Arduino 控制器使用教程

概 述

什么是 Arduino?

Arduino 是一块基与开放原始代码的 Simple i/o 平台，並且具有使用类似 java,C 语言的开发环境。让您可以快速使用 Arduino 语言与 Flash 或 Processing…等软件，作出互动作品。Arduino 可以使用开发完成的电子元件例如 Switch 或 Sensors 或其他控制器、ＬＥＤ、步进电机或其他输出裝置。Arduino 也可以独立运作成为一个可以跟软件沟通的平台，例如说：flash processing Max/MSP VVVV 或其他互动软件…

Arduino 开发ＩＤＥ界面基于开放原始码原则，可以让您免费下载使用开发出更多令人惊奇的互动作品。

什么是 Roboduino?

DFRduino 与 Arduino 完全兼容，只是在原来的基础上作了些改进。Arduino 的 IO 使用的孔座，做互动作品需要面包板和针线搭配才能进行，而 DFRduino 的 IO 使用针座，使用我们的杜邦线就可以直接把各种传感器连接到 DFRduino 上。

特色描述

1. 开放原始码的电路图设计，程式开发界面免费下载，也可依需求自己修改!!
2. DFRduino 可使用 ISP 下载线，自我將新的 IC 程序烧入「bootloader」；
3. 可依据官方电路图，简化 DFRduino 模组，完成独立云作的微处理控制器；
4. 可简单地与传感器、各式各样的电子元件连接(如：红外线,超声波,热敏电阻,光敏电阻,伺服电机等)；
5. 支援多样的互动程式 如: Flash,Max/Msp,VVVV,PD,C,Processing 等；
6. 使用低价格的微处理控制器(ATMEGA168V-10PI)；
7. USB 接口，不需外接电源，另外有提供 9VDC 输入接口；
8. 应用方面，利用 DFRduino，突破以往只能使用滑鼠，键盘，CCD 等输入的裝置的互动內容，可以更简单地达成单人或多人游戏互动。

性能描述

1. Digital I/O 数字输入/输出端共 0~13。
2. Analog I/O 模拟输入/输出端共 0~5。
3. 支持 USB 接口协议及供电(不需外接电源)。
4. 支持 ISP 下载功能。
5. 支持单片机 TX/RX 端子。
6. 支持 USB TX/RX 端子。
7. 支持 AREF 端子。
8. 支持六組 PWM 端子(Pin11,Pin10,Pin9，Pin6,Pin5,Pin3)。
9. 输入电压：接上 USB 时无须外部供电或外部 5V~9V DC 输入。
10. 输出电压：5V DC 输出和 3.3V DC 输出 和外部电源输入。
11. 采用 Atmel Atmega168V-10PI 单片机。
12. DFRduino 大小尺寸：宽 70mm X 高 54mm。

第一课 Arduino 语言

Arduino 语言是建立在 C/C++基础上的，其实也就是基础的 C 语言，Arduino 语言只不过把 AVR 单片机（微控制器）相关的一些寄存器参数设置等都函数化了，不用我们去了解他的底层，让不太了解 AVR 单片机（微控制器）的朋友也能轻松上手。

那么这里就简单的注释一下 Arduino 语言。

/**********基础 C 语言**********/

关键字：

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

语法符号：

- ;
- {}
- //
- /* */

运算符：

- =
- +
- -
- *
- /
- %
- ==
- !=

- <
- >
- <=
- >=
- &&
- ||
- !
- ++
- --
- +=
- -=
- *=
- /=

数据类型：

- boolean 布尔类型
- char 字符类型
- byte 字节类型
- int 整数类型
- unsigned int 无符号整型
- long 长整型
- unsigned long 无符号长整型
- float 实数类型
- double
- string
- array
- void

数据类型转换：

- char()
- byte()
- int()
- long()
- float()

常量：

- HIGH | LOW 表示数字 IO 口的电平，HIGH 表示高电平（1），LOW 表示低电平（0）。

- INPUT | OUTPUT 表示数字 IO 口的方向，INPUT 表示输入（高阻态），OUTPUT 表示输出（AVR 能提供 5V 电压 40mA 电流）。
- true | false true 表示真（1），false 表示假（0）。

/**********/

以上为基础 c 语言的关键字和符号，有 c 语言基础的都应该了解其含义，这里也不作过多的解释。

/**********Arduino 语言**********/

结构

- void setup() 初始化变量，管脚模式，调用库函数等
- void loop() 连续执行函数内的语句

功能

数字 I/O

- pinMode(pin, mode) 数字 IO 口输入输出模式定义函数，pin 表示为 0～13， mode 表示为 INPUT 或 OUTPUT。
- digitalWrite(pin, value) 数字 IO 口输出电平定义函数，pin 表示为 0～13，value 表示为 HIGH 或 LOW。比如定义 HIGH 可以驱动 LED。
- int digitalRead(pin) 数字 IO 口读输入电平函数，pin 表示为 0～13，value 表示为 HIGH 或 LOW。比如可以读数字传感器。

模拟 I/O

- int analogRead(pin) 模拟 IO 口读函数，pin 表示为 0～5（Arduino Diecimila 为 0～5，Arduino nano 为 0～7）。比如可以读模拟传感器（10 位 AD，0～5V 表示为 0～1023）。
- analogWrite(pin, value) - *PWM* 数字 IO 口 PWM 输出函数，Arduino 数字 IO 口标注了 PWM 的 IO 口可使用该函数，pin 表示 3, 5, 6, 9, 10, 11，value 表示为 0～255。比如可用于电机 PWM 调速或音乐播放。

扩展 I/O

- shiftOut(dataPin, clockPin, bitOrder, value) SPI 外部 IO 扩展函数，通常使用带 SPI 接口的 74HC595 做 8 个 IO 扩展，dataPin 为数据口，clockPin 为时钟口，bitOrder 为数据传输方向（**MSBFIRST** 高位在前，**LSBFIRST** 低位在前），value 表示所要传送的数据（0～255），另外还需要一个 IO 口做 74HC595 的使能控制。
- unsigned long pulseIn(pin, value) 脉冲长度记录函数，返回时间参数（us），pin 表示为 0～13，value 为 HIGH 或 LOW。比如 value 为 HIGH，那么当 pin 输入为高电平时，开始计时，当 pin 输入为低电平时，停止计时，然后返回该时间。

时间函数

- unsigned long millis() 返回时间函数（单位 ms），该函数是指，当程序运行就开始计时并返回记录的参数，该参数溢出大概需要 50 天时间。
- delay(ms) 延时函数（单位 ms）。
- delayMicroseconds(us) 延时函数（单位 us）。

数学函数

- min(x, y) 求最小值
- max(x, y) 求最大值
- abs(x) 计算绝对值
- constrain(x, a, b) 约束函数，下限 a，上限 b，x 必须在 ab 之间才能返回。
- map(value, fromLow, fromHigh, toLow, toHigh) 约束函数，value 必须在 fromLow 与 toLow 之间和 fromHigh 与 toHigh 之间。
- pow(base, exponent) 开方函数，base 的 exponent 次方。
- sq(x) 平方
- sqrt(x) 开根号

三角函数

- sin(rad)
- cos(rad)
- tan(rad)

随机数函数

- randomSeed(seed) 随机数端口定义函数，seed 表示读模拟口 analogRead(pin)函数 。
- long random(max) 随机数函数，返回数据大于等于 0，小于 max。
- long random(min, max) 随机数函数，返回数据大于等于 min，小于 max。

外部中断函数

- attachInterrupt(interrupt, , mode) 外部中断只能用到数字 IO 口 2 和 3，interrupt 表示中断口初始 0 或 1，表示一个功能函数，mode：**LOW** 低电平中断，**CHANGE** 有变化就中断，**RISING** 上升沿中断，**FALLING** 下降沿中断。
- detachInterrupt(interrupt) 中断开关，interrupt=1 开，interrupt=0 关。

中断使能函数

- interrupts() 使能中断
- noInterrupts() 禁止中断

串口收发函数

- Serial.begin(speed) 串口定义波特率函数，speed 表示波特率，如 9600，19200 等。

- int Serial.available() 判断缓冲器状态。
- int Serial.read() 读串口并返回收到参数。
- Serial.flush() 清空缓冲器。
- Serial.print(data) 串口输出数据。
- Serial.println(data) 串口输出数据并带回车符。

/**********Arduino 语言库文件**********/

官方库文件 下载地址：http://arduino.cc/en/Reference/Libraries

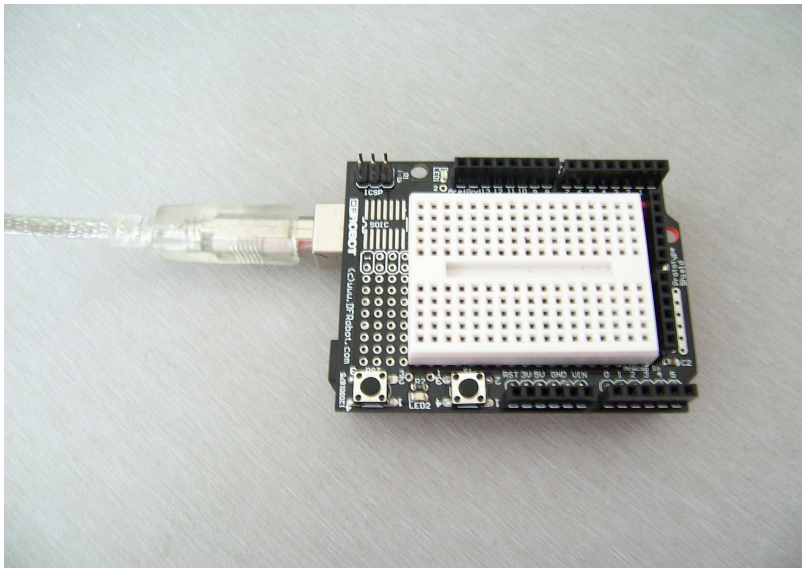
- EEPROM - EEPROM 读写程序库
- Ethernet - 以太网控制器程序库
- LiquidCrystal - LCD 控制程序库
- Servo - 舵机控制程序库
- SoftwareSerial - 任何数字 IO 口模拟串口程序库
- Stepper - 步进电机控制程序库
- Wire - TWI/I2C 总线程序库
- Matrix - LED 矩阵控制程序库
- Sprite - LED 矩阵图象处理控制程序库

/***/

第二课 Arduino 的数字、模拟端口

Arduino 控制器内带 Bootloader 程序，是系统上电后运行的第一段代码，就好比 PC 机 BIOS 中的程序，启动就进行自检，配置端口等等，当然单片机就是靠烧写熔丝位来设定上电从 boot 区启动的，使用这个程序就可以直接把从串口发来的程序存放到 flash 区中。我们在使用 Arduino 编译环境下载程序时，就先让单片机复位，启动 Bootloader 程序引导串口发过来的程序顺利写入 flash 区中，flash 可以重复烧写，因此想更新软件就是这么的方便。下面我来简单的介绍一下驱动的安装和编译环境的使用。首先连接下载程序用的下载线。

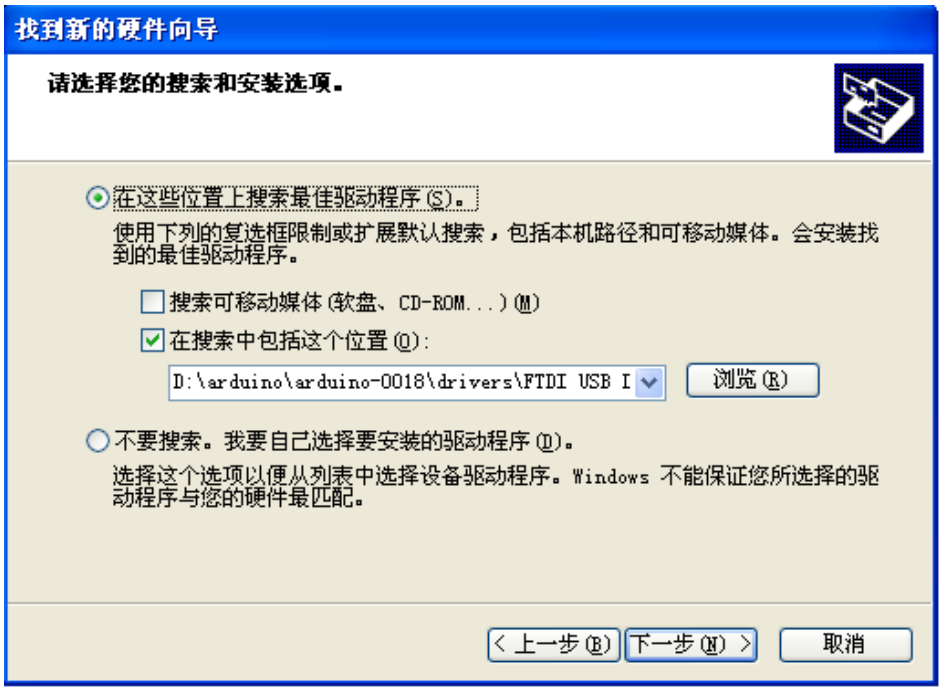
首先将数据线的圆口一端插在 Arduino328 板子上如图：



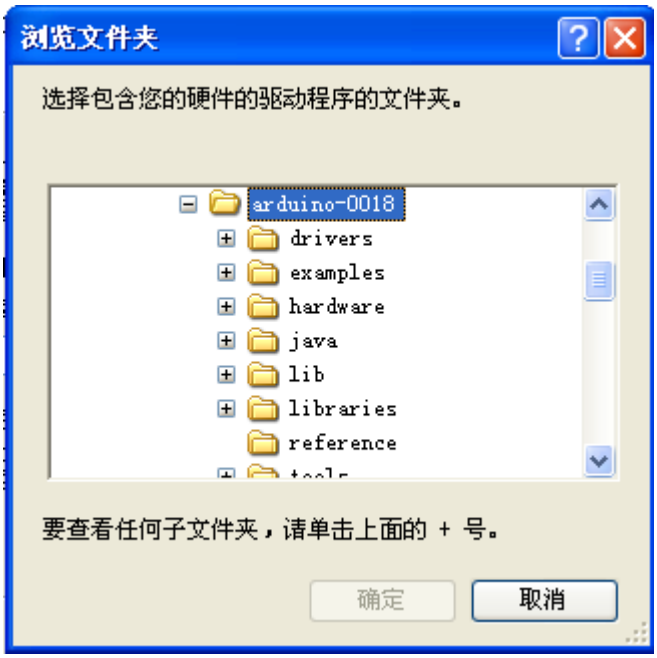
• 将数据线的扁口一端插在电脑的 USB 接口上，如下图所示：

• 插好后，Arduino328 控制板上的电源指示灯会被点亮，电脑上会出现一个对话框如图：

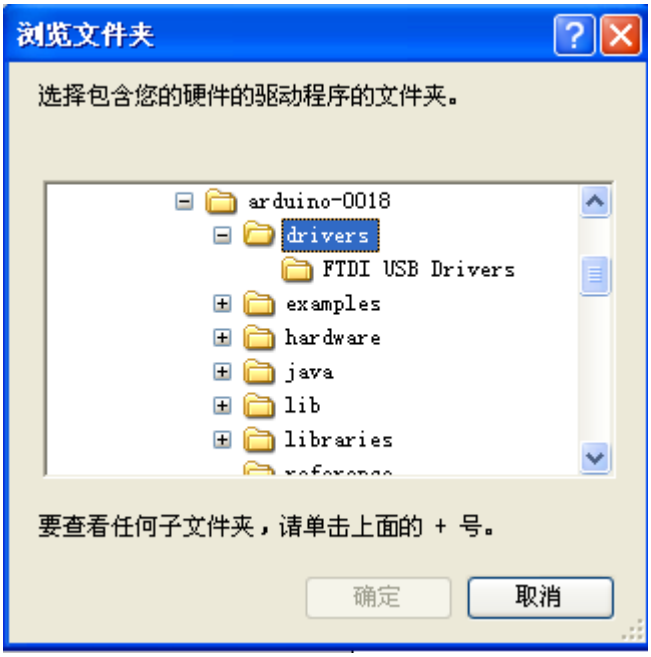
• 选择从列表或指定位置安装，点击下一步，出现如下图：



• 然后点击浏览，在出现的浏览文件夹对话框中点击光盘，在光盘下找到 arduino0018 文件夹，点击打开，会看见有 drivers 文件夹如下图所示：



• 点击 drivers 文件夹，会看到 FTDI USB Drivers 文件夹，如图：

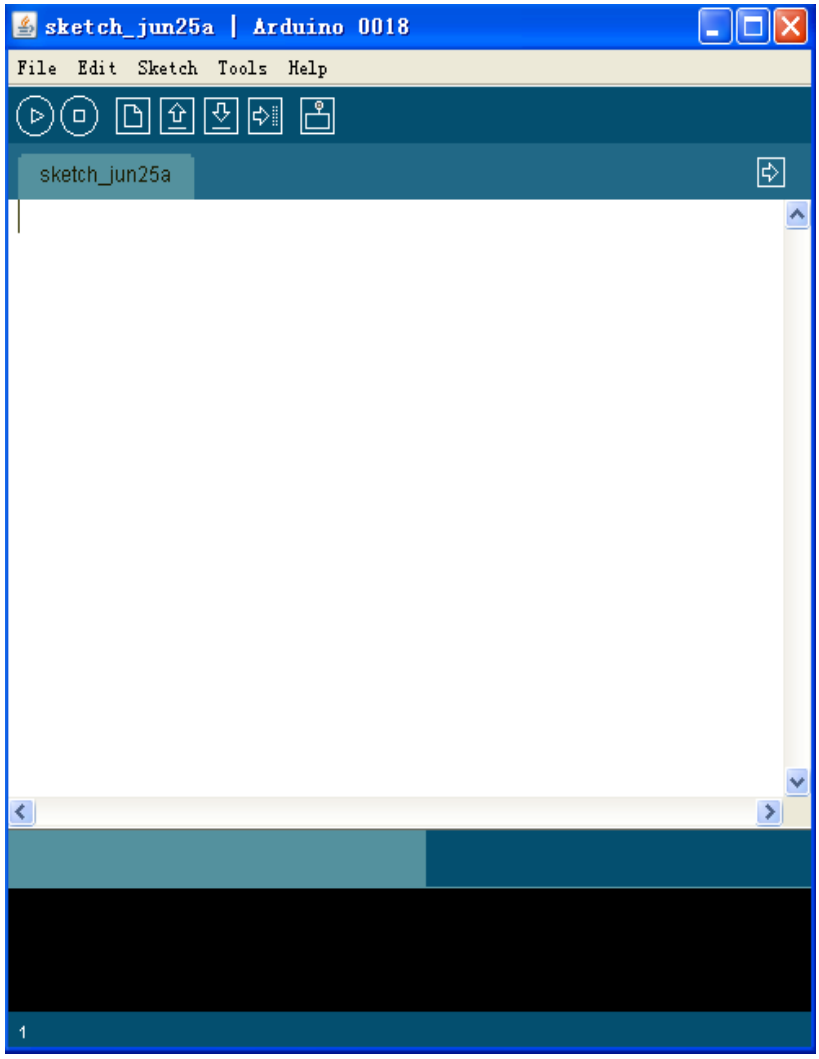


• 然后点击这个文件夹，接着点击确定，点击下一步，会出现如图对话框：

• 这时我们只要等待即可，稍后会出现如下图对话框：

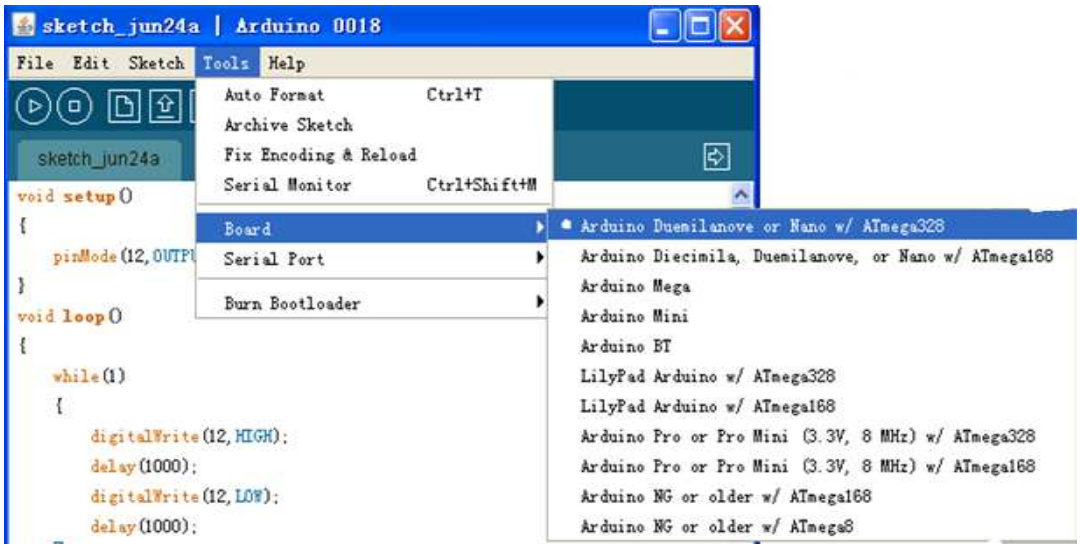
• 点击完成，这样驱动就安装好了，下次再将数据线插到电脑就不会出现安装驱动对话框了，插上数据线就可以下载程序了。

然后下载最新的编译软件 Arduino0018，解压后就可以直接使用，接下来我们就可以开始练手了，打开 Arduino0018。



Arduino 0018 开发编译环境很简洁，各个功能键功能描述如下：

接下来点 Tools->Board 选择开发板型号，



这里我们使用的是 Arduino Duemilanove 328，然后点 Tools->Serisl Port 选择串口，即 USB 映射的串口地址，

前面的工作做好后，我们就可以进入实验阶段了。

Arduino 语言是以 setup()开头，loop()作为主体的一个程序构架。官方网站是这样描述 setup()的：用来初始化变量，管脚模式，调用库函数等等，此函数只运行一次。loop()函数是一个循环函数，函数内的语句周而复始的循环执行，功能类似 c 语言中的“main();”。

Digital Output 数字输出实验

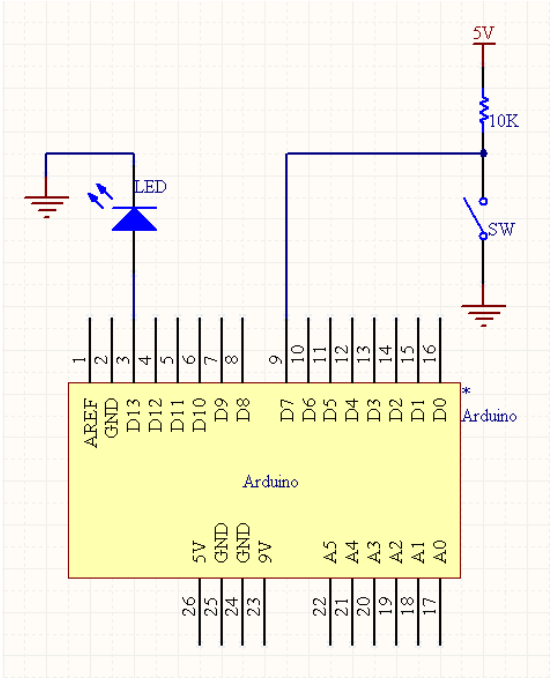
功能：使 PIN13 脚上的 LED 闪烁。

```
int ledPin = 13;                // 设定控制 LED 的数字 IO 脚
void setup()
{
   pinMode(ledPin, OUTPUT);     // 设定数字 IO 口的模式，OUTPUT 为输出
}
void loop()
{
   digitalWrite(ledPin, HIGH);  // 设定 PIN13 脚为 HIGH = 4V
   delay(1000);                 // 设定延时时间，1000 = 1 秒
   digitalWrite(ledPin, LOW);   // 设定 PIN13 脚为 LOW = 0V
   delay(1000);                 //设定延时时间
}
```

功能：PWM 使 PIN11 脚上的 LED 逐渐变亮逐渐变暗。

```
int ledPin = 11;        // 设定控制 LED 的数字 IO 脚
int val;                //定义一个变量
void setup()
{
   pinMode(ledPin, OUTPUT);   // 设定数字 IO 口的模式，OUTPUT 为输出
}
void loop()
{
  for(val=0;val<255;val++)      //变量循环+1
    {
     analogWrite(ledPin, val); //PWM 输出
     delay(50);                //设定延时时间
    }
  for(val=255;val>0;val--)     //变量循环-1
    {
     analogWrite(ledPin, val);
     delay(50);
    }
}
```

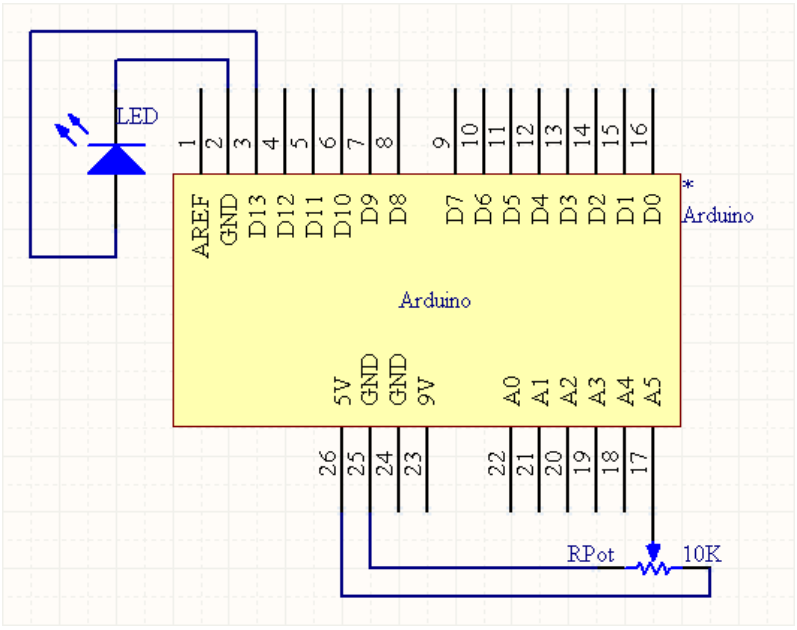
Digital Input 数字输入实验



功能：闭合开关 SW，PIN13 脚上的 LED 亮，断开开关 SW，PIN13 脚上的 LED 灭。

```
int ledPin = 13;            // 设定控制 LED 的数字 IO 脚
int switchPin = 7;          // 设定开关的数字 IO 脚
int val = 0;                //定义一个变量
void setup()
{
  pinMode(ledPin, OUTPUT);      // 设定数字 IO 口的模式，OUTPUT 为输出
  pinMode(switchPin, INPUT);    // 设定数字 IO 口的模式，INPUT 为输入
}
void loop()
{
  val = digitalRead(switchPin);                    //读数字 IO 口上的状态
  if (HIGH == val)   digitalWrite(ledPin, LOW);    // 如果开关断开，LED 灭
  else   digitalWrite(ledPin, HIGH);               // 如果开关闭合，LED 亮
}
```

Analog Input 模拟输入实验



功能：调节电位器 RPot，使输入模拟量的变化来改变 LED 闪烁的频率。

```
int ledPin = 13;                  // 设定控制 LED 的数字 IO 脚
int RPotPin = 5;                  // 设定模拟输入 IO 脚
int val = 0;                      //定义一个变量
void setup()
{
  pinMode(ledPin, OUTPUT);        // 设定数字 IO 口的模式，OUTPUT 为输出
}
void loop()
{
  val = analogRead(RPotPin);      //读模拟 IO 口上的数据
  digitalWrite(ledPin, HIGH);     // 设定 PIN13 脚为 HIGH = 4V
  delay(value);                   // 延时时间为模拟输入的值
  digitalWrite(ledPin, LOW);      // 设定 PIN13 脚为 LOW
  delay(value);                   // 延时时间为模拟输入的值
}
```

第三课 Arduino 的串口通讯

Arduino 不但有 14 个数字接口和 6 个模拟接口外，还有 1 个更为常用的串口接口。在实际应用中串口以只需要少量的几根线就能和其他串口设备通讯的优势被广应用。

串行接口按标准被分为 RS-232、RS-422、RS-485。RS-232 是在 1962 年发布的，也是目前 PC 机与通信工业中应用最广泛的一种串行接口，RS-232 采取不平衡传输方式，即所谓单端通讯。典型的 RS-232 信号在正负电平之间摆动，在发送数据时，发送端驱动器输出正电平在+5～+15V，负电平在-5～-15V 电平。我们的单片机使用的是 TTL 电平的串行协议，因此单片机与 pc 通讯时需要进行 RS-232 电平和 TTL 电平的转换，最常用的电平转换芯片是 MAX232，单片机与单片机通讯时则可以直接连接。

USB 版本的 Arduino 则是通过 USB 转成 TTL 串口下载程序的，数字口 PIN 0 和 PIN 1 就是 TTL 串口 RX 和 TX。

串口通讯中最重要的一点就的通讯协议，一般串口通讯协议都会有波特率、数据位、停止位、校验位等参数。大家不会设置也不用怕，Arduino 语言中 Serial.begin()函数就能使大家轻松完成设置，我们只需要改变该函数的参数即可，例如 Serial.begin(9600)，则表示波特率为 9600bit/s(每秒比特数 bps)，其余参数默认即可。

Arduino 语言中还提供了 Serial.available() 判断串口缓冲器状态、Serial.read()读串口、Serial.print()串口发送及 Serial.println()带换行符串口发送四个函数。

下面我们用一段代码来演示这些函数的用途。实验无须外围电路，只需要将下载的 USB 线连接即可。

```
char word;
void setup()
{
    Serial.begin(9600);        // 打开串口，设置波特率为 9600 bps
}
void loop()
{
  if (Serial.available() > 0) //判断串口缓冲器是否有数据装入
   {
     word = Serial.read();     //读取串口
     if(word=='a')             //判断输入的字符是否为 a
      {
        Serial.print("Robot ");   //从串口发送字符串
        Serial.println("is NO.1");  //从串口发送字符串并换行
      }
   }
}
```

编译下载完程序后，点红圈里的按钮，打开串口监视器；

选择串口监视器的波特率为 9600bps，在发送框里填上字母 a，点 send 发送，下面的显示框里就会显示返回的数据，如果发送其他字母，则无返回值。

第四课 Arduino 的 I2C/TWI 通讯

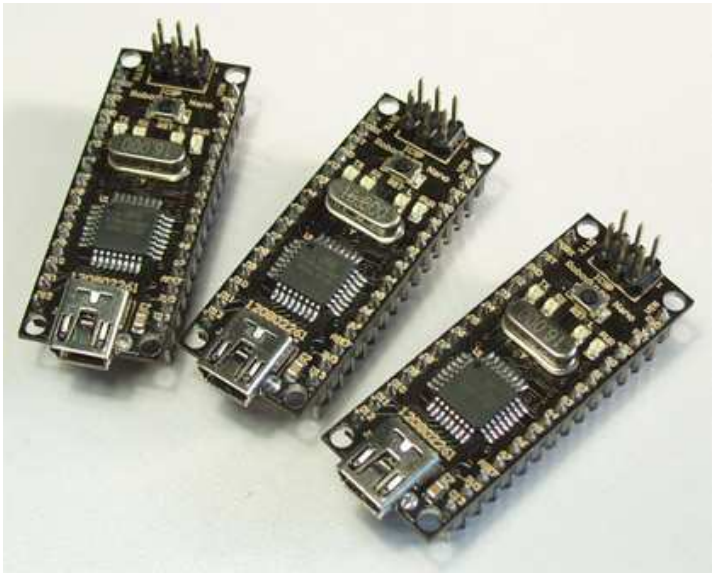
什么的 I2C 呢？I2C 即 Inter－Integrated Circuit 串行总线的缩写，是 PHILIPS 公司推出的芯片间串行传输总线。它以 1 根串行数据线（SDA）和 1 根串行时钟线（SCL）实现了双工的同步数据传输。具有接口线少，控制方式简化，器件封装形式小，通信速率较高等优点。在主从通信中，可以有多个 I2C 总线器件同时接到 I2C 总线上，通过地址来识别通信对象。

幸运的是，Arduino 已经为我们提供了 I2C 的库函数（Wire.h），这样我们就可以很轻松的玩 IIC 通讯了 。

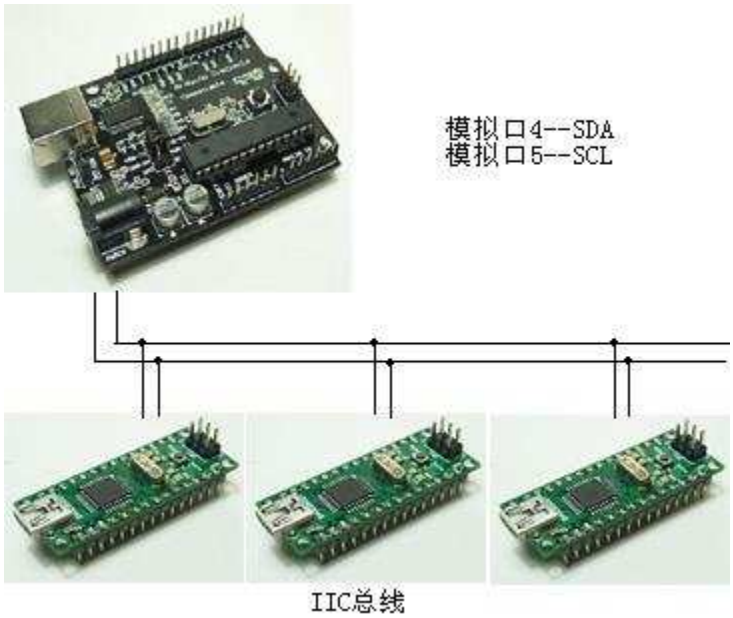
这里使用 Arduino Diecimila 做主机，2 个 Arduino Nano 做从机（不一定非要 nano 做从机，Arduino Diecimila、Mini 甚至其他 IIC 器件都可以。如果 IIC 总线上挂了多个从机，那么就要在总线上加上拉电阻。），Arduino Nano 是 Arduino 家族中的新成员，估计大家还不太熟悉吧，这里我就简单介绍一下。

Arduino Nano 实际和 Arduino Diecimila 差不多，但 Arduino Nano 与 Arduino Diecimila 相比较，Arduino Nan

o 在体积上占很大优势，并且具有 8 个模拟口，比 Arduino Diecimila 还多 2 个，还具有 USB 电源和外接电源自动切换功能，12 版的编译环境支持 nano 硬件。



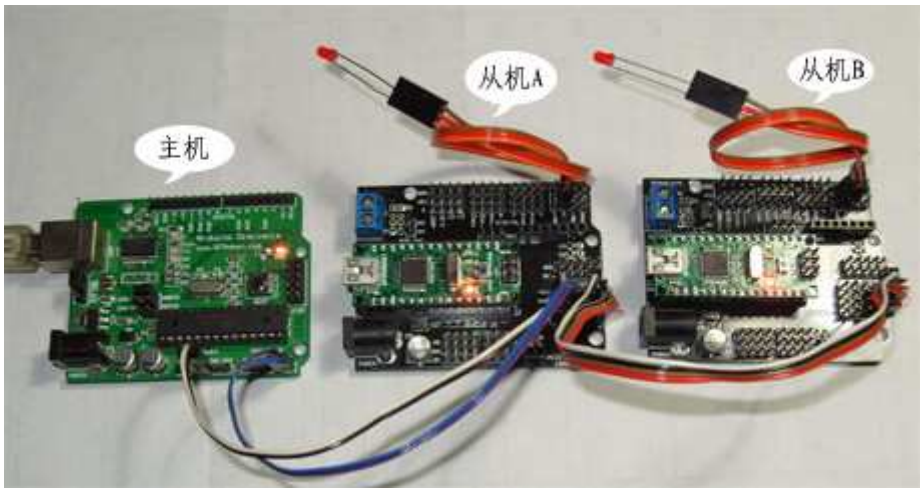
I2C 总线也是主从方式通讯，I2C 总线允许连接多个微控制器，显然不能同时存在两个主器件，先控制总线的器件成为主器件，这就是总线竞争。在竞争过程中数据不会被破坏、丢失。数据只能在主、从器件中传送，结束后，主、从器件将释放总线，退出主、从器件角色。



IIC总线

下面我们就做个简单的 I2C 通讯实验，通过编译环境的串口监视器向主机发送指令，主机收到后，再通过 I2C 总线发送给地址匹配的从机，然后从机驱动 LED 点亮。代码分主从部分，主机部分写入 Arduino Diecimila，从机部分写入 Arduino Nano。

实验实物图：



在上代码之前，我们先了解一下 IIC 的库函数 Wire.h 里面的常用函数。

- begin()　　//初始化 Wire 库，和设置 IIC 总线主从机
- begin(address)　　//带地址参数就是从机，不带就是主机
- requestFrom(address, count)　　//在启动 IIC 总线后，可以继续访问另一个地址，和访问次数
- beginTransmission(address)　　//开始给从机发送地址
- endTransmission()　　//结束本次 IIC 通讯，与上条函数成对使用
- send()　　//发送数据
- byte available()　　//用于判断数据是否有效，有效才开始接收
- byte receive()　　//接收数据
- onReceive(handler)　　//从机接收主机发来的数据
- onRequest(handler)　　//从机请求主机发送数据

主机代码：（从编译器串口监视器发送数字 1，2，3，4 来控制从机的 LED 亮与灭）

```
#include <Wire.h>
void setup()
{
  Wire.begin();             //启动 I2C 总线，地址缺省表示为主机
  Serial.begin(9600);       //启动串口，设置波特率为 9600
  Serial.println("Ready"); //发送字符
}
void loop()
{
  int val;
  if(Serial.available() > 0)  //判断串口缓冲器是否有数据装入
   {
      val=Serial.read();      //读串口
      if(val==49)//1
        {
          Wire.beginTransmission(4);  // 与地址 4 的从机连接通讯
```

```
        Wire.send(1);               // 发送数字  1 开 LED
        Wire.endTransmission();     // 停止发送
        Serial.println("49 OK");    // 串口上显示 49 OK 表示完成，49 表示为数字 1 的 ASCII 码
        delay(10);                  //
      }
    else if(val==50)//2
      {
        Wire.beginTransmission(4);  // 与地址 4 的从机连接通讯
        Wire.send(0);               // 发送数字  0 关 LED
        Wire.endTransmission();     // 停止发送
        Serial.println("50 OK");    // 串口上显示 50 OK 表示完成
        delay(10);
      }
    else if(val==51)//3
      {
        Wire.beginTransmission(5);  // 与地址 5 的从机连接通讯
        Wire.send(1);               // 发送数字  1 开 LED
        Wire.endTransmission();     // 停止发送
        Serial.println("51 OK");    // 串口上显示 51 OK 表示完成
        delay(10);
      }
    else if(val==52)//4
      {
        Wire.beginTransmission(5);  // 与地址 5 的从机连接通讯
        Wire.send(0);               // 发送数字  0 关 LED
        Wire.endTransmission();     // 停止发送
        Serial.println("52 OK");    // 串口上显示 52 OK 表示完成
        delay(10);
      }
     else Serial.println(val);
  }
}
```

从机 A 代码：（接收到主机发送的 1 点亮 LED，接收到 0 关掉 LED）

```
#include <Wire.h>
int LED = 2;
void setup()
{
  Wire.begin(4);                    // 设置从机地址为 4
  Wire.onReceive(receiveEvent);     //从机接收主机发来的数据
  pinMode(LED,OUTPUT);              // 设置 IO 口为输出模式
}
```

```
void loop()
{
  delay(100);
}
void receiveEvent(int howMany)   // 接收从主机发过来的数据
{
    int c = Wire.receive();      // 接收单个字节
    if(c==1)
      {
         digitalWrite(LED,HIGH); // 如果为 1 开 LED
      }
    else if(c==0)
      {
         digitalWrite(LED,LOW);  // 如果为 0 关 LED
      }
}
```

从机 B 代码：（接收到主机发送的 1 点亮 LED，接收到 0 关掉 LED）

```
#include <Wire.h>
int LED = 2;
void setup()
{
  Wire.begin(5);                // 设置从机地址为 5
  Wire.onReceive(receiveEvent); //
  pinMode(LED,OUTPUT);
}
void loop()
{
  delay(100);
}
void receiveEvent(int howMany)
{
    int c = Wire.receive();
    if(c==1)
      {
         digitalWrite(LED,HIGH);
      }
    else if(c==0)
      {
         digitalWrite(LED,LOW);
      }
}
```

第五课 Arduino 控制 1602 字符液晶显示

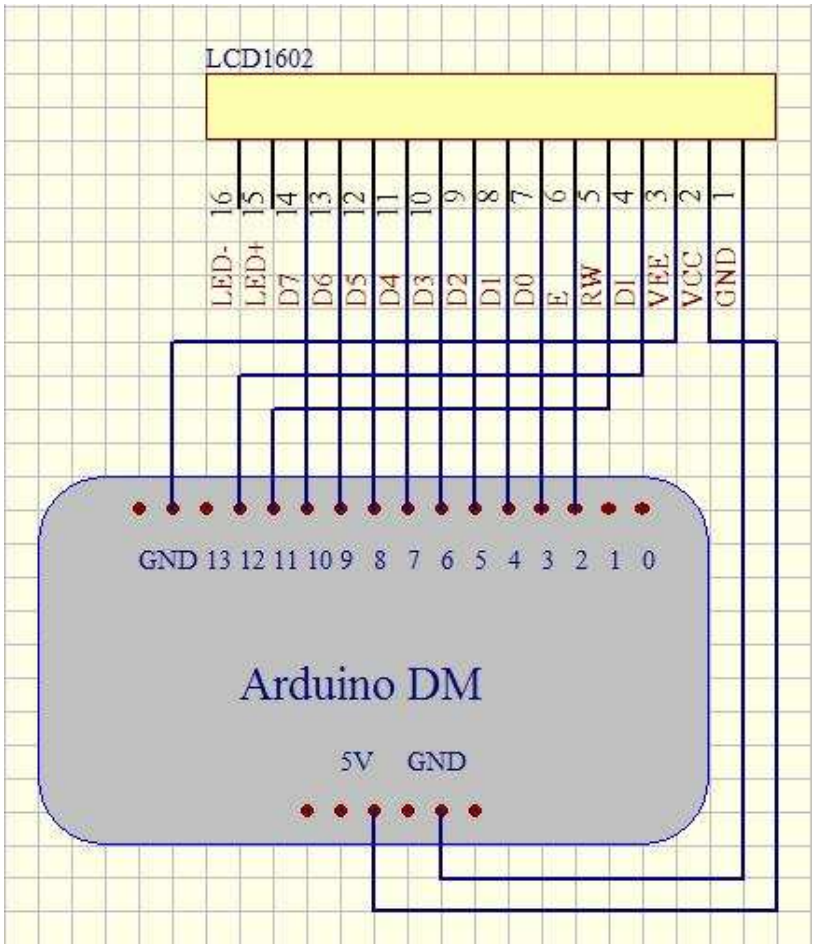
1602 字符液晶是最常用的一种，很具有代表性，1602 液晶分 4 总线和 8 总线 2 种驱动方式（关于该液晶的详细资料，大家可以自己搜索，这里就不做详细说明了）。我们用单片机驱动 1602 液晶，使用并口操作很容易就驱动起来了，但使用 Arduino 板驱动 1602 液晶，还真有点费劲，因为他只能位操作。根据官方网站提供的例程，很容易看出他们使用的是最常用的 8 总线驱动方式，然而他巧妙的使用 for 循环语句完成了位操作的赋值。来看看官方的工程代码：

```
int DI = 12;
int RW = 11;
int DB[] = {3, 4, 5, 6, 7, 8, 9, 10};//使用数组来定义总线需要的管脚
int Enable = 2;
void LcdCommandWrite(int value) // poll all the pins
{
 int i = 0;
 for (i=DB[0]; i <= DI; i++)   //总线赋值
{
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);        // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);        // pause 1 ms according to datasheet
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);        // pause 1 ms according to datasheet
}
void LcdDataWrite(int value)  // poll all the pins
{
 int i = 0;
 digitalWrite(DI, HIGH);
 digitalWrite(RW, LOW);
 for (i=DB[0]; i <= DB[7]; i++) {
   digitalWrite(i,value & 01);
   value >>= 1;
 }
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);       // send a pulse to enable
 digitalWrite(Enable,HIGH);
 delayMicroseconds(1);
 digitalWrite(Enable,LOW);
 delayMicroseconds(1);      // pause 1 ms according to datasheet
}
```

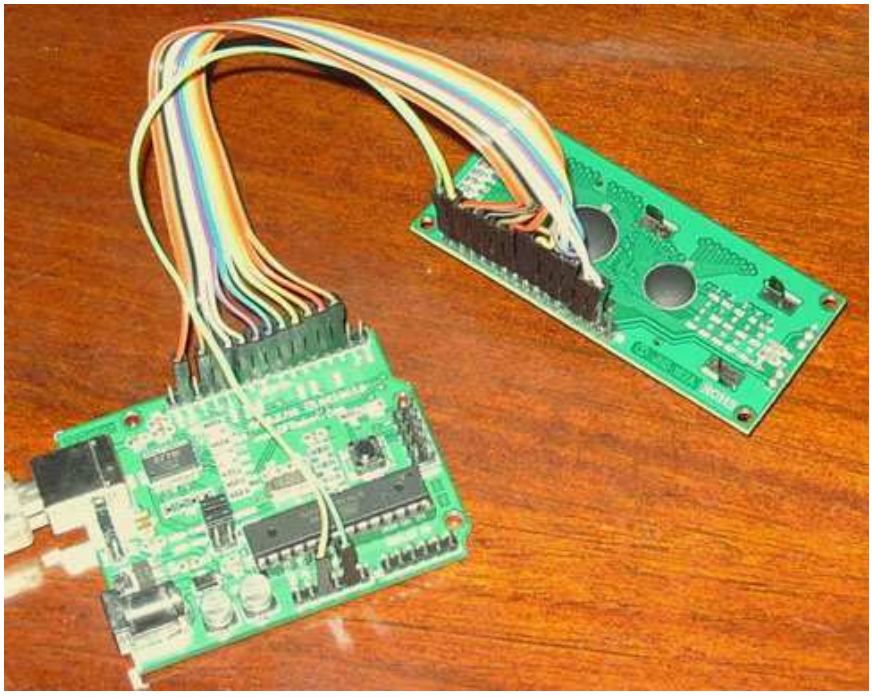
```
void setup (void) {
 int i = 0;
 for (i=Enable; i <= DI; i++)
   {
     pinMode(i,OUTPUT);
   }
 delay(100);             // initiatize lcd after a short pause   needed by the LCDs controller
 LcdCommandWrite(0x38);  // set:   // 8-bit interface, 1 display lines, 5x7 font
 delay(20);
 LcdCommandWrite(0x06);  // entry mode set:   // increment automatically, no display shift
 delay(20);
 LcdCommandWrite(0x0E);  // display control:  // turn display on, cursor on, no blinking
 delay(20);
 LcdCommandWrite(0x01);  // clear display, set cursor position to zero
 delay(100);
 LcdCommandWrite(0x80);  // display control:  // turn display on, cursor on, no blinking
 delay(20);
}
void loop (void) {
  LcdCommandWrite(0x02); // set cursor position to zero
  delay(10);             // Write the welcome message
  LcdDataWrite('H');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  LcdDataWrite(' ');
  LcdDataWrite('C');
  LcdDataWrite('a');
  LcdDataWrite('r');
  LcdDataWrite('a');
  LcdDataWrite('c');
  LcdDataWrite('o');
  LcdDataWrite('l');
  LcdDataWrite('a');
  delay(500);
}
```

实验器材：Arduino DM 一个，USB 电缆一根，LCD1602 一个，单芯杜邦线若干。

根据例程定义接线图如下：



大家需要注意的是，液晶根据不同的颜色不同的型号，对比度（VEE）调节电压也不同，一般都需要接个电位器进行调节，本实验使用的是灰膜液晶，VEE 直接接到地即可。



根据我们使用的液晶更改功能初始化设置，指令集如下表：

1602型液晶屏内部的控制器共有11条控制指令，如表4所示，

序号	指令	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	置功能	0	0	0	0	1	DL	N	F	*	*
7	置字符发生存储器地址	0	0	0	1	字符发生存储器地址(AGG)					
8	置数据存储器地址	0	0	1	显示数据存储器地址(ADD)						
9	读忙标志或地址	0	1	BF	计数器地址(AC)						
10	写数到CGRAM或DDRAM	1	0	要写的数							
11	从CGRAM或DDRAM读数	1	1	读出的数据							

它的读写操作、屏幕和光标的操作都是通过指令编程来实现的。（说明：1 为高电平、0 为低电平）

指令 1：清显示，指令码 01H,光标复位到地址 00H 位置

指令 2：光标复位，光标返回到地址 00H

指令 3：光标和显示模式设置 I/D：光标移动方向，高电平右移，低电平左移 S:屏幕上所有文字是否左移或者右移。高电平表示有效，低电平则无效

指令 4：显示开关控制。 D：控制整体显示的开与关，高电平表示开显示，低电平表示关显示 C：控制光标的开与关，高电平表示有光标，低电平表示无光标 B：控制光标是否闪烁，高电平闪烁，低电平不闪烁

指令 5：光标或显示移位 S/C：高电平时移动显示的文字，低电平时移动光标

指令 6：功能设置命令 DL：高电平时为 4 位总线，低电平时为 8 位总线 N：低电平时为单行显示，高电平时双行显示 F: 低电平时显示 5x7 的点阵字符，高电平时显示 5x10 的点阵字符

指令 7：字符发生器 RAM 地址设置

指令 8：DDRAM 地址设置

指令 9：读忙信号和光标地址 BF：为忙标志位，高电平表示忙，此时模块不能接收命令或者数据，如果为低电平表示不忙。

指令 10：写数据

指令 11：读数据

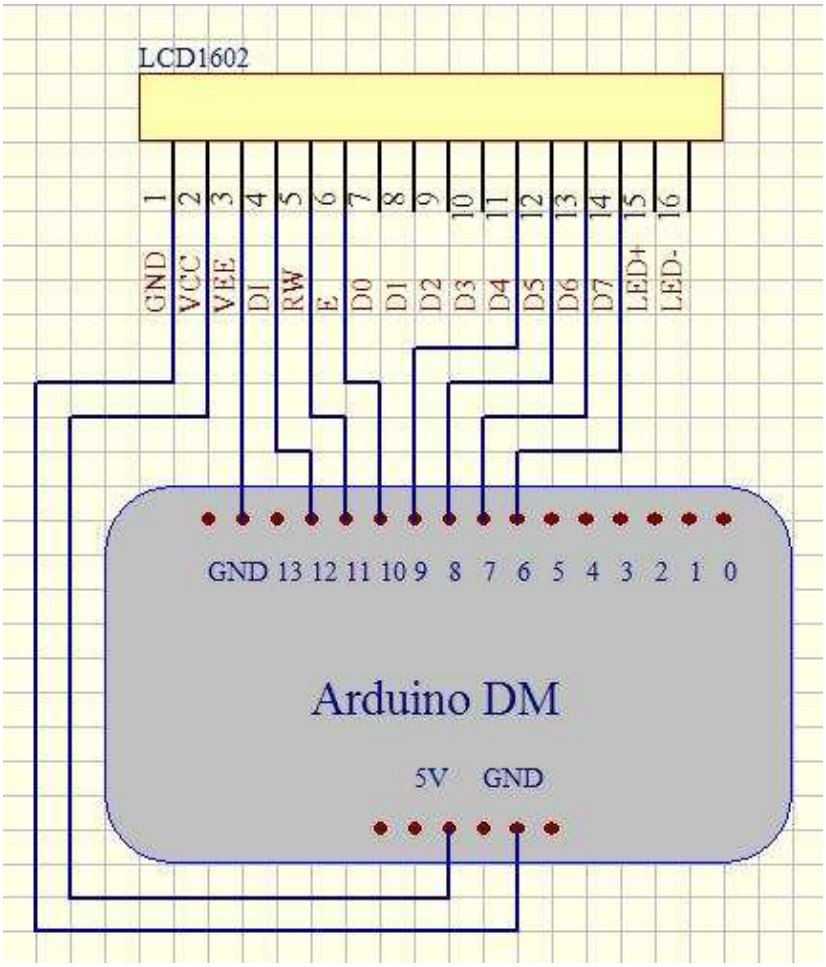
Arduino DM 的数字端口只有 14 个，那么 LCD1602 就占用了 10 个，浪费了不少资源，对我们以后的电路扩展带来了极大的不便。于是我就可以使用该液晶的 4 总线驱动方式，可以节省 3 个数字端口出来作其他扩展。本人编写的工程代码如下：

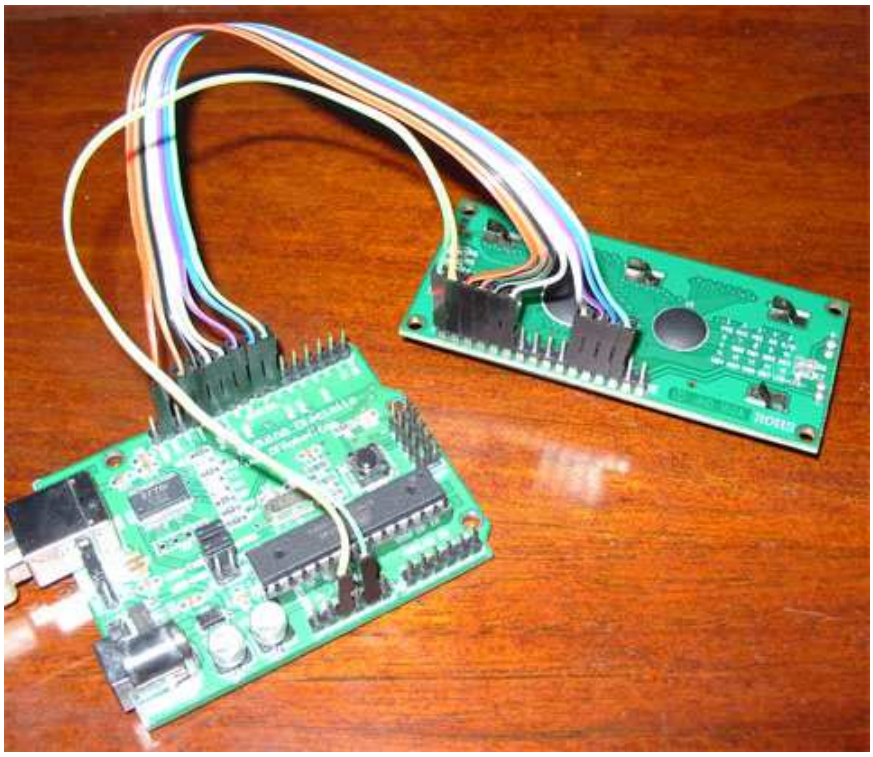
```
/*********************************************************************/
int LCD1602_RS=12;
int LCD1602_RW=11;
int LCD1602_EN=10;
int DB[] = { 6, 7, 8, 9};          //定义 4 总线数字 IO 口
char str1[]="Welcome to";          //第一行显示内容
char str2[]="Robot.cn";     //第二行显示内容

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

/*********************************************************************/
void LCD_Data_Write(int dat)          //写数据
{
 int i=0,temp;
```

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


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


/*********************************************************************/
void LCD_Write_Char( int x,int y,int dat)   //写字符
{
  LCD_SET_XY( x, y );
  LCD_Data_Write(dat);
}


/*********************************************************************/
void LCD_Write_String(int X,int Y,char *s)  //写字符串
```

```
{
    LCD_SET_XY( X, Y );    //设置地址
    while (*s)             //写字符串
    {
      LCD_Data_Write(*s);
      s ++;
    }
}

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


/*****************************************************************/
void loop (void)
{
   LCD_Command_Write(0x02);   //光标返回首地址
   delay(50);
   LCD_Write_String(3,0,str1);//第 1 行，第 4 个地址起
   delay(50);
   LCD_Write_String(1,1,str2);//第 2 行，第 2 个地址起
   while(1);
}
```

根据程序定义接线图如下：


LCD1602
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
GND VCC VEE D1 RW E D0 D1 D2 D3 D4 D5 D6 D7 LED+ LED-
GND 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Arduino DM
5V GND



编译后，下载进 Arduino，显示如图：

这个代码容易移植，以后需要用到显示的地方，都可以直接引用。

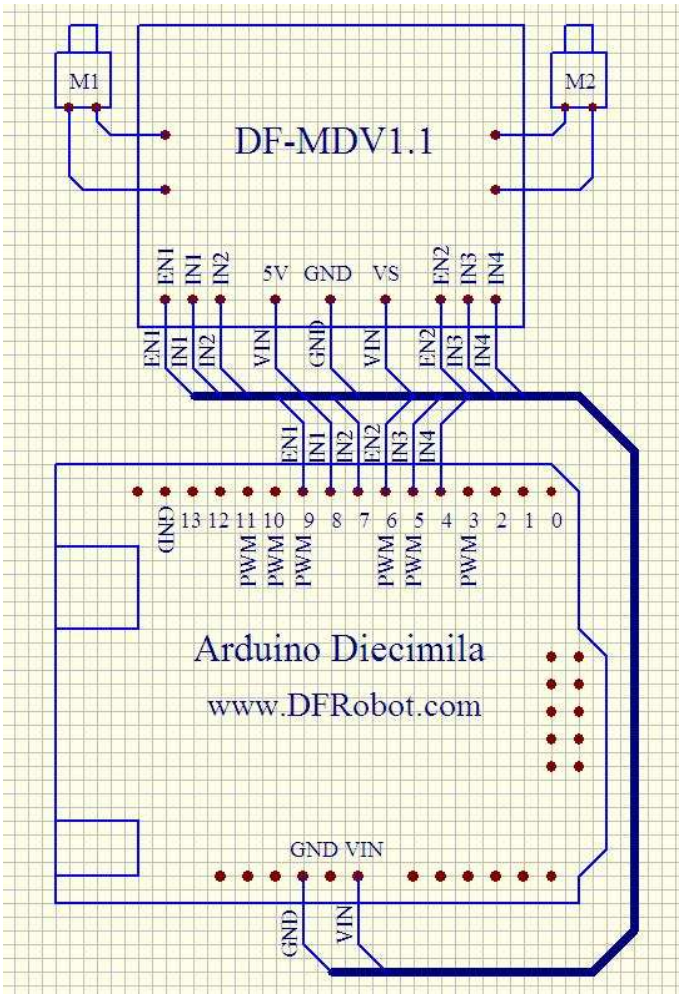
Arduino 语言关于字符液晶 4 总线的库函数，使用该库函数程序就变的更简洁了。函数例程如下：

```
#include <LCD4Bit.h>
LCD4Bit lcd = LCD4Bit(1);   //传递参数 1 和 2，表示选择 1 线和 2 线字符液晶。
lcd.clear();                //清屏
delay(1000);                //延时
lcd.printIn("arduino");     //显示字符串
lcd.cursorTo(2, 0);         //line=2, x=0
lcd.leftScroll(20, 50);     //將 LCD 上现有的文字以 50ms 的速度向左移动 20 个字符位
```

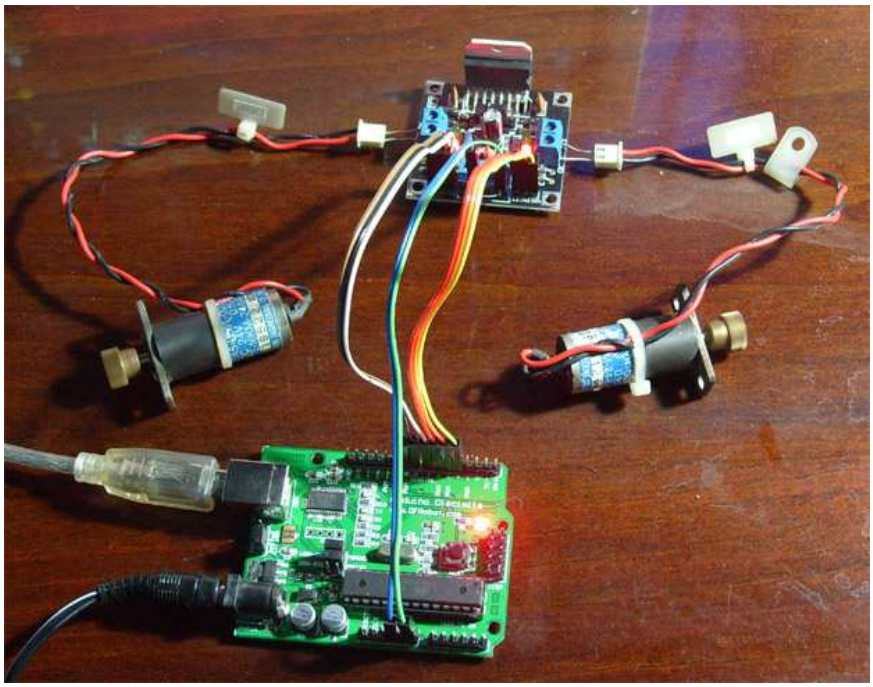
注意:若使用 LCD4Bit.h 的话,RW 需要接地.

第六课 Arduino 驱动直流电机

Arduino 控制器并不能直接驱动直流电机转动，需要配合一个电机驱动器，这里使用的是本公司的 2A 大电流电机驱动模块 DF-MDV1.2，电路连接非常简单。

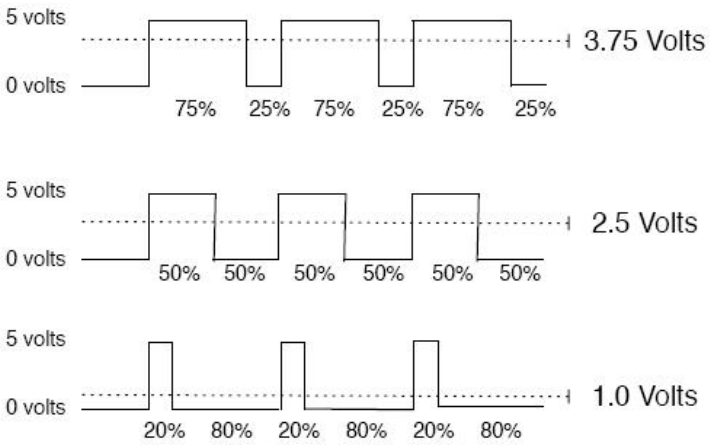


注意：按上面图连接的话，Arduino 在控制电机时需要使用外接直流电源（7V～12V），不可使用 USB 供电。



根据上图的连线方法，我们可以控制电机的正转，反转，停止，以及 PWM 调速。PWM 调速是通过调整空占比来模拟不同电压值，从而控制加到电机两端的电压高低来实现调速。

output_voltage = (on_time / off_time) * max_voltage



PWM 调速实验代码：

```
int EN1 = 9;                //使能 1
int IN1 = 8;
int IN2 = 7;
int EN2 = 6;                //使能 2
int IN3 = 5;
int IN4 = 4;
void setup()
{
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
}
void loop()
{
  int value;
  for(value = 0 ; value <= 255; value+=5)
  {
    digitalWrite(IN1,HIGH);       // IN1（IN3）和 IN2（IN4）必须相反，才能使电机转动
    digitalWrite(IN2,LOW);        // 改变电平方向，电机反转
    digitalWrite(IN3,HIGH);       // 电平相同，电机停止
    digitalWrite(IN4,LOW);
    analogWrite(EN1, value);      //PWM 调速
    analogWrite(EN2, value);      //PWM 调速
    delay(30);
  }
}
```

第七课 Arduino 读红外测距传感器 GP2D12

Arduino 读红外测距传感器 GP2D12 实例，仅供大家参考！

器材：Arduino 开发板，GP2D12，1602 字符液晶，连接线若干。

GP2D12 是日本 SHARP 公司生产的红外距离传感器，价格便宜，测距效果还不错，主要用于模型或机器人制作。

技术规格如下：

探测距离：10-80cm

工作电压：4-5.5V

标准电流消耗：33-50 mA

输出量：模拟量输出，输出电压和探测距离成比例

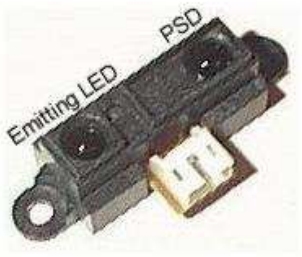


Fig. 1 - Sharp GP2D12 distance sensor

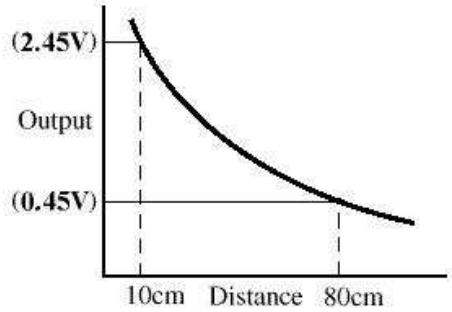
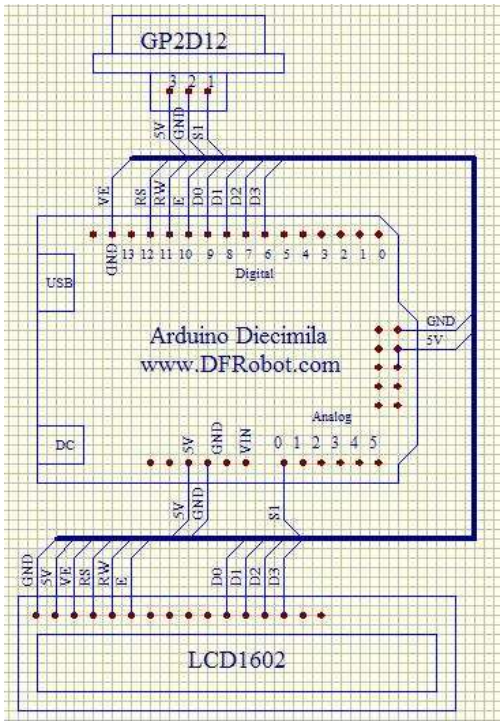


Fig. 2- Sharp GP2D12 output pattern

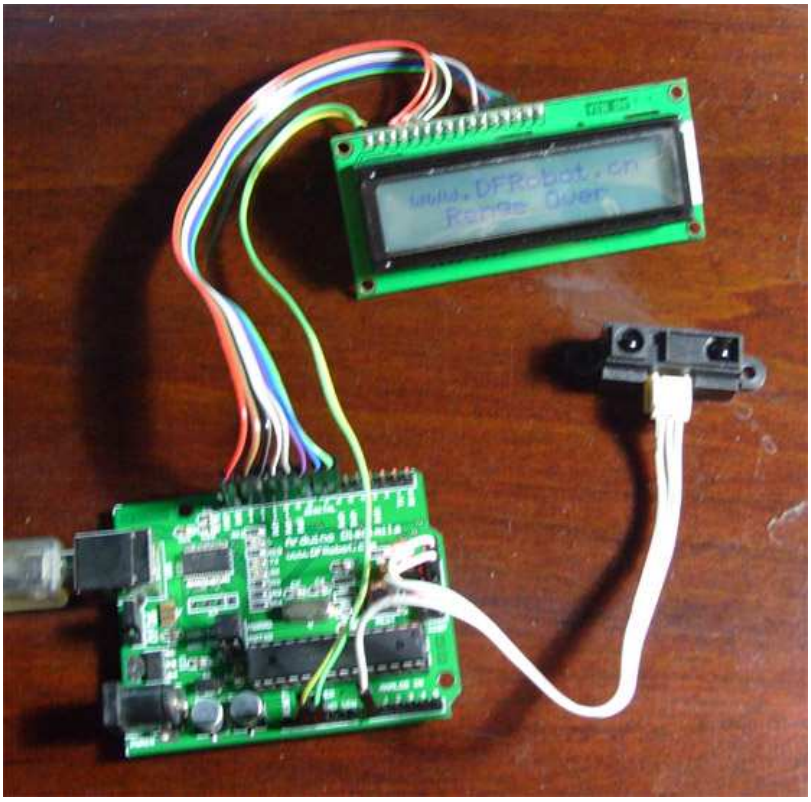
从曲线图中我们可以看出，输出电压并非是线性的，所以这个测距也就只能测个大概，如果要精度高的话就需要做非线性校正，这里我们就不讨论这个问题了。

实验原理：GP2D12 根据距离的远近输出相应的电压，经 Arduino 开发板 0 号模拟口输入，转换成数字量，根据公式计算得到需要显示的数据。

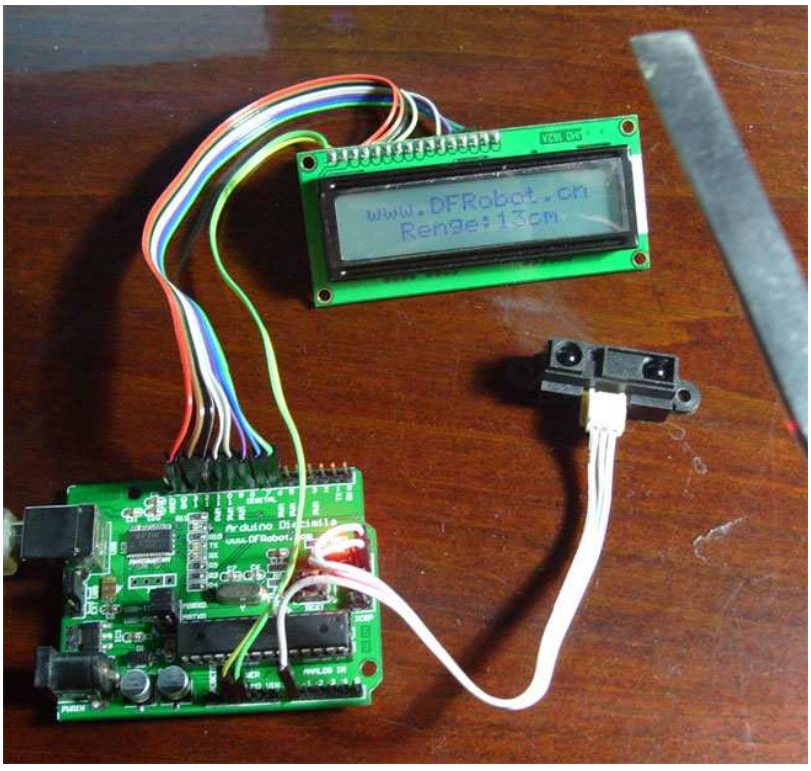
实验原理图：



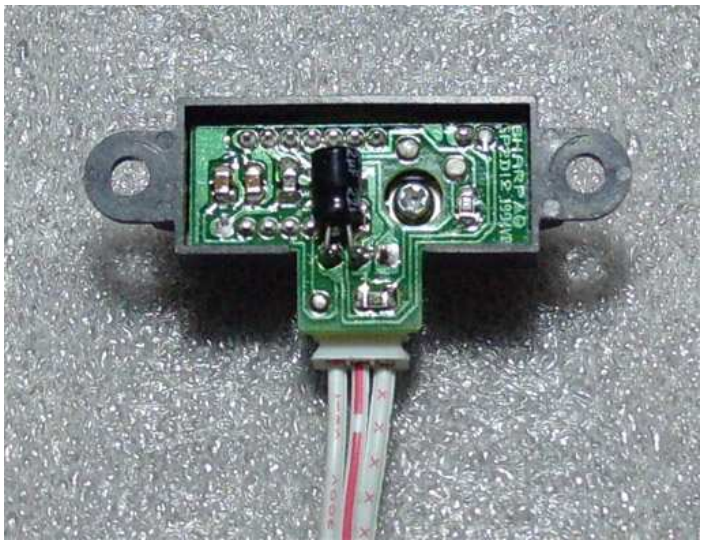
实验实物图：



没有障碍物，显示范围超出，



有障碍物时显示测量距离。



经实验，需要在 GP2D12 的电源端加个 10UF 以上的电解电容，稳定供电电压，以保证输出模拟电压更稳定。

实验代码：

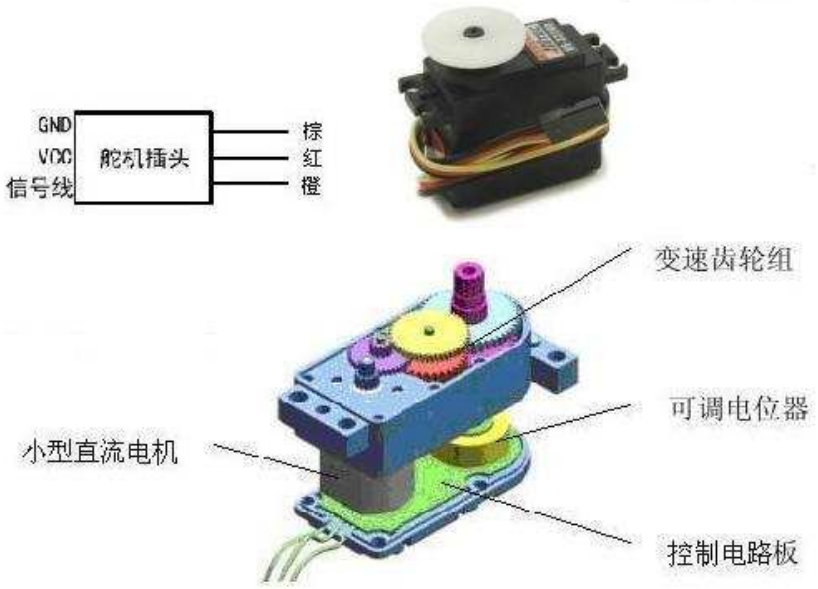
```
/*********************************************************************/
int GP2D12=0;
int ledpin = 13;
int LCD1602_RS=12;
int LCD1602_RW=11;
int LCD1602_EN=10;
int DB[] = {6, 7, 8, 9};
char str1[]="www.Robot.cn";
char str2[]="Renge:00cm";
char str3[]="Renge Over";

/*********************************************************************/
由于篇幅限制，1602 液晶部分程序省略。。。
/*********************************************************************/
void setup (void)
{
  int i = 0;
  for (i=6; i <= 13; i++)
   {
     pinMode(i,OUTPUT);
   }
  LCD_Command_Write(0x28);//4 线 2 行 2x7
  delay(50);
  LCD_Command_Write(0x06);
  delay(50);
  LCD_Command_Write(0x0c);
```

```
  delay(50);
  LCD_Command_Write(0x80);
  delay(50);
  LCD_Command_Write(0x01);
}

/*********************************************************************/
void loop (void)
{
   float temp;
   int val;
   char i,a,b;
   LCD_Command_Write(0x02);
   delay(50);
   LCD_Write_String(1,0,str1);
   delay(50);
   LCD_Write_String(3,1,str2);
   delay(50);
   while(1)
    {
      val = analogRead(GP2D12);
      temp=val/5.8;   //改变被除数，可以减小一点误差。
      val=95-temp;    //由于 GP2D12 的输出电压与距离成反比，所以需要用一个常量相减
                      //改变这个常量，可以减小一点误差。
      if(val>80)
        {
          LCD_Write_String(3,1,str3);       //超出范围显示  Renge Over
        }
      else
        {
          LCD_Write_String(3,1,str2);
          a=0x30+val/10;
          b=0x30+val%10;
          LCD_Write_Char(9,1,a);
          LCD_Write_Char(10,1,b);
        }
      delay(500);
     }
}
```

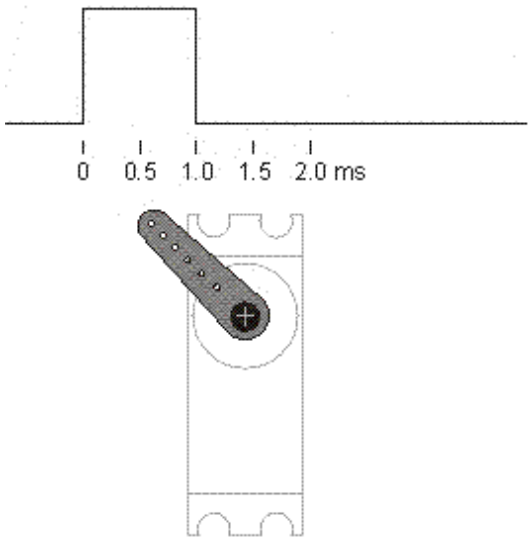
第八课 Arduino 控制舵机

舵机，又称伺服马达，是一种具有闭环控制系统的机电结构。舵机主要是由外壳、电路板、无核心马达、齿轮与位置检测器所构成。其工作原理是由控制器发出 PWM（脉冲宽度调制）信号给舵机，经电路板上的 IC 处理后计算出转动方向，再驱动无核心马达转动，透过减速齿轮将动力传至摆臂，同时由位置检测器（电位器）返回位置信号，判断是否已经到达设定位置，一般舵机只能旋转 180 度。

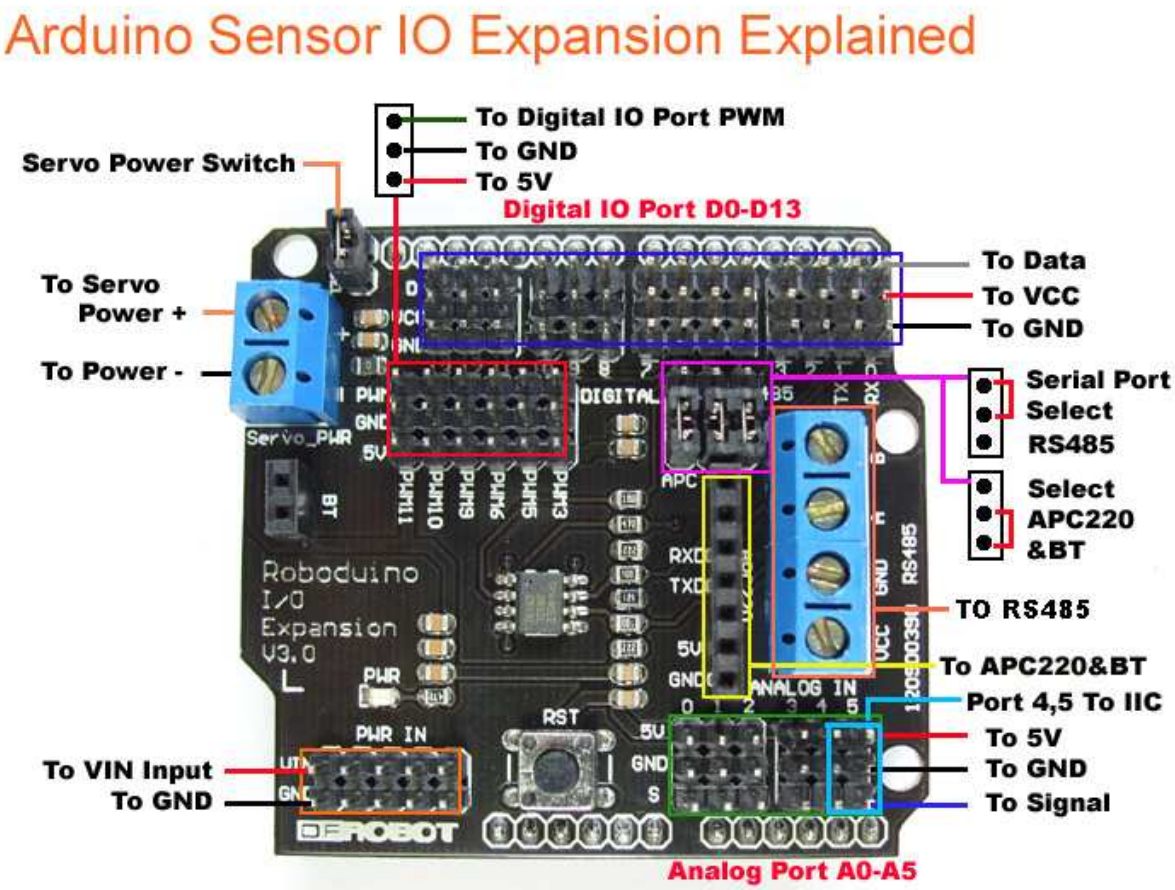
舵机有 3 根线，棕色为地，红色为电源正，橙色为信号线，但不同牌子的舵机，线的颜色可能不同，请大家注意。



舵机的转动位置是靠控制 PWM（脉冲宽度调制）信号的占空比来实现的，标准 PWM（脉冲宽度调制）信号的周期固定为 20ms，占空比 0.5～2.5ms 的正脉冲宽度和舵机的转角-90°～90°相对应。注意，由于舵机牌子不同，其控制器解析出的脉冲宽度也不同，所以对于同一信号，不同牌子的舵机旋转的角度也不同。

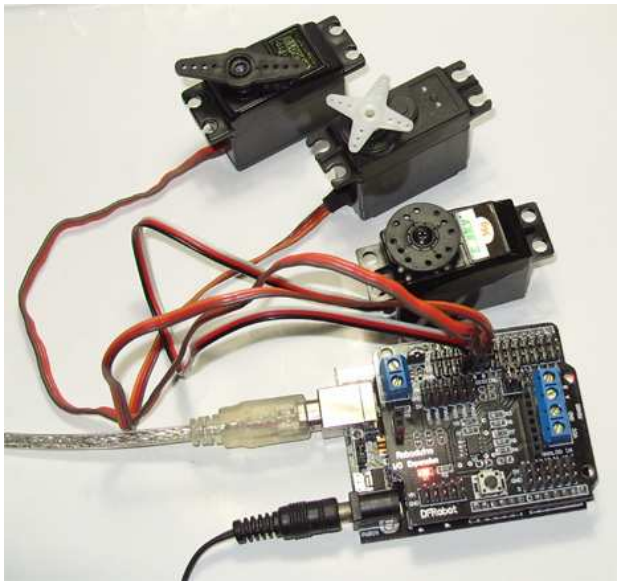


Arduino 控制器无法直接接舵机，使用本公司设计的扩展板来接舵机非常方便，先了解一下扩展板的一些特性。



-Servo Power Switch- If uses internal 5V power apply the jumper, otherwise uses external Servo power.

本实验需要扩展板一个，舵机 3 只。由于只是做演示用，舵机 5v 电源就暂时使用 Arduino 上的，但不可长时间使用，因为多个舵机转动时电流比较大，Arduino 上的电源芯片可能会因过热而保护甚至损坏，电源选择跳线需要接到外部供电，不可使用 USB 供电。



Arduino 官方网站给我们提供了一个很好的舵机调用函数库(舵机函数库下载地址)，可以让 Arduino 任意 IO 口控制舵机，下列代码使用 Arduino 的编译环境中的串口监视器发送数据来控制舵机旋转角度。

实验代码如下：

```
#include <Servo.h>
Servo servo1;    // 定义舵机 1
Servo servo2;    // 定义舵机 2
Servo servo3;    // 定义舵机 3
void setup()
{
  servo1.attach(8);              //定义舵机控制口
  servo1.setMaximumPulse(2200); //定义旋转的时间
  servo2.attach(9);
  servo2.setMaximumPulse(2200);
  servo3.attach(10);
  servo3.setMaximumPulse(2200);
  Serial.begin(19200);           //设置波特率
  Serial.print("Ready");
}
void loop()
{
  static int v = 0;
  if ( Serial.available())
   {
    char ch = Serial.read();//读取串口数据
    switch(ch)
     {
       case '0'...'9': v = v * 10 + ch - '0';//字符换算成 10 进制
                       break;
       case 'a':       servo1.write(v);     //如果数据后带 a，则表示是 servo1 的数据，比如串口发送 85a
                       v = 0;
                       break;
       case 'b':       servo2.write(v);     //如果数据后带 b，则表示是 servo2 的数据，比如串口发送 90b
                       v = 0;
                       break;
       case 'c':       servo3.write(v);     //如果数据后带 c，则表示是 servo3 的数据，比如串口发送 180c
                       v = 0;
                       break;
    }
  }
  Servo::refresh();//刷新
}
```

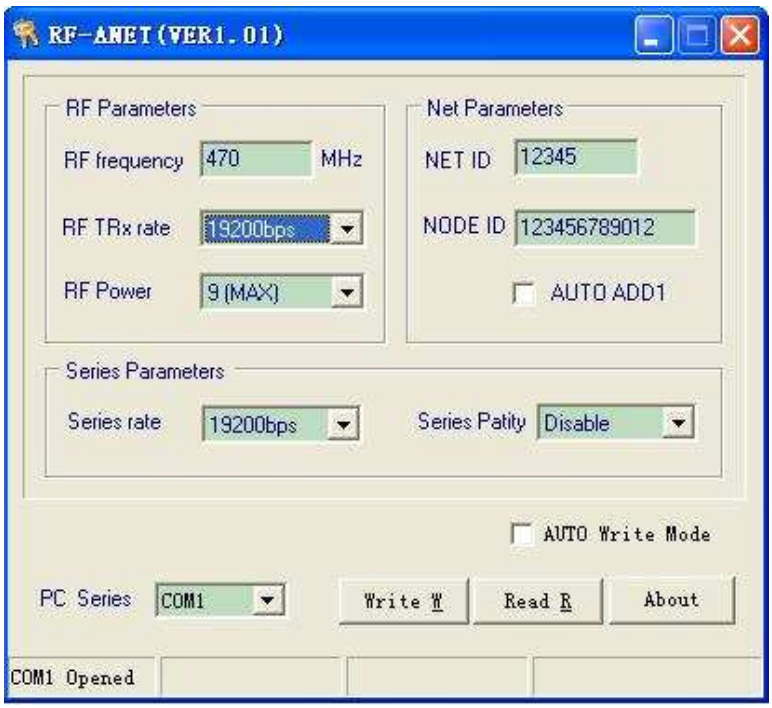
第九课 Arduino 通过无线数传通讯实验

APC220 是通过 Arduino 的串行接口 Tx 和 Rx 引脚进行通信，APC220 不但可以点对点通讯，同时还支持一点对多点通讯。下面先介绍一下 APC220 的特性和参数配置。

（1）。APC220 的指标特性：

1. 工作频率 415MHz to 455MHz 和 460MHz to 478MHz(1KHz 步进) 两种频率注意区分
2. 调制方式 GFSK
3. 频率间隔 200KHz
4. 发射功率 20mw (10 级可调)
5. 接收灵敏度 -117dBm@1200bps
6. 空中传输速率 1200 - 19200bps
7. 接口速率 1200 - 19200bps
8. 接口效验方式 8E1/8N1/801
9. 接口缓冲空间 512bytes
10. 工作湿度 10%～90%（无冷凝）
11. 工作温度 -20℃ - 70℃
12. 电源 3.3 - 5.5V（±50mV 纹波）
13. 发射电流 ≦35mA@10mW
14. 接收电流 ≦30mA
15. 休眠电流 ≦5uA
16. 传输距离 1000 米传输距离（开阔地可视距离）
17. 尺寸 39mm x 19mm x 2.8mm

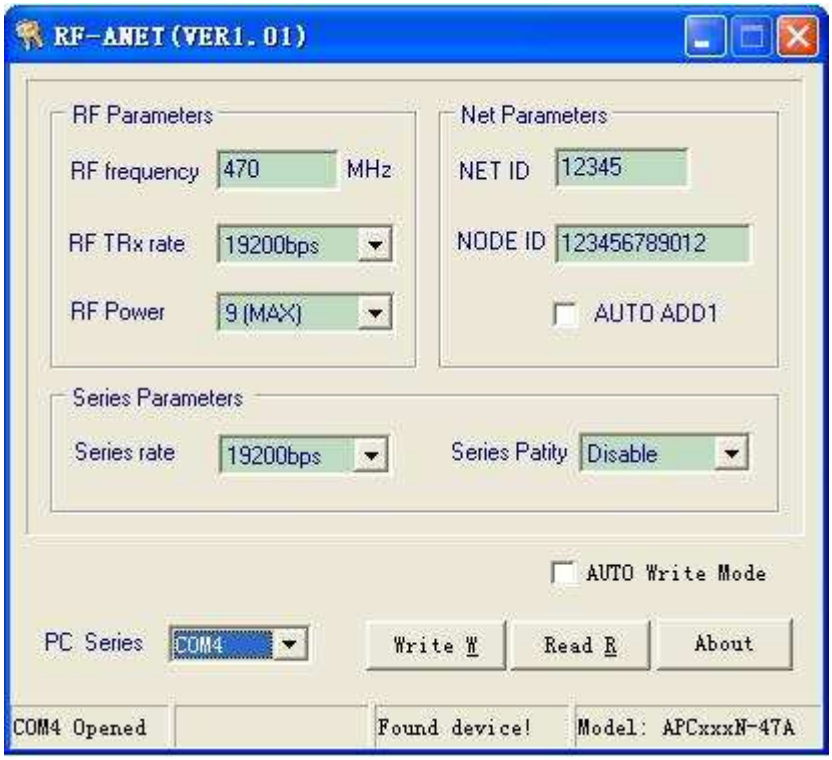
（2）。APC220 的上位机软件：



（3）。USB 转接器的使用：

用 USB 转接器连接电脑使用 APC220 伴侣设置参数，由于 USB 转接器上没有标注引脚功能，所以容易导致我们将 APC 插错位，请看下图的红色标记，避免连接错误。USB 转接器插到电脑上需要安装驱动程序，安装驱动大家应该比较熟练了吧，这里就不多讲了，驱动下载地址。

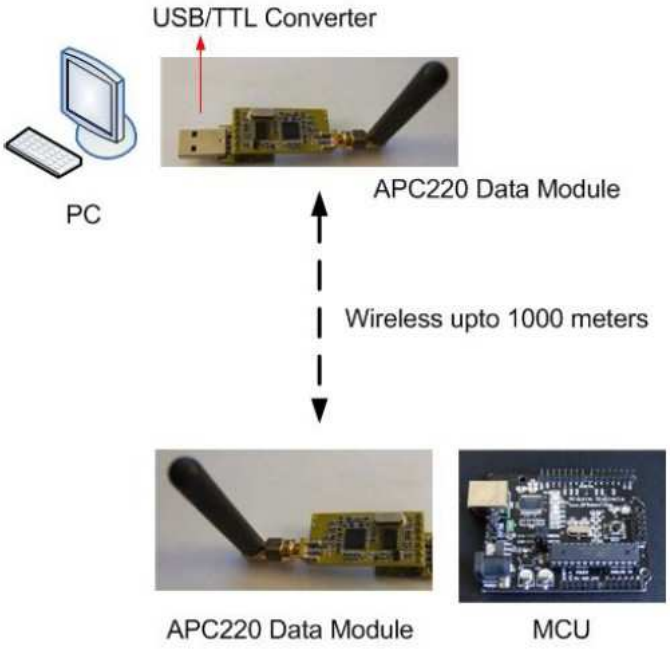
按上图的接法接好后，插到 PC 的 USB 口上，然后打开 APC 伴侣软件。

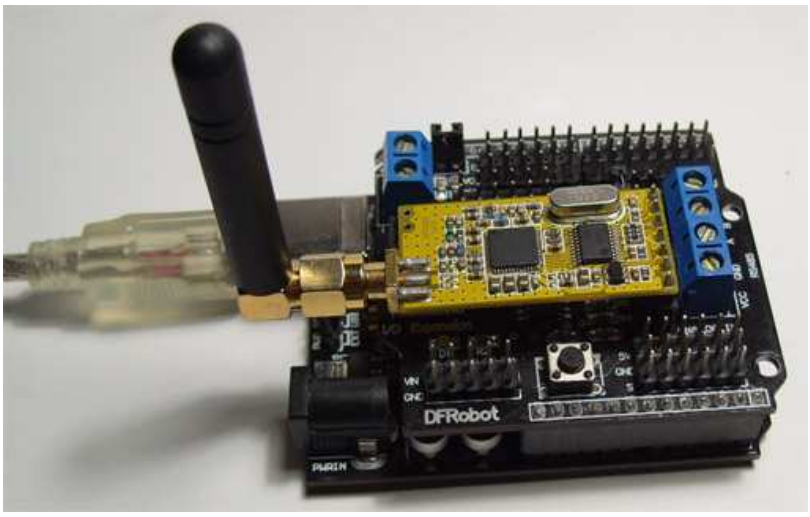


选择 PC Series，软件将会识别到硬件，设置发射频率（RF frequency）范围 431MHz - 478MHz ，空中波特率（RF TRx rate）推荐设置为 19200bps，串口波特率（Series rate）根据要求设置，这里设置为 19200bps，其余参数默认即可，最后点 Write W，完成设置（1 对 APC220 需要配置一样）。

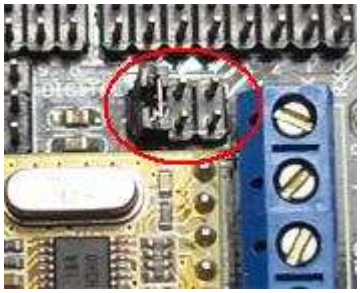
（4）。APC 与扩展板的配合：

系统连接如图：





APC220 无线数传插到 Arduino 扩展板后，注意串口选择插针的位置，插针需要拔掉。如图：



讲到这里，我们就可以用代码来验证了。接收代码如下：

```
int ledPin = 13;
int val;
void setup()
{
  pinMode(ledPin, OUTPUT);              // 数字口 PIN13 设置为输出模式
  Serial.begin(19200);                  //串口波特率为 19200
}
void loop()
{
   val = Serial.read();                 //接收数据
   if (-1 != val)
    {
       if ('A' == val)      {          //如果接收到的数据为 A ，LED 将闪烁
          digitalWrite(ledPin, HIGH);
          delay(500);
          digitalWrite(ledPin, LOW);
          delay(500);
        }
    }
}
```

PC 端，用 Arduino 编译器的串口监视器发送一个字母 A，该字母会被 APC220 模块通过无线网络发送出去。

Arduino 端，另外一个 APC220 模块接收到字母 A，通过 Serial.read()函数从 APC220 模块读取接收到的数据，判断如果是字母 A 的话，则点亮发光二极管。

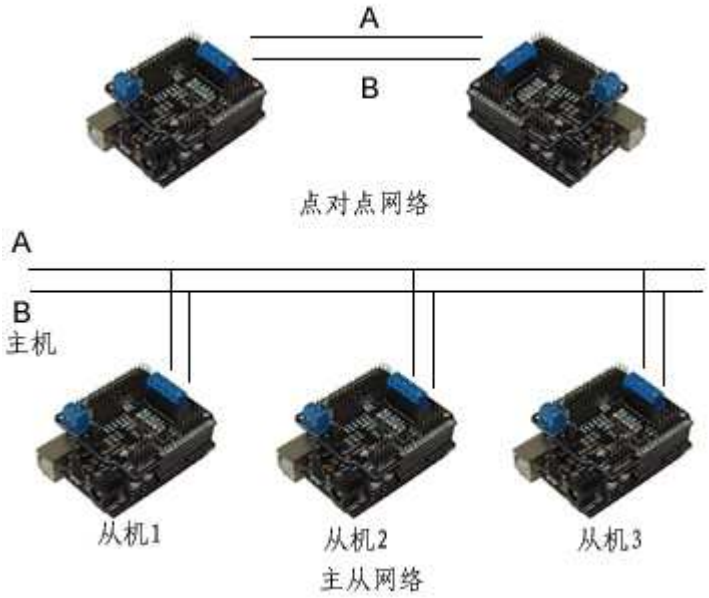
注意，下载代码到 Arduino 时，请拔掉 APC220，因为它会占用串口，导致下载失败。

第十课 Arduino 扩展板的 RS485 通讯实验

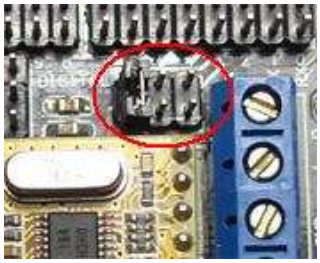
前面我们提到过串行接口按标准被分为 RS-232、RS-422、RS-485。我们的 A 板扩展板就集成了 RS485（MAX485）接口。

RS485 采用差分信号负逻辑，＋2V～＋6V 表示“0”，－ 6V～－ 2V 表示“1”。RS485 有两线制和四线制两种接线，四线制只能实现点对点的通信方式，现很少采用，现在多采用的是两线制接线方式，这种接线方式为总线式拓朴结构在同一总线上最多可以挂接 32 个结点。在 RS485 通信网络中一般采用的是主从通信方式，即一个主机带多个从机。很多情况下，连接 RS-485 通信链路时只是简单地用一对双绞线将各个接口的“A”、“B”端连接起来。其理论的通讯距离是 1200 米，速率高达 20Mbps，并可以用在强噪声的环境中正常工作，在工业通讯领域中被广泛应用。

RS485 通讯的点对点接法和主从机接法如下图：



以上 2 种接法建议都接上地线，以保证减小干扰。在主从网络中，通讯线必须手牵手地并联下去，不可以有星型连接或者分叉。RS485 协议只是把 TTL 的串口转换成差分方式的通讯接口，485 芯片的端口还是接到 Arduino 的 RX（PIN0）和 TX（PIN1）端，两线制的 RS485 是属于半双工网络，则需要有个端口来控制 AB 差分线上何时收何时发，这就会用到 Arduino 上的 PIN2。Arduino 扩展板上需要插上 485 接口选择插针，红圈处的 3 个插针都要插上：



完成上面的操作，我们就可以玩代码了，2 个简单的代码：

主机代码：

```
int EN = 2;
void setup()
{
  pinMode(EN, OUTPUT);
  Serial.begin(19200);
}
void loop()               // 发送数据
{
  digitalWrite(EN, HIGH);//使能发送
  Serial.print('A');
  delay(1000);
}
```

从机代码：

```
int ledPin = 13;
int EN = 2;
int val;
void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(EN, OUTPUT);
  Serial.begin(19200);
}
void loop()             //接收数据
{
  digitalWrite(EN, LOW);//使能接收
  val = Serial.read();
  if (-1 != val) {
    if ('A' == val) {
      digitalWrite(ledPin, HIGH);
      delay(500);
      digitalWrite(ledPin, LOW);
      delay(500);
    }
 }
}
```

代码编译后，分别下载到 2 个 Arduino 中（下载代码时请拔掉上图所示的 485 接口选择插针，因为它要占用串口，导致下载失败），然后连接 2 个扩展板上的 485 接口，A-A，B-B，GND-GND。主机 Arduino 模块会向从机不停的发送字母 A，然后从机接收到字母 A，并使数字 PIN13 上的发光二极管闪烁。