中文摘要

随着企业信息化进程的深入发展，实现企业上层管理网络与现场控制网络的无缝连接显得越来越重要。但多种现场总线标准并存而相互间无法兼容的问题却一直困扰着工业界，因此将工业以太网应用到现场控制网络无疑已成为当前研究的热点和未来发展的趋势。

Modbus 是目前应用最广泛的现场总线协议之一，1999 年又推出了在以太网上运行的 Modbus/TCP 工业以太网协议。本文设计并实现了一种基于嵌入式系统的 Modbus/TCP 协议转换网关，允许用户通过以太网对 Modbus 现场总线设备进行监控，实现了以太网与现场总线的互联。

本文针对工业控制领域的应用选择了主流的 ARM920T 内核的 32 位高性能工业级嵌入式微处理器作为硬件系统的核心，其网络接口速度可达 100Mbps。软件系统移植并实现了开放源代码的嵌入式实时操作系统 μC/OS-II 和轻量级 TCP/IP 协议栈 LwIP。系统总体设计方案在较高性价比的基础上具有最佳的可移植性和可扩展性，同时根据嵌入式系统开发的特点建立了一套高效便捷的软件开发环境。在此基础上进行应用层程序的设计，实现了嵌入式 Web 服务和 Modbus/TCP 协议转换功能，并给出了详细的程序设计方法和工作流程，还提供了 Web 方式下更新网关固件中的配置信息的解决方案。测试结果表明该网关工作稳定可靠。最后给出了实际的应用案例。

本文侧重于工程应用问题的解决，重点不在于理论研究。本文实现的嵌入式 Modbus/TCP 网关对于工业以太网与现场总线互联的深入应用具有较高的参考价值和实际意义。

关键词： Modbus/TCP，现场总线，网关，嵌入式 Web 服务器，LwIP

ABSTRACT

With the in-depth development of information technology in enterprises, it is more and more important to realize an unified and transparent interface between upper management layer of enterprise and the field control layer. But the coexisting of multiple incompatible Fieldbus has challenged the industry community for a long time. Therefore, it is no doubt that applying industry Ethernet technology to fieldwork control is becoming attractive and being considered as the trend in future.

Modbus is one of the most widely used Fieldbus protocol. Modbus/TCP industrial ethernet protocol running over the TCP/IP network was issued in 1999. This paper designs and implements a Modbus/TCP protocol gateway based on embedded system, allowing users to monitor and control Modbus field device through Ethernet, which makes the inter-connection between the industrial Ethernet and the Fieldbus available.

According to the industrial control application, the mainstream industrial ARM920T 32-bit embedded Micro Processing Unit is chosen as the core of hardware system,which Ethernet transmission speed can reach 100Mbps. Open source embedded RTOS μC/OS-II and lightweight TCP/IP protocol stack LwIP are ported to the hardware platform as software system. The total design of the system gets not only high performance cost ratio but also best portability and extensibility. At the same time, the efficient and convenient softeware developing environment is established in terms of the characters of embedded system development. Based on works above the application layer programs are designed, which provides the Web service and the convertion of the Mobus/TCP protocol, with the detailed design method and work flow described. This paper also shows how to update the configuration information of gateway in the firmware by a Web server.The test results show that the gateway works well with good stability and reliability. A practical application case of the design is also presented in the paper.

This paper focuses on engineering problems rather than the theoretical research. Embedded Modbus/TCP gateway implemented in this paper provides a valuable reference to the interconnection of the industrial ethernet with the Fieldbus.

Keywords: Modbus/TCP, Fieldbus,Gateway, Embedded Web Server, LwIP

目　　录

第一章 概述 1

1.1 课题背景.....1
1.1.1 现场总线发展历史.....1
1.1.2 现场总线发展趋势与工业以太网.....2
1.1.3 串行通信标准与 Modbus 协议.....4
1.2 本课题实现的目标和意义.....6
1.3 本文的工作内容和结构安排.....7

第二章 系统的方案选择与总体设计 9

2.1 系统的功能需求.....9
2.2 嵌入式系统概述和发展历程.....10
2.3 嵌入式系统与工业控制机.....11
2.4 嵌入式微处理器的介绍与选型.....12
2.4.1 主流的嵌入式微处理器.....12
2.4.2 ARM 微处理器介绍.....13
2.4.3 嵌入式微处理器的选型.....14
2.5 嵌入式 TCP/IP 的实现方式.....15
2.6 实时操作系统的介绍与选择.....15
2.6.1 采用实时操作系统的必要性.....15
2.6.2 著名嵌入式操作系统简介.....16
2.6.3 选用 μC/OS-II 嵌入式操作系统.....17
2.7 系统总体设计方案.....18
2.8 小结.....19

第三章 硬件系统设计.....20

3.1 AT91RM9200微处理器.....21
3.2 电源模块.....21
3.3 时钟模块.....22
3.4 复位模块.....22
3.5 FLASH 和 SDRAM.....23
3.6 调试串口模块.....25
3.7 UART0 模块.....25
3.8 以太网模块.....26
3.9 硬件系统测试.....27
3.10 硬件成本分析.....28
3.11 小结.....28

第四章 开发环境的建立与 μC/OS-II 的移植.....29

4.1 开发环境的建立 29
4.1.1 ARM 集成开发工具 ADS....29
4.1.2 引导程序 Bootloader....29
4.1.3 开发调试方法....30
4.2 ARM 体系结构的基本概念 32
4.3 启动 μC/OS-II 的初始化程序 33
4.4 μC/OS-II 操作系统的移植....36
4.4.1 os_cpu.h 文件的修改 37
4.4.2 os_cpu_c.c 文件的修改....39
4.4.3 os_cpu_a.s 文件的修改....39
4.4.4 包含 includes.h 文件 40
4.4.5 时钟节拍初始化代码 41
4.5 μC/OS-II 的测试 41
4.6 小结....42
第五章 嵌入式 TCP/IP 协议栈的实现 43
5.1 计算机网络与 TCP/IP 协议栈 43
5.2 开放源代码源的 TCP/IP 协议栈 44
5.3 LwIP 的移植 45
5.3.1 LwIP 的整体结构与进程模型....45
5.3.2 LwIP 操作系统模拟层的修改....47
5.3.3 LwIP 以太网驱动程序的结构....50
5.3.4 LwIP 接口的数据结构与函数的设计 51
5.3.5 LwIP 硬件相关的函数设计....53
5.4 小结....56
第六章 应用层程序设计 57
6.1 Web 服务的实现 57
6.1.1 嵌入式 Web 服务实现机制 57
6.1.2 HTTP 消息格式....58
6.1.3 Web 服务程序设计 59
6.2 Modbus/TCP 协议转换的实现....62
6.2.1 Modbus 通信模式....62
6.2.2 Modbus 数据帧格式 63
6.2.3 Modbus 数据模型....65
6.2.4 Modbus/TCP 协议转换程序设计 65
6.3 测试....67
6.3.1 ARP、IP、ICMP 协议与以太网驱动测试....67
6.3.2 TCP、HTTP 协议测试....68
6.3.3 Web 服务测试....69
6.3.4 Modbus/TCP 协议转换功能测试 70
6.4 应用案例....72
6.5 小结....74

第七章 总结和展望.....75

7.1 总结.....75
7.2 展望.....76

参考文献 77

附录 A 嵌入式 Modbus/TCP 网关 PCB 及实物图 80

A.1 嵌入式 Modbus/TCP 网关 PCB 图 80
A.2 嵌入式 Modbus/TCP 网关实物图 80

附录 B 以太网物理层芯片 DM9161 原理图 81

附录 C Web 服务任务和 Modbus/TCP 任务代码 82

附录 D 文件列表及各文件编译后的代码量 83

D.1 嵌入式操作系统 μC/OS-II 程序 83
D.2 LwIP 程序 84
D.3 应用层程序和其他程序.....85
D.4 编译后的结果 85

第一章 概述

随着企业信息化进程的深入发展，实现企业上层的管理网络与现场的控制网络的无缝连接显得越来越重要。基于质量分析的生产管理、与安全相关的测试监控都要求现场的仪器仪表能对现场的信息进行处理并能及时被上级监控和管理网络访问与控制，并最终纳入到企业信息管理系统统一的框架里面[1]。

目前，构成底层控制网络的现场总线技术已获得了广泛的应用。如何使这种网络结构与以太网技术相结合，实现底层生产与上层管理的紧密集成是当前研究的热点。由于以太网还不能够真正解决实时性和确定性问题，大部分现场网络仍然会首选现场总线技术并与大量的旧有设备兼容。如何将现场总线与以太网相连，使用户能通过以太网对现场设备进行监控是本课题研究的主要问题。

为此，本课题设计了一种嵌入式协议转换网关，通过 Modbus/TCP 工业以太网协议将传统的串行链路通信的 Modbus 现场总线网络与以太网互联。该网关可应用在现场控制网络的升级和改造中，满足现场控制设备与以太网互联的要求。

本章首先介绍了现场总线的发展历程和发展趋势，然后介绍了工业以太网和 Modbus 协议，最后给出了课题的意义和工作内容。

1.1 课题背景

1.1.1 现场总线发展历史

工业控制在本质上需要一种分布式实时控制系统来实现任务的传送和执行。不同设备之间的交互是通过现场的网络来进行信息传递的。国际电工委员会(IEC)对现场总线(Fieldbus)的定义是：现场总线是一种应用于生产现场，在现场设备之间，现场设备与控制装置之间实行双向、串行、多节点数字通信的技术[2]。

现场总线技术是由过程控制技术网络化、数字化、现场化发展而来的，其间大概经历了 5 个阶段[3]：

① 人工控制阶段，利用压缩空气控制大型阀门的开启和关闭。

② 模拟仪表控制阶段，开始于 50 年代。电动单元组合式模拟仪表控制系统利用 0～10mA 或 4～20mA 的电流模拟信号驱动设备运行，从而实现了电气自动化。

③ 计算机集中控制阶段，这是在 20 世纪 60 年代自动控制领域发生的一次革命。利用计算机进行多任务控制，而在计算机外部仍然使用 4～20mA 的电流模拟信号进行数据传输。这种控制方式可以实现复杂的控制，但整个系统的可靠性主要依赖于中心计算机的可靠性。

④ 集散式控制系统阶段，开始于 70 年代中期。集散式控制系统(DCS)是4C(Computer、Controller、Communication、CRT)结合的产物。它的产生是伴随微处理器和微控制器的广泛使用以及计算机可靠性的大幅提高而来的。系统由多台卫星计算机和大量智能仪表组成，采用单元组合的方式构建整个系统。

⑤ 现场总线控制系统阶段，基于现场总线的分布式系统(FCS)是继 DCS 后的又一代控制系统。现场总线是控制室自动化系统与现场智能化设备之间的一种标准化的数字通信链路，可进行全数字化、双向、多站、总线式的数字信息通信。

现场总线是当今自动化领域发展的热点之一，被誉为自动化领域的计算机局域网。现场总线用于构成工厂底层控制网络，使系统具有开放性，设备间具有互可操作性。功能块与结构的规范化使相同功能的设备间具有互换性。同时控制功能被下放到现场设备使整个控制系统的结构具有很大的离散性。它作为工业数据通信网络的基础，沟通了生产过程现场级控制设备之间及其与更高控制管理层之间的联系。这导致了自动化系统结构和设备的深刻变革。

1.1.2 现场总线发展趋势与工业以太网

计算机、通信、控制技术的迅速发展和网络技术的迅猛发展给工业自动化领域带来了新的活力和深远的影响。现场总线作为该领域令人瞩目的焦点其主要的发展趋势有两个：一是总线标准将趋于统一；二是与以太网技术相结合。

(1) 总线标准将趋于统一

目前世界上大约有四十余种现场总线正在被使用。早在 1984 年国际电工技术委员会/国际标准协会(IEA/ISA)就开始统一现场总线标准，直到2000 年初终于获得通过，但制定的 IEC 现场总线国际标准却有 8 种之多。IEC/SC65C 又于 2001 年 8 月制定出由 10 种类型现场总线组成的 IEC61158 第三版现场总线标准[4]。这十种现场总线现场标准是 TS61158、ControlNet 和 Ethernet/IP、PROFIBUS、P-Net、FF HSE、SwiftNet、WorldFIP、NTERBUS、FF H1、Profinet。

现场总线的国际标准虽然制订出来了，但由于采用不同的网络技术，现场总线技术不能实现统一。这与 IEC 于 1984 年开始制订现场总线标准的初衷是相违背的[5]。就目前的情况来看，多种总线标准共存已经是不争的事实。产生这一现状的主要原因是：第一，工业自动化技术应用于各行各业，使用一种现场总线技术不能满足所有行业的技术要求。第二，技术的发展很大程度上受到市场规律、商业利益的制约，技术标准不单是一个技术规范，也是商业利益妥协的产物[2]。异种现场总线之间无法直接进行相互通信，往往需要专用网关进行协议的转换，这种协议的不兼容问题一直困扰着工业界。

(2) 现场总线标准与以太网技术结合

随着 Internet 的迅速发展，以太网已成为 事实上的工业标准。以太网是 IEEE802.3 标准所支持的局域网标准，最早由 Xerox 公司开发，后来得到数字仪表公司、Intel 公司和 Xerox 公司的联合扩展。按照国际标准化组织开放系统互联参考模型的 7 层结构，以太网标准只定义了物理层和数据链路层。后来 APARNET 制定了 TCP/IP 高层通信协议，并把以太网作为其数据链路层和物理层的协议，从此以太网便和 TCP/IP 以及 HTTP、FTP 等应用层协议紧密地联系在一起，统称以太网技术[6]。

图 1-1 是企业管理信息系统的结构。由于以太网技术应用广泛，线缆和接口设备价格相对较低、通信速率高、软硬件资源丰富、可持续发展潜力大[7]，以太网技术已经在工业企业综合自动化系统中的资源管理层、执行制造层得到广泛应用，同时已呈现出向下延伸直接应用于工业控制现场的趋势[8]。

基于这种发展现状，越来越多的人希望以太网技术能介入过程控制层，广泛取代目前种类繁多、标准不一的现场总线技术。如果底层网络引入以太网不仅可以使企业的管理信息 系统实现垂直方向的集成，而且能降低不同厂家设备在水平层面上的集成成本，所以以太网技术向控制系统的底层网络的延伸是自然而然的。

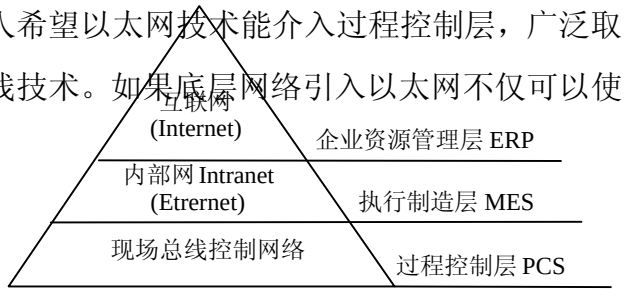


图 1-1 企业管理信息系统结构图

以太网最初是为办公自动化而设计的，采用带冲突检测的载波

侦听多路访问协议(CSMA/CD 协议)，因此缺乏确定性，比较脆弱，没有考虑工业控制的要求。在网络通讯比较繁忙的时候以太网可能发生某些节点的数据包不能及时发送，甚至长时间发送不出去的情况，同时以太网的抗电磁干扰(EMI)能力较差，若用于恶劣环境下的工业现场可能难以满足安全性方面的要求[8]。

工业以太网是以太网技术向工业控制领域渗透催生的产物，一般是指技术上与商用以太网(即 IEEE802.13 或 IEEE802.11 系列标准)兼容，但在产品设计、材质的选用、产品的强度、适用性以及实时性、可互操作性、可靠性、抗干扰性和本质安全等方面能满足工业现场的需要[9]。工业以太网基于成熟的以太网技术和 TCP/IP 技术，具有较高实时性和传输能力。

目前各大现场总线组织已纷纷开发出现场总线标准协议对应的以太网服务。国际上形成的工业以太网技术的四大阵营包括 Modbus/TCP、FF HSE 、EtherNet/IP、Profinet。其中 FF HSE 、EtherNet/IP、Profinet 是 IEC61158 第 3 版中所包含的现场总线，而 Modbus/TCP 已成为半导体工业标准 EMIE 54.922000[10]。

1.1.3 串行通信标准与 Modbus 协议

RS232/RS485 接口是计算机和工业控制领域应用最广泛的一种串行接口。从 1969 年美国电子工业协会(EIA)公布了 RS232 串行通信接口电气标准以来，该标准因为合理规定了接口的电气信号和机械要求而在全世界得到了广泛的应用[11][12]。

因为 RS232 线间干扰比较大，存在着传输距离不太远(最大传输距离 15m)和传送速率不太高(最大位速率为 20Kb/s)的问题，所以 1977 年 EIA 又制定了 RS485 标准。RS485 可以实现多点连接，采用二线差分平衡传输，最大传输距离可达 1200m(速率 100Kbit/s)，最大传送速率可达 10Mbit/s(距离 12m)。RS232 标准电平和 RS485 标准电平的转换非常容易[13]，而且很多串行通信模块同时支持这两种标准。

利用 RS485 接口可以使一个或者多个信号发送器与接收器互连，在多个设备和终端之间进行远距离数据通信，组建监控网络。因为RS232/RS485 标准只对接口的电气特性做出规定，不涉及接插件、电缆或协议，所以 RS485 总线标准不能称为现场总线，但是在此基础上用户可以建立自己的高层通信协议。RS485 具有接线少、成本低等特点，因此在许多智能仪器和设备上均配有 RS485 接口。RS232/RS485 在分布式控制 系统 和工业网络中得到 广泛 的应用， 具有 很强的生命 力。事实上

PROFIBUS，ARCNET、INTERBUS、Modbus 等现场总线的物理层都可以采用 RS485 总线标准。

最近几年 USB 技术发展迅速，在个人计算机领域逐渐取代了 RS232 接口。许多笔记本电脑上已经看不到 RS232 串行通信接口，同时许多台式计算机上 RS232 串口也从 2 个减少至 1 个甚至被完全取消。USB 接口具备支持热插拔，无需外接电源的优点，而且传输速率也大大高于串行通信速率。但 USB 只支持点对点通讯，单条 USB 电缆长度不能超过 5m，协议本身比较复杂开发难度较大，对旧硬件也支持不足，因此在工业控制领域 USB 接口并不能取代 RS232/RS485 串行通信接口[14]。

Modbus 现场总线协议是 Modicon 公司于 1978 年发明的一种工业控制器的网络协议。通过此协议，控制器相互之间、控制器经由网络和其他设备之间可以进行通信。Modbus/TCP 工业以太网协议是 Modbus 协议的另一版本，它于 1999 年被开发出来以允许用户通过以太网访问设备。

因为 Modbus 协议是属于应用层的通信协议，它与物理层的设备和电气接口无关，既支持传统的 RS232/RS485 通信标准也支持以太网通信标准。所以根据物理层层网络的不同 Modbus 协议被分成串行链路版本和运行在以太网的 Modbus/TCP 版本。

为了进一步推动 Modbus 协议在全球范围的应用，施耐德电气公司(收购了 Modicon 公司)成立了 Modbus-IDA 组织，将 Modbus 协议的所有权转交给分布式自动化接口组织(IDA，Interface for Distributed Automation)。Modbus-IDA 组织是一家非盈利的独立组织。该组织提供基础平台，使用户能得到并共享协议、应用和认证信息，最终降低使用成本。Modbus-IDA 被 IEC 授予世界领先的电气和服务标准组织“类别 D”的联络员身份，已提交了基于 TCP/IP 的 Modbus 和实时数据的发布与订阅协议(RTPSReal-time Publish/Subscribe)作为 IEC 的公共可用规范(PAS，Publicly Available Specification)的预标准。互联网编号分配管理机构(IANA，Internet Assigned Numbers Authority)也给 Modbus/TCP 协议赋予 TCP 端口 502[15]，这是其他工业以太网协议所没有的。

2004 年中国国家标准化管理委员会(SAC)也正式发布了基于 Modbus 协议的工业自动化网络规范的 3 个标准作为国家工业自动化标准：GB/Z 19582.1-2004 (Modbus 应用协议)、GB/Z 19582.2-2004(Modbus 协议在串行链路上的实现指南)、

GB/Z 19582.3-2004(Modbus 协议在 TCP/IP 上的实现指南)。

Modbus 协议作为一个具有悠久历史的事实上的工业标准加上其协议内容可以免费获得，因此 Modbus 的串行链路版本在市场上长期占有主导地位。大量的产品已直接运行或兼容 Modbus 串行链路版本。许多厂家的 PLC(可编程逻辑控制器)、HMI(人机界面)、组态软件都支持它。

Modbus/TCP 协议的开放性及用户对 Modbus 协议的熟悉程度再加上协议简单易学，使它比很多受商业利益驱使的协议取得更快的发展，受到众多第三方产品厂商、终端用户、系统集成商的广泛支持[16]。

我国许多大型项目中都采用了基于 Modbus 协议的产品和解决方案，如投资超过 450 亿人民币，总长超过 4000 公里的“西气东输”天然气输送管线项目中，整条管线的 SCADA 监控系统就是采用施耐德电气提供的基于 Modbu/TCP 的高性能自动化产品。世界上最大的水利发电项目三峡右岸电厂也采用了该自动化产品[17]。

1.2 本课题实现的目标和意义

将以太网技术应用于现场的控制层网络具有美好的前景，但由于以太网还不能够真正解决实时性和确定性问题，大部分现场层仍然会首选现场总线技术并且与大量的旧有设备兼容。当这些设备需要与以太网进行通信时就需要使用协议转换网关。

本课题实现的目标就是设计了一种嵌入式协议转换网关，通过 Modbus/TCP 协议将传统的串行链路通信的 Modbus 现场总线与以太网相连。用户经由以太网对 Modbus 设备进行数据采集以及对 Modbus 设备进行远程访问和控制。同时网关还必须提供 Web 服务功能，允许用户通过 Web 浏览器直接配置网关的 IP 地址等信息并保存到网关的非易失性存储器中。图 1-2 是嵌入式 Modbus/TCP 网关的连接示意图。

Modbus 协议采用请求响应模式，这恰好与互联网的客户机/服务器的通信模式相对应。本课题设计的嵌入式 Modbus/TCP 协议网关将 Modbus/TCP 客户机发送的 Modbus/TCP 请求帧重新封装成 Modbus 串行链路版的请求帧并转发到串行链路上，同时从串行链路上接收 Modbus 服务器的应答帧，将其封装成 Modbus/TCP 的应答帧并发送给 Modbus/TCP 客户机。

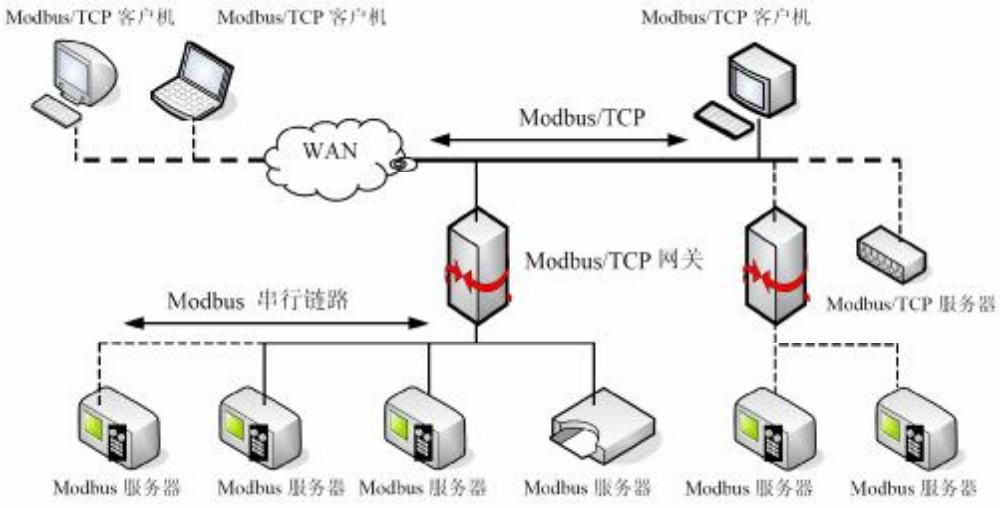


图 1-2 嵌入式 Modbus/TCP 网关连接图

虽然国内外已有公司推出类似的产品，如 MOXA、中新创科技等，但普遍价格昂贵，公开报价在 RMB1500 左右。这类产品大多采用 8 位、16 位的微处理器，没有外扩其他的功能模块，只实现协议转换，功能单一。而且这类产品的二次开发和产品功能的定制和扩展能力(如实现人机界面、web 交互、数据记录、状态报警等)非常弱，不公开源代码，仅有少数产品提供库函数调用，因此很难满足实际的需求。

本课题设计实现的嵌入式 Modbus/TCP 网关是一种理想解决方案。硬件系统采用 ARM920 核的 32 位高性能理嵌入式微处理器和 10M/100M 网络物理层芯片。软件系统采用 μC/OS-II 实时操作系统和开放源代码的 LwIP 轻量级 TCP/IP 协议。该网关采用嵌入式系统设计，硬件成本低、性能好、外围模块多；软件的开放性高、可移植性和可扩展性好。同时在嵌入式 Modbus/TCP 网关的基础上对应用层程序加以修改就能使其成为功能强大的 Modbus/TCP 工业以太网客户机或服务器。实验表明该嵌入式网关设计方案行之有效，产品具有很高的实际应用价值：

① 为现场总线与工业以太网的无缝连接提供一种方案

② 为“e 网到底”提供功能强大，移植性好的网络接口技术

③ 为工业以太网控制设备的开发提供快速原型，以缩短开发周期

④ 开发具有自主知识产权的嵌入式以太网技术

1.3 本文的工作内容和结构安排

本文主要完成了以下工作：

① 总体设计

根据系统需要实现的功能进行分析，完成硬件和软件系统的总体设计。

② 硬件系统的实现与调试

1) 主控芯片的选型，外围电路的设计和芯片选型，完成硬件原理图并绘制 PCB 电路板

2) 元器件的采购，焊接、测试，完成硬件系统

③ 软件系统的实现与调试

1) μC/OS-II 操作系统的移植

2) TCP/IP 协议栈的实现

3) 应用层程序设计、调试与测试

本文的结构安排如下:

第一章简述了课题背景、现场总线的起源与发展趋势以及 Modbus 协议的发展现状，并给出了毕业设计的核心内容及论文结构。

第二章详细阐述了嵌入式 Modbus/TCP 网关的设计方案的选择和系统的总体设计。

第三章阐述了硬件系统的设计与实现。

第四章阐述了系统开发环境的建立和 μC/OS-II 操作系统的移植和测试。

第五章阐述了 TCP/IP 协议栈的选择，详细阐述了 LwIP 轻量级 TCP/IP 协议栈在 μC/OS-II 和硬件系统上的移植包括操作系统模拟层的修改与网络驱动程序的设计。

第六章阐述了应用层程序的实现方法，包括嵌入式 Web 服务程序的实现即允许用户通过 Web 浏览器直接配置网关的 IP 地址等信息并保存和 Modbus 串行链路帧与 Modbus/TCP 帧的重新封装和转发的实现方法。最后对应用层程序进行了测试并给出了应用案例。

第七章对本课题进行总结和展望。

第二章 系统的方案选择与总体设计

嵌入式系统与一般的计算机应用系统一样，也是由硬件和软件组成的。硬件系统是指由处理器、存储器设备、输入输出接口设备等组成的系统；软件系统是各种程序的总称。软件和硬件只有紧密结合、协调一致，才能组成高性能的嵌入式应用系统。

嵌入式系统的开发过程必须遵循软件工程的原则。系统的开发可以分成三个阶段：需求分析阶段、设计阶段和实现阶段。需求分析阶段是确定要解决的问题及需要完成的目标；设计阶段主要是解决如何在给定的约束条件下完成用户的要求；实现阶段主要是解决如何在选定的硬件和软件系统上进行软、硬件的协调实现。

本章的第一节进行了系统的需求分析。在需求分析结束后，面临的一个棘手的问题就是硬件系统和软件系统的设计，因为设计的好坏直接影响实现阶段的任务完成。本章的第二、三、四节将介绍硬件系统的选型，包括开发平台的选择和处理器的选择；本章的第五、六节将介绍软件系统的选择包括操作系统的选择、TCP/IP 协议栈实现的方式。本章的第七节将给出了系统的总体设计方案。

在系统总体设计完毕之后就进入了系统的实现阶段。硬件系统的实现将在第三章进行阐述；软件系统的实现将分三章进行阐述：第四章阐述软件系统开发环境的建立和 μC/OS-II 的移植；第五章将阐述 LwIP 轻量级 TCP/IP 协议栈的实现；第六章将阐述应用层程序的实现。

2.1 系统的功能需求

本课题设计的嵌入式 Modbus/TCP 网关的主要功能是将串行链路版的 Modbus 现场总线与 Modbus/TCP 网络连接起来。网关通过以太网接口接收 Modbus/TCP 客户机发送的 Modbus/TCP 请求帧，对帧进行分析和处理之后封装成 Modbus 串行链路版的请求帧，通过网关的 RS232/RS485 的串行通信接口发送给 Modbus 串行通信网络中的 Modbus 服务器。Modbus 服务器在收到 Modbus/TCP 网关转发的请求帧以后，会向网关发送一应答帧。网关从串行通信网络中收到该应答帧之后，将其封装成 Modbus/TCP 应答帧，并通过以太网发送给 Modbus/TCP 客户机，完成上层以太

网对串行链路中的 Modbus 设备的访问和控制。

对于串行通信网络和以太网中的各个设备而言，Modbus/TCP 网关是透明的。网关需要完成 TCP 连接的管理，帧的解析和封装，数据的校验和时间管理等工作。

同时作为将来可实际运用的产品，网关的 IP 地址必须是可配置的，并且要保证断电重启之后配置信息不丢失。最完善的解决办法就是在平台上运行一个 Web 服务程序，允许用户通过 Web 浏览器直接配置网关的 IP 地址，子网掩码，子网网关地址。并将配置信息固化在网关的非易失性存储芯片中。

在系统总体设计的时候还需要考虑以下几个方面：

① 网关的实时性要求较高，因此选用运行速度较快的处理器能够及时接收和处理从串行通信口和网络通信口接收到的数据，提高网关的响应能力。选用 100Mbps 的以太网接口可以比 10Mbps 的以太网接口取得更好的网络通信速度。

② 网关最好选用低功耗处理器并能在相对恶劣的环境温度下工作。

③ 选用的处理器有较多外设功能模块，有 RS232/RS485 通用异步串行通信模块，最好带有以太网控制器。尽可能的集成一些常用的外设功能模块，如 USB、AD 转换、GPIO、中断模块，提高系统的集成度和可靠性并方便系统将来的扩展。

④ 调试方便，因为软件程序量比较大，调试工作有难度，所以需要处理器具有简单方便的在线调试功能。

⑤ 在成本许可的情况下，有尽量大的产品升级空间。拥有足够大的程序存储和运行空间，可以运行应用层程序及 TCP/IP 协议栈，支持操作系统的运行以及方便系统将来的扩展。在产品成熟之后，可根据具体的应用进行细分：售价便宜、功能单一，性能参数较低的产品可以选择相对低端的处理器，反之则选择高端的处理器。良好的硬件和软件的可移植性是实现这一目标的重要保证。

2.2 嵌入式系统概述和发展历程

嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户的应用程序等四个部分组成，用于实现对其他设备的控制、监视或管理等功能[18]。随着数字信息技术和网络技术的高速发展，消费电子、计算机、通信(3C)一体化的趋势日趋明显，

在后 PC 时代嵌入式系统受到越来越广泛的应用。美国著名未来学家尼葛洛庞帝 99 年 1 月访华时预言，4～5 年后嵌入式智能工具将是继 PC 和因特网之后最伟大的发明。

嵌入式系统的发展历程，大致经历了无操作系统阶段、简单操作系统阶段、实时操作系统阶段以及面向 Internet 阶段[19][20][21]。

无操作系统阶段：以单芯片为核心的可编程控制器形式的系统为代表。这种系统大部分应用于一些专业性极强的工业控制系统中，没有操作系统的支持，通过汇编语言程序对系统直接控制。程序运行结束后清除内存，初步具备嵌入式的应用特点。

简单操作系统阶段：20 世纪 80 年代，随着集成电路制造工艺的提高，生产了种类繁多的 CPU。此类系统开销小，效率高，但是通用性比较弱，设计出来的操作系统具有一定的兼容性和扩展性。基于这些操作系统开发了一些应用软件。

操作系统阶段：20 世纪 90 年代，随着嵌入式应用系统软件开发复杂度的不断增加以及大量高性能和多功能的高端片上系统(SOC)硬件核心芯片的推广，为基于嵌入式操作系统的应用程序开发提供了技术需求和硬件平台。嵌入式操作系统内核小、执行效率高、扩展性强，功能强大。利用操作系统提供的应用程序接口(API)编写应用程序简单方便，并可以利用操作系统进行应用程序的跨平台移植，大大提高了嵌入式软件的可重用性。嵌入式操作系统已为嵌入式产品的发展提供巨大的推动力。

面向 Internet 阶段：进入 21 世纪后，Internet 技术发展迅猛并且渗入到人们工作和生活的方方面面。同时嵌入式设备具有便利灵活、性能价格比高、嵌入性强、系统所需配置要求较低、系统专业性和实时性较强等优点，在消费类电子、工业控制、无线通信等领域得到广泛运用，可以预见嵌入式设备与 Internet 的结合将代表着嵌入式技术的真正未来。

2.3 嵌入式系统与工业控制机

由于个人计算机 PC 本身不是为工业控制所设计的，只是根据工业控制领域需要，针对工业控制方面的一些特点进行了修改，如将主板的尺寸进行缩小，采用机架式、塔式、刀片式结构，使它更加适合工业现场的摆放。这就形成了工业控制计

算机(以下简称工控机)。

嵌入式系统则对工业控制方面的应用专门进行了设计，在工作温度、抗电磁干扰、可靠性等方面都做了各种增强。因此嵌入式系统比工控机在工业控制领域具有更多的优势：

① 工控机通常采用和普通 PC 机一样的高性能 CPU，同时还接有键盘、硬盘、内存等部件，总功耗非常高，需要风扇来散热。嵌入式系统采用 16/32/64 位微处理器，处理能力接近低档的 PC 机，能够满足大部分的控制要求而功耗却非常低，为瓦级或者毫瓦级。因此嵌入式设备可以在密闭的环境中工作，不会因为风扇吸入灰尘以及工作环境中存在不良气体而影响设备的稳定工作。同时较低的功耗可以节省巨大的电能。

② 工控机内部连线多组装不方便，同时象硬盘这种设备，抗恶劣环境的能力比较差，可靠性低。嵌入式微处理器片内已集成大量外设控制模块，需要的外围器件大大减少，能真正的嵌入到设备当中，采用内建 FLASH、CF 卡存储数据，可靠性高，还能外扩各种输入输出设备如触摸屏、液晶控制器、音频控制器等，可提供功能强大的人机界面，丝毫不比工控机逊色。

③ 工控机通常基于非实时的 Windows 操作系统，而嵌入式系统可以运行各种实时操作系统(RTOS)，完全能够保证控制系统的强实时性。

④ 嵌入式微处理器可为控制系统专门设计，产品型号非常丰富，比工控机具有更宽的温度适应能力，性能更加可靠。

嵌入式系统的高实时性、低成本、专用化和高可靠的特点，克服了传统单片机控制系统功能的不足和工控机实时性和可靠性不足的缺点。随着嵌入式微处理器性能的不断提高，以及实时操作系统(RTOS)的不断发展，嵌入式系统在工业控制领域将具有更加广泛的应用前景。

2.4 嵌入式微处理器的介绍与选型

2.4.1 主流的嵌入式微处理器

嵌入式微处理器是嵌入式系统的心脏。选择一款合适的嵌入式微处理器对于整个嵌入式系统的开发具有非常重要意义。目前世界上的嵌入式微处理器的种数已经

超过千余种[22]，按照数据总线的宽度主要划分成 8 位、16 位、32 位三类。

8 位单片机是目前品种最丰富、应用最广泛的单片机，具有体积小、功耗低、功能强、性能价格比高、易于推广应用等显著优点。目前主要有 MCS-51 系列及其兼容机型，MOTOROLA 68HC05/08 系列、MICROCHIP 的 PIC 单片机以及 ATMEL 的 AVR 单片机。

16 位单片机运行速度及数据吞吐量比 8 位单片机有较大提高。目前主要有 Intel 的 MCS-96/196 系列、MOTOROLA 的 68HC12 系列等。

32 位微处理器目前发展很快，是单片机的发展趋势。随着技术的发展其开发成本和产品价格正在不断地下降。其中以 32 位 ARM 系列及 MOTOROLA 的 ColdFire、68K 系列的应用相对广泛。

2.4.2 ARM 微处理器介绍

目前 32 位微处理器的使用率正逐年增长，其中 ARM 系列微处理器最引人瞩目。ARM(Advanced RISC Machine) 公司是一家专门从事芯片 IP 设计和授权业务的英国公司，其产品有 ARM 内核以及各类外围接口[23]。ARM 处理器已遍及工业控制、消费类电子产品、通信系统、网络系统、无线系统等各类产品市场。早在 1999 年，ARM 核心的芯片生产量就已突破 1.5 亿，市场份额占了 32 位嵌入式微处理器的 50% 以上。在 2001 年度里，ARM 占据了整个 32、64 位嵌入式微处理器市场的 75%。在 2002 年度里，占据了市场的 79.5%，全世界已使用了 20 多亿个 ARM 核。

ARM 内核是一种 32 位 RISC 微处理器，具有低功耗、性价比高和代码密度高三大优点。许多一流的芯片厂商都是 ARM 的授权用户，如 Intel，Samsung，TI，Freescale，ST 等。ARM 已成为业界公认的嵌入式微处理器标准[24]。

ARM 系列微处理器无论从处理能力、功耗、市场占有率和开发工具等方面都具有不错的表现。生产 ARM 芯片的厂商众多，选择的范围很广，最重要的是 ARM 微处理器更新升级非常方便，ARM7、ARM9、ARM10 以及最新的 ARM11 的指令集是完全兼容的。

目前非常流行的 ARM 处理器核有：ARM7TDMI、ARM720T、ARM9TDMI、ARM920T、ARM940T、ARM946T、ARM966T、Xscale 等[25]。每一个系列的 ARM 微处理器除了具有 ARM 体系结构的共同特点外都会有各自的特点和应用领域。

ARM7 用于对价格和功耗敏感的低功耗产品，具备嵌入式 ICE-RT 逻辑，非常低的功耗，3 级流水线和冯·诺依曼结构，运算能力可以达到 0.9MIPS/MHz。

ARM9 系列微处理器为低功耗的 32 位 RISC 处理器，采用高性能和低功耗领先的硬宏单元设计，哈佛结构，5 级流水线，能够提供 1 .1 MIPS/MHz 的运算能力，适合用于对价位和功耗要求较高的应用[26]。

2.4.3 嵌入式微处理器的选型

在如此众多产品中，每一种产品都有其具体的市场定位，没有哪一种微处理器产品可以完全主导嵌入式系统开发。选择合适的嵌入式微处理器是一件艰难而复杂的工作，首先要熟悉众多微处理器的类型、特点，其次还要分析系统所要实现的功能和开发周期。

在进行微处理器选型时主要考虑的原则有：

① 处理器的性能：包括处理器时钟频率，内部存储器的大小。

② 外围功能模块：嵌入式微处理器或多或少的会集成外围功能模块。如果微处理片内集成了系统所需要的功能模块，则可以提高整个系统的集成度，减少元器件的数量，降低开发成本。

③ 功耗：许多消费类电子产品都要求微处理高性能、低功耗。如果用于工业控制，则对这方面的考虑相对较弱。

④ 支持的工具：方便实用的程序开发工具和调试工具对系统的实现具有巨大的推动作用，能加快开发进度，缩短开发周期，降低调试的难度等等。

⑤ 厂商的支持：厂商在推出一款微处理器的时候通常都会给出相应的技术手册、应用笔记、评估开发板及配套工具等。尽量选择厂商主要推广的产品系列，这样可以得到更多的技术支持，同时选择主流的微处理器体系架构，这样会有更多厂商的产品可供选择和替代。

AT91RM9200 是著名半导体厂商 ATMEL 力推的一款基于 ARM920T 内核的工业级嵌入式微处理器，主要针对系统控制、通信、汽车、医疗等领域。AT91RM9200 微处理器除具备 ARM 微处理器普遍的优势以外，其性能、可靠性和扩展性都大大超过普通的商业级 ARM7 核心的嵌入式微处理器，如应用比较广泛的 S3C44B0 以及已经停产的 S3C4510 芯片。在可靠性方面也优于商业级的 ARM9 核心的嵌入式

微处理器，如 S3C2410 等。综上所述，AT91RM9200 作为一款性能卓越的嵌入式微处理器，完全能担当嵌入式 Modbus/TCP 网关硬件系统的核心并满足系统功能的需求。AT91RM9200 详细的性能特性将在第三章第一节进行介绍。

2.5 嵌入式 TCP/IP 的实现方式

要实现嵌入式系统与以太网的互联就必须在嵌入式系统上运行 TCP/IP 协议栈，目前所采取的运行方式有以下三种：

① 采用固化了 TCP/IP 协议栈的专业芯片作为以太网通信接口。采用这种方式的系统开发周期短，但此类芯片往往价格偏高而且功能有限，需要外接其他功能模块，可扩展性差，应用范围较小。

② 采用 8/16 位微处理器、网络接口芯片和简化的 TCP/IP 协议。这种方式价格低廉，应用范围广泛，缺点是系统资源非常有限，系统性能不高，只适合一些比较简单的应用，系统的扩展不足。

③ 采用高性能 32 位微处理器和嵌入式操作系统，即在实时多任务操作系统上进行程序的开发和实现 TCP/IP 协议栈。系统的功能强大，扩展性和可移植性好，但不足之处是硬件成本比 8/16 微处理器实现的系统高，开发难度大，若采用非开放源代码的嵌入式操作系统和 TCP/IP 协议栈则购买价格昂贵。因此本系统将考虑采用开放源代码的嵌入式操作系统和 TCP/IP 协议栈。

2.6 实时操作系统的介绍与选择

2.6.1 采用实时操作系统的必要性

借助嵌入式操作系统的帮助可以大大提高嵌入式系统软件开发的效率和可靠性。嵌入式系统产品多少都带有实时性要求，当然不同类型产品的实时性要求是有差别的，比如在工业控制，汽车电子领域，实时性要求就比较高，要求设备对各种信息能做出及时的处理否则就可能出现故障以及灾难性的后果。在消费类电子等领域，产品的实时性要求就没有那么高。系统偶尔的延时和对信息的未及时处理虽然会有一定的影响但是不会造成太严重的后果。所以前者通常采用实时操作系统，而后者往往较多的采用非实时的操作系统。

所谓实时操作系统是指系统能及时响应外部事件的请求，在规定的时间内对该事件进行处理，并控制所有实时任务协调一致地运行的操作系统[27]。判断操作系统是否是“实时”的，一方面是在多大程度上充分发挥硬件潜力，即综合速度快慢的问题，另一方面也是反应的速度在多大的程度上得到保证的问题[28]。

嵌入式实时操作系统在目前的嵌入式应用中 使用越来越广泛，尤其在功能复杂、系统庞大的应用中显得愈来愈重要[29]：

① 嵌入式实时操作系统提高了系统的可靠性。在控制系统中不仅要求在硬件设计方面提高系统的可靠性和抗干扰性，而且也应在软件设计方面提高系统的抗干扰性，尽可能地减少设计漏洞和不可靠的隐患。实时操作系统的监视进程可以监视各进程运行状况，遇到异常情况时会采取一些利于系统稳定可靠的措施，如把有问题的任务清除掉。

② 提高了开发效率，缩短了开发周期。在嵌入式实时操作系统环境下，开发一个复杂的应用程序，通常可以按照软件工程中的解耦原则将整个程序分解为多个任务模块。每个任务模块的调试、修改几乎不影响其他模块。

③ 嵌入式操作系统充分发挥了 32 位微处理器的多任务潜力。32 位微处理器不仅速度比 8 位、16 位的快，而且它本身就是为多用户多任务操作系统而设计的。例如将 CPU 运行状态分成用户态、系统态，将系统堆栈和用户堆栈分开，允许用户从硬件和软件两方面对实时内核进行保护。

2.6.2 著名嵌入式操作系统简介

目前比较常见的嵌入式操作系统有： VxWorks 、μCLinux、eCos、μC/OS-II。VxWorks 是美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统。其良好的持续发展能力、高性能的内核、友好的开发环境、卓越的可靠性，被广泛应用于通信、航空、航天等实时性要求极高的领域中[30][31]。VxWorks 采用微内核结构，支持多种硬件环境。另外它还具有网络协议丰富、兼容性好和裁减性高等特点，具有程序动态链接和下载的功能。VxWorks 十分灵活，操作系统由 400 多个相对独立的、短小精炼的目标模块组成，用户可以根据需要选择适当的模块来裁减和配置系统。但其价格比较昂贵，对每一个应用一般还要另外收取版税，并且一般不提供源代码，只提供二进制代码。

μCLinux 是标准的 Linux 经过改造以后的 Linux 版本，目前广泛运行在 ARM、ColdFire、MIPS 等许多没有内存管理单元(MMU)的处理器上，并且专为嵌入式系统做了许多小型化的工作。这使得 μCLinux 的内核同标准的 Linux 内核相比非常之小，但是它仍保持了 Linux 操作系统的主要优点，如良好的移植性、稳定性，强大的网络功能，出色而完备的文件系统支持，以及标准丰富的 API 等。使用 μCLinux 最大的好处就是可以使用 Linux 下面无数免费的公开资源[32]。

eCos 即嵌入式可配置操作系统，最早由 Cygnus 公司发布，现由 RedHat 开发。它的一个主要技术特色就是功能强大的配置系统，可以在源码级实现对系统的配置和裁减[33]。虽然 eCos 是 RedHat 的产品但是 eCos 并不是类 Linux 的产品。与 Linux 相比，eCos 占用的资源更少，配置方法更清晰、更方便，系统层次更清晰明了，移植和增加驱动模块更加容易。同时 eCos 源码开放并且没有版税。

μC/OS-II 是 Jean J.Labrosse 编写的一个实时操作系统内核，现由 Micrium 公司开发维护。μC/OS-II 是专门为嵌入式应用设计的，除了与 CPU 硬件相关部分是用汇编语言编写外，其他绝大部分代码是用 C 语言编写。μC/OS-II 具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点，最小内核可编译至 2KB。μC/OS-II 已经移植到了几乎所有知名的 CPU 上。μC/OS-Ⅱ的源代码可供免费学习，但是使用 μC/OS-Ⅱ的产品需要购买产品生产授权[34]。

2.6.3 选用 μC/OS-II 嵌入式操作系统

本课题选择 μC/OS-II 的主要的原因有：第一，它的源代码是完全开放的，在网上可以直接下载，并且在不断的更新升级，可以根据系统的实际需要对代码进行分析和剪裁以提高系统的性能和实现的功能，如内核、信号量等各种参数都可以进行配置，这对实时性要求严格的嵌入式系统非常重要。利用 μC/OS-II 操作系统的源代码还可以对操作系统进行扩展。第二，μC/OS-II 基本上是免费的。用它来开发应用程序，不需要支付任何费用。第三，系统内核的实时性强、可靠性高。该操作系统得到了美国联邦航空管理局(Federal Aviation Administration)对用于商用飞机的符合 RTCA DO-178 标准的认证。这表明该操作系统的质量得到了认证，可以在任何场合放心使用[34]。第四，μC/OS-II 做为一个开发平台，它的可移植性很好，可以比较方便的移植到各种微处理器上，并实现处理器升级或更换处理器类型。甚至可以不用

了解系统硬件，直接在操作系统上开发应用层程序，定义每个任务和与操作系统的接口，实现应用程序的标准化，大大方便了程序开发中的分工合作和代码的修改与扩展。第五，网上有丰富的第三方软件可供下载。许多厂商和组织会给出在 μC/OS-II 下产品或者协议的示例性代码，同时 μC/OS-II 的广泛应用也为开发者提供了大量的参考资料和交流经验。

2.7 系统总体设计方案

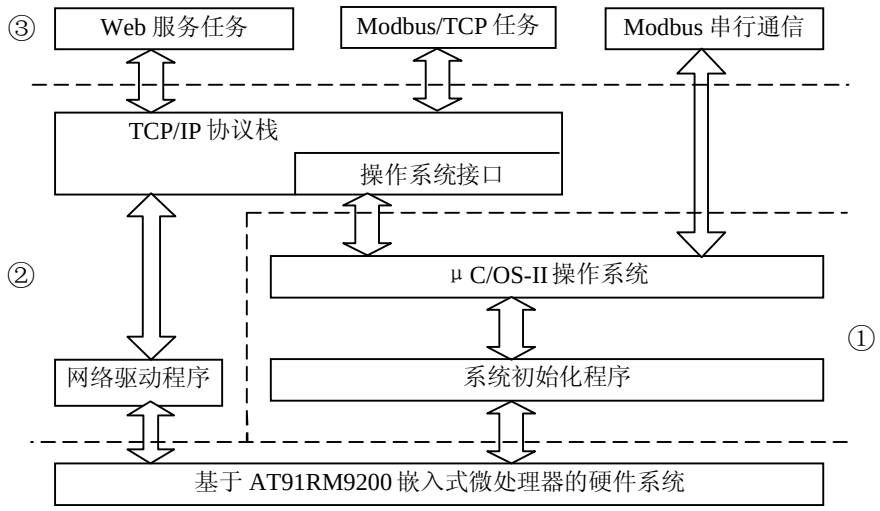


图 2-1 系统的总体设计结构

在综合各方面因素后，本课题选用了 ATMEL 公司生产的基于 ARM920T 内核的 AT91RM9200 高性能工业级嵌入式微处理器作为网关硬件系统的核心。网关的软件系统将移植 μC/OS-II 嵌入式实时操作系统和实现 TCP/IP 协议栈，并在此基础上设计应用层程序。系统的总体设计结构如图 2-1。软件系统的开发分成三个部分：

① 软件系统首先需要设计系统启动代码和移植 μC/OS-II 操作系统。因为 μC/OS-II 与硬件不直接相关，所以需要根据硬件系统专门设计初始化程序将系统的软硬件环境带到一个合适的状态，为启动 μC/OS-II 准备好正确的环境。

② 为了实现嵌入式 Modbus/TCP 网关的网络通信功能，软件系统需要将 TCP/IP 协议栈移植到 μC/OS-II 操作系统和硬件系统上。开放源代码的 TCP/IP 协议栈通常不针对特定操作系统和硬件平台，所以需要专门设计与操作系统的接口和网络硬件驱动程序。

③ 最后还需要设计应用层程序，实现通过 Modbus/TCP 协议将 Modbus 串行通

信链路与以太网相连功能和通过网关的 Web 服务对网关进行配置的功能。为此需要在 μC/OS-II 和 TCP/IP 协议栈之上编写相应的任务函数。

目前现场总线与以太网互联的设计方案还有以下几种：一，利用工业控制机实现网络的互联；二，8/16 位微处理器和通用网络接口芯片配合简化的 TCP/IP 协议栈实现网络的互联；三，固化了 TCP/IP 协议栈的专业芯片作为以太网通信接口实现网络的互联。

本课题的设计方案与以上三种以太网互联方案相比，在成本方面，大大低于第一种设计方案，仅为其 1/10～1/20；略高于第二和第三种设计方案，约为其 3～4 倍。在性能方面，略低于第一种设计方案；高于第二种和第三种设计方案。在扩展性方面，与第一种方案接近，大大高于第二种和第三种方案。在开发难度方面略高于以上三种方案。同时本课题的设计方案将采用开放源代码的嵌入式操作系统和 TCP/IP 协议栈因此在取得较高性价比的基础上可取得最佳的扩展性。

2.8 小结

本章首先对嵌入式 Modbus/TCP 网关进行需求分析。其次在对嵌入式系统与工业控制机进行比较的基础上提出课题采用嵌入式系统设计的优势；在对市场上种类繁多的嵌入式微处理器进行分类和比较后根据嵌入式微处理器选型的原则确定了硬件系统的主控芯片 AT91RM9200；在比较了几种著名的嵌入式实时操作系统和嵌入式 TCP/IP 的实现方式后提出了由开放源代码的 μC/OS-II 操作系统与 TCP/IP 协议栈组成的软件系统平台。最后提出嵌入式 Modbus/TCP 网关的总体设计方案。与其他以太网互联方案相比较，本课题的设计方案在取得较高性价比的基础上可取得最佳的扩展性。

第三章 硬件系统设计

在嵌入式系统的设计的过程中，硬件系统是首要，它是软件设计的基础。在第二章的系统总体设计中课题已经选用了 AT91RM9200 工业级嵌入式微处理器作为嵌入式 Modbus/TCP 网关硬件系统的主控芯片。为了实现网关的功能，硬件系统还需要其他的外围模块和功能电路。硬件系统的原理图和 PCB 板的设计采用 Protel DXP 专业制图软件。硬件系统的结构如图 3-1。

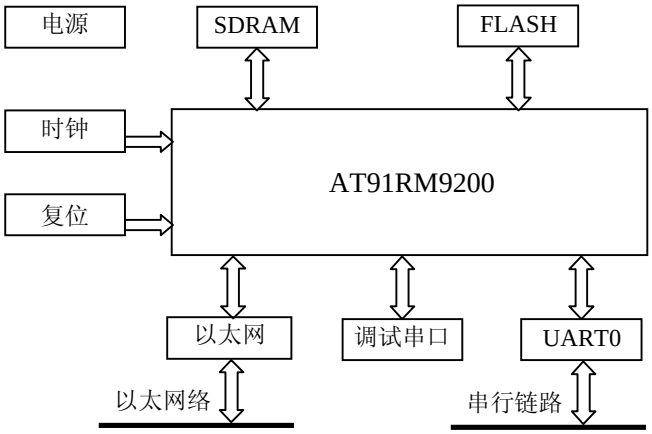


图 3-1 硬件系统的结构

① 电源模块：外部输入的直流 5V 电源经过 DC-DC 转换，得到 3.3V 和 1.8V 工作电源，供系统使用。

② 时钟模块：为 AT91RM9200 提供 18.432MHz 和 32.768KHz 的快慢晶振。

③ 复位模块：为 AT91RM9200 提供稳定的低电平复位信号。

④ FLASH ：扩展系统非易失性存储容量，用于存放程序代码、常量表和一些在系统掉电后需要保存的用户数据等。

⑤ SDRAM ：扩展系统的内存容量，主要用作程序的运行空间，存放数据和开辟堆栈。

⑥ 以太网模块：连接以太网络，接收和发送 Modbus/TCP 帧和 HTTP 消息。

⑦ 调试串口：AT91RM9200 调试串行通信接口，用于输入和输出调试信息。

⑧ UART0 通用串行通信模块：连接 Modbus 串行通信链路，用于接收和发送 Modbus 帧。

本章将详细介绍这些模块的硬件设计电路。

3.1 AT91RM9200 微处理器

AT91RM9200 是 ATMEL 公司生产的嵌入式微处理器，主要针对系统控制、通信、汽车、医疗等领域，具备先进的电源管理模块，集成了丰富的系统功能、外设模块和多种标准接口，为最大功能、最低功耗、最低成本的嵌入式系统提供了一个完整的解决方案[35][36][37]。其主要的性能特点如下：

① 内部集成了 ARM920T 一 ARM Thumb 处理器，在 180MHz 速度运行时性能高达 200MIPS，采用哈佛总线结构，内带 16KB 的数据 Cache 和 16KB 指令 Cache，支持全功能的内存管理单元(MMU)，片内带有调试通信通道的仿真器

② 片内集成有 16KB 的 SRAM 和 128KB 的 ROM

③ 外部总线 EBI 接口支持 SDRAM、Static Memory、Burst Flash 存储器，直接支持 CompactFlash，SmartMedia 和 NAND Flash

④ 增强功能的系统外围设备，增强型的时钟发生器和电源管理控制器；支持调试通信的两线 UART 通道；带有 8 个优先级的高级中断控制器；四组 32 位的 PIO 控制器可以实现高达 122 个可编程 I/O 引脚；20 通道的外设数据控制器(DMA)

⑤ 10M/100M base-T 以太网控制器模块

⑥ 2 个 USB 2.0 全速主机接口和 1 个 USB 2.0 全速从机接口

⑦ 多媒体卡接口(MCI)

⑧ 4 个通用同步/异步收发器(UART)，1 个两线接口(TWI)，1 个主从串行外围接口(SPI)，3 个同步的串行控制器(SSC)

⑨ 两个 3 通道 16 位时钟计数器

3.2 电源模块

整个系统需要多种电源，AT91RM9200 微处理器需要 3.3V 和 1.8V 两种电源供电。1.8V 用于给内核供电，3.3V 给外部总线接口的 I/O 口线和外设的 I/O 口线供电。FLASH 和 SDRAM 需要 3.3V 供电。复位模块中的 IMP811S 芯片需要 3.3V 供电。RS232 串行通信电路中的 MAX3232 芯片和 RS485 串行通信电路中的 MAX485 芯片都需要 3.3V 供电。以太网通信模块中的 DM9161 芯片需要 3.3V 供电。

本系统外部输入 5V 直流电源，使用 AMS1117- 3.3 和 AMS1117- 1.8 进行 DC-DC

变换分别得到 3.3V 电压和 1.8V 电压，最大输出电流均为为 800mA。经过测量本系统的 3.3V 和 1.8V 电源的实际工作电流均小于 50mA，因此电路完全可以满足系统的供电要求。

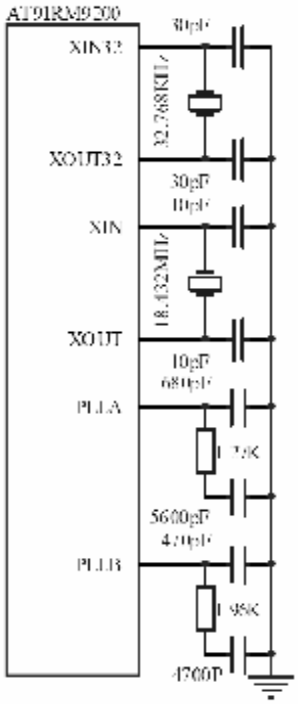


图 3-2 AT91RM9200 时钟模块

3.3 时钟模块

AT91RM9200 需要有两个时钟源，慢时钟源和主时钟源。慢时钟源采用 32.768KHz 的无源晶振。它与 CPU 的 XIN32 和 XOUT32 相连，主要用于 AT91RM9200 的启动和节能模式；主时钟源采用 18.432MHz 的无源晶振。它与 CPU 的 XIN 和 XOUT 相连，主要为 ARM920T 核与系统提供 180MHz 处理器工作时钟和 60MHz 的主时钟频率。AT91RM9200 需要两个 PLL 电路，分别与芯片的 PLLRCA、PLLRCB 相连。时钟模块的电路如图 3-2。

3.4 复位模块

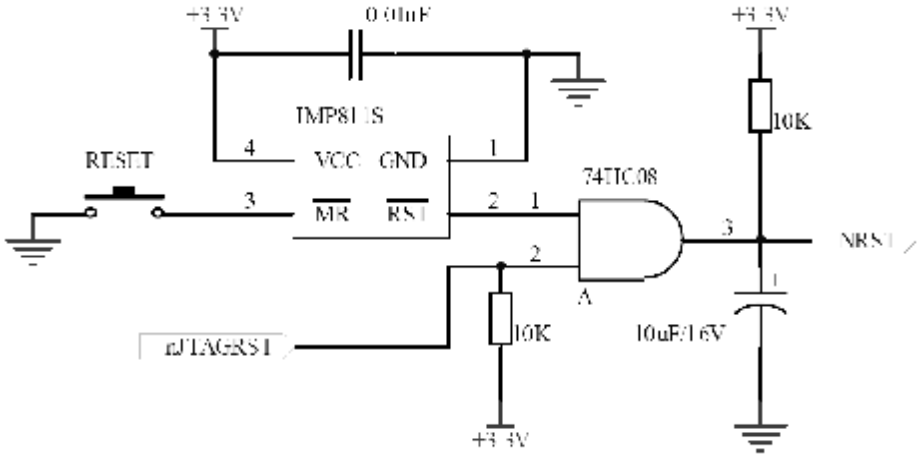


图 3-3 AT91RM9200 复位电路

AT91RM9200 的最短复位时间 92us[35]。为了使系统复位更加稳定，系统没有采用由电阻和电容构成的简单复位电路，而是采用低电压复位芯片 IMP811S。复位芯片一端接 3.3V 电源，一端接复位按钮，还有一端复位输出接 AT91RM9200 的 RESET 复位输入引脚。当电源电压低于门限电压 2.93V 或按压手动复位按钮时，IMP811S

复位输出引脚输出低电平信号进行复位，一直到电源电压高于门限电压 2.93V 或手动复位结束 140 毫秒以后，IMP811S 复位输出引脚才输出高电平，复位结束。复位模块的电路见图 3-3。

AT91RM9200 可以由 BMS 引脚的电平状态决定从片内 ROM 或者片外存储器引导。若引脚为高电平，从片内 ROM 引导。引脚为为低电平，从片外存储器引导。复位后，BMS 引脚可复用为 I/O 口。复位引导选择电路如图 3-4。

3.5 FLASH 和 SDRAM

因为 AT91RM9200 片内 ROM 和 RAM 不能满足系统的要求，所以本系统采用外扩存储器来扩展存储空间。软件系统中的操作系统代码、TCP/IP 协议栈、应用层程序和掉电后需要保存的数据都需要存放在非易失性存储器 FLASH 中。本系统选用 1 片 TSOP 封装的 8MB 容量的 NOR 型 FLASH：AM29LV640MB。AT91RM9200 从片外 FLASH 引导需要将 AT91RM9200 的芯片选择口线 NCS0 连接到 FLASH 的片选端 $\overline{CE}$。FLASH 输出使能端 $\overline{OE}$ 接 AT91RM9200 的 NOE 引脚；写使能端 $\overline{WE}$ 接 AT91RM9200 的 NWE 引脚；模式选择端 BYTE 接高电平，使 FLASH 工作在 16 位数据模式。FLASH 与 AT91RM9200 的基本连接如图 3-5 所示。

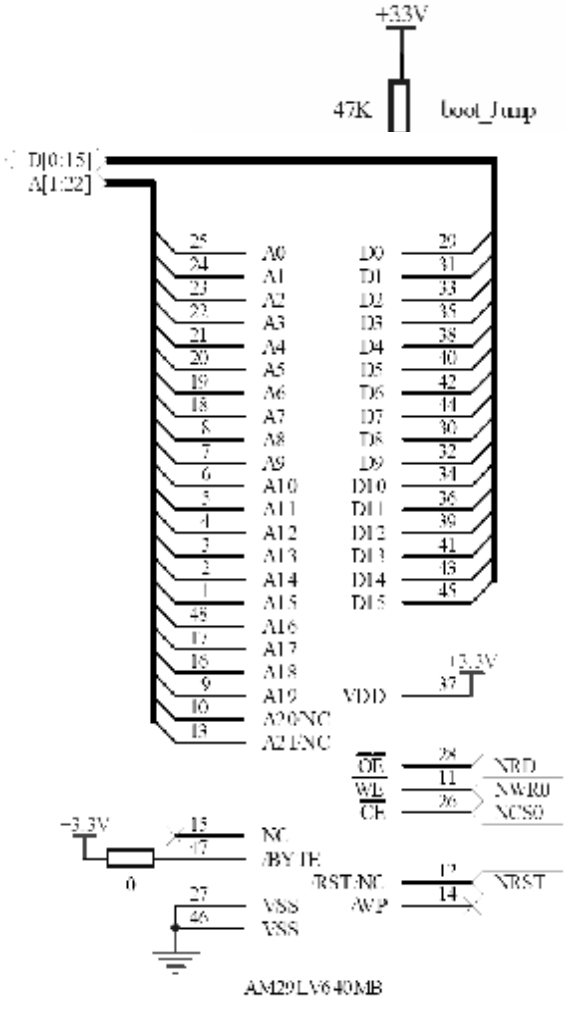


图 3-5 FLASH 与 AT91RM9200 的连接

实际上 TSOP 封装的 AM29LV640MB 的引脚与 TOSP 封装的 AM29LV160B 的引脚是兼容的。AM29LV640MB 有 8MB 的寻址空间，需要 22 根地址线，而

AM29LV160B 的存储容量较小为 2MB，因此只需要 20 根地址线，所以对前者的 A20 引脚和 A21 引脚不做功能定义。AM29LV160B 焊在 PCB 板上也能工作。因为这两种 FLASH 的区块大小不同，所以涉及 FLASH 擦写的操作的程序略有不同。在满足系统存储容量要求的情况下，采用小容量的存储芯片有可能降低成本。

系统启动后需要将 FLASH 中代码拷贝到 RAM 中去运行。AT91RM9200 片内集成 SDRAM 控制器因此本系统采用 2 片 Hynix 的 16MB 的 HY57V281620 芯片并联组成 32 位 32MB 的 RAM 空间，也可以采用 2 片 8MB 的 HY57V641620 并联组成 32 位 16MB 的存储空间，也能满足系统运行的需要。

系统用一片 SDRAM 作为高 16 位，数据线分别接 AT91RM9200 数据总线的 D[16：31]；另一片作为低 16 位，接数据总线的 D[0：15]。SDRAM 控制器对应片选信号为 NCS1，所以要将 NSC1 分别接到 2 片 DSRAM 的 nSCS 片选端。因为 SDRAM 的地址线 A10 具有预充电功能，所以 AT91RM9200 提供给 SDRAM 专用的地址线 SDA10 来代替通用地址线 A10。因为 32 位数据读写是 4 字节对齐的，所以 AT91RM9200 的地址线 A[2：11]、A[13，14]接 SDRAM 地址线 A[0：9]、A[11，12]。SDRAM 与 AT91RM9200 的连接如图 3-6 所示。

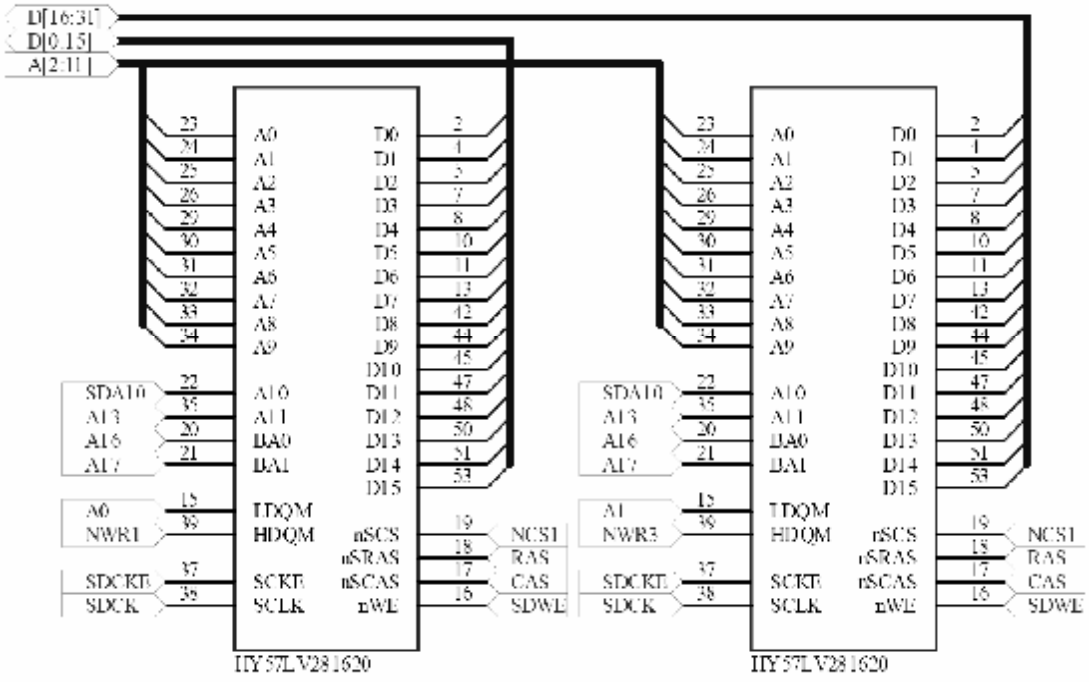


图 3-6 SDRAM 与 AT91RM9200 的连接

3.6 调试串口模块

AT91RM9200 的调试单元有一个串口调试模块，可用于多种调试与追踪目的，如观察芯片的启动，利用 XModern 协议向片内 RAM 下载用户程序，输入和输出调试信息等等。调试串口模块电路如图 3-7。

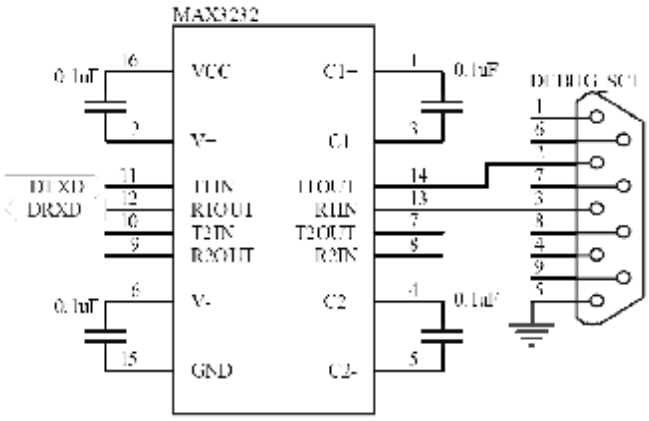


图 3-7 调试串口模块电路

该串口模块有 RXD、TXD 两个信号引脚，可以完成最基本的串行通信功能。RS232 标准所定义的高、低电平信号与 AT91RM9200 系统定义的高、低电平信号不同。AT91RM9200 的逻辑“1”对应 2V～3.3V 电平，逻辑“0"对应 0V～0.4V 电平。RS232 标准电平采用负逻辑，逻辑“1”对应-5V~15V 电平，逻辑“0”对应+5V~15V 电平。两者间通信必须进行电平的转换，因此在系统中采用了 MAX3232 芯片。

3.7 UART0 模块

AT91RM9200 有 4 个通用同步异步串行通信模块(UART)。在 UART 的 US_MR 模式寄存器的 USART_MODE 位置 1，可将 USART 设置为 RS485 通信模式。

在 RS485 模式下，通信接口需要使用 AT91RM9200 的接收串行引脚 RXD，发送串行引脚 TXD 和发送请求引脚 RTS。其操作与异步或同步模式下操作相同，并可对所有参数进行配置。不同之处在于当发送器工作时自动将 RTS 引脚拉高。RTS 引脚动作由 UART 中断使能寄存器的 TXEMPTY 位控制。RTS 引脚的电平与 UART 中断使能寄存器的 TXEMPTY 位的值相反。

本课题需要一个串行口链接 Modbus 串行通信链路，可以选用 UART0 模块并将其配置为 RS485 模式。RXD0 使用 AT91RM9200 的 PA18 引脚、TXD0 使用

AT91RM9200的 PD21 引脚。这里要注意的是 RTS0 引脚不能使用[35]上介绍的 PA21 引脚，否则会通信失败。为了解决这个问题笔者花费了很多时间，后来查阅了 ATMEL 公司公布的芯片勘误表后才发现这是 AT91RM9200 芯片本身的缺陷，只要用 PD21 引脚来代替 PA21 就可以了，具体电路参见图 3-8。

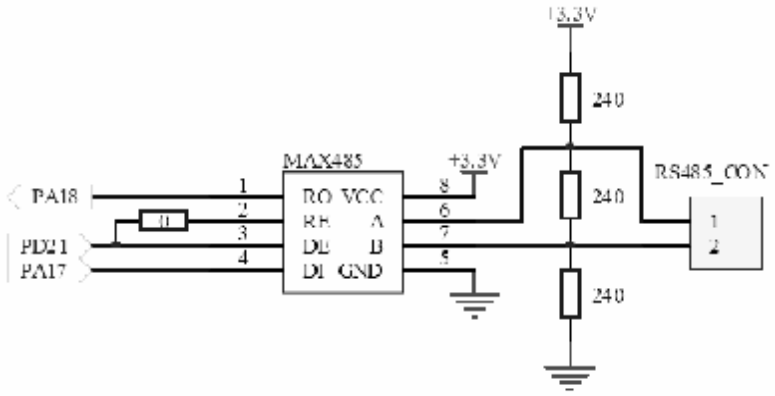


图 3-8 RS485 模式下 UART0 电路

为了方便开发和测试，在软件设计和调试的时候先将 UART0 的硬件电路和硬件驱动程序配置成 RS232 模式连接 PC 的 RS232 接口。待开发完毕后可将 UART0 再配置成 RS485 模式。RS232 模式下的 UART0 接口电路与图 3-7 相似。

3.8 以太网模块

AT91RM9200片内包含一个 EMAC 控制器。它通过 MII(媒体独立接口)或 RMII (精简媒体独立接口)方式与物理层芯片进行通信。MII 与 RMII 接口均可实现 IEEE802.3u 标准要求的 10Mbps 与 100Mbps 数据通信。RMII 与 MII 中使用相同信号引脚只不过 RMII 使用引脚更高效的映射这些信号，其中包括使用 50MHz 速率下的 2 位并行格式而不是 MII 的 25MHz 速率下的 4 位并行格式进行数据的发送和接收。

本系统选用了 DAVICOM 公司的 10/100Mbps 快速以太网物理层收发芯片 DM9161[38]。DM9161 是一款低功耗、高性能的 CMOS 芯片。它能够实现连接速度和协议选择的自动协商。它也能实现 IEEE802.3u 定义的物理层的全部功能。以太网数据信号最终通过网络隔离变压器和 RJ45 接口发送到网线上或者从网线上接收。以太网模块的连接如图 3-9。

系统选择使用 RMII 接口来传输数据，目的是减少引脚的连接数。使用 RMII

方式进行通信需要对 AT91RM9200 的 ETH_CFG 寄存器中的 RMII 位置 1。这里需要补充的一点是在对本系统的以太网部分绘制 PCB 电路板的时候一定要注意电源和地的铺设。因为以太网以 100MHz 速率进行数据传输，不恰当的设计会产生较多的电磁干扰，严重的会影响系统稳定工作，具体的铺设方法见[38]。

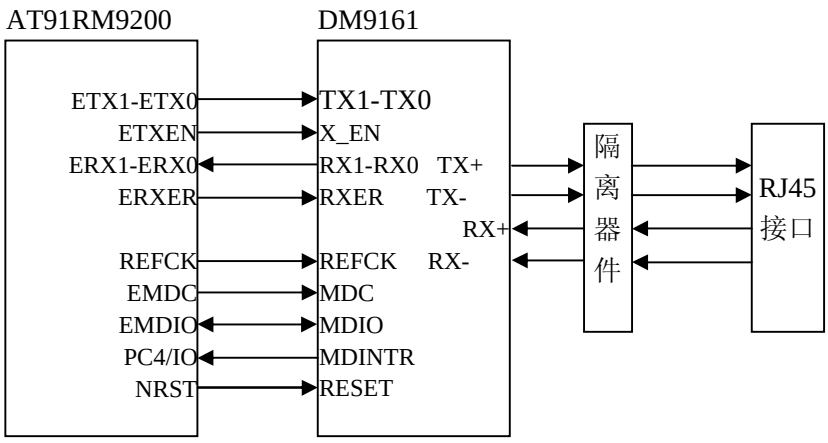


图 3-9 以太网物理层芯片 DM9161 的连接图

3.9 硬件系统测试

硬件系统测试流程如下[39]：

① 电源测试

焊接好两片 AMS1117 和电源插座以及滤波电容后，可以测量 AMS1117 的输入电压是否为+5V。输出电压是否为+3.3V 和+1.8V。

② 复位电路测试

焊接好 IMP811S 后，用万用表测量 NRST 引脚在复位按钮按下后是否为低电平。

③ 主芯片测试

AT91RM9200 以及调试串口电路焊接好以后，与 PC 机串口相连，打开超级终端新建一个连接，波特率为 115200、8 位数据位、1 位停止位、无奇偶校验、无流控制。将 AT91RM9200 的 BMS 引脚拉高，从片内 ROM 启动。观察超级终端，如果串口不停的接收到“C”说明 AT91RM9200 的片内 ROM 程序正在执行，它向串口发送“C”作为握手信号，并等待 PC 机通过 XModem 协议发送程序给它。

④ SDRAM 和 FLASH 测试

焊接好 SDRAM 和 FLASH 芯片后，可通过调试串行口将 SDRAM 和 FLASH

的测试的程序分别下载到硬件系统执行，程序负责对 SDRAM 和 FLASH 进行读写测试，自动判断写入数据是否与读出数据一致，运行结束将会给出测试结果。

⑤ 以太网和 UART0 模块测试

这两个模块的测试主要是使用 ATMEL 公司在网站上提供了测试程序样例。因为 ATMEL 公司的测试开发板与本硬件系统的 SDRAM 的芯片存在一定程度的差异，需要修改硬件系统初始化时 SDRAM 控制寄存器的配置。以太网的测试内容是实现了最简单的 ARP 和 ICMP 协议，UART0 测试内容是实现基本的串行数据的输入和输出。

3.10 硬件成本分析

表 3-1 是硬件系统主要芯片和元器件采购的零售价。如果批量采购的话售价会有很大幅度的下降。如果 SDRAM 和 FLASH 采用存储容量较小的芯片，如本章第 5 节提到的 HY57V641620 和 AM29LV160B，则硬件成本还有进一步下降的空间。

表 3-1 硬件成本分析

元器件名称	备注	数量	售价
AT91RM9200	主控芯片	1	RMB100
HY57V281620	SDRAM 芯片	2	RMB30×2
AMD29LV640B	FLASH 芯片	2	RMB40×2
DM9161	以太网物理层芯片	1	RMB20
MAX3232	RS232 电平转换芯片	1	RMB10
MAX485	RS485 电平转换芯片	1	RMB10
PCB 板制作	空白 PCB 板	1	RMB30
其他元器件	电阻、电容、电感、接插件等	若干	RMB40
			合计：RMB350

3.11 小结

本章给出了嵌入式 Modbus/TCP 网关硬件系统的整体结构、外围芯片的选型和各个模块的硬件设计电路；讨论了设计中遇到的困难和需要注意的问题；最后就硬件系统进行了测试和成本分析。

第四章 开发环境的建立与 μC/OS-II 的移植

在硬件系统实现之后就要进行软件系统的开发。根据第二章的系统总体设计方案，软件系统的开发分成三部分：μC/OS-II 操作系统的移植、TCP/IP 协议栈的实现和应用层程序的设计。本章主要阐述第一部分 μC/OS-II 操作系统的移植。在移植之前首先介绍了如何建立一个良好的软件开发环境，可以方便的对程序进行编译和下载。然后设计 μC/OS-II 操作系统的初始化程序，最后移植 μC/OS-II 操作系统包括修改与硬件相关的代码、修改与应用相关的代码和设计时钟节拍初始化代码。μC/OS-II 操作系统的工作原理和实现机制不是本文研究的重点故不在此敷述。

4.1 开发环境的建立

4.1.1 ARM 集成开发工具 ADS

选择合适的 ARM 处理器的开发工具可以加快嵌入式系统的开发进度、节约成本和提高效率。CodeWarrior for ARM Developer Suite v1.2 集成开发环境(以下简称 ADS)是 ARM 公司推出的新一代 ARM 集成开发工具。它取代了 ARM 公司先前推出的 ARM SDT 开发工具。它是一种快速而节约成本的完整软件开发解决方案，可以在 PC 上实现代码编辑、代码编译、链接、调试、项目管理等功能。

它替换并增强了 SDT 的一些模块和组成部分，其中颇具特色的一点就是它包含了 CodeWarrior IDE 集成开发环境，即包含了项目管理器、代码生成接口、语法敏感编辑器、源文件和类浏览器、源代码版本控制系统接口以及文本搜索引擎等。

ARM 公司凭借自身在技术上的优势对 ARM 系列每一种体系结构都进行了专门的优化处理，声称其代码生成工具最终生成的可执行文件最小可以比其他公司的开发工具生成的文件小 20％。

4.1.2 引导程序 Bootloader

引导程序 Bootloader 是系统上电复位后执行的程序，进行系统的初始化也可加载真正的应用程序或操作系统。因为 AT91RM9200 片内的 SRAM 仅有 16K，所以

引导程序的长度受到限制，不可能完成更多的功能，例如对 FLASH 编程、实现控制台命令、引导操作系统等。因此Bootloader 不仅包括一级引导程序 bootflash 用来实现系统的初始化准备，还包括一个强大的二级引导程序 U-Boot(Universal Boot Loader)用来完成 bootflash 没有完成的事情如载入用户程序等。

AT91RM9200在上电或复位前的最后两个时钟周期内，芯片的第 79 脚 BMS 的电平状态决定了引导存储器的类型，若为高电平，则执行 ATEML 公司固化在片内 ROM 中的引导程序，等待从调试串口下载程序。若为低电平，则将片外 FLASH 地址 0x10000000 开始的代码拷贝到内部 SRAM 中执行。bootflash 就是从 FLASH 载入 SRAM 中执行的。该程序实现的功能有：初始化微处理器、初始化 SDRAM 控制器、解压 0x10010000 地址处的 U-Boot 压缩镜像到 SDRAM 的 0x21f00000 地址处执行。

二级引导程序 U-Boot 是遵循 GNU GPL 条款而开发的开放源代码项目。U-Boot 被广泛地应用到嵌入式产品的开发中。其源代码开放，支持多种嵌入式操作系统内核和微处理器，支持控制台命令，功能设置灵活，便于进行调试、引导不同的操作系统和产品发布，驱动代码丰富。

本系统主要利用 U-Boot 强大的串口控制台命令，实现目标板程序的下载、FLASH 的烧写，以及操作系统和应用程序的搬移和引导。关于 AT91RM9200 Bootloader 引导程序移植和固化的详细方法请参考[39]。

4.1.3 开发调试方法

嵌入式系统是针对应用定制的系统，其资源非常有限，因此其开发过程和开发环境与基于 PC 机的软件开发有明显的不同。首先程序在宿主机(PC 机)上编写，然后通过编译、链接工具生成可在目标板(嵌入式 Modbus/TCP 网关硬件系统)上执行的二进制代码，最后将二进制代码从宿主机下载到目标板上运行。开发调试环境如图 4-1。

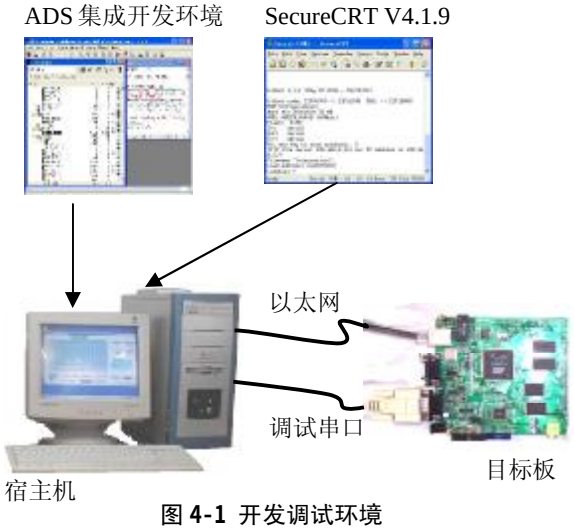


图 4-1 开发调试环境

本系统的宿主机运行 Windows XP 操作系统。它通过调试串口和以太网接口与目标板进行通信。编译、链接工具均采用 ADS 自带的工具。目标板程序的下载，FLASH 的烧写，以及 μC/OS-II 操作系统和应用层程序的引导都依赖于固化在目标板中的 Bootloader 引导程序中的 U-Boot。

在软件系统开发的时候，首先通过运行驻留在目标板中的 U-Boot 程序，从宿主机向目标板的调试串口发送 U-Boot 控制台命令，让目标板从宿主机的 TFTP 服务器上下载操作系统和应用层程序的可执行二进制代码 lwipucos.bin 文件到目标板的 SDRAM 中，然后再发送 U-Boot 命令让目标板执行下载的代码。具体的 U-Boot 操作命令如图 4-2。在软件开发的时候还可以利用 SecureCRT 串口调试工具可以从调试串口观察程序运行的情况这一方法，通过在程序中设置串口输出信息从而对程序进行调试。

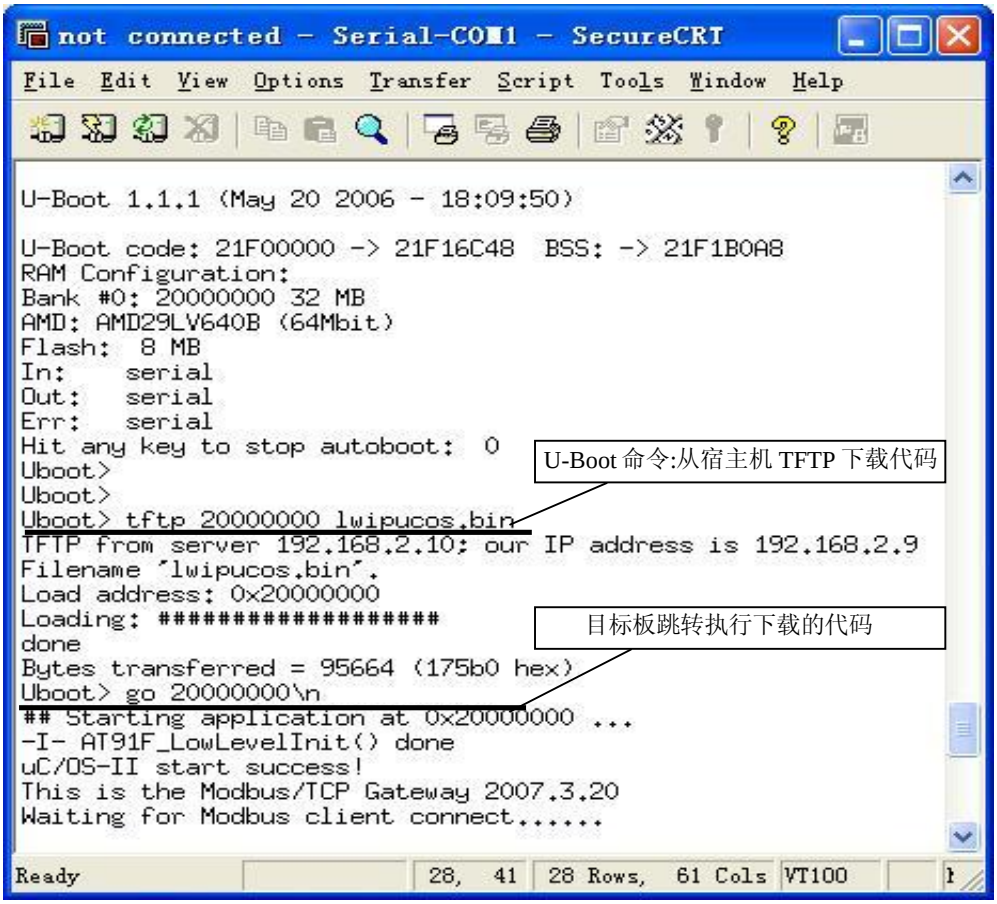


图 4-2 利用 U-Boot 从 TFTP 下载代码

这里之所以选择从宿主机的 TFTP 服务器而不是调试串口下载代码是因为网络传输速度大大快于调试串口的传输速度。二进制可执行文件 lwipucos.bin 的大小约

为 100KB。如果用串口下载该文件，波特率设置为 115200bps，一次下载需时约 9 秒，而用网络下载该文件需要的时间不到 1 秒。

在软件系统开发结束的时候要用 U-Boot 命令将最终的操作系统和应用层程序的可执行二进制代码文件 lwipucos.bin 烧写到目标板的 FALSH 区，以后每次目标板进行启动复位时，固化的 Bootloader 程序会将 FLASH 中 lwipucos.bin 自动拷贝到 SDRAM 中然后从 SDRAM 运行。烧写过程需要使用的 U-Boot 命令如下：

```
uboot>erase 1:103                                  擦除 0x10600000 地址开始的 2 个 FLASH 块
uboot>erase 1:104
uboot>tftp 20000000 lwipucos.bin                   从 TFTP 下载最终的操作系统和应用程序代码
uboot>cp.b 20000000 10600000 1FFFF                 将下载的代码烧写到 FLASH 中
uboot>setenv bootcmd cp.b 10600000 20000000 1FFFF\;go 20000000
                                                   设置 U-Boot 自启动命令
uboot>saveenv                                      保存设置内容
```

之所以将 FLASH 中的代码拷贝到 SDRAM 中运行而不直接在 FLASH 中运行不仅是因为系统访问 RAM 的速度要快于访问 FLASH 的速度，更重要的是只有当程序运行在 RAM 区时才能对 FLASH 区进行数据的烧写，同时把一些在程序运行的过程中得到的信息如用户设置的网关地址保存到非易失性存储介质上，使之掉电后仍然能保存。各种程序在 FLASH 中的分布见表 4-1。

表 4-1 片外 FLASH 中数据的分布

FLASH 块号	对应的地址	存放的内容
0	0x10000000~0x10003FFFF	Bootloader 一级引导程序：bootflash.bin
9	0x10010000~0x1000FFFF	Bootloader 二级引导程序：U-Boot.bin
87	0x10500000~0x1050FFFF	前 4 个字节存放的 Web 的表单页面文件的长度之后存放页面的内容
103	0x10600000~0x1060FFFF	μC/OS-II 操作系统和应用层程序的代码 lwipucos.bin
104	0x10601000~0x1061FFFF	μC/OS-II 操作系统和应用层程序的代码 lwipucos.bin
119	0x10700000~0x1070FFFF	网关的网络配置信息文件 netcfg.bin

4.2 ARM 体系结构的基本概念

在设计 μC/OS-II 操作系统的初始化代码和移植 μC/OS-II 操作系统之前需要了解一些关于 ARM 体系结构的基本概念。

① 处理器状态

ARM 微处理器的工作状态一般有两种，一种是 ARM 状态，执行 32 位字节对齐的 ARM 指令；另一种是 Thumb 状态，处理器执行 16 位半字对齐的 Thumb 指令。本系统运行在 ARM 状态。

② 处理器模式

ARM 处理器支持 7 种处理器模式：用户模式、快速中断模式、外部中断模式、管理模式、数据访问中止模式、未定义指令中止模式、系统模式。除用户模式和系统模式之外的 5 种模式都称为异常模式。当特定的异常出现时，处理器进入了相应的模式。每种模式都有某些附加的寄存器，以避免异常退出时用户模式的状态不可靠。

大部分应用程序都运行在用户模式下。在用户模式下系统的某些资源是不允许访问的。异常模式可以自由的切换到其他处理器模式，而用户模式不能直接切换到别的模式。

系统模式不能由任何异常而进入该模式，它与用户模式有完全相同的寄存器，但资源的访问不会受到用户模式的限制。ARM 微处理器的运行模式可以通过软件改变，也可以通过外部中断或异常处理改变。

③ 寄存器组织

ARM 微处理器共有 37 个 32 位的寄存器，其中 31 个为通用寄存器，6 个为状态寄存器。这些寄存器不能同时被访问，具体有哪些寄存器可以被访问取决于微处理的工作状态和运行模式。在任何时候，15 个通用寄存器 R0～R14、程序计数器 PC、状态寄存器 CPSR 都可以被访问的[40]。

4.3 启动 μC/OS-II 的初始化程序

μC/OS-II 操作系统运行前需要对硬件系统的片内和片外存储器、系统堆栈等进行初始化。这些工作与硬件系统相关，难以用 C 语言来实现，需要用汇编语言实现。系统在执行完初始化程序之后就将系统的软硬件环境带到一个合适的状态，为启动 μC/OS-II 准备好正确的环境，最后通过跳转指令执行 μC/OS-II 主函数 main 中的第一条 C 语句[41]。

初始化程序的设计不属于 μC/OS-II 操作系统的移植但是其质量的好坏关系到整个系统程序的运行。初始化程序会用到一些伪指令，伪指令的格式和用法是与编

译器相关的，使用 ASD 自带的编译器一定要参考 ADS 的技术手册。

初始化程序的执行步骤如下：

① 定义程序的入口点

初始化程序必须定义整个程序的入口点。配合 ADS 链接器的设置，ADS 的伪指令 Entry 标识了整个程序的入口点。

② 设置异常向量

AT91RM9200 系统上电复位后，PC 就被初始化为 0x0，程序从地址 0x0 开始执行。此时要求片外非易失性存储器 FLASH(或片内的 ROM)的起始地址为 0x0，以便能访问到正确的代码。因为FLASH 的访问速度低于 RAM，若每次产生异常后都要从 FLASH 的异常向量表跳转到相应的处理程序则会大大影响异常的响应速度。因此在复位时 AT91RM9200 自动把异常向量从地址 0x0 拷贝到片内 RAM，然后执行地址重映射，把片内 RAM 空间映射到 0x0 地址处，而将原来位于 0x0 处的片外 FLASH 映射到地址 0x10000000，片内 ROM 则映射到地址 0x00100000。这一过程虽然不容易理解但是非常重要。异常向量的设置如下：

```
;ENTRY 定义了程序的入口地址
        ENTRY
;Reset_Handler 为一个标号，标识重映射之后的复位处理代码的入口点
        B       Reset_Handler                   ；处理器复位
undefvec    B       undefvec                    ；未定义指令
swivec      B       swivec                      ；软中断
pabtvec     B       pabtvec                     ；预取指令异常
dabtvec     B       dabtvec                     ；数据异常
rsvdvec     B       rsvdvec                     ；保留
irqvec      ldr     pc, [pc,#-0xF20]            ；外部中断
fiqvec      ldr     pc, [pc,#-0xF20]    ;      快速中断
Reset_Handler
```

③ 执行硬件初始化

1) 屏蔽看门狗并禁止所有中断，避免初始化过程被意外中断

2) 在等待 256 个慢晶振时钟周期后开启主时钟晶振

3) 执行一小段循环，等待主时钟晶振工作稳定

4) 进入管理模式，依次对 PLLA，PLLB 初始化

5) 配置主系统时钟和处理器工作时钟，使他们分别工作在 60MHz 和 180MHz

6) 配置存储寄存器域 0 的寄存器，即提高片外 FLASH 的访问速度

7) 配置存储寄存器域 1 的寄存器(SDRAM 控制寄存器)，使 SDRAM 可被访问

8) 配置高级中断控制器

9) 初始化调试串口单元

④ 在 CP15 控制寄存器中，将 AT91RM9200 配置成小端模式

⑤ 初始化堆栈指针寄存器 SP。ARM 微处理器支持七种处理器模式。处理器每种模式的堆栈初始化过程就是将处理器切换到相应的运行模式再给 SP 寄存器定义一个堆栈地址，其中只有系统模式和用户模式共用一个堆栈指针。

⑥ 开放系统中断

⑦ 初始化 C 变量

一个完整的 ARM 应用程序经过 ADS 编译以后会产生三个输出段 RO，RW 和 ZI(如果使用另一种常用的 ARM 开发工具 GNU 工具对应的就是 TEXT，DATA 和 BSS 段)，其中 RO 为代码段，RW 是已初始化的全局变量，ZI 是未初始化的全局变量。

可执行映像文件最初总是存储在 FLASH(或其他类型的 ROM)中。其中的 RO 段既可在 FLASH 中运行，也可在高速的 RAM 中运行。RW 段和 ZI 段必须在可读写的 RAM 中才能正常工作[40]。

本系统开发的时候，操作系统和应用层程序的可执行代码文件 lwipucos.bin 会被 U-Boot 全部拷贝到 SDRAM 中执行，如图 4-3。在 ADS 的链接器中设置 RO 段起始地址是 0x20000000，RW 段和 ZI 段的地址紧接在 RO 段之后，因此 RW 段就不用编写程序进行搬移，但 ZI 段是需要编写程序对其在 SDRAM 中存储区域清零。

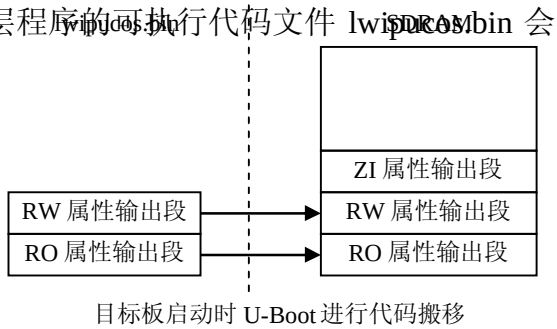


图 4-3 映像文件 lwipucos.bin 中段的搬移

笔者发现在 ADS 中 ZI 段不仅包括未初始化的全局变量和未初始化的静态局部变量也包括了初始值为 0 的全局变量和静态局部变量。如果 ZI 段不清 0，按照[34]提供的 μC/OS-II 测试方法也能测试通过但实际上初始值为 0 的全局变量和静态局部变量都没有被清零，在编写应用层程序的时候就出现了严重的问题，给开发工作造成很大的影响。初始化 C 变量的代码如下：

```
;r1、r2、r3、r4 分别指向 RO 输出段运行时存储区域界限、RW 输出段运行时起始地址、
;ZI 输出段运行时起始地址和 ZI 输出段运行时存储区域界限
    add     r2, pc,#-(8+.-CInitData)
    ldmia   r2, { r3, r4}
    mov     r2, #0      ;r2                     清 0
LoopZI
    cmp     r3, r4          ; r3                和 r4 比较
;如果 r3 指向的地址没有超出 ZI 输出段存储区域限界，则对 r3 指向的地址单元清 0，
;并使 r3 指向下一个地址。
    strcc   r2, [r3], #4
 bcc    LoopZI          ;                   跳转 LoopZI
 b      EndInitC        ;                   清 0 完毕
CInitData
;IMPORT 伪指令引入 ADS 链接器定义了的一些外部符号，这些符号中都包含字符$$。
;ARM 链接器再生成映象文件时，用它们来代表映象文件中各段的起始地址以及
;存储区域界限。
    IMPORT  |Image$$ZI$$Base|
    IMPORT  |Image$$ZI$$Limit|
    DCD     |Image$$ZI$$Base|   ; ZI        输出段运行时起始地址
    DCD     |Image$$ZI$$Limit|  ; ZI        输出段运行时存储区域界限
EndInitC
```

⑧ 运行 μC/OS-II 操作系统

完成系统的初始化之后就可以运行 μC/OS-II 操作系统了。这里推荐用 main( ) 函数作为 μC/OS-II 的入口函数，因为ADS 自带的 ARM 调试工具 AXD 默认 main( ) 函数为 C 程序的调试入口。跳转到 main( )函数的代码如下：

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

4.4 μC/OS-II 操作系统的移植

μC/OS-II 在最初设计的时候就已经考虑到了系统的可移植性。μC/OS-II 90％以上的代码都是用 C 语言编写的，但要把 μC/OS-II 移植到不同体系结构的处理器乃至同一种体系结构而具体型号不同的处理器上仍需要使用 C 语言和汇编语言共同完成一些与处理器相关的操作。移植的绝大部分工作集中在多任务切换的实现上。这部分代码主要是进行处理器状态的保存、切换、恢复，主要用汇编语言来实现[42]。

要使 μC/OS-II 在目标处理器上正常运行需要满足以下条件：第一，处理器的 C 编译器能够产生可重入代码。第二，在程序中可以打开或者关闭中断。第三，处理器支持中断，并且能产生定时中断。第四，处理器支持能够容纳一定数据的硬件堆栈。第五，处理器有将堆栈指针和其他 CPU 寄存器读出和存储到堆栈(或者内存)

的指令。AT91RM9200微处理器和 ADS 集成开发工具完全能满足这些条件。

课题采用 μC/OS-II v2.52 版本的内核，其全部源代码量大约是 6000～7000 行，共 16 个文件。移植到 ARM 处理器上，主要修改 3 个与 ARM 体系结构相关的文件：os_cpu.h、os_cpu_c.c 和 os_cpu_a.s，代码量大约是 500 行，另外还要添加 μC/OS-II 系统时钟节拍的代码。

图 4-4 给出了 μC/OS-II 的软件结构。应用层程序是开发者根据应用系统实现的功能编写的代码。μC/OS-II 与应用相关的代码是开发者通过调用内核服务功能，对 μC/OS-II 进行的裁剪。与处理器无关的代码是 μC/OS-II 操作系统的内核，它为 μC/OS-II 提供所有的系统服务，并将应用层程序和底层硬件联系起来而成为一个实时系统。要使 μC/OS-II 内核能移植到不同的处理器及硬件平台上，还需要在内核和底层硬件之间添加与处理器和硬件平台相关的代码，部分代码需要用汇编语言编写。

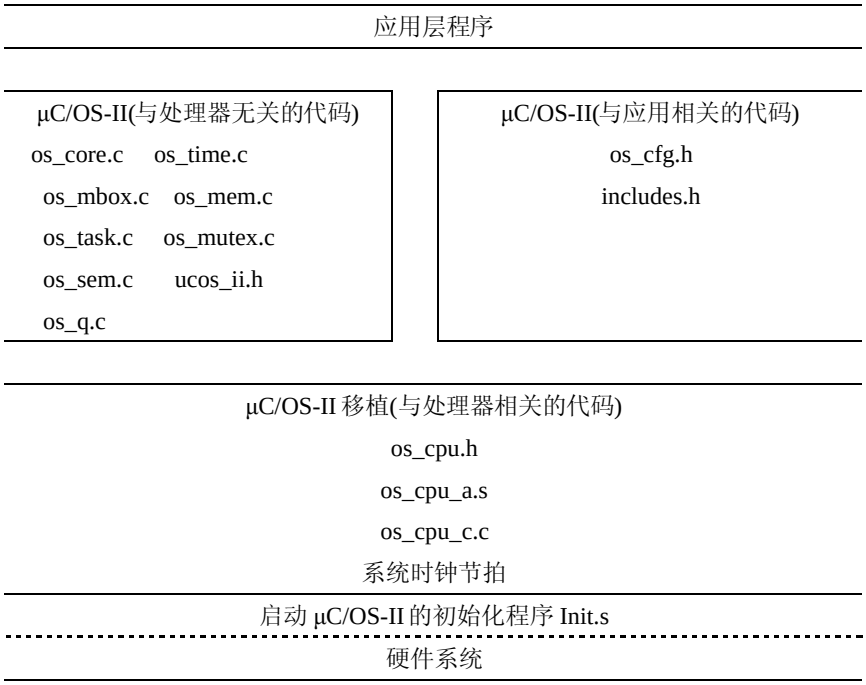


图 4-4 μC/OS-II 的软件结构

4.4.1 os_cpu.h 文件的修改

① 数据类型定义

由于不同的处理器有不同的字长，μC/OS-II 的移植需要重新定义一系列数据类型，保证其可移植性。μC/OS-II 的代码从不直接使用 C 语言的 short、int 和 long 等

数据类型，因为它们是与编译器相关的，不可移植。

ADS中的编译器定义整型(int)数据为32位，短整型(short)为16位，字符型(char)为8位。由于处理器在任务切换时，所有现场寄存器都将保存在当前运行任务的堆栈中，所以堆栈数据类型OS_STK应该与处理器的寄存器长度一致均为32位，且所有与任务相关的堆栈都必须用OS_STK声明。同理，状态寄存器类型也须设置为32位。数据类型定义如下：

```
typedef unsigned  char  BOOLEAN;   //      布尔类型
typedef unsigned  char  INT8U;      //8        位无符号整形
typedef signed    char  INT8S;      //8        位有符号整形
typedef unsigned  short INT16U;     //16       位无符号整形
typedef signed    short INT16S;     //16       位有符号整形
typedef unsigned  int   INT32U;     //32       位无符号整形
typedef signed    int   INT32S;     //32       位有符号整形
typedef unsigned  int   OS_STK;     //       堆栈入口地址均为32位宽度
typedef unsigned  int   OS_CPU_SR;  //CPU      状态寄存器均为32位宽度
```

② 代码临界区

与所有实时内核一样，μC/OS-II需要先禁止中断再访问代码的临界区，并在访问完毕后重新允许中断。这就使得μC/OS-II能够保护临界区代码不会受到多任务或中断服务程序的破坏。OS_ENTER_CRITICAL( )和OS_EXIT_CRITICAL( )分别实现开中断和关中断。在这里采用宏定义的格式，最后汇编代码MOV PC，LR实现从临界区返回。代码临界区定义如下：

```
#define  OS_ENTER_CRITICAL( )  (cpu_sr = OS_CPU_SaveSR( ))
#define  OS_EXIT_CRITICAL( )   (OS_CPU_RestoreSR(cpu_sr))
;这两个宏定义的具体内容在汇编程序 os_cpu_a.s 中。
    EXPORT  OS_CPU_SaveSR
    EXPORT  OS_CPU_RestoreSR
OS_CPU_SaveSR
    MRS    R0,CPSR          ;          复制当前程序状态寄存器的值
    ORR    R1,R0,#NO_INT    ;0xC0          屏蔽中断位
    MSR    CPSR_c,R1    ;              关闭所有中断
    MOV    PC,LR            ;          恢复PC，返回
OS_CPU_RestoreSR
    MSR    CPSR_c,R0    ;              恢复原来的程序状态寄存器的值
    MOV    PC,LR    ;                  恢复PC，返回
```

③ 堆栈生长方向

在μC/OS-II中可以配置堆栈的生长方向，绝大多数的微处理器堆栈是从高地址向低地址生长的。ARM处理器也是这种情况，所以定义如下：

```
#define  OS_STK_GROWTH       1
```

4.4.2 os_cpu_c.c 文件的修改

该文件包含了 10 个简单的 C 语言函数；OSTaskStkInit( )、OSInitHookBegin( )、OSInitHookEnd( ) 、OSTaskIdleHook( ) 、OSTaskDelHook( ) 、OSTCBInitHook( ) 和 OSTaskCreateHook( )、OSTimeTickHook( )、OSTaskSwHook( )、OSTaskStatHook( )。如果不需要做系统扩展的话，这些函数都可以直接设置为空。

在本系统 中需要初始 化中 断级 切换 标志 OSIntCtxSwFlag，将 该代码放在 OSInitHookBegin( )函数中，这样在运行 OSInit( )函数时就会执行中断切换标志的初始化，代码会更加简洁明了。在文件 os_cfg.h 中需要将常量 OS_CPU_HOOKS_EN 置 1。OSInitHookBegin( )函数中的代码如下：

```
#if OS_CPU_HOOKS_EN > 0 && OS_VERSION > 203
void  OSInitHookBegin (void)
{
   OSIntCtxSwFlag = 0;
}
#endif
```

μC/OS-II 的移植要编写任务堆栈初始化函数 OSTaskStkInt( )。任务堆栈空间由高地址至低地址依次保存着 PC、LR、R12～R0、CPSR、SPSR 寄存器的值。当用户建立任务时，系统内部调用任务创建函数 OSTaskCreate( )或 OSTaskCreateExt( )，然后调用函数 OSTaskStkInt( )来初始化任务的堆栈空间，并返回新的任务堆栈指针 SP。最后调用任务控制块初始化函数 OSTCBInit( ) 将返回的任务堆栈指针 SP 保存到该任务的 TCB 块中。堆栈看起来就像刚发生中断后所有寄存器保存到堆栈中的情形一样。任务创建后并不直接运行，而是要被任务调度函数 OSSched( )调度并满足执行条件后才能获得运行。

4.4.3 os_cpu_a.s 文件的修改

在 os_cpu_a.s 文件中主要修改以下四个函数：

① OSStartHighRdy( ) 函数

OSStartHighRdy( )在多任务系统启动函数 OSStart( )中被调用。该函数仅仅在多任务启动时被执行一次。它负责从最高优先级任务的任务控制块 TCB 中获得该任务的堆栈指针 SP，然后依次将 CPU 现场恢复，最后最高优先级的任务就开始执行，整个系统开始工作。

② OSCtxSw( ) 函数

OSCtxSw( ) 函数用来进行任务级的上下文切换。它被内核中的任务调度函数OSSched( )调用。该函数在当前任务被阻塞(如调用了OSTimeDly( )、OSSemPend( )函数)或者有更高优先级的任务需要被调度时保存当前任务的CPU状态，然后取得更高优先级任务的堆栈指针并从中恢复 CPU 现场，实现任务的调度。因为它工作在非异常模式，因此区别于中断级的上下文切换。

③ OSTickISR( ) 函数

OSTickISR( ) 函数是系统时钟节拍中断服务函数。它主要负责处理时钟中断。系统有一个周期性的节拍中断，如果在时钟中断的时候存在就绪态的高优先级任务，则执行中断级的上下文切换函数 OS_IntCtxSw( )，调度高优先级的任务。OSTickISR( )函数就不再返回，否则要在OSTickISR( )函数的末尾进行退栈操作，使中断模式下的堆栈保持平衡。系统时钟节拍频率越高，系统的实时性越高，负荷就越重。

④ OS_IntCtxSw( ) 函数

在μC/OS-II中，系统时钟节拍中断服务函数OSTickISR( )的末尾会调用中断退出函数 OSIntExit( )，检查是否有高优先级的任务等待就绪，如果有则执行OS_IntCtxSw( ) 函数调度高优先级任务执行，不再返回被中断的任务。这样可以让高优先级的任务响应速度比较快，提高系统的实时性能。中断级的上下文切换函数OS_IntCtxSw( )的基本原理与任务级的切换函数OSCtxSw( )相同，所以函数的后半部分可以直接调用OSCtxSw( )函数中的代码。

在中断状态下进行任务切换要注意当进入中断模式后，原来任务(用户模式下)的SPSR状态寄存器的值、任务的返回地址和部分寄存器的值无法在中断模式下获得，必须从中断模式切换到管理模式中去保存这些地址和寄存器的值。这对任务切换很重要。

4.4.4 包含 includes.h 文件

includes.h是主要的头文件。在所有后缀名为.C的文件的开始都包含includes.h文件。使用includes.h的好处是所有C文件都只包含一个头文件，程序简洁，可读性强。缺点是C文件可能包含一些它并不需要的头文件，额外的增加编译时间。与

上述优点相比，多花费一些时间还是可以接受的。

4.4.5 时钟节拍初始化代码

μC/OS-II 的移植还要编写时钟节拍初始化代码。时钟节拍初始化代码需要根据具体的硬件系统自行编写。本硬件系统采用的 AT91 系列微控制器的定时器/计数器模块有 3 个相同的 16 位定时器/计数器。每个通道有三个外部时钟输入、五个内部时钟输入及两个可由用户配置的多功能输入/输出信号可供选择。每个通道驱动一个可编程内部中断信号来产生处理器中断。本系统利用 AT91RM9200 片内定时器 1 提供周期性的系统时钟节拍中断。在这里选用了分频后的系统内部时钟做为定时器的输入。时钟节拍初始化需要对 AIC 高级中断控制器进行设置，具体代码如下：

```
void TC1_init (void) {
  unsigned int dummy;
  *AT91C_TC1_CCR=AT91C_TC_CLKDIS;   // 禁止中断计数器
  *AT91C_TC1_IDR=0xFFFFFFFF;        //       禁止时钟计数器的所有的中断
  dummy=*AT91C_TC1_SR;              //       清除时钟状态寄存器
//选定主时钟频率 MCK/128 为定时器输入时钟，在系统硬件初始化时 MCK 设置为 60MHz
  *AT91C_TC1_CMR=AT91C_TC_TIMER_DIV4_CLOCK|AT91C_TC_CPCTRG;
  *AT91C_TC1_CCR=AT91C_TC_CLKEN;   // 时钟计数器使能
  *AT91C_TC1_IER=AT91C_TC_CPCS;    //  开启时钟 RC 寄存器比较产生中断
  *AT91C_AIC_IDCR=(1<<AT91C_ID_TC1);  // 在 AIC 中禁止定时器时钟的中断
  *AIC_SVR18=(unsigned int)OSTickISR;    //  在 AIC 中添加定时器中断处理函数入口地址
//在 TC1 的中断源模式寄存器中设置中断触发方式为脉冲触发(非电平触发)
//以及它在高级中断控制器中的优先级 4，(0 为最低，7 为最高)
  *AIC_SMR18=(AT91C_AIC_SRCTYPE_INT_EDGE_TRIGGERED|0x4);
  *AT91C_AIC_ICCR=(1<<AT91C_ID_TC1);  // 在 AIC 中断清除命令寄存器中清除中断
  *AT91C_AIC_IECR=(1<<AT91C_ID_TC1);  // 在 AIC 中断使能命令寄存使能中断
  *AT91C_AIC_IMR =(1<<AT91C_ID_TC1);  // 在 AIC 中断屏蔽寄存器使能中断
//设置比较寄存器 RC 的值，当定时器内部计数器的值与 RC 的值相等时，定时器触发中断
//定时中断发生的频率＝定时器输入时钟/TC1_RC=60MHz/128/0x124f=100Hz ，即 10ms 发生
//一次定时中断
  *AT91C_TC1_RC=0x124f;
  *AT91C_TC1_CCR=AT91C_TC_SWTRG;  // 定时器 TC1 开始计数
}
```

最后不要忘记在 μC/OS-II 的 os_cfg.h 文件中设置系统每秒节拍数。μC/OS-II 要根据这个参数来计算时分秒的长度。因为系统定时器 1 设置为每 10ms 发生一次系统时钟中断，所以宏 OS_TICKS_PER_SEC 定义为 100。

4.5 μC/OS-II 的测试

在完成 μC/OS-II 的移植之后就要验证移植代码正确与否。在此编写了一个测试

程序。在测试程序中创建了3个任务：TASK1、TASK2、TASK3，每个任务被分配不同的优先级。各个任务被周期性调度执行并从AT91RM9200的调试串口输出本任务的名称。利用串口监控软件SecureCRT可以在宿主机上观察到目标板上μC/OS-II操作系统多任务的运行情况，观察结果如图4-5。结果证明μC/OS-II多任务的创建、多任务的启动、任务级的切换、中断级的任务切换都是正确的，移植成功。

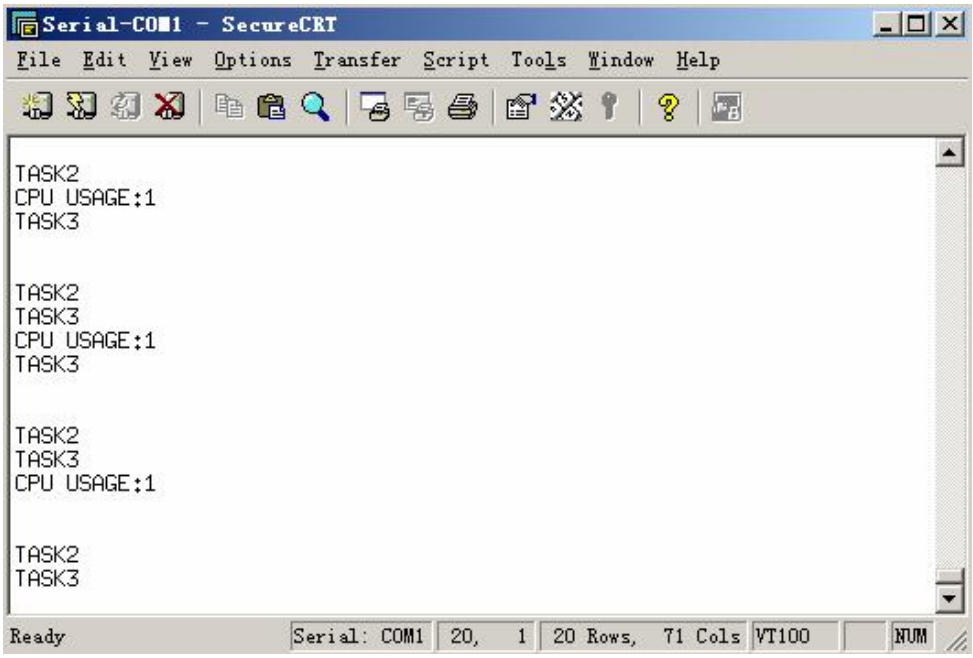


图4-5 μC/OS-II操作系统多任务的运行测试

4.6 小结

本章针对嵌入式系统开发的特点，提出了通过运行固化在硬件系统中的Bootloader程序从宿主即机的TFTP服务器下载并执行经过编译的可执行代码的程序调试方法，速度快、使用方便。在此基础上设计了启动μC/OS-II的初始化代码和时钟节拍初始化程序并移植了μC/OS-II操作系统。在这过程中笔者体会到移植μC/OS-II操作系统不仅要求对μC/OS-II本身有一定了解而且对所选用的编译链接工具、ARM体系结构乃至芯片具体的硬件模块要有清晰的认识，否则会在细节上出现问题而造成开发进度的延误。

第五章 嵌入式 TCP/IP 协议栈的实现

根据本文第二章的系统的总体设计，本章将进行软件系统开发的第二部分即在已经搭建好的软件开发调试环境和 μC/OS-II 操作系统的基础上实现网络通信协议栈。考虑到 TCP/IP 协议栈比较复杂、实现难度较大，所以课题没有专门设计 TCP/IP 协议栈。考虑到商业 TCP/IP 协议栈价格昂贵有的仅提供库函数调用并不开放源代码，所以本课题在对几种开放源代码的 TCP/IP 协议栈进行比较以后将选用合适的协议栈移植到 μC/OS-II 操作系统和硬件系统上。

5.1 计算机网络与 TCP/IP 协议栈

计算机网络是将在地理上分散的、具有独立功能的多台计算机通过通信媒体连接在一起，实现相互之间的通信和信息交换，并配以相应的网络软件，实现资源共享(硬件和软件)、数据传输，提高计算机的可靠性和可用性等目的[43]。

为了方便和清晰地研究复杂的计算机网络，针对计算机网络所执行的各种功能，1983 年国际标准化组织(ISO，International Standards Organization)颁布了“开放系统互连(OSI，Open Systems Interconnection) ”参考模型，将整个网络通信功能划分为七个层次，从上至下依次是应用层、表示层、会话层、传输层、网络层、数据链路层和物理层。按照协议的层次结构，数据从上至下每经过一层均使用该层的协议头进行封装；数据从下往上每经过一层则解除该层的协议头，从而保证传输数据的格式完全一致。

TCP/IP 协议是传输控制协议/网际协议。TCP/IP 的层次比 OSI 参考模型的七层要少，只有四层分别为：应用层、传输层、网络层和数据链路层。与 OSI 参考模型对照，它的应用层对应于 OSI 参考模型的会话层、表示层以及应用层；数据链路层对应于 OSI 的数据链路层和物理层；其他层为一一对应的关系。

TCP 协议提供端到端的连接，用于解决多用户或多任务操作系统中不同应用程序之间的通信。TCP 协议通过端口号标识不同的应用程序或进程。传输层协议主要有两个：传输控制协议 TCP 和用户数据报协议 UDP[44]。TCP 提供可靠的有连接传输，通过确认和超时重传机制保证数据的完整和正确性，通过滑动窗口机制实现拥

塞控制。一般 FTP、HTTP 等协议采用 TCP 进行传输。而 UDP 实现的是不可靠、无连接的数据报服务，它通过校验和保证数据的完整性。

IP 协议提供在同一物理网络中无连接但简单有效的点对点通信。为标识出网络中不同的点，需给该点分配一个地址即 IP 地址。每个 IP 地址都标识了网络中的一个连接。连接到多个物理网络的路由器和多宿主机必须有多个 IP 地址。如果传输过程中出现数据丢失或出错等情况，TCP 协议会自动要求数据重新传输，并重新组包。

5.2 开放源代码源的 TCP/IP 协议栈

在嵌入式系统上运行嵌入式以太网程序需要实现 TCP/IP 协议栈。TCP/IP 协议栈的实现可以参考 freeBSD TCP/IP 栈自行编写。很多商业的和开源的 TCP/IP 协议栈都参考其代码[45](包括 Linux 的 TCP/IP 协议栈、VxWorks 的 TCP/IP 协议栈、LwIP/μIP 协议栈等)。因为TCP/IP 协议栈比较复杂并且嵌入式系统的内存和处理能力比较有限，实现起来难度比较大所以本课题将选用开放源代码的嵌入式 TCP/IP 协议栈，并将其移植到 μC/OS-II 操作系统和硬件系统上来实现网络功能。

目前适用于嵌入式系统的开放源代码的 TCP/IP 协议栈有以下三种。

① LwIP 是由瑞士计算机科学院(Swedish Institute of Computer Science)的 Adam Dunkels 等开发的一套用于嵌入式系统的开放源代码的 TCP/IP 协议栈。LwIP 的含义是 Light Weight(轻型)IP 协议。LwIP 可以移植到操作系统上，也可以在无操作系统的情况下独立运行。LwIP 实现的重点是在保持 TCP 协议主要功能的基础上减少对 RAM 的占用。一般它只需要几十 K 的 RAM 和 40K 左右的 ROM 就可以运行，这使 LwIP 协议栈适合在嵌入式系统中使用，同时他还提供了可选择的 API 接口[46][47]。

② μC/IP 也是一套针对嵌入式微控制器开发的开放源码的 TCP/IP 协议栈。其设计的目标是能方便地移植到各种嵌入式操作系统上。此外他还提供一个 Web 服务的示例程序，代码更加精简[48]。

③ μIP 也是由 LwIP 作者开发的。μIP 是一套适用于 8/16 位机上的小型嵌入式 TCP/IP 协议栈。它以极小的代码量(几千字节)和内存需求量(几百字节)实现了 Internet 通信的基本协议。该协议栈用 C 语言编写并开放源代码。任何人都可以在

遵守 BSD 开源协议的基础上对源代码进行修改并以商业或非商业目的进行发布[49]。

相比之下，μIP 功能精简，比较适合资源十分有限的嵌入式系统，对于可靠性和通讯量有较高要求的场合并不适合。LwIP 和 μC/IP 是两个同级别的协议栈，代码量和功能都非常相似，对 μC/OS-II 的支持也非常好，但 LwIP 的文档管理和软件版本升级更完善。最新发布的稳定版本是 2006 年 3 月发布的 LwIP 1.1.1 。LwIP 在国内受到很多用户的青睐。本课题选用 LwIP 1.1.1 作为 TCP/IP 协议栈。

5.3 LwIP 的移植

LwIP 的移植主要涉及两部分：一是操作系统模拟层的修改，二是以太网硬件驱动程序的设计。以太网硬件驱动程序的设计包括了 LwIP 接口函数设计和硬件相关函数设计。

5.3.1 LwIP 的整体结构与进程模型

LwIP 协议栈在减少对硬件资源占用的基础上对 TCP/IP 协议进行裁剪。LwIP 以分层的 TCP/IP 协议为参照。虽然可以将 TCP/IP 协议族的每一层作为一个独立的进程存在，并严格定义每一层协议的实现和协议之间的接口，每一种协议可以随时投入到系统运行，代码比较容易理解和实现，调试方便。但这并不是一个好的实现方法，最大的问题是当数据跨层传递时将产生进程切换。当接收一个 TCP 数据段时将会引起三次进程切换即从数据链路层进程到网络层进程 再到传输层进程最后到应用层进程。对于嵌入式系统来说，操作系统中进程切换的代价相当昂贵，会占用宝贵的 CPU 处理时间和内存资源。因此LwIP 的所有协议代码独立于操作系统内核，即驻留在同一个进程，因此各层协议也不被严格的区分。虽然 LwIP 被设计运行在小型的操作系统上既不支持进程的换进换出也不支持虚拟内存，但其主要优点是可以方便地移植到不同的操作系统中。

LwIP 没有直接使用属于操作系统的函数调用及数据结构，而是用操作系统模拟层来代替这些函数的使用。所有 TCP/IP 协议都在一个进程中运行。应用层程序既可以驻留在 LwIP 进程中利用内部回调函数 tcp ip_thread( )与 TCP/IP 协议栈通讯，也可以建立单独的进程如图 5-1 中的 T_httpServer 任务(进程)和 T_modbusServer 任

务(进程)，通过操作系统的邮箱、消息队列等与 TCP/IP 进程进行通讯。本系统中 LwIP 的进程模型如图 5-1 所示。

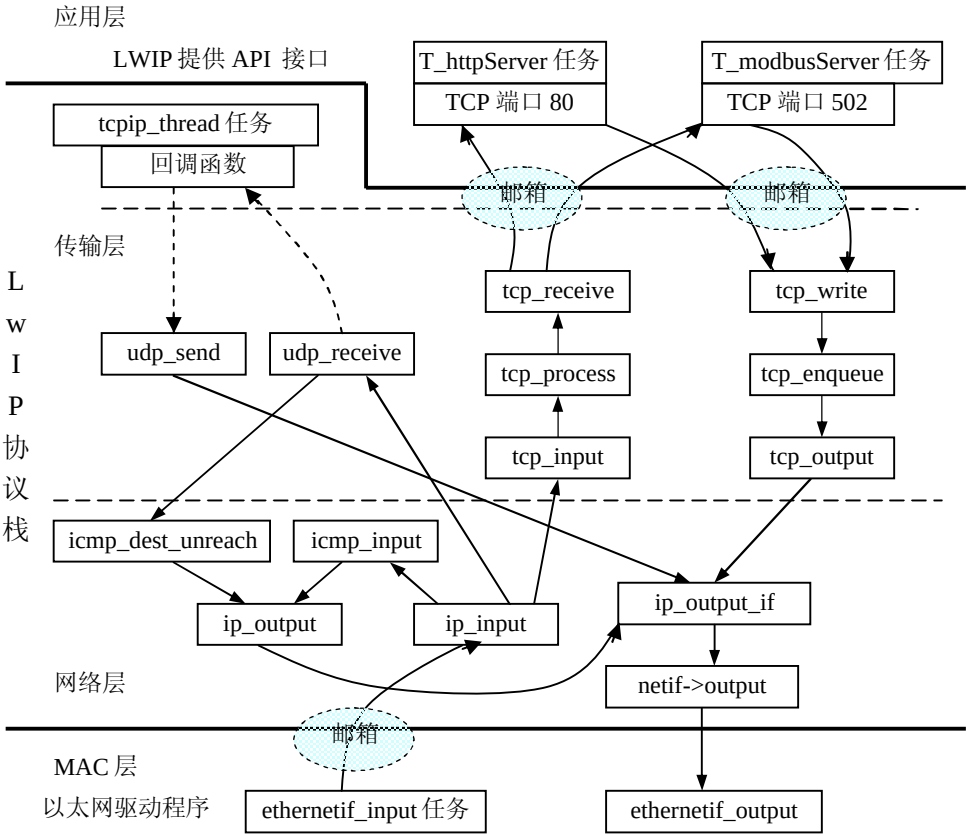


图 5-1 系统中 LwIP 的进程模型

LwIP 1.1.1 源代码目录一共有四个子目录：

① core 目录：LwIP 协议栈实现的核心代码，包括 ICMP，IP，TCP，UDP 协议的实现文件。

② api 目录：操作相对低级的 LwIP API 函数和在此基础上重新封装的兼容 BSD 的 Socket API。LwIP 协议栈提供的服务有两种方法，一是直接调用 TCP 和 UDP 模块；二是使用 LwIP API 或 Socket API 函数。后者通常用于应用程序是单独的进程。

③ Include 目录：放置 LwIP 的所有头文件。

④ netif 目录：ARP 协议的实现文件和 LwIP 网络设备驱动程序的模板 ethernetif.c。用户为自己的网络接口设计驱动程序的时候需要参考该模板中给出的驱动框架。

5.3.2 LwIP 操作系统模拟层的修改

本课题移植的 LwIP 1.1.1 是在修正了 2005 年 1 月发布的 1.1.0 版中的若干缺陷以后并经过充分测试而来的。但是自 LwIP 0.6 以后的版本就没有列出存放与硬件和操作系统相关的操作系统模拟层 arch 目录。事实上 0.5 版本以后的 LwIP 都可以参考 LwIP 0.5 版本中 arch 目录中的函数[46]，因此本课题移植的 LwIP 1.1.1 采用了 LwIP 0.5.3 的操作系统模拟层

操作系统模拟层使用统一的接口来提供定时器、进程同步及消息传递机制等此类系统服务。因此移植 LwIP 需针对 μC/OS-II 操作系统修改模拟层中的若干文件。

(1) 与处理器和编译工具相关的 include 文件

在 LwIP 0.5.3\src\arch\unix\include\arch 目录中有共有下列头文件 cc.h、cpu.h、perf.h、lib.h、sys_arch.h。

cc.h 文件包含一些与 CPU 或编译器相关的数据定义(如数据长度，大小端等)。这些参数应该与 μC/OS-II 中定义的参数一致，具体代码如下：

```
typedef unsigned char    u8_t;
typedef signed char      s8_t;
typedef unsigned short   u16_t;
typedef signed   short   s16_t;
typedef unsigned int     u32_t;
typedef signed   int     s32_t;
```

一般情况下 C 语言的结构体是 4 字节对齐的，但是在处理数据包的时候 LwIP 是根据结构体中不同数据的长度来读取相应的数据的，所以要在定义结构体的时候使用_packed 关键字，让编译器放弃字节对齐。LwIP 给出的例子是 GNU gcc[50]编译器，而本课题采用的是 ADS 集成开发环境自带的编译器，所以要将关键字__attribute__((packed))改成__packed，具体代码如下：

```
#define PACK_STRUCT_BEGIN   __packed
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_FIELD(x) __packed x
#define PACK_STRUCT_END
```

在 cpu.h 文件中可以设置处理器是工作在大端模式还是小端模式下，本系统的 AT91RM9200 微处理器工作在小端模式下，所以配置如下：

```
#ifndef BYTE_ORDER
#define BYTE_ORDER LITTLE_ENDIAN
#endif /* BYTE_ORDER */
```

lib.h 文件声明了 LwIP 中用到的一些库函数，因为在 μC/OS-II 库函数的头文件 string.h 已经声明了这些函数，所以这个文件就不需要了。

perf.h 中定义了 PERF_START 和 PERF_STOP 两个宏，若将其放在一些关键的函数前后，可以进行性能测试，在此定义为空即可。

在 lwip 0.5.3\src\arch\unix 目录下还有一个 lib_arch.c 文件，该文件定义了 4 个高低字节交换的函数，因为 LwIP1.1.1 版已经在的 inet.c 文件中实现了这些函数，所以 lib_arch.c 文件就不再需要。

(2) 与操作系统相关的数据结构与函数

LwIP 需要利用 μC/OS-II 操作系统的一些数据结构与函数实现信号量、消息、提供定时器、创建新线程。主要涉及 lwip 0.5.3\src\arch\unix 目录下的 2 个文件 sys_arch.c 和 sys_arch.h。

① sys_sem_t 信号量

LwIP 要使用信号量进行通信，所以在 sys_arch.c 中需要定义信号量的结构体并实现信号量的处理函数。由于 μC/OS-II 已经实现了信号量 OS_EVENT 的各种操作，并且与 LwIP 中的几个函数的功能是完全一样的，所以只要把 μC/OS-II 的函数重新包装就可以直接使用了，具体的函数如下：

```
typedef OS_EVENT* sys_sem_t;  //sys_sem_t       指向 OS_EVENT 结构体变量的指针
sys_sem_new( );          //                  创建一个信号量的结构
sys_sem_free( );         //                  释放一个信号量的结构
sys_sem_signal( );       //                  发送信号量
sys_arch_sem_wait( );    //                  请求信号量
```

② sys_mbox_t 消息

LwIP 中需要使用消息队列来缓冲、传递数据报文，因此要在 sys_arch.c 中实现消息队列结构以及相应的操作函数。

μC/OS-II 虽然也实现了消息队列结构 OSQ 以及对消息队列本身的一系列操作函数，但是 μC/OS-II 不对消息队列中的消息进行管理，因此 LwIP 必须在 μC/OS-II 的基础上重新实现这一功能才能使用。为了实现对消息队列内部的消息管理，LwIP 对消息队列重新进行了定义，结构体如下：

```
typedef struct {
   OS_EVENT*    pQ;                 //                  μC/OS-II 中指向事件控制块的指针
   void* pvQEntries [MAX_QUEUE_ENTRIES];   //     队列内部的消息
} TQ_DESCR, *PQ_DESCR;
typedef PQ_DESCR sys_mbox_t;
```

μC/OS-II 的 OSQ 操作可以对队列本身进行管理。μC/OS-II 中的内存管理模块可以对消息进行创建、使用、删除和回收。两部分综合起来实现了 LwIP 的消息队列功能，下面是用户需要重新实现的 LwIP 消息队列的操作函数。

```
sys_mbox_new( ); //          创建一个消息队列
sys_mbox_free( ); //         释放一个消息队列
sys_mbox_post( ); //         向消息队列发送消息
sys_arch_mbox_fetch( ); // 从消息队列中获取消息
```

③ sys_arch_timeout 结构体

LwIP 中每个与外界网络连接的线程都有自己的 timeout 属性，即等待超时时间属性。这个属性表现为每个线程都对应一个 sys_timeout 结构体队列，其中包括这个线程的 timeout 时间长度，以及超时后应调用的处理函数。处理函数会做一些释放连接、回收资源的工作。

timeout 结构体已经由 LwIP 在 sys.h 中定义好而且对结构体队列的数据操作也由 LwIP 负责，所要实现的函数如下：

```
struct sys_timeouts * sys_arch_timeouts(void)
```

这个函数的功能是返回目前正处于运行态的线程所对应的 sys_timeout 队列指针。sys_timeout 队列属于线程的属性，因此是与操作系统相关的函数，只能由用户实现。

④ sys_thread_new( ) 创建线程函数

LwIP 既可以单线程运行，即只有一个 TCP/IP 线程(tcpip_thread)负责处理所有的 TCP/IP 连接，实现各种网络程序与网络数据的交互。LwIP 也可以多线程运行，以提高效率，降低编程复杂度。

μC/OS-II 中没有线程的概念只有任务。LwIP 需要用户利用 μC/OS-II 创建新任务的 API 函数 OSTaskCreat( ) 来重写 LwIP 的创建新线程的函数，并由用户在创建 LwIP 线程的时候分配 μC/OS-II 的任务优先级。

```
sys_thread_t sys_thread_new(void (* thread)(void *arg), void *arg, int prio){
   if(curr_prio_offset < LWIP_TASK_MAX){
      OSTaskCreate(thread,(void*)arg,&T_LWIP_TASK_STK[curr_prio_offset][T_LWIP_STKSIZE-1]
            T_LWIP_START_PRIO +curr_prio_offset );
      }
}
```

5.3.3 LwIP 以太网驱动程序的结构

为了实现 LwIP 的移植，除了需要将 LwIP 的操作系统模拟层与 μC/OS-II 相结合，还需要设计 LwIP 的以太网设备驱动程序才能实现 TCP/IP 协议的具体功能。驱动程序的设计与具体的网络硬件设备直接相关，网络硬件设备层次结构如图 5-2。

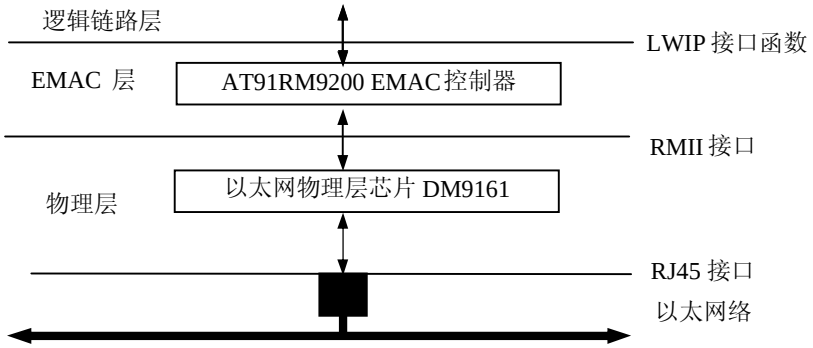


图 5-2 网络硬件设备层次结构图

本系统的 AT91RM9200 微处理器集成了以太网媒质访问层(EMAC，Ethernet MAC)控制器。EMAC 是 OSI 参考模型中的介质访问控制子层，介于物理层(PHY)与逻辑链路层(LLC，logical link layer)之间，以 IEEE 802.3u 数据帧的格式实现微处理器与物理层芯片 DM9161 之间的数据传输。EMAC 控制器通过 RMII 接口访问物理层芯片 DM9161，实现以太网数据的传输。

以太网设备驱动程序需要实现 EMAC 层的接收、发送、初始化以及中断处理功能。LwIP 提供了一个网络驱动程序的参考模板，可以在 LwIP 的源代码目录\src\netif 下找到 ethernetif.c 文件。该文件主要定义了 6 个函数。笔者认为其中 3 个是与 LwIP 的接口函数：ethernetif_init( )、ethernetif_output( )、ethernetif_input( )，另外 3 个是与硬件相关的函数：low_level_init( )、low_level_output( )、low_level_input( )。LwIP 网络数据发送和接收过程很复杂，国内关于 LwIP 的书籍很少，[46]中虽然有一些关于 LwIP 内容的介绍，但书中的程序是在软件仿真的环境中设计出来的，所以涉及 LwIP 网络设备驱动程序设计的内容就很少。网上很多内容也都是参考该书的。笔者认为 LwIP 以太网设备驱动程序应该分成与 LwIP 的接口函数和与具体硬件(AT91RM9200 的 EMAC 控制器及 DM9161)相关的函数。这些函数之间的关系如图 5-3。

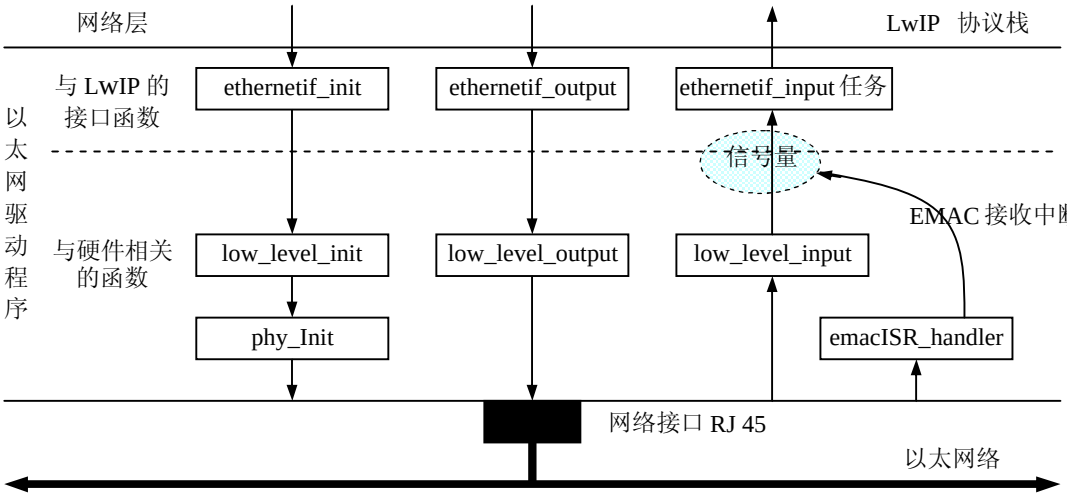


图 5-3 以太网硬件驱动程序结构

5.3.4 LwIP 接口的数据结构与函数的设计

(1) LwIP 接口数据结构 netif

LwIP 的每个物理网络接口的硬件驱动都通过一个与 BSD 中相似的网络接口数据结构表示。一个接口对应一个 netif 结构体变量。在这个结构体中定义了网络接口的属性、接收和发送函数等，具体代码如下：

```
struct netif {
  struct netif *next;          //                指向系统的下一个 netif 结构体变量
  struct ip_addr ip_addr;      //                网络接口的 IP 地址
  struct ip_addr netmask;      //                网络接口的子网掩码
  struct ip_addr gw;           //                网关地址
//这个函数将网络驱动设备接收的数据包传给 TCP/IP 协议栈
   err_t (* input)(struct pbuf *p, struct netif *inp);
//这个函数将 IP 层要发送的数据包发送到链路层
  err_t (* output)(struct netif *netif, struct pbuf *p, struct ip_addr *ipaddr);
//这个函数被 ARP 模块调用，将链路层的 pbuf 数据包发送到物理层网络
  err_t (* linkoutput)(struct netif *netif, struct pbuf *p);
  void *state;                 //                设备工作状态
  u8_t hwaddr_len;             //                硬件地址的长度
  char name[2];                //                接口的描述名
//本网络接口的编号，主要用于系统有多个物理网络接口的情况
  u8_t num;
……
};
```

(2) LwIP 接口的初始化

LwIP 初始化的时候会调用 ethernetif_init( )函数来实现网络接口 MAC 层的初始化，具体步骤如下：

① netif 网络接口结构体的初始化

② 调用 low_level_init( ) ，对与以太网硬件相关的寄存器进行初始化，即对 AT91RM9200 的引脚和中断控制器、EMAC 模块的寄存器以及物理层芯片 DM9161 的寄存器进行配置。

③ ARP 模块的初始化

④ 时间模块初始化

⑤ 创建并开启以太网接收 Ethernetif_input( )任务

(3) LwIP 接口的数据发送

在 LwIP 接口初始化的步骤 1 中，对 netif 结构体变量中的指向发送函数的指针进行了初始化，代码如下：

```
Netif->output= ethernetif_output ;
Netif->linkoutput= low_level_output ;
```

当 LwIP 的网络层向以太网发送数据的时候就会调用 netif->output 方法，ethernetif_output( ) 函数就会调用 etharp_output( ) 函数对 IP 报文进行 MAC 帧的封装：首先添加源 MAC 地址，然后根据目的 IP 地址检索 ARP 地址表，如果检索到对应的 MAC 地址，则添加目的 MAC 地址，如果没有检索到还需要发送 ARP 请求包，等待获得目的 IP 地址的 ARP 响应包，取得目的 MAC 地址以组成完整的 EMAC 帧。如果没有 ARP 响应则目的地址填写网关的 MAC 地址。最后调用 netif->linkoutput 函数，low_level_output( )函数就会将完整的 EMAC 帧发送给物理层芯片 DM9161，最终发送到网络上，完成一次以太网数据的发送。

(4) LwIP 接口的数据接收

由于有 μC/OS-II 操作系统的支持，所以笔者将以太网接收函数 ethernetif_input( ) 设计成为一个任务。该任务等待从以太网接收中断处理函数发送的信号量。如果收到信号量则调用__low_level_input( )函数从网络接口接收 EMAC 帧，解帧后将其中的 IP 报文提交 LwIP 的网络层。

这里以太网数据的接收和处理没有在中断里进行是因为以太网数据的传输速度很快。数据接收的中断处理函数执行时间不能过长，否则就会发生丢包。接收的数据包会通过 AT91RM9200 EMAC 中的 DMA 控制器保存到内存的缓冲区中。缓冲区最大可以保存 8 个长度为 1600 字节的数据包。以太网接收中断处理程序不做具体的数据接收，仅发送一个信号量，通知 ethernetif_input( ) 任务进行数据接收。

ethernetif_input 任务进入就绪态并会在下一次时钟节拍到来的时候被调度，调用 low_level_input( ) 函数从内存的缓冲区中一次将所有的以太网数据包拷贝出来。因此适当提高 μC/OS-II 的时钟节拍频率可以提高以太网接收的响应速度。

5.3.5 LwIP 硬件相关的函数设计

(1) AT91RM9200的 EMAC 控制器结构

AT91RM9200内嵌一个与 IEEE802.3 以太网标准完全兼容的 10M/100M 半双工或全双工以太网控制器。它包括 EMAC 工作逻辑和 DMA 发送接收管理部件。它管理帧的发送与接收，进行冲突检测、报头生成与检测、CRC 控制与短帧的填充。此外它还通过 MDIO/MDC 引脚实现物理层的管理。具体结构见图 5-4。

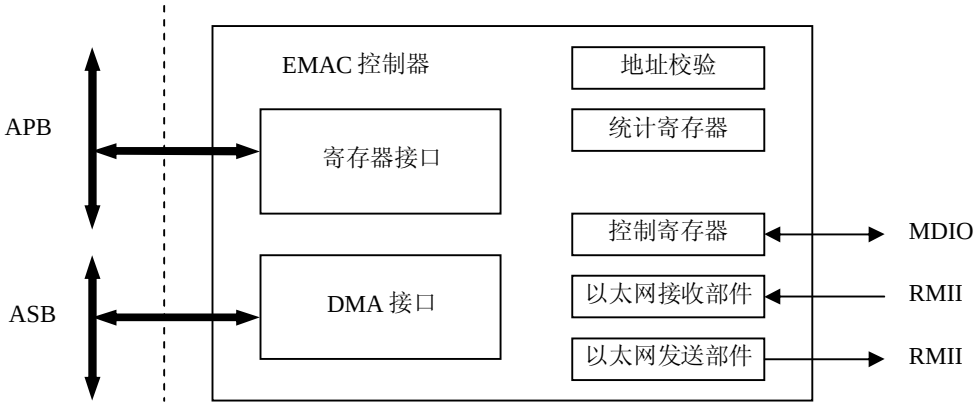


图 5-4 AT91RM9200 的 EMAC 控制器的结构

① MDIO 是数据输入/输出管理接口，作用有设置物理层芯片 DM9161 的寄存器，启动帧传输，选择双工或半双工操作模式等。

② 以太网接收部件的作用是检查前导字段的有效性，帧校验序列 FCS 是否对齐以及把接收到的帧进行地址校验和递交给 DMA 接口。

③ 以太网发送部件从 DMA 接口取出数据，添加前导字段。如果必要，填充数据(如果发送一个长度小于 64 个字节的数据包，MAC 会生成填充字节将这个数据包扩展到 64 个字节的最小限制)和 FCS，然后按照 CSMA/CD 载波监听多路访问/冲突检测协议传输数据。

④ DMA 接口通过 ARM920T 核的 ASB 先进系统总线接口连接外部存储器。它包含帧缓冲数据的管理和先进先出 FIFO 队列。它加载发送 FIFO 队列并且使用 ASB 总线主动清空接收 FIFO 队列。

⑤ 地址校验告知 DMA 部件是否应该将接收到的帧复制到内存的接收缓冲区。一个帧是否被复制取决于网络配置寄存器中是否设置为允许复制，以及外部配置引脚的状态，特殊地址和哈希寄存器的内容和帧的目的地址。

⑥ 寄存器接口通过 ARM920T 核的 APB 先进外设总线对 EMAC 的控制和状态寄存器进行配置和访问。

(2) 以太网硬件的初始化

EMAC 控制器允许在系统的内存单元中开辟多个接收缓冲区。接收缓冲区是字对齐的，最多可包含 1600 字节的数据。这些缓冲区由一张接收缓冲区说明符列表进行管理。每个接收缓冲区对应一个列表项，每个列表项有两个字：第一个字包含了接收缓冲区的地址。该字的第 0 位初始化为 0，表示缓冲区可被 DMA 控制器使用。当缓冲区接收数据后该位被置 1；第二个字为缓冲区接收状态信息，包括接收数据的长度等。该列表开辟在系统的内存单元。接收缓冲区说明符列表的首地址在 EMAC 初始化的时候写入 EMAC 接收缓冲队列指针寄存器。

以太网硬件初始化函数 static void low_level_init(st_netif*pstNetif) 执行过程如下：

① 配置 EMAC 的 RMII 引脚

② 建立接收任务使用的信号量

③ 初始化接收和发送缓冲区描述符

④ 设置 EMAC 地址

⑤ 使能 EMAC 的发送和接收

⑥ 执行物理层芯片 DM9161 初始化函数 phy_Init( )

⑦ 根据从物理层获取的网络链接信息，设置 EMAC 的链路速度和单双工方式

⑧ 在高级中断控制器 AIC 中配置 EMAC 中断入口函数

⑨ 在高级中断控制器 AIC 中开放 EMAC 中断

对于物理层芯片 DM9161 的控制是通过 EMAC 的 MDIO 接口实现的，在对芯片 DM9161 操作之前首先设置 EMAC 控制寄存器的 MPE 位使能 EMAC 的 MDIO 控制接口，然后通过 EMAC 的物理层维护寄存器 MAN 来实现物理层芯片 DM9161 的寄存器读写。

物理层芯片 DM9161 的初始化函数 phyInit( )的初始化过程如下：

① 复位物理层芯片

② 检查物理层芯片 DM9161 的 ID

③ 从 PHY 获取自动协商的结果

(3) 以太网硬件的数据发送

发送 EMAC 帧数据时要保证数据存储在连续的存储区域内，不需要字对齐。将待发送数据的首字节地址写入发送地址寄存器，再向发送控制寄存器中写入发送的字节数,然后发送通道从存储器同时读取 32 位数据并将它们置于发送队列 FIFO。当 FIFO 中载入三字后，发送部件开始帧发送。当帧正在发送时，可以对发送地址与发送控制寄存器中写入新值来启动其他帧的发送。

以太网 硬件 的 数 据 发 送 函数 static err_t low_level_output (st_netif* pstNetif,st_pbuf * pstPbuf)执行过程如下：

① 设置访问 EMAC 的互斥型信号量，并保证同一时刻只有一个任务进行 EMAC 的数据发送

② 若发送的 EMAC 帧超过 1546 字节则进行出错处理

③ 循环判断发送缓冲区是否可用，若不可用则超时退出

④ 若缓冲区可用，进入 μC/OS-II 的临界区

⑤ 清除 EMAC 的发送完成标志

⑥ 设置 DMA 发送缓冲区地址

⑦ 设置发送数据的长度(设置完毕后 EMAC 自动发送)

⑧ 等待发送完毕

⑨ 退出 μC/OS-II 的临界区

(4) 以太网硬件的数据接收

一旦接收开始，接收缓冲管理器读取 EMAC 接收缓冲队列指针寄存器指向的第一个列表项第一个字中的接收缓冲区的内存地址单元，接收的 EMAC 帧数据按 DMA 方式依次写入该地址单元所在的内存。写入完毕后列表项第一个字的位 0 自动置 1，表示该缓冲区已存放 EMAC 帧，不再被 DMA 控制器使用；第二个字中自动保存了接收的帧的长度。

若当前列表项的第一个字的位 1(环绕位)为 1，则表示已到达最后一个接受缓冲区，接收缓冲管理器就会回绕读取接收缓冲区说明符列表中第一个列表项第一个字

中的接收缓冲区的内存地址，否则读取下一个列表项第一个字中的接收缓冲区的地址。

对接收到的 EMAC 帧将检查报头、CRC 校验、长度及地址是否有效。若符合所有标准，EMAC 帧将成功存放在接收缓冲区中。若出错则接收缓冲区恢复。

以太网硬件的数据接收函数 static st_pbuf* low_level_input(st_netif*pstNetif) 执行过程如下：

① 设置访问 EMAC 的互斥型信号量，并保证同一时刻只有一个任务进行 EMAC 数据接收

② 从接收缓冲区说明符列表项中获取收到的数据包的长度

③ 如果收到数据包是有长度的，则从 LwIP 的内存缓冲池 pbuf pool 中获取缓冲块 pbuf

④ 把接收缓冲区中的数据拷贝到 pbuf 中

⑤ 对接收缓冲区对应的列表项中第一个字的位 0 清 0，将接收缓冲区归还给 DMA 控制器

5.4 小结

本章在对几种开放源代码的嵌入式 TCP/IP 协议栈进行比较之后，选用了嵌入式 TCP/IP 协议栈 LwIP。在研究了 LwIP 整体结构和进程模型的基础上，给出了基于 μC/OS-II 操作系统的 LwIP 操作系统模拟层的修改方法。在深入分析了 LwIP 驱动程序模版的基础上提出了 LwIP 以太网驱动程序的设计方法，并给出 AT91RM9200 以太网模块的 LwIP 硬件驱动程序，为软件系统应用层程序的设计做好了准备。

第六章 应用层程序设计

根据本文第二章的系统的总体设计，本章将进行软件系统开发的第三部分即在嵌入式操作系统 μC/OS-II 和 LwIP 的基础上进行应用层程序的设计，以实现嵌入式 Modbus/TCP 网关的两个主要功能：一是通过网关的 Web 服务功能对网关进行配置；二是通过 Modbus/TCP 协议将 Modbus 串行通信链路与以太网相连。在软件系统实现完毕之后将对 Modbus/TCP 网关进行测试，最后给出网关的应用案例，下面将一一介绍这些内容。

6.1 Web 服务的实现

6.1.1 嵌入式 Web 服务实现机制

万维网（World Wide Web，WWW，简称 Web）运行的基础是超文本传输协议（HTTP，Hypertext Transfer Protocol）[51]。Web 服务器就是提供 WWW 服务的主机。通过 HTTP 协议，Web 服务器可以响应客户端的请求，并提供相应的页面和数据内容的服务。Web 技术采用客户机/服务器模式，简单高效，自 1990 年提出以后经过十几年的发展，已得到了广泛的应用。目前 Web 使用的是 HTTP/1.0 和 HTTP/1.1 版协议。这两个版本协议的主要差别在于 HTTP/1.0 使用非持久连接，HTTP/1.1 则默认使用持久连接[52]。因为本课题实现的 Web 服务目前只提供基本的超文本标记语言(HTML，HyperText Markup Language) 文件，因此对 HTTP/1.1 与 HTTP/1.0 都是兼容的。

HTTP 协议实现的客户机/服务器模式是一种请求/响应行为。嵌入式 Modbus/TCP 网关将充当 Web 服务器。客户机是任意一台与网关相连的计算机。它们实现一次请求响应的具体步骤如下：

① 用户请求一个 Web 页面，如在浏览器的地址栏中输入嵌入式网关的统一资源定位器 URL 地址 http://192.168.2.9。客户机会初始化一个与服务器的 TCP 连接。嵌入式网关使用默认端口号 80 来监听客户机的连接请求，经过三次握手之后建立连接。

② 客户机通过与 TCP 连接的本地套接字发送一个 HTTP 的请求信息。信息中

包含 URL 地址 http://192.168.2.9。

③ 嵌入式网关经由与 TCP 连接的本地套接字接收这个请求后，从片外 FLASH 中取出请求的网页内容，经由本地套接字发出包含该对象的响应信息。

④ 嵌入式网关告知 TCP 关闭这个连接。

⑤ 客户机经由本地套接字接收这个响应信息。该消息标明所封装的对象是一个 HTML 文件。客户机从中取出这个文件，并在客户机的浏览器上显示对应的 Web 页面的内容。TCP 连接随后终止。

在这里嵌入式网关进行的 TCP 连接、监听、终止操作都是依赖 LwIP 的 API 函数完成的。

6.1.2 HTTP 消息格式

在 HTTP 请求响应过程中需要用到两种消息，一种是请求消息，另一种是响应消息。要实现 Web 服务的功能就需要清楚这两种消息的格式。

① HTTP 请求消息

HTTP 请求消息是客户端通过浏览器向服务器发送的，用于请求服务器提供某种类型的服务，消息格式如表 6-1 所示。

表 6-1 HTTP 请求消息格式

<table>
<tr><td colspan="2">方法</td><td>空格符</td><td>URL</td><td>空格符</td><td>版本</td><td>回车符</td><td>换行符</td><td>请求行</td></tr>
<tr><td colspan="3">头部字段名</td><td>:</td><td>值</td><td>回车符</td><td>换行符</td><td rowspan="3">头部行</td><td></td></tr>
<tr><td colspan="6">.....</td><td></td></tr>
<tr><td colspan="3">头部字段名</td><td>:</td><td>值</td><td>回车符</td><td>换行符</td><td></td></tr>
<tr><td>回车符</td><td>换行符</td><td colspan="6"></td><td></td></tr>
<tr><td colspan="8"></td><td>附属体</td></tr>
</table>

请求消息的第一行是请求行，有三个字段：方法字段、URL 字段和 HTTP 版本字段。方法字段有若干值可供选择如 GET、POST、HEAD。HTTP 请求消息绝大多数使用 GET 方法。这是浏览器用来请求对象的方法，所请求的对象就在 URL 字段中标识。POST 方法则适用于需要用户提交表单的场合。POST 方法数据量比较大。用户填写在表单各个字段的值都存放在 POST 方法请求消息的附属体中。GET 方法和 POST 方法在本课题的 Web 服务设计中都被使用到，后面会介绍 Web 服务针对这两种请求的具体处理方法。请求消息是由浏览器生成的，因此不在此做详细介绍，具体内容可以参考[52]。

② HTTP 响应消息

响应消息是服务器根据客户机提出的请求而作出响应，以提供相应的服务。消息格式如表 6-2 所示。

表 6-2 HTTP 响应消息格式

版本	空格符	状态码	空格符	原因短语	回车符	换行符	状态行
头部字段名		:	值	回车符	换行符	头	
.....						部	
头部字段名		:	值	回车符	换行符	行	
回车符	换行符						
							附属体

本课题的响应消息是由嵌入式网关生成的。其中的状态行和头部行中的内容被设计为对应网关程序中的一个宏 HTTP_HTML_HEADER，定义如下：

```
#define  HTTP_HTML_HEADER     "HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n"
```

在这里可以看到网关发送的 HTTP 响应信息的版本是 HTTP/1.1，返回状态码是 200，原因短语是 OK，表示所请求的信息在响应[illegible]返回。“\r\n”则表示回车符和换行符。ADS 编译器对程序进行编译的时候会将他们转换为 ASCII 码 0x0D 和 0x0A。头部行的内容是 Content-type: text/html\r\n，指出 HTML 文本对象包含在附属体中。头部行之后的“\r\n”表示一个空行。到这里为止状态行和头部行就全部结束了，后面的数据都属于附属体。

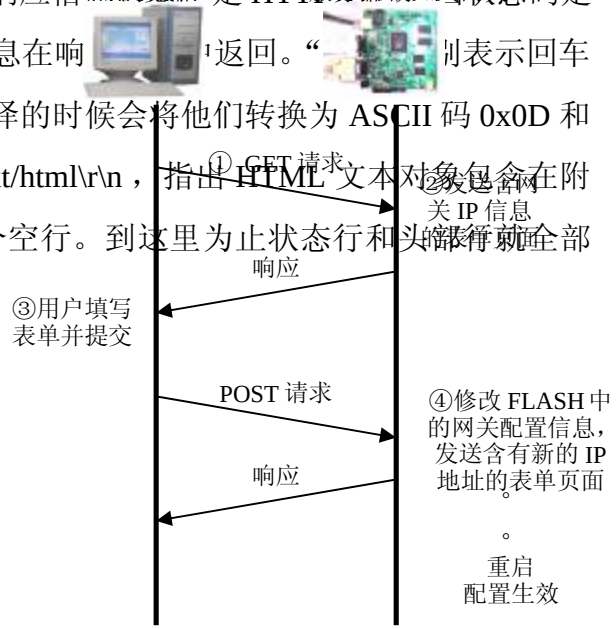


图 6-1 Web 服务两次响应过程

6.1.3 Web 服务程序设计

利用 Web 服务对嵌入式网关的信息进行设置需要经过两次请求响应过程如图 6-1。工作流程图如图 6-5。

① 网关复位启动之后，μC/OS-II 运行 T_httpServer 任务(启动 Web 服务)。用户通过浏览器向网关发出 GET 方法的请求。

② 网关收到该请求后对请求消息中的方法字段进行判断。如果是

GET 方法，则表示是第一次请求，网关执行__Handler_HTTPGet( ) 函数，将固化在片外 FLASH 中的 HTTP 表单页面和网关的配置信息返回给客户机。处理步骤如下：

```
__Handler_HTTPGet ( ) {
  1 将本文 6.1.3 中的宏 HTTP_HTML_HEADER对应的数据复制到发送队列
  2 从片外 FLASH 中取出请求的网页数据复制到发送队列
  3 对字符串__szNetCFGParam 清 0
  4 将字符串"<p>IP："添加到到字符串__szNetCFGParam 的尾部
  5 从片外 FLASH 中获取本机的 IP 地址并添加到__szNetCFGParam 的尾部
  6～9 依次添加字符串"<br>子网掩码："和本机子网掩码，字符串"<br>网关："和本机网关
地址到字符串__szNetCFGParam 的尾部
  10 将数组__szNetCFGParam 的内容复制到发送队列并发送
}
```

网关发送给浏览器的表单页面如图 6-2，图中①的中文字符是函数__Handler_HTTPGet（）中的步骤 2 是直接从片外 FLASH 中拷贝过来的；②中的中文字符是函数__Handler_HTTPGet（）中的步骤 6、8 由 ADS 开发环境自动将“子网掩码”、“网关”中文字符串转换成对应的简体中文编码并发送出来的。

③ 用户填写网关发送的表单页面，输入新的 IP 地址、子网掩码和网关地址，然后点击提交按钮向网关发出 POST 方法的请求。在这里需要对用户填写的数据进行验证，验证工作是通过嵌在表单页面中的 javascript 程序实现的，这样就不需要在服务器端对数据进行验证，减轻了网关的负担。

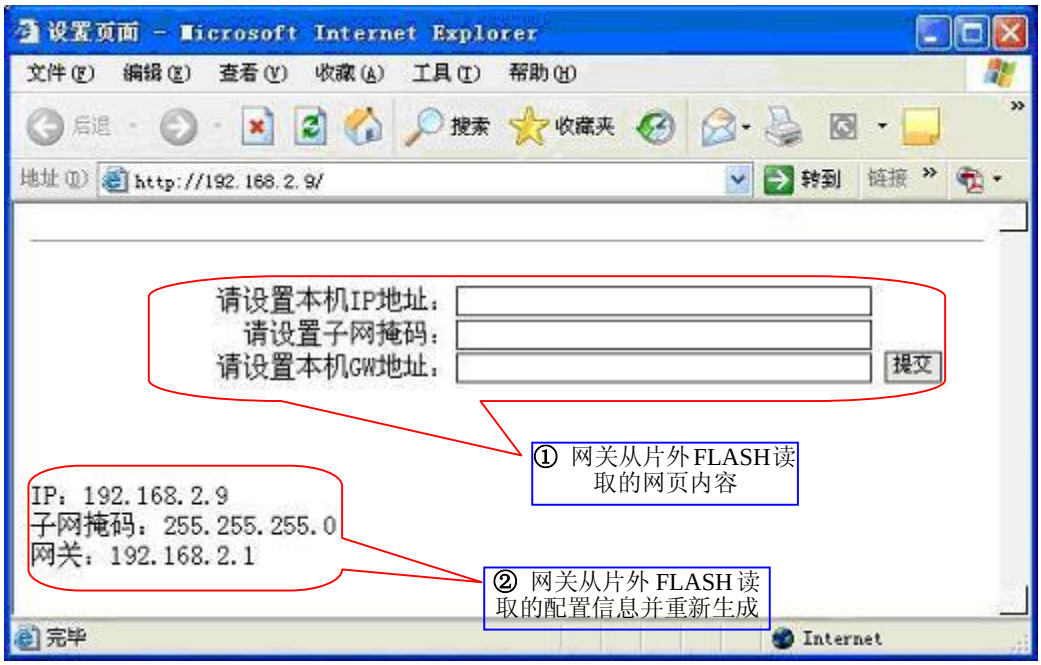


图 6-2 网关发送给浏览器的表单页面

④ 网关收到请求后对请求消息中的方法字段进行判断，发现是 POST 方法，则表示是第二次请求，网关收到的请求消息如下：

```
POST /vSetNetCFGParam HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg……
  ......
Cache-Control: no-cache

ip=192.168.2.6&nm=255.255.255.0&gw=192.168.2.1
```

在请求消息附属体中的数据就是用户提交的数据，网关调用__Handler_HTTPPost( )函数对数据进行处理，处理步骤如下：

```
__Handler_HTTPPost（）{
  //解析 POST 请求,寻找“POST /”后面的表单名 vSetNetCFGParam 对应的处理函数
  1 __pfunParsePostMethod（）
    //若找到对应的处理函数__vSetNetCFGParam( )则执行该函数
    1.1 执行函数__pstGetParamNetbuf（）获取消息附属体所在的内存的地址
    1.2 提取用户提交的网关配置参数，如 ip=192.168.2.8&nm=255.255.255.0&gw=192.168.2.1
    1.3 将参数由 ASCII 字符转换为 16 进制数，并分别存放在 3 个数组中
    1.4 执行函数 ubSetNetCFGParam( )将新的网络参数进行保存
      1.4.1 SectorErase（） // 擦除网关配置信息所在的 FLASH 块
      1.4.2 FlashProg（） // 将新的网络配置信息写到对应的 FLASH 块中
    1.5 执行函数__Handler_HTTPGet (pstConn)向浏览器发送反馈页面以检查设置是否成功
  2 若没有找到对应的处理函数则执行函数__vHTTPError( )进行出错处理
    2.1 执行函数 vHTTPRespInfo( )向浏览器返回一个 HTTP 出错响应信息
}
```

在这里网关的配置信息是存放在片外 FLASH 中的，这样可以保证网关的配置在复位重启之后不会丢失，所以网关需要擦除片外 FLASH 中原有的配置信息然后重新烧写新的信息。这一过程要求程序运行在 RAM 区，这也就是本文第 4 章中强调的网关启动之后要把 FLASH 区代码拷贝到 RAM 区中然后从 RAM 区重新运行的原因。

对片外 FLASH 的擦除和写入需要遵循一定的算法。不同厂家不同型号的 FLASH 擦写算法是有一定差别的，这里需要提一下的是 FLASH 的擦除时间比较长，本课题使用的 AM29LV640MB 擦除一个数据块需要 0.5 秒[53]，该过程不能被 μC/OS-II 中运行的别的任务打断否则就会擦除失败。笔者将该代码设置为 μC/OS-II 临界区代码加以保护，解决了 FLASH 擦除失败的问题。

在对片外 FLASH 烧写新的数据后，网关会向浏览器再发送一个反馈页面。这是通过再次调用处理第一次请求中的函数__Handler_HTTPGet( ) 实现的。该函数会再次从片外 FLASH 中取出网关配置信息并发送给客户机，而这次的数据是被更新过的了。网关配置信息更新之后并不能立刻生效，只有在网关重启复位后对 LwIP 重新进行初始化，新的配置才能生效。

6.2 Modbus/TCP 协议转换的实现

6.2.1 Modbus 通信模式

本文的第一章已经对 Modbus 协议进行了初步的介绍，Modbus 协议是应用层的协议，位于 OSI 模型的第 7 层。他可以在不同的总线或网络设备上提供客户机/服务器模式的通信[54]。根据底层网络的不同，Modbus 协议可分成串行链路版本和运行在以太网的 Modbus/TCP 协议。Modbus/TCP 协议以一种比较简单的方式将 Modbus 帧嵌入到 TCP 帧中，Modbus 通信协议结构如图 6-3。

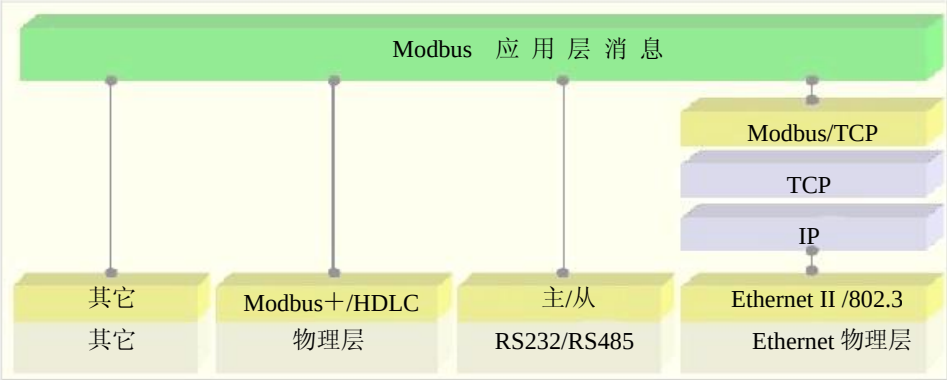


图 6-3 Modbus 通信协议结构

网关(Gateway)又叫协议转换器，是能够连接不同网络软件和硬件的产品。它们可以使用不同的格式、通信协议或结构连接起两个系统，实现不同协议网络之间的互连[55]。Modbus/TCP 网关实际上就是通过重新封装消息使它们能被另一个网络中的设备读取。

Modbus 协议采用请求响应模式，通信方法是对等的。Modbus 规定仅客户机能发起查询，服务器仅能对客户机发出的查询消息作出响应。典型的客户机是 PC 机、可编程仪表、人机界面。典型的服务器是可编程控制器。如果客户机发送请求消息，表示他期望从服务器设备得到响应。同样，当服务器接收到消息，它将建立一应答消息并返回给客户机，交互过程见图 6-4。

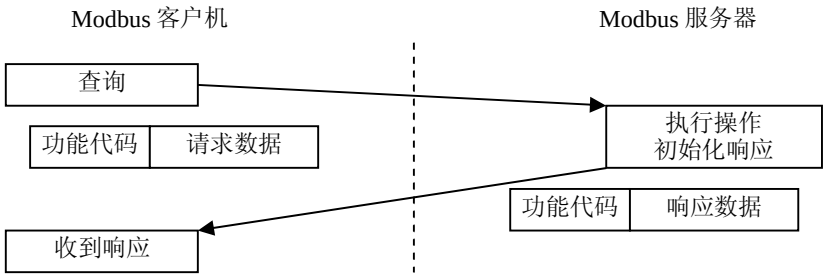


图 6-4 Modbus 客户机与服务器的请求响应过程

①客户机查询

查询消息中的功能代码指示被选中的 Modbus 服务器要执行何种功能的操作。数据段包含了服务器要执行操作的附加消息。例如功能代码 03 是要求服务器读取寄存器并返回它们的值。数据段中的消息说明要读取服务器的寄存器的地址以及读取的寄存器数量。

②服务器响应

如果服务器产生正确的响应，响应消息中的功能代码就是查询消息中的功能代码。响应消息中的数据段包括了服务器收集的数据，如寄存器值或状态。如果有错误发生，功能代码将被修改以用于指出回应消息是错误的，同时数据段包含了描述此错误消息的代码。

6.2.2 Modbus 数据帧格式

在 Modbus 串行链路上的所有设备(客户机和服务器)需要配置相同的通信参数和通信模式。串行通信参数包括波特率、奇偶校验等。通信模式包括 ASCII 模式和 RTU 模式。当设备以 ASCII 模式进行通信时，消息中的每一个 8 位的字节作为 2 个 ASCII 字符传输。当设备以 RTU 模式进行通信时，消息中的每一个 8 位的字节分成两个 4 位的 16 进制的字符传输[56]。因此在相同的波特率下 RTU 模式有更高的传输密度，应用也更广。Modbus 协议是以帧为单位进行数据传送的，Modbus 有三种类型的帧：在串行链路上运行的 ASCII 帧和 RTU 帧、在以太网上运行 Modbus/TCP 帧。

① ASCII 帧

ASCII 帧以“：”冒号字符(0x3A)表示帧的起始，以回车和换行字符（0x0D 和 0x0A）表示帧的结束。帧中其他的数据只允许为 16 进制数中的字符 0～9 和 A～F。

帧的格式见表 6-3。网络中的Modbus 服务器会不断监听冒号，如果监听到冒号则判断该帧中的地址是否为本机地址，是的话则予以响应。ASCII 帧的字符之间的最大间隔为 1 秒，若超过 1 秒，接收设备认为发生了一个错误。

表 6-3 ASCII 帧的格式

起始字符 ：	地址	功能代码	数据段	LRC 校验
1 字节	2 字节	2 字节	0～2×252 字节	2 字节

② RTU 帧

RTU 帧以超过 3.5 个字节的空闲时间开始，以 3.5 个字节的静止时间结束。帧的格式见表 6-4。帧中的数据允许为任意的数。整个帧必须连续发送，发送期间若出现超过 1.5 个字符的静止时间，则后面的数据会被认为是新的帧。

表 6-4 RTU 帧的格式

起始	地址	功能代码	数据段	CRC 校验
≥3.5 字节	1 字节	1 字节	0～252 字节	2 字节

③ Mobus/TCP 帧

Modbus/TCP 没有对 Modbus 协议本身进行修改，而是采用了一种非常简便的方式将 Modbus RTU 帧嵌入到以太网的 TCP 帧中。Modbus/TCP 帧的格式见表 6-5。

表 6-5 Modbus/TCP 帧的格式

MBAP 头				协议数据单元 (与 RTU 帧兼容)	
传输标志	协议标志	长度	单元标志(地址)	功能代码	数据段
2 字节	2 字节	2 字节	1 字节	1 个字节	0～252 个字节

Modbus/TCP 帧由 MBAP(Modbus 应用协议)头和协议数据单元组成。因为考虑到以太网 IP 包到达的顺序可能与发送的顺序不一致，所以在 MBAP 中增加了传输标志，Modbus/TCP 服务器端(Modbus/TCP 网关)需要复制它做为对传输请求的响应。MBAP 中的协议标志字段为 0 表示为 Modbus 协议。长度字段对该字段后的单元标志字段和协议数据单元中的字节数计数[57]。

Modbus/TCP 帧的目的地址由 IP 地址所确定，因此不再有专门的地址字段来区分不同的 Modbus/TCP 客户。同时 TCP 传输是可靠的数据传输方式，因此不再有数据校验。因为Modbus/TCP 协议支持承载 Modbus 串行链路版本的 RTU 帧，所以在 MBAP 中增加了一个地址字段以区分串行链路中不同的 Modbus 服务器，实现串行链路中的路由功能[55]。

6.2.3 Modbus 数据模型

表 6-6 Modbus 数据模型

基本模型	数据类型	访问类型	注释
开关量输入	1 位	只读	离散寄存器的值由服务器 I/O 系统提供数据
开关量输出	1 位	读写	线圈寄存器的值由客户机应用程序改变
寄存器输入	16 位	只读	输入寄存器的值由服务器 I/O 系统提供数据
寄存器保持	16 位	读写	保持寄存器的值由客户机应用程序改变

Modbus 基本数据模型有四种，如表 6-6。Modbus 所有要处理的数据均放在设备的存储器中。Modbus 功能代码定义的各种操作也是针对这四种基本数据模型所进行的操作。Modbus 功能代码分成三类：公共功能代码、用户定义的功能代码和保留的功能代码，具体内容参见[54]。

6.2.4 Modbus/TCP 协议转换程序设计

Modbus/TCP 嵌入式网关复位启动之后会进行一系列的初始化操作，最后会启动 T_modbusServer 任务，该任务实现 Modbus/TCP 帧与串行链路中的 Modbus RTU 帧之间的转发。整个任务的工作流程如图 6-5。

当 Modbus/TCP 客户机需要对串行链路中的服务器进行查询时，首先会向 Modbus/TCP 网关的 502 端口发起连接请求，网关监听到这一请求后会与之建立 TCP 连接，Modbus/TCP 客户机就会发送一个 Modbus/TCP 请求帧并等待网关的响应信息。网关在收到 Modbus/TCP 客户机的请求信息后会对帧进行分析处理，最后会生成一个 Modbus RTU 格式的查询帧并发送到串行链路中去，然后监听串行链路，若收到串行链路上的 RTU 响应帧，则将该帧封装成 Modbus/TCP 应答帧，发送给以太网的 Modbus/TCP 客户机，并断开连接。

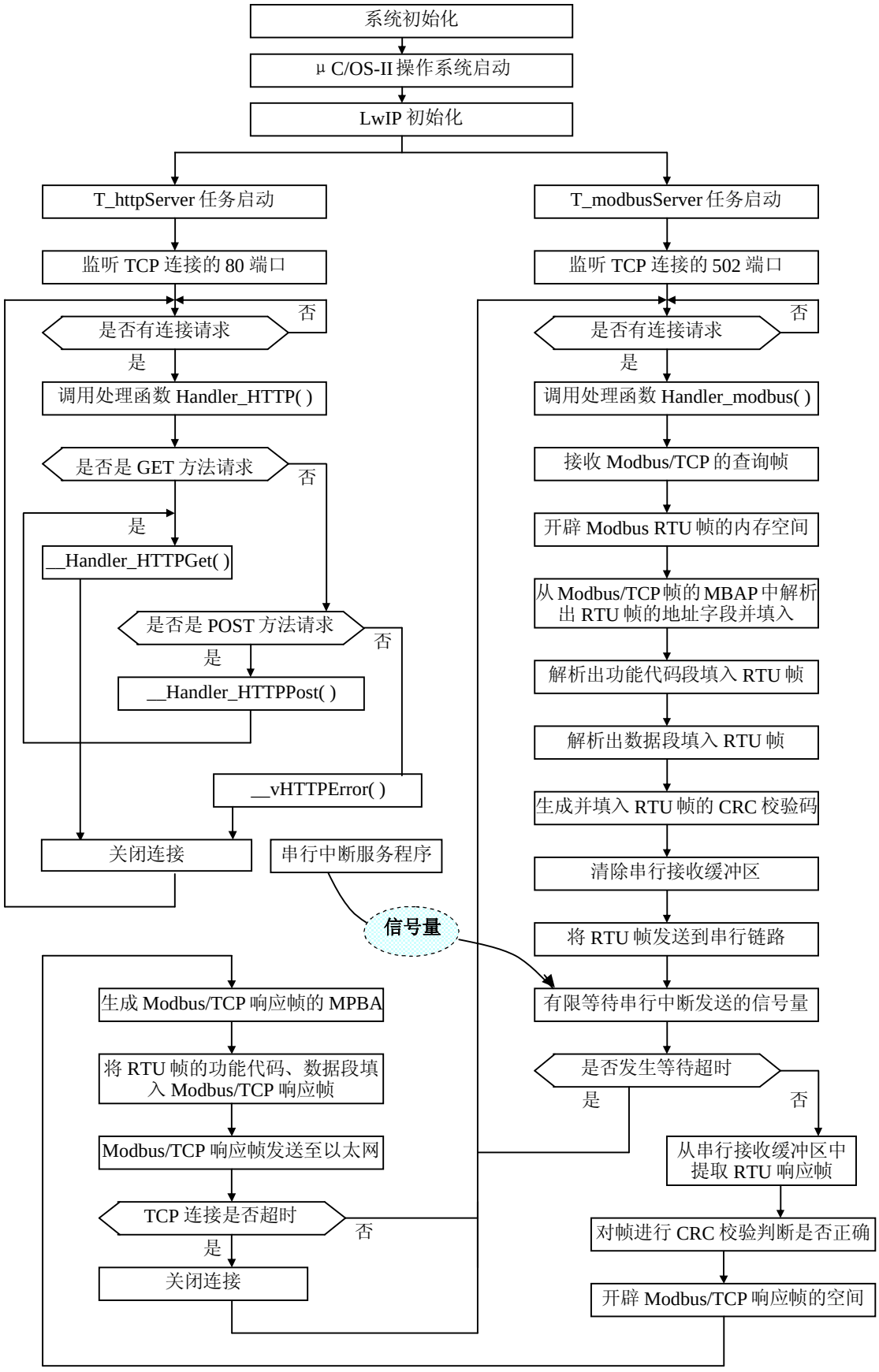


图 6-5 应用层程序流程图

程序设计过程中需要注意的是：第一，Modbus/TCP 请求帧的 MPBA 头中的各个字段需要保存。因为当收到串行链路返回的 RTU 应答帧时还需要用这些字段重新封装 Modbus/TCP 的应答帧。第二，因为 TCP 是可靠的连接，所以 Modbus/TCP 帧没有单独的校验码，所以网关需要根据接收到的 Modbus/TCP 查询帧中的功能代码、数据段、MPBA 中的地址字段的值生成 Modbus RTU 帧的 CRC 校验码。第三，Modbus 协议是请求响应机制，所以 Modbus 网关空闲时只监听以太网接口不监听串行通信接口。当需要向串行链路发送 RTU 帧时，网关首先会清空串行通信的接收缓冲区，然后再发送 RTU 帧并打开串行中断监听串行链路上的 Modbus 服务器反馈的应答帧。串行中断服务程序在收到应答帧以后会以信号量的形式通知 T_modbusServer 任务进行处理。若网关没有收到串行链路上的应答帧，则在等待一段时间以后进行超时处理。超时的长度通常由用户决定。第四，Modbus 协议采用“大端”格式表示地址和数据，而 AT91RM9200 微处理器采用了小端格式，因此对用到的一些 2 字节长度的字段要进行专门的处理，比如网关生成的 CRC 校验码为 0xA367，那么在将 CRC 校验码添加到 RTU 帧的尾部的时候应该先添加 0x67、然后再添加 0xA3，这样就符合“高位高地址”的大端格式了。

6.3 测试

6.3.1 ARP、IP、ICMP 协议与以太网驱动测试

在移植并测试好 μC/OS-II 操作系统系统的基础上，进行了 LwIP 的移植并设计了以太网驱动程序，这些步骤都完成之后就需要进行测试以确保移植的正确性。这里测试了嵌入式网关对 ARP 请求的应答和 ICMP 协议的回应应答。

首先设置嵌入式网关的 IP 地址为 192.168.2.9，MAC 地址为 00 00 00 00 01 01，然后用网线将网关与交换机相连，通过连接在交换机上的其他计算机向网关发送 ping 命令。计算机向整个网络发送 ARP 请求，网关对该请求作出 ARP 应答。然后计算机就得到了网关 IP 地址所对应的 MAC 地址，并利用 MAC 地址发送 ICMP 协议的回应请求，网关作出回应应答。ping 命令的执行结果如图 6-6。在图中可以看到 ping 命令从发送请求到收到响应的时间稳定在 6 毫秒，这个时间的长短与线路状况和网关本身的响应能力有关。事实上适当提高网关的 μC/OS-II 操作系统的时钟节

拍频率可以提高响应能力。ARP 协议的测试结果可以在计算机的控制台下用 DOS 命令 arp –a 来显示，测试结果如图 6-6。

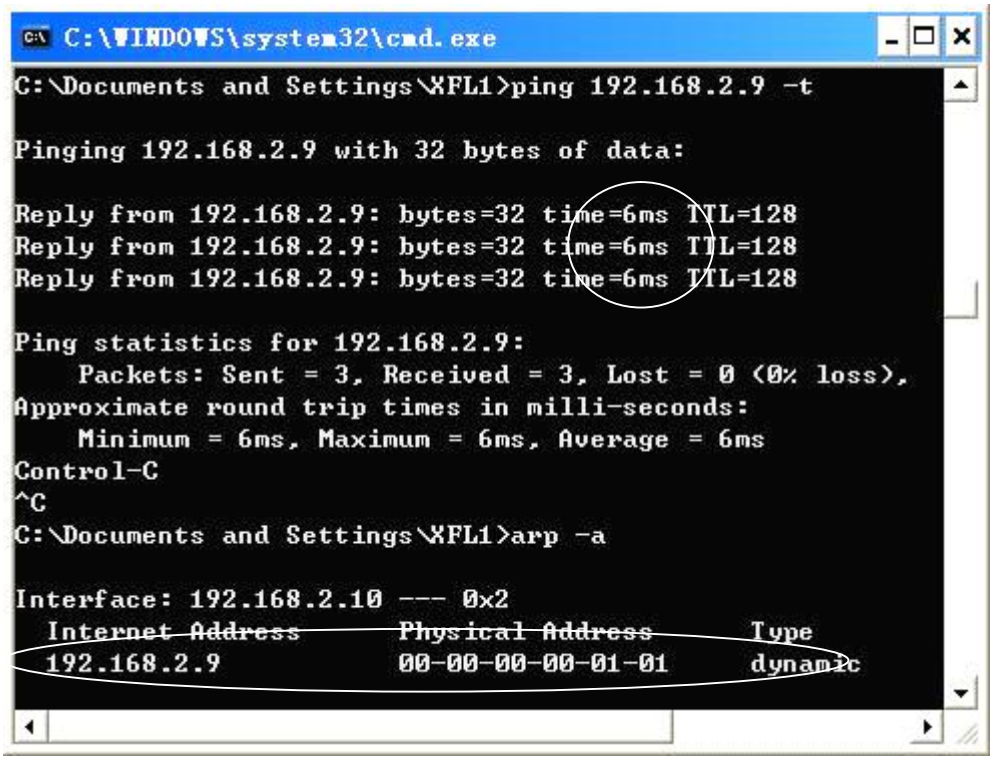


图 6-6 ping 命令的执行结果

6.3.2 TCP、HTTP 协议测试

在确保 IP 协议和以太网驱动程序正确的前提下，可以对 TCP 协议和 HTTP 协议进行测试。测试的方法是通过连接在交换机上的计算机访问嵌入式网关的 Web 页面。网关收到请求后发送一个非常简单的“HELLO”测试页面。在这里利用安装在计算机上的 Commview V4.0 软件捕捉交互的全部过程，包括 TCP 连接的三次握手、网页的发送。过程如图 6-7。图中被选中的一行就是被捕捉到的网关发送页面的数据包，其中最后一列 Delta 是指该数据包与上一个数据包的间隔时间，单位是秒。通过计算可以得知网关对 HTTP 请求的响应时间在 0.2 秒左右。

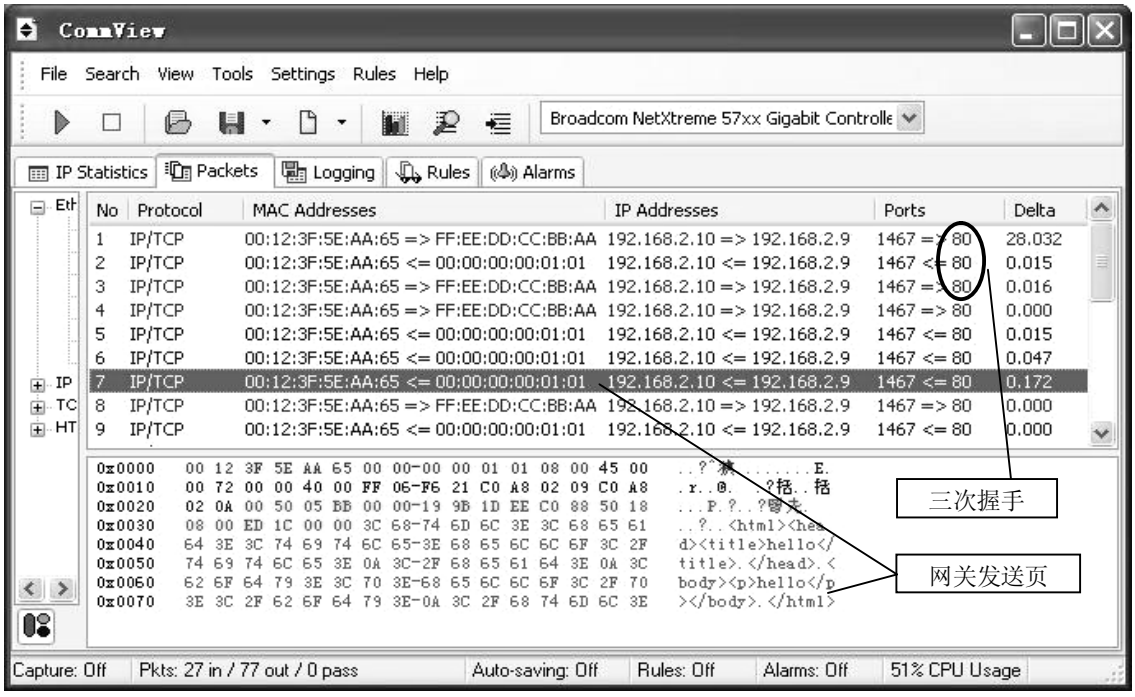


图 6-7 TCP 协议、HTTP 协议测试过程

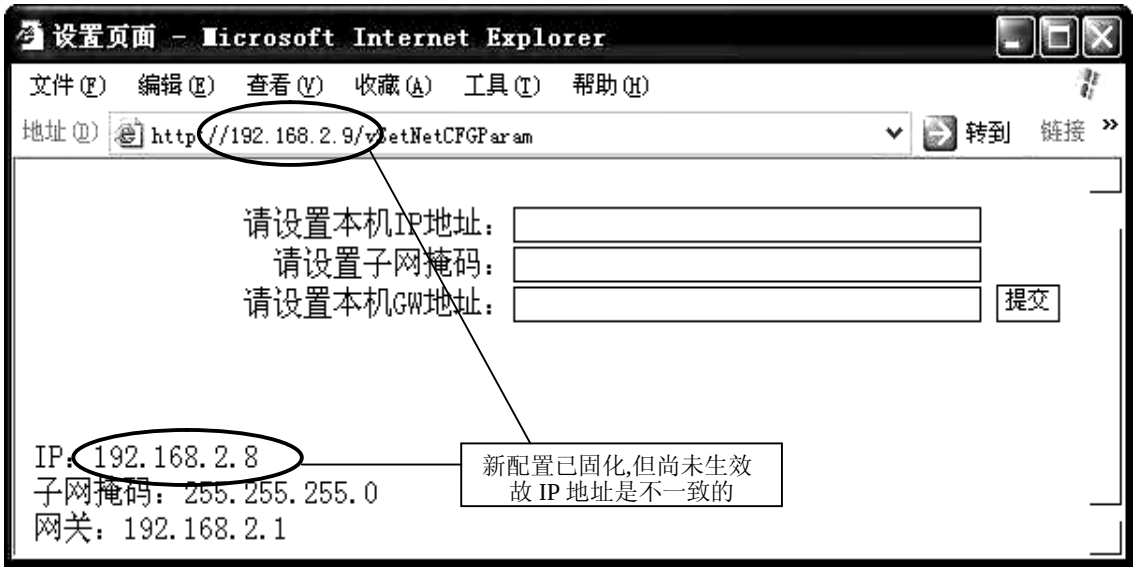


图 6-8 重新发送的表单页面

6.3.3 Web 服务测试

利用 Web服务实现对嵌入式网关 IP 地址的设置需要经过两次请求响应的过程。第一次请求响应时网关将固化在片外 FLASH 中的 HTTP 表单页面和网关的配置信

息返回给计算机。具体过程与 TCP、HTTP 协议的测试过程类似，显示页面如图 6-2。用户在表单页面中填写配置信息后点击提交按钮，网关对用户的第二次请求消息进行处理，将用户提交的配置信息烧写到网关的片外 FLASH 中进行保存，并重新发送表单页面进行第二次响应。在图 6-8 中可以看到网关的 IP 地址发生了变化。网关重启之后，新的配置信息即生效。

6.3.4 Modbus/TCP 协议转换功能测试

本课题设计的嵌入式 Modbus/TCP 协议网关从以太网 RJ45 接口接收 Modbus/TCP 客户机发送的 Modbus/TCP 请求帧，对其进行解帧和封装之后通过 RS232 接口发送到串行链路上，同时等待从 RS232 接口接收 Modbus 服务器的应答帧，对其进行解帧和封装之后从以太网 RJ45 接口发送出去。测试环境如图 6-9。

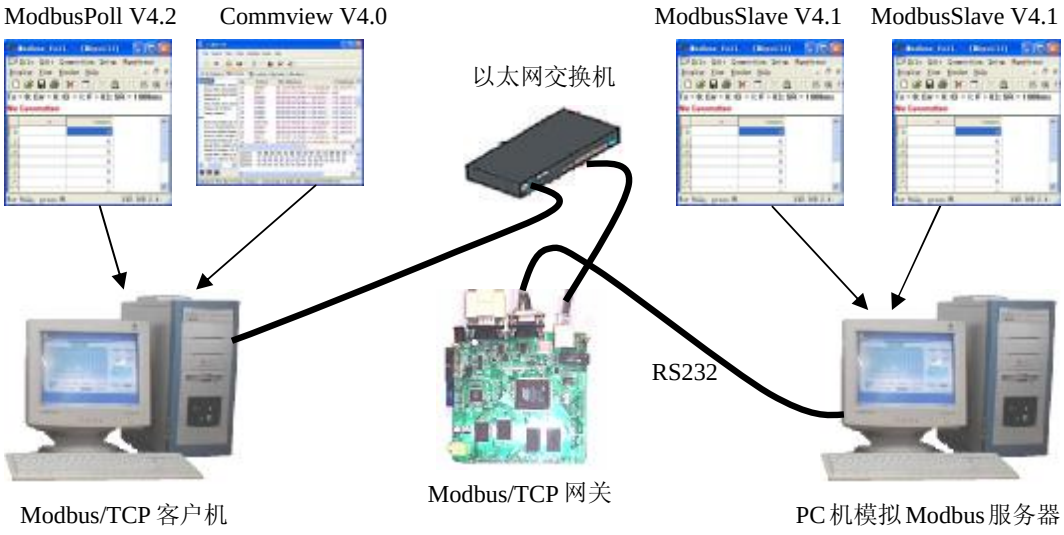


图 6-9 嵌入式 Modbus/TCP 网关测试环境

为了能比较公正和客观地对网关进行测试，课题没有自行设计测试软件而是采用了第三方的 Modbus 客户机测试工具 ModbusPoll V4.2 和服务器测试工具 ModbusSlave V4.1。为了模拟多台 Modbus 服务器的情况，采用在一台 PC 机上同时运行多个 ModbusSlave 程序，并赋予不同的设备地址和设备功能状态。通过在 Modbus/TCP 客户机运行的 Commview 软件捕捉 Modbus/TCP 客户机与 Modbus 服务器请求响应的全部过程和响应的时间。

Modbus/TCP 客户机的 IP 地址设置为 192.168.2.10，ModbusPoll 设置对方 IP 地址为 Modbus/TCP 网关地址即 192.168.2.9，采用定时扫描，扫描间隔为 1000 毫秒。

用 PC 机模拟了三台 Modbus 服务器，串行通信的波特率设置为 9600bps，无校验位，停止位为 1。第一台地址为 1，功能代码为 03，功能是读输出寄存器。第二台服务器的设备地址为 2，功能代码为 01，功能是读输出状态。第三台地址为 03，功能代码为 06，功能是设置单寄存器状态。

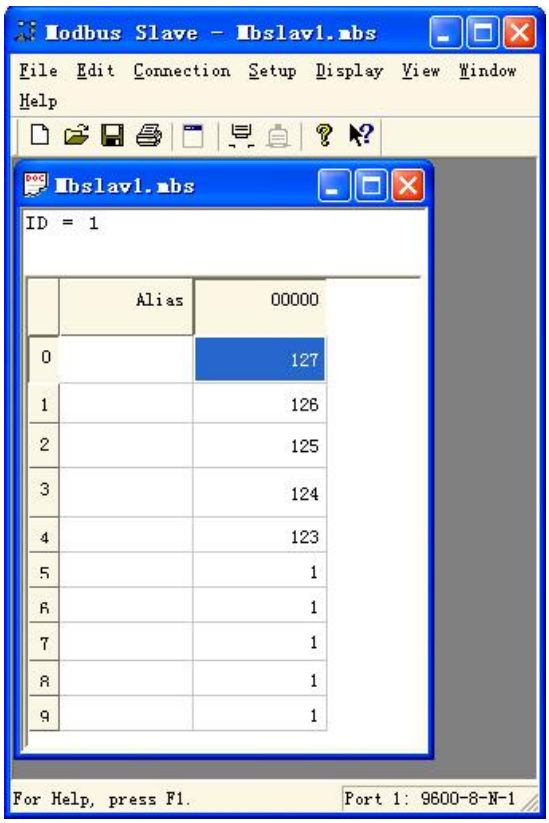


图 6-10 Modbus 服务器运行界面

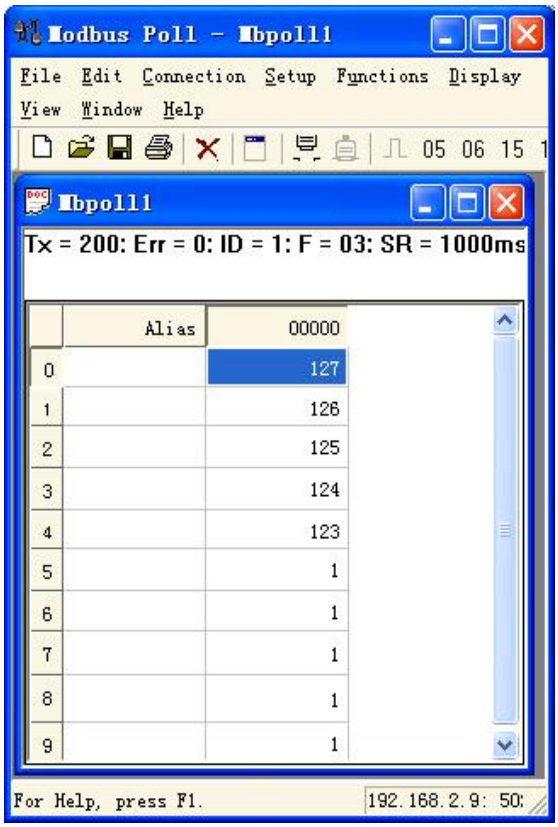


图 6-11 Modbus/TCP 客户机运行界面

测试时由 Modbus/TCP 客户机分别去查询串行通信链路中的三台 Modbus 服务器。本文以查询一台 Modbus 服务器为例。图 6-10 是串行通信链路中的地址为 1 的 Modbus 服务器 ModbusSlave 的运行界面。这里可以看到 Modbus 服务器有 10 个寄存器，并且每个寄存器都有相应的值。图 6-11 是 Modbus/TCP 客户机的 ModbusPoll 的运行界面。在图中可以看到客户机能及时查询到服务器的寄存器状态值并且连续查询 200 次也没有发生查询失败。事实上，网关连续工作 48 小时各部分均正常，主控芯片仅微微发热。网关直流 5V 电源输入电流为 55mA，总功率仅为 0.275W。图 6-12 显示了 Commview 捕捉到的 Modbus/TCP 客户机发送的 Modbus/TCP 的查询帧，图 6-13 显示了 Commview 捕捉到的 Modbus/TCP 客户机接收的 Modbus/TCP 的

响应帧。根据 Commview 捕捉到的数据包的最后一列显示的间隔时间可以计算出 Modbus/TCP 客户机一次查询响应的实际时间约为 110 毫秒。

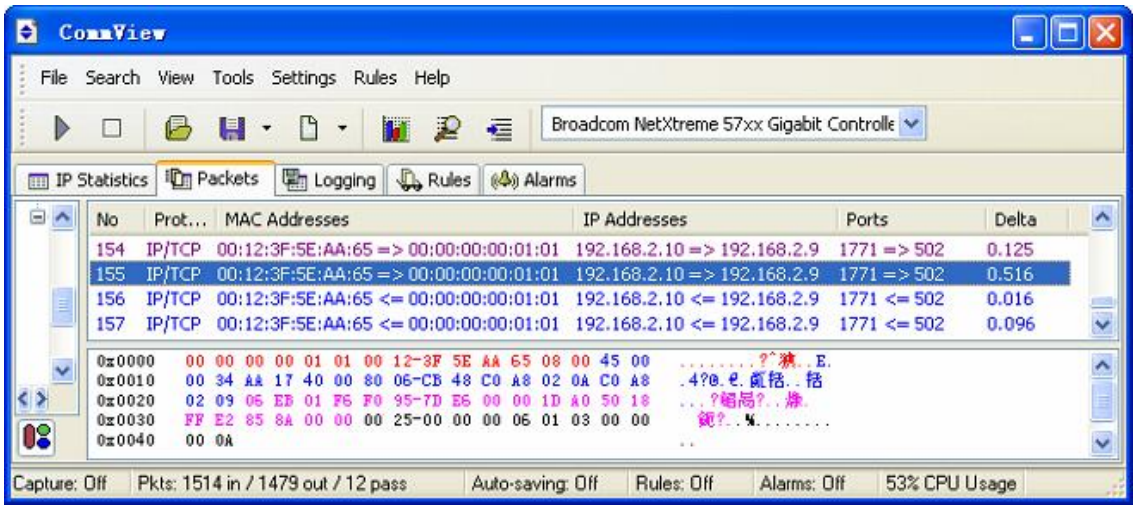


图 6-12 客户机发送的 Modbus/TCP 查询帧

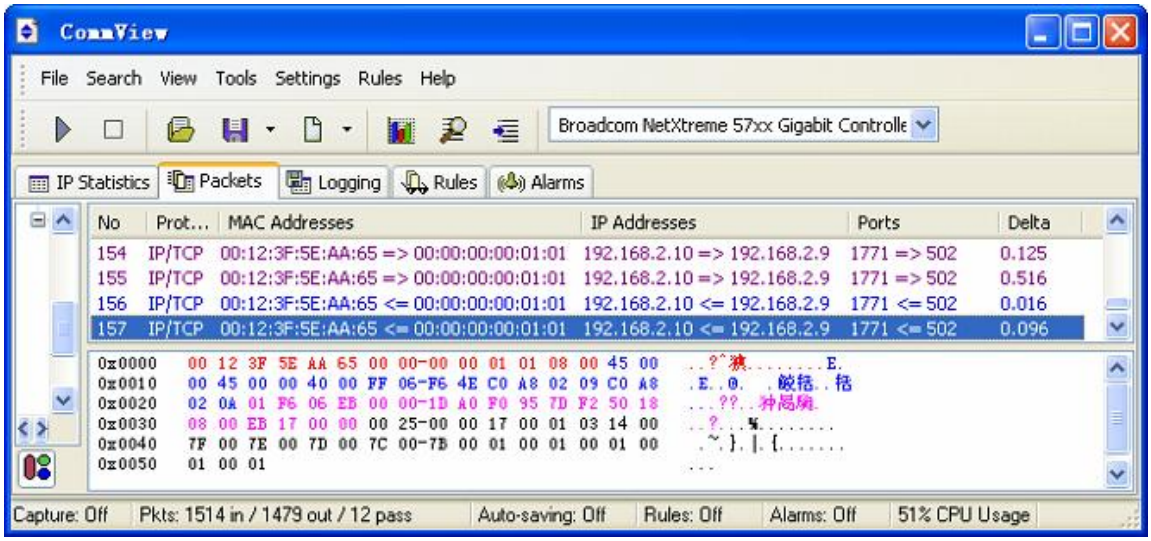


图 6-13 Modbus/TCP 客户机接收的 Modbus/TCP 响应帧

6.4 应用案例

本课题源于一家生产空调控制器的厂家。该厂家生产的控制设备支持 RS232/RS485 串行通信链路上的 RTU 帧格式的 Modbus 协议。每一个控制器都包含有微控制芯片并执行一定的功能，如水流阀门的开启和闭合，控制机组的通电和断电，对系统温度、环境温度的采集，对外部信号的响应和反馈等等。这些控制信息都可以用 Modbus 协议提供的四种数据模型即开关量输入、开关量输出、寄存器输

入、寄存器保持来表示。对这些控制设备也就是 Modbus 服务器的控制和监视都是通过连接在串行通信链路中的 Modbus 客户机来实现的。同一个串行通信链路中的设备被分配到唯一的地址。在这里使用到的 Modbus 功能代码如表 6-7。

表 6-7 需要使用的 Modbus 功能代码

功能代码	命令	功能	备注
01	读输出状态	读取开关型变量	询问一组开关型变量当前值
02	读输入状态	读取开关型变量	询问一组开关型变量当前值
03	读输出寄存器	读取模拟型/整数型变量	询问一个或多个模拟型变量当前值
04	读输入寄存器	读取模拟型/整数型变量	询问一个或多个模拟型变量当前值
05	设置单开关状态	写开关型变量	设置某单独开关型变量为 ON 或 OFF 值
06	设置单寄存器状态	写模拟型/整数型变量	设置某模拟型变量为专用值
15	设置多开关状态	多重写开关型变量	设置连续的开关型变量为 ON 或 OFF 值
16	设置多寄存器状态	多重写模拟型/整数形变量	设置连续的模拟型变量为专用值

随着企业管理信息化的迅速发展，越来越多的工厂和企业拥有了自己的信息管理网络。有用户向厂家提出将自己的中央空调系统的管理纳入到企业信息管理系统统一的框架里面，这样就能更加方便地对运行的设备进行监测，同时也有利于提高设备使用效率，减少支出。为满足客户提出的需要，厂家需要对现有的网络进行改造和升级，考虑的方案有两种：

第一种方案是将客户所有串行链路通信的 Modbus 设备全部改成用以太网通信的 Modbus/TCP 设备，用 Modbus/TCP 工业以太网协议实现控制设备的联网设置和监控，并直接连入用户的企业信息管理网络。这种方案比较适合第一次安装空调控制系统的客户，而对于老用户来说意味着需要淘汰原有的设备和网络布线，成本高昂，用户也难以接受。

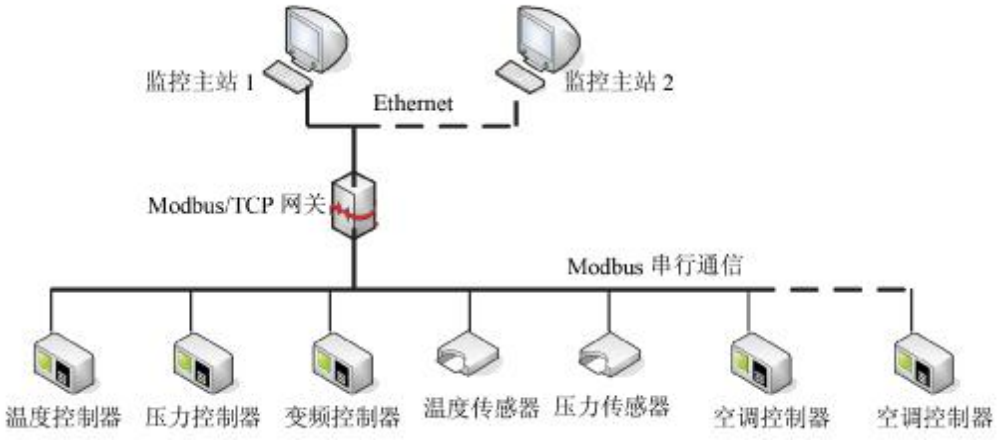


图 6-14 Modbus 串行通信设备与以太网互连方案二

第二种方案是在保留现有的控制设备和布线网络的情况下，增加一个协议转换网关即本课题实现的嵌入式 Modbus/TCP 网关，方案如图 6-14。利用 Modbus/TCP 协议将 Modbus 串行通信链路与以太网相连，配合 Modbus/TCP 客户端设备(既可以在 PC 机上安装 Modbus/TCP 客户端软件也可以安装运行 Modbus/TCP 客户端程序的嵌入式设备)，用户就可以通过企业的信息网络对整个空调系统进行监控。这种方案最大程度上保留了原有的设备，既满足了用户升级的愿望又减少了成本的支出。当然这种方案也适用于那些希望先使用 Modbus 串行通信链路设备等待以后条件成熟再连入以太网的新用户。

事实上该控制器生产厂家也非常希望涉足 Modbus/TCP 工业以太网领域，开发出自己的 Modbus/TCP 服务器端和客户端设备，对产品进行创新，最大程度地满足用户新的需求。但是新产品的开发需要一个较长的过程，因为原来串行链路中设备的控制芯片不带有以太网连接功能，因此要根据设备实现的具体功能重新选择设备的控制芯片，制作电路板，开发新的程序，最后还要进行严格的测试，这都需要大量的时间和财力，本课题的实现对此作出了初步的探索。

6.5 小结

本章在详细分析 Web交互过程的基础上给出了嵌入式 Web服务器的实现过程，并就如何从用户的 HTTP 消息中的提取网关的配置信息以及将该信息进行保存提出了具体的解决方案。本章还对 Modbus 协议的通信模式和数据帧进行研究，给出了 Modbus/TCP 帧与 Modbus RTU 帧的转发方法以及转发过程中需要注意的问题。然后通过第三方测试软件对嵌入式 Modbus/TCP 网关进行了测试，测试结果证明系统的设计方案是正确可行的。最后给出了嵌入式 Modbus/TCP 网关的应用案例。

第七章 总结和展望

7.1 总结

本文针对将工业以太网应用到现场控制网络这一研究热点，设计了一种嵌入式协议转换网关，通过 Modbus/TCP 协议将传统的串行链路通信的 Modbus 现场总线与以太网相连，使用户可以通过以太网对现场的设备进行监控。

笔者独立完成了以下工作：

① 在对嵌入式技术进行广泛而深入的研究之后，制定了本文的总体设计方案。

② 针对工业控制领域的应用选择了主流的 ARM920T 内核的 32 位高性能工业级嵌入式微处理器作为硬件系统的核心，并完成硬件电路的设计、焊接和调试。

③ 建立了一套高效便捷的软件开发环境，在此基础上设计了 μC/OS-II 操作系统的初始化程序并移植 μC/OS-II 操作系统。

④ 在对几种主流的开放源代码的 TCP/IP 协议栈进行比较的基础上，选择并实现了 LwIP。

⑤ 设计实现了嵌入式 Web 服务功能，实现了 Web 方式下网关配置信息的更新与固化；设计实现了 Modbus/TCP 的协议转换功能，给出了工作流程以及程序设计的要点；最后对网关进行了测试，测试结果证明系统设计方案正确可行。

本文的特色：

① 本文对硬件系统的核心－微处理器做了充分的比较，没有使用功能单一、性能较弱的 8 位、16 位微处理器如 51 系列单片机等，而是采用从低端市场到高端领域都获得广泛使用的 ARM 核心的微处理器。在具体的芯片选型时针对工业级应用，采用了基于 ARM920T 核心的 AT91RM9200 高性能工业级芯片，性能大大优于被广泛采用的 ARM720T 核心的商业级芯片如 S3C44B0 以及已经停产的 S3C4510 等芯片。

② 配合 AT91RM9200 的 EMAC 控制器选用 10M/100M 以太网物理层芯片，取代了以往的那种微处理器内部不含有以太网控制器而依靠外部总线挂接以太网控制器芯片的设计方案，既简化了连接电路又提高了网络传输速度。最典型的以太网控制芯片是 RTL8019 但连接速度只有 10Mbps。

③ 采用体积小巧、安全可靠的 μC/OS-II 操作系统，并配合开放源代码的 TCP/IP 协议栈 LwIP，可以非常方便地将应用层程序移植到其他 ARM 核心的微处理器乃至其他体系架构的微处理器上，具有极佳的通用性和可移植性。

④ 在深入分析了 LwIP 的基础上提出了 LwIP 以太网驱动程序的设计方法，为片内含有 EMAC 控制器的微处理器的驱动程序设计提供了参考。

⑤ 提出了一种通过 Web 服务方式获取用户提交的信息并将信息保存到非易失性存储器的设计方案，使产品具有更多的实际应用价值。

⑥ 对 Modbus/TCP 这种“年轻”的工业以太网协议进行了研究并着重给出了它与 Modbus 串行通信链路的协议转换方法。

7.2 展望

本文设计实现的嵌入式 Modbus/TCP 网关目前已经完成了实验室阶段的测试，将其转化成具体的产品还有进一步的工作要做，其中包括对系统实时性进行更深入的研究与测试；提高系统的抗干扰能力；对系统功能做进一步完善。

在本文实现的嵌入式 Modbus/TCP 网关的基础上对应用层软件加以修改就能使其成为功能强大的 Modbus/TCP 工业以太网客户机或服务器。同时市场上种类繁多的 ARM 核心的微处理器可以满足各个层次的需求，售价便宜、功能单一，性能参数较低的产品可以选用相对低端的微处理器，反之则选用高端的微处理器。

可以看到工业以太网技术应用前景非常广阔，能够将企业的“会议室”一直延伸到“生产现场”，并建立起一个统一的应用层和用户层的标准协议，同时消除各种现场总线协议之间的差异。虽然以太网完全取代现场总线还需要很长的一段时间，但随着工业以太网技术的成熟和大量产品的推出，以太网必将贯穿整个企业管理网络和底层的控制网络。本文在这背景下进行了一定程度的探索，并希望本文能为这种发展潮流起到了一定的推动作用。

参考文献

[1] 汤龙梅.CAN 总线与以太网互连系统设计与应用[D].苏州大学,2005.

[2] 李正军.现场总线及其应用技术[M].机械工业出版社,2005.

[3] 阳宪惠,徐用懋,魏庆福.现场总线技术及其应用[M].清华大学出版社,1999.

[4] 缪学勤 .新版 IEC61158 现场总线标准概论 [J].仪器仪表标准化与计量,2004,(2):18-25.

[5] 魏庆福.现场总线技术的发展与工业以太网综述.工业控制计算机,2001,(1):20-24.

[6] 王裕如.工业以太网在嵌入式系统中的应用[D].大连铁道学院,2003.

[7] 冯冬芹 ,金建祥 ,褚健 .工业以太网关键技术初探 [J].信息与控制 ,2003,32(3):219-224.

[8] 冯冬芹,金建祥,褚健.Ethernet 与工业控制网络[J].仪器仪表学报,2003,24(1):23-26.

[9] 王荣莉,雷斌.工业以太网技术的现状与发展[J].自动化博览,2004,(4):63-65.

[10] 佟为明,刘勇,赵志衡.几种主流工业以太网[J].低压电器,2005,(6):41-42.

[11] 杨启彬.现场总线控制系统通信机理的研究[D].河海大学,2002.

[12] 吴献.RS-232 通信接口芯片在 0.6um 工艺下的设计与实现[D].浙江大学,2006.

[13] 马骏骄.基于 RS-485 总线的实验教学系统研究[D].大连理工大学,2004.

[14] 边海龙,贾少华.USB2.0 设备的设计与开发[M].人民邮电出版社,2004.

[15] 华镕.开放的 MODBUS TCP(连载二)[J],自动化博览,2005,(2):73-74.

[16] 邓欣茹,丁建兴,杨翼等.Modbus/TCP 工业以太网的现 状与发展[J],2004,17(9):14-15.

[17] 厚重历史 透明未来 [EB/OL].http://www.ca800.net/news/html/2007-1-18/n34486.html,2005.11.23.

[18] Craig Hollabaugh 著.陈雷,钟书毅译.嵌入式 Linux-硬件、软件与接口[M].电子工业出版社,2003.

[19] 何立民 .嵌入式系统的 定义 与发展 历史 [J].单片 机与嵌入式系统应用,2004,(1):4-8.

[20] 陈文智.嵌入式系统开发原理与实践[M].清华大学出版社,2005.

[21] 林建民.嵌入式操作系统技术发展趋势[J].计算机工程,2001,27(10):1-4.

[22] 邵贝贝.单片机嵌入式应用的在线开发方法[M].清华大学出版社,2004.

[23] Steve Furber.ARM System-on Chip Architecture[M].Boston Pearson Education Limited,2000.

[24] 罗蕾.嵌入式实时操作系统及应用开发[M].北京航空航天大学出版社,2005.

[25] Andrew N.Sloss,Dominic Symes,Chris Wright 著,沈建华译.ARM 嵌入式系统开发-软件设计与优化[M].北京航空航天大学出版社,2005.

[26] 陈赜.ARM9 嵌入式技术及 Linux 高级实践教程[M] 北京航空航天大学出版社,2005.

[27] 陆松年.操作系统教程[M].电子工业出版社,2000.

[28] 林纲.嵌入式操作系统系统的实时性的研究[D] 浙江工业大学,2005.

[29] 宋延昭.嵌入式操作系统介绍及选型原则[J].工业控制计算机,2005,18(7):41-42.

[30] 王金刚.基于 VxWorks 的嵌入式实时系统设计[M].清华大学出版社,2004.

[31] 王学龙.嵌入式 VxWork、系统开发与应用[M].人民邮电出版社,2003.

[32] 季志均,马文丽.四种嵌入式实时操作系统关键技术分析[J].计算机应用研究,2005,22(9):4-8.

[33] 吴非,樊晓光.嵌入式实时操作系统 μC/OS-Ⅱ与 eCos 的比较[J].单片机与嵌入式系统应用,2004,(10):15-17.

[34] Jean J.Labrosse 著,.邵贝贝等译.嵌入式实时操作系统 μC/OS-II(第 2 版)[M].北京航空航天大学出版,2003.

[35] ARM920-based Microcontroller AT91RM9200,ATMEL Corporation Inc[DB/OL]. http://www.atmel.com/,2003.

[36] AT91RM9200 Development Kit User Guide,ATMEL Corporation Inc[DB/OL]. http://www.atmel.com/,2003.

[37] ARM920-based Microcontroller AT91RM9200 Summary, ATMEL Corporation Inc [DB/OL].http://www.atmel.com/,2003.

[38] DM9161E datasheet,DAVICOM Corporation Inc[DB/OL]. http://www.davicom.com/,2003.

[39] 帅辉明.SD9200-ARM9 嵌入式教学实验平台的设计与实现[D].苏州大学,2006.

[40] 李驹光.ARM 应用系统开发详解-基于 S3C4510B 的系统设计[M].清华大学出版社,2004.

[41] 朱义君.AT91 系列 ARM 微控制器体系结构与开发实例[M].北京航空航天大学出版社,2005.

[42] 邸晓鸿. μC/OS-II 在 Nios 上的移植[J].单片机与嵌入式系统应用,2004(9):79-81.

[43] 曾家智.计算机网络[M].电子科技大学出版社,2004.

[44] James F.Kurose Keith W.Ross(美).计算机网络[M].清华大学出版社,2003.

[45] freeBSD 介绍[EB/OL].http://www.freebsd.org/about.html,2006.

[46] 陈渝.李明,杨晔.源码开发的嵌入式系统软件分析与实践-基于 SkyEye 和 ARM 开发平台[M].北京航空航天大学出版社,2004.

[47] LwIP 介绍[EB/OL].http://savannah.nongnu.org/projects/lwip/,2002.

[48] μCIP 介绍[EB/OL].http://ucip.sourceforge.net,2001.

[49] μIP 介绍[EB/OL].http://www.sics.se/~adam/uip/,2006.

[50] gcc 介绍[EB/OL]http://gcc.gnu.org,2007.

[51] W.Richard Stevens著.胡谷雨,吴礼发等译.TCP/IP 详解卷 3:TCP 事务协议、HTTP、NNTP 和 UNIX 域协议[M].机械工业出版社,2006.

[52] James F.Kurose Keith W.Ross 著,申震杰,王金伦,杜江译.计算机网络-自顶向下方法与 Internet 特色[M].清华大学出版社,2002.

[53] AM29LV640B datasheet rev2.0,AMD Corporation Inc[DB/OL].htt://www.amd.com, 2003.

[54] MODBUS Application Protocol Specification V1.1a[DB/OL]. http://www.Modbus-IDA.org,2004.

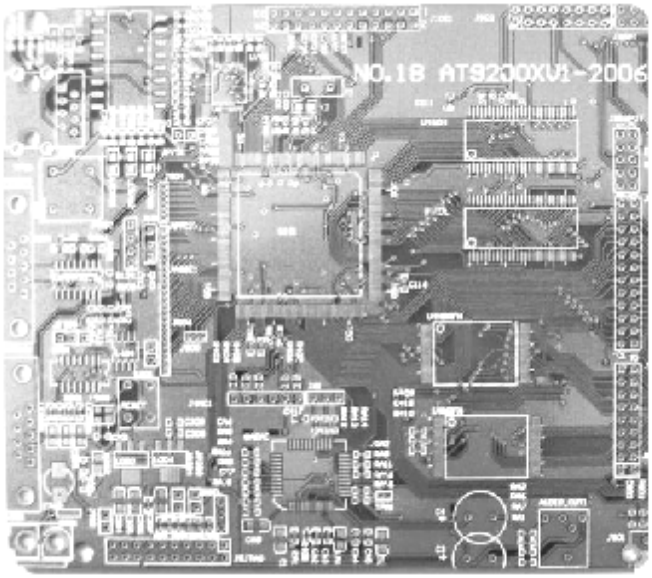
[55] 吴进时.嵌入式网关的设计与实现[D].东北大学,2002.

[56] MODBUS over Serial Line Specification & Implementation guide V1.0[DB/OL]. http://www.Modbus-IDA.org,2004.

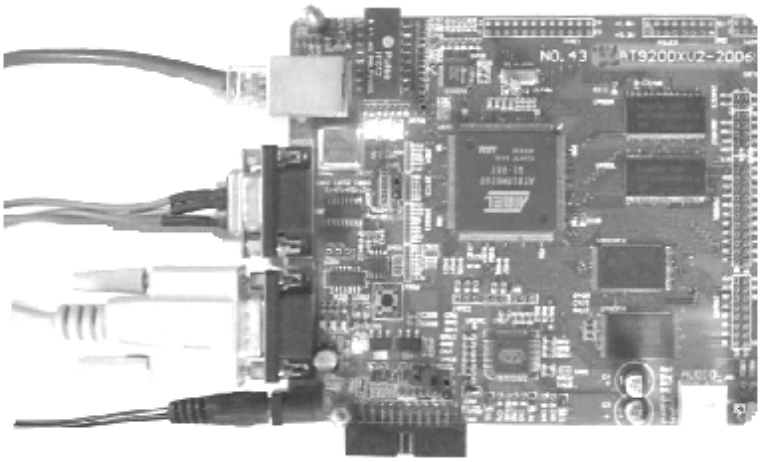
[57] MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0.a [DB/OL].http://www.Modbus-IDA.org,2004.

附录 A 嵌入式 Modbus/TCP 网关 PCB 及实物图

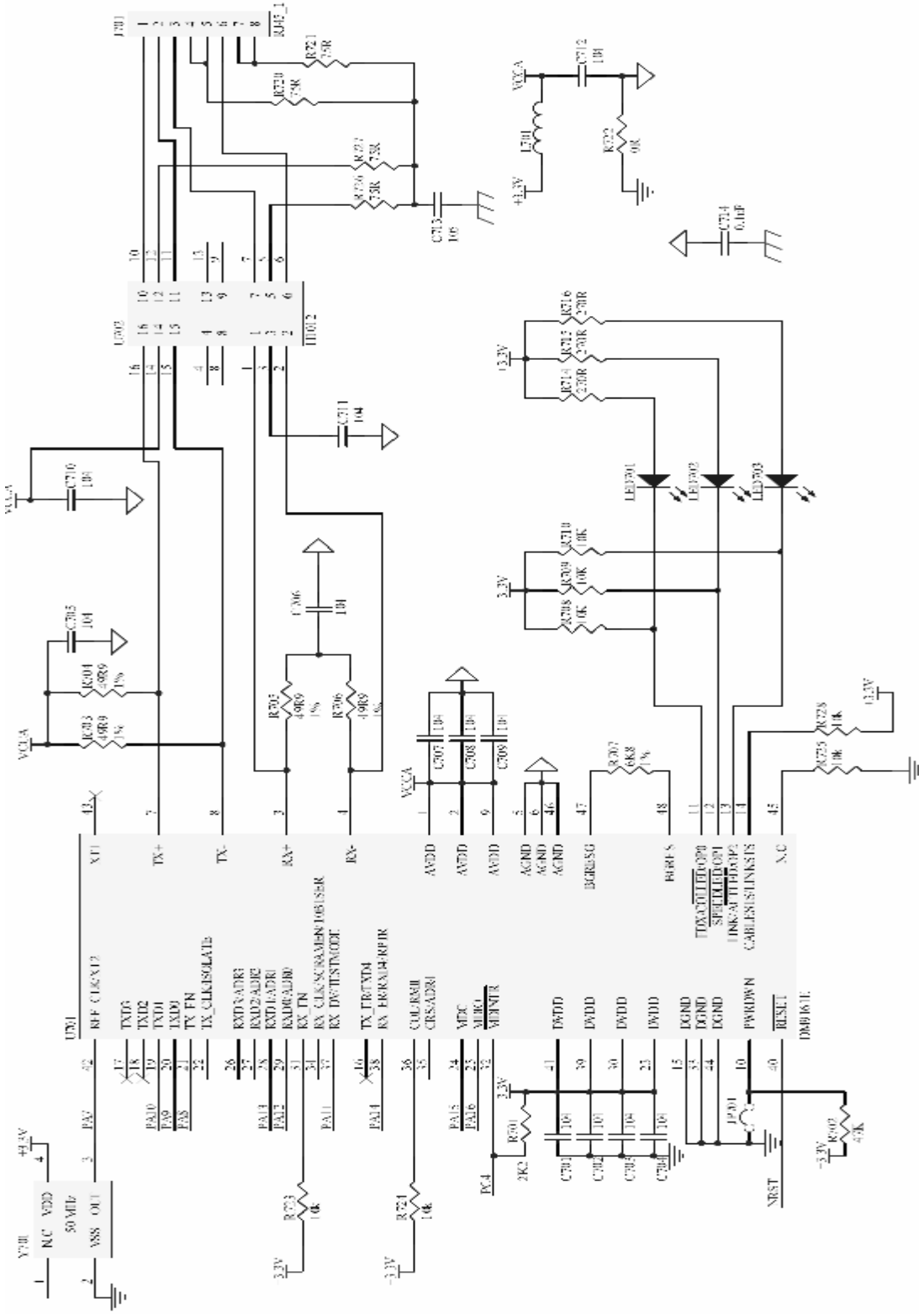
A.1 嵌入式 Modbus/TCP 网关 PCB 图



A.2 嵌入式 Modbus/TCP 网关实物图



附录 B 以太网物理层芯片 DM9161 原理图



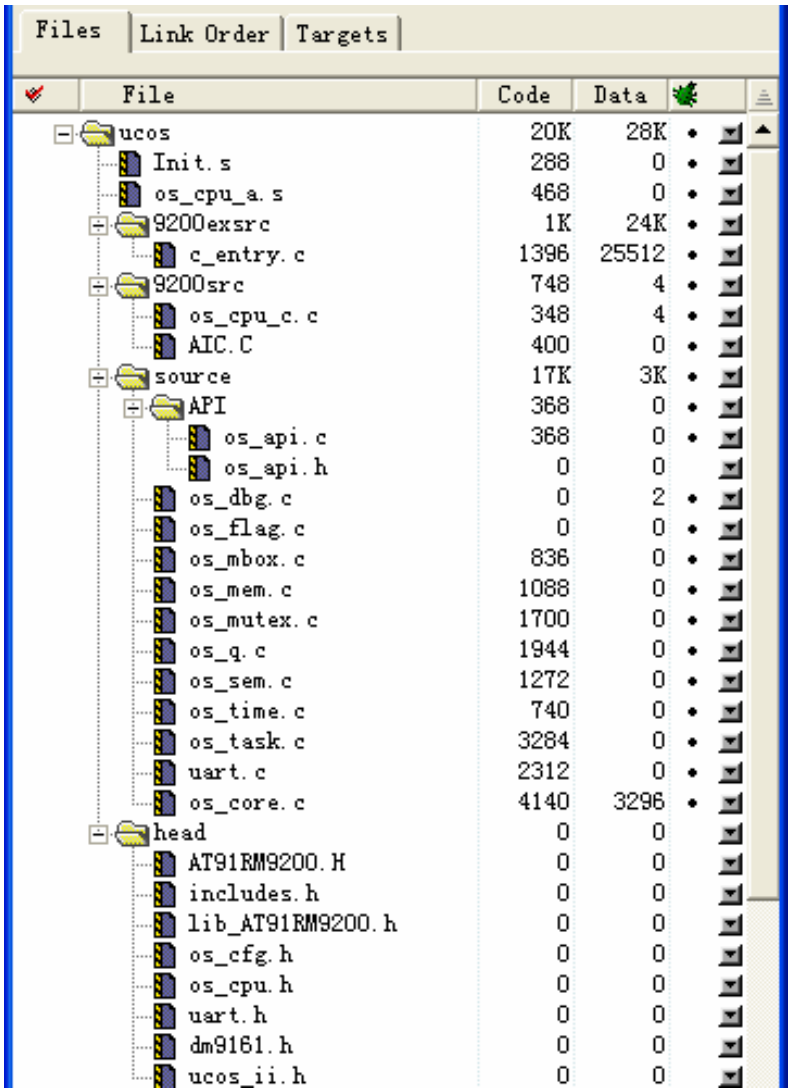
附录 C Web 服务任务和 Modbus/TCP 任务代码

```
//*--------------------------------------------------------------------------------------------------
//*  函数名称 : T_httpServer
//*  功能描述 : Web 服务任务
//*  入口参数 : <pReserved>[in]  保留
//*  出口参数 :  无
//*--------------------------------------------------------------------------------------------------
void T_httpServer(void *pReserved) {
    extern void Handler_HTTP(ST_NETCONN *pstConn);
    ST_NETCONN *__pstConn, *__pstNewConn;
    __pstConn = netconn_new(NETCONN_TCP);    //                    建立一个新的 tcp 类型的数据连接
    netconn_bind(__pstConn, NULL, HTTP_PORT);    //                将连接绑定在本地 TCP 的 80 端口
    netconn_listen(__pstConn);       //                            监听端口
    while(TRUE)    {
            __pstNewConn = netconn_accept(__pstConn);   //        接受客户端连接请求
       if(__pstNewConn != NULL)   {
          Handler_HTTP(__pstNewConn);    //                       调用函数处理连接请求
          while(netconn_delete(__pstNewConn) != ERR_OK) //        删除此次连接
             OSTimeDlyHMSM(0, 0, 0, 500);
       }
    }
}
//*--------------------------------------------------------------------------------------------------
//*  函数名称 : T_ModbusServer
//*  功能描述 : modbus/tcp 协议转换任务
//*  入口参数 : <pReserved>[in]  保留
//*  出口参数 :  无
//*--------------------------------------------------------------------------------------------------
void T_ModbusServer(void *pReserved) {
    extern void Handler_modbus(ST_NETCONN *pstConn);
    INT32U start_ticks,end_ticks;
    INT8U flag = 0;
    ST_NETCONN *__pstConn_modbus,*__pstNewConn_modbus;
    __pstConn_modbus = netconn_new(NETCONN_TCP);     //           建立一个新的 tcp 类型的数据连接
    netconn_bind(__pstConn_modbus, NULL, 502);          //        将连接绑定在本 TCP 的 502 端口上
    netconn_listen(__pstConn_modbus);       //                   监听端口
        while(TRUE)    {
      __pstNewConn_modbus =modbus_tcp_netconn_accept(__pstConn_modbus); //     接受客户端连接请求
      if(__pstNewConn_modbus != NULL)   {
          Handler_modbus(__pstNewConn_modbus);      //           调用函数处理连接请求
          start_ticks = OSTimeGet();     // OSTimeGet(void)                  获得系统时间
          flag = 1;
      }
      end_ticks = OSTimeGet();
      if(((end_ticks - start_ticks)) > 200&&(flag == 1))   {
        while(netconn_delete(__pstNewConn_modbus) != ERR_OK) //       超时删除连接
        OSTimeDlyHMSM(0, 0, 0, 500);
      }
    }
}
```

附录 D　文件列表及各文件编译后的代码量

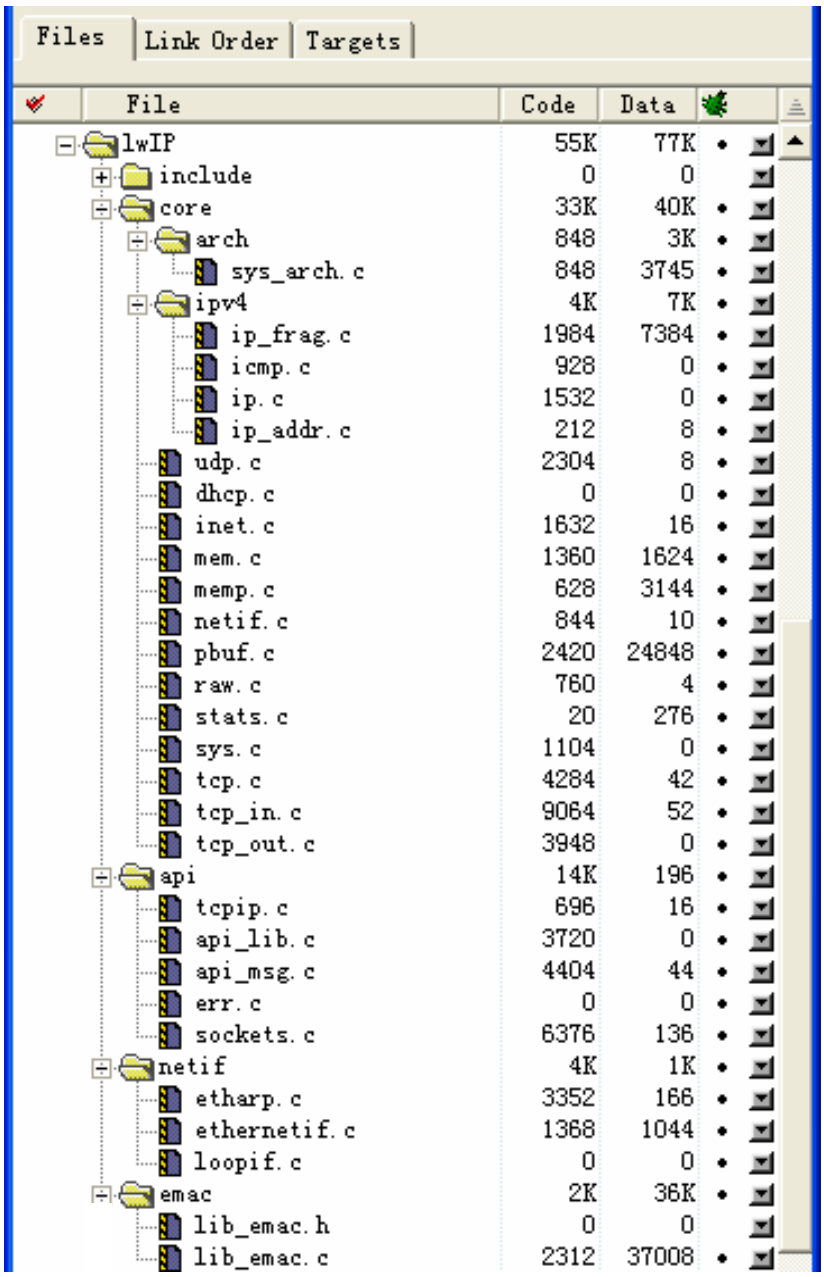
嵌入式 Modbus/TCP 网关主要由四部分程序组成：嵌入式操作系统 μC/OS-II 程序、LwIP 程序、应用层程序和其他程序。ADS 集成开发工具将各程序中的文件进行编译、链接以后会形成一个在网关的硬件系统上直接可以运行的二进制可执行文件。ADS 集成开发环境中的工程管理窗口会在各个文件的右侧列出该文件编译后产生的代码段(Code)和数据段(Data)的大小。下面将列出各部分程序及其文件。

D.1 嵌入式操作系统 μC/OS-II 程序

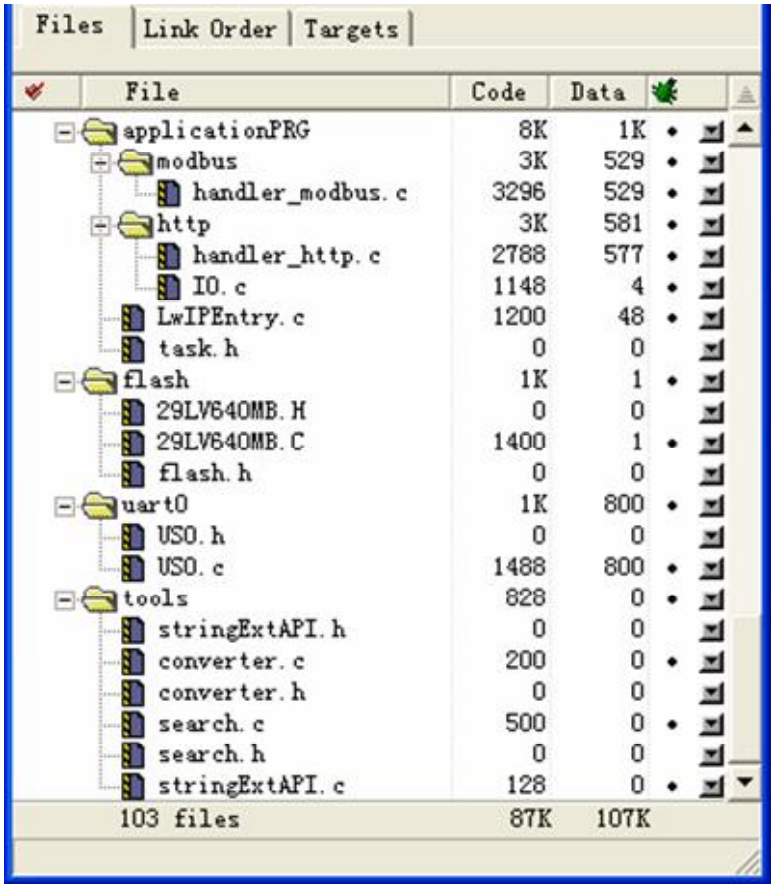
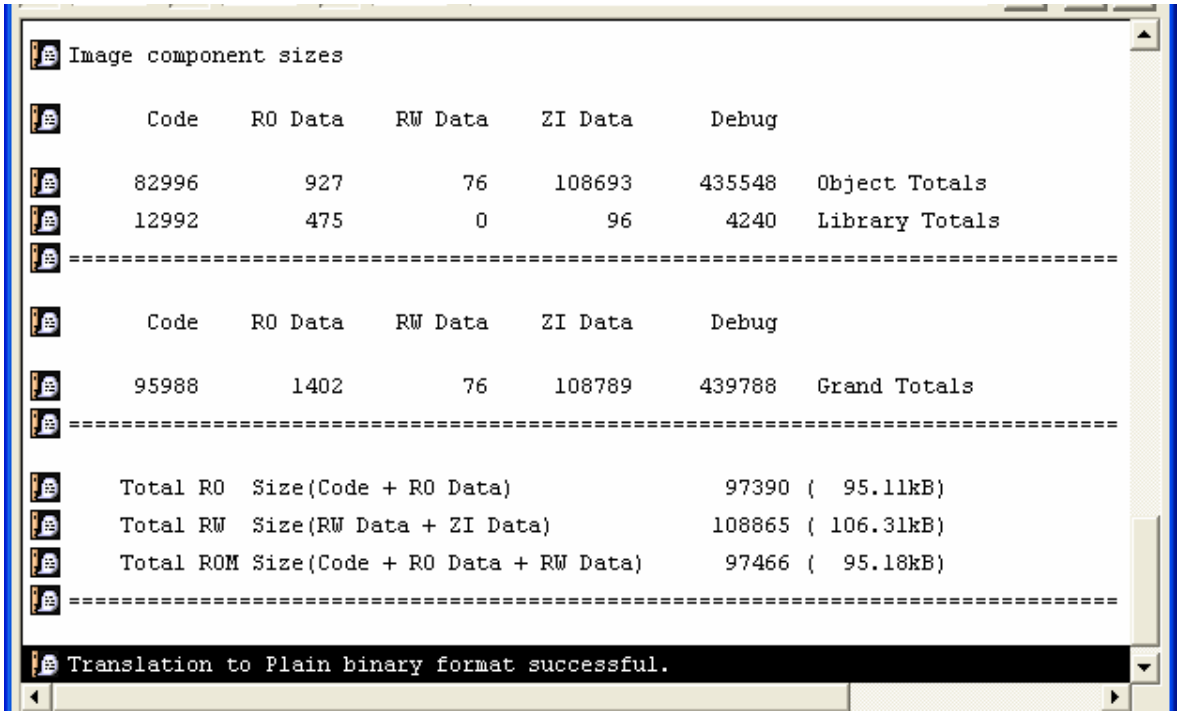


File	Code	Data
ucos	20K	28K
Init.s	288	0
os_cpu_a.s	468	0
9200exsrc	1K	24K
c_entry.c	1396	25512
9200src	748	4
os_cpu_c.c	348	4
AIC.C	400	0
source	17K	3K
API	368	0
os_api.c	368	0
os_api.h	0	0
os_dbg.c	0	2
os_flag.c	0	0
os_mbox.c	836	0
os_mem.c	1088	0
os_mutex.c	1700	0
os_q.c	1944	0
os_sem.c	1272	0
os_time.c	740	0
os_task.c	3284	0
uart.c	2312	0
os_core.c	4140	3296
head	0	0
AT91RM9200.H	0	0
includes.h	0	0
lib_AT91RM9200.h	0	0
os_cfg.h	0	0
os_cpu.h	0	0
uart.h	0	0
dm9161.h	0	0
ucos_ii.h	0	0

D.2 LwIP 程序


Files
Link Order
Targets
File
Code
Data
lwIP 55K 77K
include 0 0
core 33K 40K
arch 848 3K
sys_arch.c 848 3745
ipv4 4K 7K
ip_frag.c 1984 7384
icmp.c 928 0
ip.c 1532 0
ip_addr.c 212 8
udp.c 2304 8
dhcp.c 0 0
inet.c 1632 16
mem.c 1360 1624
memp.c 628 3144
netif.c 844 10
pbuf.c 2420 24848
raw.c 760 4
stats.c 20 276
sys.c 1104 0
tcp.c 4284 42
tcp_in.c 9064 52
tcp_out.c 3948 0
api 14K 196
tcpip.c 696 16
api_lib.c 3720 0
api_msg.c 4404 44
err.c 0 0
sockets.c 6376 136
netif 4K 1K
etharp.c 3352 166
ethernetif.c 1368 1044
loopif.c 0 0
emac 2K 36K
lib_emac.h 0 0
lib_emac.c 2312 37008

D.3 应用层程序和其他程序



D.4 编译后的结果