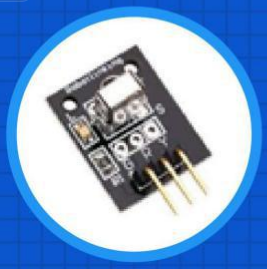
Arduino UNO R3 Getting Started Basic Suite Tutorial

V1.0.19.5.13

PACKING LIST

Stepper Motor
1PCS

Servo Motor (SG90)
1PCS

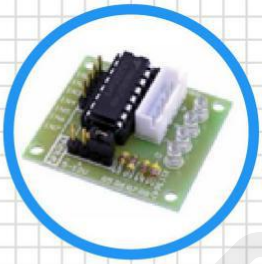


IR Receiver Module
1PCS

5V Relay
1PCS

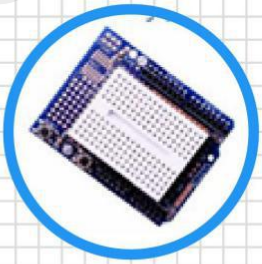
Uno R3 Controller Board 1PCS

LCD 1602 Module
(with pin header)
1PCS



2003 Stepper
Motor Driver
Board 1PCS

Power Supply
Module 1PCS



Prototype Expansion
Board 1PCS

PACKING LIST


Button(Small)
5PCS

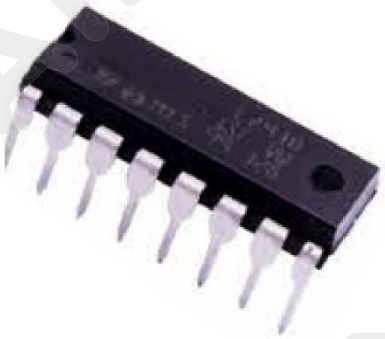

Potentiometer
1PCS


Passive Buzzer
1PCS


Active Buzzer
1PCS


Remote
1PCS

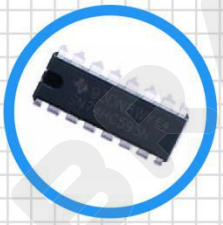
L293D
1PCS

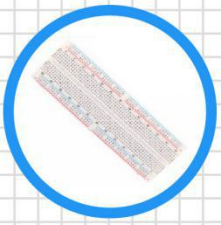



USB Cable
1PCS


Female-to-male
Dupont wire
10PCS


65 Jumper
Wire 1PCS


74HC595
1PCS


830 Tie-Points
Breadboard
1PCS

PACKING LIST

4 digit 7 - segment Display 1PCS

1 digit 7 - segment Display 1PCS

Tilt Switch 1PCS

Joystick Module 1PCS

Fan Blade and 3V DC Motor (with wire) each 1PCS

DHT 11 Temeperature and Humidity Module 1PCS

Ultrasonic Sensor 1PCS

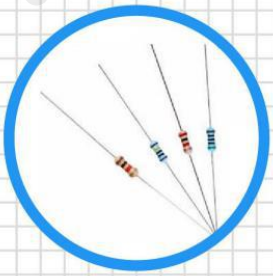
PACKING LIST

Thermistor
1PCS

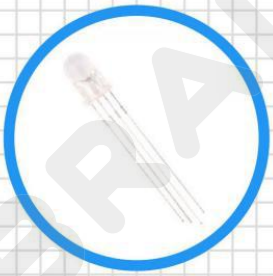
Diode Rectifier
2PCS

Photoresistor
(Photocell)
2PCS

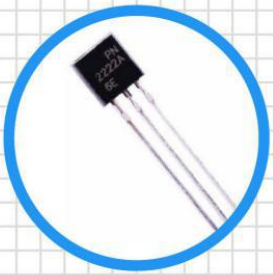
LED
20PCS



Resistor
30PCS



RGB
1PCS



NPN Transistor
PN2222
2PCS

Content

Lesson 1 Installing IDE

Introduction

The Arduino Integrated Development Environment (IDE) is the software side of the Arduino platform.

In this lesson, you will learn how to setup your computer to use Arduino and how to set about the lessons that follow.

The Arduino software that you will use to program your Arduino is available for Windows, Mac and Linux. The installation process is different for all three platforms and unfortunately there is a certain amount of manual work to install the software.

STEP 1: Go to https://www.arduino.cc/en/Main/Software and find below page.

The version available at this website is usually the latest version, and the actual version may be newer than the version in the picture.

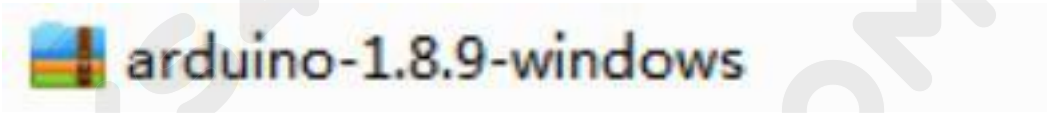
STEP2：Download the development software that is compatible with the operating system of your computer. Take Windows as an example here.

Click Windows Installer.

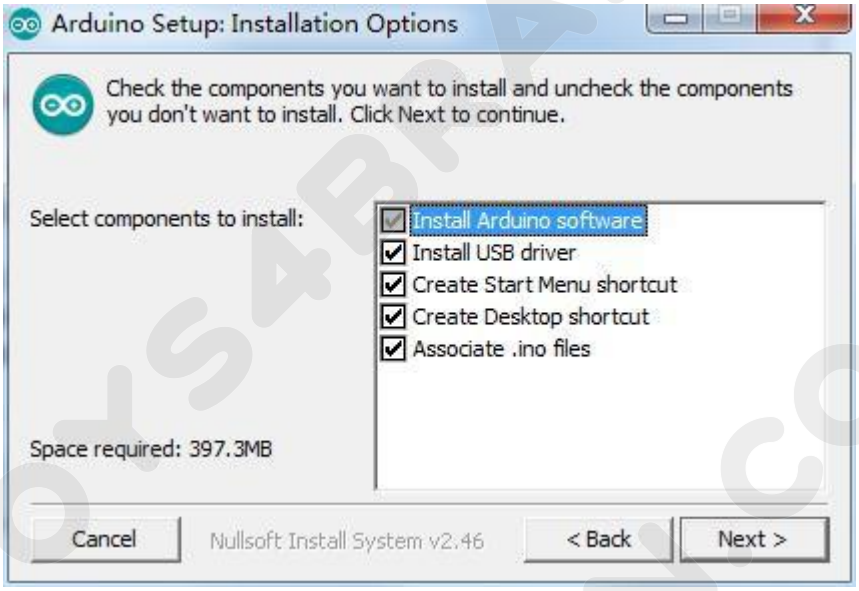
Click JUST DOWNLOAD.

Also version 1.8.9is available in the material we provided, and the versions of our materials are the latest versions when this course was made.

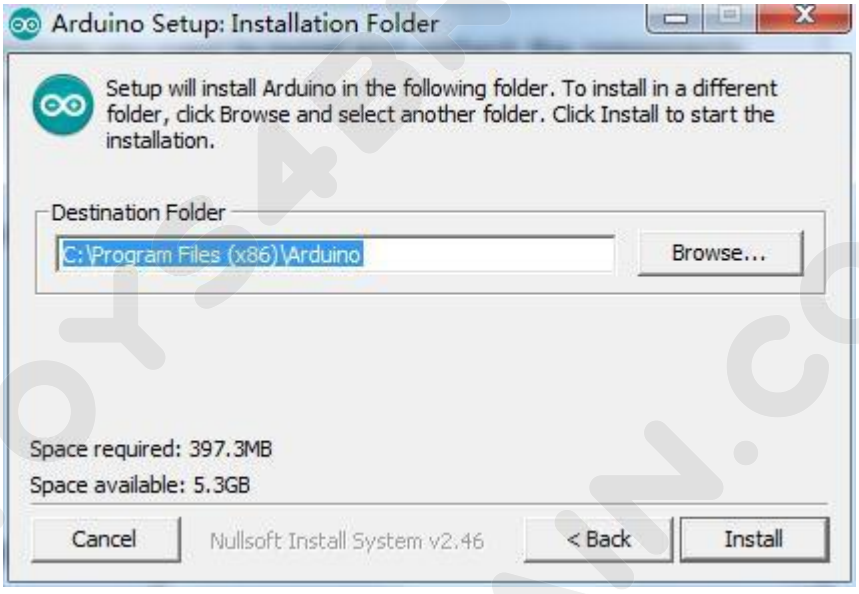
Installing Arduino (Windows)



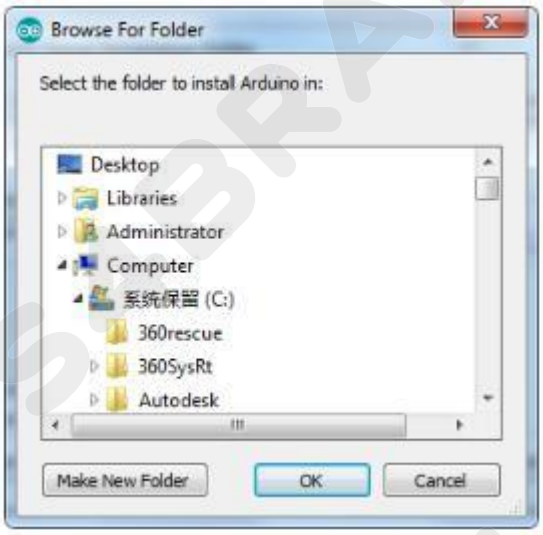
Click I Agree to see the following interface



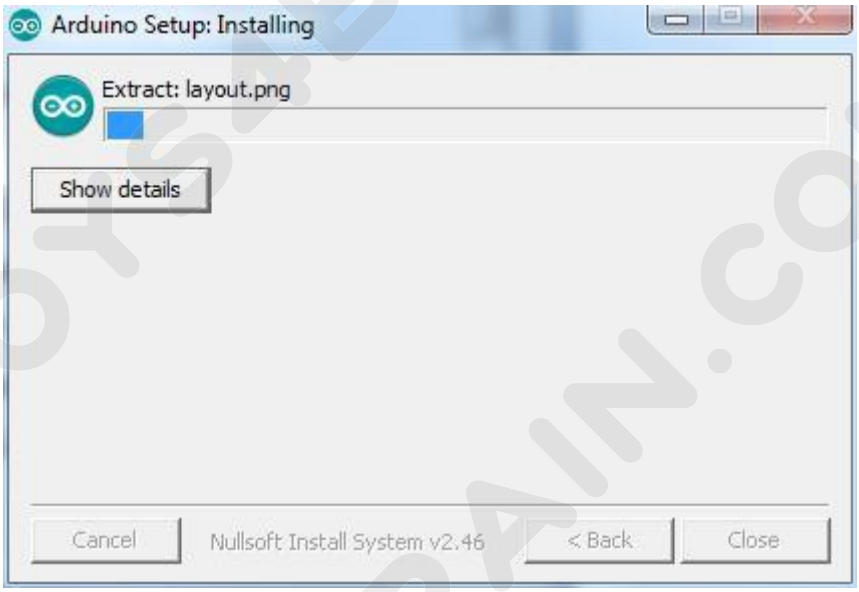
Click Next



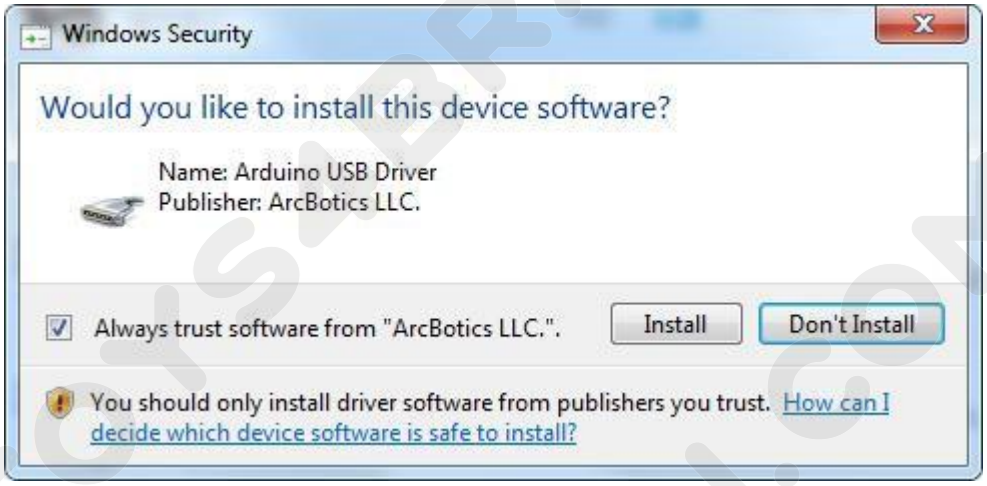
You can press Browse… to choose an installation path or directly type in the directory you want.



Click Install to initiate installation

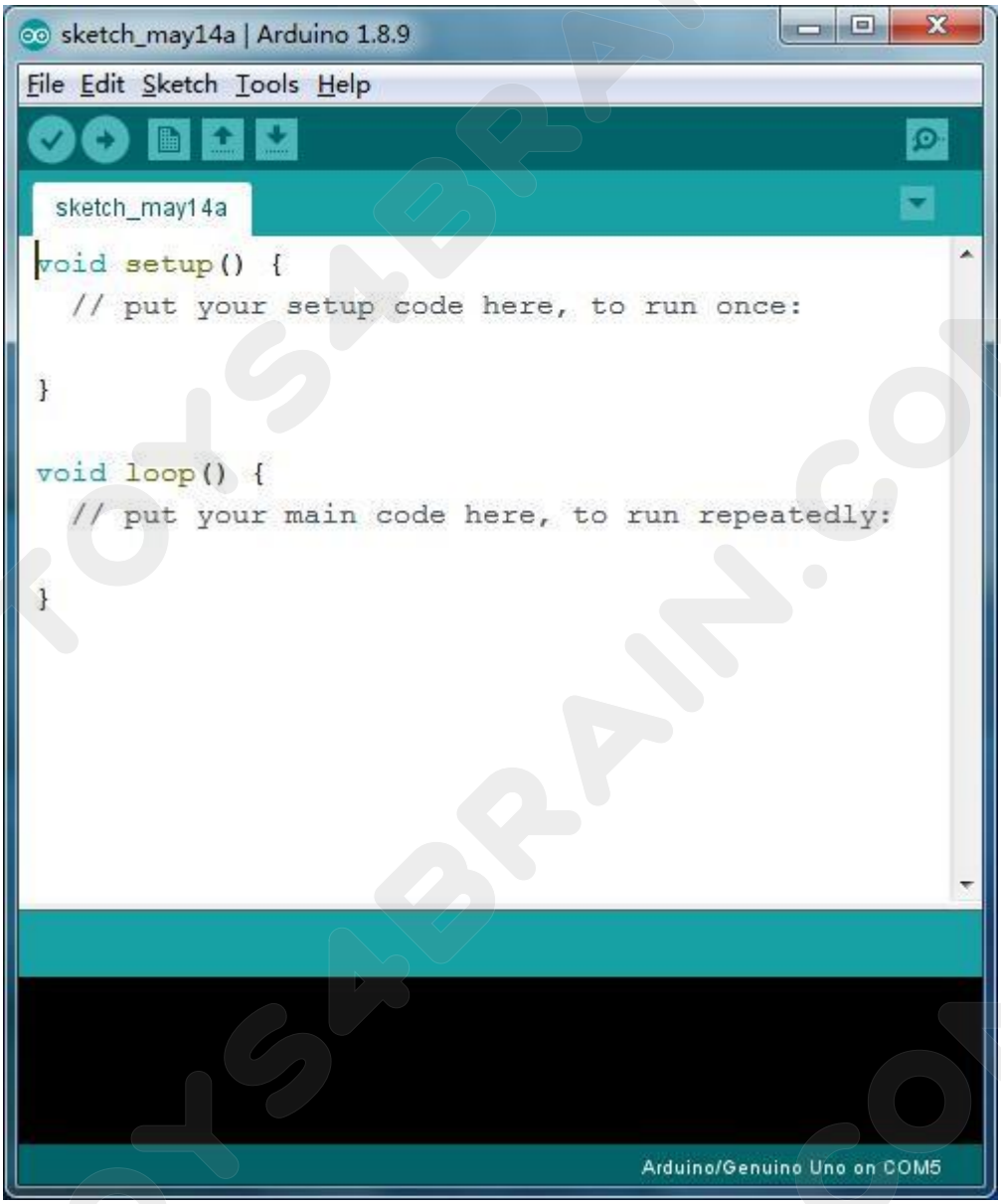


Finally, the following interface appears, click Install to finish the installation.



Next, the following icon appears on the desktop

Double-click to enter the desired development environment



Installing Arduino (Mac OS X)

Download and Unzip the zip file, double click the Arduino.app to enter Arduino IDE; the system will ask you to install Java runtime library if you don't have it in your computer. Once the installation is complete you can run the Arduino IDE.

Installing Arduino (Linux)

You will have to use the make install command. If you are using the Ubuntu system, it is recommended to install Arduino IDE from the software center of Ubuntu.

Lesson 2 Add Libraries

Installing Additional Arduino Libraries

Once you are comfortable with the Arduino software and using the built-in functions, you may want to extend the ability of your Arduino with additional libraries.

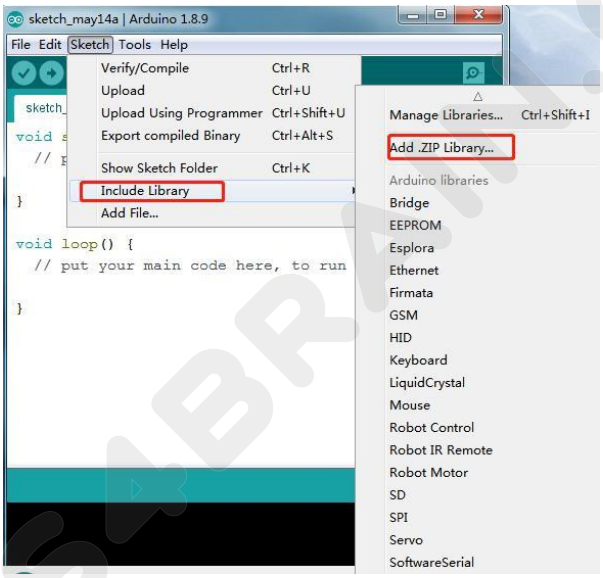
What are Libraries?

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in LiquidCrystal library makes it easy to talk to character LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. To use the additional libraries, you will need to install them.

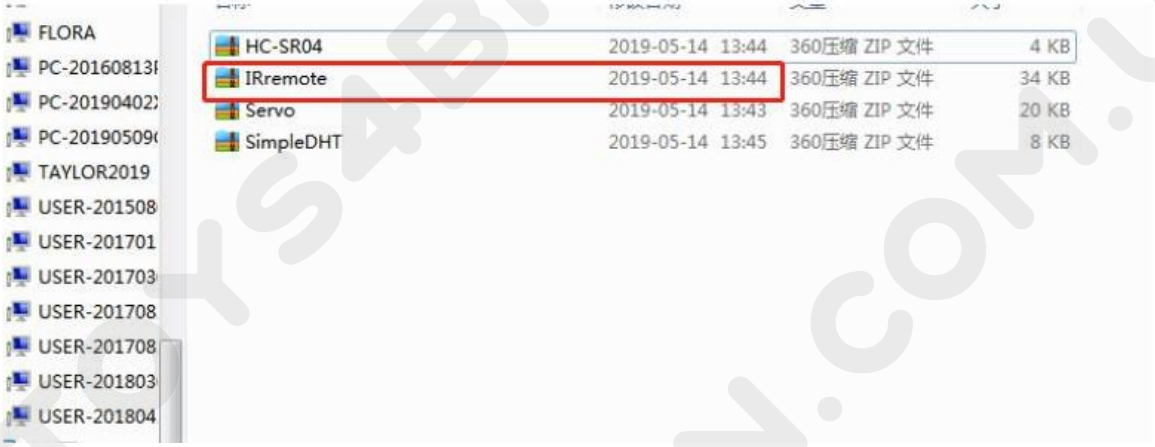
How to Install a Library

Using the Library Manager

To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.8.9). Open the IDE and click to the "Sketch" menu and then Include Library > Manage Libraries.



Then we check to see if the library is installed correctly.



Example: IRromote

Open arduino software - project - load library - add a .zip library

Add method two:

Copy the library folder to the Libraries folder in the Arduino installation directory.
Restart Arduino and the added library will take effect.

Lesson 3 Blink

Overview:

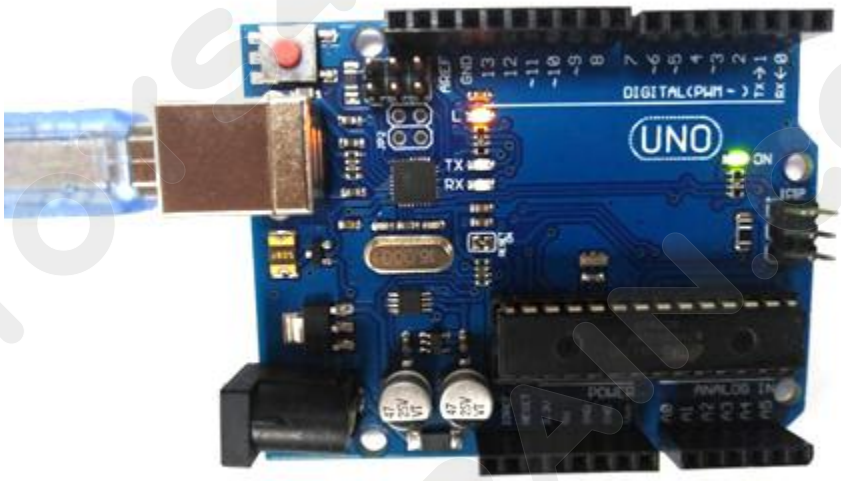
In this lesson, you will learn how to program your UNO R3 controller board to blink the Arduino's built-in LED, and how to download programs by basic steps.

Component Required:

1x Arduino UNO R3

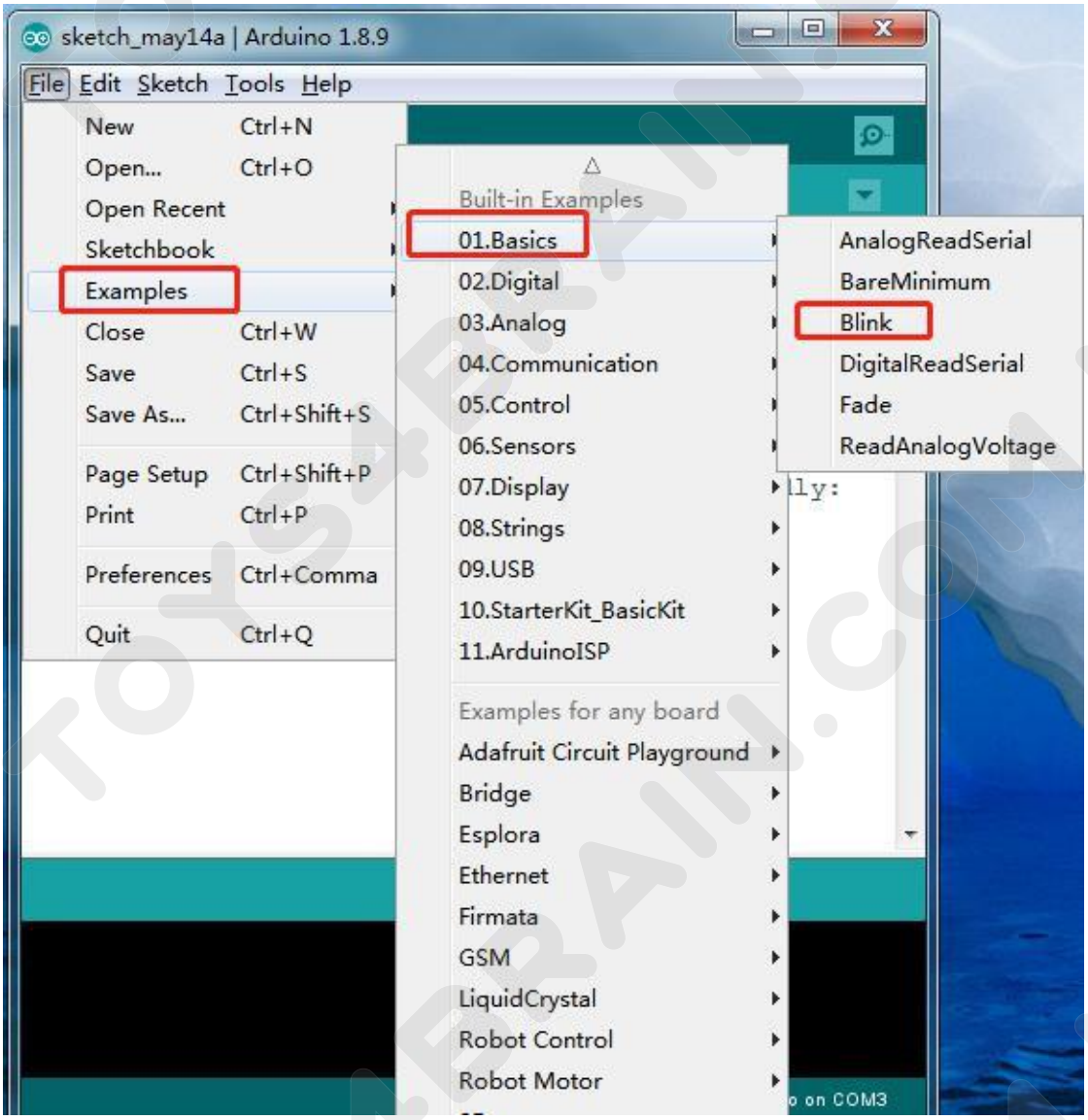
Principle:

The UNO R3 board has rows of connectors along both sides that are used to connect to several electronic devices and plug-in 'shields' that extends its capability. It also has a single LED that you can control from your sketches. This LED is built onto the UNO R3 board and is often referred to as the 'L' LED as this is how it is labeled on the board.

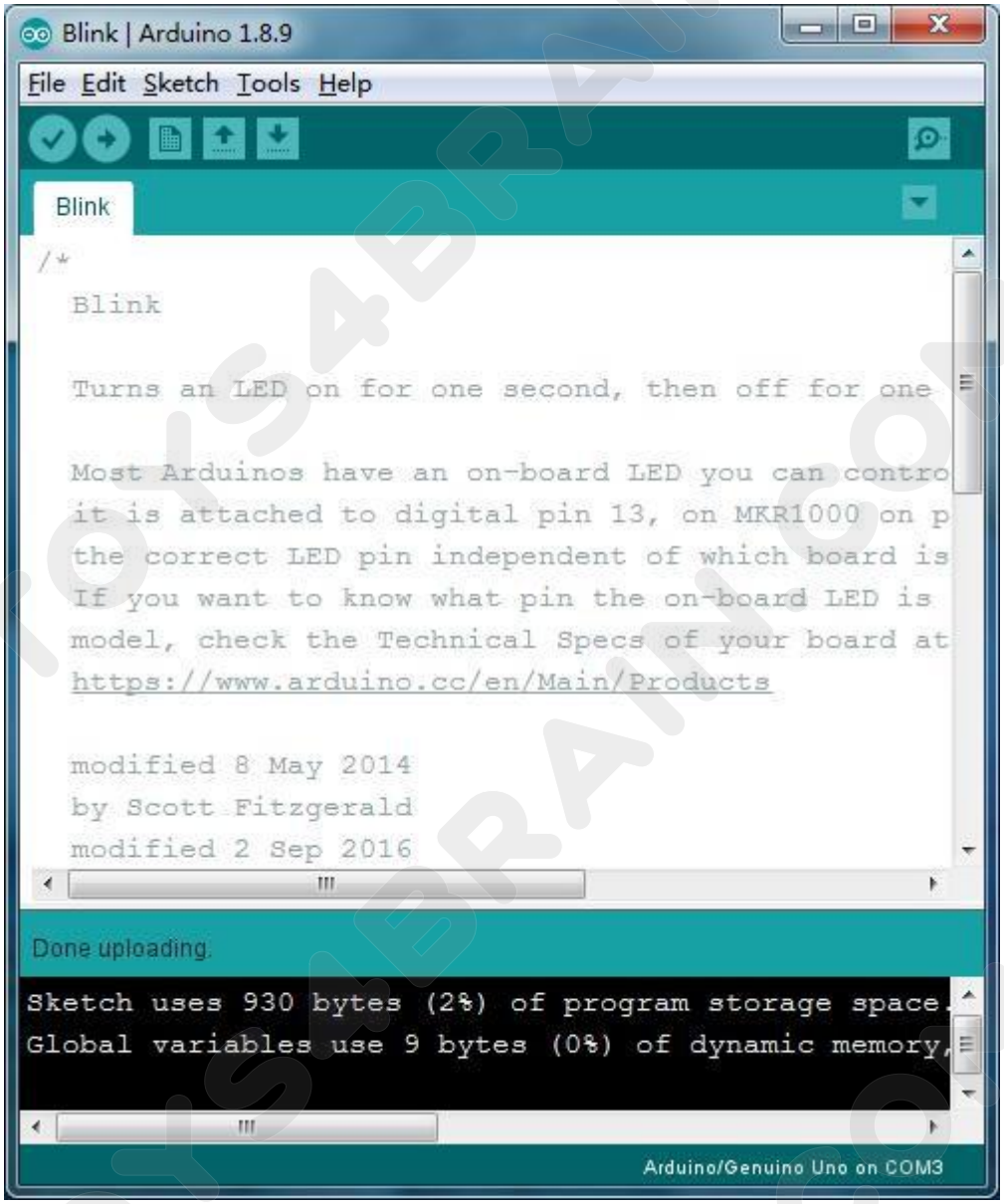


You may find that your UNO R3 board's 'L' LED already blinks when you connect it to a USB plug. This is because the boards are generally shipped with the 'Blink' sketch pre-installed.
In this lesson, we will reprogram the UNO R3 board with our own Blink sketch and then change the rate at which it blinks.
In Lesson 1, you set up your Arduino IDE and made sure that you could find the right serial port for it to connect to your UNO R3 board. The time has now come to put that connection to the test and program your UNO R3 board.
The Arduino IDE includes a large collection of example sketches that you can load up and use. This includes an example sketch for making the 'L' LED blink.

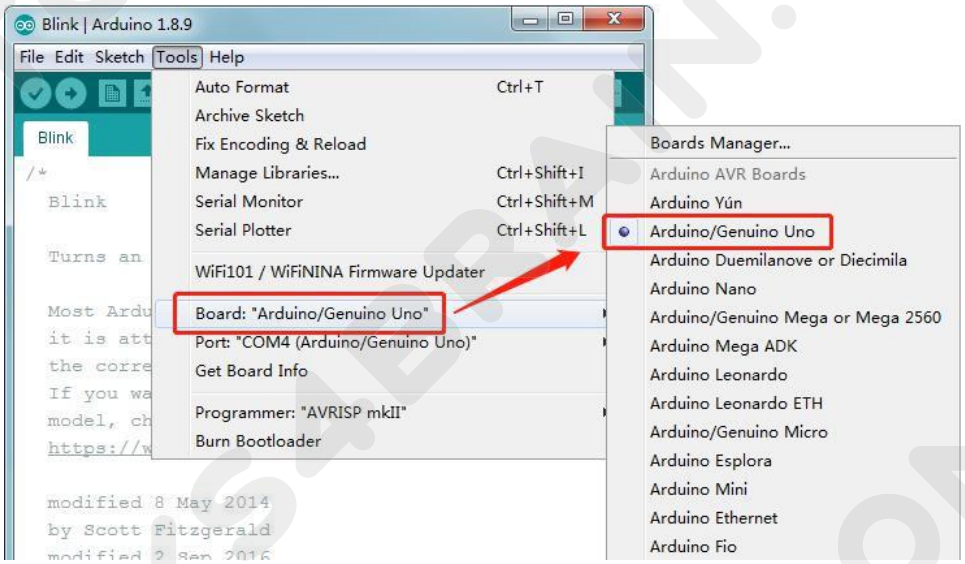
Load the 'Blink' sketch that you will find in the IDE's menu system under File > Examples > 01.Basics



Open as shown below:

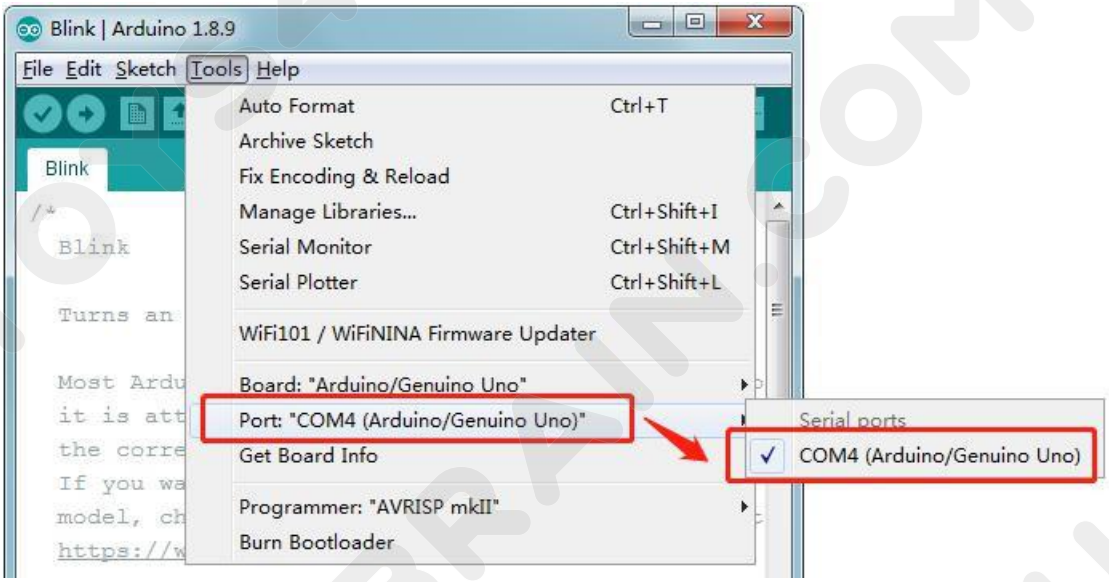


Upload code needs to select the board model Arduino/Genuino Uno



The upload code also needs to select the board port number. Our computer displays the port number of arduino as: "COM4 Arduio / Genuino Uno", you may be the other serial port number.

Note: A correct COM port should be COMX (arduino) XXX), which is certified by the standard.



Note that a huge part of this sketch is composed of comments. These are not actual program instructions; rather, they just explain how the program works. They are there for your benefit.

Everything between /* and */ at the top of the sketch is a block comment; it explains what the sketch is for.

Single line comments start with // and everything up until the end of that line is considered a comment.

The first line of code is:

int led = 13;

As the comment above it explains, this is giving a name to the pin that the LED is attached to. This is 13 on most Arduinos, including the UNO and Leonardo.

Next, we have the 'setup' function. Again, as the comment says, this is executed when the reset button is pressed. It is also executed whenever the board resets for any reason, such as power first being applied to it, or after a sketch has been uploaded.

```
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
```

Every Arduino sketch must have a 'setup' function, and the place where you might want to add instructions of your own is between the { and the }.

In this case, there is just one command there, which, as the comment states tells the Arduino board that we are going to use the LED pin as an output.

It is also mandatory for a sketch to have a 'loop' function. Unlike the 'setup' function that only runs once, after a reset, the 'loop' function will, after it has finished running its commands, immediately start again.

```
void loop() {
   digitalWrite(led, HIGH);     // turn the LED on (HIGH is the voltage level)
   delay(1000);                    // wait for a second
   digitalWrite(led, LOW);      // turn the LED off by making the voltage LOW
   delay(1000);                    // wait for a second
}
```

Inside the loop function, the commands first of all turn the LED pin on (HIGH), then 'delay' for 1000 milliseconds (1 second), then turn the LED pin off and pause for another second.

You are now going to make your LED blink faster. As you might have guessed, the key to this lies in changing the parameter in () for the 'delay' command.

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                       // wait for a second
}
```

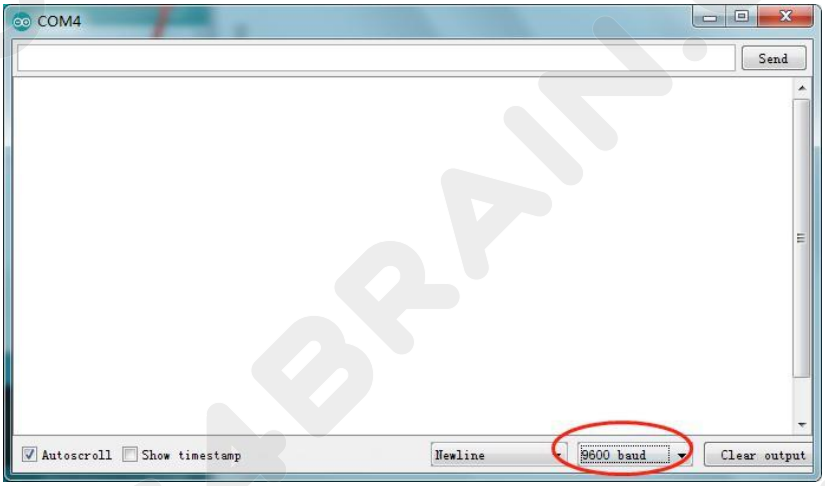
This delay period is in milliseconds, so if you want the LED to blink twice as fast, change the value from 1000 to 500. This would then pause for half a second each

delay rather than a whole second.

Upload the sketch again and you should see the LED start to blink more quickly.

Note: I want to add some information about how to open the serial monitor here.

Serial port is generally selected 9600HZ



Lesson 4 Button Control LED Delay Switch

Overview:

In this course, you will learn how to use the button control LEDs to implement the delay control function.

Component Required:

1 x Arduino Uno R3

1 x 5mm red LED

1 x 220 ohm resistor

1 x 10k ohm resistor

1 x 830 Tie Points Breadboard

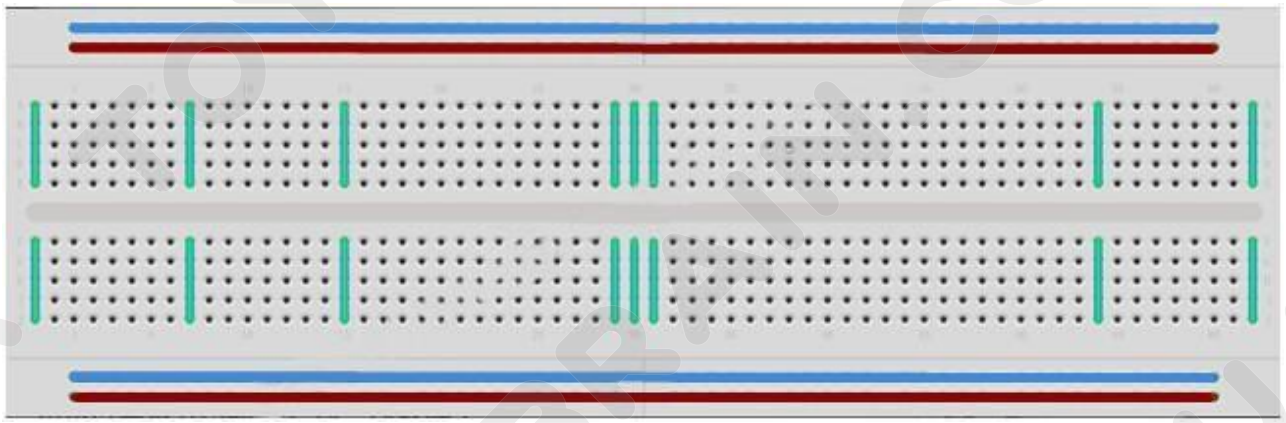
1 x button

5 x M-M wires (Male to Male jumper wires)

Component Introduction:

BREADBOARD MB-102：

A breadboard enables you to prototype circuits quickly, without having to solder the connections. Below is an example.



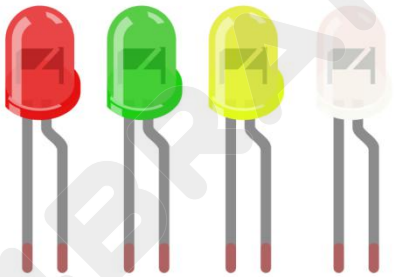
Breadboards come in various sizes and configurations. The simplest kind is just a grid of holes in a plastic block. Inside are strips of metal that provide electrical connection between holes in the shorter rows. Pushing the legs of two different

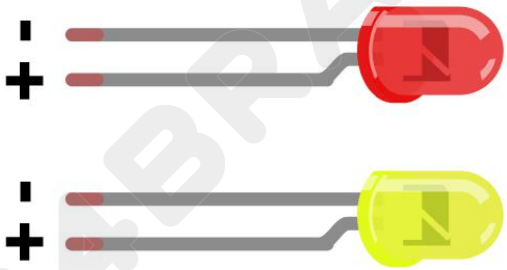
components into the same row joins them together electrically. A deep channel running down the middle indicates that there is a break in connections there, meaning, you can push a chip in with the legs at either side of the channel without connecting them together. Some breadboards have two strips of holes running along the long edges of the board that are separated from the main grid. These have strips running down the length of the board inside and provide a way to connect a common voltage. They are usually in pairs for +5 volts and ground. These strips are referred to as rails and they enable you to connect power to many components or points in the board.

While breadboards are great for prototyping, they have some limitations. Because the connections are push-fit and temporary, they are not as reliable as soldered connections. If you are having intermittent problems with a circuit, it could be due to a poor connection on a breadboard.

LED:

LED (Light Emitting Diode), which converts electrical energy into light energy, also has unidirectional conductivity and a reverse breakdown voltage of about 5V. Its forward volt-ampere characteristic curve is very steep, and the current-limiting resistor must be connected in series. In a 5V circuit, a resistor of about 400 ohms is generally used.
The longer of the two pins of the LED is the positive pole. There are two ways to connect, when the positive pole of the led through the current limiting resistor and Arduino
The I/O port is connected and the other end is grounded. At this time, when the Arduino output is high, the led is lit, and when the output is low, the led is off.
2. When the negative pole of the led is connected to the I/O port of the Arduino, the other end is connected to the 5V voltage through the current limiting resistor. At this time, Arduino
When the output is low, the led is lit, and when the output is high, the led is off.





If you do not use a resistor with an LED, then it may well be destroyed almost immediately, as too much current will flow through, heating it and destroying the 'junction' where the light is produced.

There are two ways to tell which is the positive lead of the LED and which the negative.

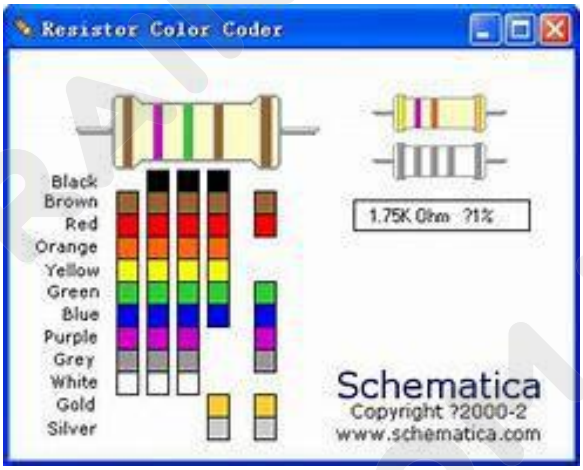
Firstly, the positive lead is longer.

Secondly, where the negative lead enters the body of the LED, there is a flat edge to the case of the LED.

If you happen to have an LED that has a flat side next to the longer lead, you should assume that the longer lead is positive.

RESISTORS:

As the name suggests, resistors resist the flow of electricity. The higher the value of the resistor, the more it resists and the less electrical current will flow through it. We are going to use this to control how much electricity flows through the LED and therefore, how brightly it shines.

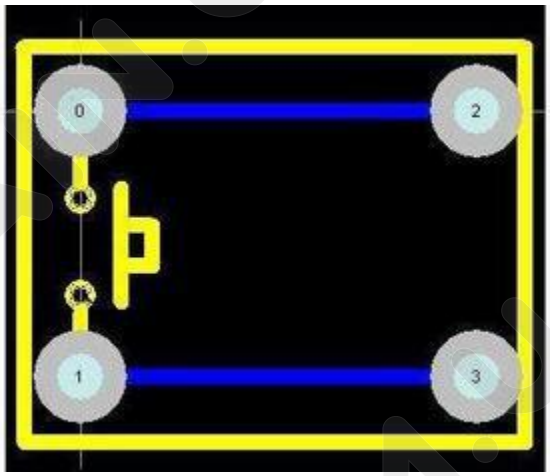


Unlike LEDs, resistors do not have a positive and negative lead. They can be connected either way around.

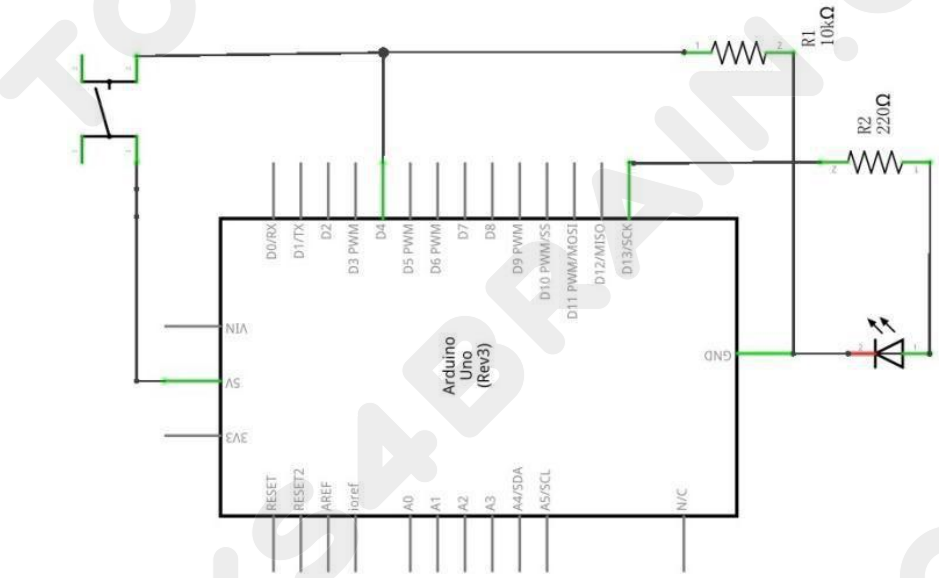
If you find this approach method t oo complicated, you can read the color ring flag on our resistors directly to determine its resistance value. Or you may use a digital multimeter instead.

Button:

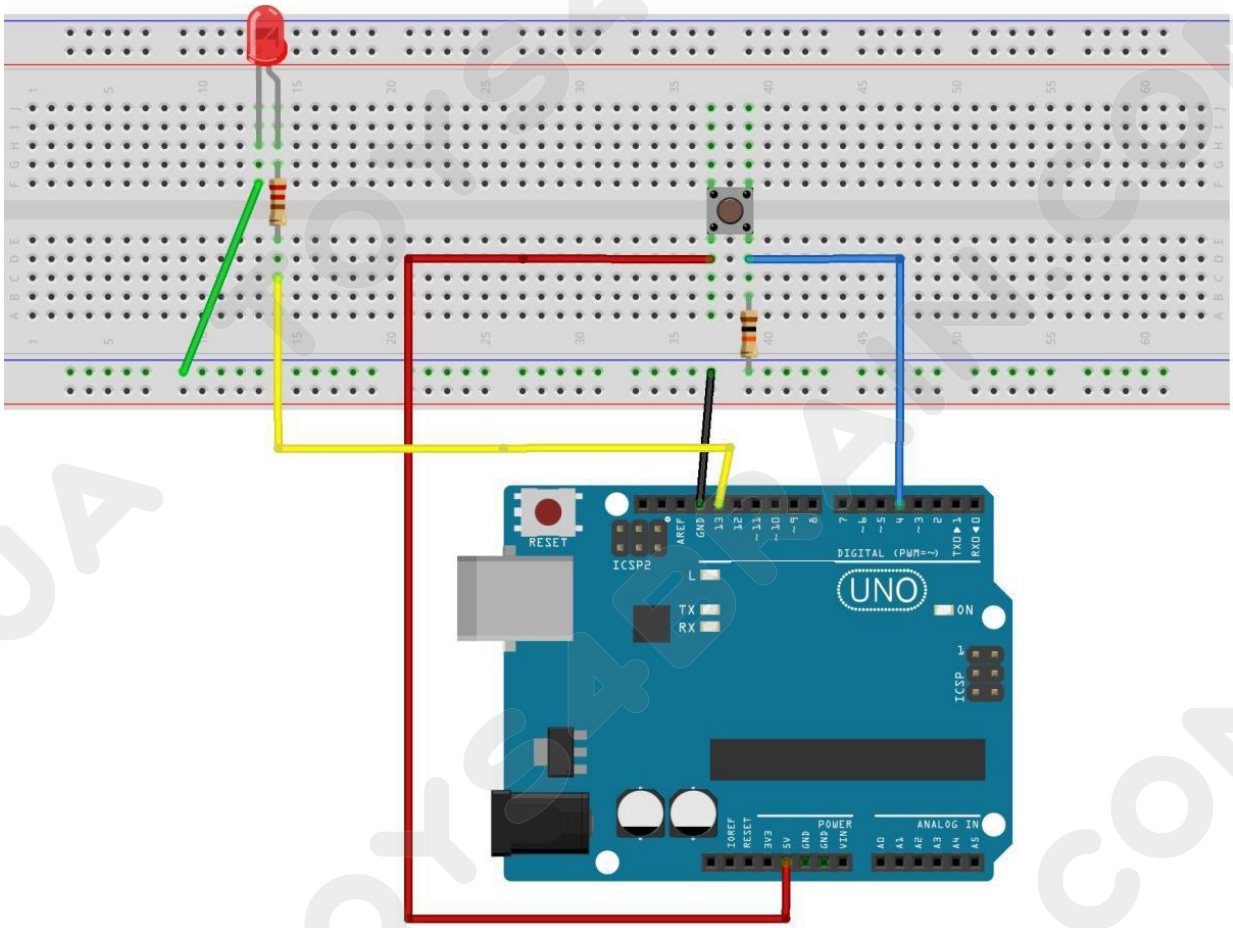
The button as an I/O port input device has seen the effect of the last use. It is actually used to control the on/off of the line, and the effect of changing the I/O port signal is achieved by adding a device such as a pull-up resistor.



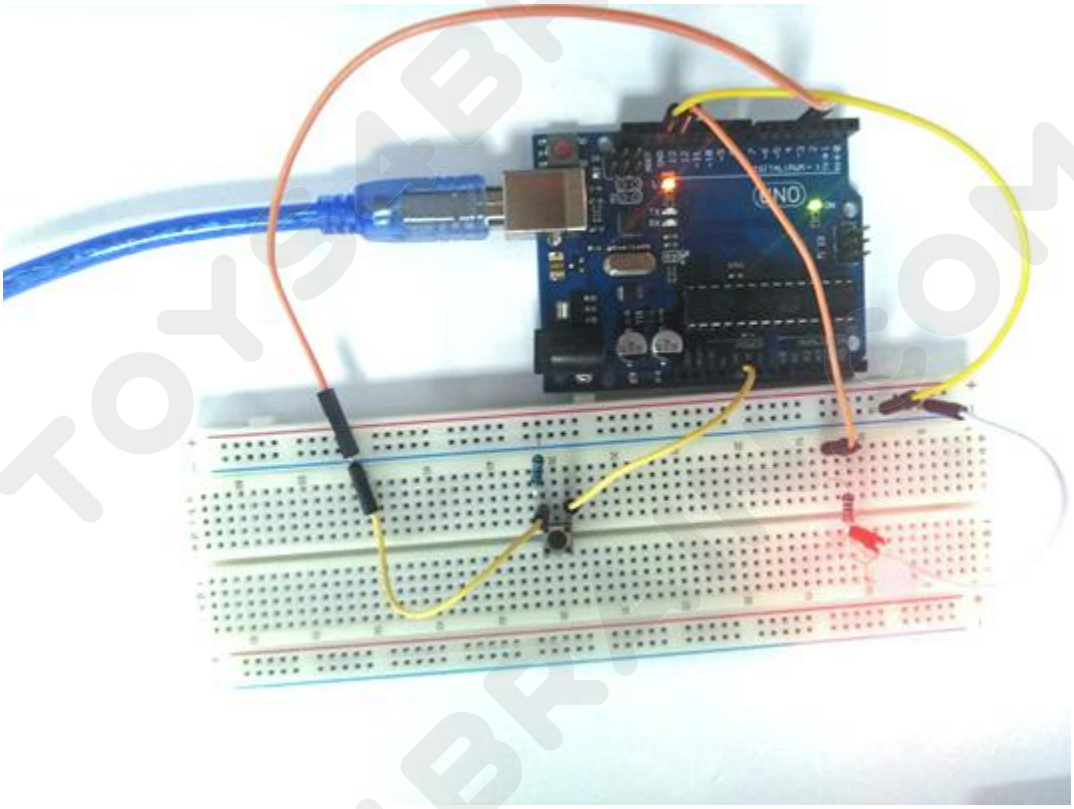
Connection Diagram:



Wiring schematic:



Physical wiring diagram:



Code:

```
void setup ()
{
   pinMode(4,INPUT);                  //Set port 4 as input and port 13 as output mode
   pinMode(13,OUTPUT);
}

void loop()
{
  int n =digitalRead(4);       //Create a variable n and assign it the state of the number 4 digital
port.
   if (n==HIGH)                      //To determine whether n is a high level, if the following statement
is executed, then skip.
       delay(1000);
       digitalWrite(13,HIGH);
       delay(5000);
       digitalWrite(13,LOW);
```

```
}
}
```

In the program we have a new statement, the judgment sentence If, please refer to the Arduino grammar manual to check the information about the If statement.

The Arduino grammar manual links are as follows:

https://www.arduino.cc/reference/en/

Lesson 5 RGB LED

Overview:

RGB LEDs are a fun and easy way to add some color to your projects. Since they are like 3 regular LEDs in one, how to use and connect them is not much different. They come mostly in 2 versions: Common Anode or Common Cathode.
Common Anode uses 5V on the common pin, while Common Cathode connects to ground.
As with any LED, we need to connect some resistors inline (3 total) so we can limit the current being drawn.
In our sketch, we will start with the LED in the Red color state, then fade to Green, then fade to Blue and finally back to the Red color. By doing this we will cycle through most of the color that can be achieved.

Component Required:

1 x Arduino Uno R3

1 x 830 Tie Points Breadboard

4 x M-M wires (Male to Male jumper wires)

1 x RGB LED

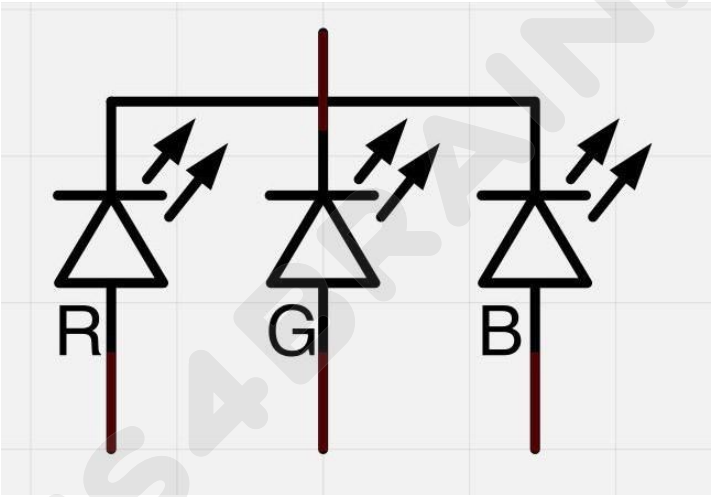
3 x 220 ohm resistors

Component Introduction

RGB:

At first glance, RGB (Red, Green and Blue) LEDs look just like regular LEDs.
However, inside the usual LED package, there are actually three LEDs, one red, one green and yes, one blue. By controlling the brightness of each of the individual LEDs you can mix pretty much any color you want.
We mix colors the same way you would mix paint on a palette - by adjusting the brightness of each of the three LEDs. The hard way to do this would be to use different value resistors (or variable resistors) as we did with in Lesson 2, but that's a lot of work! Fortunately for us, UNO R3 board has an analogWrite function that you can use with pins marked with a ~ to output a variable amount of power to the appropriate LEDs.

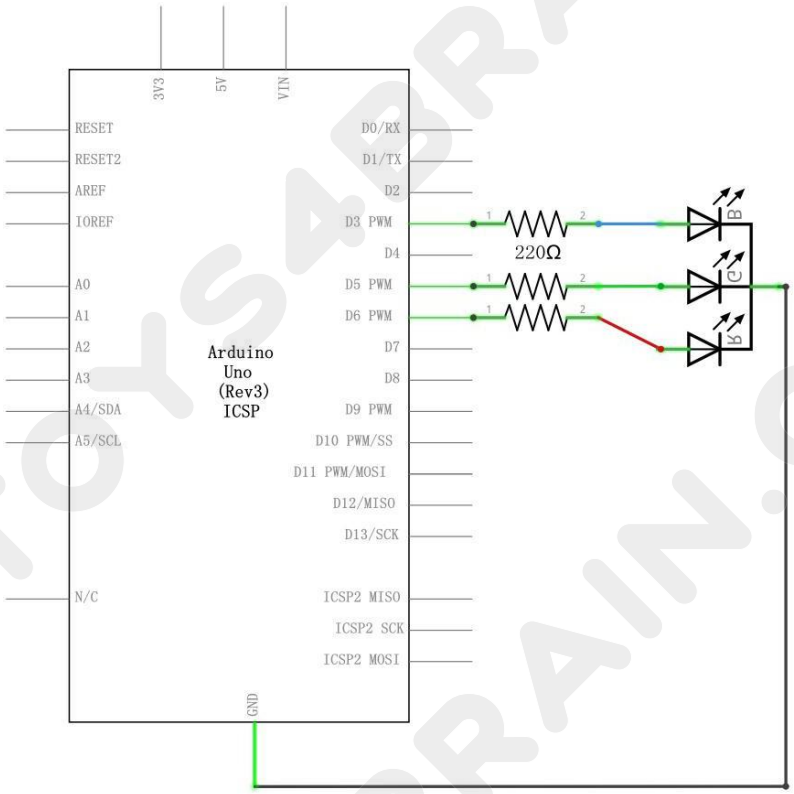
The RGB LED has four leads. There is one lead going to the positive connection of each of the single LEDs within the package and a single lead that is connected to all three negative sides of the LEDs.



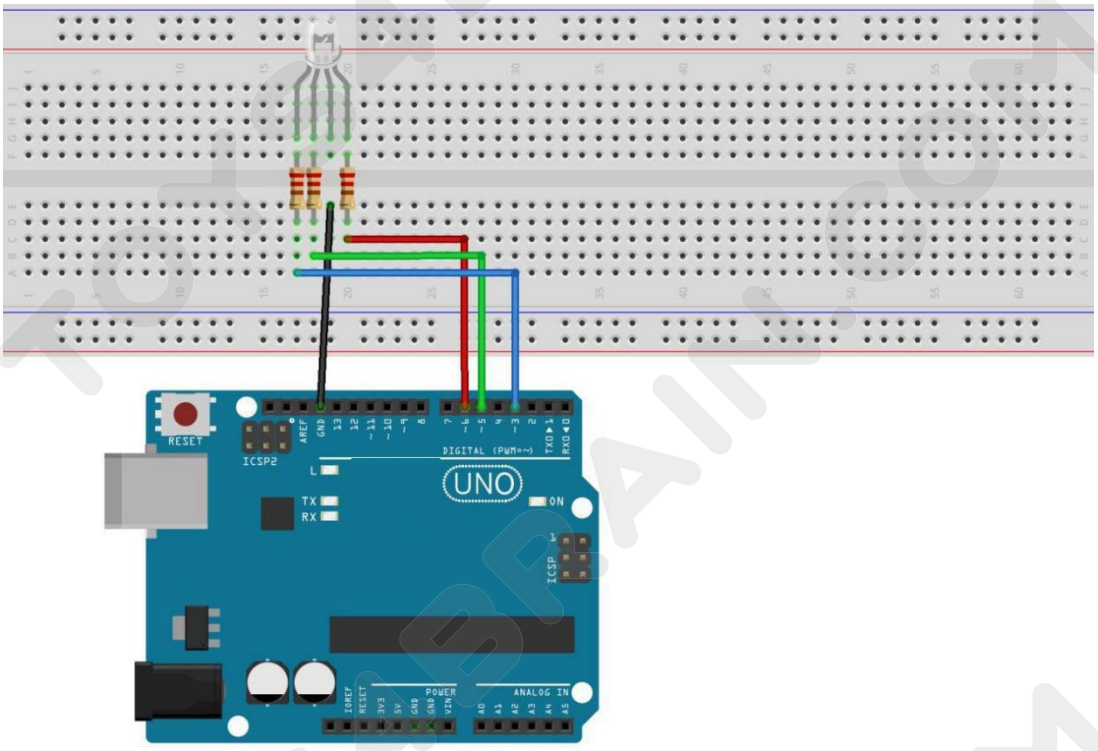
Here on the photographs you can see 4 electrode LED. Every separate pin for Green or Blue or Red color is called Anode. You will always connect “+” to it. Cathode goes to “-“(ground). If you connect it other way round the LED will not light.

The common negative connection of the LED package is the second pin from the flat side. It is also the longest of the four leads and will be connected to the ground. Each LED inside the package requires its own 220Ω resistor to prevent too much current flowing through it. The three positive leads of the LEDs (one red, one green and one blue) are connected to UNO output pins using these resistors.

Connection Diagram:



Wiring schematic:



Physical wiring diagram:

Code:

Our code will use FOR loops to cycle through the colors.

The first FOR loop will go from RED to GREEN.

The second FOR loop will go from GREEN to BLUE.

The last FOR loop will go from BLUE to RED.

Try the sketch out and then we will dissect it in some detail.....

The sketch starts by specifying which pins are going to be used for each of the colors:

```
// Define Pins
#define BLUE 3
#define GREEN 5
#define RED 6
```

The next step is to write the 'setup' function. As we have learnt in earlier lessons,

the setup function runs just once after the Arduino has reset. In this case, all it has to do is define the three pins we are using as being outputs.

```
void setup()
{
pinMode(RED, OUTPUT);
pinMode(GREEN, OUTPUT);
pinMode(BLUE, OUTPUT);
digitalWrite(RED, HIGH);
digitalWrite(GREEN, LOW);
digitalWrite(BLUE, LOW);
}
```

Before we take a look at the 'loop' function, let's look at the last function in the sketch.

The define variables

```
redValue = 255; // choose a value between 1 and 255 to change the color.
greenValue = 0;
blueValue = 0;
```

This function takes three arguments, one for the brightness of the red, green and blue LEDs. In each case the number will be in the range 0 to 255, where 0 means off and 255 means maximum brightness. The function then calls 'analogWrite' to set the brightness of each LED.

If you look at the 'loop' function you can see that we are setting the amount of red, green and blue light that we want to display and then pausing for a second before moving on to the next color.

```
#define delayTime 10 // fading time between colors
Delay(delayTime);
```

Try adding a few colors of your own to the sketch and watch the effect on your LED.

Lesson 6 Active buzzer

Overview:

In this lesson, you will learn how to generate a sound with an active buzzer.

Component Required:

1 x Arduino Uno R3

1 x Active buzzer

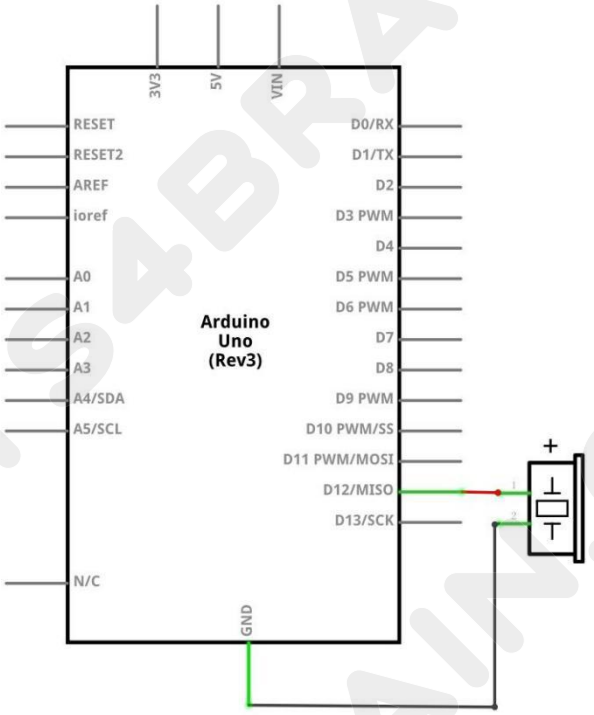
1 x 830 Tie Points Breadboard

2 x M-M wires (Male to Male jumper wires)

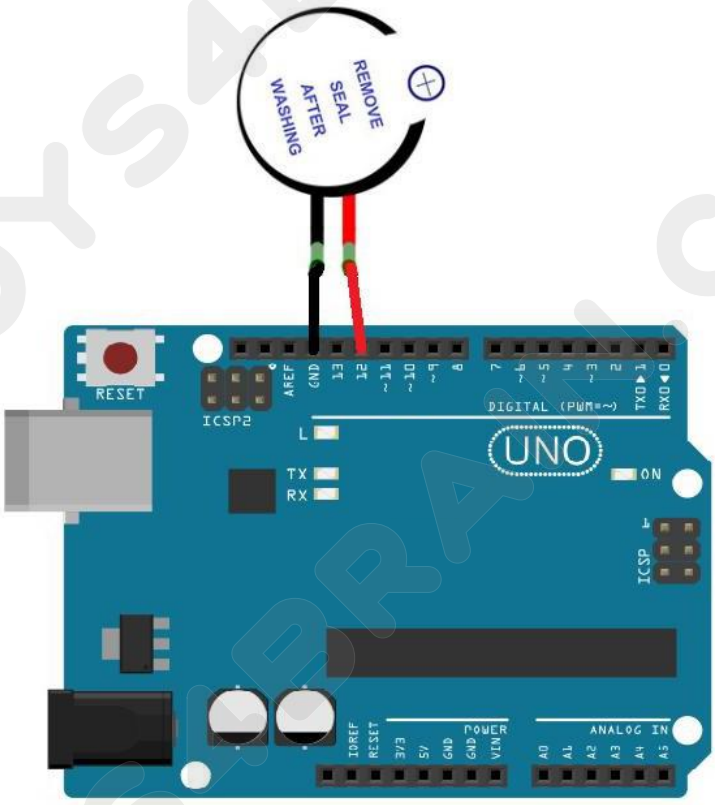
BUZZER:

Electronic buzzers are DC-powered and equipped with an integrated circuit. They are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive ones. Turn the pins of two buzzers face up. The one with a green circuit board is a passive buzzer, while the other enclosed with a black tape is an active one.

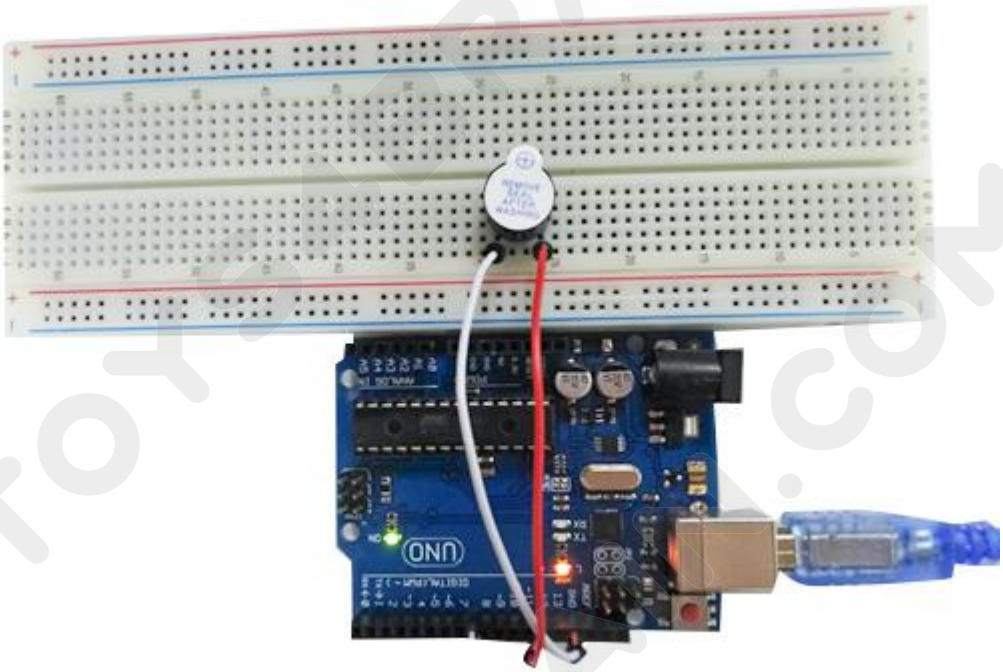
Connection Diagram:



Wiring schematic:



Physical wiring diagram:



Code：

After wiring, open the program in the code folder - Lesson 6, you need to install the relevant library file, then click Upload to upload the program.

```
#include "pitches.h"// notes in the melody:
int melody[] = {
NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_B5, NOTE_C6};
int duration = 500;    // 500 miliseconds

void setup() {    }

void loop() {
   for (int thisNote = 0; thisNote < 12; thisNote++) {
      // pin12 output the voice, every scale is 0.5 sencond
      tone(12, melody[thisNote], duration);

      // Output the voice after several minutes
      delay(1000);
   } // restart after two seconds
   delay(2000);
}
```

Lesson 7 Passive Buzzer

Overview

In this lesson, you will learn how to use a passive buzzer.

The purpose of the experiment is to generate eight different sounds, each sound lasting 0.5 seconds: from Alto Do (523Hz), Re (587Hz), Mi (659Hz), Fa (698Hz), So (784Hz), La (880Hz), Si (988Hz) to Treble Do (1047Hz).

Component Required:

1 x Arduino UNO R3

1 x Passive buzzer

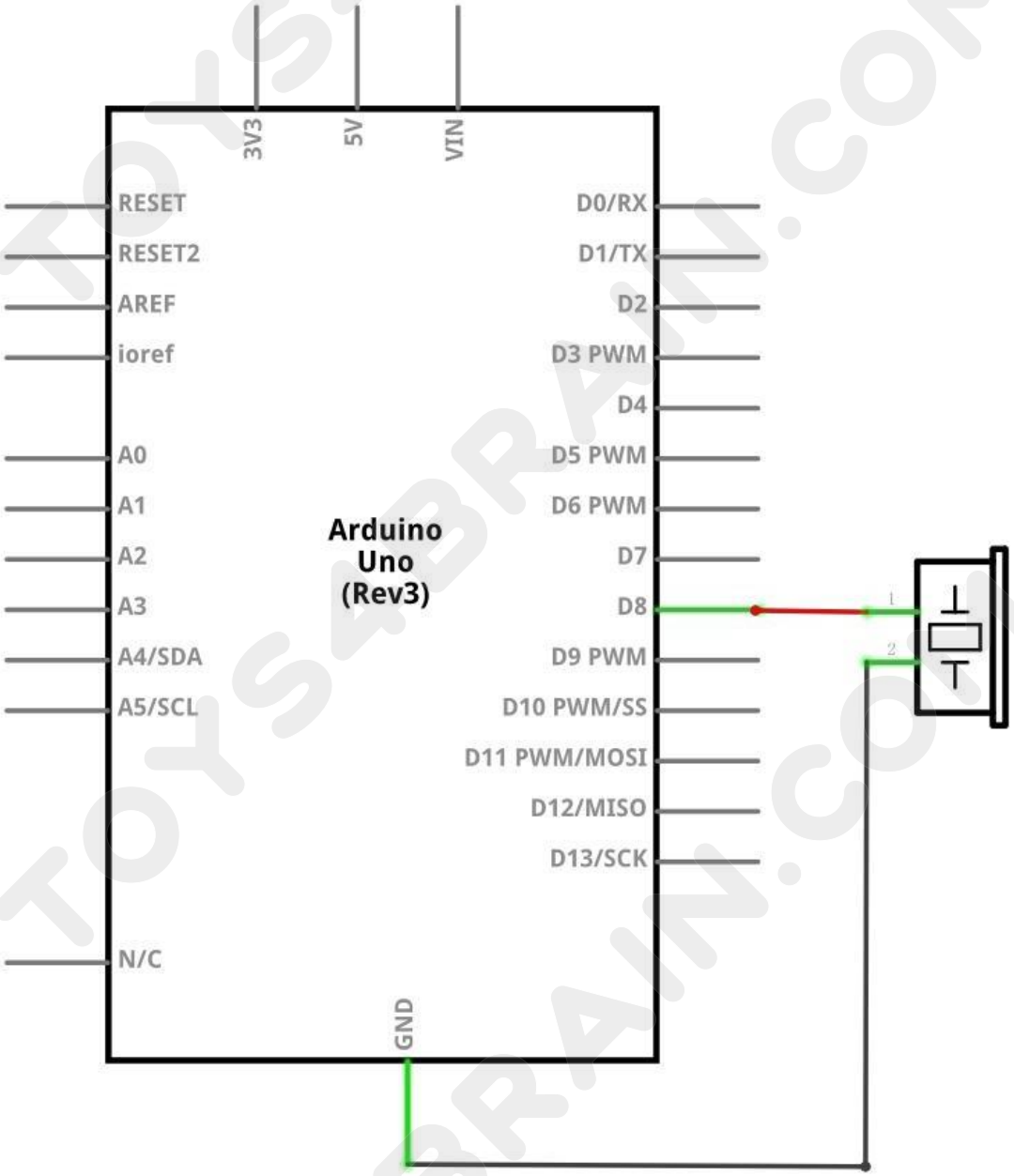
2 x F-M wires (Female to Male DuPont wires)

Passive Buzzer:

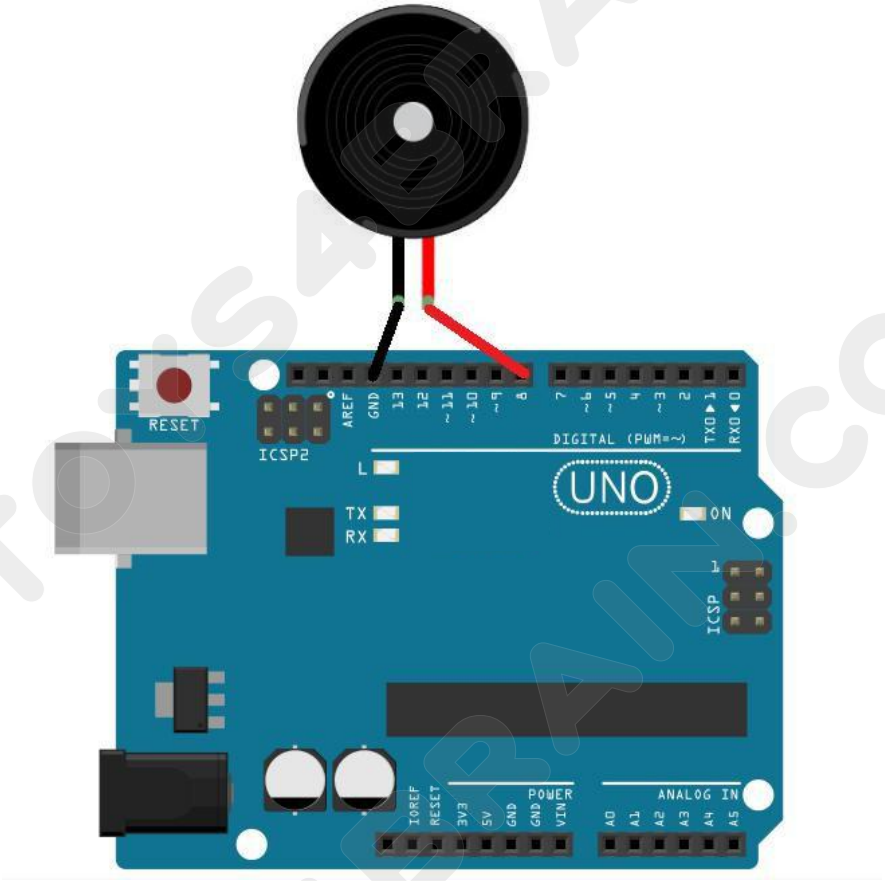
The working principle of passive buzzer is using PWM generating audio to make the air to vibrate. Appropriately changed as long as the vibration frequency, it can generate different sounds. For example, sending a pulse of 523Hz, it can generate Alto Do, pulse of 587Hz, it can generate midrange Re, pulse of 659Hz, it can produce midrange Mi. By the buzzer, you can play a song.

We should be careful not to use the UNO R3 board analog Write () function to generate a pulse to the buzzer, because the pulse output of analog Write () is fixed (500Hz).

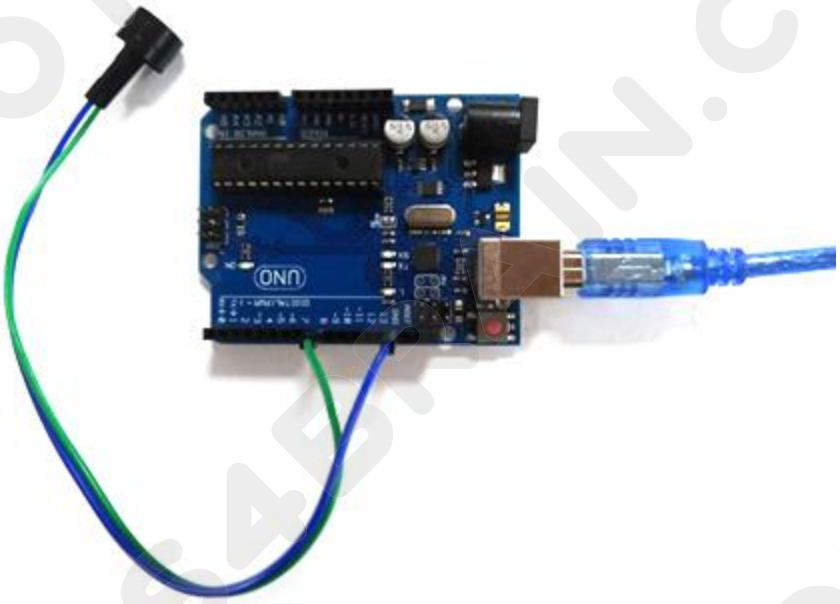
Connection Diagram:



Wiring schematic:



Physical wiring diagram:



Code:

```
void setup()
{
}

void loop()
{
for(int i=200;i<=800;i++)    //Increase the frequency from 200HZ to 800HZ in a loop
{
   pinMode(8,OUTPUT);
   tone(8,i);                                    //Output frequency on port 8
delay(5);                                        //This frequency is maintained for 5
milliseconds
}
delay(4000);                          //Hold for 4 seconds at the highest frequency
for(int i=800;i>=200;i--)
{
   pinMode(8,OUTPUT);
   tone(8,i);
delay(10);
}
}
```

In this lesson we used a new function tone(). For details on how to use it, please refer to the tone() in the Arduino grammar manual.

https://www.arduino.cc/reference/en/

Lesson 8 Tilt Ball Switch

Overview:

In this lesson, you will learn how to use a tilt ball switch in order to detect small angle of inclination.

Component Required:

1 x Arduino Uno R3
1 x Tilt Ball switch
2 x F-M wires (Female to Male DuPont wires)

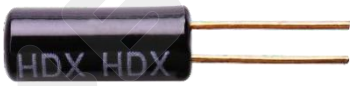
Component Introduction

Tilt sensor:

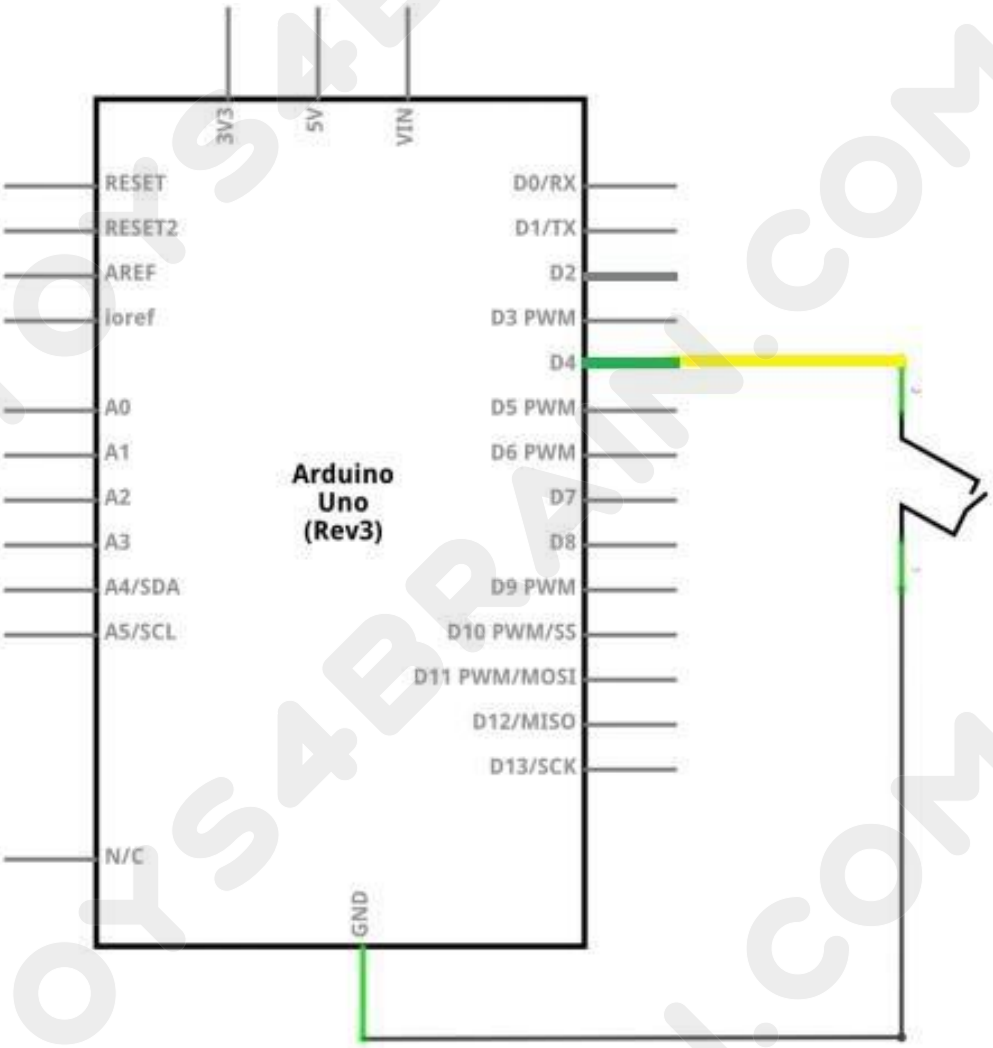
Tilt sensors (tilt ball switch) allow you to detect orientation or inclination. They are small, inexpensive, low-power and easy-to-use. If used properly, they will not wear out. Their simplicity makes them popular for toys, gadgets and appliances. Sometimes, they are referred to as "mercury switches", "tilt switches" or "rolling ball sensors" for obvious reasons.

They are usually made up of a cavity of some sort (cylindrical is popular, although not always) with a conductive free mass inside, such as a blob of mercury or rolling ball. One end of the cavity has two conductive elements (poles). When the sensor is oriented so that that end is downwards, the mass rolls onto the poles and shorts them, acting as a switch throw.

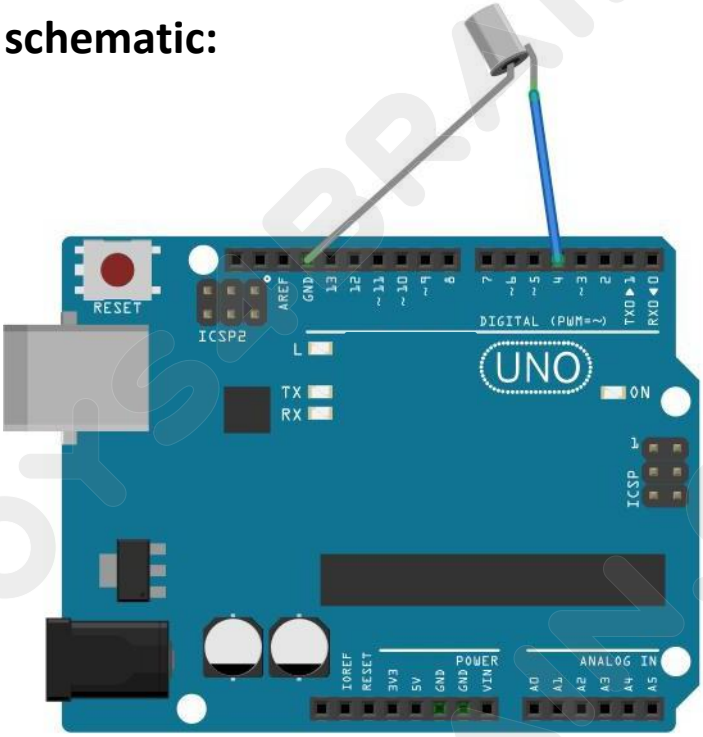
While not as precise or flexible as a full accelerometer, tilt switches can detect motion or orientation. Another benefit is that the big ones can switch power on their own. Accelerometers, on the other hand, output digital or analog voltage that must then be analyzed using extra circuitry.



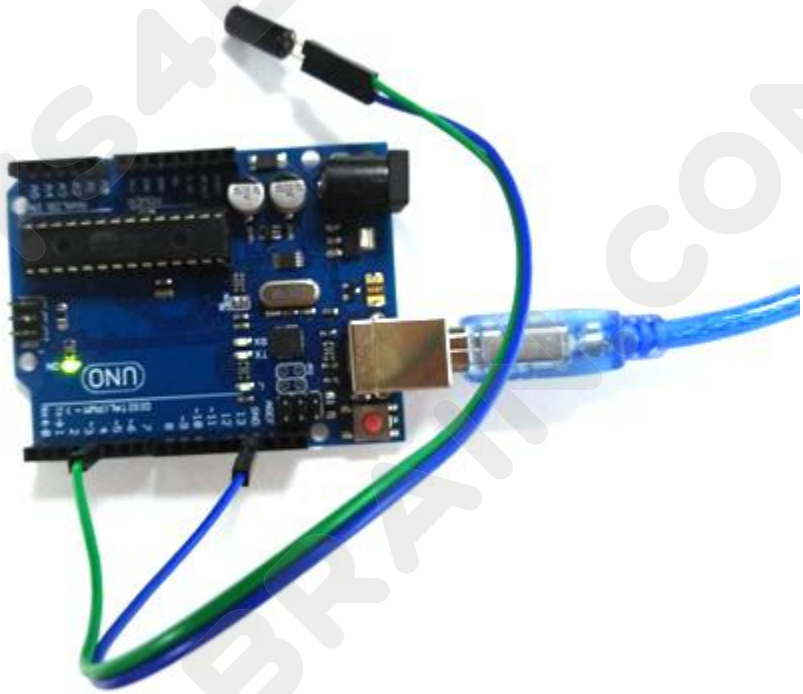
Connection Diagram:



Wiring schematic:



Physical wiring diagram:



Code:

```
const int ledPin = 13;    //the led attach to

void setup()
{
   pinMode(ledPin,OUTPUT);    //initialize the ledPin as an output

   pinMode(4,INPUT);

   digitalWrite(4, HIGH);
}
/**************************************/
void loop()
{
   int digitalVal = digitalRead(4);

   if(HIGH == digitalVal)
   {
      digitalWrite(ledPin,LOW);      //turn the led off
   }
   else
   {
      digitalWrite(ledPin,HIGH);    //turn the led on

   }
}
```

In this lesson we use a new function if...else(). For details on how to use it, please refer to the Arduino Syntax Manual.

https://www.arduino.cc/reference/en/

Lesson 9 Ultrasonic Sensor Module

Overview:

Ultrasonic sensor is great for all kind of projects that need distance measurements, avoiding obstacles as examples.
The HC-SR04 is inexpensive and easy to use since we will be using a Library specifically designed for these sensor.

Component Required:

1 x Arduino Uno R3

1 x Ultrasonic sensor module

4 x F-M wires (Female to Male DuPont wires)

Component Introduction

Ultrasonic sensor

Ultrasonic sensor module HC-SR04 provides 2cm-400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

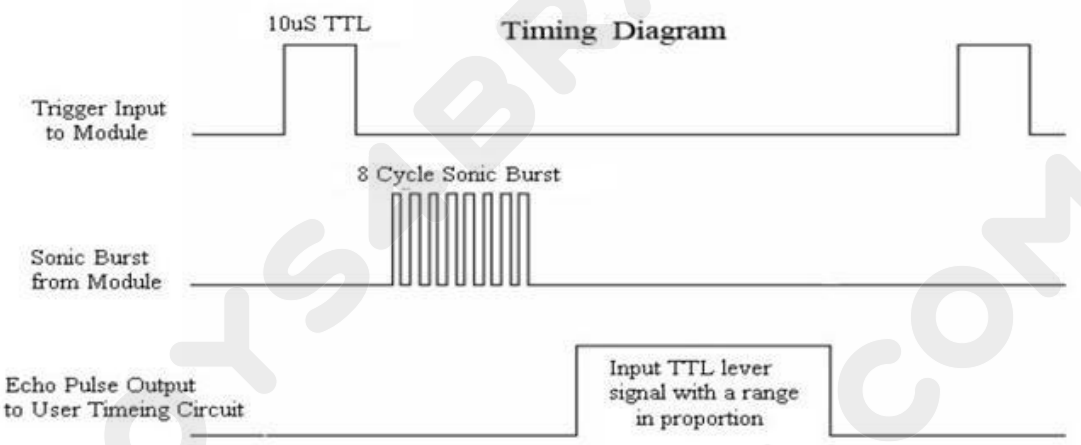
(1) Using IO trigger for at least 10us high level signal,

(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

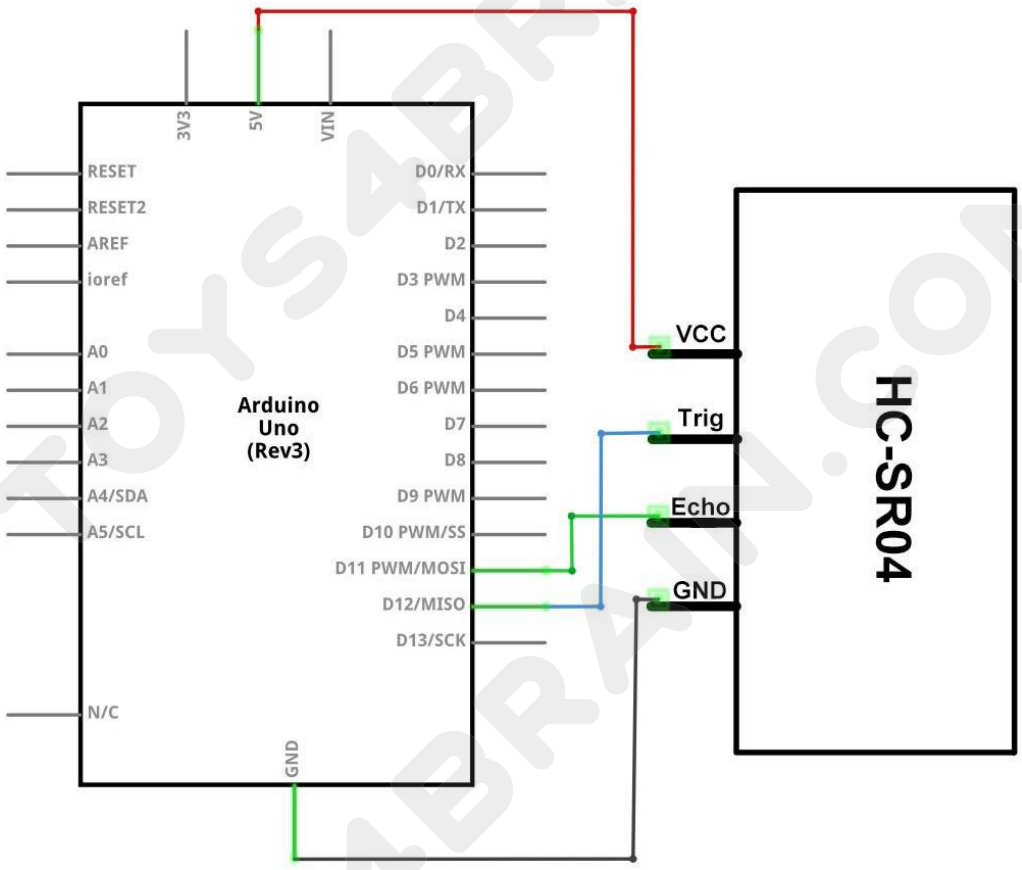
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic tore turning.

Test distance = (high level time × velocity of sound (340m/s) /2

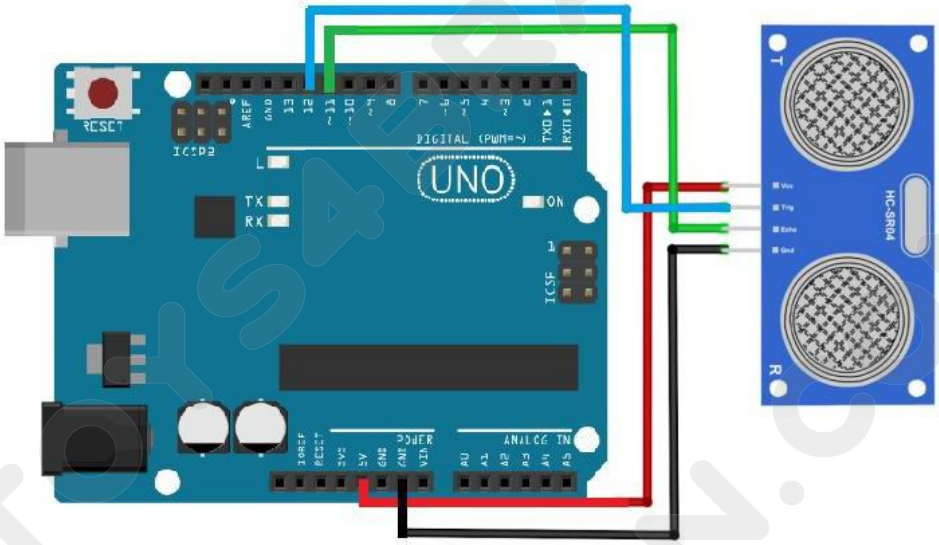
The Timing diagram is shown below. You only need to supply a short 10us pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: us / 58 = centimeters or us / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Connection Diagram:

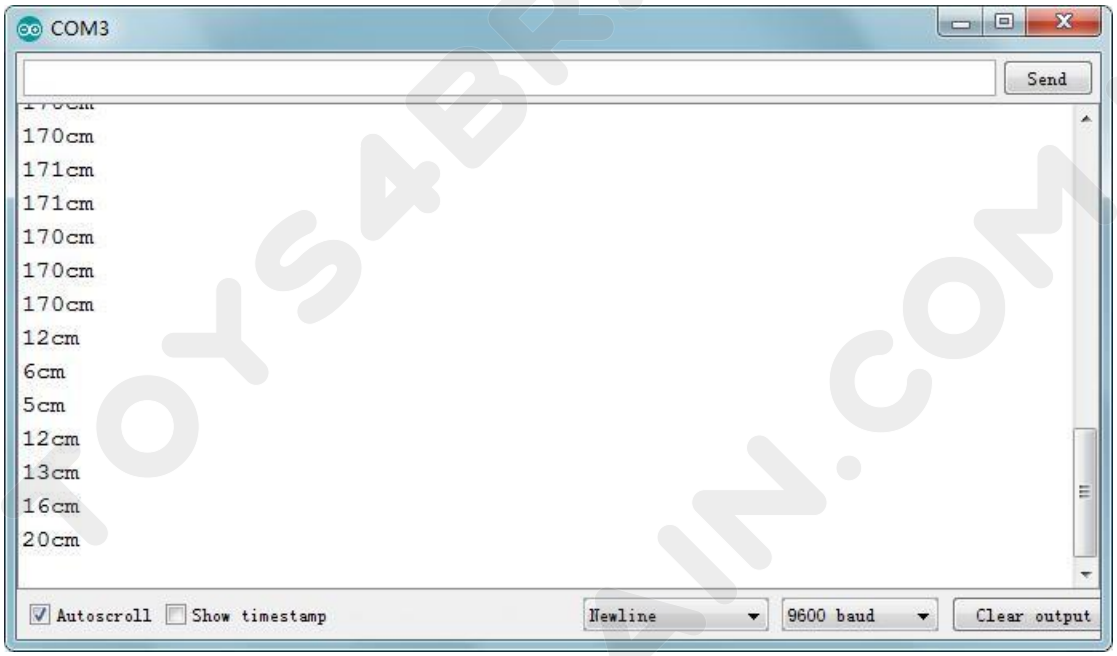


Wiring schematic:



Physical wiring diagram:

Open the serial monitor as follows:



Code:

After wiring, open the program in the code folder - Lesson 9, you need to install the relevant library file, then click Upload to upload the program.

```
#include "SR04.h"
#define TRIG_PIN 12
#define ECHO_PIN 11
SR04 sr04 = SR04(ECHO_PIN,TRIG_PIN);
long a;

void setup()
    { Serial.begin(9600)
    ; delay(1000);
}

void loop() {
    a=sr04.Distance();
    Serial.print(a);
    Serial.println("cm");
    delay(1000);
}
```

Lesson 10 DHT11 Temperature and Humidity Sensor

Overview:

In this tutorial we will learn how to use a DHT11 Temperature and Humidity Sensor.

It's accurate enough for most projects that need to keep track of humidity and temperature readings.

Again we will be using a Library specifically designed for these sensors that will make our code short and easy to write.

Component Required:

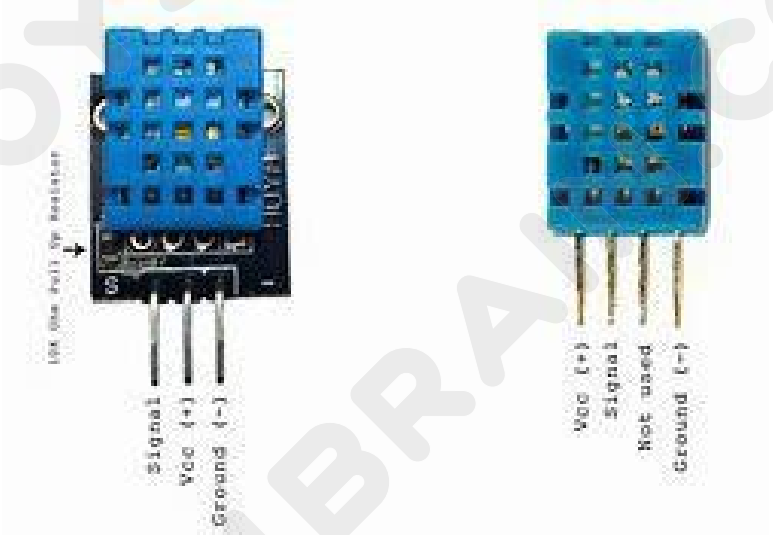
1 x Arduino Uno R3

1 x DHT11 Temperature and Humidity module

3 x F-M wires (Female to Male DuPont wires)

Component Introduction:

Temp and humidity sensor:



Product parameters:

Relative humidity:

Resolution: 16Bit

Repeatability: ±1% RH

Accuracy: At 25℃ ±5% RH

Interchangeability: fully interchangeable

Response time: 1 / e (63%) of 25℃ 6s

1m / s air 6s

Hysteresis: <± 0.3% RH

Long-term stability: <± 0.5% RH / yr in

Temperature:

Resolution: 16Bit

Repeatability: ±0.2℃

Range: At 25℃ ±2℃

Response time: 1 / e (63%) 10S

Electrical Characteristics

Power supply: DC 3.5～5.5V

Supply Current: measurement 0.3mA standby 60μA

Sampling period: more than 2 seconds

Pin Description:

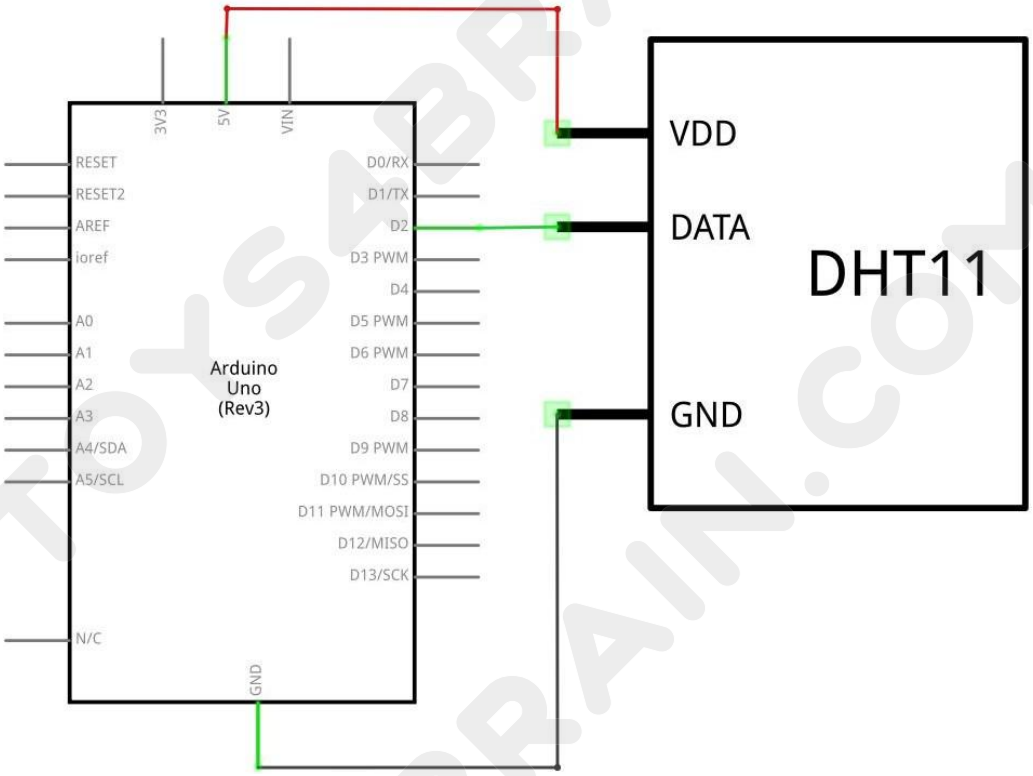
1. the VDD power supply 3.5～5.5V DC

2. DATA serial data, a single bus

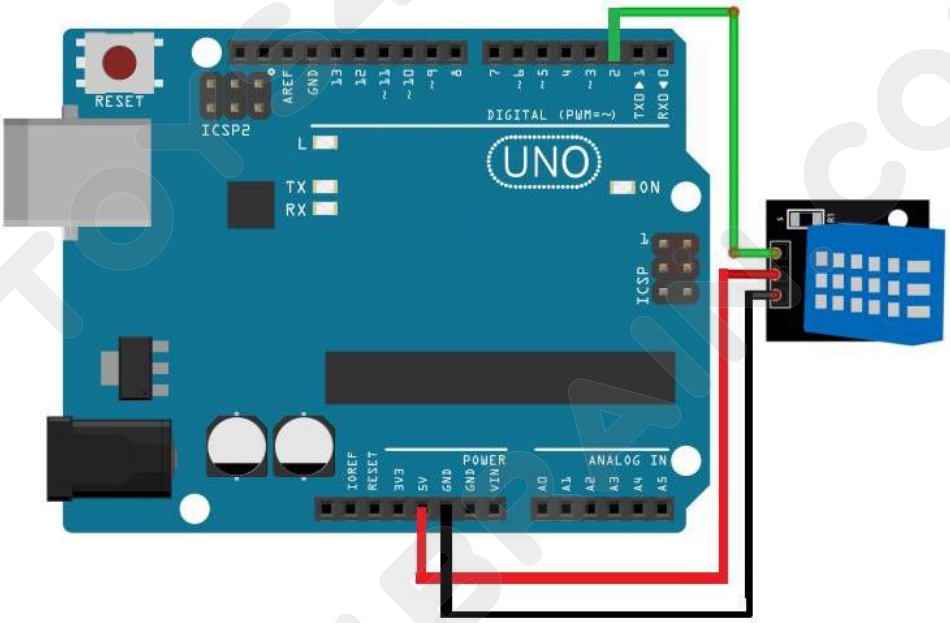
3. NC, empty pin

4. GND ground, the negative power

Connection Diagram:

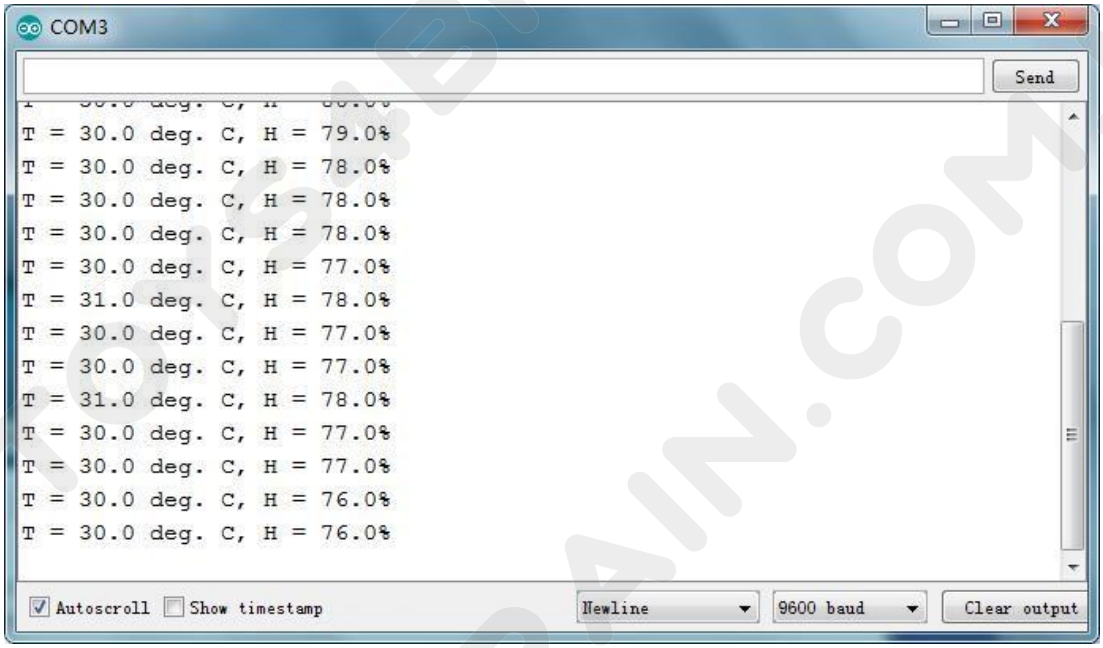


Wiring schematic:



Physical wiring diagram:

Open the serial monitor as follows:



Code:

After wiring, open the program in the code folder - Lesson 10, you need to install the relevant library file, then click Upload to upload the program.

```
#include <dht_nonblocking.h>
#define DHT_SENSOR_TYPE DHT_TYPE_11
static const int DHT_SENSOR_PIN = 2;
DHT_nonblocking dht_sensor( DHT_SENSOR_PIN, DHT_SENSOR_TYPE );

/*
 * Initialize the serial port.
 */
void setup( )
{
   Serial.begin( 9600);
}

/*
 * Poll for a measurement, keeping the statemachine alive.    Returns
 * true if a measurement is available.
 */
static bool measure_environment( float *temperature, float *humidity )
{
   static unsigned long measurement_timestamp = millis( );

   /* Measure once every four seconds. */
   if( millis( ) - measurement_timestamp > 3000ul )
   {
      if( dht_sensor.measure( temperature, humidity ) == true )
      {
         measurement_timestamp = millis( );
         return( true );
      }
   }

   return( false );
}



/*
 * Main program loop.
 */
```

```
void loop( )
{
   float temperature;
   float humidity;

   /* Measure temperature and humidity.   If the functions returns
        true, then a measurement is available. */
   if( measure_environment( &temperature, &humidity ) == true )
   {
      Serial.print( "T = " );
      Serial.print( temperature, 1 );
      Serial.print( " deg. C, H = " );
      Serial.print( humidity, 1 );
      Serial.println( "%" );
   }
}
```

Lesson 11 Servo

Overview:

The servo motor has three wires: power, ground, and signal. The power cord is usually red and should be connected to the 5V pin on the Arduino board. The ground wire is usually black or brown and should be connected to the ground pin on the Arduino board. The signal pins are usually yellow, orange or white and should be connected to the digital pins on the Arduino board. Please note that the servo system consumes considerable power, so if you need to drive more than one or two, you may need to use a separate power supply (ie the +5V pin on the Arduino) to power them. Be sure to connect the Arduino to the ground of the external power supply.

Component Required:

1 x Arduino Uno R3

1 x Servo (SG90)

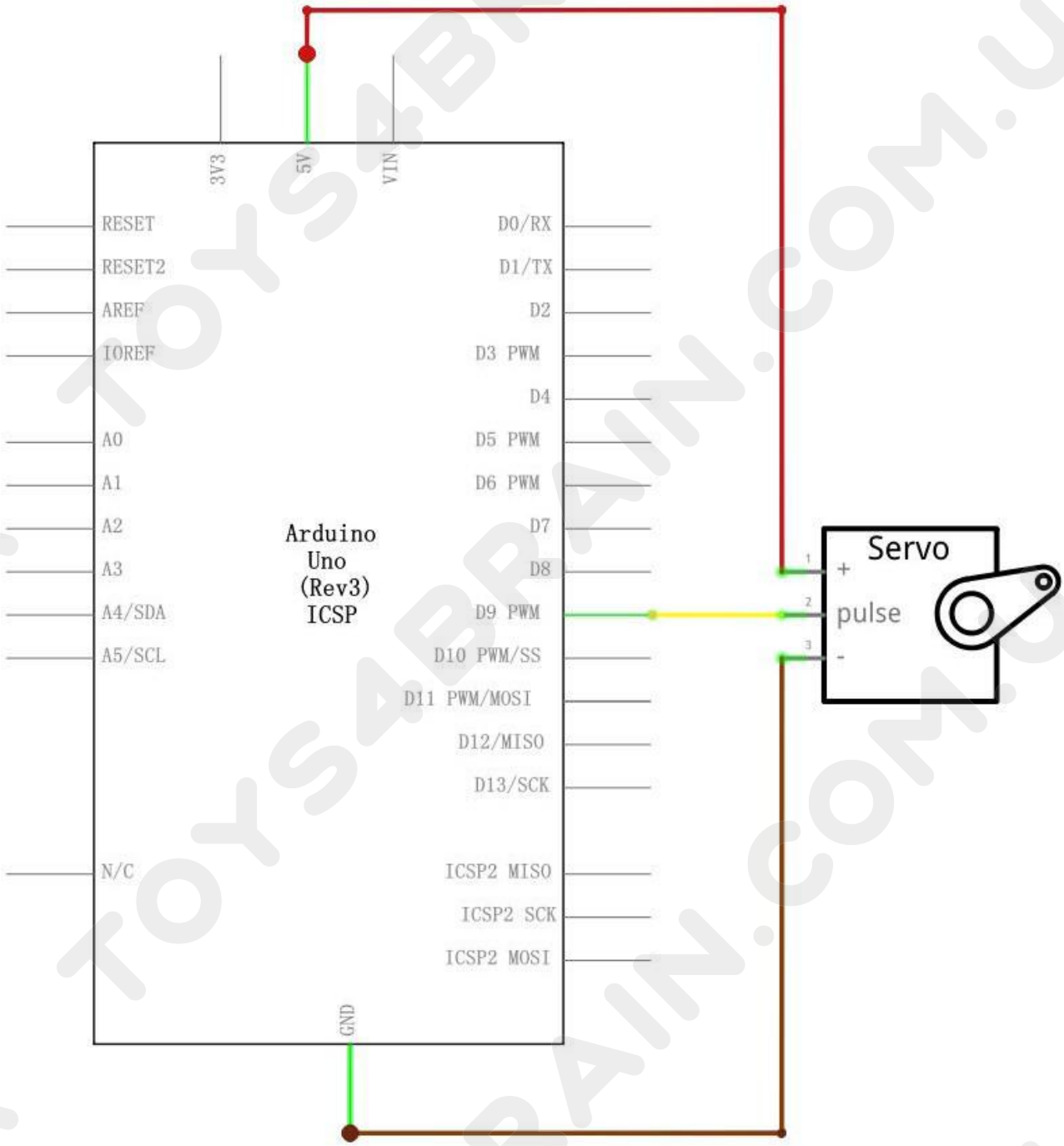
3 x M-M wires (Male to Male jumper wires)

Component Introduction

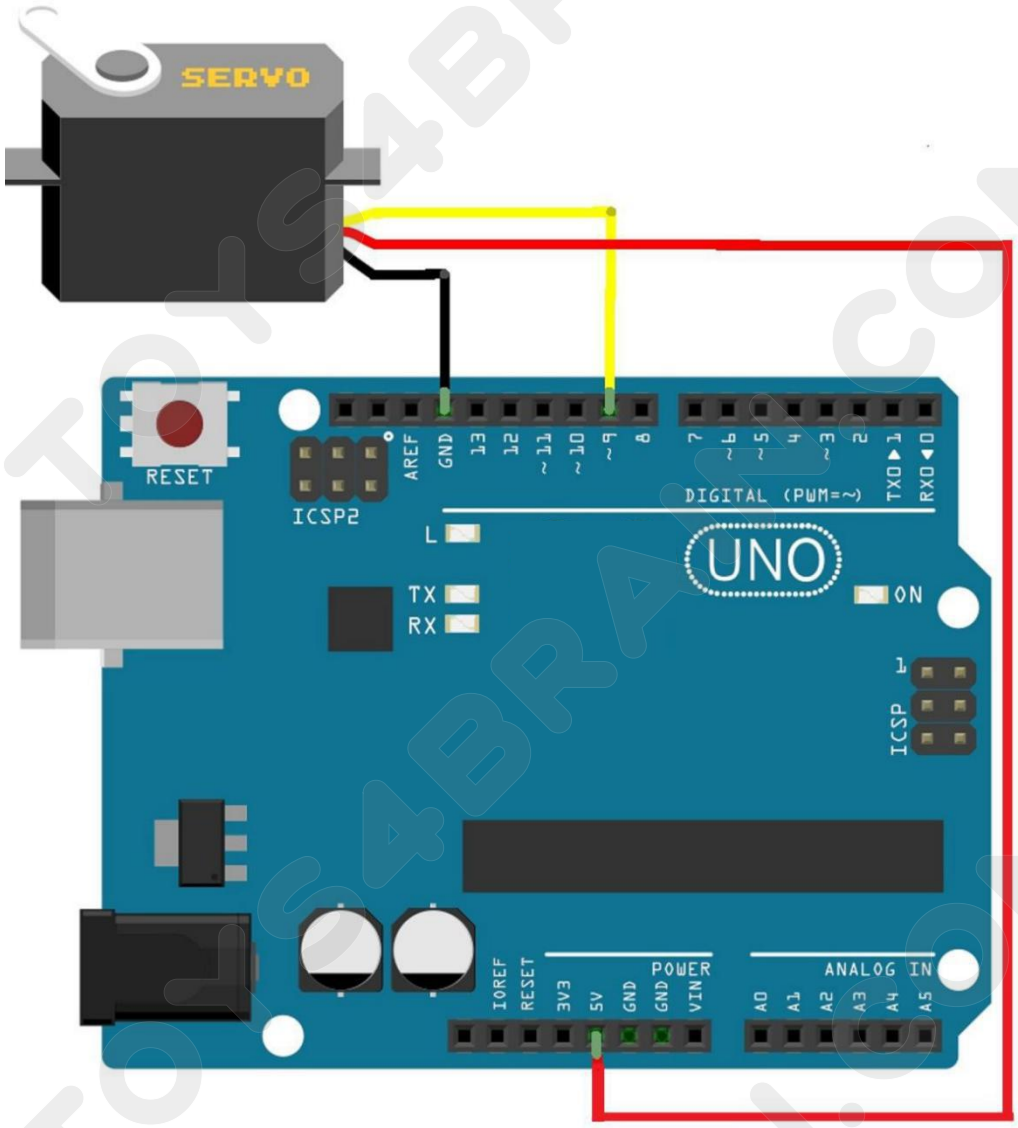
SG90

- Universal for JR and FP connector
- Cable length : 25cm
- No load; Operating speed: 0.12 sec / 60 degree (4.8V), 0.10 sec / 60 degree (6.0V)
- Stall torque (4.8V): 1.6kg/cm
- Temperature : -30~60'C
- Dead band width: 5us
- Working voltage: 3.5~6V
- Dimension : 1.26 in x 1.18 in x 0.47 in (3.2 cm x 3 cm x 1.2 cm)
- Weight : 4.73 oz (134 g)

Connection Diagram:



Wiring schematic:



Physical wiring diagram:

Code:

After wiring, open the program in the code folder - Lesson11, you need to install the relevant library file, then click Upload to upload the program.

```
#include <Servo.h>
Servo myservo;    // create servo object to control a servo
// twelve servo objects can be created on mostboards
int pos = 0;      // variable to store the servo position
void setup() {
  myservo.attach(9);    // attaches the servo on pin 9 to the servo object
}
void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);                 // tell servo to go to position in variable 'pos'
    delay(15);                          // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);                 // tell servo to go to position in variable 'pos'
    delay(15);                          // waits 15ms for the servo to reach the position
  }
}
```

Lesson 12 Stepper Motor

Overview:

In this lesson, you will learn a fun and easy way to drive a stepper motor.
The stepper we are using comes with its own driver board making it easy to connect to our UNO.

Component Required:

1 x Arduino Uno R3

1 x 830 tie-points breadboard

1 x ULN2003 stepper motor driver module

1 x Stepper motor

1 x 9V1A Adapter

1 x Power supply module

6 x F-M wires (Female to Male DuPont wires)

1 x M-M wire (Male to Male jumper wire)

Component Introduction:

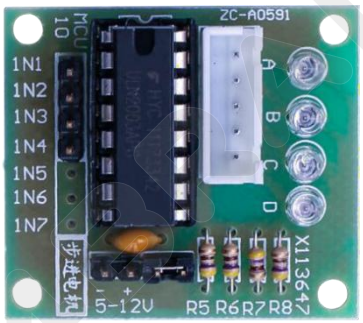
Stepper Motor:

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied. One of the most significant advantages of a stepper motor is its ability to be accurately controlled in an open loop system. Open loop control means no feedback information about position is needed. This type of control eliminates the need for expensive sensing and feedback devices such as optical encoders. Your position is known simply by keeping track of the input step pulses.

Stepper motor 28BYJ-48 Parameters

- Model: 28BYJ-48
- Rated voltage: 5VDC
- Number of Phase: 4
- Speed Variation Ratio: 1/64
- Stride Angle: 5.625° /64
- Frequency: 100Hz
- DC resistance: 50Ω±7%(25℃)
- Idle In-traction Frequency: > 600Hz
- Idle Out-traction Frequency: > 1000Hz
- In-traction Torque >34.3mN.m(120Hz)
- Self-positioning Torque >34.3mN.m
- Friction torque: 600-1200 gf.cm
- Pull in torque: 300 gf.cm
- Insulated resistance >10MΩ(500V)
- Insulated electricity power：600VAC/1mA/1s
- Insulation grade： A
- Rise in Temperature <40K(120Hz)
- Noise <35dB(120Hz,No load,10cm)

ULN2003 Driver Board

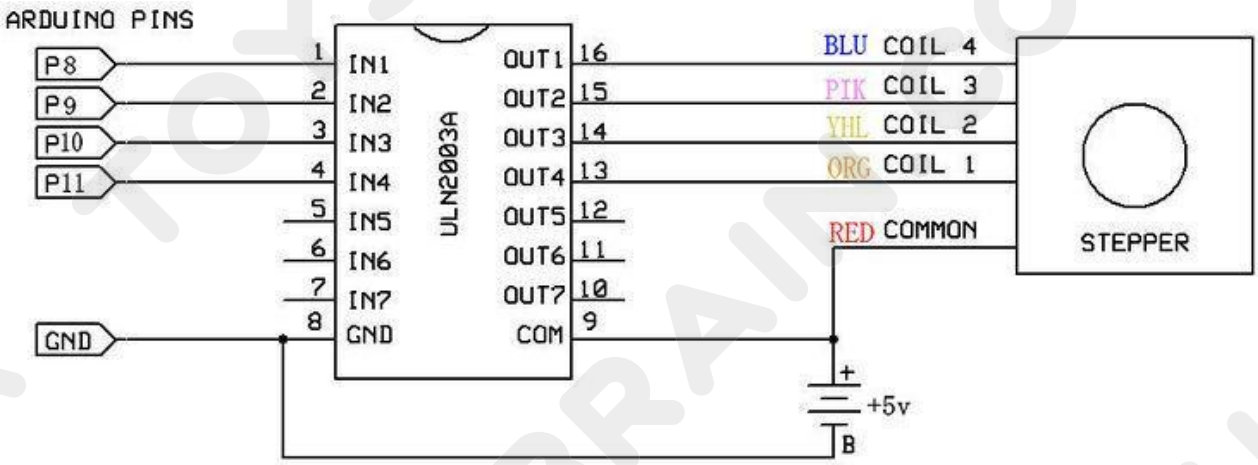


Product Description

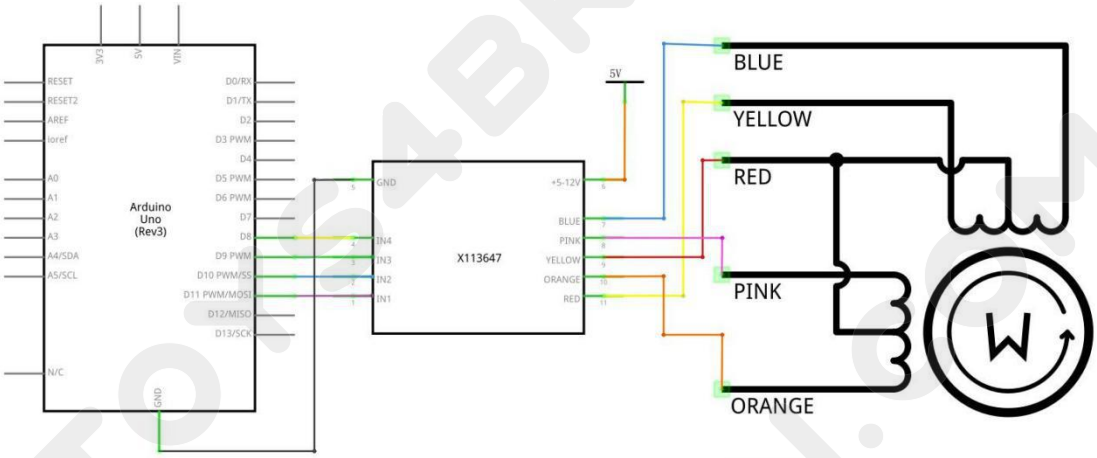
1. Size: 42mmx30mm
2. Use ULN2003 driver chip, 500mA
3. A. B. C. D LED indicating the four phase stepper motor working condition.
4. White jack is the four phase stepper motor standard jack.
5. Power pins are separated
6. We kept the rest pins of the ULN2003 chip for your further prototyping.

The simplest way of interfacing a unipolar stepper to Arduino is to use a breakout for ULN2003A transistor array chip. The ULN2003A contains seven Darlington transistor drivers and is somewhat like having seven TIP120 transistors all in one package. The ULN2003A can pass up to 500 mA per channel and has an internal voltage drop of about 1V when on. It also contains internal clamp diodes to dissipate voltage spikes when driving inductive loads. To control the stepper, apply voltage to each of the coils in a specific sequence.

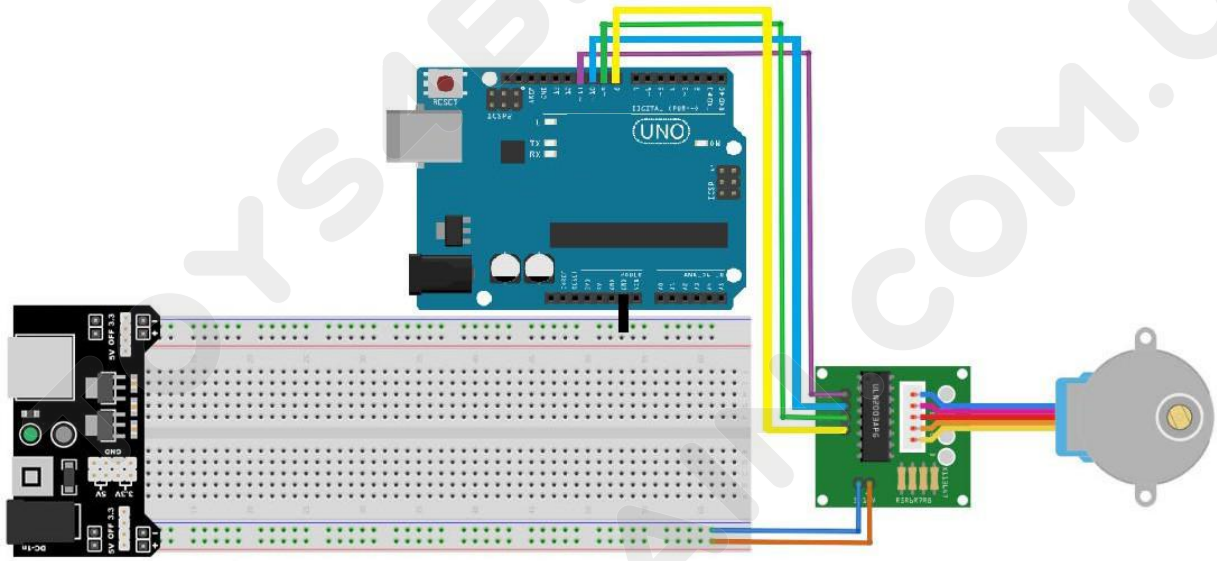
Here are schematics showing how to interface a unipolar stepper motor to four controller pins using a ULN2003A, and showing how to interface using four com



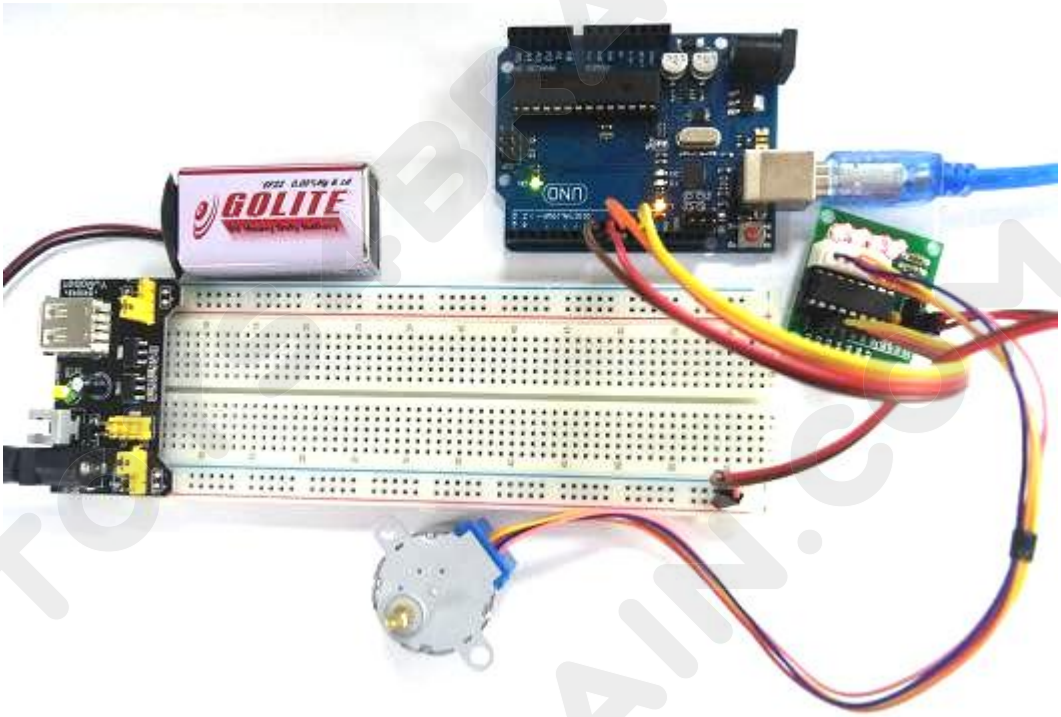
Connection Diagram:



Wiring schematic:



Physical wiring diagram:



Code:

After wiring, open the program in the code folder - Lesson 12 , you need to install the relevant library file, then click Upload to upload the program.

```
#include <Stepper.h>

const int stepsPerRevolution = 1500;
// change this to fit the number of steps per revolution
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);
void setup() {
   // set the speed at 20 rpm:
   myStepper.setSpeed(20);
   // initialize the serial port:
   Serial.begin(9600);    }
void loop() { // step one revolution    in one direction:
   Serial.println("clockwise");
   myStepper.step(stepsPerRevolution);
   delay(500);// step one revolution in the other direction:
   Serial.println("counterclockwise");
   myStepper.step(-stepsPerRevolution);
   delay(500);
}
```

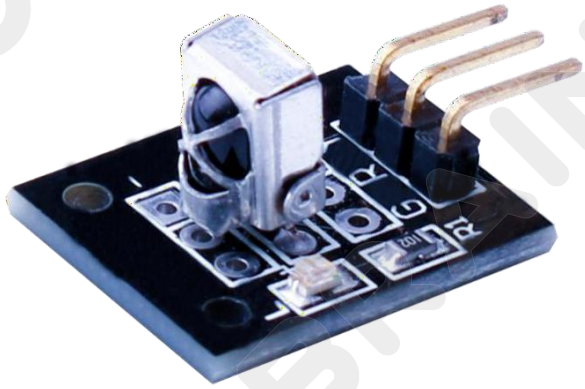
Lesson 13 IR Receiver Module

Overview

Using an IR Remote is a great way to have wireless control of your project.

Infrared remotes are simple and easy to use. In this tutorial we will be connecting the IR receiver to the UNO, and then use a Library that was designed for this particular sensor.

In our sketch we will have all the IR Hexadecimal codes that are available on this remote, we will also detect if the code was recognized and also if we are holding down a key.



Component Required:

1 x Arduino Uno R3

1x IR receiver module

1 x IR remote

3 x F-M wires (Female to Male DuPont wires)

Component Introduction

IR RECEIVER SENSOR:

IR detectors are little microchips with a photocell that are tuned to listen to infrared light. They are almost always used for remote control detection - every TV and DVD player has one of these in the front to listen for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels. IR light is not visible to the human eye,

which means it takes a little more work to test a setup.

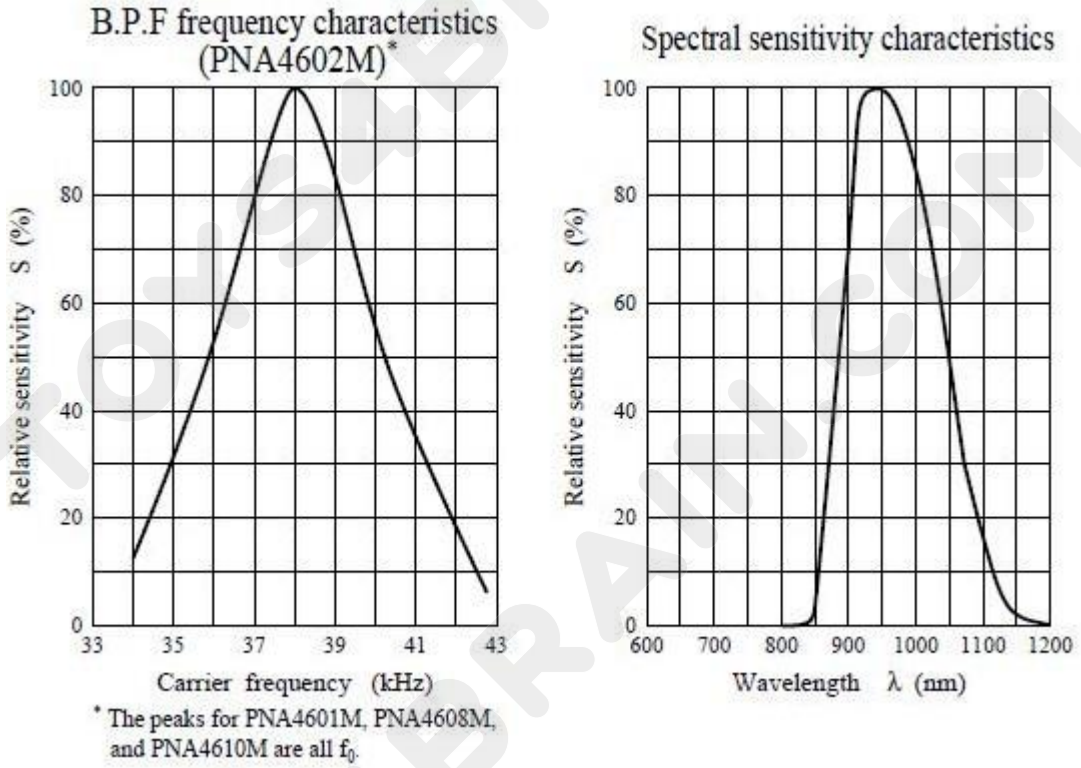
There are a few difference between these and say a CdS Photocells:

IR detectors are specially filtered for IR light, they are not good at detecting visible light. On the other hand, photocells are good at detecting yellow/green visible light, and are not good at IR light.

IR detectors have a demodulator inside that looks for modulated IR at 38 KHz. Just shining an IR LED won't be detected, it has to be PWM blinking at 38KHz. Photocells do not have any sort of demodulator and can detect any frequency (including DC) within the response speed of the photocell (which is about 1KHz)

IR detectors are digital out - either they detect 38KHz IR signal and output low (0V) or they do not detect any and output high (5V). Photocells act like resistors, the resistance changes depending on how much light they are exposed to.

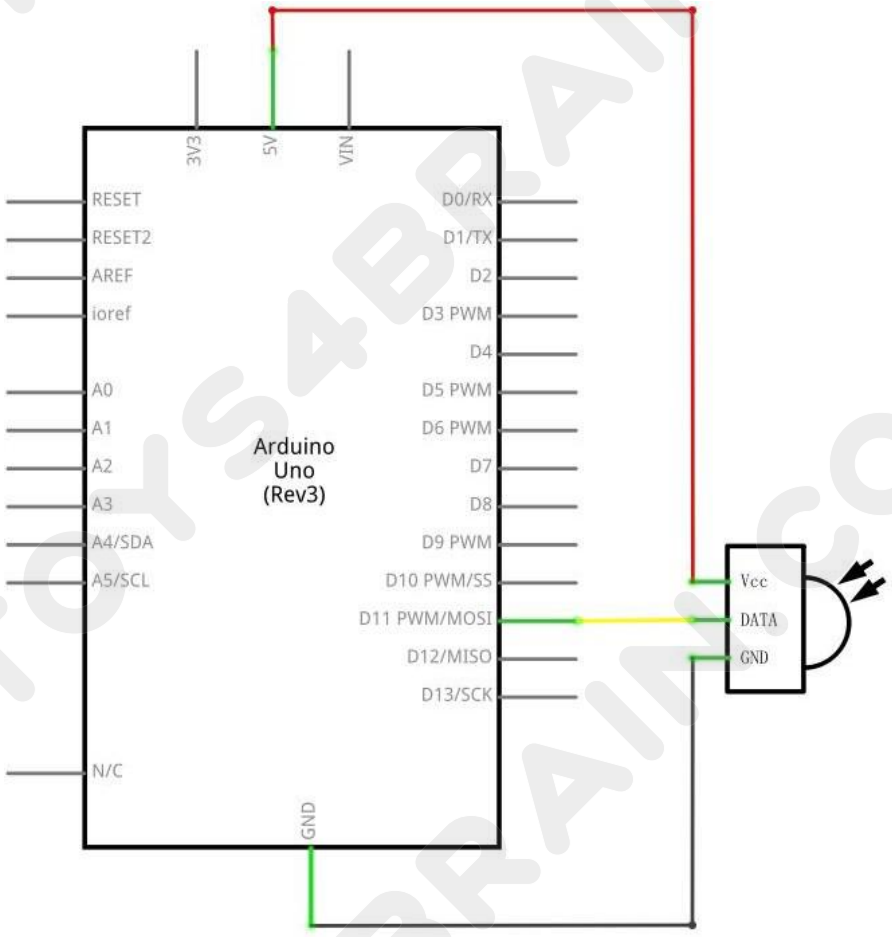
What You Can Measure



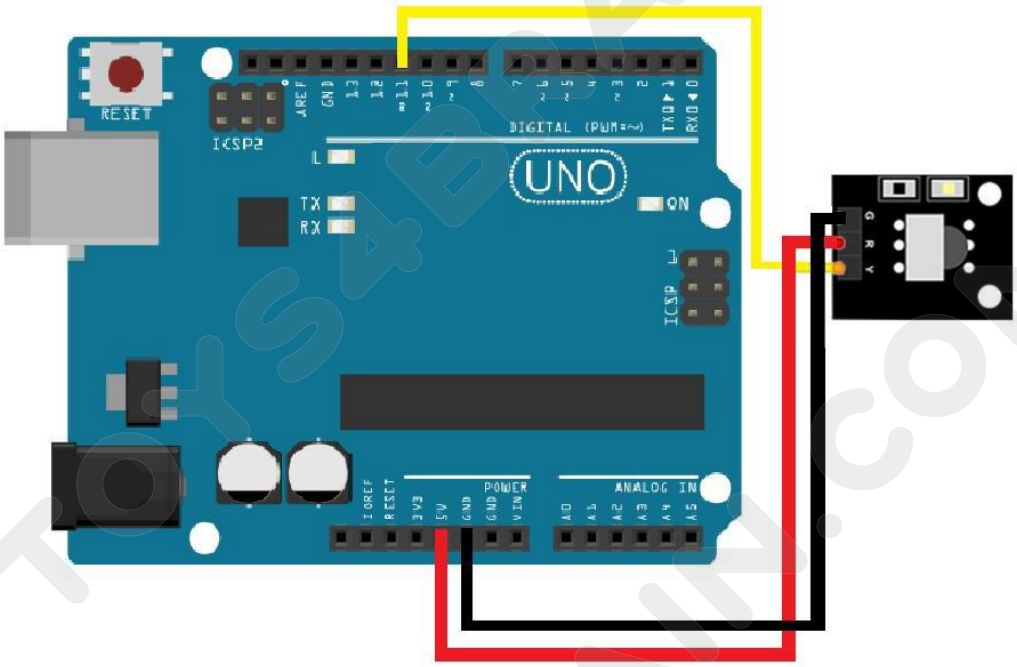
As you can see from these datasheet graphs, the peak frequency detection is at 38 KHz and the peak LED color is 940 nm. You can use from about 35 KHz to 41 KHz but the sensitivity will drop off so that it won't detect as well from afar. Likewise, you can use 850 to 1100 nm LEDs but they won't work as well as 900 to 1000nm so make sure to get matching LEDs! Check the datasheet for your IR LED to verify the wavelength.

Try to get a 940nm - remember that 940nm is not visible light!

Connection Diagram:



Wiring schematic :



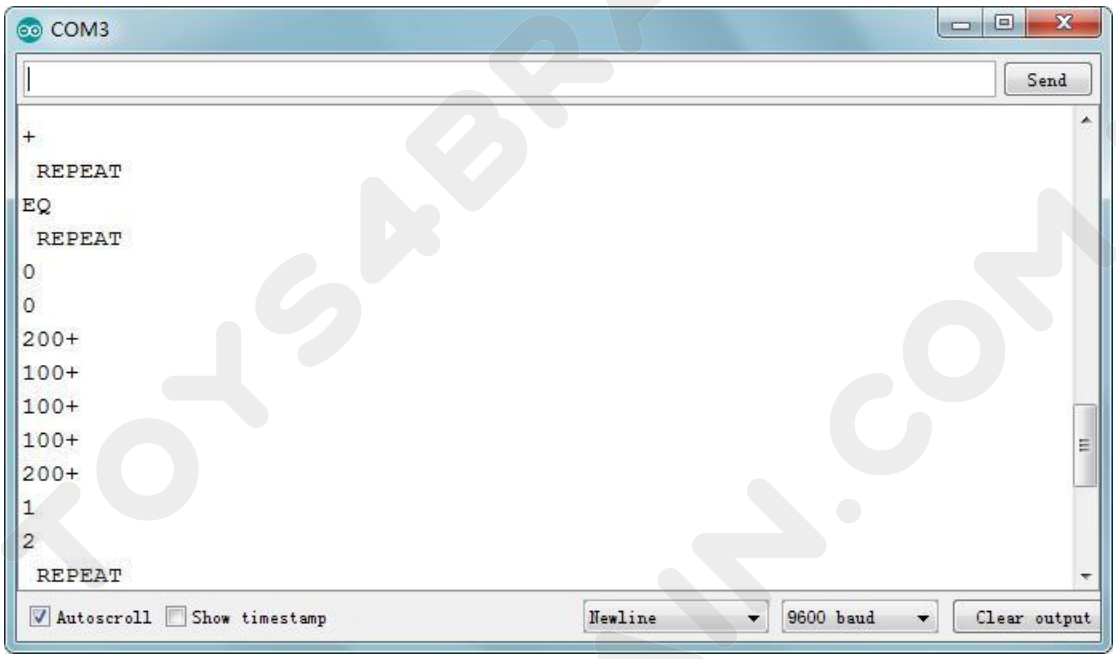
There are 3 connections to the IR Receiver.

The connections are: Signal, Voltage and Ground.

The “-” is the Ground, “S” is signal, and middle pin is Voltage 5V.

Physical wiring diagram:

Open the serial monitor as follows:



Code:

After wiring, open the program in the code folder - Lesson 13 , you need to install the relevant library file, then click Upload to upload the program.

```
#include "IRremote.h"
int receiver = 11; // Signal Pin of IR receiver to Arduino Digital Pin 11

/*-----( Declare objects )---- */
IRrecv irrecv(receiver);          // create instance of 'irrecv'
decode_results results;           // create instance of 'decode_results'

/*-----( Function )---- */
void translateIR()              // takes action based on IR code received

// describing Remote IR codes

{

  switch(results.value)

  {
  case 0xFFA25D: Serial.println("CH-"); break;
  case 0xFFE21D: Serial.println("CH"); break;
  case 0xFF629D: Serial.println("CH+"); break;
```

```
    case 0xFF22DD: Serial.println("VOL-");    break;
    case 0xFF02FD: Serial.println("VOL+");    break;
    case 0xFFC23D: Serial.println("FAST FORWARD"); break;
    case 0xFFE01F: Serial.println("-");      break;
    case 0xFFA857: Serial.println("+");      break;
    case 0xFF906F: Serial.println("EQ");     break;
    case 0xFF9867: Serial.println("100+"); break;
    case 0xFFB04F: Serial.println("200+"); break;
    case 0xFF6897: Serial.println("0");      break;
    case 0xFF30CF: Serial.println("1");      break;
    case 0xFF18E7: Serial.println("2");      break;
    case 0xFF7A85: Serial.println("3");      break;
    case 0xFF10EF: Serial.println("4");      break;
    case 0xFF38C7: Serial.println("5");      break;
    case 0xFF5AA5: Serial.println("6");      break;
    case 0xFF42BD: Serial.println("7");      break;
    case 0xFF4AB5: Serial.println("8");      break;
    case 0xFF52AD: Serial.println("9");      break;
    case 0xFFFFFFFF: Serial.println(" REPEAT");break;

    default:
      Serial.println(" other button      ");

    } // End Case

    delay(500); // Do not get immediate repeat

} //END translateIR
void setup()     /*----( SETUP: RUNS ONCE )--- */
{
    Serial.begin(9600);
    Serial.println("IR Receiver Button Decode");
    irrecv.enableIRIn(); // Start the receiver
}
/*--(end setup )---*/
void loop()     /*----( LOOP: RUNS CONSTANTLY )--- */
{
    if (irrecv.decode(&results)) // have we received an IR signal?
    {
      translateIR();
      irrecv.resume(); // receive the next value
    }
} /* --(end main loop )-- */
```

Lesson 14 LCD Display

Overview

In this lesson, you will learn how to wire up and use an alphanumeric LCD display.
The display has an LED backlight and can display two rows with up to 16 characters on each row. You can see the rectangles for each character on the display and the pixels that make up each character. The display is just white on blue and is intended for showing text.
In this lesson, we will run the Arduino example program for the LCD library, but in the next lesson, we will get our display to show the temperature, using sensors.

Component Required:

1 x Arduino Uno R3

1 x LCD1602 module

1 x Potentiometer (10k)

1 x 830 tie-points Breadboard

16 x M-M wires (Male to Male jumper wires)

Component Introduction

LCD1602

Introduction to the pins of LCD1602:

VSS: A pin that connects to ground

VDD: A pin that connects to a +5V power supply

VO: A pin that adjust the contrast of LCD1602

RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

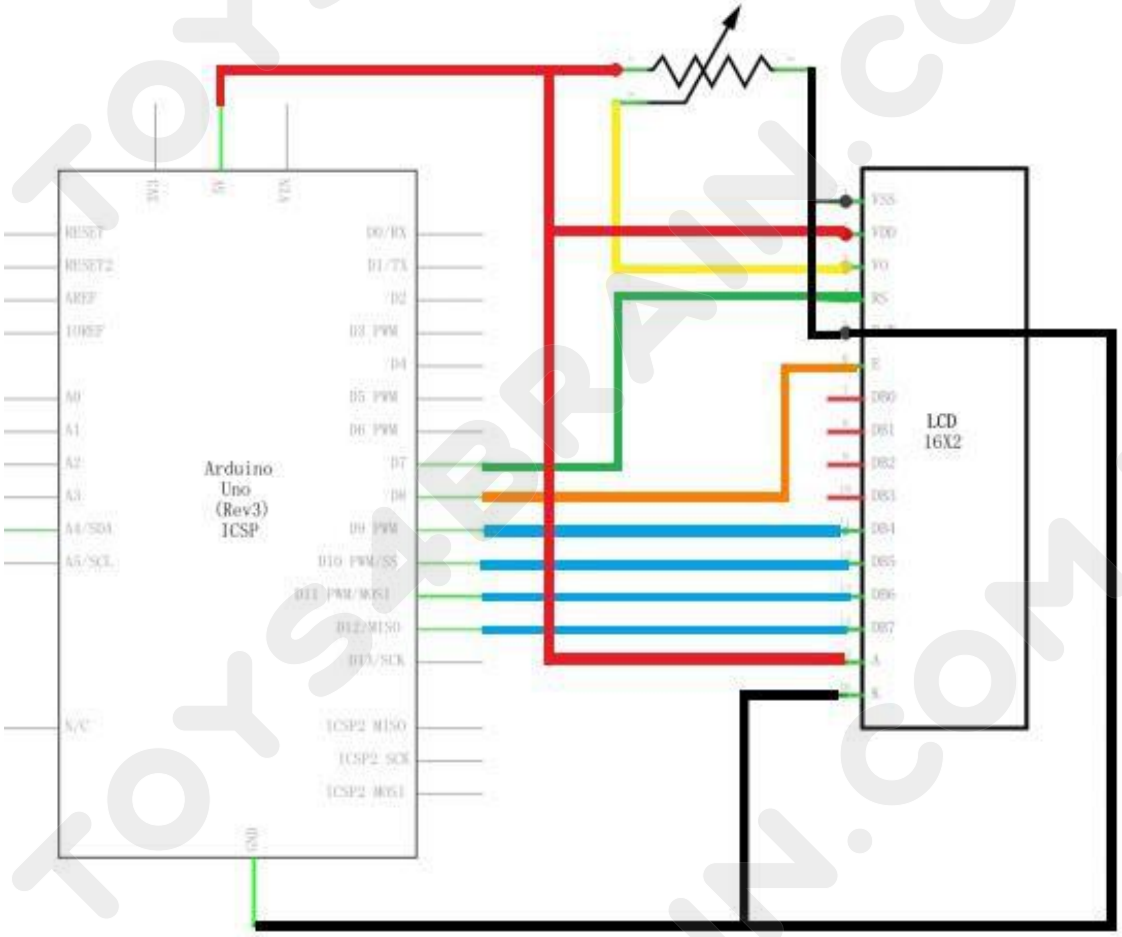
R/W: A Read/Write pin that selects reading mode or writing mode

E: An enabling pin that, when supplied with low-level energy, causes the LDC module to execute relevant instructions.

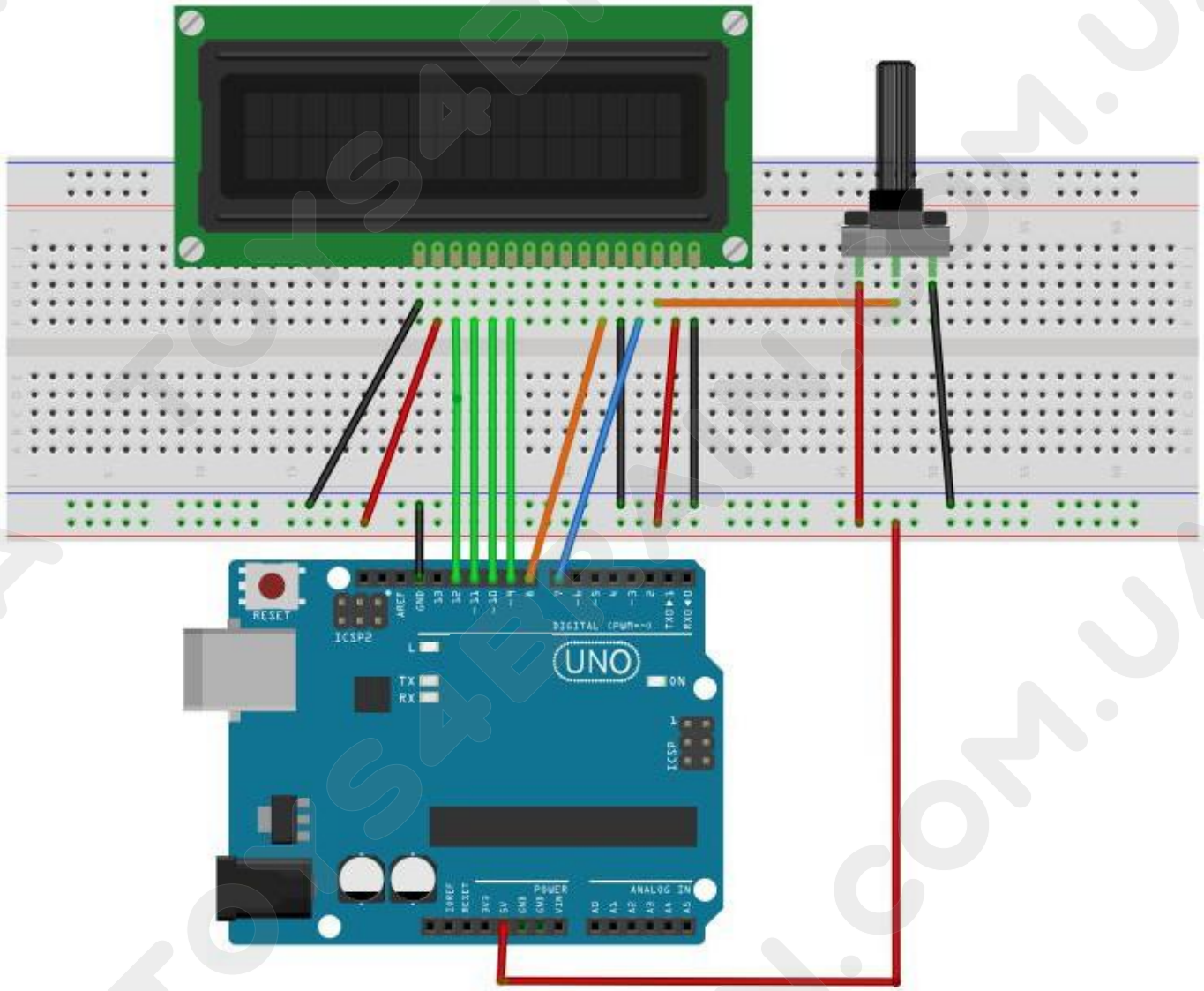
D0-D7： Pins that read and write data

A and K: Pins that control the LED backlight

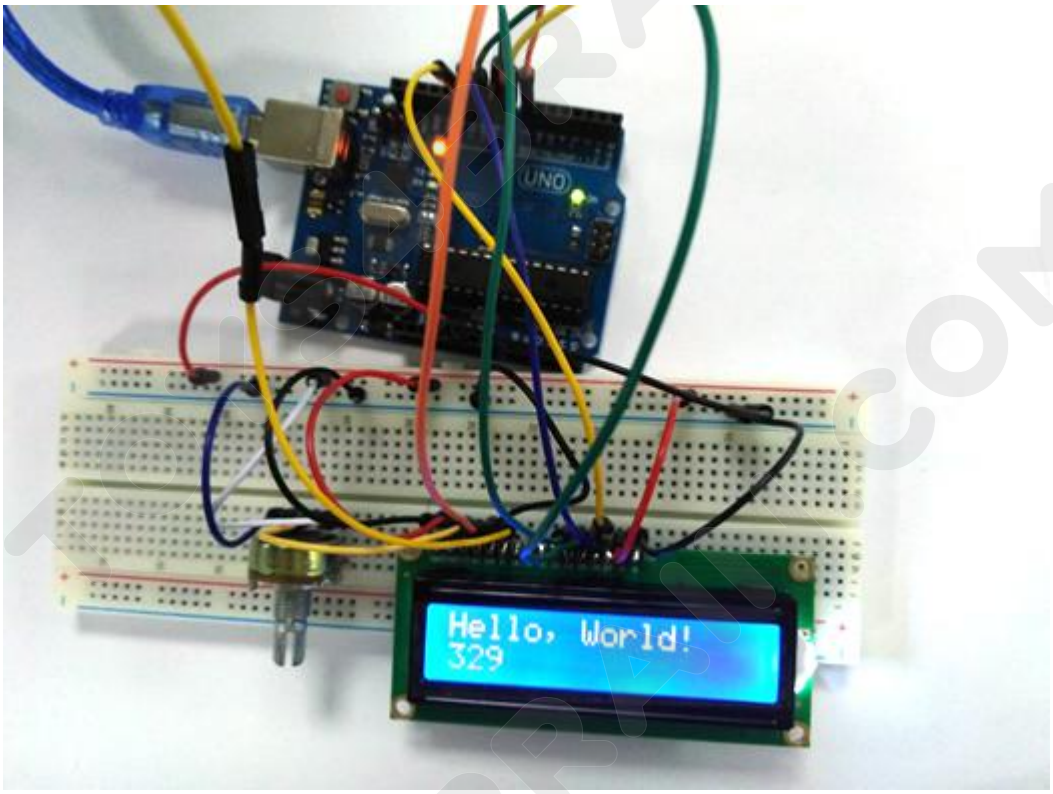
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



The LCD display needs six Arduino pins, all set to be digital outputs. It also needs 5V and GND connections.

There are a number of connections to be made. Lining up the display with the top of the breadboard helps to identify its pins without too much counting, especially if the breadboard has its rows numbered with row 1 as the top row of the board. Do not forget, the long yellow lead that links the slider of the pot to pin 3 of the display. The 'pot' is used to control the contrast of the display.

You may find that your display is supplied without header pins attached to it. If so, follow the instructions in the next section.

Code:

After wiring, open the program in the code folder - Lesson 14, you need to install the relevant library file, then click Upload to upload the program.

The first thing of note in the sketch is the line:

#include <LiquidCrystal.h>

This tells Arduino that we wish to use the Liquid Crystal library.

Next we have the line that we had to modify. This defines which pins of the Arduino are to be connected to which pins of the display.

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

After uploading this code, make sure the backlight is lit up, and adjust the potentiometer all the way around until you see the text message

In the 'setup' function, we have two commands:

lcd.begin(16, 2);

lcd.print("Hello, World!");

The first tells the Liquid Crystal library how many columns and rows the display has.

The second line displays the message that we see on the first line of the screen.

In the 'loop' function, we aso have two commands:

lcd.setCursor(0, 1);

lcd.print(millis()/1000);

The first sets the cursor position (where the next text will appear) to column 0 & row 1. Both column and row numbers start at 0 rather than 1.

The second line displays the number of milliseconds since the Arduino was reset.

Lesson 15 Thermometer Experiment

Overview

In this course, you will learn how to use a thermistor in conjunction with a serial display to display temperature.

Component Required:

1x Arduino Uno R3

1 x 220 ohm resistor

1 x Thermistor

1 x Potentiometer

1 x 830 tie-points Breadboard

3 x M-M wires (Male to Male jumper wires)

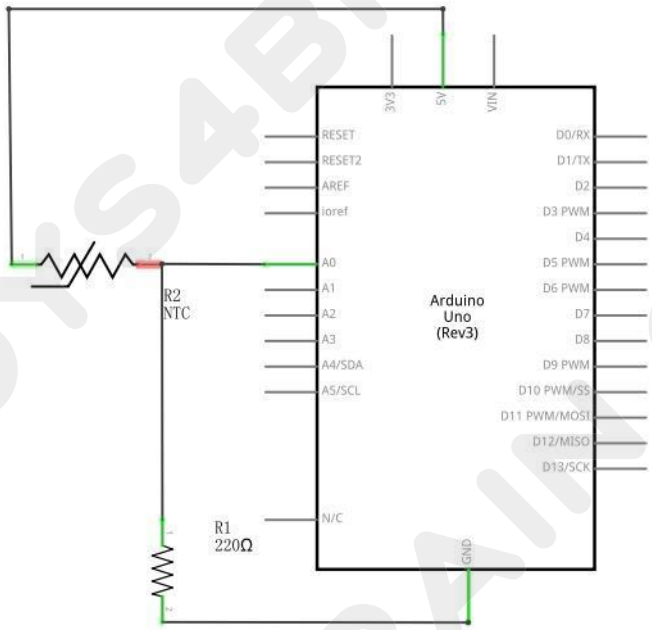
Component Introduction：

Thermistor：

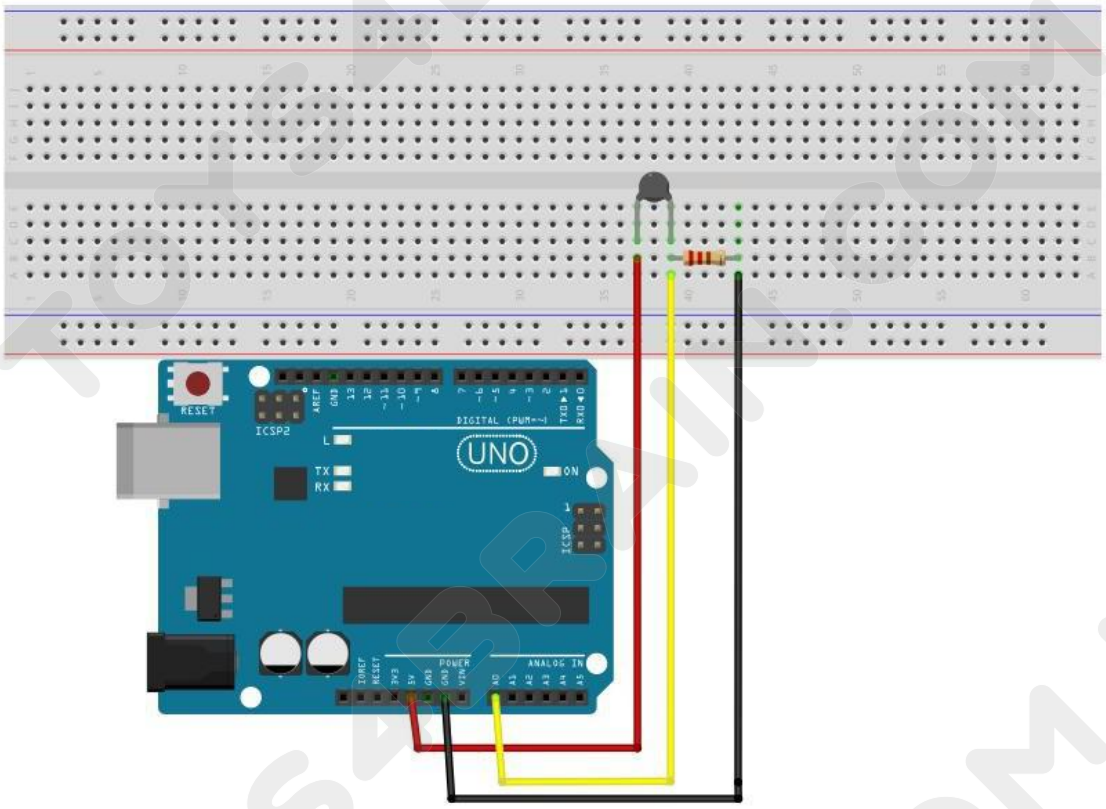
A thermistor is a thermal resistor - a resistor that changes its resistance with temperature. Technically, all resistors are thermistors - their resistance changes slightly with temperature - but the change is usually very small and difficult to measure. Thermistors are made so that the resistance changes drastically with temperature so that it can be 100 ohms or more of change per degree!

There are two kinds of thermistors, NTC (negative temperature coefficient) and PTC (positive temperature coefficient). In general, you will see NTC sensors used for temperature measurement. PTC's are often used as resettable fuses - an increase in temperature increases the resistance which means that as more current passes thru them, they heat up and 'choke back' the current, quite handy for protecting circuits!

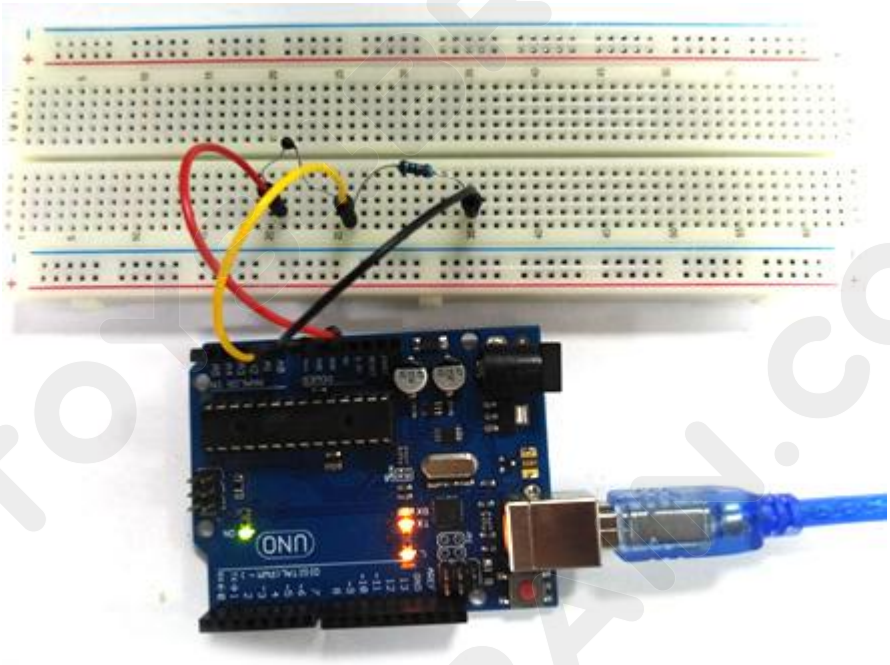
Connection Diagram:



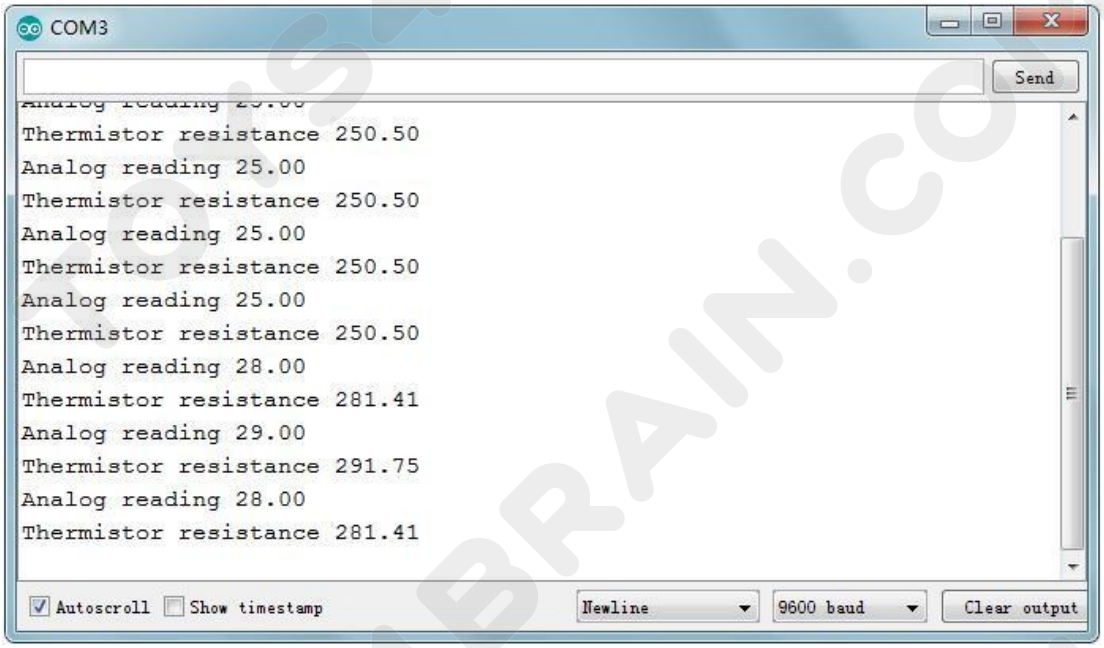
Wiring schematic :



Physical wiring diagram:



Open the serial monitor as follows:



Code:

```
// the value of the 'other' resistor
#define SERIESRESISTOR 10000
// What pin to connect the sensor to
#define THERMISTORPIN A0

void setup(void)
   { Serial.begin(9600)
   ;
}
void loop(void)
   { float reading;
   reading = analogRead(THERMISTORPIN);

   Serial.print("Analog reading ");
   Serial.println(reading);

   // convert the value to resistance
   reading = (1023 / reading)    - 1;         // (1023/ADC - 1)
   reading = SERIESRESISTOR / reading;    // 10K / (1023/ADC - 1)
   Serial.print("Thermistor resistance ");
   Serial.println(reading);

   delay(1000);
}
```

Lesson16 Analog Joystick Module

Overview:

Analog joysticks are a great way to add some control in your projects.

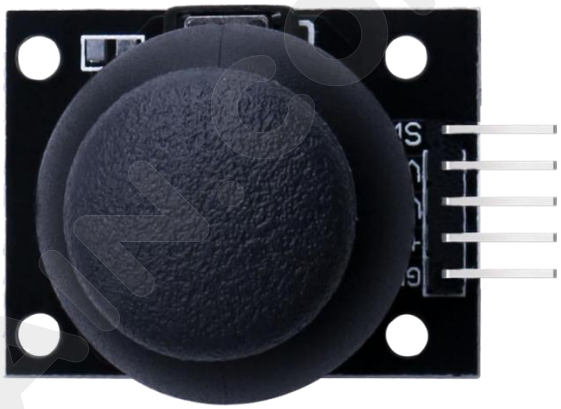
In this tutorial we will learn how to use the analog joystick module.

Component Required:

1 x Arduino UNO R3

1 x Joystick module

5 x F-M wires (Female to Male DuPont wires)



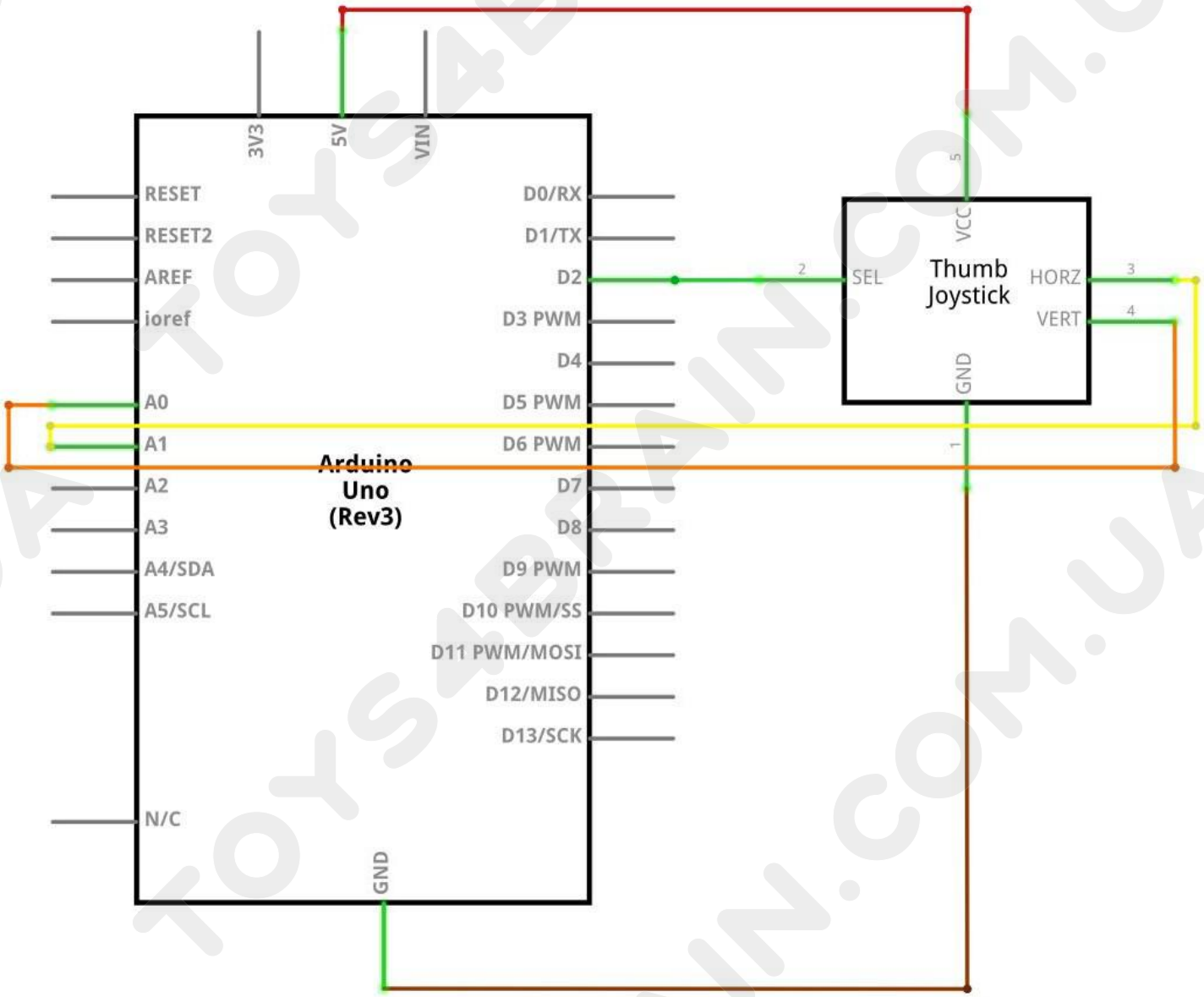
Component Introduction

Joystick

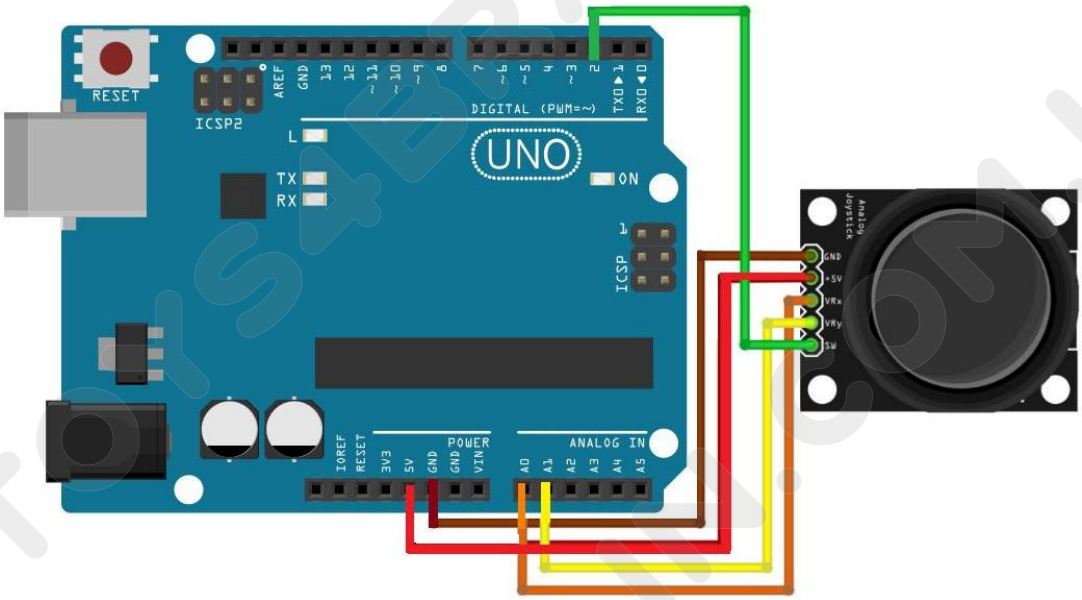
The module has 5 pins: VCC, Ground, X, Y, Key. Note that the labels on yours may be slightly different, depending on where you got the module from. The thumb stick is analog and should provide more accurate readings than simple 'directional' joysticks tact use some forms of buttons, or mechanical switches. Additionally, you can press the joystick down (rather hard on mine) to activate a 'press to select' push-button.

We have to use analog Arduino pins to read the data from the X/Y pins, and a digital pin to read the button. The Key pin is connected to ground, when the joystick is pressed down, and is floating otherwise. To get stable readings from the Key /Select pin, it needs to be connected to VCC via a pull-up resistor. The built in resistors on the Arduino digital pins can be used. For a tutorial on how to activate the pull-up resistors for Arduino pins, configured as inputs

Connection Diagram:



Wiring schematic :



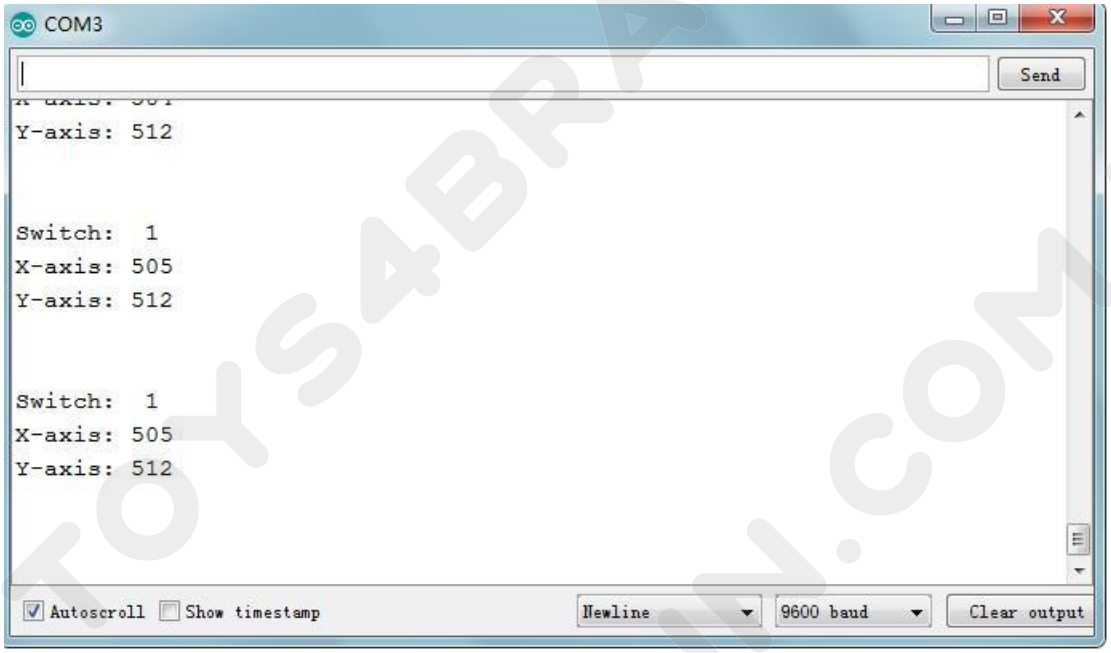
We need 5 connections to the joystick.

The connections are: Key, Y, X, Voltage and Ground.

“Y and X” are Analog and “Key” is Digital. If you don’t need the switch then you can use only 4 pins.

Physical wiring diagram:

Open the serial monitor as follows:



Code:

```
/* Arduino pin numbers*/
const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output
void setup()
{
   pinMode(SW_pin, INPUT);
   digitalWrite(SW_pin, HIGH);
   Serial.begin(9600);
}
void loop()
   { Serial.print("Switch:  ");
   Serial.print(digitalRead(SW_pin));
   Serial.print("\n");
   Serial.print("X-axis: ");
   Serial.print(analogRead(X_pin));
   Serial.print("\n");
   Serial.print("Y-axis: ");
   Serial.println(analogRead(Y_pin));
   Serial.print("\n\n");
   delay(500);
}
```

Lesson 17 Eight LED with 74HC595

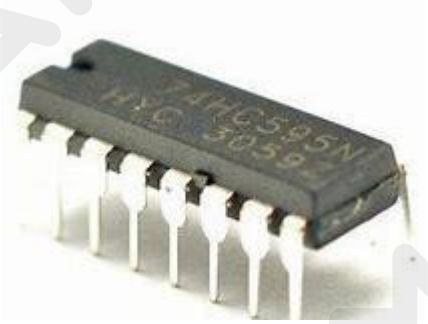
Overview:

In this lesson, you will learn how to use eight LEDs with an UNO without needing to give up 8 output pins!

Although you could wire up eight LEDs each with a resistor to an UNO pin you would rapidly start to run out of pins on your UNO. If you don't have a lot of stuff connected to your UNO. It's OK to do so - but often times we want buttons, sensors, servos, etc. and before you know it you've got no pins left. So, instead of doing that, you are going to use a chip called the 74HC595 Serial to Parallel Converter. This chip has eight outputs (perfect) and three inputs that you use to feed data into it a bit at a time.

This chip makes it a little slower to drive the LEDs (you can only change the LEDs about 500,000 times a second instead of 8,000,000 a second) but it's still really fast, way faster than humans can detect, so it's worth it!

Component Required:



1 x Arduino Uno R3

1 x 830 tie-points breadboard

8 x leds

8 x 220 ohm resistors

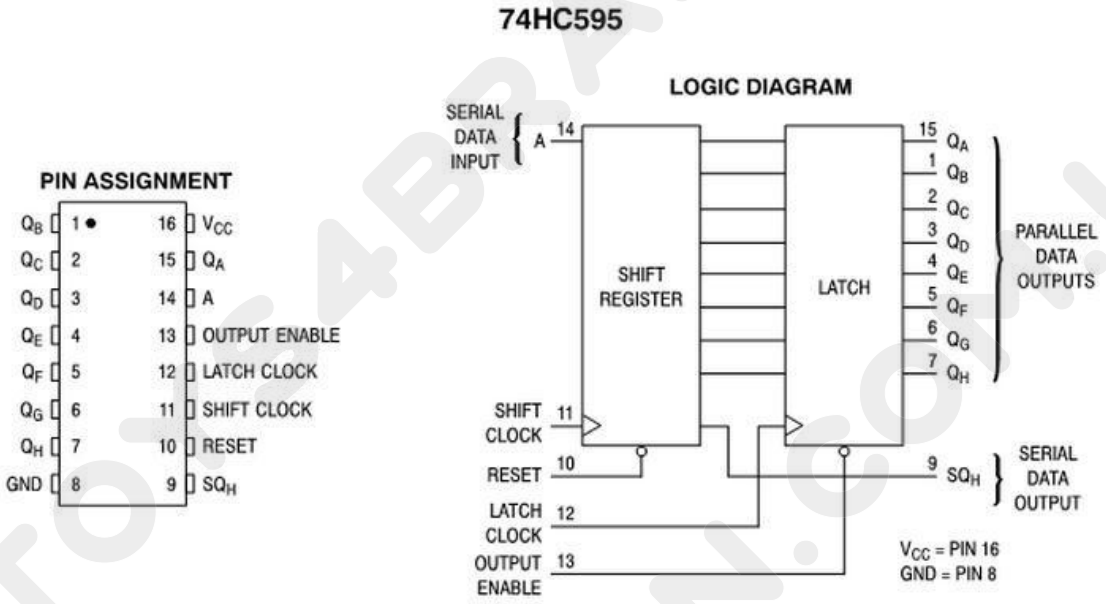
1 x 74hc595 IC

14 x M-M wires (Male to Male jumper wires)

Component Introduction

74HC595 Shift Register:

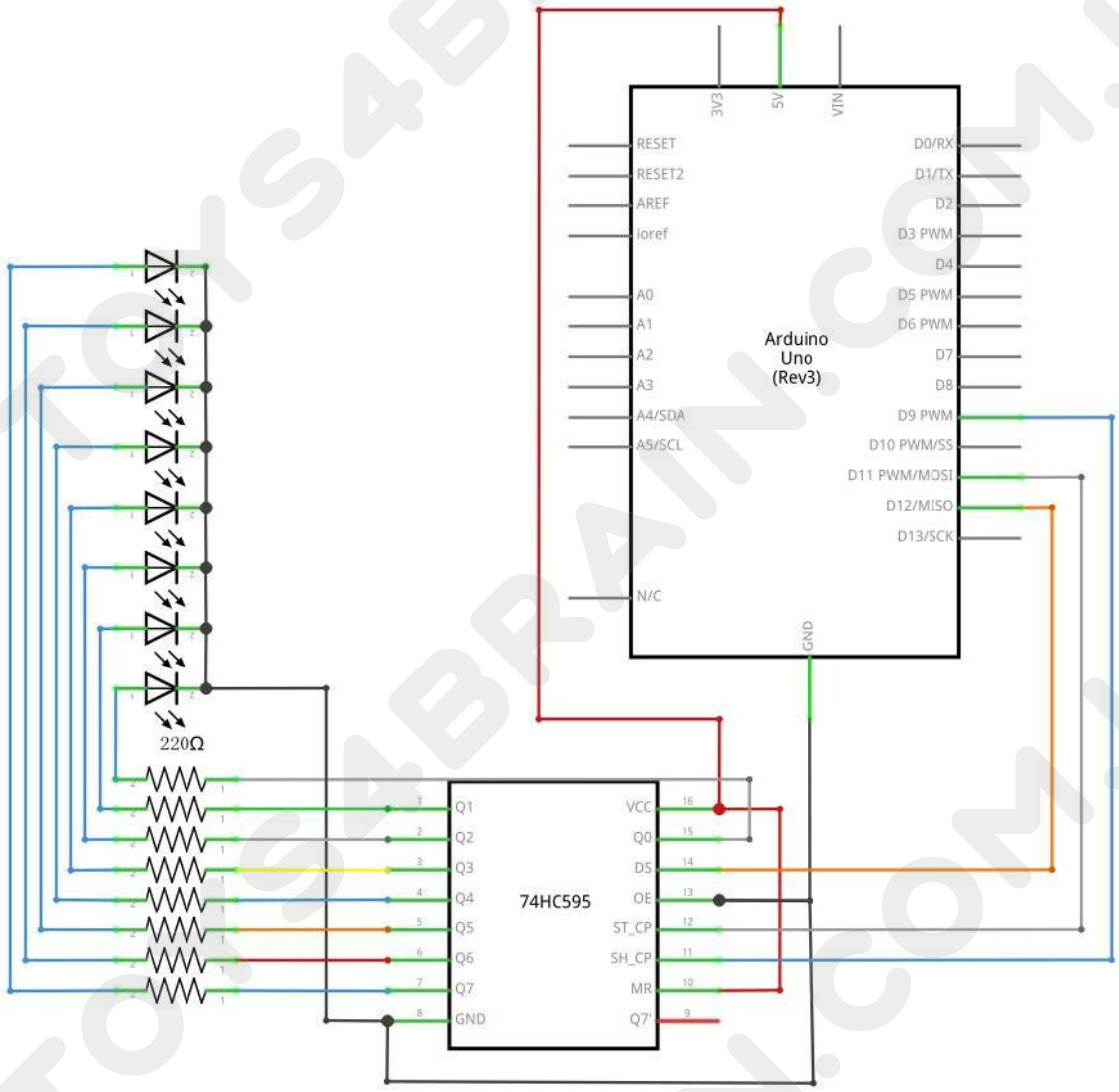
The shift register is a type of chip that holds what can be thought of as eight memory locations, each of which can either be a 1 or a 0. To set each of these values on or off, we feed in the data using the 'Data' and 'Clock' pins of the chip.



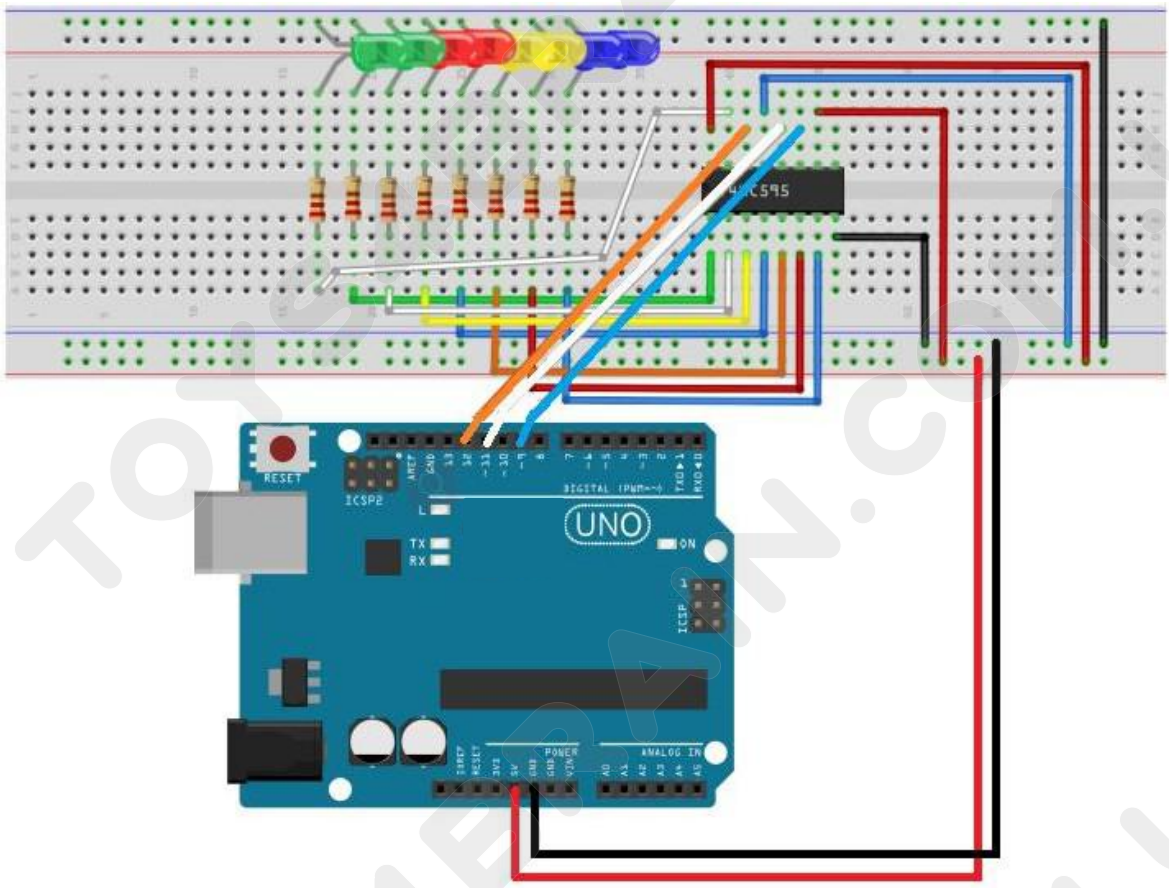
The clock pin needs to receive eight pulses. At each pulse, if the data pin is high, then a 1 gets pushed into the shift register; otherwise, a 0. When all eight pulses have been received, enabling the 'Latch' pin copies those eight values to the latch register. This is necessary; otherwise, the wrong LEDs would flicker as the data is being loaded into the shift register.

The chip also has an output enable (OE) pin, which is used to enable or disable the outputs all at once. You could attach this to a PWM-capable UNO pin and use 'analogWrite' to control the brightness of the LEDs. This pin is active low, so we tie it to GND.

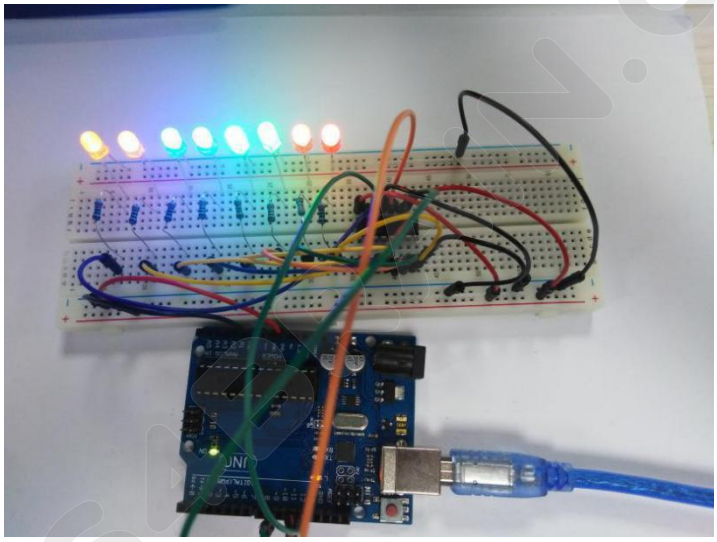
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



As we have eight LEDs and eight resistors to connect, there are actually quite a few connections to be made.

It is probably easiest to put the 74HC595 chip in first, as pretty much everything else connects to it. Put it so that the little U-shaped notch is towards the top of the breadboard. Pin 1 of the chip is to the left of this notch.

Digital 12 from the UNO goes to pin #14 of the shift register

Digital 11 from the UNO goes to pin #12 of the shift register

Digital 9 from the UNO goes to pin #11 of the shift register

All but one of the outputs from the IC is on the left side of the chip. Hence, for ease of connection, that is where the LEDs are, too.

After the chip, put the resistors in place. You need to be careful that none of the leads of the resistors are touching each other. You should check this again before you connect the power to your UNO. If you find it difficult to arrange the resistors without their leads touching, then it helps to shorten the leads so that they are lying closer to the surface of the breadboard.

Next, place the LEDs on the breadboard. The longer positive LED leads must all be towards the chip, whichever side of the breadboard they are on.

Attach the jumper leads as shown above. Do not forget the one that goes from pin 8 of the IC to the GND column of the breadboard.

Load up the sketch listed a bit later and try it out. Each LED should light in turn until all the LEDs are on, and then they all go off and the cycle repeats.

Code:

The first thing we do is define the three pins we are going to use. These are the UNO digital outputs that will be connected to the latch, clock and data pins of the 74HC595.

```
int latchPin = 11;
int clockPin = 9;
int dataPin = 12;
```

Next, a variable called 'leds' is defined. This will be used to hold the pattern of which LEDs are currently turned on or off. Data of type 'byte' represents numbers using eight bits. Each bit can be either on or off, so this is perfect for keeping track of which of our eight LEDs are on or off.

```
byte leds = 0;
```

The 'setup' function just sets the three pins we are using to be digital outputs.

```
void setup()
{
  pinMode(latchPin,  OUTPUT);
  pinMode(dataPin,  OUTPUT);
  pinMode(clockPin, OUTPUT);
}
```

The 'loop' function initially turns all the LEDs off, by giving the variable 'leds' the value 0. It then calls 'updateShiftRegister' that will send the 'leds' pattern to the shift register so that all the LEDs turn off. We will deal with how 'updateShiftRegister' works later.

The loop function pauses for half a second and then begins to count from 0 to 7 using the 'for' loop and the variable 'i'. Each time, it uses the Arduino function 'bitSet' to set the bit that controls that LED in the variable 'leds'. It then also calls 'updateShiftRegister' so that the leds update to reflect what is in the variable 'leds'. There is then a half second delay before 'i' is incremented and the next LED is lit.

```
void loop()
{
  leds = 0;
  updateShiftRegister();
  delay(500);
  for (int i = 0; i < 8; i++)
  {
    bitSet(leds, i);
    updateShiftRegister();
    delay(500);
  }
}
```

The function 'updateShiftRegister', first of all sets the latchPin to low, then calls the UNO function 'shiftOut' before putting the 'latchPin' high again. This takes four parameters, the first two are the pins to use for Data and Clock respectively.

The third parameter specifies which end of the data you want to start at. We are

going to start with the right most bit, which is referred to as the 'Least Significant Bit' (LSB).

The last parameter is the actual data to be shifted into the shift register, which in this case is 'leds'.

```
void updateShiftRegister()
{
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, leds);
    digitalWrite(latchPin, HIGH);
}
```

If you wanted to turn one of the LEDs off rather than on, you would call a similar Arduino function (bitClear) with the 'leds' variable. This will set that bit of 'leds' to be 0 and you would then just need to follow it with a call to 'updateShiftRegister' to update the actual LEDs.

Lesson 18 Photocell Experiment

Overview:

In this lesson, you will learn how to measure light intensity using an Analog Input. You will build on lesson 17 and use the level of light to control the number of LEDs to be lit.

The photocell is at the bottom of the breadboard, where the pot was above.

Component Required:

1 x Arduino Uno R3

1 x 830 tie-points breadboard

8 x leds

8 x 220 ohm resistors

1 x 1k ohm resistor

1 x 74hc595 IC

1 x Photoresistor (Photocell)

16 x M-M wires (Male to Male jumper wires)

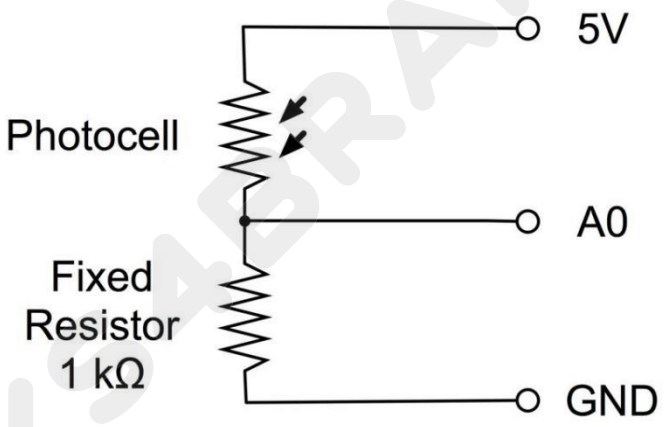
Component Introduction

PHOTOCELL:

The photocell used is of a type called a light dependent resistor, sometimes called an LDR. As the name suggests, these components act just like a resistor, except that the resistance changes in response to how much light is falling on them.

This one has a resistance of about 50 kΩ in near darkness and 500 Ω in bright light. To convert this varying value of resistance into something we can measure on an UNO R3 board's analog input, it needs to be converted into a voltage.

The simplest way to do that is to combine it with a fixed resistor.

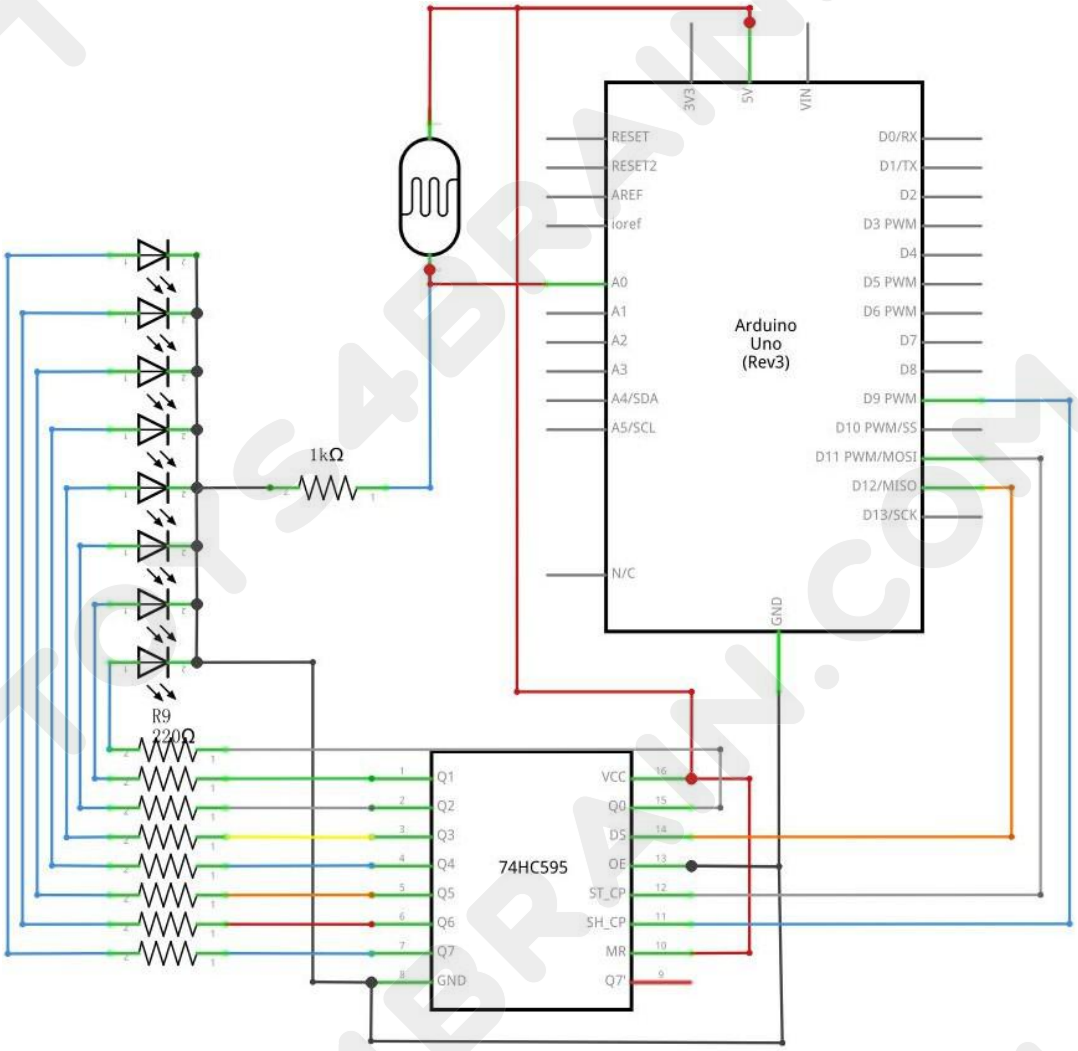


The resistor and photocell together behave like a pot. When the light is very bright, then the resistance of the photocell is very low compared with the fixed value resistor, and so it is as if the pot were turned to maximum.

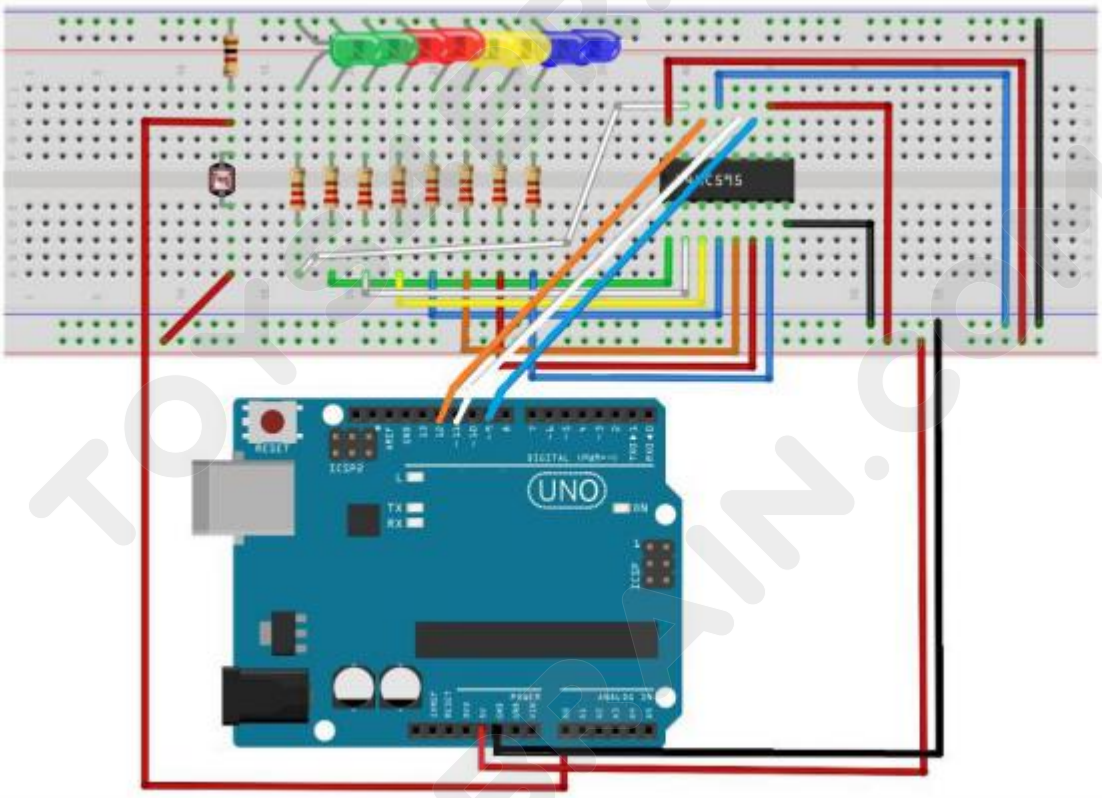
When the photocell is in dull light, the resistance becomes greater than the fixed 1 kΩ resistor and it is as if the pot were being turned towards GND.

Load up the sketch given in the next section and try covering the photocell with your finger, and then holding it near a light source.

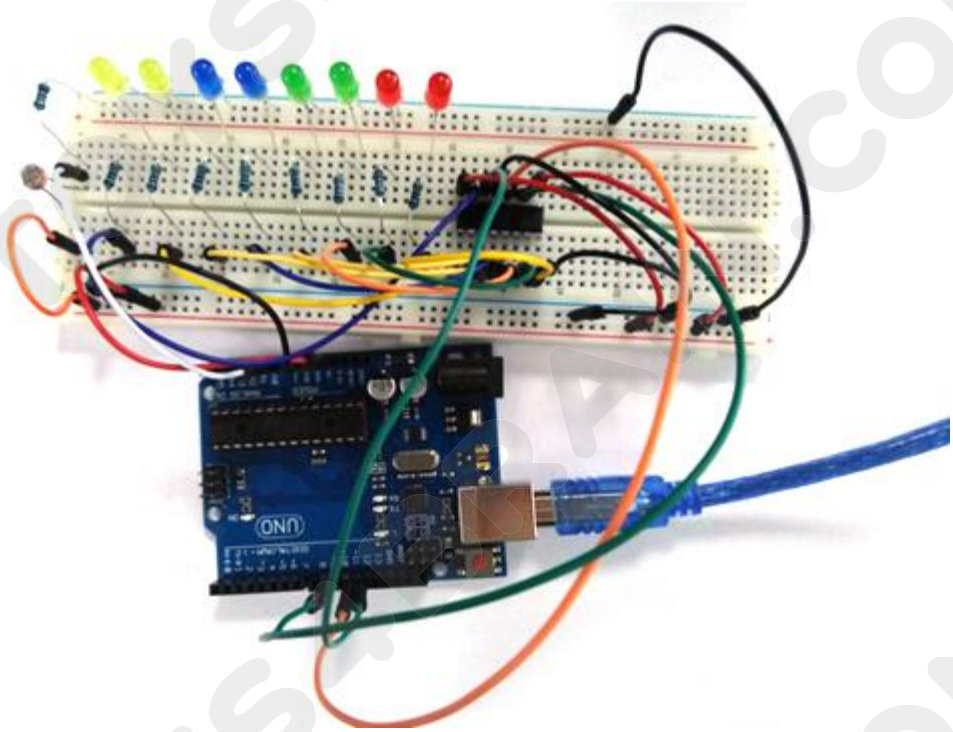
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
int lightPin = 0;
int latchPin = 11;
int clockPin = 9;
int dataPin = 12;

int leds = 0;

void setup()
{
   pinMode(latchPin,  OUTPUT);
   pinMode(dataPin,   OUTPUT);
   pinMode(clockPin, OUTPUT);
}
void updateShiftRegister()
{
     digitalWrite(latchPin, LOW);
     shiftOut(dataPin, clockPin, LSBFIRST, leds);
     digitalWrite(latchPin, HIGH);
}
void loop()
{
   int reading    = analogRead(lightPin);
   int numLEDSLit = reading / 57;    //1023 / 9 / 2
   if (numLEDSLit > 8) numLEDSLit = 8;
   leds = 0;      // no LEDs lit to start
   for (int i = 0; i < numLEDSLit; i++)
   {
      leds = leds + (1 << i);    // sets the i'th bit
   }
   updateShiftRegister();
}
```

Lesson 19 Controlling Stepper Motor With Remote

Overview

In this lesson, you will learn a fun and easy way to control a stepper motor from a distance using an IR remote control.

The stepper we are using comes with its own driver board making it easy to connect to our UNO.

Since we don't want to drive the motor directly from the UNO, we will be using an inexpensive little breadboard power supply that plugs right into our breadboard and power it with a 9V 1Amp power supply.

The IR sensor is connected to the UNO directly since it uses almost no power.

Component Required:

1 x Arduino Uno R3

1 x IR receiver module

1 x IR remote

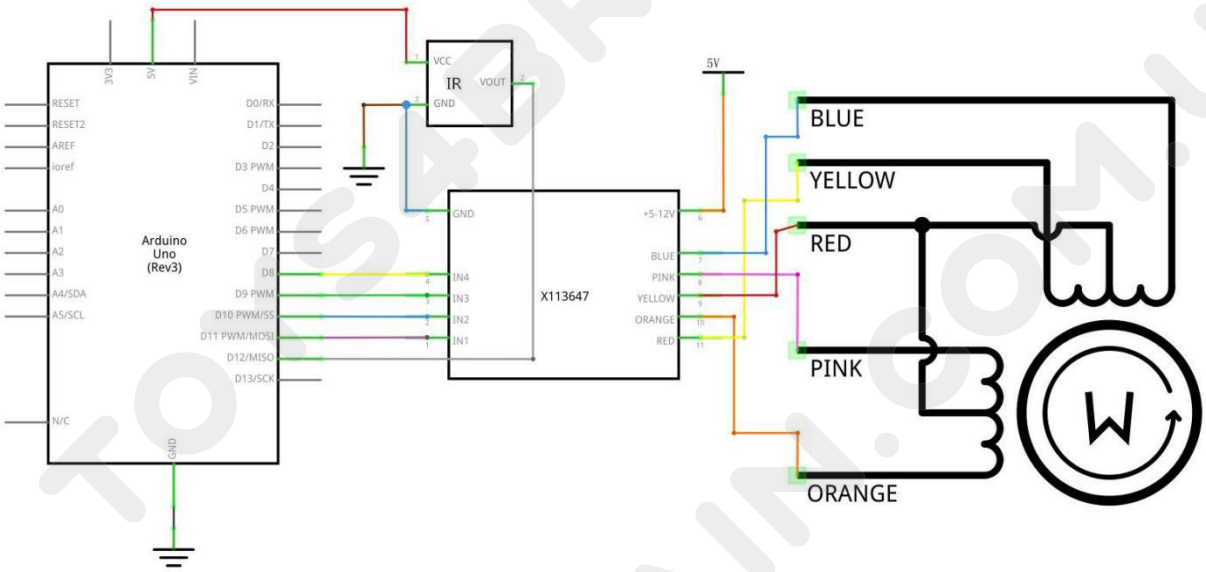
1 x ULN2003 stepper motor driver module

1 x Stepper motor

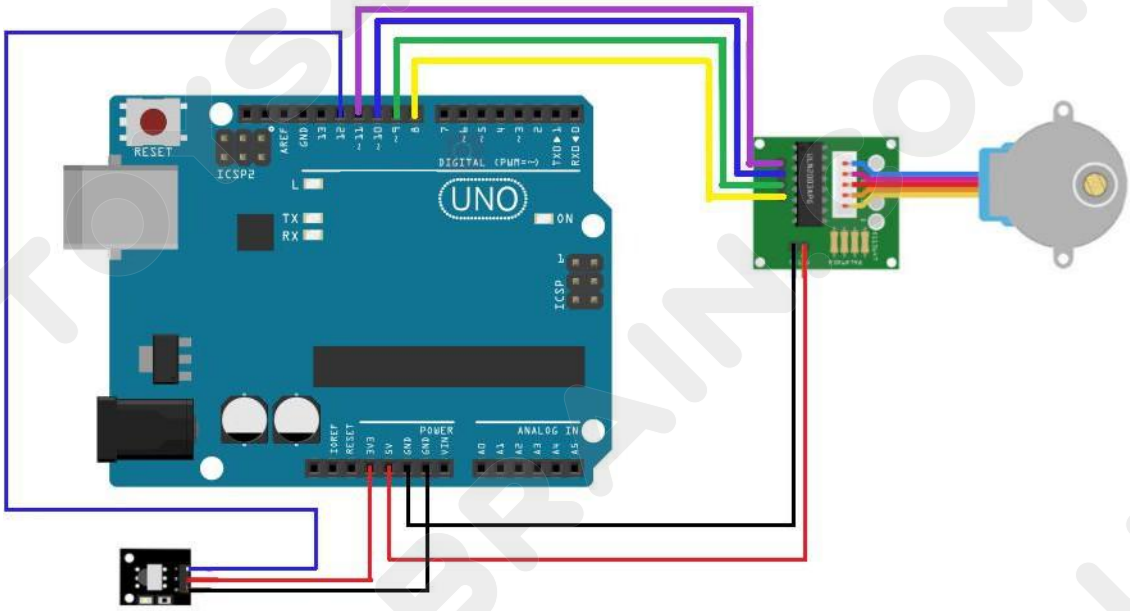
7 x F-M wires (Female to Male DuPont wires)

2 x M-M wire (Male to Male jumper wire)

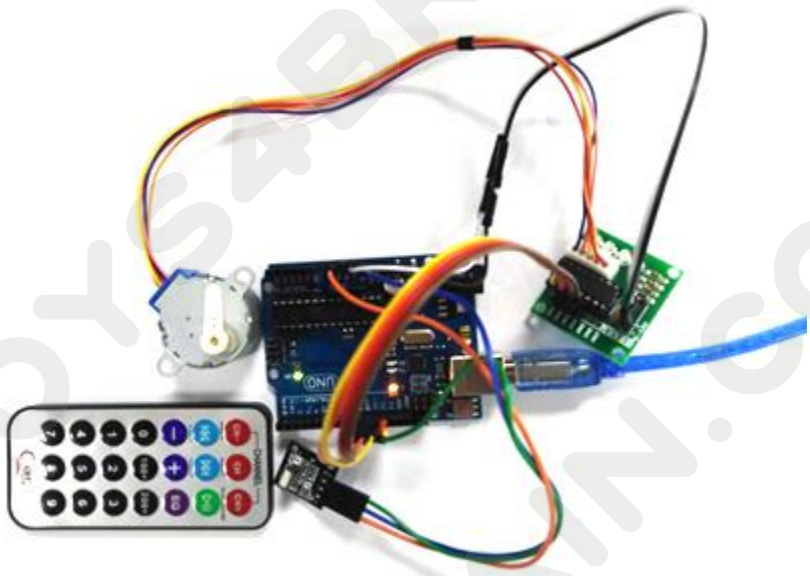
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
#include "Stepper.h"
#include "IRremote.h"

/*----- Variables, Pins----- */
#define STEPS  32   // Number of steps per revolution of Internal shaft
int  Steps2Take;  // 2048 = 1 Revolution
int receiver = 12; // Signal Pin of IR receiver to Arduino Digital Pin 6

/*-----( Declare objects )---- */
// Setup of proper sequencing for Motor Driver Pins
// In1, In2, In3, In4 in the sequence 1-3-2-4

Stepper small_stepper(STEPS, 8, 10, 9, 11);
IRrecv irrecv(receiver);     // create instance of 'irrecv'
decode_results results;      // create instance of 'decode_results'

void setup()
{
  irrecv.enableIRIn();  // Start the receiver
}

void loop()
```

```
{
if (irrecv.decode(&results))     // have we received an IR signal?

  {
    switch(results.value)

    {

      case 0xFFE01F:                                    // "—"button pressed
                        small_stepper.setSpeed(500); //Max seems to be 500
                        Steps2Take  =  2048;       // Rotate CW
                        small_stepper.step(Steps2Take);
                        delay(2000);
                        break;

      case 0xFFA857:       // "+" button pressed
                        small_stepper.setSpeed(500);
                        Steps2Take  =  -2048;     // Rotate CCW
                        small_stepper.step(Steps2Take);
                        delay(2000);
                        break;

    }

      irrecv.resume();     // receive the next value
                  digitalWrite(8, LOW);
                  digitalWrite(9, LOW);
                  digitalWrite(10, LOW);
                  digitalWrite(11, LOW);
  }

}     /* --end main loop -- */
```

Lesson 20 DC Motors

Overview

In this lesson, you will learn how to control a small DC motor using an UNO R3 and a transistor.

Component Required:

1 x Arduino Uno R3

1 x 830 tie-points breadboard

1 x L293D IC

1 x Fan blade and 3-6v motor

5 x M-M wires (Male to Male jumper wires)

Component Introduction

L293D

This is a very useful chip. It can actually control two motors independently. We are just using half the chip in this lesson, most of the pins on the right hand side of the chip are for controlling a second motor.

Product Specifications:

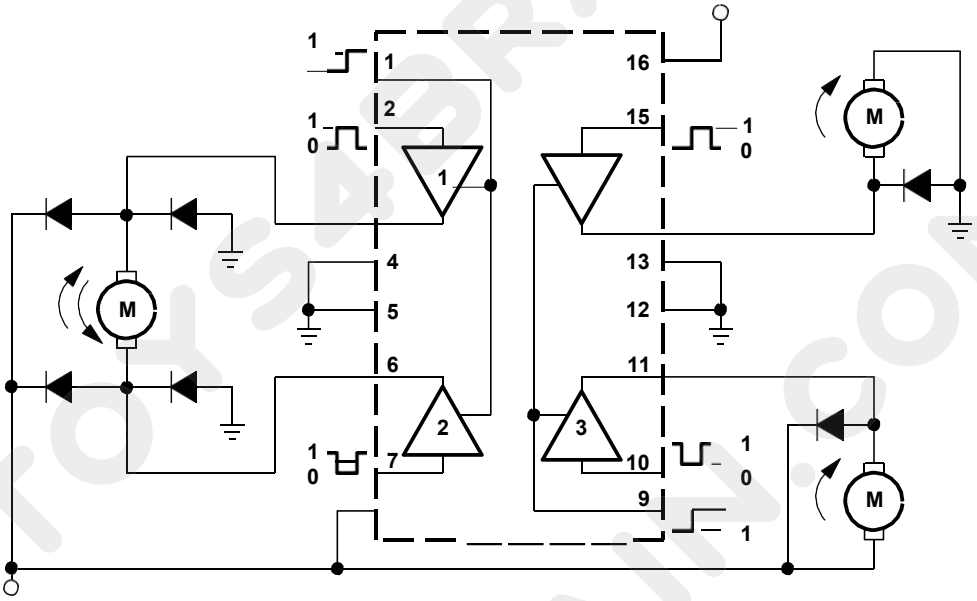
- Featuring Unitrode L293 and L293D Products Now From Texas Instruments
- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functionally Similar to SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive T ransient Suppression (L293D)

Description/ordering information

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

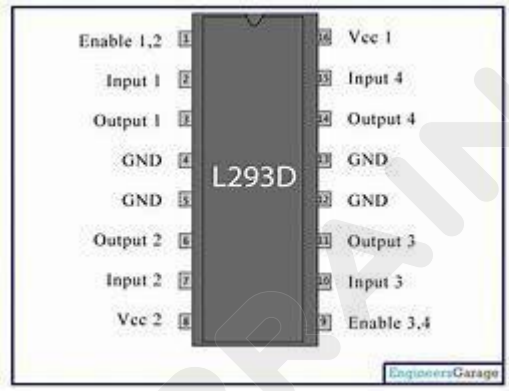
All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the

high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.



I got fed up with indecipherable pinout diagrams within datasheets, so have designed my own that I think gives more pertinent information.

There are 3 wires connected to the Arduino, 2 wires connected to the motor, and 1 wire connected to a battery



To use this pinout:

The left hand side deals with the first motor, the right hand side deals with a second motor.

Yes, you can run it with only one motor connected.

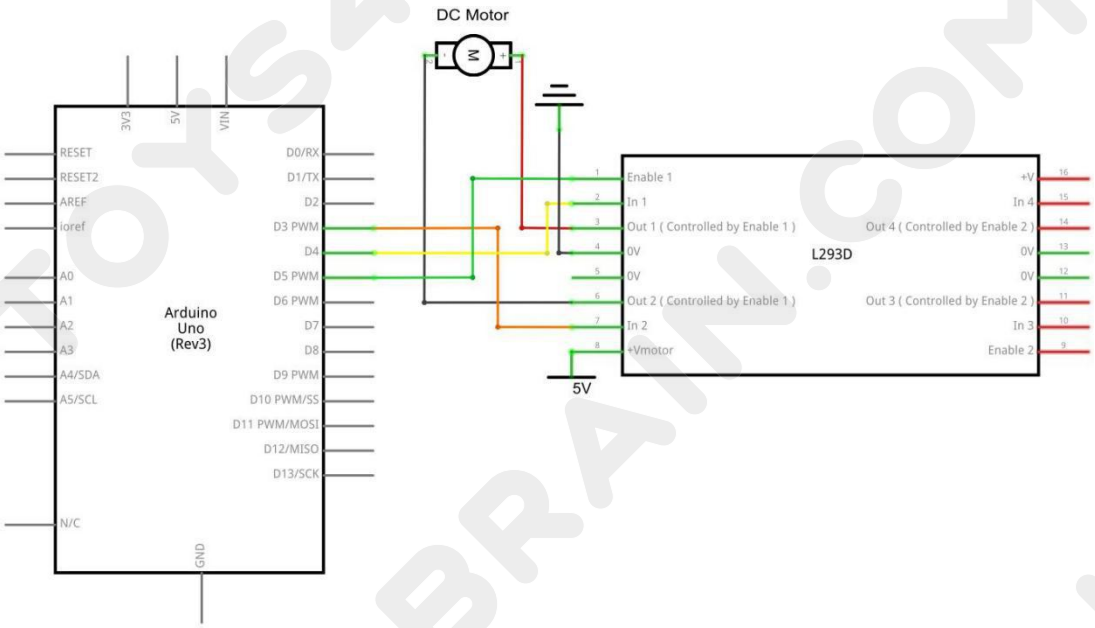
Arduino Connections

M1 PWM - connect this to a PWM pin on the Arduino. They're labelled on the Uno, pin 5 is an example. Output any integer between 0 and 255, where 0 will be off, 128 is half speed and 255 is max speed.

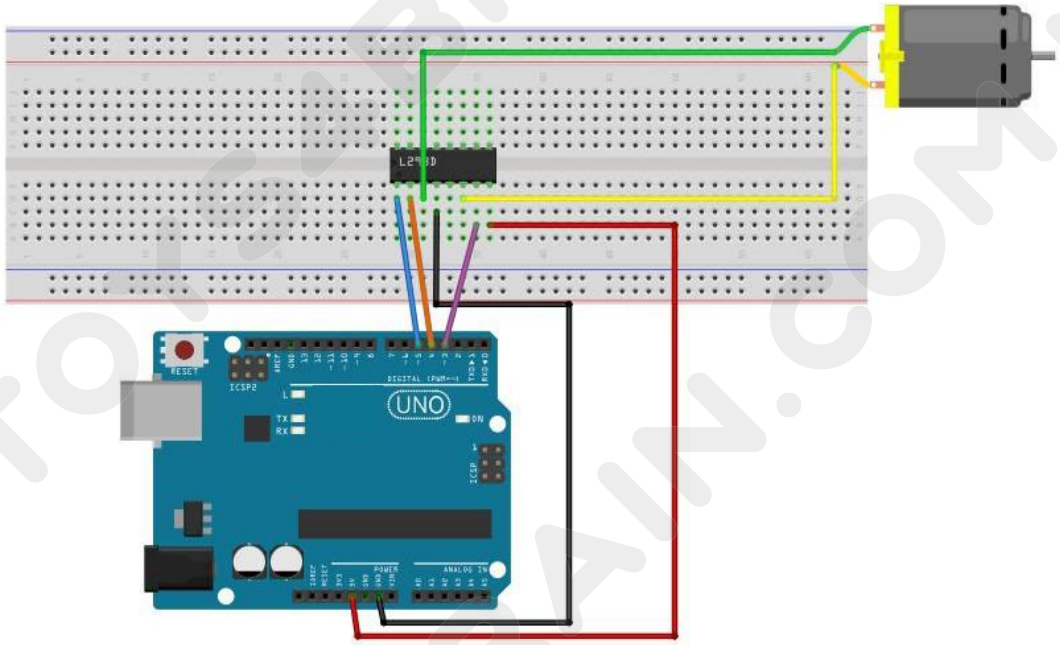
M1 direction 0/1 and M1 direction 1/0 - Connect these two to two digital Arduino pins. Output one pin as HIGH and the other pin as LOW, and the motor will spin in one direction.

Reverse the outputs to LOW and HIGH, and the motor will spin in the other direction.

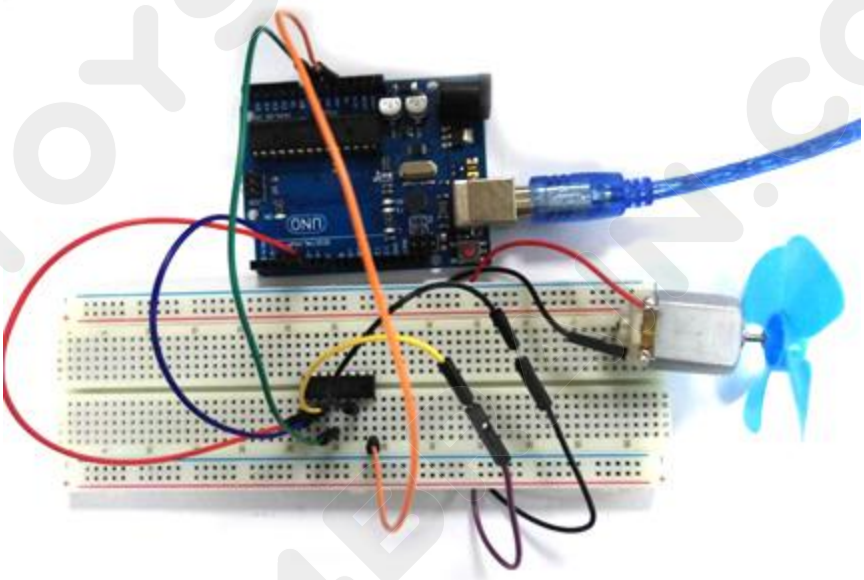
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
/************************
Exercise the motor using
the L293D chip
************************/

#define ENABLE 5
#define DIRA 3
#define DIRB 4

int i;

void setup() {
  //---set pin direction
  pinMode(ENABLE,OUTPUT);
  pinMode(DIRA,OUTPUT);
  pinMode(DIRB,OUTPUT);
  Serial.begin(9600);
}

void loop() {
  //---back and forth example
     Serial.println("One way, then reverse");
     digitalWrite(ENABLE,HIGH); // enable on
     for (i=0;i<5;i++)
     { digitalWrite(DIRA,HIGH); //one way
     digitalWrite(DIRB,LOW);
     delay(500);
     digitalWrite(DIRA,LOW);    //reverse
     digitalWrite(DIRB,HIGH); delay(500);
  }
  digitalWrite(ENABLE,LOW); // disable
  delay(2000);

  Serial.println("fast Slow example");
  //---fast/slow stop example
  digitalWrite(ENABLE,HIGH); //enable on
  digitalWrite(DIRA,HIGH); //one way
  digitalWrite(DIRB,LOW);
  delay(3000);
  digitalWrite(ENABLE,LOW); //slow stop
  delay(1000);
```

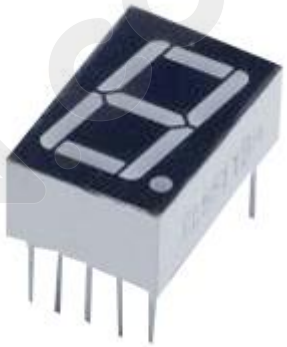
```
  digitalWrite(ENABLE,HIGH); //enable on
  digitalWrite(DIRA,LOW); //one way
  digitalWrite(DIRB,HIGH);
  delay(3000);
  digitalWrite(DIRA,LOW); //fast stop
  delay(2000);

  Serial.println("PWM full then slow");
  //---PWM example, full speed then slow
  analogWrite(ENABLE,255); //enable on
  digitalWrite(DIRA,HIGH); //one way
  digitalWrite(DIRB,LOW);
  delay(2000); analogWrite(ENABLE,180);
  //half speed delay(2000);
  analogWrite(ENABLE,128); //half speed
  delay(2000);
  analogWrite(ENABLE,50); //half speed
  delay(2000); analogWrite(ENABLE,128);
  //half speed delay(2000);
  analogWrite(ENABLE,180); //half speed
  delay(2000); analogWrite(ENABLE,255);
  //half speed delay(2000);
  digitalWrite(ENABLE,LOW); //all done
  delay(10000);
}
```

Lesson 21 A digital tube shows traffic light experiment

Overview:

After learning Lesson 17, we learned about the 74HC595 Shift Register. This lesson will use the 74HC595 shift register in combination with the red and green LEDs and the yellow LED to create a traffic light.

Component Required:



1 x Arduino Uno R3

1 x 830 tie-points breadboard

1 x 1 Digit 7-Segment Display

20 x M-M wire (Male to Male jumper wire)

1 x Red LED

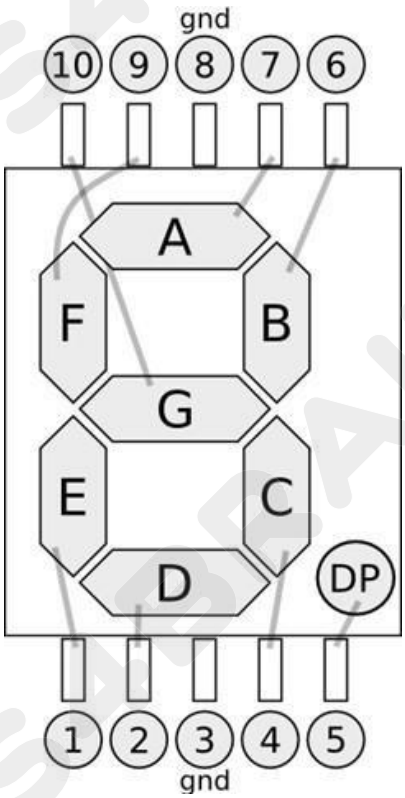
1 x Yellow LED

1 x Green LED

Component Introduction

Seven segment display

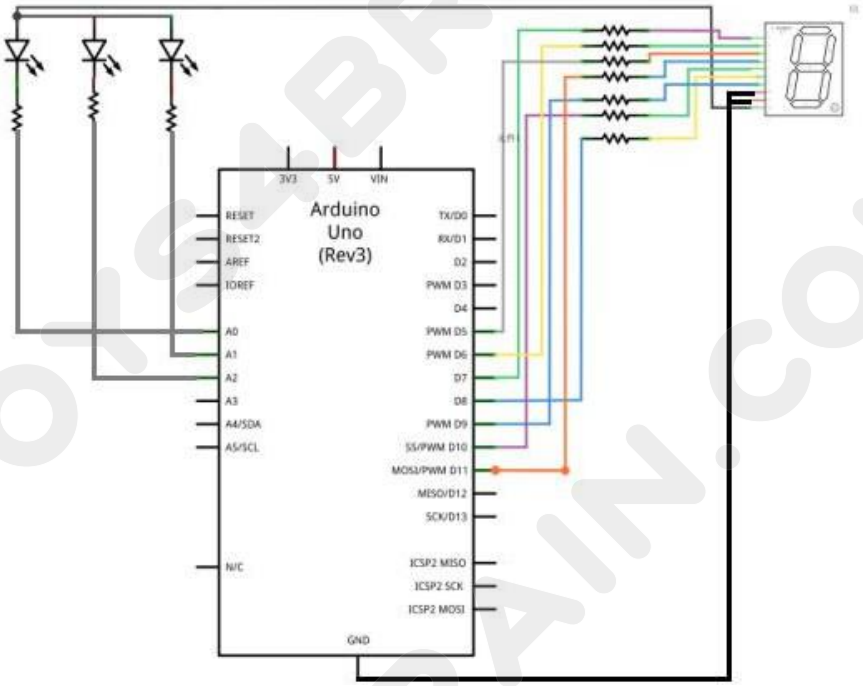
Below is the seven-segment pin diagram.



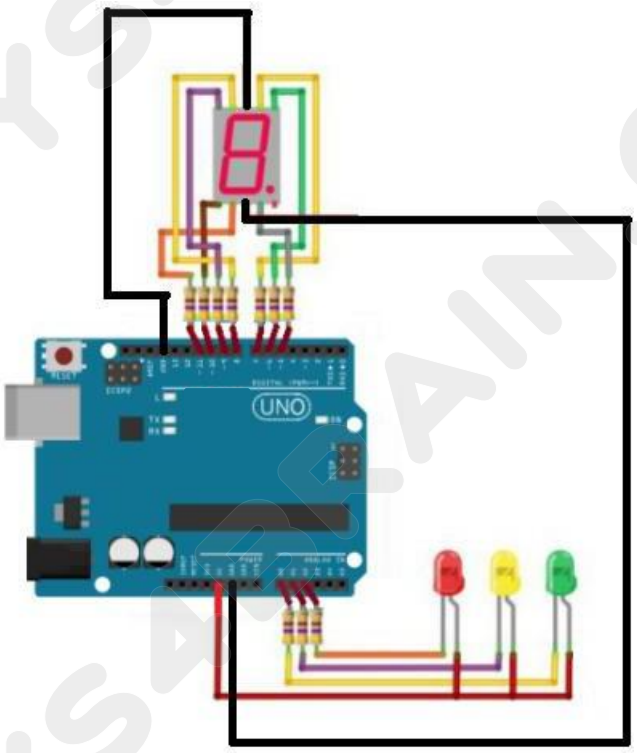
0-9 ten digits correspond with each segment are as follows (the following table applies common cathode seven segment display device, if you are using a common anode, the table should be replaced every 1 0 0 should all replaced by 1):

Display digital	dp	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
1	0	0	1	1	0	0	0	0
2	0	1	1	0	1	1	0	1
3	0	1	1	1	1	0	0	1
4	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1
6	0	1	0	1	1	1	1	1
7	0	1	1	1	0	0	0	0
8	0	1	1	1	1	1	1	1
9	0	1	1	1	1	0	1	1

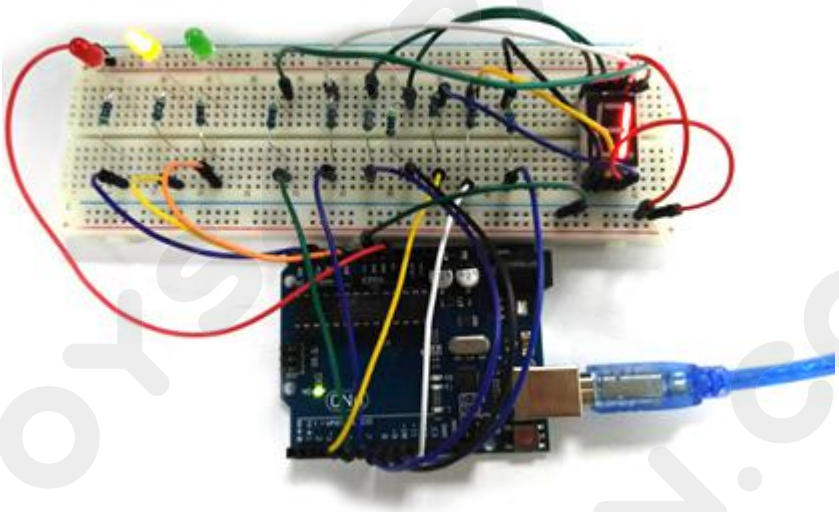
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
int a=7;
int b=6;
int c=5;
int d=11;
int e=10;
int f=8;
int g=9;

int ledG = 14;
int ledY = 15;
int ledR = 16;

//Display number 1
void digital_1(void)
{
unsigned char j;
digitalWrite(c,HIGH);//Lower the level of the digital 5 pin and light up segment c
digitalWrite(b,HIGH);//Light section b
for(j=7;j<=11;j++) //blanking
digitalWrite(j,LOW);
}
//Display number 2
void digital_2(void)
{
```

```
unsigned char j;
digitalWrite(b,HIGH);
digitalWrite(a,HIGH);
for(j=9;j<=11;j++)
digitalWrite(j,HIGH);
digitalWrite(c,LOW);
digitalWrite(f,LOW);
}
//Display number 3
void digital_3(void)
{
unsigned char j;
digitalWrite(g,HIGH);
digitalWrite(d,HIGH);
for(j=5;j<=7;j++)
digitalWrite(j,HIGH);
digitalWrite(f,LOW);
digitalWrite(e,LOW);
}


//Display number 4
void digital_4(void)
{
digitalWrite(c,HIGH);
digitalWrite(b,HIGH);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
digitalWrite(a,LOW);
digitalWrite(e,LOW);
digitalWrite(d,LOW);
}
//Display number 5
void digital_5(void)
{
unsigned char j;
for(j=7;j<=9;j++)
digitalWrite(j,HIGH);
digitalWrite(c,HIGH);
digitalWrite(d,HIGH);
digitalWrite(b,LOW);
digitalWrite(e,LOW);
}
//Display number 6
```

```
void digital_6(void)
{
unsigned char j;
for(j=7;j<=11;j++)
digitalWrite(j,HIGH);
digitalWrite(c,HIGH);
digitalWrite(b,LOW);
}
//Display number 7
void digital_7(void)
{
unsigned char j;
for(j=5;j<=7;j++)
digitalWrite(j,HIGH);
for(j=8;j<=11;j++)
digitalWrite(j,LOW);
}

//Display number 8
void digital_8(void)
{
unsigned char j;
for(j=5;j<=11;j++)
digitalWrite(j,HIGH);
}
//Display number 9
void digital_9(void)
{
digitalWrite(c,HIGH);
digitalWrite(b,HIGH);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
digitalWrite(a,HIGH);
digitalWrite(e,LOW);
digitalWrite(d,HIGH);
}
//Pin setting and initialization
void setup()
{
int i;    //defined variable
for(i=5;i<=16;i++)
pinMode(i,OUTPUT);      //Set pins 5 ~ 16 to output mode

}
```

```
void loop()
{
while(1)
{
//red light
digitalWrite(ledR,LOW);
digitalWrite(ledY,HIGH);
digitalWrite(ledG,HIGH);

digital_9();   // Show 9
delay(1000);    //Delay 1 s
digital_8();   // Show 8
delay(1000);    //Delay 1 s
digital_7();   // Show 7
delay(1000);    //Delay 1 s
digital_6();   // Show 6
delay(1000);    //Delay 1 s
digital_5();   // Show 5
delay(1000);    //Delay 1 s
digital_4();   // Show 4
delay(1000);    //Delay 1 s
digital_3();    // Show 3
delay(1000);    //Delay 1 s
digital_2();     // Show 2
delay(1000);    //Delay 1 s
digital_1();     // Show 1
delay(1000);    //Delay 1 s
//green light
digitalWrite(ledR,HIGH);
digitalWrite(ledY,HIGH);
digitalWrite(ledG,LOW);

digital_9();   // Show 9
delay(1000);    //Delay 1 s
digital_8();   // Show 8
delay(1000);    //Delay 1 s
digital_7();   // Show 7
delay(1000);    //Delay 1 s
digital_6();   // Show 6
delay(1000);    //Delay 1 s
digital_5();   // Show 5
delay(1000);    //Delay 1 s
digital_4();   // Show 4
```

```
delay(1000);      //Delay 1 s
digital_3();    // Show 3
delay(1000);      //Delay 1 s
digital_2();    // Show 2
delay(1000);      //Delay 1 s
digital_1();    // Show 1
delay(1000);      //Delay 1 s
//yellow light
digitalWrite(ledR,HIGH);
digitalWrite(ledY,LOW);
digitalWrite(ledG,HIGH);
digital_3();    // Show 3
delay(1000);      //Delay 1 s
digital_2();    // Show 2
delay(1000);      //Delay 1 s
digital_1();    // Show 1
delay(1000);      //Delay 1 s
}
}
```

Lesson 22 Four Digital Seven Segment Display

Overview

In this lesson, you will learn how to use a 4-digit 7-segment display.

When using 1-digit 7-segment display, please notice that if it is common anode, the common anode pin connects to the power source; if it is common cathode, the common cathode pin connects to the GND.

When using 4-digit 7-segment display, the common anode or common cathode pin is used to control which digit is displayed. Even though there is only one digit working, the principle of Persistence of Vision enables you to see all numbers displayed because each the scanning speed is so fast that you hardly notice the intervals.

Component Required:

1 x Arduino Uno R3

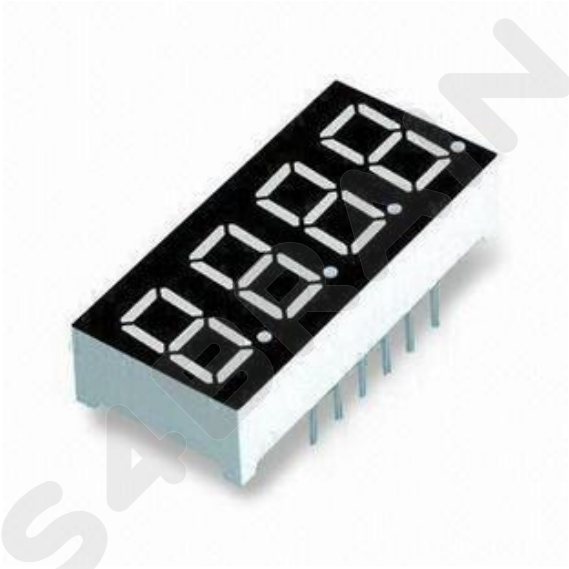
1 x 830 tie-points breadboard

1 x 74HC595 IC

1 x 4 Digit 7-Segment Display

4 x 220 ohm resistors

23 x M-M wires (Male to Male jumper wires)

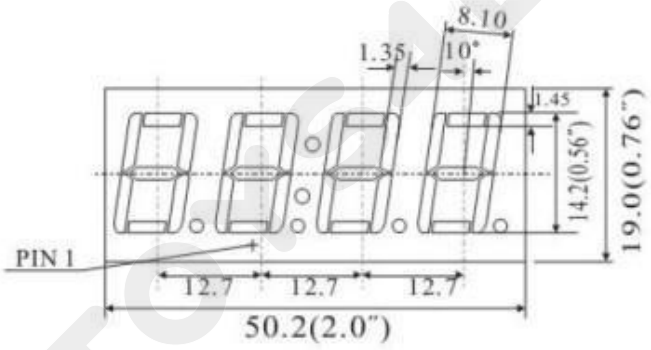


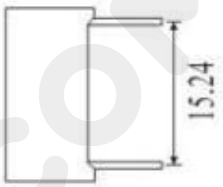
Component Introduction

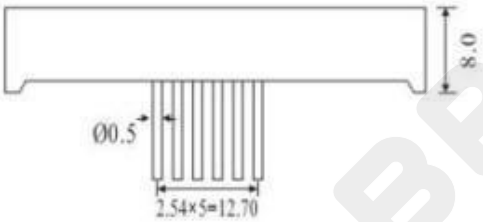
Four Digital Seven segment display

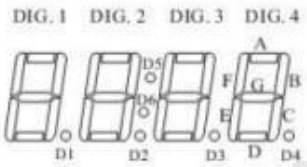
Package Dimensions

CPS05643AB



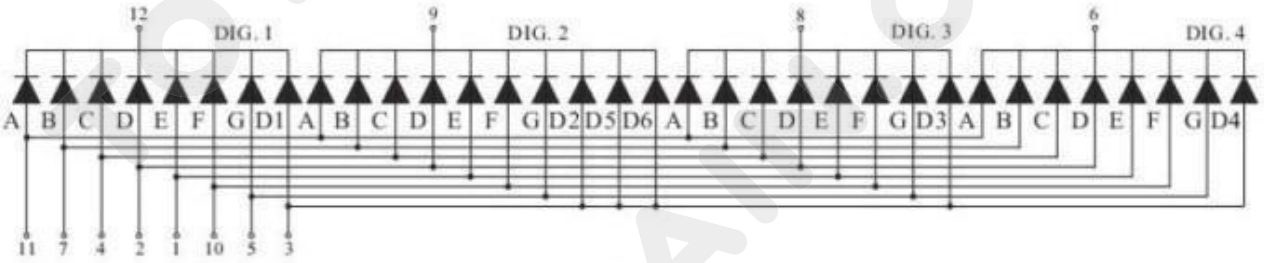




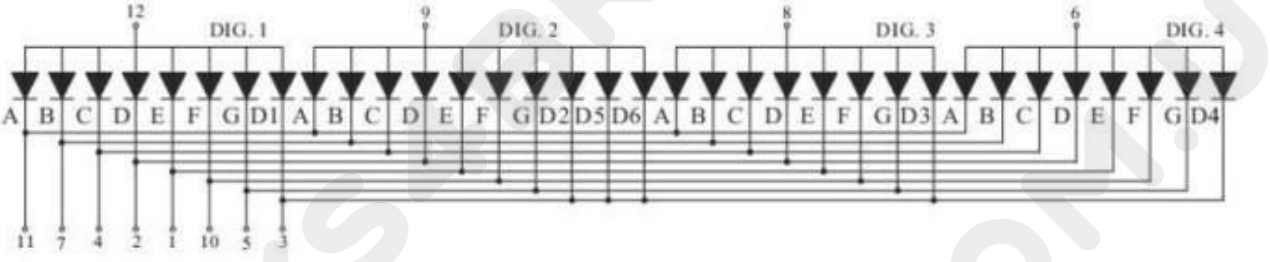


UNIT: MM(INCH) TOLERANCE: ±0.25(0.01")

Internal Circuit Diagram



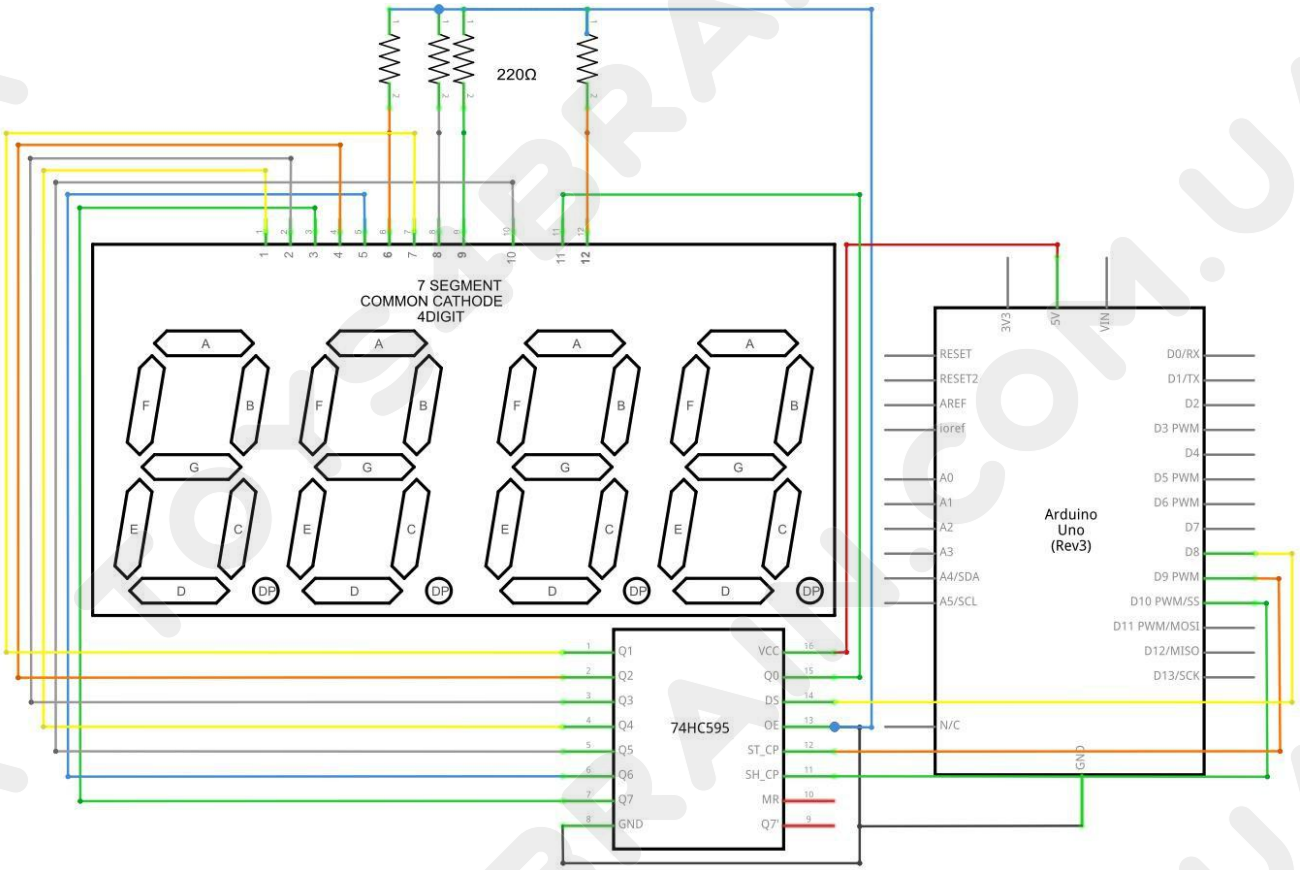
5643A



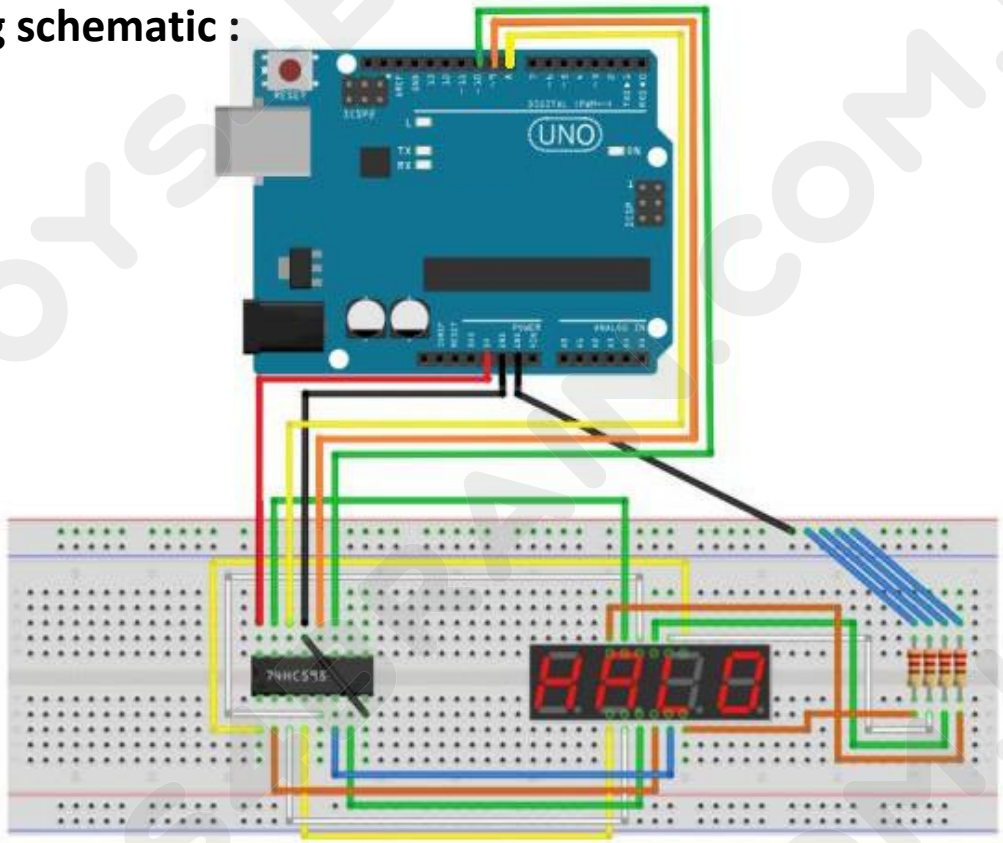
5643B

Four Digits Displays Series

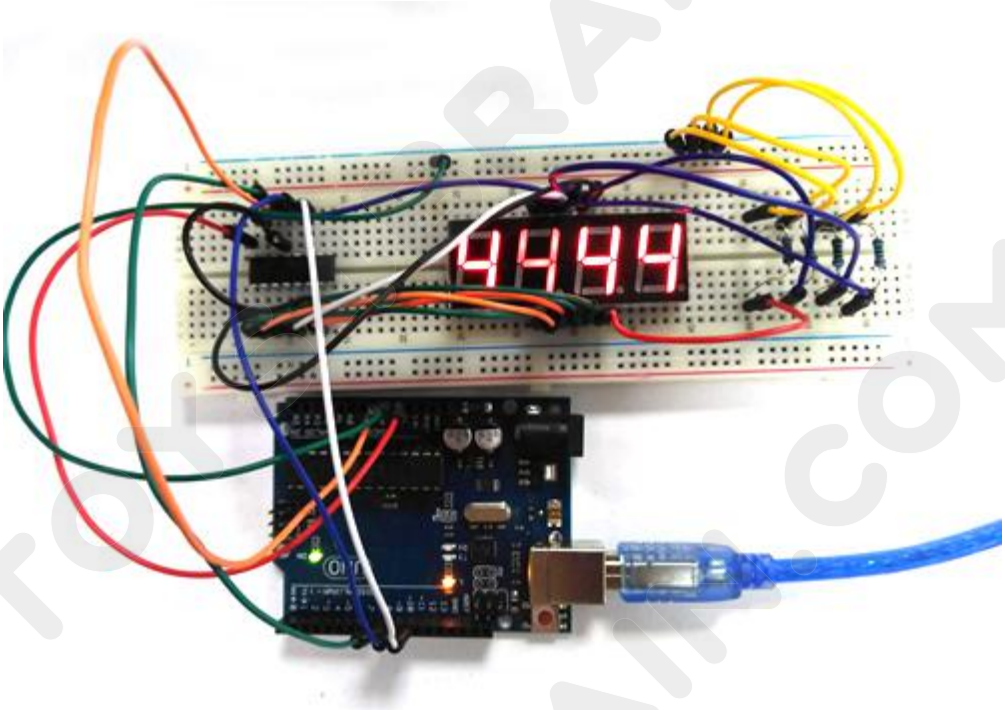
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
int latch=9;    //74HC595    pin 9 STCP
int clock=10; //74HC595    pin 10 SHCP
int data=8;     //74HC595    pin 8 DS
unsigned char table[]=
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c
,0x39,0x5e,0x79,0x71,0x00};
void setup()
   { pinMode(latch,OUTPUT)
   ;
   pinMode(clock,OUTPUT);
   pinMode(data,OUTPUT);
}
void Display(unsigned char num)
{
   digitalWrite(latch,LOW);
   shiftOut(data,clock,MSBFIRST,table[num]);
   digitalWrite(latch,HIGH);
}
```

```
void loop()
   { Display(1);
   delay(500);
   Display(2);
   delay(500);
   Display(3);
   delay(500);
   Display(4);
   delay(500);
   Display(5);
   delay(500);
   Display(6);
   delay(500);
   Display(7);
   delay(500);
   Display(8);
   delay(500);
   Display(9);
   delay(500);
   Display(10);
   delay(500);
   Display(11);
   delay(500);
   Display(12);
   delay(500);
   Display(13);
   delay(500);
   Display(14);
   delay(500);
   Display(15);
   delay(500);
}
```

Lesson 23 Relay Experiment

Overview

In this course you will learn how to use relay modules.

When the input variable (excitation) becomes a specified requirement, the relay is a component in which the amount of charge of the output circuit occurs due to a step change of an electric appliance. The company's relay modules can handle a wide range of other electrical components from 28V to 240V AC or DC power supplies. The MCU can be used to achieve the goal of timing control switches. Can be used in burglar alarm, toys, construction and other fields. A relay is an electrical control device. It has an interaction between a control system (also called an input circuit) and a control system (also known as an output circuit). Typically used in automatic control circuits, it is actually performed in the current operation of the "automatic control circuit".

Component Required:

1 x Arduino Uno R3

1 x 5v Relay

1 x 830 tie-points breadboard

1 x Fan blade and 3-6v dc motor

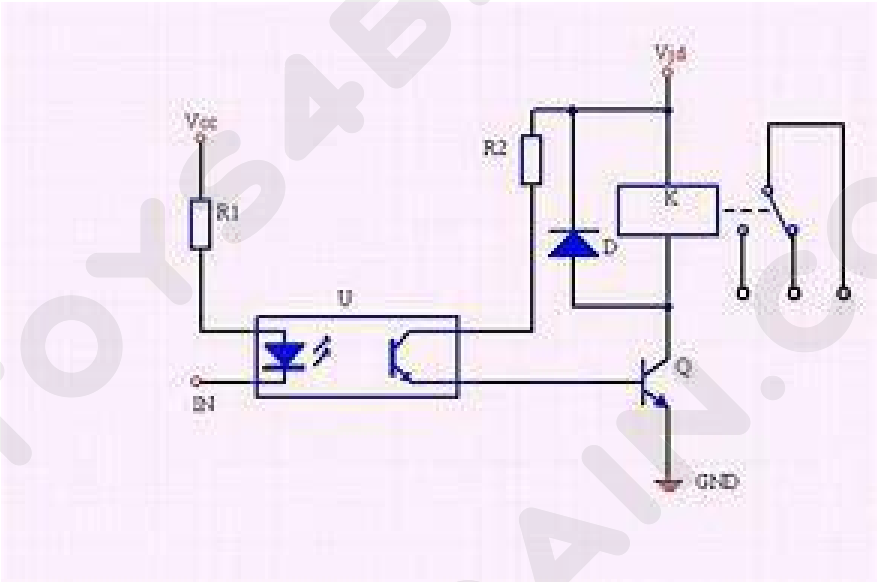
3 x M-M wires (Male to Male jumper wires)

3 x F-M wires (Female to Male DuPont wires)

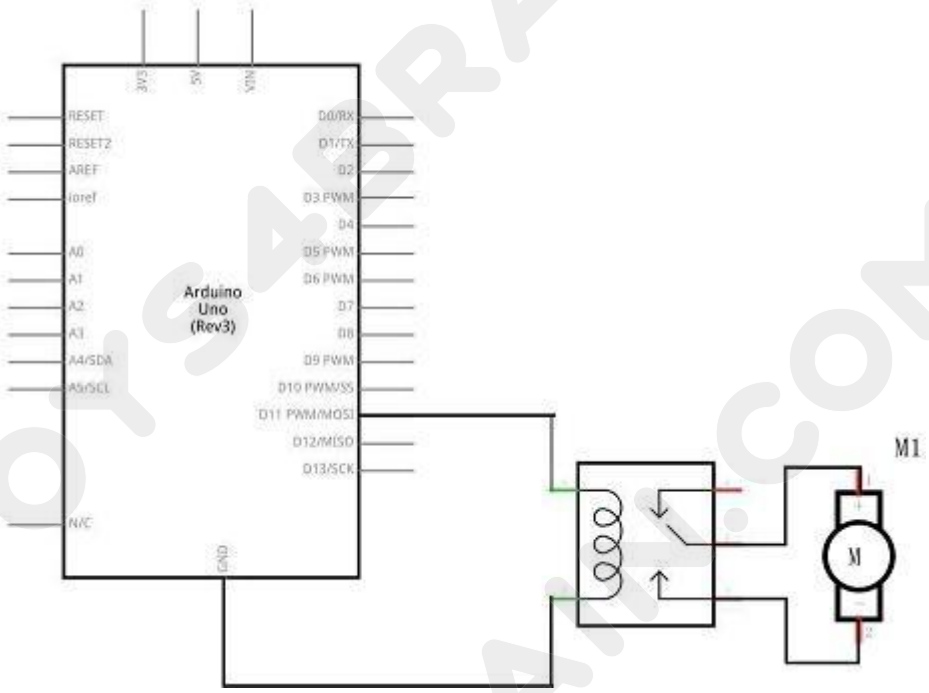
Component Introduction

Relay:

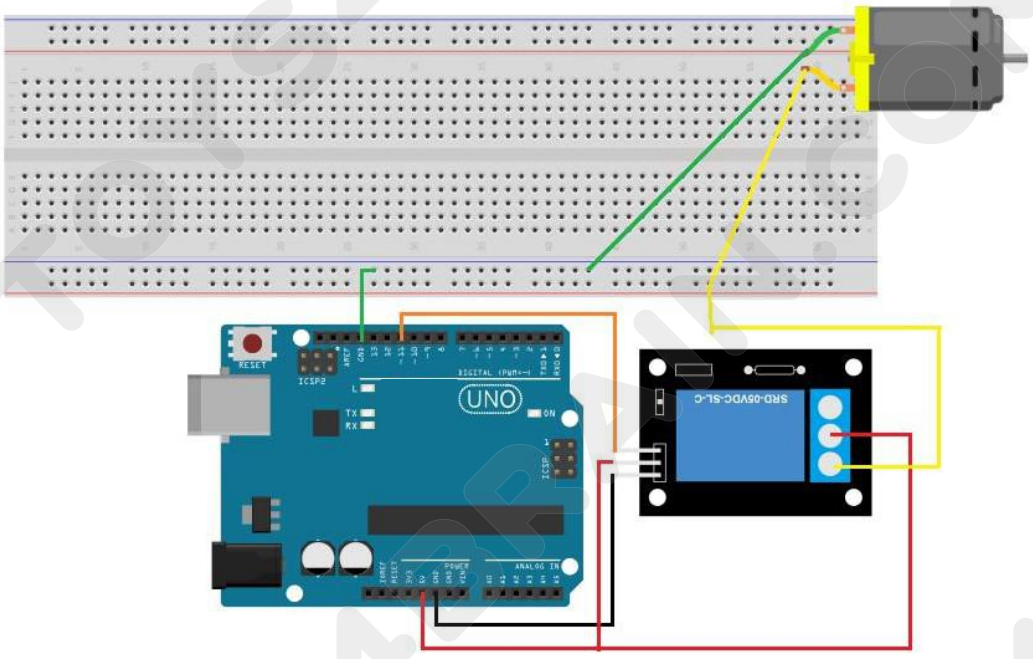
A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used as in solid-state relays. Relays are used where it is necessary to control a circuit by a low-power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal. The first relays were used in long-distance telegraph circuits as amplifiers. They repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations. A type of relay that can handle the high power required to directly control an electric motor or other loads is called a contactor. Solid-state relays control power circuits with no moving parts, instead using a semiconductor device to perform the switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults. In modern electric power systems, these functions are performed by digital instruments called "protective relays".
Below is the schematic of how to drive relay with Arduino.



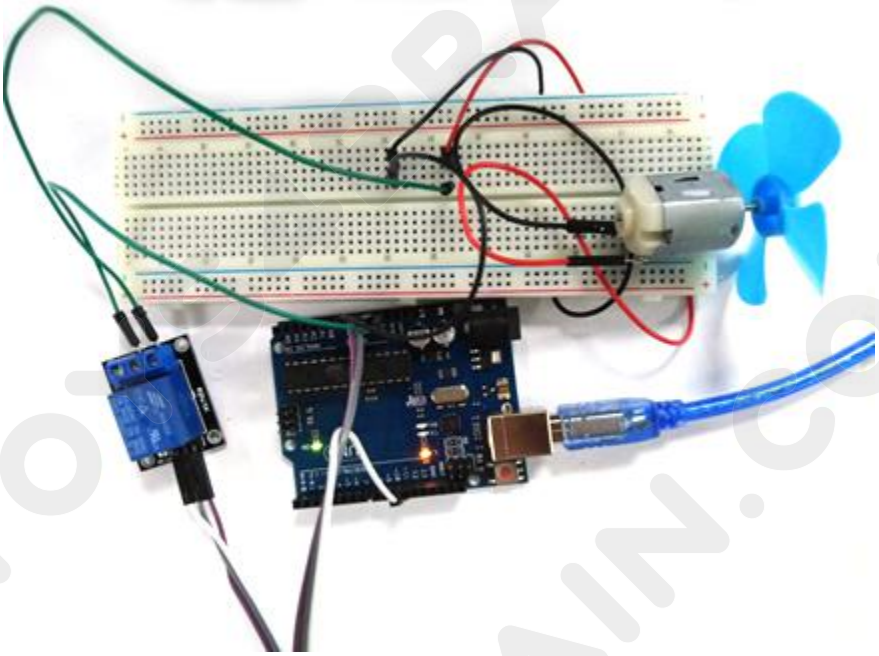
Connection Diagram:



Wiring schematic :



Physical wiring diagram:



Code:

```
 void setup()
{
   pinMode(11,OUTPUT);
}
 void loop()
{

  digitalWrite(11,HIGH);
  delay(4000);
  digitalWrite(11, LOW);
  delay(4000);

}
```

For the Arduino syntax, you can refer to the following link:

https://www.arduino.cc/reference/en/