



Adept

Ultrasonic Distance Sensor 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.








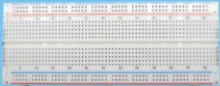

If you have any problems for learning, please contact us at support@adeept.com. We will do our best to help you solve the problem.





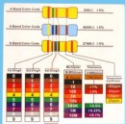
Component List

NO. 0	Name	Picture	Qty
1	Adept UNO Board(Arduino UNO)		1
2	Ultrasonic Distance Sensor		1
3	Ultrasonic Distance Sensor Holder		1
4	Servo		1
5	DC Motor		1
6	L9110 Motor Driver		1
7	Relay		1
8	LCD1602		1
9	7-Segment Display		1

NO. 0	Name	Picture	Qty
10	Active Buzzer		1
11	Analog Temperature Sensor (Thermistor)		2
12	Light Sensor (Photoresistor)		2
13	Tilt Switch		1
14	Switch		2
15	RGB LED		1
16	Red LED		10
17	Green LED		4
18	Yellow LED		4

NO. 0	Name	Picture	Qty
19	Blue LED		4
20	Resistor(220Ω)		16
21	Resistor(1 kΩ)		10
22	Resistor(10 kΩ)		5
23	Capacitor(104)		5
24	Capacitor(10uF)		2
25	Button(large)		4
26	Button(small)		5
27	Button Cap(red)		1

NO. 0	Name	Picture	Qty
28	Button Cap(white)		1
29	Button Cap(blue)		2
30	NPN Transistor(8050)		2
31	PNP Transistor(8550)		2
32	Potentiometer (10K Ω)		1
33	1N4148 Diode		2
34	1N4001 Diode		2
35	Breadboard		1
36	USB Cable		1

NO. 0	Name	Picture	Qty
37	Battery Holder		1
38	Male to Male Jumper Wires		40
39	Male to Female Jumper Wires		20
40	Header(40pin)		1
41	Band Resistor Card		1

Content

About Arduino.....	- 1 -
About Processing	- 2 -
Lesson 1 Blinking LED.....	- 3 -
Lesson 2 Controlling an LED with a button.....	- 7 -
Lesson 3 LED Flowing Lights	- 12 -
Lesson 4 Tilt Switch	- 15 -
Lesson 5 Breathing LED	- 18 -
Lesson 6 Active Buzzer.....	- 22 -
Lesson 7 Controlling Relay	- 26 -
Lesson 8 Controlling a RGB LED by PWM.....	- 29 -
Lesson 9 7-segment display	- 32 -
Lesson 10 Serial Port.....	- 36 -
Lesson 11 LCD1602.....	- 41 -
Lesson 12 Photoresistor	- 45 -
Lesson 13 Using a thermistor to measure the temperature.....	- 48 -
Lesson 14 DC motor.....	- 51 -
Lesson 15 Controlling Servo motor	- 56 -
Lesson 16 ultrasonic distance sensor	- 59 -
Lesson 17 Control a servo with ultrasonic distance sensor.....	- 62 -
Lesson 18 Move a cat.....	- 64 -

Lesson 19 Control the brightness of a photo with a photoresistor - 74 -

Lesson 20 The Brick Games..... - 80 -

Adept

About Arduino

What is Arduino?

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

ARDUINO BOARD

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

ARDUINO SOFTWARE

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

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

You need the following URL to download Arduino IDE:

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

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

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

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

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

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

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

About Processing

What is Processing?

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

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

PROCESSING SOFTWARE

Download Processing:

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

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

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

Lesson 1 Blinking LED

Overview

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

Requirement

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

Principle

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

1. What is the LED?

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

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

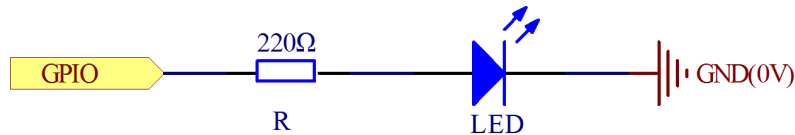
2. What is the resistor?

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

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

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

①



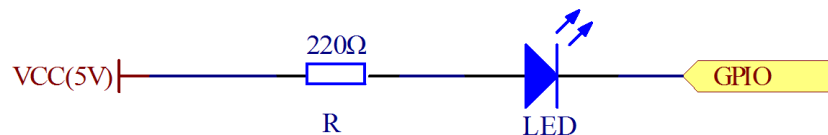
As shown in the schematic diagram above, the anode of LED is connected to Arduino's GPIO via a resistor, and the cathode of LED is connected to the ground(GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

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

$$R = U / I = 5V / (5\sim20mA) = 250\Omega\sim1K\Omega$$

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

②



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

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

3. Key functions:

● setup()

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

● loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops

consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

●pinMode()

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

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

●digitalWrite()

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

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

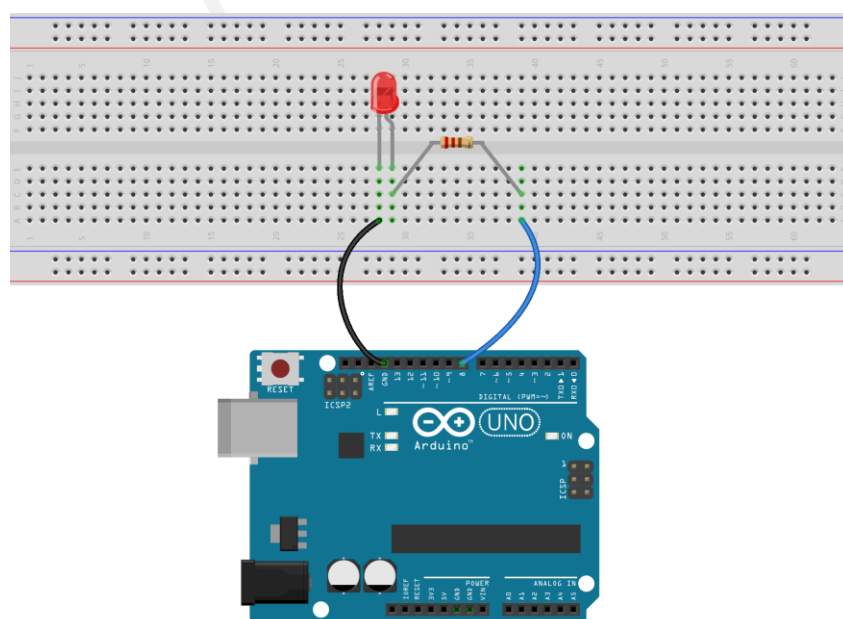
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

●delay()

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

Procedures

1. Build the circuit



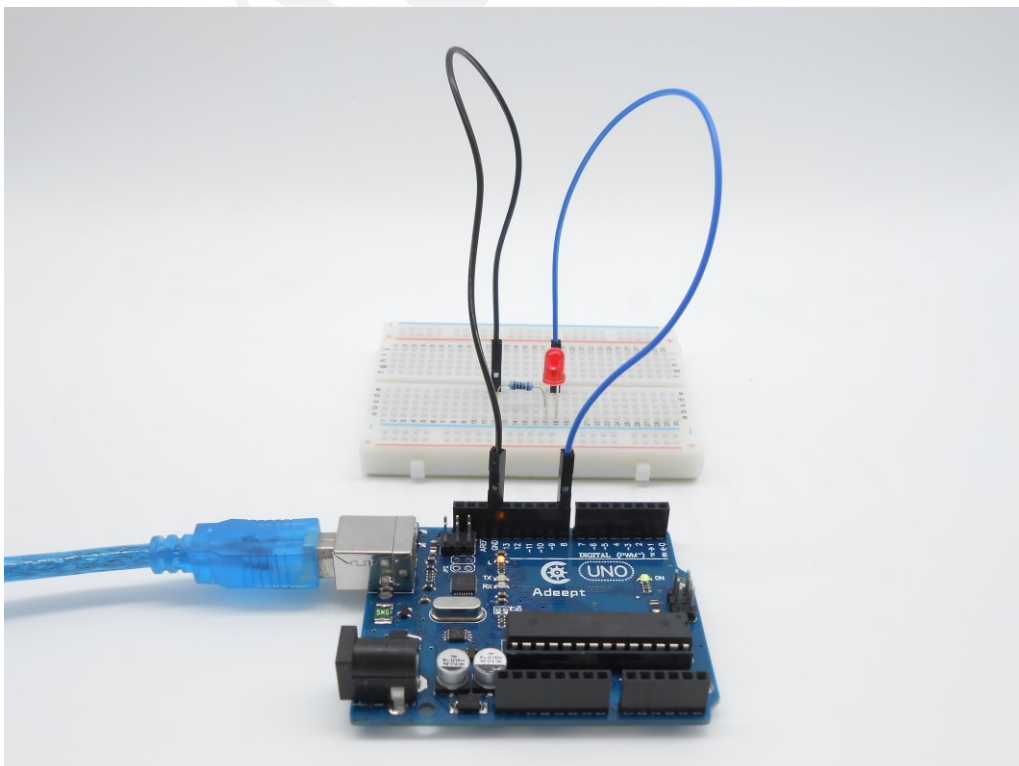
fritzing

2. Program

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

3. Compile the program and upload to Arduino UNO board

Now, you can see the LED is blinking.



Lesson 2 Controlling an LED with a button

Overview

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

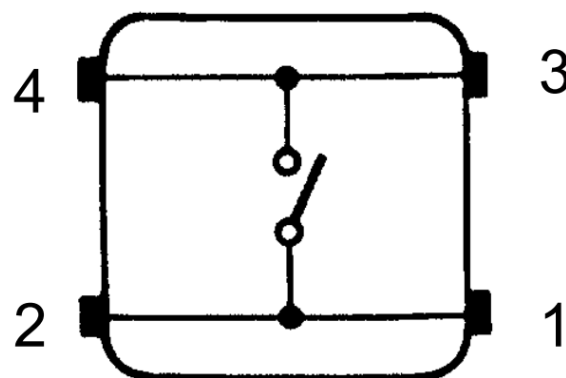
Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* Button
- 1* LED
- 1* 10K Ω Resistor
- 1* 220 Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

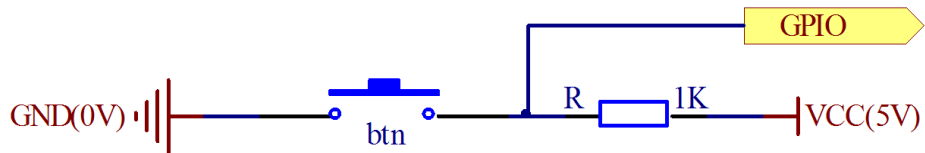
1. Button

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

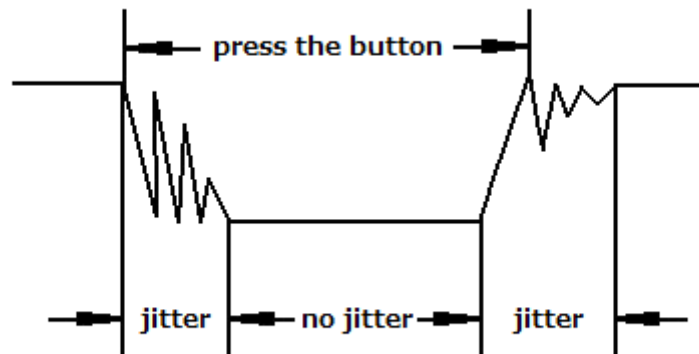


The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

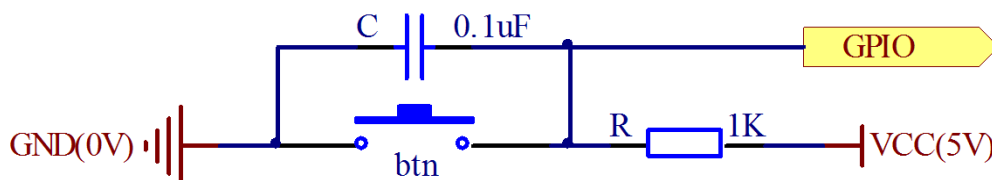
The schematic diagram we used is as follows:



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



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



2. interrupt

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

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

3. Key functions:

● `attachInterrupt(interrupt, ISR, mode)`

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

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

Syntax

`attachInterrupt(pin, ISR, mode)`

Parameters

pin: the pin number

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

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

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

● `digitalRead()`

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

Syntax

`digitalRead(pin)`

Parameters

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

Returns

HIGH or LOW

● `delayMicroseconds(us)`

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

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

Syntax

`delayMicroseconds(us)`

Parameters

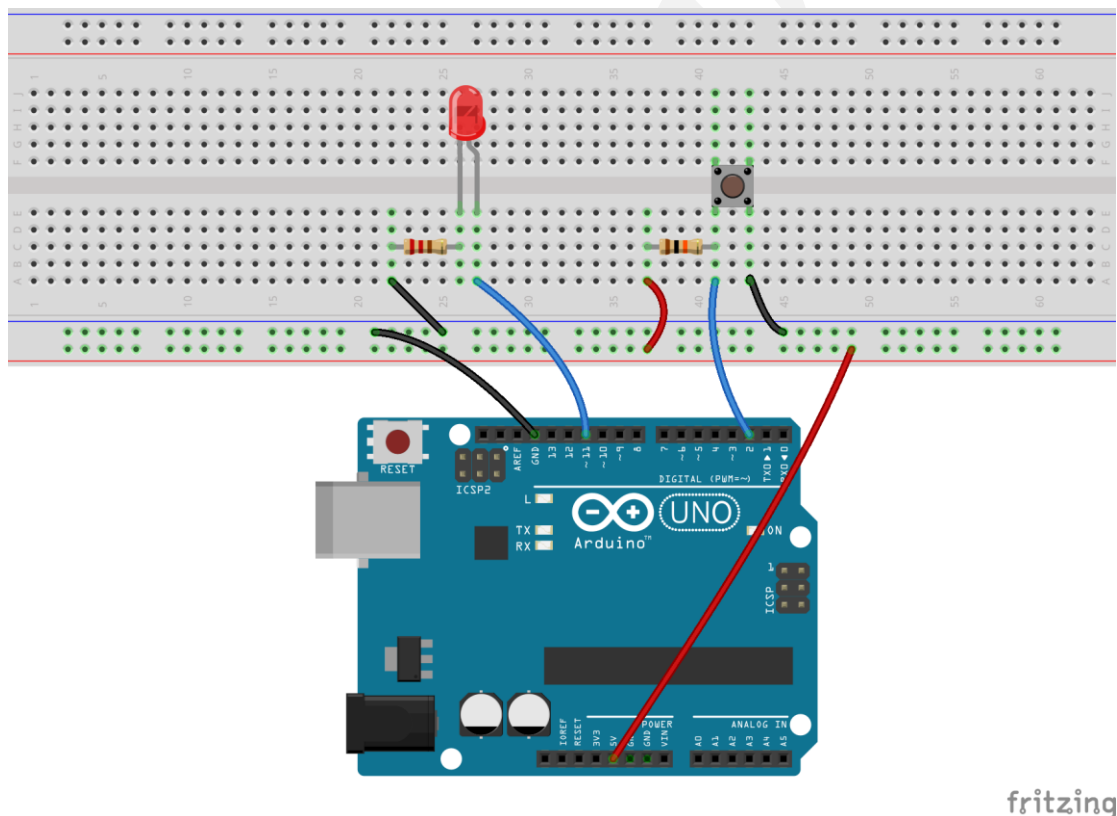
us: the number of microseconds to pause (unsigned int)

Returns

None

Procedures

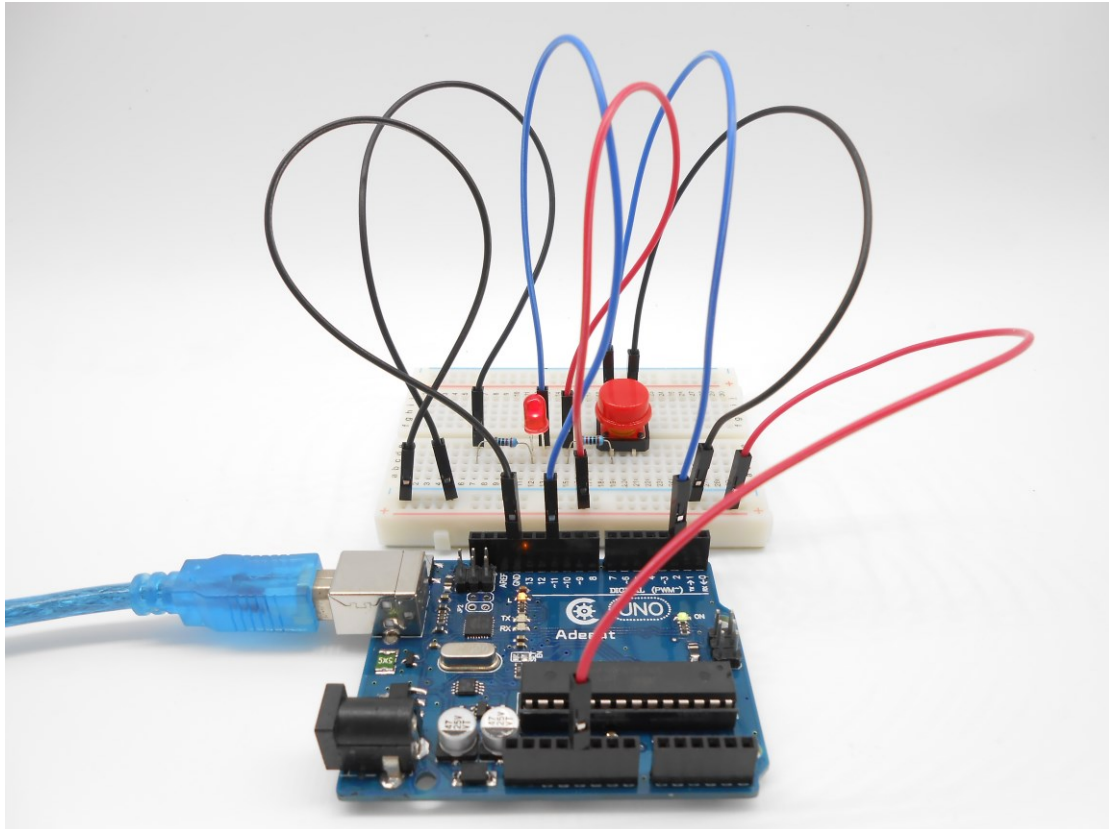
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 3 LED Flowing Lights

Overview

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

Requirement

- 1* Arduino UNO
- 1* USB cable
- 8* LED
- 8* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

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

Key function:

●for statements

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

There are three parts to the for loop header:

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

parenthesis

declare variable (optional)

initialize

test

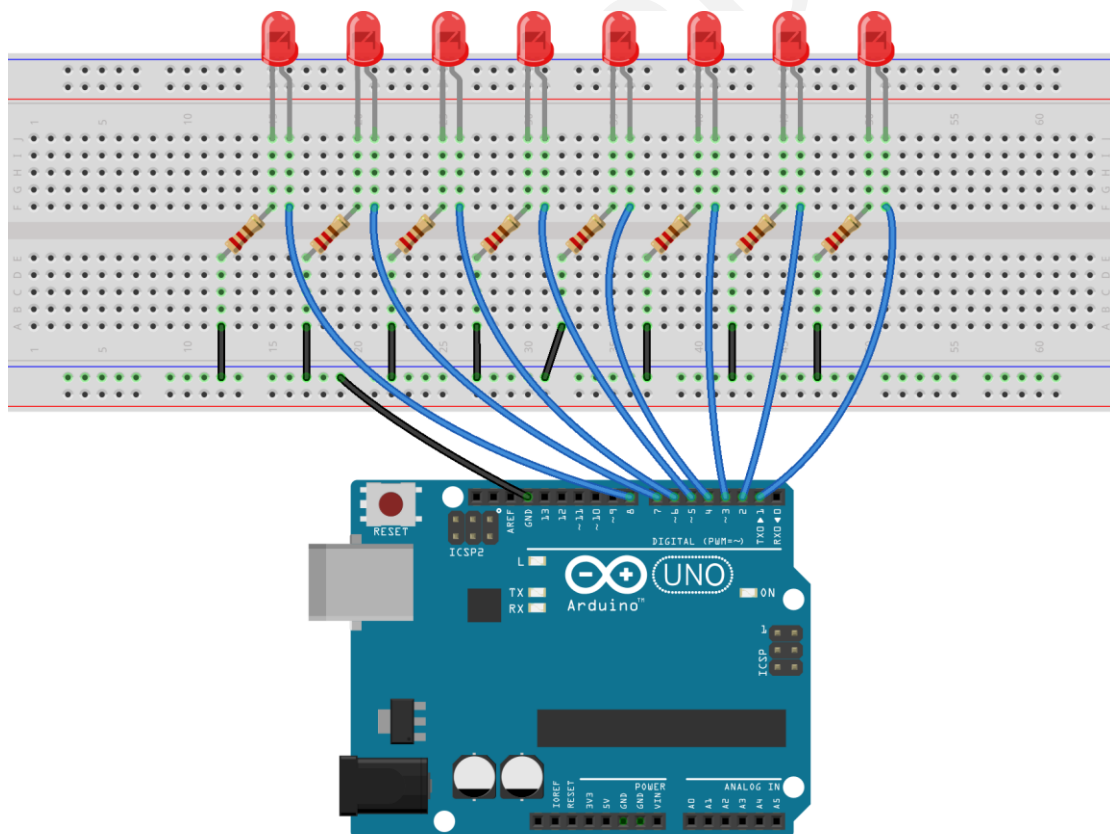
increment or decrement

```
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

1. Build the circuit



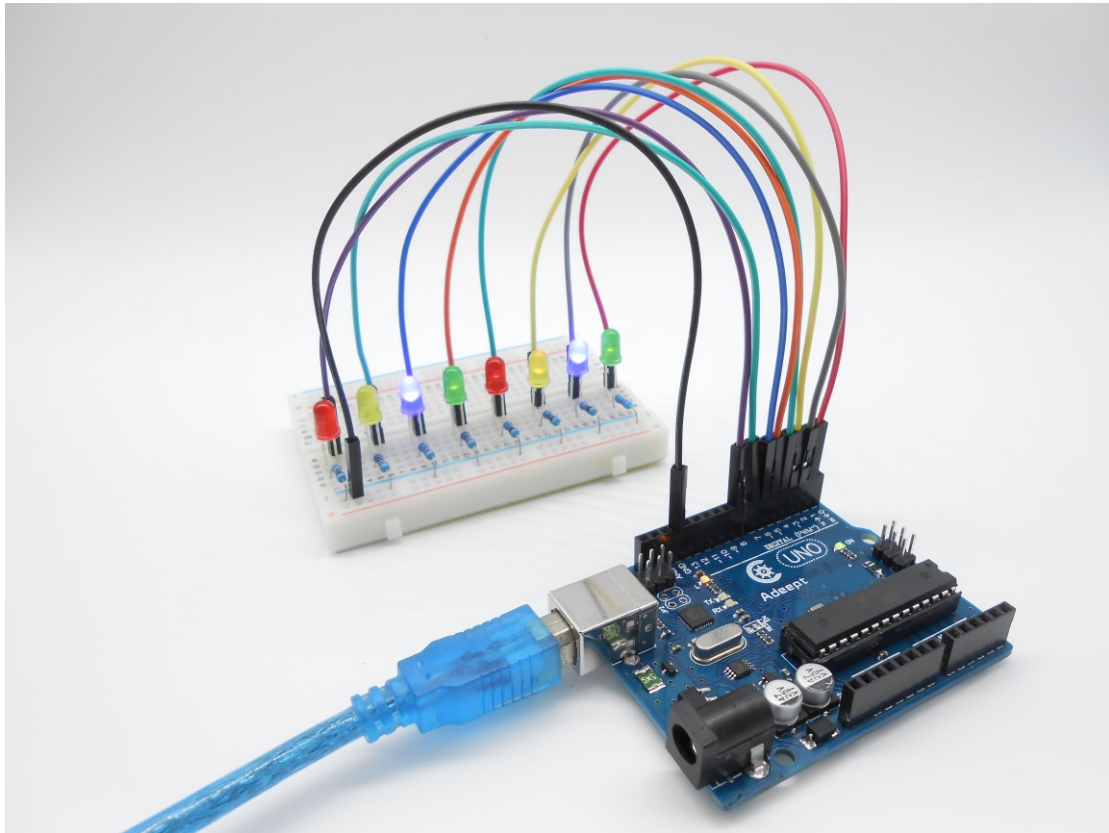
fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

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

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



Summary

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

Lesson 4 Tilt Switch

Overview

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

Requirement

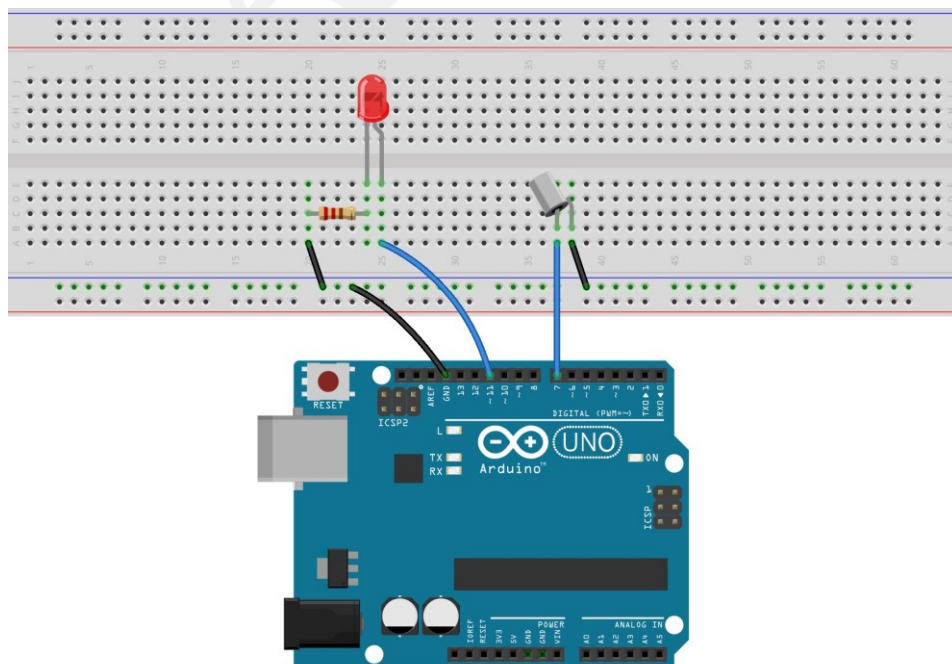
- 1* Arduino UNO
- 1* USB Cable
- 1* Tilt Switch
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper Wires

Principle

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

Procedures

1. Build the circuit



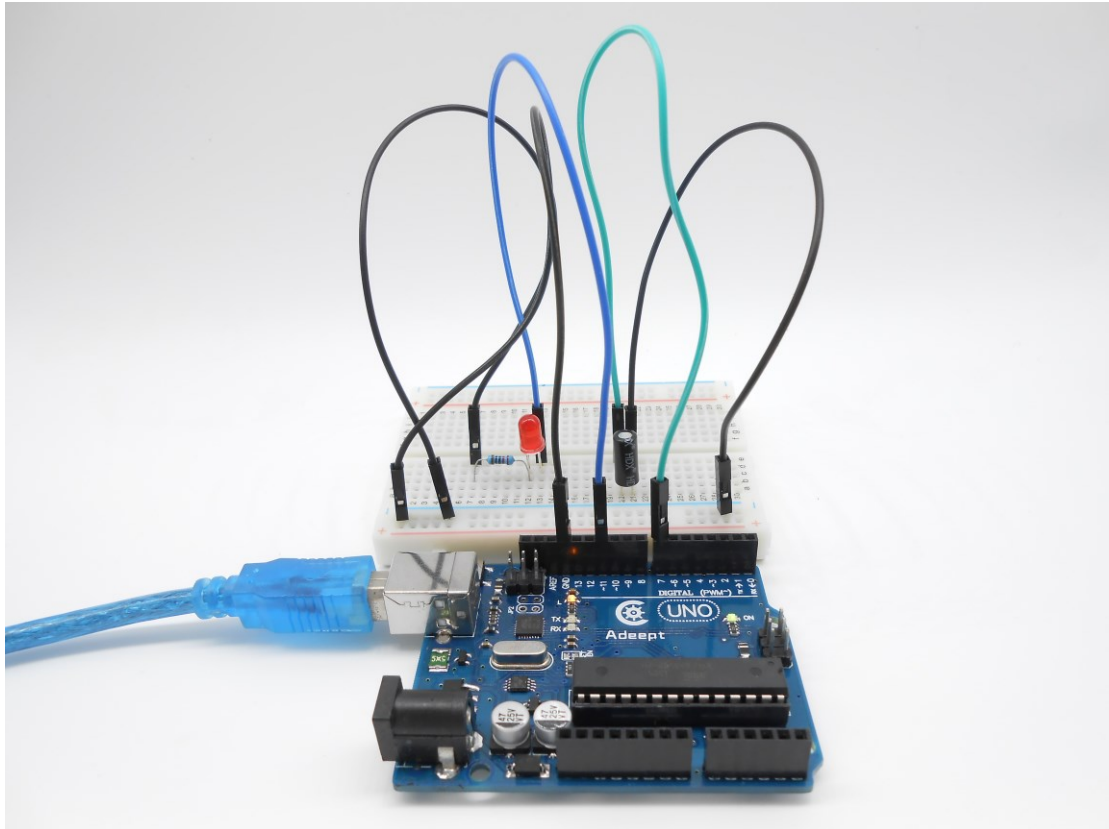
fritzing

2. Program

```
/******  
File name: 04_tiltSwitch.ino  
Description: Tilt switches to control the LED light on or off  
Website: www.adeept.com  
E-mail: support@adeept.com  
Author: Tom  
Date: 2015/05/02  
*****/  
  
int ledpin=11;          //definition digital 11 pins as pin to control the  
                        //LED  
int tiltSwitchpin=7;    //Set the digital 7 to tilt switch interface  
int val;                //Define variable val  
  
void setup()  
{  
  pinMode(ledpin,OUTPUT);    //Define small lights interface for the  
                             //output interface  
  pinMode(tiltSwitchpin,INPUT_PULLUP); //define the tilt switch  
                             //interface for input interface  
}  
  
void loop()  
{  
  val=digitalRead(tiltSwitchpin); //Read the number seven level value is  
                                  //assigned to val  
  if(val==LOW)                    //Detect tilt switch is disconnected, the  
                                  //tilt switch when small lights go out  
  { digitalWrite(ledpin,LOW); } //Output low, LED OFF  
  else                            //Detection of tilt switch is conduction,  
  //tilt the little lights up when the switch conduction  
  { digitalWrite(ledpin,HIGH); } //Output high, LED ON  
}
```

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 5 Breathing LED

Overview

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

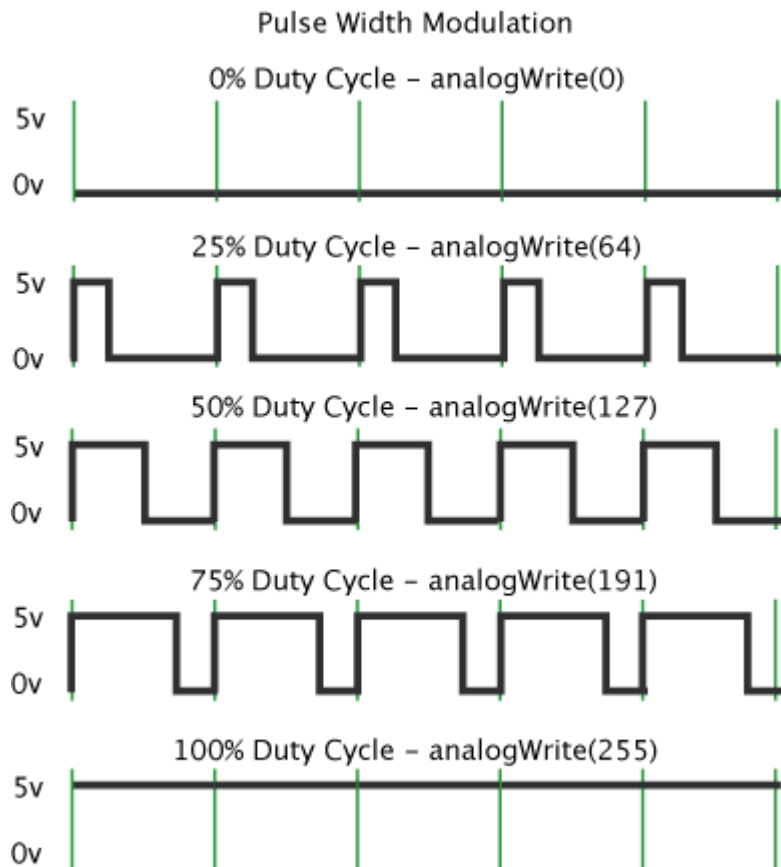
Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

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

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.



Key function:

● **analogWrite()**

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

Syntax

`analogWrite(pin, value)`

Parameters

pin: the pin to write to.

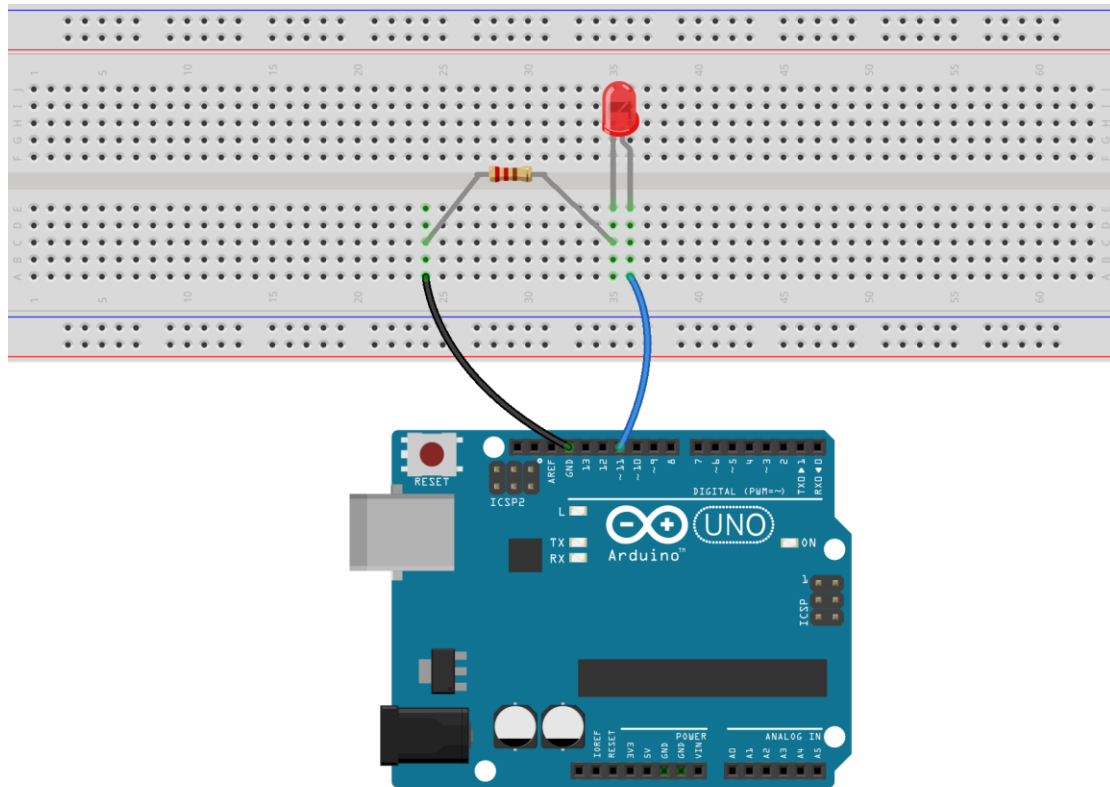
value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

nothing

Procedures

1. Build the circuit

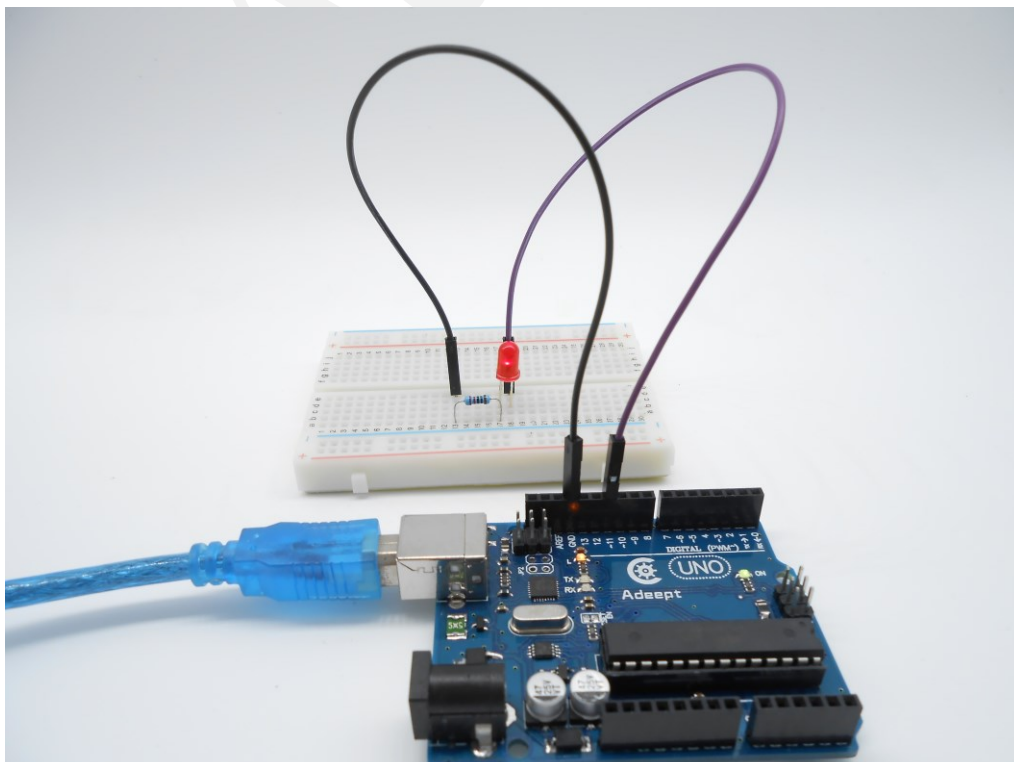


fritzing

2. Program

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

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



Summary

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

Adept

Lesson 6 Active Buzzer

Overview

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

Requirement

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

Principle

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

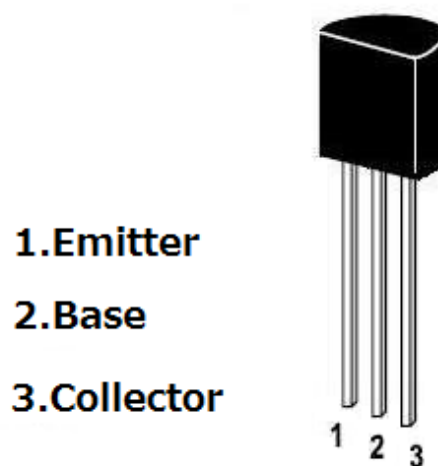


When you place the pins of buzzers upward, you can see that two buzzers are different, the buzzer that green circuit board exposed is the passive buzzer.

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

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

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



There are two driving circuit for the buzzer:

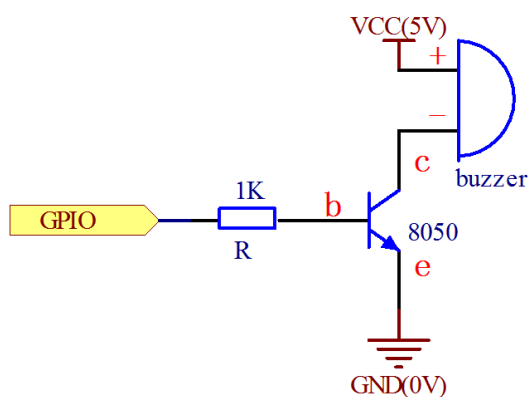


Figure1

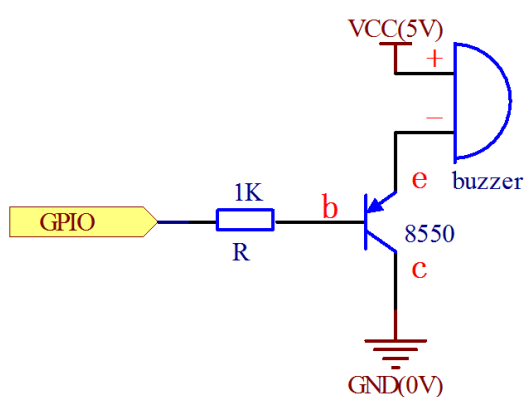


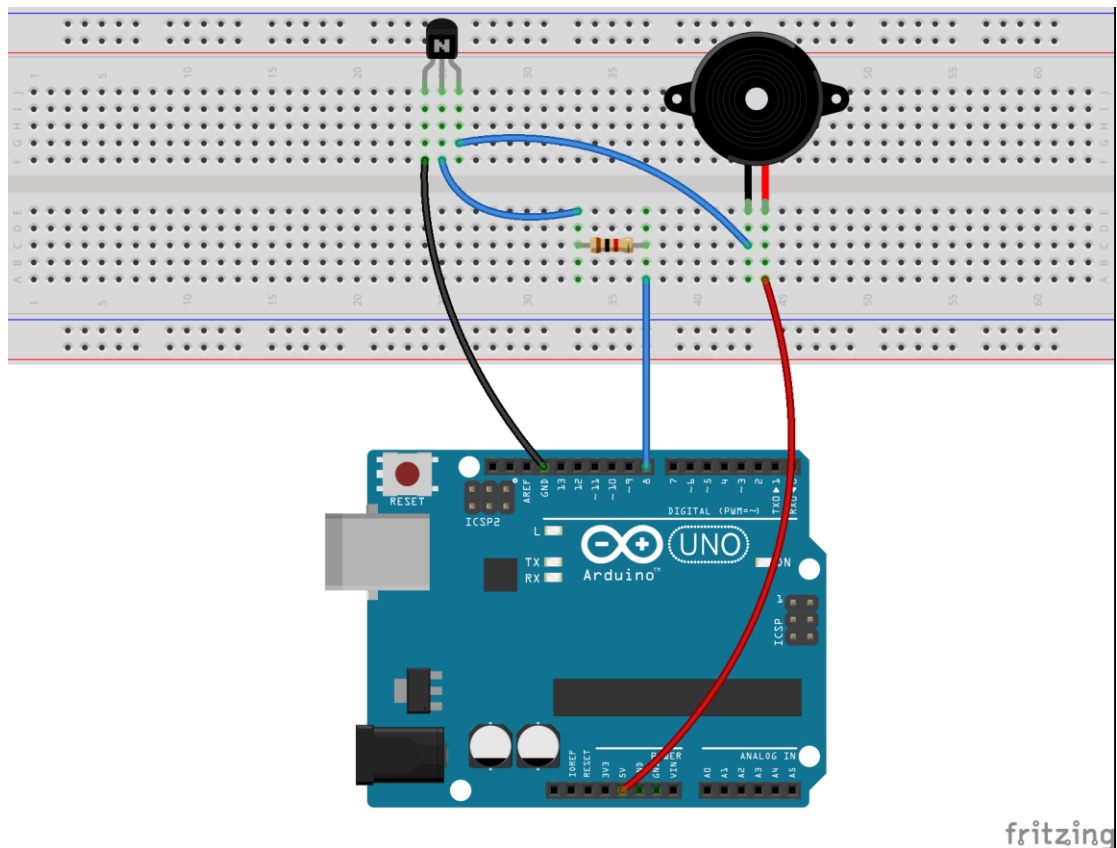
Figure2

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

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

Procedures

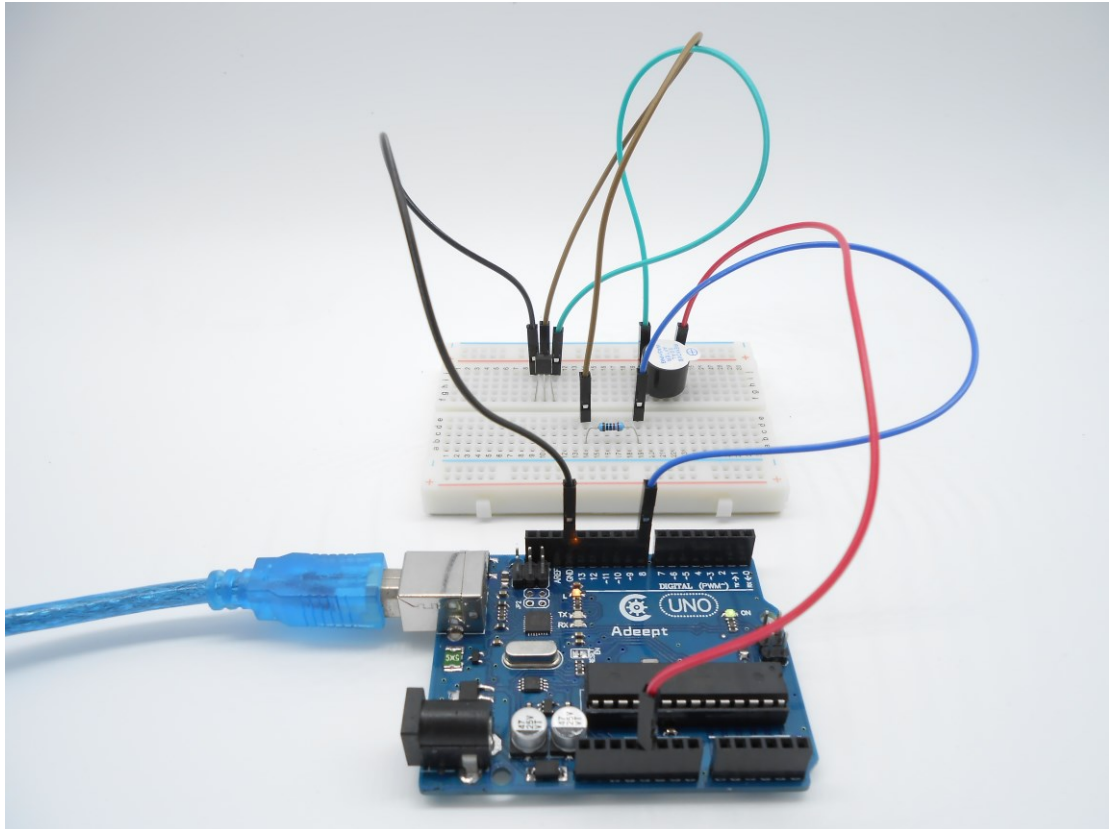
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 7 Controlling Relay

Overview

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

Requirement

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

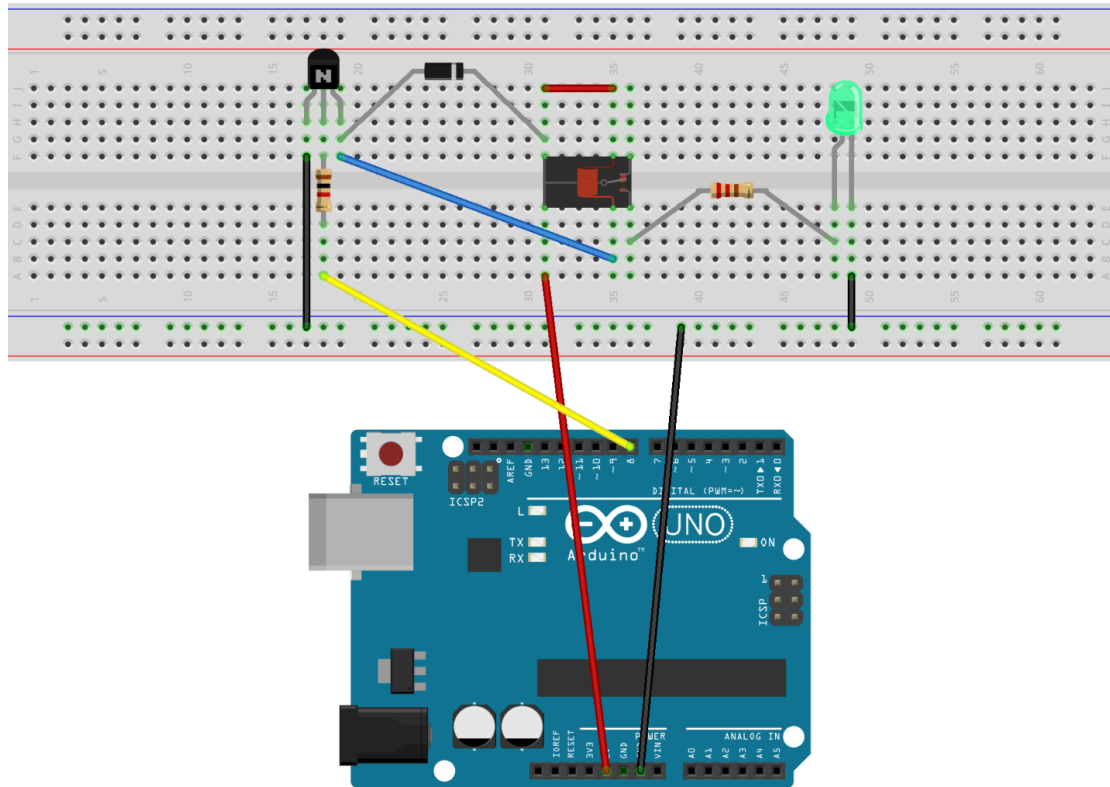
Principle

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

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

Procedures

1. Build the circuit

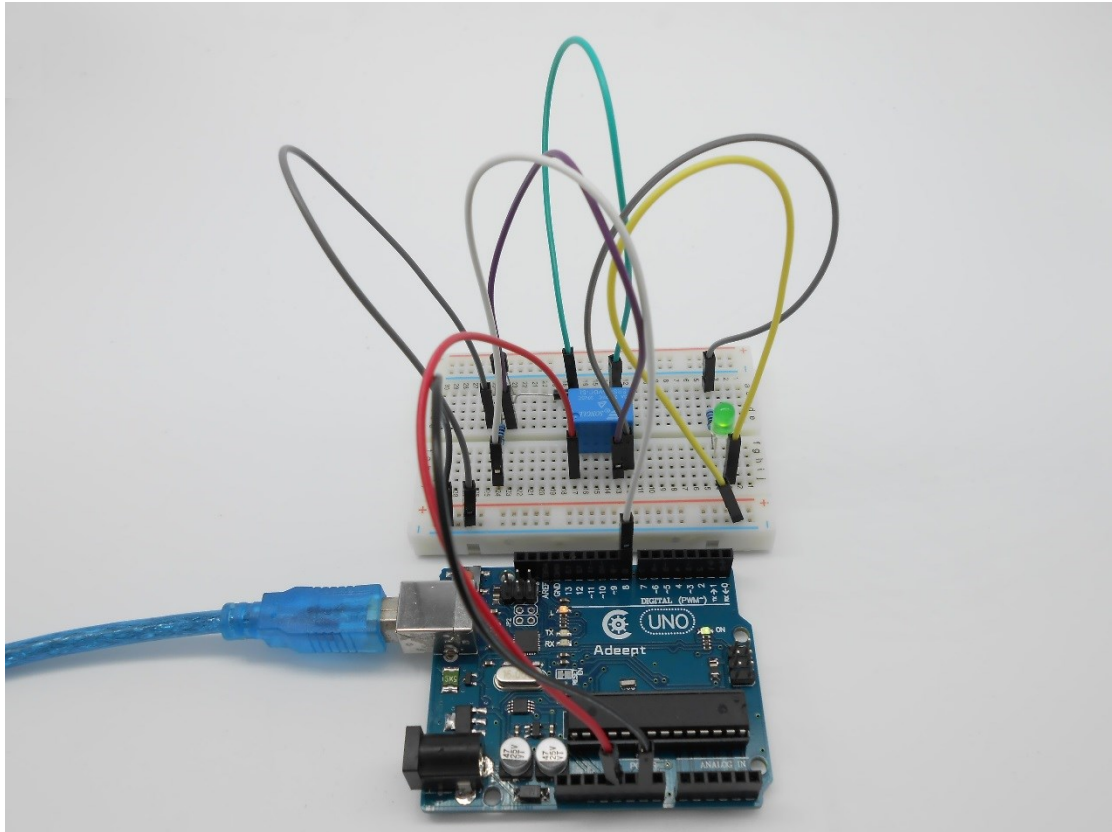


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 8 Controlling a RGB LED by PWM

Overview

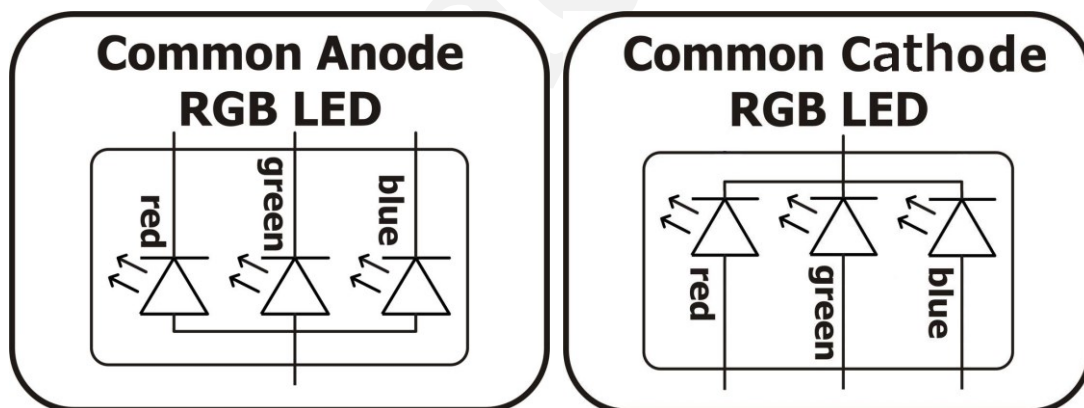
In this lesson, we will program the Arduino for RGB LED control, and make RGB LED emits a various of colors of light.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* RGB LED
- 3* 220 Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

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



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

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

Procedures

1. Build the circuit



Summary

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

Adept

Lesson 9 7-segment display

Overview

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

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* 220 Ω Resistor
- 1* 7-Segment display
- 1* Breadboard
- Several Jumper wires

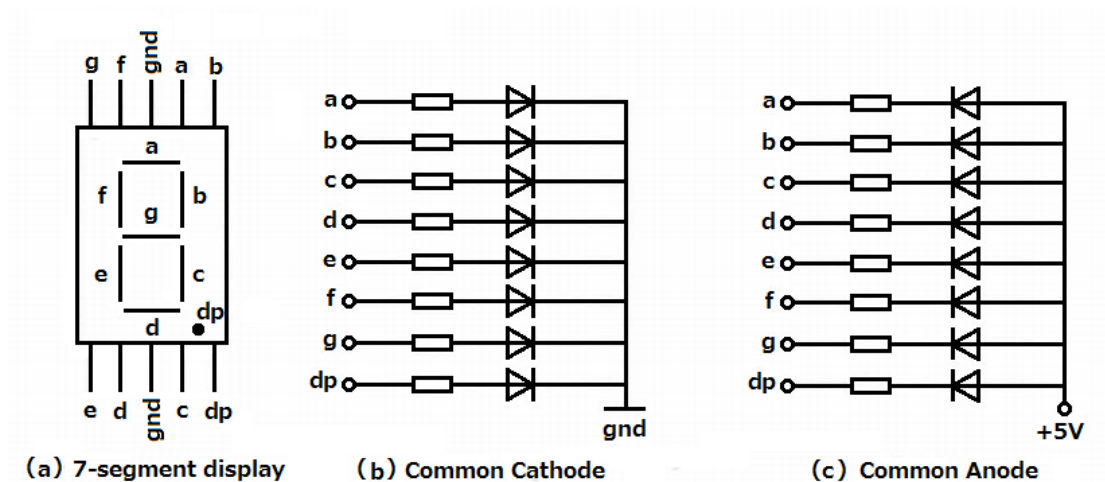
Principle

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

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

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

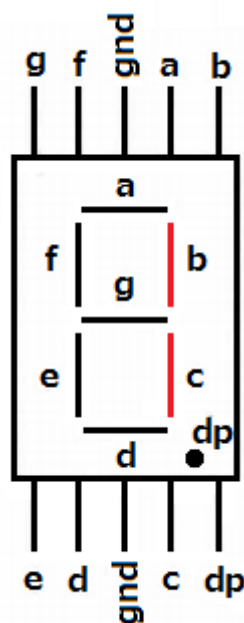
The segment display can be divided into common anode and common cathode segment display by internal connections.



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

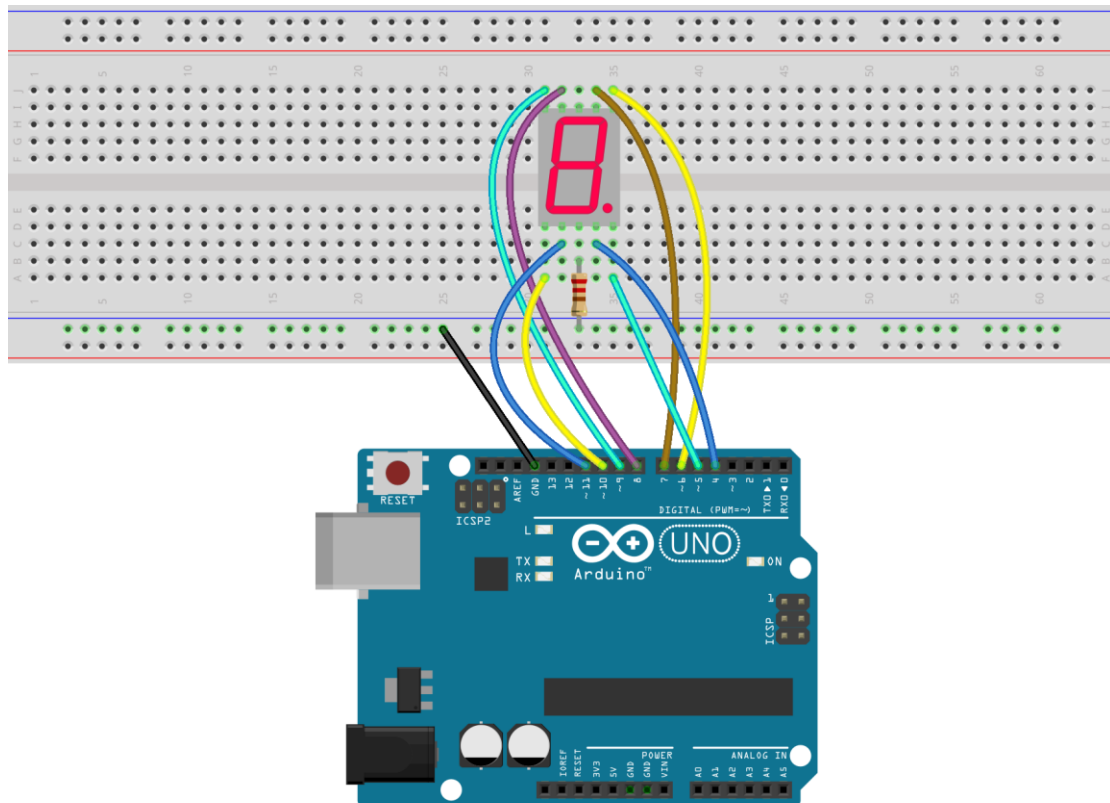
Each segment of a segment display is composed of LED, so a resistor is needed for protecting the LED.

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



Procedures

1. Build the circuit

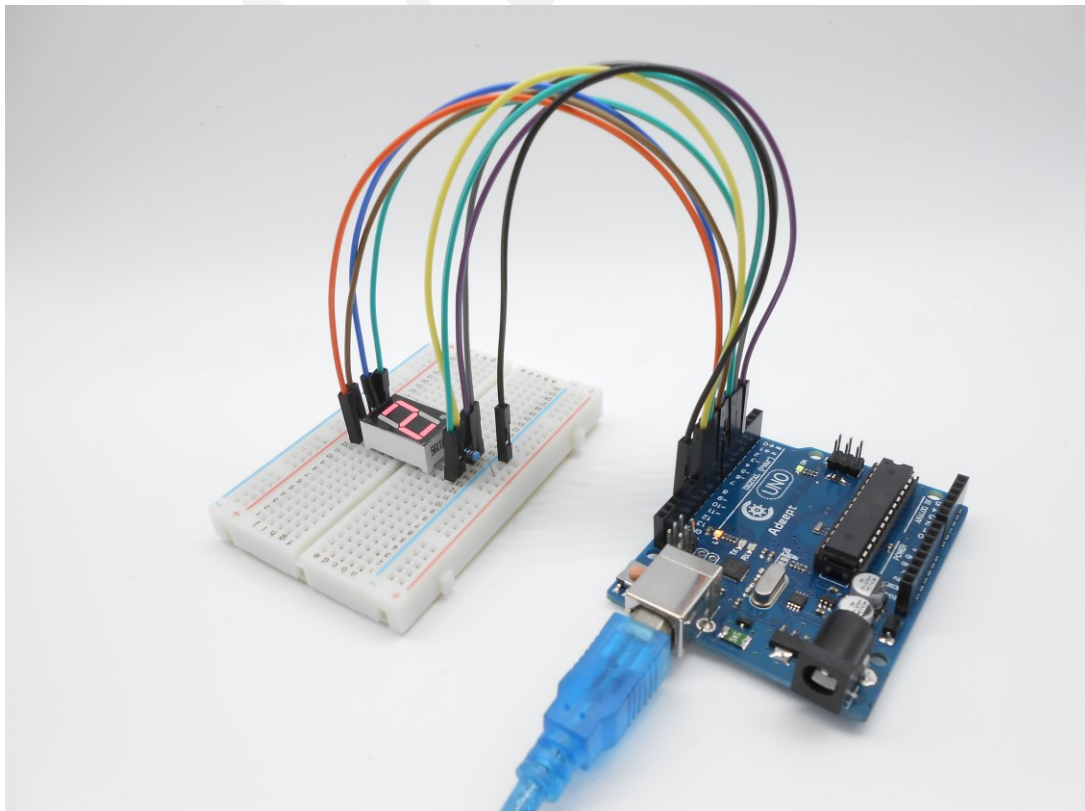


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Adept

Lesson 10 Serial Port

Overview

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

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

1. Serial ports

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

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

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

2. Key function

●begin()

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

Syntax

`Serial.begin(speed)`

Parameters

speed: in bits per second (baud) - long

Returns

nothing

●print()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Syntax

`Serial.print(val)`

`Serial.print(val, format)`

Parameters

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

Returns

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

● `println()`

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

Syntax

`Serial.println(val)`

`Serial.println(val, format)`

Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns

byte

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

● `read()`

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

Syntax

`Serial.read()`

Parameters

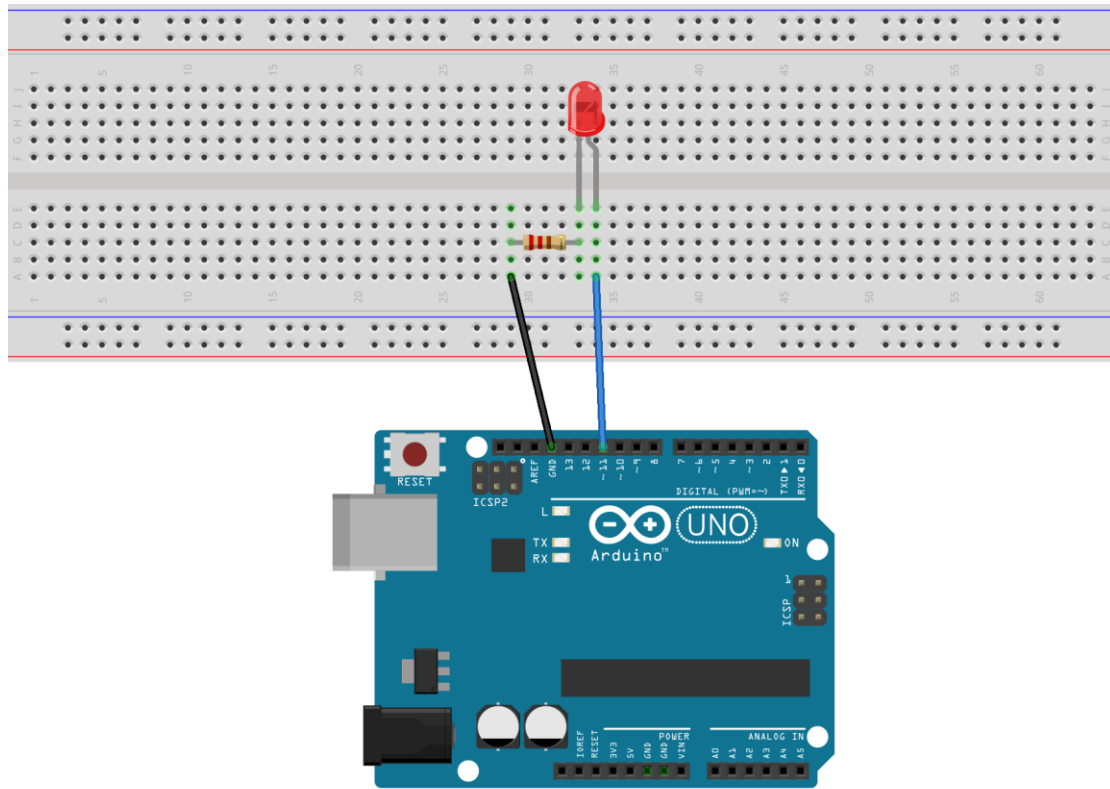
None

Returns

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

Procedures

1. Build the circuit



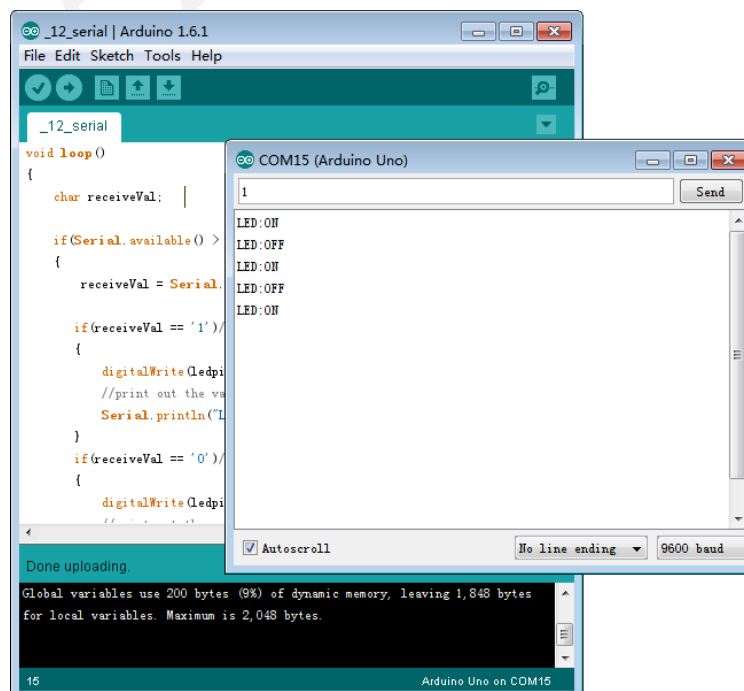
fritzing

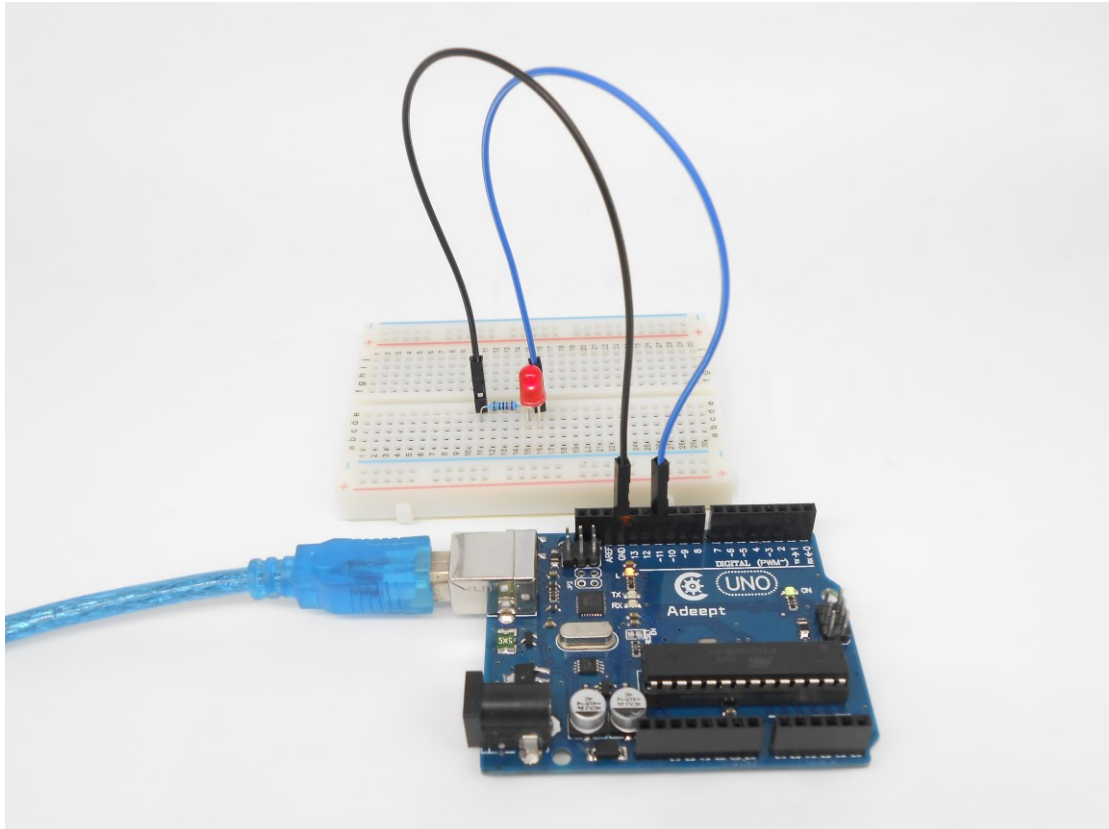
2. Program

3. Compile the program and upload to Arduino UNO board

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

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





Summary

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

Lesson 11 LCD1602

Overview

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

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LCD1602
- 1* 10K Ω Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

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

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

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

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

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

Key function:

● **begin()**

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

Syntax

`lcd.begin(cols, rows)`

Parameters

lcd: a variable of type LiquidCrystal

cols: the number of columns that the display has

rows: the number of rows that the display has

● **setCursor()**

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

Syntax

`lcd.setCursor(col, row)`

Parameters

lcd: a variable of type LiquidCrystal

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

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

● **scrollDisplayLeft()**

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

Syntax

`lcd.scrollDisplayLeft()`

Parameters

lcd: a variable of type LiquidCrystal

Example

`scrollDisplayLeft()` and `scrollDisplayRight()`

See also

`scrollDisplayRight()`

● **print()**

Prints text to the LCD.

Syntax

`lcd.print(data)`

`lcd.print(data, BASE)`

Parameters

lcd: a variable of type LiquidCrystal

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

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

Returns

byte

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

● `clear()`

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

Syntax

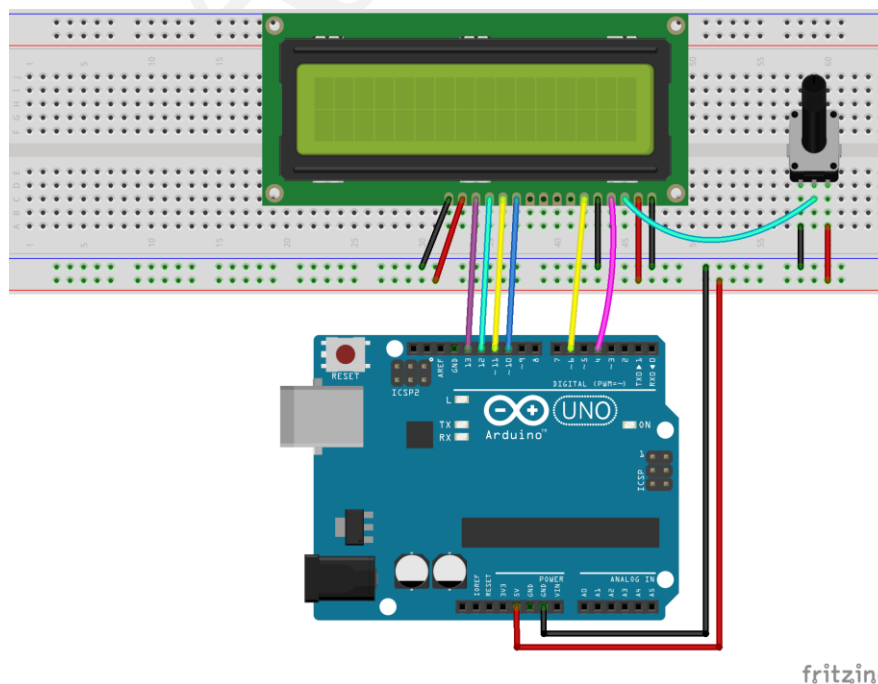
`lcd.clear()`

Parameters

lcd: a variable of type LiquidCrystal

Procedures

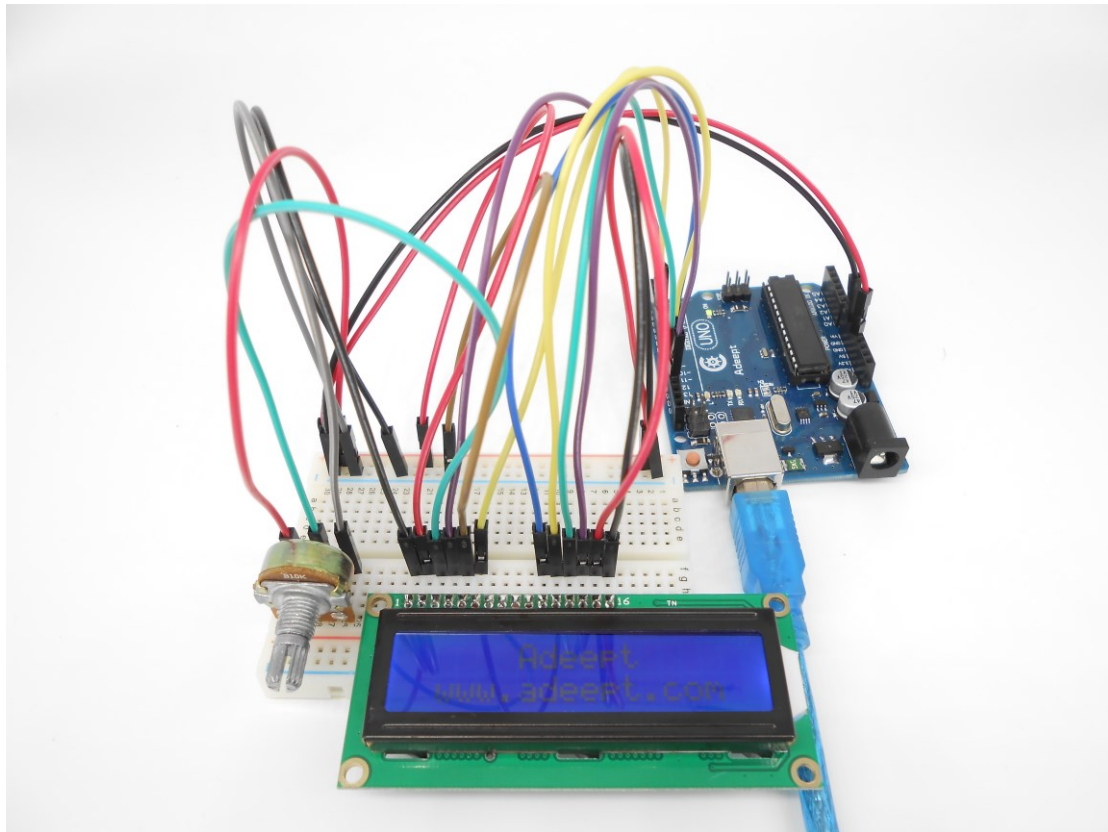
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 12 Photoresistor

Overview

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

Requirement

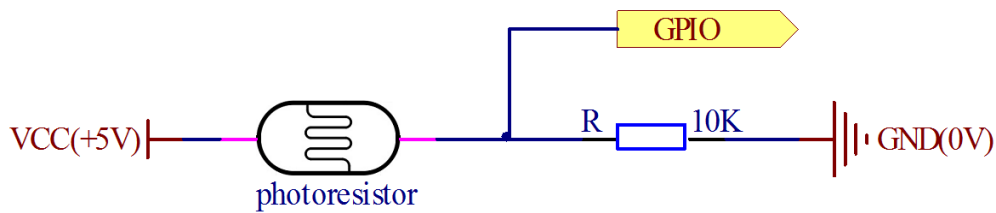
- 1* Arduino UNO
- 1* USB cable
- 1* LCD1602
- 1* Photoresistor
- 1* 10K Ω Resistor
- 1* 10K Ω Potentiometer
- 1* Breadboard
- Several Jumper wires

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

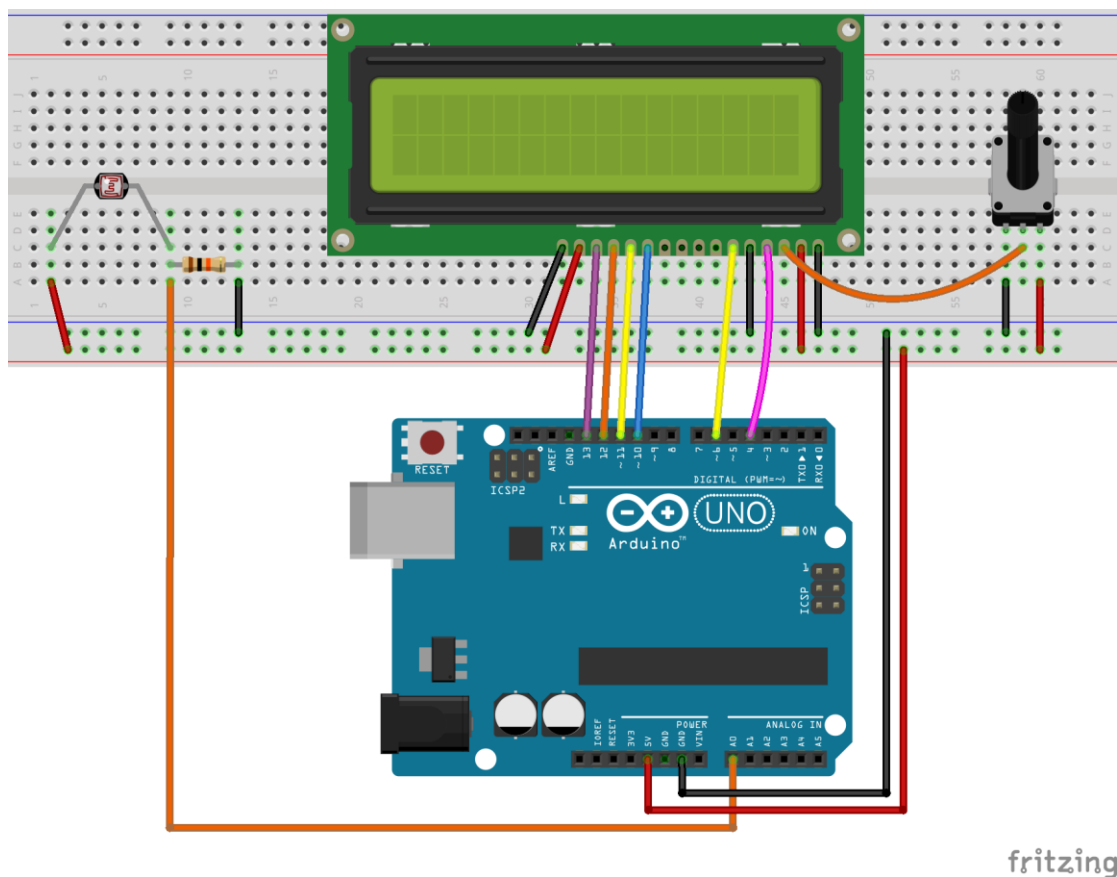
The schematic diagram of this experiment is shown below:



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

Procedures

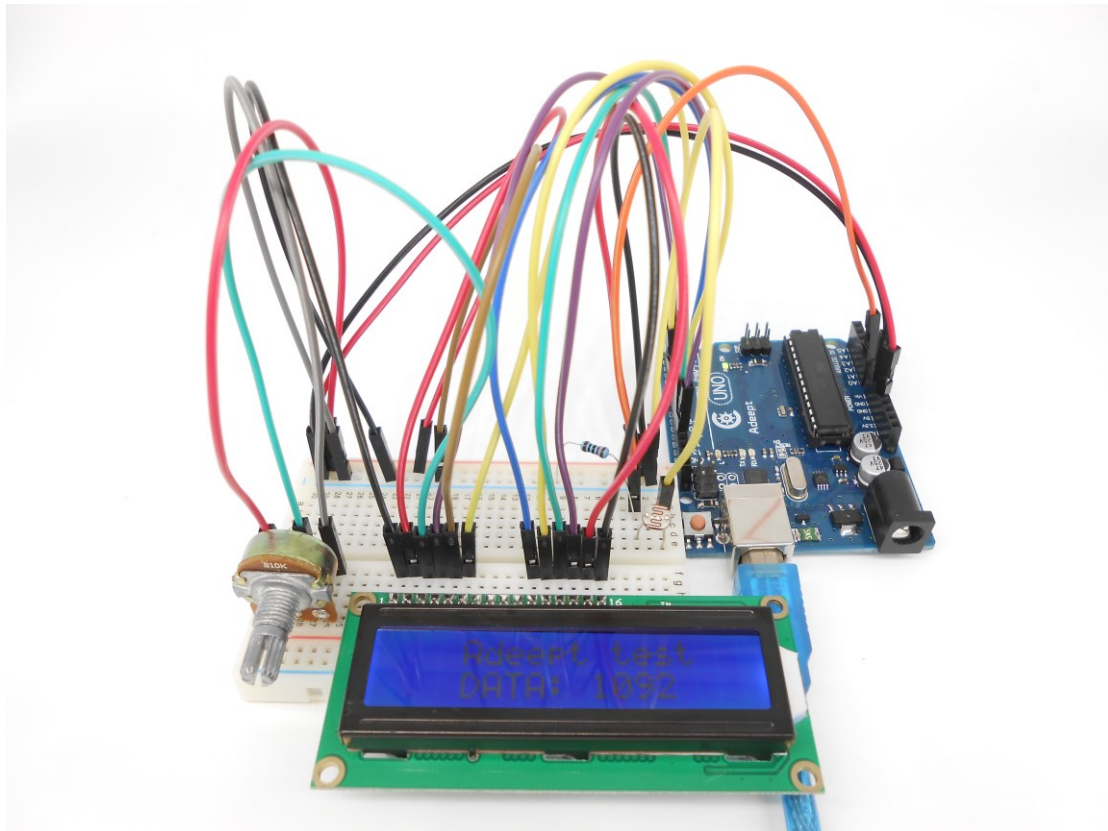
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 13 Using a thermistor to measure the temperature

Overview

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

Requirement

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

Principle

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

MF52 thermistor key parameters:

B-parameter : 3470.

25°C resistor : 10KΩ.

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

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

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

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

e : 2.718281828459 ;

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

T_1 : the Kelvin temperature that you want to measure.

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

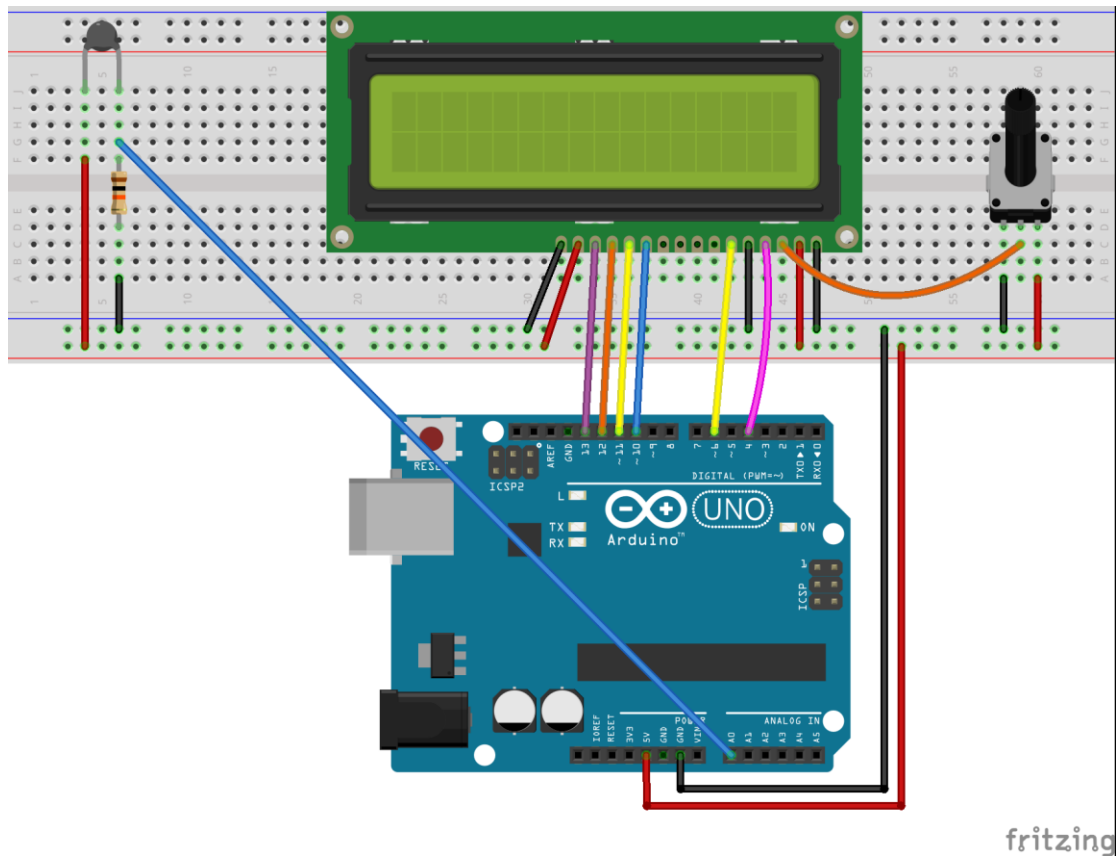
Kelvin temperature = 273.15 (absolute temperature) + degrees Celsius;

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

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

Procedures

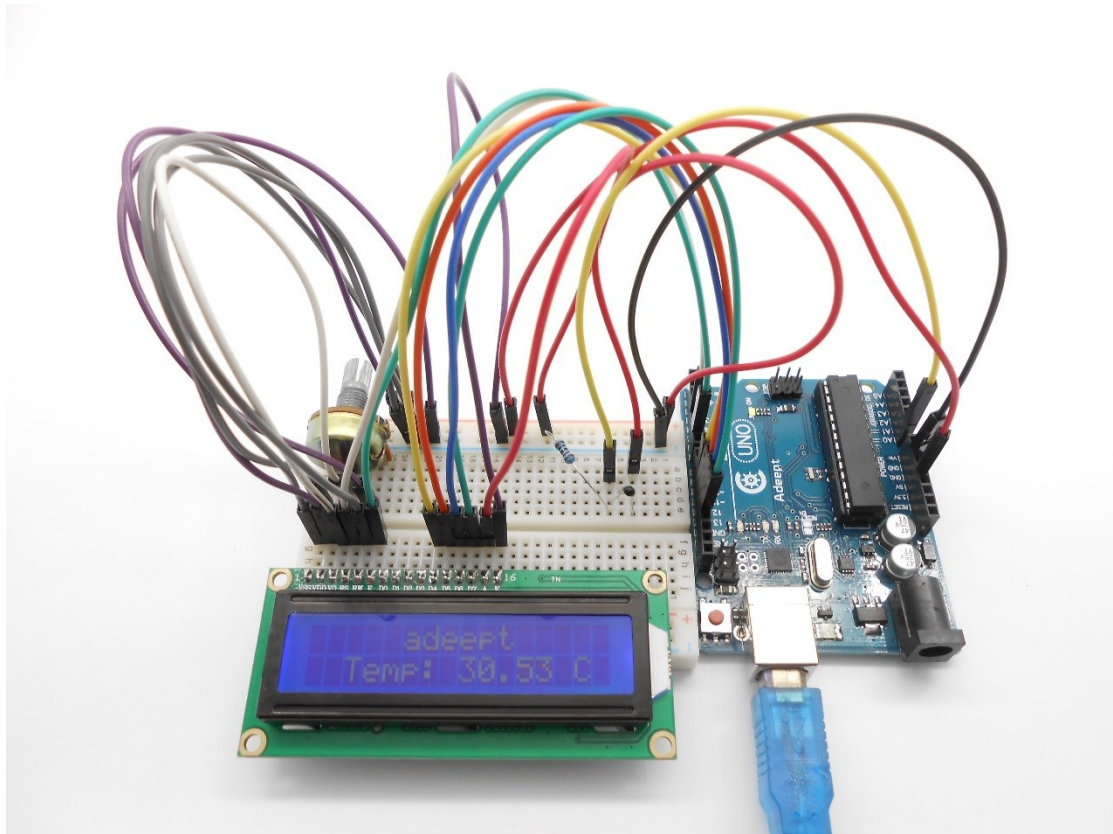
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 14 DC motor

Overview

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

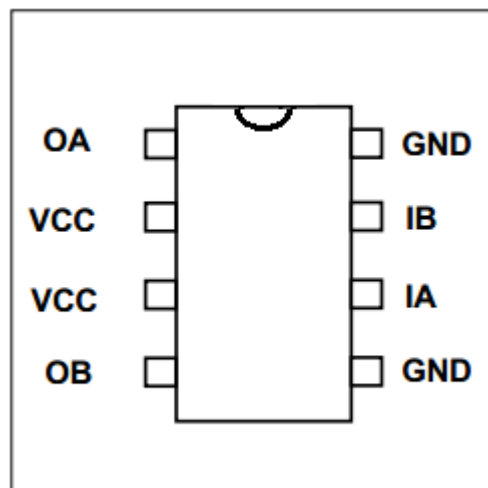
Requirement

- 1* Arduino UNO
- 1* USB Cable
- 1* L9110 DC Motor driver
- 1* DC motor
- 1* Battery holder
- 1* Breadboard
- Several Jumper wires

Principle

1. L9110

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



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

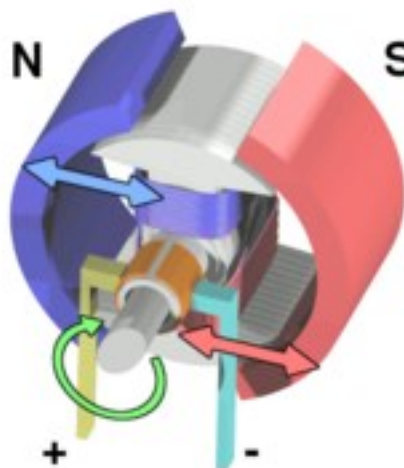
VCC: Power supply (+5V)

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

IA, IB: The input terminal of drive signal.

2. DC motor

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



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



3. Key functions

● switch / case statements

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

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

Example

```
switch (var) {  
  case 1:  
    //do something when var equals 1  
    break;  
  case 2:  
    //do something when var equals 2  
    break;  
  default:  
    // if nothing else matches, do the default  
    // default is optional  
}
```

Syntax

```
switch (var) {  
  case label:
```

```

    // statements
    break;
case label:
    // statements
    break;
default:
    // statements
}

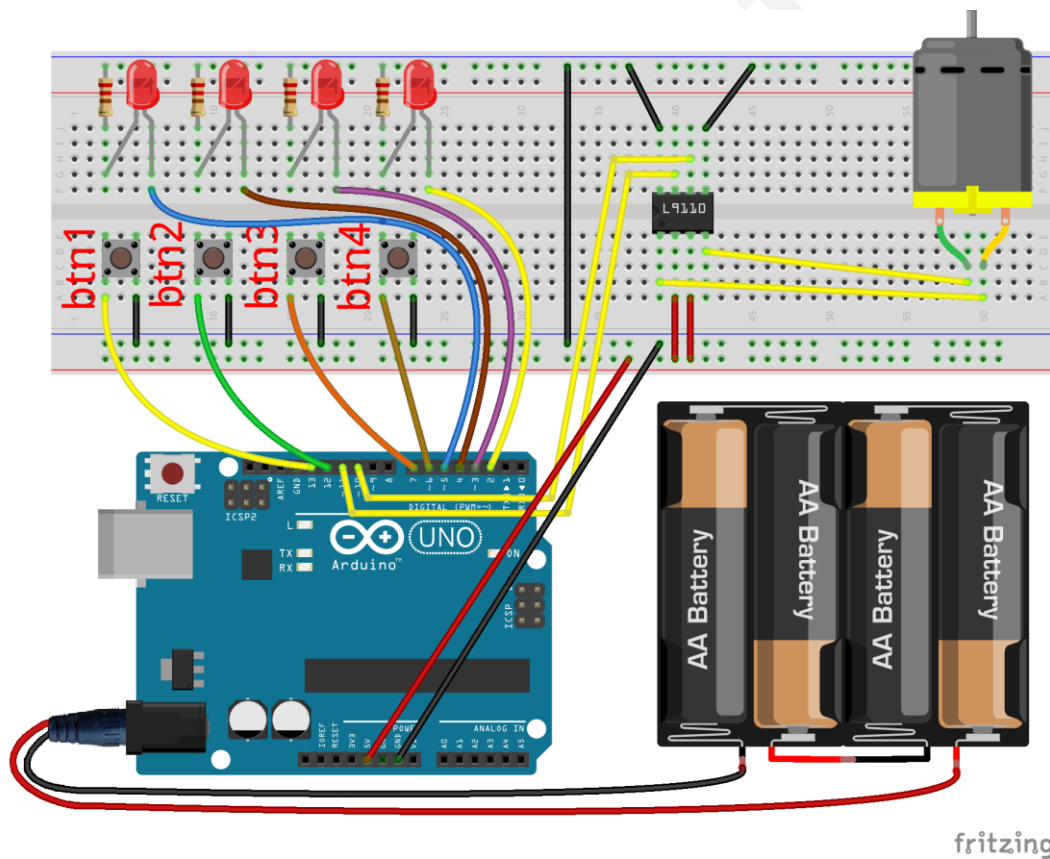
```

Parameters

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

Procedures

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



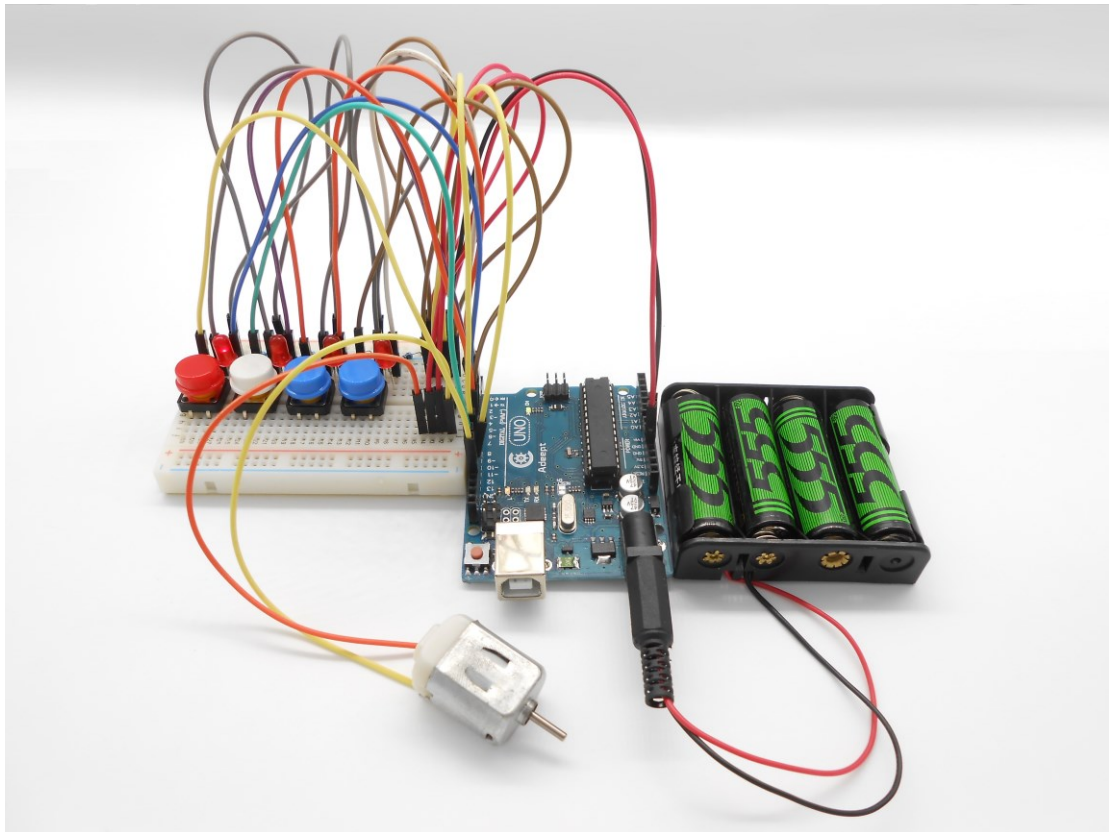
fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

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

that the current button is clicked.



Summary

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

Lesson 15 Controlling Servo motor

Overview

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

Requirement

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

Principle

1. Servo motor

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

2. Servo library

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

3. Key functions:

● `attach()`

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

Syntax

`servo.attach(pin)`

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

Parameters

servo: a variable of type Servo

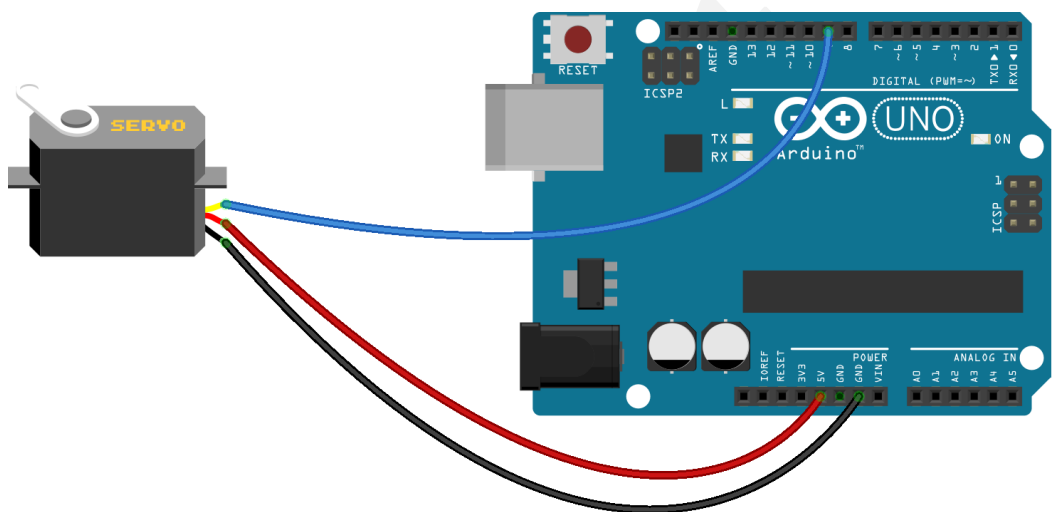
pin: the number of the pin that the servo is attached to

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

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

Procedures

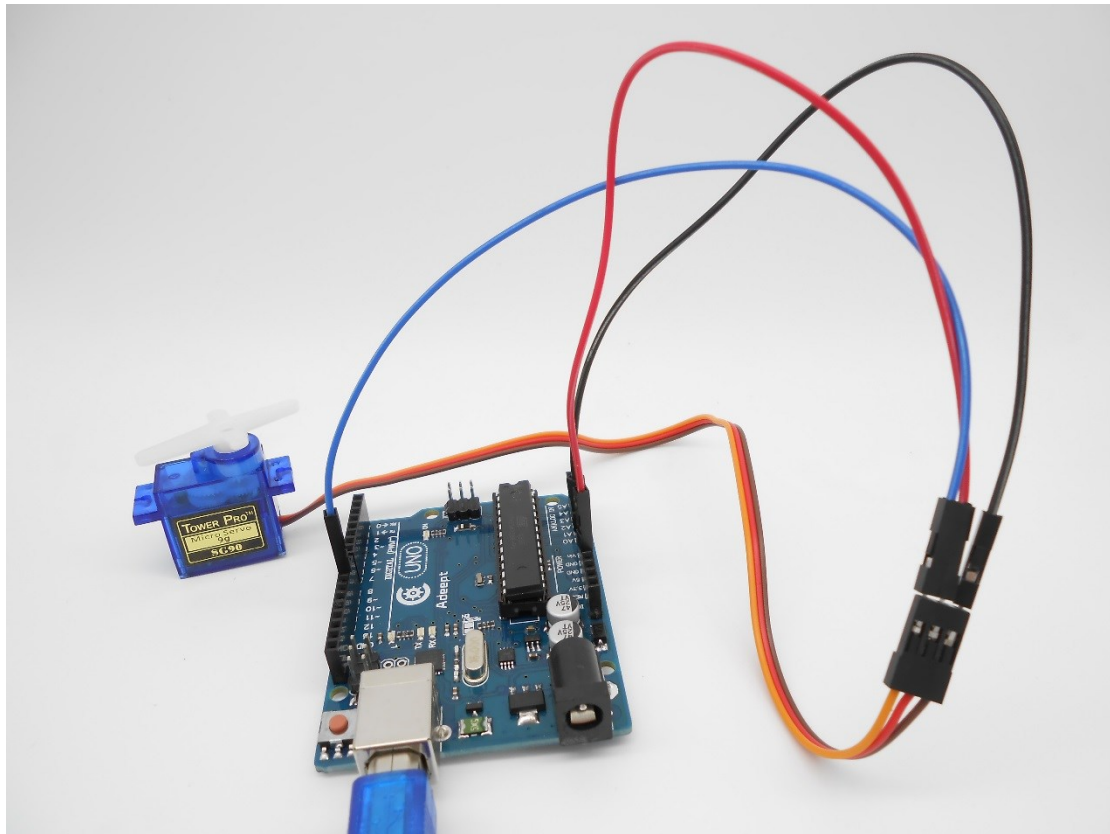
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

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



Summary

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

Lesson 16 ultrasonic distance sensor

Overview

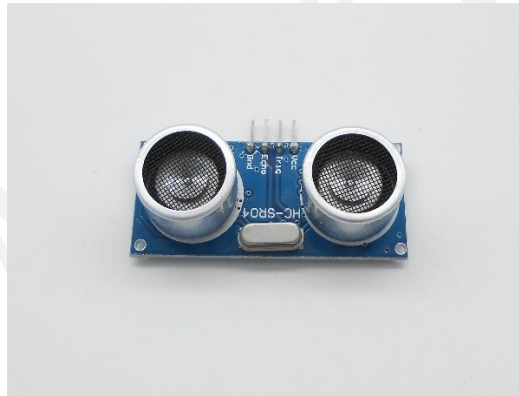
In this lesson, we will learn how to measure the distance by the ultrasonic distance sensor.

Requirement

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

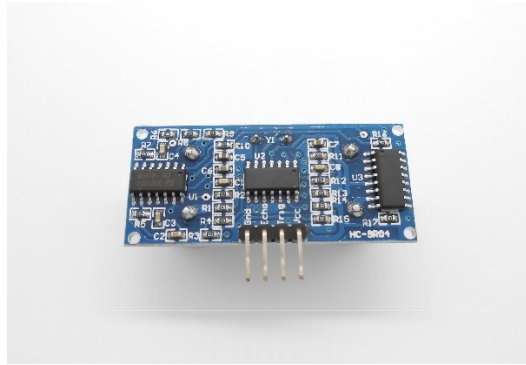
Principle

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



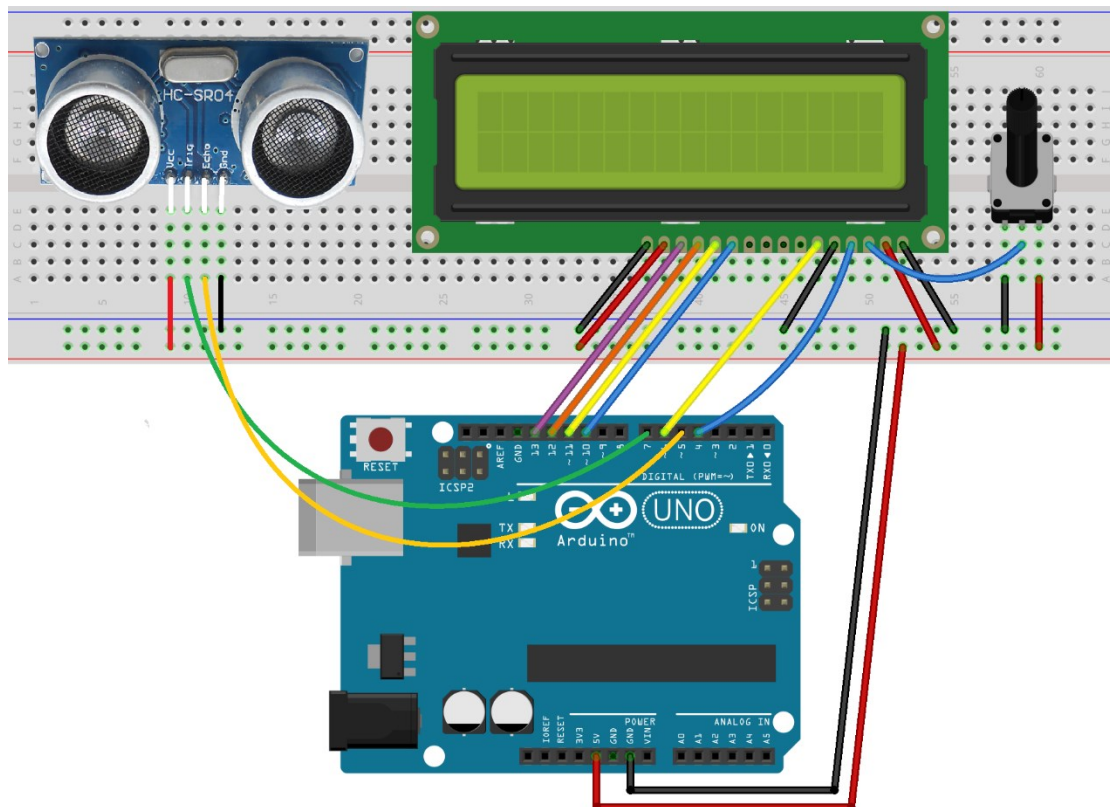
Ultrasonic sensors provide a measurement of the time it takes for sound to bounce off an object and return to the sensor. The “ping” sound pulse is generated when the pingPin level goes HIGH for two micro-seconds. The sensor will then generate a pulse that terminates when the sound returns. The width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

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



Procedures

1. Build the circuit

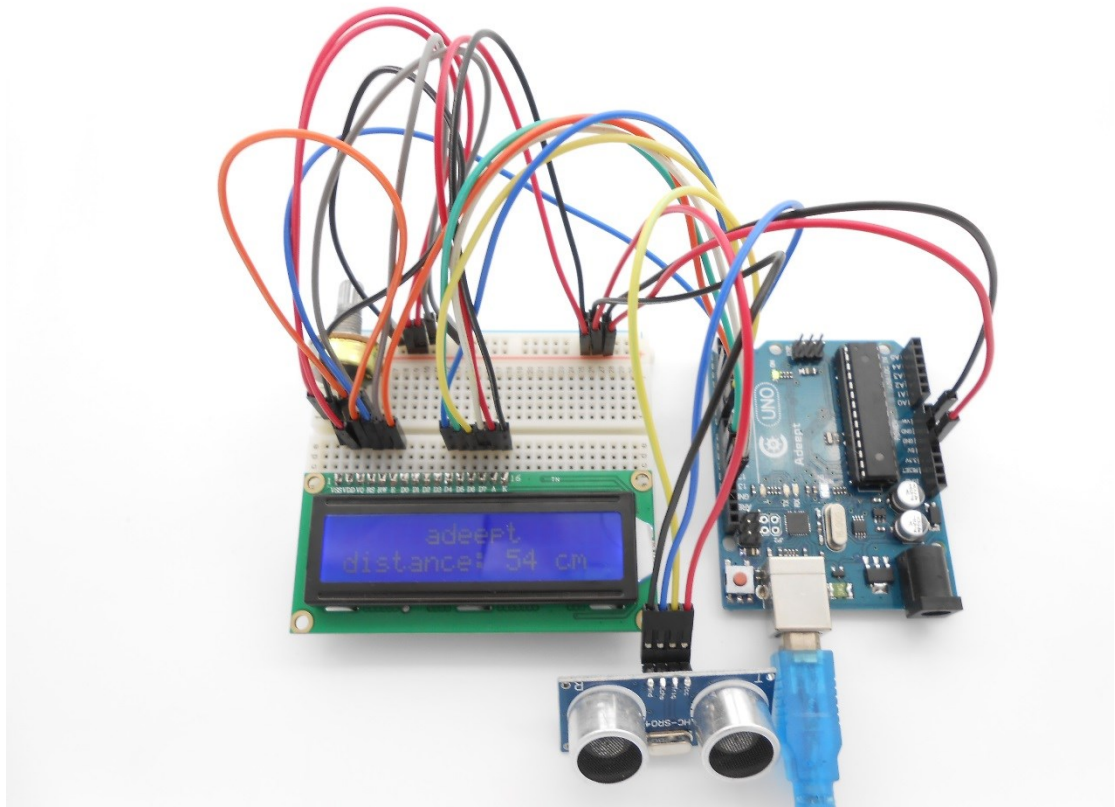


fritzing

2. Program

3. Compile the program and upload to Arduino UNO board

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



Lesson 17 Control a servo with ultrasonic distance sensor

Overview

In this lesson, we will measure the distance with the ultrasonic module, and then convert the distance into the rotation angle of the servo.

Requirement

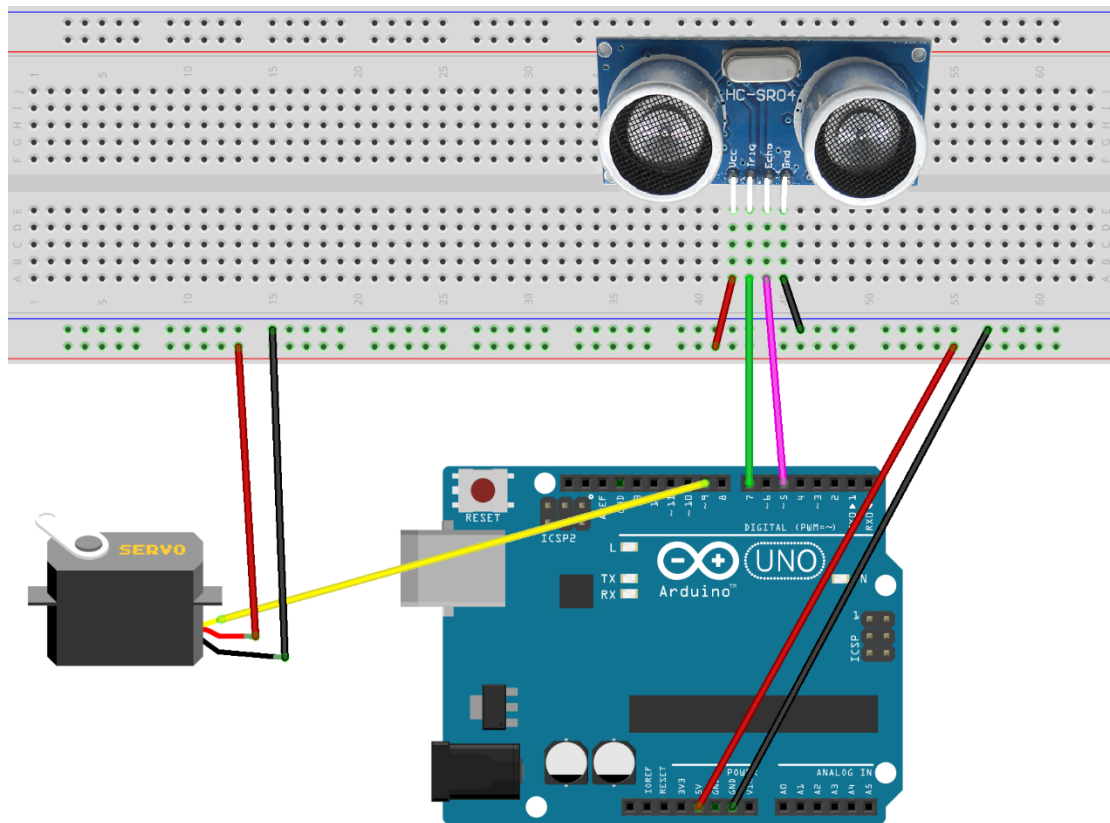
- 1* Arduino UNO
- 1* USB Cable
- 1* Ultrasonic Distance Sensor
- 1* Servo
- 1* Breadboard
- Several Jumper Wires

Principle

In this experiment, we collect the distance data between ultrasonic module and the obstacle with the ultrasonic distance sensor. Then, converting the distance data into the rotation angle of the servo by programing the Arduino UNO.

Procedures

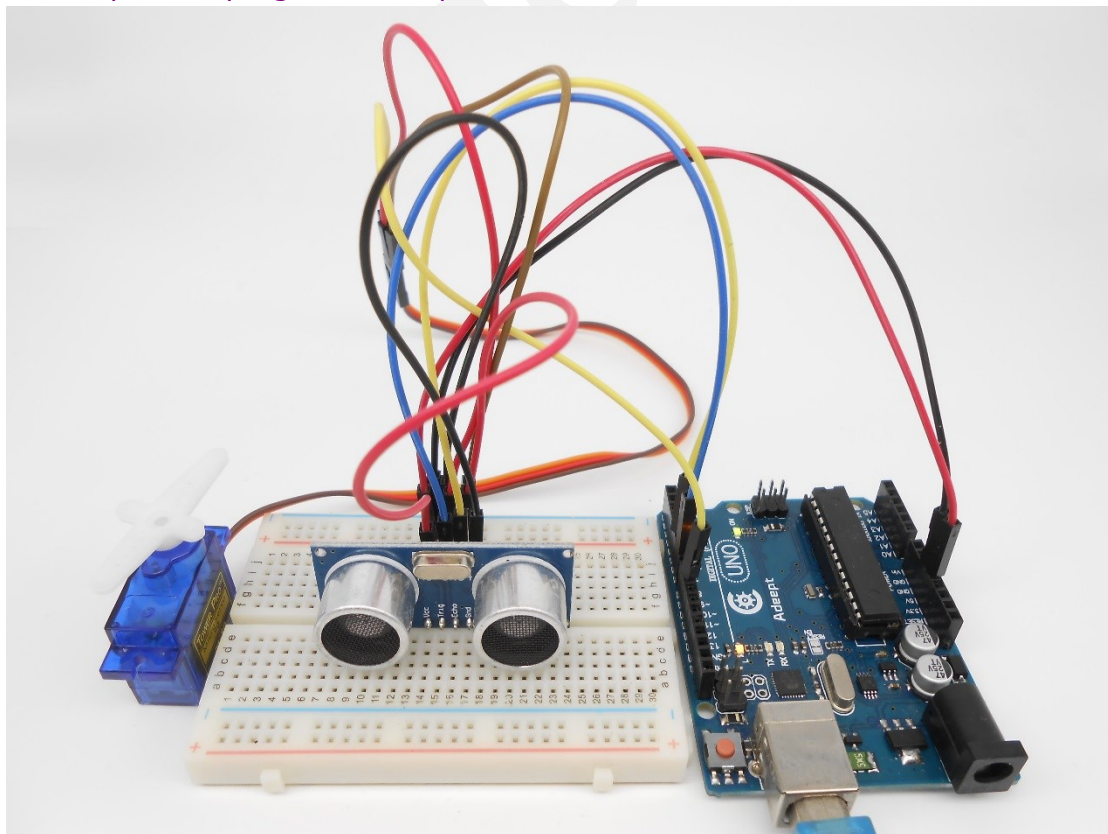
1. Build the circuit



fritzing

2. Program

3. Compile the program and upload to Arduino UNO board



Lesson 18 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* Arduino UNO
- 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.

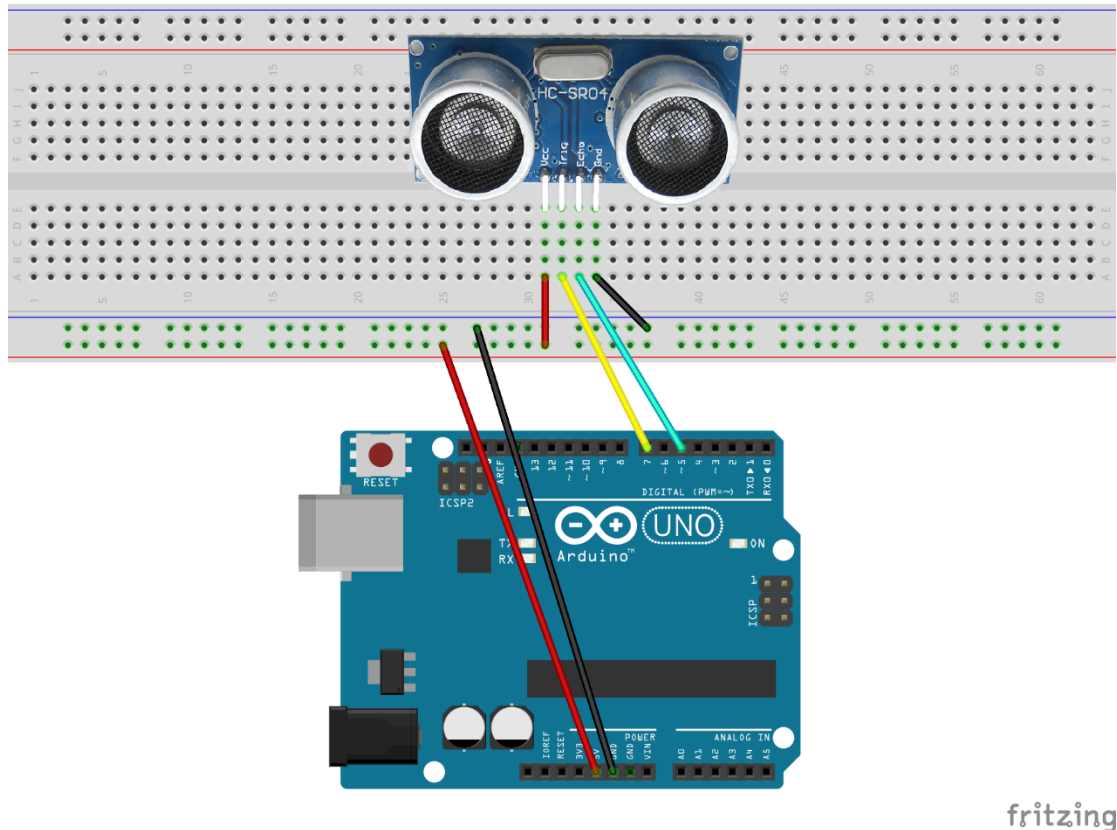
●Name: read()

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.

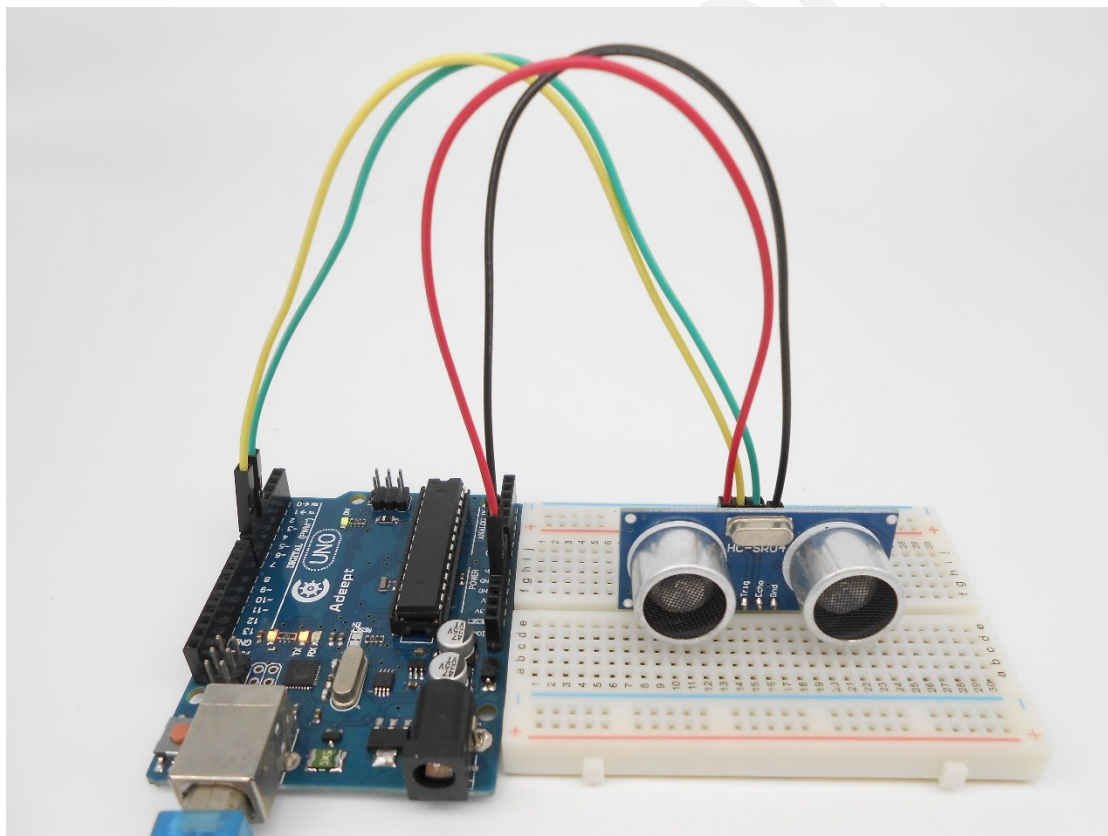
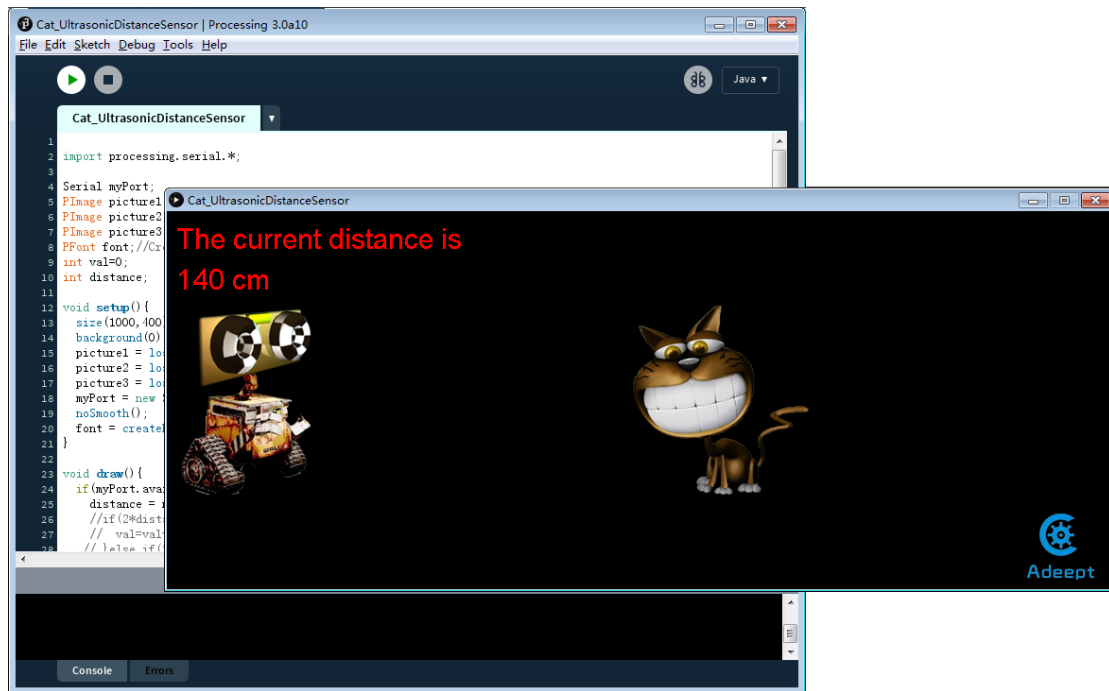
Procedures

1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board
4. Run processing software (Cat_UltrasonicDistanceSensor.pde)



Lesson 19 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* Arduino UNO
- 1* USB Cable
- 1* Light Sensor (Photoresistor)
- 1* 10K Ω 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)

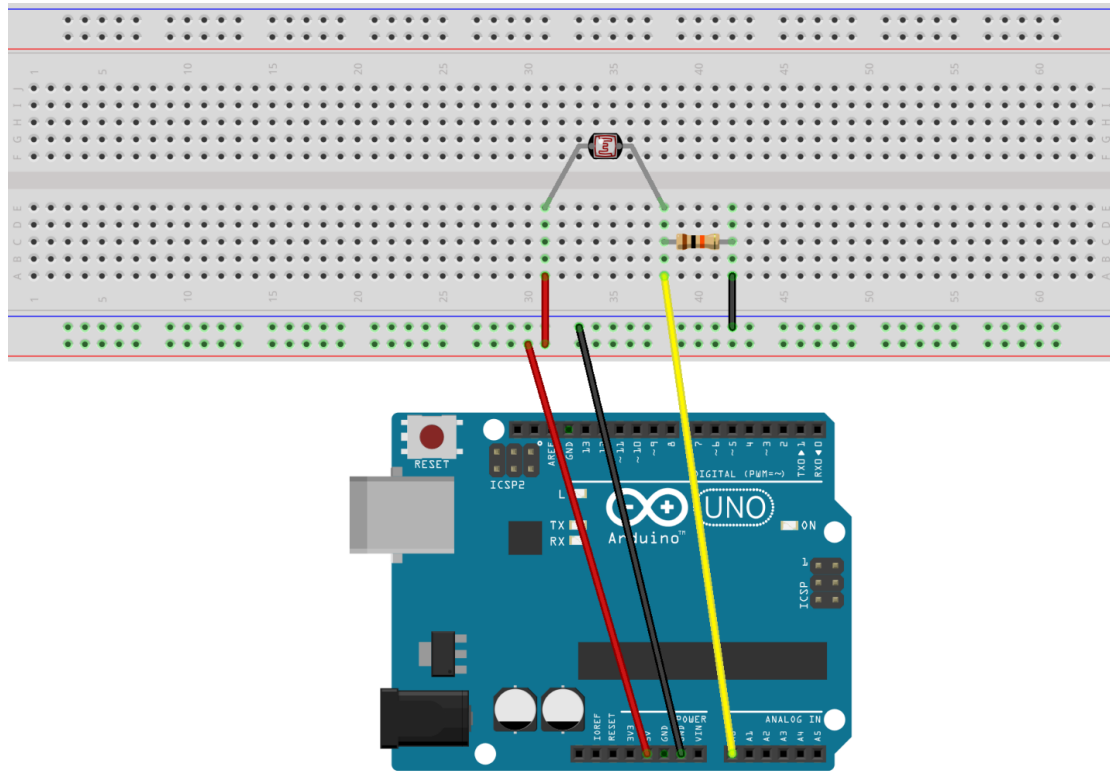
v3 float: blue or brightness value (depending on current color mode)

Returns

void

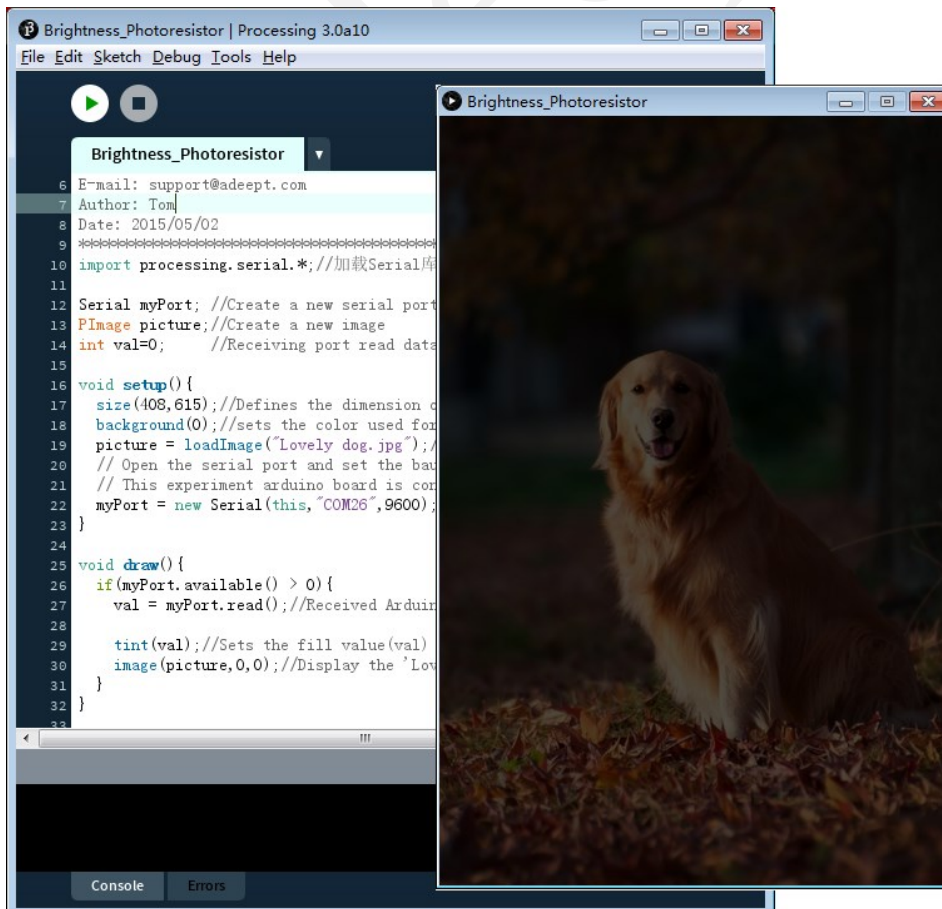
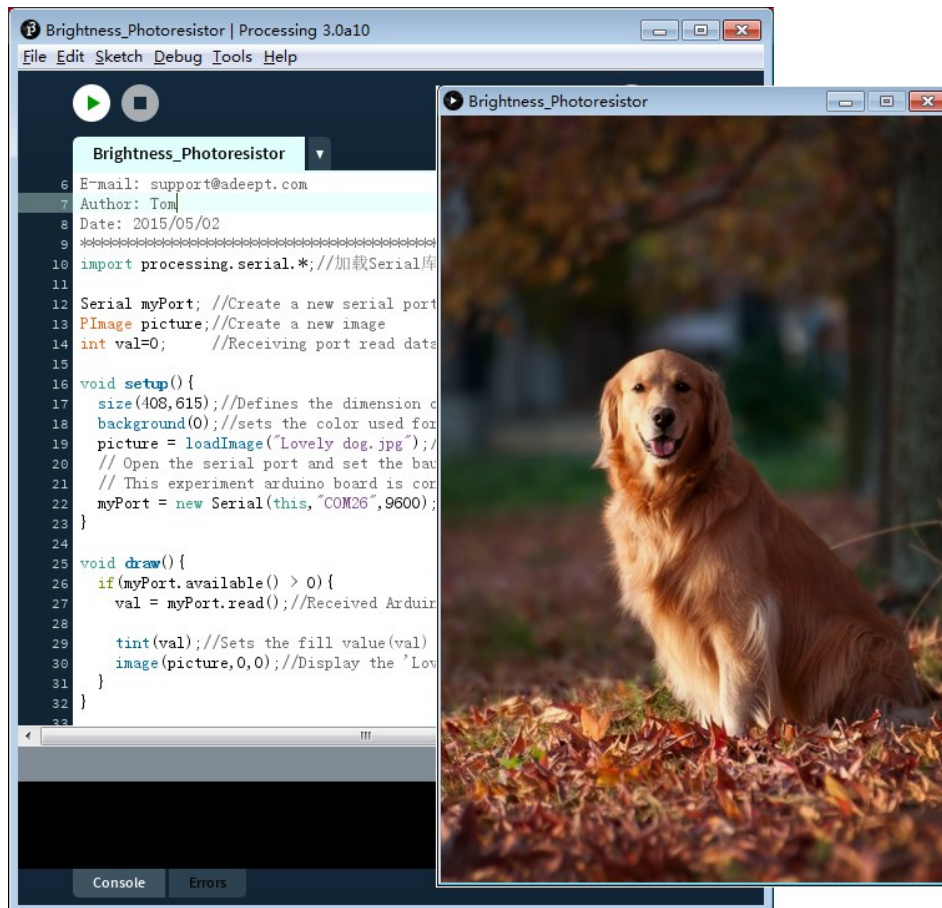
Procedures

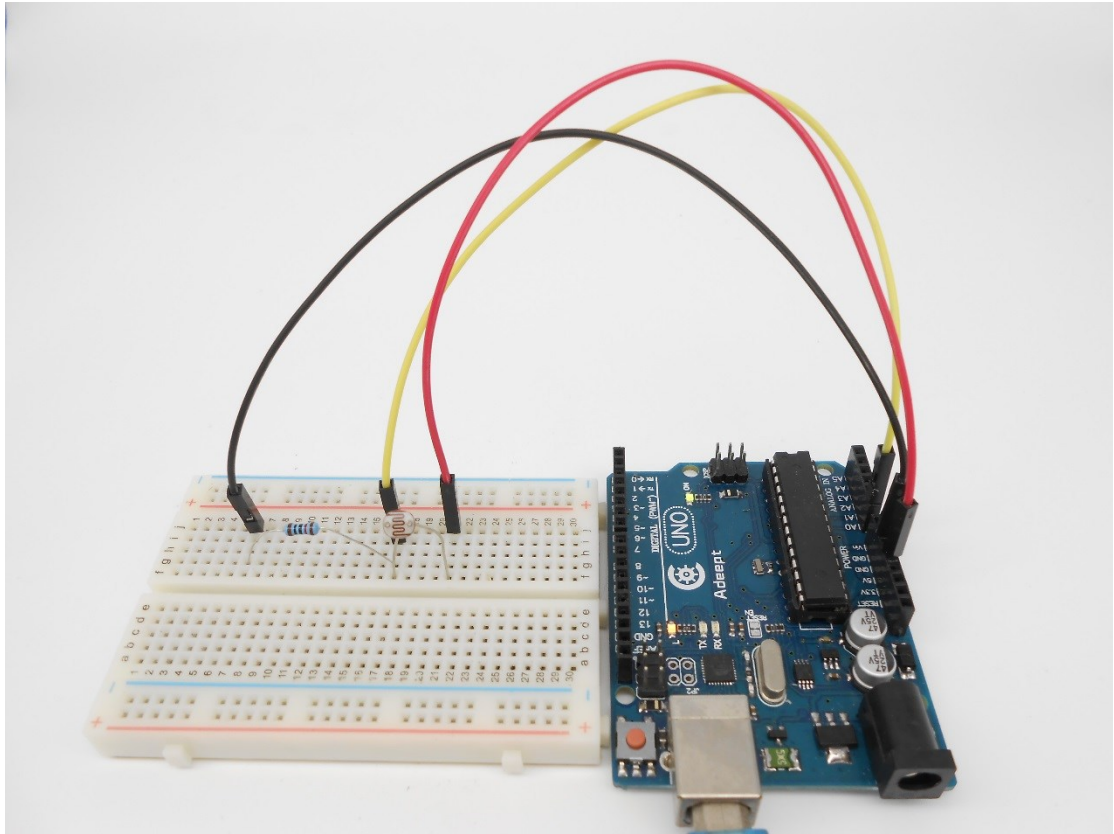
1. Build the circuit



fritzing

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





Lesson 20 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* Arduino UNO
- 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:

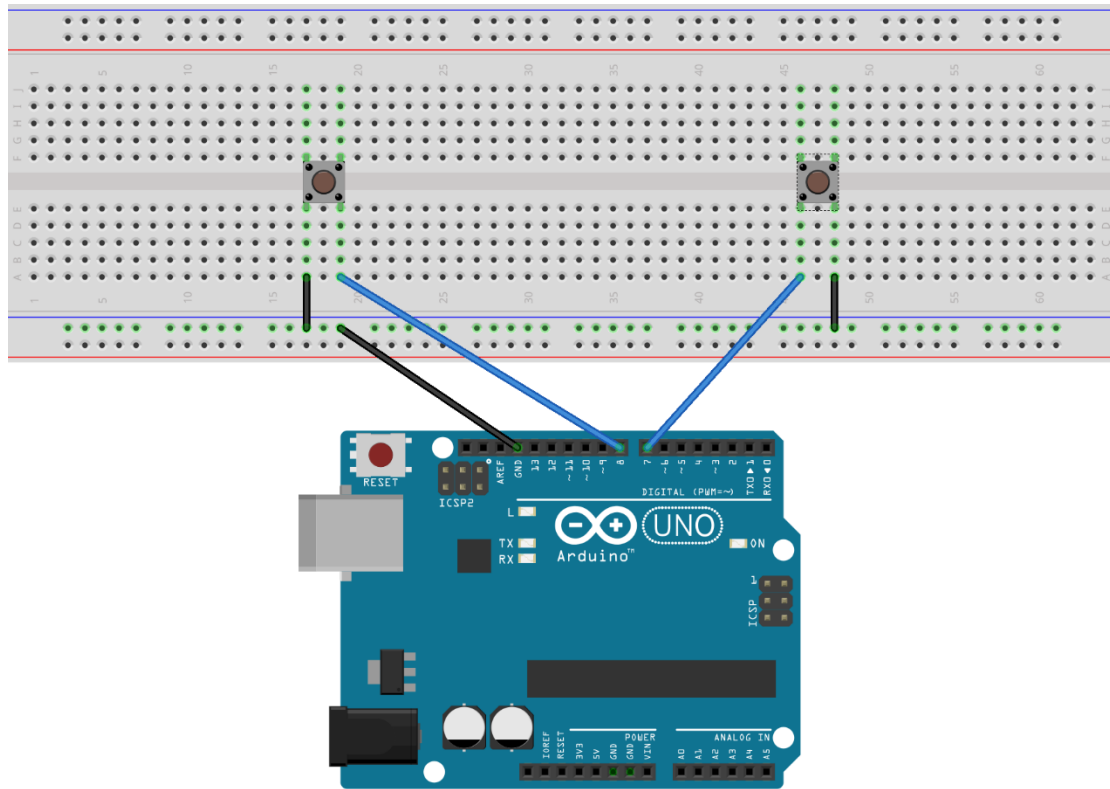
- ①When you click on a different icon ('Go', 'Play', 'back', 'Levels', 'No.1', 'No.2', 'No.3', 'Help' or 'Exit') with the mouse, you will enter the game's different interface.
- ②When you press the button on the right, the baffle moves to the right
- ③When you press the button on the left, the baffle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

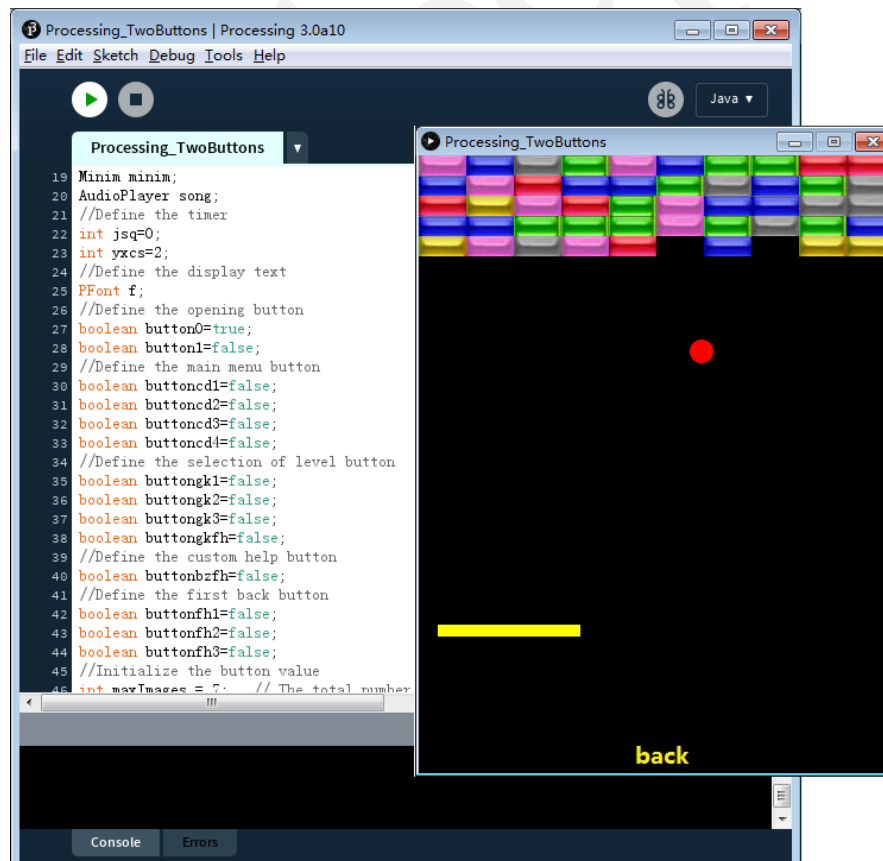
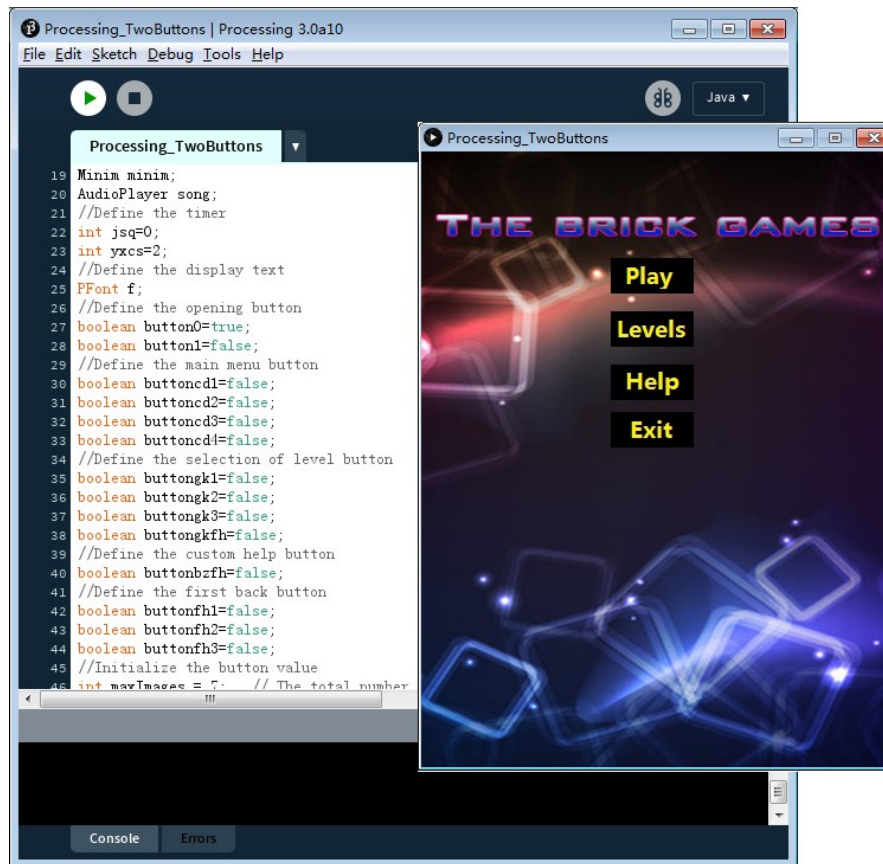
1. Build the circuit

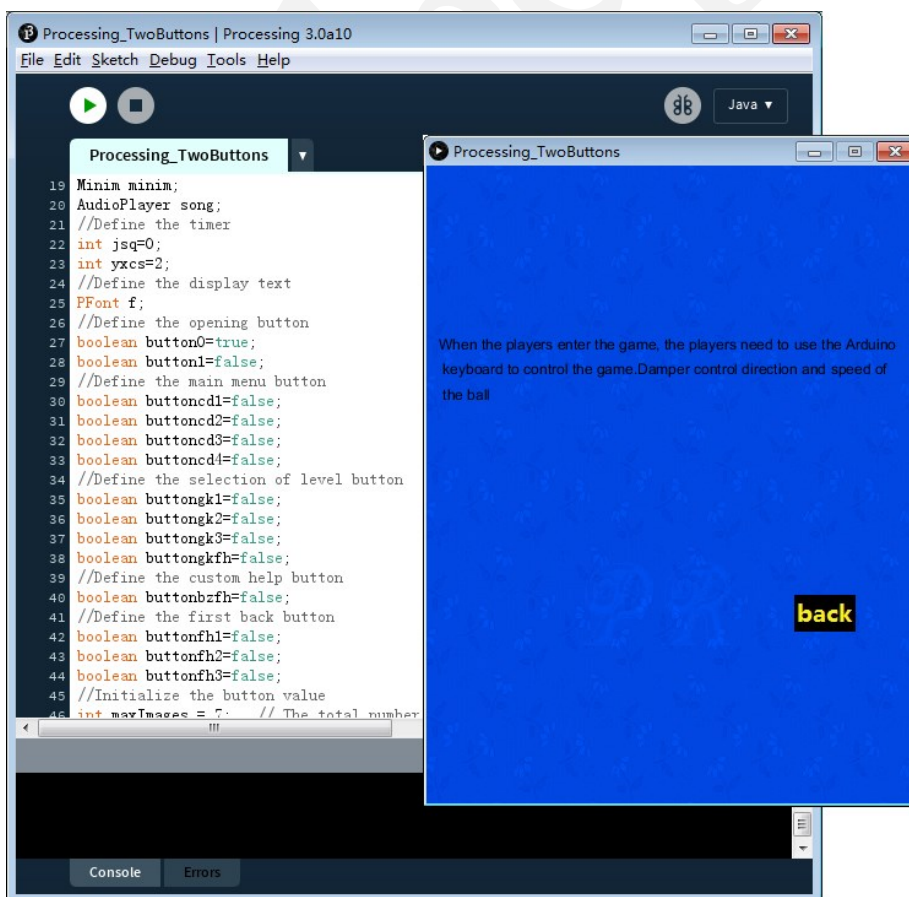
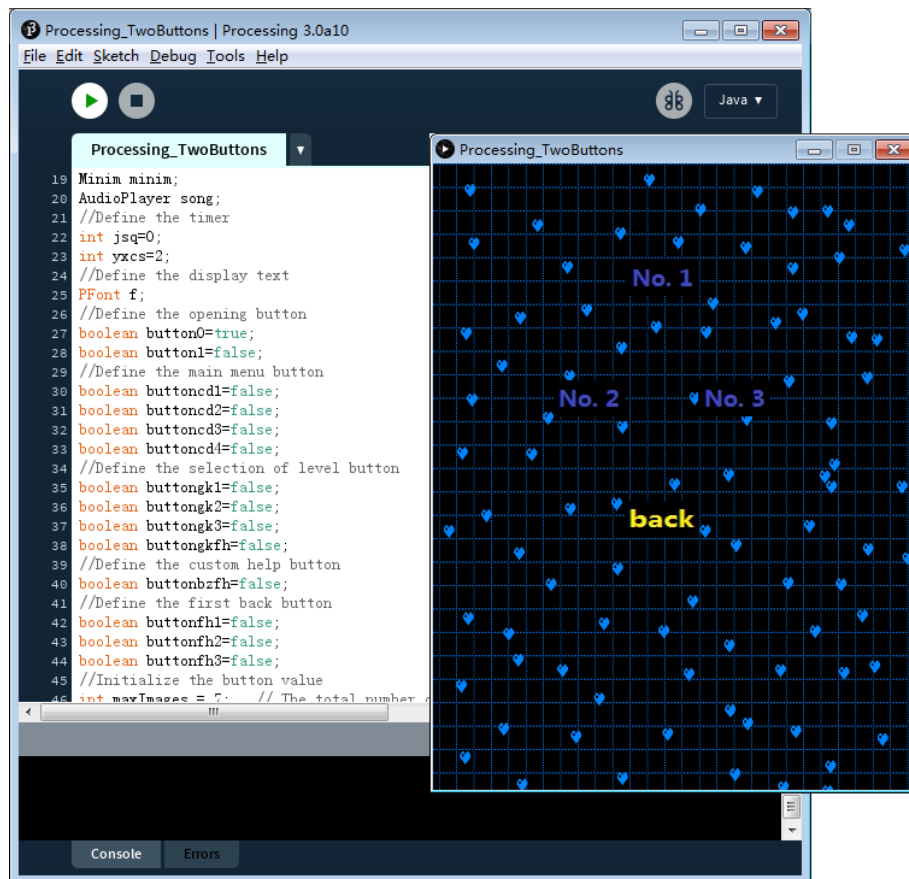


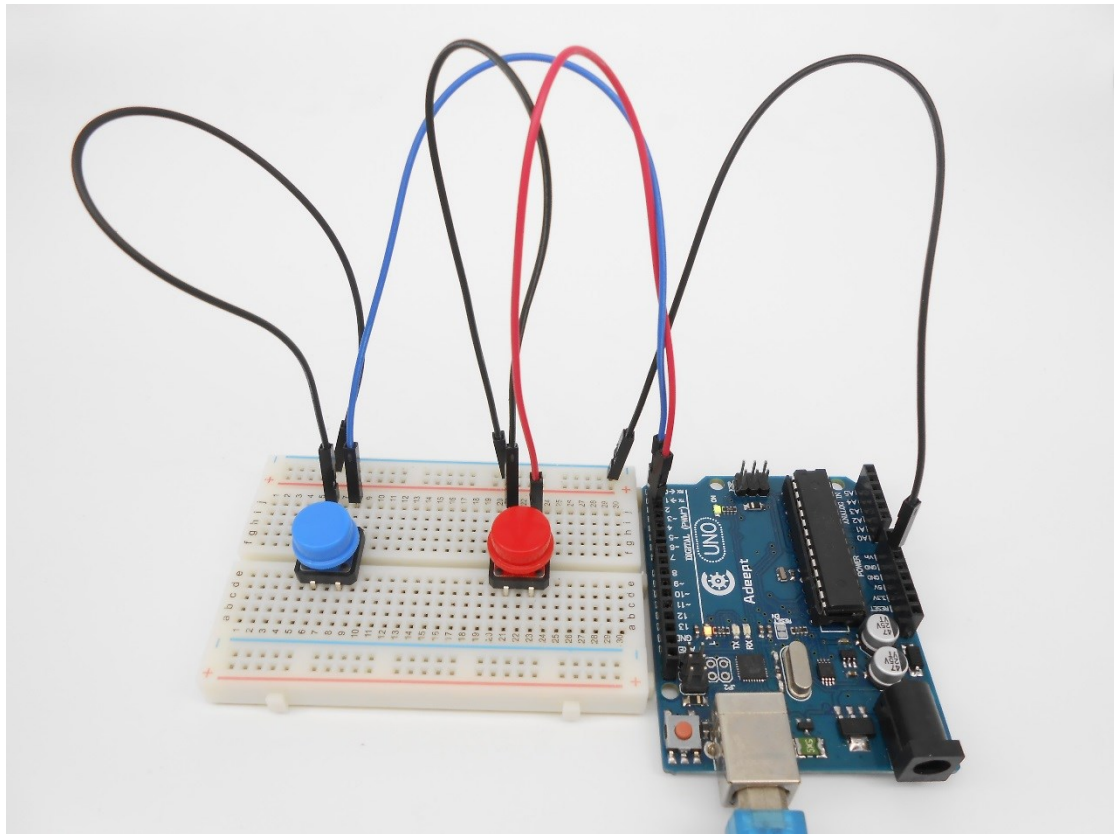
fritzing

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











Adept

Sharing Perfects Innovation

E-mail: support@adeept.com
website: www.adeept.com