温度监控系统的设计

引言

随着“信息时代”的到来，作为获取信息的手段——传感器技术得到了显著的进步，其应用领域越来越广泛，对其要求越来越高，需求越来越迫切。传感器技术已成为衡量一个国家科学技术发展水平的重要标志之一。因此，了解并掌握各类传感器的基本结构、工作原理及特性是非常重要的。

由于传感器能将各种物理量、化学量和生物量等信号转变为电信号，使得人们可以利用计算机实现自动测量、信息处理和自动控制，但是它们都不同程度地存在温漂和非线性等影响因素。传感器主要用于测量和控制系统，它的性能好坏直接影响系统的性能。因此，不仅必须掌握各类传感器的结构、原理及其性能指标，还必须懂得传感器经过适当的接口电路调整才能满足信号的处理、显示和控制的要求，而且只有通过对传感器应用实例的原理和智能传感器实例的分析了解，才能将传感器和信息通信和信息处理结合起来，适应传感器的生产、研制、开发和应用。另一方面，传感器的被测信号来自于各个应用领域，每个领域都为了改革生产力、提高工效和时效，各自都在开发研制适合应用的传感器，于是种类繁多的新型传感器及传感器系统不断涌现。温度传感器是其中重要的一类传感器。其发展速度之快，以及其应用之广，并且还有很大潜力。

为了提高对传感器的认识和了解，尤其是对温度传感器的深入研究以及其用法与用途，基于实用、广泛和典型的原则而设计了本系统。本文利用单片机结合传感器技术而开发设计了这一温度监控系统。文中传感器理论单片机实际应用有机结合，详细地讲述了利用热敏电阻作为热敏传感器探测环境温度的过程，以及实现热电转换的原理过程。

本设计应用性比较强，设计系统可以作为生物培养液温度监控系统，如果稍微改装可以做热水器温度调节系统、实验室温度监控系统等等。课题主要任务是完成环境温度检测，利用单片机实现温度调节并通过计算机实施温度监控。设计后的系统具有操作方便，控制灵活等优点。

本设计系统包括温度传感器，A/D 转换模块，输出控制模块，数据传输模块，温度显示模块和温度调节驱动电路六个部分。文中对每个部分功能、实现过程作了详细介绍。整个系统的核心是进行温度监控，完成了课题所有要求。

1 设计要求

1.1 控制要求

（1）生物繁殖培养液的温度要保证在适于细胞繁殖的温度内，这主要在控制程序设计中考虑。温度控制范围为 15 ～25，升温、降温阶段的温度控制精度要求为 0.5 度，保温阶段温度控制精度为 0.5 度 。

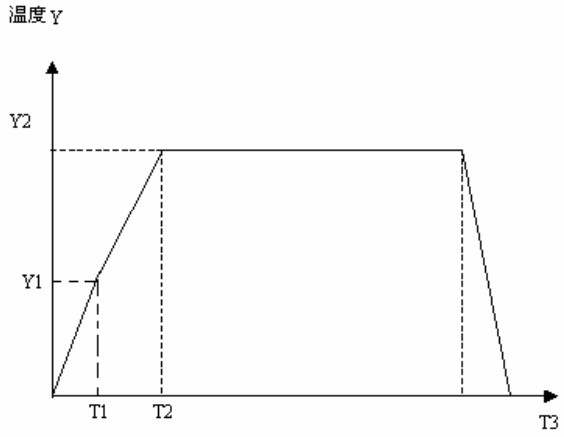


图 1.1.1 温度控制曲线

（2）微机自动调节 正常情况下，系统投入自动。

（3）模拟手动操作 当系统发生异常，投入手动操作。

（4）微机监控功能 显示当前被控量的设定值、实际值，控制量的输出。

1.2 受控对象的数学模型

生物繁殖的培养液主要用于生物的繁殖研究，而温度是影响生物繁殖的重要因素。本系统要求长时间监视培养液的温度，并对当前的温度进行控制。本控制对象为生物繁殖用培养液，采用继电器进行控制。

2 系统的硬件配置

2.1 单片机和系统总线

单片机：PIC16F877A（PIC16F877A 为美国 MICORCHIP 公司生产的带 A/D 转换的 8 位单片机）。

显示系统：商用计算机。

用户内存：256M RAM。

系统总线：RS-232-C 接口（又称 EIA RS-232-C）RS232 C 有 25 条线，，分为 5 个功能组，包括 4 条数据线，11 条控制线，3 条定时线，7 条备用线和未定义线。

操作系统：Windows 2000。

2.2 硬件介绍

计算机工作的外围电路设备

（1）温度传感器

温度传感器采用补偿型 NTC 热敏电阻其主要性能如下：

①补偿型 NTC 热敏电阻 B 值误差范围小，对于阻值误差范围在 5%的产品，其一致性、互换性良好。适合于一般精度的温度测量和计量设备。

②外型结构和尺寸：

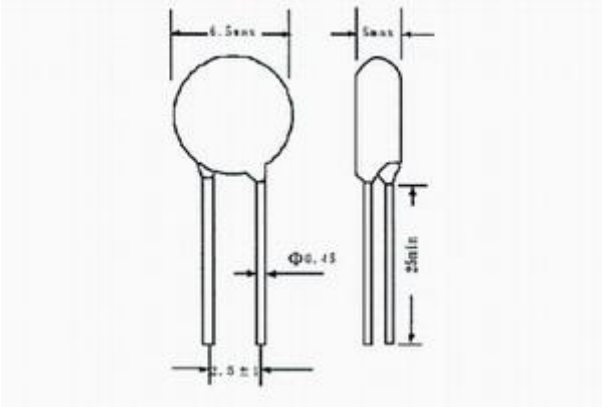


图 2.2.1 温度传感器结构尺寸图

③主要技术参数：

时间常数≤30S

测量功率≤0.1mW

使用温度范围-55～+125℃

耗散系数≥6mW/℃

额定功率 0.5W

④降功耗曲线：

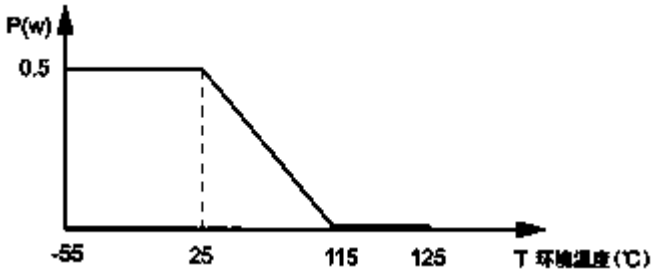


图 2.2.2 温度传感器功耗曲线图

（2）核心处理单元 MicroChip PIC16F877A 单片机

MicroChip PCI16F877A 单片机主要性能：

具有高性能 RISC CPU

仅有 35 条单字指令。

除程序指令为两个周期外，其余的均为单周期指令。

运行速度：DC-20M 时钟输入。

DC-200ns 指令周期。

8K*14 个 FLASH 程序存储器。

368*8 个数据存储器（RAM）字节。

引脚输出和 PIC16C73B/74B/76/77 兼容。

中断能力（达到 14 个中断源）。

8 级深度的硬件堆栈。

直接，间接和相对寻址方式。

上电复位（POR）。

上电定时器(PWRT)和震动启动定时器。

监视定时器（WDT），它带有片内可靠运行的 RC 振荡器。

可编程的代码保护。

低功耗睡眠方式。

可选择的振荡器。

低功耗，高速 CMOS FLASH/EEPROM 工艺。

全静态设计。

在线串行编程(ICSP)。

单独 5v 的内部电路串行编程(ICSP)能力。

处理机读/写访问程序存储器。

运行电压范围 2.0v 到 5v。

高输入/输出电流 25mA。

商用，工业用温度范围。

低功耗：

在 5v,4MHz 时典型值小于 2mA。

在 3v,32KHz 时典型值小于 20uA。

典型的静态电流值小于 1uA。

外围特征：

Timer 0 ：带有预分频的 8 位定时器/计数器。

Timer 1 ：带有预分频的 16 位定时器/计数器，在使用外部晶体时钟时在 SLEEP 期间仍能工作。

Timer 2 ：带有 8 位周期寄存器，预分频和后分频器的 8 位定时器/计数器 2 个捕捉器，比较器和 PWM 模块。

其中 ：捕捉器是 16 位的，最大分辨率为 12.5nS。

比较器是 16 位的，最大分辨率为 200nS。

PWM 最大分辨率为是 10 位。

10 位多通道模/数转换器。

带有 SPI（主模式）和 I2C（主/从）模式的 SSP。

带有 9 位地址探测的通用同步异步接收/发送（USART/RCI）。

带有 RD,WR 和 CS 控制（只 40/44 引脚）8 位字宽的并行从端口。

带有降压的复位检测电路。

（3）RS-232-C 接口电路

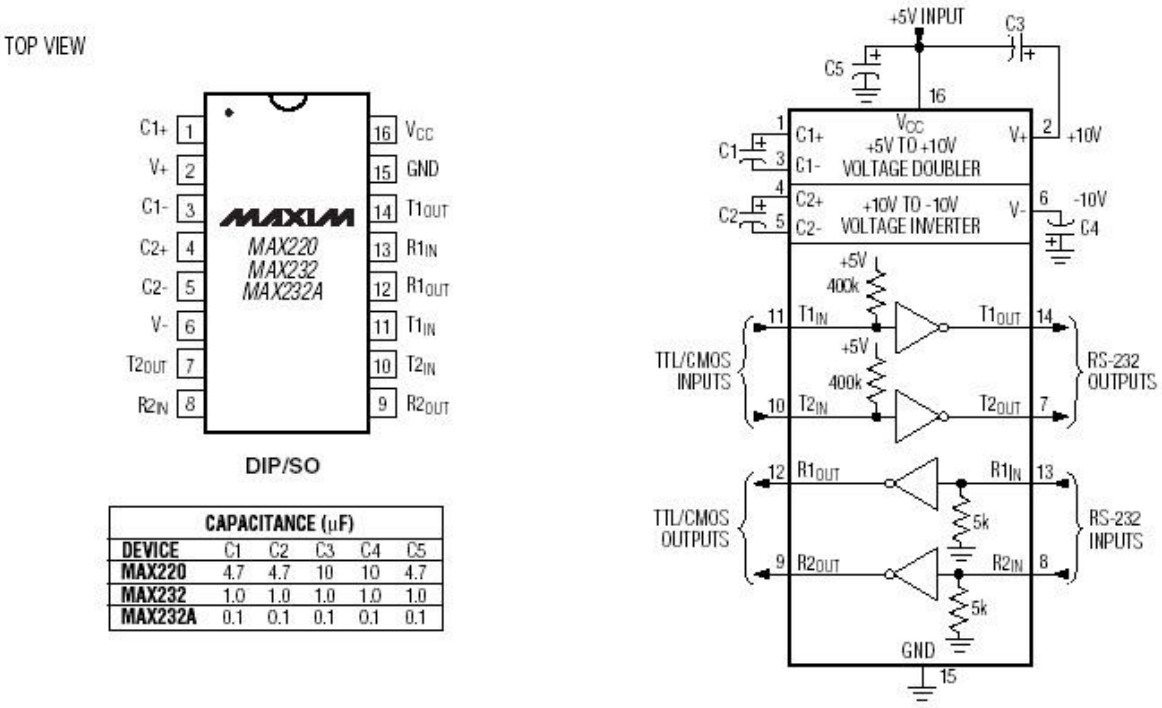
计算机与计算机或计算机与终端之间的数据传送可以采用串行通讯和并行通讯二种方式。由于串行通讯方式具有使用线路少、成本低，特别是在远程传输时，避免了多条线路特性的不一致而被广泛采用。 在串行通讯时，要求通讯双方都采用一个标准接口,使不同 的设备可以方便地连接起来进行通讯。RS-232-C 接口（又称 EIA RS-232-C）是目前最常用的一种串行通讯接口。它是在 1970 年由美国电子工业协会（EIA）联合贝尔系统、 调制解调器厂家及计算机终端生产厂家共同制定的用于串行通讯的标 准。它的全名是“数据终端设备（DTE）和数据通讯设备（DCE）之间 串行二进制数据交换接口技术标准”该标准规定采用一个 25 个脚的 DB25 连接器，对连接器的每个引脚的信号内容加以规定，还对各种信 号的电平加以规定。

①接口的信号内容 实际上 RS-232-C 的 25 条引线中有许多是很少使用的，在计算机通讯中一般只使用 3-9 条引线。RS-232-C 最常用的 9 条引线的信号。

②接口的电气特性 在 RS-232-C 中任何一条信号线的电压均为负逻辑关系。即：逻辑。“1”，-5~-15V；逻辑“0” +5~ +15V 。噪声容限为 2V。即 要求接收器能识别低至+3V 的信号作为逻辑“0”，高到-3V 的信号 作为逻辑“1” 。

③ 接口的物理结构 RS-232-C 接口连接器一般使用型号为 DB-25 的 25 芯插头座,通常插头在 DCE 端,插座在 DTE 端. 一些设备与 PC 机连接的 RS-232-C 接口,因为不使用对方的传送控制信号,只需三条接口线,即“发送数据”、“接收数据”和“信号地”。所以采用 DB-9 的 9 芯插头座，传输线采用屏蔽双绞线。

④传输电缆长度 由 RS-232C 标准规定在码元畸变小于 4%的情况下，传输电缆长度应为 50 英尺，其实这个 4%的码元畸变是很保守的，在实际应用中，约有 99%的用户是按码元畸变 10~20%的范围工作的，所以实际使用中最大距离会远超过 50 英尺。



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

图 2.3.1 Max232 结构图

（4）继电器

继电器是具有隔离功能的自动开关，广泛用于遥控，遥测，通信，自动控制，机电一体化及电力电子设备中，是最重要的控制元件之一。

继电器是在自动控制电路中起控制与隔离作用的执行部件，它实际上是一种可以用低电压、小电流来控制大电流、高电压的自动开关。在本系统中，继电器控制的自动温度调节电路和 PCI16F877A 单片机中程序构成温度自动监测电路，实现对生物培养液温度的监测和自动控制

（5）半导体降温片及电阻加热丝

①半导体制冷器是根据热电效应技术的特点，采用特殊半导体材料热电堆来制冷，能够将电能直接转换为热能，效率较高。其工作原理如图 2.5.1：

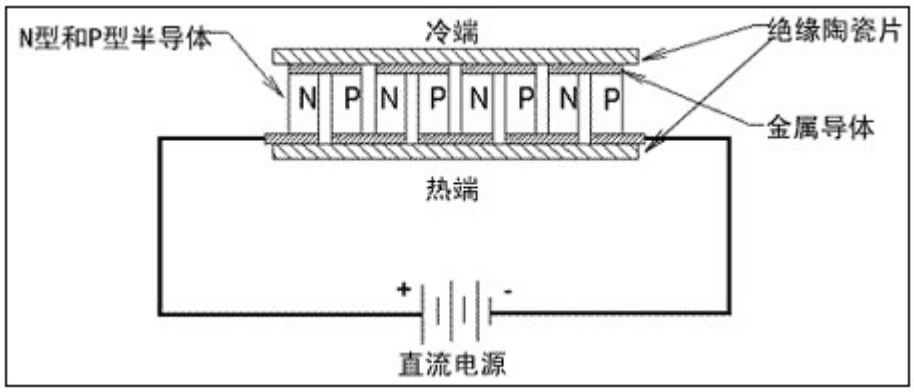
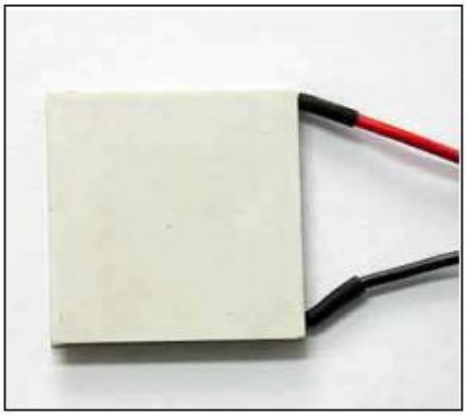


图 2.5.1 半导体降温片工作原理图

半导体制冷片由许多 N 型和 P 型半导体之颗粒互相排列而成，而 N P 之间以一般的导体相连接而成一完整线路，通常是铜、铝或其他金属导体，最後由两片陶瓷片像夹心饼乾一样夹起来，陶瓷片必须绝缘且导热良好，通上电源之後，冷端的热量被移到热端，导致冷端温度降低，热端温度升高。它的外观如图 2.5.2 所示。



正视图

侧视图

图 2.5.2 半导体降温片外观图

②本控制系统是对生物培养液进行温度监控，过快的温度变化对生物繁殖显然是不利的，因此在本系统中采用的是高阻抗小功率加热电阻丝进行温度的小范围调节。

3 温度控制系统的组成框图

采用典型的反馈式温度控制系统，组成部分见图 3.1。其中数字控制器的功能由单片机实现。

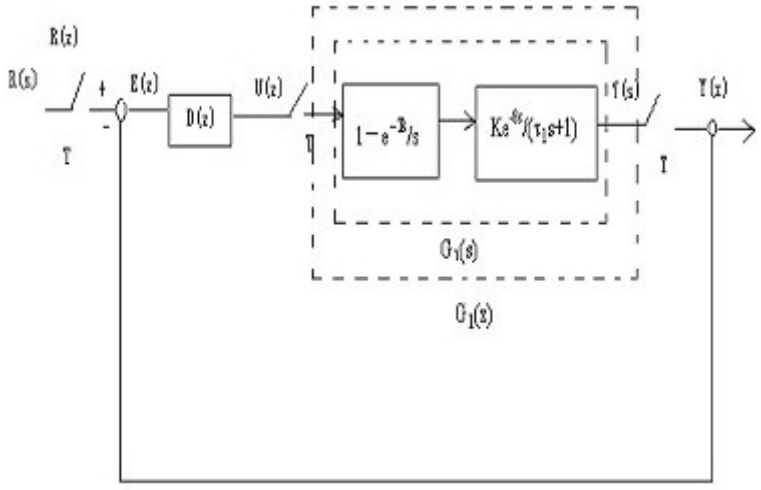


图 3.1 温度控制系统的组成框图

培养皿的传递函数为 $Gt(s)=Ke^{-ts}/(t_1 s+1), q=LT$，其中 τ_1 为电阻加热的时间常数，q 为电阻加热的纯滞后时间，qT 为采样周期。

A/D 转换器可划归为零阶保持器内，所以广义对象的传递函数为

$$G_1(s)=[Ke-qs/(t_{1s}+1)\times[(1-e-Ts)/s] \qquad (3\text{-}1\text{-}1)$$

广义对象的 Z 传递函数为

$$G_1(z)=Z\{[Ke-qs/t1s+1)]\times[(1-e^{-Ts})/s]\}=Kz^{-L-1}(1-e^{-Tt}1)/1-eT/t_1z^{-1}) \qquad (3\text{-}1\text{-}2)$$

所以系统的闭环 Z 传递函数为

$$\Phi(z)=Z[(1-e^{-Ts})/s\times e^{-qs}/(ts+1)=z^{-L-1}(1-e^{-T/t})/(1-e^{-T/t}) \qquad (3\text{-}1\text{-}3)$$

系统的数字控制器为

$$U(z)/E(z)=D(z)=\Phi(z)/G_1(z)$$

$$=(1-e^{-T/t1}z^{-1})(1-e^{-T/t})/K(1-e^{-T/e1})[1-e^{-T/t}z^{-1}]-(1-e^{-T/t})z^{-1-L} \qquad (3\text{-}1\text{-}4)$$

写成差分方程即为

$$u(k)=e^{-T/t}u(k-1)+(1-e^{-T/t})u(k-1-L)$$

$$+(1-e^{-T/t})e(k)/K(1-e^{-T/t1})-(1-e^{-T/t})e^{-T/t1}e(k-1)/K(1-e^{-T/t1}) \qquad (3\text{-}1\text{-}5)$$

令 $a_0=(1-e^{-T/t})/K(1-e^{-T/t1})$

$a_1=(1-e^{-T/t})e^{-T/t1}/K(1-e^{-T/t1})$

$b_1=e^{-T/t}$，

$b_2 = 1 - e^{-T/t}$ ，

得 $u(k) = a_0 e(k) - a_1 e(k-1) + b_1 u(k-1) + b_2 u(k-1-L)$ （3-1-6）

式中 $e(k)$ ——第 k 次采样时的偏差；

$e(k-1)$ ——第 $k-1$ 次采样时的偏差；

$u(k-1)$ ——第 $k-1$ 次采样时的偏差；

4 温度控制系统结构图及总述

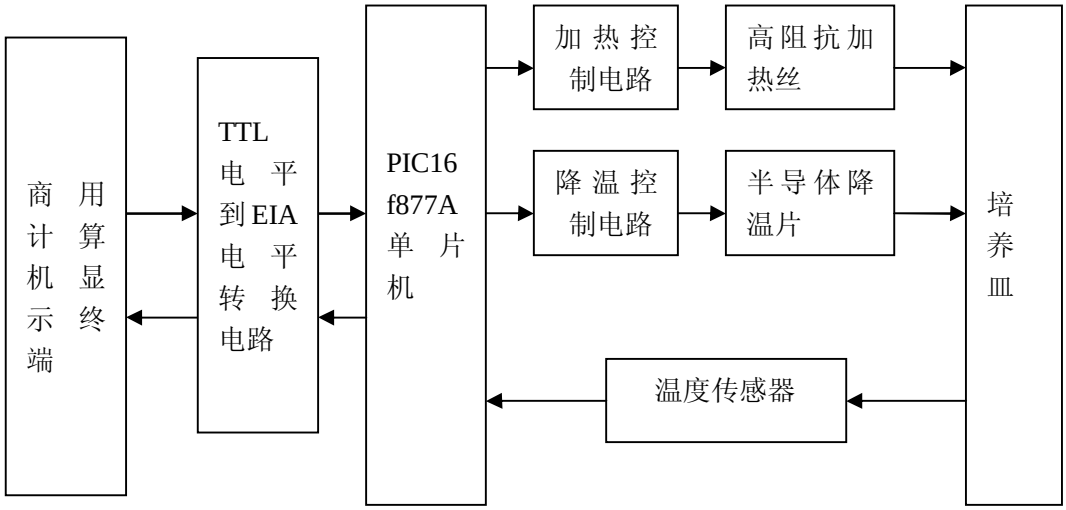


图 4.1 温度控制系统结构图

图 4.1 中温度传感器和 Micro Chip PIC16F877A 单片机中的 A/D 转换器构成输入通道，用于采集培养皿内的温度信号。温度传感器输出电压经过 A/D 转换后的数字量与培养皿内的温度给定值数字化后进行比较，即可得到实际温度和给定温度的偏差。培养皿内的温度设定值由 Micro Chip PIC16F877A 单片机中程序设定。由 Micro Chip PIC16F877A 单片机构成的数字控制器进行比较运算，经过比较后输出控制量控制由加热和降温电路构成的温度调节电路对培养皿中的培养液温度进行调节。同时通过电平转换电路把当前温度传输到商用计算机的串口中，由计算机动态的显示培养皿中的温度，正常情况下温度控制由 Micro Chip PIC16F877A 单片机自动控制。必要时，计算机也可以通过软件来强制改变培养皿中温度。

5 温度控制系统软件设计

5.1 Microchip PIC16F877A 单片机**温度控制系统软件结构图如图** 5.1.1 **所示。**

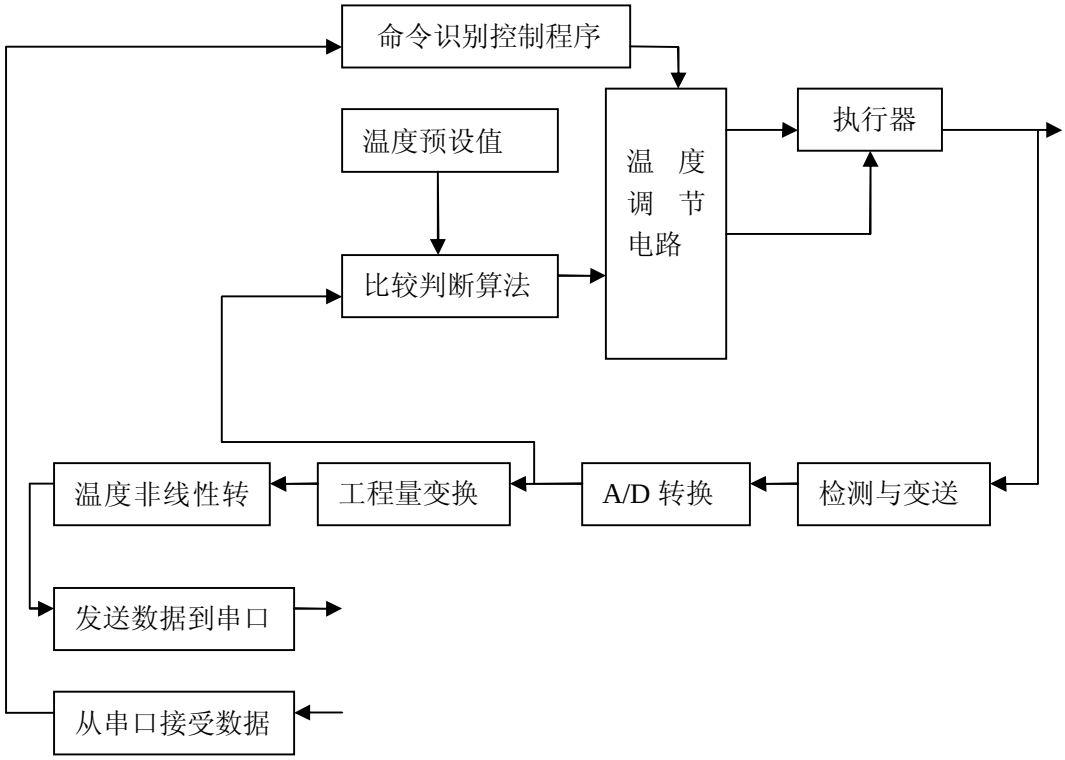


图 5.1.1 单片机温度控制系统软件结构图

5.2 单片机控制流程图

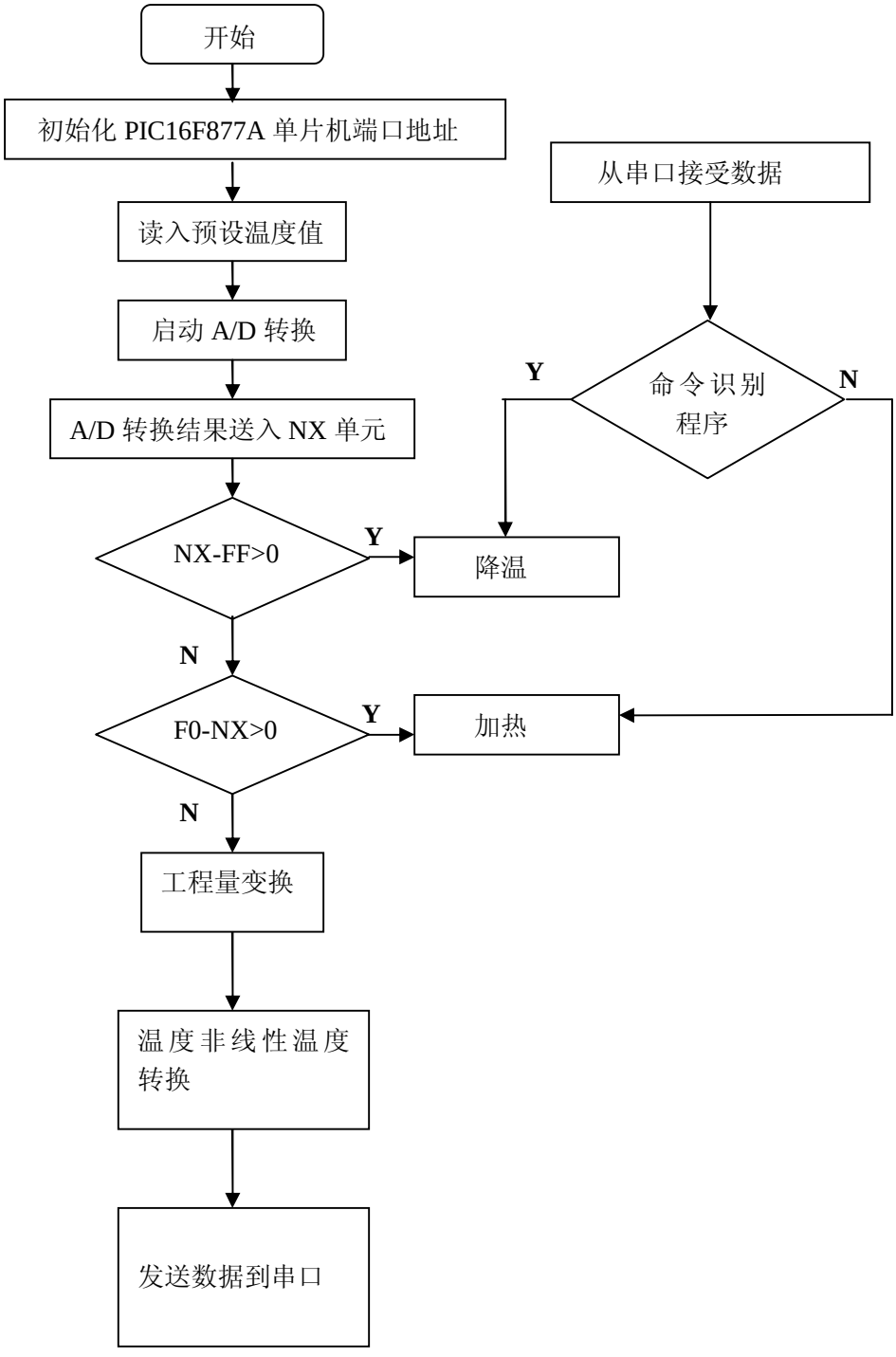


图 5.2.1 单片机控制流程图

5.3 温度变换程序模块

温度传感器在 12℃到 60℃输出 2.52V—1.02V，温度起点为 12℃，满量程为 48℃。Micro Chip PIC16F877A 单片机内嵌的 10 位 A/D 转换器对应输出的数字量为

0000000000B~1111111111B（0~5V），应用以下变换公式进行变换：

$$A_X=A_0+(A_M-A_0)(N_X-N_0)/(N_M-N_0)$$

式中，A_0 为一次测量仪表的下限。

A_M 为一次测量仪表的上限。

A_X 实际测量值。

N_0 仪表下限对应的数字量。

N_M 仪表上限对应的数字量。

N_X 测量值对应的数字量。

5.4 温度非线性转换程序模块

采用折线拟合法进行线性化处理

如图 5.4.1 所示，分为以下几段：

当 1.73V≤Ax<2.52V 时，T℃=0.06*WN+12

当 1.40V≤WN<1.73V 时，T℃=0.03*WN+25

当 1.24V≤WN<1.40V 时，T℃=0.016*WN+40

当 1.06V≤WN<1.24V 时，T℃=0.018WN+50

表 5.4.1 温度曲线实际测量数据

温度(℃)	12	13	14	15	16	17	18
电压（V）	2.52	2.48	2.47	2.44	2.40	2.39	2.37
温度(℃)	19	20	21	22	23	24	25
电压（V）	2.32	2.28	2.22	2.15	2.09	1.83	1.73
温度(℃)	26	27	28	29	30	31	32
电压（V）	1.70	1.66	1.64	1.61	1.58	1.56	1.54
温度(℃)	33	34	35	36	37	38	39
电压（V）	1.53	1.50	1.48	1.46	1.45	1.43	1.41
温度(℃)	40	41	42	43	44	45	46
电压（V）	1.40	1.38	1.37	1.35	1.32	1.30	1.29
温度(℃)	47	48	49	50	51	52	53
电压（V）	1.27	1.26	1.25	1.24	1.22	1.20	1.19
温度(℃)	54	55	56	57	58	59	60
电压（V）	1.17	1.16	1.12	1.11	1.09	1.07	1.06

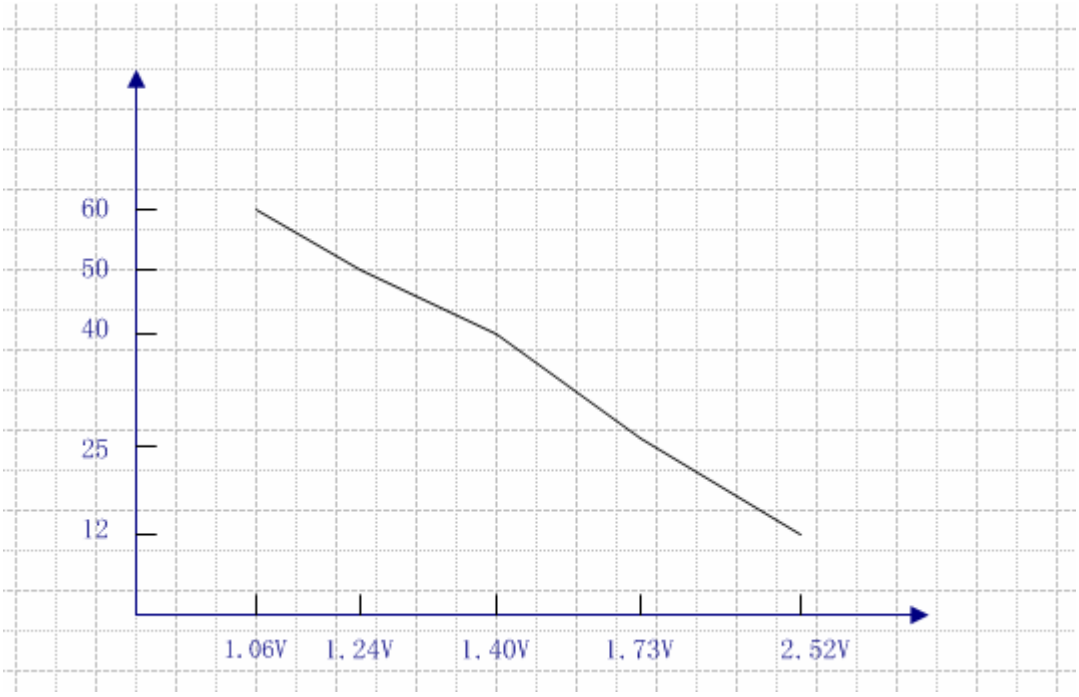


图 5.4.1 温度分段线限等效图

6 通信协议的设计

由于温度采集和实施控制是通过单片机控制系统实现，而微机完成温度监控，所以需要采用单片机和微机之间的通信协议。本设计应用条件为传输距离不超过15米的短距离数据传输，且传输数据量较小，所以采用在控制领域里应用较广泛RS232C串行通信方式。

针对近程小批量的数据通信，设计时采用3 线制（RXD ，TXD ，GND）软握手的零MODEM方式。即：将PC机和单片机的“发送数据线（TXD）”与“接收数据（RXD）”交叉连接，二者的地线（GND）直接相连而其它信号线如握手信号线均不用，而采用软件握手。这样即可以实现预定的任务，又可以简化电路设计节约了成本。

由于RS232C是早期为促进公用电话网络进行数据通信而制定的标准，其逻辑电平与TTL， MOS 逻辑电平不同。逻辑0 电平规定为+5~+15V之间，逻辑1是电平为-5 ~-15V 之间。因此在将PC机和单片机的RXD和TXD交叉连接时，必须进行电平转换。

下图即为通信时的硬件连接图，其中器件MAX232完成逻辑电平转换的任务。

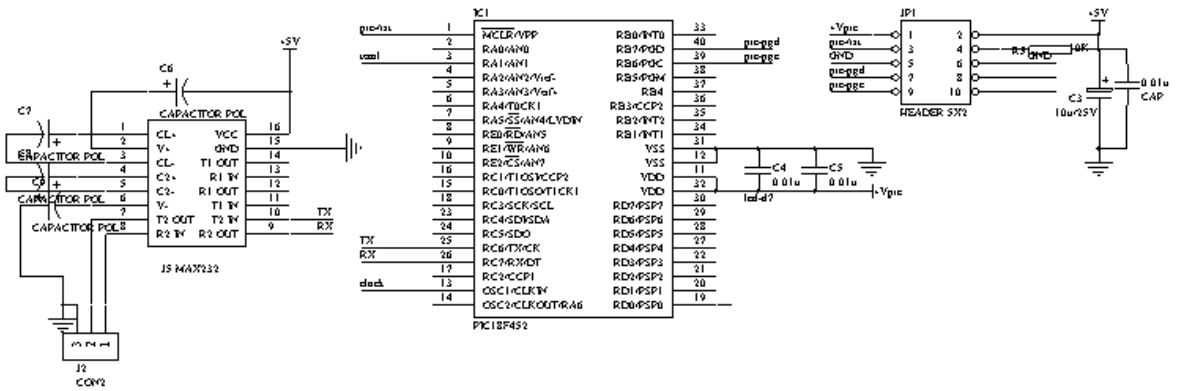


图 6.1 电平转换电路图

注：在PC机中9针RS232接口中：2线：RXD， 3线：TXD， 5线：GND
而在25针的RS232接口中：3线：RXD， 2线：TXD， 7线：GND

6.1 软件设计

在进行数据通信的软件设计时，必须解决好两个方面的问题：一是可靠性，二是速度。而这两方面的问题，可靠性是第一位的,速度只能是在可靠的基础上的速度。可靠快速转输的实现，需要PC-单片机软件以及通信协议等各个环节的可靠和其间的相互配合。

6.1.1 通信协议概述

在设计PC-单片机通信协议时，需说明一点：在本系统的实际通信中，PC机是主控者单片机只是被动接收者。采用这种通信协议较双方互为主控者时简单。

本通信协议的设计思想是基于帧传输方式。即在向RS232串口发送命令信号，应答信号及数据信号时，是一帧一帧地发送的。为了使数据快速可靠地传输，将每一帧数据唯一对应一命令帧。此时传输数据即执行命令具体如下：

(1) 在PC读数据时，遵循“读命令-等数据-报告”，即PC下达一命令，等待接收数据，根据所接收数据的正误向应用程序报告此命令的执行情况。

(2) 在PC写数据时，遵循“写命令-等回应-报告”，即PC下达一写命令（此时所要写的数据含于此命令中），等待单片机发来的“已正确接收”的回应信号，并向应用程序报告此命令执行完毕。

(3) 如果在转输过程中，其间PC或MCU所接收任何一帧信号出现错误时，均会向对方发送重发此帧信号的请求。如果连续三次转输失败，则退出通信并向应用程序报告。

6.2 通信协议说明

6.2.1信号帧分类

(1) 读命令帧：当PC读数据时，PC向PIC16F877A发送的命令信号。

(2) 写命令帧：当PC写数据时，PC向PIC16F877A发送的命令信号(内含所要写的数据)。

(3) 数据帧：当PC读数据时，PIC16F877A向PC发送的内含数据信息的信号。

(4) 正回应帧：当PC写数据时，PIC16F877A向PC报告数据已正确接收的信号。

(5) 重发命令帧：当PC读/写数据时，PIC16F877A所接收的信号帧(读/写命令帧)有误时向PC发出的请求重发信号。

(6) 放弃命令帧：当PC读/写数据时出现了使程序无法正常执行时PC或PIC16F877A向对方发出的退出通信的通知信号。

6.2.2信号帧格式

(1) 读命令帧格式

帧头标志	帧类型	器件地址	起始地址
长度	校验和	帧尾标志	

帧头标志(1 Bit)：表示此数据包属于本串口通信协议,并为是否接收此包数据的标志。

帧类型(1 Bit)：所用信号帧的识别标志,即1.2.1 信号帧分类中的各类型信号的标志字节。

器件地址(1Byte)：PC所要访问的外部器件的地址即是哪一个外部器件。

起始地址(2Byte)：PC所要访问的器件的存贮器起始地址。

长度(1Byte)：一次命令所转输的数据长度。

校验和(1Byte)：此帧信号的校验字节,为异或校验。

帧尾标志(1Byte)：此帧信号的结束标志。

(2) 写命令帧

帧头标志	帧类型	器件地址	起始地址

长度	数据区	校验和	帧尾标志

数据区:所要写的数据信息。其它分析同上。

(3) 数据帧

帧头标志	帧类型	长度	数据区	校验字	帧尾标志

长度:所转输数据的长度。

数据区:所转输的数据信息。其它分析同上。

(4) 正响应帧

帧头标志	帧类型	空	校验字	帧尾标志

空无意义:为了PIC16F877A编程的方便而加入。其它分析同上。

(5) 重发帧

帧头标志	帧类型	空	校验字	帧尾标志

其它分析同上。

(6) 放弃帧

帧头标志	帧类型	错误码	校验字	帧尾标志

错误码:

00H 执行PC命令发放弃帧回应被动退出通讯。

01H PIC16F877A 单片机方写入芯片发生错误主动通知PC退出通讯。

6.2.3 通信协议处理流程

(1) 数据分帧与数据重组

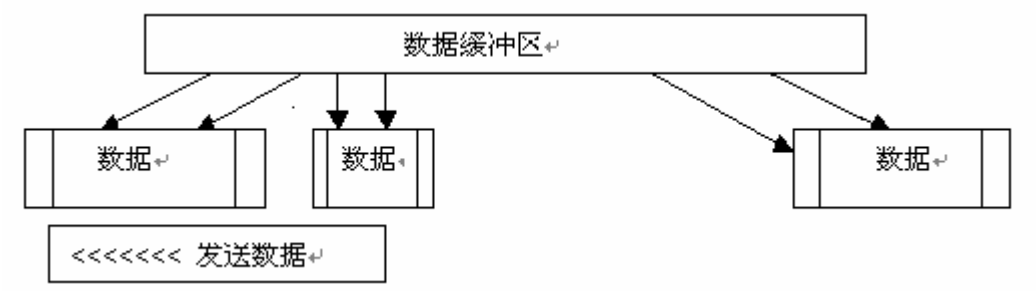


图 6.2.1 串口数据发送过程

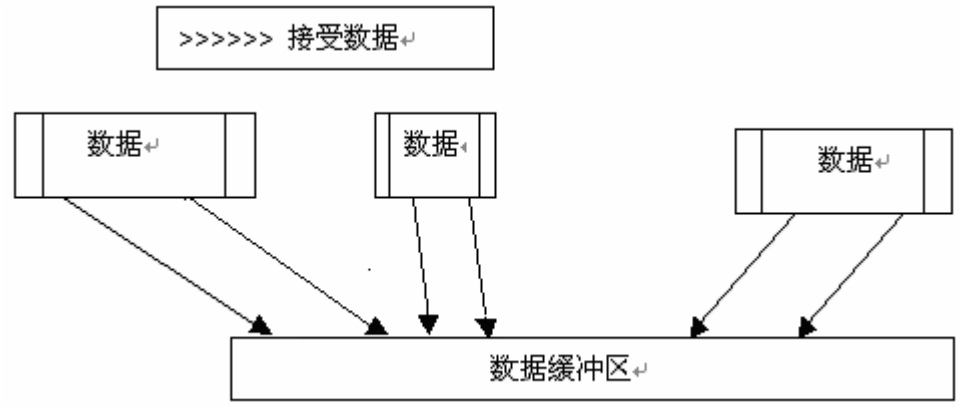


图 6.2.2 串口数据接受过程

将应用程序发送过来的数据作为一个数据流放在发送缓冲区中，通过通信协议进行

分帧——切割——发送。在接收端，分帧的数据去掉帧头重新组合到接收缓冲区中，交给应用程序处理，发送过程的示意如图6.2.1，接收过程的示意图如图6.2.2。

单片机串口通信软件设计流程图

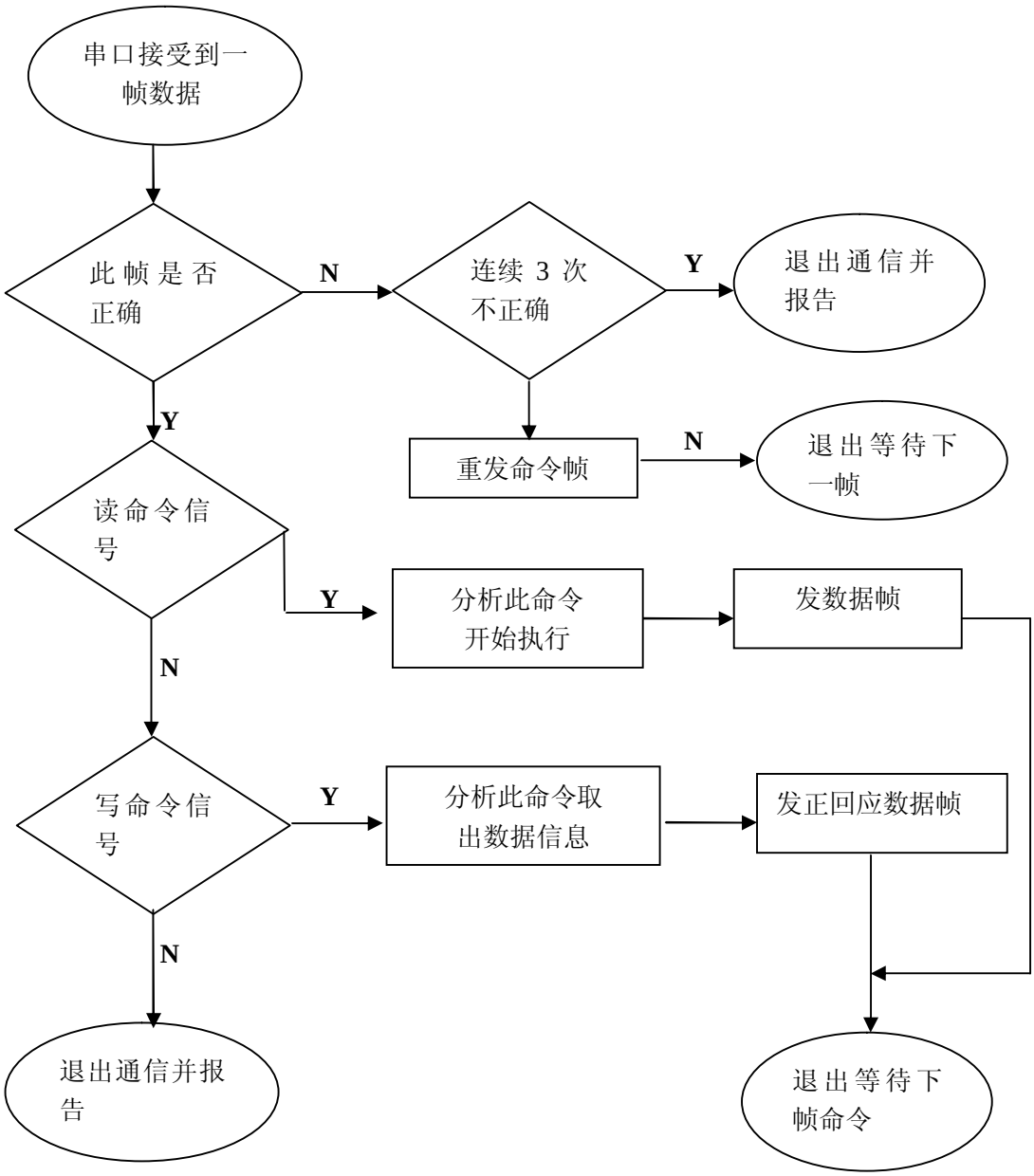


图 6.2.3 单片机串口通信软件流程图

PC 接收数据软件设计流程

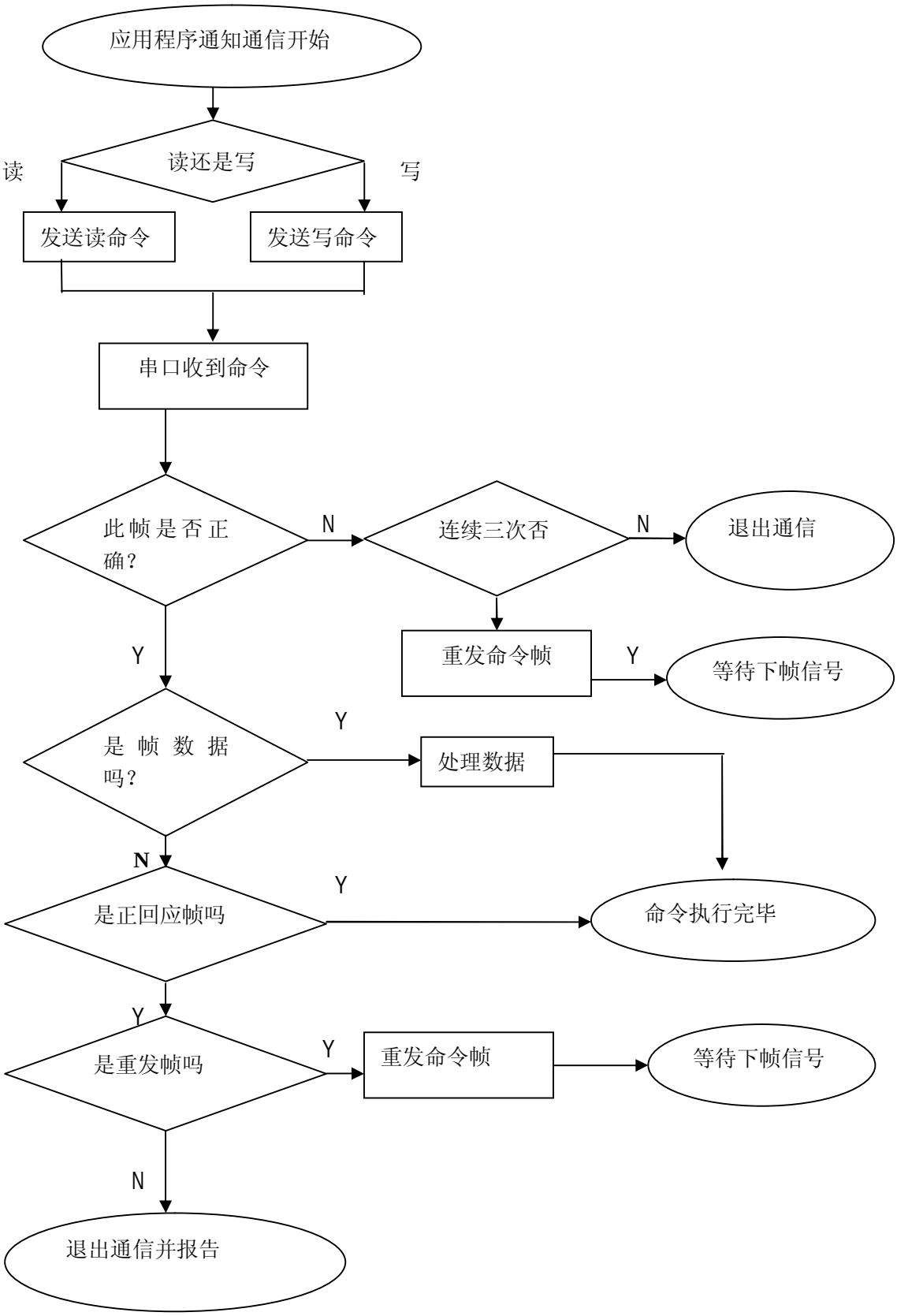


图6.2.4 PC串口通信软件设计流程图

6.3 PC 上位机的软件设计

6.3.1 PC软件设计方法的选择

在开发PC上位机的通信程序中,人们常用的编程语言可分为3类:(1) 直接面向底层硬件的汇编语言。(2) DOS环境下的高级编程语言,如: C语言等。(3) Windows环境下的高级编程语言，如：VC++等。而在这3种方式中Windows环境下的串口编程以其设备无关性，可移植性以及界面友好等特征而得到广泛应用。同时在Windows操作系统已经占据统治地位的情况下，欲开发良好的通信程序，利用Windows环境下的高级语言已渐成为必然的选择。

开发Windows环境下的串口通信程序主要有以下2种方法：

(1) 利用Windows API（Application Program Interface）用户程序接口函数；

(2) 利用ActiveX控件；

后者的主要特点是简单易学，但前者的功能更为强大控制手段更为灵活。

6.3.2 PC软件通信方式的选择

在Win32环境下串行通信有两种：主要方式即同步方式，异步方式两种方式有各自的特点。在软件设计时应根据实际情况选择合适的方式。

（1）同步方式

在同步方式中,读串口的函数试图在串口的接收缓冲区中读取规定数目的数据,直到规定数目的数据全部被读出或设定的超时时间已到时才返回。例如：（以C++ Builder编程语言为例下同）

```
……
……
COMMTIMEOUTS cto;
int timeConstant, timeMutiplier;
cto.ReadTotalTimeoutConstant = timeConstant; //设置总超时常数
cto.ReadTotalTimeoutMultiplier = timeMutiplier; //设置总超时系数
SetCommTimeouts(m_hFile,&cto); //超时设置
……
……
ReadFile (hComport,inBuffer,nWantRead,&nRealRead,NULL);//读串口
……
……
```

COMMTIMEOUTS结构用于设置超时,指定读写函数的等待时间

在ReadFile 函数中hComport 为待读串口句柄;inBuffer 为输入缓冲区大小;nWantRead 为每次调用ReadFile 时,函数试图读出的字节数;nRealRead 为实际读

出的字节数;最后一个参数值NULL 代表ReadFile将采用同步文件读写方式。

（2）异步方式

异步方式中,利用Win32 的多线程结构,可以让串口的读写操作在后台进行,而应用程序的其它部分在前台执行例如:

```
……
……
……
CreateFile(lpszPort, //打开串口
GENERIC_READ|GENERIC_WRITE,
0,
0,
……
……
OPEN_EXISTING,
FILE_FLAG_OVERLAPPED, //允许异步操作
0);
OVERLAPPED lpOverlapped;
COMMTIMEOUTS cto;
int timeConstant, timeMutiplier;
cto.ReadTotalTimeoutConstant = timeConstant; //设置总超时常数
cto.ReadTotalTimeoutMultiplier = timeMutiplier; //设置总超时系数
SetCommTimeouts(m_hFile,&cto); //超时设置
lpOverlapped.hEvent=CreateEvent (NULL.TRUE,FALSE,NULL);
……
……
……
ReadFile (hComport,inBuffer,nWantRead,&nRealRead,&lp Overlapped); //读串口
……
```

lpOverlapped 是1个OVERLAPPED 结构变量,OVERLAPPED 结构用于指出读写操作与其它操作的重叠为了实现线程间同步与通信,上面的代码中用CreateEvent 函数产生1 个人工复位事件,并将其句柄赋予lpOverlapped的hEvent成员这样,在异步读写完成时,Windows95发送该事件信号。

（3）两种方式的比较

异步方式利用多线程结构来监视通信设备，其最大优点是程序对接收数据具有自主觉察能力。一旦通信线程查询到数据已发送到串口上，线程自动向应用程序发送一个数

据接收到的消息，应用程序可用该消息来读取通信设备传来的数据。并且使用通信线程还不占用CPU时间，这样系统实际上具有了同时控制多个通信设备（如MODEM）的能力。因此在对系统强壮性要求较高的场合下应采用异步方式。

异步方式的优点也恰是同步方式的缺点。使用同步方式时容易发生线程阻塞，从而使系统性能下降。但在某些场合下，该缺点可以通过一些措施尽可能地减小，而其简单易用的优点却是很好地体现出来。如果不考虑Win95的进程和线程的问题，仅在串口有数据时，去读串口缓冲区就可以了。此时确定串口读取的时机，握手协议及软件纠错的实现是程序员应考虑的主要问题，也是减小线程阻塞所带来的负面影响的主要措施。

可以采用同步转输方式的场合有如下一些特点：

① 何时转输数据由PC机来决定，下位机只是被动接收并执行命令。

② 有限时间内，PC机命令可以执行完毕并返回结果。而不会使PC机处于长时间等待。

③ 每次所转输的数据的长度是已知的，所转输的数据量是有限且比较小。

我们在开发串行通信程序时，分别应用这两种方式开发都获得了成功。鉴于应用异步方式的安全性和普遍性

6.3.3具体实现方法

下面以C++ Builder为例，叙述PC机通信软件的实现过程：

(1)打开串口

在Win32中，串口和其他通信设备是作为文件处理的。串口的打开并闭读取以及写入所用的函数与操作文件的函数相同。

通信会话由调用CreateFile 函数打开串口开始，CreateFile 以读访问权限，写访问权限或读写访问权限“打开串口”并设定了对其是异步操作方式。还是同步操作方式调用该函数打开串口进行读写操作的例子如下：

```
mHandle = CreateFile(lpszPort, //串口名
GENERIC_READ|GENERIC_WRITE, //允许读/写
0, //独占方式串口不能共享
NULL, //安全性属性一般设为0
OPEN_EXISTING, //串口是已存在的不能建新端口
lpOverlapped, //异步方式
0 //串口无模板文件应设为0
);
```

如果调用成功函数返回串口的句柄赋给Handle，如果调用失败则函数返回INVALID_HANDLE_VALUE。

(2)初始化串口

对串口的初始化工作包括对波特率，数据位，停止位，奇偶校验位I/O 缓冲大小以及超时等参数的设置。在调用API 函数进行串口初始化时，波特率，数据位，奇偶校验停止位的信息包含于一个DCB结构中，而超时方面的信息则包含于COMMTIMEOUTS结构中，一般在用CreateFile 打开串行口后，可以调用GetCommState 函数来获取串行口的初始配置。要修改串行口的配置应该先修改DCB结构，然后再调用SetCommState函数用指定的DCB结构来设置串行口。例如：

```
DCB dcb;
GetCommState(mHandle, &dcb) //读取DCB结构
……
……
dcb.BaudRate=9600 // 设置波特率为9600b/s
dcb.ByteSize=8; // 每个字符有8位
dcb.Parity=NOPARITY; // 无校验
dcb.StopBits=ONESTOPBIT; // 一个停止位
SetCommState(hCom, &dcb) // 保存至DCB结构使设置值生效
```

调用SetupComm 函数可以设置串行口的输入和输出缓冲区的大小。如果通信的速率较高则应该设置较大的缓冲区。例如：

```
……
……
SetupComm( mHandle , 1024*2, 1024*2 ) //输入输出缓冲区的大小均为2K
……
……
```

在用ReadFile 和WriteFile 读写串行口时，需要考虑超时问题。如果在指定的时间内没有读出或写入指定数量的字符，那么ReadFile 或WriteFile 的操作就会结束。要查询当前的超时设置应调用GetCommTimeouts 函数。该函数会填充一个COMMTIMEOUTS 结构调用SetCommTimeouts 可以用某一个COMMTIMEOUTS结构的内容来设置超时。

```
……
……

TimeOuts. ReadIntervalTimeout=0 //读间隔超时
TimeOuts.ReadTotalTimeoutMultiplier=10 //读时间系数
TimeOuts.ReadTotalTimeoutConstant=100 //读时间常量
TimeOuts.WriteTotalTimeoutMultiplier=10 //写时间系数
TimeOuts.WriteTotalTimeoutConstant=100 //写时间常数
SetCommTimeouts(hCom, &TimeOuts); // 保存设置值生效
```

……

……

……

COMMTIMEOUTS结构的成员都以毫秒为单位。总超时的计算公式是：

总超时=时间系数×要求读/写的字符数+ 时间常数

异步方式读写串行口时虽然ReadFile()和WriteFile() 在完成操作以前就可能返回但超时仍然是起作用的。这种情况下，超时规定的是操作的完成时间而不是ReadFile()和WriteFile()的返回时间。

(3)读写串口

初始化工作完成以后便可以根据通信协议合理安排读/写函数ReadFile()和WriteFile()以读写各种握手信息和数据信息等。其中何时读取单片机发送过来的数据信息及应答信息是重要的。此时采取的是事件驱动法，即：设置通信资源上的事件掩码为EV_RXCHAR 。当接收到一个字符并放入缓冲区后即通知应用程序例。

```
//PC发送一组命令至单片机
WriteFile(mHandle, //串口句柄
pDataBuff, //存放数据缓种区
iLen, //所写数据的长度
pdwWritten, //已写长度操作前应置为0
lpOverlapped) //异步方式
//设置通信事件掩码
DWORD dwMask=EV_RXCHAR;
SetCommMask(m_hFile,dwMask)) //设置通信事件掩码
//等待通信事件的发生
OVERLAPPED os ;
memset( &os, 0, sizeof( OVERLAPPED ) ) ;
os.hEvent=CreateEvent(NULL TRUE FALSE NULL)
if(!WaitCommEvent(m_hFile, &dwEvtMask, &os)) // 重叠操作
if(GetLastError()==ERROR_IO_PENDING)
{
// 无限等待重叠操作结果
GetOverlappedResult(mHandle, &os, &dwTrans, true);
//事件已发生安排读操作
ReadFile(mHandle, //串口句柄
pDataBuff, //存放数据缓种区
iLen, //所读数据的长度
```

pdwRead, //实际所读长度
lpOverlapped) //异步方式
}

在上例中，我们无限等待通信事件的发生。如果通信事件一直没有发生则系统将不会继续执行。在实际程序设计中我们可以设置一时限，超过此时限通信事件未到则执行相应错误处理此时，只需将GetOverlappedResult函数替换为WaitForSingleObject函数此函数的声明形式如下：

WaitForSingleObject(
HANDLE hEvent, //事件句柄
unsigned long mTimeOuts //超时设置
)

(4) 关闭串口

通信完毕调用CloseHandle() 函数关闭串口例如

CloseHandle(mHandle); //关闭mHandle为打开串口时返回的句柄

6.4 单片机软件设计

我们知道影响数据转输产生错误的因素有：转输线分布参数上下位机间的波特率误差现场干扰等。而针对近程小批量数据的通信，下位机的波特率误差性是影响可靠通信的最主要因素。所以在单片机软件的设计时应重点考虑并设置好波特率。

6.4.1波特率

（1）波特率误差来源分析

①单片机的振荡电路是由晶体及电容C1 和C2 构成。晶振频率主要由晶体的因有频率决定，同时也与电容C1、C2及外界温度有一定的关系。另外，晶体频率的标称值与实际值也不可能完全一致。

②波特率最大允许误差分析

在异步串行通信方式1中单片机以16倍波特率的采样速率对接收数据（RXD）不断采样，一旦检测到由1到0的负跳变，16分频计数器立刻复位，使之满度翻转的时刻恰好与输入位的边沿对准。16分频计数器把每个接收位的时间分为16 份，在中间三位即7 ，8 ，9 ，状态时位检测器对RXD 端的值采样，并以3取2的表决方式确定所接收的数据位。由此可见，当波特率的误差使得在接收某位数据位时，采样点离该位的中点半位间隔时将会对该位采样两次。即：欲使接收的第N位为正确位时，须满足下式成立：

所允许的波特率误差N > 0.54

故当所传输的一帧数据为10 位时，所允许的最大的波特率允许误差为5 %对于其它常用的8位，9位，11位，一帧的串行传输，其最大的波特率允许误差分别为6.25%，5.56%，和4.5%。

③减小波特率误差的措施

我们知道使用离散度小的晶振是减小波特率误差的关键。如果，晶振的离散度已超过所允许的范围，此时不宜用其标称值，可以采用测量其波特率的方法来得出实际的晶振波特率值。

（2）单片机软件的实现

①设置通信方式和波特率的值例

……

……

……

```
MOV SCON,#50H 初始化串口设为方式1
MOV TMOD,#20H 利用定时器1为波特率发生器并设为模式2
MOV PCON,#XXH 设置SMOD值
MOV TH1,#XXH 设置定时器初始值
SETB TR1 启动定时器1
```

……

……

……

②等待接收PC机发来的信号帧并按通信协议作出相应响应。

6.5 通信协议设计结论

6.5.1通信可靠性分析

通信的可靠性主要体现在所使用通信协议的可靠性上，本通信协议的可靠性主要有两点理论基础：

(1)通过判断帧头起始字符来决定一帧的开始，这样就避免了部分数据进入到内部数据处理之中。这个可能性在1/256， 通过停止位的判断可将这个可能性再降低1/256。另外通过帧类型字节的判断可使之进一步降低。

(2)校验字将整帧信号进行异或校验则使误收的可能很小。如果将此异或校验改为CRC校验则出错的可能性更是微乎其微了。本通信所用协议具有纠错功能，这体现在当PC 发送或接收数据时，当所接收的应答信号出现失误时，将重新发送或接收此帧数据，直至接收到了正确的应答，具体在程序中最多允许连续出错三次，超过后则放弃通信。在实际应用中，应用本通信时传输距离只有几米以内而且环境干扰比较小，从而从外部因素上进一步保证了通信的可靠性。

6.5.2通信速度分析

如果在不考虑错误发生的情况下，PC 机每发送一帧数据时需要附加12 个字节，其中8 个字节用于发送4 个字节用于应答PC 机。每接收一帧数据时，需要附加13 个字节其中5 个字节用于接收8 个字节用于应答。如：按每帧传送32个字节计算的话，其发送

和接收的效率为为忽略PC和PIC16F877A单片机的处理时间计算。发送数据速率、接收数据速率计算公式如下：

发送数据速率：9600*32/44=6981bit/s

接收数据速率：9600*32/45=6826bit/s

这是理论上的速率，实际中还应包含PC和PIC16F877A单片机的处理信号帧，等待信号帧的时间。在本通信协议中，不会出现某信号帧已到达但PC或PIC16F877A单片机还未开始准备接收的现象。在实际应用中，因具体应用环境不同PC和PIC16F877A单片机处理信号帧的时间会有不同，所以具体速率值依具体应用而变化。

7 Protel99 设计原理图

（1）使用 Protel 进行电路板设计的第一步便是设计原理图，原理图决定了整个电路的基本功能，也是接下来生成网络表和设计印刷电路板的基础。

① 在 Protel 99 的初始界面下新建一个设计库，该数据库用来管理项目。

File-New-改文件名－改保存路径－OK

② 进入设计库文件中的文件夹 Document。

③ 在 Document 文件夹中新建原理图文件和印制板文件。

File-New-Schematic Document-Ok-改文件名

File-New-PCB Document-Ok-改文件名

④ 打开原理图文件。

⑤ 添加原理图文件库。

Design-Add/Remove Library- 浏览所需零件库－Add-Ok

⑥ 放置电路所需的各种元件，图件，网络标号等元器件。

Design-Add/Remove Library- 浏览所需零件库－Add-Ok

从零件库中调出元件 Place-part

⑦ 对原图元件进行布局，布线，构成一个完整的原理图。

Place-part

⑧ 编辑和调整。然后进行输出存档。

右键－Properies….Designation-Part-Footrint Save

⑨ 打印或建立报表。

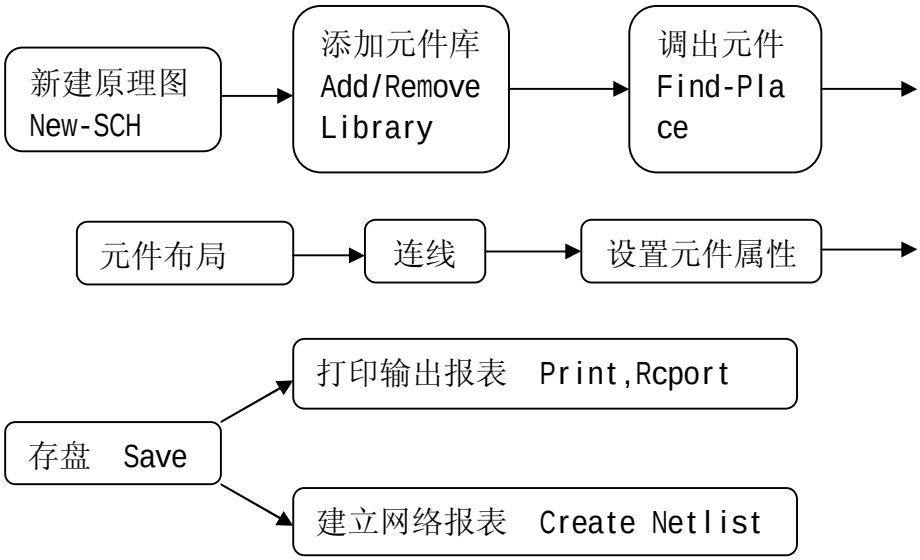


图 7.1.1 protel 设计的流程图

（2）用 PCB 系统设计 PCB 板分以下 7 个步骤：

① 有关参数的设置。这一步主要设定自动布参数、自动布线参数、板面参数等。

② PCB 板尺寸设计。在禁止布线层上，沿设计的 PCB 边画边框线，即指定自动布局的范围。这一步为自动布局打基础。同时，在上层板面（即元器件面）沿禁止布线层的边框图线放置铜线，这是 PCB 板最后成型所必须的。

③ 布局就是根据原理图上元器件之间的连接关系，并考虑电磁兼容性以及元器件的安装空间和散热等，总是将元器件放置在 PCB 电路板上适当的位置。布局的好坏直接影响 PCB 板的电气性能和布局的功能，是 PCB 板设计过程中最费时、最繁琐的。布局工作需要耐心、细致。尽管系统提供了自动布局的功能，但是一般而言都需要手工调整。

手工布局，首先载入 SCH 生成的网络表，通过手工移动元器件 PCB 板上的排列位置实现布局。移动元器件是最好打开网络连接显示，这样就能观察到相邻元器件连线的疏密。

自动布局，PCB 系统环境提供自动布局功能完成元器件放置，但在细节处最好使用手工调整。布局时要求相互间连线多的元器件应该就近放置；相互间可能造成干扰的元器件应远离：功率器件应考虑散热空间。

④ 自动布线。布线就是在元器件引脚之间放置覆铜连线的过程，这一过程可以通过手工完成，也可以自动进行。但是 Protel99 的 PCB 系统提供了强大的自动布线功能，建议使用该功能自动布线。在进行自动布线之前，设计人员必须先设计好布线参数，定义布线规则。如果不适当，可能会导致自动布线失败，即布线的成功率不高，所以这一步要特别注意

⑤ 启动设计规则检查 DRC，这一步利用 PCB 提供的 DRC 功能对完成布线的 PCB 板进行检查，这一步由软件自动完成。检查的结果输出在报告文件*.rep 中，PCB 软件将出错处在 PCB 图上显示出来，为检查、修改提供方便。

⑥ 板面字符调整。为了使设计的 PCB 板美观，并且安装焊接元器件方便，应将元器件的名称。设计值的字符参数移至元器件框外。大小合适且字符不想重叠。

⑦ 将经过 DRC 检查无误，且版面字符调整好的 PCB 设计图存盘、输出、制版。

⑧ 印刷板电路设计完成以后，整个电路板的设计项目就基本完成。存档以便进行后期的修改及完善。

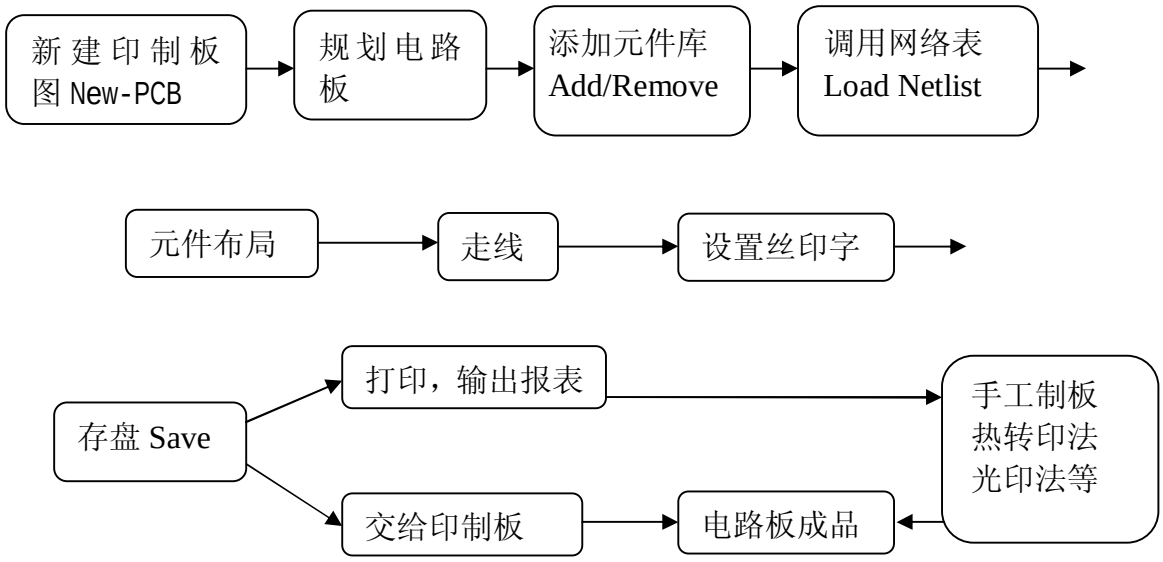


图 7.1.2 制作 PCB 板的流程

8 硬件电路板的制作

本设计中需要有 2 个继电器控制外围温度调节系统，2 个 LED 用来提示串口数据指示，还有一个 PIC16F877A 单片机，一个 Max232 电平转换器，一个有源晶体振荡器及其外围电阻电容等。在确定电路的正确性，可行性之后，开始使用 Protel 对它进行布图。

Protel 是一个很好用的电子制作工具，它还可以进行仿真。在画原理图的过程中，原理图中的元件库中可能找不到自己要找的元件，如 PIC16F877A 等，所以要自己画元件。在画原理图后，选择将元件自动编号，然后根据需要更改部分元件的编号。在定好元件编号后，使用 TOOLS 中的 ERC 进行检查，它会提示是否有编号相同的元件等错误。在 ERC 检查无误后，便可以开始封装了。同样，部分元件的封装在 PCB 库中找不到或者是有出入，如按键开关，2 位拨码开关在 PCB 库中找不到，所以需要自己根据元件的实际大小和相应的原理图中引脚编号，做出正确的封装。

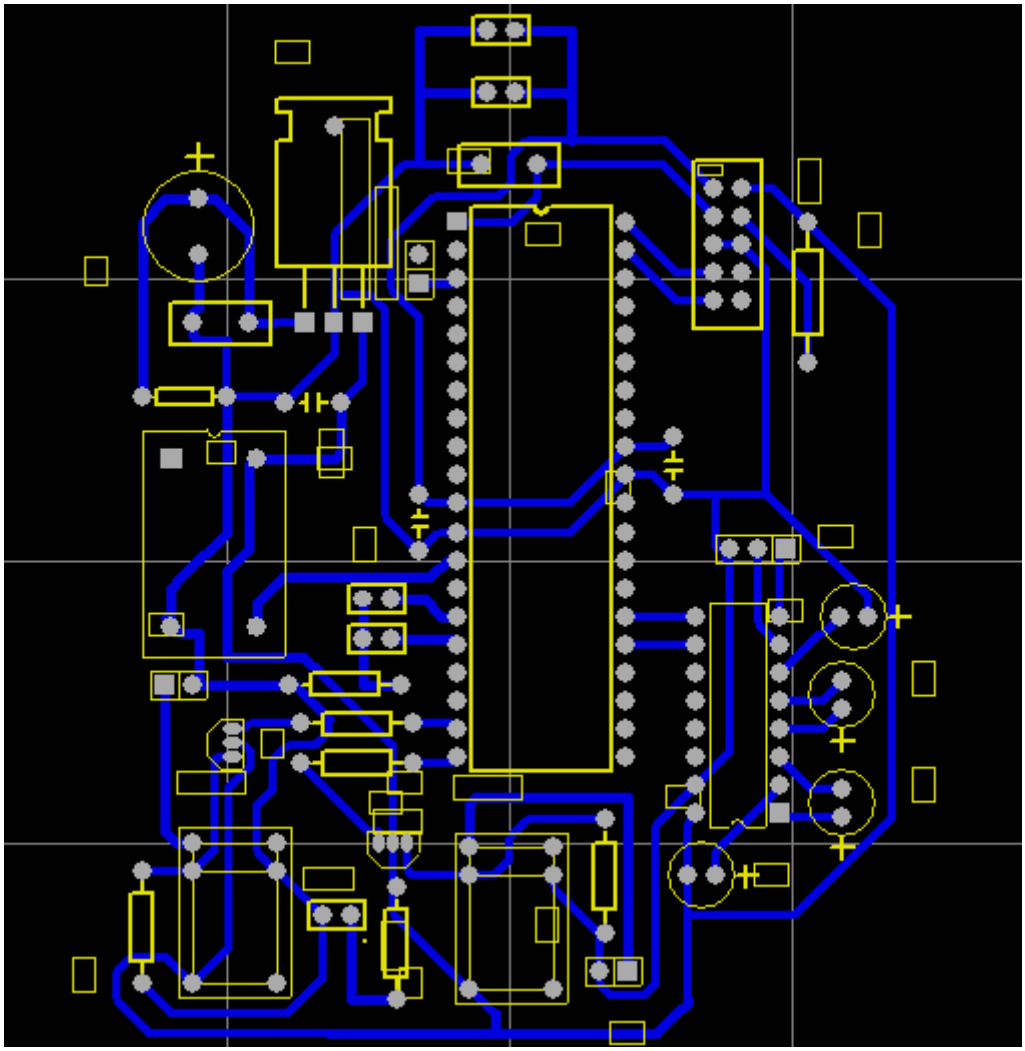


图 8.1 完整的 PCB 图

另外，可变电阻在原理图中的引脚编号和 PCB 库中的引脚编号有点出入（可在原理图中双击元件，选上 HIDDEN PINS，则可以观察到元件的引脚编号），可以在 PCB 库中将

该元件的引脚标号改成与原理图相对应的标号。在封装好全部元件后，可以生成一个元器件报表，在报表中可以清楚的看到各元件的标号和封装代号，在进一步检查完毕后就开始建立网络表。在禁止布线层中画一个边框和电路板大小一样的矩形，然后开始导入网络表，在导入网络表没有错误后，便开始正式布局了。根据原理图的走线，将器件分别拉入框中，放到合适的位置。布局完成后，先设置好安全间距为 10mil，布线层选择底层，线宽选择 25 mil，并将焊盘外径改为 40mil，内径改为 20mil（部分点要根据需要改小或改大）。然后就开始正式布线。布线不能单靠自动布线，特别是在本设计中有众多芯片，所以采用根据原理图对整个电路进行手动布线。这样可以使得整个电路看起来整齐些，在碰到有时线路布不通的时候，采用顶层短跳线的方式进行调整，从而完成整个 PCB 电路板的设计。见图 8.1。

布好 PCB 图，检查无误后，就将 PCB 图打印到转印纸上，然后熨到电路板上，腐蚀，打孔。熨板前，应把铜板用砂纸去掉表面被氧化的部分。腐蚀时，用三氯化铁加适量的开水配成三氯化铁溶液进行腐蚀，这样腐蚀会比较快，腐蚀完后用天那水把电路板清洗，接着便开始打孔（选用 0.8mm 的针头），打完孔后，用万用表测量电路线路是否连通，然后先涂上松香溶液（酒精+松香），这样焊接速度会比较快，还能防氧化，然后将其放在一边晾干。同时，测量部分器件（电阻等）是否有损坏，等电路板晾干后，就要把器件按 PCB 图来安装好。然后就可以开始焊接了。焊接时要防止虚焊和未连接上，所以在焊好后，再用万用表测量元件和线路是否连接好。检测完毕后，硬件电路板装配便完成了。

9 设计总结

通过本次温度监控系统的设计,我大有收获，在制作过程中，一定要注意的每个工作步骤的检查，确保制作成功。比如在合理布线，检查装配无误的情况下，如果还出现电路无输出的情况，那么可以肯定是原理图错误，这时就要回到原理图进行检查。总体的检查顺序应该是原理图、PCB 图、装配情况、焊接工艺。从整体来说这是一个复杂的过程，要细心谨慎，沉着冷静，反复检查，直到找到原因为止。

这次毕业设计历时至少 3 个月，从一开始的确定课题，到后来的资料查找、理论学习，再有就是近来的调试和测试过程，这一切都使我的理论知识和动手能力进一步得到频率合成电路课题中包含了通信电路和单片机部分知识，可以说是对通信电路知识的一次全面综合。在画原理图、PCB 布线、安装和调试过程中不可避免地遇到各种问题，这要求保持沉着冷静，联系书本理论知识积极地思考，实在解决不了可以请教同学或指导老师。虽然在制作过程中不可避免地遇到很多问题，但是最后还是在老师以及同学的帮助下圆满解决了这些问题，实现了整个系统设计与最后调试，相关指标达到期望的要求，很好地完成了本次设计任务。

经过四年学习的积累，在已经掌握相关专业方面知识及其它各方面知识的情况下，我认真严肃的完成了我的毕业设计。

从得到题目到查找资料，从对题目的研究设定到 PCB 电路板的制作，从电路板的调试到失败后再一次全部重新开始……在这一个充满挑战伴随挫折，充满热情伴随打击的过程中，我感触颇深，它已不仅是一个对我四年学习知识情况和我的应用动手能力的检验，而且还是对我的钻研精神，面对困难的心态，做事的毅力和耐心的考验。我在这个过程中深刻的感受到了做毕业设计的意义所在，和我一样真正投入了身心去做的人也一定会有同样的感触。

本课题的重点、难点是：

（1） 初步接触温度传感器，要对传感器的原理、结构、应用等各方面从头开始琢磨；

（2） 考虑从非电量信号到电量信号的电路实现原理以及与单片机的接口；

（3） 熟悉拉 RS-232-C 串口编程的技术；

（4） 考究调整电路的实现过程以及怎么样通过单片机来间接的控制。

通过做本课题，我了解并掌握了传感器的基本理论知识，更深入的掌握单片机的开发应用和 PC 编程控制。为以后从事单片机软硬件产品的设计开发、PC 软件开发打下了良好的基础，树立独立从事产品研发的信心，并在这种能力上得到了比较充分的锻炼。

谢 辞

在本次毕业设计中，我得到了指导老师陈紫强的热心指导。自始至终关心督促毕业设计进程和进度。帮助解决毕业设计中遇到的许多问题。还不断向我们传授分析问题和解决问题的办法，并指出了正确的努力方向，使我在毕设过程中少走很多弯路。同时，他还提供给我们专门的各种设备及场所，在调试过程中能够有充足的时间。在这里非常感谢赵老师的指导和帮助，并致以诚挚的谢意！

同时，身边的同学给了我许多的帮助。在此，我向身边关心我的同学致以诚挚的谢意！另外，系里的领导和老师也给了我们必要的指导，我也向系和年级的领导们表示衷心的感谢！最后感谢学院对我这几年的培养。

参考文献

[1] 何立民.单片机应用系统设计系统配置与接口技术[M].北京：北京航空航天大学，1990.

[2] 李晓荃.单片机原理与应用[M]. 北京:电子工业出版社，2000.

[3] 刘和平.单片机原理及应用[M].重庆：重庆大学出版社，2002 .

[4] 徐爱钧.单片机高级语言 C51 应用程序设计[M]. 北京：电子工业出版社，2002.

[5] 谢自美.电子线路设计.实验.测试(第二版) [M].武汉：华中科技大学出版社，2000.

[6] 江国强.现代数字逻辑电路.北京：电子工业出版社，2002 .

[7] 张勇.PROTEL 99SE 电路设计技术入门与应用(第一版).北京：电子工业出版社，2002 .

[8] 樊昌信.通信原理(第五版)[M].北京：国防工业出版社,2001 .

[9] Richard c.Dorf.modern conctrol systerm[M].BEIJING:Science Publishing House，2002.

[10] Donald A. Neamen. Electronic circuit analysis and design[M].Tsinghua University Press and Springer Verlag.2002.

附 录 1

（1）本设计使用的单片机程序如下：

```
#include <pic.h>
//*************************
void INIT()
{
      ADCON1=0X07;
      TRISC=0X80;
      TRISB=0X00;
       TRISD=0X00;
       RD1=0;
       RD0=0;
       TRISA=0X0f;
       TRISE=0X00;
}
//*************************
#include <pic.h>
#include "init.h"
#include "proc.h"
//*************************
unsigned char i;
unsigned int delay;
extern unsigned char a;
extern unsigned char temph;
extern unsigned char templ;
//*****************************
void main()
{
//初始化
      INIT();
      for(delay=65536;delay>0;delay--) asm("clrwdt");
      temph=0x35;
      templ=0x30;
       do
       {
           asm("clrwdt");
           PROCDIANPIN();
           RC0=0;
           RC1=0;
      }while(1);
}
#include <pic.h>
#include "tranpc.h"
```

```
//**********************
union adres
{
    int y1;
     unsigned char adre[2];
}adresult;
extern unsigned int delay;
unsigned int temp;
unsigned int y;
unsigned char receive;
unsigned char a;
extern unsigned char rxbuf[];
unsigned char temph;
unsigned char templ;
extern unsigned char i;
//*******************************
void PROCDIANPIN()
{
    ADCON0=0X89;
    ADCON1=0X84;
    ADIF=0;
    ADGO=1;
    for(delay=0x8ff;delay>0;delay--)    asm("nop");
     while(ADIF==0)
     {
         asm("clrwdt");
    }
    asm("clrwdt");
    ADIF=0;
    adresult.adre[0]=ADRESL;
    adresult.adre[1]=ADRESH;
    if((adresult.y1<=0x204)&&(adresult.y1>=0xD9))
    {
        temp=0x10;
        for(
    y=0x204;adresult.y1<=y;adresult.y1=adresult.y1+0x07)
        {
            temp++;
            if(temp==0x1a)    temp=0x20;
            if(temp==0x2a)    temp=0x30;
            if(temp==0x3a)    temp=0x40;
              if(temp==0x4a)    temp=0x50;
              if(temp==0x5a)    temp=0x60;
              if(temp==0x6a)    temp=0x70;
```

```
                if(temp==0x7a)    temp=0x80;
                if(temp==0x8a)    temp=0x90;
                if(temp==0x9a)    temp=0x100;
            }
        }
        TXPC(temp);
        RC0=1;
        RXDATAS();
        if(rxbuf[0]!=0)
        {
        if((rxbuf[0]==0x10)&&(rxbuf[1]==0xff))    receive=0xff;
        else if(rxbuf[0]==0x20)
        {
            templ=rxbuf[1];
            temph=rxbuf[2];
        }
        if(receive==0xff)
        {
            RC1=1;
            a=0xff;
        }
        }
        if(temp<=templ)
        {
            if(a!=0xff)
             RD0=1;
             else RD0=0;
        }
        else if(temp>=temph)
        {
             if(a!=0xff)
             RD1=1;
             else RD1=0;
        }
        else if((temp>=templ)&&(temp<=temph))
        {
            a=0;
            RD0=0;
            RD1=0;
        }
        for(delay=0xFff;delay>0;delay--)    asm("nop");
}
#include <pic.h>
//*************************
```

```
unsigned char txbuf[5];
unsigned char rxbuf[5];
extern unsigned int delay;
unsigned char s_uart_buf;
unsigned char rx_lo_buf;
extern unsigned char i;
//*************************
void TXPC(unsigned char byte)//9600b/s
{
    RC6 = 0;//start bit
    for(s_uart_buf=0;s_uart_buf<46;s_uart_buf++)continue;
    if(byte&0x01)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x02)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x04)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x08)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x10)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x20)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x40)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    if(byte&0x80)RC6=1;
    else RC6=0;
    for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
    RC6=1;//stop bit
    for(s_uart_buf=0;s_uart_buf<45;s_uart_buf++)asm("nop");
}
//*************************************************
unsigned char RXPC(void)//9600b/s
{
    rx_lo_buf=0;
    while(1)
    {
```

```
            if(!RC7) break;
        }
//receive start bit
        for(s_uart_buf=0;s_uart_buf<46;s_uart_buf++)continue;
//receive bit
        for(s_uart_buf=0;s_uart_buf<17;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x01;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x02;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x04;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x08;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x10;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x20;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x40;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
        if(RC7)rx_lo_buf=rx_lo_buf|0x80;
        for(s_uart_buf=0;s_uart_buf<35;s_uart_buf++)asm("nop");
//receive stop bit
        for(s_uart_buf=0;s_uart_buf<10;s_uart_buf++)asm("nop");
        return rx_lo_buf;
}

void RXDATAS()
{
//开始接收数据
         for(i=0;i<5;i++) rxbuf[i]=0;
        for(i=0x04;i>0;i--)
        {
            asm("clrwdt");
            if(RC7==0) break;
            for(delay=65535;delay>0;delay--)
            {
                asm("clrwdt");
                if(RC7==0) break;
            }
        }
        if(RC7==1)
        {
            goto rxend;
```

```
    }
    for(i=0;i<5;i++)
    {
        rxbuf[i]=RXPC();
        if(rxbuf[i]==0x21) break;
    }
rxend:
    asm("clrwdt");
}
```