第 14 章　应用实战案例

本章介绍 WAVE6000 仿真软件的使用方法，重点介绍 DS18B20、红外遥控、直流电动机、RS-232 通信、语音录放和无线通信对应的工作原理及控制方法，并设计典型应用电路以及相应的软件程序。

14.1　仿 真 软 件

本节介绍 WAVE6000 软件如何新建文件和项目，以及程序的下载方法。

14.1.1　新建文件和项目

仿真软件使用 WAVE6000（伟福 6000）。打开“文件”菜单，选择“新建文件”命令，可以建立*.ASM 汇编文件或者*.C 为语言文件（若要使用 C 语言文件，在 C:根目录下建立 COMP51 文件夹，将 COMP51.EXE 文件放入，并单击，可生成支持文件）。存放文件的路径一定要为英文的路径，然后打开“文件”菜单，选择“新建项目”命令，加入刚才建立的文件，放入模块文件，不选择包含文件，最后确认项目的名称与文件名保持一致，不需要加扩展名，软件窗口如图 14-1 所示。

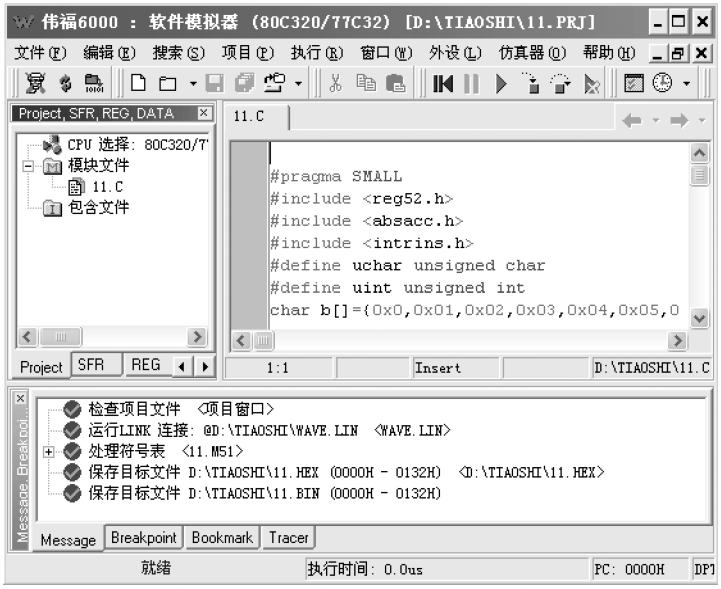

图 14-1　伟福 6000 软件窗口

如图 14-2 所示为观察数据窗口。在模拟调试的时候要用到 CPU 窗口（REG）和数据窗口，其中数据窗口的 DATA 是模拟 CPU 的内部 RAM；CODE 窗口是模拟 CPU 的 64KB 的内外 ROM；XDATA 窗口是模拟 CPU 外部 64KB RAM，在应用 MOVX A，@DPTR 或 MOV @DPTR,A 指令时调试使用；PDATA 窗口是模拟 CPU 外部 256B RAM，在应用 MOVX A,@Ri 或 MOV @Ri,A 指令时调试使用。

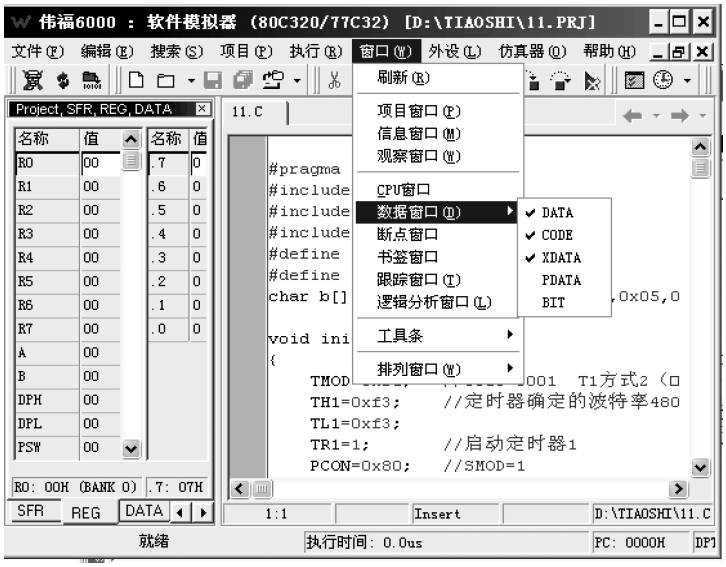

图 14-2　观察数据窗口

14.1.2　下载程序

烧录程序的软件很多，这里使用的烧录软件为 AVR_fighter.EXE，将生成的*.HEX 文件找到，应用该软件通过下载线将程序代码烧录到单片机中，窗口如图 14-3 所示。

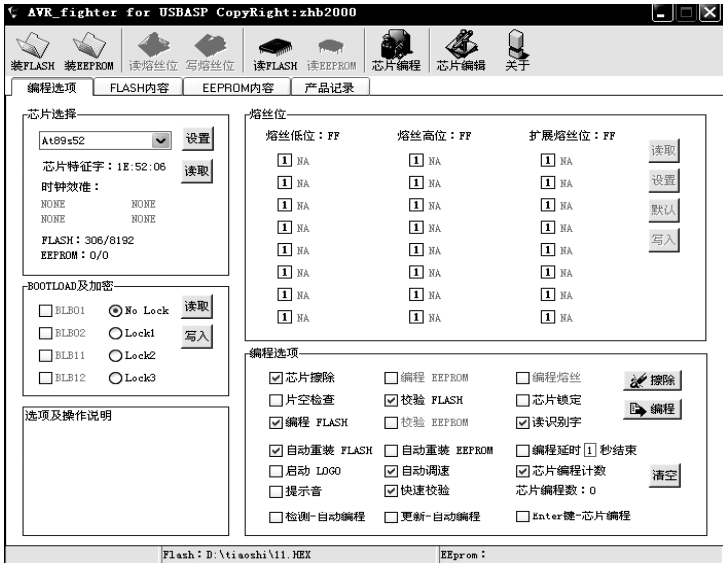

图 14-3　烧录软件窗口

14.2　温度传感器 DS18B20

温度是一种最基本的环境参数，日常生活和工农业生产中经常要检测温度。传统方式是采用热电偶或热电阻，但是由于模拟温度传感器输出为模拟信号，必须经过 A/D 转换环节获得数字信号后才能与单片机等微处理器接口，使得硬件电路结构复杂，制作成本较高。近年来，以 DSl8B20 为代表的新型单总线数字式温度传感器以其突出优点广泛应用于仓储管理、工农业生产制造、气象观测、科学研究以及日常生活中。DSl8B20 集温度测量和 A/D 转换于一体，直接输出数字量，传输距离远，可以很方便地实现多点测量；硬件电路结构简单，与单片机接口几乎不需要外围元件。

14.2.1　单总线概述

1-Wire 总线技术是近年推出的新技术。它将地址线、数据线、控制线合为 1 根信号线，既传输时钟又传输数据，而且数据传输是双向的。允许在这根信号线上挂接多个 1-Wire 总线器件。1-Wire 总线技术具有节省 I/O 资源、结构简单、成本低廉、有广阔的应用空间、便于总线扩展和维护等优点，在分布式测控系统中有着广泛应用。

1-Wire 单总线适用于单个主机系统，能够控制一个或多个从机设备。当只有一个从机位于总线上时，系统可按照单节点系统操作。当多个从机位于总线上时，系统按照多节点系统操作。所有的 1-Wire 总线器件都具有一个共同的特征：在出厂时，每个器件都有一个与其他任何器件互不重复的固定的序列号，通过它自己的序列号可以区分同一总线上的多个器件。

单总线只有一根数据线。设备主机或从机通过一个漏极开路或三态端口，连接至该数据线，这样允许设备在不发送数据时释放数据总线，以便总线被其他设备所使用。单总线端口为漏极开路，单总线通常要求外接一个 4.7～10kΩ的上拉电阻。这样，当总线闲置时，其状态为高电平。

14.2.2　单总线器件——温度传感器 DS18B20

DSl8B20 是世界上第一片支持“单总线”接口的数字式温度传感器，能够直接读取被测物的温度值。

DS18B20 采用 3 脚 TO-92 封装或 8 脚的 SOIC 封装，如图 14-4 所示，可以适应不同的环境需求。下面介绍 DS18B20 各引脚的功能。

（1）GND 为电压地。

（2）DQ 为单数据总线。

（3）V_{DD} 为电源电压。

（4）NC 为空引脚。

DS18B20 单线数字温度传感器，即“一线器件”，其具有独特的优点。

（1）采用单总线的接口方式，与微处理器连接时仅需要一条口线即可实现微处理器与

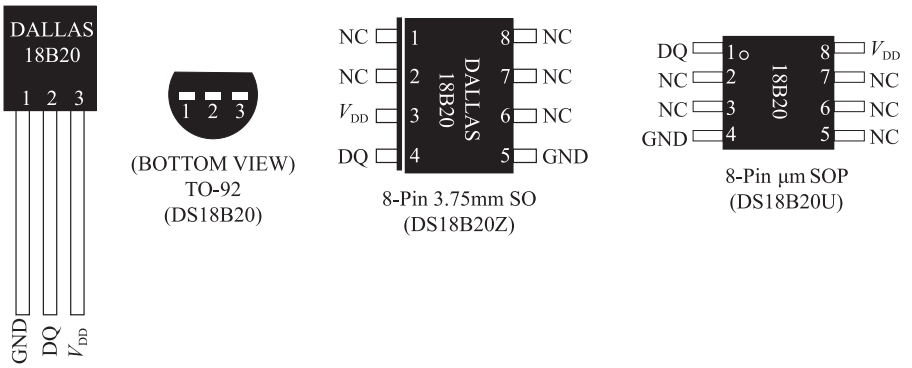

图 14-4　18B20 引脚图

DS18B20 的双向通信。单总线具有经济性好，抗干扰能力强，适合于恶劣环境的现场温度测量，使用方便等优点，使用户可轻松地组建传感器网络，为测量系统的构建引入全新概念。

（2）测量温度范围宽，测量精度高。DS18B20 的测量范围在–55～+125℃、–10～+85℃之内的测量精度可达±0.5℃，稳定度为 1%。

（3）持多点组网功能，在同一根总线上可接多个传感器，实现多点测温，构成多点测温网络，是温度场监控系统的理想选择。此外 DS18B20 是温度—电流传感器，对于提高系统抗干扰能力有很大的帮助。

（4）供电方式灵活，DS18B20 可以通过内部寄生电路从数据线上获取电源。因此，当数据线上的时序满足一定的要求时，可以不接外部电源，从而使系统结构更趋简单，可靠性更高。

（5）测量参数可配置，DS18B20 的测量分辨率可通过程序设定 9～12 位，以上都包括一个符号位，因此对应的温度量化值分别为 0.5℃、0.25℃、0.125℃、0.0625℃，芯片出厂时默认为 12 位的转换精度。

DS18B20 具有微型化、低功耗、抗干扰能力强、更经济、更宽的电压适用范围，易与微处理器接口等优点，可直接将温度转化成串行数字信号供微处理器接收处理。测温电路简单、易于实现，适合于构建测温系统，因此也就被设计者们所青睐。

1．DS18B20的内部存储器

（1）ROM 只读存储器，用于存放 DS18B20 的 ID 编码。开始的 8 位是单线产品系列编码；接下来的 48 位是芯片唯一的序列号，每个 DS18B20 的序列号都不相同；最后 8 位则是前面 56 位的 CRC 码（循环冗余校验码）。DS18B20 共 64 位 ROM，最左边的位为最高位，如表 14-1 所示，数据在出厂时设置，用户不可更改。由于每一个 DS18B20 的 ROM 数据都各不相同，因此微控制器就可以通过单总线对多个 DS18B20 进行寻址，从而实现一根总线上挂接多个 DS18B20。

表 14-1　64 位 ROM

8 位检验 CRC	48 位序列号	8 位工厂代码（10H）

（2）RAM 数据暂存器，用于内部计算和数据存取，数据在掉电后丢失。DS18B20 共 9

个字节 RAM，每个字节 8 位。如表 14-2 所示。第 1、2 字节是温度转换后的数据值信息，第 3、4 字节是高温触发器 TH 和低温触发器 TL 的易失性备份，第 5 字节为配置寄存器，它的内容用于确定温度值的数字转换分辨率，DS18B20 工作时寄存器中的分辨率转换为相应精度的温度数值，以上字节内容每次上电复位时被刷新。

表 14-2　DS18B20 存储器

序号	内容	序号	内容	序号	内容
0	温度的低八位数据	3	低位阈值	6	保留
1	温度的高八位数据	4	配置寄存器	7	保留
2	高温阈值	5	保留（全 1）	8	前八位 CRC 效验值

配置寄存器字节各位的定义如表 14-3 所示。低 5 位一直为 1，TM 是工作模式位，用于设置 DS18B20 在工作模式还是在测试模式，DS18B20 出厂时该位被设置为 0，用户不要去改动；R1 和 R0 用来设置分辨率，决定温度转换的精度位数，如表 14-4 所示。

表 14-3　配置寄存器中的各位定义

第 8 位	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位
TM（0）	R1	R0	1	1	1	1	1

表 14-4　R0、R1 位的设置

R0	R1	温度计分辨率/bit	最大转换时间/ms
0	0	9	93.75
0	1	10	187.5
1	0	11	375
1	1	12	750

出场设置默认 R0、R1 为 11。也就是 12 位分辨率，也就是 1 位代表 0.0625 摄氏度。

2．访问DS18B20的操作顺序

为了保证数据能可靠地传输，任一时刻 1-Wire 总线上只能有一个控制信号或数据。进行数据通信时应符合 1-Wire 总线协议，访问 DS18B20 的操作顺序遵循以下三步：

（1）初始化：单线总线上的所有处理均从初始化序列开始。初始化序列包括总线主机发出复位脉冲，从机送出存在脉冲，使主机知道总线上有从机设备，且准备就绪。

（2）ROM 命令：在总线主机检测到应答脉冲后，就可以发出 ROM 命令，所有 ROM 操作命令均为 8 位。共有 5 种 ROM 命令，分别是读 ROM、搜索 ROM、匹配 ROM、跳过 ROM、报警搜索。对于只有一个温度传感器的单点系统，跳过 ROM 命令特别有用，控制器不必发送 64 位序列号，从而节约了大量时间。对于 1-Wire 总线的多点系统，通常先把每一个单总线器件的 64 位序列号测出，要访问某一个从属节点时，发送匹配 ROM 命令，然后发送 64 位序列号，这时可以对指定的从属节点进行操作。

① 搜索 ROM[F0H]：当系统开始工作时，总线主机必须找出总线上所有从机设备的 ROM 代码，这样主机就能够判断出从机的数目和类型。主机通过重复执行搜索 ROM 循环，

以找出总线上所有的从机设备。如果总线只有一个从机设备，则可以采用读 ROM 命令来替代搜索 ROM 命令。在每次执行完搜索 ROM 循环后，主机必须返回至命令序列的第一步（初始化）。

② 读 ROM[33H]：该命令仅适用于总线上只有一个从机设备。允许主机直接读出从机的 64 位 ROM 代码，无须执行搜索 ROM 过程。如果该命令用于多节点系统，则必然发生数据冲突，因为每个从机设备都会响应该命令。

③ 匹配 ROM[55H]：匹配 ROM 命令跟随 64 位 ROM 代码，从而允许主机访问多节点系统中某个指定的从机设备。仅当从机完全匹配 64 位 ROM 代码时，才会响应主机随后发出的功能命令，其他设备将处于等待复位脉冲状态。

④ 跳过 ROM[CCH]：在单点总线系统中，此命令通过允许总线主机不提供 64 位 ROM 编码而访问存储器操作来节省时间。如果在总线上存在多个从机而且在跳过 ROM 命令之后发出读命令，那么由于多个从机都响应该命令，会在总线上发生数据冲突。

⑤ 报警搜索[ECH]：设置了报警标志的从机响应，该命令的流程与搜索 ROM 命令相同。该命令允许主机设备判断哪些从机设备发生了报警（如最近的测量温度过高或过低）。同搜索 ROM 命令一样，在完成报警搜索循环后，主机必须返回至命令序列的第一步。

（3）功能命令。在主机发出 ROM 命令，以访问某个指定的 DS18B20，接着就可以发出 DS18B20 支持的某个功能命令。这些命令允许主机写入或读出 DS18B20 暂存器、启动温度转换以及判断从机的供电方式。DS18B20 的指令集如表 14-5 所示。

表 14-5　DS18B20 的指令集

指令名称	指令代码	指令功能
温度变换	44H	启动 DS18B20 进行温度转换，转换时间最长为 500ms（典型为 200ms），结果存入 9 字节 RAM 中
读暂存器	0BEH	读内部 RAM 中 9 字节的内容
写暂存器	4EH	发出向内部 RAM 的第 3、4 字节写下限温度数据指令，紧跟该命令之后，传送 2 字节的数据
复制暂存器	48H	将 RAM 中第 3、4 字节的内容复制到 EEPROM 中
重调 EEPROM	0B8H	将 EEPROM 中的内容恢复到 RAM 中的第 3、4 字节
读供电方式	0B4H	读 DS18B20 的供电模式，寄生供电时 DS18B20 发送“0”，外接电源供电 DS18B20 发送“1”
读 ROM	33H	读 DS18B20ROM 中的编码
ROM 匹配(符合 ROM)	55H	发出此命令之后，接着发出 64 位 ROM 编码，访问单总线上与编码相对应的 DS18B20，使之做出响应，为下一步对该 DS18B20 的读写做准备
搜索 ROM	0F0H	用于确定挂接在同一总线上的 DS18B20 的个数和识别 64 位 ROM 地址，为操作各器件做好准备
跳过 ROM	0CCH	忽略 64 位 ROM 地址，直接向 DS18B20 发温度变换命令，适合于单片机工作
警报搜索	0ECH	该指令执行后，只有温度超过设定值上限或下限的片子才做出响应

3．控制器对DS18B20操作流程

1）初始化时序

控制器（单片机）首先发出一个 480～960μs 的低电平脉冲，然后释放总线变为高电平，

并在随后的 480μs 时间内对总线进行检测，如果有低电平出现，说明总线上有器件已做出应答。若无低电平出现，一直都是高电平说明总线上无器件应答。

作为从器件的 DS18B20，在一上电后就一直在检测总线上是否有 480～960μs 的低电平出现。如果有，在总线转为高电平并等待 15～60μs 后将总线电平拉低 60～240μs 做出响应存在脉冲，告诉主机本器件已做好准备。若没有检测到就一直检测等待。DS18B20 的初始化时序如图 14-5 所示。

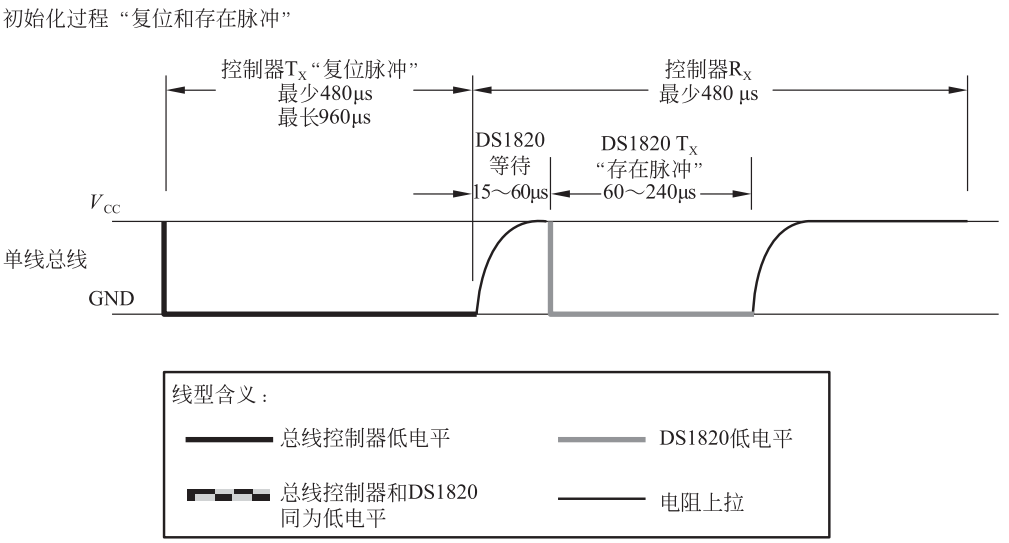

图 14-5　初始化时序图

程序代码如下：

```
/*******************************************************************************
* 函数名         : Ds18b20Init
* 函数功能       : 初始化
* 输入           : 无
* 输出           : 初始化成功返回 1，失败返回 0
*******************************************************************************/
unsigned char Ds18b20Init()
{
    unsigned int i;
    DSIO=0;              //将总线拉低 480~960μs
    i=70;
    while(i--);          //延时 642μs
    DSIO=1;//然后拉高总线，若 DS18B20 做出反应，会在 15~60μs 后将总线拉低
    i=0;
    while(DSIO)          //等待 DS18B20 拉低总线
    {
        i++;
        if(i>50000)      //等待>50ms
            return 0;    //初始化失败
    }
    return 1;            //初始化成功
}
```

2）写时序

主机发出各种操作命令都是向 DS18B20 写 0 和写 1 组成的命令字节，接收数据时也是从 DS18B20 读取 0 或 1 的过程。因此首先要搞清主机是如何进行写 0、写 1、读 0 和读 1 的。

写操作时序如图 14-6 所示。写周期最少为 60μs，最长不超过 120μs。写周期一开始作为主机先把总线拉低 1μs 表示写周期开始。随后若主机想写 0，则将总线置为低电平；若主机想写 1，则将总线置为高电平。持续时间最少 60μs 直至写周期结束，然后释放总线为高电平至少 1μs 给总线恢复。而 DS18B20 则在检测到总线被拉底后等待 15μs，然后从 15~45μs 开始对总线采样。在采样期内总线为高电平，则为 1；若采样期内总线为低电平，则为 0。

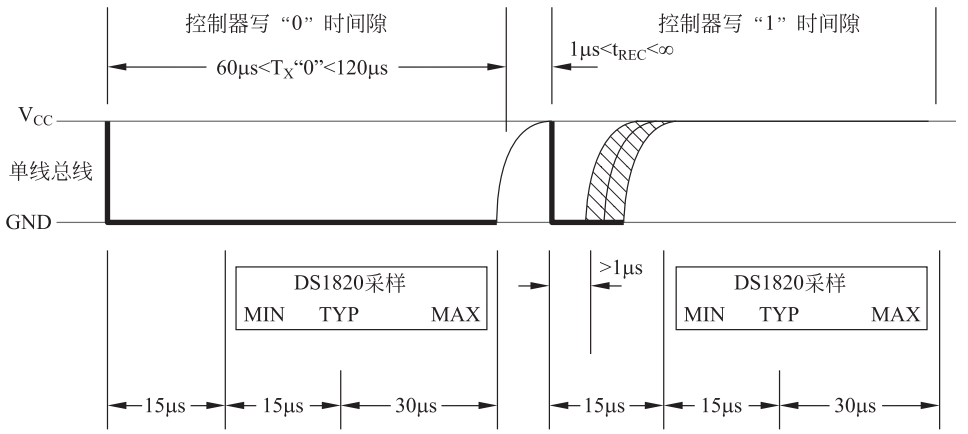

图 14-6　写操作时序图

程序代码如下：

```
/*******************************************************************************
* 函数名         : Ds18b20WriteByte
* 函数功能       : 向 18B20 写入一个字节
* 输入           : com
* 输出           : 无
*******************************************************************************/
void Ds18b20WriteByte(unsigned char dat)
{
    unsigned int i,j;
    for(j=0;j<8;j++)
    {
        DSIO=0;          //每写入一位数据之前先把总线拉低 1μs
        i++;
        DSIO=dat&0x01; //然后写入一个数据，从最低位开始
        i=6;
        while(i--); //延时 68μs，持续时间最少 60μs
        DSIO=1; //然后释放总线，至少 1μs 给总线恢复时间，才能接着写入第二个数值
        dat>>=1;
    }
}
```

3）读时序

对于读数据操作，时序也分为读 0 时序和读 1 时序 2 个过程。

读操作时序如图 14-7 所示。从主机把单总线拉低 1μs 之后就得释放单总线为高电平，以让 DS18B20 把数据传输到单总线上。作为从机，DS18B20 在检测到总线被拉低 1μs 后，便开始送出数据。若是要送出 0，就把总线拉为低电平，直到读周期结束。若要送出 1，则释放总线为高电平。主机在一开始拉低总线 1μs 后释放总线，然后在包括前面的拉低总线电平 1μs 在内的 15μs 时间内完成对总线的采样检测。采样期内总线为低电平，则确认为 0。采样期内总线为高电平，则确认为 1。完成一个读时序过程，至少需要 60μs。

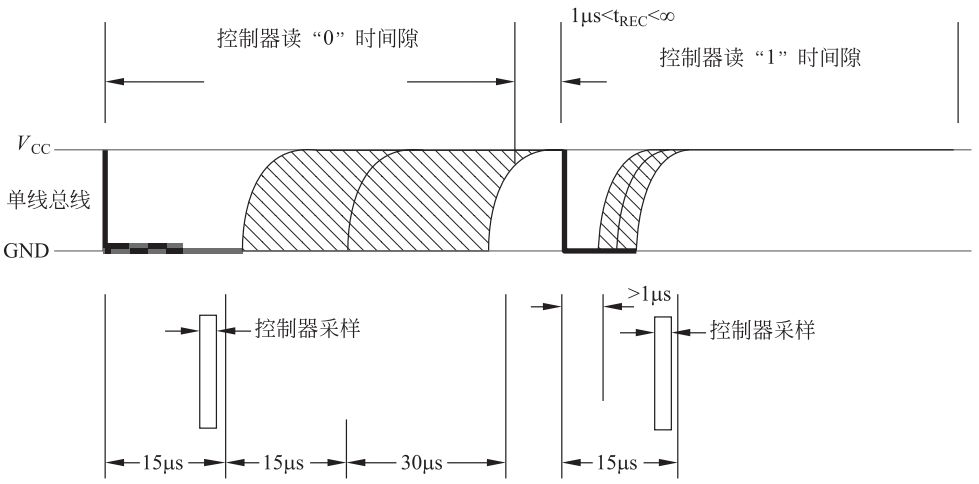

图 14-7　读操作时序图

程序代码如下：

```
/*******************************************************************************
* 函数名         : Ds18b20ReadByte
* 函数功能       : 读取一个字节
* 输入           : com
* 输出           : 无
*******************************************************************************/
unsigned char Ds18b20ReadByte()
{
    unsigned char byte,bi;
    unsigned int i,j;
    for(j=8;j>0;j--)
    {
        DSIO=0;     //先将总线拉低 1μs
        i++;
        DSIO=1;     //然后释放总线
        i++;
        i++;        //延时 6μs 等待数据稳定
        bi=DSIO;    //读取数据，从最低位开始读取
        byte=(byte>>1)|(bi<<7); /*将 byte 右移一位，然后与左移 7 位后的 bi 做与操
                                  作，注意移动之后移掉的位补 0 */
        i=4;        //读取完之后等待 48μs，再接着读取下一个数
        while(i--);
    }
    return byte;
}
```

4．DS18B20设计中应注意的几个问题

实际应用中应注意以下几方面的问题，较小的硬件开销需要相对复杂的软件进行补偿，由于 DS18B20 与微处理器间采用串行数据传送，因此，在对 DS18B20 进行读写编程时，必须严格的保证读写时序，否则将无法读取测温结果。

在 DS18B20 的有关资料中均未提及 1-Wire 上所挂 DS18B20 的数量问题，容易使人误认为可以挂任意多个 DS18B20，在实际应用中并非如此。当 1-Wire 上所挂 DS18B20 超过 8 个时，就需要考虑微处理器的总线驱动问题，这一点在进行多点测温系统设计时要加以注意。

连接 DS18B20 的总线电缆是有长度限制的。试验中，当采用普通信号电缆传输长度超过 50m 时，读取的测温数据将发生错误。当将总线电缆改为双绞线带屏蔽电缆时，正常通信距离可达 150m。这主要是由于总线分布电容使信号波形产生畸变造成的。因此，在用 DS18B20 进行长距离测温系统设计时要充分考虑总线分布电容和阻抗匹配问题。实际应用中，测温电缆线建议采用屏蔽 4 芯双绞线，其中一对线接地线与信号线，另一对接 V_{CC} 和地线，屏蔽层在源端单点接地。

14.2.3　应用电路设计

利用 DS18B20 进行温度检测，并在 LCD1602 上显示此温度值，硬件电路图如图 14-8 所示。

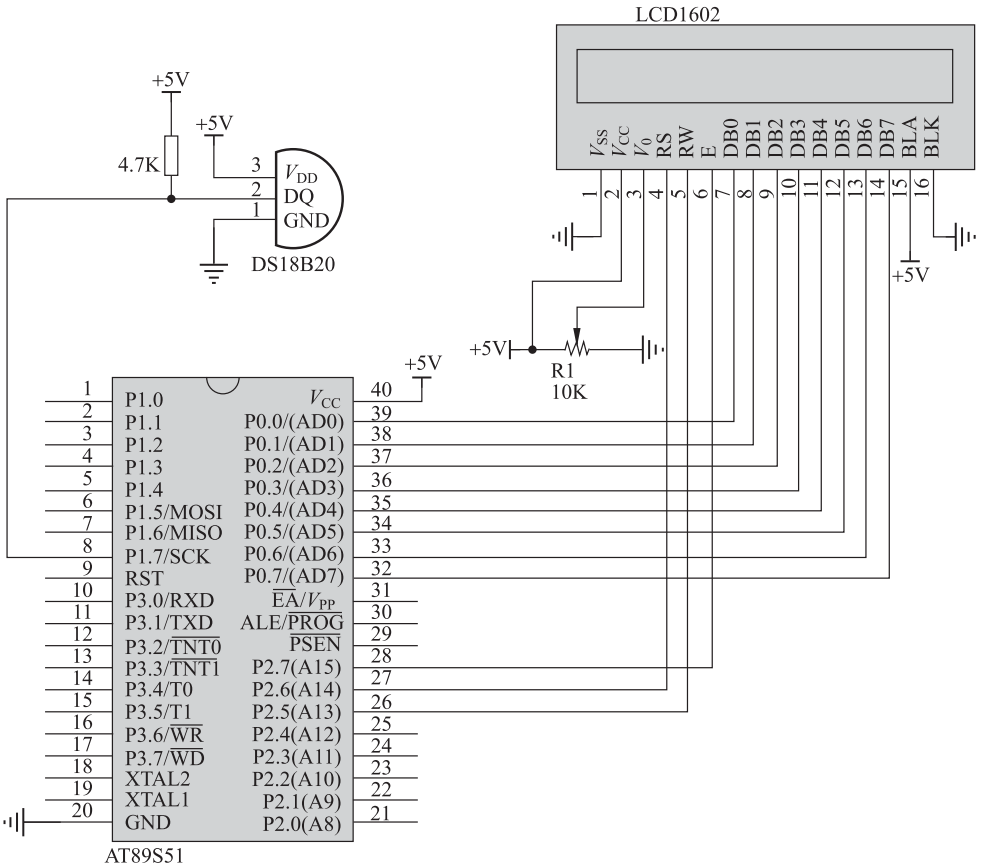

图 14-8　DS18B20 连接图

程序代码如下：

```
/*************************************************************************
* 实验名      : 1602 显示温度                  :
*************************************************************************/
#include<reg51.h>
#include"lcd.h"
#include"temp.h"
void LcdDisplay(int);
/*************************************************************************
* 函数名         : main
* 函数功能       : 主函数
*************************************************************************/
void main()
{
    while(1)
    {
            Ds18b20ChangTemp();
            LcdDisplay(Ds18b20ReadTemp());
            Delay1ms(1000);  //1s 刷一次
    }
}
/*************************************************************************
* 函数名         : LcdDisplay()
* 函数功能       : LCD 显示读取到的温度
* 输入           : v
* 输出           : 无
*************************************************************************/
void LcdDisplay(int temp)    //LCD 显示
{

  unsigned char datas[] = {0, 0, 0, 0, 0}; //定义数组
    float tp;
    LcdInit();               //初始化 LCD
    if(temp< 0)              //当温度值为负数
    {
    LcdWriteCom(0x80);       //写地址 80 表示初始地址
     LcdWriteData('-');      //显示负
    //因为读取的温度是实际温度的补码，所以当温度为负数的时候，要将它转换为实际温度
            temp=temp-1;//将读取到的温度值减 1，再取反
            temp=~temp;
            tp=temp;
            temp=tp*0.0625*100+0.5;
    //留两个小数点就*100，+0.5 是四舍五入，因为 C 语言浮点数转换为整型的时候把小数点
    //后面的数自动去掉，不管是否大于 0.5，而+0.5 之后大于 0.5 的就是进 1 了，小于 0.5
    //的就算加上 0.5，还是在小数点后面
}
else
 {
    LcdWriteCom(0x80);       //写地址 80 表示初始地址
     LcdWriteData('+');          //显示正
    tp=temp;    //因为数据处理有小数点，所以将温度赋给一个浮点型变量
                //如果温度是正的，那么正数的补码就是它本身
    temp=tp*0.0625*100+0.5;
        //留 2 个小数点就*100，+0.5 是四舍五入，因为 C 语言浮点数转换为整型的时候把小
        //数点后面的数自动去掉，不管是否大于 0.5，而+0.5 之后大于 0.5 的就是进 1 了，小
        //于 0.5 的就算加上 0.5，还是在小数点后面
}
```

```
    datas[0] = temp / 10000;
    datas[1] = temp % 10000 / 1000;
    datas[2] = temp % 1000 / 100;
    datas[3] = temp % 100 / 10;
    datas[4] = temp % 10;

    LcdWriteCom(0x82);              //写地址 80 表示初始地址
    LcdWriteData('0'+datas[0]);     //百位

    LcdWriteCom(0x83);              //写地址 80 表示初始地址
    LcdWriteData('0'+datas[1]);     //十位

    LcdWriteCom(0x84);              //写地址 80 表示初始地址
    LcdWriteData('0'+datas[2]);     //个位

    LcdWriteCom(0x85);              //写地址 80 表示初始地址
    LcdWriteData('.');              //显示 ‘ .’

    LcdWriteCom(0x86);              //写地址 80 表示初始地址
    LcdWriteData('0'+datas[3]);     //显示小数点后第 1 位

    LcdWriteCom(0x87);              //写地址 80 表示初始地址
    LcdWriteData('0'+datas[4]);     //显示小数点后第 2 位

    LcdWriteCom(0x88);              //写地址 80 表示初始地址
    LcdWriteData('C');              //显示 C 即℃
}
/************************此部分是对 18B20 的操作*********/
 temp.c 文件
#include"temp.h"
/*****************************************************************************
* 函数名        : Delay1ms
* 函数功能      : 延时函数
* 输入          : 无
* 输出          : 无
*****************************************************************************/
void Delay1ms(unsigned int y)
{
    unsigned int x;
    for(y;y>0;y--)
        for(x=110;x>0;x--);
}
/*****************************************************************************
* 函数名        : Ds18b20Init
* 函数功能      : 初始化
* 输入          : 无
* 输出          : 无
*****************************************************************************/
void Ds18b20Init()
{
    unsigned int i;
    ds=0;
    i=100;
    while(i>0)i--;
    ds=1;
    i++;
```

```
    i++;
    while(ds);
}
/*****************************************************************************
* 函数名         : Ds18b20WriteCom
* 函数功能       : 向 18B20 写入一个字节命令
* 输入           : com
* 输出           : 无
*****************************************************************************/
void Ds18b20WriteCom(unsigned char com)
{
    unsigned int i,j;
    for(j=0;j<8;j++)
    {
        ds=0;
        i++;
        i++;
        ds=com&0x01;
        i=6;
        while(i>0)i--;
        ds=1;
        i++;
        i++;
        com>>=1;
    }
}
/*****************************************************************************
* 函数名         : Ds18b20ReadByte
* 函数功能       : 读取一个字节
* 输入           : com
* 输出           : 无
*****************************************************************************/
unsigned char Ds18b20ReadByte()
{
    unsigned char byte,bi;
    unsigned int i,j;
    for(j=8;j>0;j--)
    {
        ds=0;
        i++;
        ds=1;
        i++;
        i++;
        bi=ds;
        byte=(byte>>1)|(bi<<7);  //注意，移动之后移掉那位补 0
        i=5;while(i)i--;
    }
    return byte;
}
/*****************************************************************************
* 函数名         : Ds18b20ChangTemp
* 函数功能       : 让 18b20 开始转换温度
* 输入           : com
* 输出           : 无
*****************************************************************************/
void  Ds18b20ChangTemp()
{
    Ds18b20Init();
```

```
    Delay1ms(1);
    Ds18b20WriteCom(0xcc);      //跳过 ROM 操作命令
    Ds18b20WriteCom(0x44);      //温度转换命令
    Delay1ms(950);
}
/*****************************************************************************
* 函数名         : Ds18b20ReadTempCom
* 函数功能       : 发送读取温度命令
* 输入           : com
* 输出           : 无
*****************************************************************************/
void  Ds18b20ReadTempCom()
{
    Ds18b20Init();
    Delay1ms(1);
    Ds18b20WriteCom(0xcc);      //跳过 ROM 操作命令
    Ds18b20WriteCom(0xbe);      //发送读取温度命令
}
/*****************************************************************************
* 函数名         : Ds18b20ReadTemp
* 函数功能       : 读取温度
* 输入           : com
* 输出           : 无
*****************************************************************************/
int Ds18b20ReadTemp()
{
    int temp=0;
    unsigned char tmh,tml;
    Ds18b20ChangTemp();             //先写入转换命令
    Ds18b20ReadTempCom();           //等待转换完后，发送读取温度命令
    tml=Ds18b20ReadByte();          //读取温度值共 16 位，先读低字节
    tmh=Ds18b20ReadByte();          //再读高字节
    temp=tmh;
    temp<<=8;
    temp|=tml;
//  temp1=temp;
//  t=temp1*0.0625; //18B20 的分辨率是 0.0625，将读取到的值乘以 0.0625，就是实际
                    //温度值
//  temp1=t*100 ;//+(temp1>0?0.5:-0.5);//留两位有效值，并且四舍五入
    return temp;
}
```

14.3 红 外 遥 控

红外线遥控（简称红外遥控）是目前使用最广泛的一种通信和遥控手段。红外遥控装置具有体积小、功耗低、功能强、成本低等特点，继彩电、录像机之后，在录音机、音响设备、空调机以及玩具等其他小型电器装置上也纷纷采用红外遥控。工业设备中，在高压、辐射、有毒气体、粉尘等环境下，采用红外遥控不仅完全可靠，而且能有效地隔离电气干扰。

1．红外遥控系统

通用红外遥控系统由发射和接收两部分组成。应用编/解码专用集成电路芯片进行控制

操作，如图 14-9 所示。发射部分包括键盘矩阵、编码调制、LED 红外发送器；接收部分包括光、电转换放大器、解调、解码电路。

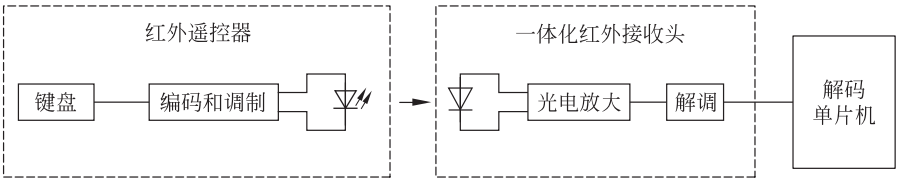

图 14-9　红外线遥控系统框图

2．遥控发射器及其编码

遥控发射器专用芯片很多，根据编码格式可以分成两类。这里介绍运用比较广泛，解码比较容易的一类，现以日本 NEC 的 UPD6121G 组成发射电路为例说明编码原理（一般家庭用的 DVD、VCD、音响都使用这种编码方式）。当发射器按键按下后，即有遥控码发出，所按的键不同，遥控编码也不同。这种遥控码采用脉宽调制的串行码，以脉宽为 0.56ms、间隔 0.565ms、周期为 1.125ms 的组合表示二制的 0；以脉宽为 0.56ms、间隔 1.685ms、周期为 2.25ms 的组合表示二进制的 1，其波形如图 14-10 所示（注意，所有接收端的波形与发射端相反）。

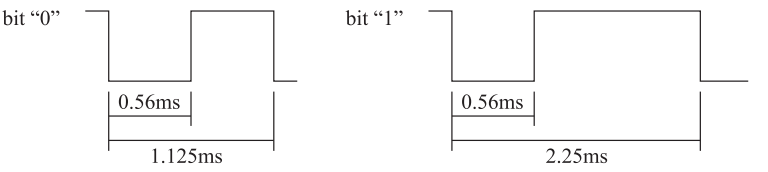

图 14-10　遥控码的 0 和 1

上述 0 和 1 组成的 32 位二进制码经 38kHz 的载频进行二次调制以提高发射效率，达到降低电源功耗的目的；然后通过红外发射二极管产生红外线向空间发射。

NEC 编码的一帧（通常按一下遥控器按钮所发送的数据）由引导码、用户码、数据码组成，如图 14-11 所示。数据反码是数据码反相后的编码（如 11110000 的反码为 00001111），主要用于验证接收的信息的准确性。注意，第二段的用户码也可以在遥控应用电路中被设置成第一段用户码的反码。

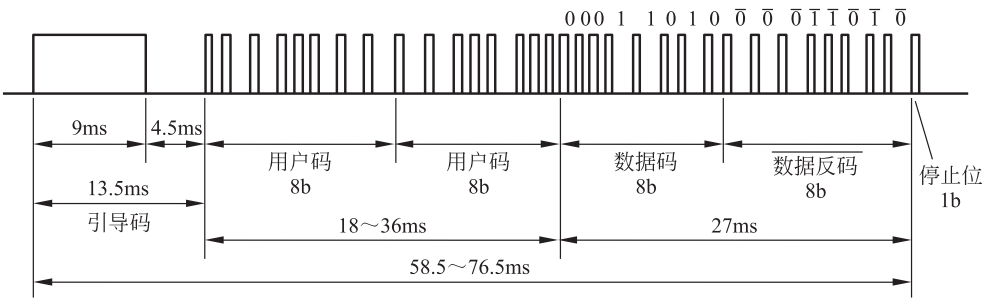

图 14-11　遥控发射信号编码波形图

UPD6121G 产生的遥控编码是连续的 32 位二进制码组，其波形图如图 14-11 所示。其中前 16 位为用户识别码，能区别不同的电器设备，防止不同机种遥控码互相干扰。该芯片的用户识别码固定为十六进制 01H；后 16 位为 8 位操作码（功能码）及其反码。UPD6121G 最多有 128 种不同组合的编码。

NEC 协议规定低位首先发送。当一个键按下超过 36ms 时，振荡器使芯片激活，将发射一组 108ms 的编码脉冲，这 108ms 发射代码由一个引导码（9ms）、一个结束码（4.5ms）、低 8 位地址码（9～18ms）、高 8 位地址码（9～18ms）、8 位数据码（9～18ms）和这 8 位数据的反码（9～18ms）组成。如果键按下超过 108ms 仍未松开，接下来发射的代码则是以 110ms 为周期的重复码，并且不带任何数据，如图 14-12 所示。

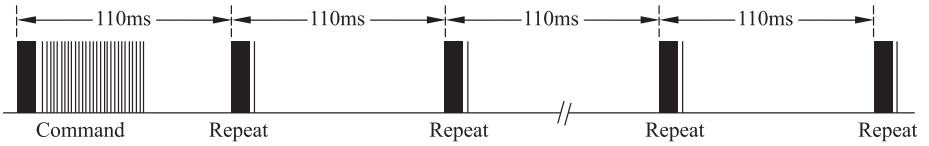

图 14-12　按键一直未松时发射信号编码波形图

即发了一次命令码之后，不会再发送命令码，而是每隔 110ms 时间发送一段重复码，直到按键被松开。重复码的格式是由 9ms 的 AGC 高电平和 2.25ms 的低电平以及一个 560μs 的高电平组成，重复码格式如图 14-13 所示。

3．遥控信号接收

接收电路可以使用一种集红外线接收和放大于一体的一体化红外线接收器，不需要任何外接元件，就能完成从红外线接收到输出与 TTL 电平信号兼容的所有工作。这种接收器的体积和普通的塑封三极管大小一样，它适合各种红外线遥控和红外线数据传输。接收器对外只有 3 个引脚（Out、GND、V_{CC}），与单片机接口非常方便，如图 14-14 所示。

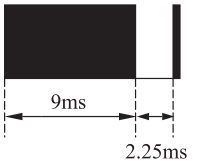

图 14-13　重复码格式

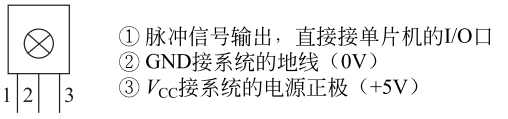

图 14-14　红外线接收器的引脚

4．应用电路设计

IR1 的脉冲输出连接到单片机中断引脚 P3.2，通过触发外部中断，进而使单片机对接收头进行解码，如图 14-15 所示。红外接收利用外部中断触发，整个解码部分都在中断子函数里，故 main()中只显示部分代码，而 1602 的显示函数使用的都是一样的头文件，如果不清楚，可以返回 1602 查看详细代码。

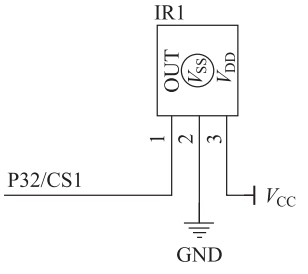

图 14-15　硬件连接图

分析：通过外部中断来读取红外发送的数据，接收到的信号脉冲与发送信号脉冲正好反向。

产生下降沿，进入外部中断 0 的中断函数，延时一下之后检测 I/O 口是否还是低电平。如果是，就等待 9ms 的低电平过去，再去等待 4.5ms 的高电平过去，接着开始接收传送的 4 组数据，如图 14-16 所示。

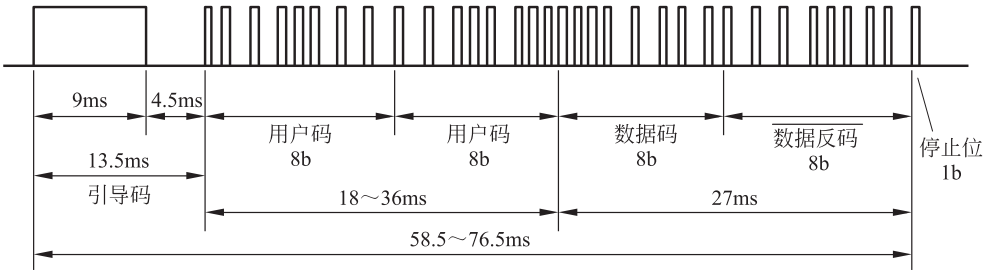

图 14-16　接收的脉冲

① 先等待 560μs 的低电平过去。

② 检测高电平的持续时间，如果超过 1.12ms 就是高电平（高电平的持续时间为 1.69ms，低电平的持续时间为 565μs）。

③ 检测接收到的数据和数据的反码进行比较，是否与等到的数据是一样的。

程序代码如下：

```
/*******************************************************************************
* 实验效果       : LCD1602 显示出读取到的红外线的值
*******************************************************************************/
 #include<reg51.h>
#include"lcd.h"
sbit IRIN=P3^2;
unsigned char code CDIS1[13]={" Red Control "};
unsigned char code CDIS2[13]={" IR-CODE:--H "};
unsigned char IrValue[6];
unsigned char Time;
void IrInit();
void DelayMs(unsigned int );
/*******************************************************************************
* 函数名         : main
* 函数功能       : 主函数
*******************************************************************************/
void main()
{
    unsigned char i;
    IrInit();
    LcdInit();
    LcdWriteCom(0x80);
    for(i=0;i<13;i++)
    {
        LcdWriteData(CDIS1[i]);
    }
    LcdWriteCom(0x80+0x40);
    for(i=0;i<13;i++)
    {
        LcdWriteData(CDIS2[i]);
    }
```

```
    while(1)
    {
        IrValue[5]=IrValue[2]>>4;           //高位
        IrValue[6]=IrValue[2]&0x0f;         //低位
        if(IrValue[5]>9)
        {
            LcdWriteCom(0xc0+0x09);         //设置显示位置
            LcdWriteData(0x37+IrValue[5]); //将数值转换为该显示的 ASCII 码
        }
        else
        {
            LcdWriteCom(0xc0+0x09);
            LcdWriteData(IrValue[5]+0x30); //将数值转换为该显示的 ASCII 码
        }
        if(IrValue[6]>9)
        {
            LcdWriteCom(0xc0+0x0a);
            LcdWriteData(IrValue[6]+0x37); //将数值转换为该显示的 ASCII 码
        }
        else
        {
            LcdWriteCom(0xc0+0x0a);
            LcdWriteData(IrValue[6]+0x30); //将数值转换为该显示的 ASCII 码
        }
    }
}
/*******************************************************************************
* 函数名         : DelayMs()
* 函数功能       : 延时
* 输入           : x
* 输出           : 无
*******************************************************************************/
void DelayMs(unsigned int x)                //0.14ms 误差 0μs
{
   unsigned char i;
   while(x--)
   {
    for (i = 0; i<13; i++)
    {}
   }
}
/*******************************************************************************
* 函数名         : IrInit()
* 函数功能       : 初始化红外线接收
*******************************************************************************/
void IrInit()
{
    IT0=1;  //下降沿触发
    EX0=1;  //打开中断 0 允许
    EA=1;   //打开总中断
    IRIN=1; //初始化端口
}
/*******************************************************************************
* 函数名         : ReadIr()
* 函数功能       : 读取红外数值的中断函数
*******************************************************************************/
void ReadIr() interrupt 0
{
    unsigned char j,k;
    unsigned int err;
    EX0=0;
    Time=0;
```

```
    DelayMs(70);
    if(IRIN==0)              //确认是否真的接收到正确的信号
    {
        err=1000;            //1000*10μs=10μs，超过说明接收了错误的信号
        while((IRIN==0)&&(err>0))   //等待前面 9ms 的低电平过去
        {
//当两个条件都为真循环，如果有一个条件为假的时候，跳出循环
            DelayMs(1);
            err--;
        }
        if(IRIN==1)                 //如果正确，等到 9ms 低电平
        {
                err=500;
                while((IRIN==1)&&(err>0))   //等待 4.5ms 的起始高电平过去
                {
                    DelayMs(1);
                    err--;
                }
                     for(k=0;k<4;k++)        //共有 4 组数据
                  {
                for(j=0;j<8;j++)             //接收 1 数据
                {
                    err=60;
                    while((IRIN==0)&&(err>0)) //等待信号前面的 560μs 低电平过去
                    while (!IRIN)
                    {
                        DelayMs(1);
                        err--;
                    }
                    err=500;
                    while((IRIN==1)&&(err>0)) //计算高电平的时间长度
                    {
                        DelayMs(1);  //0.14ms
                        Time++;
                        err--;
                        if(Time>30)
                        {
                            EX0=1;
                            return;
                        }
                    }
                    IrValue[k]>>=1;  //k 表示第几组数据
                    if(Time>=8)      //如果高电平出现（大于 565μs），那么是 1
                    {                //计算的时钟大于 1.12ms
                        IrValue[k]|=0x80;
                    }
                    Time=0;          //用完时间要重新赋值
                }
            }
        }
        if(IrValue[2]!=~IrValue[3])
        {
            EX0=1;
            return;
        }
    }
    EX0=1;
}
```

14.4 直流电动机控制

本节介绍直流电动机的工作原理和调速方法，重点介绍 PWM 调速原理及编程。

14.4.1 直流电动机工作原理及调速方法

直流电动机通过换向器将直流转换成电枢绕组中的交流，从而使电枢产生一个恒定方向的电磁转矩，电机内部结构图如图 14-17 所示

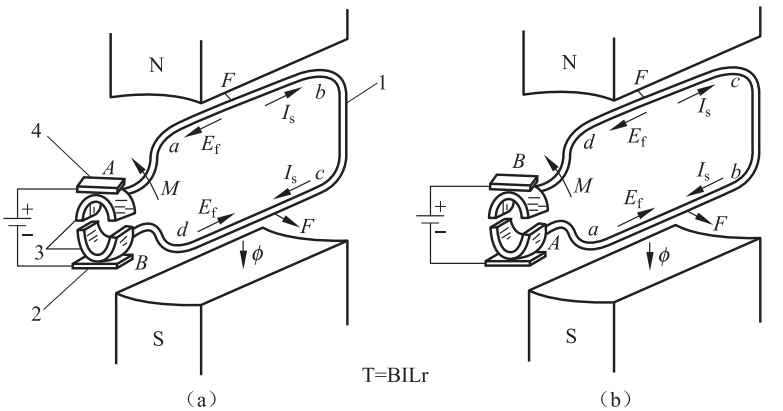

图 14-17 直流电机内部结构图

直流电机的转速公式：

$$n = \frac{U - IR}{K_e \Phi} \tag{14-1}$$

由上式可知，改变转速可有 3 种方法。

（1）改变电枢电压。

（2）改变励磁电流，即改变磁通。

（3）电枢回路串入调节电阻。

14.4.2 PWM 调速原理

PWM 控制是利用脉宽调制器对大功率晶体管开关放大器的开关时间进行控制，将直流电压转换成某一频率的矩形波电压，加到直流电机的电枢两端，通过对矩形波脉冲宽度的控制，改变电枢两端的平均电压以达到调节电机转速的目的。

1．PWM波形的产生

如图 14-18 所示，设 T=X×Ts，T1=Y×Ts，T2=Z×Ts。其中 X 为 T 周期参数，放在 20H 单元。Y 为 T1 延时参数，放在 21H 单元，Z=X–Y。Ts 为延时的时间基数，由定时器

T0 确定。定时初值放在 22H、23H 单元中。

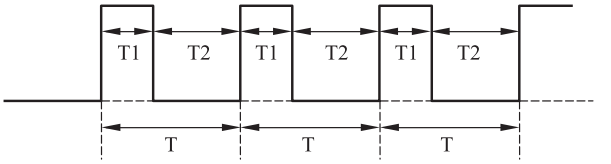

图 14-18　PWM 波形图

2．PWM波形的产生程序代码

```
        ORG 0000H
          LJMP MAIN
          ORG 000BH
          LJMP TT0          ;定时器 0 中断程序
          ORG 1000H
MAIN:   SETB P1.0           ;脉冲的高电平
          MOV R0,21H; R0=Y
        MOV TMOD, #01H
          MOV TL0,22H       ;时间基数 Ts 的定时参数
          MOV TH0,23H
          SETB TR0          ;定时中断设置
          SETB ET0
        SETB EA
L1:     CJNE R0,#00H,L2
          CPL P1.0
        MOV A,20H
        CLR C
          SUBB A,21H        ;Z=X-Y 或 Y=X-Z
          MOV 21H,A
        MOV R0,A
L2:       AJMP L1
TT0:    MOV TL0, 22H
          MOV TH0, 23H
          DEC R0
          RETI
```

3．直流电动机的驱动芯片

L293 是 SGS 公司的产品，内部包含 4 通道逻辑驱动电路。其额定工作电流为 1A，最大可达 1.5A，VSS 电压最小 4.5V，最大可达 36V；VS 电压最大值也是 36V，VS 电压应该比 VSS 电压高，否则有时会出现失控现象。

L298 芯片是一种 H 桥式驱动器，接受标准 TTL 逻辑电平信号，可用来驱动电感性负载。H 桥可承受 46V 电压，相电流高达 2.5A。L298（或 XQ298，SGS298）的逻辑电路使用 5V 电源，功放级使用 5～46V 电压，下桥发射极均单独引出，以便接入电流取样电阻。

14.4.3　应用电路设计

直流电机驱动采用 LM298 驱动芯片驱动，通过 PWM 算法控制直流电机状态，其电路图如 14-19 所示。

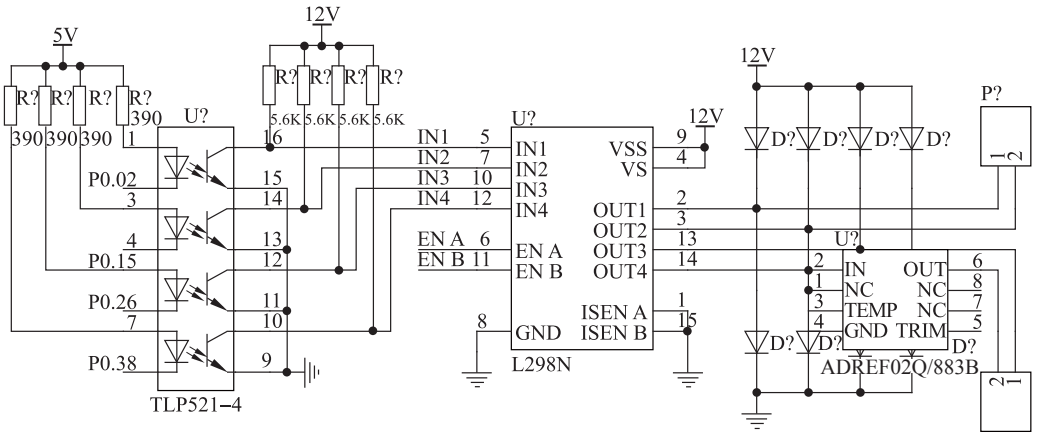

图 14-19　直流电机驱动电路图

14.4.4　软件程序设计

1．第一种编程方法

程序代码如下：

```
/* 晶振采用 11.0592M */
#include<reg51.h>
#include<math.h>
#define uchar unsigned char
#define uint unsigned int
sbit en1=P2^0;      /* L298 的 Enable A */
sbit en2=P2^1;      /* L298 的 Enable B */
sbit s1=P0^0;       /* L298 的 Input 1 */
sbit s2=P0^1;       /* L298 的 Input 2 */
sbit s3=P0^2;       /* L298 的 Input 3 */
sbit s4=P0^3;       /* L298 的 Input 4 */
uchar t=0;          /* 中断计数器 */
uchar m1=0;         /* 电机 1 速度值 */
uchar m2=0;         /* 电机 2 速度值 */
uchar tmp1,tmp2;    /* 电机当前速度值 */

/* 电机控制函数 index-电机号(1,2); speed - 电机速度(-100—100) */
void motor(uchar index, char speed)
{
      if(speed>=-100 && speed<=100)
      {
            if(index==1)        /* 电机 1 的处理 */
            {
                m1=abs(speed);  /* 取速度的绝对值 */
                if(speed<0)     /* 速度值为负则反转 */
                {
                    s1=0;
                    s2=1;
                }
```

```
                else                    /* 不为负数则正转 */
                {
                    s1=1;
                    s2=0;
                }
}
            if(index==2)                /* 电机 2 的处理 */
            {
                m2=abs(speed);          /* 电机 2 的速度控制 */
                if(speed<0)             /* 电机 2 的方向控制 */
                {
                    s3=0;
                    s4=1;
                }
                else
                {
                    s3=1;
                    s4=0;
                }
            }
      }
}

void delay(uint j)                      /* 简易延时函数 */
{
      for(j;j>0;j--);
}

void main()
{
      uchar i;
      TMOD=0x02;                        /* 设定 T0 的工作模式为 2 */
      TH0=0x9B;                         /* 装入定时器的初值 */
      TL0=0x9B;
      EA=1;                             /* 开中断 */
      ET0=1;                            /* 定时器 0 允许中断 */
      TR0=1;                            /* 启动定时器 0 */
      while(1)                          /* 电机实际控制演示 */
      {
            for(i=0;i<=100;i++)         /* 正转加速 */
            {
                motor(1,i);
                motor(2,i);
                delay(5000);
            }
            for(i=100;i>0;i--)          /* 正转减速 */
            {
                motor(1,i);
                motor(2,i);
                delay(5000);
            }
            for(i=0;i<=100;i++)         /* 反转加速 */
            {
                motor(1,-i);
                motor(2,-i);
                delay(5000);
            }
            for(i=100;i>0;i--)          /* 反转减速 */
```

```
            {
                motor(1,-i);
                motor(2,-i);
                delay(5000);
            }
      }
}
void timer0() interrupt 1           /* T0 中断服务程序 */
{
      if(t==0)                      /* 1 个 PWM 周期完成后才会接受新数值 */
      {
          tmp1=m1;
          tmp2=m2;
      }
      if(t<tmp1) en1=1; else en1=0; /* 产生电机 1 的 PWM 信号 */
      if(t<tmp2) en2=1; else en2=0; /* 产生电机 2 的 PWM 信号 */
      t++;
      if(t>=100) t=0;               /* 1 个 PWM 信号由 100 次中断产生 */
}
```

2．第二种编程方法

程序代码如下：

```
 /******单片机与外设端口定义**************/
sbit ql2 = P2^0;                //定义前侧传感器
sbit ql3 = P2^1;
sbit qr3 = P2^2;
sbit qr2 = P2^3;

sbit hl1 = P2^4;                //定义后侧传感器
sbit hl2 = P2^5;
sbit hr2 = P2^6;
sbit hr1 = P2^7;

sbit IN1 = P0^0;                //定义左轮驱动
sbit IN2 = P0^1;
sbit IN3 = P0^2;                //定义右轮驱动
sbit IN4 = P0^3;

sbit left = P3^6;               //控制左轮 PWM
sbit right = P3^7;              //控制右轮 PWM
uchar click,lmk,rmk,cs,time;    //定义 PWM 计数值、左轮脉宽、右轮脉宽、恒定转速、
                                //定时秒时间
void Softinit(void)
{
      TMOD = 0x12;              //定义 T1 为工作方式 1，构成 16 位定时器
                                //定义 T0 为工作方式 2，构成 8 位自动装载初值定时器
      TH0=0xD7                  //在晶振等于 24MHz 时，定时 20μs
                                //PWM 周期为 20μs*255=5.1ms，频率为 196Hz
       TL0 = 0xD7;
      TH1 = 0x3C;               //定时器定时 50ms，（65535-50000）/256
      TL1 = 0xAF;
      IP = 0x02;                //T0 中断优先级高，T1 次之
      ET0 = 1;                  //开定时器 T0
      ET1 = 1;                  //开定时器 T1
      EA = 1;                   //开总中断
```

```
      TR0 = 1;
      TR1 = 0;
      sec1 = 0;
      sec2 = 0;
      count = 0;
      count1 = 0;
      cs = 254;
      rmk = 160;                    //在给左右轮赋 PWM 值时，应留有余量以达到调节效果
      lmk = 160;
}
/********定义电机 PWM 调速函数*************/
void pdl(uint e1)                   //左转
{
    if( cs+e1 < 255 )
        rmk = cs+e1;
    else
        rmk=254;

    if( cs-e1 > 0 )
        lmk = cs-e1;
    else
           lmk=0;
}
void pdr(uint e1)                   //右转
{
    if( cs+e1 < 255 )
        lmk = cs+e1;
    else
        lmk=254;

    if( cs-e1 > 0 )
        rmk = cs-e1;
    else
           rmk=0;
}

/***************************************/
/***************定义前后侧传感器检测路面函数*************************/
/********************包括电机 PWM 调速函数************************/
void chuanganqi(bit m)              //定义传感器检测路面函数
{
    uchar e2;
    if(m == 0)
    {
        switch(P2 | 0xF0)           //前面传感器检测路面
        {
         case 0xF0:  e2 = 0;        //前进直行
                lmk = cs;
                rmk = cs;
                break;
         case 0xF4:   e2 = 170;     //小车向左偏调整，增大左轮脉宽值，同时减小右轮脉
                                    //宽值
                  pdr(e2);
                  break;
          case 0xF8:  e2 = 190;
                  pdr(e2);
                  break;
          case 0xF2:  e2 = 80;      //小车向右偏调整，增大右轮脉宽值，同时减小左轮脉
                                    //宽值
```

```
                pdl(e2);
                break;
         case 0xF1:  e2 = 100;
                pdl(e2);
                break;
       }
    }
    else                              //后面传感器检测路面
    {
        switch(P2 | 0x0F)
        {
         case 0x0F:    e2 = 0;      //后退直行
               lmk = cs;
               rmk = cs;
               break;
         case 0x4F:    e2 = 160;    //小车向左偏，增大左轮脉宽值，同时减小右轮脉
                                    //宽值
               pdr(e2);
               break;
           case 0x8F:  e2 = 170;
               pdr(e2);
               break;
          case 0x2F:   e2 = 150;    //小车向右偏，增大右轮脉宽值，同时减小左轮脉
                                    //宽值
               pdl(e2);
               break;
          case 0x1F: e2 = 170;
               pdl(e2);
               break;
        }
    }
}

/*****************************************/
/************定义控制电机函数***************/
void dianji(bit n)                  //定义电机驱动函数，控制小车前进和后退，由 P0
                                    //低 4 位控制 L298 的 I1～I4
{
    if(n == 0)
    {
        IN1 = 0;                    //小车前进
        IN2 = 1;
        IN3 = 0;
        IN4 = 1;
    }
    else
    {
        IN1 = 1;                    //小车后退
        IN2 = 0;
        IN3 = 1;
        IN4 = 0;
    }
}
void tingche(void)                  //停车函数
{
    IN1 = 0;
    IN2 = 0;
    IN3 = 0;
    IN4 = 0;
```

```
}
/*****定时器 T0 中断函数*********/
/*****产生 PWM 信号控制左右轮电机调速****/
/*****定时器定时 20μs，PWM 周期 20μs*255=5.1ms，频率 196Hz*****/
void time0(void) interrupt 1 using 0   //在晶振等于 24MHz 时，定时 20μs
                                       //PWM 周期为 20μs*255=5.1ms，频率为
                                       //196Hz
{
    click++;

    if(click < lmk)                    //左轮脉宽调整
        left = 1;
    else
        left = 0;

    if(click < rmk)                    //右轮脉宽调整
        right = 1;
    else
        right = 0;
}
```

14.5　RS-232 与 VB 串行通信

本节介绍 VB 软件的通信控件 MSComm 的功能，并设计 RS-232 通信应用电路和相应的代码。

14.5.1　VB 串行通信简介

VB（Micorsoft Visual Basic）是微软公司 1991 年推出的新一代 Basic 语言，它保留了原有 Basic 语言的功能和简单易用性，同时增加了图像处理、声音处理、文字处理、创建数据库和电子表格、数据通信等功能。VB 编程系统引入了部分面向对象的机制，提供了一种所见即所得的可视界面设计方法，使用户界面开发变得十分容易。

由于上位机的软件用 VB 来编写的，因此在上位机通信软件的设计中涉及如何用 VB 来实现上下位机之间的通信。一般用 VB 开发串行通信程序有两种方法：一种是利用 Windows 的通信 API 函数；另一种是采用 VB 标准控件 MSComm 来实现。

由于 VB 的通信控件 MSComm 友好、功能强大，提供了功能完善的串口数据的发送和接收功能，同时编程速度快是众人皆知的。在数据通信量不是很大时，在单片机通信领域广泛地使用 VB 的 MSComm 通信控件来开发 PC 上层通信软件。VB 6.0 的 MSComm 通信控件提供了一系列标准通信命令的接口，它允许建立串口连接，可以连接到其他通信设备（如 Modem），还可以发送命令、进行数据交换以及监视和响应在通信过程中可能发生的各种错误和事件，所以可以用它创建全双工的、事件驱动的、高效实用的通信程序。为了实现实时监测功能，数据采集处理程序采用 MSComm 事件方式。

将下位机采集的数据上传 PC，采用 MSComm 控件来实现。MSComm 控件为应用程序提供了串口通信功能，该应用程序允许通过串口发送和接收数据，信息会在系统的硬件线路上流动，此控件提供了两种方式处理信息的流动。

第一种方式是事件驱动，它是一种功能很强的处理串口活动的方法。在大多数情况下，用户需要获知事件发生的时间，在这种情况下，使用 MSComm 控件的 OnComm 事件捕获和处理这些通信事件和错误。

第二种方式是程序通过检查 CommEvent 属性的值来轮询事件和错误。

在 VB 6.0 开发环境中，MSComm 通信控件可以从 VB 的工具栏菜单的部件对话框中选择该控件添加到工具箱内，这样就可用其进行通信程序的设计。

13.5.2 应用电路设计

RS-232C 是美国电子工业协会（EIA）正式公布的，在异步串行通信中应用最广的标准总线。适用于终端设备（DTE）和数据通信设备（DCE）之间的接口。最高数据传送速率可达 19.2kb/s，最长传送电缆可达到 15m。RS-232C 标准定义了 9 根引线，对于一般的双向通信，只需使用串行输入 RXD、串行输出 TXD 和地线 GND。RS-232C 标准的电平采用负逻辑，规定+3～+15V 之间的任意电平为逻辑 0 电平，–3～–15V 之间的任意电平为逻辑 1 电平，与 TTL 和 CMOS 电平是不同的。在接口电路和计算机接口芯片中大都是 TTL/CMOS 电平，所以在通信时必须进行电平转换，以便与 RS-232C 标准的电平匹配。MAX232C 芯片可以完成电平转换这一工作。

采用 MAX232 的 RS-232 串行通信电路如图 14-20 所示，现选用其中的一路发送/接收。R1OUT 接 AT89C51 的 RXD，T1IN 接 AT89C51 的 TXD，T1OUT 接 PC 的 RD，R1IN 接 PC 的 TD。因为 MAX232 具有驱动能力，所以不需要外加驱动电路。

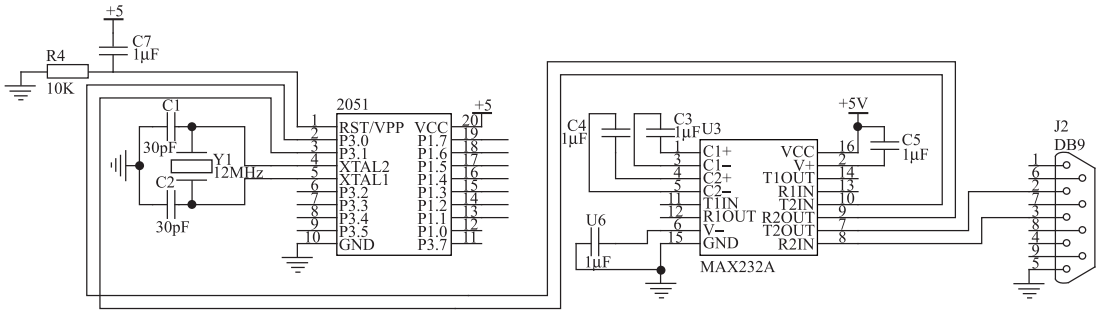

图 14-20 RS-232 串行通信电路

14.5.3 软件程序设计

1．单片机串口收发数据程序

程序代码如下：

```
#pragma SMALL
#include <reg52.h>
#include <absacc.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
```

```
char b[]={0x0,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x08,0x07,06,
05,04,03};
//*发送数据数组
void init(void)
{
    TMOD=0x21;   //0010 0001  T1 方式 2（常数自动装入的 8 位定时器/记数器）
                 // T0 方式 1（16 位定时器/计数器）
    TH1=0xf3;    //定时器确定的波特率 4800
    TL1=0xf3;
    TR1=1;       //启动定时器 1
    PCON=0x80;   //SMOD=1
    SCON=0x50;   //0101 0000 方式 1（10 异步收发，由定时器决定）
}
//*******************************************************//
void main(void)
{

    uint u,j;
    uint i;
    init();

    while(1)
    {
           while(RI==0);RI=0;
           i=SBUF;
           P1=SBUF;
           for(j=0;j<12500;j++);
           if (i==0x91)
           {
               for(u=0;u<16;u++)
               {

                   SBUF=b[u];
                   while(TI==0);TI=0;
                   for(j=0;j<12500;j++);
                }
             }
    }
}
```

2．VB程序收发数据程序

1）界面设计

接收数据界面如图 14-21 所示。

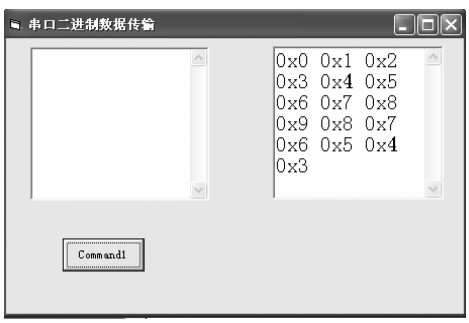

图 14-21　接收数据界面

2）MSComm1 属性设置

MSComm1 控件的属性如图 14-22 所示。

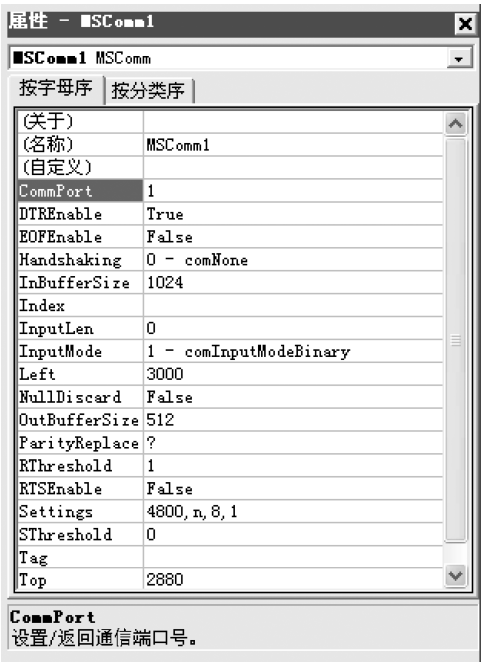

图 14-22　MSComm1 控件的属性

其中，串口号（Commport）设置为 1、2 等，表示 COM1、COM2。参数设置（Settings）格式为 B，P，D，S，其中 B 表示波特率，P 表示奇偶校验（N 为无校验，E 为偶校验，O 为奇校验），D 表示数据位数，S 表示停止位数。InputLen 设置或返回的是用 Input 从缓冲区每次读出的字节数组。

主要是设置端口号、波特率、数据位、停止位、奇偶校验位，以及以字节模式或二进制模式来发送或接收数据等属性。此外，在设置波特率时，上下位机收发双方要以相同的波特率工作。通过对 MSComm 的以上属性进行设置，可达成与下位机一致的通信协议。Settings 的属性设置根据下位机的波特率，奇偶校验位。数据位和停止位的值来确定。本设计与单片机通信格式设定波特率为 4800，无奇偶校验，8 位数据位，1 位停止位。数据传输采用二进制方式，以保证数据传输的可靠。设置好与单片机的通信格式，当缓存区接收到 9 个字符时，即引发 OnComm 事件。

3）程序代码

```
Private Sub Command1_Click()
MSComm1.Output = "1"
End Sub
Private Sub Form_Load()
MSComm1.PortOpen = True
End Sub

Private Sub MSComm1_OnComm()
   Dim InByte() As Byte
   Buf = ""
   Select Case MSComm1.CommEvent
```

```
      Case comEvReceive
        InByte = MSComm1.Input
        For i = LBound(InByte) To UBound(InByte)
          Buf = InByte(i)
          Text2.Text = Text2.Text & "0x" & Hex(Buf) & Chr(32)
        Next i
   End Select
End Sub
```

14.6　语音录放控制

美国 ISD 公司的 2500 芯片按录放时间 60s、75s、90s 和 120s 分成 ISD2560、ISD2575、ISD2590 和 ISD25120。ISD2500 系列和 1300 系列语音电路一样，具有抗断电、音质好，使用方便等优点。它的最大特点在于片内 E2PROM 容量为 480KB（1300 系列为 128KB），所以录放时间长；有 10 个地址输入端（1300 系列仅为 8 个），寻址能力可达 1024 位；最多能分 600 段；设有 OVF（溢出）端，便于多个器件级联。

ISD2560 是 ISD 系列单片语音录放集成电路的一种。这是一种永久记忆型语音录放电路，录音时间为 60s，可重复录放 10 万次。该芯片采用多电平直接模拟量存储专利技术，每个采样值可直接存储在片内单个 E^2PROM 单元中，能够非常真实自然地再现语音、音乐、音调和效果声，从而避免了一般固体录音电路因量化和压缩造成的量化噪声和“金属声”。该器件的采样频率为 8.0kHz，同一系列的产品采样频率越低，录放时间越长，但通频带和音质会有所降低。此外，ISD2560 省去了 A/D 和 D/A 转换器。其集成度较高，内部包括前置放大器、内部时钟、定时器、采样时钟、滤波器、自动增益控制、逻辑控制、模拟收发器、解码器和 480KB 的 E^2PROM。ISD2560 内部 E^2PROM 存储单元均匀分为 600 行，有 600 个地址单元，每个地址单元指向其中一行，每一个地址单元的地址分辨率为 100ms。ISD2560 还具备微控制器所需的控制接口。通过操纵地址和控制线可完成不同的任务，以实现复杂的信息处理功能，如信息的组合、连接、设定固定的信息段和信息管理等。ISD2560 可不分段，也可按最小段长为单位来任意组合分段。

14.6.1　ISD2560 引脚功能

ISD2560 的引脚如图 14-23 所示，下面介绍各引脚的主要功能。

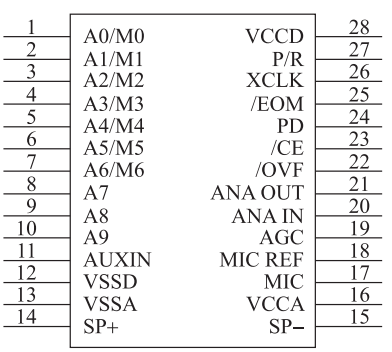

图 14-23　ISD2560 的引脚

（1）电源（VCCA、VCCD）：为了最大限度地减小噪声，芯片内部的模拟和数字电路使用不同的电源总线，并且分别引到外封装上。模拟和数字电源端最好分别走线，并应尽可能在靠近供电端处相连，而去耦电容则应尽量靠近芯片。

（2）地线（VSSA、VSSD）：由于芯片内部使用不同的模拟和数字地线，因此这两脚最好通过低阻抗通路连接到地。

（3）节电控制（PD）：该端拉高可使芯片停止工作，而进入节电状态。当芯片发生溢出，即 OVF 端输出低电平后，应将本端短暂变高以复位芯片；另外，PD 端在模式 6 下还有特殊的用途。

（4）片选（CE）：该端变低且 PD 也为低电平时，允许进行录、放操作。芯片在该端的下降沿将锁存地址线和 P/R 端的状态；另外，它在模式 6 中也有特殊的意义。

（5）录放模式（P/R）：该端状态一般在 CE 的下降沿锁存。高电平选择放音，低电平选择录音。录音时，由地址端提供起始地址，直到录音持续到 CE 或 PD 变高，或内存溢出；如果是前一种情况，芯片将自动在录音结束处写入 EOM 标志。放音时，由地址端提供起始地址，放音持续到 EOM 标志。如果 CE 一直为低，或芯片工作在某些操作模式，放音则会忽略 EOM 而继续进行下去，直到发生溢出为止。

（6）信息结尾标志（EOM）：EOM 标志在录音时由芯片自动插入到该信息段的结尾。当放音遇到 EOM 时，该端输出低电平脉冲。另外，ISD2560 芯片内部会自动检测电源电压以维护信息的完整性。当电压低于 3.5 V 时，该端变低，此时芯片只能放音。在模式状态下，可用来驱动 LED，以指示芯片当前的工作状态。

（7）溢出标志（OVF）：芯片处于存储空间末尾时，该端输出低电平脉冲以表示溢出，之后该端状态跟随 CE 端的状态，直到 PD 端变高。此外，该端可用于级联多个语音芯片来延长放音时间。

（8）话筒输入（MIC）：该端连至片内前置放大器。片内自动增益控制电路（AGC）可将增益控制在–15～24dB。外接话筒应通过串联电容耦合到该端。耦合电容值和该端的 10kΩ 输入阻抗决定了芯片频带的低频截止点。

（9）话筒参考（MIC REF）：该端是前置放大器的反向输入。当以差分形式连接话筒时，可减小噪声，并提高共模抑制比。

（10）自动增益控制（AGC）：AGC 可动态调整前置增益以补偿话筒输入电平的宽幅变化，这样在录制变化很大的音量（从耳语到喧嚣声）时就能保持最小失真。响应时间取决于该端内置的 5kΩ 电阻和从该端到 VSSA 端所接电容的时间常数。释放时间则取决于该端外接的并联对地电容和电阻设定的时间常数。选用标称值分别为 470kΩ 和 4.7μF 的电阻、电容可以得到满意的效果。

（11）模拟输出（ANA OUT）：前置放大器输出。其前置电压增益取决于 AGC 端电平。

（12）模拟输入（ANA IN）：该端为芯片录音信号输入。对话筒输入来说，ANA OUT 端应通过外接电容连至该端，该电容和本端的 3kΩ 输入阻抗决定了芯片频带的附加低端截止频率。其他音源可通过交流耦合直接连至该端。

（13）扬声器输出（SP+、SP–）；可驱动 16Ω以上的喇叭（内存放音时功率为 12.2mW，AUX IN 放音时功率为 50mW）。单端输出时必须在输出端和喇叭间接耦合电容，而双端输出则不用电容就能将功率提高至 4 倍。

（14）辅助输入（AUX IN）：当 CE 和 P/R 为高，不进行放音或处于放音溢出状态时，

该端的输入信号将通过内部功放驱动喇叭输出端。当多个 ISD2560 芯片级联时。后级的喇叭输出将通过该端连接到本级的输出放大器。为了防止噪声，建议在存放内存信息时。该端不要有驱动信号。

（15）外部时钟（XCLK）：该端内部有下拉元件，不用时应接地。

（16）地址/模式输入（AX/MX）：地址端的作用取决于最高两位（MSB，即 A8 和 A9）的状态。当最高两位中有一个为 0 时，所有输入均作为当前录音或放音的起始地址。地址端只作输入，不输出操作过程中的内部地址信息。地址在 CE 的下降沿锁存。当最高两位全为 1 时，A0～A6 可用于模式选择。

14.6.2　应用电路设计

语音控制电路如图 14-24 所示。单片机的 P0 口、P2.0 和 P2.1 引脚提供语音芯片 ISD2560 的地址/模式输入。P1.0 引脚控制语音芯片 ISD2560 的录/放模式选择，低电平位录音状态，高电平位放音状态。P1.2 引脚和语音芯片 ISD2560 的节电控制输入相连，单片机通过此引脚可以控制芯片的开关。P1.3 引脚和语音芯片 ISD2560 的片选，低电平时选中芯片。单片机的 INT0 脚、P1.4 和 ISD2560 的 EOM 标志输出相连，EOM 标志在录音时由芯片自动插入到录音信息的结尾处，放音遇到 EOM 时，会产生低电平脉冲，触发单片机中断，单片机必须在检测到此输出的上升沿后才播放新的录音，否则播放的语音就不连续，而且会产生杂声。

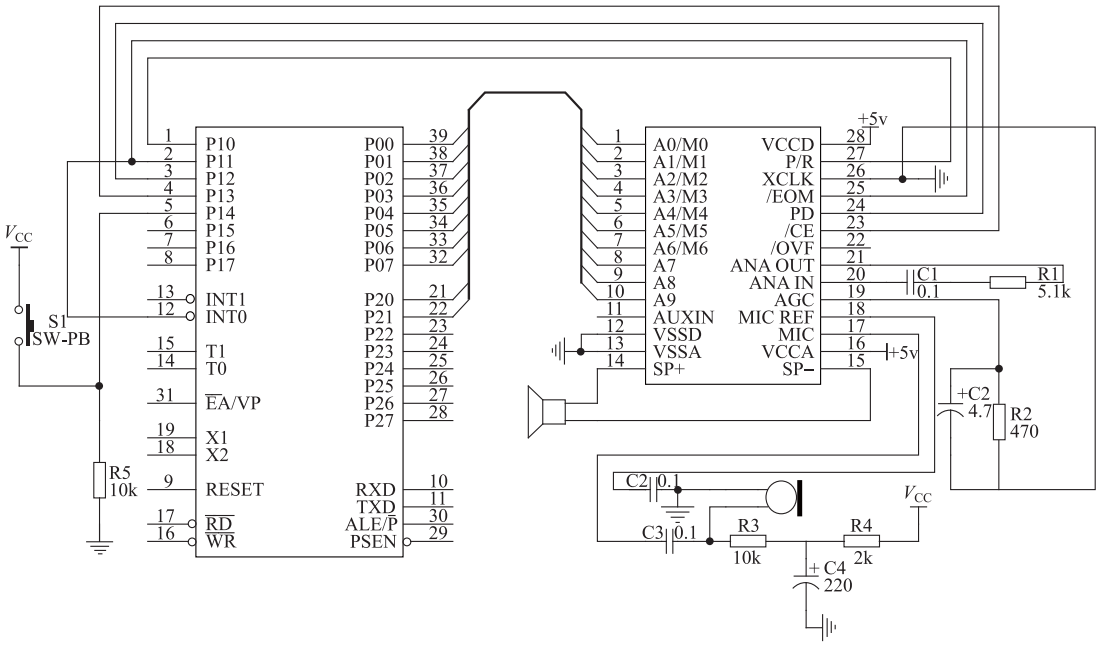

图 14-24　语音控制电路

14.6.3　软件程序设计

程序代码如下：

```
#pragma SMALL
#include <reg52.h>
#include <absacc.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar count
uchar startflag
uchar idleflag
//*定义 ISD2560 的控制引脚
sbit start=P1^4;
sbit eom= P1^1;
sbit PR= P1^0;
sbit PD= P1^2;
sbit CE= P1^3;
//*延时毫秒
void delay (uint t)
{
    uint i;
    while(t--)
    {
        for(i=0;i<125;i++)
        {}
    }
}
//*外部中断 0 程序
void out_int0 () interrupt 0 using 1
{
    EX0=0;
    pd=1;
    if(count<2)
    {
        count++;
        delay(5000);
        P2= P2&0XFC;
        P0= P0&0X00;
        Playback();
        EX0=1;
    }
    else
    {
        idleflag=1;
        count=0;
    }
}
//*主程序
void main()
{
     EA=1;
     count=0;
     startflag=0;
     idleflag=1;

     while(idleflag= =1)
     {
        if (start)
        {
           delay(10);
           if(start)
           startflag=1;
```

```
        }
        if (startflag= =1)
        {
           do
           {
               P2= P2&0XFC;
               P0= P0&0X00;
               Record();
           }
           while( start);
           startflag=0;
           PR=1;
           PD=1;
           delay(500);
           EX0=1;
           P2= P2&0XFC;
           P0= P0&0X00;
           playback();
           idleflag=0;
        }
    }
}
//*录音子函数
void record(void)
{
     ce=0;
     pd=0;
     pr=0;
}
//*放音子函数
void playback(void)
{
     ce=0;
     pd=0;
     pr=0;
}
```

14.7　思考与练习

1. 利用DS18B20检测水温或风扇温度，并在LED显示器或液晶显示器上显示此温度值。
2. 利用红外遥控来实现小车的前进、后退、左转弯以及右转弯功能。
3. 采用无线传输方式检测温度，并显示此温度值以及实现语音播报功能。