分类号 密级

U D C 编号

中 南 大 学

CENTRAL SOUTH UNIVERSITY

硕士学位论文

论 文 题 目 基于 AT91FR40162 的嵌入式通信接口系统的设计与实现

学 科、专 业 控制理论与控制工程

研 究 生 姓 名 周 立 平

导 师 姓 名 及

专业技术职称 李 劼 教授

摘 要

本文在综合评述嵌入式系统技术及其在工业控制领域中的应用现状的基础上，设计了一个适合于工业分布式测控网络的嵌入式通信接口系统。主要内容和研究结果如下：

1. 对系统进行了总体设计、规划出了系统的主要功能子模块，确定了设计方法、层次结构以及开发工具；根据应用需要，对系统的嵌入式操作系统、嵌入式处理器进行了精心选型，并分析了选型过程中的主要考虑因素。

2. 从各功能单元模块的具体电路设计入手，着重分析了 CAN、Ethernet 等关键通信电路的实现过程，同时也研究了本系统中串行通信口、电源电路、复位电路、JTAG 电路、USB 电路等子模块电路的设计和系统存储器的扩展方法以及硬件平台 PCB 板的制作和硬件调试过程。

3. 围绕系统的软件开发，设计了系统硬件配置的启动代码，并分别在51单片机以及AT91FR40162处理器上移植了实时内核μC/OS-II，详细讨论了移植的实现过程和主要难点，给出了移植效果，在系统上建立了能实现实时多任务调度的RTOS。

4. 提出了 RTOS 的完善工作，并针对硬件驱动部分的完善工作，进行了系统的串行通信接口、CAN 接口控制器 SJA1000、以太网接口 RTL8019AS 等关键驱动程序设计。最后，分析了应用程序的设计方法，并结合实际的任务分析了本系统的启动和运行过程。

本文设计的嵌入式通信接口系统对提高分布式控制系统的组网方便性、实现控制系统的通信冗余结构和过程实时处理等都有重要的意义。此外，本系统结构较简单、小巧，且功耗低，很好的体现了嵌入式系统技术的优点。

关键词 ARM 处理器，现场总线控制系统，μC/OS-II 内核，CAN 总线，嵌入式实时操作系统

ABSTRACT

Based on a comprehensive review of the technology of embedded systems and its application in industrial control areas, a communication interface system fitting for the use in industrial distributed measurement & control networks was designed in this paper. Main content and results are as follows:

1. An overall design on the system was carried out, main functional modules of the system being laid out and design methods, hierarchical structure and developing tools being determined; according to the need of application, the types of embedded processor and embedded operation system were chosen carefully and main considerations were analyzed in detail.

2. Starting from the design of embodied circuits of the modules of the system, the realization process of key communications circuits such as CAN and Ethernet interfaces were analyzed; the designs of the serial communication interface, power circuit, reset circuit, JTAG circuit and USB circuit were studied and the methods for memory expansion as well as the methods and steps for the manufacture and debug of the hardware platform (PCB blocks) were discussed.

3. Bootloader codes for the hardware configuration and initialization of the system were designed and the transplants of the real-time core, μC/OS-II, were realized respectively in 51 single-chip computer and AT91FR40162 processor; the realization process and main difficulties in the transplants were expounded; the effect of the transplants was given. A RTOS being able to realize real-time multi-task disposal was established in this system.

4. Further improvement job for the RTOS was pointed out, aiming at the improvement of the hardware drivers of the system, the design of key driving procedures for serial communication, CAN interface(SJA1000) and Ethernet interface (RTL8019AS) was carried on. Finally, design methods of application programs were analyzed, combined with actual application tasks, the starting and running process of the system was analyzed too.

This communication interface system possesses significance in the

improvement of the convenience to build networks of distributed control systems, the realization of communication redundancy structure and real-time disposal. Moreover, the system has a relatively simple and cabinet structure and low power consumption, exhibiting the advantages of embedded systems.

KEY WORDS ARM，FCS，μC/OS-II，CAN bus，Embedded **RTOS**

目 录

第一章 绪论

1.1 研究背景和现状

过去五十年，随着微电子技术、计算机通讯与网络技术、传感技术、控制理论和方法等相关学科的发展，以及模拟和数字仪表的推广，使得工业自动化技术取得了飞速发展。作为工业自动化技术的集中体现，控制系统无论内容还是形式上都有了很大的改变[1]。

同样在社会信息化进程中，此次目前嵌入式系统设备在应用数量上已经远远超过了通用计算机。在工业领域，随着嵌入式系统技术的应用，它也正逐渐改变着传统的工业生产方式。

目前，怎样将嵌入式系统技术应用到工业过程控制领域是一个值得控制系统设计者思考的课题。本文的主要工作就是针对上述情况，设计了一个适合于分布式工业测控网络通信的嵌入式通信接口系统，以期能为嵌入式系统技术在工业领域的推广应用起到抛砖引玉的作用。

1.1.1 工业控制系统研究现状

随着通信模式、电子技术和仪表等的发展，工业控制系统历经了从简单到复杂、从局部自动化到全局自动化、从低级智能到高级智能的过程[2]。当前工业过程的物理设备数量急剧增加，同时系统要求的功能不断增加，传统控制系统中点对点的通信模式，在控制系统中已达到了它的应用极限，不能满足控制系统模块化设计、实现先进控制、集成诊断、方便维护和低成本等需求[1]。

因此，以现场总线和工业以太网技术为基础的全分布控制系统是现在过程控制系统的主要形式。

现在，全分布控制系统的主要构建方式是以 FCS 或 DCS 为基础，再通过转换网关或网桥将 FCS 和 DCS 集成起来。一般形式是以 FCS 做一些小型的控制系统或是把该 FCS 作为 DCS 的子系统完成一些特定的控制功能；较典型的应用形式是将 DCS 和 FCS 结合在一起，首先利用现场总线组成现场级的设备网络，作为系统的底层，即控制系统的现场设备层，完成较低级的任务控制功能，再接入以太网(管理层)，这样可以很方便地构建复杂的系统和完成复杂的功能，也能充分结合现场总线和工业以太网的优点，体现出当前分布式控制系统的管控一体化的发展趋势。

目前，对于现场总线主要问题是现场级的智能仪表、现场总线仪表产品质量还没有过关、特别是多种现场总线标准并存，给应用者带来极大不便[3~5]；另外 FCS(fieldbus control system)的软件功能不够强大，功能模块少，在复杂控制方面

还无法完成 DCS 已有的先进控制功能。利用现场仪表只能完成一些单回路、串级控制等。同样工业以太网由于 TCP/IP 协议本身的缺陷，给控制系统的实时处理带来很大影响。

当前对于全分布控制系统的迫切需要解决的问题是：发展适合自控应用需求的工业数据通信与网络技术，即统一总线标准；加强针对具体工业过程的现场级智能设备研制；同时针对 FCS 要研究解决分布式计算、通讯延迟、异步、网络环境下的新控制技术；并迅速推出新的理论和方法来解决控制系统中分布的多个计算单元间通过基于信息包通讯互连而出现的问题，来保证控制系统的稳定性、性能指标和鲁棒性[1]。

1.1.2 嵌入式系统研究现状

当前，嵌入式系统技术推广很快，虽然现在很多大型电子公司大力投入进行基于 ARM 的嵌入式系统技术开发，但是大多数集中在移动终端通信产品、数码相机等个人电子消费品这个领域，追求的主要是高利润和产品的竞争力，在工业应用方面还无暇顾及。更有许多电子商利用市场的需要，推出形形色色的 DEMO 板或是评估板，其中大多数是 ARM 芯片的生产厂家推出芯片时的简单测试板，主要功能是测试处理器芯片的性能，因此局限性强、很难保证其稳定性。

其实嵌入式技术也是和具体行业应用紧密结合的，开发知识密度高了不少。特别是在过程工业中，它要与工业现场直接接触，所以无论工作环境，还是通信方式和信息数据的处理，都与具体对象紧密耦合在一起[6~7]。

作为嵌入式应用，系统的硬件结构平台最好是能根据实际需要[8]，精心搭制，因为嵌入式系统软硬件耦合程度高，后期软件开发工作主要在硬件平台相对固定的情况下进行，一旦出现故障，将带来很大的麻烦、甚至导致整个开发的失败。因此不要抱着省时、省力的思想，去轻信厂商提供的信息而贸然选购一些所谓的平台或是评估板再做二次开发。

1.2 研究目的和意义

综合分析了工业控制系统和嵌入式系统目前面临的情况，本文设计了一个适合分布式工业测控网络通信的嵌入式通信接口系统。

如图 1-1 所示，该系统的作用是将基于 CAN 总线的工业现场总线控制网络接入以太网。系统作为中介实现现场设备层和管理控制层之间的通信，即信息和数据交换，以达到控制的目的。现在大多数工业 CAN 控制网络与以太网的数据信息交换都是通过 PC 机的支持实现 CAN 和以太互换，这样构建的系统灵活性差，而且稳定性难以保证，一旦出现故障将导致系统瘫痪。本系统可以克服上述

弊病，可以方便地构建大型的工业控制系统，节省成本，并能很好地实现系统的通信冗余结构，贴近工业现场的应用需要。

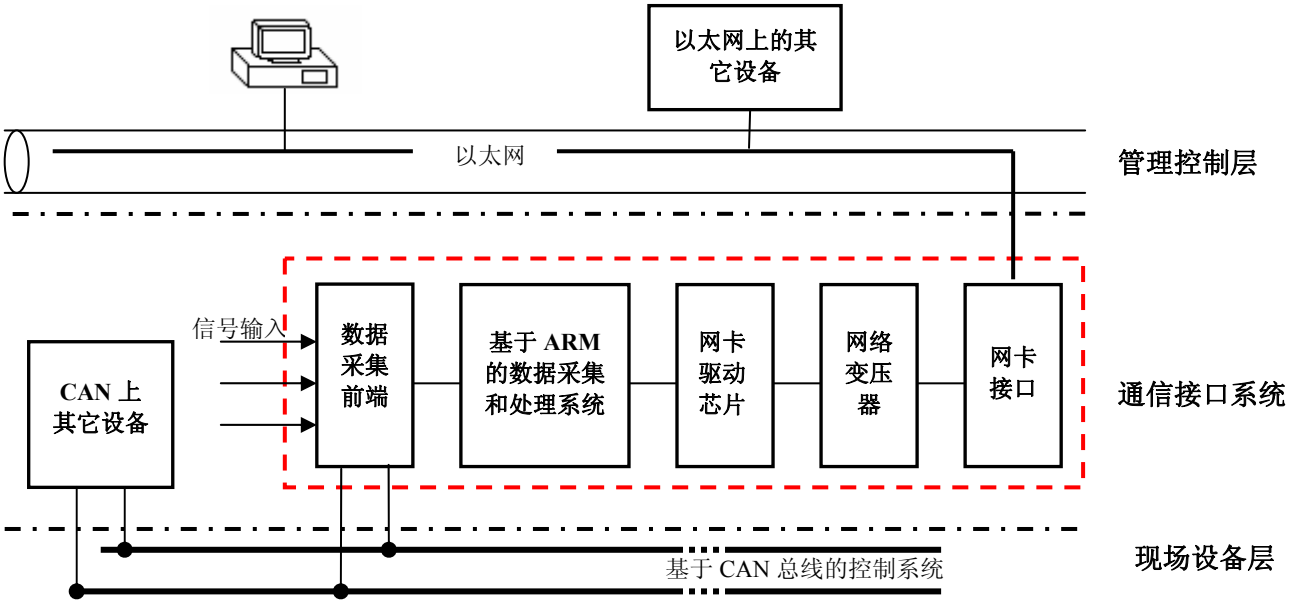

图 1-1 系统的结构原理图

如前面所述，该系统的作用是作为中介连接分布式控制系统中的两层：底层的现场设备层(基于现场总线)和管理控制层(基于以太网)。在图 1-1 中的通信接口系统主要作用是实现 CAN 和以太的数据格式转换和数据的传输。在现场设备层可以挂接多个设备，在这里只定性地提到前端数据采集块，这个数据采集块是 CAN 网络上的一个智能节点，它能采集现场设备的状态数据，也可以针对过程的数据作出一定的判断，最终将有用数据转发给到上层以太网的中心处理系统；以太网是管理控制层，通过扩展网卡把经数据采集与处理子系统的处理后的数据传入以太网，通过上位机进一步处理，对现场过程实现控制和管理。数据采集与处理子系统负责数据采集与处理、同时也是与以太网的接口。在该通信接口系统中用户还可以嵌入对过程的具体控制功能，实现控制系统功能的分散和下放。

显然，这种方法很好地将现场总线控制系统和工业以太网结合成一个系统，既能充分克服传统集散控制系统(DCS)的风险集中的不足，又能有效抑制以太网在工业应用中实时性差的缺点，是分布式控制系统的一个不错的构建方式[2]。同时为了系统的稳定性，结合现场总线的特点，在实际工程中，可以挂接两个这样的接口平台，彼此间互为备份，可以灵活地实现控制系统的通信冗余。

因此，本课题既能从实际工业应用出发，充分结合了嵌入式系统的优势，相信对过程控制能作出一定的贡献。

1.3 本文主要内容和结构

本论文主要围绕基于 μC/OS-II 和 ARM 的嵌入式通信接口系统的设计而展开，论文分成以下几个部分：

第一章 综合分析了工业控制系统和嵌入式系统的现状和面临的问题，提出了课题的研究意义和系统的总体设计思想。

第二章 讨论了系统的开发方法、开发工具，进行了系统的总体功能规划设计，确定了系统的层次结构、设计方法和软硬件的选型。

第三章 完成了系统硬件平台的设计。分别从各单元电路的具体设计入手，分析了整个系统的硬件设计过程，并给出了各单元模块的详细硬件电路图。

第四章 完成了系统的软件实现过程。主要有硬件系统启动代码的设计，并详细讨论了实时内核 μC/OS-II 在 51 单片机及 AT91FR40162 上的移植过程及难点，为系统提供了一个相对简练、实用的 RTOS。

第五章 指出了基于 μC/OS-II 的 RTOS 要完善的主要工作，针对驱动部分，完成串行通信口、CAN 控制器 SJA1000、以太网 RTL8019 等的驱动程序设计，并分析了本嵌入式通信接口系统的启动和运行过程。

最后，总结了本文的主要工作，并指出了下一步所要做的工作。

第二章 系统的总体设计和选型

本嵌入式通信接口系统是基于嵌入式系统技术的应用系统，嵌入式设备是软硬件高度耦合、知识密集度高的产品，根据具体应用需求，设计出良好的嵌入式应用系统，是设计的目标。而要做到这些首先需要对整个嵌入式系统的基本知识包括设计方法、开发工具等有个总体认识，以作出良好的开发方案。因此在这一章中介绍了系统的设计方法、如何选择开发工具，并对整个系统的功能进行了规划，完成了系统的软硬件选型，并介绍了选型过程中要注意的问题。

2.1 系统的设计方法和开发工具

2.1.1 系统的设计方法讨论

嵌入式系统的设计与一般开发方法差别很大。以一个 MP3 播放设备的设计为例，一般方法首先将考虑电路如何设计、软件要分那些模块，然后细化和分解成具体模块，再分别进行设计。嵌入式系统将把 MP3 播放器总体上看作是一个音频系统，包括 MP3 解码系统、数据存储系统、模拟播放系统、用户界面系统等部分，开发时要处理的问题是如何把这些子系统协调起来。可以看出，前者的侧重点是考虑如何实现的技术细节，因此称作产品级的设计，而后者则只需要考虑系统所需各部件的整合关系，也称作系统级的设计。

图 2-1 是嵌入式产品的设计流程[9~11]，从中可以看出：

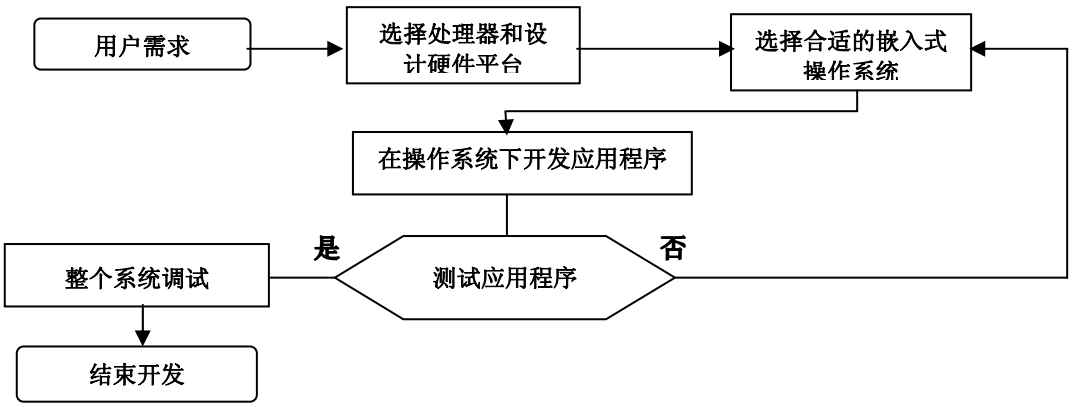

图 2-1 嵌入式系统开发流程图

首先，嵌入式系统的硬件和软件是紧密耦合在一起的，因此在设计时需要考虑具体的应用对象，不但要求有电子方面的信息，还需结合应用的对象的特点，把握其中的关键信息，再做出相应的选择。

其次，与单片机系统设计的区别。单片机系统的开发，有“系统”的思想，但它所指的“系统”更强调产品里面包含很多器件，以及器件间是怎么协同工作的。

在开发时，按照“瀑布”式流程[27]，设计过程就是通过反复硬件调试来去除错误，同时单片机系统的资源有限，不考虑操作系统的引入，因此应用程序的开发时直接对硬件操作多，使得系统缺乏灵活性、可移植性差；嵌入式系统强调的“系统”指设计时通盘考虑整个系统，相对灵活性强，而且一般考虑引入嵌入式操作系统，这样能屏蔽掉底层硬件的很多复杂信息，使得开发者通过操作系统提供的 API 函数就可以完成大部分工作，大大简化了开发过程[11]。

另外，两者之间最大的差别在设计流程上的不同。一般产品开发时它的硬件和软件采用分离开发方式，虽然也进行需求分析，也考虑软硬件接口、软硬件性能等的改进问题，但由于软硬件分开设计，各部分的修改和改进都是独自进行的，这样很难使系统综合性能达到最佳。出现的缺陷很容易导致系统集成失败，更严重的是对产品的修改可能会涉及到对整个软件结构或是硬件配置的改动。嵌入式系统设计采用软/硬件协同设计的方法，在系统功能规划后，软/硬件进行统一划分、协同设计和协同测试验证。因此能充分考虑软硬件的关系，并在设计过程的每个层次上给以测试验证，尽早发现和解决问题，避免灾难性错误的出现；同时在软件开发时系统硬件平台都是相对通用的、固定的、成熟的，减少了硬件系统引入错误的机会。

2.1.2 系统的开发和调试工具

嵌入式应用系统的开发工具一般有集成开发环境、硬件平台或评估板、JTAG 仿真调试器等。由于嵌入式实时操作系统一般作为应用系统的“系统软件”，因此一般也可将它归为开发工具，对于嵌入式系统的开发，开发工具的合理选择直接决定着下一步的开发工作的成效，因此必须慎重选择。

1．集成开发环境

目前，ARM 公司推出的专用集成开发环境(IDE)有 SDT251、ADS1.2、RealView 三种，RealView 是 ARM 公司最新推出的集成开发环境，也有第三方推出的 IDE。集成开发环境一般包括编辑软件、编译软件、链接软件、调试软件、项目管理及函数库等六部分。

用户可以根据需要选择相应的开发环境，一般各种 IDE 的编译效率不同，并且需要一定的购买费用，支持的 ARM 处理器类别也不同，考虑上述因素，本系统开发选用 ADS1.2。

ADS[13~16]的英文全称为 ARM Developer Suite，是 ARM 公司用来取代以前推出的开发工具 SDT，主要由两个部分组成：一个是工程管理(编辑及设置)界面(CodeWarrior)，一个是调试界面 AXD。ADS 起源于 SDT，对一些 SDT 的模块进

行了增强并替换了一些 SDT 的组成部分，底层的汇编器/编译器是 ARM 公司开发的产品，编译效率比别的编译器高很多，比如 Gcc；另外 ADS 支持所有 ARM 系列处理器包括最新的 ARM9E 和 ARM10，除了 SDT 支持的运行操作系统外还可以在 Windows 以及 RedHat Linux 上运行。

2. 仿真和调试工具

现在调试和仿真工具比较多，有ARM自己推出的，也有第三方推出的支持工具，主要分为指令集模拟器、驻留监控软件、JTAG仿真器、在线仿真器等几种。本系统选用ARM公司的JTAG仿真器，详细设计可参考第三章3.1.6节中的相关介绍，相对而言，JTAG仿真器价格低廉，使用方便，能满足开发需要。

它的主要特点有：能支持ARM实时在线仿真和调试，因此可以对具有JTAG接口的硬件芯片内部进行边界扫描和故障检测，这种基于JTAG协议的仿真器比较便宜，连接也很方便如图2-2；可以通过现有的JTAG边界扫描与CPU核进行通信，属于完全非插入式(不占用片上资源)调试，无需目标存储器，不占用目标系统的任何端口，而这些是普通的驻留监控软件所必需的；另外，JTAG调试的目标程序是目标板上执行，仿真更接近于目标硬件，因此，仿真结果与真实的运行环境更为接近，所以逐渐成为目前最流行的调试仿真工具。

3. 系统的开发工具

通过上两小节的介绍，本系统的开发工具选用 ADS 集成开发环境，JTAG 仿真器。整个系统的设计和调试工作都通过 ADS 配合 JTAG 仿真器进行，集成开发环境在宿主机(PC 机)上运行，这样可以充分利用宿主机上丰富的资源及良好的开发环境，生成的目标代码可以通过 JTAG 仿真器传输并装载到目标系统。

图2-2是目标系统的调试连接方法。

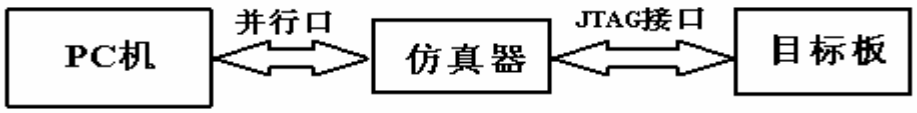

图 2-2 目标系统调试连接图

2.2 系统的总体设计

1. 系统的层次结构

本应用系统是基于嵌入式系统技术的通信接口系统，作为嵌入式产品，它的功能层次结构也遵从嵌入式产品的基本结构，可以分成硬件层、中间层、软件层和功能层[11]。参照图 2-3，各层的主要结构和功能如下：

硬件层：包括嵌入式处理器，存储器系统，通用设备接口和根据需要扩展的I/O 接口。

中间层：中间层介于硬件层与软件层之间，也称作硬件抽象层(Hardware Abstract Layer)或板级支持包(Board Support Package,BSP)，它把系统软件和低层硬件部分隔离，使得系统的底层设备驱动程序与硬件无关。

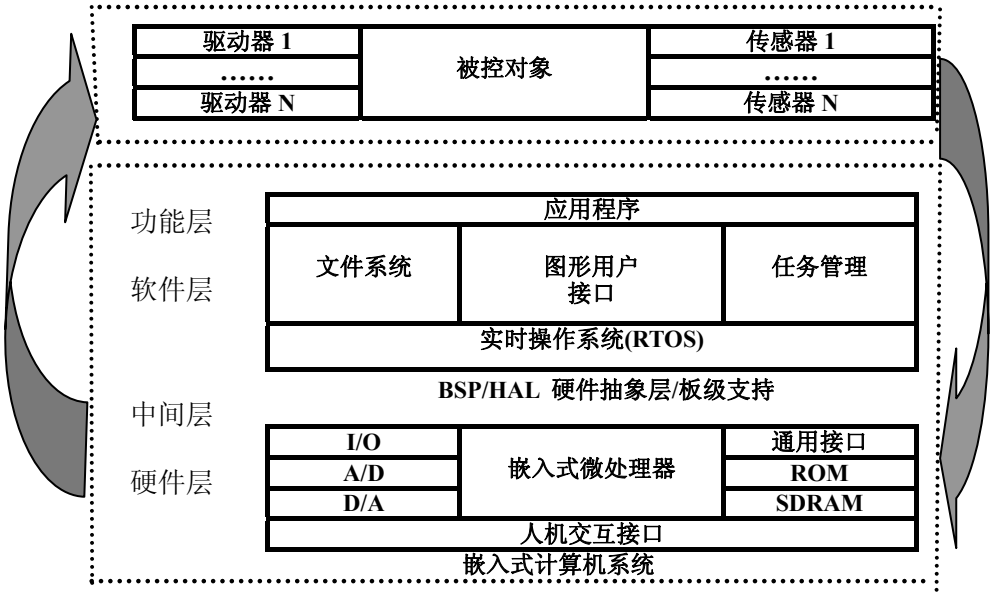

图 2-3 嵌入式系统产品的层次结构

软件层：由 RTOS，文件系统，图形用户接口以及通用组件模块等组成。RTOS 实际上是一个嵌入式目标代码中的程序，系统复位后首先执行，相当于用户的主程序，用户的其他应用程序都建立在 RTOS 之上。RTOS 是一个标准的内核，它将 CPU 时钟，中断，I/O，定时器等资源都封装起来，留给用户的是一个标准的 API 函数接口。

功能层：由基于 RTOS 开发的应用程序组成，用来完成对被控对象的控制功能。是面向对象和应用的，为方便用户操作，一般需要一个友好的人机界面。

2．系统的功能规划

本系统主要功能是作为分布式控制系统中的不同层间的通信接口平台。因此，在系统规划时首要考虑的问题是怎样解决将控制系统中的层间的通信，涉及层间的数据格式转换、信号流量大小、通信实时性和可靠性等问题，需要嵌入实时操作系统来进行多任务调度处理。同时如果工业过程比较复杂，有了操作系统后可以在通信接口系统中适量考虑部分对过程的控制功能，比如对采集的数据能进行一定的预处理。因此在系统硬件设计时，除了基本的 CAN 接口和以太网接口，要在处理器能承受范围内尽量扩充出多的通用外设和通信交互工具，这样，系统的实用性强，也可以方便后面的调试工作。

整个系统的功能结构模块图如图 2-4。

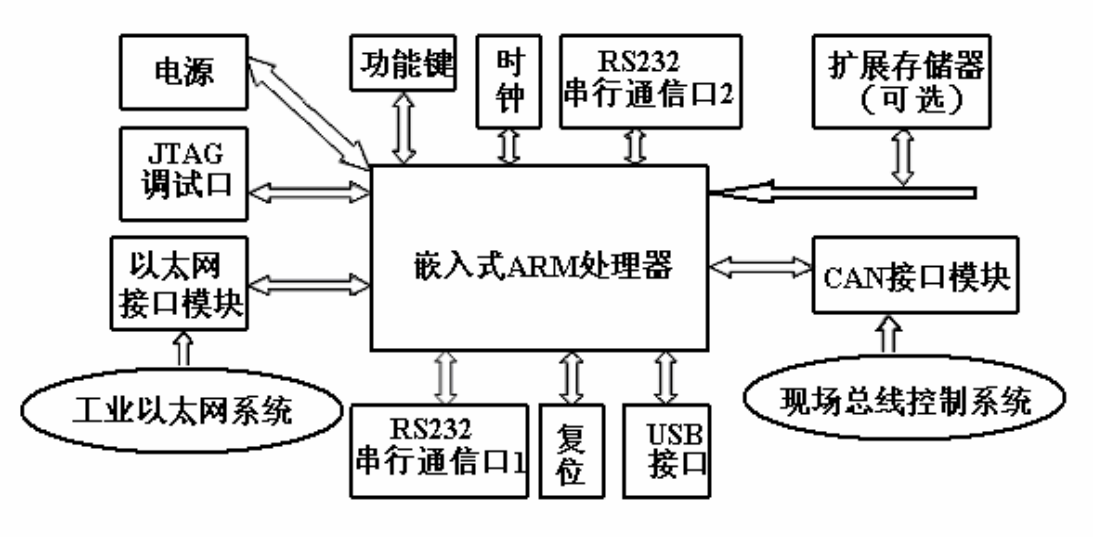

图 2-4 系统功能模块图

嵌入式 ARM 处理器内部存储资源一般较丰富，大多数都有嵌入的 Flash、SRAM。在系统工作时，Flash 作为程序存储区，SRAM 作为程序运行区，它们的大小决定于所选的具体的 ARM 处理器，在这基础上用户也可根据需要选择扩展存储器块。根据第一章的研究内容，本系统作为分布式控制系统接口系统，必须扩展基本的 CAN 接口模块，用来接受底层控制现场的数据信号；扩展以太网接口来与控制系统上层的以太网连接；同时，根据系统功能扩展和升级需要，我们考虑扩展 USB 接口，来与 USB 外设进行通信，此系统中我们暂时通过它连接一个 U 盘(USB Device)，将来可以考虑通过 USB 口设计一个数据采集系统。扩展两个串行 RS232 接口，一个用来实现下载和装载程序，这样可以通过传统的 Flash 工具进行编程；另一个串行口用来在开发过程中完成系统的调试工作，并能与 PC 机通信来监视系统的运行情况，也可以通过它发出部分 PC 机的控制数据；扩展功能键用于现场操作或是进行硬件配置。基本的模块还有系统的时钟模块、复位电路、标准 JTAG 模块。考虑到工业具体应用还可选扩展 RS-485，LCD 等，在本文中没选。

2.3 系统的选型

本系统作为分布式控制系统的通信接口系统，而且要应用于工业现场，因此必须考虑系统的可靠性、稳定性、实时性、低功耗和抗干扰性。为了满足工业环境的要求，对系统的软硬件的选型和设计要求都比较高[17~18]。

2.3.1 嵌入式操作系统选型

从系统的功能规划模块图，我们知道，系统的外设较多，在应用程序开发时，

操作的外围比较多，而且接口的 CPU 的负荷较大，另外作为工业现场控制，要求过程数据处理的实时性。所以必须考虑嵌入式操作系统的选型，这又得综合考虑多方面的因素。对于 ARM 处理器选择嵌入式操作系统主要考虑以下几个因素：

第一是具体应用对象。如果开发的嵌入式设备是和网络应用密切相关，那么应该选择用嵌入式 Linux 或者 μCLinux，而不是 μC/OS-II[19~20]。

第二是实时性要求。没有一个绝对的数字可以说明什么是硬实时，什么是软实时，实时性与选择什么样的 CPU 以及 CPU 的主频、内存以及操作系统等的参数有密切关系。一般使用加入实时补丁等技术的嵌入式 Linux，如 MontaVista Linux(2.4.17 版本)[22]，最坏的情况只有 436 微秒，而 99.9%的情况是 195 微秒。当然，如果希望更快的实时响应，如高速的 A/D 转换需要几个微秒以内的中断延时，采用 μC/OS-II 是合适的[21]；采用 Vxworks 等传统的嵌入式操作系统也可以满足这样的强实时性要求。

第三是开发工具。目前 μC/OS-II、μCLinux 和嵌入式 Linux 的开发工具与商业嵌入式操作系统工具还有一些差距，ARM CPU 上广泛流行和使用的是 ARM 公司 SDT/ADS，产品无论在功能、稳定性和众多的第三方厂商支持等方面都很好，唯一不足的是缺少对嵌入式 Linux 操作系统的支持，SDT/ADS 的升级产品 RealView 计划支持 GCC 编译器和嵌入式 Linux，但目前还没有看到。μC/OS-II 可以使用 SDT/ADS，但没有操作系统调试功能[23]。

第四是所选择的 ARM CPU 和硬件情况，象 ARM7TDMI 和 ARM940T 核是不能使用嵌入式 Linux 的，如果想用 Linux，只能用 μCLinux，如果想用 VxWorks，需要进行硬件板的 BSP(板级支持包)开发，时间长[24]。

最后是价格和技术服务。在考虑购买商业嵌入式操作系统时，会遇到是买还是自己做的问题，这是很正常的，尤其是在采用开放源代码技术时，这个问题就更加突出。有一点需要注意的是，有些产品如 VxWorks 是既按用户数收取开发费，也按用户产品售出的实际数量收取每个 VxWorks 软件运行的版税。μC/OS-II 只是收取每种产品一次性版税(不限数量)。Linux 无论是 μCLinux 还是嵌入式 Linux 是不收版税的。

根据上述原因，本系统的嵌入式操作系统选用 μC/OS-II。

2.3.2 μC/OS-II 的特点

μC/OS-II 是一个简单、高效的嵌入式实时操作系统内核，是由 Labrosse 编写的一个开放式内核，主要的特点就是源码公开，是一个占先式的内核，即已经准备就绪的高优先级任务可以剥夺正在运行的低优先级任务的 CPU 使用权，这个特点使得它的实时性比非占先式的内核要好得多。目前，它支持 x86、ARM、

PowerPC、MIPS 等众多体系结构，并有上百个商业应用实例，其稳定性和可靠性是经过实践验证的。μC/OS-II 版以上的内核都具有可抢占的实时多任务调度功能，另外它还提供了许多系统服务，例如信号量、消息队列、邮箱、内存管理、时间函数等，这些功能可以根据不同的需求进行裁减。可以说，μC/OS-II 是一个具备现代操作系统特点的 RTOS，同时它结构清晰、注解详尽，具有良好的可扩展性和可移植性，被广泛地应用于各种架构的微处理器上。

在本系统中选用 μC/OS-II 作为嵌入式操作系统，与其他操作系统比较，主要优势体现在[21]：

1．容易获取、结构简单[25]

μC/OS-II 实时内核在网上比较容易获取，而且齐全，特别是可以直接从作者网上免费下载，采用标准 C 和汇编语言编写，其中绝大部分用 C 语言，能够为大多数人接受，而且结构非常简洁。

2．容易上手、适合学习

μC/OS-II 的中文文档比较齐全，中文书籍早有出版，并且写得比较好，本身源码公开并且注释比较详尽，方便学习。相对源于 Linux 家族的操作系统，代码量小、文档更容易找到，也容易理解，中文版的资料更少了。

3．容易进行应用程序开发且移植性好

μC/OS-II 由于容易理解，并且源码公开，开发比较简单。μC/OS-II 的可移植性非常好，只要熟悉一下处理器的情况就可以很容易移植到各种处理器上，移植过程中工作量小。

4．适用性强、稳定性好

μC/OS-II 可以适用于任何内核的 ARM，包括 ARM7TDMI。别的操作系统往往需要 MMU 支持，而目前市面上常见的 ARM 芯片往往是 ARM7TDMI 的，这样使用 μC/OS-II 就比较方便了。

5．可裁减性、实时性好

μC/OS-II内核可以根据需要裁减和固化，最小内核的ROM为2KB，RAM空间决定于任务的多少。任务的切换为可剥夺性，每个时刻运行的是优先级最高的任务，能保证实时性。

2.3.3 嵌入式处理器选型

根据第一章的研究背景和课题意义，本系统的嵌入式处理器是选择ARM RISC处理器，但是现在ARM处理器类型很多，大部分都是针对某一方面的特殊要求而设计的，比如三星的大部分ARM处理器特别适合网络设备的应用和开发。在本系统中，我们既要考虑以太网的接口设计，同时还得考虑CAN这一侧的设计

需要，也包括整个系统的应用需要，这又得考虑和综合诸多因素，并进行均衡。从应用角度来说，ARM芯片的选择一般需考虑以下这些原则[26]：

(1)ARM 芯核对操作系统的支持，一般 WinCE 或 Linux 等要求处理器带 MMU(memory management unit)功能，ARM7TDMI 没有 MMU，μCLinux 和 μC/OS-II 不需要 MMU 支持，可以应用于 ARM7TDMI 核[27]。

(2)系统时钟控制器，决定了 ARM 芯片的处理速度。ARM7 的处理速度一般为 0.9MIPS/MHz，常见的 ARM7 芯片系统主时钟为 20MHz～133MHz；ARM9 的处理速度为 1.1MIPS/MHz，常见的 ARM9 的系统主时钟为 100MHz～233MHz；ARM10 最高可达到 700MHz。不同芯片对时钟的处理不同，有的芯片只有一个主时钟频率，这样的芯片可能不能同时顾及 USART 等时钟的准确性，如 Cirrus Logic 的 EP7312 等；有的芯片内部时钟控制器可以分别为 CPU 核和一些片内外围提供不同的时钟，如 PHILIPS 的 SAA7550 等芯片[28]。

(3)内部存储器容量，在不需要大容量时可以尽量选用有内置存储器的 ARM 芯片，免去扩展芯片的麻烦，造成系统复杂，影响处理速度和增大系统功耗。

(4)根据具体应用的侧重点，有针对性的选择带合适外围的芯片。比如考虑到网络应用需要，可以选择三星的处理器，是否需要 SPI 总线、AD 等。当然这只是一部分因素。还可以需考虑外设接口和并行 I/O 口复用的情况等等。

由于我们已经将嵌入式操作系统选为 μC/OS-II，根据原则(1)，将处理器选为 ARM7TDMI 核，这种核的处理器种类多，价格便宜，获得的渠道也多。综合原则(2) (3) (4)，我们必须考虑串行口通信波特率时钟的准确性，而且以太网接口、CAN 接口等在数据收发时也需要时钟频率来保证数据传送的准确性和同步性。另外，系统将在工业场合长期运行，处理器的尺寸大小、工作温度范围、内部存储器容量等直接决定着系统的抗干扰性和低功耗，同时我们希望本接口系统能分担控制系统中心处理机的部分负担，要求处理器的运算能力强大，考虑这些因素我们决定选用 ATMEL 的 ARM7TDMI 核处理器 AT91FR40162[29~30]。

2.3.4 AT91FR40162 处理器

AT91 系列嵌入式微处理器的主要特点是功耗低、非常适合于实时控制应用。AT91FR40162[29]是其中的一员，它采用 ATMEL 的高密度 CMOS 技术，能在一个单片上集成了 ARM7TDMI 核和大量的 Flash、片内 RAM 以及各种外围功能模块。这样的高集成度和非常小的脚印使其可以理想地用于对空间有限制的场合，为许多需要强大运算功能的嵌入式应用提供了高度的灵活性和高性能价格比的解决方案。

1. AT91FR40162 的特点

AT91FR40162[30]由 ARM7TDMI 核处理器 AT91R40008 和一片 16Mbit 的 Flash AT49BV1604A/1614A 集成而成。它采用单一紧凑的 121—ball BGA 封装，内部除 16Mbit 的 Flash 存储器外还集成 2Mbit 的片内 SRAM，片内集成的 SRAM 使得 CPU 性能高达 60 MIPS，比在外部扩展 SRAM 的实现方式降低了系统的功耗。Flash 存储器可使用单一器件供电，通过 JTAG/ICE 接口或厂家编程的 Flash Uploader 程序进行编程，使得 AT91FR40162 能实现在系统编程 ISP 应用。采用 BGA 封装具有芯片面积小，可以减少应用 PCB 板的面积，但是需要专用的焊接设备，无法手工直接焊接。

与一般的 AT91 系列嵌入式微处理器不同，在实际应用时需要注意以下几点：

①在 AT91FR40162 内部除了 Flash 存储器使能信号以外的所有地址、数据和控制信号，都是内部互连的。

②内部的 256KB SRAM 直接与 32 位的数据总线相连，可实现单周期访问，使用 ARM 指令集，在 66MHz 下可以提供 60MIPS 的最高性能。

③内部的 Flash 存储器通过 EBI 访问。它的主要功能是作为程序存储器。应用时应该把 Flash 存储器使能信号(NCSF)与 EBI 上的低电平有效的片选信号之一相连接。当作为引导存储器使用，那么必须使用 NCS0，同时 BMS 必须由外部拉低，以使处理器在复位后可以完成正确的 16 位取指操作。特别是引导时必须为 EBI 配置正确的标准等待状态数目，当微处理器为 66MHz 时需要 5 个标准等待状态。

④分开的 MCU 和 Flash 存储器复位输入(NRST 和 NRSTF)加大了系统的灵活性，方便用户自由地根据应用选择复位操作。对于 Flash 存储器的复位输入引退 NRSTF，当 NRSTF 输入为逻辑高电平时，存储器处于标准操作模式；当输入为低电平时，暂停当前的存储器操作并使它的输出置于高阻状态。

⑤AT91FR40162 集成了一个 AT91 Flash Uploader 的驻留引导软件。通过 AT91 Flash Uploader 软件能够向 Flash 存储器加载应用软件。

⑥用户必须确保所有的 VDDIO，VDDCORE 和所有的 GND 引腿通过最短的路线连接到各自的电源。Flash 存储器在读模式下上电。

⑦仿真功能：在三态模式下，提供通过外部引腿对 Flash 的直接访问，这使得在装配板子之前可以使用传统的 Flash 编程器对 Flash 进行编程。

⑧引导模式选择：

如果嵌入的 Flash 存储器用作引导存储器，则 BMS 输入必须由外部拉低，并且 NCS0 必须在外部与 NCS7 连接。

⑨Flash 存储器通过 EBI 以 16 位字寻址，使用地址线 A1～A20。

2．AT91FR40162 的资源

由于 AT91FR40162 处理器是由 AT91R40008 处理器和一片 2MB 的 Flash 集成而成，对照 AT91FR40162 的结构框图图 2-5[30]，具体的资源如下：

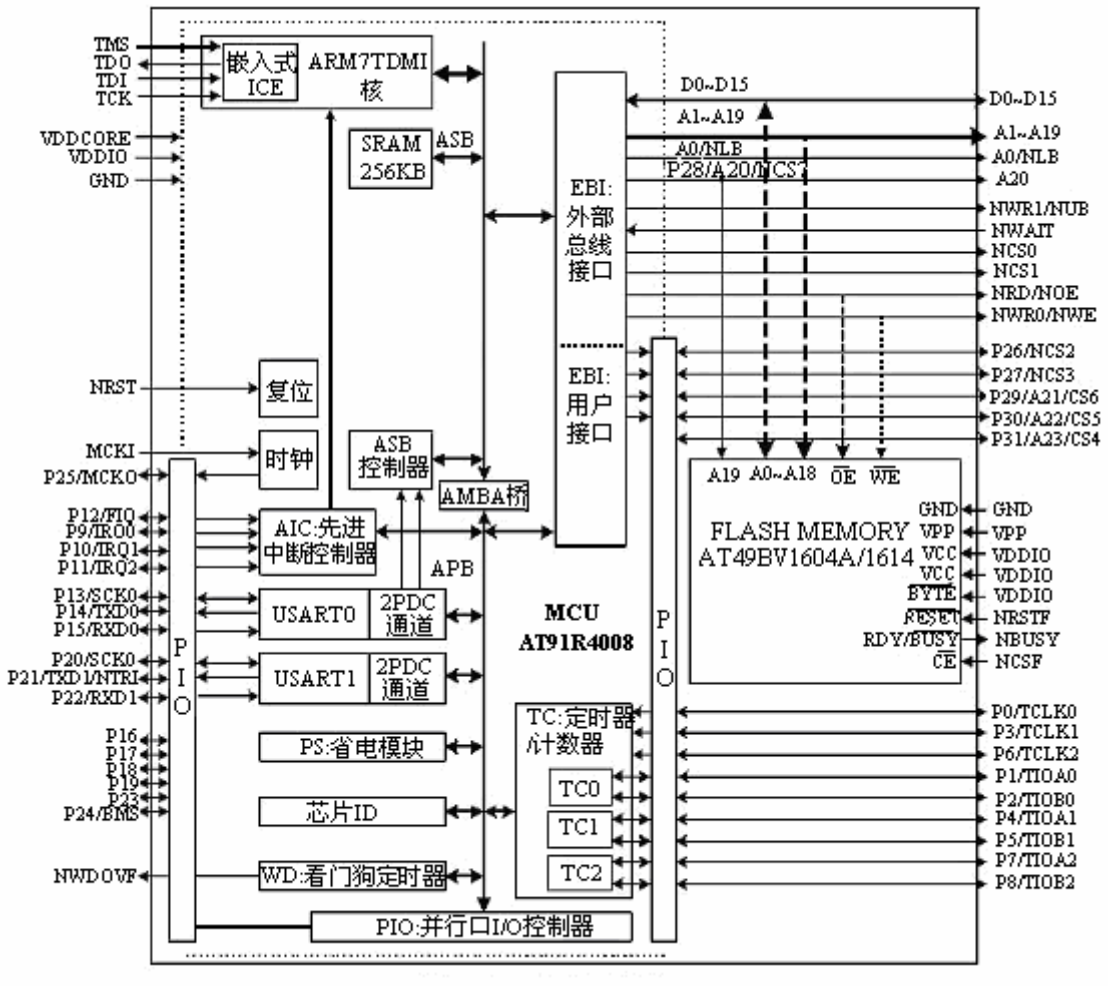

图 2-5 AT91FR40162 的功能模块结构图

- 集成了 ARM7TDMI ARM Thumb 处理器内核
  ---高性能的 32 位 RISC 体系结构
  ---高代码密度的 16 位指令集
  ---MIPS/Watt 的领先者
  ---嵌入式 ICE(In-Circuit Emulation)
- 256KB 片内 SRAM
  ---32 位数据总线宽度
  ---单时钟周期访问
- 完全可编程的外部总线接口(EBI)
  ---最大可寻址空间为 64MB
  ---最大可达 8 个片选线

---软件可编程的 8 位或 16 位外部数据总线

- 8 优先级、可单独屏蔽的向量中断控制器。
- 32 个可编程 IO 线。
- 3 通道的 16 位计数器。

  ---3 个外部时钟输入

  ---每通道有两个多功能 I/O 引脚
- 2 个 USART 串行通讯接口。

  ---每个 USART 有两个专用外围数据控制器(PDC)通道
- 可编程看门狗定时器。
- 先进的省电特性

  ---CPU 和外围可以单独静止
- 全静态工作：

  ---1.8V 时，0Hz～16MHz；

  ---2.7V 时，0Hz～33MHz；

  ---3.0V 时，0Hz～40MHz
- 16 Mbit Flash 存储器

  ---单电压读/写，90ns 访问时间

  ---20us 的快速字编程时间，200ms 的快速扇区擦除时间

  ---双存储区结构允许同时进行读和编程/擦除

  ---低功耗操作：25mA 活动电流，10μA 待机电流
- 全静态工作。在 VDDCORE 为 1.65V 且 85℃下的内部频率范围为 0Hz～70MHz.
- 2.7～3.6V 的 I/O 工作电压范围，1.65～1.95V 的内核工作电压范围。
- -40℃～85℃的温度范围。
- 121-ball 10×10×1.2mm 的 BGA 封装。

2.4 小结

本章以嵌入式系统的开发为背景介绍了系统的设计方法、结构、开发工具等，完成了系统开发工具的选择，系统的总体设计和功能规划，分析了本系统软硬件的选型过程。

第三章 系统硬件平台的设计

完成了系统的功能规划和系统选型后，关键是根据处理器特点和实际应用需要，建立系统的硬件平台。在本章中主要介绍整个系统的硬件实现过程，由于硬件外围较多，介绍时以各功能模块电路为单位进行阐述，其中重点分析了本系统的关键通信电路 CAN 接口电路和以太网接口电路的实现过程。在本章后面也介绍了存储器扩展方法、PCB 设计注意问题以及硬件调试等。

3.1 单元电路的设计

CPU 与外部设备、存储器的连接和数据交换都需要通过接口设备来实现，前者被称为 I/O 接口，而后者则被称为存储器接口。存储器通常在 CPU 的同步控制下工作，接口电路比较简单；而 I/O 设备品种繁多，其相应的接口电路也各不相同。因此在设计 I/O 接口电路是要解决 CPU 和外围设备之间的时序配合和通信联络问题、数据格式转换和匹配问题以及 CPU 的负载能力和外围设备端口选择问题。本节将以各功能电路为单位来介绍整个系统的实现过程。

3.1.1 CAN 接口电路

CAN(Controller Area Network)也称控制器局域网[33~34]，由Bosch公司制定，是目前应用最广泛的现场总线之一。它的主要特性有：

(1) 低成本；
(2) 极高的总线利用率；
(3) 很远的数据传输距离(长达10Km)；
(4) 高速的数据传输速率(高达1Mbit/s)；
(5) 可根据报文的ID决定接收或屏蔽该报文；
(6) 可靠的错误处理和检错机制；
(7) 发送的信息遭到破坏后，可自动重发；
(8) 节点在错误严重的情况下具有自动退出总线的功能；
(9) 报文不包含源地址或目标地址，仅用标识符来指示功能信息、优先级信息。

本嵌入式通信接口系统是工业底层控制网络和上层以太网的通信中介，因此 CAN 接口以及下一节的以太网接口是整个系统能否正常工作的关键。这里 CAN 接口不仅要能与低层的设备进行数据通信，而且所有的低层信号都必须通过该接口送往上层的以太网[35]。

对于 ARM 这一侧实现系统的 CAN 接口扩展可以采用两种方式：一种是直接采用 CPU 总线进行扩展，另一种是采用 I/O 口扩展。由于 ARM 本身携带的 I/O 引脚是复用的信号线，在复杂应用系统中能够使用的 I/O 资源非常有限，而

实现 SJA1000 的读写至少需要 12 个信号线(8 根数据线、读写信号、片选和地址锁存信号 ALE)，如果完全采用 I/O 方式，很难从系统获取足够的资源，同时 I/O 方式比总线方式要慢，所以采用总线方式是比较好的选择 [11、36~37]。

使用总线方式扩展 CAN 接口电路时首先要解决 ARM 处理器与 CAN 接口芯片 SJA1000[38]的时序适配问题。由于 SJA1000 是典型的 MCS-51 时序，与 ARM 的读写时序不同，需要加入转换电路，将 ARM 的标准读写时序转换为典型的 MCS-51 时序[36]。当然也可以采用软件实现，但是将加大软件开发的复杂度。在本系统中采用 GAL22V10B 构造出 ADDR 和 DATA 来区分数据和地址、ALE 来实现地址的锁存使能，来实现 AT91FR40162 对 SJA1000 的控制。这个方法硬件系统比较复杂，需要牺牲一定的 CPU 处理速度为代价，但是实现起来比较直观。电路原理图见图 3-1。

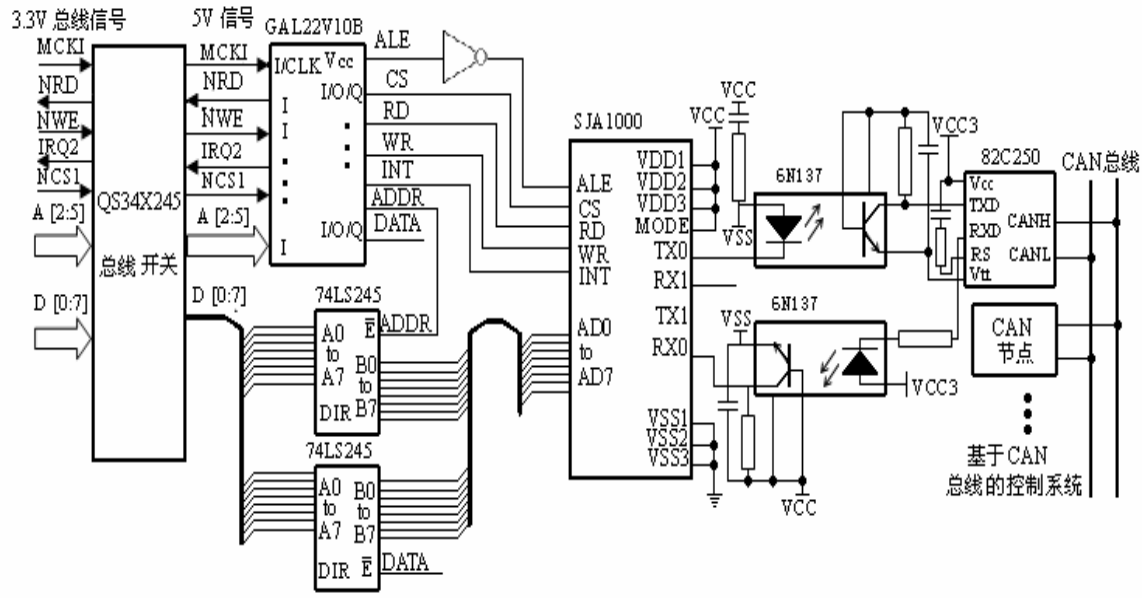

图 3-1　CAN 扩展原理图

ARM 处理器的地址和数据总线分开，可配置成 8、16 或 32 位模式，在本系统中扩展 CAN 接口电路时将之配置为 16 位模式，数据和地址一次读到总线上，通过 GAL22V10B 构造 ADDR、DATA 来区分。详细电路图见图 3-2。

扩展时还须注意 ARM 芯片 3.3V 电压与 SJA1000 5V 电压的连接问题，这里采用 QS34X245 总线开关芯片；AT91FR40162 的地址总线是复用的，8 个片选线中 4 根与 A20～A23 公用，在扩展过程中没有用复用的腿 A20～A23；为了加强 CAN 总线上信号的抗干扰能力加入了高速光耦 6N137，且光耦的供电采用独立的隔离直流电源。CAN 总线上还并联一个匹配电阻(150 欧)和两个 TVS 管，CANH 和 CANL 与地之间并联的 30pF 的小电容，这样可以消除总线上的高频干扰。

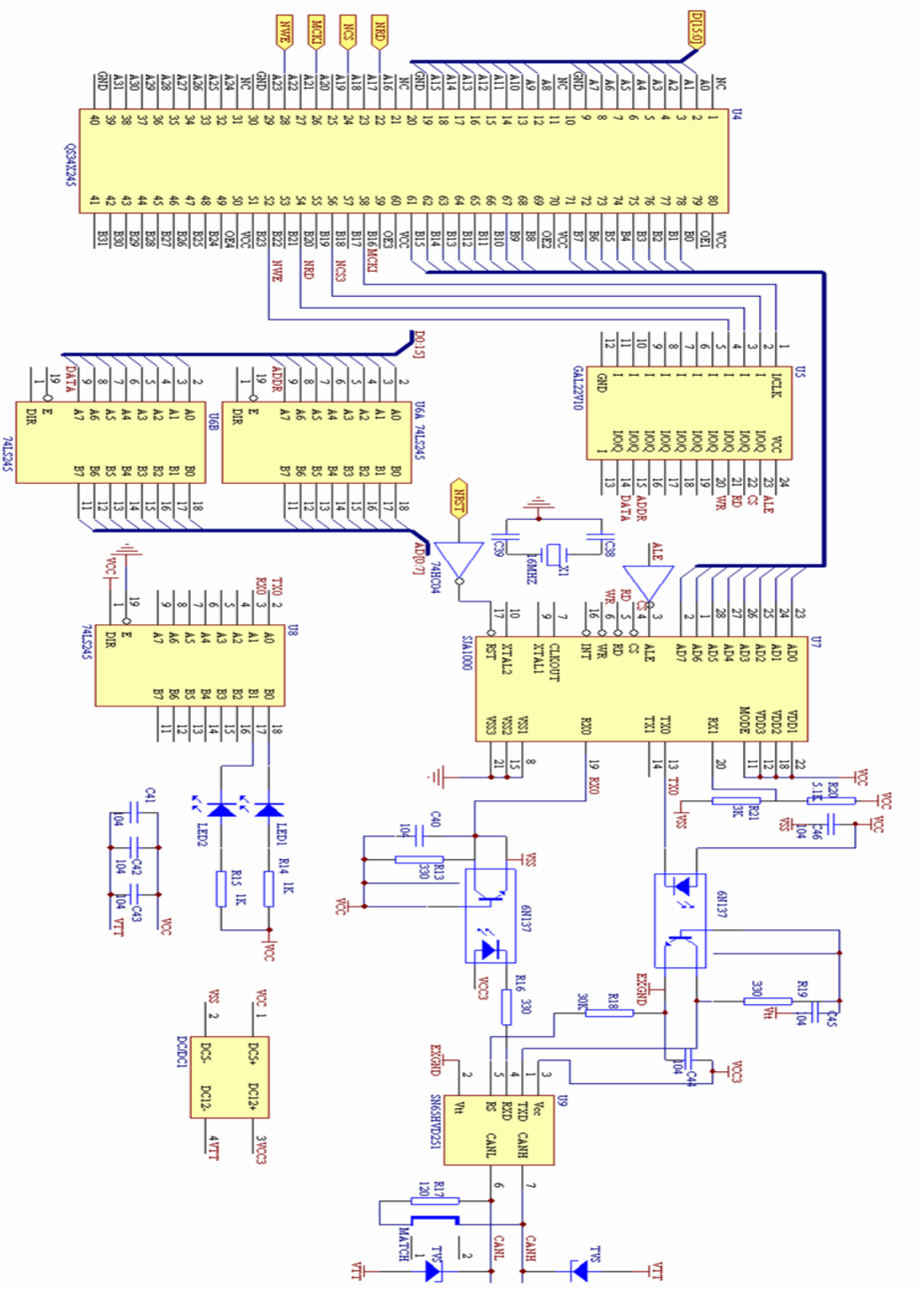

图 3-2 CAN 接口电路图

3.1.2 以太网接口电路

在 Internet 普及的今天，工业以太网是当今控制系统的发展趋势，在嵌入式设备上直接扩展以太网是构建和实现分层控制系统最方便的方式之一，同时也是嵌入式产品独特优势的体现。

在嵌入式系统中扩展以太网接口的方法有两种。

1．嵌入式处理器+网卡芯片(RTL8019AS等)

采用这个方法不要求嵌入式处理器带有通用网络接口，只要把以太网芯片连接到嵌入式处理器的总线上即可。这样通用性强，不受处理器的限制，但是处理器和网络数据交换通过外部总线(并行总线)交换数据，速度慢，同时系统可靠性不高，电路板布线复杂些。

2. 带有以太网络接口的嵌入式处理器

采用带有通用网络接口的嵌入式处理器(如MII接口)，处理器和网络数据的交换通过内部总线进行，速度快、可靠性也高一些。

由于本系统中以太网接口是作为与控制系统中以太网层的接口，而 AT91FR40162 没有内嵌 MAC 控制器，因此采用方法 1 扩展，网络接口芯片选用 RTL8019AS[39]。

RTL8019AS 是性价比高且带有即插即用功能的全双工以太网控制器，它的主要特点：符合 Ethernet II 与 IEEE802.3 标准；全双工，收发可同时达到 10Mbit/s 的速率，内置 16kB 的 SRAM，用于收发缓冲，可减低对主处理器的要求；支持 UTP、AUI、BNC 自动检测，还支持对 10BaseT 拓扑结构的自动极性修正；允许 4 个诊断 LED 引脚编程输出；RTL8019AS 内部有 2 块 RAM 区，1 块为 16KB，地址为 0x4000~0x7fff，按页存储，每 256 字节为一页，1 块为 32 字节，地址为 0x0000~0x001f。

RTL8019AS 与主机有 3 种接口工作模式：跳线方式，网卡的 I/O 和中断由跳线决定；即插即用方式，由软件进行自动配置即插即用；免跳线方式，网卡的 I/O 和中断由外界的 93C46 中的内容决定。

本系统中将 RTL8019 工作模式选为免跳线模式，适合应用于嵌入式环境，设计 3 个 LED 状态灯来监视以太网的工作状态，EEPROM 93C46A 用来保存以太网的配置信息，包括 MAC 地址[40~41]。具体接口电路见图 3-3 所示。

连接时，为保证 RTL8019AS 复位后 I/O 模式的基地址为 0x300h，SA8、SA9 置高，同时在 I/O 模式下要将 SMEMRB、SMEMVB 脚置高；为保证数据为 16 位模式传输，MCU 访问 RTL8019AS 之前提供给 AEN 一个由高到低、再由低到高变化的电平信号[42]。

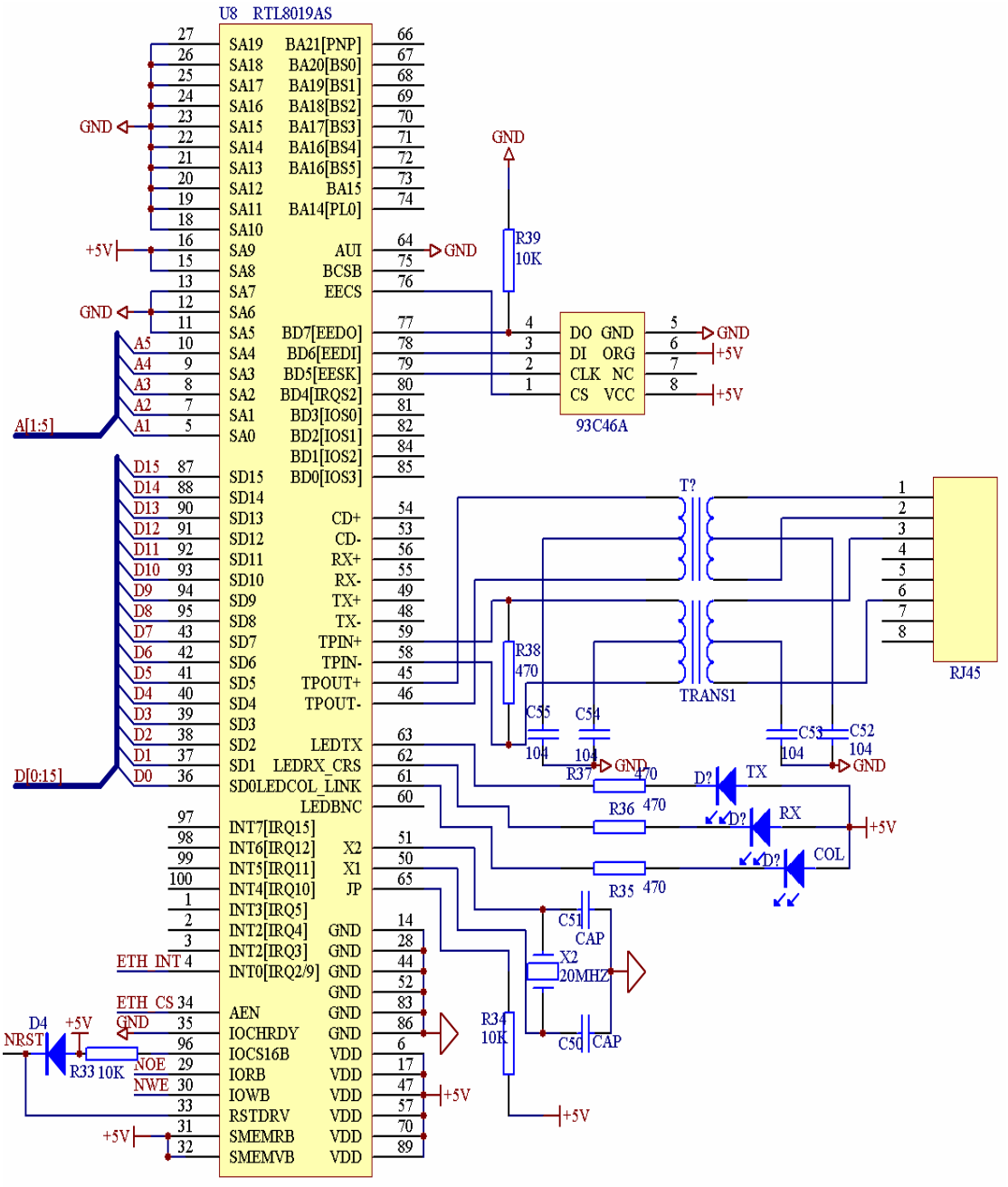

图 3-3 以太网接口图

3.1.3 USB 接口电路

USB(Universal Serial Bus)通用串行总线以其通信速率快、使用便捷已经成为当今计算机的一个标准外设接口标准。考虑到系统升级的需要，本系统中扩展了 USB 接口电路。

本系统 USB 接口电路如图 3-4，接口芯片采用 PDIUSBD12，符合 USB1.1 的标准，应用时可以通过该接口接入一个 U 盘来实现对过程信息的数据备份；因此也可以看作是一个扩充的存储器，在文件系统建立好后，可以方便地实现系统程序更新和升级[11、43]，将来可以通过该 USB 口设计一个数据采集系统。

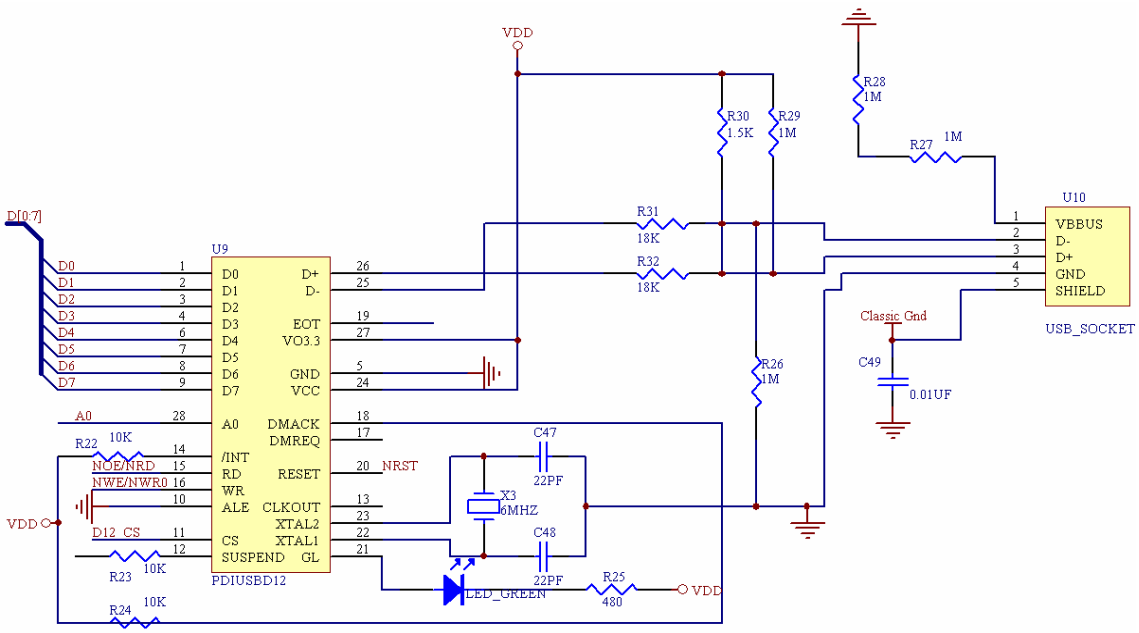

图 3-4 USB 接口电路图

3.1.4 2 个 RS-232 串行接口电路

在嵌入式系统的开发和应用中，经常需要使用串行口来配合JTAG实现对目标系统的调试；同时也习惯通过上位机本身配置的串行口来和目标系统进行通讯，以实现对过程运行情况进行监视和控制。本系统的串行通信口电路如图3-5。

系统中 USART0 用来与上位机通信，以使用户能通过串口对系统进行动态调试。由于 AT91FR40162 通过自带的 Flash Uploader 通过串口可以直接对片内的 Flash 进行烧写，所以本系统中的 USART1 专门用来支持 Flash 编程。这里接口芯片选用 MAXIM 公司的 MAX3232[42,44]，工作电压为＋3.3V，符合 AT91FR40162 的电压要求。另外说明一点，嵌入式产品开发时一般有 JTAG 仿真器，目标系统程序加载习惯通过它来实现，因此在本系统中程序加载一般通过 JTAG 写入。

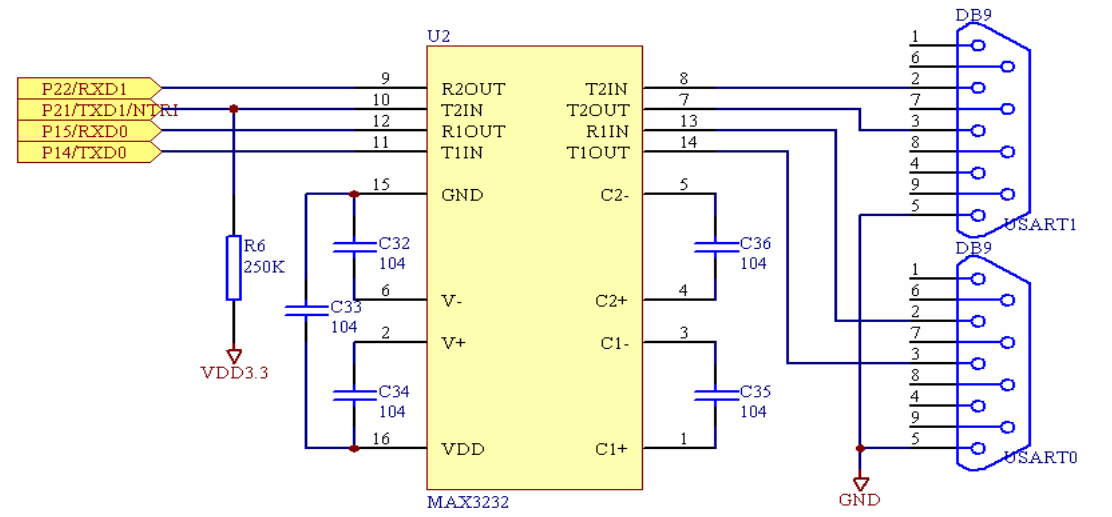

图 3-5 串行通信接口电路

3.1.5 电源电路

AT91FR40162 的 VDDCORE 引脚，为芯片内核供电，1.8V；VDDIO 引脚，为 I/O 口线供电，3.3V；VPP 为快速编程/擦除电压，因此电压种类比较多，在电源电路设计时必须采用电源转换块。一般用线性稳压芯片来实现，这样不要外围元件，节约成本，但是线性稳压器都要求输入电压比输出电压高 2V～3V 以上。

本系统的电源电路如图 3-6，为提高系统电源电路的供电质量和系统供电安全，首先将电源通过网桥 BRIDGE 整流，再经过 C2、C3 滤波，然后通过 78M05 将电源稳定至 5V，作为系统扩展 5V 的外围用；再分别通过低压差电源芯片稳压输出 3.3V 和 1.8V 的电压，供给 AT91FR40162 以及系统中扩展的 3.3V 的外围。这样输出电流大、精度高、稳定性好、功耗低。这里输入电压为 7V，电源转换芯片采用 National Semiconductor LM317MPX3.3 的 DC-DC 转换器完成 5V 到 3.3V 和 1.8V 的转换。

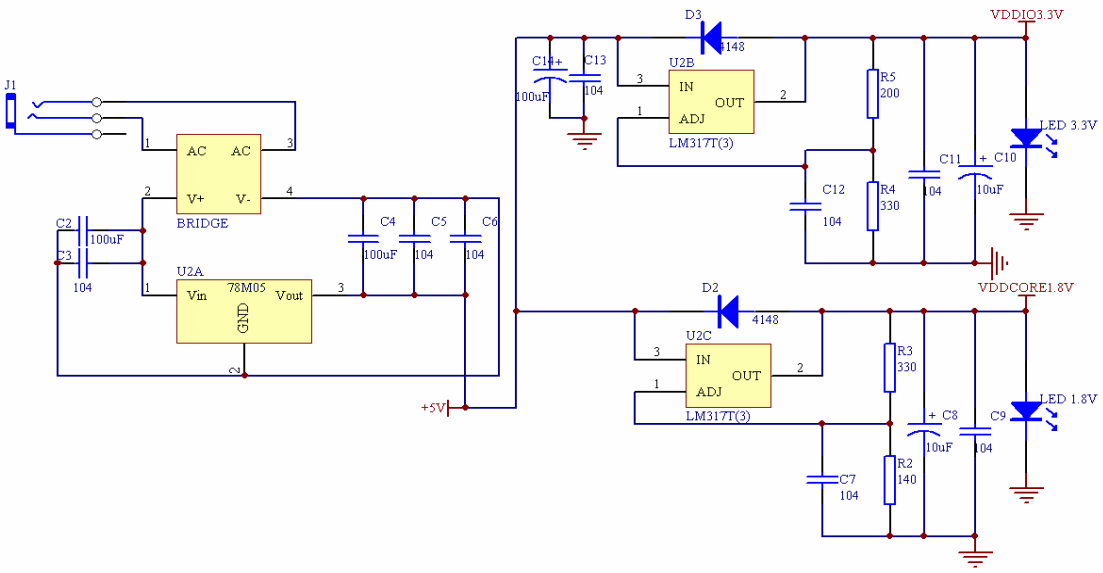

图 3-6 电源电路图

在扩展外设时要注意电源类型选择，因为 5V TTL 和 3.3V TTL 的逻辑电平是相同的，而 5V CMOS 逻辑电平与前两者是不同的，因此在设计 I/O 接口时要留意 3.3V 设备连接到 5V 设备逻辑电平是否匹配。另外，内核电压为 1.8V，因此 1.8 V 电源的电压精度要高些，以使系统有好的负载调整和瞬态响应能力[41]。

3.1.6 JTAG 仿真器电路

JTAG(Joint Test Action Group)是 1985 年制定检测 PCB 和 IC 芯片的一个标准，1990 年被修改后成为 IEEE 的一个标准，即 IEEE1149.1-1999。本系统 JTAG 接口为标准的 20 脚 JTAG 接口[45]，见图 3-7。引脚定义如下：

TCK－测试时钟输入；

TDI—测试数据输入，数据通过TDI输入JTAG；

TDO—测试数据输出，数据通过TDO从JTAG口输出；

TMS—测试模式选择，TMS用来设置JTAG口处于某种特定的测试模式；

TSRT—测试复位，可选引脚，输入引脚，低电平有效。

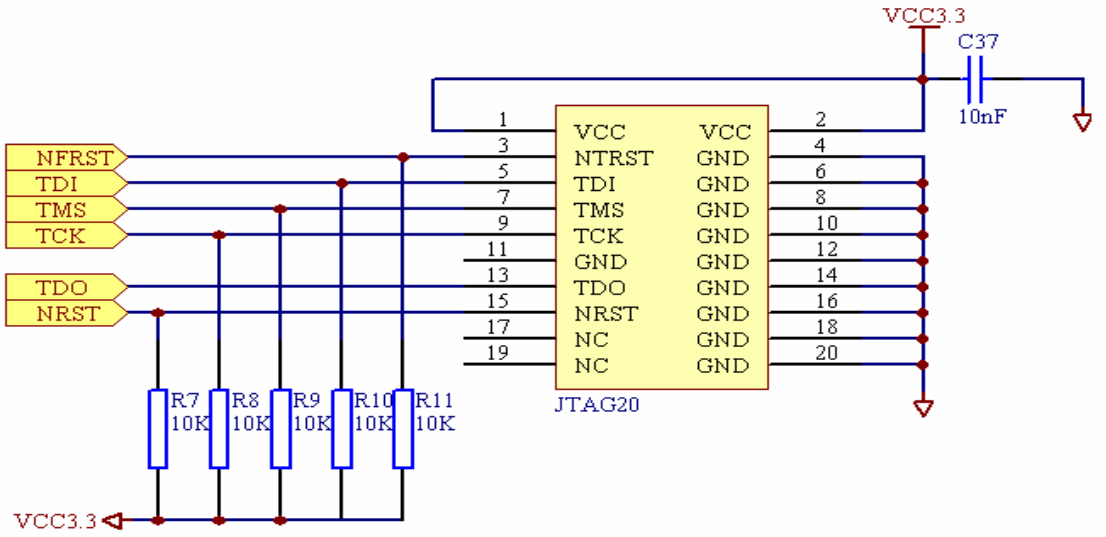

图 3-7 JTAG 电路图

3.1.7 系统复位电路

复位电路主要完成系统的上电复位和系统在运行时用户按钮复位功能。见下图 3-8，考虑到 AT91FR40162 的内部 2MB 的 Flash 复位，此处将 NFRST 和 NRST 接在一起，达到同步。

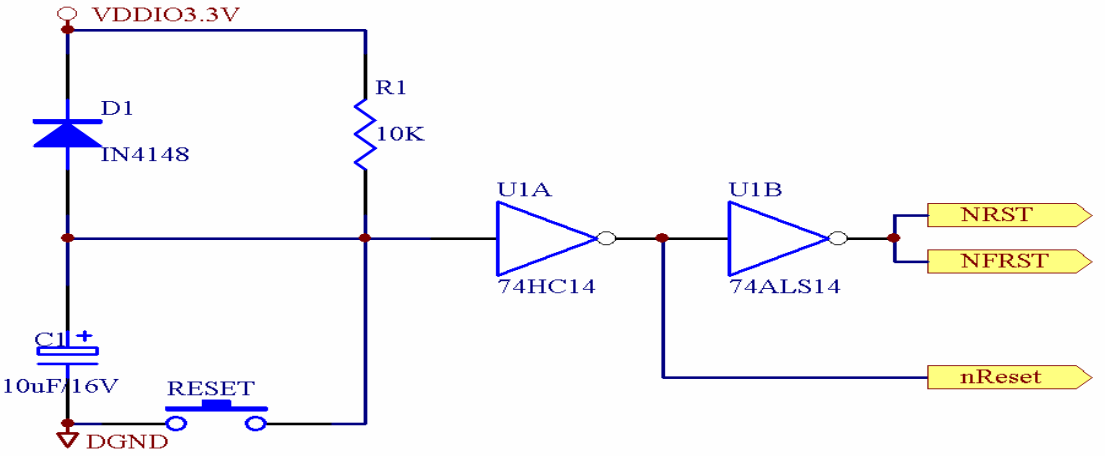

图 3-8 系统复位电路

它的工作原理：在系统上电时，通过电阻 R1 向电容 C1 充电，当 C1 两端的电压未达到高电平的门限电压时，NRST 端输出为低电平，系统处于复位状态；当 C1 两端的电压达到高电平的门限电压时，NRST 端输出为高电平，系统进入正常工作状态。当用户按下按钮 RESET 时，C1 两端的电荷被泻放掉，NRST 端

输出为低电平，系统进入复位状态，再重复以上的充电过程，系统进入正常工作状态。两极非门电路用于按钮去抖动和波形整形；nReset 端的输出状态与 NRST 端相反，以用于高电平复位的器件；通过调整 R1 和 C1 的参数，可调整复位状态的时间。

3.2 存储系统的扩展

由于 AT91FR40162 处理器在片内集成了不少外围，与其他的 ARM7TDMI 处理器不同，对片外存储器和外部外设包括部分片内外设的访问都是通过外部总线接口 EBI 接口进行，如片内的 2MB 的 flash。产生这些的主要原因在于 AT91FR40162 外部地址总线最大为 24 位，CPU 内部的 ARM7TDMI 核是 32 位，它们间的连接必须通过 EBI 转换。而且地址线的高四位(A20~A23)与片选线(CS4~CS7)是复用，这种通过将地址线转换为片选线，可以优化 EBI 以适合其外部存储器的要求，使用户可以获得更多的外部器件或对每个器件而言更大的地址范围。另外 AT91FR40162 有地址重映射功能，系统的存储器地址是可变化的。

这样，虽然使处理器的功能和灵活性得到增强，但与其它的处理器差异也增大。虽然在本系统中 CPU 片内有一定的 SRAM 和 FLASH，考虑到今后需要，有必要特别介绍 AT91FR40162 的存储器扩展。

(1) 处理器存储器地址空间分布

存储器系统的地址空间分布三个部分：处于低地址的内部 4MB 空间，主要用来给片内的 Flash；处于中间地址专门保留给 EBI 控制的外部器件(外部的存储器或外设)；高 4MB 地址空间保留给片内外设。注意系统的启动一般是通过片选线 NCS0 上外接的片外 Flash 来实现，而本系统是通过 AT91FR40162 的片内 2MB 的 Flash 来启动，它的位置定义在 0x0100,0000 处。

(2) 存储器扩展方法

AT91FR40162 处理器在扩展存储器时：如果外接存储器为 8 位，则地址线 A0～A23 有效；为 16 位时，地址线 A1～A23 有效。同时片选线 CS4、CS5、CS6、CS7 与地址线高四位 A20～A23 复用，因此可能出现以下的组合：

A20，A21，A22，A23；

A20，A21，A22，CS4；

A20，A21，CS5，CS4；

A20，CS6，CS5，CS4；

CS7，CS6，CS5，CS4；

每种组合下存储器的片数，单个存储器的最大容量和访问方式都有比较大的

差异，具体实现可以通过 EBI_CSR 的相应位进行控制。

典型的连接方法如图 3-9、3-10[18、30]：

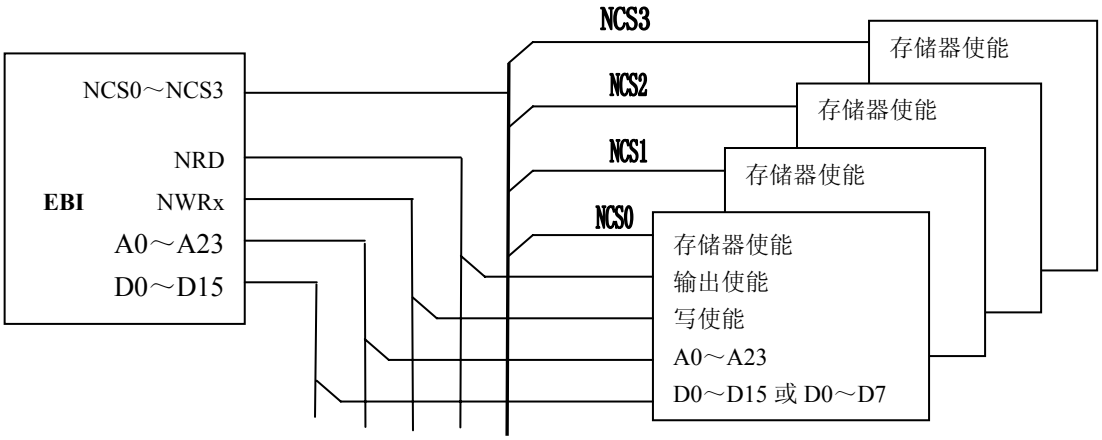

图 3-9　4 个外部存储器连接图

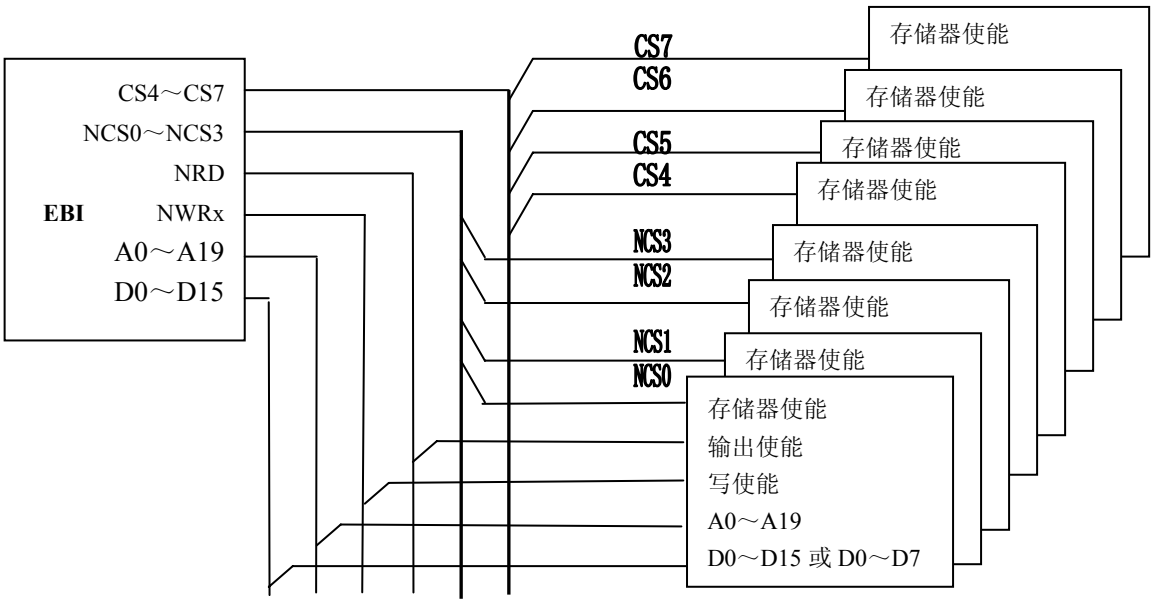

图 3-10　8 个外部存储器连接图

下面以 NCS2 上连接存储器为例示范扩展方法：

1) 图 3-11 是在 NCS2 上连接一个 512K*8 位的存储器扩展方法。

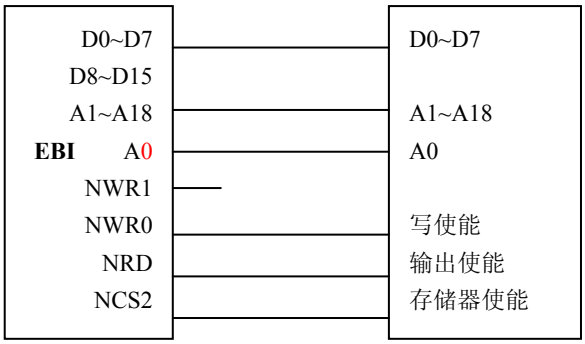

图 3-11　8 位数据总线的连接

2) 图 3-12 是在 NCS2 上连接一个 512K*16 位的存储器

图 3-12 16 位数据总线的连接

3) 在 NCS2 上连接一个 512K*16 位的存储器，可完成 8 位或 16 位访问。

AT91FR40162 通过 EBI_CSR 的 BAT 位域的控制，将 A0/NLB 作为 NLB 来使能低字节读/写；NWR0/NWE 作为 NWE 来使能写 8 位或 16 位；NRD/NOE 作为 NOE 使能读 8 位或 16 位，这样可以同时支持 8 位或 16 位的访问方式。如下图 3-13。

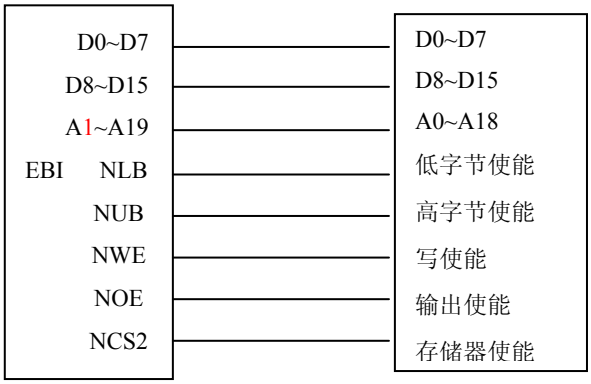

图 3-13 可 8 位或 16 位访问的 16 位存储器数据总线的连接

4) 图 3-14 是在 NCS2 上连接两个 512K*8 位的存储器

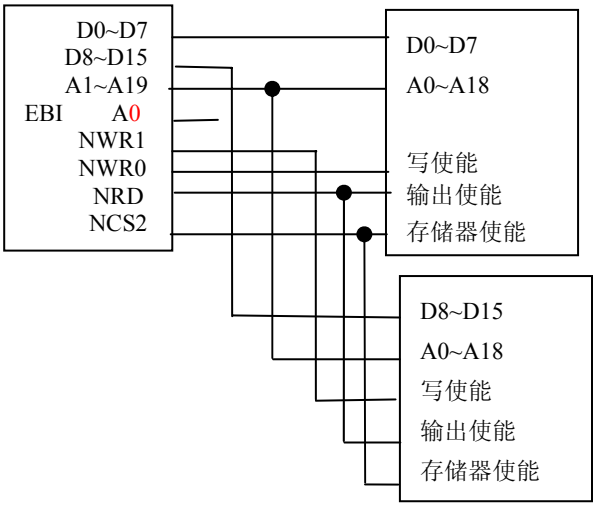

图 3-14 两个 8 位存储器数据总线连接

这里是用两片 8 位的存储器组成 16 位的存储器，并可以通过 8 位或 16 位方式访问。此时 NRD/NOE 作为 NRD 使能 8 位读或 16 位读；NWR1/NUB 作为 NWR1

使能高字节写；NWR0/NWE 作为 NWR0 使能低字节写。

因此，AT91FR40162 的存储器系统扩展比较灵活，能依用户的需要进行配置和扩展，方便实际需求。

3.3 系统硬件 PCB 设计

要使电子电路获得最佳性能，尤其是本系统要应用于工业环境，因此在 PCB 设计过程中，元器件的布局以及导线的布设是很重要的[46~47]。

3.3.1 PCB 板的布局和布线

首先，要考虑 PCB 尺寸，PCB 尺寸过大时，印制线条长，阻抗增加，抗噪声能力下降，成本也增加；过小，则散热不好，且邻近线条易受干扰，本系统的 PCB 选为 145×93mm。布局时先确定特殊元件的位置，再根据单元电路对具体元件进行布局。

1．特殊元件的布局

(1).系统中 CPU 和 RTL8019 的晶振属于高频元件，所以先将 CPU 的晶振紧挨 CPU 布置在板上的左下角，以太网的晶振也紧靠 RTL8019，这样可以减少它们的分布参数和相互间的电磁干扰以及对其它器件的影响。

(2).由于 AT91FR40162 是贴片式的 BGA 封装，本系统中特意将 CPU 独立制成一个子板，因此在扩展引线插槽时对不同类型引线进行了分组，避免带入干扰。

2．单元电路的元件布局

(1).单元电路的位置一般按照信号的流程安排，在本硬件系统中将两个串行口、USB 接口、以太网接口的接入口布置在板上的正上方，JTAG 口布置在左上角，这样使得布局便于信号流通，信号流尽可能保持一致方向。

(2).在各单元电路中的元件布局时，以各自核心芯片为中心，围绕它进行布置，这样元器件分布均匀、整齐，可以减少和缩短元件间的隐现和连接。

(3).以太网电路和 CAN 接口电路在高频下工作，串行通信口工作频率也比较高，因此特别考虑了各元器件上的分布参数，将元器件平行排列这样减小了高频的影响，增强了上述电路的抗干扰能力。

(4).电源从右上方进入印刷电路板，为提高系统的电源质量，加入了滤波器，同时电源插孔在板上离 RJ45 的驱动芯片位置比较近，引脚共接一个滤波电容，可以消除低频噪声对系统的影响。

3．PCB 板的布线

(1).主要输入输出端口的导线尽量避免相邻平行，并尽量短些，在本系统中特意为它们布置了线间地线。

(2).在印制导线拐弯处采用圆弧形，避免使用大面积铜箔，需要时，采用栅格状，利于排除铜箔与基板间粘合剂受热产生的挥发性气体。

(3).在本系统的印刷电路板中，成组的信号线有数据线、地址线和电源线，在元器件位置确定后，先完成它们的布线，尽量做到了成组、平行分布，同时尽可能的短。这样可以减少干扰，增加系统的稳定性，还可以使布线变得简单。

3.3.2 PCB 及电路抗干扰措施

1．电源线设计

AT91FR40162 的电源引线比较多，功能、电压差异都比较大，因此在设计电源线时特别做了以下几点处理：

(1).尽量增大电源线的宽度，以减小环路电阻；同时将电源线、地线的走向与数据传递的方向保持一致，可以减小噪声引入。

(2).设计时将数字地和模拟地分开。在板上将逻辑电路和线性电路分开，串口等低频电路的地采用单点并联接地；以太网等高频电路的地采用多点串联接地。

(3).尽可能加粗接地线，可以减小电流的变化对接地电位的影响，增强抗噪声能力。

(4).将各种接地线组成独立的闭环路，以提高系统抗噪声能力。

2．去耦电容配置

在设计 PCB 印制板时为 SJA1000、RTL8019AS、PDIUSBD12 等关键部件配置了适当的去耦电容。

(1).在上述芯片的输入端上跨接了 10～100uf 的电解电容器。

(2).在每个集成电路芯片上布置了一个 0.01pf 的瓷片电容。

另外，对抗噪声能力弱、关断时电源变化大的器件，如 RAM 存储器，则在芯片的电源线和地线之间直接接入去耦电容。

3.4 系统硬件调试

系统硬件调试是系统设计过程中一个很关键的步骤，它直接决定系统能否正常工作，同时也是去除系统硬件错误的最佳阶段。本系统的调试工具主要有万用表和示波器，调试时以单元电路为模块，采用分块边焊接边调试的方法。

(1).首先对照原理图仔细检查印制电路板的连线是否正确，电源和地有没有短接。无误后先焊接 CPU 插槽内部的器件，可以避免后面频繁插拔 CPU 子板；

(2).焊接电源模块，用万用表测得 3 种输出电压分别为 1.79V、3.31V、5.01V，符合设计的要求和系统电源要求的精度；

(3).焊接好独立的 CPU 子板和板上 CPU 的晶振，插上子板给系统上电，用示波器观察晶振的脉冲信号频率为 65.8763MHz；

(4).焊接 MAX3232，测得芯片上 10 腿上输入电压为 3.29V 和 8 腿输出电压为 3.31V，符合要求；

(5).焊接两个 9 针串口接头，测量 2 和 3 腿的对地电压分别为 3.29V、3.3V；

(6).焊接上 SJA1000 芯片，看上电 LED 灯是否为点亮状态，测得 13 腿上输出电压 5.05V，说明 SJA1000 正常；

(7).焊接上 RTL8019AS 模块，检查各 LED 状态灯，其中上电 LED 为点亮状态，连接上 RJ45，检查 RJ45 上的输入和输出线的对地电压为 5V；

(8).焊接 PDIUSBD12，与 USB-Socket 连接，测得 2 和 3 腿的对地电压分别为 5.05V、5V，符合要求；

(9).完成系统其他部分。

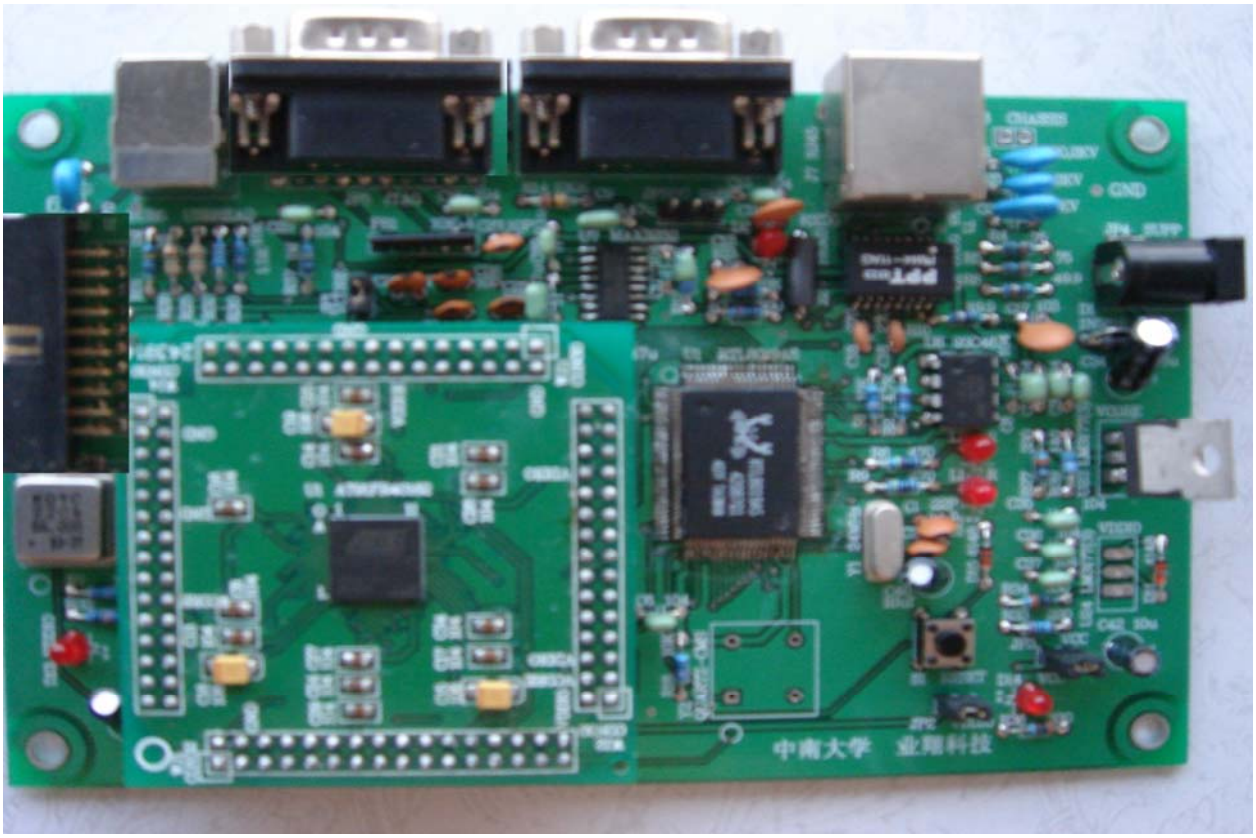

图 3-15 系统硬件实物图

图 3-15 是系统的硬件平台焊接好后的实物图，至此，系统的硬件平台初步搭好，后续的工作就是在该平台上进行系统软件代码的编写。

3.5 小结

本章特别分析了系统关键通信接口电路：CAN 接口电路和以太网接口电路的实现过程，并给出了其余各功能模块的详细电路，实现了系统的硬件平台。同时针对存储器系统的扩展进行了详细的讨论，指出了硬件平台 PCB 制作和具体调试过程，最后给出了系统的硬件实物图。

第四章 系统的软件开发

系统硬件设计和测试完成后，工作重心就转移到系统软件的开发上面来了。在这个过程中，硬件平台相对固定，主要任务是进行系统应用软件的 DEBUG 和开发了，主要工作按设计流程一般可分为三部分：首先要进行系统启动代码的设计，然后对嵌入式操作系统进行移植，最后的工作是进行操作系统的扩充和完善以及用户应用程序的开发。在这一章中主要介绍前两部分的工作。

4.1 系统启动代码设计

基于 ARM 的嵌入式系统是复杂的片上系统(SoC)[18]。在这种复杂系统中大多数硬件模块都是可进行配置的，需要由软件来设置其需要的工作状态。因此在用户的应用程序启动之前，需要有专门的一段启动代码来完成对系统的初始化，并引导系统进入 C 程序，类似于 PC 机的 BIOS 功能。由于这类代码直接面对处理器内核和硬件控制器进行编程，一般都用汇编语言实现。

由于系统上要带嵌入式操作系统，因此启动代码与操作系统的关系应该小心处理。μC/OS-II 能和应用代码一起编译运行[21]，在实现时可以将 μC/OS-II 当作应用代码和启动代码一起编译，但是，必须正确定义系统上电后最先执行的代码入口即起始地址，在系统正常运行后 μC/OS-II 才进行任务调度、配置和管理系统的资源。因此启动代码是在系统启动的开始阶段运行，完成系统硬件的初始化后再跳入到 μC/OS-II 中。

本系统中通过 AT91FR40162 片内集成的 2MB Flash 来启动，因此在启动时要注意将 NCS0 在外部与 NCS7 短接，同时将 BMS 从外部拉低，以使处理器复位后可以完成正确的 16 位的取指操作。

启动代码一般随实际目标系统和开发系统有所区别，通常包括以下部分：

- 定义入口点
- 设置中断/异常向量
- 初始化存储系统(包括地址重映射)
- 初始化堆栈指针寄存器
- 初始化中断中用到的变量
- 开中断
- 必要时改变处理器的模式
- 必要时改变处理器的状态
- 初始化 C 程序用到的存储区
- 进入 C 程序

1．定义入口点

启动程序首先必须定义入口指针，而且整个应用程序只有一个入口指针。

```
AREA      reset, CODE, READONLY
EXPORT    _main
_main                          ；定义_main
ENTRY                          ；定义入口点
```

在编译的时候，编译器需要知道整个程序的入口地址，所以在编译前要设置好相关的编译选项，如程序入口所在的目标文件。

2．设置中断/异常向量

对于 AT91FR40162，系统启动或复位后，重映射之前，ARM 中断向量必须放置在从 0 地址开始的连续 8×4 字节的空间内。当有中断或者异常发生的时候，PC 就会跳转到 0～20H 之间相应的地址去取程序代码执行，这样可以避免复杂的寻址过程，加快 CPU 的处理速度[29]。实际上在本系统中上电后是要被连接到 0x100 0000(通过片内的 Flash 启动)的地方；在 REMAP 之后，被复制到位于地址 0 的 SRAM 中。下面程序是中断/异常向量的源码，此处只是给出了一个具体跳转的目标处理函数。

```
Reset
                b           Reset     ;复位
UndefHandler
                b           UndefHandler    ;未定义
SWIHandler
                b           SWIHandler      ;软件中断
PrefetchAbortHandler
                b           PrefetchAbortHandler    ;预取指中止
DataAbortHandler
                b           DataAbortHandler        ;数据中止
IRQHandler
                b           IRQHandler          ;通用中断中止
FIQHandler
                b           FIQHandler          ;快速中断
```

在 REMAP 之后的中断/异常向量，采取的是一种相对寻址的方式。中断向量表将在执行 REMAP 命令之前被复制到片内的 RAM 区 0x30 0000，REMAP 之后就是以 0 开始的区域。DCD 标示当前地址存放的是一个数据段。

```
VectorTable
        ldr         pc, [pc, #&18]          ；软件复位
        ldr         pc, [pc, #&18]          ；未定义中止
        ldr         pc, [pc, #&18]          ；软件中断
```

```
ldr     pc, [pc, #&18]      ; 预取指中止
ldr     pc, [pc, #&18]      ; 数据中止
nop                         ; Reserved
ldr     pc, [pc,#-0xF20]    ; IRQ : 读 AIC
ldr     pc, [pc,#-0xF20]    ; FIQ : 读 AIC

DCD     SoftReset
DCD     UndefHandler
DCD     SWIHandler
DCD     PrefetchAbortHandler
DCD     DataAbortHandler
```

3．初始化存储系统

系统在上电之后，紧接着就从地址 0 开始取得第一条指令的代码执行，所以地址 0 必须在上电的时候就存在可执行的正确代码，也就是地址 0 的地方应该是 ROM 区，至少在上电之后的一小段时间内应该是 ROM 区。对于 ARM 而言，中断和异常的入口地址是 0～20H。但是 ROM 的访问速度相对较慢，每次中断发生后，都要从读取 ROM 上面的向量表开始，影响了中断速度。因此，ARM 处理器提供一种非常灵活的地址重映射方法，把 0 地址重新指向到 RAM 中去，即启动地址重映射，具体见 4.3 节。

为保证重映射之后提供正确的中断入口地址，在重映射之前就必须把中断和异常向量表拷贝到内部 RAM 中。其程序实现如下：

```
mov r8,# RAM_BASE_BOOT
                    //RAM_BASE_BOOT 是重映射前内部 RAM 区地址
add   r9, pc,#-(8+. －VectorTable)   //VectorTale 是异常向量表入口
ldmia    r9!, {r0-r7}          //读 8 个异常向量
stmia    r8!, {r0-r7}          //保存 8 个异常向量到 RAM 区
ldmia    r9!, {r0-r4}          //读 5 个异常处理程序绝对地址
stmia    r8!, {r0-r4}       //保存 5 个异常处理程序绝对地址到 RAM 区
```

AT91FR40162 提供了一个 REMAP 命令来实现重映射。上电复位的时候，0 地址的地方映射到 NCS0 所接的外围器件(此处用的片内 Flash 也可以权且当作是)，内部 RAM 映射到 0x300000 处。当把 EBI 的重映射控制寄存器的重映射命令位(EBI_RCR 的 RCB 位)置 1 之后，内部 RAM 就映射到 0 的位置，NCS0 所接的外围器件映射到 0x01000000 处。如果这个时候中断和异常向量表已经被复制了，则上电复位的一段时间内，CPU 可以在 0 地址取到可执行的有效的指令代码，然后在应用程序执行时能够以更快的速度响应中断和异常，并且可以通过指令的执行动态地进行改动。具体程序实现为，

```
Sub   r10,pc,#(8+.-InitTableEBI)
                    // InitTableEBI 是片选寄存器和重映射寄存器入口
```

```
ldr     r12, PtInitRemap          //PtInitRemap 是重映射后跳转地址
ldmia   r10!, {r0-r9,r11}         //加载片选寄存器和重映射寄存器
stmia   r11!, {r0-r9}             //进行重映射并进行寄存器配置
mov     pc, r12                   //跳转并停止流水线
DCD     InitRemap                 //重映射之后跳转目的地址
InitRemap
```

重映射之后地址分配如图 4-1 所示。

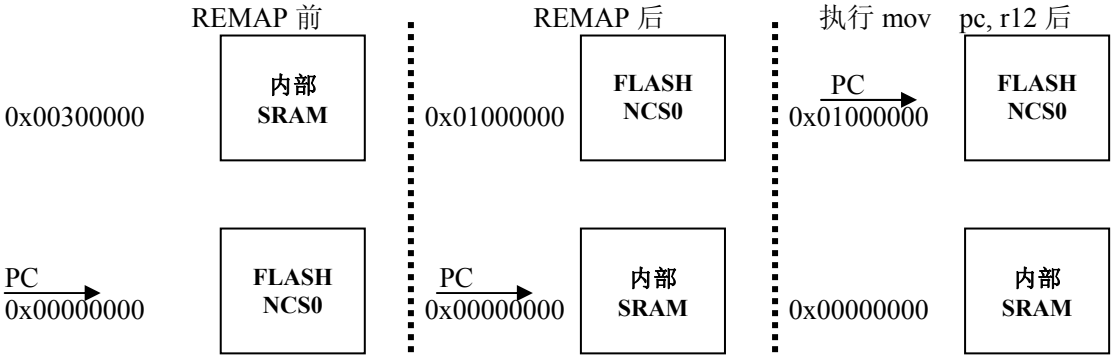

图 4-1 重映射后的地址分配

4．初始化堆栈、中断寄存器

ARM 处理器具有 7 种处理器模式：用户模式、FIQ 模式、IRQ 模式、管理模式、中止模式、未定义模式和系统模式。除用户模式外的其他模式称为特权模式。特权模式是为了响应中断或异常，或访问系统保护的资源。

每一种模式都有独立的堆栈指针寄存器(SP)，因此对使用到的模式都要定义堆栈地址给堆栈指针寄存器 SP。改变状态寄存器(CPSR)内的状态位，使处理器切换到不同模式，然后给 SP 赋值，就可以实现堆栈的初始化。堆栈的大小根据需要而定。例如 FIQ 模式堆栈的初始化程序如下：

```
ldr    r0, =TOP_EXCEPTION_STACK          //设置栈顶
msr CPSR_c,#ARM_MODE_FIQ|I_BIT|F_BIT     //切换到 FIQ
mov r13,r0                               //进行堆栈初始化
sub    r0,r0,#FIQ_STACK_SIZE             //得到栈底地址
```

5．初始化中断寄存器

AT91FR40162 内部有一个 8 优先级、可单独屏蔽的向量中断控制器，称为先进中断控制器 AIC(Advanced Interrupt Controller)。AIC 采用硬件中断向量化。硬件中断向量化使中断处理入口所需的指令减少到只有一条(详见 4.2 节)。当产生中断时，中断处理例程必须尽快读取中断向量寄存器 AIC_IRQ 的值。中断向量寄存器 AIC_IRQ 里存放当前中断所对应的中断源向量寄存器 AIC_SVR 的值。每个中断源都有自己的中断源向量寄存器 AIC_SVR。所以，在系统初始化时，必须将各个中断处理入口地址保存到对应的中断源向量寄存器 AIC_SVR 之中。

6．初始化存储区并进入 C 主程序

存储区的初始化主要是将 ROM 里带初始值的变量拷贝到 RAM 中，接着就对定义了变量名的全局变量进行零初始化操作。当所有的系统初始化工作完成以后，用 r0 传递的地址就可以进入主应用程序。

```
IMPORT      main
  ldr          r0, =main
  mov         lr,     pc
  bx           r0
END
  b     END
  END
```

4.2 AT91FR40162 的中断处理

AT91FR40162 内部集成了一个先进中断控制器 AIC，AIC 采用硬件中断向量化[18、29]，将到达中断处理入口所需的指令减少到只有一条，这样可以将中断处理的开销时间大大的降低。ARM 核的中断有 RESET、UNDEFINED、INSTRUTION、PREFETCH ABORT、DATA ABORT、SWI、IRQ、FIQ 七种，用户常用的是 IRQ 中断，像 US、TC 和 PIO 等都属于 IRQ 中断。IRQ 中断位于中断向量表第 7 位，硬件中断向量化位于地址 0x00000018 发生时，当 IRQ 发生时，PC 指针首先跳到 0x00000018 处开始执行。

LDR PC,[PC ,#-&F20]

从地址 0 算起即为(7－1)×4(ARM 指令为 32 位)＝24＝0x18。执行此指令时，由 ARM 的流水线结构可知，现时的 PC 为 0x00000018＋2×4＝0x00000020。因此指令执行后 PC 被赋予了地址 0x20－0xF20=0xFFFFF100 的内容，即为 IRQ 向量寄存器的内容，程序就跳转到相应的中断例程。

标准的 IRQ 中断过程：在 IRQ 脚给个上升沿后，当没有打开该中断的使能寄存器，没有任何反应。如果打开了使能寄存器，但在对应的屏蔽寄存器中屏蔽了该中断，也没有任何反应。在上述两个寄存器都设置完成后，中断发生，CPU 保存当前程序运行环境，跳到中断入口 0x18 地址处。如果在 0x18 处没有设置中断向量，程序就会跑飞。前面已提到中断向量一般是个跳转指令，将跳到正式的中断处理函数中，用户在处理函数中可以进行关中断，清中断等操作，然后退出中断返回现场。

4.3 ARM 重映射

1. 重映射过程分析

对于 ARM 而言，中断和异常的入口地址是 0～20H。每当有中断或者异常

发生时，PC 就会跳到 0～20H 之间的相应地址去取程序代码执行。如果中断向量处于 ROM 区，那中断处理程序只能固定在其中，不能被动态改变，由于 ROM 通常比较慢，CPU 需要加入额外的等待周期，而中断过程应该尽可能快些，所以中断向量位于 RAM 上会好一些。ARM CPU 提供了一个 REMAP 命令来解决这个问题。上电复位时，0x00000000 地址的地方映射到 NCS0 所接的外围器件，此处是片内的 Flash；片内的 RAM 映射到 0x00300000 处。如下图 4-2 所示。

当把 EBI 的重映射控制寄存器的重映射命令位(EBI_RCR 的 RCB 位)置 1 之后，内部的 RAM 就映射到 0x00 的位置。如果这时候中断和异常向量表已经被复制了，那么就能够实现所需的情况，即上电复位的一段时间内，CPU 可以在 0x00 地址取到可执行的有效的指令代码。然后在应用程序执行时，能够以更快速度响应中断和异常，并且可以通过指令的执行动态的进行改动。

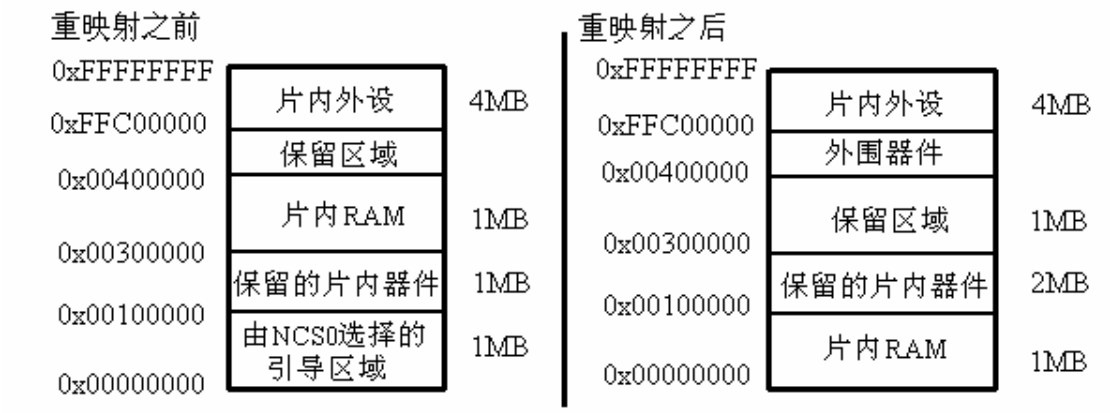

图 4-2 AT91 系列处理器的地址重映射图

2. 重映射实现方法

ARM 重映射有两种方法。第一种是 REMAP 之前先将 ROM 中的内容拷贝到 RAM，这样 REMAP 命令仍然能取到预期的下一条指令，但是这样做不是很灵活，一般不这么做。另一种方法是利用 ARM7 的三级流水线，在 REMAP 后放一条绝对跳转指令，把 PC 指针指到预定的位置开始执行。由于三级流水线的作用，REMAP 执行之前，这条跳转指令已经被取指并译码了，可以保证程序正常进行。注意这时 REMAP 必须用 ldmia 和 stmia 的方式进行，因为这样只需两条指令就完成 REMAP，正好第三条是跳转指令。

4.4 μC/OS-II 实时内核的移植准备

所谓移植 PORT，就是通过修改一个实时内核或 RTOS 中与处理器密切相关的部分，包括 CPU 字长和数据类型定义，系统中的中断如何实现、任务如何切换、系统定时实现等，使之能在其它体系结构的微处理器或微控制器上运行。因

为 μC/OS-II 的大部分代码是用 C 语言编写的，可移植性强。在实现时，仍需要采用 C 语言和汇编语言写一些与处理器硬件相关的代码，因为在 μC/OS-II 中对于处理器寄存器的读/写操作只能通过汇编语言来实现。

1. μC/OS-II 文件结构的移植处理

μC/OS-II 也称作微处理器操作系统，为了能应用于大多数微处理器，在内核设计时就充分考虑了以后移植的方便，所以它的文件结构非常简洁。我们在移植时对 μC/OS-II 内核的文件结构做如下一些处理和配置。

可以在文件 uCOS-ii.c 中定义了一些控制宏，并且包含所有与 CPU 无关代码，这样可以使得移植过程简洁，而且在移植过程中可以不去管内核里面的东西，只要保持它们的外部函数类型匹配即可。它的内容一般有：

#include "os_core.c"
#include "os_mbox.c"
#include "os_mem.c"
#include "os_q.c"
#include "os_sem.c"
#include "os_task.c"
#include "os_time.c"
#include "os_mutex.c"

OS_CPU_A.S 这个文件是移置的重点，用汇编实现的，主要是和处理器相关的一些硬件操作，包括开关中断、中断处理和一些硬件配置。

OS_CPU_C.C 这个文件包含一个和 CPU 结构相关的任务堆栈初始化函数，以及用户可以利用的一系列钩子 hook 函数，用来处理特殊硬件扩展、调试等。

移植后的 μC/OS-II 整体结构[21]如图 4-3 所示。

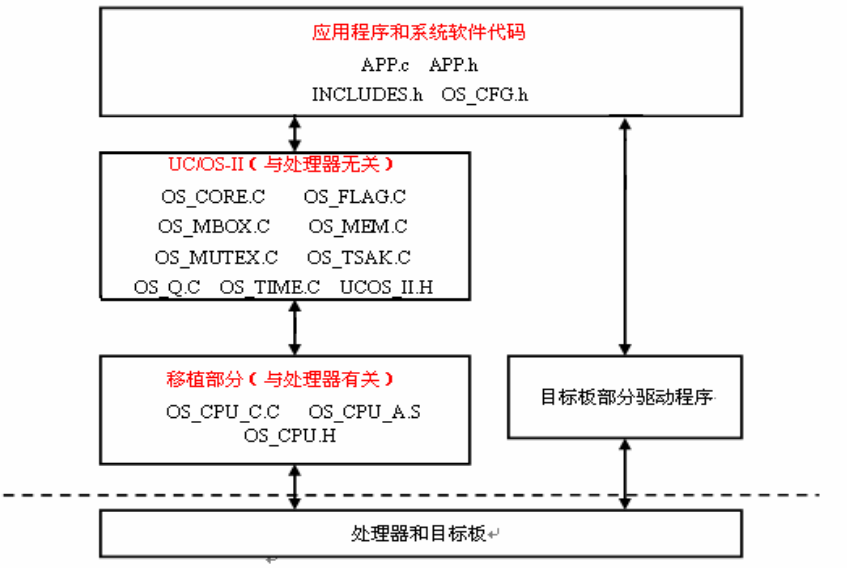

图 4-3　μC/OS_II 的文件体系结构图

OS_CPU.H 一个需要根据 CPU 的指令字长和硬件更改的头文件。

OS_CFG.H 是 μC/OS-II 的配置文件，可以通过它裁剪操作系统的大小和其它与操作系统功能模块有关的参数设定。

因此移植的准备工作主要是了解处理器的结构，再就是仔细理解 μC/OS-II 中和任务切换相关的几个函数、宏，以及它们之间的调用关系。

2. 处理器需满足的移植条件

要移植 μC/OS-II，微处理器还必须满足以下要求[11、29]：

① 处理器的 C 编译器能产生可重入代码。

② 用 C 语言就可以打开和关闭中断。

③ 处理器支持中断，并且能产生定时中断。

④ 处理器支持能够容纳一定量数据的硬件堆栈。

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。

4.5 μC/OS-II 在 51 单片机上的移植

考虑到当前 51 系列单片机应用的广泛性，开发简单，在低端产品开发中一般很自然地应用 51 单片机，在这里顺便也介绍 μC/OS-II 在 51 单片机上的移植，并通过与 ARM7 上的移植进行类比，介绍彼此间的联系和差异。

笔者在移植中采用的单片机是 SST89C54，外加 64KB 的 RAM。移植编译环境采用 Keil C51。考虑到单片机的运行速度较低，编译模式采用小模式因为 μC/OS-II 在小模式下的运行速度比大模式快很多。SP 初始化、内存清 0 等操作在 STARTUP.A51 文件中实现；Keil 的变量，子程序等的分配信息可以在.M51 文件里查找。

4.5.1.51 上的移植过程

移植过程要结合 Keil C51 的特点，以提高内核移植后代码的执行效率。因此移植的过程主要分为以下几个步骤[48]。

1. 首先将所有的外部变量的存储类型改为 xdata 类型。

由于程序中存在大量的外部变量，其中包括大型数组，在小模式下无法编译通过，所以要将所有的外部变量都申明为 xdata 类型。

2. 尽量使用指定存储类型的指针(memory-specific pointer)而不使用一般指针(generic pointer)

程序中用到的一般指针包括两类：指向缓冲区的数据指针和指向函数的函数

指针。一般情况下，使用指定存储类型的指针比使用一般指针效率要高。移植时要将程序中的 generic pointer 修改为 memory-specific pointer 类型。

(1) 缓冲区一般都定义为外部变量，而前面已经将外部变量都申明为 xdata 类型，所以对于这种情况，只要指针改为指向 xdata 数据类型的指针就可以了。

(2) 将指向函数的指针改成指向代码区，所以要将这种指针类型改为指向 code 数据类型的指针。

3．任务堆栈结构设计

任务堆栈结构设计是移植的关键部分。任务堆栈用于保存任务切换时的 context。因此必须解决堆栈起点、堆栈空间大小等的设定。

由于程序在小模式下编译，所以仿真栈在内部 RAM 中，设定仿真栈从 0xFF 地址开始向下生长，可重入函数的局部变量和函数参数将放在仿真堆栈中。因此在任务切换时需要保存的 context 有：仿真栈指针?C_IBP、仿真栈内容、硬件栈大小(用于计算 SP 的值)、硬件栈内容(包括压入硬件栈的寄存器)。

Keil C51 程序在进入中断函数以后有时需将重要寄存器压入堆栈，这就是上面所说的“压入硬件栈的寄存器”。任务堆栈结构中的寄存器的排列顺序必须和 Keil C51 程序进入中断以后的寄存器压栈顺序相同，查看中断函数的反汇编程序可以了解压栈的顺序。

TCB 结构体中 OSTCBStkPtr 总是指向用户堆栈最低地址，该地址空间内存放用户堆栈长度，在其上的空间存放系统堆栈映像，即：用户堆栈空间大小=系统堆栈空间大小+1。

SP 总是先加 1 再存数据，因此 SP 初始时指向系统堆栈起始地址(OSStack)减 1 处(OSStkStart)，很明显系统堆栈存储空间大小=SP-OSStkStart。

任务切换时，先保存当前任务堆栈内容。方法是：用 SP-OSStkStart 得出保存字节数将其写入用户堆栈最低地址内，以用户堆栈最低地址为起址，以 OSStkStart 为系统堆栈起址，由系统堆栈向用户堆栈拷贝，数据循环 SP-OSStkStart 次，每次拷贝前先将各自栈指针增 1。

恢复最高优先级任务系统堆栈方法是：获得最高优先级任务用户堆栈最低地址，从中取出“长度”，以最高优先级任务用户堆栈最低地址为起址，以 OSStkStar 为系统堆栈起址，由用户栈向系统栈拷贝，数据循环长度数值指示的次数，每次拷贝前先将各自栈指针增 1。

最后的任务栈结构如图 4-4。用户堆栈初始化时从下向上依次保存用户堆栈长度(15)、PCL、PCH、PSW、ACC、B、DPL、DPH、AR0、AR1、AR2、AR3、AR4、AR5、AR6、AR7，不保存 SP，任务切换时根据用户堆栈长度计算得出。

OSTaskStkInit 函数总是返回用户栈最低地址。

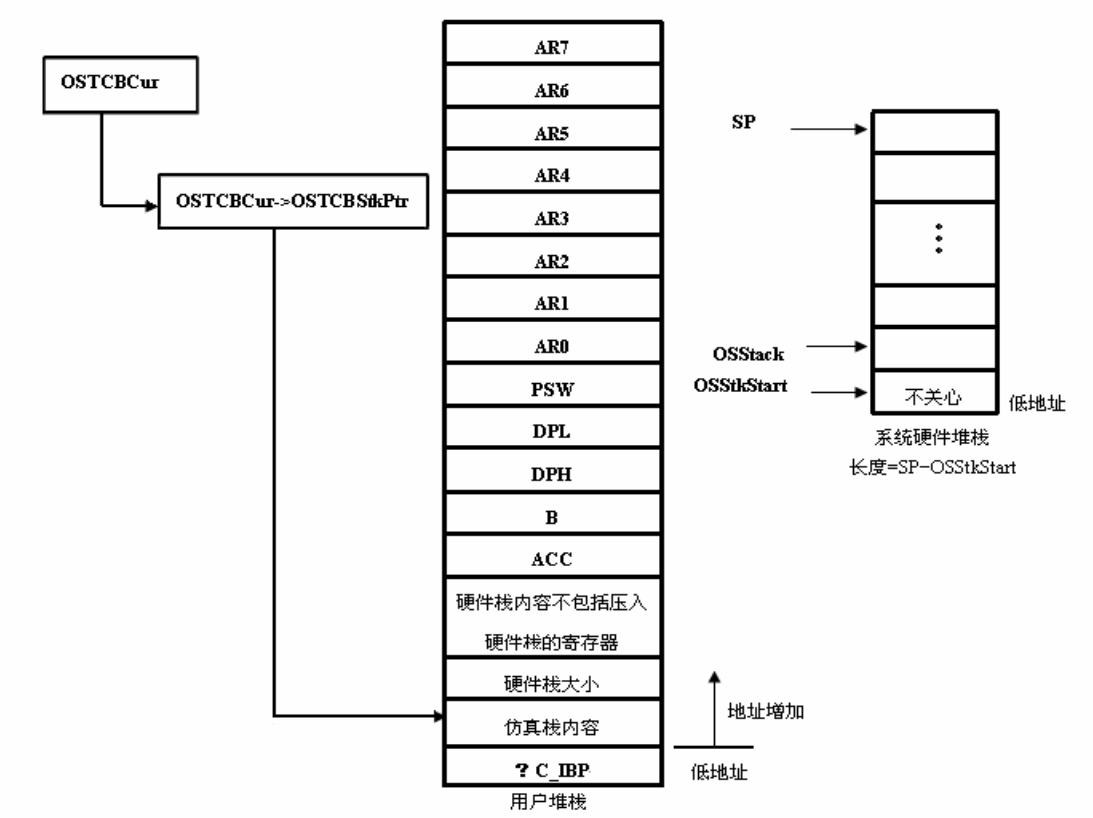

图 4-4 堆栈结构图

4.5.2 移植注意要点

在 51 上的移植过程中需要注意的要点主要有：

①因为 Keil 缺省情况下编译的代码不可重入，而多任务系统要求并发操作导致重入，所以要在每个 C 函数及其声明后标注 reentrant 关键字；

②另外 pdata、data 在 uC/OS-II 中用做一些函数的形参，但它同时又是 Keil 的关键字会导致编译错误，要把 pdata 改成 ppdata；data 改成 ddata。

③OSTCBCur、OSTCBHighRdy、OSRunning、OSPrioCur、OSPrioHighRdy 这几个变量在汇编程序中用到了，为了使用 Ri 访问而不用 DPTR，应该用 Keil 扩展关键字 IDATA 将它们定义在内部 RAM 中；

④分配堆栈空间时，只须关心大小，堆栈起点由 keil 决定，通过标号可以获得 keil 分配的 SP 起点，切勿自己分配堆栈起点，只要用 DS 通知 Keil 预留堆栈空间即可；

⑤?STACK 段名与 STARTUP.A51 中的段名相同，这意味着 Keil 在 LINK 时将把两个同名段拼在一起。这里预留了 40H 个字节，STARTUP.A51 预留了 1 个字节，LINK 完成后堆栈段总长为 41H。查看.m51 可知 Keil 将堆栈起点定在 21H，长度 41H 处于内部 RAM 中。

移植后，考虑到 51 单片机的工作原理大致相似，该系统具有通用性。

4.6 μC/OS-II 在 AT91FR40162 上的移植

相对 51 单片机的移植，考虑到 ARM7 产品的多样性，内部集成的外围差异较大，在移植中要尽量做到通用性，主要表现在 μC/OS-II 的版本选择、编译器和编译工具等的选择。同时在移植前对 μC/OS-II 的文件结构处理时也该尽量是代码保持相对的独立性[49]。

在这里，可把整个移植过程分为两个部分：与应用相关的 μC/OS-II 配置(OS_CFG.H，INCLUDES.H)；与处理器相关的 μC/OS-II 移植(OS_CPU.H，OS_CPU_A.ASM，OS_CPU_C.C)。

移植中的难点是编写与硬件有关的汇编代码，所以必须对 ARM 的硬件结构有所了解。因此在移植前先对 ARM7 和移植有关的硬件结构知识进行简单介绍。

4.6.1 ARM7 中和移植有关的硬件结构

ARM7TDMI 的处理器操作状态有 ARM 态和 THUMB 态[32]，两种状态通过 CPSR(当前程序状态寄存器)中的“T”位设置来标识，也通过置位“T”位来实现状态间的切换。在 ARM 态下执行 32 位标准 ARM 指令集指令；在 THUMB 态下处理器执行扩展的 16 位指令集指令，在该状态下能访问的寄存器没有 ARM 态下多，设计 THUMB 态的主要目的是提高指令密度。系统正常复位后，处理器一般处于 ARM 态，本系统是工作在 ARM 态，所以在移植时可以不管 THUMB。

处理器在 ARM 状态下有 7 种操作模式，如图 4-5 是各个操作模式的寄存器状态。

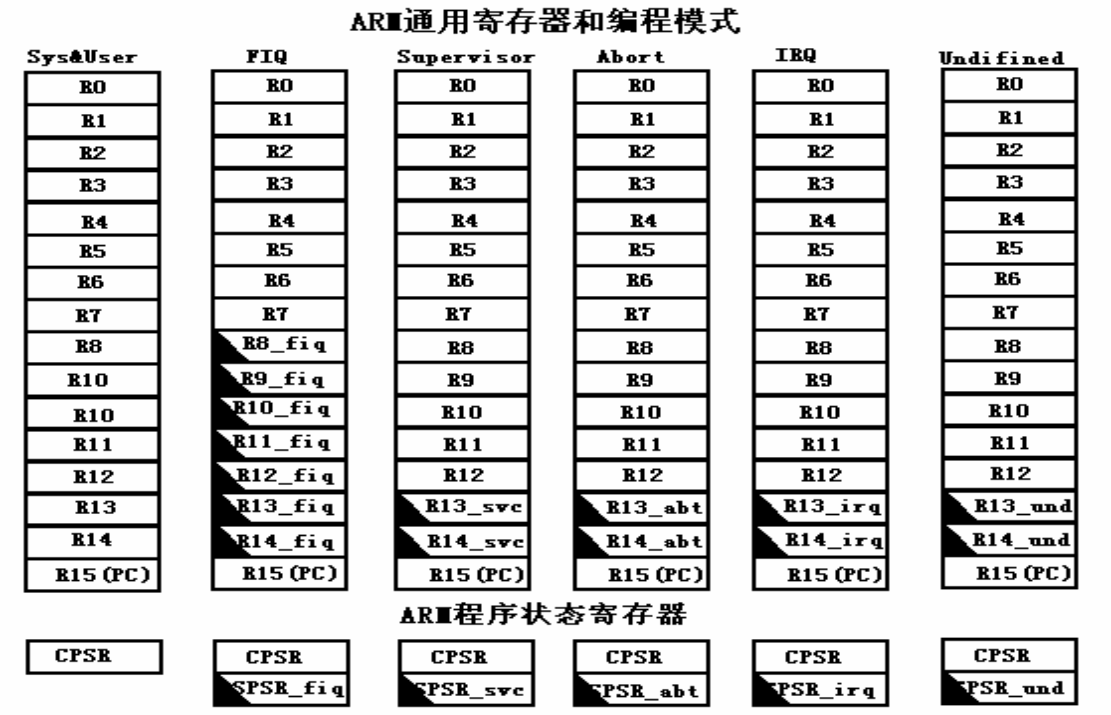

图 4-5 ARM 状态下的寄存器组织图

每一列是一种操作模式下访问的寄存器，不带黑三角的所有模式共享一个物理寄存器，每种模式都能访问到相同的数据，带三角的寄存器每种模式都有自己的一套独用物理寄存器，在别的模式下访问不到。R15 用作程序计数器 PC，R14 用作链接寄存器 LR，这两个使硬件有特殊功能。一般 R13 用作堆栈指针，其实别的寄存器也行，只不过 R13 每个模式下都各有一套。

SPSR 程序状态寄存器是用来保存进入当前处理器模式的上一个模式下的 CPSR，利于异常或中断的返回。

4.6.2 AT91FR40162 上的移植过程

本文选用的μC/OS-II内核是v2.76，是当前的最新版本。编译工具选用ADS1.2，见2.3.1节，是SDT的兼容编译器，用该编译器能很好的优化代码，一般同样的代码，比别的编译器能优化20%～30%[13]。

在移植过程中实际上可以把μC/OS-II简单地看作是一个多任务的调度器，只是在这个任务调度器之上完善并添加了和多任务操作系统相关的一些系统服务，如信号量、邮箱等。μC/OS-II的90%的代码都是用C语言写的，因此只要有相应的C语言编译器，基本上就可以直接移植到特定处理器上，这也是μC/OS-II具有良好的可移植性的原因。移植工作的绝大部分都集中在多任务切换的实现上，因为这部分代码主要是用来保存和恢复处理器现场(相关寄存器)，因此不能用C语言实现，必须使用特定的处理器汇编语言完成[49~50]。

μC/OS-II的全部源代码量大约是6000－7000行，一共有15个文件。将μC/OS-II移植到ARM处理器上，主要工作是修改三个和ARM体系结构相关的文件。下面分别介绍这三个文件的移植。

1. OS_CPU.H文件

◆ 数据类型定义

这部分的修改是和所选编译器相关的，不同的编译器会使用不同的字节长度来表示同一数据类型，比如int，在x86平台上，编译为2 bytes，如果用GNU的gcc编译器，则为4 bytes。在ADS1.2 C编译器下相关的数据类型的定义如下：

```
typedef   unsigned char   BOOLEAN;
typedef   unsigned char   INT8U;          /* Unsigned   8 bit quantity */
typedef   signed    char   INT8S;          /* Signed   8 bit quantity*/
typedef   unsigned short   INT16U;         /* Unsigned 16 bit quantity*/
typedef   signed   short   INT16S;         /* Signed 16 bit quantity*/
typedef   unsigned int     INT32U;        /* Unsigned 32 bit quantity */
typedef   signed    int     INT32S;        /* Signed     32 bit quantity */
typedef   float            FP32;          /* Single precision floating point */
typedef   double           FP64;          /* Double precision floating point */
```

```
typedef    unsigned int    OS_CPU_SR;      /* Define size of CPU status register
                                              (PSR = 32 bits) */
```

◆ 堆栈单位

因为处理器现场的寄存器在任务切换时都将会保存在当前运行任务的堆栈中，所以OS_STK数据类型应该是和处理器的寄存器长度一致的。

```
typedef    unsigned int      OS_STK;    /* Each stack entry is 32-bit wide*/
```

◆ 堆栈增长方向

堆栈由高地址向低地址增长，这个也是和编译器有关的，当进行函数调用时，入口参数和返回地址一般都会保存在当前任务的堆栈中，编译器的编译选项和由此生成的堆栈指令就会决定堆栈的增长方向。此处定义为

```
#define OS_STK_GROWTH    1
```

◆ 临界代码段的宏定义

临界代码段是指μC/OS-II能进入中断处理，处理完后能重新开中断并返回原来的现场。在这过程中，为了保护临界代码段的代码免受多任务切换和中段处理函数的破坏，同时方便移植，在μC/OS-II中定义了两个宏*OS_ENTER_ CRITICAL( )*和*OS_EXIT_ CRITICAL( )*，以及任务切换的宏定义。

```
#define OS_ENTER_ CRITICAL( ) { ARMEnableInt( );cpu_sr=OS_CPU_SR_SAVE ( );}
#define OS_EXIT_ CRITICAL( ) {ARMDisableInt( ); S_CPU_SR_Restore(cpu_sr);}
#define OS_TASK_SW( ) OSCtxSW( )
```

*OS_CPU_SR _SAVE ( )*用来保存处理器的CPSR，*OS_CPU_SR_Restore( )*用来返回CPSR值，使中断返回原来的中断点。上两个函数以及*ARMEnableInt( )*和*ARMDisableInt( )*都用汇编语言实现，在OS_CPU_A.S文件中定义，

2. OS_CPU_C.C文件

◆ 任务堆栈初始化函数*OSTaskStkInit( )*

在ARM体系结构下，任务堆栈空间由高至低依次将保存着pc、lr、r12、r11、r10、…r1、r0、CPSR、SPSR。根据*OS_STK_GROWTH*的值“1”，堆栈增长方向由高至低，*OSTaskStkInit( )*函数初始化任务堆栈，也就是在堆栈增长方向上定义每个需要保存的寄存器位置。

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *p_arg, OS_STK *ptos,
                       INT16U opt)
{
    OS_STK *stk;
    opt          =    opt;                    /* 'opt' is not used, prevent warning*/
    stk          =    ptos;                   /* Load stack pointer */
    *(stk)       = (OS_STK)task; /* Entry Point*/
    *(--stk) = (INT32U)0x14141414;            /* R14 (LR)*/
    *(--stk) = (INT32U)0x12121212;            /* R12 */
```

```
    *(--stk) = (INT32U)0x11111111;          /* R11 */
    *(--stk) = (INT32U)0x10101010;          /* R10 */
    *(--stk) = (INT32U)0x09090909;          /* R9 */
    *(--stk) = (INT32U)0x08080808;          /* R8 */
    *(--stk) = (INT32U)0x07070707;          /* R7 */
    *(--stk) = (INT32U)0x06060606;          /* R6 */
    *(--stk) = (INT32U)0x05050505;          /* R5 */
    *(--stk) = (INT32U)0x04040404;          /* R4   */
    *(--stk) = (INT32U)0x03030303;          /* R3 */
    *(--stk) = (INT32U)0x02020202;          /* R2 */
    *(--stk) = (INT32U)0x01010101;          /* R1 */
    *(--stk) = (INT32U)p_arg;               /* R0 : argument */
    *(--stk) = (INT32U)ARM_MODE_SYS;    /* CPSR(Enable both IRQ
                                            and FIQ interrupts)*/
    return (stk);
}
```

注意这里需要说明三点[50]：

①此处通过向各寄存器写入不同的值在实际过程中没什么实际意义，只是为了一些 μC/OS-II 支持的第三方编译器在调试过程中能识别相应的寄存器和方便一些调试，这些值在运行时都会被改写。

②当前任务堆栈初始化完成后，*OSTaskStkInit* 返回新的堆栈指针 stk，在 *OSTaskCreate( )*执行时将会调用 *OSTaskStkInit* 的初始化过程，然后通过 *OSTCBInit( )*函数调用将返回的 sp 指针保存到该任务的 TCB 块中。

③初始状态的堆栈其实是模拟了一次中断发生后的堆栈结构，因为任务被创建后并不是直接就能够获得执行的，而是通过 OSSched( )函数进行调度分配，满足执行条件后才能获得执行的。为了使这个调度简单一致，就预先将该任务的 pc 指针和返回地址 lr 都指向函数入口，以便被调度时从堆栈中恢复刚开始运行时的处理器现场。

◆ 系统hook函数

另外，在这个文件里面还需要实现5个操作系统规定的hook函数，如下：

OSSTaskCreateHook( )；OSTaskDelHook( )；OSTaskSwHook( )

OSTaskStatHook( )； OSTimeTickHook( )

如果没有特殊需求，则只需要简单地将它们都实现为空函数就可以了[21]。

3. OS_CPU_A.S文件

这个函数是跟处理器有关的硬件操作函数，是移植过程中的难点和重点。所有的函数都是用汇编语言实现。因此要求对ARM的指令系统和μC/OS-II的功能模块有深入的理解。

◆ 启动最高优先级任务函数*OSStartHighRdy( )*

在*OSStart( )*运行后，μC/OS-II启动多任务之后，*OSStartHighRdy( )*负责从最

高优先级任务的TCB控制块中获得最高优先级任务的堆栈指针sp，通过sp依次将cpu现场恢复，这时系统就将控制权交给用户创建的该任务进程，直到该任务被阻塞或者被其他更高优先级的任务抢占cpu。该函数仅仅在多任务启动时被执行一次，用来启动第一个，也就是最高优先级的任务执行，之后多任务的调度和切换就是由下面的函数来实现。

◆ 任务级的任务切换函数*OSCtxSw( )*

任务级的任务切换是在非异常模式下进行的，与中断级别的任务切换任务不同，它是在任务因为被阻塞而主动请求cpu调度时被执行。它的工作是先将当前任务的cpu现场保存到该任务堆栈中，然后获得最高优先级任务的堆栈指针，从该堆栈中恢复此任务的cpu现场，使之继续执行。这样就完成了一次任务切换。

◆ 中断级的任务切换*OSIntCtxSw( )*

中断级的任务切换是在时钟中断ISR(中断服务例程)中发现有高优先级任务等待的时钟信号到来时执行，它的工作是在中断退出后并不返回被中断任务，而是直接调度就绪的高优先级任务执行。这样做的目的主要是能够尽快地让高优先级的任务得到响应，保证系统的实时性能。与任务级的切换不同之处在进入中断时已经保存过了被中断任务的cpu现场，在这里就不用再进行类似的操作，只需要对堆栈指针做相应的调整。

◆ 时钟中断处理函数*OSTickISR( )*

时钟中断处理函数的主要任务是负责处理时钟中断，调用在μC/OS-II中实现的OSTimeTick函数，如果有等待时钟信号的高优先级任务，则需要在中断级别上调度其执行。两个相关的函数OSIntEnter( )和OSIntExit( )都需要在ISR中执行。此系统中采用AT91FR40162的定时器0作为μC/OS-II的系统时钟，频率为200Hz，即每5ms产生一次定时器0中断。

◆ 进入和退出中断函数*ARMEnableInt( )和ARMDisableInt( )*

*ARMEnableInt( )*和*ARMDisableInt( )*分别是进入临界区和退出临界区的宏指令，主要用于在进入临界区之前关闭中断，在退出临界区的时候恢复原来的中断状态。它的实现比较简单，可以采用方法1直接开关中断来实现，也可以采用方法2通过保存关闭/恢复中断屏蔽位来实现。此处采用方法2来实现[21]。

4.6.3 移植过程中的难点

移植 μC/OS-II 的绝大部分工作都集中在 OS_CPU_A.S 文件的移植，这个文件的实现集中体现了所要移植的目标处理器的体系结构和 μC/OS-II 的移植原理；在这个文件里，最困难的工作又集中体现在 OSIntCtxSw 和 OSTickISR 这两个函数的实现，因为这两个函数的实现是和移植的思路以及相关硬件定时器、中断寄

存器的设置有关。在实际的移植工作中，这两个地方也是比较容易出错的地方。

OSIntCtxSw 最重要的作用就是它完成了在中断 ISR 中直接进行任务的切换，以提高实时响应的速度。它发生的时机是在 ISR 执行到 OSIntExit( )时，如果发现有高优先级的任务因为等待的 time tick 到来获得了执行的条件，这样就可以马上被调度执行，而不用返回被中断的那个任务之后再进行任务切换，因为如果那样做的话就不够实时了。

实现 OSIntCtxSw 的有两种方法。

一种是通过调整 sp 堆栈指针的方法，根据所用的编译器对于函数嵌套的处理，通过精确计算出所需要调整的 sp 位置来使得进入中断时所作的保存现场的工作可以被重用。这种方法的好处是直接在函数嵌套内部发生任务切换，使得高优先级的任务能够最快的被调度执行。但是这个办法需要和具体的编译器以及编译参数的设置相关，需要较多技巧。此处的移植是基于这种方法。

另一种是设置需要切换标志位的方法，在 *OSIntCtxSw* 里面不发生切换，而是设置一个需要切换的标志，等函数嵌套从进入 *OSIntExit=>OS_ENTER_CRITICAL( ) =>OSIntCtxSw( )=>OS_EXIT_CRITICAL( )=>OSIntExit* 退出后，再根据切换标志位判断是否需要进行中断级的任务切换。这种方法的好处是不需要考虑编译器的因素，也不用做计算，但是从实时响应上不是最快，不过比较容易理解，实现起来也简单。

另外对中断级的任务切换，需要特别说明的一个问题是如何获得被中断任务的 lr_svc。因为进入中断态后，lr 变成了 lr_irq，原来任务的 lr_svc 无法在中断态下获得，这样要得到 lr_svc，就必须在中断 ISR 里面进行一次 cpu mode 强制转换，即对 CPSR 赋值为 0x000000d3，只有返回到 svc 态之后才能得到原来任务的 lr，这个对于任务切换很重要。还须留意在强制 CPSR 变成 svc 态之后，SPSR 也会相应地变成 SPSR_irq，这样就需要在强制转变之前保存 SPSR，也就是被中断任务中断前的 CPSR。

4.6.4 移植工作总结

由于各方面的原因，移植中编写的代码不一定完全正确，需要进行逐步调试。调试过程中，要善于根据具体现象来发现问题所在。根据笔者的调试经历，总结了一些移植调试过程中要注意的问题[49]。

(1) 对 μC/OS-II 的内核机理要有充分理解。可以尝试对其内核进行调试，这样可以帮助自己从更深的层次来理解嵌入式实时操作系统。只有对 μC/OS-II 的内核有了清楚地认识，才能在移植过程中发现问题的本质。

(2) ARM 处理器有七种模式和两种指令状态，除了五种异常模式外，正常工

作模式有系统模式和用户模式两种，虽然这两种状态下可访问的资源基本相同，但用户模式下用户进程不能随意更改处理器的模式，权限受限，系统模式访问所有资源而不受限制，所以最适合最作为操作系统的运行模式。

(3) 对所使用的编译器要有深刻细致的了解。由于 ADS1.2 的 C 编译器有很多优化编译选项和流水处理选项，在处理内核编译时很容易有冲突，所以，选择优化选项要慎重。加强对 ADS1.2 的 C 编译器的理解，可以节约代码编写时间。

(4) 在移植过程中，最容易出问题的就是堆栈处理，堆栈处理是操作系统移植的关键部分。可以先分析 ARM 系统自身在进行现场保护时的堆栈处理操作，然后再模拟其过程，另外要注意将需要的寄存器都保护到。由于在进行任务切换的时候采用了系统函数来进行现场保护，因此在堆栈初始化的时候，就应该按照这两个函数的操作来对任务堆栈进行初始化。

(5) 要详细分析 ARM 系统自身处理中断时的压栈操作，必须将多余的信息从堆栈中清理干净。如果对编译器不熟悉，可以通过仔细调试中断，来确定编译器的具体操作。在对 AT91FR40162 的移植中，由于 ARM 处理器有 7 种运行模式，每种运行模式下都有自己独立的寄存器，所以在处理中断的过程中，要注意运行模式的切换，避免寄存器的内容无法正确保存和恢复。

(6) 注意程序的返回地址。异常中断发生时，程序计数器 PC 所指的位置对各种不同的异常中断是不同的，所以，返回地址也是不同的。需要根据不同的中断类型，确定不同的中断返回地址偏移量，防止程序跑飞。

(7) 移植调试时最好的方法是采用模块法调试，可以先将各个模块单独调试好，再集成起来。具体可以先将内核与处理器无关部分和启动代码部分整合，再去考虑别的模块，这样便于发现问题。

(8) 测试要按照从简单到复杂的过程进行，先对 μC/OS-II 的基本功能模块如 OSInit( )、OSStart( )等进行测试，任务数也可以按照从少到多逐渐加多，另外还须注意几个给系统本身保留的优先级不能用，如果要用须在 OS-CFG..h 中进行配置修改。测试完成后要对 OS-CFG..h 的功能模块进行精简，去除不必要的配置，减少目标代码的所占用的空间，也能加快代码的执行速度。

至此，移植工作结束。

4.7 移植测试和效果

1．移植测试

对于移植后 μC/OS-II 的测试工作是验证移植效果关键的一步，可以按照由简单到复杂的过程进行。

首先，不加入任何应用代码机直接测试内核本身的运行状况，这样使得测试过程简单，能直观的发现内核移植过程中出现的错误；可以在工程中设计一个最

小工程项目的 main()函数。

```
#include “includes.h”
void main(void)
{
   OSInit();
   OSStart();
}
```

在使用 μC/OS-II 内核的任何功能模块前必须最先调用 *OSInit( )*，由于在 μC/OS-II 中至少需要一个任务，在调用 *OSInit( )*的同时 μC/OS-II 内核建立一个系统的缺省任务：空闲任务。当执行到 *OSStart()*时，意味着 μC/OS-II 将接收控制权，进行多任务调度，通过调用 *OSInit( )*、*OSStart( )*是对 μC/OS-II 的最基本测试，如果这一步过不去，前面的移植过程中必出现了致命的错误。

由于 μC/OS-II 中的功能块多，函数分布较散，各个标号和函数都要生成相应的链接对象，因此这个测试过程将出现很多编译器、汇编器及链接器的错误和警告，相对比较繁琐。如果前面内核的移植过程正确，在这一步中出现的错误主要是内核的功能函数的标识未定义或重复定义、数据类型不匹配，出现这种错误的原因是相关的头文件未包含进来或重复包含；警告主要是功能函数调用成功但是出现未申明等情况。

针对上述现象可以建立一个统一的 μC/OS-II 的应用头文件库 *includes.h* 头函数，采用条件编译方式将所有的头文件都加入进来，当应用函数需要相关定义头文件时，可以直接包含 *includes.h*。这样虽然可能占用一部分存储空间，但能方便实际应用。另外，要仔细观察出现的警告，不要掉以轻心，有些可能导致致命的错误。

当此 main( )函数成功编译并生成目标代码，这一步的测试通过，前面的移植过程基本是正确的。

第二步，在第一步的基础上加入系统硬件板的启动代码，在 Linker->ARM Linker 中定义系统的 RO Base 为 0x0100 0000；生成的目标代码为.elf 格式时 RW Base 设为 0x40,为.bin 时设为 0x00。入口地址 image entry point 也设为 0x0100 0000。最先执行的文件对象设为启动代码生成的链接对象文件 cstartup_ads.o。main()函数仍采用第一步的，编译后对照生成目标代码信息中总的 RO、RW(包括 ZI 空间)大小，要与第一步的基本保持一致，因为在 main()函数中没用启用其它的功能块，虽然加入了启动代码，但启动代码文件生成的.o 文件只占用系统的一定空间而不影响目标代码的生成空间。

将第二步的工程生成一个项目，在它的基础上逐个调用 μC/OS-II 的功能函数，逐次编译生成的目标代码写入处理器运行进行测试。具体包括 *OSTaskStkInit()*、*OSStartHighRdy()*、*OSCtxSw()*、*OSIntCtxSw()*、*OSTickISR()*等函

数的测试，具体可参考文献[20~21]。

2．移植效果

由于在移植的过程中采用了针对性的处理，移植成功后的项目模板和代码文件结构比较简练和清晰，见图 4-6、4-7。

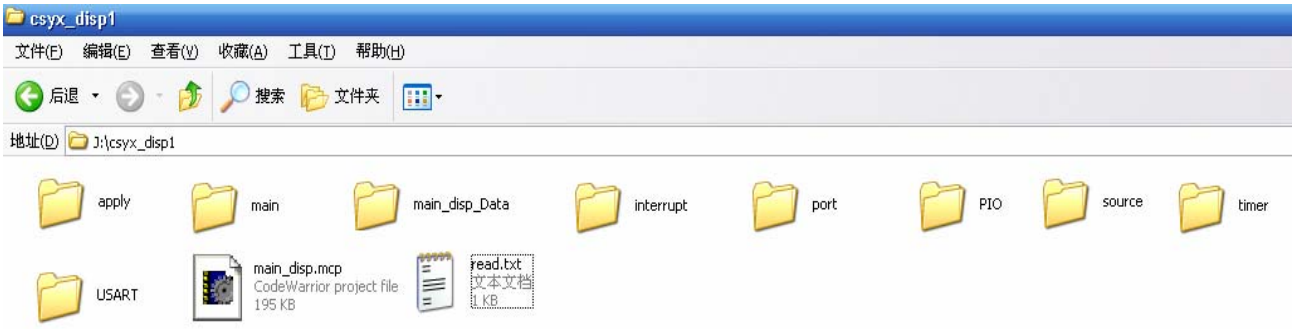

图 4-6　移植后的项目工程文件结构

因此，移植后的文件结构比较容易理解，在图 4-6 中 apply 文件夹中主要包含系统硬件平台上扩展的片外外设驱动；main 中是根据 μC/OS-II 特点设计的应用程序；Usart、timer、PIO、interrupt 等是 AT91FR40162 的片内外围的驱动库，source 中包含 μC/OS-II 中所有与处理器无关的代码；port 是移植中与处理器相关的代码部分。图 4-7 是在 ADS1.2 中移植完成后的项目文件体系结构，开发者可以将各种类型的代码统一放在一个组中，这样便于代码的优化，避免编译过程中出现的重复定义或多重编译，便于工程管理和编译。

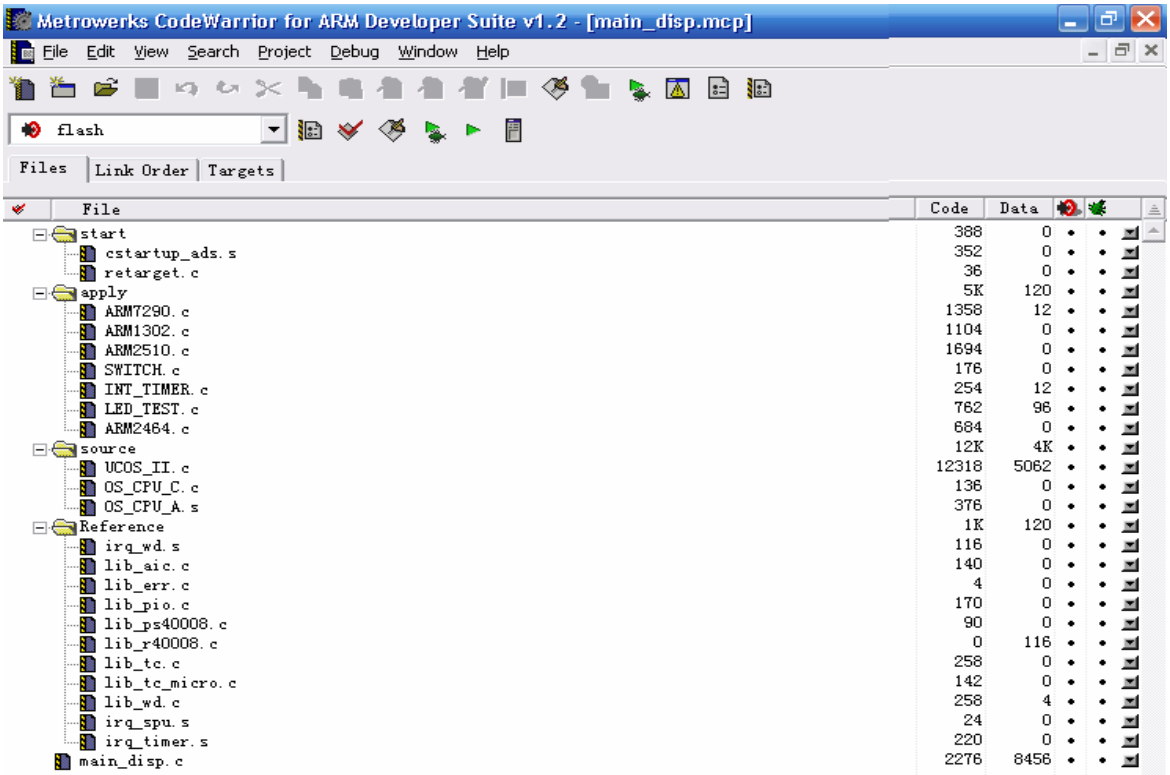

图 4-7　ADS 中的项目文件结构

另外，在移植 μC/OS-II 后，由于 μC/OS-II 的实时性特别好，而且 μC/OS-II 中各个任务执行的时间相对固定，执行时间与任务数无关，建立一个任务将 CAN 控制网络上的数据定时采集上来，然后定时发送到以太网上，这样可以大大改善系统通信的实时性和可靠性。

在 μC/OS-II 中最多可以建立 64 个任务，足够对过程对象设计不同的控制任务(应用程序)，设计者只要根据具体过程的特点，对各个任务赋予不同的优先级，这样可以对过程任务做到及时处理。同时 μC/OS-II 中有信号量、消息、消息队列、邮箱等，对于复杂应用系统中的不同任务间可以通过上述方式进行交互，使得任务可以自行挂起、中止或等待某一事件标志再执行，这样可以方便地实现复杂的多线程任务。

通过 ADS 的 AXD，采用单步执行调试方式，通过在 RAM 中观察各条指令执行的具体时间，可以得出片内外设的中断处理最快可以达到 5 微秒，对于 SJA1000、RTL8019AS 等扩展的外设，中断处理时间也在 20 微秒以内，因此系统的实时性可以得到保证。

4.8 小结

本章是整个系统实现的关键部分，在本章中详细讨论了系统上电后硬件启动代码的编写，同时针对系统的应用需要，移植了嵌入式实时内核，为系统建立了能实现多任务调度的 RTOS。

第五章 系统的外设驱动设计

至此，μC/OS-II 初步能在系统上运行了，现在主要任务是对嵌入式操作系统的功能进行完善，主要工作包括两个方面[51~52]：(一).对 μC/OS-II 功能模块以嵌入式操作系统标准进行扩充，建立一个实用的 RTOS；(二).完成硬件平台上的外设驱动程序的设计，并为操作系统建立一个标准的 API 功能函数库。在本章中将简要介绍基于 μC/OS-II 内核的扩展方法，重点介绍系统的外围驱动程序设计。

5.1 RTOS 任务扩展

μC/OS-II 内核移植完成后，对于处理器而言，仅仅是表示内核能完成多任务调度、生成的目标代码能运行了。在实际应用中，用户根本无法知道系统的运行情况，相对嵌入式设备的特征而言，还有不小差距。所需要做的工作还很多，主要集中在两个方面：

(一)在 μC/OS-II 的基础上进行扩展，将之扩展成实用的 RTOS。所要做的工作主要是以现代操作系统的特征对 μC/OS-II 的功能进行扩充，比如建立文件系统、创建图形用户接口(GUI)函数等，也包括部分实用的应用程序接口(API)函数。

(二)为系统硬件平台上的外部设备建立驱动程序并规范相应的 API 函数，为操作系统提供访问外围设备的标准统一接口函数，在实际应用时能把操作系统(系统软件)和外围设备(硬件)分离开来。

在本嵌入式通信接口应用系统中硬件外围设备较多，对于(一)的工作由于课题时间有限，没有开展，但是其重要性也很明显，它也是课题下一步的主要工作。该系统中没有设计显示模块，在调试时是通过串行口在 PC 机上看输出结果，这也是设计双串行口目的之一。如图 5-1 是基于 μC/OS-II 内核的 RTOS 的扩展整体框图。

在图 5-1 中，针对每一部分所需要完成的任务，对操作系统的各个部分划分成如下的模块[11]：

(1)系统外围设备的硬件部分

系统的外围设备的硬件部分包括 USB 接口模块、串行口模块、CAN 通信接口 SJA1000、以太网接口芯片 RTL8019 等。外围设备的硬件部分是保证系统实现指定任务的最底层部件。

(2)驱动程序模块

驱动程序是连接底层的硬件和上层的 API 函数的纽带，有了驱动程序模块，就可以把操作系统的 API 函数和底层的硬件分离开来。任何一个硬件的改变、删除、或者添加，只需要随之改变、删除、或者添加提供给操作系统的相应的驱动

程序就可以了，并不会影响到 API 函数的功能，更不会影响到用户的应用程序。同时，为了保证在实时多任务操作系统中对硬件访问的唯一性，系统的驱动程序要受控于操作系统的多任务之间的同步机制，可以使用信号量、邮箱等机制来进行协调。

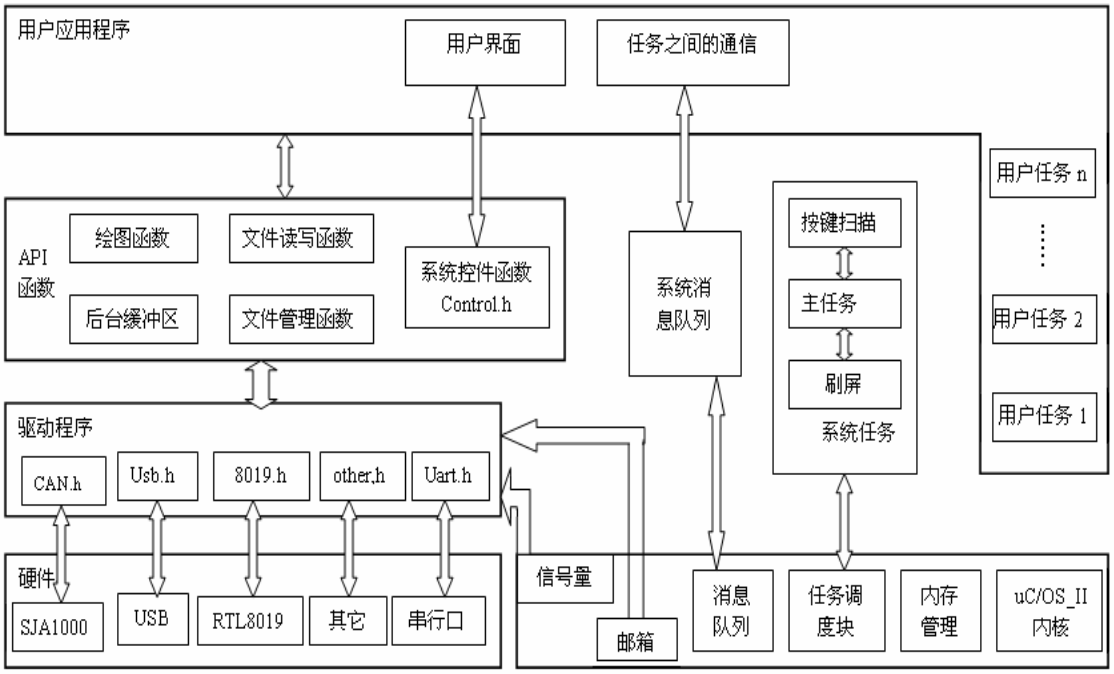

图 5-1 基于 μC/OS-II 内核扩展 RTOS 的整体框图

(3)操作系统的 API 函数

在操作系统中提供标准的应用程序接口(API)函数，可以加速用户应用程序的开发，统一应用程序的标准，同时也给操作系统版本的升级带来了方便。在 API 函数中，提供了大量的常用模块，可以大大简化应用程序的编写。

(4)操作系统的多任务管理

μC/OS-II 作为操作系统的内核，主要任务是完成多任务之间的调度和同步。

(5)系统的消息队列

系统的消息队列是 μC/OS-II 的消息队列派生出来的系统消息传递机制，用来实现系统各任务之间、用户应用程序的各个任务之间以及用户应用程序和系统各个任务之间的通信。

(6)用户应用程序

用户应用程序建立在系统的主任务(Main_Task)基础之上。用户应用程序主要通过调用系统的 API 函数对系统进行操作，完成用户的要求。在用户应用程序中也可以创建用户自己的任务。任务之间的协调主要依赖于系统的消息队列。

从图 5-1 中，可以看出(2)(3)是本系统当前迫切需要完成的工作，由于系统选用 AT91FR40162 处理器，ATMEL 公司提供了处理器各种外围的 AT91 C

Library，利用这些资源，结合 μC/OS-II 的功能函数，可以方便地编写出 RTOS 的应用程序接口函数(API)，也可以很方便的进行外围设备驱动程序的编写，这属于 RTOS 扩展的任务(二)的工作。对于(1)系统硬件已经固定，它的操作在任务(二)中实现，(4)(5)在 μC/OS-II 内部实现，属于 μC/OS-II 的功能范畴；(6)可以留待应用程序开发时再进行，应该说它属于任务(一)和任务(二)的公共工作。

5.2 外设驱动程序设计

在该系统中由于外设较多，为了加快对于外设，包括片内外设等的访问，处理器 AT91FR40162 内部集成了一个先进的中断控制器 AIC，能采用快速的硬件中断向量化来加快对外设的访问速度，见 4.2 节。下面的串行通信口、CAN 接口、以太网等都是采用中断方式(IRQ)进行与 CPU 进行数据交换，因此在驱动设计时有一定的共性，对于片外外设都是通过片选线片选，再通过 EBI 产生中断进行访问。因此在驱动设计时可以按照图 5-2 的流程进行设计，首先，在中断发生时读入 AIC 内容，判断是否是 IRQ 类型中断，然后关中断，跳到相应的中断处理函数，直至中断结束返回[12、53~54]。

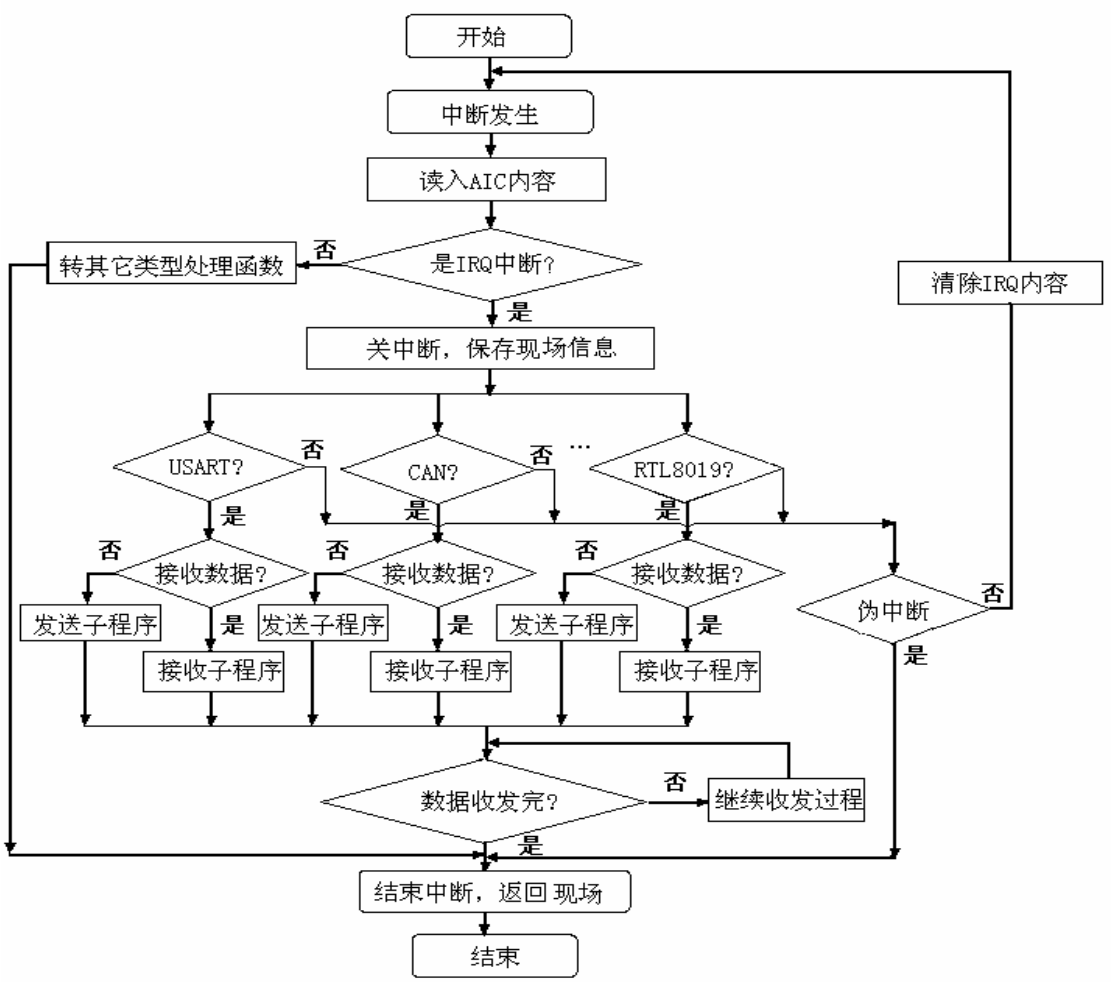

图 5-2 外设驱动程序设计流程图

5.2.1 串行口驱动设计

AT91FR40162 的 UART 单元提供两个全双工通用同步/异步收发器(USART)。UASRT 接到 APB 并与外围数据控制器 PDC 连接。每个通讯口均可工作在中断模式或 DMA 模式，也即 UART 能产生内部中断请求或 DMA 请求在 CPU 和串行口之间传送数据[55]。

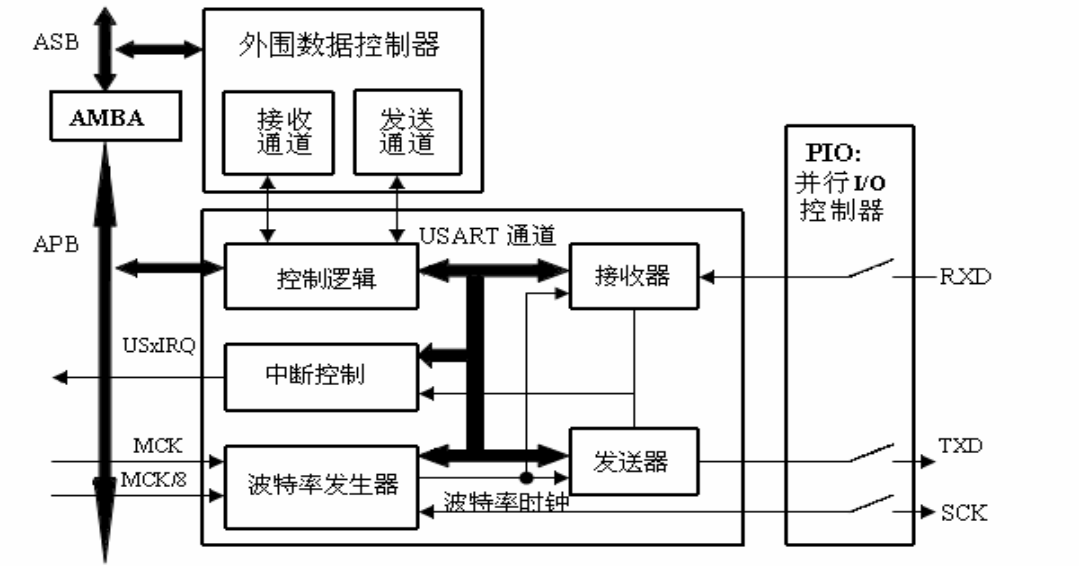

图 5-3 AT91 串行口原理图

串行口的接口函数按照功能模块可以分成下面几个函数[11]：

(1) 串口初始化函数，包括串口号、通信波特率设置等

定义：void Uart_Init(int Uartnum,int mclk,int baud);

参数说明：

Uartnum：所设定的串行口。

mclk：系统的主时钟。

baud：所设定的串行口通信的波特率。

(2) 输出字符串到串口函数

定义：void Uart_Printf(int Uartnum,char*fmt);

参数说明：

Uartnum：要输出的串口。

fmt：输出到串行口的字符串。

(3)指定串口接收字符串函数

定义：char Uart_Getch(char*Revdata,int Uartnum,int timeout);

参数说明：

Revdata：输入缓冲区。

Uartnum：设设定的串行口。

timeout：等待超时时间。

(4)向指定串行口发送数据

定义：void Uart_SendByte(int Uartnum,int data);

参数说明：

Uartnum：所设定的串行口号。

data：发送的数据。

5.2.2 CAN 接口驱动设计

对 32 位的 ARM，在一个读写周期里，地址和数据分两次送往总线，如果系统总线配置成 16 位模式，那么地址和数据会一次写到总线上去，和一般的 CPU 时序完全相同。通过总线扩展 SJA1000，可以将 SJA1000 当作是一个外部 SRAM 进行控制，通过对相应地址的寄存器读写可以初始化 SJA1000；当有数据需要输出时产生 IRQ 中断、以中断服务方式进行数据发送；接收数据时可以直接写芯片的相应数据缓冲和输出命令寄存器，即可实现 CAN 总线的收发功能。对照 SJA1000 和 ARM 的读写时序，可以发现 SJA1000 和 ARM 最大的不同在于它是 8 位总线的，而且总线采用复用方式[11、56]。

为了解决 ARM 的地址和数据总线非复用与 SJA1000 的复用总线之间的冲突，可以采用如下方法：首先 ARM 写外部总线，通过逻辑芯片实现片选信号、地址有效、读写信号非有效，同时 ALE 产生由高到低的跳变，锁存地址信息。然后 ARM 再次写或读外部总线，此时片选信号、写(或读)信号和数据有效，ALE 信号非有效，即完成了一个总线的写(或读)过程。上述方法的实质是通过两次读写 ARM 总线，根据其逻辑关系分别将总线数据作为作为地址和数据，实现对 SJA1000 的寄存器操作，其中上述逻辑功能可由 GAL22V10 实现。

SJA1000 的初始化函数 SJA1000_INIT()：主要完成工作方式的设置、接受滤波方式的设置、接收屏蔽寄存器和接受代码寄存器的设置、波特率参数设置和中断允许寄存器设置等。

发送子函数 CAN_SEND( )：负责节点报文的发送，只需要将待发的数据按特定的格式组合成一帧报文(数据帧或远程帧)，发送到 SJA1000 的发送缓冲区，然后通过中断 IRQ2 向 CPU 发出中断请求，启动 SJA1000 发送。

接收子程序 CAN_RECIVE( )：负责报文接收，可以采用查询方式接收，当接收缓冲区满，即启动接收过程，这样可能实时性差一些，可参考文献[57]。

5.2.3 嵌入式以太网接口驱动设计

在本系统中，由于选用AT91FR40162，不带网络接口，按方法一进行扩展，RTL8019AS支持即插即用和非即插即用两种模式[39、58~59]。在嵌入式系统中，网

卡不经常插拔，这样可以简化系统，因此本系统中将RTL8019AS配置为非即插即用模式，具体电路图见图3-3。

在驱动设计时将 RTL8019AS 的 RAM 的前 12 页(0x4000~0x4Bfff)作为发送缓冲区，后 52 页(0x4c0~0x7fff)作为接收缓冲区，第 0 页只有 32 个字节，用来存储以太网的物理地址。

RTL8019AS具有32个输出/输出地址，地址偏移量为00H~1FH。其中00H~0FH具有16个地址，为寄存器的地址。RTL8019AS的内部寄存器是分页的，每个寄存器是8位，在不同页面下，同一个端口对应着不同的寄存器，寄存器分为page0、page1、page2、page3，页面的访问通过CR寄存器的第6位PS1和第7位PS0来选择。复位端口包括18H~1FH共8个地址，用于RTL8019AS的复位。具体参考表5-1。

表5-1 RTL8019AS的寄存器

No(Hex)	Page0		Page1	Page2	Page3	
	[R]	[W]	[R/W]	[R]	[W]	[R/W]
00	CR	CR	CR	CR	CR	CR
01	CLD0	PSTART	PAR0	PSTART	9346CR	9346CR
02	CLD1	PSTOP	PAR1	PSTOP	BPAGE	BPAGE
03	BNRY	BNRY	PAR2	---	CONFIG0	---
04	TSR	TSR	PAR3	TPSR	CONFIG1	CONFIG1
05	NCR	TBCR0	PAR4	---	CONFIG2	CONFIG2
06	FIFO	TBCR1	PAR5	---	CONFIG3	CONFIG3
07	ISR	ISR	CURR	---	---	TEST
08	CRDA0	RSAR0	MAR0	---	CSNSAV	---
09	CRDA1	RSAR1	MAR1	---	---	HLTCLK
0A	8019D0	RBCR0	MAR2	---	---	---
0B	8019D1	RBCR1	MAR3	---	INTR	---
0C	RSR	RSR	MAR4	RCR	---	FMWP
0D	CNTR0	TCR	MAR5	TCR	CONFIG4	---
0E	CNTR1	DCR	MAR6	DCR	---	---
0F	CNTR2	IMR	MAR7	IMR	---	---
10～17	DMA 端口					
19～1F	复位端口					

嵌入式以太网驱动实现主要分成以下几个部分。

1．RTL8019AS初始化

在网卡的初始化时尽量使用RTL8019AS的默认配置和一些管脚在系统上电时的自动配置值，这样可以有固定的中断、端口地址。具体实现时可以用一片EEPROM作为配置存储器，来确定通信的端口地址、中断地址、网卡的物理地址、

工作模式、制造厂商等信息。RTL8019AS的初始化包括以下内容：

①设定临时使用的以太网物理地址。

②设定接收帧类型，设置为能接收与自己地址相匹配的数据包和广播数据包。

③设置工作模式，8位或16位，在本系统中因处理器有开放的16位数据线，所以工作于16位模式；RTL8019AS默认工作在8位模式，要使它在工作过程中以16位模式传输数据，就必须在初始化以前在AEN管脚上出现高低脉冲，才能使其工作于16位模式。

④初始化工作端口，设置成10Base-T。

⑤是否允许中断，即处理器对其响应方式可为中断或查询方式，在本系统中采用中断方式，通过IRQ中断向CPU发出申请。

⑥接收发送使能。

由于对RTL8019的操作都是通过对它的寄存器的相应位进行设置而实现的，寄存器的管理采用分页方式，下面代码实现8019的页面切换，在8019初始化以及数据收发过程中都要涉及到这些操作。

```
/* PSTART  接收缓冲区的起始页的地址*/
/* PSTOP    接收缓冲区的结束页地址*/
/* BNRY     指向最后一个已经读取的页*/
/*CURR     当前的接收结束页地址  */
   //实现 RTL8019 中的页面切换
void    EtherSetRegPage(char pagenumber)
{
pagenumber<<=6;
Ethernet_Reg00=0x22| pagenumber;
}
```

RTL8019 的初始化代码如下：

```
Ethernet_Reg00=0x21;       // CR=0x21;     //STOP |NO_DMA
Ethernet_Reg0e=0xc9;       //DCR 数据配置寄存器 16 位远端数据 dma
temp= Ethernet_Reset_Reg;
Delay(1);
Ethernet_Reset_Reg=temp;     //复位 8019
Delay(100);
   // 关闭对于配置存储器的支持
```

```
Ethernet_Reg00=0x e1;   //选中第三页
Ethernet_Reg01=0xc0;    //9346CR   EEM1=EEM0=1
Ethernet_Reg04=0x80;    //Config1 set IRQ bit
Ethernet_Reg01=0x0;     //9346CR   EEM1=EEM0=0
   //停止 8019
Ethernet_Reg00=0x21;   // CR=0x21;   //STOP |NO_DMA
Ethernet_Reg0b=0x0;    //清除远程 DMA 计数器的 MSB
Ethernet_Reg0a=0x0;    //清除远程 DMA 计数器的 LSB
    //读取 ISR，等待 ISR&ISR_RESET, 1.6ms
for(i=0;i<0xfff;i++)
{
    temp= Ethernet_Reg07;
    if(temp&RTL8019_ISR_RST)
    {
    Ethernet_Reg07=temp;
    Break;
    }
    Delay(200);
}
   //如果超时没有收到 8019 的复位信号，表示 8019 芯片有硬件问题
if(i=0xfff)
{
    Usart_Printf("Reset RTL8019 Fail.please check you hardware.\n");
      return;
}
Ethernet_Reg0d=0xe2;    //TCR=0x02;开启回环模式
Ethernet_Reg00=0x22;    //CR=0x22; //START|NO_DMA
Delay(20000);           //延时
   //配置 8019
Ethernet_Reg00=0x21; //选择页 0 的寄存器，网卡停止运行，因为没有初始化
temp= Ethernet_Reg00;
//测试 8019 的寄存器能否工作
If((temp&0x21)!=0x21)
{
```

```
Uaart_Printf("Write RTL8019 Fail.\n");
return;
}
   //配置8019 的寄存器
Ethernet_Reg0e=0x49;      //DCR 数据配置寄存器16 位数据dma
Ethernet_Reg0b=0x0;       //清除远程DMA 计数器的MSB
Ethernet_Reg0a=0x0;       //清除远程DMA 计数器的LSB
Ethernet_Reg0c=0x04;       //RCR
Ethernet_Reg0d=0x02;      //TCR=0x02 回环模式
Ethernet_Reg03=0x4c;      //BNRY
Ethernet_Reg01=0x4c;      //寄存器Pstart
Ethernet_Reg02=0x80;      //寄存器Pstop
Ethernet_Reg07=0xff;      //ISR
Ethernet_Reg0f=0x1;        //IMR open rx interrupt interrupt 1 == IRQ2/9
Ethernet_Reg04=0x45;       //TPSR
Ethernet_Reg00=0x61;       //CR_STOP | CR_NO_DMA | CR_PAGE1
   //选择页1 的寄存器
    //初始化物理地址
EtherGetMac(mac);
EtherSetMac(mac);
//初始化组播地址
EtherInitMar();
Ethernet_Reg07=0x4d;      //CURR
Ethernet_Reg00=0x21;       //回到页0
Ethernet_Reg00=0x22;       //选择页0 寄存器，网卡执行命令。
Ethernet_Reg0d=0x00;       //TCR_NO_LOOPBACK
Ethernet_Reg07=0xff;       //ISR
EtherNetOpenInterrupt();  //设置中断
EtherSetRegPage(0);        //选择页面0
```

2．RTL8019AS 发送数据

对于数据的发送，用户只要把数据写入缓冲区，启动执行命令，RTL8019AS 自动发送，一般要在 RAM 内开辟两个以太网数据包的空间作为发送缓冲区。以太网芯片自身能够完成数据包的校验，总线数据包的碰撞检测与避免，用户只需要配置发送数据的物理层地址的源地址、目的地址、数据包类型以及要发送的数

据即可，下面的代码实现 8019 的数据发送。

```
//发送 Mac 层数据包，PkData 是一个指向数据包结构的指针，包含了要发送的数据和长度
Void SendPackage(PMacHeader   machd,PackageData PkData[],int nPkdata)
{
  static U16 ptrBuffer=0x40;    //静态变量，记录发送的缓冲区的位置
  U16 traddress,totalbyte,temptype;
  int i;
  traddress=ptrBuffer<<8;    //发送缓冲区地址
  if(machd) //发送 Mac 层的数据包头，包括源地址和目的地址，数据包类型
    {
    temptype=(machd->type>>8)| (machd->type>>8);
    EtherDMAWrite16(traddress, machd->des,6);    //远程 DMA 写入
    traddess+=6;
    EtherDMAWrite16(traddress, machd->sourse,6);  //远程 DMA 写入
    traddess+=6;
    EtherDMAWrite16(traddress, &temptype,2);    //远程 DMA 写入
    traddess+=2;
    totaibyte=14;
    }
    else{ totaibyte=0;
    }
}
//发送数据，可以是多个数据包的组合
for(i=0;i<nPkdata;i++)
{
    EtherDMAWrite16(traddress, (U16*)PkData[i],pkData[i].datalength);
    totalbyte+= pkData[i].datalength;
    traddress+= pkData[i].datalength;
}
if(totalbyte<60)  //如果发送的数据不够 60 字节，补足 60 字节
    totalbyte=60;
while(Ethernet_Reg00&0x04);  //等待上一次发送的结束
 //设置发送寄存器
```

```
EthernetSetRegPage(0);
Ethernet_Reg06=totalbyte>>8;     //TBCR1
Ethernet_Reg05=totalbyte;        //TBCR0
Ethernet_Reg04=ptrBuffer;        //TPSR
Ethernet_Reg00=0x06;     //设置 TXP 位，开始发送
  //更新并记录发送缓冲区的位置
if(ptrBuffer==0x40)
    ptrBuffer==0x46;
else
    ptrBuffer==0x40;
}
```

3．RTL8019AS 接收数据

在 RTL8019AS 的初始化程序中已经设置好了接收缓冲区的位置，并且配置好了中断模式。当有一个正确的数据包到达的时候，RTL8019 会产生一个中断信号，在 ARM 中断处理程序中接收数据。

数据的接收比较简单，即通过远端 DMA 把数据从 RTL8019 的 RAM 空间读回 ARM 中处理。这里主要是对一些相关的寄存器进行操作就可以了。

5.3 应用程序设计

在 μC/OS-II 中任务也称作应用程序，是嵌入式系统设备真正体现出其控制作用的部分，与实际过程紧密相关。一般对于一个完整的嵌入式应用系统的开发，硬件的设计与调试工作仅占整个工作量的一半，系统的应用程序设计也是嵌入式系统设计非常重要的方面。与 PC 机系统类比，嵌入式系统的硬件可以比作裸机，嵌入式操作系统是系统软件，应用程序才是用户所需要的[60]。

目前，本系统中涉及具体的应用程序比较少，这里对系统的应用程序设计过程提出几点总结经验：

(1) 本系统中外围比较多，在应用程序设计时，可以考虑把生成的 API 函数以功能块为单位，再与启动代码联合一起编译，做成一个通用的工程模板，这个步骤可以在硬件测试过程中实现。这样可以节省应用程序的开发时间，利于系统各模块的集成，在具体应用过程中也可以有针对性地对功能块进行裁剪。

(2) 合理配置系统资源，这样可以最大限度发挥系统的硬件潜能，提高系统的性能。对于本系统来说，在应用程序设计时要注意合理分配 Flash、SDRAM、定时器/计数器、中断控制等资源。

(3) 由于嵌入式系统一般都应用在环境比较恶劣的场合，虽然系统在硬件设

计过程中尽量考虑过各种抗干扰等措施，不可能面面俱到，因此在应用程序设计过程中，也有必要考虑软件抗干扰设计。

5.4 系统的启动和运行过程

由于本系统的嵌入式处理器是 ARM7TDMI 核的，不带有 MMU(Memory Management Unit)。与带 MMU 部件的系统不同，基于 μC/OS-II 的 RTOS 是和用户应用程序是编译在一块，用户在设计时要清楚了解和配置各段代码执行先后次序，可以通过 ADS1.2 的参数设置界面进行配置。图 5-5 是在系统启动后，用户应用程序在系统上启动和运行的过程示意图。

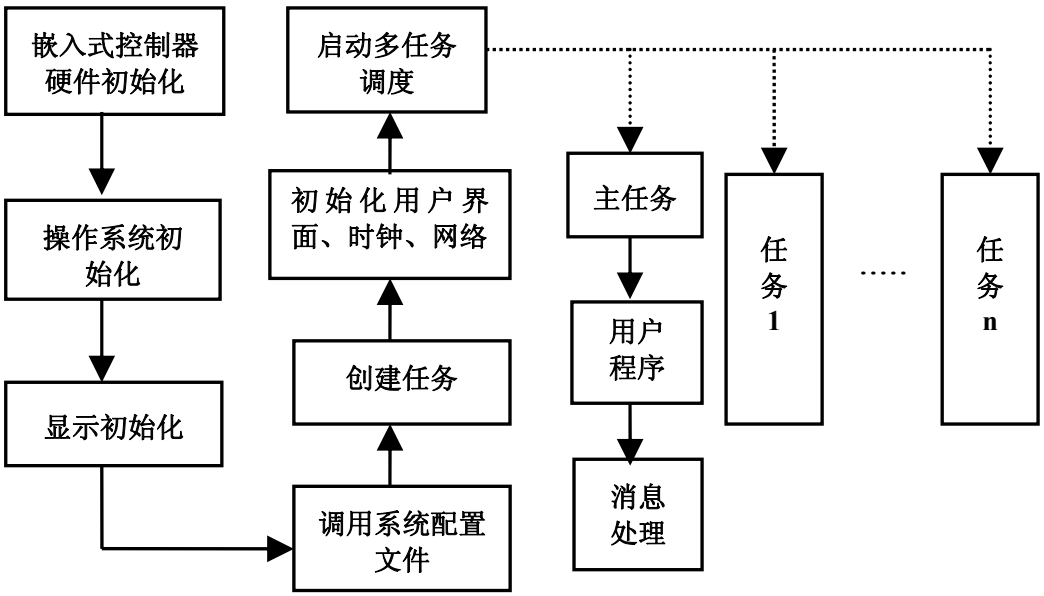

图 5-5 基于 μC/OS-II 内核的 RTOS 的启动示意图

从图中可以看出：类似于 PC 机的启动过程，嵌入式系统设备在上电后首先通过启动代码进行系统硬件信息配置；对操作系统进行初始化(启动系统定时器等)；对操作系统的显示模式进行配置；调用系统配置文件，运行系统默认任务；初始化应用过程信息，启动多任务调度，创建新的应用任务。也可以这样认为：RTOS 是一个功能强大的主控程序，它嵌入在目标代码中，系统复位后，能在硬件基础之上，为应用软件提供一个功能强大的运行环境，用户的应用程序都建立在 RTOS 之上。因此可以看出，有 RTOS 后，比直接对低层硬件编程更容易，而且可以统一对系统资源进行调度，用户只需要根据任务的优先级，为各任务分配合理的 CPU 时间即可[11]。

在调试时可以通过系统硬件平台上的串行口 0 与 PC 机的串行口通信来监视系统的运行情况。在图 5-5 的各个步骤中输出一些特定的字符来标识系统的运行情况。图 5-6 是在该系统上的一个项目工程文件编译后的结果，在该项目中特意

通过串口 0 向 PC 机输出部分运行过程中的数据信息，将该工程生成的目标代码写入系统嵌入式处理器 AT91FR40162 的片内 Flash，配置代码的启动顺序，上电后系统将自动运行。

图 5-7 是系统启动后到将所有任务运行一遍后的结果图，前面输出的英文字符串字面意思对应图 5-5 中相应的步骤，从“111”开始为 6 个具体的任务输出的运行结果。其中第一和第二个任务作用只是输出字符串“111”，“222”；第三个任务读取拨码开关的值；第四个任务是 SJA1000 收发数据，由于 SJA1000 同时收发数据，可以看到板上的收发 LED 状态灯周期闪烁；第五个任务读取功能键，第六个读取系统的时钟。由于在 μC/OS-II 中将所有任务在 main( )函数中做成一个“死循环”，后面的结果重复 6 个任务的输出结果，只是各个任务赋予的优先级不同，在运行时间次序有些差别。

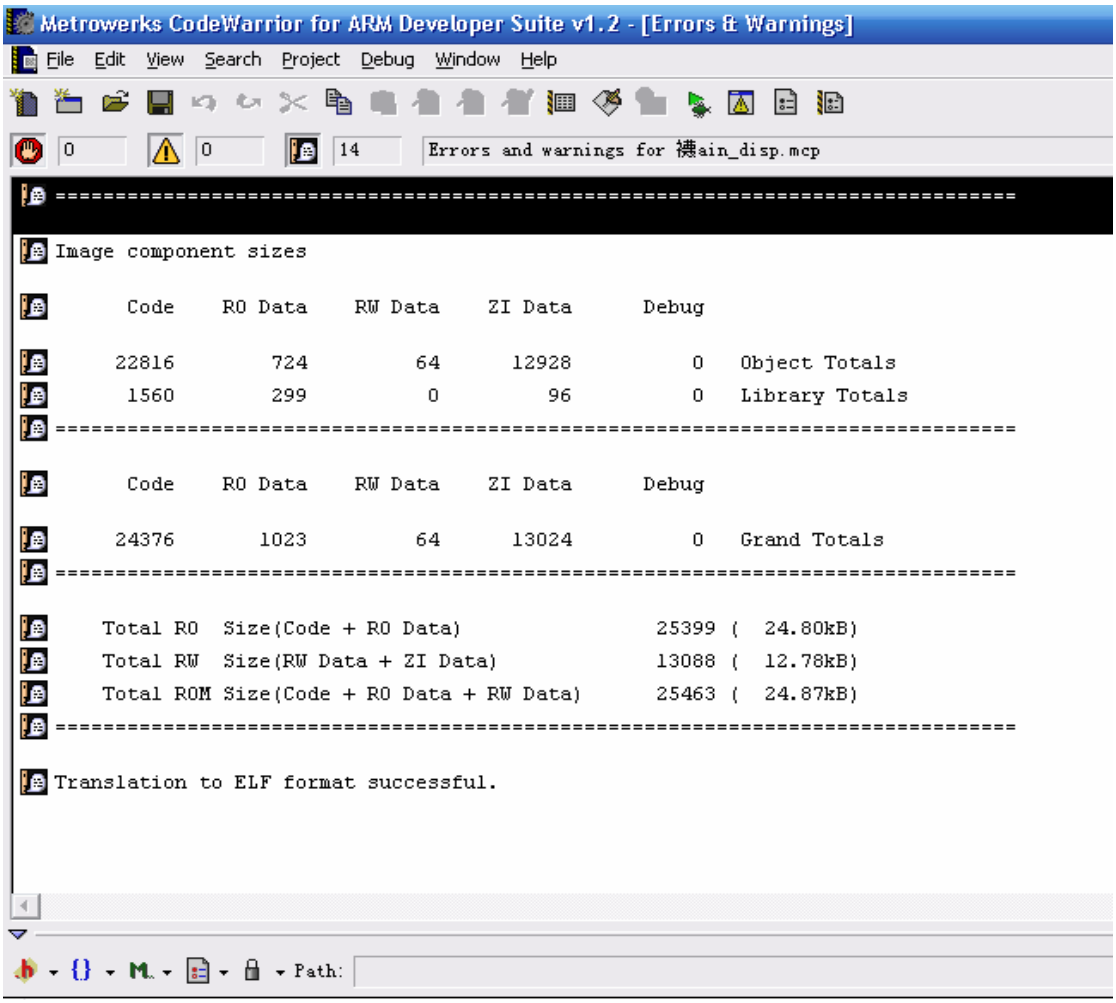

图 5-6　项目工程文件编译后的结果

```
embeded MCU and Board begin to initial
system begin to initial
display  begin to initial
system start
111
222
1. Select ID:
55
8
0
0
0
0
0
0
0
0
2. Select CAN
55
8
00
00
00
00
00
00
00
00
3. Select KB
0
4. Select DS1302
ff
ff
ff
ff
ff
```

图 5-7 系统运行的部分结果

结果表明，μC/OS-II 在本系统中的移植是成功，为它建立的驱动函数设计正确，系统的 RTOS 能够实现对复杂任务的调度，也说明本课题的研究选题定位是正确的，研究出来的成果有实际意义。

5.5 原型机测试

为了验证本系统中建立的 RTOS 的有效性，通过业翔科技发展有限公司相关技术人员配合，这里将它移植到该公司的新型电解铝槽控机的主显板上，进行了原型机的应用测试工作。

在主显板上主要以三组 4 个八段 LED 数码管显示电解铝过程控制中的参数和标识操作动作，参数主要包括系统的电压、电流，时分格式时钟显示、分秒格式时钟显示、系统操作次数统计等，控制动作主要有通过触摸开关实现的包括下料、阳升、手动、阳降、边加工、换极、出铝、打壳、抬母线等。对于电解铝过程控制，关键的参数主要是电压和电流，因此可以将整个显示部分分割成三个任务。按照 μC/OS-II 任务规范中建立三个应用程序，任务一专门用第一组八段 LED

数码管显示过程的电压；任务二专门用第二组八段 LED 数码管显示电流，这两个任务都必须通过 CAN 总线控制器 SJA1000 来接收 VF 板上送来的信号。由于人的视觉偏差为 0.1 秒，可以将任务一、二的执行时间定义为 0.1 秒执行一次，优先级定义为任务的最高和次高。第三组八段 LED 数码管通过触摸开关切换分别显示过程中别的参数和动作。具体实现过程可以分为以下几步。

①将一个通过测试的 μC/OS-II 内核移植工程导入。由于主显板的 CPU 也是采用 AT91FR40162，系统中的外围操作都是直接通过引腿进行操作，没有扩展片外的存储器，因此以前的启动代码也能直接加入。

②在工程中另建一个文件保存成 Main_Dis_Test.c 并添加到工程中来。在 Main_Dis_Test.c.c 建立一个 main()函数。

③在 main()中先对三组 LED 数码管清除显示；给 DS1302 时钟芯片赋初值；对 CAN 接口 SJA1000 初始化，系统开始计时(T1 定时时间到 LED8 闪烁)。

④系统上电后开始计数，并将计数结果通过 I^2C 总线存入 E^2PROM AT24C64 中，以后每一次上电或复位次数加“1”，记录系统的操作次数。

⑤判断 VF 板上发来的数据帧，为电压时转任务一接收 CAN 总线上发来的信号在第一组数码管上显示，否则转任务二将接收的信号作为电流值放在第二组数码管上显示，当超过 2 分钟没有收到数据，电压和电流值置 0 显示。

⑥轮询触摸键盘，根据变量 light 的值设置参数 LED 指示灯显示的位置，并通过 I^2C 总线将相应的值送到第三组数码管显示。注意在任务三中当 light 等于 1 和 2 时每秒钟发送一次时间并调显示子程序刷新第三组数码管时间显示。

⑦ 返回，重复步骤⑤、⑥。

图 5-8 是原型机的正面视图，左边是主显板的主板，右边是触摸键盘。

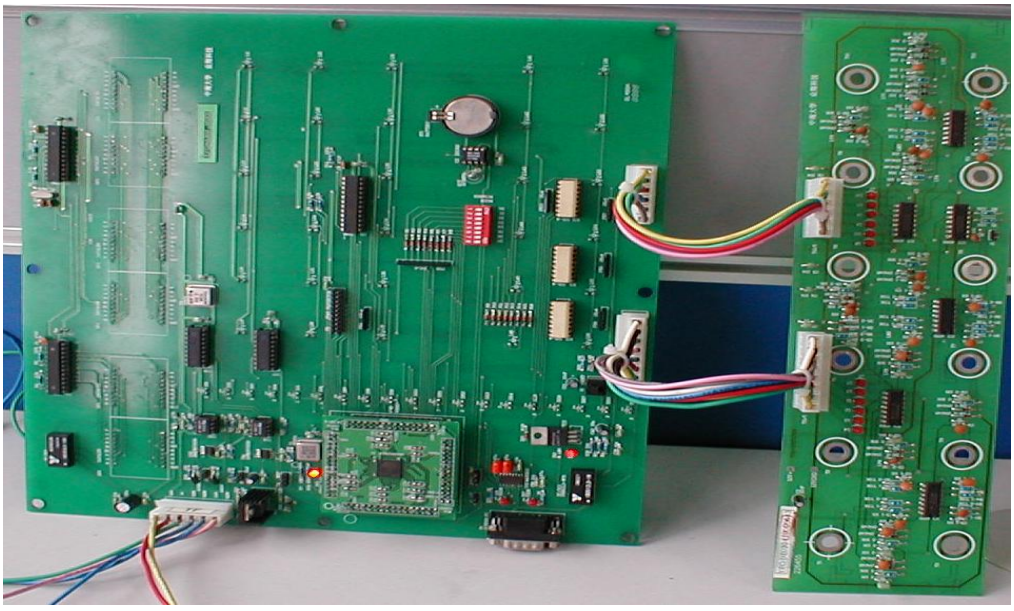

图 5-8　原型机的正面视图

图5-9是主显板上电后运行的实物效果图。此时第一二组数码管显示值不变，第三组数码管缺省显示“时分格式时钟”，按触摸键盘的“参数下翻”将切换到“分秒格式时钟”显示，同时相应的LED管下移一位，继续按键将切换到别的参数和动作显示。另外由于在主显板上设计了三组模拟 I^2C 总线来分别驱动三组 LED 数码管，时钟线公用 PIO 口的 P20 腿、发送线分别用 P18、P19、P21/TXD1 来模拟，在 P18、P19、P21/TXD1 上各接一个 LED 状态管来显示数据发送过程，因此在上电后这几个 LED 管将一直周期闪烁，这也是判断系统运行的标志之一。

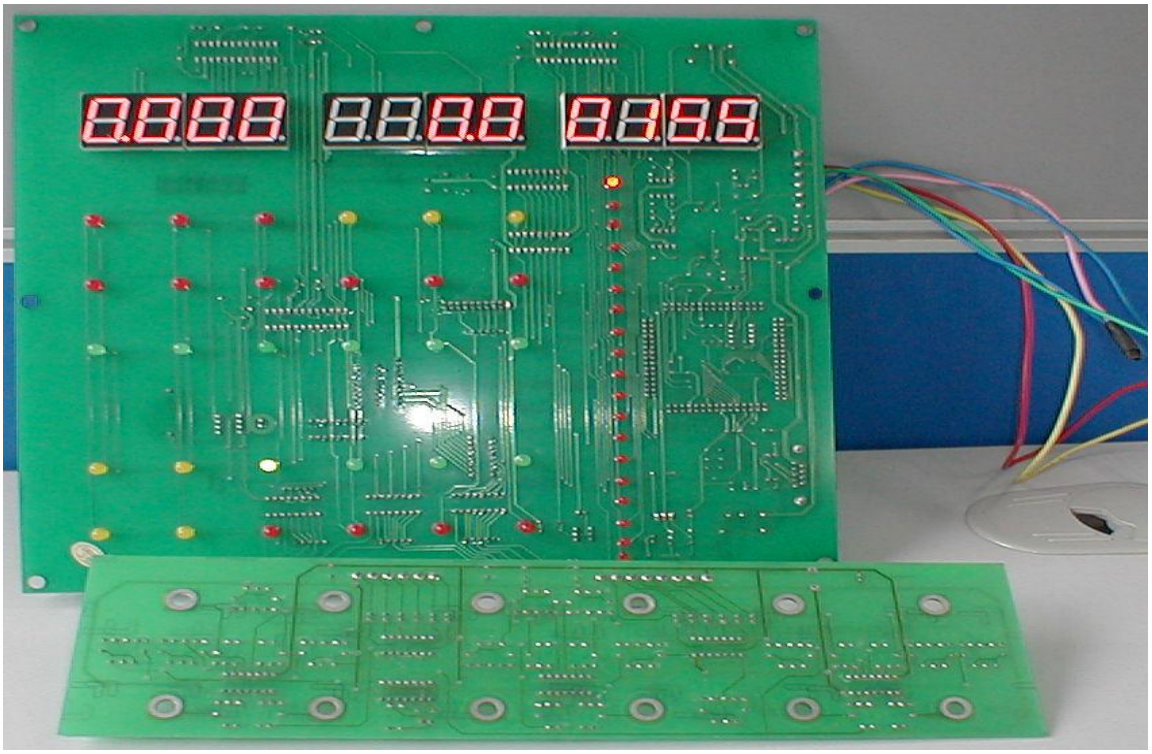

图 5-9 原型机的测试效果图

5.6 小结

在本章中首先介绍了 RTOS 功能扩展的主要工作，然后针对驱动程序这一部分工作，进行了串行通信口、CAN 接口 SJA1000、以太网接口 RTL8019AS 等关键外设的驱动程序的设计。最后给出了系统启动和运行的部分效果图及其在别的系统中的原型机测试应用。

第六章 结论与展望

本课题从实际工业应用出发，充分结合了嵌入式系统的优势，设计了一个适合分布式工业控制系统通信的嵌入式通信接口系统。

1．主要结论

(1)在本文中针对分布式控制系统的特点，以嵌入式技术为基础，巧妙构造了适合分布式控制系统中不同层次网络间的通信接口系统，思想比较先进。

(2)从系统总体设计入手，分析了系统的设计方法、开发工具的选择，系统的总体功能规划设计等，并根据实际需要，对嵌入式操作系统和嵌入式处理器进行了精心选择。较好地体现了嵌入式系统的设计流程，对类似产品的设计有很好的借鉴作用。

(3)以硬件平台的各单元模块电路的具体设计叙述了整个系统硬件平台的实现过程，并对本系统的关键部分 CAN 接口电路和以太网接口电路的设计过程特别进行了细致的分析。

(4)系统的软件设计围绕 μC/OS-II 内核的移植而展开，具体包括整个系统的启动代码设计、实时内核 μC/OS-II 的详细移植过程和移植中难点分析，为系统初步建立了一个实用的嵌入式 RTOS。

(5)指出了系统的 RTOS 的完善工作，并对其中的驱动程序部分完成了串行通信口、CAN 接口 SJA1000、以太网接口 RTL8019AS 等关键通信接口的驱动设计。并结合实际的任务，分析了本系统的启动和运行过程以及以其作为原型机的测试应用。

2．下一阶段的主要工作

(1)作为硬件平台的前端数据采集块，功能也比较简单，需要进一步完善，以太网接口这一部分还需要嵌入 TCP/IP 或是 UCP 协议，这些是以后的主要方向。

(2)对基于 μC/OS-II 的 RTOS 的功能模块还没有扩展完成，后面可以继续完成文件系统、GUI 等的设计、也包括部分驱动程序的完善；对于 GUI 这一块觉得除了实现一些基本操作系统的图形功能后可以考虑直接面向工业过程，做一个监控主界面，这样能体现嵌入式产品的特色。

(3)还可以考虑在平台上进一步改善嵌入式系统平台操作系统的实时性，使其具有较强的实时任务处理能力；

(4)尝试移植多种实时操作系统，使其使用具有广泛性；

(5)研究移植嵌入式实时数据库技术，能够对工业现场数据和状态信息进行实时保存和更新。

参考文献

[1] 中华人民共和国科技部工业自动化司.2020 年工业自动化领域科技发展规划建议[EB/OL].http://www.lodestar.com.cn/dl/news/109716.html，2005.12

[2] 金以慧.过程控制[M].北京：清华大学出版社,1993.3~8

[3] 王化祥,林慧.工业控制的应用现状和发展趋势[J].世界仪表与自动化,2003,(10):23~26

[4] 武胜林.现场总线控制系统的应用及困惑[J].世界仪表与自动化,2003,(10):18~22

[5] 丛力群.现场总线技术综述.上海宝信软件股份有限公司.2002, 9

[6] 胡翌博.基于 ARM 的嵌入式系统平台及其移植性研究[D]：[硕士学位论文].杭州：浙江大学，2004

[7] 楚红雨,李磊民,李驹光.嵌入式系统的现状与展望.中国科技论文在线,2005,9

[8] 蔡建平.关于嵌入式应用开发技术[J].单片机与嵌入式系统应,2001,(3):31~35

[9] 姚放吾.嵌入式系统的硬件/软件协同设计[J].微计算机信息,2001,(4):34~37

[10] Wolf W, Ti-Yen Yen. Embedded computing and hardware-software co-design[A], WESCON/95.Conference record: Microelectronics Communications Technology Producing Quality Products Mobile and Portable Emerging Technologies, 7-9 Nov, 1995. 450

[11] 王田苗.嵌入式系统设计与实例开发—基于 ARM 微处理器与 μCOS-II 实时操作系统[M],第 2 版. 北京：清华大学出版社，2002.1~203

[12] ARM Limited. ARM Development Guide, 2000

[13] ARM Limited. ARM Developer Suite:Assembler Guide, 2000

[14] ARM Limited. ARM Developer Suite:ADS1_2_Assembler, 2000

[15] ARM Limited. ARM Developer Suite:ADS1_2_LinkUt, 2000

[16] ARM Limited. ARM Developer Suite:ADS1_2_CodeWarrior, 2000

[17] 吕京建,肖海桥.嵌入式处理器分类与现状[EB/OL].http://www.bol-system.com, 2005,12

[18] 马忠梅,马广云,徐英慧,田泽.ARM 嵌入式处理器结构与应用基础[M].北京航空航天大学出版社，2002.185~215

[19] Pardo J,Campelo J.C.,Serrano, J.J. Robustness study of an embedded operating system for industrial applications[A].Proceedings International Computer Software and Applications Conference,Proceedings of the 28th Annual International Computer Software and Applications Conference,COMPSAC 2004,

Vol2, 2004.64-68

[20] Jean J·Labrosse, 袁勤勇译.嵌入式系统构架 [M].北京：机械工业出版社，2002. 56~77

[21] Jean J·Labrosse,邵贝贝译.μC/OS-II——源码公开的实时嵌入式操作系统[M]. 北京：中国电力出版社,2001.1~185

[22] 何小庆.选择 ARM CPU 的嵌入式操作系统.http://www.mcublog.com/more.asp?name= sharkdn&id=9048, 2005,12.

[23] Weng Guoqing,Chen Mingjun,Wang Furong. Embedded operating system and Internet applications for substation's local supervisory control[A]. Dianli Xitong Zidonghua/Automation of Electric Power Systems, Jun 25, 2004.Vol28, N12, 75~77

[24] Antoniotti Marco,Ferrari Alberto,Lavagno Luciano,Sangiovanni-Vincentelli Alberto, Sentovich Ellen. Embedded system design specification: Merging reactive control and data computation [A], Proceedings of the IEEE Conference on Decision and Control, 2001. Vol 4, 3302~3307

[25] Jean J·Labrosse.μC/OS-II 的特性介绍.http://www.ucos-ii.com, 2003, 11

[26] 周洁,杨心怀.32 位 RISC CPU ARM 芯片的应用和选型[J].电子技术应用,2002, (8): 50~52

[27] 黄清波.基于 AT91M40800 的嵌入式工业控制器的设计[D]：[硕士学位论文]. 杭州：浙江大学，2003

[28] ARM Limited. ARM7TDMI (Rev4) Technical Reference Manual, 2000

[29] 马忠梅.AT91 系列 ARM 核微处理器结构与开发[M].北京航空航天大学出版社,2002.25~45

[30] AT91FR40162 Data Sheet, ATMEL 公司，2000 年

[31] David Seal, ARM Architecture Reference Manual[M], Second Edition.2002: 15~65

[32] Steve Furber,田泽译.ARM SoC 体系结构[M].北京航空航天大学出版社,2002.12~68

[33] 邬宽明.CAN 总线原理和应用系统设计[M].北京航空航天大学出版社,1996. 78~96

[34] 阳宪惠.现场总线技术及其应用[M]. 北京：清华大学出版社，2002.75~83

[35] Yang Hua, Chen Ming-Yi, Hu Hui, Teng Yu-Hang. Research on CAN bus intelligent data acquisition system based on embedded system [J].Hunan Daxue Xuebao/Journal of Hunan University Natural Sciences.December,2005.32,(6),

110~112

[36] 黄清波,蒋鹏,王文海,孙优贤.ARM 与 MCS-51 时序转换方法研究[J].机电工程,2003(1):9~13

[37] Lee S, Kim S. Dual redundant arm system operational quality measures and their applications: static measures [J]. Control&Decision, Dec, 1990, 6(2), 3083~3088

[38] DATA SHEET SJA1000 CAN Stand-alone controller．PHILIPS Semiconductors 公司，2000.4

[39] DATA SHEET RTL8019AS.REATEK 公司，2000

[40] 范学领，陈一坚，索涛.基于 ARM 的工业级嵌入式以太网络接口实现[J].测控技术，2005, 24(4):36~38

[41] 李驹光,聂雪媛,江泽明,王兆卫.ARM 应用系统开发详解——基于 S3C4510B 的系统设计[M].北京:清华大学出版社，2003.12~96

[42] Chin Fan Saw, Lee Dack Fee. Design and implementation of intelligence embedded GSM auto-SMS system [A]. Proceedings of the Eighth IASTED International Conference on Internet and Multimedia Systems and Applications, Aug 16-18, 2004. 164~167

[43] 周立功著.DIUSBD12 USB 固件编程与驱动开发[M].北京航空航天大学出版社,2003.31~54

[44] MAXIM Limited.MAX3232Datasheet.http://www.maxim-ic.com/quick_view2.cfm, 2004,11

[45] 何希顺,张跃,何荣森.嵌入式系统中的 JTAG 接口变成技术[J].电子技术应用,2001.(9):25~29

[46] 曹跃胜.高速 PCB 设计技术[J].计算机工程与科学,1998,(11):45~46

[47] 诸邦田.电子电路实用抗干扰技术[M].北京:人民邮电出版社,1994.45~78

[48] 李章林,张立民.ANSIC 程序到 KeilC51 的移植心得.2003 年全国单片机和嵌入式系统年会论文集[A].2003 年

[49] 李明.μC/OS－II 在 SkyEye 上的移植分析.http://www.akaembed.org/Inetpub/forum/index.php, 2004, 12

[50] 杨洪亮,胡金演.μC/OS-II 在 ARM 处理器上的移植[J].微计算机信息，2005,21(5):31~34

[51] Jean J·Labrosse.μC/GUI 代码介绍.http://www.micrium.com/frames/products-ucGUI.htm, 2004,11

[52] Ron Franke.Porting application software in the embed system environment.

Electronic Engineering, 2000. 35~41

[53] Flautner K, Patel D.I, Intelligent energy management/sup TM/for portable embedded systems [A], SOC Conference, 2003.Proceedings.IEEE international [system-on-Chip], 17-20,Sept, 2003. 415

[54] Stokes Sam, Cheng Yi, Hayden Kathleen, Monemi Saeed, Chandra Rajan.Embedded system design for a smart rover[A].ASEE Annual Conference Proceedings,ASEE 2004 Annual Conference and Exposition:Engineering Education Researchs New Heights, 2004.15015~15025

[55] Namba K, Maru N. Redundant Arm Positioning Control of the Humanoid Robot by Linear Visual Servoing[A],Industrial Electronics,2004 IEEE International Symposium on Volume 1, 04-07 May, 2004,705~710

[56] Michael J. CAN bus based the instrumentation system [J].Diesel progress, 2003: (69):3~8

[57] 饶运涛,邹继军,郑勇芸著.现场总线 CAN 原理与应用技术[M].北京航空航天大学出版社，2003.154~165

[58] 葛永明,林继宝.嵌入式系统以太网接口的设计[J].电子技术应用, 2002(12): 48~51

[59] 刘国福,张屹,廖巍.DSP 与以太网的接口技术研究[J].电子技术应用, 2001(1): 38~41

[60] Olson, Michael A.Selecting and implementing an embedded database system[J]. Computer, Sep, 2000, 33(9):27~34

致谢

在本项目课题的研究、设计、调试和论文撰写过程中，我得到了导师李劼教授和丁凤其教授的悉心指导和亲切关怀。导师丰富的知识面、敏锐的思维、见微知著的洞察力、严谨的治学态度和永不懈怠的工作作风使我受益匪浅。导师的教导和精神将指导和激励我整个未来的科学探索生涯。在此对导师的关心和指导表示衷心的感谢。

非常感谢业翔科技发展有限公司常守恩高工和赵文超等的指导和帮助，他们的渊博的知识、敬业精神将对我今后的学习和工作产生巨大的影响。

另外，还要特别感谢中南大学信息科学与工程学院的陈湘涛博士后，他也为我的论文倾注很大的心血和给予很多无私的帮助。

在这三年共同学习、工作和生活过程中，还得到了师兄、师姐及同学们的宝贵支持。感谢刘代飞博士、张红亮博士所给予的无私的支持和帮助；感谢已毕业李文浩硕士、在读的杨剑硕士、陈红玲硕士，以及业翔公司所有技术人员在各方面科研工作中给了我极大的帮助和指导，没有他们的帮助，我也不可能完成我的课题。

最后，衷心感谢我的父母在我漫长的求学生涯中所给予的支持、鼓励与爱护。感谢所有关心、帮助过我的老师、同学和朋友们！也向参加论文评审的专家表示最衷心的感谢！

周立平

2006-4