



Aedept

## Ultimate kit for Arduino

Sharing Perfects Innovation

Processing



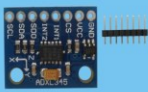
# Preface





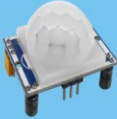


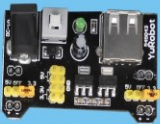

Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

This is an entry-level learning kit for Arduino. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of Arduino, and be able to make fascinating works based on Arduino.










If you have any problems for learning, please contact us at [support@adeept.com](mailto:support@adeept.com). We will do our best to help you solve the problem.










## Component List







<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
<b>1</b>	<b>Adept UNO Board(Arduino UNO)</b>		<b>1</b>
<b>2</b>	<b>ADXL345 Acceleration Sensor</b>		<b>1</b>
<b>3</b>	<b>Ultrasonic Distance Sensor</b>		<b>1</b>
<b>4</b>	<b>IR Receiver Hx1838</b>		<b>1</b>
<b>5</b>	<b>Remote Controller</b>		<b>1</b>
<b>6</b>	<b>Ps2 Joystick Module</b>		<b>1</b>
<b>7</b>	<b>Relay</b>		<b>1</b>
<b>8</b>	<b>Stepper Motor</b>		<b>1</b>
<b>9</b>	<b>ULN2003 Stepper Motor Driver Module</b>		<b>1</b>










<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
<b>10</b>	DHT-11 Temperature & Humidity Sensor		<b>1</b>
<b>11</b>	LED Bar Graph		<b>1</b>
<b>12</b>	Active Buzzer		<b>1</b>
<b>13</b>	Passive Buzzer		<b>1</b>
<b>14</b>	PIR Movement Sensor		<b>1</b>
<b>15</b>	Servo		<b>1</b>
<b>16</b>	Analog Temperature Sensor(Thermistor)		<b>2</b>
<b>17</b>	Breadboard Power Supply Module		<b>1</b>
<b>18</b>	4*4 Matrix Keyboard		<b>1</b>


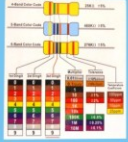


<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
19	DC Motor		1
20	L9110 DC Motor Driver		1
21	LCD1602		1
22	Dot-matrix Display		1
23	7-segment Display		1
24	4-bit 7-segment Display		1
25	Ne555 Timer		1
26	74HC595		2
27	Light Sensor (Photoresistor)		2

<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
28	Tilt Switch		2
29	Switch		2
30	RGB LED		1
31	Red LED		8
32	Green LED		4
33	Yellow LED		4
34	Blue LED		4
35	Resistor(220Ω)		16
36	Resistor(1kΩ)		10

NO. 0	Name	Picture	Qty
37	Resistor(10kΩ)		10
38	Potentiometer (10KΩ)		2
39	Capacitor(104)		5
40	Capacitor(10uF)		4
41	Button(large)		4
42	Button(small)		8
43	Button Cap(red)		1
44	Button Cap(white)		1
45	Button Cap(blue)		2

<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
<b>46</b>	<b>NPN Transistor(8050)</b>		<b>2</b>
<b>47</b>	<b>PNP Transistor(8550)</b>		<b>2</b>
<b>48</b>	<b>1N4148 Diode</b>		<b>2</b>
<b>49</b>	<b>1N4001 Diode</b>		<b>2</b>
<b>50</b>	<b>Battery Holder</b>		<b>1</b>
<b>51</b>	<b>Breadboard</b>		<b>1</b>
<b>52</b>	<b>USB Cable</b>		<b>1</b>
<b>53</b>	<b>Male to Male Jumper Wires</b>		<b>40</b>
<b>54</b>	<b>Male to Female Jumper Wires</b>		<b>20</b>

<b>NO. 0</b>	<b>Name</b>	<b>Picture</b>	<b>Qty</b>
<b>55</b>	<b>Header(40pin)</b>		<b>1</b>
<b>56</b>	<b>Band Resistor Card</b>		<b>1</b>

Adeept

# Content

About Arduino.....	- 1 -
About Processing.....	- 2 -
Lesson 1 Blinking LED.....	- 3 -
Lesson 2 Active Buzzer.....	- 8 -
Lesson 3 Controlling an LED with a button.....	- 12 -
Lesson 4 Controlling Relay.....	- 17 -
Lesson 5 Serial Port.....	- 20 -
Lesson 6 LED Flowing Lights.....	- 25 -
Lesson 7 LED bar graph display.....	- 28 -
Lesson 8 Breathing LED.....	- 32 -
Lesson 9 Controlling a RGB LED by PWM.....	- 36 -
Lesson 10 Play the Music.....	- 39 -
Lesson 11 LCD1602 display.....	- 43 -
Lesson 12 A Simple Voltmeter.....	- 47 -
Lesson 13 7-segment display.....	- 50 -
Lesson 14 A simple counter.....	- 54 -
Lesson 15 Controlling Servo motor.....	- 57 -
Lesson 16 Using a thermistor to measure the temperature.....	- 60 -
Lesson 17 IR Remoter Controller.....	- 63 -
Lesson 18 Temperature & humidity sensor DHT-11.....	- 67 -

Lesson 19 Ultrasonic distance sensor .....	- 71 -
Lesson 20 3-axis Accelerometer—ADXL345 .....	- 74 -
Lesson 21 4x4 matrix keyboard .....	- 80 -
Lesson 22 Controlling DC motor.....	- 84 -
Lesson 23 Joy stick.....	- 89 -
Lesson 24 Tilt Switch.....	- 92 -
Lesson 25 Dot-matrix display.....	- 95 -
Lesson 26 Controlling Stepper Motor.....	- 100 -
Lesson 27 Photoresistor.....	- 103 -
Lesson 28 Automatically Tracking Light Source.....	- 106 -
Lesson 29 Frequency meter .....	- 108 -
Lesson 30 Intrusion Detection based on the PIR.....	- 113 -
Lesson 31 Control a relay with IR remoter controller.....	- 116 -
Lesson 32 Control a RGB LED with IR remoter controller .....	- 118 -
Lesson 33 control a stepper motor with IR remoter controller.....	- 120 -
Lesson 34 Controlling the size of the circle by potentiometer .....	- 122 -
Lesson 35 Controlling the 3D model by PS2 Joystick.....	- 129 -
Lesson 36 Snake Game .....	- 132 -
Lesson 37 Star Wars .....	- 135 -



# About Arduino

## *What is Arduino?*

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

## **ARDUINO BOARD**

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

## **ARDUINO SOFTWARE**

You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment.

Before the development of Arduino program, the first thing you have to do is to install Arduino IDE software. The software provides you with the basic development environment that is required for developing Arduino program.

You need the following URL to download Arduino IDE:

<http://www.arduino.cc/en/Main/Software>

For different operating system platforms, the way of using Arduino IDE is different. Please refer to the following links:

Windows User : <http://www.arduino.cc/en/Guide/Windows>

Mac OS X User : <http://www.arduino.cc/en/Guide/MacOSX>

Linux User : <http://playground.arduino.cc/Learning/Linux>

For more detailed information about Arduino IDE, please refer to the following link:

<http://www.arduino.cc/en/Guide/HomePage>

# About Processing

## *What is Processing?*

Processing is a programming language, development environment, and online community. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. Initially created to serve as a software sketchbook and to teach computer programming fundamentals within a visual context, Processing evolved into a development tool for professionals. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

- » Free to download and open source
- » Interactive programs with 2D, 3D or PDF output
- » OpenGL integration for accelerated 3D
- » For GNU/Linux, Mac OS X, and Windows
- » Over 100 libraries extend the core software

## **PROCESSING SOFTWARE**

Download Processing:

<https://www.processing.org/download/>

For more detailed information about Processing IDE, please refer to the following link:

<https://www.processing.org/reference/environment/>

# Lesson 1 Blinking LED

## Overview

In this tutorial, we will start the journey of learning Arduino UNO. In the first lesson, we will learn how to make a LED blinking.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 220Ω Resistor
- 1\* LED
- 1\* Breadboard
- 2\* Jumper Wires

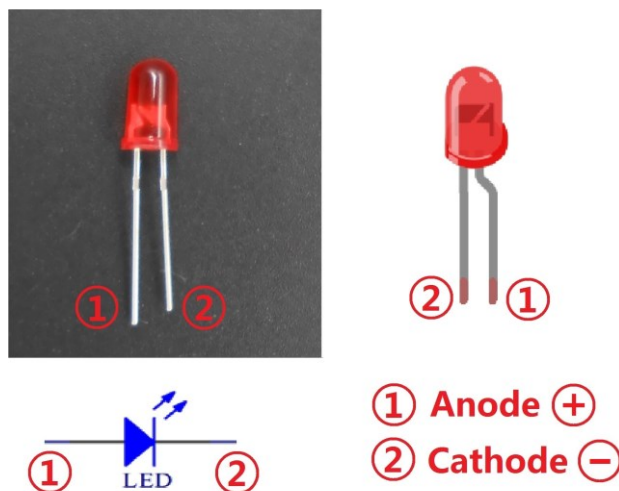
## Principle

In this lesson, we will program the Arduino's GPIO output high(+5V) and low level(0V), and then make the LED which is connected to the Arduino's GPIO flicker with a certain frequency.

### 1. What is the LED?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphide semiconductor materials. The LED has two electrodes, a positive electrode and a negative electrode, it will light only when a forward current passes, and it can be red, blue, green or yellow light, etc. The color of light depends on the materials it was made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.



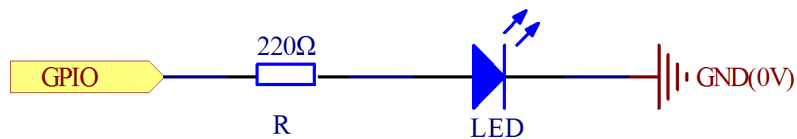
## 2. What is the resistor?

The main function of the resistor is to limit current. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm( $\Omega$ ).

The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

There are two methods for connecting LED to Arduino's GPIO:

①



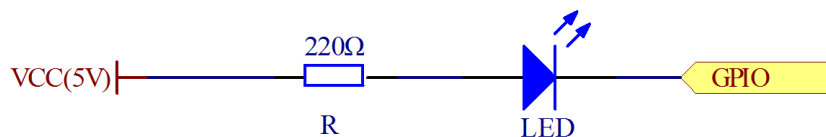
As shown in the schematic diagram above, the anode of LED is connected to Arduino's GPIO via a resistor, and the cathode of LED is connected to the ground(GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

The size of the current-limiting resistor is calculated as follows: 5~20mA current is required to make an LED on, and the output voltage of the Arduino UNO's GPIO is 5V, so we can get the resistance :

$$R = U / I = 5V / (5\sim 20mA) = 250\Omega\sim 1K\Omega$$

Since the LED has a certain resistance, thus we choose a 220ohm resistor.

②



As shown in the schematic diagram above, the anode of LED is connected to VCC(+5V), and the cathode of LED is connected to the Arduino's GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

The experiment is based on method ①, we select Arduino's D8 pin to control the LED. When the Arduino's D8 pin is programmed to output high level, then the LED will be on, next delay for the amount of time, and then programmed the D8 pin to low level to make the LED off. Continue to perform the above process, you can get a blinking LED.

### 3. Key functions:

#### ● `setup()`

The `setup()` function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

#### ● `loop()`

After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

#### ● `pinMode()`

Configures the specified pin to behave either as an input or an output.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode `INPUT_PULLUP`. Additionally, the `INPUT` mode explicitly disables the internal pullups.

#### ● `digitalWrite()`

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with `pinMode()`, its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

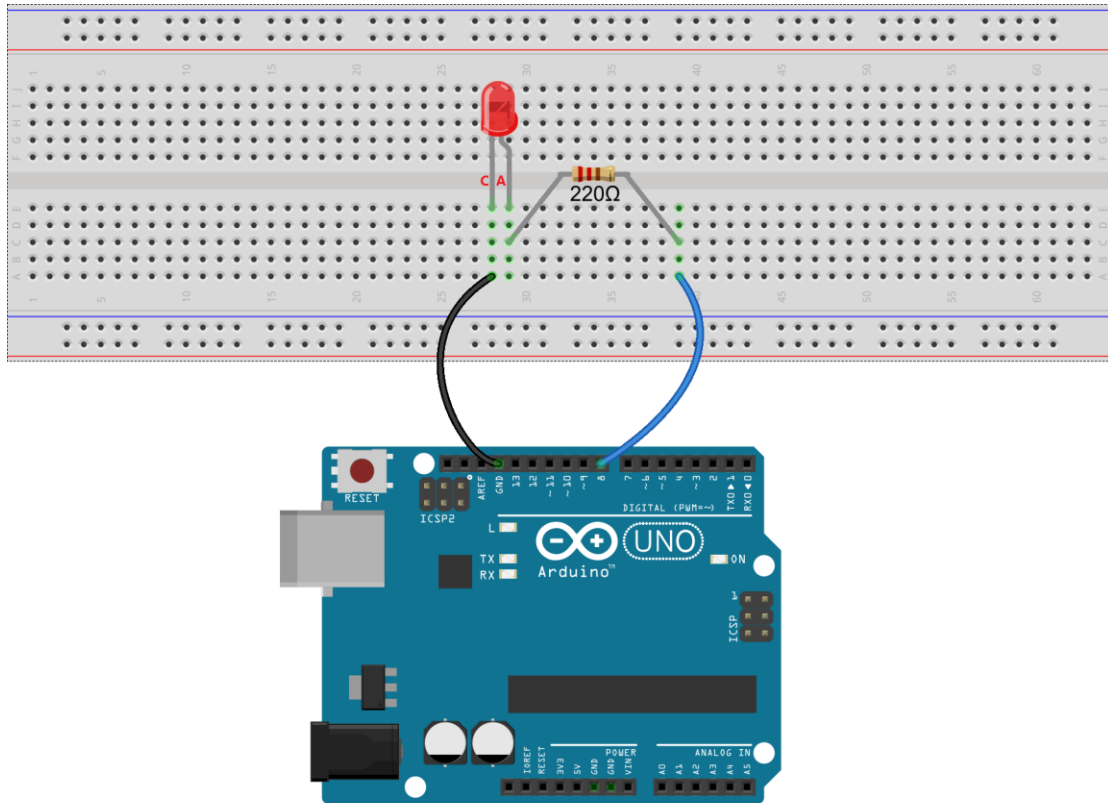
If the pin is configured as an INPUT, `digitalWrite()` will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the `pinMode()` to `INPUT_PULLUP` to enable the internal pull-up resistor.

#### ● `delay()`

Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

## Procedures

### 1. Build the circuit



fritzing

## 2. Program

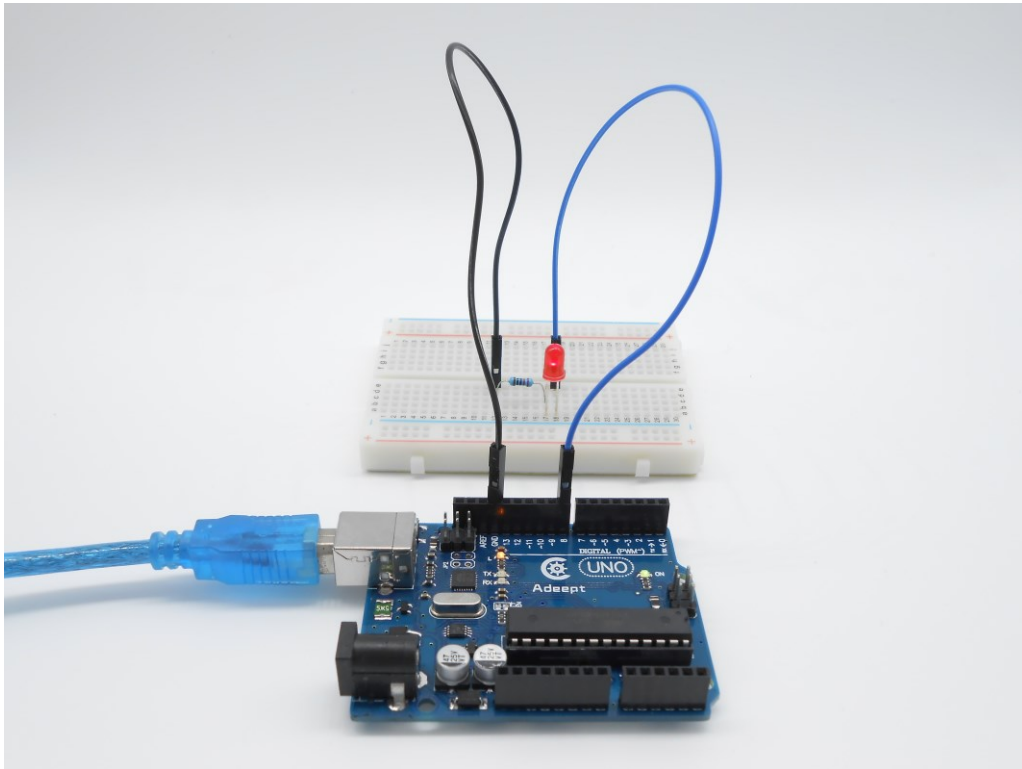
```

/*****
File name: 01_blinkingLed.ino
Description: Lit LED, let LED blinks.
Website: www.aadept.com
E-mail: support@aadept.com
Author: Tom
Date: 2015/05/02
*****/
int ledPin=8; //definition digital 8 pins as pin to control the LED
void setup()
{
  pinMode(ledPin,OUTPUT); //Set the digital 8 port mode, OUTPUT:
Output mode
}
void loop()
{
  digitalWrite(ledPin,HIGH); //HIGH is set to about 5V PIN8
  delay(1000); //Set the delay time, 1000 = 1S
  digitalWrite(ledPin,LOW); //LOW is set to about 5V PIN8
  delay(1000); //Set the delay time, 1000 = 1S
}

```

3. Compile the program and upload to Arduino UNO board

Now, you can see the LED is blinking.





## Lesson 2 Active Buzzer

### Overview

In this lesson, we will learn how to program the Arduino to make an active buzzer sound.

### Requirement

- 1\* Arduino UNO
- 1\* USB cable
- 1\* Active buzzer
- 1\* 1 k $\Omega$  Resistor
- 1\* NPN Transistor (S8050)
- 1\* Breadboard
- Several Jumper Wires

### Principle

A buzzer or beeper is an audio signaling device. As a type of electronic buzzer with integrated structure, which use DC power supply, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic equipments, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive buzzers (See the following pictures).

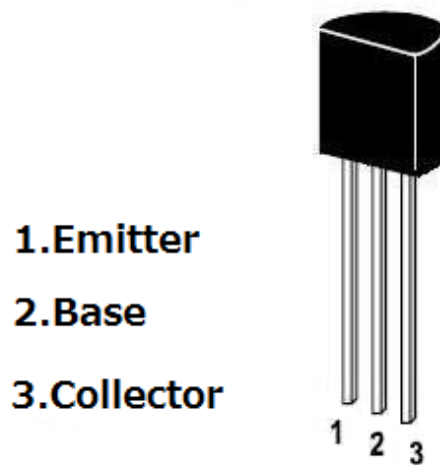


When you place the pins of buzzers upward, you can see that two buzzers are different, the buzzer that green circuit board exposed is the passive buzzer.

In this study, the buzzer we used is active buzzer. Active buzzer will sound as long as the power supply. We can program to make the Arduino output alternating high and low level, so that the buzzer sounds.

A slightly larger current is needed to make a buzzer sound. However, the output current of Arduino's GPIO is weak, so we need a transistor to drive the buzzer.

The main function of transistor is blowing up the voltage or current. The transistor can also be used to control the circuit conduction or deadline. And the transistor is divided into two kinds, one kind is NPN, for instance, the S8050 we provided; another kind is PNP transistor such as the S8550 we provided. The transistor we used is as shown in below:



There are two driving circuit for the buzzer:

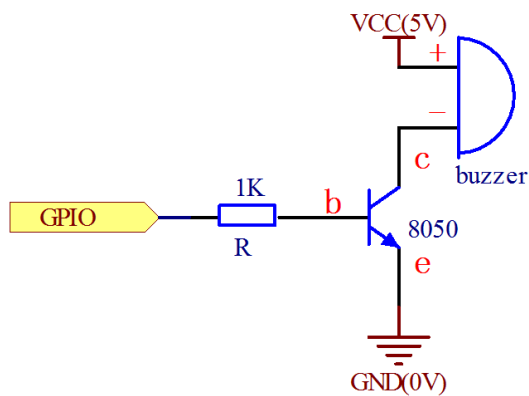


Figure1

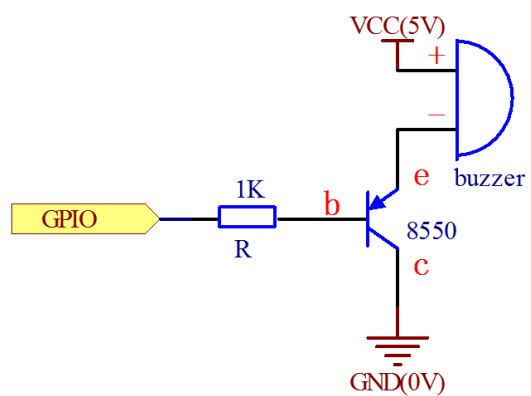


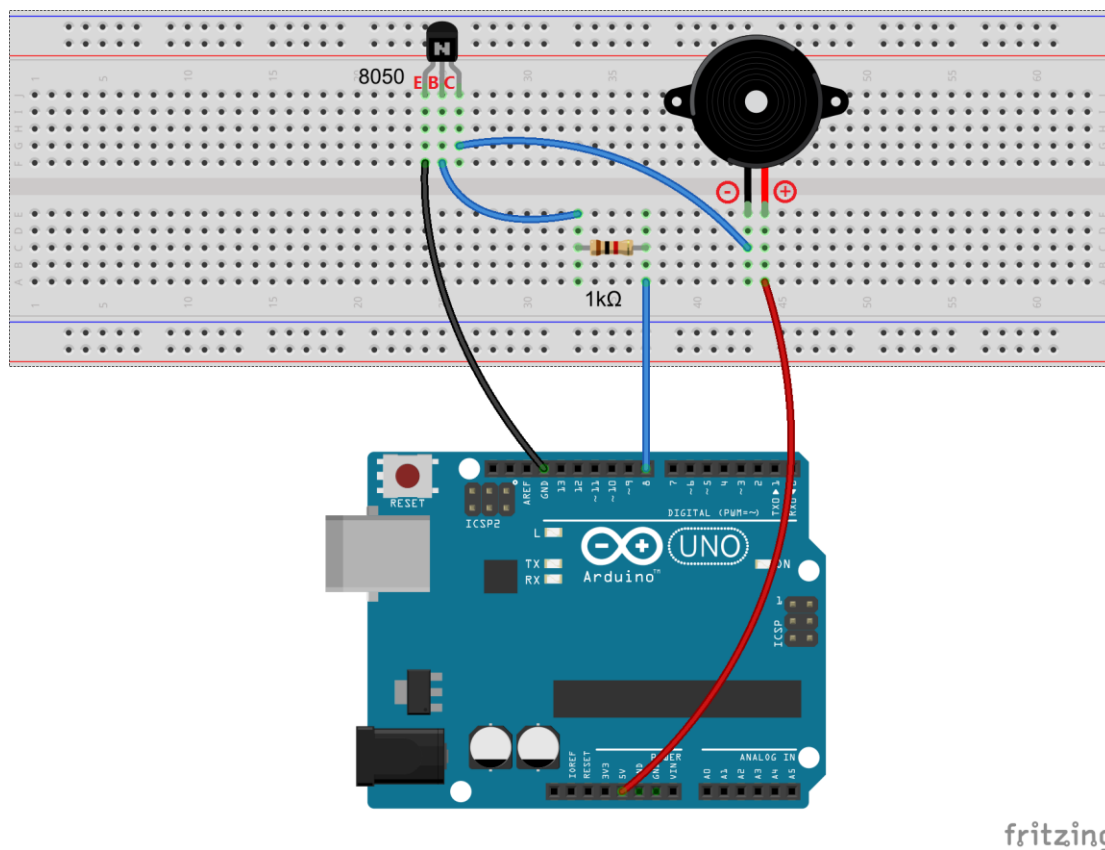
Figure2

Figure 1: Set the Arduino GPIO as a high level, the transistor S8050 will conduct, and then the buzzer will sound; set the Arduino GPIO as low level, the transistor S8050 will cut off, then the buzzer will stop.

Figure 2: Set the Arduino GPIO as low level, the transistor S8550 will conduct, and the buzzer will sound; set the Arduino GPIO as a high level, the transistor S8550 will cut off, then the buzzer will stop.

## Procedures

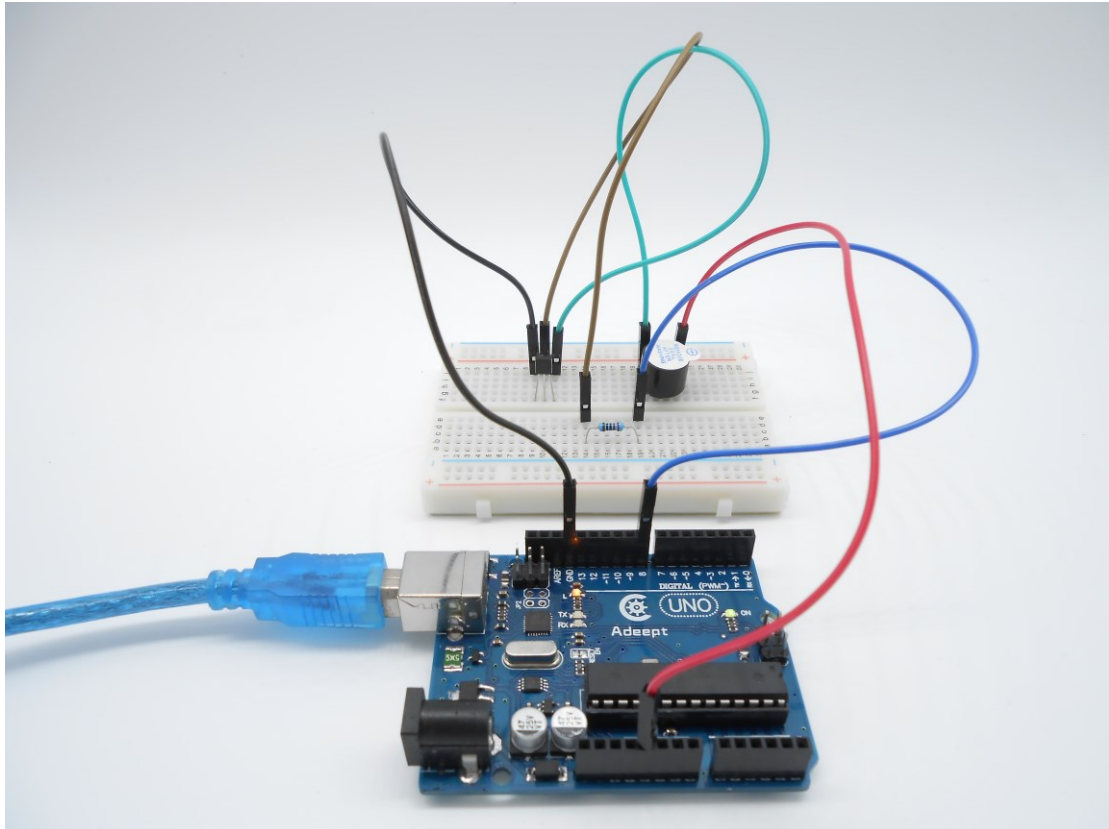
### 1. Build the circuit



### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you should be able to hear the sound of the buzzer.



## Summary

By learning this lesson, we have mastered the basic principle of the buzzer and the transistor. We also learned how to program the Arduino and then control the buzzer. I hope you can use what you have learned in this lesson to do some interesting things.

# Lesson 3 Controlling an LED with a button

## Overview

In this lesson, we will learn how to detect the state of a button, and then toggle the state of LED based on the state of the button.

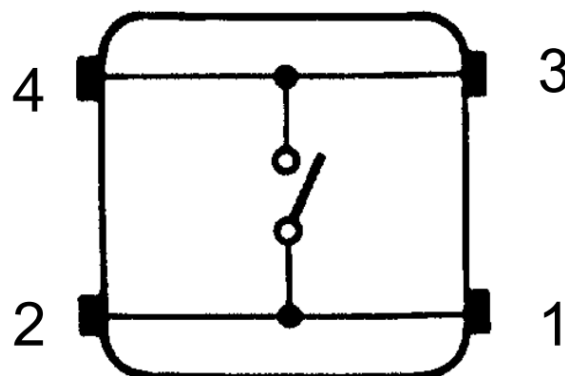
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Button
- 1\* LED
- 1\* 10K $\Omega$  Resistor
- 1\* 220 $\Omega$  Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

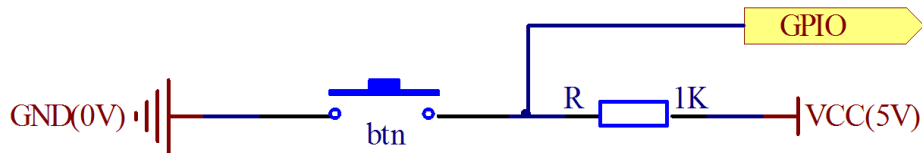
### 1. Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown in the following pictures. Pins pointed out by the arrows of same color are meant to be connected.

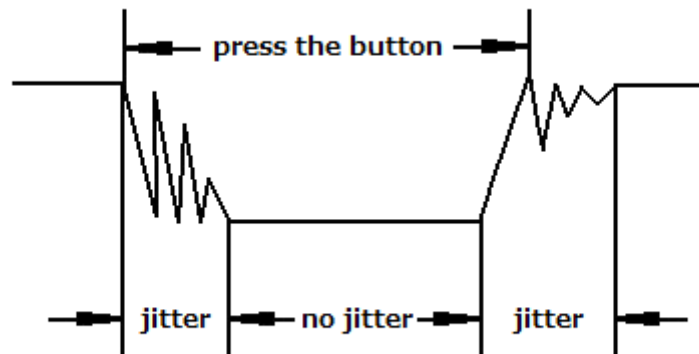


The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

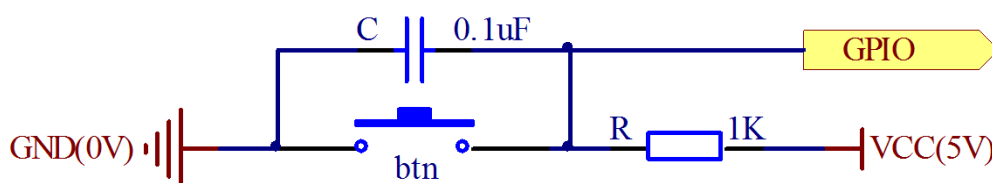
The schematic diagram we used is as follows:



The button jitter must be happen in the process of using. The jitter waveform is as the flowing picture:



Each time you press the button, the Arduino will think you have pressed the button many times due to the jitter of the button. We must to deal with the jitter of buttons before we use the button. We can through the software programming method to remove the jitter of buttons, and you can use a capacitance to remove the jitter of buttons. We introduce the software method. First, we detect whether the level of button interface is low level or high level. When the level we detected is low level, 5~10 MS delay is needed, and then detect whether the level of button interface is low or high. If the signal is low, we can confirm that the button is pressed once. You can also use a 0.1 uF capacitance to clean up the jitter of buttons. The schematic diagram is shown in below:



## 2. interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They

may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

### 3. Key functions:

#### ● `attachInterrupt(interrupt, ISR, mode)`

Specifies a named Interrupt *Service* Routine (ISR) to call when an interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).

Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be ignored (turned off) until the current one is finished. as `delay()` and `millis()` both rely on interrupts, they will not work while an ISR is running. `delayMicroseconds()`, which does not rely on interrupts, will work as expected.

#### *Syntax*

`attachInterrupt(pin, ISR, mode)`

#### *Parameters*

pin: the pin number

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

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- LOW to trigger the interrupt whenever the pin is low,
- CHANGE to trigger the interrupt whenever the pin changes value
- RISING to trigger when the pin goes from low to high,
- FALLING for when the pin goes from high to low.

#### ● `digitalRead()`

Reads the value from a specified digital pin, either HIGH or LOW.

#### *Syntax*

`digitalRead(pin)`

#### *Parameters*

pin: the number of the digital pin you want to read (int)

#### *Returns*

HIGH or LOW

#### ● `delayMicroseconds(us)`



Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use `delay()` instead.

### Syntax

`delayMicroseconds(us)`

### Parameters

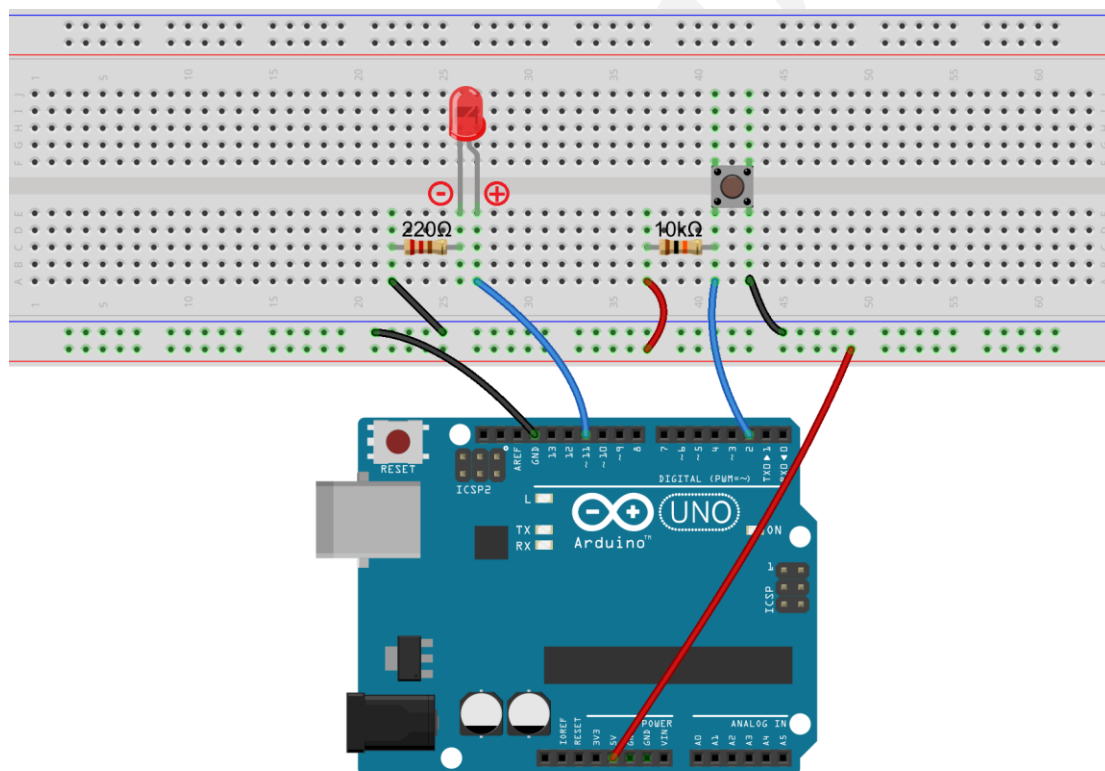
us: the number of microseconds to pause (unsigned int)

### Returns

None

## Procedures

### 1. Build the circuit

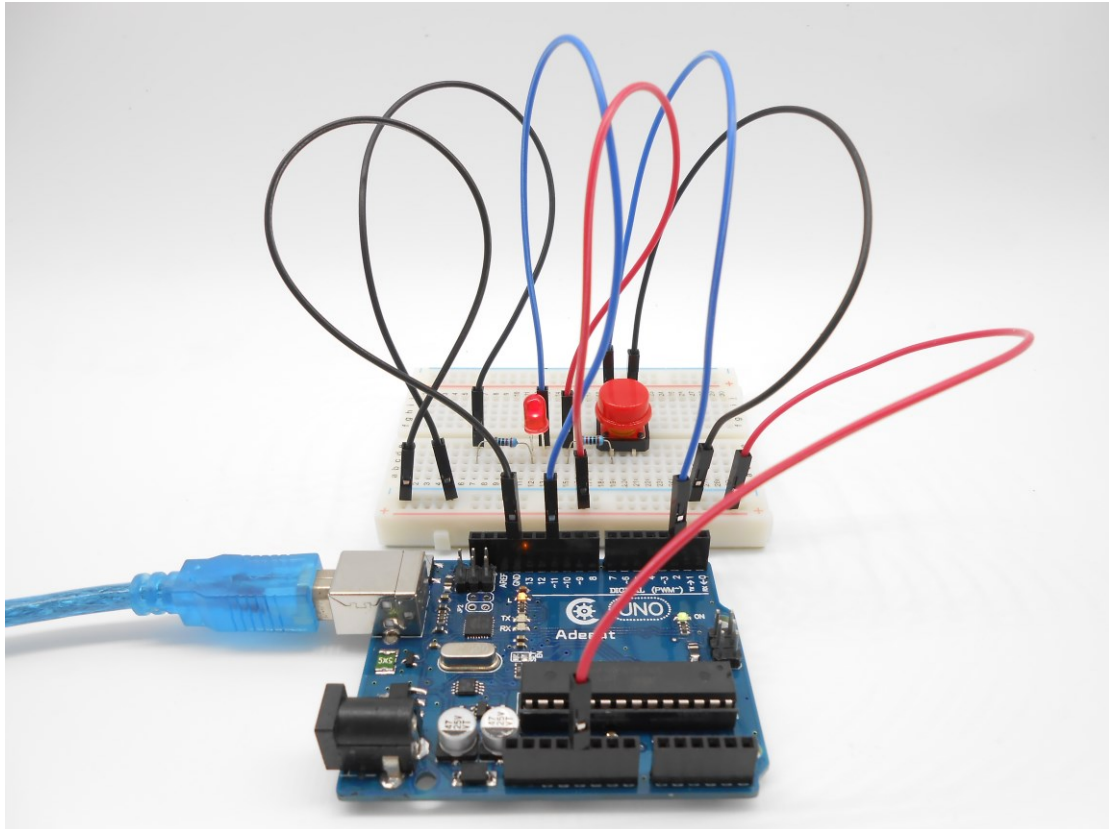


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

When you press the button, you can see the state of the LED will be toggled. (ON->OFF, OFF->ON).



## Summary

Through this lesson, you should have learned how to use the Arduino UNO detects an external button state, and then toggle the state of LED relying on the state of the button detected before.

# Lesson 4 Controlling Relay

## Overview

In this lesson, we will learn how to control a relay to cut off or connect a circuit.

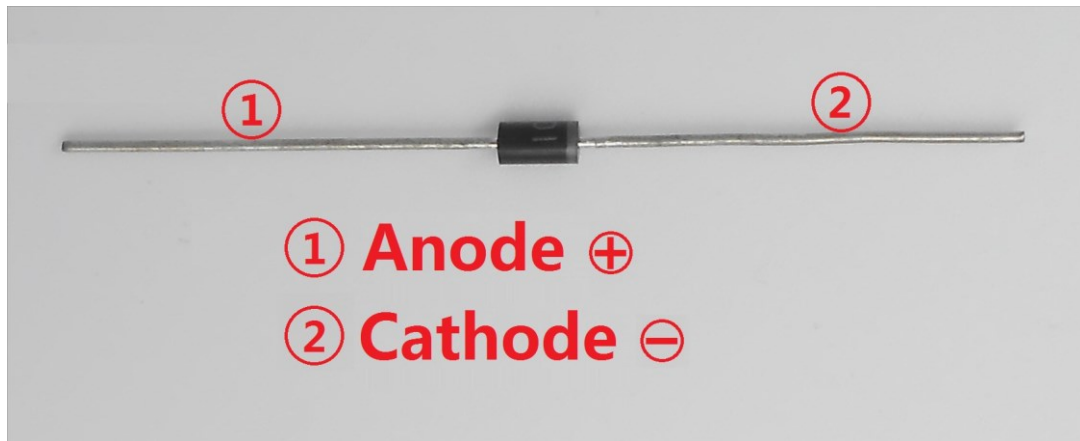
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* NPN Transistor (S8050)
- 1\* 1K Resistor
- 1\* 1N4001 Diode
- 1\* 220 $\Omega$  Resistor
- 1\* Relay
- 1\* LED
- 1\* Breadboard
- Several Jumper Wires

## Principle

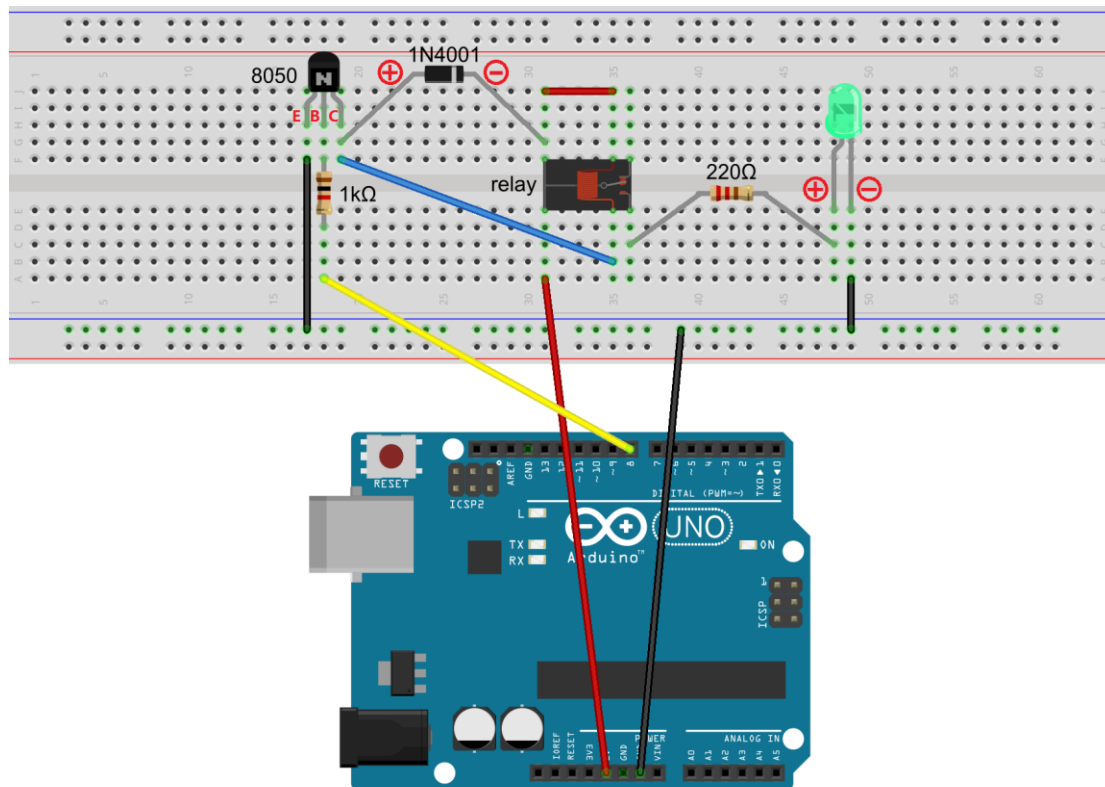
A relay is an electrically operated switch. It is generally used in automatic control circuit. Actually, it is an "automatic switch" which uses low current to control high current. It plays a role of automatic regulation, security protection and circuit switch. When an electric current is passed through the coil it generates a magnetic field that activates the armature, and the consequent movement of the movable contact(s) either makes or breaks (depending upon construction) a connection with a fixed contact. If the set of contacts was closed when the relay was de-energized, then the movement opens the contacts and breaks the connection, and vice versa if the contacts were open. When the current to the coil is switched off, the armature is returned by a force, approximately half as strong as the magnetic force, to its relaxed position. Usually this force is provided by a spring, but gravity is also used commonly in industrial motor starters. Most relays are manufactured to operate quickly. In a low-voltage application this reduces noise; in a high voltage or current application it reduces arcing.

When the coil is energized with direct current, a diode is often placed across the coil to dissipate the energy from the collapsing magnetic field at deactivation, which would otherwise generate a voltage spike dangerous to semiconductor circuit components.



## Procedures

### 1. Build the circuit

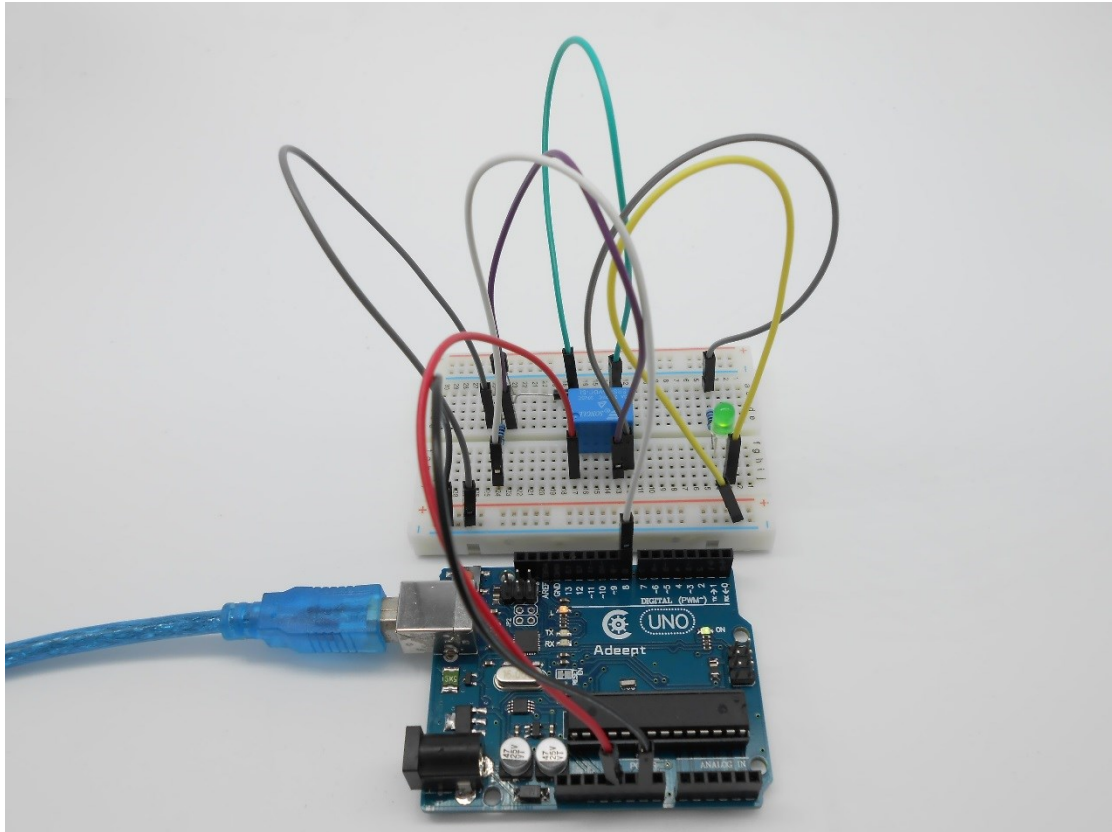


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

When the set of contacts was closed, the LED will be lit up; when the set of contacts was broke, the LED will go out.



## Summary

By learning this lesson, you have already known the basic principle of the relay, and you can also use the relay to do some creative applications.

# Lesson 5 Serial Port

## Overview

In this lesson, we will program the Arduino UNO to achieve function of send and receive data through the serial port. The Arduino receiving data which send from PC, and then controlling an LED according to the received data, then return the state of LED to the PC's serial port monitor.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LED
- 1\* 220Ω Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

### 1. Serial ports

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART). It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to `begin()`.

To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the UNO's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your UNO to your device's ground. (Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.)

### 2. Key function

#### ● `begin()`

Sets the data rate in bits per second (baud) for serial data transmission. For

communicating with the computer, use one of these rates: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

#### *Syntax*

`Serial.begin(speed)`

#### *Parameters*

speed: in bits per second (baud) - long

#### *Returns*

nothing

#### ● `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. For example:

```
Serial.print(78) gives "78"
```

```
Serial.print(1.23456) gives "1.23"
```

```
Serial.print('N') gives "N"
```

```
Serial.print("Hello world.") gives "Hello world."
```

An optional second parameter specifies the base (format) to use; permitted values are BIN (binary, or base 2), OCT (octal, or base 8), DEC (decimal, or base 10), HEX (hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example:

```
Serial.print(78, BIN) gives "1001110"
```

```
Serial.print(78, OCT) gives "116"
```

```
Serial.print(78, DEC) gives "78"
```

```
Serial.print(78, HEX) gives "4E"
```

```
Serial.println(1.23456, 0) gives "1"
```

```
Serial.println(1.23456, 2) gives "1.23"
```

```
Serial.println(1.23456, 4) gives "1.2346"
```

You can pass flash-memory based strings to `Serial.print()` by wrapping them with `F()`. For example :

```
Serial.print(F("Hello World"))
```

To send a single byte, use `Serial.write()`.

#### *Syntax*

`Serial.print(val)`



### `Serial.print(val, format)`

#### *Parameters*

val: the value to print - any data type  
format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### *Returns*

byte  
print() will return the number of bytes written, though reading that number is optional

### ● `println()`

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

#### *Syntax*

### `Serial.println(val)`

### `Serial.println(val, format)`

#### *Parameters*

val: the value to print - any data type  
format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

#### *Returns*

byte  
println() will return the number of bytes written, though reading that number is optional

### ● `read()`

Reads incoming serial data. read() inherits from the Stream utility class.

#### *Syntax*

### `Serial.read()`

#### *Parameters*

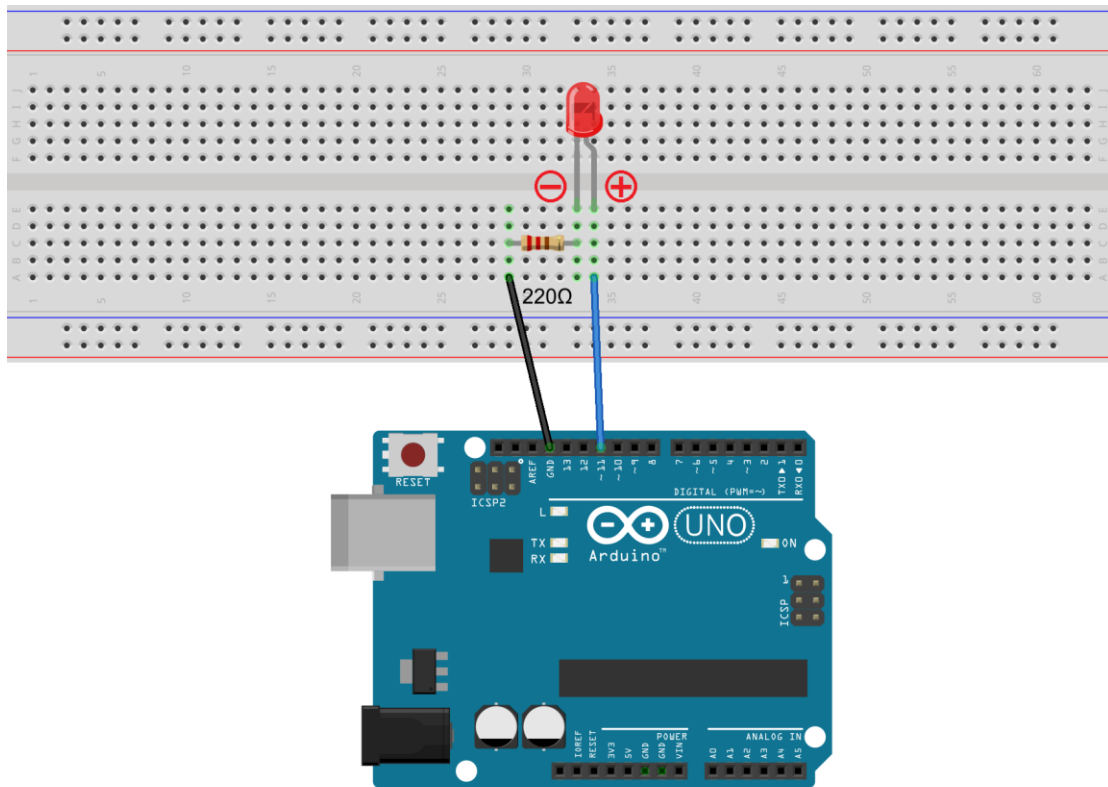
None

#### *Returns*

the first byte of incoming serial data available (or -1 if no data is available) - int

## **Procedures**

### 1. Build the circuit



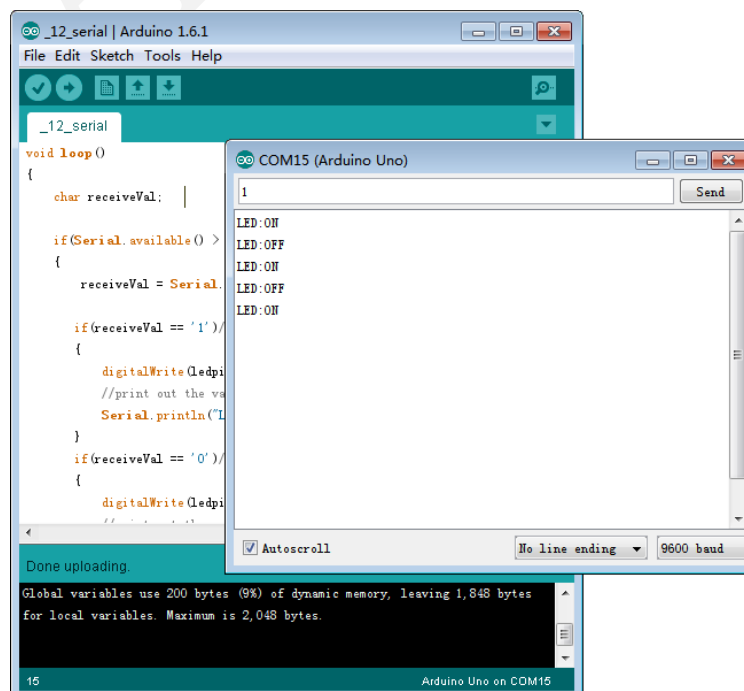
fritzing

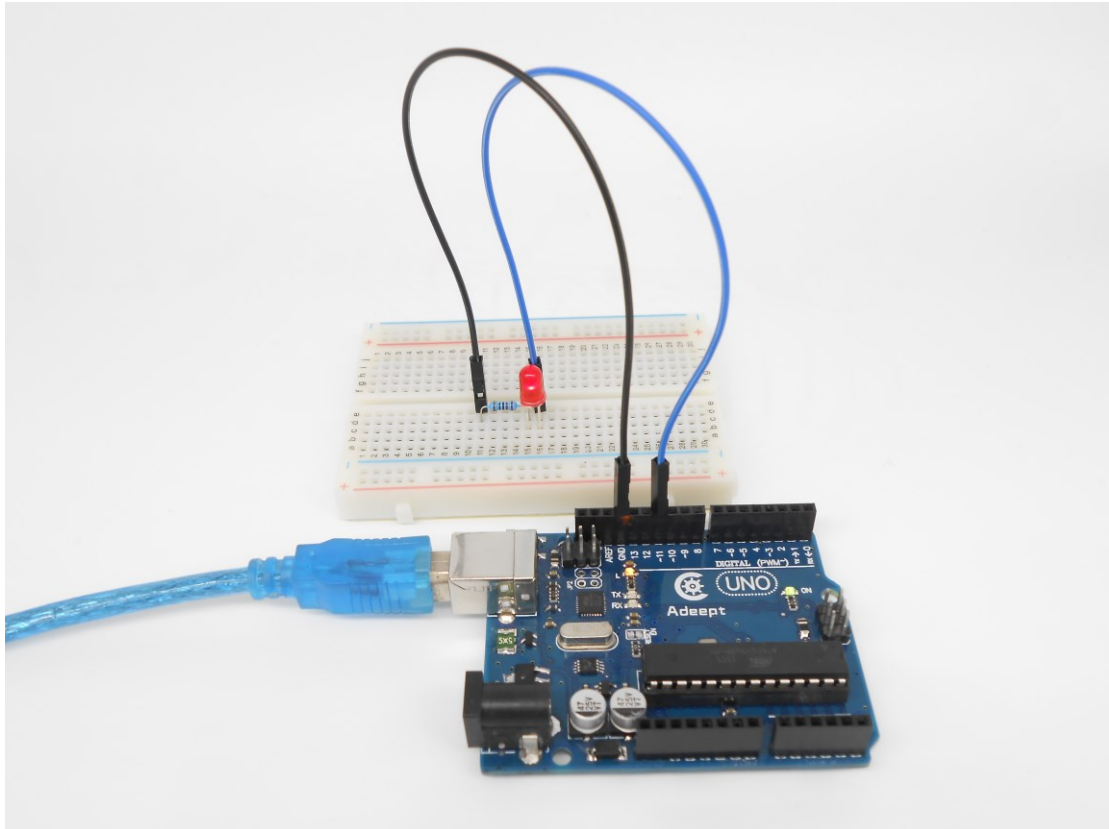
## 2. Program

## 3. Compile the program and upload to Arduino UNO board

Open the port monitor, and then select the appropriate baud rate according to the program.

Now, if you send a character '1' or '0' on the serial monitor, the state of LED will be lit or gone out.





## Summary

Through this lesson, you should have understood that the computer can send data to Arduino UNO via the serial port, and then control the state of LED. I hope you can use your head to make more interesting things based on this lesson.

# Lesson 6 LED Flowing Lights

## Overview

In the first class, we have learned how to make an LED blink by programming the Arduino. Today, we will use the Arduino to control 8 LEDs, so that 8 LEDs showing the result of flowing.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 8\* LED
- 8\* 220Ω Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

The principle of this experiment is very simple. It is very similar with the first class.

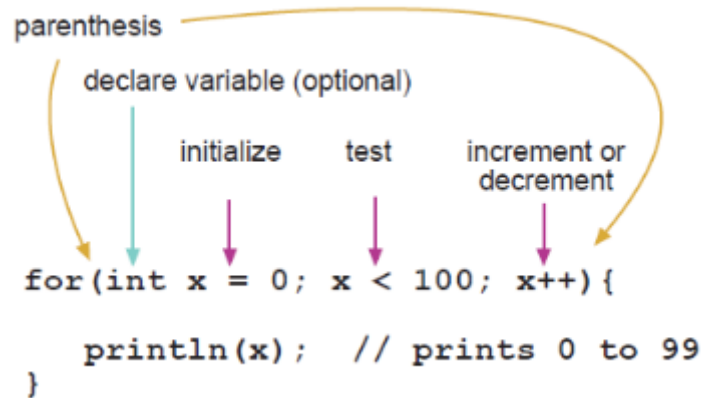
### *Key function:*

- **for statements**

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the for loop header:

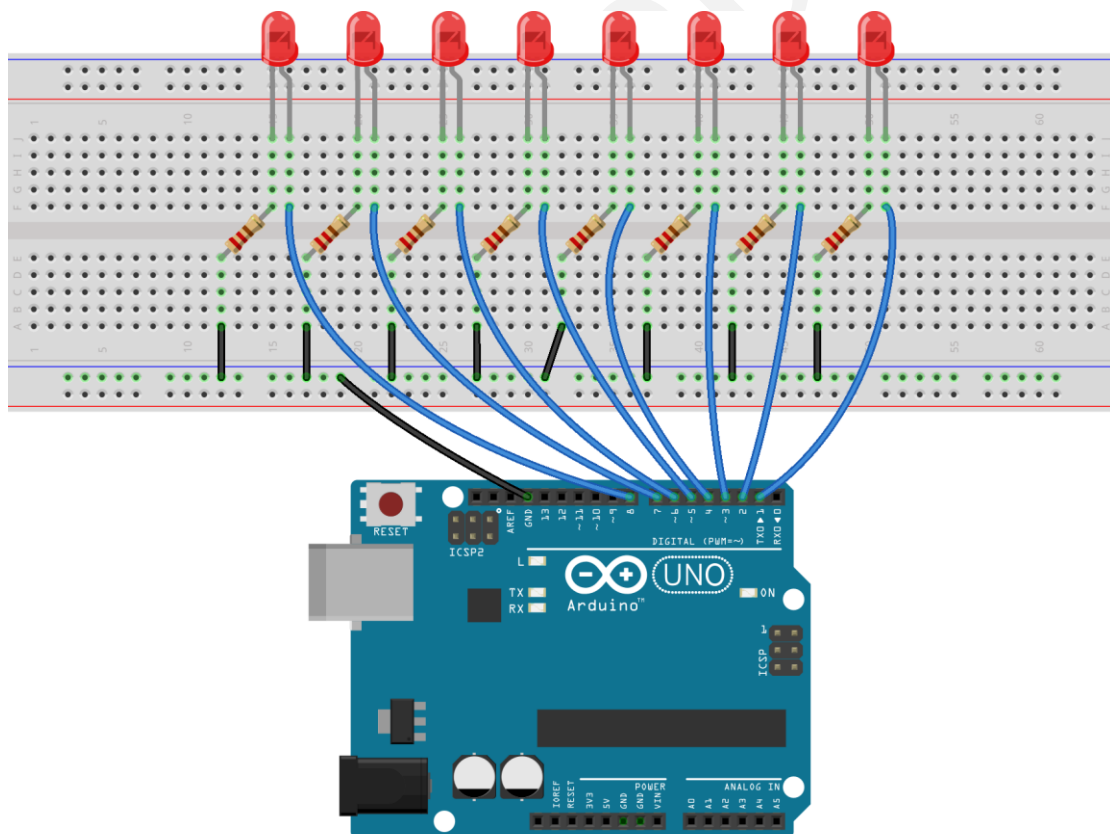
```
for (initialization; condition; increment) {  
  //statement(s);  
}
```



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.

## Procedures

### 1. Build the circuit



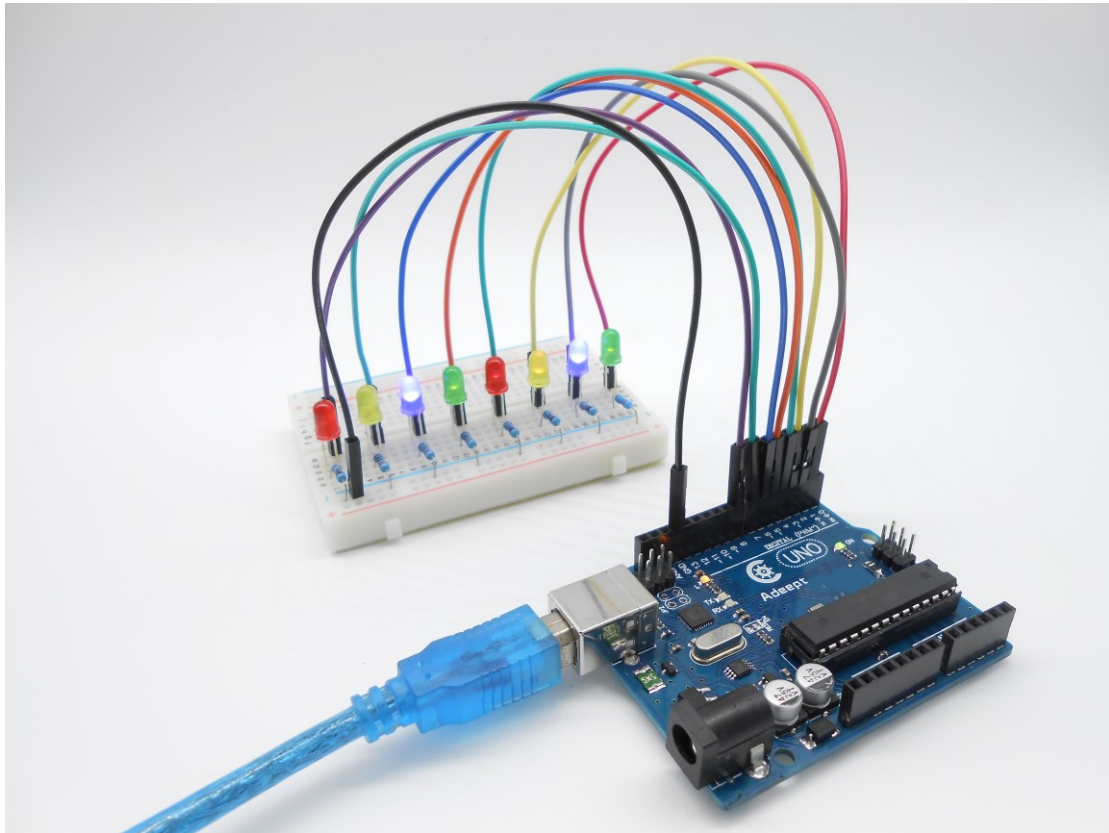
fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you should see 8 LEDs are lit in sequence from the right green one to the

left, next from the left to the right one. And then repeat the above phenomenon.



## Summary

Through this simple and fun experiment, we have learned more skilled programming about the Arduino. In addition, you can also modify the circuit and code we provided to achieve even more dazzling effect.

# Lesson 7 LED bar graph display

## Overview

In this lesson, we will learn how to control a LED bar graph by programming the Arduino.

## Requirement

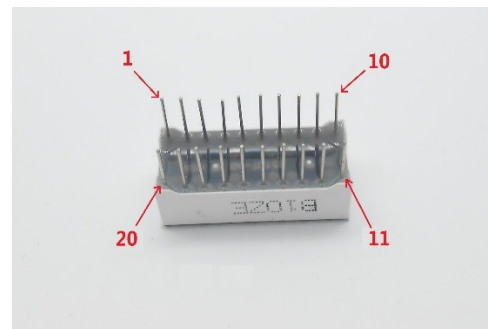
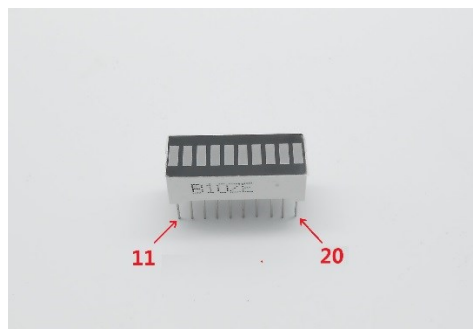
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 10K $\Omega$  Potentiometer
- 10\* 220 $\Omega$  Resistor
- 1\* LED Bar Graph
- 1\* Breadboard
- Several Jumper Wires

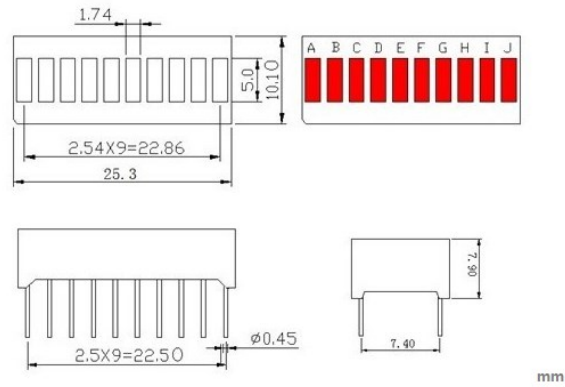
## Principle

The bar graph - a series of LEDs in a line, such as you see on an audio display is a common hardware display for analog sensors. It's made up of a series of LEDs in a row, an analog input like a potentiometer, and a little code in between. You can buy multi-LED bar graph displays fairly cheaply. This tutorial demonstrates how to control a series of LEDs in a row, but can be applied to any series of digital outputs.

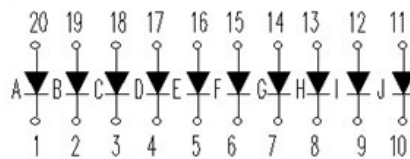
This tutorial borrows from the For Loop and Arrays tutorial as well as the Analog Input tutorial.

The sketch works like this: first you read the input. You map the input value to the output range, in this case ten LEDs. Then you set up a **for** loop to iterate over the outputs. If the output's number in the series is lower than the mapped input range, you turn it on. If not, you turn it off.





The internal schematic diagram for the LED bar graph is shown below:

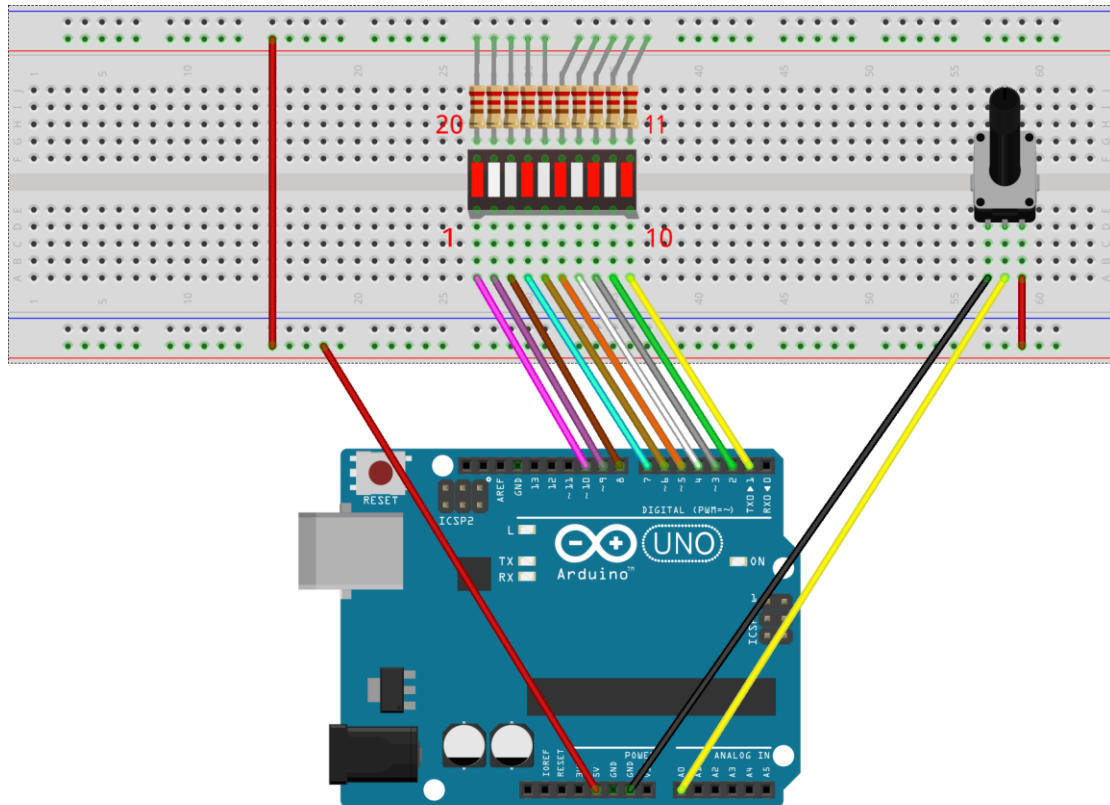


A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

## Procedures

### 1. Build the circuit



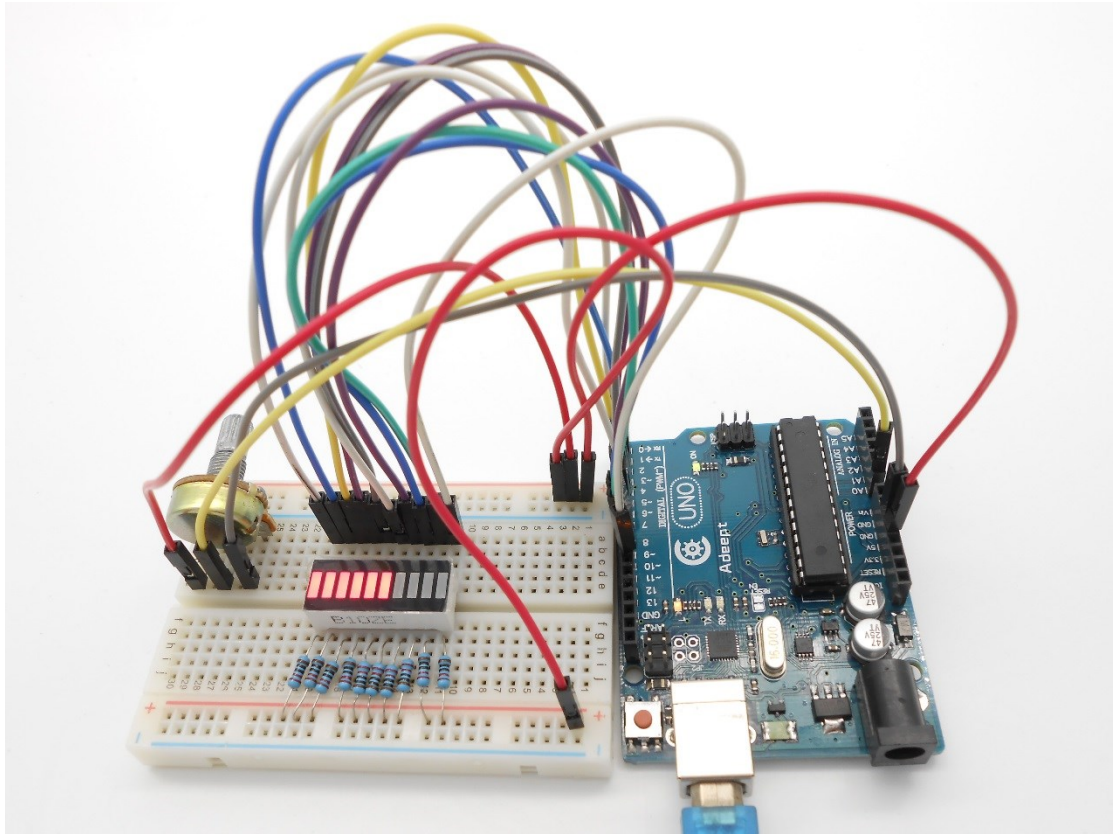


fritzing

## 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you turn the knob of the potentiometer, you will see that the number of LED in the LED bar graph will be changed.



Adept

# Lesson 8 Breathing LED

## Overview

In this lesson, we will learn how to program the Arduino to generate PWM signal. And use the PWM square-wave signal control an LED gradually becomes brighter and then gradually becomes dark like the animal's breathing.

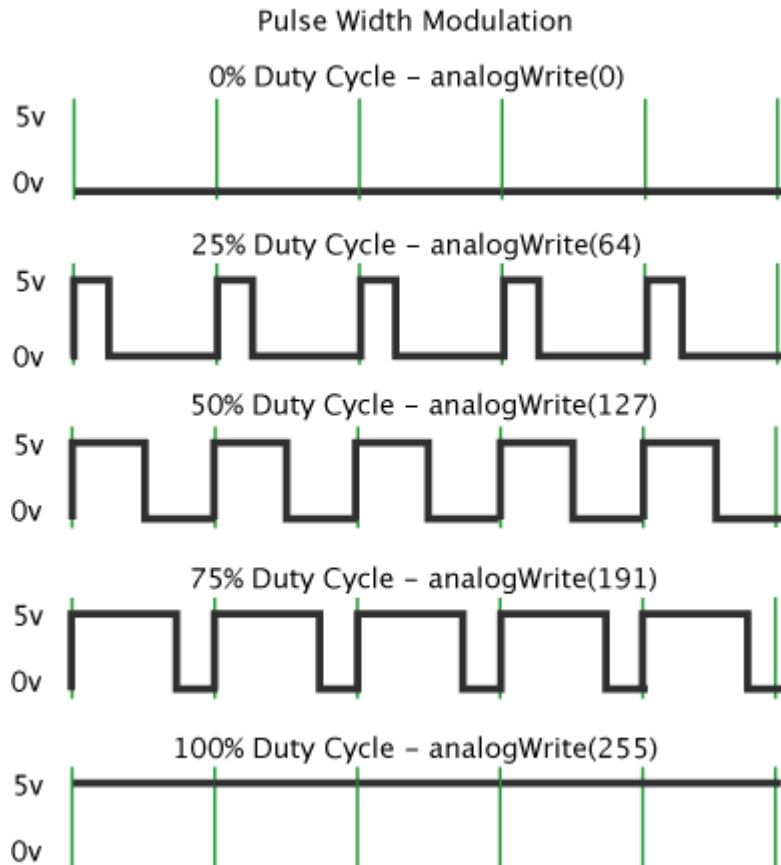
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LED
- 1\* 220Ω Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

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 the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

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.



**Key function:**

- `analogWrite()`

Writes an analog value (PWM wave) to a pin. Can be used to light an LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()` on the same pin). You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.

*Syntax*

`analogWrite(pin, value)`

*Parameters*

pin: the pin to write to.

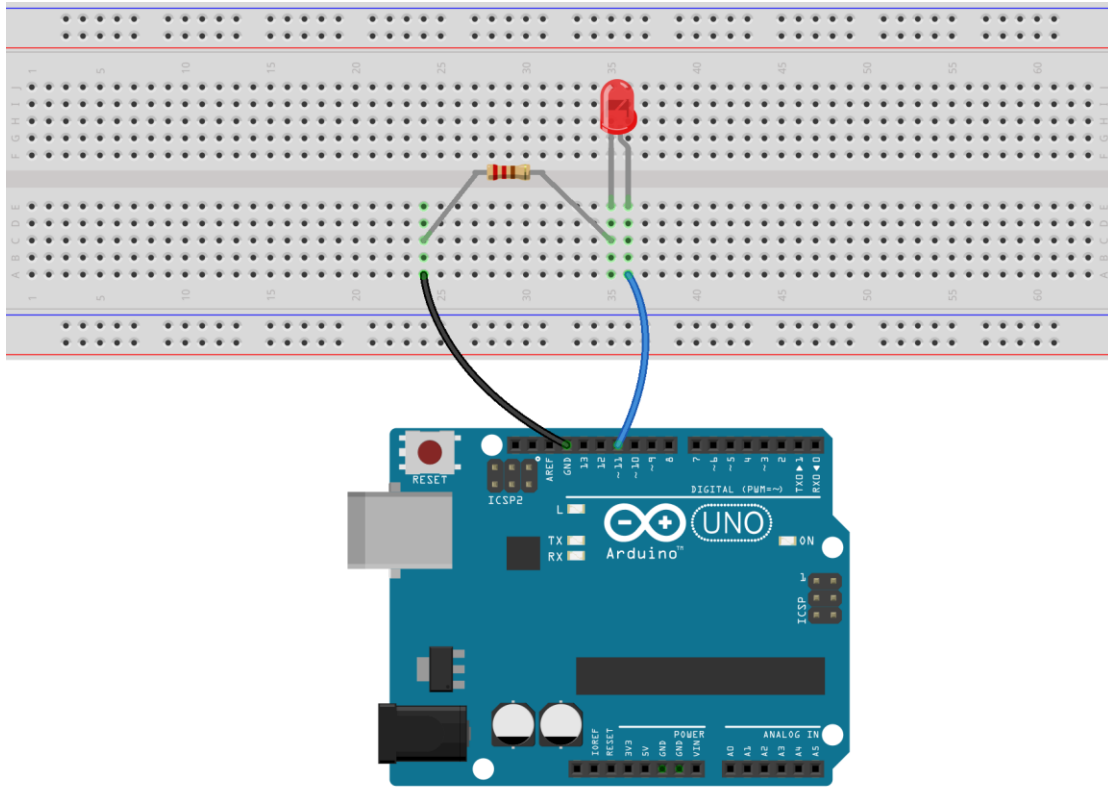
value: the duty cycle: between 0 (always off) and 255 (always on).

*Returns*

nothing

**Procedures**

1. Build the circuit

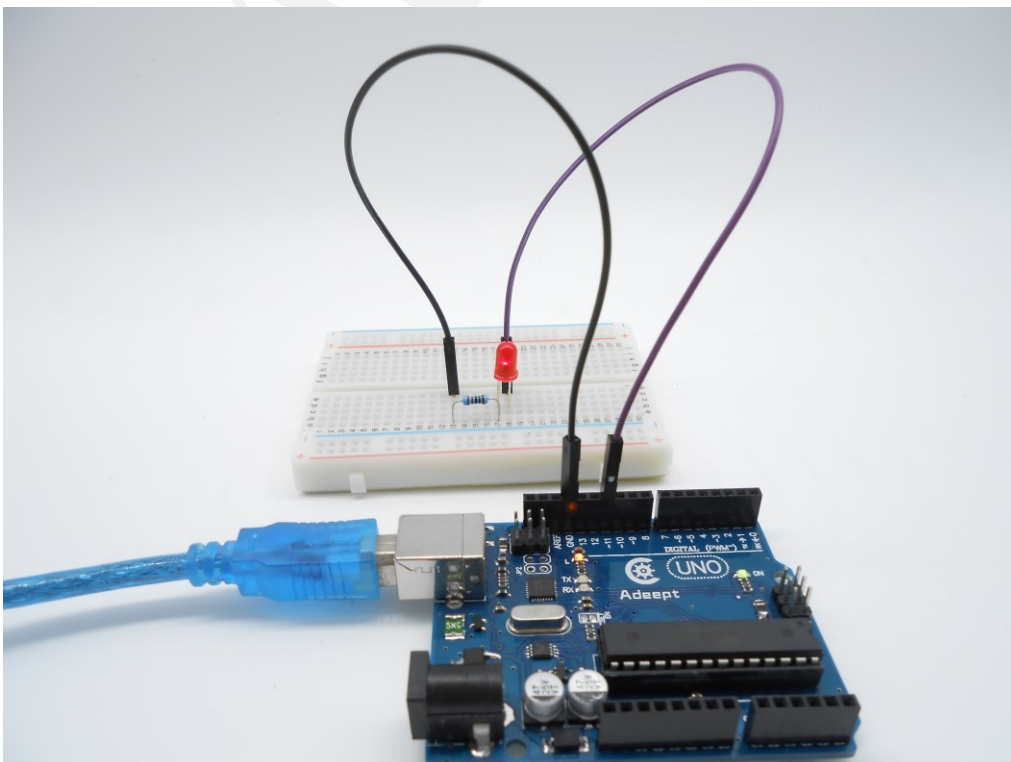


fritzing

## 2. Program

### 3. Compile the program and upload to Arduino UNO board.

Now, you should see the LED gradually from dark to brighter, and then from brighter to dark, continuing to repeat the process, its rhythm like the animal's breathing.



## **Summary**

By learning this lesson, I believe that you have understood the basic principles of the PWM, and mastered the PWM programming on the Arduino platform.

Adept

# Lesson 9 Controlling a RGB LED by PWM

## Overview

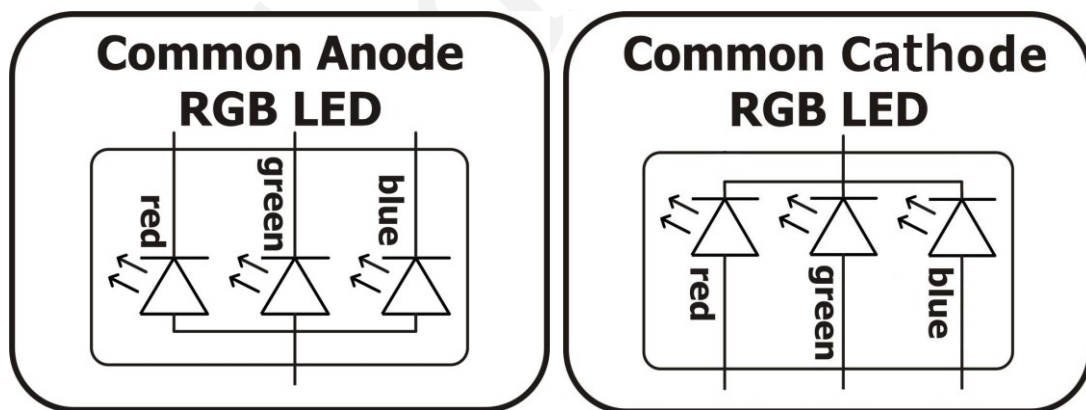
In this lesson, we will program the Arduino for RGB LED control, and make RGB LED emits a various of colors of light.

## Requirement

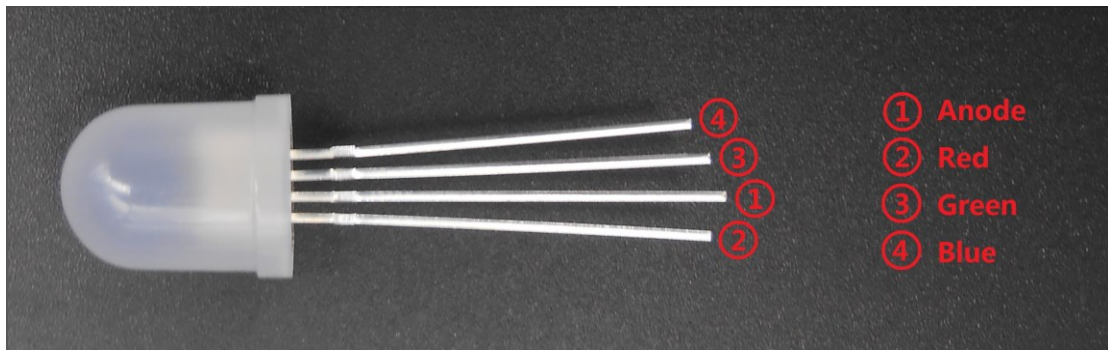
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* RGB LED
- 3\* 220 $\Omega$  Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

RGB LEDs consist of three LEDs. Each LED actually has one red, one green and one blue light. These three colored LEDs are capable of producing any color. Tri-color LEDs with red, green, and blue emitters, in general using a four-wire connection with one common lead (anode or cathode). These LEDs can have either common anode or common cathode leads.



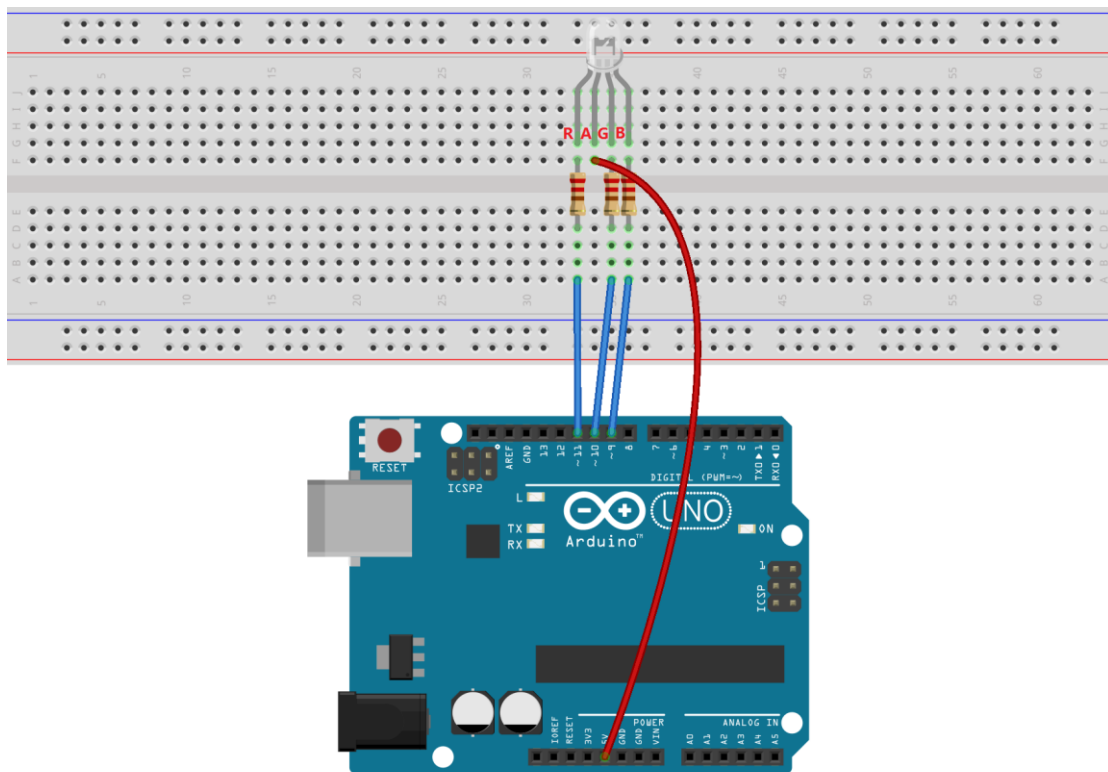
What we used in this experiment is the common anode RGB LED. The longest pin is the common anode of three LEDs. The pin is connected to the +5V pin of the Arduino, and the three remaining pins are connected to the Arduino's D9, D10, D11 pins through a current limiting resistor.



In this way, we can control the color of RGB LED by 3-channel PWM signal.

## Procedures

### 1. Build the circuit



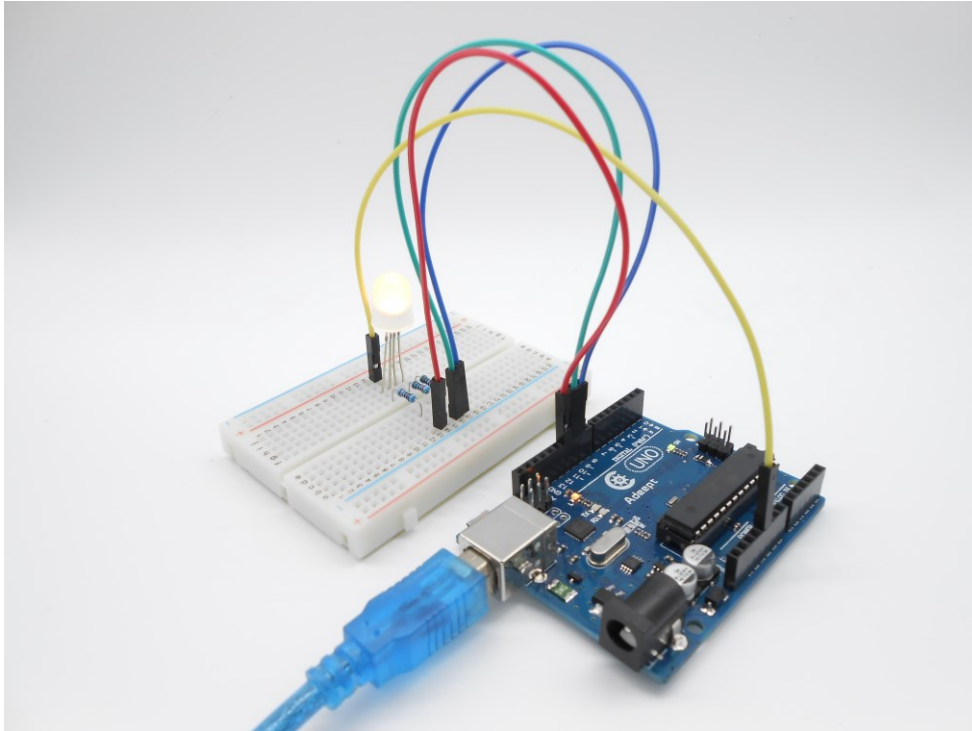
fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you can see the RGB LED emitting red, green, blue, yellow, white and purple light, then the RGB LED will be off, each state continues 1s, after repeating the above procedure.





## Summary

By learning this lesson, I believe you have already known the principle and the programming of RGB LED. I hope you can use your imagination to achieve even more cool ideas based on this lesson.

# Lesson 10 Play the Music

## Overview

In this lesson, we will program the Arduino to control a passive buzzer, and then make the passive buzzer play music.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* NPN Transistor (8050)
- 1\* 1K $\Omega$  Resistor
- 1\* Passive Buzzer
- 1\* LED
- 1\* 220 $\Omega$  Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

As long as you send the square wave signals to a passive buzzer with different frequency, then the passive buzzer will make different sound.



### *Key function:*

#### ● `tone()`

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

Only one tone can be generated at a time. If a tone is already playing on a

different pin, the call to `tone()` will have no effect. If the tone is playing on the same pin, the call will set its frequency.

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

**NOTE:** if you want to play different pitches on multiple pins, you need to call `noTone()` on one pin before calling `tone()` on the next pin.

*Syntax*

`tone(pin, frequency)`

`tone(pin, frequency, duration)`

*Parameters*

pin: the pin on which to generate the tone

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

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

*Returns*

nothing

● `noTone()`

Stops the generation of a square wave triggered by `tone()`. Has no effect if no tone is being generated.

**NOTE:** if you want to play different pitches on multiple pins, you need to call `noTone()` on one pin before calling `tone()` on the next pin.

*Syntax*

`noTone(pin)`

*Parameters*

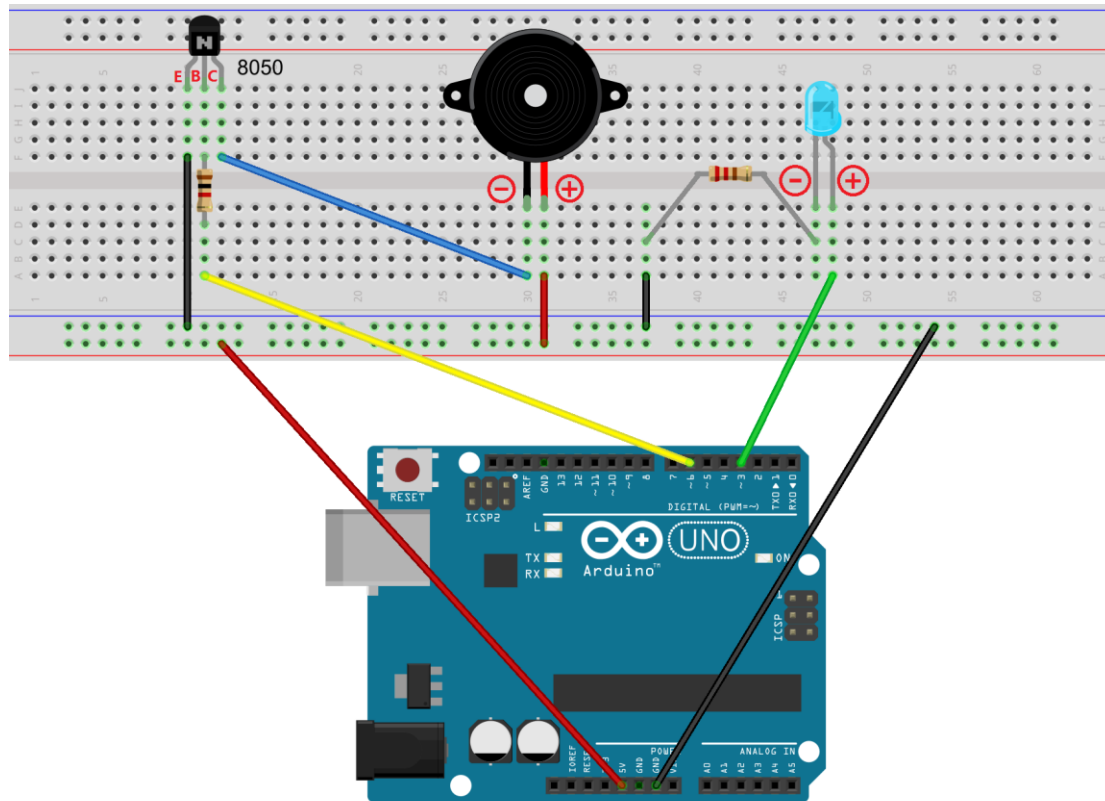
pin: the pin on which to stop generating the tone

*Returns*

nothing

## Procedures

### 1. Build the circuit

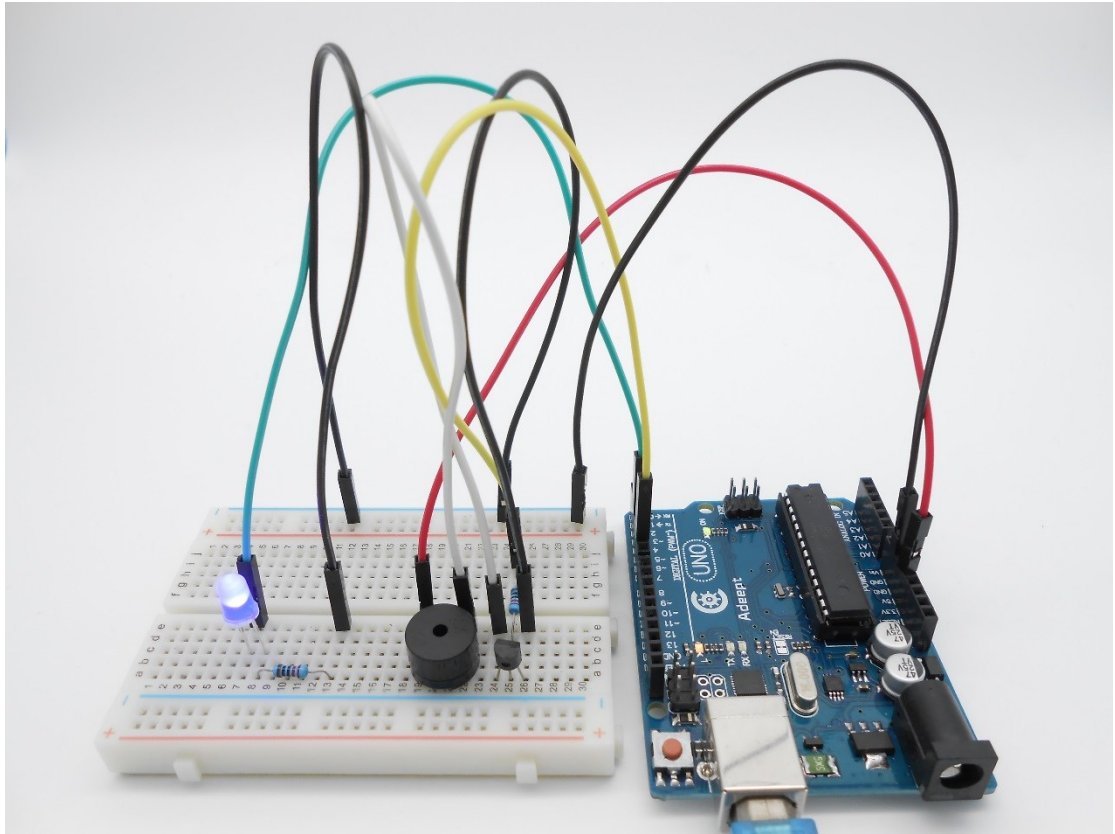


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you can hear the music of passive buzzer with blinking an LED.



Adept

# Lesson 11 LCD1602 display

## Overview

In this lesson, we will learn how to use a character display device—LCD1602 on the Arduino platform. First, we make the LCD1602 display a string "Hello Geeks!" scrolling, then display "Adept" and "www.adept.com" static.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- 1\* Breadboard
- Several Jumper Wires

## Principle

LCD1602 is a kind of character LCD display. The LCD has a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

- A register select (RS) pin that controls where in the LCD's memory you're 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.
- A Read/Write (R/W) pin that selects reading mode or writing mode
- An Enable pin that enables writing to the registers
- 8 data pins (D0-D7). The state of these pins (high or low) are the bits that you're writing to a register when you write, or the values when you read.
- There's also a display contrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and Bklt-) pins that you can use to power the LCD, control the display contrast, and turn on or off the LED backlight respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit.

The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

A potentiometer , informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

**Key function:**

● **begin()**

Specifies the dimensions (width and height) of the display.

*Syntax*

`lcd.begin(cols, rows)`

*Parameters*

lcd: a variable of type LiquidCrystal

cols: the number of columns that the display has

rows: the number of rows that the display has

● **setCursor()**

Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.

*Syntax*

`lcd.setCursor(col, row)`

*Parameters*

lcd: a variable of type LiquidCrystal

col: the column at which to position the cursor (with 0 being the first column)

row: the row at which to position the cursor (with 0 being the first row)

● **scrollDisplayLeft()**

Scrolls the contents of the display (text and cursor) one space to the left.

*Syntax*

`lcd.scrollDisplayLeft()`

*Parameters*

lcd: a variable of type LiquidCrystal

Example

`scrollDisplayLeft()` and `scrollDisplayRight()`

See also

`scrollDisplayRight()`

● **print()**

Prints text to the LCD.

## Syntax

`lcd.print(data)`

`lcd.print(data, BASE)`

## Parameters

lcd: a variable of type LiquidCrystal

data: the data to print (char, byte, int, long, or string)

BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

## Returns

byte

print() will return the number of bytes written, though reading that number is optional

### ● `clear()`

Clears the LCD screen and positions the cursor in the upper-left corner.

## Syntax

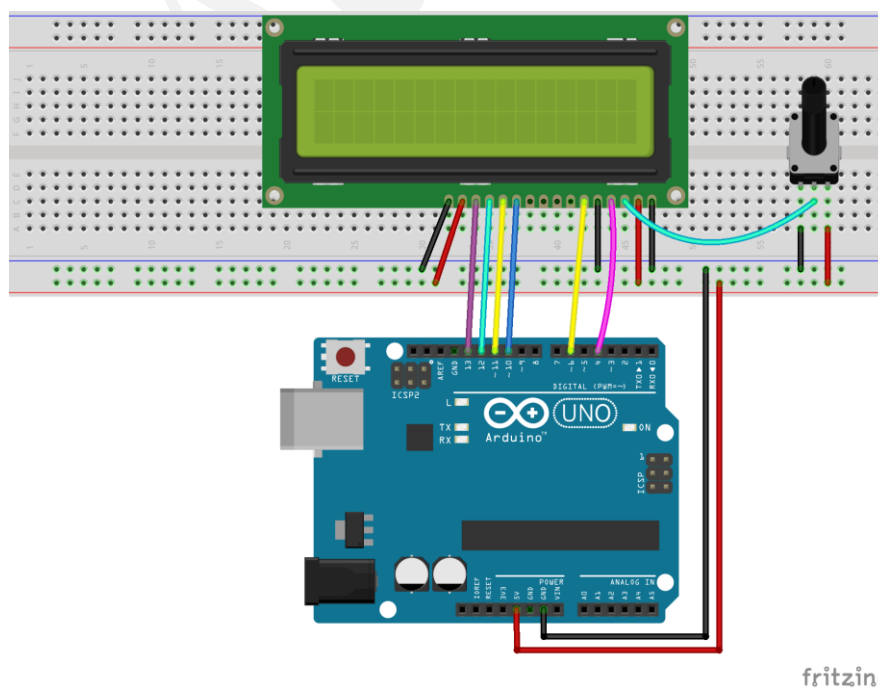
`lcd.clear()`

## Parameters

lcd: a variable of type LiquidCrystal

## Procedures

### 1. Build the circuit

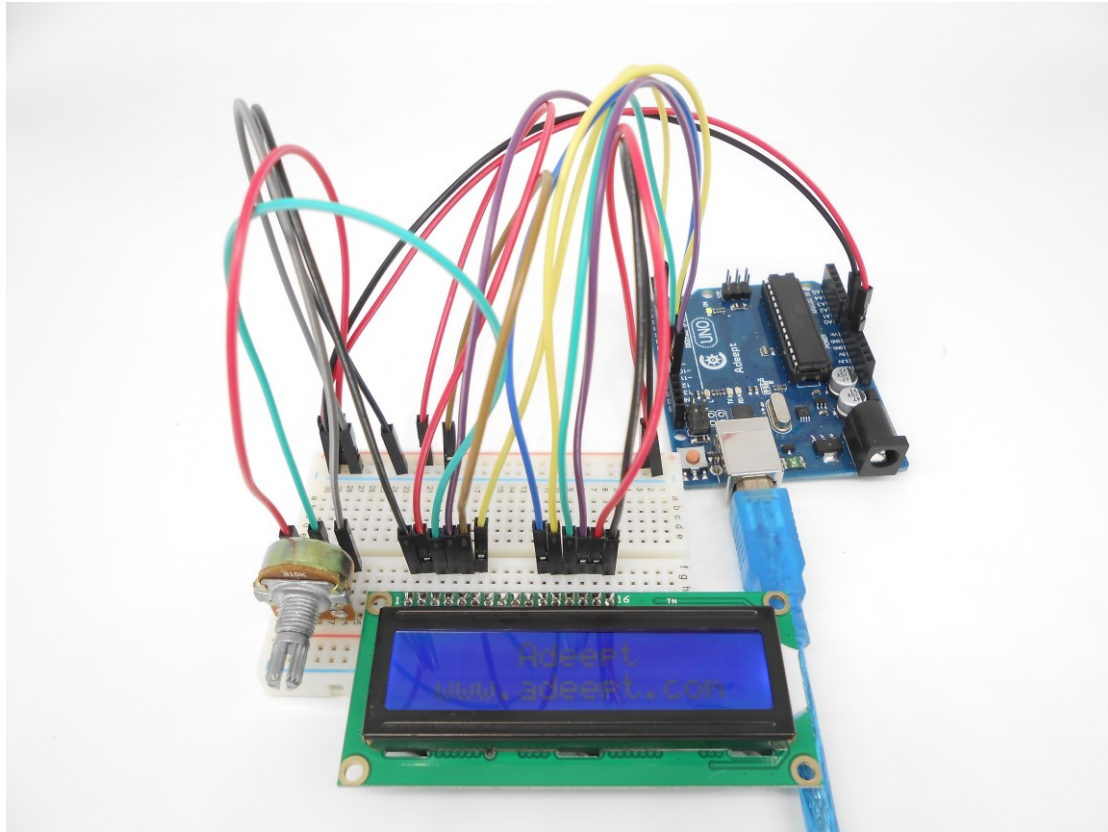


### 2. Program



### 3. Compile the program and upload to Arduino UNO board

Now, you can see the string "Hello Geeks!" is shown on the LCD1602 scrolling, and then the string "Adeept" and "www.adeept.com" is displayed on the LCD1602 static.



### Summary

I believe that you have already mastered the driver of LCD1602 through this lesson. I hope you can make something more interesting base on this lesson and the previous lesson learned.

# Lesson 12 A Simple Voltmeter

## Overview

In this lesson, we will make a simple voltmeter with Arduino UNO and LCD1602, the range of this voltmeter is 0~5V. Then, we will measure the voltage of the potentiometer's adjustment end with the simple voltmeter and display it on the LCD1602.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 2\* Potentiometer
- 1\* Breadboard
- Several Jumper Wires

## Principle

The basic principle of this experiment: Converting the analog voltage that the Arduino collected to digital quantity by the ADC(analog-to-digital converter) through programming, then display the voltage on the LCD1602.

Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from analog input 0 to the middle pin of the potentiometer. The third goes from 5 volts to the other outer pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input.

The Arduino has a circuit inside called an analog-to-digital converter that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the

opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, `analogRead( )` returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

### Key functions:

- `analogRead( )`

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit. The input range and resolution can be changed using `analogReference( )`.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

### Syntax

`analogRead(pin)`

### Parameters

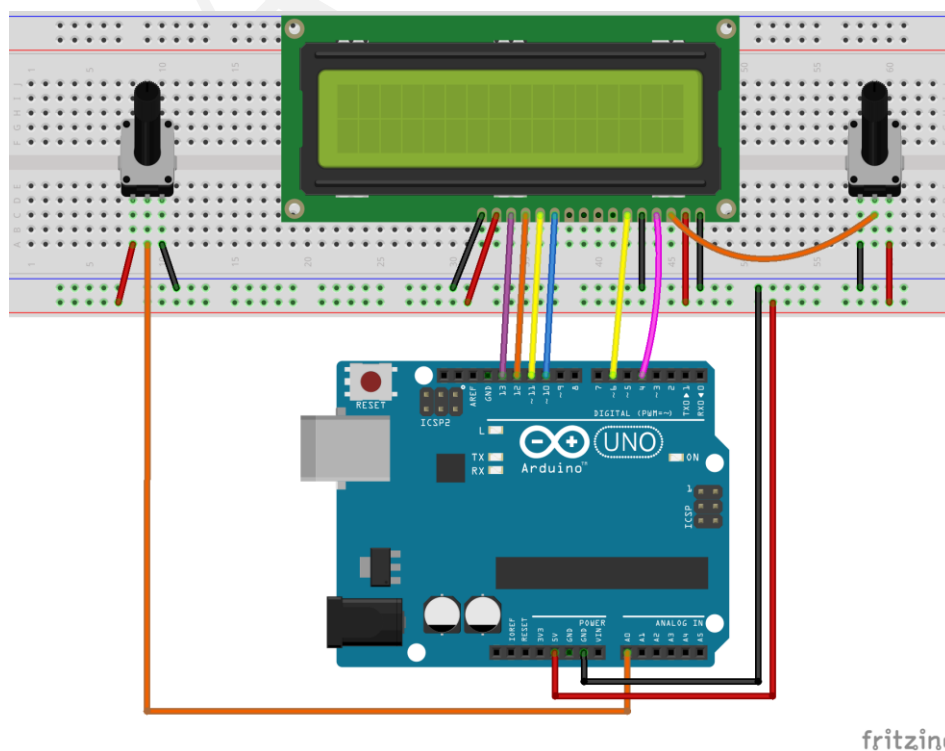
pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

### Returns

int (0 to 1023)

## Procedures

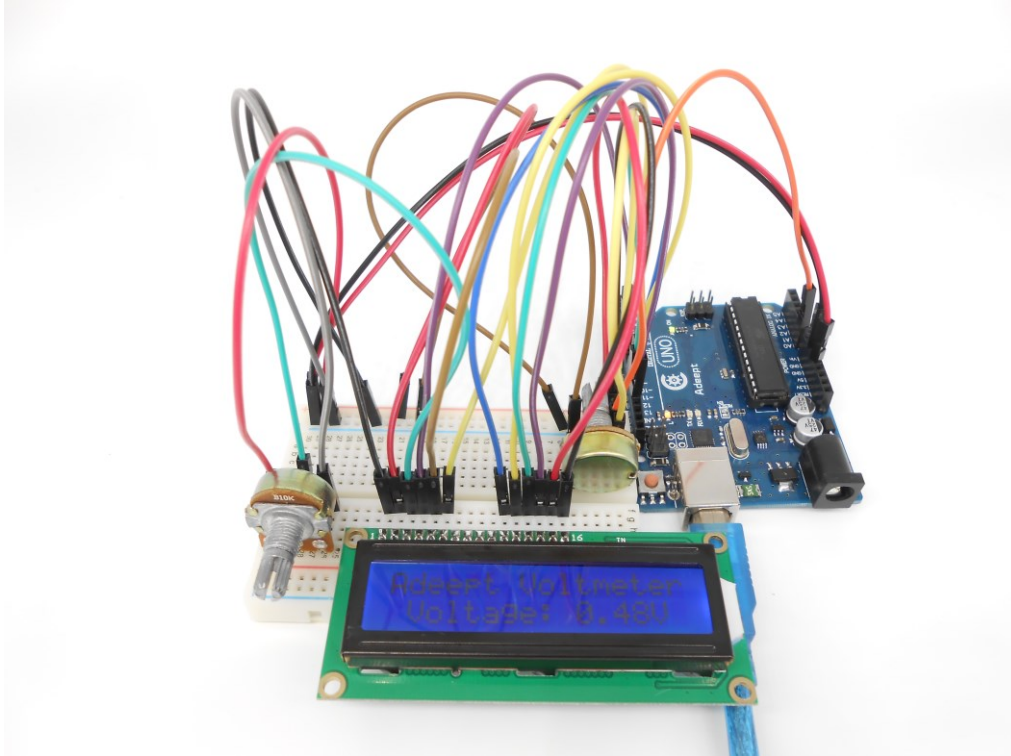
### 1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO.

Now, when you turning the shaft of the potentiometer, you will see the voltage displayed on the LCD1602 will be changed.



## Summary

The substance of voltmeter is reading analog voltage which input to ADC inside. Through this course, I believe that you have mastered how to read analog value and how to make a simple voltmeter with Arduino.

# Lesson 13 7-segment display

## Overview

In this lesson, we will program the Arduino to achieve the controlling of segment display.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 220Ω Resistor
- 1\* 7-segment Display
- 1\* Breadboard
- Several Jumper Wires

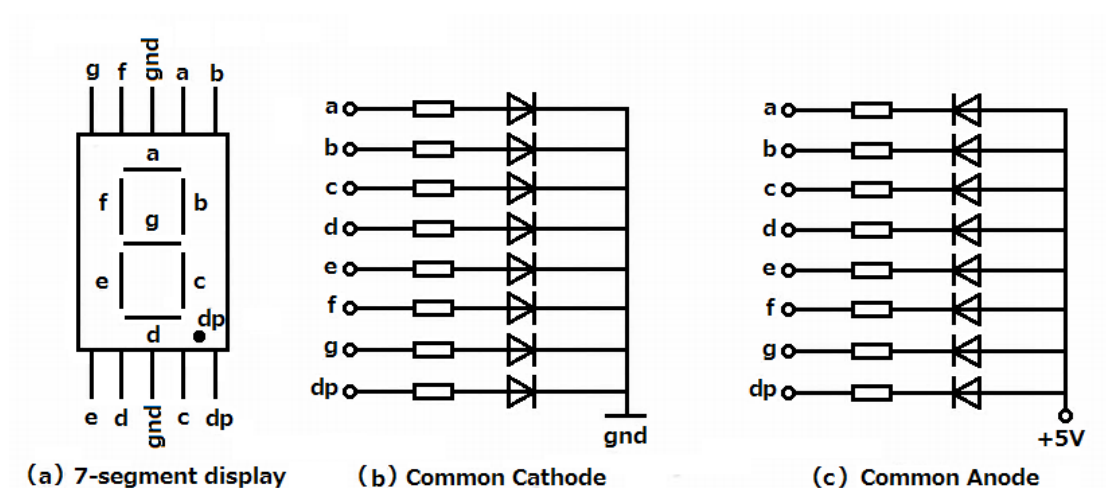
## Principle

The seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

The seven-segment display is an 8-shaped LED display device composed of eight LEDs (including a decimal point), these segments respectively named a, b, c, d, e, f, g, dp.

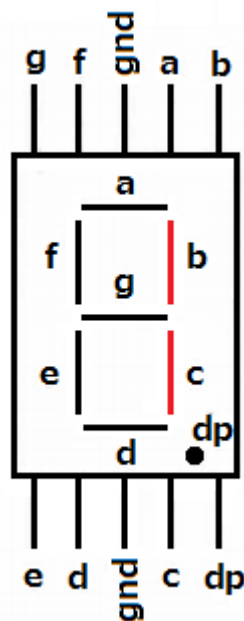
The segment display can be divided into common anode and common cathode segment display by internal connections.



When using a common anode LED, the common anode should to be connected to the power supply (VCC); when using a common cathode LED, the common cathode should be connected to the ground (GND).

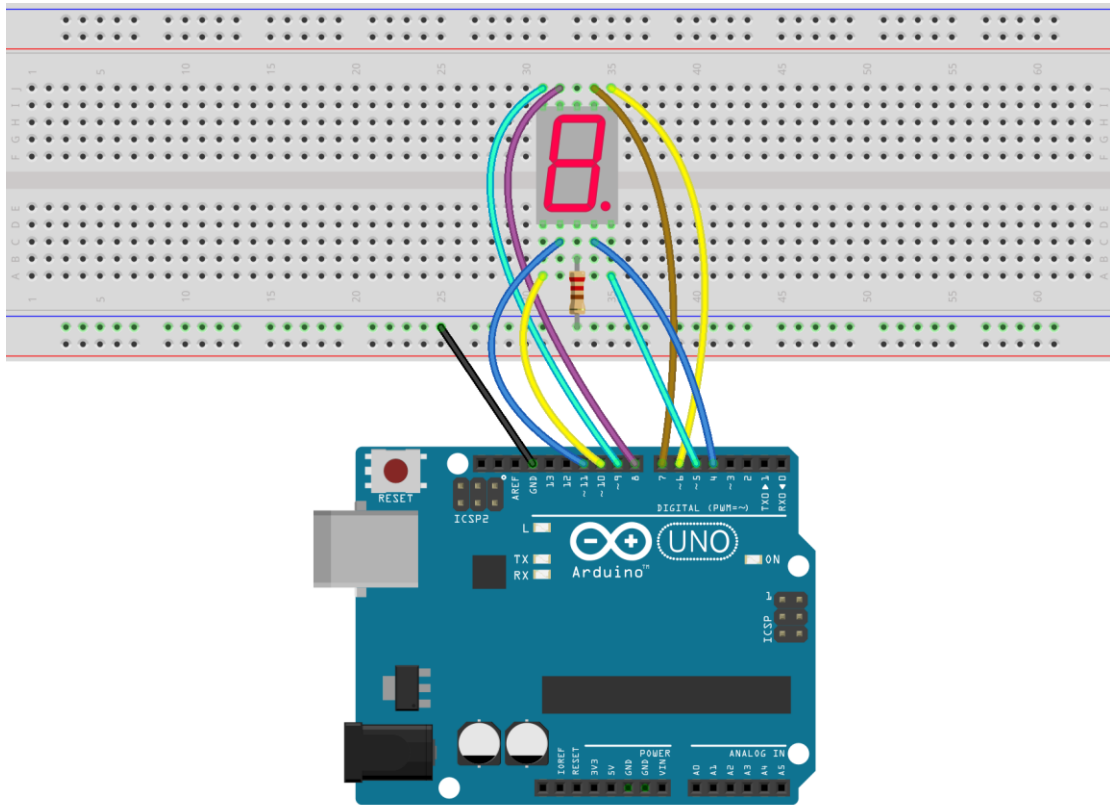
Each segment of a segment display is composed of LED, so a resistor is needed for protecting the LED.

A 7-segment display has seven segments for displaying a figure and a segment for displaying a decimal point. If you want to display a number '1', you should only light the segment b and c.



## Procedures

1. Build the circuit

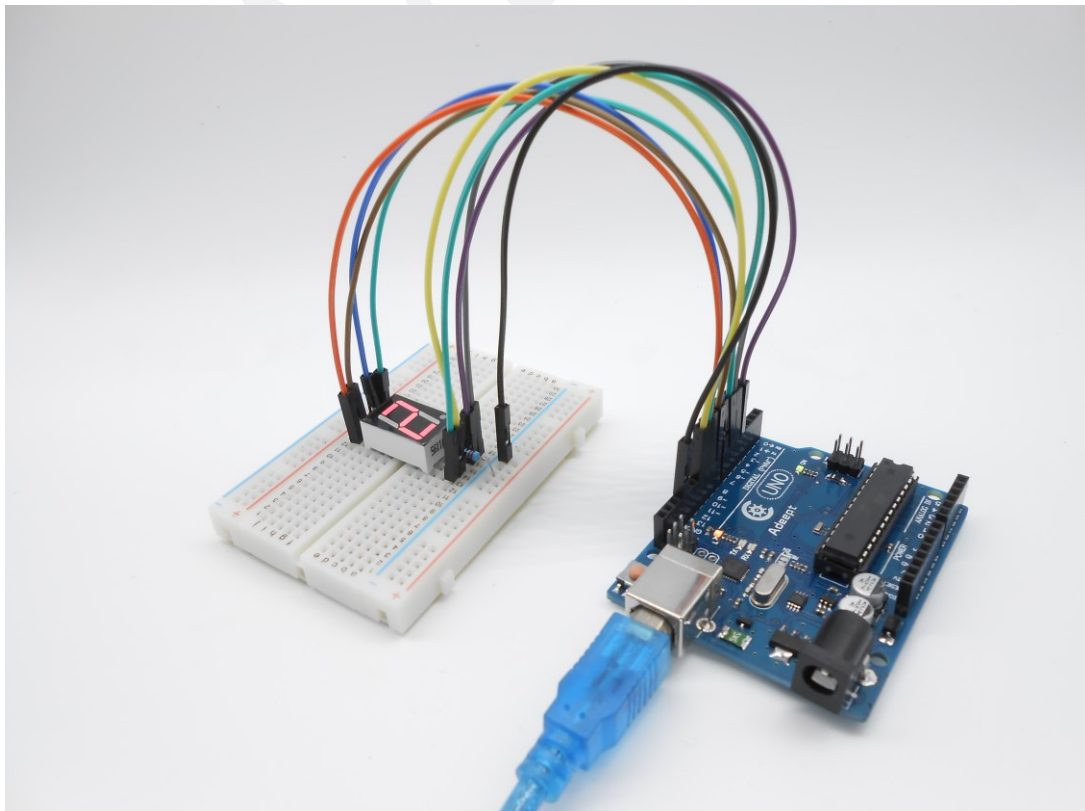


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

Now, you should see the number 0~9 are displayed on the segment display.



## Summary

Through this lesson, we have learned the principle and programming of segment display. I hope you can combine the former course to modify the code we provided in this lesson to achieve cooler originality.

Adept



# Lesson 14 A simple counter

## Overview

In this lesson, we will program the Arduino UNO to make a simple counter.

## Requirement

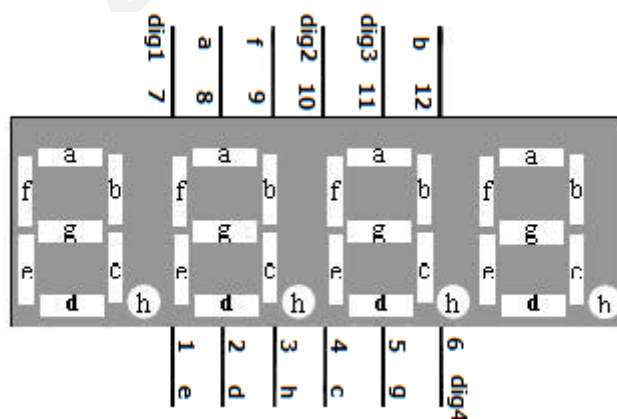
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 4-digit 7-segment Display
- 8\* 220Ω Resistor
- 2\* Button
- 1\* Breadboard
- Several Jumper Wires

## Principle

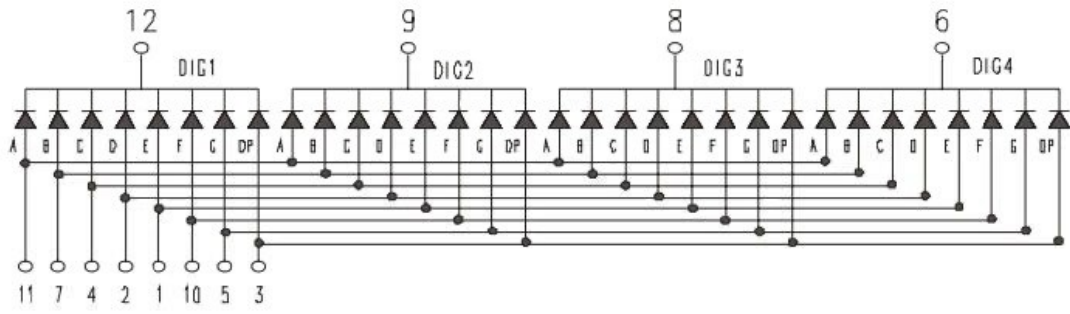
The 4-digit segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

4-digit segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

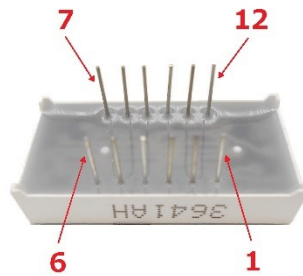
The 4-digit segment display is an 4\*8-shaped LED display device composed of 32 LEDs (including four decimal points), these segments respectively named a, b, c, d, e, f, g, h, dig1, dig2, dig3, dig4.



What we used in this experiment is a common cathode 4-digit 7-segment display. Its internal structure is shown below:

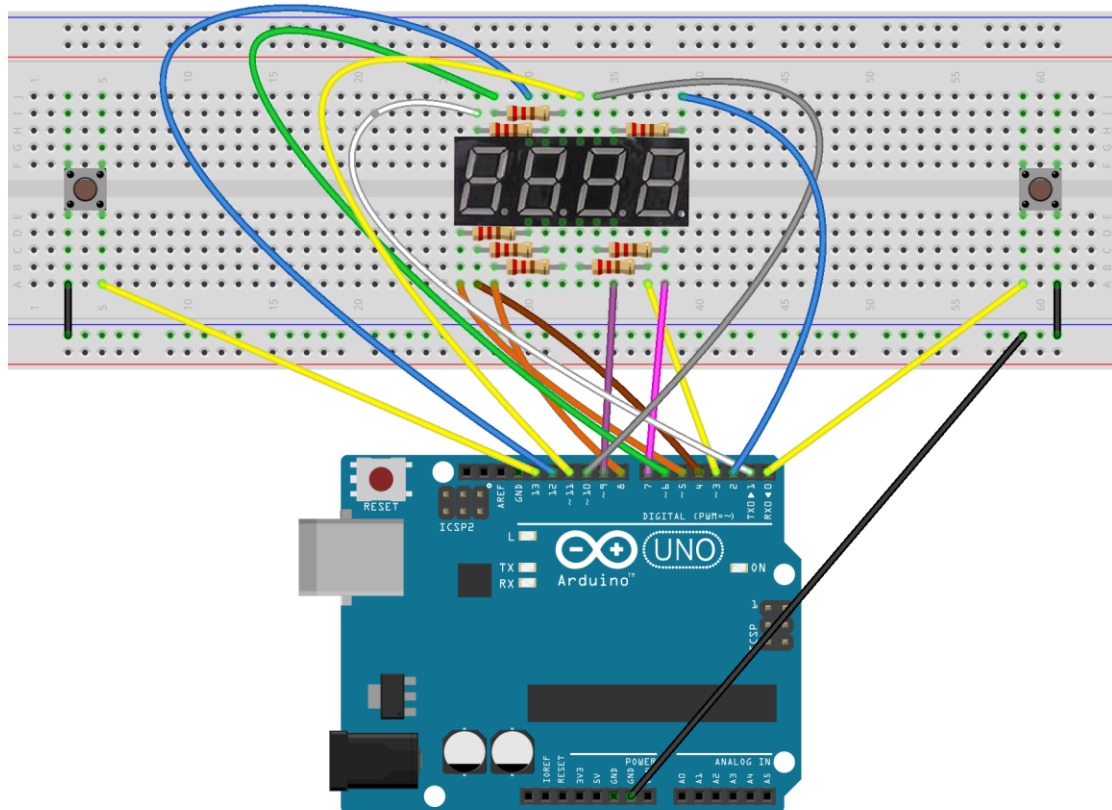


The pin number is showing below:



## Procedures

### 1. Build the circuit

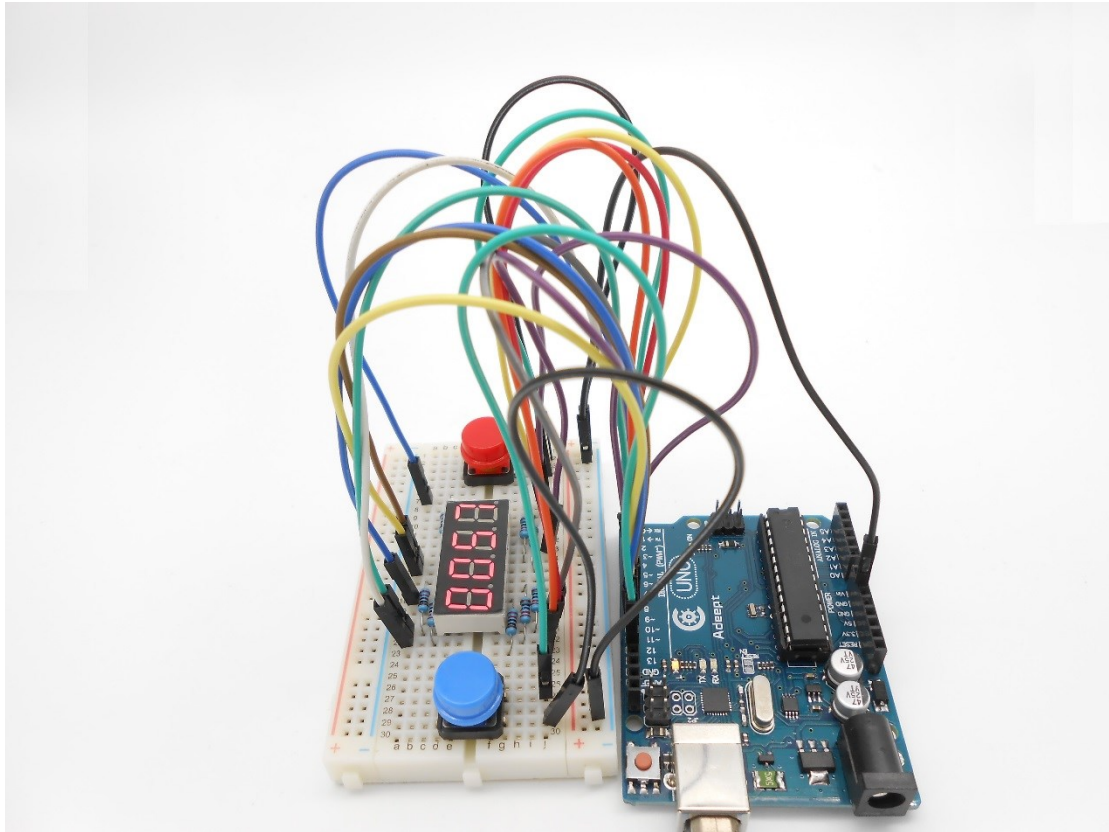


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you press one of the two buttons, the value displayed on the 4-digit 7-segment display will be changed.



### Summary

By learning this lesson, you'll find that it is so easy to make a simple counter.

# Lesson 15 Controlling Servo motor

## Overview

In this lesson, we will introduce a new electronic device (Servo) to you, and tell you how to control it with the Arduino UNO.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Servo
- Several Jumper Wires

## Principle

### 1. Servo motor

The servo motor have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note the servo motor draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together.

### 2. Servo library

This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

### 3. Key functions:

- `attach()`

Attach the Servo variable to a pin. Note that in Arduino 0016 and earlier, the Servo library supports only servos on only two pins: 9 and 10.

### Syntax

`servo.attach(pin)`

`servo.attach(pin, min, max)`

### Parameters

servo: a variable of type Servo

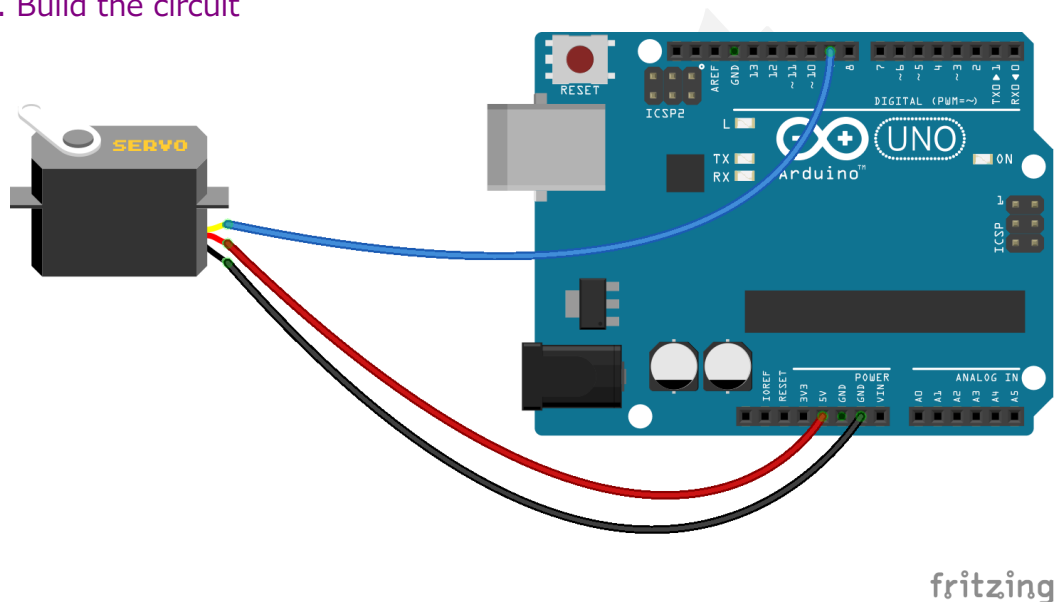
pin: the number of the pin that the servo is attached to

min (optional): the pulse width, in microseconds, corresponding to the minimum (0-degree) angle on the servo (defaults to 544)

max (optional): the pulse width, in microseconds, corresponding to the maximum (180-degree) angle on the servo (defaults to 2400)

### Procedures

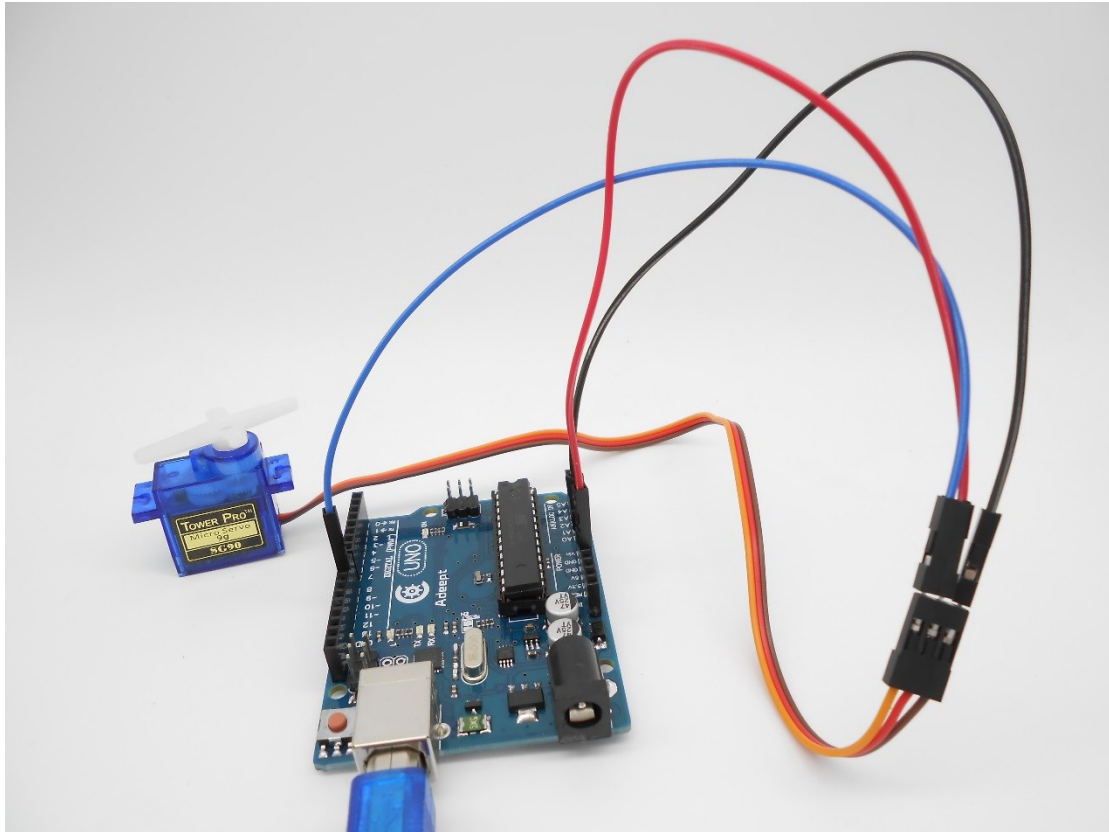
#### 1. Build the circuit



#### 2. Program

#### 3. Compile the program and upload to Arduino UNO board

Now, you should see the servo rotate 180 degrees, and then rotate in opposite direction.



## Summary

By learning this lesson, you should have known that the Arduino provided a servo library to control a servo. By using the servo library, you can easily control a servo. Just enjoy your imagination and make some interesting applications.

# Lesson 16 Using a thermistor to measure the temperature

## Overview

In this lesson, we will learn how to use a thermistor to collect temperature by programming Arduino. The information which a thermistor collects temperature is displayed on the LCD1602.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* 10KΩ Potentiometer
- 1\* 10KΩ Resistor
- 1\* Thermistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

A thermistor is a type of resistor whose resistance varies significantly with temperature, more so than in standard resistors. We are using MF52 NTC thermistor type. BTC thermistor is usually used as a temperature sensor.

MF52 thermistor key parameters:

**B-parameter : 3470.**

**25°C resistor : 10KΩ.**

The relationship between the resistance of thermistor and temperature is as follows:

$$R_{thermistor} = R * e^{\left(B * \left(\frac{1}{T_1} - \frac{1}{T_2}\right)\right)}$$

$R_{thermistor}$  : the resistance of thermistor at temperature T1

$R$  : The nominal resistance of thermistor at room temperature T2;

$e$  : 2.718281828459 ;

$B$  : It is one of the important parameters of thermistor;

$T_1$  : the Kelvin temperature that you want to measure.

$T_2$  : At the condition of room temperature 25 °C (298.15K), the standard resistance of MF52 thermistor is 10K;

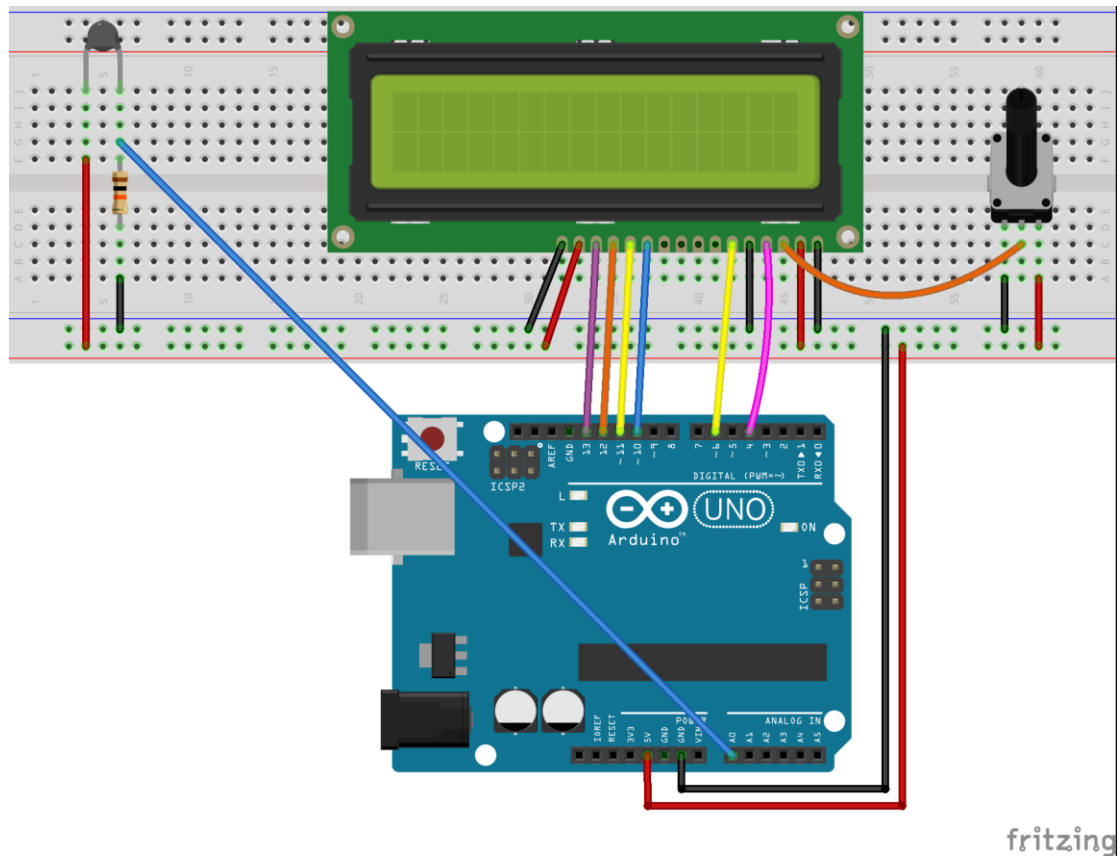
Kelvin temperature = 273.15 (absolute temperature) + degrees Celsius;

After transforming the above equation, we can get to the following formula:

$$T_1 = \frac{B}{\left(\ln\left(\frac{R_{\text{thermistor}}}{R}\right) + \frac{B}{T_2}\right)}$$

## Procedures

### 1. Build the circuit

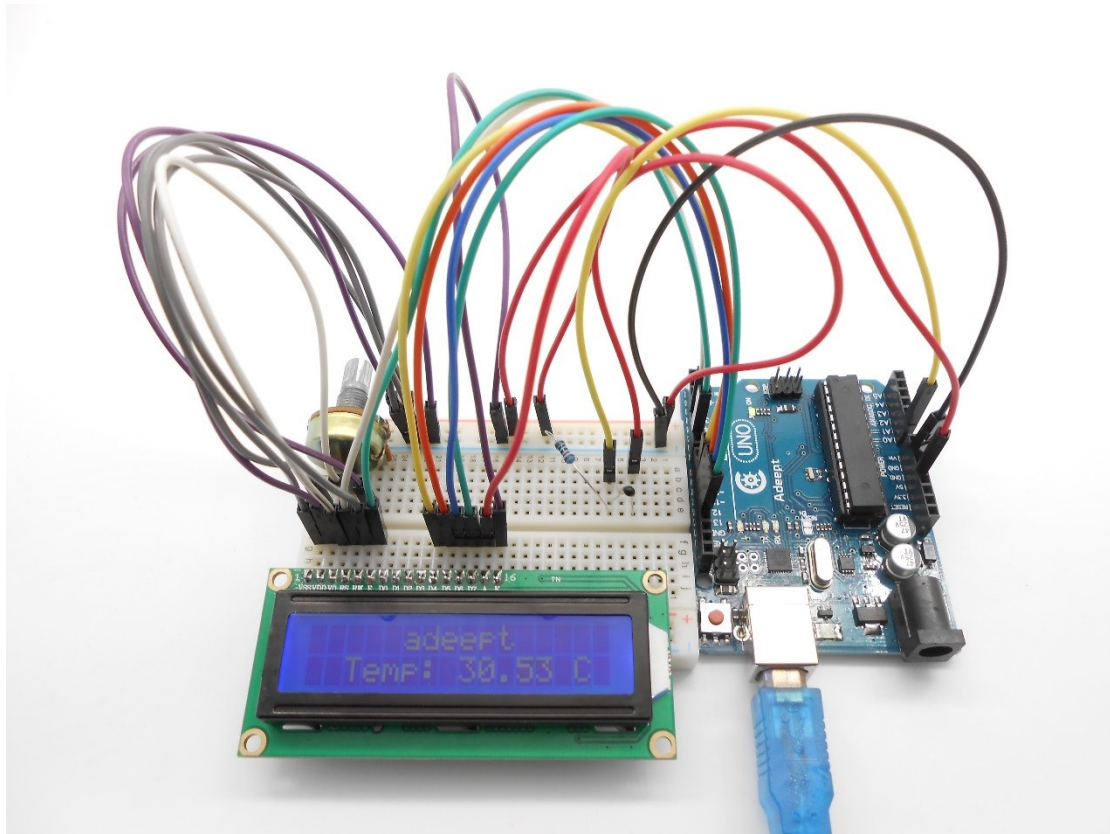


### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you can see the temperature which is collected by thermistor on the LCD1602.





## Summary

By learning this lesson, I believe you have learned to use a thermistor to measure temperature. Next, you can use a thermistor to produce some interesting applications.

# Lesson 17 IR Remoter Controller

## Overview

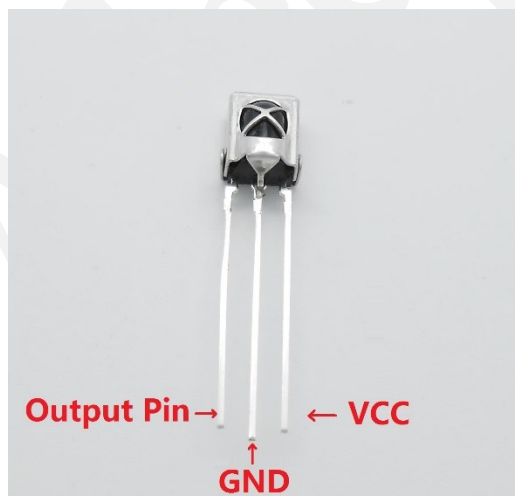
In this lesson, we will learn how to use an IR receiver to receive the remote controller signal.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* IR Receiver HX1838
- 1\* Remote Controller
- 1\* Breadboard
- Several Jumper Wires

## Principle

The IR receiver HX1838 can receive the infrared remote controller signals. The IR receiver HX1838 has only three pins (a signal line, VCC and GND). It is easy to connect with the Arduino.



The following is an infrared remote controller:



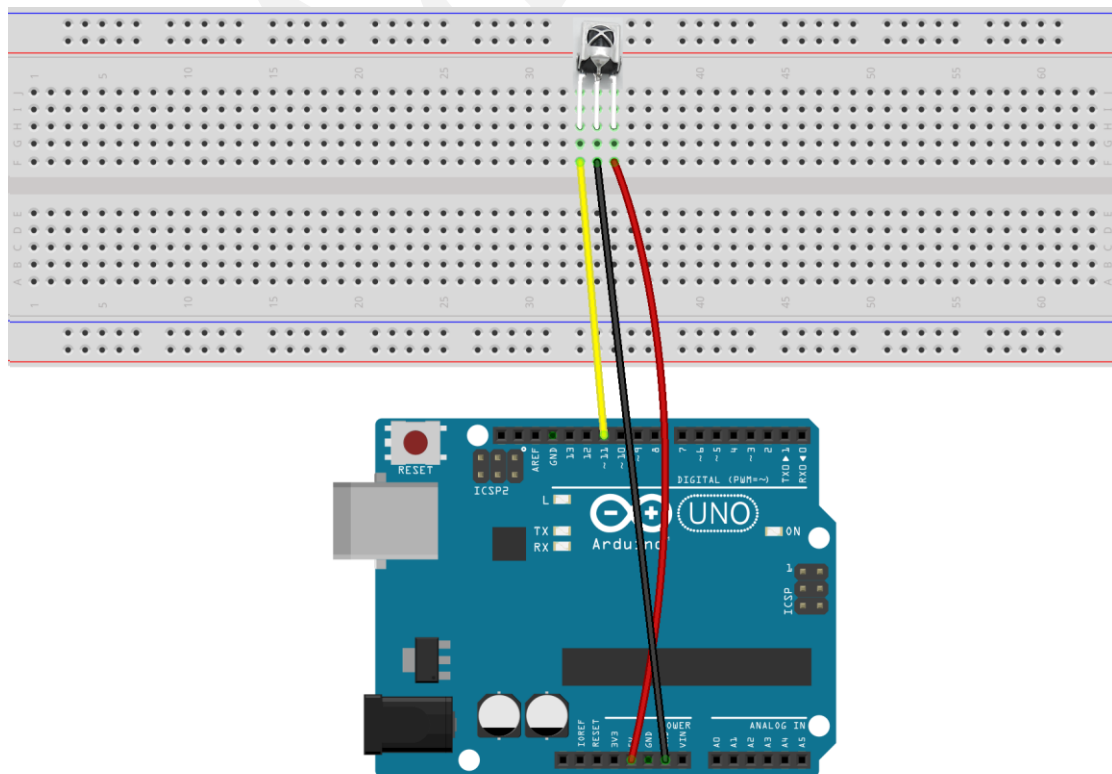
In this experiment, we program the Arduino to receive the infrared signal, and then send the received data to the serial monitor. In the program, we used the Arduino-IRremote-master library (We provided).

**Note:**

Before using this library, you have to delete the [RobotIRremote](#) directory in your Arduino IDE directory, and delete the [RobotIRremote](#) directory in system documents folder. For example, my system is windows 7, I need to delete the [RobotIRremote](#) directory in **C:\Program Files (x86)\Arduino\libraries** and **C:\Users\SJG\Documents\Arduino\libraries**. Otherwise, when you compile the program, the compiler will complain.

**Procedures**

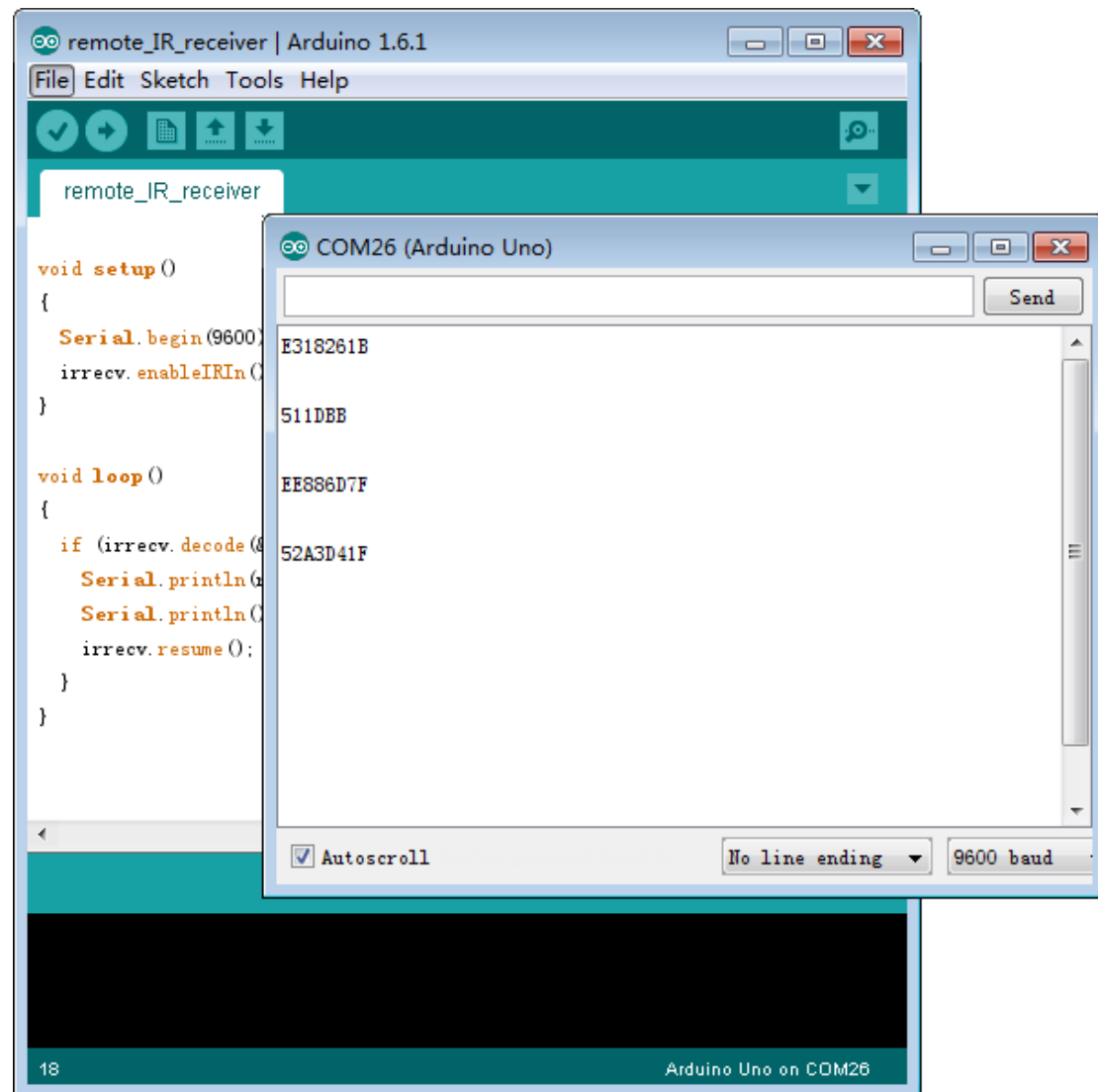
1. Build the circuit



## 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you click one of the buttons on the remote controller, you will see the button number of the remote controller is displayed on the serial monitor.



```
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn();
}

void loop()
{
  if (irrecv.decode(&irrecv.lastDecoded));
  Serial.println(irrecv.lastDecoded);
  Serial.println(irrecv.lastDecoded);
  irrecv.resume();
}
}
```

COM26 (Arduino Uno)

E318261B

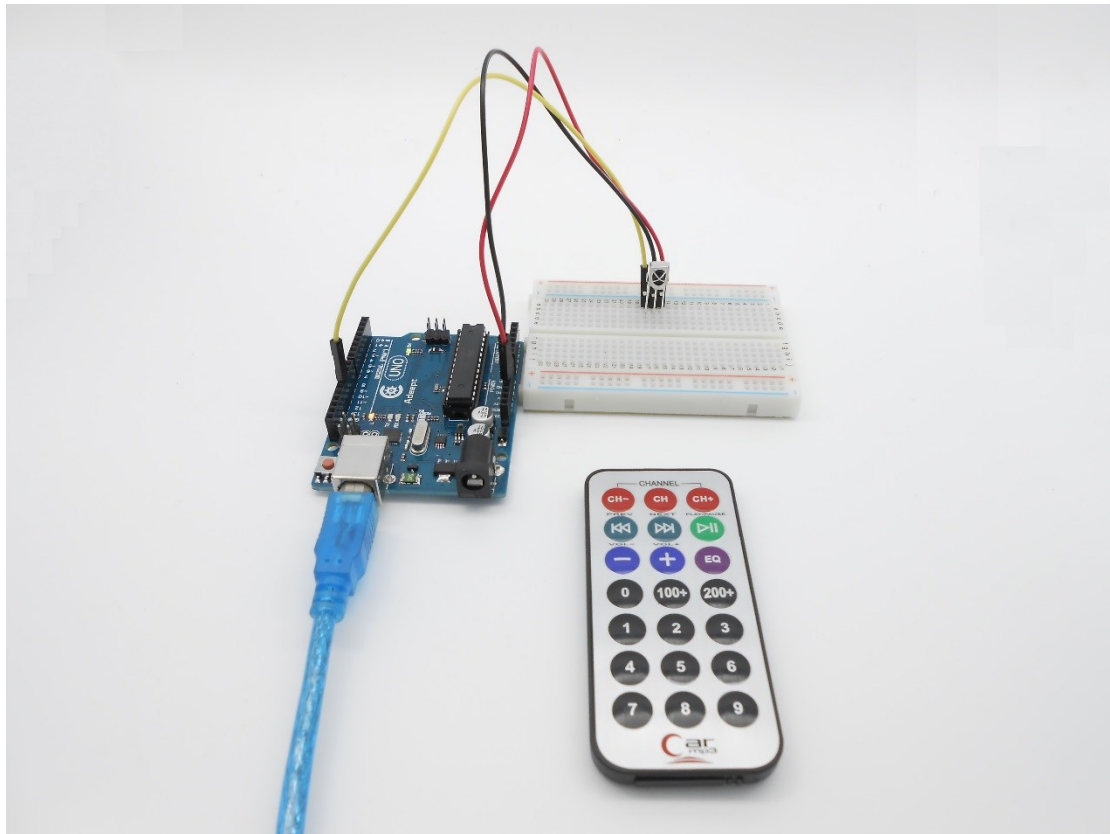
511DBB

EE88D7F

52A3D41F

Autoscroll    No line ending    9600 baud

18    Arduino Uno on COM26



## Summary

By learning this lesson, I believe you have mastered the basic principle of the infrared remote controlling.

# Lesson 18 Temperature & humidity sensor

## DHT-11

### Overview

In this lesson, we will learn how to use DHT-11 to collect temperature and humidity by programming Arduino.

### Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- 1\* DHT-11 Temperature and Humidity Sensor
- 1\* Breadboard
- Several Jumper Wires

### Principle

This DHT-11 temperature & humidity sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Item: DHT11

Measurement Range: 20-95%RH, 0-50°C

Humidity Accuracy:  $\pm 5\%$ RH

Temperature Accuracy:  $\pm 2^\circ\text{C}$

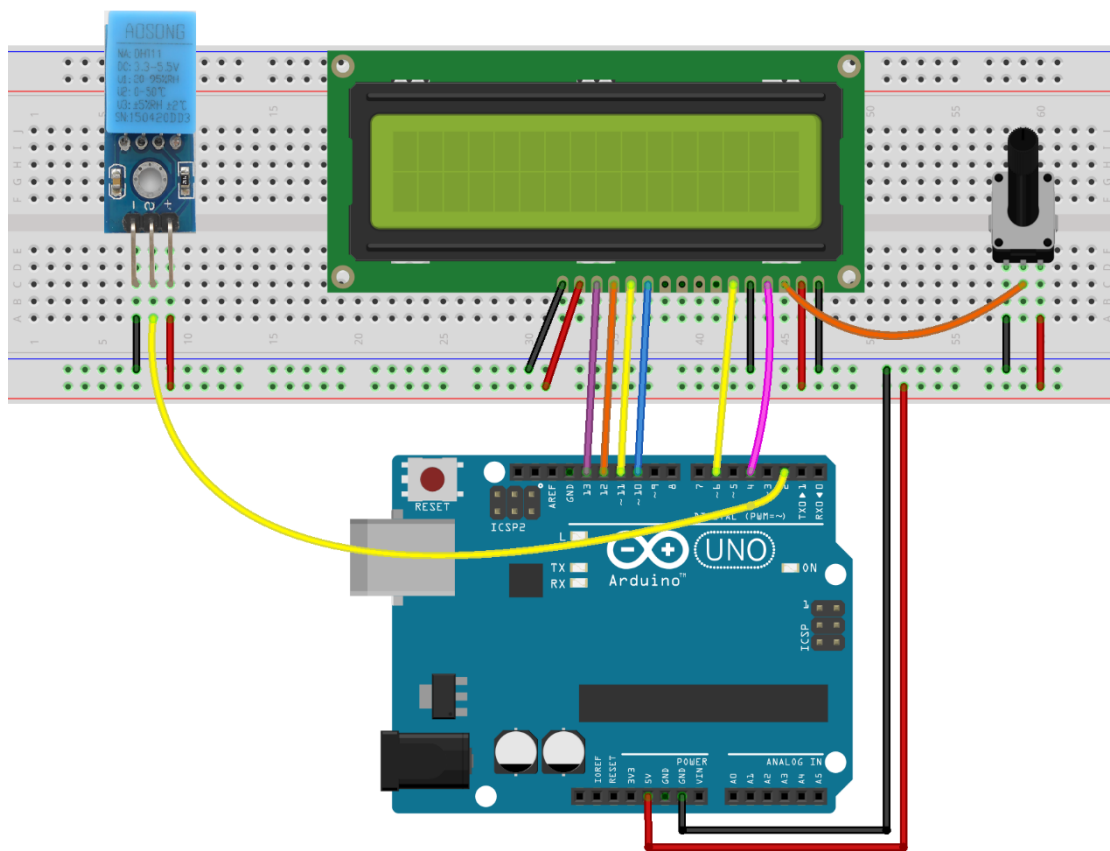
'+' : VCC (3.3~5.5V)

'-' : GND (0V)

'S' : data pin

## Procedures

### 1. Build the circuit



fritzing

### 2. Program

```
/*  
File name: 18_DHT11.ino  
Description: you can see the temperature and humidity data  
displayed on the LCD1602.  
Website: www.aadept.com  
E-mail: support@aadept.com  
Author: Tom  
Date: 2015/05/02  
*/
```

```

#include <dht11.h>
#include <LiquidCrystal.h>

dht11 DHT11;

#define DHT11PIN 2

LiquidCrystal lcd(4, 6, 10, 11, 12, 13); // Define the connection LCD pin

void setup()
{
    lcd.begin(16, 2); //set up the LCD's number of columns and rows:
    lcd.clear(); //Clears the LCD screen and positions the cursor in the upper-left corner
    delay(1000); //delay 1000ms
}

void loop()
{
    int chk = DHT11.read(DHT11PIN);

    lcd.setCursor(0, 0); // set the cursor to column 0, line 0
    lcd.print("Humidity:"); // Print a message of "Humidity: "to the LCD.
    lcd.print((float)DHT11.humidity, 2); // Print a message of "Humidity: "to the LCD.
    lcd.print(" % "); // Print the unit of the centigrade temperature to the LCD.

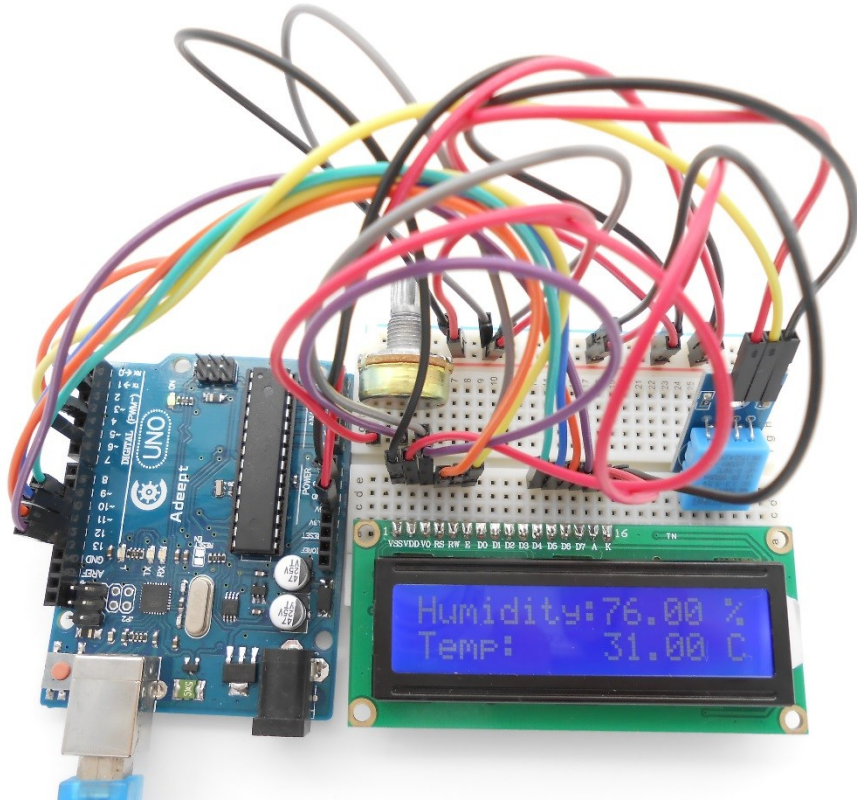
    lcd.setCursor(0, 1); // set the cursor to column 0, line 0
    lcd.print("Temp: "); // Print a message of "Temp: "to the LCD.
    lcd.print((float)DHT11.temperature, 2); // Print a centigrade temperature to the LCD.
    lcd.print(" C "); // Print the unit of the centigrade temperature to the LCD.
    delay(1000); // delay 1S
}

```

### 3. Compile the program and upload to Arduino UNO board

Now, you can see the temperature and humidity data displayed on the LCD1602.





## Summary

By learning this lesson, I believe that you should be able to measure the room's temperature and humidity. I hope you can make a simple smart home system based on this and the previous lessons.

# Lesson 19 Ultrasonic distance sensor

## Overview

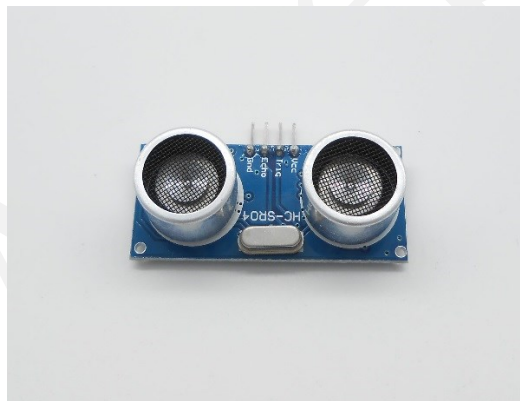
In this lesson, we will learn how to measure the distance by the ultrasonic distance sensor.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Ultrasonic Distance Sensor
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- Several Jumper Wires

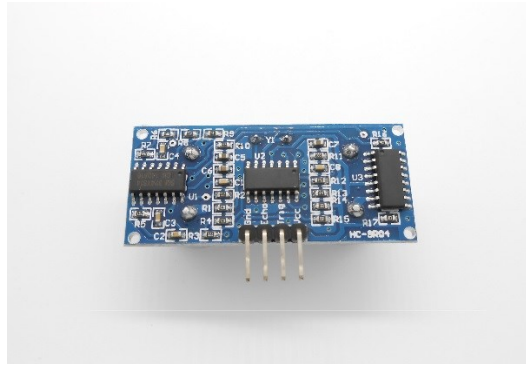
## Principle

This recipe uses the popular Parallax PING ultrasonic distance sensor to measure the distance of an object ranging from 2 cm to around 3 m.



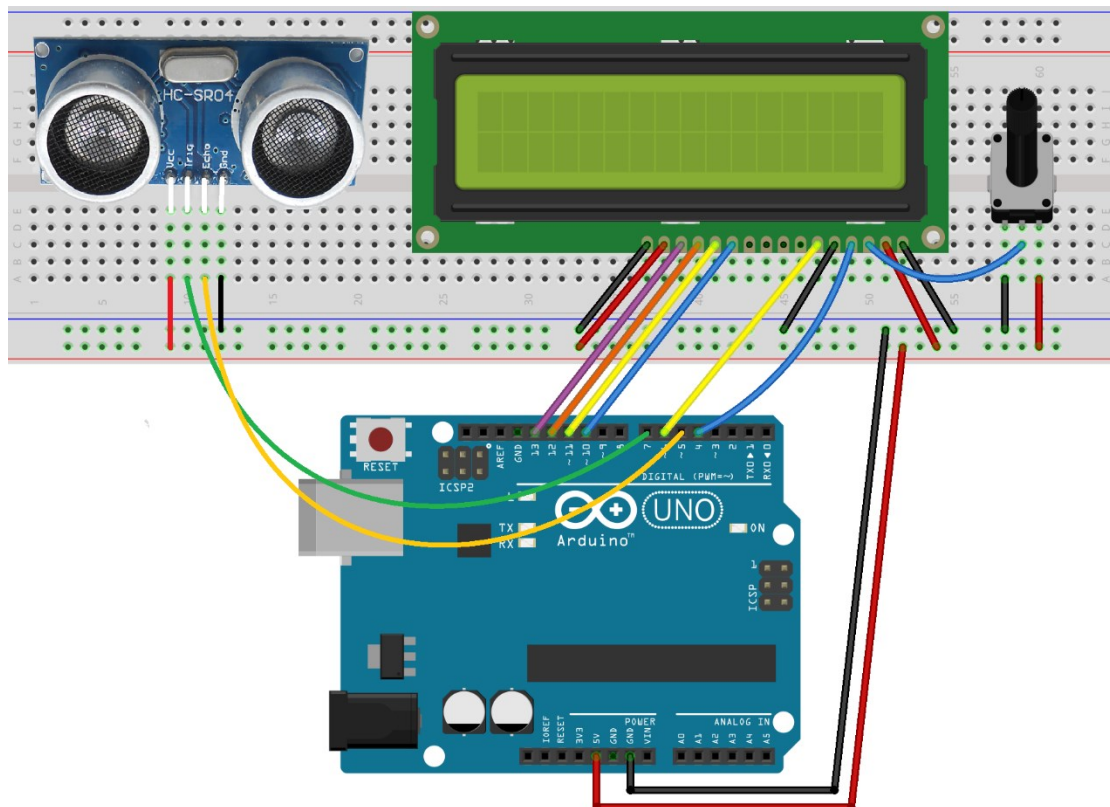
Ultrasonic sensors provide a measurement of the time it takes for sound to bounce off an object and return to the sensor. The “ping” sound pulse is generated when the pingPin level goes HIGH for two micro-seconds. The sensor will then generate a pulse that terminates when the sound returns. The width of the pulse is proportional to the distance the sound traveled and the sketch then uses the pulseIn function to measure that duration. The speed of sound is 340 meters per second, which is 29 microseconds per centimeter. The formula for the distance of the round trip is: RoundTrip = microseconds / 29.

So, the formula for the one-way distance in centimeters is: microseconds / 29 / 2



## Procedures

### 1. Build the circuit

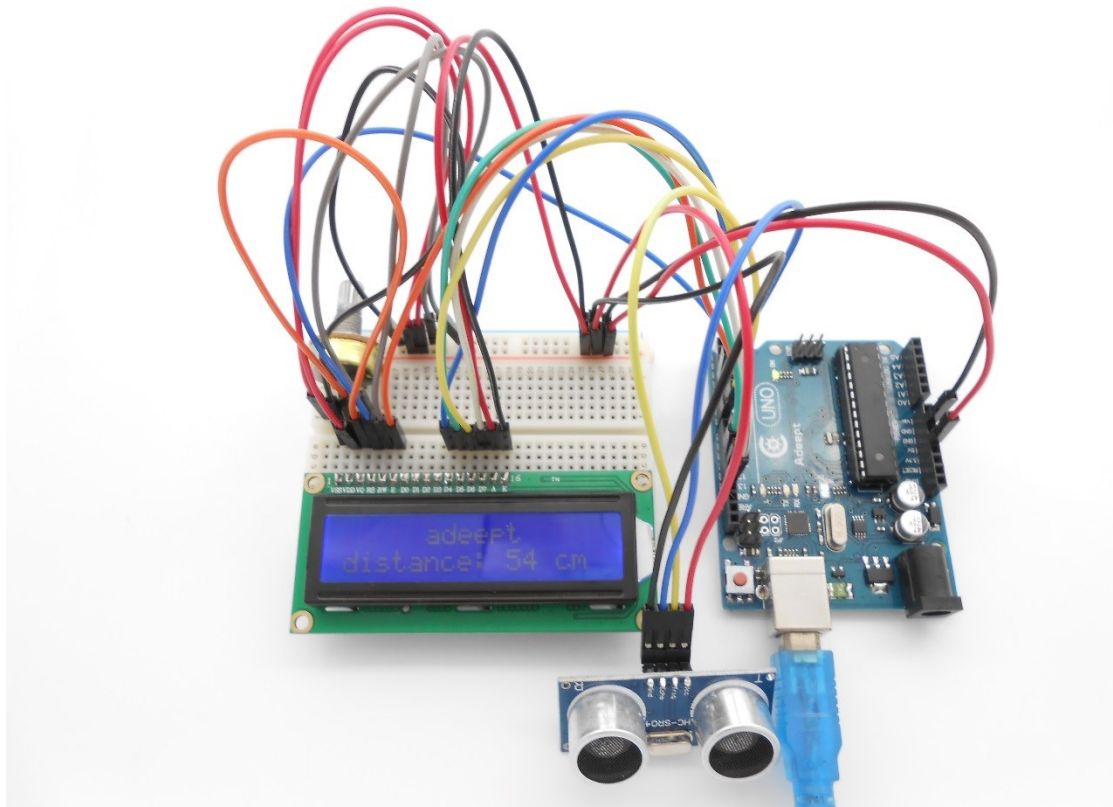


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you try to change the distance between the ultrasonic module and the obstacles, you will find the distance value displayed on the LCD1602 will be changed.



Adept1

# Lesson 20 3-axis Accelerometer—ADXL345

## Overview

In this lesson, we will learn how to use ADXL345 to collect acceleration by programming Arduino UNO, and then display the data that ADXL345 collects on the LCD1602.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- 1\* ADXL345 Acceleration Sensor
- 1\* Breadboard
- Several Jumper Wires

## Principle

### 1. ADXL345

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to  $\pm 16$  g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3-wire or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

### 2. Wire Library

This library allows you to communicate with I2C/TWI devices. I2C/TWI pins are A4 (SDA) and A5(SCL) on Arduino UNO R3 board.

### 3. Key functions:

- `Wire.begin()`
- `Wire.begin(address)`

Initiate the Wire library and join the I2C bus as a master or slave. This should normally be called only once.

#### *Parameters*

address: the 7-bit slave address (optional); if not specified, join the bus as a master.

#### *Returns*

None

#### ● `Wire.beginTransmission(address)`

Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the `write()` function and transmit them by calling `endTransmission()`.

#### *Parameters*

address: the 7-bit address of the device to transmit to

#### *Returns*

None

#### ● `Wire.write()`

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to `beginTransmission()` and `endTransmission()`).

#### *Syntax*

`Wire.write(value)`

`Wire.write(string)`

`Wire.write(data, length)`

#### *Parameters*

value: a value to send as a single byte

string: a string to send as a series of bytes

data: an array of data to send as bytes

length: the number of bytes to transmit

#### *Returns*

byte: `write()` will return the number of bytes written, though reading that number is optional

#### ● `Wire.endTransmission()`

Ends a transmission to a slave device that was begun by `beginTransmission()` and transmits the bytes that were queued by `write()`.

As of Arduino 1.0.1, `endTransmission()` accepts a boolean argument changing its behavior for compatibility with certain I2C devices.

If true, `endTransmission()` sends a stop message after transmission, releasing

the I2C bus.

If false, `endTransmission()` sends a restart message after transmission. The bus will not be released, which prevents another master device from transmitting between messages. This allows one master device to send multiple transmissions while in control.

The default value is true.

*Syntax*

`Wire.endTransmission()`

`Wire.endTransmission(stop)`

*Parameters*

`stop` : boolean. true will send a stop message, releasing the bus after transmission. false will send a restart, keeping the connection active.

*Returns*

byte, which indicates the status of the transmission:

-0:success

-1:data too long to fit in transmit buffer

-2:received NACK on transmit of address

-3:received NACK on transmit of data

-4:other error

● `Wire.requestFrom()`

Used by the master to request bytes from a slave device. The bytes may then be retrieved with the `available()` and `read()` functions.

As of Arduino 1.0.1, `requestFrom()` accepts a boolean argument changing its behavior for compatibility with certain I2C devices.

If true, `requestFrom()` sends a stop message after the request, releasing the I2C bus.

If false, `requestFrom()` sends a restart message after the request. The bus will not be released, which prevents another master device from requesting between messages. This allows one master device to send multiple requests while in control.

The default value is true.

*Syntax*

`Wire.requestFrom(address, quantity)`

`Wire.requestFrom(address, quantity, stop)`

*Parameters*

`address`: the 7-bit address of the device to request bytes from

`quantity`: the number of bytes to request

`stop` : boolean. true will send a stop message after the request, releasing the



bus. false will continually send a restart after the request, keeping the connection active.

*Returns*

byte : the number of bytes returned from the slave device

● [Wire.available\(\)](#)

Returns the number of bytes available for retrieval with read(). This should be called on a master device after a call to requestFrom() or on a slave inside the onReceive() handler.

available() inherits from the Stream utility class.

*Parameters*

None

*Returns*

The number of bytes available for reading.

● [Wire.read\(\)](#)

Reads a byte that was transmitted from a slave device to a master after a call to requestFrom() or was transmitted from a master to a slave. read() inherits from the Stream utility class.

*Syntax*

[Wire.read\(\)](#)

*Parameters*

none

*Returns*

The next byte received

● [lcd.setCursor\(\)](#)

Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.

*Syntax*

[lcd.setCursor\(col, row\)](#)

*Parameters*

lcd: a variable of type LiquidCrystal

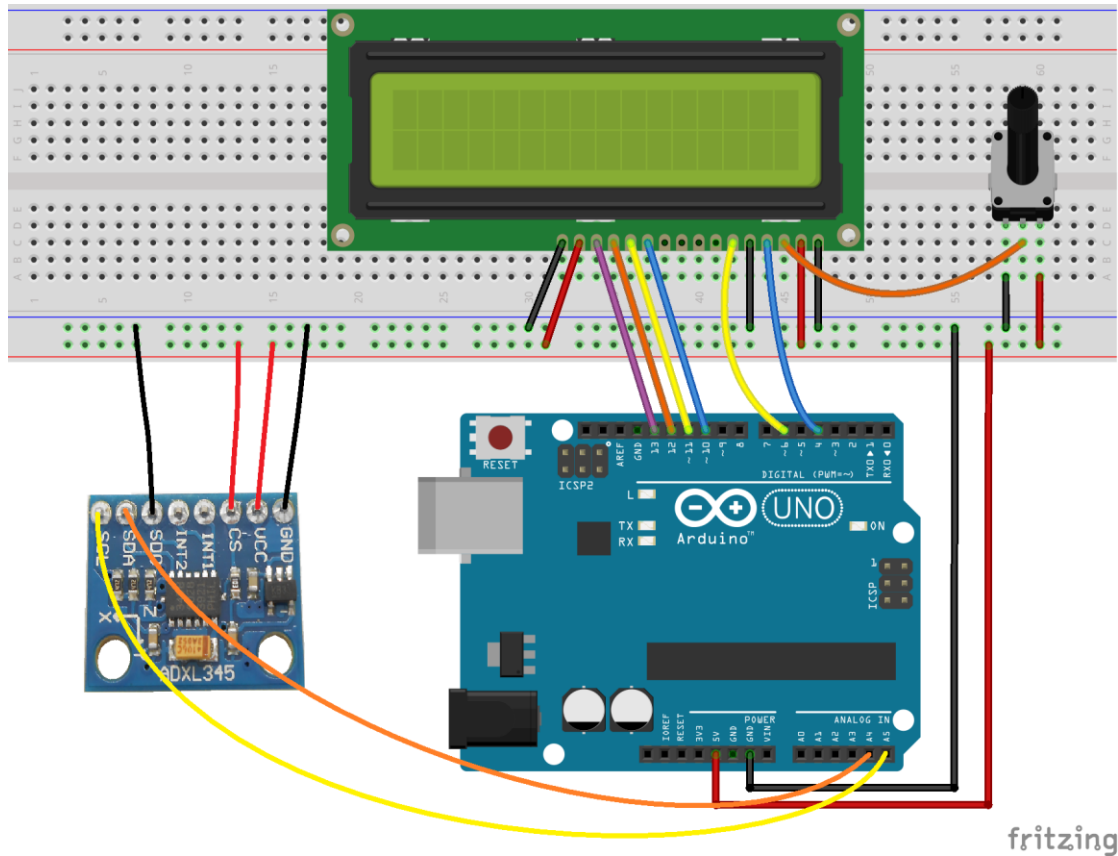
col: the column at which to position the cursor (with 0 being the first column)

row: the row at which to position the cursor (with 0 being the first row)

## Procedures

### 1. Build the circuit

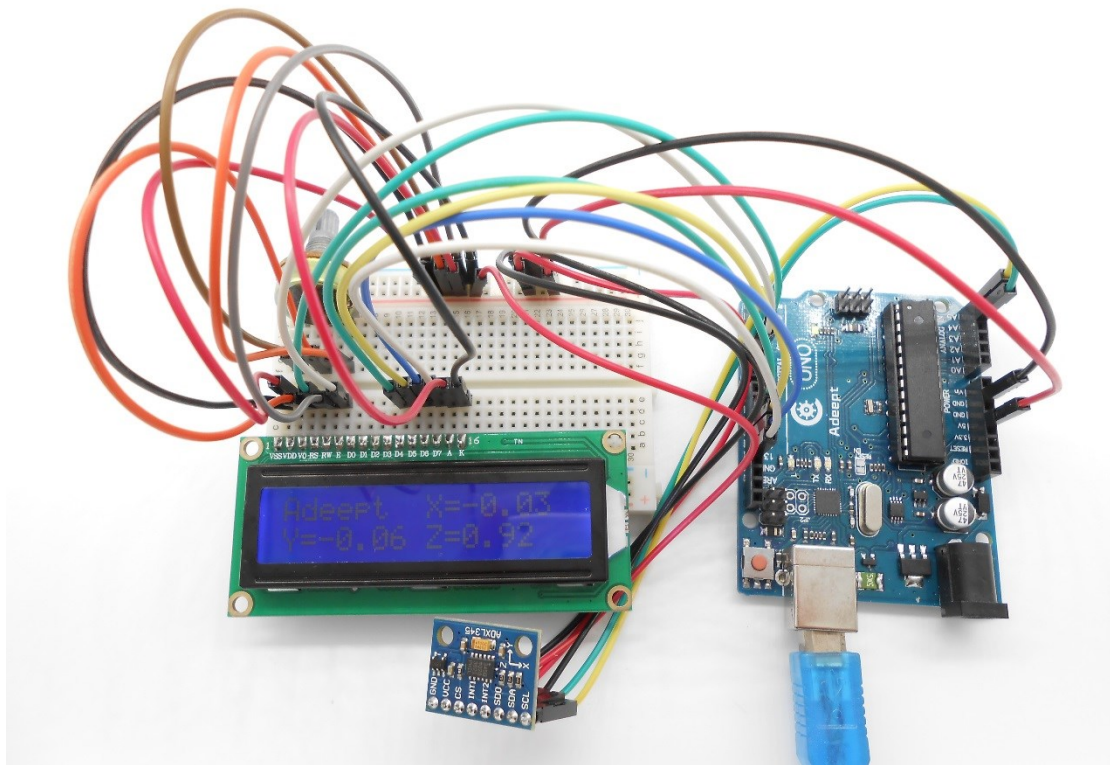




2. Program

3. Compile the program and upload to Arduino UNO board

Now, you can see the acceleration data which ADXL345 collected displayed on the LCD1602.



## Summary

By learning this lesson, I believe that you have learned the usage of the ADXL345 to collect acceleration data. Next, you can use ADXL345 to make some interesting applications.

# Lesson 21 4x4 matrix keyboard

## Overview

In this lesson, we will learn how to use the matrix keyboard.

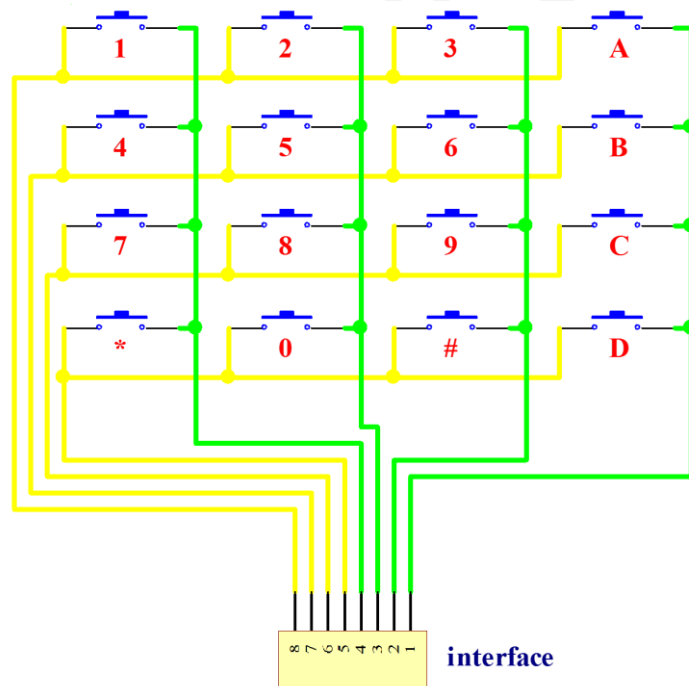
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 4x4 Matrix Keyboard
- 1\* Breadboard
- Several Jumper Wires

## Principle

In order to save the resources of the microcontroller port, we usually connect the buttons in a matrix in an actual project.

The following is the schematics of 4x4 matrix keyboard:

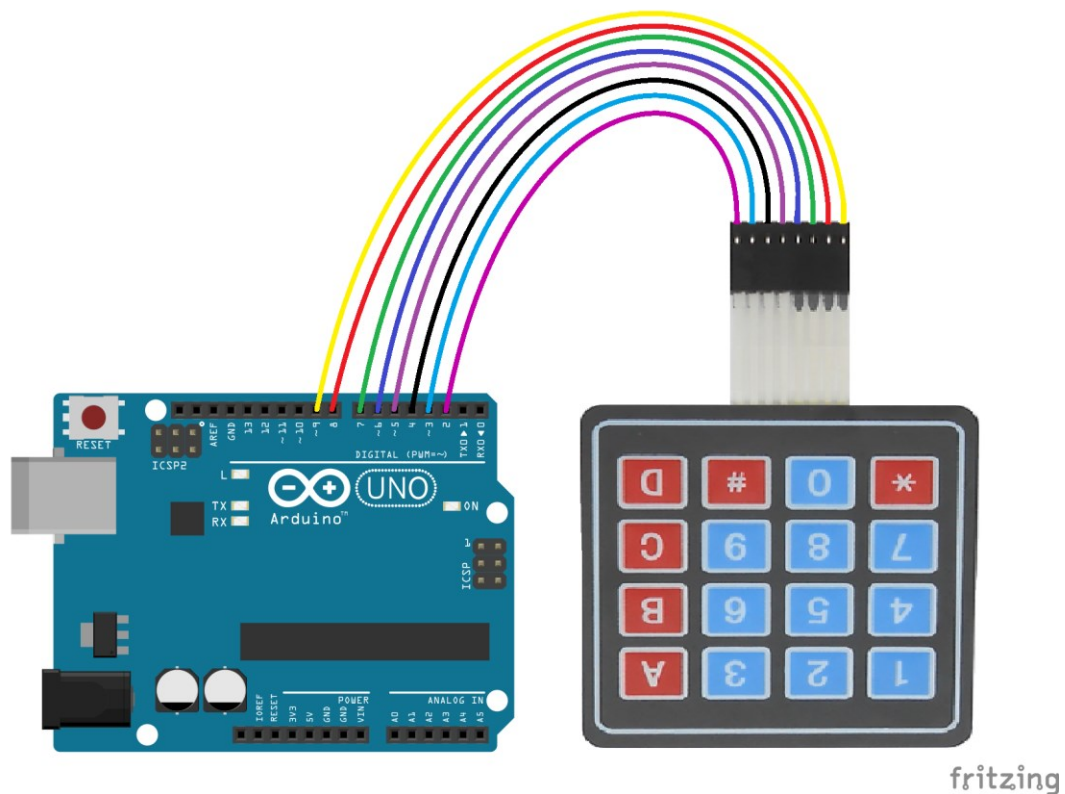




In this tutorial, we use the 'Keypad' function library. Before programming, please install the library.

## Procedures

### 1. Build the circuit

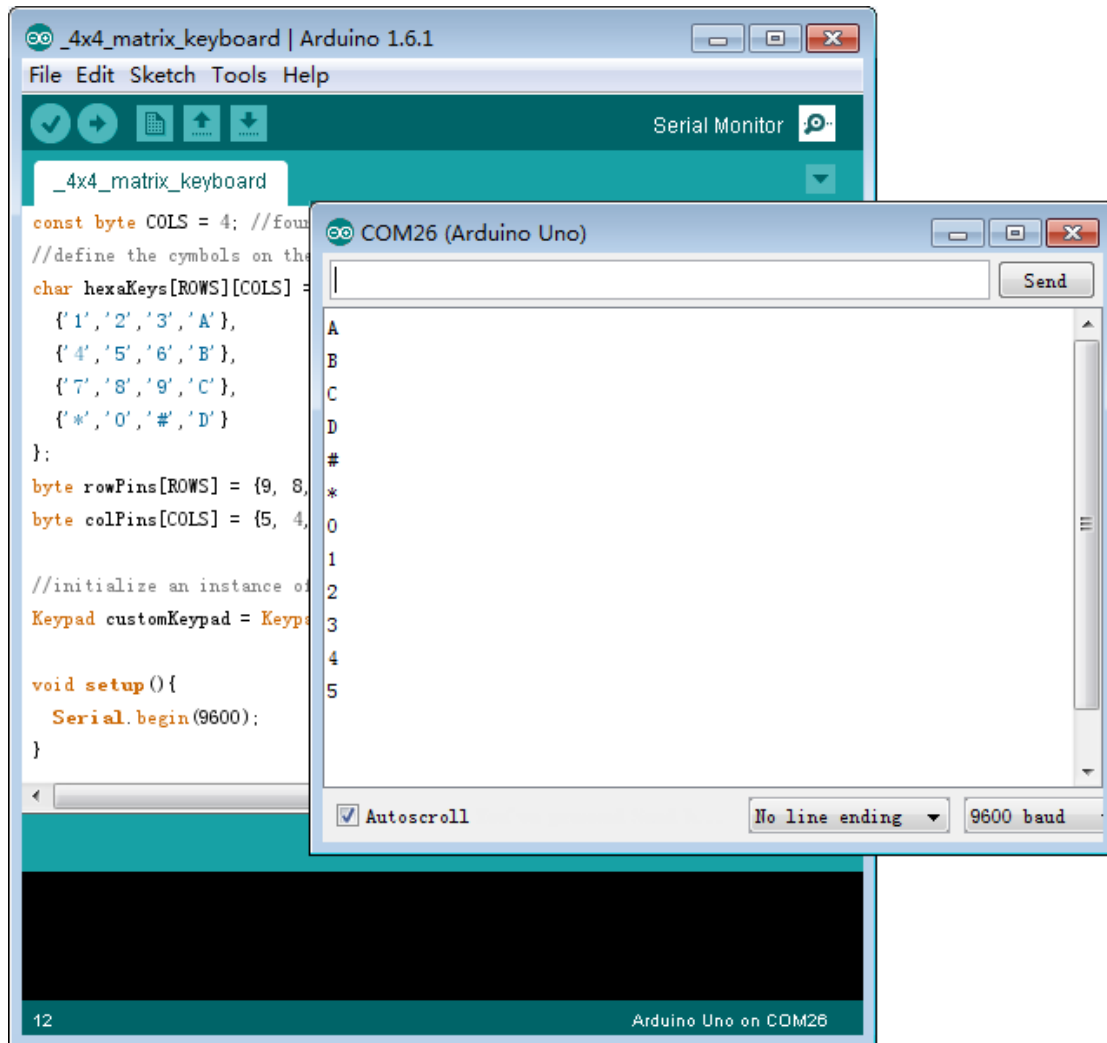


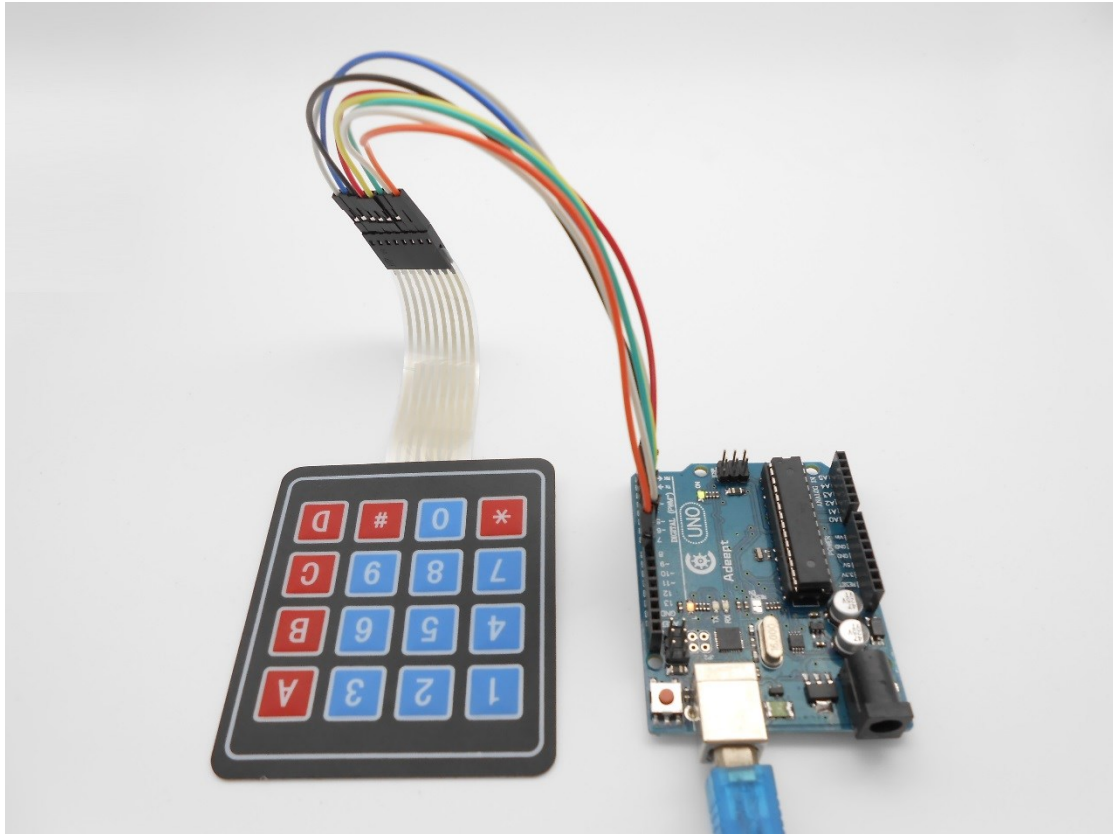
### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you click one of the button on the 4x4 matrix keyboard, you will

see the corresponding key value will be displayed on the serial monitor.





Adapt

# Lesson 22 Controlling DC motor

## Overview

In this comprehensive experiment, we will learn how to control the state of DC motor with Arduino, and the state will be displayed through the LED at the same time. The state of DC motor includes its forward, reverse, acceleration, deceleration and stop.

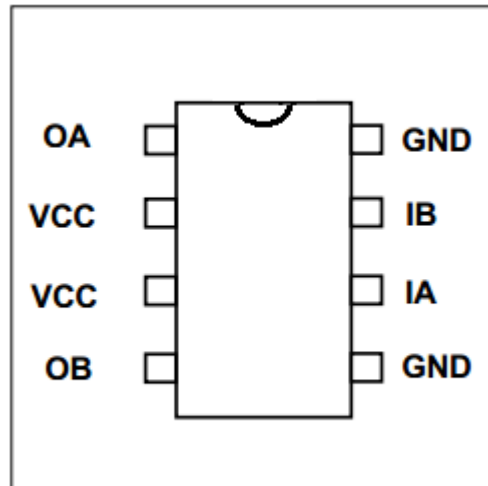
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* L9110 DC Motor Driver
- 1\* DC Motor
- 4\* Button
- 4\* LED
- 4\* 220Ω Resistor
- 1\* Battery Holder
- 1\* Breadboard
- Several Jumper Wires

## Principle

### 1. L9110

L9110 is a driver chip which is used to control and drive motor. The chip has two TTL/CMOS compatible input terminals, and possesses the property of anti-interference: it has high current driving capability, two output terminals that can directly drive DC motor, each output port can provide 750~800mA dynamic current, and its peak current can reach 1.5~2.0A; L9110 is widely applied to various motor drives, such as toy cars, stepper motor, power switches and other electric circuits.



OA, OB: These are used to connect the DC motor.

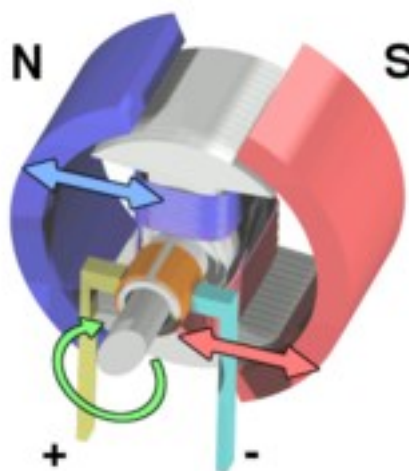
VCC: Power supply (+5V)

GND: The cathode of the power supply (Ground).

IA, IB: The input terminal of drive signal.

## 2. DC motor

A DC motor is any of a class of electrical machines that converts direct current electrical power into mechanical power. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor. Most types produce rotary motion; a linear motor directly produces force and motion in a straight line.





DC motors were the first type widely used, since they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor can operate on direct current but is a lightweight motor used for portable power tools and appliances.



### 3. Key functions

- switch / case statements

Like if statements, switch...case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

#### Example

```
switch (var) {  
    case 1:  
        //do something when var equals 1  
        break;  
    case 2:  
        //do something when var equals 2
```

```

    break;
default:
    // if nothing else matches, do the default
    // default is optional
}

```

## Syntax

```

switch (var) {
  case label:
    // statements
    break;
  case label:
    // statements
    break;
  default:
    // statements
}

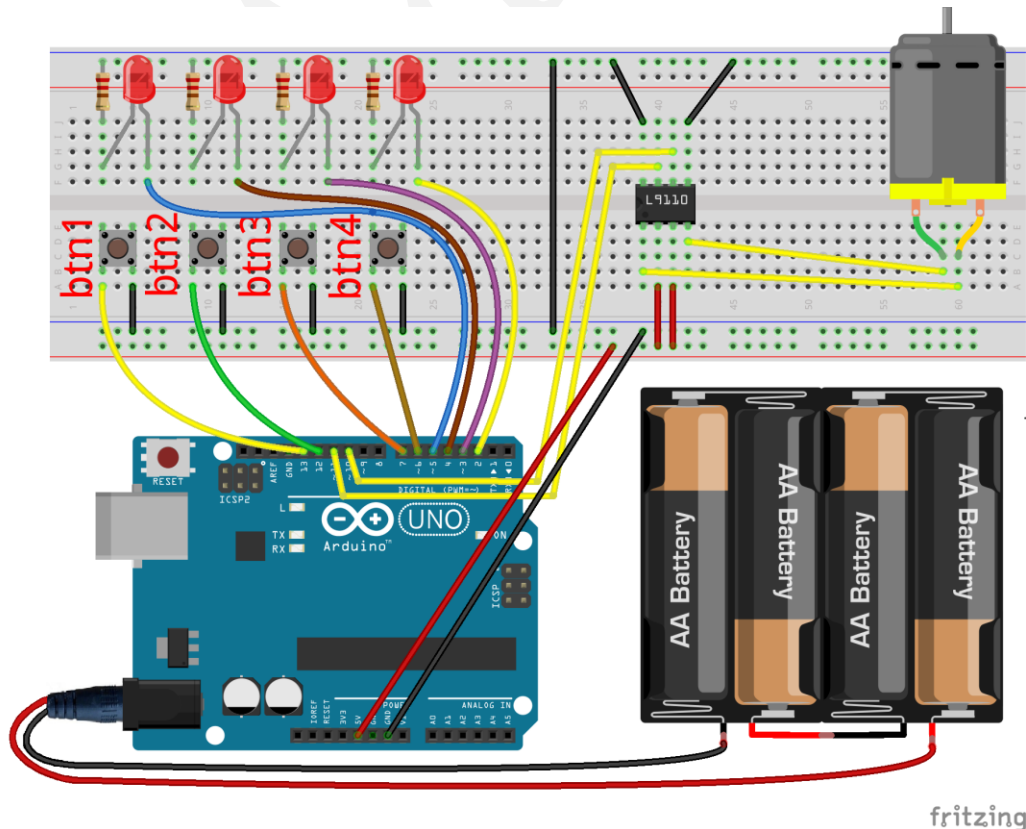
```

## Parameters

var: the variable whose value to compare to the various cases  
 label: a value to compare the variable to

## Procedures

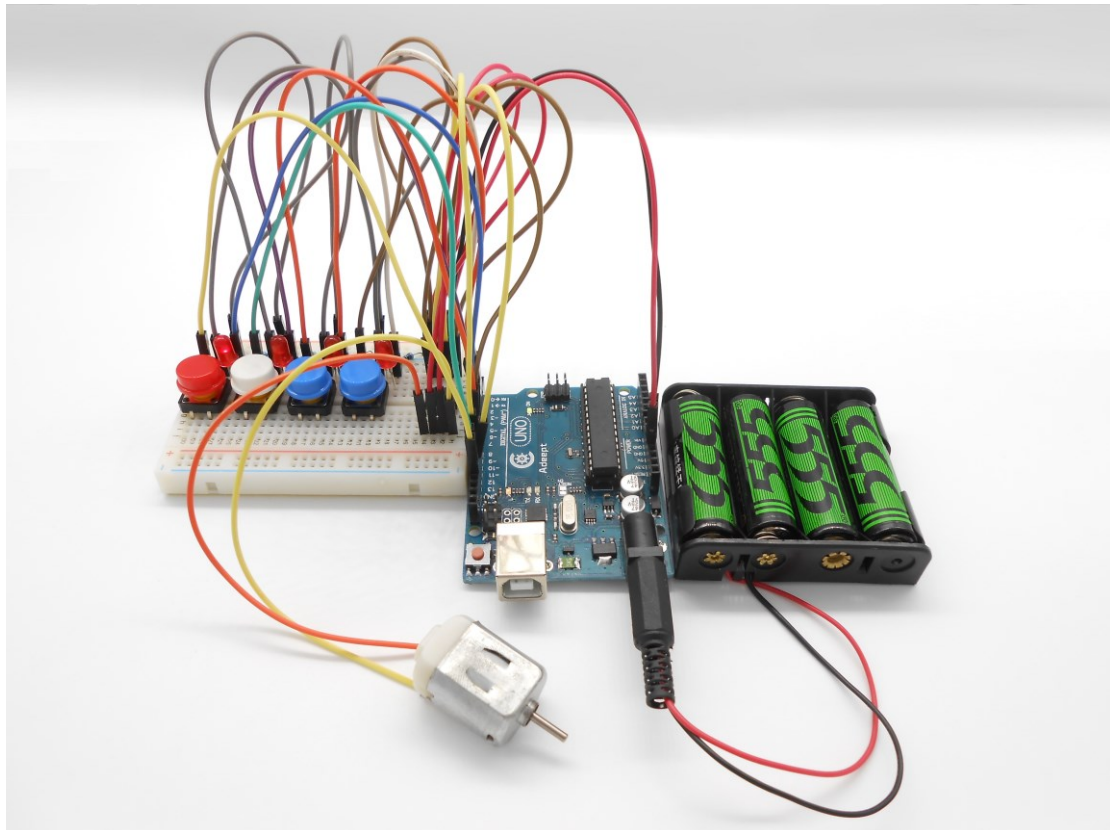
1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)



## 2. Program

### 3. Compile the program and upload to Arduino UNO board

Press the btn1 button to stop or run the DC motor; press the btn2 button to forward or reverse the DC motor; Press the btn3 button to accelerate the DC motor; Press the btn4 button to decelerate the DC motor. When one of the four buttons is pressed, their corresponding LED will be flashing which prompts that the current button is clicked.



## Summary

I think you must have grasped the basic theory and programming of the DC motor after studying this experiment. You not only can forward and reverse it, but also can regulate its speed. Besides, you can do some interesting applications with the combination of this course and your prior knowledge.

# Lesson 23 Joy stick

## Overview

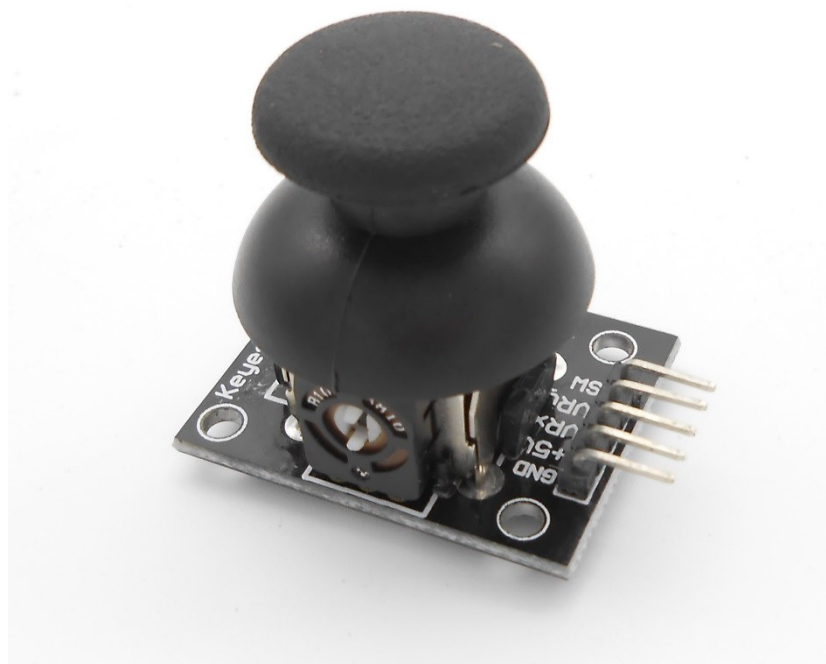
In this lesson, we will learn the usage of joy stick. We program the Arduino to detect the state of joy stick, and display the information on an LCD1602.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- 1\* Joy Stick
- 1\* Breadboard
- Several Jumper Wires

## Principle

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. A joystick, also known as the control column, is the principal control device in the cockpit of many civilian and military aircraft, either as a center stick or side-stick. It often has supplementary switches to control various aspects of the aircraft's flight.

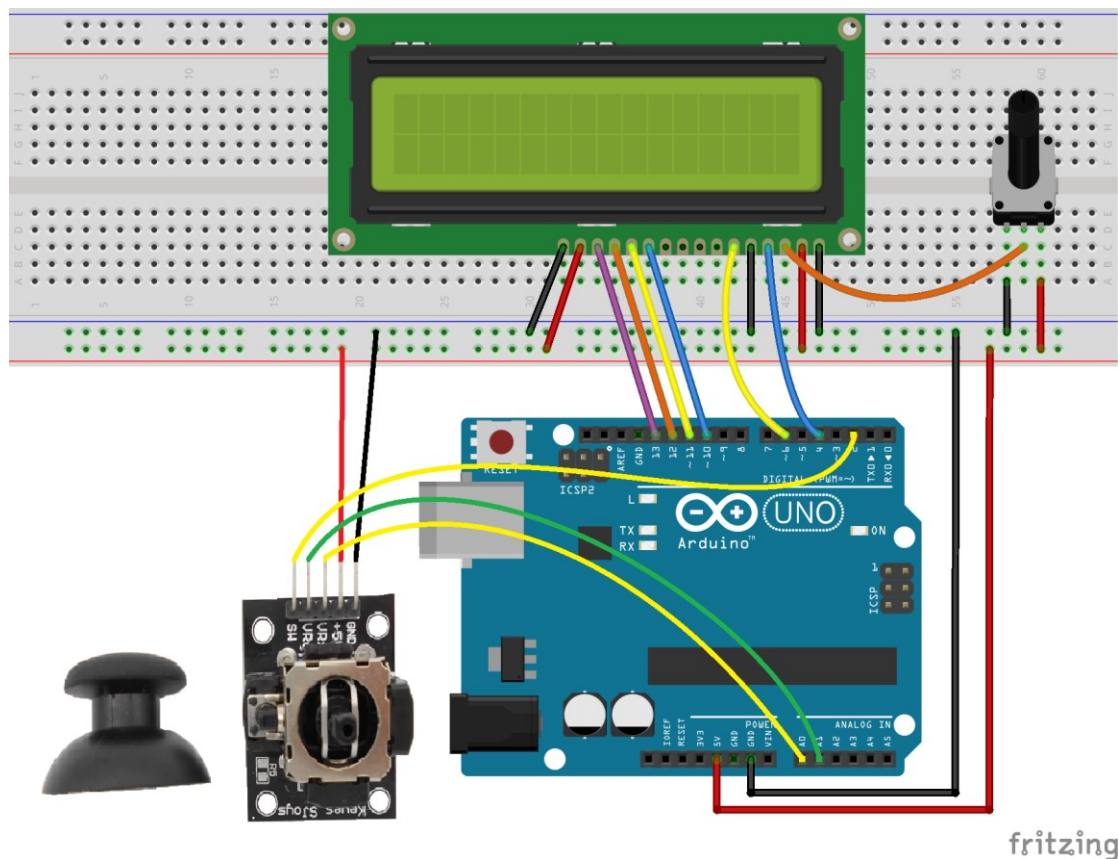


Joysticks are often used to control video games, and usually have one or more

push-buttons whose state can also be read by the computer. A popular variation of the joystick used on modern video game consoles is the analog stick. Joysticks are also used for controlling machines such as cranes, trucks, underwater unmanned vehicles, wheelchairs, surveillance cameras, and zero turning radius lawn mowers. Miniature finger-operated joysticks have been adopted as input devices for smaller electronic equipment such as mobile phones.

## Procedures

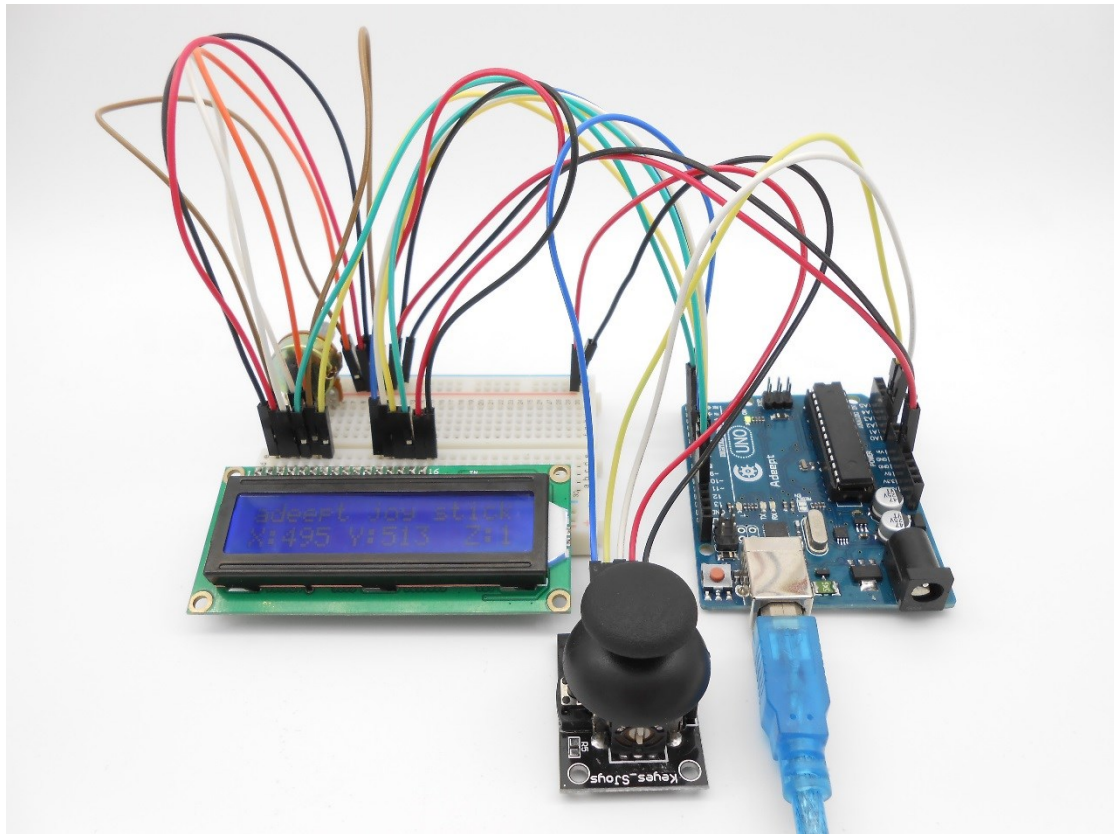
### 1. Build the circuit



### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, you can see the joy stick state information displayed on the LCD1602.



Adeci



# Lesson 24 Tilt Switch

## Overview

In this lesson, we will learn how to use the tilt switch and change the state of an LED by changing the angle of tilt switch.

## Requirement

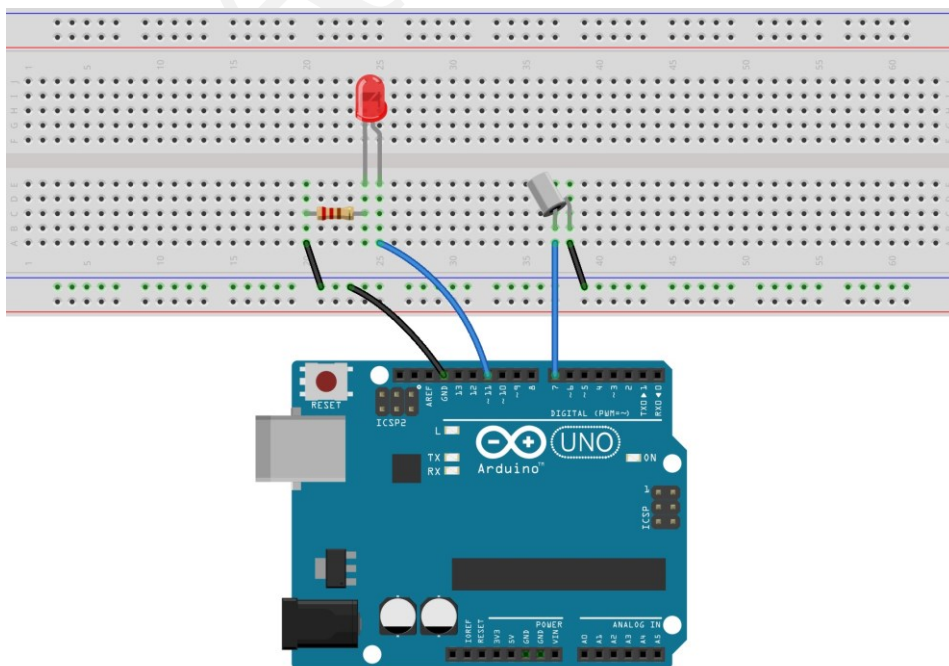
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Tilt Switch
- 1\* LED
- 1\* 220Ω Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

The tilt switch is also called the ball switch. When the switch is tilted in the appropriate direction, the contacts will be connected, tilting the switch the opposite direction causes the metallic ball to move away from that set of contacts, thus breaking that circuit.

## Procedures

### 1. Build the circuit



fritzing

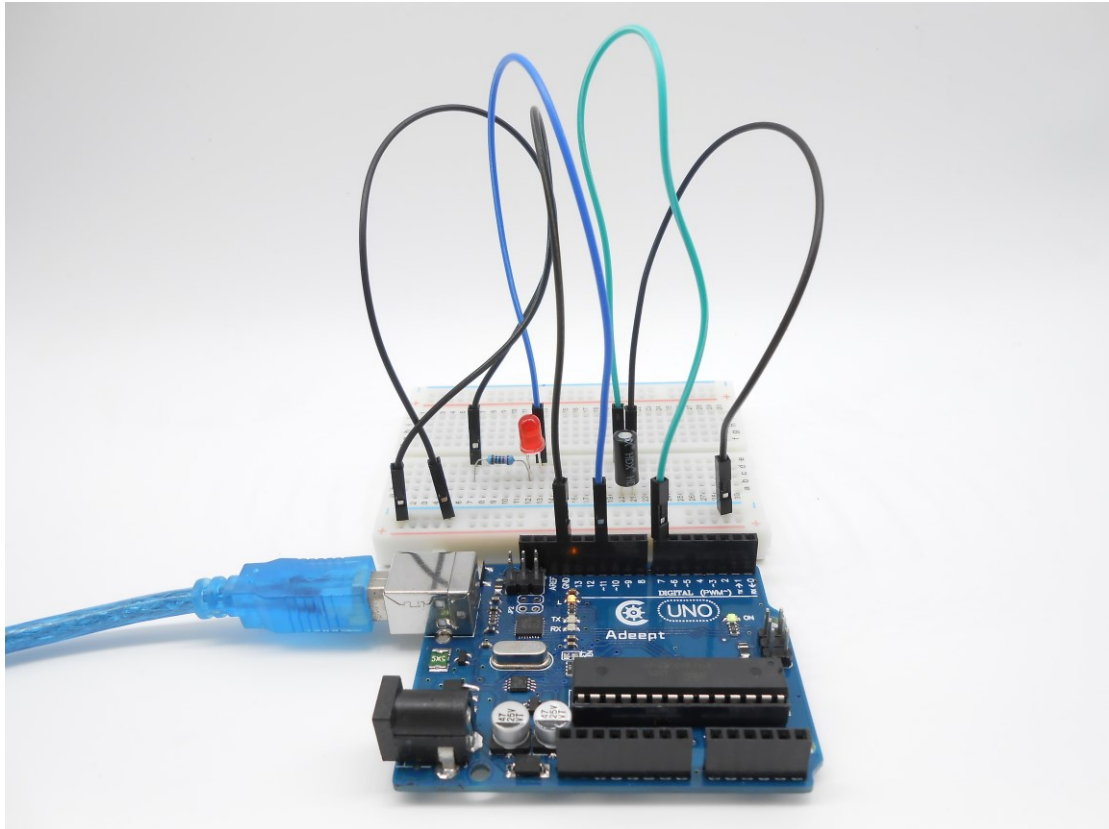
## 2. Program

```
/******  
File name: 24_tiltSwitch.ino  
Description: Tilt switches to control the LED light on or off  
Website: www.adeept.com  
E-mail: support@adeept.com  
Author: Tom  
Date: 2015/05/02  
*****/  
  
int ledpin=11;          //definition digital 11 pins as pin to control the  
                        //LED  
int tiltSwitchpin=7;   //Set the digital 7 to tilt switch interface  
int val;               //Define variable val  
  
void setup()  
{  
  pinMode(ledpin,OUTPUT); //Define small lights interface for the  
                          //output interface  
  pinMode(tiltSwitchpin,INPUT_PULLUP); //define the tilt switch  
                                       //interface for input interface  
}  
  
void loop()  
{  
  val=digitalRead(tiltSwitchpin); //Read the number seven level value is  
                                   //assigned to val  
  if(val==LOW)                     //Detect tilt switch is disconnected, the  
                                   //tilt switch when small lights go out  
  { digitalWrite(ledpin,LOW); } //Output low, LED OFF  
  else                             //Detection of tilt switch is conduction,  
  //tilt the little lights up when the switch conduction  
  { digitalWrite(ledpin,HIGH); } //Output high, LED ON  
}
```

## 3. Compile the program and upload to Arduino UNO board

Now, when you lean the breadboard at a certain angle, you will see the state of LED is changed.





## Summary

In this lesson, we have learned the principle and application of the tilt switch. Tilt switch is a very simple electronic component, but simple device can often make something interesting.

# Lesson 25 Dot-matrix display

## Overview

In this lesson, we will program to control a 8\*8 dot-matrix to realize the display of graphical and digital we want.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 8\*8 Dot-matrix
- 2\* 74HC595
- 1\* Breadboard
- Several Jumper Wires

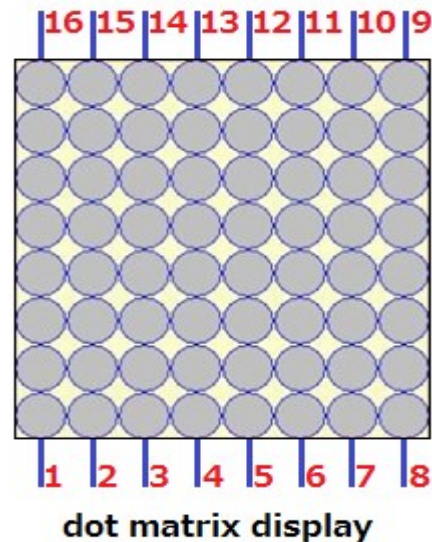
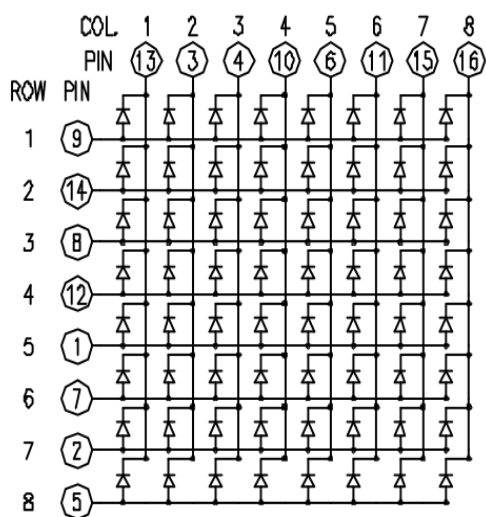
## Principle

### *1. Dot-matrix display*

A dot-matrix display is a display device used to display information on machines, clocks, railway departure indicators and many other devices requiring a simple display device of limited resolution.

The display consists of a dot-matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot-matrix controller converts instructions from a processor into signals which turns on or off lights in the matrix so that the required display is produced.

The internal structure and appearance of the dot-matrix display is as shown in below:



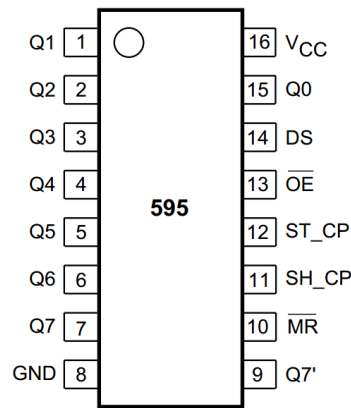
A 8\*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column is set as low level, then the LED will be lit.

A certain drive current is required for the dot-matrix display. In addition, more pins are needed for connecting dot-matrix display with controller. Thus, to save the Arduino's GPIO, driver IC 74HC595 is used in the experiment.

## 2. 74HC595

The 74HC595 is an 8-stage serial shift register with a storage register and 3-state outputs. The shift register and storage register have separate clocks. Data is shifted on the positive-going transitions of the SH\_CP input. The data in each register is transferred to the storage register on a positive-going transition of the ST\_CP input. The shift register has a serial input (DS) and a serial standard output ( Q7' ) for cascading. It is also provided with asynchronous reset (active LOW) for all 8 shift register stages. The storage register has 8 parallel 3-state bus driver outputs. Data in the storage register appears at the output whenever the output enable input (OE) is LOW.

In this experiment, only 3 pins of Arduino are used for controlling a dot-matrix display due to the existence of 74HC595.



The following is the function of each pin:

**DS:** Serial data input

**Q0-Q7:** 8-bit parallel data output

**Q7':** Series data output pin, always connected to DS pin of the next 74HC595

**OE:** Output enable pin, effective at low level, connected to the ground directly

**MR:** Reset pin, effective at low level, directly connected to 5V high level in practical applications

**SH\_CP:** Shift register clock input

**ST\_CP:** storage register clock input

### 3. Key function:

#### ● `shiftOut()`

Shifts out a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.

#### Syntax

`shiftOut(dataPin, clockPin, bitOrder, value)`

#### Parameters

`dataPin`: the pin on which to output each bit (int).

`clockPin`: the pin to toggle once the `dataPin` has been set to the correct value.

`bitOrder`: which order to shift out the bits; either `MSBFIRST` or `LSBFIRST`. (Most Significant Bit First, or, Least Significant Bit First)

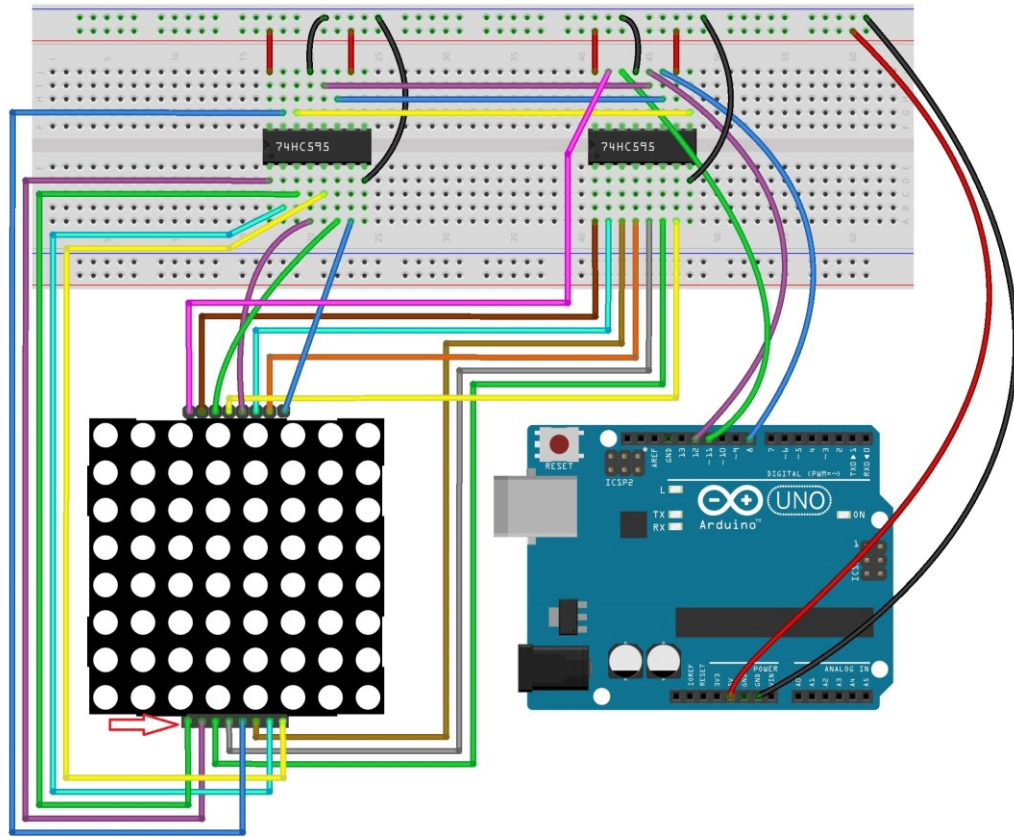
`value`: the data to shift out. (byte)

#### Returns

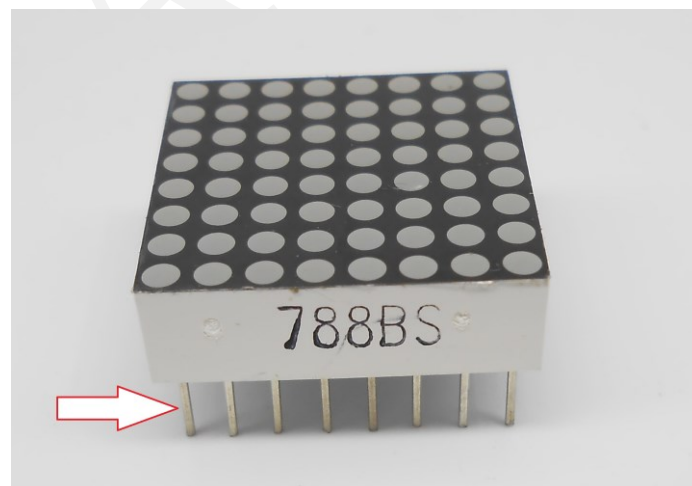
None

## Procedures

1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)

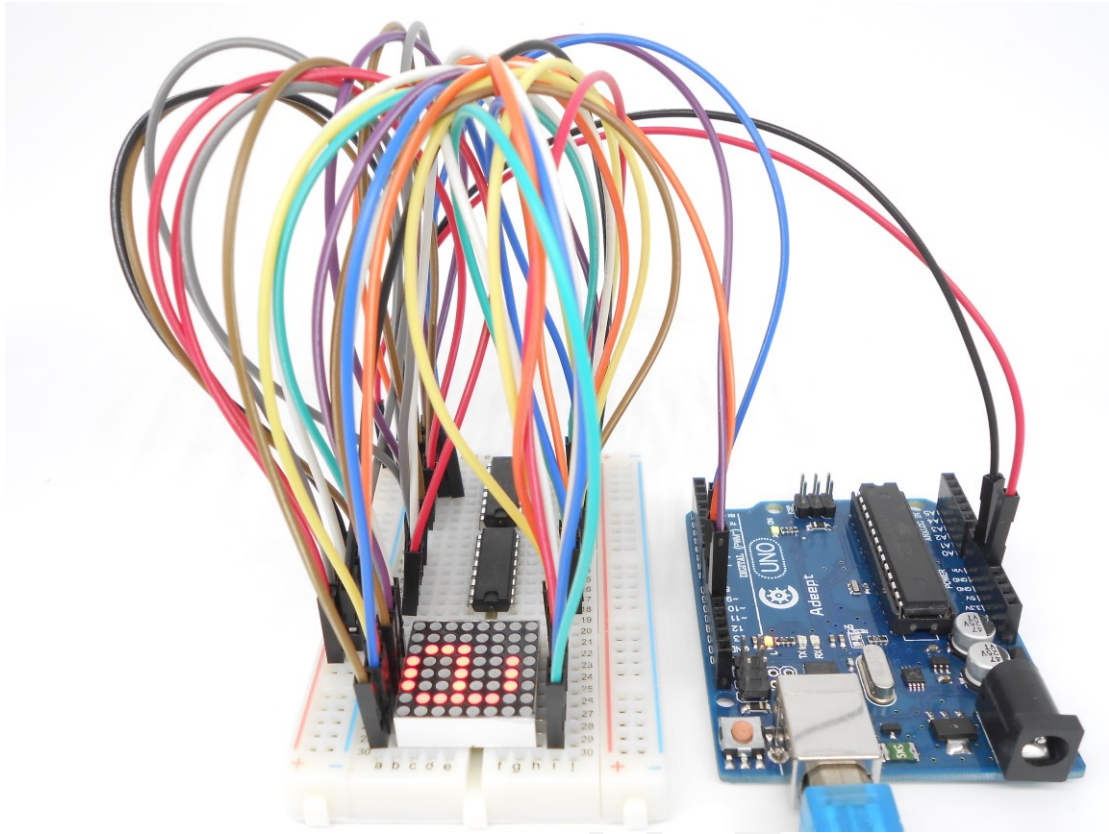


fritzing



2. Program
3. Compile the program and upload to Arduino UNO board

Now, you can see a rolling "Adept" should be displayed on the dot-matrix display.



## Summary

In this experiment, we have not only learned how to operate a dot-matrix display to display numbers and letters, but also learned the basic usage of 74HC595, then you can try operating the dot-matrix display to show other images.



# Lesson 26 Controlling Stepper Motor

## Overview

In this lesson, we will learn how to control a stepper motor.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Stepper Motor
- 1\* ULN2003 Driver Board
- Several Jumper Wires

## Principle

### 1. stepper motors

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".

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.

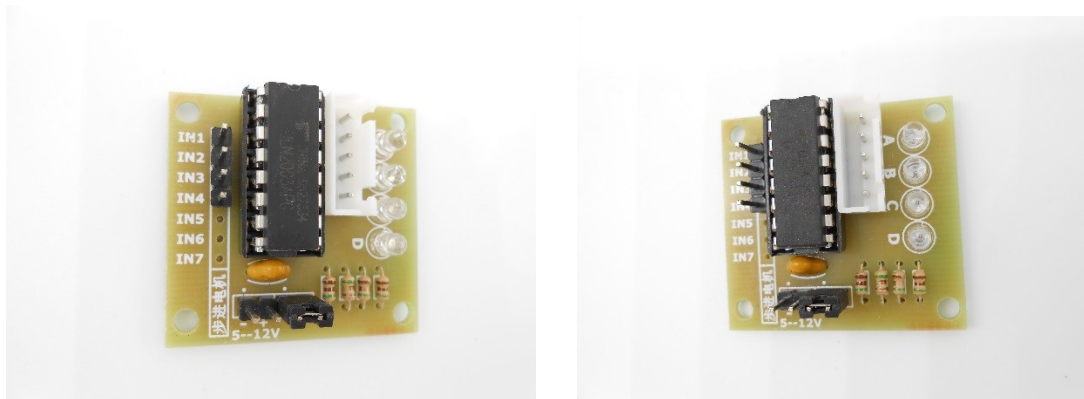


The stepping motor model that we provide is 28BYJ-48. Operating voltage is 5VDC. Phase is 4. The current is 92mA. The reduction ratio is 1/64. Using the eight-beat control the stepper motor.

### 2. ULN2003 driver board

Arduino UNO R3 board cannot directly drive stepper motors. A driver circuit is necessary, so we choose an ULN2003 driver board here as shown below. There

are four LEDs on the top. The white booth in the middle is connected to the stepper motor. The bottom is four pins used to connect with Arduino digital pins. When a pin is high, the corresponding LED will light up. The black jump hat on the right is power source input end. The driving method for stepper motor can be categorized as four-beat and eight-beat. In this experiment, we take eight-beat for example, for it is simple. You can drive the motor as long as you input HIGH to the four ports A, B, C and D in turn.



How many beats needed for the shaft to take a turn?

It is 360 degrees for the shaft to take a turn. In this experiment, we set that it takes 512 steps to take a turn. So each step will be  $360/512 = 0.7$  degrees.

## Procedures

### 1. Build the circuit

ULN2003 driver board IN1----- Arduino UNO R3 Digital pin 8

ULN2003 driver board IN2----- Arduino UNO R3 Digital pin 9

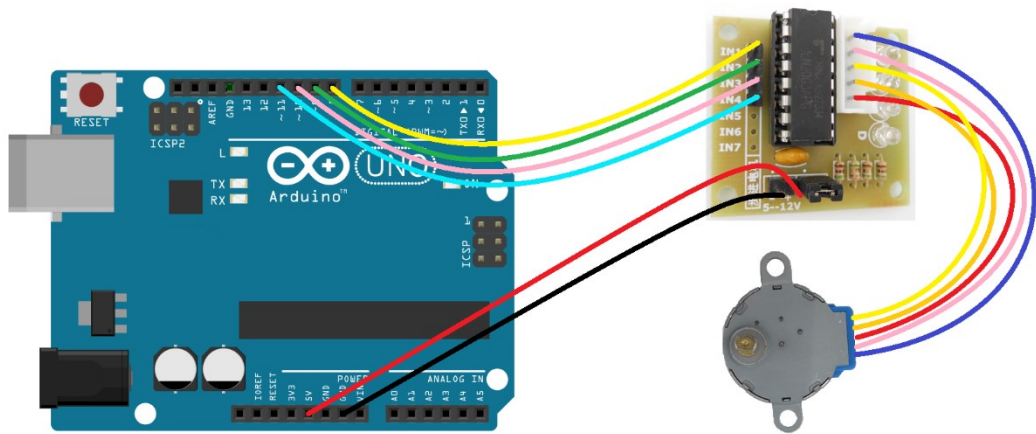
ULN2003 driver board IN3----- Arduino UNO R3 Digital pin 10

ULN2003 driver board IN4----- Arduino UNO R3 Digital pin 11

ULN2003 driver board "+" ----- Arduino UNO R3 Power pin 5V

ULN2003 driver board "-"----- Arduino UNO R3 Power pin GND



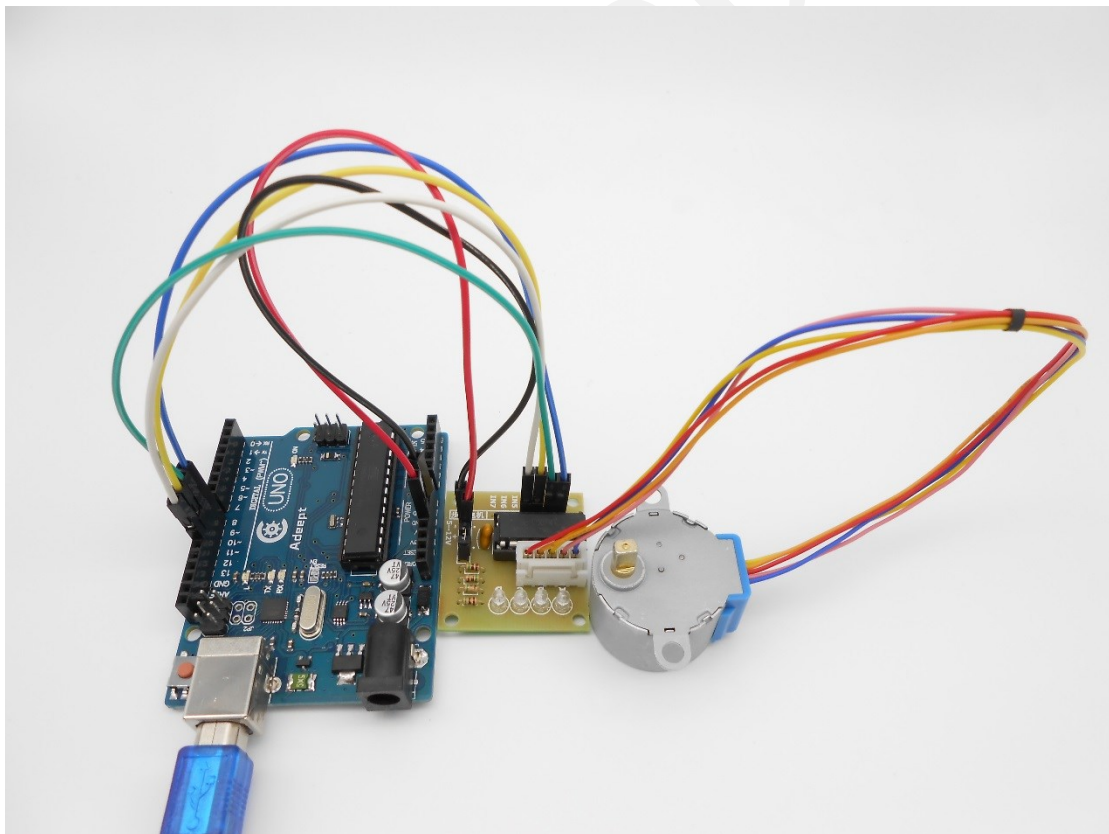


fritzing

## 2. Program

## 3. Compile the program and upload to Arduino UNO board

Now, the stepper motor can run fast in a clockwise circle, next the stepper motor can run slow in a counterclockwise circle.



# Lesson 27 Photoresistor

## Overview

In this lesson, we will learn how to measure the light intensity by photoresistor and make the measurement result displayed on the LCD1602.

## Requirement

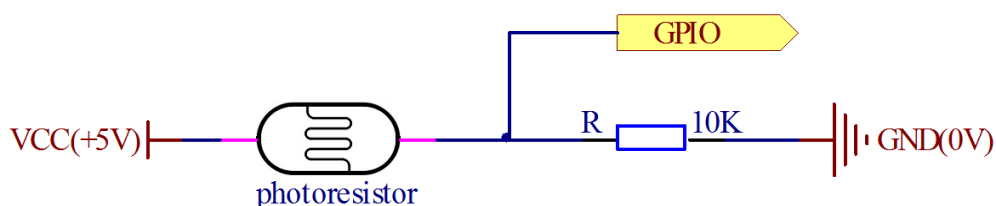
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* LCD1602
- 1\* Photoresistor
- 1\* 10K $\Omega$  Resistor
- 1\* 10K $\Omega$  Potentiometer
- 1\* Breadboard
- Several Jumper Wires

## Principle

A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with the increasing incident light intensity; in other words, it exhibits photoconductivity. A photoresistor can be applied in light-sensitive detector circuits.

A photoresistor is made of a high resistance semiconductor. In the dark, a photoresistor can have a resistance as high as a few megohms (M $\Omega$ ), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

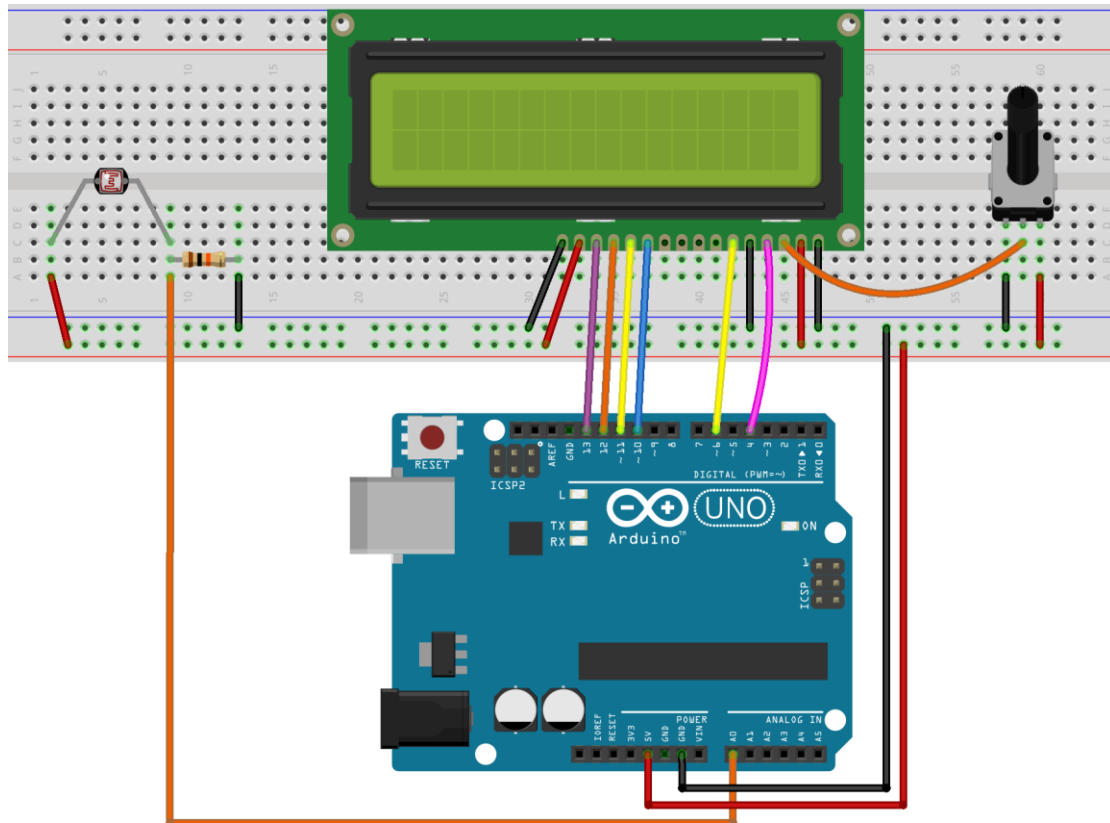
The schematic diagram of this experiment is shown below:



With the increase of the light intensity, the resistance of photoresistor will be decreased. The voltage of GPIO port in the above figure will become high.

## Procedures

### 1. Build the circuit

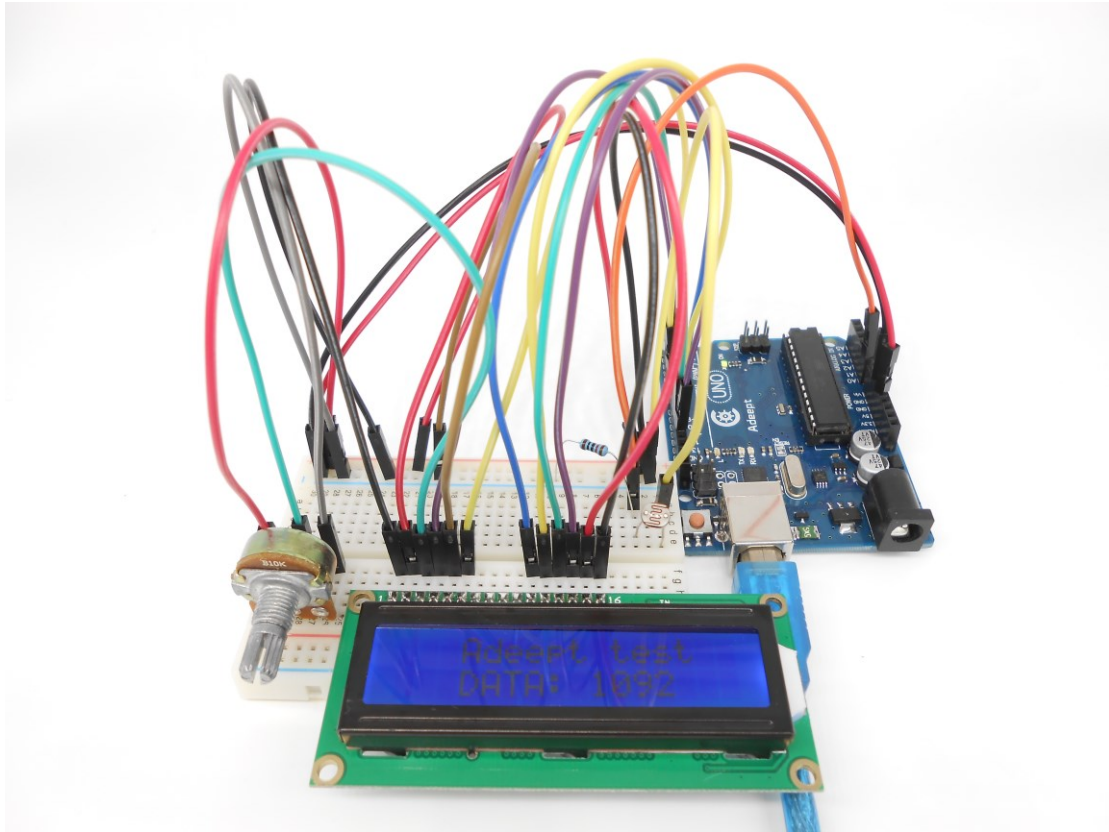


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

Now, when you try to block the light towards the photoresistor, you will find that the value displayed on the LCD1602 will be reduced. Otherwise, when you use a powerful light to irradiate the photoresistor, the value displayed on the LCD1602 will be increased.



## Summary

By learning this lesson, we have learned how to detect surrounding light intensity with the photoresistor. You can play your own wisdom, and make more originality based on this experiment and the former experiment.

# Lesson 28 Automatically Tracking Light Source

## Overview

In this lesson, we will make a light tracking system based on a servo and a photoresistor.

## Requirement

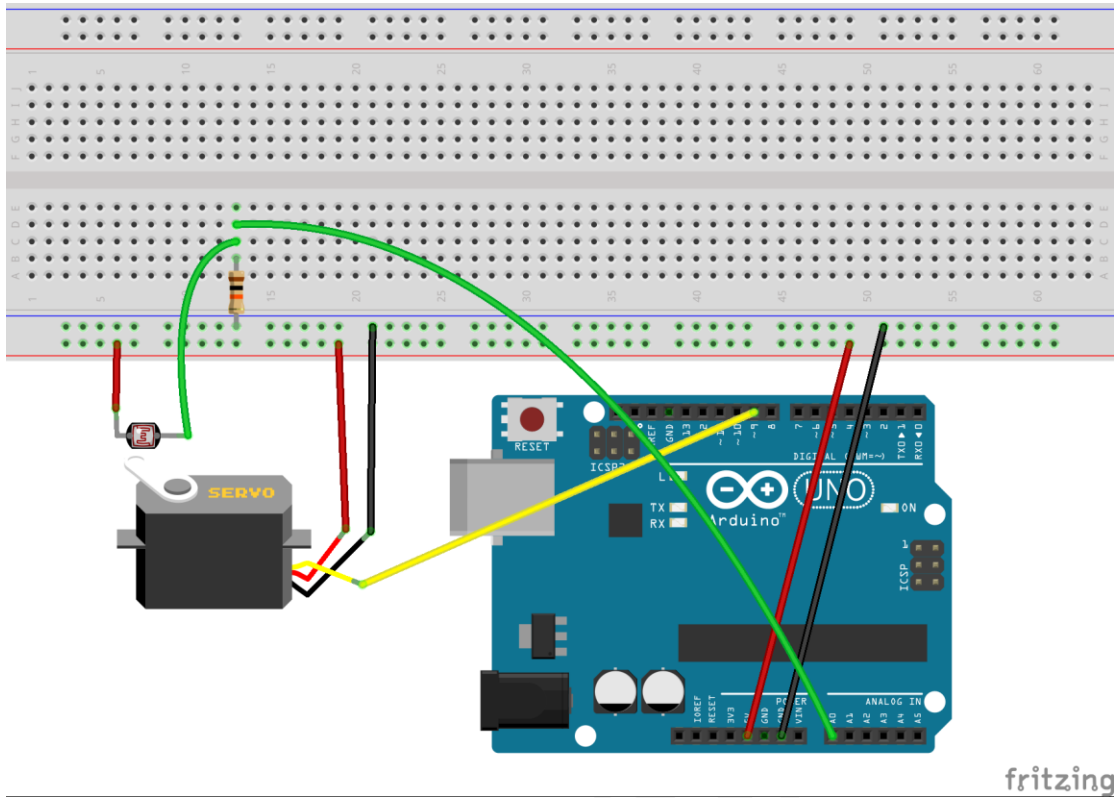
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Servo
- 1\* Photoresistor
- 1\* 10K $\Omega$  Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

In this experiment, we need to fasten the photoresistor with the horn of servo. First, we control the servo with a photoresistor to rotate from 0 ° to 180 ° to record the intensity of illumination, and then the photoresistor will stop at the brightest position.

## Procedures

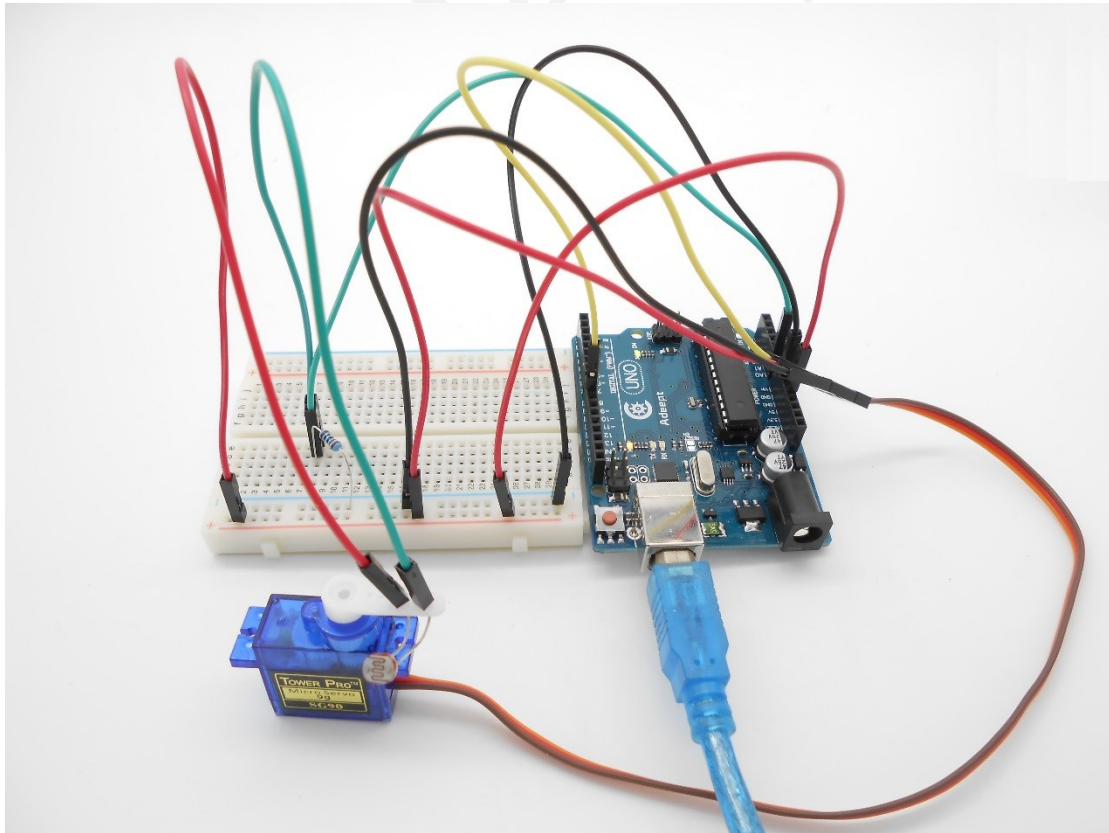
1. Build the circuit



fritzing

2. Program

3. Compile the program and upload to Arduino UNO board



# Lesson 29 Frequency meter

## Overview

In this lesson, we will make a simple frequency meter with the Arduino UNO. We will acquire the frequency of square wave which is generated by 555 timer, and then send the result to serial monitor through USB port.

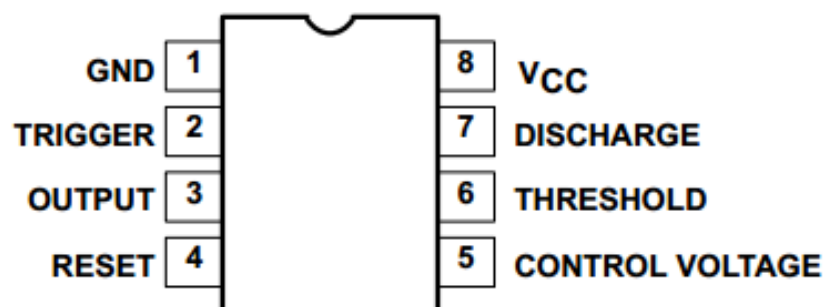
## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* NE555 Timer
- 1\* 10K $\Omega$  Resistor
- 1\* 10K $\Omega$  Potentiometer
- 2\* 104 Capacitor
- 1\* Breadboard
- Several Jumper Wires

## Principle

### 1. NE555

The 555 integrated circuit is originally used as a timer, and that is why it is called 555 timer or 555 time-base circuit. It is widely used in various electronic products because of its reliability, convenience and low price. There are dozens of components in the 555 integrated circuit, such as divider, comparator, basic R-S trigger, discharge tube, buffer and so on. It is a complex circuit and a hybrid composed of analog and digital circuit.

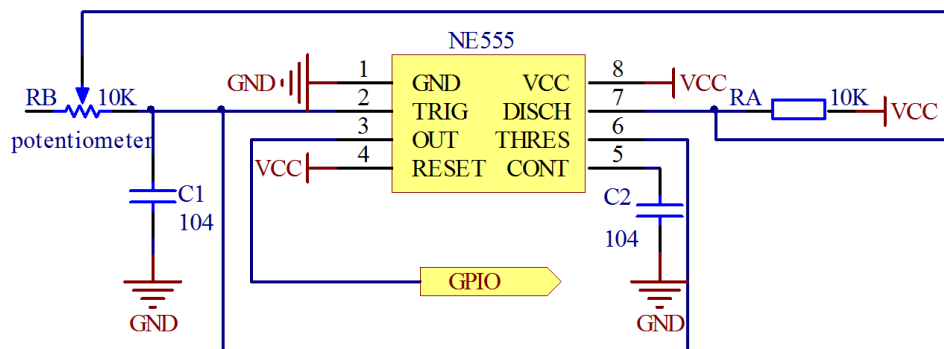


As shown in the above picture, the 555 integrated circuit is dual in-line with 8 pins package(DIP). Thereinto:



Pin 6 is the THRESHOLD for the input of upper comparator;  
 Pin 2 is TRIGGER for the input of lower comparator;  
 Pin 3 is the OUTPUT having two states of 0 and 1 decided by the input electrical level;  
 Pin 7, the DISCHARGE which has two states of suspension and ground connection also decided by input, is the output of the internal discharge tube;  
 Pin 4 is the RESET that outputs low level when supplied low voltage level;  
 Pin 5 is the CONTROL VOLTAGE that can change the upper and lower level trigger value;  
 Pin 8 (Vcc) is the power supply;  
 Pin 1(GND) is the ground.

The circuit schematic diagram used in the experiment is shown in below:



The circuit can generate a square wave signal that the frequency is adjustable. The frequency can be calculated by the formula:

$$\text{Frequency} \approx \frac{1.44}{(R_A + 2R_B) * C}$$

## 2. Key functions :

### ● pulseIn()

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds. Gives up and returns 0 if no pulse starts within a specified time out.

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

### Syntax

`pulseIn(pin, value)`



`pulseIn(pin, value, timeout)`

### Parameters

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

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

timeout (optional): the number of microseconds to wait for the pulse to start;

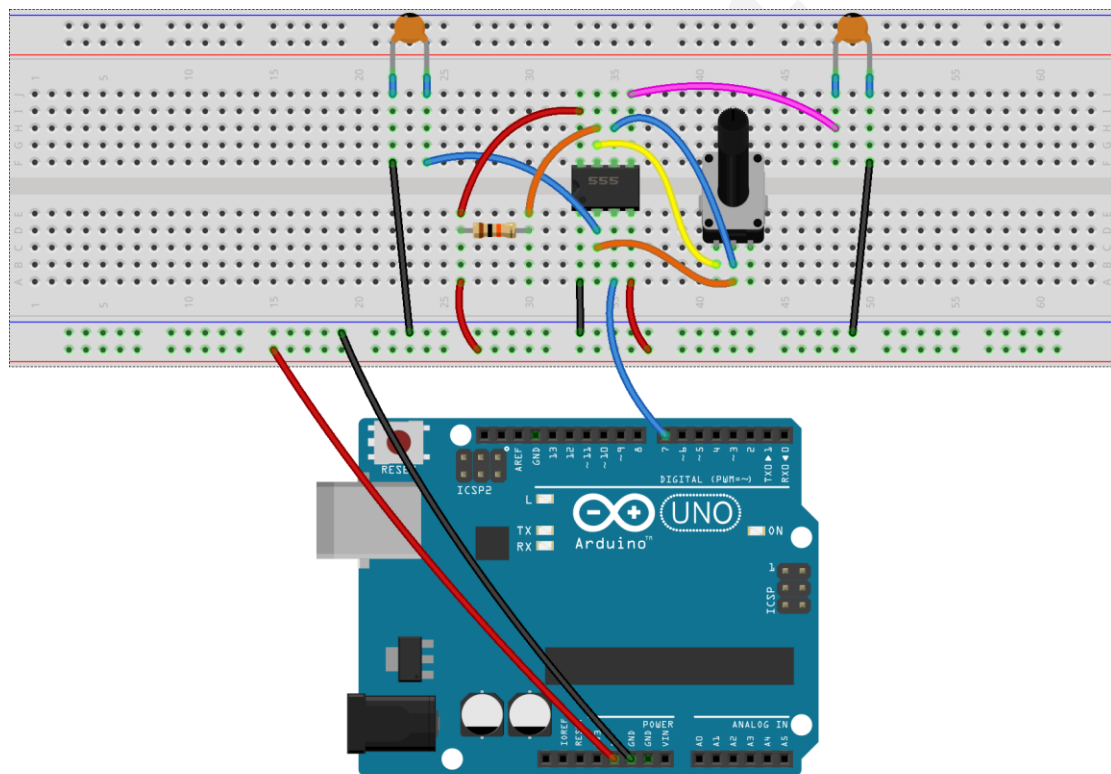
default is one second (unsigned long)

### Returns

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

## Procedures

1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)

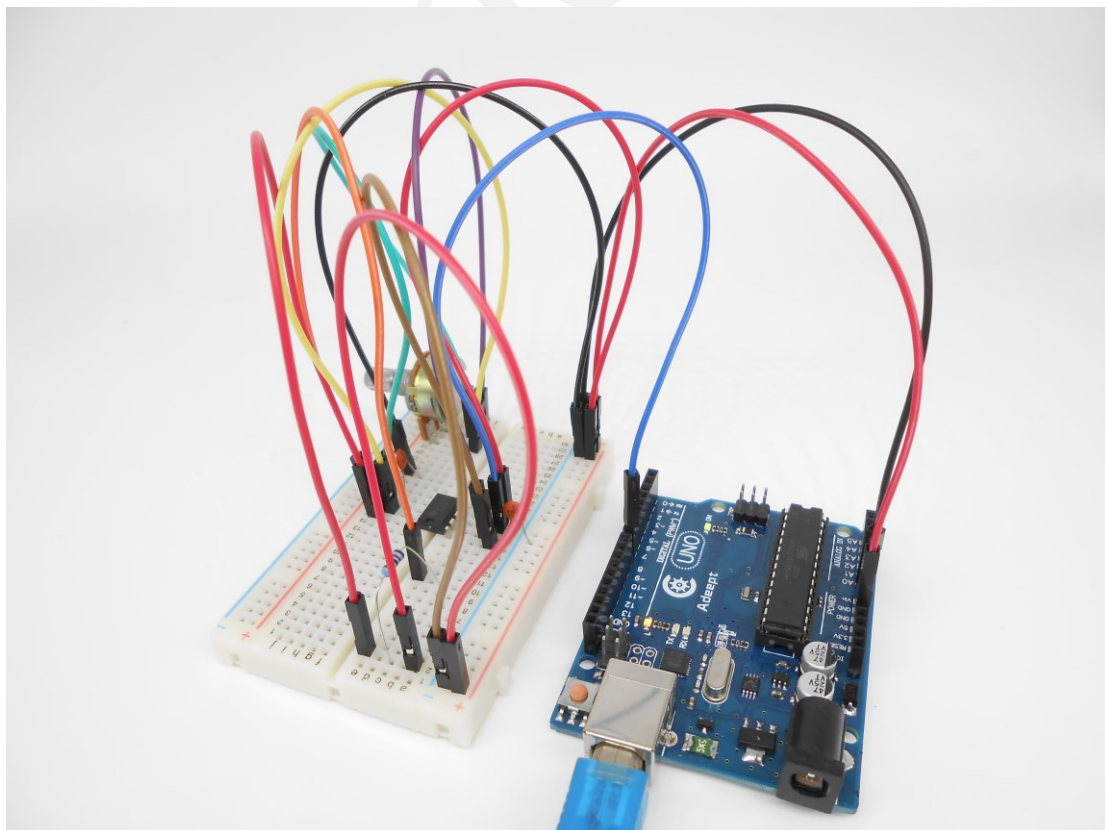
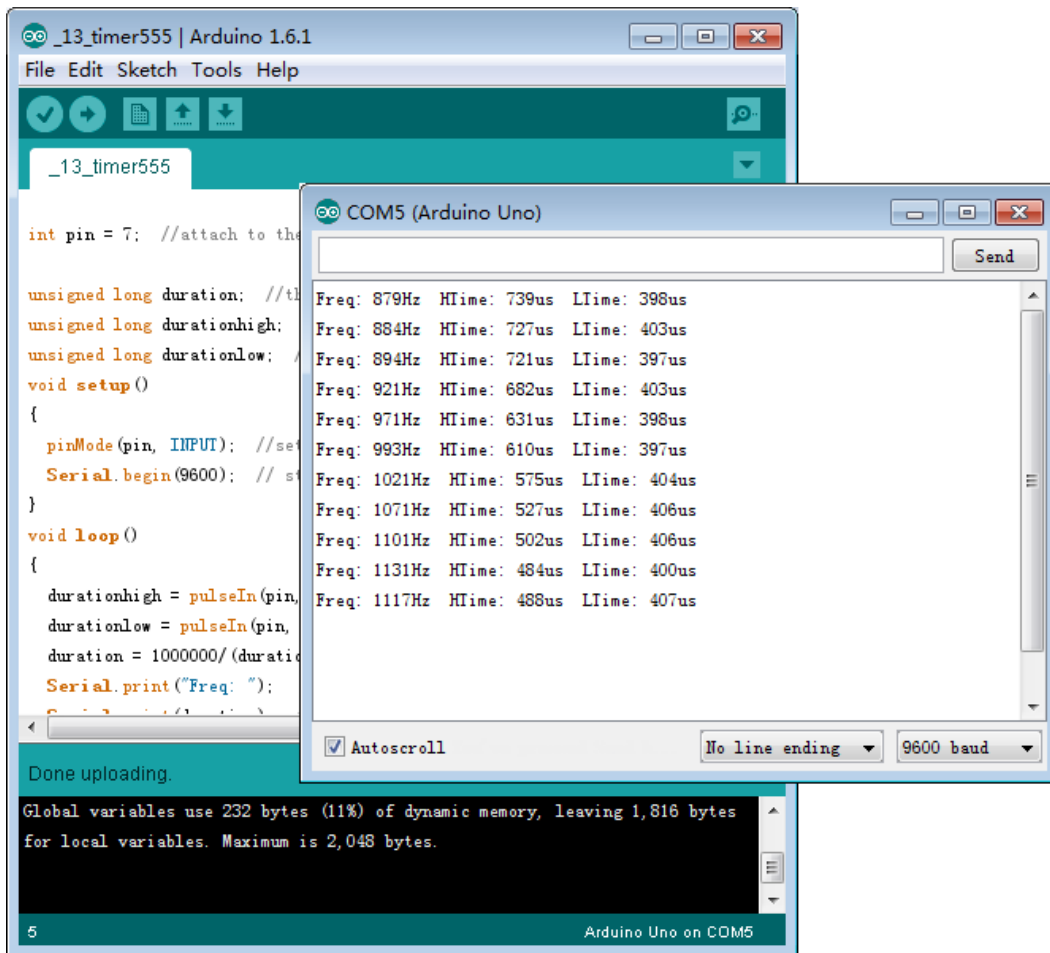


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

Now, when you rotating the potentiometer knob, the value of square wave's frequency printed on the serial monitor will be changed.



## Summary

By learning this lesson, I believe that you have mastered the principle of timer and you can make a simple frequency meter by yourself. You can display the value of square frequency to a LCD1602 by modifying the code we provided.

Adept

# Lesson 30 Intrusion Detection based on the PIR

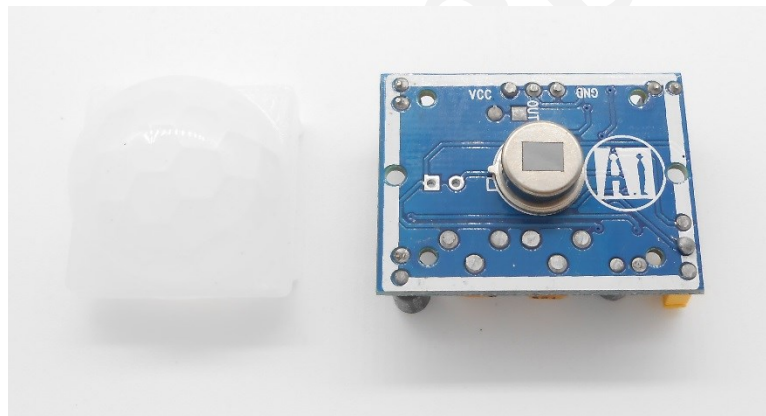
## Overview

In this lesson, we will learn how to use Passive Infrared (PIR) sensor to detect the movement nearby.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* PIR Movement Sensor
- 1\* 220 $\Omega$  Resistor
- 1\* LED
- Several Jumper Wires

## Principle



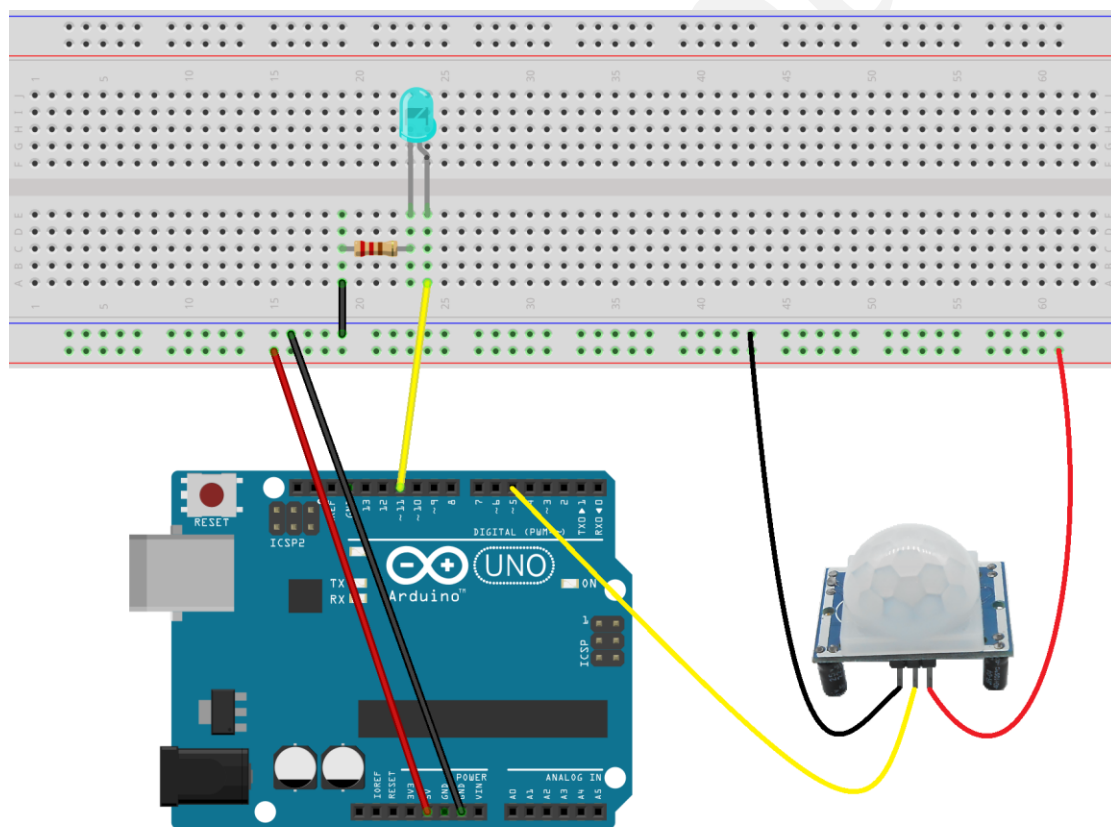
The sensor has pins marked "OUT", "-", and "+" (for Output, Gnd, and +5V).

PIR sensors respond to heat and can be triggered by animals such as cats and dogs, as well as by people and other heat sources. The 'output' pin of Passive Infrared (PIR) sensor will go HIGH when the motion has been detected.



## Procedures

### 1. Build the circuit

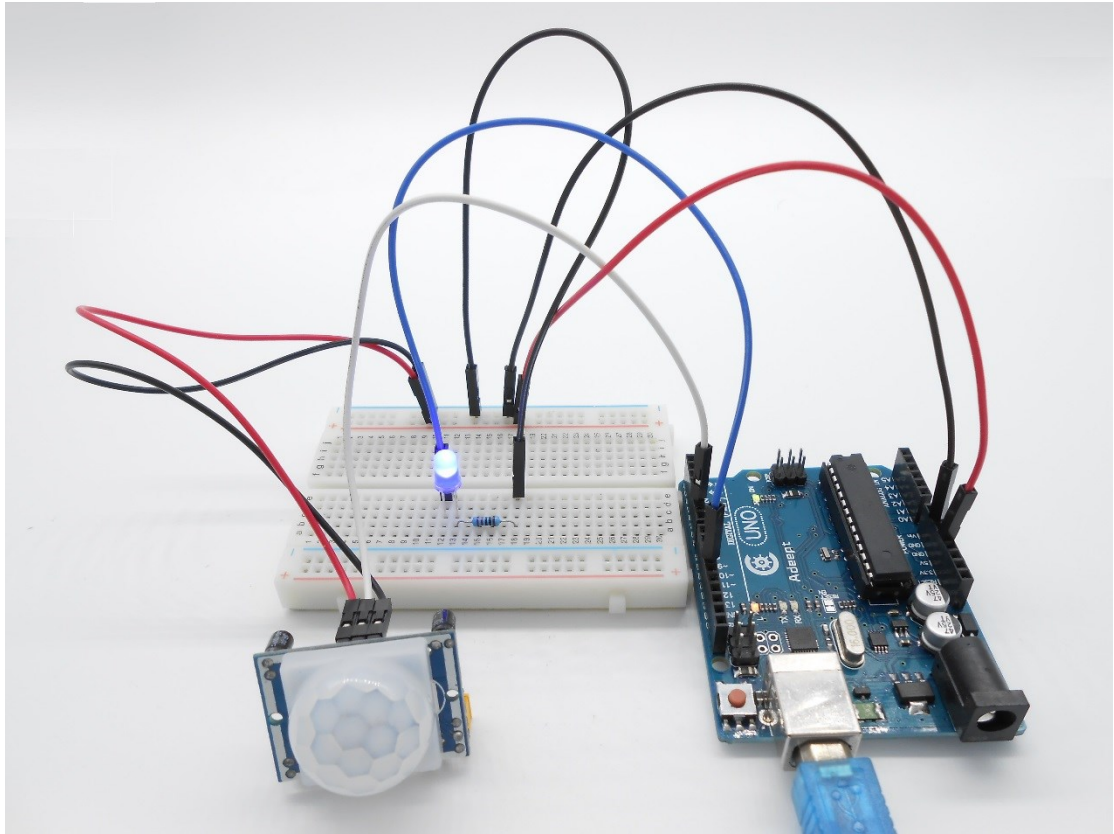


fritzing

### 2. Program

### 3. Compile the program and upload to Arduino UNO board

The LED will be turned on when the motion has been detected.



Adeet

# Lesson 31 Control a relay with IR remoter controller

## Overview

In this experiment, we will program the Arduino UNO to control a relay by remote controller.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* IR Receiver HX1838
- 1\* Remote Controller
- 1\* NPN Transistor (S8050)
- 1\* 1K Resistor
- 1\* 220 $\Omega$  Resistor
- 1\* 1N4001 Diode
- 1\* Relay
- 1\* LED
- 1\* Breadboard
- Several Jumper Wires

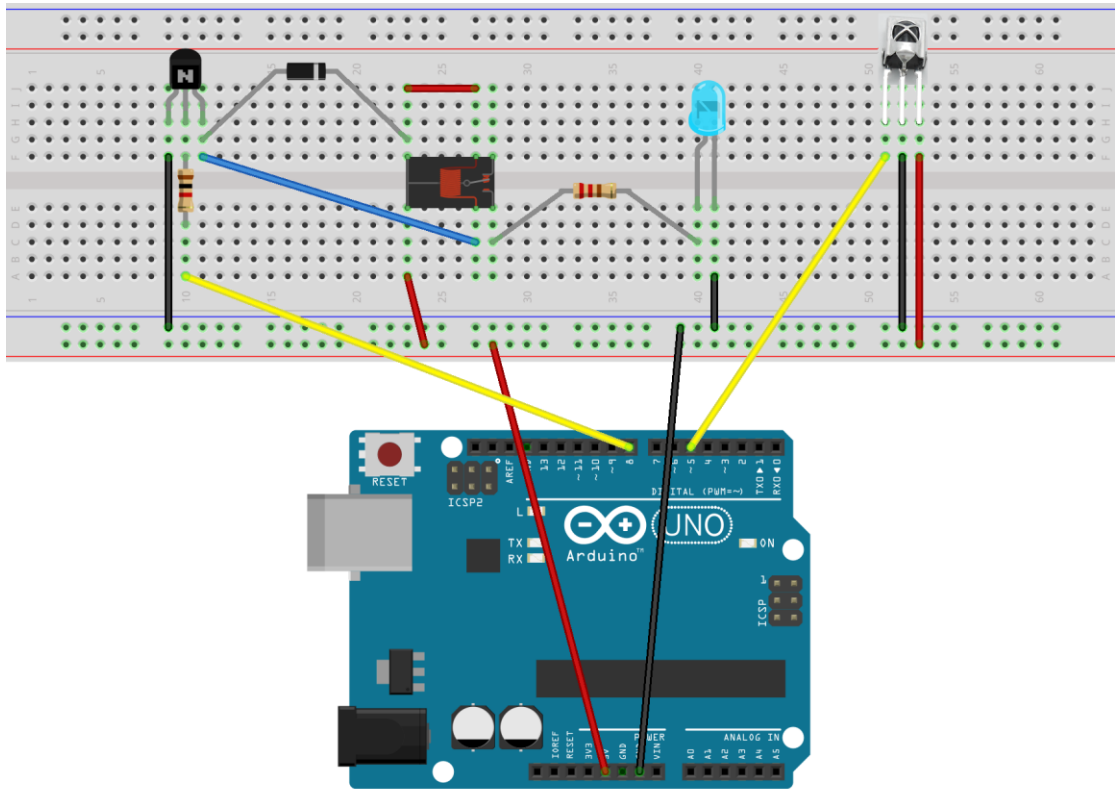
## Principle

The remote IR receiver connected to the Arduino UNO is used to receive IR signal from remote controller. If you press the different keys(key 0 or key 1) on the remote controller, the relay state will be toggled.

## Procedures

1. Build the circuit



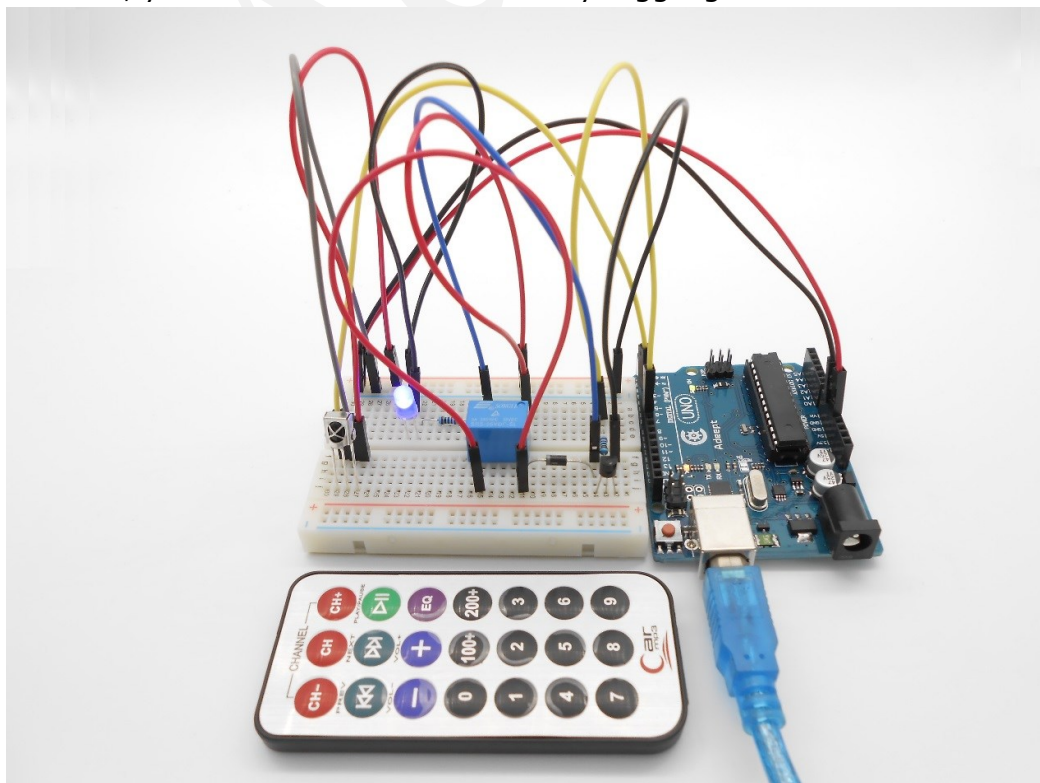


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

Now, when you press the button '0' on the remote controller, the LED is off. When you press the button '1' on the remote controller, the LED is on. At the same time, you will hear the sound of relay toggling.





# Lesson 32 Control a RGB LED with IR remoter controller

## Overview

In this lesson, we will use the remote IR receiver and the RGB to do an experiment that the infrared remote controls the RGB.

## Requirement

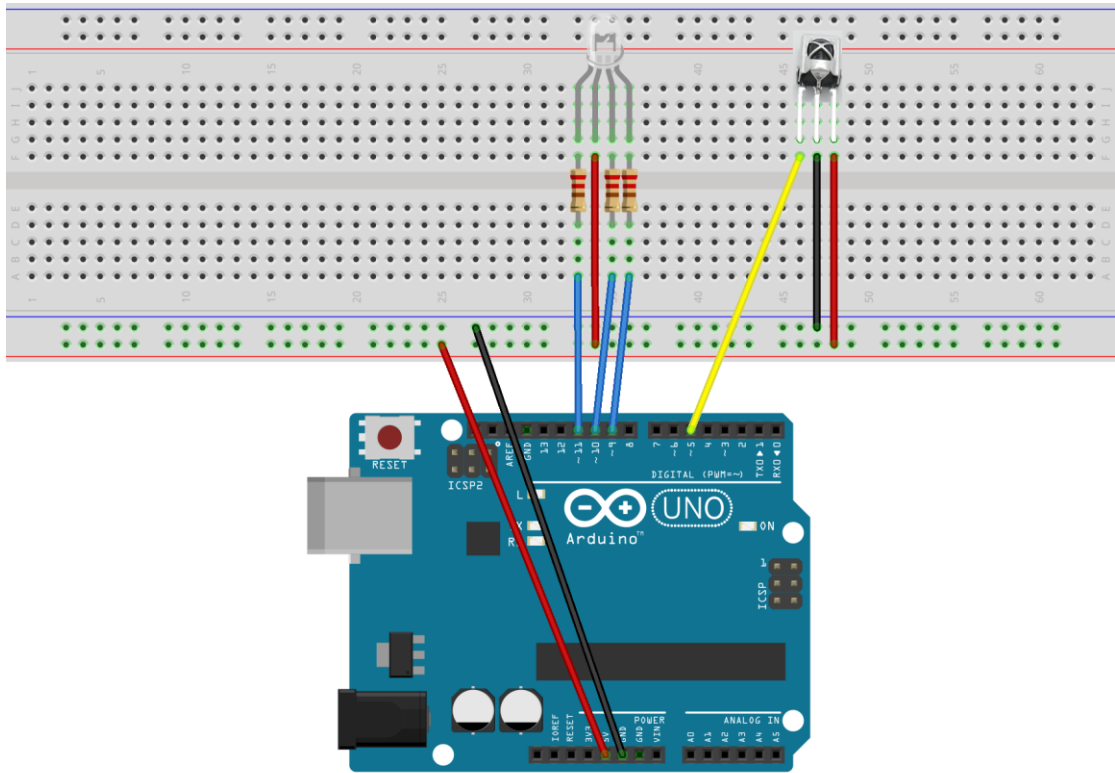
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* IR Receiver HX1838
- 1\* Remote Controller
- 1\* RGB LED
- 3\* 220Ω Resistor
- 1\* Breadboard
- Several Jumper Wires

## Principle

If you press the different key with different number on the remote controller, you will find the color of RGB LED will be changed. When you press the key with number 0, the RGB LED will be off.

## Procedures

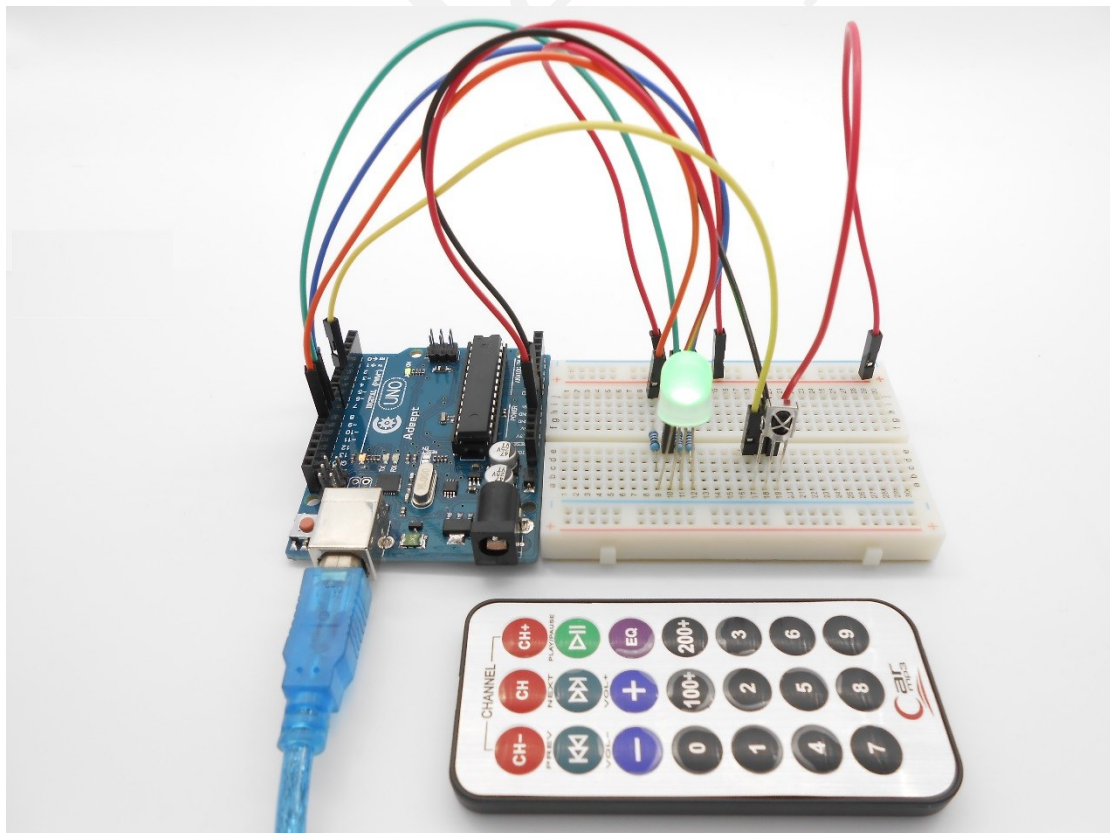
1. Build the circuit



fritzing

2. Program

3. Compile the program and upload to Arduino UNO board



# Lesson 33 control a stepper motor with IR remoter controller

## Overview

In this lesson, we will learn how to control a stepper based on a remote controller.

## Requirement

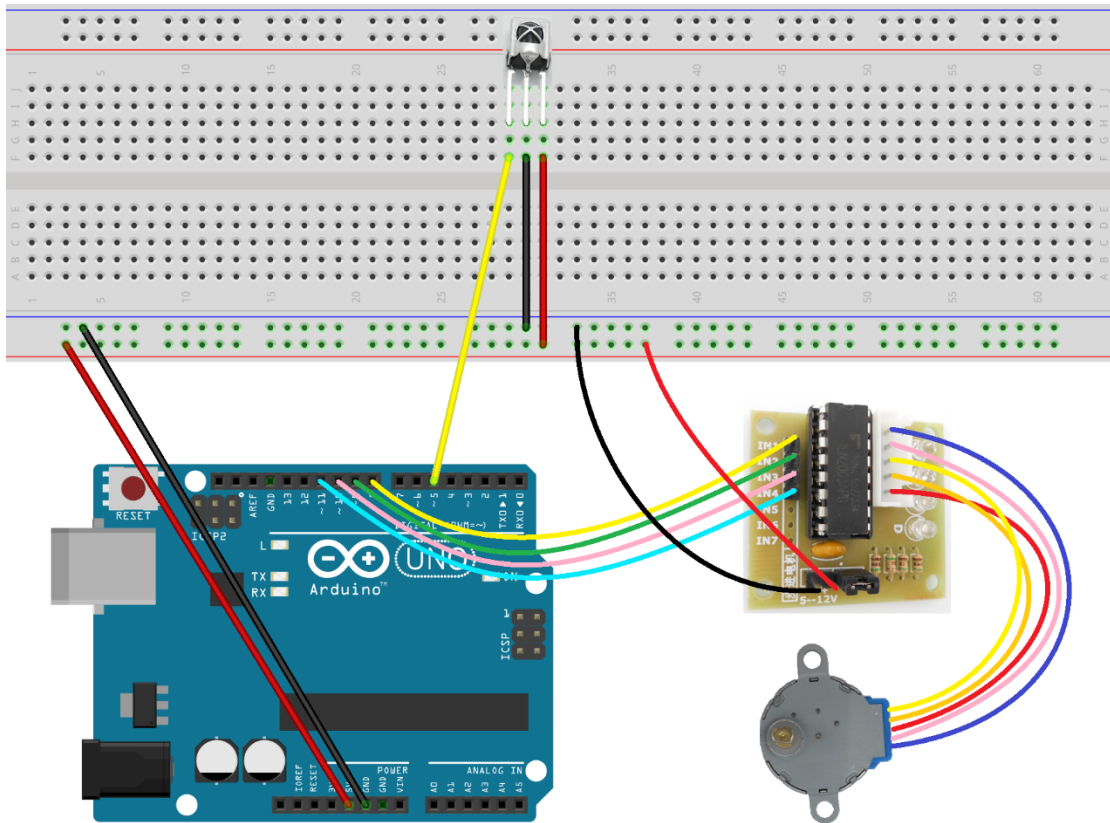
- 1\* Arduino UNO
- 1\* USB Cable
- 1\* IR Receiver HX1838
- 1\* Remote Controller
- 1\* Stepper Motor
- 1\* ULN2003 Driver Board
- 1\* Breadboard
- Several Jumper Wires

## Principle

If you press the different key on the remote controller, the different signal will be sent to the Arduino UNO, and then the Arduino can tell the stepper motor how many steps they need run and what direction. For example, when you press the key '+', the stepper motor will run clockwise, when you press the key '-', the stepper motor will run counterclockwise. When you press the key '2', the stepper motor will rotate two laps, when you press the key '5', the stepper motor will rotate five laps.

## Procedures

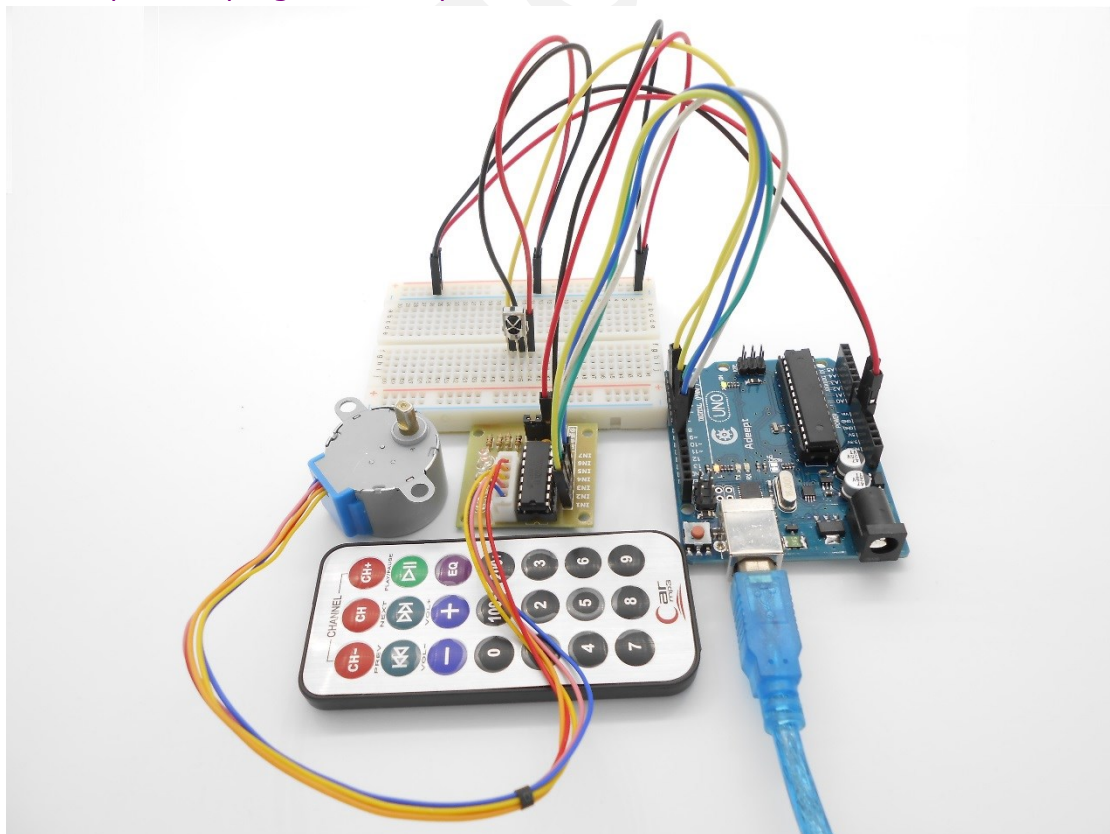
1. Build the circuit



fritzing

2. Program

3. Compile the program and upload to Arduino UNO board



# Lesson 34 Controlling the size of the circle by potentiometer

## Overview

In this lesson, we will collect the potentiometer data by programming Arduino UNO Board, and we will send the data to the processing through the serial communication to change the size of a circle.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* 10K $\Omega$  Potentiometer
- 1\* Breadboard
- Several Jumper Wires

## Principle

The experiment consists of two parts, the first is used to acquire the data from Arduino, another is the used to process the data.

### *Arduino key function:*

#### ● write()

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the print() function instead.

Syntax

`Serial.write(val)` `Serial.write(str)` `Serial.write(buf, len)`

Parameters

val: a value to send as a single byte

str: a string to send as a series of bytes

buf: an array to send as a series of bytes

len: the length of the buffer

Returns

byte

write() will return the number of bytes written, though reading that number is

optional

**Processing key function:**

● **Name:** `size()`

*Description*

Defines the dimension of the display window in units of pixels. The `size()` function must be the first line of code, or the first code inside `setup()`. Any code that appears before the `size()` command may run more than once, which can lead to confusing results.

The system variables `width` and `height` are set by the parameters passed to this function. If `size()` is not used, the window will be given a default size of 100x100 pixels.

*Syntax*

`size(w, h)`

`size(w, h, renderer)`

*Parameters*

**w** int: width of the display window in units of pixels

**h** int: height of the display window in units of pixels

renderer

String: Either P2D, P3D, or PDF

*Returns*

void

● **Name:** `println()`

*Description*

The `println()` function writes to the console area, the black rectangle at the bottom of the Processing environment. This function is often helpful for looking at the data a program is producing. Each call to this function creates a new line of output. More than one parameter can be passed into the function by separating them with commas. Alternatively, individual elements can be separated with quotes (") and joined with the addition operator (+).

Before Processing 2.1, `println()` was used to write array data to the console.

Now, use `printArray()` to write array data to the console.

Note that the console is relatively slow. It works well for occasional messages, but does not support high-speed, real-time output (such as at 60 frames per second).

*Syntax*

`println()`

`println(what)`

`println(variables)`

*Parameters*

**what** Object, String, float, char, boolean, or byte: data to print to console

**variables** Object[]: list of data, separated by commas

*Returns*

void

● **Name:** `background()`

*Description*

The `background()` function sets the color used for the background of the Processing window. The default background is light gray. This function is typically used within `draw()` to clear the display window at the beginning of each frame, but it can be used inside `setup()` to set the background on the first frame of animation or if the background need only be set once.

An image can also be used as the background for a sketch, although the image's width and height must match that of the sketch window. Images used with `background()` will ignore the current `tint()` setting. To resize an image to the size of the sketch window, use `image.resize(width, height)`.

It is not possible to use the transparency alpha parameter with background colors on the main drawing surface. It can only be used along with a `PGraphics` object and `createGraphics()`.

*Syntax*

`background(rgb)`

`background(rgb, alpha)`

`background(gray)`

`background(gray, alpha)`

`background(v1, v2, v3)`

`background(v1, v2, v3, alpha)`

`background(image)`

### *Parameters*

**rgb** int: any value of the color datatype

**alpha** float: opacity of the background

**gray** float: specifies a value between white and black

**v1** float: red or hue value (depending on the current color mode)

**v2** float: green or saturation value (depending on the current color mode)

**v3** float: blue or brightness value (depending on the current color mode)

**image** PImage: PImage to set as background (must be same size as the sketch window)

### *Returns*

void

● **Name:** `fill()`

### *Description*

Sets the color used to fill shapes. For example, if you run `fill(204, 102, 0)`, all subsequent shapes will be filled with orange. This color is either specified in terms of the RGB or HSB color depending on the `currentColorMode()`. (The default color space is RGB, with each value in the range from 0 to 255.)

When using hexadecimal notation to specify a color, use `"#"` or `"0x"` before the values (e.g., `#CCFFAA` or `0xFFCCFFAA`). The `#` syntax uses six digits to specify a color (just as colors are typically specified in HTML and CSS). When using the hexadecimal notation starting with `"0x"`, the hexadecimal value must be specified with eight characters; the first two characters define the alpha component, and the remainder define the red, green, and blue components.

The value for the `"gray"` parameter must be less than or equal to the current



maximum value as specified by `colorMode()`. The default maximum value is 255.

### *Syntax*

`fill(rgb)`

`fill(rgb, alpha)`

`fill(gray)`

`fill(gray, alpha)`

`fill(v1, v2, v3)`

`fill(v1, v2, v3, alpha)`

### *Parameters*

**rgb** int: color variable or hex value

**alpha** float: opacity of the fill

**gray** float: number specifying value between white and black

**v1** float: red or hue value (depending on current color mode)

**v2** float: green or saturation value (depending on current color mode)

**v3** float: blue or brightness value (depending on current color mode)

### *Returns*

void

● [Name: ellipse\(\)](#)

### *Description*

Draws an ellipse (oval) to the screen. An ellipse with equal width and height is a circle. By default, the first two parameters set the location, and the third and fourth parameters set the shape's width and height. The origin may be changed with the `ellipseMode()` function.

### *Syntax*

`ellipse(a, b, c, d)`

### *Parameters*

**a** float: x-coordinate of the ellipse

- b** float: y-coordinate of the ellipse
- c** float: width of the ellipse by default
- d** float: height of the ellipse by default

Returns

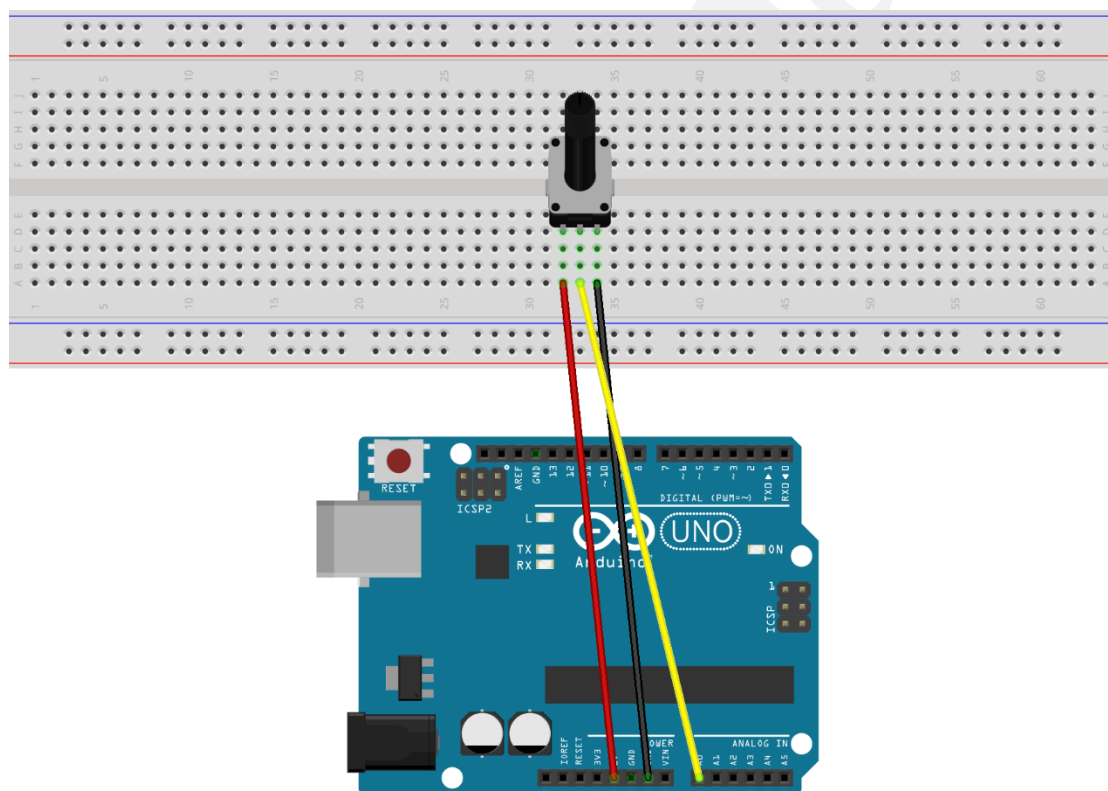
Void

**Note:**

1. In this experiment, my Arduino UNO board is connected to my computer COM26, please adjust according to actual situation.
2. If Processing has tips that you need to install the related function library, please install.

**Procedures**

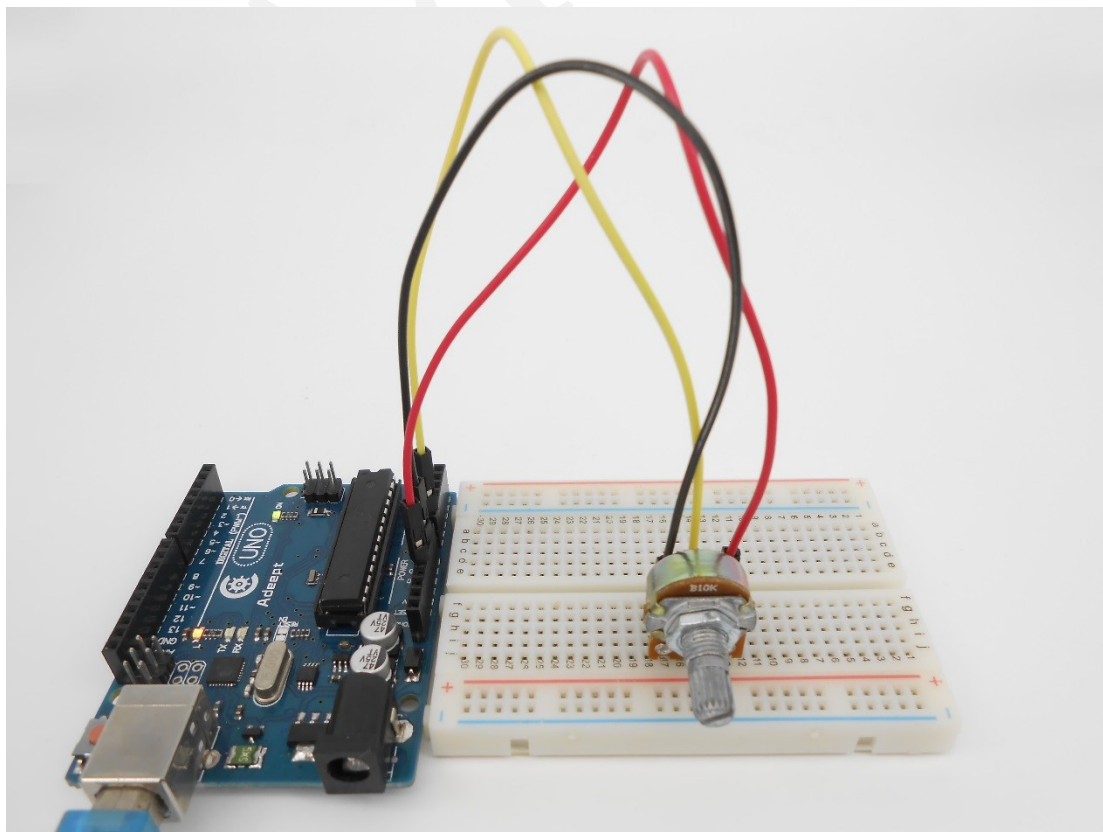
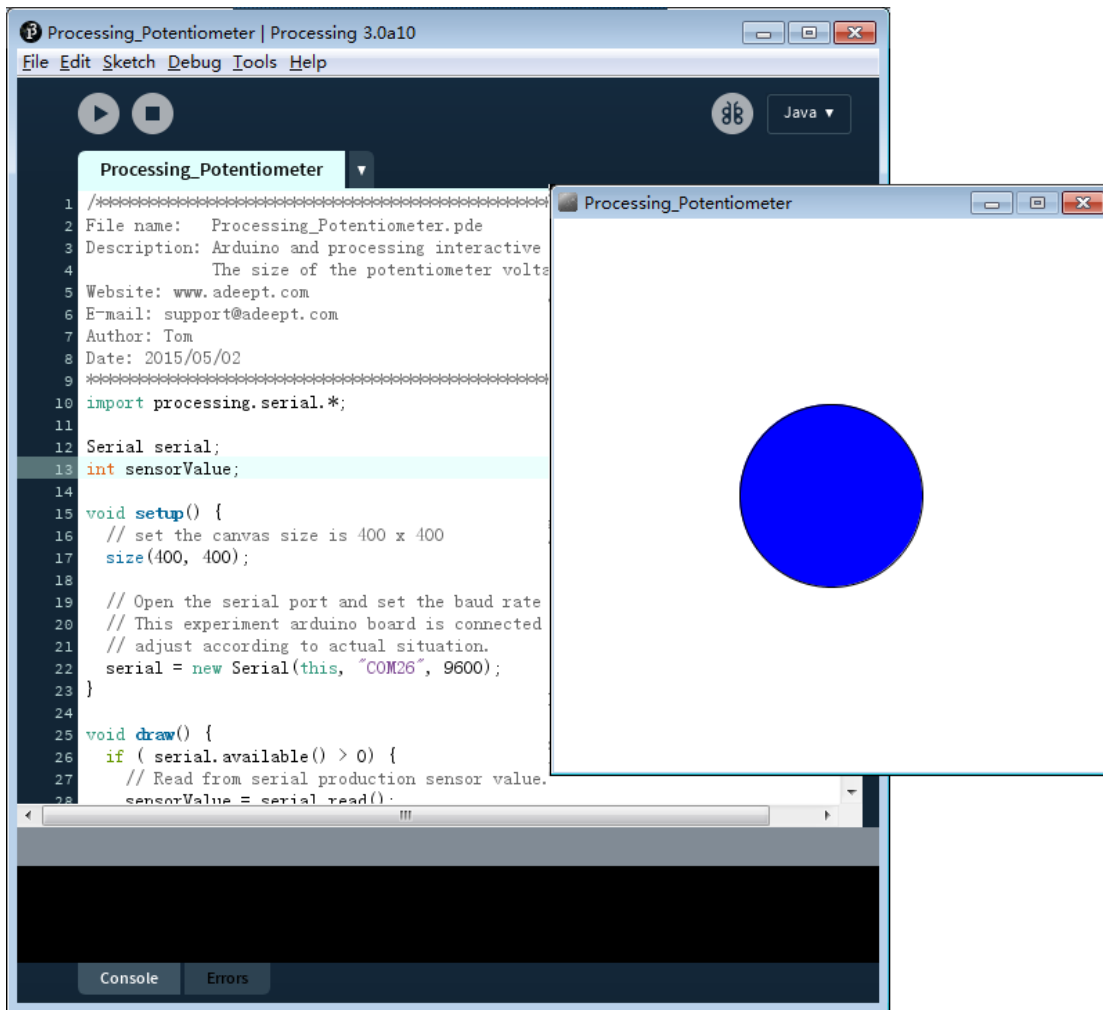
1. Build the circuit



fritzing

2. Program
3. Compile the program and upload to Arduino UNO board
4. Run processing software(Processing\_Potentiometer.pde)

Now, when you turn potentiometer, you will see a blue circle size to change on the computer.



# Lesson 35 Controlling the 3D model by PS2 Joystick

## Overview

In this lesson, we will collect the state of joy stick by programming Arduino UNO Board, and we will send the data to the processing through the serial communication.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* PS2 Joystick
- 1\* Breadboard
- Several Jumper Wires

## Principle

The experiment consists of two parts, the first is used to acquire data from the Arduino, section, another is used to process the data.

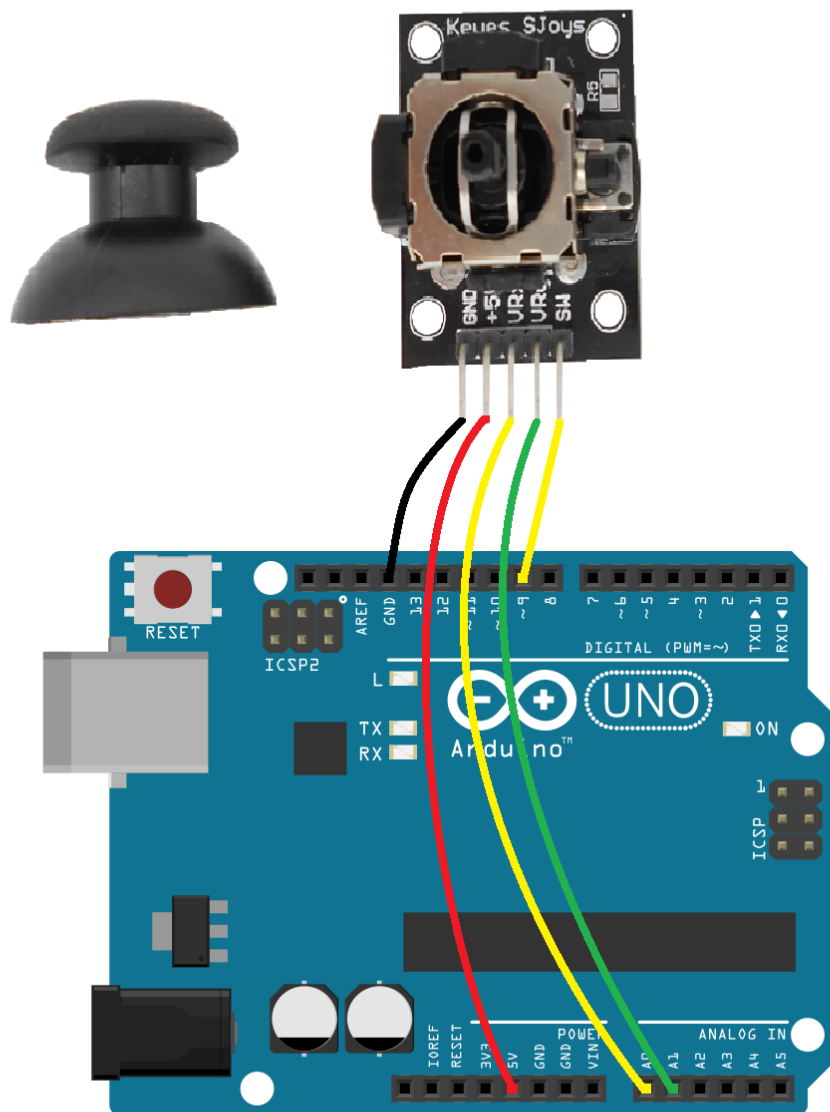
We use Arduino UNO board to collect the joystick state data, and upload the data to the computer through the serial port. The data will be processed by Processing and shown with 3D image.

### Note:

1. In this experiment, my Arduino UNO board is connected to my computer COM26, please adjust according to actual situation.
2. If the Processing has not running normally, you need to install the related function libraries.

## Procedures

1. Build the circuit



fritzing

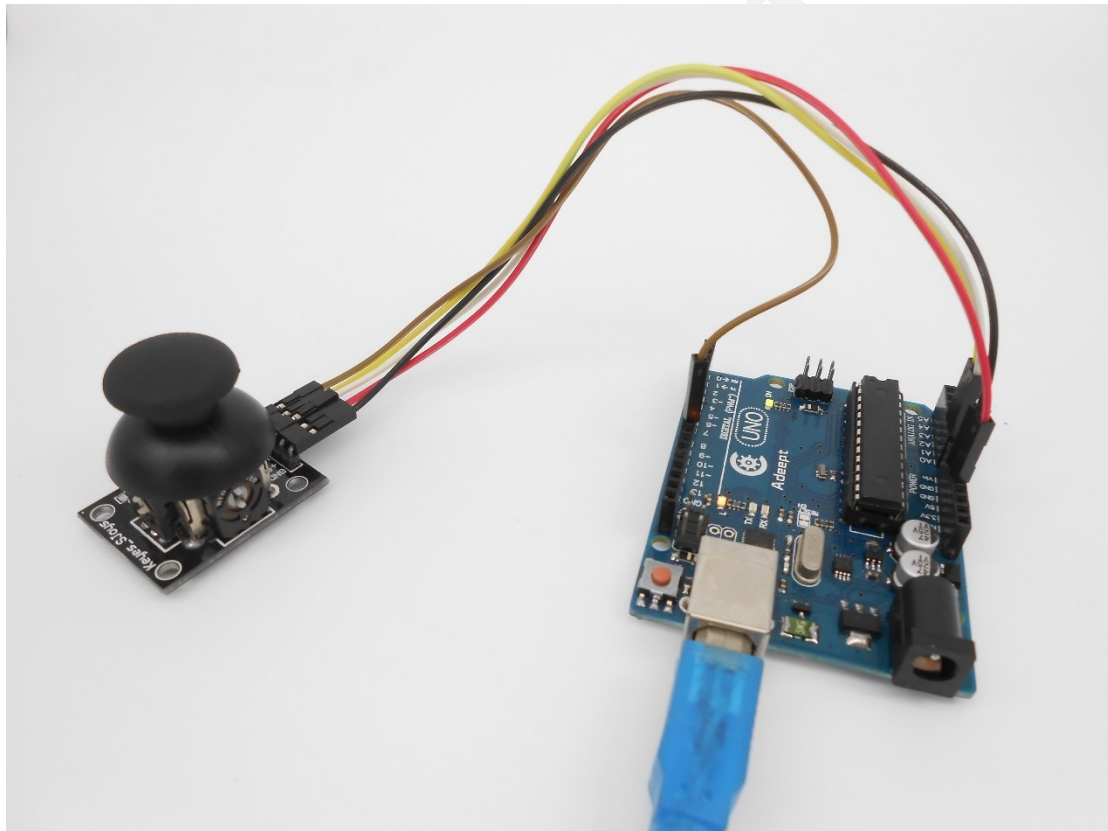
2. Program
3. Compile the program and upload to Arduino UNO board
4. Run processing software (Processing\_PS2Joystick.pde)

When you operate the joystick, the 3D model will be following operation on your computer.

```
Processing_PS2Joystick | Processing 3.0a10
File Edit Sketch Debug Tools Help

Processing_PS2Joystick
void draw() {
  background(0); //Set Background Color
  image(img, 450, 450);
  lights(); //Open lights
  readSensors(); //Read 2-Axis value
  fill(255, 0, 0); //Set the fill color
  textFont(font, 30); //Set the font size
  text("ANGLE : \n + xval: "+Rw[0]+" \n + yval: "+Rw[1]+" \n + swval: "+Rw[2], 50, 50);
  if(Rw[2]==1) {
    fill(255, 0, 255); //Set the fill color
  } else {
    fill(0, 255, 255); //Set the fill color
  }
  translate(-Rw[1]/4+300, -Rw[0]/4+300, 0); //Settings Transfer Coordinates
  box(50, 50, 250); //Draw a box 300 * 300 * 40
}

void readSensors() {
  if(myPort.available() > 0) {
    if(myPort.readBytesUntil(10, inBuffer) > 0) { //Read to determine whether the
      String inputString = new String(inBuffer);
      String inputStringArr[] = split(inputString, ','); //Data ', ' Split
      Rw[0] = int(inputStringArr[0]); //Read the X value
      Rw[1] = int(inputStringArr[1]); //Read the y value
      Rw[2] = int(inputStringArr[2]);
      Rw[0] = 515 - Rw[0]; //Rocker midpoint value 515 into 0
      Rw[1] = Rw[1] - 515; //Converted to negative (rocker line out)
    }
  }
}
```



# Lesson 36 Snake Game

## Overview

In this lesson, we will make a Snake Game based on the Processing, and play the game with two buttons.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 2\* Button
- 1\* Breadboard
- Several Jumper Wires

## Principle

The experiment consists of two parts, the first is used to acquire data from the Arduino UNO, another is used to process the data.

Play the Snake Game:

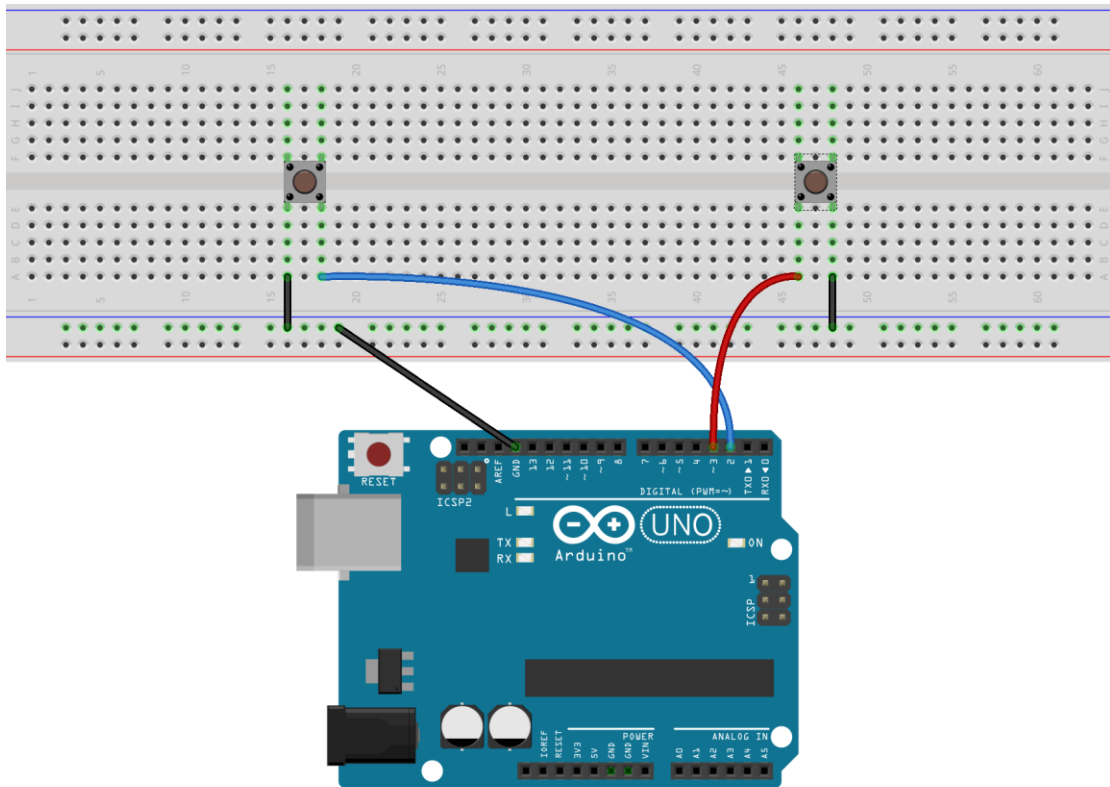
- ① When you press the right button, the snake will move to the right.
- ② When you press the left button, the snake will move to the left.

### **Note:**

1. You need to install the Sound library.
2. In this experiment, my Arduino UNO board is connected to my computer COM26, please adjust according to actual situation.
3. If the Processing has not running normally, you need to install the related function libraries.

## Procedures

1. Build the circuit



fritzing

2. Program
3. Compile the program and upload to Arduino UNO board
4. Run processing software (Snake\_Game\_Processing\_Button.pde)

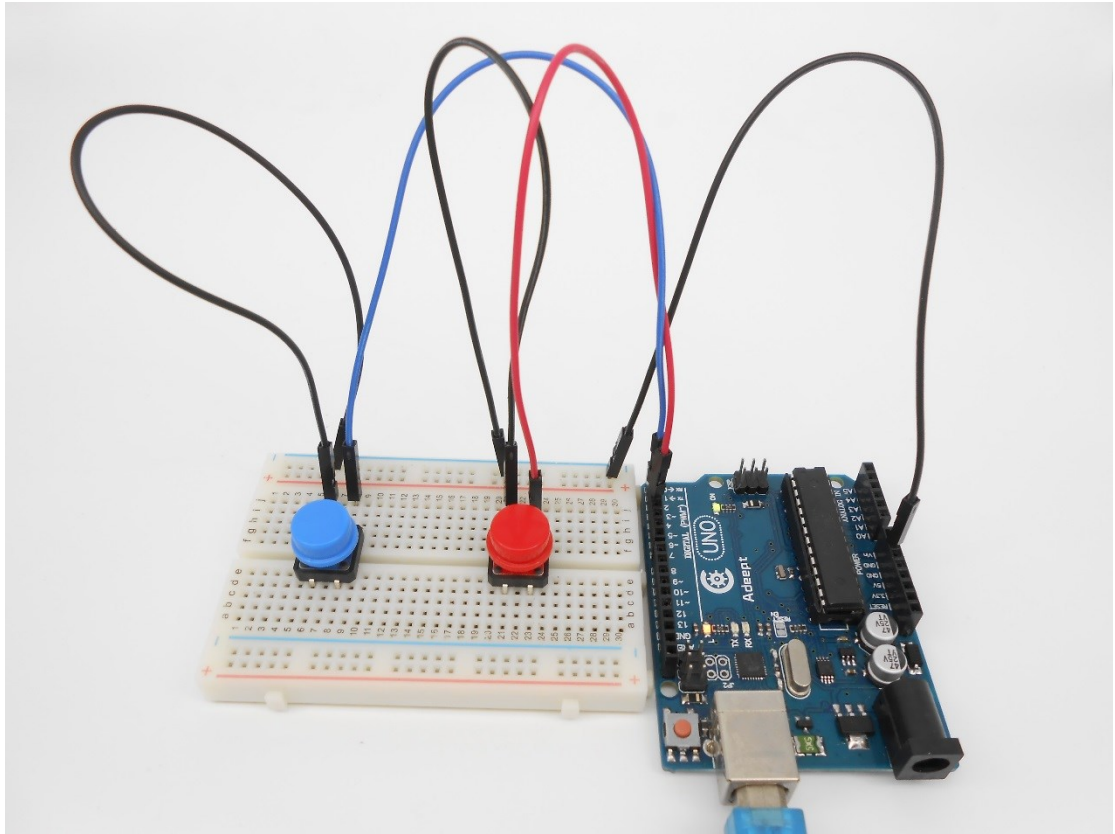
```

Snake_Game_Processing_Button | Processing 3.0a10
File Edit Sketch Debug Tools Help

Snake_Game_Processing_Button
100
101 // Use fill() to change the value or color of the text
102 fill(205);
103 text('Score: ', 8, 25);text(score, 110, 25);
104 w = width();
105 fill(229);
106 // 3 min limited
107 if(w/1000<60*2){
108   text('Time Left:      min', 8, 60);text(3-(w/60000), 110, 60);
109 }else if(w/1000<60*3){
110   text('Time Left:      sec', 8, 60);
111   fill(233, 89, 65);text(3*60-(int)(w/1000), 110, 60);
112 }
113 //else exit();
114
115 void buttonHandler()
116 {
117   if(myPort.available())>0{
118     if(myPort.readBytesUntil(10,inBuffer)>0){//Read to determine whether the wrap 10BYTE
119       String inputString = new String(inBuffer);
120       String inputStringArr[] = split(inputString,',');//Data ',' Split
121       fw[0] = int(inputStringArr[0]);//Read the X value
122       fw[1] = int(inputStringArr[1]);//Read the y value
123     }
124   }
125   if ((direction == 1) & (fw[0]!=0))
126   {
127     w0 += 2 * v * cos(theta);
128   }
129 }
130
131 Done saving.
132 100.61283
133 99.28976
134 101.04798
  
```







Adept

# Lesson 37 Star Wars

## Overview

In this lesson, we will make a star wars game based on the Processing and play the game with a PS2 joystick and a button connected to the Arduino UNO.

## Requirement

- 1\* Arduino UNO
- 1\* USB Cable
- 1\* Button
- 1\* PS2 Joystick
- 1\* Breadboard
- Several Jumper Wires

## Principle

The routine consists of two parts, the first is used to acquire data from Arduino, another is used to process the data.

Play the Star Wars Game:

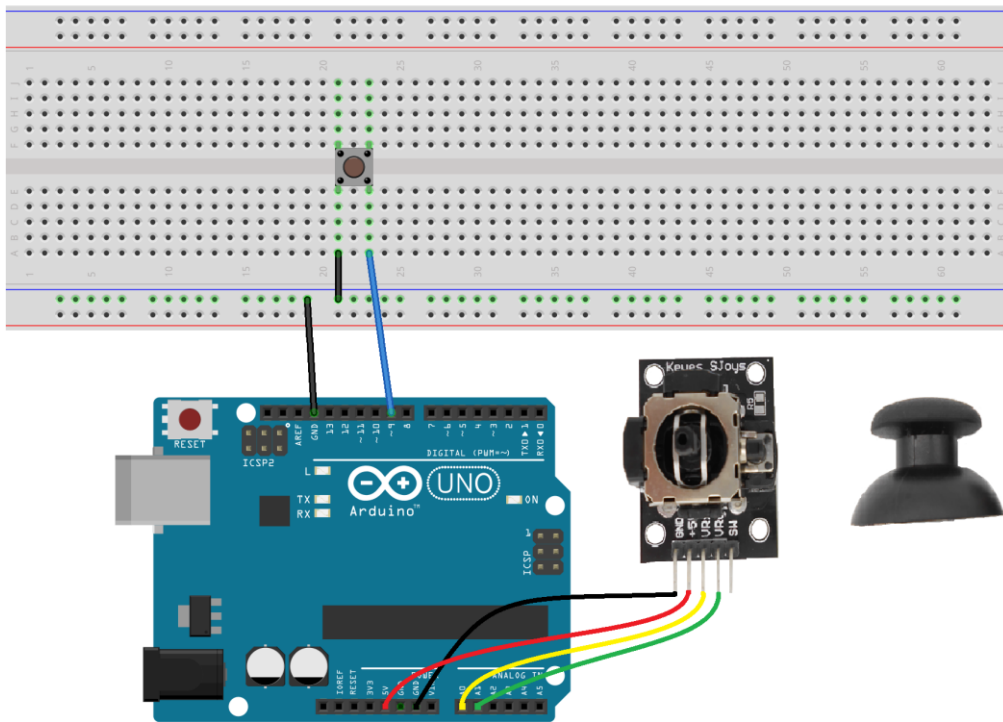
- ① When you press the button, the fighter aircraft will fire bullets
- ② When you operating the PS2 Joystick, you can change the position of movement of the fighter aircraft.

### **Note:**

1. In this experiment, my Arduino UNO board is connected to my computer COM26, please adjust according to actual situation.
2. If the Processing has not running normally, you need to install the related function libraries.

## Procedures

1. Build the circuit



fritzing

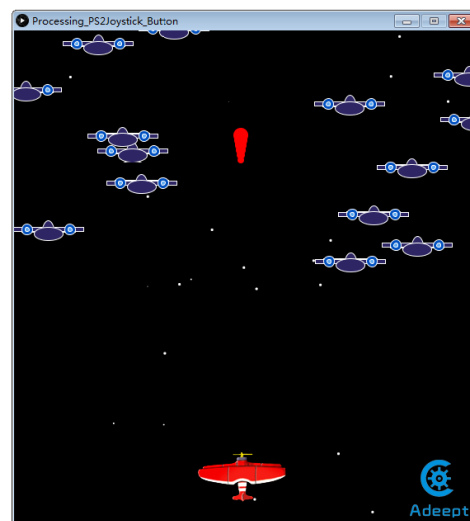
2. Program
3. Compile the program and upload to Arduino UNO board
4. Run processing software (Processing\_PS2Joystick\_Button.pde)

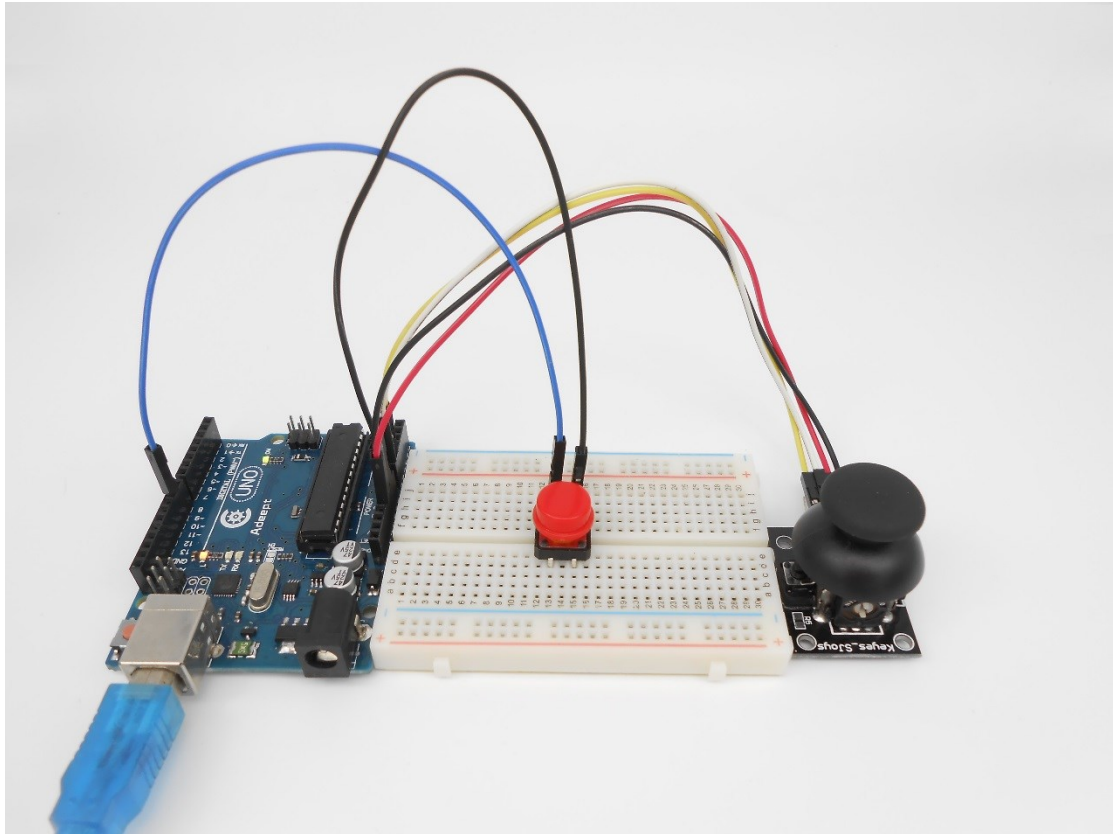
```

Processing_PS2Joystick_Button | Processing 3.0a10
File Edit Sketch Debug Tools Help

Processing_PS2Joystick_Button
254 boolean inter (Box b1) {
255     float distance=dist(x+10,y+30,b1.x+10,b1.y+6);
256     if(distance<10){
257         return true;
258     }else{
259         return false;
260     }
261 }
262
263 void readSensors() {
264     if(myPort.available()>0){
265         if(myPort.readBytesUntil(10,inBuffer)>0){ //Read to determine whether the wrap 10BY
266             String inputString = new String(inBuffer);
267             String inputStringArr[] = split(inputString, ','); //Data ', ' Split
268             Rw[0] = int(inputStringArr[0]); //Read the X value
269             Rw[1] = int(inputStringArr[1]); //Read the y value
270             Rw[2] = int(inputStringArr[2]);
271             Rw[0] = 515 - Rw[0]; //Rocker midpoint value 515 into 0
272             if (Rw[0]<0){
273                 Rw[0]=0;
274             }
275             Rw[1] = Rw[1] - 515; //Converted to negative (rocker line out)
276         }
277     }
278 }
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```





Adeci



Adeapt

**Sharing Perfects Innovation**

E-mail: [support@adeept.com](mailto:support@adeept.com)

website: [www.adeept.com](http://www.adeept.com)