合肥工业大学

硕士学位论文

基于CANopen和μC/OS-Ⅱ的CAN网络通信技术研究

姓名：赖世勋

申请学位级别：硕士

专业：信号与信息处理

指导教师：丁志中

2011-04

基于 CANopen 和 μC/OS-Ⅱ的 CAN 网络通信技术研究

摘　要

CANopen 协议作为 CAN 总线的一种应用层协议，利用 CAN 总线实时性、可靠性的优势，实现了标准化和通用性，在各种分布式工业自动化控制系统中得到广泛地认可和大量的应用。因此研发基于 CANopen 协议的现场总线通信设备，能够促进国内业界对 CANopen 技术的关注，对于推动工控自动化领域自主知识产权产品的研究和发展，有着积极的意义。

本文首先介绍 CAN 总线及其高层协议的发展历程，并按照 CAN 协议的分层结构详细分析 CAN 总线协议和 CANopen 内核的关键内容，深入研究了 CANopen 协议中各种通信对象的功能及其实现方法。

CANopen 节点的硬件设计以处理器 MC9S12XF512 为核心，并利用芯片内置的 MSCAN 控制器连接双通道光耦合器 HCPL-2630 和高速 CAN 收发器 TJA1040 构成 CAN 总线通信接口电路。同时利用芯片的外设资源，设计外扩了 RS232 通信电路、LCD 液晶显示电路、BDM 下载接口等功能电路，共同构成了 CANopen 节点硬件架构。软件设计方面，在 μC/OS-Ⅱ系统中嵌入 CANopen 内核功能，从而在处理复杂的多任务时，可以利用嵌入式系统的任务调度及管理功能，使得整个系统在运行时稳定可靠。整个系统的代码遵循模块化的设计原则，采用分层结构实现 CANopen 协议的功能，包括硬件驱动层、μC/OS-Ⅱ操作系统层和 CANopen 通信协议以及设备行规应用层。

最后利用所设计的 CANopen 网络节点进行了通信实验，测试表明开发的 CANopen 节点符合 CANopen DS301 协议规范，实现了节点的初始化、从站状态机、PDO 和 SDO 传输、Heartbeat 网络监控等功能，充分验证了本设计方案的可行性和正确性。

关键词：CAN 总线， CANopen 协议，μC/OS-Ⅱ，MC9S12XF512

Research on the CAN bus Technology based on CANopen protocol and μC/OS-Ⅱ

Abstract

As an application layer protocol of CAN bus, CANopen protocol utilizes the features of real time and reliability of CAN bus and achieves standardization and interoperability. It has been widely accepted and applied in various distributed automatic control systems in industry. Therefore research and development of field-bus communication equipment based on CANopen protocol can arouse domestic industry's concern on the CANopen technology, and it will have significant meaning in terms of accelerating the development of self-intellectual property products.

This paper first outlines the development process of CAN bus and its high level protocols. According to the hierarchical structure, the important content of CAN protocol and CANopen kernel are analyzed in detail. The function and implementation of CANopen protocol's four category communication objects are studied in depth.The hardware design based on MC9S12XF512 chip comprises MSCAN interface circuit, RS232 serial communication circuit, BDM download interface circuit and LCD liquid crystal display circuit. MSCAN controller which is integrated in the MCU connects the dual-channel optocoupler HCPL-2630 and high-speed CAN transceiver TJA1040.They work together to constitute the CAN Bus communications interface circuits. The whole frame of software design based on MC9S12XF512 and μC/OS-II includes three layers according to the data flow. They are hardware driver layer, μC/OS-II system layer，the implementation and application of CANopen protocol. The CANopen communication function is embedded into the μC/OS-II. So the task scheduling and management capabilities of embedded system could be used in dealing with complex multi-task and make the whole system stable and reliable in operation.

Through constructing a master-slave CANopen network, the communication mechanism and related parameters are tested and the experiment shows the CANopen node has a good completeness of the agreed standards, CiA DS301. The four categories of communication object's function are verified through analyzing the CAN frame data. The experiment shows that the CANopen master-slave nodes can connect with each other and exchange CAN message successfully. The communication process is proved to be effective and reliable.

Keywords: CAN bus;CANopen protocol;μC/OS-Ⅱ;MC9S12XF512

插图清单

表格清单

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标志和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 合肥工业大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签字：赖世勋　　签字日期：2011 年 4 月 28 日

学位论文版权使用授权书

本学位论文作者完全了解 合肥工业大学 有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅或借阅。本人授权 合肥工业大学 可以将学位论文的全部或部分论文内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文者签名：　赖世勋　　导师签名：　丁志中

签字日期：　2011 年 4 月 28 日　　签字日期：　2011 年 4 月 28 日

学位论文作者毕业后去向：

工作单位：　　电话：

通讯地址：　　邮编：

第一章 绪论

1.1 引言

现场总线技术在现场智能化设备和自动化系统之间实现相互操作和数据共享，是一种可以实现双向、数字化、多站点的总线式标准化数字通信链路。现场总线网络融合了先进的传感器、计算机控制系统、数字通信网等技术，在世界范围内得到了广泛地认可和大量地应用，并促进了整个工业自动化控制系统和设备的革新。国内外众多具有雄厚实力的大公司和组织都不同程度地投入资金和人员进行现场总线技术与产品的开发。

现场总线技术从 20 世纪 80 年代开始形成和发展，经过这二十年的时间，市场上已出现几种成熟的总线协议和相关的产品，它们在各自的应用领域中显示出特有的优势和强大的生命力。在 1986 年，由德国 Bosch 公司研发并最终成为国际标准的 CAN 总线技术是目前市场上使用的最广泛的现场总线技术之一[1]。CAN 总线最初应用在汽车工业，但由于其优越的性能和独特的设计，CAN 总线的应用范围也扩展到过程控制、传感器网络、纺织机械、医疗器械及楼宇自动化等多种分布式控制场合。CANopen 是自动化 CAN 用户和制造商协会(CiA,CAN In Automation)定义的标准之一[2]，协议集定义了基于 CAN 的分布式工业自动化系统的应用标准以及 CAN 应用层的通信标准。CANopen 作为一种基于 CAN 总线的应用层协议，在欧洲和美国获得巨大的成功和大量的应用，这极大地促进了国内业界对 CAN 总线及 CANopen 技术的关注和研究。

1.2 课题研究的背景

1.2.1 CAN 总线及其高层协议

控制器局域网总线 CAN(Controller area network)节点不分主从，通信方式灵活，并且采用非破坏性总线仲裁技术，节省了总线仲裁时间，传输最高速率达到 1Mbps，因此广泛运用于汽车电控系统、工业现场自动化控制等诸多领域，成为汽车计算机控制系统和嵌入式工业控制局域网的标准总线。从 1992 年奔驰公司最初在高级轿车上使用 CAN 总线技术开始，越来越多的公司采用 CAN 总线标准，半导体公司包括 Intel、Motorola、Philips、MicroChip、Siemens 等都支持 CAN 总线标准，并开发出高性能的 CAN 控制器产品。随着 CAN 总线的推广运用，国际标准化组织 ISO 于 1993 年 11 月出版 CAN 的国际标准 ISO11898[3][4]，在 1995 年对国际标准进行扩展，以附录形式说明了 29 位的 CAN 标识符，这极大推动了控制器局域网的标准化和规范化。

CAN 总线标准详细定义了物理层和数据链路层的技术规范，而没有详细定义更高层次的协议。一些企业和组织为满足其关于 CAN 总线的更复杂的应用需求，在 CAN 标准的基础上开发了多种应用层协议，包括有 DeviceNet、

CANKingdom、CANopen、J1939 等[5]。

DeviceNet 是在 90 年代中期由美国 Rockwell 公司开发的一种基于 CAN 技术的开放型、符合全球工业标准的低成本高性能通信网络。DeviceNet 专门为工厂自动化控制而定制，在美国和亚洲有着广泛地应用[2][4]。ODVA(Open DeviceNet Vendor Association)即 DeviceNet 产品开发者组织于 1995 年成立，并致力于 DeviceNet 在全球的推广和市场化。DeviceNet 协议使用 CAN2.0A 标准帧格式，在网络传输时中以生产者/消费者模式取代源/目标模式，从而使得多个节点能同时获得相同的数据。在国内，DeviceNet 规范在 2002 年 12 月被国家标准化管理委员会批准为中国的国家标准，于 2003 年 4 月开始实施，虽然 DeviceNet 进入中国的时间并不长，但是在中国已经有许多的应用。

CANKingdom 协议是由瑞士 Kvaser 公司开发并应用于各种分布式控制系统。它支持生产者/消费者模式，可以实时地控制总线行为，其 ID 采用动态分配方式进行配置，在网络刚开始运行时，每个节点都没有自己的 ID，当节点 King 到来时，它通过算法对 ID 进行分配，每个节点将得到一组 ID，当有新节点加入网络中时，必须得到 King 节点的允许[6]。在网络配置完成后，将移除 King 节点，或者使其变成普通节点加入到总线网络中运行，因此 CANkingdom 这一节点 ID 分配方式可以最大程度的使用 ID，并最多支持 255 个节点。同时 CANKingdom 网络中可以接入 DeviceNet 模块或 SDS 模块，实现 CAN 应用层协议的混合使用。

J1939 协议是由美国汽车工程师协会(SAE)所推荐的一种 CAN 总线高层协议，主要用于卡车、工程机械、农用车辆等中重型车辆内部的电子元器件之间的通信[7]。J1939 协议在 CAN 总线协议的基础上增加了应用层和网络层定义、故障诊断及网络管理的功能。在美国工程师协会的推广下，J1939 协议在多种类型的重型卡车、客车中获得应用，并且可以与多种商业实时操作系统配套使用。

1.2.2 CANopen 协议

从 1993 年开始，Bosch 公司在一个工程中研发出一个协议的原型，在此基础上发展为 CANopen 规范。到 1995 年，CANopen 规范移交到 CiA(CAN in Automation)组织，由它来维护和发展这一应用层协议[40]。作为一个开放的标准化协议，CANopen 协议的规范、结构及模块可以下载得到，这大大促进了其发展。其灵活性和开放型促使其在五年时间里成为欧洲最重要的嵌入式网络标准，并衍生出基于不同设备的大量行规。至今它已经应用于医疗设备、海事电子、公共运输、建筑自动化等多种领域。

现在欧美发达国家的公司及组织研发出先进的 CANopen 软件和硬件产品活跃在市场中，比如 Warwick Control 的 CANopen 配置工具、德国 VECTOR 公

司设计的汽车总线开发工具 CANoe、BECKHOFF 的 CANopen 工业控制模块以及飞利浦公司研发的 CANopen 开发套件等;另外一些大公司则开发出性能和兼容性良好的 CANopen 协议栈源代码，并且提供技术支持，比如 SYS TEC 公司提供的源代码、IXXAT 公司提供的 CANopen 解决方案等。由公司开发的盈利性 CANopen 源代码售价昂贵，而在免费的 CANopen 开源代码方面，主要包括 CanFestival、MicroCANopen 和 CANopenNode[8]，这些开源的 CANopen 协议栈代码大多仅支持部分的功能，可靠性和实时性相对差一些，但是具有很好的参考和应用价值。

CANopen 有着以下这些良好的网络特性：

- 网络结构简单
- 设备参数存取简单
- 网络设备可同步操作
- 实时数据交换可采用循环及事件触发等多种传输方式
- 高传输速率
- 极强的抗干扰能力及总线冲突仲裁方式
- 统一的机构(CiA)管理和维护
- 开发性的结构，具有良好的兼容性

在国内，随着汽车和工业自动化控制系统地发展，CAN 总线技术也得到越来越广泛地运用，但在我国使用的 CAN 应用层协议主要是 DeviceNet，而 CANopen 协议还需要在我国进一步的推广和运用。由于 CANopen 技术具有上述的优点,国内若干院校和公司在 CANopen 技术研究上投入的人员和资金逐渐增多，对于 CAN 总线应用层的研究也更加地深入。中国单片机公共实验室从 1999 年开始进行对 CANopen 和 J1939 协议的研究和开发。在国内的大专院校中，只有少数的学校进行关于 CANopen 协议的研究和硬件开发。天津大学将开源的 MicroCANopen 协议移植到 AT91RM9200 处理器上，实现了 CANopen 从节点的开发，并使用 PCI-CAN 转接卡在 PC 机上实现了有限功能的 CANopen 监控主站[9]。北京工业大学将开源的协议栈 CanFestival 移植到 ARM9 平台[10], 并改进微芯公司的 CANopen 从节点协议栈并移植到 TMS230F2812 中[11]，实现了功能相对完整的 CANopen 节点开发。一些公司也开发出若干具有自主产权的 CANopen 产品，如广州的致远公司研发出的 CANopen 主站设备以及主站接口卡、数据采集模块、总线分析仪等，取得了一定的市场占有率。

1.2.3 嵌入式系统概述

嵌入式系统是指在嵌入式硬件平台上运行的，用于对整个系统及其操作的部件和装置等资源进行统一协调、管理和控制的系统软件。随着信息科技的飞速发展，单片机系统的规模和功能越来越强大，从而为嵌入式系统的稳定运行

奠定了硬件的基础。与通用的计算机系统相比，嵌入式系统的专业性强，通常它是面向于某一个特定的应用并仅支持某一特定类型的硬件。目前大多数的嵌入式系统是可裁剪的，用户在使用的时候可以根据自身的需要，去除冗余的系统功能，这样可以在满足应用需求的前提下，占用最少的系统资源运行在硬件中。嵌入式系统大多都固化在单片机系统的存储器中，这样可以迅速的对外部事件进行响应，从而提高了整个系统的实时性和可靠性，而且它的功耗很低，便于在各种移动设备中使用。

嵌入式系统的种类繁多，商用的实时操作系统包括有 WindowsCE ，移动电话的 Symbian，VxWorks 等，而开源免费的嵌入式操作系统有μC/OS-Ⅱ、μCLinux 等[12]。本文所使用的μC/OS-Ⅱ系统从 1992 年以来已经广泛运用于各个领域，并且于 2000 年 7 月得到美国联邦航空管理局对于商用飞机的安全性认证，这表明μC/OS-Ⅱ系统的每个功能、函数、每行代码都通过了严格测试和考验，在稳定性和安全性方面都满足了航空设备对软件的极高要求。

1.3 课题研究的意义和主要任务

在中国，虽然 CAN 和 CANopen 技术已经在楼宇自动化、机械设备等领域得到了一定的推广和运用，但是多数公司都选择直接购买欧洲厂商开发的整个控制系统，国内在 CANopen 设备的生产和供应上和发达国家还有很大的差距。针对这一行业现状，剖析 CANopen 协议的通信机制，进行自主的软硬件平台开发，对于促进中国自主知识产权的工业控制系统的发展和 CANopen 技术的推广运用，有着积极的意义。

本课题的任务是要通过对 CAN 总线及 CANopen 协议的充分理解和掌握，设计出符合 CANopen 协议 DS301 通信规范的 CANopen 通信平台；用软件实现整个 CANopen 协议复杂的内核功能，并将其成功地应用在 FreeScale 的 16 位单片机 MC9S12XF512，设计出符合现场总线应用需求的，高性能、低成本的 CANopen 通信设备，完成 CANopen 主从节点之间数据通信。本课题完成了以下的任务和目标：

(1) 详细地学习和掌握了单片机 MC9S12XF512 的各种功能，利用单片机内部集成的 MSCAN 控制器，实现 CAN 节点底层的数据收发功能。

(2) 分析 CANopen 协议的通信机制，重点分析和研究了 CANopen 的 DS301 通信规范和 DS401 数字输入/输出设备规范。

(3) 利用 C 语言编写了 CANopen 协议栈程序，遵循模块化的设计原则，依据协议栈的分层结构，完成 CANopen 协议各个功能模块及通信对象的软件设计。

(4) 分析μC/OS-Ⅱ内核的工作原理，掌握μC/OS-Ⅱ系统多任务的实现机制；根据μC/OS-Ⅱ系统结构，修改与处理器相关的系统代码，完成μC/OS-Ⅱ系统到

MC9S12XF512 芯片的移植工作；将 CANopen 从站状态机调度功能封装成任务在μC/OS-Ⅱ系统中运行。

(5) 构建 CANopen 通信实验平台，编写程序配置 CANopen 主从节点通信参数，完成 CANopen 节点主从通信，实现 CANopen 协议的 PDO、SDO、NMT 等通信对象功能的验证。

1.4 论文的组织结构

第一章针对课题研究的背景，简要地介绍了 CAN 总线及其高层协议、CANopen 总线的历史发展和基本特点，并指出 CANopen 协议研究和开发对于现场总线及工业自动化控制领域发展的重要性，给出了课题研究的意义和主要任务。

第二章从 CAN 总线的特点入手，根据 CAN 总线的分层结构，详细阐述了物理层、数据链路层的相关知识，剖析了 CAN 总线底层的通信原理。在 CAN 协议的应用层上，对 CANopen 协议的内核功能及其通信原理进行了详尽的分析和重点阐述。

第三章详细给出了 CANopen 节点的硬件架构，重点阐述了 CAN 总线接口电路以及其他外围功能电路的设计；详细分析了μC/OS-Ⅱ系统内核的工作原理，成功移植μC/OS-Ⅱ系统在 MC9S12XF512 上运行，通过封装 CANopen 协议从站状态机，完成 CANopen 内核的嵌入。

第四章根据 CANopen 协议的功能需求，编写软件实现协议的各个模块，使其能完成 CANopen 协议内核的各项功能。

第五章构建 CANopen 主从通信网络，配置相关通信参数，分析 CAN 报文数据，进行 CANopen 节点功能的通信测试。

第六章总结了本文完成的主要研究工作，并对后续工作进行了展望。

第二章 CAN 总线和 CANopen 内核分析

2.1 CAN 总线概述

CAN 总线以其独特的设计和优越的可靠性及稳定性，大量应用于工业过程控制设备的相互连接。CAN 总线协议已通过国际标准化组织的认证，并制定了 ISO11898 标准。成熟的技术、商品化的控制器芯片、较高的性价比使得 CAN 总线在业界得到充分的重视和广泛地认可，被认为是最有前途的现场总线之一。

2.1.1 CAN 总线的特点

CAN 总线使用多主竞争的方式工作，废除了传统的节点地址编码，而对通信数据块进行编码，节点不分主从，通信方式灵活[13]。

CAN 总线采用面向报文的优先级控制方式，用标识符定义静态的报文优先权，从而满足不同设备的实时性要求。

非破坏性的总线仲裁多主系统。总线空闲时，任何节点都可以开始传送报文，优先级较低的节点会主动退出发送，而具有高优先级的节点可以最终获得总线访问权，不受影响地继续传输数据。

CAN 通过报文滤波的方式实现多种方式接收数据，不需要专门的调度。

通信距离与通信速率有关。在总线长度低于 40m 时，CAN 总线的通信速率最高可以达到 1Mbps；当总线长度达到 10Km 时，其相应的通信速率则在 5Kbps 以下。

CAN 总线的通信介质可以是双绞线、同轴电缆或者光纤，用户可以自行选择，每一帧信息都有 CRC 校验及其他检错措施，保证了极低的数据出错率。

CAN 总线拥有先进的错误检测和错误处理机制，可以在整个系统范围内保持数据的一致性，延迟时间短，出错恢复快。

CAN 总线上的节点数目取决于总线的驱动电路，最高可达到 110 个，CAN2.0A 标准下报文标识符可以达到 2032 种，而在 CAN2.0B 标准下报文标识符基本不受限制。

2.1.2 CAN 总线协议的分层结构

CAN 协议的结构参考了 ISO/OSI 七层模型，在工业控制场合所要传输的数据量相对较少且对实时性要求较高，为满足这些需求，CAN 总线协议模型只分为物理层、数据链路层及应用层[14]，其协议模型如图 2.1 所示。物理层和数据链路层的功能由芯片生产商将其固化在 CAN 控制器上，各个公司和组织则在应用层上根据自身需要研发出多种应用层协议。图 2.1 给出了 CAN 协议的分层结构，并详细给出了物理层和数据链路层的具体功能[39]。

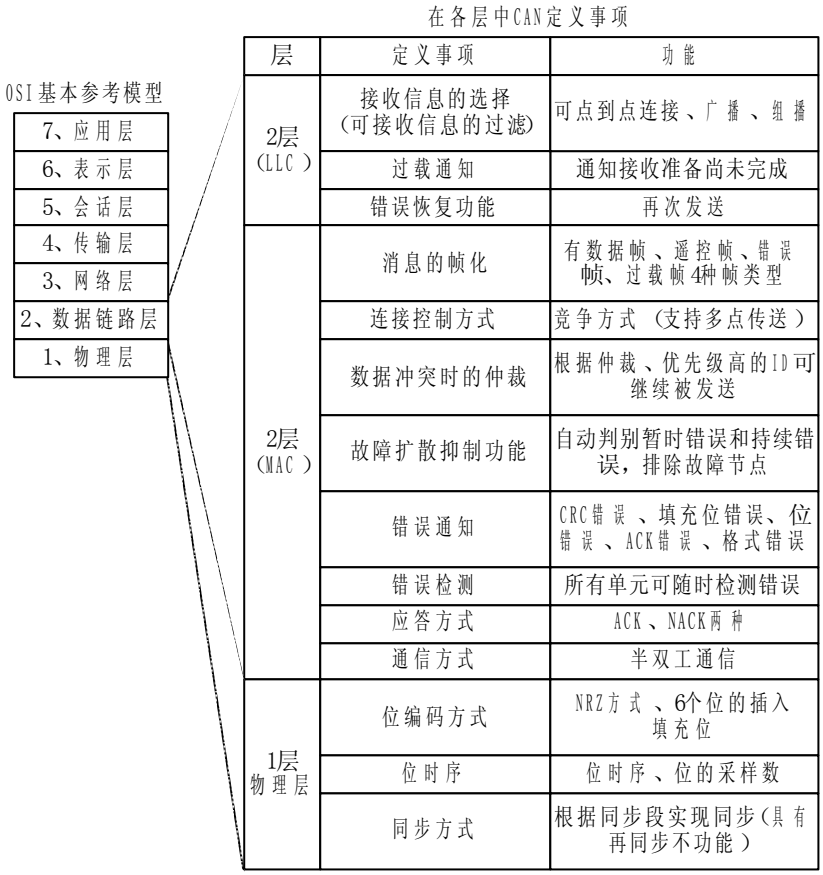

图 2.1 CAN 协议分层结构

2.2 CAN 总线协议物理层

2.2.1 CAN 总线物理层信号

CAN 总线的物理层包括整个通信系统的网络拓扑，描述了整个通信系统的物理结构。CAN 总线使用线性的总线拓扑形式，所有节点被连接到一条单独的两线电缆上。CAN 总线上的物理信号采取差分电压进行传送，从而使得驱动器能够容错以及避免总线上的噪声，包括有两种状态，CAN 总线协议规定，显性电平状态用逻辑“0”表示，隐性电平状态则用逻辑“1”表示。总线上的数值由两根导线上的 VCAN-H 和 VCAN-L 的差值表示。如图 2.2 所示，Vdiff 表示显性和隐性这两种逻辑数值，若总线上同时出现发送显性位或者隐性位的情况，则 CAN 发送驱动电路使得总线表现为显性，从而实现“0”和“1”之间的线与操作。

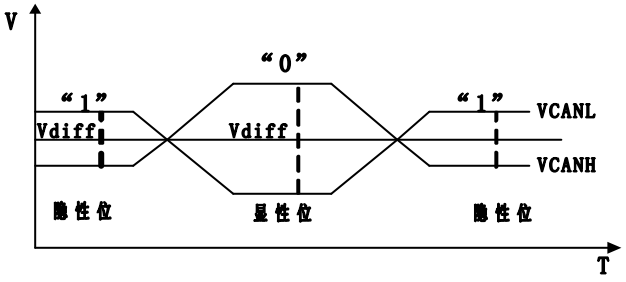

图 2.2 CAN 总线物理层信号

CAN 总线的通信速度最高可以达到 1M bit/s,由于总线上的时间延迟及电气

负载的限制，传输速率会降低。在实际应用时，可以依据总线网络上节点的数目和实际的应用需求，设定最适宜的通信速率。在同一个 CAN 总线网络中，所有的 CAN 节点必须工作在统一的通信速率下。如果某一个 CAN 节点的传输速率与其他节点不一致，则此节点会发送信号报告错误，并影响网络中其他 CAN 节点的通信。不同的 CAN 网络可以根据传输的实时性需求采用不同的通信速度，高低速 CAN 网络之间则可以通过 CAN 网关进行连接，并实现信息的交互。图 2.3 是 CAN 总线比特率与总线长度之间的大致关系。

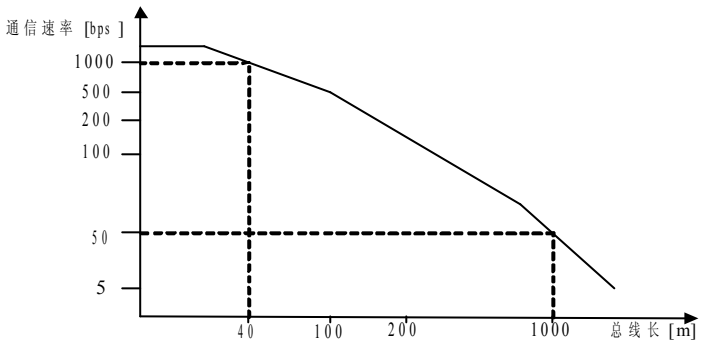

图 2.3 通信速度与最大总线长度

2.2.2 CAN 总线位定时的要求

位时间即为一位的持续时间，可划分为几个互不重叠的时间段，包括有同步段(SYNC_SEG)、传输时间段(PROP-SEG)、相位缓冲段 1(PHASE-SEG1)、相位缓冲段 2(PHASE-SEG2)[15][36]，如图 2.4 所示。在 CAN 控制器初始化时，通过配置确定总线的位时间从而确定 CAN 总线信息传输的波特率。

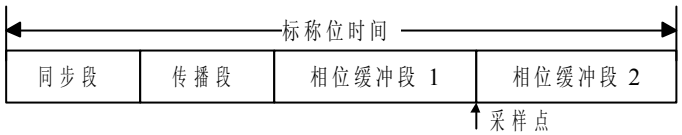

图 2.4 CAN 总线位定时

飞思卡尔芯片的集成的 MSCAN 控制器使用两个总线计时寄存器，MSCAN 总线计时寄存器 0(CANBTR0)和 MSCAN 总线计时寄存器 1(CANBTR1)来实现总线波特率的配置[16]。如图 2.5 所示 ，总线计时寄存器 0 定义了同步跳转宽度(SJW)和波特率预设(BRP)的值，SJW 用两个比特位来设置 1-4 个 Tq 时钟周期，预分频值则用 6 个比特位来设置其值的范围是 1-64。如图 2.6 所示，MSCAN 的总线计时器 1 控制 MSCAN 模块的总线定时器，其中的 SAMP 为采样位。它决定在每个位时间的内对总线数据的采样次数。若 SAMP 被置 1，则进行 3 次采样，若 CAN 总线的传输速率比较快，则最好将此位设置为 0，即每位进行 1 次采样。TSEG22~TSEG20 用来设置时间片段 2 的位时间长度和采样位置。TSEG13~TSEG10 则用来设置时间片段 1 的位时间长度和采样点位置。

一个位时间被分成三段，由一定数目的时间额度 Tq 组成。时间额度 Tq 是

指有振荡器时钟产生的一个固定的时间单位。

同步段(SYNC_SEG)：此段包含信号的同步信息

时间段 1(Time Segment 1)：此段包含传输段和相位段 1，时间段 1 可以被 TSEG1 设置成 4-16 个 Tq 时钟。

时间段 2(Time Segment 2)：此段包含相位段 2，时间长度由 TSEG2 设置为 2-8 个 Tq 时间。

通过 SJW 变量则可以调整同步跳跃宽度，可以是 1-4 个 Tq 时间。

	7	6	5	4	3	2	1	0
R W	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
复位：	0	0	0	0	0	0	0	0

图 2.5 MSCAN 总线计时寄存器 0(CANBTR0)

	7	6	5	4	3	2	1	0
R W	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
复位：	0	0	0	0	0	0	0	0

图 2.6 MSCAN 总线计时寄存器 1(CANBTR1)

位时间由振荡器频率、波特率预分频器和每位的时间额度(Tq)数量决定，位时间的值可以确定为：$\text{位时间}=\frac{\text{预分配值}}{\text{fCANCLK}}*(1+\text{时间片段1}+\text{时间片段2})$

2.3 CAN 总线数据链路层分析

2.3.1 CAN 报文帧格式

CAN2.0 标准协议分为 CAN2.0A 和 CAN2.0B，也称为 BasicCAN 和 PeliCAN，二者的区别在于 ID 的长度，Basic CAN 的 ID 长度是 11 位，PeliCAN 的 ID 长度是 29 位。CAN 的帧格式，主要由 ARBITRATION(仲裁段)、CTRL(控制段)、DATA(数据段)组成[13]。IDE 位为 1 表示此帧为 PeliCAN 的帧格式，ID 达到 29 位。RTR 位表示此帧是否为远程请求帧。CTRL 段的 DLC 则是控制数据段的数据长度，如图 2.7 所示。

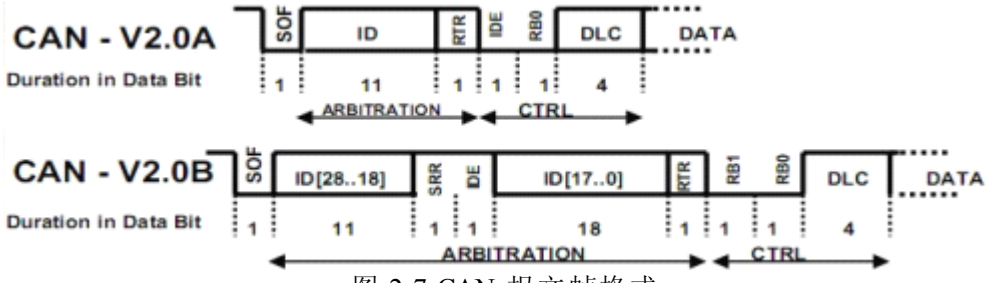

图 2.7 CAN 报文帧格式

CAN 通信主要通过以下五种帧来进行，其中数据帧使用最为频繁。

(1) 数据帧，用于发送单位向接收单位发送数据。

(2) 远程请求帧，用于接收单元向具有相同 ID 的发送单元请求数据的帧。

(3) 错误帧，用于当检测出错误时向其他单元通知错误。

(4) 过载帧，用于接收单位通知其尚未做好接收准备的帧。

(5) 帧间隔，用于数据帧和远程请求帧与前面的帧分隔开来。

任意一个节点检测到总线空闲时可以进入发送状态，发送数据时，首先发送帧起始位 SOF，它作为所有节点进行收发数据时的同步信号。在仲裁段，节点获得总线控制权后继续发送后面的数据。控制域里 DLC 的数值表明数据域所包含的数据字节数目。拥有 15 位 CRC 校验码的 CRC 域进行容错控制。

2.3.2 CAN 总线仲裁及错误处理机制

在 CAN 协议中，CAN 节点以如图 2.7 所示的固定帧格式来发送信息。所有的 CAN 节点在总线空闲时都可以开始发送新消息。当有多个 CAN 节点同时发送信息时，需要根据帧信息的标识符(Identifier，简称为 ID)来决定优先级。ID 并非代表着发送的目的节点地址，而是表示这一个 CAN 信息在访问总线时的优先级。对 CAN 信息的优先级进行仲裁时，需要通过对每个信息 ID 的比特位进行逐个的比较来确定。赢得仲裁(被确定具有最高的优先级)的 CAN 节点可继续发送消息，没有赢得仲裁的 CAN 节点则暂时停止发送工作。在 CAN 总线中，标识符数值越低则优先级越高，发送此报文的节点便可以获得总线的控制权。如图 2.8 所示，举例说明 CAN 总线的仲裁机制。

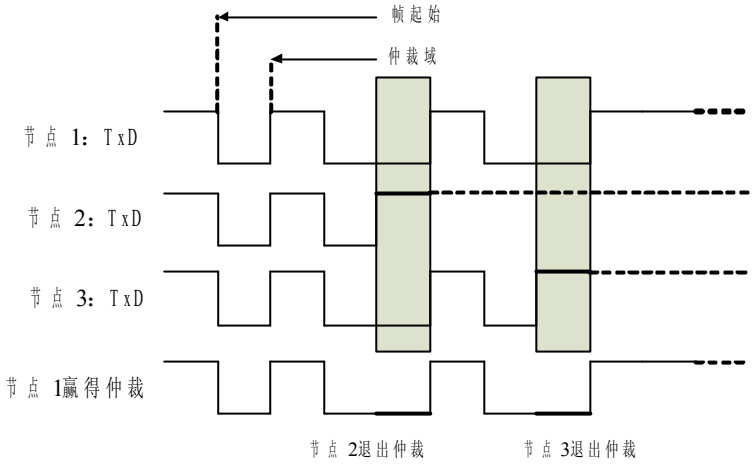

图 2.8 CAN 总线仲裁机制

当总线空闲时，CAN 节点 1、2、3 同时尝试发送报文，此时即在总线上发生冲突，需要根据节点的 ID 进行仲裁。前四个节点标识符位都相同，当发送第五个标识符位时，由于节点 2 是隐性位(逻辑数值为 1)，所以节点 2 退出总线仲裁，不再继续发送报文。同样地当发送到第八个标识符位时，节点 3 也由于其逻辑数值为 1 而自动退出仲裁。节点 1 的 ID 优先级最高，获得了总线的控制权并继续发送其报文的控制域、数据域和 CRC 域。可见 CAN 总线使用的是一种非破坏性的总线仲裁机制，一个节点获得总线通道后，其他节点不再发送报文，从而避免了总线带宽的浪费，并保证了高优先级报文的实时传输。

CAN 总线有五种类型的错误检测方法，包括位错误、填充错误、CRC 错

误、格式错误、和 ACK 错误[20]，前两个是位级的，后三个则是报文级的错误。如果一个报文出错，那么任何一个错误检测方法使得该节点不接收此报文并产生一个错误帧。

在位级检查中，每一个位受到发送器的监控。根据位填充规则，在逻辑电平相同的连续 5 位后，若下一位不是前面的反，则会产生一个位填充错误。报文级的检查包括 CRC 检查和 ACK 间隙。CRC 检查使用一个 15 位的 CRC 校验码，描述了标识符场和数据字节场的 CRC。确认场又叫做 ACK 场，包括 ACK 间隙和 ACK 界定符。发送节点的 ACK 场中，发送两个隐性位。而接收节点在接收到匹配 CRC 序列后，用显性位改写发送节点的隐性位，从而送出确认信息。若发送节点没有检测到显性位，则检测到一个应答错误，并重新发送报文。此外，在报文级还将检查那些总是隐性位的报文场，若检测到显性位则产生格式错误，包括有帧起始、帧结束、应答界定符、以及 CRC 界定符。

2.3.3 CAN 总线传输及滤波机制

CAN 总线的一大特点就是当数据帧在总线上传输时，没有节点地址的概念，每个报文通过唯一的标识符来指定其内容。若一个 CAN 节点要传送数据，只需要将数据和 ID 传送到控制器上，控制器根据图 2.7 所示的帧格式封装数据形成 CAN 报文，并通过总线仲裁机制来进行报文的发送。而总线上的其他节点通过设置自身的屏蔽过滤寄存器来对总线上的报文进行处理，并决定是否接收报文。节点的屏蔽过滤寄存器的每一位都是可编程的，可将其设置为允许或者禁止。通过这一传输方式，可以实现节点的点对点通讯、单点对多点通讯以及报文广播[21]。

2.4 CAN 应用层协议 CANopen 内核分析

2.4.1 CANopen 协议概述

CANopen 协议由一系列的子协议模块组成，应用于不同类型的设备，主要包含有通信规范 DS30x 和设备规范 DS40x[25]，如表 2.1 所示。

表 2.1 CANopen 协议集

CiA301	应用层及通信规范	CiA401	通用 I/O 模块
CiA302	附加应用层功能	CiA402	电机控制和驱动
CiA303	标准的和建议使用的线缆、管脚分配、标准化的 LED 状态显示和 SI 单元	CiA403	显示及操作
		CiA404	传感与调节器
		CiA405	可编程控制器
CiA304	安全相关的通信架构	CiA406	旋转及线性编码器
CiA305	层设置服务(LSS)和协议	CiA407	乘用信息运用规范
CiA306	CANopen 的电子数据表规范	CiA408	比例阀及液压传动规范

CANopen 的设备模型由图 2.9 中所示的三部分组成，包括通信接口、对象字典以及应用程序[17][22]：

通信接口这一功能单元提供通信对象通过网络结构来传输不同的数据项目，包括服务数据对象(SDO)、过程数据对象(PDO)、网络管理对象(NMT)以及一些特殊功能对象(SYNC、TIME 和 EMCY)。

对象字典收集着所有的数据项目，描述了所有的数据类型、通信对象和应用对象，影响着设备的应用对象、通信对象以及状态机的行为。它位于通信程序和应用程序之间，为应用程序提供接口[41]。用户编写 CANopen 的应用程序，其通信部分则通过通信对象对对象字典进行操作来实现 CANopen 通信。

应用程序由设备的功能组成，它与过程环境相互作用。

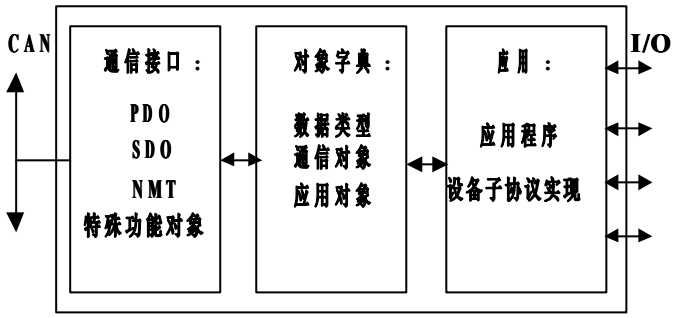

图 2.9 CANopen 设备模型

2.4.2 对象字典

CANopen 网络中所有的节点都独立拥有一个对象字典来实现标准化的设备描述，里面提供对设备所有重要数据、参数和功能的访问。这种访问机制是通过 index 和 sub-index 的逻辑地址体系来实现的，每个对象用一个 16 位的主索引和 8 位的子索引来进行寻址以确定其在对象字典中的入口[27][35]。一个节点的对象字典的有关范围在 0x1000 到 0x9FFF 之间，如表 2.2 中所述。对象字典主要分为 3 个部分，数据类型、通信规范和设备规范[17][22]。对象字典是 CANopen 协议中最重要的概念，用户在应用程序中通过通信对象对对象字典中的数据进行读写操作，从而实现 CANopen 通信。

(1) 索引值为 0001-009F 用来定义数据类型

(2) 索引值是 1000H 到 1FFFH 之间的对象定义了 CANopen 的通讯参数，每个 CANopen 设备都必须实现这些索引

(3) 高于 2000H 的索引值用来存放 CANopen 的设备规范参数

表 2.2 CANopen 对象字典

索引	对象
0000	Not Used
0001-009F	数据类型
00A0-0FFF	Reserved
1000-1FFF	通讯子协议区域(如设备类型、错误寄存器、支持的 SDO、PDO 数量)

2000-5FFF	制造商特定的规范区域
6000-9FFF	标准的设备规范区域
A000-FFFF	Reserved

2.4.3 预定义连接集

CANopen 通过被叫做“预定义连接集”(predefined connection set) 的形式实现标准化的 CAN-ID 分配，并定义了默认的强制性标识符(CAN-ID)分配表。如图 2.10 所示缺省的标识符分配基于 11 位的 CAN-ID，由高四位的功能码和低 7 位的节点 ID(Node-ID)部分组成，四位功能码决定对象的优先级，而七位的节点 ID 部分则用来区分不同的设备[26][28]。因此该 ID 分配方案可以支持至多一个主站和 127 个从站的 CANopen 网络通信。

图 2.10 预定义连接集 ID

如表 2.3 所示，预定义连接集的 ID 分配方案支持 4 个 TPDO、4 个 RPDO、1 个 SDO、1 个紧急对象和 1 个节点错误控制 ID，并且支持 NMT 网络管理对象服务和 SYNC 同步功能以及 Time Stamp 对象的广播。

表 2.3 预定义连接集

预定义连接集中的广播对象			
对象	功能码	COB-ID	对象字典中相关参数的索引
NMT	0000	0	
SYNC	0001	128(80h)	1005h、1006h、1007h
TIME STAMP	0010	256(100h)	1012h、1013h
预定义连接集中的点对点对象			
EMERGENCY	0001	129(81h)-255(FFh)	1014h、1015h
PDO1(tx)	0011	385(181h)-511(1FFh)	1800h
PDO1(rx)	0100	513(201h)-639(27Fh)	1400h
PDO2(tx)	0101	641(281h)-767(2FFh)	1801h
PDO2(rx)	0110	769(301h)-895(37Fh)	1401h
PDO3(tx)	0111	897(381h)-1023(3FFh)	1802h
PDO3(rx)	1000	1025(401h)-1151(47Fh)	1402h
PDO4(tx)	1001	1153(481h)-1279(4FFh)	1803h
PDO4(rx)	1010	1281(501h)-1407(57Fh)	1403h
SDO(tx)	1011	1409(581h)-1535(5FFh)	1200h
SDO(rx)	1100	1537(601h)-1663(67Fh)	1200h
NMT Error Control	1110	1793(701h)-1919(77Fh)	1016h、1017h

2.4.4 CANopen 通信对象

CANopen 的 DS301 通信规范定义了四类标准的通信对象[17][38]，包括过程数据对象(SDO)、网络管理对象(NMT)、服务数据对象(SDO)和其他一些特殊功能对象(SYNC、TIME、EMCY)，并且前三个通信对象是 CANopen 必须实现的基本通信对象，特殊功能对象则根据实际应用需要，是可选的。这六个通信对象采用客户机/服务器(C/S)、主从(M/S)、生产者/消费者(P/C)三种通信模式来实现[29]。通信对象与通信模式的关系如表 2.4 所示，以下对这三种通信模式进行具体说明。

表 2.4 通信对象与通信模式的关系

	PDO	SDO	NMT	SYNC	TIME	EMCY
M/S			■			
C/S		■				
P/C	■		■	■	■	■

在主从模式下，通信由主节点发起，从节点一直等待来自主节点的通信请求。CANopen 协议中与网络管理机制相关的服务使用该通信模式，即 NMT 节点和错误控制使用此通信模式,如图 2.11 所示。

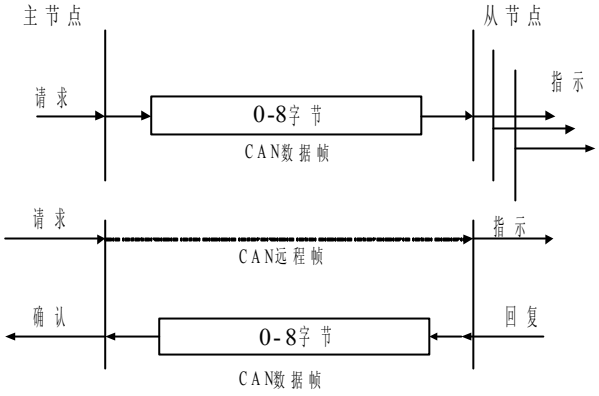

图 2.11 主从模式

在客户机/服务器信模式下实现两个设备之间的点对点通信，CANopen 协议的通信对象 SDO 采用此通信模式，设备之间的配置参数通过 SDO 进行传输，通信模型如图 2.12 所示。

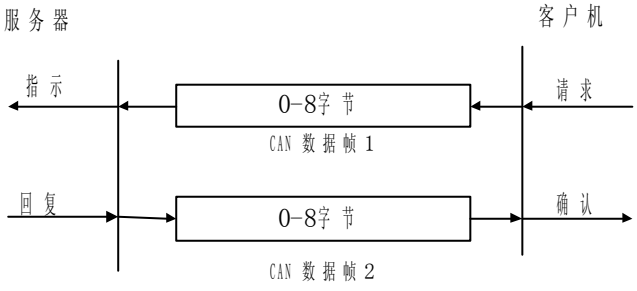

图 2.12 客户机/服务器模式

生产者/消费者模式完美的描述了 CAN 的广播通信功能。在此模式下，网络中的每一个节点能够监听到某一节点发送的信息，节点根据自身的接收滤波

设置决定是否接收信息。该模式主要应用于交换设备之间的过程数据，通信模型如图 2.13 所示

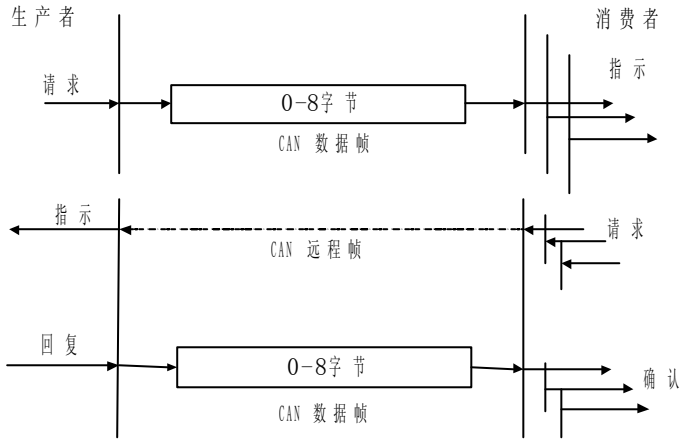

图 2.13 生产者/消费者模式

(1) PDO 通信

过程数据对象用来传输实时数据，数据传送采用生产者/消费者模式，传送字节数限制在 1-8 个字节[23][42]。每个 PDO 在对象字典中通过通讯对象和映射对象来描述，并分为发送 PDO(TPDO)和接收 PDO(RPDO)，在对象字典中的具体地址如下[30]。

TPDO 通信索引=0x1800+(TPDO_number-1)

TPDO 映射索引=0x1a00+(TPDO_number-1)

RPDO 通信索引=0x1400+(RPDO_number-1)

RPDO 映射索引=0x1600+(RPDO_number-1)

PDO 的通信参数中包含了 PDO 所使用的 COB-ID、传输类型以及禁止时间和定时器周期[17]。CANopen 通信规范将 PDO 的信息触发模式分为 3 种，包括事件驱动、定时驱动和远程请求，可以通过同步或异步的方式来传送信息。同步则是通过接收 SYNC 对象来实现，在周期性同步下，消息在每 1 到 240 个 SYNC 信息下触发。而远程帧或者特定的事件将触发 CAN 节点在非周期性同步的条件下传送信息。如表 2.5 所示，给出了不同传输类型的 PDO 传输模式，其中 Both 表示两个条件都需要，Only 表示只需要一个或者两个条件。SYNC 表示接收到同步对象，RTR 表示接收到远程帧，Event 则表示由事件触发，对象字典中使用一个 8 位无符号数来定义通信对象的传输类型。

表 2.5 PDO 传送类型定义

传输类型	触发 PDO 的条件			PDO 传输描述
	SYNC	RTR	Event	
0	Both		Both	同步，非循环
1-240	Only			每隔 1-240 个同步周期发送同步 PDO
241-251				保留
252	Both	Both		同步，仅当收到一个远程请求后发送 PDO
253		Only		异步，在收到远程请求后发送 PDO
254		Only	Only	异步，制造商特定事件
255		Only	Only	异步，当事件发生时发送 PDO

PDO 的映射机制如图 2.14 所示，它将要传送的数据定位在设备的对象字典中。通过 PDO 映射的内容，报文的发送和接收者可以知道 PDO 数据域的意义。每个节点的对象字典保存了包含这些映射关系的参数，主站可以通过 SDO 报文对 PDO 通信参数、映射参数和数据对象进行读写，从而改变这些映射参数。每个设备都有 TPDO 和 RPDO，且每个 PDO 的标识符都不同，因此 CANopen 设备可以依据不同标识符的 PDO 报文来确定其所传送和接收数据的意义。此外，PDO 允许位映射，因此传送八字节的数据至多映射 64 个参数。

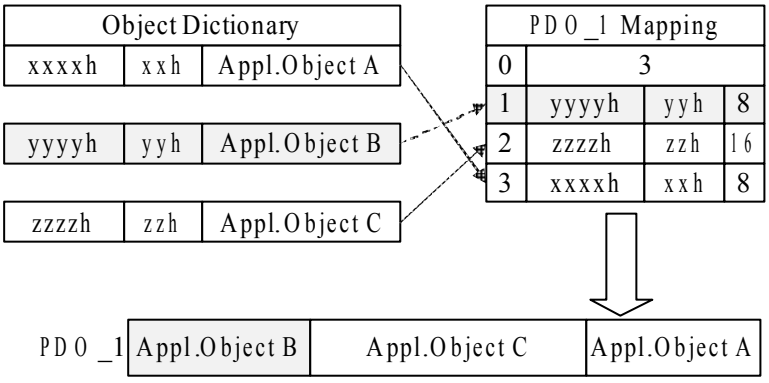

图 2.14 PDO 映射机制

举例说明，如表 2.6 中 TPDO2 映射如下，在对象字典索引 0x6002h 的子索引 2 定义的是 16 位的模拟量输入，在对象字典索引 0x6003h 的子索引 1 定义的是一个 8 位的数字量输入。

表 2.6 TPDO2 的映射参数

对象 0x1A01(由 TPDO2 映射)		
子索引	值	意义
0	2	2 个对象映射在 PDO 中
1	0x60020210	对象 0x6002，子索引 0x02，由 16 位组成
2	0x60030108	对象 0x6003，子索引 0x01，由 8 位组成

若发送此 PDO 报文，它将由 3 个字节组成，以如表 2.7 格式进行传送，CANopen 的多字节参数总是先发送低八位(LSB)。实际应用时，通过改变对象 0x1A01 的内容，PDO 的内容得到改变。

表 2.7 TPDO2 进行发送时的帧结构组成

COB-ID	Byte0	Byte1	Byte2
0x280+Node_ID	16 位数字量输入(低 8 位)	16 位模拟量输入(高 8 位)	8 位模拟量输入

(2) SDO 通信

服务数据对象 SDO 用来访问设备的对象字典，它以客户机/服务器模式来传输数据。SDO 访问对象字典将采用如图 2.15 所示的机制来通过主索引和子索引进行寻址。

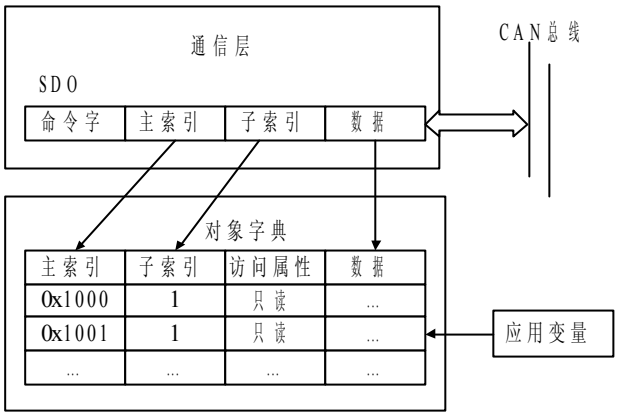

图 2.15 SDO 访问对象字典

对象字典被访问且提供请求服务的 CANopen 设备称作服务器(Server)，而访问者即为客户，也就是说 SDO 用来对一个节点的对象字典进行读写，请求读或者写操作的节点即为客户机，而被读写的节点即为服务器。SDO 包含两种传送机制，加速传送(Expedited transfer)和分段传送(Segment transfer)，前者最多传送 4 字节数据，后者可以传输大于 4 字节长度的数据，并以如下表 2.8 和表 2.9 所示的基本结构进行传送。

SDO 命令字包含以下信息：分段/加速传送(Segmented/expedited transfer)、下载/上传(Download/upload)、CAN 数据域的字节数目、请求/应答(Request/response)、一个触发位(toggle bit)使得后续的分段实现交替地清零和置位。根据不同的语法细节设置命令字，SDO 可以实现五种请求/应答协议，包括

启动域下载(Initiate Domain Download)、启动域上传(Initiate Domain Upload)、域分段上传(Upload Domain Segment)、域分段下载(Download Domain Segment)和域传送中止(Abort Domain Transfer)。

表 2.8 SDO 帧结构(客户机到服务器或者服务器到客户机)

字节 0	字节 1—2	字节 3	字节 4—7
SDO 命令字	16 位对象索引	对象子索引	**

表 2.9 分段传输下的 SDO 帧结构

字节 0	字节 1-7
命令字	可达 7 个字节的数据

当一个服务器节点收到来自客户机的下载请求时，表示需要对此服务器节点中对象字典的数据进行写操作，修改数据并存储成功后，服务器回复报文表示数据接收成功。根据 SDO 具体的语法细节，若要写数据 0xd0d1d2d3 到服务器的对象字典，即为下载操作，客户机节点将发送如表 2.10 的 SDO 帧信息。

表 2.10 客户机的 SDO 下载请求报文格式

600+SeverNodeId	23(SDO 命令字)	Index	Sub -index	D3	D2	D1	D0

若发送成功，服务器回应如表 2.11 所示的 SDO 信息。

表 2.11 服务器的 SDO 下载回应报文格式

580+SeverNodeId	60(SDO 命令字)	Index	Sub-index	00	00	00	00

若发送失败，服务器回应如表 2.12 的 SDO 信息，其中后四个字节是具体的 SDO 错误代码，用来指示是因为什么原因导致发送失败。

表 2.12 SDO 错误中止报文格式

580+SeverNodeId	80(SDO 命令字)	Index	Sub-index	SDO 中止错误代码

因此，若要写四个字节的数据 0x60120308 在节点 ID 为 5 的对象字典的主索引 0x1603，子索引 1 处，根据上述的 SDO 帧格式，其具体的数据是：605 23 03 16 01 08 03 12 60，而发送成功后，节点 5 将回复 585 60 03 16 01 00 00 00 00。

若服务器节点收到来自客户机的上传请求时，表示客户机要对此服务器对象字典中的数据进行读操作，服务器若成功读取对象字典中数据，便回复一个携带所读取数据的 SDO 信息到客户机。根据 SDO 具体的语法细节，若客户机需要发送如表 2.13 所示的报文。

表 2.13 SDO 上传服务请求报文格式

600+SeverNodeId	40(SDO 命令字)	Index	Sub -index	00	00	00	00

如果请求成功，服务器端进行回应，发送如表 2.14 所示的报文，将数据 0xd0d1d2d3 发送回客户端。

表 2.14 SDO 上传服务回应报文格式

580+SeverNodeId	43(SDO 命令字)	Index	Sub-index	D3	D2	D1	D0

同样地，若请求失败，也将中止 SDO 传输，并发送如表 2.12 所示的报文。

因此，若要从节点 ID 为 5 的对象字典的主索引 0x1603，子索引 1 处读取数据，根据 SDO 帧格式将发送以下数据：605 40 03 16 01 00 00 00 00，请求成功后，服务器端返回数据 0x60120208，具体报文数值时 585 43 03 16 01 08 02 12 60 。

(3) NMT 网络管理对象

NMT 网络管理对象采用主从模式进行通信，用于主节点对网络中的所有节点进行监控和管理，以下从三个方面来说明。整个 CANopen 的网络管理是以节点为导向，遵循一个主从式的网络结构，在网络中需要一个设备实现主节点的功能，其余节点作为 NMT 从节点。网络管理模块实现以下的功能组合，包括有 NMT 从节点加入分布式应用时初始化模块控制服务、对节点和网络通信状态进行监督的错误控制服务以及分别上传和下载网络模块配置数据的模块配置服务。一个 NMT 从设备代表网络中一个节点的网络管理功能，其模块 ID 在网络中是唯一的。

①节点启动

NMT 从节点通过发送 Boot-up 启动报文来通知主节点它已经从初始化状态进入了预操作状态。主节点根据这一信息开始对从节点进行节点保护或是发送心跳报文。其具体的帧格式如表 2.15 所示：

表 2.15 Boot-up 启动报文（从节点发送到主节点）

COB-ID	字节 0
0x700+Node ID	0

②节点状态控制

CANopen 的 NMT 从设备实现一个状态机，在上电之后对每个设备进行初始化并进入预操作状态，在这一状态下节点通过 SDO 进行参数的配置，并且不允许 PDO 通信[43]。NMT 主设备可以使得网络中所有节点或某一节点进入操作状态，在操作状态下，CANopen 设备可以激活所有的通信服务，可以通过 SDO 访问对象字典，通过 PDO 进行实时数据的通信[34]。通过控制一个设备进入停止状态来使 CANopen 设备停止 PDO 和 SDO 通信，整个状态机如图 2.16 所示。

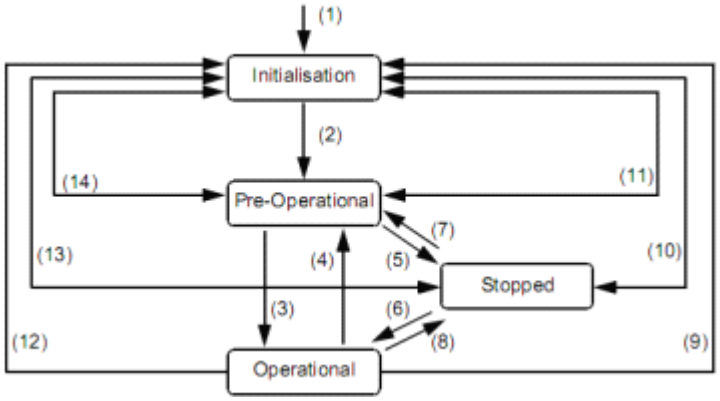

图 2.16 设备行为状态图(NMT 状态机)

NMT 主节点发送的报文格式如表 2.16 所示，第一个数据字节 CS 是命令说明符，第二个数据字节是节点地址，当节点地址部分为“0”则代表队所有节点进行寻址，即为广播。命令字的具体内容如表所示，用来完成 NMT 状态机状态的切换。表 2.17 给出了 NMT 命令字的具体功能以及在图 2.16 中相关的状态切换。

表 2.16 NMT 主节点报文(NMT 主节点→NMT 从节点)

COB-ID	字节 0	字节 1
0x000	CS	节点 ID

表 2.17 NMT 命令字及功能说明

状态转换	命令字 CS	NMT 服务
(3)，(6)	1	启动模块，使能输出，开始传送 PDO
(5)，(8)	2	停止通信，将输出设置为错误状态
(4)，(7)	128	停止 PDO 传送，但可以进行 SDO 传输
(9)，(10)，(11)	129	执行模块的复位
(12)，(13)，(14)	130	执行通信功能的复位

③错误控制

NMT 的错误控制包括节点保护和心跳协议。

节点保护服务可以让 NMT 主节点检查每个节点的当前状态，在保护过程中主节点使用被监控的从节点的保护报文标识符来发送如表 2.18 所示的远程帧，而从节点则使用如表 2.19 所示的保护报文来回复。保护报文中包含了从节点的状态编码以及一个每一条报文后都必须改变一次的跳变位(toggle bit)。如果报文中的状态及跳变位与 NMT 主节点所期望接收的位不匹配或无保护报文回应，主节点将认为从节点已出现故障。

表 2.18 节点保护远程请求帧格式

COB-ID	RTR 位
0x700+Node ID	1

表 2.19 节点保护应答帧格式

COB-ID	Byte0
0x700+Node_ID	比特位 7：toggle　　比特位 6-0：状态

主节点可以对保护报文严格地进行循环地请求，从节点也可以根据此检测到主节点时候产生故障，如图 2.17 所示。如果从节点在设定的节点生命时间(Node life time)内没有收到一个报文请求，那么从节点将认为主节点已发生故障。随后从节点可将输出设定为错误状态，并发送紧急时间报文后进入预操作状态。节点的生命时间使用位于对象字典的索引 0x100C 和 0x100D 的保护时间参数和生命时间因素来计算，二者关系如下所示：生命时间=保护时间×生命时间因素。

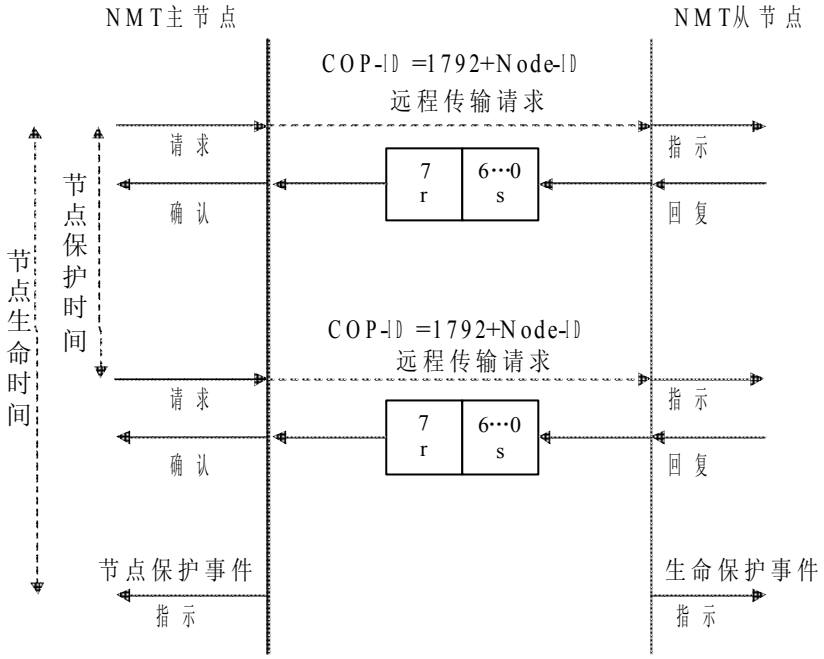

图 2.17 节点保护与生命保护

在使用心跳协议时，不需要使用远程帧。心跳的生产者周期性地发送心跳报文。一个或者多个心跳消费者将收到心跳报文。心跳消费者在心跳消费时间内对报文的接收进行监控，若在此时间内没有接收到心跳报文，则产生一个心跳事件，心跳消费者即认为生产者已发生故障，这一过程如图 2.18 所示。在节点对象字典的 0x1016 和 0x1017 定义了消费者心跳时间和生产者心跳时间。

第三章 CANopen 平台硬件架构及μC/OS-Ⅱ系统嵌入

3.1 硬件系统总体架构

整个开发平台采用飞思卡尔的 MC9S12XF512 单片机作为主芯片，它采用了高性能的 16 位处理器 HCS12，可提供丰富的指令系统，具有较强的数值计算和逻辑运算能力；其高达 512K 的大容量 FLASH 存储器具有在线编程能力，RAM 和 EEPROM 可存储各种控制参数[16]。CAN 通信模块通过单片机内部集成的 MSCAN 控制器和高速 CAN 收发器 TJA1040 来实现，为增加通信距离和抗干扰能力，在 MSCAN 控制器和收发器之间使用光耦合器 HCPL2630 来进行光电隔离。开发平台还有电源模块、BDM 编程下载接口、串口通信模块、LCD 液晶显示模块以及其他一些 I/O 口的应用。CANopen 硬件系统的整体架构如图 3.1 所示。

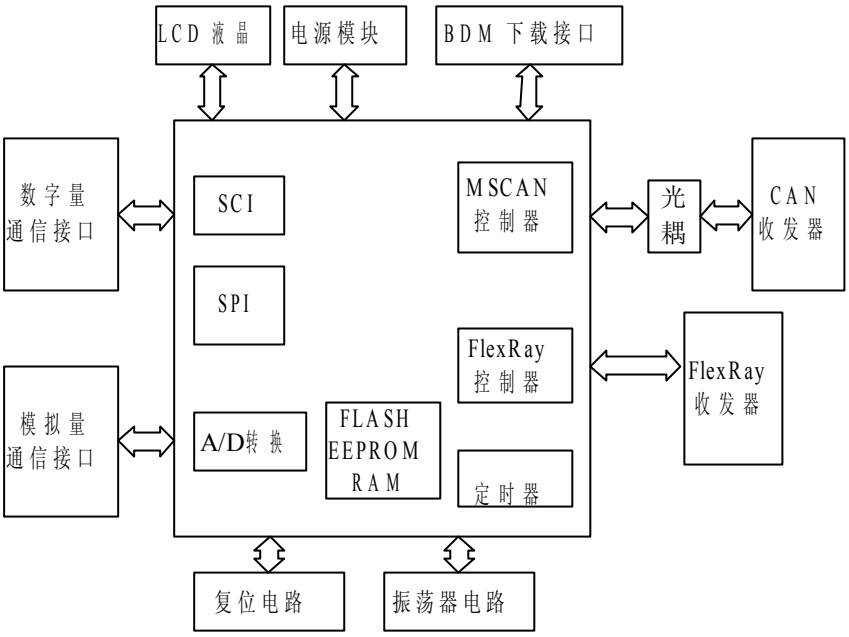

图 3.1 CANopen 硬件系统总体架构

3.2 MC9S12 芯片结构及主要功能电路设计

如图 3.2 所示为 MC9S12XF512 单片机的内部结构组成，左、右分别为内核和外设部分[16]。A 口和 B 口在扩展方式下作为分时复用的地址/数据总线，而 E 口的一部分作为控制总线在系统扩展时使用。每一个接口具有双重功能即通用 I/O 口和特殊接口功能。内部容量高达 512K 的 Flash 可以用来保存程序和原始数据，在正常工作时不用担心被改写。32K 的 RAM 存储器可以用作堆栈以及保存动态数据。4K 的 EEPROM 可以保存一些半永久的数据。16 位的 CPU12 具有 16 位乘法和 32 位除 16 位的整数乘除运算能力，内部设有指令队列，最小

总线周期仅为 40ns，所有 I/O 口与存储器统一编址。MC9S12 使用低功耗晶振，复位控制、看门狗和实时中断等配置及功能有助于系统(BDM)调试方式而无需仿真器，可以实现硬件断点、条件断点、在线调试等全部调试功能，为单片机的开发提供便利。

右边则为片上的多种外设，除了常规的定时器、串行接口、并行接口外，还包括 ATD、ECT、SPI、CAN、FlexRay 等功能。其中 ATD 是 16 个模拟输入通道，可以设定多种采样方式。而定时模块具有 8 个独立的可编程通道，每个通道都可以设置为输入捕捉/输出比较方式。集成的串行接口 SCI 有两个，SPI 有一个，可以根据应用需求对工作方式及参数进行设置。

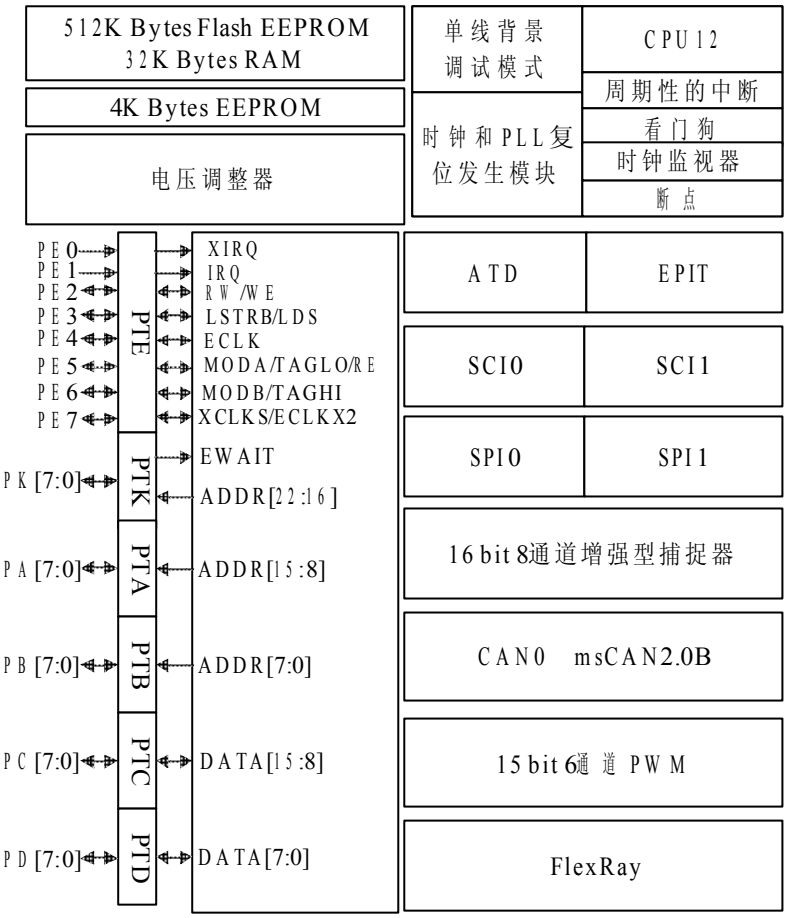

图 3.2 MC9S12XF512 内部结构

3.2.1 集成的 MSCAN 控制器

MC9S12XF512 内部集成了符合博世公司所定义的 CAN2.0A/B 协议的 MSCAN 控制器，支持标准和扩展的数据帧，最高可达到 1Mbps 的可编程比特率，拥有 5 个“先进先出”的接收缓存器和 3 个带有内部优先级选择的发送缓冲器。同时它使用灵活的可屏蔽标识符过滤器，并且可以分为 2 个 32 位、4 个 16 位或 8 个 8 位三种过滤方式。使用了可编程的时钟源，可选择总线时钟或者振荡器时钟。在发送和接收缓冲区具有已发送和接收数据的时间标记功能。

MSCAN 的模块结构如图 3.3 所示，MSCAN 控制器通过一系列的控制和状态寄存器对整个 CAN 通信的操作进行设置。

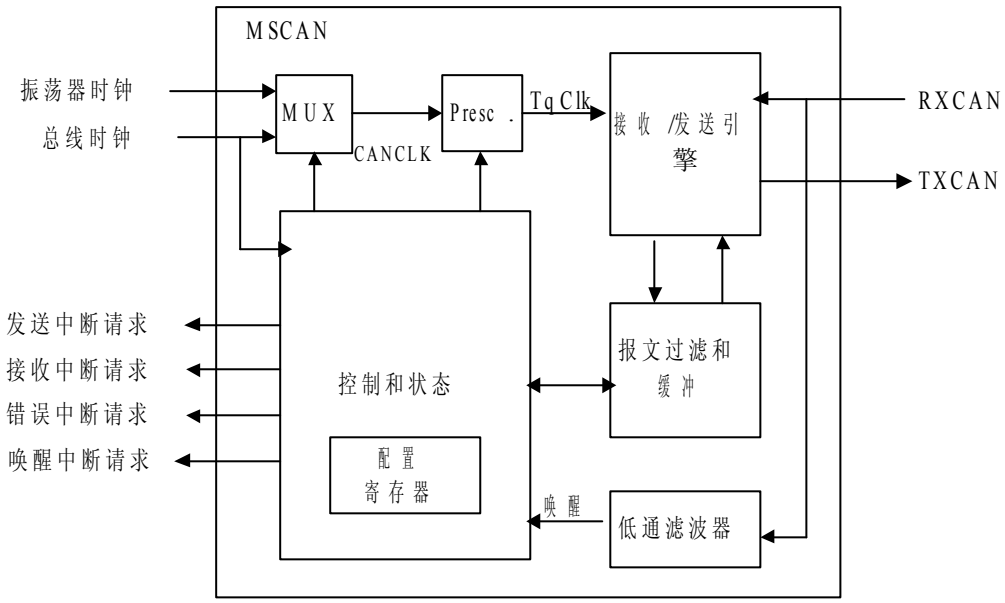

图 3.3 MSCAN 控制器

MSCAN 的发送和接收缓冲区的结构相同，每个缓冲区以寄存器的形式由 16 个字节组成，整个消息缓冲区的存储结构如图 3.4 所示，其中数据字节占 13 个，图中 ID[28:0]表示的是 29 位扩展标识符，IDE 位置 1 时，表示使用扩展标识符，RTR 用来区分数据帧和远程帧，DLC 寄存器控制一段数据所包含的字节数，TBPR 表示传送缓冲区优先级寄存器，Data Byte0-7 用来存储 8 个字节的 CAN 数据，位于最后两个字节的 Time Stamp 用来存储时间标记，它只能由 MSCAN 模块写入，CPU 只能读取。

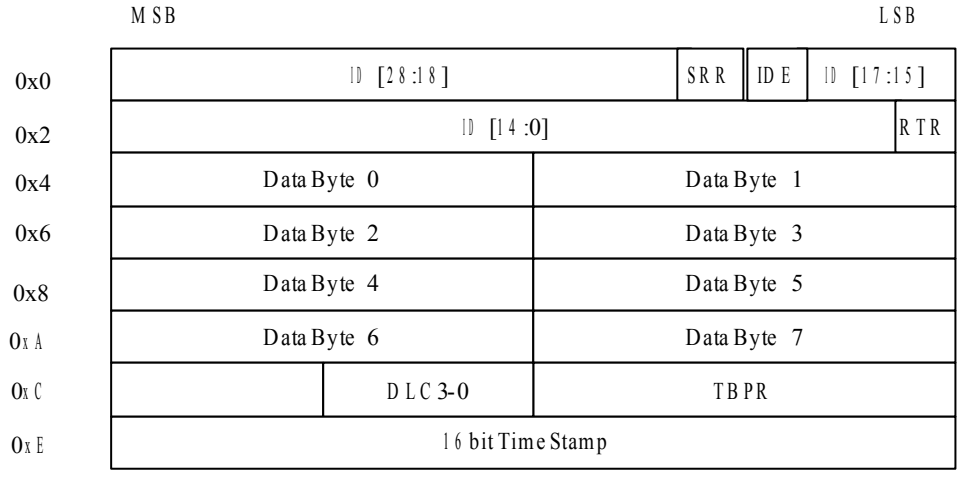

图 3.4 消息缓冲区结构

3.2.2 节点复位电路

CPU12 包含有硬件和软件复位。CPU 在复位后取得相应的中断向量并跳转到中断向量所执行的地址处执行程序，并且单片机的各个寄存器和控制位将被

预置为默认状态。MC9S12XF512 包括有上电复位、外部复位引脚引起的复位、看门狗复位和时钟监视器复位这 4 个复位源。

MC9S12XF512 的上电复位电路如图 3.5 所示，其中复位信号通过 4.7k 电阻后直接连接 VCC，再通过一个 0.1uF 的电容接地以滤除干扰信号，当按键被按下时，RST 信号接地，从而在 MCU 的 RESET 引脚外加了一个低电压使得 MCU 强制复位。

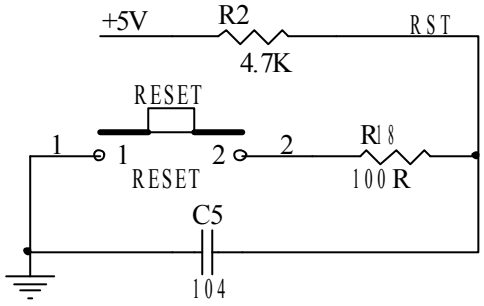

图 3.5 节点复位电路

3.2.3 CAN 通信接口电路

如图 3.6 所示，在 MSCAN 控制器与收发器之间，使用高速双通道光耦合器 HCPL-2630 来进行连接，从而很好地实现了总线上各个 CAN 节点的电气隔离，这一部分虽然增加了接口电路的复杂性，但是提高了整个 CAN 节点的稳定性和安全性能。CAN 收发器采用符合 ISO11898 标准的高速数据收发器 TJA1040[18]，TJA1040 的 CANH 和 CANL 之间并联了两个 60R 电阻并通过一个 30p 的小电容接地，从而可以滤除总线上的高频干扰，并且具有一定的防电磁辐射的能力。MSCAN 控制器的 TXD 和 RXD 引脚通过光耦合器后，连接收发器的 TXD 和 RXD 引脚，最终通过 CANH 和 CANL 引脚接入到总线中。

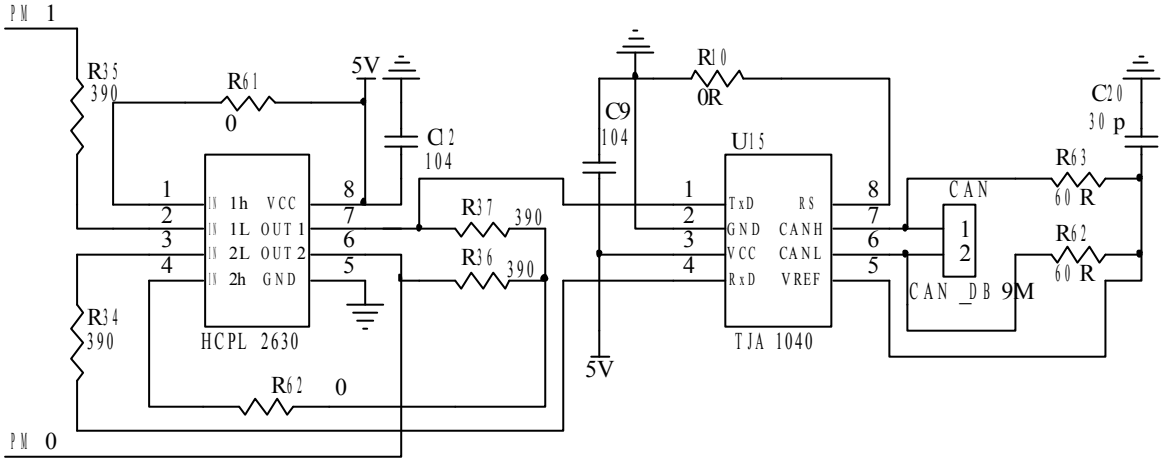

图 3.6 CAN 通信接口电路

3.2.4 串口通信电路

串口通信电路通过 MAX232CASE 芯片来实现，来自主芯片的数字信号通过此芯片后转换为串口通信的物理信号，如图 3.7 所示，其 T2IN 和 R2OUT 引

脚分别连接主芯片内部的 SCI 模块的引脚 PS1 和 PS0 来实现串口通信。

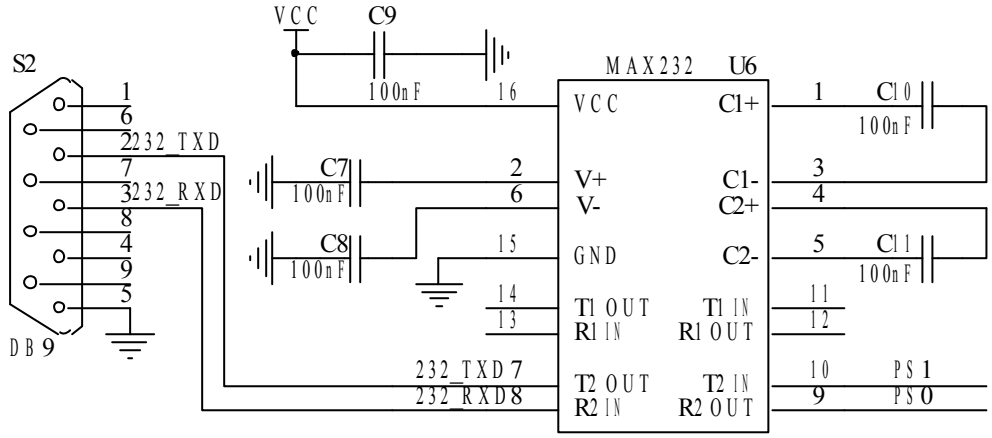

图 3.7 串口通信电路

3.2.5 BDM 下载接口电路

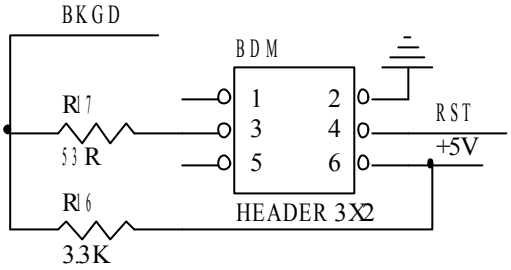

图 3.8 BDM 下载接口电路

BDM(Background Debugging Mode)是摩托罗拉公司支持的一种片上调试模式。通过 BDM 接口可以完成基本的调试功能，包括设置断点、读写内存、读写寄存器、下载程序、单步执行程序、运行程序、停止程序运行等[31]。BDM 调试器则在向目标板下载程序时使用，将单片机 FLASH 中的旧程序擦除，而且 S12 单片机的 BDM 接口提供了很多调试信息，包括有 CPU 运行时的动态信息，因此 BDM 调试器也需要有 CPU 的支持。如图 3.8 所示的电路中，BDM 使用一个 6 针的插头将信号从单片机的第 39 引脚 BKGD 中引出，并与 BDM 调试器相连接，即可实现 BDM 调试，将另一针连接 RST 实现 BDM 调试功能的复位。

3.2.6 电源模块

HC12 单片机的芯片内部使用 3V 的电压，而外部供电的电压及 I/O 端口为 5V。在如图 3.9 所示的电源模块电路中，开发板的电源在先经过电容滤波并去除高频干扰后才可以使用，并且采用了 300mA 的快速熔断器，以防止电源电流过大烧毁单片机。同时设计了 LED 灯以指示系统通电状态。

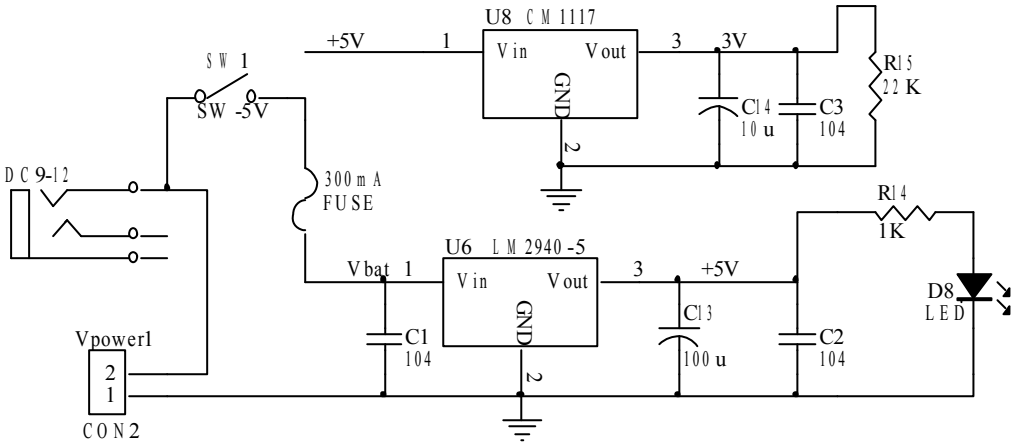

图 3.9 电源模块电路

3.3 μC/OS-Ⅱ内核的工作原理

传统的软件设计模式将主程序设计为无限循环并采用单任务顺序执行方式，同时通过设计中断来处理异步事件。在处理简单的应用时，这种方式是可行的，但当面对实时性要求较高，任务较多的情况时，系统将会变得可靠性、稳定性低，实时性较差。

CANopen 系统中引入μC/OS-Ⅱ，在处理复杂的多任务时，利用嵌入式系统的任务调度及管理功能，使得整个系统在运行时稳定可靠。

作为一个嵌入式多任务实时操作系统，μC/OS-Ⅱ可以大致分为系统内核、任务处理、时间处理、任务同步与通信、CPU 移植等 5 个部分[19]。作为一个公开源代码的占先式内核，它采用 ANSI 的 C 语言编写，并可以根据用户的需要在不同的处理器上进行移植，根据实际应用需求对系统进行固化、裁剪。它可以使各个任务独立工作，互不干涉，很容易实现准时而且无误执行，使实时应用程序的设计和扩展变得容易，使应用程序的设计过程大为减化。

3.3.1 μC/OS-Ⅱ中的任务

在实际中处理问题，习惯采用分而治之的方法，把一个大问题分解成相对简单、易于解决的小问题。现代操作系统几乎都是多任务操作系统。任务即是线程，其运行时认为 CPU 完全属于自己，μC/OS-Ⅱ就是一个能对多任务进行管理和调度的操作系统。

μC/OS-Ⅱ任务的存储结构包括任务程序代码、任务堆栈和任务控制块三个组成部分。任务程序代码包含任务执行的具体内容；任务堆栈中保存了 CPU 寄存器中的内容及任务的私有数据来满足任务切换和响应中断的需要；任务控制块中则保存了任务的堆栈指针、当前状态、优先级等一些与任务管理有关的属性。

μC/OS-Ⅱ最多可以对包括用户任务和系统任务在内的 64 个任务进行管理。因为只有一个 CPU，所以在一个具体的时刻只能允许一个任务占用 CPU，

μC/OS-Ⅱ系统

根据任务的具体情况将其分为以下的五个状态。

休眠状态：指任务驻留在程序空间中，还没有交给内核管理。通过调用OSTaskCreat()或者 OSTaskCreatExt()可以实现将任务交给内核进行管理。

就绪状态：当任务一旦建立，此任务就处于就绪状态并准备运行。由于有更高优先级的任务正在运行，因此它暂时不能运行，并存放在就绪列表中。

运行状态：指该任务经过调度器判断取得了 CPU 的控制权，任何时刻只能由一个任务处于运行状态。

等待或挂起态：正在运行的任务由于调用延时函数 OSTimeDly()或等待事件信号量的来临而将自己挂起，从而处于等待状态，此时其他任务将占用 CPU 的使用权。

中断服务状态：正在运行的任务可以被中断，除非该任务将中断关闭。被中断的任务进入中断服务程序(ISR)。若中断服务程序中存在处于就绪状态的更高优先级的任务，则在中断服务程序结束后，更高优先级的任务开始运行。

μC/OS-Ⅱ通过系统服务函数来实现不同状态之间的转换，具体转换关系以及使用的相关函数如图 3.10 所示

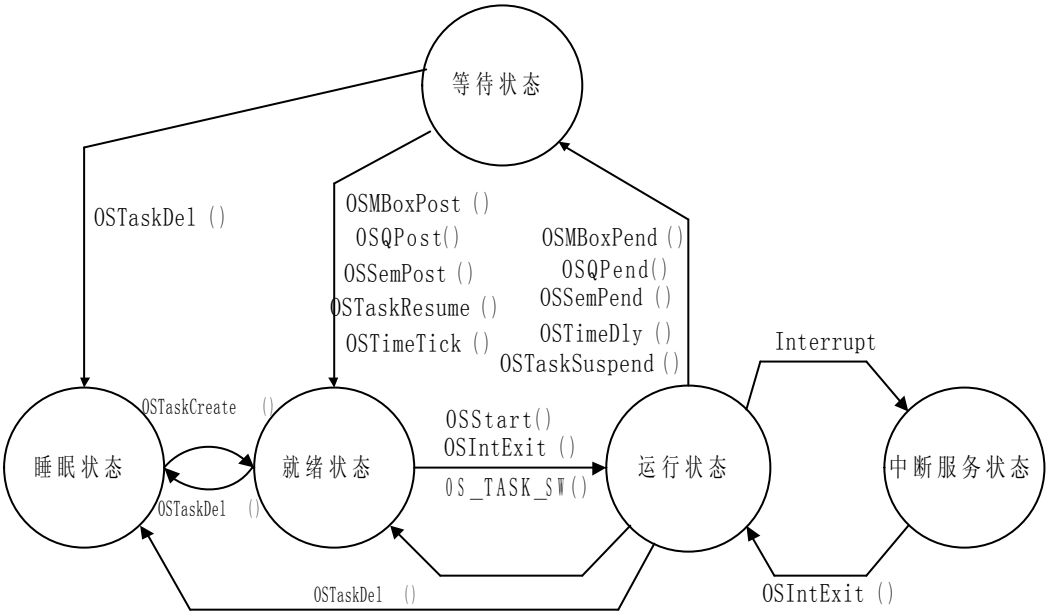

图 3.10 μC/OS-Ⅱ系统状态切换

3.3.2 μC/OS-Ⅱ的中断和时钟

μC/OS-Ⅱ响应中断的过程是当系统接收到中断请求后，若 CPU 此时处于允许中断的状态，那么系统会中止当前运行的任务，而根据中断向量的指向去运行中断服务子程序。在中断服务子程序运行完毕后，系统根据任务切换的情况返回之前中止的任务继续运行或者运行就绪的更高优先级的任务。这是因为μC/OS-Ⅱ系统的内核是可剥夺型的，在中断服务子程序运行结束后，系统会根

据情况再次进行一次任务调度。μC/OS-Ⅱ可以进行中断嵌套，全局变量OSIntNesting用来记录中断嵌套的层数。

当进入中断时，系统将运行当前任务的CPU寄存器的数值推入任务栈，此时将调用进入中断服务函数OSIntEnter()将中断嵌套层数OSIntNesting加1。在完成中断服务程序的处理后，将调用退出中断服务函数OSIntExit()来获取当前最高优先级别就绪任务的优先级，若有优先级更高的任务进入就绪状态，那么系统将获取该任务控制块的指针，执行中断级的任务切换。

μC/OS-Ⅱ的时钟通常是由一个硬件计数器定时产生周期性中断信号来实现的。每一次中断叫做一个时钟节拍，这一中断服务程序叫做OSTickISR()。在这一函数中通过调用时钟节拍服务函数OSTimeTick()来在每个时钟节拍获得每个任务的延时状态，若有任务已经达到延时时限，则将这一非挂起的任务进入就绪状态。

3.3.3 任务的同步和通信

任务的同步依赖于任务之间的通信。在μC/OS-Ⅱ中，使用信号量、邮箱和消息队列这些被称作事件的中间环节来实现任务之间的通信，并且统一用事件控制块来描述这些不同类型的事件。

信号量是一类事件，其目的是为了共享资源设立一个表示该共享资源被占用情况的标志。任务在访问共享资源前，先要对此标志进行查询，在了解资源的被占用情况后，再决定下一步的行动。信号量由两部分组成，包括16位的信号量计数值和等待该信号量的任务等待表，其中使用事件控制块成员OSEventCnt作为计数器，而利用OSEventTbl[]数组来作为等待任务表。

用户可以通过将OS_CFG.H中的OS_SEM_EN开关量设置为1，从而在μC/OS-Ⅱ中使用信号量服务。同时μC/OS-Ⅱ提供了五个函数对信号量进行建立、查询、等待、发送和请求操作。

消息邮箱是根据应用需求在内存中建立的一个数据缓冲区，当任务之间需要传递不同的数据时，可以将要传递的数据放在这个缓冲区里，从而实现了任务之间的数据通信，并把此事件控制块的成员OSEventType设置为OS_EVENT_TYPE_MBOX，将数据缓冲区的指针赋值给事件控制块成员OSEventPtr。

当需要在任务之间传递多条消息时，就需要使用消息队列，此时需要将事件控制块的成员OSEventType设置为OS_EVENT_TYPE_Q，此时事件控制块成员OSEventPtr将指向一个叫做队列控制块(OS_Q)的结构，该结构对指向消息的指针数组MsgTbl[]进行管理。

3.3.4 μC/OS-Ⅱ多任务实现机制分析

事实上，在单一CPU的情况下，并不存在真正的多任务机制，存在的是不

同的任务轮流使用 CPU，因此其本质上还是单任务的。但由于 CPU 的执行速度很快，任务可以频繁并且快速的切换，因此好像是很多任务在同时运行一样。可见，要实现多任务机制，则目标 CPU 要具有在运行期间更改 PC 指针的途径，否则无法做到切换。而目前还没有 CPU 有指令可以直接设置 PC 指针，但是一般指针可以利用 JMP，CALL 这样的指令间接地修改 PC 指针。μC/OS-Ⅱ系统使用软中断指令来修改 PC 指针，当发生中断的时候，CPU 保存当前的 PC 指针和寄存器的值到堆栈里，而后将 PC 设置为中断程序的入口地址，从而在下一个机器周期去执行中断服务程序。执行完毕后，一般再执行一条 RETI 指令，此指令将当前堆栈里的值回复到 CPU 的状态寄存器和 PC 指针中，系统因此可以在中断以前的地方继续运行。

μC/OS-Ⅱ中任务切换通过任务切换宏 OS_TASK_SW()设置软中断，并将此软中断的中断向量指向 OSCtxSw()函数。如前面所述，通过任务切换宏产生软中断后，OSCtxSw()函数将被中止任务的断点指针和 CPU 通用寄存器的值保存在堆栈中，然后将被中止任务的任务堆栈指针当前值保存到该任务的任务控制块的 OSTCBStkPtr 中。获取待运行任务的控制块后，从中获得待运行任务的堆栈指针，从该堆栈中将通用寄存器的值恢复到 CPU 中。此时 CPU 获得了待运行任务的指针，从而完成任务的切换并开始运行新的任务。μC/OS-Ⅱ中的任务级任务切换函数和中断级任务切换函数在本质上是相同的，二者分别是通过函数 OSSched()和函数 OSIntExt()来完成任务的切换。对于 OSSched()，其内部通过调用 OS_TASK_SW()来人为地产生一次中断，而对于 OSIntExt()，由于中断服务函数已经将 CPU 寄存器和中断发生前正运行的任务堆栈指针保存过，因此 OSIntCtxSw()即是 OSCtxSw()在调用钩子函数的后半部分。

3.4 μC/OS-Ⅱ在 MC9S12 芯片的移植

μC/OS-Ⅱ在移植过程中要考虑三个一般性的问题[18]。

(1)可重入函数

因为在多任务操作系统环境中，两个任务可能产生同时调用同一个函数的情况，因此需要系统提供的函数允许多个任务调用，而不会通过函数中变量的耦合引起任务之间的干扰。这样的函数即是可重入函数。一般而言，可重入函数应该使用局部变量，而不应该使用全局变量，如果一定需要使用全局变量，则需要对其做必要的保护。某一种芯片若需要移植μC/OS-Ⅱ系统，那么其编译器必须支持可重入代码的产生。

(2)时钟节拍的产生

μC/OS-Ⅱ通过硬件中断来实现系统时钟，并在时间中断服务函数中来处理与时间有关的问题。因此要移植的处理器必须具备响应中断的能力，同时应该具有开关中断的功能。

(3)任务堆栈的设计

为确保μC/OS-Ⅱ系统的正常运行，处理器内部需要具有一定数目的硬件堆栈，而且能够通过读写的指令对堆栈指针进行操作。不同的芯片内部定义的堆栈生长方向可能不相同，因此在移植时，需要根据具体情况正确地定义堆栈的格式。

μC/OS-Ⅱ系统的代码是采用 C 语言和汇编语言实现的，其中大部分的代码是用 C 语言编写，而与处理器密切相关的少数代码是采用汇编语言编写的，所以用户需要对与处理器相关的若干代码进行修改后就可以把它移植到各类 8 位、16 位和 32 位嵌入式处理器上。

μC/OS-Ⅱ系统的整个体系结构如图 3.11 所示,在进行移植时，只要修改与处理器相关的代码[32]，包括：OS_CPU.H、OS_CPU_C.C 和 OS_CPU_A.ASM，以下从与移植有关的声明和函数两方面进行代码的修改。

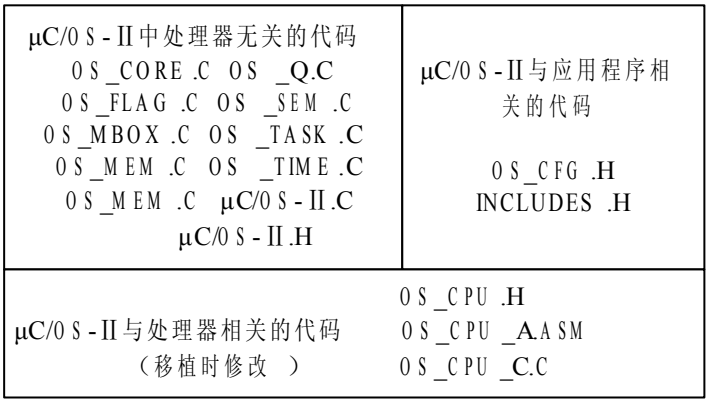

图 3.11 μC/OS-Ⅱ系统源代码结构

(1) 与移植相关的声明的修改

在 OS_CPU.H 中定义了与编译器有关的数据类型、开关中断宏、堆栈的生长方向以及任务切换宏[33]。在 MC9S12 中堆栈是按字节进行操作，因此定义堆栈数据类型 OS_STK 为 8 位。MC9S12 处理器的堆栈是由高地址向低地址增长的，所以常量 OS_STK_GROWTH 设置为 1。

μC/OS-Ⅱ在进入系统临界代码区之前要关闭中断，等到退出临界代码区后才打开中断，这样可以防止多任务环境下的其他任务或中断破坏核心数据。μC/OS-Ⅱ定义了两个宏来关闭/打开中断：OS_ENTER_CRITICAL() 和 OS_EXIT_CRITICAL()。

μC/OS-Ⅱ提供三种方法来开关中断，本移植选择如下的开关中断的方法，这种方式无论在开关中断前后，CPU 的状态寄存器都不会因为发生中断而改变。

```
#define OS_ENTER_CRITICAL()   asm tpa；asm sei; asm staa cpu_sr
#define OS_EXIT_CRITICAL()    asm ldaa cpu_sr;asm tap
```

任务切换宏 OS_TASK_SW()是在μC/OS-Ⅱ从低优先级任务切换到最高优

先级任务时被调用的，OS_TASK_SW()通过模拟一次中断过程，在中断返回的时候进行任务切换。MC9S12 提供了软中断源和陷阱中断源，这两个中断源都可以使得处理器的寄存器状态保存到将被挂起任务的堆栈中。由于芯片中不存在监控程序，所以利用 MC9S12XF512 提供的软中断指令 SWI 来定义 OS_TASK_SW()， 并将其中断服务程序的入口点指向 OSCtxSw()，如下所示

```
#define OS_TASK_SW()   asm swi
```

(2) 与移植相关的函数的修改

①任务切换函数

OSCtxSw()是一个任务级的任务切换函数，它在任务中调用。OSCtxSw()首先将被中止任务的断点指针和 CPU 通用寄存器的内容保存在任务堆栈中，然后把被中止任务的任务堆栈指针当前值保存到该任务的任务控制块的 OSTCBStkPtr 中。在获取待运行任务的任务控制块后，使 CPU 通过任务控制块获得待运行任务的任务堆栈指针，并将待运行任务堆栈中通用寄存器的内容恢复到 CPU 的通用寄存器中，最后使 CPU 获得待运行任务的断点指针，执行中断返回指令。对于中断级任务切换函数，由于中断服务函数已经将 CPU 寄存器和中断发生前正运行的任务堆栈指针保存过，因此 OSIntCtxSw()即是 OSCtxSw()在调用钩子函数的后半部分。在 MC9S12 中，当中断发生时芯片会将 CPU 寄存器推入堆栈，但是不会包括页面管理寄存器，因此需要用汇编语言加入其入栈和出栈的操作，函数代码如下所示：

```
void   OSCtxSw(void)        /*swi 中断函数*/
{
    asm
    {
        ldaa PPAGE;                 //页面寄存器 PPAGE 入栈
        psha;
        nop;
        ldx OSTCBCur;           //获取 TCB 的地址
        sts 0,x;                  //将 SP 保存到 TCB 的第一个字
    }
    OSTaskSwHook();                 //调用接口函数
    OSTCBCur = OSTCBHighRdy;   //把最高优先级的任务控制块传递给当前任务快
    asm
    {
        ldx OSTCBCur;            //获取新的 TCB 地址
        lds 0,x;                    //从当前任务控制块中获得堆栈地址
        pula;
        staa PPAGE;               //PPAGE 出栈
        nop;
        rti;
    }
}
```

②任务堆栈初始化函数

任务堆栈初始化函数 OSTaskStkInit()在创建任务的时候调用，用来在任务堆栈中按照一定顺序初始化任务最初的数据。在 MC9S12XF512 芯片中，根据

数据存放的顺序，从高到低位对 opt 参数、PC 寄存器、Y 寄存器、X 寄存器、D 寄存器、CCR 寄存器和 PPAGE 寄存器进行初始化，并最后返回堆栈指针所指向的地址。其中 PC 的值设置为任务的入口地址并存放两次，第一个值是建立扩展任务所需的；D 的值初始化为参数 pdata 的值用来传递任务参数；PPAGE 则是用来存储页面寄存器的值。

```
OS_STK *OSTaskStkInit (void (*task)(void *pd),
void *pdata, OS_STK *ptos, INT16U opt)
{
    INT16U    *stk;
    #ifdef BANKED
    INT16U    RegPage;                    //用来存储 PPPAGE
    INT16U *OffsetAddress;
    #endif
    stk = (INT16U *)ptos;               //加载堆栈指针
    *--stk = opt;                       //有一个字节空闲
    #ifdef BANKED
    OffsetAddress = (INT16U *)&task;    /* 16bit  地址*/
    RegPage = (INT8U)task;                  /* PPAGE */
    *--stk = (INT16U)*OffsetAddress;        /
    *--stk = (INT16U)*OffsetAddress;          //PC
    #endif
    *--stk = (INT16U)(0x1122);       //Y  寄存器
    *--stk = (INT16U)(0x3344);       //X  寄存器
    *--stk = (INT16U)pdata;            //D  寄存器:A,B
    *--stk = (INT16U)(0x00);        //CCR,16 bits in S12X
    #ifdef BANKED
    ((INT8U *)stk)--;                     // PPAGE 需要一个字节
    *(INT8U *)stk = RegPage;            //PPAGE
    #endif
    return((OS_STK *)stk);
}
```

③时钟节拍中断服务函数

和其他中断服务程序一样，首先 OSTickISR()在被中断任务堆栈中保存 CPU 的值，然后调用 OSIntEnter(),使得中断嵌套层数全局变量 OSIntNesting 加 1。随后通过调用 OSTimeTick()，遍历任务控制链表中的所有任务控制块，把各自用来存放延时时限的 OSTCBDly 变量减 1，若其计数值为 0，则表明该任务进入就绪状态,最后调用 OSIntExit()标志着时钟节拍中断服务子程序的结束。本次移植中时钟节拍由硬件产生，必须设置好实时时钟的控制寄存器，使得 MC9S12 芯片在产生相应中断后，调用处理程序。实时时钟控制寄存器设置如下：

```
RTICTL=0x49;     //每秒产生 100 次中断
CRGINT|=0x80;    //使能中断
```

在 MC9S12 芯片中，时钟节拍中断服务函数的实际代码如下：

```
interrupt 7 void    OSTickISR(void)
{
        asm
        {
            ldaa PPAGE;            //PPAGE 入栈
```

```
            psha;
        }
        OSIntEnter();
        OS_SAVE_SP();
        CRGFLG |= 0x80;     /*清除 rti 中断标志位 */
        OSTimeTick();
        OSIntExit();        /* 退出中断并进行任务切换*/
        asm
        {
        pula;
        staa PPAGE;          /* PPAGE 出栈*/
        nop;
        rti;
    }
}
```

μC/OS-Ⅱ系统经过上述主要的修改后在 MC9S12 芯片中运行起来。μC/OS-Ⅱ的引入使得系统开发的效率得以提高，编写程序时，只要将相关的功能封装成任务，并根据任务的轻重缓急设定优先级，启动多任务环境后，由μC/OS-Ⅱ系统来管理这些任务。

3.5 在μC/OS-Ⅱ中嵌入 CANopen

首先应该注意μC/OS-Ⅱ和 CANopen 源代码对于不同数据格式的定义是不同的。当将二者一起进行编译时，就会出现错误提示数据格式定义矛盾。因此需要对源代码进行适当的修改使得二者相同类型的数据格式的定义是一致的。μC/OS-Ⅱ中的数据格式定义在头文件 OS_CPU.H 中，而 CANopen 的数据格式定义在 applicfg.h 中，通过对比代码量，决定修改 applicfg.h 中关于数据结构的定义，使得二者在相同类型的数据结构定义是一致的，而二者独有的数据结构定义，由于不会产生矛盾，就不必修改而保持原有格式。

CANopen 内核和μC/OS-Ⅱ都使用了定时器作为各自的时钟节拍，为避免冲突，应该各自使用不同的定时器作为时钟节拍，并且考虑必须考虑定时器中断的优先级问题。CANopen 内核时钟的中断处理的程序和占用的时间相对更小，为了能更及时地响应这一中断要求，将分配给它更高的优先级。CANopen 采用定时器 0 作为时钟节拍，用来产生 1ms 的时钟节拍，而μC/OS-Ⅱ则采用定时器 3 来产生其时钟节拍。

将 CANopen 嵌入到μC/OS-Ⅱ系统的主要思想是将 CANopen 内核封装成一个任务在μC/OS-Ⅱ系统中运行。在任务的设计上，把 CANopen 内核任务的优先级设计为最低。当没有更高的任务处于就绪状态时就一直运行 CANopen 内核。其他的任务用户可以根据实际应用需求自己设定，完成这些任务就挂起，从而让 CANopen 任务有足够的时间进行数据的发送和接收。

CANopen 从站的主程序是其状态在 NMT 报文下进行切换的过程，并在四个状态下运行各自的功能函数，用 Switch 语句实现状态的转换。在移植过程中，将其封装成如下所示的一个任务：

```
    void CANopenTask(void *pdata)
{   e_nodeState lastState=unknown_state;
    pdata=pdata;
    while(1)
    {   switch(getState())
        {     case Initialisation:
                initialization();
                        .....
          case Pre_operational:
                preoperational();
                        .....
          case Operational:
                operational();
                        .....
          case Stopped:
                stopped();
                        .....}
    }
}
```

封装好 CANopen 任务后，在μC/OS-Ⅱ系统上重新定义一个主函数，完成操作系统的初始化，创建任务以及启动任务的过程，如下所示，其中 OSInit()用来完成操作系统的初始化，OSTaskCreate()函数用来创建μC/OS-Ⅱ系统的任务，通过 OSStart()函数启动操作系统，运行任务。

```
void main(void)
{
    INT8U err;
    OSInit();
    OSTaskCreate(CANopenTask，(void* )0，& CANopenTask
    [TASK_STK_SIZE-1]，TASK_CANopen_PRIO);
    .....
    OSStart();
}
```

3.6 本章小结

本章主要介绍了基于 MC9S12XF512 芯片的 CANopen 节点硬件架构，重点阐述了 CAN 总线接口电路以及其他外围功能电路的设计；详细分析了μC/OS-Ⅱ系统内核的工作原理，通过修改与移植有关的声明和函数，成功将μC/OS-Ⅱ系统移植到 MCU 中运行；将 CANopen 从站协议状态机封装成任务后在μC/OS-Ⅱ中运行，实现 CANopen 从站协议到μC/OS-Ⅱ中的嵌入。

第四章 CANopen 协议软件实现

4.1 CANopen 软件功能需求分析

CANopen 协议栈软件的实现是本课题的重点。集成有 MSCAN 控制器的 MC9S12XF512 芯片实现了 CAN 总线底层协议，在完成 CAN 总线底层驱动的编写后，CAN 总线的硬件电路可以实现基本的点对点以及一点对多点的 CAN 通信。在此基础上要完成 CANopen 协议的内核功能，必须实现以下基本的功能：

- 建立完善的对象字典。
- 能够实现 CANopen 的服务数据对象(SDO)、过程数据对象(PDO)、网络管理对象(NMT)以及一些特殊对象的功能。
- SDO 能够完成下载、上传等通信操作，且支持多个客户端。
- PDO 的传输可以支持多种触发条件，包括事件触发、远程请求触发(RTR)或周期同步触发。
- NMT 实现网络节点通信状态的控制、心跳协议和节点保护功能，使得 CANopen 主节点能够准确地检测到从站的工作和故障状态。
- 能够实现 CANopen 从站状态机，完成从站在各个工作状态之间的切换。
- 实现 SYNC 通信对象的功能。

4.2 CANopen 软件整体架构

整个 CANopen 协议栈的软件在 FreeScale 的集成开发环境 CodeWarrior 下采用 C 语言来实现，软件设计遵循模块化设计的原则，并依据协议栈的分层结构和数据流，按照如图 4.1 所示的具体结构进行设计，包括有硬件驱动层、μC/OS-Ⅱ系统层、CANopen 通信协议以及设备行规应用层。

底层提供 CAN 及设备驱动接口，CAN 驱动接口处理 CAN 的数据帧信息，获得其中的有效数据进行存储并提供给上层通信对象使用。

在中间的μC/OS-Ⅱ操作系统主要完成任务调度、同步，时钟等功能，通过创建一个 CANopen 从站任务，里面维护着 CANopen 状态机；创建一个定时器中断任务，主要用来为协议栈提供时钟机制，如心跳报文计时和节点保护服务及同步对象；针对不同的应用需求，创建 CANopen 通信任务。

顶层是 CANopen 协议栈的实现，包括μC/OS-Ⅱ任务中维护的状态机，它是整个 CANopen 从站协议运行的载体，包括一个报文接收分发处理函数，对接收到的通信对象进行分类，并分发给相应对象的处理函数；对象字典里面存储着 CANopen 协议运行需要的所有参数和数据，并通过接口函数进行读写操作；四类通信对象采用模块化设计，功能相对独立。

协议的具体实现参照 CANopen 预定义连接集，支持 3 个发送 PDO(TPDO)，3 个接收 PDO(RPDO)，1 个 SDO 以及 NMT 管理对象。同时采用结构体的方式

定义 CAN 报文、对象字典以及缓冲存储区等。

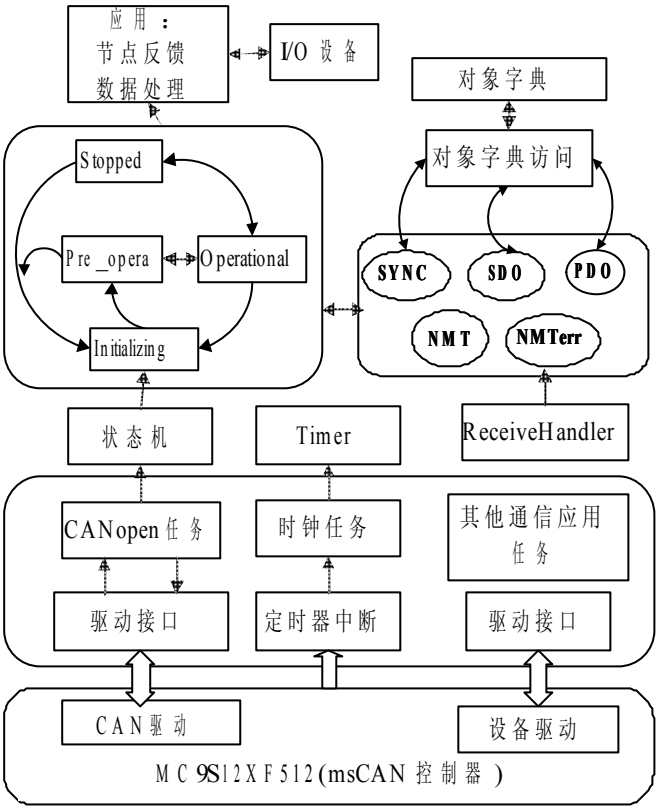

图 4.1 CANopen 协议栈软件分层结构

从节点的主程序流程如图 4.2 所示，从节点在 NMT 管理报文的作用下完成四种状态之间的互相转换，程序则采用 SWITCH 结构对接收到的 NMT 报文进行分析，从而对从节点状态进行相应的操作。节点上电后进行芯片和 CAN 控制器及其通信参数的初始化，完成后向主节点发送 Boot-up 启动报文，通知主节点它已进入预操作状态。从节点处于预操作状态时，可接收来自主节点的 SDO 报文对它的对象字典进行配置，并且可以进行心跳报文通信。当从节点接收到主节点的 NMT 报文时，将状态切换到操作状态。在这一状态下，主节点和从节点可通过 PDO 进行实时数据的交互。当再接收到 NMT 报文时，节点进入停止状态，在这一状态下不可以进行 PDO 和 SDO 通信，但是可以继续对已接收到的报文进行处理。一旦 CAN 控制器接收到新的报文，报文经过底层驱动接口的处理后，从该报文的 COB-ID 判断出该报文具体属于何种对象，并进入相应的报文处理子程序。

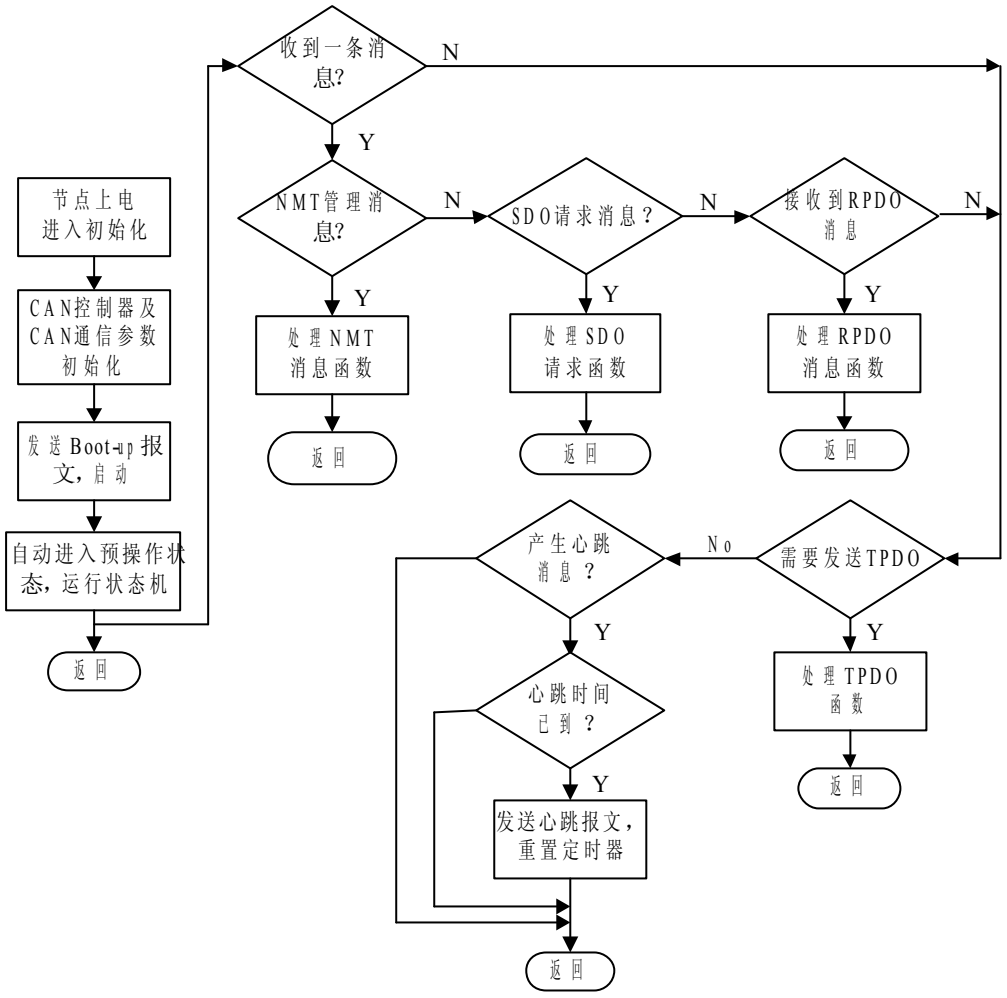

图 4.2 CANopen 从站主程序流程

4.3 CAN 底层驱动程序设计

基于 MC9S12XF512 芯片的 CAN 底层驱动程序主要包括 CAN 控制器初始化函数、CAN 报文发送程序、CAN 报文接收程序以及中断函数的设置。

4.3.1 CAN 控制器初始化程序

CAN 控制器初始化的内容主要包括 CAN 总线波特率、滤波器的设置，CAN 控制器时钟的选择。由于控制器进入初始化模式前，MSCAN 需要处于非活动的状态，因此在 MSCAN 进入初始化模式前，先使其进入休眠模式。同时 MSCAN 控制寄存器 CANCTL0 和 CANCTL1 中的 INITRQ 和 INTTAK 位用来锁定和开启寄存器，在非初始化模式下，相应的寄存器被锁定而不允许用户修改。在初始化模式下，MSCAN 控制器不工作，但是寄存器可以访问，因此可以对只有在初始化模式下才可以写的 MSCAN 总线定时寄存器和与标识符相关的寄存器进行设置，具体的初始化流程如下图 4.3 所示。

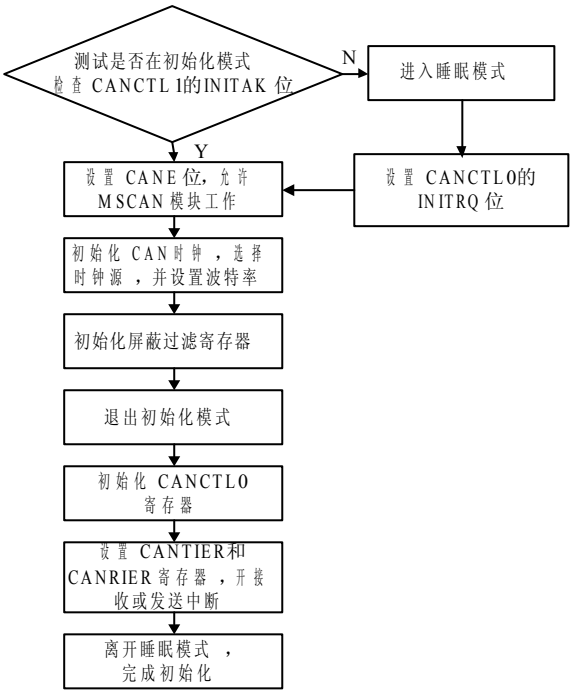

图 4.3 CAN 控制器初始化

4.3.2 CAN 报文发送

在需要传送数据时，CPU 首先根据 CANTFLG 寄存器的 TXEx 位来确认传送缓冲区是否可用。若传送缓冲器可用，则 CPU 通过写 CANTBSEL 寄存器来将指针指向此缓冲区，从而可以对相应的缓冲区进行访问。CANTBSEL 寄存器的算法特点是使用最低被置 1 的比特位作为有效指示位，若 TX1=1 且 TX0=1，则选择 TX0 作为传送缓冲区；若 TX1=1 且 TX0=0，则选择 TX1 作为发送缓冲区，通过这一方式可以简化传送缓冲区的选择。而后 CPU 将标识符、控制位以及数据存入缓冲区内并标记此缓冲区为准备发送同时清零 TXE 标志。当有多个缓冲区准备发送数据时，MSCAN 模块将通过每个缓冲区内的一个 8 位的优先级域来判断所发送数据的优先级，并根据这一内部的优先级来决定谁先发送。CAN 报文数据的发送流程如图 4.4 所示。

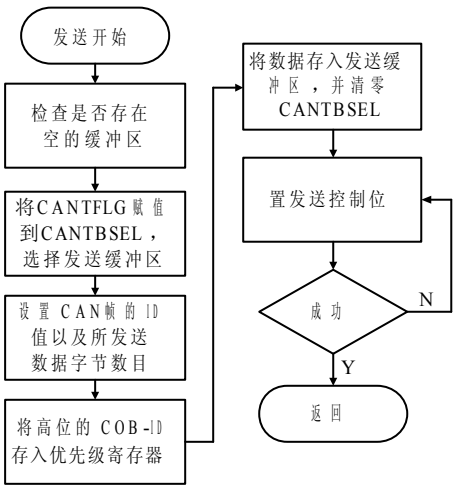

图 4.4 CAN 报文发送

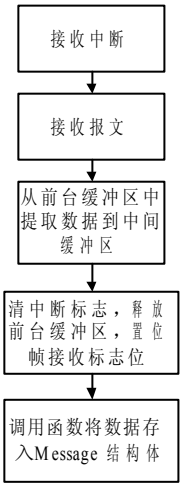

图 4.5 CAN 报文接收

4.3.3 CAN 报文接收

MSCAN 控制器有 5 个缓冲区被映射到相同的内存单元，这些缓冲区都具有 15 个字节来存放 CAN 帧的标识符、控制位、数据以及时间标记。接收数据时，每个数据由过滤器进行检查是否能通过，并写入缓冲区 RxBG 中。接收好数据后，MSCAN 将 RxBG 中的数据送到 FIFO 缓冲区中，对 CANRFLG 寄存器的 RXF 位进行置位，从而产生接收中断。CAN 节点在网络中并不是一直在活动的，因此本文在数据接收上采用中断的方式。用户通过中断服务程序将数据从前台缓冲区中读出，并将 RXF 标志进行复位来响应中断，从而释放前台缓冲区。为避免缓冲区溢出，开辟出一个队列作为中间缓冲区来存储接收到的数据，最后调用 f_can_receive()函数将中间缓冲区的数据存入 Message 结构体数组，具体流程如图 4.5 所示。

4.4 CAN 硬件接口程序设计

CAN 底层驱动程序在进行数据的发送和接收时，需要与上层应用层进行数据的交互。在这一接口程序设计利用了 CAN 的 Message 结构体，并将 CAN 数据的发送和接收操作封装在函数 f_can_send()和 f_can_receive()中。

Message 结构体如下所示，函数 f_can_send()从 Message 结构体中读取 CAN 信息帧的 COB-ID、控制位、数据长度及数据等信息存入 CAN 控制器里与发送相关的寄存器进行数据的传送。

```
typedef struct{
        Uint32 cob-id;      //   CAN 帧 COB-ID
        Uint8   rtr;        //   消息的 RTR 位，区别远程帧和数据帧
        Uint8   len;        //   数据长度
        Uint8   data[8];    //    8 字节数据域
}Message
```

在进行数据接收时，首先通过接收中断，数据进入缓存区后，通过函数 f_can_receive()，将 CAN 数据帧信息从缓存区中写入 Message 结构体数组。CANopen 通信对象通过函数对 Message 进行读或写，而不直接对缓冲区中的 CAN 数据进行操作。通过这种方式，高层协议在对 CAN 数据进行操作时，面对的是统一的保存在 Message 结构体中的 CAN 数据，因此 CANopen 通信对象不需要关心底层驱动的实现过程及执行环境，从而将 CANopen 通信协议层和底层驱动分隔开来，使得应用层的程序与具体的平台无关，便于移植。

4.5 对象字典的建立及读写

表 2.2 给出了对象字典的基本结构，在具体应用中，对象字典 1000h-100A 的部分是基本的设备参数，包括设备类型、错误寄存器、SYNC 等参数。1200h-1BFFh 部分应用于 PDO 和 SDO 的配置，对象字典的其他部分依据 DS401 等其他规范根据具体应用而设置[37]。CANopen 对象字典采用的主索引和子索引

结构形成了是一种自然的二维数组结构。因此协议栈对象字典的 C 语言实现是基于数组和数组的变型[24]。一个对象的信息应该包括：索引、子索引、数据、数据类型和访问类型。二者采用如图 4.6 所示的结构进行构建，主索引所定义的结构体中包含子索引的数目以及指向子索引的指针。子索引结构体则包含有对象的读写权限、数据类型以及数据长度，并使用指针来指向具体的对象。

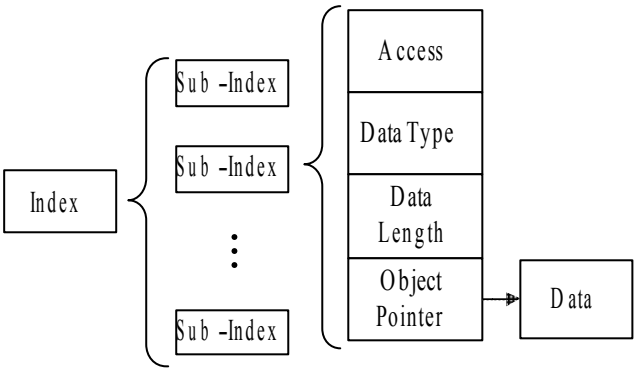

图 4.6 对象字典索引结构

主索引和子索引结构体的 C 语言实现如下所示：

```
typedef struct td_subindex
{
      enum e_accessAttribute    bAccessType;
      Uint8    bDataType;
      Uint8    size;
      void*       pObject;
} subindex;
typedef struct td_indextable
{
      subindex*     pSubindex;
      Uint8    bSubCount;
} indextable;
Uint8    OBJNAME = 0x0;
subindex Index1001[] =
{
  { RO, Uint8, sizeof(Uint8 ), (void*)&OBJNAME }
};
```

其中 bAccessType 表示数据的访问类型，分为只读、只写和可读写，bDataType 表示数据类型,指针*pObject 指向所包含的数据，通过结构体 indextable 来定义主索引，其中 bSubCount 表示此索引下子索引的数目，最后给出了对象字典中索引为 1001H 的错误寄存器的定义，RO 表示其访问类型是只读，其中 OBJNAME 中的数据即为对象字典该索引的具体内容。

对象字典的读写操作通过 getODentry 和 setODentry 两个函数来进行，而两个函数均需要调用 scanOD 函数来对整个对象字典进行扫描来获取指向对象字典的首地址及相应的偏移量。整个对象字典依据索引类别和功能不同建立为如表 4.1 所示的不同的数组区域，待扫描索引与各个区域的首索引和尾索引进行大小比较，以确定待扫描索引的具体处于哪个数组。确定具体范围后将索引与该数组首索引的首地址相减获取偏移量，并最终返回此索引的指针。

getODentry 函数通过 scanOD 获得此索引在对象字典中的具体位置后，将

指针所指向的具体数据内容的指针和数据的大小赋值到相应的变量内，从而完成对该索引的对象字典内容的读取。相应地，setODentry 函数通过对对象字典扫描确定索引具体位置后，将指向新数据的指针赋值到此索引的数据指针，从而完成对象字典的写操作。通过这种分块查找的方式，可以减少索引查询的时间，提高了对象字典的访问效率。

表 4.1 对象字典 C 语言数组分块

对象字典索引数组	索引范围	数组功能说明
CommunicationProfileArea[]	Index1000- Index1018	通信行规区域
serverSDOParameter[]	Index1200	SDO 服务器
clientSDOParameter[]	Index1280- Index1289	SDO 客户机
receivePDOParameter[]	Index1400- Index1402	RPDO 通信参数索引
transmitPDOParameter[]	Index1800- Index1802	TPDO 通信参数索引
RxPDOMappingTable[]	Index1600- Index1602	RPDO 映射参数索引
TxPDOMappingTable[]	Index1A00- Index1A02	TPDO 映射参数索引
manufacturerProfileTable[]	Index2000-Index5FFF	制造商行规索引
digitalInputTable[]	Index6000-Index61FF	通用数字输入变量区域
digitalOutputTable[]	Index6200- Index9FFF	通用数字输出变量区域

4.6 CANopen 接收报文的处理

CAN 底层驱动程序通过中断的方式进行报文的接收，因此报文的接收是随机的，这里定义一个 ReceiveHandler()函数对报文进行处理。CAN 节点经过接收中断后将信息写入缓冲区，而后调用 f_can_receive 函数，将 CAN 帧中的有效数据存入 Message 结构体。ReceiveHandler()函数从该 Message 结构体中获取 CAN 数据 COB-ID 的高四位，而根据 CANopen 预定义连接集的描述，COB-ID 的高四位是用来区分不同通信对象的功能码，因此通过判断高四位功能码来区分所接收到的通信对象，并通过一个指向函数的指针来调用相应的函数对报文进行处理，如图 4.7 所示。这些属于不同的通信对象报文依据如表 2.4 所示的不同的通信模式进行处理。若是确认式的通信模式，则节点需要访问对象字典后获取相关的数据，并向发送节点回复一个报文；而对于非确认的通信模式，节点则只需要在接收并保存好相关数据。

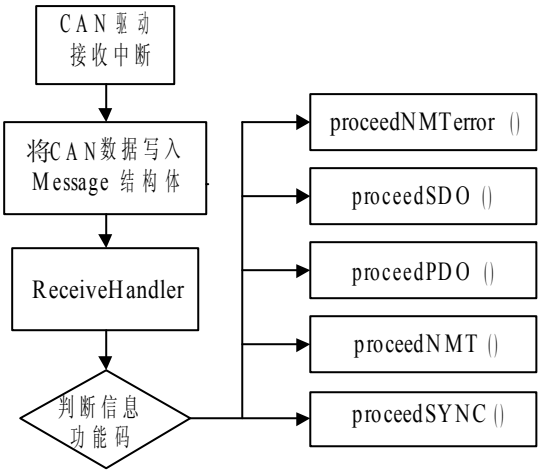

图 4.7 CANopen 报文的接收处理流程

4.7 CANopen 协议实现

4.7.1 SDO

一个节点接收到 SDO 报文后，将首先分析 SDO 帧信息的第一个字节即命令字，并根据对命令字的拆分分析来确定下一步的操作。若 CAN 节点是作为 SDO 通信中的客户机，客户机接收到的报文有两种情况：一是从服务器传送过来的上传回应，里面包含有所请求的数据。二是因为客户机之前发送过下载请求，此时客户机收到来自服务器的回复并确认，如图 4.8 所示。

若节点作为 SDO 通信中的服务器，它将收到来自客户机关于对象字典的读和写的请求。服务器接收到来自客户机的关于对象字典的写操作请求后，将接收到的数据写入对象字典的相关索引，并进一步调用 sendSDO()函数来发送 SDO 应答报文。同样地，服务器接收到来自客户机对对象字典的读操作请求后，将从对象字典相关索引中读取数据，并同样地调用 sendSDO()函数将 SDO 数据发送到客户机。SDO 报文处理的软件流程如图 4.9 所示。

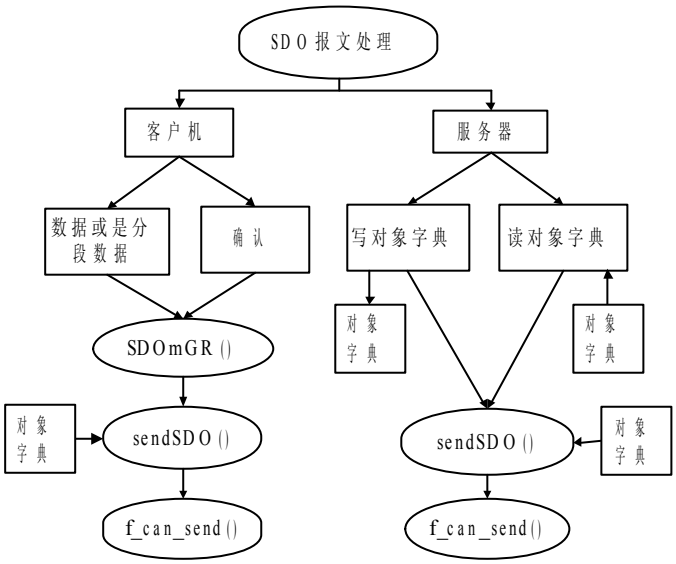

图 4.8 SDO 模块

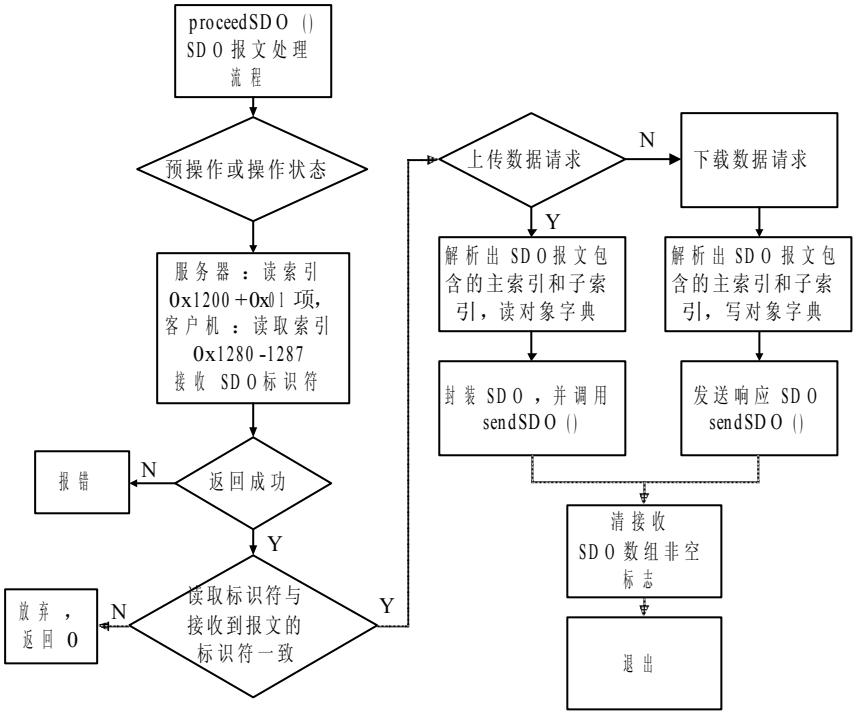

图 4.9 SDO 报文处理流程

4.7.2 PDO

PDO 模块主要分为 RPDO 报文接收和 TPDO 报文发送两方面，如图 4.10 所示。当收到 RPDO 报文时，遍历对象字典里的每一个 RPDO 对象的标识符信息，若与接收到的报文标识符相同，则表示找到了接收该 PDO 信息的对象字典项，并根据 PDO 的通讯参数和映射参数中解析报文。当收到的 RPDO 报文是数据请求时，需要在对象字典中读取相关索引的数据并立即通过 TPDO 进行回应。若接收到的 RPDO 信息只是数据时，则对相关的对象字典索引项进行写操作，将数据存储到对象字典中。报文处理函数 proceedPDO()的流程如图 4.11 所示。

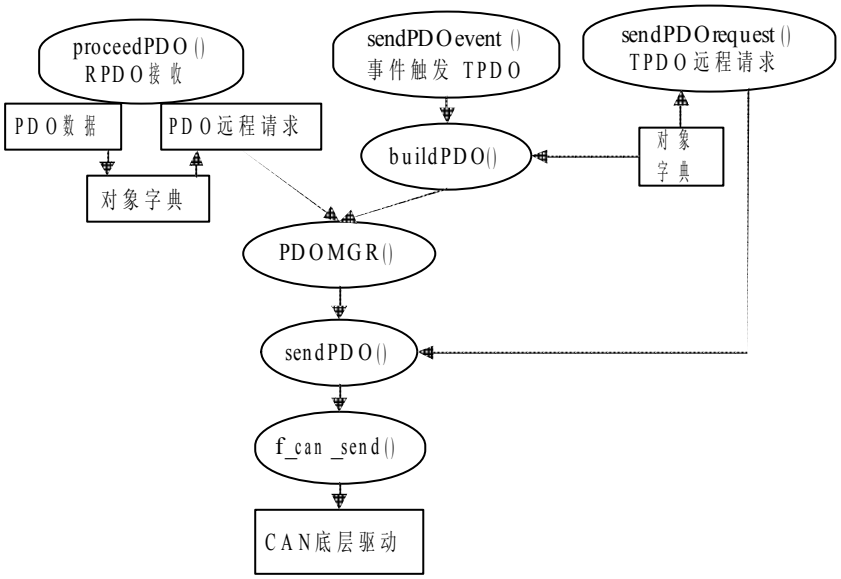

图 4.10 PDO 通信模块

同样地，当 TPDO 报文发送的是请求时，调用 sendPDOrequest()函数，从对象字典中读取 TPDO 相关索引，找到与待发送的 COB-ID 相同的 TPDO 项，而后调用 sendPDO()函数发送远程请求帧。若是发送具体的数据，则调用 buildPDO()函数，配置好 TPDO 相关的通信参数和映射参数后，再调用 sendPDO()函数发送数据，最后该函数调用底层驱动完成 PDO 报文发送。此外定义了 PDO 管理函数 PDOMGR()用来对 TPDO 报文和 RPDO 报文进行管理，若发送报文和接收报文发生冲突，则规定接收报文的优先级更高。基于事件触发的 PDO 报文发送函数流程如图 4.12 所示。

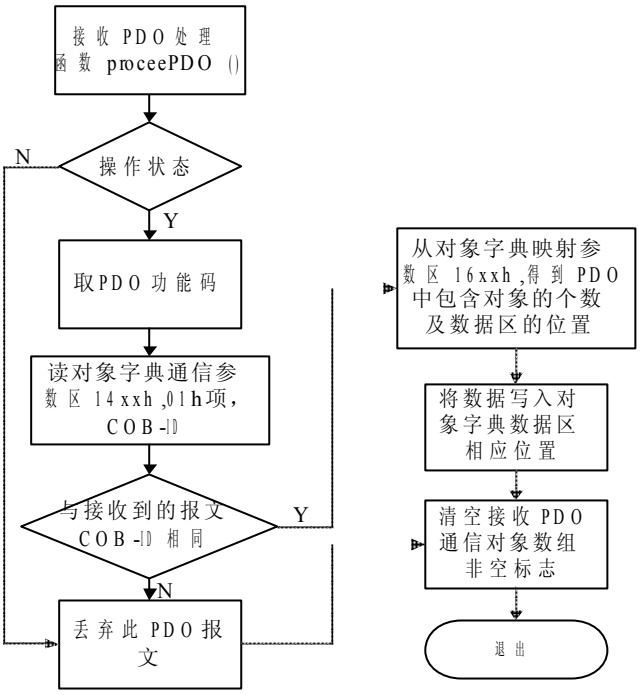

图 4.11 PDO 报文处理流程

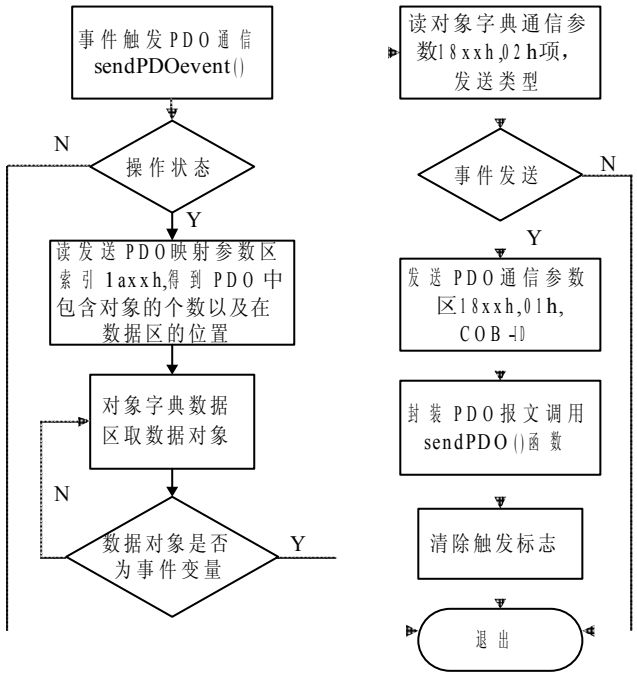

图 4.12 事件触发 PDO 通信流程

4.7.3 同步报文及紧急报文

同步报文与 PDO 报文结合共同构成了基于时间触发的 TPDO 报文发送。在对象字典中，同步对象使用三个索引定义同步报文的参数，0x1005 索引记录同步报文使用的 COB-ID，0x1006 索引为同步间隔，0x1007 索引为同步窗口宽度。

若要发送同步报文，通过 ComputeSYNC()函数，读取对象字典中相关的同步参数，若发现当前时间已超过同步间隔时间，则立即调用 sendSYNC()函数进行同步报文的发送。若接收到同步报文，则调用 proceedSYNC 同步报文处理函数，然后遍历每一个 TPDO 索引，若有相应的 TPDO 传输类型是使用同步报文触发，则针对该 TPDO 接收到的同步报文进行计数，若接收到的同步报文达到了触发 TPDO 发送的预定值时，则立即发送 TPDO 报文。同步报文的处理流程如图 4.13 所示

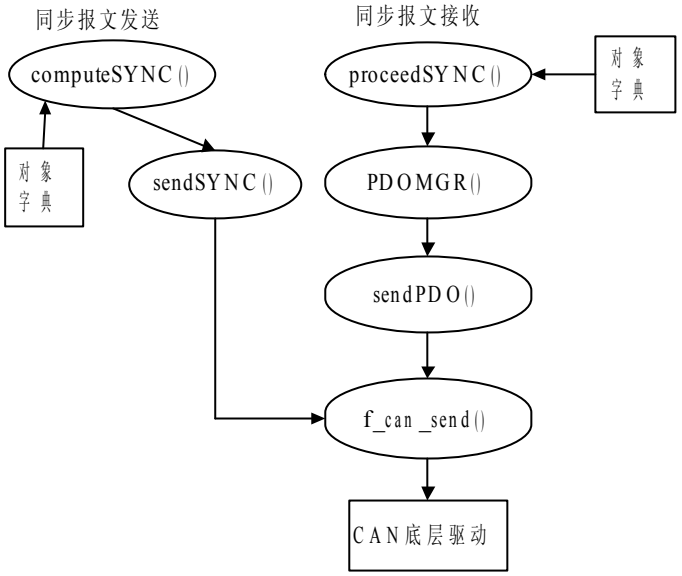

图 4.13 同步报文处理流程

在设备内部出现错误时，将触发紧急事件并通过高优先级的紧急报文来发送模块的状态。其具体的帧格式如表 4.2 所示，其中对象 1001h 是错误寄存器，对象 1002h 是模块的状态，这 32 位数据为制造商私有，由它自己指定数据的状态意义，而 1003h 则为预定义的错误区域，其中包含了 DS301 定义的紧急时间错误代码。

表 4.2 CANopen 紧急报文

COB-ID	Byte0-1	Byte2	Byte3-6	Byte7
0x80 +Node_ID	错误编码 （对象 0x1003h）	错误寄存器对象 （对象 0x1001h）	制造商自定义 错误区域（对象 1002h）	空

4.7.4 节点状态保护机制的实现

节点将通过 proceedNMTerror 函数检测接收到的是否是远程帧，若是远程帧，则将自身状态及跳变位 toggle 通过底层驱动进行发送，主节点通过该状态

报文监视从节点，实现节点保护功能。若非远程请求，则检测报文是否为Boot-up，若是此报文，则主节点根据 Boot-up 报文更新节点的 NMTable 状态表。若接收到的是心跳报文，那么重置节点的心跳计时，并通过 lifeguardCallback 函数来指示节点状态为连接良好，具体的流程如图 4.14 所示。

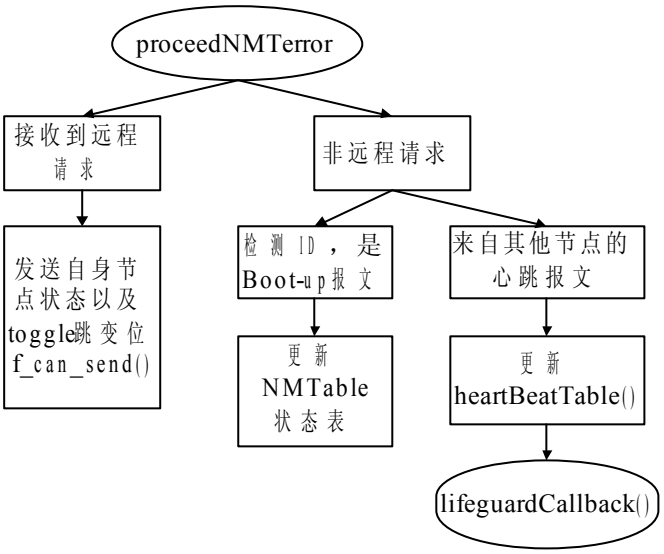

图 4.14 节点状态保护机制

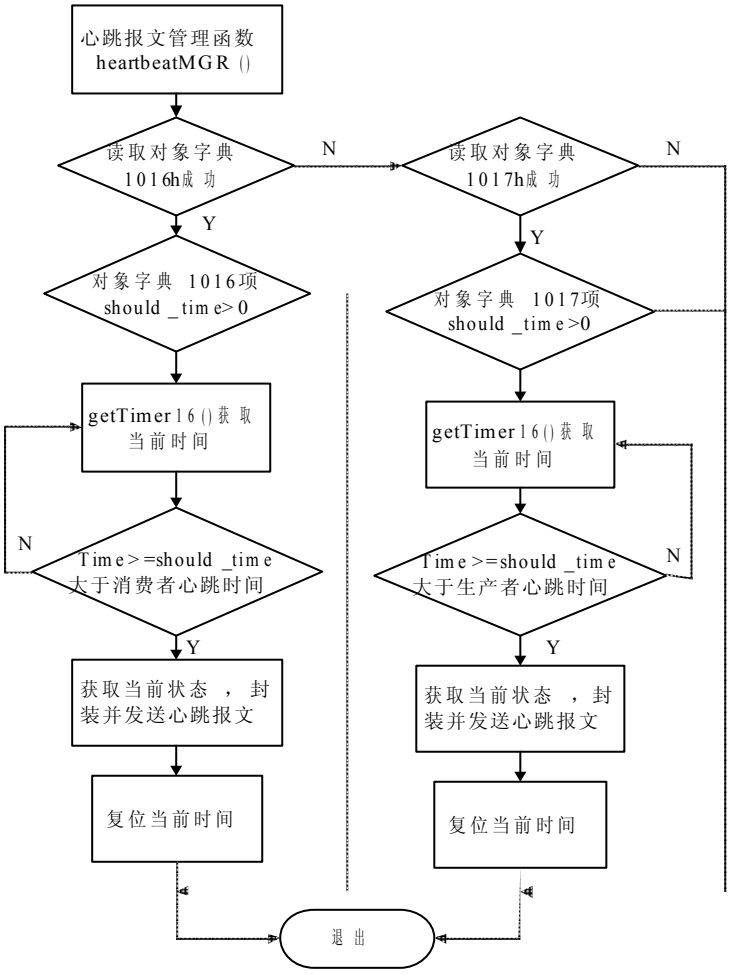

图 4.15 心跳报文管理

节点通过 heartbeatMGR()函数来检测自身是否在规定的时间内接收到心跳报文以及发送心跳报文，它通过读取对象字典中关于心跳协议的索引项 0x1016 和 0x1017 的数值，并将当前时间与之进行比较，若发生超时现象，则产生心跳事件或者通知心跳生产者清零计时并立即产生心跳报文，具体流程如图 4.15 所示。

4.8 本章小结

本章阐述 CANopen 节点的软件设计部分，根据数据流和协议栈分层结构对整个协议代码进行分层设计，包括硬件驱动层、μC/OS-Ⅱ系统层、CANopen 通信协议以及设备行规应用层；详细给出了 CANopen 软件整体架构、对象字典、四类通信对象的实现细节和相关的软件流程图。

第五章 系统调试及通信实验

5.1 通信实验目的

在以 MC9S12XF512 为核心的硬件平台上完成程序的编写和调试后，还需要进行通信测试来验证 CANopen 协议的各项功能。课题的目的在于开发符合 CANopen 协议通信规范 DS301 的通信节点，因此需要对协议所定义的四类通信对象的功能进行测试，包括有 CAN 节点的初始化、通信参数及映射参数的配置、状态切换机制的实现，CAN 数据传输，节点保护机制的实现等。

5.2 测试环境与网络架构

实验使用两个 MC9S12XF512 芯片构成的 CANopen 平台作为主从通信节点，在硬件的具体连接上，将各自的 CANL 和 CANH 引脚通过双绞线相互连接；同时利用单片机的串口通信功能将 CANopen 设备的运行状态通过 RS232 接口发送到 PC 机上，使用串口调试器软件实时地显示 CANopen 通信状态信息；并且使用 LCD 液晶显示屏将相关的 CAN 通信数据显示出来。设备使用 5V 的直流电源供电，在通信时，设计主节点的节点 ID 为 1，从节点的节点 ID 为 5，如图 5.1 所示。

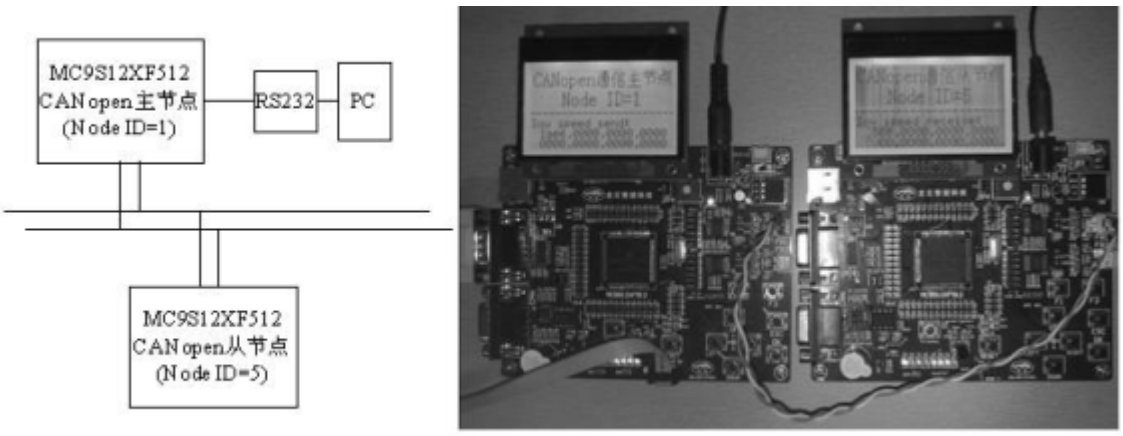

图 5.1 CANopen 通信实验组网结构图

5.3 CANopen 主从节点通信测试

5.3.1 通信数据和参数配置

在两节点的对象字典中定义了如表 5.1 所示的数字输入量，CANopen 设备将通过 PDO 和 SDO 来对这些索引下的具体数据进行通信。其中 seconds，minutes，hours，day 这四个全局变量是在一个时间函数中进行调用，该时间函数通过定时器 3 每秒产生一次中断来实现秒数、分钟、小时的递增。

表 5.1 对象字典通信参数配置

主节点(节点 ID 0x01)			从节点(节点 ID 0x05)		
主索引	子索引	数据	主索引	子索引	数据
0x6002	0x01	seconds	0x2000	0x01	seconds
0x6002	0x02	minutes	0x2000	0x02	minutes
0x6002	0x03	hours	0x2000	0x03	hours
0x6002	0x04	day	0x2000	0x04	day
0x6003	0x01	canopenErrNB	0x6000	0x00	canopenErrNB
0x6003	0x02	canopenErrVAL	0x6001	0x00	canopenErrVAL
0x6000	0x00	canopenErrNB_node5			
0x6000	0x00	canopenErrVAL_node5			

CANopen 的通信实验在主节点 0x01 和从节点 0x05 之间进行。主节点的初始化的内容包括针对 CAN 控制器底层的初始化操作、所有全局数据对象、时钟、对象字典空间。主节点完成初始化工作后将其工作状态设置为操作状态，并在这一状态下进行主从通信的网络配置。

在网络配置中，首先进行 SDO 参数的配置，将从节点 ID 0x05 加载到主节点的对象字典 SDO 索引项中。然后进行 PDO 通信参数和映射参数的配置，采用如下代码将数字输入变量 seconds，hours，days 由从节点的 TPDO1 映射到主节点的 RPDO1

```
/*配置 RPDO1，定义在索引 0x1400 和 0x1600 */
{
s_mappedVar tabMappedVar[8] ={{0x6002,4,8}, {0x6002,3,8}, {0x6002,1,8}, };
masterMappingPDO(0x1400, 0x181, tabMappedVar, 3);
}
/*配置 TPDO1，定义在索引 0x1800 和 0x1A00 */
{
s_mappedVar tabMappedVar[8]={ {0x2000,4,8}, {0x2000,3,8}, {0x2000,1,8}, };
slaveMappingPDO(0x05, 0x1800, 0x181, tabMappedVar, 3);
}
```

同样地，将数字输入变量 minutes 由从节点的 TPDO2 映射到主节点的 RPDO2，实现 minutes 的值由从节点到主节点的 PDO 传输。

```
/* 配置 RPDO2, 定义索引在 0x1401 and 0x1601 */
{
s_mappedVar tabMappedVar[8] = { {0x6002,2,8} };
masterMappingPDO(0x1401, 0x282, tabMappedVar, 1);
}
/* 配置 TPDO2, 定义索引在 0x1801 and 0x1A01*/
{
s_mappedVar tabMappedVar[8] = { {0x2000,2,8} };
slaveMappingPDO(0x05, 0x1801, 0x282, tabMappedVar, 1);
}
```

在进行用于节点保护的心跳协议设置时，设置主从节点的心跳生产者时间为 1000ms，即每 1000ms 即发送一个心跳报文，而设置心跳消费者时间为 1500ms，即表示节点必须在每 1500ms 的时间之内接收到一个心跳报文。

在对 TPDO 的通信类型进行设置时，通过以下函数设置从节点的 TPDO1

为同步触发传输，具体将主节点每 1s 发送一个同步报文，从节点每接收到一个同步报文即传输一次 TPDO1 信息，设置从节点的 TPDO2 为事件触发传输。

slavePDOTransmissionMode(0x05, 0x1800, TRANS_EVERY_N_SYNC (1));

slavePDOTransmissionMode(0x05, 0x1801, TRANS_EVENT);

从节点的主函数即是一个状态调度机，在操作状态下，设置触发其 TPDO2 发送信息的事件为 minutes 全局变量的改变。也就是说，随着计时器的变化，当 minutes 值发生改变时，就触发从节点的 TPDO2 发送这一新的 minutes 值。

系统上电后，CANopen 主从节点按照上述的配置方式进行数据的通信，并且通过 RS232 串口实时地将节点的运行状态在 PC 机上运行的串口调试器软件中显示。主节点实现了每 1s 发送同步报文，并触发从节点通过 TPDO1 发送 seconds，hours，days 变量的值，并且当 minutes 变量更新时，此事件触发从节点通过 TPDO2 将数据发送到主节点。通过 RS232 将程序进行跟踪监测，可以实测到下面的报文数据在主从节点之间进行传输，以下是系统工作流程和实测到的报文数据及相关功能分析。

5.3.2 系统工作流程

主节点按照如图 5.2 所示的流程启动初始化并进行通信参数的配置，CANopen 主节点启动后，首先发送 NMT 命令报文(81 00)对包括自身在内的所有节点进行复位操作。随后发送远程请求命令搜寻特定 ID 的从节点，如果有从节点对此远程帧进行回应，表明网络上存在预期 ID 的从节点，此时主站发送 SDO 写命令对从节点 SDO 参数进行设置，然后发送 NMT 网络管理命令(80 00)使得所有节点进入预操作状态，此时对上述的通信任务进行设置，包括 TPDO 和 RPDO 设置、PDO 传输类型、心跳报文及节点保护等。节点确认后，发送 NMT 管理命令(01 00)，使得所有节点都进入操作状态，此时便可以按照预定的通信任务进行 PDO 数据交换。

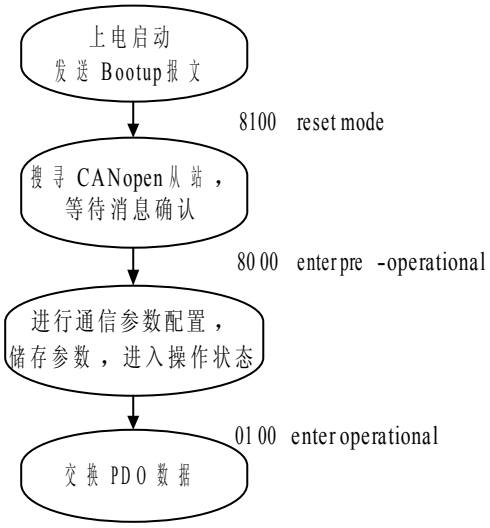

图 5.2 系统工作流程

5.3.3 CANopen 通信报文分析

通信对象	帧 ID	帧格式	帧类型	数据长度	CAN 数据	报文分析
NMT	00000701	数据帧	标准	0x01	00	主站上电启动
NMT	00000000	数据帧	标准	0x02	81 00	复位所有节点
NMT	00000705	数据帧	标准	0x01	00	从节点 5 发送启动报文
NMT	00000000	数据帧	标准	0x02	80 00	主节点发送命令,所有节点进入预操作状态
NMT	00000705	数据帧	标准	0x01	7f	节点 5 进入预操作状态
SDO	00000605	数据帧	标准	0x08	40 00 10 00 00 00 00 00	检查从节点的设备类型等参数
SDO	00000585	数据帧	标准	0x08	43 00 10 00 91 01 07 00	回复索引 0x1000 内容
SDO	00000605	数据帧	标准	0x08	40 18 10 02 00 00 00 00	读取索引 0x1018
SDO	00000585	数据帧	标准	0x08	43 18 10 02 78 56 00 00	回复
SDO	00000000	数据帧	标准	0x02	81 05	重置节点 5
SDO	00000605	数据帧	标准	0x08	2b 17 10 00 e8 03 00 00	配置心跳报文索引 0x1017
SDO	00000585	数据帧	标准	0x08	60 17 10 00 00 00 00 00	回复
SDO	00000605	数据帧	标准	0x08	2b 16 10 01 dc 05 00 00	配置心跳报文索引 0x1016
SDO	00000585	数据帧	标准	0x08	60 16 10 01 00 00 0000	回复
heartbeat	00000705	数据帧	标准	0x01	7f	心跳报文
heartbeat	00000705	数据帧	标准	0x01	7f	心跳报文
heartbeat	00000705	数据帧	标准	0x01	05	心跳报文
heartbeat	00000705	数据帧	标准	0x01	05	心跳报文
SDO	00000605	数据帧	标准	0x08	2f 00 18 00 02 00 00 00	写节点 5 的 0x1800+0x00 项
SDO	00000585	数据帧	标准	0x08	60 00 18 00 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	23 00 18 01 00 00 81 01	写节点 5 的 0x1800+0x01 项
SDO	00000585	数据帧	标准	0x08	60 00 18 01 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	2f 00 18 02 01 00 00 00	写节点 5 的 0x1800+0x02 项
SDO	00000585	数据帧	标准	0x08	60 00 18 02 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	2f 00 1a 00 03 00 00 00	设置 TPDO1 中的数据数目
SDO	00000585	数据帧	标准	0x08	60 00 1a 00 00 00 00 00	回复
SDO	00000605	数据帧	标准	0x08	23 00 1a 01 08 04 00 20	映射对象 0x2000+04 项目
SDO	00000585	数据帧	标准	0x08	60 00 1a 01 00 00 00 00	回复
SDO	00000605	数据帧	标准	0x08	23 00 1a 02 08 03 00 20	映射对象 0x2000+03 项目
SDO	00000585	数据帧	标准	0x08	60 00 1a 02 00 00 00 00	回复

SDO	00000605	数据帧	标准	0x08	23 00 1a 03 08 01 00 20	映射对象 0x2000+01 项目
SDO	00000585	数据帧	标准	0x08	60 00 1a 03 00 00 00 00	回复
SDO	00000605	数据帧	标准	0x08	2f 01 18 00 02 00 00 00	写节点 5 的 0x1801+0x00 项
SDO	00000585	数据帧	标准	0x08	60 01 18 00 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	23 01 18 01 00 00 82 02	写节点 5 的 0x1801+0x01 项
SDO	00000585	数据帧	标准	0x08	60 01 18 01 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	2f 01 18 02 FF 00 00 00	写节点 5 的 0x1801+0x02 项
SDO	00000585	数据帧	标准	0x08	60 01 18 02 00 00 00 00	从节点 5 回应主站写操作
SDO	00000605	数据帧	标准	0x08	2f 01 1a 00 01 00 00 00	设置 TPDO2 中的数据数目
SDO	00000585	数据帧	标准	0x08	60 01 1a 00 00 00 00 00	回复
SDO	00000605	数据帧	标准	0x08	23 01 1a 01 08 02 00 20	映射对象 0x2000+02 项目
SDO	00000585	数据帧	标准	0x08	60 01 1a 01 00 00 00 00	回复
SYNC	00000080	数据帧	标准	0x00	00	同步报文
SYNC	00000080	数据帧	标准	0x00	00	同步报文
SYNC	00000080	数据帧	标准	0x00	00	同步报文
PDO	00000181	数据帧	标准	0x03	00 00 05	TPDO1 报文
PDO	00000181	数据帧	标准	0x03	00 01 08	TPDO1 报文
PDO	00000282	数据帧	标准	0x01	06	TPDO2 报文

图 5.3 CANopen 通信报文数据分析

如图 5.3 所示，给出了 CANopen 通信的部分报文，并进行了报文分析。课题设计的 CANopen 节点成功地进行 CAN 数据交换，通过对 CAN 数据帧内容的分析表明，所有节点完成通信初始化和节点初始化，并发送 bootup 消息后进入预操作状态；根据需要可以发送 heartbeat 消息，实现节点保护；在操作状态下，能够完成同步传输和事件触发下的异步 PDO 传输；整个通信过程实现了 CANopen 四类通信对象的基本功能，满足了 CANopen 协议通信规范 DS301 的要求。

5.4 本章小结

本章首先阐明了通信实验的目的和内容，而后构建测试环境和网络架构。通过对报文数据的分析，验证了 CANopen 通信的有效性和可靠性。

第六章 结束语

6.1 全文总结

CANopen 协议在 CAN 总线的基础上，采用灵活多变的传输方式，具有强大的网络管理能力，并且将实时数据和服务数据分开进行传输，提高了实时数据的传输效率，因此在各种分布式工业自动化控制系统中得到广泛地认可和大量的应用。研发基于 CANopen 协议的现场总线通信设备，能够促进国内业界对 CANopen 技术的关注，对于促进工控自动化领域自主知识产权产品的研究和发展，有着积极的意义。本文剖析了 CAN 总线协议及其应用层协议 CANopen 的工作原理，并在此基础上完成了基于 MC9S12XF512 芯片和嵌入式系统 μC/OS-Ⅱ的 CANopen 通信节点的设计，主要工作内容如下：

(1) CAN 总线通信机制的研究。分析 CAN 协议物理层和数据链路层协议的具体功能和工作特点，在此基础上对单片机上的 CAN 控制器进行研究，仔细分析了 MC9S12XF512 内置的 MSCAN 控制器的特点、寄存器配置及相对应的 CAN 通信机制，在 TJA1040 收发器和 HCPL-2630 光耦合器组成的 CAN 硬件接口的基础上，完成了 CAN 底层收发接口函数的编写，并进行了基本的 CAN 节点通信实验，从而掌握 CAN 总线硬件实现原理，理解了 CAN 协议物理层和数据链路层的工作机制。

(2) μC/OS-Ⅱ系统原理的研究和应用。通过对μC/OS-Ⅱ系统核心源代码的分析，掌握μC/OS-Ⅱ的内核结构、任务管理、中断和时钟操作。通过对μC/OS-Ⅱ系统中与移植相关代码进行修改，成功将μC/OS-Ⅱ系统移植到 MC9S12XF512 芯片中运行，并进行任务调度实验。

(3) CANopen 协议的研究和应用。在 FreeScale 提供的 CodeWarrior 集成编译环境下，将 CANopen 从站内核嵌入到μC/OS-Ⅱ系统中，建立相关 CANopen 任务运行；利用 C 语言实现 CANopen 协议，并在前期底层通信实验的基础上，在 MC9S12XF512 芯片上实现 CANopen 协议通信规范 CiA DS301 各个模块的功能；其中对象字典依据索引类型进行分块定义，提高了读写对象字典时对索引的搜索效率，实现了通信对象与上层应用程序的数据共享；依据 CANopen 预定义连接集设计 PDO 和 SDO 模块，设计实现 PDO 通信参数的配置和实时数据的映射。

(4) CANopen 主从节点通信。搭建两节点的 CANopen 通信测试平台，对 CANopen 协议的通信功能进行测试，设置好相关的通信参数和数据，验证了 heartbeat 报文收发、PDO 和 SDO 通信、NMT 消息的发送和响应。

6.2 工作展望

由于时间和客观条件等因素的限制，本课题所设计的 CANopen 通信节点只

实现了基本的通信功能，距离真正在工控自动化场合应用的 CANopen 通信产品还有这一定的差距，还需要从以下几个方面进行改进和完善，并进一步通过实践检验。

(1) 为构建更完善的 CANopen 网络通信实验平台，可以使用先进的 CANopen 网络仿真和监控工具。比如 Vector 公司的 CANoe.CANopen7.2 平台，从而可以通过使用该工具方便地制定 PDO、SDO 测试用例、提高测试深度及测试质量，并且更快地分析 CAN 报文。

(2) 课题目前所进行的通信实验，只是基于 MC9S12XF512 和μC/OS-Ⅱ的 CANopen 节点，为确定 CANopen 节点所实现的功能，下一步可以将 CANopen 节点连接标准的 CANopen 设备，并与不同厂家生产的 CANopen 主从设备进行通信，从而完成一致性测试。

(3) 目前编写的 CANopen 代码只实现了 CANopen 通信规范 CiA DS301 规范的功能，因此还可以基于应用需求来完善一系列的通信规范的功能，包括 CiA302 所定义的基于 CiA301 通信机制的重要扩展，使其支持智能化的设备；CiA305 定义的 LSS 功能，用来远程配置节点 ID 和总线波特率；CiA306 定义的配置网络和设备时使用的电子数据表单等。

参考文献

[1] 饶运涛,邹继军,王进宏.现场总线 CAN 原理与应用技术[M].北京：北京航空航天大学出版社，2007:7-10.

[2] CAN in Automation Official Web Site URL:http://www.can-cia.org/.

[3] M. Farsi, K. Ratcliff, and M. Barbosa, “An Overview of Controller Area Network”[J]. Computing & Control Engineering Journal, Vol. 10, June 1999:113-120.

[4] ISO11898, Road Vehicles-CAN area network (CAN)-Part 1 Data link layer and physical signaling[S].

[5] 孙铁兵,鞠宁.CAN 总线及其高层协议[J].微处理机,2006.2(1):24-26.

[6] Lars-Berno Fredriksson, “CAN for Critical Embedded automotive networks” [J]. Micro, IEEE, Aug 2002 Volume: 22 Issue:4: 28-35.

[7] 夏继强,李晓君,曹磊,等.SAEJ1939 协议栈设计及 μC/OS-Ⅱ系统下的开发平台研究[J].汽车工程,2008(Vol.30)No.12:1069-1074.

[8] 陈在平,王峰.基于 CANopen 协议从节点研究[J]制造业自动化，2010 (Vol 32) No.2:27-30.

[9] 王宇波.嵌入式网络控制器的设计与实现[D].硕士学位论文.天津大学.2005

[10] 陈涛.汽车仪表的 CANopen 节点通信的研究与实现[D].硕士学位论文.北京工业大学硕士学位论文.2007.

[11] 肖进军.混合动力电动汽车 CANopen 总线协议的研究与实现[D].硕士学位论文.北京工业大学.2007.

[12] 任哲. 嵌入式实时操作系统 μC/OS-Ⅱ原理及应用[M].北京：北京航空航天大学出版社，2005.

[13] 叶浩峰.CANopen 总线的原理以及实现[D].硕士学位论文.华南理工大学.2005.

[14] Robert Bosch GmbH, “BOSCH CAN Specification 2.0,” Robert Bosch GmbH, Stuttgart Germany, 1991.

[15] 孙同景,陈桂友.Freescale 9S12 十六位单片机原理与嵌入式开发技术[M].北京：机械工业出版社,2008:389-391.

[16] Freescale Semiconductor Inc. MC9S12XF512 Reference Manual Rev.1.19. http://www.freescale.com.cn, 2010-05-18.

[17] CAN-in-Automation, CANopen, CAL-based Communication Profile for Industrial Systems, CiA DS-301, Version 4.0, June 16 1999.

[18] Philips Semiconductors, TJA1040 High speed CAN transceiver datasheet, Oct 14 2003.

[19] Labrosse J J. 嵌入式实时操作系统 μC/OS-Ⅱ[M]. 第二版.邵贝贝译. 北京：北京航空航天大学出版社，2003:278-283.

[20] 赵峰.CANopen 协议研究及一体化适配器开发[D].硕士学位论文.北京化工大学.2009.
[21] 潘倩姿.CAN 总线及其协议的分析研究和应用[D].硕士学位论文.南京邮电大学.2008.
[22] 陈涛.汽车仪表的 CANopen 节点通信的研究与实现[D].硕士学位论文.北京工业大学.2007.
[23] 董石峰.混合动力电动汽车车载网络 CANopen 协议及其应用研究[D].硕士学位论文.北京工业大学.2010.
[24] 宋威.CANopen 现场总线应用层协议主站的开发与实现[D].硕士学位论文.北京工业大学.2008.
[25] 孔峰，张衡，宋雪桦，等.基于 CANopen 协议的汽车控制网络初探[J].汽车工程，2007.29(7):594-596.
[26] Jianmin Duan,Jinjun xiao,Mingjie Zhang,“Framework of CANopen Protocol for a Hybrid Electric Vehicle” [C]Proceedings of the 2007 IEEE Intelligent Vehicles Symposium ,June 13-15 , 2007:906-911.
[27] Xu Zhe, Dong Shifeng, “The Design and Implementation of a CANopen Slave Stack for Powertrain Controller in Hybrid Electric Vehicle”[C] International Conference on Intelligent Computation Technology and Automation, May 11-12, 2010:755-758.
[28] Wang Bao-qiang, Wang Zhi-rou,Yang Liang-liang, “Design of EmbeddedMeteorological Data Acquisition System Based on CANopen” [C]International Conference on Electrical and Control Engineering, June 25- 27,2010:732-735.
[29] M. Farsi, K. Ratchiff, and M. Barbosa, "An Introduction to CANopen"[J]. Computing & Control Engineering Journal, Vol.10, No.4, June 1999:161-168.
[30] Minkoo Kang, Kiejin Park and Bongjun Kim. “PDO Packing Mechanism for Minimizing CANopen Network Utilization” [C] 34th Annual Conference of IEEE on Industrial Electronics IECON 2008, 10-13 Nov. 2008 : 1516 – 1519.
[31] 邵贝贝,宫辉.嵌入式系统中的双核技术[M].北京：北京航天航空大学出版社,2008:205-207.
[32] 姚念龙,尹航,姜久春.μC/OS-Ⅱ在 MC9S12A64 上的移植与应用[J].微计算机信息,2006.12-2:53-56.
[33] 徐喆,闫士珍,宋威等，基于 MC9S12DP512 和 μC/OS-Ⅱ的 CANopen 主站开发[J].计算机工程与科学,2009.31(5)：118-125.
[34] 李澄,赵辉,聂保钱.基于 CANopen 协议实现多电机系统实时控制[J].微电机,2009.42(9):53-56.
[35] 梁涛,张江涛,孔鹤旭,等.CANopen 总线智能多路温度采集从站的研制[J].仪表技术与传感器,2009(第 10 期) No.10:61-65.
[36] 吴红星,柴凤,程树康.基于 CAN 总线的混合动力汽车通讯系统工程应用[J].微电机,2007(Vol 40) No.12:68-71.

[37] CANopen device profile for digital I/O modules, Draft Standard DS-401 Revision 1.0, CAN in Automation Group, October 1996.

[38] 孙树文,杨建武,张慧慧，等.基于 CANopen 协议的分布式控制系统 I/O 从站设计[J]. 计算机测量与控制，2007.15(12):1705-1707.

[39] Mansoor Mahfoud, Subra Ganesan,“Potentials of Parallel Processing on CAN Networks” [C] SAE 2006 World Congress & Exhibition, April 2006: Session: In-Vehicle Networks (Part 1 of 3) 1-10.

[40] Aibing Ye, Huayao Zheng, “The Design and Implementation of CANopen Protocol Intelligent Analysis System” [C] WiCom '09. 5th International Conference on Wireless Communications, Networking and Mobile Computing, 24-26 Sept. 2009:1-4.

[41] Gianluca Cena,Ivan Cibrario Bertolotti,Adriano Valenzano, “Modelling CANopen Communications According to the Socket Paradigm” [C] 10th IEEE Conference on Emerging Technologies and Factory Automation,19-22 Sept. 2005:791-798.

[42] Cyrilla Jane Menon, “A Single Network Solution for Safety-Related Applications Using CANopen” [C] 31st Annual Conference of IEEE on Industrial Electronics Society, IECON, 6-10 Nov. 2005:433-438.

[43] Javier Portillo, Elisabet Estevez, Itziar Cabanes, et al, “CANopen Network for µcontroller-based Real Time Distributed Control Systems” [C] 32nd Annual Conference of IEEE on Industrial Electronics, IECON,November 2006:4644-4649.

攻读硕士学位期间发表的论文

1．Shixun Lai, et al. Study on the Implementation of CANopen Protocol for Hybrid Electric Vehicle [C]. International Conference on Computer Control and Automation (ICCCA 2011), Jeju Island, South Korea, May 1-3, 2011. (Accepted, to be indexed by EI)

致　谢

近三年的研究生生活即将结束，我也将以新的面貌步入工作岗位。在通信与信息系统实验室的学习时光让我收获的不仅仅是专业知识，更多的是研究问题处理问题的能力、对生活积极乐观的精神和对他人热情帮助的态度。

首先要感谢我的导师丁志中教授。古人云：师者，传道授业解惑者也。从丁老师身上，我真切地体会到了这句话的含义。他引领我进入了百家争鸣的学术领域，开启了我在科学研究上的旅程。丁老师渊博的知识、严谨的治学态度、以及在待人处事方面的道理使我们受益匪浅。在丁老师的谆谆教诲之下，我们才能够一次次的面对困难奋勇向前，面对挫折坚持不懈，面对胜利不骄不躁。在此，谨向导师表示最诚挚的感谢！

感谢我的同学程和生、张燚、崔龙成、郭恒飞、黄玉雷、时慧晶、徐彩臣，大家经常一起讨论、相互帮助，如兄弟姐妹般相处，给我的研究生生活带来丰富的色彩。感谢通信与信息处理实验室的所有同仁，和谐友爱的科研环境让我能将有限的精力完全投入学术活动中。

在这里，我还要特别感谢我的家人！感谢他们这么多年来一如既往的无私关怀和鼓励，使我得以顺利完成学业。

最后，我要向百忙之中抽出时间评审我论文的诸位老师表达我衷心的谢意！

作者：赖世勋

2011 年 3 月