摘　要

，由 CAN 总线是近些年新兴的一种串行现场总线于 CAN 总线的信号传输时间短，受干扰的概率低。当节点错误严重时，具有自动关闭功能，以切断该节点与总线的联系，是总线上的其他节点及通信不受影响，具有将强的抗干扰能力。以上优点使 CAN 总线成为最有前途的现场总线之一。

RS422 是一种较早的集成式串行通讯协议，使用广泛，很多变频器、可编程控制器和各种智能仪表都带有 RS422 接口。如果将其接入到 CAN 总线网络中去，更换所有仪器仪表显然很不明智。最为简单的是设计一种转换器，它可以把 RS422 总线转换成性能更好的、稳定性更强的 CAN 总线，我们这次要设计的就是一种从 RS422 到 CAN 转换的器件。它可以将仪表上传输过来的数据转换成 CAN 协议下的数据，并将其与发送到其他节点；也可以将总线上其他节点上的数据传输到带有 RS422 的仪器仪表中。这种转化器的价格低廉，性能优异，在船体的网络设计中有很广泛的实际应用 。

关键词：CAN 总线；转换器；单片机；

ABSTRACT

The CAN bus is a kind of new developing serial filedbus in recent years, because the signal transmission time of the CAN bus is short, the probability interfered is low. When the nodal mistake is serious, close the function automatically, in order to cut off the connection between this node and the bus, it does not influence other node and communication of the bus, have better anti-interference ability. The above-mentioned advantages make the CAN bus become one of the most promising filedbus.

RS422 is a kind of earlier integrating type serial communication agreement, and use extensively. A lot of frequency converters, programmable controller and various kinds of intelligent instruments have RS422 connect. If insert it to the CAN bus network, it is obviously very unwise to change all instrument and apparatus. The simplest one way is to design a kind of converter, it can change RS422 bus into the CAN bus. What we will design is a kind of device changed from RS422 to CAN. It can change the data under the RS422 agreement into the data under the CAN agreement, and sends it to other nodes; The CAN bus transmit the data to other nodes on the bus to the instrument and apparatus with RS422 too. This kind of transformation device is cheap, so there is very extensive practical application in the design of the hull.

Key words: CAN_bus; converter; Single_Chip Microcomputer

目 录

第一章 绪论……1
1.1 组合导航的产生与发展……1
1.2 CAN 总线和 RS422 总线在组合导航中的作用……2
1.3 RS422 总线介绍 ……3
1.3.1 串行通讯和 RS422 系列标准 ……3
1.3.2 RS422 电气规定 ……4
1.4 CAN 总线介绍 ……6
1.4.1 现场总线形成与发展……6
1.4.2 CAN 总线工作方式与特点 ……8
1.5 RS422 到 CAN 总线转换器设计……11
1.5.1 RS422 到 CAN 总线转换器意义 ……11
1.5.2 RS422 到 CAN 总线转换器设计要求 ……12
1.5.3 RS422 到 CAN 总线转换器设计方案……13
本章总结……13
第二章 转换器硬件设计 ……15
2.1 单片机及所用器件简介……15
2.1.1 C8051F040 单片机的特点和应用……16
2.1.2 PCA82C250 总线收发器……19
2.1.3 通讯扩展器件 TL16C554……22
2.1.4 RS422 转换器件 MAX1490A……25
2.2 电路设计软件 PROTEL 99 的使用方法……26
2.2.1 PROTEL 99 简介……26
2.2.2 电路板设计步骤……27
2.2.3 硬件电路原理图的设计……28

本章总结 ……28
第三章 转换器的软件设计……29
3.1 单片机软件程序设计过程……29
3.2 所需器件的软件和协议应用 ……30
3.3 配置 CAN 寄存器 ……32
3.3.1 CAN 报文对象初始化处理……32
3.3.2 发送与接受对象的配置……33
3.3.3 中断处理……35
3.3.4 TL16C554 初始化……37
3.4 转换器工作原理流程图……38
3.5 系统程序设计……41
3.5.1 主程序……41
3.5.2 子程序部分……42
3.5.3 两个中断子程序……43
本章小节……43
第四章 转换器的调试 ……45
结论 ……47
参考文献……48
致谢 ……49
附录 ……50

第一章 绪 论

1.1 组合导航的产生和发展

组合导航的产生和发展取决于客观需要和现实可能性。近数十年来，世界上许多先进工业国家，为了适应航行和运载工具的需要，先后研制出许多导航体制和设备，分别解决各自的特定任务。

由于单一导航系统都存在自己的局限性，有所长亦有所短，因此单一导航系统难于满足现代航行的需要，所以要求有一种准确、迅速、高性能的导航系统产生，这就是组合导航系统产生的客观需要。

从现实可能性来说，由于各种单一导航系统和设备的发展已经处于成熟而稳定的阶段，它们各自的优缺点表现明显。如惯性导航是自主式导航系统，不受外界干扰，相对精度比较高，但它的误差随时间积累；天文导航也是自主式导航系统，误差不随时间而增长，但它受气象条件和昼夜因素限制；无线电导航系统定位误差不随时间增长，但受外界干扰影响较大；卫星导航的精度高，容易做到全球、全天候导航，但它技术复杂，且目前使用的子午仪系统还不能连续提供导航信息，两次定位间隔时间较长；此外还有一些导航设备（罗经、平台罗经、计程仪等）可提供单一的或少量的导航信息，他们已发展到完善阶段。这些都给组合导航的形成提供了充分的条件。

在组合导航系统中，状态与噪声都是随机的，对来自众多导航设备的导航信息要加以综合处理，包括：对描述未来状态的随机信号的预测；随机信号与随机噪声的分离，并将分离后的随机信号进行检测等。解决这些随机信号的估计，是提高导航性能的关键。为此，经过多年的研究，直到本世纪 60 年代初卡尔曼提出

最佳线性递推估计滤波法，才为组合导航的实现提供了良好的数学工具，大大加速了组合导航系统的发展。

当前，广泛使用的导航设备已基本处于稳定时期，组合导航从原理到体制已趋于成熟，特别是电子计算机的发展日新月异，使船舶导航系统日益完善。随着微型计算机的这一异军突起，组合导航系统将会开创一个新的局面。

组合导航发展趋势是日益小型化、数字化、全自动化、既能实现自动检测，自动转换，自动试验又能自动报警、自动记录、自动打印与标绘等。

还要指出，美国第二代卫星导航的导航星全球定位系统GPS，在投入使用后，由于它能连续定位、精度高、全球全天候使用，又能提供速度、时间信息等独特优点，以惯性导航和全球定位系统 GPS 为核心的组合导航，必将使组合导航系统跨入一个全新的时代。

1.2 CAN 总线和 RS422 总线在组合导航中的应用

20 年来电子计算机技术迅速发展，并得到日益完善和广泛的应用，为组合导航的实时计算和控制创造了技术条件，因此，使组合导航系统的实现成为可能。由于采用多功能、多模式的体制。即将多种传感器、多模式工作方式及资料实事处理、滤波和逻辑职能等计算技术结合为一个整体，形成集成化组合导航。再有是将组合导航系统与其它系统组合在一起，如与自动航行系统、避碰系统、舰船控制系统及通讯监视、作战指挥系统等的组合，已完成导航以外的更广泛的功能，这将是未来所有舰船发展的方向。那么将如此多的器件与系统连接起来而形成一个完整的大系统，可以相互协调而同一工作，在连接上就要采用总线结构。总线将舰船上各个导航设备按照一定的结构连接起来，使其能相互通讯，从而使各个设备充分发挥它们的功能，更易于工作人员的控制，

提高导航控制的自动化。RS422 总线和 CAN 总线就是广泛采取的两种总线结构。它们都属于串行总线，都具有良好的通讯能力。但 RS422 总线无论是连接方式、通讯距离、通讯效果、错误检测方面都不如 CAN。本章将分别介绍 RS422 和 CAN 总线。

1.3 RS422 总线介绍

1.3.1 串行通讯和 RS422 系列标准

随着计算机系统的应用和微机网络的发展，通信功能越来越显的重要。这里所说的通信是只计算机与外界的信息交换。因此，通信既包括计算机与外部设备之间，也包括计算机和计算机之间的信息交换。由于串行通信是在一根传输线上一位一位的传送信息，所用的传输线少，并且可以借助现成的电话网进行信息传送，因此，特别适合于远距离传输。对于那些与计算机相距不远的人一机交换设备和串行存储的外部设备如终端、打印机、逻辑分析仪、磁盘等，采用串行方式交换数据也很普遍。在实时控制和管理方面，采用多台微机处理机组成分级分布控制系统中，各 CPU 之间的通信一般都是串行方式。所以串行接口是微机应用系统常用的接口。典型的串行接口的结构如图 1.1 所示

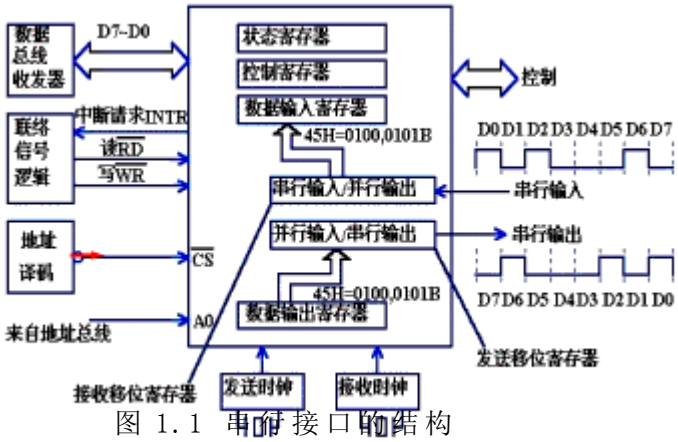

图 1.1 串行接口的结构

串行通讯的特点是：数据位传送，传按位顺序进行，最少只需一根传输线即可完成，成本低但传送速度慢。串行通讯的距离可以从几米到几千米。根据信息的传送方向，串行通讯可以进一步分为单工、半双工和全双工三种。信息只能单向传送为单工；信息能双向传送但不能同时双向传送称为半双工；信息能够同时双向传送则称为全双工。串行通讯又分为异步通讯和同步通讯两种方式。在单片机中，主要使用异步通讯方式。51 系列单片机有一个全双工串行口。全双工的串行通讯只需要一根输出线和一根输入线。数据的输出又称发送数据（TXD），数据的输入又称接收数据（RXD）。串行通讯中主要有两个技术问题，一个是数据传送、另一个是数据转换。数据传送主要解决传送中的标准、格式及工作方式等问题。数据转换是指数据的串并行转换。具体说，在发送端，要把并行数据转换为串行数据；而在接收端，却要把接收到的串行数据转换为并行数据。

1969 年，美国电子工业协会(EIA)公布了 RS-232C 作为串行通信接口的电气标准，该标准定义了数据终端设备(DTE)和数据通信设备(DCE)间按位串行传输的接口信息，合理安排了接口的电气信号和机械要求，在世界范围内得到了广泛的应用。但它采用单端驱动非差分接收电路，因而存在着传输距离不太远(最大传输距离 15m)和传送速率不太高(最大位速率为 20Kb/s)的问题。远距离串行通信必须使用 Modem，增加了成本。在分布式控制系统和工业局部网络中，传输距离常介于近距离(＜20m＝和远距离(＞2km)之间的情况，这时 RS-232C（25 脚连接器）不能采用，用 Modem 又不经济，因而需要制定新的串行通信接口标准。 1977 年 EIA 制定了 RS-449。它除了保留与 RS-232C 兼容的特点外，还在提高传输速率增加传输距离及改进电气特性等方面作了很大努力，

并增加 10 个控制信号。与 RS-449 同时推出的还有 RS-422 和 RS-423，它们是 RS-449 的标准子集。另外，还有 RS-485，它是 RS-422 的变形。RS-422、RS-423 是全双工的，而 RS-485 是半双工的。RS-422 由 RS-232 发展而来，它是为弥补 RS-232 之不足而提出的。为改 RS-232 通信距离短、速率低的缺点，RS-422 定义了一种平衡通信接口，将传输速率提高到 10Mb/s，传输距离延长到 4000 英尺（速率低于 100kb/s 时），并允许在一条平衡总线上连接最多 10 个接收器。RS-422 是一种单机发送、多机接收的双向、平衡传输规范，被命名为 TIA/EIA-422-A 标准。

1.3.2 RS422 电气规定

RS-422 标准全称是“平衡电压数字接口电路的电气特性”，它定义了接口电路的特性。图1.2a是典型的RS-422 四线接口，实际上还有一根信号地线，共5根线。分别是发送A（TXA），发送B（TXB），接收A（RXA），接收B（RXB），接地线GND，9V电源线。图1.2b是其DB9(九个引脚)连接器引脚内部的差动工作方式。由于接收器采用高输入阻抗和发送驱动器，比RS232更强的驱动能力，故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，但是不支持多对多的通信。接收器输入阻抗为4k，故发端最大负载能力是10×4k+100.（终端要接平衡电阻）。RS-422 四线接口由于采用单独的发送和接收通道，因此不必控制数据方向，各装置之间任何必须的信号交换均可以按软件方式（XON/XOFF 握手）或硬件方式（一对单独的双绞线）实现。若采用硬件握手会增加布线成本，以此如果通讯距离远，一般采取的是软件握手的方式。

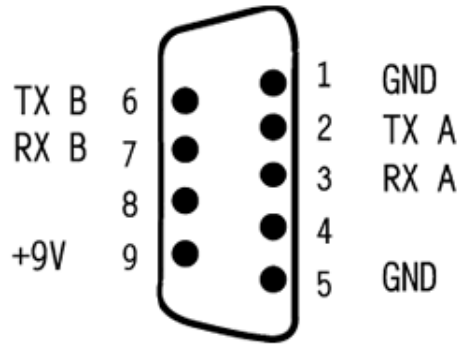

图 1.2a RS-422 四线接口

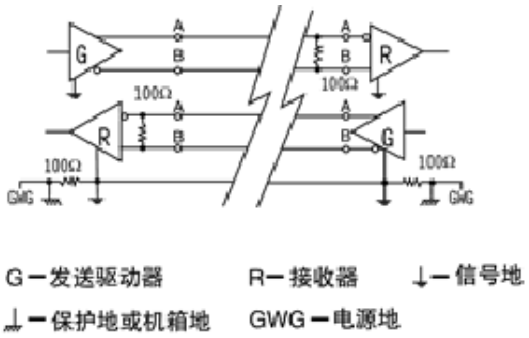

图 1.2bDB9 连接器引脚定义

RS-422 的最大传输距离为4000英尺（约1219米），最大传输速率为10Mb/s。其平衡双绞线的长度与传输速率成反比，在100kb/s 速率以下，才可能达到最大传输距离。只有在很短的距离下才能获得最高速率传输。一般100 米长的双绞线上所能获得的最大传输速率仅为1Mb/s。RS-422 需要一个终接电阻，要求其阻值约等于传输电缆的特性阻抗。在短距离传输时可不需终接电阻，即一般在300米以下不需终接电阻。终接电阻接在传输电缆的最远端。

1.4 CAN 总线介绍

1.4.1 现场总线形成与发展

随着控制、计算机、通讯、网络等技术的发展，信息交换沟通的领域正在迅速覆盖从工厂的现场设备层到控制、管理的各个层次、覆盖从工段、车间、工厂、企业乃至世界各地的市场。信息技术的飞速发展，引起了自动化系统结构的变革，逐步形成以网络集成自动化系统为基础的企业信息系统。现场总线(fieldbus)就是顺应这一形势发展起来的新技术。

现场总线是当今自动化领域技术发展的热点之一，被誉为自动化领域的计算机局域网。它的出现，标志着工业控制技术领域有一个新时代的开始，并将对该领域的发展产生重要影响。现场总线是应用在生产现场、在微机化测量控制之间实现双向串行多

结点数字通讯的系统，也被称为开放式、数字化、多点通讯的底层控制网络。它在制造业、流程工业、交通、楼宇等方面的自动化系统中有广泛的应用前景。

现场总线技术将专用微处理器植入传统的测量控制仪表，使他们各自都具有了数之计算和数字通讯能力，采用了可进行简单连接的双绞线等作为总线，把多个测量仪表联接成网络系统，并按公开、规范的通讯协议，在位于现场的多个微机化测量控制设备之间以及现场仪表与远程计算机之间，实现数据传输和信息交换，形成各种适应实际需要的自动化控制系统。简而言之，他把单个分散的测量控制设备变成网络节点，以现场总线为纽带，把它们连接成可以相互沟通信息、共同完成自控任务的网络系统与控制系统。给自动化领域带来的变化，正如众多分散的计算机被网络连接到一起，使计算机的功能、作用发生的变化。现场总线则使自控系统与设备具有了通讯能力，把它们连接成网络系统，加入到信息网络的行列。因此把现场总线技术说成是一个控制技术新时代的开端并不过分。

现场总线控制系统既是一个开放的通讯网络，又是一种全分布控制系统。它作为智能设备的联系纽带，把作为网络节点的职能设备连接为网络系统，并进一步构成自动化系统，实现基本控制、补偿计算、参数修改、报警、显示、监控、优化及控管一体化的综合自动化功能。他是一项以智能传感器、控制、计算机、数字通讯、网络为主要内容的综合技术。

由于现场总线适应了工业控制系统向分散化、网络化智能化发展的方向，它已经变成了全球工业自动化技术的热点，受到全世界的普遍关注。现场总线的出现，导致目前生产的自动化仪表、集散控制系统（DCS）、可编程控制器（PLD）在产品的体系结构、功能方面发生了较大的变革，自动化设备的制造厂家被迫面临产

品更新换代的又一次挑战。传统的模拟仪表将逐步让位于智能化数字仪表，并具备了数字通讯功能。出现了一批集检测、运算、控制功能于一体的变送控制器；出现了带有控制模块和具有故障信息的执行器，并由此大大改变了现有的设备维护管理方法。现场总线是用于过程控制现场仪表与控制室之间的一个标准的、开放的、双向的多站数字通信系统。随着计算机技术、通讯技术、集成电路技术的发展，以全数字式现场总线（FIELDBUS）为代表的互联规范，正在迅猛发展和扩大。由于采用现场总线将使控制系统结构简单，系统安装费用减少并且易于维护；用户可以自由选择不同厂商、不同品牌的现场设备达到最佳的系统集成等一系列的优点，现场总线技术正越来越受到人们的重视。近十几年由于现场总线的国际标准不能建立，现场总线发展的种类较多，约有 40 余种：如德国西门子公司 Siemens 的 ProfiBus，法国的 FIP，英国的 ERA，挪威的 FINT，Echelon 公司的 LONWorks，PhenixContact 公司的 InterBus，德国 RoberBosch 公司的 CAN，Rosemounr 公司的 HART，CarloGarazzi 公司的 Dupline，丹麦 ProcessData 公司的 P-net，PeterHan 公司的 F-Mux，以及 ASI（ActraturSensorInterface），MODBus，SDS，Arcnet，国际标准组织-基金会现场总线 FF：FieldBusFoundation，WorldFIP，BitBus，美国的 DeviceNet 与 ControlNet 等等。 现场总线的种类主要有：基金会现场总线 FF，ProfiBus，WorldFIP，ControlNet/DeviveNet，CAN 等。

1.4.2 CAN 总线工作方式与特点

各种总线特点不同，应用方式和范围也不尽相同。其中 CAN 由于其高性能、高可靠性及独特的设计，CAN 越来越受到人们的重视。国外已有许多大公司的产品采用了这种技术。CAN 最初是由德国 BOSCH 公司为汽车监测、控制系统而设计的。众所周知，现代

汽车越来越多地采用电子装置控制，如发动机的定时、注油控制、加速、刹车控制 （ASC）及复杂的抗锁定刹车系统（ABS）等。由于这些控制需检测及交换大量的数据，采用采取主从方式不但繁琐、昂贵，而且难以解决问题，采用 CAN 总线上述问题便得到了很好的解决。据资料介绍，世界上一些著名的汽车制造厂商，如 BENZ（奔驰）、BMW（宝马）、PORSCHE（保时捷）、PRLLS-ROYCE（劳斯莱斯）和 JAGUAR（美洲豹）都开始采用 CAN 总线来实现汽车内部控制系统与各检测和执行机构间的数据通讯。

CAN(Controller Area Network)称为控制局域网，属于总线式通讯网络。CAN 总线规范了任意两个 CAN 节点之间的兼容性,包括电气特性及数据解释协议，CAN 协议分为二层：物理层和数据链路层。物理层决定了实际位传送过程中的电气特性，在同一网络中，所有节点的物理层必须保持一致，但可以采用不同方式的物理层。CAN 的数据链路层功能包括帧组织形式，总线仲裁和检错、错误报告及处理，确认哪个信息要发送的，确认接收到的信息及为应用层提供了接口。CANBUS 网络上任意一个节点均可在任意时刻主动向网络上的其它节点发送信息，而不分主从。通讯灵活，可方便地构成多机备份系统及分布式监测、控制系统。网络上的节点可分成不同的优先级以满足不同的实时要求。采用非破坏性总线裁决技术，当两个节点同时向网络上传送信息时，优先级低的节点主动停止数据发送，而优先级高的节点可不受影响地继续传输数据。具有点对点，一点对多点及全局广播传送接收数据的功能。通讯距离最远可达 10KM/5KBPS，通讯速率最高可达 1MBPS/40M。网络节点数实际可达 110 个。每一帧的有效字节数为 8 个，这样传输时间短，受干扰的概率低。每帧信息都有 CRC 校验及其它检错措施，数据出错率极低，可靠性极高. 通讯介质采用廉价的双绞线即可，无特殊要求。在传输信息出错严重时，节点

可自动切断它与总线的联系，以使总线上的其它操作不受影响。CAN 网络具有如下特点：

1. CAN 属于总线式串行网络，由于其采用了许多新技术及独特的设计，与一般的通讯总线相比，CAN 总线上的数据通信具有突出的可靠性、实时性和灵活性。其特点可概括如下：CAN 为多主方式工作，网络上任意节点均可在任意时刻主动地向网络上其它节点发送信息，而从不分主从，通信方式灵活，且无需站地址等信息。利用这一特点可方便的构成多机备份系统。
2. CAN 网络上的节点信息分成不同的优先级，可满足不同的实时要求，高优先级的数据最多可在 134us 内得到传输。
3. CAN 采用非破坏性总线仲裁技术，当多个节点同时向总线发送信息时，优先级较低的节点会主动退出总线，而优先级较高的节点可不受影响的传送数据，从而大大节省了总线冲突仲裁时间。尤其是在网络负载很重的情况下也不会出现网络瘫痪情况（以太网则可能）。
4. CAN 只需通过报文滤波即可实现点对点、一点对多点及全局广播等几种方式传送数据，而无需专门的“调度”。
5. CAN 的直接通讯距离最 yuankeda10KM（速率 5kps 以下）；通讯速率最多可达 1Mbps（此时通讯距离最长 40m）。
6. CAN 上的节点数主要取决于总线驱动电路，目前可达 110 个；报文标识符可达 2030 种（CAN2.0A），而扩展标准（CAN2.0B）的报文标识符几乎不受限制。
7. 采用短帧结构，通讯距离短，受干扰率低，具有极好的检测效果。

8. CAN 的每帧信息都有 CRC 校验和其它检错措施，保证了数据出错率极低。
9. CAN 的通讯介质可以是双绞线、同轴电缆或光纤，选择灵活。
10. CAN 节点在错误影响严重的情况下具有自动关闭输出功能，以使总线上其它节点的操作不受影响。图 1.3 为总线应用原理图

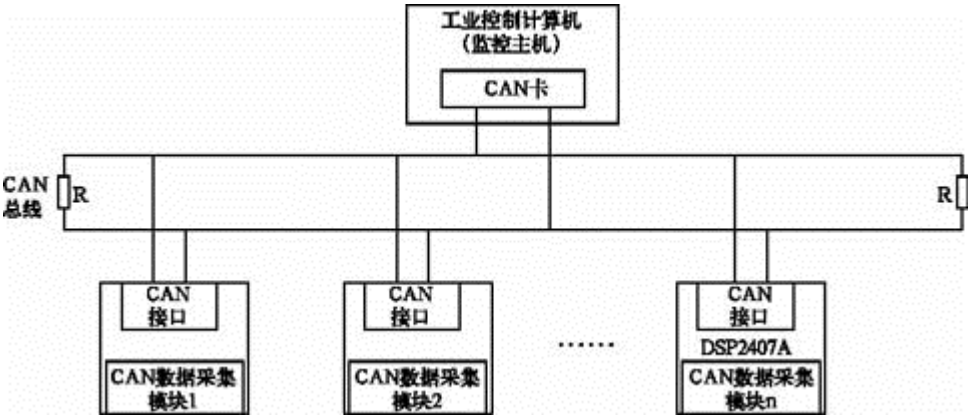

图 1.3　总线应用原理图

1.5 RS422 到 CAN 总线转换器设计

1.5.1 RS422 到 CAN 总线转换器意义

由于船舶中的很多底层控制仪器、智能仪表和可编程控制器起初都是在 DCS（集散式控制系统）的模型下传输数据的，都带有 RS422 串行数据输入输出接口，如果想把它们连接进安全性更高，实时性更好的 CAN 总线网络，那么将所有的底层控制仪器、智能仪表和可编程控制器都换成带有 CAN 接口的仪器成本很高，且要将所有的一起仪器仪表全部更换，费工费时。这时使用一个能把 RS422 转换成 CAN 的转换器就可以解决实际问题，它是一种模块化的转换器，直接使用仪器仪表上的 RS422 接口，不需要对原先任何器件进行硬件处理，就可以把底层仪表接入 CAN 总线，更有优

势的地方是它并不是一对一的，将一路的 RS422 数据与一路 CAN 数据进行转换，而是多对一的，也就是使用一个控制芯片可以转换多路 RS422 数据。转换成本将会大大降低，并且充分使用硬件资源。这种转换器在船舶导航控制、及其他的 CAN 总线网络的设计与选择中将会有很重要的作用与应用空间。

1.5.2 RS422 到 CAN 总线转换器设计要求

我们所要求设计的 RS422 到 CAN 总线转换器是基于组合导航系统的应用。它是连接组合导航系统中各个智能仪表，测控元件与工控计算机还有人工控制系统的线路，相当于系统中的交通运输线，在导航系统及其它相互连接的自动化网络系统中发挥着重要的作用。具体的设计要求如下：

1 能从各个节点、智能仪表仪器发过来的串行数据存入 C8051F040（所选泽的带有 CAN 接口的 51 单片机）中，因为发过来的数据并不只是一路，因此所作的器件要有自动辨别是由哪路串口进来的数据，并且将数据分别存入指定的存储区域中。

2 能将由符合 RS422 的数据转换成符合 CAN 总线的帧结构。并且当 CAN 总线中的其他节点需要此数据时，将数据从存储区中取出并实时的发送出去。

3 此转换器属于自动工作器件，因此高安全性时期要保证的重要的一个方面，当系统出现故障或跑飞的时候，能够自动重新工作，即具有自动重启功能。

4 结构简单，成本低廉，安装方便。由于船舶在航行过程中要不断的根据外界环境的变化来实时控制船舶的航行，而且由于船体的特殊的构造，及用途不同，船体的内部结构也不尽相同，使得船舶上面的各个部门的控制人员要实时的分享所需要的信息变得十分困难，这就阻碍了船舶的实时控制。所以设计方向应该是简单实用。结构简单，便于使用和修理；成本低廉，是要面向各

种类型的船舶；安装方便，可以在船舶上实时的使用，不需要专业人员、专业技术。

5 模块化设计，设计的转换器在硬件上不需要依靠其他仪器进行工作，是独立的，这样当转换器出现故障来不及修理时，可以用一个备用的先换上，不影响船舶内的通讯与工作。换下来的修好后可当备用的转化器使用。

1.5.3 RS422 到 CAN 总线转换器设计方案

根据具体的设计要求，我设计了一个方案，经过老师和研究生的指导，略有改动，并最终确定了设计方案。设计方案力求简单、实效、稳定使设计出来的产品具有结构简单、成本低廉、安装方便的优点。具体的设计方案如下：

用一片C8051F040单片机作为转化器的中心芯片，它来完成转换器的中心工作.它完成三种功能，一是完成软件方面的功能，能把RS422协议传输的字节和按CAN协议传输的帧相互转换；二是完成逻辑电平对CAN总线逻辑电平的转换和其他的CAN通讯协议的转换；三适当系统遇到软件或硬件故障时，能重新自动复位，使其能在无人情况下更加稳定地自动工作。PCA82C50是飞利浦公司的CANX总线收发接口电路，借此可扩大在负载能力，可支持多大110个节点相连接。串行接收发送方面一片TL16C554C串行通讯扩展单片机接收从四个MAX1490A传输过来的数据。MAX1490A完成RS422总线逻辑电平对TTL逻辑电平的转换。这样就大体上形成了转换器的结构，我也就是按照这一结构的设计方案来设计转换器的。

本章总结

绪论部分主要介绍的是当前在网络应用中的各种总线方式，如RS422总线和CAN总线，比较了他们各自的优缺点，以及发展趋势。在当前环境下，虽然DCS集散控制系统仍然占有市场较大份额，但现场总线的发展趋势是阻止不了的，通讯网络形成真正的开放

互联系统已是大势所趋。RS422总线到CAN总线的转换器正是这种条件下，有DCS向FCS（现场总线控制系统）转换下的产物，具有很高的市场应用价值。目前市场上已有这种转换器，四星电子BH-485 RS422/RS485/CAN接口保护器，中牟源动力科技的光电隔离型RS422/485中继器，但这种产品只是一对一的转换，而这次设计的一个转换器可以进行四路RS422串行数据的转换，充分利用了硬件资源，节约成本，具有更高的应用价值。

第二章 转换器硬件设计

从硬件考虑，转换器应能把符合RS485/RS422的标准的逻辑电平和符合CAN逻辑的电平相互转换，从软件考虑，桥接器应能把按RS485/RS422协议传输的字节和按CAN协议传输的帧相互转换。芯片MAX1490A完成RS422总线逻辑电平对TTL逻辑电平的转换，芯片TL16C554完成异步通讯串行通讯口的扩展，CF8051F040是带CAN控制器的通讯模块，它完成三种功能，一是完成软件方面的功能，能把RS485/RS422协议传输的字节和按CAN协议传输的帧相互转换；二是完成逻辑电平对CAN总线逻辑电平的转换和其他的CAN通讯协议；三是充当看门狗，自动复位。PCA82C50是飞利浦公司的CAN总线收发接口电路，扩大在负载能力。

2.1 单片机及所用器件简介

众所周知，近 10 年来微型计算机的发展速度是十分迅速的，其发展方向有两个方面：其一是不断推出高性能的通用微型计算机系统。这类通用微机系统，其中央处理机为一块大规模的集成电路，称为微处理器，配上数量不等的外围接口电路和内存电路，组成主机，加接终端、打印机、磁盘等 I/O 设备，构成微机系统。如 Z80 系列的 TRS-80,CROMEMCO,6502 的 APPLE,IBMPC,550 等。还有各种个人计算机和单板机，它们正取代原中小型的甚至大型的计算机的作用，并使计算机普及到各个行业甚至是家庭。通用微型机主要用作科学计算和管理。随着大规模集成电路技术的发展，微处理器的功能不断得到加强：

（1） 字长：从 8 位、16 位，到 32 位；

（2） 提高 CPU 的处理能力和处理速度；

（3） 采用先进的系统结构，适合于组成微机网络；

其二是面向控制型应用领域的单片微型计算机（亦称微控制器）的大量生产和广泛应用。世界上著名的器件公司如INTEL,ZILOG,NEC 等大公司都生产单片微型计算机系列。由于单片机具有高可靠性、超小型、价格低、容易产品化等优点，单片机在智能仪器仪表、实时工业控制、智能终端、通信设备、导航系统、家用电器等控制型应用领域具有广泛的用途，正取代各种类型的专用计算机系统。单片机与 CAN 总线连接的传统方式是将 CPU 与总线控制器和总线收发器相连后再接入总线网络，这样使 CPU 外围电路复杂化，整个系统受外部影响较大。为了解决这一问题，很多单片机厂商纷纷将 CAN 控制器集成在单片机芯片上。目前，单片机内部集成的 CAN 控制器有 Motorola 公司的 MC68HC912DG128A、Philips 公司的 P87C591、Atmel 公司的 AT89C51CC01 和 AT89C51CC02、Intel 公司的 TN87C196CA 和 TN87C196CB 以及 Cygnal 公司的 C8051F040 等。由于我们应用的主要是 C8051F04X 系列的单片机，下面我就来介绍一下 C8051F04X 系列单片机的特性。

2.1.1 C8051F040 单片机的特点和应用

C8051F040单片机是美国Cygnal公司生产的完全集成的混合信号系统级芯 SoC(System on Chip)，具有与 8051 指令集完全兼容的 CIP-51 内核。它在一块芯片上集成了构成一个单片机数据采样或控制系统所需要的几乎所有模拟和数字外设及其它功能部件。功能部件包括模拟多路选择器、可编程增益放大器、ADC、DAC、电压比较器、电压基准、温度传感器、SMbus/I2C、UART、SPI、CANbus 2.0B、可编程计数器/定时器阵列(PCA)、计数器/定时器、8b 宽的通用数字 I/O 端口、电源监视器、看门狗定时器(WDT)和时钟振荡器等。此单片机有 64Kb 在系统可编程 FLASH 程序存储器和 4352b 的 RAM，执行速度可达 25MIPS。片内 JTAG 仿真电路提供

全速、非侵入式(不使用片内资源)的电路仿真，支持断点、单步观察点、运行和停止等调试命令，支持存储器和寄存器校验和修改。图 2.1 为 C8051F040 芯片结构

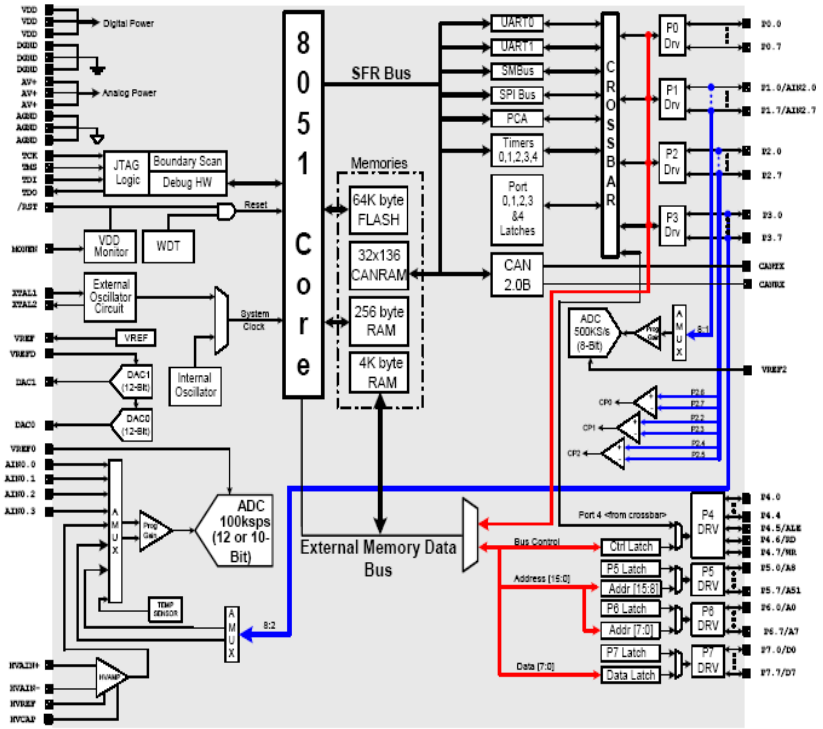

图 2.1 C8051F040 芯片结构

C8051F040 片内带有 CAN 控制器，符合 Bosch 2.0A 和 2.0B 技术规范，允许使用 CAN 协议进行串行通讯。CAN 控制器的工作速率可高达 1Mbit/s。Cygnal 公司的 C8051F04X 的 CAN 有 32 个信息目标，每个信息目标都有自己的标识符屏蔽用来对接收信息进行接收滤波。输入数据，信息目标和标识符屏蔽都存储在 CAN 信息 RAM 中。所有进行数据传输和接收滤波的协议功能都是由 CAN 控制器执行的，而微控制器单元 MCU 在此并不起作用。这样，使用 CAN 通信需要很小的 CPU 带宽。通过 MCU 内的特殊功能寄存器，MCU 可配置 CAN 控制器，访问接收到的数据以及传输数据。因此，C8051F040 可完成 CAN 总线协议的数据

链路层和应用层的所有功能，图 2.2 为 CAN 控制器内部结构

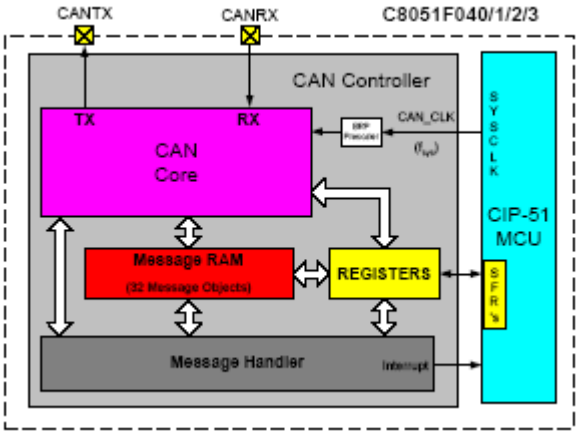

图 2.2 CAN 控制器内部工作结构

C8051F04X 系列芯片自带看门狗定时器，在系统时钟下工作，看门狗（WDT）溢出将会迫使芯片进入复位状态。为了阻止复位，WDT 必须在溢出之前利用应用程序重新启动，如果系统遇到一个硬件或软件故障，阻止系统重新启动 WDT，WDT 将会溢出并引发复位，这可以组织系统跑飞。这样拥有硬件定时复位功能，相对于软件定时复位抗干扰能力更强；可以不需任何外围器件自动进行可靠的上电复位，并且对软件运行状况和外界电源状况进行监视；实现程序失控后按自己设想的程序恢复执行，提高了系统的抗干扰能力，保证了系统的安全性和数据可靠性。CAN 内核由 CAN 协议控制器和负责消息收发的串行 / 并行转换 RX / TX 移位寄存器组成。消息 RAM 用于存储消息目标和每个目标的仲裁掩码。这种 CAN 处理器有 32 个随意配置为发送和接受的消息目标，并且每一个消息目标都有它自己的识别掩码，所有的数据传输和接收滤波都是由 CAN 控制器完成的， 而不是由 CIP-51 来完成。这样就可以减少 CAN 的处理时间。

CAN 内部寄存器中存储了所有 CAN 的控制和配置信息,其中包括控制寄存器、状态寄存 器、设置波特率的位定时寄存器 、测试寄存器、错误计数器和消息接口寄存器。通常 CAN 内核不能直接访问消息 RAM,而必须通过接口寄存器 IF1 或 IF2 来访问。另外,CIP-51 的 SFR 并不能直接访问 CAN 内部寄存器的所有单元,其配置 CAN、消息目标、读取 CAN 状态以及获取接收数据、传递发送数据都由 SFR 中的 6 个特殊寄存器 来完成,其中 CANOCN、CANOTST 和 CANOSTA 3 个寄存器可直接获取或修改 CAN 控制器中对应的寄存器,而 CANODATH、CANODATL、CANOADR3 个寄存器主要用来访问修改其它不能直接访问的 CAN 内部寄存器,其中 CANOADR 用来指出要访问寄存器的地址,CANODATH、CANODATL 这时就相当于要访问的 1 6 位寄存器的高、低字节的映射寄存器,而对它们的读写则相当于对所指向寄存器的读写。图 2.2 给出了 CIP-51 如何访问 C A N中控制寄存器和每个消息的路径图。

2.1.2 PCA82C250 总线收发器

由于Cygnal的C8051F040内部的CAN控制器只是个协议控制器,不能提供物理层驱动,所以在使用时还需外加CAN总线收发器,常用的CAN总线收发器有Philips公司的PCA82C250收发器、高速TJA1050收发器等。这里使用的是PCA82C250收发器.它可提高总线的差动发送和接收能力。它与ISO11898标准完全兼容,有三种不同的工作方式,即高速、斜率控制和待机,可根据实际情况选择。位速度可高达1Mbit/s。PCA82C250能驱动大量的总线节点即每个网络64到100个同时总线长度可达0.5 到1km。图2.3为PCA82C250器件外观图

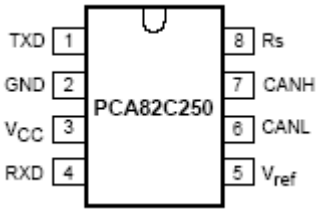

图2.3 PCA82C250器件外观图

PCA82C250 提供了一个参考电压输出VREF，它可以用于偏置CAN 协议控制器差动输入比较器的其中一个输入参考电压，在协议控制器输入处产生系统使用哪个解决方法更适合由应用和协议控制器的总线输出结构决定。

PCA82C250共有三种不同的工作模式通过控制Rs引脚选择是那一种模式。

第一种模式是高速模式：它支持最大的总线速度和长度。

第二种是斜率模式：当使用非屏蔽的总线电缆时可以考虑使用这种模式的输出转换速度可被故意降低以减少电磁辐射。

第三种是准备模式：这种模式在电池供电的应用要求系统功率消耗非常低的应用中非常有用在准备模式中传输一个报文就可以将系统激活，是收发器在准备模式和普通工作模式间切换的例子。

CAN 总线网络中可使用的最大总线线路长度基本是由下面的物理条件作用决定：

1. 连接的总线节点CAN 控制器收发器等等的循环延迟以及总线线路的延迟。

2. 由于节点间相关的振荡器容差造成位定时额度的不同。

3. 由于总线电缆的串联阻抗和总线节点的输入阻抗使信号的幅值下降。

PCA82C250可以在高速模式下使用下面的总线电缆长度其中为了获得最大的传播延迟CAN 位定时参数可以优化位速度与

总线长度的关系，表2.1为总线长度与位速率的关系。

位速度（kbit/s）	总线长度（m）
1000	30
500	100
250	250

表2.1 总线长度与位速率的关系

C8051F040与PCA82C2的通讯硬件模块硬件电路度如图2.3所示。

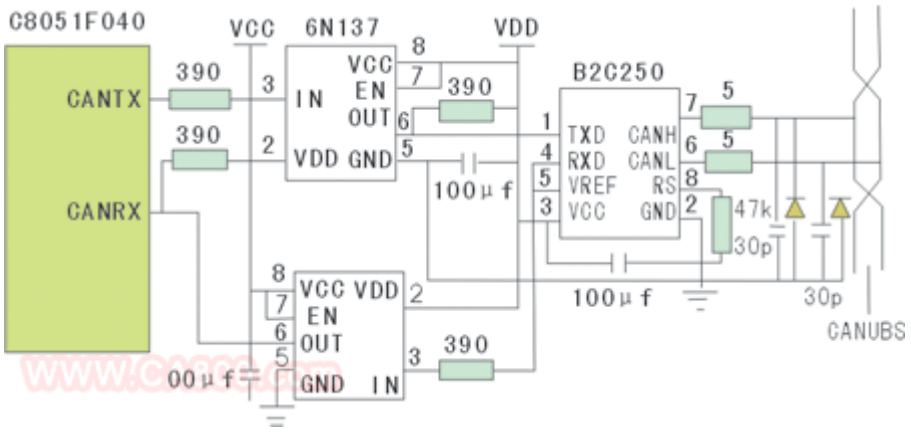

图 2.3:总线 CAN 接口电路图

为了进一步提高系统的抗干扰能力，具体的连接方式为：在CAN 控制器引脚 CANTX、CANRX 和收发器 PCA82C250 之间并不是直接相连，而是通过由高速光耦 6N137 构成的隔离电路后再与 PCA82C250 相连，这样就可以很好的实现总线上各节点的电气隔离。这部分增加了节点的复杂性，但它却提高了节点的稳定性和安全性。

在 PCA80C250 与 CAN 总线接口部分也采用了一些安全和抗干扰措施。PCA82C250 的 CANH 和 CANL 引脚各自通过一个 5Ω的电阻与 CAN 总线相连，电阻可起到一定的限流作用，从而保护 PCA82C250 免受过流的冲击。在 CANH 和 CANL 与地之间各自接一个 30pF 的小电容，可以起到滤除总线上的高频干扰和防电磁辐射的能力。另外，在 CANH 和 CANL 之间并联一个 15V 的瞬

态电压抑制二极管（TVS），可以保护 PCA80C250 在瞬间高电压情况下而不受损坏。PCA82C250 的 RS 脚上接有一个下拉电阻，电阻的大小可根据总线速率适当的调整，其值一般在 16kΩ～140kΩ之间，图 2 中选用 47kΩ。C8051F040 供电电源为 2.7V～3.6V,其所有 I/O 口允许 5V（极限值为 5.8V）输入，但是 I/O 输出电平为 VDD。而 PCA82C250 为 5V 系统，为了能够驱动其工作，在 CANTX 引脚上拉一上拉电阻，其值为 4.7kΩ。

2.1.3 通讯扩展器件 TL16C554

在异步串行通信中，一般都要求设备的串行口要符合RS232、RS485/422等串行通信标准，并按该标准定义的信号进行信息控制和数据传输，能够实现该功能的集成电路通常称为UART，即通用异步接收器／发送器。利用C8051F040构成主从式多机串行通信系统时发现，主机C8051F040要与四个RS422器件进行串行通讯，此时便有必要进行串口扩展，此时采用4通道异步收发器TLI6C554显然是一种较好的选择。TLI6C554是TI公司生产的4通道异步收发器集成芯片，在FIFO模式下，传输和接收前将数据缓冲为16字节数据包，减少了CPU的中断数量。内部包含4片改良的16C550异步传输器件，使得串行I／O更加可靠，每个信道可实现串行和并行两种连接方式的转换，每个信道的状态可以通过CPU的操作读取，可以获取操作情况或任何的错误状态。三态输出为双向数据总线和控制总线提供1TrL驱动能力、优先级中断系统控制、可编程的串行接口特性。TLI6C554的主要特点如下：内部包含4片TLI6C55~最高可达1M的波特率，且波特率发生器可用软件设定；带有16字节带错误标志的接收FIFO缓冲器；具有可编程的串行数据发送格式，数据位长度为5、6、7、8；具有偶校验、奇校验或无校验模式，停止位长度为1、1.5、2；具有标准的调制解调器接口；有68引脚PLCC和80引脚TQFP两种封装形式。TL16C554的引脚功能TL16C554

的引脚排列如图2.4所示，其主要引脚的功能如下：A0～A2为片内寄存器的选择信号端；D0～D7为双向81,2数据线；IOR为读选通端(低电平有效)；IOW为写选通端(低电平有效)；INTA～INTD为中断，A、B、C、D端(高电平有效 CSA～CSD为片选A、B、C、D(低电平有效)XTAL1、XTAL2为外部时钟端，该两脚可接晶振或外部时钟信号；RXA～RXD为串行输入A、B、C、D端；TXA～TXD为串行输出A、B、C、D端；RXRDY为接收准备好信号端(低电平有效 TXRDY为发送准备好信号端(低电平有效)。

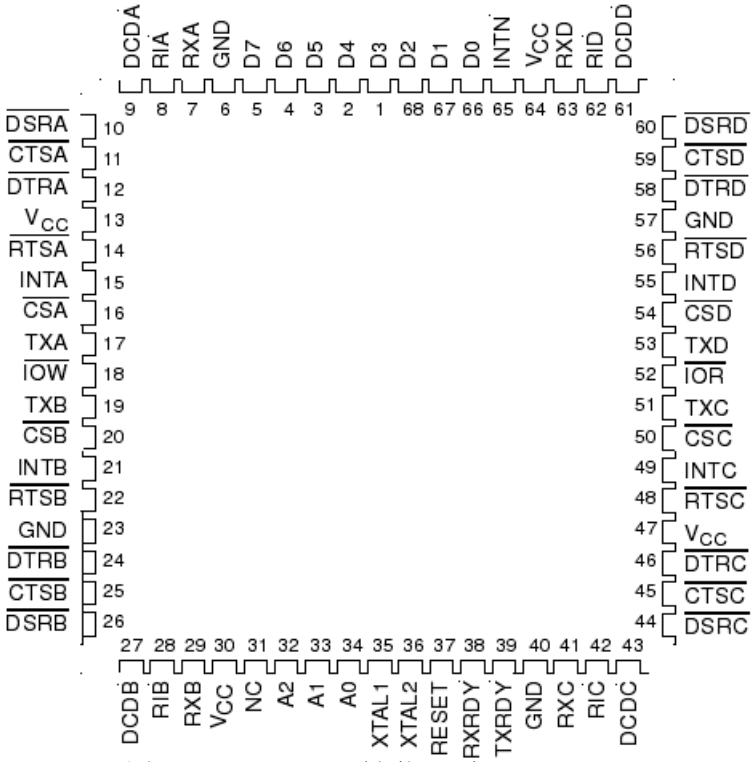

图2.4 TL16C554封装形式

TL16C554的数据线(D7～D0)可直接与CPU的数据总线的低8位相连，它们是UART的数据输入和输出通道。它的参考时钟既可以由外部提供，也可以通过一个晶振在内部产生。而UART与CPU之的通信控制通过一组信号线来实现，这组信号线包括复位控制RESET、通道选择、寄存器选择、中断请求、读选通端IOR和写选

通端IOW等。MODEM控制逻辑主要用于完成UART与RS232之间的接口通信，这些信号通过EIA驱动器驱动后均符合RS232标准。TL16C554的片内寄存器 ACE内部共有11个寄存器，这些寄存器分别用于实现通信参数的设置、对线路及MODEM的状态访问、数据的发送和接收以及中断管理等功能。可由A0、A1、A2三条片内寄存器选择线和线路控制寄存器的除法数锁存器访问位DLAB一起通过多路复用进行访问或控制ACE的任何一个寄存器。表2为访问这些寄存器时DLAB和A0、A1、A2的状态。

DLAB	A2	A1	A0	寄存器
0	0	0	0	接收/发送缓冲寄存器（RBR/THR）
0	0	0	1	中断允许寄存器（IER）
X	0	1	0	中断识别寄存器（IIR）
X	0	1	0	FIFO控制寄存器（FCR）
X	0	1	1	线路控制寄存器（LCR）
X	1	0	0	Modem控制寄存器（MCR）
X	1	0	1	线路状态寄存器（LSR）
X	1	1	0	Modem状态寄存器（MSR）
X	1	1	1	高速暂存寄存器（SCR）
1	0	0	0	波特率因子寄存器低位（DLL）
1	0	0	1	波特率因子寄存器高位（DLM）

表2 寄存器地址

因为接收发送缓冲寄存器的DLAB、A0、A1、A2各位都相同，因此还必须通过读/写信号来加以区分。利用TL16C554扩C8051F040串口的硬件电路由于异步串行通信中，字符的到达时间及间隔时间往往无法预测，此时如果采用查询方式将会浪费大量的CPU时间，降低系统的性能。在实际设计中采用中断方式，当串行口接

收到一个新的字节符或者前一个字符发送结束时，UART以中断方式通知CPU，这样可以大大提高系统的实时性。数据线D0～D7直接与C8051F0404的P0口相连，TL16C554的片内寄存器选择线A2—A0接51的低位地址P1.4-P1.6，此时C8051F040对TL16C554片内寄存器的读写操作同对外部RAM读写一样。TL16C554的片选线CSA—CSD分别接P2.0—P2.3，这样可以通过对P2.0-P2.3的清零来选通通道A～D。TL16C554的中断线INTA—INTD分别与C8051F040的P2.4—P2.7相连，并经过一个四输入或非门接C8051F040的INT1。这样当多个串口同时使用时，任一个指定通信事件的出现都将使UART产生一次通信中断，并触发外部INT1中断。在中断服务子程序中，通过对P2.4～P2.7的检测来确定中断到底来自于哪个口，并完成相应的数据处理。

TL16C554软件设计的初始化和数据收发程序等。初始化的主要任务是设置操作所需要的参数，这些参数包括串行通信时数据串的数据位数、停止位数、奇偶校验等。另外还需要设置发送和接收的波特率及中断方式。本系统中，波特率设置为9600，波特率因子寄存器的高位为00H，低位为0CH。通信格式为8位数据位，1位停止位，奇校验，线路控制寄存器的值LCR为0BH。系统工作在中断方式，应允许接收就绪中断和发送缓冲中断，相应地，中断允许寄存器的值就设置为03H。

2.1.4 RS422 转换器件 MAX1490A

MAX1490A 是一种具有完全隔离能力的 RS422 接口芯片，它的 TTL 侧和 RS422 侧实现了完全的电隔离。其中，A 和 B 为 422 侧驱动输入端，Y 和 Z 为 422 侧驱动输出端，DI 为隔离的 TTL 侧驱动输入端，RO 为隔离的 TTL 侧驱动输出端。MAX1490A 内部具有 DC/DC 隔离变换器，无需外部提供隔离电源，只需一个单一的 5V 电源。MAX1490A 结构简图如 2.5 所示

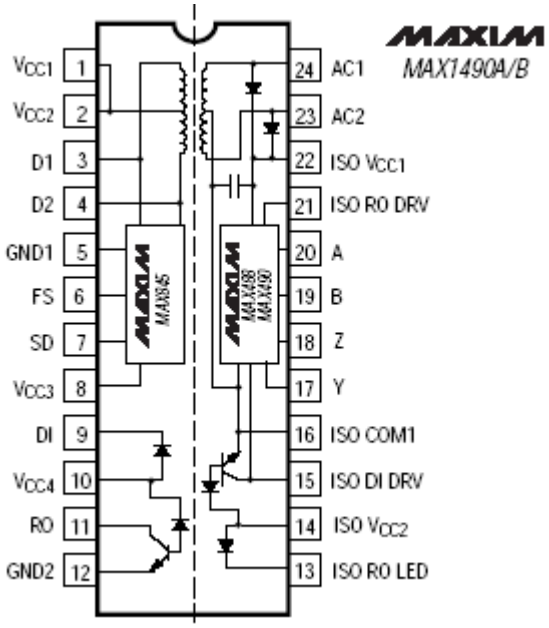

图 2.5 MAX1490A 结构简图

2.2 电路设计软件 PROTEL99 的使用方法

2.2.1 PROTEL99 简介

随着电子技术的飞速发展，各种新型电子器件和集成电路应用越来越广泛，电子系统能越来越强大，电路图也越来越复杂，印刷电路板的最先也越来越复杂和精密，以往用手工的方法绘制电路原理图和设计线路板已很难适应当前电子工业飞速发展的形势。所幸的是计算机的发展和普及较好的解决了这个问题。目前，人们可以在计算机上利用各种商品化的软件对电子线路进行有效的分析与设计。早期的市场上可以看到 EE System、PCAD、TANGO 等电子设计软件及 Protel Technology 公司推出的 Protel for DOS 版本软件。但进入 20 世纪 90 年代，计算机行业放生了翻天覆地的变化。尤其在微软公司推出 Windows 95 操作系统以后，DOS 操作系统就退居幕后。Protel 99 主要包含两大部分：（1）原理图设计系统（Advanced Schematic），它主要用于电路原理图的设计。（2）印刷电路板设计系统（Advanced PCB），它主要用于印刷电

路板的设计。

2.2.2电路板设计步骤

一般而言，设计电路板最基本的过程可以分为三大步骤

1.电路原理图的设计

电路原理图的设计主要是PROTEL99的原理图设计系统（Advanced Schematic）来绘制一张电路原理图。在这一过程中，要充分利用PROTEL99所提供的各种原理图绘图工具、各种编辑功能来实现我们的目的，即得到一张正确、精美的电路原理图。

2.产生网络表

网络表是电路原理图设计（SCH）与印制电路板设计（PCB）之间的一座桥梁，它是电路板自动灵魂。网络表可以从电路原理图中获得，也可从印制电路板中提取出来。

3.印制电路板的设计

印制电路板的设计主要是针对PROTEL99的另外一个重要部分PCB而言的，在这个过程中，我们借助PROTEL99提供的强大功能是想电路板的版面设计，完成高难度的设计工作。

原理图的设计可按下面过程来完成。

1. 设计图纸的大小

Protel99/Schematic后，首先要构思好零件图，设计好图纸的大小。图纸大小是根据电路图的规模和复杂程度而定的，设计合适的图纸大小是设计好原理图的第一步。

2. 设置Protel99/Schematic设计环境

设置Protel99/Schematic设计环境包括设置个点大小和类型，光标类型等等，大多数参数也可以使用系统默认值。

3. 旋转零件

根据电路图的需要，将零件从零件库里取出放置在图纸上，并对放置零件的序号、零件封装进行定义和设定等工作。

4. 原理图布线

利用Protel99/Schematic提供的各种工具，将图纸上的元件用有电气意义的导线、符号连接起来，构成一个完整的原理图。

5. 调整线路

将初步绘制好的电路图作进一步的调整和修改，使得原理图更加美观。

6. 报表输出

通过Protel99/Schematic提供的各种报表工具生成各种报表，其中最重要的是网络表，通过网络表为后续的电路板设计作准备。

2.3 硬件电路图的设计

了解了各个芯片的工作原理和应用和 PROTEL 99 的使用方法之后，就可以进行原理图设计了。利用四片 MAX1490A 与从仪器仪表传过来的已经经过 A/D 转换的串行数据传送进处理芯片（C8051F040）中，利用 TL16C554 穿行扩展芯片选择是哪一路的数据进来，具体的设计部分已在 2.1.3 小节中描述。从 C8051F040 的 CAN 接口连接 PCA82C250 以次扩大总线带负载能力。为了提高其抗干扰能力在 C8051F040 与 PCA82C250 之间连接高速光耦 6N137 构成隔离电路（具体见 2.1.2 PCA82C50 总线收发器）.总体设计之后的电路原理图见附件 2。

本章总结

硬件电路设计是整个转换器的基础，以后的软件设计也是根据硬件电路而设计的，硬件电路的好坏决定了系统的稳定性，硬件上的考虑周到可以使软件设计上少很多麻烦，并且使系统更稳定。通过转换器的硬件部分，使我对硬件设计有了一定的认识。提高硬件系统的稳定性需要做到以下几点：

1. 与外界容易产生干扰的输入接点、继电器的输出接电等器件部分，采用光耦隔离，使外界杂散信号无法干扰 CPU 的运行，即易受干扰的输入或输出接点以光耦隔离。

2. 没有使用到的端口引脚应接到一个固定逻辑电位上（0 或 1），以免受到外界的静电干扰，导致 CPU 运行失常而产生死机。

3. 易受杂散信号干扰处，则接一个 0.01μF 的树脂电容到机体外壳，使杂散信号的尖峰毛刺经此电容后到外壳。

4. 每一颗 IC 的 VCC 与 GND 之间接 0.01μF-0.1μF 的积层电容，以使电源电压波的波纹及杂散信号有所旁路，不至影响该 IC 的正常运行。同时也可抵消导电电路的电感性，使整个电路具有较佳的稳定性。

5.直流电源输入侧，最好加高频波滤波器，阻止从交流电源端来的各种高频波的干扰。

6.石英晶体的两脚越短越好,越接近 C8051 的 XTAL1 和 XTAL2 脚振荡效果越好，越稳定。

绘制电路图时主要注意的方面：

1. 总体电路图尽可能画在一张纸上；同时注意信号的流向，一般是从输入端画起，由左至右或由上至下按信号的流向一次画出个单元电路；对于电路图比较复杂的，应将主电路画在一张图上，而讲其余的单元电路画在一张或多张图纸上，并在各图所有端口两端标注上标号，依次说明各图纸之间的连接关系。

2. 注意电路图的紧凑协调，要求布局合理，排列均匀。图中元器件的符号应该标准化，元件符号旁边应标出型号和参数。集成电路用方框表示，并标出每跟连线的功能和管脚号。

3. 连线一般画成水平线和垂直线，并尽可能的减少交叉和拐弯。对于连接电源负极的连线一般用接地符号表示；对于连接到电源正极的连线，仅需标出电源的电压值。

第三章　转换器的软件设计

3.1 单片机软件程序设计过程

在应用单片机语言（即汇编语言）编写程序时，要根据问题要实现的目标，如被控对象的功能和工作过程要求，首先设计硬件电路；然后再根据具体的硬件环境进行程序设计。 具体应该分为以下几个部分：

1. 分析问题。要对需要解决的问题进行分析，以求对问题有正确的理解。
2. 确定算法。算法就是如何将实际问题转化成程序模块来处理。
3. 画程序流程图。一个程序按其功能可分为若干部分，通过流程图把具有一定功能的各部分有机联系起来。从而使人能够抓住程序的基本线索，对全局有完整的了解。因此，画程序流程图是程序结构设计时采用的一种重要手段。有了程序流程图，可以很容易的把较大的程序分成若干个模块，分别进行设计，最后合在一起联调。
4. 编写程序。编写出相关的程序。

编好的C语言源程序必须转换成机器码表示的目标程序，才可以执行，这种转换过程称为汇编。对单片机来说，有手工汇编和机器汇编两种方法。其中，手工汇编时当进行偏移量的计算或程序的修改，就会遇到很多的问题，稍不注意就会出错。而机器汇编由于速度快，准确度高，现在被广泛的应用。

3.2 所需各种器件的软件和协议应用

C8051F040具有64KB Flash、4352B RAM、CAN控制器2.0、2个串行接口、5个16位定时器、12位A/D转换器、8位A/D转换器及12位D/A转换器等，它内部还带有JTAG接口，使调试变得非常方便。C8051F040内部集成的CAN控制器为Bosch CAN控制器。此CAN控制器有以下几部分构成：CAN

内核、报文 RAM（与 C8051 RAM 相互独立）、报文处理状态机制和 CAN 控制寄存器。其结构框图如图 3.1 所示。

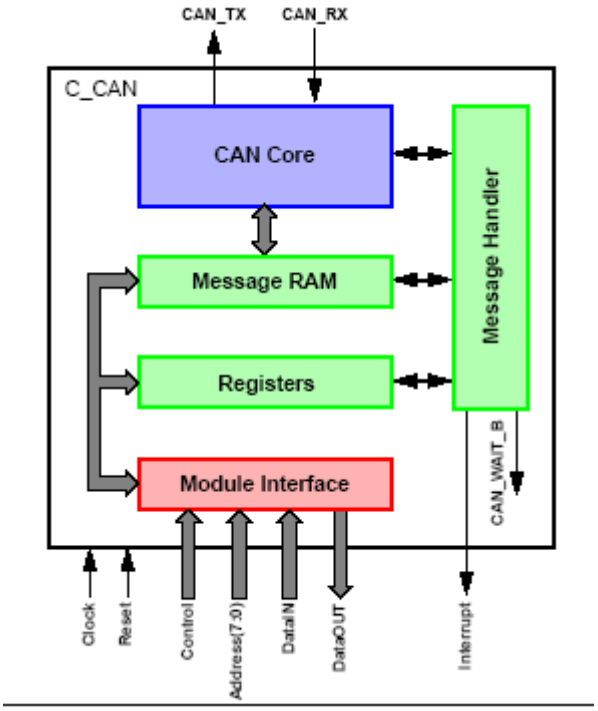

图 3.1 CAN 结构框图

在 CAN 控制器里只有三个寄存器可通过 CIP-51 中的特殊功能寄存器直接访问，其它的寄存器只能通过 CAN0ADR、CAN0DATH 和 CAN0DATL 寄以地址索引的方式间接访问。在使用 CAN 控制器时，重点和难点是对 CAN 控制器的寄存器的使用，其内部寄存器的分类及其主要功能如下：

（1）CAN 控制器协议寄存器

该协议寄存器是用来配置 CAN 控制器，处理各种中断，监控总线状态以及置控制器为测试模式。CAN 控制器协议寄存器可使用 C8051 MCU 特殊功能寄存器通过索引方式间接访问，其中有些还可以很方便的通过 C8051 内部特殊功能寄存器直接寻址来访问。这部分的寄存器有：CAN 控制寄存器

（CAN0CN）、CAN 状态寄存器（CAN0STA）、CAN 测试寄存器（CANTST）、错误计数寄存器、位定时寄存器和波特率因子扩展寄存器。其中，CAN0CN、CAN0STA 和 CANTST 可通过 C8051 MCU 特殊功能寄存器直接访问，其它的只能通过间接访问。

（2）报文对象接口寄存器

CAN 控制器中有两组报文对象接口寄存器，它们用来配置报文 RAM 中 32 个报文对象是用来向 CAN 总线发送数据，还是从 CAN 总线接收数据。当其中的一组被设置为向报文 RAM 中写数据，另一组则从报文 RAM 中读取数据。利用此接口寄存器可以避免 CPU 访问报文 RAM 与 CAN 报文接收和发送缓冲转移之间的冲突。所有的报文对象都存储在报文 RAM 里面，通过报文对象寄存器对其进行访问和配置，这些寄存器要通过 C8051 的 CAN0ADR 和 CAN0DAT 寄存器，使用间接索引地址方式来访问。

（3）报文处理寄存器

所有的报文处理寄存器都是只读寄存器。通过读取它们的值可以实时地判断相应报文对象的状态，从而使 CAN 控制器能正确运行。它们的标识位由 CAN0ADR、CAN0DATH 和 CAN0DATL 通过索引方式间接来访问。报文处理寄存器提供中断、错误、发送/接收请求和新数据信息。这部分的寄存器包括：中断寄存器、发送请求寄存器、新数据寄存器、中断队列寄存器和报文有效寄存器。

3.3 配置寄存器

那么，通过 CAN0ADR、CAN0DATH 和 CAN0DATL 寄存器以索引方式间接访问 CAN 控制器中寄存器的过程会怎样呢？因为每个 CAN 控制器寄存器都有一个索引号，如果要访问某一 CAN 控制寄存器，只需将此寄存器的索引号写入 CAN0ADR 寄存器，而数据读/写操作通过 CAN0DATH 和 CAN0DATL 来完成。例如：如果需要对位定时寄存器重新配置时，只需向 CAN0ADR 寄存器中写入 0X03，将新配置的数据的低字节写入 CAN0DATL

中，高字节写入 CAN0DATH 中。

下面就 CAN 控制器在应用时，根据所要完成功能的不同而需要做的不同配置做具体描述。这包括报文对象初始化处理、发送对象配置、接收对象配置、中断处理配置；另外，还有发送对象的更新、位定时寄存器等配置。

3.3.1 报文对象初始化处理

报文 RAM 中的报文对象（除 MsgVal、NewDat、IntPnd 和 TxRqst）配置不受芯片复位的影响。所有的报文对象在使用前必须由 CPU 来初始化为零或者被设置为无效。报文对象的配置是通过相应的接口寄存器来设置其屏蔽码、仲裁场、控制场和数据场值，而这一设置过程由相应的 IFX 命令请求寄存器来完成。当 CAN 控制寄存器中的 Init 位置零，CAN 内核中的 CAN 协议控制器状态机制和报文处理状态机制将控制 C_CAN 的内部数据流。接收到的报文通过接收滤波后都存放在报文 RAM 中，而得到传输请求的报文都要移入 CAN 内核的移位寄存器中并通过 CAN 总线传出。初始化程序包括：

1. void clear_msg_objects (void); //清空报文对象
2. void start_CAN (void); //打开 CAN

具体子程序为

```
void clear_msg_objects (void)                    //清空报文对象

{
 void watch_dog(void);                           //清看门狗
 SFRPAGE   = CAN0_PAGE;                          //指向 CAN0 页面
CAN0ADR   = IF1CMDMSK;                           // 指向命令屏蔽寄存器 1
CAN0DATL = 0xFF;              // 设置可以向所有的消息目标中写信号
 for (i=1;i<33;i++)
 {
CAN0ADR = IF1CMDRQST;                            // 复位所有消息
CAN0DATL = i;
```

```
 }
}
//CAN 开始工作
void start_CAN (void)
{
void watch_dog(void)；
SFRPAGE   = CAN0_PAGE;   //指向 CAN0 页面
CAN0CN    |= 0x41;              // 将 CCE 和 INIT 置 1 开始初使化
CAN0ADR   = BITREG;          // 指向位定时寄存器
CAN0DAT   = 0x2640;          // 设置波特率
CAN0ADR   = IF1CMDMSK;   // 指向屏蔽寄存器器 1
CAN0DAT   = 0x0087;          // 设置发送: 写入 CAN 内存, 写数据字节,
                                  // 设置 TXrqst/NewDat, clr IntPnd 位
CAN0ADR   = IF2CMDMSK;   // 指向屏蔽寄存器器 2
CAN0DATL = 0x1F;             // 设置接受 : 读 CAN 内存, 写入数据
                                  //和 clr NewDat and IntPnd 位
CAN0CN    |= 0x06;             // 允许全局中断   IE and SIE 置位
CAN0CN    &= ~0x41;          //清 CCE and INIT 位, 启动 CAN 状态机制
PCAN0=1；                      //设置中断优先级
}
```

3.3.2 发送与接收对象的配置

当报文对象作为发送与接收对象时，仲裁寄存器（ID28-0 和 Xtd 位）将被应用，它们定义了的报文识别符和类型，如果使用 11 位识别符（标准帧），那么使用的是 ID28～ID18，而 ID17～ID0 将被忽视。如果 TxIE 位被置位，则 IntPnd 位在此报文对象被成功发送后被置位；如果 RmtEn 位被置位，在接收到匹配的远程帧将引起 TxRqst 位被置位。若数据寄存器（DLC3-0，

Data0-7）将被使用，TxRqst 和 RmtEn 在数据有效前不会被置位。屏蔽寄存器（Msk28-0、Umask、Mxtd 和 MDir 位）可以用来（UMask= ‘1’）允许相同识别符的数据帧组被接收。

发送对象的配置包括发送对象的配置和接受对象的配置:

```
void can_receive_data(num);                    //CAN 口接收数据
void can_transmit_data(num);                   //CAN 口发送数据
```

具体子程序如下：

```
//初始化接受寄存器
void init_msg_object_RX (char num)
{
void watch_dog(void)；
SFRPAGE   = CAN0_PAGE;
CAN0ADR   = IF1CMDMSK;   // 指向命令屏蔽寄存器
CAN0DAT   = 0x00B8; // 设置为写，除了 ID MASD 和数据位其余都可更改
CAN0ADR   = IF1ARB1;      // 指向仲裁寄存器 1
CAN0DAT   = 0x0000;       // 设置仲裁寄存器 ID = "0"
CAN0DAT   = 0x8004;       // 仲裁寄存器 2  高字节:设置为标准标识符，不
                          //是扩展标识符
CAN0DAT   = 0x0480;       // 禁止远程帧
CAN0ADR   = IF1CMDRQST;   // 指向命令请求寄存器
CAN0DATL = num;               // 设置选择位，用这一位选择数据存储区
}
//初始化发送消息寄存器
void init_msg_object_TX (char num)
{
void watch_dog(void)；
SFRPAGE = CAN0_PAGE;
```

```
CAN0ADR = IF1CMDMSK;   // 指向命令屏蔽寄存器
CAN0DAT = 0x00B2;   // 设置为写，除了 ID MASD 和数据位其余都可更改
CAN0ADR = IF1ARB1;      // 指向仲裁寄存器 1
CAN0DAT = 0x0000;       // 设置仲裁寄存器 ID  为最高优先级
CAN0DAT = 0xA000;       //设置为标准标识符
CAN0DAT = 0x0081;       // 禁止远程帧
CAN0ADR = IF1CMDRQST; // 指向命令请求寄存器
CAN0DAT = num;          // 设置选择位，用这一位选择数据存储区
}
```

3.3.3 中断处理

在所有中断中，状态中断具有最高优先级，报文对象的中断优先级随着报文编号的增大而减小。如果有几个中断产生，那么 CAN 中断寄存器将指向优先级最高的中断，而不是按中断先后顺序排列。

状态中断通过读取状态寄存器来清除，报文中断通过清除报文对象的 IntPnd 位来清除。处于中断寄存器中的中断识别符 Intld 能表明中断的原因，如果这个寄存器的值为 0，没有中断产生；否则，有中断发生。

CPU 控制着状态寄存器的改变是否可以引起中断（CAN 控制寄存器中的 EIE 和 SIE 位）；当中断寄存器的值不为 0（CAN 控制寄存器中的 IE 位）时中断队列是否有效。CPU 有两种方式判断报文中断源，每一种是判断中断寄存器中的 Intld 位；另一种是顺序扫描中断发生寄存器。具体子程序如下：

```
//CAN  总线中断程序
void CAN_ISR (void) interrupt 19
{
  void watch_dog(void);
int status= CAN0STA;
  if ((status&0x10) != 0)       // RxOk  置位,中断由接收引起*/
```

```
 {
void watch_dog(void);
CAN0STA = (CAN0STA&0xEF)|0x07;   // 将 RxOk 复位，置 LEC 没有改
变   void can_receive_data(num);            //CAN 口接受数据
void tl16c554_transmit_data(num);          //tl16c554 口发送数据
{
if ((status&0x08) != 0)                     // TxOk 置位，中断由发送引起
 {
 void watch_dog(void);
 CAN0STA = (CAN0STA&0xF7)|0x07; // 将 TxOkk 复位，置 LEC 没有改变
 void can_transmit_data(num);               //CAN 口发送数据
}
 if (((status&0x07) != 0)&&((status&0x07) != 7))   // 错误中断, LEC 变化
 {
void watch_dog(void);
 CAN0STA = CAN0STA|0x07;
 }
}
```

3.3.4TL16C554 初始化

TL16C554系统软件设计包括TL16C554的初始化和数据收发程序，设置波特率，中断等。子程序为：

```
// 初始化 TL16C554
void init_TL16C554(void)
{
void watch_dog(void);                  //清看门狗
CSA_LCR=0x9b;                  //设置 LCR 线形控制寄存器，设定字长，停
                                  //止位，奇偶校验和 DLAB 位设置 LCR
```

```
                                    // DLL DLM 时 DLAB 位需置 1，在
                                    //设置输入输出缓冲寄存器时 DLAB 位要
                                    //清零
CSA_DLL=0x30;                  //设置波特率
CSA_DLH=0x00;
CSA_FCR=0x81;      //设置控制寄存器，允许先进先出的输入输出方式
CSA_LCR=0x1B;                  //DLAB=0
CSA_IER=0X01;                  //设置接收中断
CSA_LCR=0X1B;                  //DLAB=0
CSB_LCR=0x9b;
CSB_DLL=0x30;
CSB_DLH=0x00;
CSB_FCR=0x81;
CSB_LCR=0x1B;
CSB_IER=0X00;
CSB_LCR=0X1B;
watch_dog();
CSC_LCR=0x9b;
CSC_DLL=0x30;
CSC_DLH=0x00;
CSC_FCR=0x81;
CSC_LCR=0x1B;
CSC_IER=0X00;
CSC_LCR=0X1B;
CSD_LCR=0x9b;
CSD_DLL=0x18;
CSD_DLH=0x00;
```

```
CSD_FCR=0x81;
CSD_LCR=0x1B;
CSD_IER=0X00;
CSD_LCR=0X1B;
watch_dog();
}
```

3.4 转换器的工作原理和流程图

我所设计的 RS422 到 CAN 总线转换器的功能主要就是等待、判断和处理由有 CAN 总线和 RS422 上的节点发送和接受数据，由于系统要求实时性较强，因此不采用查询方式，而采用中断方式。当有 CAN 中断发生时，首先是判断是接受数据还是发送数据，如果是发送数据，则将由 RS422 上传过来的数据线存入 51 片内对应的存储区，然后发送到 CAN 的数据缓冲区；如果是接受数据，则首先判断是发送那个 MAX1490A，然后将其数据存入相应的数据存储区，并将数据发送到 TL16C554 片内，选择好串口后，由所选择 MAX1490A 由 TTL 转换成 RS422 总线逻辑电平，传输到智能仪表等用 RS422 传输的数据。当有外部中断发生时，要判断是哪路 MAX1490A 发送过来的数据，然后选通该路，将串行数据发送到其对应的数据存储区内，并发送到 CAN 的数据缓冲区发送到 CAN 的其它需要此数据的节点。图 3.2 为工作流程图 ：

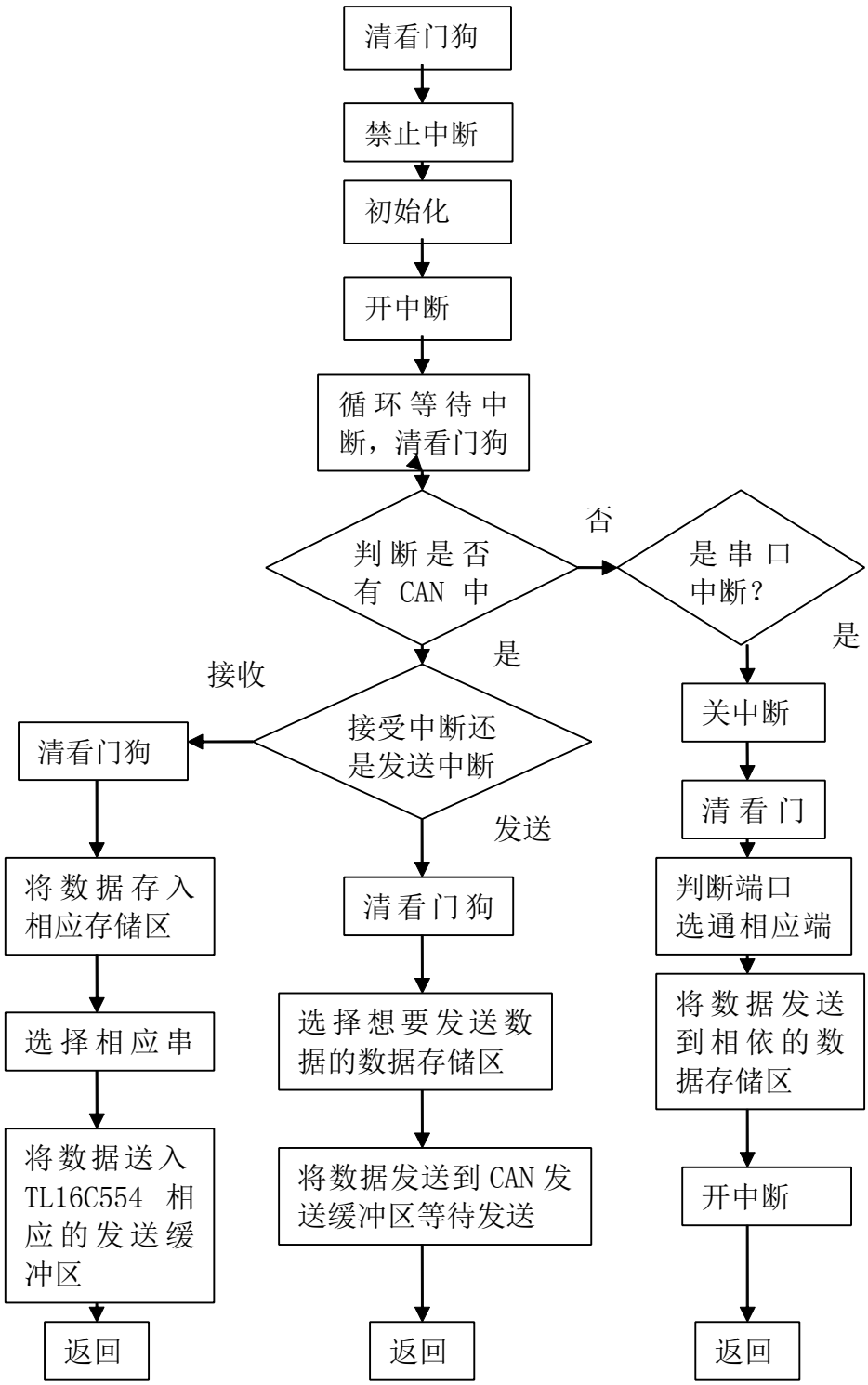
清看门狗
禁止中断
初始化
开中断
循环等待中断，清看门狗
判断是否有 CAN 中
否
是串口中断?
是
接收
是
接受中断还是发送中断
关中断
清看门狗
清看门
发送
将数据存入相应存储区
清看门狗
判断端口选通相应端
选择相应串
选择想要发送数据的数据存储区
将数据发送到相依的数据存储区
将数据送入 TL16C554 相应的发送缓冲区
将数据发送到 CAN 发送缓冲区等待发送
开中断
返回
返回
返回

图 3.2 工作流程图

在流程图的初始化中，软件编程是要对所有器件的所用的方面进行初始化。初始化部分包括：设置时钟频率，设置 C8051F040 芯片断口的工作方式，清空 51CAN 模块中所有 CAN 保温对象，初始化 CAN 发送对象，初始化 CAN 接收对象，启动 CAN，初始化通讯扩展芯片 TL16C554。其初始化具体流程图如图 3.3 所示

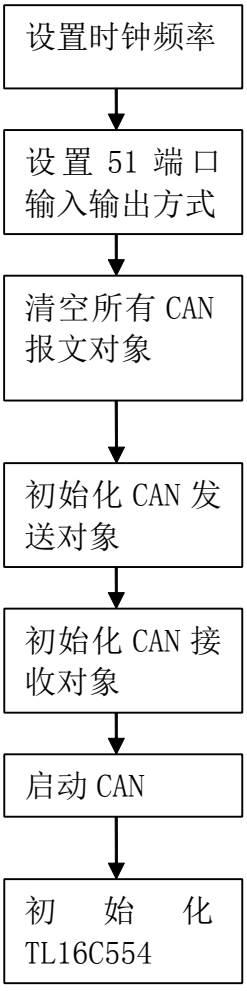

图 3.3 系统初始化流程图

3.5 系统程序设计

程序采用 C 语言设计,可以将软件个部分采用模块设计，此设计包扩一个主程序，十一个子程序，两个中断程序，分别用于处理从 CAN 口发送与接收数据和从 TL16C554 接收与发送数据。CAN 中断使用中断向量号 19，TL16C554 使用外部中断 1，各部分程序如下：

3.5.1 主程序

主程序是一个循环程序，主要作用是循环等待中断的发生，然后转向中断去处理数据。程序为：

```
////////////////////////////////////////////////////////////////////////////
void main (void) {
void watch_dog(void);                    // 清看门狗
void config_IO(void);                    //设置输入输出端口
void external_osc (void);                // 启动外部时钟
void init_TL16C554(void) ;               // 初始化 TL16C554
void   clear_msg_objects(void);          // 清 CAN 消息寄存器
void   init_msg_object_TX {num);         // 初始化发送对象
void   init_msg_object_RX (num);         // 初始化接受对象
 EIE2 = 0x20;                            // 打开 CAN 中断
                                         //使 CAN 开始工作并在其中设置波特率
void start_CAN(void);
EX1=1;                                   //打开外部中断 1
EA = 1;                                  //打开 51 中断
ET0=1;
       while (1)                         //循环等待中断
  {
 void watch_dog(void);                   // 清看门狗
 }
```

3.5.2 子程序部分

十一个子程序分别是

1. void clear_msg_objects (void); //清空报文对象
2. void init_msg_object_TX (char num); //初始化发送对象
3. void init_msg_object_RX (char num); //初始化接收对象
4. void start_CAN (void); //打开 CAN
5. void config_IO (void); //输入输出 I/O 设置

输入输出 I/O 设置，之所以设置 I/O 口，是因为现在的 51 器件其每个 I/O 口的功能并不是单一的，有两种甚至是三种作用，我们要把它设置在我们所需要的功能上，本软件所需要的 P0,P1,P2 的作用是输入输出数据，因此将其设置在在功能上。

6. void init_TL16C554(void) ; // 初始化 TL16C554
7. void watch_dog(void); // 清看门狗

看门狗设置 如果要保证系统在出现故障时能够自动复位就要用到看门狗电路，C8051F040 自带看门狗电路，只需对其进行简单的设置即可

8. void can_receive_data(num); //CAN 口接收数据
9. void can_transmit_data(num); //CAN 口发送数据
10. void tl16c554_receive_data(num); //tl16c554 口接收数据
11. void external_osc (void) //设置外部时钟频率

这里采用外部 22.4MHZ 时钟频率

3.5.3 两个中断子程序

1. CAN 中断子程序
2. TL16C554 中断子程序（具体程序见附件 2）

本章小节

在编写软件时，要使软件结构简单，功能完善，在与硬件的匹配中达到

很好的应用效果，这就要求软件的编写要有一定的技巧，在对转换器的软件编写过程中，我深深的体会到了这一点，并且对 C 语言编程有了一些自己的看法

当设计一个小的嵌入式系统时，一般都用汇编语言，在很多工程中这是一个很好的方法因为代码一般都不超过8K ，而且都比较简单。如果硬件工程师要同时设计软件和硬件，经常会采用汇编语言来做程序。 但是使用汇编的麻烦在于它的可读性和可维护性，特别当程序没有很好的标注的时候，代码的可重用性也比较低。如果使用C 的话可以很好的解决这些问题。

用C 编写的程序因为C 语言很好的结构性和模块化更容易阅读和维护。而且由于模块化用C 语言编写的程序有很好的可移植性、功能化的代码能够很方便的从一个工程移植到另一个工程从而减少了开发时间。

使用像C语言我们可以不必十分熟系处理器的运算过程。这意味着对新的处理器也能很快上手不必知道处理器的具体内部结构使得用C 编写的程序比汇编程序有更好的可移植性，现在很多处理器支持C 编译器，并且能将C语言转换成汇编语句。

我觉得C 的特点就是可以使你尽量少地对硬件进行操作，可以使你将大部分精力用于考虑算法、如何提高系统性能、稳定性，而不用花大量的时间去考虑系统内部的细节问题，减少了开发和调试的时间。

第四章 转换器的调试

RS422 到 CAN 总线的转换器的硬件电路搭好之后，软件程序写完之后，需要进行系统的调试工作，调试分为两部分。一是系统的模拟软件调试，二是硬件调试。软件调试是为了检测软件部分可能出现的错误、漏洞，然后将其完善，硬件调试测试系统与其他装备的兼容性，工作可靠性，工作温度范围，由于没有硬件测试条件，因此本次的调试工作主要是软件调试。

我们设计转换器应用系统后必须首先检测调试该系统硬件的可行性、正确性，并进一步编制应用软件。只有当应用系统硬件、软件调试通过后，才能将该系统正式应用。上述检测调试工作不可能在应用系统本身进行，这就需要有另外一种开发装置。为此，我们应用了 MCS—51 型多功能单片开发应用机。它有较大的存储容量，能方便地扩展接口，它不仅可以使用户方便地调试自己所设计的应用系统的软硬件，还可以在较大的实时控制系统中直接用作控制机。

MCS-51 开发机结合单片机的特点，设计了独特助监控程序和全套浮点运算程序，有灵活的断点设置、单步及单步调用程序、单步跟踪等功能，有较好的软件与硬件结合，使该机具有十分强的开发功能，为用户节省了大量的调试程序的时间。

要应用单片机开发机来测试所设计的硬件与软件，需要如下设备：一台测试用的 PC 机，一个 5V 电压源，单片机开发机，所设计的硬件电路板。先向测试用的 PC 机中装入单片及开发机的软件 51g6e，它在 PC 机上提供了一个平台，控制单片机开发机的模拟测试。将开发机的一个输入输出口接到 PC 机上面去，将电路板上的引出的与 PC 机向连接的两条通信线接到开发机上，最后给开发机接上 5V 电源。这样，一个测试系统就组装完成了。

通过应用单片机开发机对所设计的转换器的硬件部分及软件

部分的模拟测试，我们可以看到，硬件部分能够将数据进行转化，双向转化的实时性符合设计标准。所设计的软件部分也具有结构简单、功能完善的特点，用最优化的方式来完成设计任务，对整个的转换器的有效的工作提供了保证。

通过测试证实，我所设计的转换器具有设计要求中规定的功能与特点，可以完成要求中的导航网络通信任务，并且在使用和成本上具有其他同类设备不具有的优势，是一种十分实用的现场总线转换设备。

结论

目前，正在使用中的各种导航技术都已经趋于成熟。随着计算机的应用，使导航设备发生了急剧的变化，计算机使导航设备日益完善，并且朝着小型化、数字化和全自动化的方向发展。那么对连接各种导航设备的总线网络的性能要求也就越来越也严格，在现代工业网络中对各种总线发展也是越来越完善。

CAN 总线以其高性能、高可靠性及其独特的设计越来越受到人们的重视，并被公认为最有前途的总线之一。把 RS422 转换到 CAN 正是适应了这一趋势的发展，也许在将来，会出现 CAN 总线一统天下的局面，但在当今，由于多种总线形式并存的条件下，这种转换器有很大的实用价值。无论在导航、工业现场、过程控制中如果要联入 CAN 总线网络，都可以采用这种形式，这种转换器有很大的实用价值。

不光是 RS422 可以接入 CAN 总线网络，无论是 USB 总线，各种串行、并行总线都可以接入 CAN 总线，因为本文不仅提供了一种由 RS422 到 CAN 总线的转换器的设计，还提供了一种转换器的设计思想，也就是说对于其他的总线方式，也可以采用此种设计方法设计转换器，所以其有很广泛的应用价值。

随着科学技术的发展，新的导航技术必将出现，导航设备也必将不断的更新，但是将各种导航设备组合起来形成组合导航系统来完成导航任务这一方法不会落伍，它会随着导航事业的发展不断的发展下去，为船舶导航继续做出自己的贡献。

参考文献

[1] 阳宪惠主编. 现场总线技术及其应用[M]. 北京：清华大学出版社，1999.6

[2] 邬宽明著. CAN 总线原理和应用系统设计[M]. 北京：北京航空航天大学出版社，1996.11

[3] 甘永梅主编 .现场总线技术及其应用[M] 北京：机械工业出版社， 2000.1

[4] 穆兰编著. 单片微型计算机原理及接口技术[M]. 北京：机械工业出版社，1996.10

[5] 姚福安编著. 电子电路设计与实践[M]. 山东：山东科学技术出版社，2000.7

[6] RS422 收发器 MAX1490 说明书[S].MAXIM 公司

[7] 周立功.C51 使用技巧与实践. http://www.zlgmcu.com

[8] C8051F04X 系列单片机使用说明[S]. 新华龙电子有限公司

[9] F. Hartwich. C_CAN User’s Manual Revision 1.2 06.06.00

[10] 李肇庆 韩涛编著. 串行端口技术 国防工业出版社[M] 1999.4

[11] 周永余,陈永冰,李文魁.21 世纪初舰船组合导航系统的发展趋势[J].海军工程大学学报,2002.12

[12] 岑雪松,朱丹.SJ2000 在 CAN 总线系统节点的应用实例[J].单片机与嵌入式系统,2002.3

致　谢

在转换器的设计过程中我得到了周卫东老师的耐心指导。老师无论在大体设计上还是在具体的设计细节上都给了我极其宝贵的意见与建议，对我的设计提供了极大的帮助。

同时感谢学校机器人大赛陈东良博士生在我的设计过程中给我提供的热情帮助和有意启示。

感谢机器人大赛电路组组长冯官程在设计过程中对我所遇到细节问题上给我的技术支持与理论帮助。

感谢寝室和本班同学在生活和学习上对我的帮助。

最后，感谢自动化学院领导和老师对作者热情的关心和帮助，感谢我的母校哈尔滨工程大学多年来对我的教育和培养。

谨以此文献给所有关心和帮助我的人们。

附录

程序清单

3.4 程序用 c 语言设计，具体程序如下

```
#include <c8051f040.h>                          // SFR 定义
#include<stdio.h>// CAN 协议寄存器索引 for CAN0ADR,
#define CANCTRL          0x00          //控制寄存器
#define CANSTAT          0x01          //状态寄存器
#define ERRCNT           0x02          //错误计数寄存器
#define BITREG           0x03          //位时间寄存器
#define INTREG           0x04          //中断低字节寄存器
#define CANTSTR          0x05          //测试寄存器
#define BRPEXT           0x06          //波特率寄存器
//IF1 内部寄存器
////////////////////////////////////////////////////////////////////////////
#define IF1CMDRQST       0x08          //IF1 命令请求寄存器
#define IF1CMDMSK        0x09          //IF1 命令屏蔽寄存器
#define IF1MSK1          0x0A          //IF1 标志寄存器 1
#define IF1MSK2          0x0B          //IF1 标志寄存器 2
#define IF1ARB1          0x0C          //IF1 仲裁寄存器 1
#define IF1ARB2          0x0D          //IF1 仲裁寄存器 2
#define IF1MSGC          0x0E          //IF1 消息控制寄存器
#define IF1DATA1         0x0F          //IF1 数据寄存器 A1
#define IF1DATA2         0x10          //IF1 数据寄存器 A2
#define IF1DATB1         0x11          //IF1 数据寄存器 B1
#define IF1DATB2         0x12          //IF1 数据寄存器 B2
////////////////////////////////////////////////////////////////////////////
```

```
//IF2 内部寄存器
////////////////////////////////////////////////////////////////////////////
#define IF2CMDRQST          0x20            //IF2 命令请求寄存器
#define IF2CMDMSK           0x21            //IF2 命令屏蔽寄存器
#define IF2MSK1             0x22            //IF2 标志寄存器 1
#define IF2MSK2             0x23            //IF2 标志寄存器 2
#define IF2ARB1             0x24            //IF2 仲裁寄存器 1
#define IF2ARB2             0x25            //IF2 仲裁寄存器 2
#define IF2MSGC             0x26            //IF2 消息控制寄存器
#define IF2DATA1            0x27            //IF2 数据寄存器 A1
#define IF2DATA2            0x28            //IF2 数据寄存器 A2
#define IF2DATB1            0x29            //IF2 数据寄存器 B1
#define IF2DATB2            0x2A            //IF2 数据寄存器 B2
#define TRANSREQ1           0x40      //Transmission Rest1 Register
#define TRANSREQ2           0x41      //Transmission Rest2 Register
#define NEWDAT1             0x48            //新数据寄存器 1
#define NEWDAT2             0x49            //新数据寄存器 2
#define INTPEND1            0x50            //中断判断寄存器 1
#define INTPEND2            0x51            //中断判断寄存器 2
#define MSGVAL1             0x58            //消息有效寄存器 1
#define MSGVAL2             0x59            //消息有效寄存器 2
////////////////////////////////////////////////////////////////////////////
//定义 TL16C554
#define   CSA_RBR     XBYTE[0XFE00]          // A 接受缓冲寄存器
#define   CSB_RBR     XBYTE[0XFD00]
#define   CSC_RBR     XBYTE[0XFB00]
#define   CSD_RBR     XBYTE[0XF700]
```

```
#define   CSC_THR      XBYTE[0XFB00]          //C 发送缓冲寄存器
#define   CSA_LCR      XBYTE[0XFE03]
#define   CSA_DLL      XBYTE[0XFE00]
#define   CSA_DLH      XBYTE[0XFE01]
#define   CSA_FCR      XBYTE[0XFE02]
#define   CSA_LSR      XBYTE[0XFE05]
#define   CSA_IER      XBYTE[0XFE01]
#define   CSB_LCR      XBYTE[0XFD03]
#define   CSB_DLL      XBYTE[0XFD00]
#define   CSB_DLH      XBYTE[0XFD01]
#define   CSB_FCR      XBYTE[0XFD02]
#define   CSB_LSR      XBYTE[0XFD05]
#define   CSB_IER      XBYTE[0XFD01]

#define   CSC_LCR      XBYTE[0XFB03]
#define   CSC_DLL      XBYTE[0XFB00]
#define   CSC_DLH      XBYTE[0XFB01]
#define   CSC_FCR      XBYTE[0XFB02]
#define   CSC_LSR      XBYTE[0XFB05]
#define   CSC_IER      XBYTE[0XFB01]

#define   CSD_LCR      XBYTE[0XF703]
#define   CSD_DLL      XBYTE[0XF700]
#define   CSD_DLH      XBYTE[0XF701]
#define   CSD_FCR      XBYTE[0XF702]
#define   CSD_LSR      XBYTE[0XF705]
#define   CSD_IER      XBYTE[0XF701]
```

```
////////////////////////////////////////////////////////////////////////////////
sbit   CSA=P1^0;                          //定义管角
sbit   CSB=P2^1 ;
sbit   CSC=P2^2 ;
sbit   CSD=P2^3 ;
sbit   INTA=P2^4 ;
sbit   INTB =P2^5;
sbit   INTC=P2^6;
sbit   INTD=P2^7;
sbit   IOR =P4^6 ;                        //定义管角读信号
sbit   IOW =P4^7;                         //定义管角写信号
////////////////////////////////////////////////////////////////////////////////
//全局变量
////////////////////////////////////////////////////////////////////////////////
char num;                                 //数据区选择变量
char status;                              //CAN 状态位
int temp;
int i;
char a1,a2,a3;                            //在 tl16c554 接受数据
int k1=0,k2=0,k3=0;                       //和在 CAN 中发送数据中用到
int i1=0,i2=0,i3=0;
int da1=0,da2=0,da3=0;
int buf1 [100];                           //从串口过来的数据存储区
int buf2 [100];
int buf3 [100];
int buf4[100];
int can_RX1[4];                           //从 CAN 总线过来的数据存储区
```

```
int can_RX2[4];
int can_RX3[4];
int can_RX4[4];
sfr16 CAN0DAT = 0xD8;
//////////////////////////////////////////////////////////////////////////////
// 子程序
//////////////////////////////////////////////////////////////////////////////
void clear_msg_objects (void);                  //清空报文对象
void init_msg_object_TX (char num);             //初始化发送对象
void init_msg_object_RX (char num);             //初始化接收对象
void start_CAN (void);                          //打开 CAN
void config_IO (void);                          //输入输出 I/O 设置
void external_osc (void);                       // 启动外部时钟
void init_TL16C554(void) ;                      // 初始化 TL16C554
void watch_dog(void);                           // 清看门狗
void external_osc (void);                       // 启动外部时钟
void can_receive_data(num);                     //CAN 口接收数据
void can_transmit_data(num);                    //CAN 口发送数据
void tl16c554_receive_data(num);                //tl16c554 口接收数据
//////////////////////////////////////////////////////////////////////////////
// 主函数部分
//////////////////////////////////////////////////////////////////////////////
void main (void) {
void watch_dog(void);                           // 清看门狗
void config_IO(void);                           //设置输入输出端口
void external_osc (void);                       // 启动外部时钟
void init_TL16C554(void) ;                      // 初始化 TL16C554
```

```
void    clear_msg_objects(void);                    // 清 CAN 消息寄存器
void    init_msg_object_TX {num);                   // 初始化发送对象
void    init_msg_object_RX (num);                   // 初始化接受对象
 EIE2 = 0x20; // 打开 CAN 中断，使 CAN 开始工作并在其中设置波特率
void start_CAN(void);
EX1=1；                                             //打开外部中断 1
EA = 1;                                             //打开 51 中断
ET0=1；
 while (1)                                          //循环等待中断
 {
 void watch_dog(void);                              // 清看门狗
 }
////////////////////////////////////////////////////////////////////////////
// 设定 C8051F040
////////////////////////////////////////////////////////////////////////////
 void watch_dog(void) ；                            // 清看门狗
{
WDTCN=0XA5;
}
 void external_osc (void)                           // 设置外部时钟频率
{ int n;                                            // 局部变量用于循环
SFRPAGE = CONFIG_PAGE;          // 转向配置页配置时钟频率
OSCXCN    = 0x67;               // 打开外部时钟频率为;22.1MHz
for (n=0;n<255;n++);                                // 延迟 1ms
 {
while ((OSCXCN & 0x80) == 0);                       // 等待振荡器稳定
void watch_dog(void)；
```

```
CLKSEL |= 0x01;                                    // 转向外部时钟
}
void config_IO (void) //设置输入输出端口
{
SFRPAGE   = CONFIG_PAGE;        //管角在特殊状态寄存器设置页
XBR3      = 0x80;        // 设置 CAN TX 管角(CTX)  为 push-pull
XBR2      = 0x40;        // 允许 P0，1，2，3 端口的输入输出
}

////////////////////////////////////////////////////////////////////////////
//CAN 功能寄存器设置
////////////////////////////////////////////////////////////////////////////
//清消息寄存器
void clear_msg_objects (void)
{
 void watch_dog(void)；
 SFRPAGE   = CAN0_PAGE;
CAN0ADR   = IF1CMDMSK;                    // 指向命令屏蔽寄存器 1
CAN0DATL = 0xFF;            // 设置可以向所有的消息目标中写信号
 for (i=1;i<33;i++)
 {
CAN0ADR = IF1CMDRQST;                    // 复位所有消息
CAN0DATL = i;
 }
}
//初始化接受寄存器
void init_msg_object_RX (char num)
```

```
{
void watch_dog(void)；
SFRPAGE   = CAN0_PAGE;
CAN0ADR   = IF1CMDMSK;   // 指向命令屏蔽寄存器
CAN0DAT   = 0x00B8; // 设置为写，除了 ID MASD 和数据位其余都可更改
CAN0ADR   = IF1ARB1;      // 指向仲裁寄存器 1
CAN0DAT   = 0x0000;       // 设置仲裁寄存器 ID = "0"
CAN0DAT   = 0x8004;       // 仲裁寄存器 2  高字节:设置为标准标识符，不
                          //是扩展标识符
CAN0DAT   = 0x0480;       // 禁止远程帧
CAN0ADR   = IF1CMDRQST; // 指向命令请求寄存器
CAN0DATL = num;           // 设置选择位，用这一位选择数据存储区
}
//初始化发送消息寄存器
void init_msg_object_TX (char num)
{
void watch_dog(void)；
SFRPAGE = CAN0_PAGE;
CAN0ADR = IF1CMDMSK;   // 指向命令屏蔽寄存器
CAN0DAT = 0x00B2;   // 设置为写，除了 ID MASD 和数据位其余都可更改
CAN0ADR = IF1ARB1;      // 指向仲裁寄存器 1
CAN0DAT = 0x0000;       // 设置仲裁寄存器 ID  为最高优先级
CAN0DAT = 0xA000;       //设置为标准标识符
CAN0DAT = 0x0081;       // 禁止远程帧
CAN0ADR = IF1CMDRQST; // 指向命令请求寄存器
CAN0DAT = num;       // 设置选择位，用这一位选择数据存储区
}
```

```
//CAN 开始工作
void start_CAN (void)
{
void watch_dog(void)；
SFRPAGE   = CAN0_PAGE;     //指向 CAN0 页面
CAN0CN    |= 0x41;                  // 将 CCE 和 INIT 置 1 开始初使化
CAN0ADR   = BITREG     ;        // 指向位定时寄存器
CAN0DAT   = 0x2640;               // 设置波特率
CAN0ADR   = IF1CMDMSK;       // 指向屏蔽寄存器器 1
CAN0DAT   = 0x0087;         // 设置发送: 写入 CAN 内存, 写数据字节,
                                  // 设置 TXrqst/NewDat, clr IntPnd 位
CAN0ADR   = IF2CMDMSK;     // 指向屏蔽寄存器器 2
CAN0DATL = 0x1F;           // 设置接受 : 读 CAN 内存, 写入数据
                                  //和 clr NewDat and IntPnd 位
CAN0CN    |= 0x06;                // 允许全局中断   IE and SIE 置位
CAN0CN    &= ~0x41;       //清 CCE and INIT 位, 启动 CAN 状态机制
PCAN0=1；                         //设置中断优先级
}
 // 初始化 TL16C554
void init_TL16C554(void)
{
void watch_dog(void);              //清看门狗
CSA_LCR=0x9b;                 //设置 LCR 线形控制寄存器，设定字长，停
                                  //止位，奇偶校验和 DLAB 位设置 LCR
                                  // DLL DLM 时 DLAB 位需置 1，在
                                 //设置输入输出缓冲寄存器时 DLAB 位要
                                 //清零
```

```
CSA_DLL=0x30;                    //设置波特率
CSA_DLH=0x00;
CSA_FCR=0x81;      //设置控制寄存器，允许先进先出的输入输出方式
CSA_LCR=0x1B;                    //DLAB=0
CSA_IER=0X01;                    //设置接收中断
CSA_LCR=0X1B;                    //DLAB=0
CSB_LCR=0x9b;
CSB_DLL=0x30;
CSB_DLH=0x00;
CSB_FCR=0x81;
CSB_LCR=0x1B;
CSB_IER=0X00;
CSB_LCR=0X1B;
watch_dog();
CSC_LCR=0x9b;
CSC_DLL=0x30;
CSC_DLH=0x00;
CSC_FCR=0x81;
CSC_LCR=0x1B;
CSC_IER=0X00;
CSC_LCR=0X1B;
CSD_LCR=0x9b;
CSD_DLL=0x18;
CSD_DLH=0x00;
CSD_FCR=0x81;
CSD_LCR=0x1B;
CSD_IER=0X00;
```

```
CSD_LCR=0X1B;
watch_dog();
}
//CAN 总线中断程序
void CAN_ISR (void) interrupt 19
{
 void watch_dog(void);
int status= CAN0STA;
 if ((status&0x10) != 0)                    // RxOk 置位,中断由接收引起*/
 {
     void watch_dog(void);

CAN0STA = (CAN0STA&0xEF)|0x07;   // 将 RxOk 复位，置 LEC 没有改
变   void can_receive_data(num);            //CAN 口接受数据
void tl16c554_transmit_data(num);          //tl16c554 口发送数据
{
if ((status&0x08) != 0)                    // TxOk 置位，中断由发送引起
 {
 void watch_dog(void);
 CAN0STA = (CAN0STA&0xF7)|0x07; // 将 TxOkk 复位，置 LEC 没有改变
 void can_transmit_data(num);              //CAN 口发送数据
}
 if (((status&0x07) != 0)&&((status&0x07) != 7))   // 错误中断, LEC 变化
 {
void watch_dog(void);
 CAN0STA = CAN0STA|0x07;
 }
```

```
}
//TL16C554 中断程序
void timer1(void) interrupt 1                    //使用 1 号中断
{
temp=P2;                                         //读 p2 口
temp&=0xf0;                                      //屏蔽低四位
IOR=0;                                           //使读信号有效
switch(temp)                  //判断是哪一个串口读入数据，选择其串口
{
 case 0x10:CSA=0;                                //选择串口 A
 void tl16c554_receive_data(1);                  //tl16c554 口接收数据
CSA=1；                                          //发送数据后，关闭串口
A
break;
case 0x20:CSB=0;
void tl16c554_receive_data(2);                   //tl16c554 口接收数据
CSB=1；                                          //发送数据后，关闭串口
B
 break;
case 0x40:CSC=1;
void tl16c554_receive_data(3);                   //tl16c554 口接收数据
CSC=1；                                          //发送数据后，关闭串口 C
break;
case 0x80:CSD=1;
void tl16c554_receive_data(4);                   //tl16c554 口接收数据
CSD=1；                                          //发送数据后，关闭串口 D
 break;
```

```
default:   break;
}
void tl16c554_receive_data(num);                    //tl16c554 口接收数据
{
EA=0;                                               //关中断
switch(num)                                         //选择相应的数据区并将数据存入
 {
case 1:
void watch_dog(void);
 a1=CSA_LSR;
 if(a1&0x01)
{
 while(a1&0x01)
 {
a1=CSA_RBR;                    //将接受缓冲器 A 的数据存如数据区 1 中
buf1[i1++]=a1;
da1++;
a1=CSA_LSR;
 }
case 2:
void watch_dog(void);
 a2=CSB_LSR;
 if(a2&0x01)
{
while(a2&0x01)
{
a2=CSB_RBR;                    //将接受缓冲器 A 的数据存如数据区 1 中
```

```
buf2[i2++]=a2;
da2++;
a2=CSA_LSR;
}
case 3:
void watch_dog(void)
{WDTCN=0XA5;}
 a3=CSC_LSR;
if(a3&0x01)
{
 while(a3&0x01)
 {
a1=CSC_RBR;                    //将接受缓冲器 A 的数据存如数据区 1 中
buf3[i3++]=a1;
da3++;
a3=CSA_LSR;
}
case 4:
void watch_dog(void)
{WDTCN=0XA5;}
a4=CSD_LSR;
if(a4&0x01)
{
 while(aa&0x01)
{   a4=CSD_RBR;                    //将接受缓冲器 A 的数据存如数据区 1 中
buf4[i4++]=a1;
da4++;
```

```
a4=CSD_LSR;
 }
}
EA=1;                                          //开中断
}
void can_receive_data(num);                    //CAN 口接受数据
 {
   char i;
   SFRPAGE   = CAN0_PAGE;
   CAN0ADR   = IF2CMDMSK;
   CAN0DAT   = 0x003F;

   CAN0ADR   = IF2CMDRQST;
   CAN0DATL = num;
   switch(num)
   {
   case 1:
   void watch_dog(void);
   CSA=0;                                      //选通串口 1
   CAN0ADR   = IF2DATA1;                       // 指向数据场的第一个字节
    for(i=0;i<4;i++)                           //读取 4 个字节的数据
   {
     can_RX1[i]= CAN0DAT;                      //将数据发送到存储数据区
      CSA_THR= CAN0DAT;                       //将数据发送到 TL16C554 串口
                                               //1 的发送缓冲区等待发送
         }
        CSA=1;                                 //关闭串口 1
```

```
        break;
  case 2:
    void watch_dog(void);
    CSB=0;                                  //选通串口 2
        CAN0ADR    = IF2DATA1;              //  指向数据场的第一个字节
        for(i=0;i<4;i++)                    //读取 4 个字节的数据
       {
         can_RX2[i]= CAN0DAT;               //将数据发送到存储数据区
          CSB_THR= CAN0DAT;                 //将数据发送到 TL16C554 串
                                            //口 2 的发送缓冲区等待发送
         }
        CSB=1;                              //关闭串口 2
        break;
     case 3:
      void watch_dog(void);
     CSC=0;                                 //选通串口 3
        CAN0ADR    = IF2DATA1;              //  指向数据场的第一个字节
        for(i=0;i<4;i++)                   //读取 4 个字节的数据
       {
         can_RX3[i]= CAN0DAT;               //将数据发送到存储数据区
          CSA_THR= CAN0DAT;                 //将数据发送到 TL16C554 串口
                                            //3 的发送缓冲区等待发送
         }
        CSC=1;                              //关闭串口 3
        break;
     case 4:
      void watch_dog(void);
```

```
      CSD=0;                                  //选通串口 4
        CAN0ADR    = IF2DATA1;               // 指向数据场的第一个字节
        for(i=0;i<4;i++)                     //读取 4 个字节的数据
       {
         can_RX4[i]= CAN0DAT;                //将数据发送到存储数据区
          CSA_THR= CAN0DAT;                  //将数据发送到 TL16C554 串口
                                             //4 的发送缓冲区等待发送
         }
        CSA=1;                               //关闭串口 4
        break;
   }
void can_transmit_data(unm);                 //CAN 口发送数据
{
 switch(num)
 {
   case 1:   void watch_dog(void);
               for(i1=0;i1<da1;i1++)      //将数据发送到 CAN 发送缓冲寄存器
{
SFRPAGE = CAN0_PAGE;                        //发送数据
CAN0ADR = IF1CMDMSK;                        //指向命令屏蔽寄存器
CAN0DAT = 0x0083;
CAN0ADR = IF1ARB2;                          //指向仲裁寄存器 2
CAN0DAT|= 0x80;
CAN0ADR= IF1DATA1;                          //指向数据场的第一个字节
CAN0DAT =    buf1[i1++];          // 将数据区 1 中数据写入发送缓冲器
CAN0ADR = IF1CMDRQST;                       //指向命令屏蔽寄存器
CAN0DAT = num;                              //将报文编号写入
```

```
   }
 case 2:    void watch_dog(void);
for(i2=0;i2<da2;i2++)                //将数据发送到 CAN 发送缓冲寄存器
{
SFRPAGE = CAN0_PAGE;                 //发送数据
CAN0ADR = IF1CMDMSK;                 //指向命令屏蔽寄存器
CAN0DAT = 0x0083;
CAN0ADR = IF1ARB2;                   //指向仲裁寄存器 2
CAN0DAT|= 0x80;
CAN0ADR= IF1DATA1;                   //指向数据场的第一个字节
CAN0DAT =    buf2[i2++];             //  将数据区 2 中数据写入发送缓冲器
CAN0ADR = IF1CMDRQST;                //指向命令屏蔽寄存器
CAN0DAT = num;                       //将报文编号写入
}
case 3:    void watch_dog(void);
for(i3=0;i3<da3;i3++)                //将数据发送到 CAN 发送缓冲寄存器
{
 SFRPAGE = CAN0_PAGE;                //发送数据
 CAN0ADR = IF1CMDMSK;                //指向命令屏蔽寄存器
 CAN0DAT = 0x0083;
 CAN0ADR = IF1ARB2;                  //指向仲裁寄存器 2
 CAN0DAT|= 0x80;
 CAN0ADR= IF1DATA1;                  //指向数据场的第一个字节
 CAN0DAT =    buf3[i3++];            //  将数据区 3 中数据写入发送缓冲器
CAN0ADR = IF1CMDRQST;                //指向命令屏蔽寄存器
CAN0DAT = num;                       //将报文编号写入
}
```

```
case 4:    void watch_dog(void);
for(i4=0;i4<da1;i4++)                  //将数据发送到 CAN 发送缓冲寄存器
{
SFRPAGE = CAN0_PAGE;                   //发送数据
CAN0ADR = IF1CMDMSK;                   //指向命令屏蔽寄存器
CAN0DAT = 0x0083;
CAN0ADR = IF1ARB2;                     //指向仲裁寄存器 2
CAN0DAT|= 0x80;
CAN0ADR= IF1DATA1;                     //指向数据场的第一个字节
CAN0DAT =    buf4[i4++];               // 将数据区 4 中数据写入发送缓冲器
CAN0ADR = IF1CMDRQST;                  //指向命令屏蔽寄存器
CAN0DAT = num;                         //将报文编号写入
}
}
}
```