

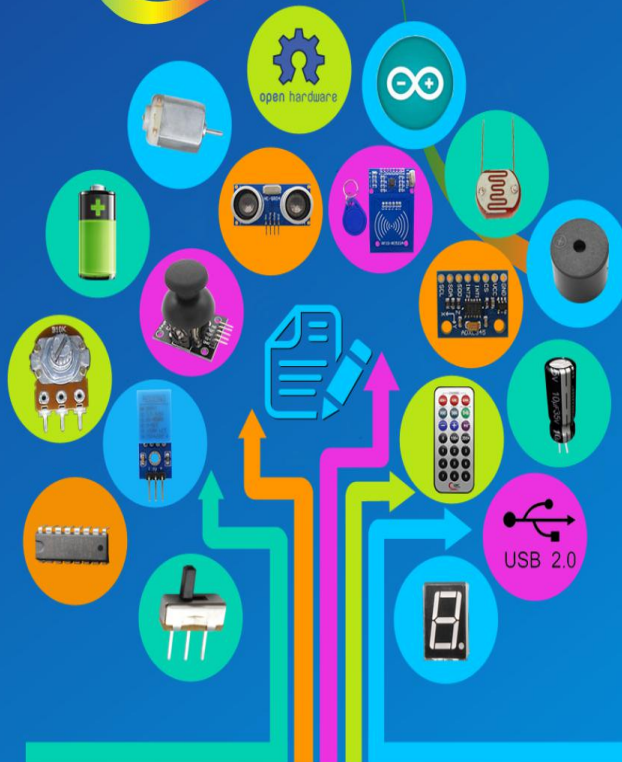


Adept

## RFID Learning kit for Arduino

Sharing Perfects Innovation

Processing




# Preface




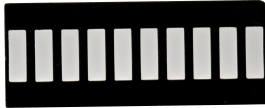




Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim 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.




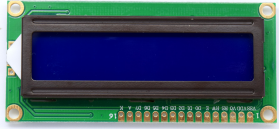
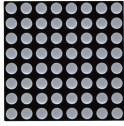




This is an entry-level learning kit for Arduino. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of Arduino, and be able to make fascinating works based on Arduino.










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

## Component List





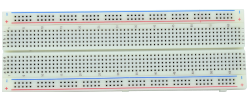

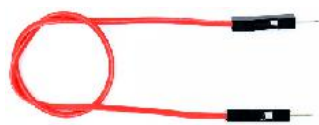
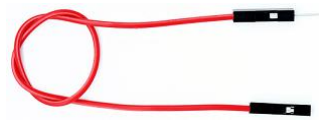
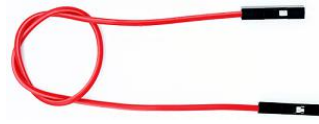
NO.	Name	Picture	Qty
1	Adept UNO R3 Board (Arduino UNO)		1
2	RC522 RFID Module		1
3	RFID ID Round Tag		1
4	RFID ID Card		1
5	ADXL345 Acceleration Sensor		1
6	Ultrasonic Distance Sensor		1
7	IR Receiver HX1838		1
8	Remote Controller		1


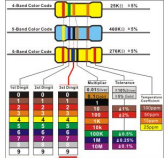
<b>9</b>	<b>PS2 Joystick Module</b>		<b>1</b>
<b>10</b>	<b>Relay</b>		<b>1</b>
<b>11</b>	<b>DHT-11 Temperature &amp; Humidity Sensor</b>		<b>1</b>
<b>12</b>	<b>LED Bar Graph</b>		<b>1</b>
<b>13</b>	<b>Active Buzzer</b>		<b>1</b>
<b>14</b>	<b>Passive Buzzer</b>		<b>1</b>
<b>15</b>	<b>Servo</b>		<b>1</b>
<b>16</b>	<b>Analog Temperature Sensor(Thermistor)</b>		<b>2</b>

17	4*4 Matrix Keyboard		1
18	DC Motor		1
19	L9110 Motor Driver		1
20	LCD1602		1
21	Dot-matrix Display		1
22	7-segment Display		1
23	4-digit 7-segment Display		1
24	74HC595		2
25	Light Sensor(Photoresistor)		2

<b>26</b>	<b>Switch</b>		<b>2</b>
<b>27</b>	<b>RGB LED</b>		<b>1</b>
<b>28</b>	<b>Red LED</b>		<b>8</b>
<b>29</b>	<b>Green LED</b>		<b>2</b>
<b>30</b>	<b>Yellow LED</b>		<b>2</b>
<b>31</b>	<b>Blue LED</b>		<b>2</b>
<b>32</b>	<b>Resistor(220Ω)</b>		<b>16</b>
<b>33</b>	<b>Resistor(1KΩ)</b>		<b>10</b>
<b>34</b>	<b>Resistor(10KΩ)</b>		<b>5</b>

35	Potentiometer(10KΩ)		2
36	Capacitor(104)		5
37	Capacitor(10uF)		2
38	Button(Large)		4
39	Button(Small)		4
40	Button Cap(Red)		1
41	Button Cap(White)		1
42	Button Cap(Blue)		2
43	NPN Transistor(8050)		2

<b>44</b>	<b>PNP Transistor(8550)</b>		<b>2</b>
<b>45</b>	<b>1N4148 Diode</b>		<b>2</b>
<b>46</b>	<b>1N4001 Diode</b>		<b>2</b>
<b>47</b>	<b>Battery Holder</b>		<b>1</b>
<b>48</b>	<b>Breadboard</b>		<b>1</b>
<b>49</b>	<b>USB Cable</b>		<b>1</b>
<b>50</b>	<b>Male to Male Jumper Wires</b>		<b>40</b>
<b>51</b>	<b>Male to Female Jumper Wires</b>		<b>20</b>
<b>52</b>	<b>Female to Female Jumper Wires</b>		<b>3</b>

53	Header(40 pin)		1
54	Band Resistor Card		1

# Contents

About Arduino.....	- 1 -
About Processing.....	- 2 -
Lesson 1 Blinking LED.....	- 3 -
Lesson 2 Active Buzzer.....	- 8 -
Lesson 3 Controlling an LED by 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.....	- 31 -
Lesson 9 Controlling an RGB LED by PWM.....	- 34 -
Lesson 10 Playing Music.....	- 37 -
Lesson 11 LCD1602 Display.....	- 40 -
Lesson 12 7-segment Display.....	- 44 -
Lesson 13 A Simple Counter.....	- 47 -
Lesson 14 Dot-matrix Display.....	- 50 -
Lesson 15 Controlling a Servo.....	- 54 -
Lesson 16 Photoresistor.....	- 57 -
Lesson 17 Measuring Temperature by a Thermistor.....	- 60 -
Lesson 18 IR Remote Controller.....	- 63 -

<b>Lesson 19 Temperature &amp; Humidity Sensor DHT-11.....</b>	<b>- 67 -</b>
<b>Lesson 20 Ultrasonic Distance Sensor.....</b>	<b>- 71 -</b>
<b>Lesson 21 3-axis Accelerometer—ADXL345.....</b>	<b>- 73 -</b>
<b>Lesson 22 4x4 Matrix Keypad.....</b>	<b>- 79 -</b>
<b>Lesson 23 Controlling DC motor.....</b>	<b>- 83 -</b>
<b>Lesson 24 PS2 Joystick.....</b>	<b>- 88 -</b>
<b>Lesson 25 A Simple Voltmeter.....</b>	<b>- 90 -</b>
<b>Lesson 26 RFID module.....</b>	<b>- 94 -</b>
<b>Lesson 27 Move a cat.....</b>	<b>- 101 -</b>
<b>Lesson 28 Control the brightness of a photo with a photoresistor.....</b>	<b>- 111 -</b>
<b>Lesson 29 Controlling the 3D Model by PS2 Joystick.....</b>	<b>- 117 -</b>
<b>Lesson 30 The Brick Games.....</b>	<b>- 120 -</b>

# 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. To begin, let's learn how to make an LED blink.

## Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 220 $\Omega$  Resistor
- 1 \* LED
- 1 \* Breadboard
- 2 \* Jumper Wires

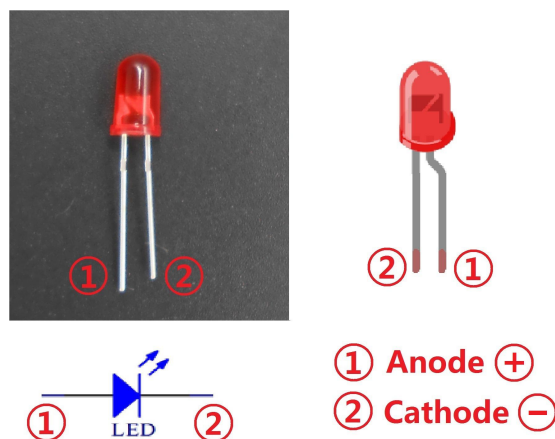
## Principle

In this lesson, we will program the Arduino's GPIO output high level (+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 one. It lights up only when a forward current passes, and it can flash red, blue, green, yellow, etc. The color of the light depends on the material it is 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 resistor?

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

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

There are two methods for connecting LED to pins of an Arduino board:

①



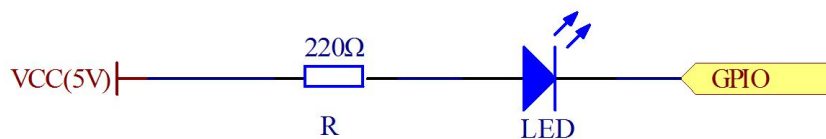
As shown in the schematic diagram, the anode of the LED is connected to Arduino's GPIO via a resistor, and the cathode to the ground (GND). When the GPIO outputs high level, the LED is on; when it outputs low, the LED is off.

The resistance of a 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 an LED is a resistor itself, here we use 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 made based on method ① – use pin D8 of the Arduino board to control an LED. When D8 is programmed to output high level, the LED will be turned on. Next, delay for some time. Then D8 is programmed to output low level to turn the LED off. Repeat the above process and you can get a blinking LED then.

## 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.

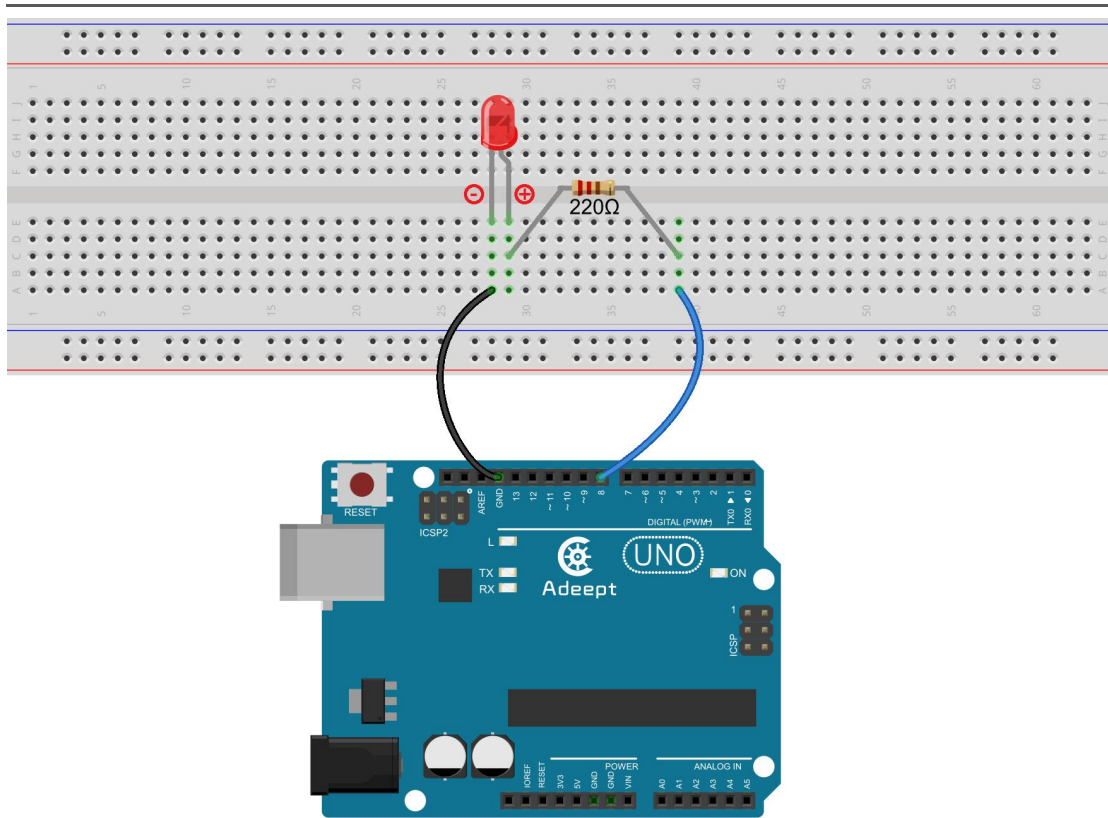
If the pin is configured as an INPUT, `digitalWrite()` will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the `pinMode()` to `INPUT_PULLUP` to enable the internal pull-up resistor.

### ● `delay()`

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

## **Procedures**

Step 1: Build the circuit



## Step 2: Program

```

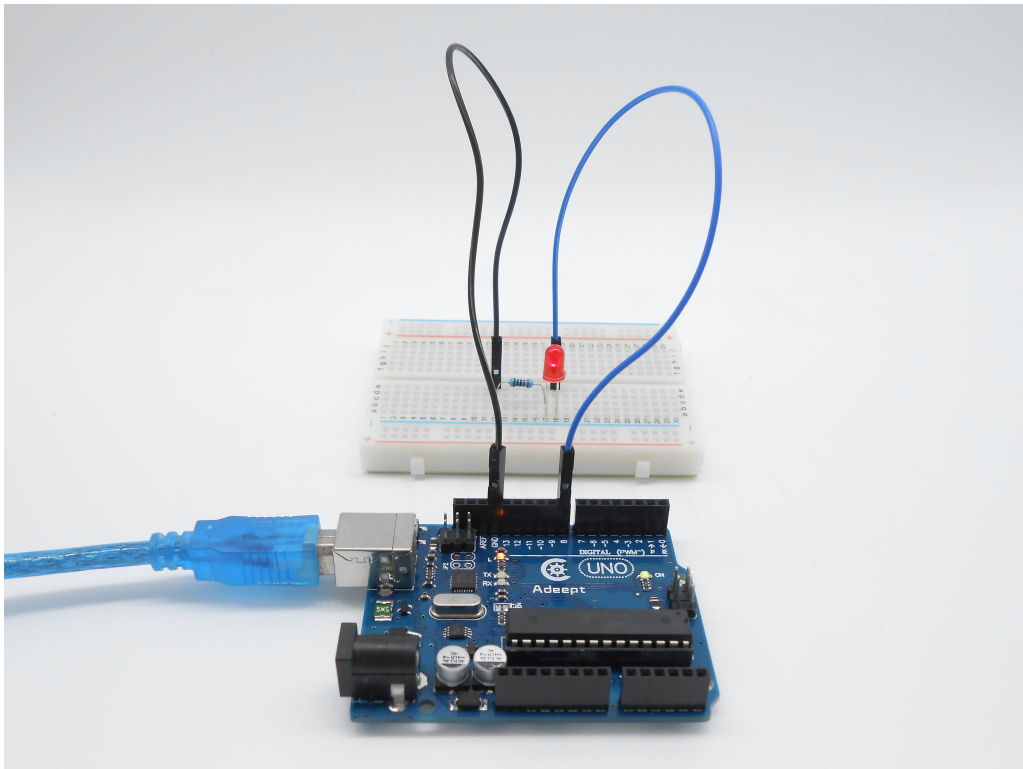
/*****
File name: 01_blinkingLed.ino
Description: Lit LED, let LED blinks.
Website: www.adeept.com
E-mail: support@adeept.com
Author: Tom
Date: 2015/05/02
*****/

int ledPin=8; //definition digital 8 pin 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
}

```

---

Step 3: Compile the program and upload to Adept UNO board  
Now you can see the LED blinking.



## Lesson 2 Active Buzzer

### Overview

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

### Components

- 1 \* Adept UNO R3 Board
- 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 an integrated structure, which uses DC power supply, buzzers 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).

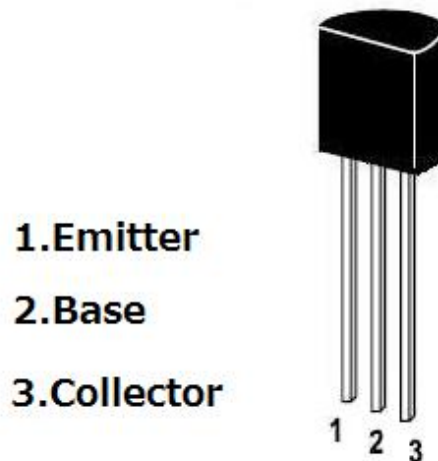


Place the pins of the buzzer face up, and then you can see the two types of buzzer are different - the buzzer with a green circuit board inside is a passive one.

In this lesson, the buzzer we used is active buzzer. Active buzzers will sound as long as they are powered. We can program to make the Arduino output alternating high and low levels to make the buzzer beep.

A slightly larger current is needed to make a buzzer beep. However, the output current of Arduino GPIO is too low, so we need a transistor to help.

The main function of a transistor is to enlarge the voltage or current. It can also be used to control the circuit conduction or deadline. Transistors can be divided into two kinds: NPN, like the S8050 we provided; PNP, like the S8550 provided. The transistor we use is as shown below:



There are two kinds of driving circuit for buzzer:

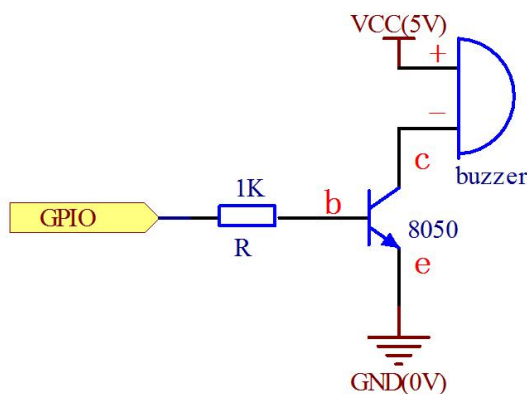


Figure 1

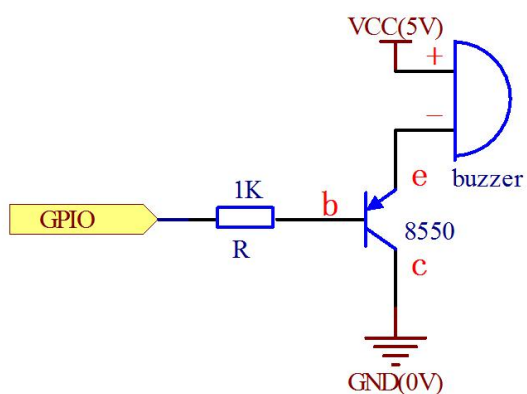


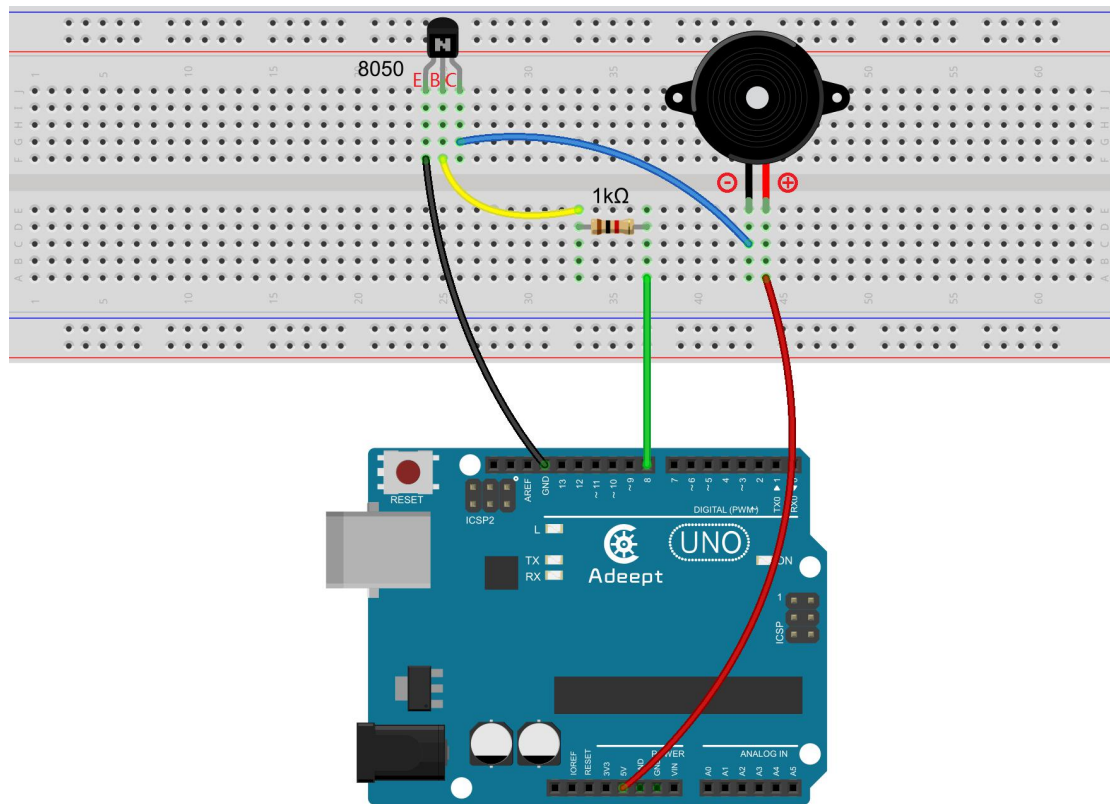
Figure 2

Figure 1: Set the Arduino GPIO as a high level. Then the transistor S8050 will conduct, and the buzzer will make sounds. Set the GPIO as low, the transistor S8050 will be de-energized, and the buzzer will stop beeping.

Figure 2: Set the Arduino GPIO as low level. The transistor S8550 will be energized and the buzzer will beep. Set the GPIO as a high, and the transistor S8550 will be de-energized, and the buzzer beeping will stop.

## Procedures

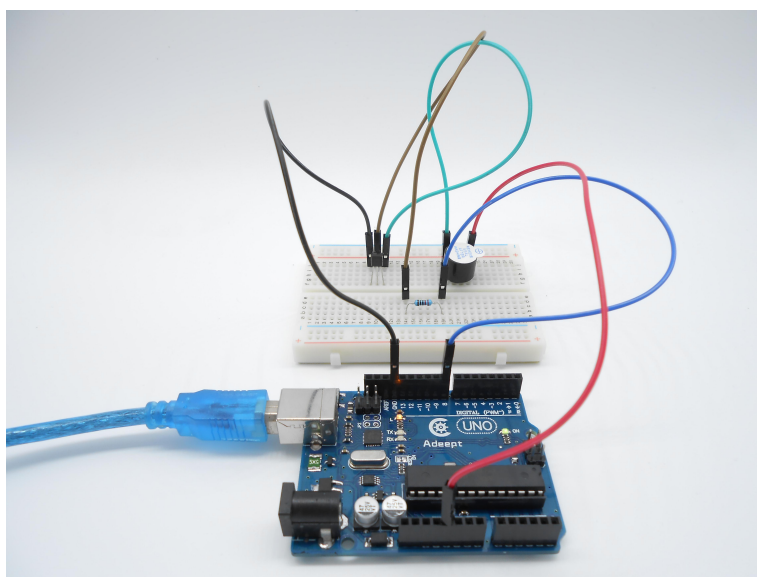
Step 1. Build the circuit



2. Program

3. Compile the program and upload to Adept UNO board

Now, you can hear the buzzer beeping.



---

## Summary

After learning this lesson, you can master the basic principle of the buzzer and transistor. Also you've learned how to program the Arduino and then control the buzzer. Now you can use what you've learned in this lesson to make some interesting things!

## Lesson 3 Controlling an LED by a Button

### Overview

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

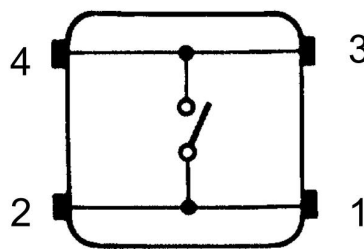
### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* Button
- 1 \* LED
- 1 \* 10k $\Omega$  Resistor
- 1 \* 220 $\Omega$  Resistor
- 1 \* Breadboard
- Several jumper wires

### Principle

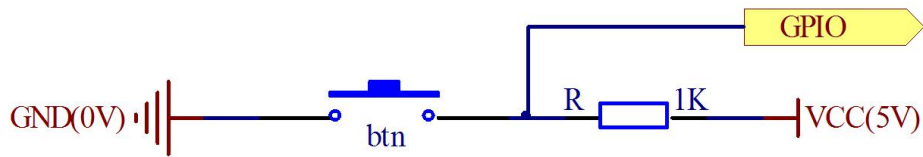
#### *1. Button*

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 below.

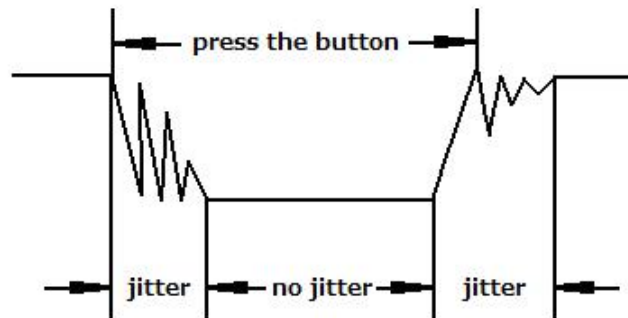


The button we used is a normally open type one. The two contacts of a button are in the off state under the normal conditions; only when the button is pressed they are closed.

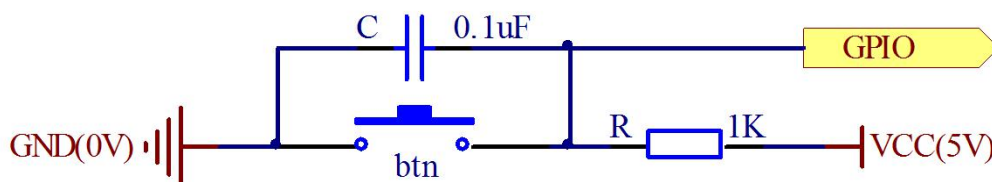
The schematic diagram is as follows:



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



Each time you press the button, the Arduino will regard you have pressed the button many times due to the jitter of the button. You should deal with the jitter of buttons before using. You can eliminate the jitter through software programming. Besides, you can use a capacitor to solve the issue. Take the software method for example. First, detect whether the level of button interface is low level or high level. If it is low level, 5~10ms delay is needed. Then detect whether the level of button interface is low or high. If the signal is low, you can infer that the button is pressed once. You can also use a 0.1uF capacitor to avoid the jitter of buttons. The schematic diagram is as shown 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:*

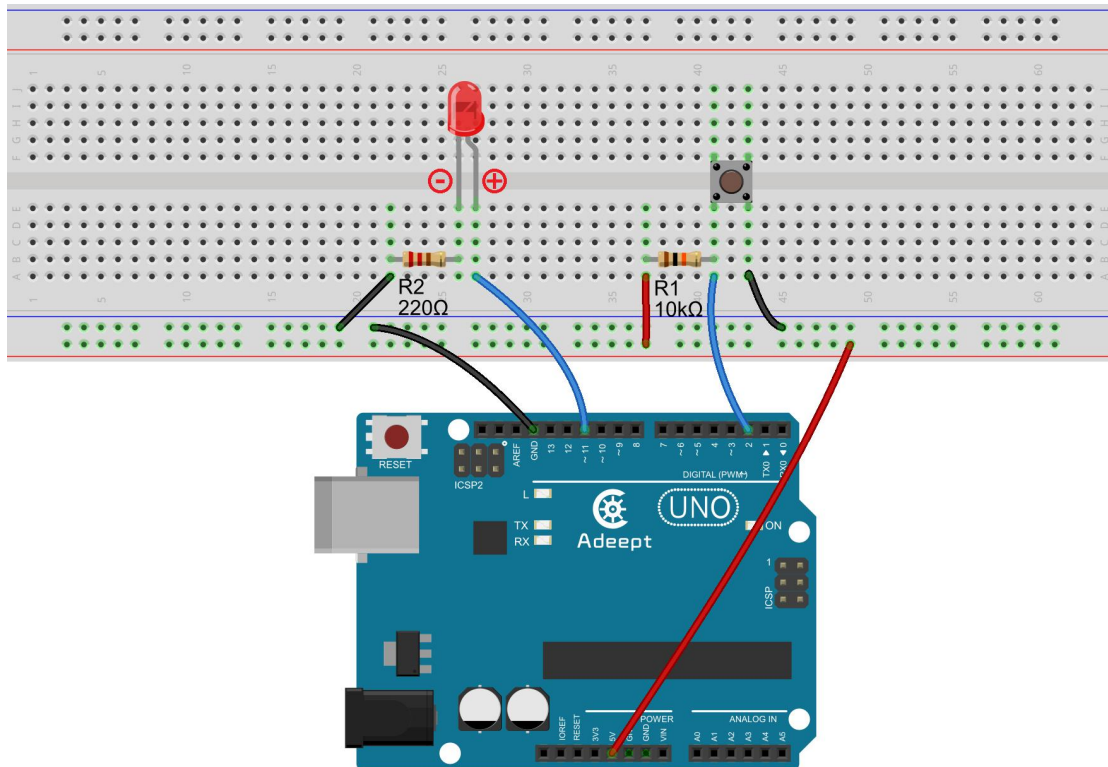
us: the number of microseconds to pause (unsigned int)

*Returns:*

None

## Procedures

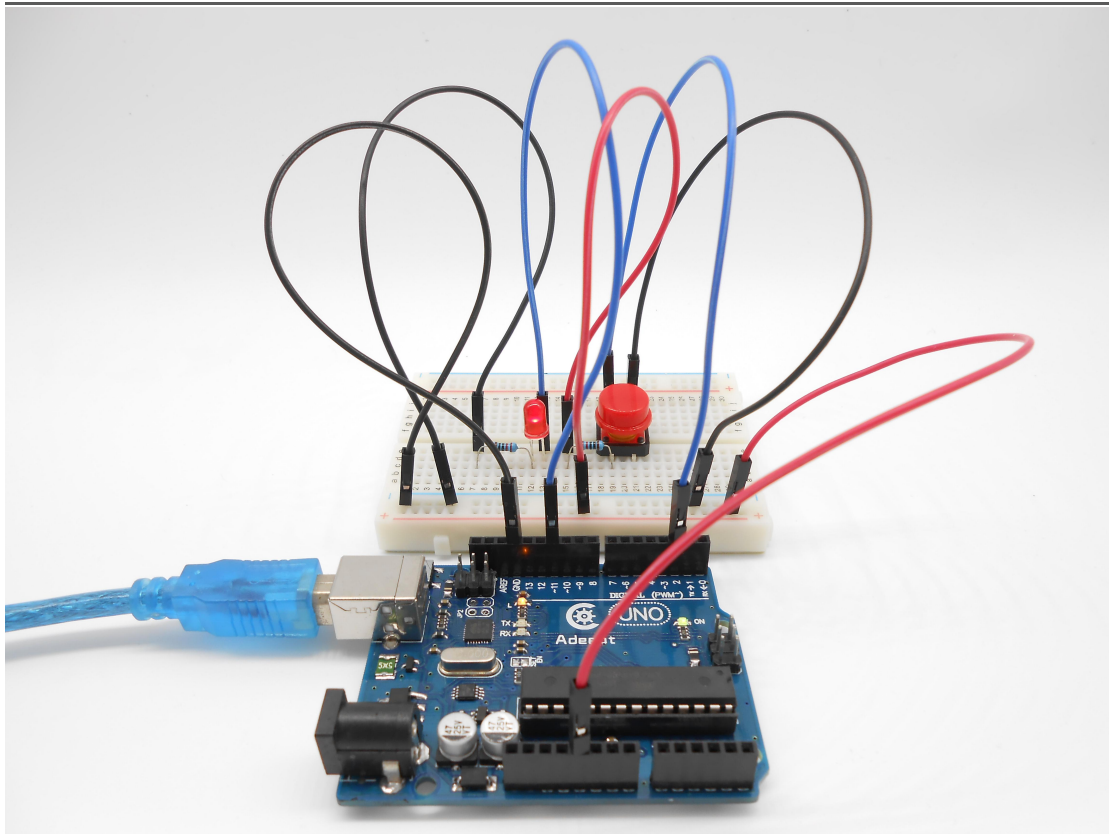
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now press the button, and you can see the state of the LED will be toggled between ON and OFF.



## Summary

Through this lesson, you should have learned how to use the Arduino UNO to detect the status of an external button, and then toggle the state of LED on/off 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 break or connect a circuit.

### Components

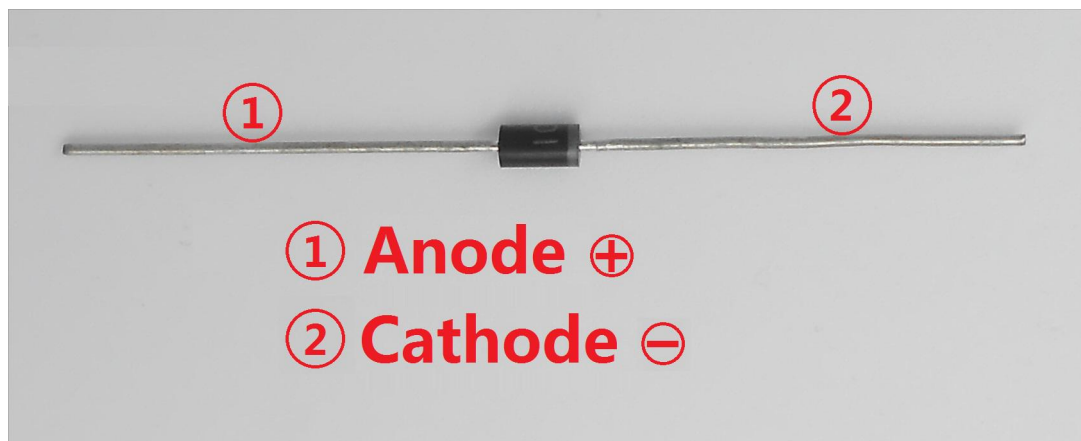
- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* NPN Transistor (S8050)
- 1 \* 1K Resistor
- 1 \* 1N4001 Diode
- 1 \* 220 $\Omega$  Resistor
- 1 \* Relay
- 1 \* LED
- 1 \* Breadboard
- Several jumper wires

### 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.

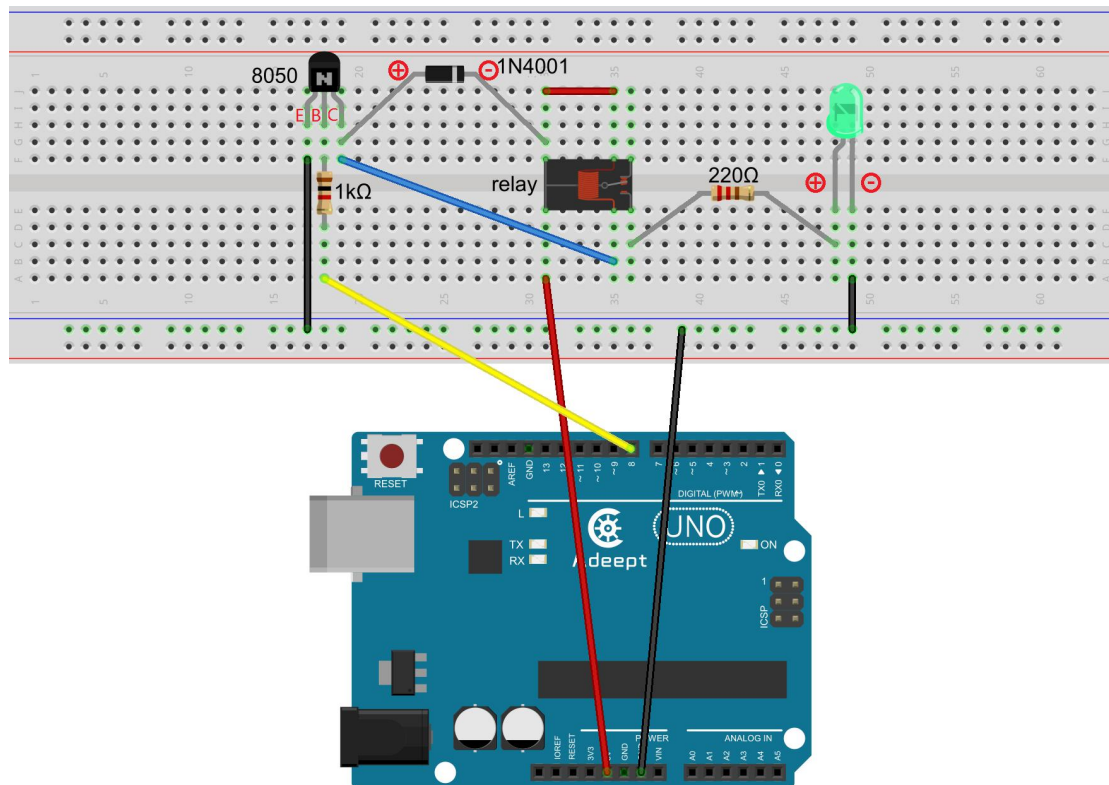
When the coil is energized with direct current, a diode is often placed across the coil to dissipate the energy from the collapsing magnetic field at deactivation, which would otherwise generate a voltage spike dangerous to

semiconductor circuit components.



## Procedures

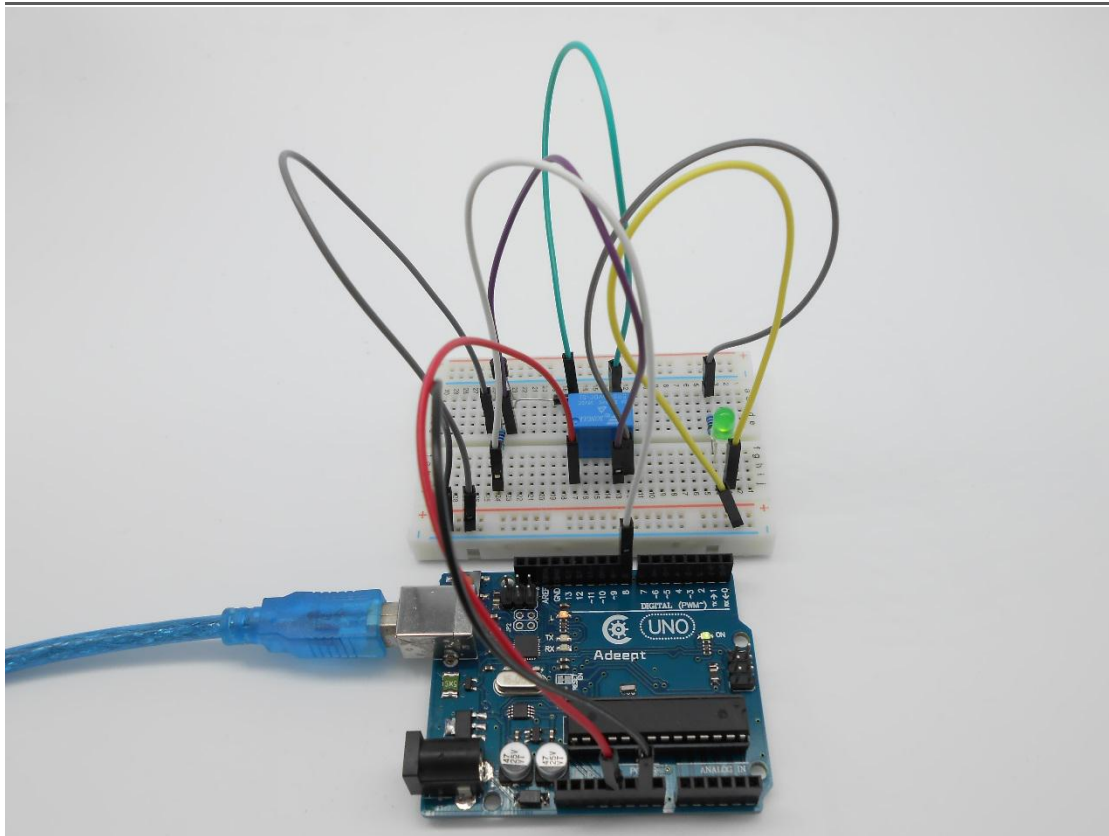
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now let's see what will happen: When the set of contacts are closed, the LED lights up; when they are open, the LED goes out.



## Summary

By this lesson, you should have learned the basic principle of relay. You can also use the relay to do some creative applications. Just give it a try!

## Lesson 5 Serial Port

### Overview

In this lesson, we will program the Arduino UNO to achieve sending and receiving data through the serial port. The Uno board receive data sent from a PC, then controls an LED according to the received data, and at last returns the state of the LED to Serial Monitor in Arduino IDE.

### Components

- 1 \* Adept UNO R3 Board
- 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:*

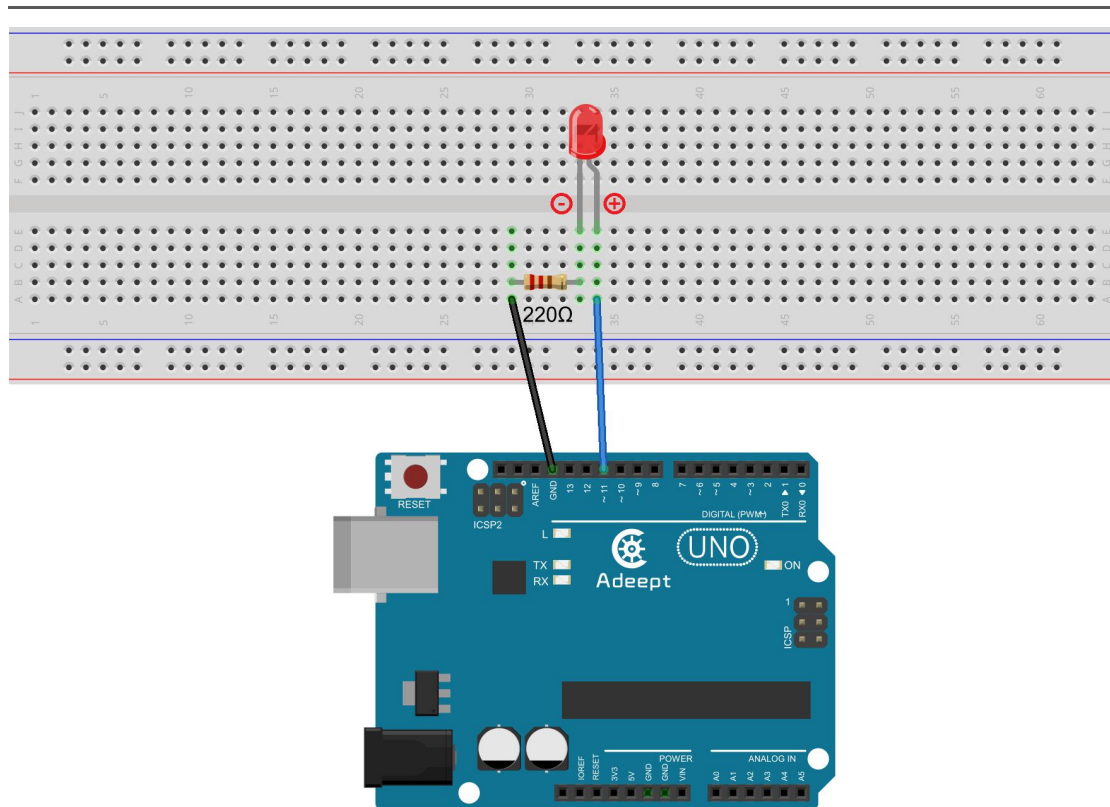
None

#### *Returns:*

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

## Procedures

Step 1: Build the circuit

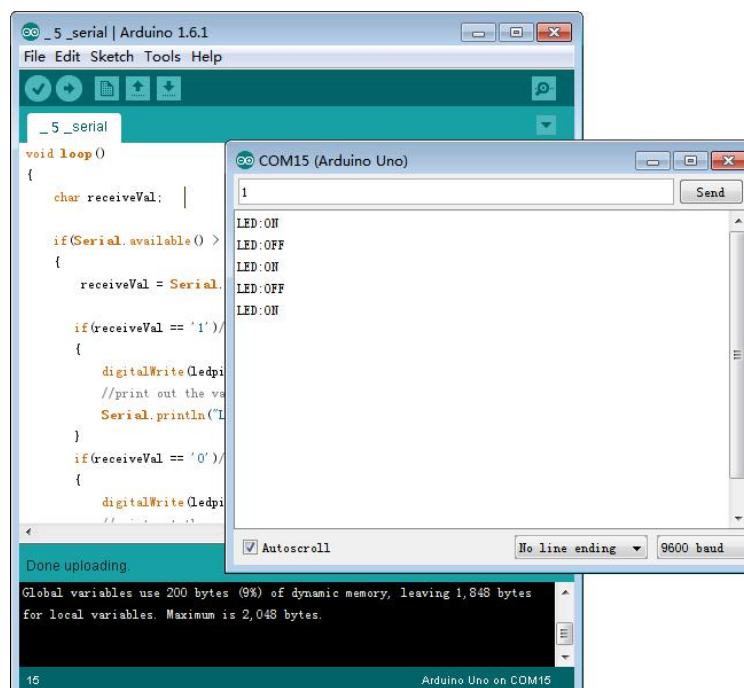


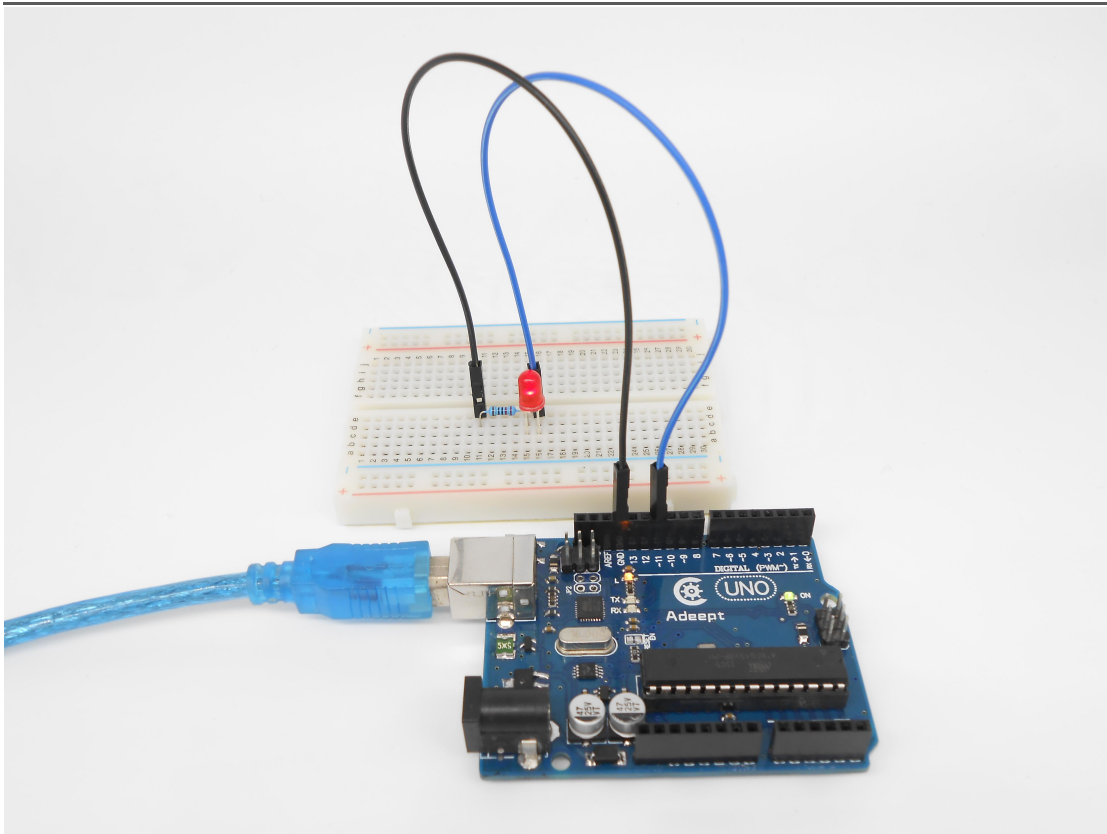
Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Open IDE, click to open Serial Monitor, and then select the appropriate baud rate according to the program.

Now, enter '1' or '0' in the textbox on the monitor, and the LED will be turned on/off.





## Summary

Through this lesson, we know that the computer can send data to Arduino UNO via the serial port, and then control the state of an LED. Now try to make more interesting things based on what you've got.

## Lesson 6 LED Flowing Lights

### Overview

In the first lesson, we have learned how to make an LED blink by programming the Arduino. Today, we will use the Arduino to control 8 LEDs to make the LEDs show the effect of flowing.

### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 8 \* LED
- 8 \* 220Ω Resistor
- 1 \* Breadboard
- Several jumper wires

### Principle

The principle of this experiment is very simple and is quite similar with that in the first lesson.

#### *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);  
}
```

parenthesis

declare variable (optional)

initialize

test

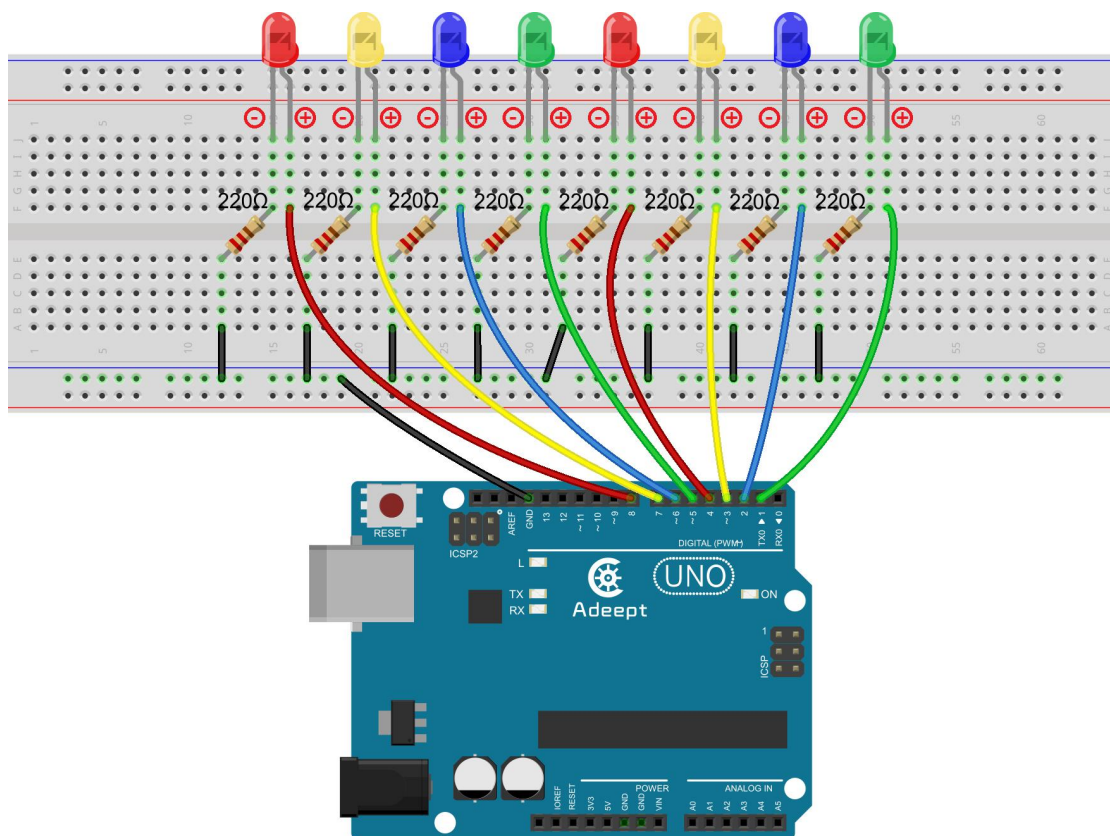
increment or decrement

```
for(int x = 0; x < 100; x++){
    println(x); // prints 0 to 99
}
```

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

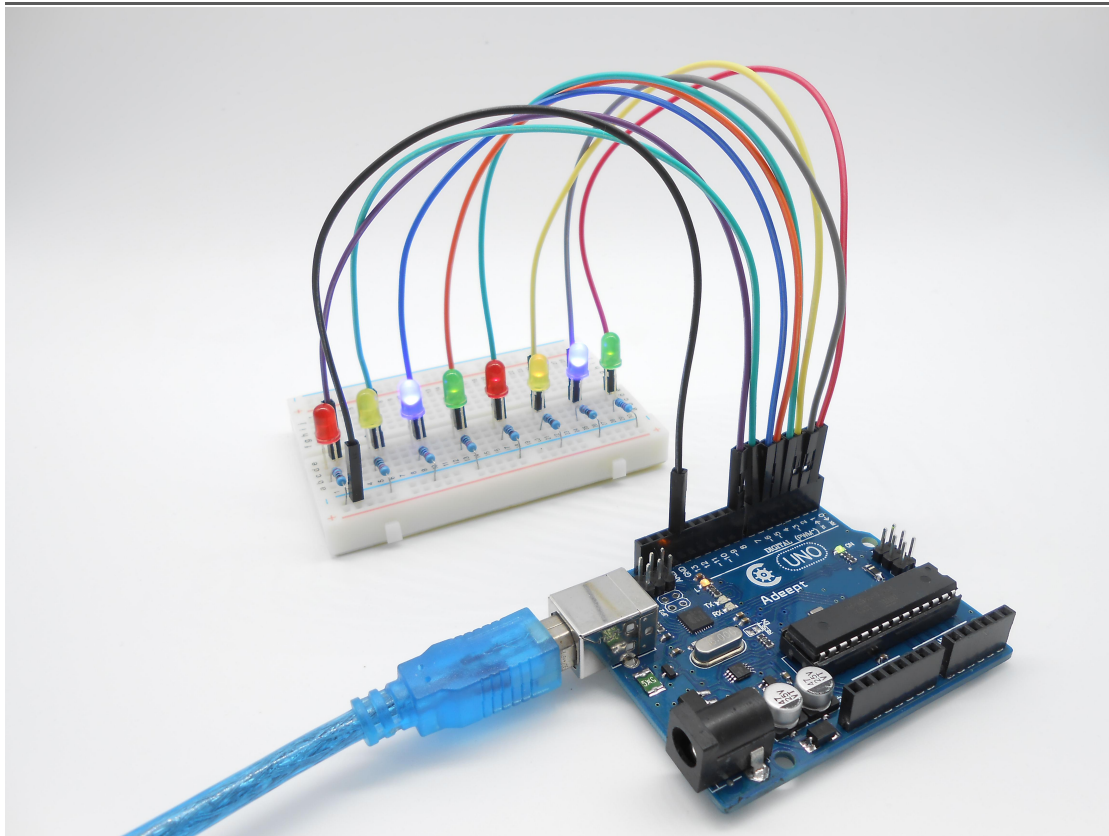
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, you can see 8 LEDs light up in sequence from the green one on the right side to others on the left, and next from the left to the right. The LEDs flash like flowing water repeatedly in a circular way.



## Summary

Through this simple but fun experiment, you should have learned more skills in programming on Arduino. In addition, you can also modify the circuit and code provided to achieve even more dazzling effects.

## Lesson 7 LED Bar Graph Display

### Overview

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

### Components

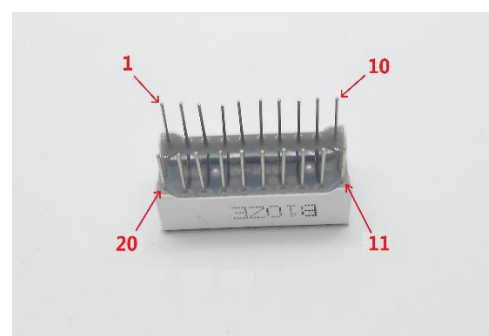
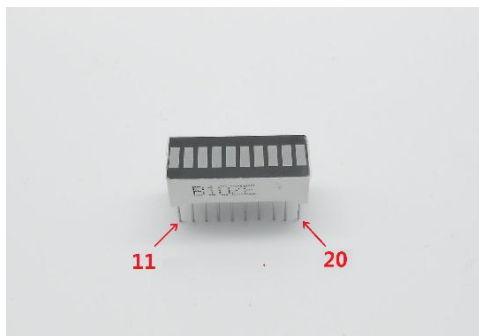
- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 10k $\Omega$  Potentiometer
- 10 \* 220 $\Omega$  Resistor
- 1 \* LED Bar Graph
- 1 \* Breadboard
- Several jumper wires

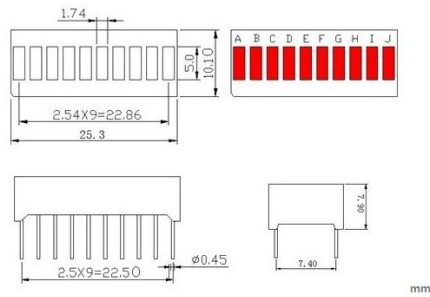
### Principle

The bar graph - a series of LEDs in a line, as you can 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.

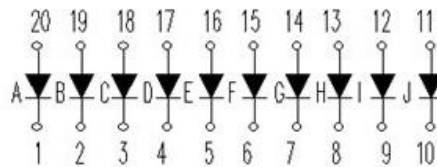
This tutorial borrows from the [For Loop and Arrays](#) tutorial as well as the [Analog Input](#) tutorial.

The sketch works like this: first read the input. Map the input to the output range which is 0-10 in this case since ten LEDs are used. Then you set up a *for* loop to iterate over the outputs. If the number in the array of the output is lower than the mapped input minimum, it is turned on. If not, it's off.





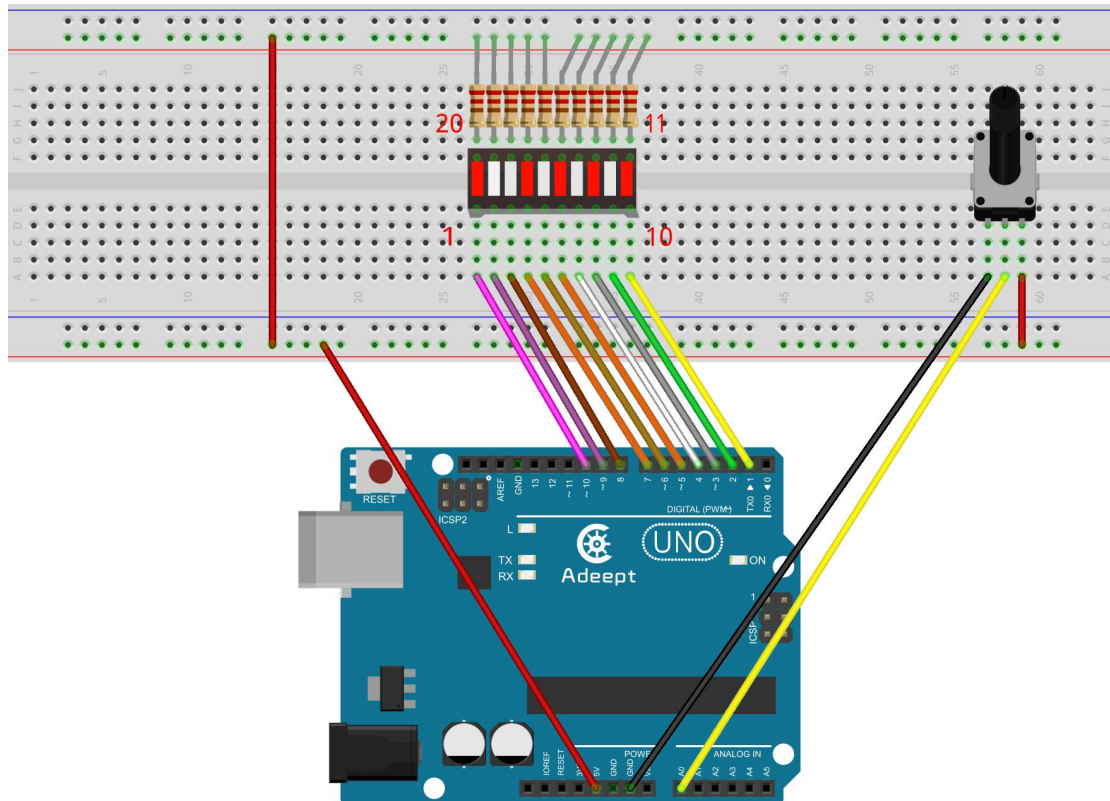
The internal schematic diagram for the LED bar graph is as shown below:



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

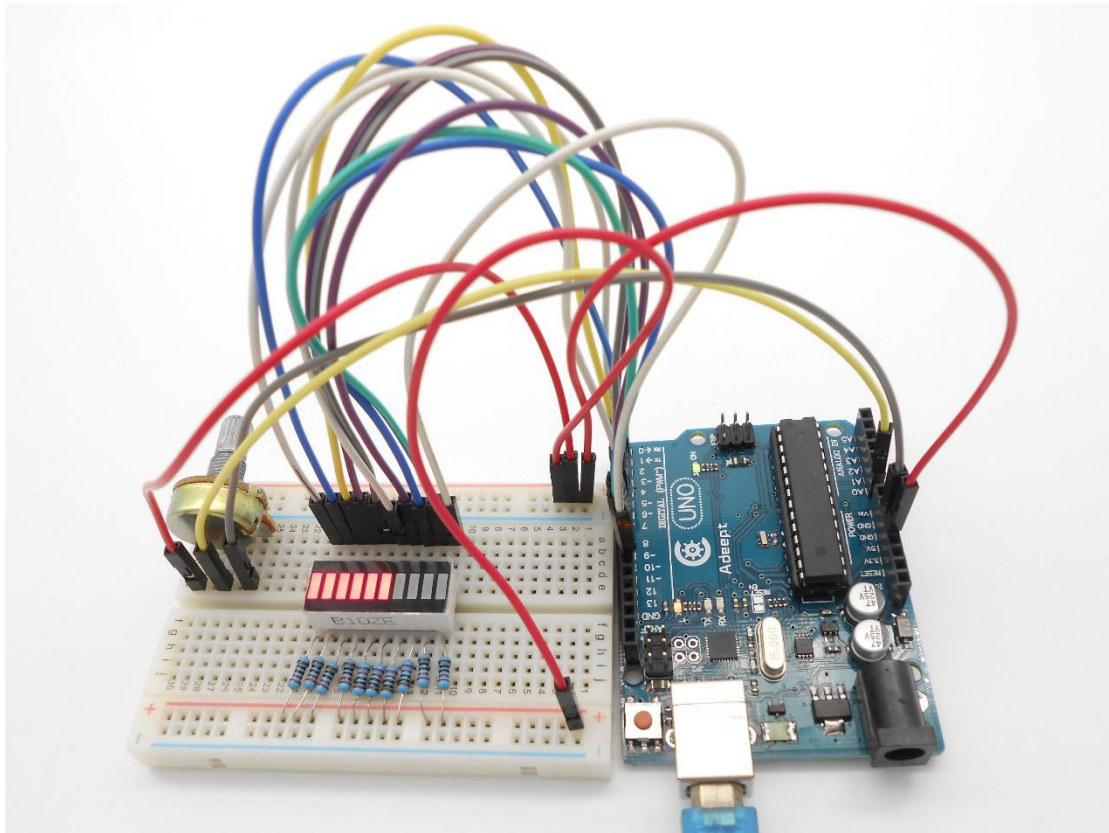
## Step 1: Build the circuit



## Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, turn the knob of the potentiometer, and you will see the number of LEDs in the LED bar graph changed.



## Lesson 8 Breathing LED

### Overview

In this lesson, we will learn how to program the Arduino to generate PWM signals. And then we use the PWM square-wave signals to control an LED gradually getting brighter and then slowly dimmer, much like human breath.

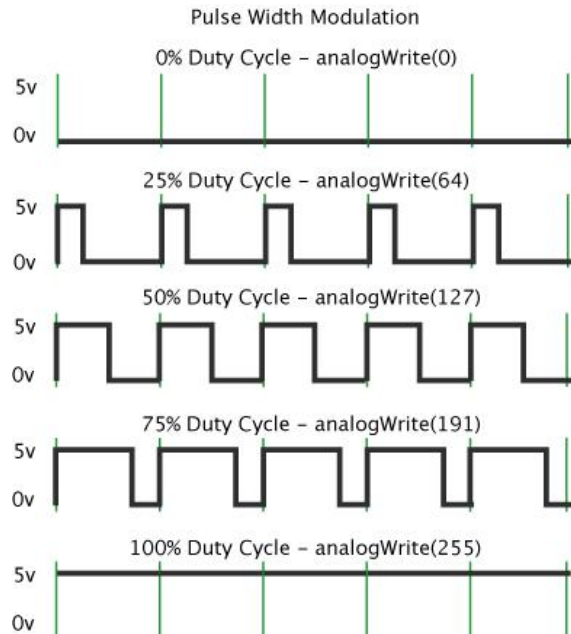
### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* LED
- 1 \* 220 $\Omega$  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 following figure, 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.

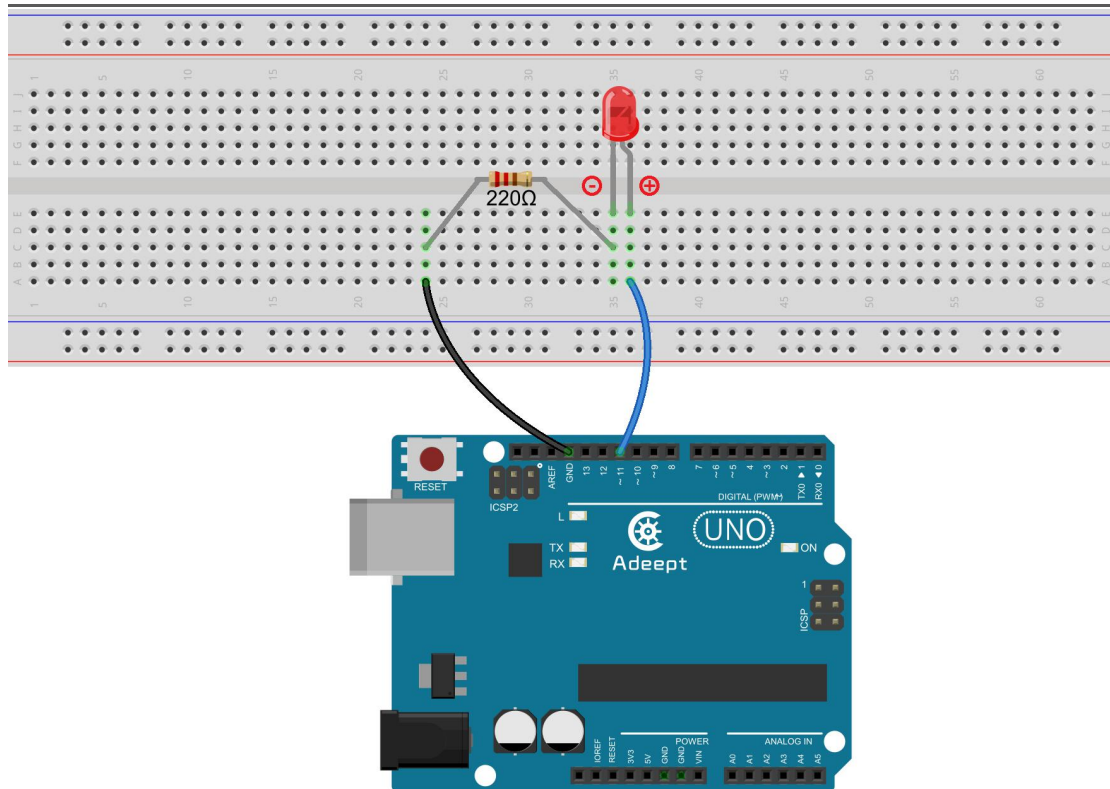
value: the duty cycle: between 0 (always off) and 255 (always on).

### *Returns:*

nothing

## **Procedures**

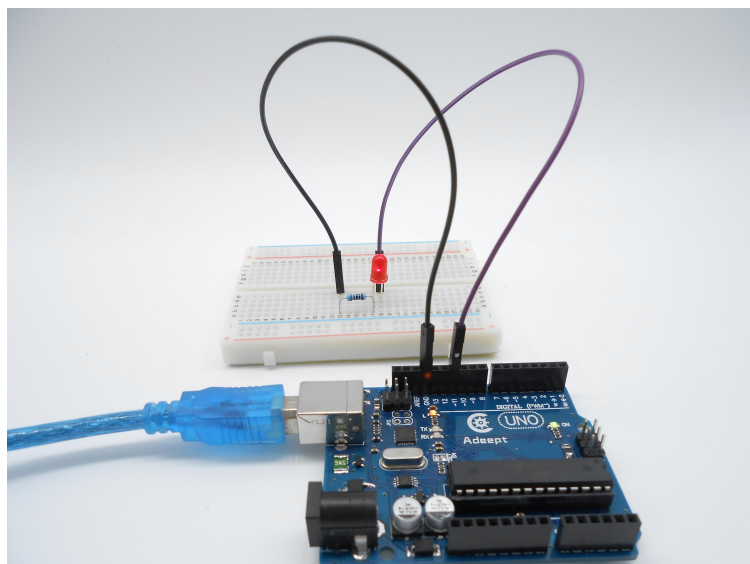
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board.

Now, you should see the LED lights up and gets gradually brighter, and then slowly turns dimmer. The process repeats circularly, and with the particular rhythm it looks like animals' breath.



## Summary

By learning this lesson, you should have mastered the basic principles of the PWM, and get skilled at the PWM programming on the Arduino platform.

## Lesson 9 Controlling an RGB LED by PWM

### Overview

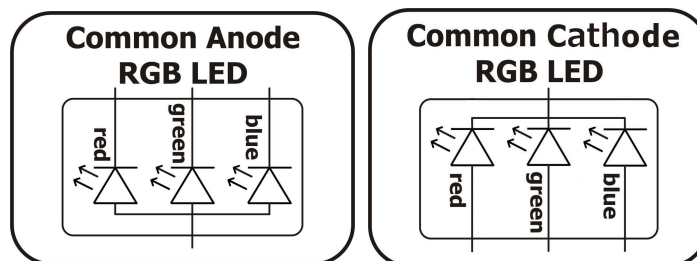
In this lesson, we will program the Arduino for RGB LED control, and make RGB LED emits a various of colors of light.

### Components

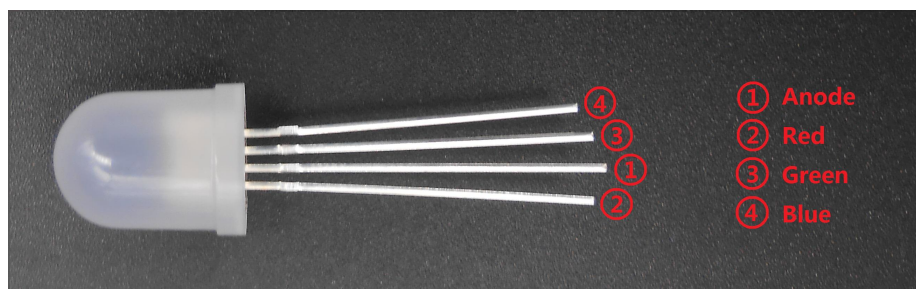
- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* RGB LED
- 3\* 220Ω Resistor
- 1 \* Breadboard
- Several jumper wires

### Principle

RGB LEDs consist of three LEDs: red, green and blue. 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).



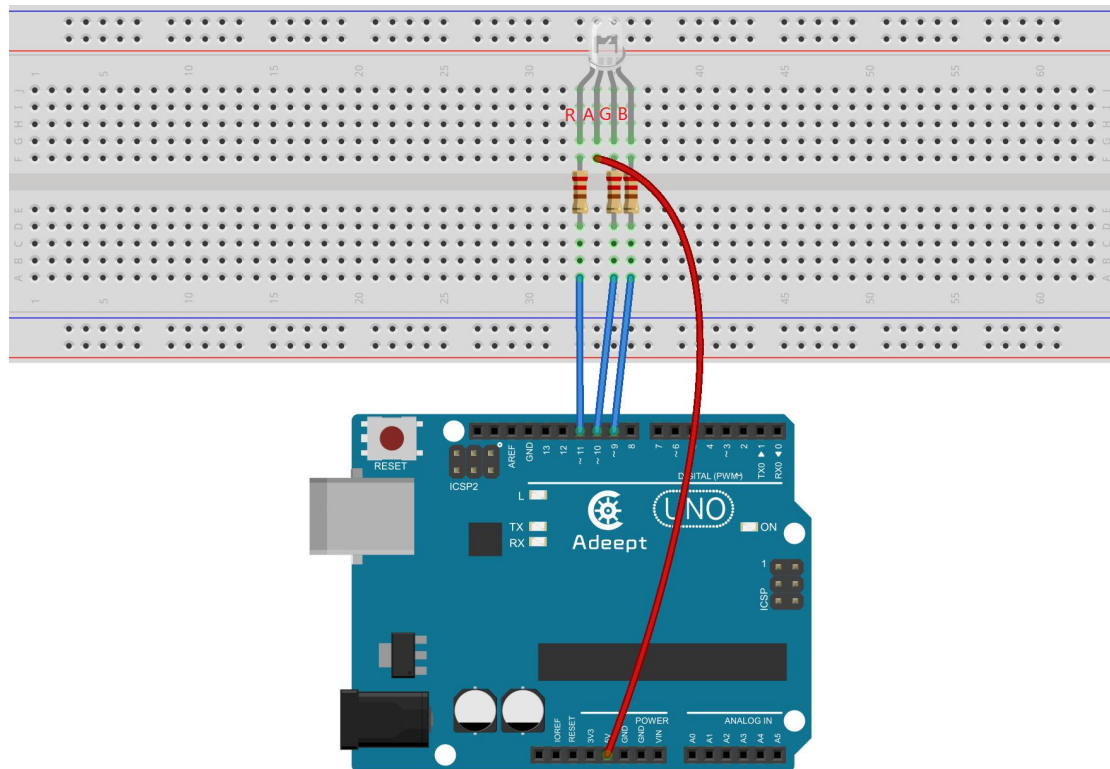
What we use in this experiment is a common anode RGB LED. The longest pin is the common anode of the three LEDs. The pin is connected to the +5V pin of the Arduino, and the rest pins are connected to pin D9, D10, and D11 of the Arduino with a current limiting resistor between.



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

## Procedures

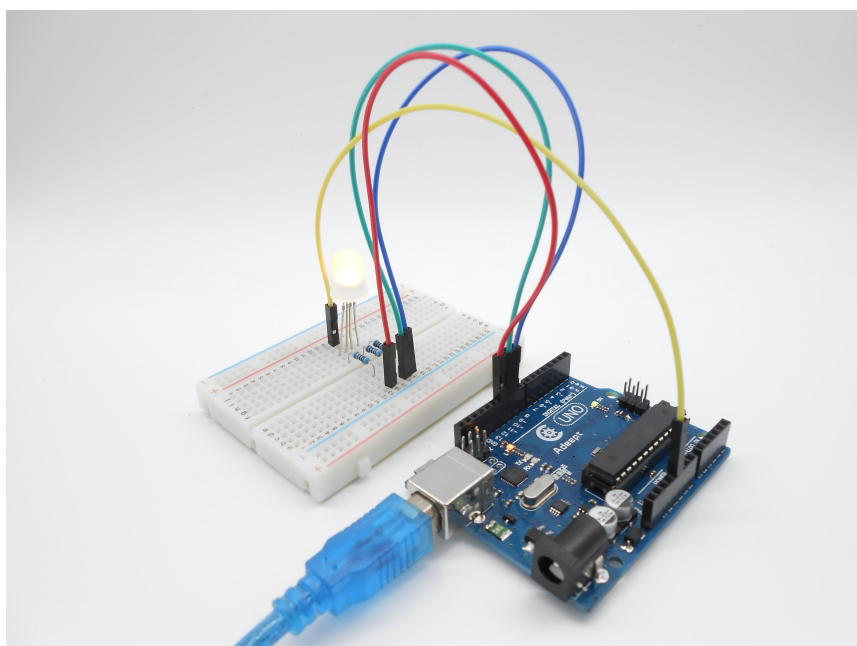
### Step 1: Build the circuit



## Step 2: Program

### Step 3: Compile the program and upload to Adept UNO board

Now, you can see the RGB LED flash red, green, blue, yellow, white and purple light, and then go out. Each state lasts for 1s each time, and the LED flashes colors repeatedly in such sequence.



## Summary

By learning this lesson, you should have already grasped the principle and the programming of RGB LED. Now you can use your imagination to achieve even more cool ideas based on what you learned in this lesson.

## Lesson 10 Playing Music

### Overview

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

### Components

- 1 \* Adept UNO R3 Board
- 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 some square wave signals to a passive buzzer with different frequencies, the buzzer will make different sounds accordingly.



### *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:

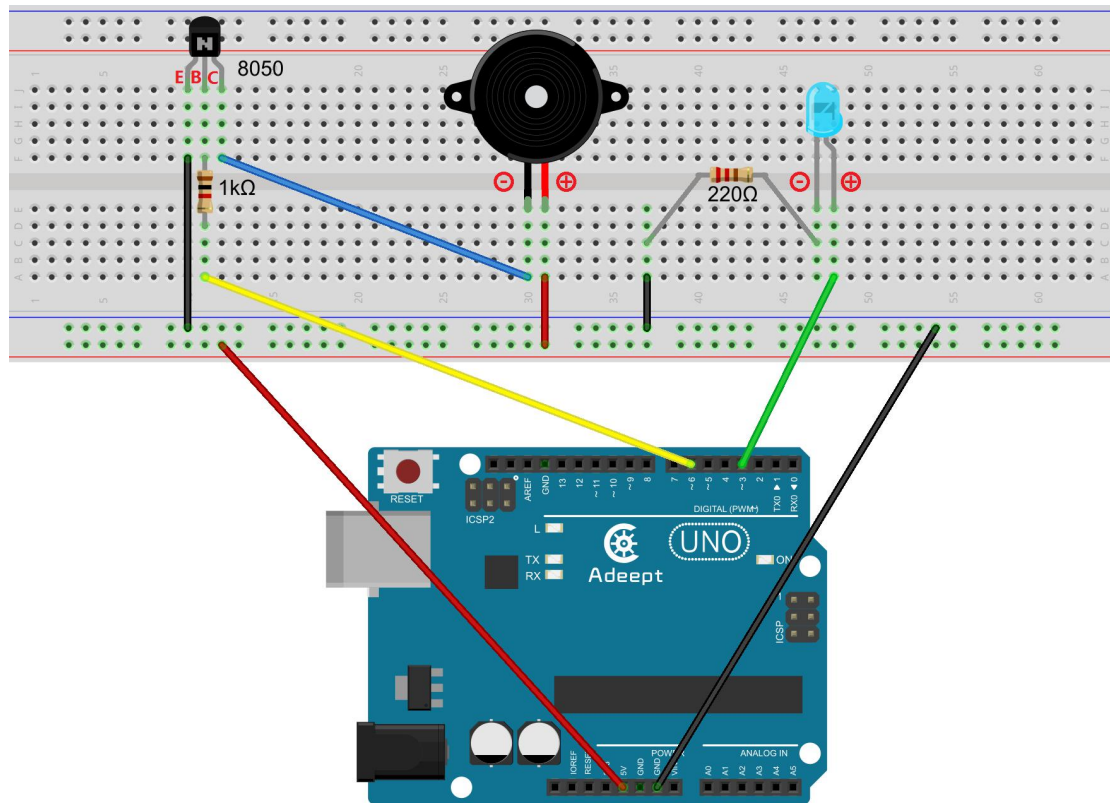
pin: the pin on which to stop generating the tone

*Returns:*

nothing

## Procedures

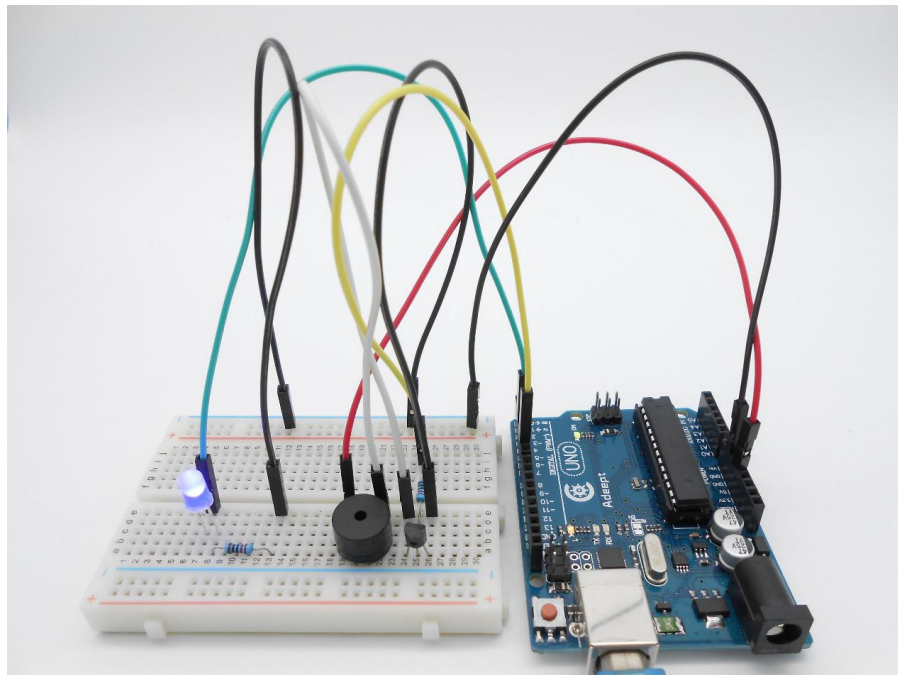
## Step 1: Build the circuit



## Step 2: Program

## Step 3: Compile the program and upload to Adept UNO board

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



## Lesson 11 LCD1602 Display

### Overview

In this lesson, we will learn how to use a character display device - LCD1602 on the Arduino platform. We first make the LCD1602 display a string "Hello Geeks!" scrolling, and then "Adept" and "www.adeept.com" statically.

### Components

- 1 \* Adept UNO R3 Board
- 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) is the bits that you're writing to a register when you write, or the values when you read.
- There are 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:*

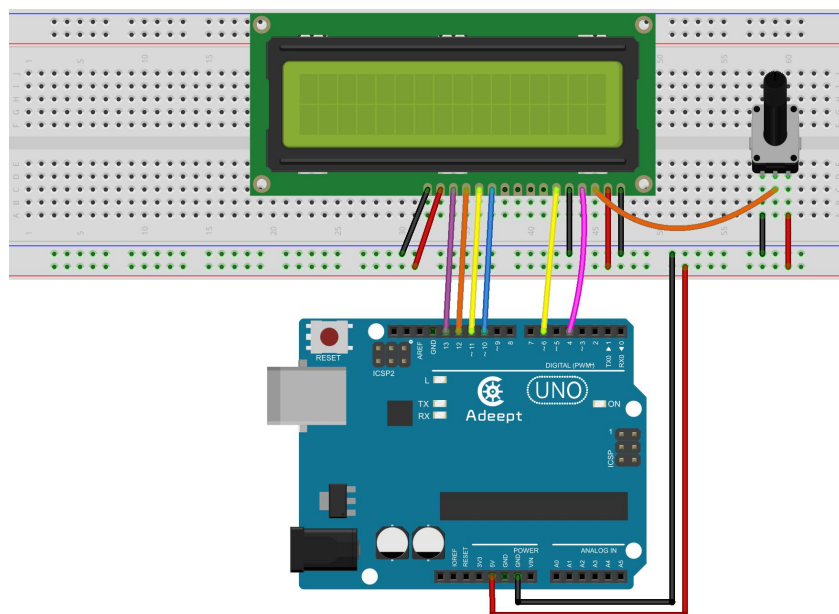
`lcd.clear()`

*Parameters:*

lcd: a variable of type LiquidCrystal

## Procedures

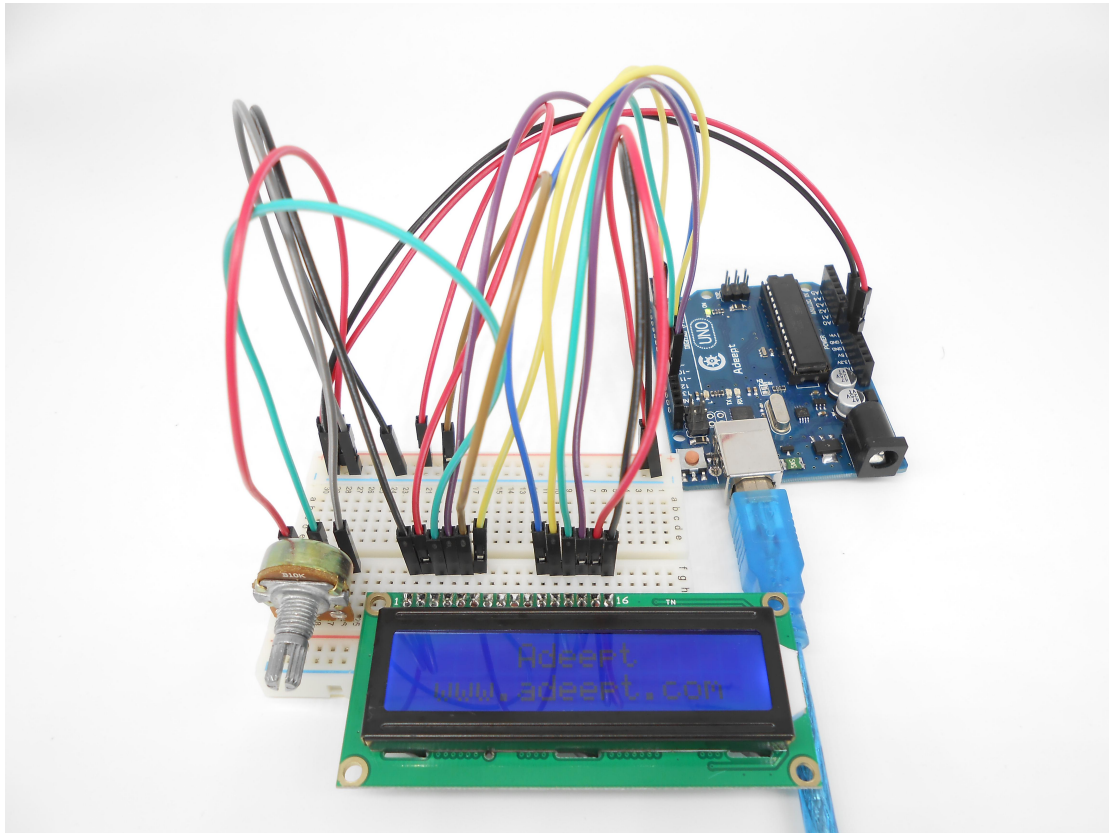
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

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



## Summary

After learning the experiment, you should have already mastered the driver of the LCD1602. Now you can make something more interesting based on this lesson and the previous lessons learned.

## Lesson 12 7-segment Display

### Overview

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

### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 220Ω Resistor
- 1 \* 7-segment Display
- 1 \* Breadboard
- Several jumper wires

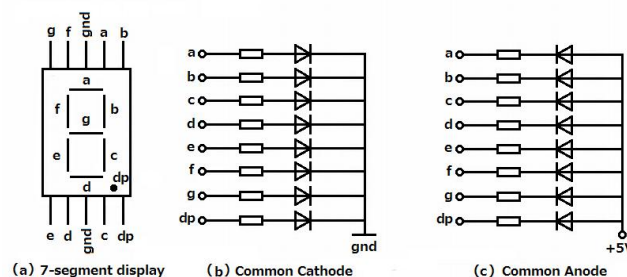
### Principle

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). The segments respectively named a, b, c, d, e, f, g, and dp.

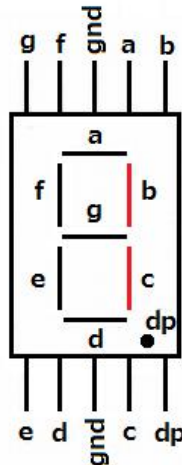
The segment display can be divided into two types: common anode and common cathode segment displays, by internal connections.



For a common-anode LED, the common anode should be connected to the power supply (VCC); for a common-cathode LED, the common cathode should be connected to the ground (GND).

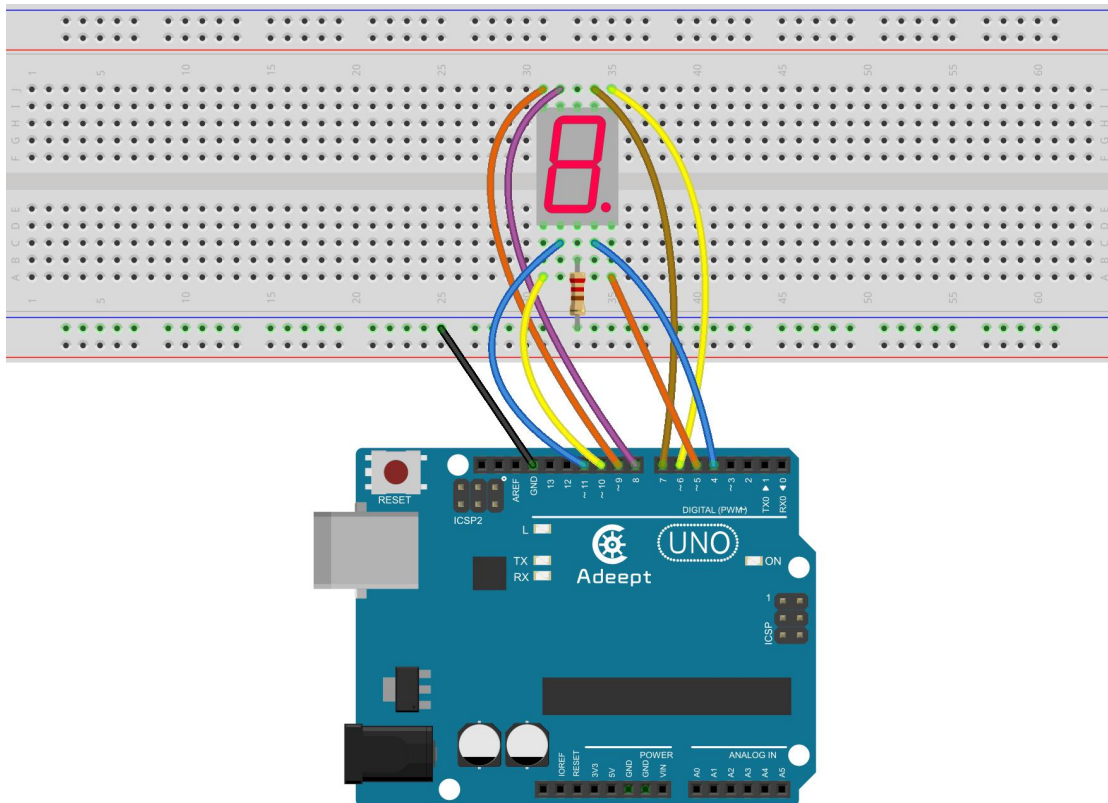
Each segment of a segment display is composed of an LED, so a resistor is needed for protecting the LED.

A 7-segment display has seven segments for displaying a figure and one more for displaying a decimal point. For example, if you want to display a number '1', you should only light the segment b and c, as shown below.



## Procedures

### Step 1: Build the circuit

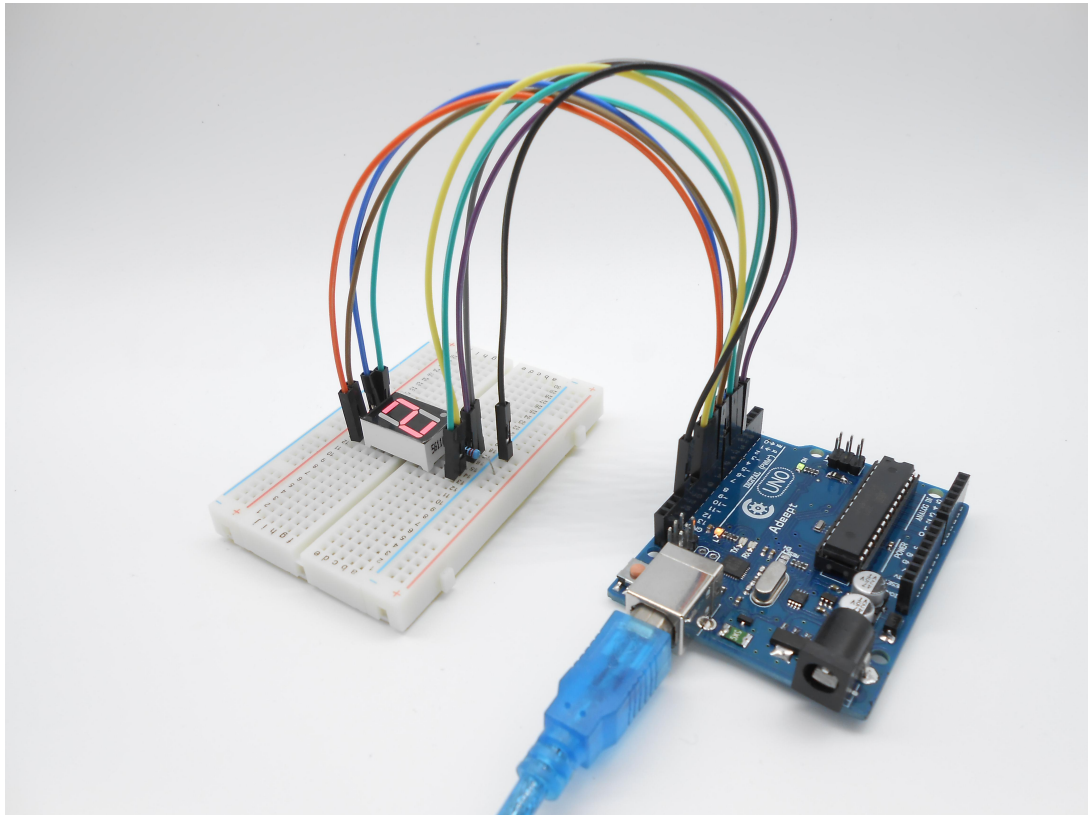


### Step 2: Program

---

Step 3: Compile the program and upload to Adept UNO board

Now, you should see the number 0~9 and characters A~F displayed in turn on the segment display.



## Summary

Through this lesson, you should have learned the principle and programming of the segment display. You can use what you've learned in the previous lessons to modify the code provided in this lesson to make cooler works.

## Lesson 13 A Simple Counter

### Overview

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

### Components

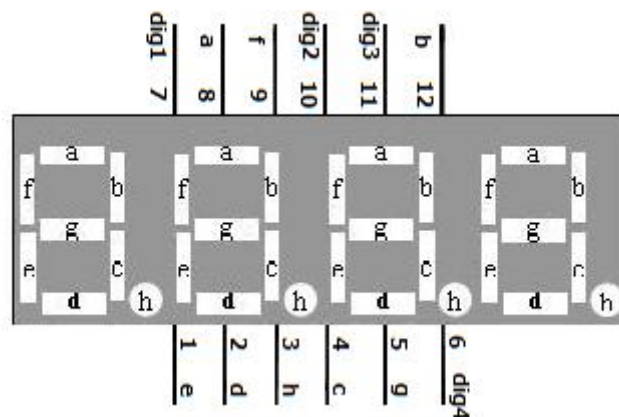
- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 4-digit 7-segment Display
- 8 \* 220Ω Resistor
- 2 \* Button
- 1 \* Breadboard
- Several jumper wires

### Principle

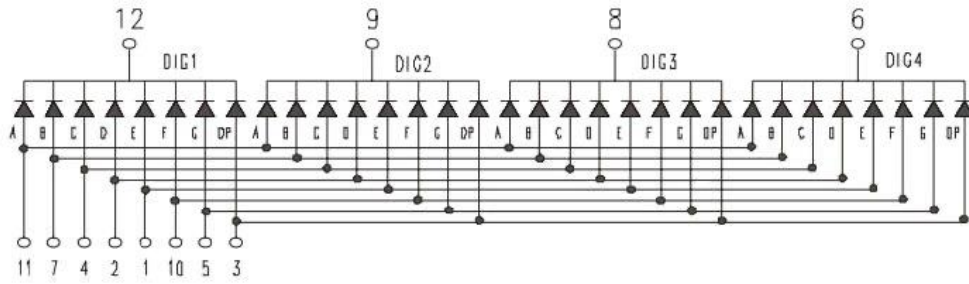
The 4-digit segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

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

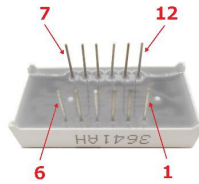
The 4-digit segment display is a 4\*8-shaped LED display device composed of 32 LEDs (including four decimal points). The segments are named respectively a, b, c, d, e, f, g, h, dig1, dig2, dig3, and dig4.



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

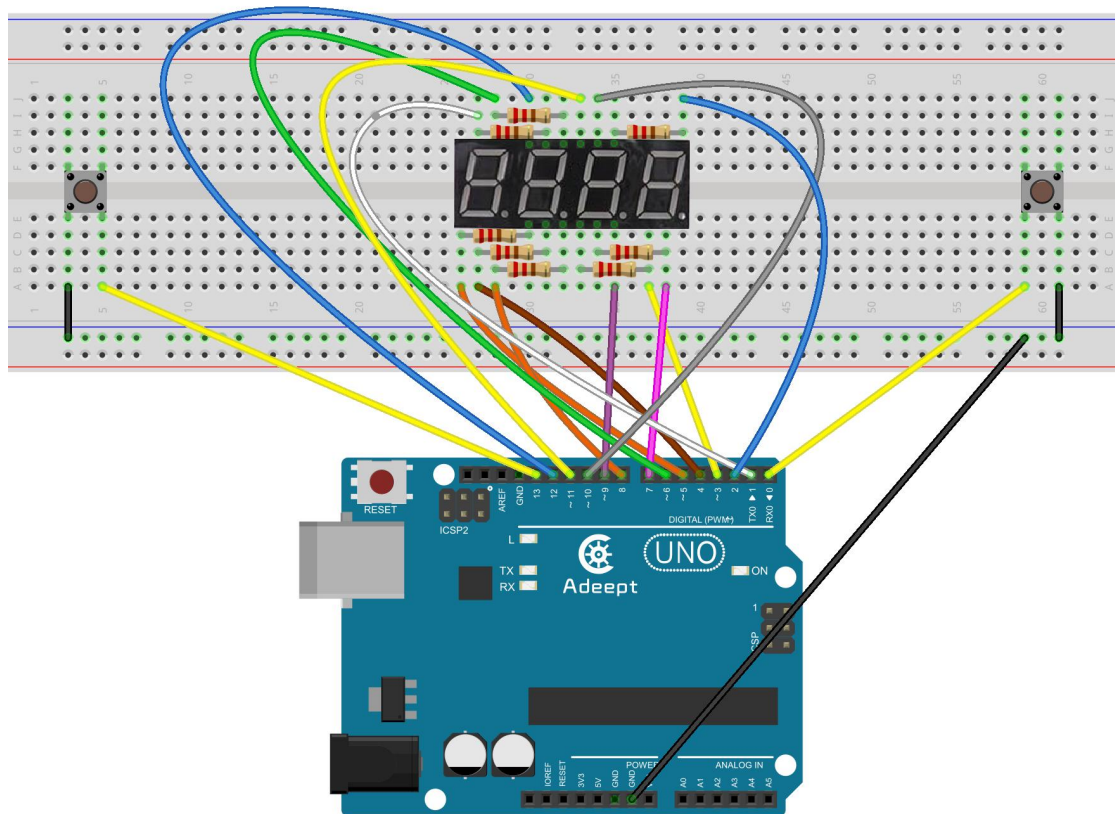


The pin number is as follows:



## Procedures

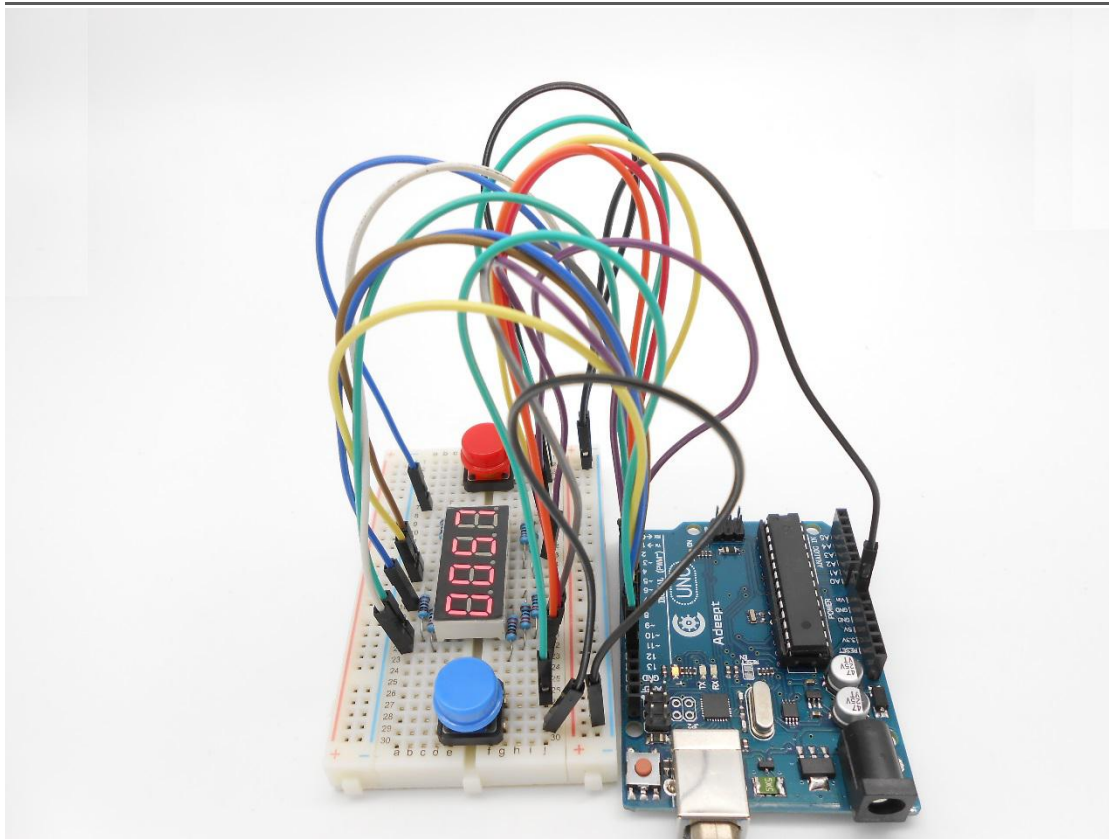
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, press one button of the two, and 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. Then, try your own ways to use this tool to impress makers!

# Lesson 14 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.

## Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 8\*8 Dot-matrix
- 2 \* 74HC595
- 1 \* Breadboard
- Several jumper wires

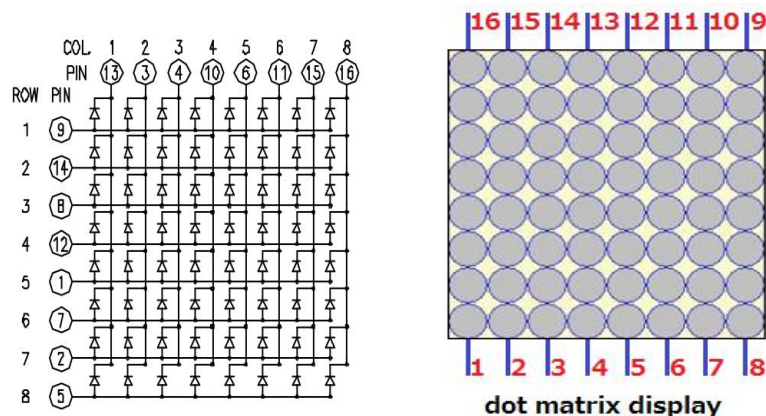
## Principle

### 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:



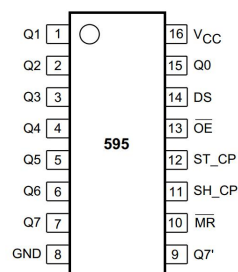
A 8\*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column is set as low level, then the LED will be lit.

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)

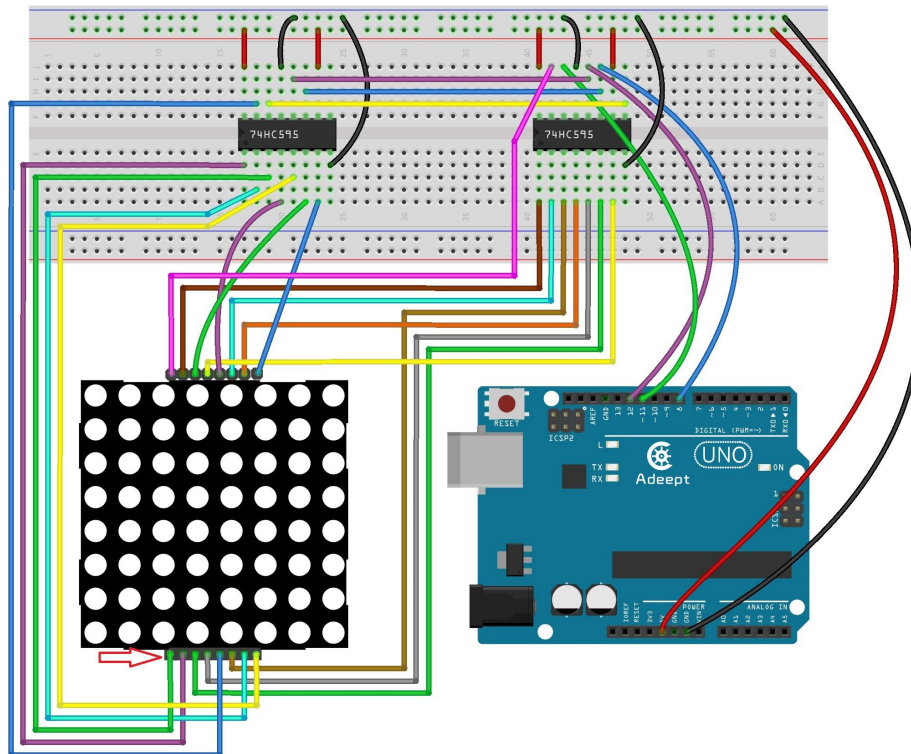
value: the data to shift out. (byte)

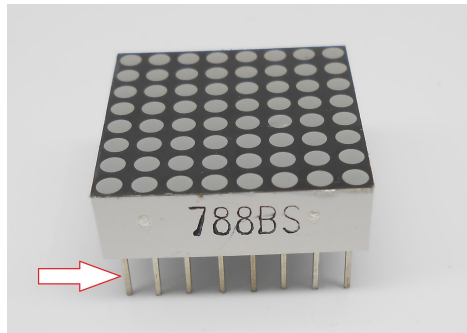
#### Returns:

None

### Procedures

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

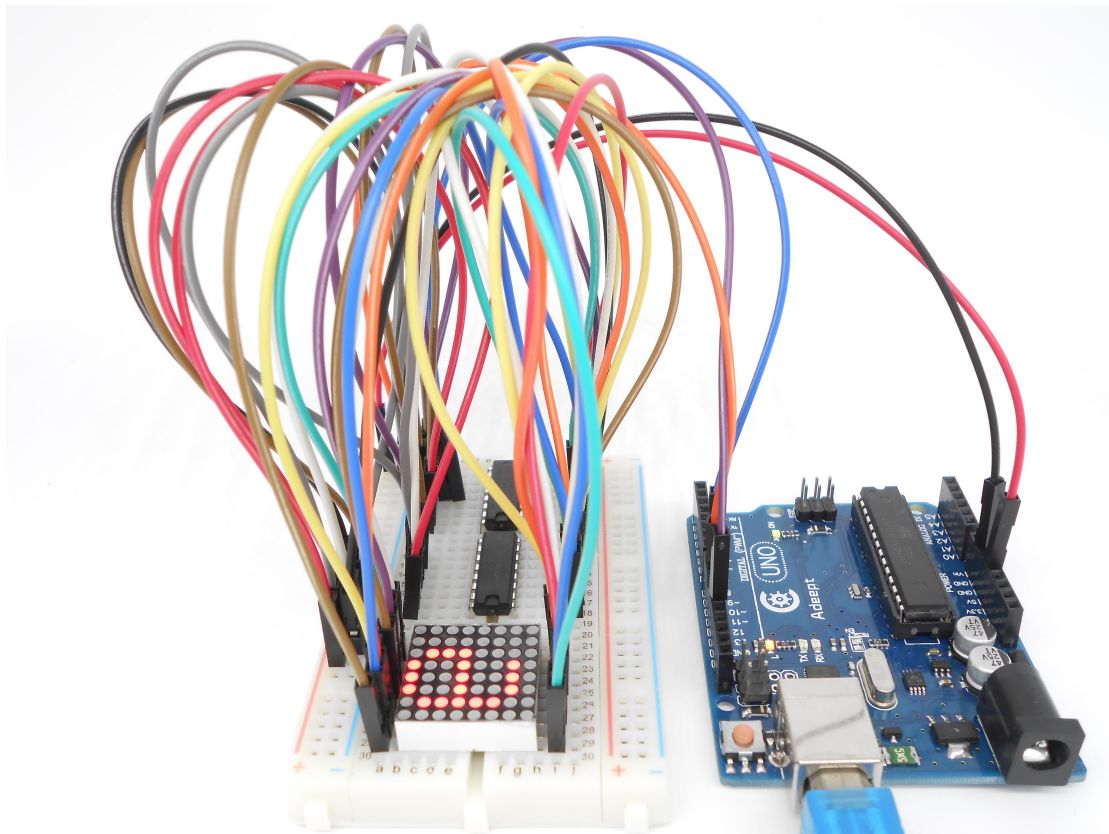




Step 2: Program

Step 3: Compile the program and upload to Adept 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 15 Controlling a Servo

## 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.

## Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* Servo
- Several jumper wires

## Principle

### *1. Servo motor*

The servo motor has 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. Usually the signal pin is yellow, orange or white, and should be connected to a digital pin on the Arduino board. Note that the servo motor draws a considerable amount of power, if you need to drive more than one or two servos, you'll probably need to power them with an extra 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

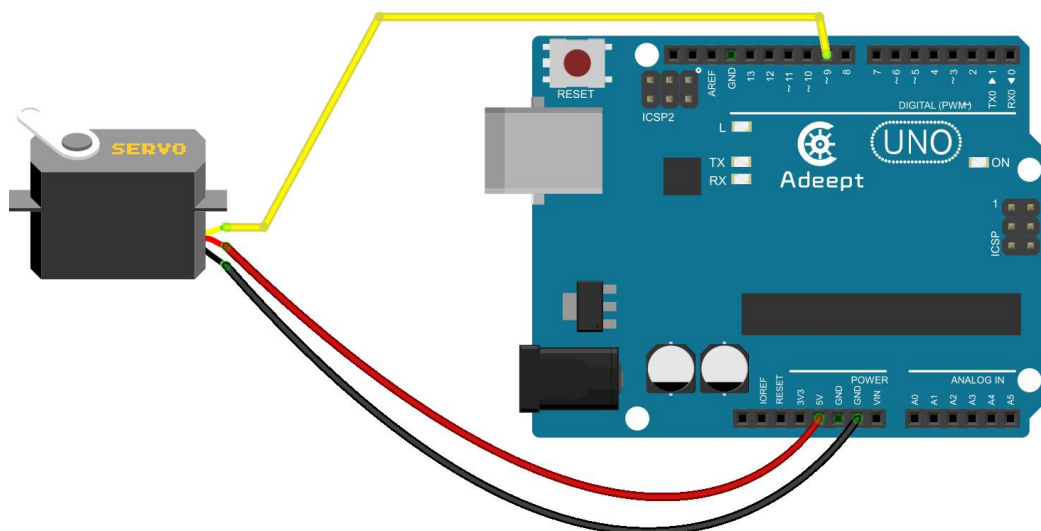
pin: the number of the pin that the servo is attached to

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

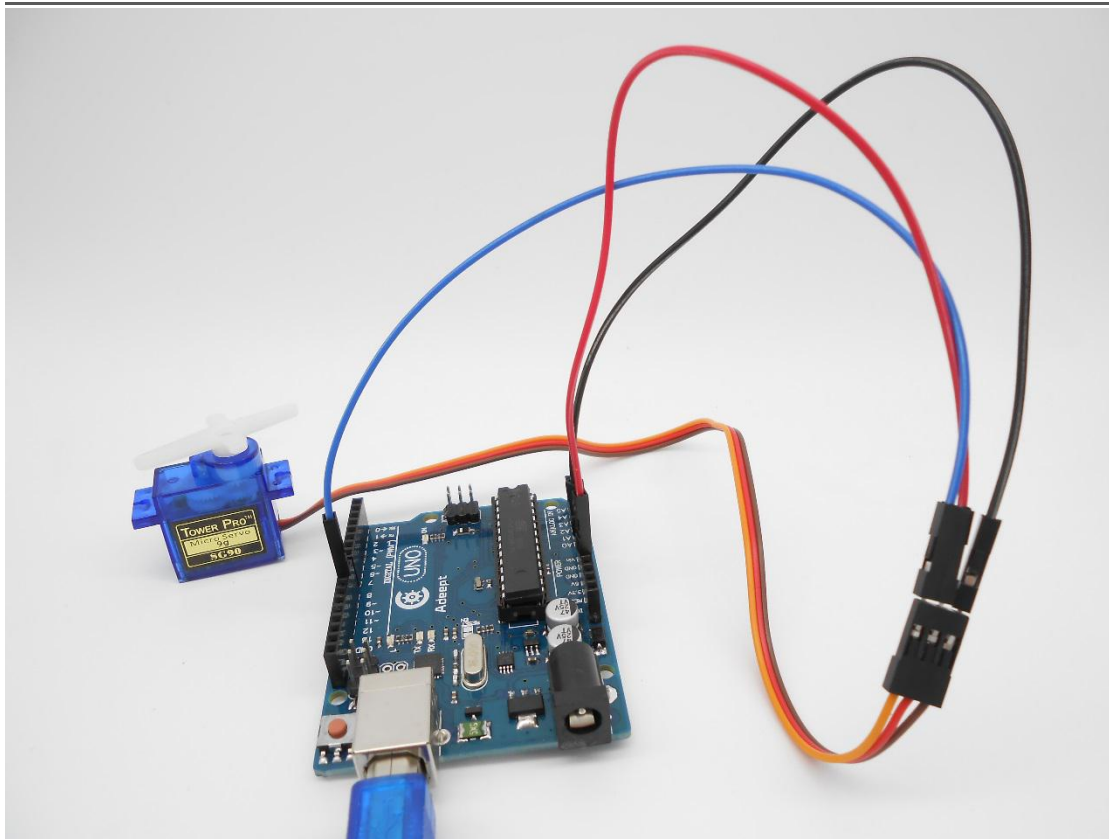
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, you should see the servo rotate from 0 to 180 degrees, and then do it in the opposite direction.



## Summary

After learning, you should have known that the Arduino provides a servo library for you to control a servo. By using this library, you can easily control a servo by programming. Just fully play your imagination and make some interesting applications!

## Lesson 16 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.

### Components

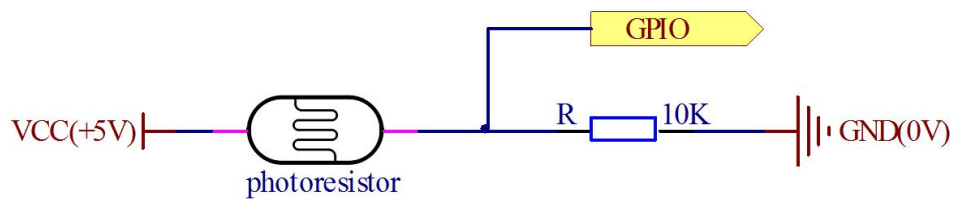
- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* LCD1602
- 1 \* Photoresistor
- 1 \* 10k $\Omega$  Resistor
- 1 \* 10k $\Omega$  Potentiometer
- 1 \* Breadboard
- Several jumper wires

### 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.

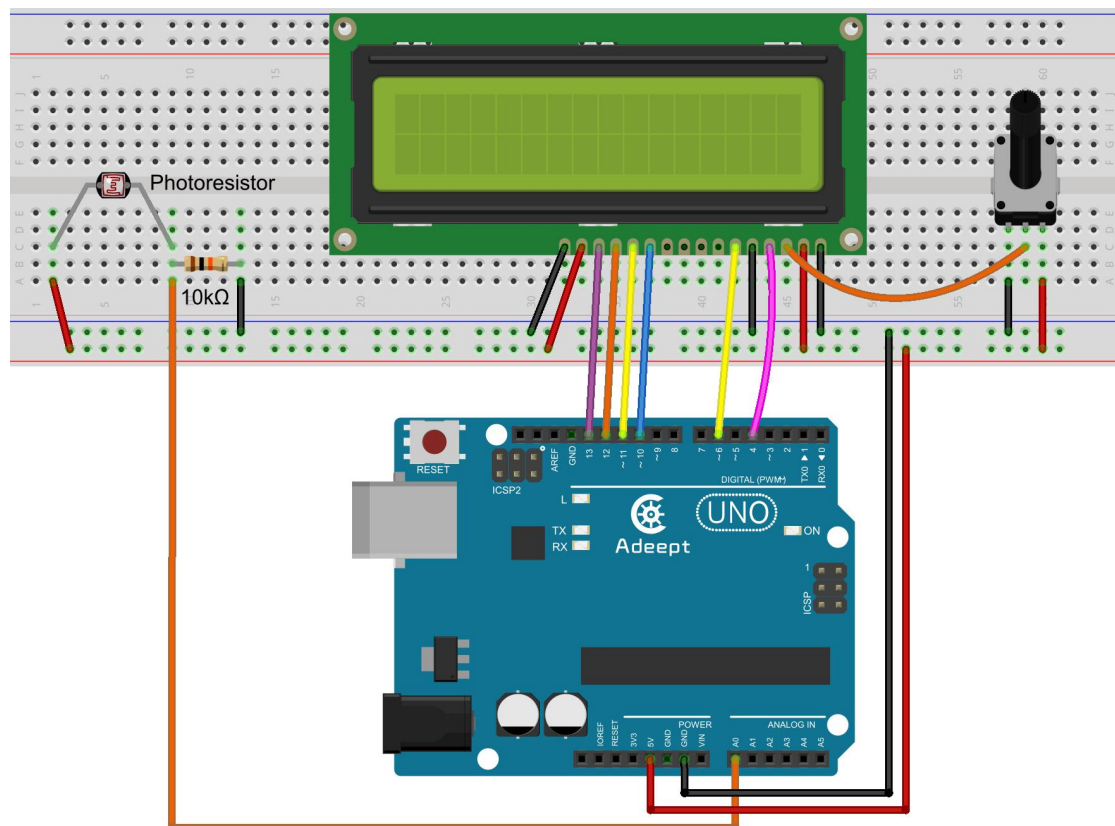
The schematic diagram of this experiment is shown below:



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

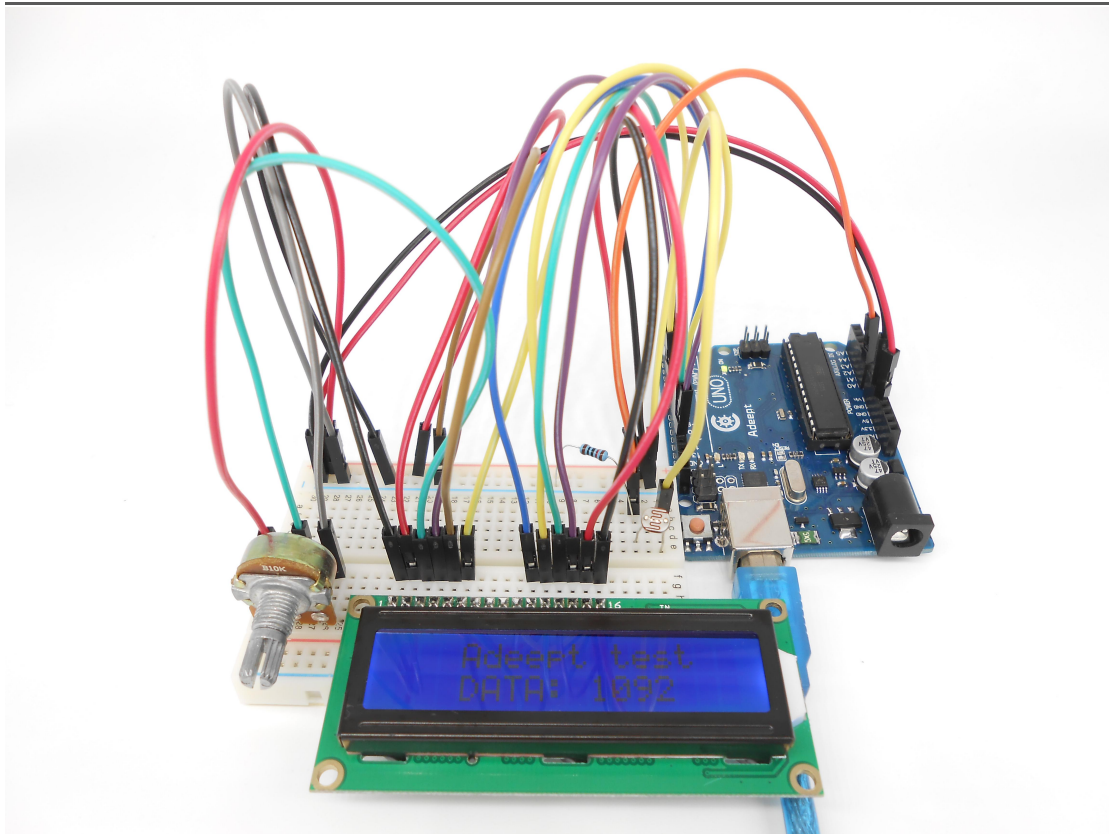
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept 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 17 Measuring Temperature by a Thermistor

## Overview

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

## Components

- 1 \* Adept UNO R3 Board
- 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. In this experiment we use an MF52 NTC-type thermistor, and it is usually used as a temperature sensor.

The key parameters of an MF52 thermistor:

**B-parameter: 3470.**

**25°C resistance: 10kΩ.**

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

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

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

**R**: the nominal resistance of the thermistor at room temperature T2

**e**: 2.718281828459

**B**: one of the important parameters of thermistor

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

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

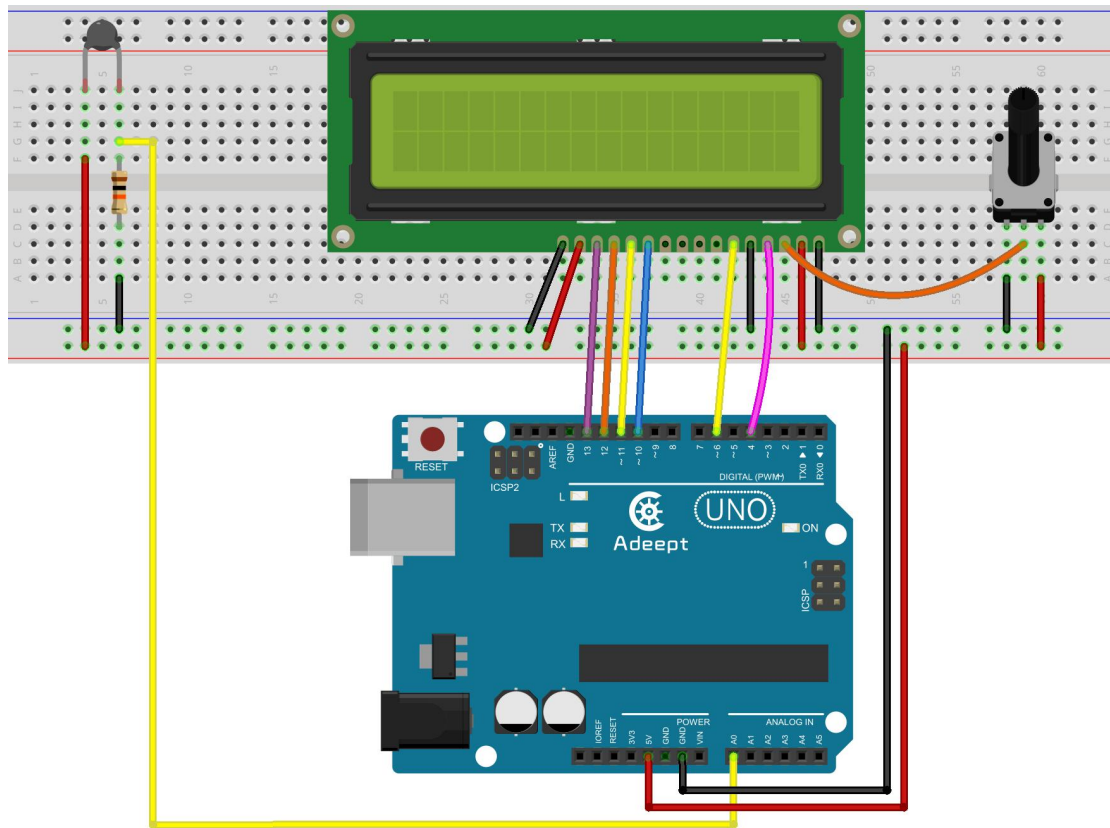
Kelvin temperature = 273.15 (absolute temperature) + degrees Celsius;

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

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

## Procedures

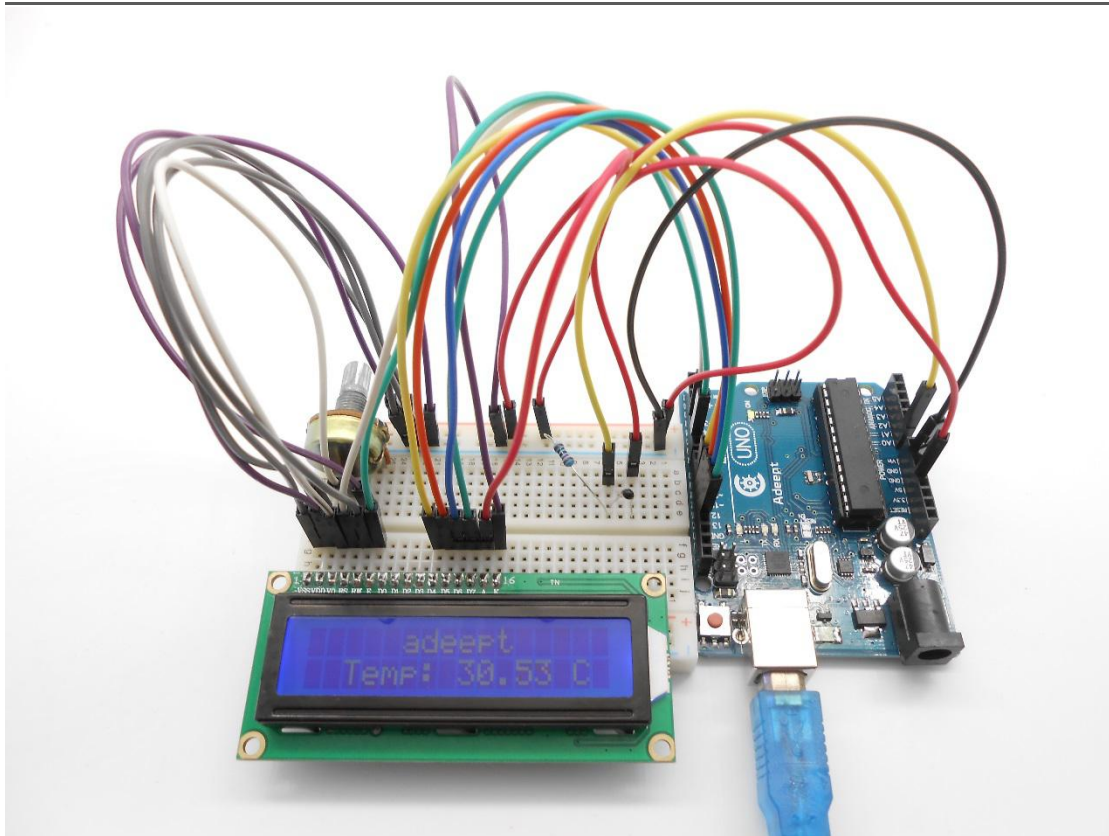
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

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



## Summary

After this lesson, you may have learned to use a thermistor to measure the temperature. Next, you can use it to make some interesting applications.

## Lesson 18 IR Remote Controller

### Overview

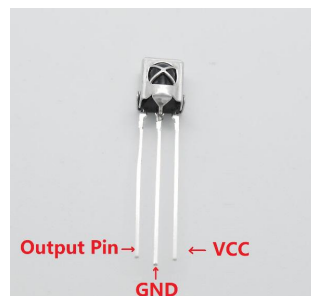
In this lesson, we will learn how to use an IR receiver to receive signals from a remote controller.

### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* IR Receiver HX1838
- 1 \* Remote Controller
- 1 \* Breadboard
- Several jumper wires

### Principle

The IR receiver HX1838 can receive signals from an infrared (IR) remote controller. It has only three pins: signal, VCC and GND. So it is simple to connect with an Arduino board.



The following figure shows an IR remote controller:



In this experiment, we program the Arduino board to receive the infrared signals, and then send the received data to Serial Monitor. In the program, we use the Arduino-IRremote-master library (provided).

### **Note:**

Before using this library, you have to delete the [RobotIRremote](#) directory in your Arduino IDE directory (check in IDE by File->Preferences, and see the path in the Browse dialog box), and delete the [RobotIRremote](#) directory in the system Documents folder. For example, if your computer is running on Windows 7, you need to delete the [RobotIRremote](#) directory in

*C:\Program Files (x86)\Arduino\libraries*

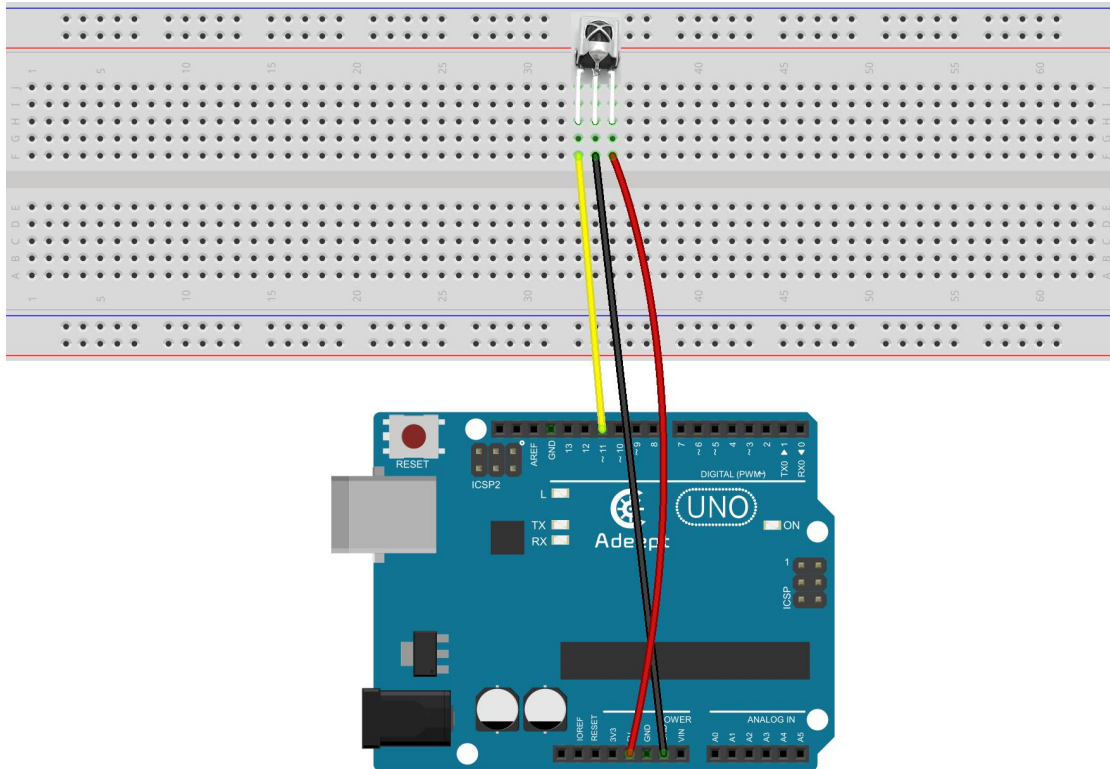
and

*C: \Users\SJG\Documents\Arduino\libraries.*

Otherwise, when you compile the program, errors will be prompted

## **Procedures**

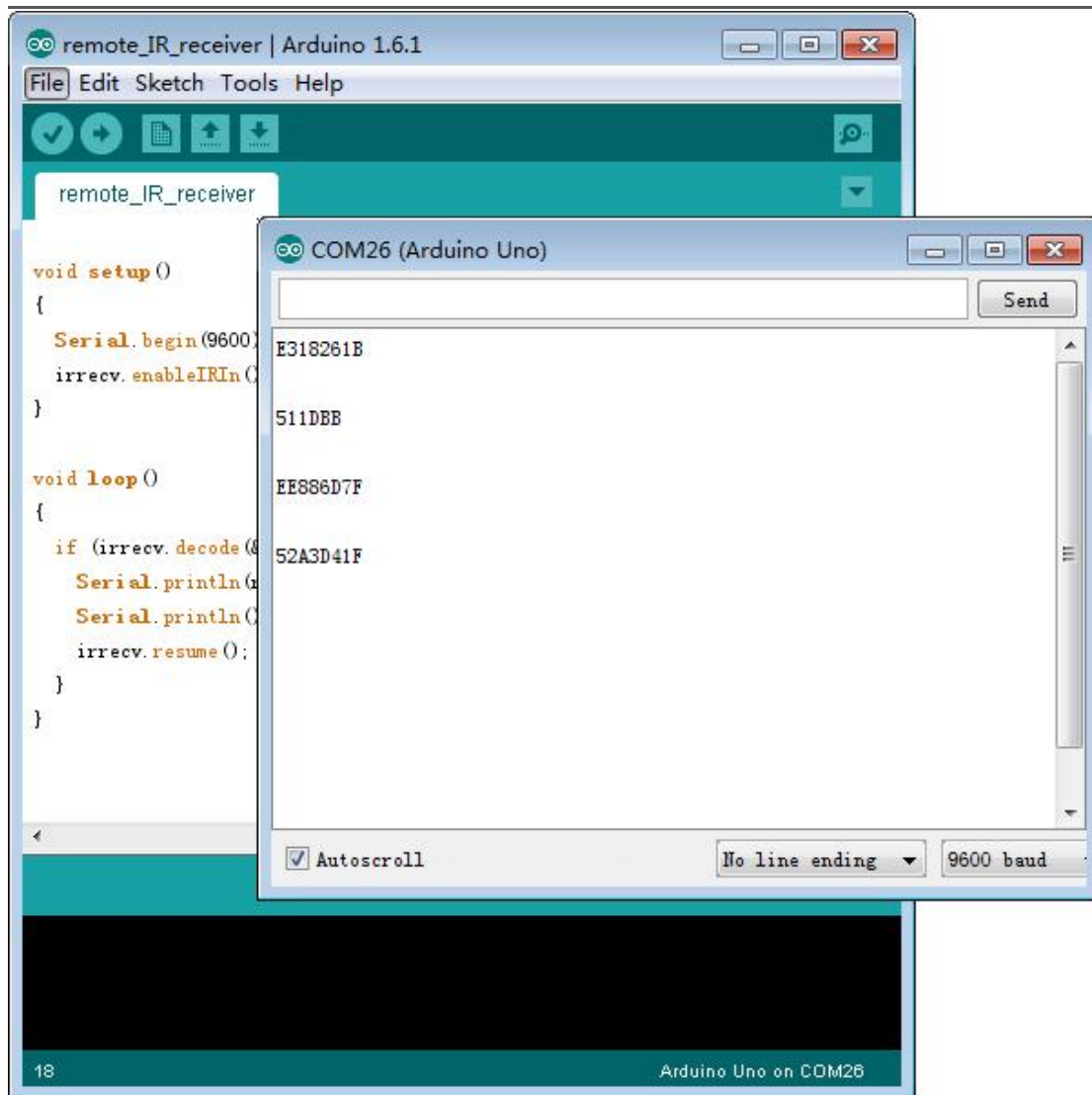
Step 1: Build the circuit

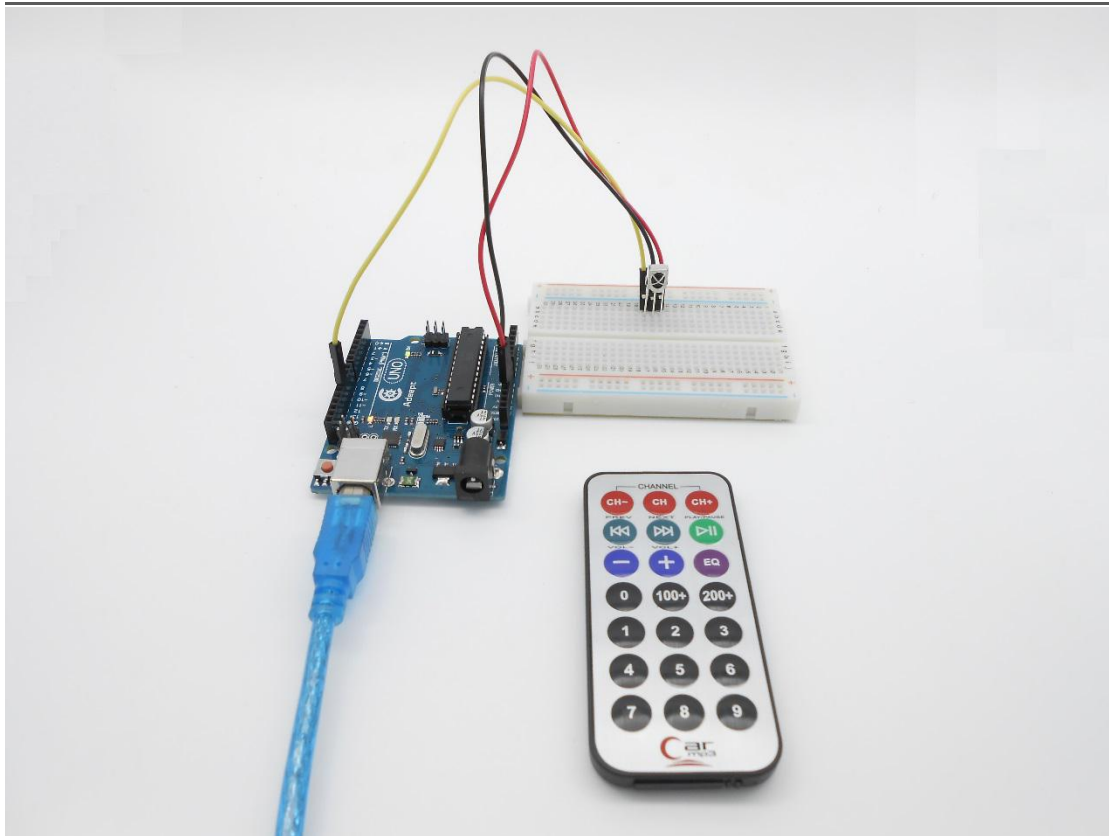


Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, press a button on the remote controller, and you will see the button number displayed on Serial Monitor.





## Summary

Now you should have mastered the basic principle of the infrared remote controlling. Try to apply the principle and make more creations!

# Lesson 19 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.

### Components

- 1 \* Adept UNO R3 Board
- 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:  $\pm 2^{\circ}\text{C}$

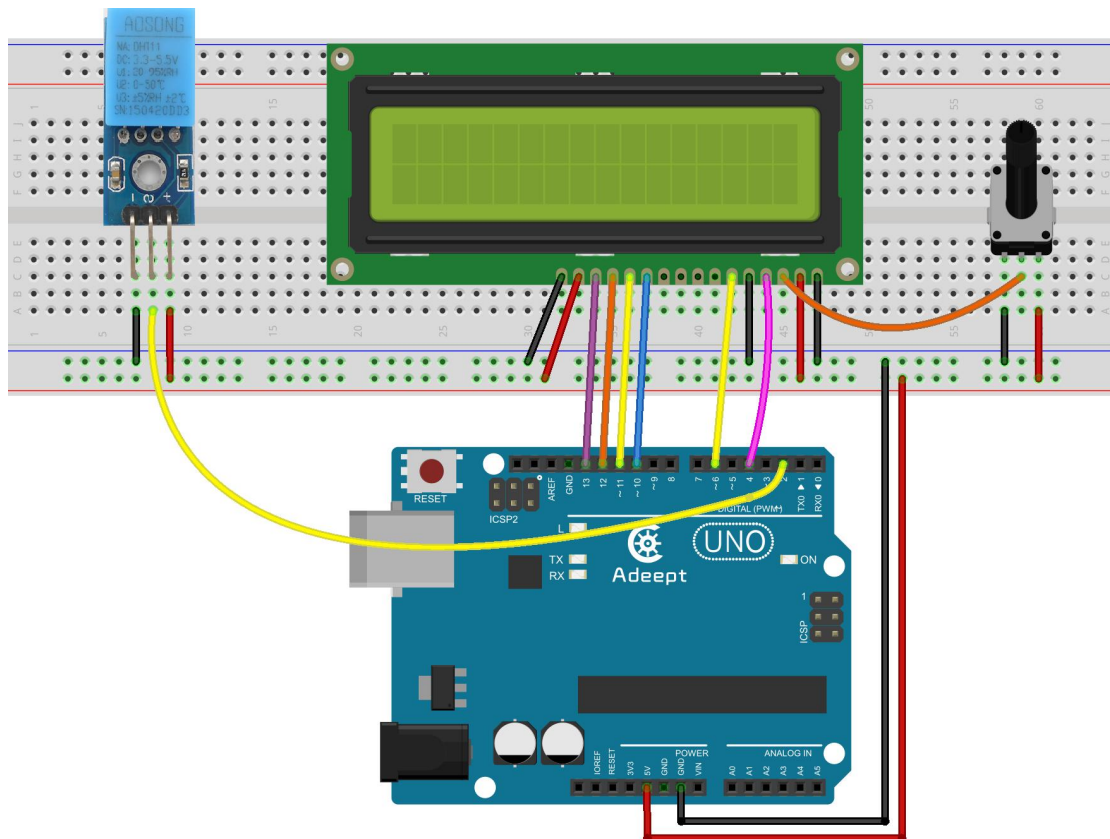
'+' : VCC (3.3~5.5V)

'-' : GND (0V)

'S' : data pin

## Procedures

Step 1: Build the circuit



Step 2: Program

```

/*****
File name: 19_DHT11.ino
Description: you can see the temperature and humidity data
            displayed on the LCD1602.
Website: www.adeept.com
E-mail: support@adeept.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
}

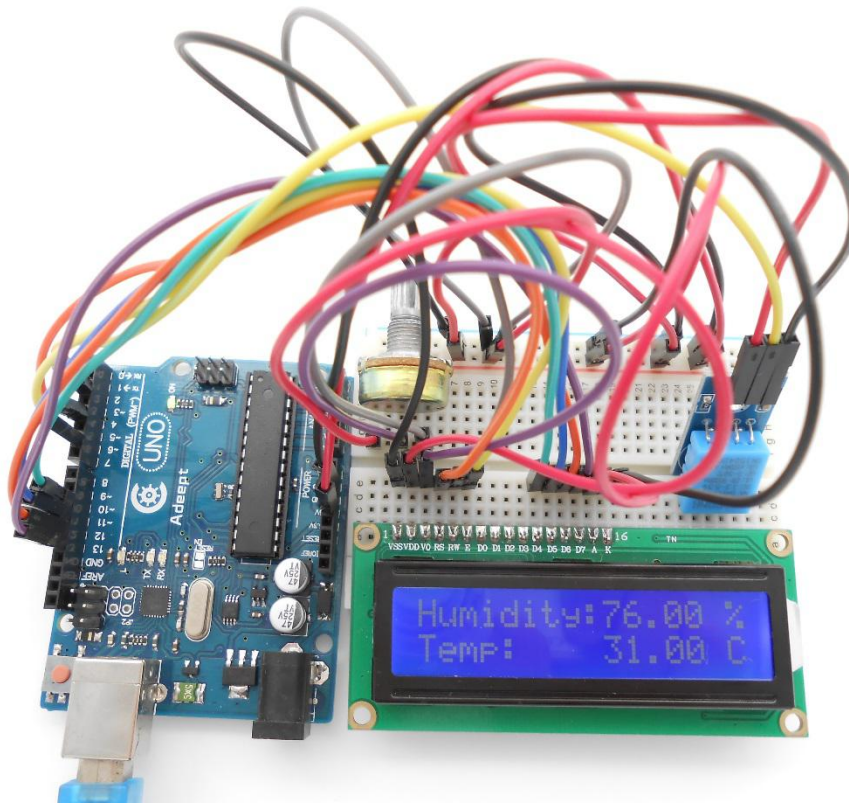
void loop()
{
    int chk = DHT11.read(DHT11PIN);

    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
}
```

Step 3: Compile the program and upload to Adept UNO board

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



## Summary

Through this lesson, you should be able to measure the room's temperature and humidity. Now try to make a simple smart home system based on this and the previous lessons.

## Lesson 20 Ultrasonic Distance Sensor

### Overview

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

### Components

- 1 \* Adept UNO R3 Board
- 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 to an object ranging from 2cm to around 3m.



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:

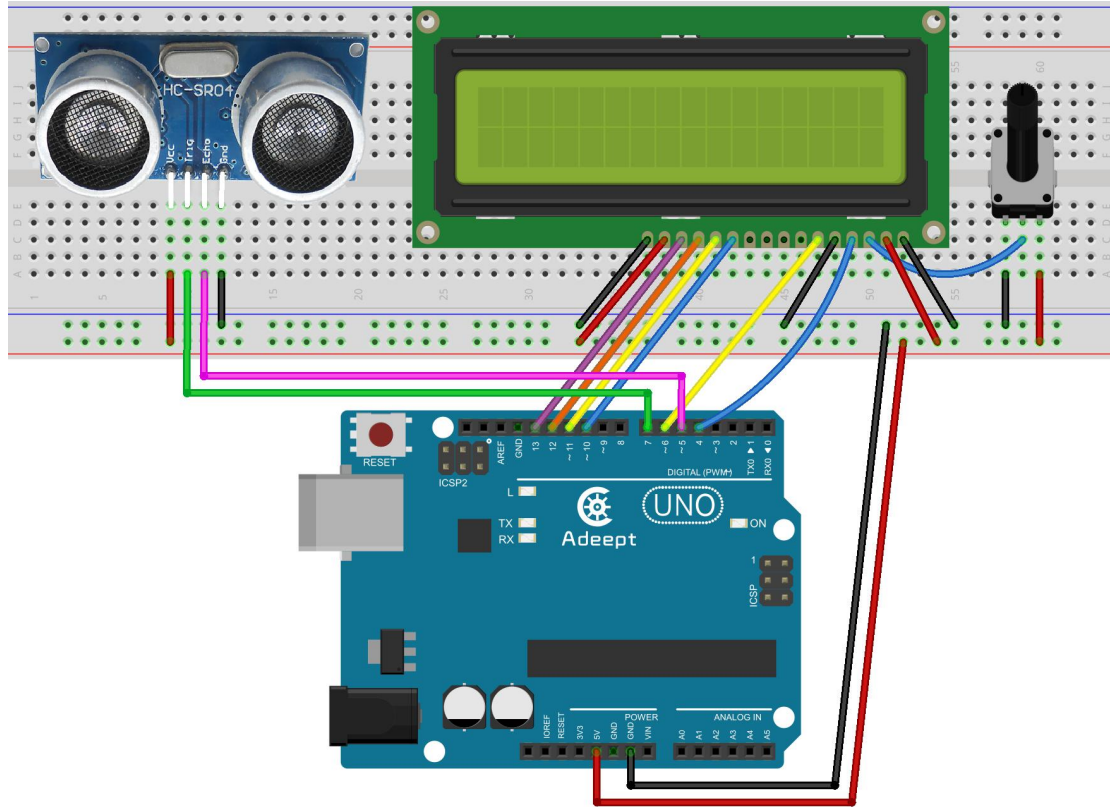
$$\text{RoundTrip} = \text{microseconds} / 29.$$

So, the formula for the one-way distance in centimeters is:

$$\text{microseconds} / 29 / 2$$

## Procedures

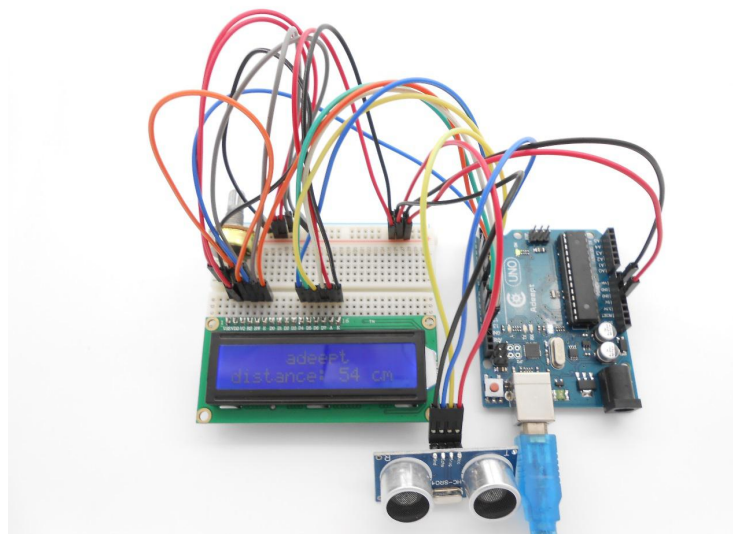
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, change the distance between the ultrasonic module and the obstacle, and you will find the distance value displayed on the LCD1602 changed.



# Lesson 21 3-axis Accelerometer—ADXL345

## Overview

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

## Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* LCD1602
- 1 \* 10kΩ 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  $1.0^\circ$ .

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:*

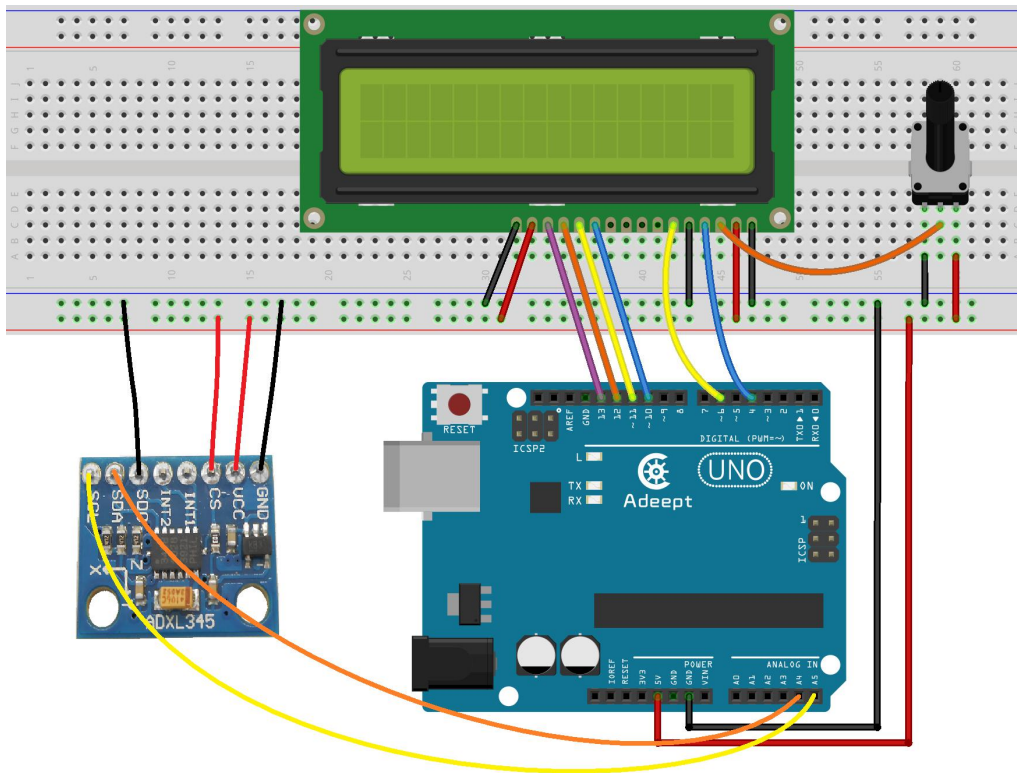
lcd: a variable of type LiquidCrystal

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

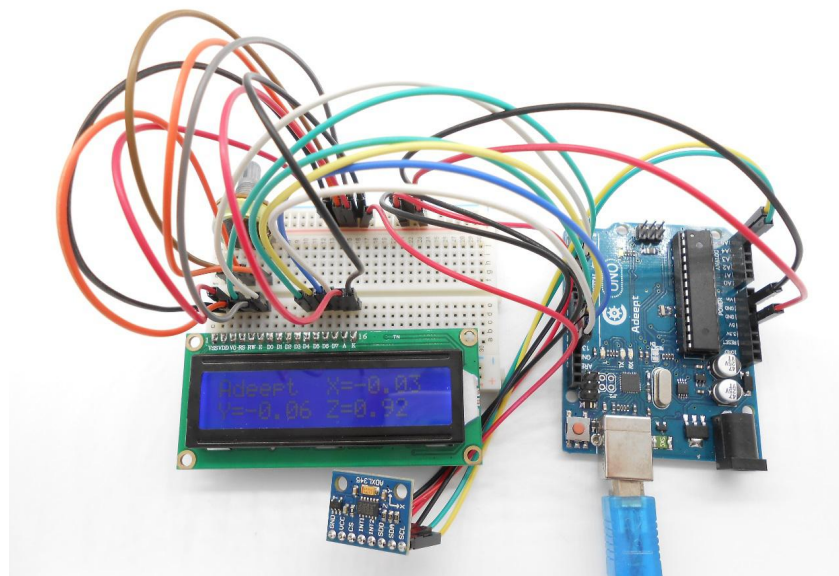
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

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



---

## Summary

After learning this lesson, you should have got the usage of the ADXL345. Next, you can use it to make more interesting applications.

## Lesson 22 4x4 Matrix Keypad

### Overview

In this lesson, we will learn how to use a matrix keypad.

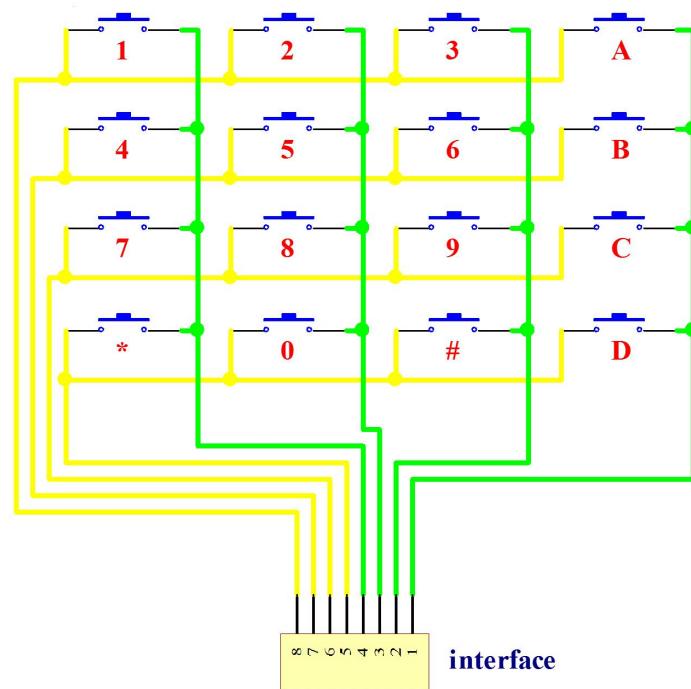
### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* 4x4 Matrix Keyboard
- 1 \* Breadboard
- Several jumper wires

### Principle

In order to save the resources of the microcontroller ports, we usually connect the buttons of a matrix in practical projects.

See the schematics of a 4x4 matrix keyboard as below:

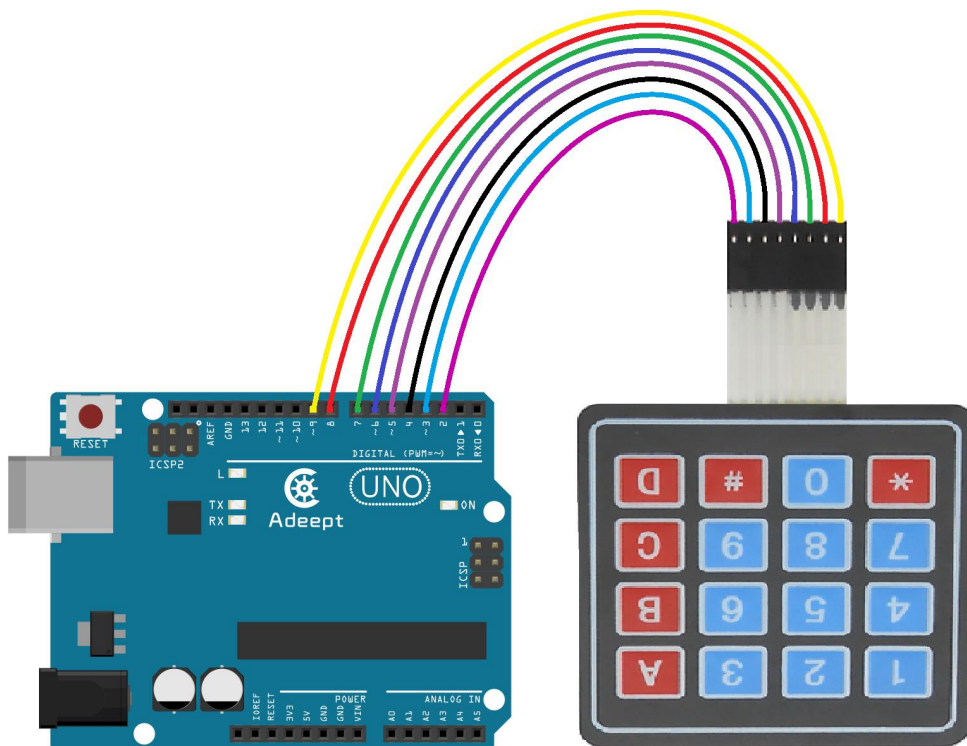




In this tutorial, we use the Keypad library. Before programming, please install the library.

## Procedures

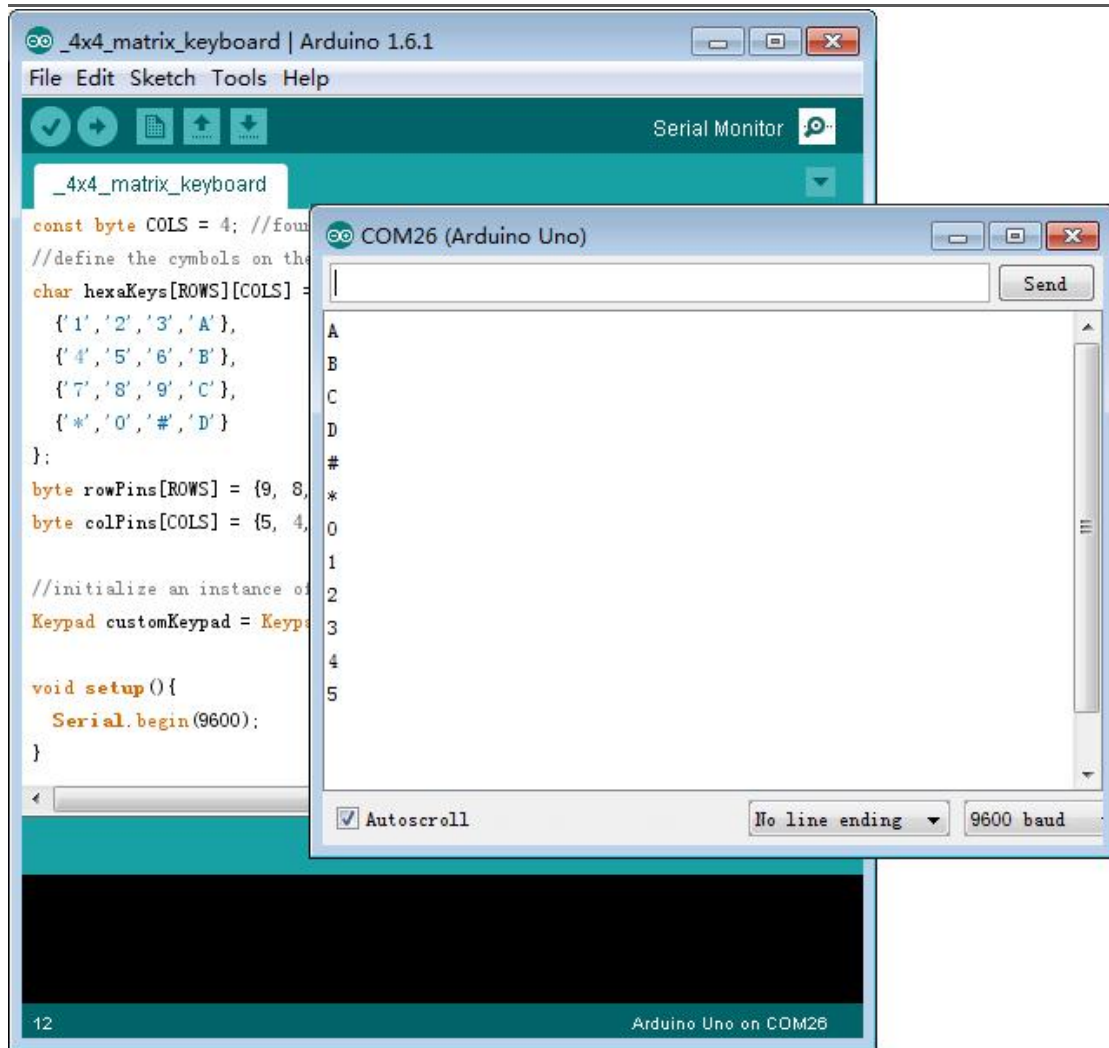
Step 1: Build the circuit

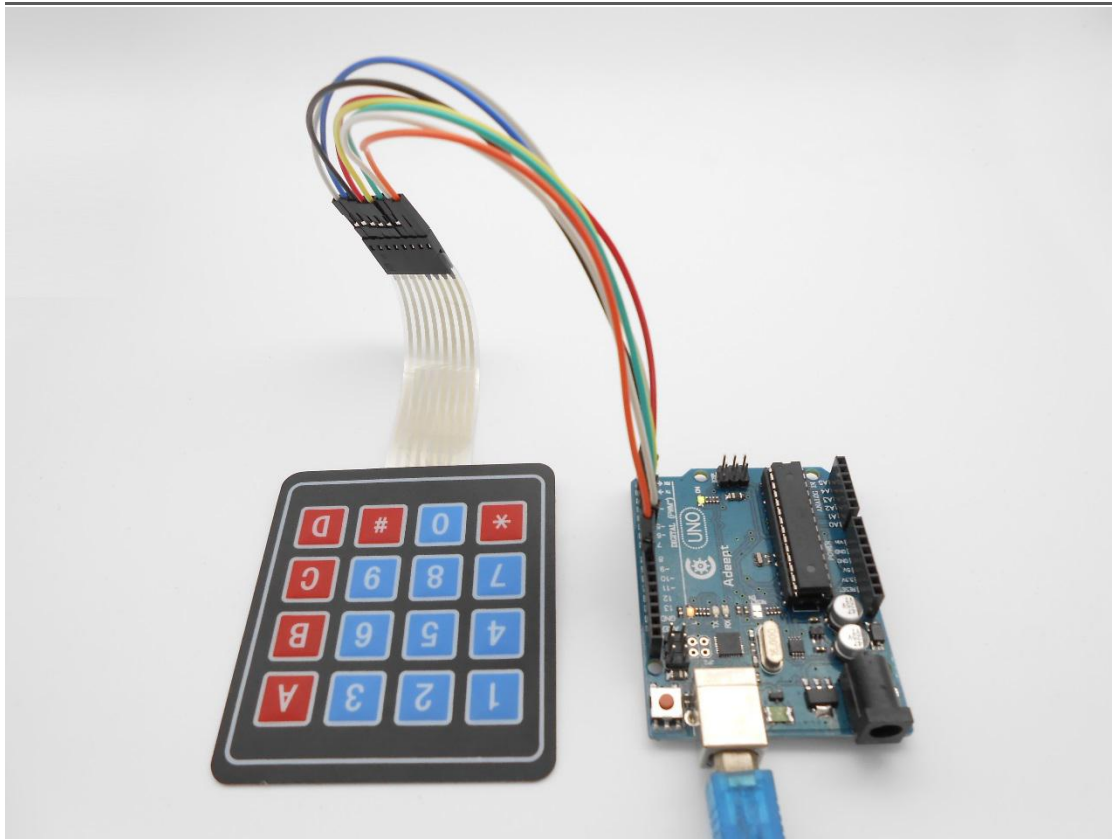


Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Now, press a button on the 4x4 matrix keyboard, and you will see the corresponding key value displayed on Serial Monitor.





## Lesson 23 Controlling DC motor

### Overview

In this comprehensive experiment, we will learn how to control the state of a DC motor with Arduino, and display the state by an LED at the same time. The state includes its running forward, reversing, acceleration, deceleration and stop.

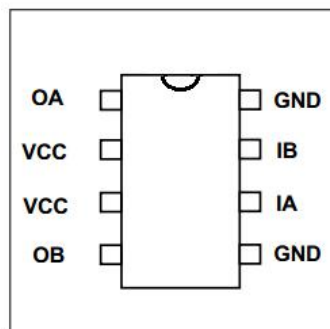
### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* L9110 DC Motor Driver
- 1 \* DC Motor
- 4 \* Button
- 4 \* LED
- 4 \* 220 $\Omega$  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 motors, and power switches and so on.



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

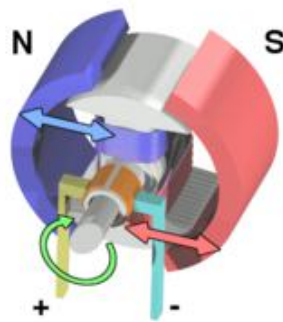
VCC: Power supply (+5V)

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  
}
```

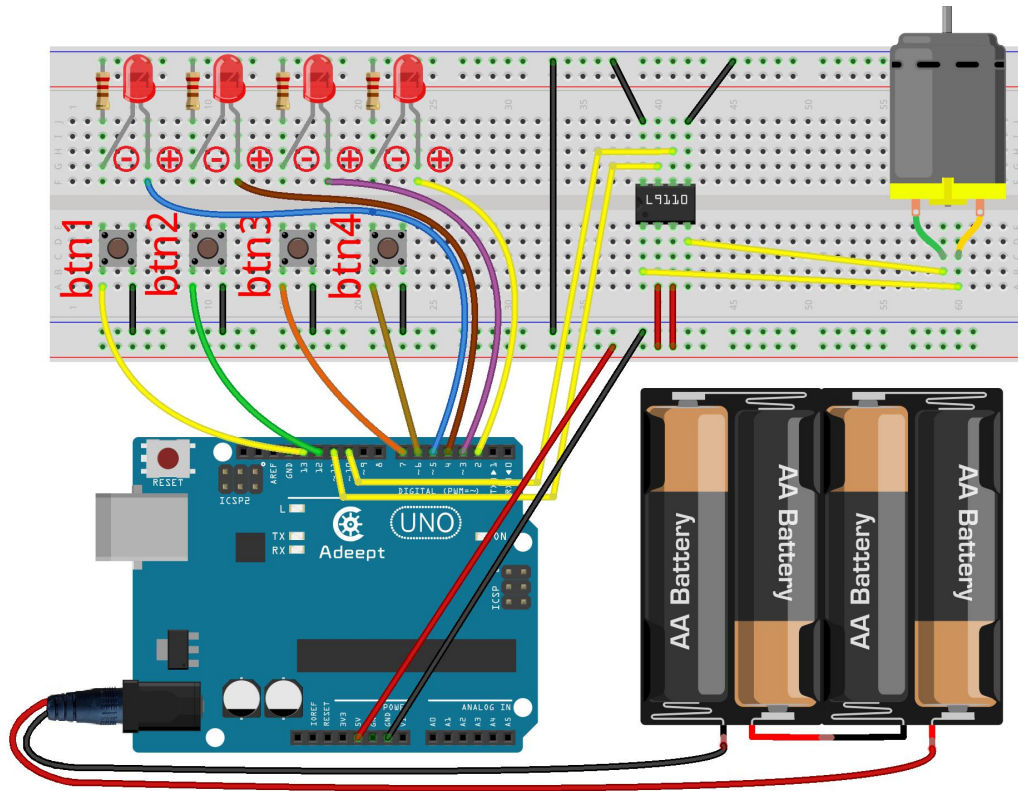
*Parameters:*

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

label: a value to compare the variable to

## Procedures

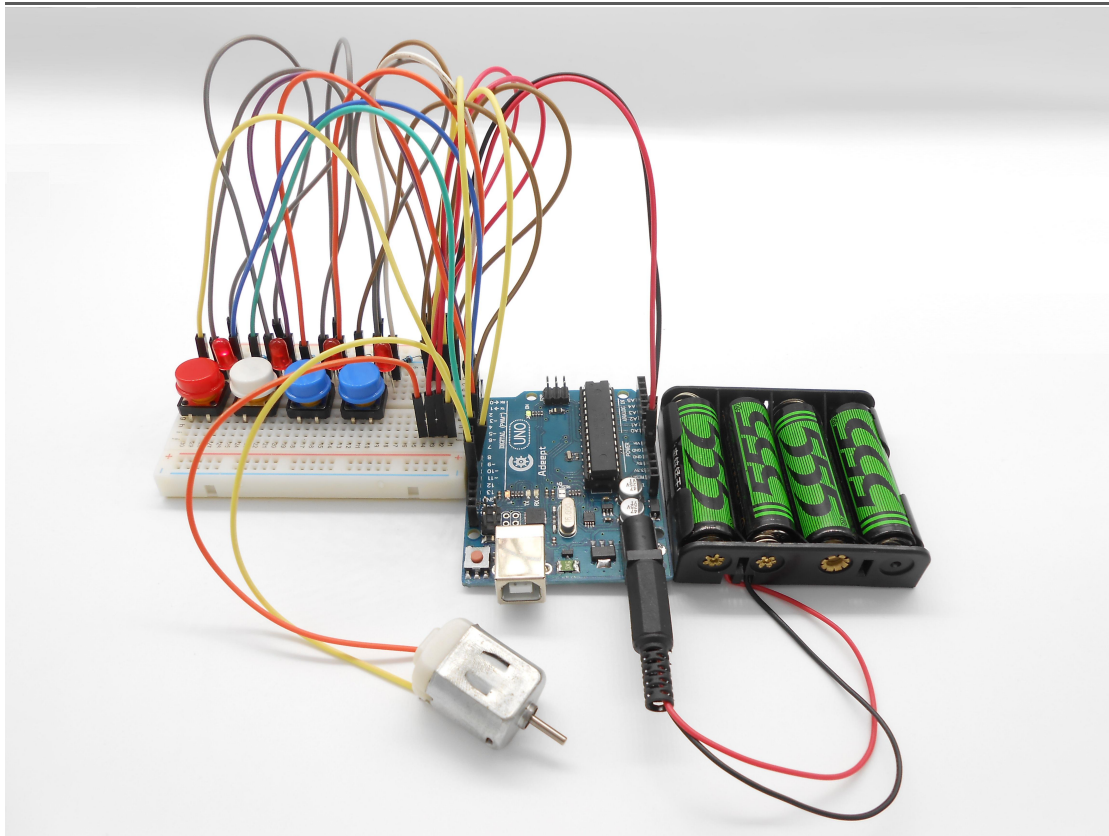
Step 1: Build the circuit (Make sure that the circuit connection is correct before powering on; otherwise it may cause the chips to burn.)



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Press the btn1 button to stop or run the DC motor; press btn2 to make it go forward or reverse; press btn3 to accelerate the motor; press btn4 to decelerate it. When any of the four buttons is pressed, the corresponding LED will flash, prompting that the current button is pressed down.



## Summary

Now you must have grasped the basic theory and programming of the DC motor after all the study. You not only can make it go forward and reverse, but also regulate its speed. Besides, you can do more awesome applications with what you've learnt.

## Lesson 24 PS2 Joystick

### Overview

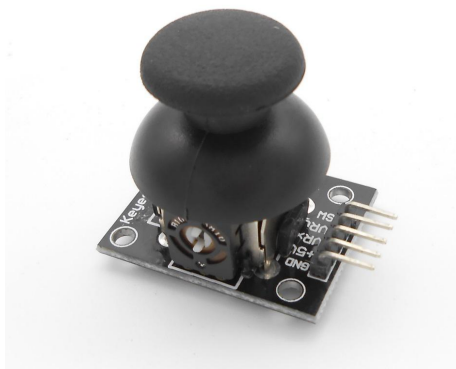
In this lesson, we will learn the usage of joy stick. We program the Arduino to detect the state of PS2 joystick, and display the data on an LCD1602.

### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* LCD1602
- 1 \* 10k $\Omega$  Potentiometer
- 1 \* PS2 JoyStick
- 1 \* Breadboard
- Several jumper wires

### Principle

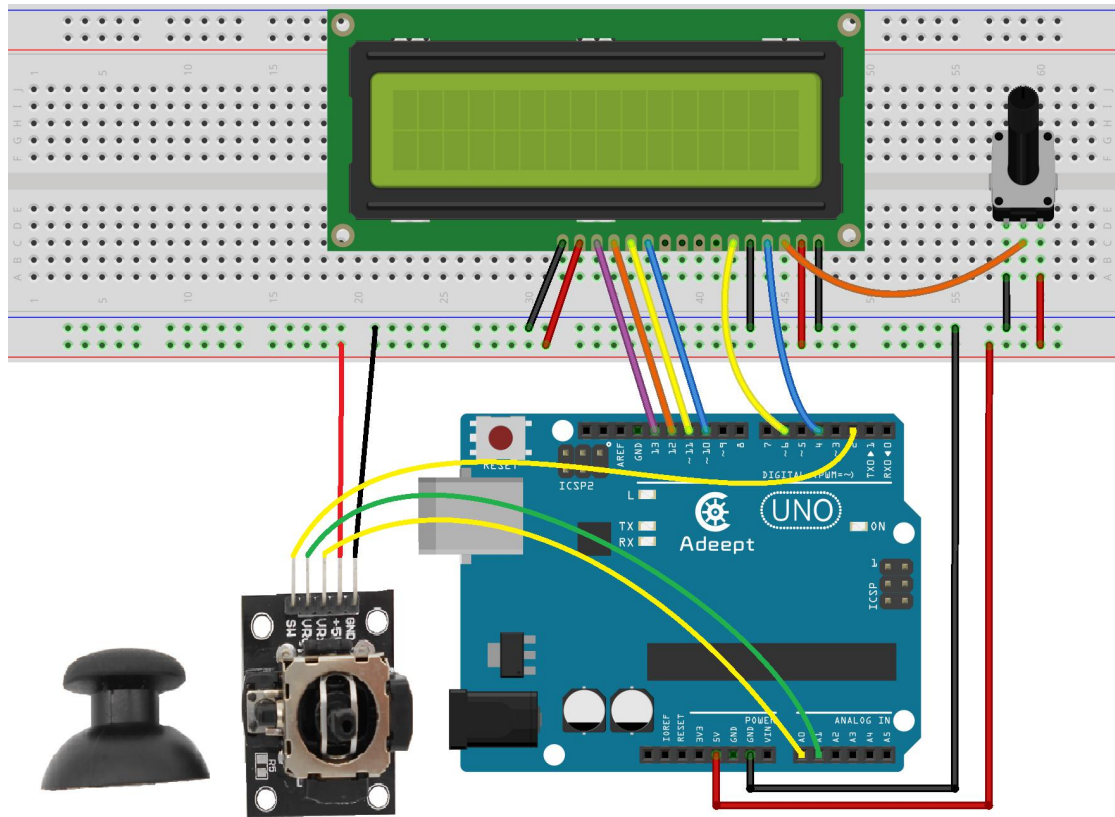
A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. A joystick, also known as the control column, is the principal control device in the cockpit of many civilian and military aircraft, either as a center stick or 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

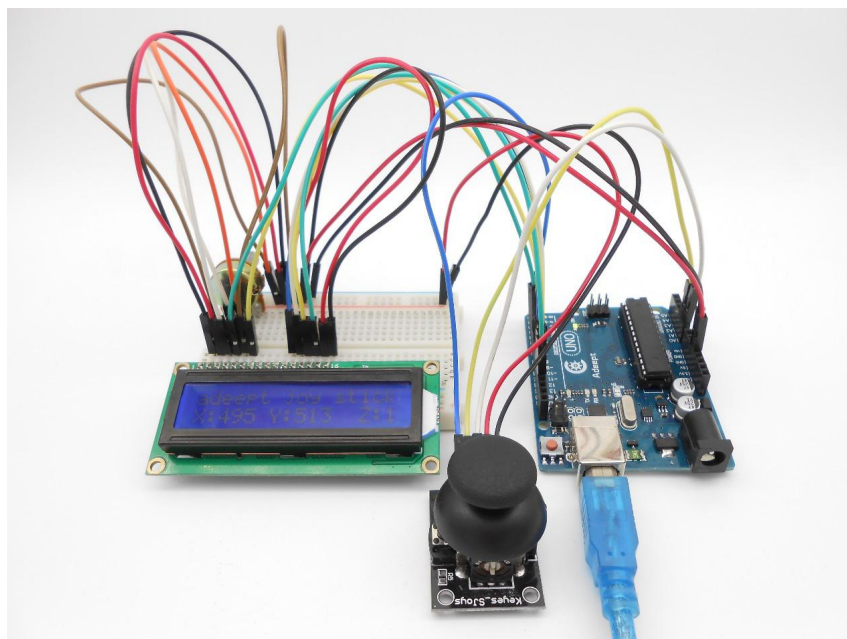
### Step 1: Build the circuit



### Step 2: Program

### Step 3: Compile the program and upload to Adept UNO board

Now, you can see the PS2 joystick state information displayed on the LCD1602.



---

## Lesson 25 A Simple Voltmeter

### Overview

In this lesson, we will make a simple voltmeter (0~5V) with Arduino UNO and LCD1602. Then, we will measure the voltage of the potentiometer (when adjusting the knob) with the simple voltmeter and display the voltage detected on the LCD1602.

### Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* LCD1602
- 2 \* Potentiometer
- 1 \* Breadboard
- Several jumper wires

### Principle

The basic principle of this experiment: Convert the analog voltage collected from Arduino to digital quantity by the ADC (analog-to-digital converter) through programming, and 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 5V to the other outer pin of the potentiometer.

By turning the shaft of the potentiometer, you change the 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 middle and the side one (connected to 5V) is close to zero (and the resistance on the other side is close to 10k Ohm), the voltage at the middle pin is close to 5V. When the resistances are reversed, the voltage at the center pin changes to about 0V, or ground. This voltage is the analog voltage that you're reading as an input.

The Arduino UNO board 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 done in the opposite direction, it's 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:*

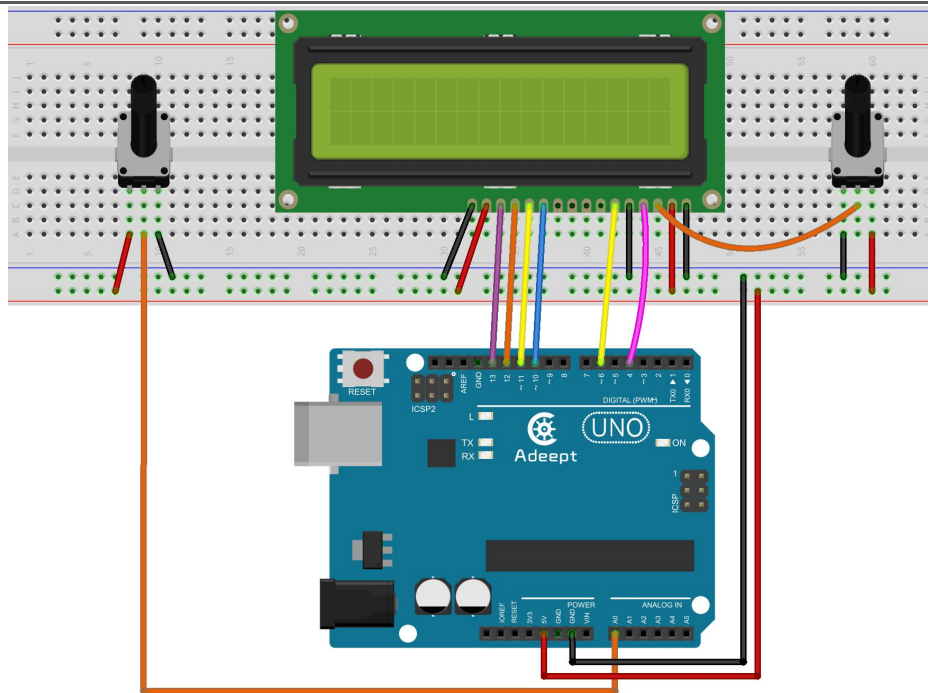
pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

### *Returns:*

int (0 to 1023)

## **Procedures**

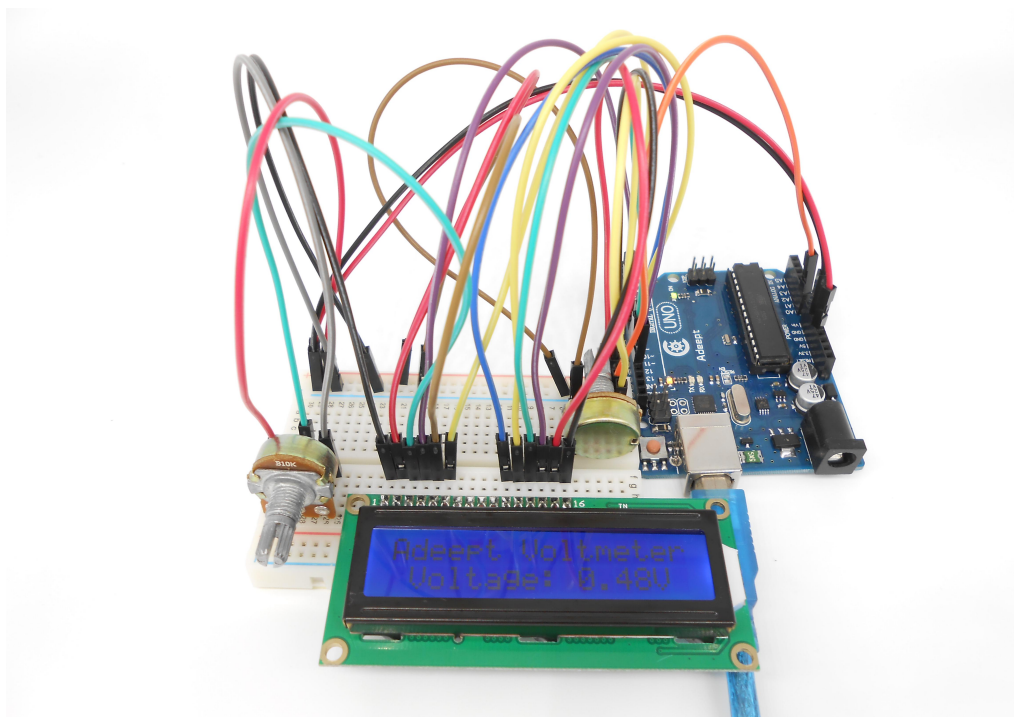
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO.

Now, turn the shaft of the potentiometer, and you will see the voltage displayed on the LCD1602 changed.



---

## Summary

The working principle of the voltmeter is reading analog voltage which is input to the ADC inside. Through this lesson, you should have mastered how to read analog value and how to make a simple voltmeter with Arduino. Try to use the voltmeter you made and feel the charm of making!

---

## Lesson 26 RFID module

### Overview

In this lesson, we will learn how to use RFID Module. We programmed the Arduino UNO to read the data which is acquired by the RFID module, and then make the ID data displayed on the LCD1602 and the serial monitor.

### Requirement

- 1\* Adept UNO R3 Board
- 1\* USB Cable
- 1\* RFID-RC522 Module
- 1\* RFID ID Round Tag
- 1\* RFID ID Card
- 1\* LCD1602
- 1\* 10K $\Omega$  Potentiometer
- 1\* NPN Transistor (8050)
- 1\* 1K $\Omega$  Resistor
- 1\* Passive buzzer
- 1\* Battery holder
- 1\* Breadboard
- Several Jumper Wires

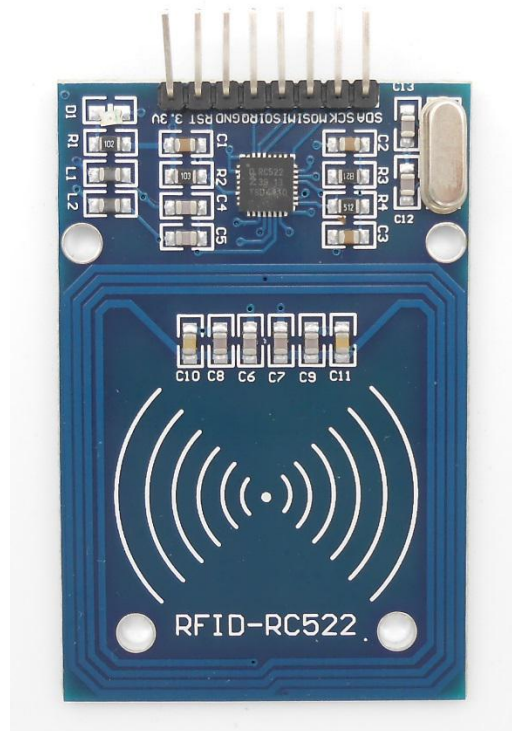
### Principle

RFID technology is used for a wide variety of applications including access control, package identification, warehouse stock control, point-of-sale scanning, retail antitheft systems, toll-road passes, surgical instrument inventory, and even for identifying individual sheets of paper placed on a desk. RFID tags are embedded in name badges, shipping labels, library books, product tags and boxes; installed in aircraft; hidden inside car keys; and implanted under the skin of animals or even people. RFID systems work on a wide range of frequencies, have a variety of modulation and encoding schemes, and vary from low-power passive devices with range of only a few millimeters to active systems that work for hundreds of kilometers.

However, all RFID systems have the same basic two-part architecture: a reader and a transponder. The reader is an active device that sends out a signal and listens for responses, and the transponder (the part generally

called the “tag”) detects the signal from a reader and automatically sends back a response containing its identity code.

A reader is shown in the following:



A transponder is shown in the following:



Different types of RFID tags fall into one of three broad categories: active,

---

passive, and battery-assisted passive.

Active tags are physically large because they require their own power supply such as a battery. They can also have a very long range because the availability of local power allows them to send high-powered responses that can travel from tens of meters to hundreds of kilometers. An active tag is essentially a combination of a radio receiver to detect the challenge, some logic to formulate a response, and a radio transmitter to send back the response. They can even have the challenge and response signals operate on totally different frequencies. The downsides are the size of the tag, a high manufacturing cost due to the number of parts required, and the reliance on a battery that will go flat eventually.

Passive tags can be much smaller and cheaper than active tags because they don't require a local power supply and have much simpler circuitry. Instead of supplying their own power, they leach all the power they need from the signal sent by the reader. Early passive tags operated on the "Wiegand effect," which uses a specially formed wire to convert received electromagnetic energy into radio-wave pulses. Some early passive RFID tags actually consisted of nothing more than a number of very carefully formed wires made from a combination of cobalt, iron, and vanadium, with no other parts at all.

Modern passive tags use a clever technique that uses current induced in their antenna coil to power the electronics required to generate the response. The response is then sent by modulating the reader's own field, and the reader detects the modulation as a tiny fluctuation in the voltage across the transmitter coil. The result is that passive tags can be incredibly small and extremely inexpensive: the antenna can be a simple piece of metal foil, and the microchips are produced in such large quantities that a complete RFID-enabled product label could cost only a few cents and be no thicker than a normal paper label. Passive tags can theoretically last indefinitely because they don't contain a battery to go flat, but their disadvantage is a very short operational range due to the requirement to leach power from the reader's signal, and lack of an actively powered transmitter to send back the response.

Passive tags typically operate over a range of a few millimeters up to a few meters.

Tags can also have a variety of different modulation schemes, including AM, PSK, and ASK, and different encoding systems. With so many incompatible

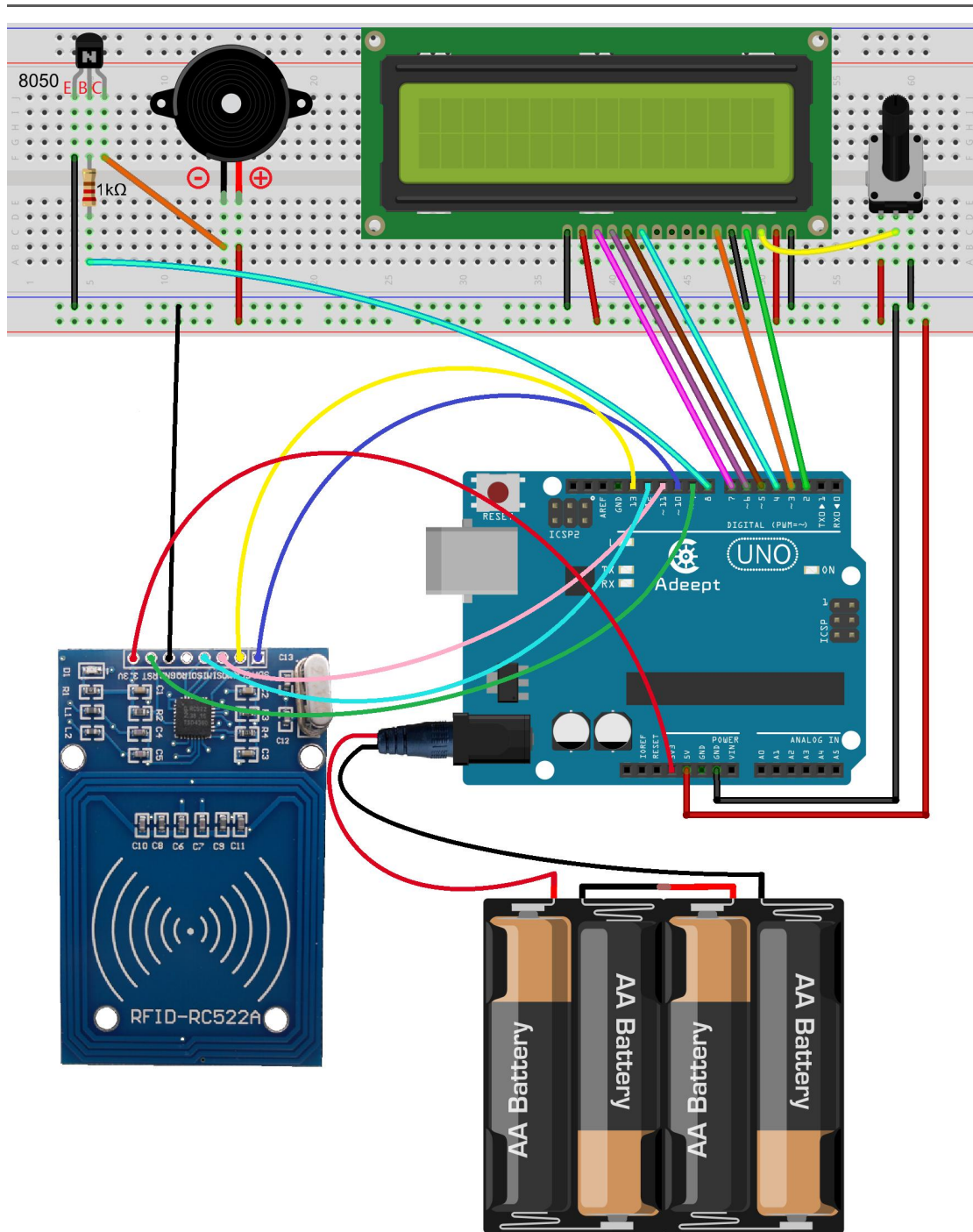
variations, it's sometimes hard to know if specific tags and readers are compatible. Generally speaking, each type of tag will only function on one specific frequency, modulation scheme, and communications protocol. Readers, on the other hand, are far more flexible and will often support a range of modulation schemes and comms protocols, but are usually still limited to just one frequency due to the tuning requirements of the coil.

Apart from the specific requirements for communicating with them, tags can also have a number of different features. The most common passive tags simply contain a hard-coded unique serial number and when interrogated by a reader they automatically respond with their ID code. Most tags are read-only so you can't change the value they return, but some types of tags are read/write and contain a tiny amount of rewritable storage so you can insert data into them using a reader and retrieve it later. However, most uses of RFID don't rely on any storage within the tag, and merely use the ID code of the tag as a reference number to look up information about it in an external database or other system.

RFID tags are produced in a wide variety of physical form factors to suit different deployment requirements. The most commonly seen form factor is a flat plastic card the same size as a credit card, often used as an access control pass to gain access to office buildings or other secure areas. The most common form by sheer number produced, even though you might not notice them, is RFID-enabled stickers that are commonly placed on boxes, packages, and products. Key fob tags are also quite common, designed to be attached to a keyring so they're always handy for operating access control systems.

## **Procedures**

### **1. Build the circuit**

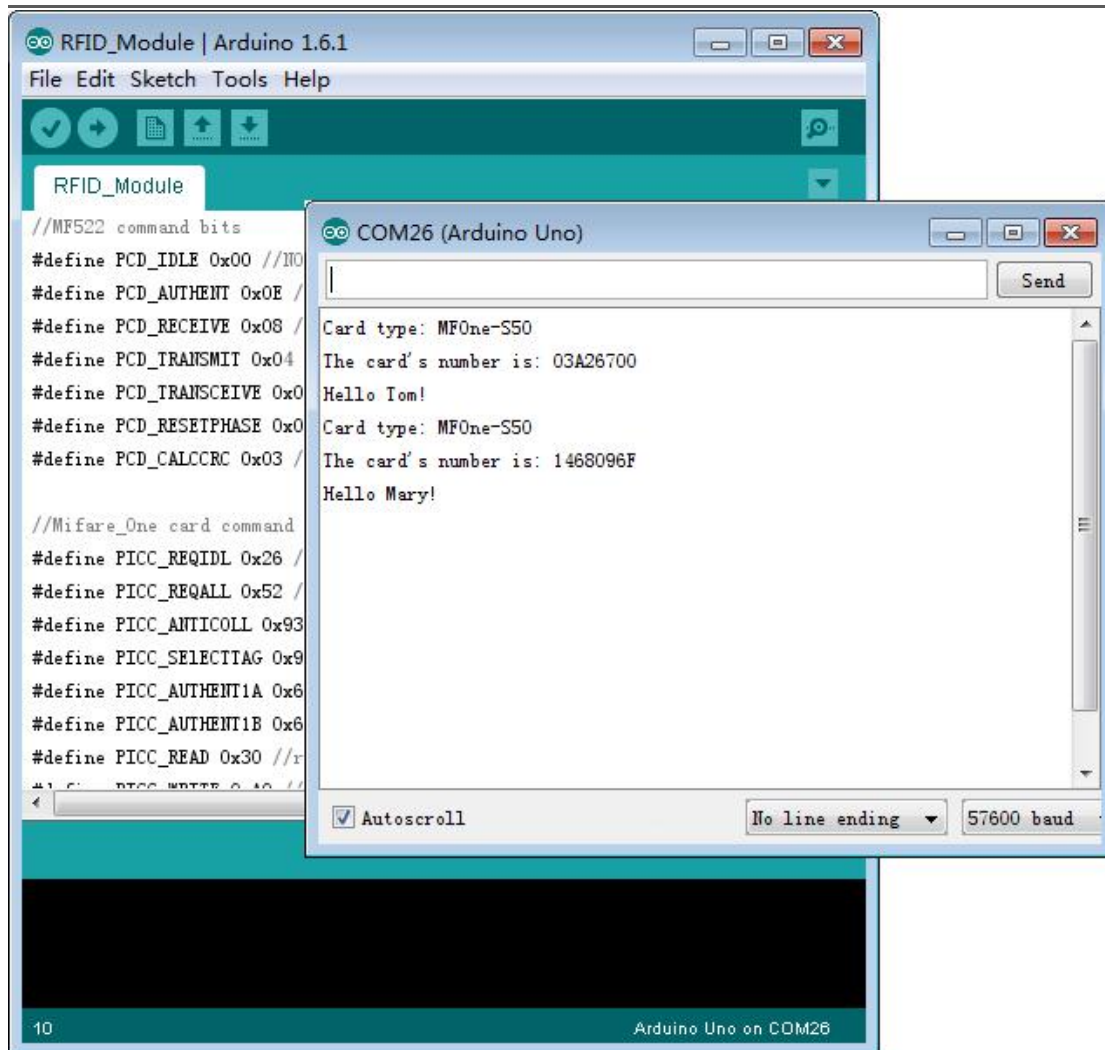


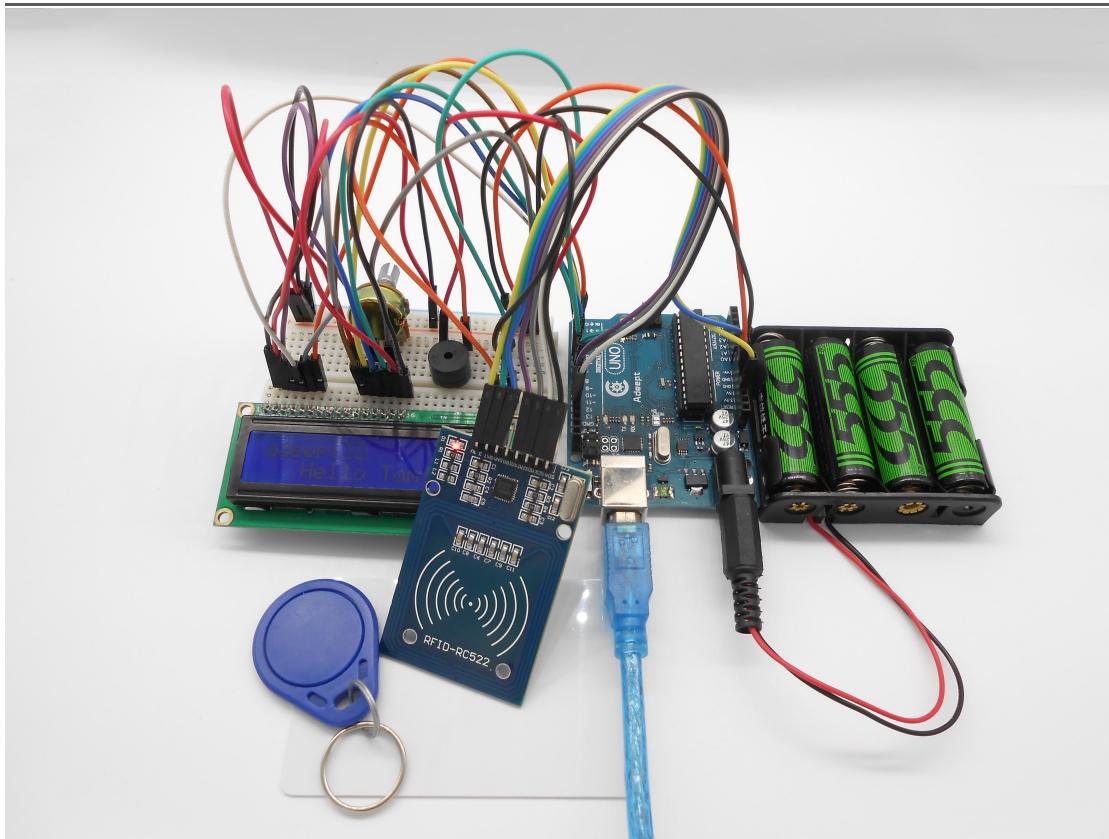
fritzing

## 2. Program

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

Now, when you close the ID card to the RFID reader, the buzzer will sound, and the ID number will be sent to the serial monitor, and it will be also displayed on the LCD1602.





## Lesson 27 Move a cat

### Overview

This is a simple interaction experiment for Arduino and Processing. We collect the distance data by programming the Arduino UNO, and send the data to the Processing via serial port, and then make a cat move according to the distance data.

### Requirement

- 1\* Adept UNO R3 Board
- 1\* USB Cable
- 1\* Ultrasonic Distance Sensor
- 1\* Breadboard
- Several Jumper Wires

### Principle

The experiment is divided into two parts. The first is used to acquire the data from ultrasonic module, another is used to process the data.

The distance data will be displayed on the screen with the form of visualization. When the distance decreases, the cat close to the robot. On the contrary, the cat move away from the robot

#### *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.

#### *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: 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: loadImage()

*Description*

Loads an image into a variable of type PImage. Four types of images ( .gif, .jpg, .tga, .png) images may be loaded. To load correctly, images must be located in the data directory of the current sketch.

*Syntax*

loadImage(filename)

loadImage(filename, extension)

*Parameters*

**filename** String: name of file to load, can be .gif, .jpg, .tga, or a handful of other image types depending on your platform

**extension** String: type of image to load, for example "png", "gif", "jpg"

*Returns*

PImage

●Name: createFont()

*Description*

Dynamically converts a font to the format used by Processing from a .ttf or .otf file inside the sketch's "data" folder or a font that's installed elsewhere on the computer. If you want to use a font installed on your computer, use the PFont.list() method to first determine the names for the fonts recognized by the computer and are compatible with this function. Not all fonts can be used and some might work with one operating system and not others. When sharing a sketch with other people or posting it on the web, you may need to include a .ttf or .otf version of your font in the data directory of the sketch

---

because other people might not have the font installed on their computer. Only fonts that can legally be distributed should be included with a sketch.

The size parameter states the font size you want to generate. The smooth parameter specifies if the font should be antialiased or not. The charset parameter is an array of chars that specifies the characters to generate.

### *Syntax*

`createFont(name, size)`

`createFont(name, size, smooth)`

`createFont(name, size, smooth, charset)`

### *Parameters*

**name** String: name of the font to load

**size** float: point size of the font

**smooth** boolean: true for an antialiased font, false for aliased

**charset** char[]: array containing characters to be generated

### *Returns*

PFont

● **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 current colorMode(). (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: `textFont()`

### *Description*

Sets the current font that will be drawn with the `text()` function. Fonts must be created for Processing with `createFont()` or loaded with `loadFont()` before they can be used. The font set through `textFont()` will be used in all subsequent calls to the `text()` function. If no size parameter is input, the font will appear at its original size (the size in which it was created with the "Create Font..." tool) until it is changed with `textSize()`.

Because fonts are usually bitmapped, you should create fonts at the sizes that will be used most commonly. Using `textFont()` without the size parameter will result in the cleanest type.

With the default and PDF renderers, it's also possible to enable the use of native fonts via the command `hint(ENABLE_NATIVE_FONTS)`. This will produce vector text in both on-screen sketches and PDF output when the vector data is available, such as when the font is still installed, or the font is created dynamically via the `createFont()` function (rather than with the "Create Font..." tool).

### *Syntax*

`textFont(which)`

`textFont(which, size)`

### *Parameters*

**which** PFont: any variable of the type PFont

**size** float: the size of the letters in units of pixels

### *Returns*

void

● Name: `text()`

### *Description*

Draws text to the screen. Displays the information specified in the first parameter on the screen in the position specified by the additional parameters. A default font will be used unless a font is set with the `textFont()` function and a default size will be used unless a font is set with `textSize()`. Change the color of the text with the `fill()` function. The text displays in relation to the `textAlign()` function, which gives the option to draw to the left, right, and center of the coordinates.

The `x2` and `y2` parameters define a rectangular area to display within and may only be used with string data. When these parameters are specified, they are interpreted based on the current `rectMode()` setting. Text that does not fit completely within the rectangle specified will not be drawn to the screen.

---

Note that Processing now lets you call `text()` without first specifying a `PFont` with `textFont()`. In that case, a generic sans-serif font will be used instead.

### Syntax

`text(c, x, y)`

`text(c, x, y, z)`

`text(str, x, y)`

`text(chars, start, stop, x, y)`

`text(str, x, y, z)`

`text(chars, start, stop, x, y, z)`

`text(str, x1, y1, x2, y2)`

`text(num, x, y)`

`text(num, x, y, z)`

### Parameters

**c** char: the alphanumeric character to be displayed

**x** float: x-coordinate of text

**y** float: y-coordinate of text

**z** float: z-coordinate of text

**chars** char[]: the alphanumeric symbols to be displayed

**start** int: array index at which to start writing characters

**stop** int: array index at which to stop writing characters

**x1** float: by default, the x-coordinate of text, see `rectMode()` for more info

**y1** float: by default, the x-coordinate of text, see `rectMode()` for more info

**x2** float: by default, the width of the text box, see `rectMode()` for more info

**y2** float: by default, the height of the text box, see `rectMode()` for more info

**num** int, or float: the numeric value to be displayed

### Returns

void

### ●Name: Serial

#### *Description*

Class for sending and receiving data using the serial communication protocol.

### ●Name: available()

#### *Description*

Returns the number of bytes available.

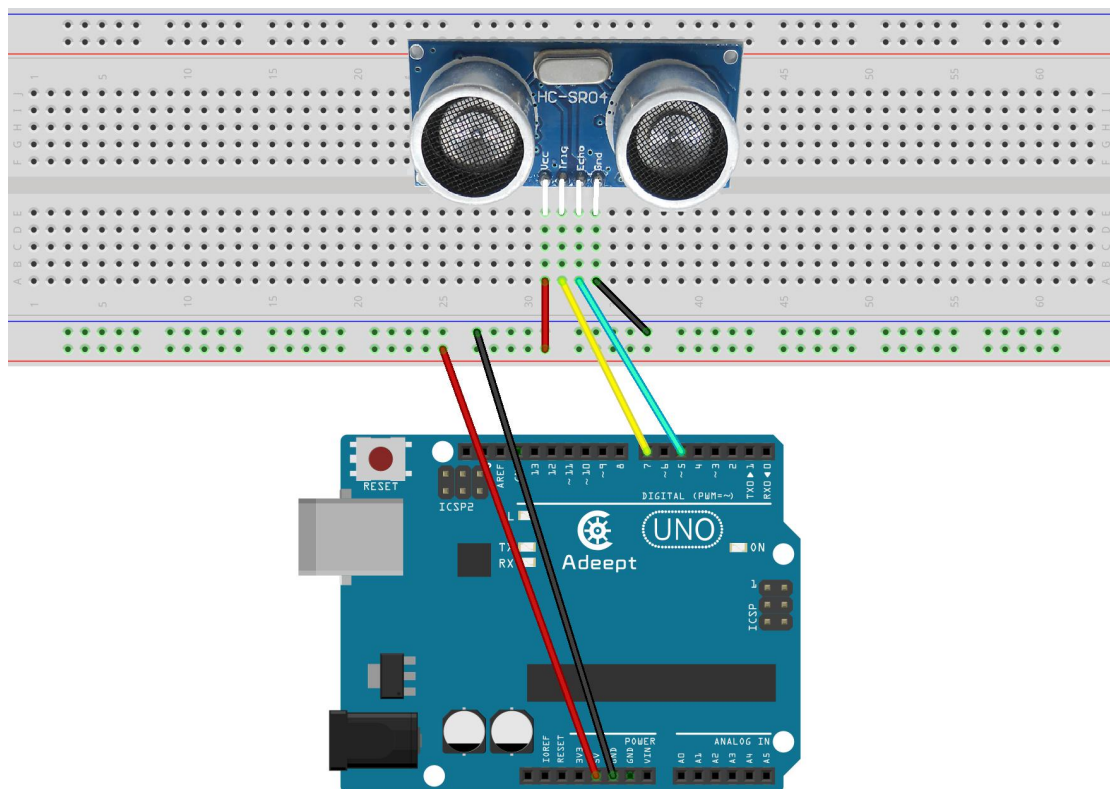
### ●Name: read()

#### *Description*

Returns a number between 0 and 255 for the next byte that's waiting in the buffer. Returns -1 if there is no byte, although this should be avoided by first checking available() to see if data is available.

## Procedures

### 1. Build the circuit

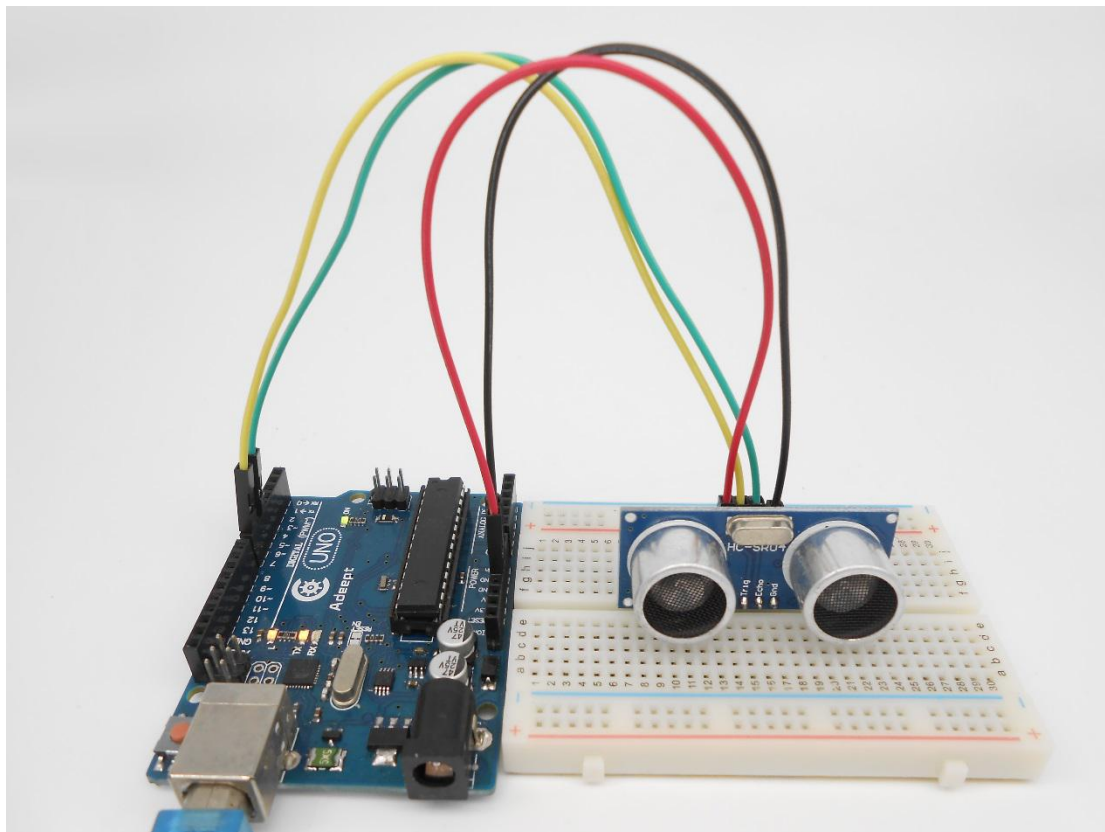
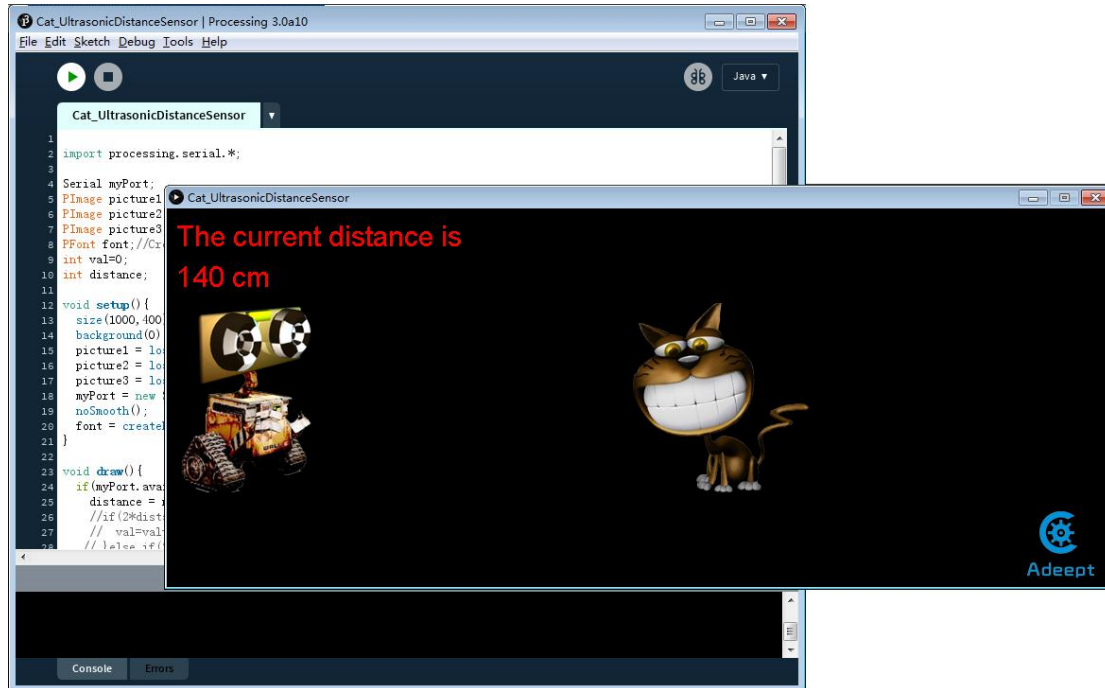


fritzing

## 2. Program

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

### 4. Run processing software (Cat\_UltrasonicDistanceSensor.pde)



## Lesson 28 Control the brightness of a photo with a photoresistor

### Overview

This is an interesting interaction experiment for the Arduino and Processing. We acquire the brightness by programming the Arduino UNO, and then change the brightness of a photo.

### Requirement

- 1\* Adept UNO R3 Board
- 1\* USB Cable
- 1\* Light Sensor (Photoresistor)
- 1\* 10K $\Omega$  Resistor
- 1\* Breadboard
- Several Jumper Wires

### Principle

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

The Arduino UNO board sends the brightness data to the Processing software via serial port, and then the Processing software changes the brightness of a image according to the data. When the photoresistor in a dark environment, the brightness of the image will be decreased. In contrast, the brightness of the image will be increased.

### *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.

### *Arduino key function:*

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

### *Description*

Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values

---

in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The `constrain()` function may be used either before or after this function, if limits to the ranges are desired.

Note that the “lower bounds” of either range may be larger or smaller than the “upper bounds” so the `map()` function may be used to reverse a range of numbers, for example

```
y = map(x, 1, 50, 50, 1);
```

The function also handles negative numbers well, so that this example

```
y = map(x, 1, 50, 50, -100);
```

is also valid and works well. The `map()` function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.

#### *Parameters*

**value:** the number to map

**fromLow:** the lower bound of the value's current range

**fromHigh:** the upper bound of the value's current range

**toLow:** the lower bound of the value's target range

**toHigh:** the upper bound of the value's target range

#### *Returns*

The mapped value.

#### *Processing key function:*

● **Name:** `tint()`

#### *Description*

Sets the fill value for displaying images. Images can be tinted to specified colors or made transparent by including an alpha value.

To apply transparency to an image without affecting its color, use white as the tint color and specify an alpha value. For instance, `tint(255, 128)` will

---

make an image 50% transparent (assuming the default alpha range of 0-255, which can be changed with `colorMode()`).

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.

The `tint()` function is also used to control the coloring of textures in 3D.

### *Syntax*

`tint(rgb)`

`tint(rgb, alpha)`

`tint(gray)`

`tint(gray, alpha)`

`tint(v1, v2, v3)`

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

### *Parameters*

**rgb** int: color value in hexadecimal notation

**alpha** float: opacity of the image

**gray** float: specifies a 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)

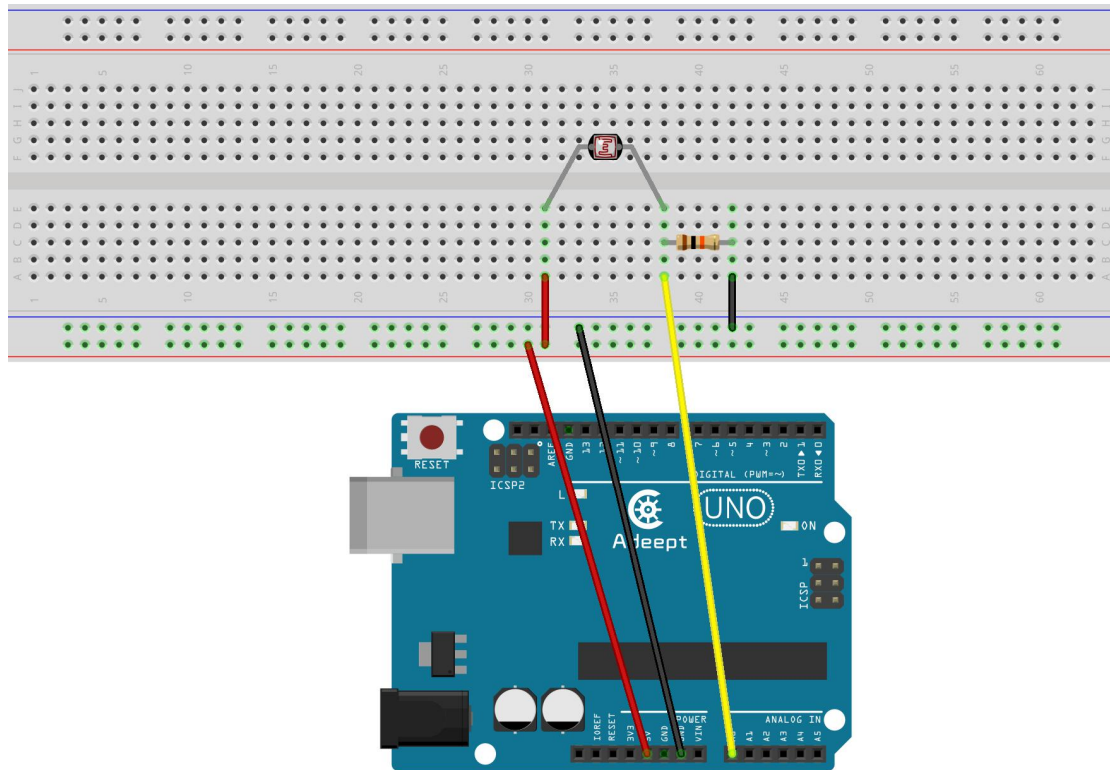
**v3** float: blue or brightness value (depending on current color mode)

### *Returns*

void

## Procedures

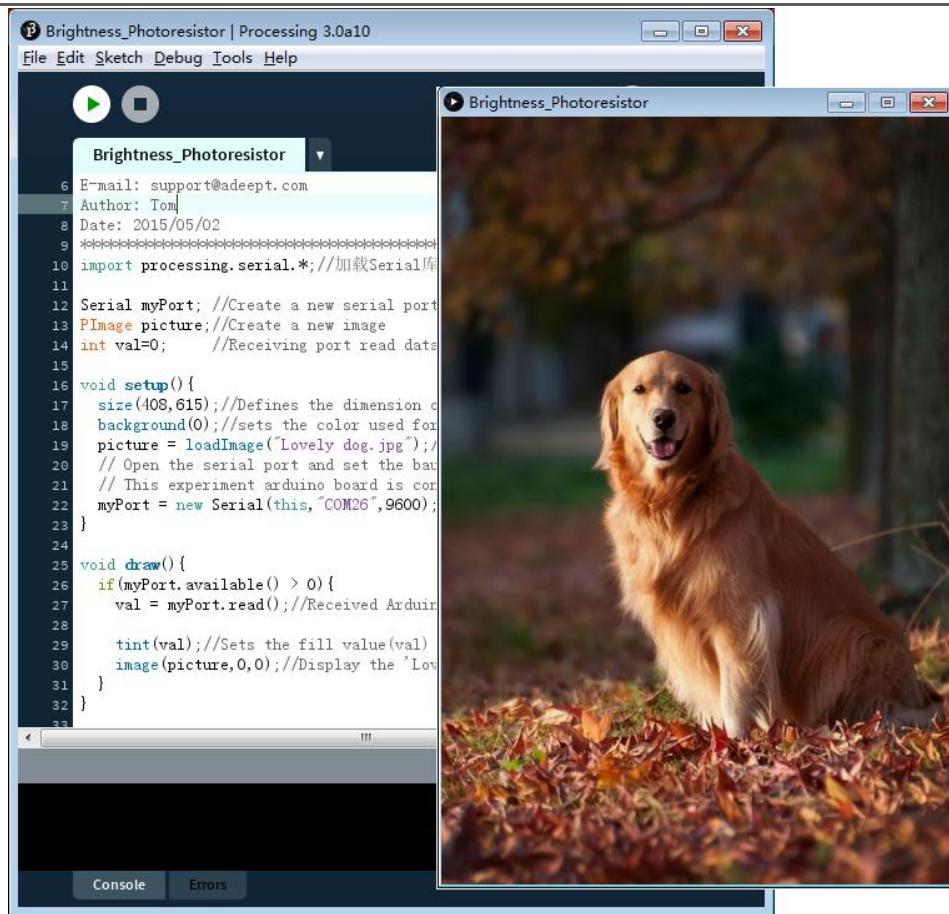
### 1. Build the circuit

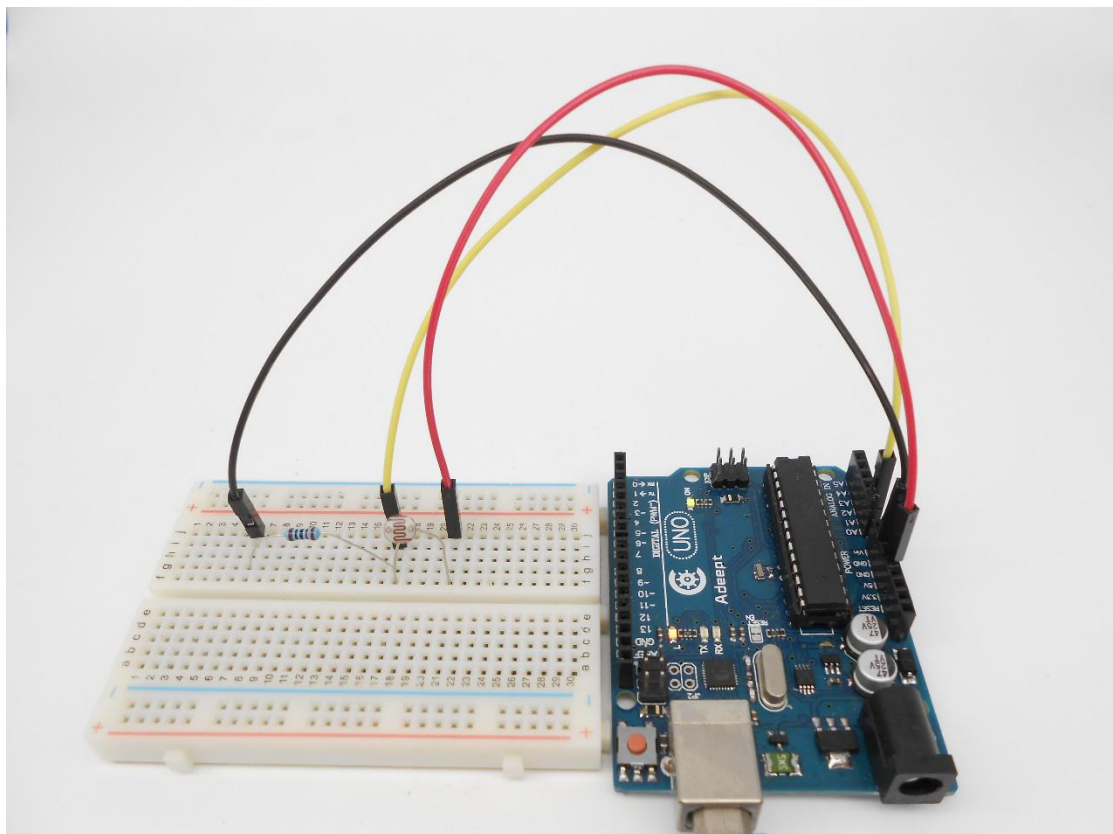
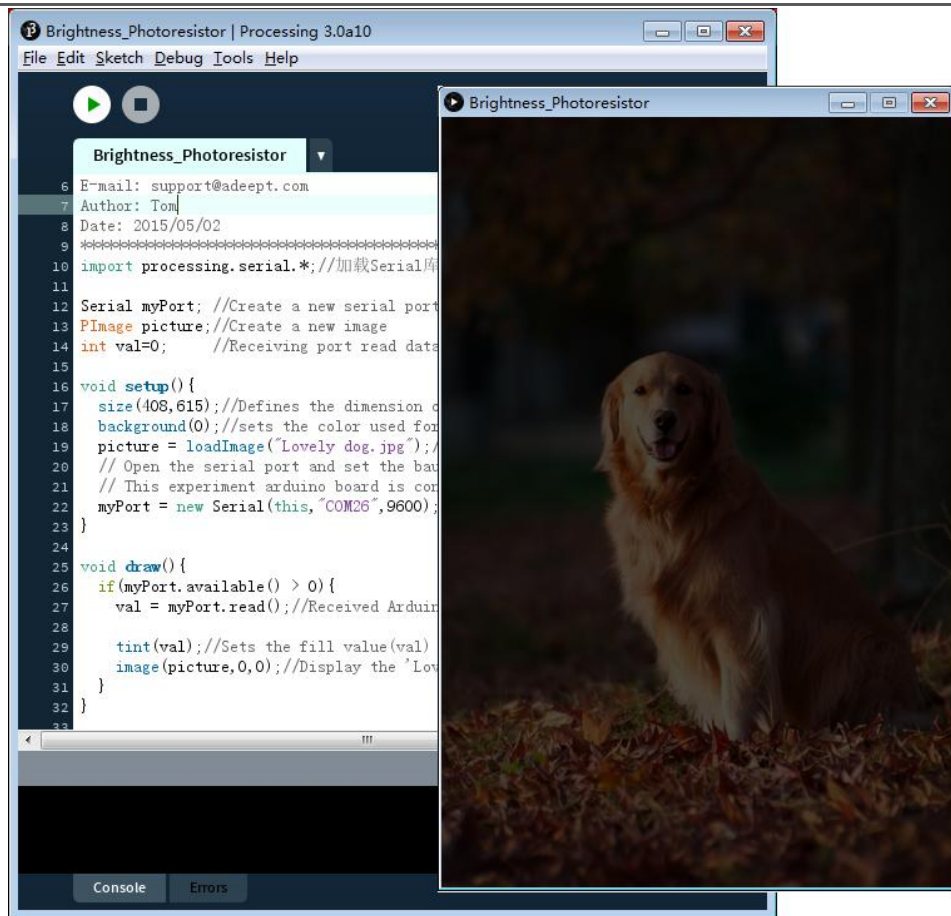


fritzing

### 2. Program

3. Compile the program and upload to Adept UNO board
4. Run processing software (Brightness\_Photoresistor.pde)





# Lesson 29 Controlling the 3D Model by PS2 Joystick

## Overview

In this lesson, we will collect the state of a joystick by programming the Arduino UNO Board, and then send the data to the Processing through the serial communication.

## Components

- 1 \* Adept UNO R3 Board
- 1 \* USB Cable
- 1 \* PS2 Joystick
- 1 \* Breadboard
- Several jumper wires

## Principle

The experiment consists of two parts: first, acquire the data from Arduino; second, process the data.

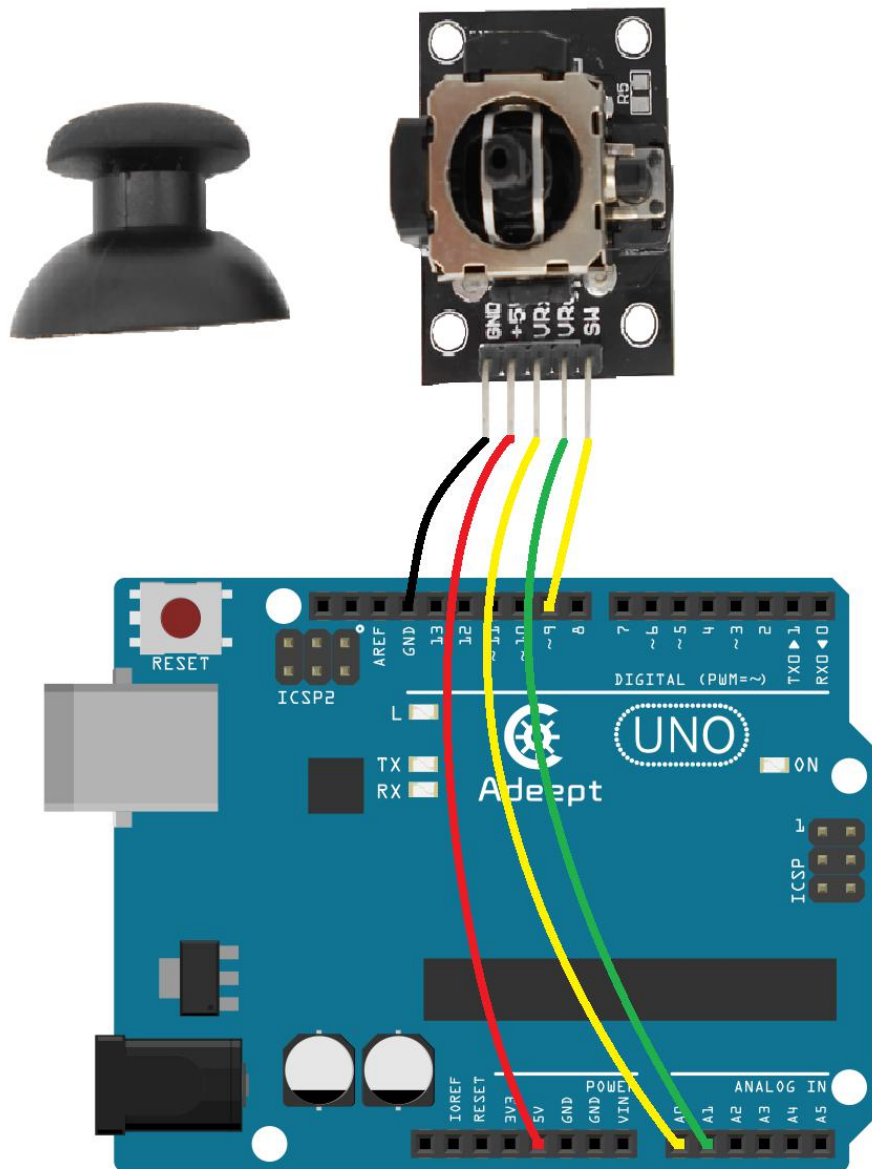
Here use the Arduino UNO board to collect data of the joystick state, and upload the data to the computer through the serial port. The data will be processed by Processing and shown with 3D image.

### *Note:*

1. In this experiment, my Arduino UNO board is connected to my computer port COM26. But it may differ in your case. So please adjust it according to your actual situation.
2. If the Processing does not run normally, you may need to install the related function libraries.

## Procedures

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Adept UNO board

Step 4: Run the Processing software (Processing\_PS2Joystick.pde)

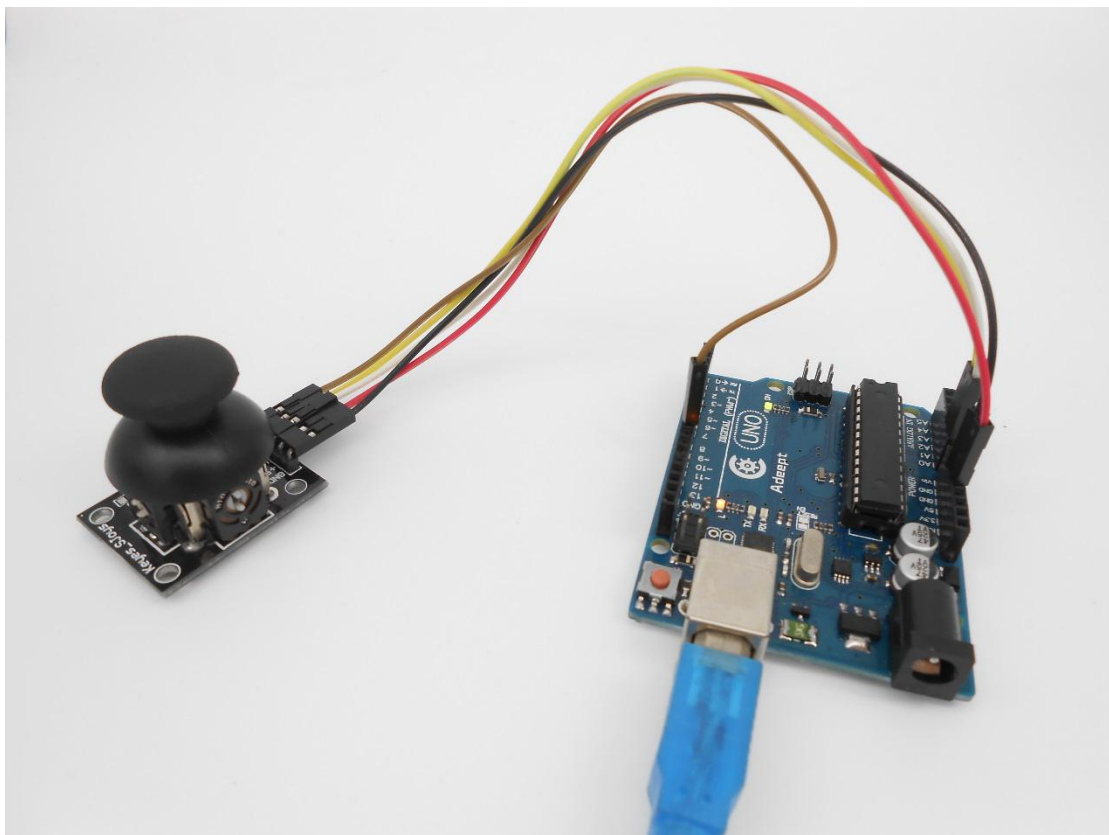
Move the joystick, and the 3D model will follow movement changes accordingly on your computer.

```

Processing_PS2Joystick | Processing 3.0a10
File Edit Sketch Debug Tools Help

Processing_PS2Joystick
33 void draw() {
34   background(0); //Set Background Color
35   image(img, 450, 450);
36   lights(); //Open lights
37   readSensors(); //Read 2-Axis value
38   fill(255, 0, 0); //Set the fill color
39   textFont(font, 30); //Set the font size
40   text("ANGLE : \n * xval: "+Rw[0]+" \n * yval: "+Rw[1]+" \n * swval: "+Rw[2], 50, 50);
41   if(Rw[2] == 1) {
42     fill(255, 0, 255); //Set the fill color
43   } else {
44     fill(0, 255, 255); //Set the fill color
45   }
46   translate(-Rw[1]/4+300, -Rw[0]/4+300, 0); //Settings Transfer Coordinates
47   box(50, 50, 250); //Draw a box 300 * 300 * 40
48 }
49
50 void readSensors() {
51   if(myPort.available() > 0) {
52     if(myPort.readBytesUntil(10, inBuffer) > 0) { //Read to determine whether the data is complete
53       String inputString = new String(inBuffer);
54       String inputStringArr[] = split(inputString, ','); //Data ', ' Split
55       Rw[0] = int(inputStringArr[0]); //Read the X value
56       Rw[1] = int(inputStringArr[1]); //Read the y value
57       Rw[2] = int(inputStringArr[2]);
58       Rw[0] = 515 - Rw[0]; //Rocker midpoint value 515 into 0
59       Rw[1] = Rw[1] - 515; //Converted to negative (rocker line out)
60     }
61   }
62 }

```



## Lesson 30 The Brick Games

### Overview

In this lesson, we will play the Brick Game with two buttons which is connected to the Arduino UNO board.

### Requirement

- 1\* Adept UNO R3 Board
- 1\* USB Cable
- 2\* Button
- 1\* Breadboard
- Several Jumper Wires

### Principle

The experiment is divided into two parts, the first part is used to acquire the data, another part used to process the data.

The Brick Game play:

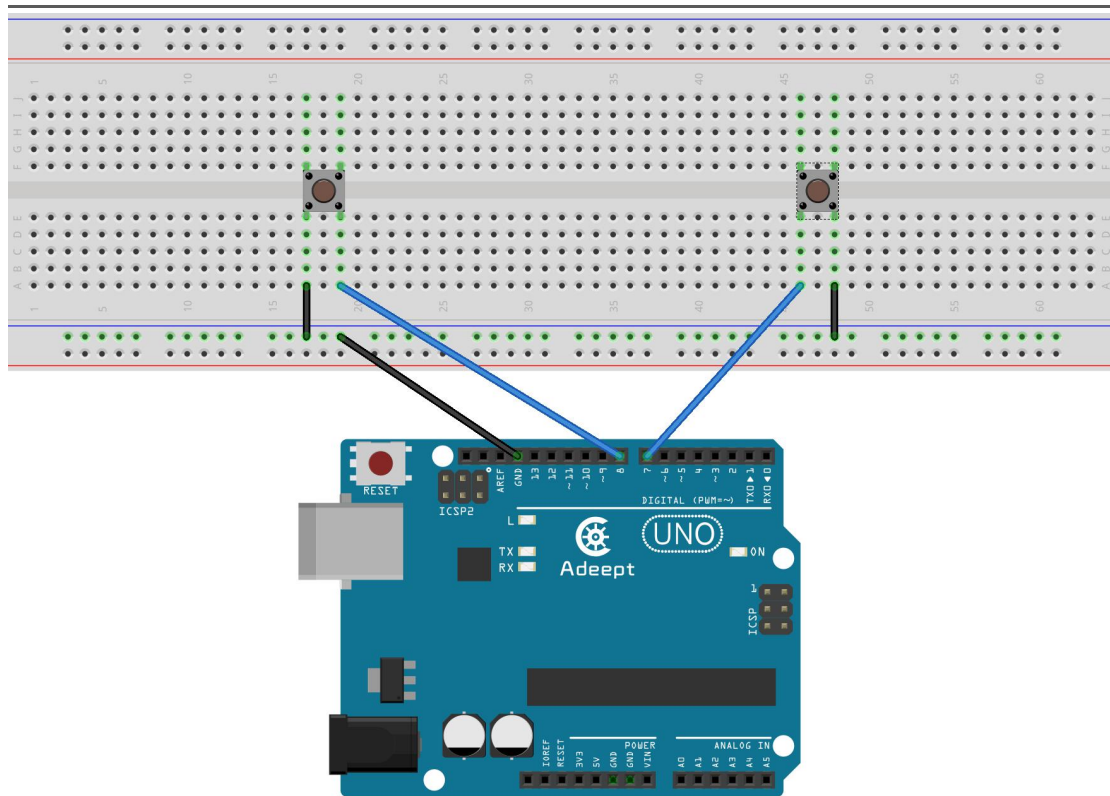
- ① When you click on a different icon ('Go', 'Play', 'back', 'Levels', 'No.1', 'No.2', 'No.3', 'Help' or 'Exit') with the mouse, you will enter the game's different interface.
- ② When you press the button on the right, the baffle moves to the right
- ③ When you press the button on the left, the baffle moves to the left.

### **Note:**

1. You need to install the Sound library, Minim library and Video 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.

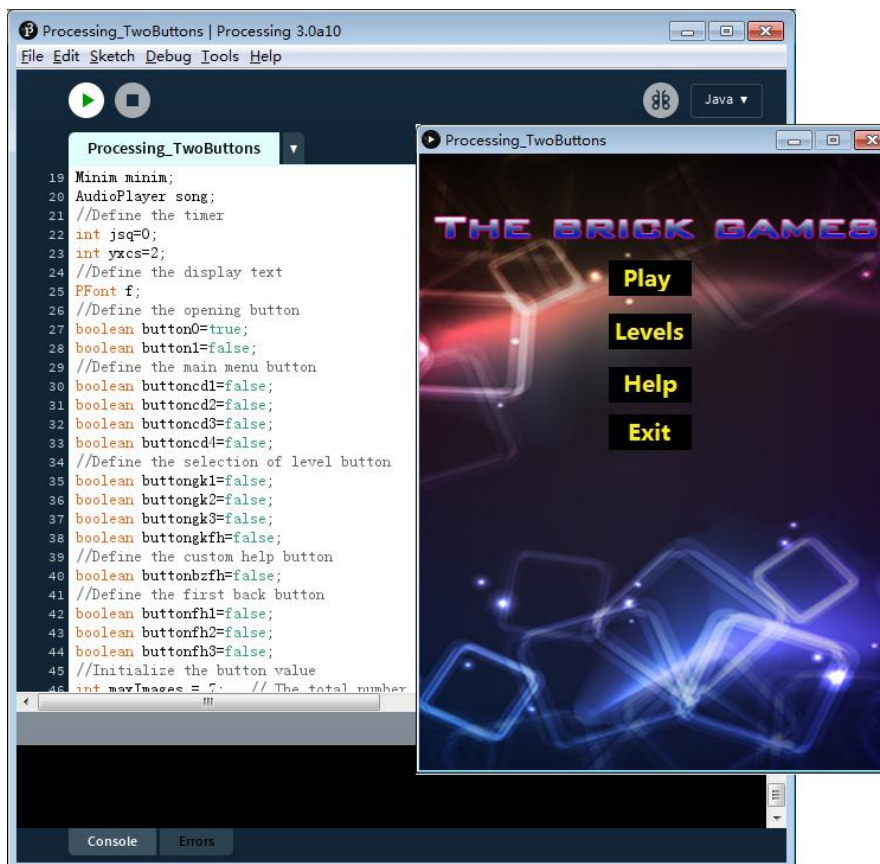
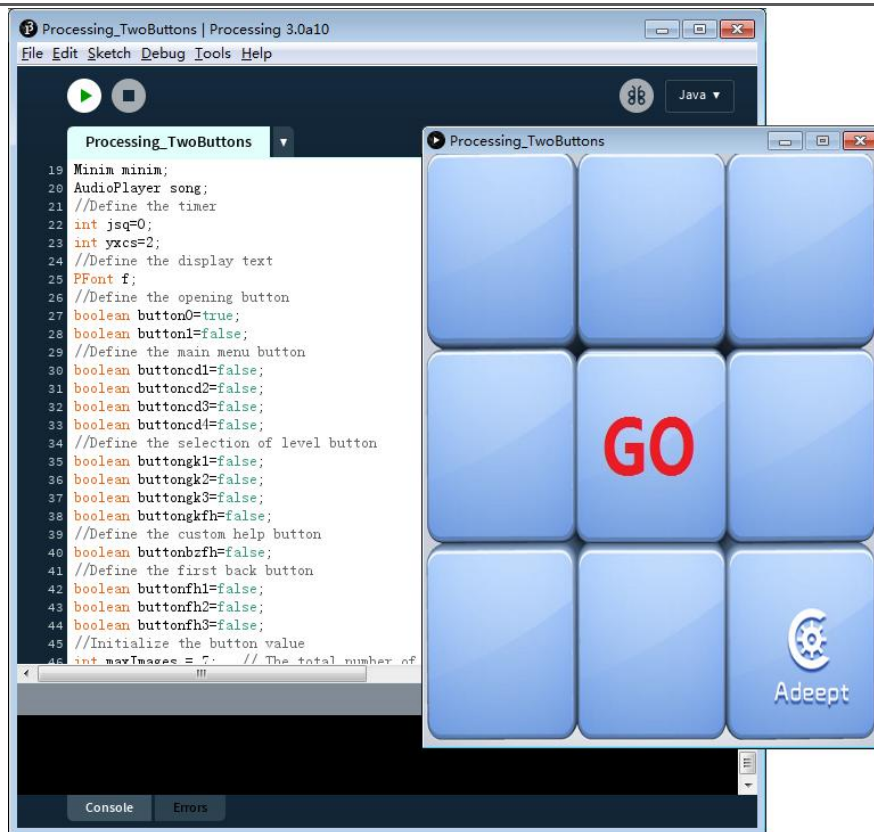
### Procedures

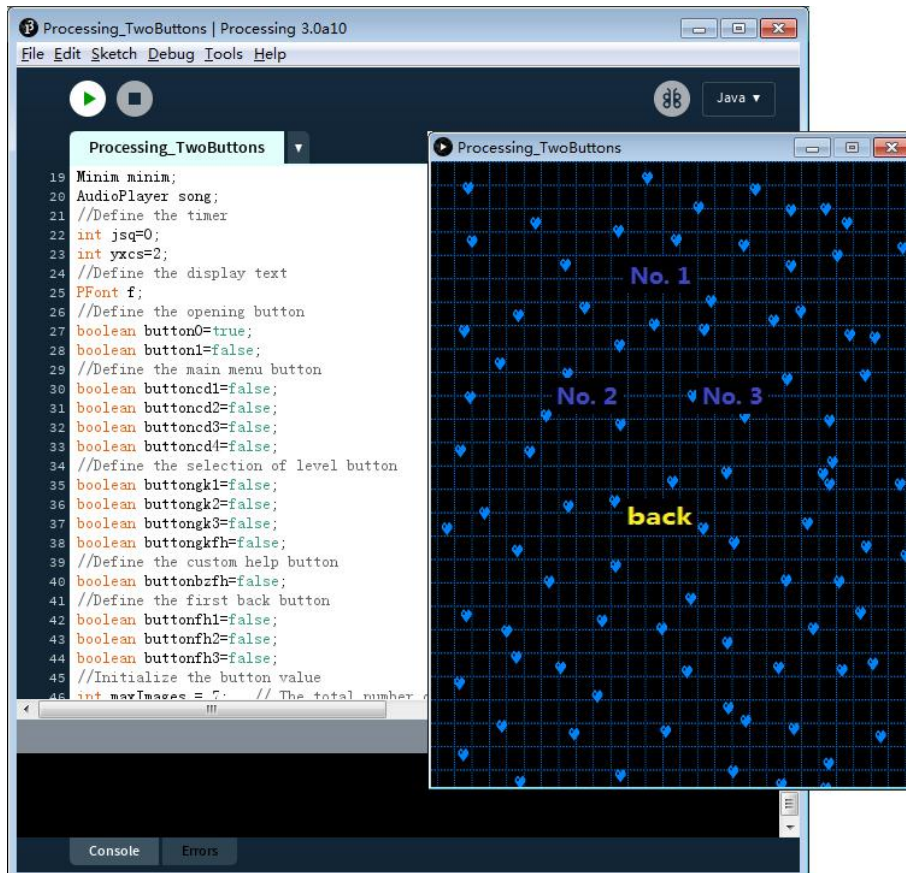
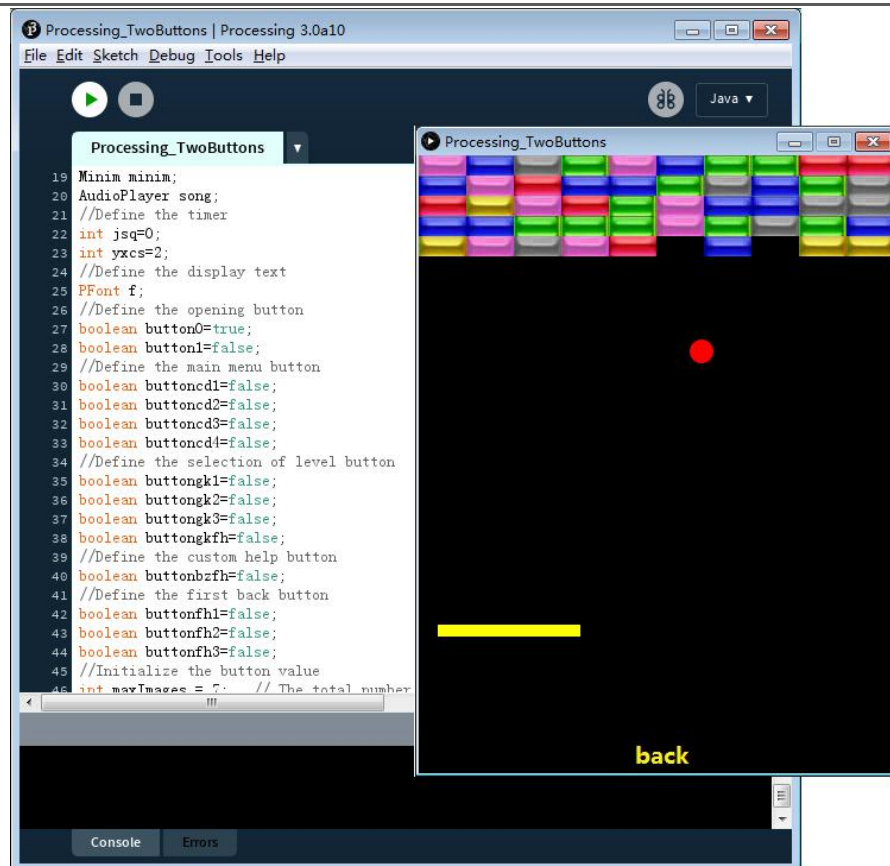
#### 1. Build the circuit

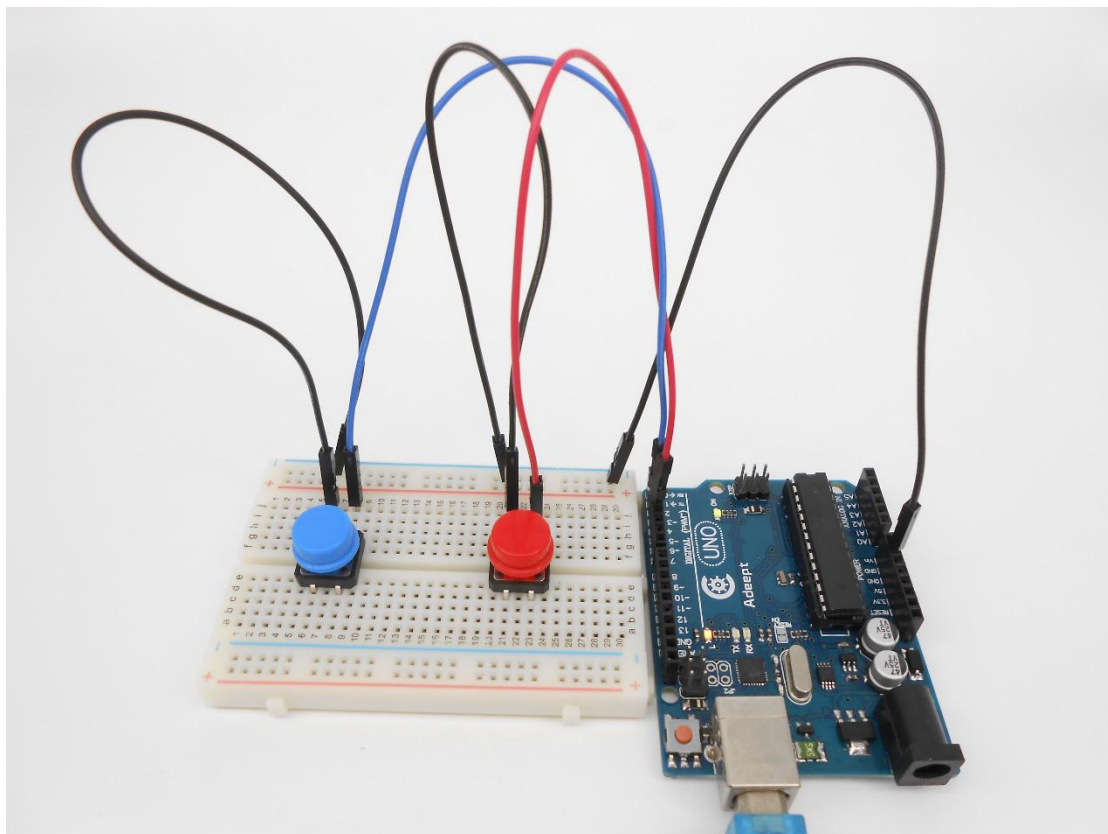
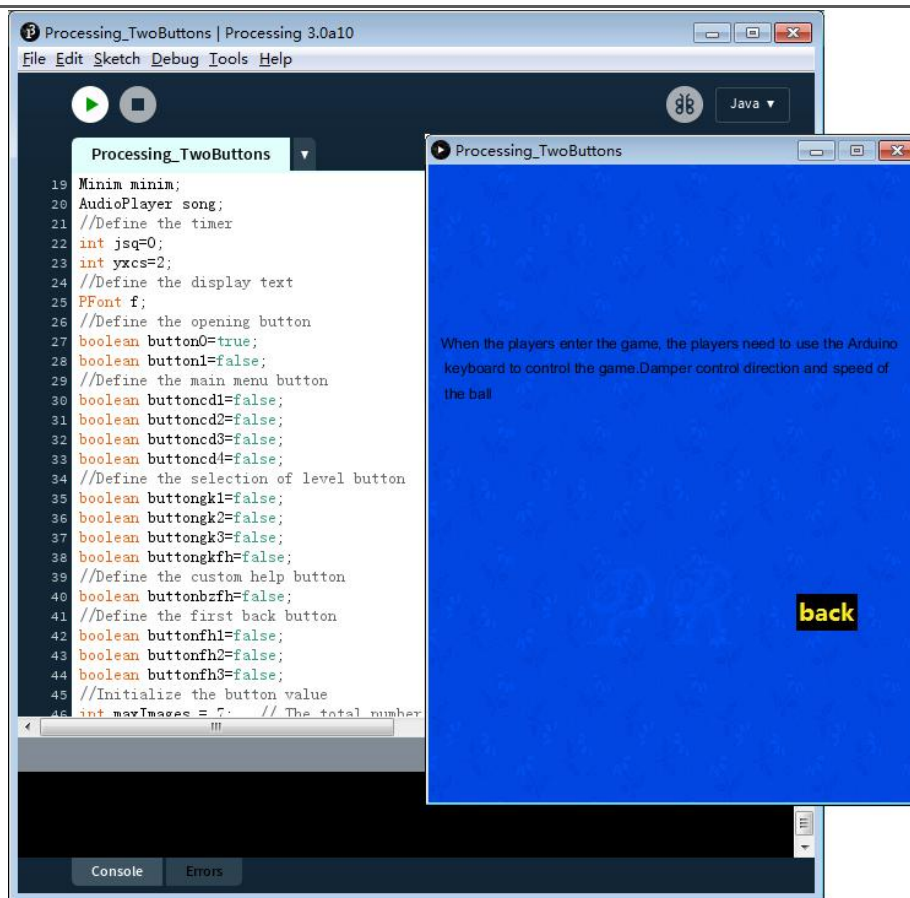


fritzing

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









Adept

**Sharing Perfects Innovation**

E-mail: [support@adeept.com](mailto:support@adeept.com)  
website: [www.adeept.com](http://www.adeept.com)