Arduino Introduction

Overview

What is Arduino?

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.
Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicated with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.
The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

Feature

- Schematic design of the open source development interface free download, and also according to the needs of their own changes
- Download the program is simple and convenient.
- Simply with the sensor, a wide range of electronic components connection (such as: LED light, buzzer, keypad, photoresistor, etc.), make all sorts of interesting things.
- Using the high-speed micro-processing controller (ATMEGA328).
- The development of language and development environment is very simple, easy to understand, very suitable for beginners to learn.

Performance

- Digital I/O 0~13.
- Analog I/O 0~5.(R3 is 0~7)
- Support ISP download function.
- Input voltage: when connected to the USB without external power supply or external 5 v ~ 9 v dc voltage input. Output voltage: 5 V dc voltage output and 3.3 V dc voltage output and external power input.
- Atmel Atmega328 micro-processing controller. Because of its many supporters, the company has developed 32-bit MCU arduino platform support.
  Arduino size: width of 70 mm X high 54 mm.

Special Port

1. **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can

supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

2. **AREF.** Reference voltage for the analog inputs. Used with analogReference().

SainSmart UNO R3

What's UNO R3?

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Performance

Revision 3 is the last SainSmart UNO development board version.

Parameter

- 3.3V/5V Supply Voltage and IO Voltage can be switched at the same time.
- More 3.3V modules supported, such as Xbee module, Bluetooth module, RF module, GPRS module, GPS module, LCD5110 Backlight and so on, but the original version can only support 5V IO.
- Controller uses SMD MEGA328P-AU chip. Add A6/A7 port.
- 5V Electric current : 500MA
- 3.3V Electric current : 50MA
- Input Voltage: 7-12V

Improvement of R3

- Working voltage 3.3V/5V is optional.
- Arduino can only work at 5V voltage. When it comes to 3.3V Level module, IO can't be connected to it. The Level should be changed, like the SD card, Bluetooth module and so on.
- Sainsmart UNO R3 can work at 3.3V voltage by switching on the button. At this time, IO port is 3.3V and it can work with 3.3V Level module. (R3 can directly use the electronic building blocks on I / O port and elicit G, V, S)

SainSmart MEGA2560 R3

Description:

This is the new MEGA2560 R3. In addition to all the features of the previous board, the MEGA now uses an ATMega16U2 instead of the ATMega8U2 chip. This allows for faster transfer rates and more memory. No drivers needed for Linux or Mac (inf file for Windows is needed and included in the Arduino IDE), and the ability to have the Uno show up as a keyboard, mouse, joystick, etc.

The MEGA2560 R3 also adds SDA and SCL pins next to the AREF. In addition, there are two new pins placed near the RESET pin. One is the IOREF that allow the shields to adapt to the voltage provided from the board. The other is a not connected and is reserved for future purposes. The MEGA2560 R3 works with all existing shields but can adapt to new shields which use these additional pins.

Features:

*Microcontroller ATmega2560
*Operating Voltage 5V
*Input Voltage (recommended) 7-12V
*Input Voltage (limits) 6-20V
*Digital I/O Pins 54 (of which 15 provide PWM output)
*Analog Input Pins 16
*DC Current per I/O Pin 40 mA
*DC Current for 3.3V Pin 50 mA
*Flash Memory 256 KB of which 8 KB used by bootloader
*SRAM 8 KB
*EEPROM 4 KB
*Clock Speed 16 MHz

SainSmart Leonardo R3

Overview:

The SainSmart Leonardo R3 is a microcontroller development board based on the ATmega32U4.
It has:
20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs),
a 16 MHz crystal oscillator,
a micro USB connection,
a power jack, an ICSP header,
a reset button.
It provides everything needed to support the microcontroller; simply power it with a AC-to-DC adapter to get started.
ATmega32u4 has built-in USB communication, eliminating the need for a secondary USB processor. Leonardo can model as the mouse and keyboard, which greatly improves the applications

Specification:

Microcontroller	ATmega32U4
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Input Voltage (limits)	6-20V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

SainSmart Nano V3

Overview:

The Sain Smart Nano for Arduino is a surface mount breadboard embedded version with integrated USB.

It is a smallest, complete, and breadboard friendly.

It has everything that Diecimila/Duemilanove has (electrically) with more analog input pins and onboard +5V AREF jumper. Physically, it is missing power jack.

The Nano is automatically sensed and switched to the higher potential source of power so there is no need for the power select jumper. Nano's got the breadboard-ability of the Boarduino and the Mini+USB with smaller footprint than either, so users have more breadboard space. It's got a pin layout that works well with the Mini or the Basic Stamp (TX, RX, ATN, GND on one top, power and ground on the other).

This new version 3.0 comes with ATMEGA328 which offer more programming and data memory space. It is two layers. That make it easier to hack and more affordable. You end up paying less with Nano than Mini and USB combined! Specifications: Microcontroller Atmel 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 32 KB (of which 2KB used by bootloader) SRAM 2 KB EEPROM 1 KB Clock Speed 16 MHz Dimensions 0.70" x 1.70".

Features:

Automatic reset during program download

Power OK, blue LED, Green (TX), red (RX) and orange (L) LED Auto sensing/switching power input

Small mini-B USB for programming

Serial monitor ICSP header for direct program download

Standard 0.1" spacing DIP (breadboard friendly)

Manual reset switch Power: The Nano can be powered via the mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

Arduino C Grammar

Arduino grammar is built on the basis of C/C + +, in fact is also the basic C grammar, Arduino grammar not only put some related parameters Settings are function change, we have no need to understand his bottom, let us to know AVR micro control unit (MCU) friend can also easy to fit in. So here I'll simple comment the Arduino grammar.

Control Structures

If
if...else
for
switch case
while
do... while
break
continue
return
goto

Further Syntax

;
{}
//
/* */

Operators

++
--
+=
-=
***=**
/=
=
+
-
*
/
%
==
!=
<
>
<=

>=
&&
||
!

Data type

boolean
char
byte
int
unsigned int
long
unsigned long
float
double
string
array
void

Constant

HIGH | LOW Said digital IO port level, HIGH Said high level(1), LOW Said low electric flat(0).

INPUT | OUTPUT Said digital IO port direction, INPUT Said input (high impedance state)

OUTPUT Said output (AVR can provide 5 v voltage and ma current).

TURE | FALSE true(1) , false(0）.

All above are the basic c grammar words and symbols, everybody can understand, and the specific use can combine experimental procedure.

Structure

- **void setup()**

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

- **void loop()**

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Function

- **Digital I/O**

pinMode(pin, mode) pin 0~13, mode is input or output.
digitalWrite(pin, value) pin 0~13, value is HIGH or LOW.
int digitalRead(pin) pin 0~13, value is HIGH or LOW.

- **Analog I/O**

int analogRead(pin) pin 0~5.
analogWrite(pin, value) pin 3, 5, 6, 9, 10, 11, value is 0 to 255

Time
delay(ms) Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)(unit ms).
delayMicroseconds(us)

Math
min(x, y) minimum value
max(x, y) maximum value
abs(x) absolute value
constrain(x, a, b) Constraint function, lower limit a upper limit b, x must be between a & b to be returned
map(value, fromLow, fromHigh, toLow, toHigh)
pow(base, exponent) extraction of square root
sq(x) square
sqrt(x) Square root

Chapter 1 Hello World!

In this chapter, we will learn use Arduino IDE serial interface tools to show the contents that we want to display in the computer.

Example code:

```
void setup()
{
      Serial.begin(9600);// opens serial port, sets data rate to 9600 bps
   Serial.println("Hello World!");
}

void loop()
{

}
```

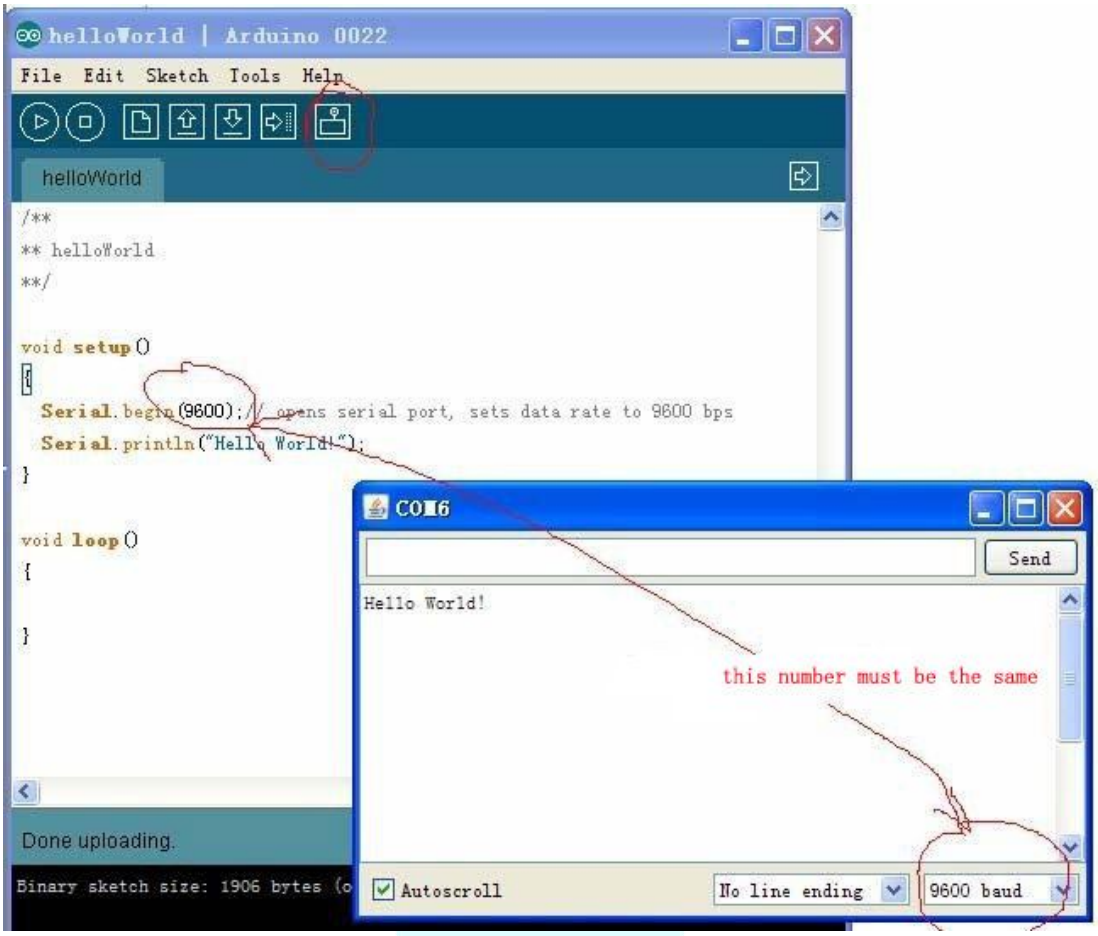
Explain:

Serial.begin(9600); The comment says 9600 bps, and just so you know bps stands for bits-per-second (we will refer to this as the baud rate). Communicate with computer, you may choose these rate “300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200”.

Operation:

1) Download code to arduino.

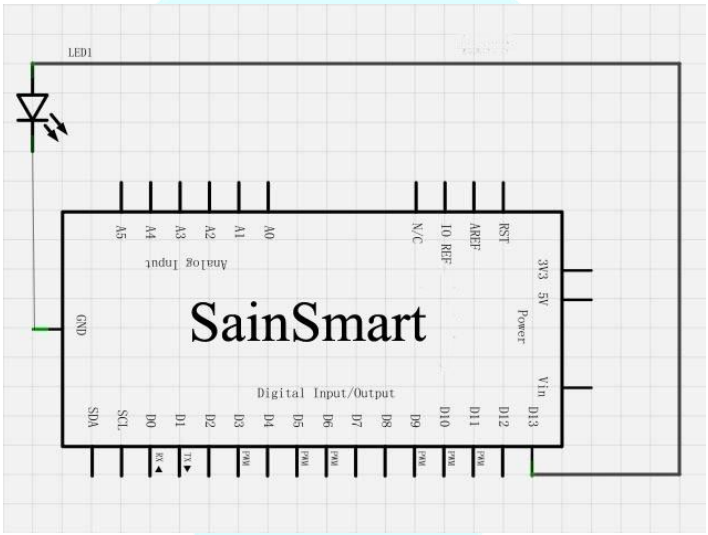
2) After download, click “tool”, pick up relevant arduino board, and relevant com. Then click “serial Monitor”, on the new open up window’s bottom right, choose the relevant rate.

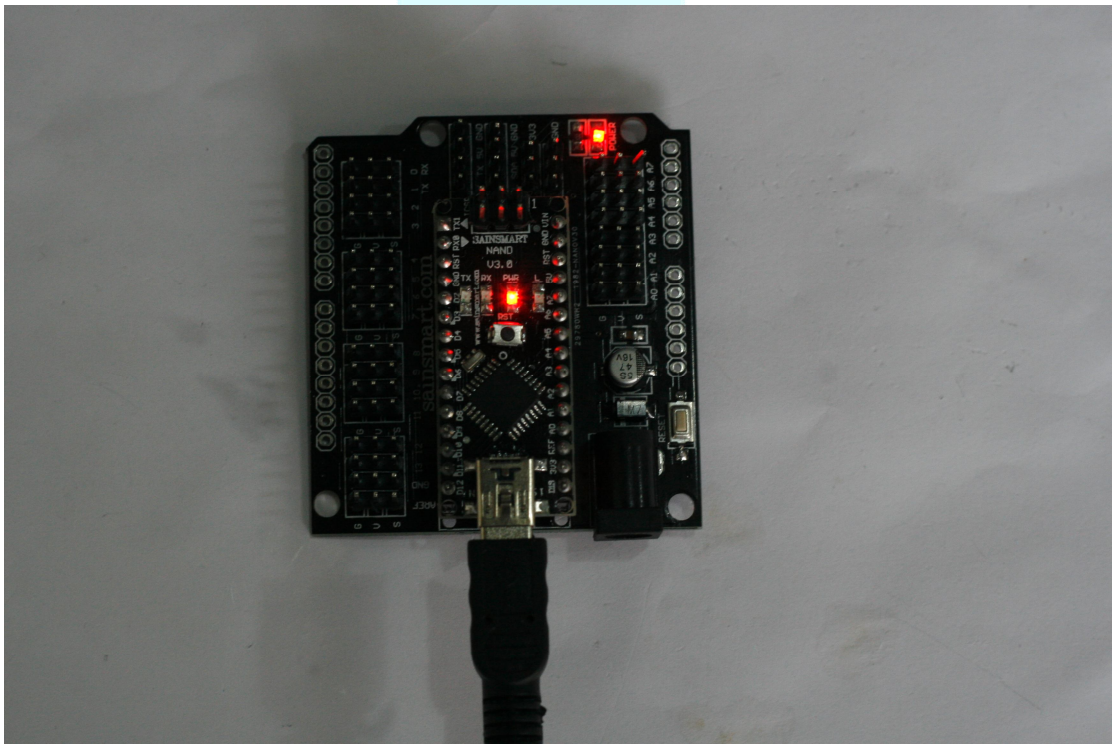


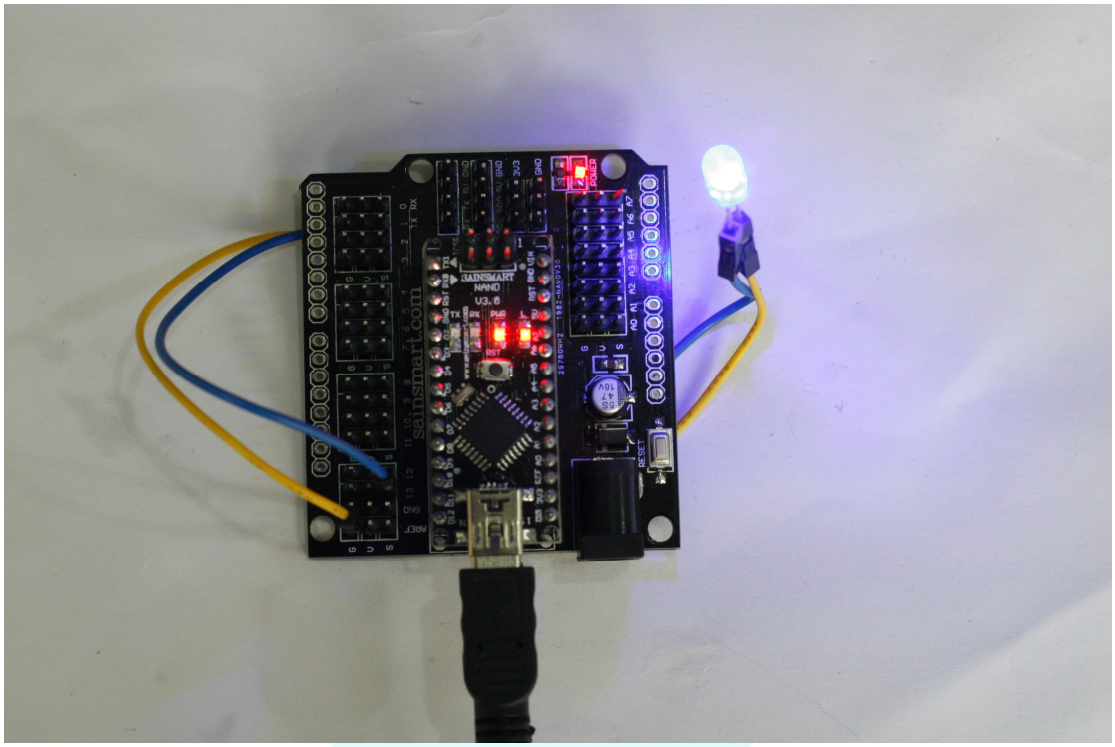
Chapter 2 Blink LED

Small LED lamp experiment is the basis of comparison of the experimental one, this time we use the motherboard comes with 13 feet of LED lights to complete the experiment, the experimental equipment we need is the Arduino which each experiment must have and USB download cable.

Next we connect small lamp in accordance with the following experimental schematic physical map.







Accordance with the good circuit after the link above figure, you can start writing programs, and we let the small LED lights flashing. Lighting on for one second and off for one second. This program is very simple. This is Arduino own routines Blink.

Example code:

```
int ledPin = 13; // define pin 13
void setup()
{
pinMode(ledPin, OUTPUT); // define interface is output
}
void loop()
{
digitalWrite(ledPin, HIGH); // light up led lamp
delay(1000); // delay 1s
digitalWrite(ledPin, LOW); // go out led lamp
delay(1000); // delay 1s
}
```

After downloading the program, you can see our 13-foot LED lights flashing, so that our small lights flicker experiment is complete.

Chapter3 LED Blink

light emitting diode

What's light emitting diode?

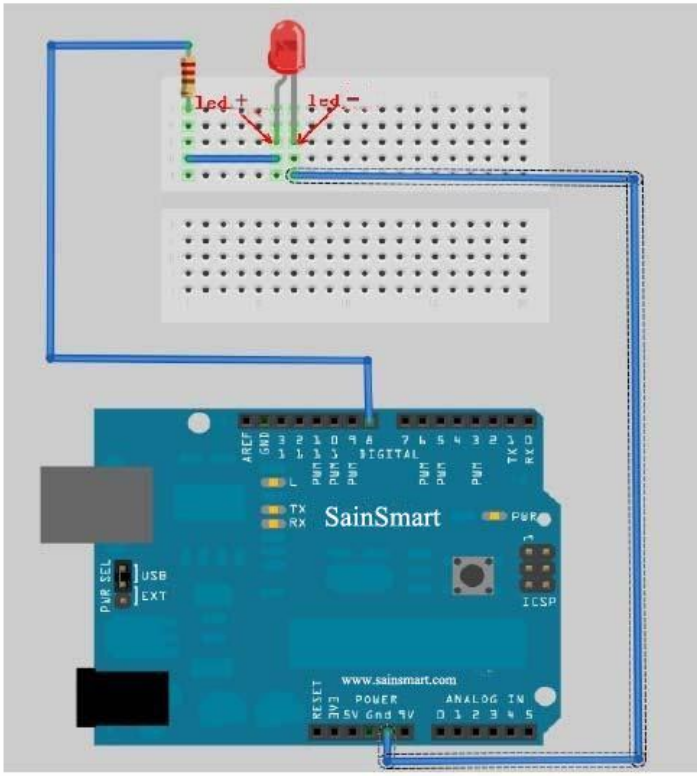
The light emitting diode referred to as LED. By gallium (Ga) and arsenic (AS) and phosphorus (P) made of a compound of the diode, when the electron and hole recombination can radiate visible light, and thus can be used to prepare a light-emitting diode in the circuit and the instrument as the indicator, or the composition of the text or digital display. Ga As P diode hair red, gallium phosphide diode green silicon carbide diode yellow.

A flashing LED lights experiment

Experiment component

- LED lamp : 1
- 220Ω resistor : 1
- Breadboard & Jumper wires

Connect your circuit as the below diagram



Example code:

```
int ledPin=8; //set IO pin of LED in control
void setup()
{
    pinMode(ledPin,OUTPUT);//set digital pin IO is OUTPUT
}
void loop()
{
    digitalWrite(ledPin,HIGH); //set PIN8 is HIGH , about 5V
    delay(1000); //delay 1000ms, 1000ms = 1s
    digitalWrite(ledPin,LOW); //set PIN8 is LOW, 0V
    delay(1000); //delay 1000ms, 1000ms = 1s
}
```

setup()

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

loop()

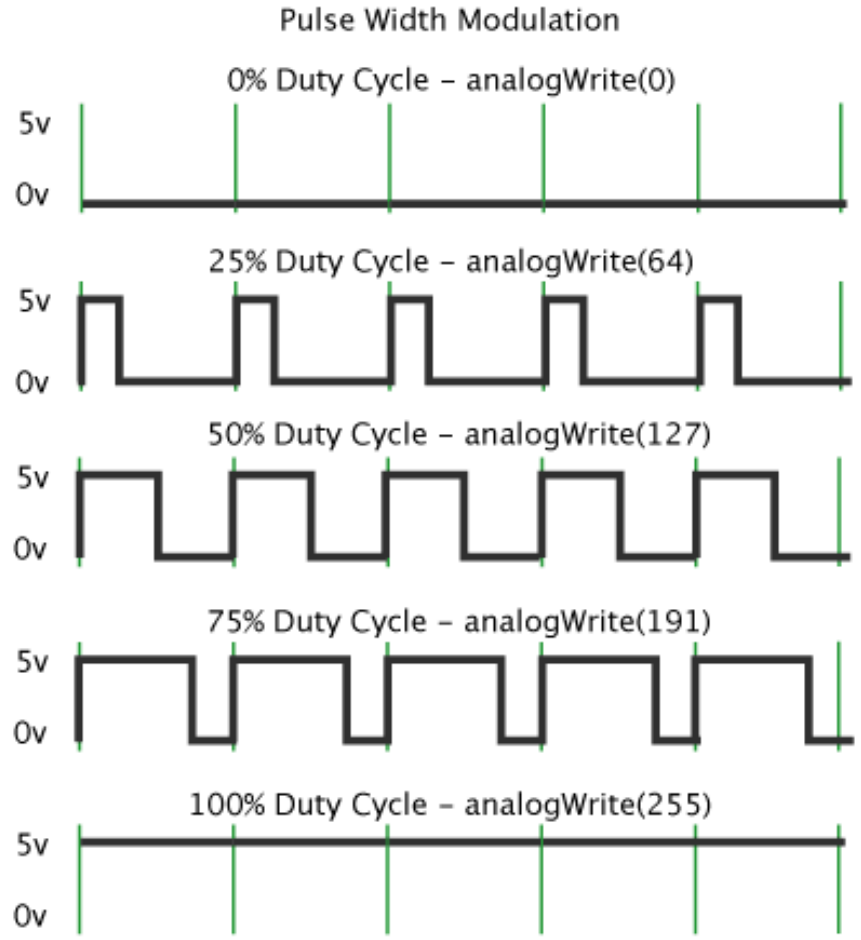
After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Chapter4 PWM

What's PWM?

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

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



For the Arduino, you write a value from 0 to 255 on a PWM pin, and the Arduino library will cause the pin to output a PWM signal whose on time is in proportion to the value written.

When it comes time for us to actually write an output voltage, the 0-255 value lacks meaning. What we want is many cases is a voltage. For our purposes, we will assume the Arduino is

running at Vcc = 5 volts. In that case, a value of 255 will also be 5 volts. We can then easily convert the desired voltage to the digital value needed using simple division. We first divide the voltage we want by the 5 volts maximum. That gives us the percentage of our PWM signal. We then multiply this percentage by 255 to give us our pin value. Here is the formula:
Pin Value (0-255) = 255 * (AnalogWrite / 5);

Arduino use analogWrite()

analogWrite()： Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). The frequency of the PWM signal is approximately 490 Hz.
On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 through 13. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11.The Arduino Due supports analogWrite() on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.You do not need to call pinMode() to set the pin as an output before calling analogWrite().The analogWrite function has nothing to do with the analog pins or the analogRead function.

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

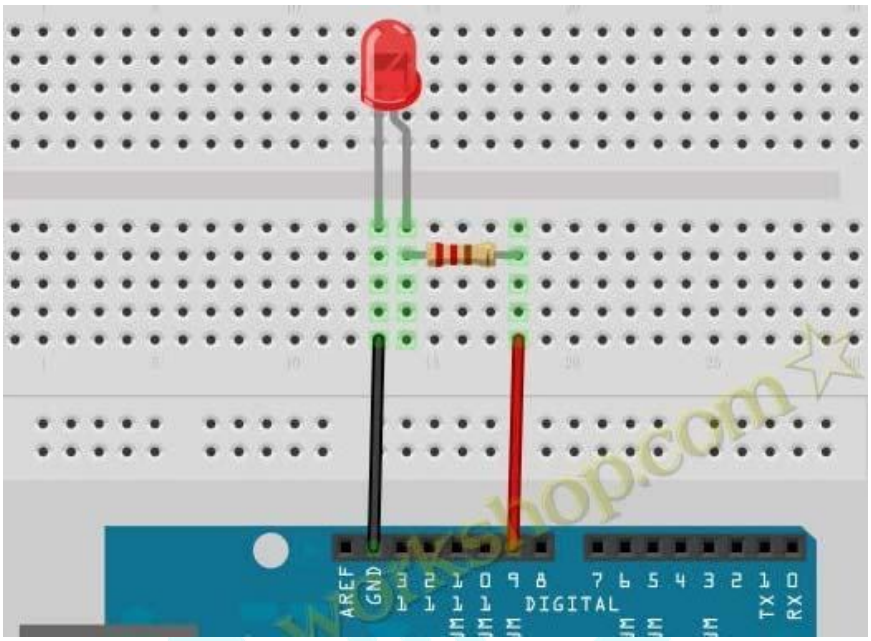
value: the duty cycle: between 0 (always off) and 255 (always on).
Notes and Known Issues

The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g 0 - 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6.

Experiment component:

1. 1 x 22oΩ resistor
2. 1 x LED
3. 1 x Breadboard

Connect your circuit as the below diagram.



Example code:

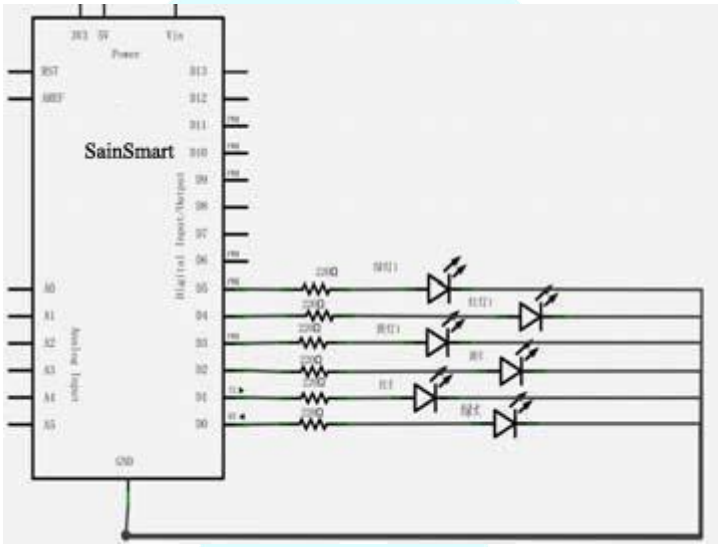
```
int brightness = 0;        //define original value of brightness, the value is brightness of LED.
int fadeAmount = 5;      //define fadeAmount，the value is the amount of brightness variations' change.

void setup()    {
   pinMode(9, OUTPUT);//     set pin9 is output
}

void loop()    {

   analogWrite(9, brightness);//write the value of brightness in pin9

      brightness = brightness + fadeAmount;//change the value of brightness

      if (brightness == 0 || brightness == 255) {
          fadeAmount = -fadeAmount ; // roll over the brightness between the highest and lowest
      }

   delay(30); //delay 30ms
}
```

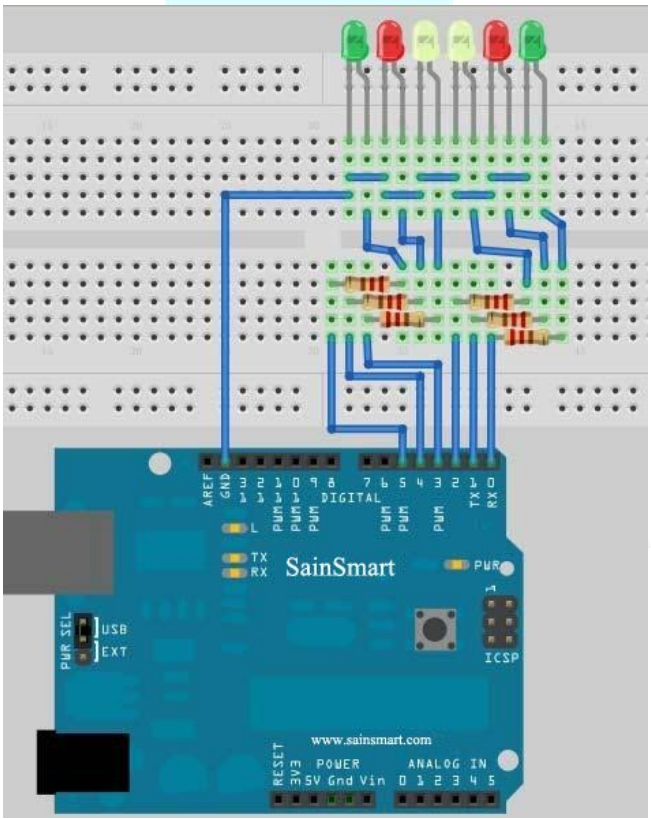
Chapter5 Advertising LED

Experiment component:

- LED lamp: 6
- 220Ω resistors: 6
- Breadboard & Jumper wires

Connect your circuit as the below diagram.





Example code

Program code is in the advertising lights program folder. Double-click to open and you will see a led2 folder, open it, you will find out a led2.pde file. Double-click the icon to open it. Then you will see that it is the arduino programming software window with the experimental program code.

```
//set in Led's digital IO pin control
int Led1 = 1;
int Led2 = 2;
int Led3 = 3;
int Led4 = 4;
int Led5 = 5;
int Led6 = 6;
//led lamp run the example 1 program
void style_1(void)
{
   unsigned char j;
   for(j=1;j<=6;j++)//every 200ms light up one of led lamps with 1~6 pin in turn
      {
      digitalWrite(j,HIGH);//light up the led lamps with j pin
            delay(200);//delay 200ms
      }
   for(j=6;j>=1;j--)//every 200ms got out one of led lamps with 6~1 pin in turn
      digitalWrite(j,LOW);//go out the led lamps with j pin
            delay(200);//delay 200ms
      }
}
//led lamp blink example program
void flash(void)
{
   unsigned char j,k;
for(k=0;k<=1;k++)//blink twice
      {
      for(j=1;j<=6;j++)//light up led lamps with 1~6 pin
         digitalWrite(j,HIGH);//light up led lamp with j pin
            delay(200);//delay 200ms
      for(j=1;j<=6;j++)//go out the led lamp with 1~6 pin
         digitalWrite(j,LOW);//go out the led lamp with j pin
            delay(200);//delay 200ms
      }
}
//led lamp run the example 2 program
void style_2(void)
{
   unsigned char j,k;
```

```
    k=1;//set k is 1
    for(j=3;j>=1;j--)
      {
      digitalWrite(j,HIGH);//light up
      digitalWrite(j+k,HIGH);//light up
            delay(400);//delay 400ms
            k +=2;//k plus 2
      }
    k=5;//set k is 5
    for(j=1;j<=3;j++)
      {
      digitalWrite(j,LOW);//go out
      digitalWrite(j+k,LOW);//go out
            delay(400);//delay 400ms
      k -=2;//k sub 2
      }
}
// led lamp run the example 3 program
void style_3(void)
{
    unsigned char j,k;//led lamp run the example 3 program
    k=5;//set k is 5
    for(j=1;j<=3;j++)
      {
      digitalWrite(j,HIGH);//light up
      digitalWrite(j+k,HIGH);//light up
            delay(400);//delay 400ms
      digitalWrite(j,LOW);//go out
      digitalWrite(j+k,LOW);//go out
      k -=2;//k sub 2
      }
         k=3;//set k is 3
         for(j=2;j>=1;j--)
            {
            digitalWrite(j,HIGH);//light up
            digitalWrite(j+k,HIGH);//light up
                  delay(400);//delay 400ms
            digitalWrite(j,LOW);//go out
            digitalWrite(j+k,LOW);//go out
                  k +=2;//k plus 2
            }
      }
      void setup()
      {
```

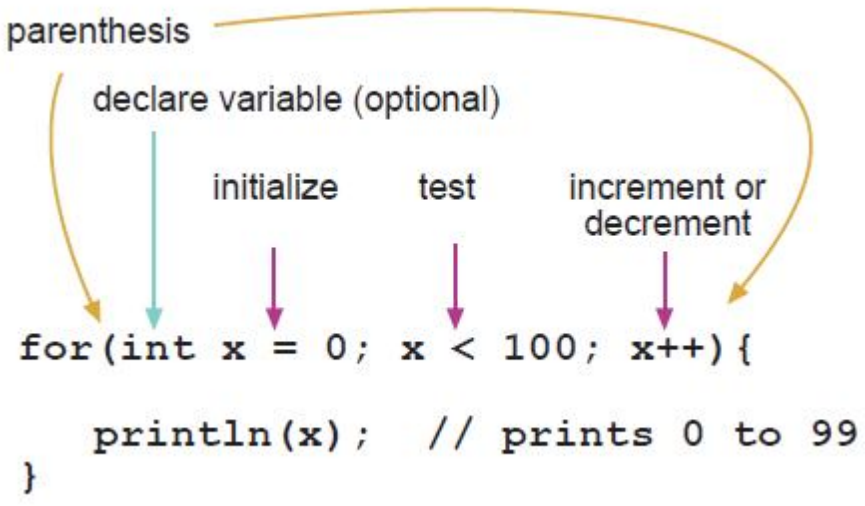
```
    unsigned char i;
    for(i=1;i<=6;i++)//set 1~6 pin output in turn
      pinMode(i,OUTPUT);//set i pin output
}
void loop()
{
    style_1();//example 1
    flash();//blink
    style_2();//example 2
    flash();//blink
    style_3();//example 3
    flash();//blink
}
```

Example code used: for(i=1;i<=6;i++)//set 1~6 pin output in turn
pinMode(i,OUTPUT);//set i pin output

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

There are three parts to the for loop header:

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



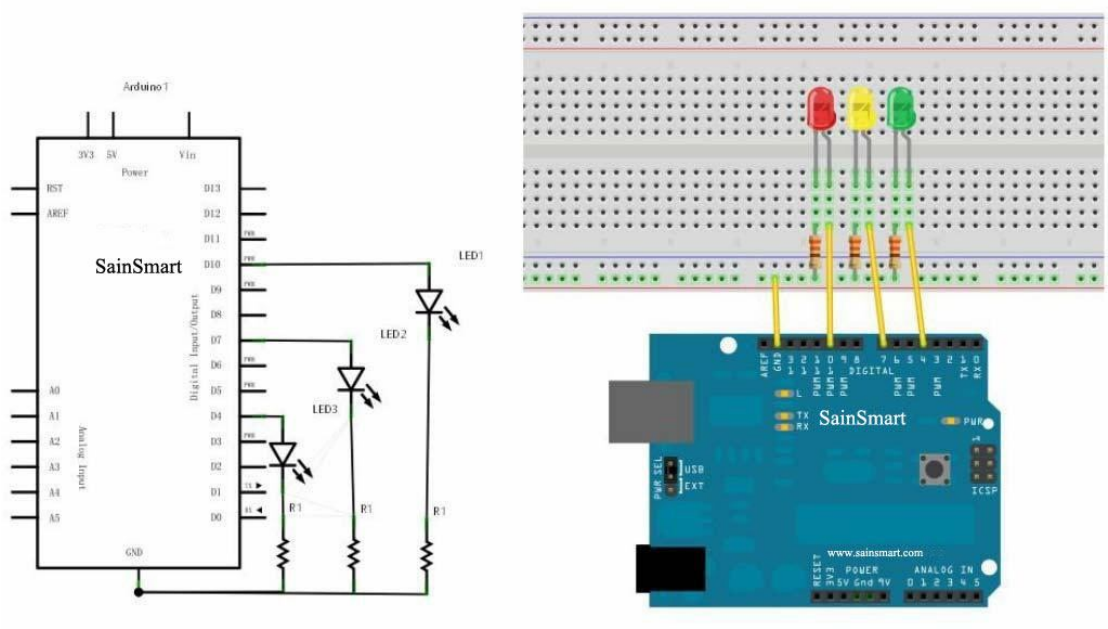
The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Chapter6 Traffic light

Experiment component:

- Red , Green , Yellow led lamp: 3
- 220Ω resistor: 3
- Breadboard & Jumper wires

Connect your circuit as the below diagram.

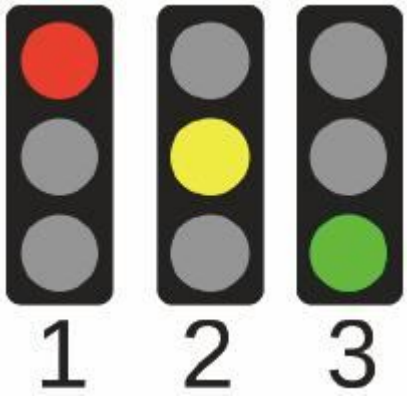


Example code:

Program code is in the traffic lights program folder. Double-click to open and you will find out a trafficLed.pde file. Double-click the icon to open it. Then you will see that it is the arduino programming software window with the experimental program code.

```
int ledred=10;      //define digital pin10 red
int ledyellow=7;       //define digital pin7 yellow
int ledgreen=4;     //define digital pin4 green
void setup()
{
    pinMode(ledred,OUTPUT);//set red pin output
    pinMode(ledyellow,OUTPUT);// set yellow pin output
    pinMode(ledgreen,OUTPUT);// set green pin output
}
void loop()
{
    digitalWrite(ledred,HIGH);//light up red lamp
    delay(1000);//delay 1000 ms = 1 s
    digitalWrite(ledred,LOW);//go out red lamp
```

```
    digitalWrite(ledyellow,HIGH);//light up yellow lamp
    delay(200);//delay 200 ms//
    digitalWrite(ledyellow,LOW);//go out
    digitalWrite(ledgreen,HIGH);//light up green lamp
    delay(1000);//delay 1000 ms
    digitalWrite(ledgreen,LOW);//go out
}
```

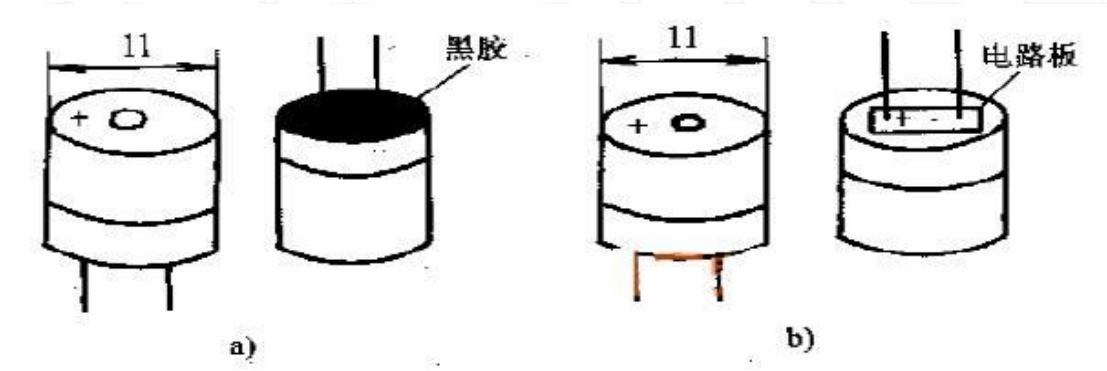


Chapter7 Buzzer

What's buzzer?

The buzzer is one integrated electronic transducers, DC voltage supply, widely used in computers, printers, copiers, alarm, electronic toys, automotive electronic equipment, telephones, timers and other electronic products for sound devices.

They can be divided into the: active buzzer (containing driver line) and passive buzzer (external drive) in their drive different way, teach you to distinguish between active buzzer and passive buzzer. A small buzzer for sale on the market now because of its small size (diameter is only 11mm), light weight, low price, solid structure, while widely used in various electrical equipment with sound, electronic production and microcontroller circuits. Appearance of active the buzzer and passive buzzer like a, b shown. a) active b) passive.

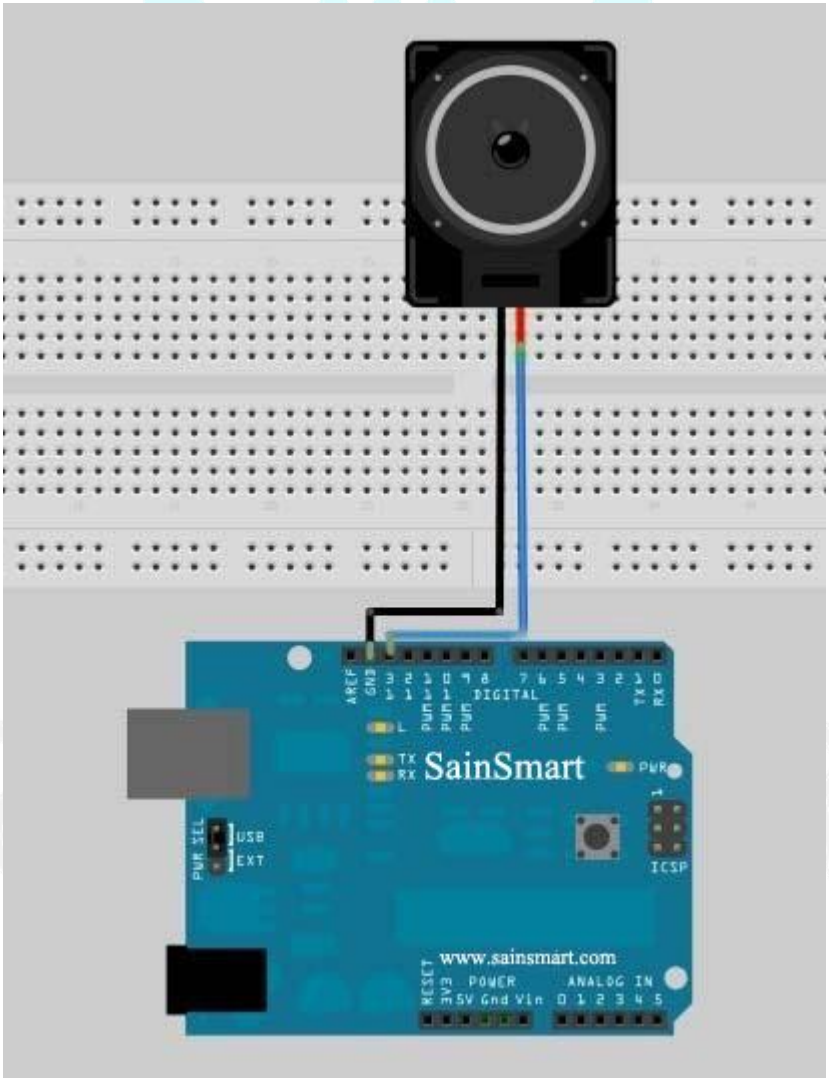


From the figure a, b appearance watching, the two buzzers seems to the same, but a closer look, the height of the two slight difference active buzzer a height of 9mm, passive buzzer b, a height

of 8 mm. As facing up to two buzzers' pin County it can be seen that there are a green circuit board is passive buzzer, no circuit board using vinyl enclosed one is active buzzer. Further determine the active and passive buzzer multimeter resistance profile Rxl file test: use a black pen touch buzzer's pin "+", red pen touch in the other pin back and forth, If you feel a click, cracking sound and resistance is only 8Ω
(Or 16Ω) which is a passive buzzer; continuing sound can issue, and the resistance is more than hundreds of Europe that is active buzzer. Active buzzer directly connected to the rated power (indicate on the new buzzer's label) can be continuous sound; rather passive buzzer and electromagnetic speaker needs to be connected to the audio output circuit can vocalization. Buzzer also can be divided into according to the constructed different,: the electromagnetic buzzer and piezoelectric buzzer;

Connect your circuit as the below diagram.

The buzzer used in this experiment with the internal drive. Circuit the buzzer positive connect directly into the digital port 13. GND socket connected to the negative terminal of the buzzer.



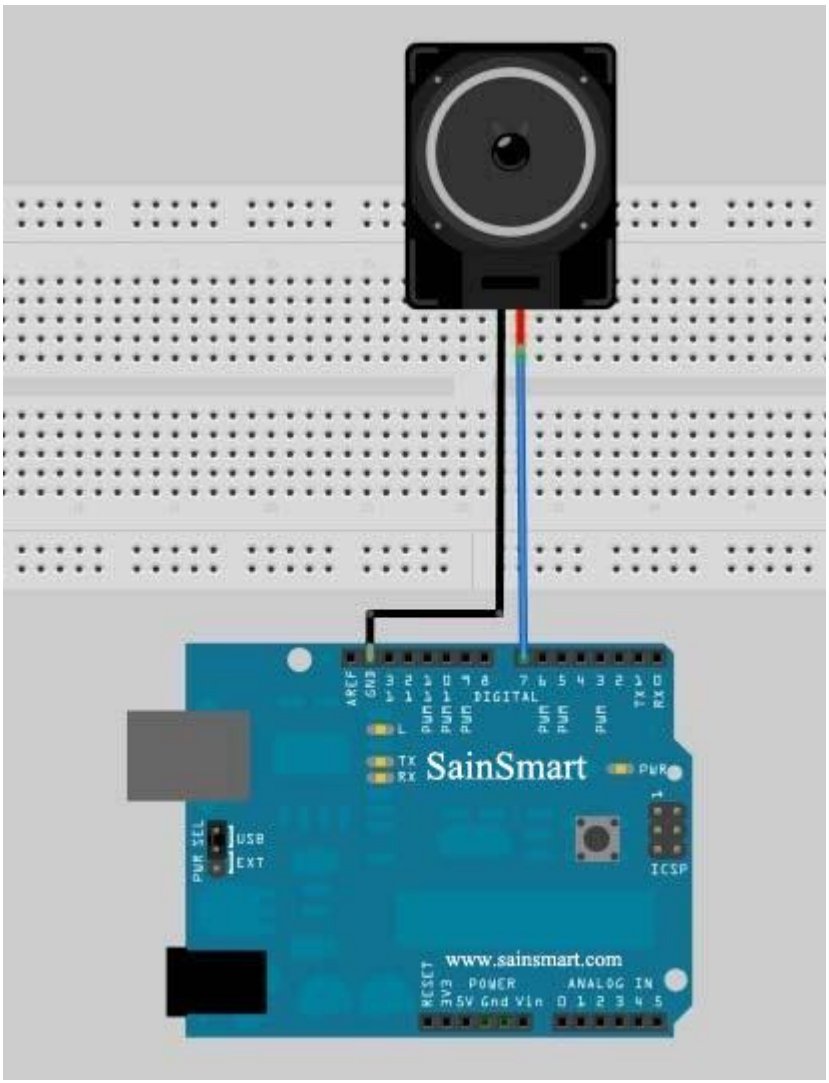
Buzzer analog ambulance siren sound experiment

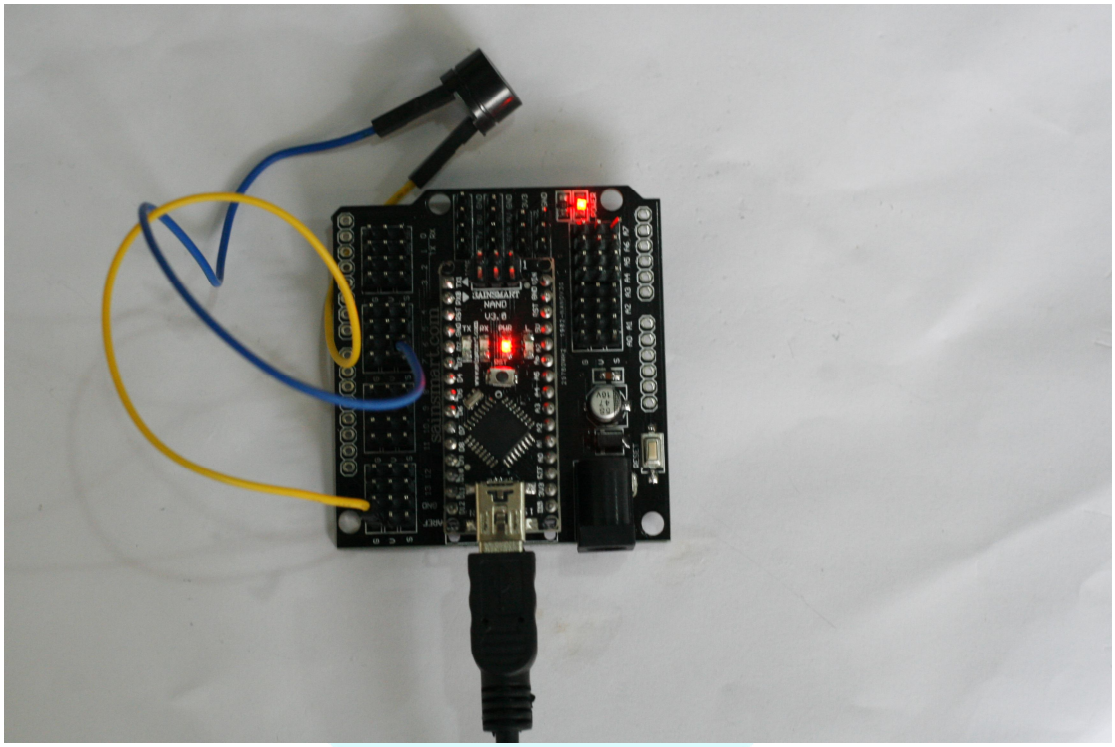
Experiment component:

Buzzer : 1

Breadboard & Jumper wires

Connect your circuit as the below diagram.





Example code

```
int buzzer=7;//set buzzer's digital pin IO in control
void setup()
{
   pinMode(buzzer,OUTPUT);//set digital pin IO OUTPUT
}
void loop()
{
   unsigned char i,j;//define i j
   while(1)
      {
      for(i=0;i<80;i++)// Output a frequency of sound
            {
         digitalWrite(buzzer,HIGH);//sound
         delay(1);//delay 1ms
         digitalWrite(buzzer,LOW);//mute
         delay(1);//delay 1ms
            }
      for(i=0;i<100;i++)// Output the other frequency of sound
            {
         digitalWrite(buzzer,HIGH);//sound
         delay(2);//delay 2ms
         digitalWrite(buzzer,LOW);//mute
         delay(2);//delay 2ms
            }
```

```
    }
}
```

while loops

Description

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while(expression){
  // statement(s)
}
```

Parameters

expression - a (boolean) C statement that evaluates to true or false

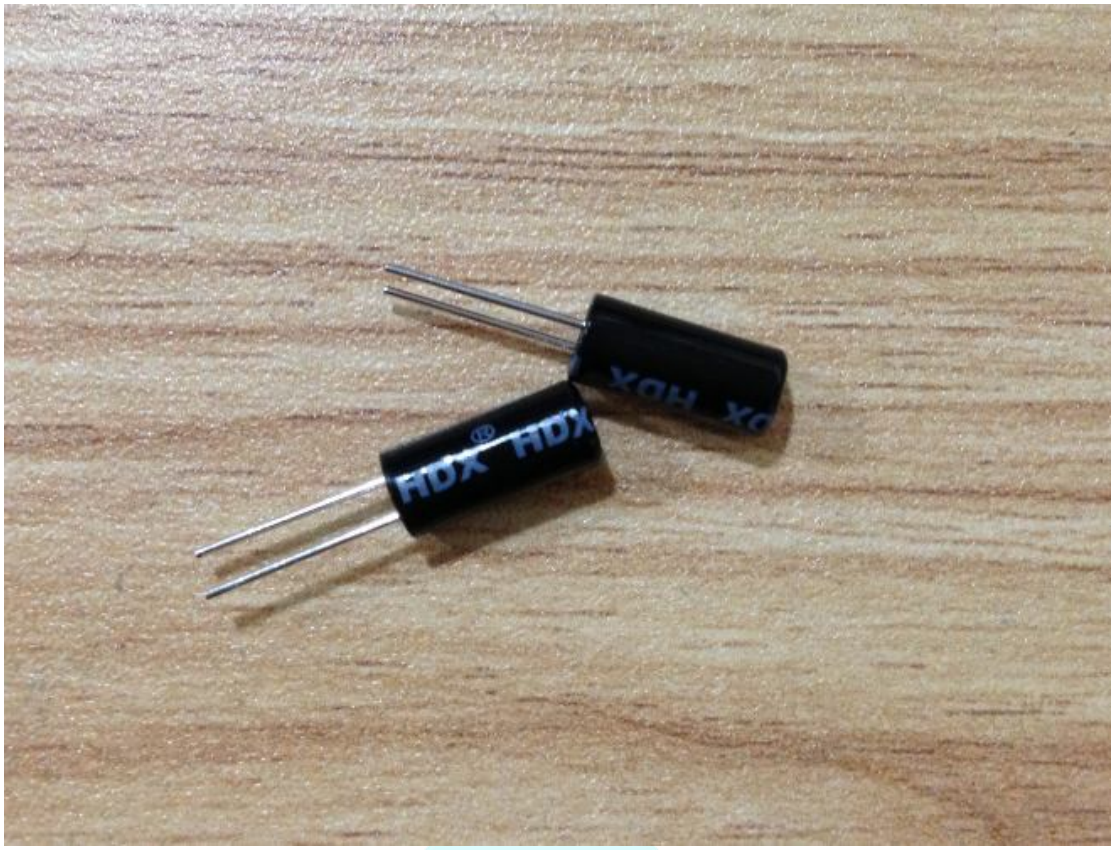
Example

```
var = 0;
while(var < 200){
  // do something repetitive 200 times
  var++;
}
```

Chapter8 Tilt switch

What's Tilt Sensor?

The tilt sensor is a component that can detect the tilting of an object. However it is only the equivalent to a pushbutton activated through a different physical mechanism. This type of sensor is the environmental-friendly version of a mercury-switch. It contains a metallic ball inside that will commute the two pins of the device from on to off and viceversa if the sensor reaches a certain angle.



Tilt switch controls led lamp light & out

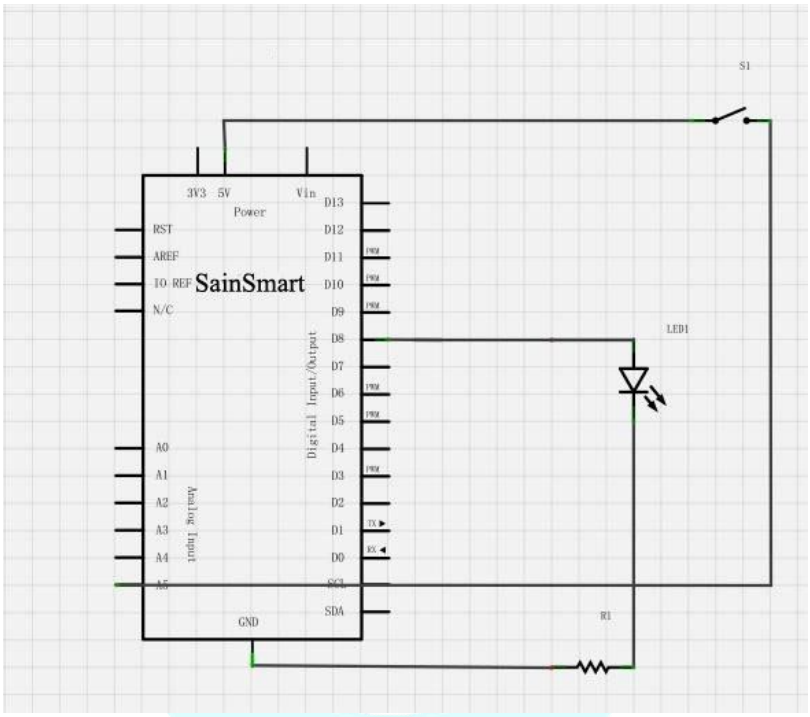
Experiment component:

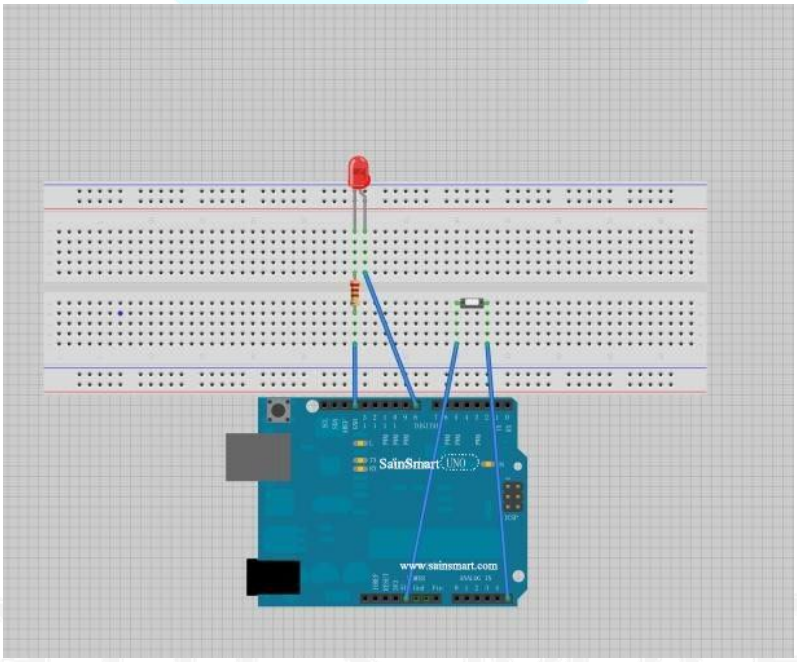
Tilt sensor : 1

Breadboard & Jumper wires

Connect your circuit as the below diagram.

Tilt switch connect to analog pin.





Example code

```
void setup()
{
      pinMode(8,OUTPUT);//set pin8 output
}
void loop()
{
      int i;//define i
            while(1)
```

```
        {
                i=analogRead(5);//read voltage values of pin5
                if(i>200)//if more than 512（2.5V）
                {
                        digitalWrite(8,HIGH);//light up led lamp
                }
                else
                {
                        digitalWrite(8,LOW);//go out led lamp
                }
        }
}
```

Chapter9 Potentiometer

What's Analog Pins?

1. A/D converter

The Atmega168 contains an onboard 6 channel analog-to-digital (A/D) converter. The converter has 10 bit resolution, returning integers from 0 to 1023. While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/output (GPIO) pins (the same as digital pins 0 - 13).
Consequently, if a user needs more general purpose input output pins, and all the analog pins are not in use, the analog pins may be used for GPIO.

2. Pin mapping

The Arduino pin numbers corresponding to the analog pins are 14 through 19. Note that these are Arduino pin numbers, and do not correspond to the physical pin numbers on the Atmega168 chip. The analog pins can be used identically to the digital pins, so for example, to set analog pin 0 to an output, and to set it HIGH, the code would look like this:

```
pinMode(14, OUTPUT);
digitalWrite(14, HIGH);
```

3. Pullup resistors

The analog pins also have pullup resistors, which work identically to pullup resistors on the digital pins. They are enabled by issuing a command such as

```
digitalWrite(14, HIGH);    // set pullup on analog pin 0
```

while the pin is an input.
Be aware however that turning on a pullup will affect the value reported by analogRead() when using some sensors if done inadvertently. Most users will want to use the pullup resistors only when using an analog pin in its digital mode.

4. Details and Caveats

The analogRead command will not work correctly if a pin has been previously set to an output, so if this is the case, set it back to an input before using analogRead. Similarly if the pin has been set to HIGH as an output, the pullup resistor will be on, after setting it back to an INPUT with pinMode.
The Atmega168 datasheet also cautions against switching digital pins in close temporal proximity to making A/D readings (analogRead) on other analog pins. This can cause electrical noise and introduce jitter in the analog system. It may be desirable, after manipulating analog pins (in digital mode), to add a short delay before using analogRead() to read other analog pins.

analogRead()

- **Description**

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

- **Syntax**

analogRead(pin)

- **Parameters**

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

- **Returns**

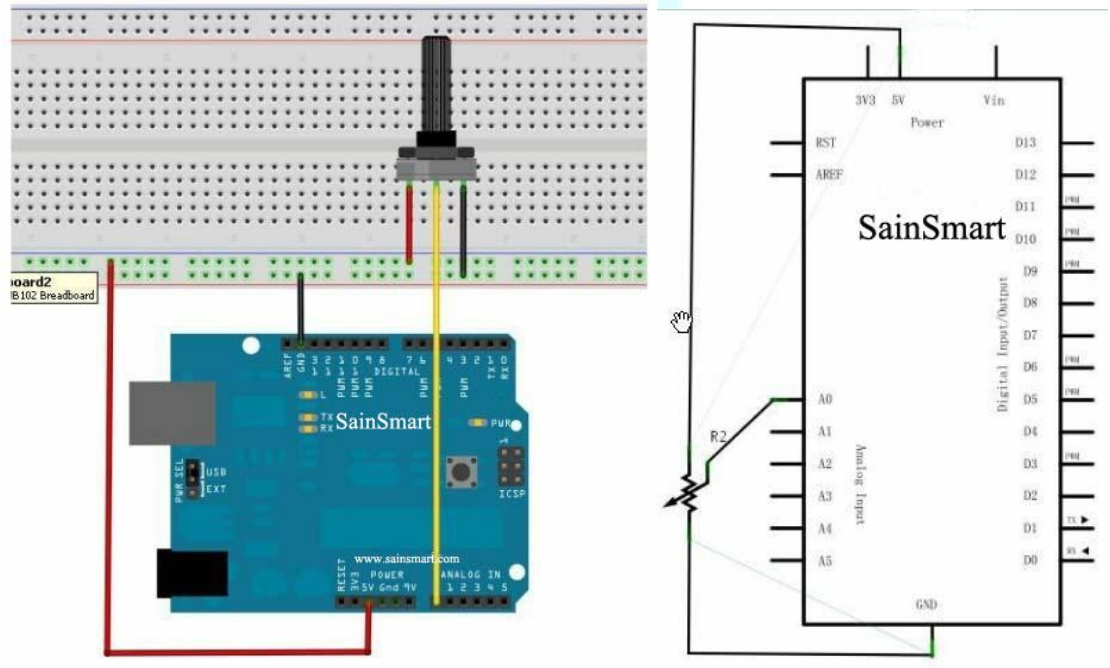
int (0 to 1023)

What's Potentiometer?

A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, that value controls the rate at which an LED blinks.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistence on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" that of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.



Example code:

```
int potpin = 0 ;      //define analog pin0
int ledpin = 13 ; //define analog pin13
int val = 0 ; //set val is0.

void setup()
{
      pinMode(ledpin,OUTPUT);//set analog pin13 output
      Serial.begin(9600);//set baud rate 9600
}
void loop()
{
      digitalWrite(ledpin,HIGH);//light up led in pin13
      delay(50);//delay 0.05s
      digitalWrite(ledpin,LOW);//go out led in pin13
      delay(50);//delay 0.05s
      val =    analogRead(potpin);//give the value of pin0 to val
      Serial.println(val) ; //print val's value
}
```

Chapter10 Photoresistor

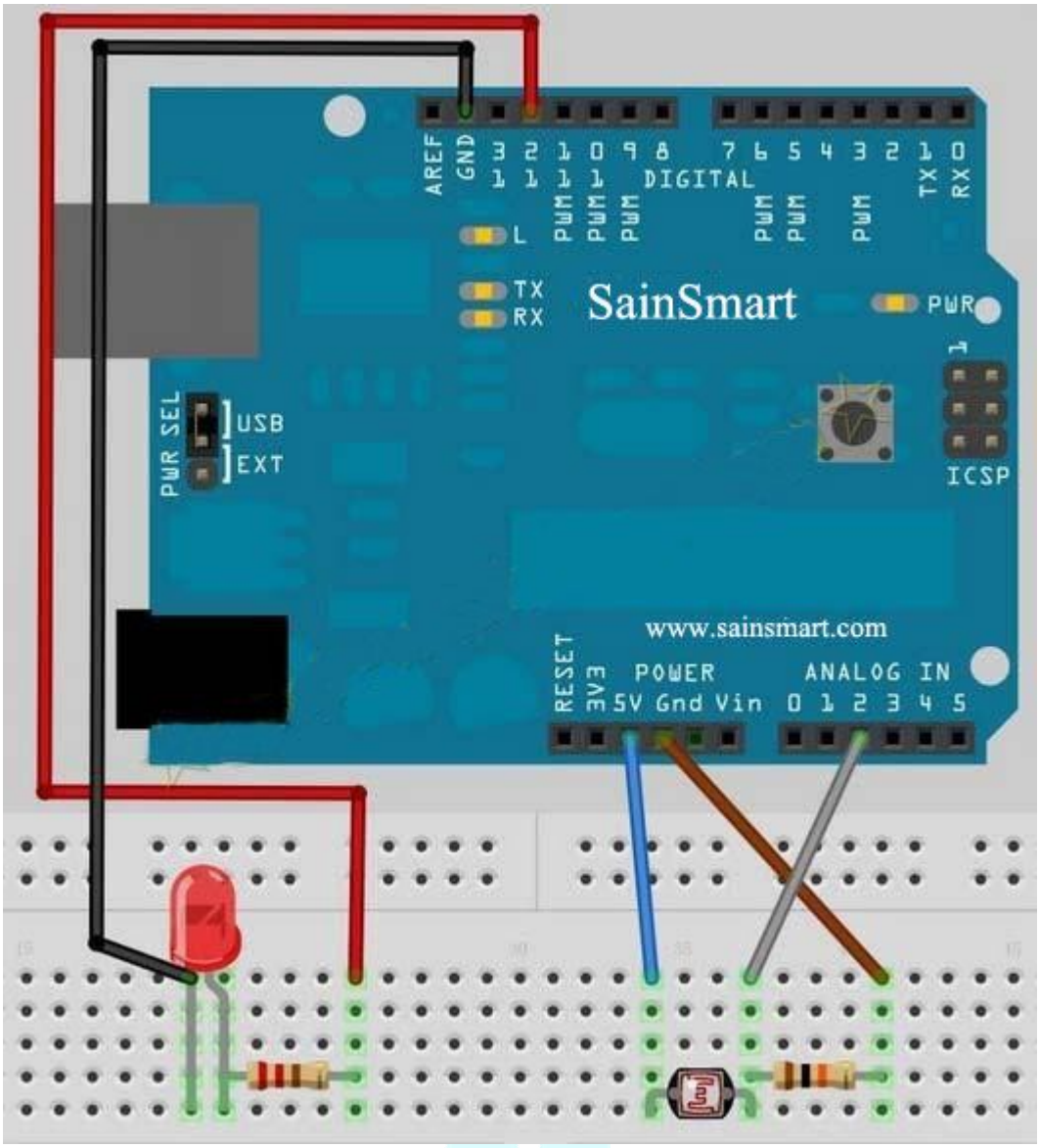
What's photoresistor?

Photoresistor, also known as light pipes, common production materials is cadmium sulfide, There are also selenium, aluminum sulfide, lead sulfide and bismuth sulfide material. these production materials having characteristics in light of a specific wavelength, its resistance decreases rapidly. This is due to the light generated carriers are involved in the electrical conductivity, under the applied electric field drift motion, so that the photosensitive resistor rapid decline.

Experiment component

- Photoresistor : 1
- Buzzer : 1
- 10K resistor : 1
- 220Ω resistor : 1
- Breadboard & Jumper wires

Connect your circuit as the below diagram.



Example code:

```
int photocellPin = 2;             //define photocellsh=2, read the value of voltage.
int ledPin = 12;          //define ledPin12 is the output port of led's level.
int val = 0;              //define original of val.



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

void loop() {
   val = analogRead(photocellPin);       //get the value from sensor
      if(val<=512){
//512=2.5V, if want the sensor be more sensitive, increase the number, or lese low the number.
```

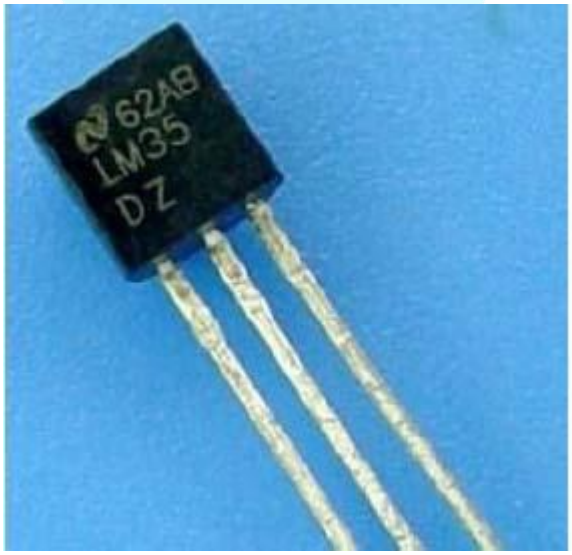
```
        digitalWrite(ledPin, HIGH); //when the value of val is less than 512(2.5V), light up led lamp
        }
        else{
        digitalWrite(ledPin, LOW);
        }
}
```

Chapter11 LM35 temperature sensor

Temperature sensor

What's temperature sensor?

The temperature sensor is that use substances of various physical properties with temperature variation of the sensor and let the temperature converted to electricity. These regularly change the physical properties of the main body temperature sensor is a core part of the temperature measuring instruments, and a wide variety. In accordance with the measurement method is divided into contact and non-contact two major categories, In accordance with the characteristics of sensor materials and electronic components into the thermal resistance and thermocouple.
Used in this experiment is the LM35 temperature sensor.

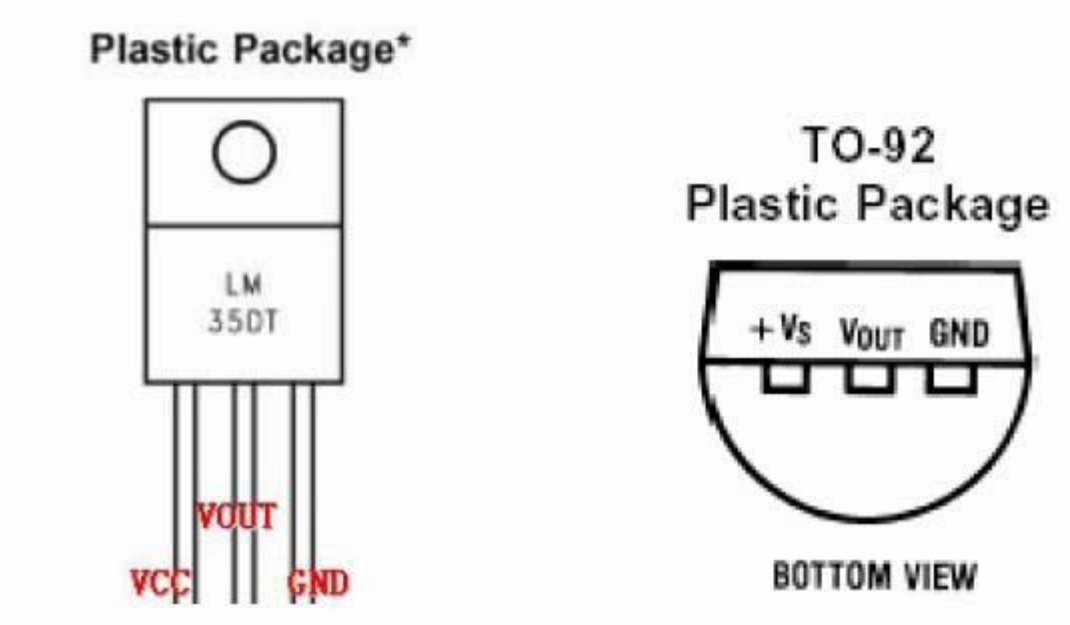


Working principle

LM35 temperature sensor output voltage linear relationship between the Celsius temperature scale，0 ℃, output is 0V, for every 1℃increases in output voltage of 10mV.

$$V_{\text{out\_LM35}}(T) = 10\,\text{mV}/^{\circ}\text{C} \times T^{\circ}\text{C}$$

LM35 pin diagram is as follows



Out can be seen from experimental cartridge of the temperature sensor, temperature sensor side is flat, and the other side is semicircular. Flat face of our own, the leftmost VCC pin (connected to +5 v), the middle of the GND pin VOUT (voltage value output pin, then the analog pins on the board), and the rightmost pin (connected board GND). Three pins, respectively, then you can use.

Temperature alarm experiment

Experiment component

- LM35 temperature sensor module*1
- Breadboard & jumper wire few

Connection

First ready experimental board; Follow the LM35 temperature sensor connection connected to VOUT is connected to an analog 0. Such temperature alarm experimental circuit connected.

Experimental principle

LM35 temperature sensor works shows that the temperature is increased by 1 ° C vout the mouth output voltage increases 10MV.

According to this principle procedures in real time reading out the analog voltage value of 0, since the analog port reads out a voltage value of 0 to 1023, i.e. 0V corresponding 0,5 V corresponds to 1023.

Application, we only need to LM35 module, analog interface, the read analog value is converted to the actual temperature.

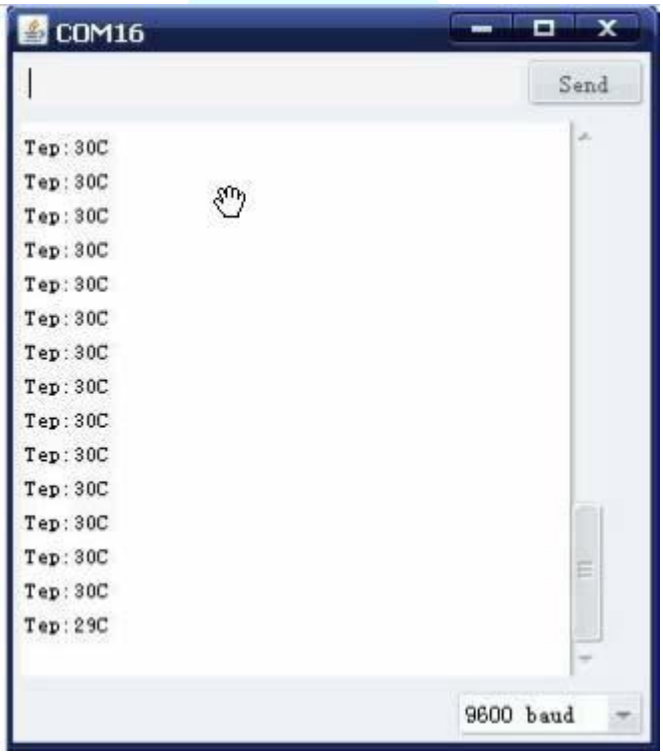
Example code

```
int potPin = 0 ;//define pin0 connect with LM35
void setup()
```

```
{
   Serial.begin(9600);
}
void loop()
{
   int val;
   int dat;

   val = analogRead(potPin);

      dat = (125*val)>>8 ; // Temperature calculation formula
      Serial.print("Tep : ") ; //print “Tep” means temperature
   Serial.print(dat) ; // print the value of dat
   Serial.println("C"); //print “C” means degree
   delay(500);//delay 0.5s
}
```

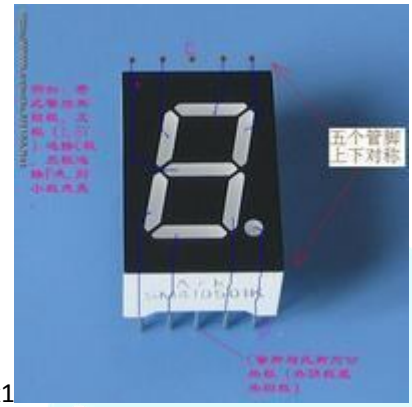
Program function

Download the program to the experimental board, open the monitor, you can see the current ambient temperature. (In fact, the temperature value a little deviation, according to the ambient temperature modify the program so that it is completely consistent with their own environment.)



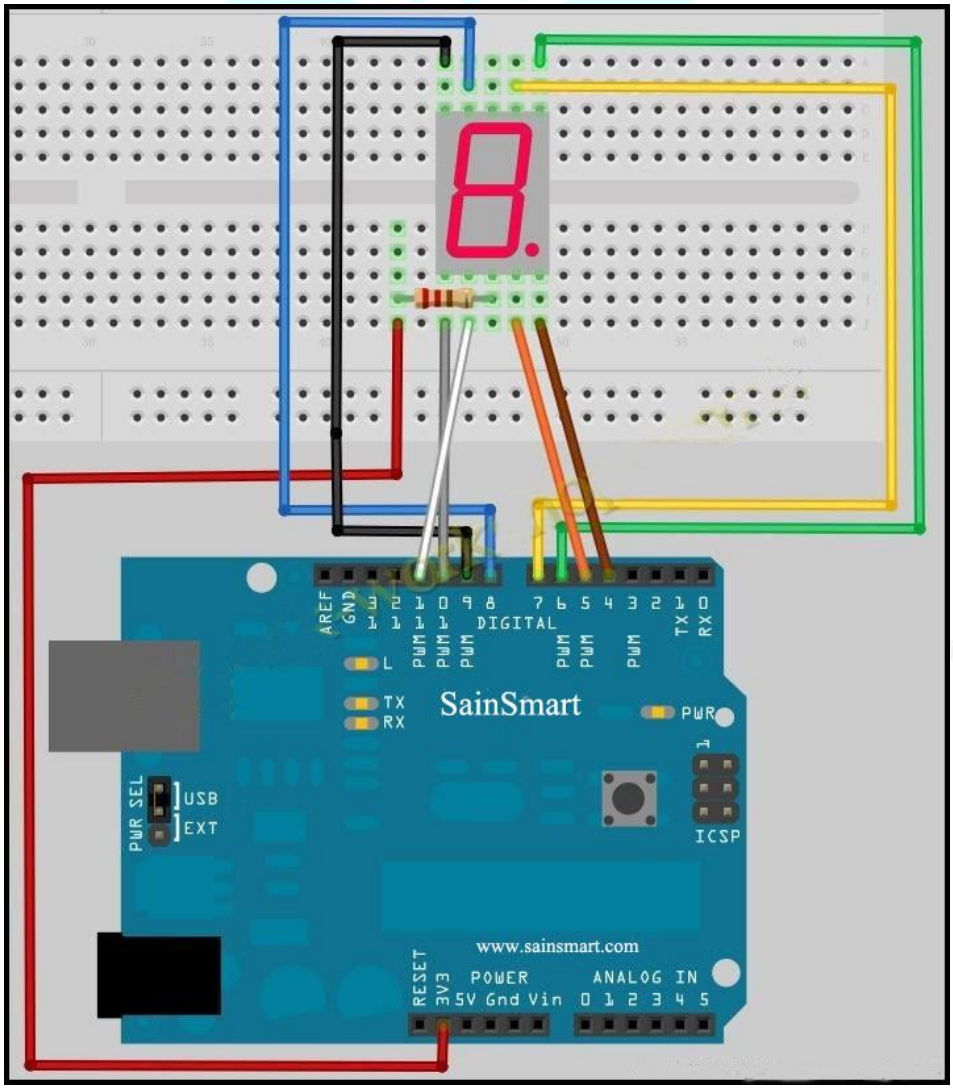
Chapter12 Nixie tube

Experiment component



- digital tube x1
- 220 Ω resistance x4
- Breadboard & jumper wire

Connect your circuit as the below diagram.



Example code

```
int a=7;
int b=6;
int c=5;
int d=11;
int e=10;
int f=8;
int g=9;
int dp=4;
//display number 1
void digital_1(void)
{
      unsigned char j;
      digitalWrite(c,LOW);// pin5 low, light up c
      digitalWrite(b,LOW);//light up b
      for(j=7;j<=11;j++)//go out else
            digitalWrite(j,HIGH);
      digitalWrite(dp,HIGH);//go out decimal point dp
}
//display number2
void digital_2(void)
{
      unsigned char j;
      digitalWrite(b,LOW);
      digitalWrite(a,LOW);
      for(j=9;j<=11;j++)
            digitalWrite(j,LOW);
      digitalWrite(dp,HIGH);
      digitalWrite(c,HIGH);
      digitalWrite(f,HIGH);
      }
// display number3
void digital_3(void)
{
      unsigned char j;
      digitalWrite(g,LOW);
      digitalWrite(d,LOW);
      for(j=5;j<=7;j++)
            digitalWrite(j,LOW);
      digitalWrite(dp,HIGH);
      digitalWrite(f,HIGH);
      digitalWrite(e,HIGH);
}
```

```
// display number4
void digital_4(void)
{
      digitalWrite(c,LOW);
      digitalWrite(b,LOW);
      digitalWrite(f,LOW);
      digitalWrite(g,LOW);
      digitalWrite(dp,HIGH);
      digitalWrite(a,HIGH);
      digitalWrite(e,HIGH);
      digitalWrite(d,HIGH);
}
// display number5
void digital_5(void)
{
      unsigned char j;
      for(j=7;j<=9;j++)
      digitalWrite(j,LOW);
      digitalWrite(c,LOW);
      digitalWrite(d,LOW);
      digitalWrite(dp,HIGH);
      digitalWrite(b,HIGH);
      digitalWrite(e,HIGH);
}
// display number6
void digital_6(void)
{
      unsigned char j;
      for(j=7;j<=11;j++)
            digitalWrite(j,LOW);
      digitalWrite(c,LOW);
      digitalWrite(dp,HIGH);
      digitalWrite(b,HIGH);
}
// display number7
void digital_7(void)
{
      unsigned char j;
      for(j=5;j<=7;j++)
            digitalWrite(j,LOW);
      digitalWrite(dp,HIGH);
      for(j=8;j<=11;j++)
            digitalWrite(j,HIGH);
}
```

```
// display number8
void digital_8(void)
{
      unsigned char j;
      for(j=5;j<=11;j++)
            digitalWrite(j,LOW);
      digitalWrite(dp,HIGH);
}
void setup()
{
      int i;//define i
      for(i=4;i<=11;i++)
            pinMode(i,OUTPUT);//set pin4~pin11 output
}
void loop()
{
      while(1)
      {
            digital_1();//number 1
            delay(2000);//delay 2s
            digital_2();
            delay(2000);
            digital_3();
            delay(2000);
            digital_4();
            delay(2000);
            digital_5();
            delay(2000);
            digital_6();
            delay(2000);
            digital_7();
            delay(2000);
            digital_8();
            delay(2000);
      }
}
```

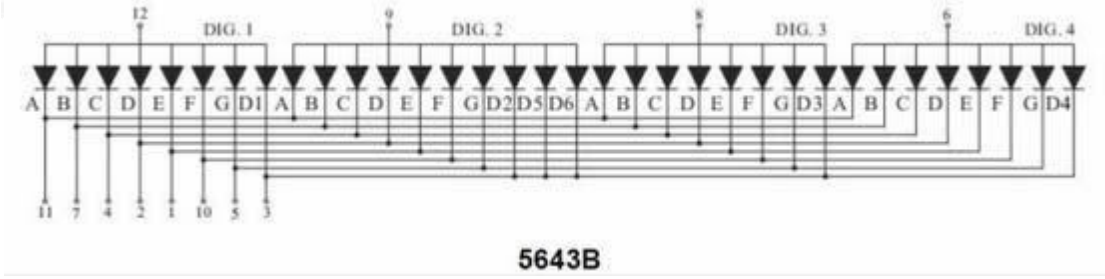
Chapter13 4-bit Nixie tube

What's digital tube?

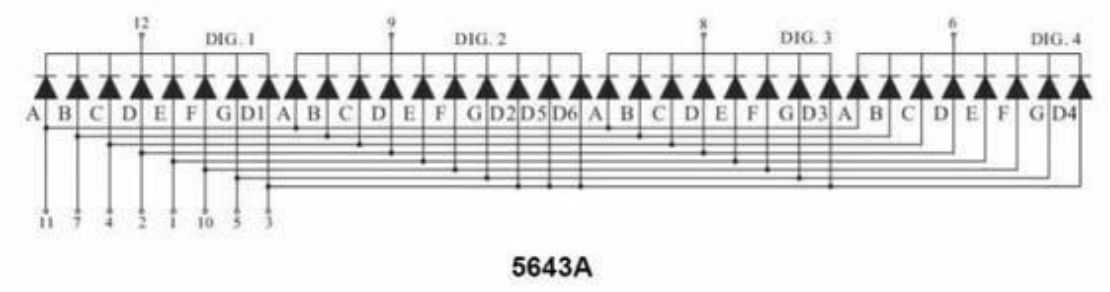
Digital tube is one kind semiconductor light emitting device. Their basic unit is light emitting diode. Digital tube is divided into 7 segment digital tube and 8 digital tube by the number of segments, 8 digital tube has one more light-emitting diode unit (a decimal point display) than 7 segment digital tube;

Digital tube has be divided into 1, 2, 4, and so on digital tube depend on how many "8" it can show.

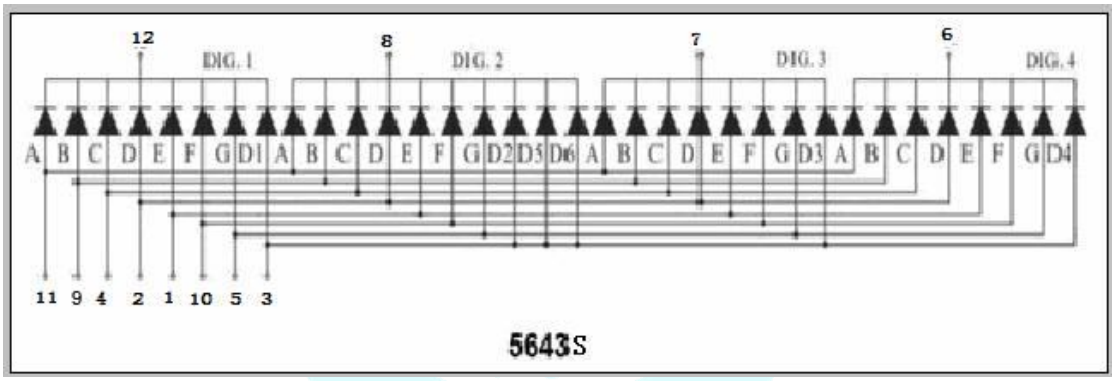
Digital tube is divided into common anode digital tube and common cathode digital tube by the connection of the light-emitting diode unit. The common anode digital tube is that connect light-emitting diode anode together to form a common anode (COM).Common anode digital tube public pole COM to +5 V, should be applied in light-emitting diode cathode when a field is low, the corresponding field lit. When a field of the cathode is high, the corresponding field is not bright.



5643B



5643A



5643S

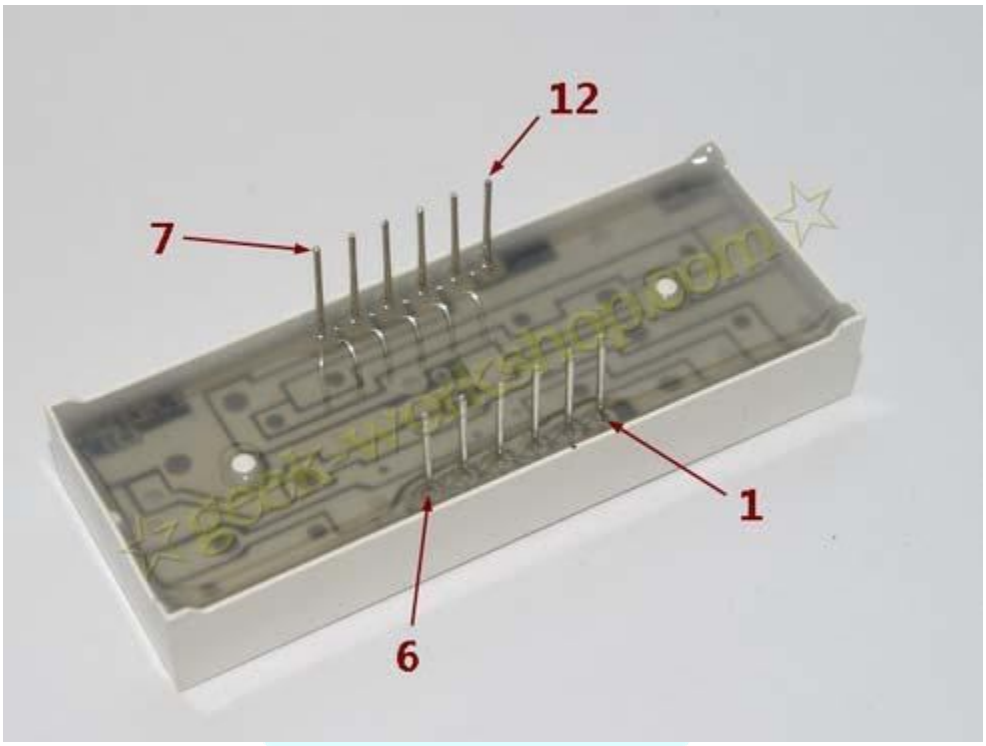
Working principle

Each segment of the digital tube is make up of the light emitting diode, and so when used with the light emitting diode, it should connect with the current-limiting resistor as well, if not the excessive current may burn light emitting diode.

The digital tube used in this experiment is a common anode common anode, The public pole COM received +5 V when the common anode Digital tube be applied. The corresponding fields are alight when a field emitting cathode of the diode is low, which are not bright when a field of the cathode is high.

Connection

One end of the current limiting resistor plugged into the digital I / O pin is connected to the other end of the not digitally tube field, the six remaining field and a decimal point followed by the return Ways to access. If public COM is common anode received a +5 V, else received a GND. There are a total 12 pin in one 4-bit digital tube.The decimal point downward when being placed in front of, lower left corner has 1-bit. The other pins' sequences are rotated counterclockwise. Upper left corner is the largest 12th pin.



Digital tube display number

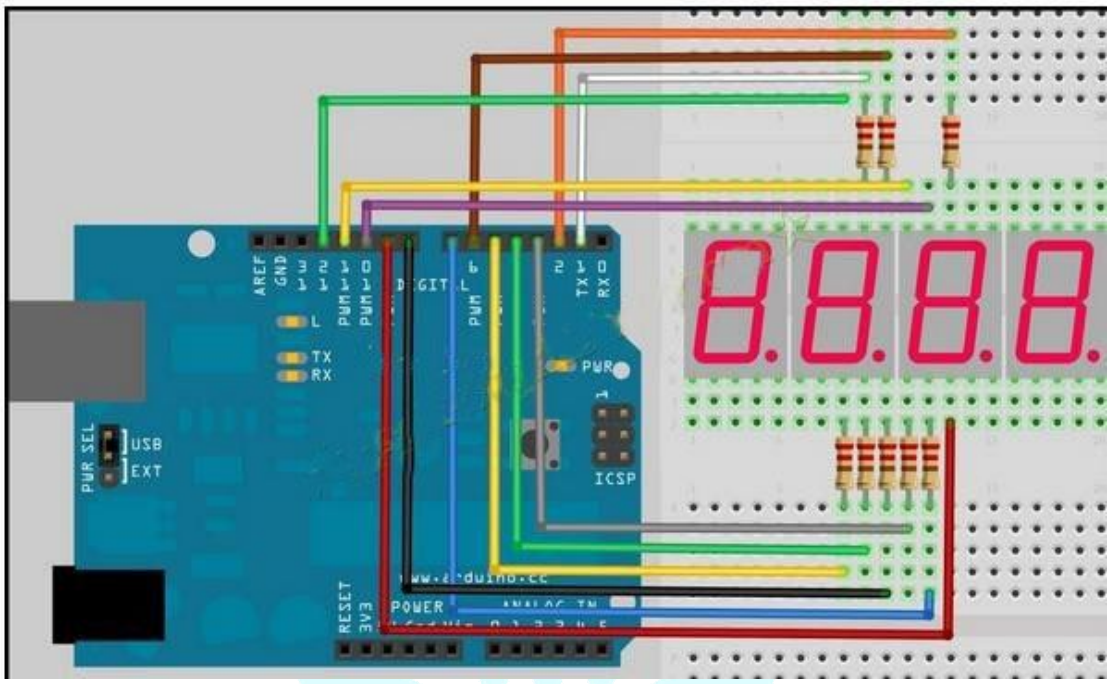
Experiment component

- 4-bit digital tube x1
- 220 Ω resistance x4
- Breadboard & jumper wire

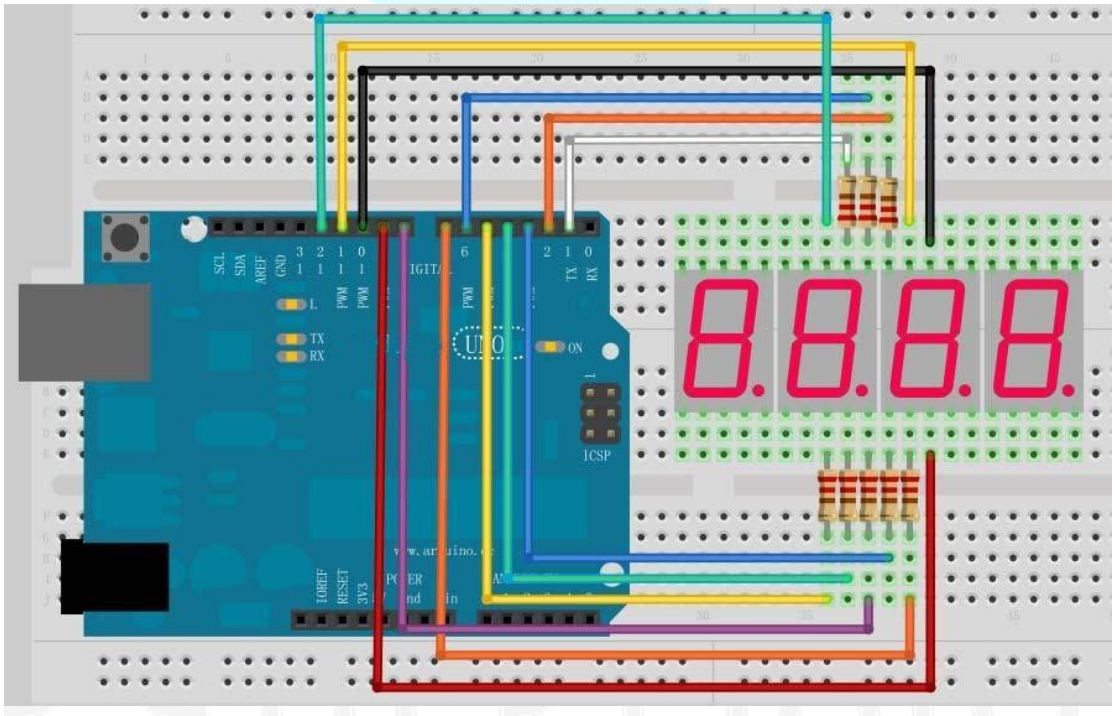
Connection

Driven digital tube current limiting resistor is certainly indispensable, there are 2 ways of limiting resistor connection. The first one is connected with D1-d4 anode, totally connect four. This connection method's benefit is needs of relatively less resistance, but generates different the digital brightness. The brightest is 1, 8 is the darkest. Another connection is use the other eight pins. The digital brightness of this method will be more like, but need more resistance. The experiments use eight 220Ω resistances.

Refer to figure below wiring for the 5643A



Refer to figure below wiring for the 5643S



Example code

This is a simple stopwatch. Its accuracy is not very high. You need to fine-tune the parameters.

```
//set anode interface
int a = 1;
int b = 2;
int c = 3;
int d = 4;
```

```
int e = 5;
int f = 6;
int g = 7;
int p = 8;
//set cathode interface
int d4 = 9;
int d3 = 10;
int d2 = 11;
int d1 = 12;
// Set variables
long n = 0;
int x = 100;
int del = 55;      // This number is fine-tuning of the clock
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(p, OUTPUT);
}

void loop()
{
   clearLEDs();
   pickDigit(1);
   pickNumber((n/x/1000)%10);
   delayMicroseconds(del);

   clearLEDs();
   pickDigit(2);
   pickNumber((n/x/100)%10);
   delayMicroseconds(del);

   clearLEDs();
   pickDigit(3);
   dispDec(3);
```

```
    pickNumber((n/x/10)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(4);
    pickNumber(n/x%10);
    delayMicroseconds(del);

    n++;

        if (digitalRead(13) == LOW)
        {
        n = 0;
        }
}

void pickDigit(int x)      //defing pickDigit(x), its role is turn on the dx port
{
    digitalWrite(d1, HIGH);
    digitalWrite(d2, HIGH);
    digitalWrite(d3, HIGH);
    digitalWrite(d4, HIGH);

    switch(x)
        {
        case 1:
        digitalWrite(d1, LOW);
        break;
        case 2:
        digitalWrite(d2, LOW);
        break;
        case 3:
        digitalWrite(d3, LOW);
        break;
    default:
        digitalWrite(d4, LOW);
        break;
        }
}

void pickNumber(int x)      //define pickNumber(x), Its role is to show digital x
{
    switch(x)
        {
```

```
    default:
        zero();
        break;
        case 1:
        one();
        break;
        case 2:
              two();
        break;
        case 3:
        three();
        break;
        case 4:
four();
        break;
        case 5:
        five();
        break;
        case 6:
              six();
        break;
        case 7:
              seven();
        break;
        case 8:
              eight();
        break;
        case 9:
        nine();
        break;
        }
}

void dispDec(int x)      // Set to open the decimal point
{
    digitalWrite(p, LOW);
}

void clearLEDs()    //clear the screen
{
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);
```

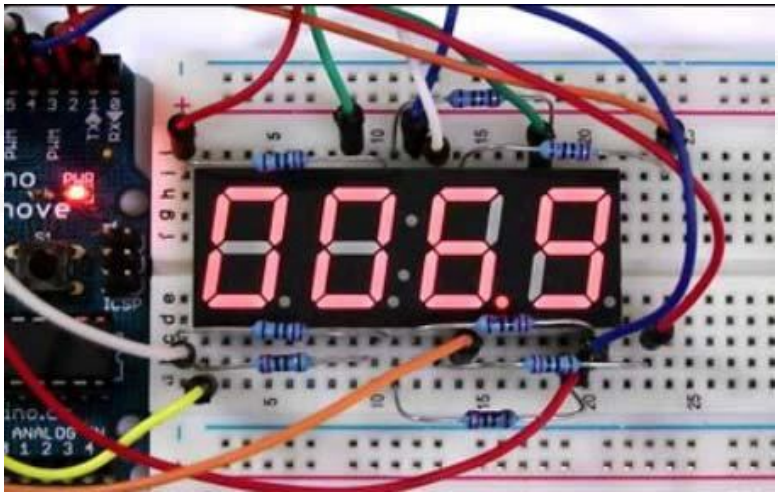
```
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    digitalWrite(p, LOW);
}

void zero()      // Define the number 0 cathode pin switch
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
digitalWrite(f, HIGH);
    digitalWrite(g, LOW);
}

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

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

void three()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
```

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

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

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

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

void seven()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
```

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

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

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

In front of setup () defined range of digital display routines, the definition of these subroutines can be easy to use in the loop (), just write the name of the subroutine and it will.

Program function

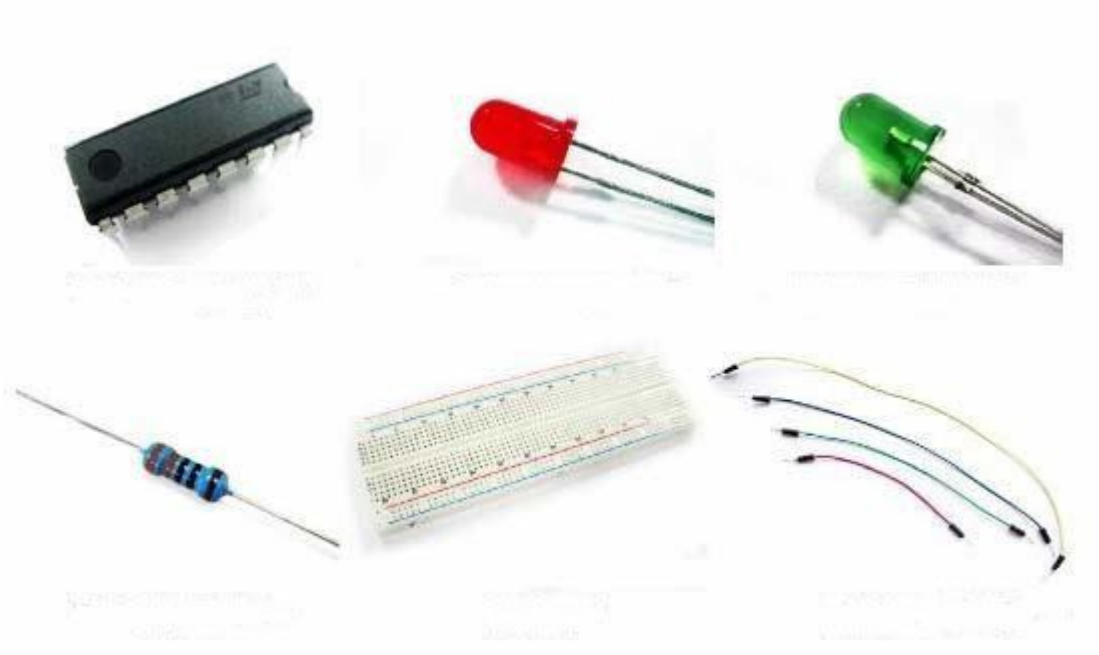


Chapter14 74HC595

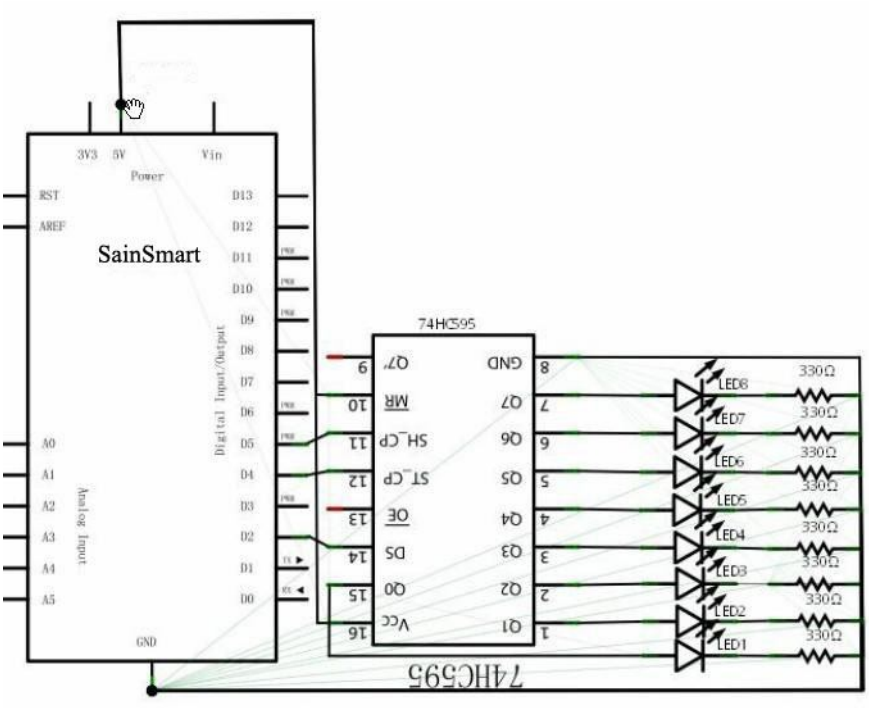
What's 74HC595?

74HC595 with 8-bit register and a memory, and has three-state output function. we use it to control 8 LED lights. Why do we choose 74HC595? If we control eight small lights just with Arduino, how many its I / O will be occupied? The answer is eight. However one arduino uno only have 20 I/O port. 8 small lights have take up too many resources. The purpose we use 74HC595 is to reduce the occupation of the number of I / O port. With 74HC595 chip, we can use the 3 digital I / O port to control 8 LED lights. Why not?

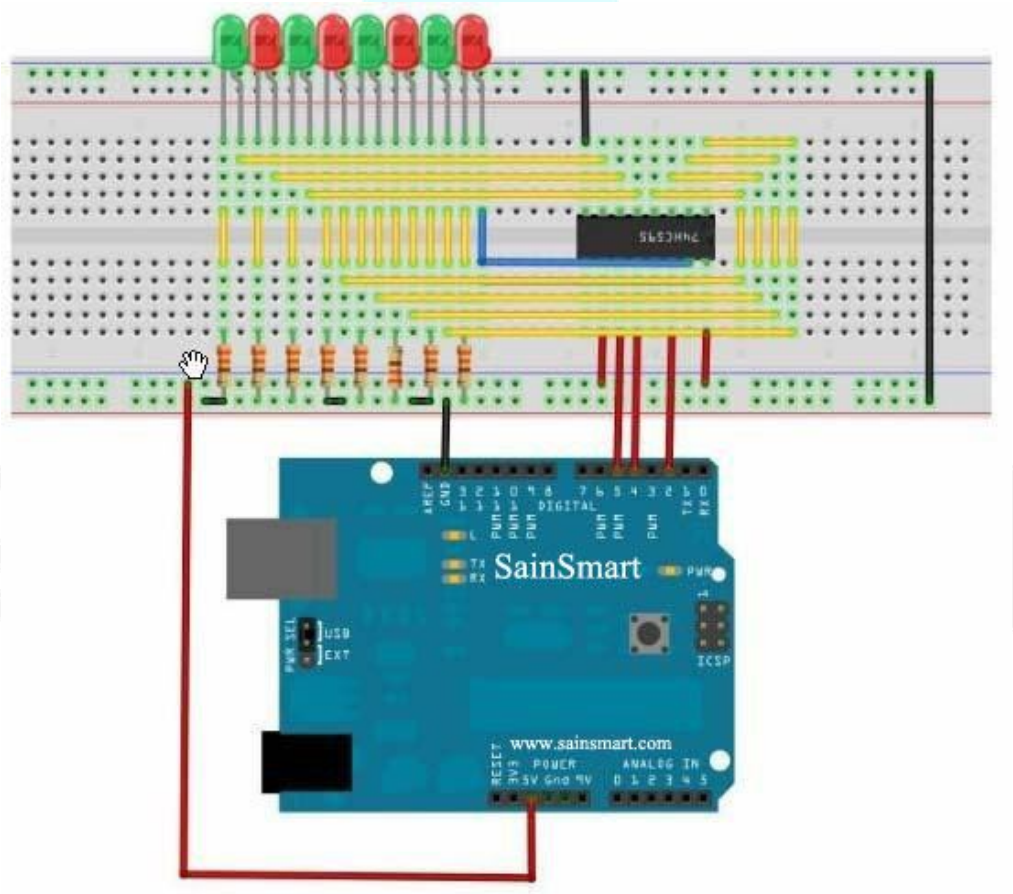
Prepare experimental components below.



Connect the circuit diagram according the schematic diagram.



This schematic seems are complex，after analysis and combined with reference we will find it very simple.



Example code

```
const int ON =      HIGH ;
const int OFF = LOW ;
int latchPin = 5;                    //connect 595 ’ pin 12
int clockPin = 4;                   //connect 595 ’s pin11
int dataPin = 2;                     // connect 595 ’s pin 14
// connect 595’s pin 16 with 5VDC
// connect 595’s pin 8 with GND

int ledState = 0;

void setup() {
   pinMode(latchPin, OUTPUT);
   pinMode(clockPin, OUTPUT);
   pinMode(dataPin, OUTPUT);
}
void loop() {
       int delayTime = 100 ;
   for(int i=0;i<256;i++)
       {
         updateLEDs(i);
         delay(delayTime);
       }
}
void updateLEDs(int value)
{
   digitalWrite(latchPin,LOW);
   shiftOut(dataPin, clockPin, MSBFIRST, value);
   digitalWrite(latchPin,HIGH);
}

void updateLEDsLong(int value)
{
   digitalWrite(latchPin,LOW);
   for(int i=0;i<8;i++)
       {
       int bit = value&B10000000;
       value = value<<1;
       if(bit==128)
             {
             digitalWrite(dataPin,HIGH);
             }
             else
             {
```

```
            digitalWrite(dataPin,LOW);
            }
      digitalWrite(clockPin,HIGH);
delay(1);
      digitalWrite(clockPin,LOW);
      }
   digitalWrite(latchPin,HIGH);
}

int bits[] = {B00000001,B00000010,B00000100,B00001000,B00010000,B00100000,
B01000000,B10000000};
int masks[] = {B11111110,B11111101,B11111011,B11110111,B11101111,B11011111,
B10111111,B01111111};
void changeLED(int led,int state)
{
      ledState = ledState & masks[led];
      if(state == ON){ ledState = ledState|bits[led]; }
   updateLEDs(ledState);
}
```

Downloaded the program into the control panel, we can see the wonderful scene of small lights flashing.

In the connection circuit process, we should pay attention to the clear relay pin position. What's more, the IN4001 diodes are divided into positive and negative. Do not look at the relay circuit is slightly complex, but the kiev program is very simple. The relay is digital signal module. By opening and closing of the relay to the transistor digital signal to control high-power devices. We use LED lights as a high-power devices here.
In program, we use digital port 8 to output high and delay for one second, one second output low, like the switch off for one second and then turned on one second.

Code

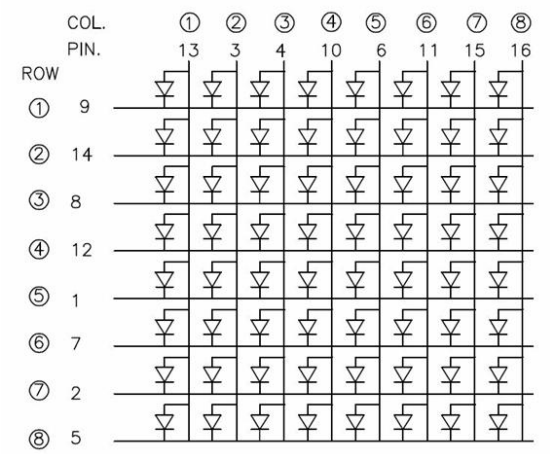
```
int relayPin = 8 ;// define digital port 8, connected to the transistor base
void setup()
{
      pinMode(relayPin,OUTPUT);// define relayPin port to be output mode
}
void loop()
{
    digitalWrite(relayPin,HIGH);// drive relay closes conduction
    delay(1000);//delay one second
    digitalWrite(relayPin,LOW);//drive relay off
    delay(1000);//delay one second
}
```

Result

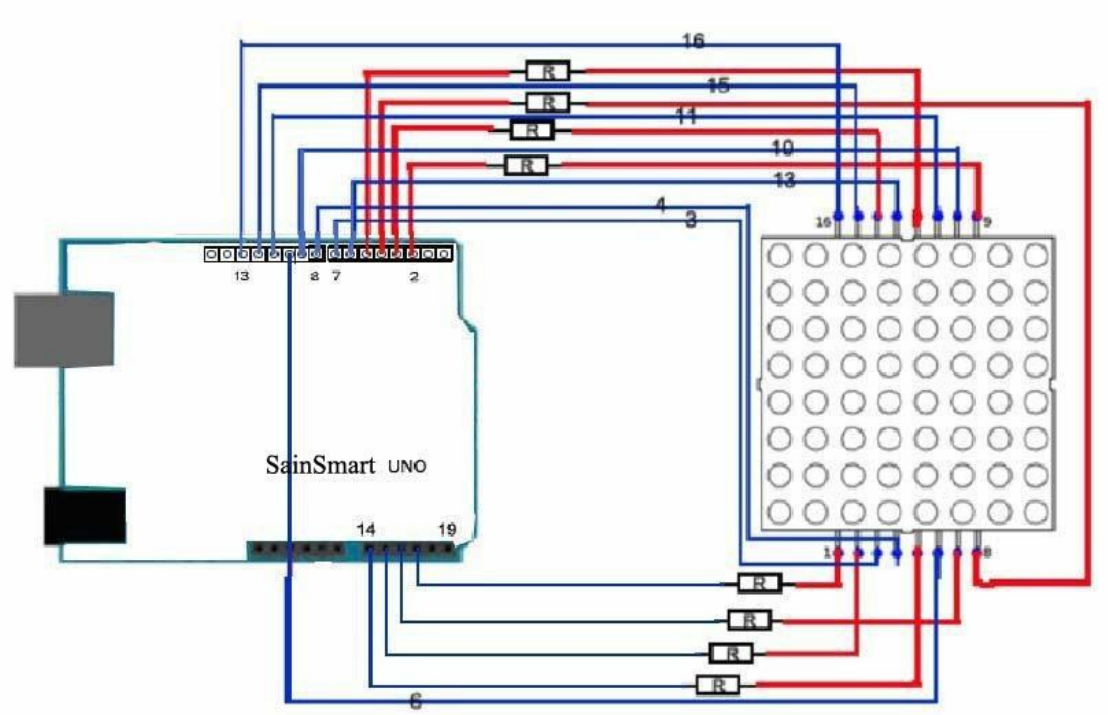
We will see small red lights and green lights flashing take turns. This is the end of this chapter's experiment, we hope that you could enjoy it and create more interactive works.

Chapter15 8x8 matrix LEDs

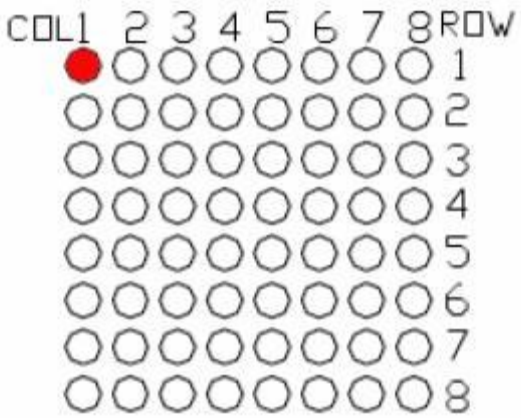
The following figure is a matrix LED internal schematic:



Wiring diagram:



One LED of LED 8X8 matrix is lit as follows：



Sample code:

```
//the pin to control ROW
const int row1 = 2; // the number of the row pin 9
const int row2 = 3; // the number of the row pin 14
const int row3 = 4; // the number of the row pin 8
const int row4 = 5; // the number of the row pin 12
const int row5 = 17; // the number of the row pin 1
const int row6 = 16; // the number of the row pin 7
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 5
//the pin to control COl
const int col1 = 6; // the number of the col pin 13
const int col2 = 7; // the number of the col pin 3
const int col3 = 8; // the number of the col pin 4
const int col4 = 9; // the number of the col pin 10
const int col5 = 10; // the number of the col pin 6
const int col6 = 11; // the number of the col pin 11
const int col7 = 12; // the number of the col pin 15
const int col8 = 13; // the number of the col pin 16


void setup(){
      int i = 0 ;
   for(i=2;i<18;i++)
      {
      pinMode(i, OUTPUT);
      }
   pinMode(row5, OUTPUT);
   pinMode(row6, OUTPUT);
   pinMode(row7, OUTPUT);
   pinMode(row8, OUTPUT);
```

```
    for(i=2;i<18;i++) {
digitalWrite(i, LOW);
        }
    digitalWrite(row5, LOW);
    digitalWrite(row6, LOW);
    digitalWrite(row7, LOW);
    digitalWrite(row8, LOW);
}
void loop(){
    int i;
        //the row # 1 and col # 1 of the LEDs turn on
    digitalWrite(row1, HIGH);
    digitalWrite(row2, LOW);
    digitalWrite(row3, LOW);
    digitalWrite(row4, LOW);
    digitalWrite(row5, LOW);
    digitalWrite(row6, LOW);
    digitalWrite(row7, LOW);
    digitalWrite(row8, LOW);

    digitalWrite(col1, LOW);
    digitalWrite(col2, HIGH);
    digitalWrite(col3, HIGH);
    digitalWrite(col4, HIGH);
    digitalWrite(col5, HIGH);
    digitalWrite(col6, HIGH);
    digitalWrite(col7, HIGH);
    digitalWrite(col8, HIGH);

    delay(1000);

    //turn off all
    for(i=2;i<18;i++) {
        digitalWrite(i, LOW);
        }

    delay(1000);
}
```

The experiment's code are as follows:

By dynamic scanning, it shows letter A in the position1 of the LED matrix.

```
#define data_ascii_A 0x02, 0x0C, 0x18, 0x68, 0x68, 0x18, 0x0C, 0x02 /*"A", 0*/
/**
 **"A"
 #define A  { //
    {0, 0, 0, 0, 0, 0, 1, 0}, //0x02
    {0, 0, 0, 0, 1, 1, 0, 0}, //0x0C
    {0, 0, 0, 1, 1, 0, 0, 0}, //0x18
    {0, 1, 1, 0, 1, 0, 0, 0}, //0x68
    {0, 1, 1, 0, 1, 0, 0, 0}, //0x68
    {0, 0, 0, 1, 1, 0, 0, 0}, //0x18
    {0, 0, 0, 0, 1, 1, 0, 0}, //0x0C
    {0, 0, 0, 0, 0, 0, 1, 0}  //0x02
}
```

Set the value to 1 , then the Led will be turn on !

The code in the folder - "8x8 the matrix LEDs experimental", can be used as a reference, made more exciting experiments.

Chapter16 Infrared remote control

Infrared receiving head

What's Infrared receiving head?

Infrared remote control signals sent a series of binary pulse code. In order to make it from other infrared signal interference during wireless transmission, typically modulated it on a particular carrier frequency, and then emitted it by the infrared-emitting diode. The infrared receiving apparatus will have to filter out other clutter, only the specific frequency of the signal and restoring it into a binary pulse code. That is demodulated.

How it work?

Built-in receiver tube infrared emission tube emitted light signal is converted to a weak signal. This signal via the IC internal amplifier amplifies. Then through automatic gain control, band pass filtering, demodulation, and waveform-shaped to restore the original encoding of the remote control transmitter, coded identification on the electrical input to circuit via a received signal output pin head.

How to connect?

Infrared receiving head has three pin:
VOUT connected to the analog port.
GND received experimental board's GND.
VCC received experimental board's +5 v.

Infrared remote control experiment

Experiment component

1. IR remote control x1
2. Infrared receiving head x1
3. Buzzer x1
4. 220 Ω resistance x1
5. Breadboard & jumper wires

Experiment principle

If you want to decode remote control, you must understand the coding system of the remote controller first. The coding system of the remote control we used is NEC protocol. Now let's learn about NEC protocol:

- **NEC protocol introduction：**

Feature：

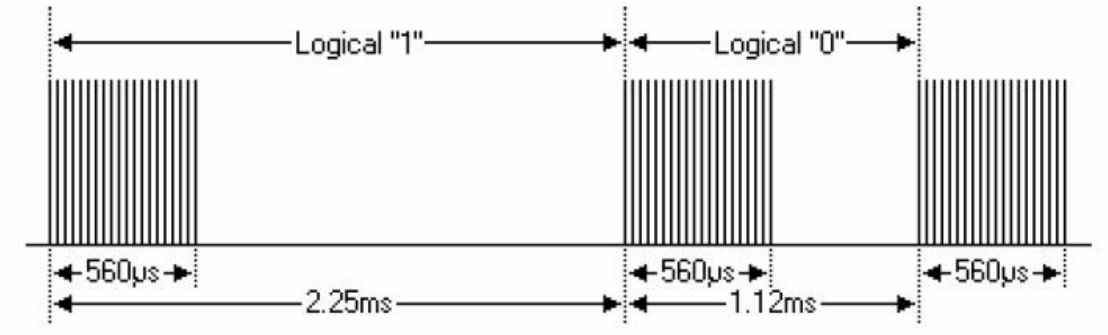
（1）8-bit address spaces, 8-bit command spaces.

（2）address bits and command bits are transmitted twice for reliability.

（3）Pulse position modulation.

（4）Carrier frequency 38khz.

（5）Every bit's time is 1.125ms or 2.25ms.



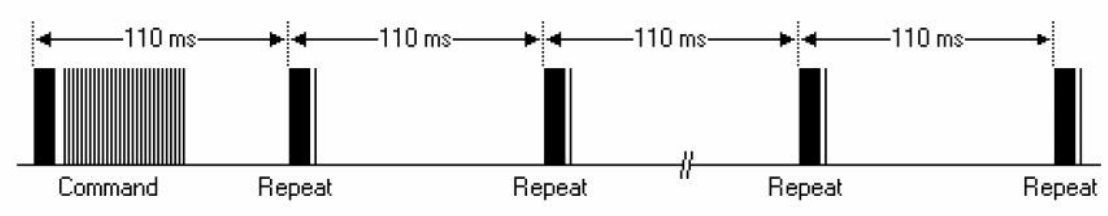
Protocol：

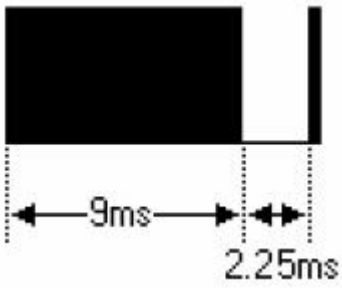
The above picture shows the typical NEC protocol pulse sequence. Note: This is the prior sending the LSB (least significant bit) agreement.

Pulse propagation's address is the 0x59 command 0x16 at the above. A message is start from a 9ms high level, followed by a 4.5ms low level (these two level made boot code) and then by the address code and command code.

Address and command transfer twice. The second time all bits are inverted, can be used for use

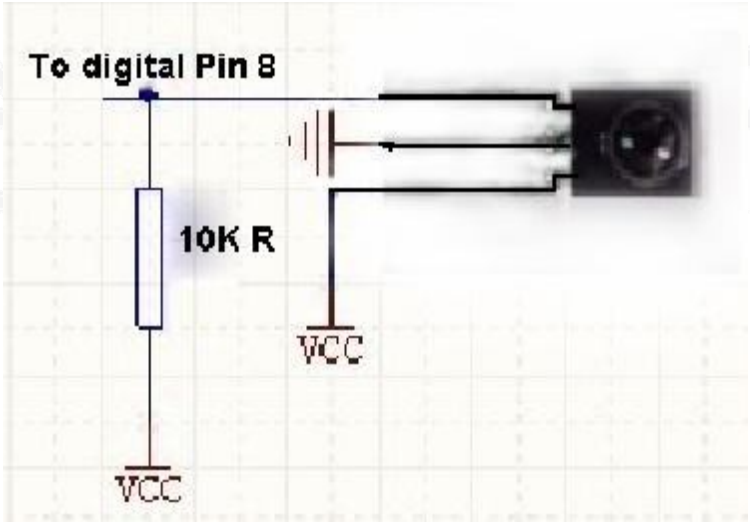
in the received message recognized. The total transmission time is constant, because the duplication of every point of its length negated. If you're not interested, you can ignore this reliability negated address and command can also expand to 16!





According to the characteristics and the receiving end of the waveform of the NEC coding, this experiment will divided receiving end's wave form into four parts: Primer searching code (9ms And 4.5ms pulse), the address code 16 (including an 8-bit address and 8-bit address is negated), the command code 16 (package
Including eight command-bit and 8-bit command negated), repeat code (9ms, 2.25ms, 560us pulse). HIGH segment of the received waveform and low section to be measured using the timer, based on the measured time to distinguish: a logical "0", a logical "1", cited seek pulse, repetitive pulses. Boot code and address code as long as the judge is correct pulse can be, without storage, but the command code must be stored, because each key command codes are different.

Connect your circuit as the below diagram.



Example code

```
#define IR_IN    8 // infrared receive

int Pulse_Width = 0;//storage pulse width
int ir_code = 0x00;//    user code value
char    adrL_code = 0x00;//Command code
char    adrH_code = 0x00;// Command code base minus one's complement

void timer1_init(void)//timer initialization function
{
   TCCR1A = 0X00;
   TCCR1B = 0X05;//set timer clock source
   TCCR1C = 0X00;
   TCNT1 = 0X00;
   TIMSK1 = 0X00;   // Ban timer interrupt overflow
}
void remote_deal(void)// Implement the decoding function
{     // data presentation
       Serial.println(ir_code,HEX);//16 Into system show
       Serial.println(adrL_code,HEX);//16 Into system show
}
char logic_value()//Judgment logic value "0" and "1" son function
{
   TCNT1 = 0X00;
   while(!(digitalRead(IR_IN))); //if low wait
   Pulse_Width=TCNT1;
      TCNT1=0;
   if(Pulse_Width>=7&&Pulse_Width<=10)//low level 560us
      {
      while(digitalRead(IR_IN));//if high wait
      Pulse_Width=TCNT1;
      TCNT1=0;
      if(Pulse_Width>=7&&Pulse_Width<=10)//high level 560us
                  return 0;
      else if(Pulse_Width>=25&&Pulse_Width<=27) //high level 1.7ms
                  return 1;
      }
   return -1;
}
void pulse_deal()//Receiving address code and command code pulse function
{
   int i;
   int j;
```

```
    ir_code=0x00;// clear
    adrL_code=0x00;// clear
    adrH_code=0x00;// clear

        // Analysis of the remote control code user code value
        for(i = 0 ; i < 16; i++)
        {
        if(logic_value() == 1) // if 1
                        ir_code |= (1<<i);//Save key value
        }
        // Analytical remote control code commands in the code
        for(i = 0 ; i < 8; i++)
        {
        if(logic_value() == 1) // if 1
                    adrL_code |= (1<<i);//save key value
        }
        // Analysis of the remote control code user code value
        for(j = 0 ; j < 8; j++)
        {
        if(logic_value() == 1) //if 1
                        adrH_code |= (1<<j);//save key value
        }
}
void remote_decode(void)// Decoding function
{
    TCNT1=0X00;
    while(digitalRead(IR_IN))// if high wait
        {
        if(TCNT1>=1563)    // When high level lasted for more than 100 ms, shows that at the
moment no key press
                {
            ir_code=0x00ff;// user code value
            adrL_code=0x00;// a byte value before Key code
            adrH_code=0x00;// a byte value after Key code
                    return;
                }
        }

        // If high level can't last for more than 100 ms
    TCNT1=0X00;
while(!(digitalRead(IR_IN))); //if low wait
    Pulse_Width=TCNT1;
        TCNT1=0;
    if(Pulse_Width>=140&&Pulse_Width<=141)// 9ms
```

```
    {

    while(digitalRead(IR_IN));//if high wait
    Pulse_Width=TCNT1;
    TCNT1=0;
    if(Pulse_Width>=68&&Pulse_Width<=72)// 4.5ms
        {
            pulse_deal();
            return;
        }
    else if(Pulse_Width>=34&&Pulse_Width<=36)//2.25ms
        {
      while(!(digitalRead(IR_IN)));// if low wait
      Pulse_Width=TCNT1;
            TCNT1=0;
      if(Pulse_Width>=7&&Pulse_Width<=10)// 560us
            {
                return;
            }
        }
    }
}
void setup()
{
  Serial.begin(9600);
  pinMode(IR_IN,INPUT);// Set infrared receiving pin for input
Serial.flush();
}
void loop()
{
  timer1_init();//Timer initialization
  while(1)
    {
    remote_decode();    // decode
    remote_deal();      // Executive decoding results
    }
```

Chapter17 1602LCD

What's 1602LCD?

Nowadays 1602LCD is application of very wide range. The initial 1602 LCD used HD44780 controller. But now various manufacturers basically adopt compatible IC with their 1602 module. Their characteristics are basically the same.

1602LCD

Display capacity: 16x2 characters;
Chip operating voltage: 4.5V~5.5V;
Operating current: 2.0mA(5.0V);
Best operating voltage: 5.0V;
Character size: 2.95x4.35(WxH) mm.

Interface pin definition

number	symbol	states	number	symbol	states
1	VSS	GND	**9**	D2	Date I/O
2	VDD	VCC	**10**	D3	Date I/O
3	VL	VO	**11**	D4	Date I/O
4	RS	(V/L)	**12**	D5	Date I/O
5	R/W	Read/write(H/L)	**13**	D6	Date I/O
6	E	enable	**14**	D7	Date I/O
7	D0	Date I/O	**15**	BLA	Backlight anode
8	D1	Date I/O	**16**	BLK	Backlight cathode

1. Two sets of power supply, a set of modules, the other one is the power of the backlight, generally using the 5V power supply.

2. VL is used to adjust the contrast. It connected in series the potentiometer is not greater than a 5KΩ. This experimental used one 1KΩ of resistor to set contrast. There are high potential connection and low potential connection. It connected in series 1KΩ resistance then connected to GND.

Basic Operation

Read status	Input	RS=L, R/W=H, E=H	Output	D0~D7=status word
Write command	Input	RS=L, R/W=L, D0~D7=command code, E= high pulse	Output	none
Read data	Input	RS=H, R/W=H, E=H	Output	D0~D7=data
Write data	Input	RS=H, R/W=L, D0~D7=data, E= high pulse	Output	none

output of the sketch on a 2x16 LCD
The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:
A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
A Read/Write (R/W) pin that selects reading mode or writing mode
An Enable pin that enables writing to the registers
8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

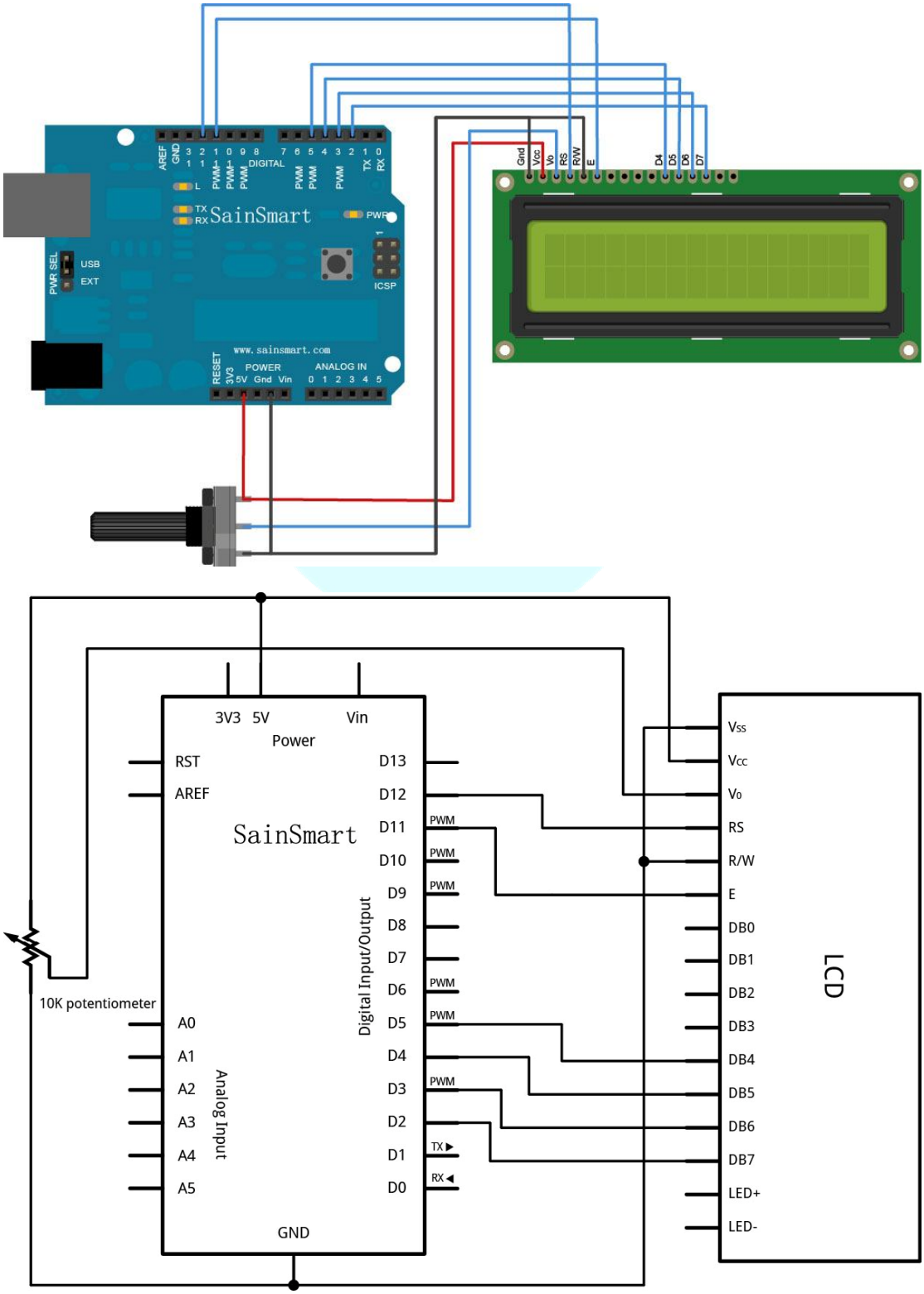
There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.
The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.
The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

Circuit

To wire your LCD screen to your Arduino, connect the following pins:

LCD RS pin to digital pin 12
LCD Enable pin to digital pin 11
LCD D4 pin to digital pin 5
LCD D5 pin to digital pin 4
LCD D6 pin to digital pin 3
LCD D7 pin to digital pin 2

Additionally, wire a 10K pot to +5V and GND, with it's wiper (output) to LCD screens VO pin (pin3).



Example code

```
// include the library code:
#include <LiquidCrystal.h>
```

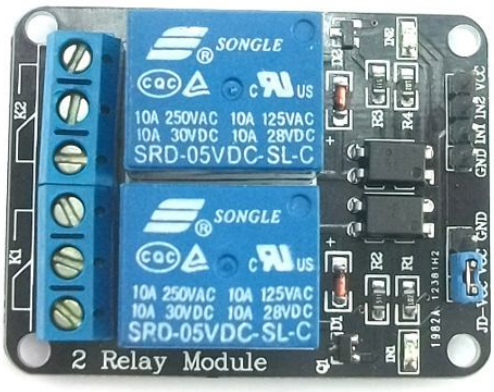
```
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

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

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

Chapter18 Relay module

What's relay?

It will be able to control various appliances, and other equipments with large current. It can be controlled directly by Micro-controller (Arduino , 8051, AVR, PIC, DSP, ARM, ARM, MSP430, TTL logic) .

This project will use 5V 2-Channel Relay interface board.



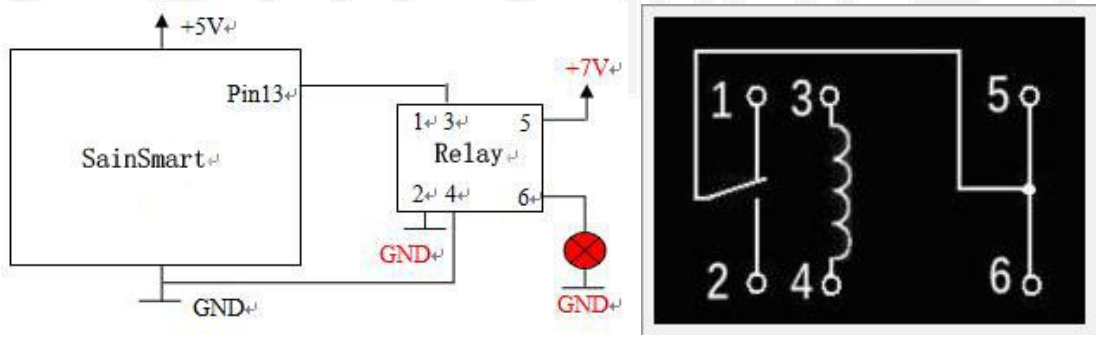
Product features:

- 5V 2-Channel Relay interface board, and each one needs 15-20mA Driver Current
- Equiped with high-current relay, AC250V 10A ; DC30V 10A
- Standard interface that can be controlled directly by microcontroller (Arduino , 8051, AVR, PIC, DSP, ARM, ARM, MSP430, TTL logic)
- Indication LED's for Relay output status

Experiment component

- Relay : 1
- LED : 1
- 10K resistor : 1
- 220Ω resistor : 1
- Breadboard & Jumper wires
- USB cable: 1

Connect your circuit as the below diagram.



Example

```
int jdqPin=13;
void setup()
{
   pinMode(jdqPin,OUTPUT);
   Serial.begin(9600);
}
void loop()
{
digitalWrite(jdqPin,HIGH);
delay(1000);
digitalWrite(jdqPin,LOW);
delay(1000);
}
```

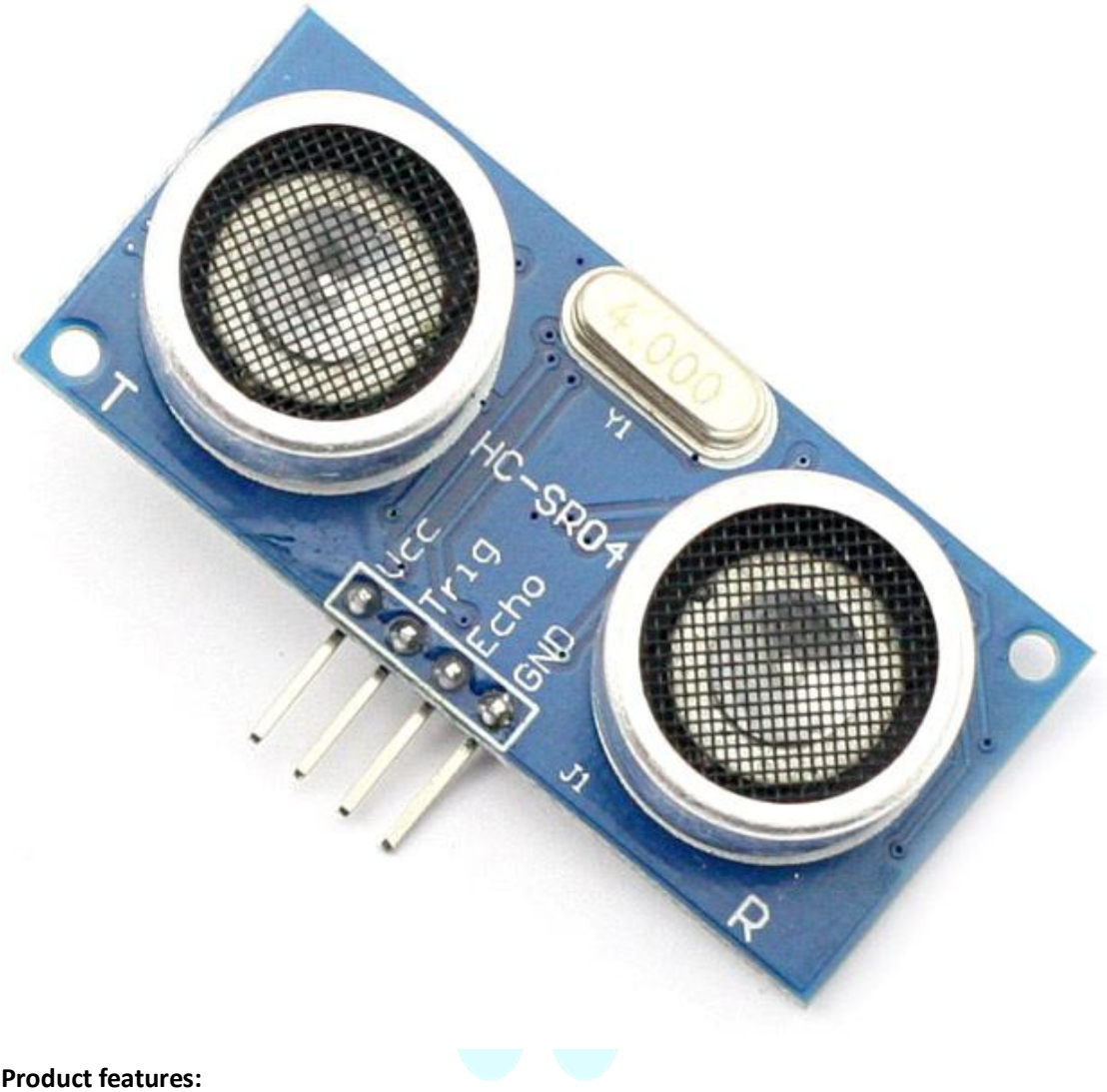
Here I introduce how to use multimeter test relay pin.

General relay has housing mark. If not, is also very simple test with a multimeter:

- 5V power supply
- Multimeter

1. Find the coil pins
   Use multimeter to measure the resistance between the pins. The value of which 2 feet is in hundreds to 1 k ohm resistance are coil pins. Note some of the relay coils are positive and negative, the reverse may not damage, but no action.
2. Find the NO (normally open) contact, NC (normally closed) contact.
   Use multimeter measuring four pins, which two pins breakover are NC contact, coil with 5v direct current, the relay action, they should be disconnected; If there is no disconnect, the internal relations is short sub.
   Add 5v direct current to coils, make the relay action, this time test with a multimeter, if that two pins which disconnect before but connect this time, they are NO contact.
   The pins which have something to do with NO contact, and have something to do with NC contact, is common port.

Chapter19 Distance sensor



Product features:

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

- Using IO trigger for at least 10us high level signal,
- The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning. Test distance = (high level time×velocity of sound (340M/S) /2

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output

- 0V Ground

If you are sourcing a ultrasonic ranging module , the HC-SR04 is good choose . Its stable performance and high ranging accuracy make it a popular module in electronic market .
Compared to the Shap IR ranging module , HC-SR04 is more inexpensive than it . But it has the same ranging accuracy and longer ranging distance.

Specifications:

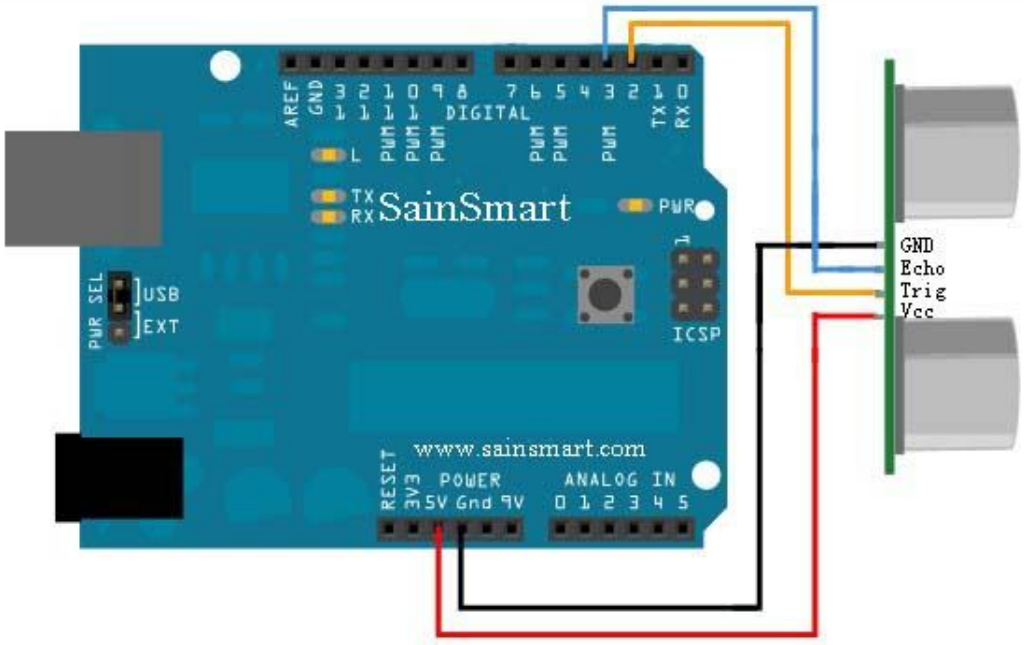
- power supply :5V DC
- quiescent current : <2mA
- effectual angle: <15°
- ranging distance : 2cm – 500 cm
- resolution : 0.3 cm

There are 4 pins out of the module : VCC , Trig, Echo, GND . So it's a very easy interface for controller to use it ranging. The all process is : pull the Trig pin to high level for more than 10us impulse , the module start ranging ; finish ranging , If you find an object in front , Echo pin will be high level , and based on the different distance,it will take the different duration of high level. So we can calculated the distance easily :

Distance = ((Duration of high level)*(Sonic :340m/s))/2

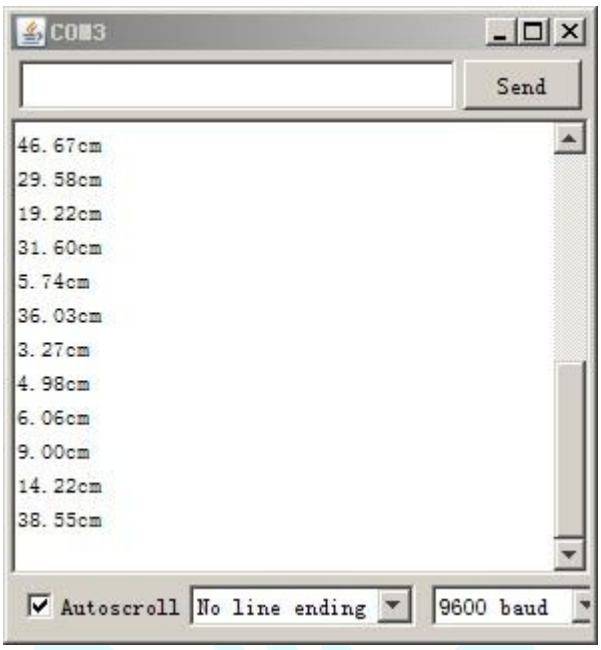
finally , look at the back of the module .All of the chip in the module have been burnish , maybe the author want to prevent the designed from plagiarism. But ultrasonic ranging module is nearly the same principle, so it's not hard to speculated that the role of the chip — I'm sure at least one 74series chip on it ;) . It is not a difficult task to crack it , but … it's at so low a price , even cheaper than your copy.

Connect your circuit as the below diagram.



Example code

```
const int TrigPin = 2;
const int EchoPin = 3;
float cm;
void setup()
  {
  Serial.begin(9600);
  pinMode(TrigPin, OUTPUT);
  pinMode(EchoPin, INPUT);
  }
void loop()
  {
  digitalWrite(TrigPin, LOW); //Low-high-low level sent a short time pulse to TrigPin
  delayMicroseconds(2);
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);

  cm = pulseIn(EchoPin, HIGH) / 58.0; //Echo time converted into cm
  cm = (int(cm * 100.0)) / 100.0; //  retain two decimal places
Serial.print(cm);
  Serial.print("cm");
  Serial.println();
  delay(1000);
  }
```



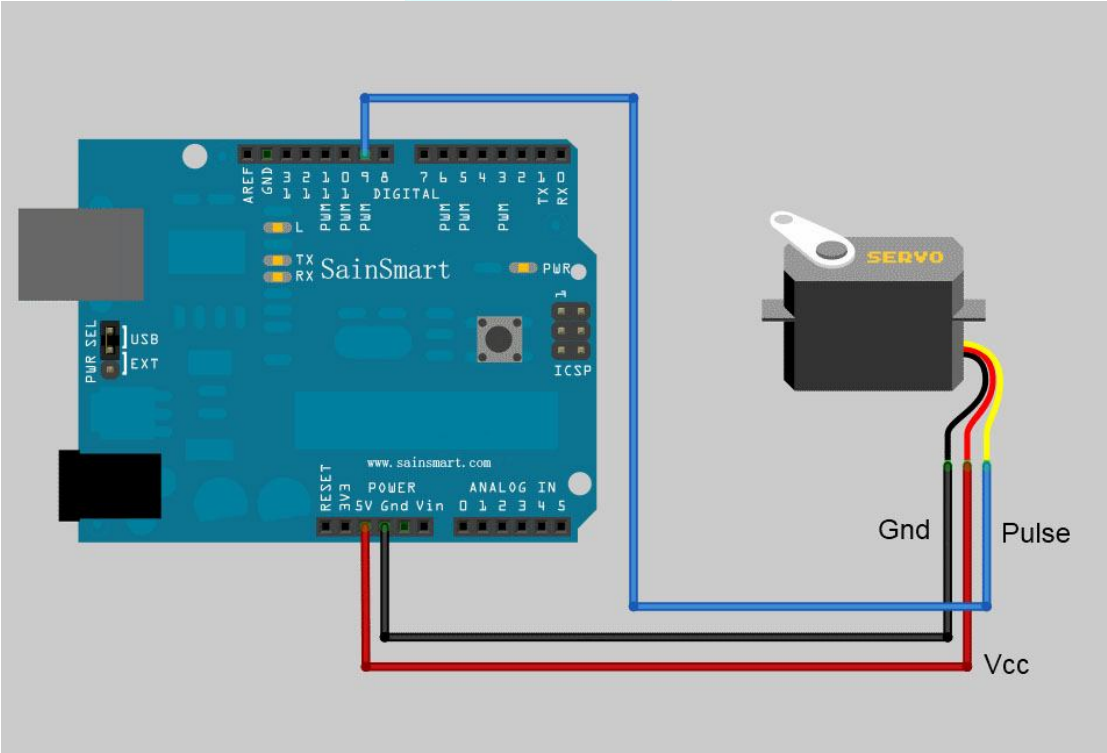
Chapter20 Servo Motor

Controlling a servo motor with an Arduino or other type of microcontroller is probably the easiest way to get started in robotics, motion art, or any other reason you may have to make your electronic project interact with the real world. Servos are very simple to interact with and in this post I'll show you how to connect one to an Arduino.

Servo motors are a specific type of motor, often used in hobby RC cars and planes, that rotate to a specific angle when a corresponding signal is applied to the pulse pin. Servo motors are very easy to program and very strong for their size. This makes them useful for a wide array of applications. The internal components of a servo motor consist of a regular DC motor, which does the actual work, a system of gears to increase the torque to the output shaft, and a circuit board and sensors to control the movement of the motor.

Wiring:

To get started controlling a servo with your Arduino, you only need to **connect three pins**. There are **two pins for power and ground**. For a small servo or just for testing, you can connect these directly to the Arduino. If you are controlling a large servo motor, you might want to use an external power source. Just remember to connect the ground from the external source to the ground of the Arduino.

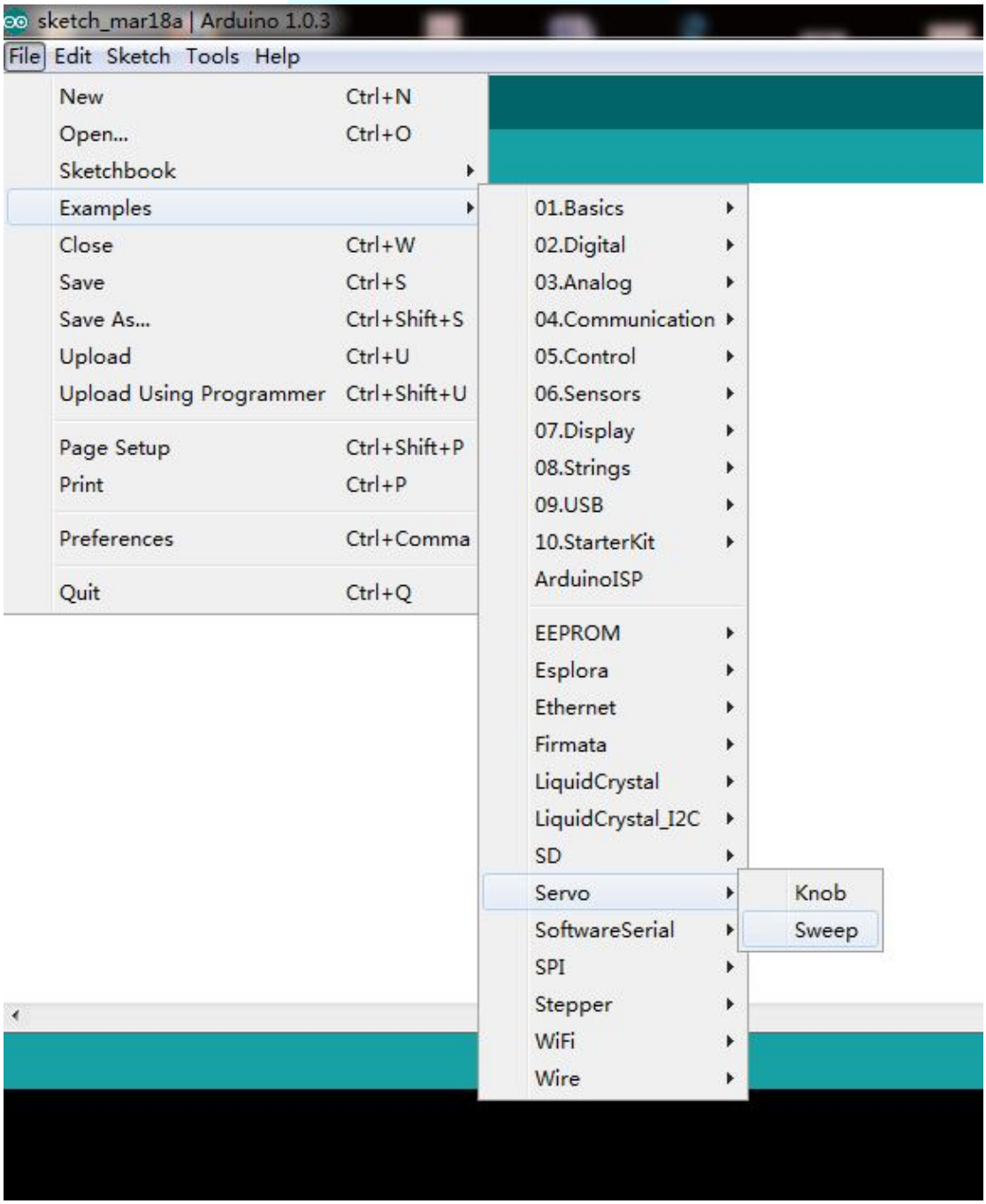


The third pin is the **pulse**, or **signal** pin. This accepts the signal from your controller that tells it what angle to turn to. The control signal is fairly simple compared to that of a stepper motor. It is just a pulse of varying lengths. The length of the pulse corresponds to the angle the motor turns to. Typically a pulse of **1.25 milliseconds causes the motor to rotate to 0 degrees** and a pulse of

1.75 milliseconds turns it 180 degrees. Any length of pulse in between will rotate the servo shaft to its corresponding angle. **Some servos will turn more or less than 180 degrees**, so you may need to experiment.

Programming:

The Arduino software comes with a sample servo sketch and servo library that will get you up and running quickly. Simply load it from the menu as shown below. Their example uses pin 9 for the pulse wire, so to keep it simple, that's what I used. You could use any of the data pins and, if you add more than one servo, you will need to. The Sweep sample simply rotates the servo back and forth from 0 degrees to 180. There is another sample sketch that uses a potentiometer as an input to control the angle of the motor, but I'll get in to that later.



The code is pretty basic and well documented. It first loads the library needed and sets up which pin to use as the output.

This line tells it to move from 0 degrees to 180 degrees one degree at a time:

for(pos = 0; pos < 180; pos += 1)

And this line tells it to move back to 0 degrees one degree at a time.

for(pos = 180; pos>=1; pos-=1)

Chapter21 XMC5883L shield

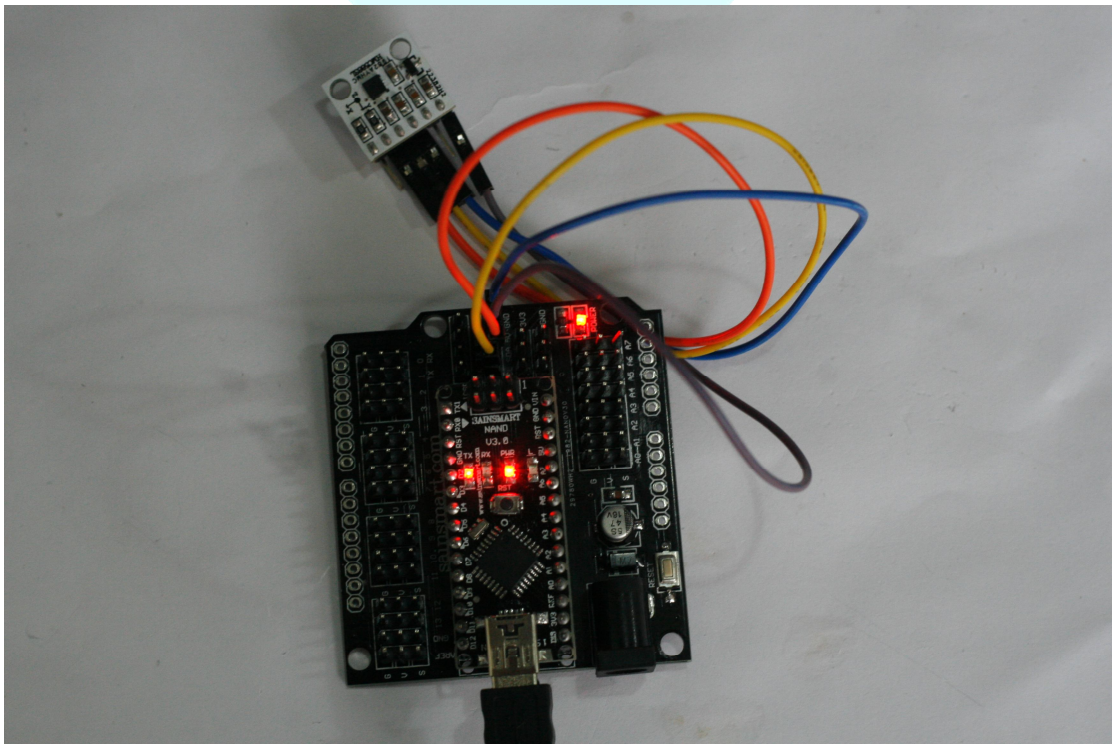
Features of HMC5883 Triple Axis Magnetoresistive Sensor:

1. Digital Output：I2C digital output interface，convenient to use。
2. Small Size: 3x3x0.9mm LCC encapsulation，suitable for mass production。
3. High precision：1—2 degree， 12 bit A/D,OFFSET, SET/RESET circuit built-in，no magnetic saturation，no integrating error.
4. automatic calibration, simplify the process。
5. self-testing circuit built-in, easy for mass test, other testing machine is not required.
6. low power dissipation：supply voltage 1.8V, Sleep mode:2.5uA Measure mode:0.6mA

Wire up:

OnlyVCC, GND, SDA, and SCL need to be connected.

- NANO GND -> HMC5883L GND
- NANO 3.3V -> HMC5883L VCC
- NANO A4 (SDA) -> HMC5883L SDA
- NANO A5 (SCL) -> HMC5883L SCL



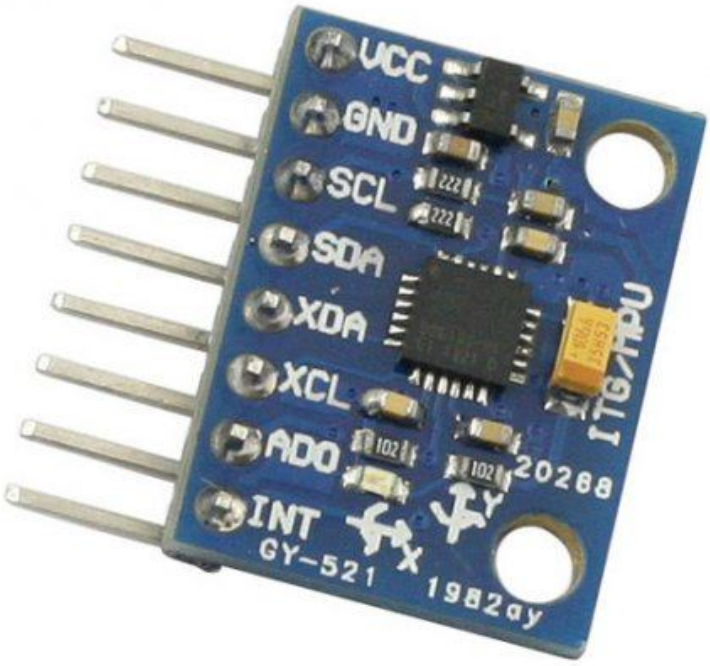
Example code

```
void setup()
{
   Serial.begin(9600);
   Wire.begin();
   compass = HMC5883L();
   compass.SetScale(1.3);
   compass.SetMeasurementMode(Measurement_Continuous);
```

```
}
void loop()
{
   MagnetometerRaw raw = compass.ReadRawAxis();
   MagnetometerScaled scaled = compass.ReadScaledAxis();
   float xHeading = atan2(scaled.YAxis, scaled.XAxis);
   float yHeading = atan2(scaled.ZAxis, scaled.XAxis);
   float zHeading = atan2(scaled.ZAxis, scaled.YAxis);
   if(xHeading < 0)        xHeading += 2*PI;
   if(xHeading > 2*PI)       xHeading -= 2*PI;
   if(yHeading < 0)        yHeading += 2*PI;
   if(yHeading > 2*PI)       yHeading -= 2*PI;
   if(zHeading < 0)        zHeading += 2*PI;
   if(zHeading > 2*PI)       zHeading -= 2*PI;
   float xDegrees = xHeading * 180/M_PI;
   float yDegrees = yHeading * 180/M_PI;
   float zDegrees = zHeading * 180/M_PI;
   Serial.print(xDegrees);
   Serial.print(",");
   Serial.print(yDegrees);
   Serial.print(",");
   Serial.print(zDegrees);
   Serial.println(";");
   delay(100);
}
```

Chapter22 MPU6050 Sensor

What's MPU6050 Sensor?



Description

Key Features

Model: GY-521

Color: Blue

Material: PCB + Plastic + copper

Chip: MPU-6050

Power supply: 3~5V

Communication mode: standard IIC communication protocol

Chip built-in 16bit AD converter, 16bit data output

Gyroscopes range: +/- 250 500 1000 2000 degree/sec

Acceleration range: +/- 2g, +/- 4g, +/- 8g, +/- 16g

Immersion Gold plating PCB, machine welding process to ensure quality

Pin pitch: 2.54mm

Great for DIY projects

Packing list: 1 x Module 2 x Pins

Specification

sDimensions (cm): 2.1 x 1.6 x 0.3

Weight (kg): 0.005

If you need more information about MPU6050, visit:
http://www.sainsmart.com/sainsmart-mpu-6050-3-axis-gyroscope-module.html

Example code

- **This code is without the algorithm, so the result is just raw data!**

```
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;

#define LED_PIN 13
bool blinkState = false;

void setup() {

    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    // initialize serial communication
    // (38400 chosen because it works as well at 8MHz as it does at 16MHz, but
    // it's really up to you depending on your project)
    Serial.begin(38400);

    // initialize device
    Serial.println("Initializing I2C devices...");
    accelgyro.initialize();
    // verify connection
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050
connection failed");
```

```
    // configure Arduino LED for
    pinMode(LED_PIN, OUTPUT);

}
    void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    // these methods (and a few others) are also available
    //accelgyro.getAcceleration(&ax, &ay, &az);
    //accelgyro.getRotation(&gx, &gy, &gz);

    // display tab-separated accel/gyro x/y/z values
    Serial.print("a/g:\t");
    Serial.print(ax);
    Serial.print("\t");
    Serial.print(ay);
    Serial.print("\t");
    Serial.print(az);
    Serial.print("\t");
    Serial.print(gx);
    Serial.print("\t");
    Serial.print(gy);
    Serial.print("\t");
    Serial.println(gz);

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);

}
```

- **This is the code with algorithm.**

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;

bool blinkState = false;

void setup() {
```

```
        Wire.begin();
        Serial.begin(38400);

        accelgyro.initialize();
  }
void loop() {

        accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        Serial.print("a/g:\t");
        Serial.print(ax/16384);
        Serial.print("\t");
        Serial.print(ay/16384);
        Serial.print("\t");
        Serial.print(az/16384);
        Serial.print("\t");
        Serial.print(gx/131);
        Serial.print("\t");
        Serial.print(gy/131);
        Serial.print("\t");
        Serial.println(gz/131);
        blinkState = !blinkState;

  }
```

Chapter23 Keypad

What's keypad?

Feature.

8P DuPont head, pitch 2.54mm, can be inserted in the Pin connection circuit;

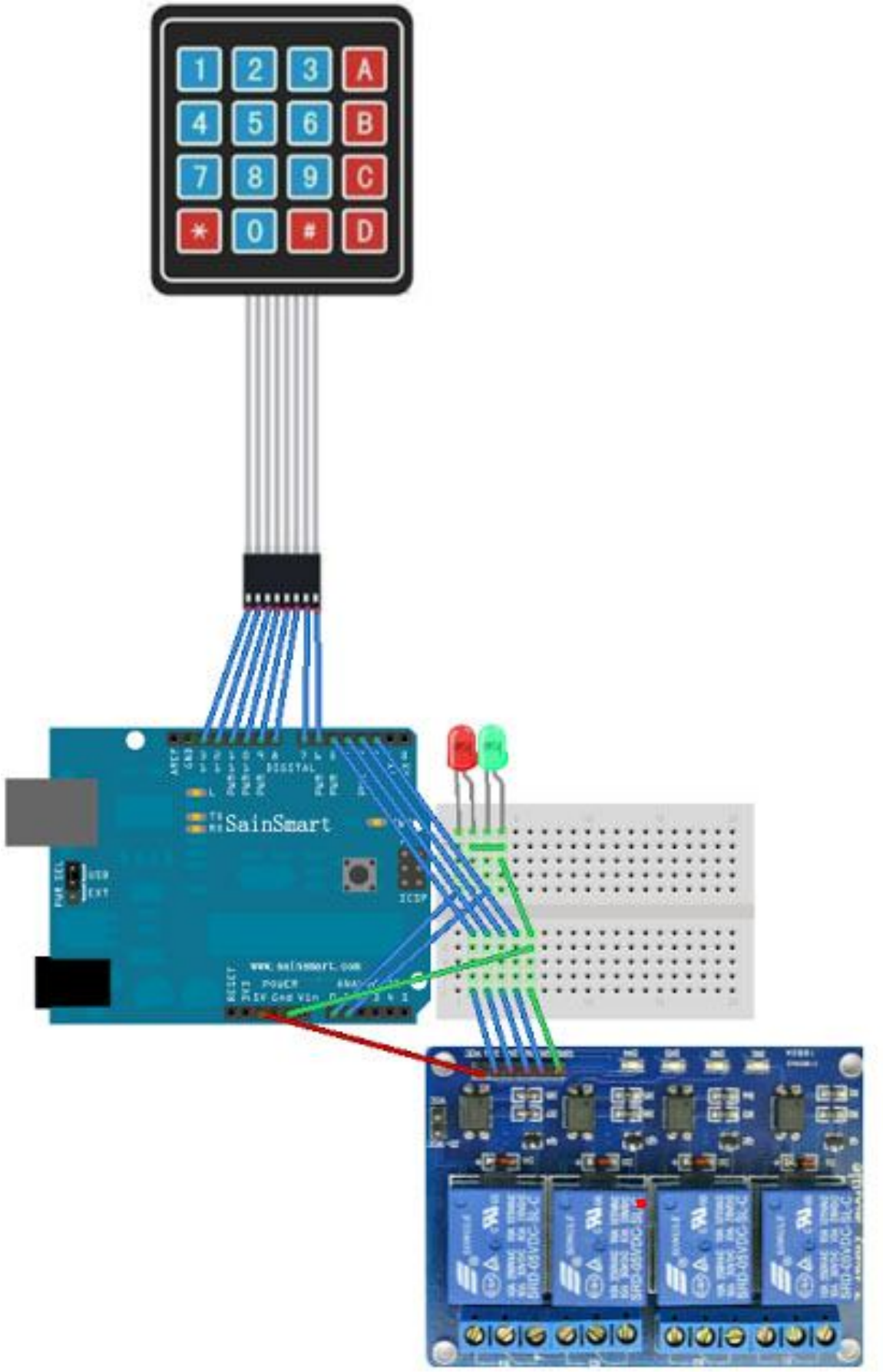
Peel off the white sticker on the back of the keyboard can be securely affixed to the surface of the chassis

Experiment component

- Relay : 1
- LED: 2
- UNO R3: 1
- Keypad : 1
- Breadboard & Jumper wires
- USB cable: 1

Wiring

Wiring up the parts is easier than it might seam. Note that D0 and D1 are used for serial programing. D13 has a resister wired into it so not good for the relay signaling. Analog pins with LEDs is a bit of a hack for some more I/O.



LEDs

Red LED positive connected to Arduino Uno Analog A0

Green LED positive connected to Arduino Uno Analog A1

Ground legs connected to Arduino Ground

Relays

VCC connected to Arduino 5v

IN1 connected to Arduino D2

IN2 connected to Arduino D3

IN3 connected to Arduino D4

IN4 connected to Arduino D5

Ground connected to Arduino Ground

Keypad

Connected to Arduino D6-D13

Power

9V battery connected to Arduino Ground and VIN when not connected to computer USB

Example code

```
/* Locked Relays
 *
 * An SainSmart Uno, Keypad, Relays and some LEDs for fun
 *
 * Using a password to enable the relays, then selective
 *       toggle the relays by key.
 *
 * Needed libraries
 * http://arduino.cc/playground/uploads/Code/Keypad.zip
 * http://arduino.cc/playground/uploads/Code/Password.zip
 */

#include <Keypad.h>/*
#include <Password.h>/*

int relay1 = 2;
int relay2 = 3;
int relay3 = 4;
int relay4 = 5;

int locked = 1;
int passinput = 0;
int lockedled = 14;
int unlockedled = 15;

long ledflashvar = 0;
long ledflashtime = 300;
```

```
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {{'1','2','3','A'},{'4','5','6','B'},{'7','8','9','C'},{'*','0','#','D'}};
byte rowPins[ROWS] = {13, 12, 11, 10};
byte colPins[COLS] = {9, 8, 7, 6};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
Password password = Password("0000");

void setup(){
   Serial.begin(9600);
   pinMode(relay1, OUTPUT);
   digitalWrite(relay1, 255);
   pinMode(relay2, OUTPUT);
   digitalWrite(relay2, 255);
   pinMode(relay3, OUTPUT);
   digitalWrite(relay3, 255);
   pinMode(relay4, OUTPUT);
   digitalWrite(relay4, 255);
   pinMode(lockedled, OUTPUT);
   digitalWrite(lockedled, 255);
   pinMode(unlockedled, OUTPUT);
   digitalWrite(unlockedled, 0);
}

void loop(){
   char key = keypad.getKey();
   if(locked){
      if(passinput){
         unsigned long ledcurrentvar = millis();
         if(ledcurrentvar - ledflashvar > ledflashtime) {
            ledflashvar = ledcurrentvar;
            digitalWrite(lockedled, !digitalRead(lockedled));
         }
      }
      else{
         digitalWrite(lockedled, 255);
      }
      digitalWrite(unlockedled, 0);
   }
   if (key != NO_KEY){
      Serial.println(key);
      password.append(key);
```

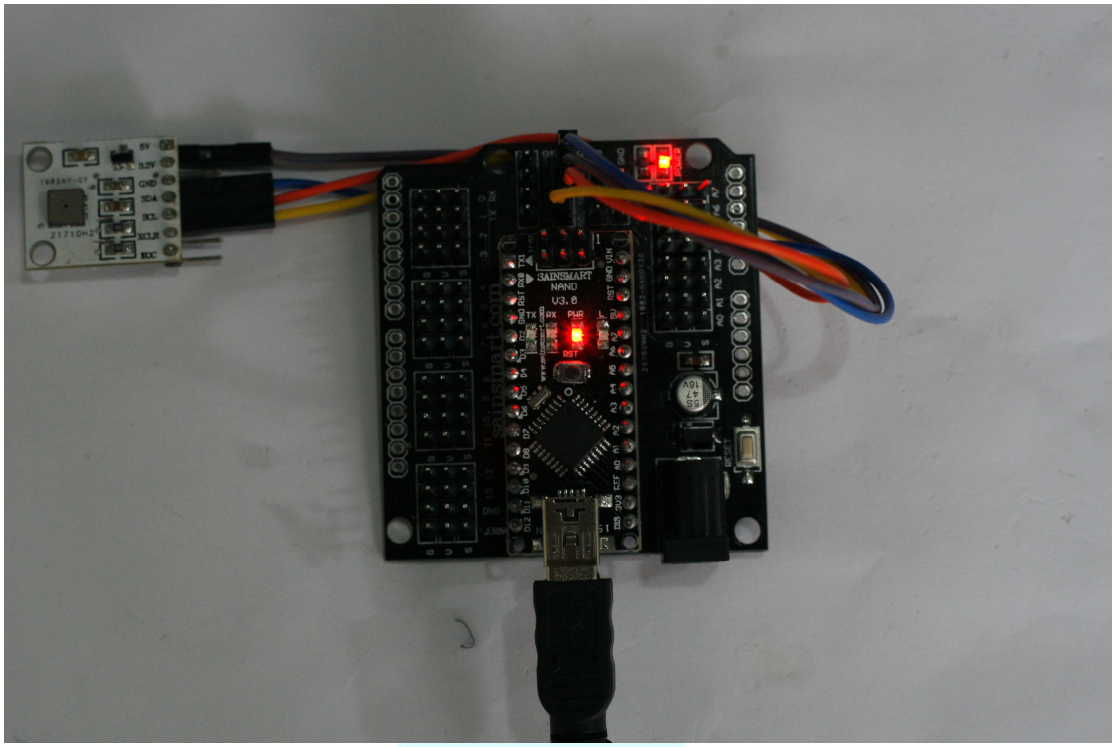
```
        passinput = 1;
        if(key == '*'){
            password.reset();
            passinput = 0;
            locked = 1;
            digitalWrite(relay1, HIGH);
            digitalWrite(relay2, HIGH);
            digitalWrite(relay3, HIGH);
            digitalWrite(relay4, HIGH);
        }
        if(password.evaluate()) {
            locked = !locked;
            password.reset();
            passinput = 0;
        }
        if(!locked) {
            passinput = 0;
            digitalWrite(lockedled, 0);
            digitalWrite(unlockedled, 255);
            switch (key) {
                case 'A':
                    digitalWrite(relay1, !digitalRead(relay1));
                    break;
                case 'B':
                    digitalWrite(relay2, !digitalRead(relay2));
                    break;
                case 'C':
                    digitalWrite(relay3, !digitalRead(relay3));
                    break;
                case 'D':
                    digitalWrite(relay4, !digitalRead(relay4));
                    break;
            }
            password.reset();
        }
    }
}
```

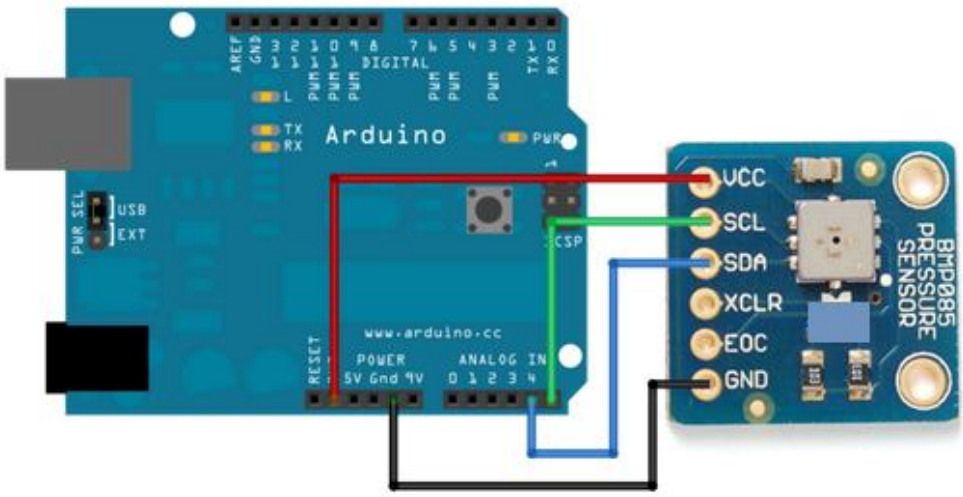
Chapter24 BMP085 shield

The BMP085 is a basic sensor that is designed specifically for measuring barometric pressure (it also does temperature measurement on the side to help). It's one of the few sensors that does this measurement, and its fairly low cost so you'll see it used a lot. You may be wondering why

someone would want to measure atmospheric pressure, but its actually really useful for two things. One is to measure altitude. As we travel from below sea level to a high mountain, the air pressure *decreases*. That means that if we measure the pressure we can determine our altitude - handy when we don't want the expense or size of a GPS unit. Secondly, atmospheric pressure can be used as a predictor of weather which is why weather-casters often talk about "pressure systems"

That's it! Now we can wire the board up to the microcontroller.





Connect the **VCC** pin to a **3.3V** power source. The V2 of the sensor board has a 3.3V regulator so you can connect it to either 3.3V or 5V if you do not have 3V available.

Connect **GND** to the ground pin.

Connect the **i2c SCL clock** pin to your i2c clock pin. On the classic SainSmart Uno/Duemilanove/MEGA/etc this is **Analog pin #5**

Connect the **i2c SDA data** pin to your i2c data pin. On the classic SainSmart Uno/Duemilanove/MEGA/etc this is **Analog pin #4**

Unfortunately, the i2c lines on most microcontrollers are fixed so you're going to have to stick with those pins.

You don't need to connect the **XCLR** (reset) or **EOC** (end-of-conversion) pins. If you need to speed up your conversion time, you can use the EOC as a indicator - in our code we just hang out and wait the maximum time possible.

Example code

```
#include <Wire.h>

#define BMP085_ADDRESS 0x77  // I2C address of BMP085

const unsigned char OSS = 0;  // Oversampling Setting

// Calibration values
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
int mb;
int mc;
int md;

// b5 is calculated in bmp085GetTemperature(...), this variable is also used in bmp085GetPressure(...)
// so ...Temperature(...) must be called before ...Pressure(...).
long b5;

short temperature;
long pressure;

void setup()
{
```

```
    Serial.begin(9600);
    Wire.begin();
    bmp085Calibration();
}

void loop()
{
    temperature = bmp085GetTemperature(bmp085ReadUT());
    pressure = bmp085GetPressure(bmp085ReadUP());
    Serial.print("Temperature: ");
    Serial.print(temperature, DEC);
    Serial.println(" *0.1 deg C");
    Serial.print("Pressure: ");
    Serial.print(pressure, DEC);
    Serial.println(" Pa");
    Serial.println();
    delay(1000);
}

// Stores all of the bmp085's calibration values into global variables
// Calibration values are required to calculate temp and pressure
// This function should be called at the beginning of the program
void bmp085Calibration()
{
    ac1 = bmp085ReadInt(0xAA);
    ac2 = bmp085ReadInt(0xAC);
    ac3 = bmp085ReadInt(0xAE);
    ac4 = bmp085ReadInt(0xB0);
    ac5 = bmp085ReadInt(0xB2);
    ac6 = bmp085ReadInt(0xB4);
    b1 = bmp085ReadInt(0xB6);
    b2 = bmp085ReadInt(0xB8);
    mb = bmp085ReadInt(0xBA);
    mc = bmp085ReadInt(0xBC);
    md = bmp085ReadInt(0xBE);
}

// Calculate temperature given ut.
// Value returned will be in units of 0.1 deg C
short bmp085GetTemperature(unsigned int ut)
{
    long x1, x2;

    x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
```

```
  x2 = ((long)mc << 11)/(x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8)>>4);
}

// Calculate pressure given up
// calibration values must be known
// b5 is also required so bmp085GetTemperature(...) must be called first.
// Value returned will be pressure in units of Pa.
long bmp085GetPressure(unsigned long up)
{
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calculate B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<<OSS) + 2)>>2;

  // Calculate B4
  x1 = (ac3 * b6)>>13;
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;

  b7 = ((unsigned long)(up - b3) * (50000>>OSS));
  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;

  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  p += (x1 + x2 + 3791)>>4;

  return p;
}

// Read 1 byte from the BMP085 at 'address'
char bmp085Read(unsigned char address)
```

```
{
  unsigned char data;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.send(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 1);
  while(!Wire.available())
    ;

  return Wire.receive();
}

// Read 2 bytes from the BMP085
// First byte will be from 'address'
// Second byte will be from 'address'+1
int bmp085ReadInt(unsigned char address)
{
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.send(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while(Wire.available()<2)
    ;
  msb = Wire.receive();
  lsb = Wire.receive();

  return (int) msb<<8 | lsb;
}

// Read the uncompensated temperature value
unsigned int bmp085ReadUT()
{
  unsigned int ut;

  // Write 0x2E into Register 0xF4
  // This requests a temperature reading
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.send(0xF4);
  Wire.send(0x2E);
```

```
  Wire.endTransmission();

  // Wait at least 4.5ms
  delay(5);

  // Read two bytes from registers 0xF6 and 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}

// Read the uncompensated pressure value
unsigned long bmp085ReadUP()
{
  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.send(0xF4);
  Wire.send(0x34 + (OSS<<6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3<<OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.send(0xF6);
  Wire.endTransmission();
  Wire.requestFrom(BMP085_ADDRESS, 3);

  // Wait for data to become available
  while(Wire.available() < 3)
    ;
  msb = Wire.receive();
  lsb = Wire.receive();
  xlsb = Wire.receive();

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb << 8) | (unsigned long) xlsb) >>
(8-OSS);

  return up;
}
```