基于ARM的嵌入式IP电话与遥操作智能家电系统设计

作者简介:邓易冬，男，1981年11月出生，2005年师从贾雨副教授，于2008年6月毕业于成都理工大学测试计量技术及仪器专业并获得工学硕士学位。

摘 要

传统的家电采用各自独立的工作模式，不同家电之间无法通信，这样就不能有效地安排各种家电协同工作，容易造成浪费。同时它们无法自动获取外界的信息，人们无法对其进行远程操作，难以满足现代生活的需求。所以开发智能化的家电及其控制系统已成为当前的研究热点。

传统的电话只能进行语音通信，它存在利用率低、功能有限和安全性不好等缺点。近年来，以ARM为代表的高性能专用微处理器的出现，以及Linux、Windows CE等操作系统的完善，使嵌入式技术迅速发展，这为智能IP电话的研发提供了软硬件基础。

现阶段家庭网关接入互联网的方式主要为有线接入，因为这种方式网络性能比无线隐定，延时性相对要小，用它来远程控制智能家电比无线网要安全可靠。要实现智能家电的网络化，如果采用PC机进行直接进行控制，或者让每台家电接入网络，这样成本很高，不利于一般家庭的普及。

为此，笔者采用基于ARM9芯片、Windows CE 4.2嵌入式操作系统的IP电话作为家电的控制中心，智能家电采用ARM9芯片和linux2.4操作系统。各个智能家电与IP电话采用串口进行通信，IP电话采用网口与因特网通信。这样可以大量的降低成本，而且通信方式比PLC和蓝牙通讯技术更安全可靠。

本文以IP电话与智能家电互联为切入点，结合ARM、嵌入式Linux和网络技术，设计出一种较为完善的IP电话与智能家电的控制系统。采用这种方式，使智能家电集电脑、电信和消费类电子产品的特征于一体，让家电具有信息的获取、加工、传递等功能，提供全方位的信息交换，帮助家电与外部保持信息交流畅通，这样可以优化人们的生活方式，节约能源费用资金。

笔者完成了系统硬件和软件设计，并进行了调试，验证了所设计系统的有效性和实用性。并力争将其拓展成为完善的智能家电控制系统。

关键词：智能家电　遥操作　IP电话　嵌入式系统

The System Design of imbedded IP Phone and Remote Operation Intelligent Electrical Appliances Based on ARM

Introduction of the author: Dengyidong, male, was born in November, 1981 whose tutor was Adjunct Professor Jiayu. He graduated from Chengdu University of Technology in Testing Technology & Instruments major and was granted the Master Degree in June, 2008.

Abstract

The traditional electrical appliances work independently. They are unable to communicate each other, so we cannot arrange each kind of electrical appliances to work effectively.It's easy to result in waste. At the same time, they are unable to gain the outside information automatically, so we are unable to carry on the long-distance operation. It's difficult to satisfy the modern life. Therefore the development of intelligent electrical appliances and their control system have become current research hot spot.

The traditional telephone can only communicate by voice. It has so many shortcomings such as the low use factor, the limited function, the bad security and so on. In recent years, the embedded technology expands rapidly, with the appearance of high performance special-purpose microprocessor such as ARM, as well as Linux operating system's and Windows CE operating system's consummation. This has provided the software and hardware foundation for the intelligent IP Phone's research and development.

At present, the family gateways connects Internet is mainly by wired way, because this manner is stable, has less time delay, is reliable compare with wireless network. To realize the intelligent electrical appliances' network, if we use PC to control intelligent electrical appliances, or let each electrical appliance connects network directly, it must cost so much. So it is hard to popularize the system for the general family.

So the author uses IP Phone, based on the ARM9 chip, Windows ce4.2 embedded operating system, as electrical appliances' control center. The intelligent electrical appliances base on the ARM9 chip and the linux2.4 operating system, each intelligent

electrical appliance carry on the correspondence with IP Phone by serial port. The IP Phone uses NIC to connect Internet. This way can reduce the cost greatly, and is safer than PLC and blue tooth communication.

This article takes the IP telephone and the intelligent electrical appliances interconnection as a breakthrough point, unifies ARM, embedded Linux and the network technology, to design one kind of more perfect IP Phone and intelligent electrical appliances' control system. It causes the intelligent electrical appliances collect computer, telecommunication and consumptive electronic products characteristic in a body, by this method. It let the electrical appliances can gain, process, transfer informations and so on. And can provide omni-directional information exchange. This can help the family and the exterior keep communication unimpeded. So it can optimize the people's life, save fund of energy.

The author has completed the system hardware design and the software design, carried on the debugging, and confirmed the validity and usability of the system. And try to make it be a consummate intelligent electrical appliances control system.

Keywords: Intelligent Electrical Appliance Remote Operation IP Phone Embedded System

独创性声明

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

学位论文作者签名：

2008年 5月 28日

学位论文版权使用授权书

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

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：

学位论文作者导师签名：

2008 年 5 月 28 日

第1章 引 言

1.1 智能家电与嵌入式系统概述

随着社会的发展，人们生活和工作方式在不断的改变，传统家居模式无法满足人们现代生活的需求。因此智能家居成了近年的研究热点。所谓智能家居就是以住宅为平台，兼备建筑、网络通信、信息家电、设备自动化，集系统、结构、服务、管理为一体的高效、舒适、安全、便利、环保的居住环境。它利用先进的计算机技术、网络通讯技术、综合布线技术、将与家居生活有关的各种子系统，有机地结合在一起，进行统筹管理。与普通家居相比，智能家居不仅具有传统的居住功能，提供舒适安全、高品位且宜人的家庭生活空间；还由原来的被动静止结构转变为具有能动智慧的工具，提供全方位的信息交换功能，帮助家庭与外部保持信息交流畅通，优化人们的生活方式，帮助人们合理安排时间，增强家居生活的安全性，甚至为各种能源费用节约资金[1][2]。

智能家居构建有很多方法，国外很多学者采用 PLC 技术、蓝牙技术组网，以 PC 为控制中心。本文提出一种采用嵌入系统实现的方法。所谓嵌入式系统是以应用为中心，以计算机技术、控制技术、通信技术和微电子技术为基础，软硬件可裁剪，适应于对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统，是计算机技术、半导体技术和电子技术在实际应用中相结合的系统[3]。通常，一个完整的嵌入式系统由处理器、外围设备、操作系统和应用程序四部分组成，主要用于实现对其他设备的实时控制、监视、管理、数据处理等各种自动化处理任务。 目前嵌入式技术广泛应用于手持设备、工业控制和医疗电子等。

1.2 智能家电控制方法

在智能家居技术中，智能化家庭控制网络是研究的关键。目前国外用得比较多的方法有三种：

1.2.1 电力载波 PLC 通迅技术

所谓 PLC，即电力线通信，是指利用电力线传输数据和语音信号的一种通信方式。该技术是把载有信息的高频信号加载于电流，然后用电线传输，接受信息的调制解调器再把高频从电流中分离出来，并传送到计算机或电话，以实现信息传递。该技术在不需要重新布线的基础上，在现有电线上实现数据、语音和视频等多业务的承载[2]。

但是，这种方案对于主要是针对北美电网设计的，由于我国电力网，环境恶劣，PLC 通信对电网产生干扰，我国一些地方电力部门禁止使用 PCL 通信。

1.2.2 蓝牙通讯技术

蓝牙通讯是一种低功率短距离的无线连接技术，其设计初衷就是将智能移动电话与笔记本电脑、掌上电脑以及各种数字化的信息设备都能不再用电缆，而是用一种小型的、低成本的无线通信设备连接起来，进而形成一种个人身边的网络，使得在其范围之内各种信息化的移动便携设备都能无缝地实现资源共享。现在国外有学者把它引入到智能家电控制中。蓝牙通讯最大传输距离为 10 米，传输距离有很大的限制，而且不能在多房间进行传输。使用蓝牙技术进行通信的设备，分为“主叫方”和“受取方”。主叫方只能同时与 7 台受取方通信，在家电数量众多的现代家庭中，这一限制影响了家庭控制网络的构建[2]。

1.2.3 基于 Internet 技术的智能家电控制

Internet 技术的成熟，使得很多研究者致力于将该技术引入到智能家电遥操作领域[2]。但因特网不是实时通信网，它采用的分组交换方式存在“时延”问题。“时延”是从信息发出到信息收取经过的时间。因特网传输的为数字编码信号，要把数字化的信号分组、打包，还要用存储—转发的路由方式传送；在接收端还要解码、复原等，因此增加了很多如编码、解码、缓存等时延。如果遇到网路拥挤的情况，等待转发可能导致随机时延，甚至还会造成数据分组丢失。

传统的控制系统中，监督命令和反馈信号都是基于时间变量的。而基于因特网的智能家电遥操作系统的控制端和被控制端很难在时间上保持同步，这样会引起整个系统的不稳定，如果不加改进地引入到智能家电控制领域，将存在一定的安全隐患。

为此，笔者在文章引入了一种基于事件的控制算法。该方法可避开随机时延问题，其理论的基本点在于引入一个不同于时间的新参变量，该参变量以传感器信息为依据，随控制过程的进行而更新，系统的理想输出控制命令是该参量的函数。在系统控制过程中，根据输入的参变量修正系统的目标输出值，使得系统操作过程成为动态过程。在本系统中通过通用 IO 口获取传感器的信息。

1.3 IP 电话

所谓 IP 电话，就是在 Internet 网上通过 TCP / IP 协议或 UDP 协议传送语音信息的电话系统。最初的 IP 电话技术，只是计算机对计算机的语音传输技术。双方用户都通过 Internet 联网，这种方法同时要具备 IP 电话软件、音频卡、麦克

风和扬声器等设备。比如我们常用的有：微软公司的 MSN、腾讯公司的 QQ 等。虽然能通话，但 PC 机体积大、价格高，因此使用范围很有单一，还算不上是真正的 IP 电话。近年来，美国一些大公司推出了用因特网传送国际长途电话的业务，实现了从普通电话机到普通电话机的 IP 电话。IP 电话已经通过网关把因特网与传统电话网联系起来，用户可以和普通电话用户一样，只要有电话机就能打 IP 电话，而通话费用远比普通电话的低[2]。但传统的 IP 电话利用率低，且功能有限，例如：只有语音功能，没有收发短信和视频功能；安全性不好，很容易被盗听；没有拒接电话功能；更改家庭住址后可能要变更电话号码等缺点。

1.4 IP 电话与智能家电通信技术

近年来，以 ARM 为代表的高性能专用微处理器的出现，以及 Linux、Windows CE 等操作系统的完善，使嵌入式技术迅速发展，这为智能 IP 电话和智能家电的研发提供了软硬件基础。目前，国外一些大电器公司，如西门子、飞利浦等都相继推出了一系列智能 IP 电话和智能家电，它们占据了大量的市场份额，而国内在这方面的研究相对落后。

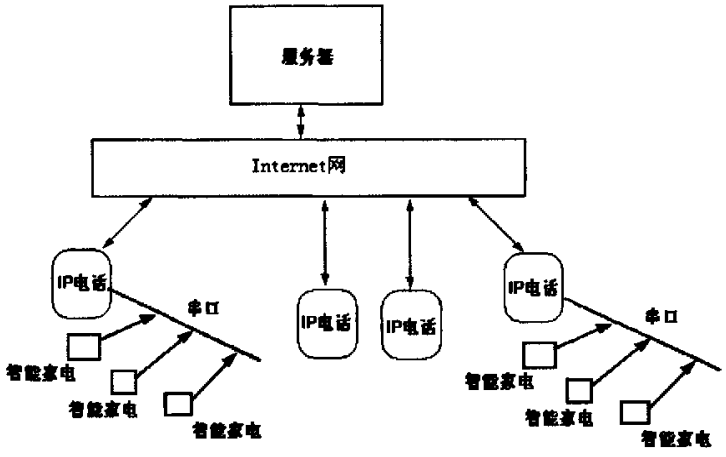

图 1-1 网络连接原理

针对我国的现状，依据家庭网络通讯具有传数据量较少的特点，采用串口实现智能化家庭控制网络，并将电话通信系统与家电遥操作控制系统集成于一体，可大量降低成本。本研究采用 ARM 芯片，基于 Windows CE 4.2(以下简称 WinCE)操作系统，构建客户端 IP 电话系统，该 IP 电话系统能实现语音、短信和遥操作智能家电等功能。采用 VC 编写服务器程序， 服务器负责各客户端 IP 电话的调度、计费等。为每一个 IP 电话分配一个固定电话号，当 IP 电话请求联结时，它首先向服务器发送 IP 地址与电话号捆绑数据包，服务建立 IP 电话与电话号码的对应关系表。打电话时，利用对方的电话号码在服务器上获得相应的 IP，这样

就可实现两个 IP 电话的通信。一方 IP 电话发送命令，经 Internet 网传输到对方后，通过串口与智能家电通信，控制智能家电，从而实现遥操作。原理如图 1-1。

由于各家电生产厂商都有不同的控制接口，这在一定程度上给控制网络带来不便。为此笔者提出了一个统一的接口，并定义了各种不同家电的接口协议，以便实现各家电与 IP 电话互联。智能家电采用 ARM9 芯片，Linux 操作系统。智能家电与 IP 电话采用串口通信，智能家电中的 ARM9 芯片采用 IO 口控制家电设备。

1.5 研究的意义和目的

尽管智能家居技术有了相当大的发展，但在智能家电网络化方面，还存在诸多的不足。本文提出了一种以嵌入式 IP 电话为控制中心的智能家电网络接入因特网的方案，采用串口对各智能家电进行网络互联，这样能通过 IP 电话对控制节点进行远程控制。同时各智能家电能通过 IP 电话这个控制中实现协同工作。对智能家居来说，其意义在于采用电话就可以随时随地监控家电的运行情况。虽然家庭子网串口通信的方式增加了少量布线，但增强了其隐定性，避免了大量无线设备的使用，从而降低了成本，而且也便于智能家电使用统一的硬件接口方案，对各种不同的家电只要修改相应的控制软件就可以满足实际需求。尽管我国智能家居在有关核心技术、独立产品等多方面落后于发达国家，但由于国内外都处于探索阶段，尚未形成垄断，同时国内又有潜在的庞大的消费群体。因此，通过计算机网络技术和嵌入式技术把 IP 电话与智能家电有机结合，在后 PC 时代 IT 产业发展中有广阔的发展前景。

1.6 研究要点

本文以研究利用 IP 电话控制智能家电为切入点，通过对该项领域的探索、研究，提出了采用 IP 电话作为控制中心的组网方案，设计了一套统一的智能家电接口，并且把一些新的控制算法引到该领域，以解决远程控制的时延问题。通过该系统，可通过远程 IP 电话及时了解家电的各种运行状态，实现智能家电的远程控制，各智能家电将收集到的各种信息送到 IP 电话，IP 电话根据数据的特点让各智能家电协同工作，使其能根据现场情况做出相应的调整和控制。为系统地说明问题，论文将研究要点在以下各章节分别详细论述。

第 2 章 基于 ARM2410 的 IP 电话硬件设计

2.1 ARM 系列芯片及选型方法

本系统采用的是 ARM2410 处理器芯片，下面对相关知识进行一个简单的介绍。ARM 公司是一家专门从事基于 RISC 技术芯片设计开发的公司，它不直接从事芯片生产，主要向其它半导体公司出售 IP 核知识产权。半导体生产商从 ARM 公司购买其设计的 ARM 微处理器核，根据各自不同的应用领域，加入适当的外围电路，从而形成自己的 ARM 微处理器芯片。这样 ARM 技术可获得更多的第三方软硬件支持，从而使整个系统成本降低，更具有竞争力。目前 ARM 芯片在工业控制、消费类电子产品、通信系统、网络系统、无线系统等领域占有广泛的市场份额。

ARM 公司开发了很多系列的 ARM 核，分别是 ARM7 系列、ARM9 系列、ARM10 系列和 StrongARM 等系列。各厂商基于 ARM 体系结构的处理器，除了具有 ARM 体系结构的共同特点以外，其中每一系列又根据其特点和应用领域不同包含的功能模块也不同，又形成不同的系列。目前生产 ARM 芯片的著名公司主要有 Intel 和三星等厂商。

在众多的 ARM 芯片家族中，为了降低产品成本，方便产品开发，不同的应用的选型是不同的。本系统中 IP 电话构建所使用的是 WinCE 操作系统，该操作系统需要使用虚拟内存，ARM9 芯片中提供的 MMU 是内存管理单元，可以满足这一要求。MMU 用来管理内存系统，它可以“动态地”重新定位内存空间，并对其进行有效的管理。它控制内存的访问权限和将虚拟地址转换为物理地址。利用 MMU 操作系统可以把逻辑地址空间跟实际的物理存储器屏蔽开来，从而使程序员只需关心操作系统给出的编程接口而无需关心底层物理存储器的调度。当然如果使用 uCLinux 则可以不需要 MMU 的支持，就可采用 ARM7TDMI 芯片，但这会增加高端多媒体软件开发难度。系统的工作频率在很大程度上决定了 ARM 芯片的处理能力。本系统开发中占用资源较多的是语音传输，ARM9 芯片的工作频率为 200MHz，可以满足这一要求。串口控制方面所占用的资源相对较少。ARM9 芯片有足够的通用 IO 接口，可以满足系统开发的要求。考虑到 IP 电话装的 WinCE 操作系统大约占 30M 的存贮空间，所以外部扩展 FLASH 存储器。本系统要用到的接口主要有 HS 音频接口、LCD 控制器、DMA 控制器、网络接口、串行口和通用 IO 口等[3]。

综合以上各因素并结合实际需要，本系统选用三星公司的 S3C2410A 微处理器，这样具有较高的性格比。下面对该芯片相关内容进行一个简单的介绍。

2.2 ARMS3C2410 芯片的特点

传统的 CISC 复杂指令集计算机结构处理器设计复杂，指令集使用率不高。而 RISC 精简指令集计算机优先选取使用频率最高的简单指令，避免复杂指令，将指令长度固定，指令格式和寻地方式种类减少。ARM 处理器是基于 RISC 技术的，它共有 37 个 32 位寄存器，被分为若干个组，这些寄存器包括：31 个通用寄存器，包括程序计数器。6 个 32 位状态寄存器，用以标识 CPU 的工作状态及程序的运行状态。同时，ARM 处理器又有 7 种不同的处理器模式，在每一种处理器模式下均有一组相应的寄存器与之对应。在所有的寄存器中，有些是在 7 种处理器模式下共用同一个物理寄存器，而有些寄存器则是在不同的处理器模式下有不同的物理寄存器。在任意一种处理器模式下，可访问的寄存器包括 15 个通用寄存器（R0～R14）、一至两个状态寄存器和程序计数器[3]。

S3C2410 芯片是三星公司推出的 32 位 RISC 处理器，适用于手持设备，数字多媒体播放设备等具有低价、低功耗和高性能等特点。采用 ARM920T 内核，具有以下特点：5 级整数流水线，提供 1.1MIPS/MHz 的哈佛结构，支持 32 位 ARM 指令集和 16 位 Thumb 指令集，支持 32 位的高速 AMBA 总线接口，全性能的 MMU，支持 Windows CE、Linux 等多种主流嵌入式操作系统，支持数据 Cache 和指令 Cache[3]。

该芯片提供了以下丰富的内部设备：MMU 虚拟存储管理，LCD 控制器，I/O 端口，RTC，触摸屏接口，内部 PLL 时钟倍频器等，消除了为系统配置额外器件的需要，大大减少了整个系统的成本。它支持 NAND FLASH 系统引导。具有更高的指令和数据处理能力。

ARM 芯片具有 RISC 体系的一般特点：它具有大量的寄存器，绝大多数操作都在寄存器中进行；通过 Load/Store 的体系结构在内存和寄存器之间传递数据；寻址方式简单，采用固定长度的指令格式。除此之外，ARM 体系结构采用了一些特别的技术，在保证高性能的同时尽量减少芯片面积和功耗。这些技术包括：在同一条数据处理指令中包含算术逻辑处理单元处理和位移处理；使用地址自动增加（减少）来优化程序中循环处理；Load/Store 指令可以批量传输数据，从而提高数据传输的效率；所有指令都可以根据前面指令执行结果，决定是否执行，以提高指令执行的效率。除了以上特点之外，ARM 体系结构还增加了一些特殊功能扩展，包括以下几个部分：Thumb 指令集，Thumb 指令集是 ARM 的一个子指令集。ARM 指令长度为 32 位，Thumb 指令长度为 16 位。使用 Thumb 指令集可以得到密度更高的代码，可以进一步减少可执行文件的大小，从而减少存贮器的大小，降低成本。长乘法指令，增加了两条长乘法的 ARM 指令。增强型 DSP 指令，包含了一些附加的指令，这些指令用于增强处理器对一些典型的

DSP 算法的处理性能。ARM 媒体功能的扩展，为嵌入式应用系统提供了高性能的音频、视频处理技术。结构如图 2-1[3]。

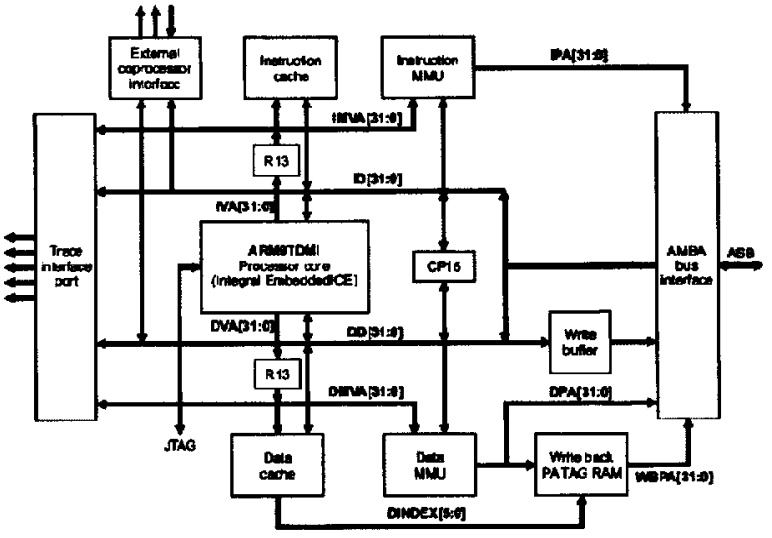

图 2-1 ARM 芯片结构特点

2.3 IP 电话硬件设计

本系统 IP 电话采用 ARM9 芯片，运行 WinCE 操作系统。IP 电话硬件组成框图如图 2-2，包括系统电路和外围接口电路。

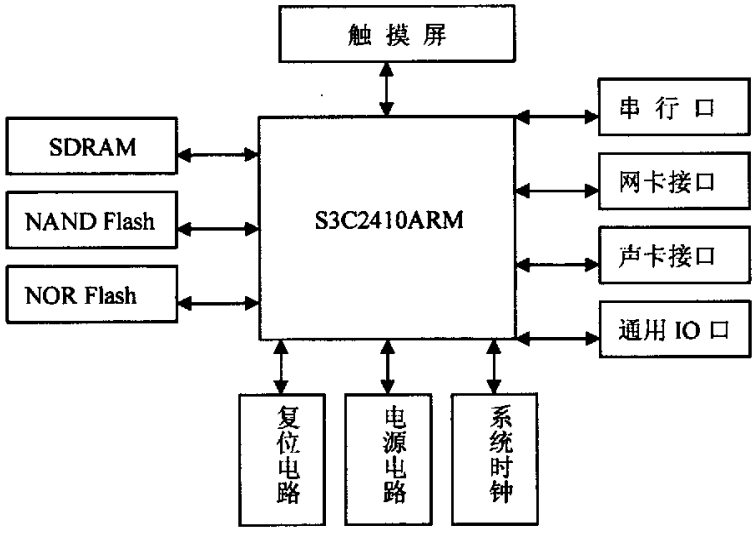

图 2-2 IP 电话硬件组成框图

2.3.1 系统板电路

在运行操作系统之前，需要对系统的软硬件资源进行合理的配置与管理。这个功能是由 BootLoader 来完成的。在嵌入式系统中，BootLoader 的作用与 PC 机上的 BIOS 类似，通过 BootlLoader 可以完成对系统板上的主要部件如 CPU、FLASH、 SDRAM、串行口等进行初始化，从而为操作系统的运行创造良好的环境。同时也可以下载文件到系统板、对 FLASH 进行擦除与编程。

BootLoader 作为系统复位或上电后首先运行的代码，存放在起始物理地址为 0x0 开始的 FLASH 存储器中。本系统采用 SST 公司的 SST39VF1601 型线性 NOR FLASH 存储 BootLoader，由并 nGCS0 片选信号用于启动引导。这样当系统启动时 0x0 地址程序开始运行。由于 IP 电话中运行的操作系统为 WinCE，整个系统镜像文件大约 30M，因此采用三星公司 K9F1208 非线性 NAND FLASH 芯片存储，由 ARM 的 NAND FLASH 控制器对它进行读写操作。系统时钟采用外部 12MHz 晶振，RTC 采用 32768Hz 晶振。

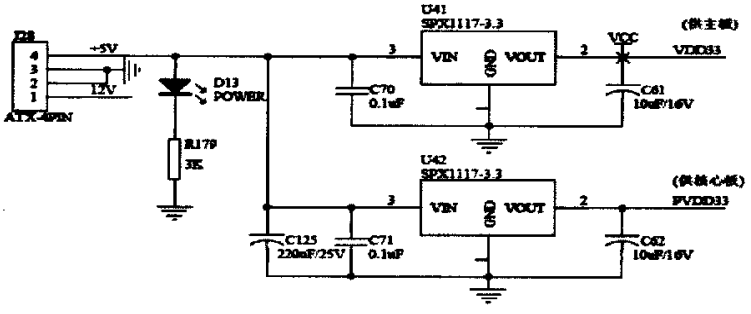

图 2-3 电源电路

由于 ARM2410 需要 1.8V 和 3.3V 两组电源，所以采用 SPX1117M3-1.8 型 LDO 芯片产生稳压电源。核心板需要扩展板提供一组 3.3V 电源，1.8V 内核电源则由核心板上的 LDO 芯片产生，电路如图 2-3[4]。

在核心板的复位电路中使用了 MAX809TD 作为电源监控复位芯片，以提高系统的可靠性。电路原理如图 2-4[4]。

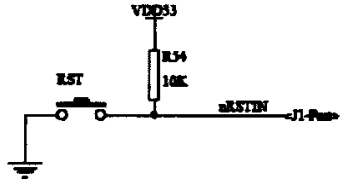

图 2-4 复位电路

S3C2410A 可以使用外部晶振或外部时钟输入作为系统时钟。外部晶振频率范围是 10MHz~20MHz。核心板采用了 12MHz 外部晶振，所以将芯片的 OM2、OM3 引脚接低电平，并将 EXTCLK 引脚接高电平。通过 S3C2410A 内部的锁相

环，可以将时钟倍频至 203MHz 作为处理器主时钟。独立时钟源 RTC 由 XTOrtc 和 XTIrtc 引脚上接的 32768Hz 石英晶振提供。电路如图 2-5[4]。

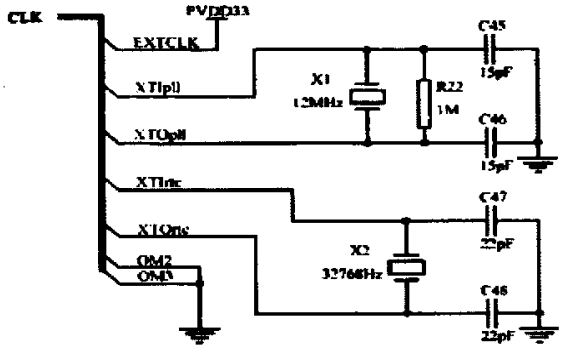

图 2-5 系统时钟

本系统扩展了一片 NOR FLASH 芯片，使用 nGCS0 片选信号，这主要是为了使用其来启动引导系统，它被分配到 Bank0 存储空间，使用 nGCS0 片选信号。

核心板上使用了 2 片 HY57V651620 芯片作为 SDRAM 存储器，并使用了 nGCS6 片选信号。通过将两芯片的数据线分别与数据总线的高 16 位和低 16 位相连，实现 32 位的数据总线宽度。由于读写操作以 4 字节为单位，并要求地址对齐，所以地址最小变化为 0x4。因此要忽略 S3C2410A 的 ADDR0 和 ADDR1 引脚。所以将 S3C2410A 的 ADDR2 引脚与 RAM 芯片的 A0 引脚连接，其它地址线依次连接。为了能够正确访问 HY57V65-620 的高/低位字节数据，将 nwBEX 信号与 RAM 芯片的 UDQM/LDQM 相连。HY57V651620 的 BA0、BA1 引脚是 SDRAM 内部 Bank 地址选择线，代表了 SDRAM 内存地址的最高位，64MB 的 SDRAM 需要 26 根地址线寻址，因此需将 BA0、BA1 连接到 ADDR24、ADDR25 引脚。

NAND Flash 存储器采用 K9F1208U0B 芯片，把它与 S3C2410A 内部已经集成的 NAND FLASH 控制器相应的引脚相连即可。

2.3.2 外围接口电路

本系统开发中用到的接口包括串口、以太网口、音频接口、LCD 及触摸屏接口等，下面对这几个方面的接口电路做简单的介绍。

RS-485 串口采用差分信号负逻辑，+2V~+6V 表示“0”，-6V~-2V 表示“1”。RS-485 有两线制和四线制两种接线，四线制只能实现点对点的通信方式，两线制接线方式为总线式拓朴结构。在 RS-485 通信网络中一般采用的是主从通信方式，即一个主机带多个从机。由于系统是 3.3V，所以使用 SP3485 电平转换。SP3485 是 3.3V 工作电源的半双工 RS-485 收发器。如图 2-6[4]。

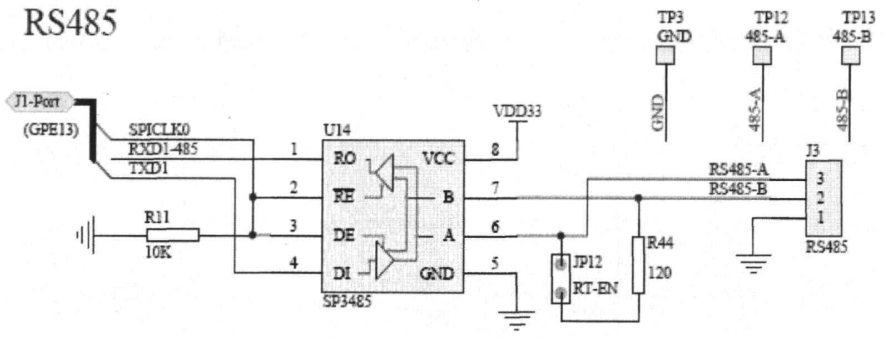

图 2-6 RS485 电源电路

本系统使用 DM9000E 芯片作为 10/100M 以太网控制器。电路使用 16 位总线方式进行控制，数据总线 DATA0-DATA15 与芯片的 SD0-SD15 相连，地址线也同样进行相对应的连接，而片选线 nGCS3 与芯片的 AEN 相连。系统中使用 S3C2410A 的地址线 ADDR2 连接到芯片的命令/数据使能端 CMD。由于 DM9000E 的工作基地址是 0x300，所以对其进行操作的地址端口是 0x300，数据端口是 0x304。因而结合 S3C2410A 的片选线得到的 32 位地址端口为 0x18000300，数据端口为 0x18000304。接口电路的连接如图 2-7[4]。

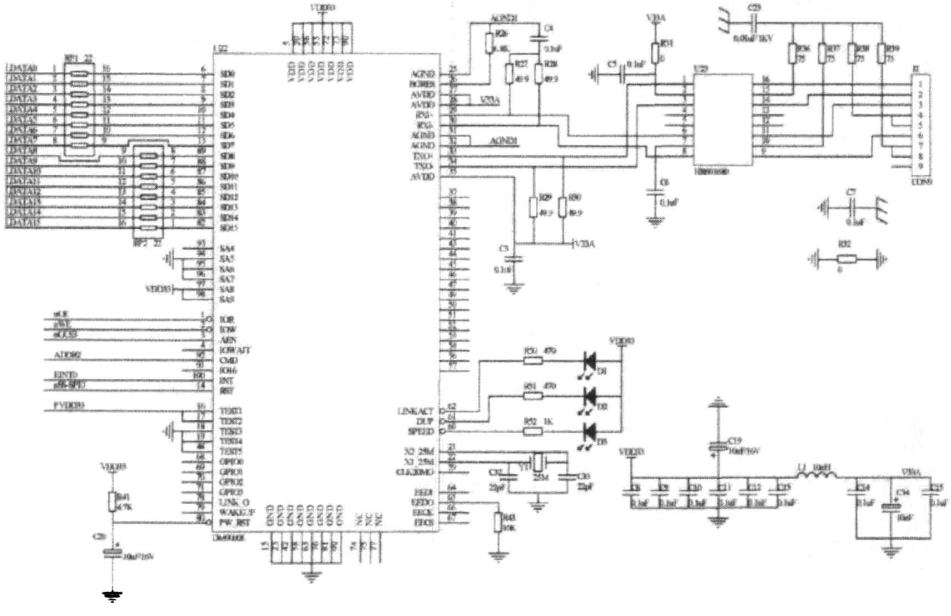

图 2-7 网卡接口电路

S3C2410A 内置有 LCD 控制器，可以支持最大 256K 色 TFT 彩色液晶显示屏和最大 4K 色的 STN 彩色液晶显示屏。本系统 IP 电话端省去了键盘，采用触模屏实现人机交互。为了能清晰的显示，选用 Sharp 公司的 LQ080V3DG01 型 8 英寸、分辨率为 640x480 的 TFT 液晶屏。此液晶屏采用 5V 电源供电，最小值为 2.3V，可直接与 S3C2410A 的控制口线相连。LQ080V3DG01 液晶屏有 18 根数据线(R、G、B 各 6 根)。S3C2410A 的控制器应选用 16BPP 模式，将 S3C2410A 的 VD2~VD7 与液晶屏的 B0~B5 相连，VD10~VD15 与 G0~G5 相连，VD18~VD23

与 R0~R5 相连。电路连接原理如图 2-8。LQ080V3DG01 液晶屏使用冷阴极背光灯管，需要高压交流电源供电。本系统采用 CXA_L10A 驱动模块作为驱动电路，如图 2-9[4]。

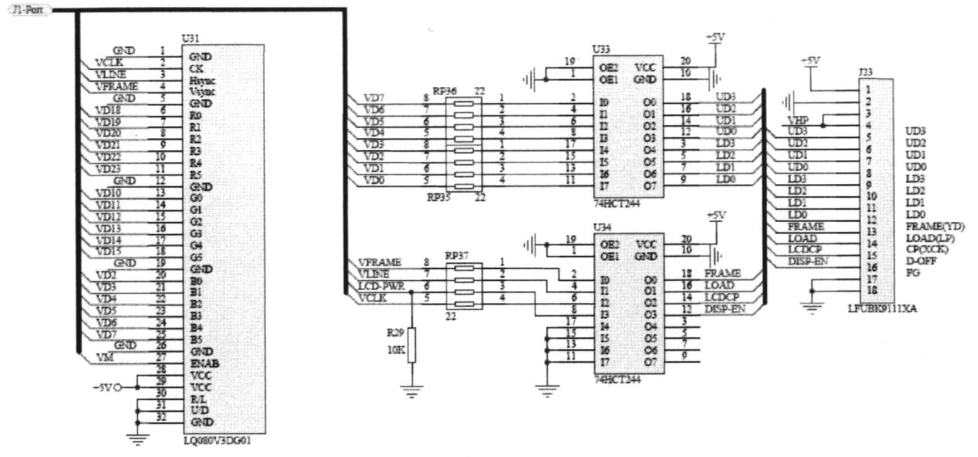

图 2-8 显卡接口电路

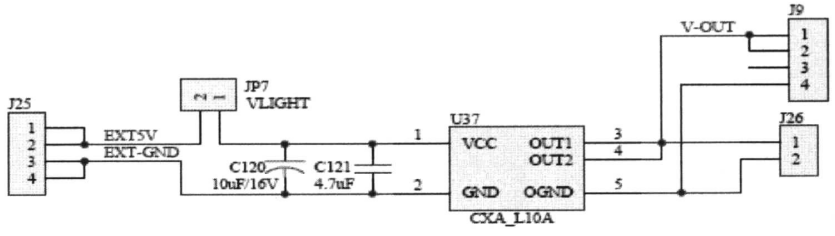

图 2-9 背光灯管驱动电路

S3C2410A 内置有 IIS 音频接口，只需要外接一个数字音频编解码器即可实现音频的输入输出。本系统中采用的数字音频编解码器为 UDA1341TS，电路如图 2-10[4]。为了扩大音量，本系统中加入了双声道功率放大电路，如图 2-11[4]。

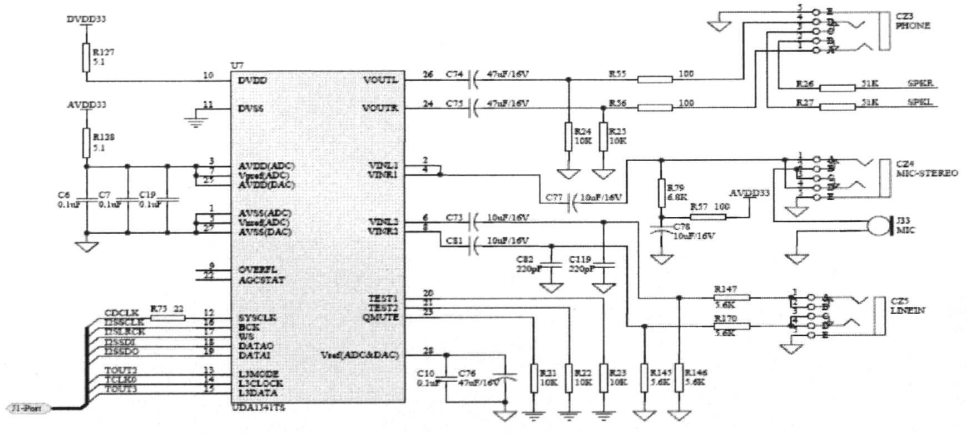

图 2-10 声卡接口电路

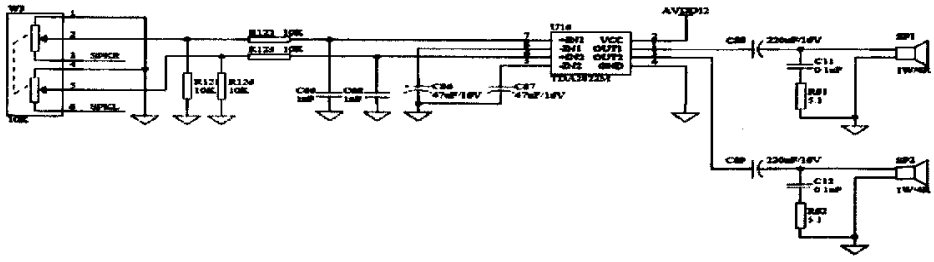

图 2-11 双声道功率放大电路

在智能家电端，控制智能家电采用 ARM 芯片的通用 IO 口，共使用 8 个 IO 口，各口的功能定义如下：IO 口 1、2 用于控制家电的电源开关，IO 口 3 用于控制家电的正逆运行模式，IO 口 4、5、6 的 8 种组合根据不同的家电实现不同的控制，IO 口 7、8 用于获取家电的运行状态，这在应用举例中将做介绍。各用于输出的接口使用 PNP 三极管作驱动，电路如图 2-12。

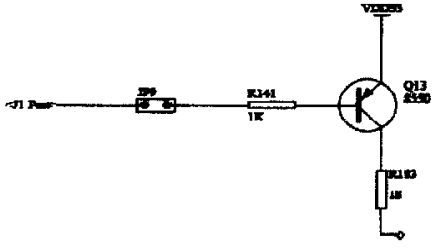

图 2-12 通用 IO 口电路

第 3 章　IP 电话端软件系统设计

3.1 嵌入式操作系统概述

嵌入式系统是以应用为中心，以计算机技术为基础，软件硬件可裁剪，对系统功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。它的目的性或针对性很强，具有软件代码小、高度自动化、响应速度快等特点。

嵌入式系统的操作系统和功能软件集成于计算机硬件系统之中，也就是软件与硬件的一体化。对上层应用开发者而言，嵌入式操作系统要求高度简练、界面友善、质量可靠、应用广泛、易开发、多任务，并且价格低廉。操作系统负责连接底层硬件和上层的应用程序，为用户提供 API 函数。这样用户无需关心底层的硬件以及驱动程序，只需利用提供的 API 函数进行开发，大量减轻了应用程序的开发难度和工作量。同时用户在微控制器上进行 TCP/IP 或者 UDP/IP 协议栈的开发难度比较大，这样可以充分利用操作系统上已有的网络协议栈，使用协议栈提供的接口函数进行应用开发[5]。

目前嵌入式操作系统，有 Vxwork、WinCE 和 Linux 等。Linux 是源代码开放的操作系统，支持不同的配置，能方便的移植到多种体系结构的嵌入式处理器中，而且成本低。WinCE 操作系统是商业化产品，价格高，源代码的封闭性大量限制了开发者使用。下面对其特点进行简要介绍。

3.2 WinCE 操作系统的特点

WinCE 是一模块化、结构化的操作系统，可以被剪裁、配置。它的内核很小，基本只实现调度、内存管理和核心 API，其它功能都在单独的模块中实现。它具有多线程、多任务、完全抢占式和出色的图形用户界面等特点，支持内存管理、文件操作与网络功能[7]。

WinCE 是面向硬件资源有限的模块化实时嵌入式操作系统。它的模块化设计使嵌入式系统和应用开发者能够方便地定制以适应一系列不同的产品，是专门为各种资源有限的硬件系统设计的。可为消费类电子设备、Web 终端、Internet 访问应用设备、专用工业控制器、移动数据获取手持设备及嵌入式通信设备提供软件系统驱动。具有以下主要特点[7]。

3.2.1 良好的软件兼容性

WinCE 继承了传统的 Windows 图形界面，使用与 Windows 平台同样的应用程序接口函数，具有 Win32 API 子集，使用同样的界面风格。因此，绝大多数的应用软件只需简单的修改和移植就可以在该平台上继续使用。在 WinCE 中，除了一些基本的 Windows 通用控件以外，还有一些专门设计的控件。

3.2.2 硬件支持能力强

WinCE 并非是专为单一装置设计的，它具有高度的灵活性和可定制性，广泛应用于消费电子产品、工业自动控制和医疗及科研设备等。支持多类硬件外围设备，如键盘、鼠标设备、触摸面板、串行口、以太网卡、调制解调器、USB 设备等。它与处理器无关，能在多种不同框架的嵌入式处理器上运行。

3.2.3 强大的网络通信功能

WinCE 提供通过各种传输介质进行网络连接和通信，用来通信的硬件有串口、网络接口、红外接口等。串口通信采用的是异步收发器 UART，它们首先把并行的数据按位串行，然后再通过数据线逐位收发，到达目标机后再把数据据并行化。WinCE 提供了 SLIP/PPP 协议，通过些协议可利用串口建立 TCP/IP 网络连接。它内置了 TCP/IP 协议栈，提供了一套开放的、支持多种协议的 WINDOWS SOCKET 网络编程接口规范。

3.2.4 稳健的实时性支持

实时性是指能够在限定时间内执行完规定的任务，并对外部的异步事件做出反应的能力。WinCE 它支持嵌套中断，允许更高优先级的中断首先得到响应，而不是等待低级别的中断服务线程完成。

3.2.5 丰富的多媒体支持

WinCE 提供丰富的多媒体技术，在语音方面，它支持波形音频，通过波形音频 API，应用程序可支持波形 I/O，可以对音频 I/O 设备进行控制。基于 DirectX API 和 Windows Media 的技术可以提供高性能的视频、音频和流式多媒体服务。因为在 IP 电话端有较的多媒体编程，所以采用 WinCE 操作系统可以减少程序设计的复杂性。

3.2.6 强大的开发工具

与其它嵌入式系统相比，WinCE 为开发人员提供了友好的开发工具，这些开发工具可帮助开发人员简化开发流程并提高开发效率。由于 WinCE 是模块化的，可以针对不同的应用平台进行定制。使用满足平台系统需求的最小软件模块和组件集合来设计嵌入式系统平台，可节省内存，并提高操作系统的性能。

系统定制采用 Platform Builder 集成开发环境。它包括适用于所有 WinCE 支持的处理器的交叉编译器，提供了所有进行设计、创建、编译、测试和调试 WinCE 操作系统平台的工具。它运行在桌面 Windows 下，开发人员可以通过交互式环境来设计和定制内核、选择系统特性，然后进行编译和调试。同时，开发人员还可以利用 Platform Builder 来进行驱动程序开发和应用程序项目的开发等。

Embedded Visual C++是 WinCE 程序主流可视化开发工具。EVC 和 VC 在界面、语法和开发流程上基本上是一样的，熟悉 VC 的程序员很快就会使用 EVC 进行开发。EVC 还提供了模拟器来模仿目标硬件进行开发和调试，方便了编程人员，加快了开发进度。

3.2.7 强大的内存管理、进程调度和中断响应能力

WinCE 有很好的内存管理功能，其中物理页面管理负责跟踪物理内存的使用情况，为换页管理抓取有效的物理页面以及释放不使用的物理页面等；虚存管理主要负责系统的地址映射以及换页管理，系统中 32 位的虚拟地址提供 4G 的虚拟空间；堆管理主要管理动态内存释放、回收等，以支持程序的动态数据结构。

WinCE 有很好的进程调度功能，是抢占式的多任务实时操作系统，它允许多达 32 个进程同时运行。进程数目的限制可以有效缓解内存系统的压力，系统作地址映射的时候，只需要映射所限的个数。线程是进程的一个实体，是 CPU 调度和分配的基本单位。在 WinCE 下，一个进程包括一个或多个线程，且提供多级别的调度能力，线程的调度采用优先级基于时间片轮转方法。每个线程在同样长度的时间片内运行，该时间片是可调的。可运行的线程处于可运行的队列中，每个优先级都对应一个队列，最多有 256 个。一个线程的时间片运行完后，系统调度策略把它安排到相应优先级队列的末尾，然后再让优先级最高的队列的第一个线程运行，这就保证了同一优先级的线程获得平等的运行权。引入线程之后，系统开销减少，两个线程之间的切换所花费的时间少，而且由于同一个进程内的线程共享进程所拥有的资源，之间的通信不需要额外的机制，简化了通信方式，加快了通信速度。

WinCE 有很好的中断响应能力，各种外设通过硬件中断和 WinCE 的核心通信。系统发现中断以后，处理的过程分为两个部分：核心态中断例程 ISR 和用户态中断例程 IST。ISR 一般要求短小精悍，效率要求严格，通常只响应设备并返

回一个中断标识给操作系统。WinCE 支持静态 ISR 和可安装 ISR。静态 ISR 只能静态编译进核心，运行时不能改变，与 IST 通信是单向的，由 ISR 到 IST。静态 ISR 支持嵌套中断，并且使用核心堆栈。可安装 ISR 则由内核管理程序从 DLL 中动态加载，和 IST 通信是双向的。多个 ISR 可以与同一个中断请求相关联，系统按照加载驱动的顺序依次调度。在可安装 ISR 中，共享内存的使用比较灵活。用户线程 IST 处理中断请求，核心接到 ISR 传给自己的中断标识之后发出一个中断事件，激活一个在该事件等待队列上的 IST，调度器就会调度这个线程工作，处理中断事务。

3.3 WinCE 操作系统的定制

WinCE 操作系统由于其模块化设计，所以可以剪裁、配置和定制。各种嵌入式硬件设备功能各异，可根据应用的需要，选择需要的操作系统功能进行不同的定制，从而生成满足开发要求的操作系统运行映像。WinCE 操作系统定制是通过 Platform Builder 工具来完成的。该工具能够根据用户的需求，选择构建具有不同内核功能的 WinCE 系统。同时，它也是一个集成的编译环境，可以为所有 WinCE 支持的 CPU 目标代码编译 C/C++程序。一旦成功地编译了一个 WinCE 系统，就会得到一个名为 nk.bin 的映像文件。将该文件下载到目标平台即可运行。

在定制操作系统之前，先对几个 WinCE 定制中用到的概念做个简要的介绍。

1、BSP 板级支持包

它是介于主板硬件和操作系统中驱动层程序之间的一层，一般认为它属于操作系统一部分，主要是实现对操作系统的支持，为上层的驱动程序提供访问硬件设备寄存器的函数包，使之能够更好的运行于硬件主板。BSP 是相对于操作系统而言的，不同的操作系统对应于不同定义形式的 BSP。它与目标板是一一对应的，BSP 中包括某一块目标板上所有外设的驱动程序。在本系统开发中采用的是三星公司的 ARM2410，因此采用的板级支持包为 SMDK2410[8]。

2、WinCE 模拟仿真器

在嵌入式系统开发过程中，并不是先做好了主板硬件后再做软件，而是采用主板硬件与软件并行开发的方式，因为这样可以提高开发效率，缩短产品进入市场的时间。在硬件还没有开发完成时，要对运行在其上的软件进行调试，采用的方法就是在 PC 机上用软件虚拟出一个相应平台的硬件环境。它具有完整硬件系统功能，运行在一个完全隔离环境中，构成一个抽象的完整计算机系统，和实际的计算机一样，具有一个指令集并使用不同的存储区域。它负责执行指令，管理数据、内存和寄存器。

这台虚拟的机器在任何平台上都提供给编译程序一个的共同的接口。编译程序只需要面向虚拟机，生成虚拟机能够理解的代码，然后由解释器来将虚拟机代码转换为特定系统的机器码执行。虽然虚拟机软件它只是运行在物理计算机上的一个应用程序，但是对于在虚拟机中运行的应用程序而言，它就像是在真正的计算机中进行工作。

在虚拟机中进行软件评测时，可能会使系统崩溃。但是，崩溃的只是虚拟机上的操作系统，而不是物理计算机上的操作系统，并且，使用虚拟机的恢复功能，可以马上恢复虚拟机到安装软件之前的状态。使用这种方法可以方便实现软件的模拟，在 Platform Builder 中有自带的模拟器，在系统定制过程中为了确保其正确性，都先在模拟器中进行过仿真，然后再编译下载到硬件平台上进行验证。

3、导出 SDK

在定制了自己的平台后，因为这种平台是为特定功能而设计的，其编程所用到软件的相关文档、范例和工具的集合也不相同。因此我们要为我们的定制平台导出特定的 SDK 软件开发工具包以供开发应用程序所使用。SDK 包含程序库、头文件、示例程序源代码和库函数使用文档，同时还包括编程指导和 API 函数以及设备驱动工具包。

3.4 WinCE 操作系统的定制流程

系统的定制是在 Platform Builder 中完成的，可增删功能模块，使之符合实际系统需求。平台的定制过程如下：

首先，选择操作系统的基本配置，并且为特定的平台选择相应的微处理器和平台支持包 BSP，它是由主板厂家提供的组件。其次，制定平台，在此阶段可开发设备驱动，适当地裁剪、添加组件和修改某些配置文件。然后，封装所需要的各功能模块，编译生成 OS 镜像文件，并用模拟仿真器进行软件测试。接着，把镜像文件下载到目标设备，进行调试。如果不能满足要求，则要进行重新配置、封装、下载及调试，直到完成平台的创建。最后导出相应的 SDK 软件开发工具包，运行后加到 EVC 中，使得可以进行特定硬件平台上的应用程序开发。

因为本系统 ARM 芯片选用的是 S3C2410 芯片，所以开发时采用三星公司提供的 BSP。首先把三星公司提供的 SDMK 文件夹放入 Platform Builder 安装目录 C:\WINCE420\PLATFORM 中。由于本系统所用网卡是 DM9000，把 SDMK 文件夹中目录_for_Public\oak\drivers\NETCARDC 下的网卡的驱动文件 DM9000 夹放入 C:\WINCE420\PUBLIC\COMMON\OAK\DRIVERS\NETCARD 目录下，再把系统的配置文件放入 C:\WINCE420\PUBLIC\COMMON\CESYSGEN 目录中。把

Platform Builder 开发平台自带的 smdk2410 特性文件删除，加入三星公司提供的特性文件，如图 3-1[8]。

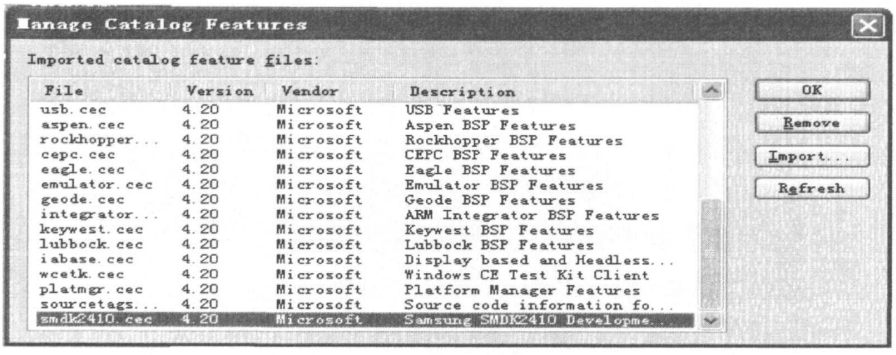

图 3-1 加入特性文件

针对特定的开发板的 BSP 和相关特性加入开发平台后，利用向导，生成针对 ARMV4 版本的镜像文件，方法如图 3-2。

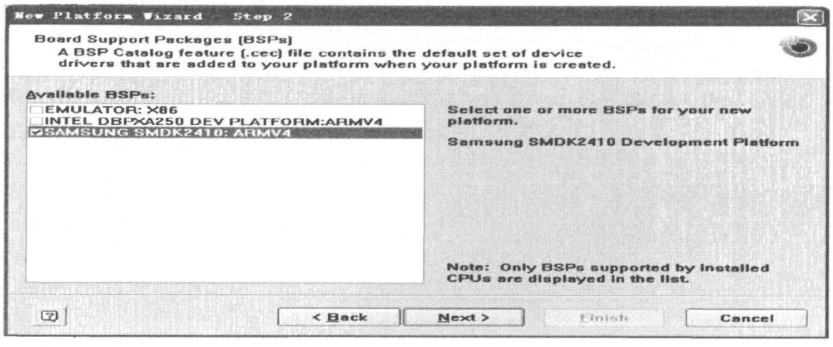

图 3-2 选择硬件平台

本系统主要实现 IP 电话功能，在定制时选择 IP 电话特性，如图 3-3。

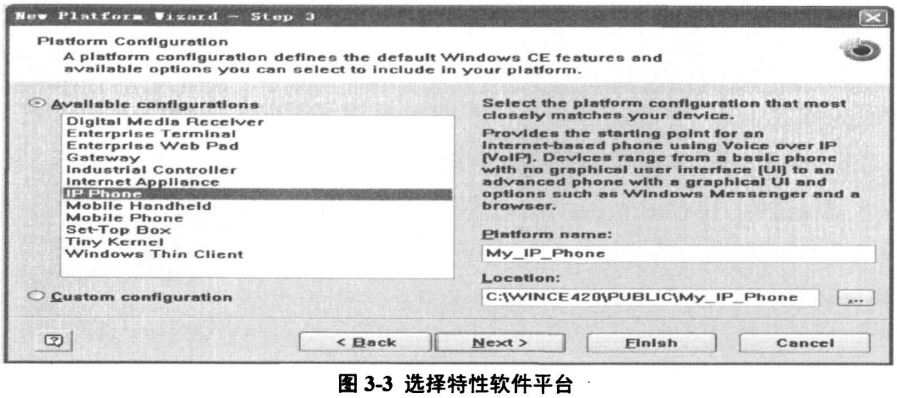

图 3-3 选择特性软件平台

加入相应特性后，开发平台如图 3-4，进行编译即可生成镜像文件。

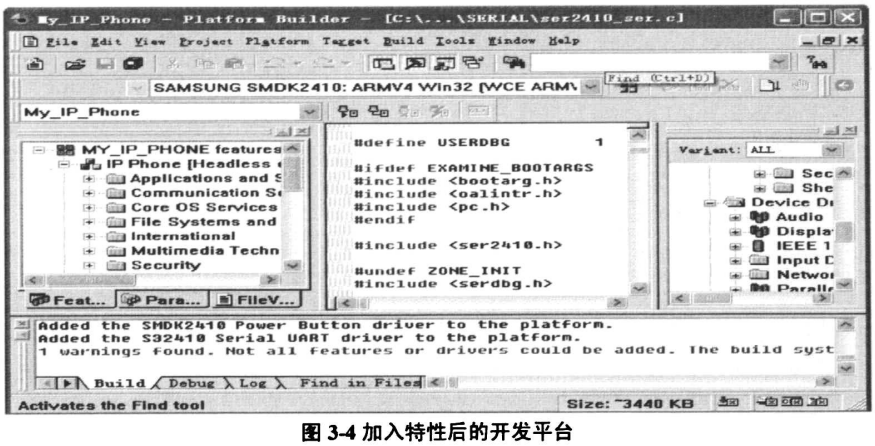

图 3-4 加入特性后的开发平台

至此，已经定制了 WinCE 操作系统的 IP 电话，要在该系统中开发应用程序，必须利用 Platform Builer 导出 SDK 安装包，以便在 ECV 开发环境安装该导出软件包。按照向导就可生成导出 SDK。方法如图 3-5 和 3-6。

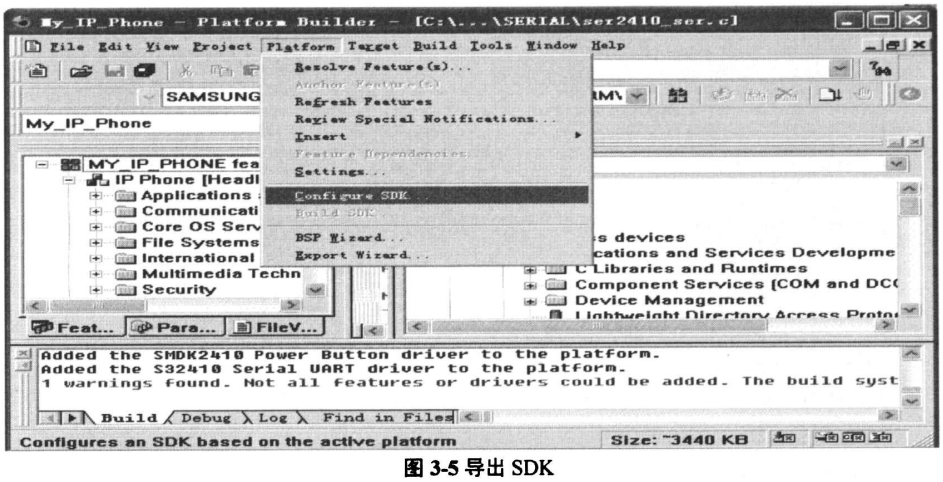

图 3-5 导出 SDK

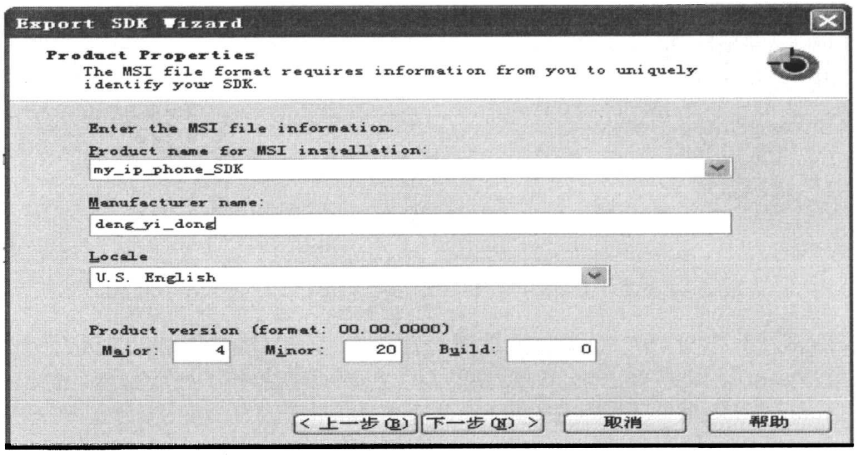

图 3-6 导出 SDK

即此，运行在 IP 电话端的 WinCE 可执行镜像文件已经生成，并且针对该平台的导出 SDK 也已经生成。

3.5 IP 电话应用程序开发

前面已经提到开发基于 WinCE 的应用软件的工具有多种，可以采用.NET 来开发，也可采用 EVC 来开发。由于本系统要求有较高的运行效率，故选择 EVC 作为开发工具，它生的代码比较优化，占用内存空间小，执行速度快。先安装 EVC 开发平台，然后再安装刚才导出的 SDK，因为导出 SDK 是针对特定硬件平台下 IP Phone 的软件开发包，安装了该软件包的 EVC 开发平台就可以用来开发目标平台的 IP 电话应用软件。

3.5.1 相关概念介绍

1、进程：是一个具有独立功能的程序关于某个数据集合的一次运行活动，可以申请和拥有系统资源，它为应用程序的运行实例，是应用程序的一次动态执行。进程由控制块、程序段、数据段三部分组成，一个进程可以包含若干线程。

2、线程：对于同一个程序，可以分成若干个独立的执行流，即线程。线程提供了多任务处理的能力。Windows 提供了两种线程，用户界面线程和工作者线程。用户界面线程通常是主线程，用来和用户交互。工作者线程通常用来处理后台运行的任务，如本程序中 IP 电话串口与智能家电通信的线程[11]。

为了向应用程序提供多线程功能，Win32 函数库中提供了操作多线程的函数，包括创建线程、终止线程、建互斥区等。

在应用程序的主线程或者其他活动线程中创建新线程的函数如下：

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  DWORD dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

如果创建成功则返回线程的句柄，否则返回 NULL。可以调用如下函数来设置线程的优先权：

```
BOOL SetThreadPriority(
  HANDLE hThread,
  int nPriority
);
```

当调用线程的函数返回后，线程自动终止。如果需要在线程的执行过程中终止则可调用函数:

```
VOID ExitThread(
```

```
  DWORD dwExitCode
);
```

如果在线程的外面终止线程，则可调用下面的函数:

```
BOOL TerminateThread(
  HANDLE hThread,
  DWORD dwExitCode
);
```

但该函数可能会引起系统不稳定，而且线程所占用的资源也不释放。

3、线程的同步

在多线程处理时，线程之间经常要同时访问一些共享资源。如果不加保护地访问这些资源，则会产生访问冲突，造成数据的不一致性。因此必须使线程同步，确保同一时间只有一个线程访问共享资源。Win32API 提供了多种同步控制对象来解决共享资源访问冲突。

Win32API 提供了一组能使线程阻塞其自身执行的等待函数。这些函数在其等待的同步对象产生了信号，或者超过规定的等待时间才会返回。在等待函数未返回时，线程处于等待状态。使用等待函数可以保证线程同步。常用的等待函数是:

```
DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD dwMilliseconds
);
```

而函数 WaitForMultipleObjects 可以用来同时监测多个同步对象，以协调多线程的执行，该函数的声明为:

```
DWORD WaitForMultipleObjects(
  DWORD nCount,
  CONST HANDLE *lpHandles,
  BOOL fWaitAll,
  DWORD dwMilliseconds
);
```

同步对象主要有互斥体对象、信号对象、事件对象和临界区。各种同步对象都有自己的特色，在本系统应用程序开发中用到的是互斥体对象。Mutex 互斥对象的状态在它不被任何线程拥有时才有信号，而当它被拥有时则无信号。Mutex 对象很适合用来协调多个线程对共享资源的互斥访问。互斥体对象在同一时刻只能被一个线程占用，当互斥体对象被一个线程占用时，若有另一线程想占用它，则必须等到前一线程释放后才能成功。

3.5.2 IP 电话原理概述

IP 电话端软件的人机交互采用触摸屏，不必使用键盘输入，从而可降低成本，方便用户。操作界面如图 3-7。

本软件实现 4 大功能，即语音通信、语音留言、短信以及远程智能家电控制。本系统的基本工作原理为：首先构建一个服务器，它拥有固定的公网 IP 地址。服务器中建有一个数据库，它包含 IP 电话的电话号码与其 IP 地址的映射关系。为每一个 IP 电话，分配一个不同的固定电话号码，该电话号码已经与 IP 电话的应用软件捆绑在一起，不同的电话烧写有不同电话号码的应用软件。这样做的一个好处是，当用户家庭地址异地搬迁时，在不同的地区，可以不用更换电话号码，只要使用同一台电话即可。

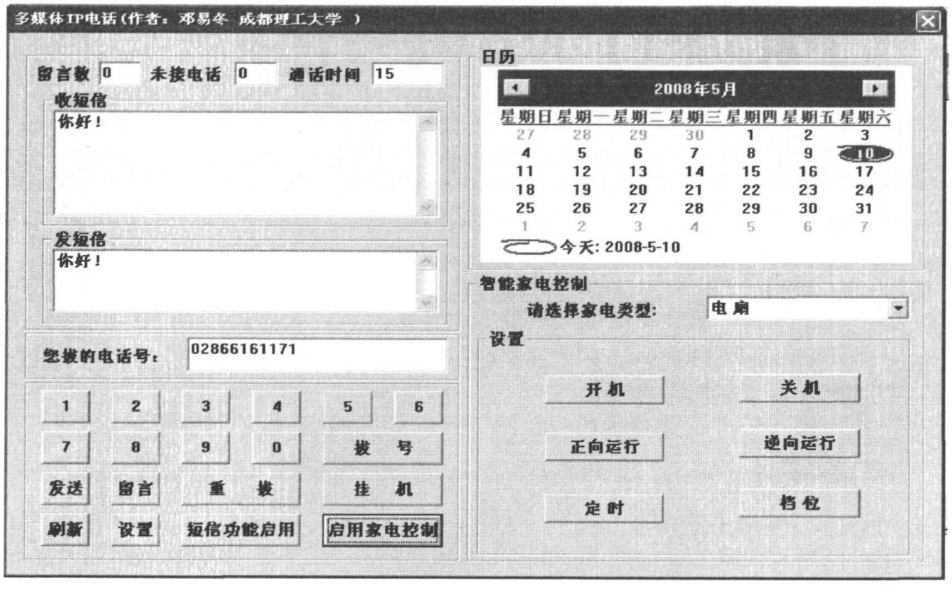

图 3-7 电话运行界面

在 IP 电话启动时，它首先向服务器发送数据，将其电话号码与其当前的 IP 地址一起发送给服务器，服务器将这个数据存入它的数据库中，这样就形成了一个 IP 电话号码与 IP 地址动态对应表。拨打电话时，它首先访问服务器，并向服务器发送 IP 地址查询请求，服务器根据请求方要求，在数据库中查找被叫方的 IP，如果能找到，则以数据包的形式发送给请求的 IP 电话，这样主叫方就可获得被叫方的 IP 地址，双方就可以进行通信了。

通过远程电话对智能家电进行控制的原理是，首先采用上述方法进行网络连接，然后一方 IP 电话发送命令，经 Internet 网传输到对方后，对方 IP 电话通串口与智能家电通信。智能家电按照接收到的命令通过处理器的通用 IO 口控制相关部件，从而实现遥操作。同样通过网络也可获得远程家电的运行状态相关信息。下面分别对各个模块的功能进行介绍。

3.5.3 拨号盘的实现原理

图中电话号码显示框，采用 Edit 控件，设置其 ID 属性 IDC_phone_number。定义全局变量 phone_number，用于存放输入的电话号码，为了方便程序设计采用 CString 类型。在本系统中，初步设定电话号码总长为 11 位。对于每一个按键的消息响函数都是先把输入的相应号码存入 phone_number 变量中，然后判断总共输入的号码是否超过 11 位，如果超过 11 位，则显示错误警告。如果未超过，则通过把当前的输入的值与原有值相连接，以形成 11 位电话号码。“1”号按钮的消息响应函数为：

```
void CPhoneDlg::OnButton1()
{
    int a;
    CString lpStr="";
    a=GetDlgItemText(IDC_phone_number,lpStr);
    phone_number=lpStr+'1';
    if(phone_number.GetLength()>11)
    {
        AfxMessageBox(_T("电话号码超过 11 位,请重新输入！"));
    }
    else
    {
        SetDlgItemText(IDC_phone_number,phone_number);
    }
}
```

其它号码按钮的消息响应函数类同。当输入满 11 位号码后，就可以启动语音、短信等相应功能键。

3.5.4 与服务器通信及短信功能的实现

面向对象程序设计是指一种将对象作为程序的基本单元，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性的程序设计模式。面向对象方法是一种把面向对象的思想应用于软件开发过程中，指导开发活动的系统方法。对象是由数据和操作组成的封装体，与客观实体有直接对应关系，一个对象类定义了具有相似性质的一组对象[35]。而继承性是对具有层次关系的类的属性和操作进行共享的一种方式。所谓面向对象就是基于对象概念，以对象为中心，以类和继承为构造机制，来认识、理解、刻画客观世界和设计、构建相应的软件系统。

VEC 开发平台是基于 C++语言的，采用面向对象程序设计思想。为了使程序便于修改和扩充，本程序把不同的功能分成不同的模块，在不同的类中实现。

在和对方通信之前，得先获得对方的 IP 地址，这一功能是由 CAskService 类来实现的，该类的源代码如下：

类的定义：

```
#include<Winsock2.h>
class CAskService
{
private:
   WORD wVersionRequested;
   WSADATA wsaData;
   int err;
   SOCKET sockClient;
   SOCKET sock_get_IP;
   HANDLE hThread;
public:
   static DWORD WINAPI  RecvProc();//线程函数，用于获取被叫方IP
   CAskService();
   virtual ~CAskService();
   void getip(char sign,std::string for_num_to_ip);
};
```

方法的实现为：

```
struct record              //用于存贮用户记录的数据结构
{
   char phone_num[12];     //电话号码
   BYTE  phone_IP[4];      //IP地址
   int      num;           //记录在数据表中的位置
   bool     is_leisure;    //是否占线
   char     sign;          //用于标记查询还是更换新IP
};
record phone_IP_record;     //用于存贮用户记录的数据结构变量
struct asked_record         //用于接收查询结果的结构
{
   char phone_num[12];     //电话号码
   BYTE  phone_IP[4];      //IP地址
}
asked_record as_red;        //用于接收查询结果的结构体变量
CAskService::CAskService()
{
   wVersionRequested = MAKEWORD( 1, 1 );
   err = WSAStartup( wVersionRequested, &wsaData );
   if ( err != 0 ) {
      return;
   }
   if ( LOBYTE( wsaData.wVersion ) != 1 ||
        HIBYTE( wsaData.wVersion ) != 1 ) {
```

```
        WSACleanup( );
        return;
    }
}
void CAskService::getip(char sign, std::string for_num_to_ip)
{
    phone_IP_record.sign=sign;
    phone_IP_record.num=5;
    phone_IP_record.phone_num[11]='\0';
    strcpy(phone_IP_record.phone_num,"02866161171");
    char *sendbuffer=(char*)&phone_IP_record;
    int length=sizeof(phone_IP_record);
    char sHostName[255];
    struct hostent FAR * lpHostEnt=gethostbyname(sHostName);
    if(lpHostEnt==NULL) //产生错误的处理办法
    {
        phone_IP_record.phone_IP[0]=0;
        phone_IP_record.phone_IP[1]=0;
        phone_IP_record.phone_IP[2]=0;
        phone_IP_record.phone_IP[3]=0;
        return GetLastError();
    }
    LPSTR lpAddr=lpHostEnt->h_addr_list[0];
    if(lpAddr)
    {
        struct in_addr inAddr;
        memmove(&inAddr,lpAddr,4);
        phone_IP_record.phone_IP[0]=inAddr.S_un.S_un_b.s_b1;
        phone_IP_record.phone_IP[0]=inAddr.S_un.S_un_b.s_b2;
        phone_IP_record.phone_IP[0]=inAddr.S_un.S_un_b.s_b3;
        phone_IP_record.phone_IP[0]=inAddr.S_un.S_un_b.s_b4;
    }
    sockClient=socket(AF_INET,SOCK_DGRAM,0);
    SOCKADDR_IN addrSrv;
    PHOSTENT hostinfo;
    int len=sizeof(SOCKADDR);
    addrSrv.sin_addr.S_un.S_addr=inet_addr("202.115.143.106");
    //设置连接服务器IP，假设服务器IP为202.115.143.106
    addrSrv.sin_family=AF_INET;
    addrSrv.sin_port=htons(6000);
    connect(sockClient,(SOCKADDR*)&addrSrv,len);
    if(sign=='a')
    {
        send(sockClient,sendbuffer,length,0);
```

```
    }
    if(sign=='b')
    {
       strcpy(phone_IP_record.phone_num, for_num_to_ip.begin());
        send(sockClient, sendbuffer, length, 0);
        hThread=CreateThread(NULL, 0, RecvProc, NULL, 0, NULL);
    }
     //向服务器请求被叫方IP，末尾用字符'b'标记;
     //向服务器发送IP电话的IP，以便让服务器更新数据
     //末尾用字符'a'标记
}
CAskService::~CAskService()
{
    closesocket(sockClient);
    WSACleanup( );
}
DWORD WINAPI CAskService::RecvProc()
{
    listen(sock_get_IP, 1);
    SOCKADDR_IN addrClient;
    int len=sizeof(SOCKADDR);
    int length=sizeof(asked_record);
    int retval;
    while(TRUE)
    {
SOCKET sockConn=accept(sock_get_IP, (SOCKADDR*)&addrClient, &len);
        retval=recv(sockConn, (char*)&as_red, length, 0);
        if(SOCKET_ERROR==retval)
            break;
    }
    return 0;
}
```

在类定义中声明相关变量，WORD wVersionRequested 用于保存 WinSock 库的版本号，结构体变量 as_red 用来保存从服务器中发送过来的数据，因为该数据在其它地方还要用到，所以它被定义为全局变量，让它与进程有同样的生命周期。结构体变量 phone_IP_record 用来存放发送到服务器中的数据，该数据被发送后不做保存。在构造函数中用 MAKEWORD 宏创建一个包含请求版号的 WORD 值，再调用库函数 WSAStartup 加载套节字。函数 getip(char sign)用来获得被叫方的 IP 地址，或者向服务器发送更新的 IP 地址数据。当参数为 a 时，向服务器发送更新的 IP 地址数据，以便让服务中登记的 IP 电话的 IP 地址是最新的。因为如果当一个 IP 电话由于某种原因停机，在下一次重新登入系统时，它的 IP 地址将会发生变化，这时服务器中原有对应该 IP 电话的 IP 地址将是不正确的，所以有

必要重新更新服务器中的数据库，以让它记录的数据与 IP 电话地址变化保持同步。当系统启动后 IP 电话会向服务器传送电话号码与 IP 捆绑的数据包。用户可以用“刷新”键启动刷新功能，以向服务器重新发送当前 IP 地址。IP 电话定时向服务器发送更新的数据。当通信完后，可以不使用该套节字，这时调 closesocket() 函数关闭已经建立连接的套节字。为了释放资源，调用了 WSACleanup( )函数终止对相应库的使用，这两个函数均放在析构函数中调用。由于在接收服务器的数据时，使用的是阻塞接收函数，为了避免程序阻塞，将该函数由线程调用。

在获取对方的 IP 地址后，就可以使用套节字向对方发送短信了。该功能是由类 class CPhoneDlg 里的函数 void CPhoneDlg::OnBUTTONsend()实现的，它是“发送”按钮的消息响应函数。该函数的原代码为：

```
extern struct asked_record as_red;
//全局变量存放收到的IP，它在别的源文件中定义，引入到本文件中
void CPhoneDlg::OnBUTTONsend()
{
   std::string stringtemp;
   std::string ip_phone_number;
   GetDlgItemText(IDC_phone_number,ip_phone_number.begin());
   GetDlgItemText(IDC_EDIT_SENDto,strSend.begin());
   serviceIP.getip('b', ip_phone_number);
   struct in_addr inAddr;
   inAddr.S_un.S_un_b.s_b1= as_red.phone_IP[0];
   inAddr.S_un.S_un_b.s_b2= as_red.phone_IP[1];
   inAddr.S_un.S_un_b.s_b3= as_red.phone_IP[2];
   inAddr.S_un.S_un_b.s_b4= as_red.phone_IP[3];
   SOCKET sockClient=socket(AF_INET,SOCK_STREAM,0);
   SOCKADDR_IN addrTo;
   addrTo.sin_family=AF_INET;
   addrTo.sin_port=htons(8000);
   addrTo.sin_addr.S_un.S_addr=inet_addr(inet_ntoa(inAddr));
   connect(sockClient,(SOCKADDR*)&addrTo,sizeof(SOCKADDR));
   send(sockClient,strSend.begin(),strSend.length()+1,0);
   SetDlgItemText(IDC_EDIT_SENDto,_T(""));
}
```

通过对象调用函数 serviceIP.getip()从服务器中获得被叫方的 IP，定义了一个 string 类对象 ip_phone_number 来存放被叫方的 IP，IP 电话在向服务器查询被叫方 IP 以及向服务器发送数据更新时，发送的都是 17 个字节长。为了方便，这里采用 string 类对象来存放收到的数据，以便从中提取出被查询到的被叫方 IP 地址。然后调用套节字发送用户输入的短信，该功能由函数 send( )实现。发送完毕后，SetDlgItemText(IDC_EDIT_SENDto,_T(""))清空输入框中的数据。

IDC_EDIT_SENDto、IDC_phone_number 分别为发短信输入编辑框和电话号码输入编辑框的 ID 号。

3.5.5 语音功能的实现原理

为了方便用户，本系统可以采用拨号方式和宽带上网，但在采用拨号上网时，需加调制解调器，相关内容文中从略。因为语音通信需要较大的网络带宽，对于拨号上网方式来说通信质量不佳，要获得良好的语音通信效果需采用宽带上网或在局域网上通信。如果仅仅是用来远程控制智能家电，则可采用拨号上网。

实现语音通信的基本原理是，IP 电话端将硬件接口采集到的声音数据通过套节字 Socket 发送到另外一端，另外一端根据得到的声音数据调用硬件接口播放声音。为了方便程序设计将其分为几个模块，服务器 Socket 模块，负责接收连接，它完成对 Socket 的发送数据和接收数据进行消息处理；客户端 Socket 模块，负责接收/发送数据，并完成对 Socket 的发送数据和接收数据进行消息处理；声卡数据的采集和播放模块，采用相关波形音频 API 函数，实现采集声音和播放声音数据。

整个过程基本原理为：首先打开录音设备，获得录音句柄，指定录音格式，分配若干用于录音的缓冲区；开始录音时，先将所有内存块提供给录音设备用来录音，录音设备就会依次将语音数据据写入内存，当一块内存写满，录音设备就会发送一个消息给相应的窗口，通知程序做出相关处理。本程序中使用 Socket 通过网络发送出去，然后把内存块置空，返还给录音设备进行录音，这样就形成一个循环录音过程[10]。

音频数据处理出类 Sound 现实，它使用相关声卡 API，负责采集声音数据和播放声音数据，它的源代码为：

```
#include <mmsystem.h>
#define BUFFER_LENGTH 1024
class CPhoneDlg;
class CSound
{
public:
    CPhoneDlg*          m_dlg;
    WAVEFORMATEX        m_soundFormat;
    HWAVEIN             m_hWaveIn;
    HWAVEOUT            m_hWaveOut;
    WAVEHDR             m_pWaveHdrIn[3];
    WAVEHDR             m_pWaveHdrOut[3];
    char                m_cBufferIn[BUFFER_LENGTH];
    char                m_cBufferOut[BUFFER_LENGTH];
public:
```

```
    void Init(CPhoneDlg * dlg);//初始化函数
    void Record();              //开始录音函数
    void Play();                //播放声音函数
    void StopRecord();          //停止录音函数
    void StopPlay();            //停止播入函数
    void FreeRecordBuffer();    //释放录音的缓冲区
    void FreePlayBuffer();      //释放播放的缓冲区
    CSound();
    virtual ~CSound();
};
```

宏BUFFER_LENGTH用来定义用来录音的内存大小。类CPhoneDlg为应用程序框架类，前向申明类，以便在CSound类中能定义CPhoneDlg类类型的指针，类中的各成员函数的作用如注释所示。成员变量m_soundFormat是结构类型变量，它用来指定录音格式。类的部分方法实现如下：

```
void CSound::Init(CPhoneDlg *dlg)
{
    int retCode;
    m_dlg=dlg;
    if(waveInGetNumDevs()==0)
    {
        AfxMessageBox(_T("未找到音频输入设备！"));
    }
    if(waveOutGetNumDevs()==0)
    {
        AfxMessageBox(_T("未找到音频输出设备！"));
    }
   m_soundFormat.wFormatTag=WAVE_FORMAT_PCM;//指定采样方式为PCM脉冲编
   //码调制方式
    m_soundFormat.nChannels=1;              //单声道
    m_soundFormat.nSamplesPerSec=8000;  //采样率
    m_soundFormat.nAvgBytesPerSec=16000;//数据率
    m_soundFormat.nBlockAlign=2;            //最小块单元
    m_soundFormat.cbSize=0;                 //附加格式信息
    m_soundFormat.wBitsPerSample=16;    //指定录音格式
    retCode=waveInOpen(&m_hWaveIn,WAVE_MAPPER, &m_soundFormat,
(DWORD)m_dlg->m_hWnd,0L,CALLBACK_WINDOW); //打开录音设备
    retCode=waveOutOpen(&m_hWaveOut,WAVE_MAPPER, &m_soundFormat,
(DWORD)m_dlg->m_hWnd,0L,CALLBACK_WINDOW); //打开放音设备
    m_pWaveHdrIn[0].lpData=m_cBufferIn;
    m_pWaveHdrIn[0].dwBufferLength=BUFFER_LENGTH;
    m_pWaveHdrIn[0].dwBytesRecorded=0;
    m_pWaveHdrIn[0].dwFlags=0;
    retCode=waveInPrepareHeader(m_hWaveIn,&m_pWaveHdrIn[0],sizeof(WAV
```

```
EHDR)); //准备内存块录音
    retCode=waveInAddBuffer(m_hWaveIn,&m_pWaveHdrIn[0],sizeof(WAVEHDR
)); //增加内存块
    m_pWaveHdrOut[0].lpData=m_cBufferOut;
    m_pWaveHdrOut[0].dwBufferLength=BUFFER_LENGTH;
    m_pWaveHdrOut[0].dwBytesRecorded=0;
    m_pWaveHdrOut[0].dwFlags=0;
    waveOutPrepareHeader(m_hWaveOut,&m_pWaveHdrOut[0],sizeof(WAVEHDR)
); //准备内存块录音
    waveOutWrite(m_hWaveOut,&m_pWaveHdrOut[0],sizeof(WAVEHDR));
}
void CSound::Record()
{
    waveInStart(m_hWaveIn);//开始录音
}
void CSound::Play()
{
    memcpy(m_cBufferOut,m_cBufferIn, BUFFER_LENGTH);
}
void CSound::StopRecord()
{
    waveInStop(m_hWaveIn);  //停止录音
    waveInReset(m_hWaveIn); //清空内存块
}
void CSound::FreeRecordBuffer()
{
    int retCode;
    retCode
=waveInUnprepareHeader(m_hWaveIn,&m_pWaveHdrIn[0],sizeof(WAVEHDR));
    m_pWaveHdrIn[0].lpData=m_cBufferIn;
    m_pWaveHdrIn[0].dwBufferLength=BUFFER_LENGTH;
    m_pWaveHdrIn[0].dwBytesRecorded=0;
    m_pWaveHdrIn[0].dwFlags=0;
    retCode=waveInPrepareHeader(m_hWaveIn,&m_pWaveHdrIn[0],sizeof(WAV
EHDR)); //准备内存块录音
  retCode=waveInAddBuffer(m_hWaveIn,&m_pWaveHdrIn[0],sizeof(WAVEHDR));
        //增加内存块
    }
```

为了节省篇幅，一些函数调用返回值检测的相关代码在此被省略了。

3.5.6 串口通信的实现

为了提高产品的通用性和可扩展性，智能家电采用 ARM2410 芯片作为其智能控制部分。智能家电基本工作原理为：主叫方 IP 电话通过网络把控制命令发送到

被叫方IP电话，被叫方把数据通过串行口发送到智能家电，智能家电根据命令改变ARM的通用IO口的电平。不同的家电定义了一组不同的操作协议，智能家电将根据电平的变化做出相应的响应，从而实了智能家电的远程控制。同时为了从远程获得智能家电的运行状态，可向智能家电发送信息获取命令，智能家电根据命令读取IO口的电平，再将相关信息发送回主叫方IP电话。

因为串口属于慢速度接口，如果让其等待接收数据，则容易造成进程的阻塞，浪费系统资源。为此在本文程序设计中采了多线程的编程方法，以实现串口异步通信。

串行端口在WinCE下属于流接口设备，是作为文件来进行处理的，而不是直接对端口进行操作。WinCE提供了相应的I/O驱动程序和接口函数，通过调用函数，可实现对串口的操作。串行口编程可分为以下几个步骤：打开串口、配置串口、发送或接收数据以及关闭串口，相关操作函数如下[11]：

1、打开串行口函数

```
HANDLE CreateFile(
  LPCTSTR lpFileName,
  DWORD dwDesiredAccess,
  DWORD dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD dwCreationDisposition,
  DWORD dwFlagsAndAttributes,
  HANDLE hTemplateFile
);
```

在WinCE中不支持异步I/O，因此参数dwFlagsAndAttributes设置为0值。函数的返回值是已打开的串行端口的句柄，用它来标识串行口，在后面函数的调用中将中到。

2、配置串行口

在Windows中外围接口通常采用一个结构体与其关联，通过改变该结构体的成员变量的值来实现对外围接口进行设置配置。其中对于串口有一个结构体设备控制块DCB与其关联，通过改变DCB的成员变量值，来设置串行口的波特率、停止位、数据位长度、校验位、流量控制等。首先通过如下函数获得当前打开的串口配置：

```
BOOL GetCommState(
  HANDLE hFile,
  LPDCB lpDCB
);
```

然后根据需要配置串行口的参数，使用如下的函数：

```
BOOL SetCommState(
  HANDLE hFile,
  LPDCB lpDCB
```

```
);
```

3、读写串行口函数

在 Wince 中读写串口所用的函数和对文件进行读写所用的函数相同，读函数原型如下：

```
BOOL ReadFile(
  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

写串口函数为：

```
BOOL WriteFile(
  HANDLE hFile,
  LPCVOID lpBuffer,
  DWORD nNumberOfBytesToWrite,
  LPDWORD lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped
);
```

4、关闭串行口函数

```
BOOL CloseHandle(
  HANDLE hObject
);
```

串口通信功能采用了面向对象的程序设计方法，类定义的原代码如下：

```
class CSerial
{
public:
   DCB m_dcb;          //设备控制块
   char filename[5];  //串口名
   char mode[sizeof("baud=115200 parity=N data=8 stop=1")]; //存贮Mode命令
   HANDLE m_hCom;                    //串口句柄
   DWORD dwInQueue;                  //接收缓冲区大小
   DWORD dwOutQueue;                 //发送缓冲区大小
   char readbuffer[100];             //存放读取数据的缓冲区
   char writebuffer[100];            //存放发送数据的缓冲区
   unsigned long written_word_num;  //写入串口的字节数
    unsigned long read_word_num;     //读入串口的字节数
public:
   void SendData();    //发送数据
   void ReceiveData(); //接收数据
   bool SetSerial();   //串口配置
   bool OpenSerial();  //打开串口
```

```
    CSerial();
    virtual ~CSerial();
};
```

类的实现代码如下：

```
struct Control_Command
{
    char name[3];           //家电名称
    bool is_right;          //数据包奇偶校验
    int  open_time[2];      //设定开机时间
    int  close_time[2];     //设定关机时间
    bool get_or_send_info; //指定是查询还是向智能家电发送命令
    bool info1;             //从家电中获得运行信息
    bool info2;             //从家电中获得运行信息
    bool open;             //开机
    bool close;            //关机
    bool roate;            //正逆向运行
    bool rate1;            //家电运行档位
    bool rate2;            //家电运行档位
    bool rate3;            //家电运行档位
}Command;
CSerial::CSerial()
{
    strcpy(filename,"COM1");
    strcpy(mode,"baud=115200 parity=N data=8 stop=1");
    dwInQueue=100;
    dwOutQueue=100;
    written_word_num=0;
}
CSerial::~CSerial()
{
    CloseHandle(m_hCom);//关闭串口
}
bool CSerial::OpenSerial()
{
    m_hCom=CreateFile(filename,GENERIC_READ|GENERIC_WRITE,0,NULL,O
PEN_EXISTING,FILE_FLAG_OVERLAPPED,0);
    if(m_hCom)
    {
        return 0;
    }
    else
    {
        SetupComm(m_hCom,dwInQueue,dwOutQueue);//分配发送和接收缓
冲区
```

```
            return 1;
        }
    }
    bool CSerial::SetSerial()
    {
        if(!GetCommState(m_hCom,&m_dcb))//获取当前DCB结构
        {
            return 0;
        }
         if(!BuildCommDCB(mode,&m_dcb))//设置传输率、奇偶校验方法，及
        //数据停止位
        {
            return 0;
        }
        if(!SetCommState(m_hCom,&m_dcb))//设置DCB结构
        {
            return 0;
        }
        return 1;
    }
    void CSerial::ReceiveData()
    {
        ReadFile(m_hCom,readbuffer,sizeof(Command),&read_word_num,NULL
);
    }
    void CSerial::SendData()
    {
        WriteFile(m_hCom,writebuffer,sizeof(Command),NULL);
    }
```

为了便于程序设计和提高系统的通用性，从智能家电的控制需求中抽象出了一个统一的控制命令，它采用一个结构体来表示，如上述程序中定义的结构体 Control_Command。各数据成员所表示的作用如注释所示，其中 name、open_time、close_time 成员变量未映射通用硬件 IO 口，open、roate 等其它成员变量映射了 ARM 的相应通用 IO 口:GPIO_E11、GPIO_E12、GPIO_H4、GPIO_H6、GPIO_H10、GPIO_F4、GPIO_H9、GPIO_B0。前六个通用 IO 口设置为输出模范，用于向智能家电发送命令，后两者为输入模式，用于从家电中获取运行状态信息。

前文已提到由于串行口读写数据速度比较慢， ReadFile、WriteFile 函数会导致程序阻塞，而且本系统中串口通信数据量少，时间不确定。因此为了提高效率，本系统中采用了多线程的程序设计方法。把读写串口的函数作为线程调用函数。由于程序调度器让线程轮回调度，因而这样实现了串口的异步工作，提高了工作效率。为了节省篇幅，代码中有部分被省略了。

实现方法为，由 CPhoneDlg 类中的“启用家电控制”按钮的消息响应函数创建两个线程，分别负责串口的数据发送和数据接收，代码如下：

```
void CPhoneDlg::Onopenserial()
{
  hthread1=CreateThread(NULL, 0, Fun1Proc, NULL, 0, NULL);
  hthread2=CreateThread(NULL, 0, Fun2Proc, NULL, 0, NULL);
}
DWORD WINAPI CPhoneDlg::Fun1Proc(LPVOID lpParameter)
{
  WaitForSingleObject(hMutex, INFINITE);
  m_serial.ReceiveData();
  ReleaseMutex(hMutex);
  return 0;
}
DWORD WINAPI CPhoneDlg::Fun2Proc(LPVOID lpParameter)
{
  WaitForSingleObject(hMutex, INFINITE);
  m_serial.SendData();
  ReleaseMutex(hMutex);
  return 0;
}
```

因为串口的数据接收区和发送缓冲区是多个线程所共享的，Socket 发送过来的数据要存入串口缓冲区中，这样如果不采取一定的保护措施，就会导致不干净的共享数据，因此这里采用了互斥对象来保持线程同步。

第4章 嵌入式 Linux 移植、驱动编写及应用程序的开发

因为智能家电主要是面向控制的，不需要多媒体功能，因此智能家电端采用 Linux 操作系统。Linux 内核占用的内存少，可以节省在内空间的开销，而且 Linux 是免费的，相对需付费的 WinCE 操作系统来说 Linux 产品成本低。

一个嵌入式 Linux 系统从软件的角度来看，通常可以分为四个层次：引导加载程序，它是系统启动后运行的第一段代码，进行软硬件的初始化； Linux 内核，特定的嵌入式硬件平台定制不同的内核和启动参数，驱动程序也包括在内；文件系统，包括根文件系统和建立于 FLASH 内存设备之上的文件系统；应用程序；实现用户需求的应用程序。下面针对各个部分进行说明。

4.1 BootLoader 特点及烧写

嵌入式 Linux 的启动过程与 PC 机类似，它也需要一个系统引导的过程。与 PC 机不同的是，嵌入式的引导是由一个叫做 BootLoader 的程序来完成的。它是系统加电后运行的第一段代码，在操作系统内核运行之前执行的一段程序。通过它，可以初始化硬件设备、建立内存空间的映射图，从而将系统的硬件带到一个合适的状态，为操作系统内核的运行准备好良好的环境。

在一个基于 ARM 的嵌入式系统中，系统在上电或复位时可根据系统引脚配置从 0x0 地址处开始执行，因而通常就在这个地址处放置 Bootloader 引导程序镜像文件，以使系统上电后能完成自动引导功能，启动 Linux 操作系统。

Bootloader 启动分为两个阶段。第一阶段包含依赖于 CPU 的体系结构硬件初始化的代码，通常用汇编语言来实现。这个阶段的任务是：基本的硬件设备初始化，为第二阶段准备 RAM 空间，复制 Bootloader 第二段代码到 RAM，设置堆栈，跳转到第二阶段的 C 程序入口点等。第二阶段通常用 C 语言完成，以便实现更复杂的功能。这个阶段的任务有：初始化本阶段要使用的硬件设备，检测系统内存映射，将内核映射和根文件系统映像从 FLASH 读到 RAM，为内核设置启动参数，调用内核[13]。

本系统在调试阶段采用了周立功单片机公司的 Bootloader，因为它同时支持 Linux 和 WinCE 两个系统的启动，而本系统用到了两种操作系统，这样方便调试。为了提高系统的启动速度，待程序调试成功后，采用另外一种 Bootloader，即 U-BOOT 来启动系统。把 U-BOOT 源代码里调试端口功能相关以及其它一些附属部件去掉，编译重新生成新的 Bootloader 镜像。

下载 Bootloader 镜像步骤：先将源代码编译生成镜像文件，在 PC 机上安装 H-JTAG 软件作为 ARM 调试代理，运行该程序检测 ARM 内核，然后使用仿真器 wiggLe 连接 JTAG 口和计算机并口。用 ADS 启动 WR_NORFLASH.mcp 工程，设置 ARM 为 NOR FLASH 启动方式，在 ADS 中选择 DebugRel 生成目标，进入 AXD 调试环境。在语句 while(1)处设置断点，然后全速运行程序。这样可以把 Bootloader 镜像烧写入从地址 0x0 开始的 NOR FLAH 中。然后复位系统，在超级终端中可看到载有目标板信息的 Bootloader 提示符[4]。

4.2 内核裁剪与文件系统创建

为了能够在 ARM 平台上运行嵌入式 Linux，需要对 Linux 操作系统进行移植和剪裁。移植就是把 Linux 操作系统源代码针对 ARM 微处理器的特点做必要的修改，使之能在新的硬件平台上正常运行。本系统采用了三星公司 ARM2410 芯片对应的 Linux 移植版本源代码，根据具体系统的需要对内核进行修改、裁剪和配置就可编译得到内核镜像。

Linux 内核的裁剪主要是加入对系统需要的模块的支持，去掉不用的模块。它实际上就是条件编译，具体受一些以宏定义形式出现的条件编译控制量的控制，而这些控制量定义又是在 make 的过程中根据配置文件自动生成的，除条件编译外，还可以通过条件连接决定一些模块的取舍。这样可以根据实际系统的需求精简内核，使其占用的空间更小，减小存储器的消耗。Linux 自带有配置工具，可通过命令 make menuconfig 对内核进行配置，它读取配置文件“smdk2410”来设定默认值。通过菜单选项，可以修改内核配置，配置完成后，在内核根目录下重新生成.config 文件，此文件包含了配置过程中对选项所做设定的全部细节。内核的许多功能和驱动程序以模块的形式存在，通过配置将它们建立在内核中。配置过程中，所选的项目根据目标板的具体硬件而定。因为本系统中用到了串行口，因此选取内核对串口的支持，并把一些在目标板中没有的相应硬件功能支持模块去掉。在完成内核的配置后，即可通过以下命令进行内核的编译：

1、make clean：　删除前面步骤留下来的文件，避免出现一些错误。

2、make dep：读取配置过程生成的配置文件来创建内核源码间的依赖关系，并在各子目录下生成.depend 文件。

3、make zlmage：生成压缩的内核映像文件 zlmage。

Linux 内核在系统启动后要安装根文件系统，文件系统是嵌入式操作系统的一部分，它的任务是对逻辑文件进行复制、删除、修改等操作管理，同时也方便用户操作文件和目录。

Linux 内核支持多种文件系统，如 JFFS 文件系统、CRAMFS 文件系统等，其中 CRAMFS 文件系统是一种只读文件系统，可以保证内核文件和应用软件在系统运行时不会被破坏，提高了系统的可靠性，特别适合于嵌入式系统。因此本系统选择 CRAMFS 文件系统作为 Linux 根文件系统。根文件系统包含内核所需的文件、用于系统管理的配置文件和可执行文件。

下面创建文件系统，该操作是在 PC 机里的 VMware 虚拟机里进行的。VMware 虚拟机里运行的是红帽 Linux9.0。步骤如下：

1、 在 Linux 操作系统环境下，生成可以虚拟成块设备的文件，取文件名为 init.img，命令如下：

$ dd if=/dev/zero of=init.img bs=4096 count=1024

其中 bs*count 为块设备大小。生成 init.img 文件以后，再对该文件进行格式化，命令如下：

$ mke2fs –m0 –F init.img

2、新建一个文件夹 ram，并将 init.img 挂接到 ram 目录，命令如下：

$ mkdir ram

$ mount init.img ram/ -o loop

这时，由于映射关系，读写 ram 目录，就等效于读写 init.img 文件。再将根文系统所需的文件和本系统的应用软件写入到 ram 目录中。往 ram 目录写完文件以后，使用如下命令：

$ umount ram

卸载 init.img，这时将已写入的文件保存到 init.img 中。

即此，根文件系统创建完毕业，再采用网口将其与内核和配置文件一起下载到 NAND FLASH 存储器里，方法在下文介绍。

4.3 交叉编译环境的搭建

由于嵌入式系统资源缺乏，它没有足够大的内存，以及其处理器的速不快等原因，因此对于嵌入式系统来说，在其中运行的操作系统的移植，以及应用软件的开发，都是在 PC 嵌入机上进行，这样充分发挥 PC 机的优点。这种方式叫做交叉编译。通交叉编译在一个平台上生成另一个平台上的可执行代码。

嵌入式系统开发环境分为宿主机和目标机。宿主机是开发平台，用于运行开发过程中的各种工具；目标机是运行和测试平台，是嵌入式系统的最终驻留环境。在宿主机和目标机之间需要通过一定的方式进行通信，发送控制指令和传输数据，同时获得目标机的状态信息，以控制整个开发与调试过程。本文开发中使用串口和以太网口进行通信。

由于整个开发过程设及到操作系统内核调试以及驱动调试，这样很容易破坏宿主机操作系统，而且很多在 Windows 环境下的帮助文档格式在 Linux 环境下没有相应的应用软件，所以为了方便，采用 PC 机与虚拟机联合工作的方式。宿主机的 RedhatLinux9 操作系统运行在 VMware 虚拟机上。由于 VMware5.5 虚拟机上无法实现串口通信，因此，开发时采用 Windows 环境下的超级终端，实现宿主机与目标机的串行口通信。

由于宿主机与目标机分别基于不同的微处理器，宿主机是基于 X86 微处理器的，而目标机是基于 ARM2410 处理器的，两种处理器具有不同的指令系统。因此，两种平台下应用程序的开发不能采用一样的编译器。在宿主机上运行的应用程序开发采用 GNU 的跨平台开发工具链 gcc，而在目标机上应用程序开发采用交叉编译器 arm-linux-gcc。

在 PC 机上开发应用程序，可以在本机上调试，而在嵌入式中开发程序，不能在本机上调试。一种常见的办法是先用针对宿主机微处里器的编译器进行编译，生成可执行文件，在宿主机上调试通过后，再用交叉编译器编译进行编译，然后再把应用软件集成到嵌入式操作系统中，最后把镜像和文件系统烧写到嵌入式存储器中，验证通过即可。但是这种方法十分麻烦，对存储器的烧写过多会影响其使用寿命，而且调试花费时间长。因此采用这种方法不利于开发的进行。

在应用程序开发过程中，采用 NFS 网络文件系统服务进行系统调试。NFS 是 Linux 操作系统下一种在网络上的机器间共享文件的方法，这里用于目标系统和主机系统共享相同的文件目录。宿主机系统就可以通过 NFS 服务向目标机系统提供一个共享文件，通过这种映射，在主机上的操作就相当于在目标机上的操作，这样就可以在宿主机上执行和调试程序而不必传送到目标系统中。而对目标机上的操作又是通过宿主机的超级终端向目标机发送命令来完成的，因此采用这种方法行之有效。

4.3.1 内核、配置文件和文件系统下载

在进行应用程序开发之前得下载操作系统镜像和文件系统。可以采用串口下载和 JTAG 口进行下载，但下载速度慢。采用周立功公司的 Bootloader 可以在目标机上创建一个 TFTP，这样就可以利用网口快速下载文件。下载工具采用的是 FlashFXP，它运行于宿主机的 Windows 环境。在 Bootloader 启动后所创建的 TFTP 的 IP 地址默认为：192.168.15.115，首先设置 PC 机的 IP 地址在同一网段，然后启动 FlashFXP，就可以烧写镜像和文件系统了。如图 4-1。

4.3.2 交叉编译器的安装

在 PC 主机上构建嵌入 Linux 开发环境，需要安装适合于目标处理器的交叉编译器和调试器。交叉编译环境就是运行在宿主机上的一套由编译器、连接器和 libc 库等组成的综合开发环境，主要由 binutils、gcc 和 glibc 几个部分组成。在编译 glibc 时，要用到 Linux 内核 include 目录中的头文件，因此在建立交叉编译

图4-1 下载内核和文件系统

环境之前还要准备好内核源代码文件。本系统使用的交叉工具链是针对 ARM 的 cross_2.95.3 版。具体安装方法为：

1、在/usr/local 目录下建立目录，命令如下：

mkdir /usr/local/arm

2、进入 arm 目录,将工具链 cross_2.95.3 复制到该目录下，然后把这个文件解宿，命令如下：

tar jxvf cross_2.95.3

3、修改/etc/profile 文件，增加编译器路径，方法如下：

```
# Path manipulation
if [`id –u `=0];then
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /local/sbin
    pathmunge /usr/locat/arm/2.95.3/bin    //增加路径
 fi
```

即此，Linux 的内核编译、文件系统制作、以及交叉编译环境搭建均已完成，接下来的工作就是编写智能家电端的驱动和应用程序。

4.4 驱动程序编写

为了使用户能够在应用程序中方便、快捷、可靠地操纵计算机硬件，在操作系统中，需要内核建立应用程序和设备之间的抽象接口，以隐藏底层硬件的差异，使得应用程序通过这一接口实现对硬件的使用和控制而不必考虑硬件的操作细节，从而避免了应用程序直接操作硬件。操作系统通过设备驱动程序来实现这一功能。它在操作系统内核中具有最高特权级、可驻留内存、可共享底层硬件处理例程。

本系统智能家电与 IP 电话通信采用的是串行口，因此需要串行口驱动。Linux 已经提供了标准设备的驱动，在配置内核时选择内核对串行口的支持。智能家电端采用通用 IO 口进行控制，它属于非标准接口，必须由设计人员根据接口定义自行设计驱动程序。

在 Linux 操作系统中通过设备驱动程序为应用程序提供了统一抽象的接口，它界于内核和设备硬件之间，隐藏了不同设备之间的细节。使得对硬件设备的操作就像对通常的文件一样，利用标准的系统调用可在设备上进行打开、关闭和读写操作。内核提供一系列称为设备入口点的函数接口，当内核接到用户系统调用的时候将启用这些函数，驱动程序的主要任务就是实现这些入口点函数[16]。

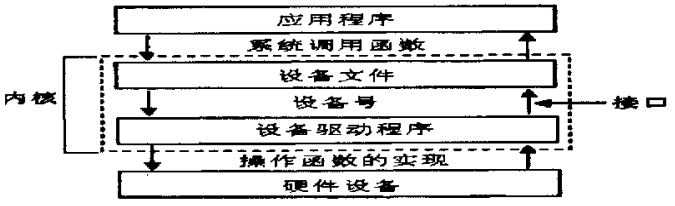

图4-2 驱动程序架构

系统中的每个设备由“设备文件”代表，每个不同的设备具有不同的主设备号，由相同的设备驱动程序控制的设备具有相同的主设备号，用次设备号来区分同类设备中的各个设备。图 4-2 为 Linux 设备驱动程序的架构。

为了方便驱动程序设计，Linux 操作系统对驱动程序定义了一套规范，驱动程序的实现要按照这个标准进行。它要求完成对设备初始化和释放，对设备进行管理，读取应用程序传送给设备文件的数据和回送应用程序请求的数据，测试和处理设备出现的错误等功能。驱动常包含以下基本操作函数的实现，根据实际需求，可有所增减[16]。

1、 init

驱动程序需要进行初始化，在把它载入系统的时候调用这个初始化方法，以实现检测设备、配置和初始化硬件、注册设备和申请中断等。

2、 open

open 方法打开设备，它会增加设备计数，以便防止文件在关闭前模块被卸载出内核。

3、release

release 方法释放申请的资源，减少设备计数。

4、 read

Linux 分内核空间和用户空间，read 方法实现将数据从内核空间复制到用户空间。

5、 write

write 方法实现将数据从用户空间复制到内核空间，以写到设备中去。

6、ioctl

ioctl 方法通过发送不同的命令来实现对硬件的控制。

在前文已经提到，采用一个结构体 Control_Command 来存储控制命令。当一个控制命令数据包传送到智能家电端时，为了方便，在应用程序中调用驱动中 ioctl 方法，对接收进来的数据包先进行编码，使命令控制数据包与一组整数建立一一映射关系，不同的家电的控制命令是不一样的，下面以电风扇的控制为例进行说明。结构体 Control_Command 中的前 6 项与 ioctl 方法无关，因此不须参与编码。对剩下的 7 项进行编码。对应关系如表 4-1。

表 4-1 控制命令表

位 / 编码	open	close	roate	rate1	rate2	rate3	get_or_send_info	功能
0x00	0	0	0	0	0	0	0	关机
0x20	0	1	0	0	0	0	0	待机
0x42	1	0	0	0	0	1	0	开机正转
0x44	1	0	0	0	1	0	0	开机正转
0x46	1	0	0	0	1	1	0	开机正转
0x48	1	0	0	1	0	0	0	开机正转
0x4A	1	0	0	1	0	1	0	开机正转
0x4C	1	0	0	1	1	0	0	开机正转
0x4E	1	0	0	1	1	1	0	开机正转
0x52	1	0	1	0	0	1	0	开机逆转
0x54	1	0	1	0	1	0	0	开机逆转
0x56	1	0	1	0	1	1	0	开机逆转
0x58	1	0	1	1	0	0	0	开机逆转
0x5A	1	0	1	1	0	1	0	开机逆转
0x5C	1	0	1	1	1	0	0	开机逆转
0x5E	1	0	1	1	1	1	0	开机逆转
0x41	1	0	0	0	0	0	1	获取信息

下面是驱动程序主要实现源代码：

```
#define DEVICE_NAME "arm2410_control"
```

```
    //模块声明
    MODULE_LICENSE("Proprietary");
    MODULE_DESCRIPTION("use this device to control electric
equipment");
    MODULE_SUPPORTED_DEVICE("Linux 2.4 & ARM2410");
    MODULE_AUTHOR("Deng-yi-dong");
    //定义一下数组，方便程序设计
    static unsigned long arm2410_control [] =
    {
      GPIO_E11,
      GPIO_E12,
      GPIO_H4,
      GPIO_H6,
      GPIO_H10,
      GPIO_F4,
      GPIO_H9,
      GPIO_B0
    };
    static int arm2410_control_read(struct file *filp,char *buf,size_t
count,loff_t *offset) // read实现方法
    {
      struct port{
      bool con6;
      bool con7;
      };
      port port1;
      prot1.con6=*arm2410_control[6];
      prot1.con7=*arm2410_control[7];
      char *p=(char *)&port1;
      copy_to_user(buf,p,sizeof(port1));
    }
    static int arm2410_control_ioctl(struct inode *inode, struct file
*filp, unsigned int cmd, unsigned long arg) //ioctl实现方法
    {
      int i=0;
      switch(cmd)  //以电风扇为例的操作方法,不同的家电操作方法不一样
      {
      case 0x00:
         fot(i=0;i<8;i++)
         {
              write_gpio_bit(arm2410_control[i], 0);
         }
      case 0x20:
           fot(i=2;i<8;i++)
```

```
    {
        write_gpio_bit(arm2410_control[i], 0);
    }
    write_gpio_bit(arm2410_control[0], 0);
    write_gpio_bit(arm2410_control[1], 1);
case 0x42:
     fot(i=1;i<4;i++)
    {
        write_gpio_bit(arm2410_control[i], 0);
    }
    write_gpio_bit(arm2410_control[5], 1);
    write_gpio_bit(arm2410_control[6], 0);
case 0x44:
     fot(i=1;i<3;i++)
    {
        write_gpio_bit(arm2410_control[i], 0);
    }
    write_gpio_bit(arm2410_control[0], 1);
    write_gpio_bit(arm2410_control[4], 1);
    write_gpio_bit(arm2410_control[5], 0);
    write_gpio_bit(arm2410_control[6], 0);
case 0x46:
     fot(i=1;i<3;i++)
    {
        write_gpio_bit(arm2410_control[i], 0);
    }
    write_gpio_bit(arm2410_control[0], 1);
    write_gpio_bit(arm2410_control[4], 1);
    write_gpio_bit(arm2410_control[5], 1);
    write_gpio_bit(arm2410_control[6], 0);
case 0x48:
     fot(i=4;i<6;i++)
    {
        write_gpio_bit(arm2410_control[i], 0);
    }
    write_gpio_bit(arm2410_control[0], 1);
    write_gpio_bit(arm2410_control[3], 1);
    write_gpio_bit(arm2410_control[1], 0);
    write_gpio_bit(arm2410_control[2], 0);
case 0x4A:
    write_gpio_bit(arm2410_control[0], 1);
    write_gpio_bit(arm2410_control[1], 0);
    write_gpio_bit(arm2410_control[2], 0);
    write_gpio_bit(arm2410_control[3], 1);
```

```
        write_gpio_bit(arm2410_control[4], 0);
        write_gpio_bit(arm2410_control[5], 1);
        write_gpio_bit(arm2410_control[6], 0);
    case 0x4C:
        write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
        write_gpio_bit(arm2410_control[2], 0);
        write_gpio_bit(arm2410_control[3], 1);
        write_gpio_bit(arm2410_control[4], 1);
        write_gpio_bit(arm2410_control[5], 0);
        write_gpio_bit(arm2410_control[6], 0);
    case 0x4E:
          write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
          write_gpio_bit(arm2410_control[2], 0);
        for(i=3;i<5;i++)
        {
           write_gpio_bit(arm2410_control[i], 1);
        }
        write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
        write_gpio_bit(arm2410_control[6], 0);
    case 0x52:
        write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
        write_gpio_bit(arm2410_control[2], 1);
        write_gpio_bit(arm2410_control[3], 0);
        write_gpio_bit(arm2410_control[4], 0);
        write_gpio_bit(arm2410_control[5], 1);
        write_gpio_bit(arm2410_control[6], 0);
    case 0x54:
        write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
        write_gpio_bit(arm2410_control[2], 1);
        write_gpio_bit(arm2410_control[3], 0);
        write_gpio_bit(arm2410_control[4], 1);
        write_gpio_bit(arm2410_control[5], 0);
        write_gpio_bit(arm2410_control[6], 0);
    case 0x56:
        write_gpio_bit(arm2410_control[0], 1);
        write_gpio_bit(arm2410_control[1], 0);
        write_gpio_bit(arm2410_control[2], 1);
        write_gpio_bit(arm2410_control[3], 0);
        write_gpio_bit(arm2410_control[4], 1);
```

```
            write_gpio_bit(arm2410_control[5], 1);
            write_gpio_bit(arm2410_control[6], 0);
        case 0x58:
            write_gpio_bit(arm2410_control[0], 1);
            write_gpio_bit(arm2410_control[1], 0);
            write_gpio_bit(arm2410_control[2], 1);
            write_gpio_bit(arm2410_control[3], 1);
            write_gpio_bit(arm2410_control[4], 0);
            write_gpio_bit(arm2410_control[5], 0);
            write_gpio_bit(arm2410_control[6], 0);
        case 0x5A:
            write_gpio_bit(arm2410_control[0], 1);
            write_gpio_bit(arm2410_control[1], 0);
            write_gpio_bit(arm2410_control[2], 1);
            write_gpio_bit(arm2410_control[3], 1);
            write_gpio_bit(arm2410_control[4], 0);
            write_gpio_bit(arm2410_control[5], 1);
            write_gpio_bit(arm2410_control[6], 0);
        case 0x5C:
            write_gpio_bit(arm2410_control[0], 1);
            write_gpio_bit(arm2410_control[1], 0);
            write_gpio_bit(arm2410_control[2], 1);
            write_gpio_bit(arm2410_control[3], 1);
            write_gpio_bit(arm2410_control[4], 1);
            write_gpio_bit(arm2410_control[5], 0);
            write_gpio_bit(arm2410_control[6], 0);
        case 0x5E:
            write_gpio_bit(arm2410_control[0], 1);
            write_gpio_bit(arm2410_control[1], 0);
            write_gpio_bit(arm2410_control[2], 1);
            write_gpio_bit(arm2410_control[3], 1);
            write_gpio_bit(arm2410_control[4], 1);
            write_gpio_bit(arm2410_control[5], 1);
            write_gpio_bit(arm2410_control[6], 0);
        case 0x41:
              read_gpio_bit(arm2410_control[6]);
              read_gpio_bit(arm2410_control[7]);
        default:
            return -EINVAL;
        }
    }
    static int arm2410_control_open(struct inode *inode, struct file
*filp) //open实现方法
    {
```

```
    int i;
    for (i = 0; i < 6; i++)
    {
        set_gpio_ctrl (arm2410_control[i] | GPIO_PULLUP_EN |
GPIO_MODE_OUT);
        write_gpio_bit(arm2410_control[i], 0);
    }
     for (i = 6; i < 8; i++)
    {
        set_gpio_ctrl (arm2410_control[i] | GPIO_PULLUP_EN |
GPIO_MODE_IN);
    }
    MOD_INC_USE_COUNT;
    printk(KERN_INFO DEVICE_NAME ": opened.\n");
    return 0;
}
static int arm2410_control_release(struct inode *inode, struct file
*filp) //release实现方法
{
    MOD_DEC_USE_COUNT;
    printk(KERN_INFO DEVICE_NAME ": released.\n");
    return 0;
}
//操作接口定义
static struct file_operations arm2410_control_fops =
{
    owner:   THIS_MODULE,
    read:  arm2410_control_read,
    ioctl:   arm2410_control_ioctl,
    open:    arm2410_control_open,
    release: arm2410_control_release,
};
static devfs_handle_t devfs_handle;//创建设备文件系统句柄
static int __init arm2410_control__init(void) //注册设备
{
    devfs_handle = devfs_register(NULL, DEVICE_NAME,
DEVFS_FL_AUTO_DEVNUM,
        0, 0, S_IFCHR | S_IRUSR | S_IWUSR, &arm2410_control_fops,
NULL);
    printk(KERN_INFO DEVICE_NAME ": Initialize OK.\n");
    return 0;
}
static void __exit arm2410_control_exit(void) //注消设备
{
```

```
    devfs_unregister(devfs_handle);
}
module_init(arm2410_control_init); //模块初始化
module_exit(arm2410_control_exit); //退出模块
```

在 Linux2.4 开始利用设备文件系统来管理设备，避免了采用主设备号和次设备号带来的局限。open 实现方法中，调用 set_gpio_ctrl 函数设置通用 IO 口的工作模式，本函数由内核和 arm-linux-gcc 编译器提供。在 8 个通用 IO 口中，有 6 个用于数据的输出，2 个用于数据的输入。在控制的应用程序中，采用了一系列智能控制算法，实现对 IO 口的读写操作。

驱动程序必须编译成.ko 文件才能加载到操作系统内核当中。它的编译通过 make 命令调用 Makefile 文件来实现。Makefile 文件是 Linux 操作系统下定义程序编译和连接规则描述的文件。make 命令就用于解释 Makefile 文件中的指令。上述驱动的 Makefile 文件代码如下：

```
EXEC = arm2410_control.ko
OBJS = arm2410_control.o
SRC  = arm2410_control.c
INCLUDE = /zylinux/kernel/include
USEINC =.
CC = arm-linux-gcc
LD = arm-linux-ld
MODCFLAGS = -O2 -Wall -D__KERNEL__ -DMODULE -I$(INCLUDE) -I$(USEINC)
-march=armv4t -c -o
LDFLAGS = -r
all: $(EXEC)
$(EXEC): $(OBJS)
      $(LD) $(LDFLAGS) -o $@ $(OBJS)
%.o:%.c
      $(CC) $(MODCFLAGS) -mapcs -c $< -o $@
clean:
      -rm -f $(EXEC) *.o *~ core
```

Makefile 文件指定了在编译和连接程序时采用的编译器、连接器、归档器等编译链接工具，所用函数库为在/zylinux/kernel/include 目录，编译的源文件为 arm2410_control.c，目标文件为 arm2410_control.o。Makefile 文件最后的 clean 命令，用于删除所有在编译过程中生成的中间文件。Makefile 文件设计完成后在驱动程序所在目录下通过 make 命令即可完成驱动程序的编译，生成可执行文件 arm2410_control.ko。驱动程序可以采用二种方法加入内核，第一种方法是直接编译进内核，但这种方法不方便调试，另一方法是在系统运行时加载进内核，这里采用的是第二种方法，使用如下命令：

```
#insmod arm2410_control.ko
```

把驱动加入内核以后，就可以启动应用程序调用，从而实现对硬件的控制。测试通过后，为了方便设备运行把驱动编译进内核。

4.5 应用程序编写

在智能家电端，系统的基本工作原理是，从 IP 电话串口接收控制命令数据，再按照上述方法，对命令数据进行编码。如果控制命令结构中 get_or_send_info 位为 1，则命令被指定为查询，这时智能家电通过两输入 IO 口，读取家电的当前运行信息，如果 get_or_send_info 位为 0，则是向智能家电发送命令。根据命令的不同，调用驱动设置通用 IO 口的电平，控制家电的运行状态，从而实现远程控制功能。

为了实现各智能家电串口能同时收发数据，采用全双工 MAX488 型 485 通讯器对 S3C2410 芯片的串口 2 进行扩展，实现 RS485 串行通讯标准。RS485 支持总线型连接方式，因此所有家电均通过 RS485 串行口与 IP 电话连接。IP 电话作为主机，各智能家电作为客户机。下面是电风扇的运行控制程序部分源代码。

在智能家电端，用于存放从 IP 电话端传来的控制命令数据所采用的变量为前文提到的 Control_Command 结构体。

为了使 IP 电话能与智能家电完成通信，双方的对串口的设置应该一样，双方应该使用一样的波特率。智能家电端的串口初始化函数为：

```
void init_ttys(int fd)
{
   struct termios options;
   bzero(&options,sizeof(options));
   cfsetispeed(&options,B115200);
   cfsetospeed(&options,B115200);
   options.c_cflag|=(CS8|CLOCAL|CREAD);
   options.c_iflag=IGNPAR;
   tcflush(fd,TCIFLUSH);
   tcsetattr(fd,TCSANOW,&options);
}
```

由于串口速度慢，且它接收命令是随机的，因此对串口的读取函数放到一个线程中去处理。该函数的实现如下：

```
void receive_data(void * parameter)  //(int fd,char *rcv_buf,int
rcv_wait)
{
   param1 *p=(param *)parameter;
   int retval;
   fd_set rfds;
   struct timeval tv;
```

```
int ret,pos=0;
tv.tv_sec=p->rcv_wait;
tv.tv_usec=0;
while(1)
{
    FD_ZERO(&rfds);
    FD_SET(p->fd,&rfds);
    retval=select(p->fd+1,&rfds,NULL,NULL,&tv);
    if(retval==-1)
    {
        perror("select()");
        break;
    }
    else if(retval)
    {
        pthread_mutex_t work_mutex; //加入互斥量
        int res=pthread_mutex_init(&work_mutex,NULL);
        if(res!=0)
        {
            perror("nutex init fail");
            exit(1);
        }
        pthread_mutex_lock(&work_mutex);
        ret=read(p->fd,p->rev_buf+pos,length);
        pthread_mutex_unlock(&work_mutex);
        pthread_mutex_destroy((&work_mutex);
        pos+=ret;
        if(p->rec_buf[pos-2]=='\r' && p->rev_buff[pos-1]=='\n')
        {
            FD_ZERO(&rfds);
            FD_SET(p->fd,&rfds);
            retval=select(p->fd+1,&rfds,NULL,NULL,&tv);
            if(!retval)
            {
                break;
            }
        }
    }
    else
    {
        printf("NO DATA!\n");
        break;
    }
}
```

```
    }
```

当把命令从 IP 电话发送到了智能家电端后，智能家电对接收的信号进行分析，对发送给本机的控制命令进行处理。因为读进程和用于处理命令的进程用到了同一全局变量，为了维护数据的一致性，程序中采用了互斥量进行多个线程同步。

在对接收的命令进行处理之前，首先要对接收到的命令做编码处理，以便把命令相对应的编码传给驱动程序。实现这个转换的函数如下：

```
int change_to_info(char *info)
{
   Control_Command *p=(Control_Command *)info;
   if(p->get_or_send_info==1)
   {
      return 0x41;
   }
   else
   {
      if(p->close==1)
      {
         return 0x20;
      }
      else
      {
         if(p->close==0 && p->open==0)
         {
            return 0x00;
         }
         else
         {
            if(p->open==1 && p->close==0)
            {
               if(p->roate==0 && p->rate1==0 && p->rate2==0 &&
p->rate3==1)
                  return 0x42 ;
               if(p->roate==0 && p->rate1==0 && p->rate2==1 &&
p->rate3==0)
                  return 0x44 ;
               if(p->roate==0 && p->rate1==0 && p->rate2==1 &&
p->rate3==1)
                  return 0x46 ;
               if(p->roate==0 && p->rate1==1 && p->rate2==0 &&
p->rate3==0)
                  return 0x48 ;
               if(p->roate==0 && p->rate1==1 && p->rate2==0 &&
```

```
p->rate3==1)
                    return 0x4A ;
                if(p->roate==0 && p->rate1==1 && p->rate2==1 &&
p->rate3==0)
                    return 0x4C ;
                if(p->roate==0 && p->rate1==1 && p->rate2==1 &&
p->rate3==1)
                    return 0x4E ;
                if(p->roate==1 && p->rate1==0 && p->rate2==0 &&
p->rate3==1)
                    return 0x52 ;
                if(p->roate==1 && p->rate1==0 && p->rate2==1 &&
p->rate3==0)
                    return 0x54 ;
                if(p->roate==1 && p->rate1==0 && p->rate2==1 &&
p->rate3==1)
                    return 0x56 ;
                if(p->roate==1 && p->rate1==1 && p->rate2==0 &&
p->rate3==0)
                    return 0x58 ;
                if(p->roate==1 && p->rate1==1 && p->rate2==0 &&
p->rate3==1)
                    return 0x5A ;
                if(p->roate==1 && p->rate1==1 && p->rate2==1 &&
p->rate3==0)
                    return 0x5C ;
                if(p->roate==1 && p->rate1==1 && p->rate2==1 &&
p->rate3==1)
                    return 0x5E ;
            }
        }
    }
}
```

当接收到命令后，根据命令的不同，采取不同的操作：如果是控制家电的命令则根据命令改变相应 IO 的电平，以实现家电的控制；如果命令是查询命令，则从两个相应端口读取 IO 电平，把这两个值与接收到的命令捆绑后，通过串口发送给 IP 电话，再从被叫 IP 电话把数据发送给主叫方 IP 电话。这样实现了控制家电和查询家电运行信息的功能。这由如下函数来实现：

```
void is_received_to_send(void * cmd_buf)
{
    /判断是否接收到命令,并做出相应处理，由线程调用
    int order=0;
```

```
command_buffer *p1=(command_buffer *)cmd_buf;
char *p2=(char *)p1->p;
Control_Commandm *p3=p1->p;
if(p3->name[0]=='f' && p3->name[1]=='a' && p3->name[2]=='n')
{
    pthread_mutex_t work_mutex;//加入互斥量进行同步
    int res=pthread_mutex_init(&work_mutex,NULL);
    if(res!=0)
    {
        perror("nutex init fail");
        exit(1);
    }
    pthread_mutex_lock(&work_mutex);
    order=change_to_info(p2);
    pthread_mutex_unlock(&work_mutex);
    pthread_mutex_destroy((&work_mutex);
    if(order==0x00)
    {
        ioctl(p1->fd,0x00,0);
        FD_ZERO(p1->p);
    }
    if(order==0x20)
    {
        ioctl(p1->fd,0x20,0);
        FD_ZERO(p1->p);
    }
    if(order==0x42)
    {
        ioctl(p1->fd,0x42,0);
        FD_ZERO(p1->p);
    }
    if(order==0x44)
    {
        ioctl(p1->fd,0x44,0);
        FD_ZERO(p1->p);
    }
    if(order==0x46)
    {
        ioctl(p1->fd,0x46,0);
        FD_ZERO(p1->p);
    }
    if(order==0x48)
    {
        ioctl(p1->fd,0x48,0);
```

```
    FD_ZERO(p1->p);
}
if(order==0x4A)
{
    ioctl(p1->fd, 0x4A, 0);
    FD_ZERO(p1->p);
}
if(order==0x4C)
{
    ioctl(p1->fd, 0x4C, 0);
    FD_ZERO(p1->p);
}
if(order==0x4E)
{
    ioctl(p1->fd, 0x4E, 0);
    FD_ZERO(p1->p);
}
if(order==0x52)
{
    ioctl(p1->fd, 0x52, 0);
    FD_ZERO(p1->p);
}
if(order==0x54)
{
    ioctl(p1->fd, 0x54, 0);
    FD_ZERO(p1->p);
}
if(order==0x56)
{
    ioctl(p1->fd, 0x56, 0);
    FD_ZERO(p1->p);
}
if(order==0x58)
{
    ioctl(p1->fd, 0x58, 0);
    FD_ZERO(p1->p);
}
if(order==0x5A)
{
    ioctl(p1->fd, 0x5A, 0);
    FD_ZERO(p1->p);
}
if(order==0x5C)
{
```

```
            ioctl(p1->fd, 0x5C, 0);
            FD_ZERO(p1->p);
        }
         if(order==0x5E)
        {
            ioctl(p1->fd, 0x5E, 0);
            FD_ZERO(p1->p);
        }
        if(order==0x41)
        {
            struct port1{
               bool con6;
               bool con7;
            };
            port1 port;
            char ch[sizeof(port1)];
            ioctl(p1->fd, 0x41, 0);
            read(p1->fd, ch, sizeof(port1));
            port=(port1)ch;
            p3->info1=port.con6;
            p3->info1=port.con7;
            memcpy(&recv_buffer_temp2, p1->p);
            send_data((void *)&recv_buffer_temp2);
            FD_ZERO(p1->p);
        }
    }
}
```

由于，命令的接收是随机的，所以命令处理函数的执行也是随机的，为此把上述处理函数放到一个线程中进行处理，如果有命令到来，则做出相应的处理。这两个线程函数如下：

```
res1=pthread_create(&a_thread, NULL, receive_data, (void*)&rcv_param
);
    if(res1!=0)
    {
        perror("thread creation failed");
        exit(1);
    }
    res2=pthread_create(&b_thread, NULL, is_received_to_send, (void*)
&command_buf);
    if(res1!=0)
    {
        perror("thread creation failed");
        exit(1);
```

```
    }
```

上述函数中 res1 和 res2 为两线程的句柄，实参 rcv_param 和 command_buf 分别为传送给线程的参数，它们都为结构休，如下：

```
struct param1   //向接收数据线程传输的函数参数
{
   int fd;
   char *rcv_buf;
   int rcv_wait;
};
param1 rcv_param;
struct command_buffer //向处理线程序函数传输的参数
{
   int fd;
   Control_Commandm *p;
};
command_buffer command_buf;
```

为了方便程序设计，存放命令用了两个变量，分别为：

```
Control_Command recv_buffer_temp1; //用于缓存接收的数据
Control_Command recv_buffer_temp2; //用于存储接收的数据
```

程序的主函数为如下，为了节省篇幅，相关函返回值的检测在此被省略了。

```
int main(int argc, char **argv)
{
   FD_ZERO(&recv_buffer_temp1);
   FD_ZERO(&recv_buffer_temp2);
   int fd;
   int res1;
   int res2;
   pthread_t a_thread;// 接收数据的线程
   pthread_t b_thread;// 处理数据的线程
   fd = open("/dev/arm2410_control", 0);
   rcv_param.fd=fd;
   rcv_param.rcv_buf=(char*)&recv_buffer_temp1;
   rcv_param.rcv_wait=1;
   send_param.buffer=(char*)&recv_buffer_temp2;
   send_param.fd=fd;
   command_buf.fd=fd;
   command_buf.p=(char*)&recv_buffer_temp1;
   res1=pthread_create(&a_thread,NULL,receive_data,(void*)
&rcv_param);
   res2=pthread_create(&b_thread,NULL,is_received_to_send,
(void*)&command_buf);
   while(1);
}
```

第 5 章　服务器端的控制软件原理与设计

前文已经提及到，要实现各个 IP 电话的通信，首先要将各个 IP 电话的电话号码与其 IP 地址建立映射关系，这一工作是由服务器来完成的。

IP 电话启动时，它首先向服务器发送出电话号码、当前的 IP 地址以及其它相关信息所构成的数据包，服务器将这个数据包存入数据库中。本文为了方便说明问题，对这个过程进行了简化，在程序中未采用数据库来处理这一过程。通过这一映射就形成了电话号码与 IP 地址动态对应表。拨打电话时，首先访问服务器，并向服务器发送 IP 号码查询请求，服务器根据请求，查找被叫方的 IP，如果能找到，则以数据包的形式发送给请求的 IP 电话，这样主叫方就可获得被叫方的 IP 地址，双方就可以进行通信。流程图如图 5-1。

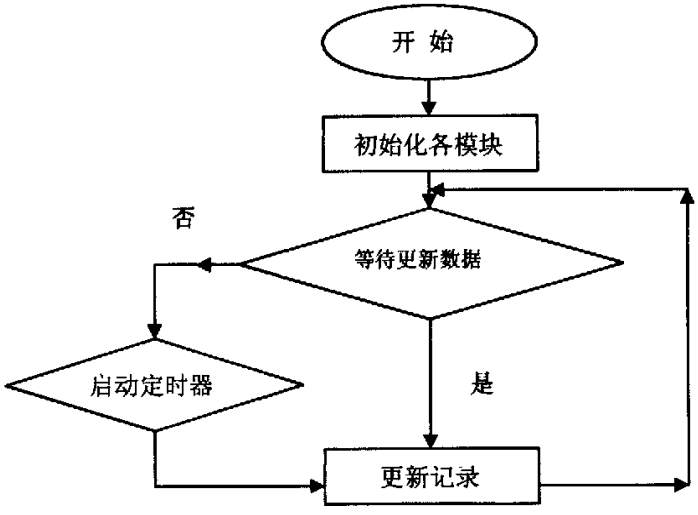

图 5-1 服务器程序流程图

为每一个 IP 电话分配一个不同的固定电话号码，该电话号码已经与 IP 电话的应用软件捆绑在一起，这样可以方便实现在不同地区将 IP 电话接入网络时，不用像传统的电话一样要更换电话号码。下文将介绍服务器的实现原理。

5.1 IP 电话与服务器通信应用层协议

要实现 IP 电话与智能家电有效地通信，必须定义一套双方交互信息的协议。为方便说明问题，把相关的协议进行了简化，文中只对主要部分做介绍。在实际应用中，可以根据需要进一步扩展。

在服务器端，用于存储各 IP 电话信息的是一个结构体数组，该结构体定义如下：

```
struct record                    //用于存储用户记录的结构
```

```
{
  std::string name;       //用户名
  char phone_num[12];     //电话号码
  BYTE  phone_IP[4];      //IP 地址
  int     num;            //记录在数据表中的位置
  bool    is_online;      //是否连接网络
  bool    is_leisure;     //是否占线
  float   charg_spend;    //通话费用
  float   charg_leave;    //剩余费用
};
```

结构体中各数据成员的作用如注释，其中 name、charg_spend、charg_leave 三个成员变量由服务器端设置，用户只可以查询。num 成员变量是为了方便查询而设定的，当 IP 电话向服务器发送数据更新请求时，服务器根据收到数据包的 num 值，将 IP 电话对应在数组中的相关信息进行更新，各一个 IP 电话发送的数据更新请求数据包的 num 值不同，具有唯一性。结构中其它各项由 IP 电话与服务器进行交互的信息来确定，它们被设置为定时更新，以确保数据能真实反映网络状态。

IP 电话以向服务器发送的数据以及服务器用于存储从 IP 电话接收的数据的变量，为如下结构体类型：

```
struct record_temp          //用于存储收到的用户记录的结构
{
  char phone_num[12];     //电话号码
  BYTE  phone_IP[4];      //IP 地址
  int     num;            //记录在数据表中的位置
  bool    is_leisure;     //是否占线
  char    sign;           //用于标记查询还是更换新 IP
};
```

其中前 4 项与 record 结构作用一样，sign 变量用于标识 IP 电话向服务器发送的数据包是用于请求查询被叫方 IP 电话号码还是用于更新服务器中相应 IP 电话对应的信息。

5.2 服务器的实现源码分析

服务器程序采用 VC 开发，为方便程序设计，采用面向对象思想进行程序设计。在服务器中，完成数据的更新以及向 IP 电话发送查询结果的功能由类 CReceive 来实现，它的定义如下：

```
class CReceive
{
public:
  static void send_info();
```

```
    CReceive();
    static DWORD WINAPI RecvProc();//出接收的线程调用
    virtual ~CReceive();
private:
    };
```

类中 send_info()函数的作用是根据 IP 电话发送的数据更新服务器记录或者向 IP 电话发送查询结果。由于服务器接收 IP 电话的查询和更新请求是随机的，为了避免函数阻塞，以便更好地提高程序运行效率，采用了多线程编程技术。由于接收数据的函数 static DWORD WINAPI RecvProc()出线程调用，它被定义为静态函数。

类的部分实现源代码如下：

```
record_temp rev_tem;
record record_IPphone[10];//用于存放 10 个用户记录
DWORD WINAPI CReceive::RecvProc()
{
    SOCKET sock=socket(AF_INET,SOCK_STREAM,0);
    SOCKADDR_IN addr;
    addr.sin_family=AF_INET;
    addr.sin_port=htons(6000);
    addr.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
    int retval;
    retval=bind(sock,(SOCKADDR*)&addr,sizeof(SOCKADDR));
    listen(sock,5);
    int len=sizeof(SOCKADDR);
    char *recvBuf=(char *)&rev_tem;
    int ret;
    while(TRUE)
    {
        SOCKET sockConn=accept(sock,(SOCKADDR*)&addr,&len);
        waitForSingleObject(hMutex,INFINITE);
        ret=recv(sockConn,recvBuf,sizeof(rev_tem),0);
        releaseMutex(hMutex);
        if(SOCKET_ERROR==ret)
        {
            break;
        }
        else
        {
            send_info();
        }
    }
    return 0;
}
```

```
void CReceive::send_info()
{
  if(rev_tem.sign=='a')//把 IP 电话上传的电话号码与 IP 对应关系存入
  //数组中
   {
strcpy(record_IPphone[rev_tem.num].phone_num,rev_tem.phone_num);
memcpy(record_IPphone[rev_tem.num].phone_IP,rev_tem.phone_IP,4);
record_IPphone[rev_tem.num].is_leisure=rev_tem.is_leisure;
   }
   if(rev_tem.sign=='b')//向 IP 电话发送查询结果
   {
      int i;
      for(i=0;i<10;i++)
      {
       if(strcmp(record_IPphone[i].phone_num,rev_tem.phone_num))
         {
strcpy(rev_tem.phone_num,record_IPphone[i].phone_num);
memcpy(rev_tem.phone_IP,record_IPphone[rev_tem.num].phone_IP,4);
            break;
         }
      }
      if(i==10)
      {
         rev_tem.phone_num[11]='\0';
         strcpy(rev_tem.phone_num,"00000000000");
      }
      struct in_addr inAddr;
      inAddr.S_un.S_un_b.s_b1= rev_tem.phone_IP[0];
      inAddr.S_un.S_un_b.s_b2= rev_tem.phone_IP[1];
      inAddr.S_un.S_un_b.s_b3= rev_tem.phone_IP[2];
      inAddr.S_un.S_un_b.s_b4= rev_tem.phone_IP[3];
      SOCKET sockClient=socket(AF_INET,SOCK_STREAM,0);
      SOCKADDR_IN addrTo;
      addrTo.sin_family=AF_INET;
      addrTo.sin_port=htons(6000);
      addrTo.sin_addr.S_un.S_addr=inet_addr(inet_ntoa(inAddr));
      connect(sockClient,(SOCKADDR*)&addrTo,sizeof(SOCKADDR));
      char *strSend=(char *)&rev_tem;
      send(sockClient,strSend,sizeof(rev_tem),0);
   }
}
```

程序中全局变量 rev_tem 具有结构体 record_temp 类型，用于存储接收到的 IP 电话请求命令数据，它在线程调用函数 RecvProc()和用于实现发送与更新的函数 send_info()共用，因此为了保证数据的唯一性，采用了线程同步技术。为了节

省篇幅，用全局结构体数组变量 record record_IPphone[10]来存放记录，它对各个 IP 电话的相关信息进行登记。

在服务器端可以对各 IP 电话实行管理，能够实现交费、查询、更新数据以及呼叫限制等功能。它的用户界面如图 5-2。

刷新功能，根据服务器中存放的对应 IP 电话的 IP 地址向 IP 电话发送数据，以更新服务器端数据，如果未收到回复数据包，则表明 IP 电话没有连接网络。

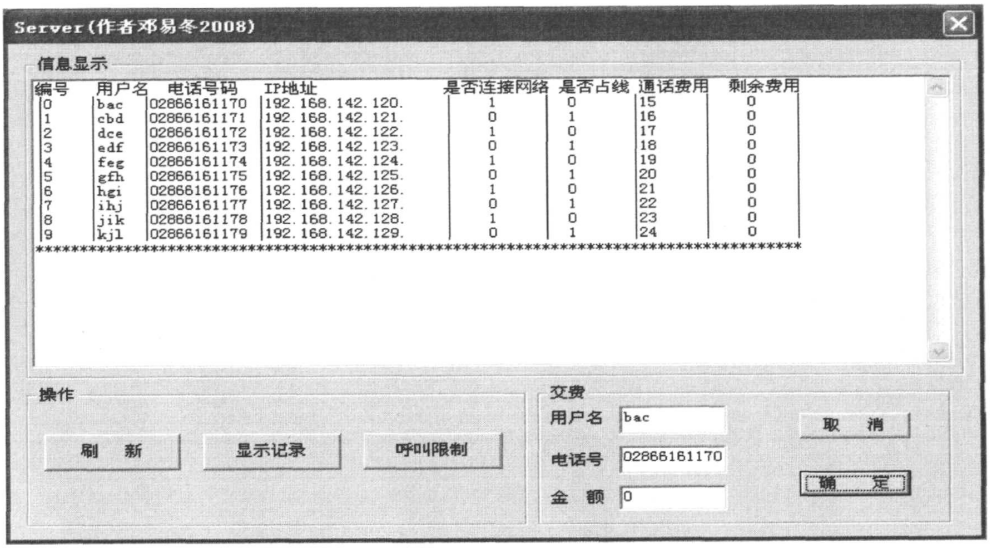

图 5-2 服务器运行界面

通过这种方式，可以实现服务器主动与 IP 电话进行的交互。呼叫限制用于限制某个电话的通话权利，当费用不足时可自动关闭对其的服务。服务器端对用户提供交费业务。

即此，系统的软件硬件开发介绍完毕，下面针对本系统的性能，进行简要介绍。

第 6 章　系统测试

由于该嵌入式系统运用于智能家电的远程控制，要求有较高的安全性和可靠性。因此必须对其关键部件和系统整体性能进行测试，以验证系统设计是否达到实际应用要求。在系统设计过程中，对相关软硬件进行了测试。由于本系统的软件设计比较复杂，涉及到语音编程、驱动编程和网络编程等诸多内容，并且是针对 WinCE、Linux 和 Windows 三种不同的操作系统编程，所以本章只简要介绍一些主要部分的测试。

6.1 软件测试原理

在系统设计中，软件设计占了大量的比重，它关系到系统的性能。因此，首先得对软件进行测试。所谓软件测试，就是使用人工和自动手段来运行或测试某个系统的过程，其目的在于检测它是否满足规定的需求或弄清预期结果与实际结果之间的差别。测试过程包括：模块测试、集成测试、系统测试[43]。

(1) 模块测试：结构化软件系统中，每个模块完成一个相对独立的子功能，此过程目的是确认模块作为单元是否能够正常运行。

(2) 集成测试：测试对象是模块间的接口，目的在于找出在模块接口及体系结构上的问题。

(3) 系统测试：将软件系统作为单一实体进行测试，以验证系统是否能够达到项目计划规定的要求。

6.2 测试方法

在程序设计过程中，各个软件模块均编译通过，并对各个软件模块进行了测试。本系统编程，在 IP 电话端涉及到语音模块、网络模块和串口通信模块等，编译各后模块间接口能正常工作。

对系统测试时，首先启动服务器，然后起动 IP 电话，再启动智能家电，各个模块均能正常运行。通过主叫 IP 电话拨打被叫方 IP，在服务器端观察到了主叫 IP 的连接请求。完成连接后，双方能进行语音通信。当同时启动多个 IP 电话时，服务器端能正常实现各个 IP 电话的调度。当启动智能家电控制功能时，通过向远程家电发送控制命令，在被控制的 IP 电话的通用 IO 口能正常检测到电平的改变，能正常控制以电扇为例的智能家电模形，符合预期要求。设计相关 IO 口的电平，IP 电话能够检测到，并将相关信息发送给远程查询电话。

本系统中对智能家电的遥操作是以电风扇为例进行说明的，下面简要介绍这一过程的软硬件测过程。

首先启动系统各功能模块，然后在 IP 电话端的家电类型栏中选择电扇作为控制对象。点击“开机”按钮，观察到电机转动，点击“关机”按钮，观察到电机停止转动，点击“正向动行”和“逆向动行”按钮，电机能实现正逆向运行。设置“定时”和“档位”按钮电机能实现定时和调速，如图 6-1。

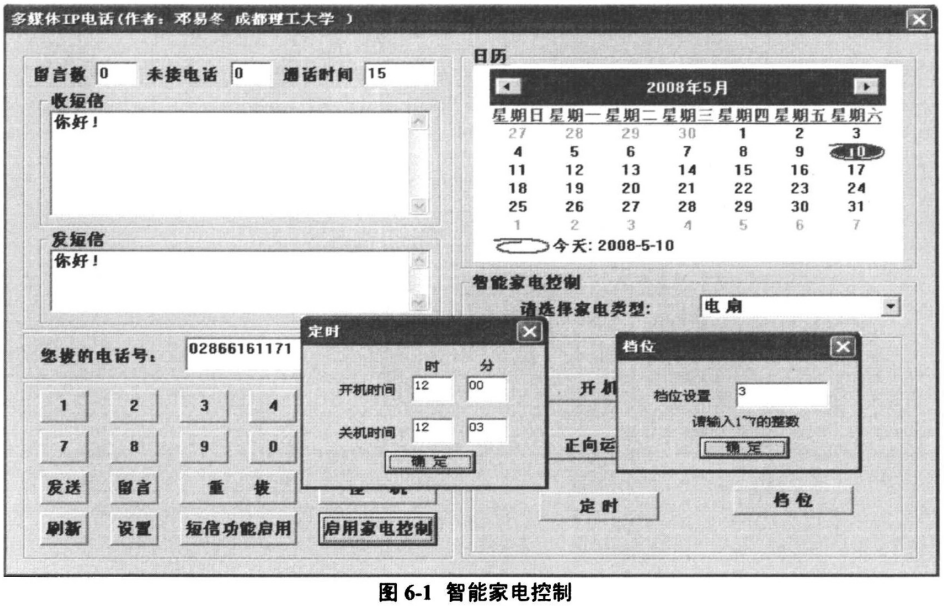

图 6-1 智能家电控制

6.3 测试结论

本系统语音通信在局域网和广域网进行了测试，语音通信质量良好。由于 NAT 穿越问题，本系统未实现局域网与广域网的通信。服务器能够完成向 IP 电话提供 IP 地址查询和数据更新的功能，能满足多路电话同时请求的要求。IP 电话与智能家电的串口通信效果良好，所发送的命令能及时收到。通过服务器可以屏避某 IP 电话的播号功能，能够完成简单的计费功能。

本系统测试性能良好，符合预期设计要求。由于系统软硬件设计相当复杂，肯定还有很多不足的地方，要作为商品投入市场，还有很多地方需要完善。

结 论

高性能的 ARM 芯片和嵌入式操作系统的完美结合将把原来基于 PC 机的众多技术引入到嵌入式领域，这为消费类电子技术提供了更广阔的空间和更完善的解决方案。目前，各国厂商在嵌入式应用领域投入了大量的科研力量，但智能家电控制领域尚无统一的标准，各公司的产品技术指标不同。

本文提出了一种有效的解决方案。系统设计以 IP 电话为控制中心，实现了语音、短信和智能家电遥操作等功能。通过 IP 电话可以对智能家电进行远程控制和从智能家电家获取其运行信息。作为智能家控制中心的 IP 电话可根据实际情况协同各智能家电的运行。采用串行口组成家电控制网络，避免了像无线连接技术的不稳定、延时长和距离受限等缺陷。这种智能家电组网方案具有实时性强、成本低、体积小、易于扩展和运行可靠等诸多优点，进行适当扩展，可作为智能家居开发应用参考方案。

由于嵌入式技术作为一种新兴的技术，参考资料较少，而且本系统涉及的知识面广，是多种技术的综合应用，加上系统开发任务比较多，时间有限等因素的制约，本系统还有很多不足的地方，恳请各位老师、专家指正。

致 谢

在论文完成之际，首先要衷心地感谢我的父母，是他们从小培养了我坚强的意志，让我能够在读研究生的三年时间里，不断进取。

感谢贾雨老师三年来的耐心帮助。

感谢舍友谢文惠，学长王福刚、赵国涛，一起在天冀方通公司工作的各位同事。

参考文献

[1] 智能家居组网技术[EB/OL].(2007) .http://www.smarthomecn.com/.

[2] 智能小区技术[EB/OL].(2007-12) .http://www.ehomecn.com/Index.html.

[3] 孙天泽、袁文菊、张海峰.嵌入式设计及驱动开发指南一基于ARM9处理器(第2版)[M].北京:电子工业出版社，2006.

[4] 周立功等.magic2410教学实验开发平台实验指导[Z].广东:致远电子限公司，2006.

[5] 周立功，陈明计，陈渝.ARM嵌入式LINUX系统构建与驱动开发范例[M].北京北京航空航天大学出版社，2006.

[6] 周立功.ARM嵌入式系统软件开发实例(二)[M].北京:北京航空航天大学出版社，2006.

[7] 何宗键. Windows CE嵌入式系统 [M].北京:北京航空航天大学出版社，2007.

[8] 薛大龙. Windows CE嵌入式系统开发从基础到实践[M]. 北京:电子工业出版社.2008.

[9] 张冬泉. Windows CE 实用开发技术 [M]. 北京: 电子工业出版社.2006.

[10] 曹衍龙，刘海英.Visual C++网络通信编程实用案例精选(第2版) [M]. 北京:人民邮电出版社.2006.

[11] 孙鑫，余安萍.VC++深入详解[M]. 北京:电子工业出版社. 2006.

[12] 求是科技.Visual C++串口通信技术与工程实践(第二版)[M].北京:人民邮电出版社.2004.

[13] 北京革新科技有限公司.嵌入式教学实验（Linux基础篇）[Z]. 北京:北京革新科技有限公司.2007.

[14] Neil Matthew Richard Stones. Linux程序设计(第3版) [M]. 北京: 人民邮电出版社. 2007.

[15] 赵星寒,刘涛.从51到ARM--32位嵌入式系统入门[M]. 北京: 北京航空航天大学出版社. 2006.

[16] 刘淼. 嵌入式系统接口设计与Linux驱动程序开发[M].北京:北京航空航天大学出版社，2007.

[17] 杜春蕾.ARM体系结构与编程[M].北京:清华大学出版社，2003.

[18] 马忠梅，马广云，徐英慧等.ARM嵌入式处理器结构与应用基础.北京:北京航空航天大学出版社，2002.

[19] ARM2410芯片说明书 [EB/OL].http://www.samsung.com.

[20] 华清远见嵌入式培训中心 李俊. 嵌入式Linux设备驱动开发详解 [M]. 北京:人民邮电出版社.2008.

[21] 华清远见嵌入式培训中心 宋宝华. Linux设备驱动开发详解 [M]. 北京:人民邮电出版社.2008.

[22] 杜华. Linux编程技术详解 [M]. 北京:人民邮电出版社.2007.

[23] 陈艳华，侯安华，刘盼盼. 基于ARM的嵌入式系统开发与实例 [M]. 北京:人民邮电出版社.2008.

[24] 周毓林，刘波. Windows CE.net 内核定制及应用开发 [M]. 北京: 电子工业出版社.2005.

[25] 王进德. 嵌入式Linux程序设计 [M]. 北京: 中国电力出版社.2007.

[26] 冯国进. 嵌入式Linux驱动程序设计从入门到精通[M]. 北京:清华大学出版社.2008.

[27] 冯吴. Linux操作系统教程[M]. 北京: 清华大学出版社.2008.

[28] 李亚锋. ARM嵌入式Linux系统开发从入门到精通[M]. 北京: 清华大学出版社.2007.

[29] 沈立强. 计算机网络基本原理与Internet实践[M]. 北京:清华大学出版社.2008.
[30] 陈明. 计算机网络设计教程（第二版）[M]. 北京: 清华大学出版社.2007.
[31] 杨云江，陈笑筑、罗淑英等. 计算机网络基础（第2版）[M]. 北京: 清华大学出版社.2007.
[32] 熊茂华，杨震伦. ARM9嵌入式系统设计与开发应用[M]. 北京: 清华大学出版社.2008.
[33] 孙鹤旭. 嵌入式控制系统[M]. 北京: 清华大学出版社.2007.
[34] 李善平. Linux与嵌入式系统 [M]. 北京: 清华大学出版社.2007.
[35] Stanley B. Lippman Barbara E. Moo Josée LaJoie. C++ Primer中文版(第4版）[M]. 北京:人民邮电出版社.2006.
[36] 智能家区布线方法[EB/OL].(2008) .http://www.jnzn.net/.
[37] 智能家电技术[EB/OL].(2008-05) .http://www.fsldbp.com/Pro/default.asp.
[38] 孙静.基于以太网智能家居控制器的研究与实现:(硕士学位论文).西安:西安科技大学，2006.
[39] Wayne Wolf.嵌入式计算系统设计原理.北京:机械工业出版社[M].2002.
[40] Douglas E.Comer.用TCP/IP进行网际互联第一卷(第五版)[M].北京:电子工业出版社[M].2007.
[41] Douglas E.Comer.用TCP/IP进行网际互联第二卷(第三版)[M].北京:电子工业出版社[M].2004.
[42] Xi Ning, Tarn T J.Action Synchronization and Control of Internet.
[43] 古乐，史九林.软件测试技术概论[M].北京:清华大学出版社，2004.