21 世纪高等院校优秀教材

智能化检测系统及仪器

张剑平　著

内 容 简 介

本书讲述了智能化检测系统及仪器的设计方法，主要内容包括：导论，信号通道，人机交流系统，系统总线和通信接口，常见的智能处理功能，常见模拟量信号的检测方法，智能化测试系统设计举例和附录共 8 个部分，可作为测试、控制、信息等电子应用类专业的学生和工程技术人员的参考书。

图书在版编目（CIP）数据

智能化检测系统及仪器/张剑平著. —北京：国防工业出版社，2005. 8
21 世纪高等院校优秀教材
ISBN 7-118-04040-1

Ⅰ. 智… Ⅱ. 张… Ⅲ. 自动检测系统
Ⅳ. TP274

中国版本图书馆 CIP 数据核字（2005）第 079862 号

（北京市海淀区紫竹院南路 23 号）
（邮政编码 100044）
北京奥鑫印刷厂印刷
新华书店经售
*
开本 787×1092 1/16 印张 14 320 千字
2005 年 8 月第 1 版 2005 年 8 月北京第 1 次印刷
印数：1—4000 册 定价：22. 00 元

（本书如有印装错误，我社负责调换）

国防书店：（010）68428422 发行邮购：（010）68414474
发行传真：（010）68411535 发行业务：（010）68472764

前　言

本书讲述了智能化检测系统及仪器的设计方法，其中有很多内容是作者根据自己多年从事本专业科研和教学的一些体会总结出来的，读者在已经具有模拟电子、数字电子、微机原理、单片机原理、程序设计等知识的基础上学习将会有更好的效果。本书在内容及编排上具有以下特点：

⑴ 突出工程特色和现代特色。工程特色是指不过分剖析原理和推导结论而以实用性为主，仅对有助于理解概念和方法的原理和结论做了讲述和推导；现代特色是指内容力求不过时，除了一些经典知识必须介绍外，尽可能介绍目前正在流行的或是前沿的技术及方法。

⑵ 考虑到互联网给工程技术人员带来查阅资料的方便性，书中对所涉及到的元器件、芯片及软件包等内容只从使用的角度做引导性、关键性的介绍，力求在不妨碍读者对整体思路理解的前提下达到篇幅少、内容多的目的。

⑶ 顾及到工程技术人员的习惯，遵循“用图纸说话”的原则，书中采用了大量的插图，相比之下文字叙述比较简捷。凡涉及软件设计的地方大部分是用程序流程图来说明思路，仅在必须用程序说明原理的地方给出了部分源程序。

⑷ 本书在全面介绍智能化检测系统设计过程中突出了硬件设计部分，书中所介绍的电路例子都是经使用或实验过的。

本书由田生喜老师主审并提出很多宝贵意见，在此表示感谢。

由于时间仓促再加水平有限，书中难免有错误和不妥之处，欢迎不吝赐教。

作　者

2005 年 4 月

目　　录

第1章　导　论

1.1　检测系统的任务

人类出现以后，一直伴随着认识世界和改造世界的过程。初期人类主要靠五官和肢体来认识周围的世界。随着人类的进化，各种检测的方法及仪器不断诞生，用来提高自身认识世界的能力。到了近代和现代，各种仪器设备的发展尤其迅猛。本书所说的检测系统就是指人们在社会活动中用来获得各种自然参数的装置。这种自然参数可以是物理量，如温度、压力、电流、速度等，也可以是化学量或其它量，如酸碱度、气味、物质成分等。

检测系统可以分为两大类。第一类是通用检测系统，以通用仪器为主，如万用表、示波器、频率计、信号发生器、频谱分析仪等，这一类检测系统通常在实验室发挥作用。第二类为专用检测系统，用于特殊的目的，如医疗、地震、航空航天、煤炭化工、冶金、加工业等，各种场合都有很多的专用仪器及设备，因此，专用检测系统品种更加繁多。

1.2　检测系统的发展

近代的检测受电子技术的影响以电子检测系统为主，其发展经历了模拟检测系统、数字检测系统、智能检测系统等3个阶段。

第1代 模拟检测系统。模拟检测系统的结构如图1.1所示，其特点是系统各部件处理的信息流为模拟信号，各部件几乎全为模拟器件，控制部分用多种旋钮及开关。模拟检测系统设计调试比较麻烦，操作复杂，稳定性差。模拟检测系统正在不断被数字及智能检测系统取代。许多模拟检测系统目前仍在使用，如模拟万用表、模拟示波器、模拟信号发生器、模拟压力表等。

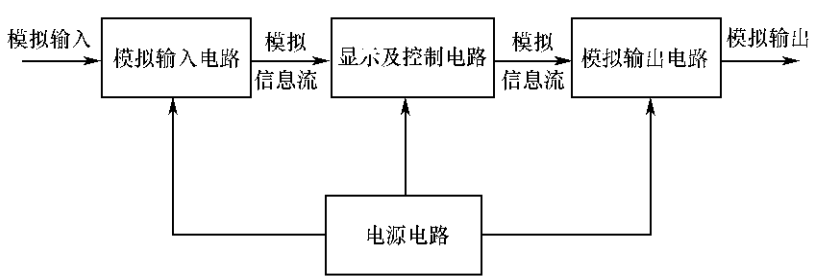


图1.1　模拟检测系统基本结构

第2代 数字检测系统。数字检测系统的结构如图1.2所示，其特点是核心处理部分

处理的信号流为数字信号，模拟信号经过 A/D 转化为数字信号，模拟输出信号是经 D/A 转换后的信号，显示为数字或经 D/A 转换后的模拟信号，控制为按钮、旋钮、开关等。数字检测系统的控制采用通用或专用数字芯片，系统还可以带有数字通信接口，操作简单，稳定性好，显示直观，精度较高，如数字万用表、取样示波器、数字频率计、数字信号发生器、数字流量表等。

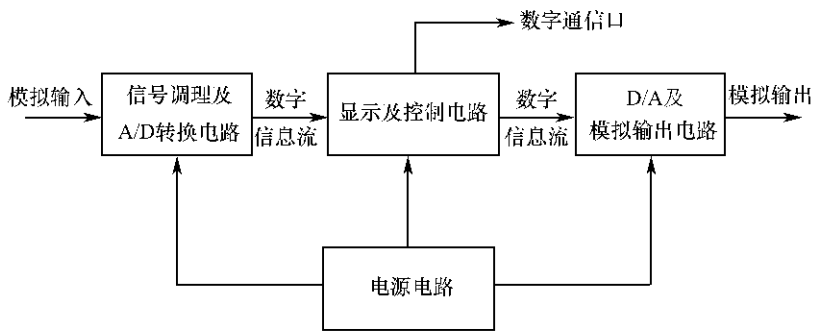


图 1.2　数字检测系统基本结构

第 3 代 智能检测系统。数字检测系统比模拟检测系统具有较大的优越性，有的可实现自动控制量程、自动调零、数字储存及数字标准接口等功能，但还是存在着功能单一、结构复杂等缺点。随着计算机的发展，智能检测系统以更大的优越性展现在人们面前。当检测系统的核心控制由电脑来完成时，称之为智能检测系统。

智能检测系统分为两类，一类是电脑内藏式，另一类是电脑扩展式。电脑内藏式智能检测系统如图 1.3 所示，它将电脑或控制器作为核心部分嵌入检测系统中，利用微机强大的功能完成检测系统的各项任务，如信号调理、A/D 转换、数字处理、数据存储、显示、打印、通信等。而电脑扩展式检测系统给使用者的感觉首先是一个微机系统，它相当于电脑系统扩展一个检测功能，又称为虚拟仪器（Virtual Instrument），其结构如图 1.4 所示。

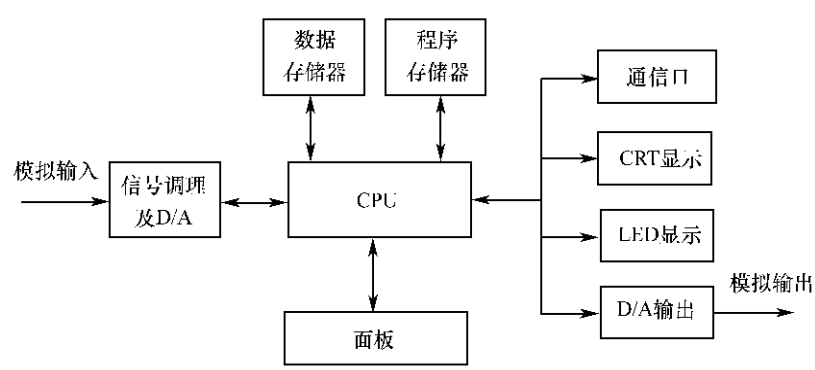


图 1.3　微机内藏式检测系统基本结构

智能仪器的特点如下。

⑴ 测量过程由程序控制可实现自动调零、自动极性识别、自动量程转换、报警、过载

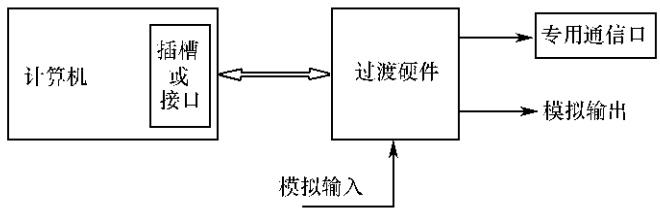


图1.4 微机扩展式检测系统基本结构

保护、非线性测量、系统自检、自学习等功能。

⑵ 可实现数据处理、改善测量精度、信号再加工、数字滤波、复杂计算、非线性修正等功能。

⑶ 具有功能多样性、功能可增可减可扩展的特性。

⑷ 灵活性好，兼容性好。

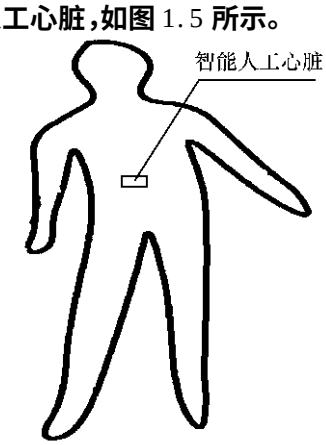
⑸ 方便实现各种通用接口，如RS-232接口、GP-IB接口、USB接口、红外接口等。

⑹ 可利用互联网等公共网络进行仪器入网实现远程测量。

⑺ 方便实现无人测量，减少人工劳动。

1.3 嵌入式系统的概念

嵌入式系统(Embedded System)是一个形象化的概念，它是计算机技术发展到一定程度时形成的。早期的计算机因造价较高，为了不浪费资源，人们在使用时尽可能地让计算机完成更多的任务，尤其是在很多民用场合，用计算机组成的检测及控制系统与计算机本身相比处于从属地位。随着微电子技术的发展，计算机发生了两大变化：其一是价格变得很低，其二是体积变得很小。计算机在全球范围内得到了较大的普及，与此同时涌现出了大量的价格较低的单片计算机。人们在设计许多应用系统时把计算机作为一个部件来使用，有的系统甚至包含多个这样的部件，就好像将计算机嵌入到用户系统中一样，因而产生了嵌入式系统的概念。例如，比较高级一点的汽车，燃料喷射系统、空调系统、音响部分、ABS系统、卫星定位系统、安全气囊系统等多处都用到单片计算机。一个非常形象而典型的嵌入式系统就是智能人工心脏，如图1.5所示。



1.5 一个典型的嵌入式系统

1.4 智能单元的选择

智能单元包括电脑（Computer）、可编程控制器（PLC）、单片机（μP）、信号处理机（DSP）等，它是智能检测系统的核心。智能单元的选择应根据实际情况来决定。一个大型的智能检测系统可能要包括多种多个智能单元 对于一个小型的智能检测系统来讲，一般只用其中之一就够了。

Computer 一般指一个完整的电脑系统，包括基本输入输出设备及常用外围设备，具有完善的操作系统。它适合于用作系统管理，通常处在用户的最上层。如果是大型的检测系统，也可将 Computer 放置在系统的底层或中间层。Computer 的选型应根据系统可靠性要求决定。

PLC 是为工业测控专门设计的高可靠产品，其功能没有 Computer 那么强大，但也具有基本的大众化操作系统，它适合于中规模的现场检测及控制。用 PLC 设计的系统便于更改和维护，相应的造价也不低。

μP 就是在一块芯片上集成了 CPU、RAM、ROM、时钟、定时/计数器、串行、并行 I/O 口等的微型计算机，有的 μP 甚至包括 A/D、D/A、模拟比较器、脉宽调制器、USB 口等，由于其功能日趋增多，称之为微控制器比单片机更准确。μP 价格比较低廉，有的只有几元人民币，非常适合于开发各种产品、部件及中小型系统。同时 μP 知识在工程技术人员中也较普及，本文在讲解内容时所用到的智能单元以 μP 为主。

DSP 是一种强化速度的 μP，主要用于信号处理及其它高速要求的场合。与单片机相比，DSP 器件具有较高的集成度、更快的 CPU 及更大容量的存储器。DSP 器件采用改进的哈佛结构，具有独立的程序和数据空间，允许同时存取程序和数据。内置高速的硬件乘法器、增强的多级流水线，使 DSP 器件具有高速的数据运算能力。DSP 器件指令执行时间比 16 位单片机快 8 倍～10 倍，完成一次乘运算的时间比单片机快 16 倍～30 倍。DSP 器件还提供了高度专业化的指令集，提高了 FFT 快速傅里叶变换和数字滤波的运算速度。但 DSP 目前价格要高一些，在满足速度要求的情况下设计者首选还是单片机。

1.5 两种典型的单片机

一、C8051F020/21 单片机

C8051F02X 是美国 Silabs 公司 F 系列单片机之一。Silabs 公司研制 μP 产品起点较高，在产品结构上具有许多闪光点。F 系列单片机有着极符合人性的一些特点，从这些特点可以很明显地看出该公司的研制人员中具有绝对内行的 μP 使用专家。虽然产品在可靠性方面还存在一些小问题，但该公司对产品的改进和更新比较迅速，使用者还是认可的。在需要 A/D、D/A、比较器、多端口、多中断等的场合使用 F02X 单片机是比较合适的，图 1.6 是 F020 的内部结构示意图。下面是该款单片机的一些特性。

⑴ 12 位 A/D：

- 可编程转换速率最大 100kb/s；

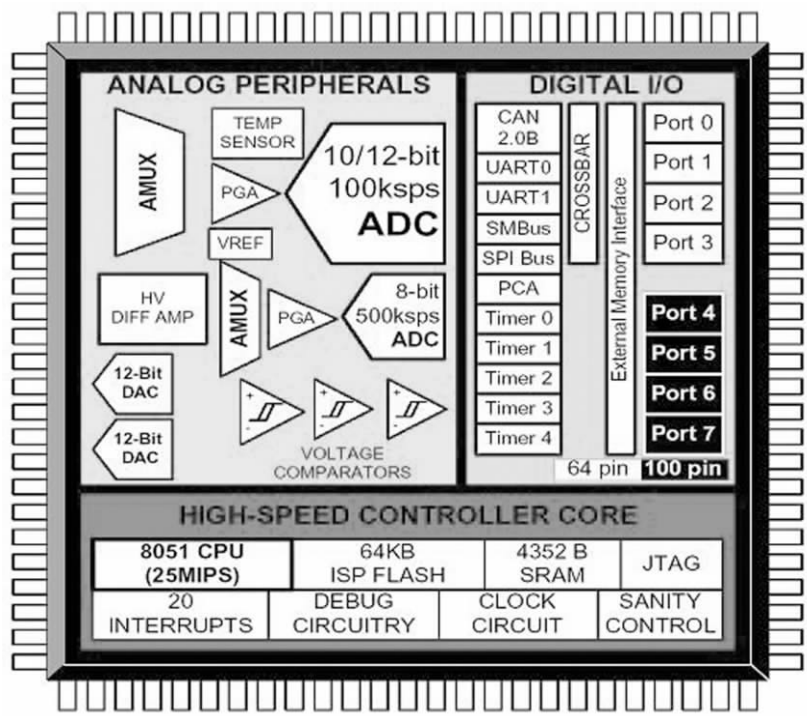


图1.6 C8051F02X单片机内部结构示意图

- 12个外部输入可编程为单端输入或差分输入；
- 可编程放大器增益16,8,4,2,1,0.5；
- 内置温度传感器；
- 16种增益设置从0.05至16。

⑵ 8位A/D：

- 可编程转换速率最大500kb/s；
- 8个外部输入可编程为单端输入或差分输入；
- 可编程放大器增益4,2,1,0.5。

⑶ 两个12位D/A。

⑷ 3个模拟比较器。

⑸ 内部电压基准。

⑹ 精确的VDD监视器和降压检测器。

⑺ 片内JTAG调试和边界扫描：

- 片内JTAG调试电路提供全速非侵入式的在系统调试无需仿真器；
- 支持断点单步观察点堆栈监视器程序跟踪存储器；
- 支持观察/修改存储器和寄存器；
- 比使用仿真芯片目标仿真头和仿真插座的仿真系统有更好的性能；
- 完全符合IEEE1149.1边界扫描标准。

⑻ 高速8051微控制器内核：

- 4352 字节内部数据 RAM 256b + 4kb；
- 64k 字节在系统可编程 FLASH 程序存储器；
- 外部 64k 字节数据存储器接口。

⑼ 64 个 I/O 口线，所有口线均耐 5V 电压。

⑽ 可同时使用的硬件 SMBus™ I²C 兼容 SPI™及两个 UART 串口。

⑾ 可编程的 16 位计数器/定时器阵列 PCA 有 6 个捕捉/比较模块。

⑿ 5 个通用 16 位计数器/定时器。

⒀ 专用的开门狗定时器双向复位时钟源。

⒁ 内部可编程 2% 振荡器最大为 25MHz。

⒂ 其它：

- 供电电压 2.7V～3.6V；
- 典型工作电流 10mA (时)；
- 多种节电休眠和停机模式。

二、P89LPC920/21/22 单片机

LPC92X 是 Philips 公司的一款较低价位的单片机，适合于许多要求高集成度、低成本的场合，可以满足多方面的性能要求。图 1.7 为其内部结构示意图。LPC920/921/922 采用了高性能的处理器结构，指令执行时间只需 2 个～4 个时钟周期，6 倍于标准 80C51 器件。LPC920/921/922 集成了许多系统级的功能，这样可大大减少元件的数目和电路板面积并降低系统的成本。其主要特性如下。

⑴ 当操作频率为 12MHz 时，除乘法和除法指令外，高速 CPU 的指令执行时间为 167ns～333ns。同一时钟频率下，其速度为标准 80C51 器件的 6 倍。只需要较低的时钟频率即可达到同样的性能，这样无疑降低了功耗和 EMI。

⑵ 操作电压范围为 2.4V～3.6V。I/O 口可承受 5V (可上拉或驱动到 5.5V)。

⑶ 4KB/8KB Flash 程序存储器，具有 1KB 可擦除扇区和 64 字节可擦除页规格，可擦除单个字节。

⑷ 2 个 16 位定时/计数器，每一个定时器均可设置为溢出时触发相应端口输出或作为 PWM 输出。

⑸ 2 个模拟比较器，可选择输入和参考源。

⑹ 增强型 UART。具有波特率发生器、间隔检测、帧错误检测、自动地址识别和通用的中断功能。

⑺ 400kHz 字节宽度的 I²C 通信端口。

⑻ 8 个键盘中断输入，另加 2 路外部中断输入。

⑼ 看门狗定时器具有片内独立振荡器，无需外接元件。看门狗定时器溢出时间有 8 种选择。

⑽ 端口“输入模式匹配”检测。当 P0 口管脚的值与一个可编程的模式匹配或者不匹配时，可产生一个中断。

⑾ 施密特触发端口输入。

⑿ 最少 15 个 I/O 口，选择片内振荡和片内复位时可多达 18 个 I/O 口。

⒀ 串行 Flash 编程可实现简单的在电路编程。Flash 保密位可防止程序被读出。

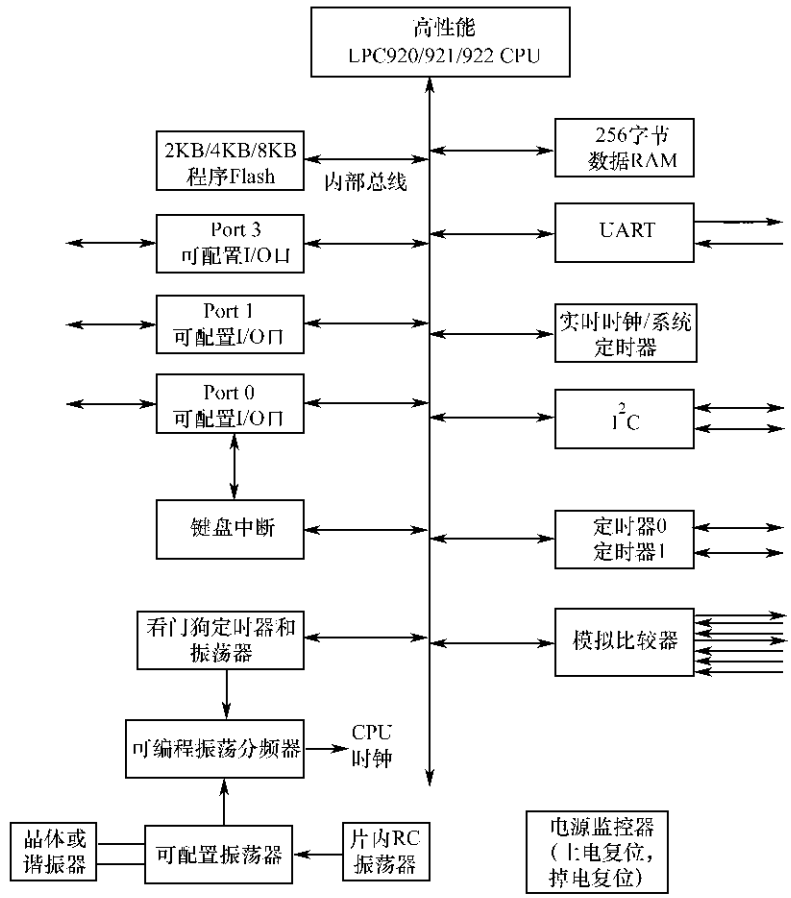


图 1.7 LPC920/921/922 **的内部结构**

(14) Flash **程序存储器可实现在应用中编程。这允许在程序运行时改变代码。**

(15) 空闲和两种不同的掉电节电模式。提供从掉电模式中唤醒功能(低电平中断输入唤醒)。典型的掉电电流为 1μA **(比较器关闭时的完全掉电状态)。**

1.6 在系调试及编程的概念

传统的单片机系统开发过程是这样的:先用仿真法或试探法调试用户程序,再用写入器将程序固化在芯片中,然后才能焊接或装配产品。随着嵌入式技术的发展,使用了 Flash **等技术作为程序或数据存储器的单片机越来越多,直接在用户板上调试并固化程序是众望所归,这就是所谓的在系调试和在系编程,在系编程又称** ISP (In System Programming)。

目前较流行的在系调试及编程是非侵入式边界扫描技术 (Joint Test Action Group, JTAG)。JTAG **是** 1985 **年制定的检测** PCB **和** IC **芯片的一个标准,**1990 **年被修改后成为** IEEE **的一个标准,即** IEEE1149.1 - 1990。**通过这个标准,可对具有** JTAG **口芯片的硬件**

电路进行边界扫描和故障检测。具有 JTAG 口的芯片都有如下 JTAG 引脚定义：

TCK——测试时钟输入；

TDI——测试数据输入，数据通过 TDI 输入 JTAG 口；

TDO——测试数据输出，数据通过 TDO 从 JTAG 口输出；

TMS——测试模式选择，TMS 用来设置 JTAG 口处于某种特定的测试模式。

上述 Silabs 公司的单片机几乎都具有 JTAG 接口，通过 JTAG 既可以在系调试也可以在系编程，给用户带来极大的方便。

有的单片机虽然没有 JTAG 口，但在出厂时在 Flash 中固化了一部分可对 Flash 进行修改的系统程序，允许用户通过单片机本身的串口（TXD、RXD）调用该程序对 Flash 进行改写，从而实现 ISP 功能，尽管不能实现在系调试也极大地方便了研发人员。例如 Philips 公司的 P89LPC92X 单片机就有此功能。

1.7 一种流行的单片机开发软件

单片机的发展越来越迅速，表现在其速度在不断提高，片上 RAM 在不断加大，片上 PROM（包括 EEPROM、OTPROM、FLASHROM 等）不断加大，这使得利用高级语言开发单片机程序变得切实可行。事实上，近年来完全用汇编语言开发单片机程序的人越来越少，用于 51 系列单片机开发的比较流行的高级语言是 C51，在这方面贡献最大的是 Keil 公司。

Keil 软件公司的 8051 单片机软件开发工具可用于众多的 8051 派生器件以实现嵌入式应用开发，其工具清单包括：C51 优化 C 编译器，A51 宏汇编器，8051 工具连接器目标文件转换器库管理器，具有 PC 机源程序调试器、模拟器、仿真器等功能的集成开发环境，8051 开发工具套件等。

使用 Keil 的开发工具时，其项目开发周期和任何软件开发项目大致一样，其步骤为：①创建 C 或汇编语言的源程序；②编译或汇编源文件；③纠正源文件中的错误；④从编译器和汇编器连接目标文件；⑤测试连接的应用程序。

为了节省学习时间提高产品开发效率，单片机使用者往往希望用同一种开发软件开发出更多的产品。Keil 公司真是迎合了使用者的愿望，Keil 的软件库几乎包括了所有的单片机厂家的产品调试驱动软件，这同时也节省了单片机生产厂为其产品研制开发手段的时间。事实上，目前用单片机生产厂提供的开发软件开发产品的人越来越少，厂家只需做开发用的适配器和评估测试板就可以了，厂家还需做的一件重要事情就是每开发成功一款产品先将 C51 套件所需的资料递交 Keil 公司。

对于一个已有一点单片机使用和 C 编程经历的人来讲，使用 C51 并非难事。C51 的基本语句和程序结构和 C 完全相同，它其实是 C 的扩展，扩展部分包括：数据类型、存储器类型、存储器模型、中断函数、实时操作系统、C51 和 A51 源文件接口等。

例如在数据声明时，语句“sbit sda = P2^7；”表示定义 sda 为 P2.7；语句“sfr16 ADC = 0x00be；”表示 ADC 的地址为 00beH；语句“unsigned char data txd[21]；”表示 txd[21]为在内部 RAM 寻址的单字节数组；语句“unsigned int xdata me；”表示 me 为在外部 RAM 寻址的整形数；语句“unsigned char code product_number_at_0x1c00；”表示 product_num-

ber 这个变量在程序代码区的 1c00H 地址。

再如在选择编译模型时，可选择 SMALL、COMPACT、LARGE 等 3 种模型：

SMALL 模型中，所有变量都默认位于 8051 内部数据存储器，这和使用 data 指定存储器类型的方式一样，此模型对于变量访问的效率很高，但所有的数据对象和堆栈必须适合内部 RAM；

COMPACT 模型中，所有变量都默认在外部数据存储器的一页内，这和使用 pdata 指定存储器类型一样，该存储器类型适用于变量不超过 256 个字节，此限制是由寻址方式所决定的，该存储器模型的效率低于 SMALL 模型对变量访问的速度要慢一些；

LARGE 模型中，所有变量都默认位于外部数据存储器，这和使用 xdata 指定存储器类型一样，该存储器类型使用数据指针 DPTR 进行寻址效率较低，特别是当变量为 2 个字节或更多字节时，该模型的数据访问比 SMALL 和 COMPACT 产生更多的代码。

图 1.8 为 Keil - C51 显示界面之一，系统的编辑、模拟仿真、实时仿真都由一个组合软件完成，借助于系统的“帮助”较容易上手，这里不再赘述。

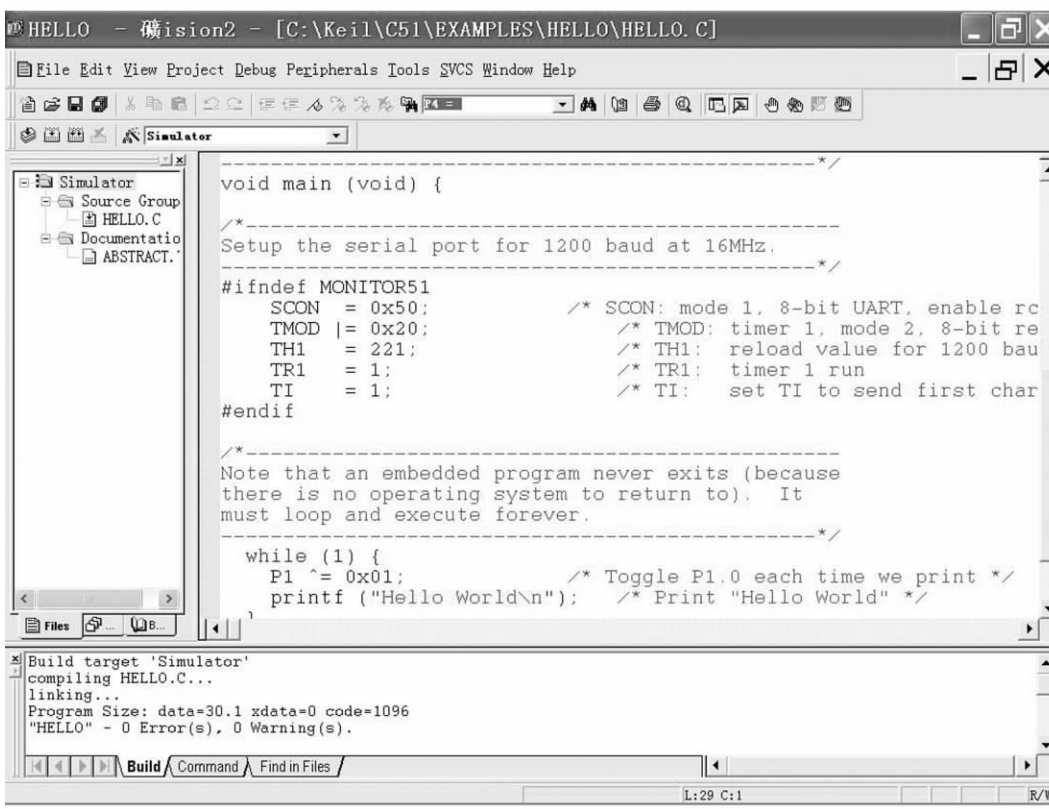


图 1.8 Keil - C51 的编辑界面举例

第 2 章　信 号 通 道

2.1　模拟量输出通道

2.1.1　D/A 及其接口电路

一、D/A 基本原理

D/A 是 Digital to Analog 的缩写，有时也用 DAC (Digital - Analog Converter) 来表示，即数字量—模拟量转换器。最常见的 D/A 是 T 型网络电阻型 D/A。图 2.1 为 8 位 T 型网络 D/A 内部结构，$S_0 \sim S_7$ 位由 $D_0 \sim D_7$ 分别控制的模拟开关，由 R 和 2R 两种电阻组成，在工艺上只需做两种电阻，这样容易控制精度。

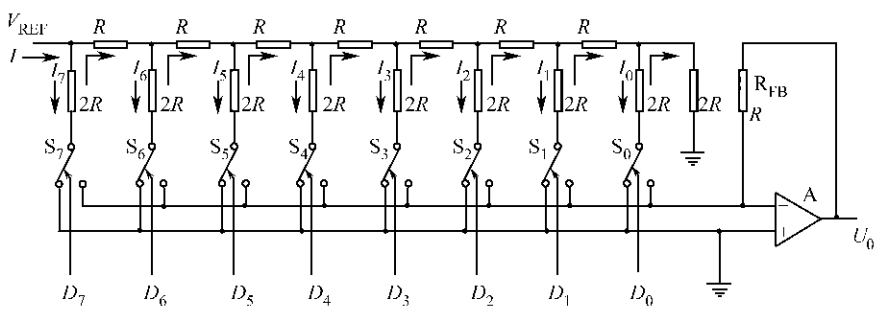


图 2.1　T 型网络 D/A 原理

现在推导一下输入输出关系。从图中弯箭头所示各处看进去等效电阻都为 2R，该等效电阻总是与垂直的电阻 2R 均分前面电流，V_{REF} 处的等效电阻为 R，当有参考电压 V_{REF} 时，总电流 $I = \dfrac{V_{REF}}{R}$，开关 $S_7 \sim S_0$ 由 D_7 - D_0 控制，$D_i = 1$ 时，支路电流 I_i 流入虚地，$D_i = 0$ 时，支路电流 I_i 流入实地：

$$I_7 = I \times \frac{1}{2} = I \times \frac{1}{2^1}$$

$$I_6 = I \times \frac{1}{4} = I \times \frac{1}{2^2}$$

…

$$I_0 = I \times \frac{1}{2^8}$$

当全部 $D_7 \sim D_0 = 11111111B = FFH$ 时，注入虚地电流为

$$I' = I_7 + I_6 + \cdots + I_0 = I \times \frac{1}{2} + I \times \frac{1}{2^2} + \cdots + I \times \frac{1}{2^8} =$$

$$\frac{V_{REF}}{R}\left(\frac{1}{2^1} + \frac{1}{2^2} + \cdots + \frac{1}{2^8}\right) =$$

$$\frac{V_{REF}}{R \times 2^8}(2^7 + 2^6 + \cdots + 2^0)$$

如果 $D_7 \sim D_0$ **为任意值** D **时，有**

$$I' = \frac{V_{REF}}{R \times 2^8}(D_7 2^7 + D_6 2^6 + \cdots + D_0 2^0)$$

$$U_0 = - I'R = \frac{V_{REF}}{2^8}(D_7 2^7 + D_6 2^6 + \cdots + D_0 2^0)$$

即

$$U_0 = - V_{REF}\frac{D}{256}$$

这说明，D/A **的输出** U_0 **为输入数字** D **和参考电压** V_{REF} **之积，由此实现** D/A **转换。**

二、D/A **与** μP **的接口设计**

D/A **分两大类。第** 1 **类就是一个基本的** T **型网络，其结构和前述一样。该类** D/A **不带数据锁存器，必须在数字输入端保证要转换的数字不变。如** DAC1020 **就属这种类型。**DAC1020 **是** 10 **位** D/A，**图** 2.2 **为其管脚图，**1 **为虚地电流输出端，**2 **为实地电流输出端，**3 **为电源地，**4 ~ 13 **为数字输入端，**14 **为电源输入，**15 **为参考电压输入端，**16 **为反馈电阻接入端。第** 2 **类** D/A **是自带数据锁存器，不需单独的数据锁存器，只需选通、写入之类的信号即可将数据写入并锁存，数据锁存后数据输入端的数据就可变化了。如** AD7545 **(**12 **位) 就属此类，图** 2.3 **为其管脚图，**1 **为虚地电流输出端，**2 **为实地电流输出端，**3 **为电源地，**4 ~ 15 **为数字输入端，**18 **为电源输入，**19 **为参考电压输入端，**20 **为反馈电阻接入端，**16、17 **分别为选通和写入输入端。**

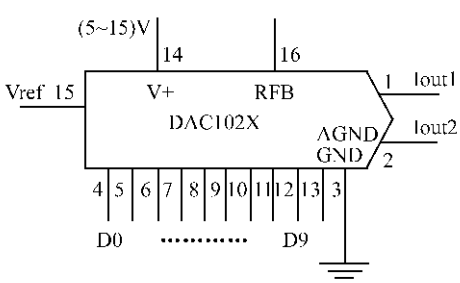


图 2.2　DAC102X **的管脚图**

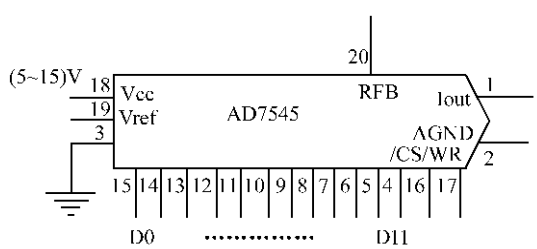


图 2.3　AD7545 **的管脚图**

(一) 不含锁存器的 D/A **与** μP **的接口**

图 2.4 **为** DAC1020 **与** μP **的接口之一，用两个锁存器锁定要转化的数据。设要转换的数据为** 1FFH，**低位译码地址为** 0000H，**高位译码器的地址位** 0100H，**程序如下：**

```
MOV A,#0FFH          ;要转换的低位数
```

```
MOV DPTR,#0000H      外设地址
MOVX @DPTR,A         送出低位数
MOV A,#01H           要转换的高位数
MOV DPTR,#0100H      外设地址
MOVX @DPTR,A         送高位数
```

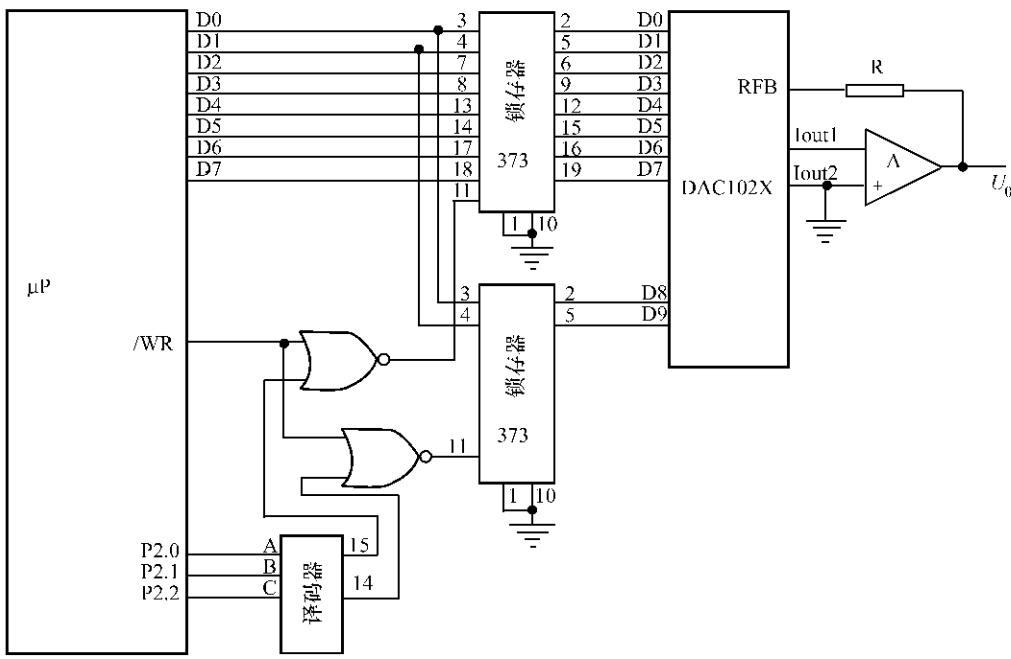


图 2.4　DAC1020 与单片机的接口

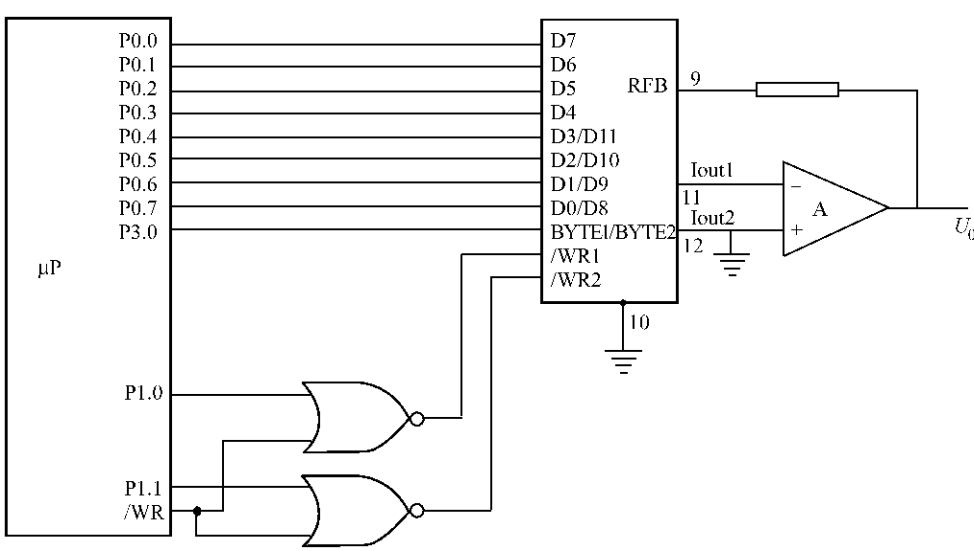


图 2.5　DAC1230 与单片机的接口

（二）带锁存器的 D/A 与 μP 接口的接口

图 2.5 为 DAC1230 与单片机的接口电路之一，DAC1230 是含有 μP 接口的 12 位数模转换器，BYTE1/BYTE2 = 1 锁定低 8 位，BYTE1/BYTE2 = 0 锁定高 4 位，用 WR1 控制；全部锁定用 WR2 控制。在本方案中，数据从单片机的 P0 口送出，P1.0、P1.1 直接产生译码信号。例如将数值 AAAH 送出的 D/A 转换程序如下：

```
MOV P0,#0AAH          ;要送出的低位数据
SETB P3.0             ;产生译码
SETB P1.1
CLR P1.0
MOVX @DPTR,A          ;产生写信号 WR1
MOV P0,#0AH           ;要送出的高位数据
CLR P3.0              ;产生译码
MOVX @DPTR,A          ;产生写信号 WR1
SETB P1.0
CLR P1.1
MOVX @DPTR,A          ;产生写信号 WR2,锁定数据
```

图 2.6 为另一种 12 位 D/A 转换器 AD7545 与 μP 的接口电路，AD7545 有 12 位数据输入，一次锁定，同样将 AAAH 的数据送出的程序如下：

```
SETB P3.0             ;选通无效
MOV P0,#0AAH          ;送低位数据
MOV A,P1              ;P1 的高 4 位不可受影响
ANL A,#0F0H           ;送高位数据
ORL A,#0AH
CLR P3.0              ;选通有效
MOVX @DPTR,A          ;产生写信号
```

2.1.2　μP 控制的波形发生器

一、阶梯波发生器

带有 μP 的 D/A 可以变换成各种波形的发生器，其硬件结构是一样的，比如我们可以用图 2.7 的电路来实现。要输出各种不同的波形仅仅是软件不同。

如果送入 D/A 的数字由 0 不断增加将形成阶梯波，下面的这段程序每隔一个 DELAY 的时间送出一个值，送出值从 0 一直到 255，由此产生一个如图 2.8 的阶梯波：

```
    CLR P2.0          ;选通
    MOV R0,#255       ;设计数器
    MOV A,#0H         ;从 0 开始
ST1：MOV P1,A
    CLR P2.1          ;写入
    NOP
```

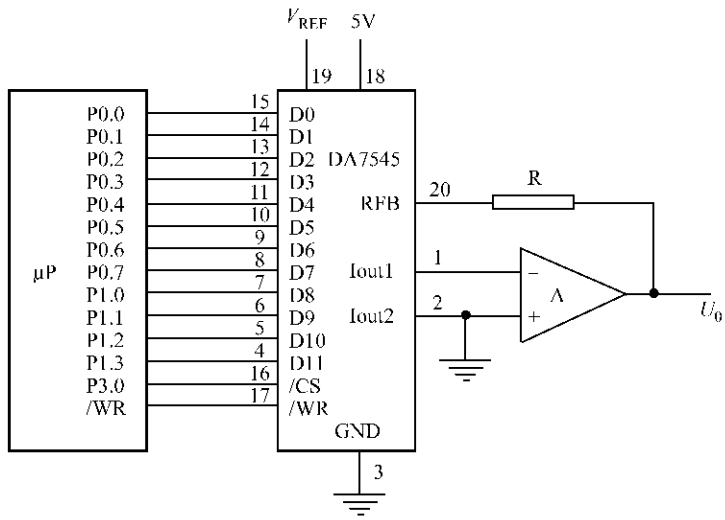


图 2.6　AD7545 与单片机的接口

```
        NOP
        SETB P2.1
        ACALL DELAY          延时
        INC A
        DJNZ Ro,ST1
LOOP：SJMP LOOP              停止
```

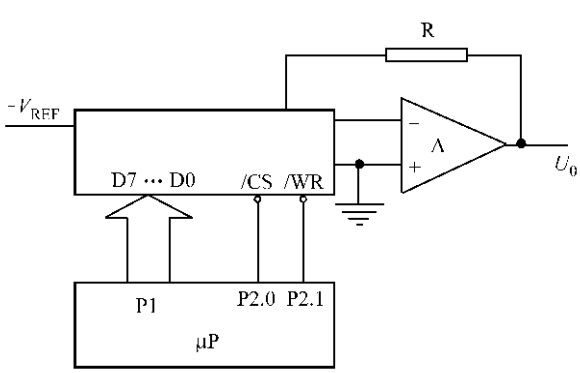


图 2.7　D/A 输出基本电路

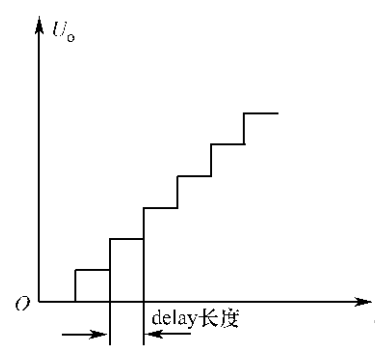


图 2.8　用 D/A 实现的阶梯波

通过调节 DELAY 的参数可产生不同斜率的阶梯波，如将参考电压 V_{REF} 变为正值将产生负阶梯波，如连同计数器 R0 的初值和终值也改变可以得到任意的正阶梯波。当延时 DELAY 值较小并且 D/A 位数较高时，可近似认为阶梯波为直线的斜波。

二、锯齿波发送器

将斜波循环产生便可实现锯齿波，图 2.9 为正锯齿波波形，图 2.10 为负锯齿波波形。同样用图 2.7 电路产生正锯齿波的程序如下。

```
        CLR P2.0
```

```
ST0：MOV R0,#255        ;设计数器,大循环开始
     MOV A,#0
ST1：MOV P1,A           ;小循环,产生一个锯齿波
     CLR P2.1
     NOP
     NOP
     SETB P2.1
     ACALL DELAY        ;延时
     INC A
     DJNZ Ro,ST1        ;小循环判断
     AJMP ST0           ;产生下一个锯齿波
```

本程序中累加器 A 初值取大数，“INC A”改为“DEC A”就可产生负向锯齿波，波形如图 2.10 所示。

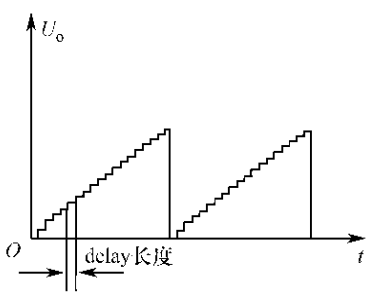


图 2.9　用 D/A 实现的正锯齿波

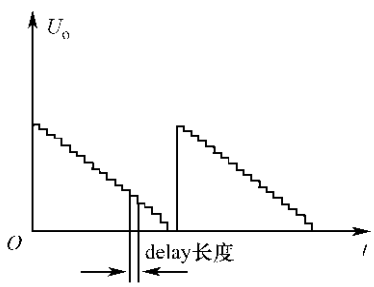


图 2.10　用 D/A 实现的负锯齿波

三、三角波发生器

将上述正向锯齿波和负向锯齿波组合起来可得三角波发送器，例如设计一个最小值为 0V 最大值为 5V 的三角波（见图 2.11），可取 V_{REF} = - 5V，程序如下：

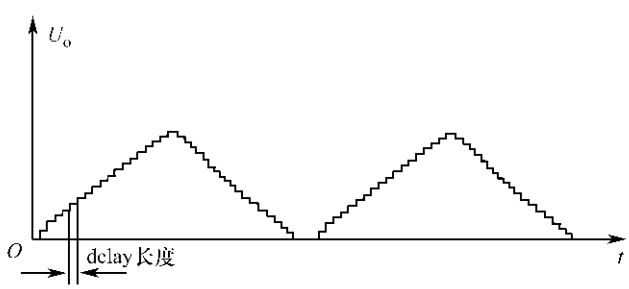


图 2.11　用 D/A 实现的三角波

```
     CLR P2.0
ST0：MOV R0,#255        ;开始一个三角波
     CLR A
ST1：MOV P1,A           ;产生一个上升波
```

```
        CLR P2.1
        NOP
        SETB P2.1
        INC A
        DJNZ R0,ST1
        MOV R0,#255
ST2：MOV P1,A          ;产生一个下降波
        CLR P2.1
        NOP
        SETB P2.1
        DEC A
        DJNZ R0,ST2
        AJMP ST0       ;产生下一个三角波
```

四、如何得到任意极性输出波形

以上电路输出的波形是单向的，例如它可能是 0～5V 的波形或是 0～－5V 的波形，如何得到任意极性输出波形呢？例如想得到－2.5V～2.5V 的三角波、－5V～5V 的梯形波、1V～8V 的三角波等，为此可以在图 2.7 的电路后面增添一个加法电路，让其输出波形与一个直流分量相加而得到所要求的波形。如图 2.12 电路，当基本 D/A 输出为 0～5V 的波形时，最后的输出将是－2.5V～2.5V 的波形。

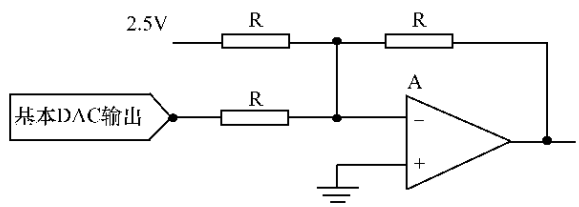


图 2.12　改变输出幅值的方法

五、矩形波发生器

矩形波发生器是用处较多且最容易实现的波形，这两种波形亦可用图 2.7 或 2.12 的电路图来实现，请读者自行编写程序。

六、正弦波发生器

正弦波发生器是最基本的仪器之一，传统的方法是 RC、LC 振荡，其缺点是不稳定、可调节性差、相移困难等。用 μP 加 D/A 实现正弦波具有灵活、方便、可程控各输出参数、准确率高，稳定性好等优点。

图 2.13 为一种正交信号发生器原理框图，μP 通过地址发送器和编码发送器顺序产生正弦及余弦代码由 D/A 输出波形。通常地址码由软件地址发送器产生，当频率极高时可用硬件地址发送器产生。正弦波发送器设计过程如下。

1. 编码表的制作

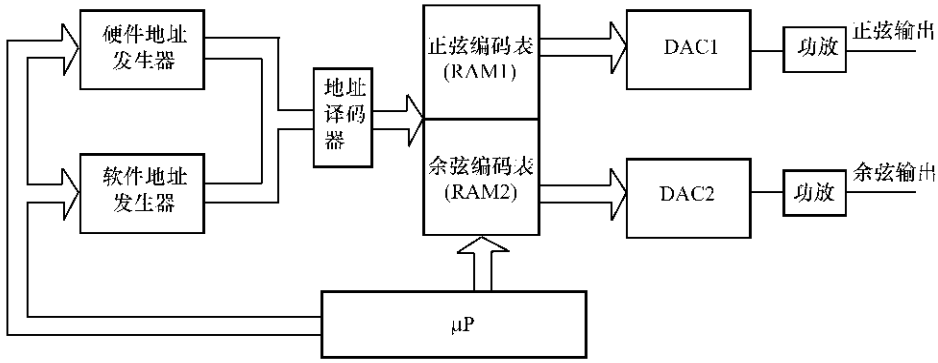


图 2.13　一种正交波输出的方案

⑴ 根据 D/A 位数确定最大值，如为 12 位 D/A 最大值为 4095，中间值为 2048。

⑵ 取波形的 1/4 分成台阶按正弦值计算好若干个点数，按照时间的均匀分配。

⑶ 根据对称关系，复制其它区域各值。

2. 求取数间隔

时间间隔实际上是从 RAM 中取数的间隔，根据要求输出的频率决定。如输出波形的频率为 100Hz，如果每个周期取点数为 1000 个，则取数间隔为 1/(100 × 1000) = 10^{-5}s。

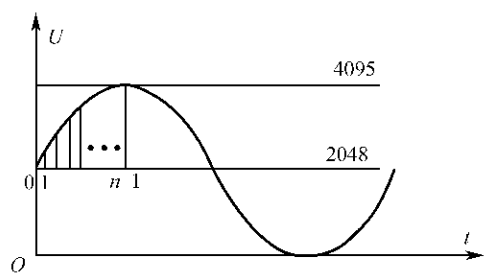


图 2.14　正弦波取点方法

3. 编写程序

例 2.1　设计一个最高频率为 1kHz 的用 8 位 D/A 和 51 单片机组成的单路输出正弦波发生器，要求频率键控、幅值用电位计调节，画出硬件原理图和软件框图。

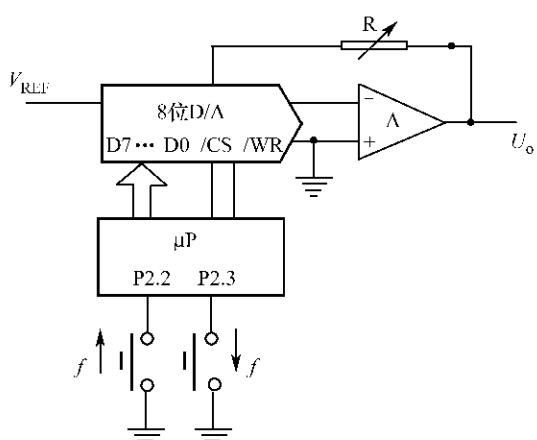


图 2.15　硬件原理图

解：硬件原理如图 2.15 所示，两个按键分别控制频率的增加和减少，电位计 R 调节输出幅值。软件分为主程序（图 2.16）和定时器中断程序（图 2.17）。

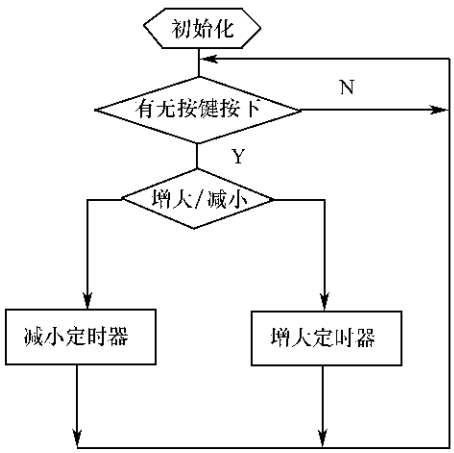


图 2.16　主程序框图

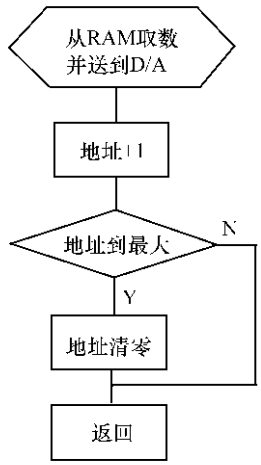


图 2.17　定时器中断流程图

七、任意波形发生器

对给定的任意波形，取其周期内间隔值存入 RAM，如图 2.18，然后用上述方法可实现其波形输出。

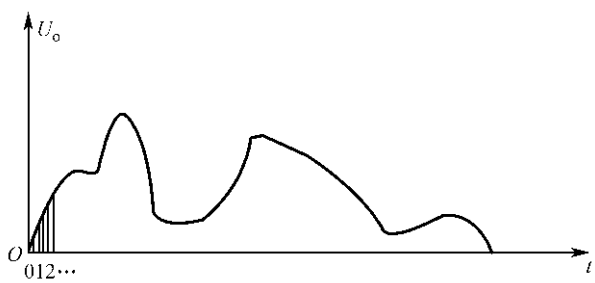


图 2.18　任意波形的一个周期内的数据分割

2.2　模拟量输入通道

2.2.1　A/D 转换器概述

A/D 是 Analog - Digital 的缩写，有时也用 ADC (Analog - Digital Converter) 表示，即模拟量—数字量转换器，其功能和 D/A 相反，A/D 的使用面要比 D/A 更宽一些。

一、A/D 的分辨力和量化误差

A/D 的分辨力是衡量 A/D 能够分辨的输入模拟量的最小变化量的技术指标，它取决于 A/D 输出的有效码的位数，如 12 位分辨力为 $1/2^{12} = 1/4096$。A/D 的输入电压的最大值乘以其分辨力可得到 A/D 的量化单位，量化单位正好是 A/D 数字输出最低有效位 (LSB) 所代表的输入量，习惯上用 LSB 来表示 A/D 的量化单位。

由于 A/D 的输入是连续的量而输出不是连续的，凡不能被量化单位整除的输入量必

然会引起转换误差（称为量化误差）。显然输出位数越高量化误差越小，量化误差应该是1个LSB（见图2.19）。如果把输入电压的量化等级改变一下可以减小量化误差，有的A/D在模拟输入端预先叠加了（1/2）LSB对应的电压，使量化误差为变为（1/2）LSB（见图2.20）。

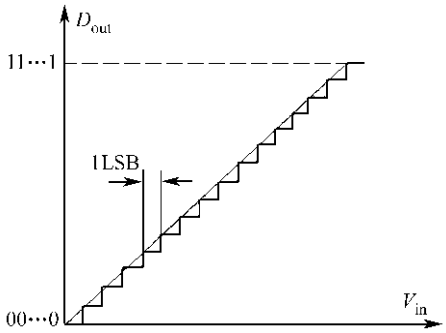


图2.19　1LSB的A/D量化误差

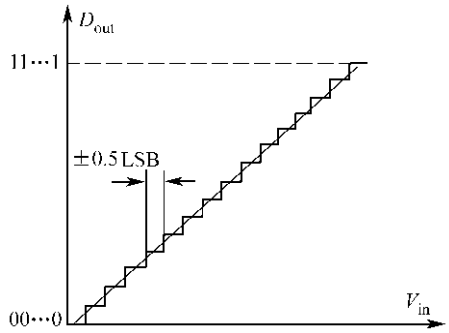


图2.20　0.5LSB的A/D量化误差

二、A/D的转换精度

A/D的转化精度是描述器件本身的精度的指标，通常用转换误差来衡量这个指标，而这里的误差是不考虑量化误差的，因为量化误差是必然存在的不可消除的。A/D的转换误差分为偏移误差、满刻度误差和非线性误差3种。

1. 偏移误差

偏移误差是指在A/D输出为0时对应的不为0的输入值，如图2.21所示。

2. 满刻度误差

满刻度误差是指A/D输出满刻度值时，对应的输入信号值与A/D的理想满量程输入值之差，如图2.22所示。

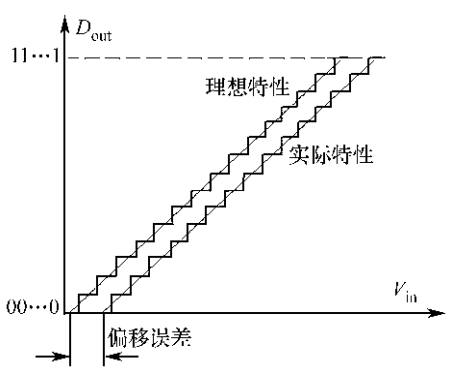


图2.21　A/D的偏移误差

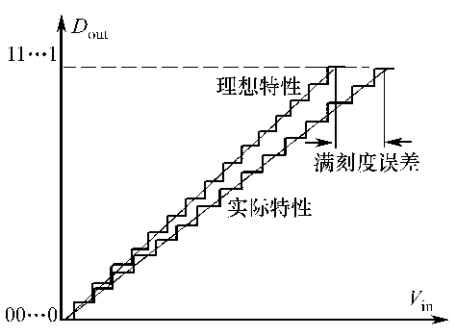


图2.22　A/D的满度误差

3. 非线性误差

非线性误差是指A/D的实际特性曲线和理想特性曲线输出值的最大误差点对应的输入值之差，如图2.23所示。

三、A/D的转换速率

A/D的转换速率一般指A/D在1s内可以完成多少次转换，也可以用完成一次A/D转换所用的时间来表示A/D的这一特性，转换速率越高越好。

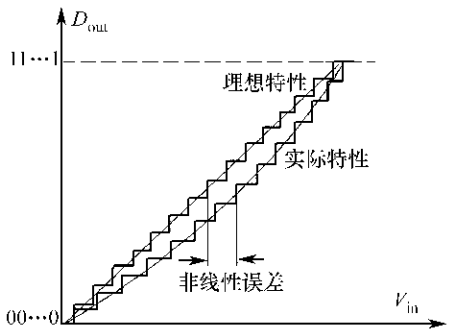


图 2.23　A/D 的非线性误差

四、A/D 的满量程输入范围

满量程范围是指在 A/D 输出从零变到最大值时输入模拟信号对应的输入范围。例如某 12 位 A/D 输出 000H 时对应输入电压是 0V，输出 FFFH 时对应输入电压为 5V，则其满量程范围是 0 ~ 5V。

2.2.2　逐次比较式 A/D 及其与 μP 接口

一、斜坡式 A/D

斜坡式 A/D 是逐次比较式 A/D 的前身，其基本原理如图 2.24 所示，START 有效（变高）后，N 位计数器值先通过单稳清零，DAC 输入为 0，输出 $V_a < V_x$，比较器输出 U_c 为高电平，时钟脉冲 f_c 可通过与门而开始计数。计数器递增时 V_a 变大，当 $V_a > V_x$ 时 U_c 变低停止计数，此时的计数值即为 A/D 转换结果。显然这种 A/D 的转换时间较长，而且转换时间随 V_x 而变化，当输入电压与参考电压相等时需 2^n 个时钟脉冲才能转换完毕。

二、逐次比较式 A/D

逐次比较式 A/D 是在斜坡式 A/D 的基础上发展而来，其转换速度大大提高。基本原理如图 2.25 所示，它由 N 位顺序脉冲发送器、控制器、N 位寄存器、DAC、比较器、三态缓冲器等组成。工作过程如下。

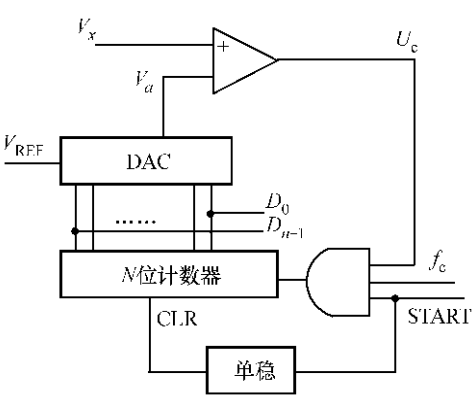


图 2.24　斜坡式 AD 的原理图

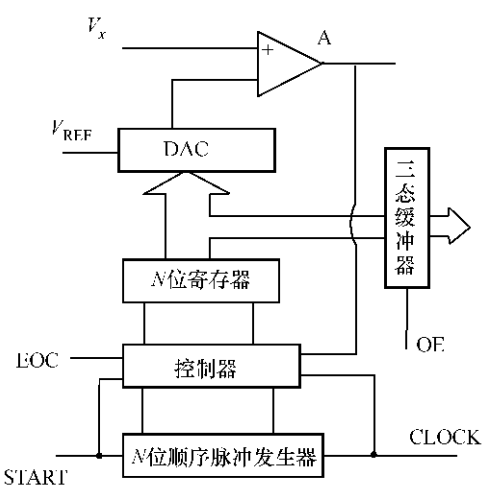


图 2.25　逐次比较式 A/D 的原理图

序脉冲发生器产生 100…0 并送入 D/A 转换后与 V_x 比较，由此决定本位 Q_{n-1} = X（X = 0 或 1）并存入寄存器；

顺序脉冲发生器产生，X100…0 并送入 D/A 转换后与 V_x 比较，由此决定本位 Q_{n-2} = X（X = 0 或 1）并存入寄存器；

……

顺序脉冲发生器产生 XXX…1 并送入 D/A 转换后与 V_x 比较，由此决定本位 Q_0 = 1 或 0 并存入寄存器，此时 N 位寄存器的值便是 A/D 转换结果，可从三态门读出。

可见，N 位的逐次比较式 A/D 做 N 次比较即可完成 A/D 转换，有时可能因控制需要还要多一个脉冲，这样一个 N 位的逐次比较式 A/D 转换时间最多为 N + 1 个时钟脉冲。逐次比较式 A/D 的常用控制信号为 转换开始信号（START）转换结束信号（EOC）时钟（f_c）参考电压（V_{REF}），如内置则不需要 读信号（OE）片选（CE）等等。

三、一些逐次比较式 A/D 与 μP 的接口

（一）ADC0809 与 μP 的接口

AD0809 是 8 位 A/D，28 脚封装，100μs 转换时间，输出带三态锁存器，有 8 路（IN0 ~ IN7）模拟量输入。选择通道由 ALE 上升沿锁定 C、B 和 A 端的值来决定，START 的上升沿用来复位内部寄存器下降沿用来启动 A/D 转换，R_- 和 R_+ 用来为内部参考电压产生电路提供电源，EOC 为转换结束标志高电平有效，OE 用来打开三态门读取数据。

图 2.26 为一种 ADC0809 与 μP 的接口电路图。完成 A/D 转换常用两种方法实现，其一为查询方式，其二为中断方式。

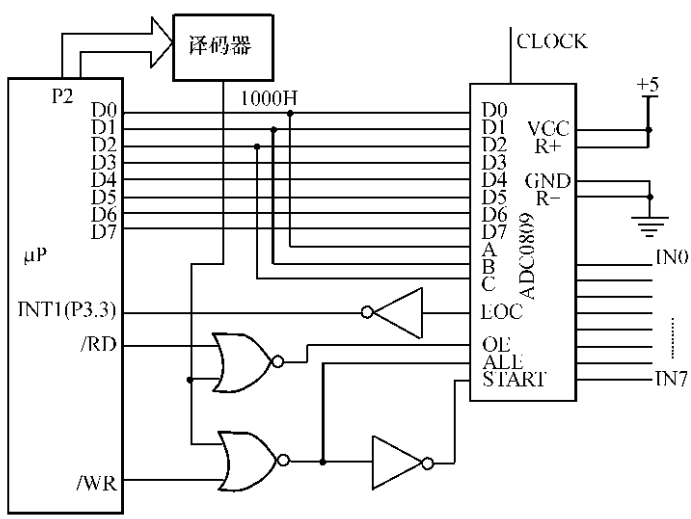


图 2.26　AD0809 与 μP 的接口

1. 查询方式

设读写地址由 8031 的 P2 口产生，译码地址为 1000H，输入通道选择 IN0，转换结果存放在单片机内部 RAM 的 30H 地址，查询的程序为：

```
MOV DPTR,#01000H          ;地址译码
MOV A,#00H
```

```
        MOVX @ DPTR,A          ;启动 A/D
        CALL DELAY             ;延时
WAIT:   JB P3.3,WAIT           ;判断是否转换完毕
        MOVX A,@DPTR           ;读入数据
        MOV 30H,A              ;存入缓冲器
```

2. **中断方式**

用中断方式完成查询方式中同样功能的程序如下。

主程序：

```
MAIN:   SETB IT1               ;设置中断
        SETB EX1
        SETB EA
        MOV DPTR, #01000H      ;地址译码
        MOV A,#0
        MOVX @DPTR,A           ;启动 AD0809
        ......
```

中断服务程序：

```
INTR1:  PUSH DPL               ;压堆栈
        PUSH DPH
        PUSH A
        MOV DPTR,#1000H        ;读入数据
        MOVX A,@DPTR
        MOV 30H,A
        MOV A,#0               ;启动下次 A/D 转换
        MOVX @DPTR,A
        POP A                  ;弹出堆栈
        POP DPH
        POP DPL
        RETI
```

这里的中断服务程序除了读取 A/D 值外还启动了下次转换。

（二） AD574 **及其接口**

AD574 **是** 12 **位逐次比较式** A/D，**转换时间** 25μs，**输出带三态锁存，双极性输入，**28 **脚封装，主要管脚有：**

$\overline{CS}$**——片选信号，低有效；**

CE**——使能信号，高有效；**

R/$\overline{C}$**——读/启动信号，高读取转换数据，低启动转换；**

12/$\overline{8}$**——输出数据长度控制信号，高** 12 **位转换，低** 8 **位转换；**

A_0**——当** R/$\overline{C}$ **为低时，**A_0 **为高启动** 8 **位** A/D **转换，**A_0 **为低启动** 12 **位转换；当** R/$\overline{C}$ **为高时，**A_0 **为高输出低** 4 **位数据，**A_0 **为低输出高** 8 **位数据。**

STS**——工作状态信号，高表示正在运转，低表示结束；**

REFin——基准电压输入端；

REFout——基准电压准输出端；

BIPOFF——单极性补偿电压输入端；

DB_{11} ~ DB_0——12 位数据读出线；

10Vin——10V 输入端；

20Vin - 20V 输入端。

AD574 可以单电源使用也可以双电源使用，双电源较为常用，图 2.27 为一种与 μP 的接口电路。请读者自行编制其转换程序。

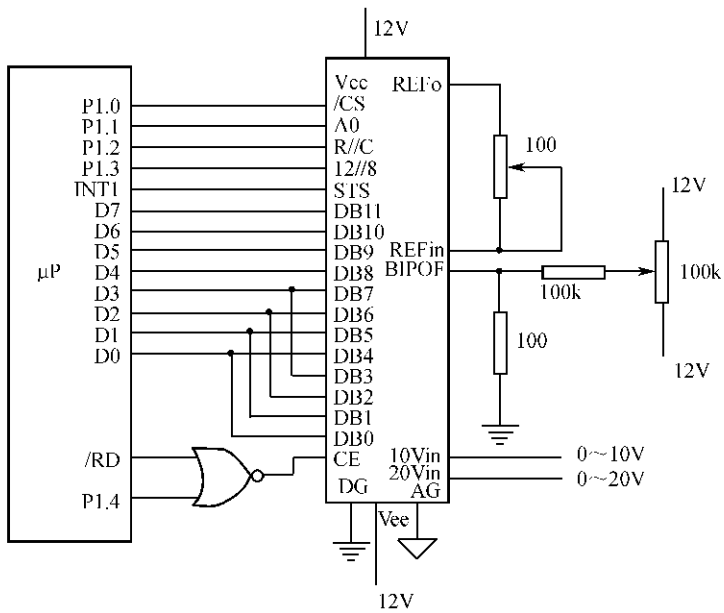


图 2.27　AD574 与 μP 的接口

（三）逐次比较式 A/D 的缺点

逐次比较式 A/D 在转换期间 V_{in}的值不可发生变化，否则将出现转换错误，因而逐次比较式 A/D 抗干扰能力较差，所以在 A/D 的前面一般要加一个采样保持器锁定电压。

2.2.3　双积分式 A/D

一、双积分式 A/D 基本原理

双积分式 A/D 基本原理见图 2.28，假设被转电压 $U_i > 0$，参考电压为 $-U_R$，启动信号 START 上升沿通过单稳产生脉冲，这个脉冲产生 4 个作用：对计数器清 0、进位值 Cy 也为 0、模拟开关 S1 选通 U_i、瞬间接通模拟开关 S2 使电容 C 放电，然后反向积分开始（称为第 1 次积分），由于 $U_i > 0$，$U_1 < 0$，$U_C > 0$，与门打开，CLOCK 可进入计数器开始计数。当计数器计满时发生溢出 Cy 变为 1，模拟开关 S1 接通 $-U_R$，此时 $Q_{n-1}\cdots Q_0$ 又从 0 开始计数，而积分器开始正向相积分（称为第 2 次积分），当 U_1 上升到略大于 0 时 U_C 变低停止计数，此刻的计数器的值即为 A/D 转换值。这里发生了两次积分过程，称为双积分式 A/D，积分器的输出 U_1 (t) 的波形如图 2.29 所示。

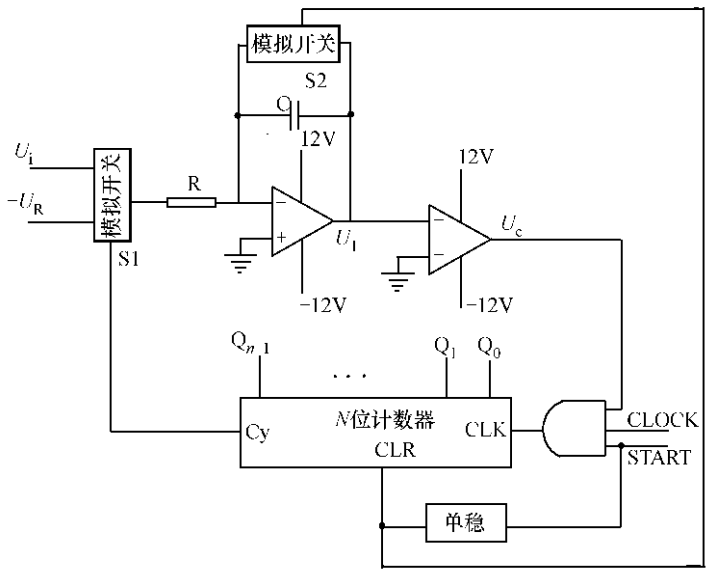


图 2.28　双积分式 A/D 基本原理

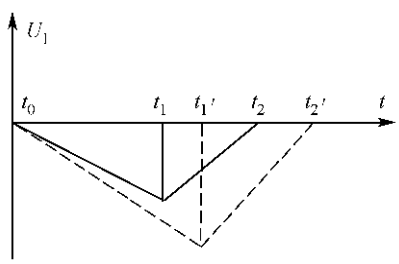


图 2.29　U_1 的变化过程

设 U_i 在某一时间是常数，则第 1 次积分为

$$U_1(t_1) = -\frac{1}{RC}\int_0^{t_1} U_I dt = -\frac{U_I}{RC}T_1$$

$$T_1 = 2^n T_C$$

$$T_C = \frac{1}{f_c}$$

第 2 次积分为

$$U_1(t_2) = U_1(t_1) + \left[-\frac{1}{RC}\int_{t_1}^{t_2}(-U_R)\,dt\right] = U_1(t_1) + \frac{U_R}{RC}T_2$$

令 $U_1(t_2)=0$，约去 RC 因子得

$$U_I T_1 = U_R T_2, T_2 = DT_C$$

即

$$D = \frac{U_I}{U_R}2^n$$

可见 D 只与 U_R 和 U_1 有关系，与 RC 无关，这一点正是我们所希望的。当 $U_i = U_R$ 时，D 为最大值，但 U_i 超过 U_R 时发生溢出。

双积分 A/D 在积分期间如果 U_i 有干扰，由于干扰一般是对称的，积分器的输出将取其平均值从而起到了滤波的作用，所以它虽然比较慢但抗干扰能力强，在实际应用中发挥了巨大的作用。

以上电路实现了 $U_i > 0$ 的 A/D 转换，如 $U_1 < 0$，可换用正的 U_R。实际处理时往往先变输入信号为单边信号再进行转换。

二、μP 控制的双积分 A/D

在智能检测系统中可用极少的硬件便可实现双积分 A/D 转换。一般利用 μP 定时器配合积分器、比较器和模拟开关来完成，由于大多数微机自带比较器使得电路进一步简化。这样的双积分 A/D 又称智能双积分 A/D。智能双积分 A/D 还可使软件计数永不发生溢出，这样可使转换精度进一步提高。常见的较有效的智能双积分 A/D 方法有两种。

(1) 方法 1

图 2.30 电路是这种智能双积分 A/D 的思路的一个例子，用 $P_{1.0}$ 和 $P_{1.1}$ 分别控制模拟开关 S_1 和 S_2，完全可以模拟图 2.28 电路的功能，而用 $P_{3.3}$ 判断比较器的状态，计数器用内部 16 位定时器 T1。工作过程如下。

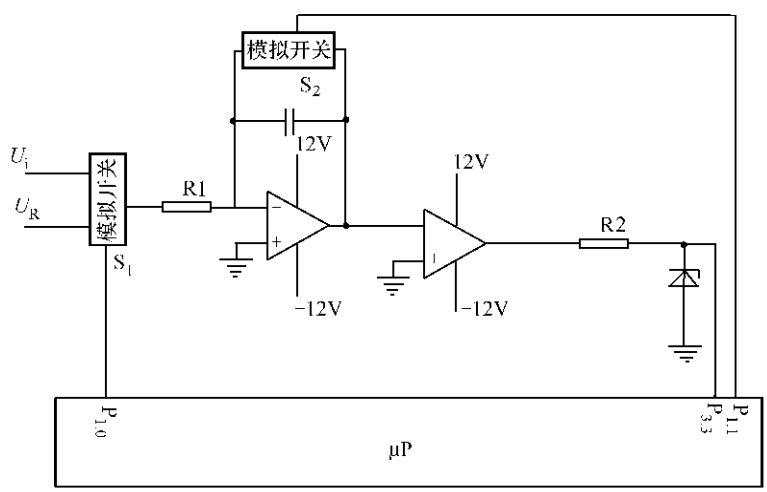


图 2.30　智能式双积分式 AD 方法 1

令 $P_{1.1} = 1$，给电容放电，令 $P_{1.0} = 0$ 选积分器输入为 U_i，反向积分开始，因积分器输出为负值，此时 $P_{3.3} = 1$，启动定时器 T1 在 T1 溢出时，令 $P_{1.0} = 1$ 选积分器输入为 $-U_R$，正向积分开始 当 $P_{3.3}$ 变低时，读取计数器 T1 的值及软件计数器的值合成 A/D 转换值。这里的软件计数器是用来记录定时器的溢出次数的，因为在 $U_x > U_R$ 时，在第 2 次积分时将会发生计数器溢出，设置软件计数器后不但放宽了输入电压的范围，而且可得到大于 16 位的 A/D 转换值，从而使分辨力也大大提高。

⑵ 方法 2

图 2.31 电路是另一种智能双积分 A/D 的思路的一个例子。设 $U_x>0$，$U_R>0$，模拟开关 S1 用来选择输入信号，模拟开关 S2 用来为电容快速放电，电容放电后以 U_x 为输入，

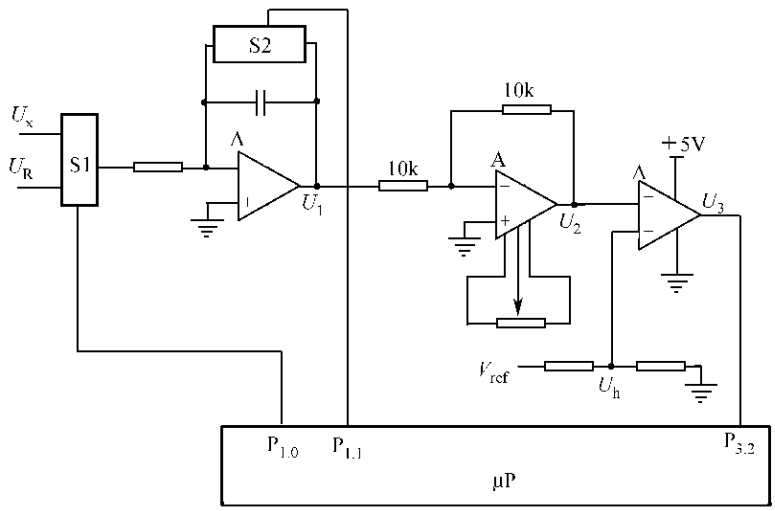


图 2.31　智能式双积分式 A/D 方法 2

积分到比较器翻转时读取计数器的值；电容再放电以 U_R 为输入，积分到比较器翻转时读取计数器的值，用两次的计数值和参考电压解算出 A/D 值。设积分电阻为 R，电容为 C，积分器后面的反相器输出为 U_2，两次积分的时间分别为 T_1 和 T_2，两次积分的计数值分别为 DT_1 和 DT_2，时钟周期为 T_c，则

第 1 次积分：

$$U_2=\frac{1}{RC}\int_0^{T_1}U_x dt=\frac{U_x}{RC}T_1=\frac{U_x}{RC}T_C DT_1$$

第 2 次积分：

$$U_2=\frac{1}{RC}\int_0^{T_2}U_R dt=\frac{U_R}{RC}T_2=\frac{U_R}{RC}T_C DT_2$$

因为 $U_2=U_2=U_h$，所以　$U_x DT_1=U_R DT_2$，即 $U_x=\dfrac{DT_2}{DT_1}U_R$

可见，只要已知道 U_R，就可求得 U_x，其值同样与 RC 无关。

2.2.4　高速 A/D

一、并行比较式 A/D

并行比较式 A/D 用多个比较器同时进行判断输入量的大小，一次性决定 A/D 输出的所有位数，因而具有很高的转换速度。图 2.32 为 3 位并行比较式高速 A/D 的内部结构示意。作为 3 位 A/D 应有 $2^3=8$ 个台阶的数字输出，为此将参考电压 V_{REF} 分为 8 份，共用 8 个电阻，首尾电阻阻值为 R，其余阻值为 2R，这样参考电压也分为 8 个台阶，如表 2.1 所列，这样得到的 8 种参考电压见表 2.1。输入电压在给定的范围内对应的比较器有不

同的组合状态，经编码后输出 000～111 的二进制码。其中编码器可用任意电路，如 PAL，GAL 等。整个过程由一个时钟控制，时钟的前半周用来采样比较结果，后半周用来编码并锁存结果，这种 A/D 转换方式速度极快，转换时间仅用了一个时钟周期，一般只需几个纳秒。但用的比较器比较多，如 8 位 A/D 需要 255 个比较器，N 位 A/D 需要 2^{N-1} 个比较器。这一类 A/D 又称为闪烁型 A/D。常见的芯片有 CA3308（转换频率 15MHz）、TLC5510（转换频率 20MHz）、TLC5540（转换频率 40MHz）等。

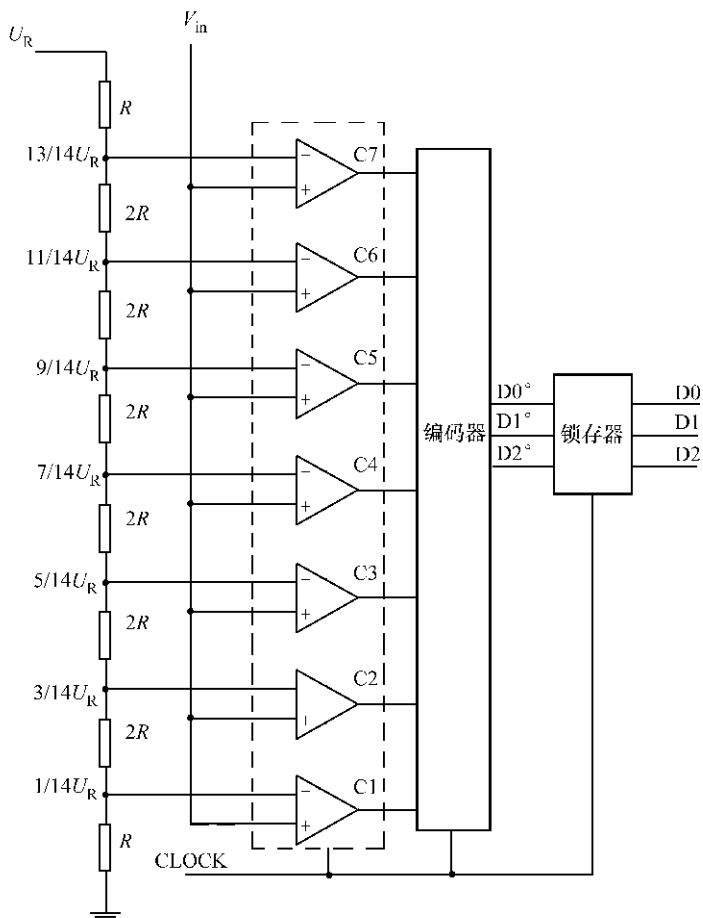


图 2.32　并行比较式 A/D 的基本原理

表 2.1　并行比较式 A/D 的参考电压

V_{in}	比较器输出							D2	D1	D0
	C7	C6	C5	C4	C3	C2	C1			
$\left(0 \sim \frac{1}{14}\right)U_R$	0	0	0	0	0	0	0	0	0	0
$\left(\frac{1}{14} \sim \frac{3}{14}\right)U_R$	0	0	0	0	0	0	1	0	0	1

$\left(\frac{3}{14}\sim\frac{5}{14}\right)U_R$	0	0	0	0	0	1	1	0	1	0
$\left(\frac{5}{14}\sim\frac{7}{14}\right)U_R$	0	0	0	0	1	1	1	0	1	1

（续）

V_{in}	比较器输出							D2	D1	D0
	C7	C6	C5	C4	C3	C2	C1			
$\left(\frac{7}{14}\sim\frac{9}{14}\right)U_R$	0	0	0	1	1	1	1	1	0	0
$\left(\frac{9}{14}\sim\frac{11}{14}\right)U_R$	0	0	1	1	1	1	1	1	0	1
$\left(\frac{11}{14}\sim\frac{13}{14}\right)U_R$	0	1	1	1	1	1		1	1	0

二、高速 A/D 与 μP 的接口

（一）高速 A/D 与 DSP 接口

DSP 可直接与速度相当的高速 A/D 相连，如图 2.33 为某一 DSP 与 CA3308 的接口电路，有关 CA3308 管脚功能请查阅相关资料。除了 DSP 外一些高速单片机也可采用类似的方法设计高速 A/D 接口，如 C8051F020 的指令周期为单时钟周期，主频可达 25MHz，可以直接控制同级别的高速 A/D。

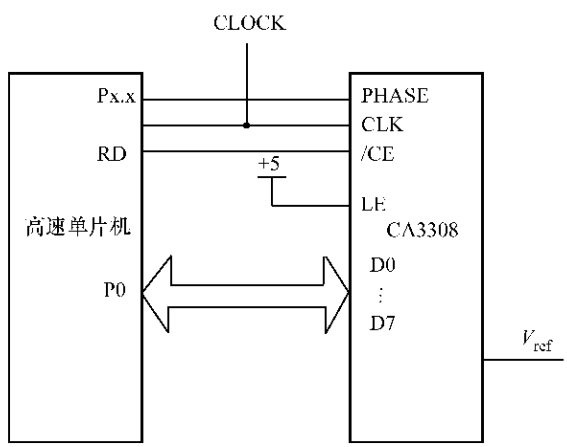


图 2.33　高速 A/D 与 DSP 的接口

（二）高速 A/D 与一般 μP 接口

⑴ 方案 1：用普通 RAM 作缓冲器

一般的低速单片机控制高速 A/D 须配合其他电路实现，图 2.34 为一种电路方案。图中 RAM 用来存取 A/D 的转换数据，地址计数器用来产生 RAM 的读写地址，时钟有高速和低速两种，低速时钟由 P1.1 产生，高速时钟为 HCLK，HCLK 不是直接对 A/D 控制而是通过控制电路产生相关的一组控制信号，控制电路的输入为 HCLK、P1.4 和地址计数器的溢出 OE，控制电路的输出包括 A/D 时钟及读信号、地址计数器的时钟信号及清零信号、RAM 的写信号等。如图所示，CLK 信号控制高速采集，采集开始地址值为 0，RAM 值采满后，地址计数器溢出信号 OE 封锁时钟 CLK 同时通知 μP 采集完毕地址也变为 0，μP

用 P1.1、P1.2、P1.3 及$\overline{RD}$信号从 RAM 中读取数据。

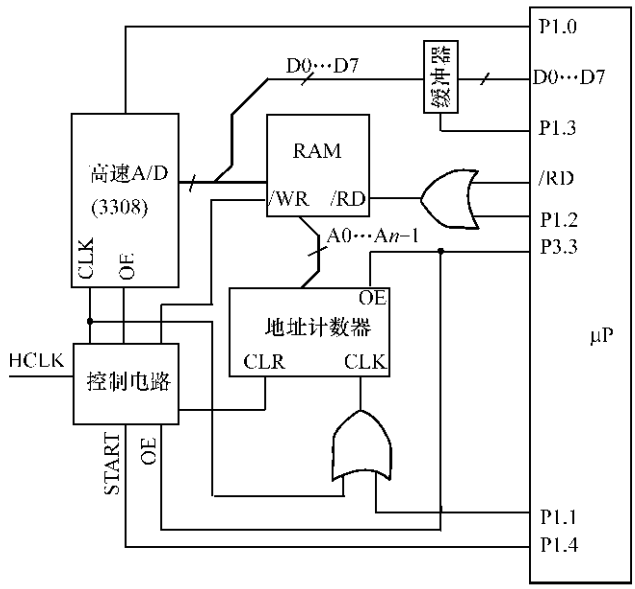


图 2.34　μP 与高速 A/D 接口方案 1

(2) 方案 2：用双 RAM 作缓冲器

本方案实质上还是方案 1 的思路，只是 RAM 改用双口 RAM，如图 2.35 所示，由于双口 RAM 具有独立的两套存取控制信号，可使电路简化一些。图中控制电路及地址发生器产生 AD 的时钟、读信号、RAM 的地址及写信号，当规定的一组数据采集完后还能产生采集完毕的 Finish 信号通知 μP。μP 可以等到 Finish 后用自己独立的地址线、数据线、读写控制线对 RAM 操作，也可以不等 Finish 到来随时对 RAM 操作，这一点方案 1 是不可比拟的。

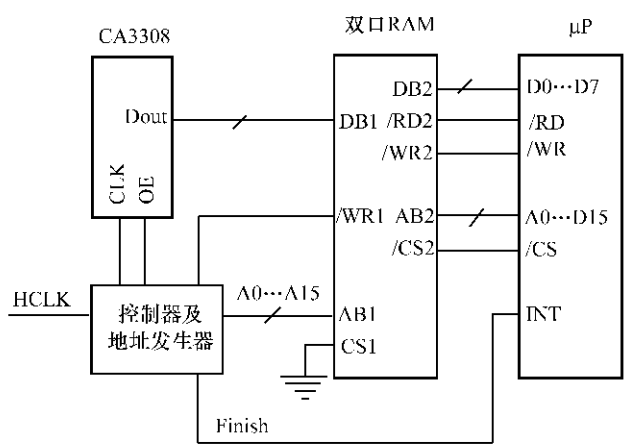


图 2.35　高速 A/D 接口方案 2

(3) 方案 3 :用双 FIFO 作缓冲器

FIFO (First In First Out) 是一种具有先进先出功能的堆栈式 RAM,FIFO 设有数据写入线、数据读出线、读写线、"读空 (NULL) "线、"写满 (FULL) "线等控制线,但 FIFO 没有地址线,每进行一次读或写的操作,FIFO 的内部地址会自动加 1,末地址操作完后会自动回到起始地址。用 FIFO 作缓冲器组成的 μP 与高速 A/D 的接口电路方案如图 2.36 所示。设置一个简单的控制电路,其输入为高速时钟 HCLK、NULL 和 FULL,其输出为控制 A/D 的 CLK、A/D 的读信号和 FIFO 的写信号,当数据采满后 FULL 一方面封死控制电路不再产生相关信号,另一方面,由 P1.0 通知 μP 读取采样值,当 μP 读完采样值后 NULL 一方面由 P1.0 通知 μP,另一方面启动控制电路产生相应信号采集下一组数据。

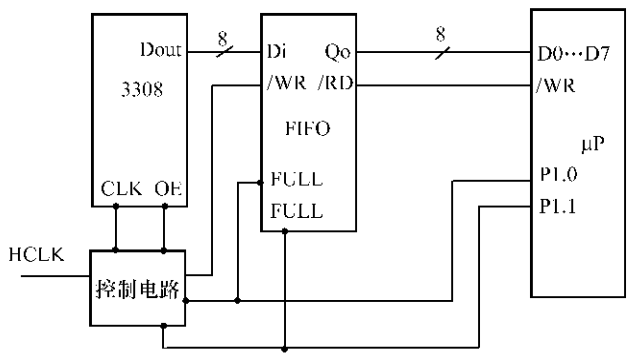


图 2.36 高速 A/D 接口方案 3

相比之下,本方案更为简单合理,目前生成的 FIFO 其 NULL 和 FULL 的控制点都是可编程的因而控制更为灵活,例如 IDT 公司的 IDT72V2X1 系列产品都具有此功能。

2.3 数据采集系统

2.3.1 DAS 概述

数据采集系统简称 DAS (Data Acquisition System),其基本结构如图 2.37 所示,各部分分述如下。

(一) 程控放大器 (PGA)

程控放大器可用多种方式,如分路切换式、改变电阻式、D/A 变换方式等,这部分在以后的章节详述。

(二) 抗混叠滤波器 (AF)

抗混叠滤波器一般用低通滤波器,可用固定特性或程控特性的滤波器,在要求不高的情况可以不用。

(三) 采样定理

设被采集的有用信号的最高频率为 f_x,采样频率为 f_s,我们来看下面的几种情况 如图 2.38 所示,当 $f_s = f_x$ 时,采到的数据只能描述一条直线,信号的幅值和频率都失真了;当 $f_s = 2f_x$ 时,采到的数据就可描述与输入信号同频的三角波,信号的频率得以描述,只是

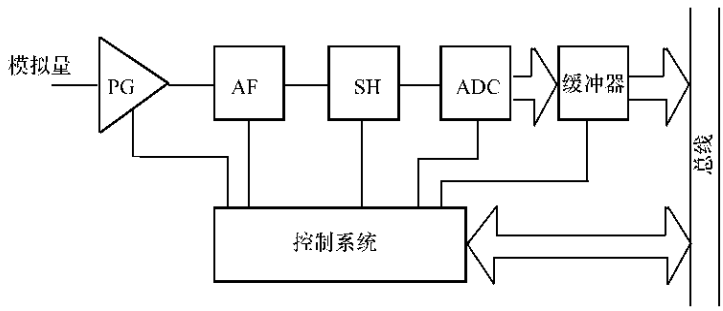


图 2.37　DAS 基本结构

幅值信号随采样点的变化有不同程度的失真 理论和实践证明，要完整地保存信号的所有信息，f_s 必须是 f_x 的 2 倍以上，这就是著名的采样定理。理论上 f_s 越大越好，在具体确定采样频率应遵循一下原则：

⑴ 在追求测试精度时根据所处理的有用信号的最高频率选择匹配的硬件，采样较高的采样频率一般要求 $f_s > 10f_x$。

⑵ 如果系统要求通用性好具有较高的带宽，应选择高速的 A/D 及其它控制电路，采样频率不宜太高，在满足采样定理的前提下可分段程控。

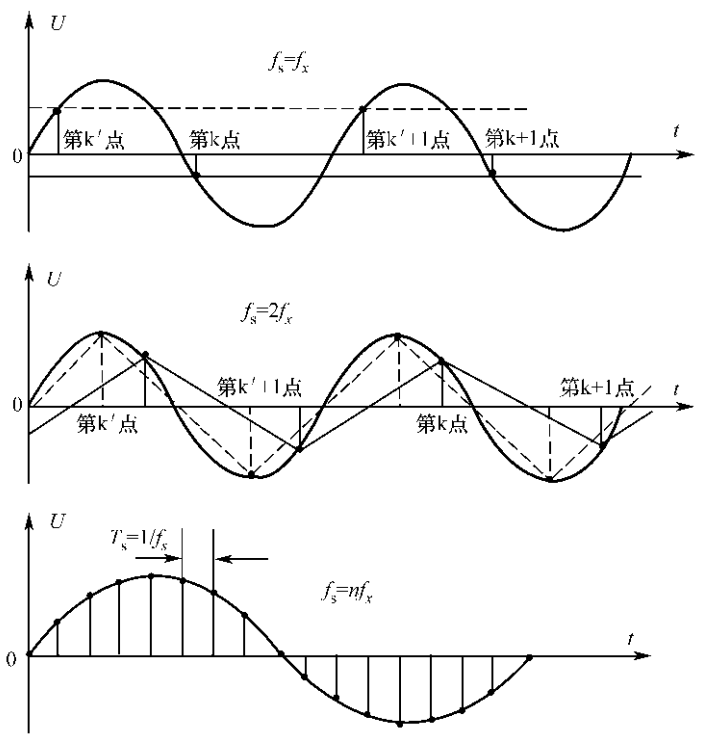


图 2.38　f_x 和 f_s 的关系描述

（四）采样保持器（S/H）

采样保持器是将输入信号瞬时值捕捉下来并保持一定时间的装置，图 2.39 为其基本

原理图。控制信号 S/H = 1，模拟开关合上，电容 C 快速充电到 V_x 的值；S/H = 0，模拟开关断开，C 无处放电，保持 V_x 的值。双积分式 A/D 可不用 S/H，但逐次比较型 A/D 必须用 S/H 否则，会带来较大的误差。采样保持器主要指标有：

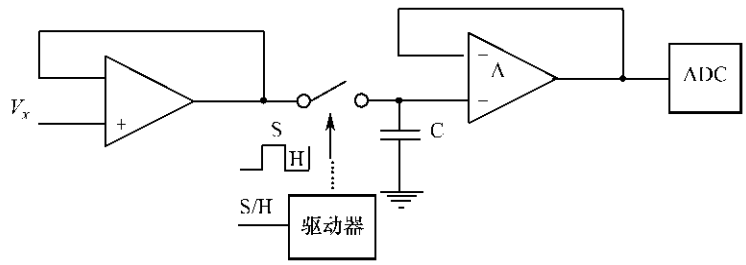


图 2.39　采样保持器原理

(1) 捕捉时间 (Acquisition Time)，主要是对 C 的充放电使 C 上电压充到 V_x 所需时间；

(2) 孔径时间 (Aperture Time)，指电压到达后开关断开的实际时间；

(3) 孔径抖动 (Aperture Jitter)，在开关断开时，C 上值的不稳定值过渡时间；

(4) 保持建立时间 (Hold Mode Setting Time)；

(5) 衰减率 Drop Rate；

(6) 传导误差 Feedthrough。

在设计系统时应根据相关技术指标精心选择合适的 S/H。

2.3.2　同步多路采集系统

DAS 往往同时处理多路信号，最常见的有 8 路、16 路、32 路等。同步采样是指各路信号的瞬时值同时采到。

同步采集系统的方式之一如图 2.40 所示，各通道都是独立的，所有的采样保持器和 A/D 转换器都是同步进行的，各通道的 PGA 和 AF 可以统一控制，亦可独立控制，根据具体情况而定。这种方式的采集系统的采集速度快，但成本高、体积大、功耗也大。

同步采集系统的方式之二如图 2.41 所示，这种方式各通道都设有采样保持器，所有采样保持器用同一信号控制，但用同一个 A/D 通过多路模拟开关进行转换。当忽略采样保持器对各路信号的衰减时可认为是同步采样，其速度比方式一要慢得多，但它省去了大量的 A/D，使系统成本大大降低。

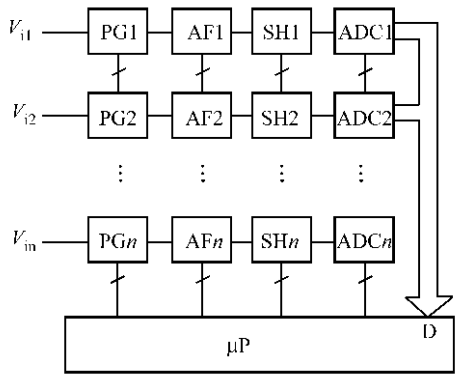


图 2.40　多 A/D 同步采集系统

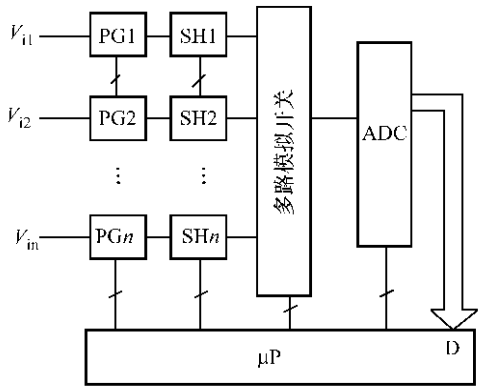


图 2.41　单 A/D 同步采集系统

2.3.3 异步多路采集系统

当各路输入信号的瞬时值不是同时得到时称之为异步多路采集系统，这种方式往往用多路模拟开关选择要进行 A/D 转换的通道，各通道是分时进行采样的。常见的异步多路采集系统如图 2.42 所示。

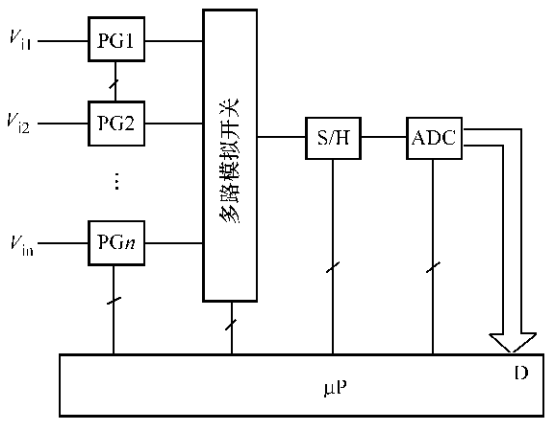


图 2.42 异步多路采集系统

2.3.4 多点巡回数据采集系统

多点巡回数据采集系统是指对多路信号按顺序进行循环采集的系统，图 2.43 为一种常见的 16 路多点巡回数据采集系统的电路方案，巡回数据采集系统从本质上讲为异步数

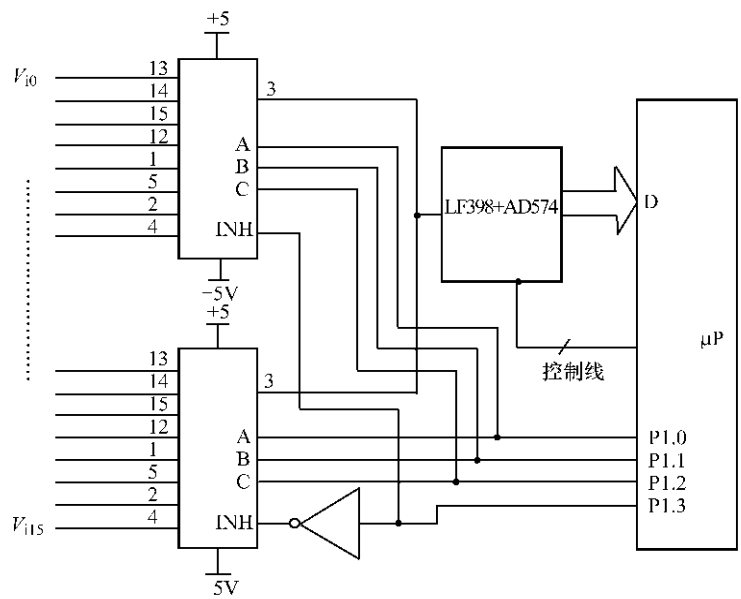


图 2.43 一种多点巡回异步采集系统

据采集系统，但当系统处理的信号频率不是很高且系统 A/D 转换速度较快时，可以在工程上满足同步采样的要求，因此这种采集系统用途比较广泛。

用 P1.3 ~ P1.0 来控制 16 路模拟开关，表 2.2 为选通模拟通道 0 ~ 15 的时对应的 P1.3 ~ P1.0 的输出值，本例中 P1.3 ~ P1.0 的值正好也是 0 ~ 15。

图 2.44 为数据采集的流程图，设 delay ( ) 为现有的延时子程序，samp (ch) 为现有的采样及 A/D 转换子程序，其中 ch 为表示通断号的变量，则用 C51 编写的完整的巡回采样程序如下：

```
unsigned int n = 1000 ;                    //设采样点数为 1000
. . . . . .
void main (void)                           //主程序
 {
unsigned int i , j ;
for  (j = 0 ;j < n ;j + + )
      {
     for  (i = 0 ;i < 16 ;i + + )
           {
           samp (i) ;                      //采样一个值
           }
     delay ( ) ;
     }
}
```

表 2.2

通道号	P1.3 ~ P1.0 (BIN)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

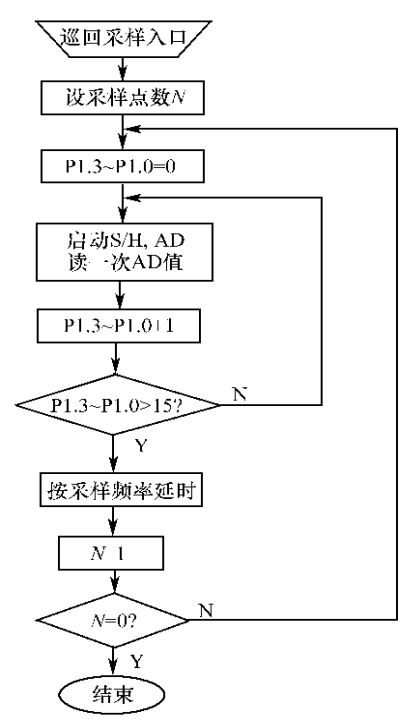


图 2.44　巡回采样流程

2.4 开关量输入输出通道

2.4.1 开关量输入通道

在智能检测系统中经常要处理一些开关量信号，所谓开关量信号是指只有两个状态的信息。从电信号的角度来看，开关量信号可以是具有一个高电平、一个低电平两个状态的信号。从工程上来说，开关量往往指一个机械触点开或闭的两种状态。

开关量输入通道就是将触点信号变为一个可以被检测系统识别的电平信号。图2.45为一个四路开关量输入通道，以开关量 K1 为例，当 K1 断开时，P1.0 = 1；当 K1 闭合时 P1.0 = 0。由此可以识别开关的状态。解决同样的问题亦可用图 2.46 电路来实现，但开关开闭得到的 P1.3 ~ P1.0 的电平正好与 2.45 电路得到的结果相反。

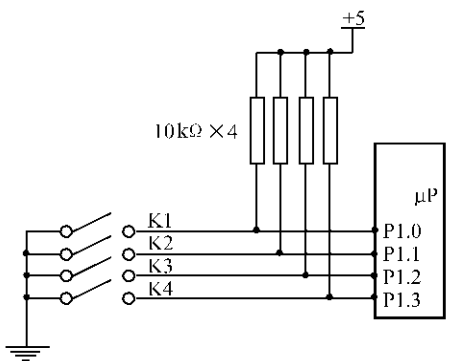


图 2.45 负跳变开关量接入方式

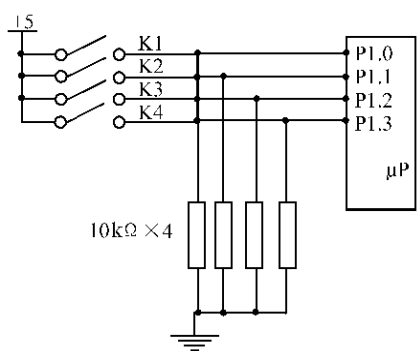


图 2.46 正跳变开关量接入方式

图 2.45 和 2.46 电路只适应于开关连线上没有较大干扰的情况。在工业现场等场合开关连线上有可能有较大的干扰。这种干扰很容易毁坏检测系统，因而大部分检测系统的开关量输入通道采用了光电隔离装置。图 2.47 为一种带光电隔离装置的 4 路开关量输入电路，它使现场开关信息和 μP 在电路上隔离，不但保证了系统的安全性，同时提高

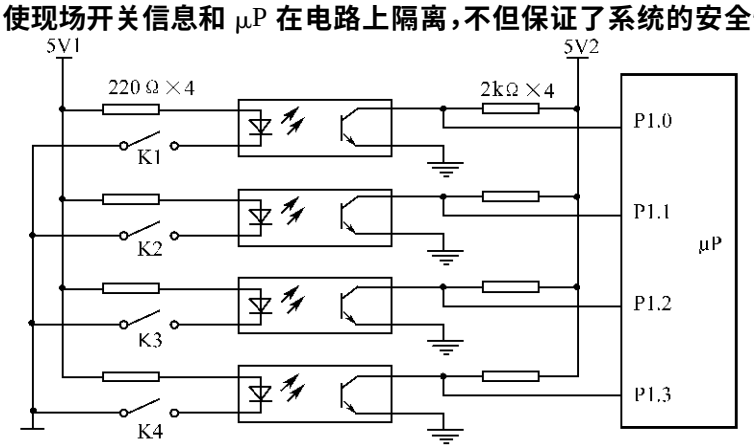


图 2.47 带光电隔离的开关量接入方式

了系统的抗干扰性。要注意的是:这里的电源 5V1 和 5V2 不可共用同一个参考地,否则光电隔离就不起作用了。

2.4.2　开关量输出通道

开关量输出通道是指由检测系统给出的触点,并且要求这个触点具有合适的带载能力。例如控制摄像头云台转动启动的触点,控制大型照明灯光的触点,控制液体压力改变的电磁阀门开闭的触点等等。在现代的智能检测系统中越来越多地要求检测系统具有一定的开关量输出能力,开关量输出一般用继电器或无触点开关来实现。

图 2.48　一种继电器的外形

一、继电器开关量输出通道

继电器是用电磁来控制机械触点开闭的元件,它使触点电路与控制电路完全隔开,具有很广泛的用途,图 2.48 为一种继电器的外形。用继电器构成的开关量输出的典型电路如图 2.49 所示,这是一个具有 4 路开关量输出的电路,其触点带载能力与所选的继电器型号有关,继电器绕组上的二极管 D1 ~ D4 和电容 C1 ~ C4是为了消除绕组产生的反向电压而设的。尽管如此,当绕组电压较大时还会有较大的干扰反馈到三极管 V1 ~ V4 的控制端,严重时会损坏 μP。因此用于工业现场等场合的开关量输出通道往往通过光电隔离后再控制继电器,如图 2.50 所示。这里同样要注意现场 12V 电源不可与检测系统电源共地。

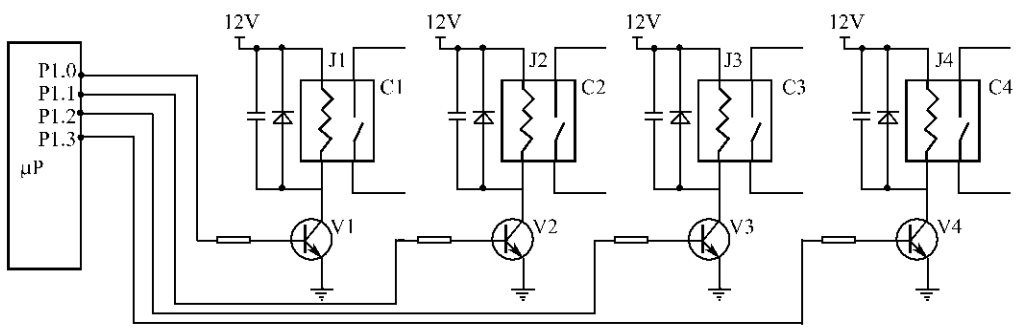


图 2.49　一种用继电器实现的开关量输出电路

二、固态继电器开关量输出通道

固态继电器 (SSR - SOLIDSTATE RELAYS) 是一种全部由固态电子元件组成的新型无触点开关器件,问世于 20 世纪 70 年代,是用开关三极管、可控硅 (晶闸管) 等半导体器件的开关特性制作的,可达到无触点无火花地接通和断开电路的目的,因此又被称为“无触点开关"。图 2.51 为常见的固态继电器外形。

(一) 固态继电器的原理及结构

SSR 按使用场合可以分成交流型和直流型两大类,它们分别用在交流或直流电源上做负载的开关。下面以交流型的 SSR 为例来说明它的工作原理。图 2.52 是它的工作原

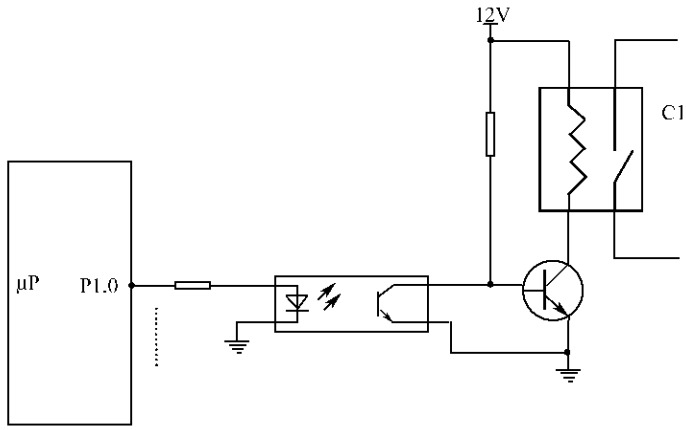


图 2.50　有光电隔离的继电器开关量输出电路

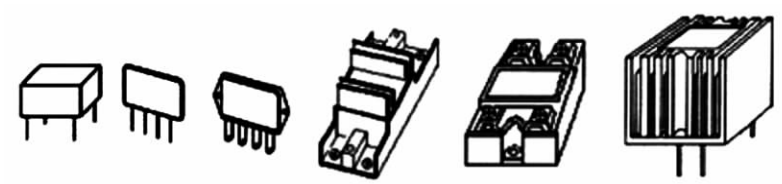


图 2.51　常见的固态继电器外形

理框图，图中的部件①～④构成交流 SSR 的主体，SSR 是一种四端器件，A 和 B 是输入端，C 和 D 是输出端。工作时只要在 A、B 上加上一定的控制信号，就可以控制 C、D 两端

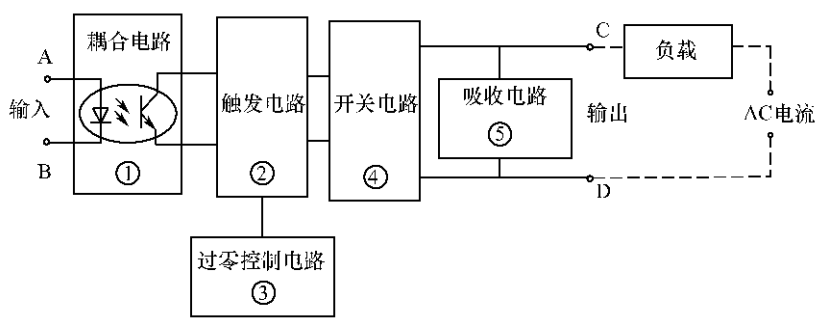


图 2.52　交流型 SSR 工作原理框图

之间的“通"和“断"，实现“开关"的功能，其中光电耦合电路的功能是为 A、B 端输入的控制信号提供一个输入/输出端之间的通道，在电气上断开 SSR 中输入端和输出端之间的联系以防止输出端对输入端的影响。触发电路的功能是产生合乎要求的触发信号，驱动开关电路④工作。开关电路一般用双向可控硅来实现。为了防止开关管产生射频干扰并以高次谐波或尖峰等污染电网，并且使开关电路导通的瞬间电流不至于太大而损坏开关管，特设过零控制电路，使触发信号是在交流负载的过零点控制开关管的通断。吸收电路是为防止从电源中传来的尖峰、浪涌电压对开关器件的冲击和干扰而设的。

直流型的 SSR 与交流型的 SSR 相比大致相同，直流型的 SSR 无过零控制电路，开关器件一般用大功率开关三极管，这里不再赘述。

（二）固态继电器的特点

SSR 成功地实现了弱信号对强电的控制。由于光耦合器的应用，使控制信号所需的功率极低，而且固态继电器所需的工作电平与 TTL、HTL、CMOS 等常用集成电路兼容，可以实现直接联接。这使 SSR 在数控和自控设备等方面得到广泛应用，在相当程度上可取代传统的继电器。

SSR 由于是全固态电子元件组成，它没有任何可动的机械部件即可实现“通"和“断"的开关功能，具有以下特点：

⑴ 没有电接触点，所以它工作高可靠、长寿命、无动作噪声；

⑵ 耐振耐机械冲击，安装位置无限制；

⑶ 很容易用绝缘防水材料灌封做成全密封形式，具有良好的防潮、防霉、防腐性能；

⑷ 在防爆和防止臭氧污染方面的性能也极佳。

这些特点使 SSR 在军事、化工、井下采煤和各种工业民用电控设备中得到广泛的应用。

（三）由固态继电器组成的开关量输出举例

如图 2.53 为基本的 SSR 控制的开关量输出电路，为了防止输入电压超过额定值，需设置一限流电阻 R 当负载为非稳定性负载或感性负载时，在输出回路中还应附加一个瞬态抑制电路，通常措施是在 SSR 输出端加装 RC 吸收回路，另一个常用的措施是在 SSR 输出端接入具有特定钳位电压的电压控制器件，如双向稳压二极管或压敏电阻等。

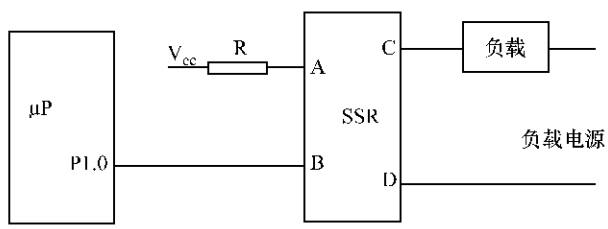


图 2.53　由固态继电器组成的开关量输出举例

第 3 章　人机交流系统

3.1　按键开关的控制

3.1.1　按键开关的接入

按键开关是指操作者用手按动时可改变状态的一类开关，有的按键开关在按下时开关触点由断开状态变为闭合状态，有的正好与之相反，最常见的是前者。有两种方式可使按键的状态变为适合于计算机处理的信号。以常开按键为例，图 3.1 (a) 可使按键产生上跳沿信号，而图 3.1 (b) 可使按键产生下跳沿信号。

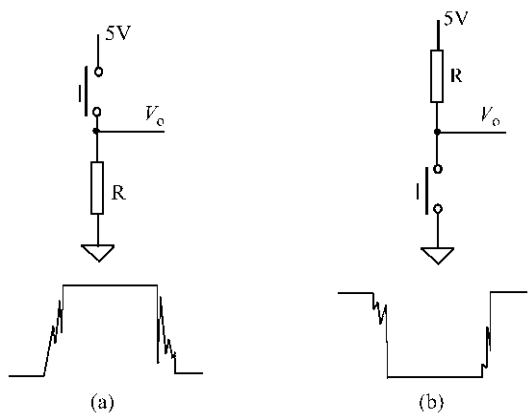


图 3.1　常开按键
(a) 按键产生上跳沿；(b) 按键产生下跳沿。

如图中的波形所示，由于机械结构的固有特性，在开关按下和放开的过程中会产生颤动，因此实际产生的电信的波形也会有颤抖，如不消除这种波形的颤动将会给系统带来误动作。

3.1.2　消除按键颤动的方法

(一) 硬件消颤法

利用单稳电路可消除颤动。图 3.2 (a) 为用单稳 74HC123 消去正脉冲按键颤动电路，只要 RC 选择合适，使单稳宽度大于按键正脉冲宽度即可消除波形的前后沿的颤抖。图 3.2 (b) 为消除负脉冲按键颤动电路。二者同样的原理，仅仅是选择单稳不同的触发端和输出端。

(二) 软件延时法

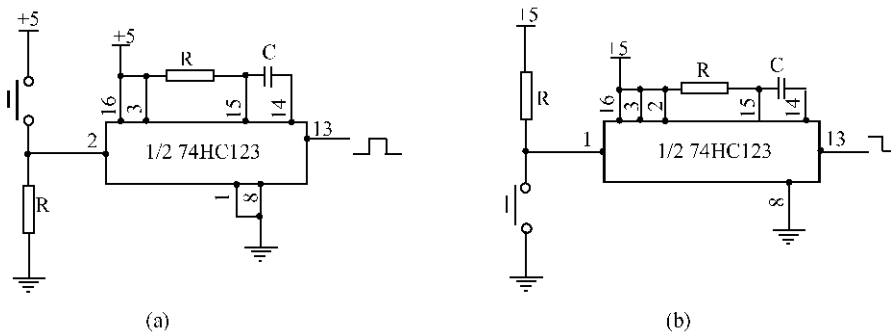


图 3.2　消除按键颤动电路

(a) 消除正跳沿颤抖的电路；(b) 消除负跳沿颤抖的电路。

如图 3.3 (a) 所示，用 P1.0 口通过查询方式扫描按键。当发现端口电平有变化时，通过软件延时来达到消颤目的，目前大多数检测系统都采样软件方式进行消颤。图 3.3 (b) 为查询按键状态的软件流程图，根据经验，软件延时时间一般取 20ms 左右，也可根据实际情况由调试决定。

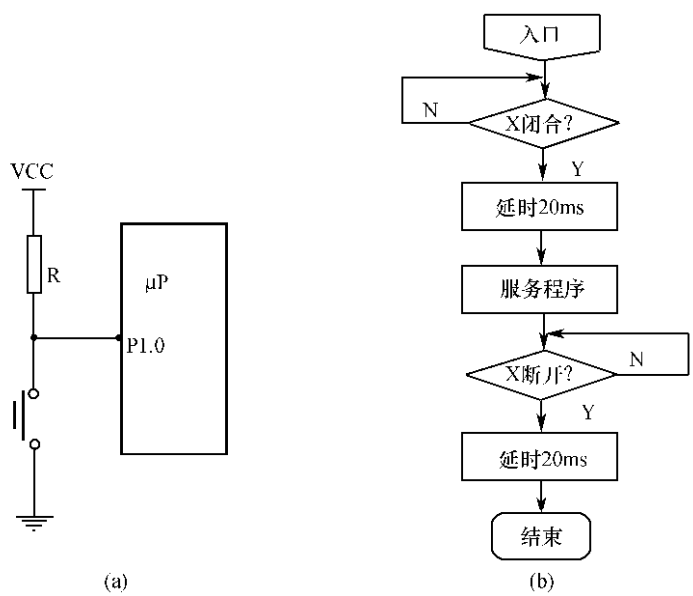


图 3.3　软件延时法消颤原理

(a) 将按键信号接入 μP；(b) 软件延时消除颤抖的流程。

3.1.3　按键信号的软件处理方式

一般情况下按键信号的处理采取查询和中断两种方式。

(一) 直接从端口查询

这种方式的典型电路及程序流程见图 3.3 (a)，设 delay (Tms) 为延时子程序，excu（）为按键要处理的服务子程序，则一种用 C51 编写的按键查询程序如下。

```
sbit AJ = P1 0；          //定义按键为 P1.0
 ......
void main (void)
 {
 ......
 while (AJ)；             //判断按键是否按下
 delay (20)；             //延时 20ms
 excu ()；                //执行相关任务
 while (!AJ)；            //判断按键是否抬起
 delay (20)；
 ......
}
```

(二) 从数据端口查询

这种方式的典型电路如图 3.4 所示，如果是一个按键只用数据线 D0 输入即可。为

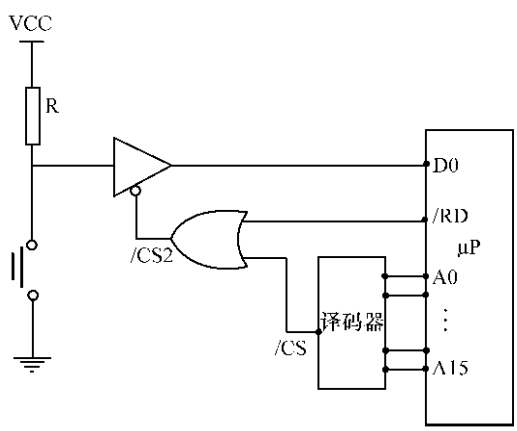


图 3.4　从数据端口查询

了和其它外设复用，数据线需有一个译码器产生选通信号，设译码地址为 1000H，完成(一) 中同样的功能其程序为

```
unsigned char xdata AJ _ at _ 0x1000；      //定义按键绝对地址
      ......

void main (void)
 {
 ......
   while (AJ & 0x01)；                      //判断按键是否按下
   delay (20)；
   excu ()；
   while (!AJ & 0x01)；
```

```
    delay (20) ;
    ......
}
```

(三) 中断方式

将图 3.3 (a) **中的** P1.0 **改用可接受中断的端口如** P3.2 (INT0)**,就可实现用中断方式处理按键功能。中断方式可设为下沿触发,为了消除颤抖,在首次进入中断后先关掉中断,并在中断服务子程序中一直等待按键松开后再退出,其部分程序为**

```
sbit        ZD  =  P3 2 ;          //定义中断输入口为 P3.2

bit         BZ  =  0 ;             //设按键产生标志
.......

void main (void)
 {
 ......
   while (1)
      {
     if (BZ)                       //判断标志
           {
           excut () ;
           bz  =  0 ;              //清标志
           }
      }
}
int0 (void ) interrupt0            //中断服务程序
 {
     EA  =  0 ;                    //关中断 ;
     BZ  =  1 ;                    //置标志 ;
     While (! ZD) ;                // 等待按键松开
     Delay (20) ;                  //延时 20ms
}
```

3.2　其它开关的控制

3.2.1　旋钮开关的接入

旋钮开关常用于检测系统的面板,图 3.5 **为一个单刀四位的旋钮开关与单片机连接电路图。开关的活动点接地,**4 **个固定点通过三态门接入** μP **的数据线,通过译码选通后由** $\overline{RD}$**信号读入,开关位置对应的值见表** 3.1**,**μP **可定时对其扫描来确认当前旋钮开关所**

处的位置，由此决定应完成的功能。一个 N 位的旋钮开关可产生 N 个状态。

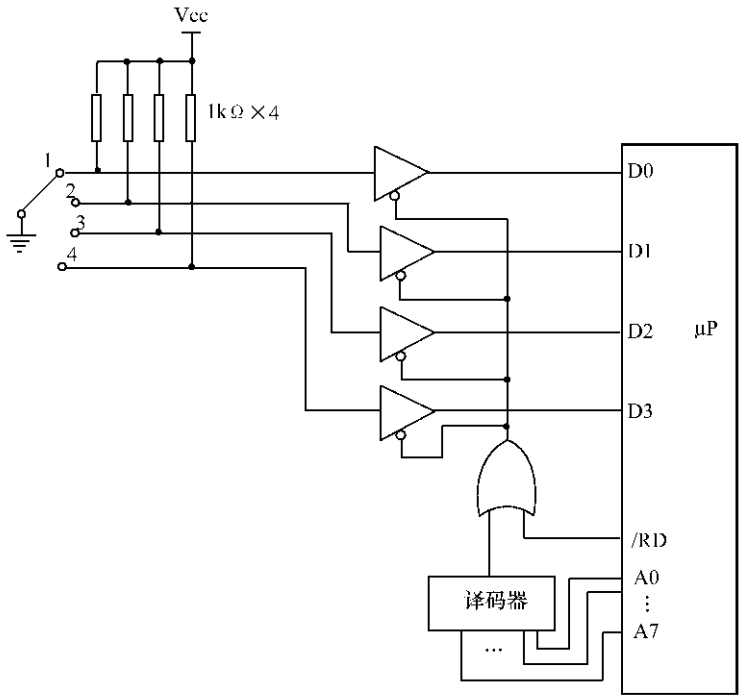


图 3.5　一种单刀 4 位开关的识别电路

表 3.1　旋钮开关的代码

开关位置	开关代码	16 进制	开关位置	开关代码	16 进制
0	1 1 1 0	E	2	1 0 1 1	B
1	1 1 0 1	D	3	0 1 1 1	7

3.2.2　拨码开关的接入

拨码开关是检测系统经常使用的功能能预置部件，如图 3.6 为 8 位拨码开关的识别电路，其码值读入方式与旋扭开关的读入方式相同。

一个 N 位拨码开关可产生 2^N 种组合状态，通常用于检测系统的出厂设置，也可由用户改变设置。

3.2.3　游戏操纵杆类开关的接入

游戏操纵杆是可以自动回到中间位置的多触点开关。图 3.7 为一个具有 4 个触点的操纵杆示意图，将活动点接地，固定点信号引出，其各个位置值与表 3.1 完全相同。因而可用相同的方式来获得手柄的位置。

设译码地址为 7FH，手柄到 0 位置表示某一游戏中目标前进；手柄到 2 位置表示目标后退；手柄到 1 位置表示目标向右；手柄到 3 位置表示目标向左。图 3.8 为实现某一游戏的程序流程。

3.3　矩 阵 键 盘

在 3.1 节中介绍了如何将几个按键接入系统。在智能检测系统中有时需多个按键来完成人机交流，当按键较多时一般采用矩阵的形式来组成键盘。

设计一个矩阵键盘与 μP 的接口，应解决以下问题：

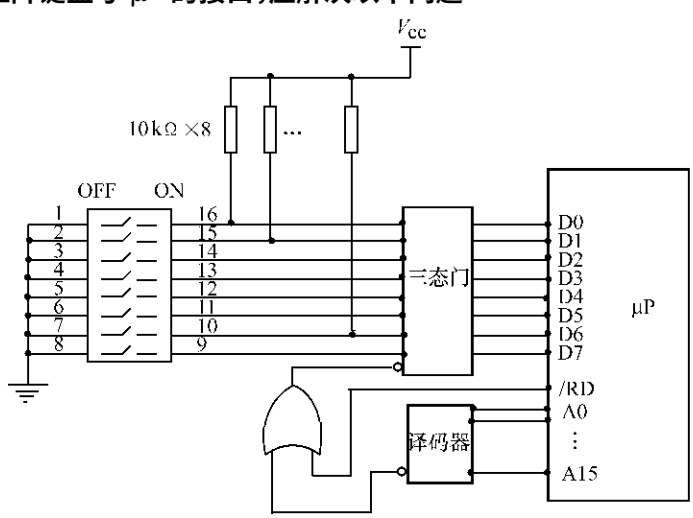


图 3.6　一种 8 位拨码开关识别电路

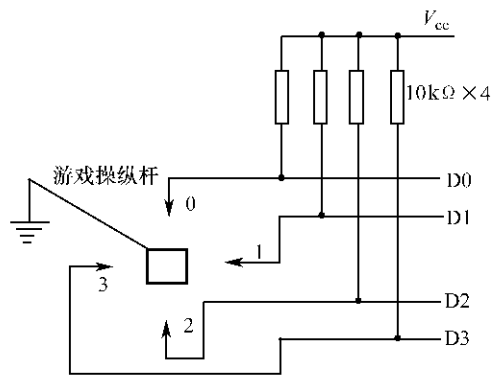


图 3.7　游戏杆类信息识别电路

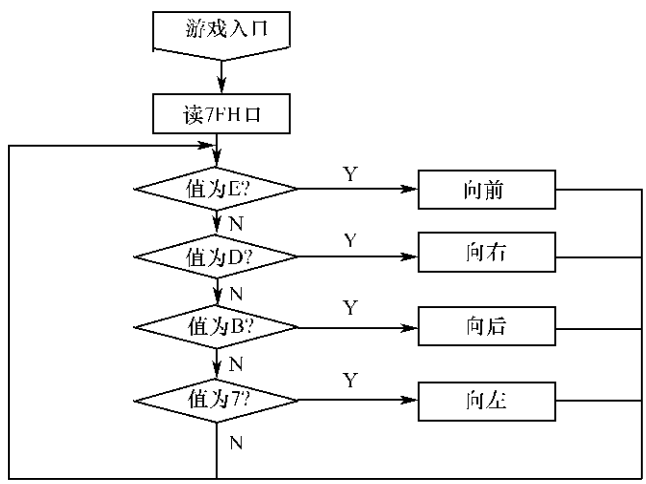


图 3.8　一种简单游戏的软件流程图

⑴ 是否有键按下？

⑵ 如果有键按下是哪一个键按下？也就是要确定键值；

⑶ 消除按键颤动；

⑷ 不管按键按多长时间仅取一次值；

⑸ 同时有一个以上键按下，做相应处理。

μP 完成上述任务的过程称为键盘扫描，扫描方式分为双口扫描方式和数据口扫描方式两种。

3.3.1　双口扫描方式

一、查询式双口扫描

双口扫描是指用两组不同的端口来完成键盘的扫描。以 4×4 键盘扫描为例。一种常见的电路如图 3.9 所示，将 P2 口设定为输出口，P1 口设定为输入口。无任何按键按下

时，P1.3 ~ P1.0 = 1111，P2 输出全 1 时按下任何键不影响 P1.3 ~ P1.0 的值。但 P2 输出为不全为 1 时则不同（见表 3.2 和表 3.3）：

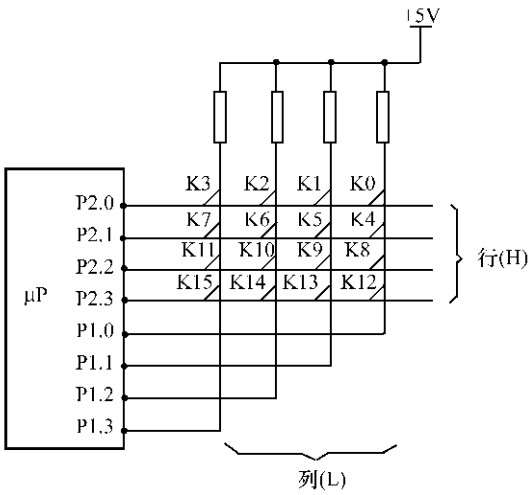


图 3.9　双口键盘扫描电路

表 3.2　端口与行号的关系

P2.3 ~ P2.0 值	行号 (H)
1　1　1　0	0
1　1　0　1	1
1　0　1　1	2
0　1　1　1	3

表 3.3　端口与列号的关系

P1.3 ~ P1.0 值	列号 (L)
1　1　1　0	0
1　1　0　1	1
1　0　1　1	2
0　1　1　1	3

如果 P2.3 ~ P2.0 = 1110，K0 按下，则 P1.3 ~ P1.0 = 1110；K1 按下，P1.3 ~ P1.0 = 1101；K0、K1 同时按下，则 P1.3 ~ P1.0 = 1100；可见某一行输出 0 时在该行上可以判断是哪个键按下。如果输出端同时有一行以上线上输出为 0 时就无法判断了，例如 P2.3 ~ P2.0 = 1100，K0 按下，则 P1.3 ~ P1.0 = 1110；K4 按下，P1.3 ~ P1.0 = 1110 为此我们依次让 P2.3 ~ P2.0 输出低电平，并不断地去读 P1.3 ~ P1.0 的值，就可以判别是哪个键按下。为了减少扫描次数，可以先令 P2.3 ~ P2.0 全部为低电平，这样只要有一键按下，P1.3 ~ P1.0 就不为 1111，由此判断是否有键按下，然后再扫描决定是哪个键按下并求得键值。图 3.10 为只有一个键按下时的键盘扫描程序流程图，由于假定只有一个键按下，所以只要扫到一个键值程序就结束。

二、中断式双口扫描

查询方式占用 CPU 大量的时间，为了克服这一缺点可采用中断方式。在图 3.9 加一个与门便可实现中断方式，如图 3.11 所示。平时令 P2.3 ~ P2.0 = 0000，当有一键按下与门就输出低电平，申请硬件中断，键盘扫描求键值的任务可放在中断服务程序之中。

三、键盘功能扩展说明

(1) 图 3.9 和图 3.11 都可扩展成更多个键的键盘，如 4 × 5、5 × 6、8 × 8 等。

(2) 同样的硬件也可以识别多个键同时按下，只要行和列实行全扫描记录所有按下去的键值即可。通过多键组合能扩展键盘的功能，并减少检测系统的误动作。例如我们可以用某几个键同时按下来启动指定的功能。

3.3.2　用数据口扫描键盘

多口扫描键盘只是在系统存在多余的端口时使用的，在实际的检测系统中，μP 的端口往往不够用，此时可采用数据端口和地址译码选通来实现扫描，其典型电路如图 3.12，该电路实现 8 × 8 = 64 键扫描，可用查询方式也可用中断方式。由锁存器送扫描线，由缓

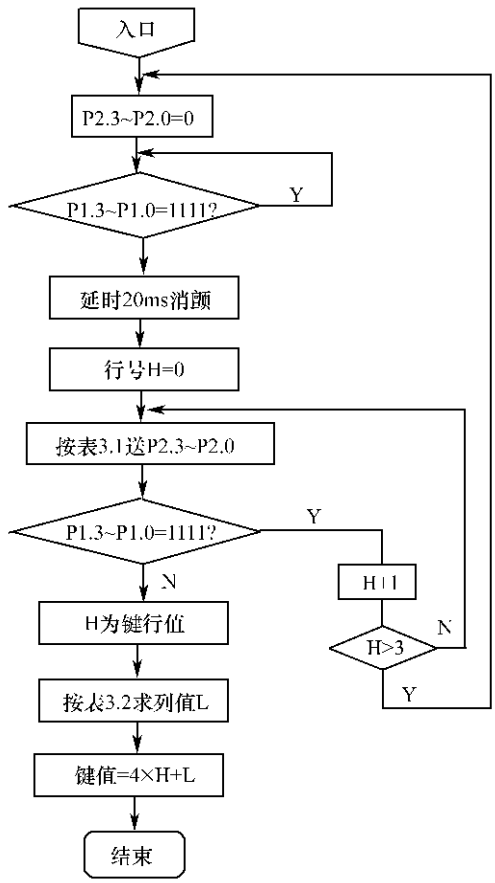


图 3.10　双口扫描键盘查询程序

冲器读数据。如读写复用译码地址 7FH，从 373 输出 00H 数值的程序为

```
MOV     DPTR,007FH          地址译码
MOV     A,    00H           要送的数据
MOVX    @DPTR,  A           送出
```

从缓冲器读取数据的程序为

```
MOV     DPTR,#007FH         地址译码
MOVX    A,  @DPTR           读入数据
```

3.3.3　键盘信息的处理方法

以上内容介绍了如何获得键值，下面简单叙述一下得知有键按下并获得键值后如何执行相应的程序。近年来微控制器的发展非常迅速，一般速度要求的可选 8 位或 16 位单片机，高速度要求的可选用 32 位信息处理单片机（DSP）。无论是哪一类微控制器，其速度都在不断提高，存储器都在不断扩大，编程使用高级语言的越来越多。为了使程序实现模块化，建议键盘扫描尽可能用中断方式。中断服务程序只求键值并根据键值发出“消息”。系统主程序及由外中断或内部定时中断启动的各模块程序根据这些“消息”来决定当前的工作。如

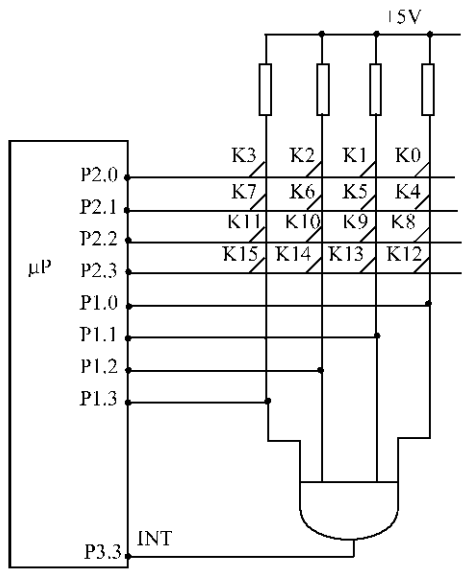


图 3.11　**带中断的双口键盘扫描电路**

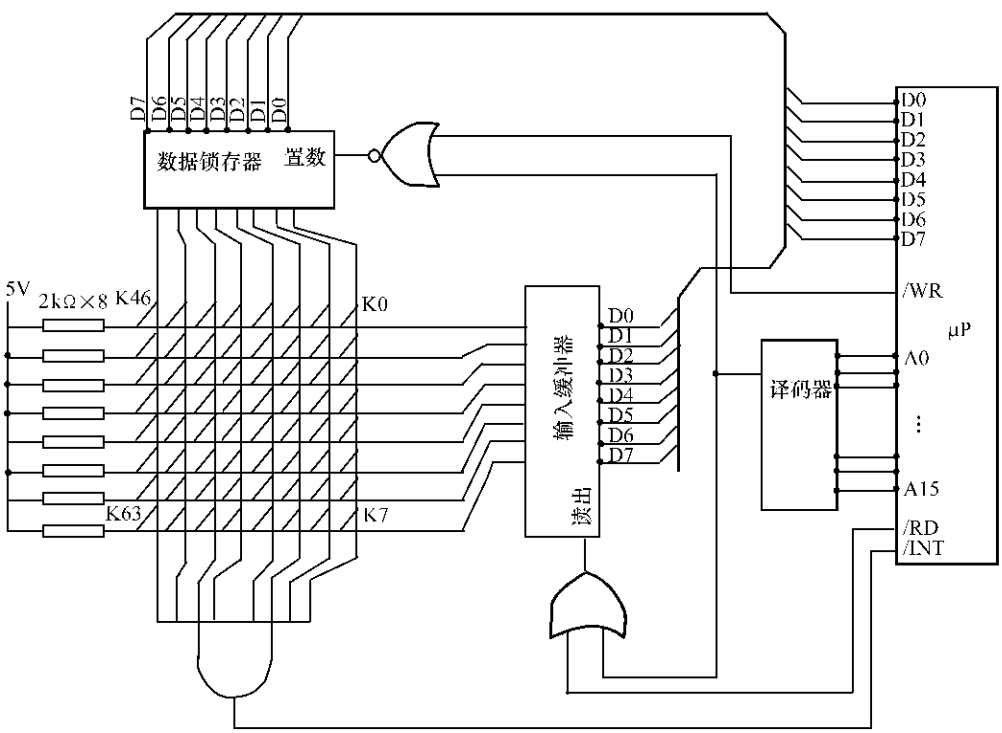


图 3.12　**用数据口对** 8×8 **键盘扫描**

图 3.13**为某一函数发生器的面板，设该函数发生器可产生方波、正弦波、三角波** 3 **种波形。面板中有** 10 **个按键功能，说明如下。**

方波 选择方波；

正弦波 选择正弦波；

三角波 选择三角波；

频率 选择显示频率或调整频率；

幅值 选择显示幅值和调整幅值；

占空比 选择显示占空比或调整占空比；

此外，按**快增**、**慢增**、**快减**、**慢减**可改变相应选中的输出频率、幅值、占空比。

键盘扫描可以用 3 ×4 = 12 键的中断方式扫描电路，其中断服务流程见图 3.14，主

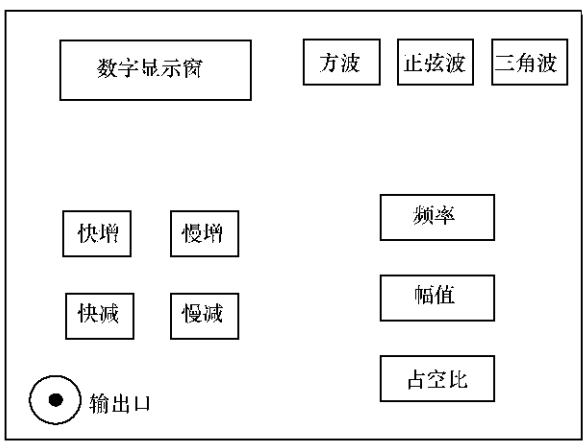


图 3.13 一个信号发生器的面板

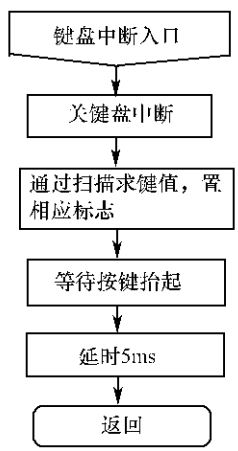


图 3.14 中断服务流程

程序的流程见图 3.15，主程序中仪器的有关设定参数存储在 E^2PROM 中，每次上电或在启动主程序时先从 E^2PROM 读取参数，并由此决定系统的工作方式、显示内容等。在参数设定有所改变时再将改变后的参数存入 E^2PROM 中。

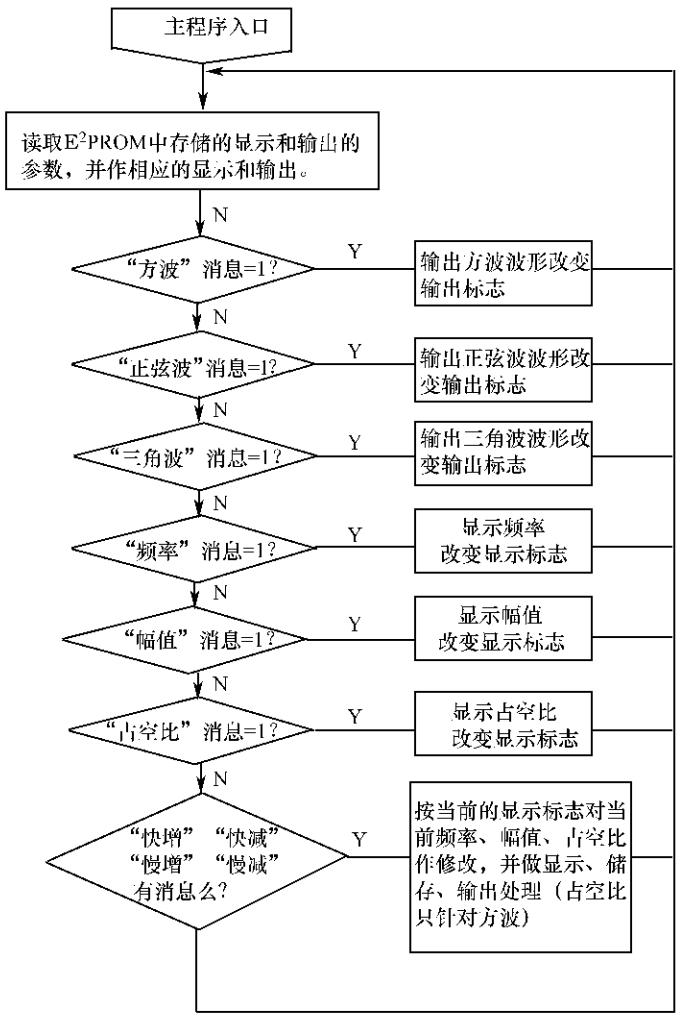


图 3.15　主程序服务流程

3.4　LED 类发光器件的显示控制

3.4.1　信号灯的显示控制

一、常见的信号灯

在检测系统及仪器设备中，经常需要用发光指示灯来表达信号。最早使用的信号灯是钨丝灯泡，其符号如图 3.16 (a) 所示。钨丝灯泡具有额定的工作电压和电流，交流直流电源都可使用且发光亮度高，其缺点是寿命短、耗电大、发光单一，因此在仪表盘等场合被

氖灯取代。氖灯是在真空玻璃泡中充入惰性气体用电极加直流电压使惰性气体电离而发光，加入不同的惰性气体会发出不同的光色。例如加入氖气可发出红光，因氖气灯用得较多，人们以氖灯作为此类信号灯的总称。氖灯的符号如图 3.16 (b) 所示。氖灯在工作时消耗电流很小，和钨丝灯泡相比氖灯具有寿命长、发光颜色多等优点。目前使用最广泛的是由发光二极管制成的信号灯，称为 LED (Light - Emitting Diode)，其符号如图 3.16 (c) 所示。发光二极管用半导体掺杂的工艺制成，属于固体发光器件，具有工作电压低、功耗小、寿命长、光色多、可靠性高等优点，是理想的信号指示灯。

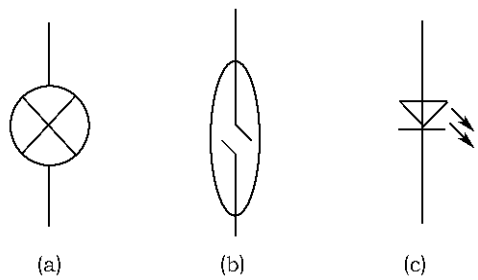


图 3.16　常见的几种信号灯符号

二、发光管的控制

单个发光二极管导通电压在 1.6V ~ 2.0V 之间，在智能检测系统中控制发光二极管可采用两种方式，如图 3.17 所示。

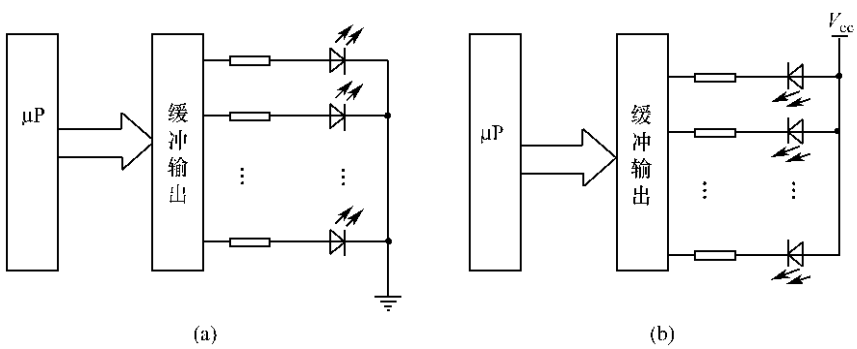


图 3.17　发光二极管控制方式

(a) 控制方式 1；(b) 控制方式 2。

在方式 1 中缓冲器输出高电平时发光管点亮，反之熄灭 在方式 2 中缓冲器输出低电平时发光管点亮，反之熄灭。这两种方式都是经常使用的，但方式 2 比方式 1 具有更多的优越性，体现在以下几方面。

(1) 方式 1 中缓冲器输出的高电平高低不同时发光管的亮度也不同，方式 2 因 V_{cc} 基本不变而不存在这个问题；

(2) 控制发光管的门电路往往吸入电流大于拉出电流，方式 2 可提供更大的驱动电流；

(3) 在用 OC 门或 OD 门驱动发光管时，方式 2 中的 V_{cc} 可以比门电路的电源电压大

很多（例如取 $V_{cc}=12V$），这给电路设计带来较大的灵活性，如图 3.18 所示。

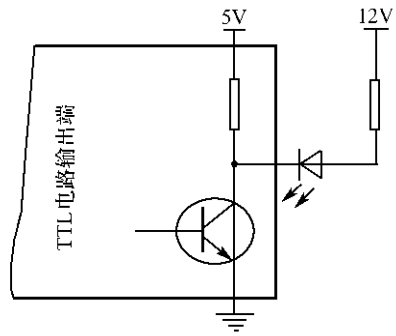


图 3.18　OC 门驱动高电压发光管

3.4.2　数码管及其显示控制

一、共阳和共阴的概念

将多个 LED 发光管组合形成一定形状可显示一些常见数码的器件称为数码管，较常见的是“8”字型数码管，需要 8 个发光管。多个数码管组合时如将所有的 LED 的正极连在一起称为共阳数码管，如将所有的 LED 的负极连在一起称为共阴数码管，分别见图 3.19（a）和图 3.19（b）。

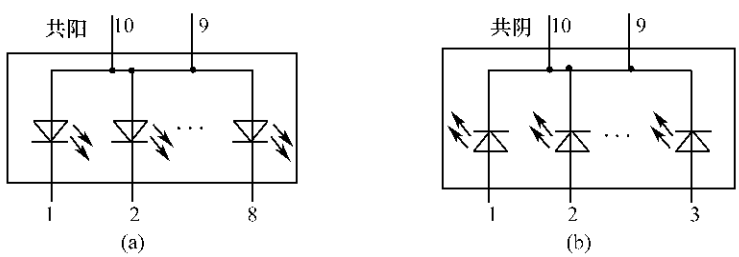


图 3.19　数码管的显示

（a）共阳数码管；（b）共阴数码管。

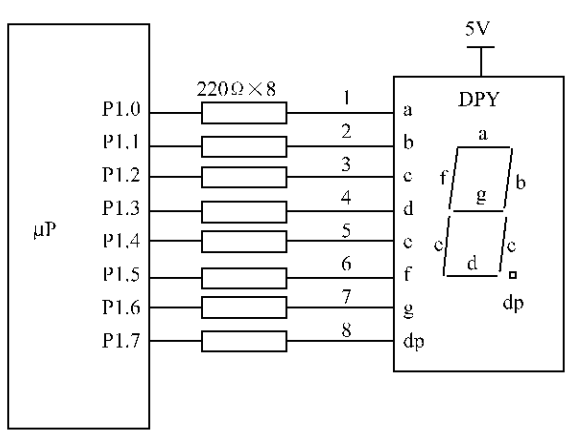


图 3.20　直接用单片机端口控制 8 段数码管

二、字型的控制

以 8 段共阳数码管为例，如图 3.20 所示，数码管段按图中顺序分别为 a hb hc hd he hf hg hdp，由单片机的 P1.0 ~ P1.7 直接驱动，假如我们要显示 0，1，2，3，4，5，6，7，8，9，b，c，E，全亮，全灭，相应的 P1 端的控制值如表 3.4 所示。

表 3.4　显示部分字符的段码控制值

显示值	P1.7 dp	P1.6 g	P1.5 f	P1.4 e	P1.3 d	P1.2 c	P1.1 b	P1.0 a	16 进制值
0	0	1	0	0	0	0	0	0	40H
1	1	1	1	1	1	0	0	1	F9H
2	1	0	1	0	0	1	0	0	A4H
3	1	0	1	1	0	0	0	0	B0H
4	1	0	0	1	1	0	0	1	99H
5	1	0	0	1	0	0	1	0	92H
6	1	0	0	0	0	0	1	0	82H
7	1	1	1	1	1	0	0	0	F8H
8	1	0	0	0	0	0	0	0	80H
9	1	0	0	1	0	0	0	0	90H
b	1	0	0	0	0	0	1	1	83H
c	1	1	0	0	0	1	1	0	C6H
E	1	0	0	0	0	1	1	0	86H
全亮	0	0	0	0	0	0	0	0	00H
全灭	1	1	1	1	1	1	1	1	FFH

按照表 3.4 可很方便地编出显示程序，例如：

显示 0 的指令为

```
MOV   P1,#40H;
```

显示 1 的指令为

```
MOV   P1,#F9H;
```

显示 E 的指令为

```
MOV   P1,#86H;
```

值得强调的是，在数码管的段控制线上必须有限流电阻，否则流过 LED 的电流会过大而损坏数码管或驱动电路。限流电阻接入的标准的方法如图 3.21 (a)，这种接法需要较多的电阻，但数码管各段显示亮度是均匀的；如果对显示均匀度要求不高，也可采用 3.21 (b) 的方法，只用一个电阻即可。

三、多位数码管的控制

(一) 静态显示

静态显示是指驱动数码管的信号在要显示的内容需要改变前一直保持一个值。因此在静态显示时发光体不间断地流过电流。如图 3.20 中当显示“8”时，P1 = 00h，一直保持不变的值。要显示多位数码管，如 4 位，必须保证 4 位数码管的显示驱动值同时存在。有

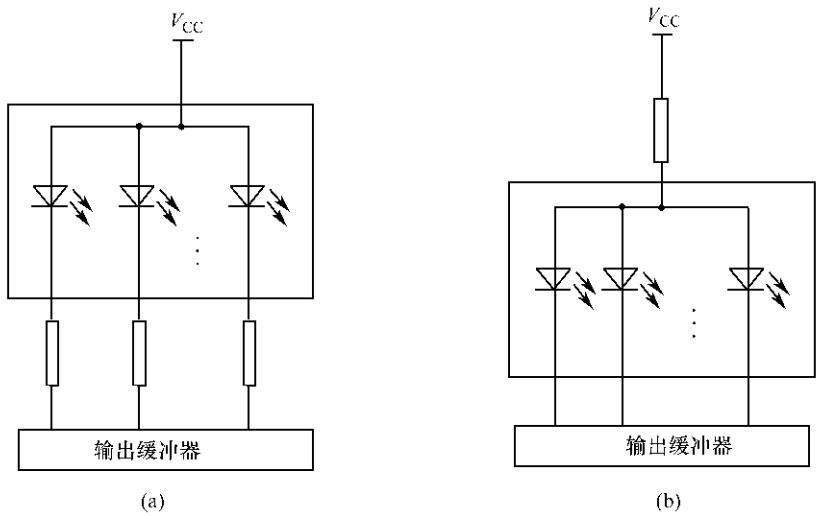


图 3.21　限流电阻的接法

(a) 单独的限流电阻；(b) 公用限流电阻。

两种方法可实现静态数码管显示驱动，一种为使用并行输入数据锁存器，另一种为串行输入数据锁存器。

1. 并行输入静态显示举例

图 3.22 为 3 位并行输入静态数码显示电路，数码管选用共阳的，图中未画出数码管的限流电阻。数据锁存器用 74HC373，数据锁存信号用地址 A15 ~ A13 的译码和写信号 $\overline{WR}$ 共同产生，从图中可以看出锁存器 S1 的译码地址为 0000H，锁存器 S2 的译码地址为 2000H，锁存器 S3 的译码地址为 3000H，假如要显示“345”，可执行如下指令：

```
MOV   DPTR,#0000H        ;L1 的地址
MOV   A,#B0H             ;L1 的显示驱动码
MOVX  @DPTR,A            ;送显
MOV   DPTR,#2000H        ;L2 的地址
MOV   A,#99H
MOVX  @DPTR,A
MOV   DPTR,#4000H        ;L3 的地址
MOV   A,#92H
MOVX  @DPTR,A
```

并行输入显示数据更新速度快，但占单片机端口也多。

2. 串行输入静态显示举例

图 3.23 为 3 位串行输入静态数码显示电路，数码管也选用共阳的。串行输入锁存器选用 74HC164，图中也未画出数码管推动限流电阻。由于采用串行输入，控制信号只须两条，一条数据线 (DATA)，一条脉冲线 (PULSE)，显示位数从理论上可无限，工程上一般不超过 20 位。假如要显示“000”~“999”之间的数码，可用如下程序实现。

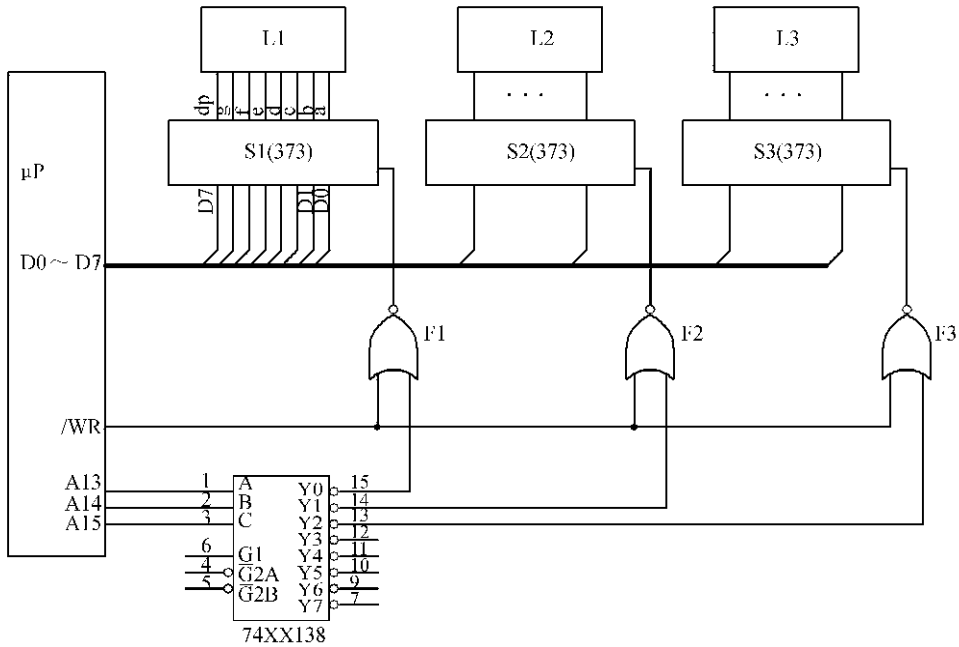


图 3.22　**并行输入的静态显示电路**

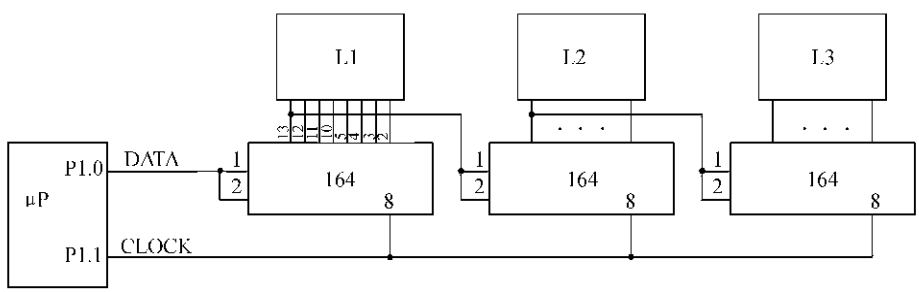


图 3.23　**串行输入的静态显示电路**

```
Sbit   Data  =  P1 0；
Sbit   Pulse  =  P1 1；
unsigned char code led _ table []  =
{0xc0，   0xf9，   0xa4，   0xb0，   0x99，   0x92，
//0       1        2        3        4        5
0x82，   0xf8，   0x80，   0x90，   0xff} ；
//6       7        8        9        10 Black 定义显示驱动码
. void display (int dis _ v)
{
    unsigned char i,j,temp；
    unsigned int dis _ bit [3]  =  {10,10,10} ；
```

```
    if (int _ dis _ v < 1000.0)
        {
        for (i = 0 ;i < 3 ;i + +)          //取每1位的值
            {
            dis _ bit [i] = int _ dis _ v - (int _ dis _ v/10) * 10 ;
            int _ dis _ v = int _ dis _ v/10 ;
            }
        }
    for (i = 0 ;i < 3 ;i + +)
        {
        temp = led _ table [dis _ bit [2 - i]] ;
        for (j = 0 ;j < 8 ;j + +)
            {
            Data = (bit) (temp&0x80) ;
            Pulse = 1 ;
            Pulse = 0 ;
            temp = temp≪1 ;
            }
        }
}
```

利用串行输入实现多组数码静态显示非常方便，只要用一个译码器选通 PULSE 即可。如图 3.24，CS1、CS2、…CS_n 分别控制各组的 PULSE 是否产生。公共的 DATA 经驱动后加入各组的 DATA 端，各组的显示位数可以相同也可以不同。

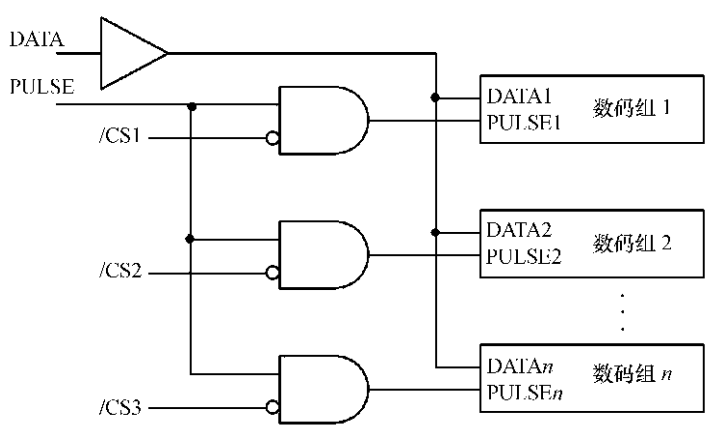


图 3.24　多组串行输入静态显示电路

（二）动态显示

静态显示容易实现，显示占用系统的时间开销小，但耗电量大，所需器件较多。在必

要时可用动态显示来代替。所谓动态显示是指用均匀脉冲来给各数码管供电，流过发光段的电流是一定频率的脉冲电流，当这个脉冲的频率高于 20Hz 时，人眼的视觉残留现象会使人感觉是静态的效果。下面举例来说明动态显示的原理。

图 3.25 为一个多组数码管的动态显示的原理图，该电路可实现 8 位 8 段数码管的动态显示，数码管为共阴。锁存器 S1 提供单位数码管显示代码（注意：由于选择共阴数码管，其值正好是表 3.4 对应值的反码），锁存器 S2 控制各位数码管的电源通断。每时每刻缓冲器 Q8 ~ Q15 只有一个低电平。例如需要数码管 L1 显示“8”时，Q7 ~ Q0 = 0ffh，而 Q8 ~ Q15 = 0FEH。图 3.26 为显示“12345678”的软件流程图。设刷新周期 T = 1 / (20Hz) = 50ms，每位的显示时间为 50 / 8 = 6.2ms，流程图中用延时法实现每位数码管持续供电时间，这样将占用 CPU 全部的时间。在实际应用过程中可用定时中断法实现，请读者自行编写出定时中断法实现同样功能的工作流程和程序代码。

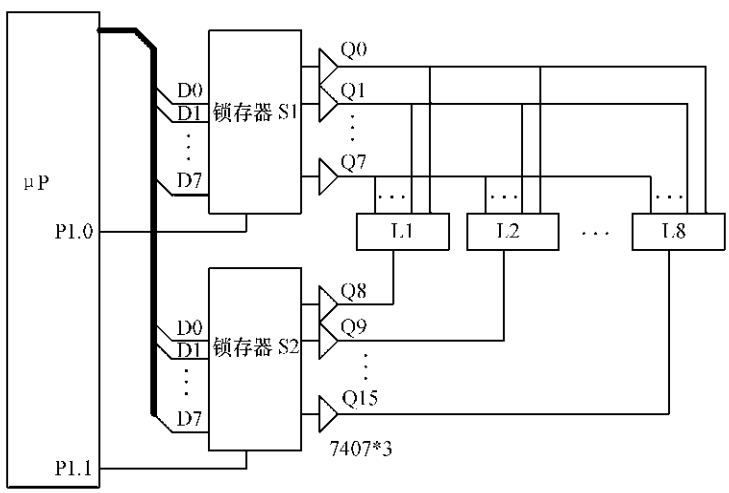


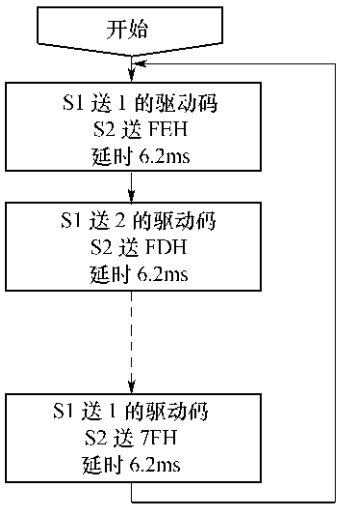
图 3.25　8 位数码管动态显示电路

3.4.3　LED 点阵及其显示控制

按照上述 8 位数码管动态数码显示原理，将数码控制线作为行，电源控制线作为列，8 个数码管共 8 × 8 = 64 个发光管就是一个 8 × 8 的点阵，将每个数码管的发光段排成一列得到如图 3.27 所示的点阵。仔细观察图 3.27 的点阵，按列看每组发光管是共阴组合，按行看每组发光管是共阳组合，所以这样的点阵可按共阳或共阴两种方式控制其动态显示。如按共阴方式可直接用图 3.25 的驱动电路。不同的是这样的点阵可以显示任何字符或简单图形而不仅仅是显示数码。例如显示“T”字符，可令如图 4.28 指示的对应的数码管点亮即可，其程序很容易编写。

按照图 3.27 原理可以制成任意的行列式发光矩阵，如 16 × 16，7 × 8，5 × 7 等，早期的点阵显示块以 5 × 7 居多，为此有的厂家还配套生产专用驱动电路，电路由 ROM 型字符发生器、计数器、译码器等组成，如图 3.29 所示。5 × 7 点阵相当于 5 列 7 段码的动态显示，

5 个 Clock 时钟信号可对字符刷新一次，工作过程如表 3.5 所列。不过，目前人们已很少用此方法来驱动点阵，而直接用软件形成要显示的段码，设计方法因点阵面积大小、几何形状而各有千秋。以 800 ×800 点阵为例，我们选用 8 ×8 的发光点阵块，共用 100 块，电路如图 3.30 所示。



3.26　**动态显示“12345678”的流程图**

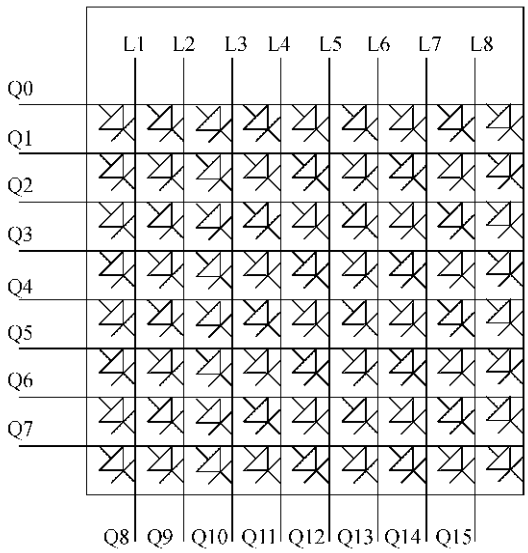


图 3.27　共阳的 8×8 点阵结构

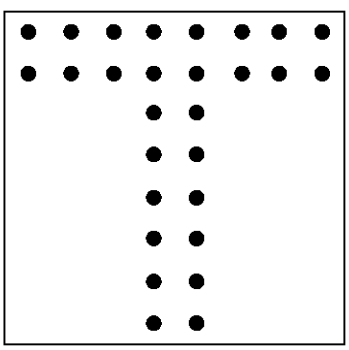


图 3.28　点阵显示字符的原理

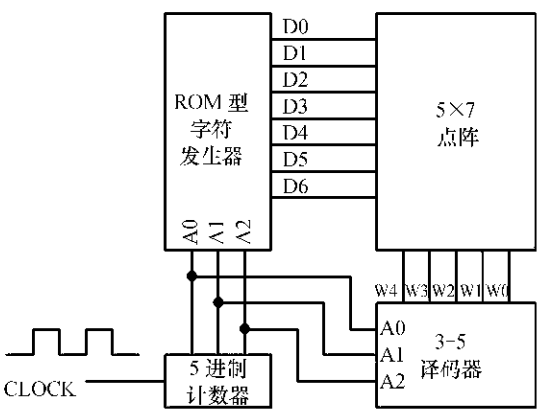


图 3.29　5×7 点阵控制电路结构

表 3.5　5 列 7 段码的动态显示过程

Clock 序号	地址 A2 ~ A0	点阵行值 (段值 D6 ~ D0)	点阵列值 (W4 ~ W0)	显示内容
0	000	第 1 列驱动码	11110	第 1 列内容
1	001	第 2 列驱动码	11101	第 2 列内容
2	010	第 3 列驱动码	11011	第 3 列内容
3	011	第 4 列驱动码	10111	第 4 列内容
4	100	第 5 列驱动码	01111	第 5 列内容
5	101	第 1 列驱动码	11110	第 1 列内容
…				

图中未画出主控单片机，只画出点阵控制的驱动电路，驱动方式为共阳，整个点阵由 H0 ~ H9 10 个大块组成，其结构完全相同。每个大块包含 10 个 8 × 8 点阵小块及其驱动电路，定义小块为 DLij，则 H0 由 DL00 ~ DL09 组成，H1 由 DL10 ~ DL19 组成，……H9 由 DL90 ~ DL99 组成。以 H0 为例，DL00 ~ DL09 的动态刷新是同时进行的，即 DL00 显示第 1 行时，DL01 ~ DL09 也在显示第 1 行；DL00 显示第 2 行，DL00 ~ DL09 也在显示第 2 行，行驱动码由锁存器 S00 ~ S09 通过 μP 给定。μP 首先以极快的速度给出 10 个驱动码，然后通过锁存器 P0 给定该行的电源控制值，例如：要显示第 1 行，则 P0 输出值 P07 ~ P00 = 01H，要显示第 2 行，则 P07 ~ P00 = 02H。P0 的输出大功率三极管可提供较大的电流，因此它同时给 H0 中 10 个小方块提供电流。

H1 ~ H9 的控制与 H0 完全相同，并且可以可以与 H0 的控制同时进行。值得指出的是，图 3.30 仅仅是实现 800 × 800 点阵的方案之一。

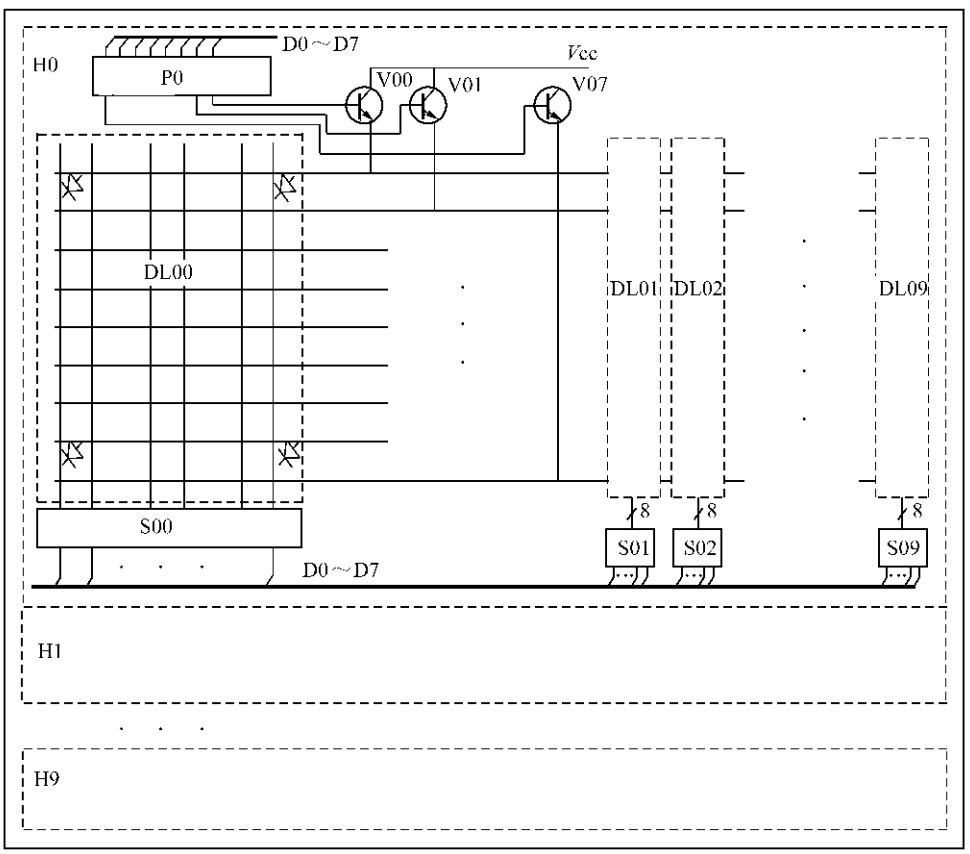


图 3.30　800 × 800 点阵控制电路举例

3.5　CRT 的显示控制

CRT (Cathode Radial Tube) 又称阴极射线管，是最常见的显示部件，具有速度快、无噪声、无磨损、显示灵活、直观方便等优点。虽然近年来不断地涌现出液晶、等离子等更先进

的显示器以克服 CRT 体积大、有辐射等缺点，但这些新型显示器还是沿用 CRT 的显示驱动模式，因此，了解 CRT 的基本原理及其显示驱动方法具有普遍的意义。

3.5.1 光栅扫描式系统

一、CRT 的结构

以单色 CR 为例，它由带显示屏的玻璃管、热灯丝、阴极、加速及聚焦系统、X 偏转极、Y 偏转极等组成，如图 3.31 所示。灯丝在通电后加热产生电子，通过阴极发射出来，再经过加速及聚焦系统变成高速的电子束，最后由 X 偏转极和 Y 偏转极控制电子束的方向打在屏幕上。由于屏幕上涂有荧光粉等材料，受电子束轰击后能放出光来让人感受。这就是 CRT 发光的基本原理，剩下的问题是如何让 CRT 显示人们要求的内容。

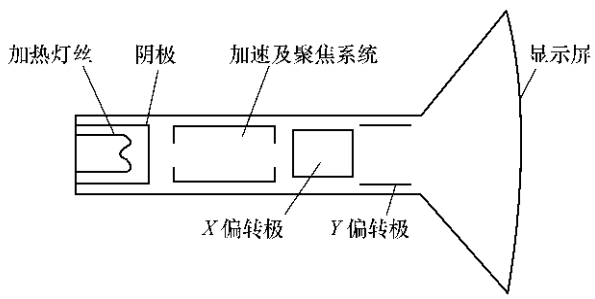


图 3.31 单色显像管的结构

二、字符显示原理

设 X 偏转极和 Y 偏转极是受 V_x 和 V_y 控制的，一般将其控制特性设计调试为 $V_x = 0$，$V_y = 0$，光点在中间；$V_x = V_{xmin}$，$V_y = V_{ymin}$，光点在左上方；$V_x = V_{xmax}$，$V_y = V_{ymax}$，光点在右下方。于是，当 V_x = 常数 C_1，V_y = 常数 C_2 时，屏幕上会形成一不动亮点；当 V_x 加矩形波（波谷为 V_{xmin}，波峰为 V_{xmax}），V_y = 常数 C 时，屏幕上形成一条亮线；当 V_x 加锯齿波、V_y 也加矩形波（波谷为 V_{ymin}，波峰为 V_{ymax}）时，整个屏幕会发光。

设 V_x 加锯齿波的周期为 T_x，V_y 加锯齿波的周期为 T_y，如图 3.32 所示。当 $T_y = nT_x$

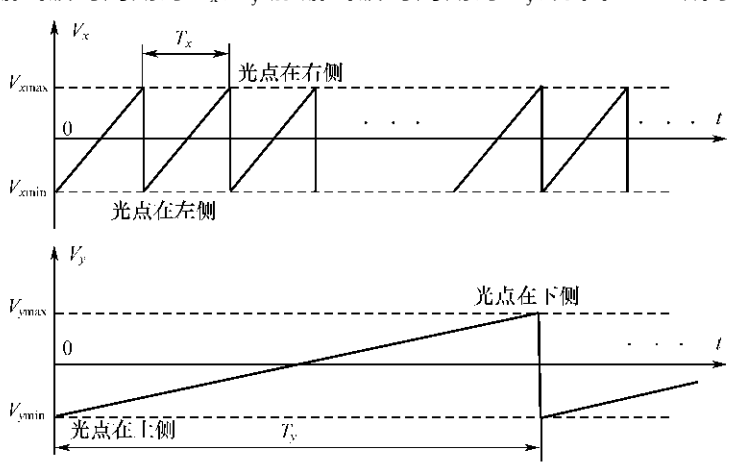


图 3.32 行扫描和场扫描控制信号

时，n 个 T_x 后为一次扫描屏幕上瞬间出现均匀的光点，下一次扫描从左上角开始且重复原轨迹，如此不停地扫描使屏幕出现持续的全亮，称之为光栅，见图 3.33。这时候，我们称 V_x 为行扫描信号或行频信号，称 V_y 为场扫描信号或场频信号。

以上情况，是假定阴极在发射电子，如果在上述扫描时，令阴极控制电压 $V_a = 0$，则会黑屏。可见在 V_a 加上控制信号，可使屏幕上任意一点变黑或变亮，我们称 V_a 为视频信号。显然，视频信号要比行频信号和场频信号都高。三者的关系见图 3.34。

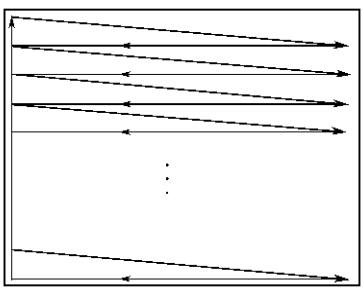


图 3.33　电子扫描在屏幕上的轨迹

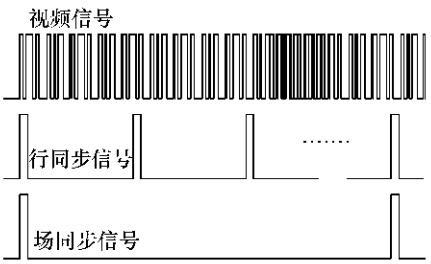


图 3.34　视频、行频和场频信号

根据人的视觉习惯，场扫描频率应在 20Hz 以上，比如取 50Hz。确定场频后要根据整个屏幕显示的点数确定行频和视频，例如早期的标准单色显示器为 24 行 80 列，共 80 × 24 = 1920 字，字符为 5 × 7 点阵。每字符之间间隔 2 个消隐点，每行之间间隔 3 行消隐行。这样，横向：(5 + 2) × 80 = 560 点 纵向：(7 + 3) × 24 = 240 点。每一帧用时间为 20ms，行频周期 $T_x = 20\text{ms}/240 = 83\text{s}$，视频周期 $T_a = 20\text{ms}/(240 \times 560) = 148\text{ns}$。假定我们要在左上角显示一个字符“T”，视频信号在前 7 行的扫描中开始位置要送上串行信号 11111、00100、00100、00100、00100、00100、00100。如果将要显示的字符按顺序存入 RAM，设法同步读出就可实现显示，这就是字符显示原理。

上述方法不但适合于字符显示，也适合无灰度等级的点阵图形显示。如要显示如图 3.35 所示的箭头，只将箭头的点阵存入 RAM 中，并按顺序读出即可。

CRT 一般是一个单独的部件，自带扫描电路及其相应的聚焦、亮度调节等电路，只要给它提供场频、行频、视频信号就可以工作了。产生这类信号的电路俗称显卡。图 3.36 为早期的单色 CRT 显卡的电路框图。

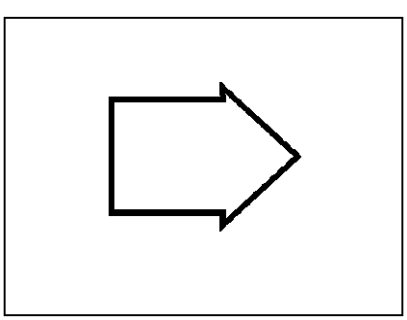


图 3.35　无灰度图形举例

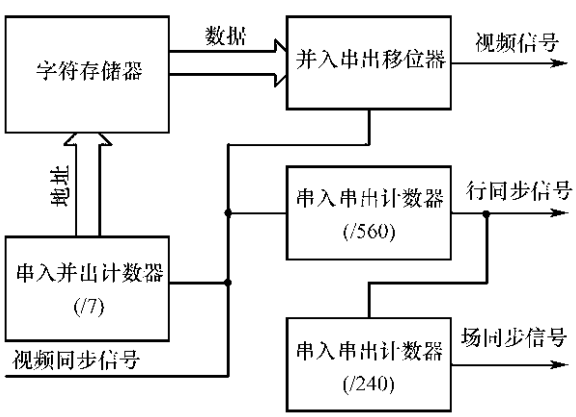


图 3.36　早期单色 CRT 显卡电路

三、图形显示原理

当视频信号为脉冲信号时，可控制荧光屏上的点的亮灭，只能显示字符或无灰度等级的点阵图形。若把视频信号变为有高度等级的同步脉冲，可以控制到对应点的明暗，从而形成具有灰度等级的图形。例如图 3.37 中与行同步、场同步对应的视频信号为锯齿波，这时每条行扫描线可以在屏上得到一条由暗变明的线条，场扫描后最终得到一个从左到右由暗到明的屏幕。实现灰度显示的视频信号是模拟信号，不能像图 3.36 那样用并行输入串行输出的寄存器实现，而要用 D/A 来产生。D/A 的输入位数决定灰度的级数，称为像位数，如 4 位 D/A 可识别 16 种灰度，5 位 D/A 可识别 32 种灰度，6 位 D/A 位可识别 64 种灰度，8D/A 位可识别 256 种灰度等等。与无灰度的字符驱动显卡相比，带灰度的显卡中存取视频信号的 RAM 就要相应变大，显示器的像素 × 位数即为一帧所需的 RAM 量。如显示器为分辨力 1024 × 768，像位数为 4，存储 1 帧的 RAM 位数为 1024 × 768 × 4 = 3145728（位）。目前市场上流行的显示器都是图形显示器，所有字符都按图形点阵处理，对其控制实质上是给显示缓冲区写入要显示的点阵。

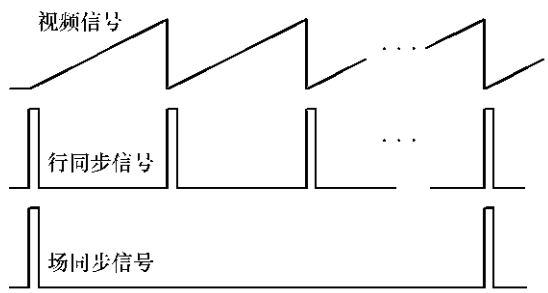


图 3.37　灰度显示举例

四、逐行扫描及隔行扫描的概念

上面的所述扫描路线如图 3.33 是左上角→右下角→左上角，称为逐行扫描，如取场周期为 50Hz，逐行扫描在 20ms 内完成一帧。还有一种方法称为隔行扫描，如图 3.38 所示，先扫描 1、3、5、7、9……，完成后再扫描 2、4、6、8……通过两次扫描完成满屏幕的扫描，此方法可降低对硬件的速度要求，但与逐行扫描相比会有闪烁感。

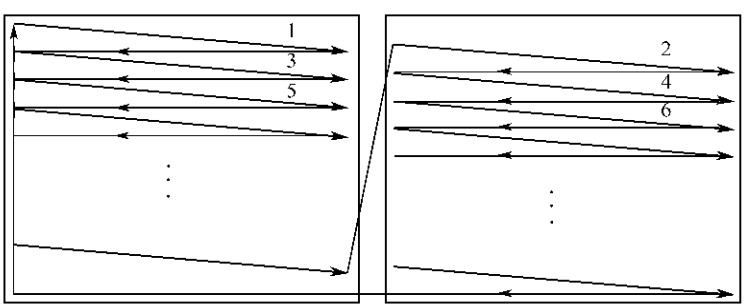


图 3.38　隔行扫描的基本原理

五、彩色显示器的原理

根据三基色原理，自然界各种颜色都可以分解为红、绿、蓝3种颜色，而三基色按不同比例混合，又可得到各种颜色。彩色电视就是根据这一原理工作的。在彩色摄像中，从被摄景物反射的彩色光，经过分光系统分解为3个基色光，在3个摄像管的光敏层上分别形成3幅基色图像。3个摄像管中的电子束受同步的偏转设备所控制，从而步调一致地从左到右、从上到下地进行扫描，依次将图像中各个像素的基色光亮度转换为电信号。这样，3个摄像管所产生的3个电信号在每一瞬间都对应于图像上的同一个点，经过放大后，摄像机输出3个基色信号。可见，彩色摄像机是彩色分解的设备，又是光—电转换设备。在彩色显像中，为了重现彩色图像，必须把3幅基色图像叠加在一起，所以彩色显像管是电—光转换设备，又是彩色合成的设备。一个显像管可以看作是组装在一个管壳中的3只单色显像管，由电子枪发射的3个电子束，分别受三基色信号电压所控制，每一电子束产生一种基色的光点。显像管偏转系统与摄像管的偏转系统是严格同步地工作，因此，每一电子束都在荧光屏上重现出相应的基色图像。由于对应于同一色素的3个基色光点相距很近，如果我们离开荧光屏一定距离，眼睛就不能分辨，看到的将是三基色相加混合产生的彩色图像。

彩色显像管以荧光粉的发光颜色作为彩色显像基色。目前，市场上的彩色CRT主要是由电子枪、偏转线圈、荫罩或荫栅、荧光粉层和玻璃外壳五大部分组成，图3.39为彩色显像管基本原理示意。这种显示器的显示原理与单色CRT基本相同，显像管内部的电子枪（阴极）通电，发出电子束，经强度控制、聚焦和加速后变成细小的电子流，由偏转线圈控制电子的方向，穿过荫罩的小孔或荫栅并经荫罩或荫栅调正，然后高速轰击到荧光屏上，荧光屏上的荧光粉被激活，就可以发出光来。图3.40指示出显像管中荫罩的作用。R、G、B三色荧光点被按不同强度的电子流点亮，就会产生各种色彩。彩显和单显的不同点在于：

⑴ 电子枪为三束，控制聚焦为3套。

⑵ 屏幕上涂有3种发光粉红色（R）、绿色（G）和蓝色（B），一般为条形。电子束通过定位系统严格打在各自的条形上。

⑶ 根据原色调和原理，由三基色可组成所有的颜色。行和场的扫描与单显的原理相同。视频信号为3种，每一种都有灰度等级，因此存储器需要扩大3倍。

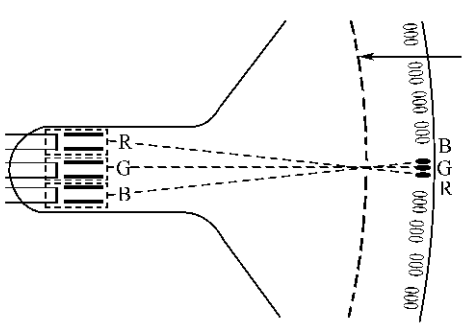


图3.39　彩色显像管基本原理

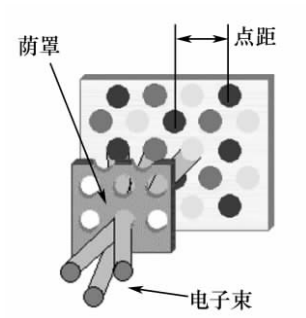


图3.40　荫罩的作用

六、彩色CRT显示控制

对于一个检测系统来说，要显示的内容不一定是预摄好的内容，它可能是一组波形、

一串数据、一幅动画或别的什么东西。

与单色显像管一样，彩色显像管的扫描等强电信号电路都安装在显示器中，这些工作是显示器生产厂完成的。彩色显卡的设计基本原理和单色显卡相同，电路设计除了产生视频、行同步信号、场同步信号外，还要有其它控制信号，比如亮度控制、聚焦控制、偏角控制等辅助信号。例如 PC 机的显卡就是完成这个功能的，PC 机显卡由于有众多厂家生产，价格非常低，因此基于 PC 机的智能检测设备采样直接购买显卡的方式更为方便快捷。基于 μP 的智能检测设备一般很少有现成的显卡出售，往往须设计适合自身的显示驱动装置，在设计时可选用专用芯片，这里不做介绍而只谈谈用用 DSP 实现对 CRT 的控制的一个设想。

当主频到达足够高，DSP 的端口配合 D/A 等少量器件可直接产生视频、行频、场频等同步信号。例如主程序先将需显示的内容按顺序存在寄存器中，定时中断按规定的时间发送相应的视频信号及同步信号，这样制作的显示驱动电路将具有更大的灵活性。

3.5.2 随机扫描式显示系统

和光栅扫描方式不同，随机扫描不是全屏扫描以点阵形式形成文字或图形，而是直接在 CRT 的 X 轴输入（行扫描控制）端和 Y 轴输入（场扫描控制）端加以连续变化的信号，通过电子束的移动形成连续的各种轨迹。当轨迹移动时要求显像管始终在发射电子，即在视频端加一个常压信号。如果在视频端加一定的脉冲信号，就可构成不连续的轨迹。本书只讨论显示连续波形的随机扫描系统原理。这个问题实质上是如何用 μP 控制普通示波器显示所要求的波形。

如图 3.41 所示，设有两路模拟电压信号 V_{i1} 和 V_{i2}，经多路开关选择后，分别经过放大和 A/D 转换采得一组数据存入 μP 的 RAM 中。设一组数据为 256 点，即 V_{i1} 和 V_{i2} 各采得 256 点，我们将 V_{i1} 和 V_{i2} 的转换值分别存在数组 Ch1〔256〕和 Ch2〔256〕中，采样频率为 f_0，μP 通过 P2 口经数模转换器 DA1 产生垂直偏转信号，通过 P3 口经数模转换器 DA2 产生水平的偏转信号。假定 DA1 和 DA2 的输出范围为 0～5V，为了利用示波器的全部窗口显示波形，需调节示波器的光点位置，并选择合适的增益使 DA1 = DA2 = 0 时光点位于左下角，DA1 = DA2 = 5V 时光点位于右上角。示波器触发方式选用 X 输入触发，剩下的就是编程序了。

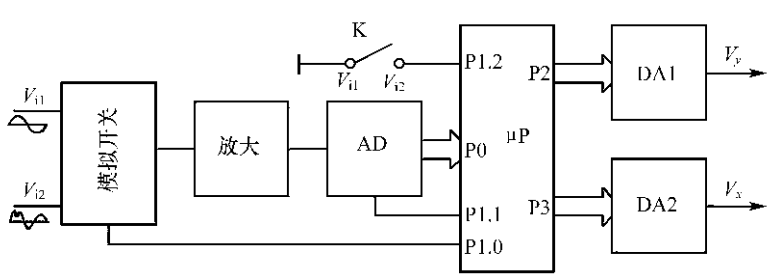


图 3.41 一个双综智能示波器的原理框图

（一）显示一路波形

图 3.42 为显示一路波形时示波器 X 轴和 Y 轴应输入的波形，实际上 X 轴送的是锯齿波，Y 轴送的才是要显示的波形。

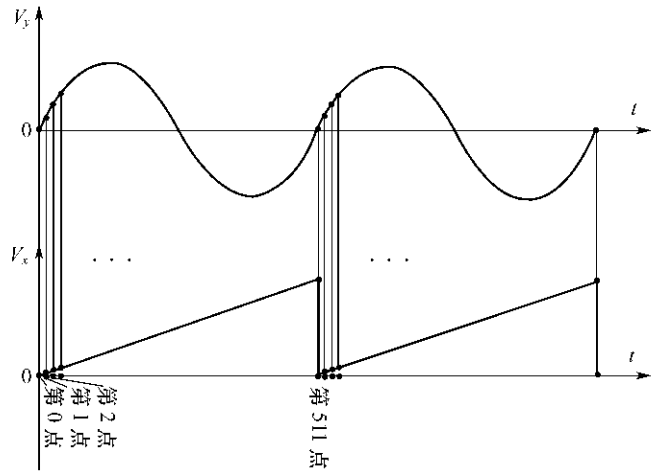


图 3.42　用随机扫描方式显示一路波形

用开关 K 来选择显示 V_{i1} 或 V_{i2}，设波形刷新的时间为 50ms，采样点为 256，则显示每一点时间为 50/256 = 0.19 (ms)。为了便于理解，我们用延时的办法来设计程序，其基本原理是 X 轴送锯齿波，锯齿波的周期为 50ms，按 0.19ms 间隔顺序在 Y 轴送出 V_{i1} 或 V_{i2} 的采样值，便可实现所选择的波形显示，程序流程如图 3.43 所示。

（二）同时显示两路波形

要显示两个波形，可使两个波形处于不同的高度，如图 3.44 所示。假定 V_{i1} 显示在上方，V_{i2} 显示在下。因 V_y 信号只有一个，我们将 DA1 的输入数字信号划分为两个区域：128 ~ 256 之间的数值映射 V_{i1} 的采样值 Ch1 [j] (j = 0 ~ 255)；0 ~ 127 之间的数值映射 V_{i2} 的采样值 Ch2 [j] (j = 0 ~ 255)。实现双波显示可采用两种方式：方式 1 为先用 25ms 的时间连续显示 V_{i1} 的波形，紧接着再用 25ms 的时间连续显示 V_{i2} 的波形 (方法如上)，总的刷新周期还是 50ms。方式 2 为用 50ms 锯齿波作为 V_x 的输出，按 0.19ms 间隔顺序在 Y 轴送出 Ch1 [j] 和 Ch2 [j] 的映射值 (两值的停留时间各为 0.19ms/2)。总的刷新周期也为 50ms，读者可试着编写一下显示程序。

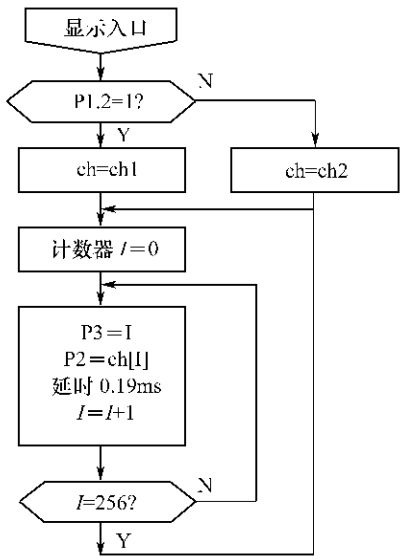


图 3.43　随机扫描显示一路波形流程

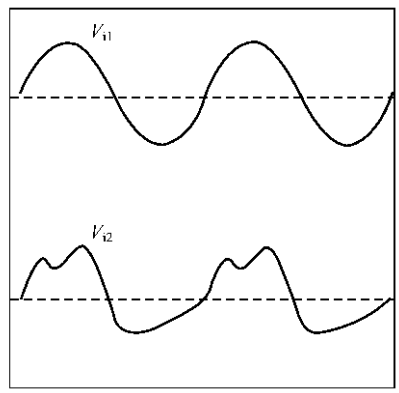


图 3.44　随机扫描法同时显示两路波形

3.6　微型打印机的控制

微型打印机俗称微打，包括针式打印机和热敏打印机两大系列，前者为点阵针击打式，打印中有击打声音发出，但可打印多层压感纸；后者为点阵感热式，打印时没有击打声音；二者各有优势，都得到广泛的应用。一般地，打印机内部都有一片单片机对打印机构进行控制，图 3.45 为常见的微型打印机的外形。下面只介绍一种典型的 WH 系列针式打印机。

图 3.45　微型打印机的常见外形

3.6.1　微型打印机的字符代码和打印命令举例

以 WH 系列打印机为例，它包含 3 个字符集。

字符集 1：包含有 6 × 8 点阵字符 224 个，代码范围 20H ~ FFH，包括 ASCII 字符及各种图形符号等；

字符集 2：包含有 6 × 8 点阵字符 224 个，代码范围 20H ~ FFH，包括德、法、俄文、日语平假名和片假名等；

字符集 3（可选）：该字符集为点阵汉字库：WHXXX5X 型微打配 12 × 12 点阵汉字 140 个，代码范围 20H ~ ABH（32 ~ 171）。这 140 个汉字可以由 WH 系列微打配备的下载软件自由设置和下载；WHXXX8X 型微打配 12 × 12 点阵的国际一、二级字库汉字和 6 × 12 点阵的国际标准 ASCII 码。汉字代码为标准汉字内码，机内码的计算方法是：区位码 + A0A0。例："湖"字的区位码是 2694，即 26 区第 94 个字，其机内码为 2694 + A0A0 = BAFE，汉字的机内码也可从打印机自带的汉字机内码表中直接查出。

WH 系列微型打印机提供众多的打印控制命令，这些命令由一字节控制码或多字节控制码序列组成。它们和市场上普通的微型打印机控制命令完全兼容，并增加了汉字打印、字符汉字旋转、字间距调整、条形码打印和图形打印等功能。

打印命令控制码和要打印的内容均以相同的方式送给打印机，WH16XXX 型打印机每行可打印 16 个字符（或 8 个汉字）；WH24XXX 型打印机每行可打印 24 个字符（或 12 个汉字）；WH40XXX 型打印机每行可打印 40 个字符（或 20 个汉字）。输入字符不足规定的个数，可用"0D"（回车换行）命令回车换行，若输入字符超过每行规定的个数，打印机

将自动回车换行。

打印命令功能如表 3.6 所列。命令由命令代码和参数两部分组成。任何命令均有命令代码部分，而命令参数部分则随命令的不同其多少也不同。表中命令代码部分不带括弧的是 16 进制码，括弧内为 ASCII 码。

表 3.6　WH 打印机的命令代码及功能举例（用十六进制码表示）

命令名称	命令代码	命令格式	功　能　说　明
选择字符集 1	1B　36 (ESC　6)	1B　36	在该命令之后的字符将使用字符集 1 的字符进行打印。字符集 1 中有 6×8 点阵字符 224 个，包括 ASCII 字符，及各种图形符号等
选择字符集 2	1B　37 (27　55)	1B　37	在该命令之后输入的代码将选择字符集 2 的字符打印。字符集 2 中有 6×8 点阵字符 224 个，代码范围 20H - FFH (32 - 225)。包括德、法、俄文、日语片假名等
选择 12×12 点阵汉字打印	1B　38 (ESC　8)	1B　38	在该命令之后输入的代码将选择 12×12 点阵汉字进行打印：WHXXX5X 型配 12×12 点阵汉字 140 个。WHXXX8X 型配 12×12 点阵的国际一、二级字库汉字和 6×12 点阵的国际标准 ASCII 码，汉字代码为标准汉字机内码
换行	0A (LF)	0A	打印机走纸一字符行，即 8 + 行间距个点行
执行 n 点行走纸	1B　4A (ESC　J)	1B　4A　n	打印纸向前进给 n 点行，1≤n≤255。这个命令不发出回车换行，它也不影响后面的换行命令
设置 n 点行间距	1B　31 (ESC　1)	1B　31　n	为后面的换行命令设置 n 点行间距，0≤n≤255，上电或初始化后 n = 3
设置字间距	1B　20　n (ESC　SP)	1B　20　n	设置字符之间的空白点数，即每打印完一字符打印机自动在字符右侧加入的空白点数。汉字的字间距加倍。0≤n≤128。上电或初始化后 n = 0
换页	0C (FF)	0C	走纸到下一页的开始位置

定义用户自定义字符	1B　26 (ESC　&)	1B　26　m n1　n2. . . n6	**这个命令允许用户定义一个字符，**m **是该用户自定义字符码，**32≤m≤61**。参数** n1**，**n2**，**. . . n6 **是这个字符的结构码。字符由** 6 ×8**点阵组成，即** 6 **列每列** 8 **点，每一列由一个字节的数据表示如下所示：** 1　2　3　4　5　6 **最高位** D7　□ □ □ ■ □ □ □ ■ ■ ■ ■ □ □ ■ □ □ □ □ □ ■ □ □ □ □ □ ■ □ □ □ □ □ ■ □ □ □ □ ■ □ □ □ □ □ **最低位** D0　□ □ □ □ □ □ n1 = 02H**，**n2 = 7CH**，**n3 = 40H**，**n4 = C0H**，**n5 = 40H**，**n6 = 0H **如果许多** ESC& **命令使用同一** m **值，只有最后一个有效，最多可定义** 30 **个字符**

(续)

命令名称	命令代码	命令格式	功 能 说 明
打印点阵图形	1B 4B (ESC K)	1B 4B n1 n2...data...	该命令打印 n1×8 点阵图形。该图形宽度为 n1 点,高度为 8 点,每一列的 8 个点由一个 8 位的字节表示,最高位在上。n1、n2 的数值表示一个 16 位的二进制数,n1 为低 8 位字节,n2 高 8 位字节,表示 ESC K 打印图形的宽度为 n2×256+n1。因微型打印头的宽度都小于 256,所以 n2 总是 0,n1 应在 1 到所配打印头的每行最大点数之间。data 是该点阵图型自左向右每一列的字节内容,数据个数必须等于 n1。 当图形高度大于 8 点时,可按每 8 点行一个图形单元分割成多个单元,不足 8 点的用空点补齐 1 2 3 4 5 6 7 8 9 10 11 12 最高位 D7 最低位 D0 02H 40H 44H 44H
删除一行	18 (CAN)	18	该命令删除该命令码之前打印缓冲区内的所有文本。它不删除该行内的任何控制码
回车	0D (CR)	0D	打印机收到本命令后,即对缓冲区内的命令和字符进行处理,按要求打印缓冲区内的全部字符或汉字
允许/禁止 16 进制形式打印	1B 22 (ESC ")	1B 22 n	如果 n=1,允许十六进制形式打印。n=0 禁止十六进制形式打印。当允许十六进制形式打印时,所有主计算机发给大于打印机的数据都将以十六进制形式打印出来,直到收到 ESC"NUL 后恢复正常打印

3.6.2 WH 系列打印机与单片机接口

WH 系列打印机与单片机的接口为串口、并口一体化,通过插拔 W1 的短路块,可使打印机与单片机的接口在并口(W1 插上时)和串口(W1 拔下时)之间切换。

可以用扁平电缆通过打印机背面的 26 芯插座与主计算机连接起来。26 芯插座各插孔的排列如图 3.46 所示。

并行接口引脚定义如下(当 W1 短路块插上时):

(1) 1 针为/STB——数据选通触发脉冲,输入信号,上升沿时打印机读入数据;

(2) 3、5、7、9、11、13、15、17 针为 DB0~DB7——数据线,单向,由主计算机输入打印机;

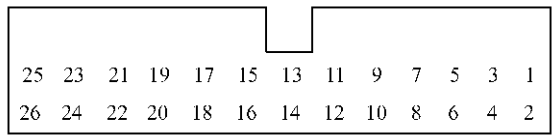


图 3.46　WH 系列打印机插座孔排列

(3) 19 针为/ACK——应答信号，是打印机输出信号，低电平有效，低电平表示打印机已接受数据；

(4) 21 针为 BUSY——"忙"信号，是打印机输出信号，高电平有效，当 BUSY = 1 时，表示打印机正在处理数据，暂时不能接受数据；

(5) 23 针为 PE——接地；

(6) 25 针为 SEL——打印机输出信号，打印机内部经电阻把 SEL 上拉至高电平时，表示打印机在线，此时打印机可以接收送来的数据；

(7) 4 针为/ERR——出错信号，是打印机输出信号，打印机内部经电阻上拉至高电平，表示打印机故障；

(8) 2、6、8、26 针为 +5V/1A 直流电源输入端；

(9) 10、12、14、16、18、20、22、24 针为 GND——接地。

串行接口引脚定义如下 (当 W1 短路块拔下时)：

(1) 19 针为 DATA——串行数据输入信号，接单片机的串行数据输出端，DATA 端的数据格式为 1 位起始位 (数据 0)，8 位数据位 (D0 ~ D7)，1 位偶；

(2) 21 针为 BUSY——"忙"信号，是打印机输出信号，高电平有效，当 BUSY = 1 时，表示打印机正在处理数据，暂时不能接受数据，该信号接单片机数据输入端；

(3) 10、12、14、16、18、20、22、24 针为 GND——接地。

图 3.47 为 WH 系列微打并行接口信号时序，图 3.48 为 WH 系列微打与 8031 并行接口，对应的打印机与 51 单片机并行接口示范驱动程序如下：

```
        BUSY    EQU     P3.3        ;BUSY 信号接/INT1 (P3.3)

        ORG     00H
        AJMP    START

        ORG     30H

START:  MOV     SP,#60H
        MOV     R7,#00H             ;R7 为查表程序中相对于表首的偏移量
AGAIN:  MOV     DPTR,#DATAS
        MOV     A,R7
        MOVC    A,@A + DPTR
        INC     R7
        CJNE    A,#0FFH,NEXT
```

```
        AJMP    START
NEXT：  CALL    PRINT
        AJMP    AGAIN

DATAS：DB       ‘1’,‘2,‘3,‘4,‘5,‘6,‘7,‘8    ;十六进制数符
        DB      ‘9,‘0,‘A,‘B,‘C,‘D,‘E,‘F
        DB      13
        DB      0ECH， 0BFH， 0BBH， 0CDH      ;炜煌
        DB      13
        DB      0FFH

PRINT： JB      BUSY，PRINT                   ;打印一个字符子函数
        MOV     DPTR,#7FFFH
        MOVX    @DPTR,A
        RET

        END
```

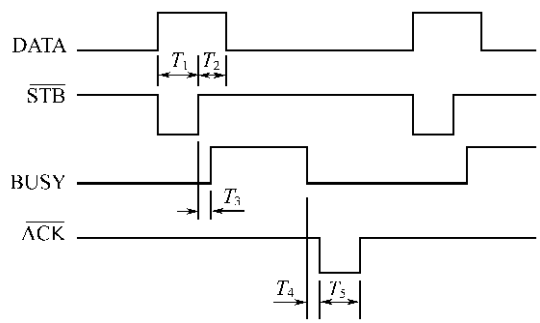


图 3.47　WH **系列微打并行接口信号时序图图**

($T_1>20\mu s, T_2>30\mu s, T_3<40\mu s, T_4<5\mu s, T_5\approx4\mu s$)

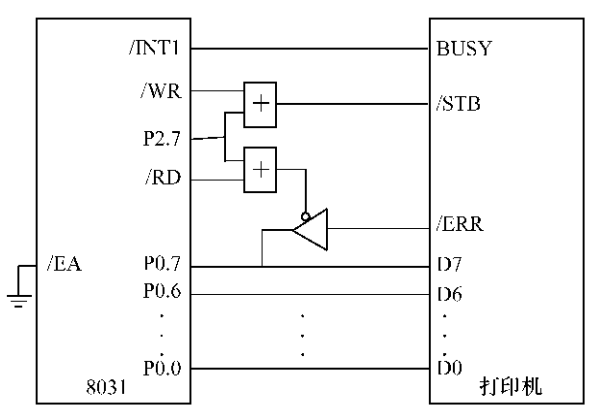


图 3.48　WH **系列微打与** 8051 **并行接口**

第 4 章　系统总线和通信接口

4.1　内 总 线

4.1.1　内总线的定义

内总线又称为局部总线，是系统内部各模块的公共信息通道，如电源地、数据线、地址线、控制线等。内总线一般采用并行总线，采用内总线有以下优点：

⑴ 各模块的设计可通用化；

⑵ 具有互换性，损坏一部分只需换该部分即可；

⑶ 只要留有足够的插口，随时可扩展系统的功能；

⑷ 改变其中一些模块可以改变仪器的功能，等等。

4.1.2　内总线举例

一、S - 100 总线

S - 100 总线由美国 MITS 公司 1976 年提出，是早期适应于 8080CPU 系列的总线，共 100 条线：数据线 16 条；地址线 24 条；控制线 11 条；DMA 线 8 条；状态线 8 条；矢量中断线 8 条；其他用途信号线 16 条；电源线地线 9 条。

S - 100 总线的主要缺陷是布线不太合理，时钟信号位于控制信号中间容易产生干扰，地线少，引脚多，几何尺寸大，易变形，目前已极少用。

二、STD 总线

STD 总线由美国 Pro - log 公司 1979 年提出，其设计是按工业现场标准设计的，比 S - 100有较高的可靠性，模块尺寸固定，长方形小板结构，具有较合理的电路设计。图 4.1 为一种 STD 总线的模板。

STD 总线共 56 条线：逻辑电源 6 条 1 ~ 6；数据线 8 条 7 ~ 14；地址线 16 条 15 ~ 30；控制线 22 条 31 ~ 52；辅助电源 4 条 53 ~ 56。

STD 总线适合于 8 位机，20 世纪 80 年代开始在我国流行，近几年采用 STD 总线设计系统也少见了，主要原因是 STD 总线引出接口多为 IDC 插头座，可靠性差，连线不方便。另一个原因是工业 PC 工业总线的迅速发展占掉了较大的市场份额。

三、PC 总线

PC 总线是基于个人 PC 机的内总线，分为商业级和工业级两种。工业级 PC 总线与商业级 PC 总线主要区别在于结构不同，工业 PC 总线采用抗震机箱和无源母板，具有较高的可靠性，又称为工控机。图 4.2 为一种工控机机箱，图 4.3 为一种工控机的无源模板。

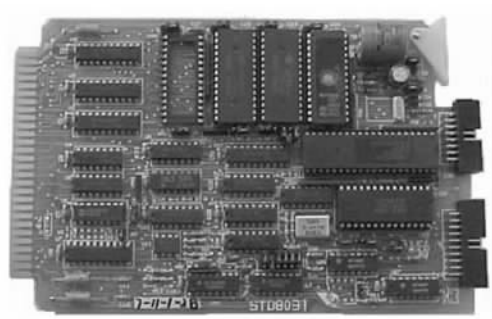


图 4.1　一种 STD 总线模板

图 4.2　一种工控机机箱

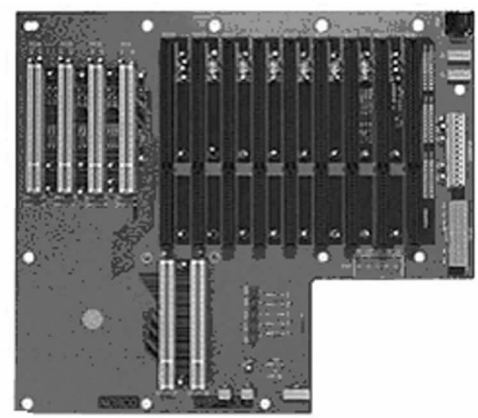


图 4.3　一种工控机母板

PC 总线的发展经历了许多阶段，版本变更也较多，下面对几种有代表性的版本作一个简介。

（一）PC/XT 总线

IBM 公司 1981 年推出了以 8088 为 CPU 的新一代个人计算机，为增加扩充能力设计了总线，该总线被称为 PC/XT 总线，传输率为 2.38MB/s。PC/XT 总线是一种开放式结构的计算机总线，底板总线有 62 个引脚，支持 8 位双向数据传输和 20 位寻址空间，有 8 个接地和电源引脚、25 个控制信号引脚、1 个保留引脚。总线底板上有 5 个系统插槽，用于 I/O 设备与 PC 机连接。该总线的特点是把 CPU 视为总线的惟一主控设备，其余外围设备均为从属设备。PC/XT 总线具有可靠简便、使用灵活等优点，并减少了自定义引脚，提高了总线的兼容性。由于 IBM 推出 PC 机较为仓促，PC/XT 总线技术也有些不足，如总线布置较为混乱，对信号完整性及频率方面考虑不够，总线频宽也较低。

（二）ISA 总线

80286 微处理器推出之后，IBM 决定开发功能比 PC/XT 更强大的 PC/AT 新机型。由于原 PC/XT 总线与新机器性能指标不匹配，同时又要保证新机型必须与 PC/XT 的原有软硬件兼容，这就要求必须对新机型的总线重新设计。IBM 公司在 PC 总线基础上增加 36 个引脚，形成了 AT 总线。即从 1982 年以后，逐步确立的 IBM 公司工业标准体系结构，简称为 ISA（Industry Standard Architecture）总线。ISA 总线传输率为 8MB/s，ISA 总线提供

了执行系统基本的存储器、I/O 和 DMA 等功能所需要的信号及定时规范。ISA 总线插头座具有 98 个引脚，包括接地和电源引脚 10 个、数据线 16 个引脚、地址线 27 个引脚、各控制信号引脚 45 个，图 4.4 为 ISA 总线的插槽。

图 4.4　ISA 总线插槽

（三）PCI 总线

PCI（Peripheral Component Interconnect）总线是当前最流行的总线之一，它是由 Intel 公司推出的一种局部总线。它定义了 32 位数据总线，且可扩展为 64 位。PCI 总线主板插槽的体积比原 ISA 总线插槽还小，支持突发读写操作，最大传输速率可达 132MB/s，可同时支持多组外围设备。PCI 局部总线不兼容 ISA 总线及 ISA 及 EISA、VESA 等 ISA 总线的变种总线。PCI 总线不受制于处理器，是基于奔腾等新一代微处理器而发展的总线。PCI 局部总线的存取延误小、总线主控及同步操作独立于 CPU，具有低成本、高效益、兼容性强等优点。图 4.5 为 PCI 总线的插槽，图 4.6 为用于 PCI 总线的一种模板。

图 4.5　PCI 总线插槽

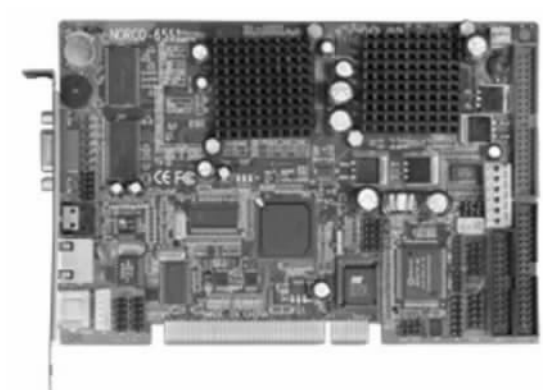


图 4.6　一种 PCI 模板

（四）PC104 工业总线

第 1 块 PC104 萌发于 1987 年，总线规范是在 1992 年才公布的，在此之前已有许多厂家生产 PC104 兼容产品。1992 年 IEEE 开始着手为 PC 和 PC/AT 总线制定一个精简的 IEEE P996 标准时将 PC104 作为 IEEE P996.1 推广。PC104 是一种专门为嵌入式控制而定义的工业控制总线。其信号定义和 PC/AT 基本一致，从逻辑上讲 PC104 总线就是 ISA 总线的翻版，但电气和机械规范却完全不同，是一种优化的、小型堆栈式结构的嵌入式控

制系统。

PC104 与普通 PC 总线控制系统的主要不同点如下。

⑴ 小尺寸结构：标准模块的机械尺寸是 3.6 英寸 ×3.8 英寸，即 96mm ×90mm；

⑵ 堆栈式连接：去掉总线背板和插板滑道，总线以“针”和“孔”形式层叠连接，即 PC104 总线模块之间总线的连接是通过上层的针和下层的孔相连，这种层叠封装有极好的抗震性；

⑶ 轻松总线驱动：减少元件数量和电源消耗，4mA 总线驱动即可使模块正常工作，每个模块能耗 1W ~ 2W。

PC104 总共有 104 个管脚，分 P1 和 P2 两部分，P1 为 64 针兼容 8 位机，P2 为 40 针，为了适应 PCI 总线，P1 和 P2 全部用上才能兼容兼容 16 位机。图 4.7 为一种 PC104 总线的模板。

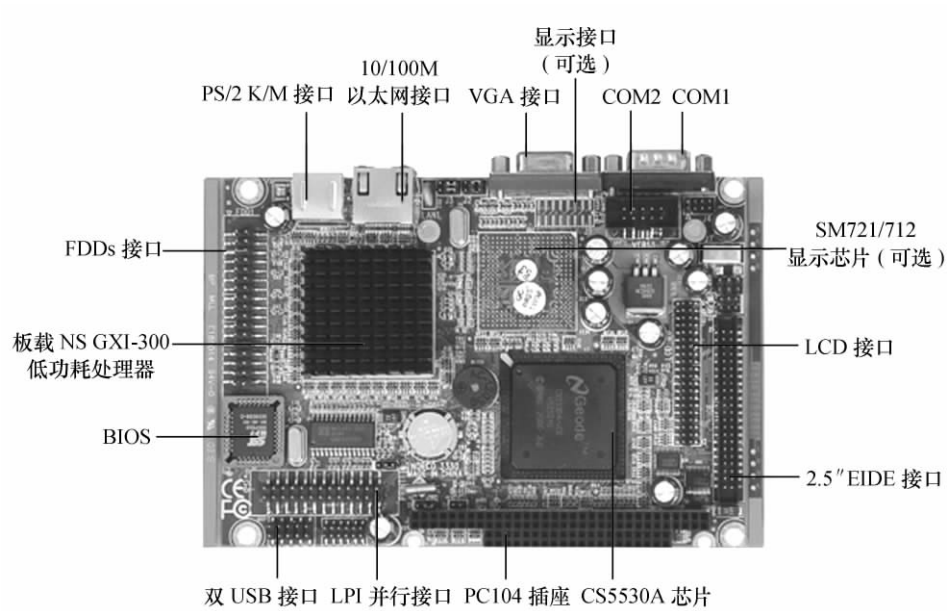


图 4.7　一种 PC104 总线模板

4.1.3　I^2C 总线

一、I^2C 总线的特点

I^2C 总线不是并行总线而是一种串行总线，其速率可达 100kb/s 以上，通信距离较短，一般用在系统内部模块或芯片之间的通信，因此我们把 I^2C 总线归为内总线。I^2C 即 Inter - Integrated Circuit的简称，是由 PHILIPS 公司开发的，产生于在 20 世纪 80 年代，最初为音频和视频设备开发，如今主要在服务器管理中使用，其中包括单个组件状态的通信。例如管理员可对各个组件进行查询，以管理系统的配置或掌握组件的功能状态，如电源和系统风扇。可随时监控内存、硬盘、网络、系统温度等多个参数，增加了系统的安全性，方便了管理。I^2C 总线最主要的优点是其简单性和有效性。由于接口直接在组件之上，因此 I^2C 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互

联成本。总线的长度可高达 25 英尺，并且能支持 40 个组件。I^2C 总线支持多主控（multimastering），其中任何能够进行发送和接收的设备都可以成为主总线。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。

二、I^2C 总线的工作原理

I^2C 总线通过 SDA（串行数据线）及 SCL（串行时钟线）两根线在连到总线上的器件之间传送信息，并根据地址识别每个器件。采用 I^2C 总线的 IC 器件，其内部不仅有 I^2C 接口电路，而且将内部各单元电路按功能划分为若干相对独立的模块，通过软件寻址实现片选，减少了器件片选线的连接。CPU 可通过指令将某个功能单元电路挂靠或摘离总线，还可对该单元的工作状况进行检测，从而实现对硬件系统的既简单又灵活的扩展与控制。I^2C 总线接口电路结构如图 4.8 所示。

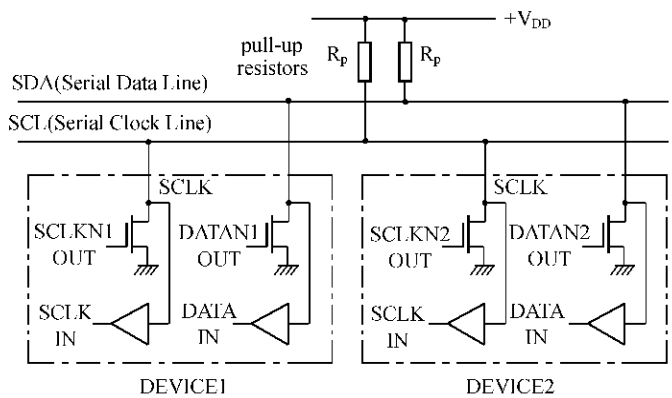


图 4.8　I^2C 总线接口电路结构

I^2C 总线则根据器件的功能通过软件程序使其可工作于发送或接收方式。当某个器件向总线上发送信息时，它就是发送器（也叫主器件），而当其从总线上接收信息时，又成为接收器（也叫从器件）。主器件用于启动总线上传送数据并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。I^2C 总线的控制完全由挂接在总线上的主器件送出的地址和数据决定。在总线上，既没有中心机，也没有优先机。总线上主和从的关系不是一成不变的，而是取决于此时数据传送的方向。SDA 和 SCL 均为双向 I/O 线，通过上拉电阻接正电源。当总线空闲时，两根线都是高电平。连接总线的器件的输出级必须是集电极或漏极开路，以具有线与功能。在 I^2C 总线上传送信息时的时钟同步信号是由挂接在 SCL 时钟线上的所有器件的逻辑“与"完成的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号下跳为低电平，将使 SCL 线一直保持低电平，使 SCL 线上的所有器件开始低电平期。此时，低电平周期短的器件的时钟由低至高的跳变并不能影响 SCL 线的状态，于是这些器件将进入高电平等待的状态。当所有器件的时钟信号都上跳为高电平时，低电平期结束，SCL 线被释放返回高电平，即所有的器件都同时开始它们的高电平期。其后，第 1 个结束高电平期的器件又将 SCL 线拉成低电平。这样就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件确定，而时钟高电平时间由时钟高电平期最短的器件确定。

三、数据的传送

SDA 线上的数据必须在时钟的高电平周期保持稳定数据线的高或低电平状态，只有在 SCL 线的时钟信号是低电平时才能改变，见图 4.9。在数据传送过程中，必须确认数据传送的开始和结束。在 I^2C 总线技术规范中，开始和结束信号（也称启动和停止信号）的定义如图 4.10 所示。当时钟线 SCL 为高电平时，数据线 SDA 由高电平跳变为低电平定义为“开始"信号；当 SCL 线为高电平时，SDA 线发生低电平到高电平的跳变为“结束"信号。开始和结束信号都是由主器件产生。在开始信号以后，总线即被认为处于忙状态；在结束信号以后的一段时间内，总线被认为是空闲的。这种主机与从机之间的连接通常是在总线的输出端，而输出端的电路结构为 I^2C 总线的从机。这种主机与从机之间的连接通常是在总线的输出端，而输出端的电路结构又总是开漏输出或集电极开路输出。

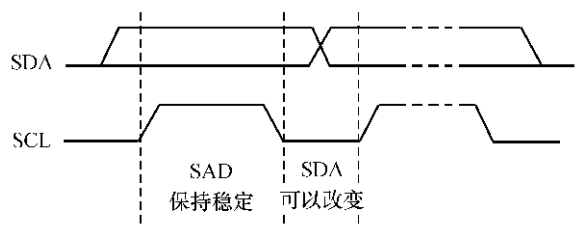


图 4.9　I^2C 总线位传输

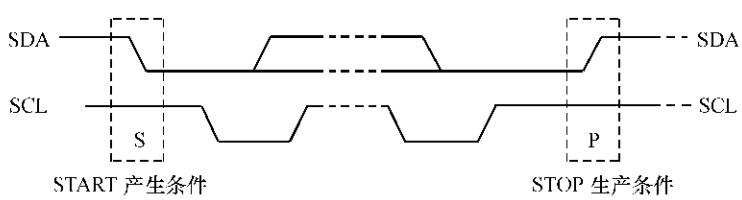


图 4.10　I^2C 起始和停止条件

通常数据传送要由主机发出启动信号和时钟信号，向所控从机发出一个地址、一个读写位和一个应答位，其中地址位为 7 位数据，在实际控制中，一般一次只能传送一个 8 位数据，并以一个停止位结束。在实际应用中，往往被传送的数据位数会超过 8 位，也就是说总会有多字节传送，这时必须在传送数据地址结束后再传送一个副地址。因此，被传送的字节没有限制，但每一个字节后面必须有一位应答位。应答位由从器件将 SDA 变低而产生，主器件在接收应答位时需将 SDA 变高。但从器件也可不应答，即应答位处于高电平。

当操作控制系统时，I^2C 总线的主机将发出启动信号，使数据线 SDA 由高电平变为低电平，同时时钟线 SCL 也发出时钟信号。I^2C 总线的数据传送格式是：在 I^2C 总线开始信号后，送出的第 1 个字节数据是用来选择从器件地址的，其中前 7 位为地址码，第 8 位为方向位（R/W）。方向位为“0"表示发送，即主器件把信息写到所选择的从器件；方向位为“1"表示主器件将从从器件读信息。开始信号后，系统中的各个器件将自己的地址和主器件送到总线上的地址进行比较，如果与主器件发送到总线上的地址一致，则该器件即为

被主器件寻址的器件，其接收信息还是发送信息则由第 8 位（R/W）确定。

在 I^2C 总线上每次传送的数据字节数不限，但每一个字节必须为 8 位，而且每个传送的字节后面必须跟一个认可位（第 9 位），也叫应答位（ACK）。数据的传送过程如图 4.11 所示。每次都是先传最高位，通常从器件在接收到每个字节后都会作出响应，即释放 SCL 线返回高电平，准备接收下一个数据字节，主器件可继续传送。如果从器件正在处理一个实时事件而不能接收数据时（例如正在处理一个内部中断，在这个中断处理完之前就不能接收 I^2C 总线上的数据字节），可以使时钟 SCL 线保持低电平，从器件必须使 SDA 保持高电平，此时主器件产生一个结束信号，使传送异常结束，迫使主器件处于等待状态。当从器件处理完毕时将释放 SCL 线，主器件继续传送。

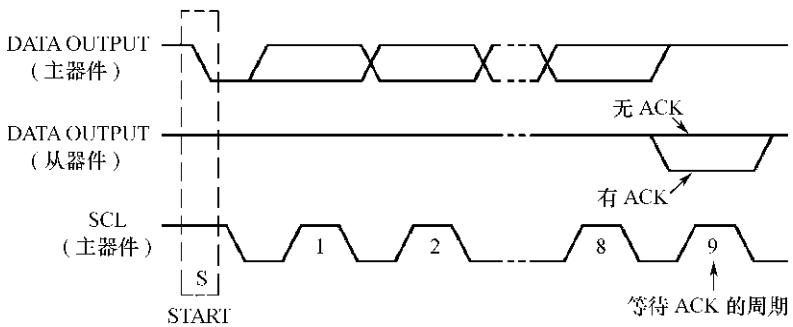


图 4.11　I^2C 总线的响应过程

四、I^2C 接口举例

（一）硬件连接

X24C04 是 XICOR 公司的 CMOS 4096 位串行 EEPROM，内部组织成 512 × 8 位，工作

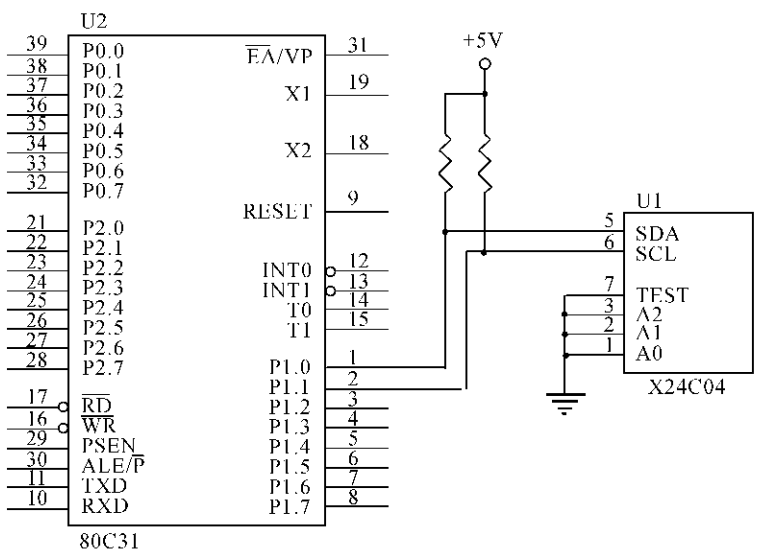


图 4.12　X24C04 与单片机的接口

于从器件方式，每个字节可擦写 100 万次，数据保存时间大于 40 年。与 MCS - 51 单片机接口如图 4.12 所示，X24C04 有 8 个管脚，电源和地是 8 脚和 4 脚，SDA 接 P1.0，SCL 接 P1.1，TEST 在使用中须接地，A2、A1 决定芯片在系统中的地址，这里选择“00”，A0 也是地址，但对 X24C04 来讲无用，必须接地。这样主器件对 X24C04 的操作命令如图 4.13 所示，其中 1010 为 X24C04 的器件地址（Device Type Identifier）；A2、A1 的值与管脚值（Device Address）相同；BA 为字节选择（Hihg Order Word Address），BA = 0 时表示对低 256 字节操作，BA = 1 时表示对高 256 字节操作；其它如上所述。

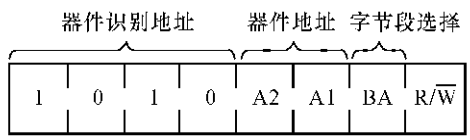


图 4.13　X24C04 的控制命令

（二）随机单字节写

如图 4.14 所示，在开始后先发写命令（10100000B），等到回应（ACK）后再发要写入的地址，等到回应后继发要写入的数据，等到回应后再发结束信号。

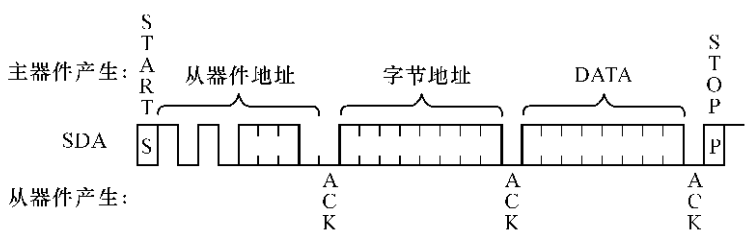


图 4.14　向 X24C04 某一地址写一个字节

（三）写一组数据

如图 4.15 所示，在开始后先发写命令（10100000B），等到回应（ACK）后再发要写入的地址，等到回应后连续发要写的数据（不超过 16 个），每个数据必须等到回应，最后发结束信号。

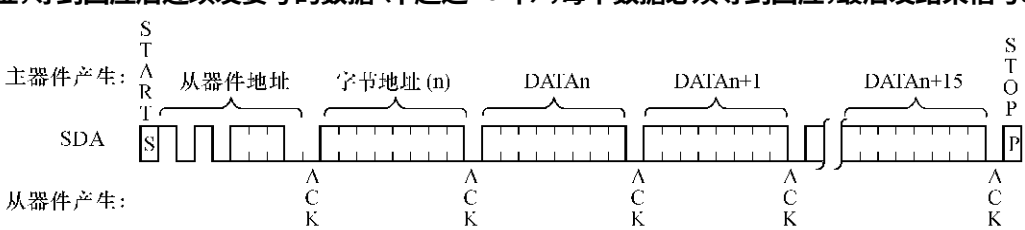


图 4.15　向 X24C04 写一串数据

（四）读当前地址的数据

如图 4.16 **所示，在开始后先发读命令（**10100001B**），等到回应（**ACK**）后接收一个字节，发一个回应或等待一个节拍发结束信号。**

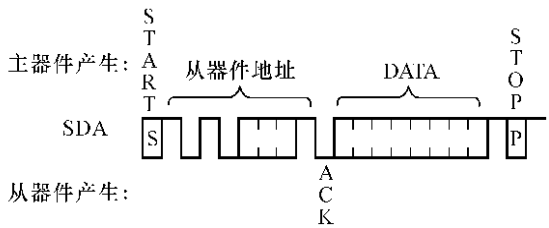


图 4.16　**从** X24C04 **当前地址读一个字节**

（五）从指定地址读一个或一组数据

如图 4.17 所示，在开始后先发写命令（101000000B），回应（ACK）后发要读的地址，等到回应后发开始及读命令（101000001B），等到回应后接收一个字节，发一个回应或等待一个节拍发结束信号，在此过程中接收到第 1 个字节后也可继续接收下一个字节，但同样不能超过 16 个字节，如图 4.18 所示。

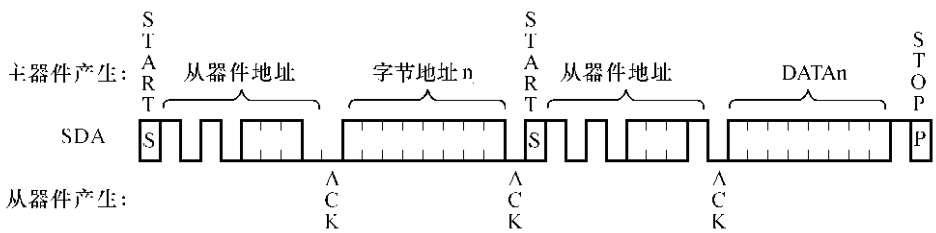


图 4.17　向 X24C04 某一地址读一个字节

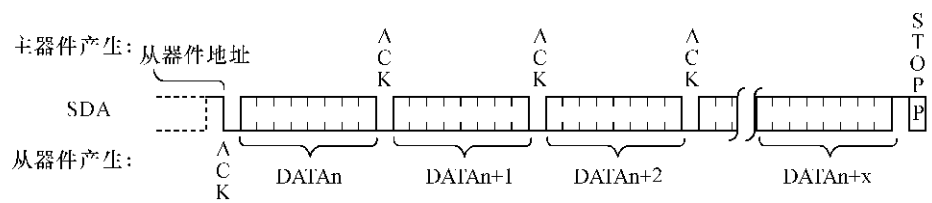


图 4.18　从 X24C04 读一串数据

在图 4.17 中所示，在开始后先发写命令（101000000B），等到回应（ACK）后发要读的地址，再发读命令（101000001B），等到回应后接收一个字节，发一个回应或等待一个节拍发结束信号。

（六）程序举例

如果单片机具有 I^2C 口，对其初始化后直接对口地址操作就可以完成相应的读写，包括单字节读写、多字节读写等，随着单片机的不同，其操作方式也略有不同，这里不在赘述。当单片机没有 I^2C 口时需要编写模拟 I^2C 口的程序，完成对 I^2C 器件的控制，而这样的模拟程序更能反映 I^2C 总线的工作过程。下面是用 C51 编写的向指定地址写一个字节和从指定地址读一个字节的程序。

```
sbit   scl  =  P1 1；
sbit   sda  =  P1 0；              //X24C04 I²C 端

//- - - - - - - - - - - - - - - - - - - - - - - - -
//向指定地址写一个字节的程序
//- - - - - - - - - - - - - - - - - - - - - - - - -
bit c04 _ write (unsigned char address，unsigned char write _ v)        //8
 {
   ok = 0；                  //ok = 0 success
   scl  =  0；               //
```

```
    make _ stop ( ) ;          / / init c04
    make _ start ( ) ;
    byte _ w (0xa0) ;          / / slAve Add  = 10100000B , B0 = 0 写入
    byte _ w (address) ;
    byte _ w (write _ v) ;
    make _ stop ( ) ;
    return  (ok) ;
}
/ / - - - - - - - - - - - - - - - - - - - - - - - -
/ / 从指定地址读一个字节的程序
/ / - - - - - - - - - - - - - - - - - - - - - - - -
unsigned char c04 _ read (unsigned char address)
 {
    unsigned char v ;
    scl  =  0 ;                / /
    make _ stop ( ) ;          / / init c04
    make _ start ( ) ;
    byte _ w (0xa0) ;          / / slAve Add  = 10100000B , B0 = 0 写入
    byte _ w (address) ;
    make _ start ( ) ;
    byte _ w (0xa1) ;          / / slAve Add  = 10100001B , B0 = 1 读出
    v  =  byte _ r ( ) ;
    make _ stop ( ) ;
    return  (v) ;
}
/ / - - - - - - - - - - - - - - - - - - - - - - - -
/ / 产生“start”的程序
/ / - - - - - - - - - - - - - - - - - - - - - - - -

void make _ start (void)
 {
    scl  =  0 ;
    delay (100) ;
    sda  =  1 ;
    delay (100) ;
    scl  =  1 ;
    delay (100) ;
    sda  =  0 ;
    delay (100) ;
```

```
    scl = 0;
    delay(100);
}
//-----------------------
//产生"stop"的程序
//-----------------------
void make_stop(void)
{
    sda = 0;
    delay(100);
    scl = 1;
    delay(100);
    sda = 1;
    delay(100);
    scl = 0;
    delay(100);              //continue controlling the bus
}

//-----------------------
//延时程序
//-----------------------
void delay(unsigned int i)       //11
{
    do
    {
    i--;
    }
    while(i > 0);
}
//-----------------------
//有回应的读一个字节的子程序
//-----------------------
//from c04 read a byte have ACK
unsigned char byte_r(void)
{
    unsigned char read_v = 0;
    register char i;
    for(i=0;i<8;i++)
    {
```

```
        read _ v * = 2;
        scl = 1;
        delay(100);
        if(sda) read _ v + +;
        scl = 0;
        delay(100);
    }
    sda =1;                    //产生一个 ACK
    delay(2);
    sda = 0;
    scl = 1;
    delay(20);
    scl = 0;
    delay(20);
    sda = 1;
    delay(20);
    return(read _ v);

}
//- - - - - - - - - - - - - - - - - - - - - -
//写一个字节并等待回应的子程序
//- - - - - - - - - - - - - - - - - - - - - -
bit byte _ w(unsigned char write _ v)
 {
    bit ack = 0;
    register char i;
    for(i =0;i <8;i + +)
       {
         delay(100);
         if(write _ v & 0x80) sda =1;
         else sda =0;
         delay(100);
         write _ v = write _ v ≪ 1;
         scl = 1;
         delay(100);
         scl = 0;
       }
    i = 0;
    sda = 1;
```

```
    delay(10);
    scl = 1;
    do {
      i++;
      delay(80);
      ack = sda;
      }
    while (ack == 1 && i<255);
    scl = 0;
    delay(100);
    return (ack);
}
```

4.1.4 内总线的选择原则

(1) 尽可能用通用性强的标准总线,这样的总线产品生产批量大,成本低,可维护性强,技术支持多。

(2) 在满足使用条件下,选择线数较小、几何尺寸较小、接插件可靠的总线,这样可减少体积,增加可靠性,降低成本。

(3) 没有合适的总线选取时,可以自行设计非标准总线。

(4) 如所设计系统规模不大、接口少、功能单一、没有可扩展性等要求时,尽可能不用总线,以提高系统可靠性,降低系统成本。

4.2 系统的外总线

外总线系指系统除具有能够执行所有本地任务的功能外专门设立的与外部设备交换数据或信息的接口,因此外总线又称通信接口。

4.2.1 GP-IB 通用接口简介

一、GP-IB 接口的特点

GP-IB 是 General Purpose Interface Bus 的简称,GP-IB 最早由美国 HP 公司研制,1975 年 IEEE 将其改进,制定 IEEE488 标准,目前较多称为 GP-IB。

GP-IB 协议又称听者/讲者/控者协议,在工作时必须有一个控者,每一时刻只能有一个讲者,可有多个听者,其物理层是 24 芯叠式结构插头座,电缆为 24 芯,系统中可以连接的仪器不超过 15 台,互连长度不超过 20m,传送方式为并行方式,最大速率 1MB/s,适应于轻微干扰的试验室或现场。

二、信号线定义

GP-IB 接口的连接形式如图 4.19 所示,GP-IB 接口的 24 条导线分为 8 条数据线、3 条挂钩联络线及 5 条接口管理线,共 16 条信号线,其余为逻辑地线。

(1) 数据线 8 条:$D_1 \sim D_8$,双向,数据、命令及地址都用它。

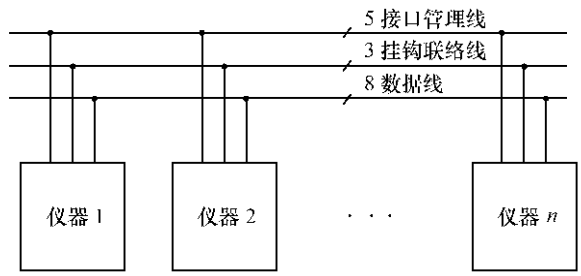


图 4.19　GP - IB 接口的连接形式

⑵ 数据挂钩联络线 3 条：

DAV (DATA VALID) 数据有效线，低电平表示有效，当数据线上出现有效数据时，讲者置其为低电平，示意听者接收数据。

NRFD (NOT READY FOR DATA) 数据未就绪线，被指定的听者中有一个未准备好接收数据，NRFD 就为低，它示意讲者暂不要发出信息。

NDAC (NOT DATA ACCEPTED) 数据未收到线，被指定的听者中有一个听者未收到数据就为低，它示意讲者保持数据线上的信息。

⑶ 接口管理线 5 条：

ATN (ATTENTION) 控者用注意线：用来指明数据线上的数据类型，ATN 为低电平，表示 $D_1 \sim D_8$ 为接口信息，反之为仪器信息。接口信息表示挂在总线上的信息为控制仪器的信息如命令、设备地址等，而仪器信息是挂在总线上的信息为各仪器真正需要交换的数据信息，如数据、远程控制命令等。

IFC (INTERFACE CLEAR) 控者用接口清除线，IFC 为低电平接口复位。

REN (REMOTE ENABLE) 控者用远程控制线，REN 为低电平表示仪器处于远程工作状态，面板手工操作停用；REN 为高电平表示仪器处于本地工作方式。

SRQ (SERVICE REQUEST) 服务请求线，所有设备与该线线与，任一设备将该线变低表示向控者申请服务。

EOI (END OR IDENTIFY) 结束或识别线，EOI 与 ATN 配合使用，在 EOI 为低、ANE 为高时表示讲者已传完一组数据；在 EOI 为低、ANE 为低时控者要进行识别操作，各设备将其状态放在数据线上。

三、工作过程描述

GP - IB 口将许多设备连到一起时，必须首先指定一个控者，整个通信过程按控者由用户预先编好的程序来运行。一般情况下控者是由带计算机的设备来完成的，控者规定哪个是讲者，哪些是听者。然后在控者的控制下，在数据线上通过接口信息协调各仪器的接口操作进而达到交换仪器信息的目的。举例来说，如图 4.20 所示，实现对一个待测放大器的幅频特性进行测量，并将测试结果打印出来。计算机令信号发送器产生幅值固定、频率可变的正弦信号，由频率计测出信号的频率，由数字电压表测出放大器的输出幅值，所测多组结果送给计算机计算后，求得幅频特性并交给打印机打印出来。工作流程如下：

⑴ 控制器发出 REN 消息，使系统中所有装置都处于控者控制之下。

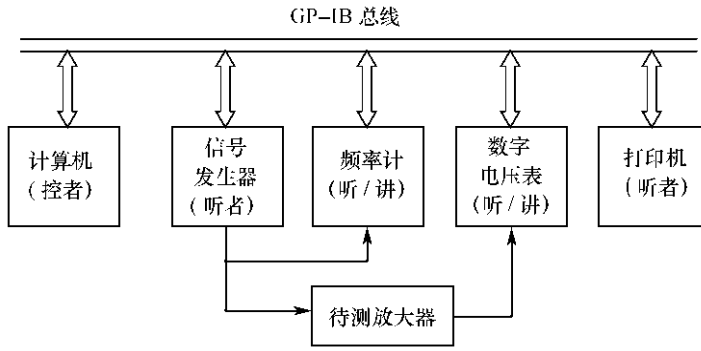


图 4.20　GP - IB 总线应用举例

⑵ 控制器发出 IFC 消息,使系统中所有装置都处于初始状态。

⑶ 控制器发出信号发送器的听地址,信号发送器接收地址后成为听者。

⑷ 控制器向信号发送器发出一个程控命令,使信号发送器输出一个指定范围频率的正弦信号,幅值固定为一个值(如 100mV)。

⑸ 控制器取消信号发送器的听受命状态。

⑹ 控制器发频率计的听地址,频率计成为听者后测量输入信号的频率。

⑺ 控制器发频率计的讲地址,取消频率计的听受命状态,控制器使自己变为听者接收由频率计发来的频率测量值。

⑻ 控制器发数字电压表的听地址,数字电压表成为听者后测量输出信号的幅值。

⑼ 控制器发数字电压表的讲地址,取消数字电压表的听受命状态,控制器使自己变为听者接收由数字电压表发来的幅值测量值。

从⑶ ~ ⑼测的一组值,不断重复这个过程可得到多组测量值。设测到要求的组数后,步骤序号为 X,则

……

(X) 控制器计算好描述幅频特性的一组数据,发打印机的听地址,控制器作为讲者把数据发给打印机,并命令打印机打出幅频特性。

四、GP - IB 接口评价

GP - IB 最早是为仪器之间通信设计的,在电气上采用并行数据连接,在当时背景下比串行通信要快很多,因此在仪器通信即其它检测系统的近距离通信中发挥了巨大的作用。随着串行通信技术的不断提高,GP - IB 通信也受到了巨大的挑战,这是因为串行通信除了具有连接简单、成本低、距离远等优点外,速度也在不断提高,目前有的串行接口已经远远超过了 GP - IB 的通信速度。所以近年来除仪器行业以外,人们在设计新的检测系统时基本上不考虑使用 GP - IB 接口,除非用户有特殊要求。仪器行业因历史原因为了仪器之间的配套,即使是新设计的仪器有时也不得不配置 GP - IB 接口,所以 GP - IB 接口同样有自己的用武之地。也许 GP - IB 接口在考虑到机械及电气的兼容性的前提下会有飞跃性的改进。

4.2.2　RS－232C 简介

一、RS - 232 概述

RS - 232C 是美国电气工业协会 EIA (Electronic Industries Association) 1969 年推荐的在异步串行通信 (Universal Asynchronous Receiver & Transmitter,UART) 中应用较广的总线,其中 RS——Recommended Standard,232——标识符,C——表示经过三次修改。RS - 232C 全称是"使用二进制进行交换数据的终端设备 DTE (Data Terminal Equipment) 和数据通信设备 DCE (Data Communication Equipment) 之间的接口",计算机、外设、显示终端等为 DTE,而调制解调器等为 DCE。图 4.21 指明 RS - 232C 在通信电路中的位置。

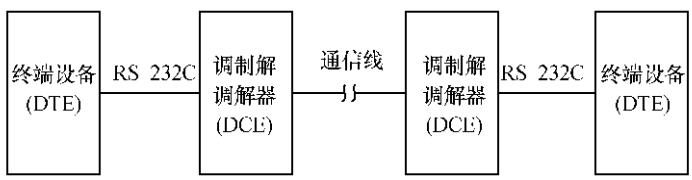


图 4.21　RS - 232C 在电路中的位置

RS - 232C 接口为 25 针或 9 针的 D 型插座,近几年为了节省空间,9 针座用者居多。各插针定义见表 4.1。RS - 232C 接口在工作时,逻辑 0 电平为 +5V ~ +15V,逻辑 1 电平为 - 5V ~ - 15V;RS - 232C 的最大传输速率为 20KB/s,相应的最大传输距离为 15m;串行通信分为同步方式和异步方式,一般使用异步方式;通信介质可选同轴电缆、双绞线、光纤等。

表 4.1　RS - 232C 信号定义

25 针	9 针	简称	各称 (传输方向)	功　能
1	1	保护地	保护地	连外壳
2	3	TXD	发送 DTE→DCE	DTE 发送串行数据端
3	2	RXD	接收 DTE←DCE	DTE 接受串行数据端
4	7	RTS	请求发送 DTE→DCE	DTE 请求发送
5	8	CTS	清除发送 DTE←DCE	DCE 已准备好接收 (清除发送)
6	6	DSR	数传设备就绪 DTE←DCE	DCE 准备就绪
7	5		信号地	
8	1	DCD	数据载波检测 DTE←DCE	DCE 接收到远程信号
20	4	DTR	终端就绪 DTE→DCE	DTE 准备就绪
22	9	RI	振铃指示 DTE←DCE	DCE 告 DTE 线路已妥

二、RS - 232C 在系统的连接

(一) 远程连接

当通信距离超过 15m 时就要采用远程连接,图 4.22 为一种常见的连接方式。远距离信息传送靠两个 MODEM 之间的通信介质完成,通信距离取决于介质的性能和波特率的高低。RTS、DSR、CTS、DTR 用来进行数据交换前的握手。

(二) 近程连接

在通信距离小于 15m 时,可省去 MODEM 直接用 RS - 232C 信号线相连。图 4.23 为

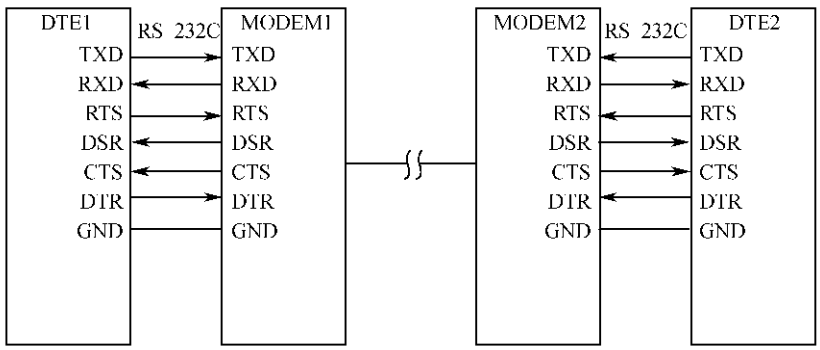


图 4.22　RS - 232C 远程连接方式

带握手通信的连接方式，图 4.24 为不带握手通信的连接方式，这种方式又称三线方式，因连线简单，应用也最为广泛。

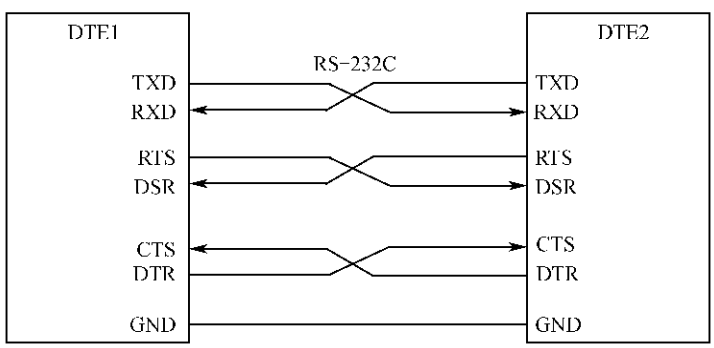


图 4.23　带握手的近程连接方式

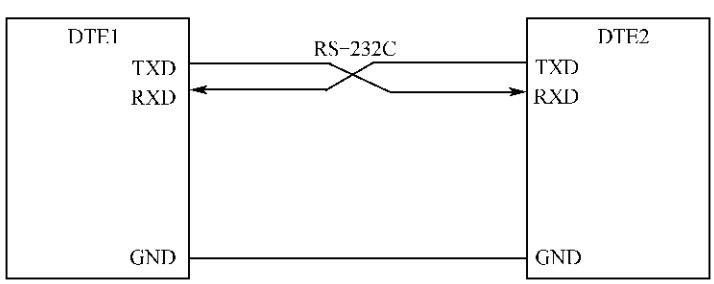


图 4.24　不带握手的近程连接方式

三、单片机的 RS - 232C 接口设计

几乎所有的单片机都带有异步数据收发口（UART），有的带有一个以上的 UART，如 C8051F020 带有两个独立的 UART。这里要强调指出，UART 口并不是 RS - 232C 口，UART 可以转换成 RS - 232C，也可以转换成 RS - 422、RS - 485 等其它接口。UART 的通信协议在许多教科书上都有详尽的叙述，这里只介绍 UART 和 RS - 232C 接口的转换。

由单片机组成的 RS - 232C 接口大都采用三线方式连接，由于 UART 也是三线，因此只需电平转换就可实现其功能。图 4.25 为早期的 UART 和 RS - 232C 的转换电路，其中 MC1488 为发送器，MC1489 为接收器，发送器需用 ±12V 电源，接口电路必需有 +5V 电源和 ±12V 电源，这将增加接口的体积和成本。为此有的公司开发了单一 +5V 电源供电的发送器，最典型的是 MAXIM 公司的收发器 MAX232，图 4.26 为该芯片的基本使用连接电路。MAX232 芯片包含一套 +5V 电源转 ±12V 电源的电路，只需外接 4 个小电容就可以工作。它把一个发送器和一个接收器做到一起，一片 MAX232 就可使 UART 变为三线 RS - 232C 口，因此 MAX232 系列芯片很受欢迎。

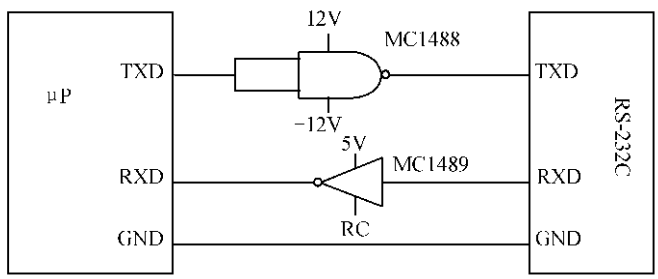


图 4.25　早期的 UART 和 RS - 232C 的转换电路

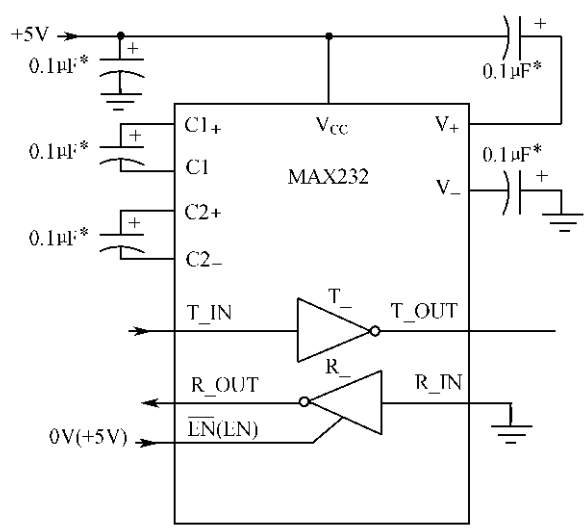


图 4.26　MAX232 的基本使用连接电路

4.2.3　RS422/485 通信接口

一、RS - 449

RS - 232C 通过电平转换使信号从驱动方到达接收方后有较大的幅值，保证传送数据的准确性。但信号是单端的对地信号易于引入附加电平。附加电平的来源有两方面，如图 4.27 所示，其一为线路中的干扰 e_n，其二为驱动器和接收器之间的电位差 V_s。当 e_n 和

V_s 的共同作用使接收端的电平达不到规定值时，传输信息将出现错误，在干扰较大时通信根本就无法进行，因而单端传送信号通信方式其通信速度不可能有较大的提高。为此美国电气学会 EIA (Electronic Industres Association) 在 1977 年制定了 RS - 449 串行通信接口，完全兼容 RS - 232，且传输速率快。RS - 449 设置有两种标准接口，一种为 36 脚，一种为 9 脚。

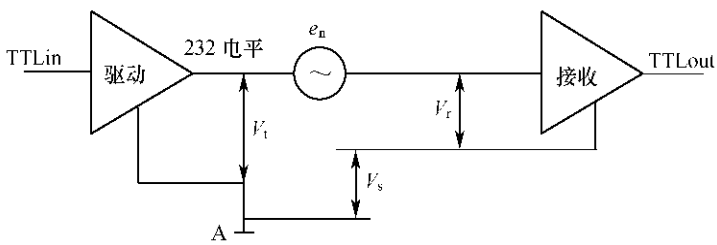


图 4.27　单端信号中的附加电平

RS - 449 和 RS - 232 的最大区别在于信号在导线上传输方法不同。RS - 449 采用差分信号传送，如图 4.28 所示，信号驱动端采用平衡驱动器，信号 A 和信号 B 有一个共同的参考点并以相反的方向变化，二者的差值作为有用驱动信号。由于传输导线往往采用双绞线，当有干扰时，叠加在两根导线上的干扰 e_{nA} 和 e_{nB} 相差不大，对有用信号的影响较小，此外这种差分传输还可以消除驱动点和接收点之间的电位差，因此这种方法传输距离远，传输速度高。例如在 1200m 的双绞线上其传输速率可达 90KB/s。目前使用最多的是 RS - 449 的子集 RS422 和 RS485。

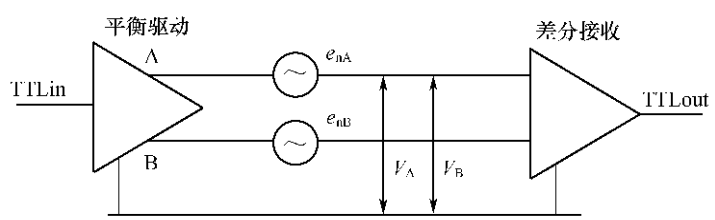


图 4.28　双端信号中的附加电平可抵消

二、RS - 422

典型的 RS - 422 接口是四线传输接口，它允许在相同传输线上连接多个接收节点，最多可接 10 个节点。即一个主设备 (Master)，其余为从设备 (Salve)，从设备之间不能通信，所以 RS - 422 支持点对多的双向通信。RS - 422 四线接口由于采用单独的发送和接收通道，可实现全双工通信，RS - 422 的最大传输速率为 10MB/s，当采用双绞线时在 1200m 时最大传输速率为 90KB/s，在 120m 时可达 1MB/s。双绞线的长度与传输速率成反比。只有在很短的距离下才能获得最高速率传输。RS - 422 需要一终接电阻，要求其阻值约等于传输电缆的特性阻抗。在 300m 以下不需终接电阻，终接电阻接在传输电缆的最远端，习惯上终端电阻取 120Ω。

常用的 RS - 422 收发器是 MAX488、MAX490，图 4.29 为 MAX488/490 的管脚图，图

4.30 为用 MAX488/490 组成的 RS - 422 接口点对点通信连线图。

三、RS485

RS - 485 是从 RS - 422 基础上发展而来的，是 RS - 422 的变种，RS - 485 许多电气规定与 RS - 422 相近。RS - 485 可以采用二线或四线方式，四线制可实现全双工，二线制只能半双工。二线制 RS - 485 连线简单方便，成本又低，因此使用较普遍。RS - 422 的传输电平在 - 7V ~ +7V 之间，而 RS - 485 传输电平是 - 7V ~ +12V 之间，一个发送器可驱动 32 个接收器，总线上可接到 32 个设备。

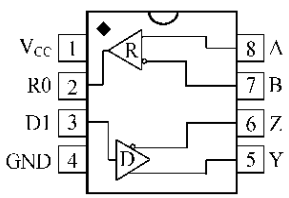


图 4.29　MAX488/490 管脚图

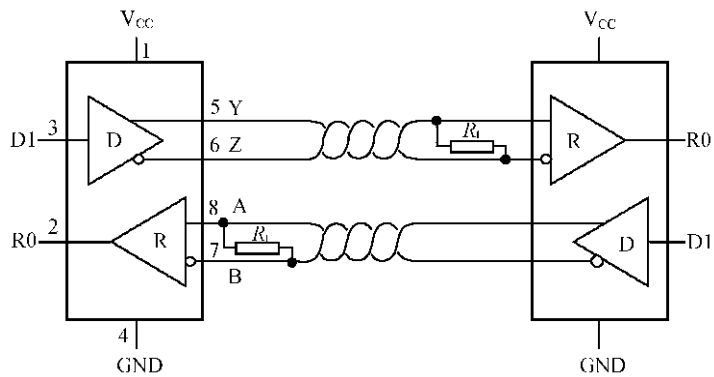


图 4.30　用 MAX488/490 组成的 RS - 422 通信电路

在 RS - 485 总线上通过程序的协调，每台设备都可以实现接收或发送功能，距离较远时每台设备都应有终端电阻，RS - 485 传输速率与 RS - 485 相同。

RS - 485 的产生反过来影响了 RS - 422，许多厂家进而推出了兼容二者的收发器，这使得 RS - 422 和 RS - 485 的界限变得模糊起来。

常见的 RS - 485 收发器是 MAX481/3/5/7，图 4.31 为 MAX481/3/5/7 的管脚图，图 4.32 为用 MAX481/3/5/7 组成的 RS - 485 接口点对点通信连线图，图 4.33 为用 MAX481/3/5/7 组成的 RS - 485 点对多通信连线图。

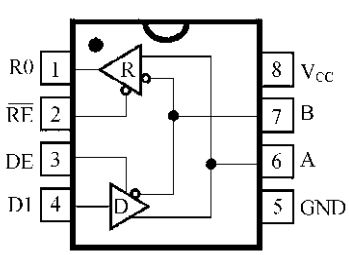


图 4.31　MAX481/3/5/7 管脚图

需要指出的是，作为半双工工作的 RS - 485 接口，在同一时刻发送时不可接收，接收时不可发送。设备的端口在接收时应将自己的发送端关闭令 $D_E = 0$ 在发送时将自己的接收端关闭，令 $R_E = 1$。请读者自行写一段如图 4.33 多机通信的程序。

4.2.4　USB 通用串行总线

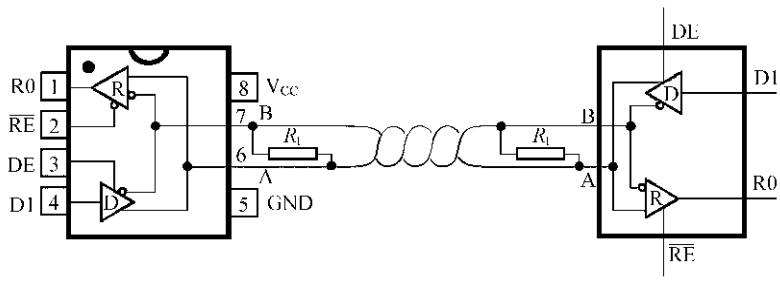


图 4.32　两台设备的 RS - 485 连线图

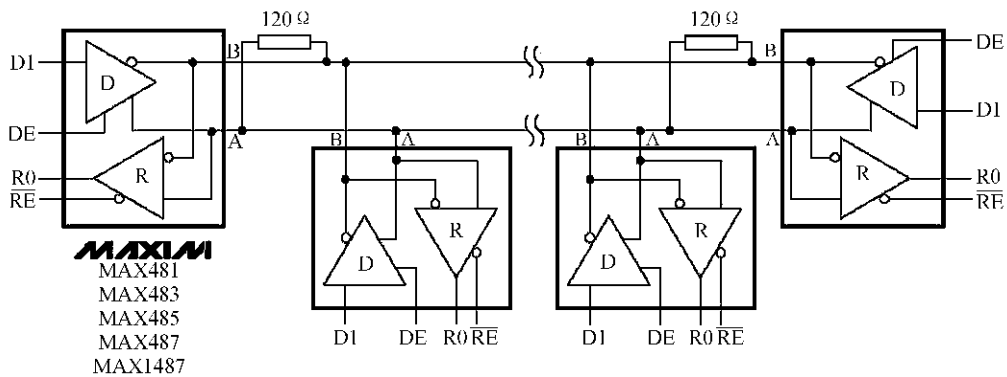


图 4.33　多台设备的 RS - 485 连线图

一、USB 协议概述

USB（Universal Serial Bus）总线协议是以 Intel 及 Compaq、Microsoft、IBM、NEC 等共 7 家公司在 1994 共同制定并推出的串行接口标准，目前流行的是 1.1 版本及 2.0 版本。USB 可以连接 127 个外设，1.1 版本数据传输率可达 12Mb/s，而 2.0 版本数据传输率可达 480Mb/s。USB 允许外设在主机和其它外设工作时进行连接配置使用及移除，即所谓的即插即用（Plug & Play），同时 USB 总线的应用可以清除 PC 上过多的 I/O 端口而以一个串行通道取代，使系统与外设之间的连接更容易。

二、USB 的总线拓扑结构

USB 总线的物理连接是一种分层（Tier）的菊花链结构集线器（HUB），每个星形结构的中心机就是主机的根（Hub），用户可以将外设节点（Node）或附加的 Hub 与之相连，这些附加的 Hub 可以连接另外的外设以及下层 Hub。USB 支持最多 5 个 Hub 层以及 127 个外设，图 4.34 描述了 USB 的物理拓扑结构。

三、USB 的物理层

USB 的物理接口包括电气结构和机械结构。USB 通过一个四线电缆来传输信号与电源，如图 4.35 所示。其中 D_+ 和 D_- 是一对差分的信号线，而 VBus 和 GND 则提供了 5V 的电源，它可以给一些设备（包括 Hub）提供一定的电流。USB 信号线在高速模式下必须使用带有屏蔽的双绞线，而且最长不能超过 5m，而在低速模式时中可以使用不带屏蔽或不是双绞的线，但最长不能超过 3m。

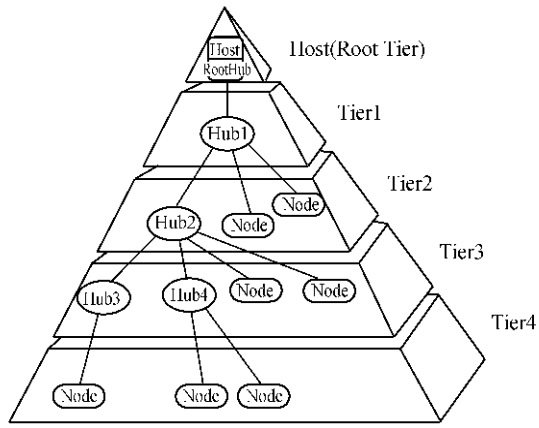


图 4.34　USB 口的拓扑结构

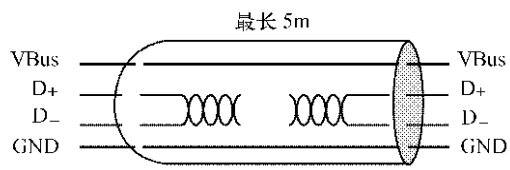


图 4.35　USB 口连接定义

四、USB 接口中的几个概念

（一）USB 主机及 SUB 设备

USB 主机是指在 USB 系统中处于中心地位并且对 USB 接口及其连接的设备管理的主导设备，而其它连接的设备称为 USB 设备。主机控制着所有的 USB 设备，只有主机允许，USB 设备才有权利访问主机。

（二）端点和管道

端点（Endpoint）是 USB 系统用来交换数据的特定逻辑地址，这个端点不止一个，每个端点都有自己的特性和用途，对主机来说，不同的端点实际上就是对应的不同的数据缓冲区 对设备来说，不同的端点对应不同的硬件电路。主机只能通过端点与设备进行通信。每个端点在设备出厂时就已定义好，在 USB 系统中每一个端点都有惟一的地址，每个端点都有一定的特性，这些特性包括端点号、传输方式、总线访问频率、带宽、数据包的最大容量等。除端点 0 外，端点必须在设备配置后才能生效，端点 0 通常为控制端点，用于设备初始化参数等，真正的数据传输是通过其它端点进行的。

管道（Pipe）是 USB 系统通信驱动程序和端点组成的通信通道，每个设备在配置后有一个惟一的管道，管道只有主机和设备连接配置生效后才能形成。管道的序列号是主机临时给定的，当设备从主机移去时管道同时取消。

（三）USB 设备的描述符

USB 设备的描述符是对 USB 的属性说明。标准 USB 设备有 5 种 USB 描述符，分别

是设备描述符、配置描述符、接口描述符、端点描述符和字符串描述符。USB 描述符是通过端点 0 来读取的。

五、USB 接口的结构

USB 接口的结构如图 4.36 所示，无论是 USB 主机还是 USB 设备，都可以大致分为 3 个层次，即 USB 接口控制器及串行通信口层、USB 接口管理程序层和用户通信程序层。其中 USB 接口控制器及串行通信口是 USB 接口的主体部分，它按照相应的 USB 协议用硬件电路来完成数据包的收发和与本机的信息交换；USB 接口管理程序负责与 USB 接口控制器的信息交换，它是 USB 通信的最底层程序，它包括 USB 口具备功能的所有程序模块如控制器初始化、数据包的读写、状态反馈、端点切换等；用户通信程序是指面向高层用户的专用程序，它是为完成特定的任务而调用 USB 接口管理程序的集合。

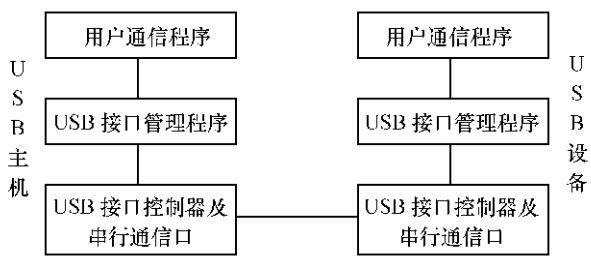


图 4.36　USB 接口的结构

六、USB 接口工作过程

USB 设备可以即插即用，但在使用之前必须对设备进行配置，一旦设备连接到某一个 USB 的节点上，USB 就会产生一系列的操作来完成对设备的配置，这种操作被称为总线枚举过程。只有枚举成功了，接口才能正常工作。USB 的工作过程如下。

⑴ USB 设备接入节点时（无源设备插入主机节点或有源设备重新供电），所连接的节点检测出端口上有设备接入向主机报告；

⑵ 主机通过询问节点以获取确切的信息；

⑶ 主机得知设备连接到哪个节点上并向这个节点发出复位命令；

⑷ 设备上电所有的寄存器复位并且以缺省地址 0 以及端点 0 响应命令；

⑸ 主机通过缺省地址与端点 0 进行通信并赋予设备空闲的地址，以后设备就对该地址进行响应；

⑹ 主机读取设备描述符确认设备的属性；

⑺ 主机依照读取的 USB 描述符来进行配置，如果设备所需的 USB 资源得以满足，就发送配置命令给设备，该设备就可以使用了，枚举过程结束；

⑻ 当通信任务完成后，该设备被移走时（无源设备的拔出主机节点或有源设备断电），节点依然要报告主机，主机关闭端口释放相应资源。

七、基于 μP 的 USB 接口的设计

USB 接口有诸多优点，一个智能检测系统有时也需要设置 USB 接口。比如说，为 μP 设置一个 USB 接口是多数 μP 应用者的共同心愿。值得欣慰的是，有的 μP 厂商已捷足先登，推出了具有 USB 接口的产品，如 Silabs 公司的 C8051F360/1 就有一个 USB 口，这类

产品只要按照其使用手册编程即可实现 USB 接口功能。但目前大部分 μP 还不具备 USB 口，需通过专用芯片实现其 USB 接口功能。

有多家公司不断推出 USB 接口专用芯片，其中较早推出的是 PHILIPS 公司的产品，这里我们以其一款典型的符合 USB1.1 协议的芯片 PDIUSBD12 为例，来说明 μP 的 USB 接口设计思路。

(一) DIUSB12 芯片简介

PDIUSB12 是一个性能较优的 USB 器件，通常用于微控制器系统，通过并行接口与微控制器进行通信，而且支持 DMA 传输。该器件采用模块化的方法实现了一个 USB 接口，能适应大多数设备类规范的设计，如成像类、大容量存储类、通信类、打印类和人工输入设备等。PDIUSB12 内部结构如图 4.37 所示，各部分说明如下。

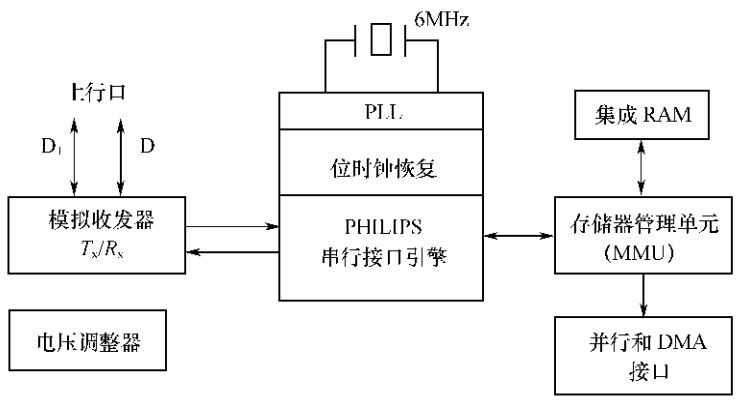


图 4.37　PDIUSBD12 内部结构图

(1) 模拟收发器：集成的收发器接口可通过终端电阻直接与 USB 电缆相连。

(2) 电压调整器：片内集成了一个 3.3V 的调整器用于模拟收发器的供电，该电压还作为输出连接到外部 1.5kΩ 的上拉电阻，可选择 PDIUSBD12 提供的带 1.5kΩ 内部上拉电阻的软件连接技术。

(3) PLL：片内集成了 6MHz ~ 48MHz 时钟乘法 PLL，PLL 的工作不需要外部元件。

(4) 位时钟恢复：位时钟恢复电路使用 4X 过采样规则，从进入的 USB 数据流中恢复时钟，它能跟踪 USB 规定范围内的抖动和频漂。

(5) Philips 串行接口引擎 (PSIE)：它实现了全部的 USB 协议层，完全由硬件实现而不需要软件的参与。

(6) GoodLink™：GoodLink™技术可提供良好的 USB 连接指示。在枚举中 LED 指示根据通信的状况间歇闪烁。当 PDIUSBD12 成功地枚举和配置后，LED 指示将一直点亮，随后与 PDIUSBD12 之间成功的传输 (带应答) 将关闭 LED，处于挂起状态时，LED 将会关闭。该特性为 USB 器件、集线器和 USB 通信状态向用户提供了良好的指示。

(7) 存储器管理单元 (MMU) 和集成 RAM：在以 12Mb/s 的速率传输并与微控制器并

口相连时，MMU 和集成 RAM 作为 USB 之间速度差异的缓冲区，允许微控制器以它自己的速率对 USB 信息包进行读写。

⑻ 并行和 DMA 接口：一个普通的并行接口定义成易于使用、快速而且可以与主流的微控制器直接接口，对一个微控制器而言，PDIUSBD12 看起来就像一个带 8 位数据总线和一个地址位（占用 2 个位置）的存储器件。PDIUSBD12 支持多元和非多元的地址和数据总线。还支持主端点与本地共享 RAM 之间直接读取的 DMA 传输。支持单周期和突发模式的 DMA 传输。

（二）PDIUSBD12 管脚配置（见表 4.2）

表 4.2　PDIUSBD12 管脚的描述

管脚	符 号	类型	描　　述
1	DATA < 0 >	IO2	双向数据位 0
2	DATA < 1 >	IO2	双向数据位 1
3	DATA < 2 >	IO2	双向数据位 2
4	DATA < 3 >	IO2	双向数据位 3
5	GND	P	地
6	DATA < 4 >	IO2	双向数据位 4
7	DATA < 5 >	IO2	双向数据位 5
8	DATA < 6 >	IO2	双向数据位 6
9	DATA < 7 >	IO2	双向数据位 7
10	ALE	I	地址锁存使能。在多路地址/数据总线中，下降沿关闭地址信息锁存。将其固定为低电平用于单地址/数据总线配置
11	CS _ N	I	片选（低有效）
12	SUSPEND	I，OD4	器件处于挂起状态
13	CLKOUT	O2	可编程时钟输出
14	INT _ N	OD4	中断（低有效）
15	RD _ N	I	读选通（低有效）
16	WR _ N	I	写选通（低有效）
17	DMREQ	O4	DMA 请求
18	DMACK _ N	I	DMA 应答（低有效）
19	EOT _ N	I	DMA 传输结束（低有效）。EOT _ N 仅当 DMACK _ N 和 RD _ NWR _ N 一起激活时才有效
20	RESET _ N	I	复位（低有效且不同步）。片内上电复位电路，该管脚可固定接 VCC
21	GL _ N	OD8	GoodLink LED 指示器（低有效）
22	XTAL1	I	晶振连接端 1（6MHz）
23	XTAL2	O	晶振连接端 2（6MHz）。用外部时钟信号连接 XTAL1，XTAL2 应当悬空
24	V_{CC}	P	电源电压（4.0V ~ 5.5V）。要工作在 3.3V，对 V_{CC} 和 $V_{OUT3.3}$ 脚都提供 3.3V

25	D_-	A	USB D - 数据线
26	D_+	A	USB D + 数据线
27	$V_{OUT3.3}$	P	3.3V 调整输出。要使器件工作在 3.3V，对 V_{CC} 和 $V_{OUT3.3}$ 脚都提供 3.3V
28	A0	I	地址位。A0 = 1 选择命令指令，A0 = 0 选择数据。在多路地址/数据总线配置时可忽略，应将其接高电平
注：表中类型的含义为：O2——2mA 驱动输出；OD4——4mA 驱动开漏输出；OD8——8mA 驱动开漏输出；IO2——2mA 输入/输出			

（三）PDIUSBD12 与 μP 的连接

图 4.38 为 PDIUSBD12 与 80C51 的连接方式，当硬件连接好后，剩下的就是程序的编制了，而要编制程序必须按照 PDIUSB12 的命令进行。PDIUSB12 的命令字分为 3 种：初始化命令、数据流命令和通用命令字。有关命令的使用请参考 PDIUSB12 的数据手册。

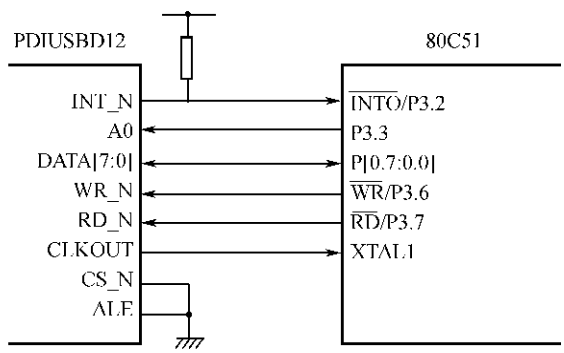


图 4.38　PDIUSBD12 与 80C51 的连接

（四）μP 的编程的思路

要对 μP 编程必须彻底消化 PDIUSBD12 的命令及电气特性，这是需要一定时间的，好在厂商提供了典型电路相对应的较完整的程序模块，一般用户在消化 PDIUSBD12 的命令及电气特性的基础上，按照自己的电路修改厂商提供的源程序即可开始调试。下面就 μP 的编程的思路做一简单介绍。

我们的目标就是使 PDIUSBD12 在 USB 上达到最大的传输速率。外围设备例如打印机、扫描仪、外部的海量存储器和数码相机都可使用 PDIUSBD12 在 USB 上传输数据。这些设备的 CPU 要忙于处理许多设备控制和数据以及图像处理等任务。PDIUSBD12 的固件设计成完全的中断驱动。当 CPU 处理前台任务时，USB 的传输可在后台进行。这就确保了最佳的传输速率和更好的软件结构，同时简化了编程和调试。

后台 ISR 中断服务程序和前台主程序循环之间的数据交换，通过事件标志和数据缓冲区来实现，原理如图 4.39 所示。例如，PDIUSBD12 的批量输出端点可使用循环的数据缓冲区。当 PDIUSBD12 从 USB 收到一个数据包，那么就对 CPU 产生一个中断请求，CPU 立即响应中断。在 ISR 中，固件将数据包从 PDIUSBD12 内部缓冲区移到循环数据缓冲区，并在随后清零 PDIUSBD12 的内部缓冲区，以使能接收新的数据包。CPU 可以继续它

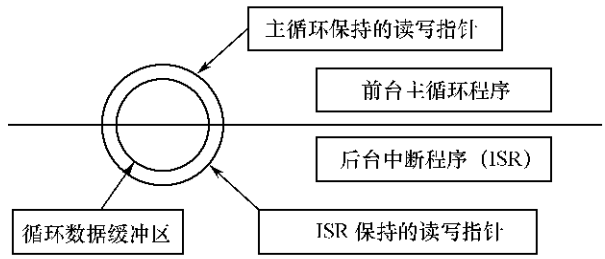


图 4.39　主程序和中断程序的分工

的前台任务。CPU 在主循环中检查循环缓冲区内是否有新的数据，并开始其它的前台任务。对于这种结构，主循环只检查循环缓冲区内需要处理的新数据。而中断服务程序能够以最大可能的速度进行数据的传输。

相似地，控制端点在数据包处理时采用了同样的概念。ISR 接收和保存数据缓冲区中的控制传输，并设置相应的标志寄存器。主循环向协议处理程序发出请求。由于所有的标准器件、级别和厂商请求都是在协议处理程序中进行处理，ISR 得以保持它的效率，而且一旦增加新的请求只需要在协议层进行修改。

（五）主要程序模块介绍

1. 各模块的作用

整个 USB 程序具有积木式的结构，如图 4.40 所示，图中箭头方向即为数据传送方向。各模块的功能如下。

⑴ 硬件提取层（EPPHAL. C）：对单片机的 I/O 口、数据总线等硬件接口进行操作。

⑵ PDIUSBD12 命令接口（D12CI. C）：对 PDIUSBD12 器件进行操作的模块子程序集。

⑶ 中断服务程序（ISR. C）：当 PDIUSBD12 向单片机发出中断请求时，读取 PDIUSBD12 的中断传输来的数据，并设定事件标志“EPPFLAGS”和 Setup 包数据缓冲区“CONROL _ XFER”传输给主循环程序。

⑷ 标准请求处理程序（CHAP _ 9. C）：对 USB 的标准设备进行处理。

⑸ 厂商请求处理程序（PROTODMA. C）：对用户添加的厂商请求进行处理。

⑹ 主循环程序（MAINLOOP. C）：发送 USB 请求、处理 USB 总线事件和用户功能处理。

各模块的程序详见附录 1，这里仅描述一下 MAINLOOP. C 和 ISR. C 的流程图。

前面已经讲过，后台 ISR 中断服务程序和前台主程序循环之间的数据交换是通过事件标志和数据缓冲区来实现的。下面的结构体就是事件标志“EPPFLAGS”和 Setup 包数据缓冲区“CONROL _ XFER”。

```
USB 事件标志
typedef union _ epp _ flags
{  struct _ flags
   {  unsigned char timer                    :1 ;//时间溢出
      unsigned char bus _ reset              :1 ;//总线复位标志
```

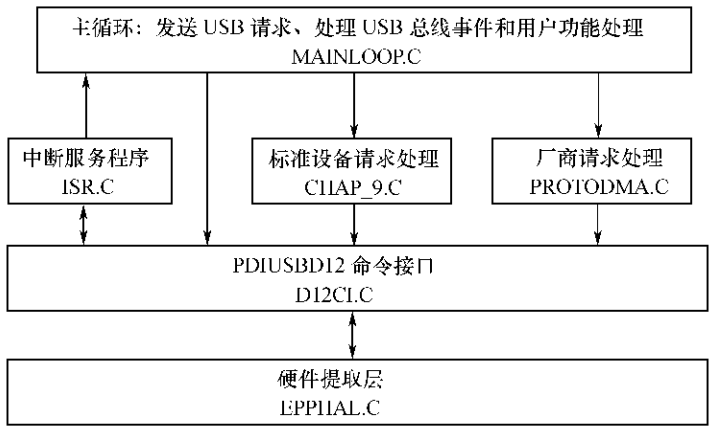


图 4.40　各程序模块和数据流向

```
        unsigned char suspend              :1 ;//挂起改变标志
        unsigned char setup _ packet       :1 ;//收到 Setup 包
        unsigned char remote _ wakeup      :1 ;//远程唤醒标志
        unsigned char in _ isr             :1 ;//USB 中断服务标志
        unsigned char control _ state      :2 ;//控制端点处理状态 0 :IDEL 空闲状态
                                           //1 :TRANSMIT  数据发送状态
                                           //2 :RECEIVE  数据接收状态
        unsigned char configuration        :1 ;//配置标志 (0 :未配置 ;1 :已配置)
        unsigned char commond              :1 ;//未使用
        unsigned char ep1 _ rxdone         :1 ;//端点 1 收到数据标志
        unsigned char ep2 _ rxdone         :1 ;//端点 2 收到数据标志
        unsigned char ep1buf _ full        :1 ;//端点 1 输出双缓冲区满标志
        unsigned char ep2buf _ full        :1 ;//端点 2 输出双缓冲区满标志
    } bits;
unsigned short value;
} EPPFLAGS;
USB 设备请求寄存器
typedef struct _ device _ request
{   unsigned char bmRequestType;   //请求类型 (数据的传输方向、类型、接收器)
    unsigned char bRequest;        //USB 请求
    unsigned short wValue;         //USB 请求值
    unsigned short wIndex;         // USB 请求索引
    unsigned short wLength;        //计算长度
} DEVICE _ REQUEST;
```

```
Setup 包数据缓冲区
typedef struct _ control _ xfer
{    DEVICE _ REQUEST DeviceRequest；//USB 设备请求结构体,8 字节
     unsigned short wLength；          //传输数据的总字节数
     unsigned short wCount；           //传输字节数统计
     unsigned char * pData；           //传输数据的指针
     unsigned char dataBuffer[MAX _ CONTROLDATA _ SIZE]；//请求的数据
} CONTROL _ XFER；
```

上面的结构体定义在 MYLOOP. H 的库函数里。在程序中定义了两个全局变量的结构体来进行前后台的数据交流,定义如下：

```
EPPFLAGS          bEPPflags；
CONTROL _ XFER    ControlData；
```

其中,bEPPflags 用来标记前后台之间的工作状态,ControlData 用来保存 Setup 包请求的类型和请求的数据。

2. 主循环 (MAINLOOP. C)

主循环在整个固件结构中处于最上层,它的作用是非常重要的。设备上电后,主机通过设备的上拉电阻产生信号变化来检测新的设备连接。PDIUSBD12 片内有 1.5kΩ 的 Soft Connect 上拉电阻,默认状态下不与 V_{CC} 相连,允许系统微控制器来决定与 USB 建立连接的时间。MCU 一旦上电就需要初始化其所有端口、存储区、定时器和中断服务程序。之后 MCU 将重新连接 USB。这些过程是很重要的,因为它确保了在 MCU 准备好服务 D12 之前 D12 不会进行操作。

主循环会检测 USB 事件标志。例如当主循环程序检测到 bEPPflags. bits. setup _ packet = 1 时,主循环就调用协议控制程序 (control _ handler ()) 来区分是 USB 标准请求还是厂商请求,并根据请求类型 (ControlData. DeviceRequest. bRequest) 来跳转到相应的子程序进行进一步的处理,譬如在完成枚举过程的时候。

另外,主循环还包括人机接口的代码,在这里,用户可以添加代码,实现自己想要实现的功能。对本程序来说,主循环的用户代码部分要实现两个功能：第一,USB 设备能够接收到 PC 机的数据 (包括用户命令)；第二,要根据用户发给的命令来实现将设备采集到的数据按要求传送到 PC 机显示给用户。在主程序中有一个“While (True)”的循环。其中最后一个判断为“中断接收中用户部分”,主程序在这里查询 bEPPflags. bits. ep2 _ rxdone 是否为 1 (在中断服务程序中已经说明,bEPPflags. bits. ep1 _ rxdone 为 1 说明端点缓冲区已经放了程序采集来的数据),如果 bEPPflags. bits. ep2 _ rxdone 为 1,就把 bEPPflags. bits. ep2 _ rxdone 清 0,然后读取 EpBuf 中的信息 (本程序为 6 个字节的命令) 并判断用户需要什么操作,再根据判断的结果去执行这些操作。本程序所要求的操作也就是前面所说“户代码部分要实现的两个功能”,程序会把输出的数据放到指定的地址,把要输入的数据放到端点缓冲区等待主机的读取。在主程序中还有一个“MAINLOOP 中用户部分”。它主要对采集到的数据进行处理。主循环的流程图见图 4.41。

3. 中断服务子程序 (ISR. C)

中断服务程序代码就是处理由 PDIUSBD12 产生的中断,它主要负责将数据从 PDI-

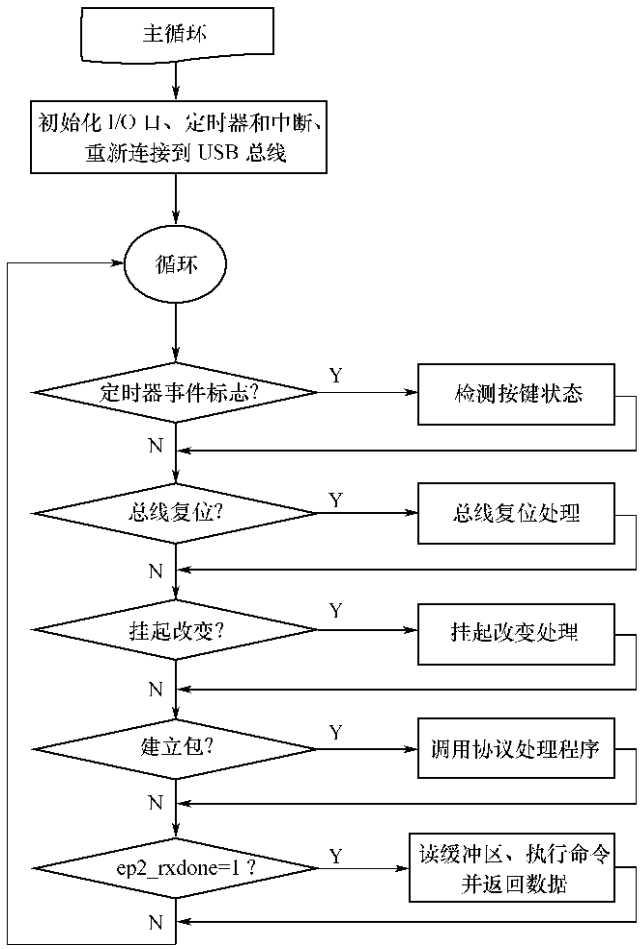


图 4.41　主循环程序流程

USBD12 的内部 FIFO 取回到 CPU 存储器，并建立正确的事件标志，以通知主循环程序进行处理。在 ISR 的入口，固件使用 D12 _ ReadInterruptRegiste () 来决定中断源，然后将进入相应的子程序进行处理，其流程图如图 4.42 所示。ISR 与前台主循环通过事件标志“EPPFLAGS”HE 和数据缓冲区“CONTROL _ XFER”进行通信。

主循环与 ISR 之间的任务分配是这样的：ISR 从 D12 接收数据，而主循环对数据进行处理。当 ISR 收集了足够数据时，它只通知主循环已经准备好等待处理。例如，在 OUT 数据阶段建立包时，ISR 将建立包和 OUT 数据包都存入“CONTROL _ XFER”缓冲区中，然后将“set _ up _ packet”标志位送到主循环，这将减少主循环不必要的服务时间，并且简化了主循环的编程。

4.2.5　CAN 总线简介

一、CAN 总线概述

CAN 全称为 Controller Area Network，即控制器局域网，是国际上应用最广泛的现场

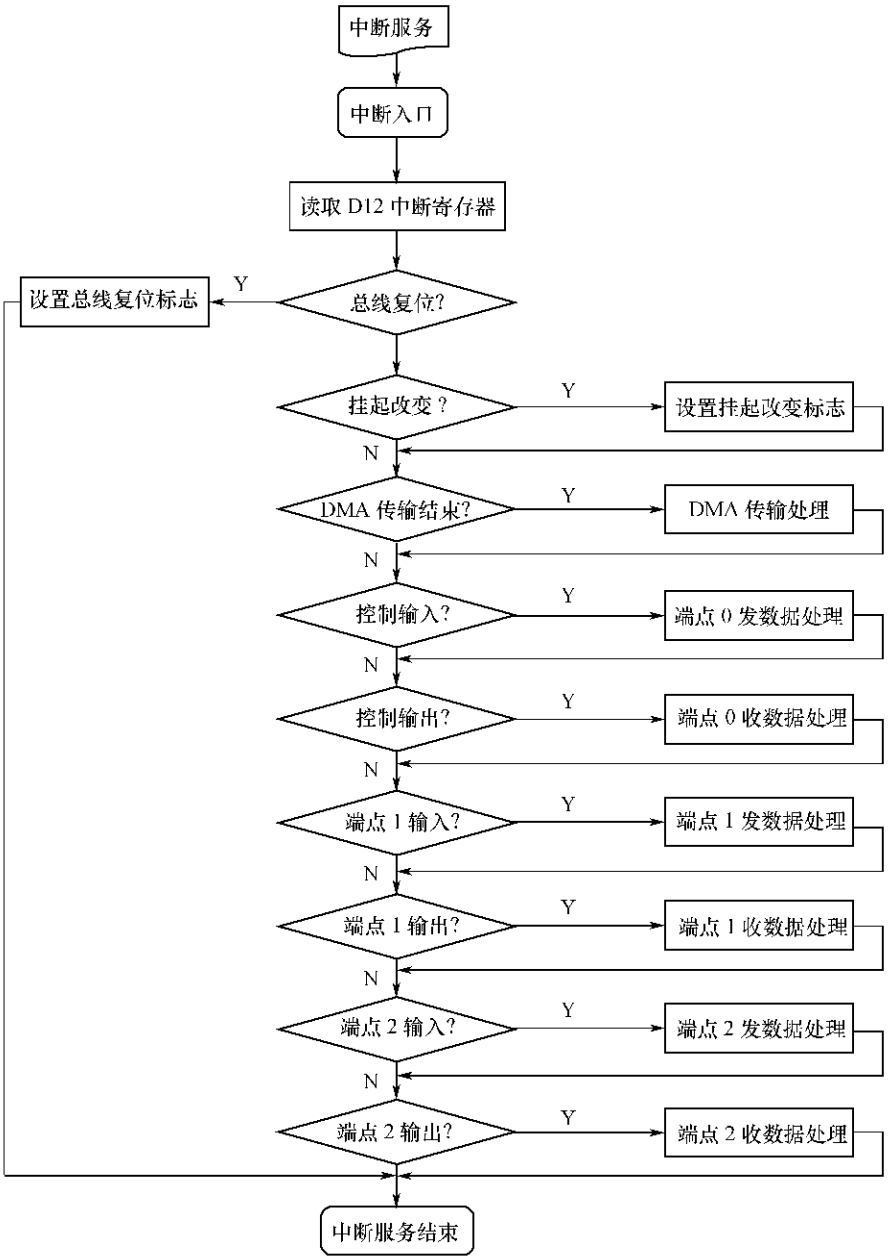


图 4.42　中断服务程序流程

总线之一。CAN 最初出现在 20 世纪 80 年代末的汽车工业中，由德国 Bosch 公司最先提出。当时由于消费者对于汽车功能的要求越来越多，而这些功能的实现大多是基于电子操作的，这就使得电子装置之间的通信越来越复杂，同时意味着需要更多的连接信号线。提出 CAN 总线的最初动机就是为了解决现代汽车中庞大的电子控制装置之间的通信，减少不断增加的信号线。于是他们设计了一个单一的网络总线，所有的外围器件可以被挂接在该总线上。1993 年 CAN 已成为国际标准 ISO 11898（高速应用）和 ISO 11519（低速

应用）。目前流行的 CAN 协议为 CAN2.0B。CAN 具有十分优越的特点，使人们乐于选择。这些特性包括：

⑴ 低成本；

⑵ 极高的总线利用率；

⑶ 很远的数据传输距离（长达 10km）；

⑷ 高速的数据传输速率（高达 1Mb/s）；

⑸ 可根据报文的 ID 决定接收或屏蔽该报文；

⑹ 可靠的错误处理和检错机制；

⑺ 发送的信息遭到破坏后，可自动重发；

⑻ 节点在错误严重的情况下具有自动退出总线的功能；

⑼ 报文不包含源地址或目标地址，仅用标志符来指示功能信息、优先级信息。

二、CAN 基本结构

一个完整的具有 CAN 总线功能的系统如图 4.43 所示，它由 CAN 收发器、CAN 控制器、含 CPU 的智能芯片组成。同 485 总线类似，CAN 总线收发器也是采用平衡发送差动接收的方式，但 485 总线仅仅是总线驱动和接收部分，其数据收发部分是靠 UART 完成的（见图 4.44），而 CAN 总线不但包括收发器，而且自身含有比 UART 更合理的收发协议控制器。

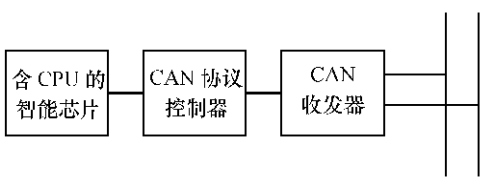


图 4.43　CAN 总线基本结构

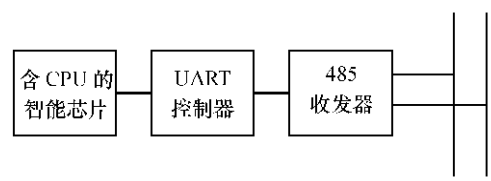


图 4.44　485 通信的基本结构

CAN 协议的一个最大特点是废除了传统的站地址编码，而代之以对通信数据块进行编码。采用这种方法的优点可使网络内的节点个数在理论上不受限制，这种按数据块编码的方式，还可使不同的节点同时接收到相同的数据，这一点在分布式控制系统中非常有用。数据段长度最多为 8 个字节，可满足通常工业领域中控制命令、工作状态及测试数据的一般要求。同时，8 个字节不会占用总线时间过长，从而保证了通信的实时性。CAN 协议采用循环冗余码校验 CRC（Cyclic Redundancy Check）并可提供相应的错误处理功能，保证了数据通信的可靠性。CAN 卓越的特性、极高的可靠性和独特的设计，特别适合工业过程监控设备的互连，因此，越来越受到工业界的重视，并已公认为最有前途的现场总线。

CAN 能够使用多种物理介质，例如双绞线、光纤等。最常用的就是双绞线。信号使用差分电压传送，两条信号线被称为“CAN_H”和“CAN_L”，静态时均是 2.5V 左右，此时状态表示为逻辑“1”，也可以叫做“隐性”。用 CAN_H 比 CAN_L 高，表示逻辑“0”，称为“显形”，此时，通常电压值为：CAN_H = 3.5V 和 CAN_L = 1.5V。

三、基于 μP 的 CAN 接口的设计

同 USB 接口一样，要了解 CAN 协议及其硬件全部特性也需要一定的时间，但作为应

用设计者可以凭借厂方提供的技术支持很快设计出 CAN 的接口。更可喜的是，已经有一些厂家推出了带有 CAN 控制器的单片机，例如 Silabs 公司的 C8051F40/1/2/3、Philips 公司的 P8XC591，都带有 CAN2.0B 协议的控制器，对于此类单片机，只要为单片机配上 CAN 收发器就可以按照说明书直接编程操纵 CAN 接口了，其硬件和软件的设计相对要简单一些。对于不带 CAN 协议控制器单片机来说，要为它设计 CAN 接口，除了配置 CAN 收发器外，还必须配置 CAN 协议控制器。常见的 CAN 收发器有 PCA82C250/1、PCA82C252、TJA1050、TJA1053/4 等 常见的 CAN 协议控制器有 82C200、SJA1000 等。

近几年来单片机发展非常迅速，随着 CAN 总线应用的进一步普及，自带 CAN 控制器的单片机将会越来越多，因此利用独立的 CAN 控制器设计 CAN 接口将会变成历史。下面我们以一款典型的带 CAN2.0B 控制器的单片机 C8051F040/1/2/3 为例，说明如何实现 μP 的 CAN 接口。

F040/1/2/3 具有和 F020/21 (请参阅第 1 章) 相当的功能，但在性能上做了较大的改进。比如说，F020/1 的模拟电压输入不能超过 5V，而 F040/1/2/3 具有 60V 通用模式输入范围的高压差分放大，偏置调节范围为 - 60V ~ + 60V ；F020/1 是两个模拟比较器，F040/1/2/3 变为 3 个模拟比较器 ；最大的改动是在 I^2C、UART、SPI、JTEG 接口不减少的情况下增加了 CAN2.0B 接口。

C8051F040/1/2/3 内部 CAN 功能部分和系统 MCU 的结构关系如图 4.45 所示，系统由兼容 51 的 MCU、CAN 控制器核、消息 RAM、消息处理器寄存器和预分频器等组成，其中 CAN 控制器核是完全按照 CAN2.0B 设计的核心控制器 ；消息 RAM 可以存储 32 个事件消息 ；消息处理器用来对消息优先排队管理产生中断申请 ；寄存器是为所有 CAN 工作的初始化及操作而设置的，每个寄存器在 MCU 中都有相应的地址 ；预分频器用来设置 CAN 通信的位速率。

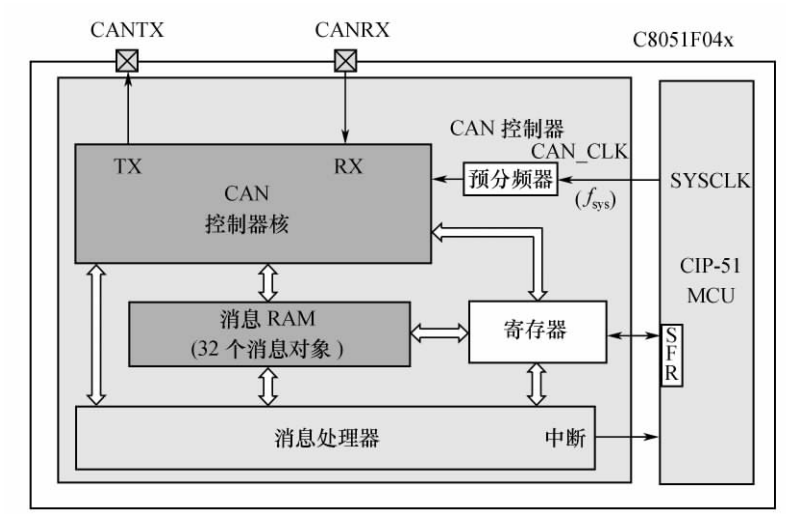


图 4.45　C8051F041 的内部 CAN 控制部分

图 4.46 为用 F041 组成的 CAN 接口电路，其中，TJA1050 为 CAN 收发器，113 为高速光电隔离器，J1 和 J2 为两种不同形式的引线端子。对 F040 来说，其 CAN 通信信号的接

收和发送是分别通过 CANRX 和 CANTX 来完成的。CAN 通信是半双工的，在接收数据时 CAN 总线上的数据以 CANH 和 CANL 的显性或隐性电平经 TJA1050 收发器变成 TTL 形式的电平从 RXD 端输出，再经过 113 光隔后输入 F041 的 CANRX 端 在发送数据时 F041 由 CANTX 产生信号经光隔后进入 TJA1050 收发器的输入端 TXD，由 TJA1050 变成 CANH 和 CANL 的显性或隐性电平。这里之所以要设立光隔，是为了提高系统的可靠性，因为 CAN 一般是用在工业现场，总线上的各种干扰较多，有了光隔后使总线和系统相互隔离总线上的强干扰不至于损坏系统。这样一来，图中的 5Vcan 必须单独供电，好在 TJA1050 只消耗很小的电流，所有，一般情况下只要用一个极小功率的DC - DC变换器就可解决问题。

虽然实现 F040 /1 /2 /3 的 CAN 接口电路较简单，但要彻底消化其 CAN 编程还是要一定时间的，好在厂家可给出典型的例程供我们移植，这样我们在较短的时间即可实现所要求的功能。与图 4. 46 相对应的一个典型程序请参阅附录 2。

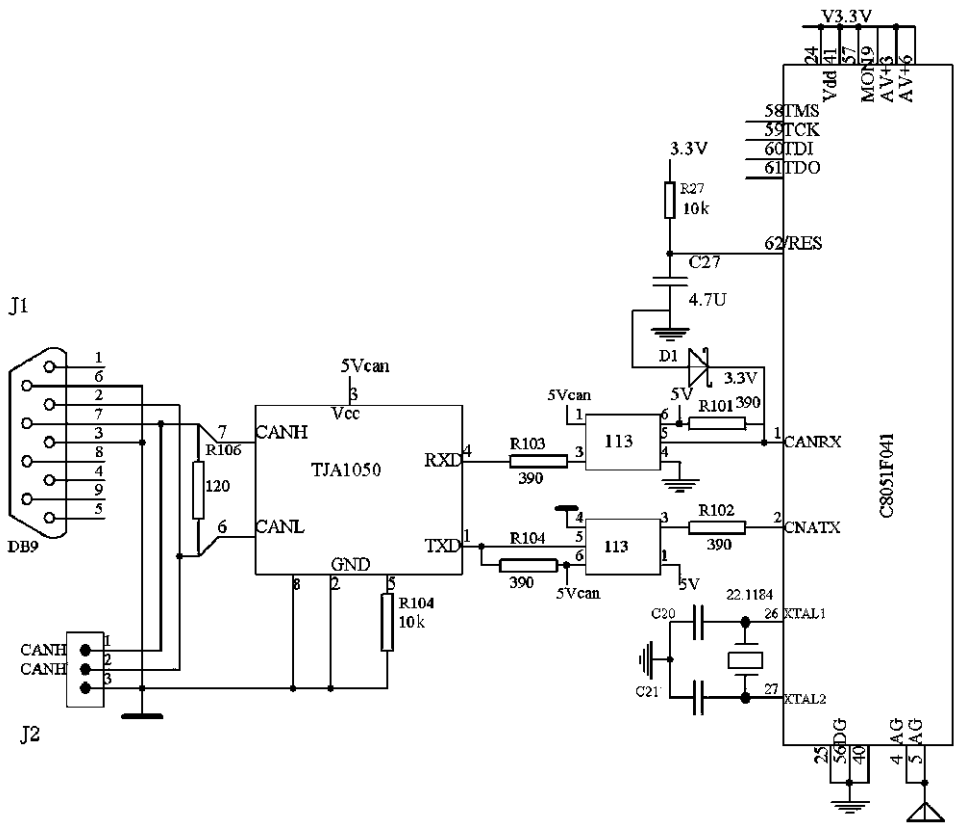


图 4. 46　由 C8051f041 组成的 CAN 接口电路

第 5 章　常见的智能处理功能

5.1　硬件故障自检

5.1.1　硬件自检分类

在智能检测系统中，我们可以用智能单元强大的功能对系统自身进行全面的检测，比如说键盘是否正常、LED 有无故障、外围设备是否联机等，这个过程称为自检。自检可以用显示字符、发出提示音等方式告诉操作者，自检可以分为开机自检、周期性自检、随机自检 3 种方式。

⑴ 开机自检

开机自检是指在打开电源 (POWER. ON) 或复位 (RESET) 后系统自动进行一次自检，自检完成后报告自检结果，发现问题及时处理。

⑵ 周期性自检

周期性自检简称周检，是指系统按一定周期自动进行一次自检，自检周期可按时间来设定，如一周自检一次或一月自检一次；自检周期也可按使用次数来设定，如每使用 100 次自检一次或每使用 2000 次自检一次。无论是按哪种方式都应使自检时不影响正常使用，比如将周检放在系统开机时进行，而不能放在系统使用中进行。在两次周检间隔内系统可能存在问题，只有等周检后方可发现。

⑶ 随机自检

随机自检是指根据实际使用情况随时进行的自检，比如在系统使用中发现不太正常，可进行一次全面自检，或在系统空闲时进行一次自检等。随机自检可以人工通过键盘或远程通信给出执行指令，也可由系统通过程序判断而自动给出执行指令。

5.1.2　自检算法

一、ROM 类存储器的检测

ROM 类存储器包括：ROM、PROM、EPROM、E^2PROM 等，ROM 中存放的是程序或常数，不可有差错，否则系统无法运行。

如表 5.1 所列，假定要检验从地址 A_1 到 A_n 的一段 ROM 中的值，我们可采用奇偶校验法或累加和校验法。

(一) 奇偶校验法

在地址 $A_1 \sim A_n$ 外另存一个单元为校验字，其取值使某一列有奇数个 1 称为奇校验，其取值使某一列为偶数个 1 称为偶校验。

以奇校验为例，在设计自检程序时，将 $A_1 \sim A_{n+1}$ 个单元按列进行异或运算，结果为 1 校验通过，反之出错。奇偶校验法可以发现同一位上奇数个错误。

（二）累加和校验法

如表 5.2 所列，同样假定被校验 ROM 单元为 $A_1 \sim A_n$，首先将 n 个单元中的内容相加结果存在 A_{n+1}。

校验时，先计算一遍 $A_1 \sim A_n$ 的累加和并与 A_{n+1} 的值相比较，相同则校验通过，否则出错。累加和校验法不能确诊是哪一列出问题。

表 5.1　奇偶校验法

地址	内　容
A_1	11011101
A_2	10111011
…	…
A_n	11010011
A_{n+1}	01101000

表 5.2　累加和校验法

地址	内　容
A_1	11001011
…	…
A_n	10110000
A_{n+1}	11010101

二、RAM 的检测（见图 5.1）

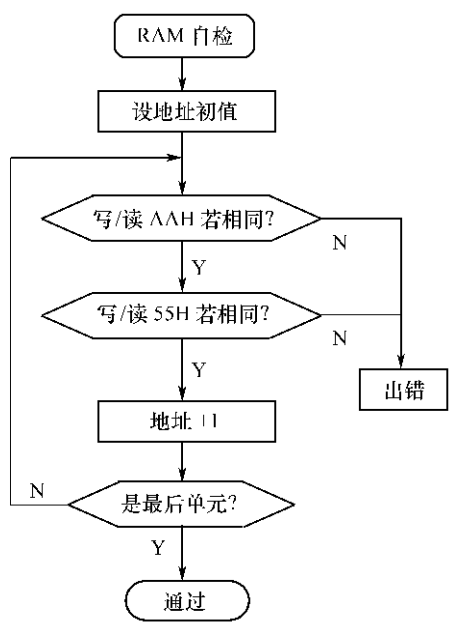


图 5.1　RAM 自检的算法

RAM 可写可读，如写入和读出相等则无误，为了防止位与位之间在短路上连线的情况，不可写/读 OOH、FFH 来判断，一般用写/读 AAH 和 55H 来判断，即写入 AAH 读出看是否为 AAH，再写入 55H 读出看是否为 55H。由于写读 AAH 和 55H 不能判定非相邻数据线的短路，在要求更高的场合可采用其它方式校验，如循环 1 移位法或循环 0 移位法等。

三、显示类器件的自检

（一）数码管显示器的自检

数码管显示器的自检可用多种方式，常见的方法为将数码管的显示段（如 8 段数码

管的 a,b,c,…,h) 顺序点亮再顺序熄灭,还可以让数码管轮流显示规定的字符 (如 1,2,3,4,5,6,7,8,9,0),其基本原则是判定显示及驱动电路有无短路或断路等故障。

(二) 点阵式显示器的自检

点阵式显示器可参考数码管的自检方式,但由于点阵式显示器的发光点太多,如采用全段通电顺序点亮再熄灭的方式将耗费较多的时间,为此可将显示器分成数个区域,每个区域同时进行顺序点亮再顺序熄灭的方法来检验。

(三) CRT 显示器的自检

CRT 显示器是电子束扫描,不存在短路及断路的问题,只要能全亮或全暗就表示扫描没问题,但还需要进一步检测其光栅大小、聚焦、亮度、色彩等指标,可以编一些典型的画面显示来全面验证这些指标。

总之,显示类器件的自检需借助操作者的观察来判断。

四、键盘的自检

键盘的自检也需配合操作者的动作,每按下一个键,CPU 获得这个键值时可用声光等形式反馈一个信号。如一个具有 LED 数码管显示的系统在检验其"0 ~ 9"10 个按键时,可以采用按"几"显示"几"的方法。

五、数据采集通道自检

对于模拟量数据采集通道,随电路的不同可采用不同的形式达到自检的目的。如图 5.2 所示的 4 通道数据采集电路中,P1.1、P1.0 用来选择通道号,为了验证每一通道的功能及其准确性,增加 M2、M3 两个模拟开关和 V_{ref} 参数电压,在自检时输入信号全部断开,令 P1.2 = 1,选择 M3 输入为 V_{ref},在 P1.1、P1.0 取不同的值时,通过 M2 数据采集的每一通道的输入都是 V_{ref},此时如 A/D 结果都在 V_{ref} 输入对应值范围内时则系统无误。

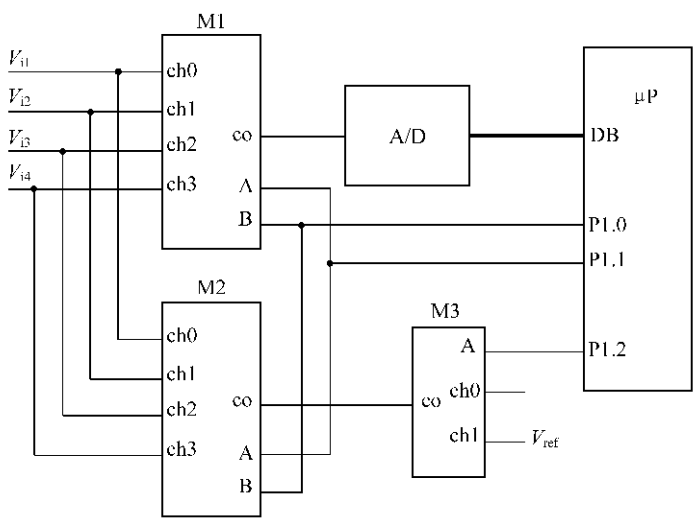


图 5.2 一种数据采集通道的自检方法

六、模拟量输出通道自检

模拟量输出通道自检的目的,是检验 D/A 输出是否正确,此时可增加一个附加的 DAS 取样检验,如果系统本身具有 A/D 和 D/A,可以占用 DAS 的其中一路来达到检验的

目的。图 5.3 为一种模拟量输出通道自检的电路。

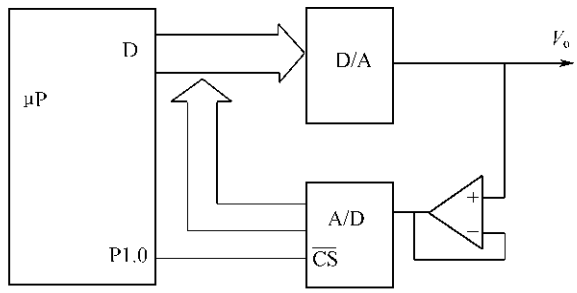


图 5.3　一种模拟量输出通道自检的电路

七、开关量输出通道自检

有些开关量的输出是很重要的，应通过自检来检验其是否真正执行了命令。比如说地铁的门在关闭后应确认一下其是否真正关闭，以避免发生事故。图 5.4 为一种自检方法，由 P1.0 通过光隔 G1 和继电器 J1 决定门控液压装置是否接通 220V 电源。当车门真正关闭时，车门行程开关 K1 将闭合，通过光隔 G2 使 P1.1 信号发生改变，由此判断车门是否真正闭合。

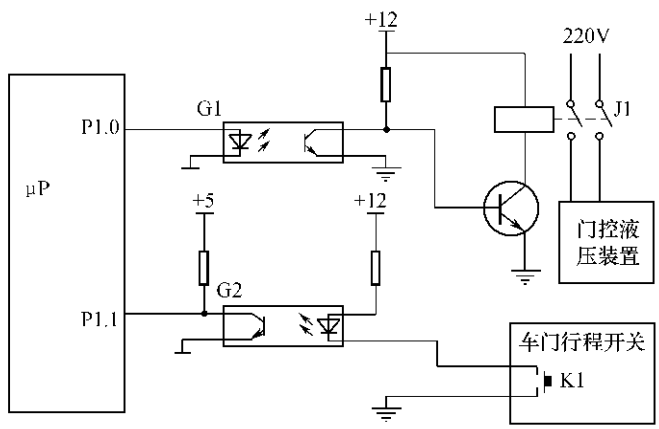


图 5.4　一种开关量输出通道的自检方法

5.2　自动测量功能

5.2.1　自动量程转换

一、自动量程转换的定义

为了使测量系统具有较高的分辨力和准确性，我们总希望测到的值尽可能达到 A/D 的满量程。如某 12 位 A/D 的满量程最大输入为 5V，如果信号的最大输入值是 5V，将可得到最大为 4095 的转换值。有些信号较小，可能达不到 5V，需变大增益；而有些过大会

使 A/D 饱和，又需降低增益。自动改变系统的增益，使输入信号最大值不同的信号都可达到满量程的测量精度，称为自动量程转换。

二、自动量程转换的方法举例

自动量程转换首先需要一个程控增益放大器 PGA，有了 PGA 后，自动量程转换实质上就是求 PGA 控制值的过程。

（一）输入信号为直流时 PGA 控制值的算法

设输入信号为直流电压，A/D 满量程为 ±5V，PGA 的增益变化范围为 1 ~ 1000，数据采集如图 5.5 所示。用试探法通过采样值确定 PGA 的控制值，软件流程如图 5.6 所示，其基本思路为先试探求一个 A/D，看 A/D 值是否在 3072 ~ 4090 之间，如果不是，改变增益一直到达到目的。

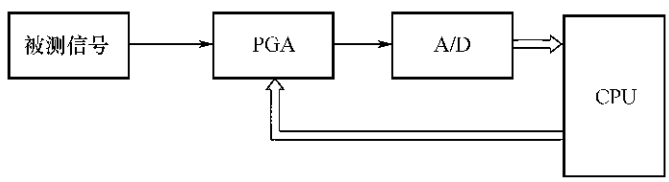


图 5.5　通过判断采样值确定 PGA 增益的方案

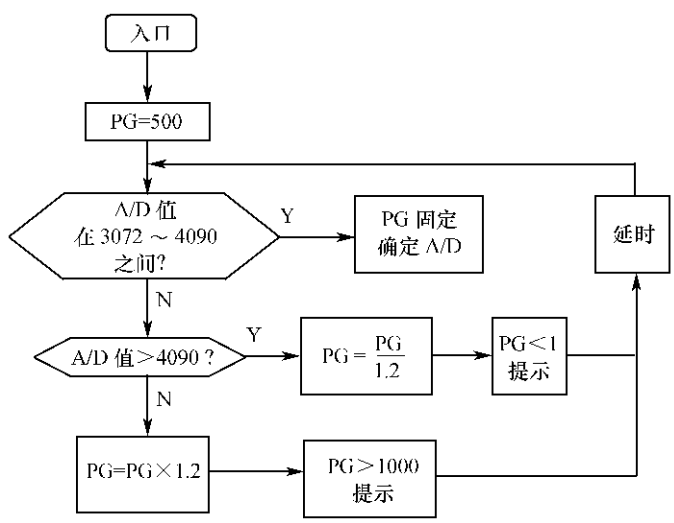


图 5.6　直流输入时的一种 PGA 算法

（二）输入信号为交流时 PGA 控制值的算法

如输入为交流信号，先采大于一个输入信号周期的一组 A/D 值，求出采样的最大值。使最大值尽可能符合 A/D 的满量程而输出又不饱和。算法和图 5.6 基本相同。

（三）配合比较器的 PGA 算法

一般情况下，被测信号都可以视为交流信号。为了提高自动量程转换的速度并得到可靠的 PGA 控制值，可采样如图 5.7 所示的方案。首先将被测信号经 PGA 放大后通过全波整流及峰值保样电路变为直流信号，该直流信号再通过上下限比较器变成开关量，由

P1.1 和 P1.0 判断信号的幅值是否在规定的区域内。假定 A/D 的满量程输入值为 5V，我们可规定信号在 3.8V～4.8V 区域时 PGA 的增益较合适，用 3.8V 和 4.8V 作为比较器的双门限。

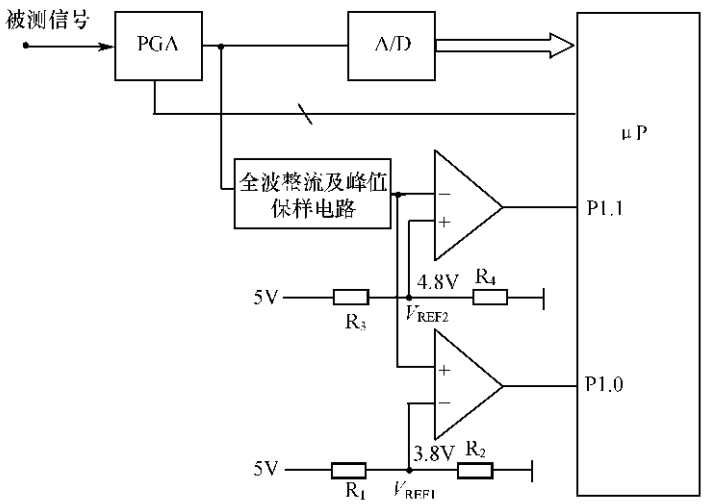


图 5.7　配合比较器的 PGA 控制值求法电路

三、PGA 的设计举例

程控放大器是指可以通过程序或指令控制而改变其增益等性能的放大器，英文缩写为 PGA (Programming Gain Amplifier)。PGA 在现代测控系统中是经常会用到的，随着各种新型元器件的不断发展，PGA 的实现并非难事，但由于 PGA 必须具有可以实现自动调节增益的功能，因而其精度总会因使用一些元器件受到影响。在精密测量场合对 PGA 的精度要求比较高，实现较精密的 PGA 是众望所归。

PGA 一般分为增益分挡变化和增益连续变化的两种类型，下面分别讨论。

(一) 增益分挡变化 PGA 设计

常言道，万变不离其宗，程控放大器千变万化但脱离不了 3 种基本类型，即反相放大器、同相放大器及差分放大器。由于同相放大器会引入共模干扰，而差动放大器实现程控较麻烦，人们往往用反相放大器的原型来设计程控放大器。

图 5.8 为一种形式的 PGA，通过模拟开关改变输入电阻进而达到改变增益的目的。

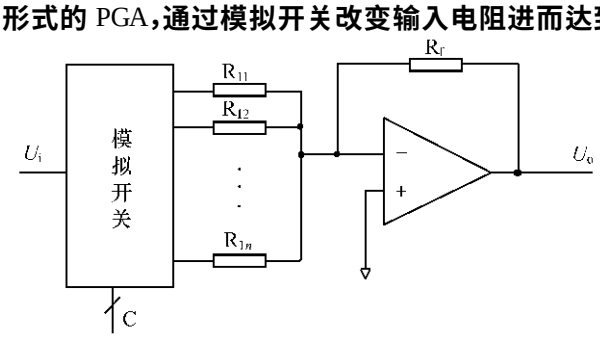


图 5.8　改变输入电阻的分挡 PGA

设增益为 G,则

$$G = -\frac{R_f}{R_{1i} + R_{on}}$$

式中,R_{1i}为被选中的输入电阻,R_{on}为模拟开关的导通电阻。可见模拟开关对放大器的增益是有影响的,R_{1i}越小这个影响越大。

图 5.9 为另一种形式的 PGA,通过模拟开关改变反馈电阻进而达到改变增益的目的。

$$G = -\frac{R_{fi} + R_{on}}{R_1}$$

式中,R_{fi}为被选中的反馈电阻,R_{on}为模拟开关的导通电阻。同样,模拟开关对放大器的增益是有影响的,R_{fi}越小这个影响越大。

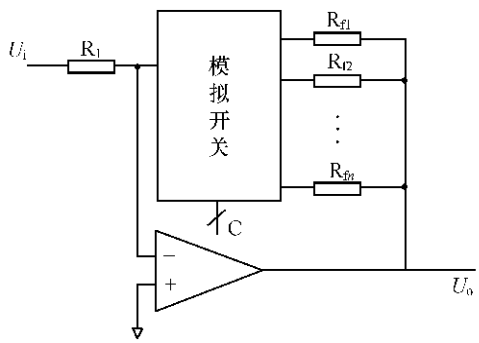


图 5.9　改变反馈电阻的分挡 PGA

除增益受 R_{on}影响之外,图 5.8 电路还有一个缺点,那就是输入信号的幅值不能超过模拟开关的电源电压范围,否则会损坏模拟开关。图 5.9 的电路由于运放的输入端电位为 0,输出端可以控制在一定范围内,因而输入信号的幅值可以高一些,因此图 5.9 电路比图 5.8 电路更好一些。但二者因模拟开关的导通电阻造成的增益误差是不可避免的,如图 5.9 中取 $R_{fi} = 100k\Omega$,$R_1 = 1k\Omega$,理想的增益应为 100/1 = 100,当选择 4000 系列模拟开关时,R_{on}约为 0.5kΩ,实际增益为 100 + 0.5/1 = 100.5。更令人烦恼的是,这个误差并不是一成不变的,不能按系统误差去消除,这是因为模拟开关的导通电阻随温度在变化。表 5.3 是图 5.9 电路的增益温漂测定结果。

表 5.3　图 5.9 电路的增益温漂测定结果

温度	实测增益	增益误差/%	温度	实测增益	增益误差/%
40	100.43	0.43	0	100.21	0.21
20	100.32	0.32	- 20	100.15	0.15

为了克服图 5.8 和图 5.9 的这些缺点,可采用图 5.10 和 5.11 所示的电路,这种电路对于 n 级的分挡增益需要 n 个基本放大器,经过 n 路模拟开关和一个电压跟随器输出,由于电压跟随器的输入阻抗极高,模拟开关的导通电阻对增益的影响可以忽略。因而各级增益完全取决于所选电阻,例如当选择 0.01% ~ 25×10^{-6}的精密电阻时,增益的精度也可达到同样等级的精度。这种电路用了较多的运放看起来好像有点奢侈,但它对运放的

要求较低，可用一般的运放来实现，因此电路的造价是很低的。

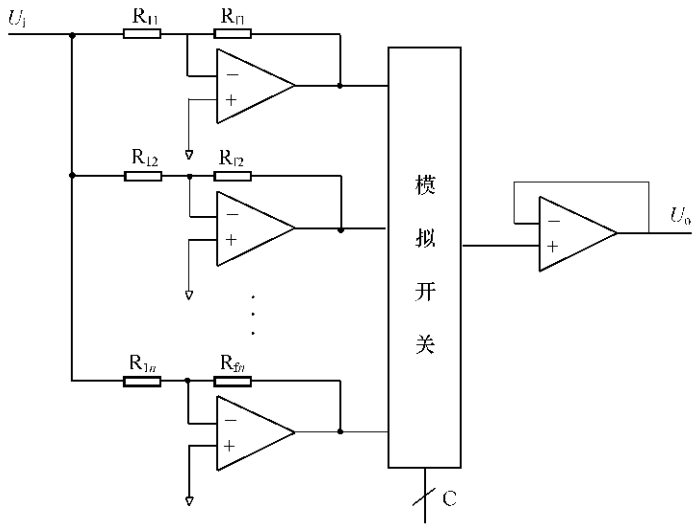


图 5.10　消除模拟开关导通影响电阻的反相分挡 PGA

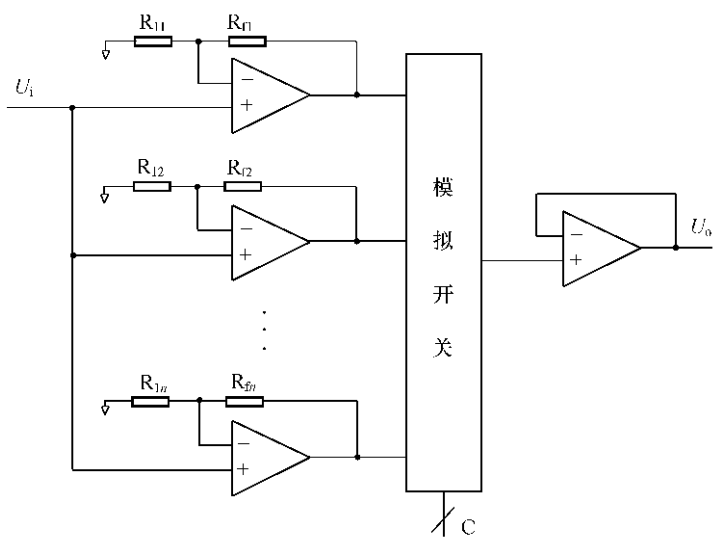


图 5.11　消除模拟开关导通电阻影响的同相分挡 PGA

（二）增益可连续变化的 PGA 设计

1. 用数字电位计实现 PGA

数字电位计是一种单片集成电路，其基本结构原理如图 5.12，它由多个相同的电阻、模拟开关、译码电路、非易失性数据寄存器、接口电路等组成。n 个模拟开关每次接通 1 个，由译码器决定。译码器的输入由非易失性寄存器的内容决定，非易失性寄存器的内容由接口电路通过控制信号修改，修改后掉电不丢失可保存 100 年以上，每个寄存器可承受 100000 次数据擦写。

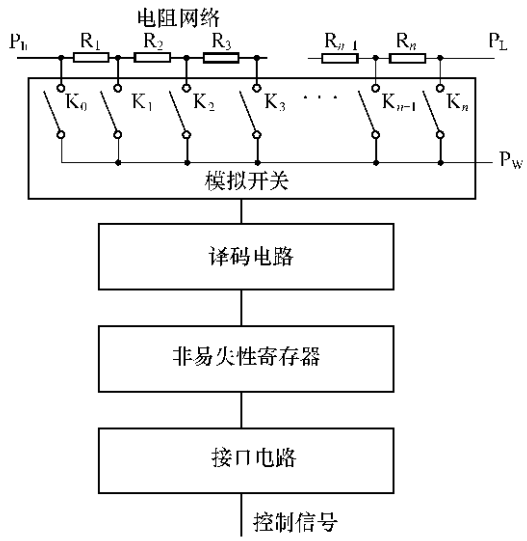


图 5.12　数字电位计结构示意

数字电位计的滑动抽头点少的有 32 个，多的达 1024 个以上，其控制接口常见的有 I^2C、SPI、步进脉冲等形式，如 Xicor 公司生产的 X9221 是一个双数字电位计，20 脚封装，具有 64 个滑动抽头点，总电阻从 2kΩ ~ 50kΩ 可选，接口为 I^2C；X9110 是单数字电位计，14 脚封装，具有 1024 个滑动抽头点，总电阻从 5kΩ ~ 100kΩ 可选，接口为 SPI。数字电位计的应用范围很广，诸如：

- **改变电压放大器的增益；**
- **为比较器和检测器提供可编程的直流电压基准；**
- **控制音频电路的音量；**
- **修整电压放大器电路中的偏移电压误差；**
- **设置稳压器的输出电压；**
- **调整惠斯通电桥电路中的电阻；**
- **控制滤波器电路中的增益特性频率和品质因数；**
- **设置传感器信号调节电路中的比例因子和零点；**
- **更改定时器电路的频率和占空比；**
- **改变 RF 电路中引脚二极管衰减器的直流偏压；**
- **为反馈电路提供一个控制变量；**
- **调整液晶显示屏的对比度；**
- **控制通信系统中发光二极管发送器的功率电平；**
- **设置和调节无线系统中 RF 功率放大器的直流偏压点；**
- **控制音频和家庭娱乐系统的增益；**
- **为 RF 无线系统中的调谐器提供可变的直流偏压；**
- **设置温度控制系统中的工作点；**
- **控制工业系统中传感器的工作点；**

- 调整人工智能系统中的偏移和增益误差等。

在这里我们用数字电位计代替反相放大器的输入电阻或反馈电阻来实现 PGA，图 5.13、5.14、5.15 为常见的 3 种利用数字电位计组成的程控放大器。当选用滑动抽头点数较多的数字电位计调节增益时，可认为增益变化是连续的。

如上所述，图 5.14 要比 5.13 好一些。以图 5.14 所示电路为例，设数字电位计由 n 个电阻 R 组成，P_W 所在位置为 i，则

$$G = -\frac{R \times i + P_{on}}{R_1}$$

式中，R_{on} 为数字电位计内部模拟开关的导通电阻，可见其增益也受 R_{on} 影响。此类电路对于一般精度的程控放大是较适合的，例如音频放大、电机转速控制等。在用于精密测量场合时它有以下不足。

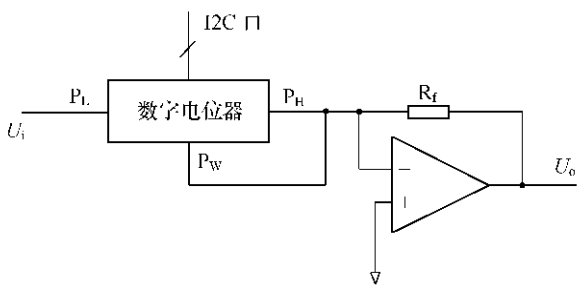


图 5.13　用数字电位计组成的 PGA 方案 1

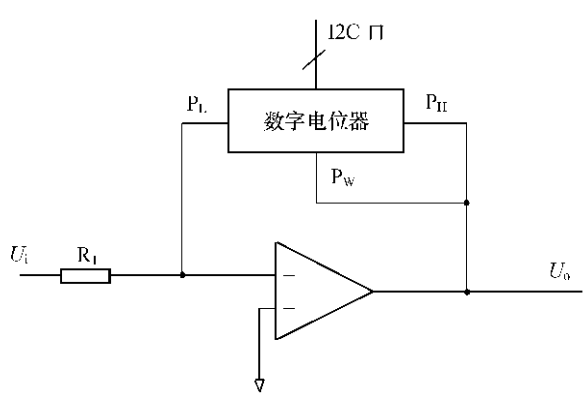


图 5.14　用数字电位计组成的 PGA 方案 2

(1) R_{on} 的影响；

(2) n 个电阻 R 不可能完全等同，i 不同时 R × i 也不同，导致 G 的误差；

(3) 数字电位计的温度系数和外接电阻 R_1 的温度系数不可能完全相同，在温度变化时会导致 G 产生误差。

为此我们可将图改用图 5.15 电路。在本方案中设滑动抽头点 i 从 P_L 端开始计数，则有

$$G = -\frac{R \times (n - i)}{R \times i} \approx \frac{n - i}{i}, i \neq 0$$

这个结果是令人满意的。首先，它利用运放的高输入阻抗使模拟开关的 R_{on} 的影响消除；其次，它利用正态分布的统计原理，使分子和分母各自消除由 R 的不均匀造成的误差，不但使 R 的不均匀造成的增益误差减到最少，而且克服了电路的增益温漂。

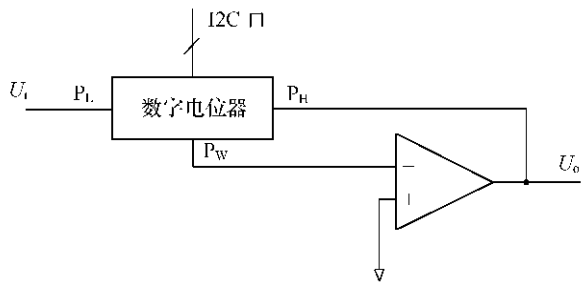


图 5.15　用数字电位计组成的 PGA 方案 3

2. 用 DAC 实现 PGA

将图 2.1 的 T 型网络组成的基本数模转换器（DAC）的接法改变一下，即 R_{fb} 接输入信号，V_{ref} 接运放的输出，可得图 5.16 所示电路。用同样的方法可推得

$$U_O = -\frac{U_i 2^n}{D}$$

可见用 DAC 也能实现 PGA，$G = -\frac{2^n}{D}$，$D \neq 0$，当 D 取最大值为 $2^n - 1$ 时，$G \approx -1$。事实上，图 5.16 电路是经常用的，但要记住控制值 D 不可为零，否则相当于开路。

在选择 $n \geqslant 12$ 的 DAC 时，增益 G 的变化基本上是连续的。因 T 型网络的 DAC 的精度较高，这样的 PGA 控制精度也较高，在高精度检测场合可以考虑使用此方案。

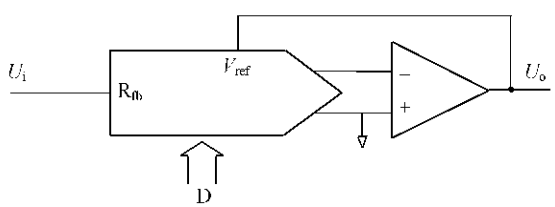


图 5.16　用 D/A 来实现 PGA

5.2.2　自动触发电平调节

在智能测控系统及仪器中，经常要控制一些事件的发生，这些事件的发生条件的设定就是所谓的自动触发电平调节。如示波器在自动触发方式时要用一定的输入信号幅值产生示波触发，计数器要用一定的输入信号幅值来触发计数功能，一些运行装置在突发事件发生时用某一模拟量的值来启动数据记录器等等。

这类情况都可以用如图 5.17 的电路方案来实现自动触发电平调节。比较器的一端接输入信号，另一端接 DAC 的输出，DAC 的输出由系统设定，当 DAC 输出不同的值时事

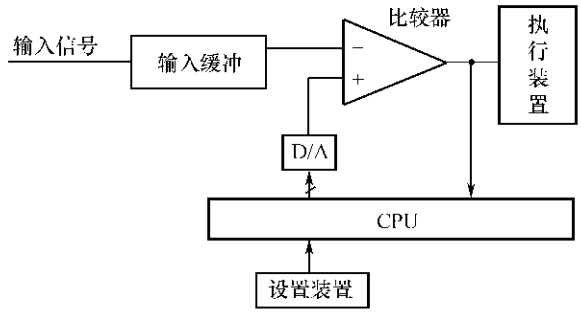


图 5.17　自动触发电平调节示意

件发生的条件就发生了变化。

5.2.3　零点问题

零点是指一个检测系统在输入为零时的输出值，一般情况下，人们希望系统的零点是一个不变的常数，更多情况下希望系统的零点是零值，例如电压表在输入电压为 0 时，其输出值应为零，电子秤在未称货物时其指示值也为 0。但实际情况往往不尽如人意，这是因为所有的电压放大器的零点都存在着零点漂移现象即零漂。零漂不但影响系统精度，当零漂超过一定值时还会使系统不能正常工作，所以系统调零是必须的。

早期的电子检测系统大都是通过调节电位器来达到调零的目的，对于智能检测系统来说，完全可以自动实现这些功能。

一、自动调零

（一）用数字电位器实现自动调零

用数字电位器实现调零的电路如图 5.18 所示，输入信号 U_i 经放大后变为 U_{o1}，在输入信号 $U_i=0$ 时，U_{o1} 未必为零，此时主控制器可通过 I^2C 口调节数字电位计使 $U_{o2}=0$。

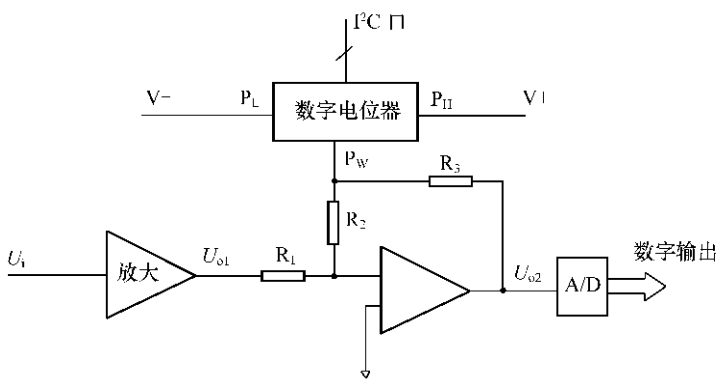


图 5.18　用数字电位器实现自动调零

（二）用 D/A 实现自动调零

用与图 5.18 相同的加法器原理，通过 DAC 给出调节电压同样可以实现自动调零，基本电路如图 5.19 所示。

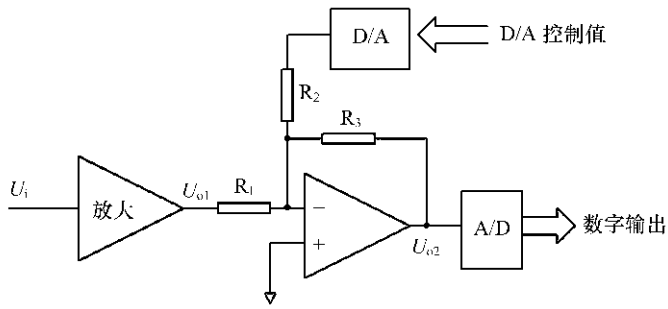


图 5.19　用 D/A 实现自动调零

二、自动回零

自动回零是在确认输入为零时，不管此时 A/D 转换值是否为 0，都记下该值作为零值 D_Z，然后将 A/D 的转换值 D 与零值 D_Z 相减作为输入量的 A/D 值参与运算，其结果使检测系统的输出在输入为零时必然也为零，这种方法在智能检测系统中应用非常普遍。但它是有条件的，即 D_Z 不能过大，D_Z 值过大时须先经过自动调零，再进行自动回零。

对于一般的放大器来说零漂是小范围的，如选择 OP07 作为前置放大时，在常温下零漂可以控制在 mV 级范围内，此时采用自动回零是非常有效的方法。

5.3　测量精度的提高

5.3.1　随机误差处理方法

测量误差是由于测量过程中一系列因素造成的，随机误差是指在相同条件下多次测量误差的绝对值和符号的变化没有确定规律。例如，同一个人拿同一卡尺量同一工件，每次测量的结果与工件的实际尺寸都会有一定的误差，误差可能是正差也可能是负差，可能这次大一些下次又小一些，这就是典型的随机误差。

设被测信号的平均值为 S，在 N 次测量中所得值为 S_1、S_2…S_i…S_n，在第 i 次测量中因噪声引起的成分为 n_i，有用信号为 S_i，N 次测量的信号成分之和为 $\sum_{i=1}^{n} S_i = N \cdot S$，N 次测量的总噪声因有正有负，故用均方值表示为

$$\sqrt{\sum_{i=1}^{n} n_i^2} = \sqrt{N \cdot n^2} = \sqrt{N} \cdot n$$

其中，n 表示噪声的平均值，信噪比 $= \frac{N \cdot s}{\sqrt{N} \cdot n} = \sqrt{N}\frac{s}{n}$，此式表明：N 增大时信噪比也增大，当测量次数为 N 时，信噪比提高到原来平均信噪比 s/n 的 $\sqrt{N}$ 倍。因此，我们常常用 $\frac{1}{N}\sum_{i=1}^{n} x_i$ 求值，N 根据精度、实时性来选择合适的值。

5.3.2　系统误差的处理

系统误差是指在相同条件下，多次测量误差的绝对值和符号恒定或呈现一定规律。系统误差靠增加测量次数是不能解决问题的，针对不同的系统误差需采用不同的处理方法。

一、利用误差模型修正系统误差

不同的测试系统具有不同的系统误差，先通过分析建立误差模型，再由误差模型求出误差修正公式。例如，可以为某一同相放大器建立图 5.20 所示的模型，由于运放本身失调电压、失调电流等的影响，再加上外围电阻误差、连接电路等影响，放大器实际特性和理想特性有区别，必然会引起系统误差。将系统误差折合在输入端用 e 表示。

在理想情况下，$U_o = \frac{R_1 + R_2}{R_1} U_x$，由于输入端误差 e 的作用，当测得 $U_o = 0$ 时，发现实际

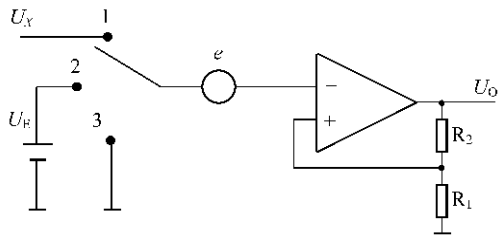


图 5.20　直流放大器的误差模型

的 $U_x \neq 0$ 当测得输出为某一值如 $U_o = U_{oc}$ 时，对应的 $U_{xc} \neq U_{oc}\dfrac{R_1}{R_1 + R_2}$。为此可设 $U_x = b_1 U_0 + b_0$，作为实际的测量模型，只要求出 b_1、b_0 即可求得实际的输入输出数学公式。为此，在输入端加一个转换开关，求两组输入输出值：

开关置于 3，使输入为 0，测得 U_{oZ}；

开关置于 2，使输入为基准值 U_E，测得 U_{oE}，得方程组

$$\begin{cases} b_1 U_{oZ} + b_0 = 0 & (1) \\ b_1 U_{oE} + b_0 = U_E & (2) \end{cases}$$

联合求解：(2) - (1)，$b_1 (U_{oE} - U_{oZ}) = U_E$，$b_1 = \dfrac{U_E}{U_{oE} - U_{oZ}}$

(2) + (1)，$b_0 = \dfrac{1}{2}[U_E + b_1 (U_{oZ} + U_{oE})] = \dfrac{U_E}{2}\left(\dfrac{U_{oE} - U_{oZ} + U_{oZ} + U_{oE}}{U_{oE} - U_{oZ}}\right) = \dfrac{U_E U_{oE}}{U_{oE} - U_{oZ}}$

最后求得

$$U_x = \frac{U_E}{U_{oE} - U_{oZ}} U_o + \frac{U_E U_{oE}}{U_{oE} - U_{oZ}}$$

当测得一值 U_o 时，对应的 U_x 用上式修正即为真值。本例适合于一般的线性系统。

二、利用校正数据表修正系统误差

如果对系统无法建立模型，可以通过建立校正数据表的方法修正误差，步骤为

(1) 在系统的输入端逐次加入一组已知的标准值 $x_1, x_2, \cdots, x_n$，并实测出对应的测量结果 $y_1, y_2, \cdots, y_n$；

(2) 用 y_i 作映射，将 $(x_1, x_2, \cdots, x_n)$ 存储在 PROM 或带掉电保护的 RAM 中建立了一张校准数据表；

(3) 实际测量时令微处理器根据实测的 y_i 去访问内存，读出其中的 x_i，x_i 即为经过修正的测量值；

(4) 若实际测量 y 值介于某两个标准点 y_i 和 y_{i+1} 之间，为了减少误差，可在查表基础上作内插计算来进行修正。采用内插技术可以减少校准点从而减少内存空间，最简单的内插是线性内插。线性内插的原理如图 5.21 所示，设测得输出值为 y，y 介于 y_i 和 y_{i+1} 之间，对应的实际被测值为 x，由于 A 点和 C 点相距很近，为此 A、B、C 3 点认为在一条直线上，以斜率 AC 代替斜率 AB，即

$$\frac{x_{i+1} - x_i}{y_{i+1} - y_i} = \frac{x - x_i}{y - y_i}$$

则有

$$x - x_i = \frac{x_{i+1} - x_i}{y_{i+1} - y_i}(y - x_i)$$

$$x = x_i + \frac{x_{i+1} - x_i}{y_{i+1} - y_i}(y - x_i)$$

由此求得被测量 x。

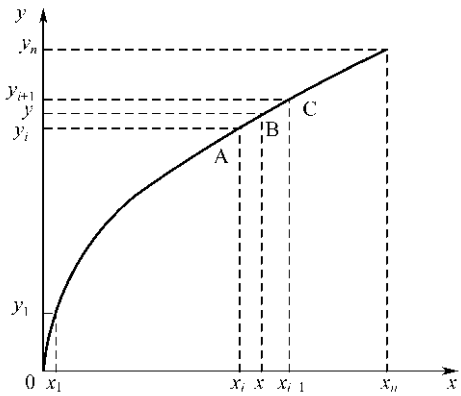


图 5.21　线性内插法的原理

三、曲线拟合法修正系统误差

（一）曲线拟合基本原理

严格地来讲，任何系统都不可能是线性的，因此，在精密测量时都要做一定的修正。插值法可适合于大多数非线性系统，但有两个缺点：其一是存储数据修正表需要较多的内存；其二是获得这组数据需要大量的时间去测试，这一点是最不能容忍的。曲线拟合只需少量的测试点就可较准确地得到所有的被测值，在检测系统中得到广泛的应用。下面举例说明其基本原理和方法。

曲线拟合就是寻找一个与系统实际特性相吻合的函数，并求出函数的有关参数，用函数来代替该段数据。这个函数从原则上讲可以是任何连续函数，但一般情况下，为了计算方便选用多项式，当多项式为一阶函数时就是直线拟合，直线拟合是实际曲线拟合的特例。

图 5.22 为某检测系统的实际输入输出曲线。根据观察和经验，可用三阶多项式来拟合，设三阶表达式为 $a_3x^3 + a_2x^2 + a_1x + a_0 = y$，取其代表性的 4 组数据 (x_1, y_1)、(x_2, y_2)、(x_3, y_3)、(x_4, y_4) 代入，可得一组方程：

$$\begin{cases} a_3x_1^3 + a_2x_1^2 + a_1x_1 + a_0 = y_1 \\ a_3x_2^3 + a_2x_2^2 + a_1x_2 + a_0 = y_2 \\ a_3x_3^3 + a_2x_3^2 + a_1x_3 + a_0 = y_3 \\ a_3x_4^3 + a_2x_4^2 + a_1x_4 + a_0 = y_4 \end{cases}$$

联立可求得 a_3、a_2、a_1、a_0，从而以函数 $y = a_3x^3 + a_2x^2 + a_1x + a_0$ 来代替该段曲线，计算

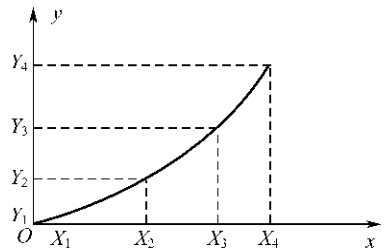


图 5.22　曲线拟合举例

时用嵌套形式 $((a_3x + a_2)x + a_1)x + a_0$，可以减少乘法量。

例 5.1：已知某厂生产的铂电阻 PT1000 温度传感器的电阻为温度的 2 次函数，并得到 10 、50 和 100 3 个温度点附近的 3 组参数，用 VB 编一个程序求 PT1000 的表达式。

解：设温度 T 和电阻 R 的关系为 $R = aT^2 + bT + c$，设计图 5.23 所示的操作界面，当鼠

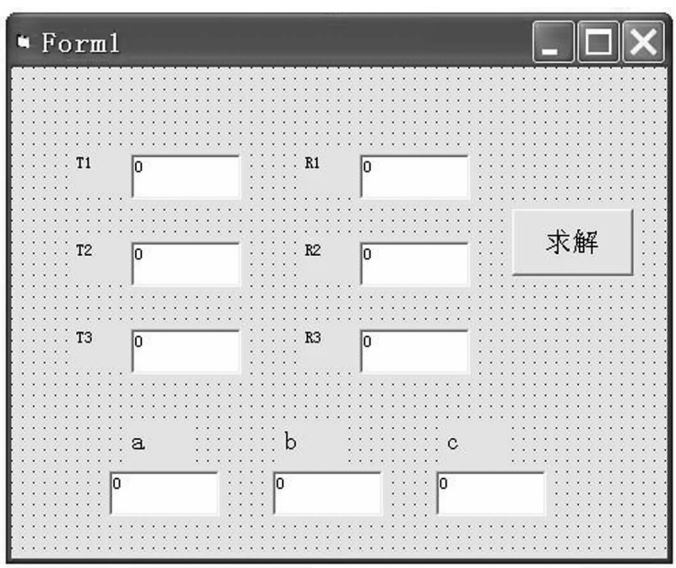


图 5.23　$T = aR^2 + bR + c$ 的界面

标单击求解时，程序自动获取（T1，R1），（T2，R2），（T3，R3）组成三元方程组，并将求得结果显示出来。用 VB 编写的代码如下：

```
Dim T1 As Single
Dim T2 As Single
Dim T3 As Single
Dim R1 As Single
Dim R2 As Single
Dim R3 As Single
Dim a As Single
```

```
Dim b As Single
Dim c As Single

Private Sub Command1 _ Click ()
T1 = Text1 (0) . Text
R1 = Text1 (1) . Text
T2 = Text1 (2) . Text
R2 = Text1 (3) . Text
T3 = Text1 (4) . Text
R3 = Text1 (5) . Text
a = (R2 - R3 - (T2 - T3) * (R1 - R2) / (T1 - T2)) / ((T2 * T2 - T3 * T3) - (T2 - T3) * (T1 * T1 - T2 * T2) / (T1 - T2))
b = (R1 - R2 - a * (T1 * T1 - T2 * T2)) / (T1 - T2)
c = R1 - b * T1 - a * T1 * T1
Text1 (6) . Text = a
Text1 (7) . Text = b
Text1 (8) . Text = c
End Sub
```

程序编译后运行，如输入 3 **组测试值**（9.96，1038.875），（54.98，1213.125），（99.04，1384.829），**解算结果为** a = 0.0002980113，b = 3.851149，c = 1000.488，**见图** 5.24。

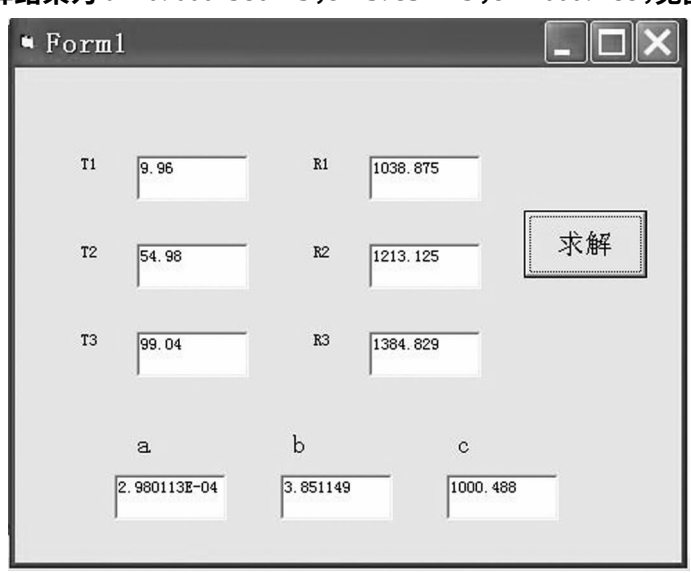


图 5.24　$T = aR^2 + bR + c$ **的系数求解结果**

（二）分段曲线拟合

如果所测结果呈现不规则曲线，不可能用一个连续函数描述，可以分段进行拟合。如图 5.25 **的曲线可分为** 5 **段：**OA **段可用二阶多项式；**AB **段可用三阶多项式；**BC **段可用三**

阶多项式；CD 段可用二阶多项式；DE 可用一阶多项式（直线拟合）。

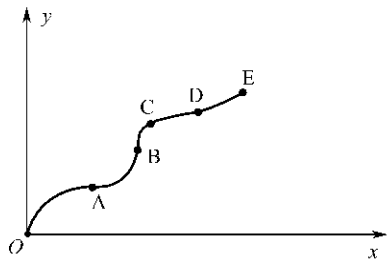


图 5.25　分段拟合法举例

曲线拟合可遵循以下原则：

⑴ 在满足精度条件下尽量用低阶数曲线拟合，这样可以节省计算时间提高实时性。

⑵ 分段时尽可能等距离分段以简化程序设计。

5.3.3　粗大误差的处理

粗大误差是指测量值明显偏离实际值的误差。产生粗大误差的原因有测量方法不当、操作疏忽或失误、测量条件突然变化等。粗大误差对应的测量值应剔除，处理步骤为

⑴ 求算术平均值

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{n} x_i$$

⑵ 求各项的剩余误差 $v_i = x_i - \bar{x}$，当 $|v_i|$ 明显地变大时可判断为是粗大误差，去掉该次测量值。

5.4　数 字 滤 波

被测信号中往往会有干扰或噪声，严重时会影响测量结果，甚至会使系统瘫痪，为此系统中常常用屏蔽、硬件滤波等方法处理。很多情况下数字滤波可以不用硬件而起到滤波的效果，它可以取代或配合硬件滤波器等达到抑制干扰的目的。

5.4.1　中值滤波法

取 n 个值（n 取奇数）$x_1, x_2, \cdots, x_n$，将其排序为 $x'_1 > x'_2 > \cdots > x'_i > \cdots > x'_n$，取中间值作为该次测量值，称为中值滤波。n 一般取值不会太大，如取 3、5、7、…、15 等，该法适合于直流或缓慢变化的信号，不适合于信号变化剧烈的场合。滤波程序分为两步：①排序：可用冒泡法，沉底法等；②取中值。

5.4.2　平均滤波法

（一）简单平均滤波法

采样 n 个值，用公式 $x = \frac{1}{n}\sum_{i=1}^{n} x_i$ 求值，称为简单平均滤波法。为了使除法简单、省

时，n 可取 2 的幂值，从而用移位法实现除法。

（二）去极值平均滤波法

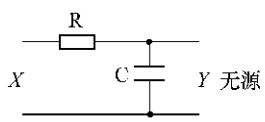
采样 n 个值，求其最大值、最小值去之，再求平均值，n 可取 2 的幂 +2。

（三）移动平均滤波法

以上两法须先采 n 个点才可计算，需时间较长，为此可把上次采样值和新采样值一块平均算之。设总采样点数为 n，原采样值用 n - m 个，新采样值用 m 个，将原队列中前 m 个舍去，剩下的 n - m 个和新增值 m 个组成新队列一块平均。

5.4.3　低通数字滤波

低通滤波器分无源和有源滤波器两种，如图 5.26 为一阶无源低通滤波器，图 5.27 为一阶有源滤低通滤波器。数字低通滤波器实际上就是一阶差分方程计算式。



5.26　无源一阶低通滤波器

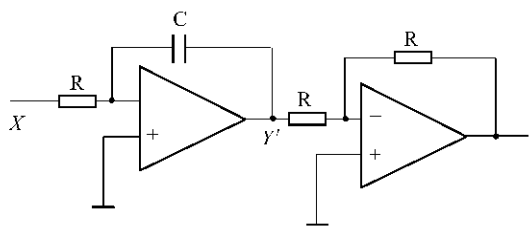


图 5.27　有源一阶低通滤波器

设低通的传递函数

$$G(s) = \frac{y(s)}{x(s)} = \frac{1}{\tau s + 1}$$

式中 $\tau = RC$，对应的一阶数字滤波可用以下的差分方程式实现：

$$y(n) = (1 - \alpha)\, y(n - 1) + \alpha x(n)$$

其中 $\alpha < 1$，$\alpha = 1 - e^{-T/\tau}$。这里 τ 为滤波平滑系数；T 为采样周期；x (n) 为本次采样值；y (n) 为本次滤波的输出值；y (n - 1) 为上次滤波的输出值。一般取值 $T \ll \tau$，所以 $\alpha \ll 1$。这种数字滤波对缓慢变化的有用信号中的高频干扰滤波是非常有效的。在实际工作中人们往往根据经验取一个 α 值进行滤波，如取 $\alpha = \frac{1}{4}$，则得差分方程为

$$y(n) = y(n - 1)\ \frac{3}{4} + x(n)\ \frac{1}{4}$$

第 6 章　常见模拟量信号的检测方法

6.1　信 号 概 述

6.1.1　信号的分类

对于检测系统来讲，信号可分为已知信号、未知信号两种。已知信号又称为标准信号，是可以控制的，如低频信号发生器产生的固定频率、幅值、相位的正弦波、三角波、方波、锯齿波；又如高频信号发生器可产生的高频正弦波、正交波；标准无线电波发生器发生的标准无线电波；基准电源产生的恒压源、恒流源等。未知信号就是尚未确定其特征参数的待测信号。检测系统所研究的未知信号往往是通过各种接收器或传感器产生的待测信号，此类信号一般已确定其信号的类型，比如说它是直流还是交流，是电压还是电流等，但并不知道其具体参数如瞬时值、幅值、频率等。已知信号已在第 2 章研究过了，未知信号的检测是本章的研究重点。

6.1.2　传感器及变送器

传感器是指直接将自然界的变量（如物理量、化学量等）转化为测控系统可识别的量（如电压、电流等）的装置。由于传感器输出的值的类型不尽相同，即使是同类型其规格大小也不尽相同，为了使测控系统变得简单化、标准化，需要一个装置将传感器的输出信号变成标准的信号，这个装置就是变送器。变送器一般将信号变为标准的电压或电流，其后面可直接接标准的显示仪表或接具有标准输入的工业控制器。传感器及变送器在系统中的作用如图 6.1 所示。

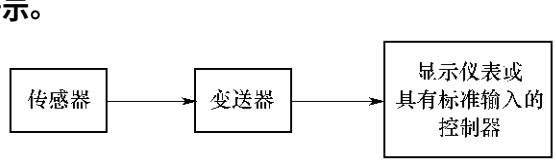


图 6.1　传感器和变送器的地位

6.1.3　模拟信号的基本分类

检测系统常见的模拟量可以分为以下类型：

⑴ 直流电压型；

⑵ 交流电压型；

⑶ 直流电流型；

⑷ 交流电流型；

⑸ 电阻型；
⑹ 电容型；
⑺ 频率型；
⑻ 时间型；
⑼ 相位型；
⑽ 数字协议型，等等。

6.2　各类信号的检测方法

6.2.1　电压类信号的检测方法

由于 A/D 的输入信号是直流电压，电压类模拟信号居多。电压类信号又可分为直流电压和交流电压两类，传统的检测系统及低速检测系统对直流电压和交流电压是分别对待的，而较高速的智能检测系统可不加区分而直接统一处理。如图 6.2 所示信号 V 是含有直流电压和多种交流电压的信号，可先用 A/D 变换成数字信号，再通过 FFT 变换求出其参数，如直流分量、幅值、周期、有效值等。

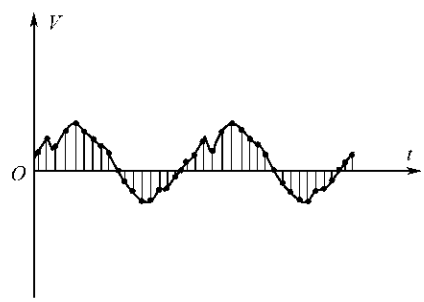


图 6.2　对一电压信号的采样结果

在 A/D 变换前要求得适合于 A/D 输入端的较大信号，被测信号比较小需要放大，而被测信号比较大时又需要衰减。由于信号中含有噪声，放大时连同噪声也一起放大。为此可先滤去大部分噪声后再放大。总之，信号不同，其处理方法也不相同。一般的 A/D 的前置处理系统如图 6.3 所示，其中放大器是必需的，其余的组件可根据实际情况选用。下面对各部分作一简述。

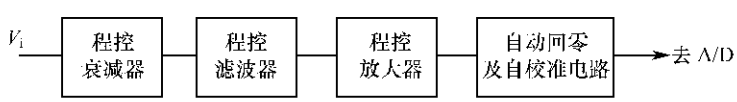


图 6.3　A/D 电路的前置处理部分

一、程控衰减器

当被测信号的幅值较大时，须使用衰减器使信号适合于 A/D 的量程。如输入电压为 220V，A/D 的量程为 5V，需衰减 50 倍左右。为了使电路能适合于测试较大范围的信号，

在智能测试系统中往往采用程控衰减器。图 6.4 为一种程控衰减器。

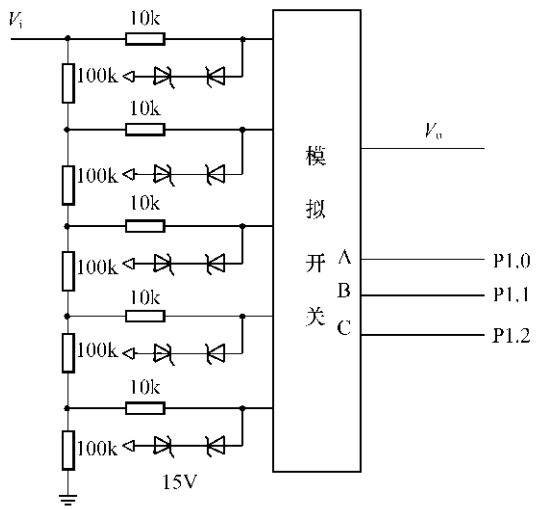


图 6.4　一种程控衰减器电路

该衰减器使用了 5 个 100kΩ 电阻将输入电压 V_i 分为 5 段，分别是：1/5、2/5、3/5、4/5、5/5。用试接法来判断输入信号的大小而确定选用某一挡。由于模拟开关的最大输入电压为 ±15V，故选用 ±15V 的稳压管保护其输入端不被毁坏，也可用其它保护方法。

二、程控滤波器

如果输入信号含有较大幅值的干扰时，应先滤波。如图 6.5 所示的被测信号中含有较大的高频干扰信号，个别尖峰干扰几乎和有用信号的幅值相同，而尖峰部分包含的信号是不可丢失的。信号不放大，尖峰部分已达到 A/D 的量程，如果直接采样的话，所得信息的信噪比太小。此时可先用低通 (LP) 滤波，经过 LP 后信号的信噪比变大，可继续放大提高精度。

根据有用信号和噪声的频率不同，可选用不同的滤波器，如低通滤波器 (LP)、高通滤波器 (HP)、带通滤波器 (BP) 等。必要时可选用或设计程控滤波器，程控滤波器可在传统的 RC 滤波器、LC 滤波器的基础上改制，也可直接选用开关电容型滤波器。开关电容型滤波器是集成单片滤波器，因其体积小、易于程控而深受欢迎。图 6.6 为开关电容滤波器的控制方式示意，其控制端只须一路数字脉冲信号，滤波器的特征频率 (如带通的 f_0，低通的 f_H，高通的 f_L 等) 是脉冲输入端的频率的 1/N (N 可设定)。如 N = 32，当脉冲输入端的频率为 3200Hz 时，其特征频率为 100Hz。

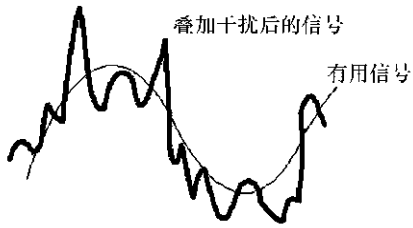


图 6.5　含有干扰的被测信号举例

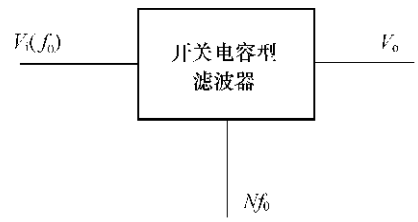


图 6.6　开关电容型滤波器控制方式

三、仪表放大器

由于仪器仪表对放大器的要求较高，因此凡是具有较高的精度、较高的稳定性、较低的漂移等优秀指标的放大器，都可以称为仪表放大器。最常见的一种仪表放大器如图 6.7

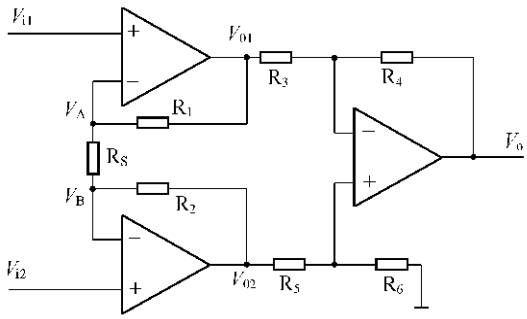


图 6.7　仪表放大器的基本原理

所示，该种放大器由 3 个低漂移、低失调运放组成，当取 $R_1 = R_2$、$R_3 = R_5$、$R_4 = R_6$ 时，由运放“虚短路”和“虚断路”原理可得

$$V_A = (V_{01} - V_{02}) \frac{R_S + R_1}{2R_1 + R_S}$$

$$V_B = (V_{01} - V_{02}) \frac{R_1}{2R_1 + R_S}$$

$$V_A - V_B = V_{i1} - V_{i2} = \frac{R_S}{2R_1 + R_S}(V_{01} - V_{02})$$

而

$$V_0 = \frac{R_4}{R_3}(V_{02} - V_{01})$$

所以

$$V_0 = \frac{2R_1 + R_S}{R_S}\frac{R_4}{R_3}(V_{i2} - V_{i1})$$

当取 $R_1 = 10k\Omega$，$R_S = 10k\Omega$ 时

$$V_0 = 3\frac{R_4}{R_3}(V_{i2} - V_{i1})$$

作为固定放大器经常这样设计，如取 $R_4 = 100k\Omega$，$R_3 = 10k\Omega$，增益为 30。该放大器输入阻抗高，作前置级放大具有良好的特性。有的公司利用此电路原形设计出单片的仪表放大器。如 AD 公司的 AD620 将 $R_1 \sim R_6$ 全部固定，将 R_S 引出作为外接来决定增益，深受广大用户欢迎。但作为高精度放大器，AD620 有其不足之处，因内部集成电阻和外接 R_S 温度系数不可能保持一致，当温度变化范围较大时会造成误差。

四、隔离放大器

隔离放大器定义 输入端和输出端之间没有共同的参考电位点，即输入端的任意一端和输出端的任意一端是绝缘的。一般要求绝缘电压能达 1000V 以上，如图 6.8 为测试隔

离放大器的隔离特性的电路，在输入端和输出端加上 1000V 以上的直流电压或交流电压并串入安培表，通电 1min 以上安培表指示始终为零。隔离放大器具有以下优点：

⑴ 输入端的干扰不会直接到达输出端；

⑵ 输出端和输入端可建立各不相同的电位参考点；

⑶ 多路通道都使用隔离放大器时相互之间不会有影响。

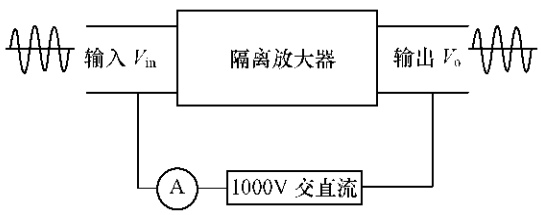


图 6.8　隔离放大器的外部特性测试

放大器隔离的方法有变压器和光电两种方式，前者如 AD202，AD210 等，后者如 SLC800 等。变压器隔离型采用调制解调电路以厚膜封装的形式出现，虽然体积大、价格高，但在精密测量系统中目前还是首选的产品；光电隔离型体积小、价格低，具有很广阔的前景，但目前其精度还较差一些，有待进一步改进。

五、关于屏蔽的自举

输入信号输入端常常用屏蔽的电缆引入，屏蔽一般和外壳或电源的地连到一起。为了提高系统对输入信号的共模干扰的抑制能力，有时屏蔽不和电源地相连而和信号的中心电位相连。图 6.9 为一种屏蔽自举电路，假如在某一时刻，$V_{01}=10V$，$V_{02}=4V$，这说明 V_{i1} 和 V_{i2} 输入了共模信号，共模信号往往是从较长的屏蔽电缆线上引入的。共模信号较大时会使输入端的两运放都处于饱和状态不能工作，采用如图的自举电路后，屏蔽上的电位 $V_{G'}=V_G=\frac{V_{01}+V_{02}}{2}=\frac{10V+4V}{2}=7V$，这样做保证了屏蔽上的电位浮动在所处理的信号的中点，从而提高系统的共模抑制能力。

六、求交流平均值的电路

在许多场合我们只需测量交流电压的平均值，这时可采用如图 6.10 所示的电路，该电路实质上是全波整流滤波电路。运放 A1 为正向放大器提供一定增益使输入阻抗提高，运放 A2 和 A3 组成有精密整流及滤波。在 V_a 的正半周期 A2 输出为负，D2 导通，D1 截止，R_4 为反馈电阻，$V_b=-V_a\frac{R_4}{R_3}$ 在 V_a 的负半周期，A2 输出为正。D1 导通，D2 截止，b 点的电位等于 a'点的电位，$V_b=0$。可见 V_b 为半波整流输出。A3 组成加法器式滤波电路。按照图中的电阻取值，即 $R_9=R_{10}=20k\Omega$，$R_3=R_4=R_8=10k\Omega$，当电容 C 开路时，在 V_a 的正半周 $V_o=2V_b+V_a$ 在 V_a 的负半周 $V_o=V_b+0$，V_o 为 V_a 的全波整流；当电容 C 取合适的值时，V_o 为全波整流后的平均值。图 6.11 为波形合成的全过程。

6.2.2　电流类信号的检测

一、传统的手动分挡测量方法

测量电流的基本原理是通过电阻将电流变换电压，图 6.12 为一种用数字电压表分挡

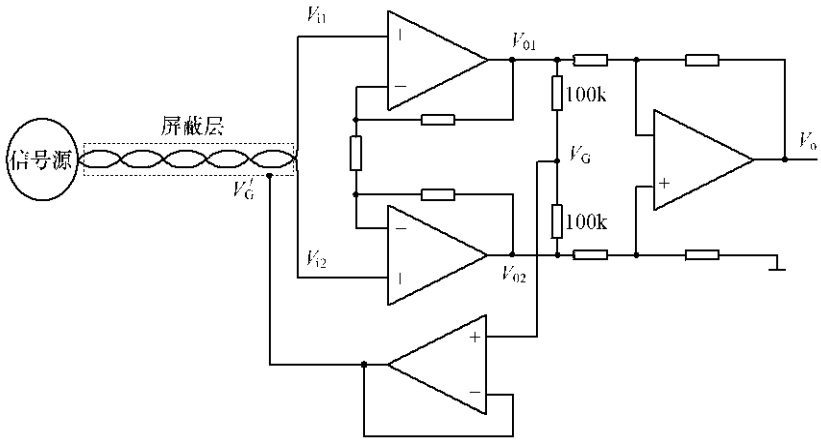


图 6.9　**一种屏蔽自举的电路**

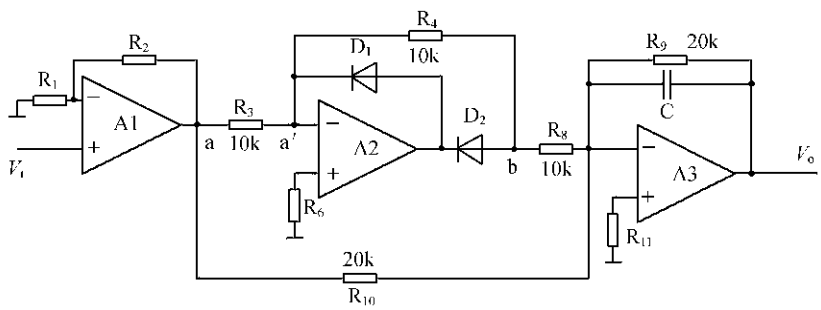


图 6.10　**一种精密整流滤波电路**

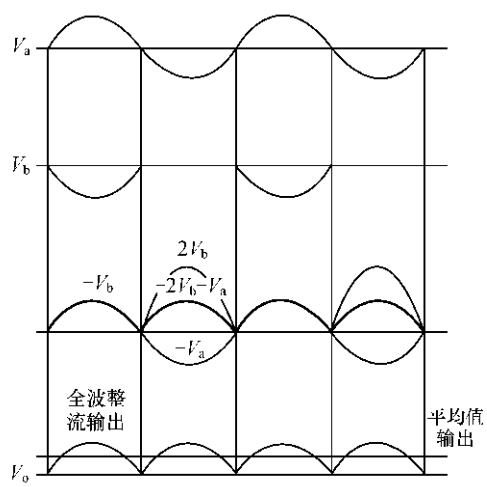


图 6.11　**精密整流电路工作过程**

测量直流电流的基本电路，该电路将输入电流分为 20A、200mA、20mA、2mA 4 **个量程，转**

换电阻用 0.01Ω、0.99Ω、9Ω、90Ω 4 个电阻串联，将 4 种量程的电流接入电路的不同点，使得每种量程的电流在满量程时得到的电压都是 0.2V，从而用 0.2V 的数字电压表配合不同的显示单位及小数点位置，指示被测电流的大小。这种方法是数字万用表常用的测量方法。

二、自动分挡测量法

在自动测试系统中，一般以电流信号的最大值确定所需电阻，如最大值为 100mA，A/D的输入最大值为 10V，可选电阻为 0.1kΩ，如果将自动量程分为 4 个挡位，可用 4 个 25Ω 的电阻串联，通过模拟开关引出不同的信号，类似于前面所述的 PGA 控制方式，电路如图 6.13 所示。这种方法对于直流电流和交流电流的测量都适用。

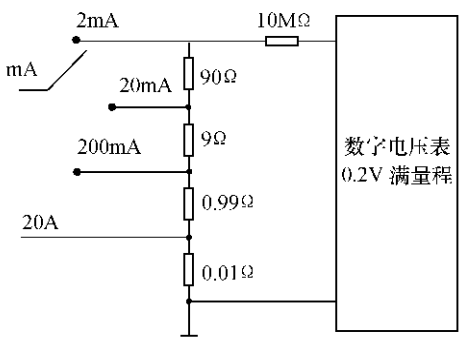


图 6.12　用数字电压表分档测量直流电流

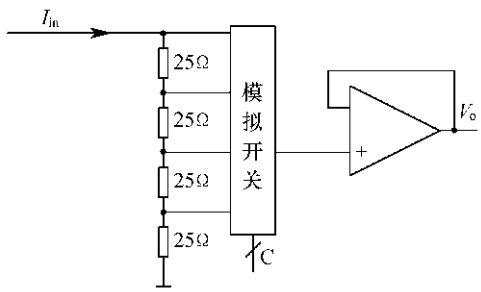


图 6.13　一种自动测量电流的输入电路

6.2.3　相位型信号的检测

在检测系统中，相位定义为同频的两路信号之间的相位之差。严格来讲是指两路正弦信号的相位差，但如果是方波、三角波等均匀波形时，也可求其基波的相位差。当所处理的信号频率不是很高时，也可以用单片机求解相位信号。

一、软件分析法

如图 6.14 所示，假如被测信号是不含直流分量的标准的正弦波 X_1 和 X_2，用同步采样的方法将两路信号量化，对其进行分析，求得 X_1 的两个同类过零点，求得 X_2 的一个同类过零点（这里同类过零点是指都是由正到负或都是由负到正的过零点），由采样频率和采样点数通过 X_1 的两个同类过零点求得信号的周期 T，通过 X_1 的过零点与 X_2 的过零点之间的时间差为 ΔT。设 $X_1 = A_m \sin(\omega t)$，$X_2 = B_m \sin(\omega t + \varphi)$，$\varphi = \dfrac{\Delta T}{T} 360°$。

这种方法是借助数据采集来完成的，其精度受采样点数和采样频率的限制，但在需要同步采样的场合可以兼而求得，图 6.15 为一种求相位的采集电路。

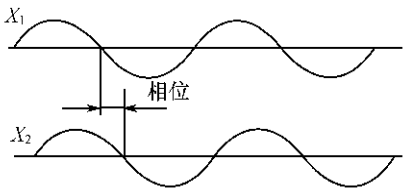


图 6.14　相位的定义

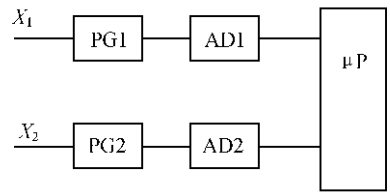


图 6.15　软件分析法求相位

二、过零比较器法

设 X_1、X_2 为不含直流分量的正弦波或三角波，将 X_1、X_2 分别经过两个过零比较器变为方波，利用两个方波的上升沿或下降沿的时间差和其中一个方波的周期可求得相位，算法如上。图 6.16 为用中断法通过过零比较器输出的下降沿求相位的电路，过零比较器的整形过程见图 6.17。这里要求单片机内部定时器的计数频率与被测信号频率相比足够高，例如相位测试分辨力为 0.1，定时器的时钟频率应为被测信号频率的 3600 倍。用过零比较器法不需要 A/D 转换器。

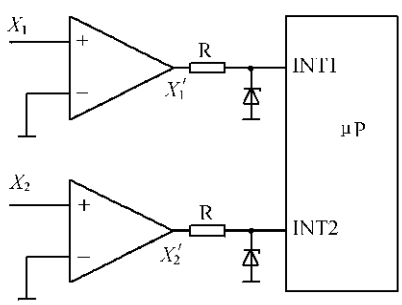


图 6.16　过零比较器法求相位的电路

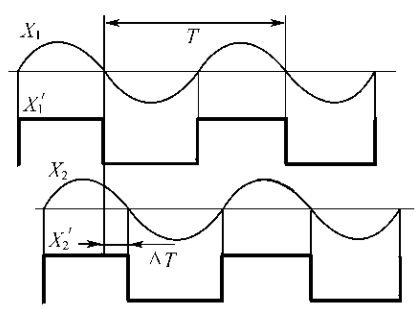


图 6.17　过零比较器整形过程

6.2.4　时间型信号的检测

时间型信号又称为脉宽型信号。相当于记录时间的秒表，将被测信号经电平转换变为电平适合于微处理器处理的信号，如果时间不是太短，可直接利用微处理器的定时器求得。如图 6.18 的电路可以用查询的方式采样电平求取时间 在信号的上升沿启动内部定

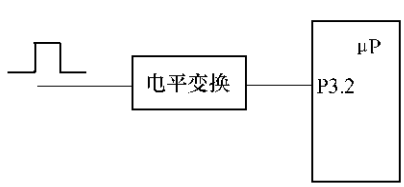


图 6.18　一种用单片机求时间的电路

时器，在信号的下降沿关闭定时器，最后用定时器的计数值和时基确定所求的时间值。设测试的时间为变量 test _ time，时间单位为 ms，单片机选用 89C52，不断更新数据测得时间

的程序如下：

```
#include  < AT89X52. h >

sbit bit _ v  =  P3 2；

unsigned int T1 _ flow _ counter；
float test _ time；//
bit data test _ begin  =  0；



//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//                              主程序
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
void main (void)
 {
//- - - - - - - - - - - 通用初始化 - - - - - - - - - - -
TH1  =  0x64；     //T1 方式 2 作 0. 1ms 秒定时 TH1 = A0H，晶振用 12MH
TL1  =  0x64；     //TC = 12000000 /12 /100 (64h) = 10000HZ，自动装载
TMOD  =  0x20；  //T1 方式 2
EA  =  1；

while (1)
   {//- - - - - - - - - main loop begin - - - - - - - - -
  if (！bit _ v && ！test _ begin)
      {
     TR1  =  1；            //start Timer1
     test _ begin  =  1；
     }
  if (test _ begin && bit _ v)
      {
     TR1  =  1；  //stop Timer1
     test _ time  =  (T1 _ flow _ counter * 65536  +  TH1 * 256  + TH1) /10. 0；
     test _ begin  =  0；
     }
  } //- - - - - - - - main loop end - - - - - - - - -
}



//- - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
//Timer1 中断程序
//- - - - - - - - - - - - - - - - - - - - - - - - - -
void T1 _ int (void) interrupt 3
{
T1 _ flow _ counter + + ;
}
```

该程序用 Timer1 作为定时计数器，为了防止溢出设置一个软件计数器 T1 _ flow _ counter，Timer1 每中断一次 T1 _ flow _ counter 就自动加 1，由 Timer1 中断程序完成。

如果被测时间极短，单片机的计数器时基不能满足测试要求时，需要另外设计电路。图 6.19 为求极短时间的电路之一。电路中串行输出高速计数器及与门等电路全部采用高速器件，P1.0 首先将计数器置零，再 P1.1 用高电平打开计数器与门等待被测信号的升沿到来后启动计数，一直到被测信号下降沿到来后结束计数。高速计数器设计成具有串行输出的功能，μP 用 P3.0 和 P3.1 将计数器的值取回。

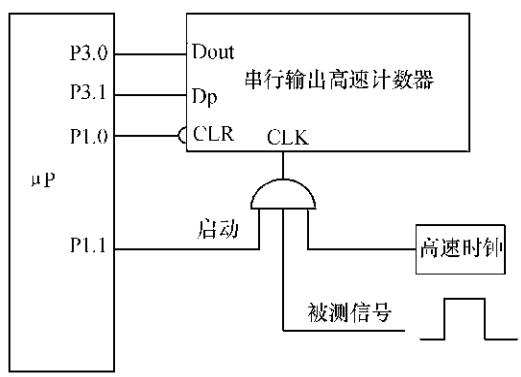


图 6.19　求极短时间的电路之一

6.2.5　电阻型信号的检测

一、恒流法测电阻

测量电阻的最简单的方法是根据欧姆定理以一个恒定电流通过电阻先变成电压再求之。图 6.20 为恒流法测电阻的基本电路，R_x 为被测电阻，I_C 是已知的恒流源。图 6.21 为常见的恒流源产生电路之一，图中 V_e 为基准电压源，R_0 为标准电阻，I_C 为流过负载的电流。因 $V_- = V_+ = V_e$，$I_C = \dfrac{V_e}{R_0}$，在一定的温度范围内 I_C 就是一个定值。用恒流法测电阻时，参考电流的误差会直接反映在测量值中。

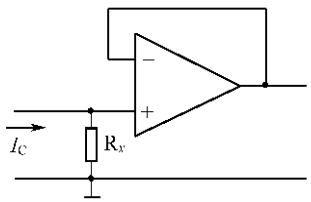


图 6.20　恒流法测电阻

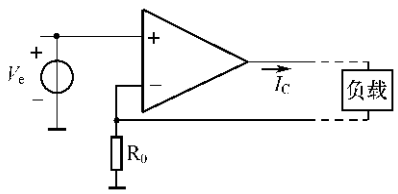


图 6.21　一种恒流源产生电路

二、恒压法测电阻

如图 6.22 电路所示，设 V_{ref} 为恒定的电压，R_0 为标准电阻，则

$$V_0 = \frac{V_{ref}R_0}{R_X + R_0}, R_X = \frac{V_{ref}R_0}{V_0} - R_0$$

用恒压法测电阻时，参考电压的误差会直接反映在测量值中。

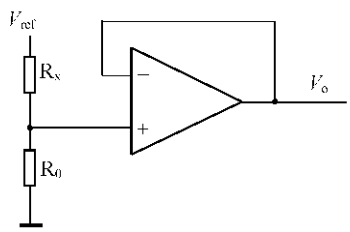


图 6.22　恒压法测电阻

三、恒阻法测电阻

如图 6.23 所示，I_C 为电流源，先让 I_C 通过 R_o 产生电压经 A/D 转换得 D_o，再让 I_C 通过要测电阻 R_{xi} 产生电压经 A/D 转换得 D_i，设 A/D 转换的系数为 K，即

$$D = KV_o \text{ 或 } V_o = \frac{1}{K}D$$

设 V_{oi} 为 I_C 流过 R_x 产生的电压，V_{oo} 为 I_C 流过 R_o 产生的电压，因 $I_C \times R_{xi} = V_{ox}$，$I_c \times R_o = V_{oo}$，故

$$\frac{V_{oi}}{R_{Xi}} = \frac{V_{oo}}{R_o} = I_C, R_{Xi} = \frac{V_{oi}}{V_{oo}}R_o = \frac{\frac{1}{K}D_i}{\frac{1}{K}D_o}R_o = \frac{D_o}{D_i}R_o$$

可见电阻的测试结果与电流源 I_C 无关，而只与标准电阻有关。此外，该方法测试结果与模拟开关的导通电阻也无关系，而且非常适用于同时检测多个电阻。

四、积分法测电阻

积分法的基本原理是用同一个直流电压、同一电容通过不同的电阻使积分值达到同一值，记录各次积分的时间值，用标准电阻的积分时间和被测电阻的积分时间解算被测电阻值。事实上积分法也是一种恒阻法。图 6.24 为积分电路，其中 R_o 是标准电阻，R_{xi} 为被测电阻，后面再加上比较器及单片机电路就可以用程序来实现求电阻功能，请参阅第 2 章相关内容。设输入电压为 V_r，比较器的参考电压为 V_h，计数器的时基为 T_c，标准电阻积

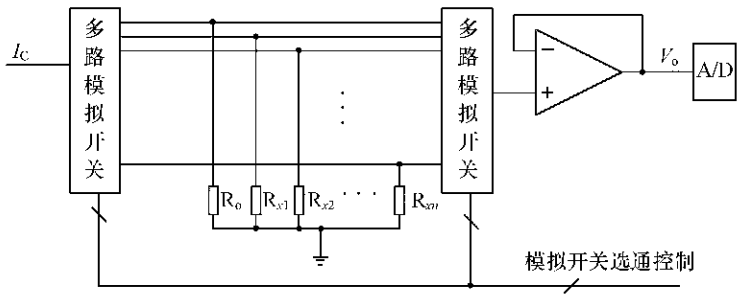


图 6.23　恒阻法测电阻

分时间为 T_1，计数值为 D_1，被测电阻积分时间为 T_2，计数值为 D_2，则

标准电阻积分：

$$V_h = \frac{1}{R_o C}\int_0^{T_1} V_r dt = \frac{V_r}{R_o C}T_1 = \frac{V_r}{R_o C}T_c D_1$$

被测电阻积分：

$$V_h = \frac{1}{R_{xi} C}\int_0^{T_2} V_r dt = \frac{V_r}{R_{xi} C}T_2 = \frac{V_r}{R_{xi} C}T_c D_2$$

因为 $V_h = V_h$，所以

$$R_{xi} = \frac{D_2}{D_1}R_o$$

可见，在不考虑模拟开关的导通电阻时，所求结果只与 R_o 有关而与电容 C、输入电压 V_r、比较器参考电压 V_h 无关。所以，积分法对于测量较大的电阻是一种较好的方法。

6.2.6　电容型信号的检测

一、积分法测电容

积分法求电容的思路和积分法求电阻的思路相同，图 6.25 为积分器的部分电路，C_o 为标准电容，C_{xi} 为待测电容，标准电容积分时间为 T_1，计数值为 D_1，被测电容积分时间为 T_2，计数值为 D_2，很容易推得

$$C_{xi} = \frac{D_2}{D_1}C_o$$

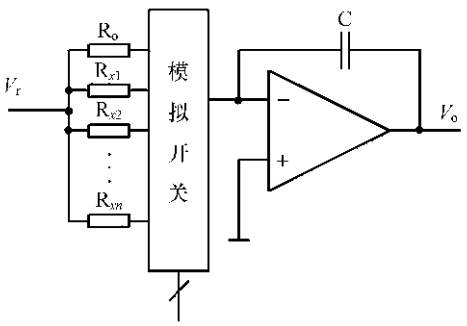


图 6.24　积分法测电阻

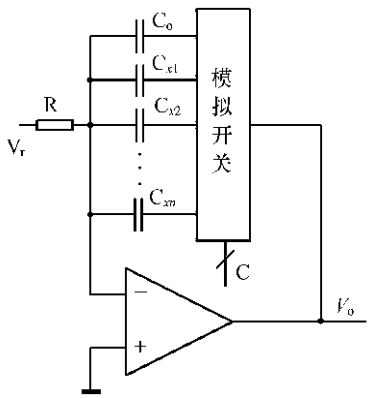


图 6.25　积分法求电容

二、相位法测电容

相位法测电容如图 6.26，先用 D/A 产生正弦波 A_o，A_o 一路通过纯电阻分压电路产生

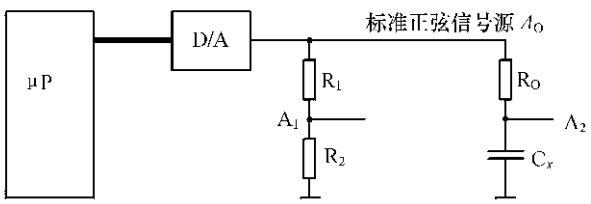


图 6.26　相位法求电容

电压信号 A_1，一路通过被测电容 C_x 与标准电阻 R_o 的分压电路产生电压 A_2，A_2 与 A_1 之间必产生相位，设 A_1 的相位为 0，A_2 与 A_1 的相位差为 φ，则有

$$A_2 = A_o \frac{\frac{1}{j\omega C_x}}{R_o + \frac{1}{j\omega C_x}} = A_o \frac{1}{1 + j\omega R_o C_x}$$

相位

$$\varphi = - \arctan \omega R_o C_x$$

由此式通过反函数求得 C_x。

三、频率法测电容

频率法测电容是将电容组成振荡电路，利用振荡频率与电容的关系，通过频率求电容的值。如图 6.27 为方案之一，用 C_x 和运放组成一个方波发生器，方波的振荡周期 $T = 2R_3 C_x \ln\left(1 + \frac{2R_1}{R_2}\right)$，振荡频率 $f = \frac{1}{T}$，可见，通过求 T 或 f 都可以求出电容 C_x 来。

6.2.7　频率及周期性信号的检测

一、频率及周期性信号的特点

频率和周期性信号如图 6.28 所示。由于频率和周期互成倒数关系，对于智能检测系

统来说，计算并不是主要考虑的问题，主要考虑测量精度要高，电路尽可能要简单。频率量和周期量是数字脉冲型信号，其幅值的大小与被测值无关，但幅值过小达不到 TTL 电平时 μP 将不能识别，幅值过大时又会损伤测量电路，所以该类信号也要有前置放大及衰减电路，以使测量仪器具有较宽的适应性。此外，被测型号也可能带有一定的干扰信号，因此加适当的低通滤波也是必需的，这部分内容这里不再叙述，可参考前面有关章节的内容。

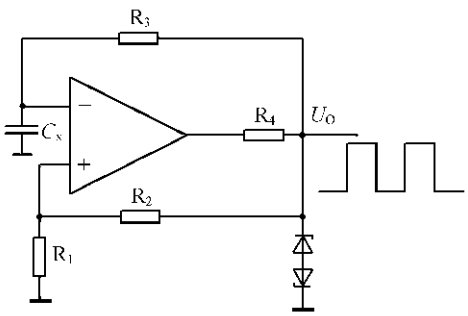


图 6.27　积分法求电容

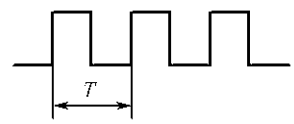


图 6.28　频率或周期性信号

二、频率测量基本电路

图 6.29 的电路适合于测量频率不十分高的频率量，将被测信号 V_f 经过放大、衰减、滤波及整形电路后变成一个标准的 TTL 信号，直接加在 μP 的计数端，用被测脉冲作为时

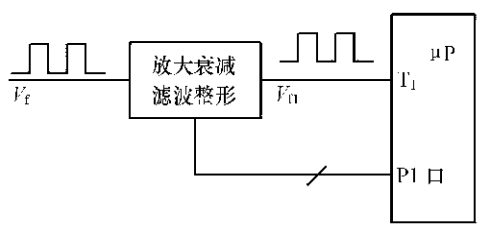


图 6.29　基本的频率测量电路

钟触发 μP 内部计数器进行计数。μP 内部另设一个定时器，在规定的时间看能计多少数，由此求得被测信号的频率。设规定时间为 T_0，计数器的计数值为 N，被测信号的频率为 f，则

$$f = \frac{N}{T_0}\ (Hz)$$

在图 6.29 中，设经转换后的 V_{f1} 的频率在 1kHz ~ 100kHz 之间，频率测试结果为变量 test _ frq，频率单位为 Hz，单片机选用 89C52，测到一次频率值就退出。程序如下：

```
#include  < AT89X52. h >

float test _ frq ;
long int T1 _ flow _ counter,T0 _ flow _ counter ;
bit data test _ end ;
```

```
//*************************************
//                    主程序
//*************************************
void main (void)
 {
//----------通用初始化----------

TH1  =  0xA0;        //T1 方式 2 作 0.1ms 秒定时,TH1 = A0H,晶振用 12MH
TL1  =  0xA0;        //TC = 12000000/12/100 (64h) = 10000Hz, 自动装载
TMOD  =  0x21;       //T0 方式 1 计数,用外部时钟

TH0  =  0;
TH1  =  0;
while (!test_end);
text_frq  =  T0_flow_counter*65536  +  TH0*256  +  TL0;
}

//----------------------------------
//Timer0 中断程序
//----------------------------------
void T0_int (void)  interrupt 1
 {
T0_flow_counter++;
}



//----------------------------------
//Timer1 中断程序
//----------------------------------
void T1_int (void)  interrupt 3
 {
T1_flow_counter++;
if (T1_flow_counter  ==  10000)        //10000*0.1ms = 1s
    {
   TR0  =  0;//stop TR0
   test_end  =  1;
   }
}
```

其中，Timer0 中断服务程序用来记录计数器的溢出的，Timer1 中断每 0.1ms 产生一次，中断 10000 次正好为 1s。

当被测信号的频率较高时，如 f > 20MHz，有可能单片机的速度不能支持计数器正常工作，此时可采用图 6.30 电路，将被测信号经过一个宽带设计的放大、衰减、滤波及整形电路后，先进入一个高速分频器（如 10 分频）后，再进入单片机计数端，选择合适的分频数可处理任意高的频率信号。

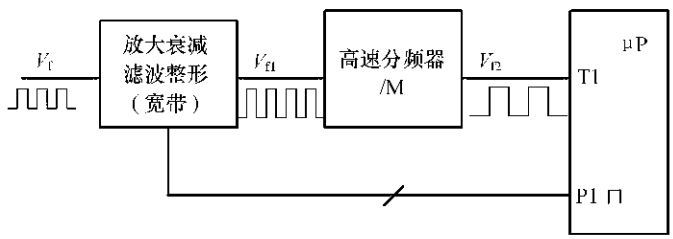


图 6.30　高频频率测量电路

三、周期测量基本电路

当被测信号频率较低时，如果还测其频率，不但刷新时间长，而且测量精度也将变低，比如 50Hz 的频率在 1s 只能计 50 个数，按 1s 的刷新一次的设置，其测试精度只有 ±1Hz。所以当被测信号的频率较低时，应该反过来测信号的周期，这样才能提高测量精度和刷新频率。

图 6.31 为测量周期的基本电路。信号 V_t 经过放大、衰减、滤波及整形电路后，变为 TTL 电平 V_{t1}，然后 V_{t1} 再经过 2 分频变为 50% 占空比的对称方波 V_{t2}，V_{t2} 接入 μP 的中断口如 INT1 时，V_{t2} 的正脉冲宽度正好是被测信号的周期值，μP 可用 INT1 上升沿启动内部计数器开始计数，再用 INT1 下降沿结束计数器，由此计算被测信号周期。设内部计数器时钟周期为 T_c，计数值为 N，则被测信号的周期 $T = NT_c$。如果要得到被测信号的频率求其倒数即可。

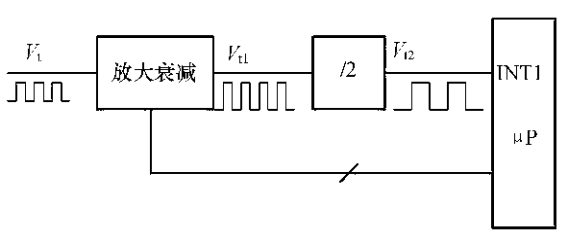


图 6.31　较高频频率测量电路

可见，无论要得到频率值还是周期值，都可以遵循高频则测频率、低频则测周期的原则。这样做的结果是不管被测信号在什么频段内都可以达到要求的测量精度，这种方法又称为等精度测量频率法或等精度测量周期法。

四、通用频率计（计数器）的一般电路

图 6.32 为一个通用频率计或计数器的基本电路示意图。该方案可测频率也可测周

期，也可做计数器，通过键盘选择，显示用数码管显示。可预置数高速分频器由 I/O1 口控制 程控放大、衰减电路由 I/O2 口控制 键盘及显示由 μP 的 I/O3 控制 操作指令由键盘人工给定。输入信号经放大、衰减、滤波及整形电路后，先通过可预置数的高速分频器，再进入 μP 的 T1 和 INT1 的并联口，在检测时 μP 用试探法初步判断信号频率的高低，然后决定是用测频率法还是用测周期法求出频率或周期。

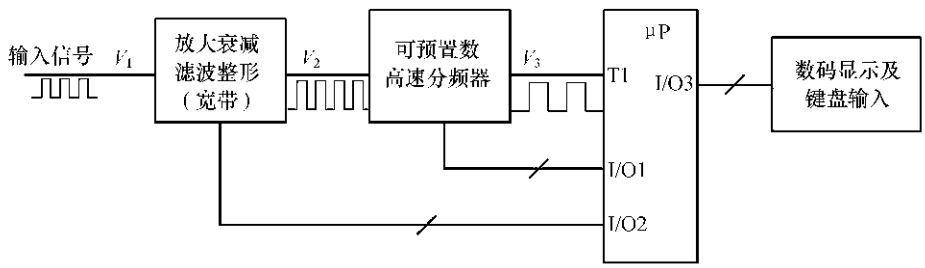


图 6.32　通用频率计（计数器）的一般电路

6.2.8　数字协议型信号的检测

所谓数字协议型信号，是指建立在标准或者非标准协议上的数字信号，常见的标准数字协议型信号在第 4 章已作了较多的介绍，如 I^2C 型、RS - 232 型、RS - 422 型、RS - 485 型、USB 型、CAN 型等。非校准型的数字协议一般用在检测系统内部，外部接口往往采用标准型。由于数字传感器会逐步取代模拟传感器，而数字传感器的普及必然会使模拟信号消化在传感器内部，因此数字协议型信号将会是“未来型”信号。

第 7 章　智能化检测系统设计举例

7.1　设计智能化检测系统的一般步骤

对于一个设计者来讲，面对的是一个个的课题任务，每当遇到一个检测系统课题时，应采取以下步骤来完成。

第 1 步，全面了解课题的内容，特别要搞清课题要解决的问题，同时也要到用户方调研，最后做出详细的设计任务书。

第 2 步，根据设计任务书制定设计方案。如果课题组人数超过一人时，最好让每个人都独立拿出意见或方案，最后经比较、择优、综合制定方案。在选取方案时要从总体的先进性、可靠性、成本、制作周期、可维护性等各方面考虑。尽可能选取已掌握的熟练的技术，在有较高把握的前提下也可选用创新性的新技术，这不但可提高系统的先进性，也可为系统的升级储备技术。

第 3 步，根据技术方案，草拟系统的详细使用说明书，并提交用户审核，经过用户修改认可。

第 4 步，设计详细的电子硬件线路原理图、机械及其它各部分的图纸以及软件详细流程图。

第 5 步，按照原理图对系统中的某些关键部分做局部的试验以验证其可行性，并测试其能否达到技术要求，这里包括硬件、软件及其它部分。全部试验做完后确定图纸。

第 6 步，设计电路 PCB 图，制作线路板，完成机械结构等的加工、具体软件的全面设计、元器件的采购等工作。

第 7 步，组装调试，如发现有原来未曾料到的问题，应及时补救并修改设计资料。

第 8 步，全面检验，在此基础上测试其性能指标，在满足技术要求后提交用户使用。整理全部资料，尤其要把系统的优点、缺点都总结在案。

在以下所举的例子中，因篇幅的限制只写出设计思路和一些关键部分的细节。

7.2　智能温度传感器的设计

7.2.1　智能传感器的概念

传感器是指直接将自然界的变量（如物理量、化学量等）转化为测控系统可识别的量（如电压、电流等）的装置。通常传感器由敏感元件和转换元件组成。其中敏感元件是指能直接感受被测量的部分 转换元件是指传感器中能将敏感元件输出量转换为适于传输和测量的电信号部分。

智能传感器又称为数字传感器，系指内含微控制器并用微控制器管理其主要功能的一类传感器。随着测控系统自动化、智能化的发展，要求传感器准确度高、可靠性高、稳定性好，而且具备一定的数据处理能力，并能够自检、自校、自补偿。传统的传感器已不能满

足这样的要求，需要利用计算机技术与传感器技术相结合，弥补其性能的不足，由此而产生了功能强大的智能传感器。在信息技术高度发达的今天，传感器的智能化和多功能化将成为传感器发展的重要方向。和传统的传感器相比，智能传感器具有以下优点。

⑴ 具有逻辑判断、统计处理功能。可对检测数据进行分析、统计和修正，还可进行线性、非线性、温度、噪声、响应时间、交叉感应以及缓慢漂移等的误差补偿，提高了测量准确度。

⑵ 具有自诊断、自校准功能。可在接通电源时进行开机自检，也可在工作中进行运行自检，并可实时自行诊断测试以确定哪一组件有故障，提高了工作可靠性。

⑶ 具有自适应、自调整功能。可根据待测物理量的数值大小及变化情况，自动选择检测量程和测量方式，提高了检测适用性。

⑷ 具有组态功能。可实现多传感器、多参数的复合测量，扩大了检测与使用范围。

⑸ 具有记忆、存储功能。可进行检测数据的随时存取，加快了信息的处理速度。

⑹ 具有数据通信功能。智能化传感器具有数据通信接口，能与计算机直接联机，相互交换信息，提高了信息处理的质量。

⑺ 具有近地控制功能。可进行一些与检测对象有关的控制，提高测试质量，展宽测试范围。

并不是所有的智能传感器都具有以上功能，在设计具体的智能传感器时往往是根据实际需要来确定其功能。下面举一个智能温度传感器设计的例子。

7.2.2　数字温度传感器

温度是与人类紧密相关的物理量，在许多场合都必须对温度进行测量。随着人类的进步，温度测量的方法也在不断发展。在步入数字时代的今天，温度传感器也在从模拟温度传感器转向智能温度传感器。一般的智能温度传感器是由微型计算机实现其基本功能并且具有协议型输出的传感器。在这方面美国 DALLAS 公司是走在前面的，该公司将感温元件和微控制器封装在一个芯片中，具有 IC 口或单总线输出口，可以测量温度范围为 - 55　~125　，分辨力为 0.5　。在精度要求不高或测量范围不太宽的场合，可以很方便地用于各种智能检测设备中。但是数字温度传感器的发展并没有就此终结，在测量要求精度更高的场合（例如：分辨力要求为 0.01　）或在测量范围要求更宽的场合（例如：测量范围为 - 80　~ +300　），就必须设计符合要求的智能温度传感器。

智能温度传感器的一般结构如图 7.1 所示，它分为温度敏感元件和信号处理器件两大部分。由于信号处理器件在工作时或多或少会发热，为了不影响温度敏感元件对被测目标的感应精度，应将这两部分保持一定的距离。信号处理器件的工作温度范围在 - 55　~ 125　之间，当被测温度超过这个范围时，应将这两部分保持相当的距离或采取一定的隔离措施。

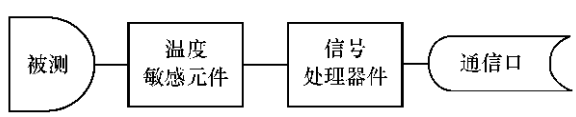


图 7.1　数字温度传感器的结构

7.2.3 电路方案

常见的温度敏感器件有热电偶、热敏二极管、铂电阻、热敏电阻等。这里以铂电阻和热敏电阻为例来完成智能温度传感器的设计。铂电阻和热敏电阻都是以电阻值的变化来感受温度的，前者阻值变化小而线性度好，成本也较高；后者阻值变化大，但线性度差，成本也较低。与力、加速度等其它物理量相比，温度信号的变化比较缓慢，对温度信号的处理不需过高的 CPU 速度，可以选择速度一般，成本较低的微控制器来担任信号处理器件。图 7.2 为一种电阻类智能温度传感器的硬件原理图。电路由 U1 ~ U4 等 4 个集成电路与电阻电容等分立元件组成。其中 U1 为模拟开关 CD4051；U2 为运算放大器 TL062；U3 为单片机 LPC922；U4 为单 5V 供电的 TTL 电平与 RS232C 电平转换器件 MAX232，该器件自带 5V 到 ±12V DC - DC 转换电路。为了尽可能减少本装置体积和成本，U2 的正负电源直接取 U4 的 V + 和 V - 的输出。而 U1 的 +5V、- 5V 电源由 U4 的输出 V +、V - 经稳压管 D1、D2 降压后供给。

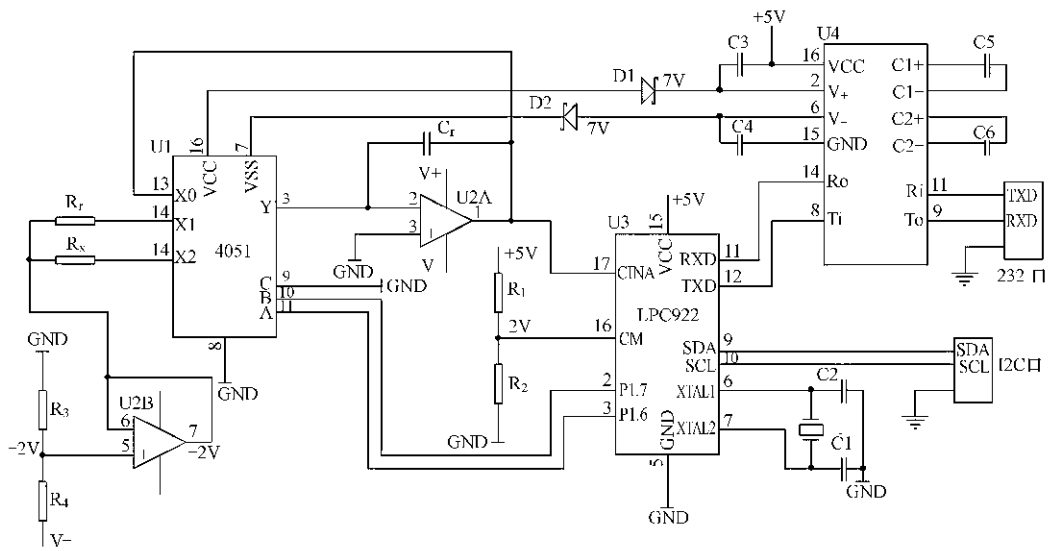


图 7.2 电路基本原理图

电阻测试采用被测电阻积分计时与标准电阻积分计时相对照的解算原理，以 - 2V 作为积分器的输入，步骤如下。

(1) 送模拟开关控制值 A = 0 及 B = 0，选 X0 - Y 通路对电容 C 放电持续 1ms，使 $C_{in}A = 0$。

(2) 送模拟开关控制值 A = 1 及 B = 0，选 X1 - Y 通路启动单片机内部 16 位定时器，- 2V 通过标准电阻 R_r 积分使 $C_{in}A$ 电位上升，$C_{in}A = -\frac{1}{R_rC}\int_0^{T_1}(-2V)\,dt = \frac{2T_1}{R_rC}$，当 $C_{in}A$ 上升到 CM 给定值 2V 时产生中断，单片机定时器停止计数，记此时的计数值为 D_1，积分用时间为 T_1。

(3) 重复步骤 (1) 使 $C_{in}A = 0$。

⑷ 送模拟开关控制值 A = 0 及 B = 1，选通 X2 - Y 通路启动单片机内部 16 位定时器。 - 2V 通过被测电阻 R_x 积分使 $C_{in}A$ 电位上升，$C_{in}A = -\frac{1}{R_xC}\int_0^{T_2}(-2V)\,dt = \frac{2T_2}{R_xC}$，当 $C_{in}A$ 上升到 CM 的给定值 2V 时产生中断，单片机定时器停止计数，记此时的计数值为 D_2，积分用时间为 T_2。

⑸ 计算 因为 $\frac{2T_1}{R_rC} = \frac{2T_2}{R_xC}$，即 $\frac{T_1}{R_r} = \frac{T_2}{R_x}$，设计数值和时间的关系为 T = KD。T 为计数时间，K 为转换系数，D 为计数值，则 $T_1 = KD_1$，$T_2 = KD_2$。代入式 $\frac{T_1}{R_r} = \frac{T_2}{R_x}$ 可求得 $R_x = \frac{D_2 \times R_r}{D_1}$。

可见被测电阻 R_x 只与两次积分的数字 D_1、D_2 及参考电阻 R_r 有关，而与积分输入电压及电容值无关，这一点正是我们所希望的。而 D_1、D_2 几乎是在同一时间完成的，这样求出的电阻 R_x 具有很高的准确性。

7.2.4　软件设计

当求得电阻之后，温度的求法仅仅是一个计算的问题。对于铂电阻来说一般用一个二次多项式就可以计算出较高精度温度值，计算公式为 $R = A(0) + A(1)T + A(2)T^2$。对热敏电阻来说，其温度变化范围大，线性度较差，曲线拟合也较复杂。如某厂生产的热敏电阻需用 5 个系数来逼近，计算公式为

$$T = A(0) + A(1)\ln R + A(2)(\ln R)^2 + \cdots + A(4)(\ln R)^4 + A(5)(\ln R)^5 - 273.15$$

其中对数的计算可采用级数的展开，用嵌套叠加原理来计算。

在温度计算好后，用浮点数的形式保存在寄存器中，当与之连接的主机有命令索取温度值时及时送走，传感器在平常处于不断的更新温度值的状态。单片机共设两个中断：比较器中断和通信中断。为了保证积分计数值的准确，比较器中断设为最高。整个系统的流程图见图 7.3，从左到右分别为主程序流程图、通信中断流程图、比较器中断流程图。

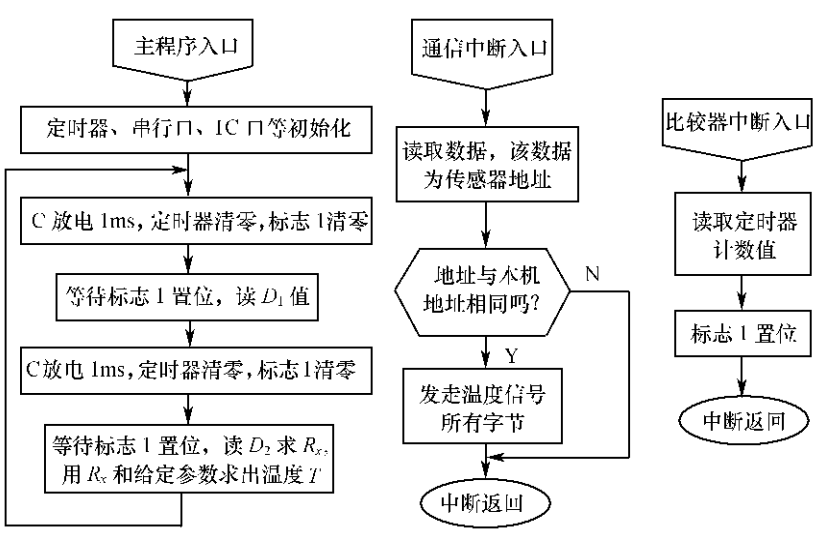


图 7.3　软件流程图

7.3 电子皮带秤

7.3.1 皮带运输机简介

皮带运输机又称带式输送机，是一种连续运输机械，也是一种通用机械。皮带运输机被广泛应用在港口、电厂、钢铁企业、水泥、粮食以及轻工业的生产线。它可以运送散状物料，也可以运送成件物品。工作过程中噪声较小，结构简单。皮带运输机可用于水平或倾斜运输。

皮带运输机由皮带、机架、驱动滚筒、改向滚筒、承载托辊、回程托辊、张紧装置、清扫器等零部件组成。在大型港口或大型冶金企业，皮带运输机得到广泛的应用。其总长度可达十几千米。皮带运输机的输送能力可以为几百千克/小时到万吨/小时，皮带的宽度可以从 100mm 到 2600mm。图 7.4 是皮带机运输机的一种。

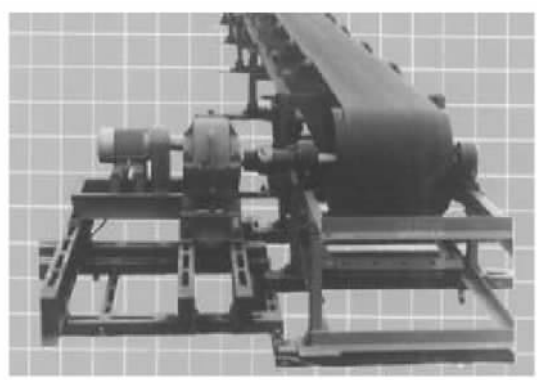


图 7.4 一种电子皮带机的图片

7.3.2 电子皮带秤基本原理

电子皮带秤是用来测量通过皮带运输机的物料的重量的。其基本原理是，连续测量通过皮带单位距离的物料重量，同时相应测量皮带移动了多少个单位距离，在一段时间内将每个单位距离的重量累计起来就是这段时间皮带所运输的货物的总量。

在测量通过皮带单位距离的货物重量时，需要该段距离的皮带连同货物一起悬浮在若干个传感器上（一般为 4 个传感器），为此必须有一个秤架将传感器和皮带连接起来。秤体的设计应该满足以下要求：

⑴ 秤体需分为固定秤架和活动秤架两部分，固定秤架和地面（或皮带支架）连接，而活动秤架通过自身的拖棍和皮带接触；

⑵ 活动秤体和固定秤体之间用若干个传感器柔性连接，各传感器的受力必须均匀平衡；

⑶ 活动秤体要用不影响垂直受力的特殊方法固定，以保证在皮带运行时活动秤体不会前后左右晃动；

⑷ 秤体要有不变形、抗震动、防锈等功能，以保证皮带秤的测量精度和寿命。此外，在活动秤架上还需安装一个速度传感器用来测量皮带移动的速度。

图 7.5 为一种电子皮带秤的秤架，图 7.6 为常见的秤体结构示意图。

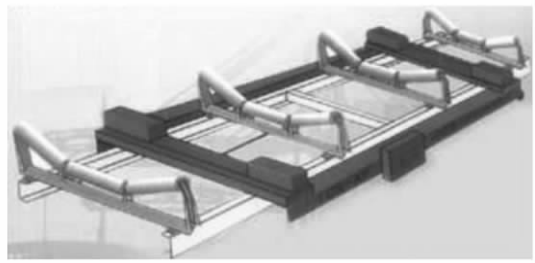


图 7.5　一种电子皮带秤架的图片

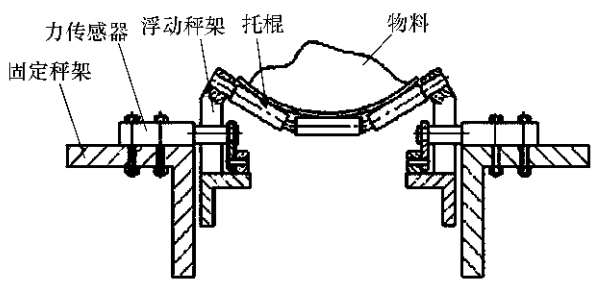


图 7.6　电子皮带秤的秤体结构

7.3.3　一种电子皮带秤的设计方案

图 7.7 为一种电子皮带秤的硬件方案，就系统组成的主要部分叙述如下。

（一）主控制器

主控制器用单片机 8031 系统完成。8031 系统包括程序存储器 EPROM、数据存储器 RAM、译码器、驱动器以及锁存器等电路，其中 RAM 要求带有后备电池及 RAM 数据掉电保护电路，要求系统在掉电后 RAM 能靠后备电池保存数据达 1 个月以上。

（二）货物重量采集

代表皮带秤货物的重量由 FS1 ~ FS4 等 4 个力传感器并联输出，进入前置放大器 F1，F1 除放大外还具有调零和调满度功能，分别由调零电位计和调满度电位计来完成。经放大后的信号进入 S/H 电路 F1 及 A/D 电路 F4。8031 系统用 P1.0、P1.2 和 Data Bus 完成数据采集，此部分是一个中速度的数据采集，采样保持器选用 LF398，A/D 转换选用 AD574，具体电路可参考第 2 章有关内容。

（三）皮带速度信号采集

在图 7.8 所示的电路中，SS 为磁电式速度传感器，其输出脉冲的频率代表皮带速度的高低，输出信号的幅值与皮带的速度成正比。该传感器不用电源，当皮带运行较慢时速度传感器输出幅值较低，需经过放大后才能使用，为此设放大器 F2 先对其进行放大，然

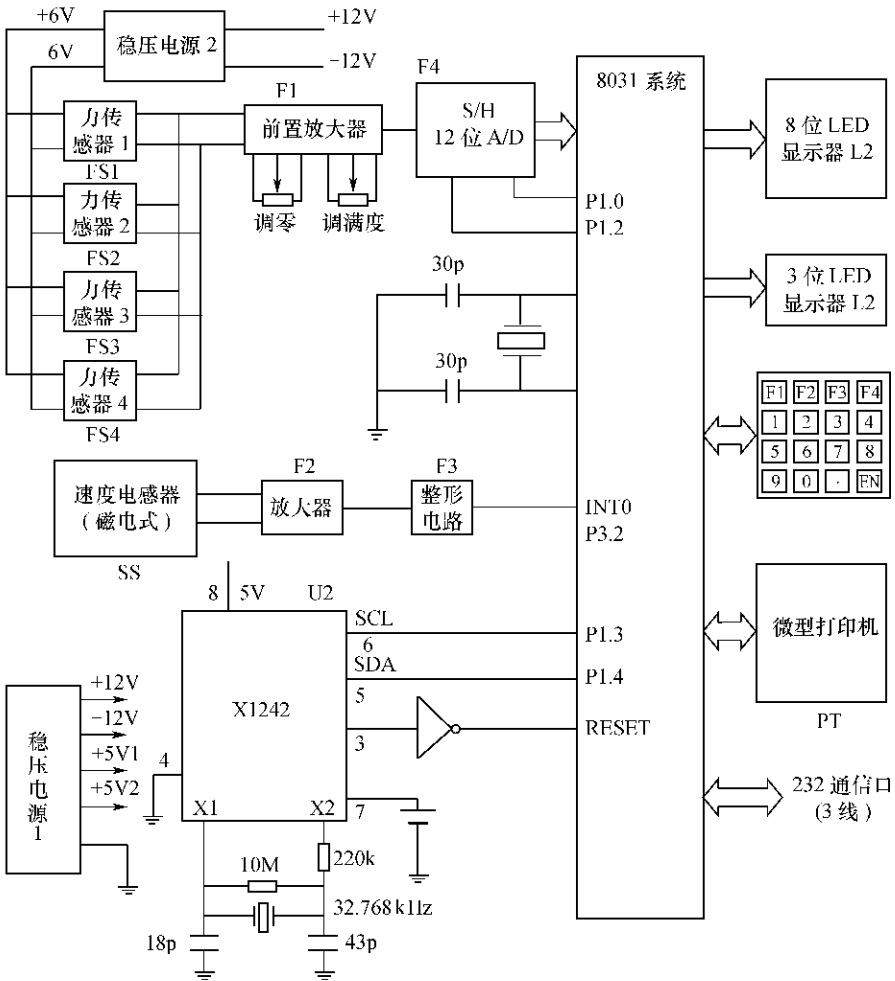


图 7.7　一种电子皮带秤的硬件方案

后再用整形电路 F3 将其变为 TTL 数字电平。A1、R_1、R_2 组成同向放大器，增益为 100。皮带低速运行时 A1 输出正弦信号，皮带高速运行时 A1 输出对称削顶的正弦信号；A2、R_3、R_4 组成比较器，将 A1 的输出信号变为对称方波信号，并利用其迟滞特性滤掉信号中边沿抖动；R_5 和 D1 最终将信号变为 3.2V 的正向脉冲信号，进入系统的外中断入口 INT0，8031 通过一段中断服务程序来获得皮带运行的速度信号。

（四）实时时钟及 E^2PROM 存储器等电路

该部分电路由 X1242 芯片构成，它与主机的通信用 I^2C 口来实现，具有以下功能。

⑴ 实时时钟功能，自动产生包括秒、分、时、日、月、年、星期、自动闰月处理并可以再设置的功能。

⑵ 有 2KB 的 E^2PROM，用此存放皮带秤系统的设置参数和计量累计值，包括日累计、月累计、年累计、总累计，而班累计和瞬时累计值存放在 RAM。由于 RAM 可保证一个

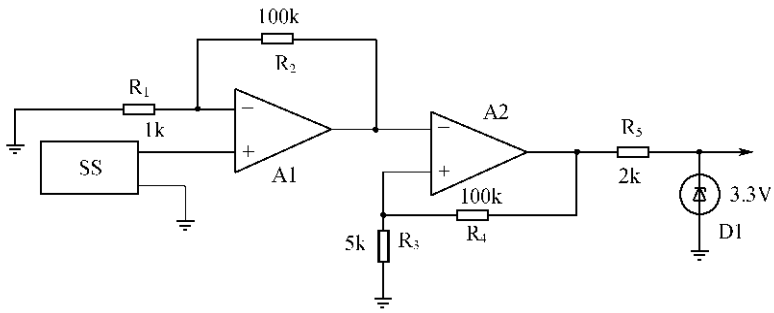


图 7.8　速度信号处理电路

月以上的断电数据保护，而一般的使用单位停电时间一般不超过一个月。因此用 X1242 来备份累计数据不仅保证数据不会丢失，而且两套数据可相互校验，从而保证数据的可靠性。

(3) 有看门狗及自动掉电复位功能。X1242 上电后便可产生复位信号，工作时发现 V_{CC} 小于某个值也能产生复位信号。而且 X1242 还有一个可调整的看门狗定时器，当系统软件死机后，将不能周期性地给这个定时器发清零信号，定时器就会溢出而产生复位信号，从而保证系统不会死机，这就是所谓的看门狗复位。

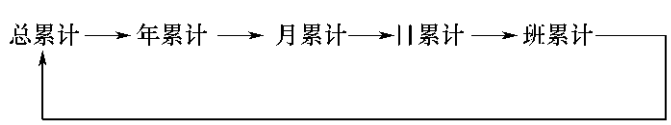
(五) 其它电路

在本方案中，8031 系统还具有 8 位数码管显示电路、3 位数码管显示电路、4 × 4 = 16 键键盘输入电路、微型打印机驱动电路、RS - 232 接口电路等电路 系统电源包括 +5V1、+5V2、±12V、±6V，其中 +5V1 用于系统的逻辑电路供电，+5V2 用于 LED 数码管供电，±12V 用于模拟放大器供电，±6V 由 ±12V 二次稳压产生专用于力传感器供电。8031 的基本输入输出口除 P1.0 ~ P1.4 和 P3.2 作为输入信号控制外，其余输入输出口有的直接输入输出，有的经过扩展后输入输出，用来完成显示、打印、键盘扫描、通信等功能。

(六) 键盘功能说明

键盘有 F1 ~ F4 等 4 个功能键，0 ~ 9 等 10 个数字键，另有一个小数点键和 Enter 键，共 16 个键。功能键定义如下。

F1——选择显示功能，复位后显示总累计，按一下改变一次显示方式，顺序为



F2——设置参数，按 F2 后要求输入密码，密码认可后，显示器 L2 显示参数号，显示器 L1 显示参数值，然后用数字键进行修改，修改后用 Enter 确认。

F3——进入标定，标定包括空皮带的零值设置和实物过后的总重量的确认等。

F4——打印一次数据。

7.3.4 软件流程

图 7.9 为主程序流程图，图 7.10 为键盘中断流程图，图 7.11 为外中断 INT0 的中断流程图，图 7.12 为定时器 Timer0 的中断流程图。中断级别为 INT0 最高，Timir0 次之，键盘中断再次之。

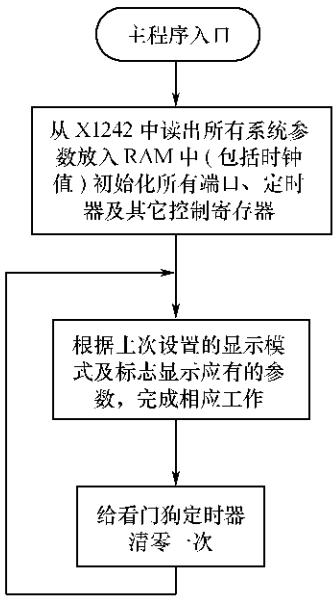


图 7.9　主程序流程

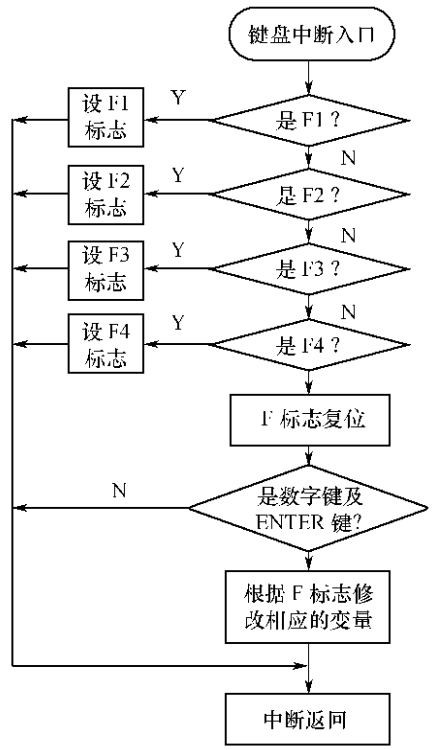


图 7.10　键盘中断流程

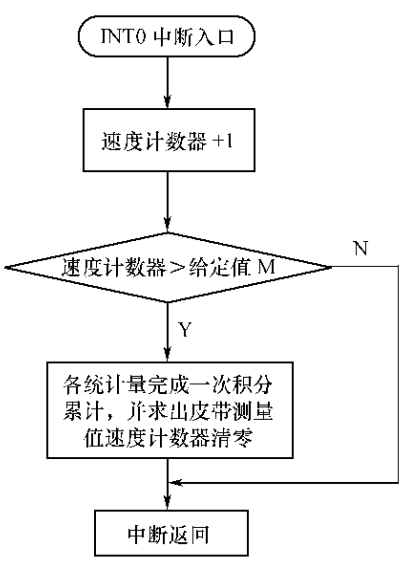
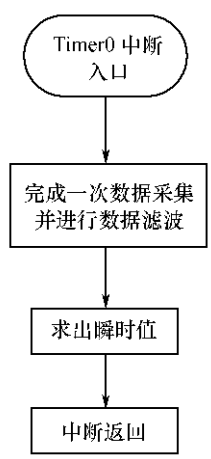


图 7.11　INTO 中断流程



7.12　Timer0 **中断流程**

7.4 智能热表的设计

7.4.1 热表的功能

暖气供热部门向住户收费的传统方法是按照住户的居住面积和供暖时间来计算，随着时代的进步和能源管理市场的进一步规范，这种收取暖气费的方法暴露出的问题日趋明显，如供热不均、能源浪费、收费困难等。于是许多国家提出民用建筑安装热表实行分户计量收费，近几年在一些国家已经开始实施，在我国部分城市也开始了试点工作。热表也称热量计，是衡量住户使用热量的依据，它和电度表一样必须准确可靠才能达到预期的目的。但热表的设计难度却比电度表大的多，因为电度表是计量用电量的，停电时不需计量，可直接使用电网电源，而供热系统在停电时也可运行，不能完全依赖电网作为热表的工作电源。为了便于安装、减小体积、降低成本，热表大都采取了干电池供电方式。

要适应全面数字化的时代，热表应具有数字显示和红外抄表功能。为避免住户通过热表窃热的可能性，设置铅封等防止窃热的功能也成为热表设计的主要方面。这样热表的生产成本不会很低，对住户来说应属于耐用品，加之为了减少安装的次数，要求干电池有效工作的时间越长越好，因而必需用特殊的方法使热表按超低功耗运行才能达到目的。我们为此做过较多的理论探讨和大量实验，获得了较理想的效果。

7.4.2 总体方案

为了便于安装，热表的外形结构和体积往往很接近于自来水表，大多数公司几乎不约而同选用了 CR123A - 3V 相机电池作为供电电池。图 7.13 为智能化热表的总体方案，系统由微控制器、流量测量、温度测量、液晶显示、红外接口、供电部分、信号发生器以及显示选择等部分组成。

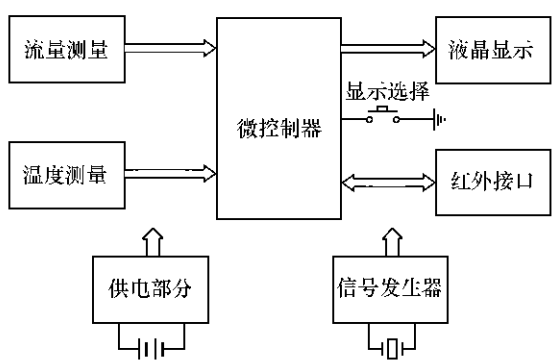


图 7.13 智能热表的总体方案

微控制器是热表的核心，从满足功能要求、低功耗要求和成本多方面考虑，我们选用了 TI 公司的 MSP430X325（以下简称微控制器），该控制器具有 84 段液晶显示驱动、14 位 A/D、512B RAM、16KKB 程序存储器，其它定时器中断等都可满足要求，开发时可用 EPROM 型，厂家还支持 OTP 型及 ROM 型，批量生产时可用之。430 系列微控制器工作电

压为 1.8V ~ 3V，非常适合于干电池供电，其最大的特点是极小的工作电流，在待机方式下工作电流可以小于 1μA，而且待机期间端口的工作状态不会改变并可随时唤醒。

流量测量完成的是对流过热表的液体的质量的计量，这里的测量对象是流体的速度，由流体速度、面积、密度和温度可确定流体的质量，它是计算热量的参数之一。

温度测量其实是测量用热区（如房间）暖气入口温度和出口温度，入口温度可以证明供暖设施的好坏，出入口温度之差和流体质量用来计算用热区所消耗的热量。

液晶显示是用来显示供暖总热量、供暖总流量、供暖总时间、流速、出入口温度、日期、时间、工作状态、故障代码等内容的。

红外接口是供管理人员抄表和专业人员设置热表用的。

信号发生器产生 X1、X2 和 X3 等 3 种特殊的信号（见下文说明）供热表自身使用，其电路由低频晶振和 74HC 系列门电路组成，工作电流极小。

7.4.3　流量信号的获取

热表分为上下隔离的两层，下层中安装一个灵敏的塑料水轮，只要流体流动，水轮就转动，其转速和流速成正比。水轮的上端浇注有两个成 180°的小磁块，水轮转动时磁块做圆周运动，在热表的上层的底部有磁敏元件用来感受磁块的运动，由此求得水轮的转速。其基本电路如图 7.14 所示。

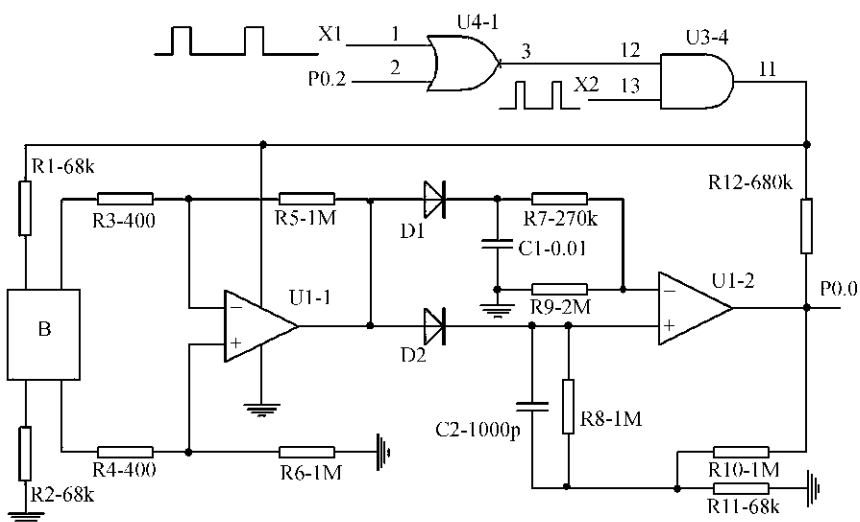


图 7.14　流量信号的获取电路

按照民用热表的要求，热表的流量应在（0.05 ~ 5）m^3/h 之间，经理论计算和实验，水轮的转速应在（15 ~ 1500）r/min 之间。由于水轮每转一次，磁敏元件可产生两个脉冲，所以实际的被测流量频率是 0.5Hz ~ 50Hz。为了达到低功耗的目的，我们采取以下措施：①磁敏元件选用线性霍尔器件以微安级电流供电；②信号的放大和整形用 TI 公司的 TLV2702 芯片，该芯片由一个运放和一个比较器组成，工作电流只有 1.4μA，带宽也可满足要求；③整个测流量电路用调制脉冲供电，而且要做到在停止供暖期间也停止该电路供

电。

X2 为给测流量电路供电的调制脉冲信号，考虑到输入信号的最高频率为 50Hz，调制频率取 50Hz × 10 = 500Hz。为了尽可能降低功耗，脉冲占空比选 25%；X1 为循检脉冲，按照输入信号的最低频率为 0.5Hz，我们取 X1 的周期为 4s，占空比也为 25%，其正脉冲宽度为 1s，在这 1s 期间即使水轮工作在最低频率也有机会捕捉到一个脉冲。流量的检测过程为：不管有无电源，因比较器 U1 - 2 的 V - 电位高于 V + 电位，总是输出低电平使 P0.0 = 0，假定 P0.2 = 0，或门 U4 - 1 输出为循检脉冲，与门 U3 - 4 输出端提供 1s 宽的调制电源，此时如水轮不转，则 P0.0 = 0，如水轮转动，则霍尔器件不断有信号变化，通过 R3、R4、R5、R6 和运放 U1 - 1 组成的差分放大后，一路经 D1 和 C1 保持峰值于比较器U1 - 2反相端，该信号基本不变接近于静态，另一路直接到比较器正相端随输入信号变化，从而使比较器输出随输入变化的调制脉冲，再经过 D2 和 C2 检波变成与输入同频的脉冲进入 P0.0 产生中断，该中断唤醒处于休眠的微控制器，然后微控制器置 P0.2 = 1，这时测流量电路的供电变为连续的 X1，微控制器以不断被唤醒的方式开始正式的流量测量，但一经发现流量低于 $0.02m^3/h$ 时又会置 P0.2 = 0。

现在计算一下此部分电路的平均工作电流。设 I_1 为停止供暖期间的平均工作电流，I_2 为全速供暖期间的平均工作电流。

$$I_1 = \left(I_{U1} + \frac{3V}{R_1 + R_2} + \frac{3V}{R_{12}}\right) \div 4 \div 4 = 1.4\mu A$$

$$I_2 = \left(I_{U1} + \frac{3V}{R_1 + R_2} + \frac{3V}{R_{12}}\right) \div 4 = 5.7\mu A$$

7.4.4 温度信号的获取

对于自带有 14 位 A/D 的微控制器来说，A/D 转换不是问题，关键是用最少的电能来达到目的。供暖一般都以软化水作为热介质，温度波动性较小，对温度的采样频率可以很低（我们选取 32s 一次的采样频率）。温度传感器一般采用厂方配对的铂电阻，由于铂电阻的阻值不可能做得很大，选用 1kΩ 阻值的 PT1000 算是较大的了，不管用电流方式或电压方式都会流过铂电阻毫安级的电流，这对热表来讲是不能容忍的。为此我们采取如图 7.15 所示的方案。A0、A1 和 A2 为微控制器的模拟量输入端口，用 P0.1 控制与门 U3 - 2

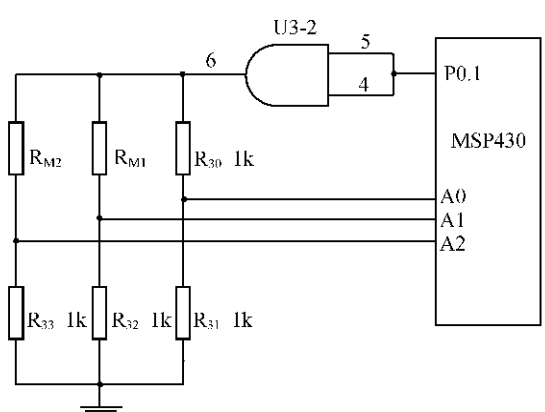


图 7.15 温度信号的获取电路

的输出作为电压输出，电阻 $R_{30} \sim R_{33}$ 均为标准电阻，精度高于 0.50 ，每隔 32sP0.1 输出一次高电平 U_h，在 A0、A1 和 A2 端分别产生电压 U_{A0}、U_{A1} 和 U_{A2}，微控制器对此 3 路信号 A/D 转换的结果分别为 D_{A0}、D_{A1} 和 D_{A2}。设 K 为 A/D 转换系数，则有

$$D_{A0} = K \cdot U_{A0} = 0.5 \cdot K \cdot U_h$$

$$D_{A1} = K \cdot U_{A1} = K \cdot U_h \cdot R_{32} / (R_{32} + R_{M1})$$

由此导出

$$R_{M1} = R_{32} (2D_{A0} + D_{A1}) / D_{A1}$$

同理

$$R_{M2} = R_{32} (2D_{A0} + D_{A2}) / D_{A2}$$

可见所求的代表温度的电阻值与 A/D 转换系数 K、供电电压 U_h 没有关系，这正是我们所希望的。A/D 转换在几毫秒内即可完成，转换完毕立即令 P0.1 = 0，因而整个测温电路的耗电是非常小的。

设此部分的平均工作电流为 I_3，考虑到电路的建起时间，我们给 3 路 A/D 足够的时间 20ms：

$$I_3 = \left(\frac{3V}{R_{30} + R_{31}} + \frac{3V}{R_{M1} + R_{32}} + \frac{3V}{R_{M2} + R_{33}} \right) \times \frac{0.02}{32} = 2.8\mu A$$

7.4.5　红外接口的实现

红外接口是供管理人员抄表和专业人员设置热表用的，它分接收和发送两个环节。当系统处于休眠状态时，红外操作请求必需能将其唤醒，系统在发送红外信号时消耗电能是必需的，为了减少功耗，在不需发送红外信号时应将发送部分彻底关断，而在休眠时接收部分用最少的电能来达到目的。图 7.16 为该部分电路。

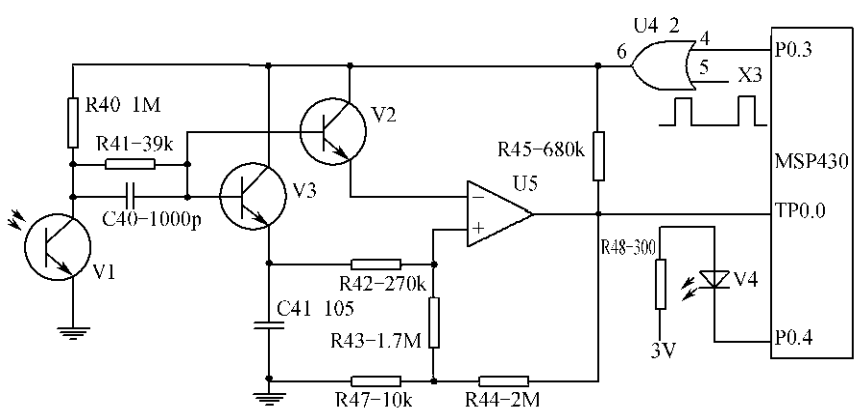


图 7.16　红外通信接口电路

红外接口通信时热表为从机，要不断地扫描有无通信命令信号。扫描用间断地给接收电路供电的方法来实现，此部分用到了信号发生器产生的信号 X3，X3 与 X2 同频，但占空比只有 10% 即 200ms，在休眠期间微控制器令 P0.3 = 0，或门 U4 - 2 每隔 4s 输出一个

200ms 的脉冲为红外接收器提供电源。V1 为红外接收管无信号时截止，三极管 V3 和电容 C41 保持 V1 输出的峰值并分压后送到比较器 U5 的 V_+ 端，V2 将变化的 V1 输出送给 U5 的 V_- 端。没有通信命令时 U5 输出低电平，一旦有通信命令 U5 就会输出脉冲信号，其第 1 个正沿首先唤醒微控制器并置 P0.3 = 1，使接收电路得到连续的电源，等到通信完成后微控制器恢复 P0.3 = 0 继续回到休眠状态。这里的比较器也选用 TLV2702，整个接收电路的平均工作电流可以非常小。红外发送用 P0.4 变低来实现，不用时令 P0.4 为高阻态可进一步降低功耗。而抄表大约一个月一次，红外发送电流可忽略不计。

设此部分的平均工作电流为

$$I_4 = ((3V - 0.7V) / (R40 + R41 + R42 + R43 + R47) + 3V/R45 + I_{U5}) \times 0.2/4 = 0.51\mu A$$

7.4.6 软件流程

从以上设计可以看出，微控制器平常处于休眠状态以极低的电流维持，当有中断时才工作很短的时间。为了能及时刷新显示、自动记录时间和识别按键，电路设计成 X3 信号和按键也可唤醒微控制器，整个系统的程序流程如图 7.17 所示。

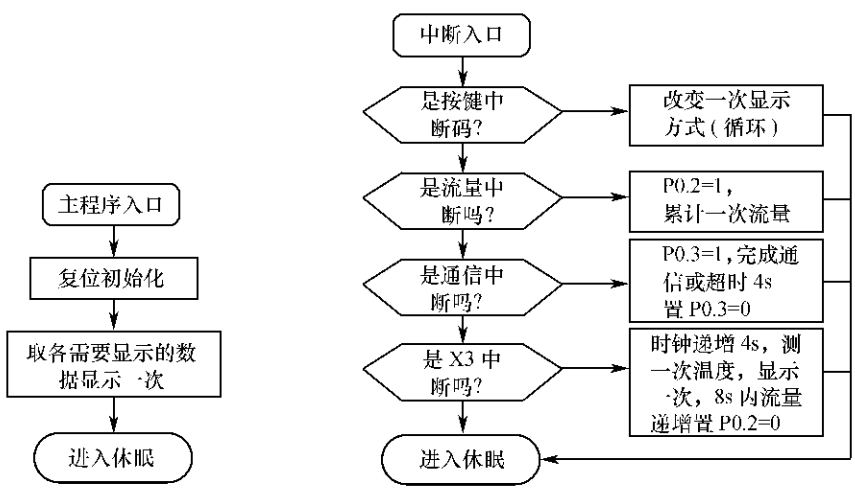


图 7.17 热表工作流程图

7.4.7 总结

除了流量测量、温度测量和红外接口以外，信号发生器用常规设计方法工作电流也可以控制在 1μA 以下，微控制器包括液晶显示在内其它部分的工作电流在休眠时可小于 5μA。根据以上分析，整个系统在停止供暖期间总电流大约是 1.4 + 2.8 + 0.51 + 1 + 5 = 10.71μA。由于微控制器唤醒时将有约 0.3mA 的工作电流，尽管微控制器工作时间比休眠时间少，在供暖期间系统的工作电流也会有增加，增加量与流量成正比。一般情况下供暖流量都比较小，而且供暖时间大都在 6 个月之内，因此按照本方案设计的热表，其电池的有效工作时间应在 6 年以上。

7.5　压力开关智能化动态测试系统设计

7.5.1　问题的提出

压力开关是空调系统的控制部件，它具有控制压缩机的开停、系统过压保护及欠压保护等功能，从而保证系统在正常工作压力范围内可靠地工作。

目前，各压力开关生产厂对压力开关的检测普遍采用精密压力表读数的检测方法，有的厂家是在给压力开关加压和减压的过程中通过听声音读数的方法来检测的，有的除了听声音外还安装了指示灯，操作人员在观看指示灯和听取声响的同时要及时准确地读取数据。此类方法存在的主要问题有：

⑴ 由于精密压力表一般是在实验室使用，在生产现场长时间、高频率地使用很快就会使其示值超差，所以这种测试方法不适合于大批量生产的需要；

⑵ 精密压力表的刻线很密，很容易出现误读、误判的现象，而且由于工人操作疲劳及人员的主观原因等均为产生读数差异的因素；

⑶ 测试过程长，劳动生产率低。

事实证明：这些检测方法已不能适应现代工业生产中大批量高品质的要求，尤其不能适应汽车和家电工业中对产品高可靠性的要求。

在压力开关的生产过程中，如果没有精密的质量检验设备则产品质量没有保证；没有高质量的自动化检验设备则生产效率难以提高。为此需要研制一种高精度、高速度、智能化的压力开关检验系统。

7.5.2　系统硬件设计方案

压力开关检测属于动态测量，设计的宗旨是用电脑全部代替或部分代替工人的操作，进而达到提高劳动效率和提高检测质量的目的。为此采用如下方案：

⑴ 采用先进的长寿命的高速压力传感器获取压力信号；

⑵ 采用高速的高精度信号放大和 A/D 转换器将压力信号变为数字信号；

⑶ 利用中断方式快速捕捉压力开关的动作点压力；

⑷ 利用计算机的汉字及动画界面使操作直观易懂，大大缩短工人的培训周期。

一、硬件原理框图

系统的基本组成如图 7.18 所示。从压力传感器及被测开关来的信号，先进入数据采

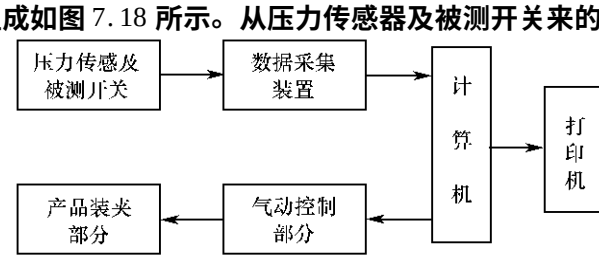


图 7.18　系统组成框图

集装置将模拟信号及开关信号变为数字信号再送入计算机，计算机对所得到的数据进行处理后显示被测开关的有关参数并判断合格与否，同时计算机还能根据操作者的指令控制产品的装夹。

二、数据采集装置

数据采集装置是设备的核心部分，其性能的好坏直接影响着系统的优劣，为此我们采取如图 7.19 所示的方案 图中前置放大和偏置调节将压力传感器来的信号放大并调整为适当的幅值进入 A/D 转换器 被测产品产生的触点信号及系统操作按钮信号经整形电路变为 TTL 电平信号进入数据缓冲器 ；气动部分由电磁阀驱动，控制电磁阀的开关量由数据锁存器产生 数据锁存器和缓冲器的由读者译码器和 PC 总线控制信号产生。

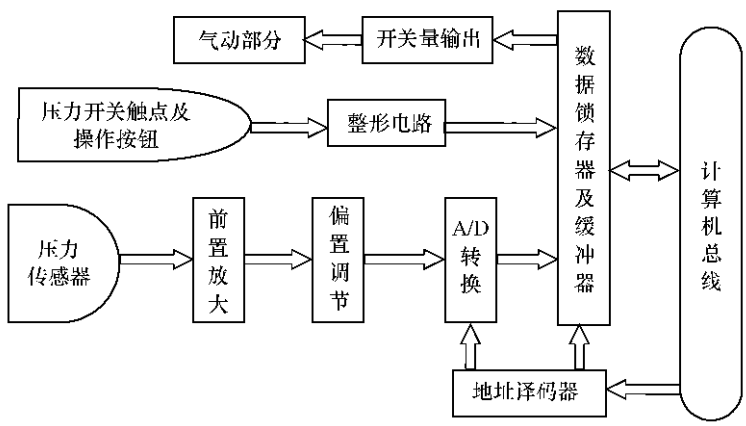


图 7.19　数据采集装置原理框图

三、压力信号处理

图 7.20 为压力信号的处理电路，从压力传感器来的压力信号由端子 S21 和 S24 进入

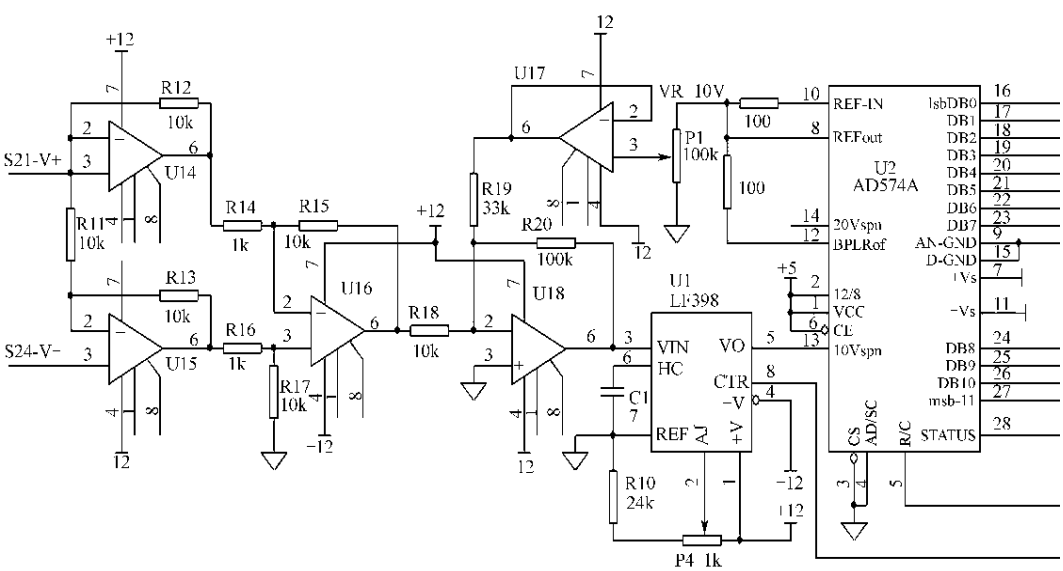


图 7.20　压力信号的处理

放大器的 V + 和 V - 端，放大器由 U14 ~ U18 等 5 个运放组成，其中 U14、U15 和 U16 组成基本的仪表放大器（原理见 6.2.1 节），增益为 30，U18 为加法比例放大器，主回路的增益为 R20/R18 = 10，这样整个压力信号放大的增益为 30 × 10 = 300，对于输出最大值为 30mV 的压力传感器来说，放大后进入 A/D 的最大值为 30mV × 300 = 9000mV = 9V，这正好符合 AD574 的 10V 量程输入。U17 的作用是调节主回路的零点，U17 组成电压跟随器，其输入由 AD574 的基准输出 10V 电压经电位计 P1 调节产生，其输出接在加法器的另一端，当压力输入为零时调节 P1 时使主回路输出为 - 4.6V ~ 4.9V 就达到目的。

这里选用的压力传感器是恒流 1.5mA 供电的，图 7.21 是恒流源发生电路，为了保证恒流源的稳定性，该部分电路的电源是单独的。首先由系统 +5V 经过 DC - DC 变换产生 15V 电源，再由稳压器 L200C（U20）稳成 12V 电源，R94 和 R95 取相等的值其中点和系统的模拟地相连，Z4 是 1.2V 基准电源，从电位计 P3 分得的电压也是和基准电压一样稳定的值 V_{BA}。由于流过负载 RL 的电流即流过 R91 的电流，而 $I_{R91} = V_{BA}/R91 = V_{BA}/0.15$，调节电位计 P3 总可以使 $I_{RL} = I_{R91} = 1.5mA$。

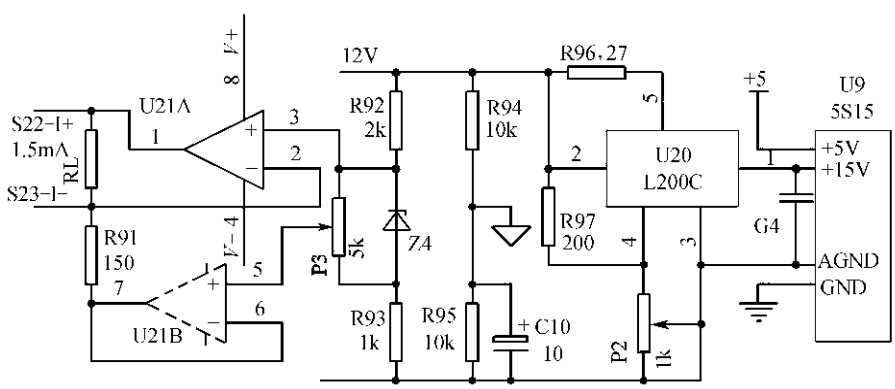


图 7.21　恒流源发生电路

在这部分电路中，凡涉及增益及其恒流源产生的电阻都要求选用万分之五精度等级、优于 30×10^{-6} 的温度系数的电阻。

四、PC 接口电路设计

接口电路如图 7.22 所示，其功能是：

（1）从能控制 S/H 及 A/D 电路，并能从 A/D 取走数据；

（2）能识别压力开关触点的通断信号和人工操作按钮信号；

（3）能输出一组开关量用来控制电磁阀。

实际上这个接口电路实现的是一些基本的数据输入输出功能，我们选用 3 片 74LS244 作为数据输入，其中 U3、U4 的一半用来缓冲 AD574 的转换数据，U4 的另一半和 U6 用来读取开关量。数据输出选用两片 74LS373，其中 U5 用来产生 S/H 及 A/D 的控制信号，U6 用来产生 kgl 输出。PC 的接口信号只用到 8 条数据线（D0 ~ D7）、10 条地址线（A0 ~ A9）、3 条控制线（AEN、IOR、IOW）共 21 条信号线，还用到 PC 总线的 +5V 电源。译码器由 GAL16V8 来实现，GAL 的 5 条输出线分别控制 U3 ~ U7。接口操作译码地址如

表 7.1 所列。

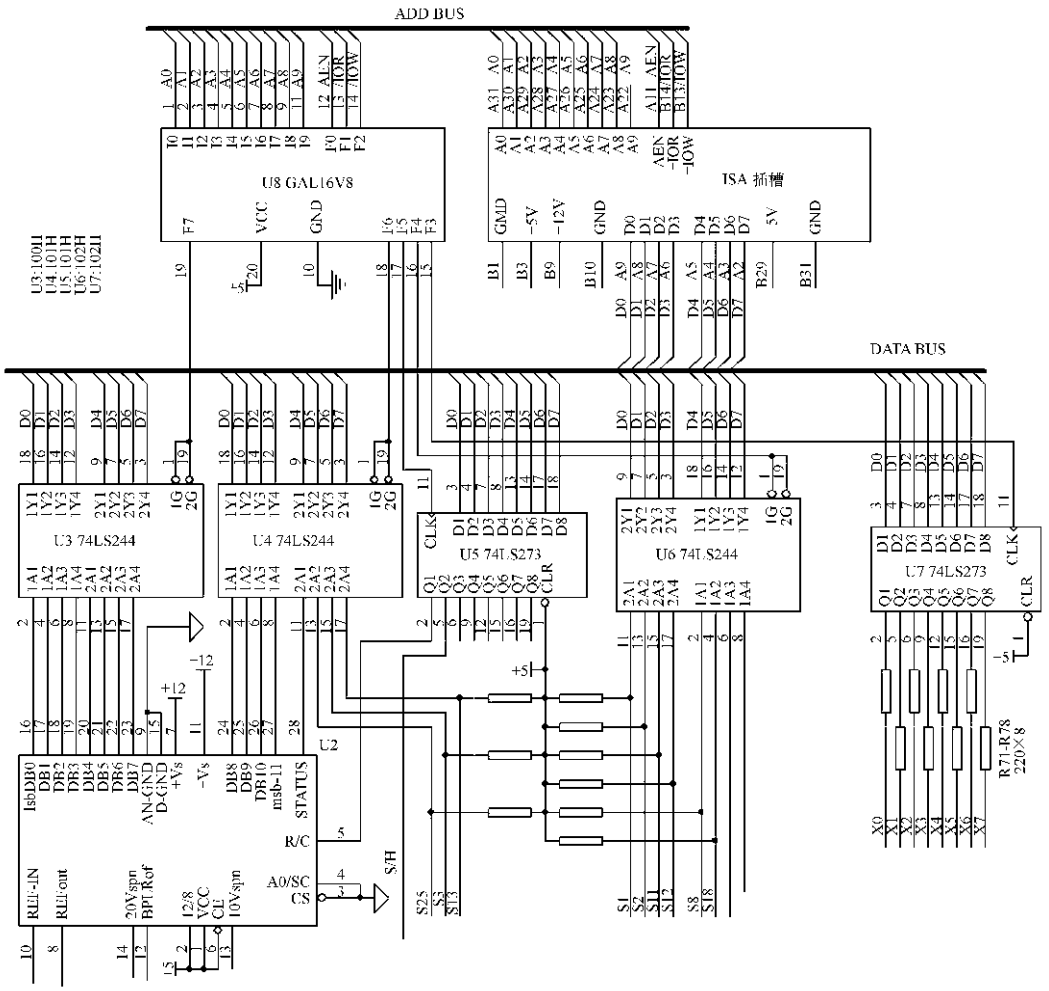


图 7.22 PC 机模拟量输入、开关量输入及开关量输出电路

表 7.1 PC 接口操作译码地址

被操作器件	操作类型	译码地址
U3 (74LS244)	读	100H
U4 (74LS244)	读	101H
U5 (74LS373)	写	101H
U3 (74LS244)	读	102H
U3 (74LS373)	写	102H

从接口输出数据的汇编语言为

……

mov dx，write _ add

```
mov al，out _ data
out dx，al
……
```

从接口输入数据的汇编语言为

```
……
mov dx，in _ add
in al，dx
mov in _ data，al
……
```

7.5.3　系统软件方案

一、软件方框图

检测软件分为 8 个部分，如图 7.23 所示。

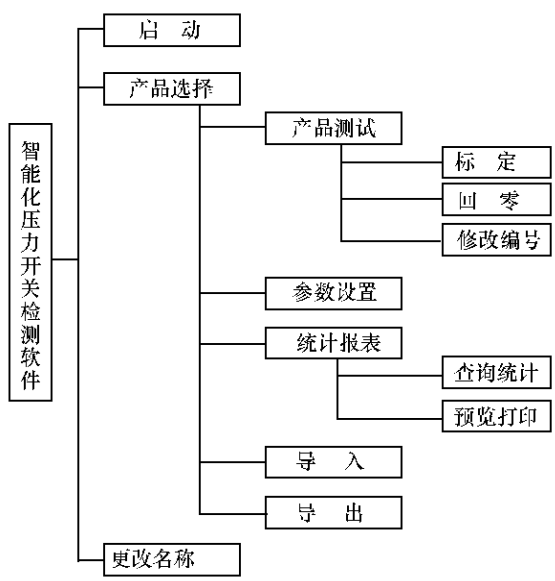


图 7.23　软件框图

二、软件特点

⑴ 使用了多媒体效果使操作简化；

⑵ 尽量使用工作专用语言使系统应用简单明了，与手工操作规程有较好的兼容性；

⑶ 显示屏上有模拟的动态指针压力表，附合了工作人员的操作习惯；

⑷ 工作人员可根据要求用“所见即所得”的方式产生多种统计报表；

⑸ 设计导入导出功能，方便工作人员对数据进行备份，并确保磁盘空间的正常应用；

⑹ 界面操作可以使用触摸屏，适合于工厂的现场环境。

三、部分程序功能简介

（一）测试产品选择

本系统可以一次设定测试 6 种产品，当产品有变化时可以随时更改，操作界面如图 7.24 所示。

（二）测试功能选择

测试功能包括“进入测试”、“进入自检”、“参数设置”、“报表统计”、“导出”、“导入”，操作界面如图 7.25 所示。

图 7.24 产品选择界面

图 7.25 系统基本操作界面

（三）系统自检功能

系统自检功能包括各路电磁阀的功能检验及其气路气密性检验、压力显示值的标定等，前者的操作界面如图 7.26 所示，点击相关按钮即可实现相应功能。在本界面下点击“标定”命令在密码确认后，就进入压力标定界面进行压力标定。

图 7.26 系统自检界面

（四）参数设定

在更换参数产品时要作相关参数设定，这些参数包括压力开关触点动作压力的合格

范围、压力给定过程的区间参数等，如图 7.27 所示，参数的输入可以用键盘输入也可以通过触摸屏输入。

图 7.27　测试设置界面

为了系统的安全性，参数设定和系统自检功能中所述的压力标定一样在进入前需确认密码，密码确认的界面如图 7.28 所示。

图 7.28　密码确认界面

（五）测试模块

测试模块是系统的主要模块，在工作时绝大部分时间显示的就是测试界面，如图7.29所示。该界面显示内容包括：

⑴ 实时压力值指示；

图 7.29 产品测试界面

⑵ 被测产品触点变化模拟指示；

⑶ 被测产品的编号显示；

⑷ 被测产品的触点动作压力实测结果；

⑸ 测试合格或不合格指示。

系统按双工位测试设计，即每次可测两个产品，除了产品的装卸需人工进行外，其它都是自动完成的。在测试过程中，需要人操作的按钮只有两个，其一是“开始”按钮，用来启动一次测试的开始；其二是“急停”按钮，用来在发生意外情况时结束本次测试。上位机程序用 VB 编写，产品测试过程按如下思路进行。

数据采集用线程（W302dll. dll）实现，主程序启动测试模块后线程就启动了，线程的任务是用超过每秒 1000 次的速度不停地采集压力信号和所有的开关量信号，当发现压力开关触点有变化时，及时记下当时的压力值并以发消息的方式通知主程序；主程序主要的任务有两个，其一是根据预先设定的区间控制压力的快增、慢增、快减、慢减，其二是根据当前的压力值和加减压状态、线程是否发出的消息来判断应测触点的测试结果。部分源程序请参阅附录 3。

7.5.4 系统的性能指标

系统的要求主要体现在测试精度和测试速度两方面，前者指的是动态测试过程中压力开关触点变化时测得压力值和实际压力值的差别，后者指测试一个压力开关需要多长时间。经标定和使用方检验该设备可达到如下指标。

⑴ **测试精度：**< 0.2%。
⑵ **标定精度：**< 0.1%。
⑶ **测试速度：**1 个/13s。
⑷ **电源电压：**180V ~ 250V。
⑸ **低压气源：**0.3MPa ~ 0.7MPa。
⑹ **高压气源：**4MPa。

附录 1　MyUSB 源程序

单片机：C8051F020
USB **芯片：**PDIUSBD12

（一） 头文件

头文件共 6 **个，分别为**"d12ci.h"、"myloop.h"、"epphal.h"、"usb100.h"、"chap_9.h" **和**"protodma.h**，其中后** 4 **个完全由厂商提供未做任何修改这里没有列出。**

```
//=========================================
// File Name:      D12CI.H
//=========================================
#ifndef __D12CI_H__
#define __D12CI_H__

#define D12_NOLAZYCLOCK             0x02
#define D12_CLOCKRUNNING            0x04
#define D12_INTERRUPTMODE           0x08
#define D12_SOFTCONNECT             0x10
#define D12_ENDP_NONISO             0x00
#define D12_ENDP_ISOOUT             0x40
#define D12_ENDP_ISOIN              0x80
#define D12_ENDP_ISOIO              0xC0

#define D12_CLOCK_12M               0x03
#define D12_CLOCK_4M                0x0b
#define D12_SETTOONE                0x40
#define D12_SOFONLY                 0x80

#define D12_DMASINGLE               0x00
#define D12_BURST_4                 0x01
#define D12_BURST_8                 0x02
#define D12_BURST_16                0x03
#define D12_DMAENABLE               0x04
#define D12_DMA_INTOKEN             0x08
#define D12_AUTOLOAD                0x10
#define D12_NORMALPLUSSOF           0x20
#define D12_ENDP4INTENABLE          0x40
#define D12_ENDP5INTENABLE          0x80 // bug fixed in V2.1

#define D12_INT_ENDP0OUT            0x01
#define D12_INT_ENDP0IN             0x02
#define D12_INT_ENDP1OUT            0x04
#define D12_INT_ENDP1IN             0x08
#define D12_INT_ENDP2OUT            0x10
#define D12_INT_ENDP2IN             0x20
#define D12_INT_BUSRESET            0x40
#define D12_INT_SUSPENDCHANGE       0x80
#define D12_INT_EOT                 0x0100
```

```
#define D12_SETUPPACKET                 0x20

#define D12_BUFFER0FULL                 0x20
#define D12_BUFFER1FULL                 0x40

#define D12_FULLEMPTY           0x01
#define D12_STALL               0x02

void D12_SetAddressEnable(unsigned char bAddress, unsigned char bEnable);
void D12_SetEndpointEnable(unsigned char bEnable);
void D12_SetMode(unsigned char bConfig, unsigned char bClkDiv);
void D12_SetDMA(unsigned char bMode);
unsigned char D12_GetDMA(void);
unsigned short D12_ReadInterruptRegister(void);
unsigned char D12_SelectEndpoint(unsigned char bEndp);
unsigned char D12_ReadLastTransactionStatus(unsigned char bEndp);
unsigned char D12_ReadEndpointStatus(unsigned char bEndp);
void D12_SetEndpointStatus(unsigned char bEndp, unsigned char bStalled);
void D12_SendResume(void);
unsigned short D12_ReadCurrentFrameNumber(void);
unsigned short D12_ReadChipID(void);

unsigned char D12_ReadEndpoint(unsigned char endp, unsigned char len, unsigned char * buf);
unsigned char D12_WriteEndpoint(unsigned char endp, unsigned char len, unsigned char * buf);
void D12_AcknowledgeEndpoint(unsigned char endp);

unsigned char D12_ReadMainEndpoint(unsigned char * buf); // V2.2

unsigned char inportb(unsigned int addr);
void outportb(unsigned int addr, unsigned char Data);

#endif

//=====================================
// File Name:           Myloop.H
//=====================================

#ifndef __MAINLOOP_H__
#define __MAINLOOP_H__

//*************************************************************************
// basic #defines
//*************************************************************************
#define MAX_ENDPOINTS          (unsigned char)0x3

#define EP0_TX_FIFO_SIZE     16
#define EP0_RX_FIFO_SIZE     16
#define EP0_PACKET_SIZE       16

#define EP1_TX_FIFO_SIZE     16
#define EP1_RX_FIFO_SIZE     16
#define EP1_PACKET_SIZE       16

#define EP2_TX_FIFO_SIZE     64
#define EP2_RX_FIFO_SIZE     64
#define EP2_PACKET_SIZE       64
```

```
#define USB_IDLE                    0
#define USB_TRANSMIT                1
#define USB_RECEIVE                 2

#define USB_CLASS_CODE_TEST_CLASS_DEVICE                    0xdc
#define USB_SUBCLASS_CODE_TEST_CLASS_D12                    0xA0
#define USB_PROTOCOL_CODE_TEST_CLASS_D12                    0xB0
//*****************************************************************************
// masks
//*****************************************************************************
#define USB_RECIPIENT                   (unsigned char)0x1F
#define USB_RECIPIENT_DEVICE            (unsigned char)0x00
#define USB_RECIPIENT_INTERFACE         (unsigned char)0x01
#define USB_RECIPIENT_ENDPOINT          (unsigned char)0x02

#define USB_REQUEST_TYPE_MASK           (unsigned char)0x60
#define USB_STANDARD_REQUEST            (unsigned char)0x00
#define USB_CLASS_REQUEST               (unsigned char)0x20
#define USB_VENDOR_REQUEST              (unsigned char)0x40

#define USB_REQUEST_MASK                (unsigned char)0x0F

#define DEVICE_ADDRESS_MASK             0x7F
//*****************************************************************************
// macros
//*****************************************************************************
#define SWAP(x)     ((((x) & 0xFF) << 8) | (((x) >> 8) & 0xFF))

#define MSB(x)      (((x) >> 8) & 0xFF)
#define LSB(x)      ((x) & 0xFF)

#define FALSE       0
#define TRUE        (!FALSE)

//*****************************************************************************
// basic typedefs
//*****************************************************************************
typedef unsigned char       UCHAR;
typedef unsigned short      USHORT;
typedef unsigned long       ULONG;
typedef unsigned char       BOOL;
//*****************************************************************************
// structure and union definitions
//*****************************************************************************
typedef union _epp_flags
{
    struct _flags
    {
        unsigned char timer                 : 1;
        unsigned char bus_reset             : 1;
        unsigned char suspend               : 1;
        unsigned char setup_packet          : 1;
        unsigned char remote_wakeup             : 1;
        unsigned char in_isr                : 1;
        unsigned char control_state         : 2;

        unsigned char configuration         : 1;
        unsigned char verbose               : 1;
        unsigned char ep1_rxdone            : 1;
```

```
            unsigned char setup_dma                 : 2; // V2.3
            unsigned char dma_state             : 2;
            unsigned char power_down            : 1; // Smart Board
            unsigned char ep2_rxdone            : 1; //zzz2003.1.6
            unsigned char ep2_txdone            : 1; //zzz2003.1.6

        } bits;
        unsigned short value;
} EPPFLAGS;

typedef struct _device_request
{
        unsigned char bmRequestType;
        unsigned char bRequest;
        unsigned short wValue;
        unsigned short wIndex;
        unsigned short wLength;
} DEVICE_REQUEST;

typedef struct _IO_REQUEST {
        unsigned short    uAddressL;
        unsigned char     bAddressH;
        unsigned short    uSize;
        unsigned char     bCommand;
} IO_REQUEST, *PIO_REQUEST;

#define MAX_CONTROLDATA_SIZE 8

typedef struct _control_xfer
{
        DEVICE_REQUEST DeviceRequest;
        unsigned short wLength;
        unsigned short wCount;
        unsigned char * pData;
        unsigned char dataBuffer[MAX_CONTROLDATA_SIZE];
} CONTROL_XFER;

//****************************************************************************
// USB utility functions
//****************************************************************************
void fn_usb_isr();

extern void suspend_change(void);
extern void stall_ep0(void);
extern void disconnect_USB(void);
extern void connect_USB(void);
extern void reconnect_USB(void);
extern void init_unconfig(void);
extern void init_config(void);
extern void single_transmit(unsigned char * pData, unsigned char len);
extern void code_transmit(unsigned char code * pRomData, unsigned short len);

extern void control_handler();
extern void check_key_LED(void);
extern void setup_dma();

void dma_start(PIO_REQUEST);

#define IN_TOKEN_DMA   1
```

```
#define OUT_TOKEN_DMA       0

#define DMA_BUFFER_SIZE             256

#define DMA_IDLE    0
#define DMA_RUNNING    1
#define DMA_PENDING    2

#define SETUP_DMA_REQUEST               0x0471
#define GET_FIRMWARE_VERSION       0x0472
#define GET_SET_TWAIN_REQUEST      0x0473
#define GET_BUFFER_SIZE                 0x0474

typedef struct _TWAIN_FILEINFO {
        unsigned char     bPage;       // bPage bit 7 - 5 map to uSize bit 18 - 16
        unsigned char     uSizeH;       // uSize bit 15 - 8
        unsigned char     uSizeL;       // uSize bit 7 - 0
} TWAIN_FILEINFO, *PTWAIN_FILEINFO;

#endif
```

（二）程序

程序包括主程序‘ Myloop.c **”、中断服务子程序‘** Isr.c **”、基本输入输出子程序‘** D12ci.c **”、基本协议程序“** CHAP_9.c **”、厂商请求程序“** PROTODMA.c **”、硬件接口控制程序“** EPPHAL.c **”，其中后** 3 **个也完全由厂商提供未做任何修改这里没有列出。**

```
//=========================================================
// File Name:      myloop.c
//=========================================================

#include <c8051f020.h>
#include "epphal.h"
#include "d12ci.h"
#include "myloop.h"
#include "usb100.h"
#include "chap_9.h"
#include "protodma.h"

//-------------------------------------------------------------------------------
// Global CONSTANTS
//-------------------------------------------------------------------------------
//#define SYSCLK 16000000                                 // approximate SYSCLK frequency in Hz

//USB 标准请求
code void (*StandardDeviceRequest[])(void) =
{
        get_status,
        clear_feature,
        reserved,
        set_feature,
        reserved,
        set_address,
        get_descriptor,
        reserved,
        get_configuration,
        set_configuration,
        get_interface,
        set_interface,
```

```
        reserved,
        reserved,
        reserved,
        reserved
};

//用户厂商请求
code void (*VendorDeviceRequest[])(void) =
{
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        reserved,
        read_write_register,
        reserved,
        reserved,
        reserved
};

/*
//***************************************************************************
//    Public static data
//***************************************************************************
*/

extern EPPFLAGS bEPPflags;
extern unsigned long ClockTicks;
extern unsigned char idata GenEpBuf[];
extern IO_REQUEST idata ioRequest;

extern unsigned char ioSize, ioCount;
extern unsigned char idata EpBuf[];

CONTROL_XFER ControlData;
BOOL bNoRAM;

code char * _NAME_USB_REQUEST_DIRECTION[] =
{
"Host_to_device",
"Device_to_host"
};

code char * _NAME_USB_REQUEST_RECIPIENT[] =
{
"Device",
"Interface",
"Endpoint(0)",
"Other"
};

code char * _NAME_USB_REQUEST_TYPE[] =
```

```
{
"Standard",
"Class",
"Vendor",
"Reserved"
};

code char * _NAME_USB_STANDARD_REQUEST[] =
{
"GET_STATUS",
"CLEAR_FEATURE",
"RESERVED",
"SET_FEATURE",
"RESERVED",
"SET_ADDRESS",
"GET_DESCRIPTOR",
"SET_DESCRIPTOR",
"GET_CONFIGURATION",
"SET_CONFIGURATION",
"GET_INTERFACE",
"SET_INTERFACE",
"SYNC_FRAME"
};


void help_devreq(unsigned char typ, unsigned char req)
{
        typ >>= 5;

        if(typ == USB_STANDARD_REQUEST) {
        }
        else {
                if(bEPPflags.bits.verbose)
                        /*printf("Request Type = %s, bRequest = 0x%bx.\n", _NAME_USB_REQUEST_TYPE[typ],
                                req)*/;
        }
}



//中断设置
void init_special_interrupts(void)
{
        IT0 = 0;                //bianyan
        EX0 = 1;
        PX0 = 1;
        ENABLE;
}

//I/O 口初始化程序
void init_port()
{
//      P0 = 0xFF;
//      P1 = 0xFF;
//      P2 = 0xFF;
//      P3 = 0xFF;
        MCU_D12CS = 0x0;
        D12SUSPD = 0;
```

```
}

/*Serial Port */
/*Mode                    = 1    /8-bit UART
   Serial Port Interrupt       = Disabled              */
/*Receive                 = Enabled      */
/*Auto Addressing         = Disabled      */
//串行口设置
void init_serial(void)
{
        SCON0 = 0X52;           //0101 0000
//      PCON    = 0X00 | PCON;//0000 0000
        TMOD = 0X20;            //0010 0000
        TR1 = 1;    //TCON = 0x69;          //0100 1001
        ET1 = 0;    //not prove T1 int
        TH1 = 0x10;
        TL1 = 0x10;

}


//返回 stall 应答
void stall_ep0(void)
{
        D12_SetEndpointStatus(0, 1);
        D12_SetEndpointStatus(1, 1);
}

//断开 USB 总线
void disconnect_USB(void)
{
        // Initialize D12 configuration
        D12_SetMode(D12_NOLAZYCLOCK, D12_SETTOONE | D12_CLOCK_12M);
}

//连接 USB 总线
void connect_USB(void)
{
        // reset event flags
        DISABLE;
        bEPPflags.value = 0;//清除所有状态
        ENABLE;

        // V2.1 enable normal+sof interrupt
        D12_SetDMA(D12_ENDP4INTENABLE | D12_ENDP5INTENABLE);

        // Initialize D12 configuration
        D12_SetMode(D12_NOLAZYCLOCK|D12_SOFTCONNECT,D12_SETTOONE | D12_CLOCK_12M);
}

//重新连接到 USB 总线
void reconnect_USB(void)
{
        D12SUSPD = 0;
        disconnect_USB();


                ;
```

```
	connect_USB();

}

//恢复到未配置状态
void init_unconfig(void)
{
//	unsigned char i;

	D12_SetEndpointEnable(0); /* Disable all endpoints but EPP0. */
}

//设置配置状态
void init_config(void)
{
	D12_SetEndpointEnable(1); /* Enable    generic/iso endpoints. */
}

//从端点号 1 发送数据
void single_transmit(unsigned char * buf, unsigned char len)
{
	if( len <= EP0_PACKET_SIZE) {
		D12_WriteEndpoint(1, len, buf);
	}
}

//发送端点号 1 建立代码
void code_transmit(unsigned char code * pRomData, unsigned short len)
{
	ControlData.wCount = 0;
	if(ControlData.wLength > len)
		ControlData.wLength = len;

	ControlData.pData = pRomData;
	if( ControlData.wLength >= EP0_PACKET_SIZE) {
		D12_WriteEndpoint(1, EP0_PACKET_SIZE, ControlData.pData);//发送 16 字节数据
		ControlData.wCount += EP0_PACKET_SIZE;
		DISABLE;
		bEPPflags.bits.control_state = USB_TRANSMIT;
		ENABLE;
	}
	else {
		D12_WriteEndpoint(1, ControlData.wLength, pRomData);//发送 16 字节内数据
		ControlData.wCount += ControlData.wLength;
		DISABLE;
		bEPPflags.bits.control_state = USB_IDLE;
		ENABLE;
	}
}

//DMA 建立子程序
void setup_dma()
{

}

//请求处理子程序
```

```
void control_handler()
{
	unsigned char type, req;

	type = ControlData.DeviceRequest.bmRequestType & USB_REQUEST_TYPE_MASK;
	req = ControlData.DeviceRequest.bRequest & USB_REQUEST_MASK;

	help_devreq(type, req); //显示设备请求

	if (type == USB_STANDARD_REQUEST)
		(*StandardDeviceRequest[req])();//调用标准请求
	else if (type == USB_VENDOR_REQUEST)
		(*VendorDeviceRequest[req])();//调用厂商请求
	else
		stall_ep0();
}
//*****************************************************
void main(void)
{
	unsigned char i;
	unsigned char Acounter=1,Bcounter=1;
	long counter = 0;
	bit c04_read_done = 0;
//------------------通用初始化----------------------
	DISABLE;
	WDTCN = 0XDE;
	WDTCN = 0XAD;
//	OSCICN = 0X15;		//1**1 0110
//	OSCXCN = 0X67;			//select extenal osc,before this internal osc=2MH
//	delay(50);
//	while(!OSCICN & 0x80);
//	OSCICN = 0X1a;
	ENABLE;
	XBR0 = 0X04;
	XBR1 = 0X04;   //0000 0100	P0.0 as int0
	XBR2 = 0X40;

//-------------测试准备工作--------------------
	P0MDOUT= 0xc0;			//1100 0000p0.6 p0.7 is push-pull
	P1MDOUT = 0xff;			//port1 is push-pull out
	P2MDOUT = 0x3f;			//p2.5~0 is push-pull;
	P3MDOUT = 0x3f;			//"0011 0000",p3.5    p3.4 is push-pull out


//	MCU_D12CS = 1;

  //---------------------有关 USB 口的初始化----------------------
	P3MDOUT = P3MDOUT | 0xa0;		//10100000
//	MCU_D12RST = 0;
//	delay(50);
//	MCU_D12RST = 1;
	init_port();//初始化 I/O 口
	init_serial();//初始化串行口

	//注:串行口是用来外扩 LCD 和键盘,用于查询显示当前的工作状态
	//在 USB Smart Board 标准配置中并未带该 LCD 和键盘,这里给出的程序仅供参考
```

```
//      init_timer0();//初始化定时器 0
        init_special_interrupts();//设置中断

//      MCU_D12CS = 0x1;

        MCU_D12CS = 0x0;

        D12_ReadChipID();

        if(MCU_SWM0 == 0 && MCU_SWM1 == 0) {
                MCU_D12RST = 0;//DMA 设置
                MCU_D12RST = 1;
                D12_SetDMA(0x0);
        }

        bEPPflags.value = 0;
        reconnect_USB();//重新连接 USB
/*
        if((i = D12_GetDMA()) == 0xC3) {
                D12_SendResume();//发送恢复处理
        }
        else {
                bEPPflags.value = 0;
                reconnect_USB();//重新连接 USB
        }
*/
        /* Main program loop */

        while( TRUE ){

                if (bEPPflags.bits.timer){
                        DISABLE;//定时器溢出,检测按键状态
                        bEPPflags.bits.timer = 0;
                        ENABLE;

                        if(bEPPflags.bits.configuration)//设备未配置返回
                                check_key_LED();
                }


                if (bEPPflags.bits.bus_reset) {//设备复位处理
                        DISABLE;
                        bEPPflags.bits.bus_reset = 0;
                        ENABLE;
                        // Release D12's SUSPEND pin after bus reset
                        // Enable 74HCT123 pulse generation before disconnect
                        D12SUSPD = 1;
                } // if bus reset

                if (bEPPflags.bits.suspend) {//挂起改变处理
                        DISABLE;
                        bEPPflags.bits.suspend= 0;
                        ENABLE;

                        if(D12SUSPD == 1) {//挂起处理
                        /*      D12SUSPD = 0;
                                P0 = 0xFF;
```

```
                    P1 = 0xFF;
                    P2 = 0xFF;
                    P3 = 0xFF;
                    D12_SetDMA(0xC3);
                    D12SUSPD = 1;
                    PCON |= 0x02;
                    while (1);   */
                }
            } // if suspend change

            if (bEPPflags.bits.setup_packet){//Setup 包处理
                DISABLE;
                bEPPflags.bits.setup_packet = 0;
                ENABLE;
                control_handler();//调用请求处理子程序
                D12SUSPD = 1;
            } // if setup_packet

            if(bEPPflags.bits.setup_dma != 0) {//DMA 状态改变处理
                DISABLE;
                bEPPflags.bits.setup_dma --;
                ENABLE;
                setup_dma();
            } // if setup_dma
            if(bEPPflags.bits.ep2_rxdone)
               {//---------中断接收中用户部分---------
                bEPPflags.bits.ep2_rxdone = 0;

               }//---------中断接收中用户部分---------


            //------------------MAINLOOP 中用户部分-------------------

            //-----------------MAINLOOP 中用户部分-----------
        } // Main Loop
}


//=====================================================
// File Name:            ISR.C
//=====================================================


#include <c8051f020.h>

#include "epphal.h"
#include "d12ci.h"
#include "myloop.h"
#include "usb100.h"

extern void bus_reset(void);

extern void ep0_txdone(void);
extern void ep0_rxdone(void);

extern void ep1_txdone(void);
extern void ep1_rxdone(void);
```

```
extern void main_txdone(void);
extern void main_rxdone(void);

extern void dma_eot(void);

/*
//*****************************************************************************
//    Public static data
//*****************************************************************************
*/

EPPFLAGS bEPPflags;

/* Control endpoint TX/RX buffers */
extern CONTROL_XFER ControlData;

/* ISR static vars */
unsigned char idata GenEpBuf[EP1_PACKET_SIZE];
unsigned char idata EpBuf[EP2_PACKET_SIZE];
IO_REQUEST idata ioRequest;
unsigned char ioSize, ioCount;
unsigned long ClockTicks = 0;

extern BOOL bNoRAM;



//USB 中断处理
usb_isr() interrupt 0
{
        DISABLE;
//      P3IF = P3IF & 0x7f;             //clear biao zhi
        fn_usb_isr();
        ENABLE;
}

//USB 中断服务子程序
void fn_usb_isr()
{
        unsigned int i_st;

        bEPPflags.bits.in_isr = 1;

        i_st = D12_ReadInterruptRegister();//读取中断寄存器

        if(i_st != 0) {
                if(i_st & D12_INT_BUSRESET) {
                        bus_reset();//USB 总线服务
                        bEPPflags.bits.bus_reset = 1;
                }

                if(i_st & D12_INT_EOT)
                        dma_eot();//DMA 传输结束

                if(i_st & D12_INT_SUSPENDCHANGE)
                        bEPPflags.bits.suspend = 1;//挂起改变

                if(i_st & D12_INT_ENDP0IN)
```

```
            ep0_txdone();//端点 0IN 中断
        if(i_st & D12_INT_ENDP0OUT)
            ep0_rxdone();//端点 0OUT 中断
        if(i_st & D12_INT_ENDP1IN)
            ep1_txdone();//端点 1IN 中断
        if(i_st & D12_INT_ENDP1OUT)
            ep1_rxdone();//端点 1OUT 中断
        if(i_st & D12_INT_ENDP2IN)
            main_txdone();//端点 2IN 中断
        if(i_st & D12_INT_ENDP2OUT)
            main_rxdone();//端点 2OUT 中断
    }

    bEPPflags.bits.in_isr = 0;
}

//总线复位处理子程序
void bus_reset(void)
{

}

//端点 0OUT 中断
void ep0_rxdone(void)
{
    unsigned char ep_last, i;

    ep_last = D12_ReadLastTransactionStatus(0); //清中断标志

    if (ep_last & D12_SETUPPACKET) {
        //接收到 SETUP 包
        ControlData.wLength = 0;
        ControlData.wCount = 0;

        if( D12_ReadEndpoint(0, sizeof(ControlData.DeviceRequest),
            (unsigned char *)(&(ControlData.DeviceRequest))) != sizeof(DEVICE_REQUEST) ) {
            //SETUP 包出错,返回
            D12_SetEndpointStatus(0, 1);
            D12_SetEndpointStatus(1, 1);
            bEPPflags.bits.control_state = USB_IDLE;
            return;
        }

        ControlData.DeviceRequest.wValue = SWAP(ControlData.DeviceRequest.wValue);
        ControlData.DeviceRequest.wIndex = SWAP(ControlData.DeviceRequest.wIndex);
        ControlData.DeviceRequest.wLength = SWAP(ControlData.DeviceRequest.wLength);

        //对控制端点的输入/输出进行应答
        D12_AcknowledgeEndpoint(0);
        D12_AcknowledgeEndpoint(1);

        ControlData.wLength = ControlData.DeviceRequest.wLength;
        ControlData.wCount = 0;

        if (ControlData.DeviceRequest.bmRequestType & (unsigned
char)USB_ENDPOINT_DIRECTION_MASK) {
            //从主机传输数据
```

```
                bEPPflags.bits.setup_packet = 1;
                bEPPflags.bits.control_state = USB_TRANSMIT;          /* get command */
            }
            else {
                if (ControlData.DeviceRequest.wLength == 0) {
                    bEPPflags.bits.setup_packet = 1;
                    bEPPflags.bits.control_state = USB_IDLE;          /* set command */
                }
                else {
                    if(ControlData.DeviceRequest.wLength > MAX_CONTROLDATA_SIZE) {
                        //接收数据长度为 0
                        bEPPflags.bits.control_state = USB_IDLE;
                        D12_SetEndpointStatus(0, 1);
                        D12_SetEndpointStatus(1, 1);
                    }
                    else {
                        bEPPflags.bits.control_state = USB_RECEIVE;     //设置接收状态
                    }
                } // set command with data
            } // else set command
        } // if setup packet

        else if (bEPPflags.bits.control_state == USB_RECEIVE) {
            //接收数据
            i =    D12_ReadEndpoint(0, EP0_PACKET_SIZE,
                ControlData.dataBuffer + ControlData.wCount);
            ControlData.wCount += i;
            if( i != EP0_PACKET_SIZE || ControlData.wCount >= ControlData.wLength) {
                //数据接收完毕
                bEPPflags.bits.setup_packet = 1;
                bEPPflags.bits.control_state = USB_IDLE;
            }
        }
        else {
            bEPPflags.bits.control_state = USB_IDLE;//进入等待状态
        }
}

//端点 0IN 处理
void ep0_txdone(void)
{
        short i = ControlData.wLength - ControlData.wCount;
        D12_ReadLastTransactionStatus(1); //清中断标志位
        if (bEPPflags.bits.control_state != USB_TRANSMIT)
            return;//非发送状态,返回

        if( i >= EP0_PACKET_SIZE) {
            //剩下数据大于 16 字节,发送 16 字节
            D12_WriteEndpoint(1, EP0_PACKET_SIZE, ControlData.pData + ControlData.wCount);
            ControlData.wCount += EP0_PACKET_SIZE;
            bEPPflags.bits.control_state = USB_TRANSMIT;
        }
        else if( i != 0) {
            //发送剩下数据
            D12_WriteEndpoint(1, i, ControlData.pData + ControlData.wCount);
            ControlData.wCount += i;
            bEPPflags.bits.control_state = USB_IDLE;
```

```
        }
        else if (i == 0){
                D12_WriteEndpoint(1, 0, 0); //发送完毕,发送 0 字节
                bEPPflags.bits.control_state = USB_IDLE;
        }
}

//DMA 结束处理
void dma_eot(void)
{
}

//端点 1OUT 处理
void ep1_txdone(void)
{
        D12_ReadLastTransactionStatus(3); //清中断标志位

}

//端点 1IN 处理
void ep1_rxdone(void)
{
        unsigned char len;

        D12_ReadLastTransactionStatus(2); //清中断标志位
        len = D12_ReadEndpoint(2, sizeof(GenEpBuf), GenEpBuf);//读取数据
        if(len != 0)
                bEPPflags.bits.ep1_rxdone = 1;//标志接收到数据
}

//主端点 OUT 控制
void main_txdone(void)
{
//      unsigned char len,epstatus;

        D12_ReadLastTransactionStatus(5); //清中断标志位
        bEPPflags.bits.ep2_txdone = 1;          //标志发走数据
}

//主端点 IN 控制
void main_rxdone(void)
{
        unsigned char len,epstatus;

        D12_ReadLastTransactionStatus(4); //清中断标志位

        //接收数据
        len = D12_ReadEndpoint(4, 64, EpBuf);
        epstatus=D12_ReadEndpointStatus(4);
        epstatus &= 0x60;
        if (epstatus == 0x60)
                len = D12_ReadEndpoint(4, 64, EpBuf);//读取双缓冲区数据
        if(len != 0)//zzz
                bEPPflags.bits.ep2_rxdone = 1;//标志接收到数据//zzz
}

//==========================================
```

```
// File Name:            D12ci.c
//=====================================

#include <c8051f020.h>
#include "myloop.h"
#include "d12ci.h"
#include "absacc.h"
#include "epphal.h"

#define D12_DATA 0x00          //0xff02
#define D12_COMMAND 0x01       //0xff03

extern EPPFLAGS bEPPflags;

//设置地址使能
void D12_SetAddressEnable(unsigned char bAddress, unsigned char bEnable)
{
    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0xD0);//输出指令
    if(bEnable)
        bAddress |= 0x80;
    outportb(D12_DATA, bAddress);//设置地址

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;
}

//设置端点使能
void D12_SetEndpointEnable(unsigned char bEnable)
{
    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0xD8);//输出指令
    if(bEnable)
        outportb(D12_DATA, 1);//设置端点允许
    else
        outportb(D12_DATA, 0);//设置端点禁止

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;
}

//模式设置
void D12_SetMode(unsigned char bConfig, unsigned char bClkDiv)
{
    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0xF3);//设置模式
    outportb(D12_DATA, bConfig);
    outportb(D12_DATA, bClkDiv);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;
```

```
}

//DMA 工作方式设置
void D12_SetDMA(unsigned char bMode)
{
/*      if(bEPPflags.bits.in_isr == 0)
            DISABLE;

        outportb(D12_COMMAND, 0xFB);//设置 DMA 工作方式
        outportb(D12_DATA, bMode);

        if(bEPPflags.bits.in_isr == 0)
            ENABLE;
*/
}

//读取中断寄存器
unsigned short D12_ReadInterruptRegister(void)
{
        unsigned char b1;
        unsigned int j;

        outportb(D12_COMMAND, 0xF4);//读取中断寄存器
        b1 = inportb(D12_DATA);
        j = inportb(D12_DATA);

        j <<= 8;
        j += b1;

        return j;
}

//端点选择
unsigned char D12_SelectEndpoint(unsigned char bEndp)
{
        unsigned char c;

        if(bEPPflags.bits.in_isr == 0)
            DISABLE;

        outportb(D12_COMMAND, bEndp);//端点选择
        c = inportb(D12_DATA);

        if(bEPPflags.bits.in_isr == 0)
            ENABLE;

        return c;
}

//读取最后传输状态
unsigned char D12_ReadLastTransactionStatus(unsigned char bEndp)
{
        outportb(D12_COMMAND, 0x40 + bEndp);//返回最后传输状态
        return inportb(D12_DATA);
}

//读取端点状态
unsigned char D12_ReadEndpointStatus(unsigned char bEndp)
```

```
{
    unsigned char c;

    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0x80 + bEndp);//读取端点状态
    c = inportb(D12_DATA);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;

    return c;
}

//设置端点状态
void D12_SetEndpointStatus(unsigned char bEndp, unsigned char bStalled)
{
    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0x40 + bEndp);//设置端点状态
    outportb(D12_DATA, bStalled);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;
}

//传输恢复
void D12_SendResume(void)
{
    outportb(D12_COMMAND, 0xF6);//设置发送恢复
}

//读取当前帧号
unsigned short D12_ReadCurrentFrameNumber(void)
{
    unsigned short i,j;

    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0xF5);//读取当前帧数目
    i= inportb(D12_DATA);
    j = inportb(D12_DATA);

    i += (j<<8);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;

    return i;
}

unsigned short D12_ReadChipID(void)
{
    unsigned short i,j;
```

```
    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, 0xFD);
    i=inportb(D12_DATA);
    j=inportb(D12_DATA);
    i += (j<<8);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;

    return i;
}
```

//读取端点数据

```
unsigned char D12_ReadEndpoint(unsigned char endp, unsigned char len, unsigned char * buf)
{
    unsigned char i, j;

    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, endp);
    if((inportb(D12_DATA) & D12_FULLEMPTY) == 0) {
        if(bEPPflags.bits.in_isr == 0)
            ENABLE;
        return 0;
    }

    outportb(D12_COMMAND, 0xF0);
    j = inportb(D12_DATA);
    j = inportb(D12_DATA);

    if(j > len)
        j = len;

    for(i=0; i<j; i++)
        *(buf+i) = inportb(D12_DATA);

    outportb(D12_COMMAND, 0xF2);

    if(bEPPflags.bits.in_isr == 0)
        ENABLE;

    return j;
}
```

//写端点

```
unsigned char D12_WriteEndpoint(unsigned char endp, unsigned char len, unsigned char * buf)
{
    unsigned char i;

    if(bEPPflags.bits.in_isr == 0)
        DISABLE;

    outportb(D12_COMMAND, endp);
    i = inportb(D12_DATA);
```

```
        outportb(D12_COMMAND, 0xF0);
        outportb(D12_DATA, 0);
        outportb(D12_DATA, len);

        for(i=0; i<len; i++)
                outportb(D12_DATA, *(buf+i));

        outportb(D12_COMMAND, 0xFA);

        if(bEPPflags.bits.in_isr == 0)
                ENABLE;

        return len;
}

//设置端点应答
void D12_AcknowledgeEndpoint(unsigned char endp)
{
        outportb(D12_COMMAND, endp);
        outportb(D12_COMMAND, 0xF1);
        if(endp == 0)
                outportb(D12_COMMAND, 0xF2);
}
/*
//输出数据
void outportb(unsigned int Addr, unsigned char Data)
{
        *((unsigned char xdata *) Addr) = Data;
}

//输入数据
unsigned char inportb(unsigned int Addr)
{
        return *((unsigned char xdata *) Addr);
}
*/
void outportb(unsigned int Addr, unsigned char Data)
{
        unsigned char P2_v;             //take p2 volue
        MCU_D12CS = 0;
        P2_v = P2;
        P2 = P2_v & 0xcf;               //close u21 u22 data bus
        if(Addr) MCU_A0 = 1;
        else MCU_A0 = 0;
        P74OUT = 0xff;                  //p7,p6 p5 p4 is push-pull
        P7 = Data;
        P4 = P4 & 0x7f; //    /wr=0 length_h must < 0x3ff
        P4 = P4 & 0x7f; //delay
        P4 = P4 | 0x80;   //    /wr=1
        P4 = P4 | 0x80;   //delay
        P74OUT = 0x00;                  //p7,p6 p5 p4 is open-d
        P2 = P2_v | 0x30;               //reset P2 volue
//      MCU_D12CS = 1;
}

//输入数据
unsigned char inportb(unsigned int Addr)
{
```

```
    unsigned char temp_v;
    unsigned char P2_v;          //take p2 volue
    MCU_D12CS = 0;
    P2_v = P2;
    P2 = P2_v & 0xcf;            //close u21 u22 data bus
    MCU_A0 = 0;
    P74OUT = 0x3f;               //p7 is open-d,p6 p5 p4 is push-pull
    P7 = 0xff;
   P4 = P4 & 0xbf;      //    /rd=0
    P4 = P4 & 0xbf;      //delay
    temp_v = P7;
    P4 = P4 | 0x40;   //    /rd=1
    P4 = P4 | 0x40;   // delay
    P2 = P2_v | 0x30;            //reset P2 volue
//  MCU_D12CS = 1;
    return (temp_v);
}
```

附录 2　C8051F040/1 的 CAN 接口操作程序

```
//-----------------------------------------------------------------------------
// CAN1.c
//-----------------------------------------------------------------------------
//
//
// DEVICE: C8051F040
//
// AUTHOR: LS
//
// TOOLS:    Keil C-compiler and Silicon Labs IDE
//
//
// CAN1.c and CAN2.c are a simple example of configuring a CAN network to
// transmit and receive data on a CAN network, and how to move information to
// and from CAN RAM message objects.    Each C8051F040-TB CAN node is configured
// to send a message when it's P3.7 button is depressed/released, with a 0x11
// to indicate the button is pushed, and 0x00 when released. Each node also has
// a message object configured to receive messages. The C8051 tests the
// received data and will turn on/off the target board's LED. When one target
// is loaded with CAN2.c and the other is loaded with CAN1.c, one target
// board's push-button will control the other target board's LED, establishing
// a simple control link via the CAN bus and can be observed directly on the
// target boards.
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////

#include <c8051f040.h>                                  // SFR declarations

// CAN Protocol Register Index for CAN0ADR, from TABLE 18.1 of the C8051F040
// datasheet
////////////////////////////////////////////////////////////////////////////////
#define CANCTRL                 0x00                    //Control Register
#define CANSTAT                 0x01                    //Status register
#define ERRCNT                  0x02                    //Error Counter Register
#define BITREG                  0x03                    //Bit Timing Register
#define INTREG                  0x04                    //Interrupt Low Byte Register
#define CANTSTR                 0x05                    //Test register
#define BRPEXT                  0x06                    //BRP Extension             Register
////////////////////////////////////////////////////////////////////////////////
//IF1 Interface Registers
////////////////////////////////////////////////////////////////////////////////
#define IF1CMDRQST              0x08                    //IF1 Command Rest          Register
#define IF1CMDMSK               0x09                    //IF1 Command Mask          Register
#define IF1MSK1                 0x0A                    //IF1 Mask1                 Register
#define IF1MSK2                 0x0B                    //IF1 Mask2                 Register
#define IF1ARB1                 0x0C                    //IF1 Arbitration 1         Register
#define IF1ARB2                 0x0D                    //IF1 Arbitration 2         Register
```

```
#define IF1MSGC                 0x0E                 //IF1 Message Control     Register
#define IF1DATA1                0x0F                 //IF1 Data A1             Register
#define IF1DATA2                0x10                 //IF1 Data A2             Register
#define IF1DATB1                0x11                 //IF1 Data B1             Register
#define IF1DATB2                0x12                 //IF1 Data B2             Register
////////////////////////////////////////////////////////////////////////////
//IF2 Interface Registers
////////////////////////////////////////////////////////////////////////////
#define IF2CMDRQST              0x20                 //IF2 Command Rest        Register
#define IF2CMDMSK               0x21                 //IF2 Command Mask        Register
#define IF2MSK1                 0x22                 //IF2 Mask1               Register
#define IF2MSK2                 0x23                 //IF2 Mask2               Register
#define IF2ARB1                 0x24                 //IF2 Arbitration 1       Register
#define IF2ARB2                 0x25                 //IF2 Arbitration 2       Register
#define IF2MSGC                 0x26                 //IF2 Message Control     Register
#define IF2DATA1                0x27                 //IF2 Data A1             Register
#define IF2DATA2                0x28                 //IF2 Data A2             Register
#define IF2DATB1                0x29                 //IF2 Data B1             Register
#define IF2DATB2                0x2A                 //IF2 Data B2             Register
////////////////////////////////////////////////////////////////////////////
//Message Handler Registers
////////////////////////////////////////////////////////////////////////////
#define TRANSREQ1               0x40                 //Transmission Rest1 Register
#define TRANSREQ2               0x41                 //Transmission Rest2 Register

#define NEWDAT1                 0x48                 //New Data 1              Register
#define NEWDAT2                 0x49                 //New Data 2              Register

#define INTPEND1                0x50                 //Interrupt Pending 1     Register
#define INTPEND2                0x51                 //Interrupt Pending 2     Register

#define MSGVAL1                 0x58                 //Message Valid 1         Register
#define MSGVAL2                 0x59                 //Message Valid 2         Register

////////////////////////////////////////////////////////////////////////////
//Global Variables
////////////////////////////////////////////////////////////////////////////
char MsgNum;
char status;
int i;
int MOTwoIndex = 0;
int MOOneIndex = 0;
int StatusCopy;
int RXbuffer [4];
int TXbuffer [8];
int MsgIntNum;
int Temperature;
sbit BUTTON = P3^7;
sbit LED = P1^6;
sfr16 CAN0DAT = 0xD8;



////////////////////////////////////////////////////////////////////////////
// Function PROTOTYPES
////////////////////////////////////////////////////////////////////////////

// Initialize Message Object
void clear_msg_objects (void);
```

```
void init_msg_object_TX (char MsgNum);
void init_msg_object_RX (char MsgNum);
void start_CAN (void);
void transmit_turn_LED_ON (char MsgNum);
void transmit_turn_LED_OFF (char MsgNum);
void receive_data (char MsgNum);
void external_osc (void);
void config_IO (void);
void flash_LED (void);
void test_reg_write (char test);
void stop_CAN (void);


//////////////////////////////////////////////////////////////////////////////
// MAIN Routine
//////////////////////////////////////////////////////////////////////////////
void main (void) {

   // disable watchdog timer
   WDTCN = 0xde;
   WDTCN = 0xad;

   //configure Port I/O
   config_IO();

   // switch to external oscillator
   external_osc();


//////////////////////////////////////////////////////////////////////////////
// Configure CAN communications
//
// IF1 used for procedures calles by main program
// IF2 used for interrupt service procedure receive_data
//
// Message Object assignments:
//    0x02: Used to transmit commands to toggle its LED, arbitration number 1
//
//////////////////////////////////////////////////////////////////////////////

   // Clear CAN RAM
   clear_msg_objects();

   // Initialize message object to transmit data
   init_msg_object_TX (0x02);

   // Initialize message object to receive data
   init_msg_object_RX (0x01);

   // Enable CAN interrupts in CIP-51
   EIE2 = 0x20;

   //Function call to start CAN
   start_CAN();

   //Global enable 8051 interrupts
   EA = 1;

   //Loop and wait for interrupts
```

```
   while (1)
      {
         if (BUTTON == 0){
            while (BUTTON == 0){}
            transmit_turn_LED_OFF(0x02);}
         else {
            while (BUTTON == 1){}
            transmit_turn_LED_ON(0x02);}
      }
}


//////////////////////////////////////////////////////////////////////////////
// Set up C8051F040
//////////////////////////////////////////////////////////////////////////////

// Switch to external oscillator
void external_osc (void)
{
   int n;                                  // local variable used in delay FOR loop.
   SFRPAGE = CONFIG_PAGE;                  // switch to config page to config oscillator
   OSCXCN    = 0x77;                       // start external oscillator; 22.1 MHz Crystal
                                           // system clock is 22.1 MHz / 2 = 11.05 MHz
   for (n=0;n<255;n++);                    // delay about 1ms
   while ((OSCXCN & 0x80) == 0); // wait for oscillator to stabilize
   CLKSEL |= 0x01;                         // switch to external oscillator
}

void config_IO (void)
{
   SFRPAGE    = CONFIG_PAGE;               //Port SFR's on Configuration page
   XBR3       = 0x80;         // Configure CAN TX pin (CTX) as push-pull digital output
   P1MDOUT |= 0x40;           // Configure P1.6 as push-pull to drive LED
   XBR2       = 0x40;         // Enable Crossbar/low ports
}

//////////////////////////////////////////////////////////////////////////////
//CAN Functions
//////////////////////////////////////////////////////////////////////////////


//Clear Message Objects
void clear_msg_objects (void)
{
   SFRPAGE    = CAN0_PAGE;
   CAN0ADR    = IF1CMDMSK;        // Point to Command Mask Register 1
   CAN0DATL = 0xFF;               // Set direction to WRITE all IF registers to Msg Obj
   for (i=1;i<33;i++)
      {
         CAN0ADR = IF1CMDRQST; // Write blank (reset) IF registers to each msg obj
         CAN0DATL = i;
      }
}

//Initialize Message Object for RX
void init_msg_object_RX (char MsgNum)
{
   SFRPAGE    = CAN0_PAGE;
   CAN0ADR    = IF1CMDMSK;    // Point to Command Mask 1
```

```
   CAN0DAT    = 0x00B8;          // Set to WRITE, and alter all Msg Obj except ID MASK
                                 // and data bits
   CAN0ADR    = IF1ARB1;         // Point to arbitration1 register
   CAN0DAT    = 0x0000;          // Set arbitration1 ID to "0"
   CAN0DAT    = 0x8004;          // Arb2 high byte:Set MsgVal bit, no extended ID,
                                 // Dir = RECEIVE
   CAN0DAT    = 0x0480;          // Msg Cntrl: set RXIE, remote frame function disabled
   CAN0ADR    = IF1CMDRQST; // Point to Command Request reg.
   CAN0DATL = MsgNum;            // Select Msg Obj passed into function parameter list
                                 // --initiates write to Msg Obj
   // 3-6 CAN clock cycles to move IF register contents to the Msg Obj in CAN RAM
}

//Initialize Message Object for TX
void init_msg_object_TX (char MsgNum)
{
   SFRPAGE = CAN0_PAGE;
   CAN0ADR = IF1CMDMSK;    // Point to Command Mask 1
   CAN0DAT = 0x00B2;          // Set to WRITE, & alter all Msg Obj except ID MASK bits
   CAN0ADR = IF1ARB1;         // Point to arbitration1 register
   CAN0DAT = 0x0000;          // Set arbitration1 ID to highest priority
   CAN0DAT = 0xA000;          // Autoincrement to Arb2 high byte:
                              // Set MsgVal bit, no extended ID, Dir = WRITE
   CAN0DAT = 0x0081;          // Msg Cntrl: DLC = 1, remote frame function not enabled
   CAN0ADR = IF1CMDRQST; // Point to Command Request reg.
   CAN0DAT = MsgNum;          // Select Msg Obj passed into function parameter list
                              // --initiates write to Msg Obj
   // 3-6 CAN clock cycles to move IF reg contents to the Msg Obj in CAN RAM.
}

//Start CAN
void start_CAN (void)
{
   /* Calculation of the CAN bit timing :

   System clock               f_sys = 22.1184 MHz/2 = 11.0592 MHz.
   System clock period t_sys = 1/f_sys = 90.422454 ns.
   CAN time quantum           tq = t_sys (at BRP = 0)

   Desired bit rate is 1 MBit/s, desired bit time is 1000 ns.
   Actual bit time = 11 tq = 996.65ns ~ 1000 ns
   Actual bit rate is 1.005381818 MBit/s = Desired bit rate+0.5381%

   CAN bus length = 10 m, with 5 ns/m signal delay time.
   Propagation delay time : 2*(transceiver loop delay + bus line delay) = 400 ns
   (maximum loop delay between CAN nodes)

   Prop_Seg = 5 tq = 452 ns ( >= 400 ns).
   Sync_Seg = 1 tq

   Phase_seg1 + Phase_Seg2 = (11-6) tq = 5 tq
   Phase_seg1 <= Phase_Seg2,    =>    Phase_seg1 = 2 tq and Phase_Seg2 = 3 tq
   SJW = (min(Phase_Seg1, 4) tq = 2 tq

   TSEG1 = (Prop_Seg + Phase_Seg1 - 1) = 6
   TSEG2 = (Phase_Seg2 - 1)                    = 2
   SJW_p = (SJW - 1)                           = 1

   Bit Timing Register = BRP + SJW_p*0x0040 = TSEG1*0x0100 + TSEG2*0x1000 = 2640
```

```
   Clock tolerance df :

   A: df < min(Phase_Seg1, Phase_Seg2) / (2 * (13*bit_time - Phase_Seg2))
   B: df < SJW / (20 * bit_time)

   A: df < 2/(2*(13*11-3)) = 1/(141-3) = 1/138 = 0.7246%
   B: df < 2/(20*11)                   = 1/110 = 0.9091%

   Actual clock tolerance is 0.7246% - 0.5381% = 0.1865% (no problem for quartz)
   */

   SFRPAGE    = CAN0_PAGE;
   CAN0CN    |= 0x41;             // Configuration Change Enable CCE and INIT
   CAN0ADR    = BITREG     ;    // Point to Bit Timing register
   CAN0DAT    = 0x2640;          // see above

   CAN0ADR    = IF1CMDMSK;    // Point to Command Mask 1
   CAN0DAT    = 0x0087;          // Config for TX : WRITE to CAN RAM, write data bytes,
                                 // set TXrqst/NewDat, clr IntPnd

   // RX-IF2 operation may interrupt TX-IF1 operation
   CAN0ADR    = IF2CMDMSK;    // Point to Command Mask 2
   CAN0DATL = 0x1F;             // Config for RX : READ CAN RAM, read data bytes,
                                // clr NewDat and IntPnd
   CAN0CN    |= 0x06;            // Global Int. Enable IE and SIE
   CAN0CN    &= ~0x41;           // Clear CCE and INIT bits, starts CAN state machine
}

//Transmit CAN frame to turn other node's LED ON
void transmit_turn_LED_ON (char MsgNum)
{
   SFRPAGE    = CAN0_PAGE;    // IF1 already set up for TX
   CAN0ADR    = IF1CMDMSK;    // Point to Command Mask 1
   CAN0DAT    = 0x0087;          // Config to WRITE to CAN RAM, write data bytes,
                                 // set TXrqst/NewDat, Clr IntPnd
   CAN0ADR    = IF1DATA1;      // Point to 1st byte of Data Field
   CAN0DATL = 0x11;             // Ones signals to turn LED's light ON in data A1 field
   CAN0ADR    = IF1CMDRQST; // Point to Command Request Reg.
   CAN0DATL = MsgNum;          // Move new data for TX to Msg Obj "MsgNum"
}

//Transmit CAN Frame to turn other node's LED OFF
void transmit_turn_LED_OFF (char MsgNum)
{
   SFRPAGE    = CAN0_PAGE;    // IF1 already set up for TX
   CAN0ADR    = IF1DATA1;      // Point to 1st byte of Data Field
   CAN0DATL = 0x00;             // Zero signals to turn LED's light ON in Data A1 field
   CAN0ADR    = IF1CMDRQST; // Point to Command Request Reg.
   CAN0DATL = MsgNum;          // Move new data for TX to Msg Obj "MsgNum"
}


// Receive Data from the IF2 buffer
void receive_data (char MsgNum)
{
   char virtual_button;
   SFRPAGE    = CAN0_PAGE; // IF1 already set up for RX
   CAN0ADR    = IF2CMDRQST;// Point to Command Request Reg.
```

```
   CAN0DATL = MsgNum;       // Move new data for RX from Msg Obj "MsgNum"
                            // Move new data to a
   CAN0ADR    = IF2DATA1;   // Point to 1st byte of Data Field

   virtual_button = CAN0DATL;
   if (virtual_button == 0x11)     //Ones is signal from other node to turn LED ON
      LED = 1;
   else    LED = 0;                         //Otherwise turn LED OFF (message was one's)
}

////////////////////////////////////////////////////////////////////////////
//Interrupt Service Routine
////////////////////////////////////////////////////////////////////////////
void ISRname (void) interrupt 19
{
   status = CAN0STA;
   if ((status&0x10) != 0)
      {                                    // RxOk is set, interrupt caused by reception
         CAN0STA = (CAN0STA&0xEF)|0x07;          // Reset RxOk, set LEC to NoChange
         /* read message number from CAN INTREG */
         receive_data (0x01);                 // Up to now, we have only one RX message
      }
   if ((status&0x08) != 0)
      {                                    // TxOk is set, interrupt caused by transmision
         CAN0STA = (CAN0STA&0xF7)|0x07;          // Reset TxOk, set LEC to NoChange
      }
   if (((status&0x07) != 0)&&((status&0x07) != 7))
      {                                    // Error interrupt, LEC changed
         /* error handling ? */
         CAN0STA = CAN0STA|0x07;                 // Set LEC to NoChange
      }
}
```

附录 3　压力开关测试仪测试模块程序

```
//=====================================================
//                    测试界面程序
//=====================================================
’2002-11  测试
Option Explicit
Dim Modulus, zero As Single
Dim SoundCri As Integer
Dim BSoundSwitch As Boolean
Const RowCount = 12: Const ColCount = 12
Dim ArraySwitch(0 To 5) As Integer
Dim SoundTime As Integer
Dim TempTsL, TempTsR As Long
Dim TsL(1 To 9) As Long
Dim TsR(1 To 9) As Long
Dim Aj(1 To 6) As Single
Dim Aa(1 To 6) As Single
Dim BlnEligible(1 To 8) As Integer
Dim BlnEligibleL(1 To 8) As Integer
Dim BlnEligibleR(1 To 8) As Integer
Dim BlnAllEligibleL As Boolean
Dim BlnAllEligibleR As Boolean
Dim BSoundEligibleL As Boolean
Dim BSoundEligibleR As Boolean
Dim SngTestValue(1 To 9) As Single
Dim SngTestValueL(1 To 9) As Single
Dim SngTestValueR(1 To 9) As Single
Dim BlnRetry As Boolean
Dim BlnTcxt As Boolean
Dim Flag, FlagL, FlagR As Boolean
Dim YS, IntYs As Integer
Const Pi = 3.14159265358979
Const MinMark = 0.4
Dim ColockRadius, CenterRadius, X0, Y0, X1, Y1, X2, Y2, Cx0, Cy0, Cx1, Cy1, Cx2, Cy2, ShowValue, Test
As Single

Private Sub CmdStart_Click()
    TimRead.Enabled = False
    TimReadJJ.Enabled = False
    BlnRetry = True
    Timer1.Enabled = True
End Sub


Private Sub ComdEnd_Click()
    BlnTcxt = True
    TimerTcxt.Enabled = True

End Sub

Private Sub Comzero_Click()
```

```
    Dim number As Long
    Dim sum As Long
    sum = 0
    For number = 0 To 7
        sum = sum + CInt(GetMN1)
    Next number
    zero = sum / number
    TxtTarget.Text = 0
    On Error GoTo Err1
        Conn.Execute "update series set    zero=" & zero & "" _
            & " where serid=" & CurSerId & ""
        MsgBox "回零完成", vbOKOnly, "提示信息"
    Exit Sub
        Unload Me
Err1:
        MsgBox Err.Description, vbOKOnly + vbExclamation, "错误提示！"
End Sub

Private Sub Form_Activate()
    DrawColock
    ClrK1
    ClrK1r
    ClrPs7
    SetPs0
    SetPs1
    SetPs2
    SetPs3
    SetPs4
    SetPs5
    SetPs6
    TxtTarget.Text = FormatNumber((CInt(GetMN1) - zero) * Modulus, 3)
    TimMn1.Enabled = True
    Timer2.Enabled = True
    Timer1.Enabled = False
    TimJT.Enabled = True
    TimReadJJ.Enabled = False
    TimerButten = True
    TimerTcxt.Enabled = False
End Sub

Private Sub Form_Load()
    'FrmCent Me
    LabelName.Caption = CurSerName
    LabelName.Tag = CurSerId
    Timer1.Enabled = False
    WriteToGrid
    StandardValue
    DrawSwitch 3, 1
    DrawSwitch 3, 9
    BlnRetry = False
    BlnTcxt = False
    Flag = False
    ArraySwitch(0) = 3
    ArraySwitch(1) = 2
    ArraySwitch(2) = 1
    ArraySwitch(3) = 0
    ArraySwitch(4) = 2
    ArraySwitch(5) = 3
```

```
End Sub
Private Sub DrawColock()
    Dim n, i As Integer
    Picture1.Height = 4000
    Picture1.Width = 4000
    Picture1.ScaleHeight = -100
    Picture1.ScaleTop = 50
    Picture1.ScaleWidth = 100
    Picture1.ScaleLeft = -50
    Picture1.DrawWidth = 2
    ColockRadius = (Picture1.ScaleWidth - 5) / 2
    Picture1.Circle (0, 0), ColockRadius, 0, Pi * 105 / 60, Pi * 75 / 60
    i = 0
    Picture1.DrawWidth = 1
    For n = 75 To -15 Step -1
        X0 = ColockRadius * Cos(n * Pi / 60)
        Y0 = ColockRadius * Sin(n * Pi / 60)
        X1 = (ColockRadius - 4) * Cos(n * Pi / 60)
        Y1 = (ColockRadius - 4) * Sin(n * Pi / 60)
        Picture1.Line (X0, Y0)-(X1, Y1)
        If (n Mod 5 = 0) And (n Mod 10 <> 0) Then
            X2 = (ColockRadius - 6) * Cos(n * Pi / 60)
            Y2 = (ColockRadius - 6) * Sin(n * Pi / 60)
            Picture1.Line (X0, Y0)-(X2, Y2)
            If n < 25 Then
                Picture1.CurrentX = X2 - 6
                Picture1.CurrentY = Y2 + 1
            Else
                Picture1.CurrentX = X2 + 1
                Picture1.CurrentY = Y2
            End If
            Picture1.Print Format(MinMark * i, "##0.0")
            i = i + 1
        End If
    Next n
    CenterRadius = 1.5
    Picture1.DrawWidth = 1
    Picture1.Circle (0, 0), CenterRadius, vbBlue
End Sub

Private Sub WriteToGrid()
    Dim Intx As Integer
    Dim Inty As Integer
    With FlexGrid
        .Enabled = False
        .Cols = ColCount
        .Rows = RowCount
        .MergeCells = flexMergeFree
        .Row = 0
        For Intx = 1 To ColCount - 1
            .Col = Intx
            .MergeRow(0) = True
            If Intx <= 2 Then
                .Text = "          实测值(左)"
            Else
                If Intx = 3 Then
                    .Text = ""
                Else
                    If Intx <= 7 Then
```

```
                    .Text = "                标准值与公差"
                Else
                    If Intx = 8 Then
                        .Text = ""
                    Else
                        If Intx <= 10 Then
                            .Text = "        实测值(右)"
                        End If
                    End If
                End If
            End If
        End If
    Next Intx

    .Row = 1
    For Intx = 1 To ColCount - 1
        .Col = Intx
        .MergeRow(1) = True
        If Intx = 1 Then
            .Text = "压力值"
        Else
            If Intx = 2 Then
                .Text = "时    差"
            Else
                If Intx = 3 Then
                    .Text = ""
                Else
                    If Intx <= 5 Then
                        .Text = "     标准压力值"
                    Else
                        If Intx = 6 Then
                            .Text = "公    差"
                        Else
                            If Intx = 7 Then
                                .Text = "时    差"
                            Else
                                If Intx = 8 Then
                                    .Text = ""
                                Else
                                    If Intx = 9 Then
                                        .Text = "压力值"
                                    Else
                                        If Intx = 10 Then
                                            .Text = "时    差"
                                        End If
                                    End If
                                End If
                            End If
                        End If
                    End If
                End If
            End If
        End If
    Next Intx
    .Col = 0
    For Inty = 1 To RowCount - 1
        .Row = Inty
        .MergeCol(0) = True
```

```
                If Inty <= 1 Then
                    .Text = ""
                Else
                    If Inty <= 4 Then
                        .Text = "高　压"
                    Else
                        If Inty = 5 Then
                            .Text = ""
                        Else
                            If Inty <= 7 Then
                                .Text = "中压"
                            Else
                                If Inty = 8 Then
                                    .Text = ""
                                Else
                                    .Text = "低　压"
                                End If
                            End If
                        End If
                    End If
                End If
            Next Inty
            .Col = 4: .Row = 2: .Text = "OFF"
            .Col = 4: .Row = 3: .Text = "ON"
            .Col = 4: .Row = 4: .Text = "DIFF"
            .Col = 4: .Row = 7: .Text = "OFF"
            .Col = 4: .Row = 6: .Text = "ON"
            .Col = 4: .Row = 9: .Text = "ON"
            .Col = 4: .Row = 10: .Text = "OFF"
            .Col = 4: .Row = 11: .Text = "DIFF"
            .ColWidth(0) = 500
            .ColWidth(1) = 1500
            .ColWidth(2) = 1500
            .ColWidth(3) = 10
            .ColWidth(4) = 800
            .ColWidth(5) = 1200
            .ColWidth(6) = 1200
            .ColWidth(7) = 1200
            .ColWidth(8) = 10
            .ColWidth(9) = 1500
            .ColWidth(10) = 1500
            .ColWidth(11) = 2
            .RowHeight(5) = 10
            .RowHeight(8) = 10
        End With
End Sub

Private Sub InitTestValue(FColor As Long, BColor As Long, Value As Variant, RowX As Integer, ColY As Integer)
        Dim IntCol As Integer
        With FlexGrid
            .Row = RowX
            .Col = ColY
            .Text = Value
            .CellForeColor = FColor
            .CellBackColor = BColor
            .CellFontBold = True
        End With
```

```
End Sub

Private Sub Timer0_Timer()
    Timer1.Enabled = False
    TimRead.Enabled = False
    TimReadJJ.Enabled = False
    TimerSound.Enabled = False
    TimerButten.Enabled = False
    TimJT.Enabled = False
    ClrPs0
    SetPs7

    ClrPs4
    ClrPs5
    ClrPs6
    SetPs1
    SetPs2
    SetPs3
    TimerTcxt.Enabled = True
    Timer0.Enabled = False
End Sub

Private Sub Timer1_Timer()
    Dim i As Integer
    Dim j As Double
    With FlexGrid
        For i = 2 To 11
            InitTestValue &H80000002, &HFAEBD6, " ", i, 1
            InitTestValue &H80000002, &HFAEBD6, " ", i, 2
            InitTestValue &H80000002, &HFAEBD6, " ", i, 9
            InitTestValue &H80000002, &HFAEBD6, " ", i, 10
        Next i
    End With
    LabTestNum.Caption = "0"
    LabTestNumr.Caption = "0"
    If BlnRetry Then
        TxtBh.Text = TxtBh.Text
        TxtBhr.Text = TxtBhr.Text
    Else
        TxtBh.Text = CreateProductID
        TxtBhr.Text = Format(Right(TxtBh.Text + 1, PserBitCount), "00000000")
    End If
    For i = 1 To 8
        BlnEligible(i) = True
    Next i
    BlnAllEligibleL = True
    BlnAllEligibleR = True
    BSoundEligibleL = True
    BSoundEligibleR = True
    For i = 1 To 9
        SngTestValue(i) = -1
        SngTestValueL(i) = -1
        SngTestValueR(i) = -1
    Next i
    For i = 1 To 9
        TsL(i) = -1
        TsR(i) = -1
    Next i
        ClrPs0
```

```
            SetPs1
            SetPs7
            SetPs2
            SetPs3
            SetPs4
            SetPs5
            SetPs6

            ClrK1
            ClrK1r
            IntYs = 0
            TimYS.Enabled = True
            Timer1.Enabled = False
End Sub


Private Sub Timer3_Timer()

End Sub



'Private Sub Timer2_Timer()
'       TxtTarget.Text = FormatNumber((CInt(GetMN1) - zero) * Modulus, 3)
'End Sub


Private Sub TimerButten_Timer()
      If GetKs = 2 Then
                  BlnRetry = False
                  TimerButten.Enabled = False
                  Timer1.Enabled = True
      End If
End Sub

Private Sub TimerSound_Timer()
      SoundTime = SoundTime + 1
      If SoundTime = 10 Then
            TempTsL = GetTs
            TempTsR = GetTsr
            TimerSound.Enabled = False
      End If
End Sub


Private Sub TimerTcxt_Timer()
      If TxtTarget.Text < 0.05 Then
            ClrPs7
            SetPs0

            SetPs1
            SetPs2
            SetPs3
            SetPs4
            SetPs5
            SetPs6
            TimJT.Enabled = True
            TimerButten.Enabled = True
            TimerTcxt.Enabled = False
```

```
            If BlnTcxt Then Unload Me
        Else
            ClrPs0
            SetPs7

            ClrPs5
            SetPs1
            SetPs2
            SetPs3
            SetPs4
            SetPs6
        End If
End Sub

Private Sub TimJT_Timer()
        If GetKs = 1 Then
        Timer1.Enabled = False
        TimRead.Enabled = False
        TimReadJJ.Enabled = False
        TimerSound.Enabled = False
        TimerButten.Enabled = False
        TimJT.Enabled = False
        ClrPs0
        SetPs7

        ClrPs4
        ClrPs5
        ClrPs6
        SetPs1
        SetPs2
        SetPs3
        TimerTcxt.Enabled = True
        End If
End Sub

Private Sub TimMn1_Timer()
        TxtTarget.Text = FormatNumber((CInt(GetMN1) - zero) * Modulus, 3)
        Test = 75 - (CInt(GetMN1) - zero) * Modulus / MinMark * 10
        If TxtTarget.Text > 3.6 Then
            Test = 3.6
        End If
'       If TxtTarget.Text < 0 Then
'           TxtTarget.Text = 0
'       End If

        Call MoveClock(Test)
        'TimMn1.Enabled = False
End Sub
Private Sub MoveClock(ShowValue As Single)
        Cx1 = (CenterRadius - 0.5) * Cos(ShowValue * Pi / 60 + Pi / 2)
        Cy1 = (CenterRadius - 0.5) * Sin(ShowValue * Pi / 60 + Pi / 2)
        Cx2 = (CenterRadius - 0.5) * Cos(ShowValue * Pi / 60 - Pi / 2)
        Cy2 = (CenterRadius - 0.5) * Sin(ShowValue * Pi / 60 - Pi / 2)
        Cx0 = (ColockRadius - 12) * Cos(ShowValue * Pi / 60)
        Cy0 = (ColockRadius - 12) * Sin(ShowValue * Pi / 60)
        LinePoint(0).X1 = Cx0: LinePoint(0).Y1 = Cy0: LinePoint(0).X2 = 0: LinePoint(0).Y2 = 0
        LinePoint(1).X1 = Cx0: LinePoint(1).Y1 = Cy0: LinePoint(1).X2 = Cx1: LinePoint(1).Y2 = Cy1
        LinePoint(2).X1 = Cx0: LinePoint(2).Y1 = Cy0: LinePoint(2).X2 = Cx2: LinePoint(2).Y2 = Cy2
End Sub
```

```
Private Sub TimRead_Timer()

    If Aj(1) < TxtTarget.Text And TxtTarget.Text < Aa(1) Then
                ClrPs0
                SetPs7
            ClrPs1
            SetPs2
            SetPs3
            SetPs4
            SetPs5
            SetPs6
            If Aj(1) * 1.02 < TxtTarget.Text And TxtTarget.Text < Aa(1) * 0.98 Then
                State 1, 2
                State 9, 2
            End If

    ElseIf Aa(1) < TxtTarget.Text And TxtTarget.Text < Aj(2) Then
                ClrPs0
                SetPs7
            SetPs1
            SetPs2
            ClrPs3
            SetPs4
            SetPs5
            SetPs6
            ClrK1
            ClrK1r

    ElseIf Aj(2) < TxtTarget.Text And TxtTarget.Text < Aa(2) Then
                ClrPs0
                SetPs7
            ClrPs1
            SetPs2
            SetPs3
            SetPs4
            SetPs5
            SetPs6
            If Aj(2) * 1.02 < TxtTarget.Text And TxtTarget.Text < Aa(2) * 0.98 Then
                State 1, 4
                State 9, 4
            End If

    ElseIf Aa(2) < TxtTarget.Text And TxtTarget.Text < Aj(3) Then
                ClrPs0
                SetPs7
            ClrPs1
            ClrPs2
            ClrPs3
            SetPs4
            SetPs5
            SetPs6
            ClrK1
            ClrK1r

    ElseIf Aj(3) < TxtTarget.Text And TxtTarget.Text < Aa(3) Then
                ClrPs0
                SetPs7
            ClrPs1
```

```
                SetPs2
                SetPs3
                SetPs4
                SetPs5
                SetPs6
                If Aj(3) * 1.02 < TxtTarget.Text And TxtTarget.Text < Aa(3) * 1 Then
                    State 1, 6
                    State 9, 6
                End If

    ElseIf TxtTarget.Text > Aa(3) Or TxtTarget.Text > 3.5 Then
                    ClrPs0
                    SetPs7
                SetPs1
                SetPs2
                SetPs3
                ClrPs4
                ClrPs5
                SetPs6
                ClrK1
                ClrK1r
                TimRead.Enabled = False
                TimReadJJ.Enabled = True
    Else
                ClrK1
                ClrK1r
    End If
End Sub

Private Sub State(Direct As Integer, Stadue As Integer)
    Dim Mn2, TempK1 As Single
    If Direct = 1 Then
            Mn2 = GetMN2
            TempK1 = GetK1
    Else
            Mn2 = GetMN2r
            TempK1 = GetK1r
    End If

    Select Case Stadue
        Case 2
            If TempK1 = 1 Then
                    SngTestValue(8) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(8), 3)
                    DrawSwitch 2, Direct
                    Music 8, Direct
                    CompEligible 0, Direct
                    Flag = True
            End If
        Case 4
            If TempK1 = 1 Then
                        SngTestValue(5) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(5), 3)
                        DrawSwitch 1, Direct
                        Music 5, Direct
                        CompEligible 1, Direct
                        Flag = True
            End If
        Case 6
            If TempK1 = 1 Then
                    SngTestValue(1) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(1), 3)
```

```
                    DrawSwitch 0, Direct
                    Music 1, Direct
                    CompEligible 2, Direct
                    Flag = True
                End If
            Case 8
                If TempK1 = 1 Then
                    SngTestValue(2) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(2), 3)
                    DrawSwitch 1, Direct
                    'Music 2, Direct
                    CompEligible 3, Direct
                    Flag = True
                End If
            Case 10
                If TempK1 = 1 Then
                    SngTestValue(4) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(4), 3)
                    DrawSwitch 2, Direct
                    CompEligible 4, Direct
                    Flag = True
                End If
            Case 12
                If TempK1 = 1 Then
                    SngTestValue(7) = FormatNumber((CInt(Mn2) - zero) * Modulus * Para(7), 3)
                    DrawSwitch 3, Direct
                    'Music 7, Direct
                    CompEligible 5, Direct
                    Flag = True
                End If
        End Select

        If Flag Then
            If Direct = 1 Then
                ClrK1
            Else
                ClrK1r
            End If
            Flag = False
        End If
End Sub
Private Sub UpDateRecL()
        Dim TestDate As String
        Dim ProductnumL As String
        Dim BlnE As Boolean
        If CInt(LabTestNum.Caption) >= 6 Then
            BlnE = BlnAllEligibleL And True
        Else
            BlnE = BlnAllEligibleL And False
        End If

        If BLowSwitch And BHightSwitch Then
            BlnAllEligibleL = BlnAllEligibleL And BlnEligibleL(8) And BlnEligibleL(2) And BlnEligibleL(3)
And BlnEligibleL(4) And BlnEligibleL(6) And BlnEligibleL(7)
        Else
            BlnAllEligibleL = BlnAllEligibleL And BlnEligibleL(1) And BlnEligibleL(2) And BlnEligibleL(3)
And BlnEligibleL(5) And BlnEligibleL(6) And BlnEligibleL(7)
        End If
            '*****************************20030902
        If BlnAllEligibleL Then
            TxtBh.BackColor = vbGreen
```

```
    Else
        TxtBh.BackColor = vbRed
    End If
        '*****************************20030902
    TestDate = "#" & Date & "#"
    ProductnumL = CurSerName & TxtBh.Text
    On Error GoTo Err1
        Conn.Execute "update product set serID=" & CurSerId & "' where productnum=" & ProductnumL
& "'"
        Conn.Execute "update product set hoff=" & SngTestValueL(1) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set hon=" & SngTestValueL(2) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set hdiff=" & SngTestValueL(3) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set moff=" & SngTestValueL(4) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set mon=" & SngTestValueL(5) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set mdiff=" & SngTestValueL(6) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set loff=" & SngTestValueL(7) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set lon=" & SngTestValueL(8) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set ldiff=" & SngTestValueL(9) & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set testdate=" & TestDate & " where productnum=" & ProductnumL
& "'"
        Conn.Execute "update product set eligible=" & BlnAllEligibleL & " where productnum=" &
ProductnumL & "'"
        Conn.Execute "update product set hoffts=" & TsL(1) & " where productnum=" & ProductnumL &
"'"
        Conn.Execute "update product set honts=" & TsL(2) & " where productnum=" & ProductnumL & "'"
        Conn.Execute "update product set hdiffts=" & TsL(3) & " where productnum=" & ProductnumL &
"'"
        Conn.Execute "update product set moffts=" & TsL(4) & " where productnum=" & ProductnumL &
"'"
        Conn.Execute "update product set monts=" & TsL(5) & " where productnum=" & ProductnumL &
"'"
        Conn.Execute "update product set mdiffts=" & TsL(6) & " where productnum=" & ProductnumL &
"'"
        Conn.Execute "update product set loffts=" & TsL(7) & " where productnum=" & ProductnumL & "'"
        Conn.Execute "update product set lonts=" & TsL(8) & " where productnum=" & ProductnumL & "'"
        Conn.Execute "update product set ldiffts=" & TsL(9) & " where productnum=" & ProductnumL &
"'"
    Exit Sub
Err1:
    MsgBox Err.Description, vbOKOnly + vbExclamation, "错误提示！"
End Sub
Private Sub UpDateRecR()
    Dim TestDate As String
    Dim ProductnumR As String
    Dim BlnE As Boolean
    If CInt(LabTestNumr.Caption) >= 6 Then
        BlnE = BlnAllEligibleR And True
    Else
        BlnE = BlnAllEligibleR And False
    End If
```

```
    If BLowSwitch And BHightSwitch Then
        BlnAllEligibleR = BlnAllEligibleR And BlnEligibleR(8) And BlnEligibleR(2) And BlnEligibleR(3)
And BlnEligibleR(4) And BlnEligibleR(6) And BlnEligibleR(7)
    Else
        BlnAllEligibleR = BlnAllEligibleR And BlnEligibleR(1) And BlnEligibleR(2) And BlnEligibleR(3)
And BlnEligibleR(5) And BlnEligibleR(6) And BlnEligibleR(7)
    End If
    '*****************************20030902
    If BlnAllEligibleR Then
        TxtBhr.BackColor = vbGreen
    Else
        TxtBhr.BackColor = vbRed
    End If
    '*****************************20030902
    TestDate = "#" & Date & "#"
    ProductnumR = CurSerName & TxtBhr.Text
    On Error GoTo Err1
        Conn.Execute "update product set serID= '" & CurSerId & "' where productnum='" & ProductnumR
& "'"
        Conn.Execute "update product set hoff=" & SngTestValueR(1) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set hon=" & SngTestValueR(2) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set hdiff=" & SngTestValueR(3) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set moff=" & SngTestValueR(4) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set mon=" & SngTestValueR(5) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set mdiff=" & SngTestValueR(6) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set loff=" & SngTestValueR(7) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set lon=" & SngTestValueR(8) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set ldiff=" & SngTestValueR(9) & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set testdate=" & TestDate & " where productnum='" & ProductnumR
& "'"
        Conn.Execute "update product set eligible=" & BlnAllEligibleR & " where productnum='" &
ProductnumR & "'"
        Conn.Execute "update product set hoffts=" & TsR(1) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set honts=" & TsR(2) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set hdiffts=" & TsR(3) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set moffts=" & TsR(4) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set monts=" & TsR(5) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set mdiffts=" & TsR(6) & " where productnum='" & ProductnumR &
"'"
        Conn.Execute "update product set loffts=" & TsR(7) & " where productnum='" & ProductnumR & "'"
        Conn.Execute "update product set lonts=" & TsR(8) & " where productnum='" & ProductnumR & "'"
        Conn.Execute "update product set ldiffts=" & TsR(9) & " where productnum='" & ProductnumR &
"'"
    Exit Sub
```

```
Err1:
    MsgBox Err.Description, vbOKOnly + vbExclamation, "错误提示！"
End Sub

Private Sub Music(TsValue As Integer, Direc As Integer)
  '5mon-6 2hon-3 1hoff-2 7loff-10 8lon-9 4moff-7
  '          3      4                          9       11
    SoundTime = 1
    TimerSound.Enabled = True
    If Direc = 1 Then
        TsL(TsValue) = TempTsL
       If BSoundSwitch Then
            If SoundCri < TsL(TsValue) Then
                BSoundEligibleL = False
                If TsValue = 1 Or TsValue = 2 Or TsValue = 3 Or TsValue = 5 Or TsValue = 8 Then
                    InitTestValue &H80000005, &HFF&, TsL(TsValue), TsValue + 1, Direc + 1
                Else
                    If TsValue = 4 Or TsValue = 7 Then
                        InitTestValue &H80000005, &HFF&, TsL(TsValue), TsValue + 3, Direc + 1
                    Else
                        InitTestValue &H80000005, &HFF&, TsL(TsValue), 11, Direc + 1
                    End If
                End If
            Else
                BSoundEligibleL = True
                If TsValue = 1 Or TsValue = 2 Or TsValue = 3 Or TsValue = 5 Or TsValue = 8 Then
                    InitTestValue &H80000005, &HC000&, TsL(TsValue), TsValue + 1, Direc + 1
                Else
                    If TsValue = 4 Or TsValue = 7 Then
                        InitTestValue &H80000005, &HC000&, TsL(TsValue), TsValue + 1, Direc + 1
                    Else
                        InitTestValue &H80000005, &HC000&, TsL(TsValue), TsValue + 1, Direc + 1
                    End If
                End If
            End If
        End If
    Else
        TsR(TsValue) = TempTsR
        If BSoundSwitch Then
            If SoundCri < TsR(TsValue) Then
                BSoundEligibleR = False
                If TsValue = 1 Or TsValue = 2 Or TsValue = 3 Or TsValue = 5 Or TsValue = 8 Then
                    InitTestValue &H80000005, &HFF&, TsR(TsValue), TsValue + 1, Direc + 1
                Else
                    If TsValue = 4 Or TsValue = 7 Then
                        InitTestValue &H80000005, &HFF&, TsR(TsValue), TsValue + 3, Direc + 1
                    Else
                        InitTestValue &H80000005, &HFF&, TsR(TsValue), 11, Direc + 1
                    End If
                End If
            Else
                BSoundEligibleR = True
                If TsValue = 1 Or TsValue = 2 Or TsValue = 3 Or TsValue = 5 Or TsValue = 8 Then
                    InitTestValue &H80000005, &HC000&, TsR(TsValue), TsValue + 1, Direc + 1
                Else
                    If TsValue = 4 Or TsValue = 7 Then
                        InitTestValue &H80000005, &HC000&, TsR(TsValue), TsValue + 1, Direc + 1
                    Else
                        InitTestValue &H80000005, &HC000&, TsR(TsValue), TsValue + 1, Direc + 1
```

```
                        End If
                    End If
                End If
            End If
        End If
End Sub

Private Sub StandardValue()
    Dim Recser As New ADODB.Recordset
    Dim Inty As Integer
    Recser.Open "select * from series where serid=" & CurSerId & "", Conn, adOpenStatic, adLockReadOnly, adCmdText
    With FlexGrid
        .TextMatrix(2, 5) = FormatNumber(Recser.Fields!highoff, 3)
        .TextMatrix(3, 5) = FormatNumber(Recser.Fields!highon, 3)
        .TextMatrix(4, 5) = FormatNumber(Recser.Fields!highdiff, 3)
        .TextMatrix(2, 6) = FormatNumber(Recser.Fields!highofftol, 3)
        .TextMatrix(3, 6) = FormatNumber(Recser.Fields!highontol, 3)
        .TextMatrix(4, 6) = FormatNumber(Recser.Fields!highdifftol, 3)

        .TextMatrix(7, 5) = FormatNumber(Recser.Fields!middleoff, 3)
        .TextMatrix(6, 5) = FormatNumber(Recser.Fields!middleon, 3)
        .TextMatrix(7, 6) = FormatNumber(Recser.Fields!middleofftol, 3)
        .TextMatrix(6, 6) = FormatNumber(Recser.Fields!middleontol, 3)

        .TextMatrix(9, 5) = FormatNumber(Recser.Fields!lowon, 3)
        .TextMatrix(10, 5) = FormatNumber(Recser.Fields!lowoff, 3)
        .TextMatrix(11, 5) = FormatNumber(Recser.Fields!lowdiff, 3)
        .TextMatrix(9, 6) = FormatNumber(Recser.Fields!lowontol, 3)
        .TextMatrix(10, 6) = FormatNumber(Recser.Fields!lowofftol, 3)
        .TextMatrix(11, 6) = FormatNumber(Recser.Fields!lowdifftol, 3)
        For Inty = 2 To 11
            .TextMatrix(Inty, 7) = Recser.Fields!SoundCri
        Next Inty
        For Inty = 2 To 11
          InitTestValue &H80000002, &HFAEBD6, " ", Inty, 1
          InitTestValue &H80000002, &HFAEBD6, " ", Inty, 2
          InitTestValue &H80000002, &HFAEBD6, " ", Inty, 9
          InitTestValue &H80000002, &HFAEBD6, " ", Inty, 10
        Next Inty
        Modulus = Recser.Fields!Modulus
        zero = Recser.Fields!zero
        YS = Recser.Fields!YS
        Aj(1) = Recser.Fields!P1j
        Aj(2) = Recser.Fields!P2j
        Aj(3) = Recser.Fields!P3j
        Aj(4) = Recser.Fields!P4j
        Aj(5) = Recser.Fields!P5j
        Aj(6) = Recser.Fields!P6j
        Aa(1) = Recser.Fields!P1a
        Aa(2) = Recser.Fields!P2a
        Aa(3) = Recser.Fields!P3a
        Aa(4) = Recser.Fields!P4a
        Aa(5) = Recser.Fields!P5a
        Aa(6) = Recser.Fields!P6a
        BHightSwitch = Recser.Fields!HightSwitch
        BLowSwitch = Recser.Fields!LowSwitch
        BSoundSwitch = Recser.Fields!SoundSwitch
        SoundCri = Recser.Fields!SoundCri
```

```
            Para(1) = Recser.Fields!parahoff
            Para(2) = Recser.Fields!parahon
            Para(4) = Recser.Fields!paramoff
            Para(5) = Recser.Fields!paramon
            Para(7) = Recser.Fields!paraloff
            Para(8) = Recser.Fields!paralon
            If BLowSwitch Then
                .TextMatrix(9, 5) = ""
                .TextMatrix(9, 6) = ""
            Else
                .TextMatrix(11, 5) = ""
                .TextMatrix(11, 6) = ""
            End If
            If BHightSwitch Then
                .TextMatrix(3, 5) = ""
                .TextMatrix(3, 6) = ""
            Else
                .TextMatrix(4, 5) = ""
                .TextMatrix(4, 6) = ""
            End If
        End With
        Set Recser = Nothing
End Sub

Private Sub DrawSwitch(Switch As Integer, directi As Integer)
    If directi = 1 Then
        Select Case Switch
            Case 3
                Linecolose(2).Visible = True
                LineOpen(0).Visible = False
                Linecolose(1).Visible = False
                LineOpen(1).Visible = True
                Linecolose(0).Visible = False
                LineOpen(2).Visible = True
                Shape2.FillColor = &HFF00&
                Shape3.FillColor = &HFF&
                Shape4.FillColor = &HFF&
            Case 2
                Linecolose(2).Visible = True
                LineOpen(0).Visible = False
                Linecolose(1).Visible = False
                LineOpen(1).Visible = True
                Linecolose(0).Visible = True
                LineOpen(2).Visible = False
                Shape2.FillColor = &HFF00&
                Shape4.FillColor = &HFF&
                Shape3.FillColor = &HFF00&
            Case 1
                Linecolose(2).Visible = True
                LineOpen(0).Visible = False
                Linecolose(1).Visible = True
                LineOpen(1).Visible = False
                Linecolose(0).Visible = True
                LineOpen(2).Visible = False
                Shape2.FillColor = &HFF00&
                Shape3.FillColor = &HFF00&
                Shape4.FillColor = &HFF00&
            Case 0
                Linecolose(2).Visible = False
```

```
                LineOpen(0).Visible = True
                Linecolose(1).Visible = True
                LineOpen(1).Visible = False
                Linecolose(0).Visible = True
                LineOpen(2).Visible = False
                Shape2.FillColor = &HFF&
                Shape3.FillColor = &HFF00&
                Shape4.FillColor = &HFF00&
        End Select
    Else
        Select Case Switch '0-3,1-4,2-5
            Case 3
                Linecolose(5).Visible = True
                LineOpen(3).Visible = False
                Linecolose(4).Visible = False
                LineOpen(4).Visible = True
                Linecolose(3).Visible = False
                LineOpen(5).Visible = True
                Shape5.FillColor = &HFF00&
                Shape7.FillColor = &HFF&
                Shape6.FillColor = &HFF&
            Case 2
                Linecolose(5).Visible = True
                LineOpen(3).Visible = False
                Linecolose(4).Visible = False
                LineOpen(4).Visible = True
                Linecolose(3).Visible = True
                LineOpen(5).Visible = False
                Shape5.FillColor = &HFF00&
                Shape7.FillColor = &HFF&
                Shape6.FillColor = &HFF00&
            Case 1
                Linecolose(5).Visible = True
                LineOpen(3).Visible = False
                Linecolose(4).Visible = True
                LineOpen(4).Visible = False
                Linecolose(3).Visible = True
                LineOpen(5).Visible = False
                Shape5.FillColor = &HFF00&
                Shape7.FillColor = &HFF00&
                Shape6.FillColor = &HFF00&
            Case 0
                Linecolose(5).Visible = False
                LineOpen(3).Visible = True
                Linecolose(4).Visible = True
                LineOpen(4).Visible = False
                Linecolose(3).Visible = True
                LineOpen(5).Visible = False
                Shape5.FillColor = &HFF&
                Shape7.FillColor = &HFF00&
                Shape6.FillColor = &HFF00&
        End Select
    End If
End Sub
Private Sub CompEligible(Eligible As Integer, Direction As Integer)
Dim i As Integer
    If Direction = 1 Then
        LabTestNum.Caption = Str(Eligible + 1)
    Else
```

```
        LabTestNumr.Caption = Str(Eligible + 1)
    End If
  Select Case Eligible
    Case 0 '8
        If Not BLowSwitch Then
            If Abs(FlexGrid.TextMatrix(9, 5) - SngTestValue(8)) < FlexGrid.TextMatrix(9, 6) Then
                InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(8), 3), 9, Direction
                BlnEligible(1) = True
            Else
                InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(8), 3), 9, Direction
                BlnEligible(1) = False
            End If
            If Direction = 1 Then
                SngTestValueL(8) = SngTestValue(8)
                SngTestValueL(9) = -1
                BlnEligibleL(1) = BlnEligible(1)
               'BlnAllEligibleL = BlnEligible1 And BlnAllEligibleL
            Else
                SngTestValueR(8) = SngTestValue(8)
                SngTestValueR(9) = -1
                BlnEligibleR(1) = BlnEligible(1)
                'BlnAllEligibleR = BlnEligible1 And BlnAllEligibleR
            End If
        End If
    Case 1 '5
        If Abs(FlexGrid.TextMatrix(6, 5) - SngTestValue(5)) < FlexGrid.TextMatrix(6, 6) Then
            InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(5), 3), 6, Direction
            BlnEligible(2) = True
        Else
            InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(5), 3), 6, Direction
            BlnEligible(2) = False
        End If
        If Direction = 1 Then
            SngTestValueL(5) = SngTestValue(5)
            BlnEligibleL(2) = BlnEligible(2)
            'BlnAllEligibleL = BlnEligible2 And BlnAllEligibleL
        Else
            SngTestValueR(5) = SngTestValue(5)
            BlnEligibleR(2) = BlnEligible(2)
            'BlnAllEligibleR = BlnEligible2 And BlnAllEligibleR
        End If
    Case 2 '1
        If Abs(FlexGrid.TextMatrix(2, 5) - SngTestValue(1)) < FlexGrid.TextMatrix(2, 6) Then
            InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(1), 3), 2, Direction
            BlnEligible(3) = True
        Else
            InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(1), 3), 2, Direction
            BlnEligible(3) = False
        End If
        If Direction = 1 Then
            SngTestValueL(1) = SngTestValue(1)
            BlnEligibleL(3) = BlnEligible(3)
            'BlnAllEligibleL = BlnEligible3 And BlnAllEligibleL
        Else
            SngTestValueR(1) = SngTestValue(1)
            BlnEligibleR(3) = BlnEligible(3)
            'BlnAllEligibleR = BlnEligible3 And BlnAllEligibleR
        End If
    Case 3 '3
```

```
        If BHightSwitch Then
            SngTestValue(3) = FormatNumber(Abs(SngTestValue(1) - SngTestValue(2)), 3)
            If Abs(FlexGrid.TextMatrix(4, 5) - SngTestValue(3)) < FlexGrid.TextMatrix(4, 6) Then
                InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(3), 3), 4, Direction
                BlnEligible(4) = True
            Else
                InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(3), 3), 4, Direction
                BlnEligible(4) = False
            End If
            If Direction = 1 Then
                SngTestValueL(3) = SngTestValue(3)
                SngTestValueL(2) = -1
                BlnEligibleL(4) = BlnEligible(4)
                'BlnAllEligibleL = BlnEligible4 And BlnAllEligibleL
            Else
                SngTestValueR(3) = SngTestValue(3)
                SngTestValueR(2) = -1
                BlnEligibleR(4) = BlnEligible(4)
                'BlnAllEligibleR = BlnEligible4 And BlnAllEligibleR
            End If
        Else '2
            If Abs(FlexGrid.TextMatrix(3, 5) - SngTestValue(2)) < FlexGrid.TextMatrix(3, 6) Then
                InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(2), 3), 3, Direction
                BlnEligible(5) = True
            Else
                InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(2), 3), 3, Direction
                BlnEligible(5) = False
            End If
            If Direction = 1 Then
                SngTestValueL(2) = SngTestValue(2)
                SngTestValueL(3) = -1
                BlnEligibleL(5) = BlnEligible(5)
                'BlnAllEligibleL = BlnEligible5 And BlnAllEligibleL
            Else
                SngTestValueR(2) = SngTestValue(2)
                SngTestValueR(3) = -1
                BlnEligibleR(5) = BlnEligible(5)
                'BlnAllEligibleR = BlnEligible5 And BlnAllEligibleR
            End If
        End If
    Case 4 '4
        If Abs(FlexGrid.TextMatrix(7, 5) - SngTestValue(4)) < FlexGrid.TextMatrix(7, 6) Then
            InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(4), 3), 7, Direction
            BlnEligible(6) = True
        Else
            InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(4), 3), 7, Direction
            BlnEligible(6) = False
        End If
        If Direction = 1 Then
            SngTestValueL(4) = SngTestValue(4)
            BlnEligibleL(6) = BlnEligible(6)
            'BlnAllEligibleL = BlnEligible And BlnAllEligibleL
        Else
            SngTestValueR(4) = SngTestValue(4)
            BlnEligibleR(6) = BlnEligible(6)
            'BlnAllEligibleR = BlnEligible And BlnAllEligibleR
        End If
    Case 5 '9
        If Abs(FlexGrid.TextMatrix(10, 5) - SngTestValue(7)) < FlexGrid.TextMatrix(10, 6) Then
```

```
            InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(7), 3), 10, Direction
            BlnEligible(7) = True
        Else
            InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(7), 3), 10, Direction
            BlnEligible(7) = False
        End If
        If Direction = 1 Then
            SngTestValueL(7) = SngTestValue(7)
            BlnEligibleL(7) = BlnEligible(7)
            'BlnAllEligibleL = BlnEligible And BlnAllEligibleL
        Else
            SngTestValueR(7) = SngTestValue(7)
            BlnEligibleR(7) = BlnEligible(7)
            'BlnAllEligibleR = BlnEligible And BlnAllEligibleR
        End If
        If BLowSwitch Then
            SngTestValue(9) = FormatNumber(Abs(SngTestValue(7) - SngTestValue(8)), 3)
            If Abs(FlexGrid.TextMatrix(11, 5) - SngTestValue(9)) < FlexGrid.TextMatrix(11, 6) Then
                InitTestValue &H80000005, &HC000&, FormatNumber(SngTestValue(9), 3), 11,
Direction
                BlnEligible(8) = True
            Else
                InitTestValue &H80000005, &HFF&, FormatNumber(SngTestValue(9), 3), 11, Direction
                BlnEligible(8) = False
            End If
            If Direction = 1 Then
                SngTestValueL(9) = SngTestValue(9)
                SngTestValueL(8) = -1
                BlnEligibleL(8) = BlnEligible(8)
                'BlnAllEligibleL = BlnEligible And BlnAllEligibleL
            Else
                SngTestValueR(9) = SngTestValue(9)
                SngTestValueR(8) = -1
                BlnEligibleR(8) = BlnEligible(8)
                'BlnAllEligibleR = BlnEligible And BlnAllEligibleR
            End If
        End If
    End Select

    If BSoundSwitch Then
        If Direction = 1 Then
            If BlnAllEligibleL And BSoundEligibleL Then
                BlnAllEligibleL = True
            Else
                BlnAllEligibleL = False
            End If
        Else
            If BlnAllEligibleR And BSoundEligibleR Then
                BlnAllEligibleR = True
            Else
                BlnAllEligibleR = False
            End If
        End If
    End If
End Sub


Private Sub SaveRecL()
    Dim TestDate As String
```

```
    Dim ProductnumL As String
    Dim BlnE As Boolean
    If CInt(LabTestNum.Caption) >= 6 Then
        BlnE = BlnAllEligibleL And True
    Else
        BlnE = BlnAllEligibleL And False
    End If

    If BLowSwitch And BHightSwitch Then
        BlnAllEligibleL = BlnAllEligibleL And BlnEligibleL(8) And BlnEligibleL(2) And BlnEligibleL(3)
And BlnEligibleL(4) And BlnEligibleL(6) And BlnEligibleL(7)
    Else
        BlnAllEligibleL = BlnAllEligibleL And BlnEligibleL(1) And BlnEligibleL(2) And BlnEligibleL(3)
And BlnEligibleL(5) And BlnEligibleL(6) And BlnEligibleL(7)
    End If
      '*****************************20030902
    If BlnAllEligibleL Then
        TxtBh.BackColor = vbGreen
    Else
        TxtBh.BackColor = vbRed
    End If
        '*****************************20030902
    TestDate = "#" & Date & "#"
    ProductnumL = CurSerName & TxtBh.Text
    On Error GoTo Err1
        Conn.Execute "insert into product (serID,productnum,hoff,hon,hdiff,moff,mon,mdiff,loff,lon,ldiff,
testdate,eligible,hoffts,honts,hdiffts,moffts,monts,mdiffts,loffts,lonts,ldiffts)" _
            & " VALUES ('" & CurSerId & "','" & ProductnumL & "'," _
            & SngTestValueL(1) & "," & SngTestValueL(2) & "," & SngTestValueL(3) & "," _
            & SngTestValueL(4) & "," & SngTestValueL(5) & "," & SngTestValueL(6) & "," _
            & SngTestValueL(7) & "," & SngTestValueL(8) & "," & SngTestValueL(9) & "," _
            & TestDate & "," & BlnAllEligibleL & "," _
            & TsL(1) & "," & TsL(2) & "," & TsL(3) & "," _
            & TsL(4) & "," & TsL(5) & "," & TsL(6) & "," _
            & TsL(7) & "," & TsL(8) & "," & TsL(9) & ")"
    Exit Sub
Err1:
    MsgBox Err.Description, vbOKOnly + vbExclamation, "错误提示！"
End Sub
Private Sub SaveRecR()
    Dim TestDate As String
    Dim ProductnumR As String
    Dim BlnE As Boolean
    If CInt(LabTestNumr.Caption) >= 6 Then
        BlnE = BlnAllEligibleR And True
    Else
        BlnE = BlnAllEligibleR And False
    End If

    If BLowSwitch And BHightSwitch Then
        BlnAllEligibleR = BlnAllEligibleR And BlnEligibleR(8) And BlnEligibleR(2) And BlnEligibleR(3)
And BlnEligibleR(4) And BlnEligibleR(6) And BlnEligibleR(7)
    Else
        BlnAllEligibleR = BlnAllEligibleR And BlnEligibleR(1) And BlnEligibleR(2) And BlnEligibleR(3)
And BlnEligibleR(5) And BlnEligibleR(6) And BlnEligibleR(7)
    End If
        '*****************************20030902
    If BlnAllEligibleR Then
        TxtBhr.BackColor = vbGreen
```

```
    Else
        TxtBhr.BackColor = vbRed
    End If
        '*****************************20030902
    TestDate = "#" & Date & "#"
    ProductnumR = CurSerName & TxtBhr.Text
    On Error GoTo Err1
    Conn.Execute "insert into product (serID,productnum,hoff,hon,hdiff,moff,mon,mdiff,loff,lon,ldiff,testdate,eligible, hoffts,honts,hdiffts,moffts,monts, mdiffts,loffts,lonts,ldiffts)" _
        & " VALUES ('" & CurSerId & "','" & ProductnumR & "'," _
        & SngTestValueR(1) & "," & SngTestValueR(2) & "," & SngTestValueR(3) & "," _
        & SngTestValueR(4) & "," & SngTestValueR(5) & "," & SngTestValueR(6) & "," _
        & SngTestValueR(7) & "," & SngTestValueR(8) & "," & SngTestValueR(9) & "," _
        & TestDate & "," & BlnAllEligibleR & "," _
        & TsR(1) & "," & TsR(2) & "," & TsR(3) & "," _
        & TsR(4) & "," & TsR(5) & "," & TsR(6) & "," _
        & TsR(7) & "," & TsR(8) & "," & TsR(9) & ")"
    Exit Sub
Err1:
    MsgBox Err.Description, vbOKOnly + vbExclamation, "错误提示！"
End Sub

Private Sub TimReadJJ_Timer()
    If Aa(4) > TxtTarget.Text And TxtTarget.Text > Aj(4) Then
            ClrPs0
            SetPs7
        SetPs1
        SetPs2
        SetPs3
        ClrPs4
        SetPs5
        SetPs6
        If Aa(4) * 0.98 > TxtTarget.Text And TxtTarget.Text > Aj(4) * 1.02 Then
            State 1, 8
            State 9, 8
        End If

    ElseIf Aj(4) > TxtTarget.Text And TxtTarget.Text > Aa(5) Then
            ClrPs0
            SetPs7
        SetPs1
        SetPs2
        SetPs3
        ClrPs4
        SetPs5
        ClrPs6
        ClrK1
        ClrK1r

    ElseIf Aa(5) > TxtTarget.Text And TxtTarget.Text > Aj(5) Then
            ClrPs0
            SetPs7

        SetPs1
        SetPs2
        SetPs3
        SetPs4
        ClrPs5
```

```
            SetPs6
            If Aa(5) * 0.98 > TxtTarget.Text And TxtTarget.Text > Aj(5) * 1.02 Then
                State 1, 10
                State 9, 10
            End If

    ElseIf Aj(5) > TxtTarget.Text And TxtTarget.Text > Aa(6) Then
                ClrPs0
                SetPs7
            SetPs1
            SetPs2
            SetPs3
            ClrPs4
            ClrPs5
            ClrPs6
            ClrK1
            ClrK1r

    ElseIf Aa(6) > TxtTarget.Text And TxtTarget.Text > Aj(6) Then
                ClrPs0
                SetPs7
            SetPs1
            SetPs2
            SetPs3
            ClrPs4
            ClrPs5
            SetPs6
            If Aa(6) * 0.98 > TxtTarget.Text And TxtTarget.Text > Aj(6) * 1.02 Then
                State 1, 12
                State 9, 12
            End If

    ElseIf Aj(6) > (CInt(GetMN1) - zero) * Modulus Then
            ClrPs7
            SetPs0

            SetPs1
            SetPs2
            SetPs3
            SetPs4
            SetPs5
            SetPs6
            ClrK1
            ClrK1r
            If BlnRetry Then
                UpDateRecL
                UpDateRecR
            Else
                SaveRecL
                SaveRecR
            End If
        TimReadJJ.Enabled = False
        TimerButten.Enabled = True
    Else
            ClrK1
            ClrK1r
    End If
End Sub
```

```
Private Sub TimS_Timer()
    If GetKs = 3 Then
        ClrPs7
        SetPs0

        SetPs1
        SetPs2
        SetPs3
        SetPs4
        SetPs5
        SetPs6
        TimS.Enabled = False
        TimerButten.Enabled = True
        TimJT.Enabled = True
    End If
End Sub

'Private Sub TxtTarget_Change()
'     TimMn1.Enabled = True
'End Sub

Private Sub TimYS_Timer()
    IntYs = IntYs + 1
    If IntYs = YS Then
        ClrPs0
        ClrPs1
        SetPs7
        SetPs2
        SetPs3
        SetPs4
        SetPs5
        SetPs6
        TimRead.Enabled = True
        TimYS.Enabled = False
    End If
End Sub
//============================================================
//                    通用模块
//============================================================
Option Explicit '口令是否正确
Public PubBlnPasWrdOK As Boolean

Public Const PserBitCount = 8 '产品序号位数

Public Conn As New ADODB.Connection
Public CurSerId As String
Public CurSerName As String

Public BHightSwitch As Boolean
Public BMiddleSwitch As Boolean
Public BLowSwitch As Boolean
Public Para(1 To 9) As Single
Public PointTarget As Integer    '当前指针值

Public Declare Function GetK1 Lib "W302DLL.dll" Alias "_sGetLKP" () As Long
Public Declare Function GetKa Lib "W302DLL.dll" Alias "_sGetLKa" () As Long
Public Declare Function GetMN2 Lib "W302DLL.dll" Alias "_sGetLMn" () As Long
Public Declare Function GetTs Lib "W302DLL.dll" Alias "_sGetLTs" () As Long
```

```
Public Declare Function ClrK1 Lib "W302DLL.dll" Alias "_sClrLKP" () As Integer

Public Declare Function GetK1r Lib "W302DLL.dll" Alias "_sGetRKP" () As Long
Public Declare Function GetKar Lib "W302DLL.dll" Alias "_sGetRKa" () As Long
Public Declare Function GetMN2r Lib "W302DLL.dll" Alias "_sGetRMn" () As Long
Public Declare Function GetTsr Lib "W302DLL.dll" Alias "_sGetRTs" () As Long
Public Declare Function ClrK1r Lib "W302DLL.dll" Alias "_sClrRKP" () As Integer

Public Declare Function GetMN1 Lib "W302DLL.dll" Alias "_sGetMn" () As Long
Public Declare Function GetKs Lib "W302DLL.dll" Alias "_sGetKs" () As Long


Public Declare Function SetPs0 Lib "W302DLL.dll" Alias "_sSetPs0" ()
Public Declare Function SetPs1 Lib "W302DLL.dll" Alias "_sSetPs1" ()
Public Declare Function SetPs2 Lib "W302DLL.dll" Alias "_sSetPs2" ()
Public Declare Function SetPs3 Lib "W302DLL.dll" Alias "_sSetPs3" ()
Public Declare Function SetPs4 Lib "W302DLL.dll" Alias "_sSetPs4" ()
Public Declare Function SetPs5 Lib "W302DLL.dll" Alias "_sSetPs5" ()
Public Declare Function SetPs6 Lib "W302DLL.dll" Alias "_sSetPs6" ()
Public Declare Function SetPs7 Lib "W302DLL.dll" Alias "_sSetPs7" ()

Public Declare Function ClrPs0 Lib "W302DLL.dll" Alias "_sClrPs0" ()
Public Declare Function ClrPs1 Lib "W302DLL.dll" Alias "_sClrPs1" ()
Public Declare Function ClrPs2 Lib "W302DLL.dll" Alias "_sClrPs2" ()
Public Declare Function ClrPs3 Lib "W302DLL.dll" Alias "_sClrPs3" ()
Public Declare Function ClrPs4 Lib "W302DLL.dll" Alias "_sClrPs4" ()
Public Declare Function ClrPs5 Lib "W302DLL.dll" Alias "_sClrPs5" ()
Public Declare Function ClrPs6 Lib "W302DLL.dll" Alias "_sClrPs6" ()
Public Declare Function ClrPs7 Lib "W302DLL.dll" Alias "_sClrPs7" ()

Public Declare Sub sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
Public Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As
String, ByVal uFlags As Long) As Long

'Public Declare Function sleep Lib "winmm.dll" (ByVal lpszSoundName As String, ByVal uFlags As Long) As
Long


Public Sub Main()
    打开链接
    Conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & App.Path &
"\series.mdb;Persist Security Info=False"
    Conn.Open
    FrmFlash.Show

End Sub

Public Sub FrmCent(FrmX As Form)
    FrmX.Top = (Screen.Height - FrmX.Height) / 2
    FrmX.Left = (Screen.Width - FrmX.Width) / 2
End Sub

'生成产品编号
Public Function CreateProductID() As String
    Dim RecProduct As New ADODB.Recordset

    打开产品库，读取编号
    RecProduct.Open "select max(right(productnum," & PserBitCount & "))  as ProductNumMax from
```

```
product where serid=" & CurSerId & "", Conn, adOpenStatic, adLockReadOnly, adCmdText

    If IsNull(RecProduct.Fields!ProductNumMax) Then
        CreateProductID = "00000001"
    Else
        CreateProductID = Format(Right(RecProduct.Fields!ProductNumMax, PserBitCount) + 1,
"00000000")
    End If

End Function
```

参 考 文 献

1 王远. 模拟电子技术. 北京：机械工业出版社，1994
2 高光天. 仪表放大器应用. 北京：科学出版社，1995
3 童诗白等. 模拟电子技术基础. 北京：高等教育出版社，2001
4 康华光. 电子技术基础. 北京：高等教育出版社，1988
5 阎石. 数字电子技术基础. 北京：高等教育出版社，2001
6 张永瑞. 电子测量技术基础. 西安：西安电子科技大学出版社，1994
7 何立民. 单片机应用系统设计. 北京：北京航空航天大学出版社，1990
8 赵茂泰. 智能仪器原理及其应用. 北京：电子工业出版社，1999
9 余雷声等. 微型计算机原理及接口技术. 北京：化学工业出版社，1995
10 中国集成电路大全编委会. 中国集成电路大全——TTL 集成电路. 北京：国防工业出版社，1985
11 中国集成电路大全编委会. 中国集成电路大全——CMOS 集成电路. 北京：国防工业出版社，1985
12 金革等编译. 可编程逻辑阵列 FPGA 和 EPLD. 合肥：中国科技大学出版社，1996
13 饶运涛等编著. 现场总线 CAN 原理及其应用技术. 北京：北京航空航天大学出版社，2003
14 Dhananjay. V. Gadre 著，并行端口编程. 韩永彬等译. 北京：中国电力出版社，2000
15 赵新民. 智能仪器原理基设计. 哈尔滨：哈尔滨工业大学出版社，2000
16 胡大可. MSP430 系列超低功耗 16 位单片机原理及其应用. 北京：北京航空航天出版社，2000
17 徐爱钧等. 单片机高级语言 C51 应用程序设计. 北京：电子工业出版社，2001
18 魏庆福等. STD 总线工业控制机的设计与应用. 北京：科学出版社，1991
19 李朝青. PC 机及单片机设计通信技术. 北京：北京航空航天大学出版社，2000
20 张剑平. 普通运放的单电源应用探讨. 电子科学技术，1988 年第 4 期
21 张剑平. 不随输入频率变化的移相放大器. 电子技术应用，1988 年第 9 期
22 张剑平. 电子皮带秤传感器的选择及秤体设计探讨. 传感器世界，2002 年 2 期
23 张剑平. 外置式虚拟示波器的设计探讨. 电子世界，2001 年 11 期
24 张剑平. 传感器的发展方向及数字传感器的地位. 传感器世界，2001 年 10 期
25 张剑平. MS430 系列单片机及其应用. 仪器仪表与分析监测，2002 年 2 期