ARDUINO BASIC STARTER KIT

Module: KT0004

Product Description:

Make it easy to learn ARDUINO.
This is the basic Starter Kit, developed specially for those beginners who are interested in Arduino. You will have a set of Arduino's most common and useful electronic components. What's more. We will offer you a detailed tutorials including project introduction and their source codes.You may learn about Arduino through using these basic projects. This kit will help you control the physical world with sensor.

Keyestudio Basic Starter Kit includes:

1X Arduino Uno R3 board
8x 220 Ω resistor
5x 1K Ω resistor
5x 10K Ω resistor
1x 10K Ω Pot
1x Joy stick Module & Knob
1x LED - RGB, Module
1x Buzzer (passive)
1x Buzzer (active)
4x button switch 6x6
10x button switch 12x12
1x LM35 Temp Sensor
2x Ball tilt sensor
3x Photo Resistor LDR
1x Flame sensor IR
1x IR receiver
1x IR remote control
1x 7-seg LED 1x module
1x 7-seg LED 4x module
5x LED - Blue
5x LED - Red
5x LED - Yellow
1x IC 74HC595N 16-pin DIP
1x Stepper Drive Module
1x1602 LCD
1x9g sevro
1x Relay, Module
1x Stepper motor 5-wire
1x 170-pin Breadboard Shield
1x 830-pin Breadboard
1x 6-cell AA Battery holder
1x 10 way ribbon cable socket
30x Assorted mail jumpers
1x 40 Way SIL header pins
1x USB cable
1x Project box 230x165x60mm

More at http://wiki.keyestudio.com/index.php/Main_Page

Project Index

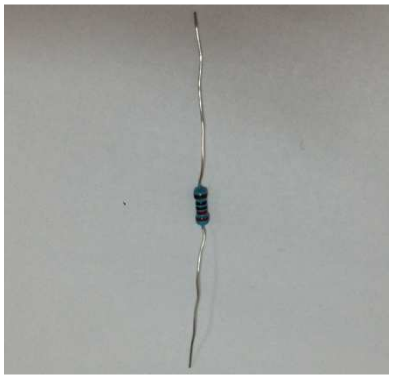
Flashing LED / Blinking LED:

LED light experiment is one of the basic experiments, The LED of Arduino has been used to the experiment, this time we use other I/O port and the external line LED lamp to finish the experiment, experiment components we need

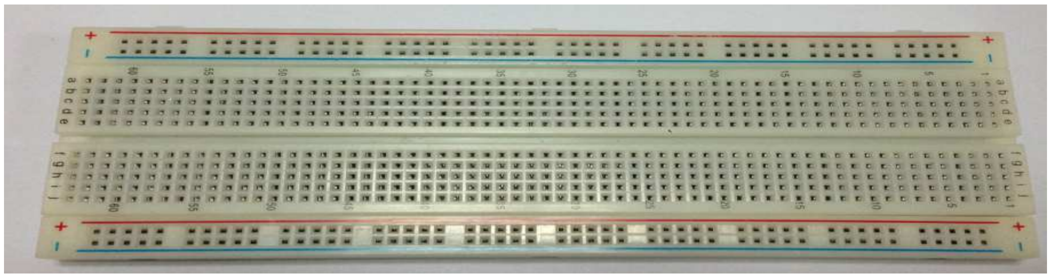
Arduino UNO controller and USB download cable experiments list is as follows:

1. Red 5mm LED*1

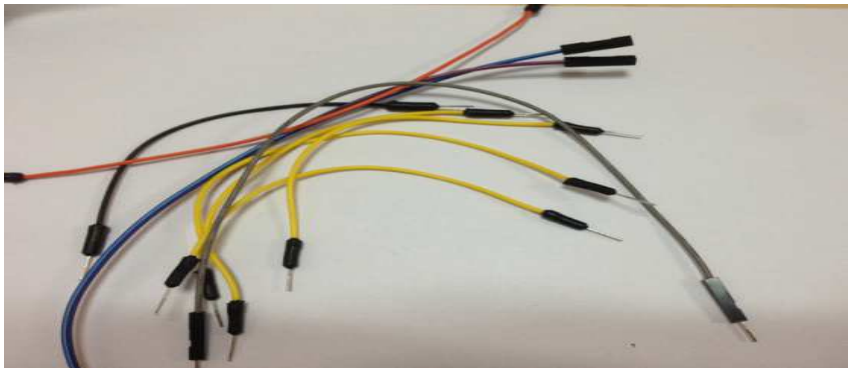
2. 220Ω directly insert resistor*1



3. Breadboard*1

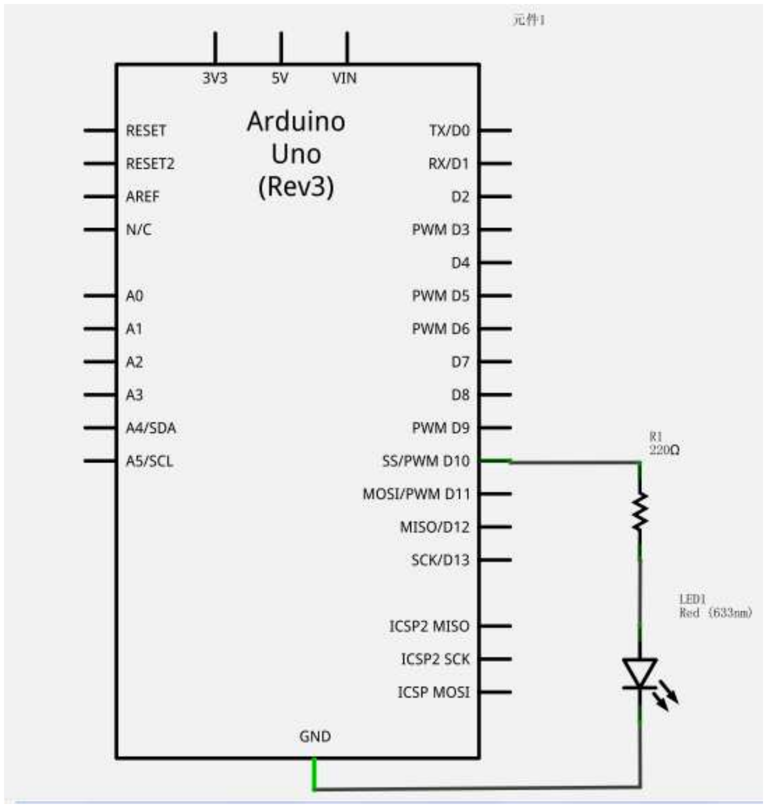


4. Breadboard jumper wires*1 lot



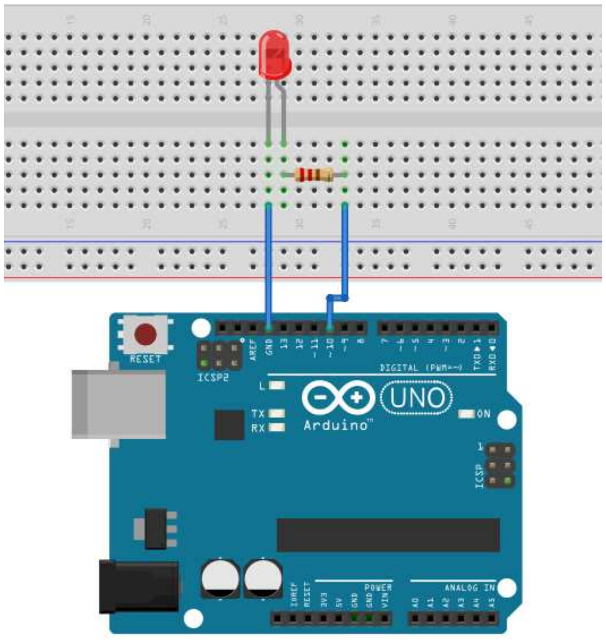
The next step we follow the lights Experimental schematic link physically as shown below, here we use on the digital port interface side. The use of light-emitting diode LED, to connect with a current limiting resistor, here a 220 ohm resistance, otherwise the current will be too high and burn the light emitting diode.

Experiment schematic



diagram

Connection diagram figure:



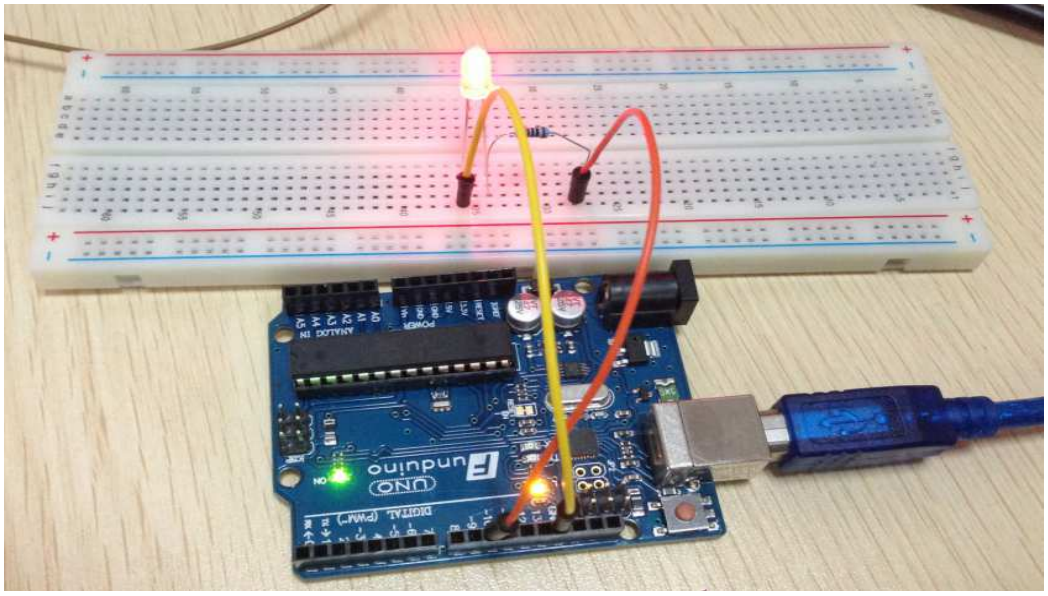
According to the above circuit, you can start writing programs, that flashes the LED ON for 1 second and 1 seconds OFF. This procedure is very simple and similar to the Arduino built-in routines except that it changes the Blink 13 digital port to number 10.

Reference program：

```
 int ledPin = 10;                              // Define Pin number 10

   void setup()
 {
 pinMode(ledPin, OUTPUT);          // set this pin as output
 }

 void loop()
 {
 digitalWrite(ledPin, HIGH);         // light ON
 delay(1000);                               // wait for 1 sec
 digitalWrite(ledPin, LOW);          // light OFF
 delay(1000);                               // wait for 1 sec.
 }                                                // END program
-----------------------------------------------------------
```

After download this program you can see our 10 port is connected with lights and flashing, so that our experimental result is LED keep flashing, interval is approximately one second.

The lights Blinking experiment is completed. Thank you.



##

Runing LEDs:

1) Experiment devices

- Led light x6

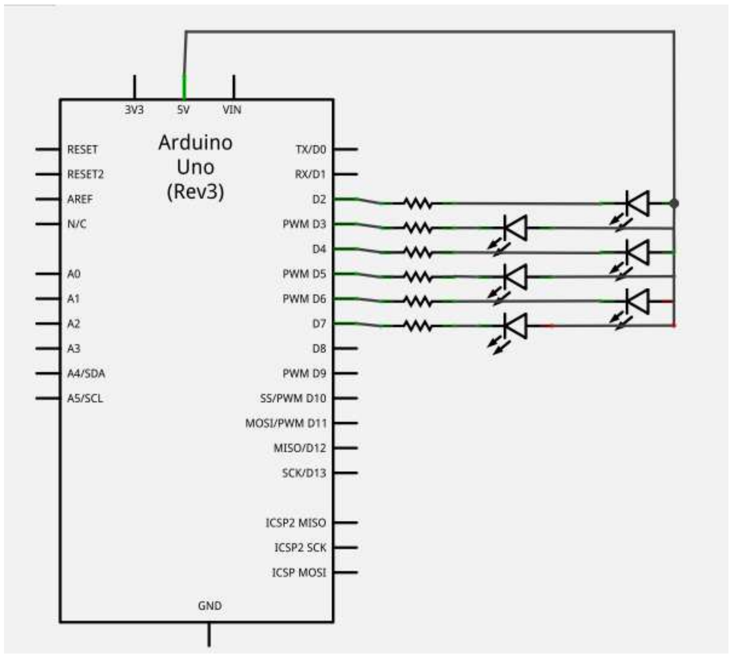
220Ω resistor x6

- Colorful breadboard experimental jumpers

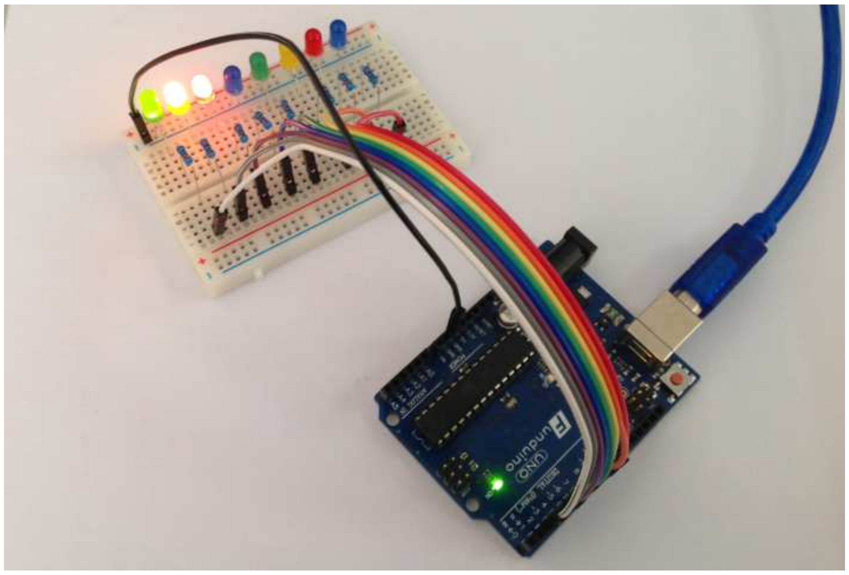
2) The experimental connection

According to the wiring method of diodes, six LED lights were attached to a digital pins 2~7. As shown in fig.:

Wiring flow water light experiment:



Connection：



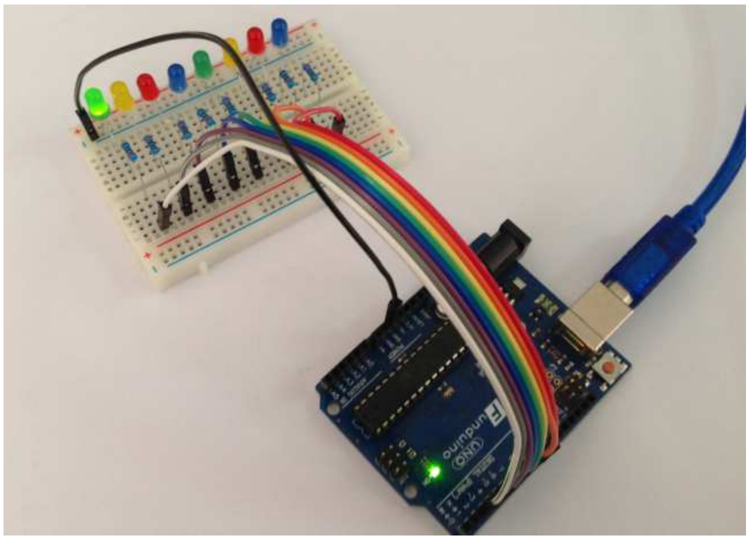
3) The experiment principle

In life we often see some billboards composed of various colors of LED lights. LED lamp in different locations constantly changing which form various effect. This section is the use of LED to program and simulate advertising lamp effect.

Program for Reference:

```
  int BASE = 2 ;    // The first LED connected I/O pin.
  int NUM = 8;      // number of LED used.

  void setup() {
  for (int i = BASE; i < BASE + NUM; i ++)    //Add +1
  {
  pinMode(i, OUTPUT);      // Set digital I/O pin as outputs
}
}

  void loop() {
  for (int i = BASE; i < BASE + NUM; i ++)    //Add +1
  {
  digitalWrite(i, LOW);       // Set the digital I/O pin output to "low", ie gradually turn off the lights
  delay(200); //Wait 200 milliseconds.
}
  for (int i = BASE; i < BASE + NUM; i ++)    //Add +1
  {
digitalWrite(i, HIGH);       // Set the digital I/O pin output to "high", ie gradually turn on the lights
delay(200); //Wait 200 milliseconds
}
}   // END program
```

Results: can see the LED light on the run.



##

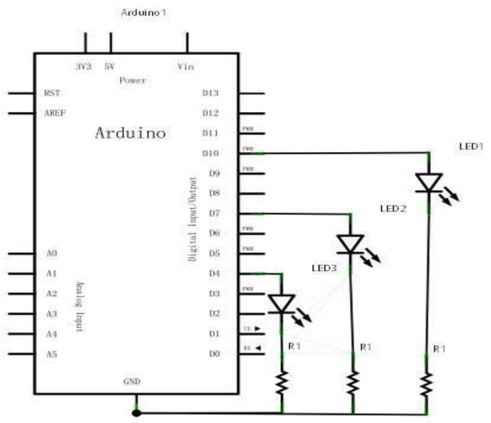
Traffic light design:

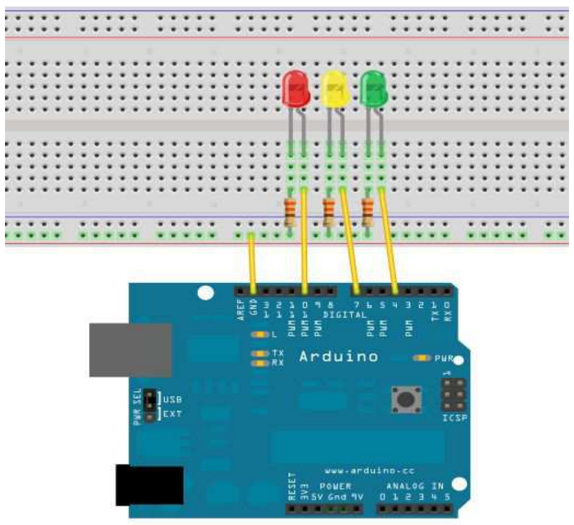
Above we have completed the control experiment of single lamp, then we'll do a slightly more complex traffic light experiment a bit, actually you can see out of this experiment is to extend the above single lamp experiment into 3 color light, we can realize our simulation traffic light experiment. We need to complete the experiment device with the Arduino controller and the download line and also need hardware are as follows:

Red M5 lamp LED*1
Yellow M5 Lamp LED*1
Green M5 Lamp LED*1
220Ωresistor *3
Breadboard*1
Breadboard jumpers*1lot

Ready for the element we can be started, we can infer other things from one fact in accordance with the above lights flashing experiment, below we provide reference schematic diagram, we use are 10, 7, 4, the digital interface.

Schematic diagram：



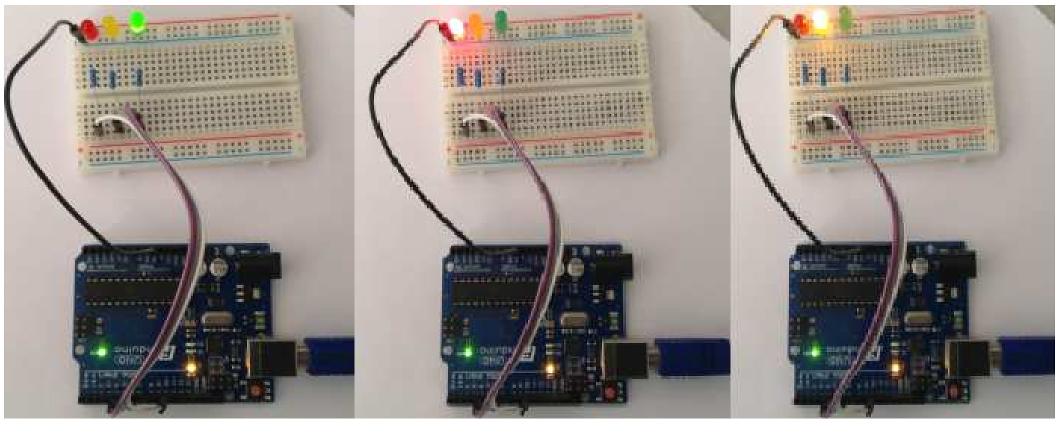


Since it is a simulation of traffic lights, traffic lights red yellow green lights flashing time need to simulate the real, we use Arduino delay () function to control the delay time, relatively the C language is much simpler.

Reference program :

```
int redled =10;                    // Define Red pin 8
int yellowled =7;                 // Define Yellow pin 7
int greenled =4;                  // Define Green   pin 4

void setup()
{
pinMode(redled, OUTPUT);          //Set Red pin as output
pinMode(yellowled, OUTPUT);       // Set Yellow pin as output
pinMode(greenled, OUTPUT);        // Set Green pin as output
}
void loop()
{
digitalWrite(greenled, HIGH);     // Green ON
delay(5000);                      //wait 5 seconds
digitalWrite(greenled, LOW);      // Green OFF
for(int i=0;i<3;i++)              // Flashes alternately three times with yellow blinking effect
{
delay(500);                       //wait 0.5 of a second
digitalWrite(yellowled, HIGH);    // Yellow ON
delay(500);                       //wait 0.5 of a second
digitalWrite(yellowled, LOW);     // Yellow OFF
}
delay(500);                       //wait 0.5 of a second
digitalWrite(redled, HIGH);       // Red ON
delay(5000);                      //wait 5 seconds
digitalWrite(redled, LOW);        // Red OFF
}                                 // End program
```

The download process is completed we can see our own design of traffic light control.
Note: this circuit design is very similar with flow water light.
Results: the green light will flash for 5 seconds, and then turn off, the yellow light flash 3 cycles, the red light flash for 5 seconds, followed by cycle.
Experiment is completed, thank you.



##

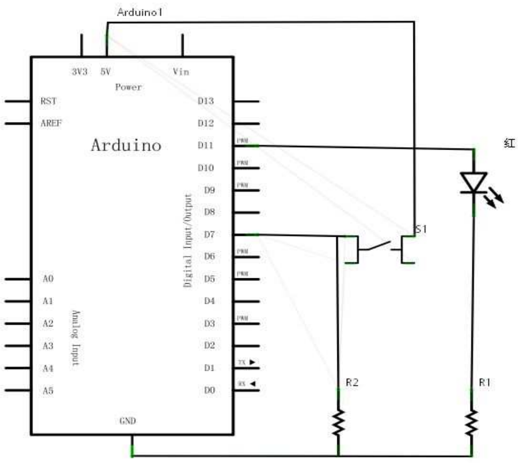
Key control LED:

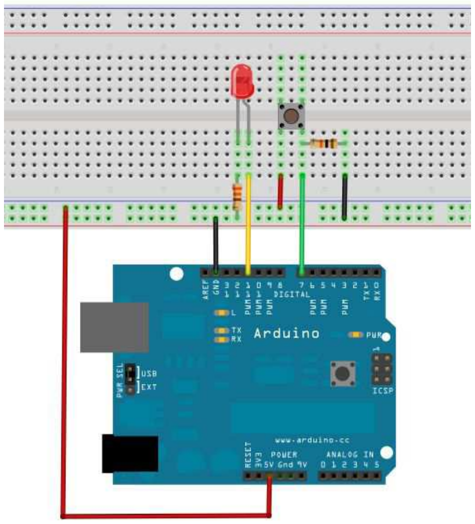
I/O means the INPUT interface and OUTPUT interface, small experiments so far we design are applied to the output function of Arduino I/O. In this experiment, we will work on the input function. Using Arduino I/O port to read external equipment output value, we complete an experiment combined input output using a button and an LED, so that we can simply understand the role of I/O. Key switch, everyone should know, which belongs to the switch quantity (digital) component, pressed for closure (conducting) state. To complete this

experiment we need to use the elements are as follows :

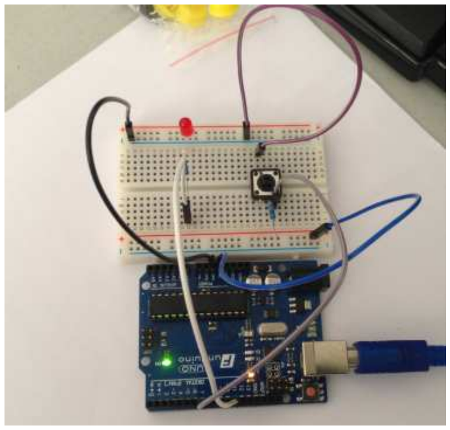
Key switch*1
Red M5 lamp LED*1
220Ωresistor*1
10KΩresisitor*1
Breabboard*1
Breadboard jumpers*1 lot

We will connect the key to digital port pin 7, red LED and attach to digital port pin 11 the tactile switch. Arduino controller can be used to pick up the switch keys and the lights. Can use ant port pin 2 to 13, try not to use port 0 and 1 as this is the ports for programming and serial monitoring, leave this to the experts. See interface circuit according to the diagram below.





Connection :



Below we begin to programming, let the lights light up when button is pressed, according to the previous study that the program can be easily written, relative to the previous several experiments with this experimental program add a conditional statement, here we use the if statement, the Arduino program writing statement is based on the C language, so the conditional statements C naturally are also applied to Arduino, like while, switch etc.. Here we use the IF () statement which is simple and easy to understand for everyone to do demo.

We analyze the circuit when the button is pressed, the digital pin 7 can be read at a high level, then we make the digital pin 11 output high level can make the LED ON, we judge if digital pin 7 is low level in the program, if it is, to low the digital pin 11 output for low, LED OFF. The same principle as above.

Program for reference :

```
int ledpin=11;               //set LED pin
int inpin=7;                 //set switch pin
int val;                     // define the variable val.

 void setup()
{
pinMode(ledpin,OUTPUT);   // set LED pin as output
pinMode(inpin,INPUT);       //set switch pin as input
}

void loop()
{
val=digitalRead(inpin);            // read digital pin 7 level assigned to val.
if(val==LOW)                       // Detect whether the button is pressed or not
{ digitalWrite(ledpin,LOW);}
else
{ digitalWrite(ledpin,HIGH);}
}                                      // END program
```

The experimental results, and the phenomenon: when the button is pressed, LED lights up, not pressed, LED does not light up.

This is pretty useless circuit as power to the switch will do the same. So let's make it latch.

Using the same circuit and only changing the software.

Program for reference：

```
int ledpin=11;          //set LED pin
int inpin=7;          //set switch pin
int val;                  // define the variable val.
boolean lastButton = LOW;      //boolean = True or Fauls, High or Low, On or Off, 0 or 1.
boolean ledOn = false;          // current state of LED

  void setup()
{
pinMode(ledpin,OUTPUT);    // set LED pin as output
pinMode(inpin,INPUT);    //set switch pin as input
}

void loop()
{
val=digitalRead(inpin); // read digital pin 7 level assigned to val.
if(val==LOW && lastButton ==LOW)      // if it is, the buttonState is HIGH:
{
      // turn LED on:
   ledOn = !ledOn;                // ! = invert last value
   lastButton = HIGH;
  }
else
{
    // turn LED off:
    lastButton = digitalRead(inpin);              //set last button action
    }
 {
  digitalWrite(ledpin, ledOn);
  }
 }                   // END program
```

Copy this program, the lights with keys experiment is completed, the principle of this experiment is very simple, widely used in a variety of circuit and electric appliances, real life, everyone is not difficult to find that in a variety of devices, such as mobile phone when you press any button backlight lamp will light up, this is the typical application. You can think of LED as a relay, it can control the 220V lamp!

##

Tilt switch:

Tilt switch control LED light

1. The experimental device:

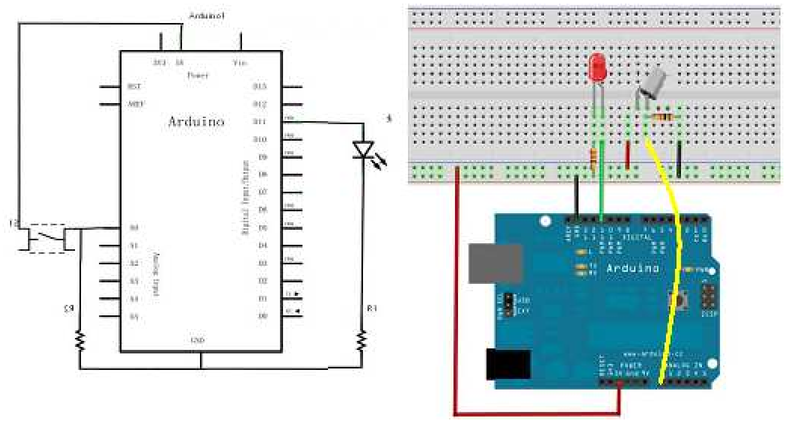
Tilt ball switch

Led x1

220Ω resistor

Colorful breadboard jumper lines: a certain number

The physical connection diagram:



The the LED is connected to the digital pin 8, tilt ball switch connected to the analog pin 5.

The experiment principle:

When one side of the switch is tilt below the horizon, switch get through, analog port voltage value is about 5V (digital binary is 1023), light up led lights. When the other end tilt lower than the switch horizontal position, analog port voltage value is about 0V (digital binary is 0), off the LED lamp. Analog port voltage value in the procedure is higher than about 2.5V (the digital binary is in the representation of 512), can know whether the tilt ball switch get through.

Reference program :

```
 void setup()
{
  pinMode(11,OUTPUT);                //set pin 8 output (LED)
}
void loop()
{
int i;// Defining value i
while(1)
{
  i=analogRead(0);                   //read analog
  if(i>512)                          //If more than 512 (2.5V)
  {
    digitalWrite(8,LOW);             //LED ON
  }
  else                              // otherwise
   {
    digitalWrite(8,HIGH);            //LED ON
  }
 }
}
```

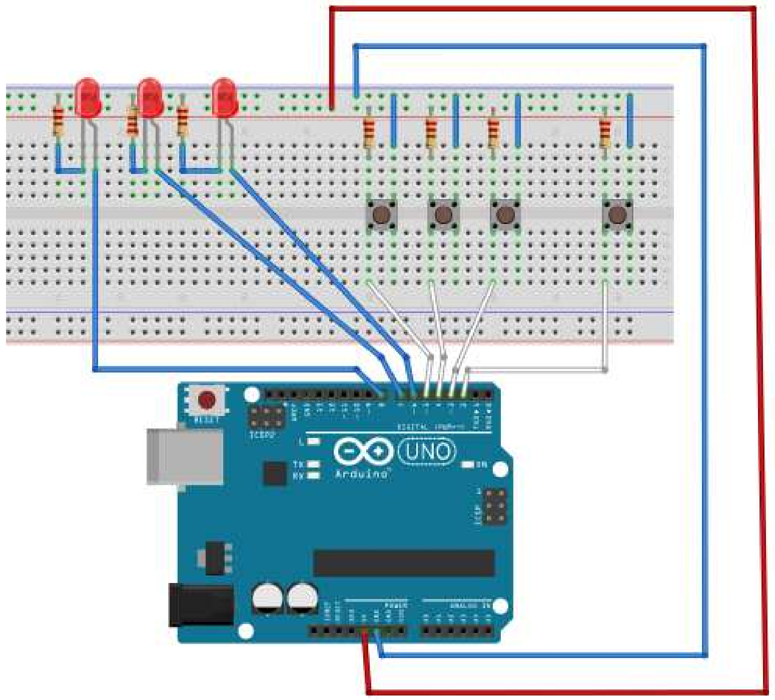
The experimental phenomenon: the hand hold the bread board, when the inclination to a certain extent, LED is lit. No tilt, LED is not lit.

Inclined to a certain extent, LED is lit, control in practical applications is usually relay.

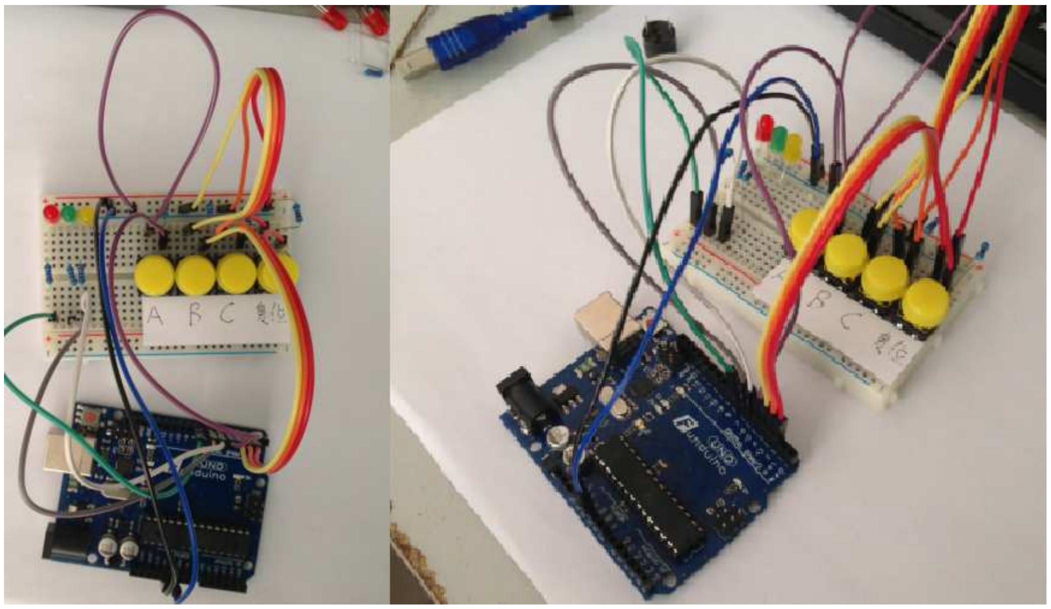
###

Quiz Game Controller, First to respond experimental design

After the completion of the above experiments that a lot of friends can independently finish the experiment, this experiment is the key control the lamp experiment expansion into 3 keys corresponding to 3 small LED, occupying 7 digital I/O interface with a Quiz master reset key.



The circuit connection diagram



Program for reference:

```
int redled=8;           //set LED Red
int yellowled=7;        //set LED Yellow
int greenled=6;         //set LED Greed
int redpin=5;           //set LED Red
int yellowpin=4;        //set LED Yellow
int greenpin=3;         //set LED Greed
int restpin=2;          //set reset button
int red;
int yellow;
int green;
void setup()
```

```
{
pinMode(redled,OUTPUT);             //set output LED
pinMode(yellowled,OUTPUT);          //set output LED
pinMode(greenled,OUTPUT);           //set output LED
pinMode(redpin,INPUT);              //set input SW
pinMode(yellowpin,INPUT);           //set input SW
pinMode(greenpin,INPUT);            //set input SW
}
void loop()                         // Press the key to start scan cycle.
{
red=digitalRead(redpin);
yellow=digitalRead(yellowpin);
green=digitalRead(greenpin);
if(red==LOW)RED_YES();
if(yellow==LOW)YELLOW_YES();
if(green==LOW)GREEN_YES();
}

void RED_YES()                      //The red light is ON until the reset button is pressed, ending the cycle
{
  while(digitalRead(restpin)==1)
  {
   digitalWrite(redled,HIGH);
   digitalWrite(greenled,LOW);
   digitalWrite(yellowled,LOW);
  }
  clear_led();
}
void YELLOW_YES()                   //The yellow light is ON until the reset button is pressed, ending the cycle
{
  while(digitalRead(restpin)==1)
  {
  digitalWrite(redled,LOW);
  digitalWrite(greenled,LOW);
  digitalWrite(yellowled,HIGH);
  }
  clear_led();
}
void GREEN_YES()                    //The green light is ON until the reset button is pressed, ending the cycle
{
  while(digitalRead(restpin)==1)
  {
  digitalWrite(redled,LOW);
  digitalWrite(greenled,HIGH);
  digitalWrite(yellowled,LOW);
  }
  clear_led();
}
void clear_led()                //clear all LED’s
{
  digitalWrite(redled,LOW);
  digitalWrite(greenled,LOW);
  digitalWrite(yellowled,LOW);
}
```

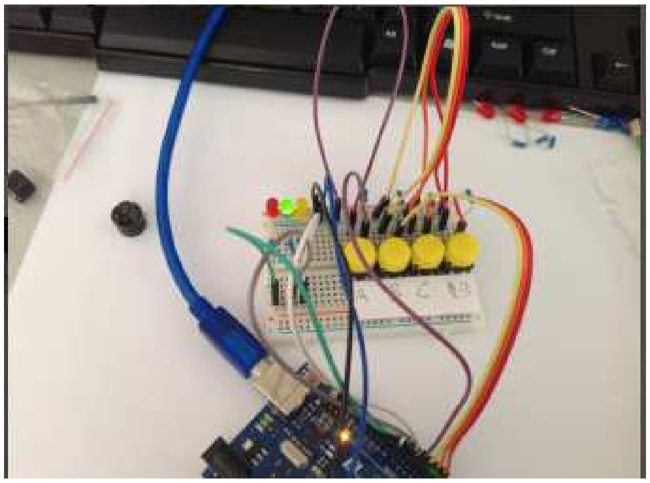
The experimental phenomenon, the responder who reacted first by pressing, there LED will light up!

Then press the REST button to reset the LED to OFF.

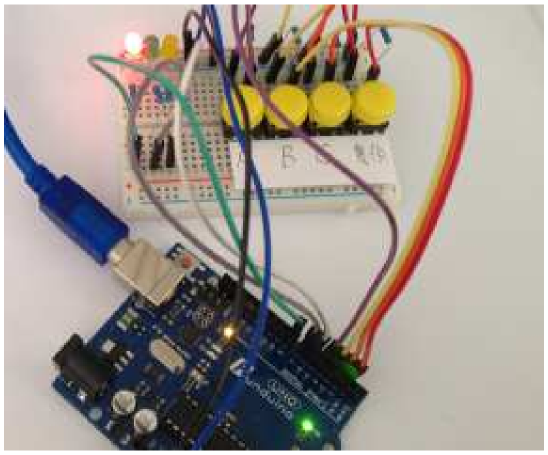
After downloading this program, our own simple responder is complete.

Screenshot:

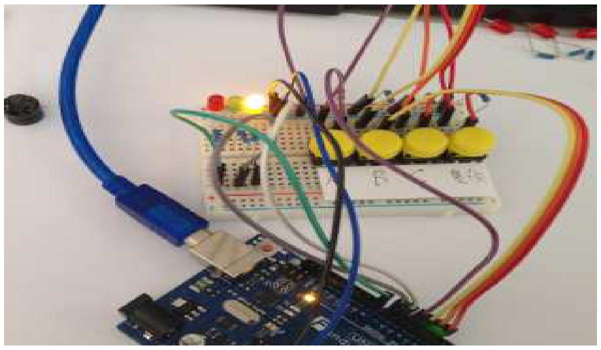
B Response successfully：Green light flash



B Response successfully：Red light flash



C Response successfully：Yellow light flash



##

Control RGB LED module

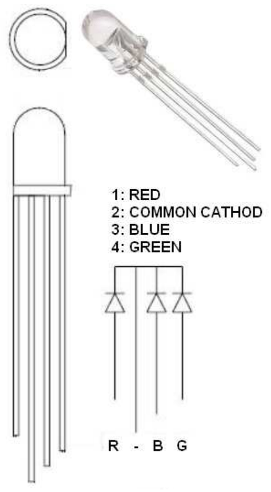
The display principle, the principle is in a LED lamp, there are three kinds of colors of light, red, green, blue. The driving voltage on each color of light is not the same. The brightness is not the same. They are combined together, form a variety of colors.

3 colors input of the module

1. R = Input for Red
2. G = Input for Green
3. B = Input for Blue

Features of the module:

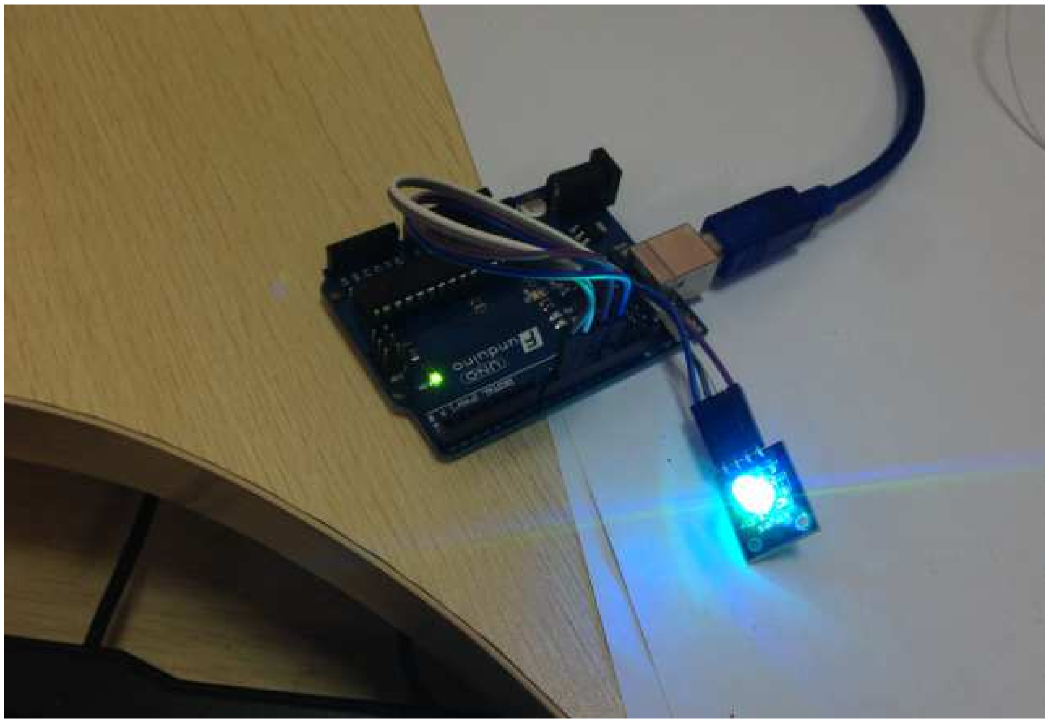
R,G,B, module has 3 colors that can be controlled by a MCU to create a full color spectrum.



Program for reference:

```
int ledPin = 13;              // LED is connected to digital pin 13
int redPin = 11;              // R red LED pin pin 11
int greenPin = 9;             // G green LED pin pin 9
int bluePin = 10;             // B blue LED pin pin 10

void setup()
{
  pinMode(ledPin, OUTPUT);     //set at output
  pinMode(redPin, OUTPUT);     //set at output
  pinMode(greenPin, OUTPUT); //set at output
  pinMode(bluePin, OUTPUT);   //set at output
}

void loop()                    // run over and over again
{
          // Basic colors:
          color(255, 0, 0);    //red
          delay(1000);         //wait 1 second
          color(0,255, 0);     //green
          delay(1000);         //wait 1 second
          color(0, 0, 255);    //blue
          delay(1000);         //wait 1 second
```

```
        // Example blended colors:
        color(255,255,0);        //yellow
        delay(1000);             //wait 1 second
        color(255,255,255);      // white
        delay(1000);             //wait 1 second
        color(128,0,255);        // purple
        delay(1000);             //wait 1 second
        color(0,0,0);            //all OFF
        delay(1000);             //wait 1 second
}

void color (unsigned char red, unsigned char green, unsigned char blue)    //Color control
function
{
        analogWrite(redPin, 255-red);
        analogWrite(bluePin, 255-blue);
        analogWrite(greenPin, 255-green);
}
```



##

Controlling LEDs with 74HC595:

74HC595 is simply an 8 bit shift register and a memory, and a three state output function. Here we use it to control 8 LED lights. Why do we need to use the 74HC595 to control the lights? There will be many friends will ask this question, I would like to ask is if we use Arduino control 8 small lights to occupy how many I/O? The answer is 8, but how many I/O ports Arduino UNO has? Including the simulation interface is about 20.These 8 LED's take too many resources. We can reduce the number I/O ports using the 74HC595. Using 74HC595 we can use 3 digital I/O port to control 8 LED lights. Here are the components we need to prepare.

74HC595 directly insert chip*1

Red 5mm LED*4

Green 5mm LED*4

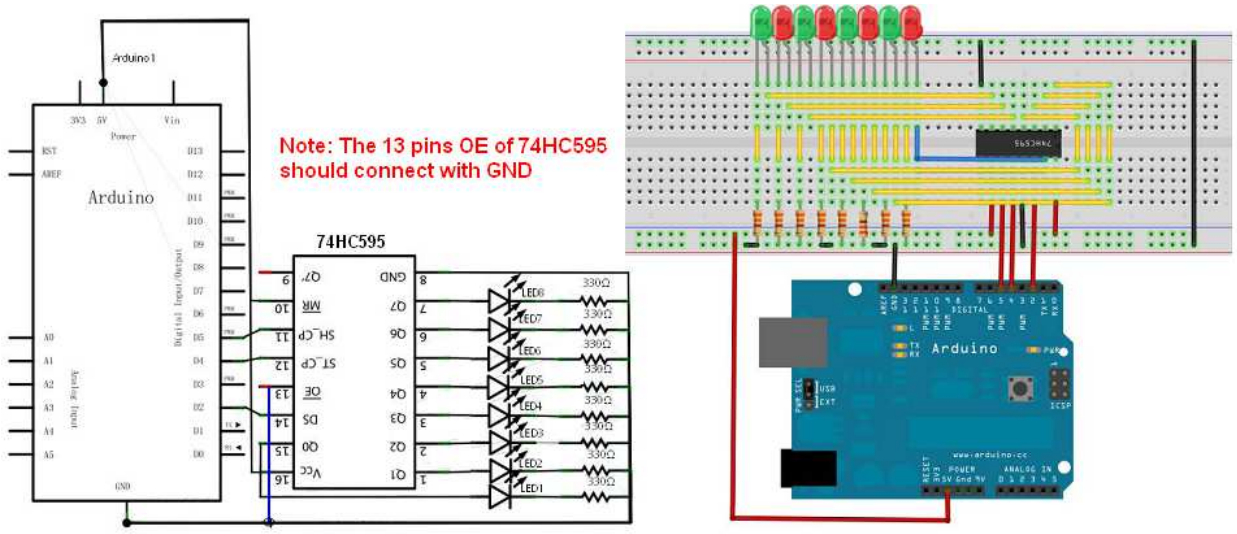
220Ω resistor*8

Breadboard*1

Breadboard jumper lines*1 lot

Ready for the element we connect circuit according to the diagram below.

Note: The 13 pins OE of 74HC595 should connect with GND



This circuit diagram may seem complicated. After we carefully analysis and combined with real projects will find it easy.

Program for reference:

```
int data = 2;               //set pin 14 of 74HC595 data input SER
int clock = 5;              // set pin 11 of 74HC595 clock
int latch = 4;              // set pin 12 of 74HC595 Output memory latch line RCLK
int ledState = 0;
const int ON = HIGH;
const int OFF = LOW;
void setup()
{
pinMode(data, OUTPUT);      // set output
pinMode(clock, OUTPUT);     // set output
pinMode(latch, OUTPUT);     // set output
}
```

```
void loop()
{
for(int i = 0; i < 256; i++)
{
updateLEDs(i);
delay(500);
}
}
void updateLEDs(int value)
{
digitalWrite(latch, LOW);                    // latch OFF
shiftOut(data, clock, MSBFIRST, ~value);  //Serial data output, high first
digitalWrite(latch, HIGH);                   // latch ON
}
```

You can see 8 lights flashing beautifully after you uploaded the program.
The experimental phenomenon is to see eight LED display eight bit binary number, cycle 1.

##

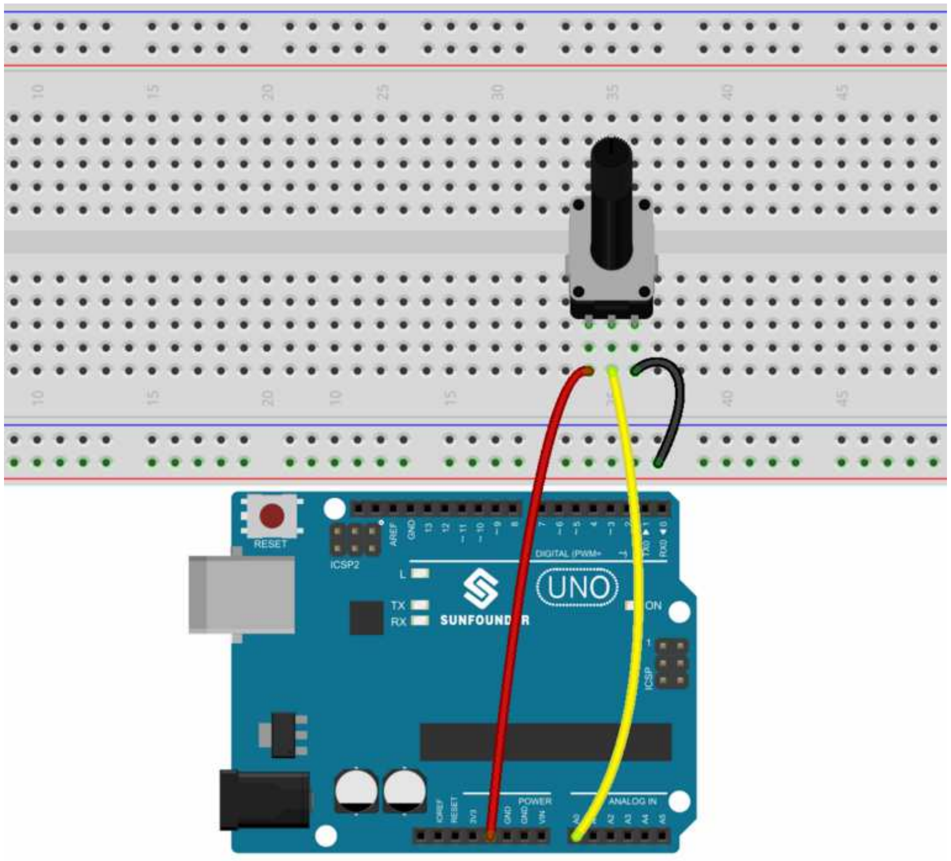
Analog value read:

This experiment we have to start learning about the use of analog I/O interface, Arduino has Analog 0 to 5, totally 6 analog interfaces, the 6 interface can also be counted as interface function reuse, besides the analog interface, the 6 interfaces can be used as digital interfaces, the number is 14 – 19, after simply understanding, let's begin our experiment. Potentiometer is a typical familiar value output component. This experiment is completed by it.

Components for use：

Potentiometer *1
Breadboard*1
Breadboard jumper lines *1 lot

In this experiment, we will read out analog values which are transited by potentiometer resistance values, and then displayed on the screen, is the example of the application that we are required to master after the completion of this experimental functions.
We connect the physical map first according to the following circuit:

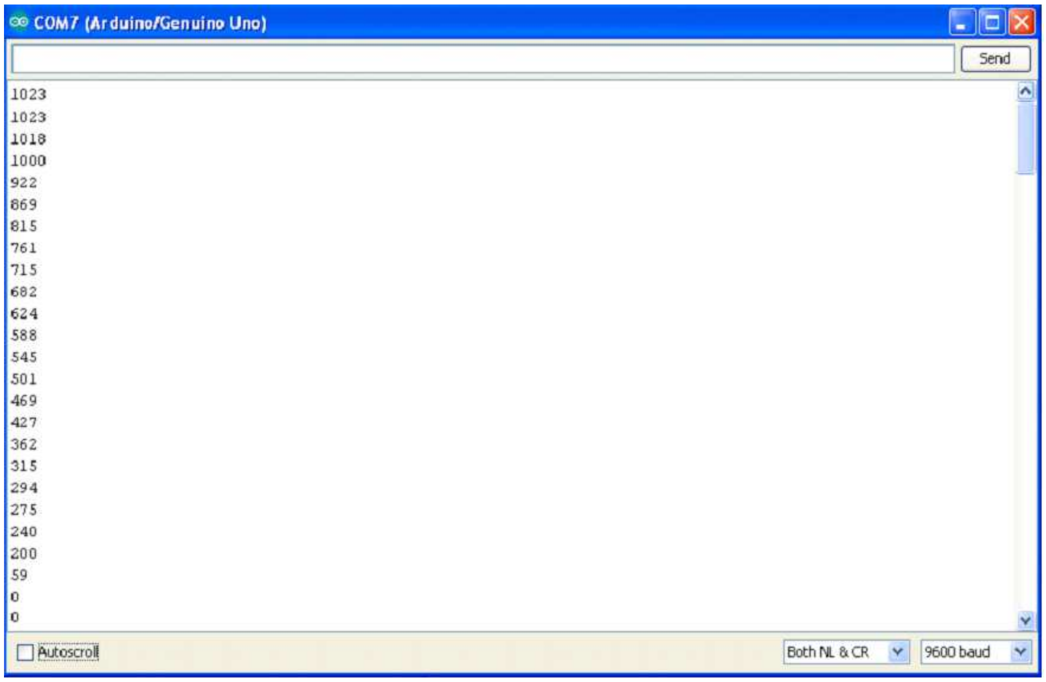


We use the analog interface 0.
Program is also very simple, a analogRead (); statement can read analog export value, Arduino 328 is a 10 bit A/D collection, so read analog value range is 0-1023, there is a difficult problem in this experimental procedure that displaying on the screen of the numerical problems, learning it is also very simple. We must first set baud rate in voidsetup (), displaying the number belongs to Arduino and PC communication, so the baud rate of Arduino should be as the same as to the correct value showed by PC software sets, otherwise it will be garbled or is not displayed, one button in the lower right angle of Arduino software monitor window can set the baud rate, baud rate set here should be the same as the baud rate which is set with the program void setup ().The program should set the baud rate as Serial.begin(); statement in parentheses is baud rate value. Second is the display numerical statement, Serial.print (); or Serial.println (); the difference is that the latter shows the value and enter automatically, the former does not, have introduced more about the statement before.

Reference program :

```
int potpin=0;                //set pot to A0
int ledpin=13;               //set onboard LED
int val=0;// variable val will be defined and assigned an initial value of 0

void setup()
{
pinMode(ledpin,OUTPUT);          //set output LED
Serial.begin(9600);// Set the baud rate to 9600
}
void loop()
{
digitalWrite(ledpin,HIGH);           //LED ON
delay(50);                           //wait 0.05 seconds

digitalWrite(ledpin,LOW);            //LED OFF
val=analogRead(potpin);              // Read the value of analog interface 0 and assign it to val
Serial.println(val);                 //print value to serial monitor
}
```

Reference program uses Arduino digital 13 built-in LED lights, read a value each time, lights will flash for once.
Below is the readout value simulation.
The analog value is read out:



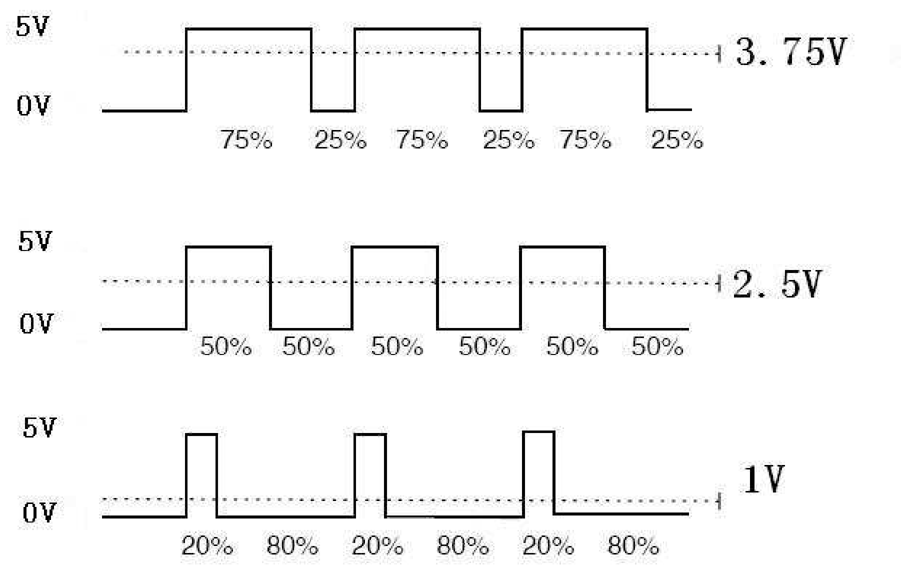
```
1023
1023
1018
1000
922
869
815
761
715
682
624
588
545
501
469
427
362
315
294
275
240
200
59
0
0
```

The experimental phenomenon: when you rotate the potentiometer knob can see changes in the numbers on the screen, read analog value this method will always be with us, simulated read is very commonly used functions, because many sensors are analog value output, we read that the simulated values after the appropriate algorithm processing, can be applied to the function we need to realize.

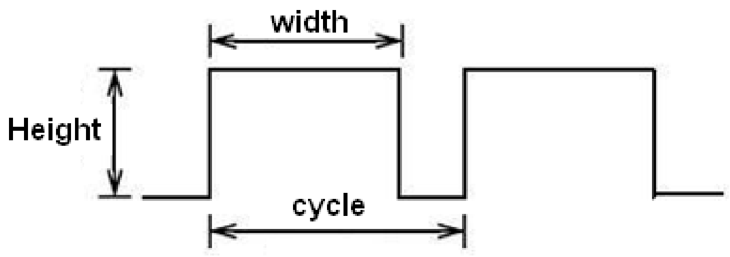
##

PWM regulates the brightness of the LED:

Pulse width modulation (PWM) is a kind of digital coding method to analog signal level, because the computer can not output analog voltage, can output 0 or 5V digital voltage value, we can make code for a specific level of analog signal through the use of high resolution counter, using the duty cycle of the square wave modulation. PWM signal is still of digital, because at any moment, the DC power supply of full magnitude is 5V (ON), or 0V (OFF). Voltage or current source is added to the simulated load by a pass (ON) or off (OFF) the repeated pulse sequence. That's when pass (ON), the DC power supply is added to the load, which is broken; when off, the power supply is disconnected. As long as enough bandwidth, any analog value can be encoded using the PWM. The output voltage value is calculated by the on and the off time. The output voltage = (connect time/ pulse time) * maximum voltage value



PWM is used in many places, lighting lamps, motor speed, sound production etc.
Here are the three basic parameters of PWM:



1. The amplitude of pulse width (minimum / maximum)
2. The pulse period (reciprocal of pulse times in 1 second)

3. The voltage level (such as：0V-5V)

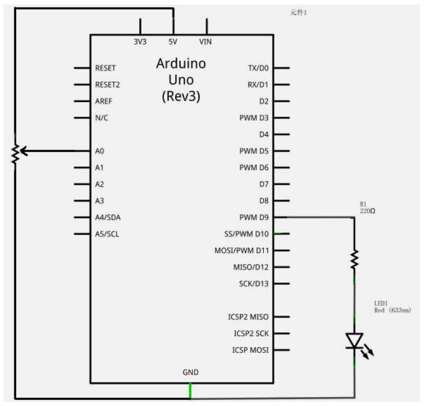
The Arduino controller has 6 PWM interface are respectively 3, 5, 6, 9, 10, 11digital interface. We have done the button control light experiment. it is the digital signal control digital interface experiment. We did the potentiometer experiment also, this time we have to complete an experiment potentiometer control light.

The components are required：

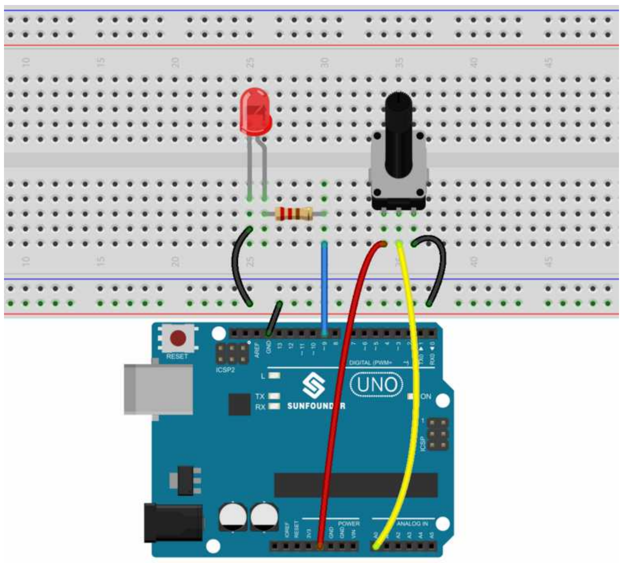
potentiometer module*1
red LED 5mm*1
220Ω direct insert resistor
Breadboard*1
Breadboard jumper lines*1 lot

Potentiometer is for analog value input to analog interface, light we connect to the PWM interface, which produce different PWM signal can make the lights change brightness difference.

We connect the physical map according to the diagram below first.



Below is a physical connection diagram, connection mode is not fixed,you can choose the simplest way to connect.

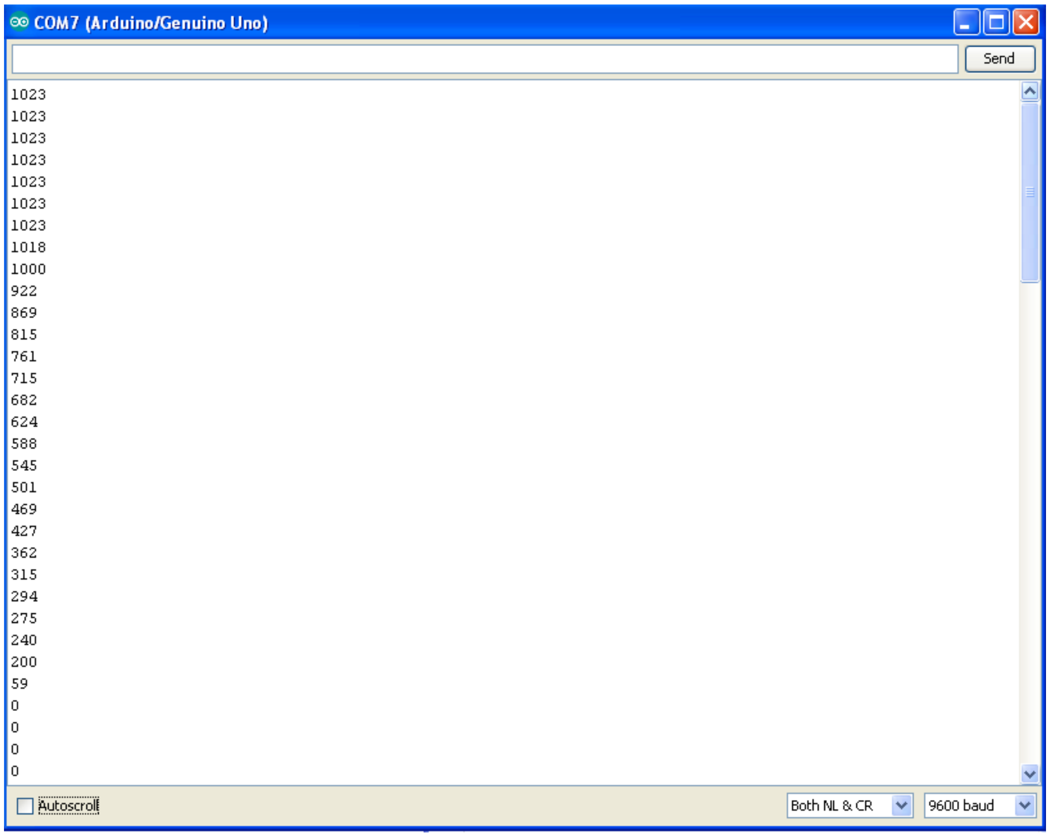


In the process of writing programs, we use Analog write analogWrite (PWM interface, analog value) function, for the Analog write to the analogWrite () function, this function usage is also very simple, we read potentiometer analog value signal in the experiment and assign it to the PWM interface, correspondingly produce light brightness changes, and then display the analog value on the screen. We can understand this program assign more the analog value to the PWM interface part in the experimental program that we read analog value, we provide a reference source below.

Reference program：

```
int potpin=0;                    //Set Pot pin to A0
int ledpin=11;                   //set Led pin to 9
int val=0;                       // temporarily store the variable values from the sensor

void setup()
{
pinMode(ledpin,OUTPUT);          //set LED pin to output
Serial.begin(9600);              // open and set the baud rate   9600
                                 // Note: The analog interface is automatically set to input
}

void loop()
{
val=analogRead(potpin);          // Read the analog value of the pot and assign it to val.

Serial.println(val);             // Show val variable in serial monitor
analogWrite(ledpin,val/4);       // Turn on the LED and set the brightness (PWM output = val. Max = 255
delay(10);//wait 0.01 of a second
}
```

Upload the sketch and open the the Serial Monitor under the Tab TOOLS and see as you adjust the pot the LED will change brightness and the serial monitor value will also change.



```
1023
1023
1023
1023
1023
1023
1023
1018
1000
922
869
815
761
715
682
624
588
545
501
469
427
362
315
294
275
240
200
59
0
0
0
0
```

##

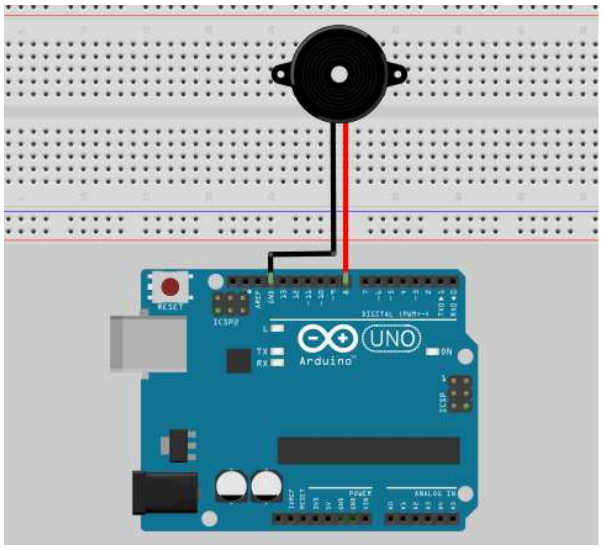
Buzzer sounds:

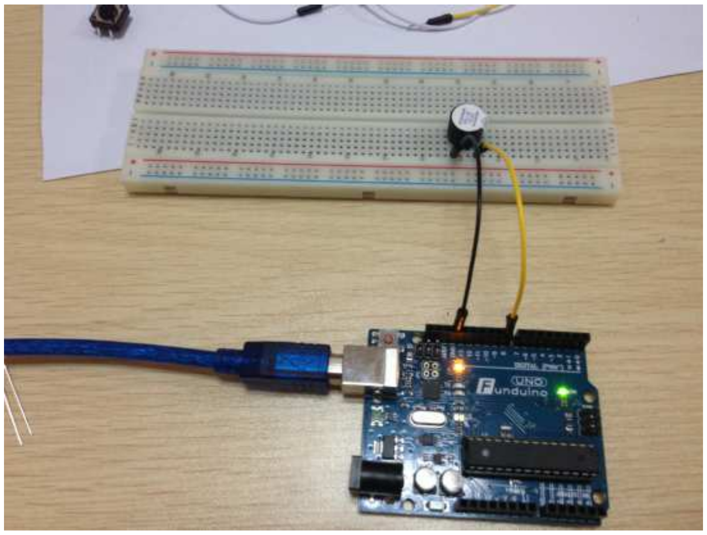
There are many interactive works with Arduino can be completed, the most common is the most commonly used sound and light show, we has been in use LED lights in the experiment before, the experiment will let everybody circuit make a sound, the most common components which can make voice is a buzzer and a horn, by contrast, the buzzer is simpler easy to use and so it is used in this experiment.

The component for use：

Buzzer*1--------------8pins connect the mark'+'
Key *1
Breadboard*1
Breadboard jumper lines*1 lot

Connect circuit as the diagram below:





When connecting the circuits, you should note that the buzzer has positive and negative points, can be seen in the connection diagram of physical projects that the buzzer has two connection wire, red and black This circuit connect procedure is very simple, and the front button control lamp is similar experimental procedures, because the control interface buzzer is also that the digital interface output high level can control buzzer.

Reference program :

```
int buzzer=8;                      //set pin8 for buzzer
void setup()
{
  pinMode(buzzer,OUTPUT);          //set output
}

void loop()
{
unsigned char i,j;                 //define variables
while(1)
 {
for(i=0;i<80;i++)                  //play a sound at a frequency
 {
      digitalWrite(buzzer,HIGH);   //buzzer ON
      delay(1);                    //wait 0.001 of a second or 1ms
      digitalWrite(buzzer,LOW);    //buzzer OFF
      delay(1);                    //wait 0.001 of a second or 1ms
    }
    for(i=0;i<100;i++)             //play a sound at a different frequency
    {
      digitalWrite(buzzer,HIGH);   //buzzer ON
      delay(2);                    //wait 0.002 of a second or 2ms
      digitalWrite(buzzer,LOW);    //buzzer OFF
      delay(2);                    //wait 0.002 of a second or 2ms
    }
  }
}
```

After uploading the program, the Buzzer experiment is finished.
Phenomenon: The buzzer keeps ringing.

##

Light to sound:

This experiment is very simple, but very useful, is also very interesting. This circuit is very common, I hope readers will remember this usage, infer other things from one fact, this is the goal. According to the light intensity, control buzzer sound frequency, sound intensity bigger, more rapid.

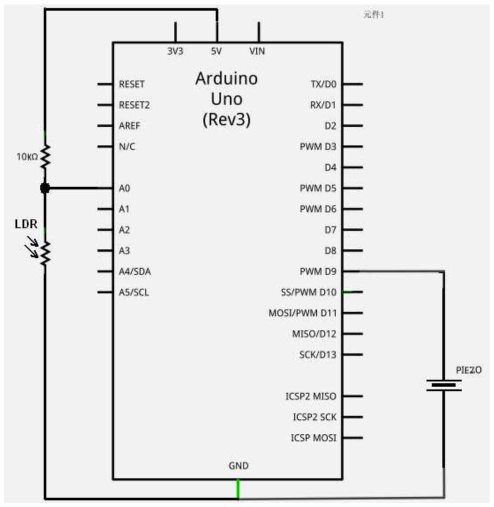
The effect is obvious.

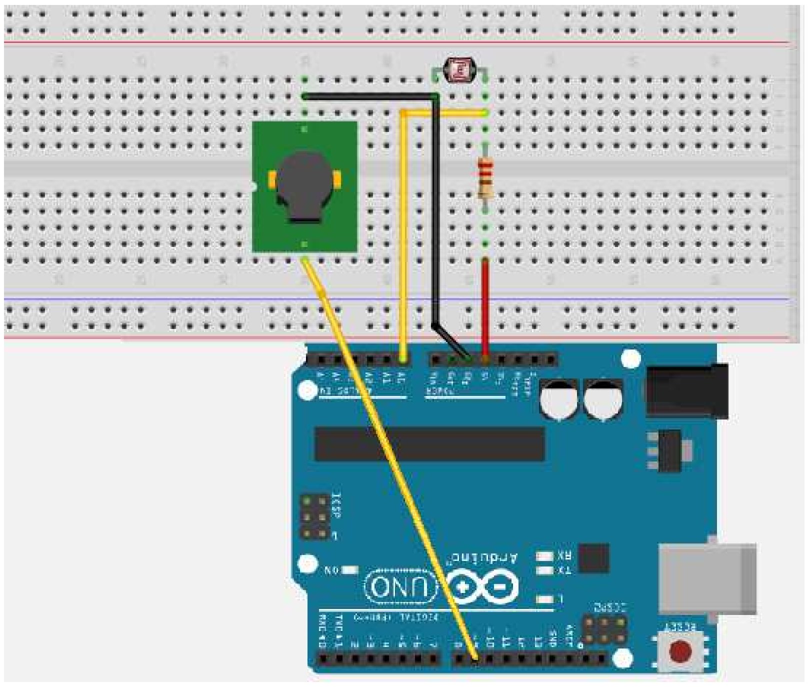
1. Experiment devices:

- photo resistor x1
- buzzer x1

Photo resistor

Physical circuit connection map





The experimental principle

Applying the methods of reading analog port voltage value, the photo resistor is directly connected to the analog port, read analog value out, to control buzzer sound frequency with the analog value, the stronger the light, the higher frequency of the buzzer

Reference program :

```
int buzzer = 9; //set buzzer to pin 9
int LDR_guangming = 0;                              // Define the input of the LDR
int val;

void setup()
{
pinMode(buzzer,OUTPUT);          //Set output
}
void alarm_out(int del)          // Sound frequency control function
{
delay(del);                      // Changing the frequency by changing the delay is very simple
digitalWrite(buzzer,HIGH);
  delay(del);
  digitalWrite(buzzer,LOW);
}

void loop()
{
val=analogRead(LDR_guangming);    // Read the value of analog interface 0 and assign it to val
    if(val<700)
      {
        alarm_out(val);          // Pass the read val value to the frequency control function
      }
}
```

The experiment phenomenon: After the uploading to the experiment board, you can use a flashlight or other light source to illuminate photo resistor, you will hear light buzzer sound frequency change when it is lit.

##

Light Controlled LED:

Complete all kinds of experiments above, we have some knowledge and understanding to use the Arduino, after mastering input and output digital quantity, input analog quantity and the production of PWM, we can start using some sensors appliance.

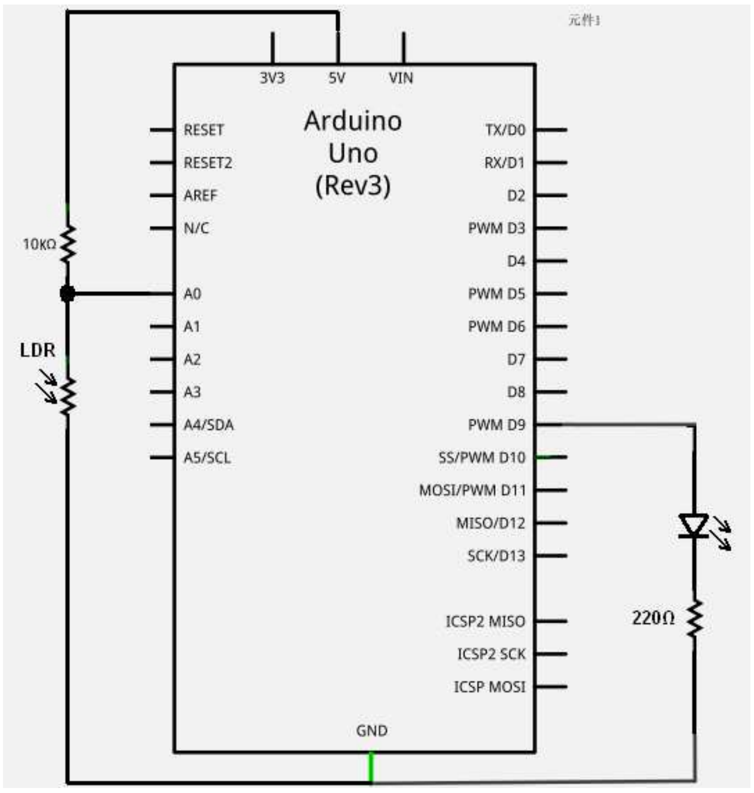
Photo resistor is a resistor using semiconductor photoelectric effect form resistor value changed by incident light intensity; incident light intensity is stronger, resistance decreases, the incident light is weak, resistance increases. Photo resistors are generally used for light measurement, light control and light conversion (the light changes turn into electrical changes).

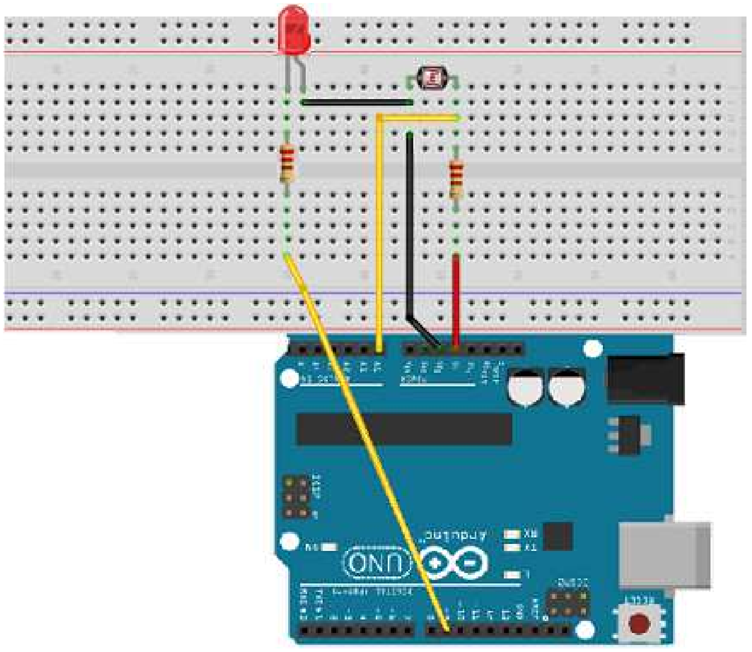
Photo resistor and can be widely applied to various light control circuit, such as lamp control and adjustment and so on, can also be used for optical switch.

In this experiment we first conduct a simple photosensitive resistance using experiment. Photo resist since it is according to the intensity change resistance element, we need to simulate read analog values, this experiment could learn from PWM interface experiment, change the potentiometer to photo resistor and LED lamp brightness will be a corresponding change when the light intensity is not the same.

Components for use:
Photo resistor*1
Red M5 lamp LED*1
10KΩlamp resistor*1
220Ωlamp resistor*1
Bread board*1
Bread board jumper lines*1 lot

Connect circuit according to the following diagram.





Connected and you can write program, the experimental procedure and experimental procedures PWM similar place is that only in the PWM value can be slightly modified according to our present circuit (modify places you can see reference).

Reference program :

```
int potpin=0;                    //Set input pin LDR
int ledpin=9;                    //set output pin
int val=0;                       // Define the variable val

void setup()
{
pinMode(ledpin,OUTPUT);          //set output
Serial.begin(9600);              //seria baud rate 9600
}

void loop()
{
val=analogRead(potpin);          // Read the analog value of the sensor and assign it to val
Serial.println(val);             // Display val variable value
  analogWrite(ledpin,val);       // Turn on LED and set brightness (PWM output max 255)
delay(10);                       //wait 0.01 of a second
}
```

Here the sensor return value is divided by 4, the reason is the analog input analogRead () function returns the value range is 0 to 1023, while the analog output analogWrite () function of the output values range from 0 to 255. Downloaded the program, we try to see the lights have the corresponding change when change the photo resistor light intensity. In daily life applications of photo resistor is very broad, usage is also a lot, we can do better interactive work According to this experiment.

##

Application experiment with relay:

The relay is an automatic control device when input value reaches a certain value (electric, magnetic, acoustic, optical, heat), output value will change. We often need to use the weak control strong in life, that is often said that

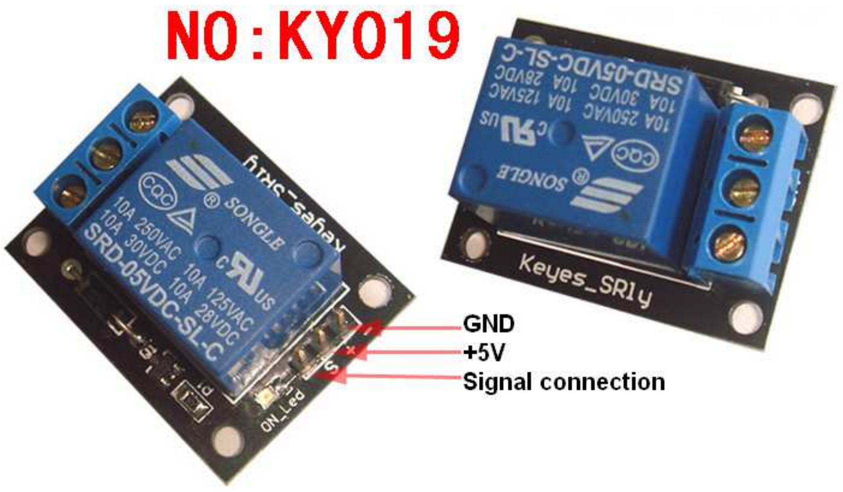
High current to control the small current, like electrical power control the fan by using Arduino controller we can use the relay,

For beginners, for safety reasons, in this experiment, we will not use electrical power, we are here using LED lights to complete the demonstration experiment.

Components for use:

Relay module*1

jumper lines*1 lot



1, the new 5V relay module

2, can be used as the MCU development board module, can be used as a home appliance control

3, the TTL control signal

4, control of DC or AC signal, can control the 220V AC load.

5, there is a normally open and a normally closed contact

6, the power indicator light

7, control lamp, attracting bright, disconnect not bright

8, transistor driver increases relay coil, control foot high impedance.

9, control foot has a pull-down circuit, to prevent false relay triggering.

Relay which belongs to the digital signal module, we control equipment through the relay digital signal making relay on and off,

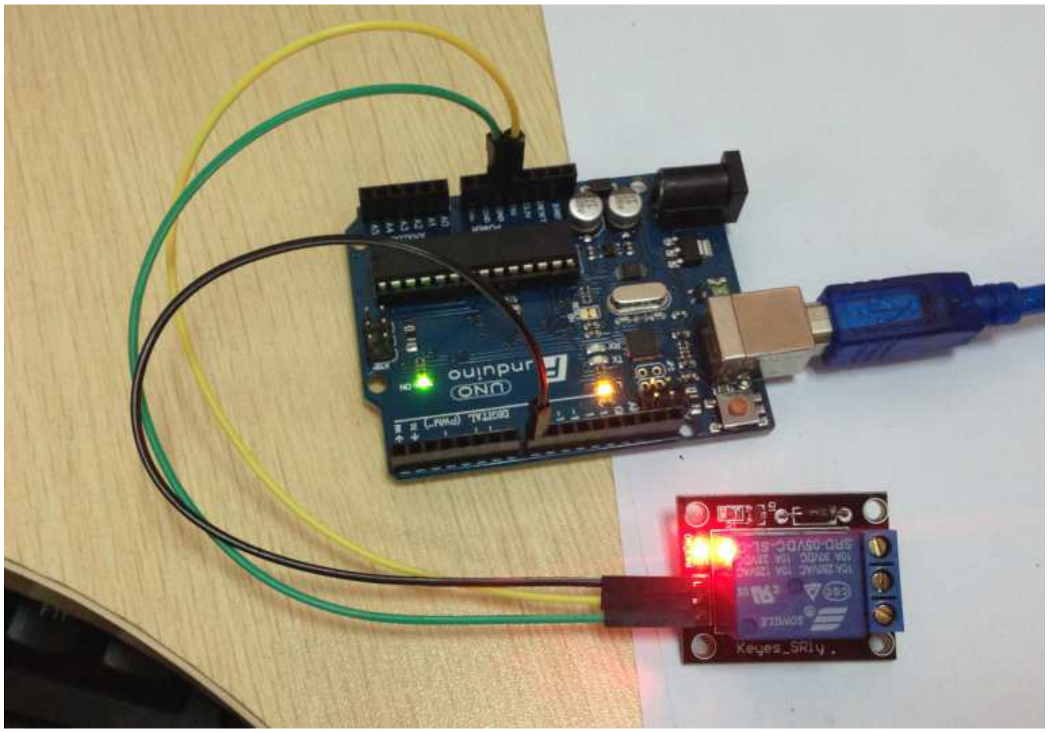
Here we use two LED lights as large power equipment, we use the Arduino controller of digital interface 8, output

High level delay after 1 seconds, the output low level for 1 seconds, is switch off for a second and then connected for a second.

Reference program :

```
int RelayPin =8;               //Set pin8 to relay
void setup()
{
pinMode(RelayPin, OUTPUT);     // Set output
}

void loop()
{
```

```
digitalWrite(RelayPin, HIGH);       //relay ON
delay(1000);                        //wait 1 second
digitalWrite(RelayPin, LOW);        //relay OFF
delay(1000);                        //wait 1 second
}
```



With relay module, we can control the 220V lamp as simple as controlling led. Upload the program we can see our red and green lights flashing in turns.

##

Fire Alarm:

1. The flame sensor “PHOTOTRANSISTOR”

The flame sensor (i.e. the infrared receiver) is used to search for the heat source for robot. The sensor is especially sensitive to the flame.

2. Working principle
Flame sensor utilize the feature that infrared is very sensitive to the flame, using flame receiving tube to detect the flame, and then changes flame brightness into the signal level, input to the central processor, the central processor to make programs according to the signal change.

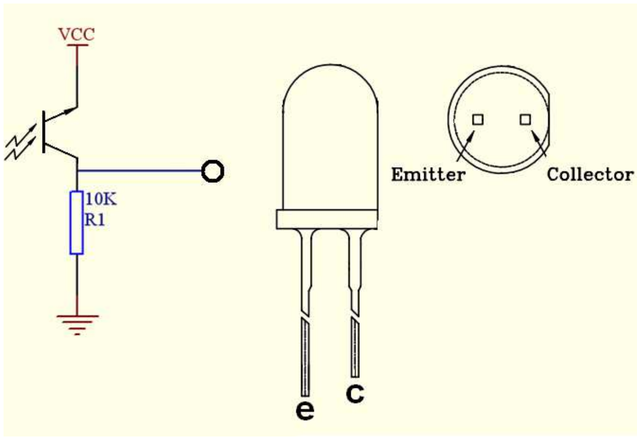
3. The flame sensor connection
Short lead of the infrared receiving transistor is negative, long lead is positive. According to the chart below, the emitter is connected to the 5V supply. The collector connected with 10K resistor and is also the point for the output, the other side of the resistor connects to GND.

2. Fire alarm test

1. Experiment devices

flame sensor x1 “phototransistor”
buzzer x1
10K resistor x1
Colorful breadboard experimental Jumper: certain quantity

2. The experimental connection

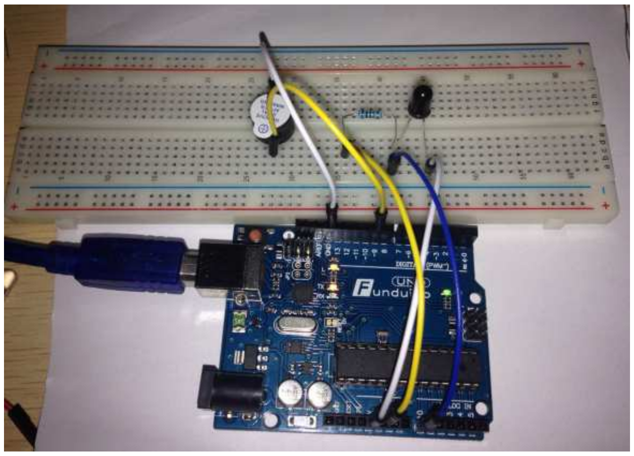


1）The connection of buzzer

Firstly, according to the Arduino curriculum connect control board, prototype board, bread board, download cable. Take out the buzzer from the experiment box, according to the laboratory buzzer connecting method of the second part. buzzer is connected to the digital 9 port. Complete the buzzer connection.

2）The connection of flame sensor

Take out the flame sensor from the experimental box, according to the wiring method of flame sensor described in this section, the flame sensor is connected to the analog A0 port. Complete the connection. The buzzer is connected to 9 pin, flame sensor is connected to analog 0 pin.



3. The experiment principle

The two conditions when it is near the flames and is not near flames, voltage values read by analog port are changing. Actually when you measure with a multimeter can get to know that when it is not near the flames analog voltage value is about 0.3V, when it is near the flames analog voltage value is about 1.0V,the closer the flames, analog voltage value is higher.

So in the start of this program, we can store an analog port voltage value i when there is no flame. Then keep repeat reading analog port voltage value j, and different value k=j-i, K is compared with 0.6V. If different value K is greater than 0.6V (Digital binary value is 123), when it is near fire, the buzzer will make sounds for alarming; if the difference value is less than 0.6V, the buzzer does not ring.

Reference program :

```
int flame=0;//set sensor to A0
  int Beep=9;//set buzzer to pin 9
  int val=0;// Define digital variables

  void setup()
{
    pinMode(Beep,OUTPUT);              //set output
  pinMode(flame,INPUT);                //set input
  Serial.begin(9600);                  //set serial at baud rate 9600
  }

void loop()
{
    val=analogRead(flame);             // Read the simulated value of the flame sensor

    Serial.println(val);               //print serial value
    if(val>=600)                       // buzzer sounds when val is greater than 600
    {
      digitalWrite(Beep,HIGH);
      }else
      {
          digitalWrite(Beep,LOW);
        }
      delay(500);
}
```

The experimental result and phenomenon:

This program can simulate alarm when it is in a flame, when it is not and everything is normal, when there is flame, alarm promptly.

###

LM35 Temperature Sensor:

LM35 is very common and easy temperature sensor element to be used, the components of the application need only one LM35 element, only use an analog interface, the difficulty lies in transit the algorithm analog value into the actual temperature.

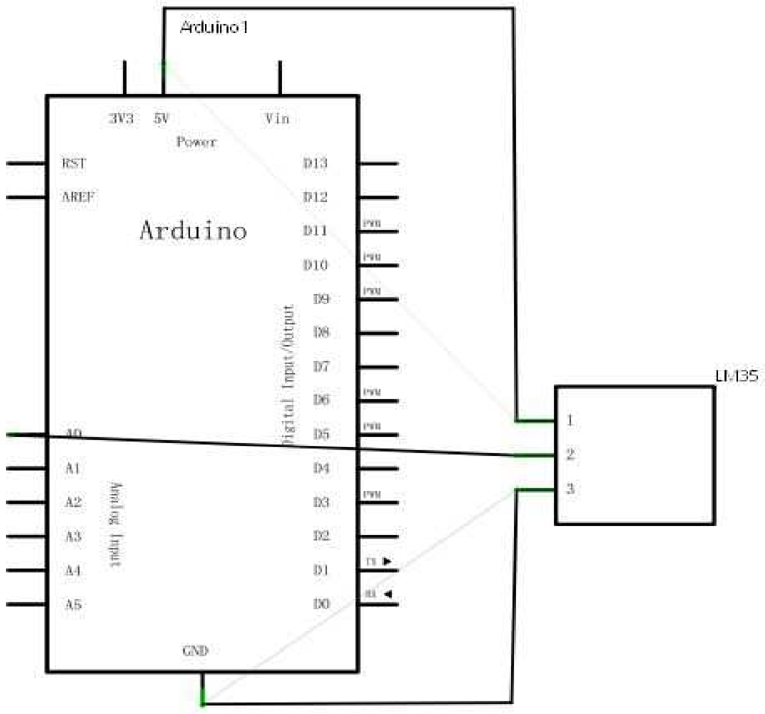
Components for use:

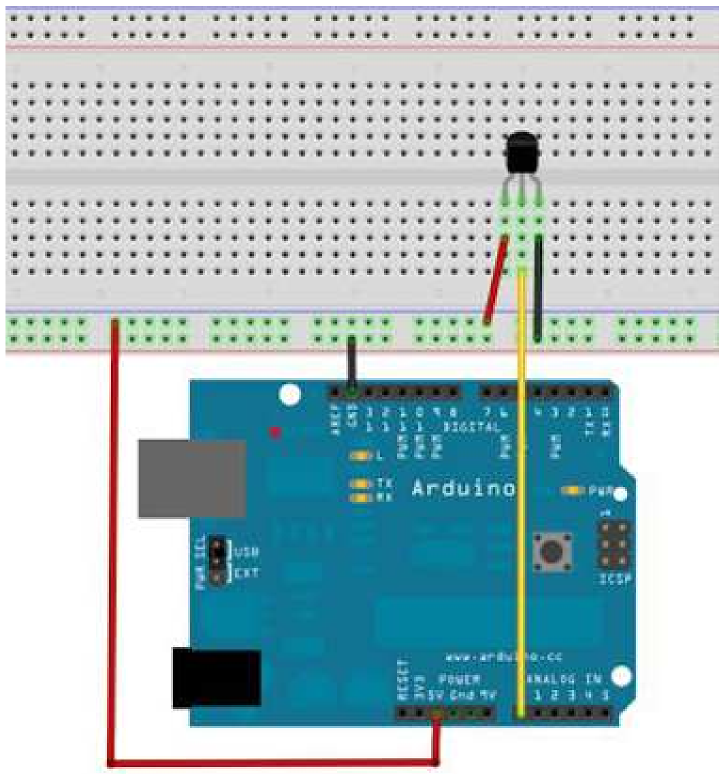
LM35*1

Breadboard*1

Breadboard jumper lines*1 lot

Connect circuit according to the following diagram:

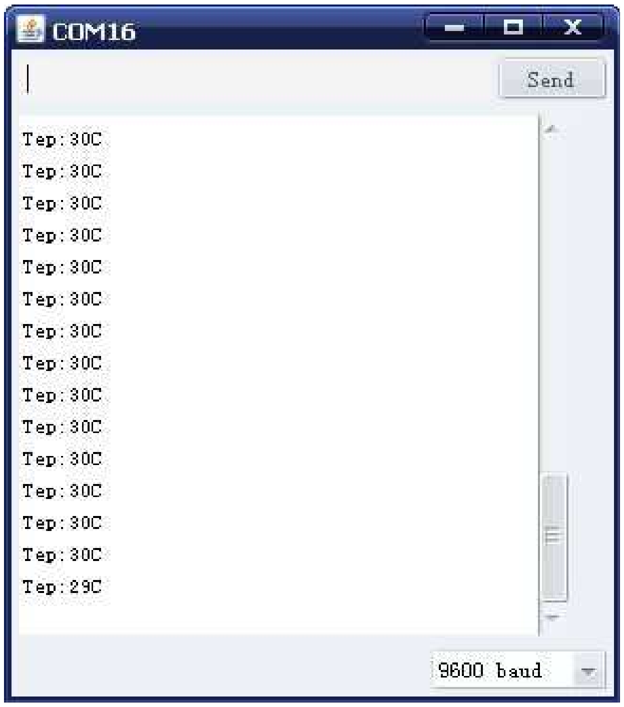




Reference program :

```
int potPin = 0;              //set Port A0 for LM35
void setup()
{
Serial.begin(9600);          //set serial baud rate 9600
}

void loop()
{
int val;                     //define veriable val
int dat;                     //define variable dat
val=analogRead(0);           //read the analog value of the sensor and assign it to val
dat=(125*val)>>8;            //temperature calculation formula for LM35
Serial.print("Tep:");        //print string “Tep: “
Serial.print(dat);           //print value dat
Serial.println("°C");        // print string “ °C“ and next reading in new line
delay(500);                  //wait ½ second
}
```

After downloading the program to open a monitoring window can see the current temperature.



```
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:30C
Tep:29C
```

##

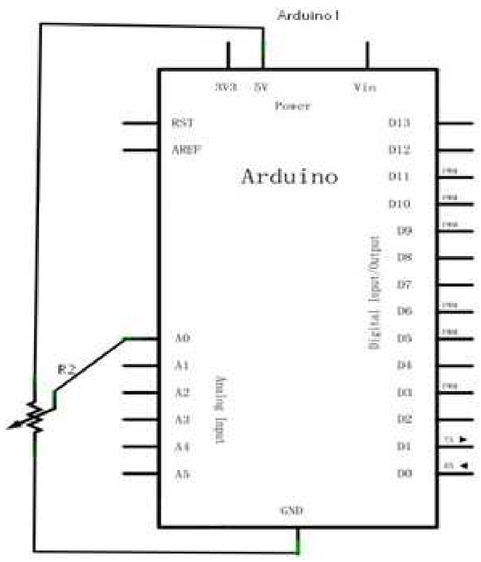
Digital Voltmeter Experiment:

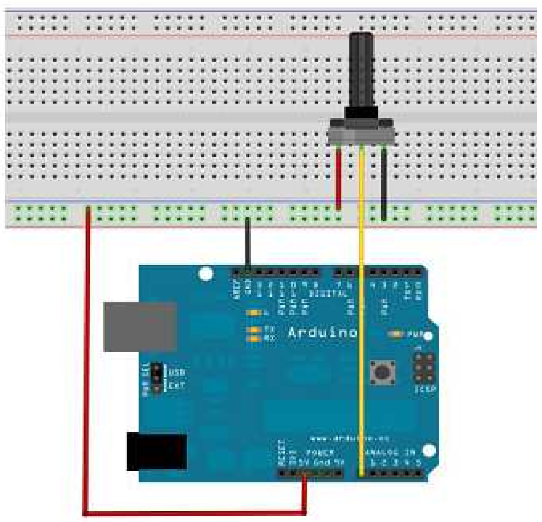
Almost keep the same with last experiment, only the data after calculation.
Components for use:

Potentiometer*1
Breadboard*1
Breadboard jumper lines*1 lot

In this experiment, we will turn potentiometer resistance into analog values read out, and then display on the screen. This is the example of the application that we are also required to master when we need to complete our own experiment functions. We must first connect

physical projects according to the following diagram





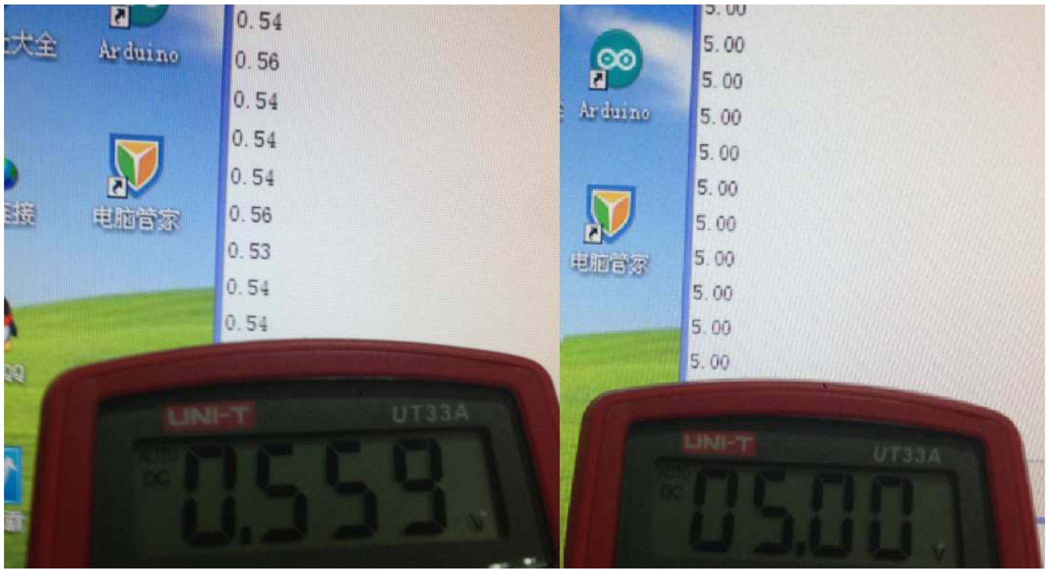
We use analog interface A0.
Program is also very simple, and more explanation is on the statement before
Here Introduction is not much to say.

Reference program:

```
int potpin=0;           //set pot to pin A0
int ledpin=13;          //set LED pin 13
int val=0;              // variable val will be defined and assigned an initial value of 0
int v;                  // variable v
```

```
void setup()
{
   pinMode(ledpin,OUTPUT);        //set output
   Serial.begin(9600);            //set serial baud rate 9600
}

void loop()
{
   digitalWrite(ledpin,HIGH);     //set LED ON
   delay(50);                     //wait 0.05 of a second
   digitalWrite(ledpin,LOW);      //set LED OFF
   delay(50);                     //wait 0.05 of a second
      val=analogRead(potpin);     // read the value of analog interface 0 and assign it to val
v=map(val,0,1023,0,500);          //Function Description map(x,Amin,Amax,Bmin,Bmax)
                                  // set analog value of 0 to 1023 and adjust to 0 to 500 in other
                                  words divide by 2.046.
Serial.println((float)v/100.00);   //print vale v
}
```

Reference program uses Arduino digital 13 built-in LED lights, read a value each, lights will flash.
Programming is in, open the serial port.
Below is the readout value simulation. And with the voltmeter to measure voltage of input port A0. There are pictures.



```
0.54
0.56
0.54
0.54
0.54
0.56
0.53
0.54
0.54
```

```
5.00
5.00
5.00
5.00
5.00
5.00
5.00
5.00
5.00
5.00
5.00
```

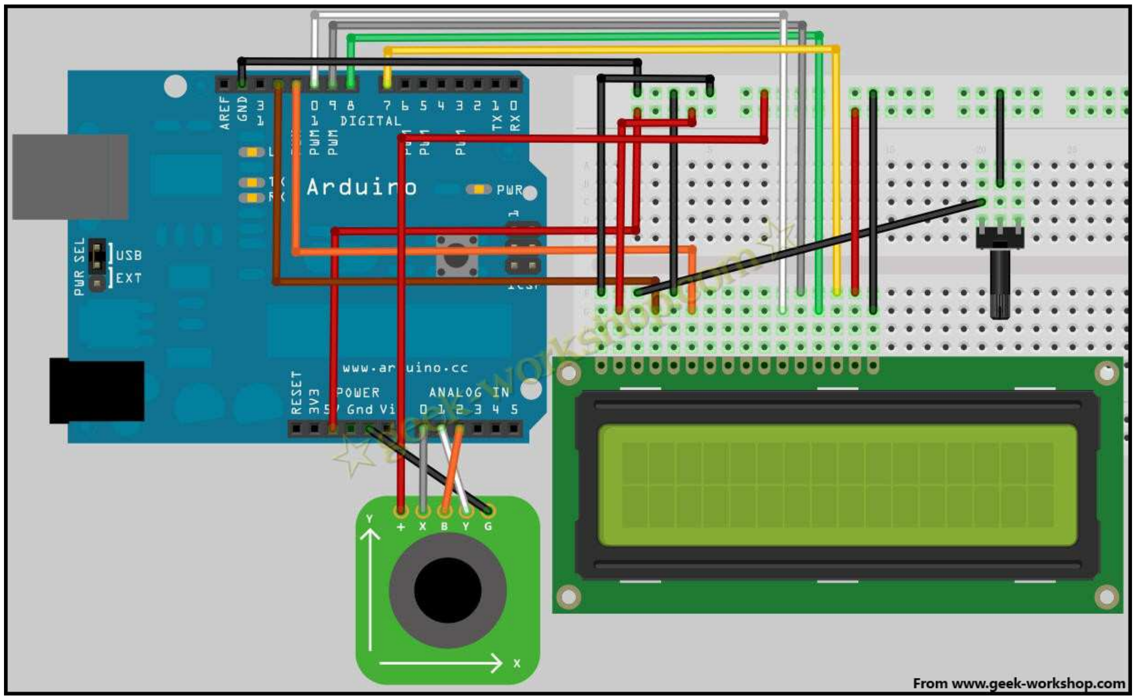
The experiment here is done, when you rotate the potentiometer knob can see changes in the numbers on the screen, according to the results, a voltmeter is quite accurate.

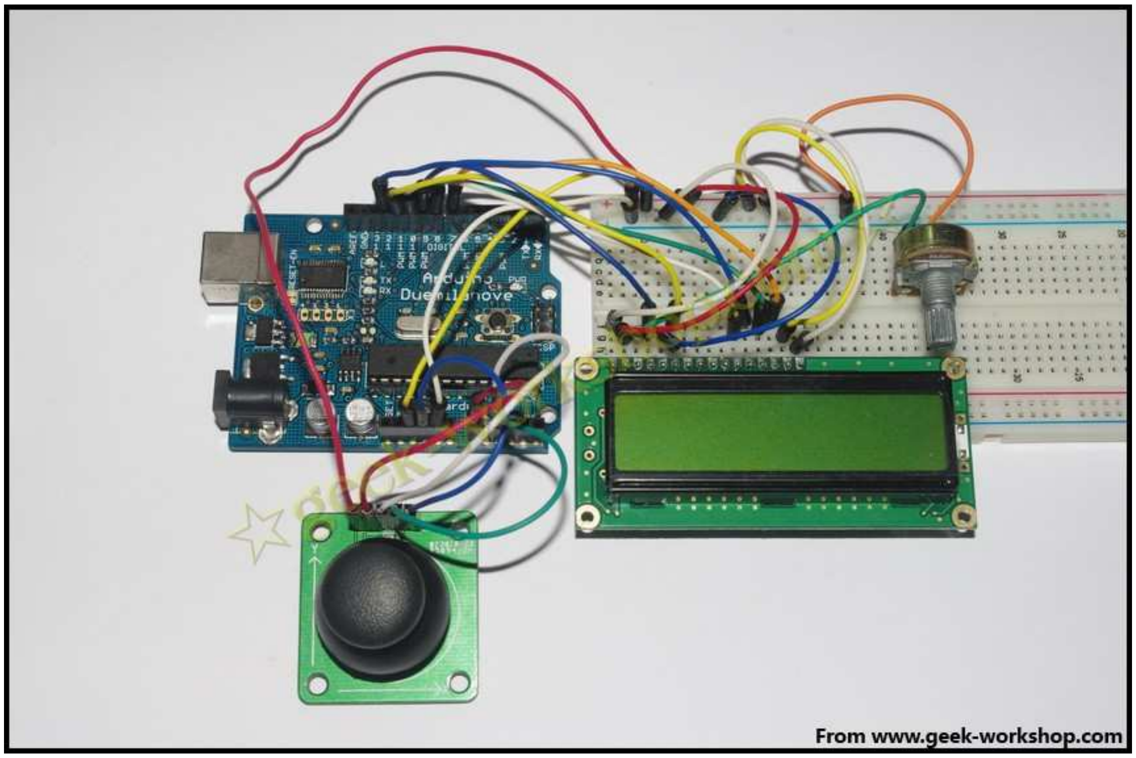
##

PS2 Joystick Nodule:

PS2 Joystick module generally can be used to control car and so on, its main structure is two 10K potentiometer and a key switch. Five ports are respectively VCC, X, Button, Y, GND.

Schematic wiring:



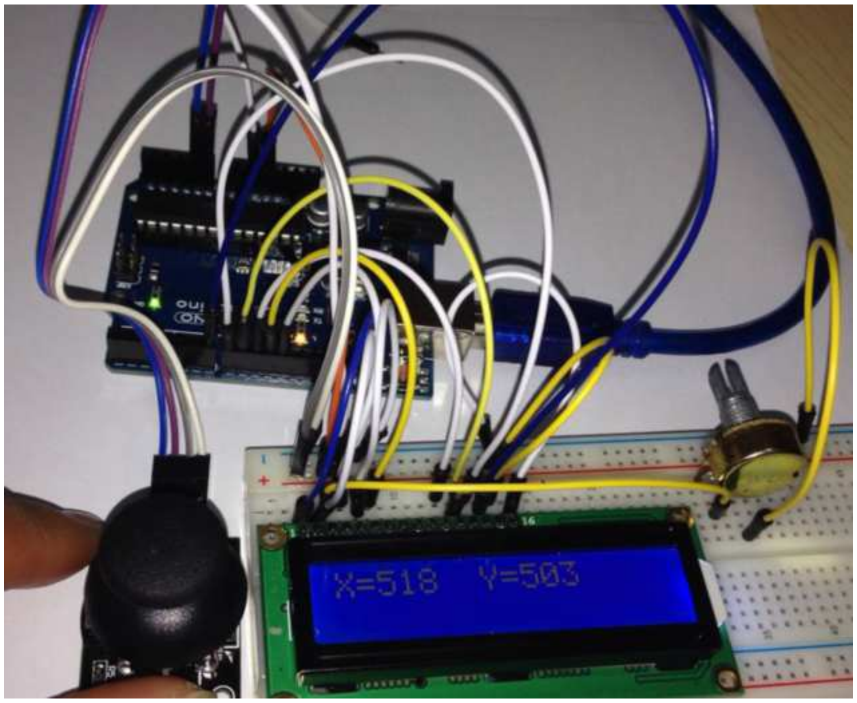


Reference program:

```
#include <LiquidCrystal.h>                // Call the LiquidCrystal
LiquidCrystal lcd(12, 11, 10, 9, 8, 7);   // Setting up the wiring
int xpotPin = 0;                          //set pot X-axis to pin A0
int ypotPin = 1;                          //set pot Y-axis to pin A1
int xval=0;                               //set veriable X
int yval=0;                               //set veriable Y
```

```
void setup()
{
   pinMode(xpotPin,INPUT);                    //set output X
   pinMode(ypotPin,INPUT);                    // set output Y
   lcd.begin(16, 2);                          // initialize LCD 16 character by 2 lines
   delay(1000);                               //wait 1 second
}

void loop ()
{
   xval = analogRead(xpotPin);                //read X value
   yval = analogRead(ypotPin);                // read Y value
   lcd.clear();                               // Clear screen
   lcd.setCursor(0, 0) ;                      // Set cursor position to the first position on the first line
   lcd.print("X=");                           // string print to screen X=
   lcd.print(xval);                           // print X value
   lcd.setCursor(7, 0) ;                      // Set cursor position to the 8th position of the first line
   lcd.print("Y=");                           // string print to screen Y=
   lcd.print(yval);                           // print X value
   delay(100);                                // wait 0.1 of a second
}
```

Results: push the Joystick, the resistance value of the potentiometer is changed, the corresponding voltage will change. The digital on LCD screen is a voltage digital output of each shaft.
Here is the test result map:



##

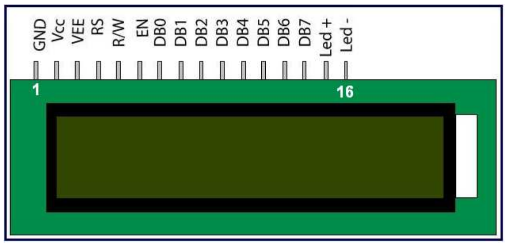
Liquid Crystal Display:

This test uses arduino to directly drive 1602 LCD text
The 1602 liquid crystal is very extensive in the application, the 1602 liquid crystal used originally is the HD44780 controller, now the 1602 module of each manufacturer basically adopts the compatible IC, so the characteristic is basically the same.

1602LCD main technical parameters

The display capacity is 16×2 characters;
The chip operating voltage is 4.5 ~ 5.5V;
Operating current is 2.0mA (5.0V);
The best working voltage of the module is 5.0V;
The character size is 2.95 x 4.35 (W x H) mm.

1602 LCD Interface Pin Definitions



Pin N°	Symbol	Level	Description	Arduino Pin	generally not used
1	Vss	0V	Ground	+5V	
2	Vdd	+5.0V	Supply Voltage	GND	
3	VO	variable	Adjustable for contrast	resistor	
4	RS	H/L	H: Data / L: Instruction code	12	
5	R/W	H/L	H: Read / L: Write	GND	**X**
6	E	H,H>L	Enable signal	11	
7	DB0	H/L	Data bus line	**X**	**X**
8	DB1	H/L	Data bus line	**X**	**X**
9	DB2	H/L	Data bus line	**X**	**X**
10	DB3	H/L	Data bus line	**X**	**X**
11	DB4	H/L	Data bus line	D5	
12	DB5	H/L	Data bus line	D4	
13	DB6	H/L	Data bus line	D3	
14	DB7	H/L	Data bus line	D2	
15	BLA	+5.0V	Anode of LED backlight	+5V	
16	BLK	0V	Cathode of LED backlight	GND	

Interface Description:

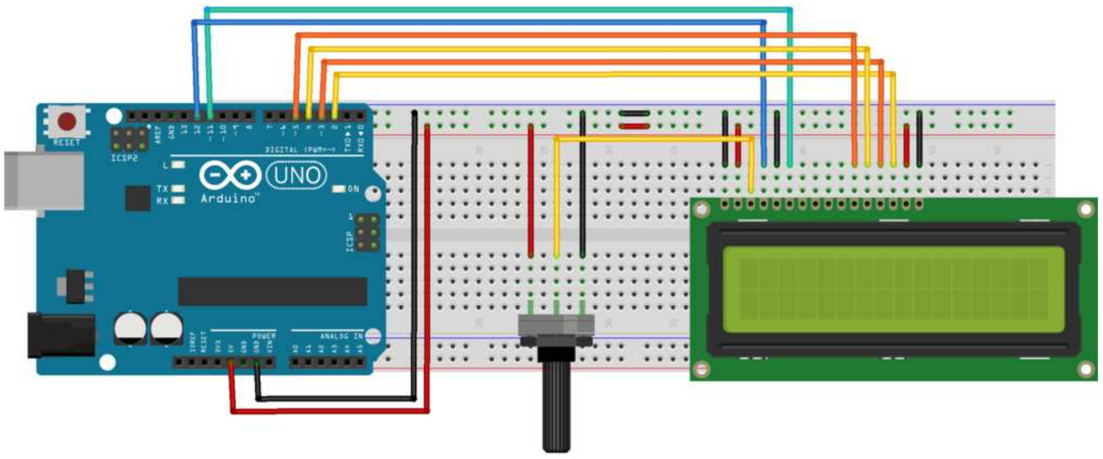
1. Two sets of power supply One set is the power supply of the module One set is the power supply of the backlight board Normally all use 5V power supply. The test backlight can also work with 3.3V power supply.
2. V0 is the pin to adjust the contrast, and the potentiometer with no more than 10KΩ pot is used for adjustment and set the contrast.
3. RS is a pin on many LCDs. It is a command/data selection pin. When the pin is high, data operation is performed; when it is low, command operation is performed.
4. RW is a read/write selection terminal. The pin level is high to indicate that the LCD is to be read; when it is low, it indicates that a write operation is to be performed. Generally not used and tied to GND.
5. E. Enable after the signal on the bus is stable, the data is read by a positive pulse.

When the pin is high, the bus is not allowed to change.

6. D0-D7 8-bit bidirectional parallel bus used to transmit commands and data.
7. BLA is the positive supply for the backlight, and BLK is the negative supply for the backlight.

In normal use, the 8-bit connection basically fills up the Arduino's digital port. If you want to connect several sensors without a port, how can you handle it? We can use 4-bit connection.

1602 directly communicates with arduino, according to the product manual described, divided into 8-bit connection method and 4-bit connection method, we first use 8-bit connection method for experiments. Hardware connection as shown below

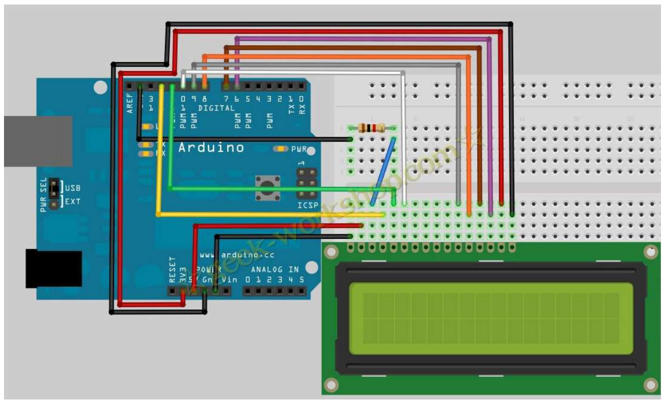


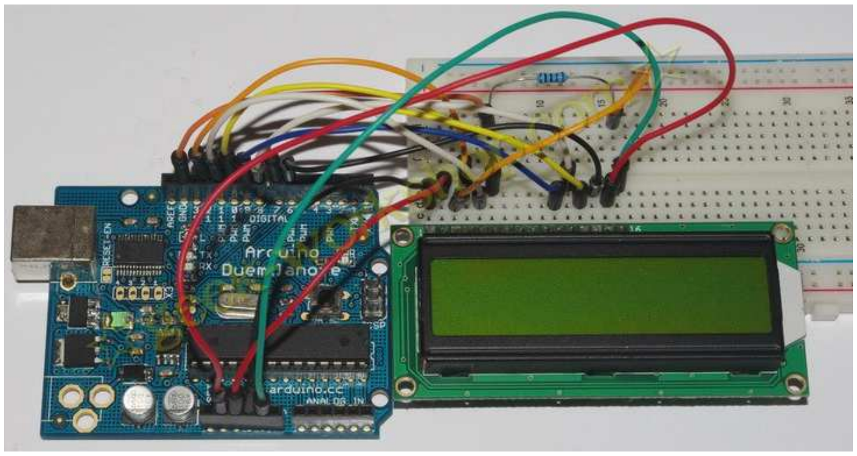
Reference program: 1

```
// http://www.arduino.cc/en/Tutorial/LiquidCrystal
  #include <LiquidCrystal.h>                // include the library
                        // initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);             // (RS, E, D4, D5, D6, D7)

void setup() {
        lcd.begin(16, 2);                         // set up the LCD's number of columns and rows:
        lcd.print("hello, world!");                // Print a message to the LCD.
}

void loop() {



    lcd.setCursor(0, 1);                          // set the cursor to column 0, line 1
                                                  // (note: line 1 is the second row, since counting
begins with 0):
    lcd.print(millis() / 1000);                   // print the number every seconds since reset:
}
```

Another 4 Bit Connection

Bit connection hardware connection method as shown below





With a different approach to the sketch will also change to the layout.

After the hardware is connected, upload the following code to the control board to see the effect.

Reference program: 2

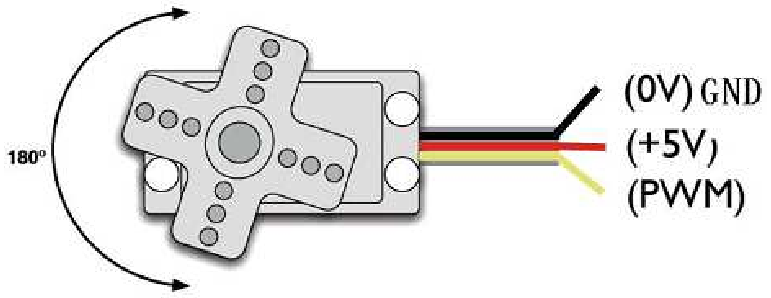
```
int LCD1602_RS=12;                    //set RS pin
int LCD1602_RW=11;                    //set R/W pin
int LCD1602_EN=10;                    //set Enable pin
int DB[] = { 6, 7, 8, 9};             //set data pins
char str1[]="Welcome to";             //set word to print
char str2[]="My LCD";                 //set word to print
char str3[]="using ";                 //set word to print
char str4[]="4-bit interface";        //set word to print

void LCD_Command_Write(int command)
{
int i,temp;
digitalWrite( LCD1602_RS,LOW);
digitalWrite( LCD1602_RW,LOW);
digitalWrite( LCD1602_EN,LOW);
temp=command & 0xf0;
for (i=DB[0]; i <= 9; i++)
{
   digitalWrite(i,temp & 0x80);
   temp <<= 1;
}
digitalWrite( LCD1602_EN,HIGH);
delayMicroseconds(1);
digitalWrite( LCD1602_EN,LOW);
temp=(command & 0x0f)<<4;
for (i=DB[0]; i <= 10; i++)
{
   digitalWrite(i,temp & 0x80);
   temp <<= 1;
}
digitalWrite( LCD1602_EN,HIGH);
delayMicroseconds(1);
digitalWrite( LCD1602_EN,LOW);
}

void LCD_Data_Write(int dat)
{
int i=0,temp;
digitalWrite( LCD1602_RS,HIGH);
digitalWrite( LCD1602_RW,LOW);
digitalWrite( LCD1602_EN,LOW);
temp=dat & 0xf0;
for (i=DB[0]; i <= 9; i++)
{
   digitalWrite(i,temp & 0x80);
   temp <<= 1;
}
digitalWrite( LCD1602_EN,HIGH);
delayMicroseconds(1);
digitalWrite( LCD1602_EN,LOW);

temp=(dat & 0x0f)<<4;
for (i=DB[0]; i <= 10; i++)
{
   digitalWrite(i,temp & 0x80);
   temp <<= 1;
}
digitalWrite( LCD1602_EN,HIGH);
delayMicroseconds(1);
digitalWrite( LCD1602_EN,LOW);
}

void LCD_SET_XY( int x, int y )
{
  int address;
  if (y ==0)     address = 0x80 + x;
  else           address = 0xC0 + x;
  LCD_Command_Write(address);
}

void LCD_Write_Char( int x,int y,int dat)
{
  LCD_SET_XY( x, y );
  LCD_Data_Write(dat);
}
```

```
void LCD_Write_String(int X,int Y,char *s)
{
      LCD_SET_XY( X, Y );      // Setting address
       while (*s)                  // Write a string
      {
        LCD_Data_Write(*s);
        s ++;
      }
}

void setup (void)
{
  int i = 0;
  for (i=6; i <= 12; i++)
    {
      pinMode(i,OUTPUT);
    }
  delay(100);
  LCD_Command_Write(0x28);                    //4 Line 2 Line 5x7
  delay(50);
  LCD_Command_Write(0x06);
  delay(50);
  LCD_Command_Write(0x0c);
  delay(50);
  LCD_Command_Write(0x80);
  delay(50);
  LCD_Command_Write(0x01);
  delay(50);
}

void loop (void)
{
    LCD_Command_Write(0x01);
    delay(50);
    LCD_Write_String(3,0,str1);// Line 1, from the 4th address
    delay(50);
    LCD_Write_String(1,1,str2);// Line 2, from the second address
    delay(5000);
    LCD_Command_Write(0x01);
    delay(50);
    LCD_Write_String(0,0,str3);
    delay(50);
    LCD_Write_String(0,1,str4);
    delay(5000);
 }

##############################################################################
```

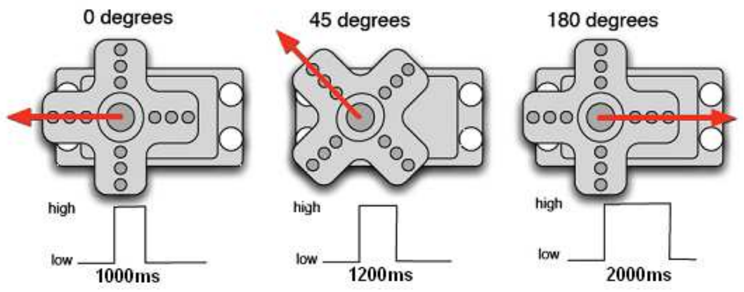
Servo Motor Control:

The servo is a kind of positioning driver mainly used by the RC hobby enthusiasts. Its working principle is to steering gear from the receiver or signal the microcontroller, the interior has a reference circuit, the cycle is 20ms, the width of the reference signal 1.5ms, compares the voltage of DC bias voltage was obtained with potentiometer, voltage difference output. Through the circuit board IC judge the direction of rotation, and then drives the non core motor starts to rotate, the power is transmitted to the swing arm through the reduction gear, at the same time by the position detector to signal, whether we've reached the positioning. Applies to those control systems need to angle changing. When the motor speed must, through the reduction gear drive cascading rotary potentiometers, the voltage difference is 0, the motor stops rotating. The rotation angle range of steering gear is 0 degrees to 180 degrees.

The servo has many specifications, but all of the actuator is connected respectively with three lines, brown, red, orange three colors to distinguish, as the servo with different brand, color will vary, brown for the ground line, red for the positive power supply line, orange for the signal line.



The rotation angle is regulated by PWM (pulse width modulation) signal duty cycle to achieve the standard, PWM (pulse width modulation) signal cycle is fixed at 20ms (50Hz), pulse width distribution theory should be between 1ms to 2ms, but, in fact the pulse width is from 0.5ms to 2.5ms, angular width and the 0 ° ~ 180 ° should be relatively. It is noteworthy places, because the brand is different, for the same signal, different brands of steering gear rotation angle will be different.



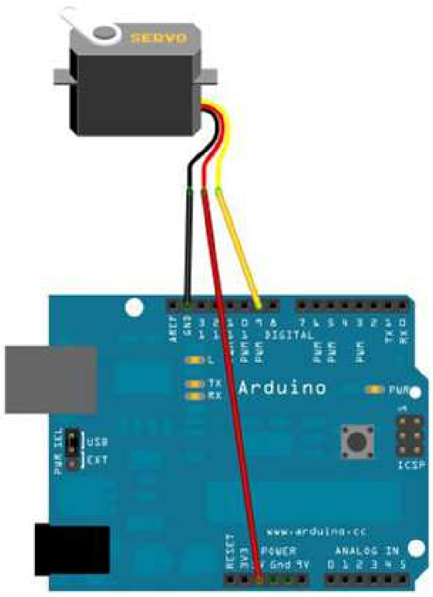
To understand the basic knowledge we can learn to control a servo, need the components need only a few actuator, one jumper is ok.

RB—412 servo*1

Breadboard jumper lines*1 lot

Method of using Arduino to control the servo has two kinds, one is the duty ratio of different square wave by ordinary digital sensor interface Arduino, simulate a PWM signal to position the servo, the second is to control the servo by directly using Servo library on Arduino, the advantage of this kind of control method is the programming, but it can only control 1 small servo because Arduino own 5V regulation is rated at 350mA. Also only the ports marked with an (~) sing can only be used as PWM outputs. The power supply ability of the Arduino is limited, so when the need control more than 1 servo, use an external power supply.

Method 1:



The servo is connected to the with interface using digital pin 9.
Write a program to let the servo rotate to the position corresponding to the angle number that user input, and will be printed to the screen angle.

Reference program A:

```
int servopin=9;                    //set servo
int myangle;                       //variable for the angle

int pulsewidth;                    //variable for the pulse width
int val;
void servopulse(int servopin,int myangle)
{
pulsewidth=(myangle*11)+500;       // control the angle to a pulse width of 500-2480
digitalWrite(servopin,HIGH);       // set the servo interface level to high
delayMicroseconds(pulsewidth);     // microseconds delayed, pulse width
digitalWrite(servopin,LOW);        //set the servo interface level to low
delay(20-pulsewidth/1000);
}

void setup()
{
pinMode(servopin,OUTPUT);          //set output
Serial.begin(9600);                //set serial baud rate 9600
Serial.println("servo=o_seral_simple ready" ) ;
}

// Convert a number from 0 to 9 to a 0 to 180 angle and let the LED blink a corresponding
number of times
```

```
void loop()
{
val=Serial.read();                      //read the value of the serial port
if(val>'0'&&val<='9')
{
val=val-'0';                            //convert feature quantities to numeric variables
val=val*(180/9);                        //turn numbers into angles
Serial.print("moving servo to ");
Serial.print(val,DEC);
Serial.println();
for(int i=0;i<=50;i++)                  //give the servo enough time to turn it to a specified angle
{
servopulse(servopin,val);               // reference pulse function
}
}
}
```

Method B:
The first a explanation of the Servo function with the Arduino, to introduce several commonly used statements about servo function.

1.attach（interface）——Set the interface, only the number 9 or 10 interface can use.

2. write（angle）——the statement used to set the servo rotation angle of, angle range can be set is 0 ° to 180 °.

3. read（ ）——The servo is used to read the statement, can be understood as the last value of order write ().

4. attached（ ）——Judge whether the servo parameters are sent to the actuator's interface.

5. detach（ ）—— Split the servo and its interface, the interface (the number 9 or 10 interface) may continue to be used as PWM interface.

Note: the above statement's writing format is the "servo variable name. specific statement ().For example, myservo.attach(9) is still connected to the number 9 interface.

Reference program B:

```
#include <Servo.h>                      //add library
Servo myservo;                          //variable name
void setup()
{
myservo.attach(9);                      //set as output on pin 9
}

void loop()
{
myservo.write(90);                      // Set the servo rotation angle
}
```

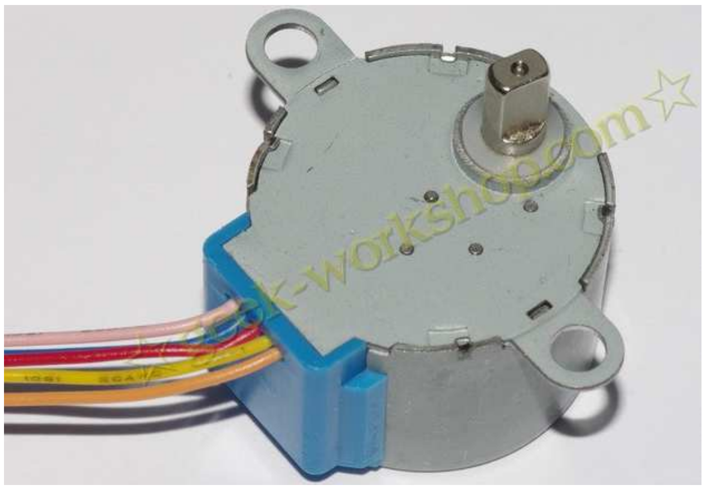
The above two methods is to control the servo, the advantages and disadvantages for each people according to their own preferences and the need to choose.

###

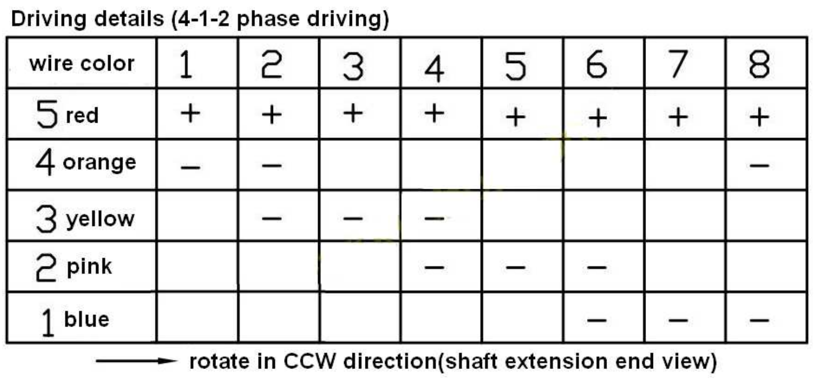
Stepper Motor:

Stepper motor is an implementing agency transforms the electricity pulse as the angular displacement. Generally speaking when stepper driver receive a pulse signal, it drives stepper motor rotates a fixed angle in the direction already set (so called the stepper angle). You can control the number of pulses to control the amount of angular displacement, so as to achieve the purpose of accurate positioning; you also can control the pulse frequency to control motor rotation speed and acceleration, so as to achieve the purpose of adapting speed.

The following is the stepper motor used in experiment



Before the use of stepper motor must carefully check the specification, confirm is the four phase or two-phase, each line to connect, in this experiment, the stepping motor is four phase (min of 5 wires), different color lines are defined as below:



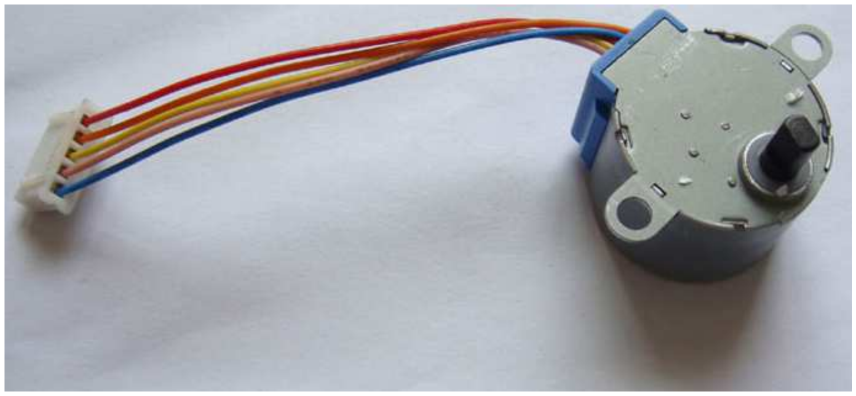
Driving details (4-1-2 phase driving)

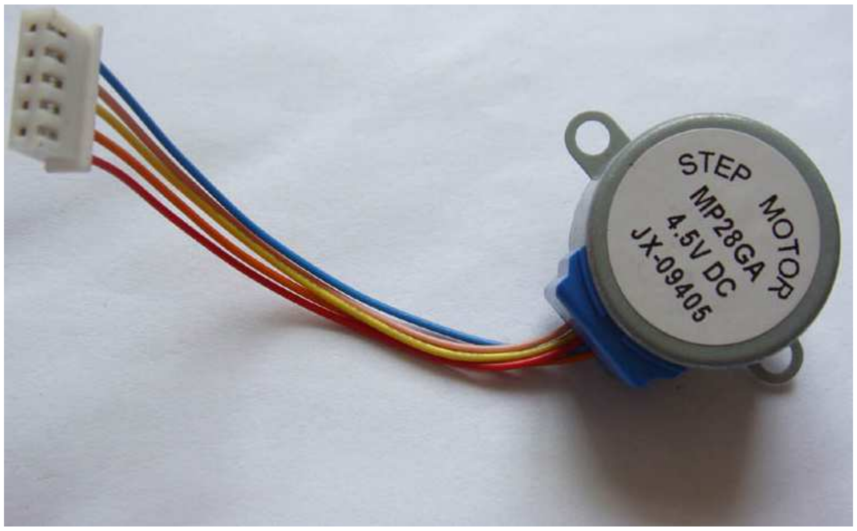
wire color	1	2	3	4	5	6	7	8
5 red	+	+	+	+	+	+	+	+
4 orange	–	–						–
3 yellow		–	–	–				
2 pink				–	–	–		
1 blue						–	–	–

$\longrightarrow$ rotate in CCW direction(shaft extension end view)

Specification: stepper motor

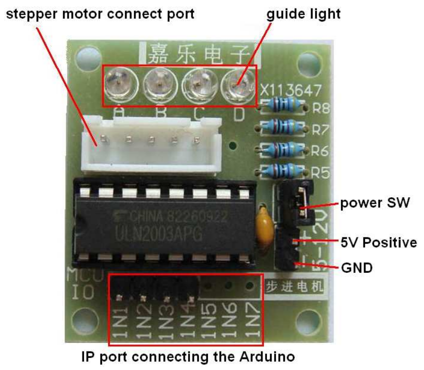
Diameter: 28mm
Voltage: 5V
Stepping angle: 5.625 x 1/64
Speed down ratio: 1/64
The 5 wire 4 phase can use ordinary ULN2003 chip driver, also can be connected 2 phase
The stepping motor no-load power consumption below 50mA, with 64 times the speed reducer, the output torque is large, can drive heavy load, very suitable for the development board to use.
Note: This stepper motor with 64 times the speed reducer, and not with the reducer of the stepper motor speed is slower, compared, for convenient observation, can be stuck with a piece of cardboard in the output shaft.





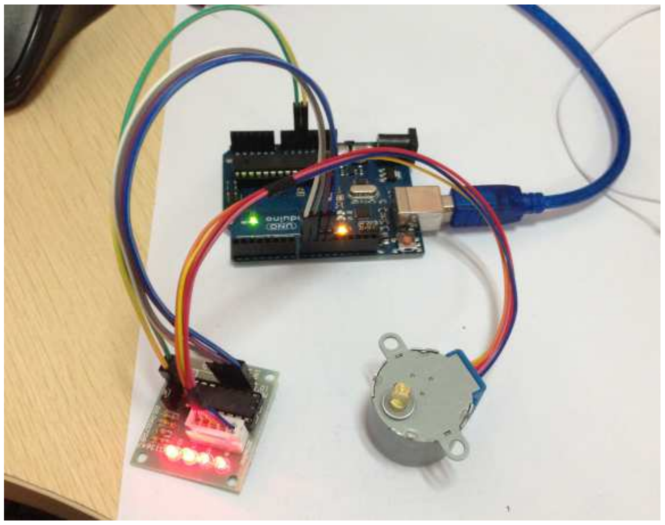
Stepper motor (five wire four phase) driver board (UL2003) test plate

Stepper motor driver board (UL2003) test module



Size：31×35mm

Hardware connection



Reference program :

```
/*
* Stepper motor follows potentiometer rotation
* (or other sensors) use analog port 0 input
* Use the Stepper.h library file that comes with the arduino IDE
*/
#include <Stepper.h>
#define STEPS 100      // Here is how many steps the stepper motor rotates for 1 revelation
Stepper stepper(STEPS, 8, 9, 10, 11);       //attached to set stepper motor pins
int previous = 0;                           // defining variables

void setup()
{
   stepper.setSpeed(90);                    // set the motor speed to 90 steps per minute
 }

 void loop()
 {
   int val = analogRead(0);                 // read analog pot value
   stepper.step(val - previous);   // The steps to moves is the current reading minus historical readings
     previous = val;                        // Save previous readings
 }
```

Experimental results: The stepper motor rotates slowly.

##

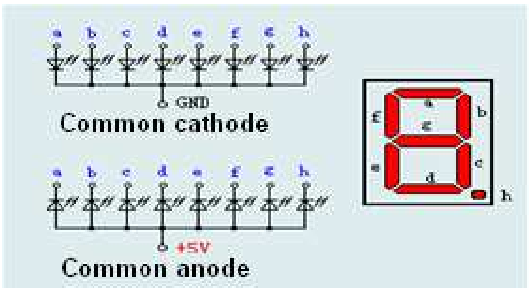
7 Segment x1 Display:

Numeric display is a common universal display device for displaying numeric, in daily life, for example: the electromagnetic oven, full automatic washing machine, solar energy water temperature display, electronic clock...It is necessary to master the numeric display display principle.

Numeric display is a semiconductor light emitting device (LED). Numeric display according to the section is divided into seven segment numeric display with a decimal point, used in this experiment is seven segment numeric display. According to the light emitting diode unit connection mode is divided into common anode numeric display and Common cathode Numeric display.

Common anode numeric display refers to all the anodes of light emitting diode are connected together forms common (COM) numeric display. Common anode numeric display with the anode should connect public pole COM to +5V in the application. When the cathode of a field of light emitting diode is in low electric level, the corresponding field will light up. When the cathode of a field is in high level, the corresponding field is not bright.

Common cathode numeric display refers to all the cathodes of light emitting diode are connected together forms common (COM) numeric display. Common anode Numeric display with the anode should connect public pole COM to the ground GND in the application. When the anode of a field of light emitting diode is low, the corresponding field will light. When the anode of a field is in high level, the corresponding field is not light.



Each segment of LED display is composed of an emitting diode, so when using the light emitting diode, also need to connect a current limiting resistor, otherwise the current may exceed the maximum forward current of the light emitting diode and burnout. This experiment use the common cathode LED display, common cathode LED display in the application should be public pole connected to the GND, when the anode of a field of emitting light diode is in low electric level, the corresponding field on the point of extinction. When the anode of a segment of light emitting diode is low, the corresponding segment will not light. When the anode of a segment is in high level, the corresponding segment will light.

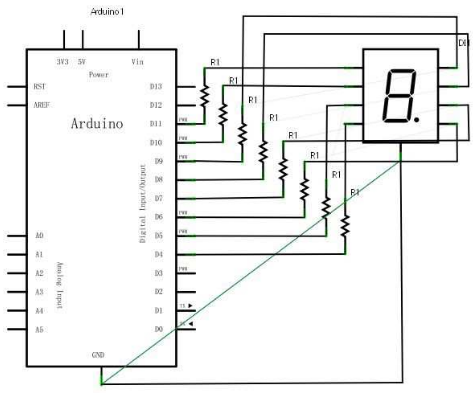
The experiment components for use

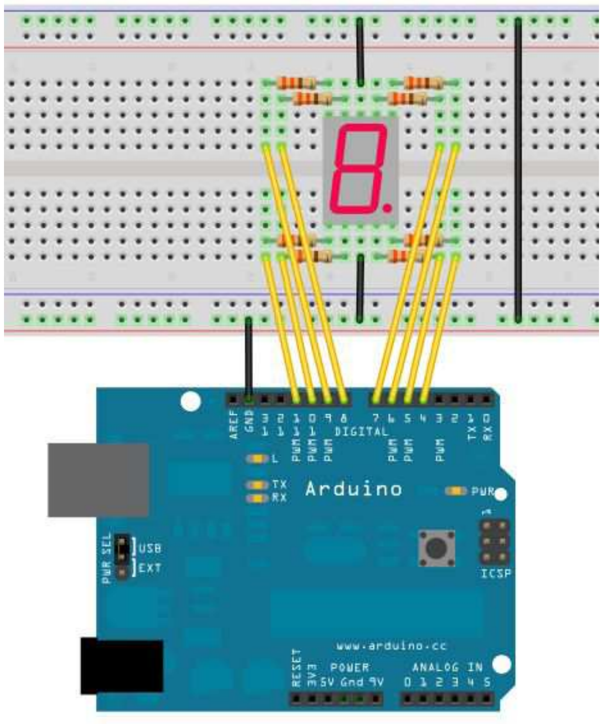
The seven segment numeric display*1

220ΩLamp resistor*8

Breadboard*1 Breadboard jumper lines*1 lot

Connection diagram to connect the circuit below.





A total of seven segment numeric display with a decimal point. When the numeric display displays numeric, as long as light the corresponding segment can be light this experiment is done. For example, let the numeric display display the number 1, then B, C segment can be light. Displays each number as in a subroutine. The main program updates every 1 sec. Adjust the int delay_set to 100. The display will run through the number every 0.1 of a second. Each segment display time is determined by the delay period, for longer delay in time adjust the delay period accordantly. For shorter delay visa versa.

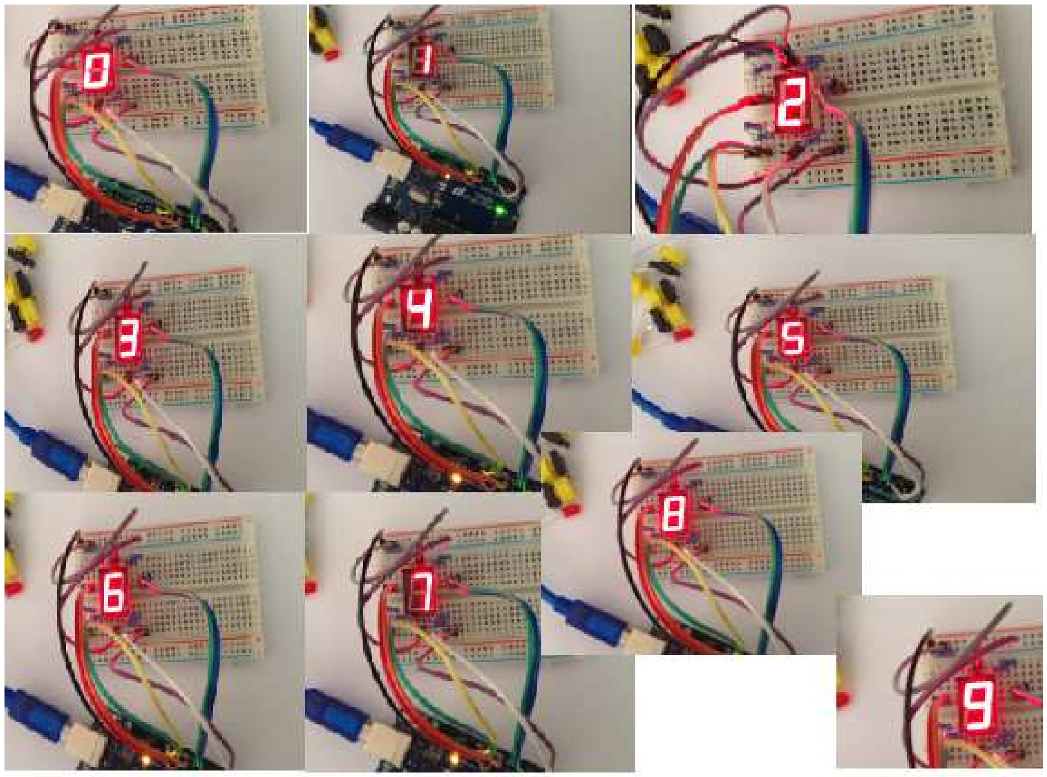
Reference program:

```
int a=7;                  //set seg. a to pin 7
int b=6;                  // set seg. b to pin 6
int c=5;                  // set seg. c to pin 5
int d=10;                 // set seg. d to pin 10
int e=11;                 // set seg. e to pin 11
int f=8;                  // set seg. f to pin 8
int g=9;                  // set seg. g to pin 9
int dp=4;                 // set seg. dp to pin 4
int delay_set=1000;       // main delay time

void digital_0(void)      //Call for the number (0)
  {
unsigned char j;
digitalWrite(a,HIGH);     //a = ON
digitalWrite(b,HIGH);     //b = ON
digitalWrite(c,HIGH);     //c = ON
digitalWrite(d,HIGH);     //d = ON
digitalWrite(e,HIGH);     //e = ON
digitalWrite(f,HIGH);     //f = ON
digitalWrite(g,LOW);      //g = OFF
digitalWrite(dp,LOW);     //dp OFF
}

void digital_1(void)      //Call for the number (1)
{
unsigned char j;
digitalWrite(c,HIGH);     //c = ON
digitalWrite(b,HIGH);     //b = ON
for(j=7;j<=11;j++)        // remaining segments OFF
digitalWrite(j,LOW);
digitalWrite(dp,LOW);     //dp OFF
}
void digital_2(void)      //Call for the number (2)
{
unsigned char j;
digitalWrite(b,HIGH);     //b = ON
digitalWrite(a,HIGH);     //a = ON
for(j=9;j<=11;j++)
digitalWrite(j,HIGH);     //j = ON
digitalWrite(dp,LOW);     //dp OFF
digitalWrite(c,LOW);      //c = OFF
digitalWrite(f,LOW);      //f = OFF
}
void digital_3(void)      //Call for the number (3)
```

```
{
digitalWrite(g,HIGH);          //g = ON
digitalWrite(a,HIGH);          //a = ON
digitalWrite(b,HIGH);          //b = ON
digitalWrite(c,HIGH);          //c = ON
digitalWrite(d,HIGH);          //d = ON
digitalWrite(dp,LOW);          //dp OFF
digitalWrite(f,LOW);           //f = OFF
digitalWrite(e,LOW);           //e = OFF
}
void digital_4(void)           //Call for the number (4)
{
digitalWrite(c,HIGH);          //c = ON
digitalWrite(b,HIGH);          //b = ON
digitalWrite(f,HIGH);          //f = ON
digitalWrite(g,HIGH);          //g = ON
digitalWrite(dp,LOW);          //dp OFF
digitalWrite(a,LOW);           //a OFF
digitalWrite(e,LOW);           //e OFF
digitalWrite(d,LOW);           //d OFF
}
void digital_5(void)           //Call for the number (5)
{
unsigned char j;
digitalWrite(a,HIGH);          //a = ON
digitalWrite(b, LOW);          //b OFF
digitalWrite(c,HIGH);          //c = ON
digitalWrite(d,HIGH);          //d = ON
digitalWrite(e, LOW);          //e OFF
digitalWrite(f,HIGH);          //f = ON
digitalWrite(g,HIGH);          //g = ON
digitalWrite(dp,LOW);          //dp OFF
}
void digital_6(void)           //Call for the number (6)
{
unsigned char j;
for(j=7;j<=11;j++)
digitalWrite(j,HIGH);          //j = ON
digitalWrite(c,HIGH);          //c = ON
digitalWrite(dp,LOW);          //dp OFF
digitalWrite(b,LOW);           //b OFF
}
void digital_7(void)           //Call for the number (7)
{
unsigned char j;
for(j=5;j<=7;j++)
digitalWrite(j,HIGH);          //j = ON
digitalWrite(dp,LOW);          //dp OFF
for(j=8;j<=11;j++)
digitalWrite(j,LOW);           //j OFF
}
void digital_8(void)           //Call for the number (8)
```

```
{
unsigned char j;
for(j=5;j<=11;j++)
digitalWrite(j,HIGH);           //j = ON
digitalWrite(dp,LOW);           //dp OFF
}
void digital_9(void)            //Call for the number (9)
{
unsigned char j;
digitalWrite(a,HIGH);           //a = ON
digitalWrite(b,HIGH);           //b = ON
digitalWrite(c,HIGH);           //c = ON
digitalWrite(d,HIGH);           //d = ON
digitalWrite(e, LOW);           //e OFF
digitalWrite(f,HIGH);           //f = ON
digitalWrite(g,HIGH);           //g = ON
digitalWrite(dp,LOW);           //dp OFF
}

void setup()
{
int i;                          //define variable
for(i=4;i<=11;i++)
pinMode(i,OUTPUT);              // set 4 to 11 pins for output
}

void loop()
{
while(1)
{
digital_0();                    //Display number 0
delay(delay_set);               //wait delay set time
digital_1();                    //Display number 1
delay(delay_set);               //wait delay set time
digital_2();                    //Display number 2
delay(delay_set);               //wait delay set time
digital_3();                    //Display number 3
delay(delay_set);               //wait delay set time
digital_4();                    //Display number 4
delay(delay_set);               //wait delay set time
digital_5();                    //Display number 5
delay(delay_set);               //wait delay set time
digital_6();                    //Display number 6
delay(delay_set);               //wait delay set time
digital_7();                    //Display number 7
delay(delay_set);               //wait delay set time
digital_8();                    //Display number 8
delay(delay_set);               //wait delay set time
digital_9();                    //Display number 9
delay(delay_set);               //wait delay set time
}
}
```

Experiment phenomenon　: digital display display circularly 0 to 9 digital

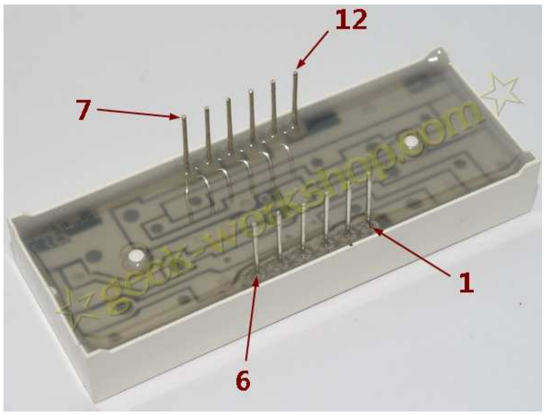


##

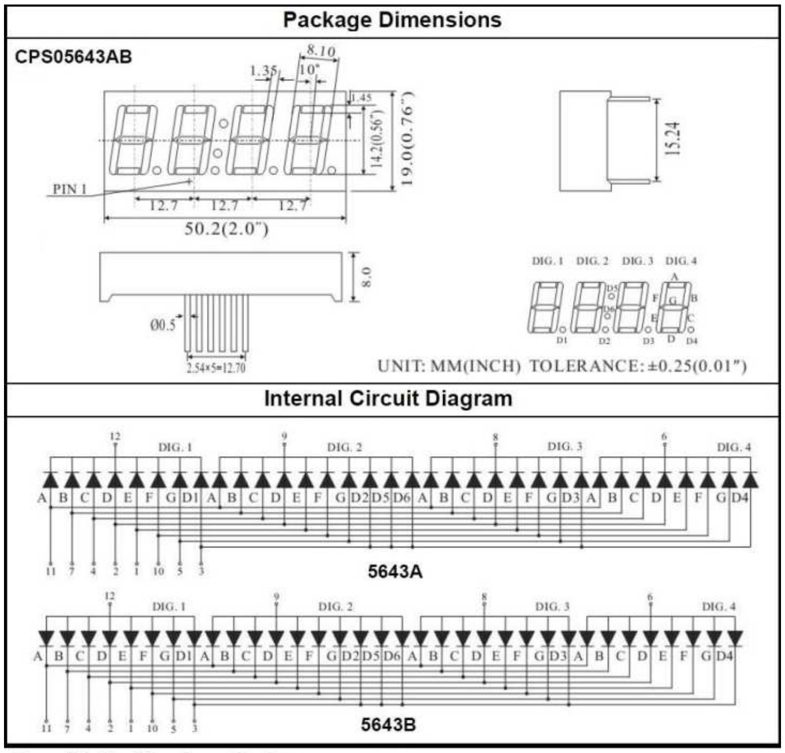
7 Segment x4 Display:

We conduct experiment using the Arduino to drive a common cathode four numeric display. Driver numeric display current limiting resistor is absolutely indispensable, a current limiting resistor of two kinds of connection, one is connecting in the D1-D4 anode, a total of 4. This method needs less resistance, but will produce different numeric brightness on each display bit, the 1 is the brightest, the darkest 8. Another method is connecting with other 8 pins, the brightness of this connection is even, but with more resistance. In this experiment use 8 220Ω resistors (because there is no 100 Ω resistor, so use 220Ω ohm instead and 100 ohm brightness is high).

4 numeric displays has a total of 12 pins, decimal point is down on the front, the lower left corner for 1, other pins to rotate in counterclockwise order. The upper left corner is the largest 12 pin.

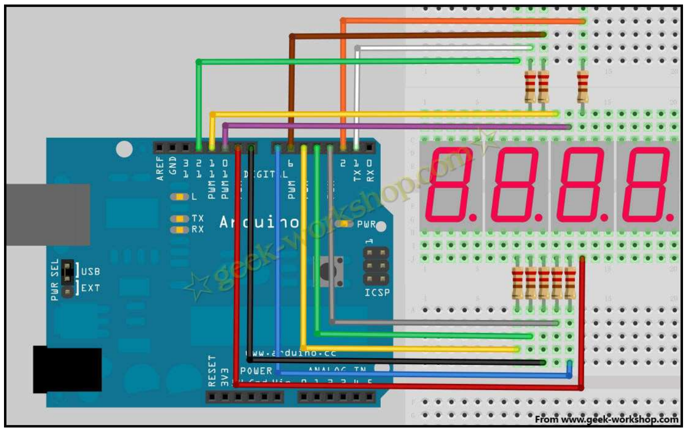


The numeric display manual



Four Digits Displays Series

Below is the hardware connection diagram



Reference program:

```
    int a = 1;                          //set up the cathode interface
    int b = 2;
    int c = 3;
    int d = 4;
    int e = 5;
    int f = 6;
    int g = 7;
    int dp = 8;
    int d4 = 9;                         //set up the anode interface
    int d3 = 10;
    int d2 = 11;
    int d1 = 12;
    long n = 1230;                      //set the variable
    int x = 100;
    int del = 55;                       //Here the value of the clock to fine-tune

    void setup()        {
      pinMode(d1, OUTPUT);
      pinMode(d2, OUTPUT);
      pinMode(d3, OUTPUT);
      pinMode(d4, OUTPUT);
      pinMode(a, OUTPUT);
      pinMode(b, OUTPUT);
      pinMode(c, OUTPUT);
      pinMode(d, OUTPUT);
      pinMode(e, OUTPUT);
      pinMode(f, OUTPUT);
      pinMode(g, OUTPUT);
      pinMode(dp, OUTPUT);
    }

void loop()    {
  Display(1, 1);
  Display(2, 2);
  Display(3, 3);
  Display(4, 4);
}
```

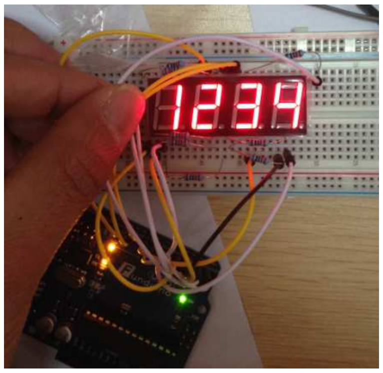
```
void WeiXuan(unsigned char n)     {
       switch(n)
        {
       case 1:
          digitalWrite(d1,LOW);
          digitalWrite(d2, HIGH);
          digitalWrite(d3, HIGH);
          digitalWrite(d4, HIGH);
         break;
         case 2:
          digitalWrite(d1, HIGH);
          digitalWrite(d2, LOW);
          digitalWrite(d3, HIGH);
          digitalWrite(d4, HIGH);
             break;
          case 3:
             digitalWrite(d1,HIGH);
           digitalWrite(d2, HIGH);
           digitalWrite(d3, LOW);
           digitalWrite(d4, HIGH);
             break;
          case 4:
           digitalWrite(d1, HIGH);
           digitalWrite(d2, HIGH);
           digitalWrite(d3, HIGH);
           digitalWrite(d4, LOW);
             break;
             default :
             digitalWrite(d1, HIGH);
           digitalWrite(d2, HIGH);
           digitalWrite(d3, HIGH);
           digitalWrite(d4, HIGH);
             break;
          }
}

void Num_0()     {
   digitalWrite(a, HIGH);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, HIGH);
   digitalWrite(e, HIGH);
   digitalWrite(f, HIGH);
   digitalWrite(g, LOW);
   digitalWrite(dp, LOW);
}

void Num_1()     {
   digitalWrite(a, LOW);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, LOW);
```

```
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);
  digitalWrite(dp, LOW);
}

void Num_2()    {
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, LOW);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  digitalWrite(dp, LOW);
}

void Num_3()    {
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  digitalWrite(dp, LOW);
}

void Num_4()    {
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
  digitalWrite(dp, LOW);
}

void Num_5()    {
  digitalWrite(a, HIGH);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
  digitalWrite(dp, LOW);
}

void Num_6()    {
  digitalWrite(a, HIGH);
```

```
   digitalWrite(b, LOW);
   digitalWrite(c, HIGH);
   digitalWrite(d, HIGH);
   digitalWrite(e, HIGH);
   digitalWrite(f, HIGH);
   digitalWrite(g, HIGH);
   digitalWrite(dp, LOW);
}

void Num_7()     {
   digitalWrite(a, HIGH);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, LOW);
   digitalWrite(e, LOW);
   digitalWrite(f, LOW);
   digitalWrite(g, LOW);
   digitalWrite(dp, LOW);
}

void Num_8()     {
   digitalWrite(a, HIGH);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, HIGH);
   digitalWrite(e, HIGH);
   digitalWrite(f, HIGH);
   digitalWrite(g, HIGH);
   digitalWrite(dp, LOW);
}

void Num_9()     {
   digitalWrite(a, HIGH);
   digitalWrite(b, HIGH);
   digitalWrite(c, HIGH);
   digitalWrite(d, HIGH);
   digitalWrite(e, LOW);
   digitalWrite(f, HIGH);
   digitalWrite(g, HIGH);
   digitalWrite(dp, LOW);
}

void Clear()     {                                              //clear screen

   digitalWrite(a, LOW);
   digitalWrite(b, LOW);
   digitalWrite(c, LOW);
   digitalWrite(d, LOW);
   digitalWrite(e, LOW);
   digitalWrite(f, LOW);
   digitalWrite(g, LOW);
   digitalWrite(dp,LOW);
```

```
}

void pickNumber(unsigned char n)     {            //selection
   switch(n)
   {
     case 0:Num_0();
     break;
     case 1:Num_1();
     break;
     case 2:Num_2();
     break;
     case 3:Num_3();
     break;
     case 4:Num_4();
     break;
     case 5:Num_5();
     break;
     case 6:Num_6();
     break;
     case 7:Num_7();
     break;
     case 8:Num_8();
     break;
     case 9:Num_9();
     break;
     default:Clear();
     break;
   }
}

void Display(unsigned char x, unsigned char Number)     {            //Display x as coordinates of Number as a digit
   WeiXuan(x);
   pickNumber(Number);
   delay(1);
   Clear() ;                                    //clear screen
}
```

Upload the following code to the control board, have a look at effect chart.



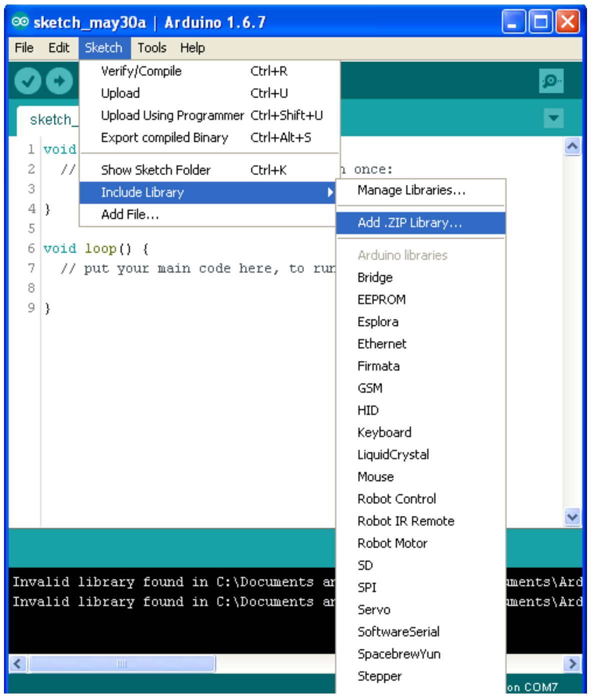
The experiment result is displaying 1234 on numeric display.

Note: you need to be patient, careful, do not connect the wrong wires, when not display correctly, first check the wiring.

##

Infrared Remote Control:

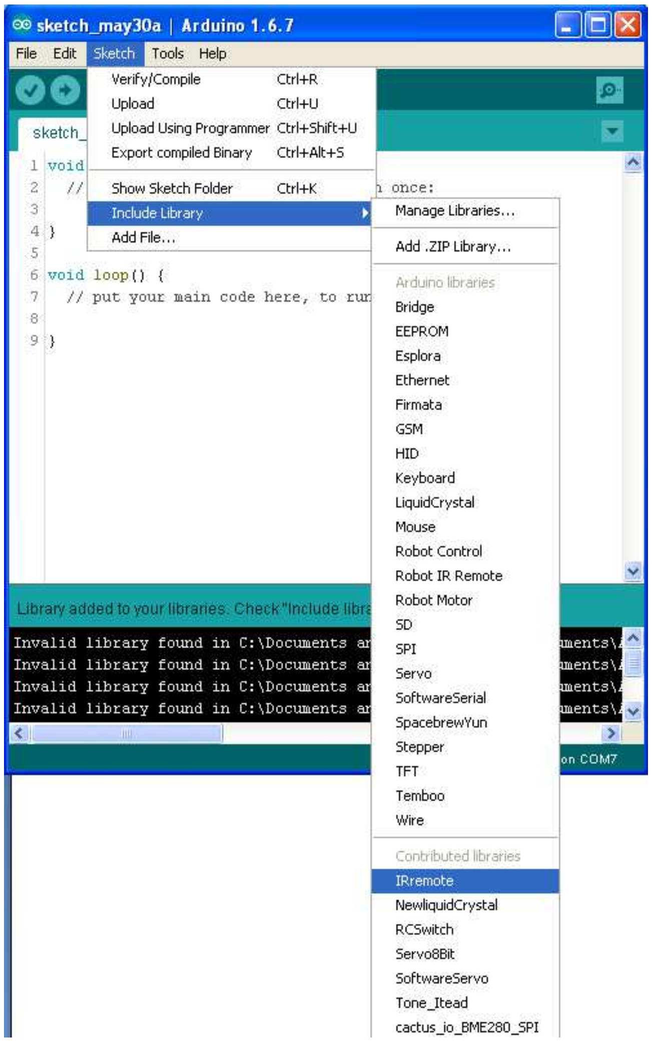
Note： Download the IRremote zip file from **https://github.com/z3t0/Arduino-IRremote/releases** and see **2.1.0 - Stable Release** and download Arduino-IRremote-dev.zip and place it in a new folder under C:\......\Arduino\libraries\New. Open the Arduino IDE and open the tabs Sketch\Include Library and click on Add.ZIP Library. Locate the zip file and click on it.



```
void
  //                                  n once:

}

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

}
```

```
Invalid library found in C:\Documents an                     uments\Ard
Invalid library found in C:\Documents an                     uments\Ard
```

Open tabs Sketch\Include Library, check down the list to see if installed.



```
void
  //                                  n once:

}

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

}
```

Library added to your libraries. Check "Include libra

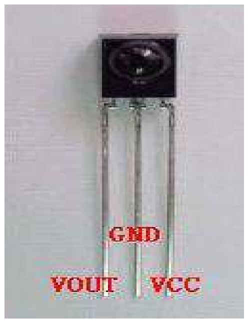
```
Invalid library found in C:\Documents an                     uments\
Invalid library found in C:\Documents an                     uments\
Invalid library found in C:\Documents an                     uments\
Invalid library found in C:\Documents an                     uments\
```

NB: If not installed the compiler will not complete and the upload will fail.

1. The infrared receiving head

Infrared remote controller sends signal is a series of binary pulse code. In order to interference from the other infrared signal in the wireless transmission process, usually the first of its modulation on the specified carrier frequency, and then by the infrared emission diode launch out, and the infrared receiving device to filtering the clutter signal and receives the specified frequency and restore it to the binary pulse code that is demodulated.
Working principal
The optical signals emitted by the built-in infrared receiving tube are converted to weak signals, the signal is amplified through the IC internal amplifier, and then through the automatic gain control, bandpass filtering, demodulation, waveform shaping transits into the original code emitted by remote controller, input the code into code recognition circuit on electric devices via the signal output pin of the receiving head.
The pin and the connection of the infrared receiving head

When Connect the VOUT to analog port, GND connects to the GND on experimental panel, VCC connects to the +5v on the experimental panel.



The infrared receiving head has 3 pins.

2. Infrared remote control experiment

1. Experiment devices

Infrared remote controller x1
Infrared receiver head x1
LED light x6
220Ωresistor x6
Multi color breadboard lines x a certain quantity.

2. Experiment connection

The board is connected; then, the infrared receiving head is connected in accordance with the above method, the VOUT connects the digital 11 port pins, the LED lamp through the resistor connects to the digital pin 2, 3, 4, 5, 6, 7. Complete the part of circuit connection.

3. The experimental principle

To decode a remote controller must know the remote controller encoded mode. Encoded mode this product uses is: NEC protocol. Here's a look at the NEC protocol:

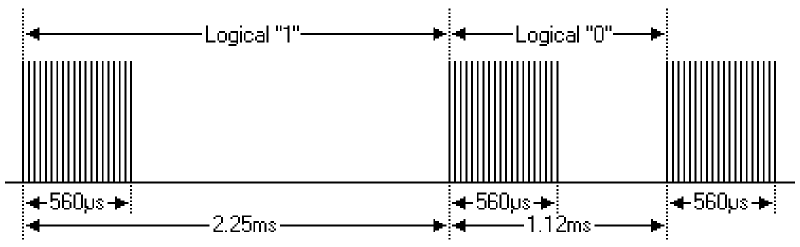
·NEC protocol:

Feature：

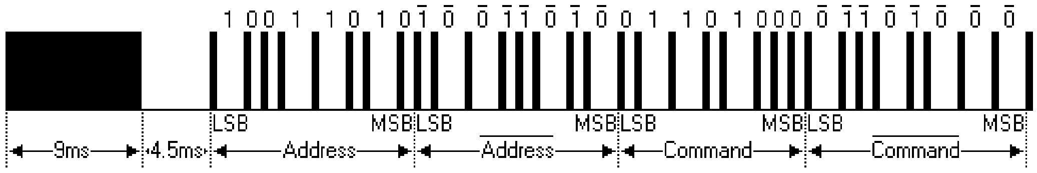
(1) 8 bit address bits, 8 bit command bit
(2) In order to reliability address bits and command bits transmitted two times
(3) Pulse position modulation
(4) The carrier frequency 38kHz
(5) The time of each bit is 1.125ms or 2.25ms

A logical 0 and 1 is defined as below

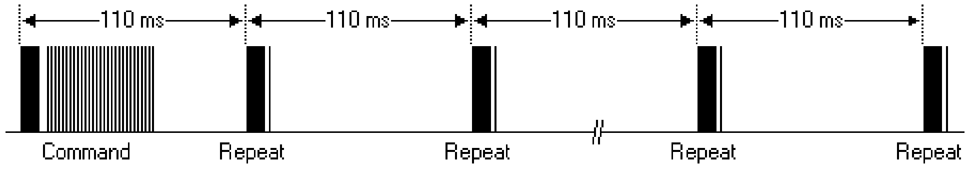
Protocol is as follows :



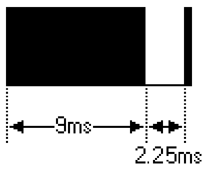
• The pulse for pressing key and release immediately:



The picture above shows the typical pulse sequence of NEC protocol. Note: this first sends LSB (lowest) protocol. On the top of that pulse transmission address is the 0x59, command is 0x16. A message start from a high level of a 9ms, followed by a low level of a 4.5ms (return two segment level forms guidance code) followed by the address code and order code. The address and command transfer for two times. The second time all bits are inverted, which can be used to confirm the received message. The total transmission time is constant, because every point is repeatable with inverted length. If you're not interested, you can ignore the reliability of inversion, can expand the address and command, to each of the 16 bits!
The transmitted pulse for button is pressed a period of time and then release:

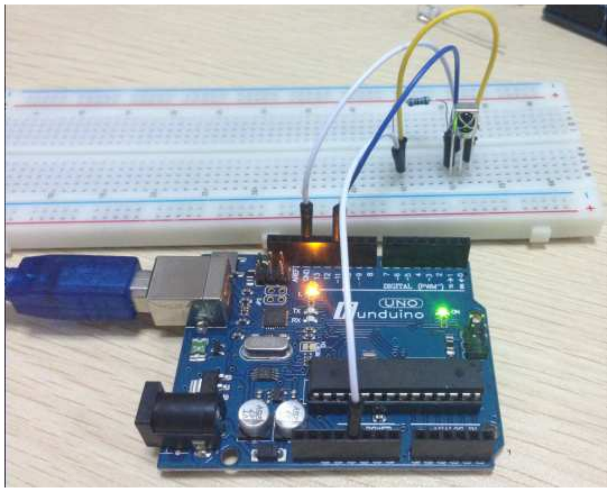


Sending a command at a time and pressing the buttons on the remote control all the time. When the button is pressed, the first 110ms's pulse is as above, repeat code transmission for one time in every 110ms. Return a repetition code is composed of a 9ms high level pulse and a low level of 2.25ms and a 560 μs high level.
·Repetition pulse



Note: after the pulse waveform is integrated into the receiving head, it is decoding, signal amplifying and shaping in the integrated receiver head, therefore, should pay attention to: in the absence of infrared signal, its output is high level and voce vice. The output signal level is opposite to emission port. The receiving port pulse we can see through the oscilloscope, so understand the program with waveform.

Circuit diagram:

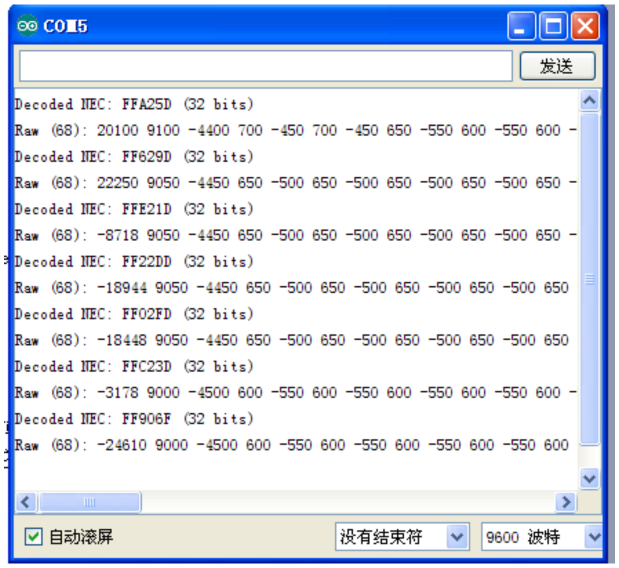


Reference program :

```
#include <IRremote.h>                 // new library
int RECV_PIN = 11;
int LED1 = 2;
int LED2 = 3;
int LED3 = 4;
int LED4 = 5;
int LED5 = 6;
int LED6 = 7;
long on1    = 0x00FFA25D;
long off1 = 0x00FFE01F;
long on2 = 0x00FF629D;
long off2 = 0x00FFA857;
long on3 = 0x00FFE21D;
long off3 = 0x00FF906F;
long on4 = 0x00FF22DD;
long off4 = 0x00FF6897;
long on5 = 0x00FF02FD;
long off5 = 0x00FF9867;
long on6 = 0x00FFC23D;
long off6 = 0x00FFB047;
IRrecv irrecv(RECV_PIN);
decode_results results;
                        // Dumps out the decode_results structure.
                        // Call this after IRrecv::decode()
                        // void * to work around compiler issue
                        //void dump(void *v) {
                        //    decode_results *results = (decode_results *)v

void dump(decode_results *results) {
   int count = results->rawlen;
   if (results->decode_type == UNKNOWN)
      {
        Serial.println("Could not decode message");
      }
   else
     {
      if (results->decode_type == NEC)
         {
           Serial.print("Decoded NEC: ");
         }
      else if (results->decode_type == SONY)
         {
           Serial.print("Decoded SONY: ");
         }
      else if (results->decode_type == RC5)
         {
           Serial.print("Decoded RC5: ");
         }
      else if (results->decode_type == RC6)
         {
            Serial.print("Decoded RC6: ");
          }
        Serial.print(results->value, HEX);
        Serial.print(" (");
        Serial.print(results->bits, DEC);
        Serial.println(" bits)");
      }
        Serial.print("Raw (");
        Serial.print(count, DEC);
        Serial.print("): ");
    for (int i = 0; i < count; i++)
        {
          if ((i % 2) == 1) {
          Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
        }
      else
        {
          Serial.print(-(int)results->rawbuf[i]*USECPERTICK, DEC);
        }
      Serial.print(" ");
        }
          Serial.println("");
        }

void setup()
  {
    pinMode(RECV_PIN, INPUT);
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(LED3, OUTPUT);
    pinMode(LED4, OUTPUT);
    pinMode(LED5, OUTPUT);
    pinMode(LED6, OUTPUT);
    pinMode(13, OUTPUT);
    Serial.begin(9600);
    irrecv.enableIRIn(); // Start the receiver
  }
int on = 0;
unsigned long last = millis();

void loop()
{
    if (irrecv.decode(&results))
      {
                                // If it's been at least 1/4 second since the last
                                // IR received, toggle the relay
        if (millis() - last > 250)
          {
            on = !on;
              // digitalWrite(8, on ? HIGH : LOW);
            digitalWrite(13, on ? HIGH : LOW);
            dump(&results);
          }
        if (results.value == on1 )
            digitalWrite(LED1, HIGH);
        if (results.value == off1 )
            digitalWrite(LED1, LOW);
        if (results.value == on2 )
            digitalWrite(LED2, HIGH);
        if (results.value == off2 )
            digitalWrite(LED2, LOW);
        if (results.value == on3 )
            digitalWrite(LED3, HIGH);
        if (results.value == off3 )
            digitalWrite(LED3, LOW);
        if (results.value == on4 )
            digitalWrite(LED4, HIGH);
        if (results.value == off4 )
            digitalWrite(LED4, LOW);
        if (results.value == on5 )
            digitalWrite(LED5, HIGH);
        if (results.value == off5 )
            digitalWrite(LED5, LOW);
        if (results.value == on6 )
            digitalWrite(LED6, HIGH);
        if (results.value == off6 )
            digitalWrite(LED6, LOW);
        last = millis();
        irrecv.resume(); // Receive the next value
    }
}
```

Program function:
Decoding code pulse emitted by the remote controller, implement appropriate action according to the results of decoding. In this way you can control your device with a remote control, let it receives your command.

screenshot



```
Decoded NEC: FFA25D (32 bits)
Raw (68): 20100 9100 -4400 700 -450 700 -450 650 -550 600 -550 600 -
Decoded NEC: FF629D (32 bits)
Raw (68): 22250 9050 -4450 650 -500 650 -500 650 -500 650 -500 650 -
Decoded NEC: FFE21D (32 bits)
Raw (68): -8718 9050 -4450 650 -500 650 -500 650 -500 650 -500 650 -
Decoded NEC: FF22DD (32 bits)
Raw (68): -18944 9050 -4450 650 -500 650 -500 650 -500 650 -500 650
Decoded NEC: FF02FD (32 bits)
Raw (68): -18448 9050 -4450 650 -500 650 -500 650 -500 650 -500 650
Decoded NEC: FFC23D (32 bits)
Raw (68): -3178 9000 -4500 600 -550 600 -550 600 -550 600 -550 600 -
Decoded NEC: FF906F (32 bits)
Raw (68): -24610 9000 -4500 600 -550 600 -550 600 -550 600 -550 600
```

✓ 自动滚屏　没有结束符　9600 波特

##