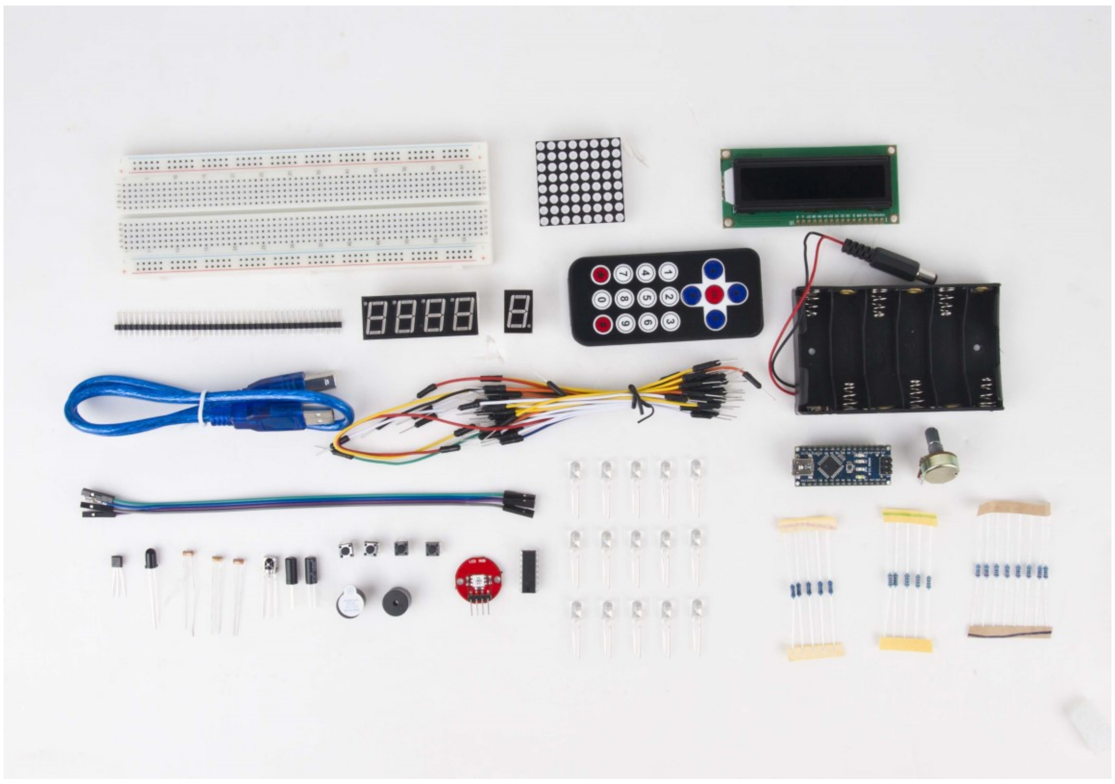
User Manual

For IDUINO Starter kit(KTS09)



www.openplatform.cc

1. Overview

1.1 what is Arduino?

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

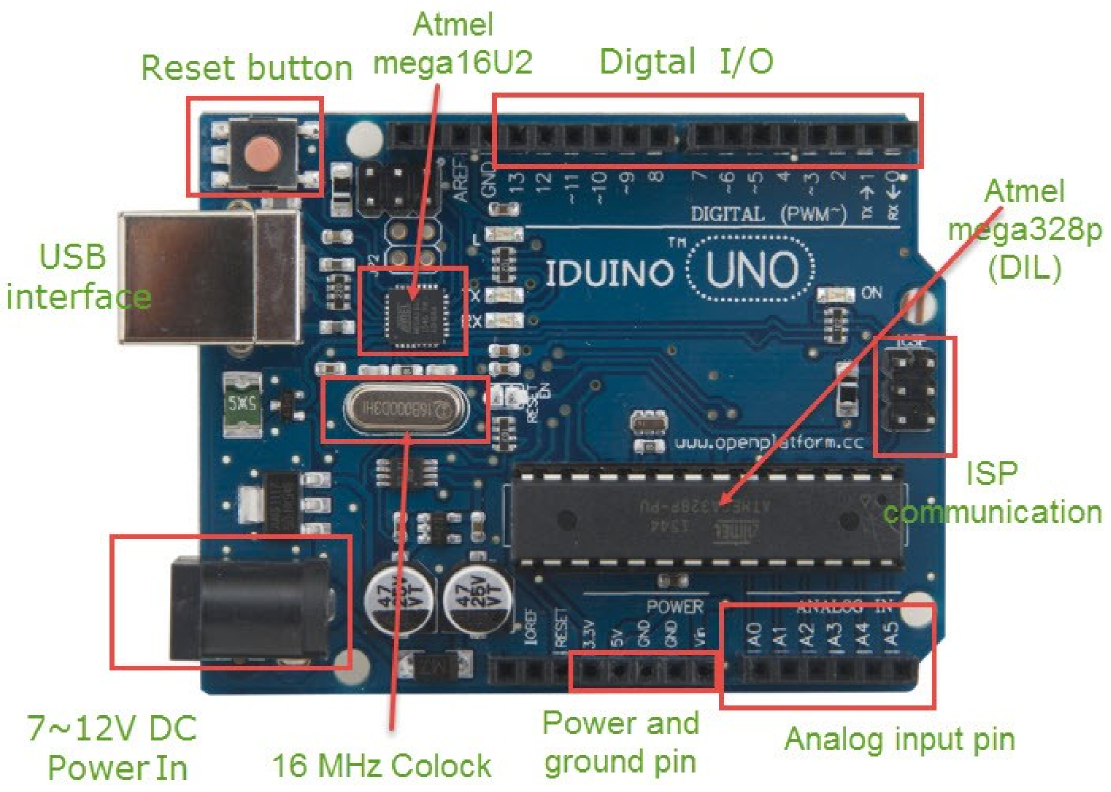
The official website is www.arduino.cc and www.arduino.org.

1.2 what is IDUINO ?

Because of the arduino technology is totally opensource, so anyone can use this facility to create more valuable products.

IDUINO is a series of Ardunio opensource products collection, which includes not only motherboard, but hundreds of sensors and modules used for Arduino board, and many kinds of Arduino Starter Kit, many kinds of Arduino projects, many kinds of car chassis , expansion board, accessories , Arduino DIY 3D Printer.

IDUINO are more focused on manufacturing and constructing Arduino project system.



1.3 What's the difference between Arduino and IDUINO?

For the development, IDUINO is just a different brand comparing with the Arduino development.

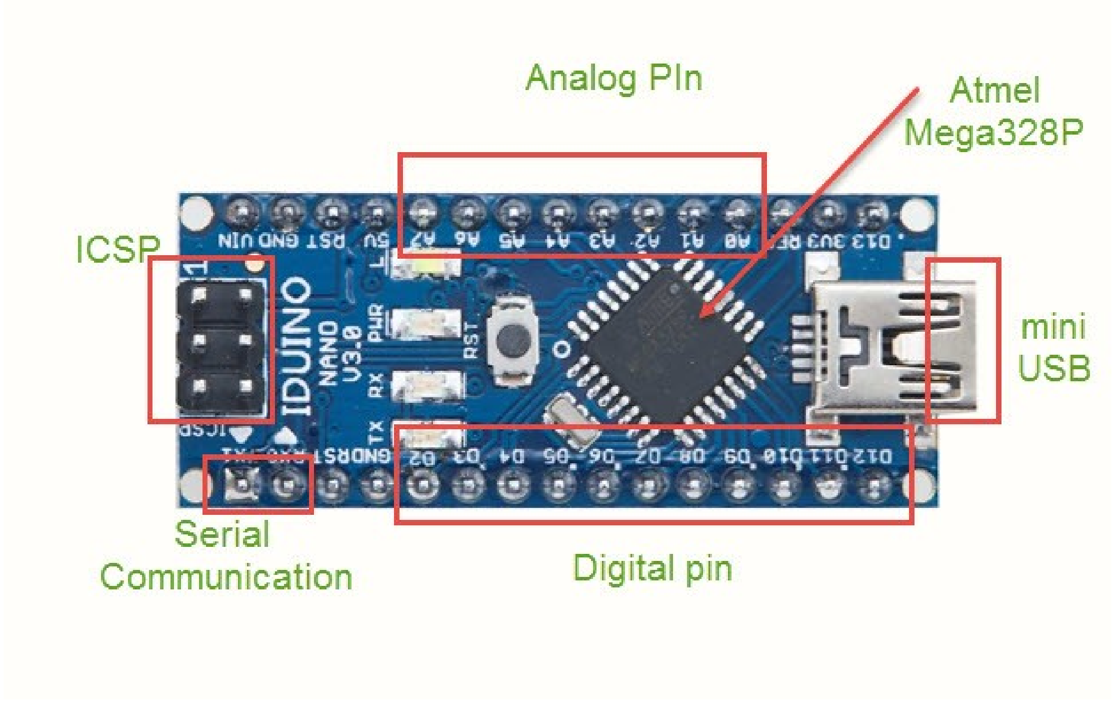
For other categories, IDUINO's quantity exceeds Arduino a lot.

2 Content list

- 1×Iduino Nano
- 1×Solderless Breadboard
- 1×USB Cable
- 1× 4 Digit Seven Segment Display
- 1× 1 Digit Seven Segment Display
- 1× 8*8 Dot Matrix Holder
- 1× 6AA Battery Holder
- Assorted Resistors
- Assorted LEDs
- RGB LED Module
- 4×Tactile Switches
- 2×Piezo Sounders
- Assorted Jumpers
- 1× 74HC595 Shift Register
- 1× 50 K Potentiometer
- 2×Light Dependant Resistors
- 1× Infrared Photo Diode
- 1×Infrared Remote Receiver
- 1×Analog Temperature Sensor
- 1× 17 Button IR Remote Control
- 1× 40-pin Header
- 1× Two Channel Relay Module

3. IDUINO Nano

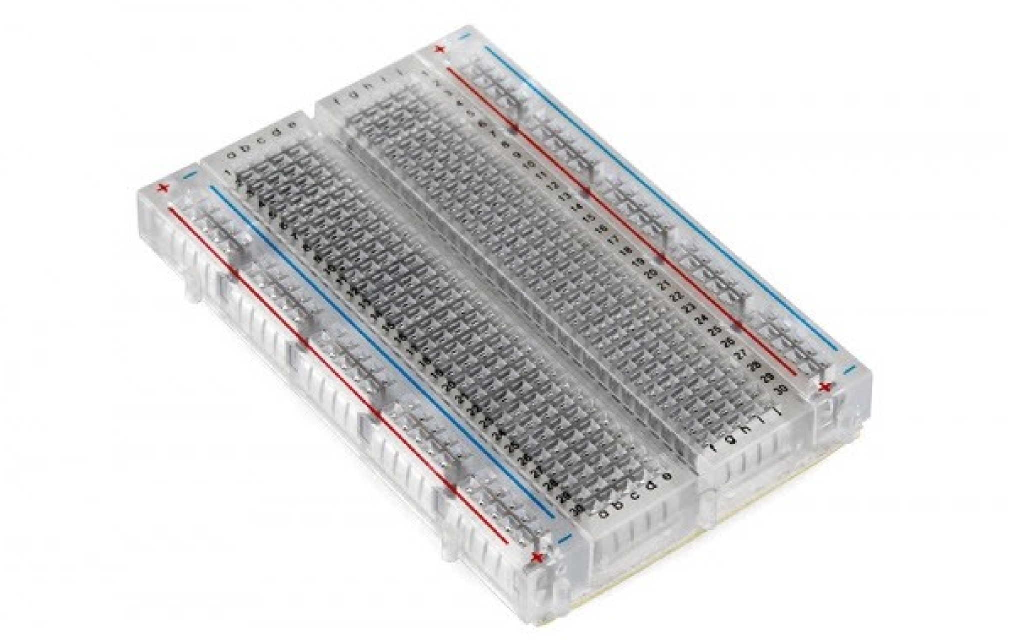
The IDUINO Nano is a small, complete, and breadboard-friendly board based on the ATmega328 or ATmega168.It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. The Nano was designed and is being produced by Gravitech.



Specifications:

Microcontroller	Atmel ATmega168 or ATmega328
Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz
Dimensions	0.73" x 1.70"
Length	45 mm
Width	18 mm
Weigth	5 g

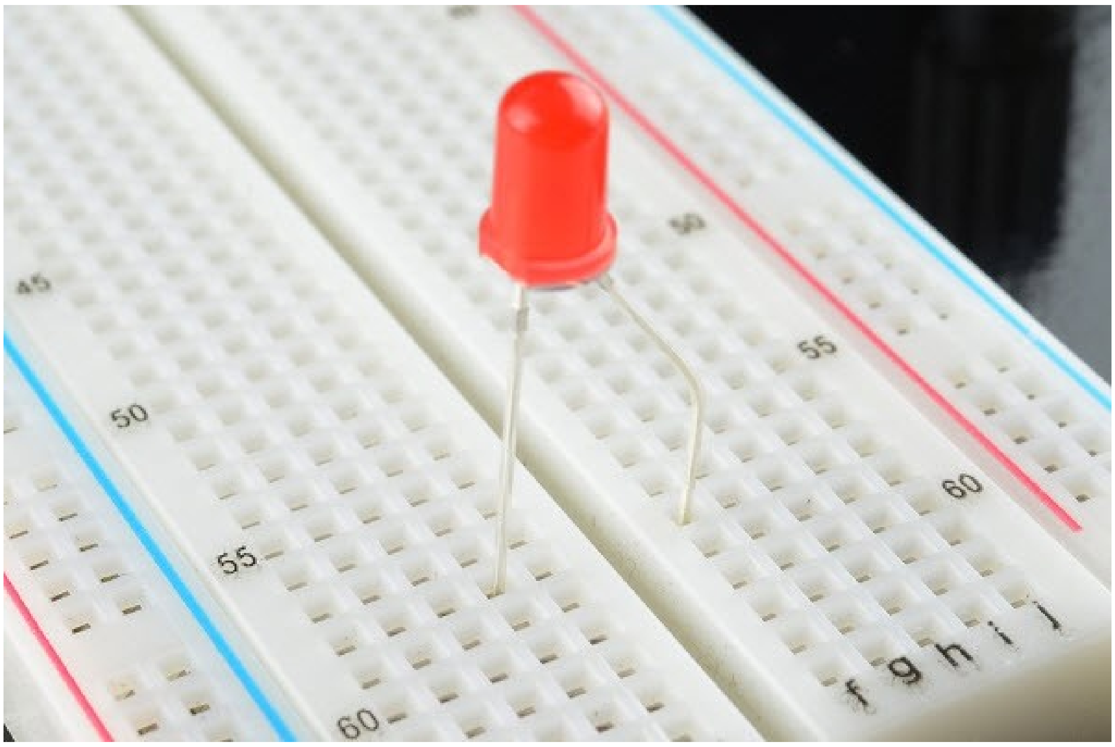
4. chapter 1: Using Breadboard

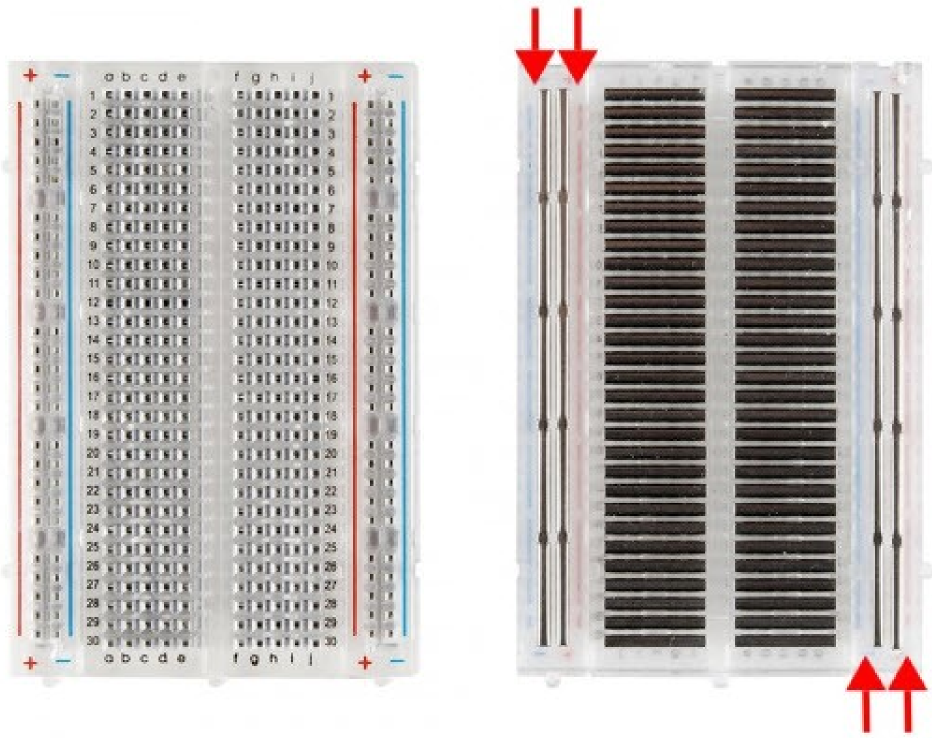


Introduction

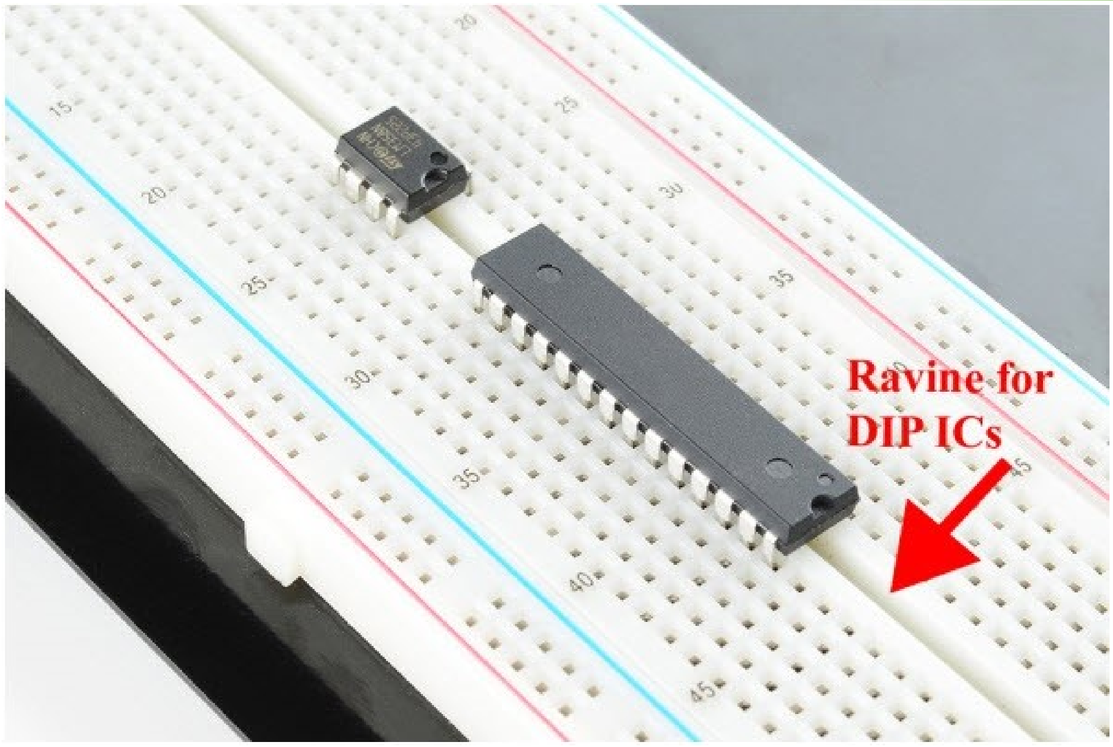
Breadboards are one of the most fundamental pieces when learning how to build circuits. In this tutorial, you will learn a little bit about what breadboards are, why they are called breadboards, and how to use one. Once you are done you should have a basic understanding of how breadboards work and be able to build a basic circuit on a breadboard.

Now that we've seen how the connections in a breadboard are made, let's look at a larger, more typical breadboard. Aside from horizontal rows, breadboards usually have what are called power rails that run vertically along the sides.

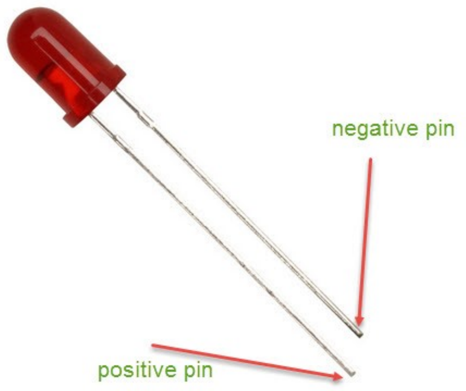




And these DIP chips (salsa anyone?) have legs that come out of both sides and fit perfectly over that ravine. Since each leg on the IC is unique, we don't want both sides to be connected to each other. That is where the separation in the middle of the board comes in handy. Thus, we can connect components to each side of the IC without interfering with the functionality of the leg on the opposite side.



5. chapter 2: LED blinking



Introduction

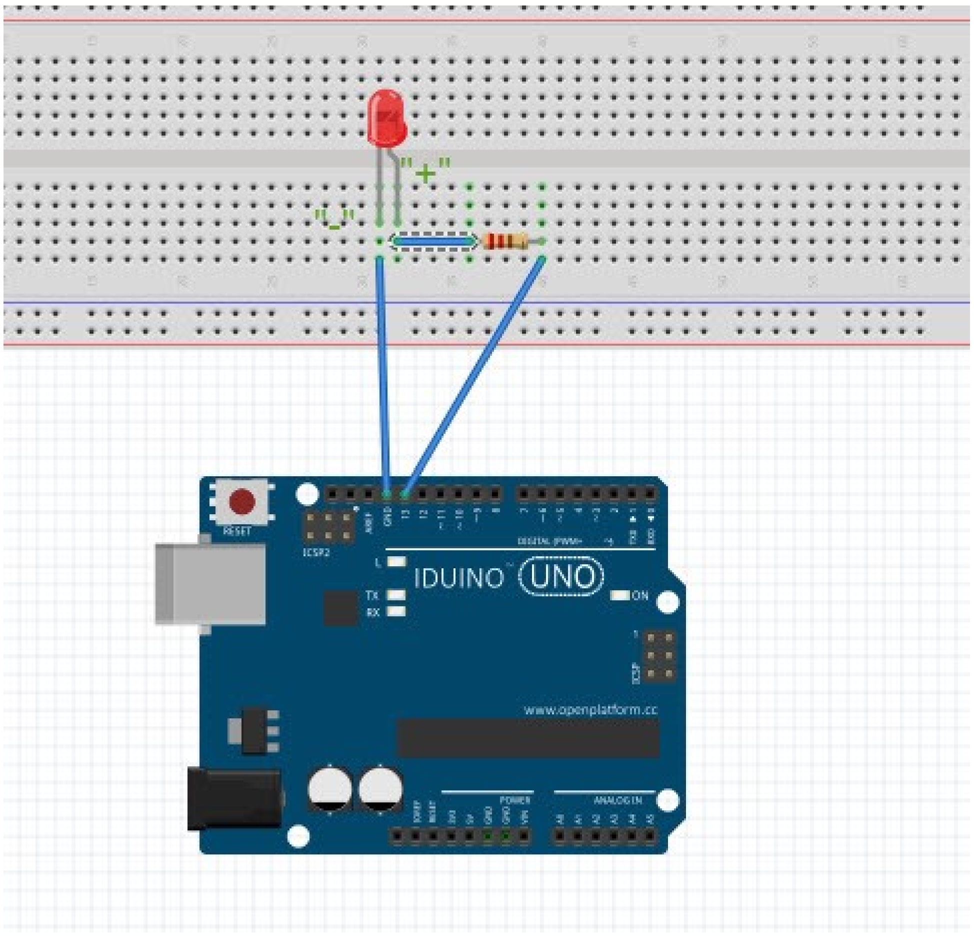
Blinking LED experiment is quite simple. In the "Hello World!" program, we have come across LED. This time, we are going to connect an LED to one of the digital pins rather than using LED13, which is soldered to the board. Except an Arduino and an USB cable, we will need extra parts as below:

Hardware required

1. Red M5 LED*1
2. 220Ω resistor*1
3. Breadboard*1
4. Breadboard jumper wires* several

We follow below diagram from the experimental schematic link. Here we use digital pin 10. We connect LED to a 220 ohm resistor to avoid high current damaging the LED.

Circuit connection:



Sample program

*******code begin*******

```
int ledPin = 10; // define digital pin 10.
void setup()
{
pinMode(ledPin, OUTPUT);// define pin with LED connected as output.
}
void loop()
```

```
{
digitalWrite(ledPin, HIGH); // set the LED on.
delay(1000); // wait for a second.
digitalWrite(ledPin, LOW); // set the LED off.
delay(1000); // wait for a second
}
*******code begin*******
```

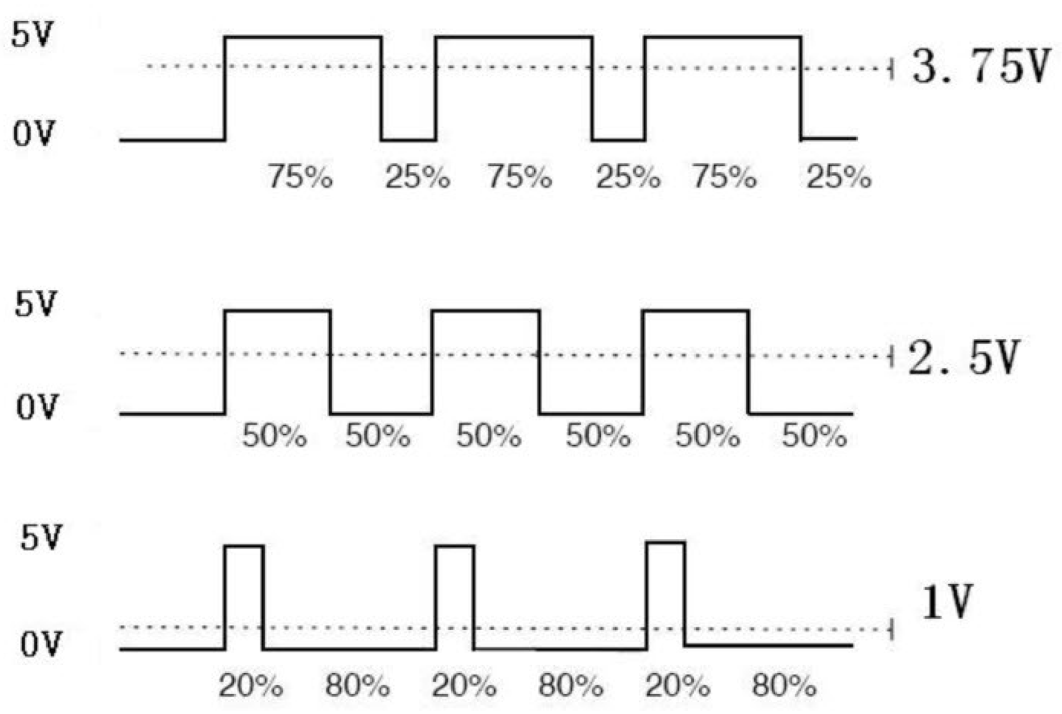
Result

After downloading this program, in the experiment, you will see the LED connected to pin 10 turning on and off, with an interval approximately one second. The blinking LED experiment is now completed.

6. Chapter3: PWM gradational LED

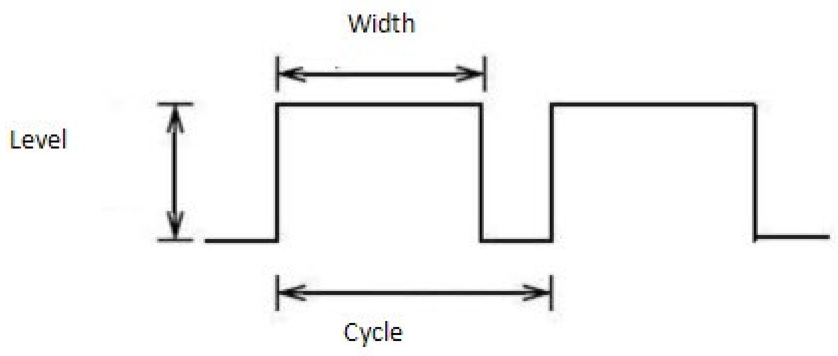
Introduction

PWM, short for Pulse Width Modulation, is a technique used to encode analog signal level into digital ones. A computer cannot output analog voltage but only digital voltage values such as 0V or 5V. So we use a high resolution counter to encode a specific analog signal level by modulating the duty cycle of PMW. The PWM signal is also digitalized because in any given moment, fully on DC power supply is either 5V (ON), or 0V (OFF). The voltage or current is fed to the analog load (the device that uses the power) by repeated pulse sequence being ON or OFF. Being on, the current is fed to the load; being off, it's not. With adequate bandwidth, any analog value can be encoded using PWM. The output voltage value is calculated via the on and off time. Output voltage = (turn on time/pulse time) * maximum voltage value.



PWM has many applications: lamp brightness regulating, motor speed regulating, sound making, etc.

The following are the three basic parameters of PMW:



1. The amplitude of pulse width (minimum / maximum)
2. The pulse period (The reciprocal of pulse frequency in 1 second)
3. The voltage level(such as：0V-5V）

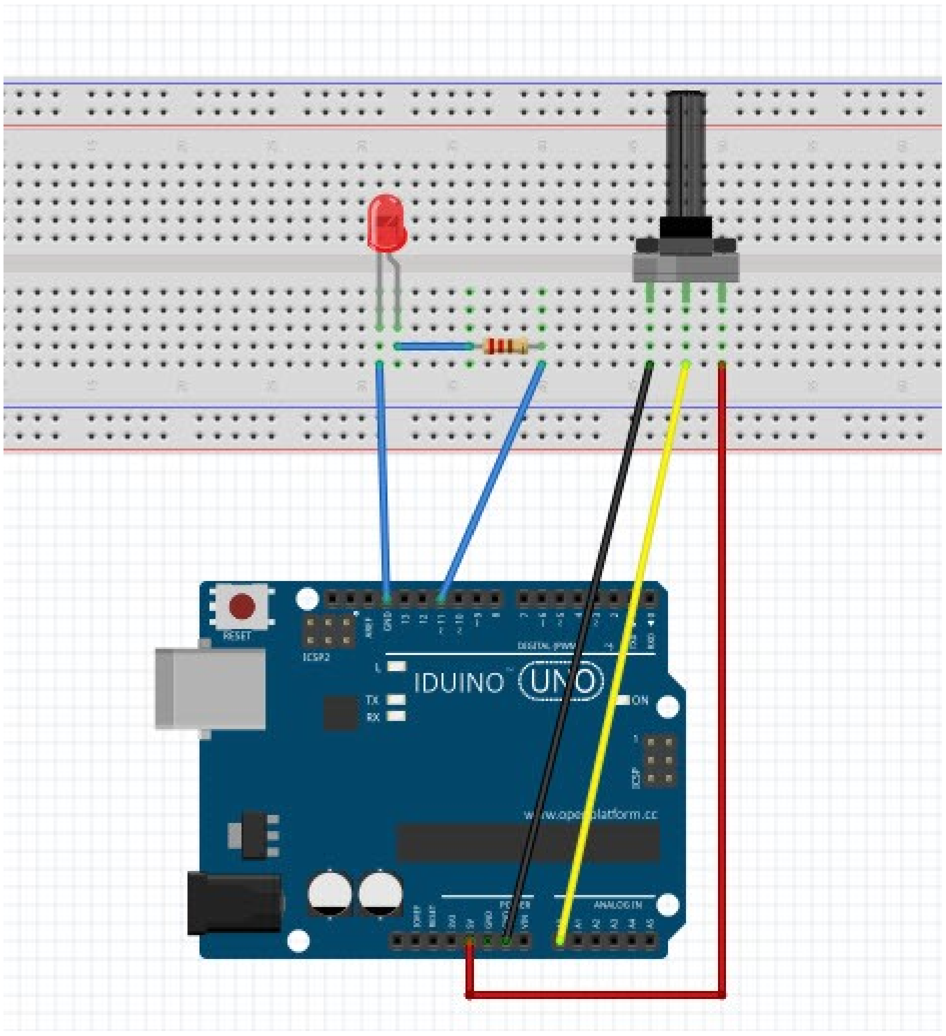
 There are 6 PMW interfaces on Arduino, namely digital pin 3, 5, 6, 9, 10, and 11. In previous experiments, we have done "button-controlled LED", using digital signal to control digital pin, also one about potentiometer. This time, we will use a potentiometer to control the brightness of the LED.

Hardware required

- Variable resistor*1
- Red M5 LED*1

- 220Ω resistor
- Breadboard*1
- Breadboard jumper wires

Circuit connection



Sample program

In the program compiling process, we will use the analogWrite (PWM interface, analog value) function. In this experiment, we will read the analog value of the potentiometer and assign the value to PWM port, so there will be corresponding change to the brightness of the LED. One final part will be displaying the analog value on the screen. You can consider this as the "analog value reading" project adding the PWM analog value assigning part. Below is a sample program for your reference.

*******code begin*******

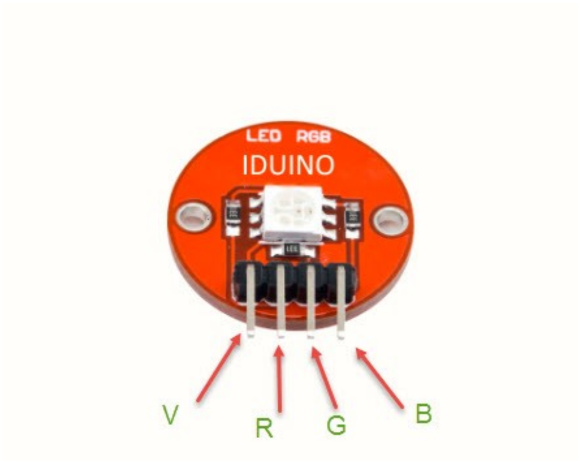
```
int potpin=0;// initialize analog pin 0
int ledpin=11;//initialize digital pin 11（PWM output）
int val=0;// Temporarily store variables' value from the sensor
void setup()
```

```
{
pinMode(ledpin,OUTPUT);// define digital pin 11 as “output”
Serial.begin(9600);// set baud rate at 9600
// attention: for analog ports, they are automatically set up as “input”
}
void loop()
{
val=analogRead(potpin);// read the analog value from the sensor and assign it to val
Serial.println(val);// display value of val
analogWrite(ledpin,val/4);// turn on LED and set up brightness（maximum output of
PWM is 255）
delay(10);// wait for 0.01 second
}
*******code end*******
```

Result

After downloading the program, when we rotate the potentiometer knob, we can see changes of the displaying value, also obvious change of the LED brightness on the breadboard.

7. Chapter 4: RGB LED Module



Introduction

SMD RGB LED module consists of a full-color LED made by R, G, B three pin PWM voltage input can be adjusted. Primary colors (red / blue / green) strength in order to achieve full color mixing effect. Control of the module with the Arduino can be achieved Cool lighting effects.

Specification

- Red Vf: 1.8 to 2.1V

- Green Vf: 3.0 to 3.2V
- Blue Vf: 3.0 to 3.2V
- Red color: 620-625 nm
- Green color: 520-525 nm
- Blue color: 465-470 nm
- Red brightness @ ~20mA: 600-800 mcd
- Blue brightness @ ~20mA: 800-1000 mcd
- Green brightness @ ~20mA: 1500-2000mcd

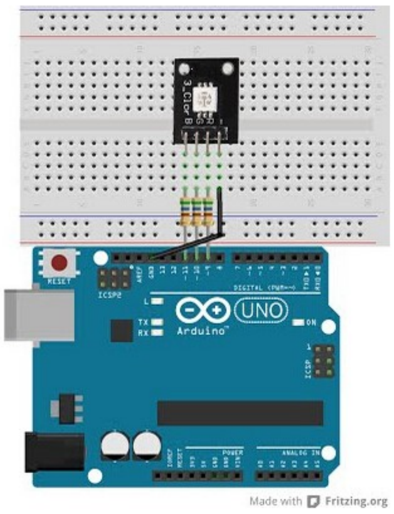
Pin Instructions

Pin Name	Description
"R"	Red light
"G"	Green light
"B"	Blue light
"-"	Ground

Example

In this example, we blink an LED and using an RGB LED we can generate any color our heart desires.

Here is the physical connection(color is different):



Example Code

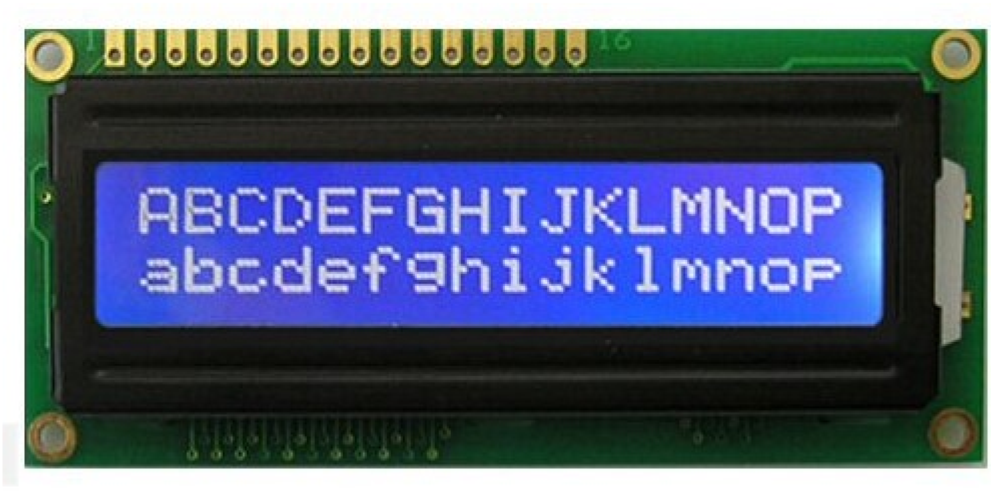
********Code begin********

```
//RGB LED pins
int ledDigitalOne[] = {10, 11, 9}; //the three digital pins of the digital LED
                                   //10 = redPin, 11 = greenPin, 9 = bluePin

const boolean ON = HIGH;        //Define on as LOW (this is because we use a common
                                 //Anode RGB LED (common pin is connected to +5
volts)
const boolean OFF = LOW;      //Define off as HIGH

//Predefined Colors
const boolean RED[] = {ON, OFF, OFF};
const boolean GREEN[] = {OFF, ON, OFF};
const boolean BLUE[] = {OFF, OFF, ON};
const boolean YELLOW[] = {ON, ON, OFF};
const boolean CYAN[] = {OFF, ON, ON};
const boolean MAGENTA[] = {ON, OFF, ON};
const boolean WHITE[] = {ON, ON, ON};
const boolean BLACK[] = {OFF, OFF, OFF};

//An Array that stores the predefined colors (allows us to later randomly display a
color)
const boolean* COLORS[] = {RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, WHITE,
BLACK};

void setup(){
  for(int i = 0; i < 3; i++){
    pinMode(ledDigitalOne[i], OUTPUT);      //Set the three LED pins as outputs
  }
}

void loop(){

/* Example - 1 Set a color
    Set the three LEDs to any predefined color
*/
    setColor(ledDigitalOne, YELLOW);       //Set the color of LED one

/* Example - 2 Go through Random Colors
  Set the LEDs to a random color
*/
    //randomColor();

}

void randomColor(){
  int rand = random(0, sizeof(COLORS) / 2);    //get a random number within the range
of colors
  setColor(ledDigitalOne, COLORS[rand]);    //Set the color of led one to a random
color
  delay(1000);
```

```
}

/* Sets an led to any color
      led - a three element array defining the three color pins (led[0] = redPin, led[1] =
greenPin, led[2] = bluePin)
      color - a three element boolean array (color[0] = red value (LOW = on, HIGH = off),
color[1] = green value, color[2] =blue value)
*/
void setColor(int* led, boolean* color){
  for(int i = 0; i < 3; i++){
      digitalWrite(led[i], color[i]);
  }
}

/* A version of setColor that allows for using const boolean colors
*/
void setColor(int* led, const boolean* color){
    boolean tempColor[] = {color[0], color[1], color[2]};
    setColor(led, tempColor);
}
```

********Code End********

8. Chapter 5: LCD1602 screen



Introduction

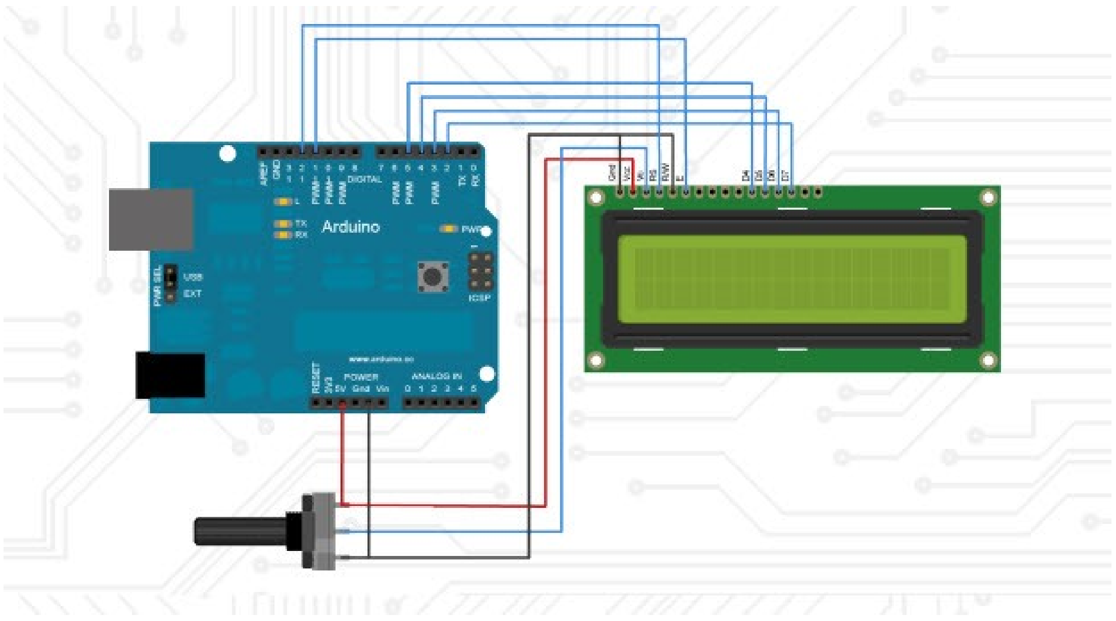
In this experiment, we use an Arduino to drive the 1602 LCD.
1602 LCD has wide applications. In the beginning, 1602 LCD uses a HD44780 controller. Now, almost all 1602 LCD module uses a compatible IC, so their features are basically the same.

1602LCD main parameters:
Display capacity: 16 * 2 characters.
Chip operating voltage: 4.5 ~ 5.5V.
Working current: 2.0mA (5.0V).
Optimum working voltage of the module is 5.0V.
Character size: 2.95 * 4.35 (W * H) mm.

Pin description of 1602 LCD

Pin name	Pin description	Pin name	Pin description
VSS	Power GND	D2	Date I/O
VDD	Power positive	D3	Date I/O
VL	LCD voltage bias signal	D4	Date I/O
RS	Select data/command(V /L)	D5	Date I/O
R/W	Select read/write(H/L)	D6	Date I/O
E	Enable signal	D7	Date I/O
D0	Date I/O	BLA	Back light power posit ive
D1	Date I/O	BLK	Back light power nega tive

Circuit connection



Example code

*******code begin*******

#include span style="color: #cc6600;">LiquidCrystal.h> //Include LCDs LibraryLiquidCrystal lcd(12, 11, 5, 4, 3, 2); //Attach LCDs and Arduino pin comunnication

int time; //Entire variable declaration(time)

void setup()//setup section

{

lcd.begin(16, 2); //LCD begins. dimension: 16x2(Coluns x Rows)

lcd.setCursor(0, 0); // Positions the cursor in the first column (0) and the firt row (1) at LCD

lcd.print("LiquidCrystal.h"); //LCD write comand"LiquidCrystal.h"

lcd.setCursor(0, 1); // Positions the cursor in the first column (0) and the second row (1) at LCD

lcd.print("GarageLab"); // LCD write command "GarageLab"

```
}



void loop()

{

lcd.setCursor(13, 1); // Positions the cursor on the fourteenth column (13) and the
second line (1) LCD

lcd.print(time); // Write the current value of the count variable in the LCD

delay(1000); // Waits for 1 second

time++; // Increment count variable



if(time == 600) // If the variable temp get to 600 (10 minutes), ...

    {

    time = 0; //... resets the count variable

    }



}
```

*******code End*******

9. Chapter6: 2-Channel 5V relay

Introduction

Arduino Relay Shield employs high quality relay with two channels input and two channels output. It can be connected to 250V/10A AC element or 24V/10A DC element to the maximum, therefore, it can be used to control lights, motors and etc.

The modularized design makes it easy to connect to Arduino expansion board. The output state of the relay is shown by a LED for the convenience of actual application.

Specification

Control signal: TTL voltage
Rated load:
10A 250VAC
10A 125VAC
10A 30DC
10A 28VDC
Rated Through-current: 10A(NO) 5A(NC)
Max Switching Voltage: 250VAC 30VDC
Contact actuation time: ＜10ms
Definition of module pins:
i) Pin 1- Pin 2----Controlling end
ii) Power supply (VCC)
iii) Ground (GND)

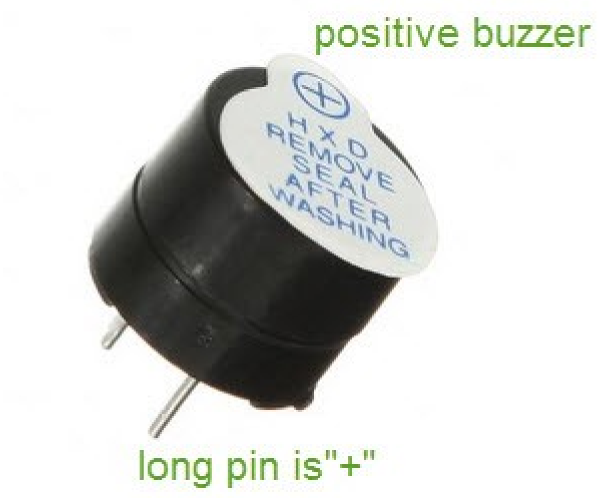
Pinout

Pin Name	Description
"Vcc"	Power(5V DC)
"GND"	Gnd
"in1"	Singal pin, connected with Arduino and control Relay 1
"in2"	Singal pin, connected with Arduino and control Relay 2
"COM"	Common pin, which usually directly connect with the" Gnd" unless you want to change the TTL mode(default the HIGH level activate)
"NO"	Normally Open Connection
"NC"	Normally Closed Connection
"C"(middle pin)	Common Connection, Which connected with the power for the load.

Example code

```
*********Code begin*********
#define RELAY1    6
#define RELAY2    7

void setup()
{
// Initialise the Arduino data pins for OUTPUT
   pinMode(RELAY1, OUTPUT);

   pinMode(RELAY2, OUTPUT);
}

 void loop()
{
     digitalWrite(RELAY1,LOW);                // Turns ON Relays 1
     delay(2000);                                          // Wait 2 seconds
     digitalWrite(RELAY1,HIGH);              // Turns Relay Off



     digitalWrite(RELAY2,LOW);                // Turns ON Relays 4
     delay(2000);                                          // Wait 2 seconds
     digitalWrite(RELAY2,HIGH);              // Turns Relay Off
 }
*********Code End*********
```

10. Chapter7: Positive buzzer

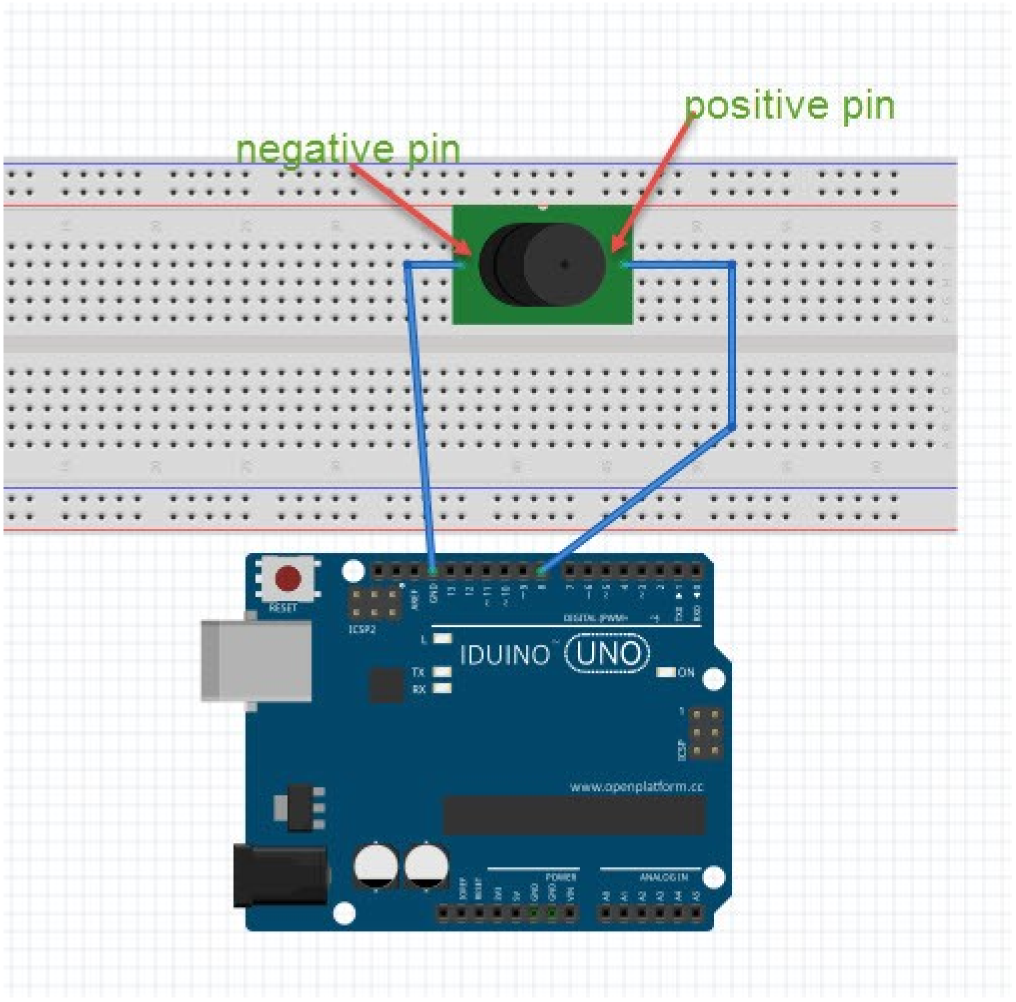


Introduction

Active buzzer is widely used on computer, printer, alarm, electronic toy, telephone, timer etc as a sound making element. It has an inner vibration source. Simply connect it with 5V power supply, it can buzz continuously.
Hardware required

Buzzer*1
Key *1
Breadboard*1
Breadboard jumper wires

Circuit connection



Example code

*******code begin*******
```
/////////////////////////////////////////////////////////////
int buzzer=8;// initialize digital IO pin that controls the buzzer
```

```
void setup()
{
  pinMode(buzzer,OUTPUT);// set pin mode as "output"
}
void loop()
{
digitalWrite(buzzer, HIGH); // produce sound
}
/////////////////////////////////////////////////////////
```

*******code End*******

Result

After downloading the program, the buzzer experiment is completed. You can see the buzzer is ringing.

11.Chapter8: Photo Resistor

Introduction

After completing all the previous experiments, we acquired some basic understanding and knowledge about Arduino application. We have learned digital input and output, analog input and PWM. Now, we can begin the learning of sensors applications.

Photo resistor (Photovaristor) is a resistor whose resistance varies according to different incident light strength. It's made based on the photoelectric effect of semiconductor. If the incident light is intense, its resistance reduces; if the incident light is weak, the resistance increases. Photovaristor is commonly applied in the measurement of light, light control and photovoltaic conversion (convert the change of light into the change of electricity).

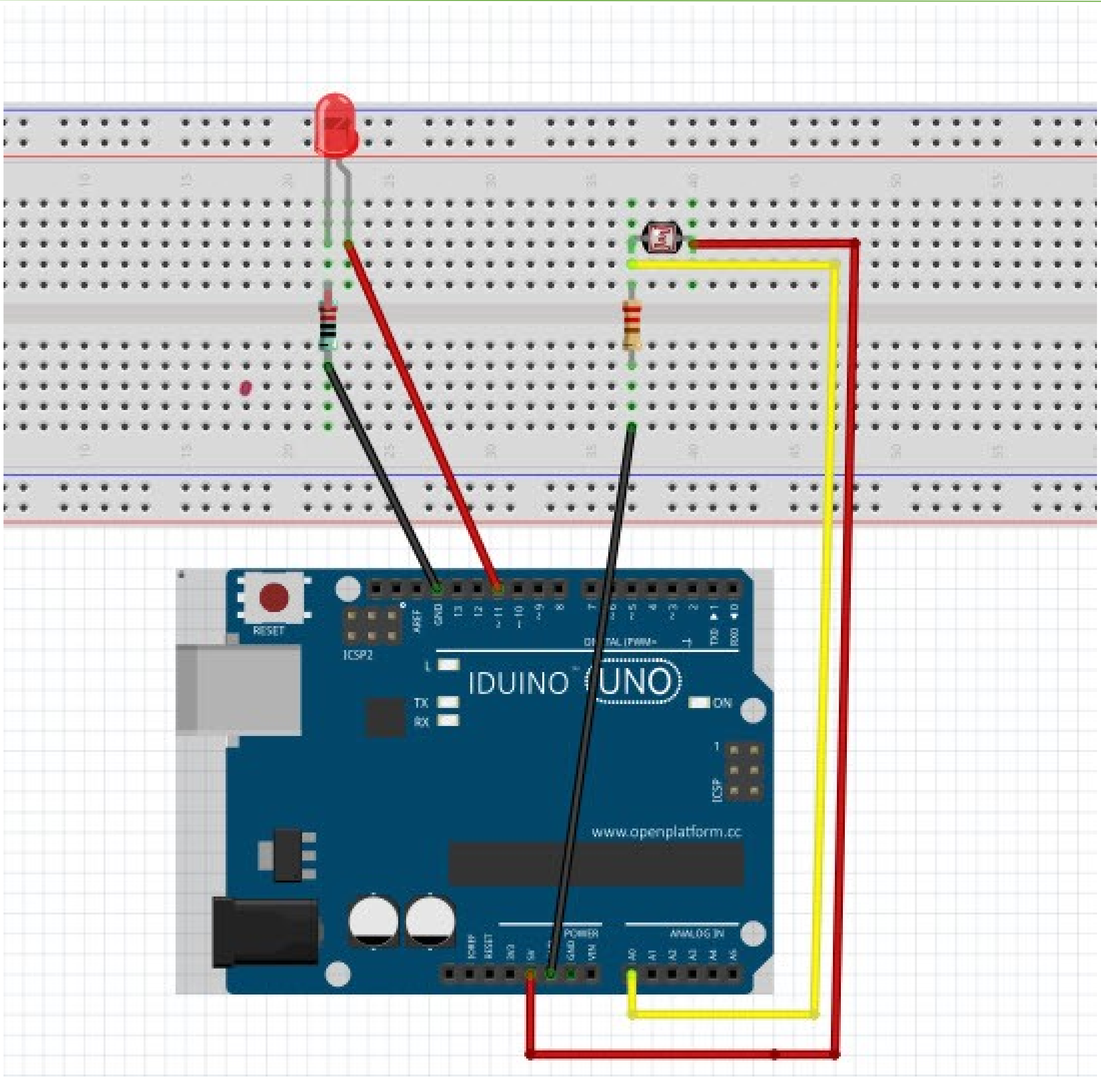
Photo resistor is also being widely applied to various light control circuit, such as light control and adjustment, optical switches etc.

We will start with a relatively simple experiment regarding photovaristor application. Photovaristor is an element that changes its resistance as light strenth changes. So we will need to read the analog values. We can refer to the PWM experiment, replacing the potentiometer with photovaristor. When there is change in light strength, there will be corresponding change on the LED.

Hardware required

Photo resistor*1
Red M5 LED*1
10KΩresistor*1
220Ωresistor*1
Bread board*1
Bread board jumper wires

Circuit connection



Example code

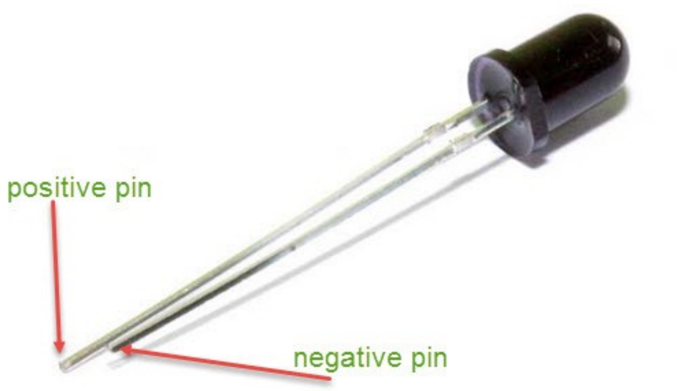
*******code begin*******

```
///////////////////////////////////////////////////////////
int potpin=0;// initialize analog pin 0, connected with photovaristor
int ledpin=11;// initialize digital pin 11, output regulating the brightness of LED
int val=0;// initialize variable va
void setup()
{
pinMode(ledpin,OUTPUT);// set digital pin 11 as "output"
Serial.begin(9600);// set baud rate at "9600"
}
void loop()
{
val=analogRead(potpin);// read the analog value of the sensor and assign it to val
Serial.println(val);// display the value of val
analogWrite(ledpin,val);// turn on the LED and set up brightness（maximum output
value 255）
delay(10);// wait for 0.01
}
////////////////////////////////////////////////////////////
*******code End*******
```

Result

After downloading the program, you can change the light strength around the photovaristor and see corresponding brightness change of the LED. Photovaristors has various applications in our everyday life. You can make other interesting interactive projects base on this one.

12. Chapter9: Flame sensor

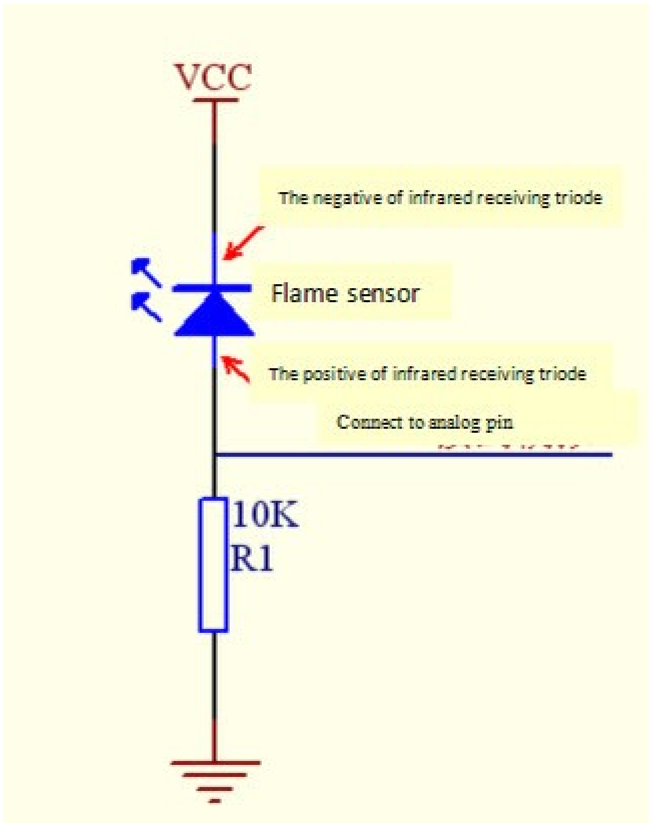


Introduction

Flame sensor (Infrared receiving triode) is specially used on robots to find the fire source. This sensor is of high sensitivity to flame. Below is a photo of it.

Working principle:
Flame sensor is made based on the principle that infrared ray is highly sensitive to flame. It has a specially designed infrared receiving tube to detect fire, and then convert the flame brightness to fluctuating level signal. The signals are then input into the central processor and be dealt with accordingly.
Sensor connection

The shorter lead of the receiving triode is for negative, the other one for positive. Connect negative to 5V pin, positive to resistor; connect the other end of the resistor to GND, connect one end of a jumper wire to a clip which is electrically connected to sensor positive, the other end to analog pin. As shown below:


VCC
The negative of infrared receiving triode
Flame sensor
The positive of infrared receiving triode
Connect to analog pin
10K
R1

Hardware required

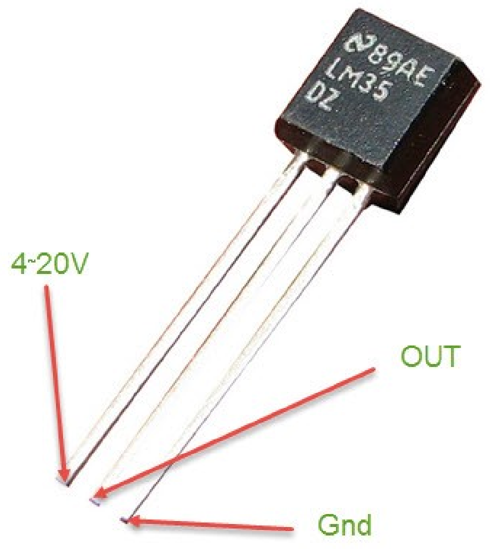
Flame sensor *1
Buzzer *1
10K resistor *1
Breadboard jumper wires

Example Code

```
*******code End*******
int flame=0;// select analog pin 0 for the sensor
int Beep=9;// select digital pin 9 for the buzzer
int val=0;// initialize variable
void setup()
{
  pinMode(Beep,OUTPUT);// set LED pin as “output”
pinMode(flame,INPUT);// set buzzer pin as “input”
Serial.begin(9600);// set baud rate at “9600”
}
void loop()
{
  val=analogRead(flame);// read the analog value of the sensor
  Serial.println(val);// output and display the analog value
  if(val>=600)// when the analog value is larger than 600, the buzzer will buzz
```

```
 {
   digitalWrite(Beep,HIGH);
   }else
   {
      digitalWrite(Beep,LOW);
     }
   delay(500);
}
```
*******code End*******

13 Chapter10: LM35 Temperature Sensor



Introduction

LM35 is a common and easy-to-use temperature sensor. It does not require other hardware. You just need an analog port to make it work. The difficulty lies in compiling the code to convert the analog value it reads to celsius temperature.

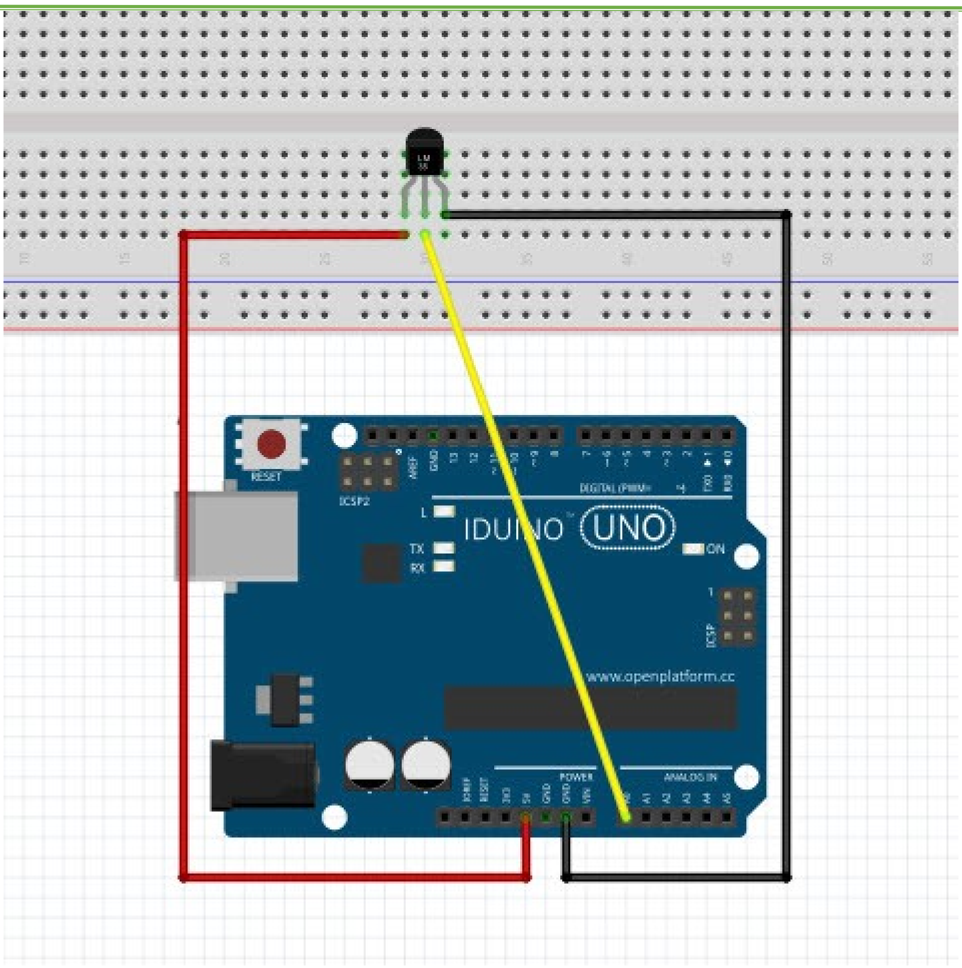
Hardware required

LM35*1

Breadboard*1

Breadboard jumper wires

Circuit connection



Example code

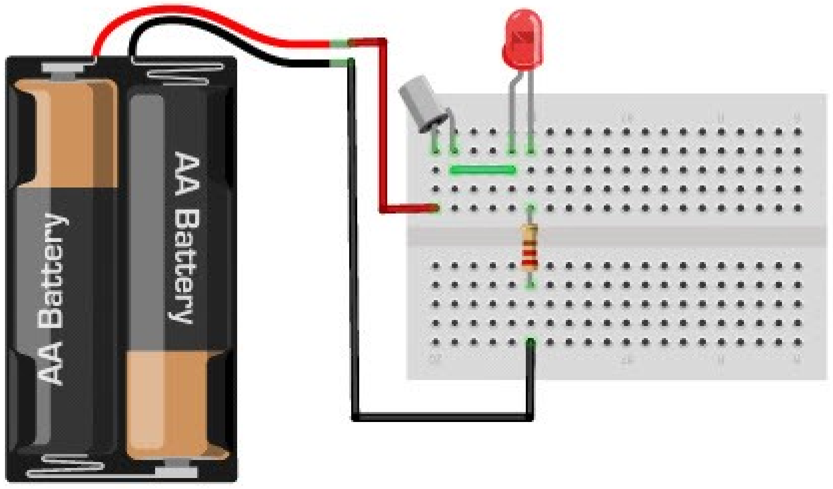
```
*******code begin*******
int potPin = 0; // initialize analog pin 0 for LM35 temperature sensor
void setup()
{
Serial.begin(9600);// set baud rate at”9600”
}
void loop()
{
int val;// define variable
int dat;// define variable
val=analogRead(0);// read the analog value of the sensor and assign it to val
dat=(125*val)>>8;// temperature calculation formula
Serial.print("Tep:");// output and display characters beginning with Tep
Serial.print(dat);// output and display value of dat
Serial.println("C");// display “C” characters
delay(500);// wait for 0.5 second
}
*******code End*******
```

Result
After downloading the program, you can open the monitoring window to see current temperature.

14. Chapter11: Tilt sensor switch

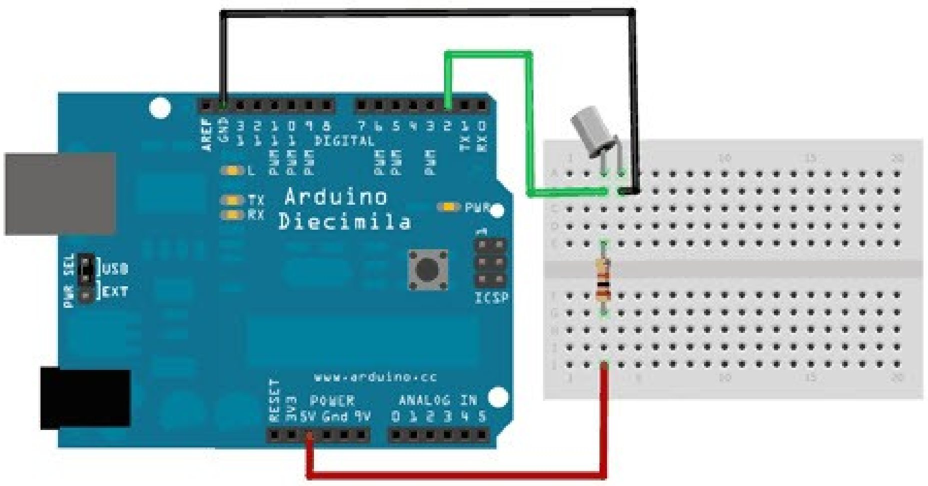
Tilt sensors allow you to detect orientation or inclination. They are small, inexpensive, low-power and easy-to-use. If used properly, they will not wear out. Their simplicitiy makes them popular for toys, gadgets and appliances. Sometimes they are referred to as "mercury switches", "tilt switches" or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

Simple Tilt-Activated LED
This is the most basic way of connecting to a tilt switch, but can be handy while one is learning about them. Simply connect it in series with an LED, resistor and battery. Tilt to turn on and off.



Reading Switch State with a Microcontroller

Note that the layout above shows a 10K pullup resistor but for the code I use the 'built-in' pullup resistor that you can turn on by setting an input pin to HIGH output (its quite neat!) If you use the internal pull-up you can skip the external one.



Example code

```
*******code begin*******
int inPin = 2;              // the number of the input pin
int outPin = 13;            // the number of the output pin

int LEDstate = HIGH;        // the current state of the output pin
int reading;                // the current reading from the input pin
int previous = LOW;         // the previous reading from the input pin
```

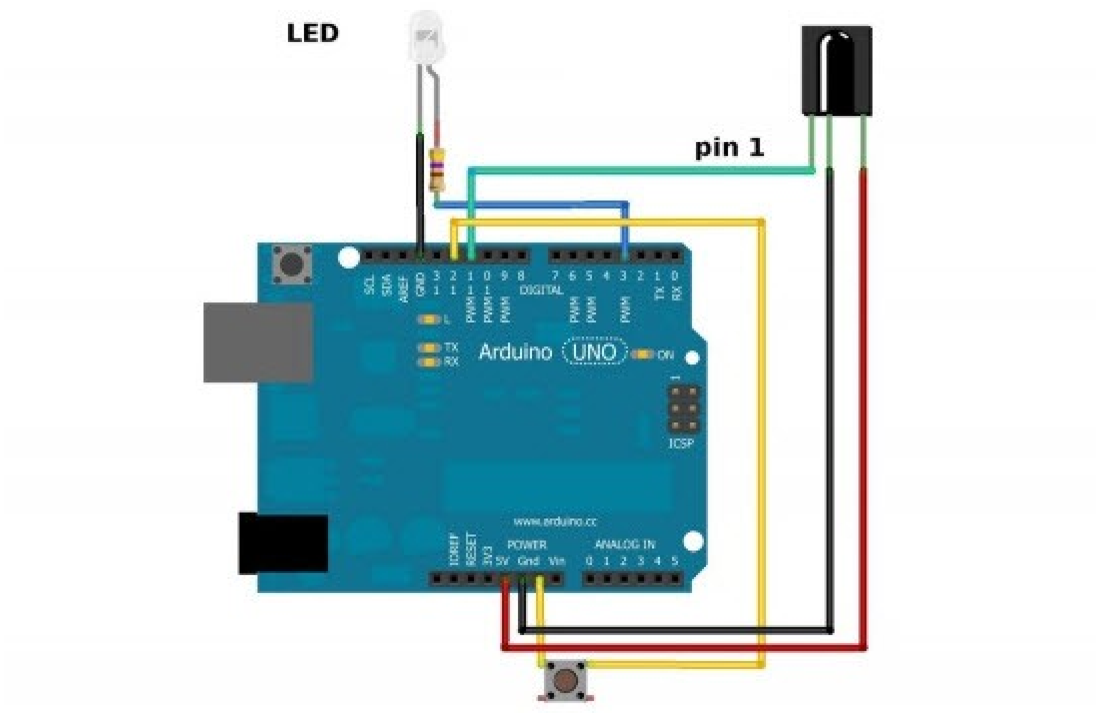
```
// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 50;   // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  digitalWrite(inPin, HIGH);   // turn on the built in pull-up resistor
  pinMode(outPin, OUTPUT);
}

void loop()
{
  int switchstate;

  reading = digitalRead(inPin);

  // If the switch changed, due to bounce or pressing...
  if (reading != previous) {
    // reset the debouncing timer
    time = millis();
  }

  if ((millis() - time) > debounce) {
     // whatever the switch is at, its been there for a long time
     // so lets settle on it!
     switchstate = reading;

     // Now invert the output on the pin13 LED
    if (switchstate == HIGH)
      LEDstate = LOW;
    else
      LEDstate = HIGH;
  }
  digitalWrite(outPin, LEDstate);

  // Save the last reading so we keep a running tally
  previous = reading;
}
```

*******code End*******

15. Chapter12: infrared receiver

IR, or infrared, communication is a common, inexpensive, and easy to use wireless communication technology. IR light is very similar to visible light, except that it has a slightltly longer wavelength. This means IR is undetectable to the human eye - perfect for wireless communication. For example, when you hit a button on your TV remote, an IR LED repeatedly turns on and off, 38,000 time a second, to transmit information (like volume or channel control) to an IR photo sensor on your TV.

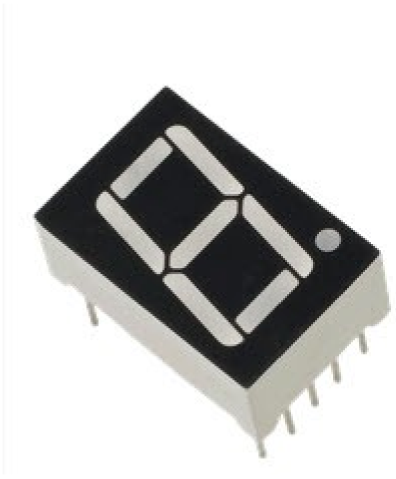
Circuit connection

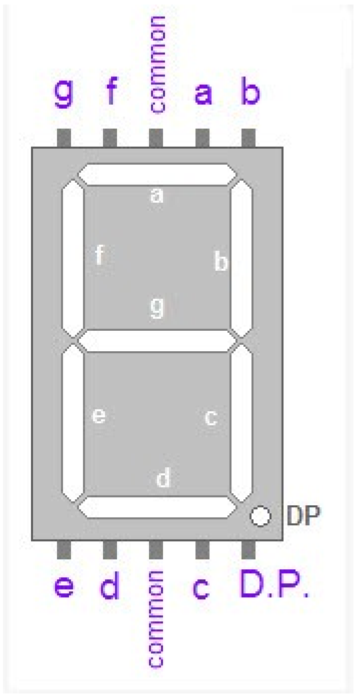


Example code

```
*******code begin*******
#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
}
*******code End*******
```

16 Chapter13: 1-bit digit 7-segment display



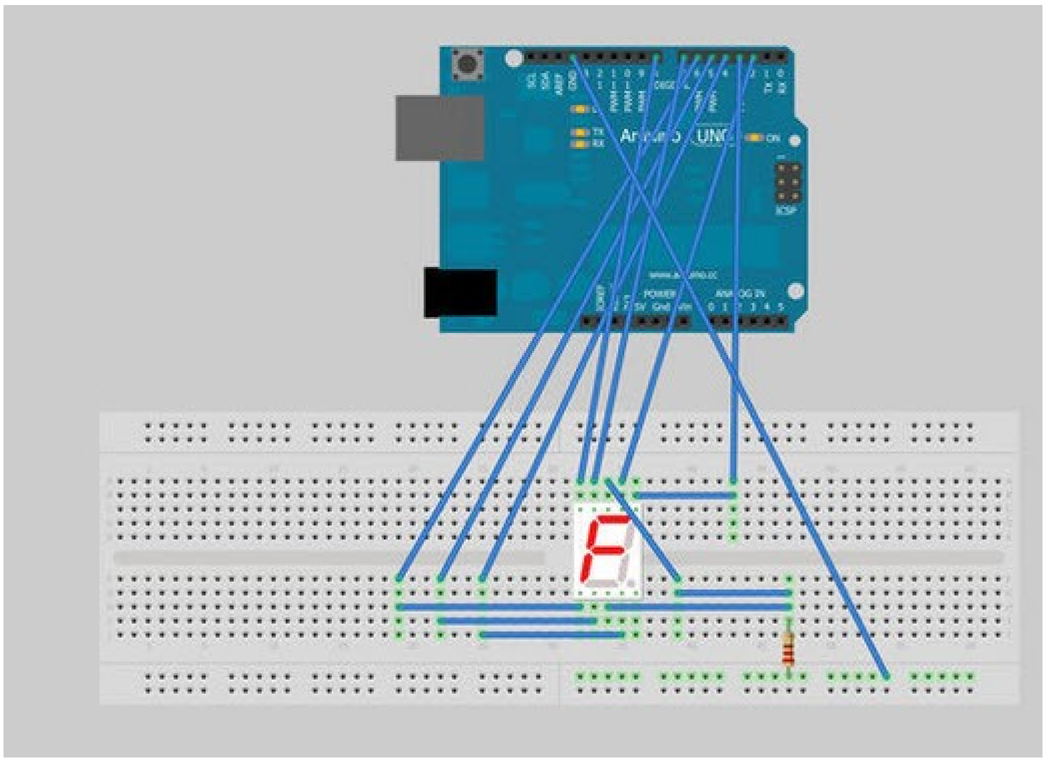


Introduction

LED segment displays are common for displaying numerical information. It's widely applied on displays of electromagnetic oven, full automatic washing machine, water temperature display, electronic clock etc. It is necessary that we learn how it works. LED segment display is a semiconductor light-emitting device. Its basic unit is a light-emitting diode (LED). LED segment display can be divided into 7-segment display and 8-segment display according to the number of segments. 8-segment display has one more LED unit (for decimal point display) than 7-segment one. In this experiment, we use a 8-segment display. According to the wiring method of LED units, LED segment displays can be divided into display with common anode and display with common cathode. Common anode display refers to the one that combine all the anodes of LED units into one common anode (COM).

For the common anode display, connect the common anode (COM) to +5V. When the cathode level of a certain segment is low, the segment is on; when the cathode level of a certain segment is high, the segment is off. For the common cathode display, connect the common cathode (COM) to GND. When the anode level of a certain segment is high, the segment is on; when the anode level of a certain segment is low, the segment is off.

Circuit connection



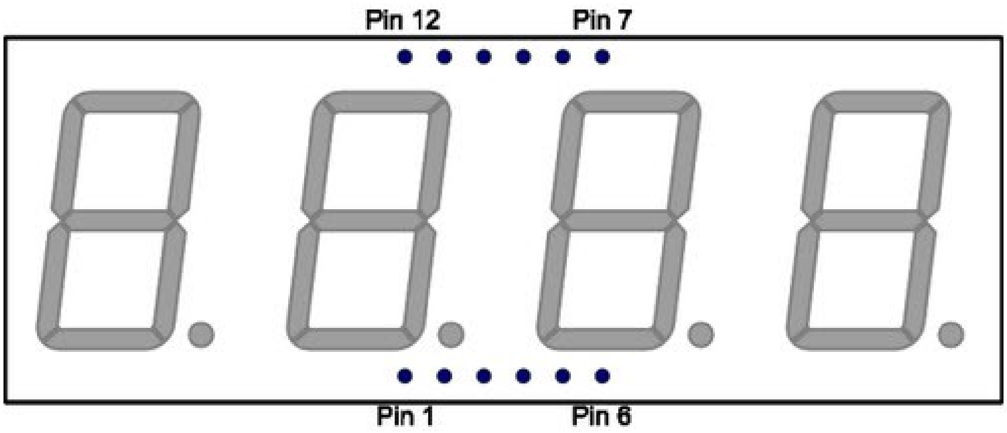
Example code

*******code begin*******

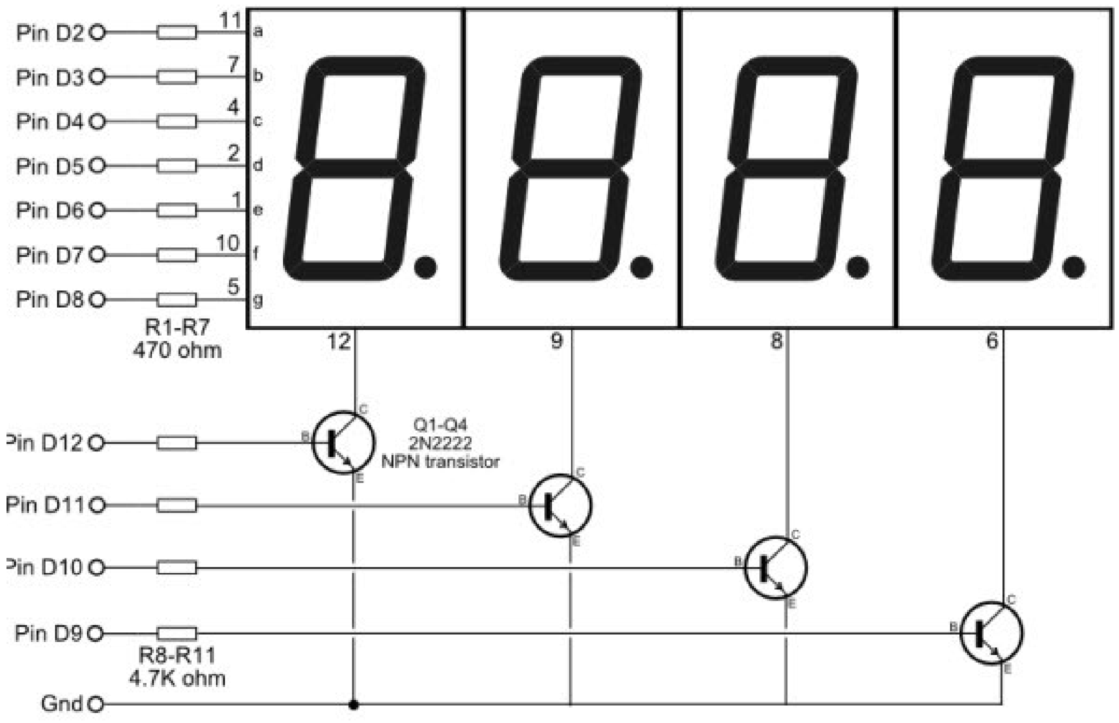
```
// Define the LED digit patters, from 0 - 9
// Note that these patterns are for common cathode displays
// For common anode displays, change the 1's to 0's and 0's to 1's
// 1 = LED on, 0 = LED off, in this order:
//                                                  Arduino pin: 2,3,4,5,6,7,8
byte seven_seg_digits[10][7] = { { 1,1,1,1,1,1,0 },   // = 0
                                                                                  {
0,1,1,0,0,0,0 },   // = 1
                                                                                  {
1,1,0,1,1,0,1 },   // = 2
                                                                                  {
1,1,1,1,0,0,1 },   // = 3
                                                                                  {
0,1,1,0,0,1,1 },   // = 4
                                                                                  {
1,0,1,1,0,1,1 },   // = 5
                                                                                  {
1,0,1,1,1,1,1 },   // = 6
```

```
                                                                        {
1,1,1,0,0,0,0 },   // = 7
                                                                        {
1,1,1,1,1,1,1 },   // = 8
                                                                        {
1,1,1,0,0,1,1 }     // = 9
                                                                        };

void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  writeDot(0);   // start with the "dot" off
}

void writeDot(byte dot) {
  digitalWrite(9, dot);
}

void sevenSegWrite(byte digit) {
  byte pin = 2;
  for (byte segCount = 0; segCount < 7; ++segCount) {
    digitalWrite(pin, seven_seg_digits[digit][segCount]);
    ++pin;
  }
}

void loop() {
  for (byte count = 10; count > 0; --count) {
   delay(1000);
   sevenSegWrite(count - 1);
  }
  delay(4000);
}
*******code End*******
```

17. Chapter14: 1-bit digit 7-segment display



Viewed from top



Example code:

```
  /*This is an example of how to drive a 7 segment LED display from an ATmega
without the use of current limiting resistors.
  This technique is very common but requires some knowledge of electronics - you do
run the risk of dumping too
  much current through the segments and burning out parts of the display. If you use
the stock code you should be ok, but
  be careful editing the brightness values.

  This code should work with all colors (red, blue, yellow, green) but the brightness
will vary from one color to the next
  because the forward voltage drop of each color is different. This code was written
and calibrated for the red color.

  This code will work with most Arduinos but you may want to re-route some of the
pins.

  7 segments
  4 digits
  1 colon
  =
  12 pins required for full control

  */
```

*******code Begin*******

```
int digit1 = 11; //PWM Display pin 1
int digit2 = 10; //PWM Display pin 2
int digit3 = 9; //PWM Display pin 6
int digit4 = 6; //PWM Display pin 8

//Pin mapping from Arduino to the ATmega DIP28 if you need it
//http://www.arduino.cc/en/Hacking/PinMapping
int segA = A1; //Display pin 14
int segB = 3; //Display pin 16
int segC = 4; //Display pin 13
int segD = 5; //Display pin 3
int segE = A0; //Display pin 5
int segF = 7; //Display pin 11
int segG = 8; //Display pin 15

void setup() {
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);

  pinMode(digit1, OUTPUT);
  pinMode(digit2, OUTPUT);
  pinMode(digit3, OUTPUT);
  pinMode(digit4, OUTPUT);

  pinMode(13, OUTPUT);
}

void loop() {

  //long startTime = millis();

  displayNumber(millis()/1000);

  //while( (millis() - startTime) < 2000) {
  //displayNumber(1217);
  //}
  //delay(1000);
```

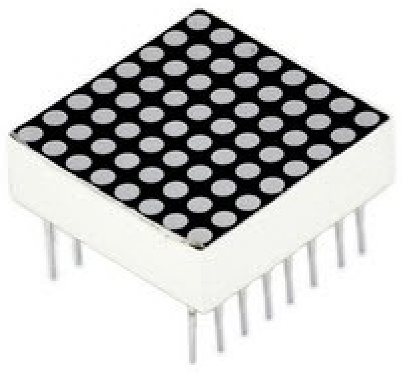
```
}

//Given a number, we display 10:22
//After running through the 4 numbers, the display is left turned off

//Display brightness
//Each digit is on for a certain amount of microseconds
//Then it is off until we have reached a total of 20ms for the function call
//Let's assume each digit is on for 1000us
//If each digit is on for 1ms, there are 4 digits, so the display is off for 16ms.
//That's a ratio of 1ms to 16ms or 6.25% on time (PWM).
//Let's define a variable called brightness that varies from:
//5000 blindingly bright (15.7mA current draw per digit)
//2000 shockingly bright (11.4mA current draw per digit)
//1000 pretty bright (5.9mA)
//500 normal (3mA)
//200 dim but readable (1.4mA)
//50 dim but readable (0.56mA)
//5 dim but readable (0.31mA)
//1 dim but readable in dark (0.28mA)

void displayNumber(int toDisplay) {
#define DISPLAY_BRIGHTNESS  500

#define DIGIT_ON  HIGH
#define DIGIT_OFF  LOW

  long beginTime = millis();

  for(int digit = 4 ; digit > 0 ; digit--) {

    //Turn on a digit for a short amount of time
    switch(digit) {
    case 1:
      digitalWrite(digit1, DIGIT_ON);
      break;
    case 2:
      digitalWrite(digit2, DIGIT_ON);
      break;
    case 3:
      digitalWrite(digit3, DIGIT_ON);
      break;
    case 4:
      digitalWrite(digit4, DIGIT_ON);
```

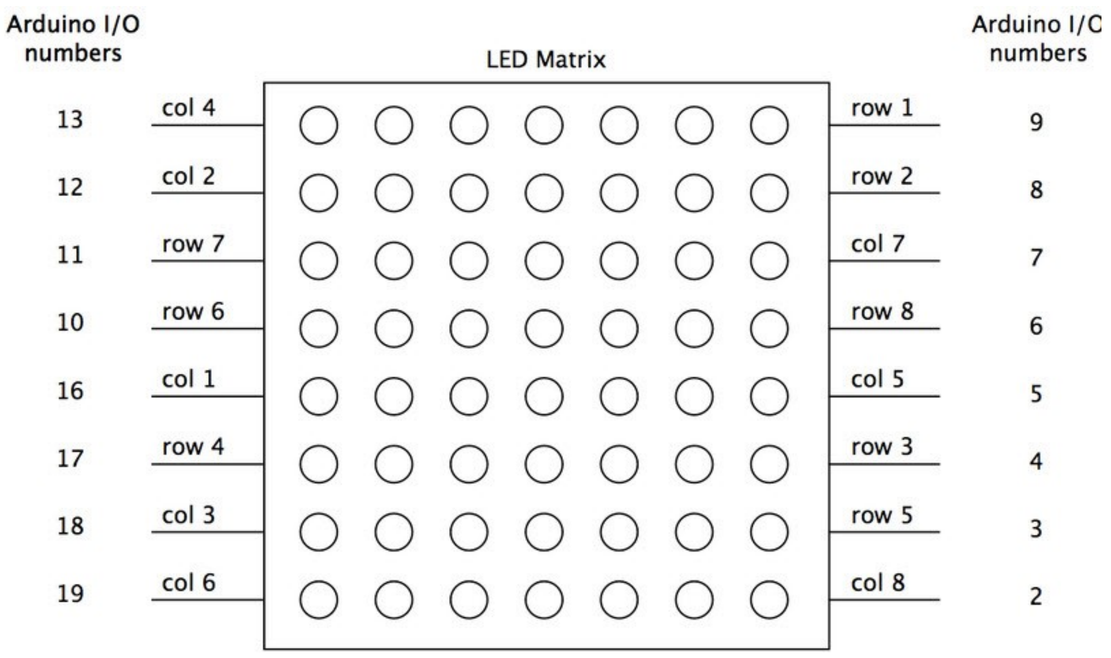
```
          break;
        }

        //Turn on the right segments for this digit
        lightNumber(toDisplay % 10);
        toDisplay /= 10;

        delayMicroseconds(DISPLAY_BRIGHTNESS); //Display this digit for a fraction of a
second (between 1us and 5000us, 500 is pretty good)

        //Turn off all segments
        lightNumber(10);

        //Turn off all digits
        digitalWrite(digit1, DIGIT_OFF);
        digitalWrite(digit2, DIGIT_OFF);
        digitalWrite(digit3, DIGIT_OFF);
        digitalWrite(digit4, DIGIT_OFF);
    }

    while( (millis() - beginTime) < 10) ; //Wait for 20ms to pass before we paint the
display again
}

//Given a number, turns on those segments
//If number == 10, then turn off number
void lightNumber(int numberToDisplay) {

#define SEGMENT_ON    LOW
#define SEGMENT_OFF HIGH

    switch (numberToDisplay){

    case 0:
        digitalWrite(segA, SEGMENT_ON);
        digitalWrite(segB, SEGMENT_ON);
        digitalWrite(segC, SEGMENT_ON);
        digitalWrite(segD, SEGMENT_ON);
        digitalWrite(segE, SEGMENT_ON);
        digitalWrite(segF, SEGMENT_ON);
        digitalWrite(segG, SEGMENT_OFF);
        break;

    case 1:
```

```
    digitalWrite(segA, SEGMENT_OFF);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_OFF);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_OFF);
    digitalWrite(segG, SEGMENT_OFF);
    break;

case 2:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_OFF);
    digitalWrite(segD, SEGMENT_ON);
    digitalWrite(segE, SEGMENT_ON);
    digitalWrite(segF, SEGMENT_OFF);
    digitalWrite(segG, SEGMENT_ON);
    break;

case 3:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_ON);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_OFF);
    digitalWrite(segG, SEGMENT_ON);
    break;

case 4:
    digitalWrite(segA, SEGMENT_OFF);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_OFF);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_ON);
    digitalWrite(segG, SEGMENT_ON);
    break;

case 5:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_OFF);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_ON);
```

```
        digitalWrite(segE, SEGMENT_OFF);
        digitalWrite(segF, SEGMENT_ON);
        digitalWrite(segG, SEGMENT_ON);
        break;

    case 6:
        digitalWrite(segA, SEGMENT_ON);
        digitalWrite(segB, SEGMENT_OFF);
        digitalWrite(segC, SEGMENT_ON);
        digitalWrite(segD, SEGMENT_ON);
        digitalWrite(segE, SEGMENT_ON);
        digitalWrite(segF, SEGMENT_ON);
        digitalWrite(segG, SEGMENT_ON);
        break;

    case 7:
        digitalWrite(segA, SEGMENT_ON);
        digitalWrite(segB, SEGMENT_ON);
        digitalWrite(segC, SEGMENT_ON);
        digitalWrite(segD, SEGMENT_OFF);
        digitalWrite(segE, SEGMENT_OFF);
        digitalWrite(segF, SEGMENT_OFF);
        digitalWrite(segG, SEGMENT_OFF);
        break;

    case 8:
        digitalWrite(segA, SEGMENT_ON);
        digitalWrite(segB, SEGMENT_ON);
        digitalWrite(segC, SEGMENT_ON);
        digitalWrite(segD, SEGMENT_ON);
        digitalWrite(segE, SEGMENT_ON);
        digitalWrite(segF, SEGMENT_ON);
        digitalWrite(segG, SEGMENT_ON);
        break;

    case 9:
        digitalWrite(segA, SEGMENT_ON);
        digitalWrite(segB, SEGMENT_ON);
        digitalWrite(segC, SEGMENT_ON);
        digitalWrite(segD, SEGMENT_ON);
        digitalWrite(segE, SEGMENT_OFF);
        digitalWrite(segF, SEGMENT_ON);
        digitalWrite(segG, SEGMENT_ON);
        break;
    case 10:
        digitalWrite(segA, SEGMENT_OFF);
        digitalWrite(segB, SEGMENT_OFF);
        digitalWrite(segC, SEGMENT_OFF);
        digitalWrite(segD, SEGMENT_OFF);
        digitalWrite(segE, SEGMENT_OFF);
        digitalWrite(segF, SEGMENT_OFF);
        digitalWrite(segG, SEGMENT_OFF);
        break;
    }
}
*******code End*******
```

18. Chapter15: 8*8 LED Matrix

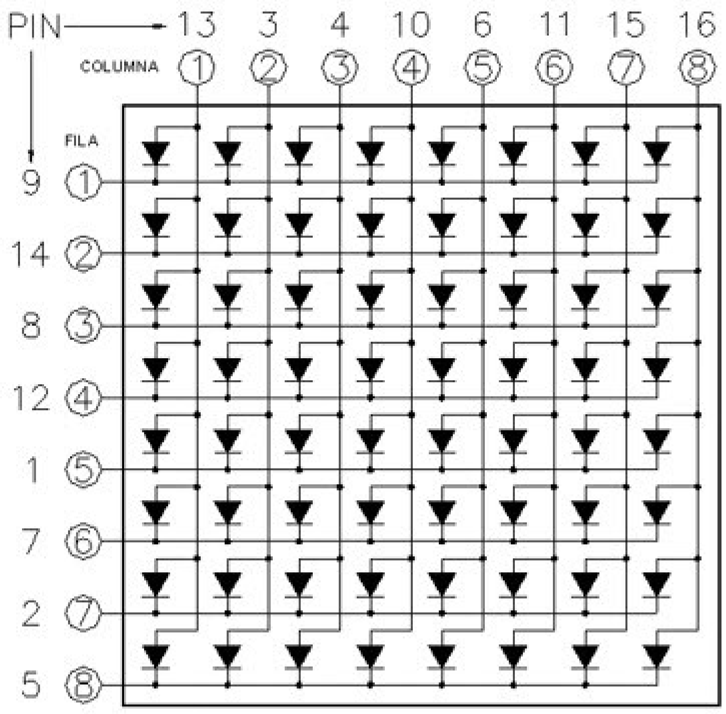




Introduction

With low-voltage scanning, LED dot-matrix displays have advantages such as power saving, long service life, low cost, high brightness, wide angle of view, long visual range, waterproof, and numerous specifications. LED dot-matrix displays can meet the needs of different applications and thus have a broad development prospect. This time, we will conduct an LED dot-matrix experiment to experience its charm firsthand.

The 8*8 dot-matrix is made up of sixty-four LEDs, and each LED is placed at the cross point of a row and a column. When the electrical level of a certain row is 1 and the electrical level of a
certain column is 0, the corresponding LED will light up. If you want to light the LED on the first
dot, you should set pin 9 to high level and pin 13 to low level. If you want to light LEDs on the first row, you should set pin 9 to high level and pins 13, 3, 4, 10, 6, 11, 15 and 16 to low level. If you want to light the LEDs on the first column, set pin 13 to low level and pins 9, 14, 8, 12, 1, 7, 2 and 5 to high level.
The internal view of a dot-matrix is shown as follows:



Example Code

```
*******code Begin*******
// set an array to store character of "0"
unsigned char Text[]={0x00,0x1c,0x22,0x22,0x22,0x22,0x22,0x1c};
void Draw_point(unsigned char x,unsigned char y)// point drawing function
{ clear_();
    digitalWrite(x+2, HIGH);
    digitalWrite(y+10, LOW);
    delay(1);
}
void show_num(void)// display function, call point drawing function
{
  unsigned char i,j,data;
  for(i=0;i<8;i++)
  {
      data=Text[i];
      for(j=0;j<8;j++)
      {
          if(data & 0x01)Draw_point(j,i);
          data>>=1;
      }
  }
}
void setup(){
```

```
int i = 0 ;
for(i=2;i<18;i++)
{
    pinMode(i, OUTPUT);
  }
  clear_();
}
void loop()
{ show_num();
}
void clear_(void)// clear screen
{for(int i=2;i<10;i++)
  digitalWrite(i, LOW);
  for(int i=0;i<8;i++)
  digitalWrite(i+10, HIGH);
}
```

*******code End*******