摘 要

随着现代工业和科学技术的发展，现场总线技术在工业生产和科学研究等领域都获得了巨大的发展。本课题结合浙江中控科教仪器有限公司的 AE2000A 型过程控制实验系统，设计了基于 CANopen 通信的控制器，以现场总线 CANopen 协议为过程控制系统的网络环境，实现对多路参数的控制。根据系统需求设计嵌入式系统硬件、系统软件、CANopen 通讯、多参数控制任务等，并对设计结果进行验证。

硬件部分以飞思卡尔公司的 16 位控制器 MC9S12XDP512 为核心控制芯片，设计了 A/D、D/A、以及 CAN 通信模块，构成 CANopen 从节点嵌入式硬件系统，给出硬件的设计及调试过程。

在 CANopen 协议栈方面，提出了在 MC9S12XDP512 平台上实现 CANopen 网络从节点的方法，并将协议栈移植到 μC/OS-II 操作系统上，大大提高了 CANopen 协议栈的应用灵活性和可扩展性。

应用方面，介绍了基于 PID 算法的液位、压力、流量多参数控制系统，设计了实时操作系统下的液位、压力、流量的控制任务，并与 CANopen 通讯任务结合。最终完成对本课题设计的 CANopen 从节点的验证。

关键字 CANopen 协议；μC/OS-II 操作系统；多参数控制

ABSTRACT

As the modern industrial and scientific technological development, field bus in industrial production, scientific research and other fields have also been tremendous development. The issue uses Zhejiang Science and Education Instrument Co., Ltd. AE2000A-process control of the experimental system to develop the controller based on CANopen communication which is the network environment of process control system. Introduce how to design the embedded hardware system, software system, CANopen communications, multi-parameters control task according to system requirements, and test the system.

The hardware is based on Freescale controllers MC9S12XDP512 as the core chip, including A/D, D/A, as well as CAN communication module. Design the CANopen slave node embedded system. This article introduces how to design and debug the hardware. The key principle of hardware is also given.

The system software design refers to a method to realize a slave node based on the MC9SX12DP512 platform. And port the code on μC/OS-II operation system. This new method greatly reducing the development cycle and increase the flexility of the code.

The application is focused on the liquid level PID control system. This article introduced how to design liquid level, pressure, and flow PID control system task in real-time operation system and how to combine it with CANopen communication task.At last test the whole system.

KEY WORDS: CANopen Protocol; μ C/OS-II Operation System; Multi-parameters Control

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京工业大学或其它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了谢意。

签名：张卓　　日期：2008.5.30

关于论文使用授权的说明

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：张卓　　导师签名：徐子珏　　日期：2008.5.30

第 1 章 绪论

1.1 研究的背景

现场总线在控制系统中的应用使各领域的综合自动化控制技术飞速发展，成为控制领域的一个新热点。CAN(Controller Area Network)总线是现场总线领域中一种很有前途的技术，具有可靠性高、抗干扰强、开发简单、造价低廉以及短帧传输和无破坏仲裁技术的优点。它是一种多主方式的串行数据通讯总线，网络中没有节点的地址信息，而是通过标识符来识别不同的报文，采用非破坏性总线仲裁技术。

20世纪80年代初期，由于欧洲汽车工业发展的需要，最先由德国Bosch公司提出CAN总线方案以解决汽车控制装置间的通信问题。为促进CAN的发展，1992年在欧洲成立了国际用户和厂商协会（CAN in Automation，简称CiA）。CAN本身并非一个完整的协议，只包括物理层和数据链路层两个底层协议，没有定义应用层，这为其应用提供了较大的灵活性，也成为它的缺点。要进行高效率的通讯还要进一步开发高层协议。为了满足不同的应用领域和不同用户的需求，现已开发出了很多基于CAN的协议，目前已发展成为工业标准的高层协议有：CAL/CANopen、DeviceNet等。CANopen由CiA(CAN in Automation)的会员开发，是CAL(CAN in Application)的一个子集，广泛用于机械制造、车辆、铁路、船舶、制药和食品加工等工业控制领域，DeviceNet是由美国Wockwell的A-B公司开发的开放式网络标准，属于设备级网络总线，包括传感器和执行机构，现已成为IEC标准(IEC62026)，成为CAN高层协议的标准之一[1]。

本课题主要是针对CANopen协议栈的开发应用。CANopen协议栈的开发可以有以下三种方案[2]，本课题采用方案2。

方案 1：在 CANopen-Chip 基础上开发 CANopen 节点。

方案 2：通过对 CANopen 协议栈的二次开发，在单片机上实现嵌入式 CANopen 节点。

方案 3：利用 CANopen Master API 在 PC 机上实现 CANopen 节点。

1.2 课题相关技术研究现状

1.2.1 CAN 总线

CAN总线全称是“控制器局域网(Controller Area Network)”，是一种串行通信协议，能够有效支持分布式实时控制，支持高速网络和低成本的多线制网络，具有可靠性高、应用广泛的优点。高速CAN应用于汽车电子、发动机控制单元、传感器、防滑系统等，通信速率可达1Mbit/s，低速CAN可应用于车灯控制、电动门窗等车身控制，可以取代接线配线装置。根据ISO/OSI参考模型，如图1-1所示，CAN被细分为不同的层次[5][33]。

(1) 数据链路层

包括逻辑链路控制子层和介质访问控制子层。在CAN2.0A规范中，数据链路层的LLC子层和MAC子层的服务及功能分别被解释为“目标层”和“传输层”。逻辑链路子层作用范围包括：为远程数据请求以及数据传输提供服务；确定LLC子层接受的报文中那些报文实际上被验收；为恢复管理和过载通知提供手段。介质访问控制子层MAC的作用主要是传送的规则，也就是控制帧的结构、执行仲裁、错误检测、错误的标定、故障的界定。总线上什么时候开始发送报文及什么时候开始接受报文，均在MAC字层里确定。

(2) 物理层

物理层的作用是在不同节点之间根据所有的电气属性进行位的实际传输。

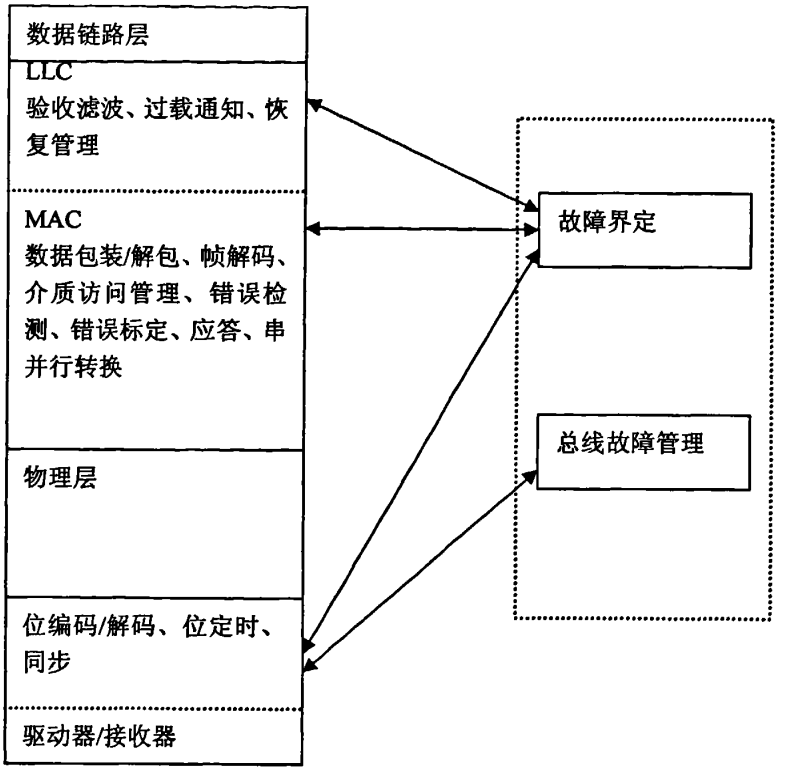


图 1-1 CAN 的 ISO/OSI 参考模型的层结构
Fig1-1 ISO/OSI Layer Architecture of CAN

CAN 报文有两种不同的帧格式，不同之处为标识符域的长度不同，含有 11 位标识符的帧称为标准帧，含有 29 位标识符的帧称为扩展帧[8]。ID 是 CAN 报文的标识符，11 位/29 位。RTR 如果值为 0，代表该报文是普通报文，包含数据位。 RTR 如果值为 1，代表报文是远程请求报文，不包含数据。DN 是数据位。通常一个 CAN 报文包括 0 到 8 个数据字节。

由于CAN总线采用了许多新技术及独特设计，CAN总线的数据通信具有突出的可靠性、实时性和灵活性。已成为国际标准化组织ISO11898标准，具有如下特性[6][7]：

- CAN 采用多主方式工作，网络上任意节点均可以在任意时刻主动地向网络上的其它节点发送信息，而不分主从，通讯方式灵活。
- CAN 信息帧采用短帧结构，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。
- CAN 协议废除了传统的站地址编码，而是对通讯数据块进行编码，使网络中的节点个数在理论上不受限制，网络中的不同节点同时接到相同的数据，使总线上传输的信息总量减少。
- CAN 网络上的信息可分成不同的优先级，满足不同的实时性要求。

● CAN 采用非破坏性总线裁决技术(CSMA/CD)，大大节省了总线冲突裁决时间；最重要的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）。

● CAN 网络具有点对点、一点对多点和全局广播等几种通讯方式；具有极好的检错效果，CAN 的每帧信息都具有 CRC 校验和其它检错措施，保证了错误的输出率极低。

● CAN 的直接通讯距离最远可达 10km（速率 5kbps 以下）；通讯速率最高可达 1Mbps（此时距离长 40m）；最多可接节点达 110 个。

由于具有以上特性，CAN 总线越来越受到人们的重视，目前已有许多大公司的产品采用了这一技术。CAN 总线的应用范围很广，其中汽车和交通领域占应用总数的 80%以上，在工业控制、楼宇自动化、机器人领域、装载器械、嵌入式网络、混合引擎控制、和医学电子、电话系统等行业都有着成功的应用。目前，支持 CAN 协议的有 Intel、Motorola、Philips、Siemens、NEC、Silioni、Honeywell 等百余家国际著名大公司。

1.2.2 CANopen 协议

从 OSI 网络模型的角度来看，CAN 总线仅仅定义了第 1 层（物理层）、第 2 层（数据链路层）。实际设计中，这两层完全由硬件实现，无需再为此开发相关软硬件。同时，CAN 只定义物理层和数据链路层，没有规定应用层，本身并不完整，需要一个高层协议来定义 CAN 报文中的 11/29 位标识符、8 字节数据的使用。CANopen 协议预研阶段是 1995 年在以 Bosch 公司为首的团队下开发的，后来成为 CiA(CAN in Automation)定义的标准之一，它是基于 CAN 总线的高层协议，是一个开放的、标准化的高层协议；这个协议支持各种 CAN 厂商设备的互用互换性，能够实现在 CAN 网络中提供标准的、统一的系统通讯模式，提供设备功能描述方式，执行网络管理功能。在 OSI 模型中，CAN 标准、CANopen 协议之间的关系如图 1-2 所示[9][10][29][32]。

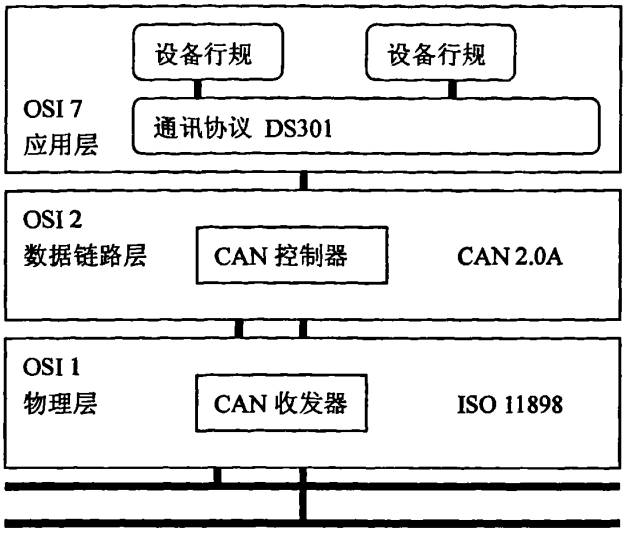


图 1-2 CANopen 在 ISO/OSI 层结构中的位置

Fig1-2 The Position of CANopen in ISO/OSI Layer Architecture

CANopen 协议详细规定了通讯模式、网络管理以及相关参数的设定和不同领域应用的典型模式等内容。CANopen 通讯模型定义了 4 种报文[11][12][30]：

管理报文(NMT)、过程数据对象(PDO)、服务数据对象(SDO)、预定义报文或特殊功能对象。管理报文(NMT)作用是管理网络和 ID 分配，例如初始化网络，配置和网络管理。过程数据对象(PDO)对小型的数据进行高速数据交换，通常采用事件触发、循环或请求方式发送，一个 PDO 最大可传输 8 字节数据。服务数据对象(SDO)主要用于在设备配置过程中传输参数以及传输大数据块。SDO 传送报文可以不受长度的限制。预定义报文或特殊功能对象主要包括同步报文(SYNC)、时间戳对象(Time Stamp)、紧急事件报文(Emergency)、节点保护报文(Node Guarding)。

CANopen最核心的部分是对象字典(Object Dictionary)。简单地说，对象字典就是一张表，承载着整个网络所具备的数据并描述CANopen的配置和节点功能。CANopen网络中，通信对象和应用对象之间，以对象字典为接口，应用程序通过读写对象字典，就可以实现CANopen通信。对象字典是一个有序的表格；每个对象采用一个16位的主索引值来寻址，共有65,535个，每个主索引最多可以有256个子索引。子索引是一个8位的数据，是为了允许访问数据结构中的单个元素而定义的，每个主索引至少包含一个子索引[13]。对象字典的结构参照表1-1。

表 1-1 对象字典结构

Table1-1 The Structure of Object Dictionary

索引	对象
0000h	保留
0001h-001Fh	基本数据类型
0020h-003Fh	复杂数据类型
0040h-005Fh	生产商相关数据类型
0060h-007Fh	设备描述的基本数据类型
0080h-009Fh	设备描述的复杂数据类型
00A0h-0FFFh	保留
1000h-1FFFh	通讯参数
2000h-5FFFh	制造商的特殊设备描述文件
6000h-9FFFh	标准设备描述文件
A000h-BFFFh	标准接口描述文件
C000h-FFFFh	保留

1.2.3 CANopen 协议栈介绍

CANopen 协议的说明是可以免费下载的，要根据协议自行开发 CANopen 协议栈，大约需要 12 个月甚至更长时间来达到适合嵌入式应用的比较稳定的结果。现在市场上的 CANopen 协议栈软件大约要 7 万至 10 万人民币，商用 CANopen 协议栈代码价格昂贵，若购买现成的协议栈，不但会增加开发成本，还会增加开发难度。因此进行 CANopen 协议的开发，首先想到的是如何利用现有的开源代码。在开源代码方面，现在能够找到的有两个组织的 CANopen 协议栈可供使用。

一个是由美国的嵌入式系统研究会（Embedded Systems Academy）支持的微型 CANopen 项目（Micro Canopen）[3]。该项目提供的开源 CANopen 协议栈并不支持所有的 CANopen 功能，仅作为学习目的公开。它的体积很小，可以在 8051 位控制器上实现，并只需要 4K 字节的代码空间和 170 字节的数据空间。

另一个是由法国的 Loli Tech 资助的 CANfestival 项目[4]。与微型 CANopen 项目不同的是，该项目是一个由开源社区 LGPL 授权的较为完整的 CANopen 协议栈，需要 40K 字节以上的代码空间。

1.2.4 嵌入式系统

由于硬件的限制，在使用MCU设计嵌入式系统的初期，程序设计人员得到的是只有硬件系统的“裸机”，没有任何类似于操作系统的软件作为开发平台，对CPU,RAM等这些硬件资源的管理工作都必须由程序员自己编写程序来解决，从而使程序员工作的十分辛苦，并使程序的开发效率菜极低。现在由于技术进步，单片系统硬件的规模越来越大，功能越来越强，从而给运行嵌入式操作系统提供保证。运行在嵌入式硬件平台上，对整个系统及其所操作的部件、装置等资源进行统一协调、指挥和控制的系统软件就叫做嵌入式操作系统。

嵌入式操作系统按应用范围划分，可分为通用型嵌入式操作系统和专用型嵌入式操作系统。通用型可应用于多种应用环境，例如常见的 Windows CE、vXWorks、μCLinux 及μC/OS 等；专用型嵌入式操作系统则用于一些特定的领域，例如移动电话的Symbian、手持数字设备（PDA）的Palm OS 等[14]。

1.3 课题的任务

1、分析CANopen高层协议，本课题主要实现CANopen从节点的基本功能。

2、CANopen 从节点模块的硬件设计与实现。本课题采用飞思卡尔公司的MC9S12XDP512作为CPU，CAN总线通讯收发芯片由TJA1050、配合两个高速光耦来实现，用来完成数据通讯和控制任务，从节点模块还包括D/A转换、A/D转换、LED、LCD接口、键盘接口、Flexray控制器、USB、以太网芯片、继电器模块等，便于扩展应用。

3、 实现基于实时操作系统μC/OS-II的CANopen网络从节点，完成CAN总线数据通讯、网络管理和相应的控制任务。

4、探索CANopen实时数据传输的改进，增强CAN总线的实时性，进一步增强工业控制网络的控制效果。

1.4 本章小结

本章介绍了课题的研究背景及相关技术的研究现状，详细阐述了CAN网络的分层结构、CANopen协议的通信对象的定义、CANopen协议栈的分类以及嵌入式系统的种类，最后介绍了本课题的研究任务。

第 2 章 从站代码的设计

2.1 CANopen 协议中的 CAN 报文

CANopen 报文以 CAN 标准报文为依据，11 位标识符中的前 4 位是报文的功能码，后 7 位是节点号，共同组成 COB-ID 以区分不同节点的不同报文。不同通信对象具有不同的功能码[31]，按优先级从高到低排列，如表 2-1 所示。

表 2-1 功能码定义

Table2-1 The Definition of Functionn Code

报文类型	功能码	COB-ID 范围
NMT	0000	000h
SYNC	0001	080h
EMERGENCY	0001	081h-0FFh
TIME	0010	100h
PDO1（发送）	0011	181h-1FFh
PDO1（接收）	0100	201h-27Fh
PDO2（发送）	0101	281h-2FFh
PDO2（接收）	0110	301h-37Fh
PDO3（发送）	0111	381h-3FFh
PDO3（接收）	1000	401h-47Fh
PDO4（发送）	1001	481h-4FFh
PDO4（接收）	1010	501h-57Fh
SDO（发送）	1011	581h-5FFh
SDO（接收）	1100	601h-67Fh
NMT Error Control	1110	701h-77Fh

2.1.1 网络管理报文

网络管理报文(NMT)由主站发出，用来更改从节点运行状态，CAN 标准报文的功能码之后的字节是 RTR 位，该位为 0 表示报文是 NMT 报文。RTR 之后的字节是节点状态位，节点分别处于预运行、运行、停止状态时，该字节取值为 80、01、02。节点状态位之后是 NMT 发往的节点号，如果要向所有节点广播信

息，节点号是 00。

2.1.2 节点保护报文

节点保护机制是指主节点向从节点发送读取状态的报文，从节点将自己的状态回复给主节点，以确定从节点是否工作正常。例如，如果主节点向节点号为 5 的从节点发送报文 705 读取从节点状态，从节点此时处于停止状态，则返回报文 705 05，主节点发送 000 80 05，将从节点状态改为预运行，主节点再次发送 705，则从节点返回报文 705 FF。

2.1.3 心跳报文

从节点进入预运行状态时，如果心跳报文功能被启动，就会周期性的发送心跳报文。对象字典索引 1017 子索引 00 表示发送心跳报文间隔的时间。例如要使节点号为 5 的从节点每隔 100ms 发送一次心跳报文，主节点发送 SDO 格式为：605 2B 17 10 00 64 00 00 00，其中 605 是发送 SDO 的 COB-ID，2B 代表该 SDO 的命令字，17 10 代表索引 1017，00 代表子索引，64 是发送心跳报文间隔的时间的 16 进制表示。

2.1.4 服务数据对象

服务数据对象(SDO)用来读写对象字典。请求读写的是客户，被请求的是服务器，读是下载，写是上传。SDO 通讯传送优先级较低的信息，一次可传送任意多个字节。本课题中支持的 SDO 报文格式为 SDO 上传/下载加速传输。即所有的 SDO 都包含 8 个字节长度。例如，如果主节点向从节点索引为 0x1400，子索引为 2 里写入 1 个字节的数据 0xFD，则主节点发送数据帧：605 2F 00 14 02 FD 00 00 00。从节点响应：585 60 00 14 02 00 00 00 00。如果主节点向从节点索引为 0x1603，子索引为 1 里写入 4 个字节的数据 0x60120208，则主节点发送数据帧：605 23 03 16 01 08 02 12 60。从节点响应：585 60 03 16 01 00 00 00 00。如果主节点读从节点索引为 0x1400，子索引为 2 里 1 个字节的数据 0xFD，则主节点发送数据帧：605 40 00 14 02 00 00 00 00。从节点响应 585 4F 00 14 02 FD 00 00 00。如果主节点读从节点索引为 0x1603，子索引为 1 里 4 个字节的数据 0x60120208，则主节点发送数据帧 605 40 03 16 01 00 00 00 00 从节点响应 585 43 03 16 01 08 02 12 60。

2.1.5 过程数据报文

过程数据对象(PDO)用来传输实时数据，分为周期、非周期传送模式和同步、非同步传送模式，数据内容由 SDO 来配置。一次可传送 0 到 8 个字节。

如果要配置发送 PDO，例如要求是配置发送 PDO 所在对象字典的索引项 1800+n，cobId 为 0x387，PDO 发送按照同步周期发送，依次包含数据 X（2 个字节）和 Y（4 个字节）。

数据 X 被定义在索引 0x6000 子索引 03 中

数据 Y 被定义在索引 0x2010 子索引 21 中

配置步骤如下：

- 向索引 1800+n 子索引 01 中写入 cobId
- 子索引 02 中写入发送模式，1-0xF0 代表每收到 t 个 SYNC 报文发送一次 PDO，FD 代表每收到 PDO 请求时发送一次 PDO，FF 代表每当事件到来时发送一次 PDO。
- 向代表 PDO 映射的索引 1A00+n 子索引 0 中写入包含在 PDO 中的数据个数
- 子索引 01 写入要发送的第一个数据的索引子索引和数据长度，写入 6000 03 08
- 子索引 02 写入要发送的第二个数据的索引子索引和数据长度，写入 2010 21 20

比如要配置节点号为 5 的从节点的 1802 索引，使其每 3 个同步周期发送 1 次 PDO，SDO 配置报文如下定义：

605 23 02 18 01 00 00 07 38

605 2F 02 18 02 03 00 00 00

605 2F 02 1A 00 02 00 00 00

605 23 02 1A 01 08 03 00 60

605 23 02 1A02 20 21 10 10

如果要配置接收 PDO，例如要求是配置接收 PDO 所在对象字典的索引项 1400+n，cobId 为 0x183，PDO 按照同步周期接收，依次包含数据 X（2 个字节）和 Y（4 个字节）。

数据 X 被定义在索引 0x6000 子索引 03 中

数据 Y 被定义在索引 0x2010 子索引 21 中

配置步骤如下：

- 向索引 1400+n 子索引 01 中写入 cobId
- 子索引 02 中写入接收模式，1-0xF0 代表每收到 t 个 SYNC 报文接收一次

PDO，FD 代表每收到 PDO 请求时接收一次 PDO，FF 代表每当事件到来时接收一次 PDO。

- 向代表 PDO 映射的索引 1600+n 子索引 0 中写入包含在 PDO 中的数据个数
- 子索引 01 写入要接收的第一个数据的索引子索引和数据长度，写入 6000 03 08
- 子索引 02 写入要接收的第二个数据的索引子索引和数据长度，写入 2010 21 20

比如要配置节点号为 5 的从节点的 1802 索引，使其每 3 个同步周期发送 1 次 PDO，SDO 配置报文如下定义：

605 23 02 14 01 00 00 03 18

605 2F 02 14 02 03 00 00 00

605 2F 02 16 00 02 00 00 00

605 23 02 16 01 08 03 00 60

605 23 02 16 02 20 21 10 10

2.1.6 同步报文

同步报文是针对主节点来说的，从节点一般不发送同步报文。对象字典索引 1007 子索引 00 表示主节点发送同步报文的周期。例如要使主节点每隔 1000ms(0x27)发送一次心跳报文，配置格式为 601 23 06 10 00 10 27 00 00。使之开始发送同步报文，向主节点索引 1005 子索引 0 中写入 0x40000080。

2.2 从站协议栈的总体结构

CANopen 从协议栈的设计采用状态机的形式，根据 CANopen 协议 DS301 的规定，节点共有 4 种状态，分别是初始化、预运行、运行、停止。根据节点所处的不同状态。节点上电后，进入初始化状态，对 CAN 硬件进行初始化，设置节点号，建立对象字典 1005H、1006H、1007H、1017H 索引项，然后自动进入预运行状态。进入与运行状态的节点可以接收来自主节点除 PDO 之外所有的报文，并启动心跳报文管理函数。从预运行状态进入运行状态是由主节点发送 NMT 命令来启动的。在运行状态的从节点可以发送并接受 PDOI 报文。停止状态的从节点只能接受主节点的 NMT 报文。从协议栈整体流程图如图 2-1 所示。

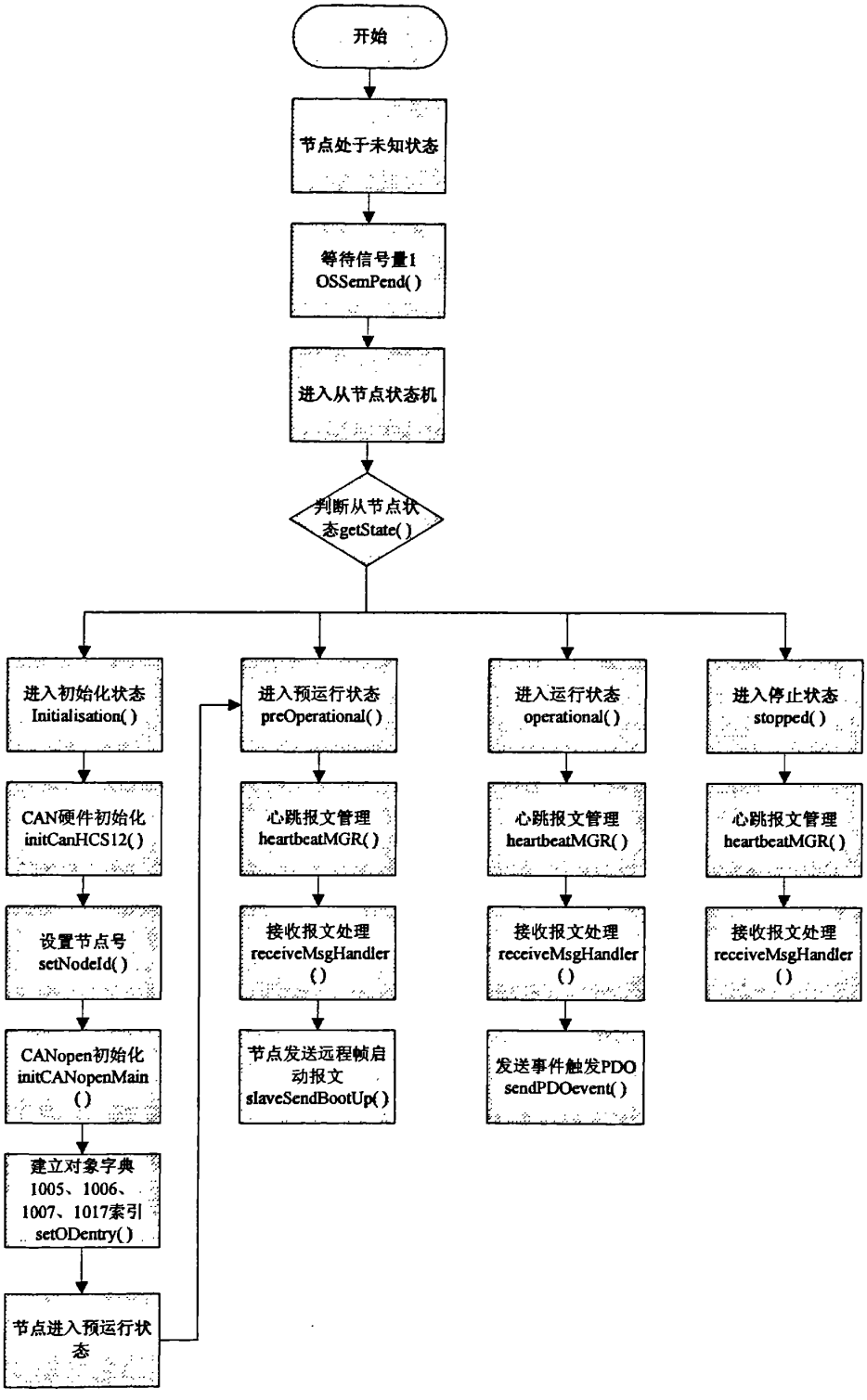


图 2-1 从协议栈总体流程图

Figure 2-1 The Total Process of Slave Stack

2.2.1 报文的底层处理过程

从节点上电启动后，一直循环运行状态机，等待来自主节点的管理报文，当报文到来的时候，基于中断的接收报文机制产生中断，中断由 MC9S12XDP512 的硬件产生，中断向量是在 CodeWorrior 的.prm 文件中定义，定义如下：

VECTOR ADDRESS 0xFF92　can4HdlRcv

其中，0xFF92 是中断函数 can4HdlRcv 的中断向量地址。VECTOR ADDRESS 是用来定义中断向量的格式。

中断发生后 CANopen 处理接收报文。f_can_receive( )与中断函数 void interrupt　can4HdlRcv(void)成对出现。接收过程为：首先中断发生，定义的指向报文堆栈的 t_pointerStack 类型的结构体 ptrMsgRcv 中的 ptrMsgRcv.w 被置位，即向报文堆栈中写信息使能，该操作导致了中断函数开始接收报文。

```
typedef struct {
  UNS8 w ; /* 接收使能*/
  UNS8 r ; /* 发送使能 */
} t_pointerStack;
```

中断发生后，CAN 接收寄存器 CANRCVDTA 里的数据全赋值给 RCOBID，RLEN，Rdata[]这三个变量。

```
RCOBID=IO_PORTS_16(CAN4+ CANRCVID) >> 5;
RLEN=IO_PORTS_8(CAN4 + CANRCVLEN) & 0x0F;
Rdata[0]=IO_PORTS_8(CAN4 + CANRCVDTA + 0);
Rdata[1]=IO_PORTS_8(CAN4 + CANRCVDTA + 1);
Rdata[2]=IO_PORTS_8(CAN4 + CANRCVDTA +2);
Rdata[3]=IO_PORTS_8(CAN4 + CANRCVDTA + 3);
Rdata[4]=IO_PORTS_8(CAN4 + CANRCVDTA + 4);
Rdata[5]=IO_PORTS_8(CAN4 + CANRCVDTA + 5);
Rdata[6]=IO_PORTS_8(CAN4 + CANRCVDTA + 6);
Rdata[7]=IO_PORTS_8(CAN4 + CANRCVDTA + 7);
IO_PORTS_8(CAN4 + CANRFLG) |= 0x01;
IO_PORTS_8(CAN4 + CANCTL0) |= 0x80;
```

CAN 接收中断触发了 f_can_receive( )。CANopen 协议中规定了有 6 种通信对象，分别是 NMT、SYNC、TIME STAMP、PDO、SDO、NODE GUARD 等，只有确定了报文的类别，才能去发挥不同报文应起的作用。通过定义不同功能函数的方法来实现报文的解析。功能函数包括：proceedNMTstateChange( )，改变从节点状态；proceedSYNC( )，用于接收同步报文；proceedPDO( )，处理 PDO；

proceedSDO()，处理 SDO。如图 2-2 所示。

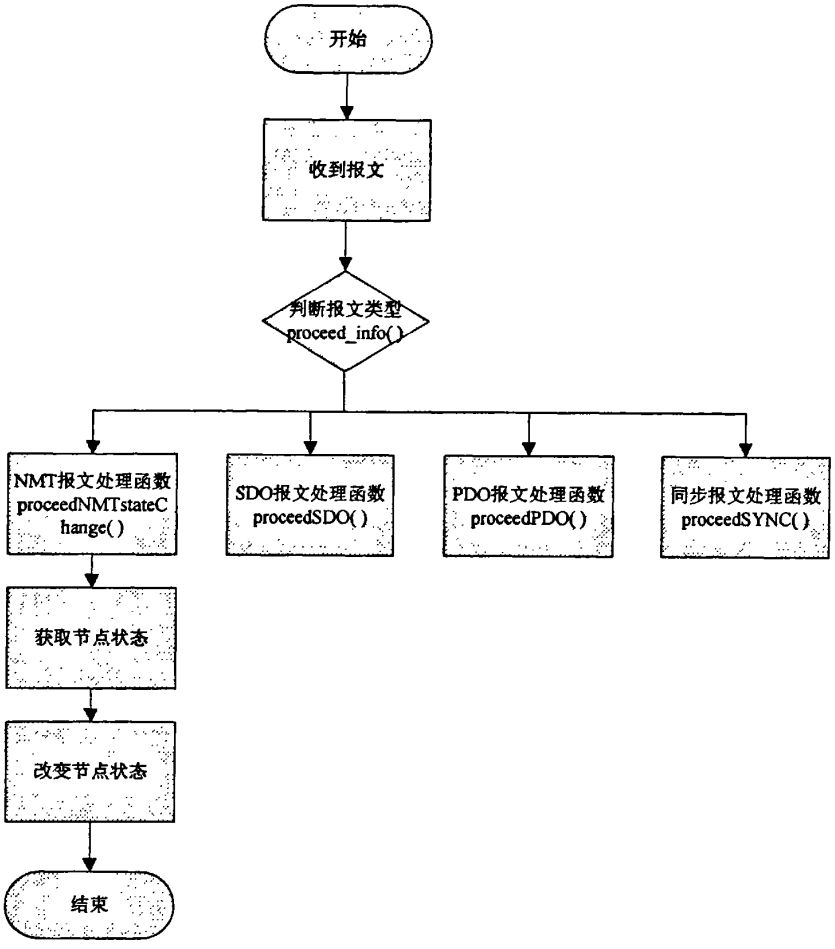


图 2-2 接收报文流程图

Figure 2-2 The Process of Receiving Messages

2.3 SDO 报文的处理

2.3.1 SDO 报文分析

SDO 用来访问设备的对象字典。访问者被称为客户(client)，对象字典被访问并且提供所请求服务的 CANopen 设备被称作服务器(server)。客户的 CAN 报文和服务器的应答 CAN 报文都包含 8 字节的数据(尽管不是所有的数据都有意义)。一个客户请求对应一个服务器应答。

SDO 有两种传输机制：

- 加速传输(Expedited transfer)：最多传输 4 字节的数据

- 分段传送(Segmented transfer)：传输数据长度大于 4 字节

SDO 基本结构如下：

Client→ Server / Server →Client

Byte0	Byte1-2	Byte3	Byte4-7
SDOCommandSpecifier	对象索引	对象字索引	数 据 **

或者是

Client→ Server / Server →Client

Byte0	Byte1-70
SDO Command Specifier	最大 7 字节数据

SDO Command Specifier（SDO 命令字）包含如下信息：

- 下载/上传(Download/upload)
- 请求/应答(Request/response)
- 分段/加速传送(Segmented/expedited transfer)
- CAN 帧数据字节长度
- 用于后续每个分段的交替清零和置位的触发位(toggle bit)

CANopen 中 SDO 的功能根据主从的不同而分别充当 server 和 client，本课题中的 CANopen 节点是从节点，所以从节点是 server，根据 DS301 的规定，SDO 对 server 的操作可以分为，向 server 对象字典中写数据和从 server 对象字典中读数据。

当判断收到的报文是何种 SDO 之后，就开始对 SDO 作相应处理。将 SDO 中的数据逐一赋值给 SDO 专用的结构体变量，包括 8 个字节的数据，SDO 命令字，数据长度，然后打包成要发送出去的 SDO 报文，将 SDO 发送出去。

发送 SDO 的函数包括两部分功能。首先是将处理完后的 SDO 解析，获取对象字典索引号为 1200 的项，赋值给待发送的 SDO 结构体，然后调用 f_can_send( ) 函数，发送给底层的硬件。

整体流程图如图 2-3 所示。

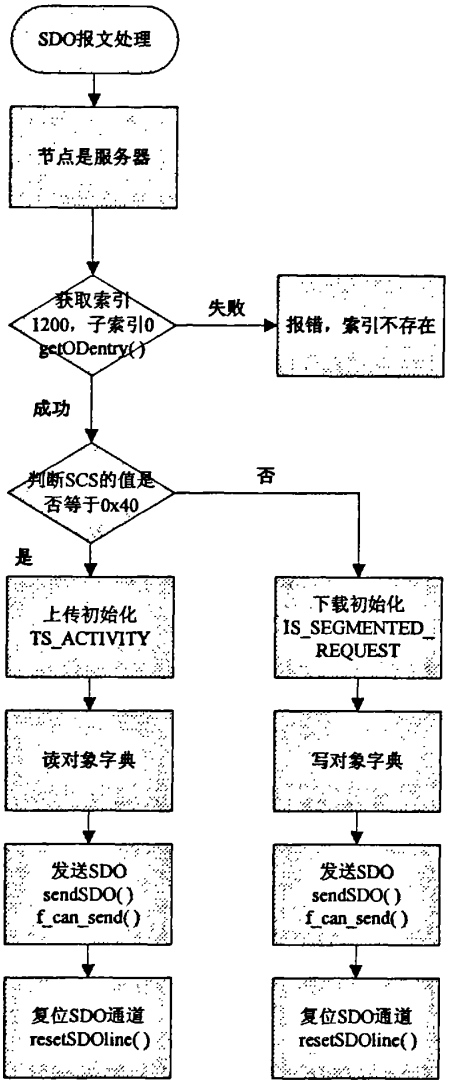


图 2-3 处理 SDO 报文流程图

Figure 2-3 The Process of Dealing with SDO Messages

2.4 PDO 报文的处理

2.4.1 PDO 报文分析

PDO 用来传输实时数据，数据从一个创建者传到一个或多个接收者。数据传送限制在 1 到 8 个字节，PDO 通讯并没有协议规定，PDO 的数据内容只由它的 CAN ID 定义，假定创建者和接收者知道这个数据内容。

每个 PDO 在对象字典中用 2 个对象描述，包括：

PDO 通讯参数，说明哪个 COB-ID 将被 PDO 使用，传输类型，禁止时间和定时期周期。

PDO 映射参数，包含一个对象字典中对象的列表，这些对象映射到 PDO 里，包括它们的数据长度。创建者和接收者必须知道这个映射，以解释 PDO 内容。

映射应用对象到 PDO 中是在设备对象字典中描述的。如果设备（创建者和接收者）支持可变 PDO 映射，那么使用 SDO 报文可以配置 PDO 映射参数。

PDO 可以有多种传送方式，包括同步（通过接收 SYNC 对象实现同步）、非周期、（由远程帧预触发传送，或者由设备子协议中规定的对象特定事件预触发传送）、周期（传送在每 1 到 240 个 SYNC 消息后触发）、异步、由远程帧触发传送、由设备子协议中规定的对象特定事件触发传送。

表 2-2 中给出了由传输类型定义的不同 PDO 传输模式，传输类型为 PDO 通讯参数对象的一部分，由 8 位无符号整数定义。

表 2-2 PDO 传输模式

Table2-2 The Transmit Modes of PDO

传输类型	触发 PDO 的条件（B=二者必须，O=单独或二者必须）			PDO 传输
	SYNC	RTR	Event	
0	B	-	B	同步，非循环
1-240	O	-	-	同步，循环
241-251	-	-	-	Reserved
252	B	B	-	同步，在 RTR 之后
253	-	O	-	异步，在 RTR 之后
254	-	O	O	异步，制造商特定事件
255	-	O	O	异步，设备子协议特定事件

说明：

SYNC——接收到远程帧 SYNC-object

RTR——接收到远程帧

Event——例如数值改变或者定时器中断。

传输类型为：1 到 240 时，该数字代表两个 PDO 之间的 SYNC 对象的数目。

一个 PDO 可以指定一个禁止时间，即定义两个连续 PDO 传输的最小间隔时间，避免由于高优先级信息的数据量太大，始终占据总线，而使其他优先级较低的数据无力竞争总线的问题。禁止时间由 16 位无符号整数定义，单位 100ms。

一个 PDO 可以指定一个事件定时周期，当超过定时时间后，一个 PDO 传输

可以被触发（不需要触发位）。事件定时周期由 16 位无符号整数定义，单位 1ms。

PDO 通过 CAL 中存储事件类型的 CMS 对象实现。PDO 数据传送没有上层协议，而且 PDO 报文没有确认（一个 PDO 需要一个 CAN-ID）。每个 PDO 报文传送最多 8 个字节（64 位）数据。

2.4.2 从节点 PDO 的功能及设计

PDO 有读写两种协议，读 PDO 向对方发一个数据请求的报文后还需要接收数据响应，写 PDO 将数据直接发送而不需要响应。因此在 PDO 接收过程中如果是 PDO 数据，则程序将数据存储在对象字典中；如果是 PDO 请求，则程序在取出对象字典相应的发送数据后立即作出响应。

定义了两个接口函数 PDOmGR 和 sendPDO。PDOmGR 是 PDO 管理接口函数，用于协调 PDO 发送和接收的冲突问题，规定接收数据优先级较高。定义了当前 PDO 的状态，当下个 PDO 到来时如果当前的 PDO 处于接收占用状态，则忽略发送并返回发送失败，否则继续发送并返回成功。sendPDO 完成 pdo 数据结构数据向 Message 数据结构的转换，最后将数据发给协议栈通用接口 f_can_send( ) 继而传送给操作系统设备驱动，完成一次 PDO 请求。

PDO 发送过程也分成数据和请求两种：发送请求较简单，sendPDOrequest( ) 函数首先判断对象字典有无映射，若有则直接调用 sendPDO( )将请求信息发送出去；发送数据则要另外执行函数 buildPDO( )，功能为设置 COBID、设置 PDO 号、设置映射参数、设置传输类型、调整低位字节首先发送等一系列动作。

sendPDOevent( )是一个基于状态机的函数。共有 8 种状态。当主节点配置从节点的 PDO 时，从节点初始状态为 1 状态，主节点开始获取从节点对象字典索引的 1A00 项，该索引正是发送 PDO 的映射参数索引。如果该项存在，则证明该从节点有可以发送的数据，状态机跳到 2 状态。2 状态时主节点检查从节点 1A00 象子索引的个数，遍历一遍然后跳到状态 3。状态 3 是再次检查对象字典是否读到了，然后跳到状态 4。状态 4 的作用是确定 PDO 映射索引里的相关数据真正的地址，也就是说检索到真正的数据的位置。然后跳到状态 5。状态 5 中获取了 1800 的对象字典索引项，即为发送 PDO 的对象字典索引。将状态 4 中确定的数据所存放的对象字典索引，子索引，数据长度，数据一起放入 1800 中。跳到状态 7，检查 1800 索引中的发送类型子索引项，如果该值为 255，则是事件驱动的 PDO。然后跳入状态 8，开始将数据打包成待发送的 PDO。

再由 sendPDO( )函数将 PDO 报文赋值给指向报文的指针。具体过程同 SDO 的发送过程。不再赘述。然后由 f_can_send( )发送给硬件。发送 CAN 报文不使用中断机制，采用查询方法。f_can_send( )调用 canMsgTransmit( )函数，当 CAN

寄存器中的 CANTFLG 标志位置位的时候，开始将待发送报文放到发送寄存器中。

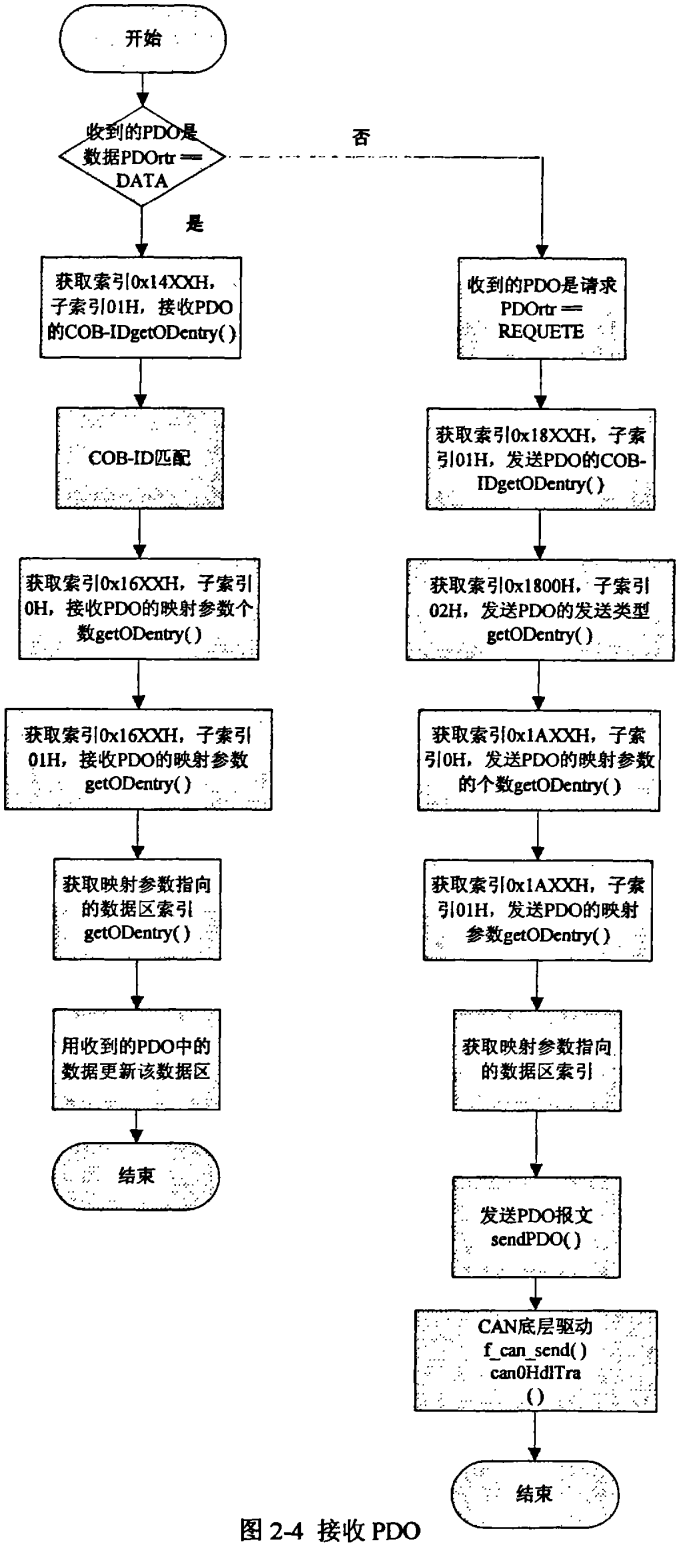


图 2-4 接收 PDO

Figure 2-4 Receive PDO Messages

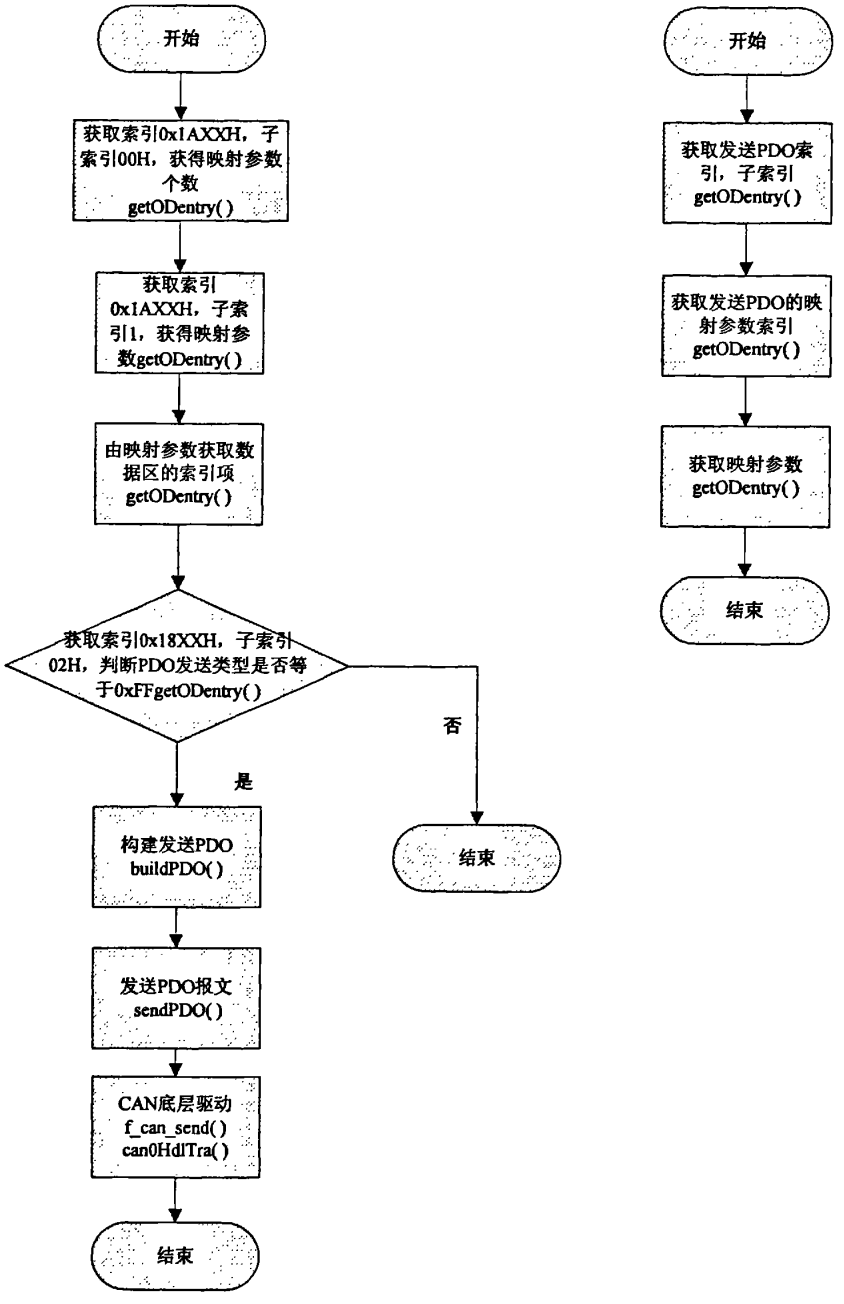


图 2-5 发送事件 PDO

Figure 2-5 Transmit PDO Based on Event

2.5 节点保护与心跳报文

节点保护和心跳报文用于监控节点状态。节点保护是指主节点监控从节点状态，通过发送节点保护 NMT 报文来实现。心跳报文机制是指主节点发送远程请求到一个特定节点，节点给出应答报文，应答报文中包含从节点的状态信息。CANopen 协议中，规定对象字典索引项 1017H 为生产者心跳报文周期索引，该索引保存了从节点产生心跳报文的周期，如果值为 0，则表示从节点不启用心跳报文机制。主节点如果希望从节点产生心跳报文，需要修改从节点的该对象字典，修改值就是从节点产生心跳报文的周期。本课题中的从节点首先将其对象字典的 0x1017 项的值定义为 0，刚启动的从节点是不产生心跳报文的。因此，对于从节点的监控是采用节点保护的形式。主节点向从节点发送节点保护的 NMT 报文，从节点向主节点发送包含自己状态的报文。

如果主节点修改了从节点对象字典的 0x1017 项的值，使其不为 0，那么从节点就会启动节点保护机制。按照主节点写入的周期值来周期性发送心跳报文。heartbeatMGR( )用来处理心跳报文，首先读从节点的对象字典 0x1016 项，如果该项不存在，表明该从节点不具备监控其他从节点心跳报文的功能，也就是说，网络中没有其他从节点向本从节点发送心跳报文。接下来读从节点的 0x1017 项，如果该项值为 0，则不产生心跳报文，如果已被主节点修改为不为 0 的数，则从节点启动心跳报文，并由 f_can_send( )将心跳报文发送出去。

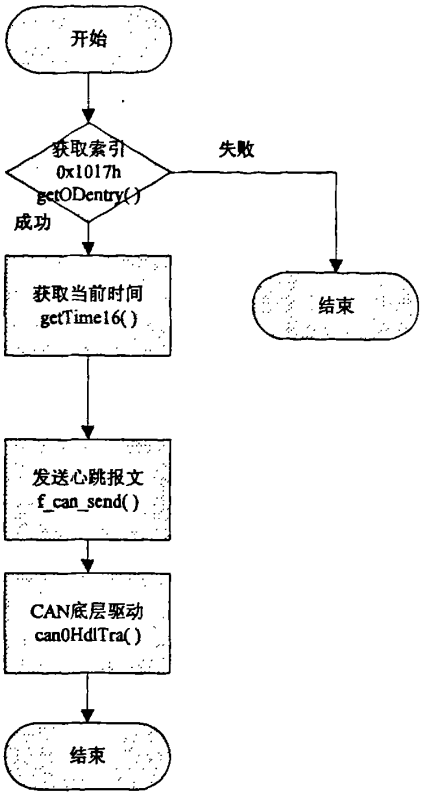


图 2-6 发送心跳报文

Figure 2-6 Transmit Heartbeat Messages

2.6 与硬件的接口

CANopen 从站有多处需要产生定时中断，例如心跳报文周期性产生，同步报文时间窗，MC9S12XDP512 具有 8 路增强型定时器，任意一路都可以用来产生中断。初始化定时器相关寄存器之后，进入时间中断函数 timeInterrupt( )。

f_can_send( )和 f_can_receive( )是底层的 CAN 报文发送和接收函数。通过对 CAN 相关寄存器进行操作来实现。f_can_send( )首先对 CAN 的发送标志位寄存器 CANTFLG 进行判断，如果不为 0，证明有报文需要发送，将报文的标识符、报文长度、优先级和数据依次写入相应寄存器，从而实现 CAN 报文的发送。

2.7 本章小结

本章给出了 CANopen 协议中的报文格式，包括 NMT、节点保护、心跳报文、SDO、PDO，并设计了从节点代码的结构，主要包括协议栈接收报文的底层处理

流程，应用层中 SDO、PDO 以及节点保护报文的报文发送和接收。

第 3 章 硬件平台设计与实现

3.1 控制器的选择

系统以飞思卡尔16位高性能嵌入式单片机MC9S12XDP512作为控制与运算单元。S12X微控制器特性如下[16][40]：

- 512KB的闪存（计划的版本达到32KB到1KB的闪存）
- 40MHz的增强CPU
- XGATE模块
- 32KB的RAM、4KB的EEPROM（电可擦可编程只读存储器）
- 5路控制器区域网络(CAN)通信接口(仅限于HCS12XD)
- 6路串行通信接口/本地互联网(SCI/LIN)、3路串行外设接口(SPI)、2路集成电路间总线(I^2C)通信接口
- 高级中断功能
- 增强的捕捉定时器
- 优先中断定时器
- 10位模数转换器(ADC)
- 8信道的脉冲宽度调制(PWM)
- 带有追踪缓存的片上单线后台调试模式(BDM)
- 在-40℃至125℃的温度范围内运行XGATE协处理器功能
- 外设协处理器
- 可编程的直接存储器访问(DMA)控制器
- 实时中断处理程序
- 虚拟外设控制器开发

S12X系列保持了与HCS12系列的高度引脚和代码兼容性，并达到五倍于HCS12微控制器的性能，二者的高度兼容性可以容易地升级并实现S12X系列的高速率。S12X采用增强内核和增强外围设备，提高总线速度最高可达40MHz（HSC12只能达到25MHz），并且具备完全的CAN功能，改进了中断处理能力[17]。S12X系列与HCS12具有显著的不同。S12X系列采用一个增强CPU，可以额外处理172条指令，从而显著提高性能和和效率，并改进了存储器变换、代码效率和32位计算。同时，利用S12X的新特性来优化应用设计以获得更高的性能表现。图3-1是控制器与被控对象整体连接图：

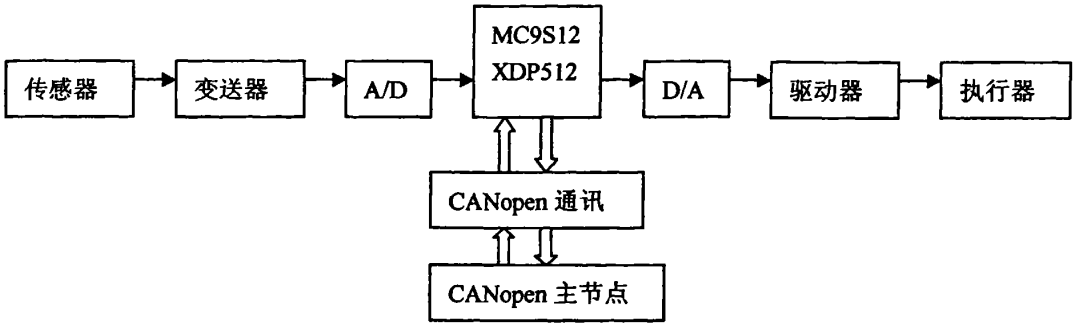


图 3-1 控制器与被控对象连接图

Figure 3-1 The Connecting Framework of Controller and Control Object

3.2 硬件平台构成

控制电路板分底板及核心板两部分，整体框架如图3-2所示。核心板包括处理器MC9S12XDP512及外围电路、电源模块、BDM模块、复位电路、串口模块及144引脚扩展槽等，其它元器件均在底板设计，包括CAN接口电路、A/D、D/A、USB接口、Flexray接口、以太网接口、继电器模块等。本课题主要对底板进行原理图及PCB布线设计，并对电路板的接地进行抗干扰处理。这里主要介绍被测信号A/D采集模块、控制信号D/A转换模块以及CAN通信模块的设计方法。

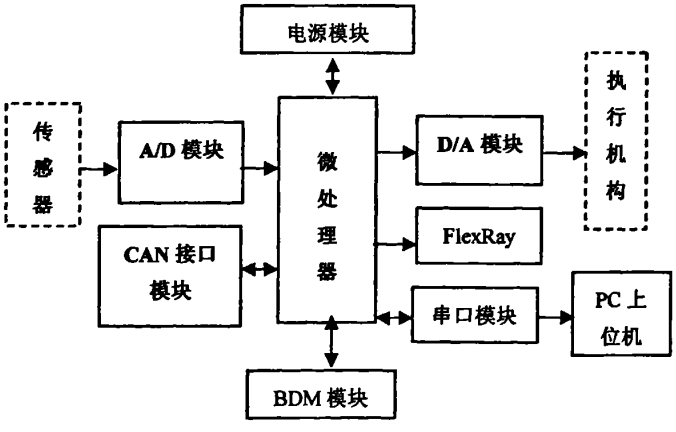


图 3-2 控制器硬件平台结构图

Figure 3-2 The Framework of Controller Hardware Platform

3.2.1 A/D 模块设计

智能仪表需要采集传感器的各种信号，由于被控对象的各类信号均为标准信号1-5V，所以需要使用模数转换模块(A/D)来实现。MC9S12XDP512内置了16路/8路可选的10位/8位的A/D模块，该模块的特性概括如下[16]：

- 8/10位精度；
- 10位数字量转换仅需7us；
- 采样缓冲放大器；
- 可编程的采样时间；
- 数据左/右对齐方式，无符号/有符号结果；
- 外部触发转换通道；
- 转换完成中断；
- 模拟/数字量复用输入通道；
- 1-16顺序转换长度；
- 连续转化模式；
- 多通道扫描；

A/D模块内部可以分为三个部分[18]：IP总线接口、转换模式控制/寄存器列表、自定义模拟量。框架图如图3-3所示。

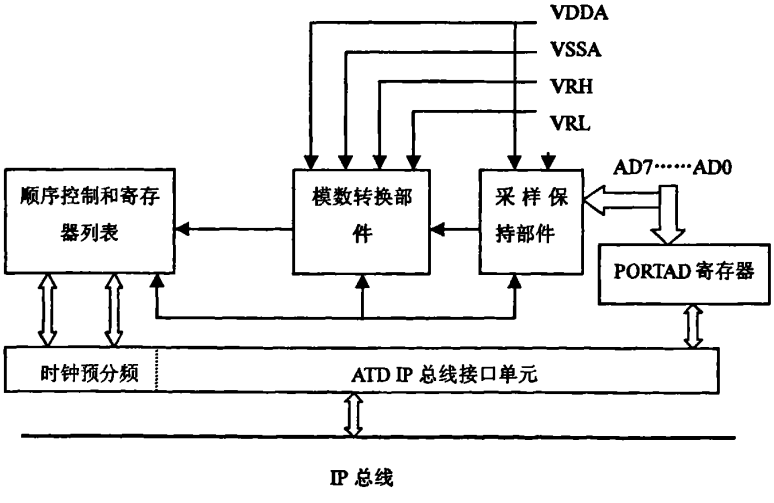


图 3-3 A/D 模块内部框架

Figure 3-3 The Inner Framework of A/D Model

IP总线接口负责该模块与总线的连接，实现A/D模块和通用I/O的目的。转换模式控制寄存器列表中有控制该模块的所有寄存器。自定义模拟量负责实现模拟量到数字量的转换。为了与外部信号同步进行A/D转换，A/D有一个外部触发转换通道，用户可以选择触发方式（沿触发、电平出发）。A/D模块设有时钟分频

机制。最大转换始终为2MHz，最小转换时钟为500kHz，要确定芯片内部总线时钟，使分频后的时钟处于两者之间，否则可能得不到正确结果。还可以设定转换结果为有符号或者是无符号数。例如VRH为5.12V，VRL为0V时，输入5.12V，8位有符号的结果为-$7F，无符号的结果是$FF.A/D模块允许设置顺序转换，最大的顺序转换序列长度是8。

在电路设计上，目标系统的输入电压范围是0到5V，而电路板的供电电压也是0到5V，A/D电路的参考电压可以选择0V和5V。数字电源和模拟电源要分开设计，数字地和模拟地也要分开设计，通常采用在芯片的供电电源和数字地上加一个比较大的电感作为模拟电源和模拟地。芯片VDDA接模拟电源，VSSA接模拟地，若AD输入电压范围和电源电压相同，则参考高电平VRH接VDDA，VRL接VSSA。具体电路如图3-4所示：

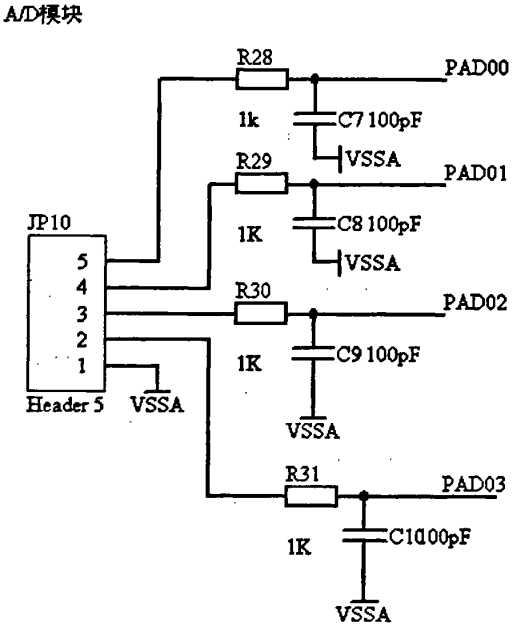


图 3-4 A/D 电路图

Figure 3-4 The Circuit of A/D

3.2.2 D/A 模块设计

在智能仪表的应用中需要控制器用模拟信号来控制外部器件，作为智能仪表的控制器，输入信号和输出信号都是标准信号，需要将该模拟量转换成数字量，但 MC9S12XDP512 芯片上没有模数转换器(DAC)，可以考虑用一个简单的外部 RC 电路来构成一个低通滤波器。基本原理是利用 PWM 模块用程序来控制波形占空比、周期、相位的波形，在 PWM 输出口加入一个低通滤波。这样的 DAC 的优点是它的分辨率可以由软件来设置，并且可以降低系统成本。

通过反复实验，确定 RC 滤波电路的 R=8K，C=0.01uF 和 0.1uF 时，滤出的波形很平整。低通滤波器出来的信号接一个电压跟随器，目的是避免电压跟随器之前的负载分压的影响。要使输出阻抗尽可能地大。然后接 Howland 电流泵。作用是将电压值转换成电流值。利用 LM358 芯片有两路放大电路，一路用来做电压跟随器，一路用来做 Howland 电流泵。使用双电源供电。目标对象的输入电流为 4-20mA，所以将电压跟随器输出的电压接一个 250 欧姆的电阻，可以将 1-5V 的电压转化成 4-20mA。电流泵的电桥满足平衡即可。1-5V 可以由占空比寄存器中写入 0x00 到 0xCC 的值来获得。具体电路如图 3-5 所示：

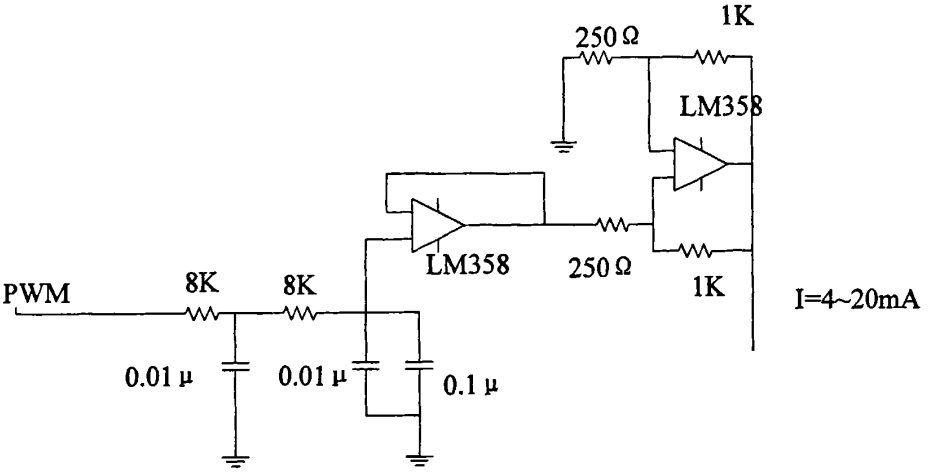


图 3-5 PWM 用作 D/A 电路图

Figure 3-5 The Circuitof PWM for D/A

3.2.3 CAN 通道设计

图3-6所示为本系统CAN原理框图。CAN 收发器TJA1050 将控制器[19]转换的TTL 数据转换成CAN 标准电平，由CANH、CANL发给CAN总线；反之为接收过程。控制器和收发器TJA1050之间添加光电耦合器件和电源隔离器件，提高总线的抗干扰能力。

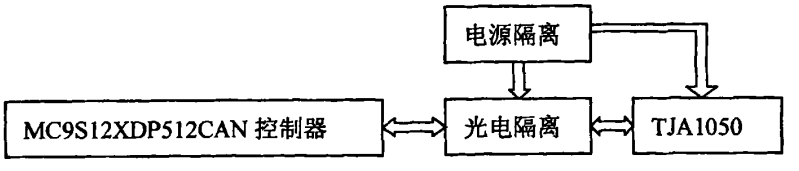


图 3-6 CAN 节点原理框图

Fig3-6 CAN Node Schematic Diagram

电源隔离芯片选用 DCP010505，光电耦合器件选用 6N137，两者配合使用，

可以提高总线的抗干扰能力。

CAN 收发芯片选用Philips公司生产的TJA1050 、用以替代 82C250 的高速 CAN 总线驱动器。该器件与ISO11898 标准完全兼容[19]，除了具有 82C250 的主要特性以外，在某些方面的性能还作了很大的改善，如具有强电磁干扰下宽共模范围的差动接收能力，优化了输出信号CANH和CANL之间的耦合，降低了信号的电磁辐射(EMI)。

具体电路如图3-7所示[36]：

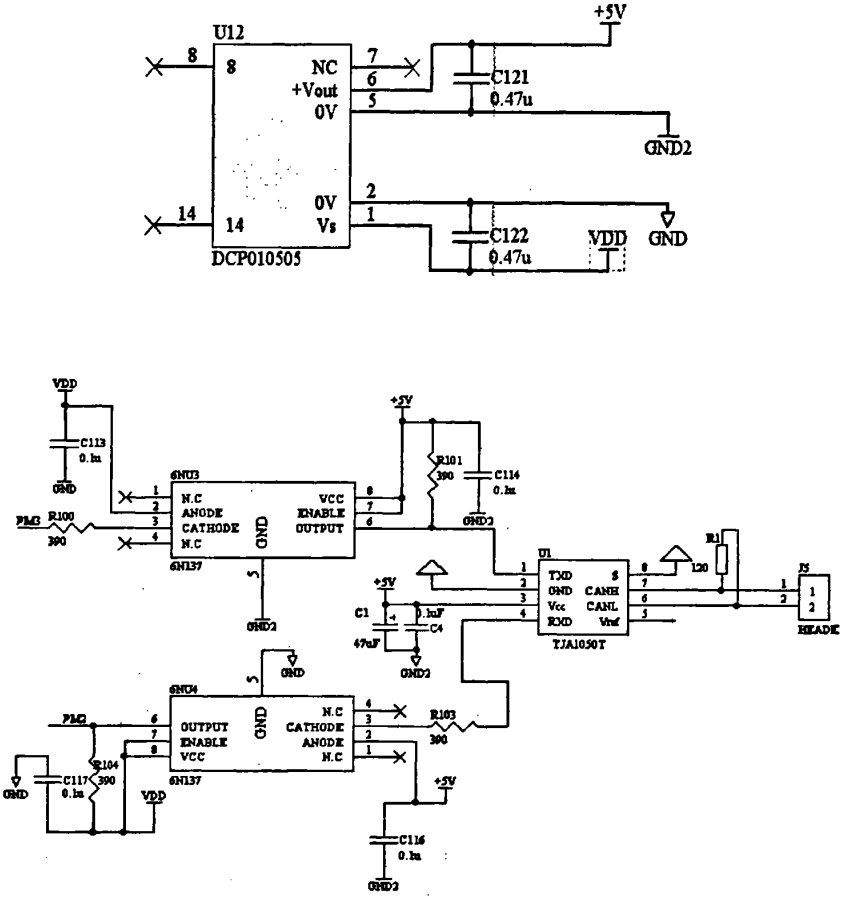


图 3-7 CAN 接口电路图

Figure 3-7 The Circuit of CAN Interface

3.3 本章小结

本章首先提出了系统总体设计方案和开发流程，建立比较清晰的设计思路。硬件资源方面简要描述电路板各芯片的主要功能，其中介绍 A/D 模块、D/A 模块、CAN 模块的开发过程，详细阐述了调试和开发的难点，为下文设备驱动程序设计和 CANopen 协议栈的移植提供稳定通畅的硬件平台。

第 4 章 移植 CANopen 从协议栈

4.1 开发环境的建立

在嵌入式系统软件中，最初的嵌入式微处理器的裸片上没有任何程序，因此一般都是在 PC 机上建立集成开发环境，在集成开发环境中编写操作系统及应用软件源代码并进行编译调试，然后通过仿真器下载到目标板上进行板极调试，最后烧写到 Flash 里完成开发[20]。

为飞思卡尔单片机提供商用软件的公司有很多，其中 Merowerks 是 Motorola 与 1999 年收购的、独立运作的子公司，CodeWarrior 是公司的、专门面向飞思卡尔所有 MCU 与 DSP 嵌入式应用开发的软件工具。CodeWarriorfor S12 是面向以 HC12 或 S12 为 CPU 的单片机嵌入式应用开发的软件包。包括集成开发环境 IDE、处理器专家库、全芯片仿真、可视化参数显示工具、项目工程管理器、C 交叉编译器、汇编器、链接器以及调试器。支持用户使用汇编、C 及 C++语言开发。可进行软件仿真也可以和 USB HCS08/HCS12 Multilink 一起连接目标板进行板级调试。交叉编译调试环境如图 4-1 所示。

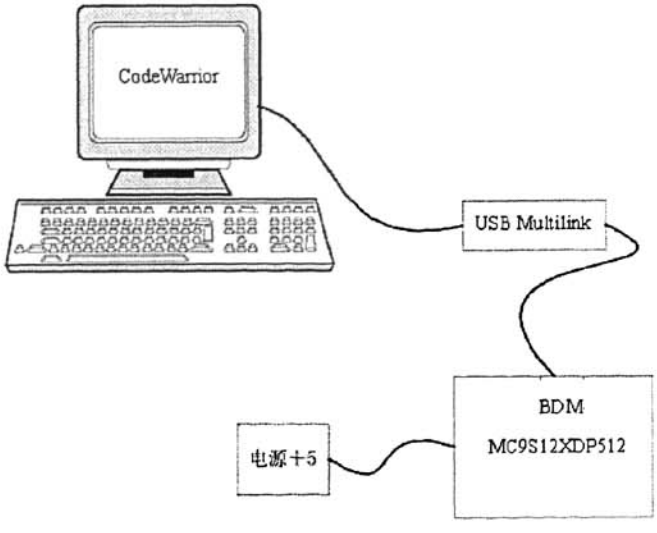


图 4-1 交叉编译环境

Fig4-1 Cross Compilation Environment

图中的仿真器是美国 PEMICRO 公司生产的嵌入式开发工具，USB HCS08/HCS12 Multilink BDM 是一款性价比较高的编程调试工具，支持 HCS08，HCRS08，HCS12 系列的所有型号单片机。通过 BDM 接口实时在线调试 HCS08，RS08 和 HCS12 系列单片机快速 FLASH 在线编程，可以为调试者提供单步、断

点（包括条件断点），并可查看寄存器值、内存值、堆栈等，极大方便了基于S12的嵌入式软件调试。

从节点开发系统包括：MC9S12XDP512 开发板、S12 的集成开发环境CodeWarrior 4.6、USB Multilink 仿真器。表 4-1 对各文件夹中的代码做出说明。

表 4-1 代码结构

Table4-1 The Structure of Code

文件路径	文件说明
CanOpen/CanOpenMain	包括所有与处理器相关的.c 文件
CanOpen/include/hc12	applicfg.h 处理器硬件配置相关的.h 文件
CanOpen/include/hc12	timerhw.h、interrupt.h 定时器、中断相关的.h 文件
CanOpen/CanOpenDriverHC12	针对 HC12 平台的 CANopen 驱动
CanOpen/AppliSlave_HC12	appli.c、objdict.c 应用程序和描述对象字典的代码

工程的建立以 AppliSlave_HC12 中的 appli.c 文件为基础，把相关的.c 文件添加至工程里。

(1) appli.c 里面包括以下.c 文件，在这里选择把相关的.c 文件用包含到主文件里，这样不但使工程简单，也不用考虑限制版的 CodeWarrior 的添加文件个数的限制。.c 文件包括以下几个文件：canOpenDriver.c、init.c、interrupt.h、objacces.c、sdo.c、pdo.c、objdict.c、lifegrd.c、timer.c、timerhw.c、nmtSlave.c、nmtMaster.c、sync.c、variahw.c。

(2) 以下.h 文件是针对处理器的，这些不必添加到工程里，也是使用 include 的方法加进分别的文件。regs.h、exit.h、interrupt.h、param.h、portsaccess.h、ports_def.h、ports.h、applicfg.h、interrupt.h、timerhw.h、CanOpenMain.h。

(3) 以下文件是 HC12 的相关驱动。在这些文件中定义了所有硬件相关的文件，这些文件与 CANopen 无关，只是与不同的硬件平台有关。这部分也是改动最大的部分。candriver.h、canOpenDriver.h、error.h、interrupt.c、timerhw.c、variahw.c、regbase.h。

4.2 协议栈移植

从站的主程序是一个状态机的轮换，按照 CANopen 协议的规定，从节点启动运行后会有四个状态，分别是初始化、预运行、运行、停止。代码中分别用

Initialisation、Pre_operational、Operational、Stopped 四种状态来表示。用 switch 语句实现状态的切换。

```
switch( getState( ) ) {
case Initialisation:
……
case Pre_operational:
……
case Operational:
……
case Stopped:
……}
```

四个状态的功能函数分别是 initialization( )、preOperational( )、operational( )、stopped()。初始化函数 initialisation( )中包含的程序流程图如图 1 所示。包括中断函数和 CAN 驱动初始化函数 initCanHCS12( )里的 CAN 波特率结构体。

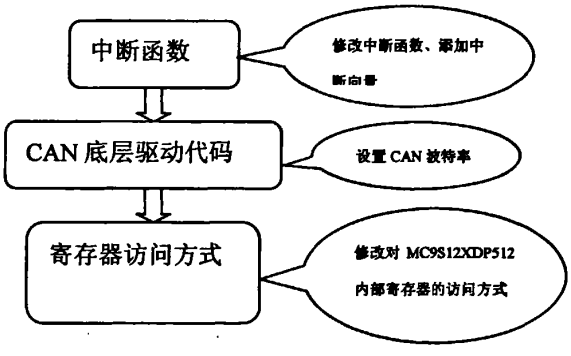


图 4-2 从站协议栈初始化流程

Figure 4-2 The Slave Stack Initial Process

4.2.1 中断函数

(1) CodeWarrior编译环境对中断函数的声明和定义写法如下：

```
void interrupt timer3Hdl(void);
pragma CODE_SEG__NEAR_SEG_NON_BANKED
void interrupt timer3Hdl(void)
{...}
pragma CODE_SEG_DEFAULT
```

注 意 #pragma CODE_SEG __NEAR_SEG NON_BANKED 和 #pragma

CODE_SEG_DEFAULT要成对使用，目的是加上声明，将中断函数放入非分页地址。因为中断矢量只有16位，无法在分页地址中寻址。因此，中断函数必须放入非分页地址。

(2) 添加中断向量

在 CodeWarrior 中建立工程时，生成了几个默认的文件夹，其中.prm 文件夹中定义的是连接使用到的参数，这些是与硬件相关的，可以根据使用的单片机的型号来修改这些参数。该文件夹下的 P&E_Multilink_CyclonePro_linker.prm 和 Full_Chip_Simulation_linker.prm 这两个文件用于定义目标代码的装载地址，可以根据需要修改这个文件。其中，MY_RAM 是程序的数据区，MY_ROM 是程序的代码区。

VECTOR 0 _Startup 表示把单片机的 0FFFF 处的复位向量设为这个程序的入口地址。具体中断向量可以查看数据手册。全部中断向量定义如下：

```
VECTOR 0 _Startup
VECTOR ADDRESS 0xFFE8 timer3Hdl
VECTOR ADDRESS 0xFFE6 timer4Hdl
VECTOR ADDRESS 0xFF90 can4HdlTra
VECTOR ADDRESS 0xFF92 can4HdlRcv
VECTOR ADDRESS 0xFF96 can4HdlWup
VECTOR ADDRESS 0xFF94 can4HdlErr
```

(3) 开中断和关中断语句写法如下，编译器支持C语言内嵌汇编。

```
void unlock (void)
{
__asm("cli");
}
void lock (void)
{
unsigned short mask;
__asm
{
tpa:tsei:"=d"(mask);
}
}
```

4.2.2 CAN 底层驱动代码的设计

CAN驱动程序在从站初始化中作用很重要，它定义了与CAN相关的数据结构和功能函数，依照CAN2.0A协议中定义的CAN报文的格式编写程序，设置了CAN总线通信速率、CAN寄存器等。需要说明的地方是CAN总线初始化中的数据结构定义。该函数位于从节点初始化过程中的initCanHCS12( )函数里，该函数主要是初始化与CANopen相关的硬件，主要是指MC9S12XDP512的硬件。例如函数里的canBusInit是如下结构的结构体：

```
typedef struct {
UNS8   cswai; /* 等待模式为低电压或正常 */
UNS8   time; /* 时间戳定时器使能 (1/0)  */
UNS8   cane; /* CAN使能 (yes=1)    */
UNS8   clksrc; /* 时钟源选择 */
UNS8   loopb;   /* 测试模式 (1/0)*/
UNS8   listen;/* CAN监听*/
UNS8   wupm; /* 低通滤波唤醒(yes=1/no=0)*/
canBusTime clk;   /* 时钟系统初始化的值 */
canBusFilterInit   fi; /* 报文接收寄存器的初始化值 */
 } canBusInit;
```

在candriver.h文件中定义变量bi为canBusInit类型的结构体：

```
extern canBusInit bi;
```

变量bi0就是canBusInit 类型的结构体：

```
const canBusInit bi0 = {
0, /* 等待模式为正常 */
0, /* 时间戳定时器关闭  */
1, /* CAN使能 */
0, /* MSCAN时钟源为晶振时钟 */
0,  /* 自收自发测试模式关闭*/
0, /* CAN监听关闭*/
0,/* 低通滤波唤醒关闭*/
{/* 时钟系统初始化的值 */
1, /* clksrc */
3, /* brp */
0, /* sjw */
0, /* samp */
```

```
1, /* tseg2 */
12,/* tseg1 */
},
{ /* 报文接收寄存器的初始化值 */
0x01, /* 选择设置为4个16 bits 缓冲寄存器*/
0x00, 0xFF, /* filter 0 hight accept all msg */
0x00, 0xFF, /* filter 0 low accept all msg */
0x00, 0xFF, /* filter 1 hight filter all of msg */
0x00, 0xFF, /* filter 1 low filter all of msg */
0x00, 0xFF, /* filter 2 hight filter most of msg */
0x00, 0xFF, /* filter 2 low filter most of msg */
0x00, 0xFF, /* filter 3 hight filter most of msg */
0x00, 0xFF, /* filter 3 low filter most of msg */
}
};
```

4.2.3 寄存器访问方式的设计

使用 CodeWarrior 建立工程，系统默认添加了 mc9s12xdp512.c 和 mc9s12xdp512.h 两个文件，这里包含了对于 CPU 寄存器的定义，地址访问方式。如果不使用系统默认添加的头文件，可以另外写一个头文件来定义寄存器地址。文件名是 ports_def.h，这个文件相当于建立工程时系统默认添加的 mc9s12xdp512.h。在 portsaccess.h 中定义了对寄存器的访问方式，具体如下：

```
extern volatile unsigned char _io_ports[ ];
#define _io_ports ((char*)(0))
#define IO_PORTS_8(adr)   _io_ports[adr]
#define IO_PORTS_16(adr)   *((unsigned volatile short*) (_io_ports + (adr)))
```

这样，在代码中每次访问寄存器的时候，都可以使用以下的格式来访问，比如

IO_PORTS_8(PORTB) = 0xFF

将FF放入寄存器PORTB中

IO_PORTS_16(CAN0IDAR1)= 0xABCD;

将AB放入寄存器CAN0IDAR1，将CD放入寄存器CAN0IDAR1+1中。

4.2.4 其他改动

将变量声明写在其他的语句之前。由于编译器的原因，变量声明写在赋值语

句之后会报错。具体改动的文件如表4-2所示。

表 4-2 改动细节

Table4-2 Modification Details

文件名	语句位置
CanOpenDrive.c	457 行 UNS8 NewPtrw
CanOpenDrive.c	510 行 UNS8 NewPtrw
Sdo.c	380行 c=0;s=0;
Sdo.c	UN32 ObjDict
Sync.c	pSize=&size
…	…

在调试 SDO 的过程中，一直可以接收主节点发送的 SDO 报文，但是在解析的时候就会出错。具体表现在能够正确接收，正确解析，在发送的时候出现格式错误。程序中存储报文采用了堆栈的机制，即设置了一个二维数组来按格式存放报文。参数是按照从右到左的顺序压入栈中的，也就是说从倒数第一个参数入栈开始，最后留在寄存器中的参数还是第一个参数。如果参数不固定，压栈的顺序同样是从右到左。所以在函数中，重新修改了与报文的存储空间相关的函数，使代码满足 CodeWarrior 的压栈顺序。

4.3 本章小结

本章提出了一种构建 CANopen 从节点平台的方法。代码的编译采用专门面向以 HC12 或 S12 为 CPU 的单片机嵌入式应用开发的软件包 CodeWarrior for S12。可视化强，能够大大提高开发人员的效率，缩短开发周期。

第 5 章 CANopen 从站在 μC/OS-II 上的移植

5.1 基于实时操作系统的协议栈整体设计

作为控制器的核心，嵌入式软件采用何种方式编写对控制器的性能发挥起到了至关重要的作用。仅仅由循环和中断服务程序组成的嵌入式程序在面对多任务、多协议的复杂应用时显得不能满足要求[15][16]。随着内存容量的迅速扩大以及价格的下降和嵌入式微处理器的处理速度的提升，操作系统进入到嵌入式系统中已成为可能并已经得到了广泛应用。根据控制器的需求在控制器上移植操作系统是必须的也是可行的。基于操作系统的控制器软件整体设计如图 5-1 所示

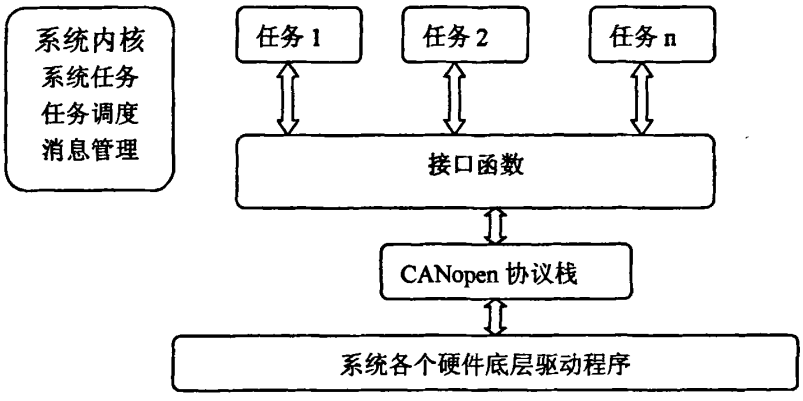


图 5-1 软硬件平台结构图

Figure 5-1 The Framework of Software and Hardware Platform

5.2 μC/OS-II 系统及其在控制器上的移植

μC/OS-II 是由 Jean J.Labrosse 于 1992 年编写的一个嵌入式多任务实时操作系统，并在《嵌入式系统编程》上公开源代码[21]。最早这个系统叫做 uC/OS，后来经过近 10 年的应用和修改，在 1999 年推出了，并在 2000 年得到了美国联邦航空管理局对用于商用飞机的、符合 RTCA DO-178B 标准的认证，从而证明具有足够的稳定性和安全性。因此，μC/OS-II 有能力适应工业网络控制的要求。

μC/OS-II 是用 C 语言和汇编语言编写，所以用户只要做很少的工作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 的构思巧妙，结构简洁精炼，可读性很强，具备实施操作系统的全部功能，经过适当的扩展可以有较为广泛的应用。

μC/OS-II 的体系结构如图 5-2 所示。

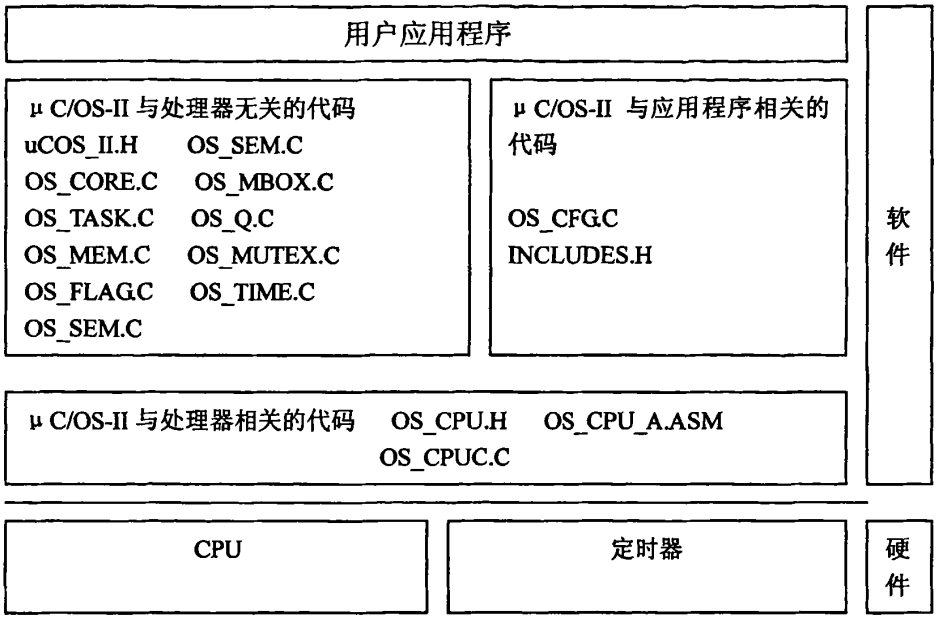


图 5-2 μC/OS-II 的体系结构

Figure 5-2 The Structure of μC/OS-II

5.2.1 μC/OS-II 的移植条件

要移植 μC/OS-II，目标处理器必须满足以下要求：

1、处理器的 C 编译器能产生可重入代码。可重入代码指的是一段代码可以被多个任务同时调用，而不必担心会破坏资源。

2、用 C 语言可以打开和关闭中断。

MC9S12XDP512 支持在 C 语言程序中调用汇编程序，μC/OS-II 通过在汇编语言中可以完成打开和关闭中断。

3、处理器支持中断，并能产生定时中断(10-100Hz)之间。是通过处理器产生的定时器中断来实现多任务之间的调度的，MC9S12XDP512 中有 8 路增强型定时器，可用其产生定时中断。

4、处理器支持足够的 RAM，保存全局变量和作为多任务环境下的任务堆栈。MC9S12XDP512 内部有 14K 的 RAM，能够满足条件。

5、处理器有将堆栈指针和其他 CPU 寄存器读出和存储到堆栈或内存中的指令。μC/OS-II 进行任务调度时，会将当前任务的 CPU 寄存器保存到此任务的任务堆栈中，然后再从另一个任务的堆栈中恢复原来的工作寄存器，继续运行另一个任务。所以，寄存器的入栈和出栈是多任务调度的基础。MC9S12XDP512 的

汇编指令中有将堆栈指针和其它 CPU 寄存器读出和存入堆栈内或内存的指令，满足该条件。

因此，将 μC/OS-II 移植到 MC9S12XDP512 上是可行的。

5.2.2 μC/OS-II 在 MC9S12XDP512 上的移植

编译后的 μC/OS-II 的内核大概有 6-10K，RAM 的占用与系统中的任务数有关。任务堆栈要占用大量的 RAM 空间，堆栈的大小取决于任务的局部变量、缓冲区大小及可能的中断嵌套的层数。所以系统必须要有足够的 RAM 资源。为了减少 RAM 用量，尽量减少内核中不必要的模块（包括邮箱、消息队列、内存管理等），去掉了任务挂起、唤醒和删除等扩展功能，支持任务的创建和管理，也保留了信号量模块来用于任务间的通信[37]。

移植中所需要修改的公共头文件 include.h，这个文件定义使用内核中的哪些模块。Include.h 会被所有的 C 源程序引用。是主头文件，在所有后缀名为.c 的文件的开始都包含 include.h 文件，文件中可以进行内核裁剪。

还要修改和 CPU 相关的三个文件，分别是 OS_CPU.H、OS_CPU_A.ASM（可合并入 OS_CPUC.C）、OS_CPUC.C。

1、OS_CPU.H 文件中定义了硬件相关的基本信息，包括数据类型、代码临界区、堆栈增长方向、OS_TASK_SW( )函数的定义。

(1) 重新定义数据结构。由于不同的处理器有不同的字长，针对 MC9S12XDP512 控制器，μC/OS-II 的移植需要重新来定义一系列的数据结构。

```
typedef unsigned char BOOLEAN;
typedef unsigned char INT8U;
typedef signed char INT8S;
typedef unsigned int INT16U;
typedef signed int INT16S;
typedef unsigned long INT32U;
typedef signed long INT32S;
typedef float FP32;
typedef double FP64;
#define BYTE INT8S
#define UBYTE INT8U
#define WORD INT16S
#define UWORD INT16U
#define LONG INT32S
#define ULONG INT32U
```

(2) MC9S12XDP512 的堆栈是由高地址向低地址方向增长的，所以常量 OS_STK_GROWTH 必须设置为 1。

(3) 进入临界段代码 OS_ENTER_CRITICAL( )和脱离临界段代码 OS_EXIT_CRITICAL( )采用宏定义方法实现，给凡是有临界代码宏调用的函数都增加一个局部变量 cpu_sr，用以保存进入临界段代码前的 CCR 寄存器，即保存中断标志位的状态，这个局部变量在脱离该函数后会自动释放掉，实现代码如下：

```
#if OS_CRITICAL_METHOD == 3
#define OS_ENTER_CRITICAL()  cpu_sr = OS_CPU_SR_Save()
#define OS_EXIT_CRITICAL()   OS_CPU_SR_Restore(cpu_sr)
#endif
```

在汇编文件 OS_CPU_A.S 中定义如下：

```
xdef OS_CPU_SR_Save
xdef   CPU_SR_Save
xdef   OS_CPU_SR_Restore
xdef   CPU_SR_Restore

CPU_SR_Save:
OS_CPU_SR_Save:
tfr ccr,b
sei
rtc
CPU_SR_Restore:
OS_CPU_SR_Restore:
tfr b,ccr
rtc
```

(4) 在 μC/OS-II 中，OS_TASK_SW( )用来实现任务切换。就绪任务的堆栈初始化应该模拟一次中断发生的样子，堆栈中应该按进栈次序设置好各个寄存器的内容。OS_TASK_SW( )函数模拟一次中断过程，在中断返回的时候进行任务切换。MC9S12XDP512 可以采用软中断指令 SWI 实现任务切换。中断服务程序的入口点必须指向汇编函数 OSCtxSw( )。OS_TASK_SW( )用以下宏定义实现。

```
#define OS_TASK_SW()  __asm swi;
```

2、OS_CPU_A.ASM 中需要改写 4 个函数：OSStarHighRdy( )、OSCtxSw( )、OSIntCtxSw( )和 OSTickISR( )。

(1) OSStartHighRdy( )函数由 OSStart( )函数调用，功能是运行优先级最高的就绪任务。在调用 OSStart( )之前，必须先调用 OSInit( )，并且已经至少创建了

一个任务。为了启动任务，OSStarHighRdy( )首先找到当前就绪的优先级最高的任务（OSStarHighRdy( )中保存有优先级最高任务的任务控制块 TCB 的地址），并从任务的任务控制块(OS_TCB)中找到指向堆栈的指针，然后从堆栈中弹出全部寄存器的内容，运行 RTE 中断返回。由于任务创建时的堆栈结构就是按中断后的堆栈结构初始化的，执行 RTE 指令后就切换到新任务。

OSStarHighRdy( )的代码采用 C 语言中内嵌汇编的方法编写。

```
OSStartHighRdy:
call OSTaskSwHook ；用户可以在任务切换时定义一些动作
ldab  #$01    ；多任务
stab   OSRunning
ldx   OSTCBHighRdy   ；指向当前就绪的优先级最高的任务的任务控制块
lds    0,x    ；将该指针放入S12的SP寄存器中
pula   ；获取PPAGE寄存器的值
staa   PPAGE    ；保存在S12的PPAGE寄存器中
rti   ；运行该任务
```

(2) OSCtxSw( )是一个任务级的任务切换函数，在任务中调用，区别在于在中断程序中调用 OSIntCtxSw( )。可以通过一条软中断指令 SWI 来实现任务切换。软中断向量指向 OSCtxSw( )。在 μC/OS-II 中如果任务调用了某个函数，而该函数的执行结果可能造成系统新任务的调度（例如试图唤醒一个优先级更高的任务），则在函数末尾会调用 OSSched( )；如果 OSSched( )将查找当前就绪的优先级最高的任务，若不是当前任务，则判断是否需要进行任务调度，并找到该任务控制块 OS_TCB 的地址，将该地址拷贝到变量 OSStarHighRdy 中，然后通过宏定义 OS_TASK_SW( )执行软中断进行任务切换。在此过程中，变量 OSTCBCur 始终包含一个指向当前运行任务 OS_TCB 的指针。

```
OSCtxSw:
ldaa   PPAGE ；获取当前PPAGE寄存器的值
psha   ；将当前 PPAGE 寄存器的值压入当前任务堆栈
ldy    OSTCBCur ；OSTCBCur->OSTCBStkPtr = Stack Pointer
sts    0,y
call   OSTaskSwHook ；调用用户任务切换函数 OSTaskSwHook( )
ldx    OSTCBHighRdy ；OSTCBCur  = OSTCBHighRdy
stx    OSTCBCur
ldab   OSPrioHighRdy  ；  OSPrioCur = OSPrioHighRdy
stab   OSPrioCur
lds    0,x    ；将该指针放入S12的SP寄存器中
```

```
pula    ；获取PPAGE寄存器的值
staa    PPAGE    ；保存在S12的PPAGE寄存器中
rti    ；运行该任务
```

(3) OSTickISR( )函数

在 μC/OS-II 中，当调用 OSStart( )启动多任务环境后，时钟中断的使用是非常重要的。在时钟中断程序中负责处理所有与定时相关的工作，如任务的延时、等待操作等等。在时钟中断中将查询处于等待状态的任务，判断是否延时结束，否则将重新进行任务调度。可以使用硬件定时器来实现。

和 μC/OS-II 其他中断服务程序一样，OSTickISR( )首先在被中断任务堆栈中保存 CPU 寄存器的值，然后调用 OSIntEnter( )。μC/OS-II 要求在中断服务程序开头调用 OSIntEnter( )，其作用是将记录中断嵌套层数的全局变量 OSIntNesting 加 1。如果不调用 OSIntEnter( )直接将 OSIntNesting 加 1 也是可以的。随后，OSTickISR( )调用 OSTimeTick( )，检查所有处于延时等待状态的任务，判断是否有延时结束就绪的任务。在 OSTickISR( )的最后调用 OSIntExit( )，如果在中断中（或其他嵌套的中断）有更高优先级的任务就绪，并且当前中断为中断嵌套的最后一层，OSIntExit( )将进行任务调度。

```
OSTickISR:
ldaa    PPAGE ；获取当前PPAGE寄存器的值
psha    ；    将当前 PPAGE 寄存器的值压入当前任务堆栈
inc     OSIntNesting
ldab OSIntNesting    ；时钟节拍中断服务子程序检查 是否 OSIntNesting=1
cmpb    #$01
bne     OSTickISR1
ldy     OSTCBCur    ；OSTCBCur->OSTCBStkPtr = Stack Pointer
sts     0,y
OSTickISR1:
call    OSTickISR_Handler ；调用C语言程序 OSTickISR_Handler( )
cli
call    OSIntExit
pula    ；获取PPAGE寄存器的值
staa    PPAGE    ；保存在S12的PPAGE寄存器中
rti
```

3、OS_CPUC.C 文件中需要定义 6 个函数：OSTaskStkInit( )、OSTaskCreateHook( )、OSTaskDelHook( )、OSTaskStatHook( )、OSTaskSwHook( )、OSTimeTickHook( )。

实际需要定义的只有 OSTaskStkInit( )、其他 5 个函数需要声明但不一定有实际内容。要使用这些函数，需要将 OS_CFG.h 中的#define OS_CPU_HOOKS_EN 1 ，设为 0 表示不使用这些函数。

唯一必要的函数是 OSTaskStkInit( )，其它 5 个函数是扩展的关联函数，可以只是声明，当用户需要扩展时才需要编写。OSTaskStkInit( )由任务创建函数 OSTaskCreate( )调用，用来初始化任务的堆栈。初始状态的堆栈模拟发生一次中断后的堆栈结构，按照中断后的进栈次序预留各个寄存器存储空间；而中断返回地址指向任务代码的起始地址。当调用 OStaskCreate( )创建新任务时，需要传递的参数有任务代码的起始地址、参数指针(pdata)、任务堆栈顶端的地址、任务的优先级。堆栈初始化工作结束后，OSTaskStkInit( )返回新的堆栈栈顶指针，OSTaskCreate( )将指针保存在任务的 OS_TCB 中。

4、用户应用程序的编写

建立两个任务，分别打印输出不同的语句。打印输出"运行任务 1"和"运行任务 2"。

```
void main (void)
{
    INT8U   err;
OSInit();  //操作系统初始化
OSTaskCreate(AppTaskCreate  ,(void*)0,&AppStartTaskStk[TASK_STK_SIZE-
1],TASK_START_PRIO);  //创建任务
OSStart();  //运行操作系统
}
static  void  AppTaskCreate (void)
{
    INT8U   err;
    (void)p_arg;
    Hardware_Init();       //硬件初始化，包括时钟、串口初始化
   OSTaskCreate(AppTask1,(void
*)0,&AppTask1Stk[TASK_STK_SIZE-1],TASK_1_PRIO);
    OSTaskCreate(AppTask2,(void
*)0,&AppTask2Stk[TASK_STK_SIZE-1],TASK_2_PRIO);
    EventSem1= OSSemCreate(1);建立信号量，初始计数值为 1
    EventSem2= OSSemCreate(1);建立信号量，初始计数值为 1

}
```

```
static void   AppTask1 (void *p_arg)
{
    INT8U err;
    (void)p_arg;
     p_arg=p_arg;
    while(1)
    {
       OSSemPend(EventSem1, 0, &err);
       Printf("运行任务 1");//执行任务 1 打印输出语句
       OSSemPost(EventSem2);//发出信号量，执行任务 2
    }

}
static void   AppTask2 (void *p_arg)
{
    INT8U err;
    (void)p_arg;
    p_arg=p_arg;
    while (1)
    {
        OSSemPend(EventSem2, 0, &err);
         Printf("运行任务 2") ;//执行任务 2 打印输出语句
         OSSemPost(EventSem1);执行任务 1
    }
}
```

以上移植工作完成后，内核便可以在 MC9S12XDP512 上运行，为移植 CANopen 从站提供了软件开发平台，从而完成应用的目的。

5.3 从节点代码的移植

移植到 μC/OS-II 操作系统上的协议栈运行的流程图如图 5-3 所示。如前文所讲，没有移植到 μC/OS-II 操作系统上的协议栈是一个状态机，要将协议栈移植到操作系统上，就需要把这个状态机封装成一个任务。

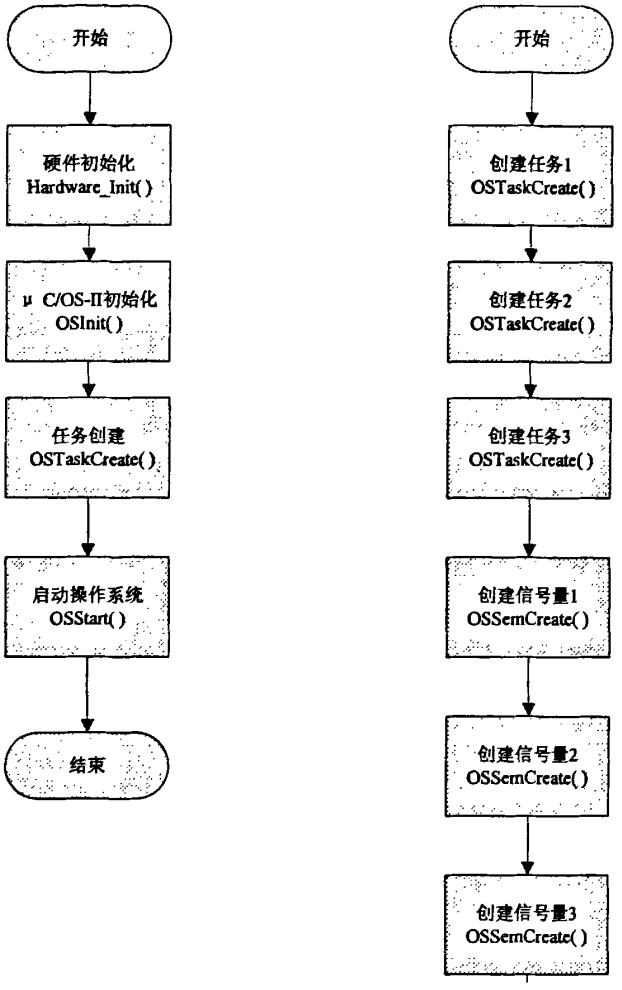


图 5-3 CANopen 从站在 μ C/OS-II 上的运行流程图

Figure 5-3 Process of CANopen Slave Node Running on μ C/OS-II

5.3.1 主函数的编写

将从站代码移植到操作系统上需要重新定义一个主函数，包括操作系统初始化、创建任务和启动任务。

```
void main (void)
{
    INT8U   err;
    Hardware_Init();    //相关硬件的初始化
    Printf("进入 ucos-II 操作系统!");  //打印输出调试语句
    OSInit();  //操作系统初始化
    OSTaskCreate(AppTask1,(void
```

```
*)0,&AppTask1Stk[TASK_STK_SIZE-1],TASK_START_PRIO);//创建任务
        OSStart( ); //启动任务
    }
    Hardware_Init( )函数定义如下：
    void Hardware_Init (void)
  {
     #if PLL_EN > 0
     PLL_Init();
    #endif
     OSTickISR_Init( );
     initTimerClk( );
     initTimer( );
     SCI_INIT( );
}
```

函数中除了初始化与操作系统相关的硬件，还初始化了 CANopen 协议栈用到的定时器，initTimerClk( )和 initTimer( )，注意定时器中断要在启动任务之前初始化。 AppTask1( )是操作系统创建的 CANopen 从站任务函数。SCI_INIT( )函数用来初始化串口模块，用来打印输出调试信息。

5.3.2 任务封装

CANopen 从站的主函数是一个状态调度机，将该主函数封装成一个任务，程序如下：

```
static void   AppTask1 (void *p_arg)
{
INT8U err;
e_nodeState lastState = Unknown_state;
(void)p_arg;
p_arg=p_arg;
while(1) { /* 从节点状态机 */
   switch( getState() ) {
 case Initialisation:
……
 case Pre_operational:
……
 case Operational:
```

```
……
 case Stopped:
……
    }
}
```

5.3.3 中断函数的处理

OS_CPU_A.S 和 OS_CPU.C 是操作系统中与 CPU 有关的代码文件，分别用汇编和 C 语言编写。在 OS_CPU_A.S 中，对 CANopen 从站代码所用到的中断函数进行定义，原代码中使用了 6 个中断函数，定义格式如下：

```
……
    xdef    timer3Hdl_A
    xdef    timer4Hdl_A
    xdef    can4HdlTra_A
    xdef    can4HdlRcv_A
    xdef    can4HdlWup_A
    xdef    can4HdlErr_A
……
……
    xref    timer3Hdl
    xref    timer4Hdl
    xref    can4HdlTra
    xref    can4HdlRcv
    xref    can4HdlWup
    xref    can4HdlErr
……
```

其中 xdef 是在汇编文件里定义的中断函数，xref 是在 C 文件里定义的中断函数，是被调用的函数。作用是通知操作系统，有中断发生，需要进行处理。程序可以仿照操作系统本身自带的中断函数的写法。然后需要对中断向量进行定义，方法与不带操作系统的移植类似，在 P&E_Multilink_CyclonePro_linker.prm 和 Full_Chip_Simulation_linker.prm 这两个文件中定义目标代码的装载地址。

```
VECTOR ADDRESS 0xFFE0 OSTickISR
VECTOR ADDRESS 0xFFE8 timer3Hdl_A
VECTOR ADDRESS 0xFFE6 timer4Hdl_A
VECTOR ADDRESS 0xFF90 can4HdlTra_A
```

VECTOR ADDRESS 0xFF92 can4HdlRcv_A
VECTOR ADDRESS 0xFF96 can4HdlWup_A
VECTOR ADDRESS 0xFF94 can4HdlErr_A
VECTOR ADDRESS 0xFFF6 OSCtxSw

5.4 CANopen 协议栈的验证

将移植在 μC/OS-II 实时操作系统上的从站与 PC 机主节点进行通讯，采用 CAN232 监视报文，并将相应信息从串口打印输出。分别用超级终端和 CAN/232 监控结果。测试平台和测试报告如图 5-4、表 5-1 所示。CAN/232 监控结果见附录。

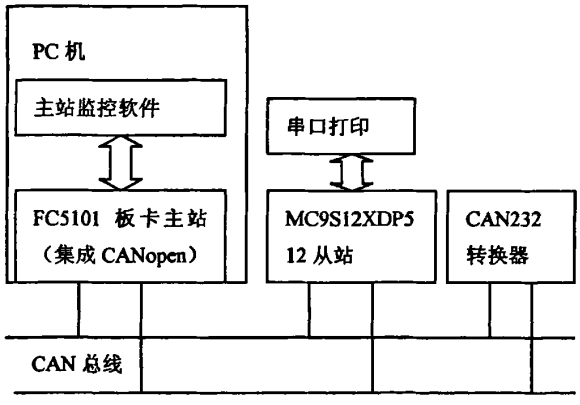


图 5-4 CANopen 从站验证结构图
Fig5-4 CANopen Communication Structure

表 5-1 CANopen 从站验证结果

Table5-1 CANopen Communication Result

功能	描述	结果
SDO	基本的 SDO 加速传输，更改对象字典。	完成，工作正常，
NMT 状态转换	通过 NMT 命令，主站控制从站的状态转换。	完成，工作正常。
心跳报文	从站定时向网络发送心跳报文，对象字典 1017h 可读可写。	完成，工作正常。
节点保护机制	主节点定时通过远程帧请求节点的运行状态。	完成，工作正常。
接收 PDO	从节点接收主站发送的 PDO，并更新相应对象字典数据区。	完成，工作正常。
发送 PDO	从节点至少具有事件触发类型，同步类型和远程帧触发三种类型的发送。	可以动态更改 COB-ID，无法更改传输类型。
接收同步报文	根据同步信息发送 PDO。	收到同步报文后遍历所有的 PDO，找出同步 PDO。

5.5 本章小结

本章首先提出了基于实时操作系统的协议栈整体设计思路，并对嵌入式实时操作系统 μC/OS-II 及其在 MC9S12XDP512 上的移植进行介绍。然后将 CANopen 从站移植到 μC/OS-II 实时操作系统，接着对本课题设计的 CANopen 节点作出验证。验证结果表明，基于 μC/OS-II 实时操作系统的 CANopen 从节点通信正常，CANopen 从节点通信设计成功。

第 6 章 基于 CANopen 通讯的控制系统

本课题拟设计一个数据采集及过程控制节点，该节点采集变送器输入的标准模拟量信号，通过软件控制算法，计算出控制标准模拟量信号传送给驱动器和执行器。整个系统具备 CAN 通讯接口功能，通讯采用 CANopen 协议。

6.1 被控过程对象

被控对象由过程控制实验装置 AE200A 型过程控制实验系统提供，包括流量、液位、压力等参数。其检测信号、控制信号及被控信号均采用 ICE 标准，即电压 1-5V，电流 4-20mA。实验系统供电要求：三相 380 伏交流电，外形尺寸：18501450900mm，重量：300Kg。参数控制系统控制对象描述如表 6-1 所示，控制回路结构如表 6-2 所示。

表 6-1 控制对象描述

Table6-1 The Description of Control Object

检测装置	控制对象
扩散硅压力液位传感器	上、下水箱液位
扩散硅压力液位传感器	小流量水泵压力
电磁流量计	小流量泵动力支路流量

表 6-2 控制回路结构

Table6-2 The Structure of Control Loop

检测装置	输入信号	控制器	D/A	输出信号	执行器
液位传感器	4-20mA/1-5V	AD模块	PWM波用作D/A	4-20mA/1-5V	电动调节阀
电磁流量计	4-20mA/1-5V	AD模块	PWM波用作D/A	4-20mA	电动调节阀

6.2 控制系统的结构

系统通信采用 CANopen 通信，采用先进的 CAN 总线对过程参数进行全面的实时控制和管理。系统结构如图 6-1 所示。系统包括一个主站和一个或多个从站，主站在网路中负责监控整个网络中的过程参数并负责在上位机显示；向从站

发送给定值，进行远程控制；实现网络配置。从站负责液位、压力和流量的控制以及向主节点实时传输过程参数的值。

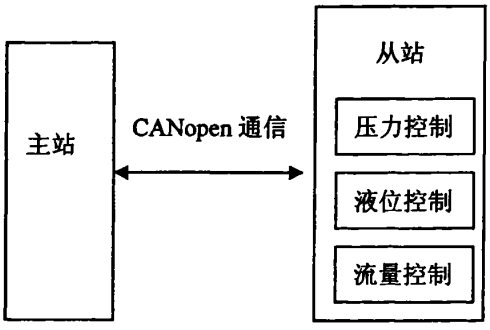


图 6-1 测控系统结构

Figure6-1 Structure of measurement and control system

控制器框图如图 6-2 所示。控制器采用 MC9S12XDP512，输入输出信号均为标准信号。

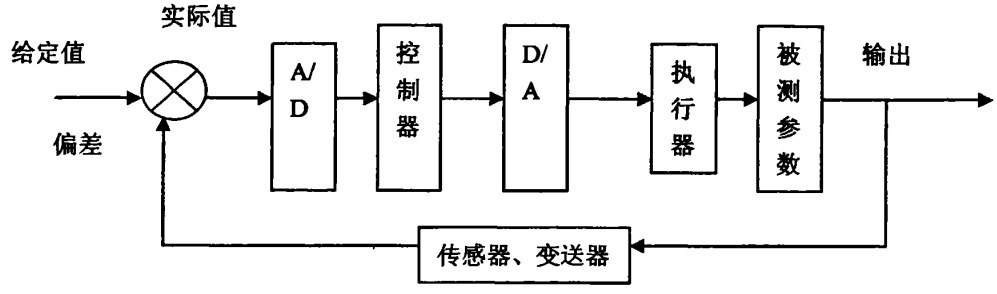


图 6-2 多参数控制系统

Fig6-2 The Multi-Parameters Control System

6.3 PID 控制器的设计

程序由主函数、其他功能函数和一个定时中断组成。主函数中首先进行系统初始化，包括 AD 和 PWM 的初始化。主函数的最后是一个死循环，用来不断检测变量并且输出控制量。利用 MC9S12XDP512 的 Timer overflow 中断来进行定时，每间隔 1 秒把一个 bool 标志位置 1。当 while(1)死循环检测到 bool 值为 1 的时候，便执行 AD 采集和 PID 算法。以下设计均以液位控制算法为例，流量、压力的设计思想与之类似。

由于 PID 算法要知道当前目标值（设定值）、当前测量值（用于计算当前误差——微分）、上一次测量值（用于计算变化值——比例）、前第二次测量值（用于计算变化率——微分），所以要求程序具有存储历史数据的功能。

为提高程序可扩展性、灵活性和可调试性，设置了长度为 10 的历史数据记录数组。每次采样之前都会把最旧的数据更新掉，最新的数据放进来。就像是一个长度固定的队列，该队列在不断更新中，维护一个长度为 10 的最新数据组。也就是说，从当前 0 时刻开始，数组的第一个位置存放的是 0 时刻的数据，第二个位置是-1.5 时刻的数据，依次类推，第 n 个位置存放的就是-(n-1)*1s 时刻的数据。这样，定时器每到 1 秒，PID 算法和采样就执行一次。PID 的离散增量公式为：

$$pid = kp*(En(0)-En(-1))+k1*ki*En(0)+kd*(En(0)-2*En(-1)+En(-2))$$

int En(int n)为一个自定义函数，用来计算第 n 个历史值和设定值的差。PID 控制函数仅仅是一个公式，通过增量的方式发送给 PWMDTY7 寄存器。液位控制函数设置为 PIDControl( )，具体流程图如图 6-3 所示。

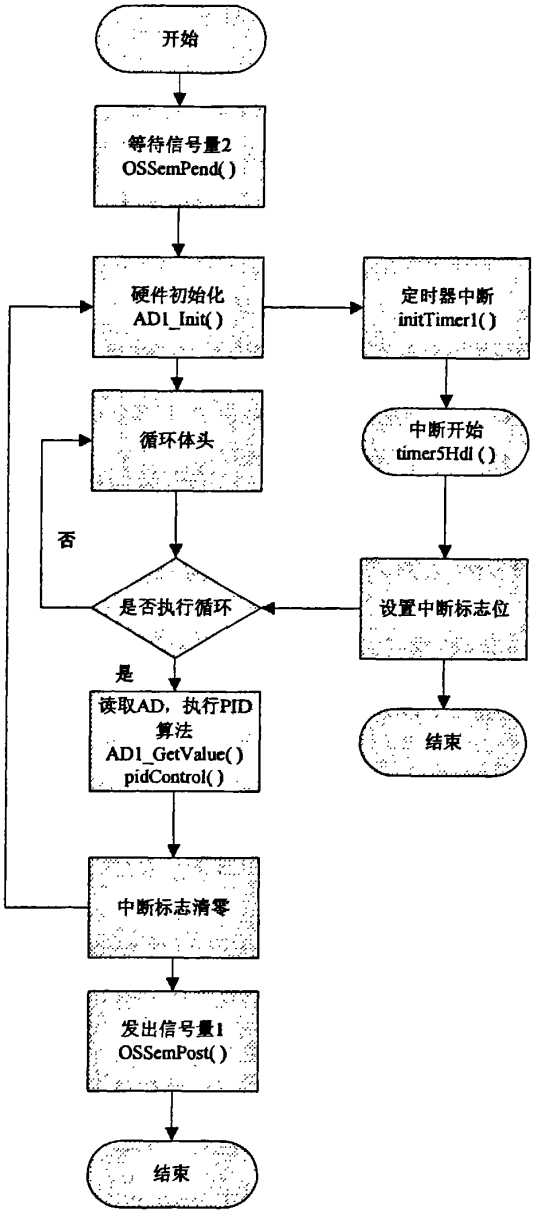


图 6-3 PID 控制流程图

Fig6-3 The Process of PID Control

6.4 CANopen 通讯的实现

6.4.1 从站对象字典的定义

智能仪表的通讯采用 CANopen 的通讯方式，需要传到总线上的数据包括传感器测量值和控制器输出的控制信号[35]，通讯和应用之间的接口是对象字典，它

存储了所有与应用有关的数据，设备模型如图 6-4 所示。

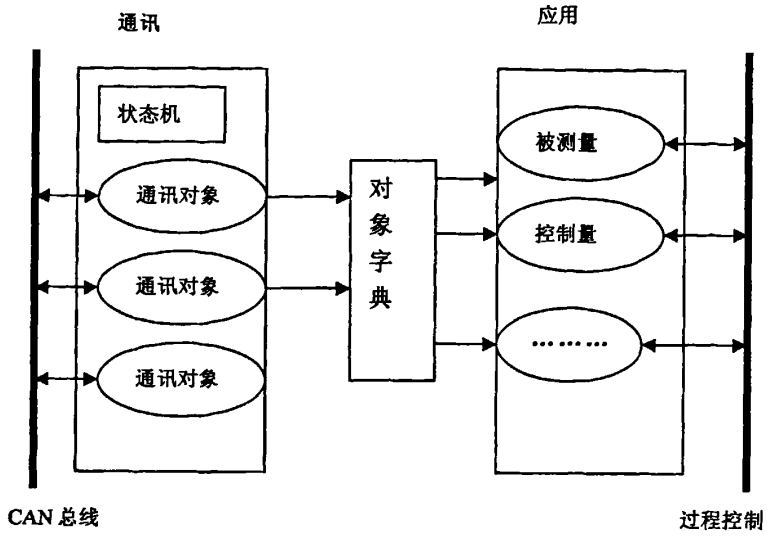


图 6-4 设备通讯模型

Fig6-4 The Model of Device Communication

液位传感器输出 4-20mA 电流信号，通过 250 欧的电阻转换成 1-5V 电压信号。流量积算仪输出 4-20mA 电流信号，通过 250 欧的电阻转换成 1-5V 电压信号，MC9S12XDP512 通过变频器控制水泵的流量。监测控制的流量包括小流量泵动力支路流量和单相格兰富水泵动力支路流量。根据以上分析该控制系统的对象字典具体定义如表 6-3 所示。

表 6-3 控制系统对象字典分配

Table6-3 The distribution of OD for measurement and control system

6000H	0	子索引个数
	1	上水箱液位
	2	下水箱液位
6001H	0	子索引个数
	1	上水箱液位设定值
	2	下水箱液位设定值
6002H	0	子索引个数
	1	小流量泵动力支路流量
	2	单相格兰富水泵动力支路流量
6003H	0	子索引个数
	1	小流量泵动力支路流量设定值
	2	单相格兰富水泵动力支路流量设定值

从站 PDO 通信和映射参数的定义如表 6-4、表 6-5、表 6-6、表 6-7 所示。

表 6-4 TPDO1

Table6-4 TPDO1

索引	子索引	值
1800H	01	181H
1800H	02	255
1A00H	01	60000116H
1A00H	02	60000216H

表 6-5 TPDO2

Table6-5 TPDO2

索引	子索引	值
1800H	01	281H
1800H	02	255
1A00H	01	60020120H
1A00H	02	60020220H

表 6-6 RPDO1

Table6-6 RPDO1

索引	子索引	值
1400H	01	201H
1400H	02	255
1600H	01	60010120H
1600H	02	60010220H

表 6-7 RPDO2

Table6-7 RPDO2

索引	子索引	值
1400H	01	301H
1400H	02	255
1600H	01	60030120H
1600H	02	60030220H

发送 TPDO1 时，将索引为 6000H 子索引为 1 和 2 的数据映射到 TPDO1 中，向主节点发送上水箱液位和下水箱液位。

接收到 RPDO1 时，将其中的数据映射到索引为 6001H 子索引为 1 和 2 的对

象字典中，保存主节点发送的上水箱液位设定值和下水箱液位设定值。

发送 TPDO2 时，将索引为 6002H 子索引为 1 和 2 的数据映射到 TPDO2 中，向主节点节点发送小流量泵动力支路流量和单相格兰富水泵动力支路流量。

接收到 RPDO2 时，将其中的数据映射到索引为 6003H 子索引为 1 和 2 的对象字典中，保存主节点发送的小流量泵动力支路流量设定值和单相格兰富水泵动力支路流量设定值。从节点对象字典[27][28][38][39]其它索引定义如表 6-8 所示。

表 6-8 从节点对象字典定义

Table6-8 The Definition of Slave Node Object Dictionary

索引	描述	初始值
1000H	设备类型	0
1017H	生产者心跳报文时间	
00H	从节点产生心跳报文的间隔	0
1018H	设备信息	
01H	制造商 ID	0x1234
02H	产品代码	0x5678
03H	版本号	0
04H	序列号	0
1280H	客户 SDO 参数	
00H	子索引个数	3
01H	发送 SDO 的 COB-ID（客户到服务器）	601H
02H	接受 SDO 的 COB-ID（服务器到客户）	581H
03H	SDO 服务器的节点号	01H

6.4.2 构建 PID 控制任务

操作系统 μC/OS-II 中任务之间的通信采用信号量的方法，首先需将 OS_CFG.H 中将配置常数置为 1，这样才能支持信号量。因为只要 μC/OS-II 支持信号量，就必须有 OSSemCreate( )，OSSemPend ( )以及 OSSemPost( )函数，所以它们不能被单独禁止[21]。

OSSemCreate( )用于建立信号量，并对信号量赋予初始计数值，该初始值为 0-65535 的一个数，OSSemPend ( )是任务等待一个信号量，OSSemPost( )是任务发出一个信号量，在本课题中，建立两个任务，任务 1 是 CANopen 协议栈任务，任务 2 是多参数控制系统任务，每个任务拥有一个信号量，用以下语句实现信号量的建立。

```
EventSem1= OSSemCreate(1);
```

EventSem2= OSSemCreate(1);

以液位控制为例，在运行 CANopen 任务之初，运行函数 OSSemPend(EventSem1, 0, &err)，即为 CANopen 任务在等待液位控制任务发出的信号量 1，在 CANopen 任务结束时运行 OSSemPost(EventSem2)函数，即为 CANopen 任务向液位控制任务发出信号量 2。同理，在液位控制任务之初，运行函数 OSSemPend(EventSem2, 0, &err)，即为液位控制任务在等待 CANopen 任务发出的信号量 2，在液位控制任务结束时运行 OSSemPost(EventSem1)函数，即为液位控制任务向 CANopen 任务发出信号量 1。

6.4.3 通讯数据的传输

过程控制中的各类参数，由 MC9S12XDP512 的 A/D 模块采集到主控制器中，转换结果保存在寄存器中，从站的协议栈对象字典数据区的索引定义采取如下方式，定义一个结构体：

```
typedef struct td_subindex
{
    enum e_accessAttribute   bAccessType;// 数据的访问类型
    UNS8    bDataType; // 定义该索引包含什么数据
    UNS8    size;        // 数据大小
    void*   pObject;    // 指向具体数据的指针
} subindex;
```

定义Index6000[]为该类型的结构体数组，

```
subindex Index6000[] =
{
   { RW, uint32, sizeof (UNS32), (void*)&adValue0},
   { RW, uint32, sizeof (UNS32), (void*)&adValue1},
   { RW, uint32, sizeof (UNS32), (void*)&adValue2}
};
```

其中变量adValue0的值在函数AD_GetValue( )中定义，该函数用来获取AD转换结果寄存器(ATD Conversion Result Registers)ATD0DR0H和ATD0DR0L的值，并赋给变量adValue0。

```
void AD_GetValue(void) //读取AD转换结果
{
    INT8U err;
   adValue0=  IO_PORTS_16(ATD0DR0L);// 将 Result  Register 的值赋给
//adValue0
```

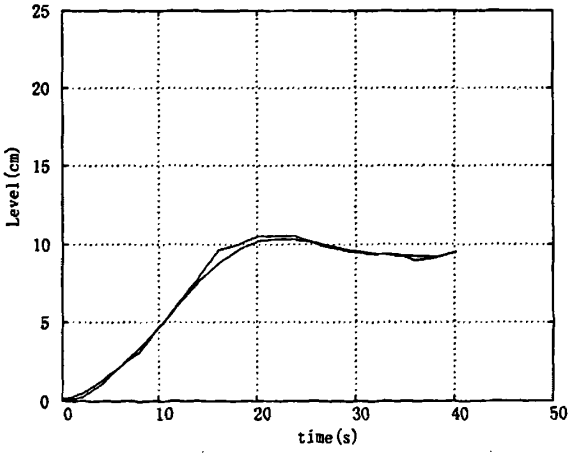
}

将定义数据区对象字典索引项的结构体中的指向数据的指针指向寄存器中的数值，就可以用寄存器中的实时数据来实施更新对象字典的数据区部分。同理，控制信号可以赋值给相应的数据区索引，这便完成了过程控制参数与 CANopen 相结合的步骤。

6.4.4 实验与调试

在实际调试过程中，发现 D/A 电路输出不正确，没有控制效果，经检查发现 PWM 频率过高，由于执行机构不能以非常高的频率动作，必须要求能够正确还原占空比，所以要求 PWM 周期在 1-2 秒左右才能正常工作，降低 PWM 的频率后解决问题。另外，如果 LM358 采用正负 5V 供电，当控制量也就是电动调节阀的开度为 100%时，控制器输出电压为 4.69V，LM358 实际输出只有 3.67V，提高 LM358 的正负供电电压后解决问题。调试过程中还遇到过 PDO 显示不正确的问题，经检查程序，发现是对象字典数据区内的数据类型与 PDO 报文中数据类型不匹配，修改为同类型的数据后问题解决。

首先分别调试操作系统下的控制任务，对 PID 参数进行整定，经过反复试验，整定出被控量的 PID 参数，由 Matlab 绘制的采集数据曲线可以看出，均达到较好效果。液位控制如图 6-5 所示，系统超调量大约在 10%，稳定时间 70S，稳态误差控制结果较为理想。流量控制的结果如图所示，该控制可以使电磁流量计的值稳定在设定值附近，达到比较好的控制效果。



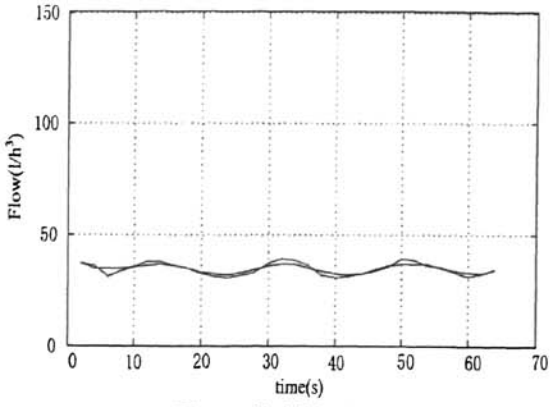


图 6-5 控制效果图

Fig6-5 The Control Result

图 6-6 是 CANopen 通讯串口监控的输出信息的一部分，可以看出，基于事件驱动的 PDO 在 AD 的值发生变化时向主节点发送 PDO 报文，PDO 中包含映射索引 0x1A00 指向的数据区，索引为 6000，子索引为 2 的数据 0XB9。CANopen 能够将 A/D 模块采集的数据成功发送到 CAN 总线上。

图 6-6 CANopen 通信实验

Fig6-6 The Final Result of CANopen

主节点对从节点的监控也通过发送 PDO 来实现。如图 6-7 所示，从节点接到主节点发送的 PDO，用接收到的 PDO 中的数据更新了自己的对象字典。

图 6-7 CANopen 通信实验

Fig6-7 The Final Result of CANopen

图 6-8 是 PDO 采集数据实验监控结果，可以看出，主节点发送状态改变命令之后，从节点进入运行状态，开始发送 PDO，PDO 被配置成为事件触发，只要 AD 采集的数据不为 0，就会发送 PDO，下图的报文是从节点发送 PDO 的过程，可以看出 TPDO1 和 TPDO2 的数据随 AD 的值变化而发送数据有所变化，每隔 1 秒采集两个参数，实现了实时参数采集的功能。

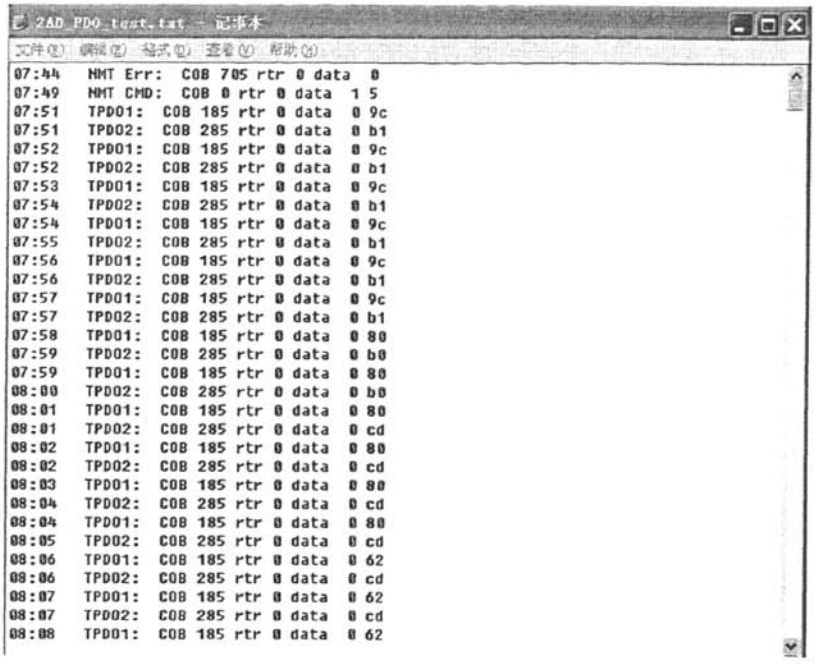


图 6-8 PDO 通信实验

Fig6-8 The Final Result of PDO

图 6-9 是主站的上位机监控界面，主从站之间通过 PDO 传递实时数据，从节点采集到的实时数据通过发送 PDO 发送给主站，在监控界面上打点显示，主站可以通过上位机界面的给定设定值对从站进行控制。

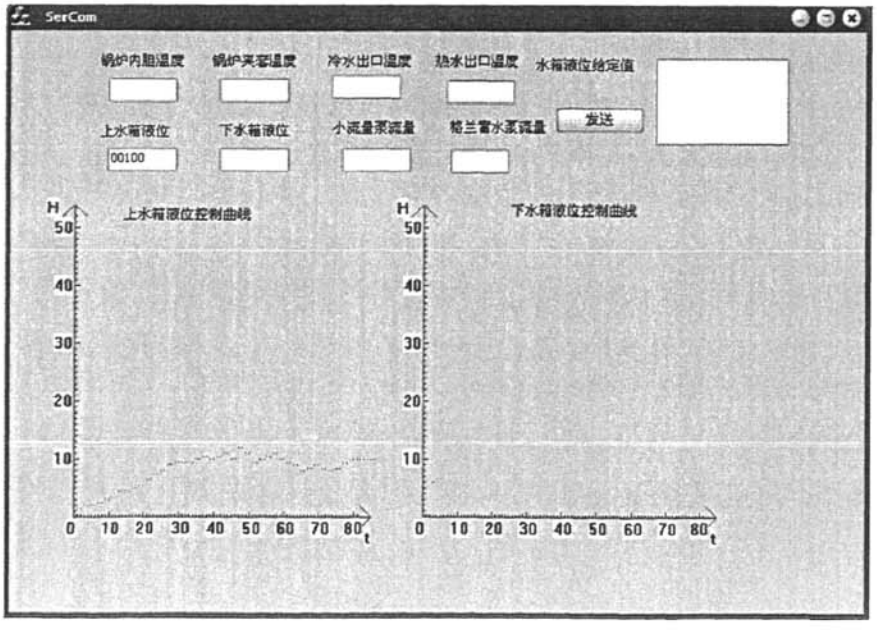


图 6-9 主站监控界面

Fig6-9 The Monitor Interface of Master Node

6.5 本章小结

本章介绍了通过MC9S12XDP512对给定参数进行标准信号的采集，PID控制，并输出标准控制信号，以达到用PID控制器的方法来控制液位、压力、流量。最后将测量数据和控制数据均通过CANopen协议进行传输，利用了CAN总线实时、可靠的优点，得到较满意的效果。

结 论

1、课题的总结

课题按照预先设计的任务，依次完成了 MC9S12XDP512 硬件平台扩展板的设计和调试，接口驱动程序设计，成功移植 μC/OS-II 操作系统并且系统运行稳定。在此基础上，向控制器中移植了 CANopen 从协议栈，实现了节点启动、SDO 通信、PDO 通信、心跳报文、节点保护等功能，构成 CANopen 从节点。

设计并移植了基于 μC/OS-II 操作系统的 CANopen 从协议栈，对从 CANopen 节点进行了深入的验证，通过分析与主节点通信采集到的运行数据，表明本课题设计的 CANopen 从节点运行正常。

在 μC/OS-II 操作系统下建立多参数控制的任务，运用 PID 控制算法，对液位、压力、流量进行控制并将数据实时发送给 CANopen 主节点，完成了 CANopen 通讯在仪表方面的探索性应用。

通过本课题的研究，对 CANopen 协议栈的设计和应用有了一定深度的理解，将来可将此技术应用到混合动力汽车各种 ECU 间的通讯或其他现场总线的相关设计过程中。

2、课题的展望

本课题设计并实现了基于 μC/OS-II 的 CANopen 从节点，并将其应用到智能仪表方面，完成液位、压力、流量的控制以及 CANopen 通讯。CANopen 在国内应用并不算广泛，而基于操作系统的 CANopen 协议栈可以大大提高应用的灵活性和可扩展性，希望本课题可以起到抛砖引玉的作用。后期工作可在本课题的基础上继续完成 CANopen 的应用。在仪表方面，可以改进控制算法，达到更准确、可靠性更高的控制目标。如果将系统利用在混合动力汽车上，可以在 μC/OS-II 操作系统下构建新的任务，例如汽车防抱死(ABS)的控制任务、发动机控制任务等，使不同从节点具有各自不同的控制功能，同时兼备 CANopen 的通讯功能，进而组成 CANopen 网络。

参考文献

1 马秋霞，郇极. CANopen现场总线从设备协议一致性测试系统研究. 制造业自动化. 2005，01

2 CANopen主节点设计，中国单片机公共实验室

3 CANopen 的开源网站：http://canfestival.sourceforge.net/

4 微型 CANopen 网站： http://www.microcanopen.com/

5 饶运涛. 现场总线CAN原理与应用技术. 北京航空航天大学出版社，2003：14～36

6 邬宽明. CAN总线原理和应用系统设计. 北京航空航天大学出版社，1996：5～20

7 邬宽明. CAN总线系统设计中的几个问题. 1998，9：18~20

8 CAN－bus 规范 V2.0 版本. CAN Specification V2.0.BOSCH 公司

9 刘建昌，钱晓龙，冯立. CAN 总线及 Devicenet 现场总线.基础自动化. 2003，8：1~3

10 CANOpen Protocol[EB/OL].http://www.can-cia.de/canopen/Protocol,2002-5.

11 CANopen Communication Profile for Industry System Based on CAL, CiA Draft Standard 301 October 1996.

12 CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02 Date:13 February 2002

13 Olaf Pfeiffer, Andrew Ayre, Christian Kerdel. Embeded Networking with CAN and CANopen[M]. RTC Bookl. 2003

14 任哲. 嵌入式实时操作系统 μC/OS-II 原理及应用. 北京航空航天大学，2005，8：4～13

15 孙瑜，张根宝. 工业自动化仪表与过程控制. 西北工业大学出版社, 2003：

16 Freescale semiconductor,Data Sheet of MC9S12XDP512.Rev.2.15,July 2006

17 Freescale 的官方网站：http://www.freescale.com

18 邵贝贝. 单片机嵌入式应用的在线开发方法. 清华大学出版社，2004：6～24

19 TJA1050 High speed CAN transceiver.1999,Sep 27

20 王宇波. 嵌入式网络控制器的设计与实现. 天津大学硕士学位论文. 2005(1)：24~29

21 Jean J Labrosse 邵贝贝等译. 嵌入式实时操作系统 μC/OS-II（第二版）. 北京航空航天大学出版社，2003，8：1～13

22 宋晓强. CAN bus 高层协议 CANopen 的研究以及在模块化 CAN 控制器上的

实现. 天津大学硕士学位论文. 2004：11~20

23 刘宝坤. 计算机过程控制系统. 机械工业出版社，2001：138~160

24 闫永跃，李庆周，于树新. 智能 PID 控制综述. 可编程控制器与工厂自动化. 2006, 12：9~12

25 W.J.M.Kickert,et.Application of a Fuzzy Controller in a warmwater Plant. Automat.1976,12(4):301~308

26 杨红科. 基于CAN总线的液位模糊控制系统的设计及研究. 南京航空航天大学硕士学位论文. 2005(2)：40~48

27 陈骥. 基于CANOpen高级协议和ED调度算法的电动汽车网络协议研究. 天津大学硕士学位论文. 2003(12)：21~30

28 吕宁. CAN现场总线网络环境通讯协议研究与实现. 西安电子科技大学硕士学位论文. 2004(1)：26~32

29 McLaughlin.R.T.CAN Overview.CANopen Implementation.1997,6Oct:1~27

30 M.Farsi,K.Ratcliff.Manuel Barbosa,"An introduction to CANopen".Computer& Control Engineering Journal,August 1999:161-168

31 Farsi M.,Ratcliff K.,Controlling with CANopen，IEE Review.Volume 44,Issue 5,17 Sept. 1998 Page(s):229 - 231

32 FARSIM.an dB ARBOSAM,CANopen implementation application to industrial networks,Research Studies Press Ltd.ISBN 0-86380-247-82000

33 Hui Guo,Ying Jiang,Application Layer Definition and Analyses of Controller Area Network Bus for Wire Harness Assembly MachineComputational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on Nov. 2006 Page(s):11 - 11

34 Farsi M.,Ratcliff K.,CANopen: the open communications solution,Industrial Electronics,1997.Proceedings of the IEEE International Symposium on 7-11 July 1997 Page(s):112 - 116

35 Farsi M.,Ratcliff K.,CANopen: configure and device testing,Factory Communication Systems,1997. Proceedings.1997 IEEE International Workshop on 1-3 Oct. 1997 Page(s):373~380

36 Hao Jianmin,Guo Kai,Cheng Hong,Ren Na;Design of Micro-oxidation power control system based on LPC2119，Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference onAug. 16 2007-July 18 2007 Page(s):1-849 - 1-853

37 Jae-Ho Lee,Heung-Nam Kim,Implementing priority inheritance semaphore on

uC/OS real-time kernel,Software Technologies for Future Embedded Systems,2003. IEEE Workshop on15-16 May 2003 Page(s):83 - 86

38 Cena.G. Valenzano.A.A protocol for automatic node discovery in CANopen networks Industrial Electronics. IEEE Transactions 2003, 3 June: Volume50 419~430

39 Cena.G Valenzano.A.Efficient polling of devices in CANopen networks. Emerging Technologies and Factory Automation. Proceedings. ETFA '03. IEEE Conference 2003, 16-19 Sept. Volume 1: Page(s):123 - 130

40 Lew Boon Kian.Test cost saving and challenges in the implementation of /spl times/6 and /spl times/8 parallel testing on freescale 16-bit HCS12 microcontroller product family.Electronic Design, Test and Applications, 2006. DELTA 2006. Third IEEE International Workshop on 17-19 Jan. 2006 Page(s):7

附 录

序号 传输方向 时间标识 帧 ID 帧格式 帧类型 数据长度 数据

0x00000000 接收 0x0023a701 0x00000705 数据帧 标准帧 0x01 00

0x00000001 接收 0x00240780 0x00000000 数据帧 标准帧 0x02 82 05

0x00000002 接收 0x002409f4 0x00000705 数据帧 标准帧 0x01 00

0x00000003 接收 0x00269ecb 0x00000605 数据帧 标准帧 0x08 40 00 10 00 00 00 00 00

0x00000004 接收 0x0026a20f 0x00000585 数据帧 标准帧 0x08 43 00 10 00 00 00 00 00

0x00000005 接收 0x0027022f 0x00000605 数据帧 标准帧 0x08 40 18 10 01 00 00 00 00

0x00000006 接收 0x00270571 0x00000585 数据帧 标准帧 0x08 43 18 10 01 34 12 00 00

0x00000007 接收 0 0x002761ca 0x00000605 数据帧 标准帧 0x08 40 18 10 02 00 00 00 00

0x00000008 接收 0x00276512 0x00000585 数据帧 标准帧 0x08 43 18 10 02 78 56 00 00

0x00000009 接收 0x0027c166 0x00000605 数据帧 标准帧 0x08 40 18 10 03 00 00 00 00

0x0000000a 接收 0x0027c4b4 0x00000585 数据帧 标准帧 0x08 43 18 10 03 64 13 00 00

0x0000000b 接收 0x00282102 0x00000605 数据帧 标准帧 0x08 40 18 10 04 00 00 00 00

0x0000000c 接收 0x00282456 0x00000585 数据帧 标准帧 0x08 43 18 10 04 64 79 00 00

0x0000000d 接收 0x0028809e 0x00000605 数据帧 标准帧 0x08 40 17 10 00 00 00 00 00

0x0000000e 接收 0x002883c6 0x00000585 数据帧 标准帧 0x08 4b 17 10 00 00 00 00 00

0x0000000f 接收 0x0028dfee 0x00000605 数据帧 标准帧 0x08 2b 17 10 00 f4 01 00 00

0x00000010 接收 0x0028e2f2 0x00000585 数据帧 标准帧 0x08 60 17 10 00 00 00 00 00

0x00000011 接收 0x0028e31f 0x00000705 数据帧 标准帧 0x01 7f
0x00000012 接收 0x0028e510 0x00000705 数据帧 标准帧 0x01 7f
0x00000013 接收 0x0028e700 0x00000705 数据帧 标准帧 0x01 7f
.....
0x00000116 接收 0x002adaf9 0x00000605 数据帧 标准帧 0x08 40 00 18 01 00 00 00 00
0x00000117 接收 0x002ade32 0x00000585 数据帧 标准帧 0x08 43 00 18 01 85 01 00 00
0x00000118 接收 0x002ade5f 0x00000705 数据帧 标准帧 0x01 7f
0x00000119 接收 0x002ae04 0x00000705 数据帧 标准帧 0x01 7f
0x0000011a 接收 0x002ae23f 0x00000705 数据帧 标准帧 0x01 7f
.....
0x0000018a 接收 0x002bba17 0x00000605 数据帧 标准帧 0x08 23 00 18 01 85 01 00 80
0x0000018b 接收 0x002bbd1e 0x00000585 数据帧 标准帧 0x08 60 00 18 01 00 00 00 00
0x0000018c 接收 0x002bbd4b 0x00000705 数据帧 标准帧 0x01 7f
0x0000018d 接收 0x002bbf3b 0x00000705 数据帧 标准帧 0x01 7f
0x0000018e 接收 0x002bc12b 0x00000705 数据帧 标准帧 0x01 7f
.....
0x000001f6 接收 0x002c8915 0x00000605 数据帧 标准帧 0x08 40 00 18 02 00 00 00 00
0x000001f7 接收 0x002c8c44 0x00000585 数据帧 标准帧 0x08 4f 00 18 02 00 00 00 00
0x000001f8 接收 0x002c8c71 0x00000705 数据帧 标准帧 0x01 7f
0x000001f9 接收 0x002c8e61 0x00000705 数据帧 标准帧 0x01 7f
0x000001fa 接收 0x002c9051 0x00000705 数据帧 标准帧 0x01 7f
.....
0x00000263 接收 0x002d5a54 0x00000605 数据帧 标准帧 0x08 2f 00 18 02 ff 00 00 00
0x00000264 接收 0x002d5d40 0x00000585 数据帧 标准帧 0x08 60 00 18 02 00 00 00 00
.....
0x000002d4 接收 0x002e33a3 0x00000605 数据帧 标准帧 0x08 23 00 18 01 85 01 00 00

0x000002d5 接收 0x002e3687 0x00000585 数据帧 标准帧 0x08 60 00 18 01 00 00 00 00

.....

0x000003a2 接收 0x002fc09f 0x00000605 数据帧 标准帧 0x08 40 00 14 01 00 00 00 00

0x000003a3 接收 0x002fc3cc 0x00000585 数据帧 标准帧 0x08 43 00 14 01 05 02 00 00

.....

0x00000409 接收 0x00308741 0x00000605 数据帧 标准帧 0x08 23 00 14 01 05 02 00 80

0x0000040a 接收 0x00308a42 0x00000585 数据帧 标准帧 0x08 60 00 14 01 00 00 00 00

.....

0x00000476 接收 0x00315835 0x00000605 数据帧 标准帧 0x08 40 00 14 02 00 00 00 00

.....

0x00000477 接收 0x00315b86 0x00000585 数据帧 标准帧 0x08 4f 00 14 02 00 00 00 00

.....

0x000004dd 接收 0x00321d89 0x00000605 数据帧 标准帧 0x08 2f 00 14 02 ff 00 00 00

0x000004de 接收 0x00322089 0x00000585 数据帧 标准帧 0x08 60 00 14 02 00 00 00 00

.....

0x00000543 接收 0x0032e103 0x00000605 数据帧 标准帧 0x08 23 00 14 01 05 02 00 00

0x00000544 接收 0x0032e3f8 0x00000585 数据帧 标准帧 0x08 60 00 14 01 00 00 00 00

.....

0x000005be 接收 0x0033ce27 0x00000000 数据帧 标准帧 0x02 01 00

0x000005bf 接收 0x0033cfe4 0x00000705 数据帧 标准帧 0x01 05

0x000005c0 接收 0x0033d1ed 0x00000705 数据帧 标准帧 0x01 05

.....

0x0000068a 接收 0x00356f19 0x00000080 数据帧 标准帧 0x00

0x0000068b 接收 0x00357245 0x00000705 数据帧 标准帧 0x01 05

0x0000068c 接收 0x0035744b 0x00000705 数据帧 标准帧 0x01 05
0x0000068d 接收 0x00357638 0x00000705 数据帧 标准帧 0x01 05
0x0000068e 接收 0x0035783e 0x00000705 数据帧 标准帧 0x01 05

攻读硕士学位期间所发表的学术论文

1 徐喆，张卓，闫士珍. 基于μC/OS-II 的 CANopen 从节点的实现. 计算机系统应用. 已录用

2 徐喆，张卓，闫士珍. CANopen 网络主节点对象字典的定义及其数据结构. 计算机应用研究. 2008，9

3 徐喆，闫士珍，宋威，张卓. 基于 MC9S12DP512 和μC/OS-II 的 CANopen 主站开发. 计算机工程与科学. 2009，4

致 谢

本论文的工作是在导师徐喆副教授的悉心指导下完成的，导师严谨的治学态度和科学的工作方法给了我极大的帮助和影响。衷心感谢三年来她对我的关心和指导。徐喆老师在学习上严格要求，生活上关怀入微并且在研究和学习方法上给予了正确的引导，老师兢兢业业的治学态度，认真负责的工作作风和宽宏大度的为人处世的方式都将对我以后的工作和生活产生深远影响。

还要感谢电子信息与控制工程学院自动化专业检测方向的所有老师，段建民老师的课题给予了我们研究汽车电子的机会，吴晴老师在我的开题阶段给予了我许多帮助和支持，陈双叶、余春暄老师在硬件设计方面给予我耐心指导，綦慧、许家群、于涌川老师在每次中期汇报都给予我中肯的意见和建议，在此真诚感谢各位老师。

同时还要感谢同课题组宋威、闫士珍、张明杰同学对我的帮助，感谢同实验室的焦圣伟、董石峰、雷杨杰等同学在我日常生活和学习中给予的建议和帮助，感谢陈静同学在最后论文格式调整方面给予的热心帮助，感谢各位朋友给与我的鼓励和信任！

课题的研究过程中浏览过大量的自由软件社区论坛，论坛的贡献者们曾帮助我解决了许多难题，感谢这群贡献者。

另外特别要感谢父母，在我成长和求学的过程中一如既往的关心和支持。

最后衷心感谢在百忙中为论文审阅而付出辛勤劳动的各位专家学者。