

1766931

分类号: TP23

密级:

天津理工大学研究生学位论文

# 基于嵌入式系统的 CANopen 协议分析研究

(申请硕士学位)

学科专业: 控制理论与控制工程

研究方向: 计算机控制与管理

作者姓名: 王 峰

指导教师: 陈在平 教授

2010 年 1 月



**Y1750635**

**Thesis Submitted to Tianjin University of Technology for  
the Master's Degree**

# **Analysis and Study of CANopen Protocol Based on Embedded System**


By  
**Wang Feng**

Supervisor  
**Prof. Chen Zaiping**

**January, 2010**

## 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和取得的研究成果，除了文中特别加以标注和致谢之处外，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 天津理工大学 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了谢意。


学位论文作者签名：

签字日期：2010年2月1日

## 学位论文版权使用授权书

本学位论文作者完全了解 天津理工大学 有关保留、使用学位论文的规定。特授权 天津理工大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编，以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复本和电子文件。

(保密的学位论文在解密后适用本授权说明)

学位论文作者签名：

导师签名：

签字日期：2010年2月1日

签字日期：2010年2月1日

## 摘 要

CANopen 作为一种开放的、标准化的 CAN 总线应用层协议，在国内外的应用已经深入到了各个领域，并且还在进一步发展。但是，相对于广阔的应用范围，CANopen 协议的研究和开发在我国还没有得到足够的重视。目前基于该协议规范的通信接口关键技术仍然被国外若干公司所垄断，国内有关 CANopen 协议的分析与研究仍然处于起步阶段，具有非常大的发展空间。

从这一实际情况出发，本文首先分析了 CANopen 总线在国内外的研究现状，介绍了本文的研究内容与意义。其次对 CANopen 的协议规范进行了阐述，分析了 CANopen 的协议结构、对象字典和通信对象，并详细说明了本文使用的开源协议栈 CanFestival。再次，本文对嵌入式实时操作系统  $\mu\text{C}/\text{OS-II}$  的原理和特点进行了阐述。在此基础上，本文搭建了协议研究的目标平台，并详细介绍了基于嵌入式系统的 CANopen 通信接口在此硬件平台上的实现方法。之后，论文提出了一个有效的实验方案，并按照此方案对本文开发的 CANopen 通信接口进行了实验，以证明研究的有效性。最后对论文研究的内容进行了总结和展望。

本文通过借鉴国外的先进经验，利用开源代码在目标平台上对 CANopen 协议栈进行了研究，并提出了 CANopen 协议规范的通信接口开发和实验的有效方案，从而为今后 CANopen 协议从节点的开发研究奠定了坚实的基础。这一方案对于缩短开发周期，降低开发难度，节约开发成本，加速 CANopen 在国内的应用与推广具有重要的现实意义。

**关键词：**CANopen  $\mu\text{C}/\text{OS-II}$  嵌入式系统 现场总线 协议栈 单片机

## Abstract

As an open, standardized application layer protocol of CAN-bus, CANopen is widely applied into the various fields at home and abroad, and is still in the further development. But compare with the range of application, the analysis and research of the CANopen protocol product have not yet been given sufficient attention in China, the most important technology of the protocol interface is still monopolized by foreign companies and domestic on CANopen protocol analysis and research is still in its infancy, So there is a great space for the development of CANopen product.

This paper firstly analyzes the research status of CANopen at home and abroad, and introduces the content and meaning of this paper. Secondly, the CANopen protocol specifications are described, analyzed the CANopen protocol structure, object dictionary and communication objects are also introduced in detail, and the CanFestival, which is an open-source CANopen protocol stack was recommend particularly. In the third place,  $\mu\text{C}/\text{OS-II}$ , a kind of embedded operating system is analysed. On this basis, this paper presented the target platform, and realized of the CANopen communication interface based on  $\mu\text{C}/\text{OS-II}$  with the certain hardware. Then the paper presented the method of testing, and tested the communication interface with this method, in order to improve the effect of design. At last, there are summary and prospect about the research of content of the paper.

The paper realizes CANopen protocol by transplants a foreign open-source protocol stack, and make it runs as a task of embedded operating system. This paper presents an effective program of developing the communication interface of CANopen, and laies the foundation for the development of slave node. This program have very important practical significance for shortening the development cycle, reducing the development difficulty, saving the development costs and speeding up the CANopen application in China.

**Key words:** CANopen,  $\mu\text{C}/\text{OS-II}$ , Embedded System, Fieldbus, Protocol Stack, MCU

# 目 录

<b>Abstract</b> .....	v
<b>第一章 绪论</b> .....	1
1.1 CANopen 简介 .....	1
1.2 国内外研究现状分析 .....	2
1.3 课题研究目的与意义 .....	2
1.4 课题研究内容 .....	3
1.5 本章小结 .....	3
<b>第二章 CANopen 协议分析</b> .....	4
2.1 CANopen 协议简介 .....	4
2.1.1 网络参考模型 .....	4
2.1.2 设备模型 .....	5
2.2 CANopen 对象字典 .....	5
2.3 CANopen 通信对象 .....	6
2.3.1 SDO .....	6
2.3.2 PDO .....	7
2.3.3 NMT .....	8
2.3.4 特殊功能对象 .....	10
2.4 典型开源 CANopen 协议栈 .....	11
2.4.1 CanFestival .....	11
2.4.2 MicroCANopen .....	12
2.4.3 CANopenNode .....	12
2.5 开源协议栈 CanFestival .....	12
2.5.1 CanFestival 的特点 .....	12
2.5.2 CanFestival 的结构 .....	13
2.5.3 CanFestival 的移植方法 .....	14
2.6 本章小结 .....	15
<b>第三章 嵌入式操作系统 <math>\mu</math>C/OS-II</b> .....	16
3.1 嵌入式操作系统 .....	16
3.1.1 嵌入式操作系统的定义及特点 .....	16
3.1.2 常用的开源嵌入式操作系统 .....	16

3.2 $\mu\text{C}/\text{OS-II}$ 概述 .....	18
3.2.1 $\mu\text{C}/\text{OS-II}$ 的特点 .....	18
3.2.2 $\mu\text{C}/\text{OS-II}$ 文件结构 .....	19
3.3 $\mu\text{C}/\text{OS-II}$ 的任务 .....	19
3.3.1 任务的存储结构 .....	19
3.3.2 任务的优先级 .....	20
3.3.3 任务的状态 .....	21
3.4 本章小结 .....	22
第四章 协议研究的目标平台设计 .....	23
4.1 引言 .....	23
4.2 相关元件介绍 .....	23
4.2.1 主控芯片 AT90CAN128 .....	23
4.2.2 隔离 CAN 收发器 CTM1050 .....	24
4.3 硬件电路设计 .....	25
4.4 本章小结 .....	29
第五章 基于嵌入式系统的协议软件设计 .....	30
5.1 可行性分析 .....	30
5.2 开发工具简介 .....	30
5.3 $\mu\text{C}/\text{OS-II}$ 在目标平台上的移植 .....	31
5.3.1 需进行的工作 .....	31
5.3.2 OS_CPU.H 的修改 .....	31
5.3.3 OS_CPU_C.C 的修改 .....	32
5.3.4 OS_CPU_A.S 的修改 .....	33
5.4 CanFestival 在目标平台上的移植 .....	36
5.4.1 需进行的工作 .....	36
5.4.2 目标处理器相关头文件的编写 .....	36
5.4.3 目标控制器驱动文件的编写 .....	38
5.4.4 协议功能应用程序的编写 .....	40
5.4.5 对象字典的编写 .....	41
5.5 CanFestival 在 $\mu\text{C}/\text{OS-II}$ 上的封装 .....	42
5.6 Makefile 文件的编写 .....	43
5.7 本章小结 .....	45
第六章 通信接口有效性实验 .....	46

6.1 有效性实验方案 .....	46
6.2 实验过程与结果 .....	46
6.3 本章小结 .....	52
第七章 总结与展望 .....	53
参考文献 .....	54
发表论文和科研情况说明 .....	57
致 谢 .....	58



## 第一章 绪论

### 1.1 CANopen 简介

CAN (Controller Area Network, 控制器局域网) 总线是一种由德国 BOSCH 公司推出, 用于汽车控制系统数据通信的现场总线。由于其具有成本低、可靠性高、抗干扰能力好和实时性强等显著特点, CAN 总线的应用范围迅速扩大, 在各个行业都得到了广泛的应用。为了应对 CAN 总线使用范围快速扩大而带来的对于标准化的进一步要求, PHILIPS SEMICONDUCTORS 于 1991 年发布了 CAN 技术规范 (第二版)。该技术规范包括 CAN 2.0A 和 CAN 2.0B 两部分, 前者给出了曾在 CAN 技术规范版本 1.2 中定义的 CAN 报文格式, 后者则给出了标准及扩展两种报文格式<sup>[1]</sup>。

由于 CAN 总线只定义了 ISO/OSI 模型中物理层和数据链路层的协议规范, 而没有规定应用层规范, 所以很多应用场合都需要使用基于 CAN 总线的高层协议, 来规定 CAN 报文中的标识符和数据的使用。目前, 已经有多个组织开发出了基于 CAN 总线的高层协议, 使得 CAN 总线的功能和应用范围都得到了极大的延伸<sup>[2]</sup>。目前, 比较常用的高层协议标准有 CAL、CANopen、DeviceNet、CAN kingdom 等。在这些常用的高层协议中, CANopen 和 DeviceNet 具有相对较广的应用范围。

CANopen 是一种由 CiA (CAN-in-Automation, 自动化 CAN 用户和制造商协会) 定义的 CAN 应用层协议。它采用面向对象的思想, 具有非常理想的模块性和适应性。CANopen 协议主要包括通讯子协议和设备子协议两部分内容:

#### 1) 通讯子协议 (Communication Profile)

通讯子协议 (CiA DS-301) 定义了基本的数据通信方式及其特性, 并规定了对象字典的主要形式及其通讯对象、通讯参数。

#### 2) 设备子协议 (Device Profile)

设备子协议 (CiA DSP-401~DSP-XXX) 定义了若干类特定设备的行为规范。对于某类设备中对象字典的每个对象, 都有对应于该类设备的设备子协议来描述其功能、名称、索引、子索引、数据类型等各项参数。

相对于其他常用的现场总线, CANopen 具有以下优点<sup>[3]</sup>:

- 1) CANopen 没有版权限制, 其全部协议文件都可以在 CiA 官方网站免费获得, 是一个真正开放的协议, 并非由某家大公司垄断。
- 2) CANopen 的物理层全部采用 CAN 总线标准, 因此其硬件成本极低, 购买十分方便。
- 3) CANopen 协议非常精练, 容易理解, 便于开发驱动程序。

## 1.2 国内外研究现状分析

CANopen 协议在国外的应用范围非常广泛。目前,已经有很多公司开发出了形形色色的 CANopen 产品,具有代表性的有:Northampton 公司出品的 CANopen 开发工具;Downers Grove 公司开发的 CANopen 模块;Woodhead 公司研发的支持 CANopen 的 BradCommunications 系列板卡;PEAK 公司生产的 PCANopen Magic Pro 开发套件;Philips 公司出品的 CANopenIA Developer's Kit 组态软件等<sup>[4]</sup>。

此外,国外的一些组织也开发出了成熟的开源 CANopen 协议栈,如 CanFestival, CANopenNode 和 MicroCANopen 等。

在我国, CANopen 应用产品的开发尚处于初级阶段,因此,相关方面的研究具有非常大的发展空间。在近期,我国学者得到的主要研究成果有:

宋晓梅、贾佳开发了基于 PIC18F258 微控制器的伺服电机 CANopen 控制模块<sup>[5]</sup>;

宋晓梅、陈锦妮基于 ARM 处理器 LPC2292,完成了在  $\mu\text{C}/\text{OS-II}$  操作系统中 CANopen 与 TCP/IP 之间的协议转换<sup>[6]</sup>;

张晶、汪滨琦、崔大海设计开发了针对仿真转台控制系统的 CANopen 通信模块<sup>[7]</sup>;

蒋智康、宋春宁、宋绍剑在 3 台 PIC18 单片机上开发了使用 CANopen 进行通信的温度测控系统<sup>[8]</sup>;

邓遵义、宁祎基于 AT89C51CC03 微处理器实现了应用于伺服电机上的 CANopen 控制模块<sup>[9]</sup>;

徐喆、张卓、闫士珍利用国外开源 CANopen 代码在 MC9S12XDP512 平台上实现了 CANopen 从节点<sup>[10]</sup>;

龚树强、潘旭东、王广林通过移植和修改开源 CANopen 协议栈,实现了基于 LPC2129 的伺服阀性能的自动测试<sup>[11]</sup>;

孙汉旭、曹志杰、贾庆轩分析了 CANopen 协议在双轮移动倒立摆机构控制系统应用中的快速性与稳定性问题<sup>[12]</sup>;

马艳歌、贾凯、徐方采用 CANopen 协议实现了 DSP 系统与上位机之间的通讯<sup>[13]</sup>;

涂智杰、何永义、李一青、周其洪使用 TMS320F2812 实现了 CANopen 与 Modbus 之间的协议转换<sup>[14]</sup>。

## 1.3 课题研究目的与意义

本课题的研究目的是基于国内外已有的资料,选定目标平台,并在该平台上进行基于嵌入式实时操作系统的 CANopen 协议分析与研究。在所有的 CAN 总线高层协议中,只有 CANopen 和 DeviceNet 的应用较为广泛,本课题之所以选择 CANopen 进行分析研究,是因为 CANopen 与 DeviceNet 相比有以下优势<sup>[15]</sup>:

1) CANopen 网络中的节点数最多可达 127 个,而 DeviceNet 网络中的节点数最多只有 64 个。

2) CANopen 协议可以通过软件设置,将系统根据用户的需求最大限度地专用化和

最优化，而无须改变硬件设置。而DeviceNet没有此项功能。

- 3) CANopen物理层全部采用CAN总线通用设备，可以在CAN总线之上直接建立CANopen高层协议。而DeviceNet具有严格的物理层协议标准。
- 4) CANopen协议完全开放，无版权限制，任何人都可以在CiA的网站上轻易获得。而DeviceNet并非对所有人公开协议。

CANopen作为一个具有代表性的CAN总线高层协议，早已得到了各个领域的广泛承认。尤其是在其发源地欧洲，CANopen早已在基于CAN总线的高层协议标准中居于领导地位<sup>[16]</sup>。但是，CAN总线在我国的应用层次一直偏低，多数情况下都处于传输互不兼容的简单应用层协议的层面上，并没有大范围地使用CANopen协议。同时，国内自主开发的CANopen相关产品尚不成熟，开发周期较长，成本较高，扩展性较差，难以顺应使用环境的变化而扩展新的功能，因此仍不能摆脱对国外公司产品的依赖。总体来说，相对于应用，CANopen的协议研究和产品开发在我国没有得到足够的重视，具有非常大的发展空间。

综上所述，进行CANopen协议的分析研究具有重要的现实意义。

## 1.4 课题研究内容

本课题主要分析与研究CANopen的网络通信协议，通过移植国外开源代码CanFestival协议栈来实现CANopen协议规范，并将该协议栈作为嵌入式操作系统 $\mu\text{C}/\text{OS-II}$ 中的一个任务运行，达到缩短开发周期，降低开发难度，节约开发成本的目的。

简单来说，本课题的研究内容包括如下六个部分：

- 1) 分析CAN高层协议CANopen；
- 2) 研究开源CANopen协议CanFestival和嵌入式操作系统 $\mu\text{C}/\text{OS-II}$ ；
- 3) 根据系统需要进行方案分析和器件选型，设计协议研究的目标平台；
- 4) 将嵌入式实时操作系统 $\mu\text{C}/\text{OS-II}$ 移植到目标平台；
- 5) 移植应用层协议栈CanFestival，并将其作为一个任务封装到 $\mu\text{C}/\text{OS-II}$ 中；
- 6) 开展通信接口的有效性实验。

## 1.5 本章小结

本章简要介绍了CAN网络高层协议CANopen，重点介绍了CANopen协议在国内外的研究进展，并说明了该技术在我国发展的情况。接着结合以上内容阐述了课题研究的目标与意义，最后给出了本课题的主要研究内容。

## 第二章 CANopen 协议分析

CANopen 架构于 CAN 总线之上，是一种常用的 CAN 总线高层协议。为了保证不同厂商生产的产品间的兼容性，CiA 不仅为 CANopen 定义了通讯子协议，还为许多重要的设备类型定义了设备子协议。通过遵循这些设备子协议，不同厂商开发出的同类型设备能轻易实现互操作<sup>[17]</sup>。

其中，通讯子协议 CiA DS-301 中定义了基本的通信对象 SDO、PDO、NMT 及特殊功能对象，并规定了它们的特性；设备子协议定义了各种设备类型的标准及行为规范，如 I/O 模块子协议 CiA DSP-401、伺服驱动设备子协议 CiA DSP-402、人机界面子协议 CiA DSP-403 等。此外，编码器、控制器和可编程控制器等设备类型都在相应的子协议中得到了定义。

### 2.1 CANopen 协议简介

#### 2.1.1 网络参考模型

由于 CAN 网络的局限性，它只定义了 ISO/OSI 模型中的网络物理层和数据链路层<sup>[18]</sup>，而没有定义 CAN 2.0A/B 报文中的 11/29 位标识符及 8 字节数据<sup>[19]</sup>。CANopen 作为基于 CAN 网络的应用层协议，使用了 CAN 网络的通讯和服务子集，并提供了标准的系统通讯模式，很好地弥补了这一缺陷<sup>[20]</sup>。

CANopen 协议与 CAN 网络之间的关系如图 2-1 所示：

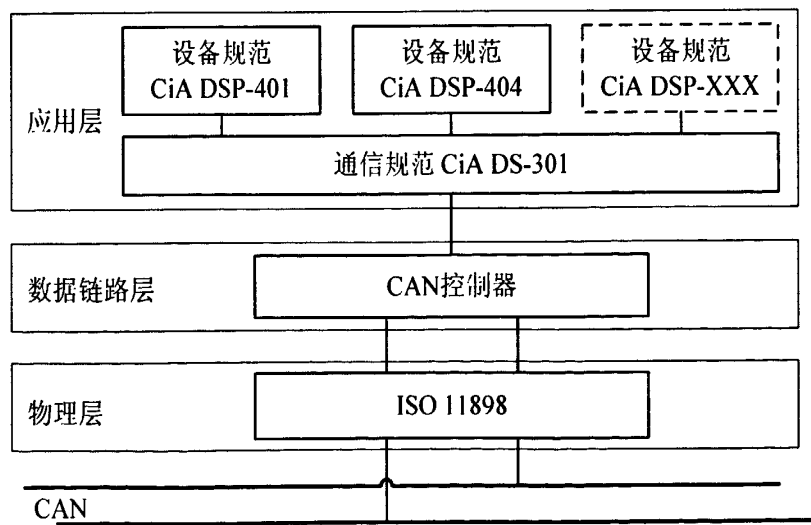


图 2-1 CANopen 与 CAN 的关系

### 2.1.2 设备模型

CANopen 的设备模型如图 2-2 所示。它分为通信接口、对象字典、应用程序三个部分<sup>[21]</sup>。通信接口提供了现场总线上通信对象的发送和接收服务；对象字典描述了所在设备上的数据种类、通信对象和应用对象，为应用程序提供了接口；应用程序提供了内部控制功能，并为硬件提供了接口。

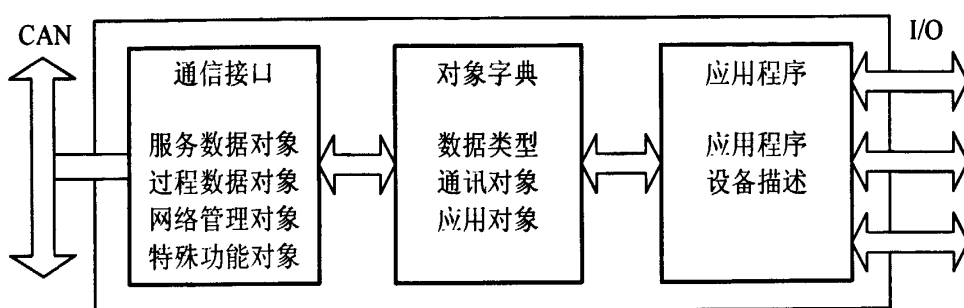


图 2-2 CANopen 设备模型

## 2.2 CANopen 对象字典

对象字典 (Object Dictionary) 是 CANopen 协议的核心，连接了 CANopen 设备的通信接口和应用程序。CANopen 网络中的每个节点都需要有一个对象字典，其中包含了描述该设备网络行为的所有参数。通过访问节点的对象字典，系统可以得到节点状态，确定通信模式，进行网络管理。同时，通过定义对象字典中的参数，也可以在设备中扩展新的功能<sup>[22]</sup>。

对象字典是一个预定义的，可以在网络中访问的，有序的对象组，通过 EDS (Electronic Data Sheet, 电子数据文档) 进行描述，对象字典中每个对象的元素都可以使用一个 16 位的索引和一个 8 位的子索引进行寻址。对象字典的通用结构参照表 3-1，其中索引值低于 0x0FFF 的对象是数据类型定义，0x1000 到 0x9FFF 之间是节点功能定义。

在对象字典中，索引 0x0001 到 0x001F 定义了静态数据类型，包含标准数据类型，如布尔型，整数，浮点数，字符串等。这些数据类型的定义仅供参考，不能被数据读取或写入。索引 0x0020 到 0x003F 定义了复杂数据类型，这种类型是预定义的由简单数据类型所组合成的标准结构，如 PDOCommPar, SDOPParameter。0x0040 到 0x005F 是设备制造商在 CANopen 协议中提供的复杂数据类型之外，为特定设备自定义的复杂数据类型。

此外，设备子协议中也可以定义数据类型。其中，静态数据类型可以在 0x0060 到 0x007F 索引区间定义，复杂数据类型可以在索引 0x0080 到 0x009F 之间定义。索引 0x1000 到 0x1FFF 之间为通讯子协议定义区间，这一区间包含了 CAN 网络的通信参数，如设备类型、错误寄存器、支持的 PDO 数量等，适用于所有设备。0x2000 到 0x5FFF

是设备制造商特定子协议区域，制造商可以在此区间为设备自定义子协议。在标准化的设备上，索引 0x6000 到 0x9FFF 包含了标准的设备子协议。设备中所有可通过网络读写的数据对象都在此定义，描述了设备的参数和功能。在这一区间内，最多可描述 8 个不同的模块，也就是说，通过对这一区域的定义，可以实现最多由 8 个模块所组成的多功能设备。

表 2-1 CANopen 对象字典通用结构

索引	对象
0000	未使用
0001 - 001F	静态数据类型
0020 - 003F	复杂数据类型
0040 - 005F	制造商规定的复杂数据类型
0060 - 007F	设备子协议规定的静态数据类型
0080 - 009F	设备子协议规定的复杂数据类型
00A0 - 0FFF	保留
1000 - 1FFF	通讯子协议
2000 - 5FFF	制造商特定子协议
6000 - 9FFF	标准的设备子协议
A000 - FFFF	保留

2.3 CANopen 通信对象

CANopen 协议规定了 4 类通信对象：SDO（Service Data Object，服务数据对象）、PDO（Process Data Object，过程数据对象）、NMT（Network Management，网络管理对象）以及特殊功能对象<sup>[23]</sup>。这一节将对 CANopen 协议规定的通信对象进行一个简要的介绍。

2.3.1 SDO

SDO 用于传送较大的低优先级数据，通常用来传送组态数据或配置设备。它的传输采用客户机/服务器通讯方式，通过两个 CANopen 设备之间的点对点通信来实现可靠的数据传输。其中，SDO 客户机是指要求访问某设备的对象字典的设备，SDO 服务器是指拥有要被访问的对象字典的设备。在使用 SDO 进行通信时，客户机通过使用索引和子索引来定义服务器对象字典中的对象，并通过 SDO 进行访问。同时，由于 SDO 是确认服务类型，所以一个 SDO 通信需要两个带有不同标识符的 CAN 报文，其中一个报文由 SDO 服务器使用，另一个报文由 SDO 客户机使用<sup>[24]</sup>，如图 2-3 所示。

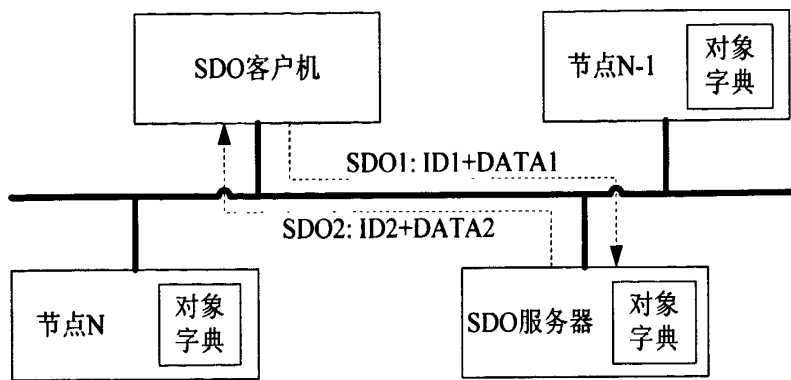


图 2-3 SDO 的通信

SDO 可以传送任何长度的数据。当传送的数据长度小于 4 字节时，采用加速传送（Expedited Transfer）方式进行传送；而当传送的数据长度大于 4 字节时，数据将被分拆成几个 CAN 报文，采用分段传送（Segmented Transfer）方式进行传送。由于在分段传送时一个数据将会被分拆成几段，因此在发送 SDO 的第一个报文后，其后的每段中都可以包含 7 个字节的数据，而最后一段可以包含一个终止符。两种传送方式中 SDO 的基本结构如图 2-4 和 2-5 所示。

服务器到客户机/客户机到服务器

字节0	字节1-2	字节3	字节4-7
SDO命令字	对象索引	对象子索引	最大4字节数据

图 2-4 SDO 的加速传送方式

服务器到客户机/客户机到服务器

字节0	字节1-7
SDO命令字	最大7字节数据

图 2-5 SDO 的分段传送方式

在 SDO 基本结构中，命令字可包含如下信息：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、触发位（Toggle Bit）。其中触发位在分段传送的每个分段中会交替进行清零和置位。

2.3.2 PDO

PDO 采用生产者/消费者方式进行传输，用于从一个作为生产者的发送方对一个作为消费者的接受方传输实时控制参数、变量，或从一个作为生产者的发送方对多个接收方以广播形式进行传输<sup>[25]</sup>，如图 2-6 所示。PDO 的优先级高于 SDO，但一个 PDO 最大只能传输 8 个字节的数据。

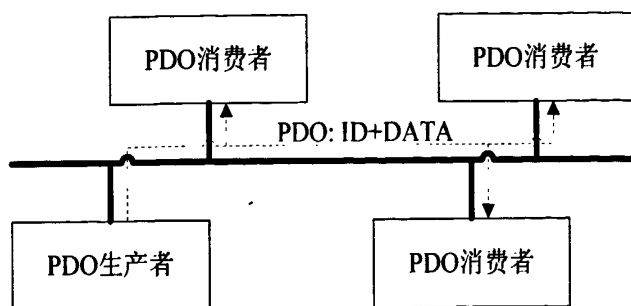


图 2-6 PDO 的通信

PDO 通讯既没有协议规定，也不需进行确认。它的数据内容只由它的 COB-ID 定义，仅能对应于设备对象字典中的条目，为应用程序提供预先确定的访问接口。也就是说，PDO 的生产者和消费者都必须了解 PDO 报文中所含信息的意义。这就要求 PDO 报文中的数据类型和应用对象都是由设备对象字典中相应索引和子索引所对应的参数一一映射的。因此，在设备对象字典中对象的索引和子索引发生变化之后，就必须使用 SDO 报文来重新配置 PDO 映射参数。

在对象字典中，每个 PDO 都由通讯参数和映射参数两个部分构成。通讯参数包括该 PDO 使用的 COB-ID、传输类型、禁止时间和定时器周期<sup>[26]</sup>。映射参数包括对象字典中对象的索引和子索引，PDO 的生产者和消费者必须知道映射参数，以解释 PDO 报文中信息的含义。此外，如果设备支持可变 PDO 映射，那么就能够使用 SDO 报文来配置 PDO 映射参数。

### 2.3.3 NMT

NMT 用于对网络设备进行管理、控制，并报告设备故障。NMT 采用主从结构，一个网络中只有一个 NMT 主节点，其余的为从节点，如图 2-7 所示。

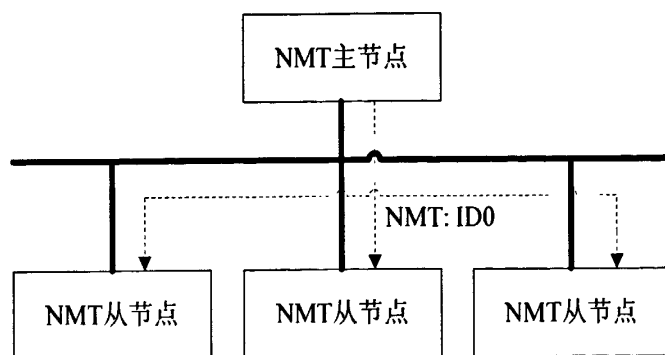


图 2-7 NMT 的通信

每个 CANopen 从节点都有初始化 (Initialization)、预操作 (Pre-Operational)、操作 (Operational) 和停止 (Stopped) 4 个状态。NMT 报文能在任何时候使所有节点，或是部分节点转换到相应的工作状态，主节点可以使用 NMT 来控制从节点在各个状态之间的转换，如图 2-8。NMT 主节点可以对从节点提供状态管理、节点保护等服务，还可以监控从节点的状态，以检查某个不发送 PDO 的从节点是否已经脱离 CANopen 总线，同



时检测设备是否出现网络接口错误。

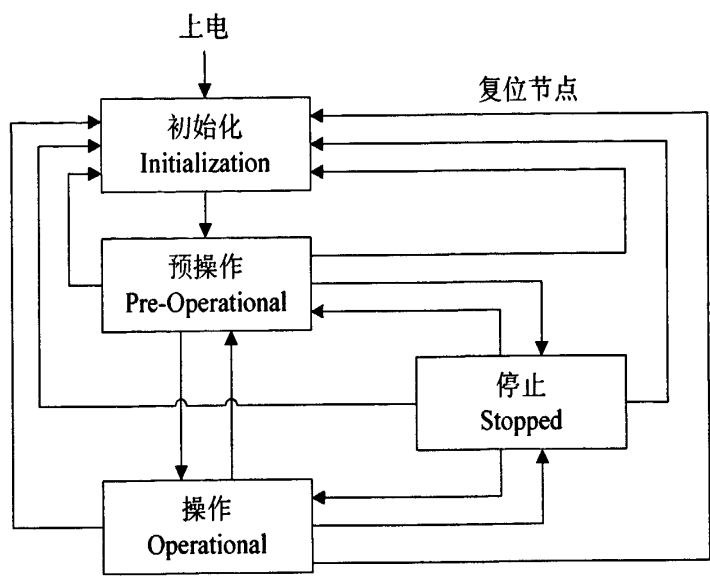


图 2-8 节点状态转换

在 CANopen 从节点上电后，会自动进行初始化，并进入预操作状态。节点处于预操作状态时不能进行 PDO 通信，但可以通过配置工具通过 SDO 配置节点。

NMT 主节点可以将某一个从节点切换到操作状态，也可以同时将所有从节点均切换到操作状态。当节点处于操作状态时，可以进行所有的通信操作，并且可以通过 SDO 对从节点的对象字典进行修改操作。

通过将设备切换到停止状态，可以中断所有 SDO 和 PDO 的通信。在停止状态下，节点除 NMT 对象外不允许任何通信。此外，停止状态可以被用来实现某些程序中的功能，这种行为的定义属于设备配置文件的范畴。

初始化状态又分成三个子状态<sup>[27]</sup>。在 Reset\_Application 子状态中，制造商特定子协议和标准化设备子协议两个区域中的参数均被设置成缺省值；在 Reset\_Communication 子状态中，通讯子协议区域的参数被设定为它们的通电值，也就是上一次存储的参数；第三个子状态是初始化状态，节点在上电后或复位后会自动地进入此状态。

NMT 的报文由 CAN 报文的报头和两字节数据组成，如图 2-9。其中，第一个字节是 NMT 命令字，表示节点所请求的服务类型，第二个字节是需要寻址的节点 ID。当节点 ID 为 0 时，NMT 主节点会寻址所有从节点。

NMT主节点到NMT从节点		
COB-ID	字节0	字节1
0x00	NMT服务类型	节点ID

图 2-9 NMT 报文格式

2.3.4 特殊功能对象

CANopen 协议中所定义的特殊功能对象主要包括同步对象、时间戳对象和应急对象<sup>[28]</sup>。

1) 同步对象（Synchronization Object）

同步对象采用生产者/消费者通讯方式，是由同步对象生产者向同步对象消费者提供的基本时钟信号，它为同步对象消费者提供了标准的通信周期<sup>[29]</sup>。同步对象消费者接收到此信号后才进行它们的同步任务。

同步对象的发送周期由对象字典中索引值为 0x1005 的参数所规定，可以在节点启动时由配置工具进行更改。为了使同步对象可以被及时发送，同步对象拥有很高的优先级，通常为 128，这是默认情况下的最高优先级。一般来说，同步对象不能传送任何数据。

2) 时间戳对象（Time Stamp Object）

时间戳对象传送的是 1984 年 1 月 1 号到当前日期的天数和当日 0 时到当前时间的毫秒数，一共 6 个字节。时间戳对象的作用是为对时间要求非常苛刻的设备提供毫秒级的高精度同步协议。

时间戳对象的发送周期由对象字典中索引为 0x1012 的参数规定。某些情况下，由于传输速率的降低，在大型网络的关键应用中的设备需要非常准确的同步信号，甚至需要精度为微秒级的本地时钟。这时就可以通过选择时间戳对象这一高分辨率的同步协议来调整本地时钟不可避免的漂移。

3) 应急对象（Emergency Object）

应急对象用于发送中断类型错误警告。它会在设备内部发生严重错误的情况下被触发，并使用最高优先级发送到其他设备。节点每当发生错误时都会发送应急对象，并且只发送一次。但是，如果设备内部没有错误发生的话，节点就绝对不会发送应急对象。

应急对象的报文格式如图 2-10 所示。

NMT主节点到NMT从节点

COB-ID	字节0-1	字节2	字节3
0x80+Node ID	应急错误代码	对象寄存器	制造商特定的错误区域

图 2-10 应急对象报文格式

在上图中，应急对象的错误代码可以通过 CANopen 的通信子协议来定义，预定义的错误代码如表 2-2 所示。设备的错误寄存器和其他相关信息也可以在设备子协议中进行定义<sup>[30]</sup>。

表 2-2 应急对象的错误代码

应急错误代码	代码功能描述		应急错误代码	代码功能描述
00xx	错误重启或无错误		62xx	用户软件错误
10xx	通用错误		63xx	数据集错误
20xx	电流错误		70xx	附加模块错误
21xx	设备输入端电流错误		80xx	监视器错误
22xx	设备内电流错误		81xx	通信错误
23xx	设备输出端电流错误		8110	CAN 超时
30xx	电压错误		8120	错误认可
31xx	主电压错误		8130	保险错误
32xx	设备内电压错误		8140	从总线关闭中恢复
33xx	输出电压错误		82xx	协议错误
40xx	温度错误		8210	未处理 PDO
41xx	外界温度错误		8220	数据超长
42xx	设备内部温度错误		90xx	外部错误
50xx	设备硬件错误		F0xx	附加功能
60xx	设备软件错误		FFxx	设备自定义
61xx	自身软件错误			

2.4 典型开源 CANopen 协议栈

近年来，已经有很多公司和组织开发出了成熟的商用 CANopen 协议栈。但商用协议栈高品的价格会为开发增加较多成本，而自行开发协议栈又会耗费太多的精力和时间。因此可以利用开源的 CANopen 协议栈来进行协议的研究和开发，以达到加快开发速度，降低开发难度的目的。目前，主要有 CanFestival、MicroCANopen 和 CANopenNode 三种符合 LGPL 和 GPL 协议的常用开源 CANopen 协议栈。

2.4.1 CanFestival

CanFestival 是一个由 Lolitech 资助的完整 CANopen 协议栈，由 Edouard TISSERANT 于 2001 年开始编写。CanFestival 具有非常成熟的代码，目前已经更新到了 3.0 RC3 版，可在 PC 和微控制器上实现较为完整的主节点或从节点功能。它的代码使用 ANSI-C 编写，具有良好的可移植性，并且完全支持 DS-301 V4.02 标准。此外，CanFestival 还提供了带有图形用户界面的对象字典编辑器，从而为设备对象字典的编写提供了方便。

关于 CanFestival 的其他特性将在下文详细阐述。

### 2.4.2 MicroCANopen

MicroCANopen 是由 Embedded Systems Academy 开发的开源 CANopen 协议栈，源代码可从其官方网站上免费获取，目前已经更新到了 3.30 版。MicroCANopen 的代码相当精简，只保留最低限度的功能，因此它仅能应用于嵌入式网络系统，通常只用做单一功能模块。

MicroCANopen 的特点如下<sup>[32]</sup>：

- 1) 功能精简。MicroCANopen 并不完全与 CANopen 规范相一致，它只提供最基本的 CANopen 通讯功能，其他功能都需要通过另外的配置程序和分析程序完成。
- 2) 硬件需求极低。在 8051 单片机上，MicroCANopen 只需要 4K 字节的代码空间和 170 字节的 RAM 就可实现最低限度的功能。即使实现全部的 CANopen 功能，也不需要很高的硬件需求。
- 3) 简单易学。得益于 MicroCANopen 代码的简洁，即使是 CANopen 的初学者也可以快速上手。开发项目时，开发者可以只包含自己所需的功能，并忽略不需要的功能，从而快速开发所需的项目。
- 4) 源码配置简单。MicroCANopen 3.30 版可以包含由 EDS (Electronic Data Sheet) 编辑器所生成的 C 语言源码。通过使用 EDS 编辑器，开发者可以得到所需的 CANopen 功能，并生成 MicroCANopen 中所需的 C 语言源码。
- 5) 对 MicroCANopen Plus 的良好移植性。Embedded Systems Academy 还开发了一个具备大多数 CANopen 功能的 MicroCANopen Plus 版本，MicroCANopen 用户可以方便地将自己的项目移植到 MicroCANopen Plus 上。

### 2.4.3 CANopenNode

CANopenNode 由斯洛文尼亚的 Janez Paternoster 编写，是一个基于 Microchip 公司 PIC18 微控制器的开源 CANopen 协议栈，目前最新的版本是 1.10 版。它可以实现主节点或从节点的功能，提供多个设备之间在 CANopen 网络上的串行通信<sup>[31]</sup>。CANopenNode 的功能齐全，协议接口完善，完全遵从于 CiA DS-301 标准和 CANopen 指示灯标准 CiA DR-303-3。

CANopenNode 目前仅支持 Microchip 公司生产的 PIC18 微控制器，Philips 公司生产的 SJA1000 网络控制器和 Borland C++ 5.02 编译器的组合。此外，CANopenNode 也可以移植到其他控制器上。

## 2.5 开源协议栈 CanFestival

### 2.5.1 CanFestival 的特点

相对于其他常用的国外开源 CANopen 协议栈，CanFestival 具有许多明显的优势<sup>[33]</sup>：

- 1) CanFestival 为开发者提供了许多工具，以提高开发的便利性。例如用于生成节

点对象字典源代码的对象字典编辑器，以及便于开发者自由配置编译选项的配置脚本。

- 2) CanFestival 能够运行于多种类型的平台。CanFestival 源代码由 ANSI-C 编写，驱动和例程的编译情况仅取决于具体的编译工具。在目前最新的 3.0 RC3 版本中，官方提供了对于多种硬件平台的驱动。此外，CanFestival 可以在任意类 Unix 系统下编译和运行，如 Linux 和 FreeBSD。
- 3) CanFestival 协议功能完整，完全符合 CANopen 标准。CanFestival 完全支持 2002 年 2 月发布的 CiA DS-301 V4.02 标准，并支持 CiA DS-302 中的简明 DFC 协议。

2.5.2 CanFestival 的结构

CanFestival 3.0 RC3 源代码的目录结构如表 2-3 所示。

表 2-3 CanFestival 文件目录结构

文件路径	文件说明
./src	与处理器无关的 CANopen 协议栈 ANSI-C 源代码
./include	针对各处理器的头文件
./drivers	针对各硬件的驱动
./examples	用于测试的程序
./objdictgen	带有图形用户界面的对象字典编辑器
./doc	说明文档

在 CanFestival 中，所有源代码可以分为四大部分，分别是目标接口、CAN 接口、CanFestival 库文件以及主/从节点的应用。其中，CanFestival 库文件是整个协议的核心，包括调度管理 (timer.c)，节点管理 (对象字典访问 objaccess.c、状态机 state.c)，CANopen 协议 (服务数据对象 sdo.c、过程数据对象 pdo.c、同步对象 sync.c、自动波特率对象 lss.c、网络管理对象 nmtMaster.c 和 nmtSlave.c)。这些文件在移植时是不需要修改的。主/从节点的应用包括节点状态反馈和设备对象字典的定义。目标接口包括节点硬件的驱动以及对于操作系统的接口。也是进行移植时的主要修改对象。

由于周期性发射同步信号、心跳报文或 SDO 超时信号需要设定一系列定时信号来提醒系统进行这些工作，所以 CANopen 节点必须能实现定时功能。CanFestival 在 timer.c 中执行了一个微型调度程序，它可以使用一个定时器来模拟许多定时器。该调度程序会建立并管理一个警报表，并且在规定的时间发出信号。

图 2-11 展示了 CanFestival 的代码功能结构以及它们之间的关系。

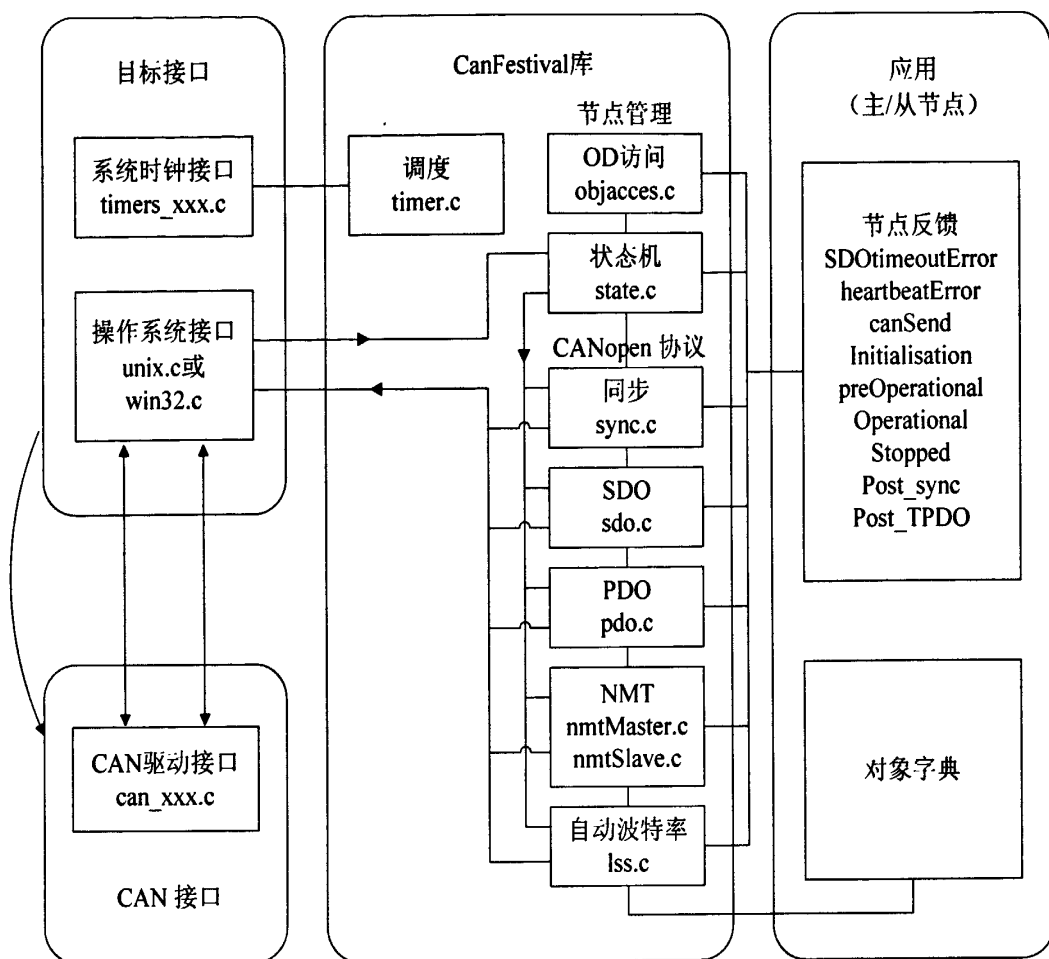


图 2-11 CanFestival 代码功能结构

### 2.5.3 CanFestival 的移植方法

在 CanFestival 3.0 RC3 的源代码中，已经提供了 Motorola MC9S12DP256 微控制器的驱动代码，因此可以直接使用 GNU GCC 编译器编译程序并烧写至 MC9S12DP256 控制器。之后通过 CAN 控制器 PCA82C250 将 MC9S12DP256 的 CAN0 端口（PM0，PM1）连接到 CANopen 网络，并通过 MAX232 将芯片上的 PORTS（TxD0）端口连接到波特率设置为 19200bps 的控制台上，就可以在 MC9S12DP256 上实现该协议栈。

该协议栈也可以方便地移植到其他控制器上。在移植之前，首先必须确保目标控制器具有 40K 以上的代码空间和 2K 以上的 RAM。相对于自己重新编写代码来说，选择官方文件包中提供的某个最接近目标控制器平台的源代码，并在其基础上进行修改显然会使移植过程更加简化。在移植时，最好使用源代码中提供的例程作为移植目标节点的基础，并进行修改。同样，也可以使用官方提供的对象字典源文件来作为移植目标的对象字典，并在其基础上进行修改。移植过程中需要修改的文件包括：include 文件夹中与目标处理器相关的头文件、drivers 文件夹中的目标控制器驱动文件以及各相关文件夹中的 Makefile 文件。

### 2.6 本章小结

作为一种常用的 CAN 总线应用层协议，CANopen 在国内外各领域都得到了广泛的应用。本章详细分析了 CANopen 的协议结构、对象字典和通信对象，研究了国外的典型开源 CANopen 协议栈的结构与特点，提出了进行 CANopen 协议规范通信接口开发的有效方案，并给出了 CanFestival 在微控制器上的移植思路。

## 第三章 嵌入式操作系统 $\mu\text{C}/\text{OS-II}$

### 3.1 嵌入式操作系统

#### 3.1.1 嵌入式操作系统的定义及特点

嵌入式操作系统是为了实现使对象智能化的目的，而嵌入对象中进行控制的专用计算机系统。它集软硬件于一身，负责系统资源的分配与调度，具有通用操作系统的基本特点<sup>[34]</sup>。

与常规计算机系统相比，嵌入式操作系统有如下特点：

- 1) 微型化高。由于嵌入式操作系统能使用的资源通常较少，所以嵌入式操作系统不能占用太多资源。嵌入式操作系统必须在满足功能的前提下，尽可能地做到微型化以节约资源。
- 2) 专用性强。由于嵌入式操作系统中所使用的硬件和软件通常都是为某特定功能所设计，所以嵌入式系统通常具有较高的专用性。
- 3) 实时性好。由于嵌入式系统被广泛使用于工业应用甚至国防应用中，所以嵌入式系统必须有很强的实时性。为了使系统具备快速响应的能力，嵌入式操作系统必须设计成可剥夺型内核。
- 4) 裁剪性好。由于嵌入式操作系统具有很强的专用性，因此每一个使用场合的软硬件配置也不尽相同。这就要求嵌入式系统具有很好的裁剪性，以便根据不同的应用场合，在满足应用的条件下实现最合理的配置，从而实现通用性与专用性之间的平衡。
- 5) 移植性好。由于嵌入式操作系统的应用范围非常广泛，为了适应不同的硬件平台，嵌入式操作系统必须具有非常好的移植性，从而可以在对文件修改最少的情况下稳定移植于不同的平台。
- 6) 可靠性高。由于嵌入式操作系统通常应用于工业控制等场合，甚至还会应用于军事、交通等对人民生命财产安全息息相关的行业，因此嵌入式操作系统必须具有极高的可靠性，甚至还需要具有必要的容错机制，从而进一步提高系统的可靠性。
- 7) 功耗低。由于嵌入式操作系统通常应用于使用电池的小型系统，因此嵌入式操作系统必须具有低功耗的特点，以便更好地延长系统的使用时间。

#### 3.1.2 常用的开源嵌入式操作系统

目前，有许多源码开放的嵌入式操作系统被广泛使用，但是源码开放的嵌入式操作系统不一定是免费的。有些嵌入式操作系统只是内核开源，并且在进行科研和非盈利用



途时可以免费使用，当使用在商业用途上时仍需收取费用。

#### 1) $\mu$ C/OS-II

$\mu$ C/OS-II 是一个简洁易用的基于优先级的抢占式多任务实时内核，由 Jean J.Labrosse 开发，并于 2000 年得到了美国联邦航空管理局的安全性及稳定性认证。 $\mu$ C/OS-II 是一个小型的嵌入式操作系统，它只提供了任务管理、任务的通信同步和简单的存储管理三项基本服务<sup>[35]</sup>。

$\mu$ C/OS-II 在商业应用时，需要支付一次性的产品版税。但作为个人和学校学习使用及非商业应用时，无需支付任何费用。此外， $\mu$ C/OS-II 还提供  $\mu$ C/FS、 $\mu$ C/GUI、 $\mu$ C/View 等收费的周边程序和工具。2009 年 9 月 15 日，Jean J.Labrosse 发布了  $\mu$ C/OS-II 操作系统的升级版  $\mu$ C/OS-III，但  $\mu$ C/OS-III 并不免费，且不再开放源代码。

关于  $\mu$ C/OS-II 的更多信息，将在下文中详细介绍。

#### 2) $\mu$ CLinux

$\mu$ CLinux 基于 Linux 系统开发，遵循 GNU 协议，是一个源代码完全开放的系统，专门应用于没有 MMU (Memory Management Unit, 内存管理单元) 的嵌入式处理器。由于硬件中没有 MMU，所以  $\mu$ CLinux 不能使用 Linux 中的虚拟内存管理技术，并且不支持内存中的页交换。

但是，由于  $\mu$ CLinux 是从 Linux 的内核派生出来的，因此  $\mu$ CLinux 继承了 Linux 的绝大部分优良特性，包括体积小、稳定性好、移植性好、网络功能优良、文件系统支持完备、API 函数丰富等。这些特性都为良好运行嵌入式程序提供了坚实的基础。

但是， $\mu$ CLinux 是一种分时操作系统，而不是实时操作系统。因此，该系统更适用于对实时性要求较低的应用情况。而在实时性要求非常高的场合，就必须对  $\mu$ CLinux 进行实时性改造<sup>[36]</sup>。另外， $\mu$ CLinux 只支持 32 位处理器，复杂度也高，因此需要的资源较多，移植难度较大。

#### 3) FreeRTOS

FreeRTOS 是一款开放源代码的微型实时内核，是近年来比较流行的一个嵌入式实时操作系统。它免费提供下载、无版税，并可无偿应用于商业应用。同时，FreeRTOS 还为多种处理器和开发工具提供了官方移植包。

FreeRTOS 可以有多个优先级相同的任务，它们按照时间片来轮流处理，并且不限制可创建任务数量和可用优先级数量<sup>[37]</sup>。此外，FreeRTOS 非常简洁，其内核仅由 3 个 C 语言文件组成，因此它具有最低限度的硬件资源开销，更适合嵌入式系统的开发。

但是，RTOS 的发展时间相对较短，使用范围不是很广泛。并且相对于其他嵌入式操作系统，FreeRTOS 任务间通讯得只提供了消息队列和信号量的实现，无法以后进先出的顺序向消息队列发送消息。此外，FreeRTOS 的扩展只支持 TCP/IP，而没有其他外延支持，因此扩展度较差。

#### 4) eC/OS

$\text{eC}/\text{OS}$  是由 Redhat 公司推出的嵌入式实时操作系统，非常适合用于嵌入式系统。该系统将操作系统做成静态库的方式，从而让应用程序通过连接产生出具有操作系统特性的应用程序。

$\text{eC}/\text{OS}$  使用 GNU Toolchain 维护，并且模块化程度非常好。此外， $\text{eC}/\text{OS}$  支持多种 CPU 架构，复杂度也较低。同时，该系统针对硬件的驱动架构简单，移植容易，并且占用的空间也比较小。

但是， $\text{eC}/\text{OS}$  支持的文件系统和通信协议较少，调度模式也较少。此外，该系统在实际中应用并不广泛，所以开发时可得到的技术支持相对有限。

5) AVR $\text{X}$

AVR $\text{X}$  是使用 WINAVR 编写的 AVR 单片机专用嵌入式系统，目前最新的版本是 2.6。由于 AVR $\text{X}$  使用汇编语言编写，所以该操作系统的代码很小，即使是包含所有功能的完整代码也只有 2K 字节<sup>[38]</sup>。

在实际应用中，由于系统不一定会用到所有功能，因此代码可以被进一步精简，通常只需要 500~700 字节。AVR $\text{X}$  包含了 34 个 API 接口，主要用于任务、旗语、时间管理、信息队列、单步调试支持等模块中。

由于 AVR $\text{X}$  是面向 AVR 单片机开发的系统，因此它与硬件系统具有无与伦比的契合度。但是，正因为它是一种专用的操作系统，因此很难移植到其他平台上，这就决定了其应用范围非常狭窄，以至于无法提供完整的技术支持。

3.2  $\mu\text{C}/\text{OS-II}$  概述

3.2.1  $\mu\text{C}/\text{OS-II}$  的特点

$\mu\text{C}/\text{OS-II}$  是典型的微内核实时操作系统，提供了任务调度、任务管理、时间管理和任务通信等基本功能，其体系结构如图 3-1 所示。

与处理器无关的代码		与应用程序 相关的代码
OS_CORE.C	OS_FLAG.C	OS_CFG.H INCLUDES.H
OS_MBOX.C	OS_MEM.C	
OS_Q.C	OS_SEM.C	
OS_TIME.C	OS_MUTEX.C	
OS_TASK.C	uCOS_II.C	
uCOS_II.H		
与处理器相关的代码		
OS_CPU.H	OS_CPU_A.ASM	OS_CPU_C.C
CPU		时钟

图 3-1  $\mu\text{C}/\text{OS-II}$  体系结构

$\mu$ C/OS-II 只抽象和封装了宿主对象的 CPU 和定时器，并未提供别的硬件抽象层<sup>[39]</sup>。除了极少部分与处理器密切相关的代码使用了汇编语言编写以外，它的绝大部分代码都是使用 C 语言编写的。这种特性赋予了其极强的可移植性，在选定具体硬件后，只需更换一个硬件抽象层，就可完成所有的移植工作。

此外， $\mu$ C/OS-II 采用了基于优先级的可剥夺型内核，系统中每个任务都有一个唯一的优先级，因此它的实时性非常好，非常适用于强实时性场合。

3.2.2  $\mu$ C/OS-II 文件结构

得到  $\mu$ C/OS-II 源码并解压后，会得到 Software 文件夹，所有与程序相关的源代码都在该文件夹下。表 3-1 中列出了  $\mu$ C/OS-II 的目录结构。

表 3-1  $\mu$ C/OS-II 的目录结构

文件路径	文件说明
\SOFTWARE\BLOCKS	子程序模块目录
\SOFTWARE\HPLISTC	与范例 HPLIST 相关的文件
\SOFTWARE\TO	与范例 TO 相关的文件
\SOFTWARE\uCOS-I	与 $\mu$ C/OS-II 相关的所有文件
\SOFTWARE\uCOS-II\EX1_x86L	实例源代码
\SOFTWARE\uCOS-II\EX2_x86L	实例源代码
\SOFTWARE\uCOS-II\EX3_x86L	实例源代码
\SOFTWARE\uCOS-II\Ix86L	依赖于处理器类型的代码
\SOFTWARE\uCOS-II\SOURCE	与处理器类型无关的源代码

其中，SOURCE 文件夹就是我们进行移植时所需要修改的文件夹。

3.3  $\mu$ C/OS-II 的任务

实时系统中的每一个任务都对应一个外部事件。每当一个外部事件发生时，都会有一个任务被触发，来响应和处理外部事件。由于  $\mu$ C/OS-II 是可剥夺型内核，所以当前任务会一直保持运行状态，直到拥有更高优先级的任务来剥夺其处理器使用权<sup>[40]</sup>。当然，当前任务也可以主动交出处理器使用权。

3.3.1 任务的存储结构

在  $\mu$ C/OS-II 中，从代码角度来看，任务就是类似于如下代码的一个具有死循环的函数：

```
void taskExample(void *pdata)
```

```
{
    for(;;)
    {
        任务代码;
    }
}
```

而从任务的存储结构角度看， $\mu\text{C}/\text{OS-II}$  的任务由程序代码、任务堆栈和任务控制块三个部分组成。其中，任务程序代码是任务程序的有效执行部分，任务堆栈用来保存任务的当前执行环境，而任务控制块用来保存任务的属性。 $\mu\text{C}/\text{OS-II}$  中任务的存储结构如图 3-2 所示。

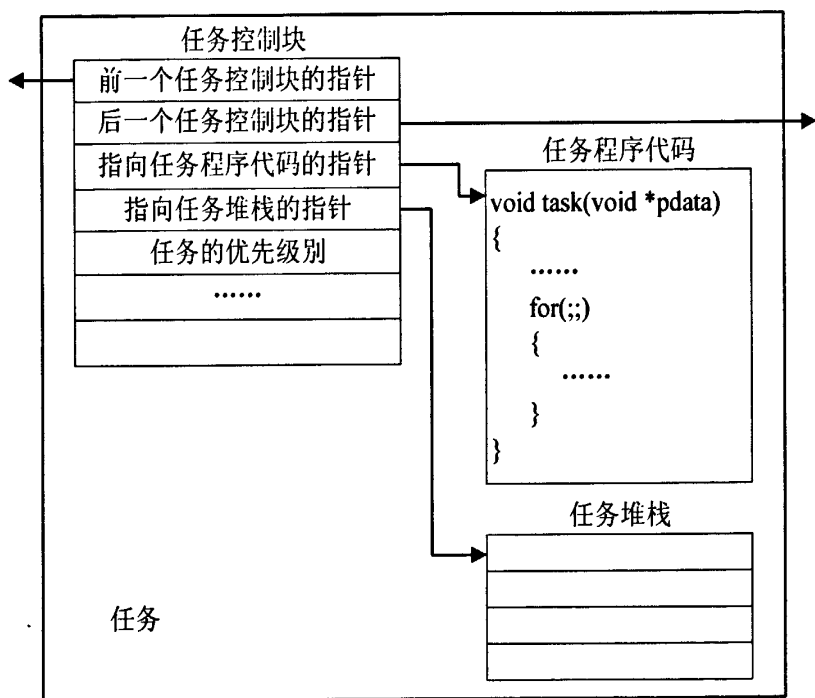


图 3-2 任务的存储结构

其中，任务控制块存放了该任务前一个任务和后一个任务的控制块指针。当系统中有多个任务时，这两个指针会将所有任务连成一个链表。此外，任务控制块中还有指向任务程序代码的指针和指向任务堆栈的指针，以及存放了任务优先级别的寄存器。因此，任务控制块是嵌入式系统管理任务的依据。

3.3.2 任务的优先级

$\mu\text{C}/\text{OS-II}$  中的每个任务都拥有一个唯一的优先级，以便系统确定处理器分配顺序。 $\mu\text{C}/\text{OS-II}$  共有 64 个优先级，用 0-63 的数字表示。其中，数字越大表示优先级越低，0 是最高优先级，63 是最低优先级。

在实际应用中，为了节约资源，用户也可以定义配置文件 `OS_CFGH` 中的宏

`OS_LOWEST_PRIO`，来声明应用中的最低优先级。当该宏被定义后，系统中可用的优先级就为 0 到 `OS_LOWEST_PRIO`，共 `OS_LOWEST_PRIO+1` 个。

为了使处理器在没有用户任务时仍然有任务可运行，因此系统定义了空闲任务，并将最低优先级 `OS_LOWEST_PRIO` 赋给空闲任务。用户应用程序中必须使用该空闲任务。此外，为了便于用户了解处理器使用率， $\mu\text{C}/\text{OS-II}$  还定义了可选的统计任务。如果应用程序中使用了统计任务，系统会自动将优先级 `OS_LOWEST_PRIO-1` 赋给统计任务。

### 3.3.3 任务的状态

由于在  $\mu\text{C}/\text{OS-II}$  中，处理器属于独占资源，因此，在某一个时刻，最多只有一个任务可以得到处理器使用权。因此， $\mu\text{C}/\text{OS-II}$  中的任务根据是否占有资源、是否等待资源以及是否被中断等情况，被分为睡眠状态、就绪状态、运行状态、等待状态以及中断服务状态。下面分别介绍这几种状态：

- 1) 睡眠状态。在该状态下，任务没有被分配任务控制块，或被剥夺了任务控制块。也就是说，代码驻留在程序空间，但并未由操作系统管理。
- 2) 就绪状态。如果任务被分配了任务控制块，并在任务就绪表中进行了就绪登记，任务就进入了就绪状态。
- 3) 运行状态。处于就绪状态的任务如果获得了处理器的使用权，则该任务就进入了运行状态。由于处理器属于独占资源，所以任何时刻有且只有一个任务处于运行状态。并且，只有当所有优先级高于该任务的任务都转为等待状态的情况下，任务才能从就绪状态转为运行状态。
- 4) 等待状态。任务在运行状态时，当需要等待一段时间后再运行，或等待一个事件发生时，处理器的使用权就会被交给其他任务，从而使该任务进入等待状态。
- 5) 中断服务状态。处于运行状态的任务响应中断申请后，就会终止当前任务的运行，而去执行中断服务程序，这时任务的状态就叫做中断服务状态。

任务在系统的控制下，会根据具体情况来转换自己的状态。任务状态的转换见图 3-3 所示。

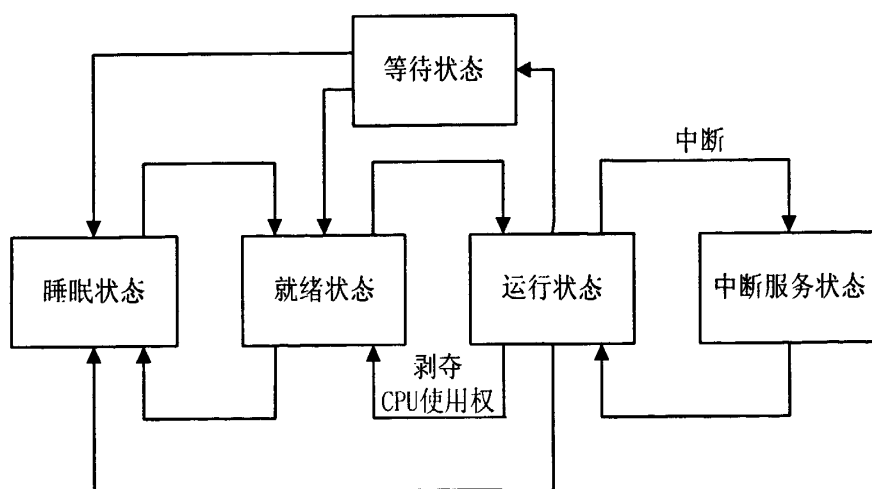


图 3-3 任务状态的转换

### 3.4 本章小结

本章介绍了嵌入式操作系统的定义和特点,并简要介绍了几种最常用的嵌入式操作系统,对于其中应用最广泛的  $\mu\text{C}/\text{OS-II}$ ,本文详细介绍了其特点、文件结构,及其任务的存储结构、优先级和状态转换。

## 第四章 协议研究的目标平台设计

### 4.1 引言

目标平台的选定是进行协议研究的前提。本文的目标平台需要实现 CANopen 协议处理和 CAN 通讯等功能,为简化 CAN 总线接口电路,提高平台硬件可靠性,本课题的主控芯片采用集成了 CAN 控制器的 AT90CAN128,并使用带隔离的 CAN 总线收发器 CTM1050 来实现 CANopen 接口电路。

### 4.2 相关元件介绍

#### 4.2.1 主控芯片 AT90CAN128

##### 1) AT90CAN128 的特点

AT90CAN128 是 Atmel 公司推出的 8 位单片机,它将 AVR 内核和支持 CAN2.0A/B 的 CAN 控制器集成在一个芯片中,适用于大多数应用在 CAN 网络之上的工业应用。

AT90CAN128 的主要特点有<sup>[41]</sup>:

- (1) 高效的 AVR 内核,16MIPS 的处理速度;
- (2) 丰富的指令集,以及 32 个通用工作寄存器;
- (3) 符合 ISO16845 标准的 CAN2.0A/B 控制器;
- (4) 128K 字节的 Flash、4K 字节的 EEPROM、4K 字节的 SRAM;
- (5) 4 个具有比较模式以及 PWM 功能的定时器/计数器;
- (6) 具有片内振荡器的可编程看门狗定时器;
- (7) 与 IEEE 1149.1 规范兼容的 JTAG 测试接口。

##### 2) AT90CAN128 的引脚

AT90CAN128 的外部引脚如图 4-1 所示:

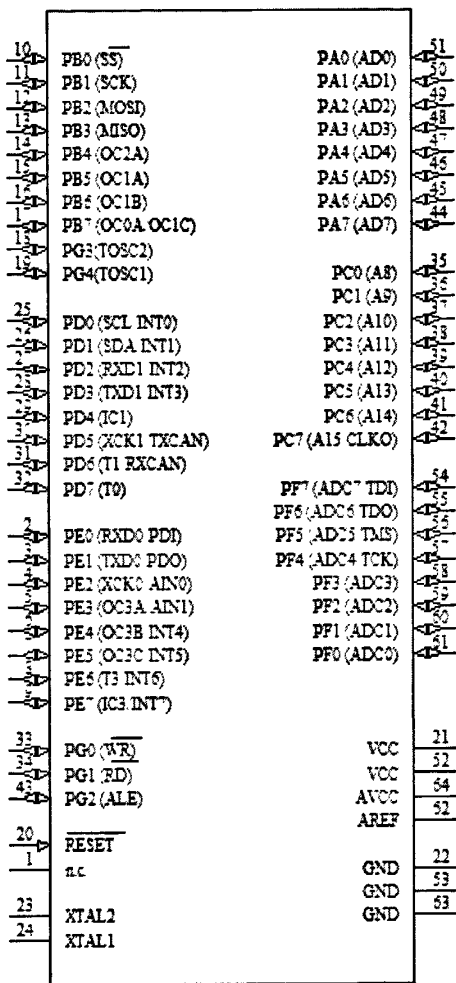


图 4-1 AT90CAN128 引脚

从上图可以看出，该芯片共有 64 条引脚，且大多数皆可复用，以实现不同的特殊功能。在本文中所用到的具体功能将在下一节中详细介绍。

### 4.2.2 隔离 CAN 收发器 CTM1050

#### 1) CTM1050 的特点

CTM1050 是广州致远电子有限公司生产的一款带隔离的高速 CAN 收发器模块，用于 CAN 控制器与 CAN 总线之间。该芯片内部集成了所有必需的 CAN 隔离及 CAN 收、发器件，用来将 CAN 控制器的逻辑电平转换为 CAN 总线的差分电平，此外还具有对 CAN 控制器与 CAN 总线之间的 2500V 直流隔离功能及静电释放保护功能。

CTM1050 的主要特点有<sup>[42]</sup>：

- (1) 具有隔离、静电释放保护功能，隔离电压为 DC 2500V；
- (2) 带有完全符合 ISO11898 标准的 CAN 收发器，通讯速率最高达 1Mbps；
- (3) 电磁辐射极低，电磁抗干扰性极高；
- (4) 高低温特性好，能满足工业级产品技术要求；
- (5) 无需外加其他元件就可直接使用。



2) CTM1050 的引脚及定义  
CTM1050 的引脚如图 4-2 所示：

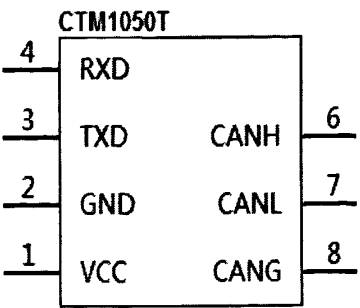


图 4-2 CTM1050 的引脚

CTM1050 的引脚定义如表 4-1 所示：

表 4-1 CTM1050 引脚定义

引脚号	引脚定义	引脚含义
1	V <sub>in</sub>	+5V 输入
2	GND	电源地
3	TXD	CAN 控制器发送端
4	RXD	CAN 控制器接收端
6	CANH	CANH 信号线连接端
7	CANL	CANL 信号线连接端
8	CANG	隔离电源输出地

4.3 硬件电路设计

目标平台的硬件结构如图 4-3 所示。

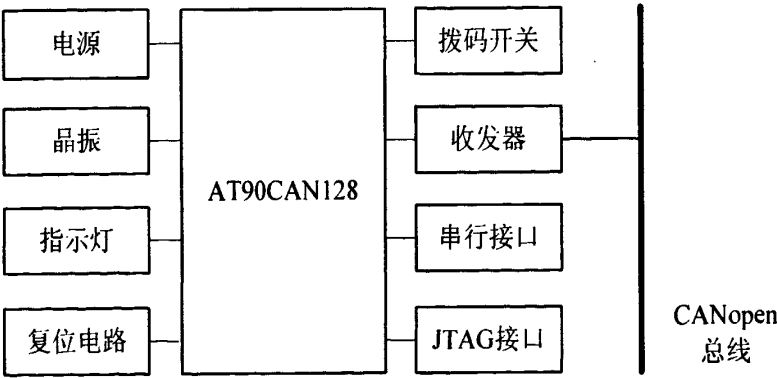


图 4-3 目标平台硬件结构

在本文中，使用 AT90CAN128 单片机作为主控制器，并承担 CAN 控制器功能。此外，围绕主控制器设计了电源模块，复位电路，8M 晶振，串行接口以及 JTAG 仿真器

接口等，还引出了单片机的 PORTA 端口，以便给后续研究留出扩充空间。主控制器部分电路如图 4-4 所示。

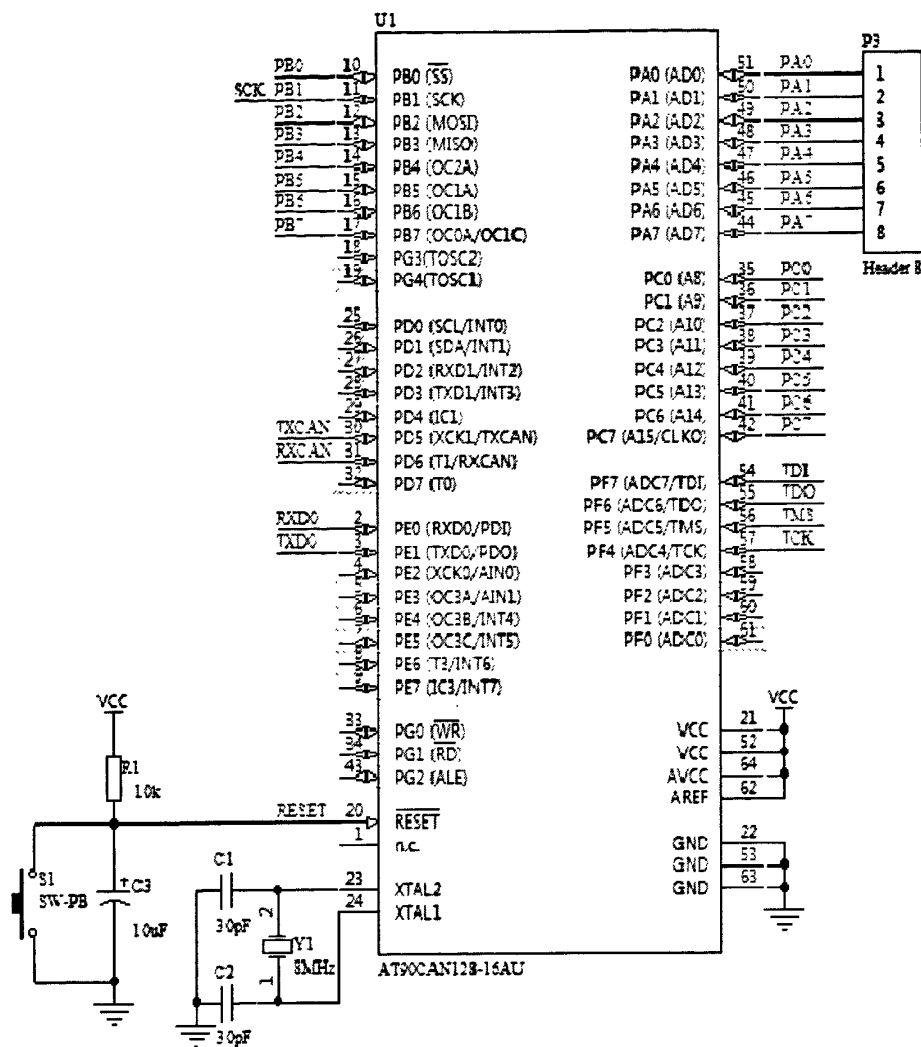


图 4-4 主控制器部分电路图

串行接口部分电路如图 4-5 所示。该部分电路使用了一片 MAX232CSE 芯片来进行电平的转换。

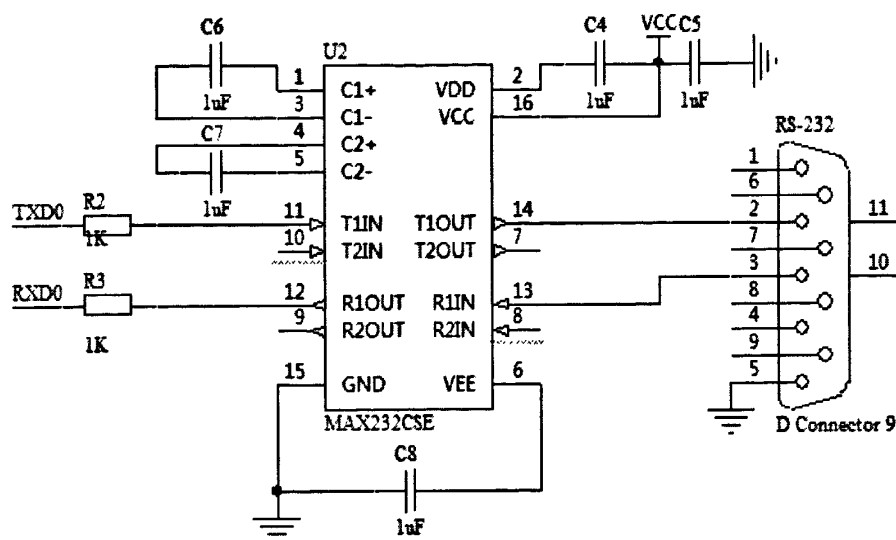


图 4-5 串行接口电路

JTAG 仿真器接口如图 4-6 所示。

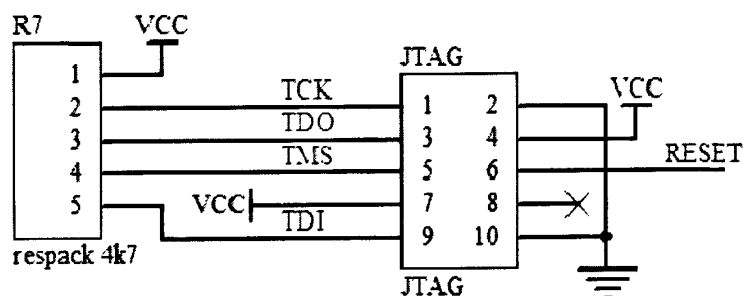


图 4-6 ISP 和 JTAG 接口电路

PORTB 端口使用低电平触发，作为 LED 输出，该部分电路如图 4-7 所示。

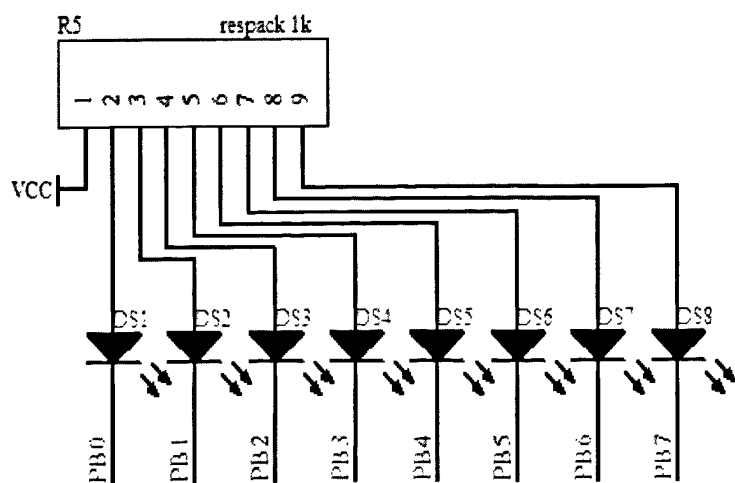


图 4-7 LED 显示电路图

PORTC 端口使用低电平触发的 8 位拨码开关来实现对 Node-ID 的设定，如图 4-8。

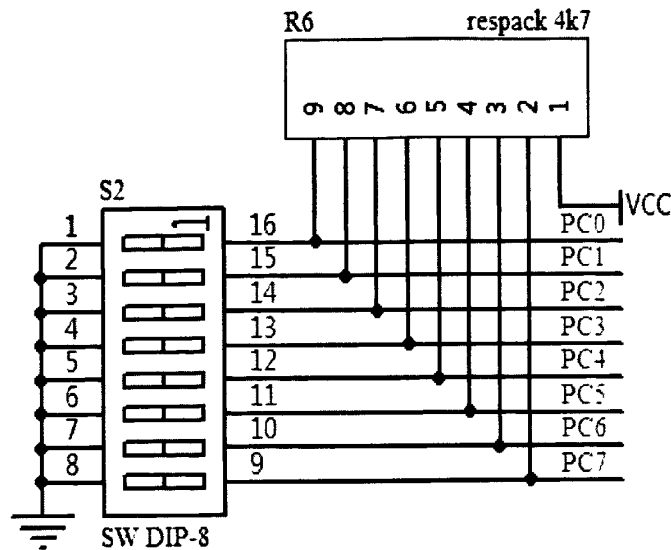


图 4-8 拨码开关电路图

本文中的 CAN 接口电路部分使用 CTM1050 收发器，见图 4-9。为了提高平台的稳定性和安全性，必须在 CAN 总线和 CAN 收发器之间设计光电隔离电路。在传统的设计方案中，要实现带隔离的 CAN 收发电路，不仅需要 CAN 收发器 TJA1050 或 PCA82C250，还需要在 CAN 收发器和 CAN 控制器的 TXCAN 和 RXCAN 端口之间加入高速光耦芯片 6N137 以及小功率 DC/DC 隔离，并搭配电阻等其他元器件才能实现<sup>[43]-[48]</sup>。但在本设计中，只利用了一片 CTM1050 接口芯片就实现了带隔离的 CAN 收发电路，从而简化了电路，方便了使用，同时降低了成本，提高了良率。

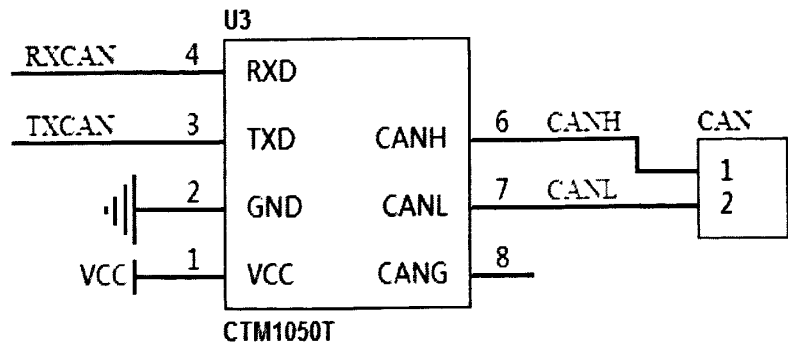


图 4-9 CAN 通信接口电路图

最终的目标平台 PCB 见图 4-10 所示。

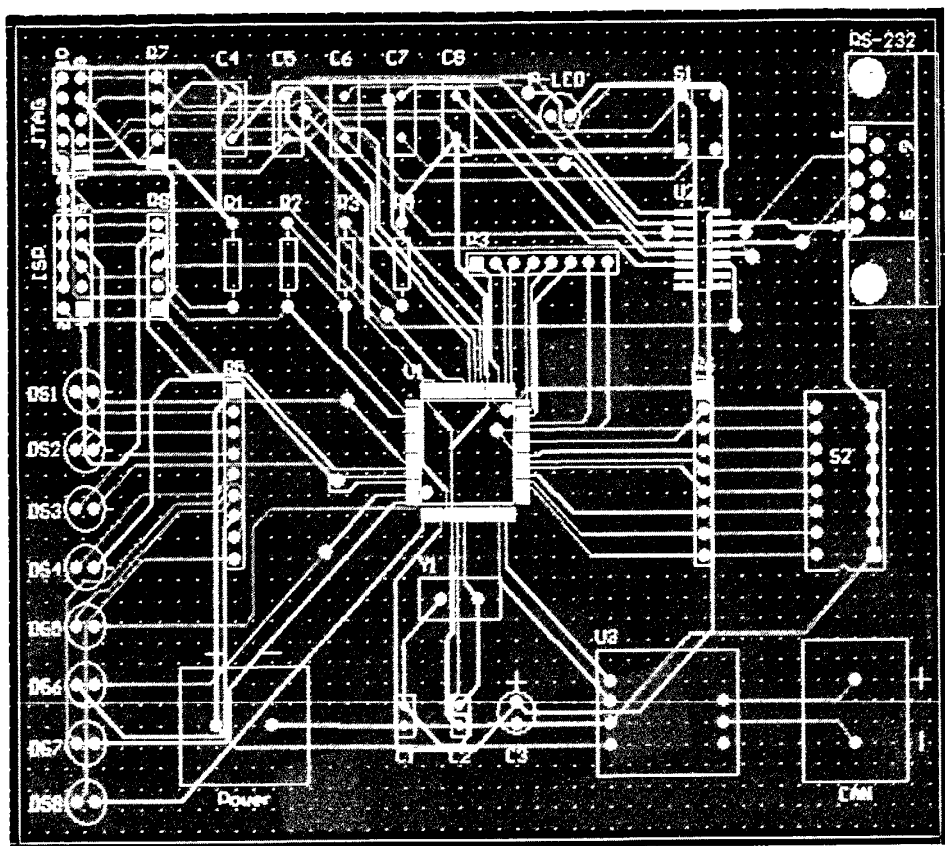


图 4-10 目标平台 PCB 电路

4.4 本章小结

本章对目标平台中使用的带有 CAN 控制器的单片机 AT90CAN128、隔离 CAN 收发器 CTM1050 的结构和功能做了简要介绍，给出了目标平台硬件结构图和电路图，并详细说明了各器件完成的功能及电路设计。

## 第五章 基于嵌入式系统的协议软件设计

### 5.1 可行性分析

所谓对程序的移植,是指通过对代码的修改,使该程序可以在选定的目标平台上运行<sup>[49]</sup>。本章的目的就是移植  $\mu\text{C}/\text{OS-II}$  V2.85 和 CanFestival 3.0 RC3 协议栈,并将协议栈作为一个任务封装到  $\mu\text{C}/\text{OS-II}$ ,使其可以在 AT90CAN128 上顺利运行。

要将  $\mu\text{C}/\text{OS-II}$  移植到某个微控制器上,则该微控制器必须满足如下要求<sup>[50]</sup>:

- 1) 控制器的 C 编译器能够产生可重入代码;
- 2) 控制器可以在程序中打开或者关闭中断;
- 3) 控制器支持中断,并且能产生定时中断;
- 4) 控制器支持能够容纳一定量数据的硬件堆栈;
- 5) 控制器有将堆栈指针和其他处理器寄存器存储、读出到堆栈或内存的指令。

本文选定的 AT90CAN128 单片机完全符合以上要求,所以将  $\mu\text{C}/\text{OS-II}$  移植到 AT90CAN128 上是完全可行的。

同时,移植 CanFestival 的条件是单片机必须具有 40K 以上的代码空间和 2K 以上的 RAM,而 AT90CAN128 具有 128K 字节的可编程 Flash 和 4K 字节的 RAM,因此可以选用该单片机作为 CanFestival 移植的目标控制器。

总之,本文的硬件系统完全满足移植条件。

### 5.2 开发工具简介

由于 CanFestival 的源代码是由 Linux 下的 GCC 编写,所以本课题需要使用一个与 GCC 相关的编译器来进行开发。GCC (GNU Compiler Collection, GNU 编译器套装),是一套由 GNU 所开发的,遵循 GPL 及 LGPL 许可证的开源软件编译器,是类 UNIX 操作系统的标准编译器,可以将多种语言编译成特定处理器的汇编代码。实际上,由于 GCC 在所有平台上都使用同一个前端处理程序,编译后会产生相同的代码,因此它实际上也是跨平台编译器的标准。

WinAVR 是一个高效、开源的开发工具,能在 Windows 平台下提供对 AVR 单片机的支持。它包括了 GCC 编译器,可以将 C/C++ 源代码编译成 AVR 汇编,并且编译生成的 HEX 和 COF 文件可以使用 AVRStudio 进行仿真。因此,本课题采用 WinAVR 的 20081205 版本作为开发工具。

5.3 μC/OS-II 在目标平台上的移植

5.3.1 需进行的工作

本节的移植对象是 2.85 版 μC/OS-II，目标平台是 AT90CAN128。  
μC/OS-II 作为一个微内核，只对计算机的处理器和硬件时钟进行了抽象的封装，而没有提供其他硬件抽象层，所以在移植 μC/OS-II 时，主要工作就是用 C 语言和汇编语言完成一些与处理器相关的代码，添加一个硬件的抽象层。而系统内核的各种服务文件，如 OS\_CORE.C、uC/OS-II.C 等与处理器无关，因此无需移植。需要移植的文件及其作用如表 5-1 所示。

表 5-1 移植 μC/OS-II 时需修改的文件

OS_CPU.H	有些内容在系统移植时要根据工程实际使用的处理器进行修改；有些无须修改，如数据类型的定义部分。
OS_CPU_I.H	包含在所有汇编语言文件中，用以声明中断服务程序。 该文件包含宏和 I/O 端口定义，必须移植。
OS_CPU_C.C	集中了所有与处理器相关的 C 语言代码模块，必须移植。
OS_DBG.C	提供了调试器功能。可以移植。
OS_CPU_A.S	集中了所有与处理器相关的汇编代码模块，必须移植。

5.3.2 OS\_CPU.H 的修改

OS\_CPU.H 中，主要定义了一些数据类型。由于 C 语言中的数据类型直接与处理器相关，为了保持良好的直观性和可移植性，μC/OS-II 中并没有直接采用 C 语言中的数据类型定义，而重新定义了一些整数数据类型。根据所选用的 AT90CAN128 单片机内核的特性，将这些数据重新定义如下：

```
typedef unsigned char    BOOLEAN;           // 布尔型数
typedef unsigned char    INT8U;              // 无符号 8 位数
typedef signed char      INT8S;              // 带符号 8 位数
typedef unsigned int     INT16U;             // 无符号 16 位数
typedef signed int       INT16S;             // 带符号 16 位数
typedef unsigned long int INT32U;            // 无符号 32 位数
typedef signed long int  INT32S;            // 带符号 32 位数
typedef float            FP32;               // 单精度浮点数
typedef unsigned char    OS_STK;            // 堆栈入口宽度为 8 位
typedef unsigned char    OS_CPU_SR;         // 状态寄存器为 8 位
```

之后，需要在程序中选择开关中断的实现方法。为保护临界段代码的执行过程不受多任务或中断服务程序的破坏，μC/OS-II 定义了两个宏 OS\_ENTER\_CRITICAL()和

OS\_EXIT\_CRITICAL()进行关中断和开中断的操作，它们都可以通过设置状态寄存器 SREG 中的中断屏蔽位来实现。其中，OS\_ENTER\_CRITICAL()可以清除中断屏蔽位，从而屏蔽中断；OS\_EXIT\_CRITICAL()写入中断屏蔽位，从而允许中断。此外，为了尽量缩短中断中禁止时间， $\mu\text{C}/\text{OS-II}$  还可以通过将宏 OS\_CRITICAL\_METHOD 分别定义为 1、2、3 来选择三种开关中断方式。在本课题中选择第一种方式，即用简单指令开关中断。因此需要编写如下代码：

```
#define OS_CRITICAL_METHOD 1
```

最后，我们需要定义堆栈的增长方向。 $\mu\text{C}/\text{OS-II}$  中支持堆栈从上往下生长和从下往上增长，通过定义 OS\_STK\_GROWTH 的值来改变。由于 AVR 单片机的堆栈为从上往下增长，所以需要在 OS\_CPU.H 中需要将 OS\_STK\_GROWTH 定义为 1，即：

```
#define OS_STK_GROWTH 1
```

在  $\mu\text{C}/\text{OS-II}$  中，OS\_TASK\_SW()的功能是实现任务切换，因此在进行宏定义时需要直接将其定义为任务级切换函数 OSCtxSw()，代码为：

```
#define OS_TASK_SW() OSCtxSw()
```

### 5.3.3 OS\_CPU\_C.C 的修改

文件 OS\_CPU\_C.C 中有 10 个函数，分别为：OSTaskStkInit()、OSInitHookBegin()、OSInitHookEnd()、OSTaskCreateHook()、OSTaskDelHook()、OSTaskSwHook()、OSTaskStatHook()、OSTimeTickHook()、OSTCBInitHook()和 OSTaskIdleHook()。实际上，除 OSTaskStkInit()之外，其他 9 个函数是用户自己定义的。如果不使用它们，只需通过将 OS\_CFG.H 里的 OS\_CPU\_HOOKS\_EN 定义为 0，就可以在 OS\_CPU\_C.C 中只声明它们而不编写具体函数。本课题中并不使用这些函数，因此将它们定义为空函数。

因此，涉及到本文，OS\_CPU\_C.C 文件中需要修改的只有 OSTaskStkInit()函数。函数 OSTaskStkInit()由 OSTaskCreate()或 OSTaskCreateExt()调用，用来初始化任务堆栈，使其恢复为发生过一次中断后时任务的堆栈结构。对于 AT90CAN128，在发生中断后，除了自动保存的程序计数器 PC 之外，还需要保存状态寄存器 SREG、通用寄存器 R0 至 R31 及 SP 的值。WinAVR 规定，R24、R25 放置的是向任务传递的参数，所以寄存器需要特殊处理。

本段函数如下所示：

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT16U
opt)
{
    OS_STK* stk;
    opt=opt;      //由于 opt 未使用，所以编写此语句防止编译器报警
    stk=ptos;     //栈顶指针
    *stk-- = (INT8U)(((INT16U)task)&0xFF);
    *stk-- = (INT8U)(((INT16U)task)>>8);
    *stk-- = (INT8U)0x00;      //R0 = 0x00
```



```

*stk-- = (INT8U)0x01;          //R1 = 0x00
*stk-- = (INT8U)0x02;          //R2 = 0x00
.....
*stk-- = (INT8U)(((INT16U)pdata)&0xFF);
*stk-- = (INT8U)(((INT16U)pdata)>>8); // R24、R25 中为任务传递的参数
*stk-- = (INT8U)0x26;
*stk-- = (INT8U)0x27;
.....
*stk-- = (INT8U)0x31;
*stk-- = (INT8U)0x80;          //SREG=0x80, 开全局中断
return ((OS_STK *)stk);
}

```

### 5.3.4 OS\_CPU\_A.S 的修改

移植  $\mu\text{C}/\text{OS-II}$  时, 需要改写 OS\_CPU\_A.ASM 中的以下函数: OSStartHighRdy()、OSCtSw()、OSIntCtSw()、OSTickISR()。这些函数与单片机底层密切相关, 需要用汇编语言编写。

#### 1) OSStartHighRdy()

使就绪状态的任务开始运行的函数是 OSStart(), 而 OSStartHighRdy() 就是由 OSStart() 函数调用, 用来运行优先级最高的就绪态任务。

由于用户在恢复最高优先级任务的寄存器的時候, 需要检查 OSRunning 的状态来知道哪类函数在调用它, 因此函数 OSStartHighRdy() 必须调用函数 OSTaskSwHook()。当 OSRunning 为 FALSE 时, 表示 OSStartHighRdy() 在调用它; OSRunning 为 TRUE 时, 表示正常的任务切换在调用它。

编写的函数如下:

OSStartHighRdy:

```

.if OS_CPU_HOOKS_EN > 0
    CALL    OSTaskSwHook        ;调用任务切换接口函数 OSTaskSwHook()
#endif
    LDS     R16,OSRunning        ;设置标识系统开始运行的变量
    INC     R16
    STS     OSRunning,R16        ;OSRunning=TRUE, 表明多任务调度开始

;让 Z 指针指向高优先级任务的任务控制块 OSTCBHighRdy
    LDS     R30,OSTCBHighRdy    ;R30 指向优先级最高的任务的任务控制块
    LDS     R31,OSTCBHighRdy+1  ;准备运行

```

;高优先级任务的栈顶指针存入 Y 指针

```

LD      R28,Z+
OUT     _SFR_IO_ADDR(SPL),R28
LD      R29,Z+
OUT     _SFR_IO_ADDR(SPH),R29      ;从任务控制块中获取堆栈指针
POPRS   ;更新所有寄存器内容
RET     ;开始执行当前任务

```

## 2) OSCtxSw()

OSCtxSw()是一个任务级任务切换函数，只能在任务中被调用。它的任务是：保存当前任务现场、保存当前任务的任务栈指针到当前任务的任务控制块、切换最高优先级任务为当前任务、使 SP 指向最高优先级任务的任务栈的栈顶、恢复新任务的运行环境。由于 ATmega128 不支持软中断指令，所以必须通过汇编程序来模拟中断，代码如下：

OSCtxSw:

```

PUSHRS   ;保存任务当前环境
;保存当前任务软件堆栈的栈顶指针到当前任务的任务控制块 TCB
LDS      R30,OSTCBCur      ;Z 指针指向当前任务的任务控制块
LDS      R31,OSTCBCur+1
IN       r28,_SFR_IO_ADDR(SPL)
ST       Z+,R28            ;保存堆栈指针到 Y (R29:R28)寄存器中
IN       r29,_SFR_IO_ADDR(SPH)
ST       Z+,R29

#if OS_CPU_HOOKS_EN > 0
CALL     OSTaskSwHook      ;调用 OSTaskSwHook
#endif

LDS      R16,OSPrioHighRdy  ;把最高优先级的任务切换为当前任务
STS      OSPrioCur,R16

;将最高优先级任务的任务控制块切换为当前任务的任务控制块
LDS      R30,OSTCBHighRdy  ;Z 指针指向最高优先级任务的任务控制块
LDS      R31,OSTCBHighRdy+1 ;准备运行任务
STS      OSTCBCur,R30      ;OSTCBCur = OSTCBHighRdy
STS      OSTCBCur+1,R31
LD       R28,Z+            ;得到最高优先级任务的软件堆栈栈顶指针，并存入 Y
OUT      _SFR_IO_ADDR(SPL),R28
LD       R29,Z+
OUT      _SFR_IO_ADDR(SPH),R29
POPRS   ;更新所有寄存器内容
RETI    ;任务切换

```

## 3) OSIntCtxSw()

OSIntCtxSw()是中断级的任务切换函数。在  $\mu\text{C}/\text{OS-II}$  中, 由于中断的产生可能会引起任务切换, 因此在中断服务程序的最后会调用 OSIntExit()函数检查任务就绪状态。而 OSIntExit()通过调用 OSIntCtxSw()来从 ISR 中执行切换功能。该函数的代码如下:

OSIntCtxSw:

#if OS\_CPU\_HOOKS\_EN > 0

CALL OSTaskSwHook ;调用 OSTaskSwHook

#endif

LDS R16,OSPrioHighRdy ;把最高优先级的任务切换为当前任务

STS OSPrioCur,R16

;把最高优先级任务的任务控制块切换为当前任务的任务控制块

LDS R30,OSTCBHighRdy ;Z 指针指向最高优先级任务的任务控制块

LDS R31,OSTCBHighRdy+1 ;准备运行任务

STS OSTCBCur,R30 ;OSTCBCur = OSTCBHighRdy

STS OSTCBCur+1,R31

LD R28,Z+ ;得到最高优先级任务的软件堆栈栈顶指针, 并存入 Y

OUT \_SFR\_IO\_ADDR(SPL),R28

LD R29,Z+

OUT \_SFR\_IO\_ADDR(SPH),R29

POPRS ;更新所有寄存器内容

RET

#### 4) OSTickISR()

OSTickISR()是时钟节拍中断服务函数, 用户必须在调用 OSStart()后立刻调用该函数以允许时钟节拍中断。该函数的代码如下:

TIMER0\_COMP\_vect:

OSTickISR:

PUSHRS ;保存所有寄存器值

LDS R16,OSIntNesting ;通知  $\mu\text{C}/\text{OS-II}$  中断服务程序

INC R16

STS OSIntNesting,R16 ;中断嵌套计数器加 1

CLZ ;清零 Z 标志位, 为下面的比较做好准备

CPI R16,1 ;比较 OSIntNesting 是否为 1

BREQ OSTickISR2 ;如果是 1, 则跳转至 OSTickISR2

SEI ;开中断

CALL OSTimeTick ;调用 OSTimeTick 函数更新时钟

CALL OSIntExit ;调用 OSIntExit, 通知系统 ISR 末端

POPRS ;更新所有寄存器值

RET;任务切换

5. 4 CanFestival 在目标平台上的移植

5.4.1 需进行的工作

在移植 CanFestival 协议栈时，需要修改的文件包括：include 文件夹中与目标处理器相关的头文件、drivers 文件夹中的目标控制器驱动文件以及各相关文件夹中的 Makefile 文件等。需要移植的文件夹如表 5-2 所示：

表 5-2 需要移植的文件夹

路径	说明
CanFestival/include	目标处理器硬件相关头文件
CanFestival/drivers	目标处理器的驱动文件
CanFestival/examples	应用程序和对象字典代码

5.4.2 目标处理器相关头文件的编写

CanFestival/include 文件夹中包含与处理器硬件相关的.h 文件。在该文件夹下新建文件夹 AVR，然后把 CanFestival/include/win32 下的文件全部拷贝过来，并进行修改。

为防止头文件中的宏被多次定义，所有文件中都要采用#ifndef、#define 与#endif 来进行条件编译，例如：

```
#ifndef __SAMPLE__
#define __SAMPLE__
/*在此处进行宏定义*/
#endif
```

1) applicfg.h

由于与 μC/OS-II 相同的理由，CanFestival 中同样没有直接采用 C 语言中的数据类型而定义了一些整数数据类型。因此，在 CanFestival 的移植过程中，首先还是需要定义数据类型。代码如下：

```
typedef char INT8S;
typedef unsigned char INT8U;
typedef signed int INT16S;
typedef unsigned int INT16U;
typedef short INT16S;
typedef unsigned short INT16U;
typedef long INT32S;
```

```
typedef unsigned long INT32U;
typedef float REAL32;
typedef double REAL64;
```

## 2) canDef.h

Atmel 公司在其官方网站上提供了 AT90CAN128/64/32 的 CAN 底层驱动。因此，我们可以直接使用这些 CAN 底层驱动的原型及宏，从而更快捷、更准确地进行文件的移植，如：

```
#define CAN_PORT_IN      PIND
#define CAN_PORT_DIR     DDRD
#define CAN_PORT_OUT     PORTD
#define CAN_INPUT_PIN    6
#define CAN_OUTPUT_PIN   5
#define ERR_GEN_MSK ((1<<SERG)|(1<<CERG)|(1<<FERG)|(1<<AERG))
#define INT_GEN_MSK ((1<<BOFFIT)|(1<<BXOK)|(ERR_GEN_MSK))
.....
```

## 3) canDriver.h

本文件主要定义了几个与硬件相关的宏，并声明了一些用户程序需要调用的函数。这些函数有：

```
unsigned char canInit(unsigned int bitrate);
unsigned char canSend(Message *msg);
unsigned char canReceive(Message *msg);
```

下面是对接收信箱和发送信箱进行宏定义的程序。需要注意的是，由于其中用到了 canDef.h 中所定义的宏，因此需要包含该文件。

```
#include "canDef.h"
#define NB_RX_MOB 13
#define NB_TX_MOB (NB_MOB - NB_RX_MOB) //发送信箱长度
```

由于 CAN 报文为 8 字节，所以在这段程序中接收信箱的长度最小值为 8。在 AT90CAN128 中，信箱 (MOB) 有 15 个，分别为 MOB0-MOB14。

## 4) config.h

该函数中包含了对 GCC 编译器的配置，晶振频率也在此定义。由于我们在硬件设计中选择了 12MHz 的晶振，所以我们需要修改如下语句：

```
#define FOSC 12000 //12M 晶振
```

## 5) timerscfg.h

该文件是与时钟功能直接相关的头文件。这里有一点需要注意，无论使用哪种微控制器，当 TIMEVAL 少于 32 位时，定时器都不能工作。所以这里必须将 TIMEVAL 定义为 unsigned long，即 INT32U，语句如下：

```
#define TIMEVAL INT32U
```

### 5.4.3 目标控制器驱动文件的编写

该文件夹中的文件是 CanFestival 在所选目标处理器平台下的驱动文件。与头文件的移植过程相似，这些文件的移植也需要在已有代码的基础上重新编写。

#### 1) canDriver.c

该文件夹中的程序需要头文件 canDriver.h，所以必须在开始包含该头文件。

在 canDriver.c 中，必须进行的工作是编写函数 canInit()、canSend() 以及 canReceive()。其中，canInit() 的作用是对硬件进行初始化，并且为 CANopen 协议栈启动定时器；canSend() 的作用是从 CANopen 协议栈发送 CAN 信息；canReceive() 的作用是向 CANopen 协议栈传送已接收到的 CAN 消息。

由于代码较长，这里仅以 canReceive() 为例介绍函数的编写。该函数的参数为指向 AT90CAN128 上 CAN 控制器接收到的报文的指针 m，返回值为 1 或 0，如果报文成功接收，返回值为 1，反之为 0。

```
unsigned char canReceive(Message *msg)
{
    unsigned char i;

    if(msg_received == 0)
        return 0;                                // 接收报文失败，返回 0

    for(i = 0; i < RX_MOB; i++)
    {
        // 寻找报文的第一位
    }
    if(i < RX_MOB)                                // 找到报文
    {
        // 读取 COB-ID
        // 获得传输请求
        // 读取报文长度
        // 从 CAN 控制器信箱中逐位读取信息
        // 重置控制器信箱，准备下一次接收
        // 成功接收报文，返回 1
    }
    else
    {
        msg_received = 0;                        // 重置计数器
        return 0;                                // 接收报文失败，返回 0
    }
}
```

#### 2) timerSet.c

该文件定义了 CANopen 协议栈中的时钟在 AT90CAN128 平台上的实现，本课题中使用了 AT90CAN128 中的 16 位定时器 3。在该文件中，需要编写 `initTimer()`、`setTimer()`、`tellTimer()` 三个函数。`initTimer()` 用来初始化定时器并开中断；`setTimer()` 用来设置定时器；`tellTimer()` 用来通知协议栈本次调用共花费的时间。这里以 `initTimer()` 为例介绍函数的编写。

为了使用方便，为输出比较寄存器 OCR3B 利用宏定义设置一个别名：

```
#define TimerAlarm    OCR3B
void initTimer(void)
{
    TimerAlarm = 0;           // OCR3B 寄存器设为 0
    TCCR3B = 1 << CS31 | 1 << CS30; // TCCR3B 的第 0 位和第 1 位设置为 1
    TIMSK3 = 1 << OCIE3B;      // 开中断
}
```

TCCR3B 是定时器 3 的控制寄存器 B，其结构如图 5-1 所示：

	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	-	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
读/写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

图 5-1 定时器 3 控制寄存器 TCCR3B

其中，Bit 7 (ICNC3)为输入捕捉噪声抑制器位，置位该位后将使能输入捕捉噪声抑制功能；Bit 6 (ICES3)为输入捕捉触发沿选择位，该位选择使用 ICP3 上的哪个边沿触发捕获事件；Bit 5 为保留位，在写 TCCR3B 时，该位必须写入“0”；Bit 4 (WGM33)和 Bit 3 (WGM32) 可选择波形发生模式位；Bit 2 (CS32)、Bit 1 (CS31)、Bit 0 (CS30)为时钟选择位，这 3 位用于选择 T/C 的时钟源，如表 5-3 所示。

表 5-3 时钟选择位描述

CS32	CS31	CS30	说明
0	0	0	无时钟源 (T/C 停止)
0	0	1	$clk_1/1$ (无预分频)
0	1	0	$clk_1/8$ (米白预分频器)
0	1	1	$clk_1/64$ (米白预分频器)
1	0	0	$clk_1/256$ (米白预分频器)
1	0	1	$clk_1/1024$ (米白预分频器)
1	1	0	外部 Tn 引脚，下降沿驱动
1	1	1	外部 Tn 引脚，上升沿驱动

### 5.4.4 协议功能应用程序的编写

CanFestival/examples 中的代码是应用程序和对象字典，协议所要实现的功能在这里定义，需要完全重新编写。

#### 1) canSlave.c

该文件用来实现协议功能。它在初始化系统后，即进入一个无限循环，在这个循环内调用其他函数进行信息的处理。

首先要编写系统初始化的程序。

```
void sysInit()
{
    PORTC=0xFF;           // PORT C 输出低电平
    DDRC=0x00;           // PORT C 的作用是控制拨码开关
    PORTD=0x2C;          // PD5 为输出，PD6 为输入
    DDRD=0x2A;           // PD5 为 TXCAN，PD6 为 RXCAN
    /*定时器 0 为主时钟，在 CTC 模式下工作*/
    TCCR0A |= 1 << WGM01 | 1 << CS01 | 1 << CS00;
    TIMSK0 = 1 << OCIE0A;
    OCR0A = (unsigned char)(F_CPU / 64 * CYCLE_TIME/1000000 - 1); //更新时钟
}
```

最后，就要编写带有限循环的任务函数：

```
void canSlave(void *pdata)           // 任务函数
{
    pdata=pdata;    //建立任务时的参数。由于没有用到该参数，由此语句防止报错
    OSStatInit();   //初始化 μC/OS-II 中的统计任务
    sysInit();      // 系统初始化
    initTimer();    // 时钟初始化
    nodeId = readID(); // 读取 Node-ID
    setNodeID (nodeID); // 设置 Node-ID
    setState(Initialisation); // 初始化协议栈状态

    while(1)        // 主循环开始
    {
        /*检查 Node-ID 是否更改*/
        if(!(nodeID == readID()))
        {
            // 若有更改，保存新的 Node-ID
            // 停止节点
            // 更改 node ID
        }
    }
}
```



```
        // 置为预操作状态
    }
    /*消息处理*/
    if (canReceive(&msg))                // 接收到消息
        canDispatch(&msg);              // 处理消息
    }
}
```

5.4.5 对象字典的编写

要实现 CANopen 协议意味着必须定义一个对象字典，涉及到本文中，就必须编写 ObjDict.c 文件。该文件包含对象字典的定义，并且还提供了索引表，用来帮助协议栈直接访问对象字典入口。

在编辑对象字典时，可以使用 CanFestival/objdictgen 文件夹中的对象字典编辑器来进行操作。该编辑器是一个 Python 应用程序，它依赖于 WxPython 来实现所在平台上的 GUI。因此，Windows 用户必须安装 Python（2.4 版以上）和 WxPython（2.6.3.2 版本以上）才可以使用该程序。本文使用了 Python 2.6.4 和 WxPython 2.8.10.1 版本，安装这两个软件后，就能运行对象字典编辑器程序。

图 5-2 给出了对象字典编辑器的主界面。在该程序中，上半部分可以选择对象字典中的某一区域，左下部分为这一区域对象字典中的索引，右下部分为子索引，可以在这里进行具体的修改。修改完成后，可以使用该软件的生成对象字典命令来自动生成对象字典的 C 语言代码。

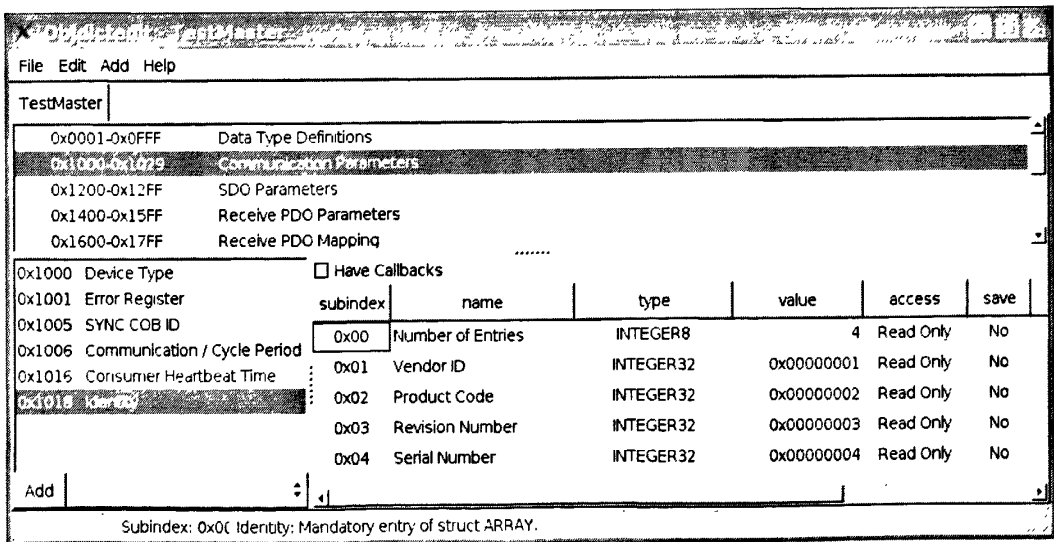


图 5-2 对象字典编辑器

在对象字典编辑器中，可以建立一个全新的对象字典，也可以编辑已有的对象字典代码。本文选择建立一个新对象字典，在该程序中可以修改名称、Node-ID、类型，并可选择所要添加的索引并进行修改，如图 5-3 所示。修改完成后，可以通过编辑器中的

Build Dictionary 命令生成所需对象字典的 ObjDict.c 文件和 ObjDict.h 文件。本文中定义的部分对象字典条目如表 5-4 所示。

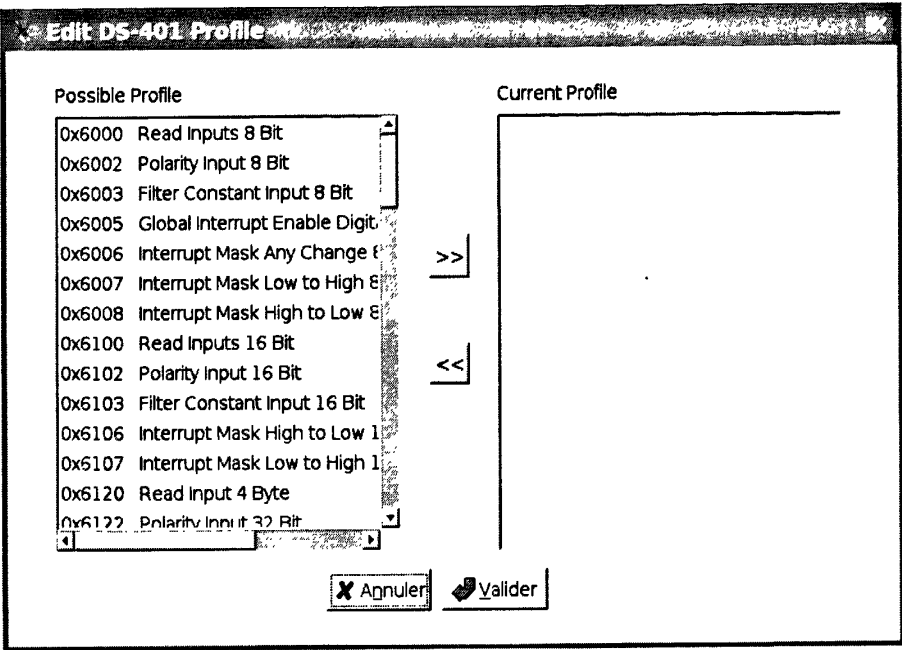


图 5-3 修改索引

表 5-4 部分对象字典条目

索引	子索引	索引值	含义
0x1000	-	0x30191	设备类型
0x1001	-	0x0	错误注册
0x1005	-	0x80	同步 COB-ID
0x1014	-	0x80	应急 COB-ID
0x1018	1	0x01	供应商 ID
	2	0x02	设备代码
	3	0x03	修订号
	4	0x04	序列
.....	.....	.....	.....

5.5 CanFestival 在 μC/OS-II 上的封装

将 CanFestival 代码移植到操作系统上，需要定义一个主函数。该主函数调用了三个函数，分别进行初始化任务、创建任务、启动任务的工作。代码如下：

```
void main()
{
    OSInit();
    OSTaskCreate(
        canSlave,
        (void*)0,
        &canSlaveStk[canFestivalStk_Size-1],
        canSlave_PRIO
    );
    OSStart();
}
```

其中，OSInit()的作用是对  $\mu\text{C}/\text{OS-II}$  的所有全局变量和数据结构进行初始化，同时创建空闲任务 OSTaskIdle，并赋予它最低的优先级和永远的就绪状态。

OSTaskCreate()用来创建任务，给任务代码分配一个任务控制块，并通过任务控制块把任务代码和任务堆栈关联起来形成一个完整的任务。主要完成三项工作：任务堆栈的初始化、任务控制块的初始化和任务调度。OSTaskCreate()函数有 4 个参数，分别是指向任务的指针 canSlave、传递给任务的参数(void\*)0、指向任务堆栈栈顶的指针 &canSlaveStk[canFestivalStk\_Size-1]和任务的优先级 canSlave\_PRIO。其中，由于 AVR 单片机的堆栈是从上向下生长的，因此任务堆栈栈顶地址为堆栈的最末尾空间。

最后，需要调用函数 OSStart()来启动调度任务。

## 5.6 Makefile 文件的编写

Makefile文件是GCC中的特色之处。它描述了整个工程的编译、连接等规则，规定了工程中的哪些源文件需要编译、如何编译、编译顺序、需要创建哪些库文件、如何创建这些库文件以及如何产生目标可执行文件等规则<sup>[51]</sup>。当用户编写好Makefile文件之后，就可以通过make命令来解释Makefile文件中的指令，从而自动编译源文件并得到目标文件。

由于Makefile有自己的书写格式、关键字、函数，因此对于初学者来说较难上手<sup>[52]</sup>。因此我们可以使用工具来帮助我们初步编写Makefile，如WinAVR中提供的MFile。

MFile的界面如图5-4所示。

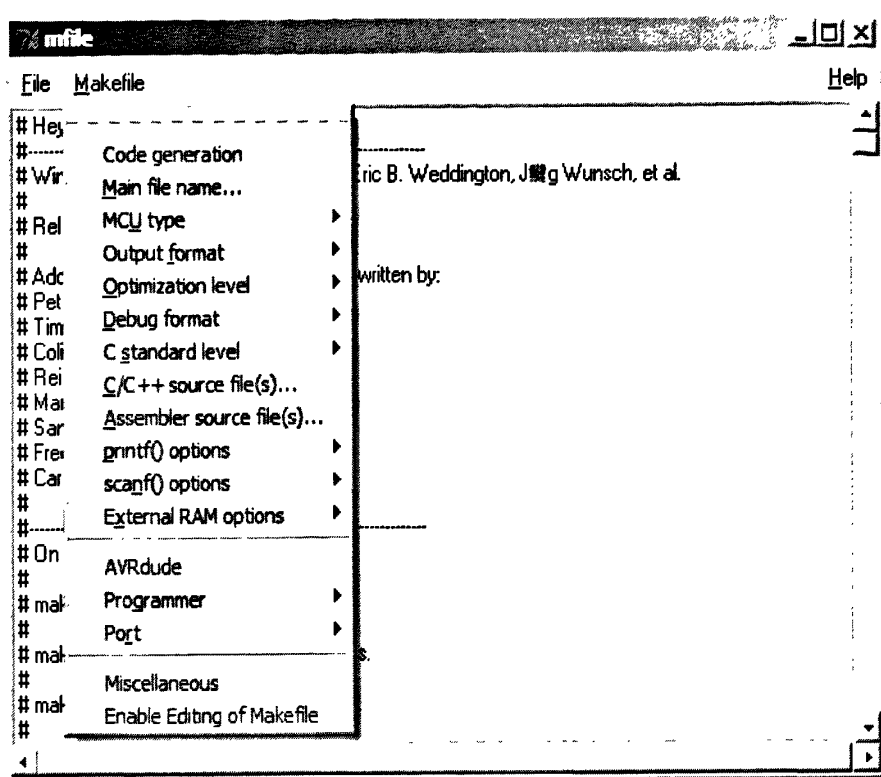


图 5-4 MFile 程序界面

通过设置Makefile菜单中的选项，可以进行一些基本的设置。步骤如下：

1) Main File name

此处输入所需输出目标文件的文件名CanSlave。不需要输入扩展名。

2) MCU type

在弹出菜单中选择所选择的单片机AT90CAN128。

3) Output format

选择所期望的最终输出格式，本文选择ihex。

4) Optimization level

优化级别选择，可选择0，1，2，3，s。本文中选择s。

5) Debug format

调试格式选择。该选项需要根据所使用的AVR Studio的版本来选择，由于本文使用AVR Studio 4.12进行调试，因此本选项选择ELF/DWAF-2 (AVR Studio 4.11+)。

6) C standard level

选择gnu99。

7) C/C++ source file(s)

加入工程中的所有C/C++源文件，多个文件之间使用空格隔开。

8) Assembler source file(s)

加入工程中的所有汇编语言源文件，需要以大写的S为后缀，多个文件之间使用空格隔开。

9) printf() options

选择none/standard。

10) scanf() options

选择none/standard。

11) External RAM options

由于未使用外部RAM，因此选择none。

设置完成后,将Makefile保存到CanFestival/examples/CanSlave目录下,选择WinAVR中Tools菜单的Make All命令进行编译,得到目标文件CanSlave.hex,编译结果见图5-5。

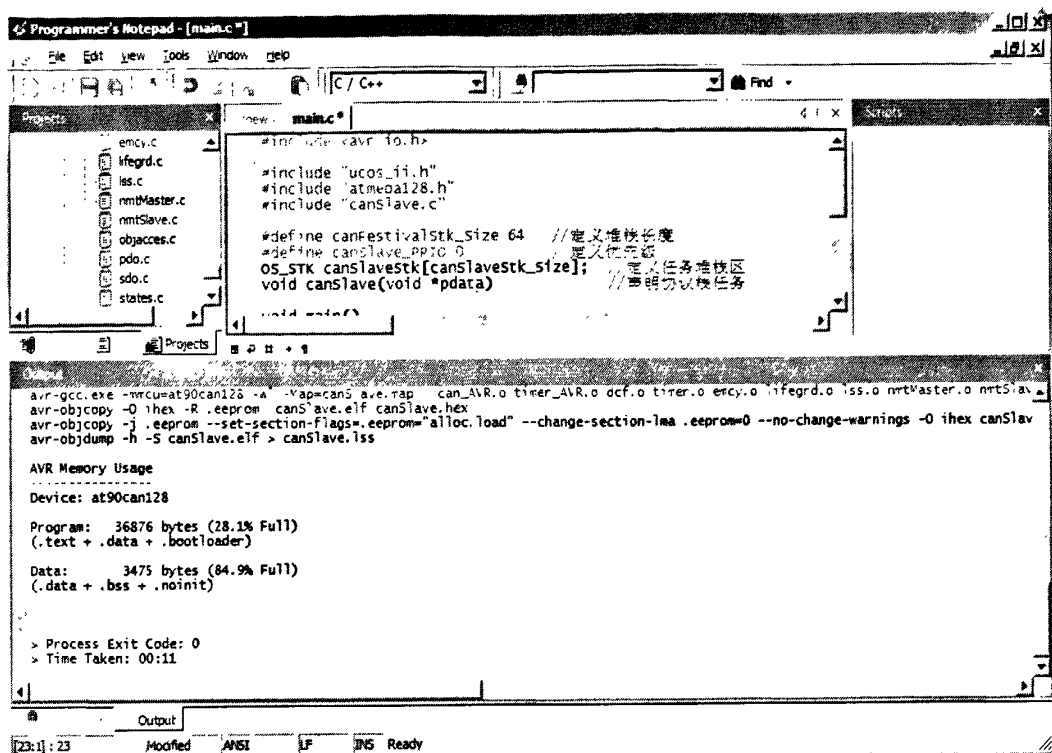


图 5-5 目标文件编译结果

## 5.7 本章小结

本章详细介绍了 CANopen 协议软件的设计过程。首先分析了开发的可行性，并介绍了开发工具 WinAVR，其次详细介绍了  $\mu\text{C}/\text{OS-II}$  和 CanFestival 在 AT90CAN128 平台上的移植方法，最后介绍了将 CanFestival 作为一个任务在  $\mu\text{C}/\text{OS-II}$  中的封装方法和 Makefile 文件的编写方法。

## 第六章 通信接口有效性实验

作为基于嵌入式系统的 CANopen 协议研究的重要方面,本章利用 Woodhead 公司生产的 BradCommunications Interface Card for CANopen PCU-CANIO 接口卡搭建了一个对于 CANopen 接口通信有效性的实验平台,并且按照此方案对本文开发的 CANopen 协议通信接口进行了实验,以证明研究的有效性和可行性。

### 6.1 有效性实验方案

使用由 Woodhead 公司出品的 CANopen 接口卡 PCU-CANIO 作为主节点,并将本文中开发的 CANopen 通信接口软硬件平台作为从节点,在此基础上,使用接口卡所附带的应用软件 applicomIO 2.3 作为配置及监控工具,就可以建立一个 CANopen 网络实验平台。该实验平台的结构如图 6-1 所示:

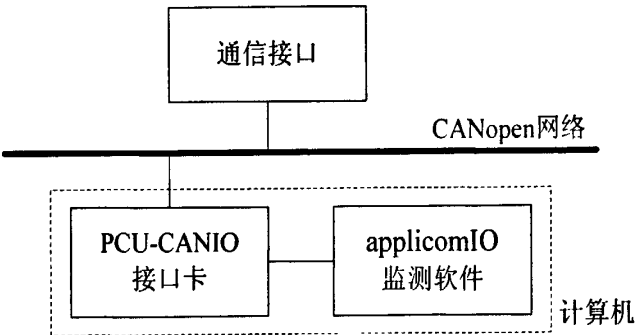


图 6-1 CANopen 网络实验平台

BradCommunications Interface Card for CANopen PCU-CANIO 接口卡是由 Woodhead 公司开发的工业 CANopen 通信卡。它使用 PCI 接口接入计算机,并采用 Philips 公司的 SJA1000 作为 CAN 控制器,具有完整的通信功能。PCU-CANIO 接口卡支持 CiA DS-301 v4.0、DS-302 v4.02 通信子协议和 DSP-401、DSP-402、DSP-403、DSP-404、DSP-406 设备子协议,通过该板卡可快速而经济地建立工业用 CANopen 网络<sup>[53]</sup>。此外,该板卡使用 applicomIO 2.3 SP4 软件进行板卡驱动,以及 CANopen 网络的配置和监测。

### 6.2 实验过程与结果

利用这一方案,就可以对本文所开发的 CANopen 通信接口功能进行有效性实验。

在计算机中安装 BradCommunications Interface Card for CANopen PCU-CANIO 板卡,并安装 applicomIO 2.3 SP4 应用软件。在安装该软件时,需要在选择协议栈界面选

择 CANopen 协议栈，从而使软件支持 CANopen 协议，如图 6-2 所示。

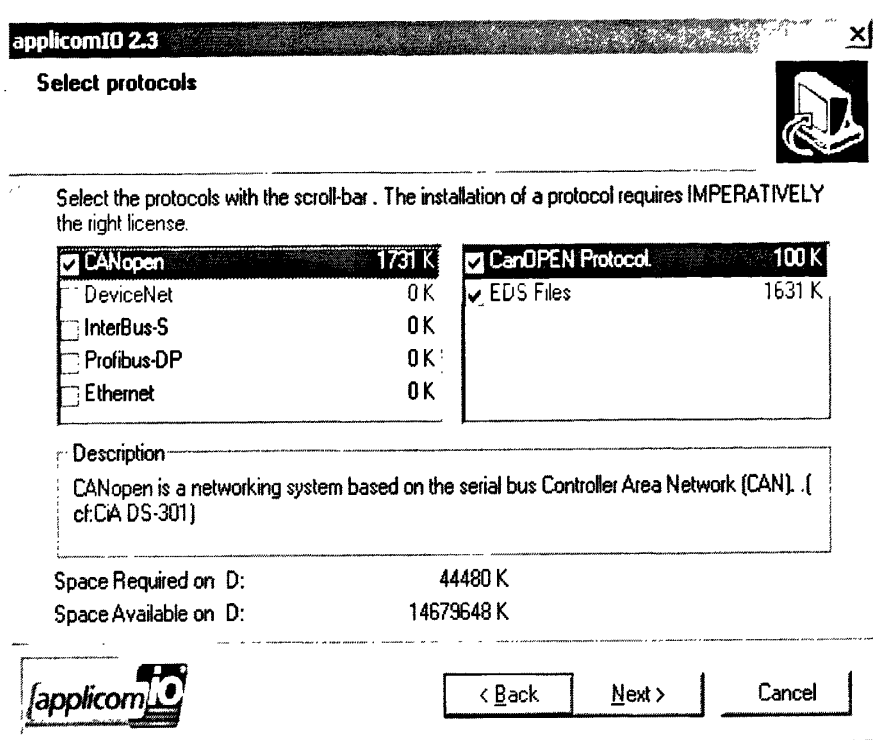


图 6-2 选择 CANopen 协议栈

在 applicomIO 的主界面中，在 Board configuration 上点击右键，选择 Add Board 并按照提示操作，就可以添加已在计算机中安装的 PCU-CANIO 板卡，如图 6-3 所示。

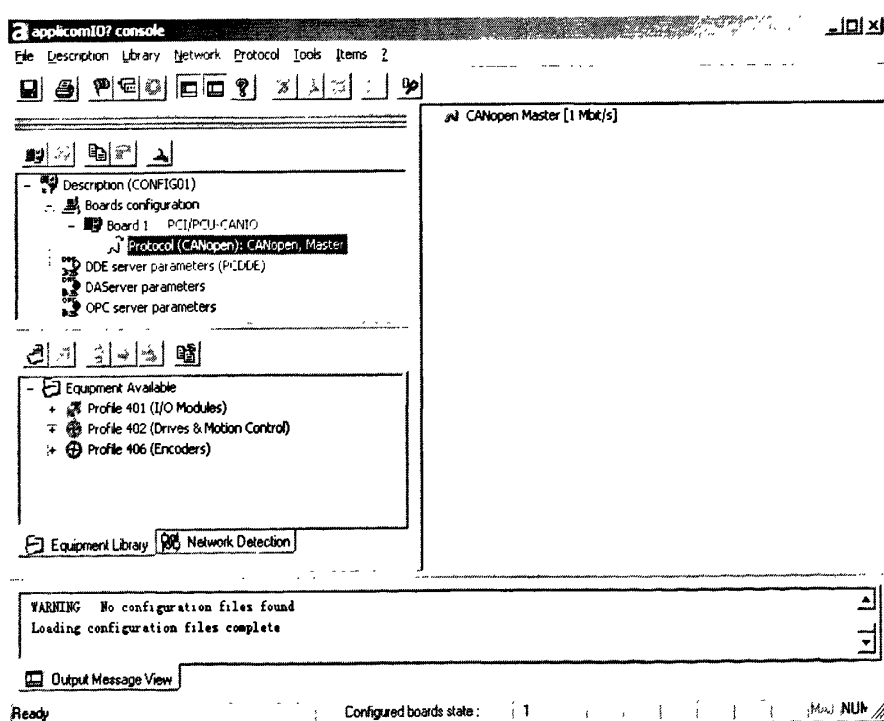


图 6-3 添加 PCU-CANIO 板卡

将板卡和通信接口都连接到由 CANopen 网络中。需要注意的是，PCU-CANIO 板卡是通过一个 9 针接口连接到 CANopen 网络上的，该 9 针接口的引脚定义如表 6-1 所示。

表 6-1 板卡接口引脚定义

引脚号	定义
1	未使用
2	CAN_L
3	0V 接地（可选）
4	未使用
5	未使用
6	0V 接地（可选）
7	CAN_H
8	未使用
9	24V 电源（可选）

在窗口的 Network Detection 选项卡中，点击 Read NetworkConfiguration 按钮，就可以在弹出的 General Configuration 页面中进行波特率、COB-ID、同步周期等参数的设置。本实验采用的通信波特率为 250 Kbit/s，其他参数的设置如图 6-4 所示。

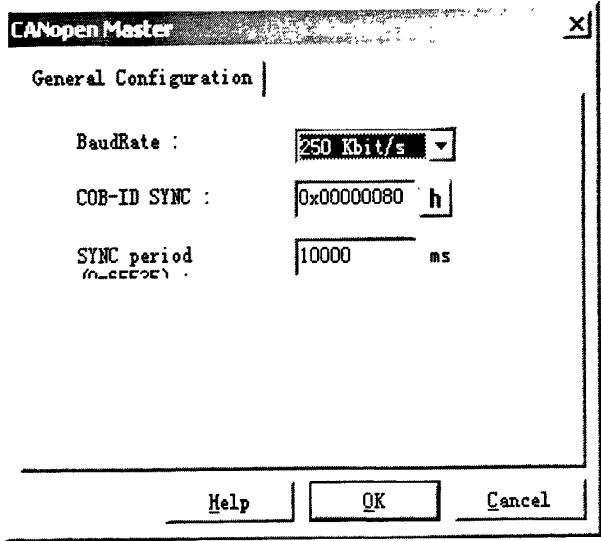


图 6-4 通信波特率设置

设置完成波特率后，点击 Network Detection 选项卡中的 Online Action 按钮，板卡便会自动搜索网络中匹配该波特率的设备，如图 6-5。



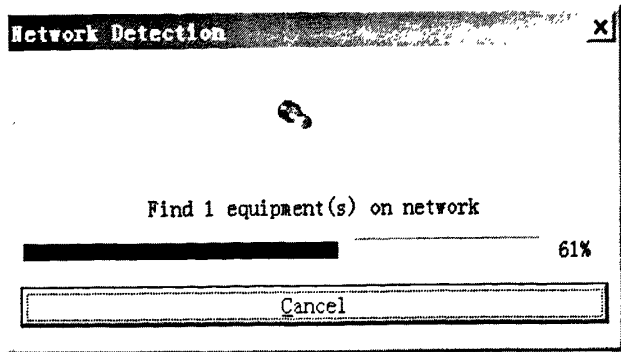


图 6-5 板卡搜索节点

搜索完成后，被搜索到的节点就会显示在 applicomIO 的 Network Detection 选项卡中，并在 OutputMessage View 窗口中显示当前搜索信息，如图 6-6。

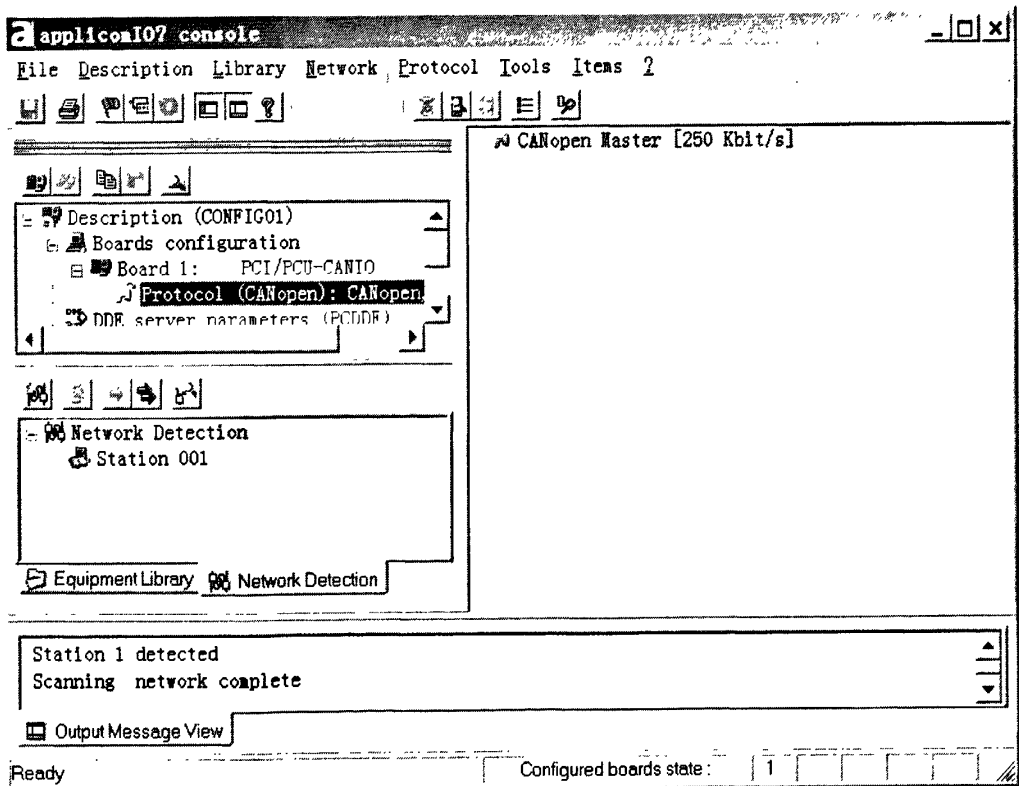


图 6-6 网络中搜索到的节点

点击 Network Detection 选项卡中的 Online Action 按钮，就可以在该窗口中对 CANopen 总线进行监测，并对网络中的从节点进行管理，如图 6-7 所示。在 Online Action 窗口的 Network 选项卡内，通过对窗口右下部的按钮进行操作，就可以实现 NMT 通信，以便于切换网络内所有节点的状态，如 Pre-Operational、Operational、Stopped 等，即可进行系统状态机与通信功能的有效性实验。同时，窗口的右上部分显示了 CANopen 网络中报文的即时收发情况与网络收发的数据总量；窗口左部显示了板卡运行情况与网络负载情况，从而反映出 CANopen 网络中的网络状态。

通过网络管理实验表示，本文所开发的网络接口与主节点通信正常，从而证明其

系统状态与网络通信功能均正常运行。

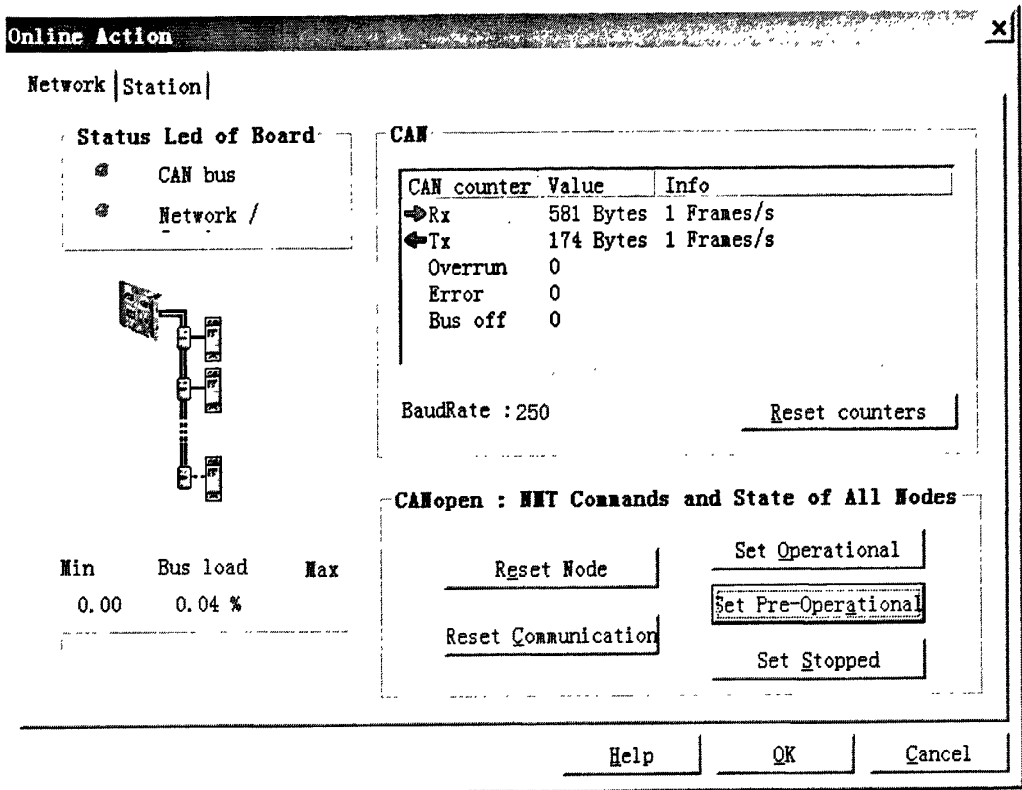


图 6-7 网络管理窗口

此外，在 Online Action 窗口的 Station 选项卡中，可通过 SDO 对通信接口对象字典中的条目进行存取操作实验，以证明通信接口中对象字典的有效性。以索引 0x1005，子索引 0x00 中的数据为例，该段数据的值为 0x80，其功能为定义同步通信所需的 COB-ID。在对象字典源文件中，该段数据定义如下：

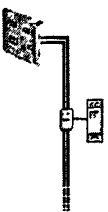
```
UNS32 ObjDict_obj1005 = 0x80;
ODCallback_t ObjDict_Index1005_callbacks[] =
{
    NULL,
};
subindex ObjDict_Index1005[] =
{
    { RW, uint32, sizeof(UNS32), (void*)&ObjDict_obj1005 }
};
```

在 Station 选项卡中，填写 Index 为 0x1005，Sub-index 为 0x00，点击 Read 按钮，就可在该按钮旁的数据窗口中成功得到该段数据的值 80 00 00 00，如图 6-8 所示。

**Online Action**

Network Station

Station Number (0-1)




**Object dictionary access via SDO**

Index (0x0-0xFFFF)   Sub-index (0x0-0xF)

**Read**

**Write**

 ACCESS OK TO THE OBJECT DICTIONARY

**MMT Commands and Node State**

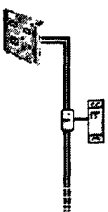
图 6-8 对象字典的读取

在 Write 按钮右侧的窗口中写入数据，如 66 66 66 66，并点击 Write 按钮，就可以通过 SDO 将该数据写入对象字典中的相应位置中，如图 6-9。

**Online Action**

Network Station

Station Number (0-1)




**Object dictionary access via SDO**

Index (0x0-0xFFFF)   Sub-index (0x0-0xF)

**Read**

**Write**

 ACCESS OK TO THE OBJECT DICTIONARY

**MMT Commands and Node State**

图 6-9 对象字典的写入

为了验证数据是否已被正确写入，在确认索引和子索引无误后，点击 Read 按钮可再次读出该位置的数据，可见写入的数据完全正确，如图 6-10。由此可见，本文所开发的网络通信接口可以进行对象字典的读取和写入，从而证明通信接口的对象字典与服务通信对象等功能均运行正常。

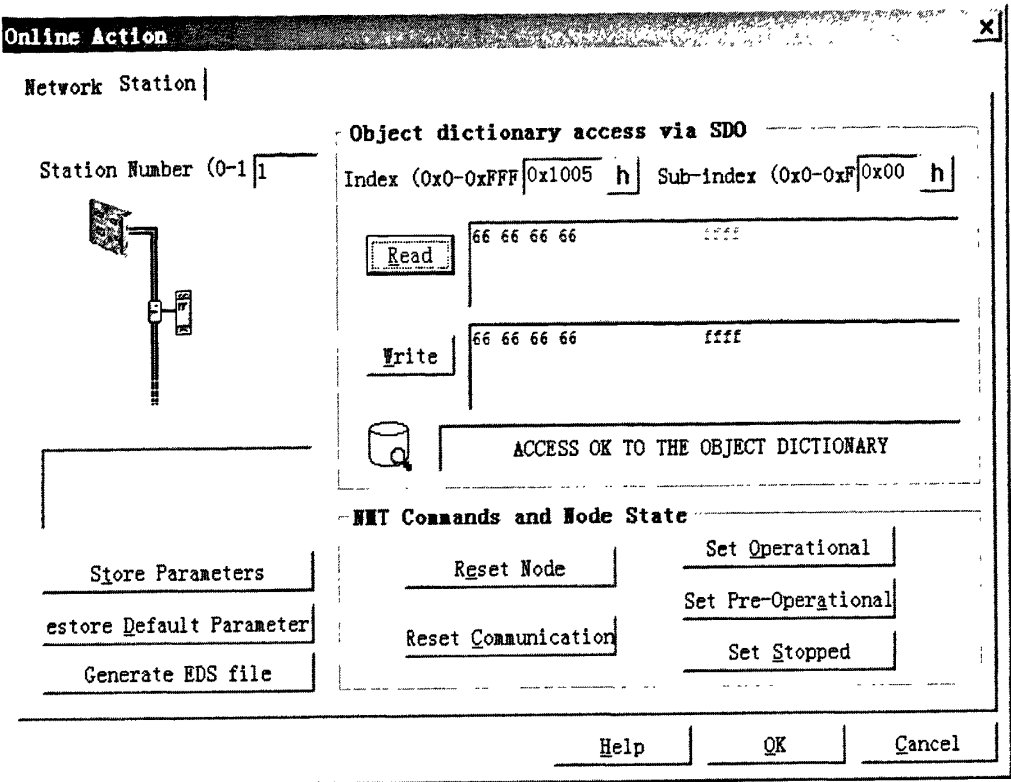


图 6-10 对象字典的验证

此外，本文所开发的通信接口还通过了超过 48 小时的连续运行实验。实验证明，该通信接口具有良好的稳定性。

### 6.3 本章小结

本章搭建了一个简单有效的系统实验平台，并利用该平台进行了 CANopen 协议通信接口的功能实验。实验证明，本文所开发的通信接口软硬件功能完全正常，达到了预期的设计要求。

## 第七章 总结与展望

本文首先分析了 CANopen 总线在国内外的研究现状,其次对 CANopen 的协议规范进行了阐述,简介了 CANopen 的协议结构、对象字典和通信对象,并详细说明了本文使用的开源协议栈 CanFestival。再次,本文对嵌入式实时操作系统  $\mu\text{C}/\text{OS-II}$  的原理和特点进行了阐述。之后,本文搭建了基于 AT90CAN128 的协议研究目标平台,开展了在该平台基础上对 CANopen 协议规范的研究,通过将开源协议栈 CanFestival 移植并封装到嵌入式系统  $\mu\text{C}/\text{OS-II}$  中,从而实现了基于嵌入式系统的 CANopen 网络通信接口。在此基础上,论文提出了一个有效的系统实验方案,并依据此方案对本文开发的 CANopen 通信接口进行了实验,以证明研究的可行性。实验证明,本文所开发的通信接口软硬件功能完全正常,达到了预期的设计要求,从而为今后 CANopen 协议从节点的开发研究奠定了坚实的基础。

CANopen 作为一种非常有竞争力的总线标准,在国内外的应用已经深入到了各个领域,并且还在进一步发展。但是,基于 CANopen 协议规范的通信接口关键技术仍然被国外若干公司所垄断,国内目前有关 CANopen 从节点的研究与开发仍然处于起步阶段。本文通过移植开源协议栈,并将其封装为嵌入式实时操作系统  $\mu\text{C}/\text{OS-II}$  中的任务运行,对于缩短开发周期,降低开发难度,节约开发成本,加速 CANopen 在国内的应用与推广具有重要的现实意义,并为课题的发展留出了较大的空间。

但是,由于本人的水平和时间有限,本文中还有一些不足之处,许多工作还有待于进一步的完善和研究。例如, $\mu\text{C}/\text{OS-II}$  系统中只有一个 CanFestival 协议栈任务,并未真正发挥嵌入式实时系统的全部优点。这也为课题今后的发展留下了接口,诸如:

- 1) 开发更加完善的,可应用于工业控制的 CANopen 主从节点;
- 2) 在主从节点的基础上,添加其他协议任务,从而实现多协议的共存;
- 3) 在多协议共存的基础上,添加协议转换任务,从而实现协议之间信息的转换。

CANopen 是一种非常有价值的总线标准。虽然目前我国的 CANopen 产品仍然比较落后,但相信通过国内各个单位不断的研发与改进,我国一定会逐步缩小与世界先进水平的差距,达到甚至超过世界先进水平。

## 参考文献

- [1] 阳宪惠.现场总线技术及其应用[M].北京:清华大学出版社,1999.
- [2] Wang Junbo,Xu Bugong.Analysis and implementation of the CANopen protocol[J].Control & Automation,2006,6[6]:104~158.
- [3] 郇极,杨斌,魏继光.一种开放式的现场总线协议 CANopen[J].制造业自动化,2002,24(10):33~38.
- [4] 孙健,陶维青.CAN 应用层协议 CANopen 浅析[J].仪器仪表标准化与计量,2006,2:22~24.
- [5] 宋晓梅,贾佳.CANopen 协议在伺服电机控制系统中的实现[J].单片机与嵌入式系统应用,2006,6:5~20.
- [6] 宋晓梅,陈锦妮.基于 ARM 的 CAN 总线与以太网通信网关设计[J].西安工程大学学报,2008,22(2):201~213.
- [7] 张晶,汪滨琦,崔大海.仿真转台控制系统的 CAN 总线通信设计.应用科技[J],2007,34(11):4~24.
- [8] 蒋智康,宋春宁,宋绍剑.PIC18 单片机的 CANopen 通信协议[J].单片机与嵌入式系统应用,2008,9:24~27.
- [9] 邓遵义,宁祎.CANopen 协议剖析及其在伺服电机控制中的实现[J].机电工程,2007,24(8):39~41.
- [10] 徐喆,张卓,闫士珍.基于 uC/OS-II 的 CANopen 从节点的实现[J].计算机系统应用,2008,7:113~118.
- [11] 龚树强,潘旭东,王广林.CAN 总线在电液伺服阀性能测试系统中的应用[J].宇航计测技术,2008,28(4):39~43.
- [12] 孙汉旭,曹志杰,贾庆轩.基于 CANopen 协议的双轮移动倒立摆控制系统.机电工程[J],2008,25(3):4~42.
- [13] 马艳歌,贾凯,徐方.基于 DSP 的 CANopen 通讯协议的实现[J].微计算机信息,2006,22(2-2):146~237.
- [14] 涂智杰,何永义,李一青,等.基于 DSP 的 CANopen 转 Modbus 网关的设计.工业仪表与自动化装置[J],2008,3:55~56.
- [15] 何睿,江庭弘.DeviceNet 标准与 CANopen 标准的比较[J].中国标准化,2008(5):59~60.
- [16] 吴爱国,刘莉.CAN 总线控制系统的应用层协议 CANopen 剖析[J].微计算机信息(嵌入式与 SOC),2003,19(3):27~75.
- [17] Farsi M.,Ratcliff K.,Barbosa,M..An introduction to CANopen[J].Computing & Control Engineering Journal.1999,10[4]:161~168.
- [18] BOSCH Corporation.CAN Specification Version 2.0[S].1991.
- [19] 北京博控自动化技术有限公司.CANopen 协议介绍[Z].2005.
- [20] Menon C.J..A single network solution for safety-related applications using CANopen[J].Industrial Electronics Society,2005,11:6~10.

- [21] 王俊波,胥布工.CANopen 协议分析与实现[J].微计算机信息(嵌入式与 SOC),2006,22(6-2): 104~158.
- [22] CAN in Automation. Application Layer and Communication Profile, CiA Draft Standard 301 Version 4.02[S].2002.
- [23] 任玮蒙,陶维青.基于 CAN 总线的高层协议 CANopen[J].自动化技术与应用,2007,26(4): 127~130.
- [24] Minkoo Kang,Kiejn Park,Bongjun Kim. PDO packing mechanism for minimizing CANopen network utilization[J]. IECON 2008 - 34th Annual Conference of IEEE Industrial Electronics Society, 2008:1516~1519.
- [25] 宋晓强.CANbus 高层协议 CANopen 的研究以及在模块化 CAN 控制器上的实现[D].天津:天津大学,2004.
- [26] 何光宇,胡正,秦霆镐,王健.针对工业控制的 Canopen 系统[J].微计算机信息(测控自动化),2003,19(12):5~6.
- [27] 孟昭.基于 CANopen 协议的 CAN 总线控制系统研究[D].北京:北京工业大学,2008.
- [28] Zhao Jianguang,Yang Jianwu,Sun Shuwen. The development and application of CANopen compliant I/O slave[J]. Microcomputer Information,2007.23(23):9-11.
- [29] 甘露.嵌入式 CAN 应用层协议控制器研究与实现[D].成都:西南交通大学,2007.
- [30] H.Boterenbrood. CANopen: high-level protocol for CAN-bus[Z].2000.
- [31] Janez Paternoster. CANopenNode Manual[Z].2006.
- [32] Embedded Systems Academy.MicroCANopen v3.30 Manual[Z].2007.
- [33] Lolitech.CanFestival v3.0 Manual[Z].2007.
- [34] 任哲,潘树林,房红征.嵌入式操作系统基础  $\mu$ C/OS-II 和 Linux[M].北京:北京航空航天大学出版社,2006.
- [35] 江英.基于网络监控的嵌入式总线控制器的设计[D].哈尔滨:哈尔滨理工大学,2005.
- [36] 杨学民,徐雅斌. $\mu$ Clinux 内核研究及实时性能的实现[J].辽宁工业大学学报(自然科学版),2009,29(2):83~86.
- [37] 刘滨,王琦,刘丽丽.嵌入式操作系统 FreeRTOS 的原理与实现[J].单片机与嵌入式系统应用,2005(7):8~11.
- [38] 卫进.AVR 单片机的 RTOS—AVRX 应用[J].单片机与嵌入式系统应用,2006(1):18~20.
- [39] Jean J Labrosse.  $\mu$ C/OS-II, The Real-Time Kernel [M]. Lawrence,Kansas: R&D Publications,1998.
- [40] 何永泰,肖丽仙.AVR 单片机中移植  $\mu$ C/OS-II 的研究[J].微计算机应用, 2007,28(1):96~100.
- [41] Atmel Corporation.AT90CAN32/64/128 User Manual Rev. 7679H-CAN-08/08[Z].2008.
- [42] 广州致远电子有限公司.CTM1050 高速 CAN 隔离收发器产品数据手册 Rev 1.20[Z]. 2006.
- [43] 王宇波.嵌入式网络控制器的设计与实现[D].天津:天津大学,2005.
- [44] 王贵建.基于 CAN 总线的分布式发动机控制系统设计[D]. 哈尔滨:哈尔滨工程大学,2006.

- [45] 刘涛.CAN 总线接口电路设计中的关键问题[J]. 工矿自动化,2007,2(1):2~3.
- [46] 宋娟,于智宏.CAN 总线接口电路设计[J].河南机电高等专科学校学报,2007,15(3):17~19.
- [47] 李积英.基于 AT90CAN128 单片机 CAN 总线实现方案的研究[J].兰州交通大学学报(自然科学版),2007,26(1):32~34.
- [48] Zhao Jianguang,Yang Jianwu,Sun Shuwen. The development and application of CANopen compliant I/O slave[J]. Microcomputer Information,2007.23(23):9-11.
- [49] 金蕾. $\mu$ C/OS-II 在 AVR 单片机中移植的关键技术[J]. 电脑知识与技术,2006,32:96~97.
- [50] 姚旭影. $\mu$ C/OS-II 操作系统在 AVR 单片机上的移植[J]. 现代电子技术,2006(12):53~55.
- [51] 徐海兵.GNU Make 中文手册 ver-3.8[Z].2004.
- [52] 陈皓,李大锋.跟我一起写 Makefile[Z].2005.
- [53] Woodhead.Direct-Link CANopen User Manual[Z].2007.



## 发表论文和科研情况说明

发表的论文：

- [1] 陈在平，王峰，“基于 CANopen 协议从节点研究”，《制造业自动化》，2009 年 11 月

## 致 谢

本论文的工作是在导师陈在平教授的悉心指导下完成的，陈在平教授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。在此衷心感谢研究生学习期间陈教授对我的关心和指导。

罗克韦尔实验室的贾超老师悉心指导我完成了本论文中的实验工作，在此向贾老师表示衷心的感谢。

在研究生就读期间，杨慧、李志国、李旭、尤丽静、吴凡、谷秀平、李沛然、冯亮、冯金钊等同学对我的学习和生活等各方面都给予了热情的帮助，在此向他们表达我的感激之情。与优秀人才共事，使我在竞争中获益，对于能和你们一起度过研究生的两年半时光，我深感荣幸。

最重要的是感谢我的家人。你们的包容和付出，使我能够在学校专心完成学业。我以你们为荣，谨将此文献给你们。

# 基于嵌入式系统的CANopen协议分析研究

作者：[王峰](#)  
学位授予单位：[天津理工大学](#)

本文链接：[http://d.g.wanfangdata.com.cn/Thesis\\_Y1750635.aspx](http://d.g.wanfangdata.com.cn/Thesis_Y1750635.aspx)