SUPER STARTER KIT

BEGINNERS GUIDE

QUAD STORE
WWW.QUADSTORE.IN

Preface

Quad Store 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.

Contents

Getting Started with Arduino

What is an Arduino?

Arduino is an open-source physical computing platform designed to make experimenting with electronics more fun and intuitive. Arduino has its own unique, simplified programming language, a vast support network, and thousands of potential uses, making it the perfect platform for both beginner and advanced DIY enthusiasts.
www.arduino.cc

A Computer for the Physical World:

The friendly blue board in your hand (or on your desk) is the Arduino. In some ways you could think of Arduino as the child of traditional desktop and laptop computers. At its roots, the Arduino is essentially a small portable computer. It is capable of taking inputs (such as the push of a button or a reading from a light sensor) and interpreting that information to control various outputs (like a blinking LED light or an electric motor).
That's where the term "physical computing" is born - an Arduino is capable of taking the world of electronics and relating it to the physical world in a real and tangible way. Trust us - this will all make more sense soon.

Arduino UNO SMD R3

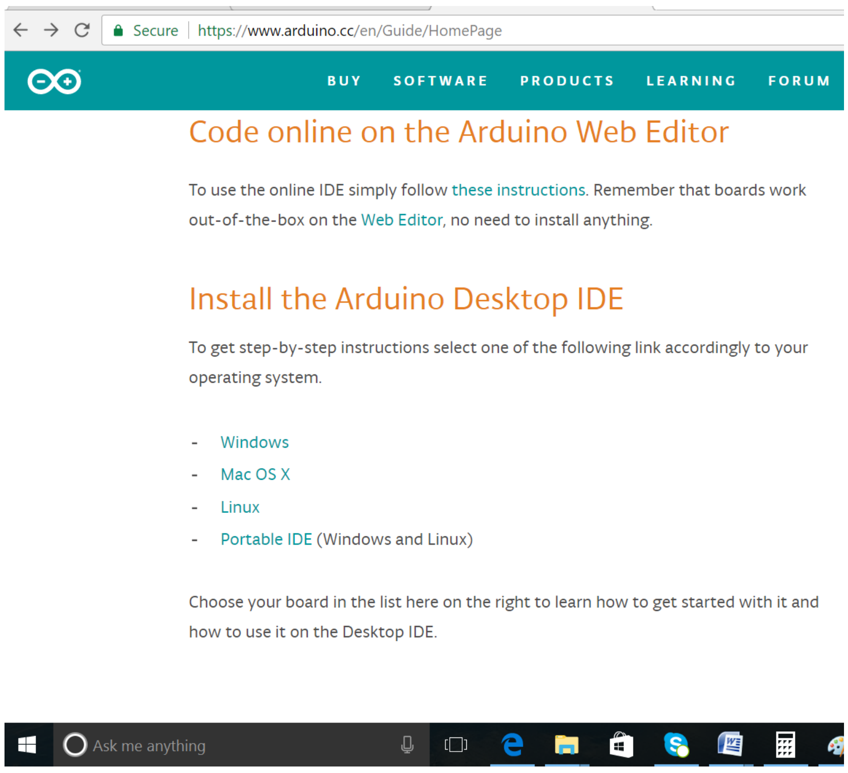
The Arduino Uno is one of several development boards based on the ATmega328. We like it mainly because of its extensive support network and its versatility. It has 14 digital input/output pins (6 of which can be PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. Don't worry, you'll learn about all these later.

Installing Arduino IDE and Using Uno R3 board

STEP-1: Download the Arduino IDE (Integrated Development Environment)

Access the Internet:
In order to get your Arduino up and running, you'll need to download some software first from www.arduino.cc (it's free!). This software, known as the Arduino IDE, will allow you to program the Arduino to do exactly what you want. It's like a word processor for writing programs. With an internet-capable computer, open up your favorite browser and type in the following URL into the address bar:

www.arduino.cc/en/Main/Software



Code online on the Arduino Web Editor

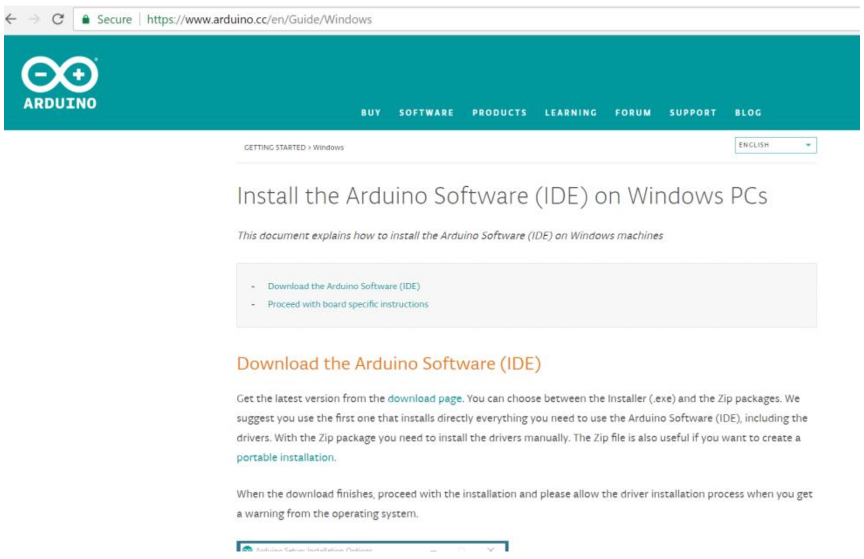
To use the online IDE simply follow these instructions. Remember that boards work out-of-the-box on the Web Editor, no need to install anything.

Install the Arduino Desktop IDE

To get step-by-step instructions select one of the following link accordingly to your operating system.

- Windows
- Mac OS X
- Linux
- Portable IDE (Windows and Linux)

Choose your board in the list here on the right to learn how to get started with it and how to use it on the Desktop IDE.


https://www.arduino.cc/en/Guide/Windows

GETTING STARTED > Windows

Install the Arduino Software (IDE) on Windows PCs

This document explains how to install the Arduino Software (IDE) on Windows machines

- Download the Arduino Software (IDE)
- Proceed with board specific instructions

Download the Arduino Software (IDE)

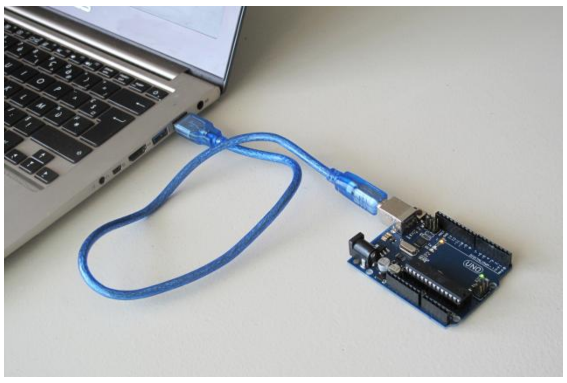
Get the latest version from the download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.

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

STEP-2: Connect your Arduino Uno to your Computer:

Use the USB cable provided in the kit to connect the Arduino to one of your computer's USB inputs.



STEP-3: Install Drivers

Depending on your computer's operating system, you will need to follow specific instructions. Please go to the URLs below for specific instructions on how to install the drivers onto your Arduino Uno.

Windows Installation Process:
Go to the web address below to access the instructions for installations on a Windows-based computer.

http://arduino.cc/en/Guide/Windows

Macintosh OS X Installation Process:
Macs do not require you to install drivers. Enter the following URL if you have questions. Otherwise proceed to next page.

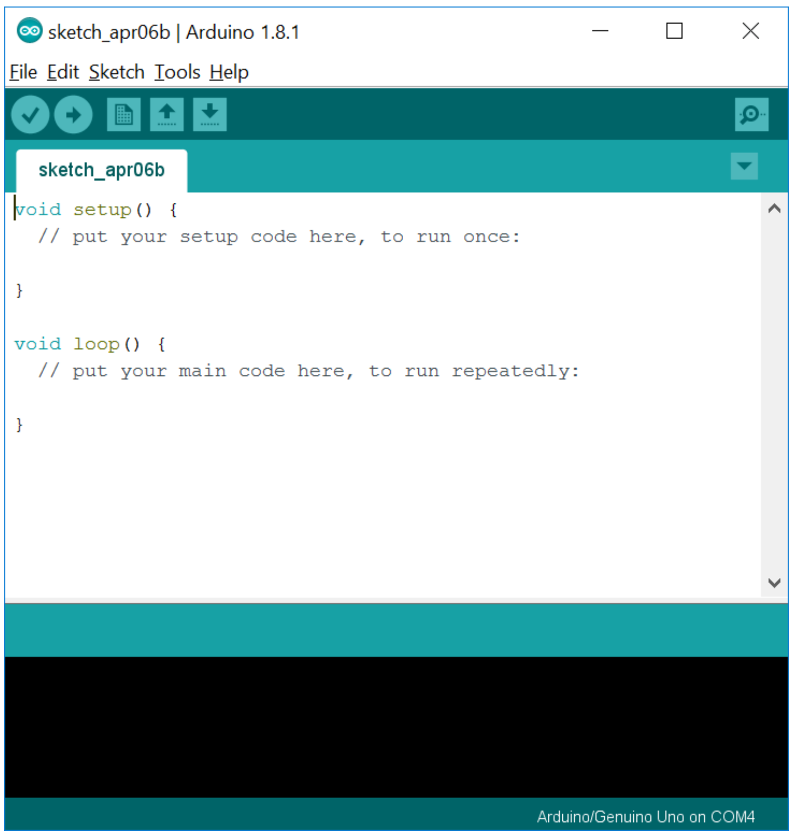
http://arduino.cc/en/Guide/MacOSX

Linux:
32 bit / 64 bit, Installation Process Go to the web address below to access the instructions for installations on a Linux-based computer.

http://www.arduino.cc/playground/Learning/Linux

STEP-4: Open the Arduino IDE

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away, this is just an introduction. The step is to set your IDE to identify your Arduino Uno.


sketch_apr06b | Arduino 1.8.1
File Edit Sketch Tools Help
sketch_apr06b
Arduino/Genuino Uno on COM4

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

}

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

}
```

GUI (Graphical User Interface)

Verify
Checks your code for errors compiling it.

Upload
Compiles your code and uploads it to the configured board. See uploading below for details.
Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

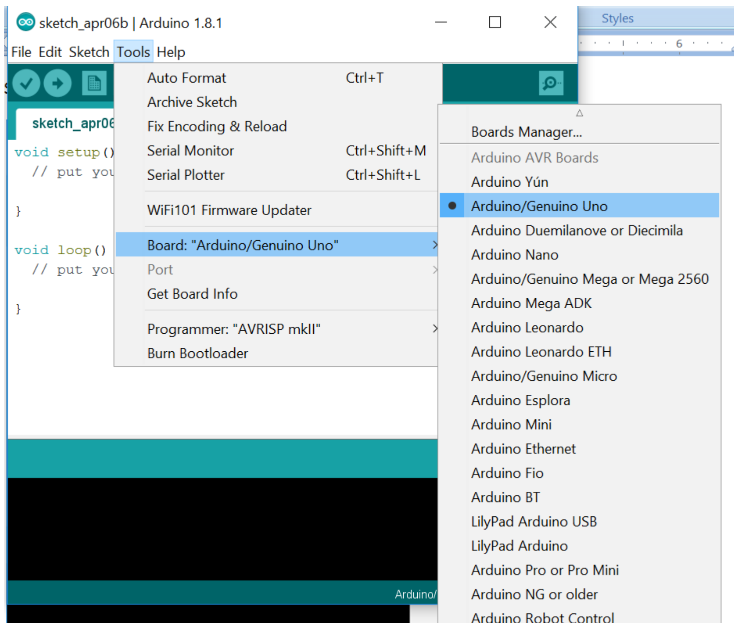
New
Creates a new sketch.

Open
Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.
Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File | Sketchbookmenu instead.

Save
Saves your sketch.

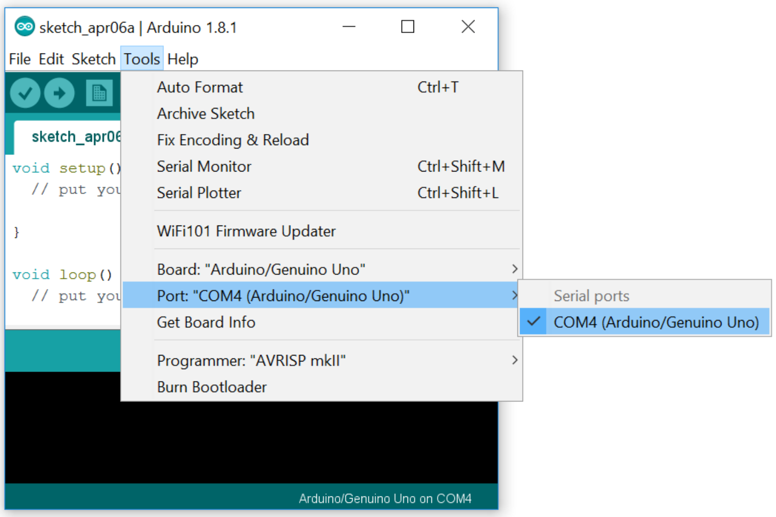
Serial Monitor
Opens the serial monitor.

STEP-5: Select your board: Arduino Uno



STEP-6: Select your Serial Device

Windows: Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be com3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.


sketch_apr06a | Arduino 1.8.1
File Edit Sketch Tools Help
sketch_apr06
void setup()
// put you
}
void loop()
// put you
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM4 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM4 (Arduino/Genuino Uno)
Arduino/Genuino Uno on COM4

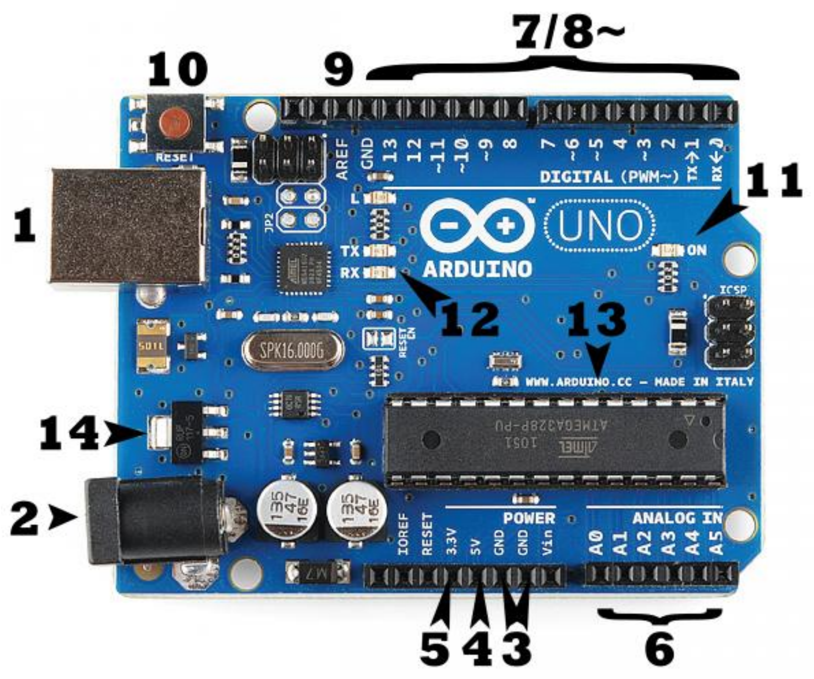
Mac OS: Select the serial device of the Arduino board from the Tools > Serial Port menu. On the Mac, this should be something with /dev/tty.usbmodem (for the Uno or Mega 2560) or /dev/tty.usbserial (for older boards) in it.

Linux: http://playground.arduino.cc/Learning/Linux

About Arduino Uno R3 board

What's on the board?

There are many varieties of Arduino boards that can be used for different purposes. Some boards look a bit different from the one below, but most Arduino have the majority of these components in common:



Power (USB / Barrel Jack)

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply that is terminated in a barrel jack. In the picture above the USB connection is labeled (1) and the barrel jack is labeled (2).

.

NOTE: Do NOT use a power supply greater than 20 Volts as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.
Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjuction with a breadboard and some wire. They usually have black plastic 'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- **GND** **(3):** Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- **5V** **(4)** **& 3.3V** **(5):** As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
- **Analog** **(6):** The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- **Digital** **(7):** Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
- **PWM** **(8):** You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM). We have a tutorial on PWM, but for now, think of these pins as being able to simulate analog output (like fading an LED in and out).
- **AREF** **(9):** Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset Button

Just like the original Nintendo, the Arduino has a reset button **(10).** Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code doesn't repeat, but you want to test it multiple times. Unlike the original Nintendo however, blowing on the Arduino doesn't usually fix any problems.

Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there's a tiny LED next to the word 'ON' **(11).** This LED should light up whenever you plug your Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong. Time to re-check your circuit!

TX RX LEDs

TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear – once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs **(12).** These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we're loading a new program onto the board).

Main IC

The black thing with all the metal legs is an IC, or Integrated Circuit **(13).** Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC. If you want to know more about the difference between various IC's, reading the datasheets is often a good idea.

Voltage Regulator

The voltage regulator **(14)** is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says – it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don't hook up your Arduino to anything greater than 20 volts.

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

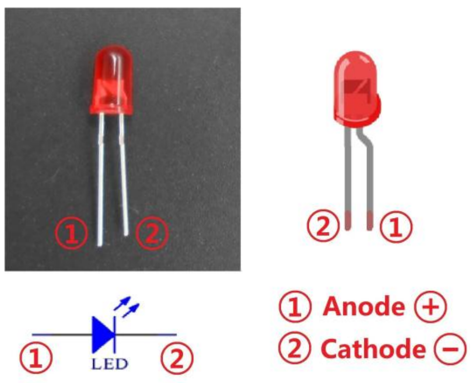
Principle:

In this lesson, we will program the Arduino's GPIO output high 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.



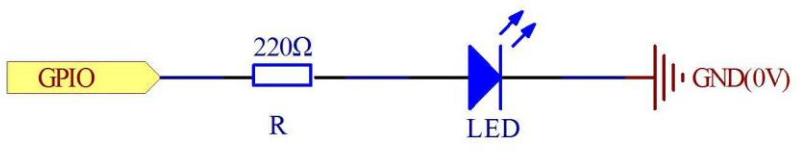
2. What is resistor?

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(Ω).

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



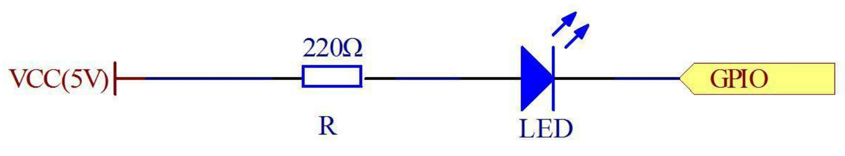
As shown in the schematic diagram, the anode of the LED is connected to Arduino's GPIO via a resistor, and the cathode to the ground (GND). When the GPIO outputs high level, the LED is on; when it outputs low, the LED is off.

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:

$$R = U / I = 5V / (5\sim20mA) = 250\Omega\sim1k\Omega$$

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.

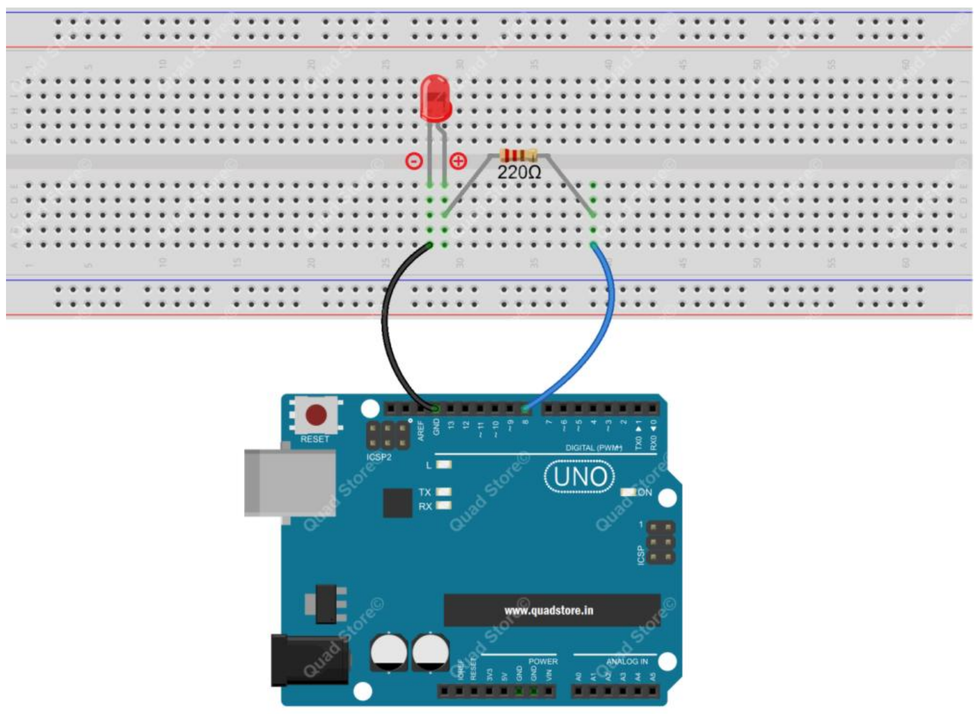
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

●delay()

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

Procedure:

Step 1: Build the circuit as below:



Step 2: Program : You can copy paste the below program in the IDE or open the code directly from the "CODE" folder that comes with the DVD/from the downloaded Zip folder.

```
/**********************************************************
File name:   01_blinkingLed.ino
Description:  Lit LED, let LED blinks.
Website: www.quadstore.in
**********************************************************/
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
}
```

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

Lesson 2 - 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.

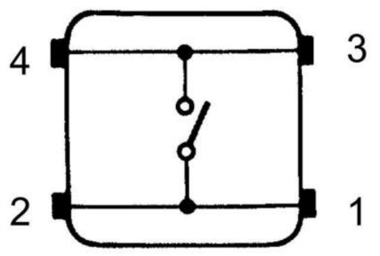
Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Button
- 1 * LED
- 1 * 10kΩ Resistor
- 1 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

Principle

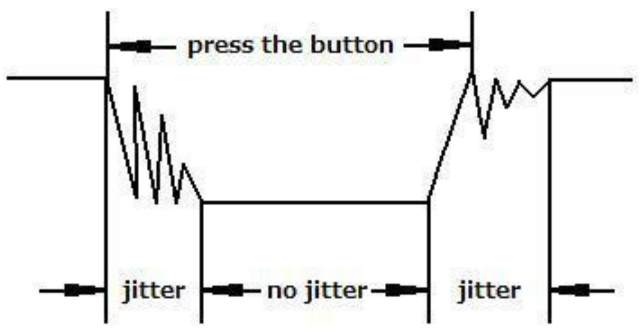
1. Button

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.

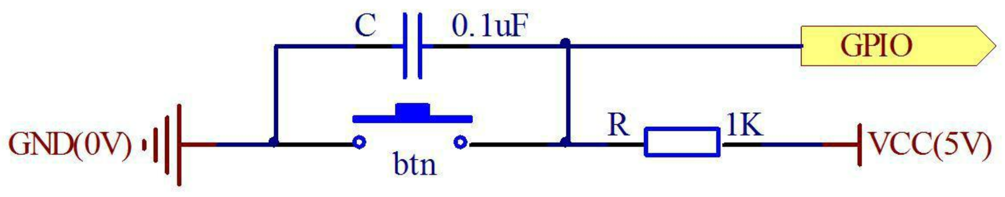


The button we used is a normally open type one. The two contacts of a button are in the off state under the normal conditions; only when the button is pressed they are closed.

The 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

. 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.

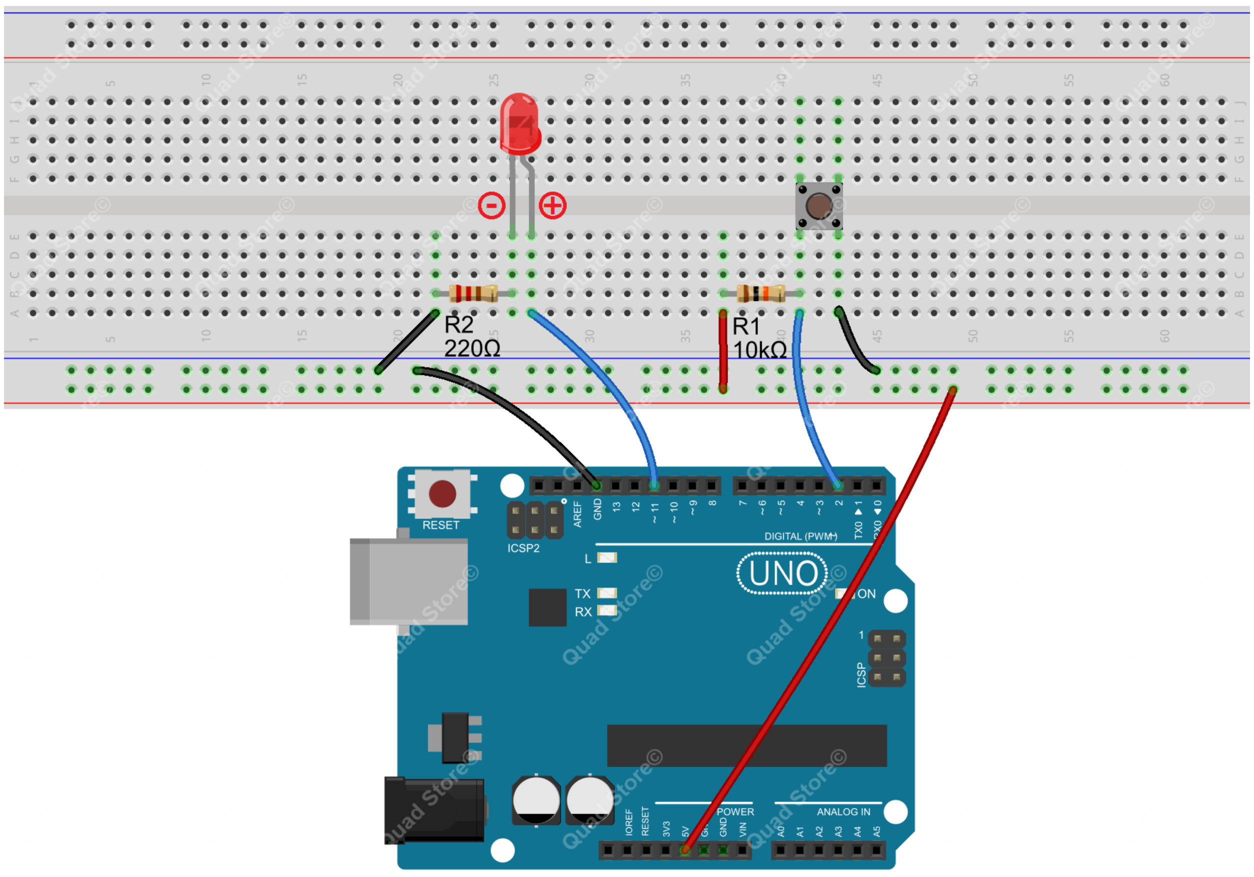
Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax:
delayMicroseconds(us)
Parameters:

us: the number of microseconds to pause (unsigned int) Returns:
None

Procedure:

Step 1: Build the circuit



Step 2: Program: Open /Copy the code from the “CODE” Folder

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

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

Lesson 3 - 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 receives 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 * 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()

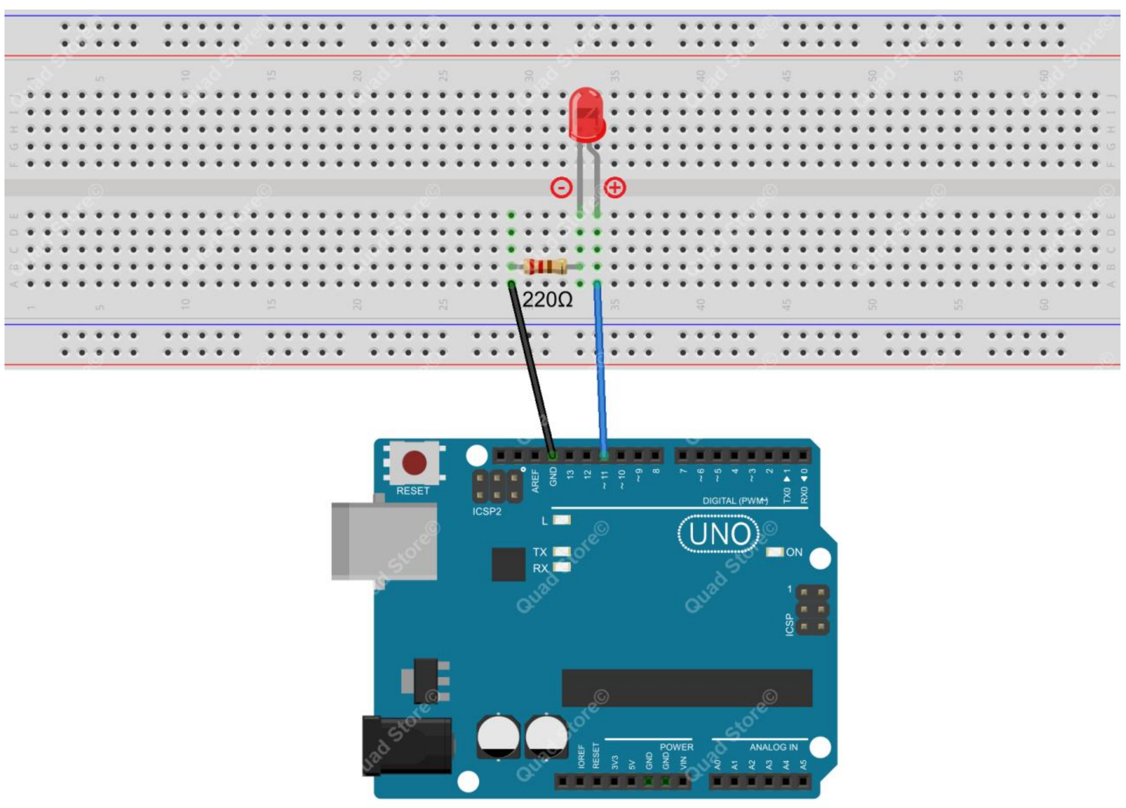
Reads incoming serial data. read() inherits from the Stream utility class. Syntax:
Serial.read()

Parameters: None

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

Procedure:

Step 1: Build the circuit

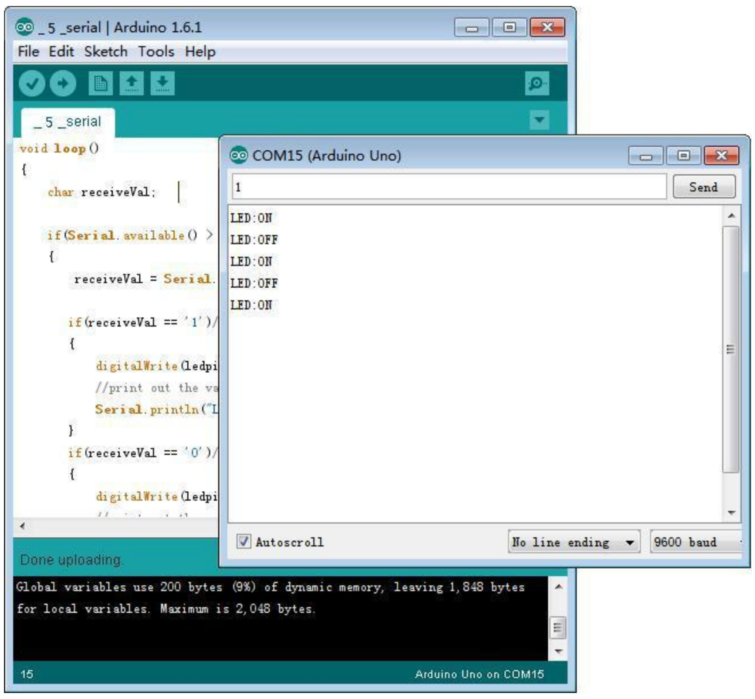


Step 2: Program

Step 3: Compile the program and upload to Arduino 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.



```
void loop()
{
   char receiveVal;

  if(Serial.available() >
   {
      receiveVal = Serial.

     if(receiveVal == '1')/
     {
         digitalWrite(ledpi
         //print out the va
         Serial.println("L
     }
     if(receiveVal == '0')/
     {
         digitalWrite(ledpi
```

```
1
```

```
LED:ON
LED:OFF
LED:ON
LED:OFF
LED:ON
```

```
Global variables use 200 bytes (9%) of dynamic memory, leaving 1,848 bytes
for local variables. Maximum is 2,048 bytes.
```

Lesson 4 - 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 * Arduino UNO
- 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.

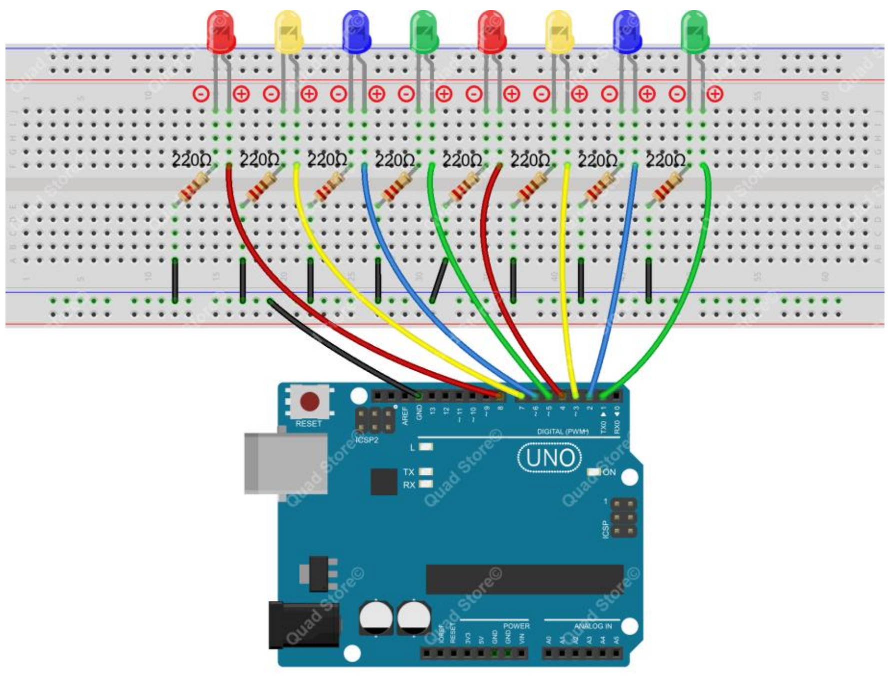
There are three parts to the for loop header:

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

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.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino 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.

Lesson 5 - 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 * 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 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()

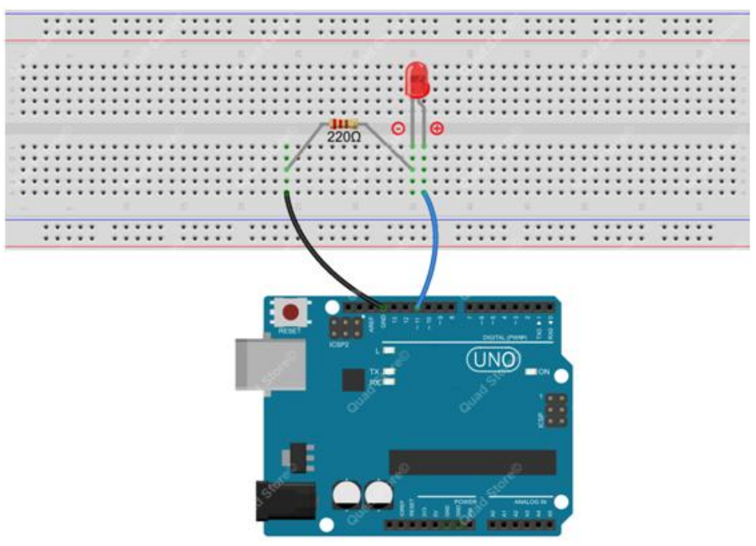
Writes an analog value (PWM wave) to a pin. Can be used to light an LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or 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

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino 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.

Lesson 6 - 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 "QuadStore" and “www.quadstore.in” statically.

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * LCD1602
- 1 * 10kΩ 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()

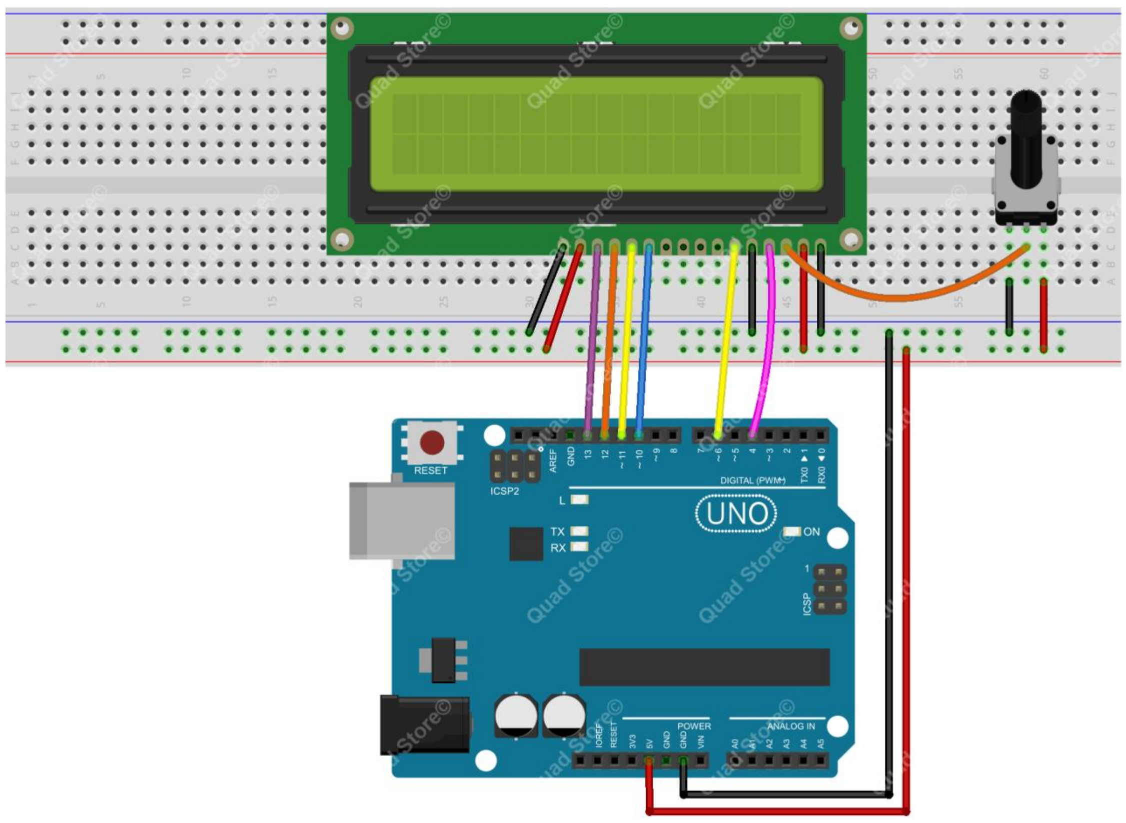
Clears the LCD screen and positions the cursor in the upper-left corner. Syntax:
lcd.clear()

Parameters:

lcd: a variable of type LiquidCrystal

Procedure:

Step 1: Build the circuit



Step 2: Program

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

Now, you can see the string "Hello Geeks!" shown on the LCD1602 scrolling, and then the string "QuadStore" and "www.quadstore.in" displayed statically.

Lesson 07: 7-segment Display

Overview

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

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * 220Ω Resistor
- 1 * 7-segment Display
- 1 * Breadboard
- Several jumper wires

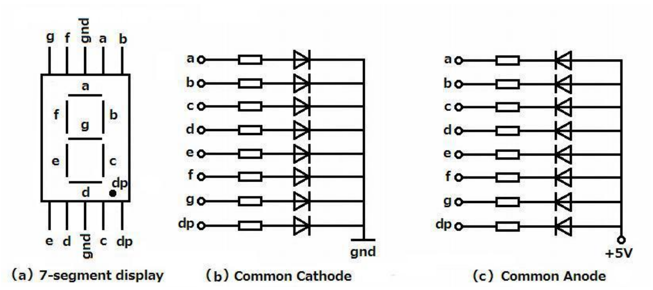
Principle

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

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

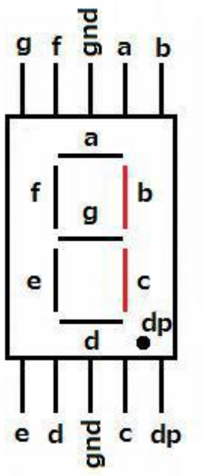
The seven-segment display is an 8-shaped LED display device composed of eight LEDs (including a decimal point). The segments respectively named a, b, c, d, e, f, g, and dp.

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



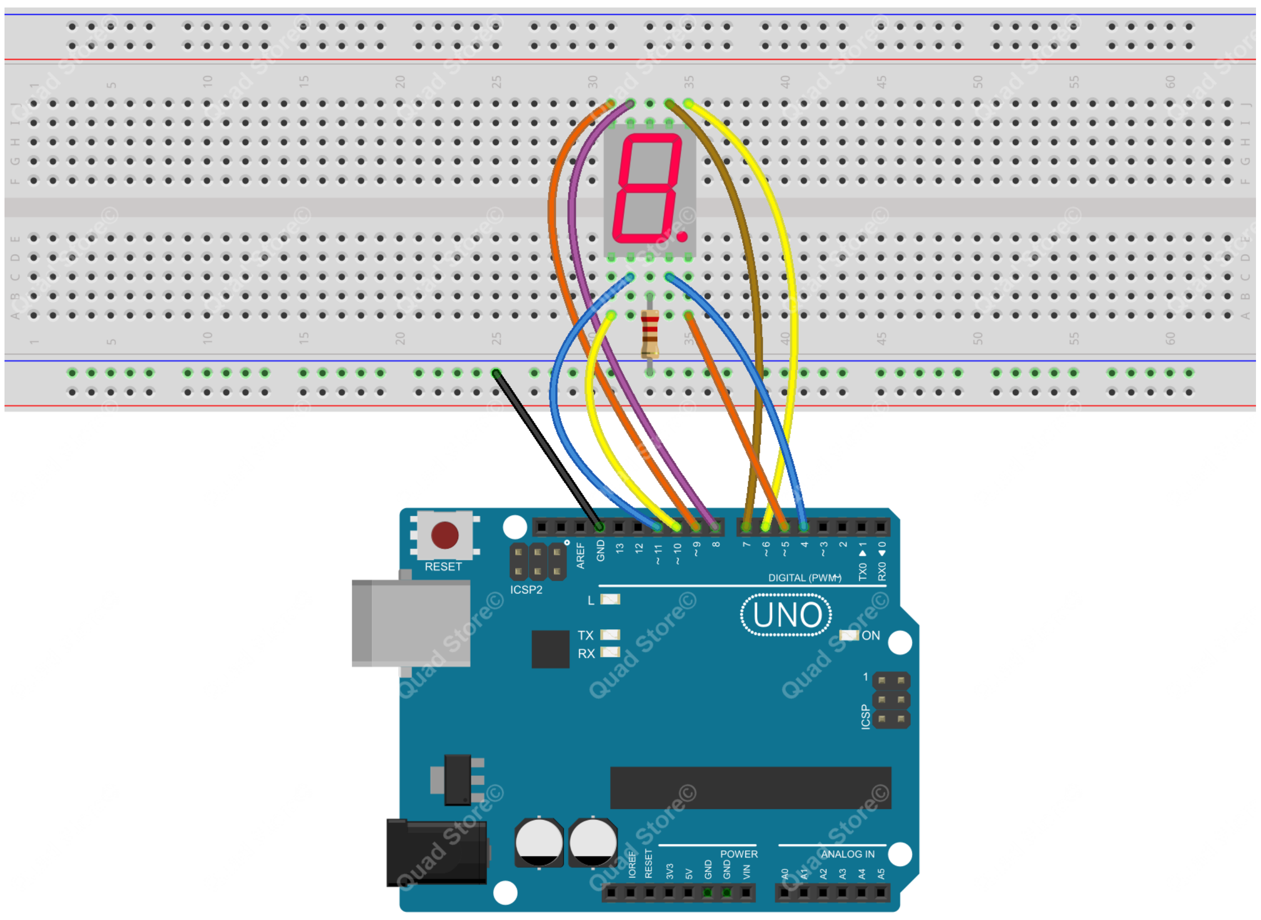
For a common-anode LED, the common anode should be connected to the power supply (VCC); for a common-cathode LED, the common cathode should be connected to the ground (GND).
Each segment of a segment display is composed of an LED, so a resistor is needed for protecting the LED.

A 7-segment display has seven segments for displaying a figure andone 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.



Procedure:

Step 1: Build the circuit



Step 2: Program

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

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

Lesson 08 - A Simple Counter

Overview

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

Components

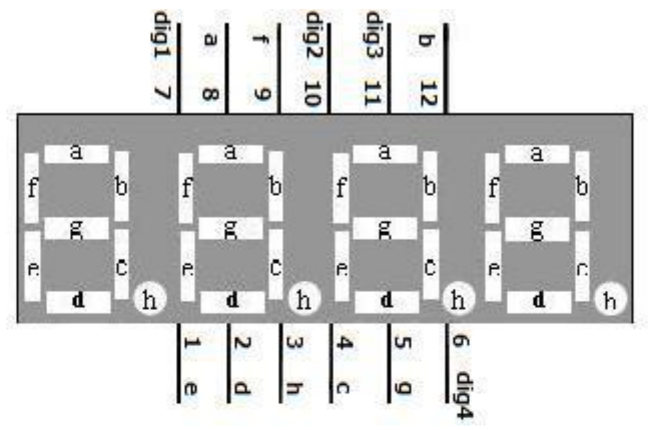
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * 4-digit 7-segment Display
- 5 * 220Ω Resistor
- 3 * 220Ω Resistor
- 2 * Button
- 1 * Breadboard
- Several jumper wires

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.

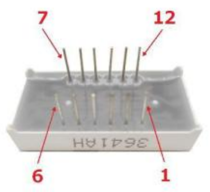
4-digit segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

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



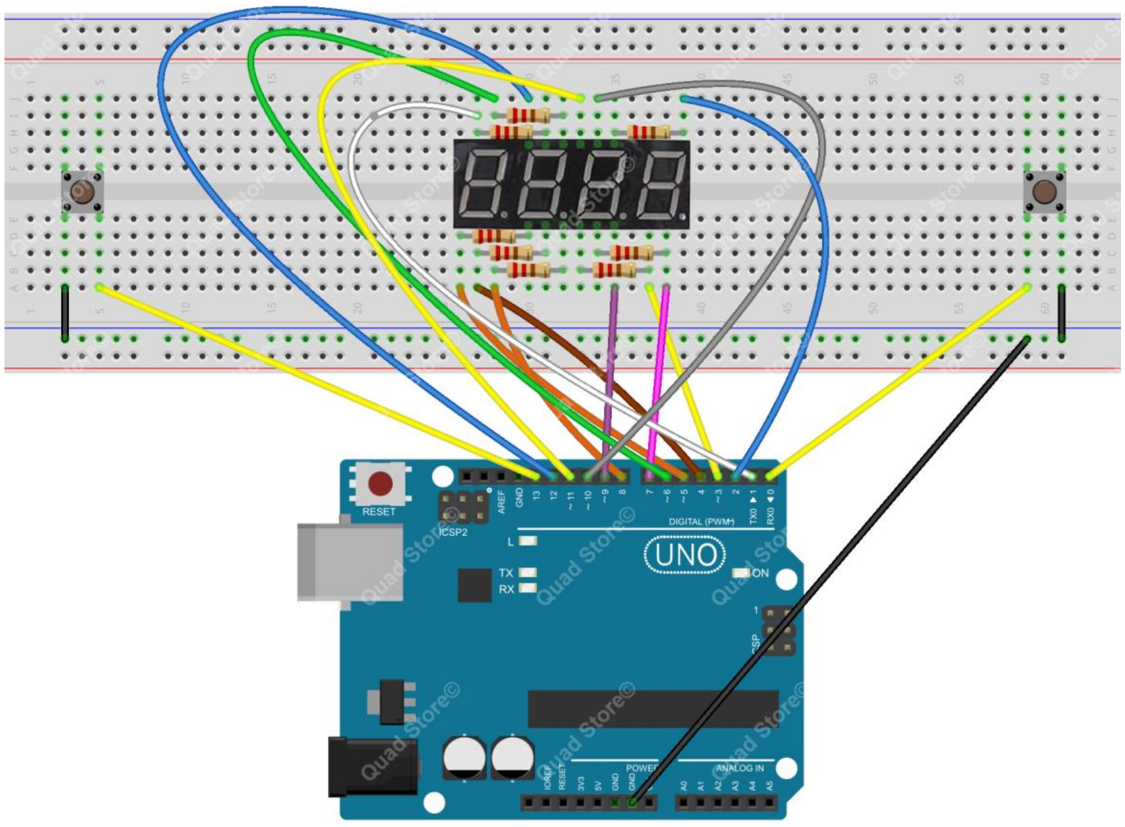
What we use in this experiment is a common cathode 4-digit 7-segment display. Its internal structure is as shown below:

The pin number is as follows:



Procedure:

Step 1: Build the circuit



Step 2: Program

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

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

Lesson 09 - 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 * Arduino UNO
- 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

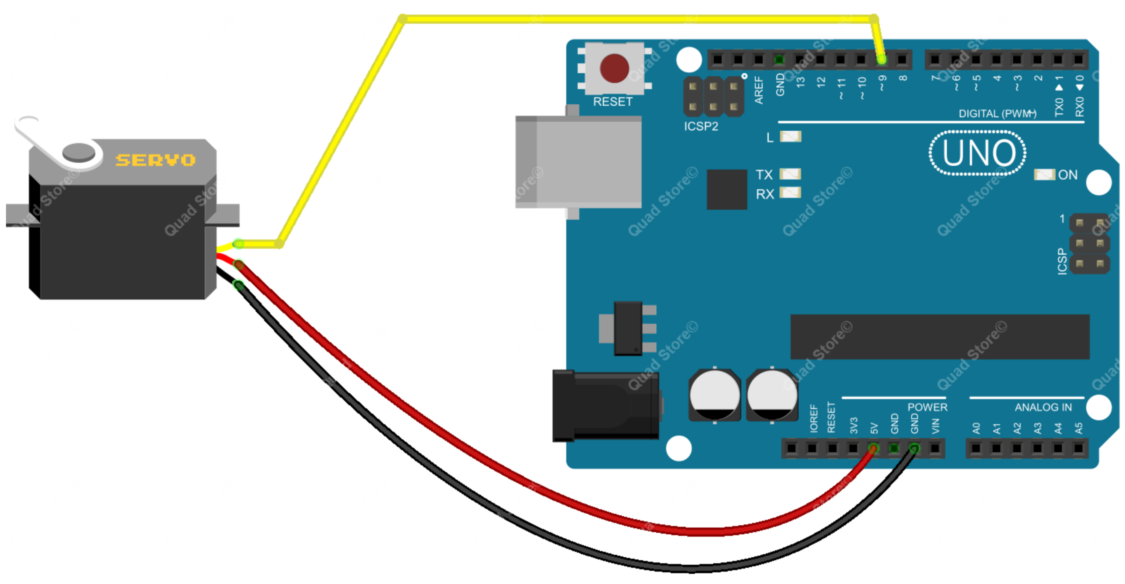
pin: the number of the pin that the servo is attached to

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)

Procedure:

Step 1: Build the circuit



Step 2: Program

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

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

Lesson 10 - IR Remote Controller

Overview

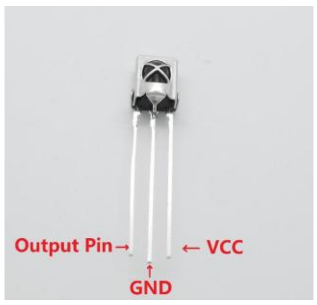
In this lesson, we will learn how to use an IR receiver to receive signals from a remote controller.

Components

- 1 * Arduino UNO
- 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. There could be different types of remote as shown below but their functionality is the same.

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

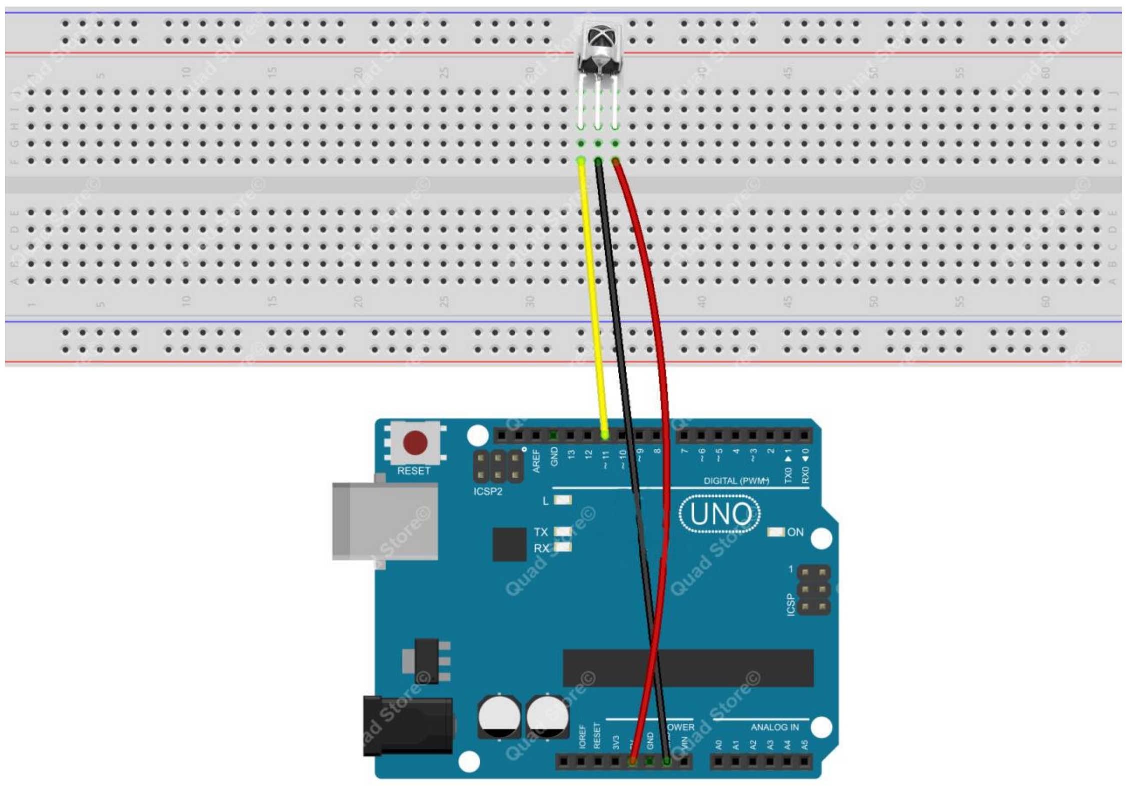
and

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

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

Procedure:

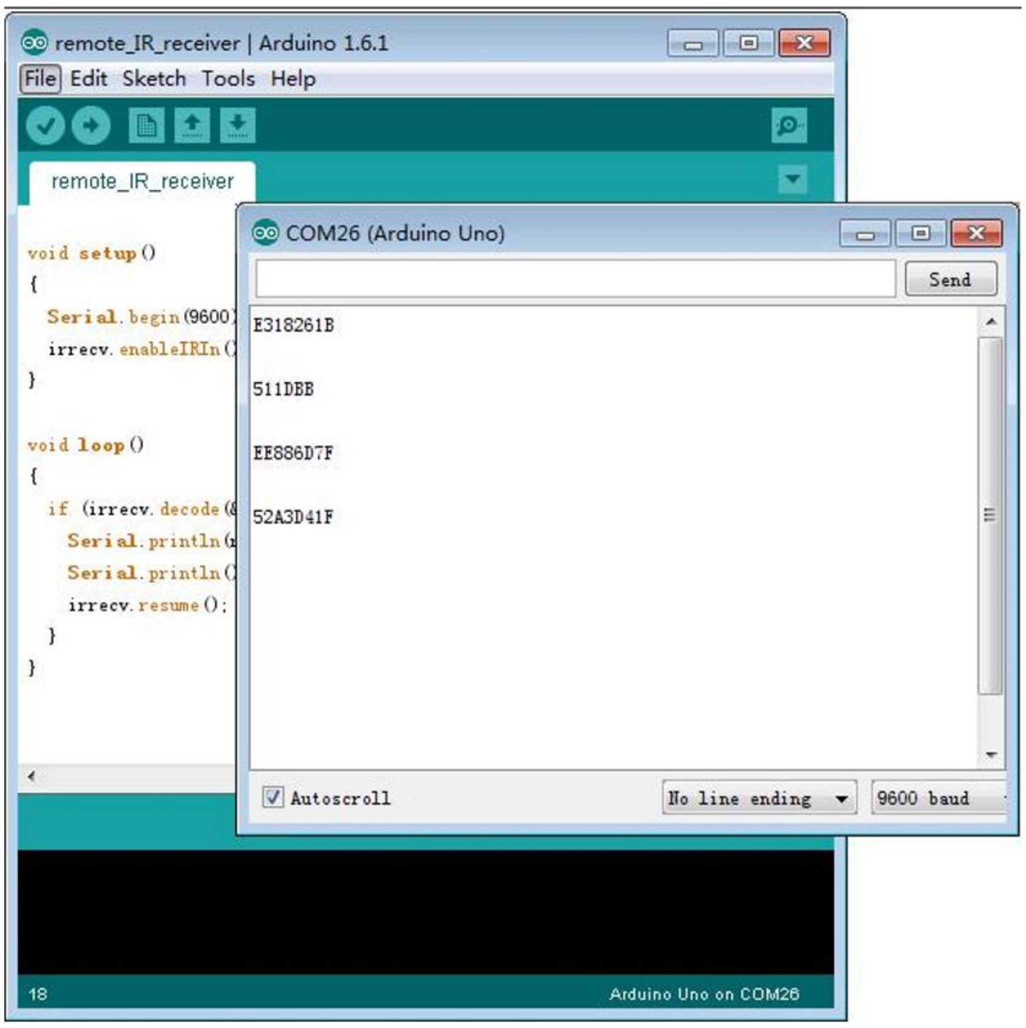
Step 1: Build the circuit



Step 2: Program

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

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


remote_IR_receiver | Arduino 1.6.1
File Edit Sketch Tools Help
remote_IR_receiver
COM26 (Arduino Uno)
Send

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

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

```
E318261B

511DBB

EE886D7F

52A3D41F
```

☑ Autoscroll   No line ending   9600 baud

18   Arduino Uno on COM26

Lesson 11 - Ultrasonic Distance Sensor

Overview

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

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Ultrasonic Distance Sensor
- 1 * LCD1602
- 1 * 10kΩ 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:

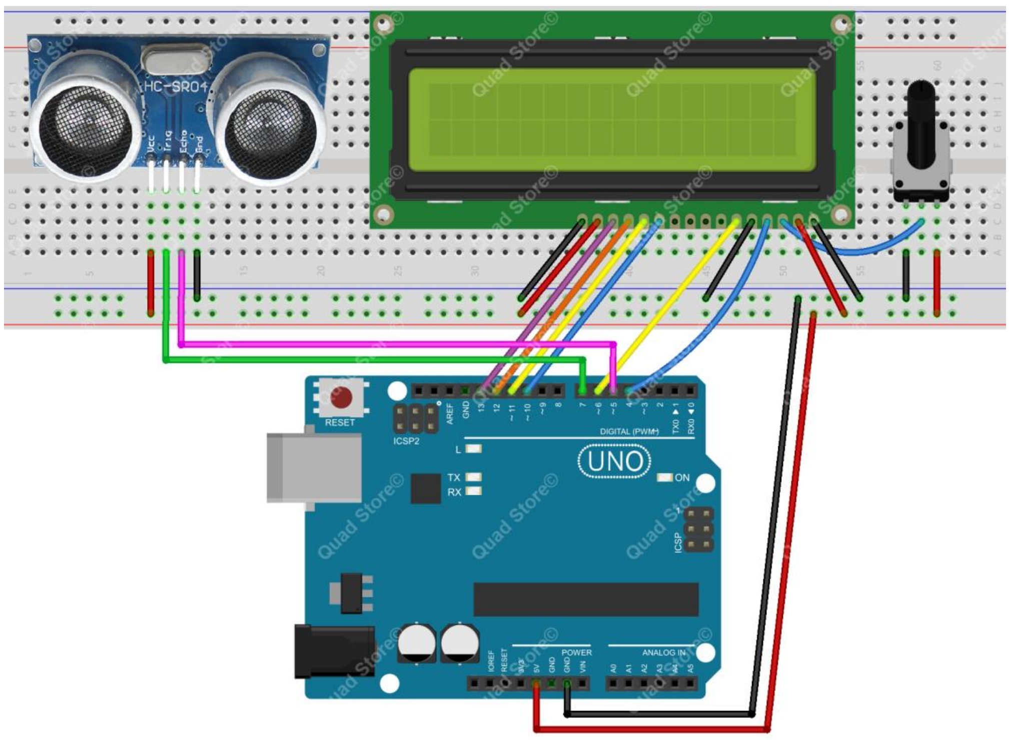
RoundTrip = microseconds / 29.

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

microseconds / 29 / 2

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino 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 12 - 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.

Components

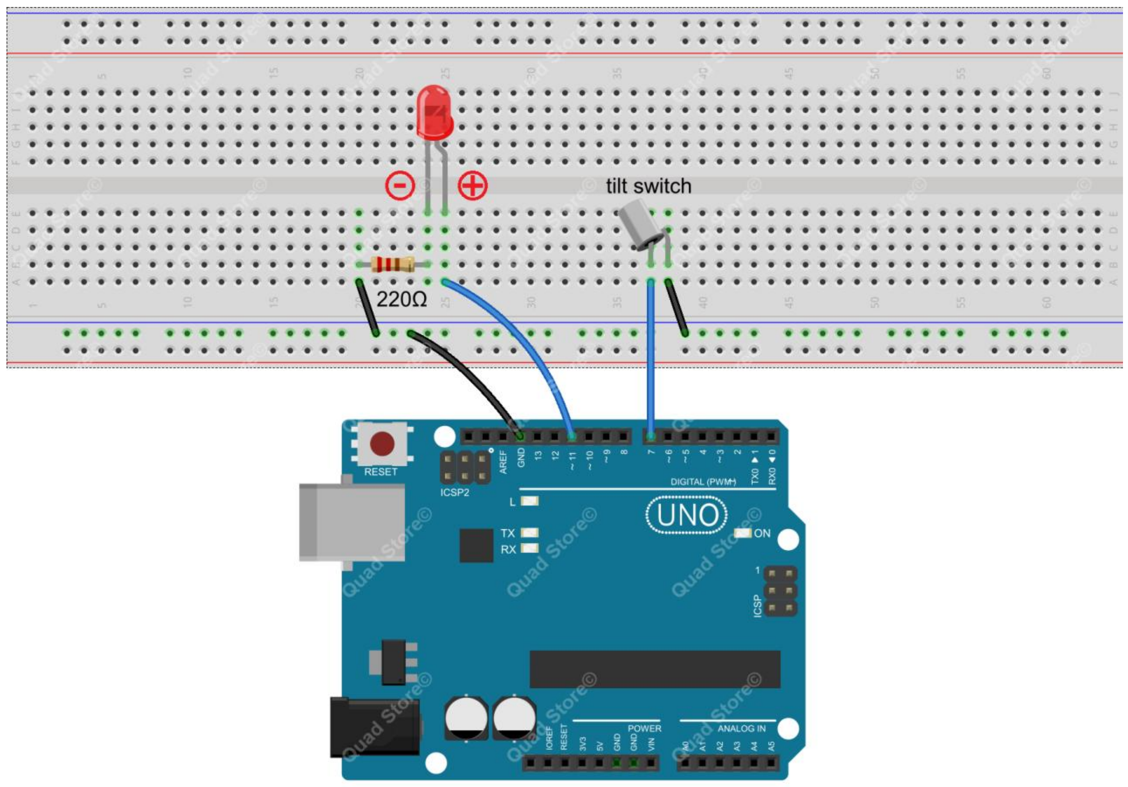
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Tilt Switch (SW-520D)
- 1 * LED
- 1 * 220Ω Resistor
- 1 * Breadboard
- Several jumper wires

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.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 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.

Lesson 13 - Controlling Stepper Motor

Overview

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

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Stepper Motor
- 1 * ULN2003 Driver Board
- Several jumper wires

Principle

1. Stepper motors

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

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

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

2. ULN2003 driver board

Arduino UNO R3 board cannot directly drive stepper motors. A driver circuit is necessary, so we choose an ULN2003 driver board here as shown below. There are four LEDs on the top. The white booth in the middle is connected to the

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

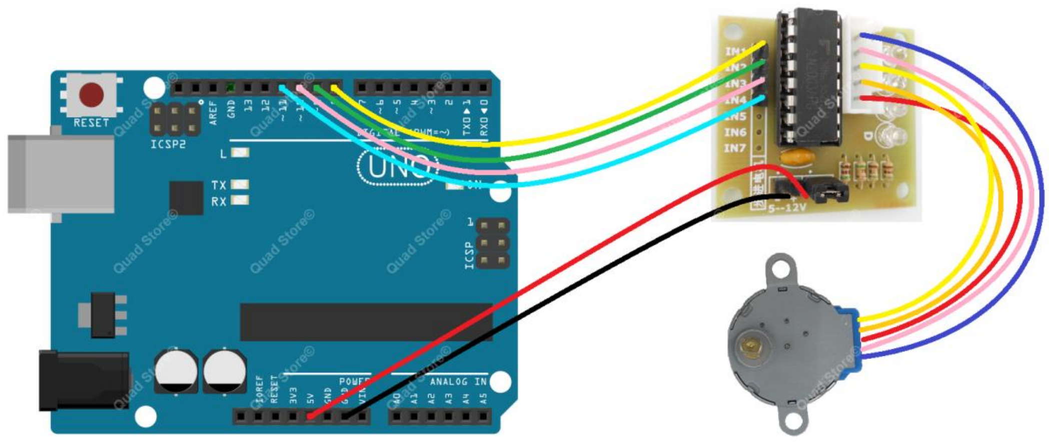
How many beats needed for the shaft to take a turn？

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

Procedure:

Step 1: Build the circuit

ULN2003 driver board IN1-----------------------	Arduino UNO R3 Digital pin 8
ULN2003 driver board IN2-----------------------	Arduino UNO R3 Digital pin 9
ULN2003 driver board IN3-----------------------	Arduino UNO R3 Digital pin 10
ULN2003 driver board IN4-----------------------	Arduino UNO R3 Digital pin 11
ULN2003 driver board "+" -----------------------	Arduino UNO R3 Power pin 5V
ULN2003 driver board "-"-----------------------	Arduino UNO R3 Power pin GND



Step 2: Program

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

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

Lesson 14 - 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

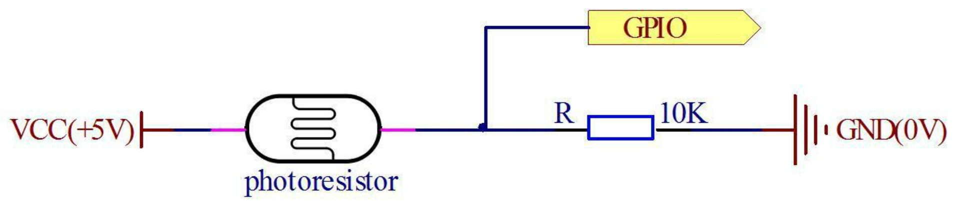
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * LCD1602
- 1 * Photoresistor
- 1 * 10kΩ Resistor
- 1 * 10kΩ Potentiometer
- 1 * Breadboard
- Several jumper wires

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 (MΩ), 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.

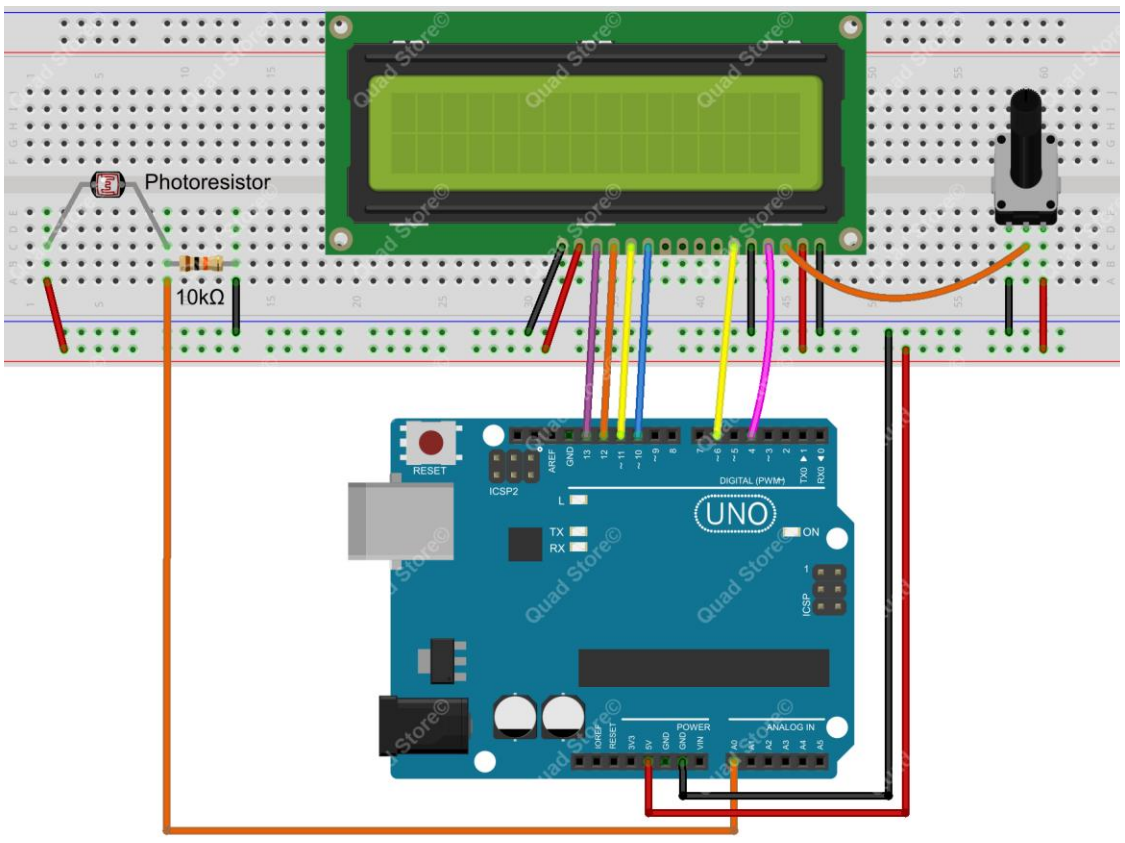
The schematic diagram of this experiment is shown below:



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.

Procedure:

Step 1: Build the circuit



Step 2: Program

Step 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.

Lesson 15 – LM35 Temperature Sensor

Overview:

In this tutorial, we will learn how to measure the temperature using LM35 temperature sensor.

Components:

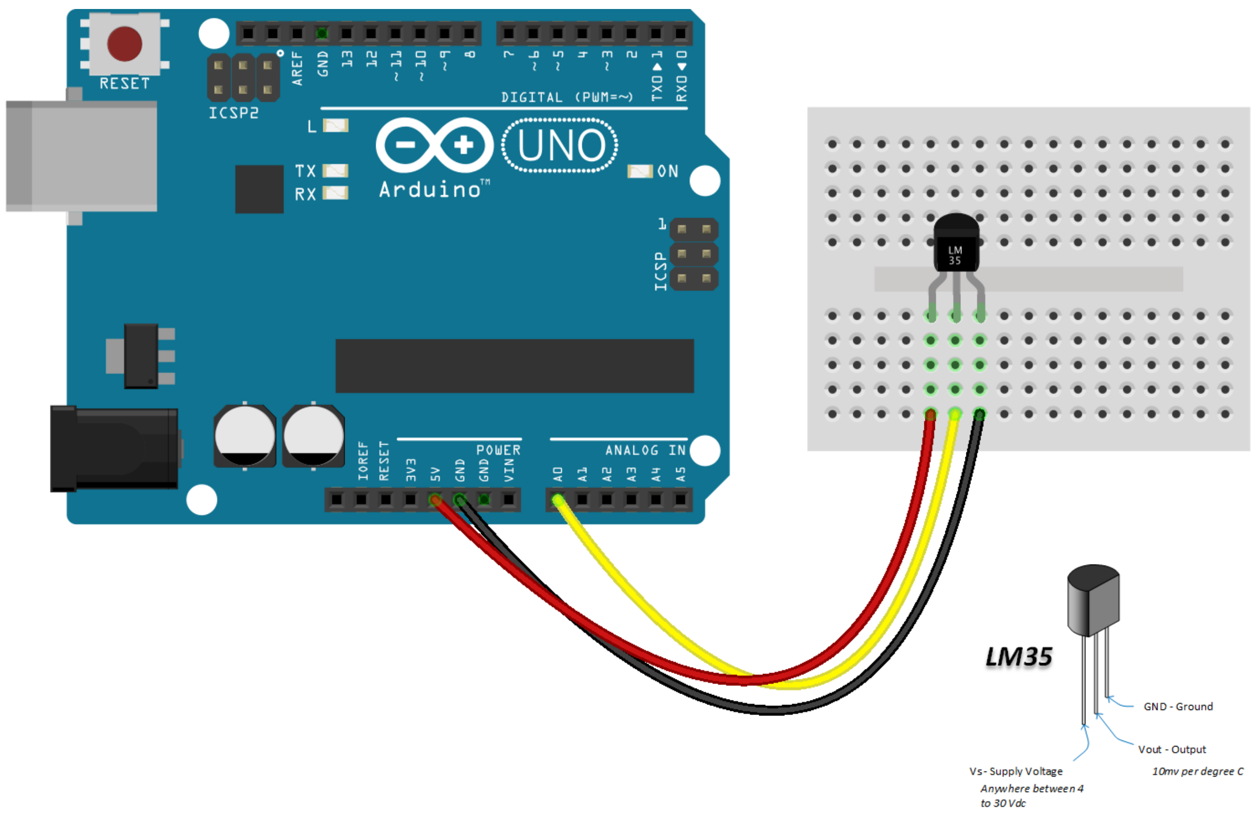
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * LM35 (Temperature Sensor)
- 1 * Breadboard
- 3 * Jumper Wires

Principle:

In this tutorial, we will learn how to measure the temperature using LM35 temperature sensor.

Procedure:

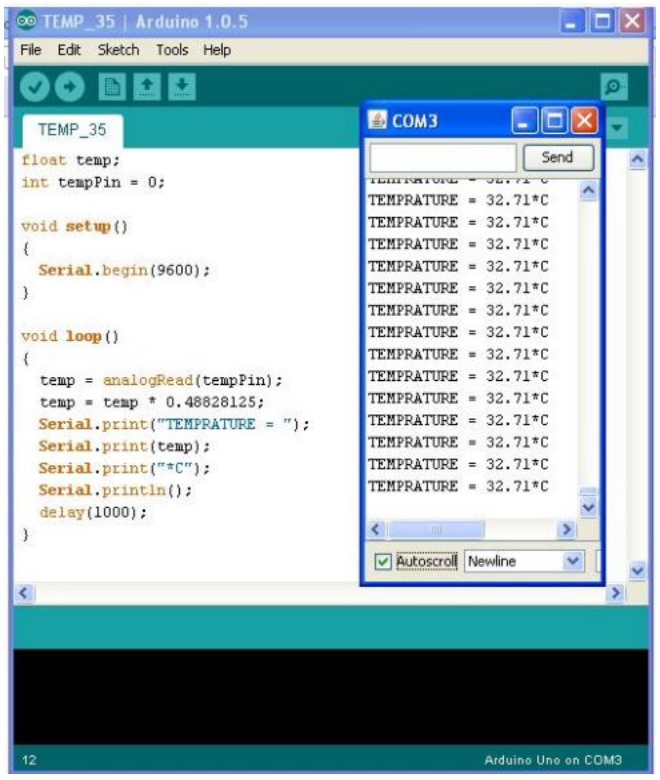
Step 1: Build the circuit as below:



Step 2: Program : You can copy paste the below program in the IDE or open the code directly from the “CODE” folder that comes with the DVD/from the downloaded Zip folder.

```
/*************************************************************
File name:   01_lm35temp.ino
Description:  Lit LED, let LED blinks.
Website: www.quadstore.in
*************************************************************/
int val;
int tempPin = 1;

void setup()
{
Serial.begin(9600);
}
void loop()
{
val = analogRead(tempPin);
float mv = ( val/1024.0)*5000;
float cel = mv/10;
float farh = (cel*9)/5 + 32;

Serial.print("TEMPRATURE = ");
Serial.print(cel);
Serial.print("*C");
Serial.println();
delay(1000);

/* uncomment this to get temperature in farenhite
Serial.print("TEMPRATURE = ");
Serial.print(farh);
Serial.print("*F");
Serial.println();


*/
}
```

Step 3: Compile the program and upload to Arduino UNO board. Now you can see the temperature being displayed in your IDE.



```
float temp;
int tempPin = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  temp = analogRead(tempPin);
  temp = temp * 0.48828125;
  Serial.print("TEMPRATURE = ");
  Serial.print(temp);
  Serial.print("*C");
  Serial.println();
  delay(1000);
}
```

```
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
TEMPRATURE = 32.71*C
```

Lesson 16 – Flame Sensor

Overview:

In this tutorial, we will learn how to use flame sensor and deduct the flame.

Components:

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Flame Sensor
- 1 * Breadboard
- Jumper Wires

Principle:

In this tutorial, we will learn how to use flame sensor and deduct the flame.

Procedure:

Step 1: Build the circuit as below:

Step 2: Program: You can copy paste the below program in the IDE or open the code directly from the “CODE” folder that comes with the DVD/from the downloaded Zip folder.

```
/*********************************************************
File name:   01_flamesen.ino
Description:  Flame Sensor.
Website: www.quadstore.in
*********************************************************/
int flameSensorPin = 0; // the cell and 10K pulldown are connected to a0
int flameSensorReading; // the analog reading from the analog resistor divider

void setup(void) {
Serial.begin(9600);
}

void loop(void) {

flameSensorReading = analogRead(flameSensorPin);

Serial.print("Analog reading = ");
Serial.println(flameSensorReading); // the raw analog reading delay(1000);

}
```

Step 3: Compile the program and upload to Arduino UNO board. Bring the flame near to the sensor and see the change in the value in IDE.

Lesson 17 – Active Buzzer

Overview:

In this tutorial, we will learn how to use Active Buzzer.

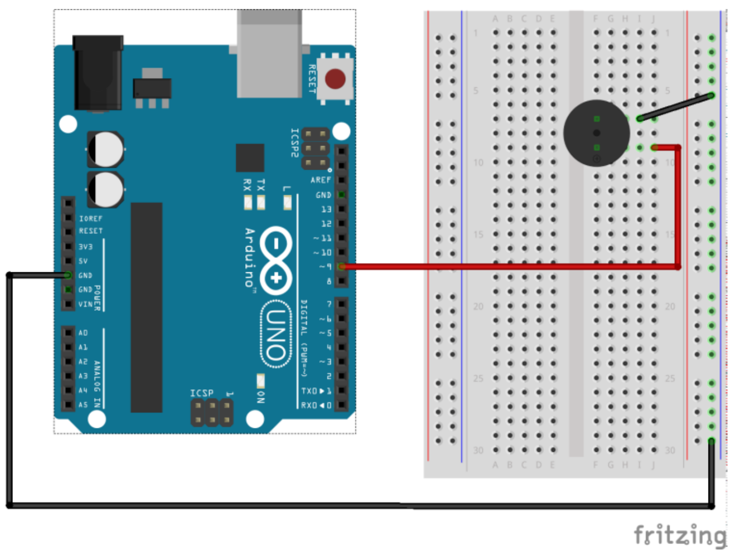
Components:

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Active Buzzer (Has a white sticker on top of the buzzer)
- 1 * Breadboard
- 2 * Jumper Wires

Principle:
Active Buzzer Arduino module, it produces a single-tone sound when signal is high

Procedure:

Step 1: Build the circuit as below:



Step 2: Program : You can copy paste the below program in the IDE or open the code directly from the "CODE" folder that comes with the DVD/from the downloaded Zip folder.

```
/************************************************************
File name:   _aBuzzer.ino
Description:  Active Buzzer.
Website: www.quadstore.in
************************************************************/
int buzzerPin = 9;

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

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

Step 3: Compile the program and upload to Arduino UNO board. The code will continually turn the buzzer on and off generating a series of short high-pitched beeps.

Lesson 18 – Passive Buzzer

Overview:

In this tutorial, we will learn how to use Active Buzzer.

Components:

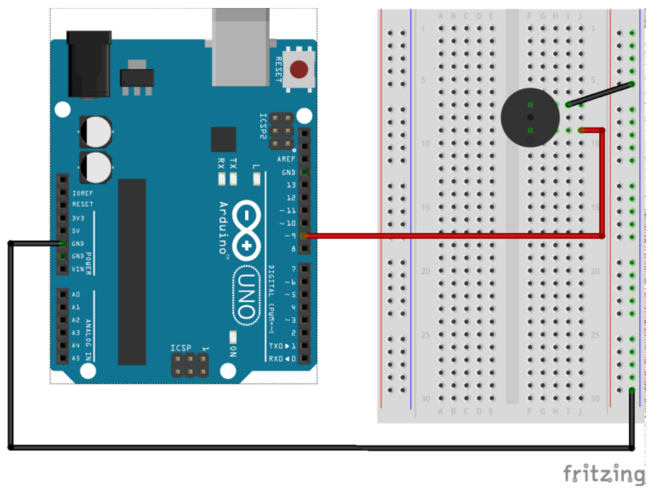
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * Passive Buzzer
- 1 * Breadboard
- 2 * Jumper Wires

Principle:

Passive piezoelectric buzzer can generate tones between 1.5 to 2.5 kHz by switching it on and off at different frequencies either using delays or PWM

Procedure:

Step 1: Build the circuit as below:



Step 2: Program : You can copy paste the below program in the IDE or open the code directly from the "CODE" folder that comes with the DVD/from the downloaded Zip folder.

```
/************************************************************
File name:   _pBuzzer.ino
Description:  Passive Buzzer.
Website: www.quadstore.in
************************************************************/
int buzzer = 9; // set the buzzer control digital IO pin

void setup() {
            pinMode(buzzer, OUTPUT); // set pin 9 as output
}

void loop() {
            for (int i = 0; i < 80; i++) {  // make a sound
                        digitalWrite(buzzer, HIGH); // send high signal to buzzer
                        delay(1); // delay 1ms
                        digitalWrite(buzzer, LOW); // send low signal to buzzer
                        delay(1);
            }
            delay(50);
            for (int j = 0; j < 100; j++) { //make another sound
                        digitalWrite(buzzer, HIGH);
                        delay(2); // delay 2ms
                        digitalWrite(buzzer, LOW);
                        delay(2);
            }
            delay(100);
}
```

Step 3: Compile the program and upload to Arduino UNO board. The code will generate two different tones by turning on and off the buzzer at different frequencies using a delay.

Lesson 19: Controlling Stepper Motor by IR Remote

Overview

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

Components

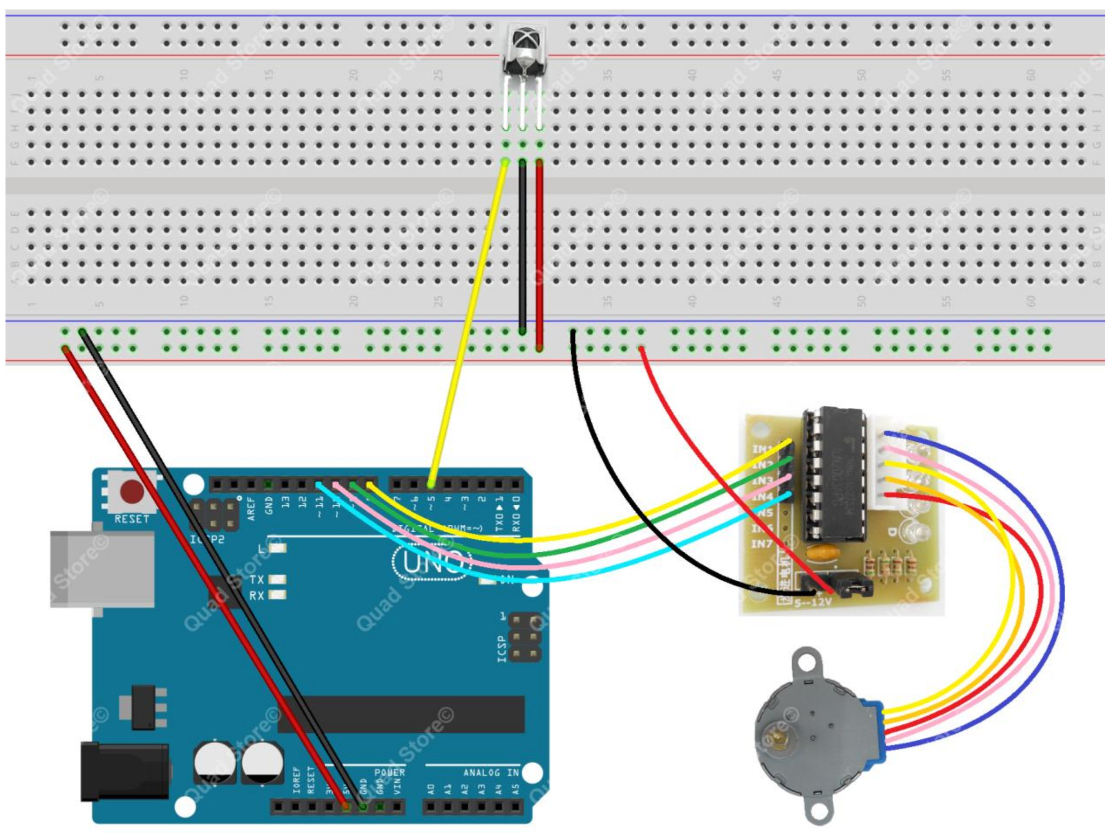
- 1 * Arduino UNO
- 1 * USB Cable
- 1 * IR Receiver HX1838
- 1 * Remote Controller
- 1 * Stepper Motor
- 1 * ULN2003 Driver Board
- 1 * Breadboard
- Several jumper wires

Principle

If you press a certain key on the remote controller, the corresponding signal will be sent to the Arduino UNO, and then the Arduino can tell the stepper motor how many steps it needs to run in which direction. For example, when you press the key '+', the stepper motor will run clockwise; press the key '-', the motor will run counterclockwise; press the key '2', it will rotate two laps; press the key '5', it will rotate five laps.

Procedure:

Step 1: Build the circuit



Step 2: Program

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

Lesson 20 - Controlling Size of a Circle by Potentiometer

Overview

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

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 1 * 10kΩ Potentiometer
- 1 * Breadboard
- Several jumper wires

Principle

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

Arduino key function:

●write()

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

Syntax:

Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

Parameters:

val: a value to send as a single byte

str: a string to send as a series of bytes

buf: an array to send as a series of bytes

len: the length of the buffer

Returns:

byte

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

Processing key function:

●Name: size()

Description:

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

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

Syntax:

size(w, h)

size(w, h, renderer)

Parameters:

w	int: width of the display window in units of pixels
h	int: height of the display window in units of pixels

renderer

String: Either P2D, P3D, or PDF

Returns:

void

●Name: println()

Description:

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

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

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

Syntax:

```
println()

println(what)

println(variables)
```

Parameters:

what	Object, String, float, char, boolean, or byte: data to print to console
variables	Object[]: list of data, separated by commas

Returns:

void

●Name: background()

Description:

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

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

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

Syntax:

```
background(rgb)
```

background(rgb, alpha)

background(gray)

background(gray, alpha)

background(v1, v2, v3)

background(v1, v2, v3, alpha)

background(image)

Parameters:

rgb	int: any value of the color datatype
alpha	float: opacity of the background
gray	float: specifies a value between white and black
v1	float: red or hue value (depending on the current color mode)
v2	float: green or saturation value (depending on the current color mode)
v3	float: blue or brightness value (depending on the current color mode)
image	PImage: PImage to set as background (must be same size as the sketch window)

Returns:

void

●Name: fill()

Description:

Sets the color used to fill shapes. For example, if you run fill(204, 102, 0), all subsequent shapes will be filled with orange. This color is either specified in terms of the RGB or HSB color depending on the 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:ellipse()

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

Syntax:

ellipse(a, b, c, d)

Parameters:

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

Returns:

Void

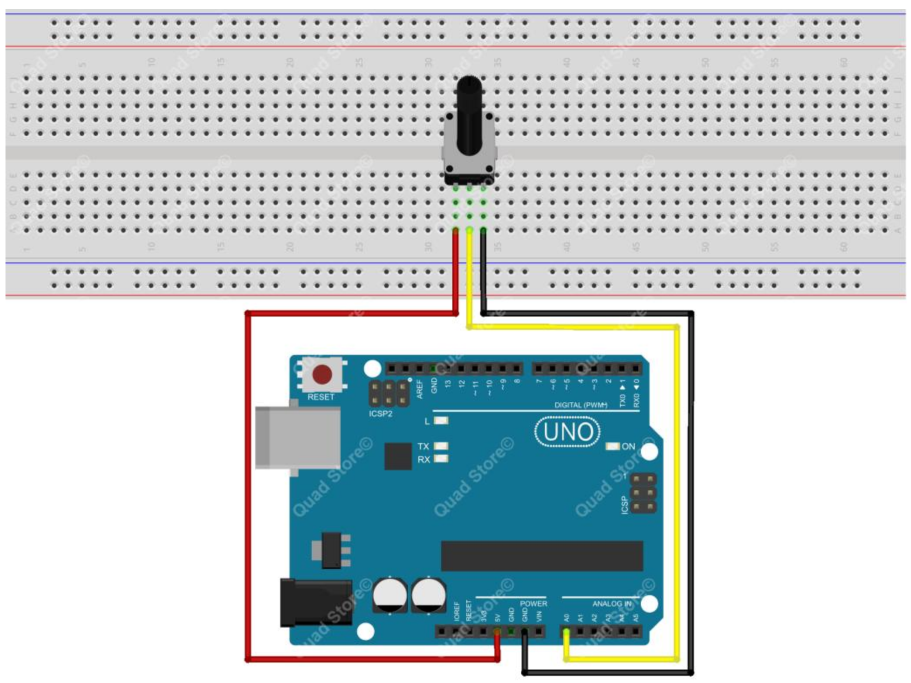
Note:

1. In this experiment, my Arduino UNO board is connected to my computer port COM26. But it may differ in your case. So please adjust it according to your actual situation.

2. If Processing prompts that you need to install the related function library, please do it.

Procedure:

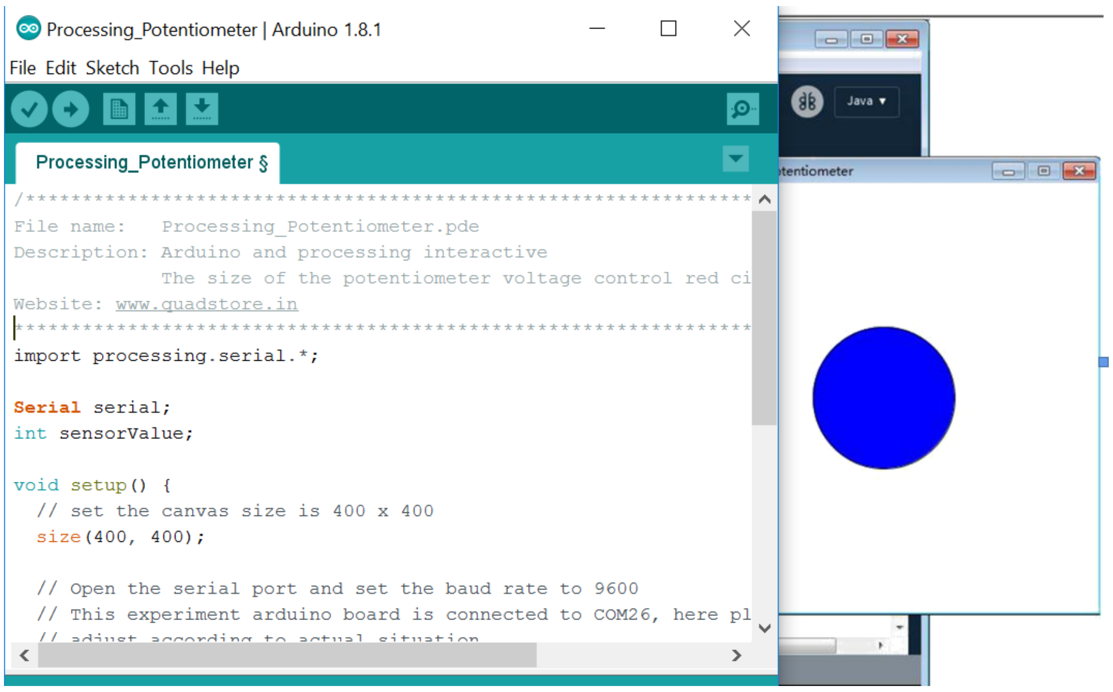
Step 1: Build the circuit



Step 2: Program

Step 3: Compile the program and upload to Arduino UNO board Step 4: Run the Processing software (Processing_Potentiometer.pde)

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


Processing_Potentiometer | Arduino 1.8.1
File Edit Sketch Tools Help
Processing_Potentiometer §
Java
tentiometer

```
/*****************************************************************
File name:   Processing_Potentiometer.pde
Description: Arduino and processing interactive
             The size of the potentiometer voltage control red ci
Website: www.quadstore.in
*****************************************************************
import processing.serial.*;

Serial serial;
int sensorValue;

void setup() {
  // set the canvas size is 400 x 400
  size(400, 400);

  // Open the serial port and set the baud rate to 9600
  // This experiment arduino board is connected to COM26, here pl
  // adjust according to actual situation
```

Lesson 21 - Snake Game

Overview

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

Components

- 1 * Arduino UNO
- 1 * USB Cable
- 2 * Button
- 1 * Breadboard
- Several jumper wires

Principle

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

Play the Snake Game:

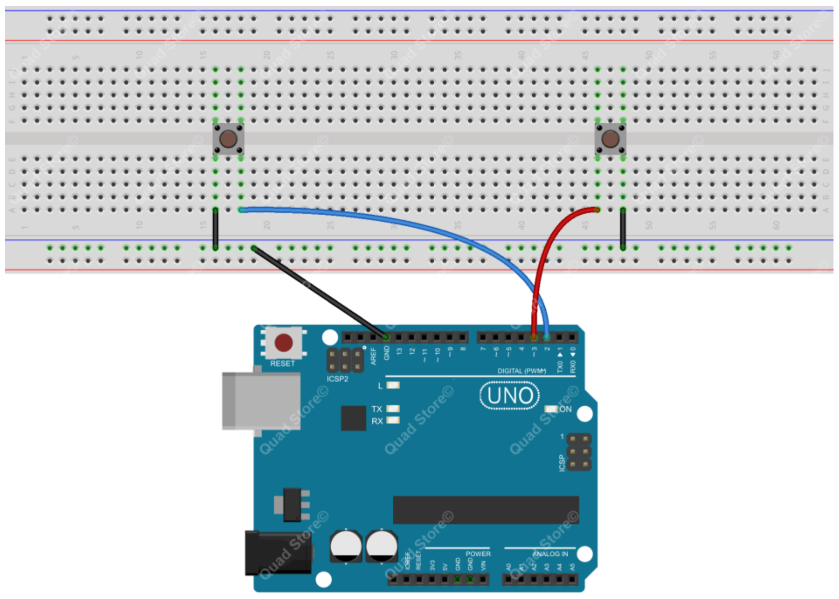
1. When you press the right button, the snake will move to the right.
2. Press the left button, and the snake will move to the left.

Note:

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

Procedure:

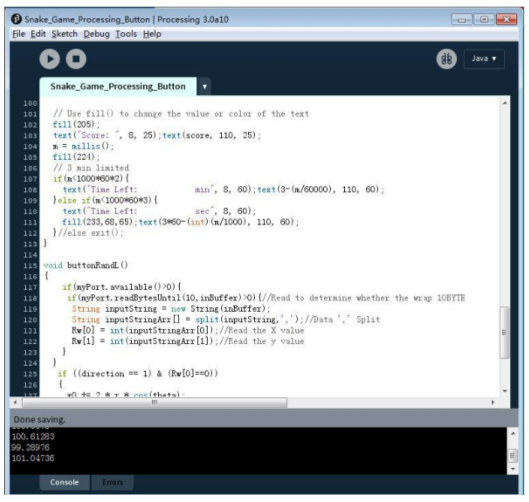
Step 1: Build the circuit



Step 2: Program

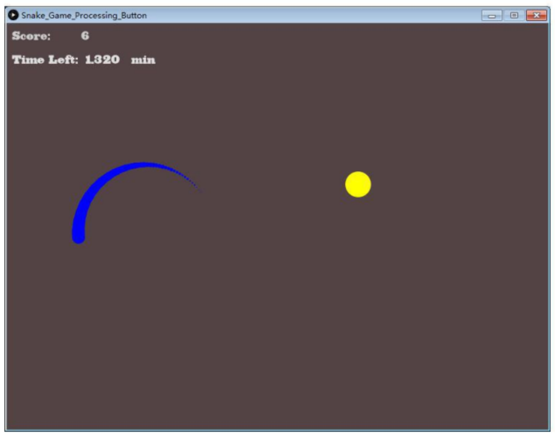
Step 3: Compile the program and upload to Arduino UNO board

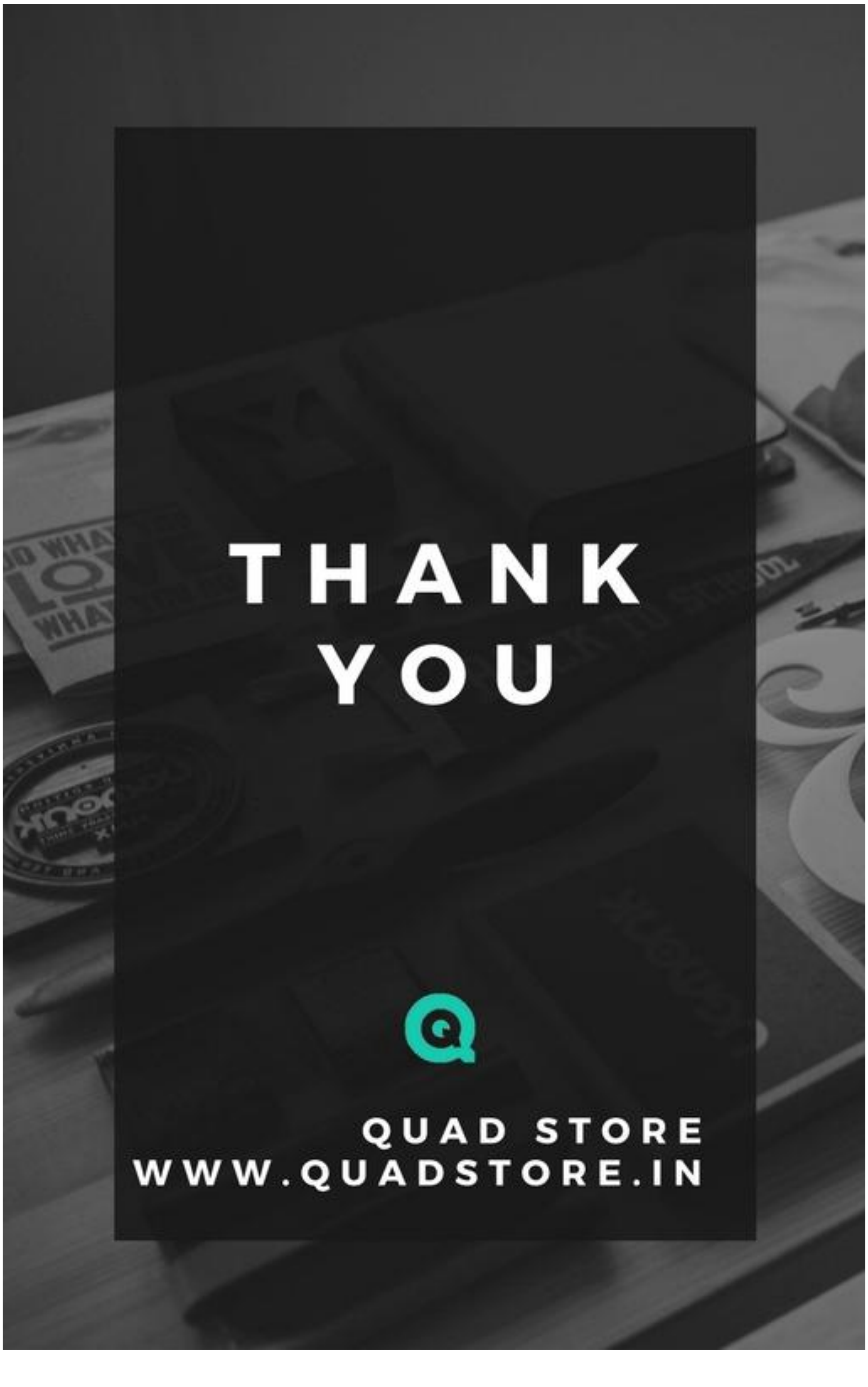
Step 4: Run the Processing software (Snake_Game_Processing_Button.pde)


Snake_Game_Processing_Button | Processing 3.0a10
File Edit Sketch Debug Tools Help
Java
Snake_Game_Processing_Button
Done saving.
Console
Errors

```
  // Use fill() to change the value or color of the text
  fill(205);
  text("Score: ", 8, 25);text(score, 110, 25);
  m = millis();
  fill(224);
  // 3 min limited
  if(m<1000*60*2){
    text("Time Left:            min", 8, 60);text(3-(m/60000), 110, 60);
  }else if(m<1000*60*3){
    text("Time Left:            sec", 8, 60);
    fill(233,68,65);text(3*60-(int)(m/1000), 110, 60);
  }//else exit();
}

void buttonRandL()
{
    if(myPort.available()>0){
     if(myPort.readBytesUntil(10,inBuffer)>0){//Read to determine whether the wrap 10BYTE
      String inputString = new String(inBuffer);
      String inputStringArr[] = split(inputString,',');//Data ',' Split
      Rw[0] = int(inputStringArr[0]);//Read the X value
      Rw[1] = int(inputStringArr[1]);//Read the y value
    }
  }
   if ((direction == 1) & (Rw[0]==0))
   {
```

```
100.61283
99.28976
101.04736
```





THANK YOU

QUAD STORE
WWW.QUADSTORE.IN