大连海事大学

课程设计报告

课程名称：计算机微机原理课程设计

成　　员 1：　2220161554　刘昊

成　　员 2：　2220160486　汪钇钰

设计时间：　2019.2.25…2019.3.8

目录

一、课程设计目的

《微机原理与汇编语言》是一门实践性和实用性都很强的课程，本次课程设计是在课程学习结束后，为使学生进一步巩固课堂和书本上所学知识，加强综合能力，充分理解和运用所学到的知识，通过简单的应用系统的设计，提高系统设计水平，启发创新思想。通过本课程设计希望达到以下目地：

(1)培养资料搜集和汇总的能力，培养总体设计和方案论证的意识；

(2)提高硬件，软件设计与开发的综合能力；

(3)提高软件和硬件联合调试的能力；

(4)熟练掌握相关测量仪器的使用方法；

(5)掌握相关开发软件，仿真软件的使用方法。

二、课程设计内容与要求

2.1 设计题目

双机数据采集系统设计

2.2 设计要求

设计一套结构如图 2.2-1 所示的双机数据采集系统。上位机负责向下位机下达命令，并对下位机采集的数据进行处理；下位机则根据上位机的命令对工作现场的各种开关量和模拟量数据进行采集，以某种通信方式传递给上位机，并根据上位机的命令对现场各种设备实施控制。发挥功能中还要用到远程监控机实现远程监控功能。

图 2.2-1 双机数据采集系统框图

所设计系统要求实现如下功能：

a.基本功能

实现数据采集，双机通信和基本形式的数据显示。功能要求：

(1)上位机发送一个启动命令给下位机；

(2)下位机接收到启动命令后开始周期性地（0.5 秒为周期）采集 1 路模拟量数据和 1 路开关量数据，并将模拟量转换成数字量在数码管上实时显示，然后把模拟量数据和开关量数据发送给上位机；

(3)上位机接收到数据后在显示器上显示。

(4)上位机发送停止命令给下位机，下位机结束采集。

b.扩展功能

在实现基本功能的基础上，编写可视化图形用户界面上位机程序。功能要求：

(1)下位机扩展为采集 2 路模拟量数据；

(2)上位机发送选择命令来决定下位机采集哪一路模拟量，并且上位机把收到的一路模拟量数值减半后实时回送给下位机；下位机把收到的回送来的数据进

行数模转换后用万用表或示波器显示。

(3)如果上位机收到的数据大于某个阈值，则以某种方式报警，并通知下位机点亮某个 LED 灯。

c.发挥功能

在实现扩展功能的基础上可选择增加的功能：

(1)上位机以图形化的方式显示模拟信号和对应的开关量信号；

(2)对双机通信中的数据包进行校验（如采用 CRC 校验）；

(3)上位机将采集到的数据通过网络口发送给远程监控机（PC 机或手机），能在远程监控机上显示。

(4)其他自己想发挥的功能等。

2.3 组织与实施

组员共同完成：

(1)根据所给任务，组员共同讨论得出芯片选择方案、端口分配方案、数据格式、硬件连线方案作为程序设计的参照。

(2)根据所给任务，组员共同讨论出下位机概要设计、上位机概要设计、上位机与下位机数据交互方式。明确整个系统的数据传输过程、数据处理。

(3)组员共同根据概要设计方案编写下位机程序、根据概要设计方案编写上位机程序。

2.4 评分标准

题目	
完成的主要工作 ①基本功能的全部实现：双机根据所规定数据格式周期性进行数据交互、命令收发、模拟量开关量传输、晶体管显示数据。 ②扩展功能的全部实现：选择线路并采集线路、上位机数据处理返回下位机、下位机模数转换并显示、阈值报警。 ③动态曲线显示模拟量和开关量的实现：上位机以图形化的方式显示模拟信号和对应的开关量信号	
成员分工 ①根据所给任务，组员共同讨论得出芯片选择方案、端口分配方案、数据格式、硬件连线方案作为程序设计的参照。 ②根据所给任务，组员共同讨论出下位机概要设计、上位机概要设计、上位机与下位机数据交互方式。明确整个系统的数据传输过程、数据处理。 ③组员共同根据概要设计方案编写下位机程序、根据概要设计方案编写上位机程序。	
综合评语（设计方案、实践环节、问题解答、设计报告）	
成绩 **成员 1 学号：** **成员 1 姓名：** **成员 2 学号：** **成员 2 姓名：**	

三、课程设计步骤

3.1 确定任务

(1)设计数据包格式
(2)选择硬件并分配端口
(3)概要设计程序
(4)编写下位机程序
(5)编写上位机程序
(6)硬件连线并运行测试
(7)设计远程监控机程序

3.2 总体/概要设计

3.2.1 设计数据包格式

数据包格式采用如图 3.2.1-1 所示形式。

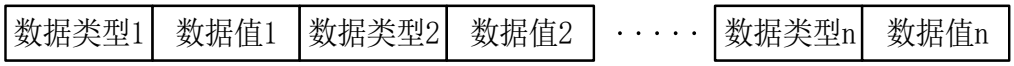

图 3.2.1-1 数据包格式

Figure 3.2.1-1 Packet format

3.2.2 选择硬件

硬件选择入表 3.2.2-1 所示。

硬件名称	数量
16 位微处理器 8086	1
可编程串行接口 8250	1
可编程计数器/定时器 8253	1
中断控制器 8259	1
模数转换器 ADC0809	1
模数转换器 DAC0832	1
3 态 8 位缓冲器 244	1
8 位数据/地址锁存器 273	1

表 3.2.2-1 硬件选择表

Table 3.2.2-1 Hardware Selection Table

3.2.3 芯片端口选择方案

CPU 端口使用情况如表 3.2.3-1 所示。

端口	连接芯片
0480h(内置)	8250
04a0h	8255
04b0h	273
04c0h	8259
04d0h	0809
04e0h	244
04f0h	保留

表 3.2.3-1 CPU 端口使用情况表

Table 3.2.3-1 CPU Port Usage Table

3.2.4 下位机程序概要设计

下位机程序基本步骤：

(1)初始化所有芯片

(2)等待 8250 接收数据并确定是哪一命令

(3)如果接收到@1.命令则利用 8253 和 8259 每 0.5s 产生一个中断，每次中断后利用 0809 将下位机电压采集转换为数字量并发送到上位机，再利用 244 采集开关量并发送至上位机。

(4)如果接收到@2.命令则利用 0809 采集电压值转换为数字量后发送到上位机，上位机会处理数据后发送回来，下位机接收处理后的数据再用 0832 转换为模拟量用电压表或示波器显示。

(5)如果接收到@s.命令则结束当前功能。

下位机程序流程图如图 3.2.4-1 所示。

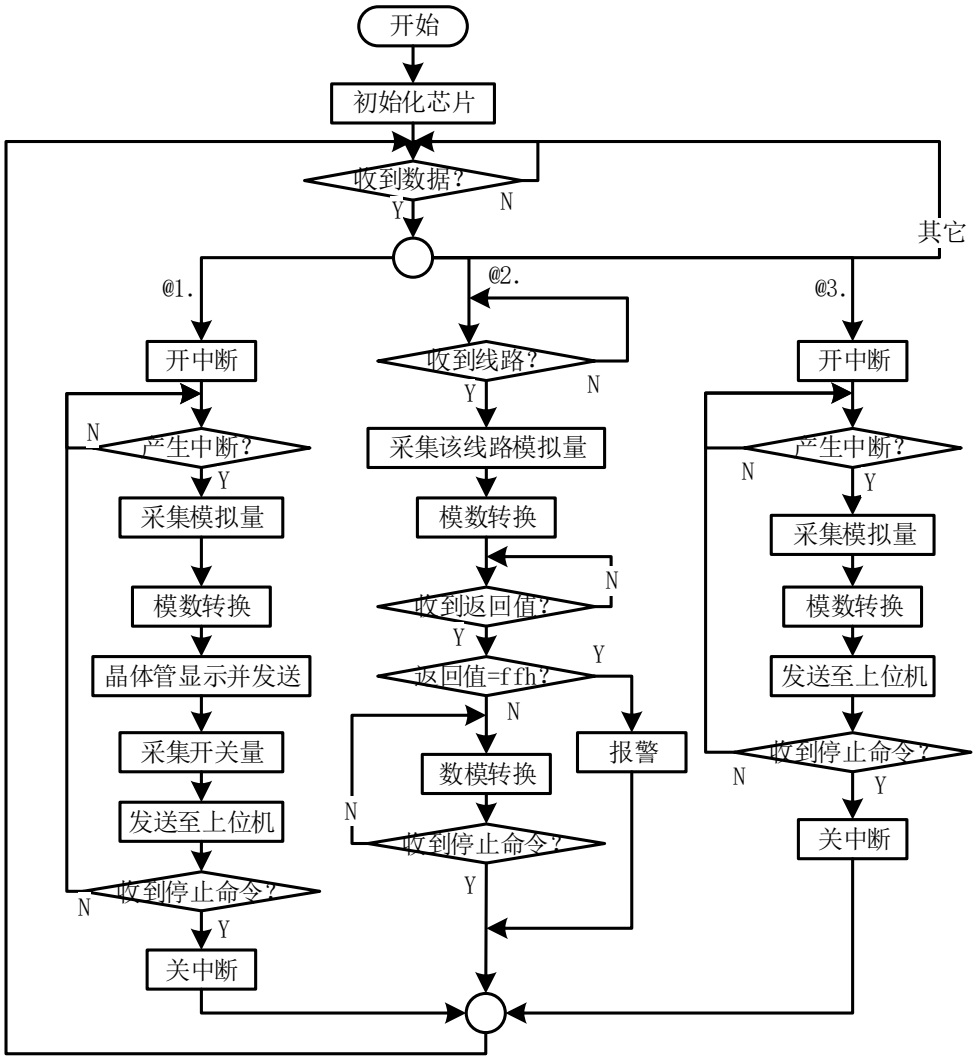

图 3.2.4-1 下位机程序流程图

Figure 3.2.4-1 Lower computer program flow chart

3.2.5 上位机程序概要设计

为了设计带图形用户界面的串口通信上位机程序，组员经过讨论选择了便于设计图形化界面且可用方便调用计算机底层接口的 C#作为开发语言。

上位机程序基本步骤：

(1) 寻找计算机可用串口，打开和关闭指定串口。

(2)获取串口基本状态并显示。

(3)实现基本串口数据收发功能，要求可以选择字符串收发和十六进制收发。

(4)通过按钮发送命令给下位机，处理下位机数据，与下位机交互以完成要求功能。

(5)将串口的开关量显示，将串口采集的模拟量以曲线图的形式动态显示。

3.2.6 远程监控机程序概要设计

组员经过讨论选择用远程访问Linux系统执行上位机程序来实现远程操作下位机，具体方案为将上位机程序后端提取，选择Linux系统上位机连接网络并于与下位机连接。则可用用任意操作系统(Windos/Linux/Mac/ Android/IOS等)通过SSH与上位机连接并执行程序。

由于实验室机器无法连接网络，本功能暂时未得到测试。

3.3 硬件研制过程

由于实验箱器件固定，硬件研制过程本小组进行较简单，试验箱接线如表3.3-1所示。

芯片	接线
8253	CS0<->CS8253 OUT0<->8253CLK2 OUT2<->IR0 CLK3<->8253CLK0
273	CS1<->CS273 O0…O1<->LED1…LED2
8259	CS2<->CS8259 INT<->INT INTA<->INTA
0809	CS3<->CS0809 AN0<->ADIN0 AN0<->ADIN1 EOC<->IN0
244	CS4<->CS244 IN0<->EOC IN7<->K1 IN6<->K2 IN5<->K3 IN4<->K4
0832	CS5<->CS0832

表3.3-1 试验箱接线表

Table 3.3-1 Test box wiring table

实际连线如图 3.3-1 所示。

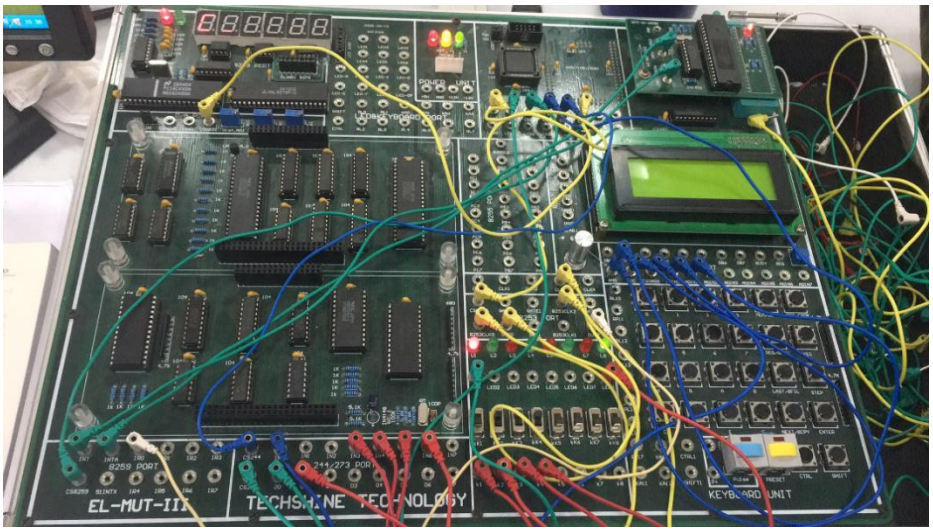

图 3.3-1 实验箱连线图

Figure 3.3-1 Experimental box connection diagram

3.4 软件研制过程

3.4.1 下位机-串行通信初始化模块

本系统用 8250 与上位机进行串行通信，初始化设置波特率为 9600、无中止符、无校验位、停止位 2 位、数据位 8 位、关闭所有中断。

初始化时线路控制寄存器(LCR)写入状态如图 3.4.1-1。

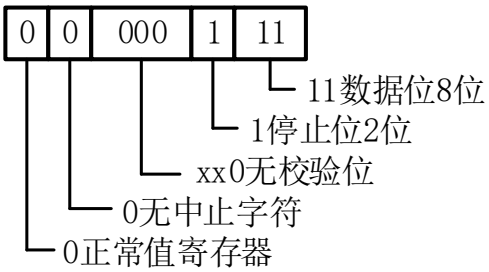

图 3.4.1-1 线路控制寄存器(LCR)写入状态图

Figure 3.4.1-1 Line Control Register (LCR) Write Status Diagram

初始化代码如下：

```
;1.设置线路控制寄存器(LCR)
mov  bx,0486h            ;0480h+3*2 LCR 的接口
mov  dx,bx
mov  ax,80h              ;10000000
                         ;1 除数寄存器 0 无中止字符 xx0 无校验位
                         ;0 停止位 1 位 00 数据位 5 位
out  dx,ax

;2.设置波特率
mov  bx,0480h
mov  dx,bx
```

```
mov  ax,0ch              ;00001100 = 9600
out  dx,ax

;3.设置中断允许寄存器(IER)
mov  bx,0482h            ;0480h+1*2 IER 的接口
mov  ax,0h               ;00000000 禁止产生所有中断
                         ;0000 0 禁止 Modem 状态变化中断
                         ;0 禁止接收数据出错状态中断
                         ;0 禁止发送保持寄存器空中断
                         ;0 禁止接收缓冲器满中断
out  dx,ax

;4.再次设置线路控制寄存器(LCR)
mov  bx,0486h            ;0480h+3*2 LCR 的接口
mov  dx,bx
mov  ax,07               ;00000111
                         ;0 正常值寄存器 0 无中止字符 xx0 无校验位
                         ;1 停止位 2 位 11 数据位 8 位
out  dx,ax
```

3.4.2 下位机-定时与中断初始化模块

8253 采用方式 2，设置计数器 0 计数值 927Ch(≈0.1s)，设置计数器 2 计数值 000ah，将 out 信号发送给 8259，达到周期为 0.1*5=0.5s 地产生中断。

8253 控制字如图 3.4.2-1 所示。

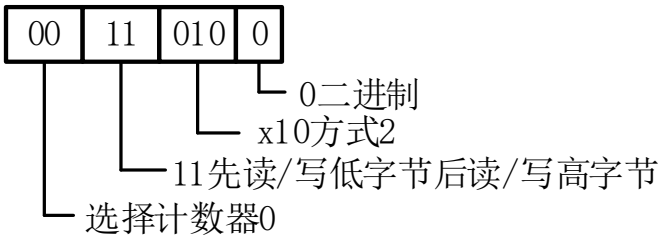

图 3.4.2-1 8253 控制字

Figure 3.4.2-1 8253 Control Word

8259ICW1 如图 3.4.2-2 所示。

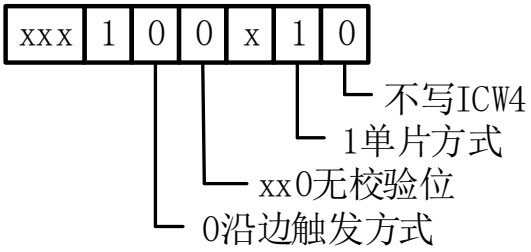

图 3.4.2-2 8259ICW1

Figure 3.4.2-2 8259ICW1

初始化代码如下：

```
cli                      ;先关所有中断

;1.设置 ICW1
mov  dx,04c0h            ;04c0h+0*2 为写 ICW1 接口
mov  ax,12h              ;00010010 xxx 1 0 沿边触发方式 x 1 单片方式 0 不写 ICW4
```

```
out  dx,ax

;2.设置 ICW2
mov  dx,04c2h            ;04c0h+1*2 第一次访问为写 ICW2 接口
mov  ax,80h              ;10000000 中断类型 80h
out  dx,ax

;3.设置 ICW3
;mov dx,04c2h            ;04c0h+1*2 没有级联时第二次访问为写 ICW4 接口
;mov ax,00h              ;00000000 000 0SFNM 中断为一般嵌套 0BUF 无缓冲 0 从片
                         ;AEOI 自动结束为不自动清除 ISR 0CPU 类型为 8080/8085/Z80
;out      dx,ax

;4.安装中断向量
mov  ax,0
mov  ds,ax               ;中断向量表位于内存最开始的 1KB，段地址为 0

mov  si,200h             ;初始化中断向量表，80H*4=200H
mov  ax,offset hint      ;中断函数的偏移地址
mov  ds:[si],ax          ;将中断函数的偏移地址存入地址 200h 内
mov  si,202h             ;si+1*2 为段地址
mov  ds:[si],100h        ;将中断函数的段地址 0100h 存入地址 202h 内
                         ;(代码段的内存起始地址为 01100H,代码段段地址 0100H)
```

3.4.3 下位机-接收上位机数据函数

通过 8250 接收上位机发送来的数据，其中可用先通过线路状态寄存器（LSR）的 D0 位来确定缓冲器是否收到数据，当 D0 为 1 时表示数据接收缓冲器收到一个数据，即接收数据准备好，当 CPU 读取完后，D0 为 0。

因此接收函数可先循环等待 D0 为 1 后开始进入接收部分。

接收函数代码如下：

```
func_receive:

mov  bx,0480h            ;8250 端口
mov  dx,bx
add  dx,0ah              ;8250 的 LSR 接口
wait2r:
in   al,dx
test al,01h              ;检验 D0，1 表示数据缓冲寄存器收到一个数据，CPU 读走数据后为 0
jnz  recvok              ;D0=1，接收缓冲器收到数据，开始接收数据。
jmp  wait2r              ;D0=0，循环等待接收
recvok:
mov  dx,bx
in   al,dx
ret
```

3.4.4 下位机-发送数据给上位机函数

通过 8250 将数据发送给上位机，其中可用通过线路状态寄存器（LSR）的 D5 来确定发送保持寄存器是否为空，如果 D0 为 1 表示发送保持寄存器空，当 CPU 将字符写入发送保持寄存器后该位为 0，等待上位机接收后该位重新为 1。

因此发送数据时可用先等待上一次数据被接收后即 D0=1 时再发送。

发送数据函数代码如下：

```
func_send:

push ax                  ;将需要发送的数据压入堆栈
mov  bx,0480h            ;8250 端口
mov  dx,bx
add  dx,0ah              ;8250 的 LSR 接口
wait2t:
in   al,dx
test al,20h              ;D5=1 表示发送保持寄存器为空，可接受下一个要发送的字符
                         ;CPU 将字符写入发送保持寄存器后，该位为 0
jnz  sendok              ;zf=0,表示 D5 为 1，开始发送
jmp  wait2t              ;否则，循环等待上一个字符被上位机接收
sendok:
```

```
pop  ax
mov  dx,bx
out  dx,ax
ret
```

3.4.5 下位机-模数转换模块

通过 0809 读取实验箱旋钮电压，其中可用通过读取转换结束信号 EOC 的值来确定转换是否完成。本小组将 EOC 接入 244 寄存器，通过读取 244 的值来读取 EOC 的值，从而判断是否转换完成。转换完成后再处理得到的数字量。

数模转换代码如下：

```
;0809 接 8086 端口为 04d0h
mov  dx,04d0h            ;04d0h+0*2 为 0809 线路 1 接口
mov  ax,34h              ;00110100 启动通道 0
out  dx,ax

wait0809finish_1:

mov  dx,04e0h            ;CS244 接 8086 端口为 04e0h
in   ax,dx
and  ax,1                ;读取转换结束信号 EOC
                         ;EOR 在 0809 在正在转换时为 0，其余时间为 1
cmp  ax,1
jne  wait0809finish_1    ;如果 EOC 为 0 说明还在转换则循环等待 0809 完成转换

mov  ax,00h
call func_send           ;先发送 00h 作为提示为模拟量

mov  dx,04d0h
in   ax,dx               ;读取转换结果
and  ax,0ffh             ;取低 8 位
```

3.4.5 下位机-采集开关量模块

通过开关量与 244 连接，需要采集开关量时读取 244 的值即可。这里由于在设计时没有考虑到模数转换模块已经占用了 244 的一个输入端，因此采集开关量时本小组仅采集前 4 个开关量。实际情况下，可用用 8255 芯片达到更好的效果。

同时，在测试时发现试验箱小灯泡高电平为暗，低电平为亮，因此在采集时要做一些简单的位处理。

采集开关量模块代码如下：

```
mov  dx,04e0h            ;CS244 接 8086 端口为 04e0h
in   ax,dx               ;读取开关值
and  ax,0f0h             ;后四个开关有效
xor  ax,0f0h             ;实际情况开关向上为 0，取反显示
and  ax,0ffh
```

3.4.6 下位机-数模转换模块

用 0832 进行数模转换，实验箱数模转换采用双缓冲方式，因此在转换时需要输入两次数据。

数模转换模块代码如下：

```
mov  dx,04a0h            ;0832 接 8086 端口为 04a0h
out  dx,ax               ;打开第一级锁存
add  dx,2
out  dx,ax               ;打开第二级锁存
```

3.4.7 下位机-晶体管显示模块

本系统使用通用的可编程序的键盘、显示接口器件 8279 芯片。

晶体管显示模块代码如下：

```
mov  di,offset segcod
mov  ax,08h              ;工作方式，16 位，左入
mov  dx,con8279
out  dx,ax
mov  ax,90h
mov  dx,con8279
```

```
out  dx,ax              ;写显示 RAM 命令，地址自增
mov  dx,dat8279
push bx
and  bx,0f0h            ;取高 4 位
mov  cl,4
shr  bx,cl
add  di,bx
mov  al,cs:[di]
mov  ah,0
out  dx,ax              ;写 RAM0
nop
nop
mov  di,offset segcod
pop  bx
and  bx,0fh             ;取低 4 位
add  di,bx
mov  al,cs:[di]
mov  ah,0
out  dx,ax              ;写 RAM1
```

3.4.8 下位机-阈值报警模块

本系统利用 LS273 与灯泡连接，当接收到超过阈值的报警命令时，会触发报警灯泡点亮暗交替。经过小组测试发现，本实验箱的灯泡是低电平为亮，高电平为灭。因此需要做一些简单的位处理。

阈值报警模块代码如下：

```
and  ax,0ffh            ;如果是命令是@2.，则进入报警功能模块
cmp  ax,0FFH
jz   alarm
noalarm:
push ax
mov  dx,04b0h           ;273 端口地址
mov  ax,-1
out  dx,ax
pop  ax
jmp  datashow
alarm:
push ax
and  ax,0ffh
mov  dx,04b0h           ;273 端口地址
xor  cx,0ffh
mov  ax,cx
out  dx,ax              ;LED 灯亮暗交替
pop  ax
```

3.4.9 上位机-界面设计

利用 VisualSudio2010(C#)根据所需要的功能画出上位机程序的用户图形界面，其中命令按钮在打开串口时是不可以访问的，防止出现错误，如图 3.4.9-1 所示。

图 3.4.9-1 上位机程序图形界面 1

Figure 3.4.9-1 Host computer program graphical interface 1

当打开串口后，命令按钮都可以访问，如图 3.4.9-2 所示。

图 3.4.9-2 上位机程序图形界面 2

Figure 3.4.9-2 Host computer program graphical interface 2

3.4.10 上位机-触发动作设计

(1)初始化窗口：打开程序后执行初始化窗口并扫利用 C#的 SerialPort 类的类方法 GetPortNames 描本计算机所有可用串口装载入串口选择框。

(2)打开串口：根据选择的串口参数打开所选串口，并赋值功能按钮 Enable 为 True 即可以访问。

(3)数据发送：点击发送按钮，利用 SerialPort 类的实例方法 Write 将数据以二进制或者字符串的形式发送给下位机。

(4)数据接收：数据接收是实时进行的，这里使用 SerialPort 类的属性 DataReceive 来安装数据接收触发函数。在实际测试中，本小组发现在.net 2.0 以后加强了安全机制,，不允许在 winform 中直接跨线程访问控件的属性。因此我们在初始化时加入 Control.CheckForIllegalCrossThreadCalls = false;以禁止跨线程访问控件属性。

(5)图形绘制：利用 C#的自带的 Chart 类可以方便的绘制图形，点击开始绘图后发送命令给下位机。在接收到数据时，如果开启了绘制功能，会将得到的数据绘制到动态显示屏上。并且将开关量实时现在在以选择框显示的开关状态上。

(6)数据采集：点击开始采集将受到的数据根据选择的功能操作后再发送给下位机。

四、实验结果

基本功能实验结果下图 4-1 所示。

图 4-1 基本功能实验结果图

Figure 4-1 Basic function experiment results

扩展功能实验结果下图 4-2 所示。

图 4-2 扩展功能实验结果图

Figure 4-2 Extended function experiment result graph

发挥功能实验结果下图 4-3 所示。

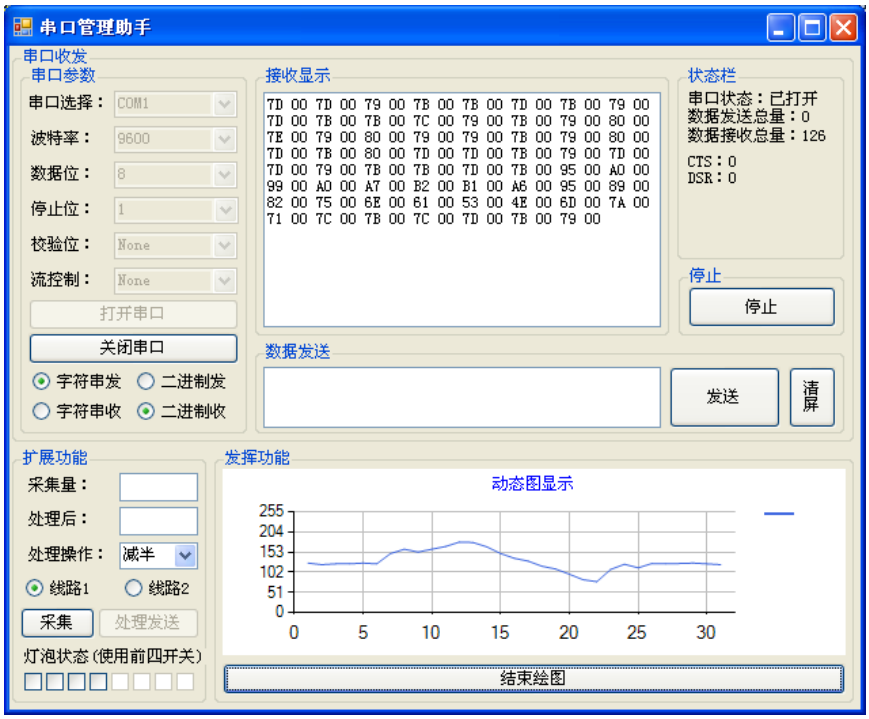

图 4-3 发挥功能实验结果图

Figure 4-3 Functional experiment results

五、实验总结

5.1 问题与解决

5.1.1 多指令选择问题

本系统要涉及到多个不同的支线以达到不同的功能，如果逐个比较各个字符，会导致与当某个字符比较失败后之前的所有字符都已释放，无法再与下一个指令比较。

因此小组经过讨论决定采用内存存储接收到的数据，再进行选择支线。其中temporder 用于存储命令，addr0809 用于存储两路模拟量的地址。

5.1.2 代码跳转越界

汇编语言条件转移指令转移地址的偏移量限制在-128～+127 字节范围内，采用相对转移方式（跳转时以当前地址为基准加上相对偏移量进行跳转，一般是在本地址段内跳转）。所以在整合所有模块后发现代码跳转越界。

小组讨论首先想到的办法时通过提取代码减少重复代码，但是效果不好。反思后，认为芯片初始化部分过长，使得偏移跳转太大。讨论后决定将初始化部分以子程序的形式运行。解决了代码跳转越界的问题。

5.1.3 停止命令的接收

由于程序的接收函数在没有收到函数时会一直循环等待接收函数，因此如果没有发送停止命令则会无限循环等待，导致程序不在进行。

小组讨论后决定在接收停止命令前先检测 LSR 的 D0 位，如果为 1 则调用接收命令，如果为 0 则不调用直接进入下一个循环。

5.2 不足与改进

5.2.1 LS244 输入缓冲寄存器冲突

本系统的模数转换模块和开关量采集模块都运用到了 LS244 输入寄存缓冲器，由于 LS244 输入寄存缓冲器仅有 8 个接口，所以无法采集所有开关量。

再总结实验时小组讨论得出用 8255 芯片更优。

5.2.2 模块间整合杂乱

在编写各个模块的时候都测试通过，但由于编写下位机程序经验不足，导致没有设计好结构就开始盲目的整合模块。最终导致程序测试、调式十分困难。以后的汇编程序编写将先好好设计结构后再编写。

六、附录

6.1 下位机程序

```
con8279  equ 0492h
dat8279  equ 0490h

code segment       ;define code segment
assume cs:code
org 0100h

start:
     jmp start1
     segcod db 3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh,77h,7ch,39h,5eh,79h,71h
     temporder dw ?
     addr0809  dw ?
start1:
;=========================8250 初始化=========================

;step1:write to the division register to set the baud rate

     mov  bx,0480h
     mov  dx,bx
     add  dx,6      ;LCR
     mov  ax,80h
     out  dx,ax

     mov  dx,bx
     mov  ax,0ch         ;000ch---9600      0cH=12
     out  dx,ax

     add  dx,2
     mov  ax,0h
     out  dx,ax

;step2:set the frame format
     add  dx,4      ;LCR again
     mov  ax,07          ;no pe(无校验位),8 bit, 1 stop
     out  dx,ax

;step3:disable interrupts
     mov  dx,bx
     add  dx,2      ;address of the Interrupt Enable register
     mov  ax,0
     out  dx,ax          ;禁止产生中断

;=========================8250 初始化=========================

;step4:receive and transmit
;=========================开始命令接收=========================
randt:
     call recv
     cmp  ax,'@'
     jnz  randt

     mov  temporder,01h      ;标记命令 1
     ;mov ax,88h
     ;call    send
     mov  addr0809,04d0h
     call recv
     cmp  ax,'1'
     jz   tail

     mov  temporder,02h      ;标记命令 2
     mov  addr0809,04d2h
     cmp  ax,'2'
     jnz  randt
```

```
tail:
     call recv
     cmp  ax,'.'
     jnz  randt
;=======================开始命令接收=======================


     call intr      ;中断
waiting:
     cmp  ax,55h
     jne  waiting       ;没发生中断，则等待


;=======================功能运行=======================
;>>>开始
     mov  dx,addr0809   ;(ADC0809 的地址)
        mov   ax,34h         ;(00110100)
        out   dx,ax          ;启动通道 0
wait1:
     mov  dx,04e0h  ;CS244
     in   ax,dx          ;读 EOC
        and   ax,1
        cmp   ax,1
        jne   wait1          ;如果 EOC=0,waiting....
     ;mov ax,00h
     ;call    send
        mov   dx,addr0809
        in    ax,dx          ;读转换结果
;-----------数码管显示-----------
         push ax
         and  ax,0ffh
        mov   bx,ax
        nop
disp:
         mov  di,offset segcod
         mov  ax,08h             ;工作方式，16 位，左入
         mov  dx,con8279
         out  dx,ax
         mov  ax,90h
         mov  dx,con8279
         out  dx,ax              ;写显示 RAM 命令，地址自增
         mov  dx,dat8279
         push bx
         and  bx,0f0h           ;取高 4 位
         mov  cl,4
         shr  bx,cl
         add  di,bx
         mov  al,cs:[di]
         mov  ah,0
         out  dx,ax               ;写 RAM0
         nop
         nop
         mov  di,offset segcod
         pop  bx
         and  bx,0fh             ;取低 4 位
         add  di,bx
         mov  al,cs:[di]
         mov  ah,0
         out  dx,ax                ;写 RAM1
         pop  ax
;-----------数码管显示-----------
     cmp  temporder,01h ;如果是命令是 02，则触发报警功能模块
     jz   datashow
;=======================超过阈值报警=======================
     and  ax,0ffh
     cmp  ax,3FH
     ja   alarm
noalarm:
     push ax
```

```
     mov  dx,04b0h  ;273 端口地址
     mov  ax,-1
     out  dx,ax
     pop  ax
     jmp  datashow
alarm:
     push ax
     and  ax,0ffh
     mov  dx,04b0h  ;273 端口地址
     mov  ax,0
     out  dx,ax
     pop  ax

;=======================超过阈值报警=======================
datashow:
     call send      ;发送给上位机

     cmp  temporder,02h ;如果是命令是 01，则采集开关量，命令 02 则 da 转换
     jz   datrans
;=======================上位机读取下位机开关值模块=======================
button:
     ;mov ax,11h
     ;call    send
     mov  dx,04e0h  ;CS244
     in   ax,dx          ;读开关值
     and  ax,0f0h        ;后四个开关有效
     xor  ax,0f0h        ;实际情况开关向上为 0,取反显示
     call send
     jmp recvd
;=======================上位机读取下位机开关值模块=======================
;--------------DA 转化---------------
datrans:
     ;call recv     ;接收上位机处理后发送的数据
     ;mov dx,04f0H    ;PORTA 为 D/A 端口号
     ;out dx,ax       ;往 D/A 输出数据
     ;add dx,2
     ;out dx,ax
     ;out dx,ax        ;往 D/A 输出数据
     jmp recvd      ;功能二结束
;--------------DA 转化---------------
;===============功能二结束===============
     ;mov ax,0
;=======================停止命令接收=======================
recvd:
     mov  bx,0480h
     mov  dx,bx
     add  dx,0ah         ;通信线路状态寄存器
     in   al,dx
     test al,01h         ;检验 D0 是否为 1，1 表示数据缓冲寄存器收到一个数据，CPU 读走数据后为 0
     jz   notrecv        ;(zf=0)若为 1，表示没有接收到数据

     call recv
     cmp  ax,'s'
     jnz  recvd

     jmp  randt

notrecv:
;=======================停止命令接收=======================

     jmp  waiting

;=======================中断服务程序=======================
hint:
     mov  ax,55h
     iret
;=======================中断服务程序=======================
```

```
;=========================recv 函数=========================
recv:
     mov  bx,0480h
     mov  dx,bx
     add  dx,0ah         ;通信线路状态寄存器
wait2r:
     in   al,dx
     test al,01h
;检验 D0 是否为 1，1 表示数据缓冲寄存器收到一个数据，CPU 读走数据后为 0
     jnz  recvok         ;(zf=0)若为 1，接受完成
     jmp  wait2r
recvok:
     mov  dx,bx
     in   al,dx
     RET
;=========================recv 函数=========================

;=========================send 函数=========================
send:
     push ax               ;char to be sent
     mov  bx,0480h
     mov  dx,bx
     add  dx,0ah           ;LSR
wait2t:
     in   al,dx
     test al,20h
;D5=1 表示发送保持寄存器为空，可接受下一个要发送的字符
;CPU 将字符写入发送保持寄存器后，该位为 0
     jnz  sendok         ;zf=0,表示 D5 为 1
     jmp  wait2t
sendok:
     pop  ax
     mov  dx,bx
     out  dx,ax
     ret
;=========================send 函数=========================


;====================0.5s 中断====================
intr proc
     push ax
     push bx
     push cx
     push dx

     mov  dx,04a6h ;控制寄存器   0000 0100 1010 0110
     mov  ax,34h         ;计数器 0，方式 2
     out  dx,ax

     mov  dx,04a0h ;0000 0100 1010 0000    计数器 0 的地址
     mov  ax,7Ch
     out  dx,ax
     mov  ax,92h
     out  dx,ax          ;计数值 927Ch

     mov  dx,04a6h ;控制寄存器 0000 0100 1010 0110
     mov  ax,0b4h        ;计数器 2，方式 2   1011 0100
     out  dx,ax
     mov  dx,04a4h ;计数器 2 的地址 0000 0100 1010 0100
     mov  ax,0Ah
     out  dx,ax
     mov  ax,0      ;计数值 0Ah
     out  dx,ax

next:
     nop
     Cli

;8259 初始化
```

```
    mov  dx,04c0h
    mov  ax,13h         ;ICW1, ICW4 NEEDED
    out  dx,ax

    mov  dx,04c2h
    mov  ax,80h         ;ICW2 中断类型 80h
    out  dx,ax

    mov  ax,03H
    out  dx,ax          ;ICW4

    mov  ax,00h         ;OCW1, 开放所有中断
    out  dx,ax

;安装中断向量
    mov  ax,0
    mov  ds,ax          ;中断向量表位于内存最开始的 1KB，段地址为 0

    mov  si,200h        ;初始化中断向量表，80H*4=200H
    mov  ax,offset hint
    mov  ds:[si],ax     ;偏移地址
    add  si,2
    mov  ds:[si],100h   ;代码段的内存起始地址为 01100H,代码段段地址 0100H

    mov  cx,0      ;指示灯输出
    mov  ax,0      ;中断标识
    sti
    pop  dx
    pop  cx
    pop  bx
    pop  ax
    RET
intr ENDP
;====================0.5s 中断====================

;========================程序终止========================
stop:
    code ends           ;end of code segment
    end  start              ;end assembly
;========================程序终止========================
```

6.2 上位机程序

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Windows.Forms.DataVisualization.Charting;

namespace 串口管理程序
{
    public partial class Form1 : Form
    {
        //串口收发功能变量声明
        private SerialPort serialPorts;                     //串口通信类
        private string[] portNames = null;                  //字符串数组，存储可用串口名
        private byte[] portBuffer;                          //收发数据的二进制缓存区
        private int sendNumber = 0;                         //数据发送总量
        private int revNumber = 0;                          //数据接收总量

        //模拟量动态图显示变量声明
        private Queue<double> dataQueue = new Queue<double>(1000);

        //判断接收的是开关量还是模拟量
        private int isAnalog = 1;
```

```
private int isSwitchingValue = 0;
//
private int nNum = 1;

//界面构建
public Form1()
{
    InitializeComponent();

    //动态图初始化
    InitChart();

    //这个类中我们不检查跨线程的调用是否合法
    //因为.net 2.0 以后加强了安全机制,, 不允许在 winform 中直接跨线程访问控件的属性
    Control.CheckForIllegalCrossThreadCalls = false;
}

//窗体初始化
private void Form1_Load(object sender, EventArgs e)
{
    //变量定义
    serialPorts = new SerialPort();
    this.portBuffer = new byte[10000];
    //SerialPort.GetPortNames 返回当前计算机的串口端口名数组
    this.portNames = SerialPort.GetPortNames();

    //将可用串口显示在串口列表中
    if (this.portNames.Length > 0)
    {
        this.comboBox1.Items.AddRange(this.portNames);
    }
    else
    {
        MessageBox.Show("未检测到当前计算机的串口！", "提示");
    }

}

//按下打开串口按钮
private void button1_Click(object sender, EventArgs e)
{
    if (this.comboBox1.Text == "")
    {

        MessageBox.Show("请先选择串口号！","提示");

        return;
    }
    try
    {
        this.serialPorts.PortName = this.comboBox1.Text.ToString();
        if (!this.serialPorts.IsOpen)
        {
            this.serialPorts.BaudRate = int.Parse(comboBox2.Text);          //波特率
            this.serialPorts.DataBits = int.Parse(comboBox3.Text);          //数据位
            this.serialPorts.StopBits = (StopBits)int.Parse(comboBox4.Text);//停止位
            //this.serialPorts.Parity = (Parity)string.Parse(comboBox5.Text);//校验位
            //this.serialPorts.Parity = (Parity)int.Parse(comboBox5.Text);  //校验位
            //this.serialPorts.Handshake
            this.serialPorts.Open();
            this.label2.Text = "串口状态：已打开";
            this.button1.Enabled = false;
            this.button2.Enabled = true;
            this.button3.Enabled = true;
            this.button5.Enabled = true;
            this.button6.Enabled = true;
            this.button8.Enabled = true;
            this.comboBox1.Enabled = false;
            this.comboBox2.Enabled = false;
```

```
                    this.comboBox3.Enabled = false;
                    this.comboBox4.Enabled = false;
                    this.comboBox5.Enabled = false;
                    this.comboBox6.Enabled = false;

                    //timer 计时器开始
                    this.timer1.Start();

                    //串口接收处理函数
                    serialPorts.DataReceived+=new
SerialDataReceivedEventHandler(serialport_DataReceived);

                }

            }
            catch (IOException eio)
            {
                MessageBox.Show("打开串口异常：" + eio);
            }
        }

        //按下关闭串口按钮
        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                if (this.serialPorts.IsOpen)
                {
                    this.serialPorts.Close();
                    this.label2.Text = "串口状态：未打开";
                    this.button1.Enabled = true;
                    this.button2.Enabled = false;
                    this.button3.Enabled = false;
                    this.button5.Enabled = false;
                    this.button6.Enabled = false;
                    this.button8.Enabled = false;
                    this.comboBox1.Enabled = true;
                    this.comboBox2.Enabled = true;
                    this.comboBox3.Enabled = true;
                    this.comboBox4.Enabled = true;
                    this.comboBox5.Enabled = true;
                    this.comboBox6.Enabled = true;

                    //timer 计时器结束
                    this.timer1.Stop();

                }
            }
            catch (IOException eio)
            {
                MessageBox.Show("打开串口异常：" + eio);
            }
        }

        //按下发送按钮
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                if (this.serialPorts.IsOpen)
                {

                    string sendContent = this.textBox2.Text.ToString();

                    //字符串形式发送
                    if (this.radioButton1.Checked)
                    {
                        this.serialPorts.Write(sendContent);
                        this.sendNumber += sendContent.Length;                //记录发送量
```

```
            }

            //十六进制形式发送(待测试)
            //将输入的字符串按照空格逗号分组
            else
            {
                string sendNoNull = sendContent.Trim();
                string sendNoComma = sendNoNull.Replace(',', ' ');
                string sendNoComma1 = sendNoComma.Replace("0x", "");
                string sendNoComma2 = sendNoComma1.Replace("0X", "");

            }
            /*else
            {
                Byte[] sendContent = new Byte[1];
                sendContent[0] = Byte.Parse(this.textBox2.Text);
                this.serialPorts.Write(sendContent,0,1);
            }
            */
            this.label8.Text = "数据发送总量：" + this.sendNumber;
        }
        else
        {
            MessageBox.Show("请先打开串口","提示");
        }

    }
    catch (IOException eio)
    {
        MessageBox.Show("串口发送异常：" + eio);
    }
}
/* SerialPort::Write()函数MSDN：
 * Write(Byte[],Int32,Inte32)：使用缓冲区中的数据将指定数量的字节写入串行端口
 * Write(char[],Int32,Inte32)：使用缓冲区中的数据将指定数量的字节写入串行端口
 * Write(String)：将指定的字符串写入串行端口
 */

private void timer1_Tick(object sender, EventArgs e)
{
    if (serialPorts.CtsHolding)
    {
        label10.Text = "CTS：1";
    }
    else
    {
        label10.Text = "CTS：0";
    }

    if (serialPorts.DsrHolding)
    {
        label11.Text = "DSR：1";
    }
    else
    {
        label11.Text = "DSR：0";
    }

}

//串口接收处理函数
private void serialport_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    //二进制接收数据
    if (radioButton4.Checked)
    {
        int receivedContent = serialPorts.ReadByte();
        this.revNumber += 1;
        this.textBox1.Text += receivedContent.ToString("X2");
```

```
                this.textBox1.Text += " ";

                //如果开启动态图显示功能且为奇数个收到的数据
                if (this.button6.Text == "结束绘图" && this.revNumber % 2 == 1)
                {
                    //大于 100 个数先出列达到动态图效果
                    if (this.dataQueue.Count > 30)
                    {
                        this.dataQueue.Dequeue();
                    }
                    //添加采集数
                    this.dataQueue.Enqueue(receivedContent);

                    this.chart1.Series[0].Points.Clear();
                    for (int i = 0; i < this.dataQueue.Count; i++)
                    {
                        this.chart1.Series[0].Points.AddXY((i+1),
this.dataQueue.ElementAt(i));
                    }

                    //isAnalog 置零
                    //this.isAnalog = 0;
                }

                //如果开启动态图显示功能且为偶数个收到的数据
                if (this.button6.Text == "结束绘图" && this.revNumber % 2 == 0)
                {
                    //
                    showLight(receivedContent);
                    //isSwitchingValue 置零
                    //this.isSwitchingValue = 0;
                    //this.isAnalog = 1;
                }

                //如果开启了采集功能
                if(this.button5.Text == "结束采集")
                {
                    this.textBox3.Text = receivedContent.ToString();
                    int returnNum = receivedContent / 2;
                    this.textBox4.Text = returnNum.ToString();
                    this.serialPorts.Write(Convert.ToString((char)returnNum));

                }

            }

            //字符串接收数据
            else
            {
                string receivedContent = serialPorts.ReadExisting();           //字符串接收
                this.revNumber += 1;
                this.textBox1.Text += receivedContent;
                this.textBox1.Text += " ";
            }

            this.label16.Text = "数据接收总量：" + this.revNumber;


        }

        //按下清屏按钮
        private void button4_Click(object sender, EventArgs e)
        {
            this.textBox1.Text = "";
        }

        //按下采集按钮
        private void button5_Click(object sender, EventArgs e)
        {
```

```
    if(this.button5.Text == "开始采集")
    {
        if (!radioButton4.Checked)
        {
            MessageBox.Show("本功能只在二进制接收数据下开启", "提示");
        }
        else
        {
            this.button5.Text = "结束采集";
            this.serialPorts.Write("@2.");
        }
    }
    else
    {
        this.button5.Text = "开始采集";
    }
}


/// <summary>
/// 初始化动态图
/// </summary>
private void InitChart()
{
    //定义图表区域
    this.chart1.ChartAreas.Clear();
    ChartArea chartArea1 = new ChartArea("C1");
    this.chart1.ChartAreas.Add(chartArea1);
    //定义存储和显示点的容器
    this.chart1.Series.Clear();
    Series series1 = new Series(" ");
    series1.ChartArea = "C1";
    this.chart1.Series.Add(series1);
    //设置图表显示样式
    this.chart1.ChartAreas[0].AxisY.Minimum = 0;
    this.chart1.ChartAreas[0].AxisY.Maximum = 255;
    this.chart1.ChartAreas[0].AxisX.Interval = 5;
    this.chart1.ChartAreas[0].AxisX.MajorGrid.LineColor=System.Drawing.Color.Silver;
    this.chart1.ChartAreas[0].AxisY.MajorGrid.LineColor=System.Drawing.Color.Silver;
    //设置标题
    this.chart1.Titles.Clear();
    this.chart1.Titles.Add("S01");
    this.chart1.Titles[0].Text = "模拟量显示";
    this.chart1.Titles[0].ForeColor = Color.Blue;
    this.chart1.Titles[0].Font = new System.Drawing.Font("Microsoft Sans Serif", 8F);
    //设置图表显示样式
    this.chart1.Series[0].Color = Color.RoyalBlue;

    this.chart1.Titles[0].Text = "动态图显示";
    this.chart1.Series[0].ChartType = SeriesChartType.Line;

    /*
    if (rb1.Checked)
    {
        this.chart1.Titles[0].Text = string.Format("模拟量 {0} 显示", rb1.Text);
        this.chart1.Series[0].ChartType = SeriesChartType.Line;
    }
    //if (rb2.Checked) {
    //    this.chart1.Titles[0].Text = string.Format("XXX {0} 显示", rb2.Text);
    //    this.chart1.Series[0].ChartType = SeriesChartType.Spline;
    //}
    */
    this.chart1.Series[0].Points.Clear();
}

//点击开始绘图按钮
private void button6_Click(object sender, EventArgs e)
{
    if (this.button6.Text == "开始绘图")
```

```
    {
        if (!radioButton4.Checked)
        {
            MessageBox.Show("本功能只在二进制接收数据下开启", "提示");
        }
        else
        {
            this.button6.Text = "结束绘图";
            this.serialPorts.Write("@1.");
        }
    }
    else
    {
        this.button6.Text = "开始绘图";
    }


}

private void radioButton3_CheckedChanged(object sender, EventArgs e)
{

}

private void chart1_Click(object sender, EventArgs e)
{

}

//按下停止按钮
private void button8_Click(object sender, EventArgs e)
{
    //发送停止命令，命令可用自定义
    this.serialPorts.Write("s");
}

private void showLight(int lightState)
{
    lightState = lightState / 16;

    if (lightState % 2 == 0)
    {
        checkBox4.Checked = false;
    }
    else
    {
        checkBox4.Checked = true;
    }

    lightState = lightState / 2;
    if (lightState % 2 == 0)
    {
        checkBox3.Checked = false;
    }
    else
    {
        checkBox3.Checked = true;
    }

    lightState = lightState / 2;
    if (lightState % 2 == 0)
    {
        checkBox2.Checked = false;
    }
    else
    {
        checkBox2.Checked = true;
    }
```

```
            lightState = lightState / 2;
            if (lightState % 2 == 0)
            {
                checkBox1.Checked = false;
            }
            else
            {
                checkBox1.Checked = true;
            }
        }
    }
}
```