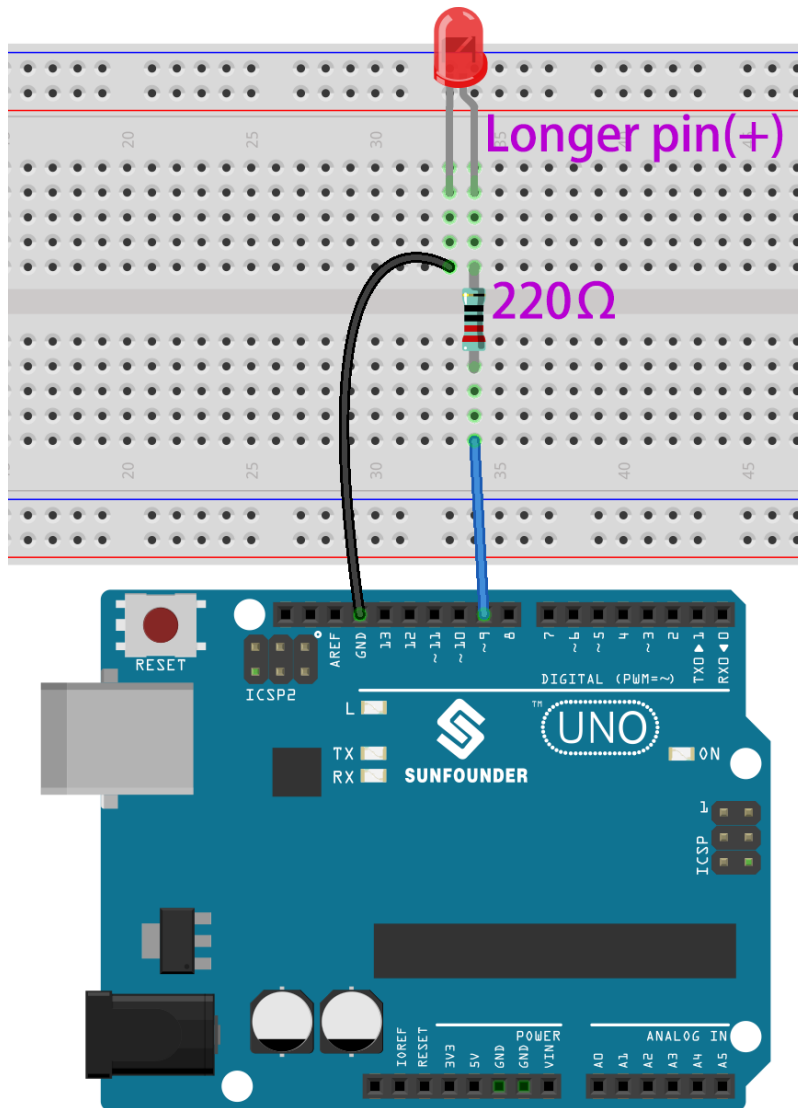
As an example, LEDs usually consume 20mA of current, and their voltage drop is about 1.8V. According to Ohm's law, if we use 5V power supply, we need to connect a minimum of 160ohm  $((5-1.8)/20\text{mA})$  resistor in order not to burn out the LED.

### Control circuit with Arduino

Now that we have a basic understanding of Arduino programming and electronic circuits, it's time to face the most critical question: How to control circuits with Arduino.

Simply put, the way Arduino controls a circuit is by changing the level of the pins on the board. For example, when controlling an on-board LED, it is writing a high or low level signal to pin 13.

Now let's try to code the Arduino board to control the blinking LED on the breadboard. Build the circuit so that the LED is connected to pin 9.



Next, upload this sketch to the Arduino development board.

```
int ledPin = 9;
int delayTime = 500;

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

void loop() {
  digitalWrite(ledPin,HIGH);
  delay(delayTime);
  digitalWrite(ledPin,LOW);
  delay(delayTime);
}
```

This sketch is very similar to the one we used to control the blinking of the on-board LED, the difference is that the value of `ledPin` has been changed to 9. This is because we are trying to control the level of pin 9 this time.

Now you can see the LED on the breadboard blinking.

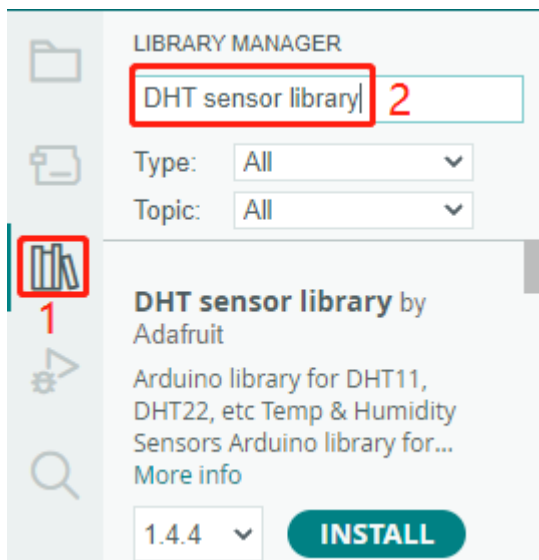
## 2.4 How to add libraries? (Important)

A library is a collection of pre-written code or functions that extend the capabilities of the Arduino IDE. Libraries provide ready-to-use code for various functionalities, allowing you to save time and effort in coding complex features.

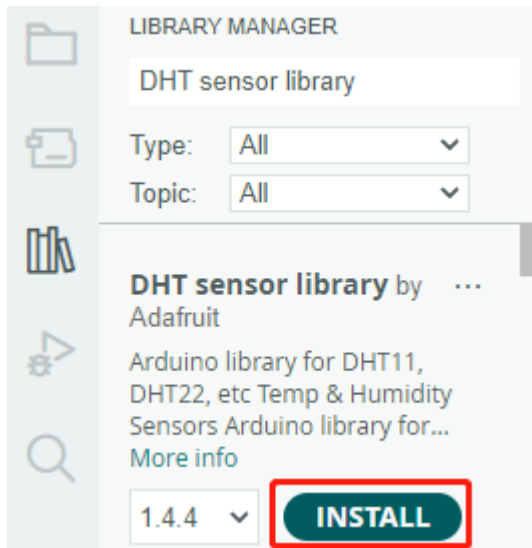
Many libraries are available directly through the Arduino Library Manager. You can access the Library Manager by following these steps:

1. In the **Library Manager**, you can search for the desired library by name or browse through different categories.

**Note:** In projects where library installation is required, there will be prompts indicating which libraries to install. Follow the instructions provided, such as “The DHT sensor library library is used here, you can install it from the Library Manager.” Simply install the recommended libraries as prompted.



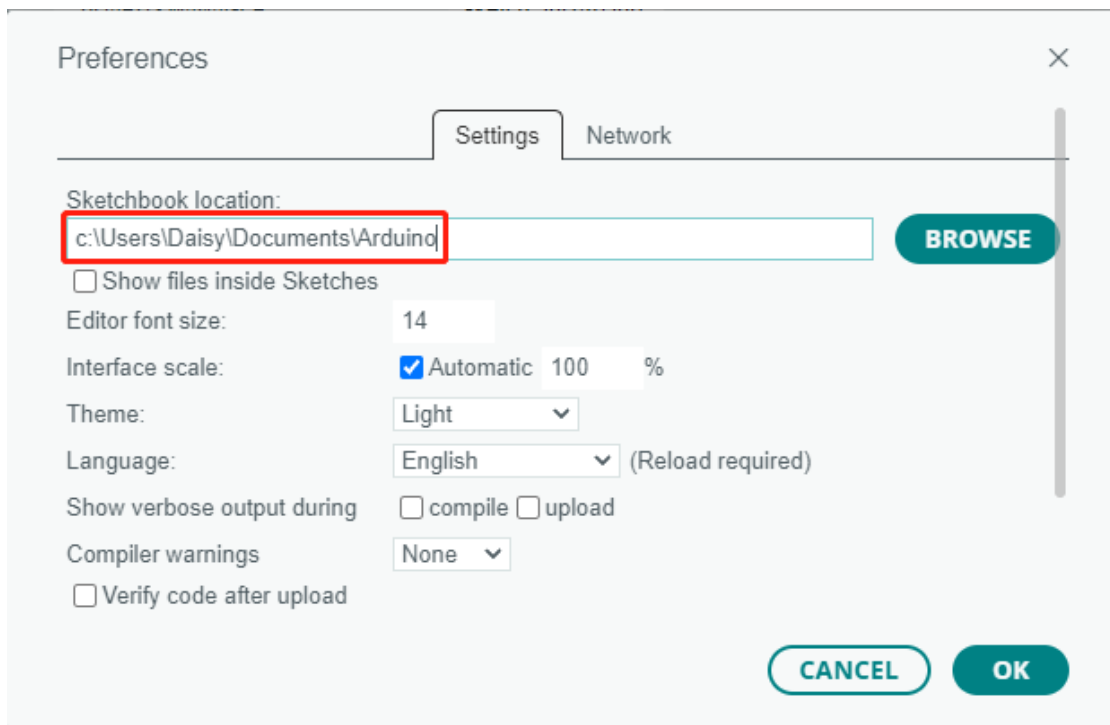
2. Once you find the library you want to install, click on it and then click the **Install** button.



3. The Arduino IDE will automatically download and install the library for you.

**Note:** The libraries installed using either of the above methods can be found in the default library directory of the Arduino IDE, which is usually located at `C:\Users\xxx\Documents\Arduino\libraries`.

If your library directory is different, you can check it by going to **File -> Preferences**.





## PROJECTS

By following this chapter, Arduino users will gain a solid understanding of the board's features and capabilities, as well as learn how to utilize the Uno in their projects. The chapter also offers step-by-step tutorials and sample code examples to help users get started with programming and interfacing different sensors, actuators, and displays using the Uno R4 Minima board.

Whether you are a beginner or an experienced Arduino enthusiast, this chapter provides valuable resources to enhance your knowledge and skills with the Arduino Uno R4 Minima board, enabling you to unleash your creativity and bring your project ideas to life.

You can download the code package for the SunFounder Uno R4 Basic Kit from the following link:

You can download the code package for the SunFounder Uno R4 Basic Kit from the following link:

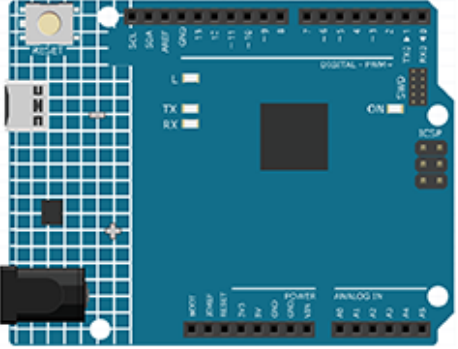

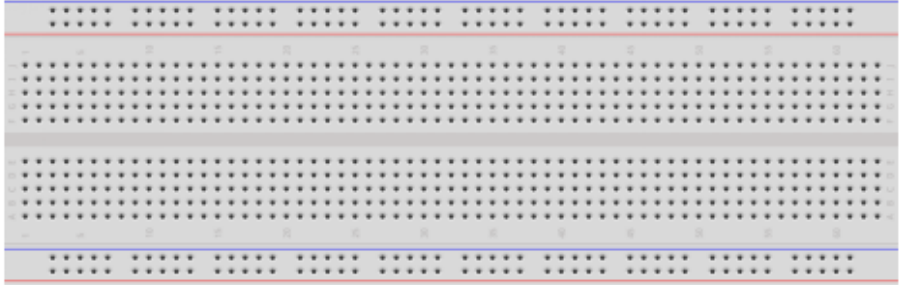


- [SunFounder Uno R4 Basic Kit](#)
- Or check out the code at [.](#)

## 3.1 Lesson 1 Blinking LED

### 3.1.1 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.

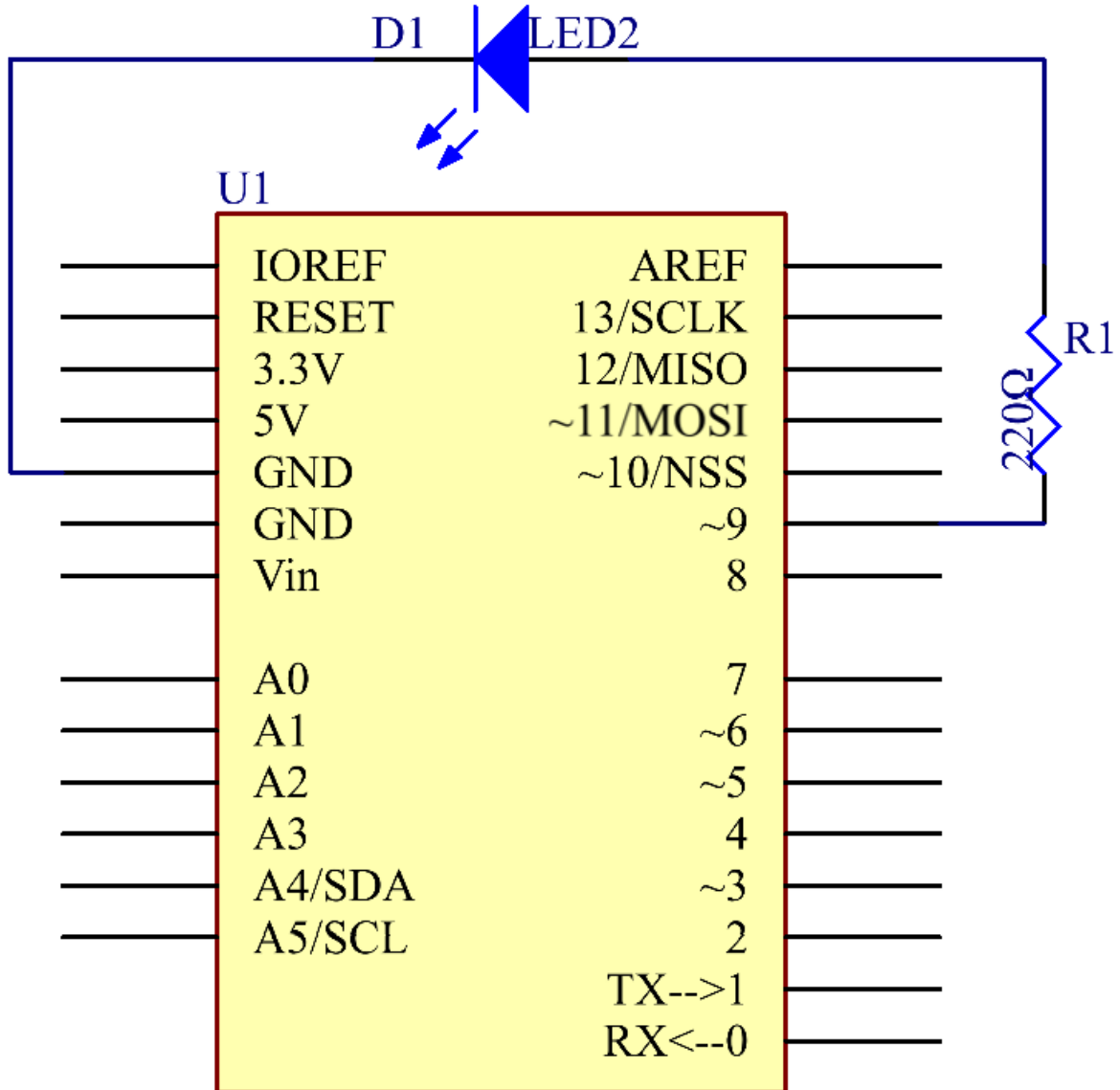
### 3.1.2 Components

<p>1 * Arduino Board</p> 	<p>1 * USB Cable</p> 
<p>1 * Breadboard</p> 	<p>1 * Resistor (220Ω)</p> 
	<p>1 * LED</p> 

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LED*
- *Resistor*

### 3.1.3 Schematic Diagram

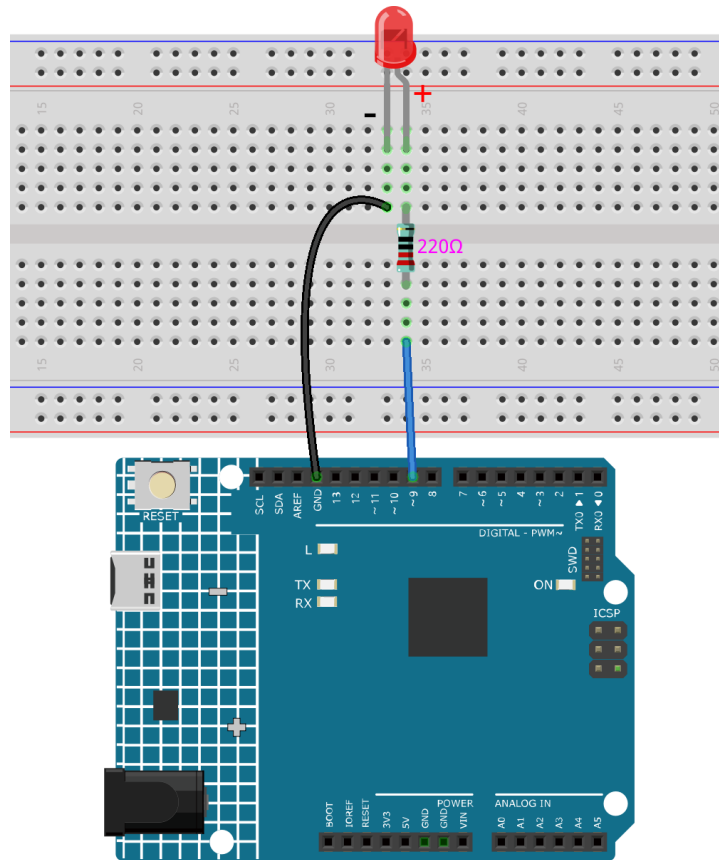
Connect one end of the 220ohm resistor to pin 9 of the Uno and the other end to the anode (the long pin) of the LED, and the cathode (the short pin) of the LED to GND. When the pin 9 outputs high level, the current gets through the current limiting resistor to the anode of the LED. And since the cathode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



### 3.1.4 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\_1\_Blinking\_LED.ino code file in the path of r4-basic-kit-main\Code\Lesson\_1\_Blinking\_LED

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

**Step 4:** Upload the sketch to the board.

You should now see the LED blinking.

### 3.1.5 Code

#### 3.1.6 Code Analysis

##### 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.

##### 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.

#### 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.

### 3.1.7 Experiment Summary

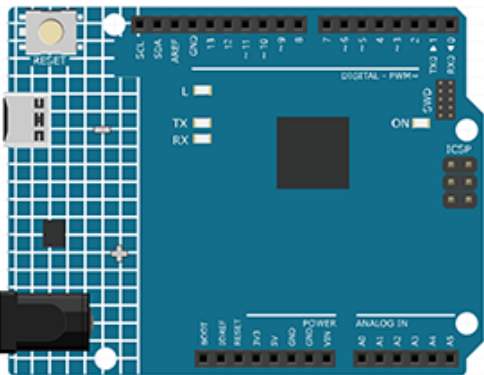

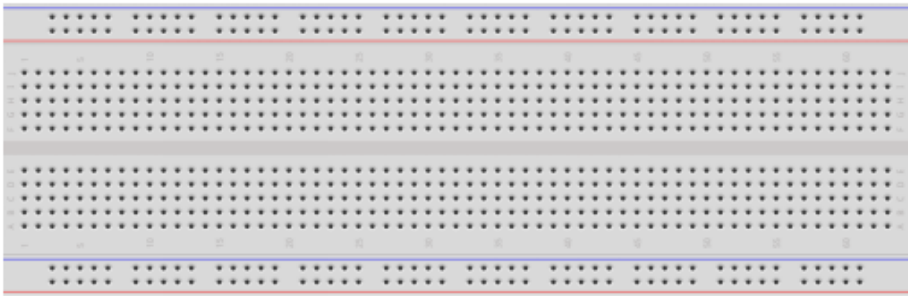



Through this experiment, you have learned how to turn on an LED. You can also change the blinking frequency of the LED by changing the *num* value in the delay function *delay(num)*. For example, change it to **delay(250)** and you will find that the LED blinks more quickly.

## 3.2 Lesson 2 Flowing LED Lights

### 3.2.1 Introduction

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

### 3.2.2 Components

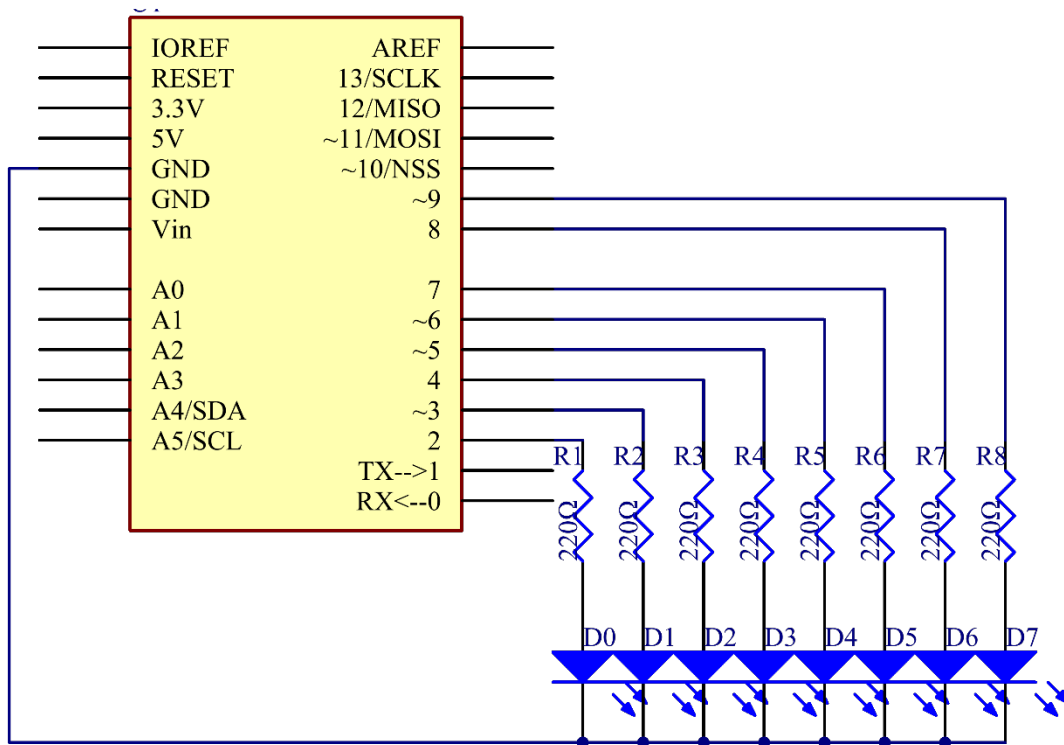
<p>1 * Arduino Board</p> 	<p>1 * USB Cable</p> 
<p>1 * Breadboard</p> 	<p>Several Jump Wires</p>  <p>8 * Resistor (220Ω)</p>  <p>8 * LED</p> 

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LED*

- Resistor

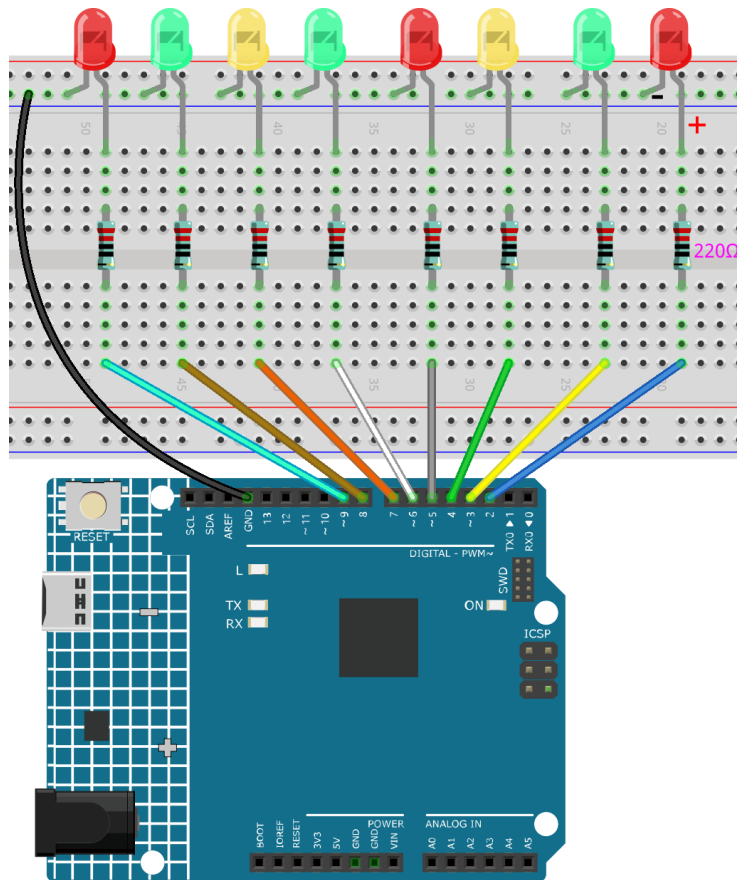
### 3.2.3 Schematic Diagram

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



### 3.2.4 Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.2.5 Code

#### 3.2.6 Code Analysis

**for() statement**

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

(continues on next page)



(continued from previous page)

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

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

### Set flowing led lights

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
}
```

### **Experiment Summary**

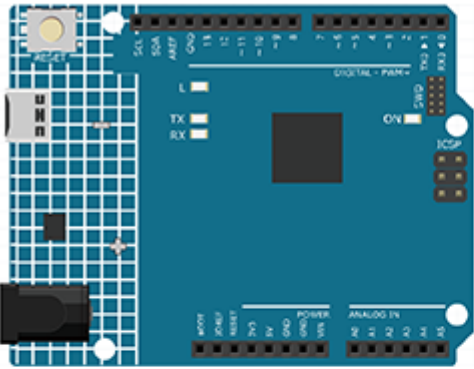



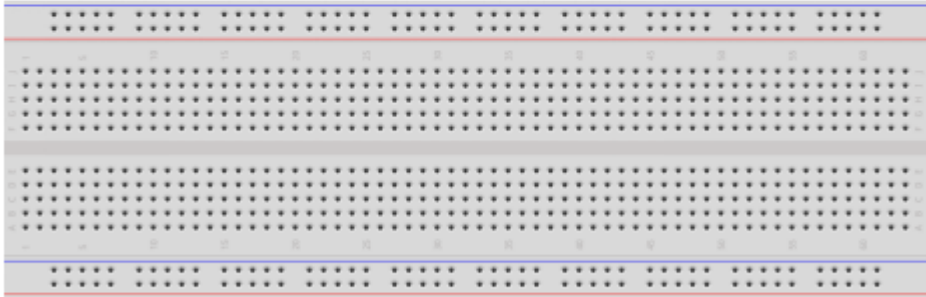


Through this experiment, you have learned how to use `for()` statement which is a very useful statement when you want to short the code.

## **3.3 Lesson 3 Controlling LED by Button**

### **3.3.1 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.

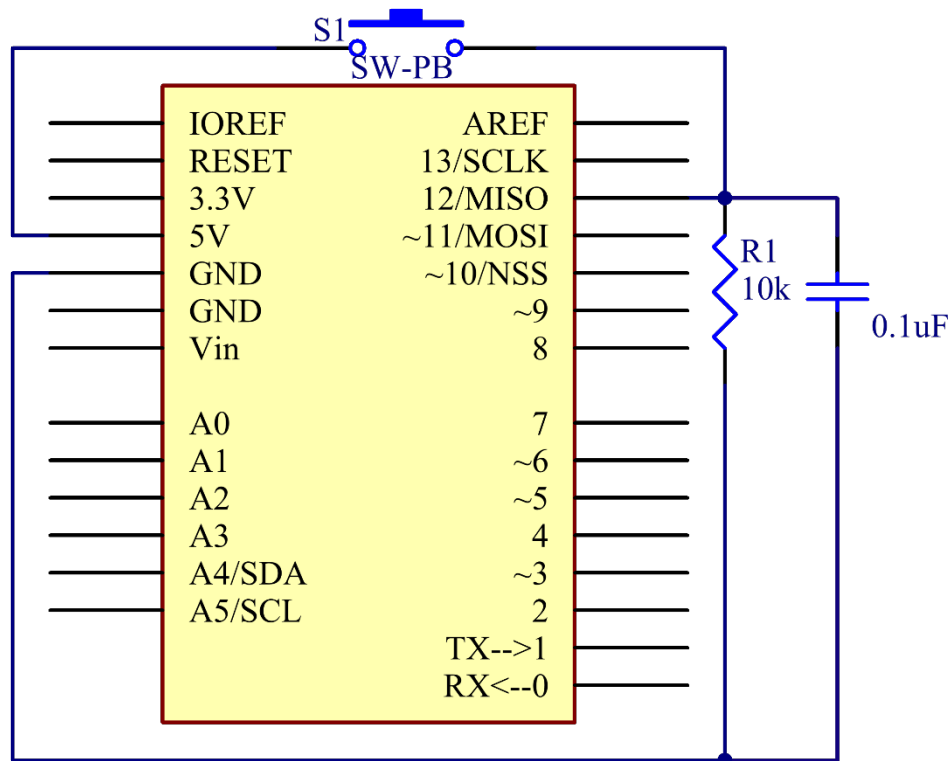
### 3.3.2 Components

<p>1 * Arduino Board</p> 	<p>1 * Resistor (10kΩ)</p> 	<p>1 * Button</p> 
	<p>1 * 104 Capacitor</p> 	
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *Capacitor*
- *Button*

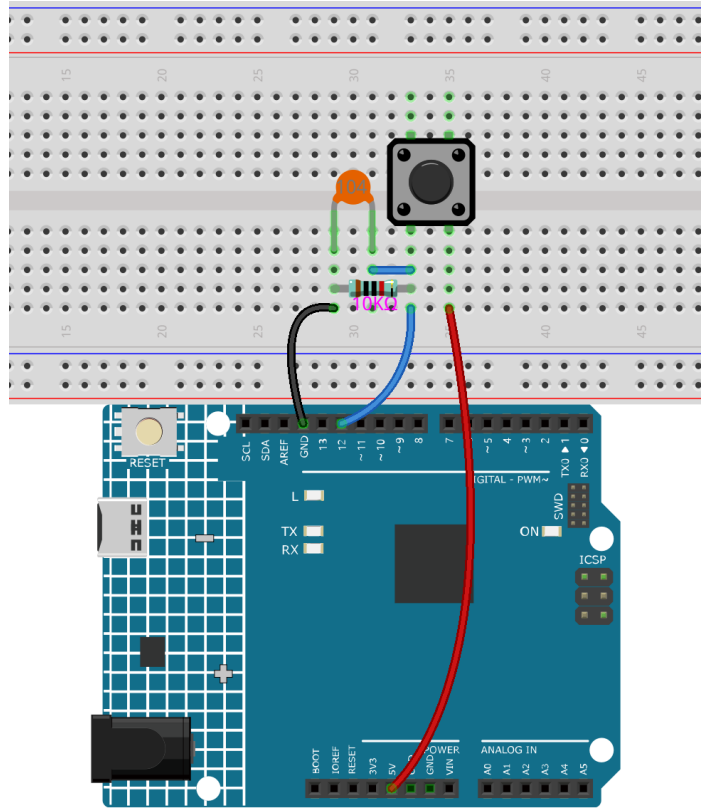
### 3.3.3 Schematic Diagram

Connect one end of the buttons to pin 12 which connects with a pull-down resistor and a 0.1uF (104) capacitor (to eliminate jitter and output a stable level when the button is working). Connect the other end of the resistor to GND and one of the pins at the other end of the button to 5V. When the button is pressed, pin 12 is 5V (HIGH) and set 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.



### 3.3.4 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.

### 3.3.5 Code

#### 3.3.6 Code Analysis

**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.

**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.

### 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.

### Press the button to make the buzzer sound

```
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*.

## 3.3.7 Experiment Summary

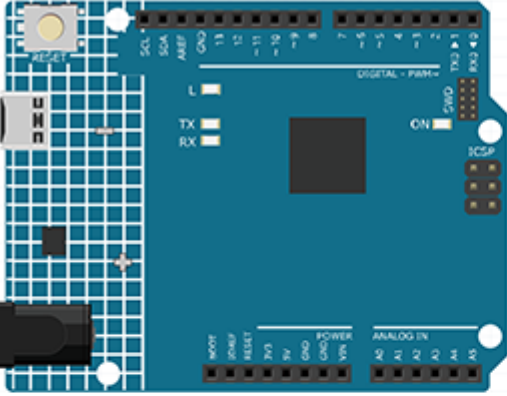


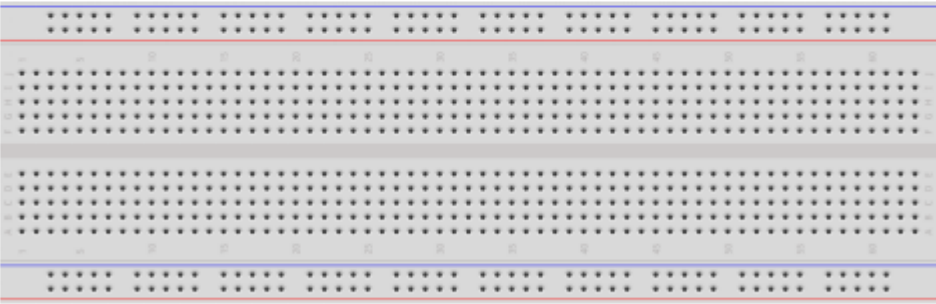


You can also change the code to: when the button is pressed, if (*buttonState*=HIGH). The LED goes out (*digitalWrite*(*ledPin*, LOW)). When the button is released (the *else*), the LED lights up ((*digitalWrite*(*ledPin*, HIGH)). You only need to replace the code in **if** with those in **else**.

## 3.4 Lesson 4 Doorbell

### 3.4.1 Introduction

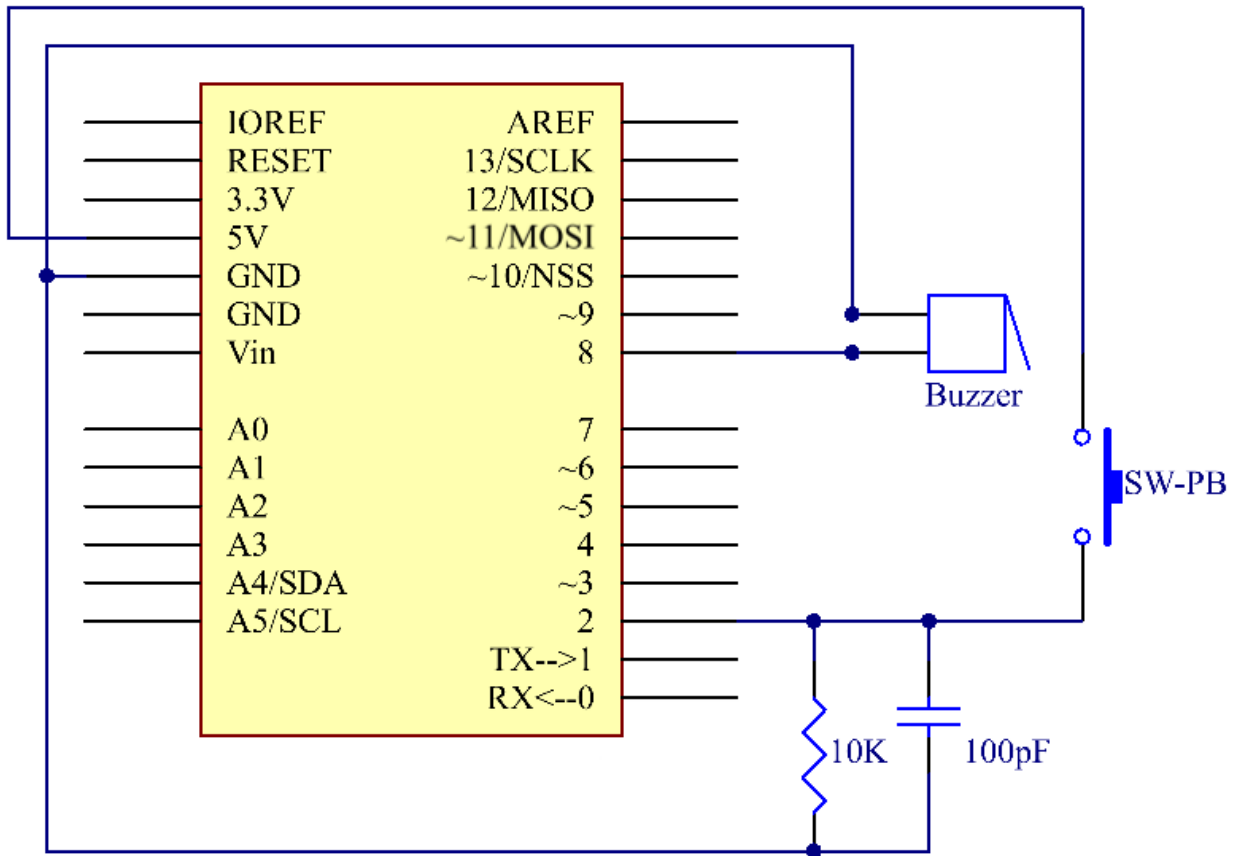
A buzzer is a great tool in your experiments whenever you want to make some sounds. In this lesson, we will learn how to drive an active buzzer to build a simple doorbell.

### 3.4.2 Components

<p>1 * Arduino Board</p> 	<p>1 * Buzzer (Active)</p> 	<p>1 * 104 Capacitor</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *Capacitor*
- *Button*
- *Buzzer*

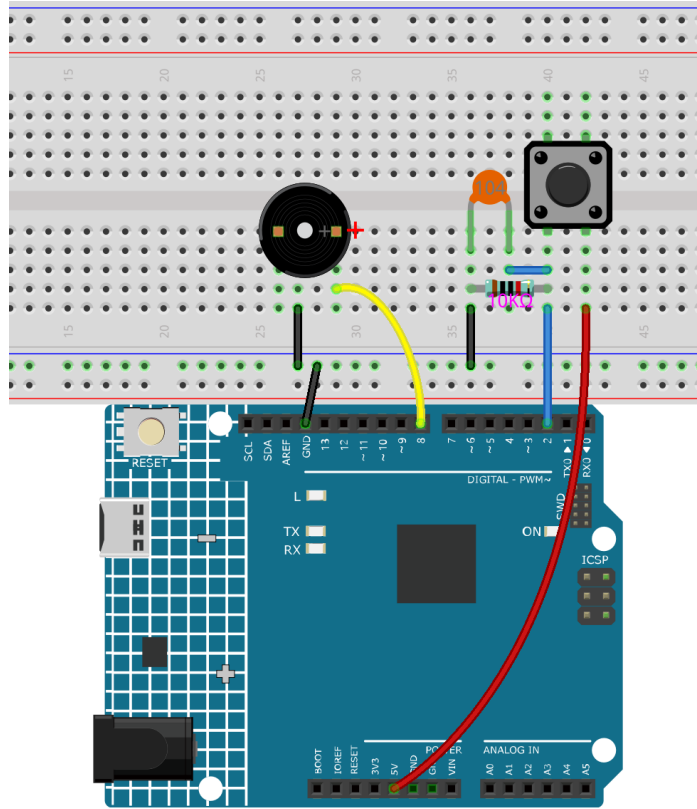
### 3.4.3 Schematic Diagram



### 3.4.4 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.

### 3.4.5 Code

### 3.4.6 Code Analysis

**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.

**Set the input and output status of the pins**

```
void setup()
```

(continues on next page)

(continued from previous page)

```
{
    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.

### 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.

### Press the button to make the buzzer sound

```
if (buttonState == HIGH ) //When press the button, run the following code.
{
    for (i = 0; i < 50; i++)
        /*When i=0, 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 = ibefore + 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
    }
}
```

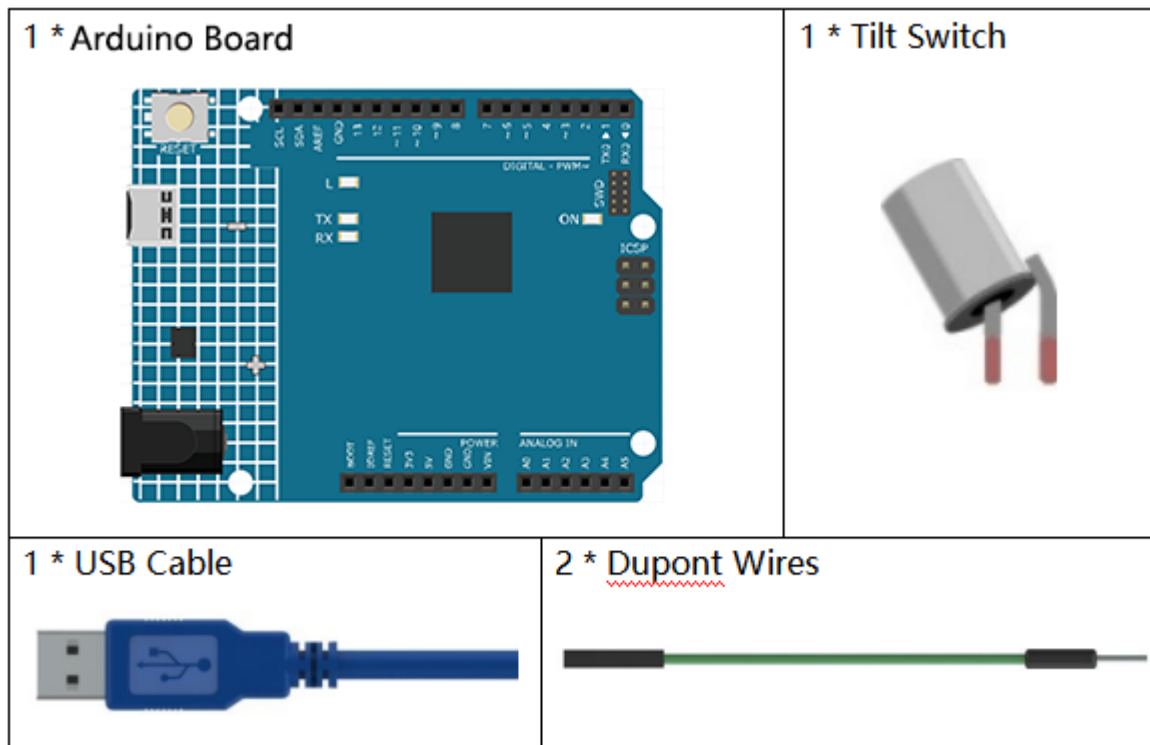
In this part, when the **buttonState** is High level, then let the buzzer beeping in different frequency which can simulate the doorbell.

## 3.5 Lesson 5 Tilt Switch

### 3.5.1 Introduction

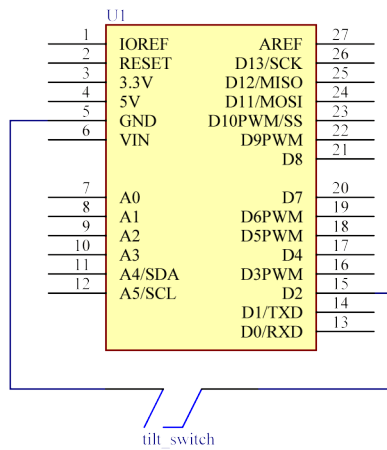
The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

### 3.5.2 Components



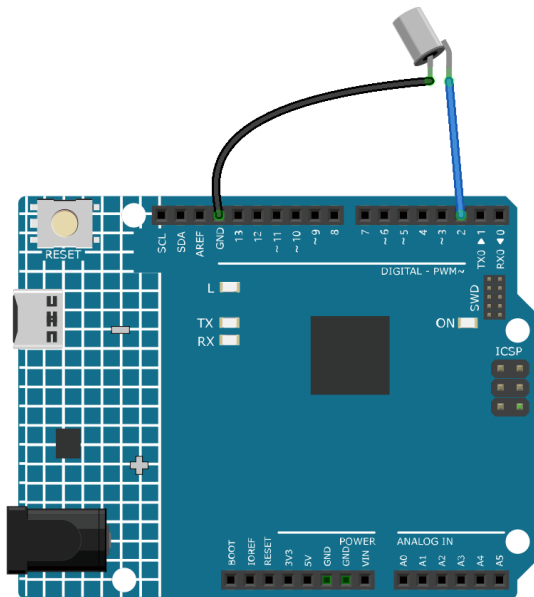
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Tilt Switch*

### 3.5.3 Schematic Diagram



### 3.5.4 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.

### 3.5.5 Code

### 3.5.6 Code Analysis

```
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
    }
}
```

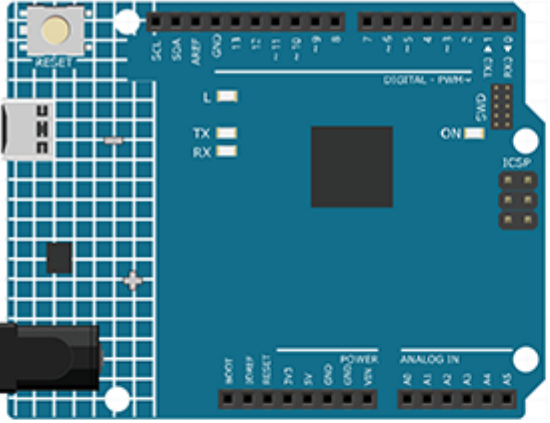






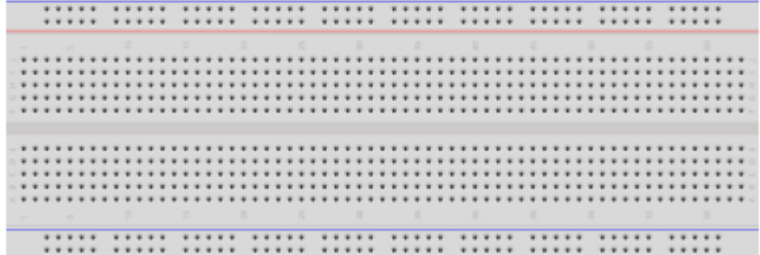


The whole code are very simple, one pin of the tilt switch is connected to pin2, another pin is connected to GND, when tilt the switch, the two pins of the switch will be connected to GND, then let the LED on the pin13 lights up.

## 3.6 Lesson 6 Relay

### 3.6.1 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 micro-controller 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.

### 3.6.2 Components

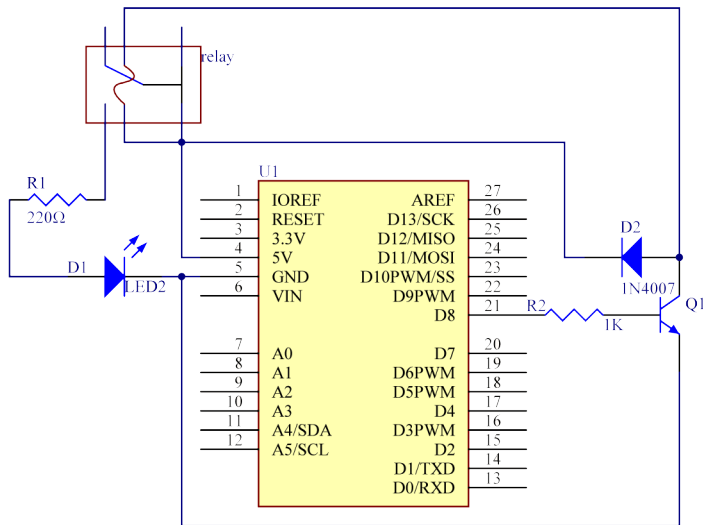
<p>1 * Arduino Board</p> 	<p>1 * Resistor (220Ω)</p> 	<p>1 * LED</p> 
	<p>1*Resistor(1kΩ)</p> 	
<p>1 * NPN Transistor</p> 	<p>1 * Diode 1N4007</p> 	<p>1* Relay</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *Relay*

- *Transistor*
- *Diode*
- *LED*

### 3.6.3 Schematic Diagram

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.



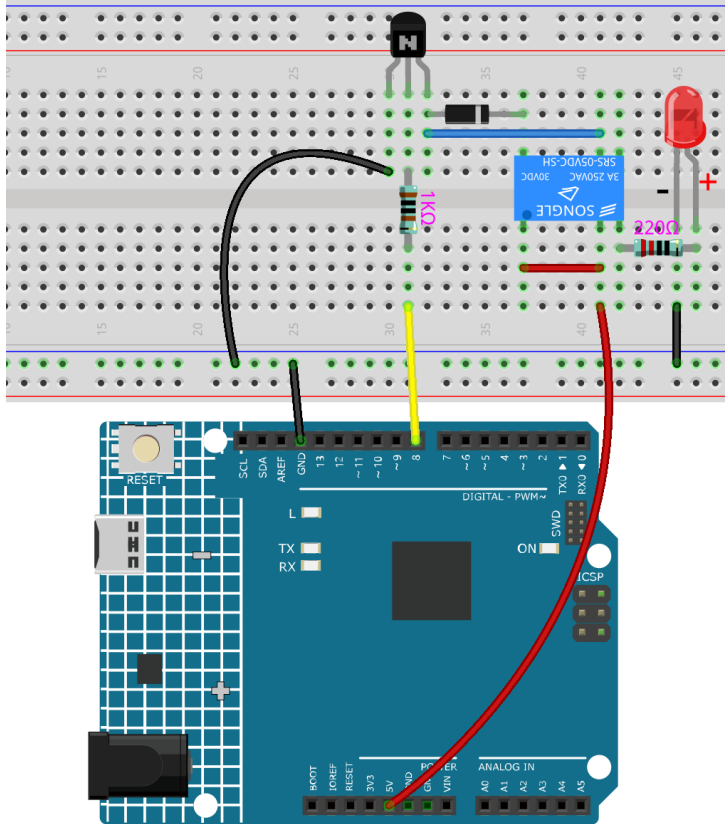
**Function of the freewheeling diode:** When the voltage input changes from High (5V) to Low (0V), the transistor changes from saturation (three working conditions: amplification, saturation, and cut-off) to cut-off, the current in the coil suddenly has no way to flow through. At this moment, without 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.

### 3.6.4 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.

### 3.6.5 Code

### 3.6.6 Code Analysis

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

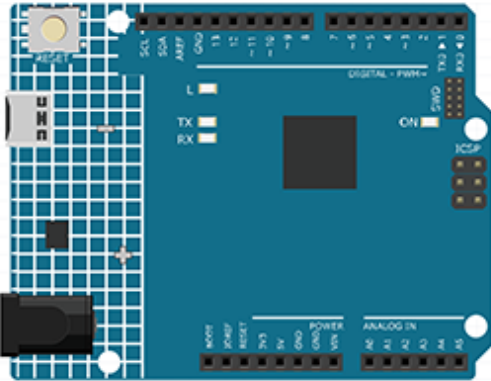


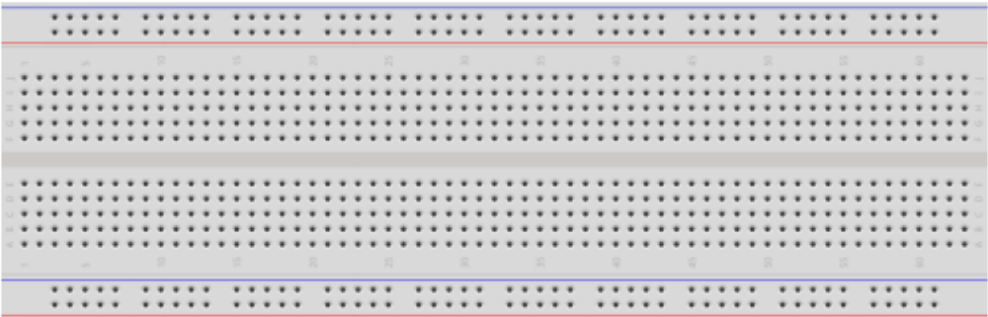


The code in this experiment is simple. First, set relayPin as HIGH level and the LED connected to the relay will light up. Then set relayPin as LOW level and the LED goes out.

## 3.7 Lesson 7 RGB LED

### 3.7.1 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.

### 3.7.2 Components

<b>1 * Arduino Board</b> 	<b>3 * Resistor (220Ω)</b> 	<b>1 * RGB LED</b> 
<b>1 * Breadboard</b> 		
<b>1 * USB Cable</b> 	<b>Several Jump Wires</b> 	

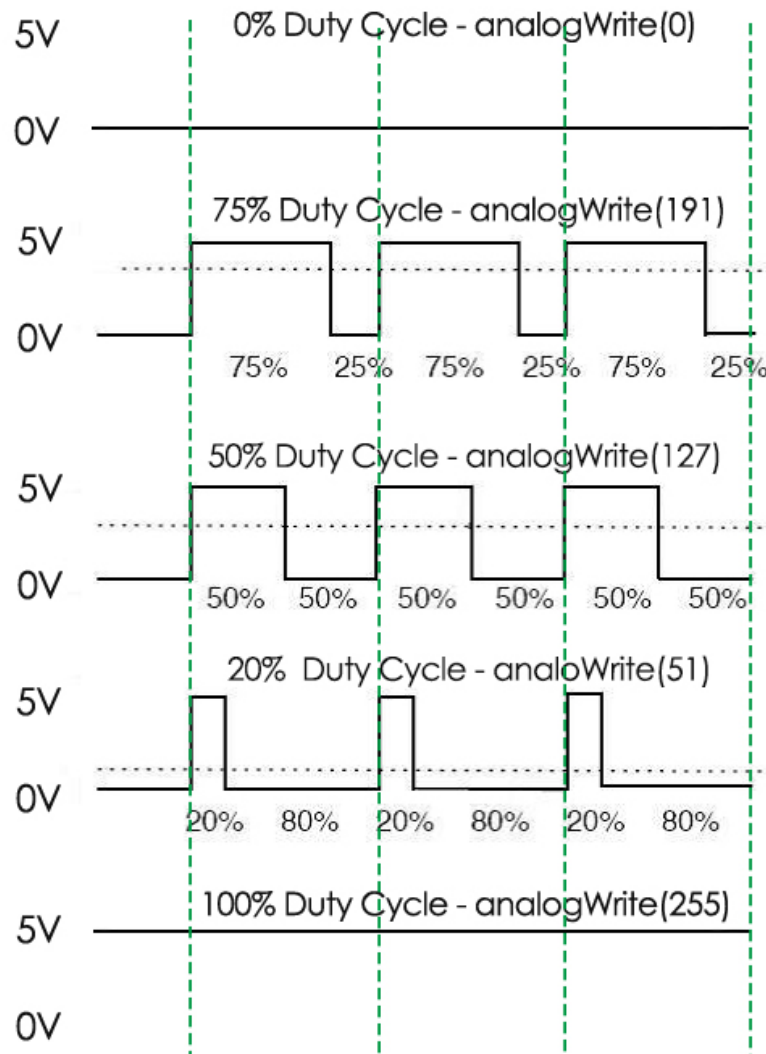
- *Arduino Uno R4 Minima*

- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *RGB LED*

### 3.7.3 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.

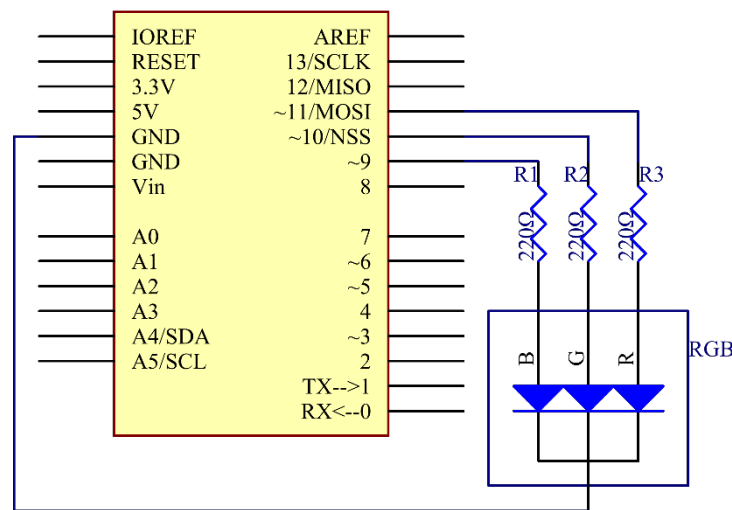


A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.

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.

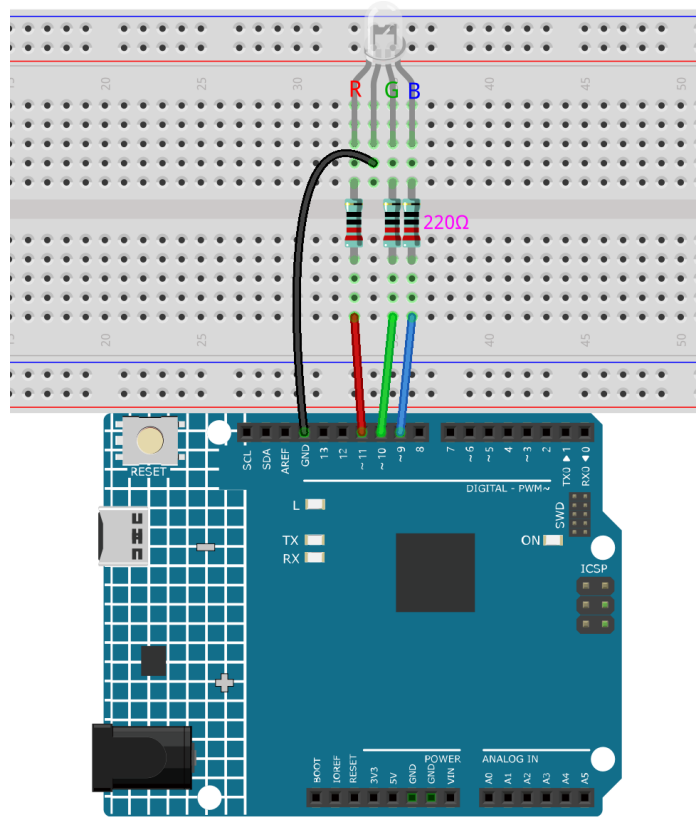
### 3.7.4 Schematic Diagram

On the UNO board, 356 and 9 to 1. 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.



### 3.7.5 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.

### 3.7.6 Code

### 3.7.7 Code Analysis

#### Set the color

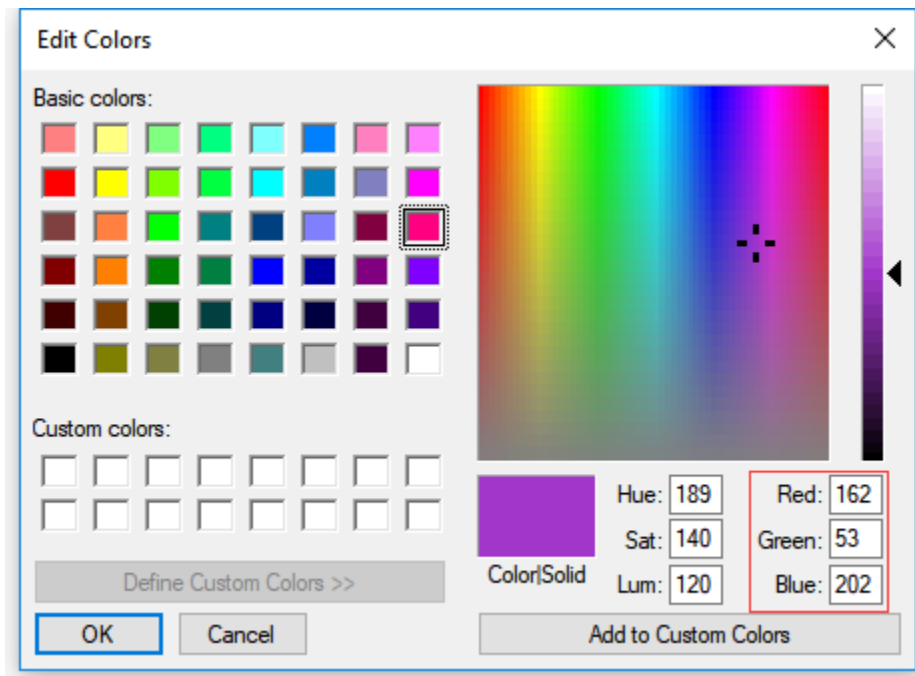
Here use the `color()` function to set the color of the RGB LED. In the code, it is set to flash 7 different colors.

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
    .....
}

```

**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, 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.

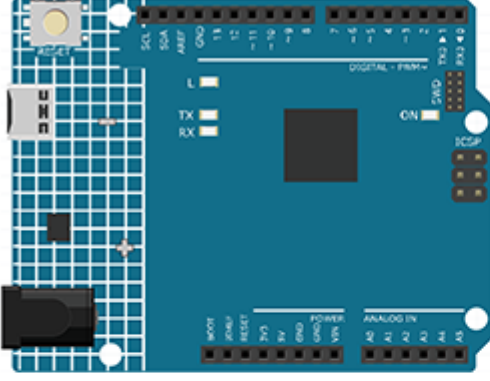





**analogWrite():** Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *pinMode()* to set the pin as output before calling *analogWrite()*.

## 3.8 Lesson 8 Controlling an LED by Potentiometer

### 3.8.1 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.

### 3.8.2 Components

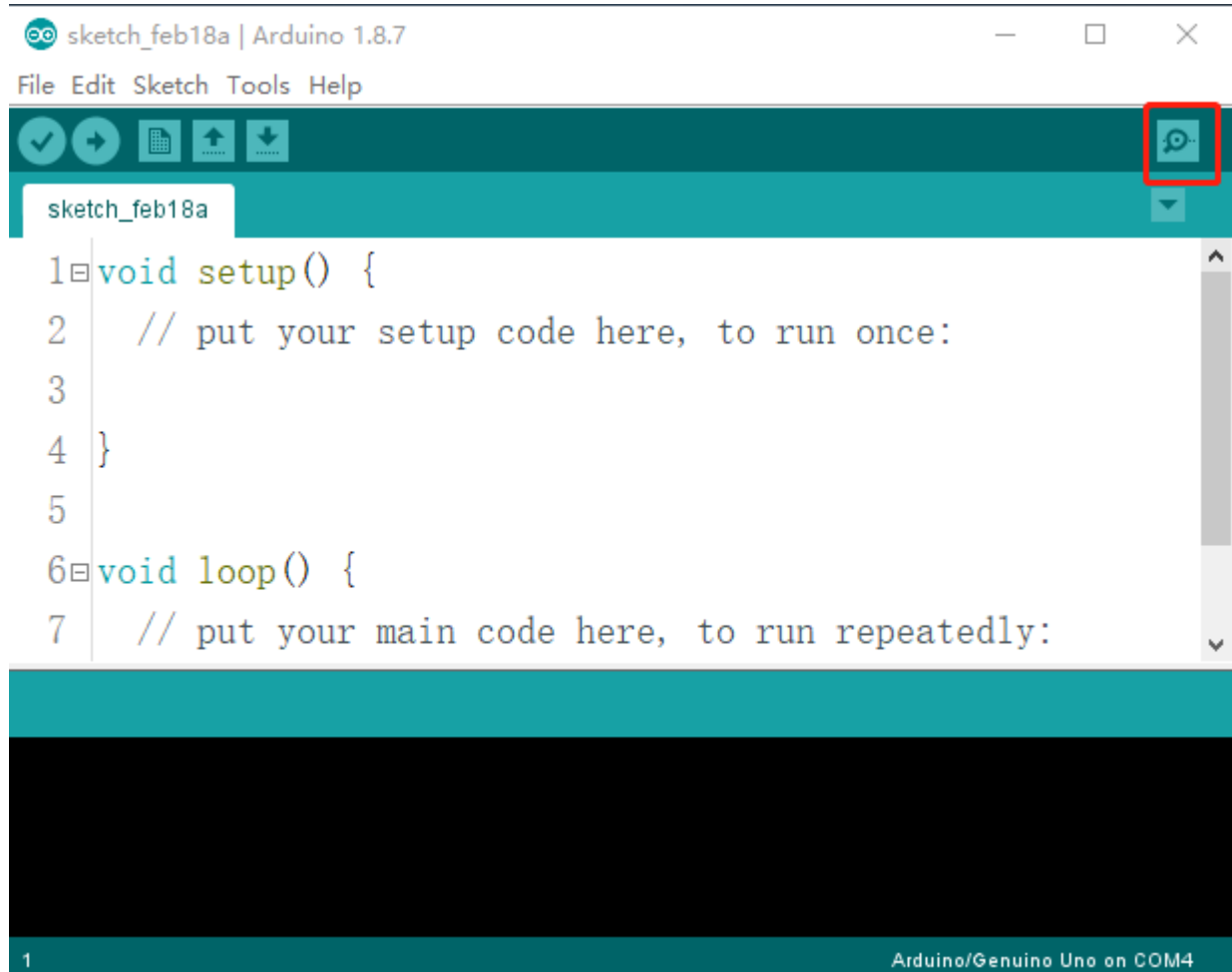
<p>1 * Arduino Board</p> 	<p>1 * Resistor (220Ω)</p> 	<p>1 * LED</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LED*
- *Resistor*
- *Potentiometer*

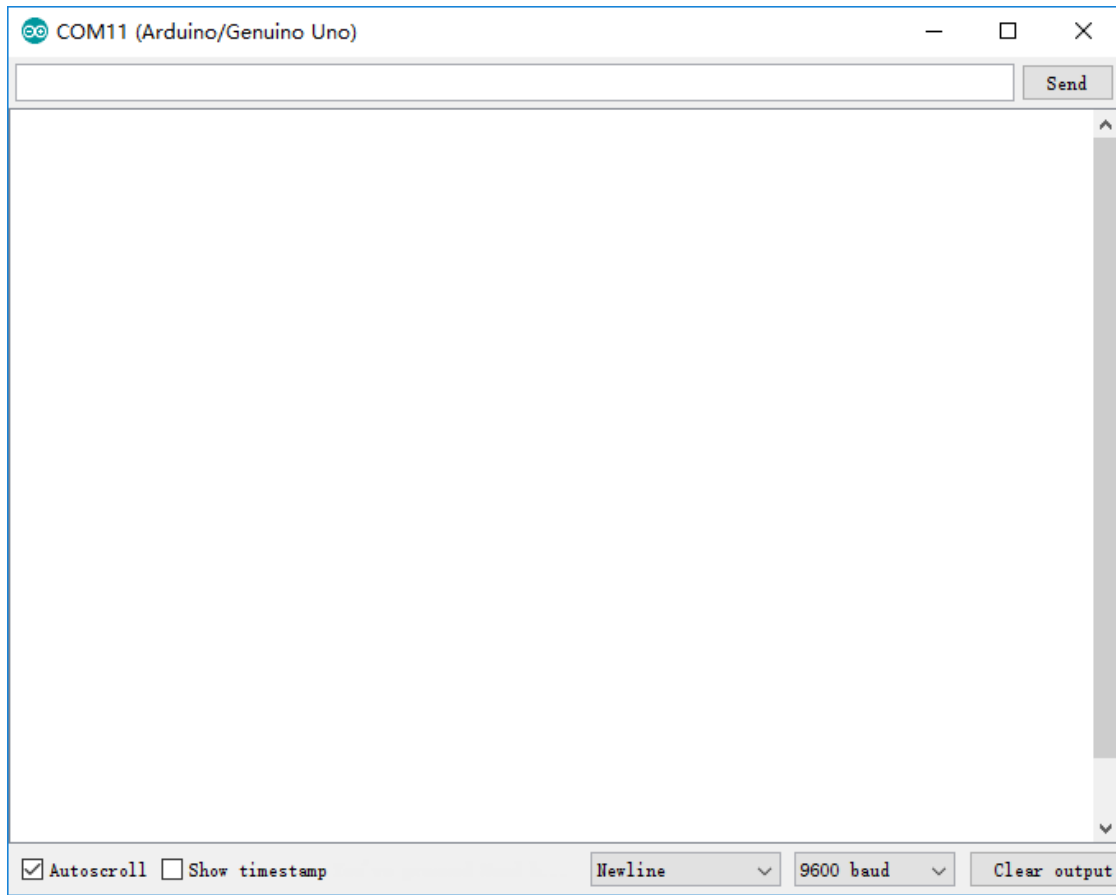


### 3.8.3 Serial Monitor

Serial Monitor is used for communication between the Uno 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.

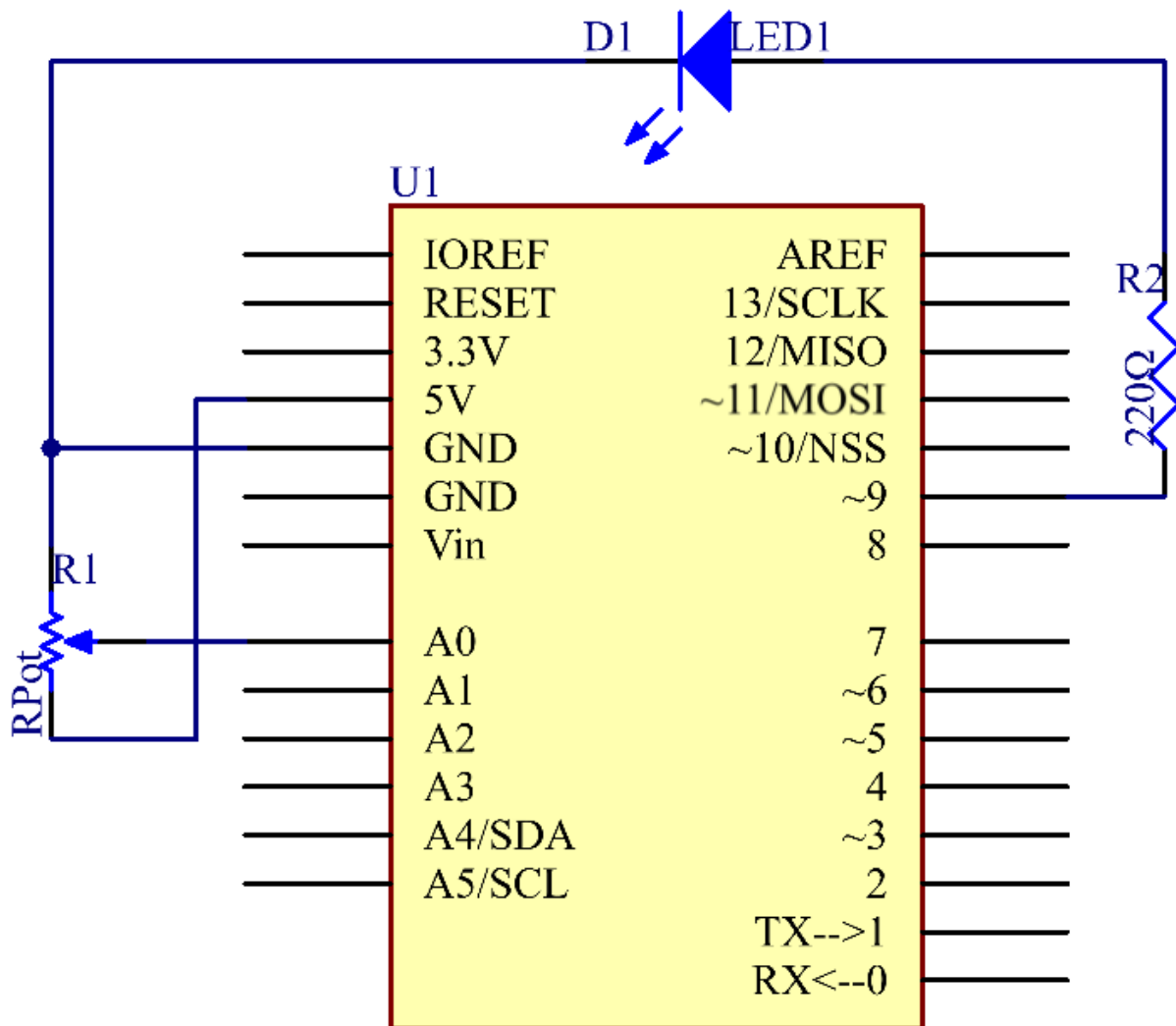


Here, the Serial Monitor serves as a transfer station for communication between your computer and the Uno board. First, the computer transfers data to the Serial Monitor, and then the data is read by the Uno board. Finally, the Uno will perform related operations. Click the icon at the top right corner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



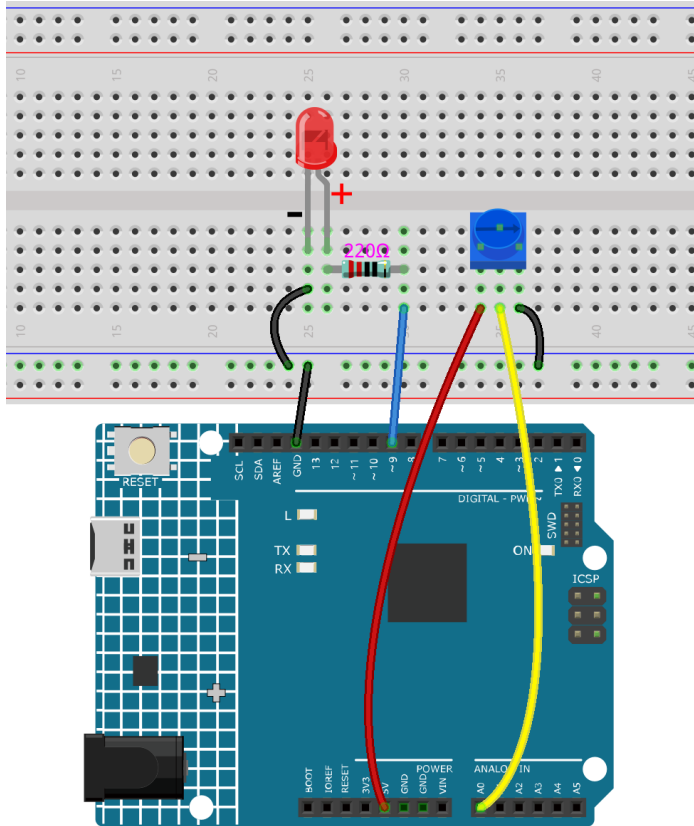
### 3.8.4 Schematic Diagram

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.



### 3.8.5 Experimental Procedures

**Step 1:** Build the circuit.



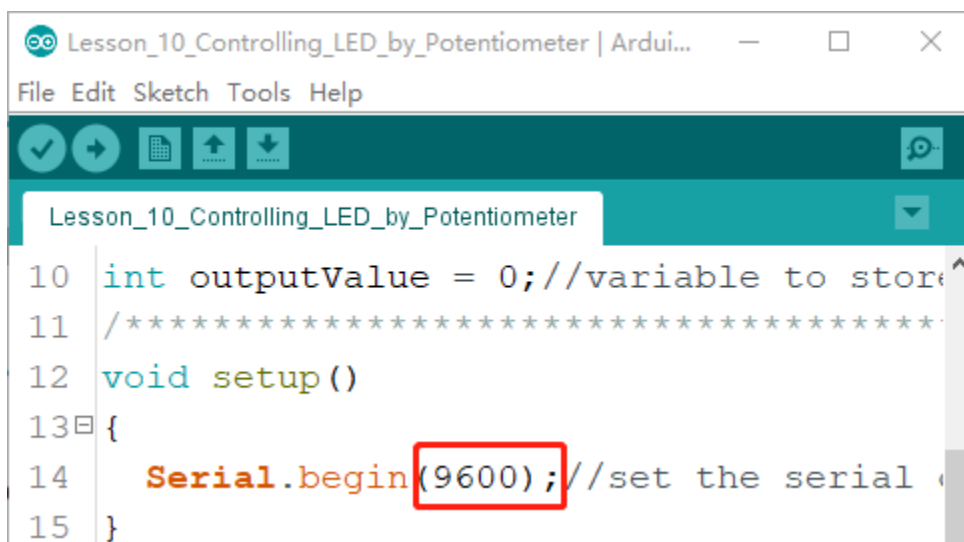
**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

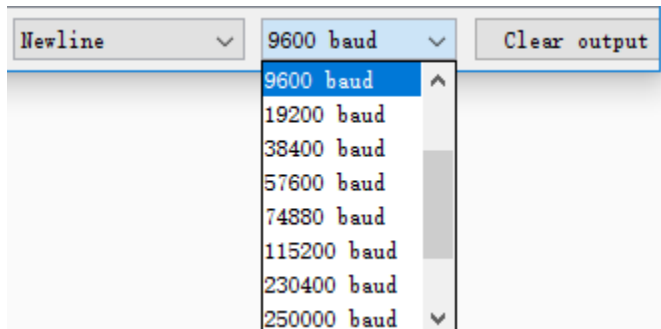
**Step 5:** Open the Serial Monitor.

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.



**Step 6:** 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.

### 3.8.6 Code

#### 3.8.7 Code Analysis

##### 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.

##### 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'`).

##### 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, fromLow, fromHigh, toLow, toHigh)` 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
```

**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()*.

### 3.8.8 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);

digitalWrite(ledPin, HIGH);

delay(inputValue);

digitalWrite(ledPin, LOW);

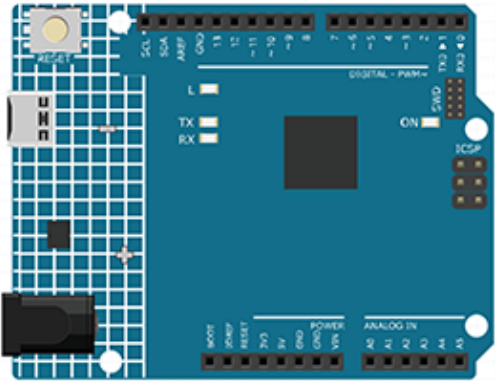
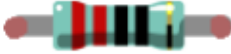

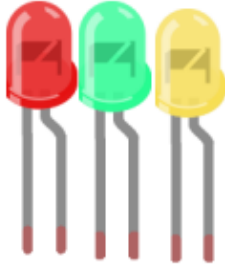

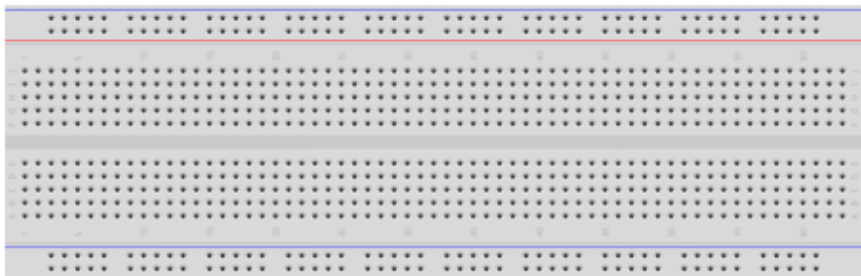


delay(inputValue);
```

## 3.9 Lesson 9 Photoresistor

### 3.9.1 Introduction

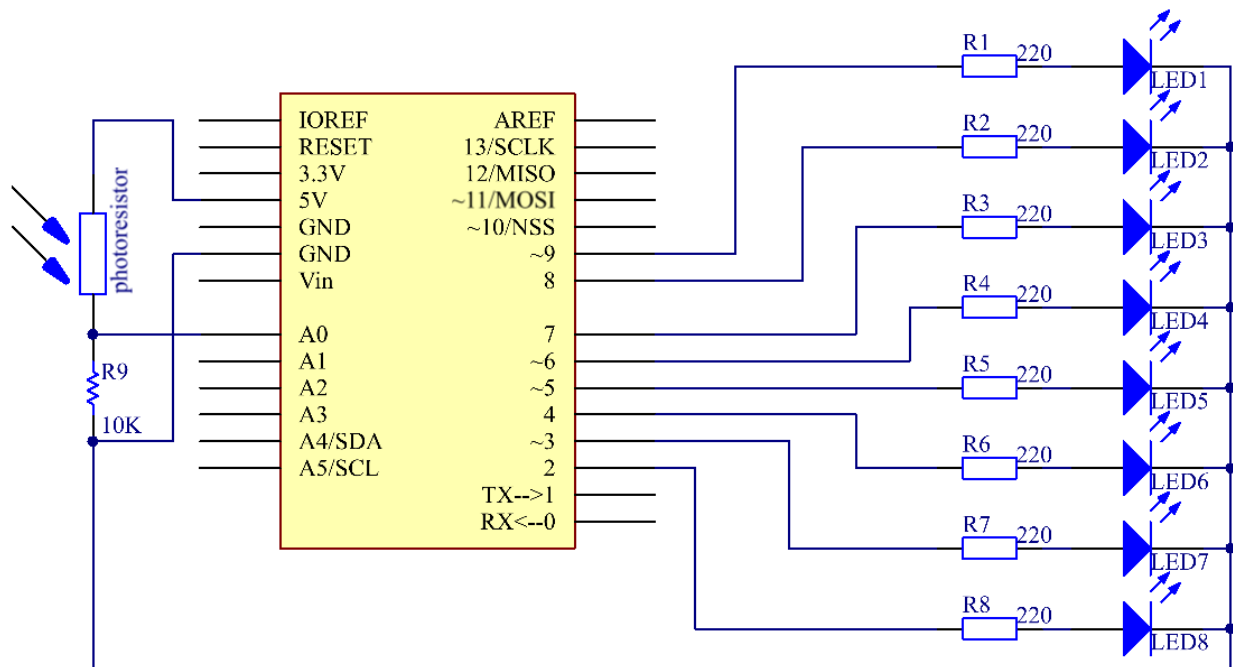
In this lesson, you will learn to how to measure light intensity using a photoresistor. The resistance of a photoresistor changes with incident light intensity. If the light intensity gets higher, the resistance decreases; if it gets lower, the resistance increases.

### 3.9.2 Components

<p>1 * Arduino Board</p> 	<p>8 * Resistor (220Ω)</p>  <p>1 * Resistor (10KΩ)</p> 	<p>8 * LED</p> 
<p>1 * Photo resistor</p> 	<p>1 * Breadboard</p> 	
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LED*
- *Resistor*
- *Photoresistor*

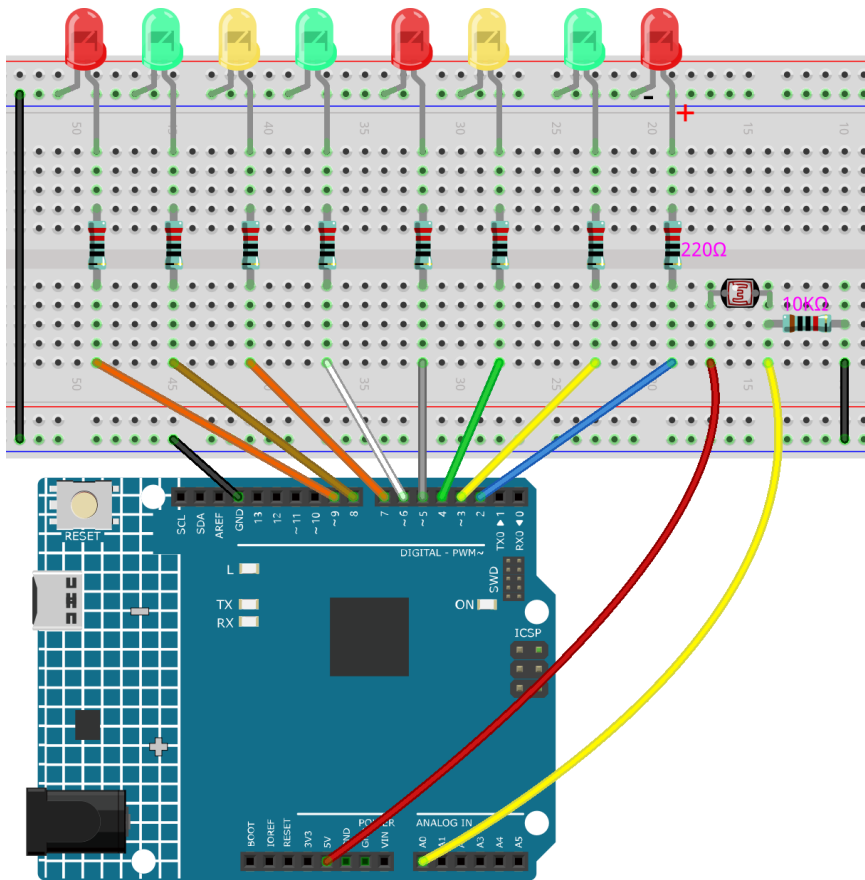
### 3.9.3 Schematic Diagram



### 3.9.4 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 photoresistor, 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.

### 3.9.5 Code

### 3.9.6 Code Analysis

Set the variables

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

const int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 9}; // 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.

### 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.

### 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():** Thiscommand 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').

### 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),  $(1023-0)/(8-0)=127.875$

0-12	128-2	2	56-38	384-	5	12-63	640-7	7	68-89	896	-
7.875	55.75	3.625		511.5	9.375		67.25	5.125		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.

## 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].

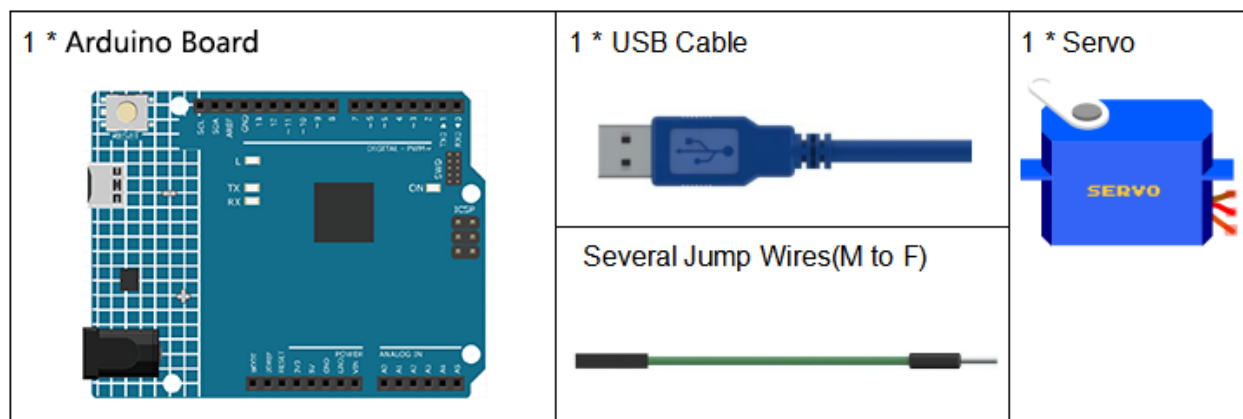
## 3.10 Lesson 10 Servo

### 3.10.1 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.

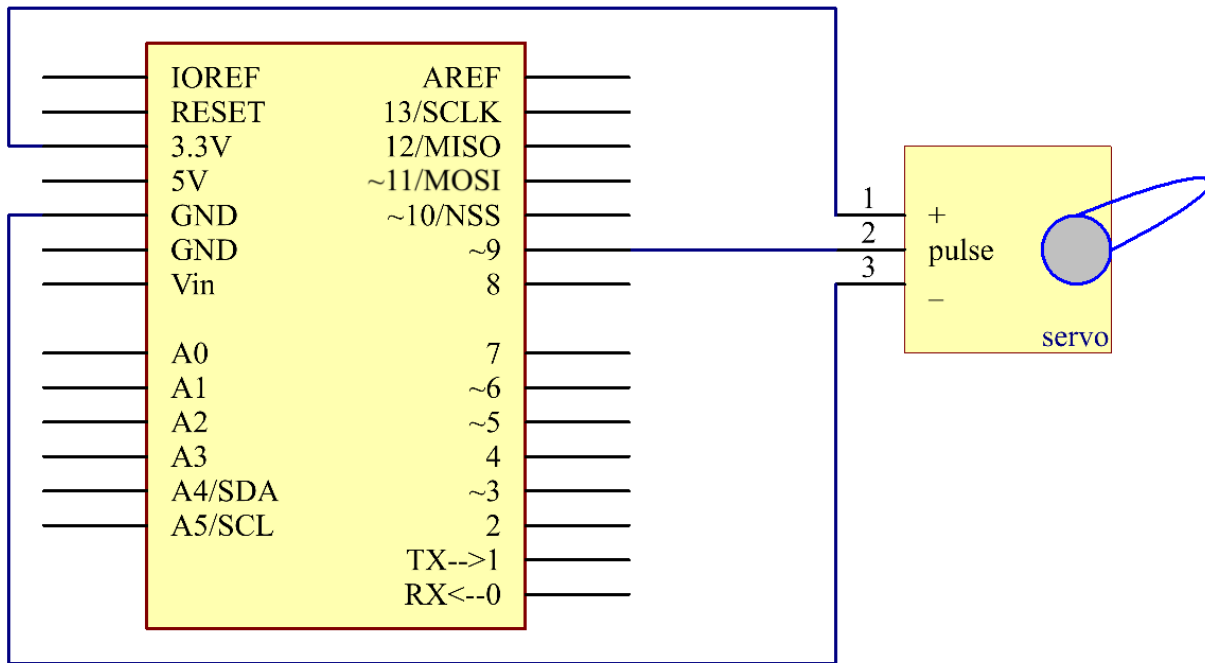
A servo has three wires: the brown wire is GND, the red one is VCC, and the orange one is signal line.

### 3.10.2 Components



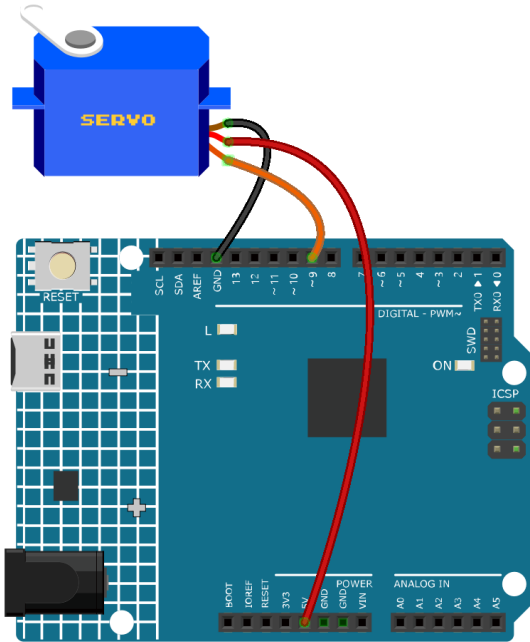
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Servo*

### 3.10.3 Schematic Diagram



### 3.10.4 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.

### 3.10.5 Code

### 3.10.6 Code Analysis

#### 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*.

#### 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
```

(continues on next page)

(continued from previous page)

```
}
```

**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.

### 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
    }
}
```

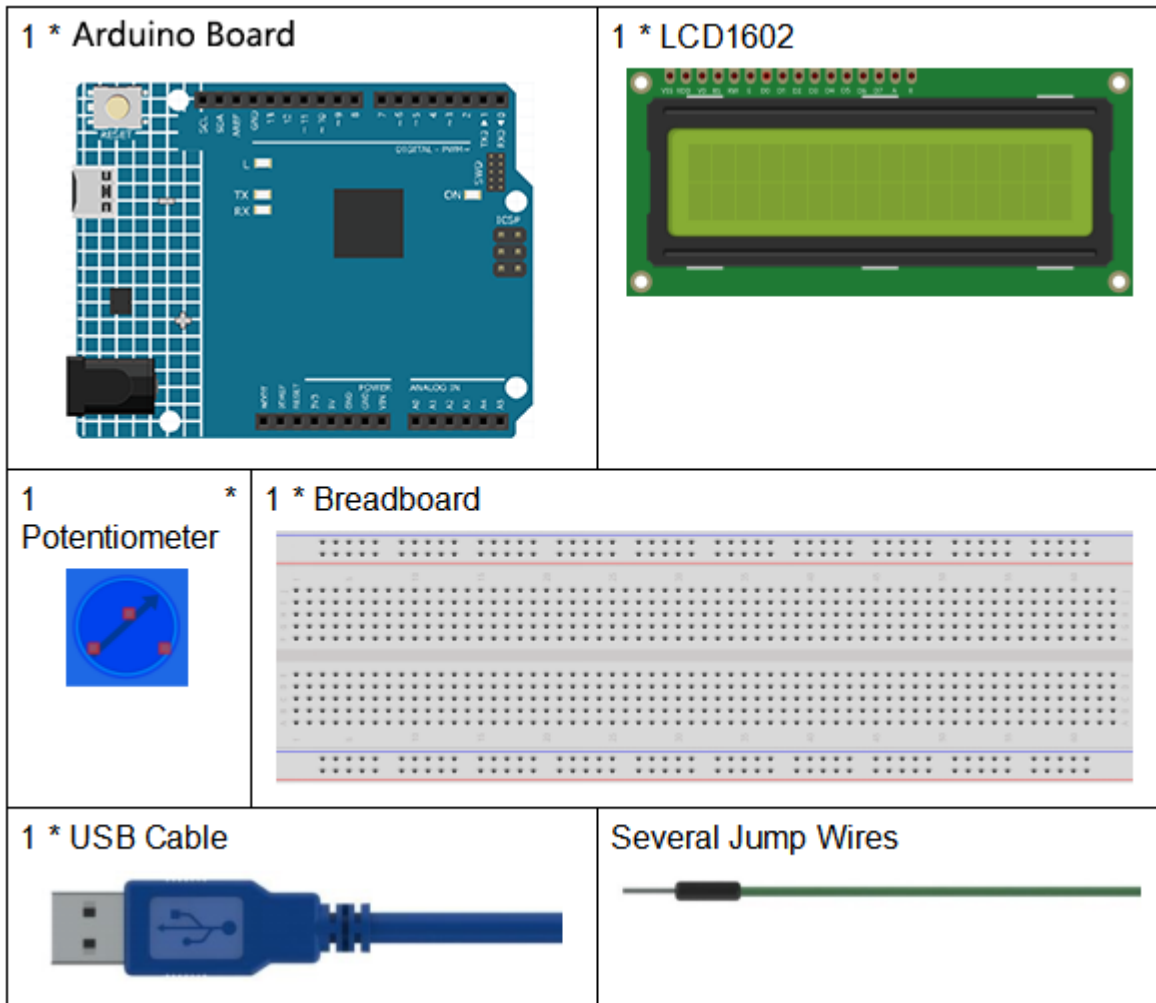
Use 2 for() statement to write 0 - 180 to the servo, so that you can see the servo rotate from 0 to 180 angle, then turn back to 0.

## 3.11 Lesson 11 LCD1602

### 3.11.1 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!

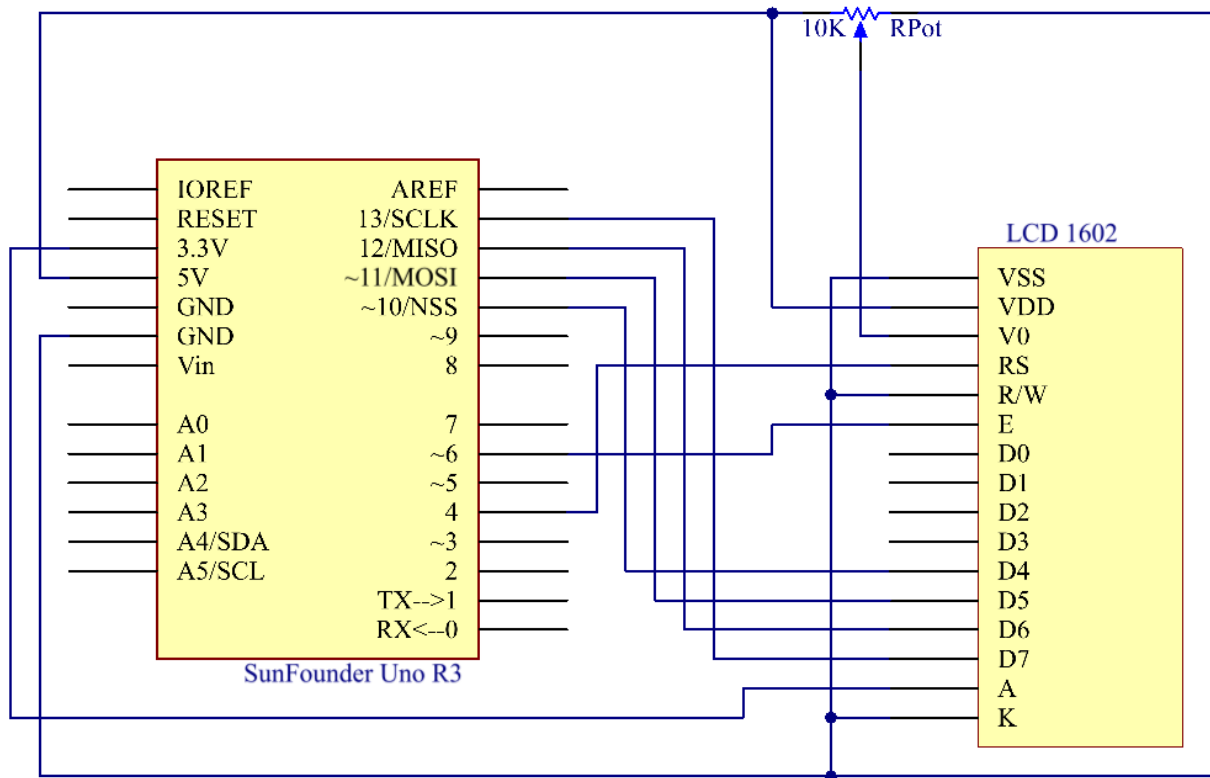
### 3.11.2 Components



- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LCD1602*
- *Potentiometer*

### 3.11.3 Schematic Diagram

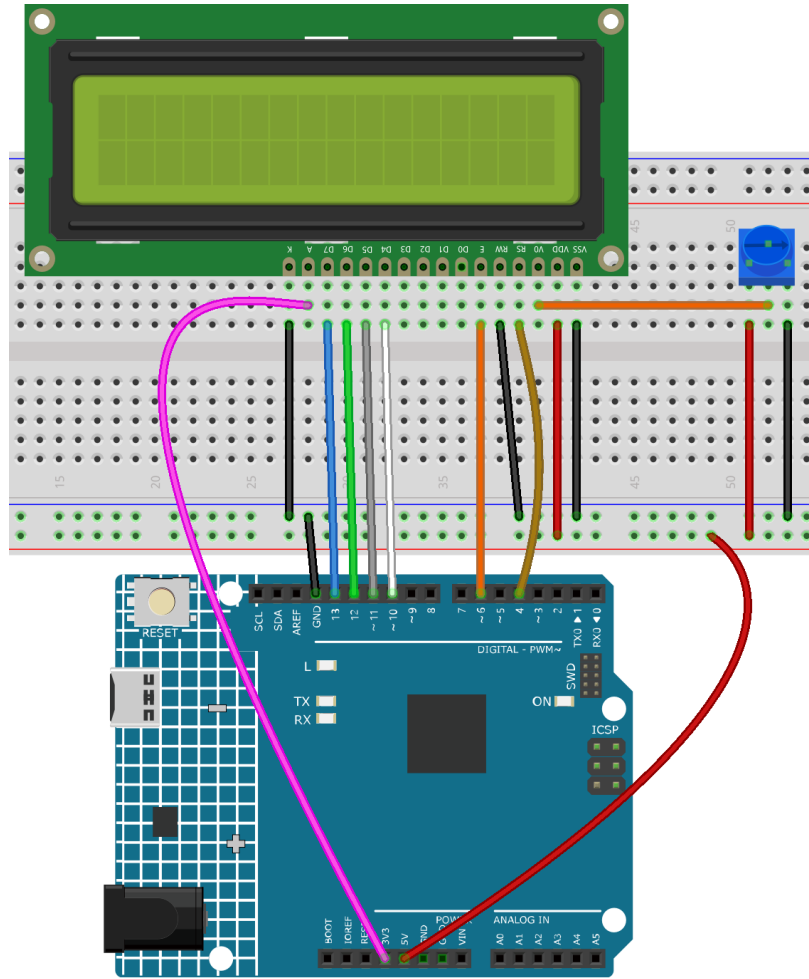
Connect K to GND and A to 3.3 V, and then the backlight of the LCD1602 will be turned on. Connect VSS to GND and the LCD1602 to the power source. Connect VO to the middle pin of the potentiometer - with it you can adjust the contrast of the screen display. Connect RS to D4 and R/W pin to GND, which means 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.



### 3.11.4 Experimental Procedures

**Step 1:** Build the circuit.





**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

---

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

---

You should now see the characters “SunFounder” and “hello, world” rolling on the LCD.

### 3.11.5 Code






### 3.11.6 Code Analysis

**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`.

 examples	2016/8/9 16:32	文件夹	
 src	2016/8/9 16:32	文件夹	
 keywords.txt	2016/7/26 21:16	UltraEdit Docum...	1 KB
 library.properties	2016/7/26 21:16	PROPERTIES 文件	1 KB
 README.adoc	2016/7/26 21:16	ADOC 文件	2 KB

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`.

### 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: `array1[]` 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).

### 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 `LiquidCrystal()` 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.

### 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.
```

### 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.

### 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.
```

(continues on next page)

(continued from previous page)

```

    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
}

```

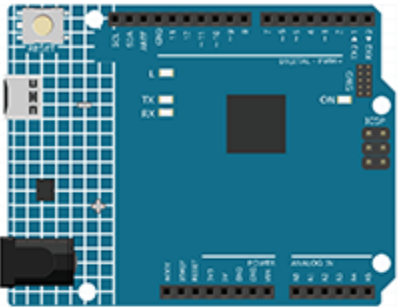


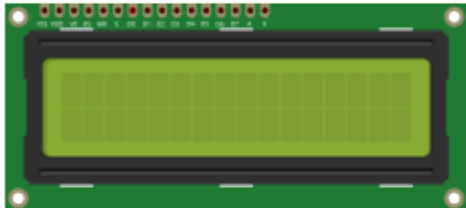

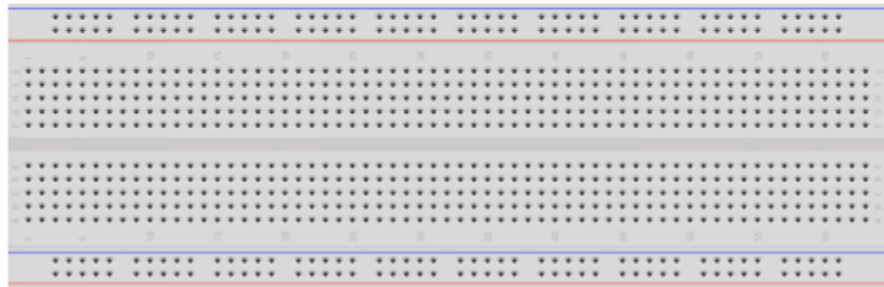


Similarly, the code is to display the elements in `array2[]` on the LCD. Therefore, you will see “SunFounder” scroll in the top line of the LCD, move left until it disappears. And then in the bottom line, “hello, world ! ” appears, scrolls to the left until it disappears.

## 3.12 Lesson 12 Thermistor

### 3.12.1 Introduction

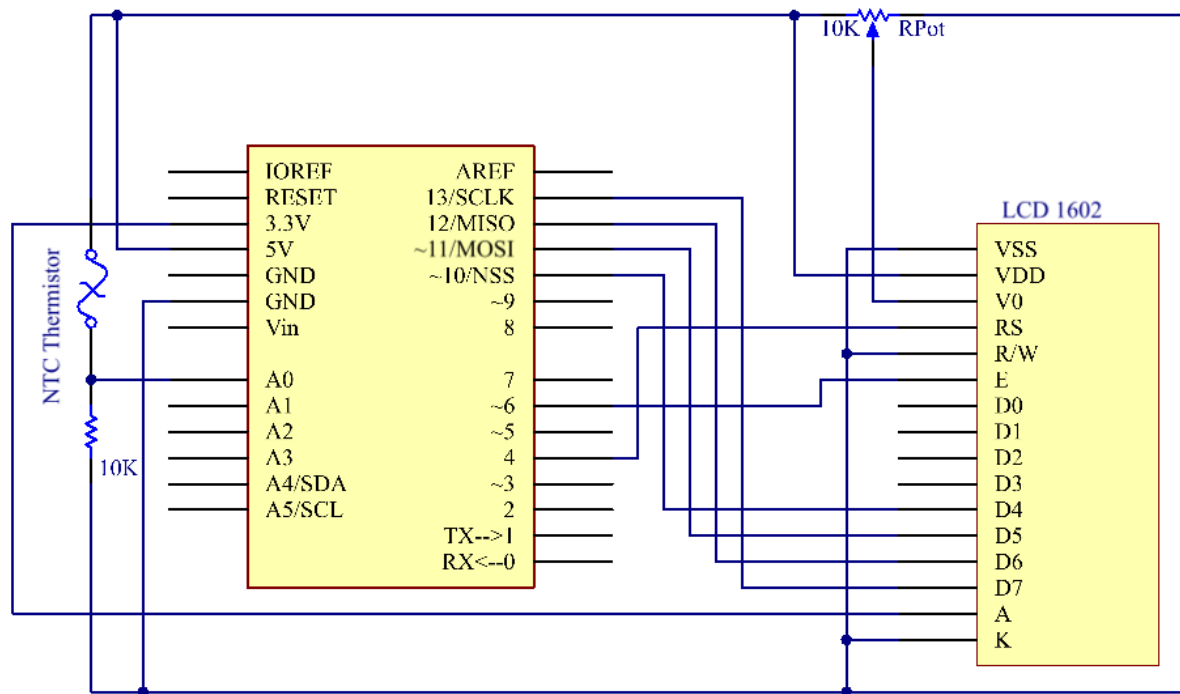
We've learnt many devices so far. To make more things, you need to have a good command of more knowledge. Today we're going to meet a thermistor. It is similar to photoresistor in being able to change their resistance based on the outer change. Different from photoresistor, resistance of thermistor varies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

### 3.12.2 Components

<p>1 * Arduino Board</p> 	<p>1 * <u>Thermistor</u></p> 	<p>1 * Potentiometer</p> 
<p>1 * LCD1602</p> 	<p>1 * Resister (10KΩ)</p> 	
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *Potentiometer*
- *Thermistor*
- *LCD1602*

### 3.12.3 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:

$$RT = RN \exp B(1/T_K - 1/T_N)$$

**RT**: resistance of the NTC thermistor when the temperature is **TK**.

**RN**: resistance of the NTC thermistor under the rated temperature which is  $T_N$ .

**TK** is a Kelvin temperature and the unit is K.

**TN** 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:

$$R = R_0 \exp B[1/(T + 273) - 1/(T_0 + 273)]$$

B, short for beta, is a constant. Here it is 4090. **R<sub>0</sub>** is 10k ohms and **T<sub>0</sub>** 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:

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

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

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

So, this relation can be formed:

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

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

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

$$V = a/205$$

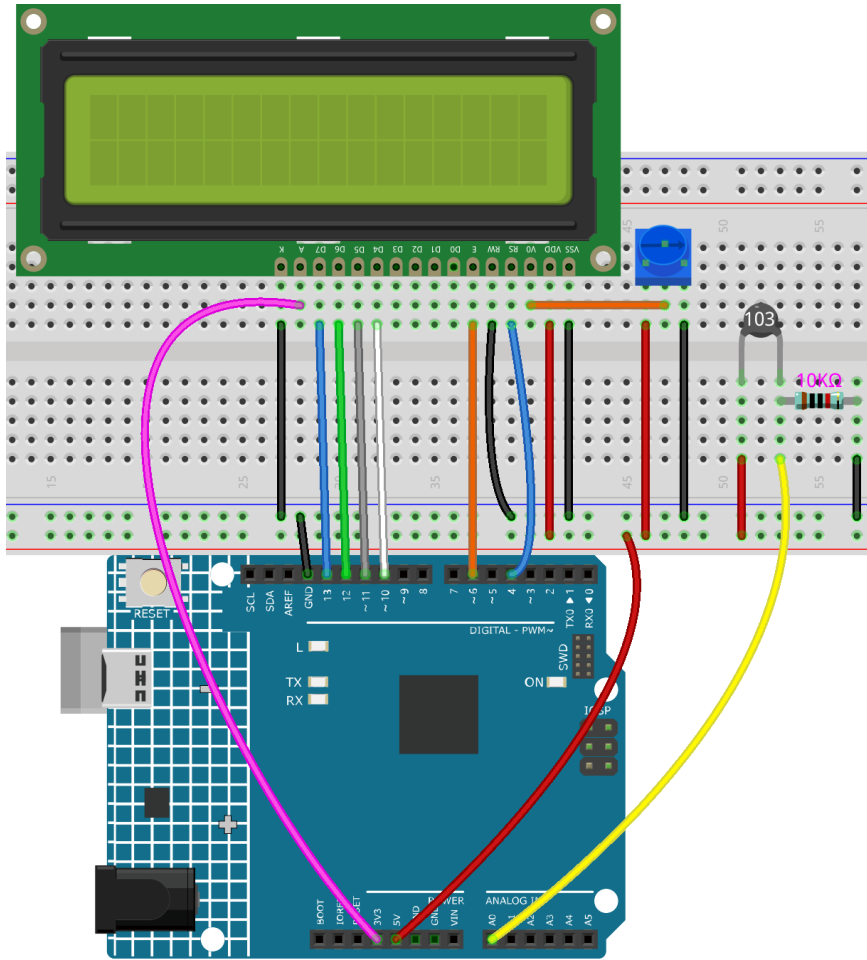
Then replace V in the relation  $R = (5 \times 10k / V) - 10k$  with the expression, and we can get this:  $R = 1025 \times 10k/a - 10k$ .

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

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

### 3.12.4 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.

### 3.12.5 Code

### 3.12.6 Code Analysis

**Set the variables**

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

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

**Get the temperature**

```
void loop()
{
    //read thermistor value
    long a = analogRead(analogPin);
    //the calculating formula of temperature
    float tempC = beta / (log((1025.0 * 10 / a - 10) / 10) + beta / 298.0) - 273.0;
    float tempF = 1.8 * tempC + 32.0;
```

The value of A0 (thermistor) is read, then the Celsius temperature is calculated by the formula, and then the Celsius temperature is converted to Fahrenheit temperature by the formula.

#### 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.
// Print a centigrade temperature to the LCD.
lcd.print(tempC);
// Print the unit of the centigrade temperature to the LCD.
lcd.print(char(223)); // print the unit " °C "
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
}
```

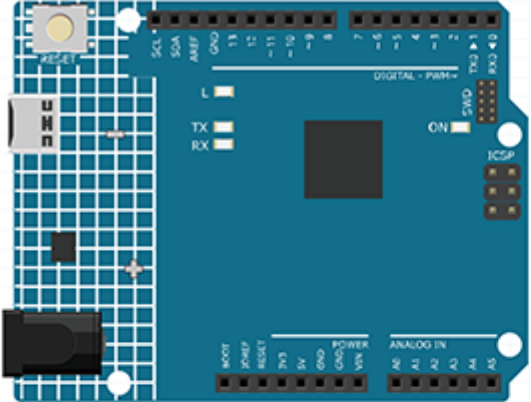


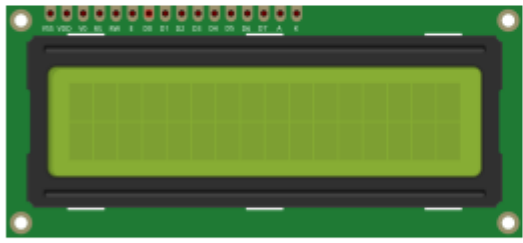
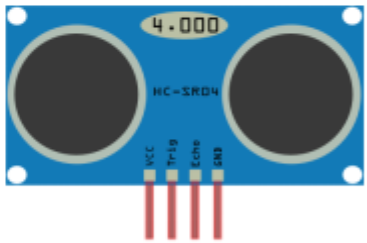


## 3.13 Lesson 13 Ultrasonic

### 3.13.1 Introduction

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

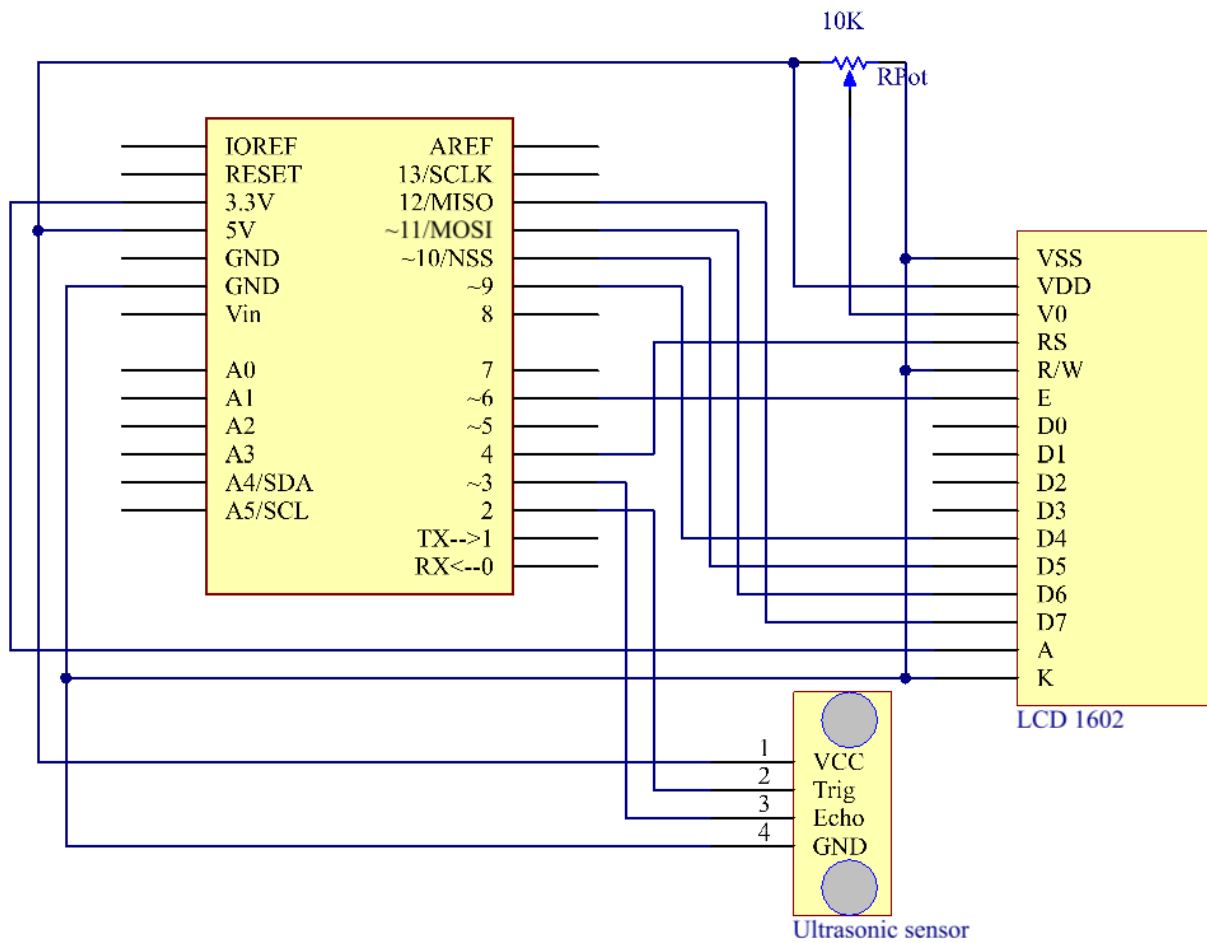


### 3.13.2 Components

<p>1 * Arduino Board</p> 	<p>1 * USB Cable</p>  <p>Several Jump Wires</p> 
<p>1 * LCD1602</p> 	<p>1 * Ultrasonic Module</p> 
<p>1 * Breadboard</p> 	<p>1 * Potentiometer</p> 

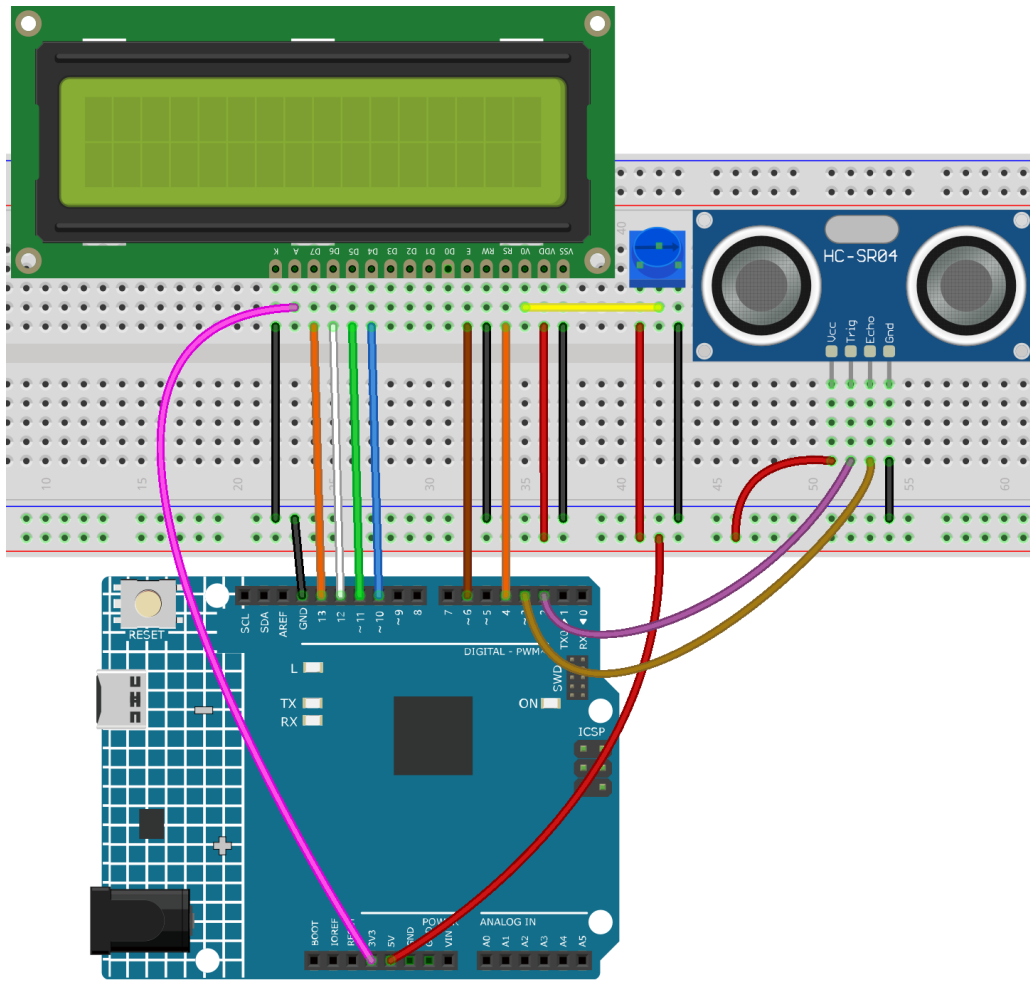
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Potentiometer*
- *Ultrasonic Module*
- *LCD1602*

### 3.13.3 Schematic Diagram



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

### 3.13.5 Code

### 3.13.6 Code Analysis

#### 1. Initialize the ultrasonic sensor and LCD1602

```
#include <LiquidCrystal.h>

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

const int trigPin = 2; // trig pin on the ultrasonic sensor attach to pin2 .
const int echoPin = 3; // echo pin on the ultrasonic sensor attach to pin3.
```

(continues on next page)

(continued from previous page)

```

void setup() {
  lcd.begin(16, 2); // set the position of the characters on the LCD as Line 2, Column 16
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
}

```

## 2. Display the distance on the LCD1602

```

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

```

## 3. Convert the time to distance

```
float readSensorData(){// ...}
```

PING is triggered by a HIGH pulse of 2 or more microseconds. (Give a short LOW pulse beforehand to ensure a clean HIGH pulse.)

```

digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

```

The echo pin is used to read signal from PING, a HIGH pulse whose duration is the time (in microseconds) from the sending of the ping to the reception of echo of the object.

```
microsecond=pulseIn(echoPin, HIGH);
```

The speed of sound is 340 m/s or 29 microseconds per centimeter.

This gives the distance travelled by the ping, outbound and return, so we divide by 2 to get the distance of the obstacle.

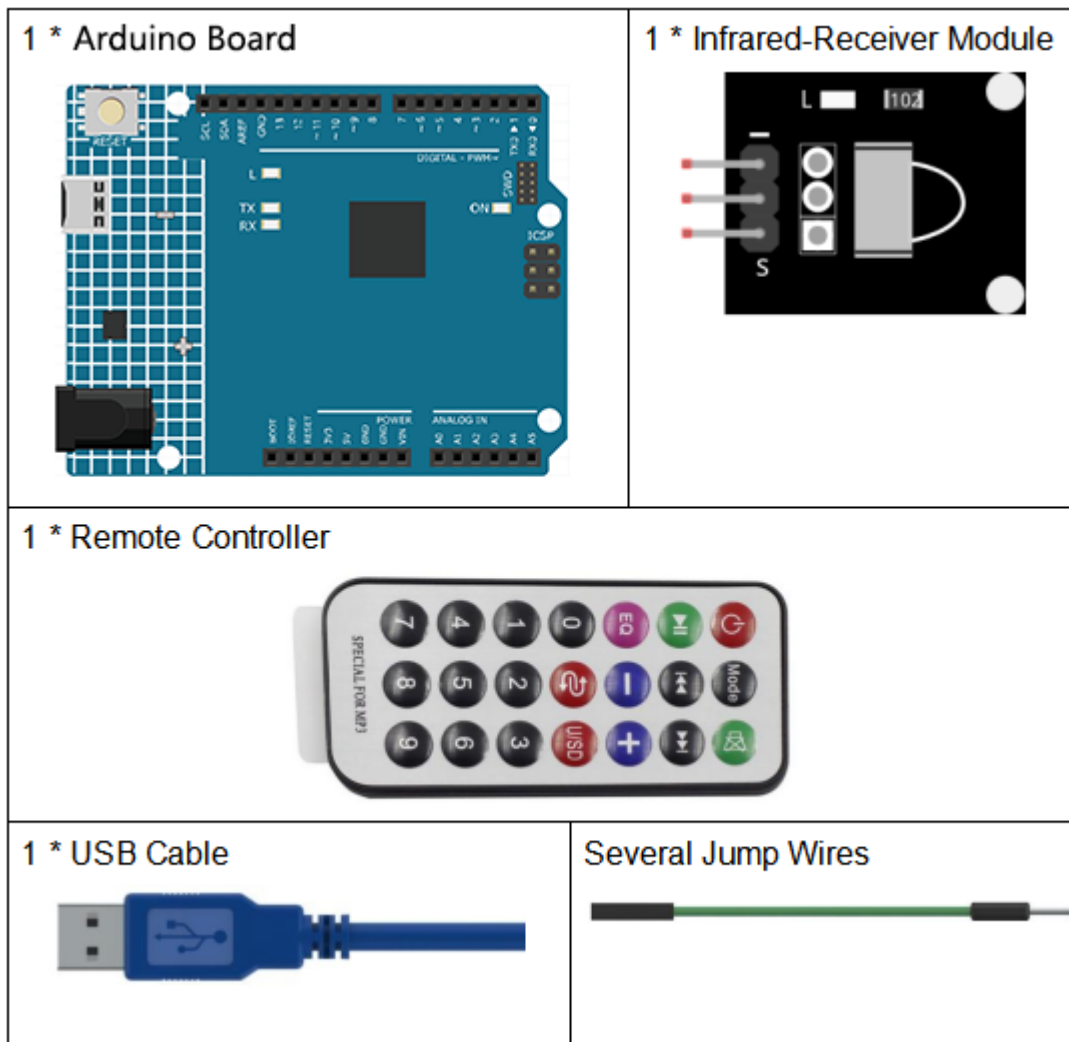
```
float distance = microsecond / 29.00 / 2;
```

## 3.14 Lesson 14 Infrared-Receiver

### 3.14.1 Introduction

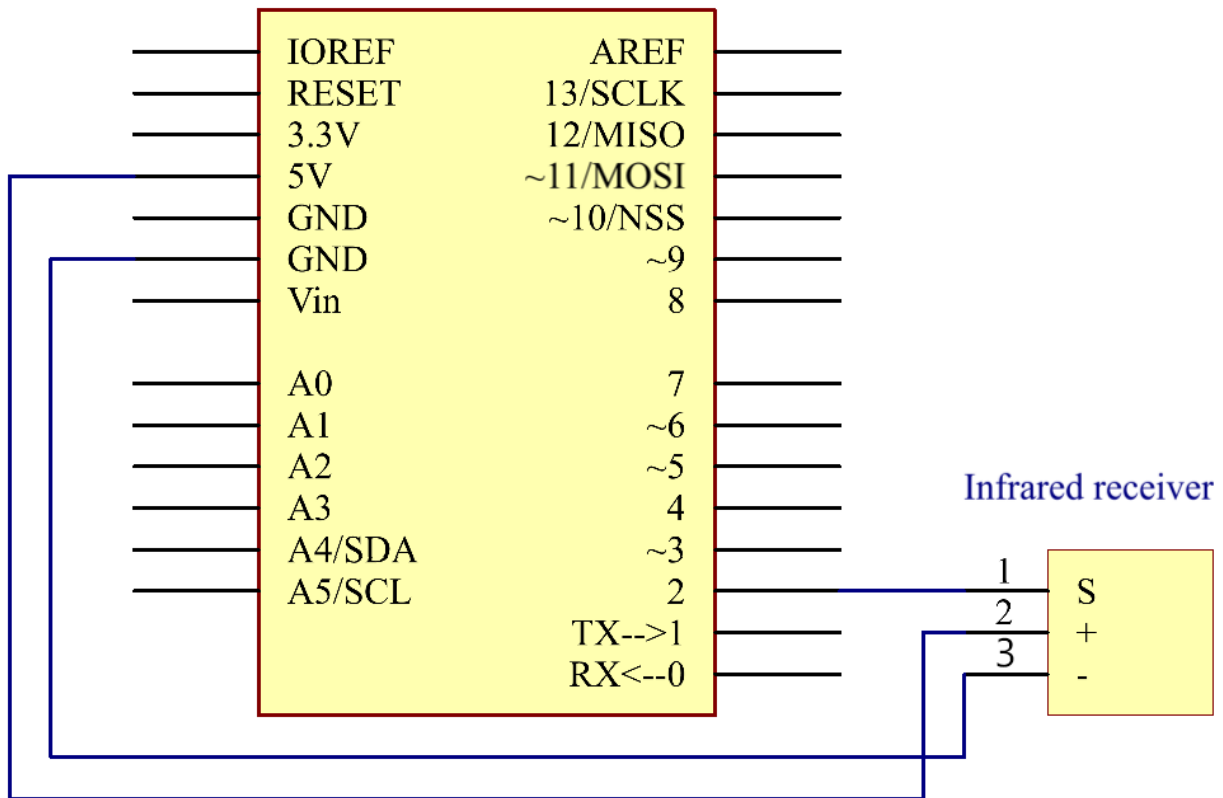
An infrared-receiver is a component that receives infrared signals and can independently receive infrared ray and output signals compatible with TTL level. It's similar with a normal plastic-packaged transistor in size and it is suitable for all kinds of infrared remote control and infrared transmission.

### 3.14.2 Components



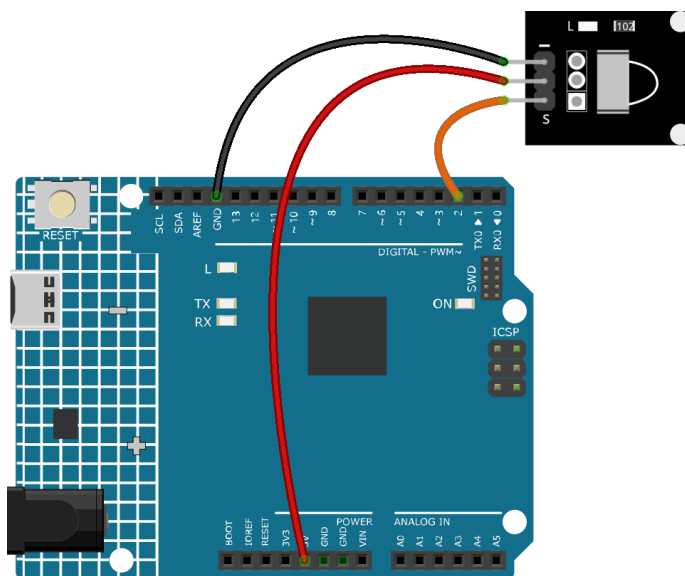
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *IR Receiver Module*

### 3.14.3 Schematic Diagram



### 3.14.4 Experimental Procedures

**Step 1:** Build the circuit.



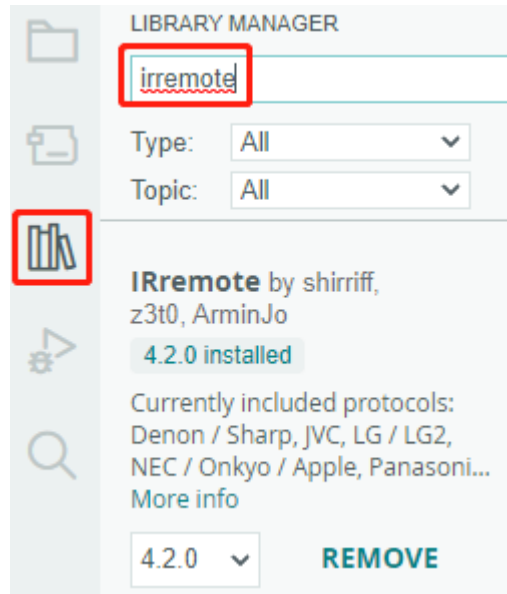
**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

After uploading the codes, you can see that the current value of the pressed button of IR Remote Controller displays on the serial monitor.

- The IRremote library is used here, you can install it from the **Library Manager**.



**Note:**

- There is a transparent plastic piece at the back of the remote control to cut off the power and pull it out before you use the remote control.
- Please gently press the button on the remote to avoid invalid data FFFFFFFF.

### 3.14.5 Code

### 3.14.6 Code Analysis

This code is designed to work with an infrared (IR) remote control using the IRremote library. Here's the breakdown:

1. Include Libraries: This includes the IRremote library, which provides functions to work with IR remote controls.

```
#include <IRremote.h>
```

2. Defines the Arduino pin to which the IR sensor's signal pin is connected and declares a variable to store the last decoded IR value.

```
const int IR_RECEIVE_PIN = 11; // Define the pin number for the IR Sensor
String lastDecodedValue = ""; // Variable to store the last decoded value
```

3. Initializes serial communication at a baud rate of 9600. Initializes the IR receiver on the specified pin (IR\_RECEIVE\_PIN) and enables LED feedback (if applicable).

```
void setup() {  
    Serial.begin(9600); // Start serial  
    ↪communication at 9600 baud rate  
    IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK); // Start the IR  
    ↪receiver  
}
```

4. The loop runs continuously to process incoming IR remote signals.

```
void loop() {  
    if (IrReceiver.decode()) {  
        String decodedValue = decodeKeyValue(IrReceiver.decodedIRData.  
    ↪command);  
        if (decodedValue != "ERROR" && decodedValue != lastDecodedValue) {  
            Serial.println(decodedValue);  
            lastDecodedValue = decodedValue; // Update the last decoded  
    ↪value  
        }  
        IrReceiver.resume(); // Enable receiving of the next value  
    }  
}
```

- Checks if an IR signal is received and successfully decoded.
- Decodes the IR command and stores it in decodedValue using a custom decodeKeyValue() function.
- Checks if the decoded value is not an error and is different from the last decoded value.
- Prints the decoded IR value to the serial monitor.
- Updates the lastDecodedValue with the new decoded value.
- Resumes IR signal reception for the next signal.

## 3.15 Lesson 15 Humiture Sensor

### 3.15.1 Introduction

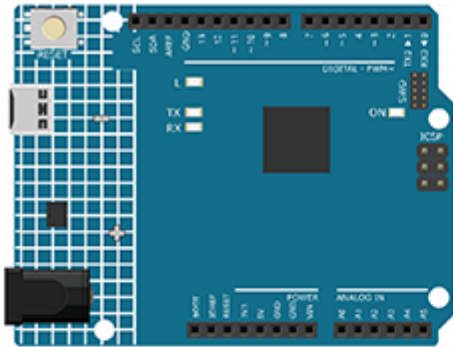
The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature and humidity sensing technology are applied to ensure that the product has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

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

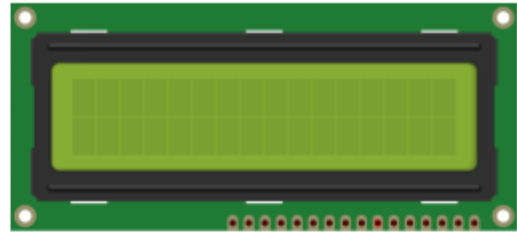


### 3.15.2 Components

1 \* Arduino Board



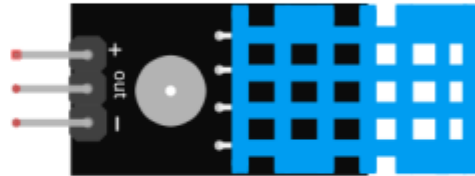
1 \* LCD1602



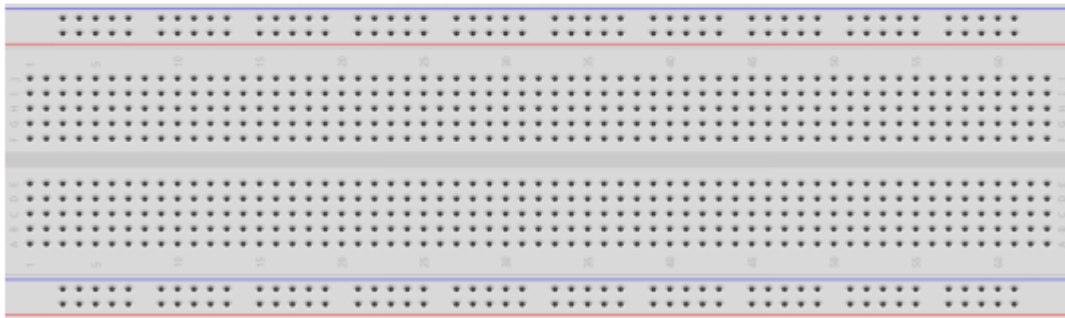
1 \* Potentiometer



1 \* Humiture Sensor Module



1 \* Breadboard



1 \* USB Cable

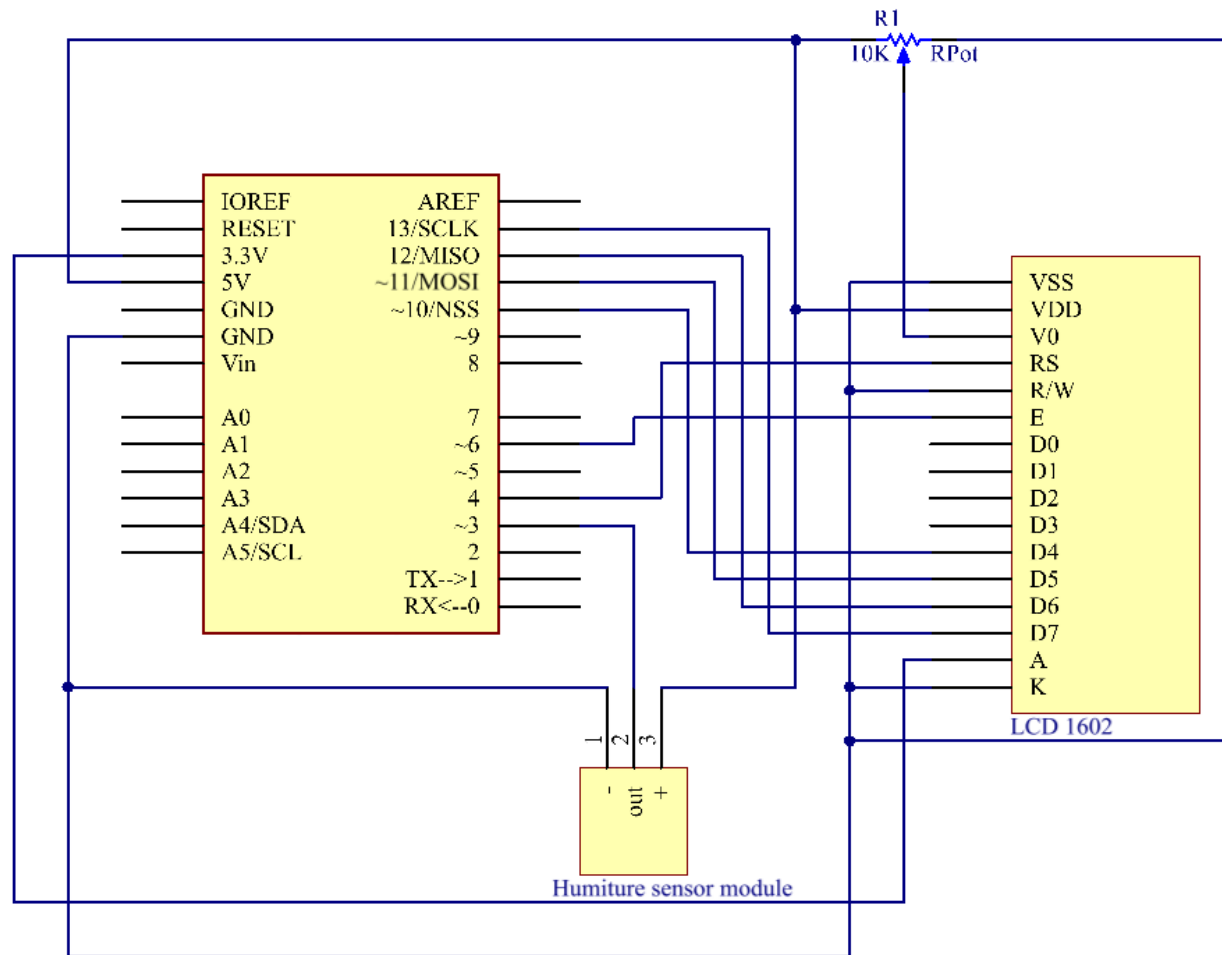


Several Jump Wires



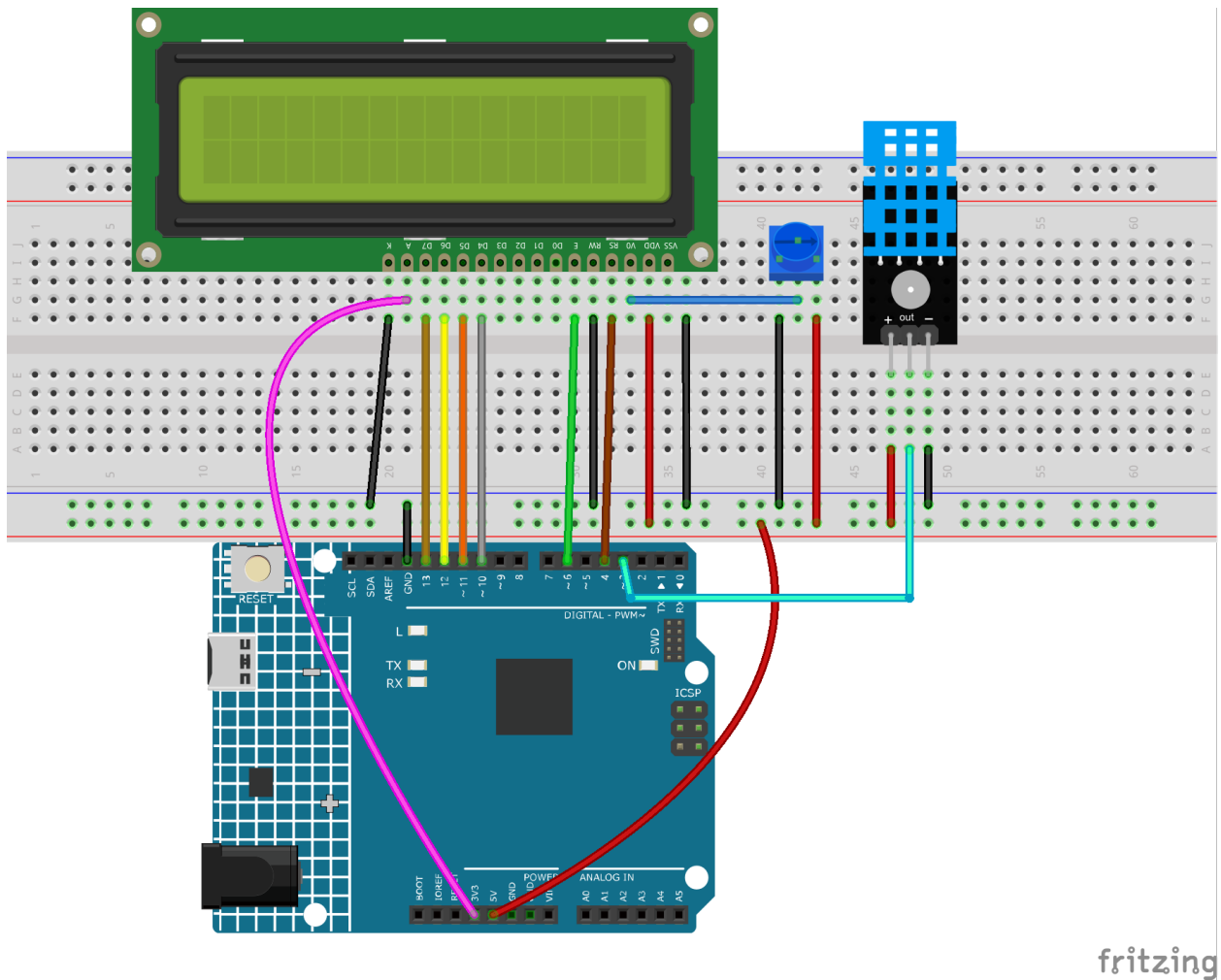
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *LCD1602*
- *Potentiometer*
- *Humiture Sensor Module*

### 3.15.3 Schematic Diagram



### 3.15.4 Experimental Procedures

**Step 1:** Build the circuit.



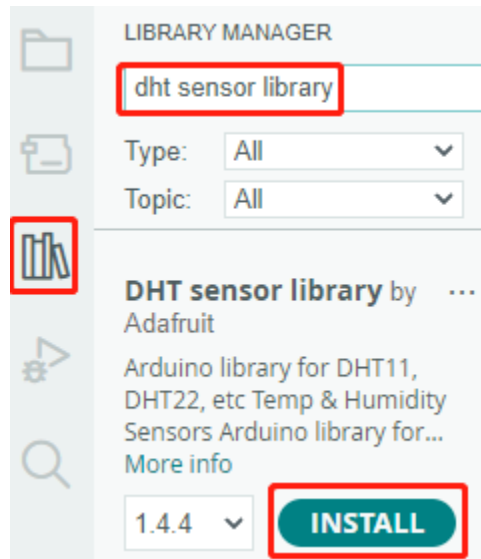
**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

**Note:**

- The DHT sensor library is used here, you can install it from the **Library Manager**.



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

### 3.15.5 Code

### 3.15.6 Code Analysis

1. Includes the DHT.h library, which provides functions to interact with the DHT sensors. Then, set the pin and type for the DHT sensor.

```
#include "DHT.h"
#include <LiquidCrystal.h>           //
LiquidCrystal lcd(4, 6, 10, 11, 12, 13); // initialize the LCD1602

#define DHTPIN 3           // Set the pin connected to the DHT11 data pin
#define DHTTYPE DHT11      // DHT 11

DHT dht(DHTPIN, DHTTYPE);
```

2. Initializes the LCD1602, the serial monitor and the DHT sensor.

```
void setup() {
  lcd.begin(16, 2); // set up the LCD's number of columns and rows:
  Serial.begin(9600); //set the baud bit to 9600bps
  dht.begin();
}
```

3. In the loop() function, read temperature and humidity values from the DHT11 sensor, and print them to the LCD1602.

```
void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
```

(continues on next page)

(continued from previous page)

```

    // Sensor readings may also be up to 2 seconds 'old' (it's a very slow
    ↪ sensor)
    float humidity = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float temperture = dht.readTemperature();

    // Check if any reads failed and exit early (to try again).
    if (isnan(humidity) || isnan(temperture)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }
    // DISPLAY DATA
    lcd.setCursor(0, 0);
    lcd.print("Tem:");
    lcd.print(temperature, 1); //print the temperature on lcd
    lcd.print(" C");
    lcd.setCursor(0, 1);
    lcd.print("Hum:");
    lcd.print(humidity, 1); //print the humidity on lcd
    lcd.print(" %");
    delay(200); //wait a while
}

```

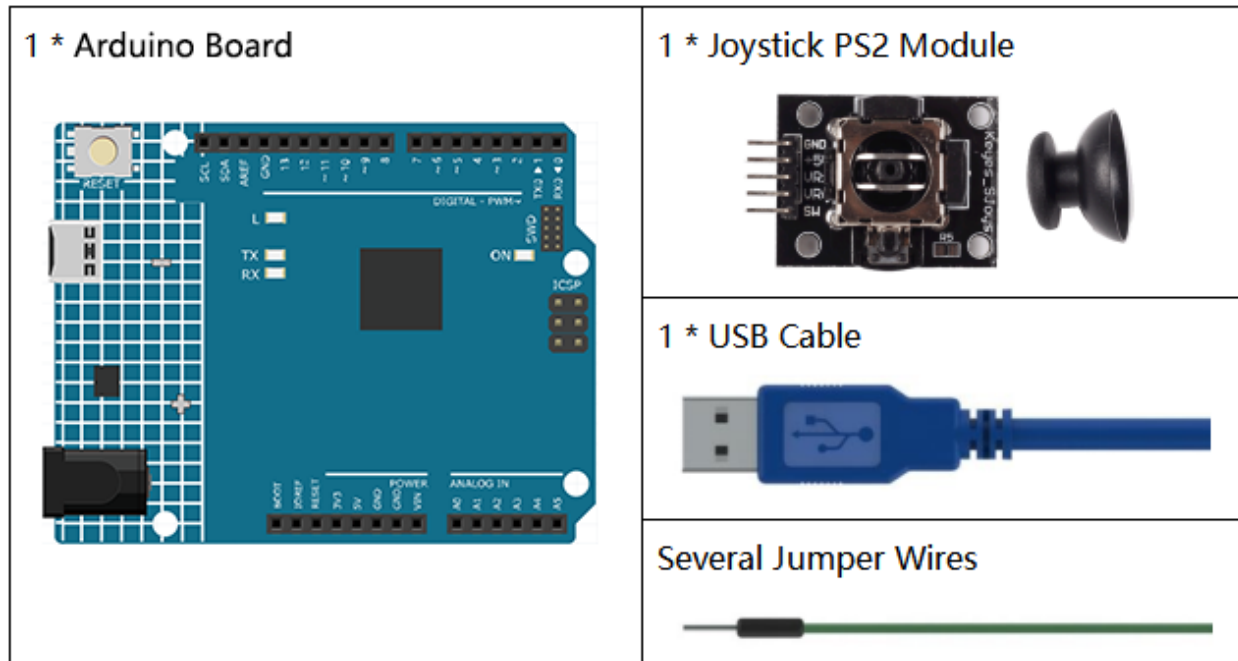
- The `dht.readHumidity()` function is called to read the humidity value from the DHT sensor.
- The `dht.readTemperature()` function is called to read the temperature value from the DHT sensor.
- The `isnan()` function is used to check if the readings are valid. If either the humidity or temperature value is NaN (not a number), it indicates a failed reading from the sensor, and an error message is printed.

## 3.16 Lesson 16 Joystick PS2

### 3.16.1 Introduction

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

### 3.16.2 Components

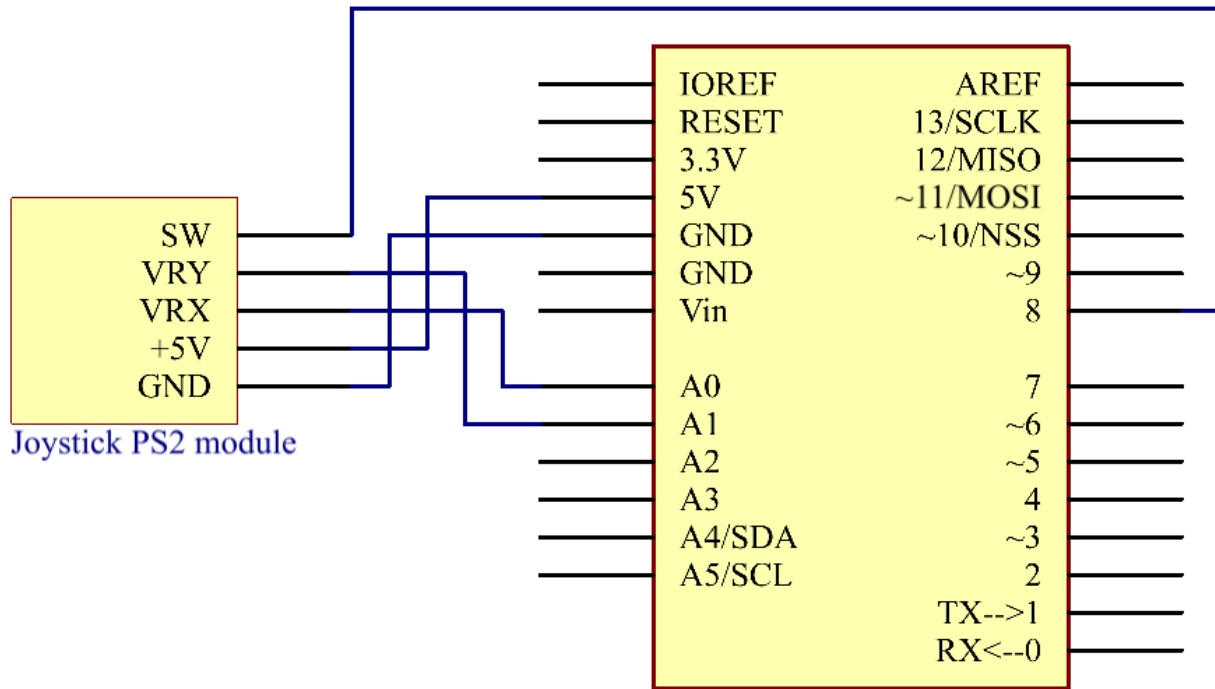


- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Joystick Module*

### 3.16.3 Schematic Diagram

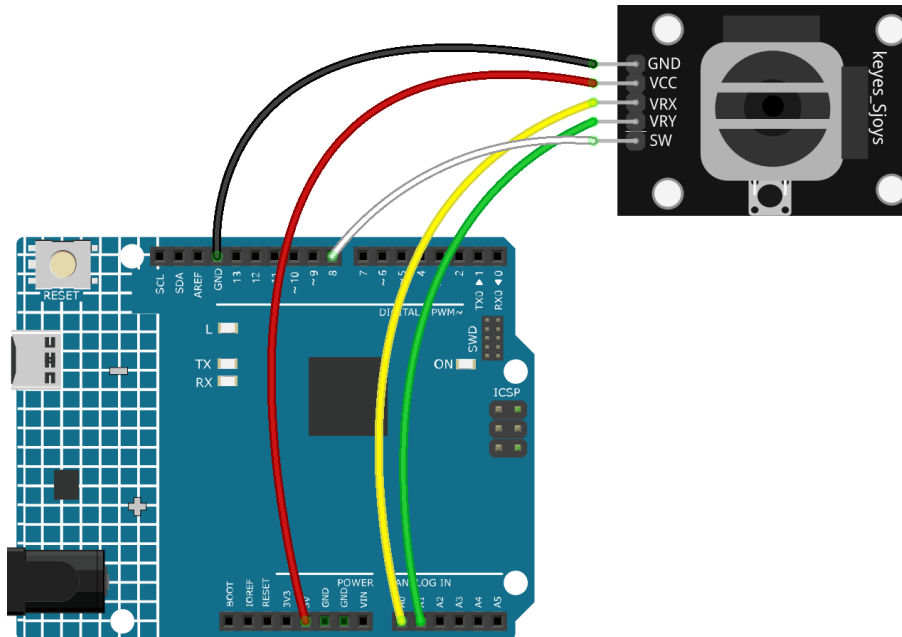
This module has two analog outputs (corresponding to XY biaxial offsets) and one digital output representing whether it is pressed on Z axis. The module integrates power indicator and can display operation condition.

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



### 3.16.4 Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.16.5 Code

#### 3.16.6 Code Analysis

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

```
void loop()
{
    Serial.print("X: ");
    Serial.print(analogRead(xPin), DEC); // print the value of VRX in DEC
    Serial.print(" | Y: ");
    Serial.print(analogRead(yPin), DEC); // print the value of VRX in DEC
    Serial.print(" | Z: ");
    Serial.println(digitalRead(swPin)); // print the value of SW
    delay(50);
}
```

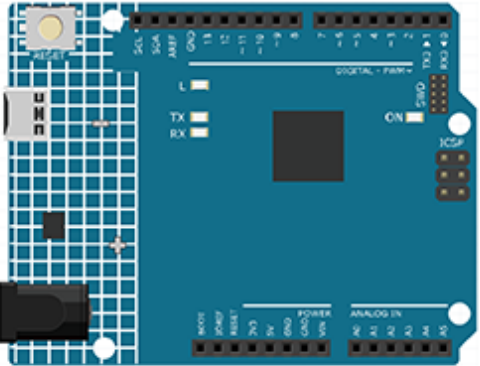


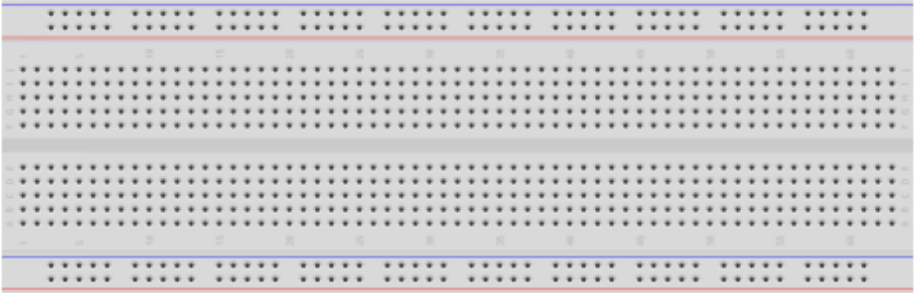


## 3.17 Lesson 17 7-Segment Display

### 3.17.1 Introduction

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



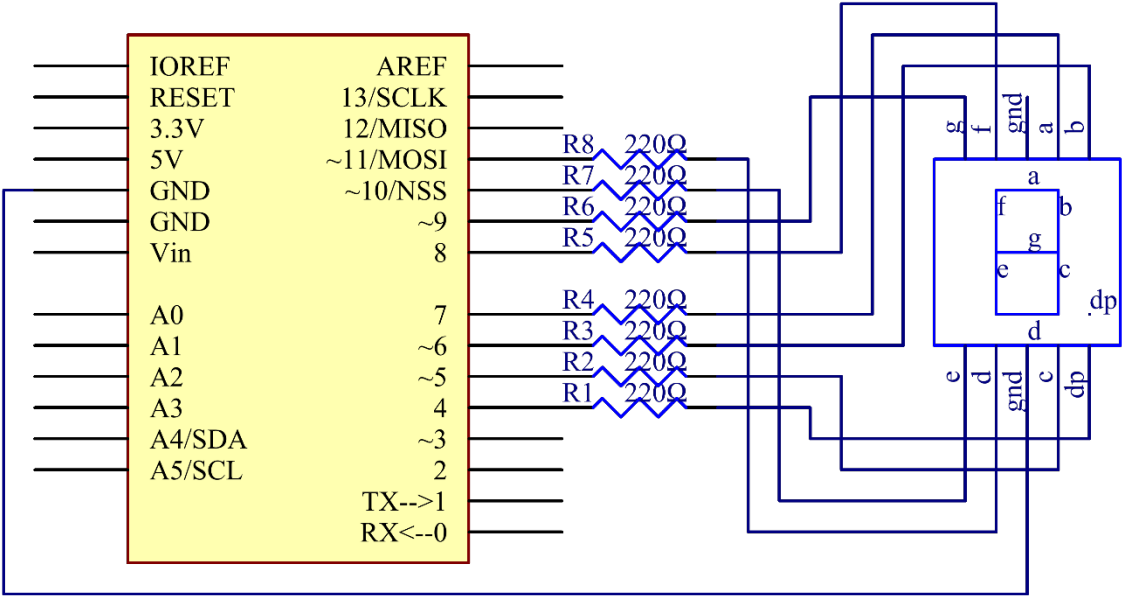
### 3.17.2 Components

<p>1 * Arduino Board</p> 	<p>8 * Resistor (220Ω)</p> 	<p>1 * 7-segment display Common Cathode</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *7-segment Display*

3.17.3 Schematic Diagram

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

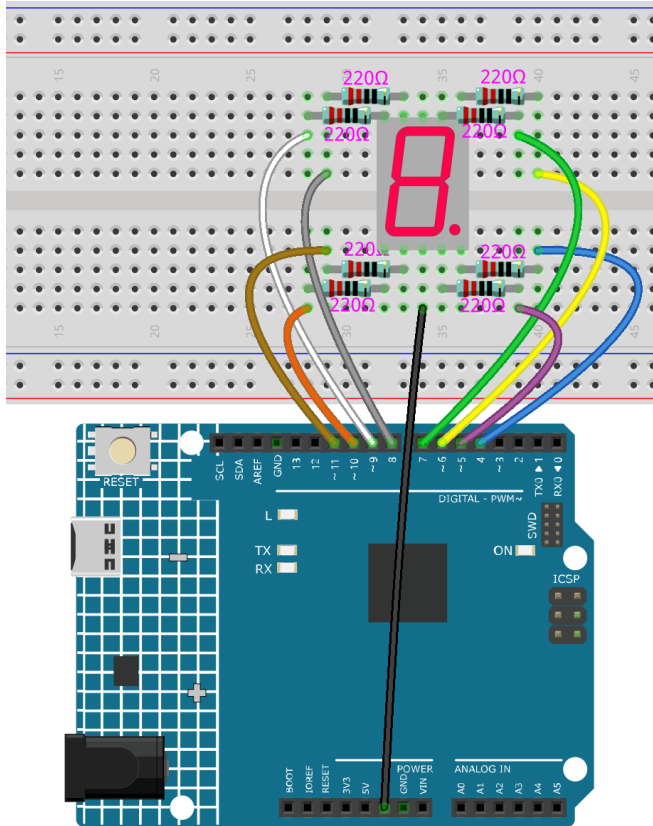


3.17.4 Experimental Procedures

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

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

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



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.17.5 Code

### 3.17.6 Code Analysis

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

**Call the function in loop()**

```
digital_1(); //display 1 to the 7-segment
delay(1000); //wait for a second

digital_2(); //display 2 to the 7-segment
delay(1000); //wait for a second

digital_3(); //display 3 to the 7-segment
delay(1000); //wait for a second
```

(continues on next page)

(continued from previous page)

```
digital_4(); //display 4 to the 7-segment
```

Calling these functions into the loop() is to let the 7-Segment display 0-F. The functions are shown below. Take *digital\_2()* for example:

#### Detailed analysis of digital\_2()

```
void digital_2() //display 2 to the 7-segment
{
  turnOffAllSegments();
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(g, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(d, HIGH);
}
```



First, we need to understand how the numeral **2** appears on the 7-Segment display. It is achieved by powering on segments a, b, d, e, and g. In programming, pins connected to these segments are set to a High level while c and f are set to Low level. We start by using the function `turnOffAllSegments()` to turn off all segments and then light up the required ones.

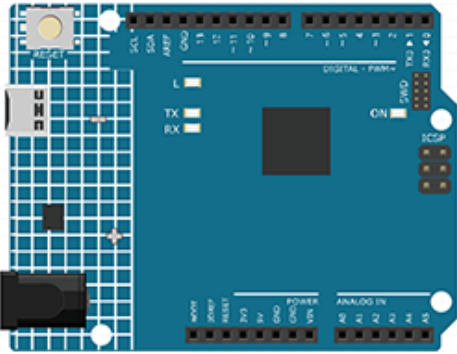


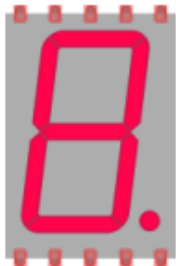
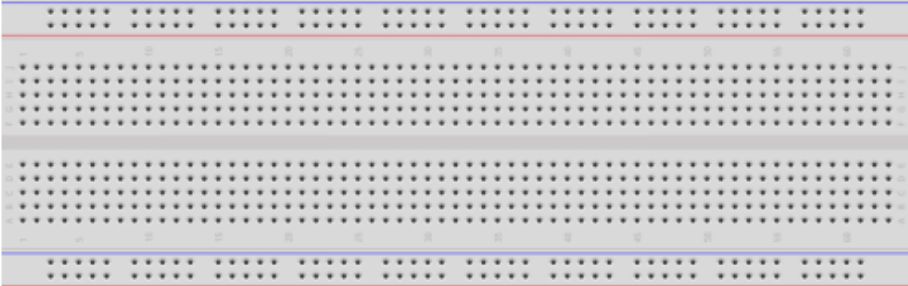


After running this part, the 7-segment will display **2**. Similarly, the display of other characters are the same. Since the letters b and d in upper case, namely **B** and **D**, would look the same with **8** and **0** on the display, they are displayed in lower case instead.

## 3.18 Lesson 18 74HC595

### 3.18.1 Introduction

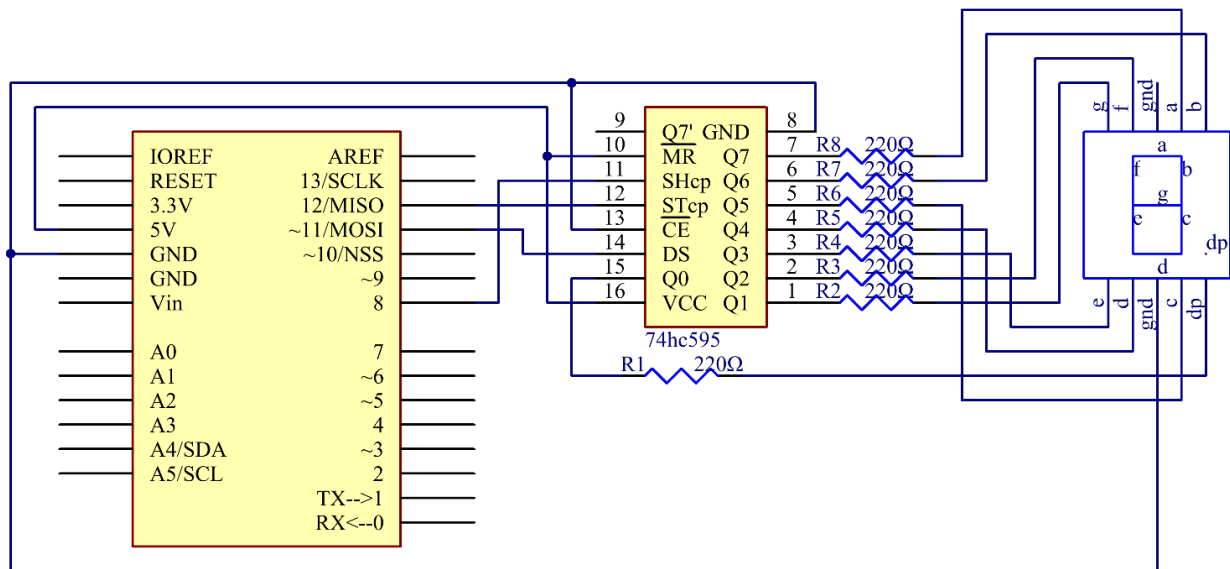
Generally, there are two ways to drive a single 7-segment display. One way is to connect its 8 pins directly to eight ports on the Uno board, which we have done previously. Or you can connect the 74HC595 to three ports of the UNO board and then the 7-segment display to the 74HC595. In this experiment, 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!

### 3.18.2 Components

<p>1 * Arduino Board</p> 	<p>8 * Resistor (220Ω)</p>  <p>1 * 74HC595</p> 	<p>1 * 7-segment Display</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

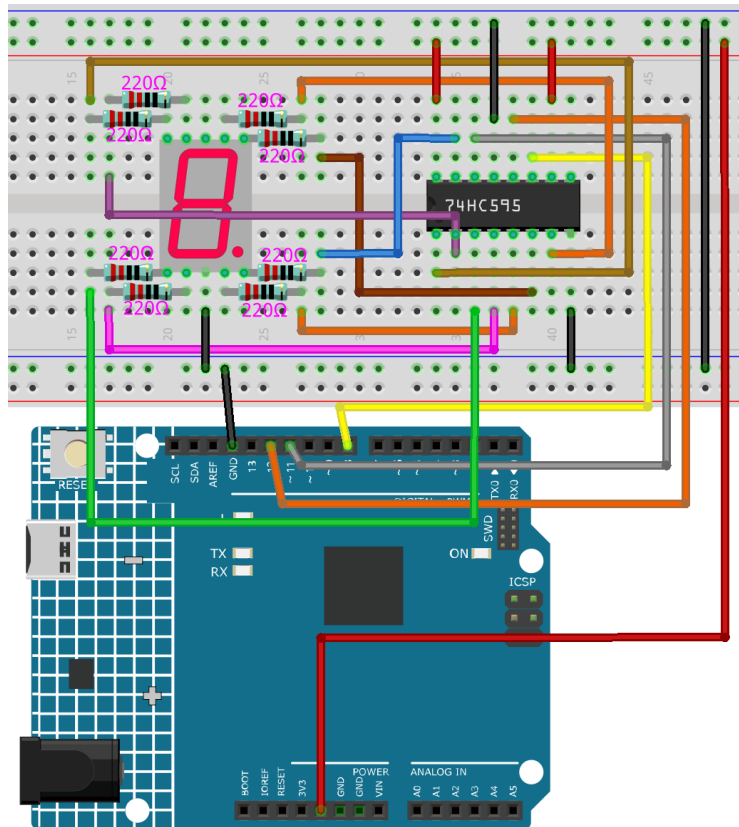
- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *7-segment Display*

In the experiment MR (pin10) is connected to 5V (HIGH Level) and OE (pin 1) 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.



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

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



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.18.5 Code

### 3.18.6 Code Analysis

Set the array elements

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

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

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

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

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

**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,dataArray[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.

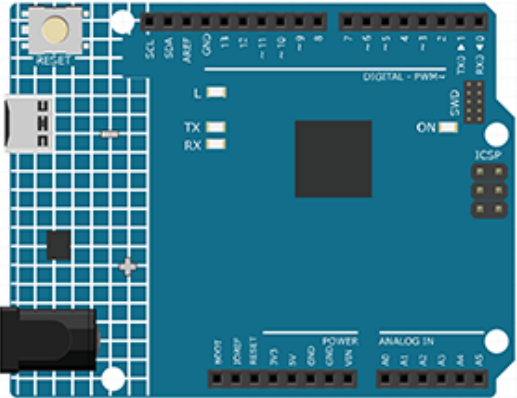
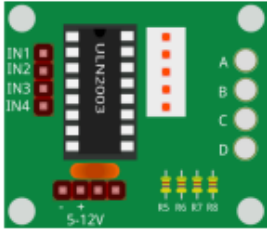
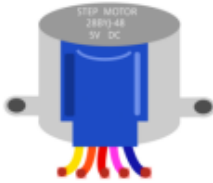

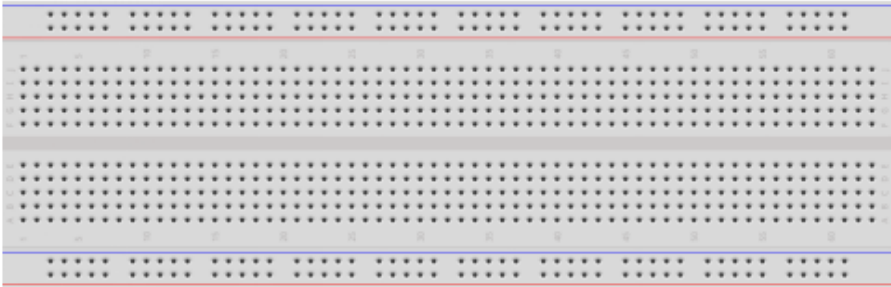


A byte of data will be transferred into the memory register after 8 times. Then the data of memory register is output to the bus (Q0-Q7). You will see a character is displayed on the 7-segment. Then delay for 1000ms. After that line, go back to *for()*. The loop repeats until all the characters are displayed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

## 3.19 Lesson 19 Stepper Motor

### 3.19.1 Introduction

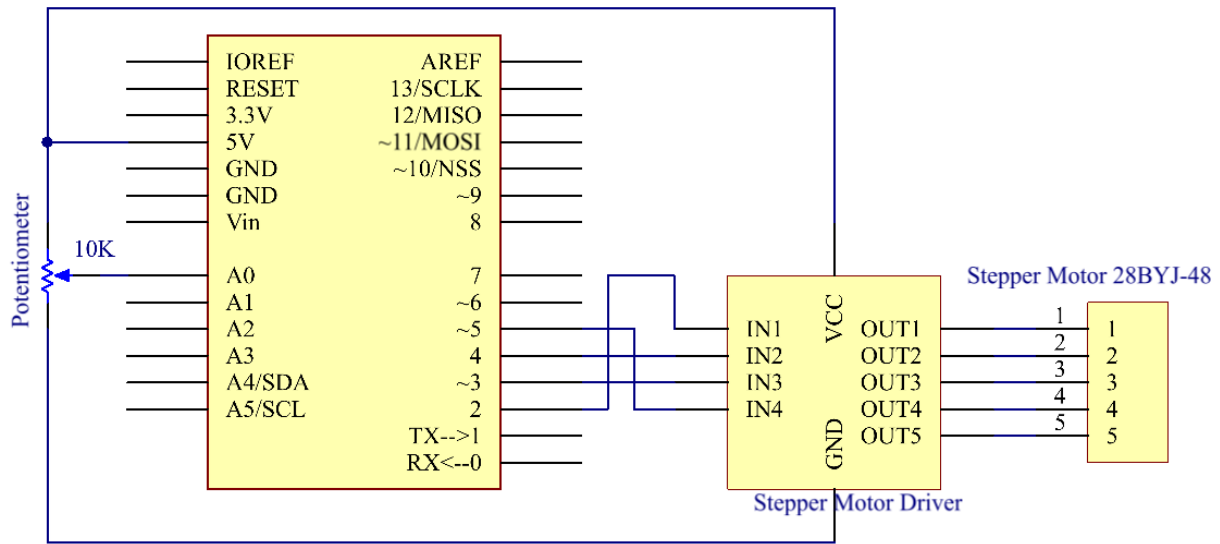
Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely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

### 3.19.2 Components

<p>1 * Arduino Board</p> 	<p>1 * Stepper Motor Driver</p> 	<p>1 * Stepper Motor</p>  <p>1 * Potentiometer</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Potentiometer*
- *Stepper Motor*

### 3.19.3 Schematic Diagram

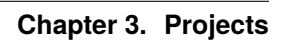


### 3.19.4 Experimental Procedures

#### Step 1: Build the circuit

The wiring between Stepper Motor Driver board and Uno board:

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



### 3.19.5 Code

### 3.19.6 Code Analysis

#### Initialize the stepper

```
#include <Stepper.h> //include a head file

//the steps of a circle

#define STEPS 2048

//set steps and the connection with MCU

Stepper stepper(STEPS, 2, 3, 4, 5);

//available to store previous value

int previous = 0;
```

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

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

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

#### Code Analysis 21-2 setSpeed() function

```
//speed of per minute

stepper.setSpeed(100); //set the motor speed in rotations per minute(RPMs)
```

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

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

#### loop() function

```
void loop()
{
    //get analog value

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

    //current reading minus the reading of history

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

    //store as previous value

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

The main program is to read the value of A0 first and then set the number of steps of stepper motor rotation according to the value of A0.

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

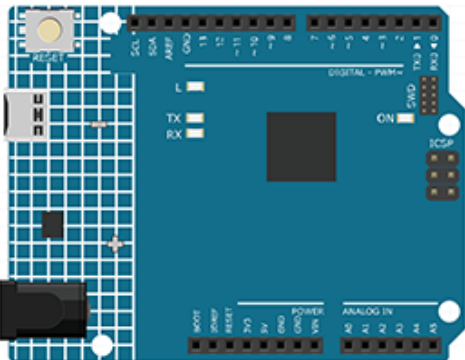


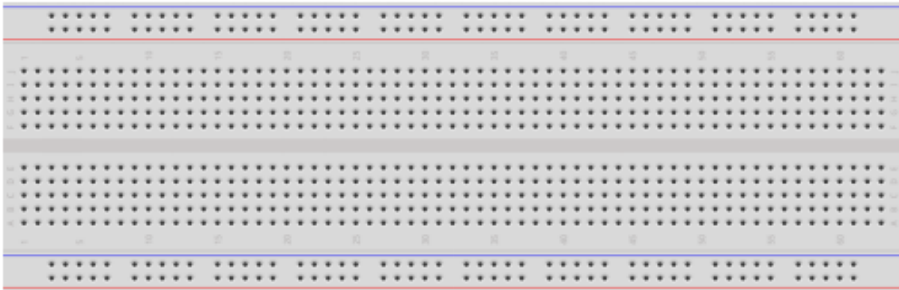


**steps:** the number of steps to turn the motor - positive to turn one direction, negative to turn the other (int).

## 3.20 Lesson 20 Simple Creation-Stopwatch

### 3.20.1 Introduction

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

### 3.20.2 Components

<p>1 * Arduino Board</p> 	<p>8 * Resistor (220Ω)</p> 	<p>1 * 4-Digit 7-Segment Display</p> 
<p>1 * Breadboard</p> 		
<p>1 * USB Cable</p> 	<p>Several Jump Wires</p> 	

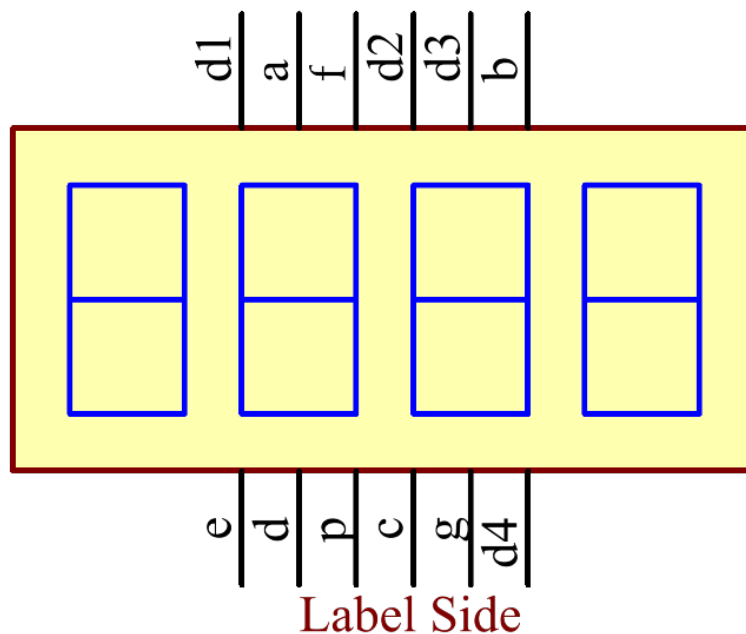
- *Arduino Uno R4 Minima*

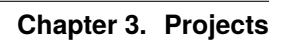
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *4-Digit 7-Segment Display*

### 3.20.3 Schematic Diagram

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

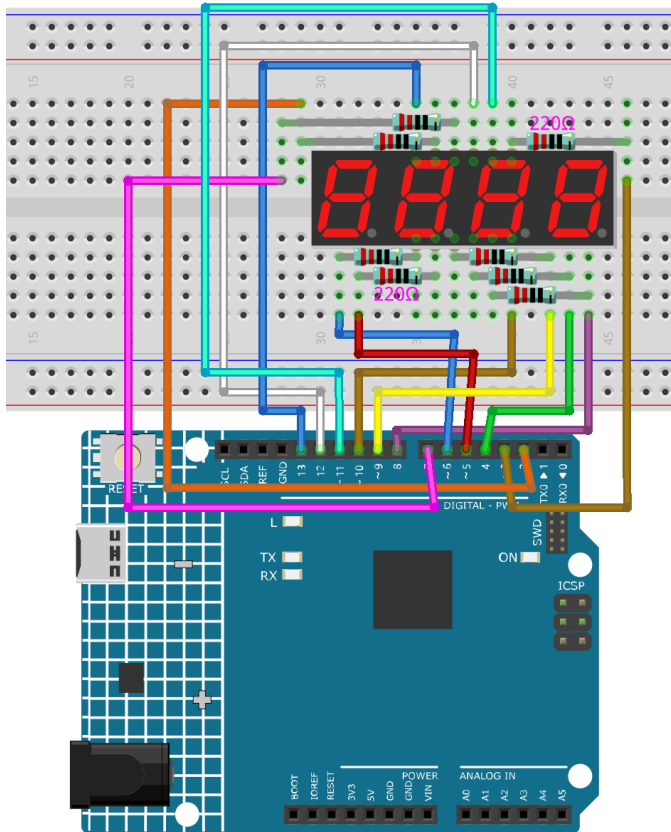
The schematic diagram of the 4-digit 7-segment display is as shown below:





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





**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.20.5 Code

### 3.20.6 Code Analysis

In essence, this code uses the principle of multiplexing to display a 4-digit number on a 7-segment display. By rapidly switching between digits and displaying one digit at a time, it gives the illusion of all digits being displayed concurrently. The stopwatch functionality is achieved by using the built-in `millis()` function to track time and increment the displayed number every second.

1. Variable and Constant Definitions:

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

long n = 0; // Variable to store the current stopwatch number
int del = 5; // Delay time (in milliseconds) to keep each digit illuminated
unsigned long previousMillis = 0; // Store the last time the stopwatch
    ← incremented
const long interval = 1000; // One-second interval (in milliseconds)
```

- `segmentPins` and `digitPins` arrays define the pins that are connected to the segments and the digits of the 7-segment display, respectively.
- `n` is a long variable that keeps track of the current stopwatch number, starting from 0 and incrementing.
- `del` is a delay time to maintain the display of the current digit before transitioning to the next one.
- `previousMillis` and `interval` are related to timing to decide when to increment the stopwatch.

## 2. 7-Segment Patterns for Numbers:

The 2D array `numbers` defines how each of the numbers 0-9 is represented on a common-cathode 7-segment display. Each sub-array has 8 values (either HIGH or LOW), corresponding to the 7 segments and a decimal point. This pattern helps in driving the appropriate segments for each number.

```
byte numbers[10][8] = {
  {HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW, LOW}, // 0
  {LOW, HIGH, HIGH, LOW, LOW, LOW, LOW, LOW},    // 1
  {HIGH, HIGH, LOW, HIGH, HIGH, LOW, HIGH, LOW},  // 2
  {HIGH, HIGH, HIGH, HIGH, LOW, LOW, HIGH, LOW},  // 3
  {LOW, HIGH, HIGH, LOW, LOW, HIGH, HIGH, LOW},   // 4
  {HIGH, LOW, HIGH, HIGH, LOW, HIGH, HIGH, LOW},  // 5
  {HIGH, LOW, HIGH, HIGH, HIGH, HIGH, HIGH, LOW}, // 6
  {HIGH, HIGH, HIGH, LOW, LOW, LOW, LOW, LOW},   // 7
  {HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW}, // 8
  {HIGH, HIGH, HIGH, HIGH, LOW, HIGH, HIGH, LOW}  // 9
};
```

## 3. Setup Function:

```
void setup() {
  // Configure all segment and digit pins as OUTPUT
  for (int i = 0; i < 8; i++) {
    pinMode(segmentPins[i], OUTPUT);
  }
  for (int i = 0; i < 4; i++) {
    pinMode(digitPins[i], OUTPUT);
    digitalWrite(digitPins[i], HIGH); // Initially turn off all digits.
    // (for common-cathode displays, HIGH is OFF)
  }
}
```

- All segment and digit pins are set to OUTPUT mode since they will drive the segments and digits of the display.
- Initially, all the digits are turned off, denoted by writing HIGH for a common-cathode display.

## 4. Main Loop:

```
void loop() {
  // Check if a second has passed since the last increment
  if (millis() - previousMillis >= interval) {
    previousMillis += interval; // Update the last increment time
    n = (n + 1) % 10000; // Increment the stopwatch number and wrap
  }
```

(continues on next page)

(continued from previous page)

```

↪around at 9999
    }

    displayNumber(n); // Display the current stopwatch number on the 7-
↪segment display
}

```

- This section checks if the interval (which is set to 1000ms or 1 second) has passed since the last increment of the stopwatch. If so, it increments the number.
- The number is then displayed on the 7-segment using the displayNumber() function.

#### 5. displayNumber() Function:

```

// Function to display a 4-digit number on the 7-segment display
void displayNumber(long num) {
    for (int digit = 0; digit < 4; digit++) {
        clearLEDs(); // Turn off all segments and digits
        pickDigit(digit); // Activate the current digit
        int value = (num / (int)pow(10, 3 - digit)) % 10; // Extract the
↪specific digit from the number
        pickNumber(value); // Illuminate the segments to display the digit
        delay(del); // Keep the digit illuminated for a short time
    }
}

```

- This function breaks down the 4-digit number into individual digits and displays each digit one at a time in rapid succession. This creates the illusion of all digits being displayed simultaneously due to persistence of vision.
- For each digit, the function first clears all LEDs, selects the appropriate digit using pickDigit(), and then displays the number on that digit using pickNumber().
- The delay (del) ensures each digit is visible for a short time before transitioning to the next.

#### 6. pickDigit() Function:

This function is responsible for selecting (or turning on) one of the four digits on the 7-segment display. This is achieved by setting the corresponding digit pin to LOW.

```

void pickDigit(int x) {
    digitalWrite(digitPins[x], LOW); // Turn ON the selected digit (for
↪common-cathode displays, LOW is ON)
}

```

#### 7. pickNumber() Function:

Given a single number (0-9), this function drives the 7-segment display's segments to show that number. It uses the previously defined numbers array to know which segments to turn on/off.

```

void pickNumber(int x) {
    for (int i = 0; i < 8; i++) {
        digitalWrite(segmentPins[i], numbers[x][i]); // Set each segment
↪according to the pattern for the given number
    }
}

```

#### 8. clearLEDs() Function:

As the name suggests, this function turns off all segments and digits. It's used to ensure that only one digit is active at a time during the multiplexing process in the `displayNumber()` function.

```
void clearLEDs() {  
    for (int i = 0; i < 8; i++) {  
        digitalWrite(segmentPins[i], LOW); // Turn off all segments  
    }  
    for (int i = 0; i < 4; i++) {  
        digitalWrite(digitPins[i], HIGH); // Turn off all digits  
    }  
}
```

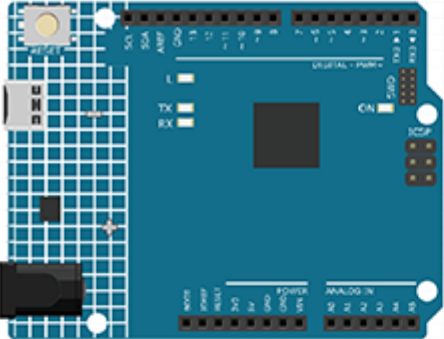










## 3.21 Lesson 21 Simple Creation-Answer Machine

### 3.21.1 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.

### 3.21.2 Components

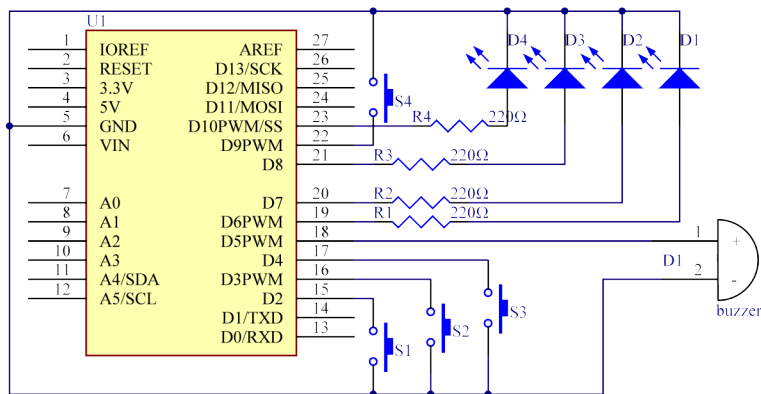
<b>1 * Arduino Board</b> 		<b>4 * Resistor (220Ω)</b> 	<b>4 * Button</b> 
		<b>1 * Active Buzzer</b> 	
<b>1 * Red LED</b> 	<b>1 * Yellow LED</b> 	<b>1 * Green LED</b> 	<b>1 * Blue LED</b> 
<b>1 * Breadboard</b> 			
<b>1 * USB Cable</b> 		<b>Several Jump Wires</b> 	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *LED*

- *Button*
- *Buzzer*

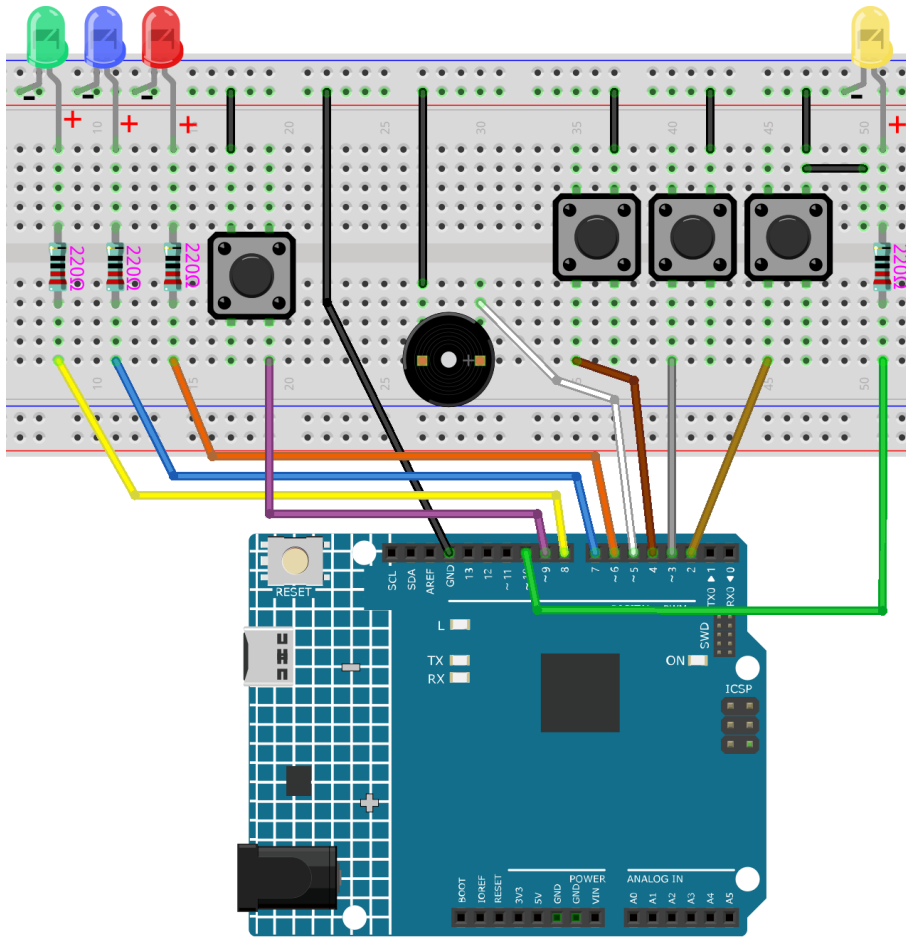
### 3.21.3 Schematic Diagram

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.



### 3.21.4 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.

### 3.21.5 Code

### 3.21.6 Code Analysis

The code for this experiment may be a bit long. But the syntax is very simple.

This code uses 6 nested if statements.

- The first if statement is used to determine if button 4 is pressed.
- The second if statement is used to determine again if button 4 is pressed, which is used to prevent false touches. If it is confirmed that it is pressed, the flag will be 1 and the LED will be lit.
- The third if statement is used to determine the value of flag, if it is 1 (button 4 is pressed), the value of button 1, 2 and 3 are read at this time.
- The fourth - six if statements are used to determine if buttons 1, 2, and 3 are pressed, and if they are pressed, then the LED is lit and the buzzer is sounded.

**Alarm() function**

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

This function is used to set the length and frequency of the sound emitted by the buzzer.

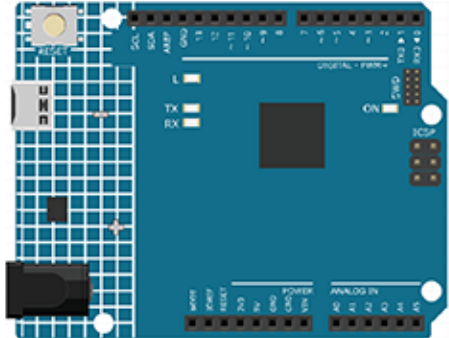
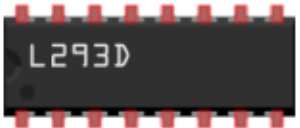



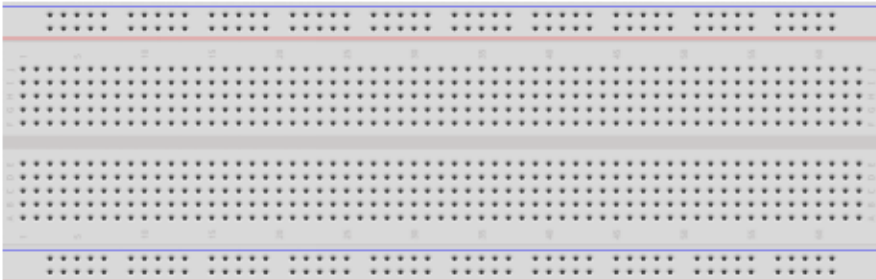



## 3.22 Lesson 22 Simple Creation-Small Fan

### 3.22.1 Introduction

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



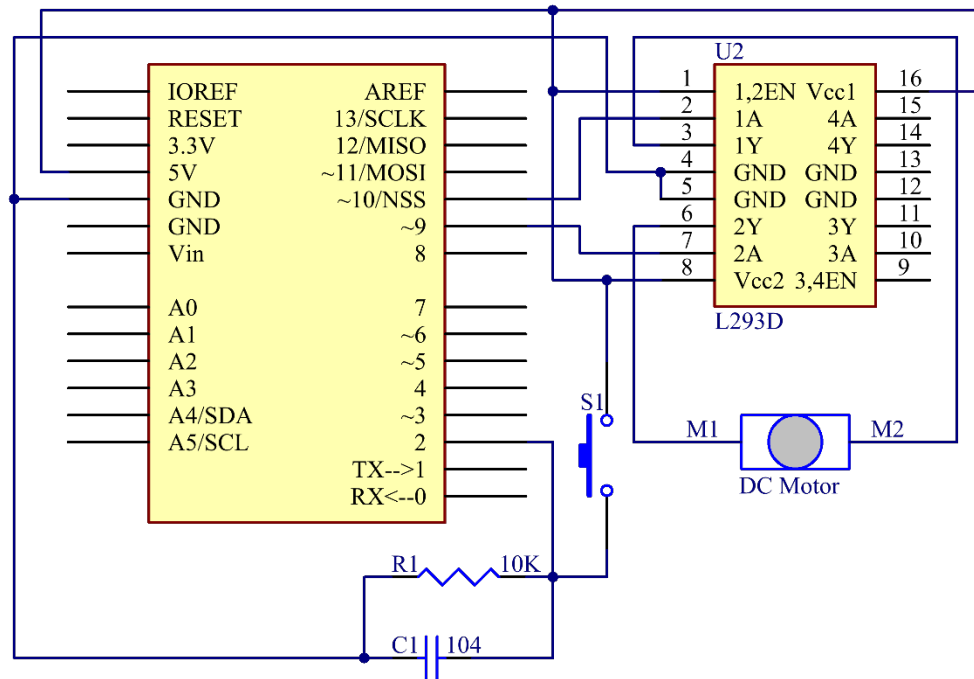
### 3.22.2 Components

1 * Arduino Board	1 * L293D	1 * 104 Capacitor
		
1 * Breadboard	1 * Small DC Motor	1 * Button
		
	1 * Fan	1 * Resistor (10kΩ)
		
1 * USB Cable	Several Jump Wires	
		

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*
- *Capacitor*
- *Button*
- *L293D*
- *DC Motor*

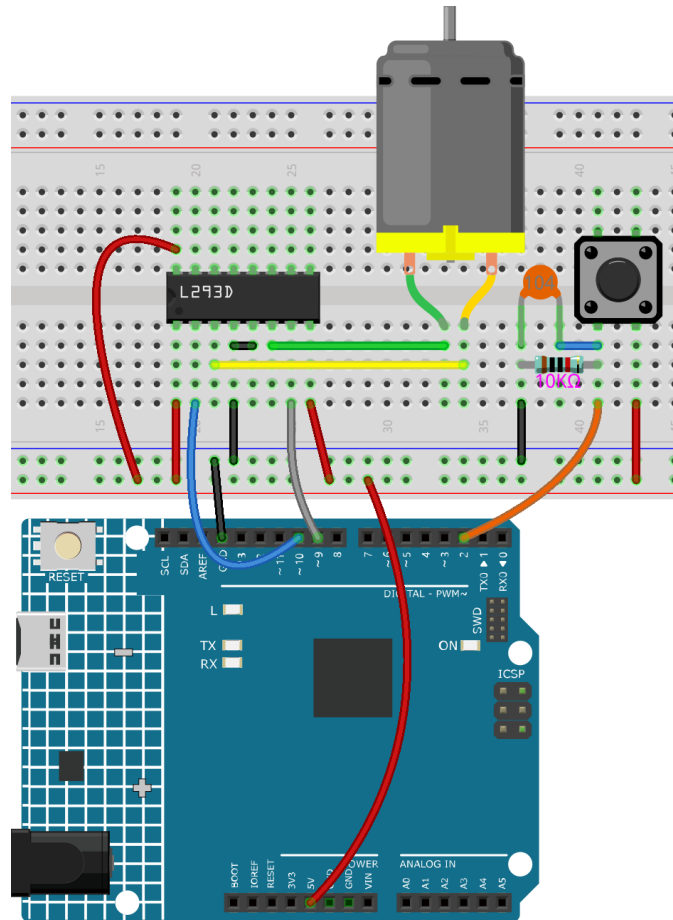
### 3.22.3 Schematic Diagram

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



### 3.22.4 Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.22.5 Code

### 3.22.6 Code Analysis

This code nests five `if` statements to determine the button press status.

- The first `if` statement is used to determine if the button is pressed.
- The second `if` statement is used to determine if 50ms has elapsed.
- The third `if` statement is used to determine if the button has been pressed after 50ms, so as to avoid false touches.
- The fourth `if` statement is used to record the number of button presses, adding 1 to `stat` for each press.
- The fifth `if` statement is used to determine if the number of button presses is greater than 4. If so, `stat` is cleared to zero.

**switch() statement**

```
switch(stat)
{
case 1:
    clockwise(rank1);// When stat=1, set the rotate speed of the motor as rank1=150
    break;
case 2:
    clockwise(rank2);// When stat=2, set the rotate speed of the motor as rank1=200
    break;
case 3:
    clockwise(rank3);// When stat=3, set the rotate speed of the motor as rank1=250
    break;
default:
    clockwise(0);
}
```

The switch statement, like the if statement, switch case allows the programmer to control the flow of the program with different code executed under various conditions. In particular, the switch statement compares the value of a variable with the value specified in the case statement. When a case statement is found whose value matches the value of a variable, the code in that case statement is run. If there is no break statement, the switch statement will continue to execute the following expression until break or until it reaches the end of the switch statement.

In this part of the code.

- If stat = 1, let the fan rotate at speed rank1(150).
- If stat = 1, let the fan rotate at speed rank2(200).
- If stat = 1, let the fan rotate at speed rank3(250).
- If stat = 0, let the fan rotate at speed 0.

#### clockwise() function

```
void clockwise(int Speed)//
{
    analogWrite(motorIn1,0);
    analogWrite(motorIn2,Speed);
}
```

This function sets the speed of the motor: write Speed to pin 9 and 0 to pin 10. The motor rotates in a certain direction with the value of Speed.

### 3.22.7 Experiment Summary

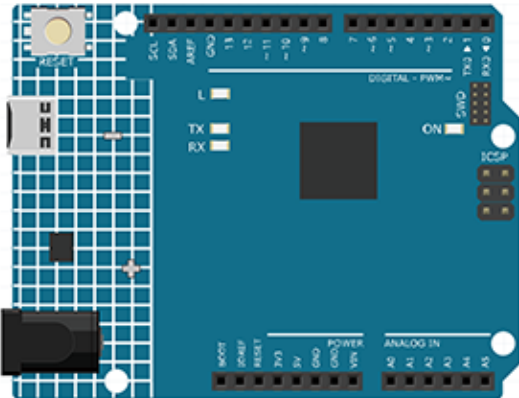





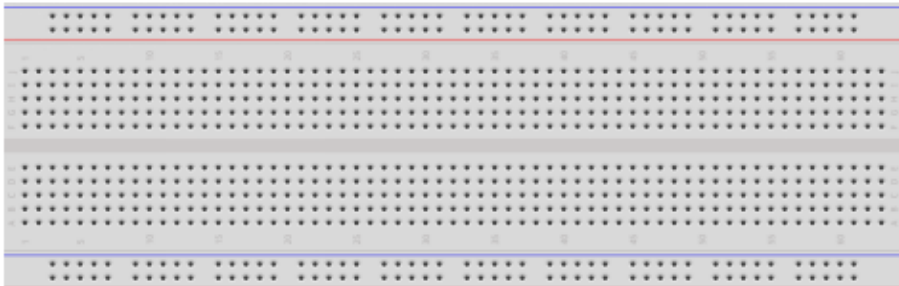



In this experiment, you can also control the motor to rotate or not. Just connect pin 1, 2EN of the L293D to an I/O port of the control board. Set 1, 2EN as High level, and the motor will start rotating; set it as Low level, it will stop the rotating.

## 3.23 Lesson 23 Simple Creation - Digital Dice

### 3.23.1 Introduction

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

### 3.23.2 Components

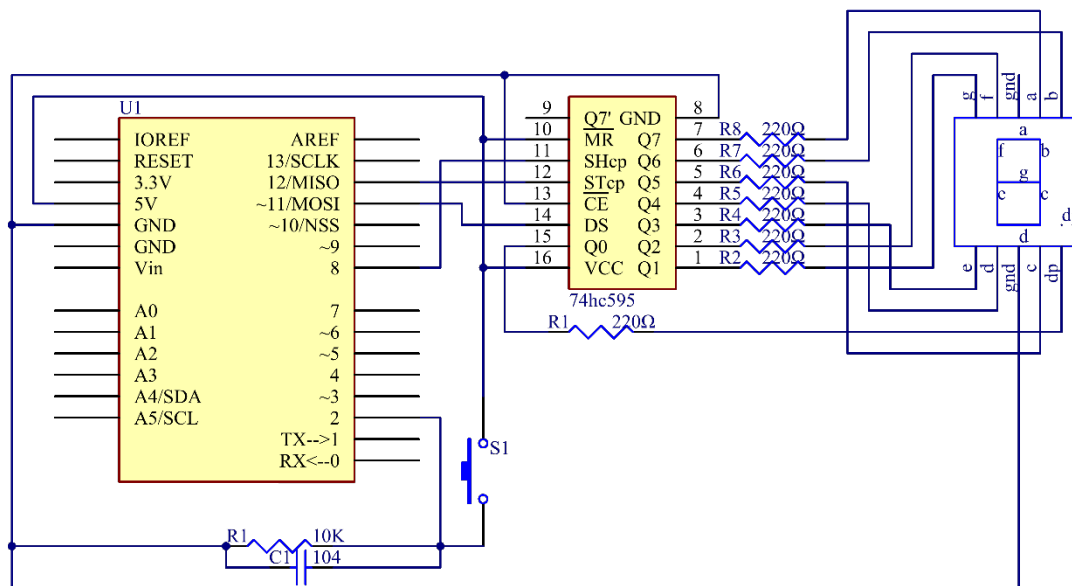
<div>1 * Arduino Board</div> <div></div>	<div>1 * 104 Capacitor</div> <div></div>	<div>1 * Button</div> <div></div>
	<div>8 * Resistor (220Ω)</div> <div></div>	<div>1 * 74HC595</div> <div></div>
	<div>1 * Resistor (10kΩ)</div> <div></div>	
<div>1 * Breadboard</div> <div></div>	<div>1 * 7-segment Display</div> <div></div>	
<div>1 * USB Cable</div> <div></div>	<div>Several Jumper Wires</div> <div></div>	

- *Arduino Uno R4 Minima*
- *Breadboard*
- *Jumper Wires*
- *Resistor*

- 7-segment Display
- 74HC595
- Button
- Capacitor

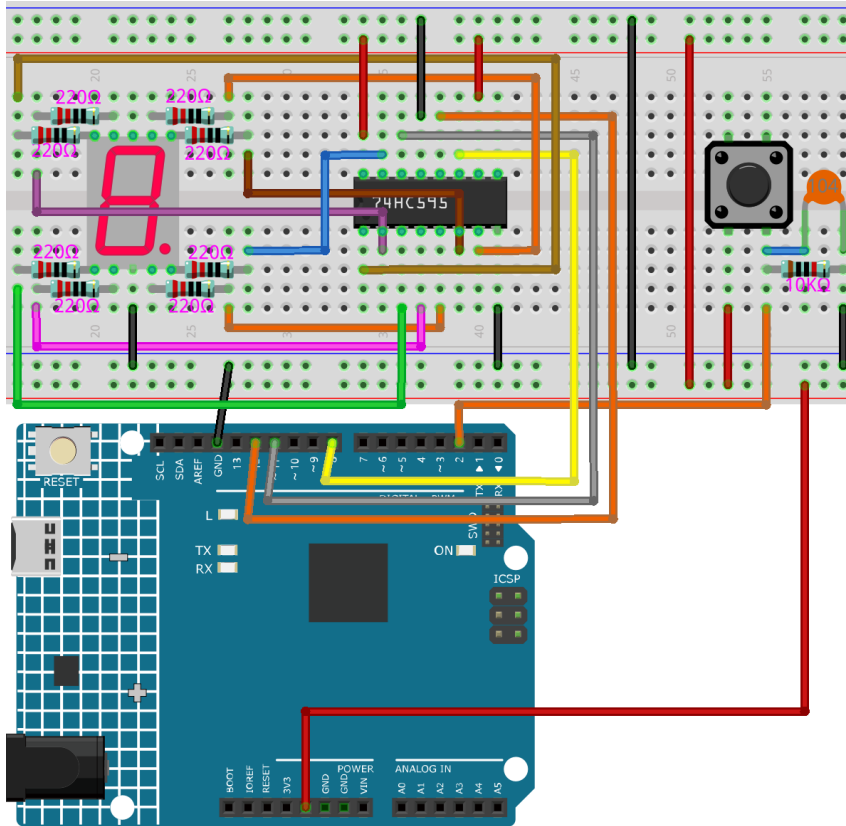
### 3.23.3 Schematic Diagram

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



### 3.23.4 Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Open the code file.

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

**Step 4:** Upload the sketch to the board.

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

### 3.23.5 Code

### 3.23.6 Code Analysis

The initial random number comes from A0

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

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

**Digital Dice**

```
void loop()
{
    int stat = digitalRead(keyIn); //store value read from keyIn
    if(stat == HIGH) // check if the pushbutton is pressed
```

If yes, the corresponding pin is high level.

```
{
  num ++; // num adds 1
  if(num > 1)
  {
    num = 0;
  }
}
```

If num > 1, clear the value. This is to prevent repeated pressing. So just count it as once no matter how many times you press.

```
Serial.println(num); // print the num on serial monitor
if(num == 1) //when pushbutton is pressed
{
  randomNumber = random(1,7); //Generate a random number in 1-7
  showNum(randomNumber); //show the randomNumber on 7-segment
  delay(1000); //wait for 1 second
  while(!digitalRead(keyIn)); //When not press button,program stop here.
```

Make it keep displaying the last random number.

```
int stat = digitalRead(keyIn);
```

Read the state of the button again.

```
if(stat == HIGH) // check if the pushbutton is pressed
```

If yes, run the code below.

```
{
  num ++; // num+1=2
  digitalWrite(ledPin,HIGH); //turn on the led
  delay(100);
  digitalWrite(ledPin,LOW); //turn off the led
  delay(100);
  if(num >= 1) // clear the num
  {
    num = 0;
  }
}
}
//show random numbers at 100 microseconds intervals
//If the button has not been pressed
randomNumber = random(1,7);
showNum(randomNumber);
delay(100);
}
```

**showNum() function**

```
void showNum(int num)
{
```

(continues on next page)



(continued from previous page)

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

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



## THANK YOU

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

### Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

---

**Note:** After submitting the questionnaire, please go back to the top to view the results.

---



## **COPYRIGHT NOTICE**

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.