摘要

实验教学是理工科院校一门十分重要的课程。随着科学技术的发展和教育理念的更新，如何使实验教学更好地将理论和实践相结合、创造新的教学模式、建设一流的实践基地，全面提升高校实验教学的综合水平和可持续发展能力，是值得研究和急待解决的问题。本文利用 ARM7TDMI 系列微处理器功耗低、性价比高、支持多种接口、支持多种嵌入式操作系统等优点，搭建了以三星 S3C44B0X 微处理器为核心的嵌入式开发平台。并选用具有 RS232 接口的 Siemens MC35iT GPRS 终端为 GPRS 网络接入设备。该 GPRS 终端性能稳定出色，使用 AT 指令集控制，配合嵌入式系统开发板使用，构成了嵌入式无线通信实验开发平台。完成了整个实验平台硬件的配置和调试。使该套实验系统具有性价比高、易维护、易升级扩充、承担试验项目范围广等优点。不但符合现阶段开放实验对硬件平台的严格的要求，还能跟上电子技术发展的步伐，在未来开放实验项目不断增加，对硬件要求越来越高的情况下继续发挥作用。基于该实验硬件平台，本文还面对我校高年级本科学生及研究生设计了 3 个基于嵌入式系统的开放实验项目：Boot Loader 的设计，uClinux 内核及 ROMFS 根文件系统映象文件的制作，短消息及语音通信应用程序设计。在给出实验目的、功能要求及相关理论知识的同时提出了从实验项目总体设计到具体实施方法、结论的一整套实验设计方案，作为开放实验项目的参考实现方案。上述实验项目以市场需求为导向，以信息集成为基础，把创新开发、计划拟订和流程设计等各个环节集成到实验教学中，提高了实验教学的综合集成水平。这 3 个实验项目在设计上单元与系统相结合，循序渐进，层次分明。其开放的实验形式，能激发实验参与者的热情，留出了发挥和创新的空间，在提高其技术能力的同时培养项目的管理和组织能力。在集成产品与过程开发（当前国内外高科技行业普遍关注的全新的研发管理模式）下，实验教学吸收其中面向市场、团队运作等理念，对于拓宽视野具有积极意义。

关键词：开放实验 嵌入式系统 ARM GPRS uClinux 短消息

Abstract

The experiment object is a very impotent course in Technical University. With the developing of the micro electric and the education theory, how to teach both in knowledge and in practice, how to create a new model of experiment and how to build a top experiment base become a very urgent problem and need to be solved quickly. ARM7TDMI microprocessor has the advantages of low-power, cost-effictive and high performance, supporting many kinds of interface and operation system .According to these outstanding advantages, this paper uses the Samsung S3C44B0X microprocessor as the core to build an embedded experiment plat form. At the same time we chose siemens MC35IT with RS232 interface as the wireless network accessing device to work with the experiment developing board. And these two devices form the whole embedded wireless communication experiment system. We have finished confining and debugging the hardware to make this system work. This embedded experiment system has these advantages: cost-effective and high performance, easy to maintain and upgrade, easy to explore new experiment area. Meanwhile this experiment system not only fit the demand of nowadays experiment well but also can go with the developing of electronic science and work well. Base the hardware plat form of this experiment system, we design 3 open experiment projects to face the high grand batchers and masters. They are: Boot Loader design, uClinux kernel image and ROMFS file system image making, SMS and audio communication program design. We list the aim of experiment, the support theory and the function demand of the experiment. We also design the project from top to details as the assistant document. These projects take the market as the guide and base the information integration to put creative developing, plan making and flow design together into one experiment. And effectively improve the level of experiment teaching. Also these open means of experiment, stimulate the participator's passion and give them the space of creation. During the experiment not only the participator's technical ability has been improved but also their ability of project management and organization. Using the theory of market and teamwork which comes from product integration and developing in experiment design has very active mean in exploring our sight of view.

Key word: open experiment embedded system ARM uClinux SMS

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，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：贺晓维　　日期：2005 年 1 月 10 日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、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。本人授权电子科技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应遵守此规定）

签名：贺晓维　　导师签名：[illegible]

日期：2005 年 1 月 10 日

第一章 引言

1.1 课题的来源及意义

随着科学技术的飞速发展，计算机及其相关技术得到进一步的发展，计算将不再局限于传统的 PC 和服务器环境，网络计算和移动计算将很快成为人们日常生活的一部分，并逐渐呈现出普及计算（Pervasive Computing）模式。

作为普及计算的支撑技术，嵌入式实时系统正逐步应用到越来越多的领域，包括智能过程控制、航天航空、交通、飞行控制、通信、多媒体、办公自动化、实时模拟、虚拟现实、医疗电子、军事电子、信息家电等领域。

而在大学校园中，虽然开设了嵌入式系统课程但嵌入式系统结合无线通信的学生实验项目却很少。基于这种考虑，希望能开发出一个学生实验项目既涉及到嵌入式系统及嵌入式操作系统又包括无线通信技术，三者相互接合，也迎合了目前电子技术发展的趋势。让学生在实验中对这三个领域的知识能够有较深入的了解并且相互贯通，实践动手能力得到提高。

1.2 课题主要内容

本课题基于 ARM7 嵌入式微处理器，利用其性价比高，功能丰富，接口完善，可扩展性强等优点将最新的移动通信技术与嵌入式操作系统融合在一起。提出一个崭新的嵌入式无线通信开放实验方案。完成了实验硬件系统的选型，嵌入式无线通信实验平台的搭建和调试。并在此基础上设计了具体的开放实验内容：Boot Loader 的设计，uClinux 根文件系统的制作，短信及语音通信应用程序设计。同时给出实验项目的理论依据和目的，以及从实验的总体设计方案、功能要求到具体实施方法、实验结论等一系列完整的内容，作为开放实验项目实施方案的参考。

1.3 课题研究思路及创新

嵌入式系统及无线通信技术是当今电子科技发展的两大潮流。如何将这两者结合起来，就为我们的实验项目设计提供了一个总体思路。同时也是它区别与以往单一方向实验的第一个创新之处。同时虽然学校也开设了嵌入式系统设计课程但相关的配合实验很少，且有以下两点不足[1]：

1、开放式实验教学缺乏有效的运行机制：主要服务对象是大学 2-4 年级本科学生，大多数实验属于验证型，无论是形式还是内容离真正意义上的

“开放”均相距甚远。

2、能力培养的盲区：关于能力培养，过去一直强调的是“宽口径”、“综合能力”，但执行时主要定义在技术层面，而对于在国内外高科技行业所需要的研发模式、工程、管理和组织等知识和能力培养，几乎是空白。

随着高校教学内容的从新整合，毕业设计、课程设计和研究生创新能力培养等需求的提出，需要实验教学不仅是内容，而且在过程上要有新的思路，以增强开放式实验教学的互动性。本课题所提出的实验方案正好能够弥补上述两点不足，同时也使其具备了另外两个创新之处：

1. 对实验参与者的知识掌握水平要求高。
2. 与以往验证型实验根本不同，是真正开放型实验。提供所需的硬件平台及相关的理论知识。只提出实验要求，而不给出具体的实施方案。不但从技术层面锻炼参与者还注重对参与者项目管理和组织能力的培养。

第二章 实验环境分析

2.1 嵌入式无线通信实验系统框图

通常的嵌入式系统主要是由宿主机（PC），及目标板构成。由于本系统需要GPRS 实现对移动网络的接入，故主要由三部分组成，其框图如下：

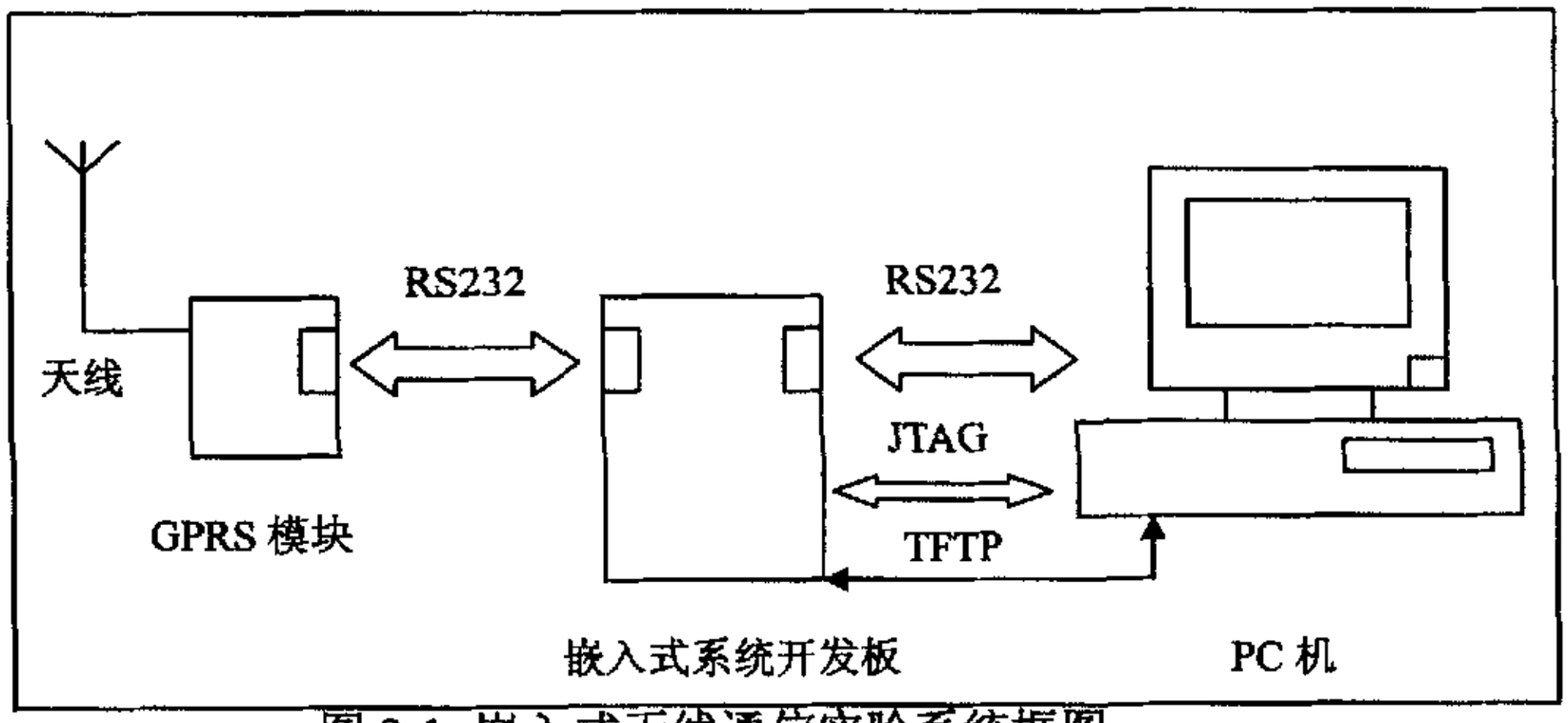


图 2-1 嵌入式无线通信实验系统框图

PC 机：作为整个实验系统的控制端。接收命令行输入，输出实验系统所打印的信息：如实验系统的硬件信息，操作成功与否的信息，短信内容等。

嵌入式系统开发板：在 uClinux 操作系统下运行应用程序。是程序执行的主体，根据输入的命令行完成相关操作。如：与 GPRS 终端及 PC 主机进行数据交换和处理。

GPRS 终端： 主机输入命令的最终执行者。通过外接天线完成移动网接入、短信收发、语音通话等功能。使用 RS232 接口与实验开发板相连，并以此完成数据交换。

2.2 嵌入式无线通信实验系统开发平台的建立

2.2.1 实验开发板的选择

选择开发板实际上就是选择使用什么样的嵌入式微处理器，因为微处理器的性能好坏直接影响到整个开发板的功能和稳定性。目前市场上流行的微处理器虽然种类繁多，功能不一。但是它们大多数都采用了 ARM 内核。ARM（Advanced RISC Machines），即可以认为是一个公司的名字，也可以认为是对一类微处理器的统称，并且还可以认为是一种技术的名字。

传统的 CISC（Complex Instruction Set Computer，复杂指令集计算机）结构有其固有的缺点，即随着计算机技术的发展而不断引入新的复杂的指令集，为支持这些新增的指令，计算机的体系结构会越来越复杂，然而，在 CISC 指令集的各种指令中，其使用频率却相差悬殊，大约有 20%的指令会被反复使用，占整个程序代码的 80%。而其余 80%的指令却不经常使用，在程序设计中只占 20%，显然这种结构是不合理的。

基于以上的不合理性，1979 年美国加州大学伯克利分校提出了 RISC（Reduced Instruction Set Computer，精简指令集计算机）的概念，RISC 并非只是简单地减少指令，而是把着眼点放在了如何使计算机的结构更加简单合理地提高运算速度上。RISC 结构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，指令格式和寻址方式种类减少；以控制逻辑为主，不用或少用微码控制等措施来到达上述目的。

目前，RISC 体系结构还没有严格定义，一般认为，RISC 体系结构应具有以下特点[2]：

★ 采用固定长度的指令格式，指令归整、简单、基本寻址方式有 2—3 种；

★ 使用单指令周期，便于流水线操作执行；

★ 大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/存储指令可以访问存储器，以提高指令的执行效率；

除此以外，ARM 体系结构还采用了一些特别的技术，在保证高性能的前提下尽量缩小芯片的面积，并降低功耗：

★ 所有的指令都可以根据前面的执行结果决定是否被执行，从而提高指令的执行效率；

★ 可用加载/存储指令批量传输数据，以提高数据的传输效率；

★ 可在一条数据处理指令中间同时完成逻辑处理和移位处理；

★ 在循环处理中使用地址的自动增减来提高运行的效率；

当然，与 CISC 架构相比较，虽然 RISC 架构有以上优点，但决不能认为 RISC 架构就可以取代 CISC 架构，事实上 RISC 和 CISC 各有优势，而且界限并不那么明显。现代的 CPU 往往采用 CISC 的外围，内部加入了 RISC 的特性，如超长指令集 CPU 就是融合了 RISC 和 CISC 的优势，成为未来的 CPU 发展方向之一。

2.2.1.1 ARM 微处理器的特点

采用 RISC 架构的 ARM 微处理器具有如下特点[2]：

★ 体积小、低功耗、低成本、高性能；

★ 支持 Thumb（16 位）/ARM（32 位）双指令集，能很好地兼容 8 位/16 位

器件；

★ 大多数数据操作都在寄存器中完成；

★ 大量使用寄存器，指令执行速度更快；

★ 寻址方式灵活简单，执行效率高；

★ 指令长度固定；

ARM7 系列微处理器为低功耗的 32 位 RISC 处理器，具有如下特点：

★ 极低的功耗，适合对功耗要求较高的应用，如便携式产品；

★ 提供 0.9MIPS/MHz 的三级流水线结构；

★ 代码密度高并兼容 16 位的 Thumb 指令集；

★ 对操作系统的支持广泛，包括 WindowsCE、Linux、Palm OS 等；

★ 指令系统与 ARM9 系列、ARM9E 系列和 ARM10 系列兼容，便于产品的升级换代。

★ 主频最高可达 130MIPS，高速的运算处理能力能胜任绝大多数的复杂应用；

★ 具有嵌入式 ICE—RT 逻辑，调试开发方便。

ARM7 系列的微处理器包括几中核：ARM7TDMI、ARM7TDMI—S、ARM720T、ARM7EJ。其中，ARM7TDMI 是目前使用最广泛的 32 位嵌入式 RISC 处理器、属低端 ARM 处理器核。TDMI 的基本含义为：

T：支持 16 位压缩指令集；

D：支持片上 Debug；

M：内嵌硬件乘法器（Multiplier）；

I：嵌入式 ICE，支持片上断点和调试点。

2.2.1.2 ARM 微处理器的应用领域

到目前为止，ARM 微处理器及技术的应用几乎已经深入到各个领域。

★ 工业控制领域：

作为 32 位的 RISC 架构，基于 ARM 核的微控制器芯片不但占据了高端微控制器市场的大部分份额，同时也逐渐向低端微控制器应用领域扩展，ARM 微控制器的低功耗、高性价比，向传统的 8 位/16 位微控制器提出了挑战。

★ 无线通信领域：

目前已有 85%以上的无线通信设备采用了 ARM 技术，ARM 以其高性能和低成本的特点，在该领域的地位日益巩固。

★ 网络应用：

随着宽带技术的推广，采用 ARM 技术的 ADSL 芯片正逐步获得竞争优势。

此外，ARM 在语音及视频处理上进行了优化，并获得广泛支持，也对 DSP 的应用领域提出了挑战。

★ 消费类电子产品：

ARM 技术在目前流行的数字音频播放器，数字机定盒和游戏机中得到广泛采用。

★ 成像和安全产品：

现在流行的数码相机和打印机中绝大部分采用 ARM 技术。手机中的 32 位 SIM 智能卡也采用了 ARM 技术。

2.2.1.3 实验开发板微处理器选择及其主要指标

鉴于 RISC 架构和 ARM 微处理器的上述特点，在工业控制、通信、网络、金融、军事等各行业都已获得了广泛的应用，随着国内外嵌入式应用领域的进一步发展，ARM 微处理器必然会获得更加广泛的重视和应用。作为创新实验项目的开发，更应该紧扣业界的发展趋势，让学生在实验中获得最新最实用的知识。所以对开发板的核心—嵌入式微处理器的选择，我们锁定在 ARM 系列的微处理器上。ARM7 系列嵌入式微处理器是目前应用得最广的 32 位 RISC 处理器，其结构较 ARM9、ARM10 简单，配合外围电路及各种类型的接口可以完成各种复杂的任务，适应各种开发对接口的要求，并且其适用范围符合该嵌入式试验系统的要求。基于上述考虑，我们选择了基于 S3C44B0（ARM7TDMI）微处理器的开发板。其主要的配置如下：

★中央处理器：

S3C44B0X (Samsung)，ARM7TDMI

★外部存储器：

2M Bytes Nor Flash （SST39VF160）

8M Bytes SDRAM （K4S641632H）

★扩展网口：

10M 网口，RTL8019AS

★LCD 接口：

支持单色、4 级灰度、16 级灰度、256 色STN 液晶屏，最大支持640x480/256 色 STN 屏

★USB Device 接口：

USB1.1 规范,PDIUSBD12

★USB Host 接口：

USB1.1 规范，SL811HST

★串口：

两个标准三线RS232 接口，其中COM2 支持硬件流控制

★时钟源：

内部实时时钟（备有掉电电池）

★IDE 接口

★音频输出

CS4334＋TDA7050

★Smart Media Card 接口

★四个小按键，四个LED

★一个蜂鸣器

★一个PS/2 接口

板上硬件资源分配入下：

★系统片选及地址空间

nGCS0【0x0000_0000】：FLASH（SST39VF160）

nGCS1【0x0200_0000】：未使用

nGCS2【0x0400_0000】：IDE/ATA

nGCS3【0x0600_0000】：RTL8019AS

nGCS4【0x0800_0000】：PDIUSBD12（ USB DEVICE ）

nGCS5【0x0A00_0000】：SL811HST（USB HOST 或者是USB DEVICE）

nGCS6【0x0C00_0000】：SDRAM （K4S641632H）

nGCS7【0x1000_0000】：SmartMedia Card

★中断分配

INT0：USB D12

INT1：RTL8019

INT2：IDE/ATA

INT3：TouchPad(LCD)

INT4：USB SL811(与KEY1 共用)

INT5：KEY2

INT6：PS2 口（KEY3 共用）

INT7：PS2 口（KEY4 共用）

★系统板设定：

BANK0 总线宽度为16 bit；

小端（Little Endian）模式；

开发板结构框图如图2-2。

S3C44BOX功能配置如下[3]：

- 2.5V ARM7TDMI CPU内核，带8KB缓存（SAMBA2总线结构，最高可达66MHz）；
- 外部存储控制器（FP/EDO/SDRAM控制器，片选逻辑）；
- LCD控制器（DSTN最高达256色），1个LCD专用DMA通道；
- 2个通用DMA通道，2个外围DMA通道连接外部请求引脚；
- 2个UART通道；
- 1个IIS总线控制器通道；
- 5个PMW定时器通道，1个内部定时器通道；
- 看门狗定时器；
- 8个10bit ADC通道；
- 电源管理；
- 片上PLL时钟发生器；

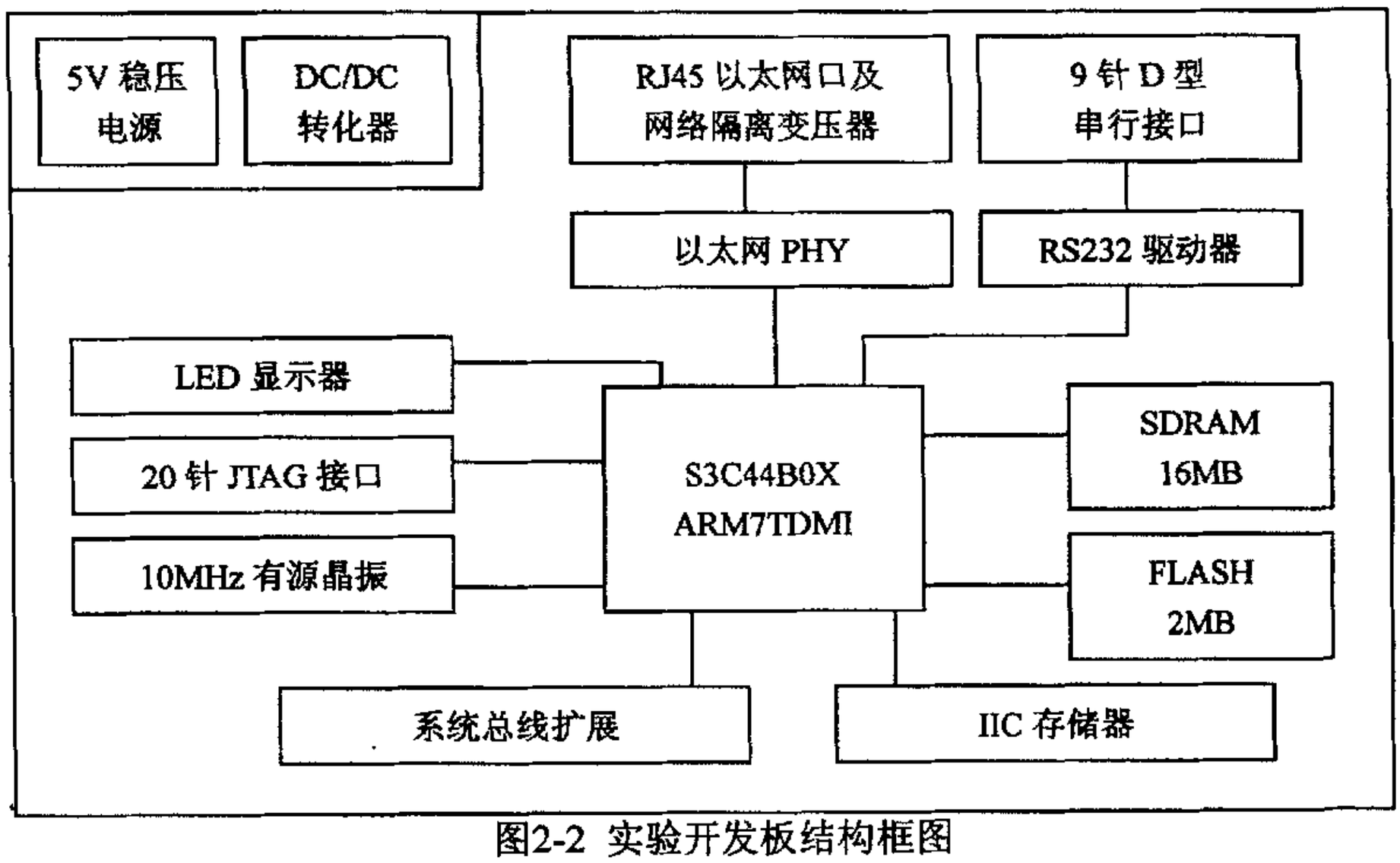


图2-2 实验开发板结构框图

其中的核心—微处理器结构框图如图2-3所示。

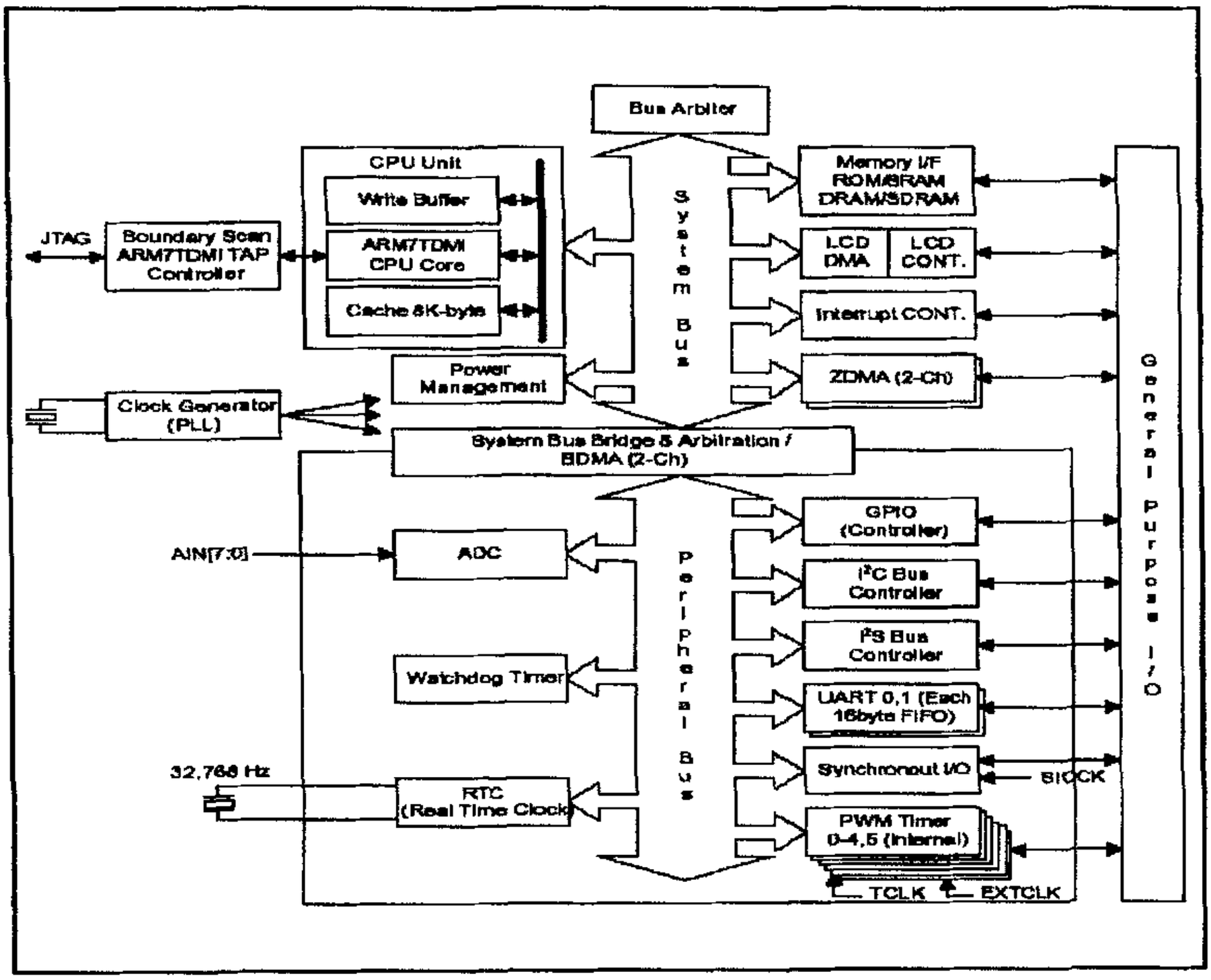


图2-3 S3C44B0X微处理器结构框图

2.2.1.4 ARM7 微处理器的指令结构

ARM7 微处理器支持两种指令集：ARM 指令集和 Thumb 指令集。其中，ARM 指令为 32 位，Thumb 指令位 16 位。Thumb 指令集为 ARM 指令集的功能子集，与等价的 ARM 代码相比，可节省 30%—40%以上的存储空间，同时具备 32 位代码的所有优点。

2.2.1.5 ARM 体系结构的存储器格式

ARM 体系结构将存储器看作是从零地址开始的字节的线性组合。从零字节到三字节放置第一个存储的字数据，从第四个字节到第七个字节放置第二个存储的字数据，依次排列。作为 32 位的微处理器，ARM 体系结构所支持的最大寻址空间为 4GB（2^{32}字节）。

ARM 体系结构可以用两种方法存储字节数据：大端格式、小端格式。

大端格式：

在这种格式中，字数据的高字节存储在低地址中，而字数据的低字节则存放在高地址中，如下图 2-4 所示：

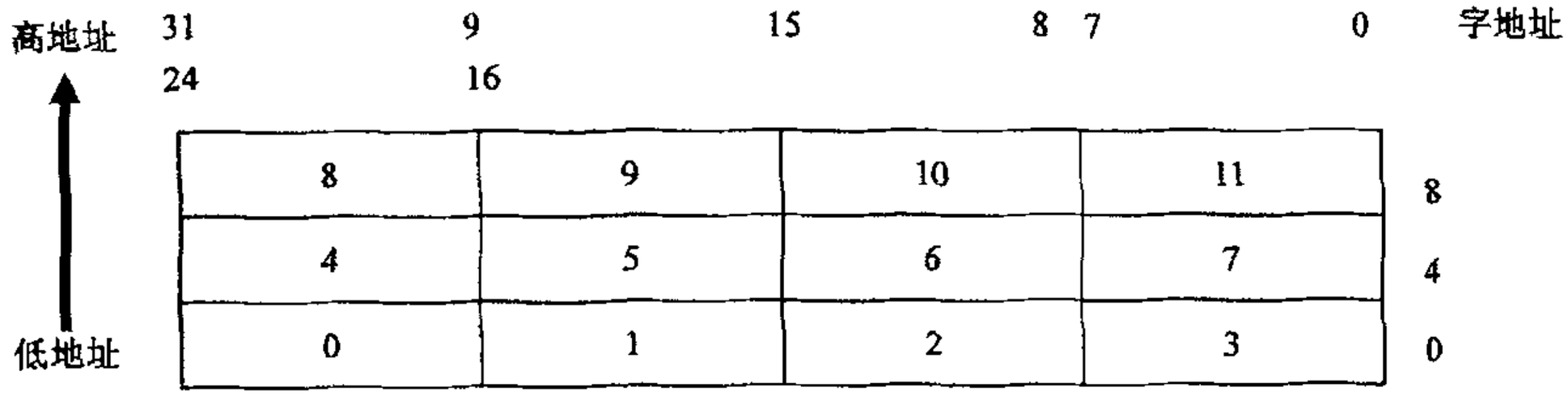


图 2-4 大端格式

小端格式：

与大端存储格式不同，在小端存储格式中，低地址中存放的是字数据的低字节，高地址存放的是字数据的高字节，如下图 2-5 所示：

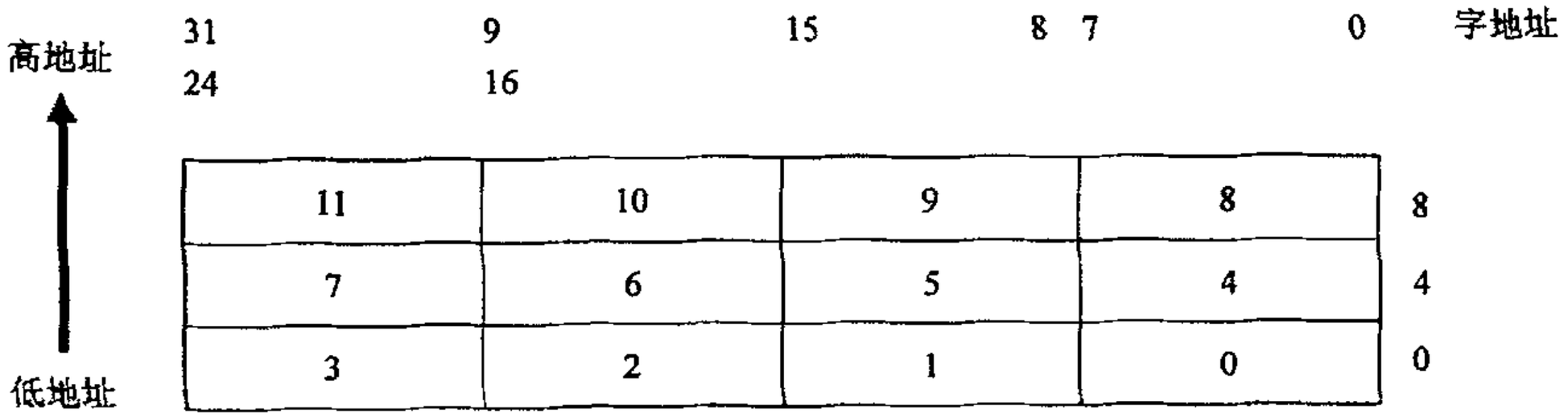


图 2-5 小端格式

2.2.2 嵌入式操作系统的选择

对嵌入式系统而言，应用程序可以直接在芯片上运行而不需要操作系统。但是随着嵌入式系统应用领域的不断拓展，系统往往需要同时处理多个事件。单一的直接运行在芯片上的应用程序越来越不能满足应用对实时性，多任务性等的要求。为了能更好的调度多任务，更好的利用系统资源，系统函数及专用函数库接口，方便快捷地进行嵌入式软件的开发，就必须选择嵌入式操作系统（EOS，Embedded Operation System)。嵌入式操作系统的引入大大的提高了系统开发的效率，减小了工作量，提高了嵌入式软件的可移植性和稳定性。

一个优秀的嵌入式操作系统除了具备一般操作系统的基本功能，如任务调度，中断处理，同步与互斥机制，文件系统外，还需要具备以下特点[2][4]：

★ 良好的可移植性。

嵌入式硬件平台呈多样化发展，芯片更新速度快。嵌入式操作系统必须要能对硬件平台有更好的适应性。

★ 装载与卸载。

嵌入式系统的专用性，要求嵌入式操作系统必须要能根据需要装载和卸载。即可裁减性。

★ 小巧。

嵌入式系统所能提供的硬件资源有限，所以嵌入式操作系统必须小巧，尽可能的占用更少的硬件资源，以满足硬件的限制。

★ 高可靠性和稳定性。

★ 提供强大的网络功能，支持 TCP/IP 及其他网络协议。

★ 友好的人机交互界面。

Linux、WindowsCE、Vxwork、UcOS等是目前最流行的嵌入式操作系统。WindowsCE是精简的Windows95版本，从技术角度讲，它并不是优秀的嵌入式操作系统。因其是非开放性的操作系统，所以缺少个性，第三方很难实现定制。嵌入式系统追求高效、节能，WindowsCE则比较笨拙，占用过大的内存，应用程序庞大。Linux是源码开放，可裁减定制的操作系统平台，是发展未来嵌入式设备的绝佳选择。并且Linux更小、更稳定，其源代码可以从网上免费下载得到，成本更低，非常适合用于嵌入式系统的开发。但是S3C44B0为了降低硬件成本及运行功耗，在设计中取消了内存管理单元MMU（Memory Management Unit）功能模块，使得一般的嵌入式Linux操作系统不能运行。

2.2.2.1 uClinux 的特点

uClinux 念作“you see Linux”，这个英文单词中，u 来自希腊文，表示 Micro，小的意思，C表示Control，控制的意思，所以uClinux就是Micro-Control-Linux，字面上的理解就是"针对微控制领域而设计的 Linux 系统"。

uClinux从Linux2.0/2.4内核派生而来，不但沿袭了主流Linux的绝大部分特性，它专门针对没有MMU的微处理器，并且为嵌入式系统做了许多小型化的工作，适用于没有虚拟内存或MMU的微处理器。并且uClinux为了支持没有MMU的处理器而对标准的Linux作出了修正，它保留了操作系统的所有特性，为硬件平台更好地运行各种程序提供了保证。在GNU通用公共许可证（GNU CPL）的保证下，运行uClinux操作系统的用户可以使用几乎所有的Linux API函数，不会因为没有MMU而受到影响。虽然它的体积很小，但uClinux仍然保留了Linux的大多数的优点[2][4]：

★ 稳定、良好的移植性；

★ 优秀的网络功能；

★ 对各种文件系统完备的支持；

★ 以及标准丰富的 API；

它的主要特征如下：

★ 通用 Linux API；

★ 内核体积小于 512KB，内核加上文件系统小于 900KB；

★ 完整体积小于 512KB，内核加上文件系统小于 900KB；

★ 支持其它大量网络协议；

★ 支持各种文件系统，包括 NFS、ext2、romfs and JFFS、MS-DOS 和 FAT16；

★ 支持各种典型的处理器构架，包括 ARM、PowerPC、X86 等；

2.2.2.2 uClinux 的基本结构

鉴于 uClinux 上述明显优与其它嵌入式操作系统的优点，并且考虑到其能为后续网络应用相关的嵌入式实验的开发提供的极大便利。所以本课题选择 uClinux 作为嵌入式操作系统。uClinux 的基本架构如图 2-6：

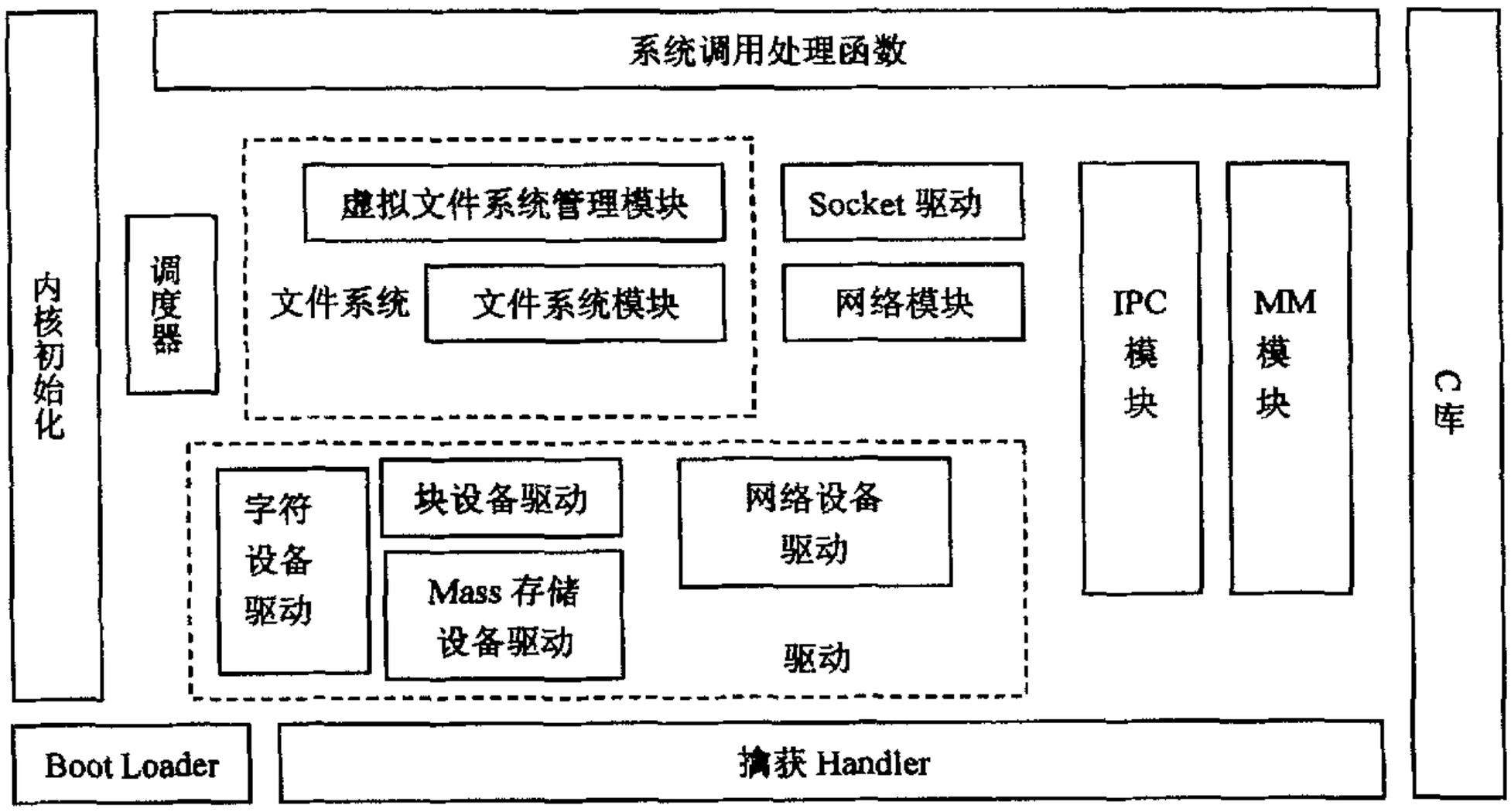


图2-6 uClinux基本架构

◆ Boot Loader：负责Linux内核的启动，它初始化系统资源，包括SDRAM。这部分代码用于建立Linux内核运行环境。没有Boot Loader uClinux就无法启动，对Boot Loader的设计是我们整个嵌入式无线通信实验的基础，也是一个重要的实验项目。

◆ 内核初始化：Linux内核的入口点是start kernel()函数。它初始化内核的其它部分，包括捕获、IRQ通道、调度、设备驱动、标定延迟循环，最重要的是能够fork“init”进程，以启动整个多任务环境。

◆ 系统调用函数 / 捕获函数：在执行完“init”程序后，内核对程序流不再有

直接地控制权。此后，它的作用仅仅是处理异步事件(例如硬件中断)和为系统调用提供进程。

- 设备驱动：设备驱动占据了Linux内核很大部分。同其它操作系统一样，设备驱动为它们所控制的硬件设备和操作系统提供接口。
- 文件系统：Linux最重要的特性之一就是对多种文件系统的支持。这种特性使得Linux很容易地同其它操作系统共存。文件系统的概念使得用户能够查看存储设备上的文件和路径而无须考虑实际物理设备的文件系统类型[2]。

2.2.3 无线网络接入设备选择

选择接入设备首先必须明确的一点是：接入何种网络。目前的无线网络有GSM、GPRS、CDMA2000 3种，GSM属于2G网络，会被后两者迅速替代，所以不考虑接入到GSM网。GPRS、CDMA2000采用的是2.5G网络技术，是未来一段时间内中国国内将会普遍用到的。虽然说 CDMA2000 采用的码分多址技术在理论上比 GPRS 更为先进，但是在实际情况下由于网络条件等一些外部因素的影响，在国内其表现并不如GPRS理想。所以我们考虑接如到GPRS网络。

2.2.3.1 GPRS网络的构成

GPRS(General Packet Radio Service)是通用分组无线业务的简称[5]。GPRS是GSM Phase2.1规范实现的内容之一，能提供比现有GSM网9.6kbit/s更高的数据率。GPRS采用与GSM相同的频段、频带宽度、突发结构、无线调制标准、跳频规则以及相同的TDMA帧结构。因此，在GSM系统的基础上构建GPRS系统时，GSM系统中的绝大部分部件都不需要作硬件改动，只需作软件升级。

构成GPRS系统的方法是：

(1) 在GSM系统中引入3个主要组件

1.GPRS服务支持结点(SGSN, Serving GPRS Supporting Node)

2.GPRS网关支持结点(GGSN, Gateway GPRS Support Node)

3.分组控制单元(PCU)

(2) 对GSM的相关部件进行软件升级。 GPRS系统原理如图2-7所示：

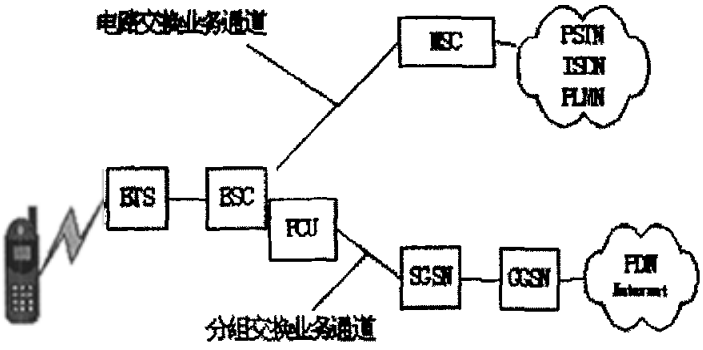


图 2-7 GPRS 系统原理图

ETSI 指定了 GSM900、1800 和 1900 三个工作频段用于 GSM，其中 GSM900 频段还有 G1(E-GSM)频段和 P 频段。相应地，GPRS 也工作于这三个频段，包括 GSM900 的 G1 频段和 P 频段，当然，GPRS 可以限制每个小区只工作于 P 频段。如表 2-1 所示了 GSM 和 GPRS 的工作频段。

表 2-1 GSM 和 GPRS 的工作频段

900MHz 频段	G1 频段上行频率(原 E－GSM)	880---890MHz
	P 频段上行频率	890---915MHz
	G1 频段下行频率(原 E－GSM)	925---935MHz
	P 频段下行频率	935---960MHz
	双工间隔	45MHz
	载频间隔	200kHz
1800MHz 频段	上行频率	1710---1785MHz
	下行频率	890---915MHz
	双工间隔	95MHz
	载频间隔	200kHz
1900MHz 频段	上行频率	1850---1910MHz
	下行频率	1930---1990MHz
	双工间隔	80MHz
	载频间隔	200kHz

现有的 GSM 移动台(MS)，不能直接在 GPRS 中使用，需要按 GPRS 标准进行改造(包括硬件和软件)才可以用于 GPRS 系统。GPRS 定义了 3 类 MS：

★A 类可同时工作于 GPRS 和 GSM；

★B 类可在 GPRS 和 GSM 之间自动切换工作；

★C 类可在 GPRS 和 GSM 之间人工切换工作。

GPRS 被认为是 2G 向 3G 演进的重要一步，不仅被 GSM 支持，同时也被北美的 IS-136 支持。

2.2.3.2 GPRS 网络主要实体

GPRS 网络主要实体包括 GPRS 支持节点、GPRS 骨干网、本地位置寄存器 HLR、短消息业务网关移动交换中心(SMS-GMSC)和短消息业务互通移动交换中心(SMS-IWMSC)、移动台、移动交换中心(MSC)/拜访位置寄存器(VLR)、分组数据网络(PDN)等[5][6]。

(1) GPRS 支持节点(GSN)：

GPRS 的支持节点 GSN 是 GPRS 网络中最重要的网络节点，包含了支持 GPRS 所需的功能。GSN 具有移动路由管理功能，可以连接各种类型的数据网络，并可以连到 GPRS 寄存器。GSN 可以完成移动台和各种数据网络之间的数据传送和格式转换。GSN 是一种类似于路由器的独立设备，也与 GSM 中的 MSC 集成在一起。在一个 GSM 网络中允许存在多个 GSN。GSN 有两种类型：SGSN 和 GGSN。

SGSN 是为移动终端(MS)提供业务的节点(即 Gb 接口由 SGSN 支持)。在激活 GPRS 业务时，SGSN 建立起一个移动性管理环境，包含关于这个移动终端(MS)的移动性和安全性方面的信息。SGSN 的主要作用就是记录移动台的当前位置信息，并且在移动台和 SGSN 之间完成移动分组数据的发送和接收。

GGSN 通过配置一个 PDP 地址被分组数据网接入。它存储属于这个节点的 GPRS 业务用户的路由信息，并根据该信息将 PDU 利用隧道技术发送到 MS 的当前的业务接入点，即 SGSN。GGSN 可以经 Gc 接口从 HLR 查询该移动用户当前的地址信息。GGSN 主要是起网关作用，它可以和多种不同的数据网络连接，如 ISDN 和 LAN 等。另外，GGSN 也又被称作 GPRS 路由器。GGSN 可以把 GSM 网中的 GPRS 分组数据包进行协议转换，从而可以把这些分组数据包传送到远端的 TCP/IP 或 X.25 网络。

SGSN 与 GGSN 的功能既可以由一个物理节点全部实现，也可以在不同的物理节点上分别实现。它们都应有 IP 路由功能，并能与 IP 路由器相连。当 SGSN 与 GGSN 位于不同的 PLMN 时，通过 Gp 接口互联。SGSN 可以通过任意 Gs 接口向 MSC/VLR 发送定位信息，并可以经 Gs 接口接收来自 MSC/VLR 的寻呼请求。

(2) GPRS 骨干网：

GPRS 中有内部 PLMN 骨干网和外部 PLMN 骨干网内部门 PLMN 骨干网是指位于同一个 PLMN 上的并与多个 GSN 互联的 IP 网。外部 PLMN 骨干网是指位于不同的 PLMN 上的并与 GSN 和内部 PLMN 骨干网互联的 IP 网，如图 2-8 所示。

每一个内部 PLMN 骨干网都是一个 IP 专网，且仅用于传送 GPRS 数据和 GPRS

信令。IP 专网是采用一定访问控制机制以达到所需安全级别的 IP 网。两个内部 PIMN 骨干网是使用边界网关(BG，Border Gateways)和一个外部 PLMN 骨干网并经 Gp 接口相连的，外部 PLMN 骨干网的选择取决于包含有 BG 安全功能的漫游协定，BG 不在 GPRS 的规范之列。外部 PLMN 可以是一个分组数据网。

在同一个 PLMN 骨干网内，骨干网是图 2-9 中虚线方框内的部分。在 GPRS 骨干网内部，各 GSN 实体之间通过 Gn 接口相连，它们之间的信令和数据传输都是在同一传输平台中进行的，所利用的传输平台可以在 ATM、以太网、DDN、ISDN、帧中继等现有传输网中选择。

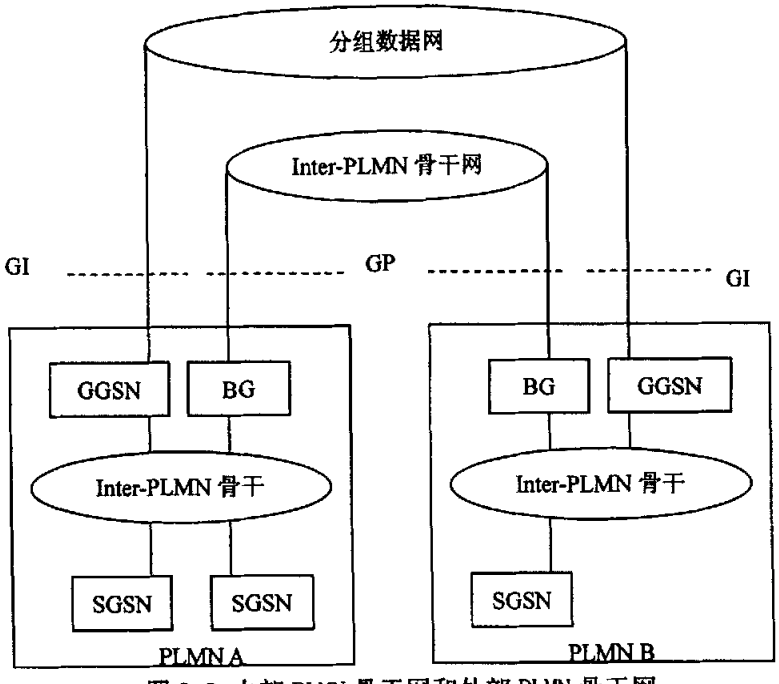


图 2-8 内部 PLMN 骨干网和外部 PLMN 骨干网

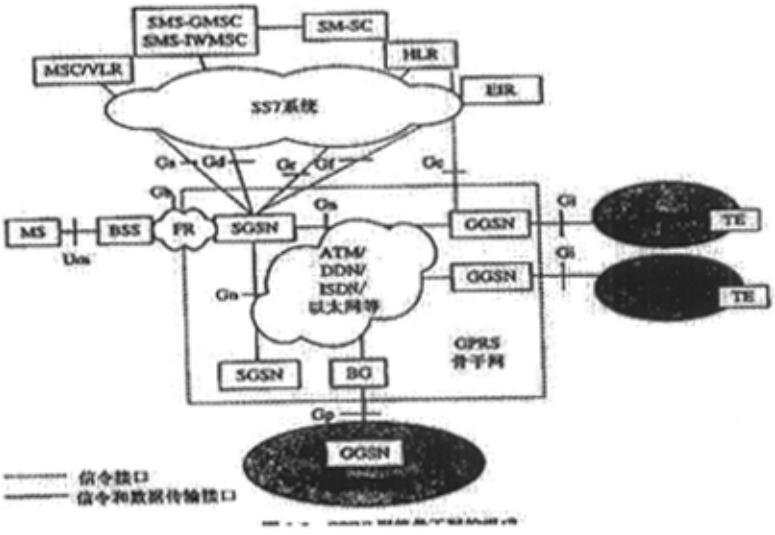


图 2-9 GPRS 网络骨干网的组成

（3）本地位置寄存器(HLR)：

在 HLR 中有 GPRS 用户数据和路由信息。从 SGSN 经 Gn 接口或 GGSN 经 Gc 接口均都可访问 HLR，对于漫游的 MS 来说，HLR 可能位于另一个不同的 PLMN 中，而不是当前的 PLMN 中。

（4） 消息业务网关移动交换中心（SMS-GMSC）和短消息业务互通移动交换中心（SMS－IWMSC)：

SMS-GMSC 和 SMS-IWMSC 经 Gd 接口连接到 SGSN 上，这样就能让 GPRS MS 通过 GPRS 无线信道收发短消息(SM)。

（5） GPRS 移动台：

GPRS MS 能以三个运行模式中的一个进行操作，其操作模式的选定由 MS 所申请的服务所决定：即仅有 GPRS 服务，同时具有 GPRS 和其他 GSM 服务，或依据 MS 的实际性能同时运行 GPRS 和其他 GSM 服务。

A 类(Class-A)操作模式：MS 申请有 GPRS 和其他 GSM 服务，而且 MS 能同时运行 GPRS 和其他 GSM 服务。

B 类(Class-B)操作模式：一个 MS 可同时监测 GPRS 和其他 GSM 业务的控制信道，但同一时刻只能运行一种业务。

C 类(Class-C)操作模式：MS 只能应用于 GPRS 服务。

（6） 移动交换中心(MSC)和拜访位置寄存器(VLR)：

在需要 GPRS 网络与其他 GSM 业务进行配合时选用 Gs 接口，如利用 GPRS 网络实现电路交换业务的寻呼，GPRS 网络与 GSM 网络联合进行位置更新，以及 GPRS 网络的 SGSN 节点接收 MSC/VLR 发来的寻呼请求等。同时 MSC/VLR 存储 MS(此 MS 同时接入 GPRS 业务和 GSM 电路业务)的 IMSI 以及 MS 相连接的 SGSN 号码。

（7） 分组数据网络(PDN)：

PDN 提供分组数据业务的外部网络。移动终端通过 GPRS 接入不同的 PDN 时，采用不同的分组数据协议地址。

2.2.3.3 GPRS 的特点

GPRS 主要特点[5][6]如下：

★GPRS 采用分组交换技术，高效传输高速或低速数据和信令，优化了对网络资源和无线资源的 利用。

★定义了新的 GPRS 无线信道，且分配方式十分灵活：每个 TDMA 帧可分配 1 到 8 个无线接口时隙。时隙能为活动用户所共享，且向上链路和向下链路的分配是独立的。

★ 支持中、高速率数据传输，可提供 9.05 ---171.2kbit/s 的数据传输速率(每用户)。GPRS 采用了与 GSM 不同的信道编码方案。

★ GPRS 网络接入速度快，提供了与现有数据网的无缝连接。

★ GPRS 支持基于标准数据通信协议的应用，可以和 IP 网、X.25 网互联互通。支持特定的点到点和点到多点服务，以实现一些特殊应用如远程信息处理。GPRS 也允许短消息业务(SMS)经 GPRS 无线信道传输。

★ GPRS 的设计使得它既能支持间歇的爆发式数据传输，又能支持偶尔的大量数据的传输。它支持四种不同的 QoS 级别。GPRS 能在 0.5 ---1 秒之内恢复数据的重新传输。GPRS 的计费一般以数据传输量为依据。

★ 在 GSM PLMN 中，GPRS 引入两个新的网络节点：一个是 GPRS 服务支持节点(SGSN)，它和 MSC 在同一等级水平，并跟踪单个 MS 的存储单元，实现安全功能和接入控制。节点 SGSN 通过帧中继连接到基站系统。另一个是 GPRS 网关支持节点 GGSN，GGSN 支持与外部分组交换网的互通，并经由基于 IP 的 GPRS 骨干网和 SGSN 连通。

★ GPRS 的安全功能同现有的 GSM 安全功能一样。身份认证和加密功能由 SGSN 来执行。其中的密码设置程序的算法、密钥和标准与目前 GSM 中的一样，不过 GPRS 使用的密码算法是专为分组数据传输所优化过的。GPRS 移动设备(ME)可通过 SIM 访问 GPRS 业务，不管这个 SIM 是否具备 GPRS 功能。

★ 蜂窝选择可由一个 MS 自动进行，或者基站系统指示 MS 选择某一特定的蜂窝。MS 在重选择另一个蜂窝或蜂窝组(即一个路由区)时会通知网络。

★ 为了访问 GPRS 业务，MS 会首先执行 GPRS 接入过程，以将它的存在告知网络。在 MS 和 SGSN 之间建立一个逻辑链路，使得 MS 可进行如下操作：接收基于 GPRS 的的 SMS 服务、经由 SGSN 的寻呼、GPRS 数据到来通知。

★ 为了收发 GPRS 数据，MS 会激活它所想用的分组数据地址。这个操作使 MS 可被相应的 GGGSN 所识别，从而能开始与外部数据网络的互通。

★ 用户数据在 MS 和外部数据网络之间透明地传输，它使用的方法是封装和隧道技术：数据包用特定的 GPRS 协议信息打包并在 MS 和 GGSN 之间传输。这种透明的传输方法缩减了 GPRS PLMN 对外部数据协议解释的需求，而且易于在将来引入新的互通协议。用户数据能够压缩，并有重传协议保护，因此数据传输高效且可靠。

★ GPRS 可以实现基于数据流量、业务类型及服务质量等级(QoS)的计费功能，计费方式更加合理，用户使用更加方便。

★ GPRS 的核心网络层采用 IP 技术，底层款可使用多种传输技术，很方便地实现与高速发展的 IP 网无缝连接。

2.2.3.4 GPRS 终端的选择及相关技术指标

GPRS终端是嵌入式无线通信实验系统的关键设备。目前，市场上可供选择的GPRS模块不多，使用最广泛的就是西门子公司的MC35及MC35I，其性能稳定，使用方便。鉴于上述优点，本课题选用MC35IT GPRS终端做为整个系统的GPRS接入设备。其核心板为MC35 GPRS模块。

MC35IT有丰富的AT指令，通过RS232与控制端相连，是传统调制解调器与GPRS无线通信系统相结合的一种数据通信终端设备，也被称为无线调制解调器。MC35IT将基带和射频电路集于一体，提供标准的AT命令接口，为数据、语音、短信、传真提供高速、稳定、可靠的传输。其主要技术指标如下[7][8]：

★双频带:EGSM900/GSM1800

★B类GPRS移动台

★适用于GSM 2/2+

★输出功率：功率级4(2w)在EGSM900

功率级1(IW)在GSM1800

★AT命令控制

★波特率：可选波特率300bPs—115kbps，自动波特率4.8-115kbps

★SIM应用工具箱

★工作电压范围：3.3—5.5V

★功耗：空闲模式(EGSM900/1800)：10Ma/10mA

语音模式(EGSM900/\800)：300 mA(均值) 2.0A(峰值)

睡眠模式:3mA

掉电模式:100μA

★正常工作温度范围：-20℃—+55℃

存放温度：-20℃

主要数据传输形式：

SMS短消息：

◆点对点MO和MT

◆SMS单元广播

◆文本和分组数据单元模式

FaX(传真)

Data(数据)：

◆CSD传输速度2.4、4.8、9.6、14.4kbps

◆USSD

◆非透明模式

◆V.110

◆GPRS:最大数据传输速率:85.6kbps(下行链路)

◆编码方案:CS1，2，3，4

◆PPP协议栈

数据接口：

MC35IT的数据输入/输出接口实际上是一个串行异步收发器，它符合ITU—TRS232接口标准，它有固定的参数:8位数据位和1位停止位，无校验位，波特率在300bps—115kbps之间可选，硬件握手信号用RTSO/CTSO，软件流量控制用XON/XOFF，CMOS电平，支持标准的AT命令集。通过这一接口可以用AT命令切换操作模式，可以使它处于语音、数据、短信息或传真模式。

MC35IT GPRS终端各接口如图2-10所示。

图2-10 MC35IT数据接口[9]

2.3 嵌入式无线通信实验系统的优点

综上所述，该实验系统具有如下几个优点：

★ 支持如USB、LCD、SM等多种接口，便于以后对实验项目进行扩充，及系统升级

★ 采用流行的uClinux操作系统，紧扣嵌入式系统发展的脉搏。随着uClinux功能的不断增强，在进行后续实验方案设计时能获得更大的设计空间，能完成更多的实验设计功能。

★ 成本，维护费用低。随着嵌入式系统的发展。开发板上所使用的主要芯片的价格会不断下跌，所以无论是其制造成本及维护的费用都会很低廉

★ 采用优质的GPRS模块，不但能开发无线通信及语音业务方面的实验，还可以无线接入internet，完成无线网应用方面的实验。

★ 试验项目范围广。

所以说该套实验系统的配置不但完全能适应现阶段实验的要求还具有很强的可扩充性，为未来设计更多的实验项目创造了条件，并且经济实用。

第三章 Boot Loader 及 uClinux 根文件系统实验实例

3.1 实验设计目的及要求

Boot Loader 是学习嵌入式系统的一个非常重要的环节。它与硬件紧密结合，通过对它的结构，原理的学习并实际动手编写简单的 Boot Loader 程序将有助于更深入地理解嵌入式系统。所以同建立 uClinux 的根文件系统一样，是一个非常重要的实验项目，做好这两个实验能为以后应用程序设计的实验打下良好的软硬件知识基础。这两个实验项目可以让学生熟悉该实验平台，掌握 ARM 的体系结构及其启动初始化过程，能熟练的应用 Thumb 指令及 C 编写初始化程序。同时还可以了解 uClinux 的文件系统，掌握内核的编译方法及根文件系统的制作方法。可以说这两个实验是一切嵌入式实验项目的准备试验。

整个实验涉及到的内容比较多，实验参与者引入项目管理内容，将项目分工提出进度计划表，并建立责任矩阵[1]，按期评估。

3.2 Boot Loader 实验项目的方案设计

3.2.1 Boot Loader 的概念及设计相关理论基础

在上一章中，我们介绍了 uClinux 操作系统的四个层次，首先就提到了 Boot Loader。Boot Loader 是在操作系统内核运行之前运行的一段小程序。通过这段小程序，初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。即是我们说所的引导加载程序。

在嵌入式系统中，通常没有像 PC 机 BIOS 那样的固件程序，因此整个系统的加载启动任务就完全由 Boot Loader 来完成。对于我们的实验开发板，由于 flash 接的是微处理器 Bank0 的片选信号。映射的地址为 0x00000000。而实验系统在上电或复位时从地址 0x00000000 处开始执行，所以在这个地址处安排系统的 Boot Loader 程序。由于 Boot Loader 与硬件结合非常紧密，所以对于不同结构的 CPU 其 Boot Loader 程序都是不同的。正是由于这样，我们必须要对 S3C44B0X 的寄存器结构有个整体的了解。

ARM 处理器共有 37 个寄存器[2][4]，分为若干个组（Bank），这些寄存器包括：

- 31 个通用寄存器，包括程序计数器（PC 指针），均为 32 位的寄存器；
- 6 个状态寄存器，用以标识 CPU 的工作状态及程序的运行状态，均为 32 位，目前仅使用了其中的一部分。

通用寄存器包括 R0—R15，可分为 3 类：

- 未分组寄存器 R0—R7；
- 分组寄存器 R8—R14；
- 程序计数器 R15；

同时，ARM 处理器又有 7 种不同的处理器模式。分别为：

- 用户模式（usr）：ARM 处理器正常的程序执行状态；
- 快速中断模式（fig）：用于高速数据传输或通道处理；
- 外部中断模式（irq）：用于通用的中断处理；
- 管理模式（svc）：操作系统使用的保护模式；
- 数据访问终止模式（abt）：当数据或指令预取终止时进入该模式。可用于虚拟存储及存储保护；
- 系统模式（sys）：运行具有特权的操作系统任务；
- 未定义指令终止模式（und）：当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真；

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。

在所有的运行模式下，未分组寄存器都指向同一物理寄存器，它们未被系统用作特殊的用途，因此，在中断或异常处理进行运行模式转换时，由于不同的处理器运行模式均使用相同的物理寄存器，可能会造成寄存器中数据的破坏。

对于分组寄存器，它们每一次所访问的物理寄存器与处理器当前的运行模式有关。对于 R8-R12，每个寄存器对应两个不同的物理寄存器，当使用 fig 模式时，访问寄存器 R8_fig-R12_fig；当使用除 fig 模式以外的其他模式时，访问寄存器 R8_usr-R12_usr。对于 R13、R14，每个寄存器对应 6 个不同的物理寄存器，其中一个是用户模式和系统模式共用，另外 5 个物理寄存器对应其他 5 种不同的运行模式。寄存器 R13 在 ARM 指令中常用作堆栈指针，但也可以使用其他的寄存器作为堆栈指针。而在 Thumb 指令集中，某些指令强制性的要求使用 R13 作为堆栈指针。R14 也称作子程序连接寄存器（Subroutine Link Register）或连接寄存器 LR。当执行 BL 子程序调用指令时，R14 中得到 R15（程序计数器 PC）的备份。他情况下，R14 用作通用寄存器。R15 用作程序计数器（PC），虽然也可以用作通用寄存器，但一般都不那么使用，因为对 R15 的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。由于 ARM 体系结构采用了多级流水线技术，对 ARM 指令集而言，PC 总是指向当前指令的下两条指令的地址，即 PC 的值为当前指令的地址值加 8 个字节。

总体说来，在 ARM 状态下，任一时刻可以访问以上所讨论的 16 个通用寄存

器和 1-2 个状态寄存器。在非用户模式（特权模式）下，则可以访问到特定模式分组寄存器。

Boot Loader 在嵌入式系统固态存储设备中的空间位置如图 3-1 所示。

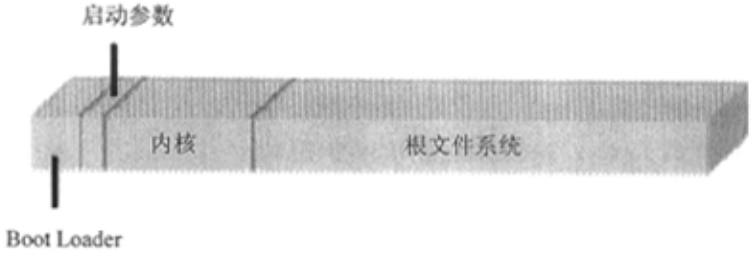


图 3-1 固态存储设备的典型空间分配结构

3.2.1.1 Boot Loader 的操作模式

Boot Loader 一般有两种不同的操作模式："启动加载"模式和"下载"模式，这种区别仅对于 Boot Loader 的开发人员才有意义。但从最终用户的角度看，Boot Loader 的作用就是用来加载操作系统，而并不存在所谓的启动加载模式与下载工作模式的区别。

★ 启动加载（Boot loading）模式：

这种模式也称为"自主"（Autonomous）模式。也即 Boot Loader 从目标机上的 falsh 上将操作系统加载到 RAM 中运行，整个过程并没有用户的介入。这种模式是 Boot Loader 的正常工作模式。

★ 下载（Downloading）模式：

在这种模式下，目标机上的 Boot Loader 将通过串口连接或网络连接等通信手段从主机（Host）下载文件，比如：下载内核映像和根文件系统映像等。从主机下载的文件通常首先被 Boot Loader 保存到目标机的 RAM 中，然后再被 Boot Loader 写到目标机上的 FLASH 类固态存储设备中。Boot Loader 的这种模式通常在第一次安装内核与根文件系统时被使用；此外，以后的系统更新也会使用 Boot Loader 的这种工作模式。我们将设计 Boot Loder 工作于这两种模式。并象终端用户提供操作系统加载命令，通过命令行输入来加载操作系统。

3.2.2 Boot Loader 功能设计

Boot Loader 需完成功能如下：

◆ARM 各寄存器、堆栈、SDRAM 的初始化；

◆串口、I/O 口、网口等外围设备的初始化；

◆时钟频率的初始化；

◆操作系统的下载，FLASH 与 SDRAM 间数据的搬运，及引导进入操作系统。

3.2.3 Boot Loader 的总体设计

在编写 Boot Loader 程序之前，我们必须明白这段程序的任务是什么，要做些什么样的工作。而由于 Boot Loader 的实现依赖于 CPU 的体系结构，针对不同的 CPU 需要编写不同的代码，如果全部用汇编来写那么代码将不能移植到其它结构不同的CPU上。因此我们将Boot Loader 设计为 stage1 和 stage2 两大部分。依赖于 CPU 体系结构的代码，比如设备初始化代码等，都放在 stage1 中，并且都用汇编语言来实现，以达到短小精悍的目的。而 stage2 则用 C 语言来实现，这样在后续的实验开发中可以不断对功能进行扩展和加强，而且代码会具有更好的可读性和可移植性。

Boot Loader 的 stage1 完成的功能可多可少，但是必须包括以下步骤：

◆ 硬件设备初始化。

◆ 设置堆栈。

◆ 为加载 Boot Loader 的 stage2 准备 RAM 空间。

◆ 拷贝 Boot Loader 的 stage2 到 RAM 空间中。

◆ 跳转到 stage2 的 C 入口点。

Boot Loader 的 stage2 必须包括以下基本的步骤：

◆ 初始化本阶段要使用到的硬件设备。

◆ 下载内核及根文件系统的映象文件到实验开发板

◆ 将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中。

◆ 加载内核映像和根文件系统映像。

◆ 调用内核。

3.3 Boot Loader 设计方案的具体实施

3.3.1 Boot Loader stage1 的功能

在涉及到具体汇编代码前，有一些术语[2]必须要说明一下：

◆映象文件（image）：指一个可执行文件，在执行的时候被加载到处理器中。它是 ELF（Executable and linking format）格式的。

- 段（section）：描述映象文件的代码或数据块。
- RO：是 Read-only 的简写形式。一般存放的代码。
- RW：是 Read-write 的简写形式。一般是存放初始化的数据。
- ZI：是 Zero-initialized 的简写形式。一般是存放零初始化数据。
- 输入段(input section)：它包含代码、初始化数据或描述了在应用程序运行之前必须要初始化为 0 的一段内存。
- 输出段（output section）：它包含了一系列具有相同 RO、RW、ZI 属性的输入段。
- 域（Regions）：在一个映象文件中，一个域包含了 1-3 个输出段。多个域组织在一起就构成了最终的映象文件

3.3.1.1 硬件的初始化

任何一个系统启动的最初任务都是硬件初始化。我们自己的 Boot Loader 也不例外，其主要目的是为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境[2][4]。它包括以下步骤：

1. 屏蔽所有的中断。为中断提供服务是操作系统的设备驱动程序的责任，因此在 Boot Loader 的执行全过程中不必响应任何中断。中断屏蔽可以通过写 ARM 的 CPSR 寄存器来完成[3]。代码段如下：

```
ldr r0,=INTMSK ； INTMSK 中断屏蔽寄存器，地址为 0x01e0000c
ldr r1,=0x07fffff ； 禁止所有中断
str r1,[r0]
```

2. 设置 CPU 的速度和时钟频率。代码段如下：

```
ldr r0,=LOCKTIME ； LOCKTIME 锁定时间计数值寄存器，地址为
                                0x01d8000c
ldr r1,=0xfff ； 初始值
str r1,[r0]

PLLSETSTART
ldr r0,=PLLCON ； PLLCON 锁相环控制寄存器，地址为 0x01d80000
ldr r1,=((M_DIV<<12)+(P_DIV<<4)+S_DIV) ； 设定系统主时钟频
                                                率
str r1,[r0]
ldr r0,=CLKCON ； CLKCON 时钟控制寄存器，地址为 0x01d80004
ldr r1,=0x7ff8          ;所有功能单元块时钟使能
```

```
str r1,[r0]
```

3.堆栈初始化。设置堆栈指针是为执行 C 语言代码作准备。把 sp 的值设置在 RAM 空间距最顶端 1.5KB 的地方(堆栈向上生长)。因为在不同的工作模式下所访问的堆栈指针寄存器（R13）不同[3]，所以要初始化各种工作模式的堆栈。代码段如下：

```
mrs r0,cpsr   ;复制 cprs 寄存器到 r0
bic r0,r0,#MODEMASK   ;MODEMASK:0x1f
orr  r1,r0,#UNDEFMODE|NOINT    ;UNDEFMODE:0X1B   NOINT:0XC0,
                                设为未定义指令终止模式禁止FIQ
                                和 IRQ 中断
msr cpsr_cxsf,r1
ldr sp,=UndefStack   ;0xc7ffb00

orr r1,r0,#ABORTMODE|NOINT
msr cpsr_cxsf,r1   ;数据访问终止模式
ldr sp,=AbortStack   ;0xc7ffd00

orr r1,r0,#IRQMODE|NOINT
msr cpsr_cxsf,r1   ;外部中断模式
ldr sp,=IRQStack   ;0xc7ffe00

orr r1,r0,#FIQMODE|NOINT
msr cpsr_cxsf,r1   ;快速中断模式
ldr sp,=FIQStack   ;0xc7fff00

bic r0,r0,#MODEMASK|NOINT
orr r1,r0,#SVCMODE
msr cpsr_cxsf,r1   ;管理模式
ldr sp,=SVCStack   ;0xc7ffb00
```

4. RAM 初始化。包括正确地设置系统内存控制器的功能寄存器以及各内存库控制寄存器等共有 13 个需要初始设置的寄存器。设置代码段如下：

```
adr r0, ResetHandler   ;取得 ResetHandler 的地址
ldr r1, =ResetHandler
sub r0, r1, r0   ;
```

```
ldr r1, =SMRDATA   ;取得这 13 个寄存器的初始设置段地址
sub r0, r1, r0   ;r0 指向 13 个寄存器初始化段
ldmia r0, {r1-r13}   ;将 13 个初始值推入到 r1-r13 中;
ldr r0, =0x01c80000   ;将 r0 指向 BWSCON, 地址为 0x01c80000.
stmia r0, {r1-r13}   ;将初始值填入这 13 个寄存器
```

其中从 SMRDATA 标号开始的一段连续的地址中存放了这 13 个寄存器的初始化数据.

3.3.1.2 为 stage2 准备 RAM 空间

为了获得更快的执行速度，我们设计把 stage2 加载到 RAM 空间中来执行，因此必须为加载 Boot Loader 的 stage2 准备好一段可用的 RAM 空间范围。但是这个空间范围到底需要多大呢。考虑到 stage2 是 C 语言执行代码，因此在考虑空间大小时，除了 stage2 可执行映象的大小外，还必须把堆栈空间也考虑进来。

另外 ARM 的寻址空间虽然为 4G，但实际映射的固态存储器的地址范围并没有那么大，往往只有几十兆，对于我们 S3C44B0X 开实验板来说，在 Bank6 的片选信号上我们接了 8M 的 SDRAM，而 Bank6 映射的起始地址为 0x0c000000，进而 RAM 的空间范围为 0x0c000000-0x0c7fffff，这时只要保证在 ADS 开发环境下指定程序的 RAM 运行空间在该范围内就可以避免使用到无效的 RAM 空间。但就一般情况而言，可以对 RAM 空间是否有效进行测试，其流程图如图 3-2。

RAM 空间的初始化流程图（图 3-3）及代码如下：

```
    Ldr r2, BaseOfBSS   ;数据段基地址
    Ldr r3, BaseOfZero   ;未初始化的数据段基址
0
    cmp r2, r3   ;比较 r2、r3 的值
    ldrcc r1, [r0], #4   ;r2<r3,就将 r0 的值赋给 r1,r0 地址+4，r0 的
                          值就是 RW 段的值
    strcc r1, [r2], #4   ;r2<r3,就将 r1 的值赋给 r2 指向的地址,r2 地
                          址+4
    bcc     %B0   ;跳回标号 0 处执行,直到 r2=r3
    mov r0,    #0   ;将 0 装入 r0
```

```
    ldr r3,    EndOfBSS   ;将不包含初始化数据数据段的尾地址装入 r3
1
    cmp r2,    r3   ;比较 r2 和 r3,此时 r2 值为 BaseOfZero
    strcc r0, [r2], #4   ;r2<r3,就将 r0 的值装入 r2 指向的地址,r2 地
                          址+4
    bcc    %B1   ;跳回标号 1 处执行,直到 r2=r3
```

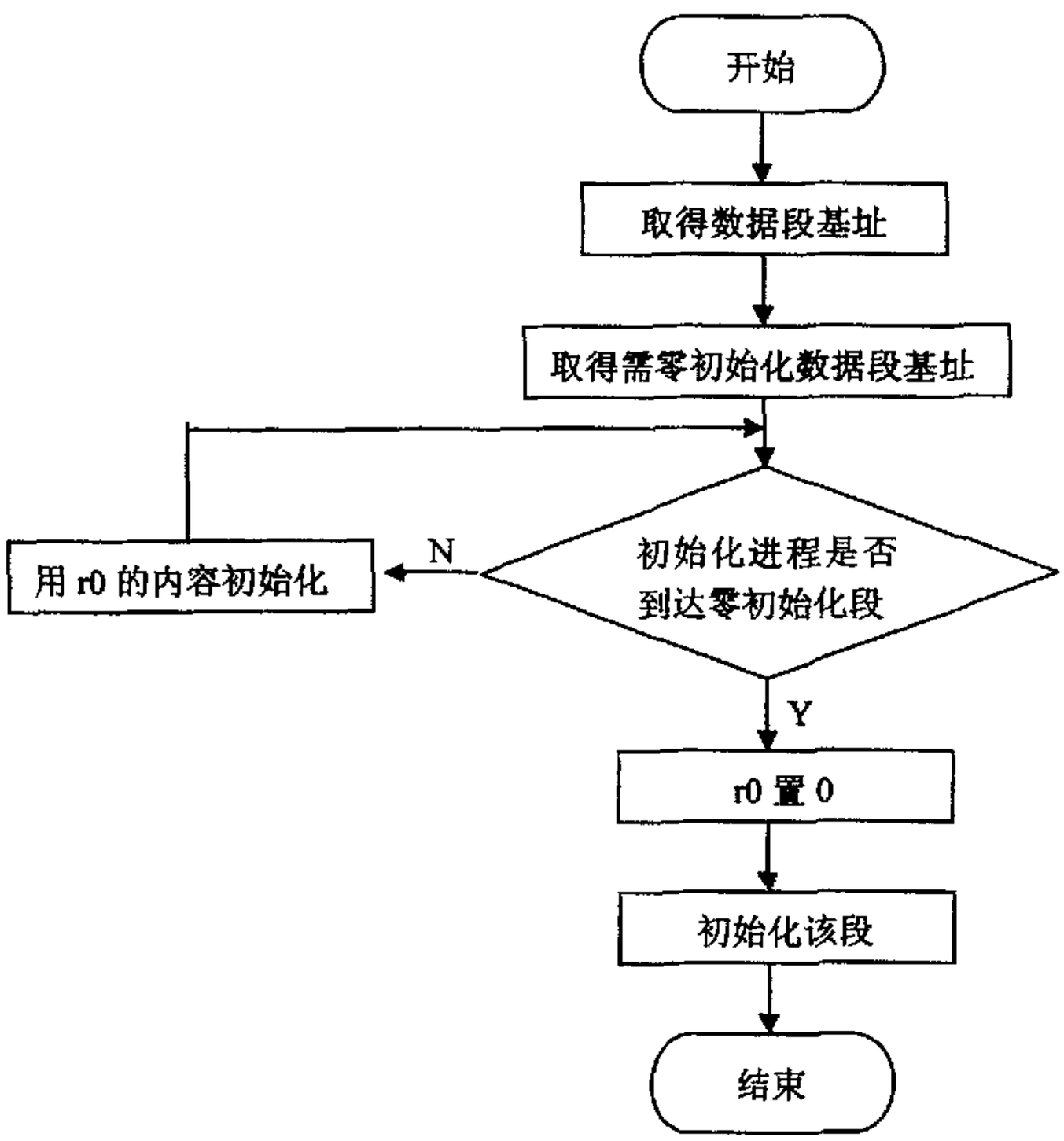


图 3-3 RAM 空间初始化

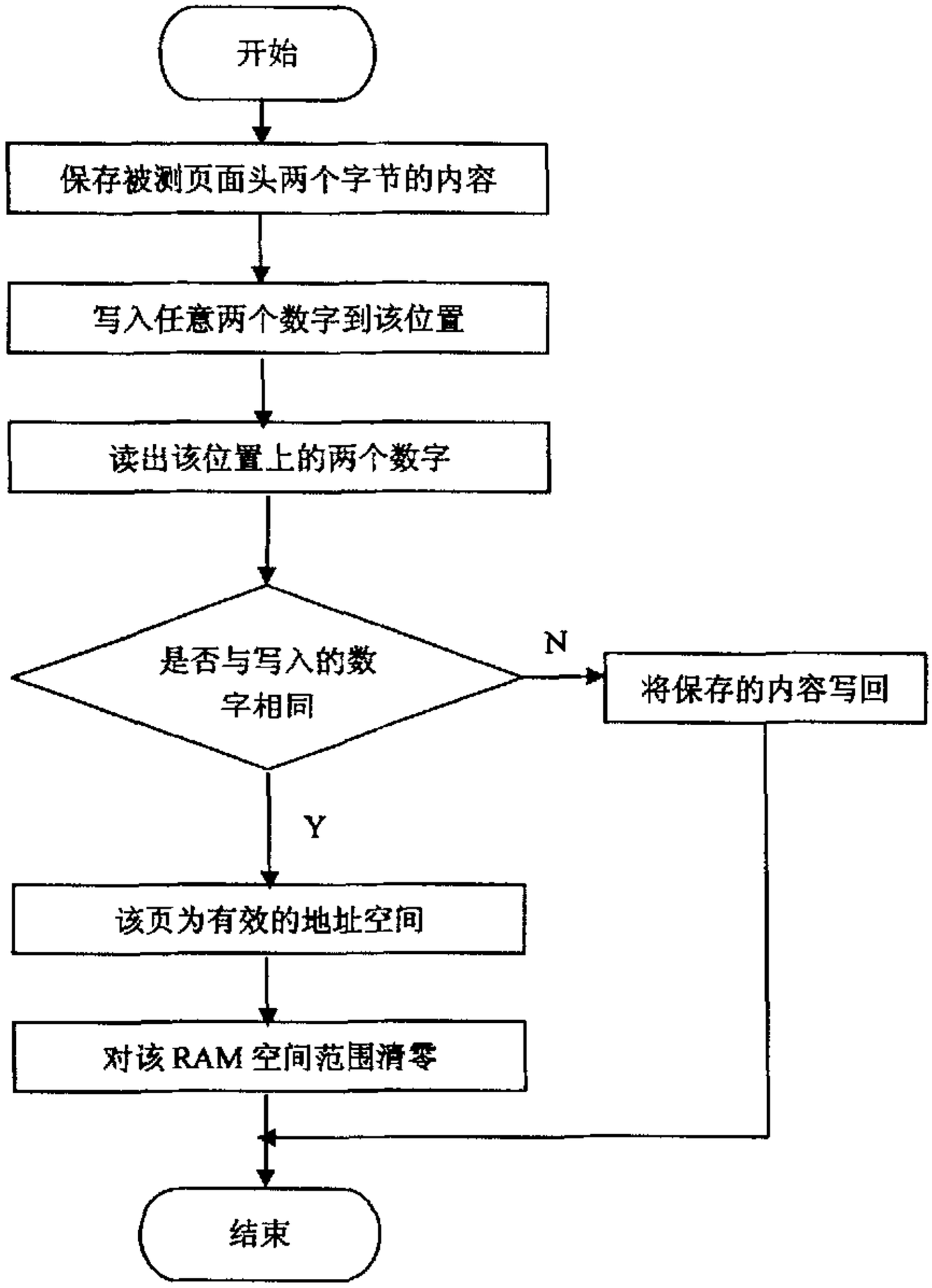


图 3-2 RAM 空间测试流程图

3.3.1.3 拷贝 stage2 到 RAM

这里我们专门在 Boot Loader 中设计了一个拷贝程序，负责将 stage2 的代码搬运到 RAM 中，让它在 RAM 中运行，以获得更快的速度。其整个搬运过程的代码段如下：

```
;//计算拷贝程序在 flash 中的实际位置//
 ldr r2,    =CopyProcBeg   ;拷贝程序的起始地址
 sub r1, r2, r1    ;r1 为 R0 段基址，计算偏移量
 add r0, r0, r1    ;r0=0, 为 flash 的起始地址, 算出了拷贝程序在 flash
                          中的地址, 该值赋给 r0
```

```
    ldr r3,    =CopyProcEnd   ;拷贝程序的结束地址

    ;//将拷贝程序复制到 RAM 中//

0
    ldmia r0!, {r4-r7}    ;将以 r0 所指的连续地址空间的内容写到 r4-r7
                          中,并将最后地址写回 r0
    stmia r2!, {r4-r7}    ;将 r4-r7 的内容写回到 r2 所指的联系地址空间
                          中,并将最后地址写回到 r2
    cmp r2, r3    ;较 r2 与 r3 的内容,看是否到拷贝终止地址
    bcc %B0    ;跳转到标号 0 处执行,直到 r2=r3

;//开始用 flash 中的拷贝程序复本将所有剩下的代码复制到 ram 中//

ldr   r3, TopOfROM
ldr   pc, =CopyProcBeg   ; 将程序指针指向 RAM 中拷贝程序的起始地址

;//本段将代码由实际烧入的地址拷贝到 ro-base 所指定的位置
    只拷贝 CopyProcEnd 以后的代码,即 stage2 的代码//

CopyProcBeg
0
   ldmia  r0!, {r4-r11}
   stmia  r2!, {r4-r11}
   cmp    r2, r3
   bcc    %B0
CopyProcEnd
```

上述两段代码所用到的 BaseOfBSS、EndOfBSS、BaseOfROM、TopOfROM 等的定义如下：

```
BaseOfROM DCD |Image$$RO$$Base|    ;为 BaseOfROM 分配一段空间并用
                                   Image 中 RO 段基址的值初始化
TopOfROM DCD  |Image$$RO$$Limit|   ;RO 段的大小
BaseOfBSS DCD |Image$$RW$$Base|    ;RW 段的基址
BaseOfZero DCD |Image$$ZI$$Base|   ;ZI 段基址
```

EndOfBSS DCD |Image$$ZI$$Limit| ;ZI 段的大小

经过上述这些执行步骤后，系统的物理内存布局如图 3-4 所示。

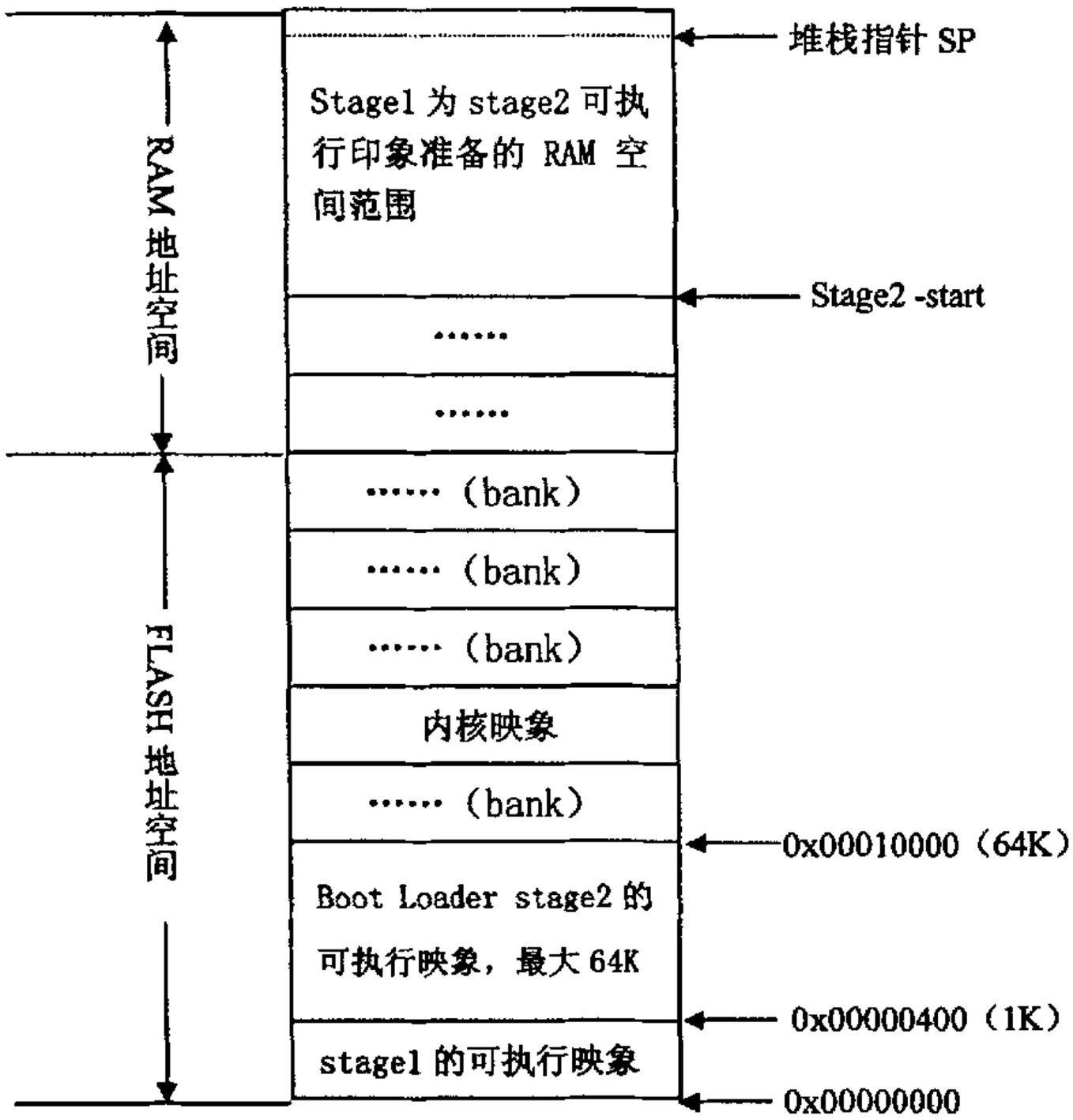


图 3-4 系统物理内存布局

在上述一切都就绪后，就可以跳转到 Boot Loader 的 stage2 去执行了。在 ARM 系统中，通过修改 PC 寄存器为合适的地址来实现。

3.3.2 Boot Loader Stage2 的功能

stage2 的代码用 C 语言实现，以便于实现更复杂的功能和取得更好的代码可读性和可移植性。当 stage1 的工作完成，为 stage2 准备好运行环境后就可以跳转进 main() 函数了。直接把 main() 函数的起始地址作为整个 stage2 执行映像的入口点是最直接的想法。但是这样做将无法处理 main（）返回的情况。为解决这个问题，可以写一段 ARM 的汇编程序，作为 main（）函数的外部包裹。该程序如下：

```
Lable
    BL      Main        ;从汇编进入 C 语言代码空间
    B       Lable
```

这样当 main() 函数返回后，我们又用一条跳转指令跳转到标号处继续执行程序，也就重新执行 main() 函数。

3.3.2.1 硬件初始化

进入到C语言空间后，同样需要初始化本阶段用到的硬件设备[3]，它们包括：

- 初始化 I/O 端口及两个串口，以便对终端用户输出 I/O 信息及与 GPRS 模块通信。其代码段如下：

```
void Port_Init(void)     //初始化 I/O 口
{
   //PORT A GROUP
   PCONA
   位名称          BIT              描述
   PA9             [9]          0 = Output 1 = ADDR24
   PA8             [8]          0 = Output 1 = ADDR23
   PA7             [7]          0 = Output 1 = ADDR22
   PA6             [6]          0 = Output 1 = ADDR21
   PA5             [5]          0 = Output 1 = ADDR20
   PA4             [4]          0 = Output 1 = ADDR19
   PA3             [3]          0 = Output 1 = ADDR18
   PA2             [2]          0 = Output 1 = ADDR17
   PA1             [1]          0 = Output 1 = ADDR16
   PA0             [0]          0 = Output 1 = ADDR0
   //
   rPCONA = 0x3ff;

   //PORT B GROUP
   rPCONB
   位名称          BIT              描述
   PB10            [10]         0 = Output 1 = nGCS5
   PB9             [9]          0 = Output 1 = nGCS4
   PB8             [8]          0 = Output 1 = nGCS3
   PB7             [7]          0 = Output 1 = nGCS2
   PB6             [6]          0 = Output 1 = nGCS1
   PB5             [5]          0 = Output 1 = nWBE3/nBE3/DQM3
   PB4             [4]          0 = Output 1 = nWBE2/nBE2/DQM2
   PB3             [3]          0 = Output 1 = nSRAS/nCAS3
   PB2             [2]          0 = Output 1 = nSCAS/nCAS2
   PB1             [1]          0 = Output 1 = SCLK
   PB0             [0]          0 = Output 1 = SCKE
   //
```

```
rPDATB = 0x1cf;
rPCONB = 0x1cf;

//PORT C GROUP
//BUSWIDTH=16
PCONC
```

位名称	BIT	描述
PC15	[31:30]	00 = Input 01 = Output 10 = DATA31 11 = nCTS0
PC14	[29:28]	00 = Input 01 = Output 10 = DATA30 11 = nRTS0
PC13	[27:26]	00 = Input 01 = Output 10 = DATA29 11 = RxD1
PC12	[25:24]	00 = Input 01 = Output 10 = DATA28 11 = TxD1
PC11	[23:22]	00 = Input 01 = Output 10 = DATA27 11 = nCTS1
PC10	[21:20]	00 = Input 01 = Output 10 = DATA26 11 = nRTS1
PC9	[19:18]	00 = Input 01 = Output 10 = DATA25 11 = nXDREQ1
PC8	[17:16]	00 = Input 01 = Output 10 = DATA24 11 = nXDACK1
PC7	[15:14]	00 = Input 01 = Output 10 = DATA23 11 = VD4
PC6	[13:12]	00 = Input 01 = Output 10 = DATA22 11 = VD5
PC5	[11:10]	00 = Input 01 = Output 10 = DATA21 11 = VD6
PC4	[9:8]	00 = Input 01 = Output 10 = DATA20 11 = VD7
PC3	[7:6]	00 = Input 01 = Output 10 = DATA19 11 = IISCLK
PC2	[5:4]	00 = Input 01 = Output 10 = DATA18 11 = IISDI
PC1	[3:2]	00 = Input 01 = Output 10 = DATA17 11 = IISDO
PC0 [1:0]		00 = Input 01 = Output 10= DATA16 11 = IISLRCK

```
//
    rPDATC = 0x0001;
    rPCONC = 0x5f540554;
```

```
  rPUPC  = 0x0300;  /*允许上拉电阻连接到对应脚*/
//PORT D GROUP
```

PCOND

位名称	BIT	描述
PD7	[15:14]	00 = Input 01 = Output 10 = VFRAME 11 = Reserved
PD6	[13:12]	00 = Input 01 = Output 10 = VM 11 = Reserved
PD5	[11:10]	00 = Input 01 = Output 10 = VLINE 11 = Reserved
PD4	[9:8]	00 = Input 01 = Output 10 = VCLK 11 = Reserved
PD3	[7:6]	00 = Input 01 = Output 10 = VD3 11 = Reserved
PD2	[5:4]	00 = Input 01 = Output 10 = VD2 11 = Reserved
PD1	[3:2]	00 = Input 01 = Output 10 = VD1 11 = Reserved
PD0	[1:0]	00 = Input 01 = Output 10= VD0 11 = Reserved

```
//
rPDATD= 0x55;
rPCOND= 0xaaaa;
rPUPD = 0x00;
//PORT E GROUP
```

PCONE

位名称	BIT	描述
PE8	[17:16]	00 = Reserved(ENDIAN) 01 = Output 10 = CODECLK 11 = Reserved
PE7	[15:14]	00 = Input 01 = Output 10 = TOUT4 11 = VD7
PE6	[13:12]	00 = Input 01 = Output 10 = TOUT3 11 = VD6
PE5	[11:10]	00 = Input 01 = Output 10 = TOUT2 11 = TCLK in
PE4	[9:8]	00 = Input 01 = Output 10 = TOUT1 11 = TCLK in
PE3	[7:6]	00 = Input 01 = Output 10 = TOUT0 11 = Reserved
PE2	[5:4]	00 = Input 01 = Output 10 = RxD0 11 = Reserved
PE1	[3:2]	00 = Input 01 = Output

		10 = TxD0 11 = Reserved
PE0	[1:0]	00 = Input 01 = Output 10= Fpllo out 11 = Fout out

```
//
 rPDATE = 0x357;
 rPCONE = 0x556b;
 rPUPE  = 0x6;
```

```
//PORT F GROUP
 PCONF
```

位名称	BIT	描述
PF8	[21:19]	000 = Input 001 = Output 010 = nCTS1 011 = SIOCLK 100 = IISCLK Others = Reserved
PF7	[18:16]	000 = Input 001 = Output 010 = RxD1 011 = SIORxD 100 = IISDI Others = Reserved
PF6	[15:13]	000 = Input 001 = Output 010 = TxD1 011 = SIORDY 100 = IISDO Others = Reserved
PF5	[12:10]	000 = Input 001 = Output 010 = nRTS1 011 = SIOTxD 100 = IISLRCK Others = Reserved
PF4	[9:8]	00 = Input 01 = Output 10 = nXBREQ 11 = nXDREQ0
PF3	[7:6]	00 = Input 01 = Output 10 = nXBACK 11 = nXDACK0
PF2	[5:4]	00 = Input 01 = Output 10 = nWAIT 11 = Reserved
PF1	[3:2]	00 = Input 01 = Output 10 = IICSDA 11 = Reserved
PF0	[1:0]	00 = Input 01 = Output 10= IICSCL 11 =Reserved

```
//
 rPDATF = 0x0;
 rPCONF = 0x22445a;
 rPUPF  = 0x1d3;
```

```
//PORT G GROUP
 PCONG
```

位名称	BIT	描述
PG7	[15:14]	00 = Input 01 = Output 10 =IISLRCK 11 = EINT7
PG6	[13:12]	00 = Input 01 = Output

		10 = IISDO 11 = EINT6
PG5	[11:10]	00 = Input 01 = Output
		10 = IISDI 11 = EINT5
PG4	[9:8]	00 = Input 01 = Output
		10 = IISCLK 11 = EINT4
PG3	[7:6]	00 = Input 01 = Output
		10 = nRTS0 11 = EINT3
PG2	[5:4]	00 = Input 01 = Output
		10 = nCTS0 11 = EINT2
PG1	[3:2]	00 = Input 01 = Output
		10 = VD5 11 = EINT1
PG0	[1:0]	00 = Input 01 = Output
		10 = VD4 11 = EINT0

```
//
    rPDATG = 0xff;
    rPCONG = 0x00ff;
    rPUPG  = 0x00;

    rSPUCR=0x7;
    rEXTINT=0x0; /*所有的外部硬件中断为低电平触发*/
}
void Uart_Init(int mclk,int baud)   /*串口初始化程序*/
{
    int i;
    if(mclk==0)
   mclk=MCLK;
    rUFCON0=0x0;      /*禁止 FIFO*/
    rUFCON1=0x0;
    rUMCON0=0x0;
    rUMCON1=0x0;
/*初始化 UART0 相关寄存器*/
    rULCON0=0x3;      /*8 位数据，1 位停止，无奇偶校验*/
    rUCON0=0x245;     /*接收为边缘触发，发送为电平触发。禁止超
                        时中断，使能接收中断*/
    rUBRDIV0=( (int)(mclk/16./baud + 0.5) -1 );/*波特率计算*/
/*初始化 UART1 相关寄存器*/
    rULCON1=0x3;    /*8 位数据，1 位停止，无奇偶校验*/
```

```
    rUCON1=0x245; /*接收为边缘触发，发送为电平触发。禁止超时中
                    断，使能接收中断*/
    rUBRDIV1=( (int)(mclk/16./baud + 0.5) -1 );
}
```

设定系统主频。代码段如下：

```
void PllValue_Init(int mdiv,int pdiv,int sdiv)
{
int i = 1;

 rPLLCON = (mdiv << 12) | (pdiv << 4) | sdiv; /*rpllcon 为
                                              PLL 控制寄存
                                              器*/


while(sdiv--)
    i *= 2;

MCLK = (10000000*(mdiv+8))/((pdiv+2)*i); /*固定计算公式，
                                          10000000 为外部
                                          晶振频率*/
}
```

初始化网络设置等其代码如下：

```
IP_ADDRESS = IP4_ADDR(192,168,3,100);
MASK_ADDRESS = IP4_ADDR(255,255,255,0);
GATE_ADDRESS = IP4_ADDR(192,168,3,1);
```

IP4_ADDR()函数将采用 IPV4 协议地址转换为十六进制数并填入一个字中。

设备初始化完成后，输出一些打印信息。提示用户可以进行进一步的操作了。

3.3.2.2 内核映象文件及根文件系统的下载

我们设计编写 Boot Loader 的最终目的是要在实验板上运行 uClinux。这就需要我们将 uClinux 的内核及根文件系统的映象文件下载到我们的实验板上。要完成这样的任务，根据实验板上的硬件接口配置，有两种实现途径：

（1） 利用网口，使用 TFTP 下载；

（2） 利用串口下载；

使用网络，由于现阶段没有操作系统的支持，对从主机接收 TFTP 包的一些

处理需要自己编程实现，不但增加了编程的难度也增大了 Boot Loader 的体积。而使用串口虽然速度上不及 TFTP 快，但毕竟要传输的文件不大，编程比较容易实现。而且目前许多运行在 PC 上的串口工具都有发送文件的功能。鉴于上述考虑，我们决定使用串口来完成传输内核和根文件系统映象文件的到 SDRAM 的任务，配合一段搬运程序，就可以将它们放到 flash 中我们要求的位置[10][11][12]。其代码段如下：

```
int ComLoad(int argc, char *argv[])
{
   int i,size;
unsigned   char   *buf=(unsigned   char   *)DFT_DOWNLOAD_ADDR;/*
                                          DFT_DOWNLOAD_ADDR 为
                                          宏定义，默认的目的下
                                          载    地    址    为
                                          0x0c008000*/
   unsigned char RxTmp[8];
   if(argc<2)   /*采用默认的下载目的地址*/
      download_addr = DFT_DOWNLOAD_ADDR;
   else {
      unsigned long tmp;
      tmp = strtoul((unsigned char*)argv[1]);
      download_addr = (tmp==-1)?download_addr:tmp;
   }
   buf = (unsigned char *)download_addr;
   printf("Now   downloadfile   from   uart0   to   0x%x...\n",
download_addr);
   i = 0;
   while(i<4)
      RxTmp[i++] = getch();/*发送的文件的头部，占 4 个字节*/
    i = 0;
    size = *(unsigned long *)RxTmp - 4;/*发送的文件头部为发送
                                        文件的长度*/
while(i<size)
    buf[i++] = getch();
download_len = size;
```

```
        printf("Download File Size = %x\n", download_len);
        puts("Dwonload success\n");
        return 0;
    }
```

上面的程序执行完以后，文件是放在以 0x0c008000 开始的 SDRAM 中的，下面的程序段则是负责将文件搬运到 flash 中。我们的实验板的 flash 采用的是 SST39VF160。其有 20 根地址线，寻址空间为（2^{20}=1M），数据位宽为 16 位。当执行编程操作时，会需要检验是否将数据写到了正确的位置，其算法流程如图 3-5。

根据该算法，flash 编程代码段如下

```
int FlashProg(unsigned int ProgStart, unsigned short *DataPtr, unsigned
int WordCnt)
{
    unsigned short ok, count;
    unsigned short i, j;

    ProgStart += ROM_BASE;/*ROM_BASE 为宏定义,为 0*/
    ok = 1;

    for( ; WordCnt && ok; ProgStart+=2, DataPtr++, WordCnt--)
    {
        j = *DataPtr;
        CMD_ADDR0 = 0xaaaa;/*完成 flash 编程的必要步骤, CMD_ADDR0、
                                    CMD_ADDR1 为宏定义*/
        CMD_ADDR1 = 0x5555;
        CMD_ADDR0 = 0xa0a0;
        *(volatile unsigned short *)ProgStart = j;

        count = 10000;
        while(count --)/*测试是否正确写入了要写的数据*/
        {
            i = *(volatile unsigned short *)ProgStart&0x40;
            if(i!=*(volatile unsigned short *)ProgStart&0x40) /*比较据
                                                        第 6 位*/
```

```
            continue;
          if((*(volatile unsigned short *)ProgStart)==j)   /*是否为真
                                                          实的写入数据*/
            break;
      }
      if(count == 0) ok = 0;
  }
      return ok;
  }
```

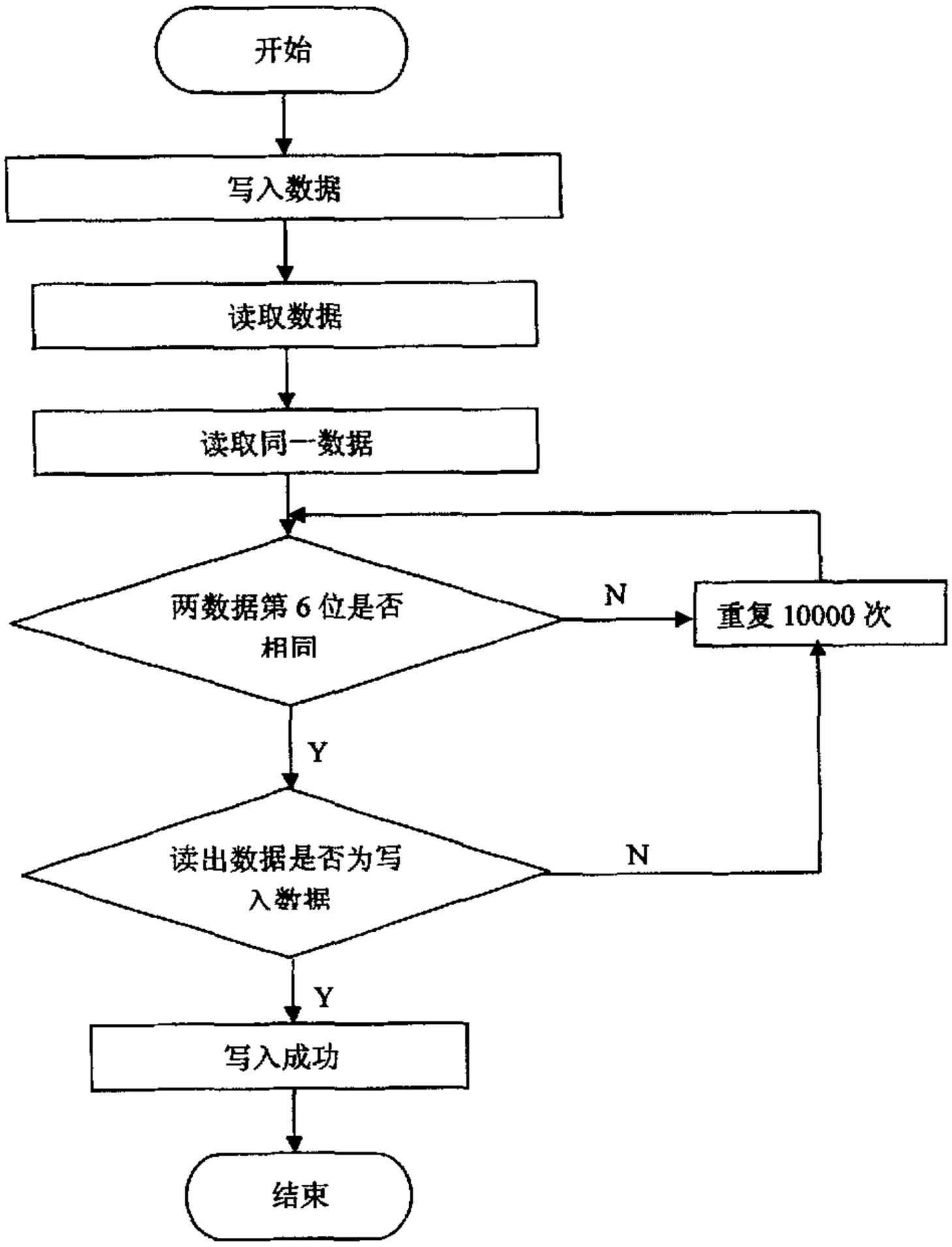


图 3-5 Flash 编程检验流程

实现了以上两个函数以后，我们的 Boot Loader 就具备了最基本的功能，就可以准备开始加载内核，开始运行 uClinux 了。

3.3.2.3 加载内核映像和根文件系统映像

由于嵌入式系统的硬件资源非常有限，所以对内存的使用需要事先进行规划。要使用 uClinux 操作系统需要两个映象文件：内核映象和根文件系统映象，在启动系统时它们都需要写入内存。所以在对内存进行规划时包括两方面：

- 内核映像所占用的内存范围，基地址和映像的大小；
- 根文件系统所占用的内存范围，基地址和映像的大小。

说到这里，就需要说说 uClinux 映象文件的执行方式。一种方式是内核映象文件直接在 FLASH 中运行。而另一种方式是先将其解压到 SDRAM 中，再在 SDRAM 中运行。由于在 SDRAM 中的运行速度比 FLASH 快，所以我们将采用后一种方式。对于内核映像，因为嵌入式 Linux 的内核一般都不操过 1MB，所以我们将其拷贝到 SDRAM 地址 0x0c300000 处。这样当开始启动 uClinux 时，压缩的 uClinux 内核将自动解压到 0x0c008000 处开始运行。当然也可以修改为其它的地址范围，但这样做就需要对 uClinux 源代码包的 makefile 文件进行相应的修改，操作起来非常麻烦，所以我们不对它进行改动。

而对根文件系统映像，我们将其拷贝到 0x0c000000+0x00100000 开始的地方。

像 ARM 这样的嵌入式微处理器都是在统一的内存地址空间中寻址 Flash 等固态存储设备的，所以从 Flash 上读取数据与从 RAM 单元中读取数据的操作并没有什么不同。完成从 Flash 设备上拷贝映像的工作代码段如下：

```
s = (unsigned char *)prog_s_addr;/*内核映象在 FLASH 中的起始地址*/
r = (unsigned char *)prog_r_addr;/*搬运到 RAM 中的起始地址*/
        size = __ROM_SIZE - prog_s_addr;/*计算映象文件大小，字节*/
        for(i=0; i<size; i++)
            r[i] = s[i];/*搬运*/
```

3.3.2.4 调用内核

Boot Loader 调用 Linux 内核的方法是直接跳转到内核的第一条指令处，对 S3C44B0X 来说也即直接跳转到 0x0c300000 地址处。

当调转后 uClinux 就开始启动。我们的 Boot Loader 使用 mrun 命令，设置启动参数，将内核映象从 flash 拷贝到地址 0x0c300000 处，并开始运行，将控制权交给操作系统，进而启动操作系统。由于 mrun 命令使用的函数调用及宏定义比较多，这里就不给出源代码了。

完成了上述任务以后，Boot Loader 设计实验项目完成

3.4 uClinux 根文件系统建立实验需完成的任务

在 Linux 环境下完成内核映象文件 zimage 和根文件系统映象文件 romfs.img 的制作

3.5 uClinux 根文件系统建立实验的具体实施

3.5.1 建立 arm-Linux 交叉开发环境

我们要在 uClinux 下进行应用程序的开发调试，首先就必须在宿主机上建立交叉开发环境，使宿主机能够编译生成实验开发板能够运行的代码。我们宿主机的开发环境采用 Red Hat Linux9.0 (kernel 2.4.18, gcc3.2, glibc2.2.93)。接下来就需要准备一份 uClinux 的源代码，好在 uClinux 是一个源代码开放的操作系统，在网上很容易就能下载到。这里我们使用的是：

- 纯 Linux-2.4.x 内核（已做好对 S3C44B0X 的移植）；
- uClinux-dist-20030522.tar.gz

另外还有就是 ARM 的交叉编译器，让我们能在 PC 机上编译得到运行于 ARM7 微处理器上的操作系统内核。我们从网上下载得到 ARM 交叉编译器：arm-elf-tools-20030314.sh[13][14]。在主机上执行以下命令：

```
sh arm-elf-tools-20030314.sh
```

这个命令执行后会在开发主机上自动建立一个 uClinux-ARM 的交叉编译环境。

交叉编译环境建立好以后，我们就要着手对 uClinux 内核进行重新编译，生成操作系统内核和 ROMFS 根文件系统。

3.5.2 uClinux 文件系统的生成

首先在 PC 主机上解压 uClinux-dist-20030522.tar.gz 到/work 目录下，命令如下：

```
cd /work
tar jxvf uClinux-dist-20030522.tar.gz
```

之后会在/work 目录下生成 uClinux-dist 目录。

进入该目录，加入 S3C44B0 的在 make config 时的厂商/产品选项。方法如下：

在 uClinux-dist\vendors\Samsung 下新建 S3C44B0 目录，将 uClinux-dist\vendors\Samsung\4510B 下的内容全部复制到 S3C44B0 目录下。

这里面有几个文件需要修改：

“config.linux-2.4.x”这个是 linux 内核编译配置选项文件。现在针对 S3C44B0 我们要修改的是# System Type 到# General setup 之间的内容。修改如下：

```
#
# System Type
#
# CONFIG_ARCH_DSC21 is not set
# CONFIG_ARCH_CNXT is not set
# CONFIG_ARCH_SWARM is not set
CONFIG_ARCH_S3C44B0=y                    #指明是处理器类型是 S3C44B0
# CONFIG_ARCH_ATMEL is not set
CONFIG_NO_PGT_CACHE=y
CONFIG_CPU_32=y
# CONFIG_CPU_26 is not set
CONFIG_CPU_ARM710=y
CONFIG_CPU_WITH_CACHE=y
# CONFIG_CPU_WITH_MCR_INSTRUCTION is not set
CONFIG_SERIAL_S3C44B0=y                  #使用 S3C44B0 的串口
DRAM_BASE=0x0c000000                     #SDRAM 起始是地址
DRAM_SIZE=0x00800000                     #SDRAM 大小
FLASH_MEM_BASE=0x00000000                #FLASH 起始地址
FLASH_SIZE=0x00200000                    #FLASH 大小
    # General setup
```

以后的 make 都以 CONFIG_ARCH_S3C44B0=y 这选项来解决是编译和 S3C44B0 相关的其他选项。

这样在 make menuconfig 后在 Vendor/Product 下可以看到有 Samsung/S3C44B0 的选项了。

跟着我们要编译源码得到内核映象文件，具体操作步骤如下：

1.解压 uClinux-2.4.24.tar.bz2 到/work 目录下。在 Red Hat Linux9.0 桌面下启动终端，输入以下命令：

```
cd /work
tar jxvf uClinux-2.4.24.tar.bz2
```

之后会在/work 目录下生成 linux-2.4.x 的目录。

2.进入 linux-2.4.x 目录，执行 make menuconfig，出现配置菜单，移动到 load an alternate configuration file 选项并回车，在文件名输入框输入 kernel_44b0.cfg 后再回车返回主菜单，再选 exit 退出，退出时选 Y 确认保存设置。

3.执行 make dep 和 make zImage,完成后会在 arch/armnommu/boot 目录下生成压缩内核 zImage。保存该压缩内核到/work 目录下备用，因为在后续的制做 ROMFS 文件系统的过程中/boot 目录下的压缩内核会被使用。

接下来就是生成 ROMFS 文件系统了，其步骤如下：

1.用/work 目录下的 linux-2.4.x 替换/work/uClinux-dist/linux-2.4.x 文件夹。

2.回到/work/uClinux-dist 目录：cd /work/uClinux-dist。输入命令：make menuconfig。选择 Linux-2.4.x 内核及 uclibc。这里我们用 0522dist 来编译生成 ROMFS 根文件系统。

3.在 user application 配置上选上 boa、ping、console shell、sash、tftp、ppp、http、telnet、busybox 等应用程序。

4.make dep;

5.make lib_only;

6.make user_only;

7.make romfs;

执行完上述命令后就会在/work/uClinux-dist 目录下出现 ROMFS 目录,我们需要的 RMOFS 根文件系统的印象文件 ROMFS.img 就保存在里面.

最后通过局域网将压缩内核印象文件及根文件系统印象文件传输到开发板的 SDRAM 中,再由 Boot Loader 搬运到 FLASH 中.这样就在开发板上建立了 uClinux 环境.

主机回到 Windows 环境，打开串口调试工具，设置好连接。启动实验板，在 Boot Loader 信息打印完后，输入命令 mrun。uclinux 就开始启动了。在 sash 提示符后输入命令：LS 就可以看见整个操作系统的目录，使用 CD 命令可进入具体的某个文件夹。

3.6 实验结论

通过上述两个实验项目，学生能够掌握 ARM 微处理器的结构和启动的具体过程同时熟悉了 ARM 编程开发过程及在 uClinux 下文件的编译过程。在完成了两个实验项目，在嵌入式无线通信实验开发板上建立了 uClinux 的开发环境后，由于有了一定的嵌入式开发基础，就可以进一步开始应用程序开发的实验。

第四章 短信及语音通信应用程序设计实验实例

4.1 实验的目的及要求

嵌入式系统的灵魂是嵌入式软件的设计。通过实验，能深刻了解 GPRS 终端的工作过程和相关的 AT 控制指令。熟悉 Linux 操作系统和相关的系统函数，熟练地在 Linux 环境下用 C 语言编程，使用交叉编译工具编译连接程序，并使用 GDB 对程序进行调试。掌握填加 uClinux 应用程序的方法。

整个实验涉及到的内容比较多，实验参与者引入项目管理内容，将项目分工，提出进度计划表，并建立责任矩阵，按期评估。程序设计要给出流程图

4.2 短信及语音通信应用程序设计实验方案设计

4.2.1 短信及语音通信应用程序设计实验的相关准备知识

4.2.1.1 Linux 与 uClinux 的异同

嵌入式系统结构如下：

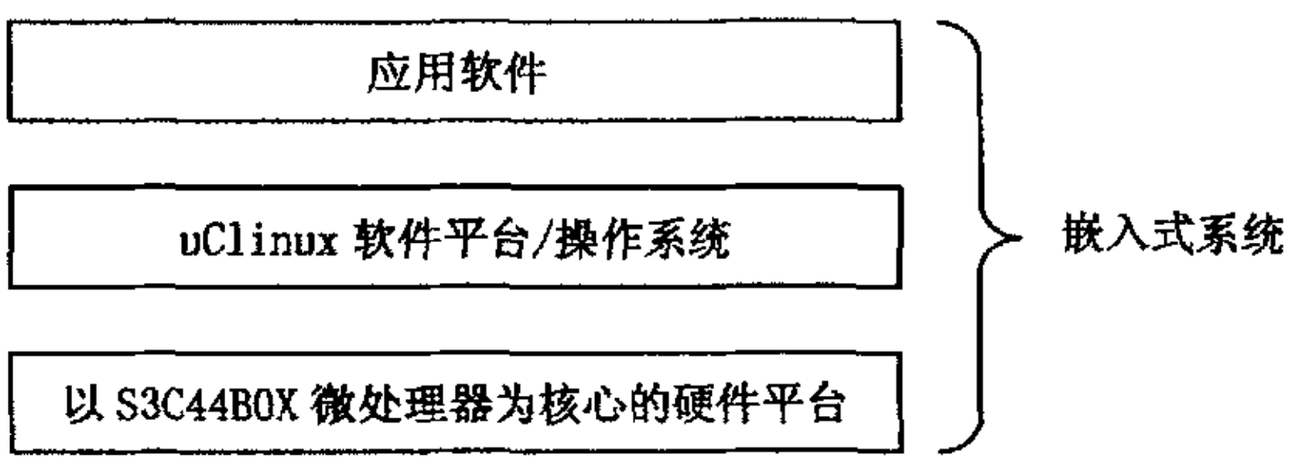


图 4-1 嵌入式系统结构

虽然 uClinux 是标准 Linux 的一个分支[13][14]，但是对于很多已经在标准 Linux 环境中工作得很好的程序，却并不能直接在 uClinux 环境上运行。原因有两方面：一方面，是由于嵌入式的 uClinux 所使用的处理器和普通 PC 不同，指令集、CPU 结构上的差异导致 uClinux 上运行的程序需要专门为该类型处理器交叉编译产生；另一方面，uClinux 是为没有内存管理单元（MMU）的处理器、控制器设计的，并做了较大幅度的精简，所以，在标准 Linux 上可以使用的一些函数和系统调用在 uClinux 上有可能就行不通了。其差异主要体现在以下几个方面：

◆ 没有内存保护：

导致的结果是：即使由无特权的进程来调用一个无效指针，也会触发一个地址错误，并潜在地引起程序崩溃，甚至导致系统的挂起。显然，在这样

的系统上运行的代码必须仔细编程，并深入测试来确保健壮性和安全。

对于普通的 Linux 来说，需要运行不同的用户程序，如果没有内存保护将大大降低系统的安全性和可靠性；然而对于嵌入式 uClinux 系统而言，由于所运行的程序往往是在出厂前已经固化的，不存在危害系统安全的程序侵入的隐患，因此只要应用程序经过较完整的测试，出现问题的概率就可以控制在有限的范围内。

◆ 没有虚拟内存

主要导致下面几个后果：

首先，由内核所加载的进程必须能够独立运行，与它们在内存中的位置无关。实现这一目标的第一种办法是一旦程序被加载到 RAM 中，那么程序的基准地址就“固定”下来；另一种办法是产生只使用相对寻址的代码称为“位置无关代码”。uClinux 对这两种模式都支持。

其次，要解决内存分配和释放的问题。非常动态的内存分配会造成内存碎片，并可能耗尽系统的资源。对于使用了动态内存分配的那些应用程序来说，增强健壮性的一种办法是用预分配缓冲区池的办法来取代 malloc()调用。由于 uClinux 中不使用虚拟内存，进出内存的页面交换也没有实现，因为不能保证页面会被加载到 RAM 中的同样位置。在普通计算机上，操作系统允许应用程序使用比物理内存(RAM)更大的内存空间，这往往是通过在硬盘上设立交换分区来实现的。但是，在嵌入式系统中，通常都用 FLASH 存储器来代替硬盘，很难高效地实现内存页面交换的存取，因此，对运行的应用程序都限制其可分配空间不大于系统的 RAM 空间。

◆ 系统接口不同

最大的不同就是没有 fork()和 brk()系统调用。 调用 fork()将复制出进程来创建一个子进程。在 Linux 下，fork()是使用 copy-on-write 页面来实现的。由于没有 MMU，uClinux 不能完整、可靠地复制一个进程，也没有对 copy-on-write 的存取。为了弥补这一缺陷，uClinux 实现了 vfork()，当父进程调用 vfork()来创建子进程时，两个进程共享它们的全部内存空间，包括堆栈。子进程要么代替父进程执行（此时父进程已经进入睡眠状态）直到子进程调用 exitI()退出，要么调用 exec()执行一个新的进程，这个时候将产生可执行文件的加载。即使这个进程只是父进程的拷贝，这个过程也不能避免。当子进程执行 exit()或 exec()后，子进程使用 wakeup 把父进程唤醒，父进程继续往下执行。

注意，多任务并没有受影响。哪些旧式的、广泛使用 fork()的网络后台程序的确是需要修改的。由于子进程运行在和父进程同样的地址空间内，在

一些情况下，也需要修改两个进程的行为。

很多现代的程序依赖子进程来执行基本任务，使得即时在进程负载很重时，系统仍可以保持一种“可交互”的状态，这些程序可能需要实质上的修改来在 uClinux 下完成同样的任务。如果一个关键的应用程序非常依赖这样的结构，那就不得不对它重新编写了。

◆ 应用程序库

uClinux 小型化的另一个做法是重写了应用程序库，相对于越来越大且越来越全的 glibc 库，uClibc 对 libc 做了精简。

uCLinux 对用户程序采用静态链接的形式，这种做法会使应用程序变大，但是基于内存管理的问题，也就是基于没有 MMU 的特性，只能这样做，同时这种做法也更接近于通常嵌入式系统的做法。

uClibc 提供大多数的类 UNIX 的 C 程序调用。如果应用程序需要用到 uClibc 中没有提供的函数，这些函数可以加到 uClibc 中、或者作为一个独立的库、或者加到应用程序上面来进行链接。

除了上述几点重要的区别以外，总的来说，在 uClinux 上的开发和标准 Linux 还是很类似的。通常可以按照下面的步骤去设计和调试：

◆建立基于以太网的开发环境；

◆如果所设计的程序和硬件的关联不大，那么一定要在标准 Linux 上先编译和调试通过。灵活地使用 gcc 和 gdb。

◆将用 GCC 编译好的应用程序，用交叉编译工具（arm-gcc）来编译。遇到库不支持的函数需要自己把函数的实现做成另外一个库供应用程序使用。如果是 uClinux 本身不支持的调用，那么就需要改写代码了。

◆通过网络(nfsmount)运行交叉编译成功的应用程序；

◆如果程序工作初步正常，那么就可以进一步在板子上测试了；否则，需做修改重新编译，尤其要检查与 uClinux 的内存特性有关的代码。

4.2.1.2 AT 指令集

（1）GPRS系统相关指令[15]：

查询IMEI号：

AT+GSN

MC35回应IMEI号，应与包装盒上的相同。

（2）语音通信相关指令：

拨号

ATD 029

MC35回应：

OK

表示拨号成功。

有电话呼入，MC35回应

RING

ATA命令：接听

ATH命令：挂机

（3）SMS相关的指令（GSM07.05）：

AT+CMGC

发出一条短消息命令

AT+CMGD

删除SIM卡内存的短消息

AT+CMGF

选择短消息信息格式：0-PDU;1-文本

AT+CMGL

列出SIM卡中的短消息PDU/text模式：

0/“REC UNREAD”-未读

1/“REC READ”-已读

2/“STO UNSENT”-待发

3/“STO SENT”-已发

4/“ALL”-全部的）

AT+CMGR

读短消息

AT+CMGS

发送短消息

AT+CMGW

向SIM内存中写入待发的短消息

AT+CMSS

从SIM内存中发送短消息

AT+CNMI

显示新收到的短消息

AT+CPMS

选择短消息内存

AT+CSCA

短消息中心地址

AT+CSCB

选择蜂窝广播消息

AT+CSMP

设置短消息文本模式参数

AT+CSMS

选择短消息服务

4.2.1.3 对短消息的控制的三种模式

◆Block Mode

◆基于AT命令的PDU Mode

◆基于AT命令的Text Mode

使用Block模式需要手机生产厂家提供驱动支持，目前，PDU Mode 已取代 Block Mode, Text Mode比较简单，易于实现，便于演示，是我们课题用到的一种模式。PDU Mode也很常用，我们会在以后的讨论中给出它的一些规则和定义

4.2.1.4 重要指令详解

在对详细解释如下重要的AT命令前,所使用的参数符号意义如下:

<cr>:回车;

<da>:字符串格式的目的手机号;

<mr>:信息号;

<err>:出错信息号;

<Length>:8进制TP数据单元的长度;

<mt>:根据不同的编码方式,短信采用的不同的存储规则;

<bm>:根据不同的编码方式,蜂窝广播信息采用的不同的存储规则;

<index>:短信序号;

◆AT+CMGS

功能：发送一条短消息。

如果此时MC35T处于Text Mode（即“AT+CMGF?<CR>”返回“1”）

AT+CMGS=<da><cr>//回车

text is entered<crtl-z>

<crtl-z>的ASCII码为1A.

如果短消息发送成功，则返回“OK”，并显示信息号：

+CMGS: <mr>

如果短消息发送失败，则返回如下信息：

+CMS ERROR: <err>

实例：

AT+CMGS="目的手机号 " <cr>

>短信内容 crtl-z ；

如果此时MC35T处于PDU Mode（即“AT+CMGF?<cr>”返回“0”）

AT+CMGS=<length><cr>

PDU is given<crtl-z>

如果短消息发送成功，则返回“OK”，并显示信息号：

+CMGS: <mr>

如果短消息发送失败，则返回如下信息号：

+CMS ERROR: <err>

◆AT+CNMI

功能：当有新的短消息到来时，MC35IT产生提示

该指令的完整语法如下：

AT+CNMI=[<mode>][,<mt>][,<bm>][,<ds>][,<bfr>]

mode - 通知方式：

0 - 不通知TE。

1 - 只在数据线空闲的情况下，通知TE；否则不通知TE。

2 - 通知TE。在数据线被占用的情况下，先缓冲起来，待数据线空闲，再行通知。

3 - 通知TE。在数据线被占用的情况下，通知混合在数据中一起传输。

mt - 消息储存或直接转发到TE:

0 - 储存到默认的内存位置

1 - 储存到默认的内存位置，并且向TE发出通知(包括class 3)

2 - 对于class 2，储存到SIM卡，并且向TE发出通知；对于其它class，直接将消息转发到 TE

3 - 对于class 3，直接将消息转发到 TE；对于其它class，同mt=1

bm、ds、dfr一般不用设置。

当设置mode=2，mt=1时，如果有新的短消息来到，则MC35T将自动返回下列提示：

+CMTI: “SM”, <index>

此时读出<index>，然后用“AT+CMGR”指令即可读出短消息内容。

4.2.1.5 PDU数据格式分析

由于PDU[15]格式比较复杂，所以我们借助实例来分析。例如，我们要将字符“你好”发送到目的地“13880792249”

PDU字符串为：

08 91 683188702942F9 11 00 0D 91 683188702942F9 00 08 00 04 4F60597D

（1）08—短信息中心地址长度。指（91）+（683108200805F0）的长度。

（2）91—短信息中心号码类型。91是TON/NPI遵守International/E.164标准，指在号码前需加'+'号；此外还有其它数值，但91最常用。

（3）683108200805F0—短信息中心号码（成都）。由于每两位数字位置进行了交换，所以实际号码应为：8613800731500（字母F是指长度减1）。这需要根据不同的地域作相应的修改。

（1）、（2）、（3）通称短消息中心地址（Address of the SMSC）。

（4）11(&h)—文件头字节。

11&h=00010001&b

表4-1

BIT	7	6	5	4	3	2	1	0
NAME	TP-RP	TP-UDHI	TP-SPR	TP-VFP	TP-RD	TP-MTI		
VALUE	0	0	0	1	0	0	0	1

各信息为具体的设置及含义如下：

应答路径—TP-RP（TP-Reply-Path）：0—不设置；1—设置

用户数据头标识—TP-UDHL（TP-User-Data-Header-Indicator）：0—不含任何头信息；1—含头信息

状态报告要求—TP-SPR（TP-Status-Report-Request）：0—需要报告；1—不需要报告

有效期格式—TP-VPF（TP-Validity-Period-Format）：00—不提供（Not present）；10—整型（标准）；01—预留；11—提供8位字节的一半（Semi-Octet Represented）

拒绝复制—TP-RD（TP-Reject-Duplicates）：0—接受复制；1—拒绝复制

信息类型提示—TP-MTI(TP-Message-Type-Indicator)：00—读出(Deliver)；01—提交（Submit）

（5）00—信息类型（TP-Message-Reference）

（6）0D—被叫号码长度。

（7）91—被叫号码类型（同(2)）。

（8）3176378290F9—被叫号码，同样也经过了位置的交换，实际号码为“13677328099”。

（6）、（7）、（8）通称目的地址（TP-Destination-Address）。

（9）00—协议标识TP-PID（TP-Protocol-Identifier）

BIT No. 7 6 5 4 3 2 1 0 ，每一位的具体含义如下：

Bit 7与Bit 6： 00—如下面定义的分配Bit 0—Bit 5；01—参见GSM03.40协议标识完全定义；10—预留；11—为服务中心(SC)特殊用途分配Bit 0—Bit 5。一般将这两位置为00。

Bit 5：0—不使用远程网络，只是短消息设备之间的协议；1—使用远程网络。

Bit 0-Bit 4：00000—隐含；00001—电传；00010—第3分组电话传真；00100—语音；00101—欧洲无线信息系统（ERMES）；00110—国内系统；10001—任何基于X.400的公用信息处理系统；10010—Email。

（10）08—数据编码方案TP-DCS（TP-Data-Coding-Scheme）

BIT No. 7 6 5 4 3 2 1 0，其每位的具体含义如下：

Bit 7与Bit 6 :一般设置为00；

Bit 5：0—文本未压缩，1—文本用GSM标准压缩算法压缩；

Bit 4：0—表示Bit 1、Bit 0为保留位，不含信息类型信息；1—表示Bit 1、Bit 0含有信息类型信息；

Bit 3与Bit 2：00—默认的字母表，01—8bit，10—USC2（16bit），11—预留；

Bit 1与Bit 0：00—Class 0，01—Class 1，10—Class 2（SIM卡特定信息），11—Class 3。

（11）00—有效期TP-VP（TP-Valid-Period）

表4-2

VP value	相应的有效期
00 to 8F	(VP+1)*5 分钟
90 to A7	12小时+(VP-143)*30分钟
A8 to C4	(VP-166)*1天
C5 to FF	(VP-192)*1 周

（12）04—用户数据长度TP-UDL（TP-User-Data-Length）

（13）4F60597D—用户数据TP-UD（TP-User-Data）“你好”（unicode编码）

所以PDU格式看似复杂，其实可总结为[SCA]、[TPUD]两大部分。即（1）-（3）为一部分，后面的（4）-（13）为一部分。（4）-（12）共有15个字节。在我们以后使用AT+CMGS（PDU模式）输入TPUD长度时带来了方便，只需用15加上用户数据的长度。

4.2.2 短信和语音应用程序功能设计

应用程序要实现的功能如下：

★ 短信的发送；

★ 短信的接收；

★ 短信的管理：读取，删除；

★ 语音呼叫；

可扩展的功能如下：

★ 接入 internet，实现网页浏览；

★ LCD 显示；

★ 触摸屏；

4.2.3. 短信和语音应用程序的整体设计

由于实验开发板与GPRS终端使用串口[16]进行数据的交换和实现控制功能，所以该应用程序要实现短信的发送，接受，读取，删除等短信相关的管理功能，以及和语音服务有关的拨号，接听等功能都需要和串口发生联系。好在使用了操作系统，底层对串口怎么操作我们不需要关心，而只需要调用系统提供的相关系统函数就可以了。根据要实现的功能，可以用一个大的功能模块实现一个功能，再由一个主函数根据要执行的任务调用管理各个模块，完成操作。程序采用模块化的设计，使程序便于修改升级，便于后续功能的扩展。其整体的结构如图4-2。

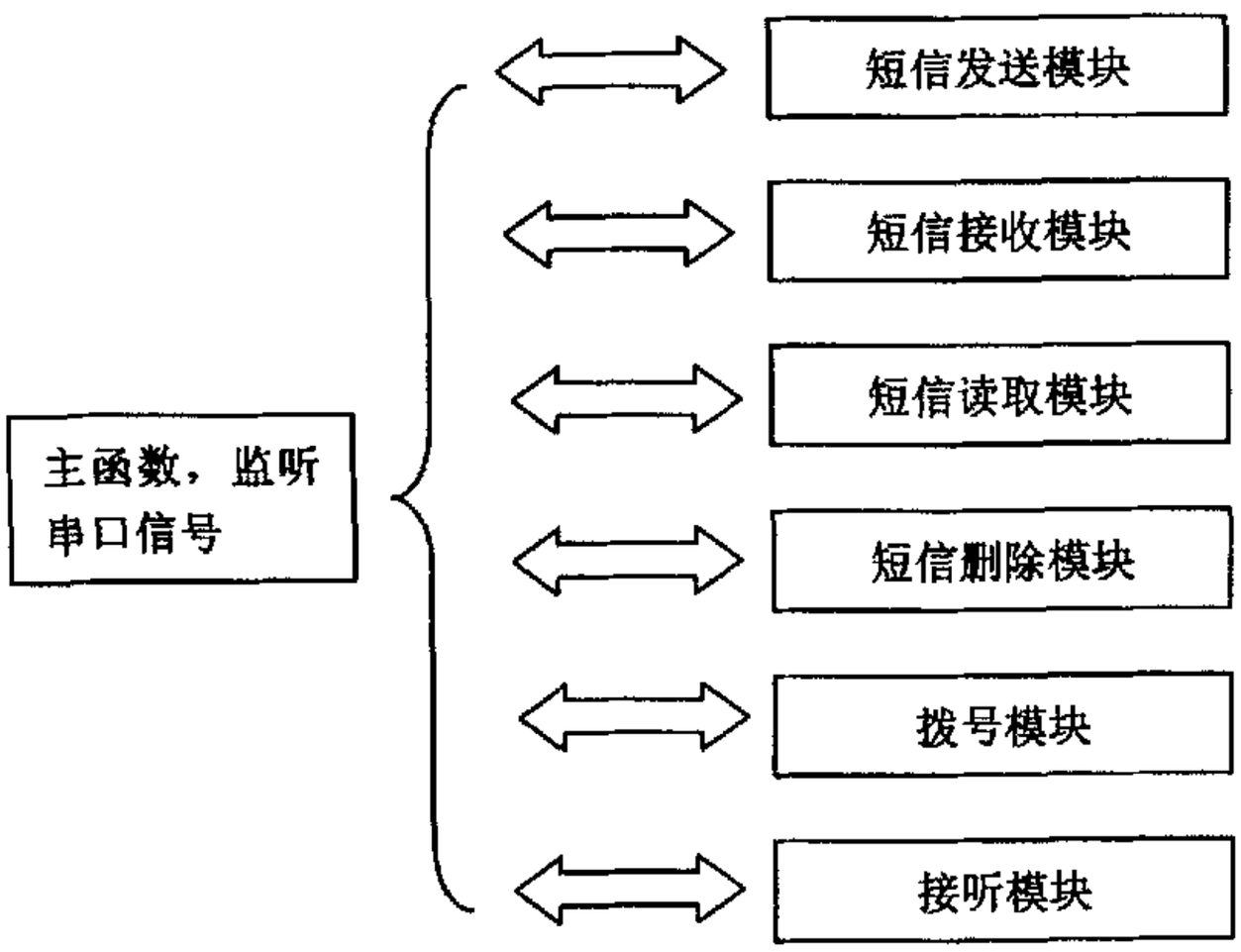


图4-2 主程序框图

4.2.4 各功能模块分析

4.2.4.1 短信发送功能

为实现该功能又会调用到其它更小的功能模块，其功能设计框图及相互关系如图4-3所示：

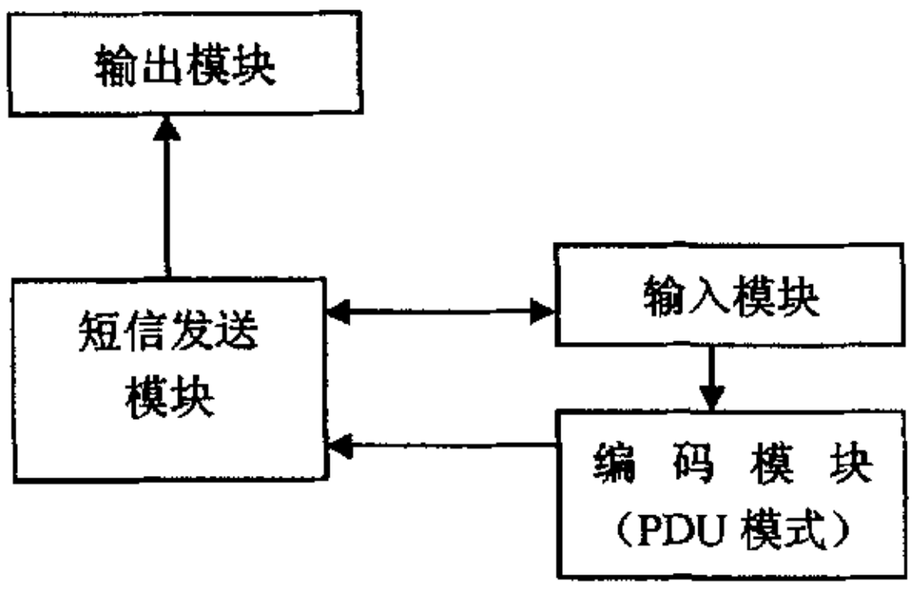


图4-3 短信发生功能框图

其中输入、输出模块负责从两个串口读取信息及向串口打印信息。而编码模块在使用PDU模式的短信时会用到，之所以再此设计这样一个功能模块是为了方便以后对程序功能扩展。而如果用Text模式，其短信格式相对简单容易实现，则可以直接在发送模块中编码实现。

4.2.4.2 短信接收及读取功能

之所以将这两个功能和在一起是因为它们用到的函数基本相同，接收主要靠调用短信读取功能模块实现，所以我们将它们放在了一起来讲。

其功能框图如图4-4

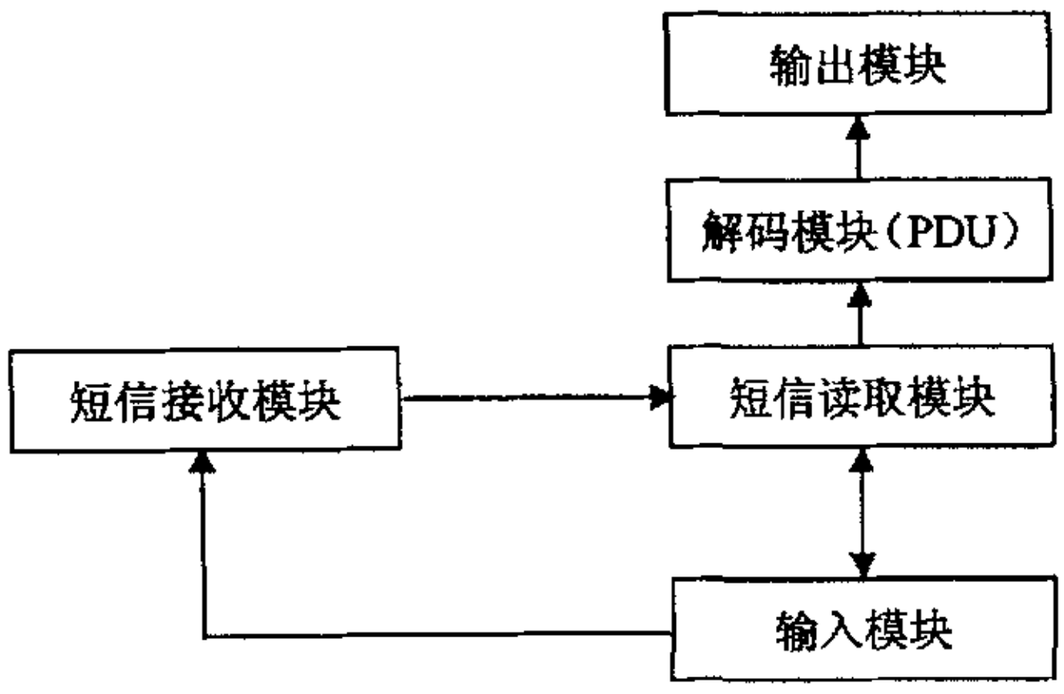


图4-4 短信接收功能框图

与接收模块相似，解码模块只有在使用PDU模式时才会被调用。

4.2.4.3 短信删除及语音服务相关功能模块

把这几个功能模块放在一起来分析是因为它们的AT指令及操作程序相对简单，使用到的子模块也较少。其功能模块如图4-5：

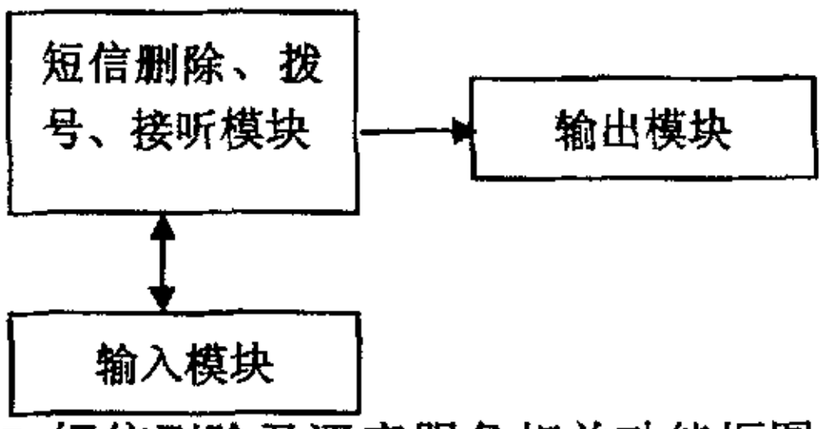


图4-5 短信删除及语音服务相关功能框图

4.3 短信和语音应用程序设计实验的具体实施

4.3.1 具体函数及相互关系分析

上述分析中，每个子模块都可以用自己编写的函数来实现。

4.3.1.1 使用到的函数

每个功能模块都是由几个不同的函数来实现，它们有的是每一个功能模块都会用到的“通用”函数，有的则是“专用”函数。具体如表7-1所示。通过列表分析，我们已经对要编写的函数及相对应的AT指令一目了然了。但是函数要互相配合工作才能完成一个大的整体功能。所以，我们还需要分析函数间的关系及各个函数的接口。

表4-3 应用程序功能模块所使用到的函数

功能模块	用到的函数	用到的AT指令
短信发送模块	readcom（）；从串口读数据 SMS_encode（）；短信PDU格式编码 SMS_send（）；发送短信 writecom（）；写串口	AT+CMGS
短信接收模块	readcom（）；从串口读数据 SMS_decode（）；短信PDU格式解码 SMS_read（）；读取短信 writecom（）；写串口	AT+CMGR
短信读取模块	readcom（）；从串口读数据 SMS_decode（）；短信PDU格式解码 SMS_read（）；读取短信 writecom（）；写串口	AT+CMGR
短信删除模块	readcom（）；从串口读数据 SMS_delet（）；删除短信	AT+CMGD

	writecom（）；写串口	
拨号	readcom（）；从串口读数据 dial（）；拨号 writecom（）；写串口	ATD
接听	readcom（）；从串口读数据 answer（）；接听 hong（）；挂机 writecom（）；写串口	ATA ATH

4.3.1.2 函数功能及接口分析

这里我们将针对每个功能模块来具体分析。并图形化。

------▶ ：函数接口

——▶ ：数据流向

短信发送功能模块：

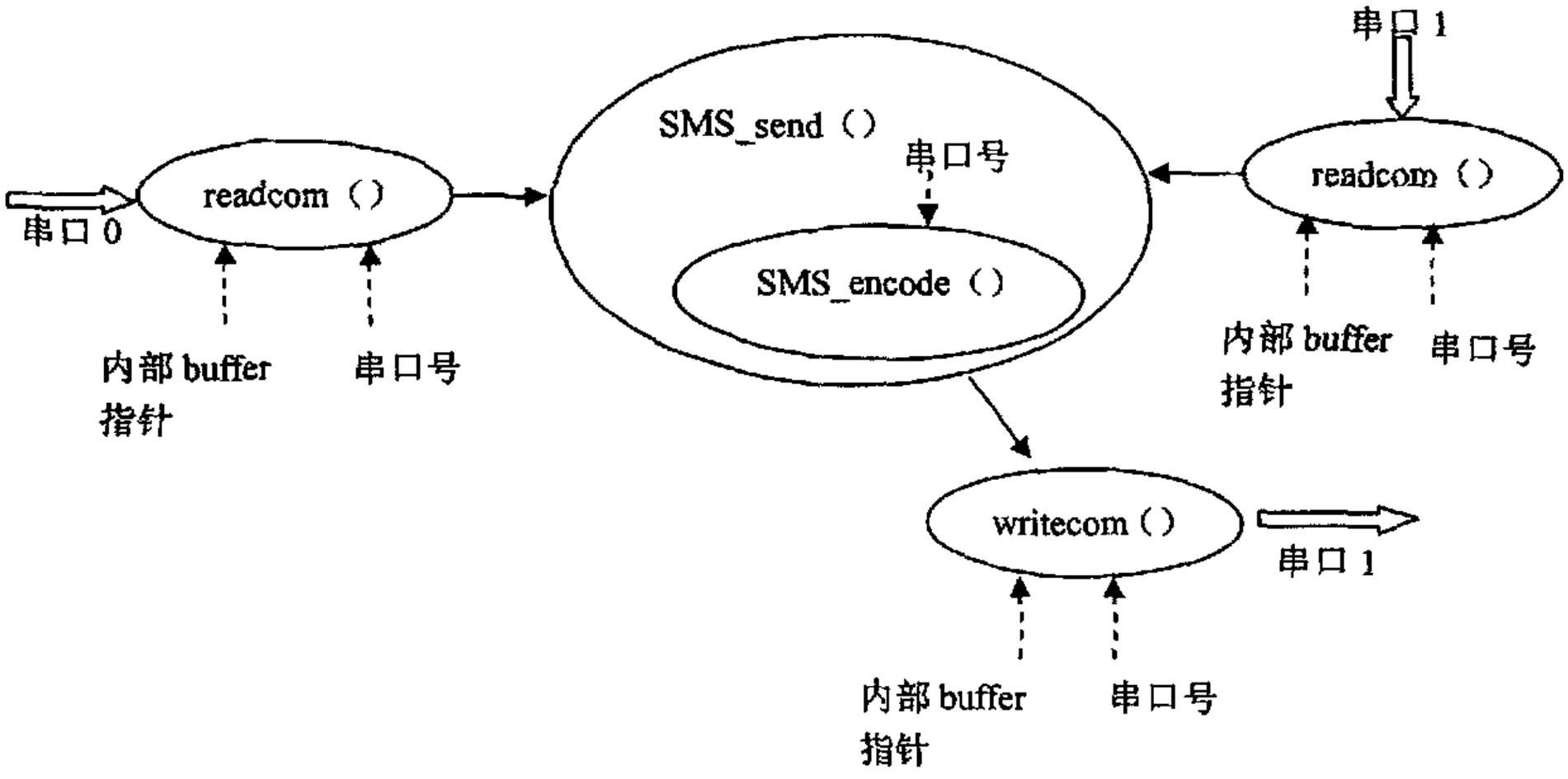


图4-6 短信发送模块

与4.2.4.2节的设计相似，其中SMS_encode（）函数是个可选的函数，在短信采用PDU模式时才会用到，这里之所以提出来，是为以后对功能进行扩展打下基础。

短信接收功能模块：

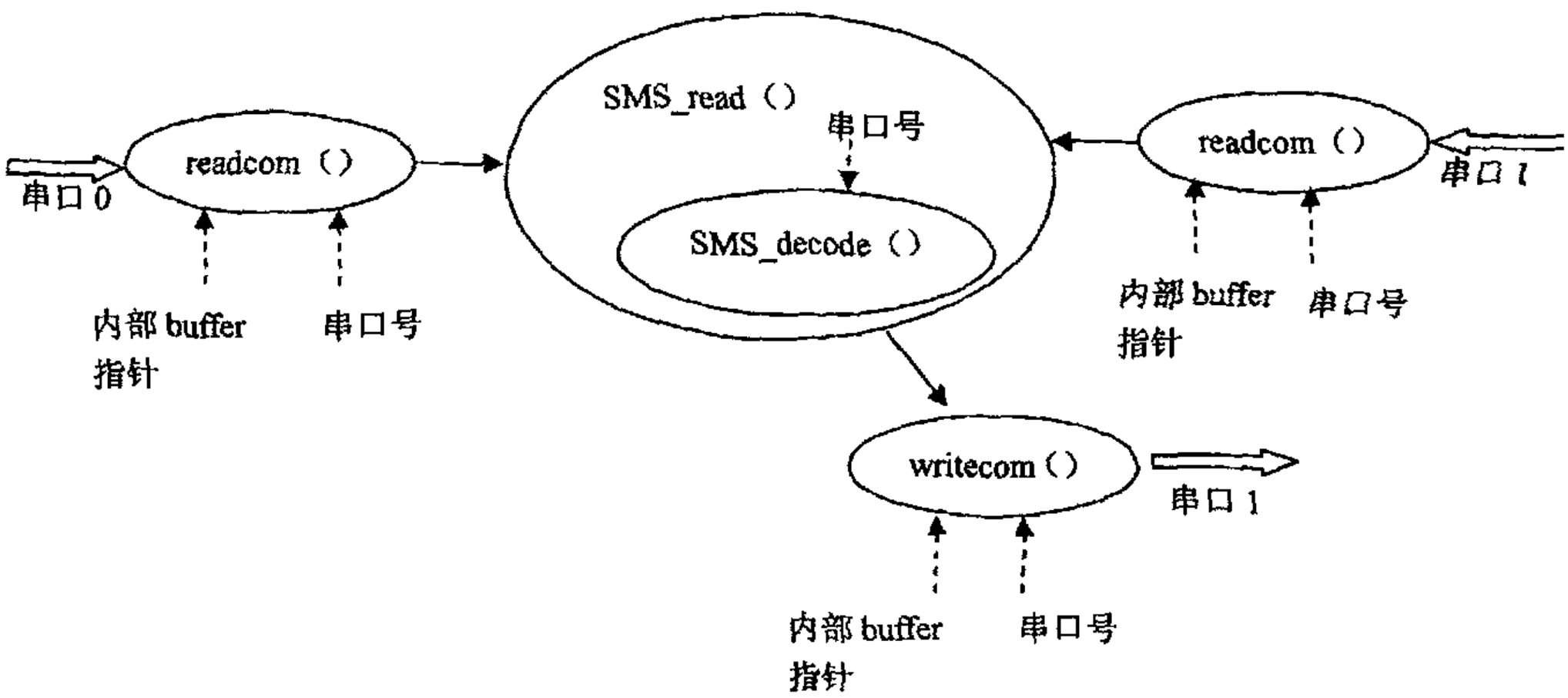


图4-7 短信接收模块

短信读取、删除及语音服务相关的功能模块由于功能结构相同，所以可用下图统一表示。

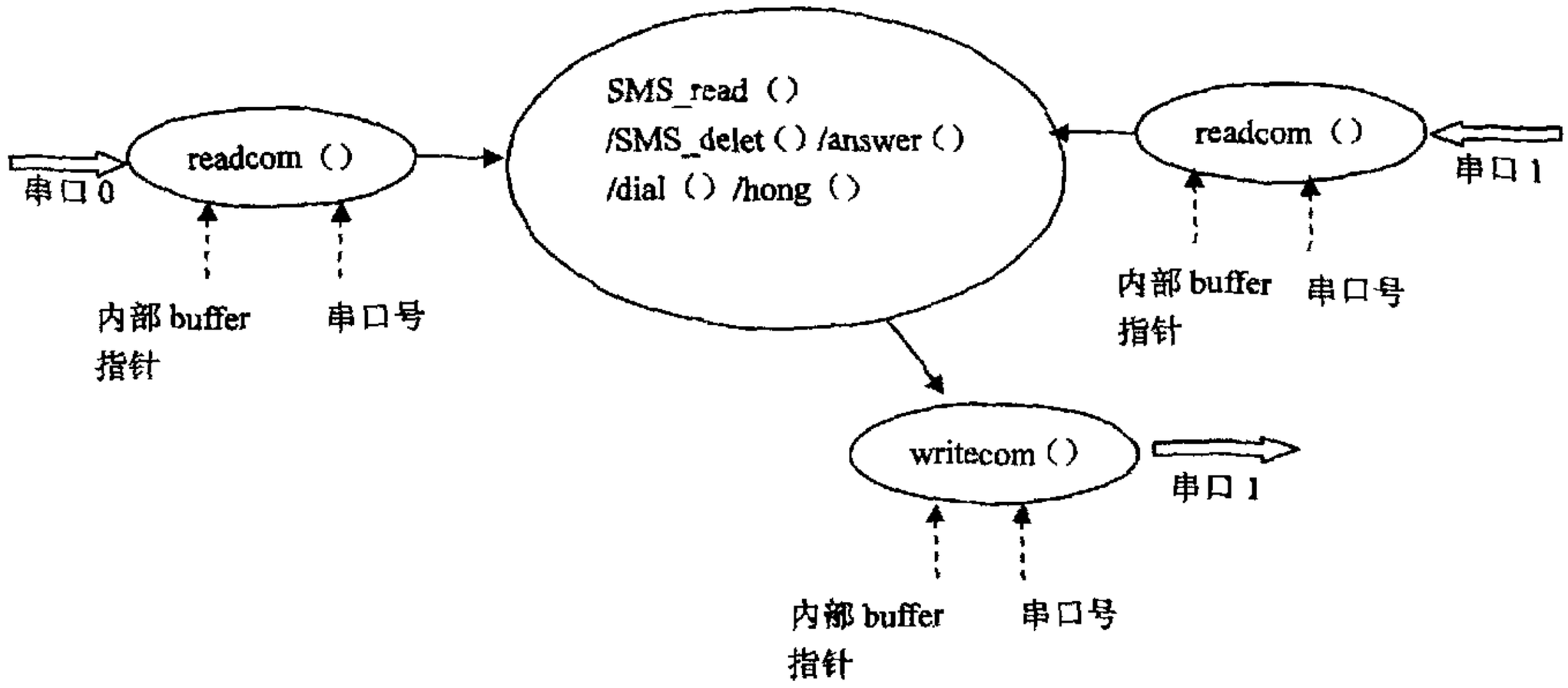


图4-8 短信读取、删除及语音服务相关模块

这样一来，函数间数据的流向以及接口就清楚了，我们就可以开始着手对具体的函数代码进行编写.

4.3.2 代码编写

4.3.2.1 程序流程图

在对每个具体的功能模块编写程序时，程序的流程图能帮助我们提高编程的

效率,使程序的条理更加清晰,结构也更合理.所以各功能模块的流程图如下:

短信发送功能:

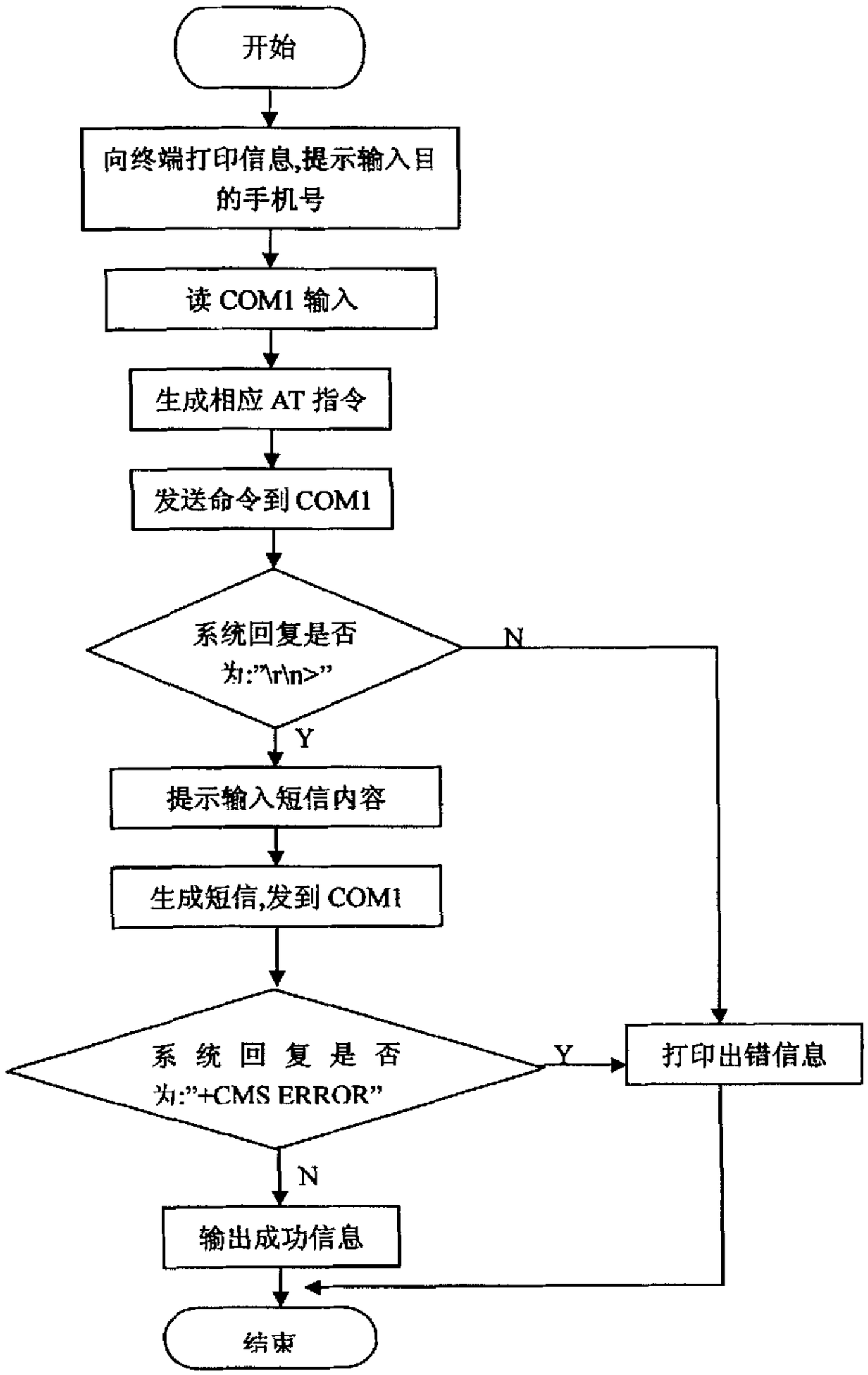


图4-9 短信发送流程图

短信接收\读取功能:

短信的接收功能模块调用了短信读取功能模块,前者在具体实现上其实就是后者,不同之处就是它们在主函数数中执行的条件不一样.所以将它们合并在一

起讨论.

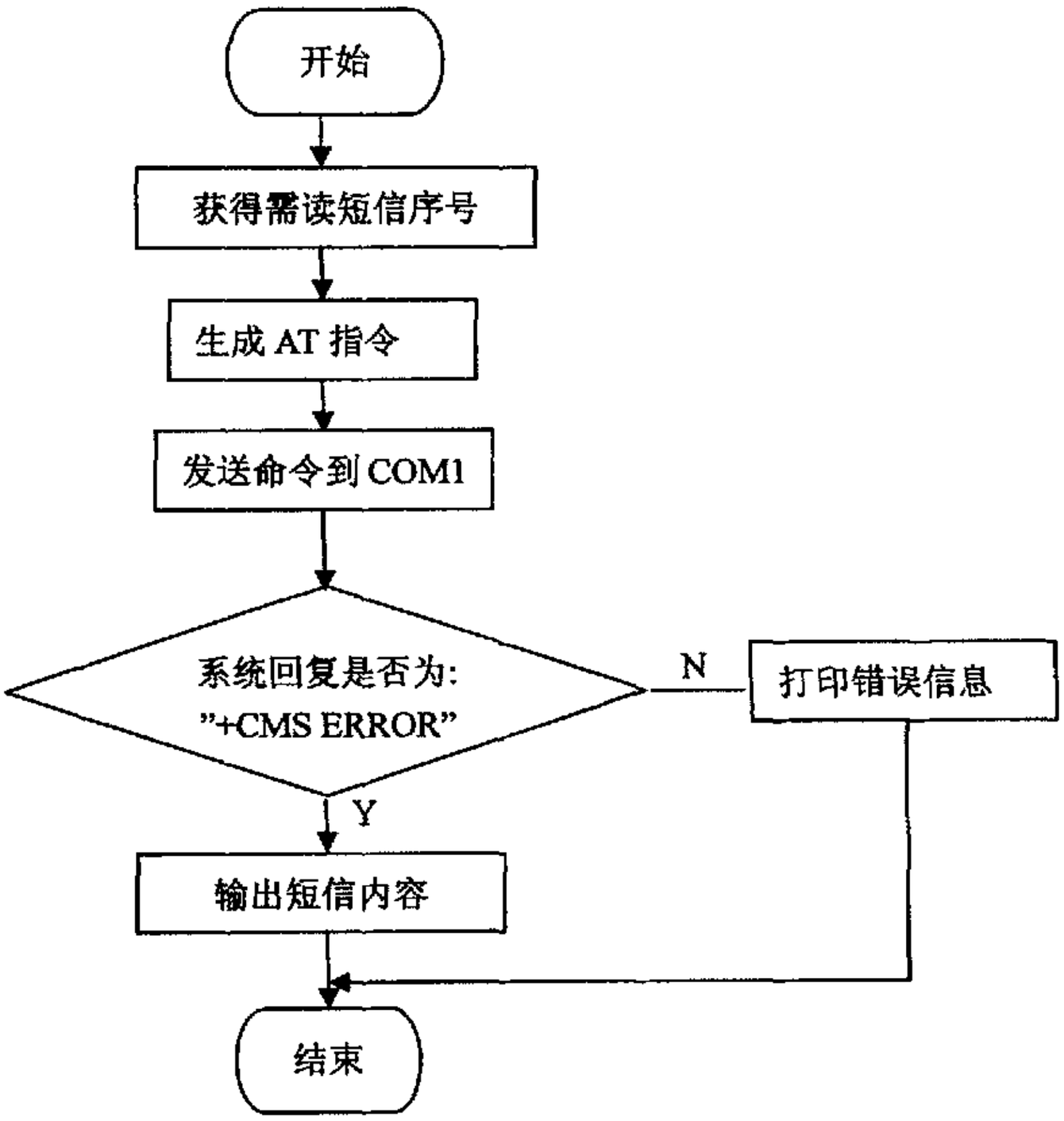


图4-10 短信接收/读取流程图

短信删除模块:

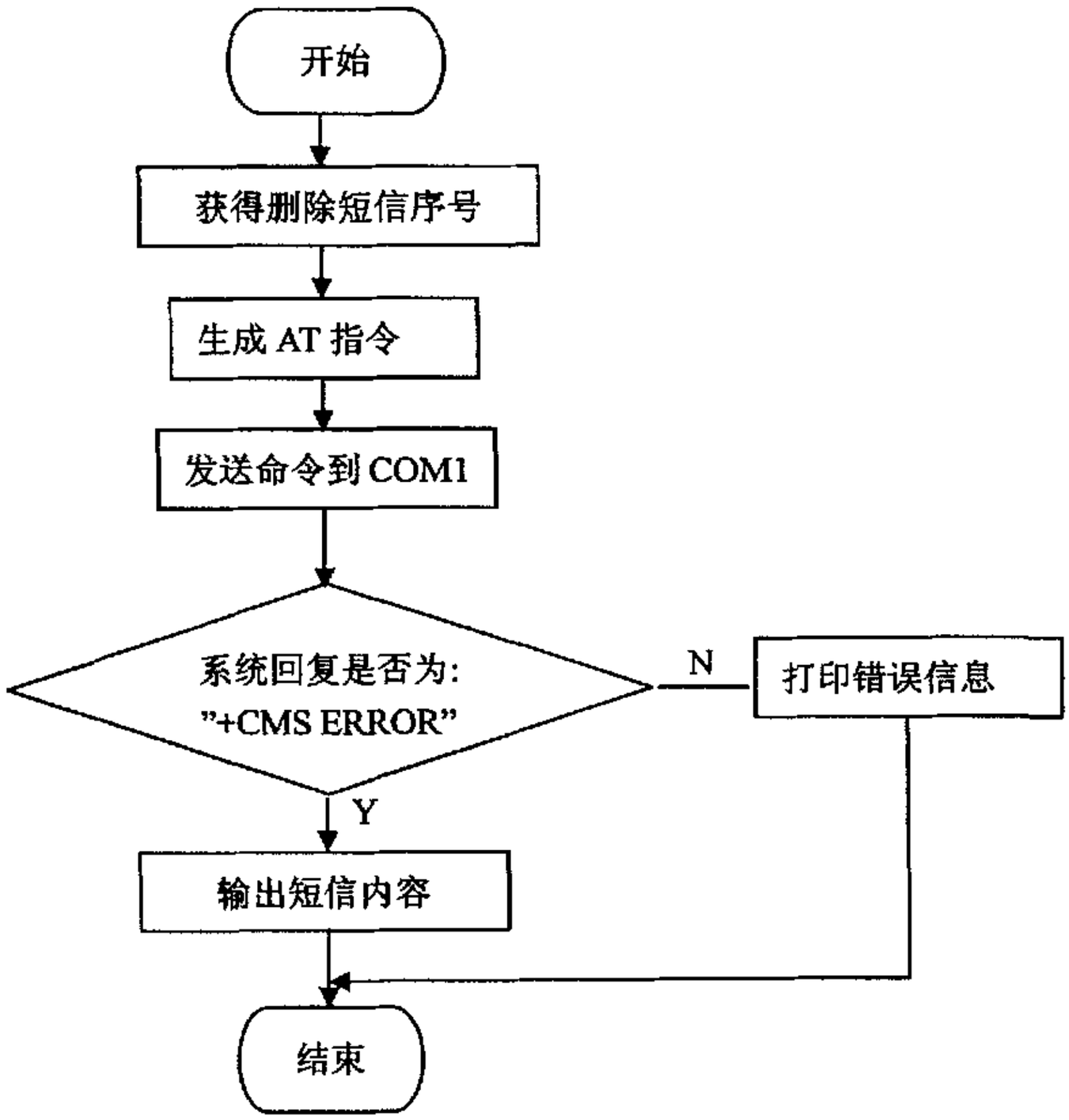


图4-11 短信删除流程图

由于语音服务的相关应用程序的处理过程简单,这里就不给出其具体的程序实现流程图了。

上述就是本应用程序几个主要功能模块的流程图,整个的处理过程已经变得非常的清晰,剩下的工作就是按照流程图的过程及数据流向,以具体的代码来实现整个功能。

4.3.2.2 具体代码编写

根据以上的分析,我们发现readcom()和writecom()函数使用得最为频繁,是"通用"函数.所以我们的代码编写从这两个函数如手,但是由于是对串口进行操作,需要对串口根据应用程序的要求进行配置和初始化[11][12][13][14],其代码和注释如下:

```
int initcom(int n)
{
int fd;
struct termios newtio;
if(n= =0)
{
```

```
fd=open("/dev/ttyS0",O_RDWR|O_NOCTTY|O_NDELAY);/*可读写,该程序不
                                                是此端口的控制终端,
                                                不关注DCD信号线状态
                                                */

if(fd<0)
  {
   printf("Fail to open com0!\n");
   return -1;
   }
   else
   newtio.c_cflag=B115200|CS8|CLOCAL|CREAD;/*波特率115200,8位数据,1
                                            位停止,无奇偶校验,使能接
                                            收并设置本地状态*/
}
if(n==1)
{
  fd=open("/dev/ttyS1",O_RDWR|O_NOCTTY|O_NDELAY);
   if(fd<0)
    {
    printf("Fail to open com1!\n");
    return -1;
    }
    else
    newtio.c_cflag=B19200|CS8|CLOCAL|CREAD;
}
    newtio.c_iflag=IGNPAR|ICRNL;/*忽略奇偶错误*/
    newtio.c_oflag=0;
newtio.c_lflag=ICANON;/*使能规范输入*/
tcflush(fd,TCIFLUSH);/*清空输入/输出队列*/
fcntl(fd,F_SETFL,0);/*如果输入缓冲区中无字符则阻塞该调用,直到有字
                      符输入*/
tcsetattr(fd,TCSANOW,&newtio);/*设置立即生效*/
close(fd);
```

```
    return 0;
}
```

termios[17][18]结构中定义了如波特率、字符大小等参数。fd则是文件描述符，接收open系统调用的返回值，为正则表示open操作成功。Close也是一个系统调用，有来关闭进行文件操作的对象。

在设置好了串口之后，就可以进行对串口的读写操作了。相应的代码段如下：

```
int writecom(char *buffer, int n, int datalen)   /*写串口函数,buffer为
                                                 输出缓冲区,n为串口
                                                 号,datalen为写入长度
                                                 */
{
 int len=0;
 int fd;
  if(n= =0)
    fd=open("/dev/ttyS0",O_RDWR|O_NOCTTY|O_NDELAY);
  else
    fd=open("/dev/ttyS1",O_RDWR|O_NOCTTY|O_NDELAY);
len=write(fd,buffer,datalen);/*将buffer中的数据写入串口,长读为
                              buffer的长度.返回实际写入的长度*/
if(len<0)
 printf("Write com failed!\n");
close(fd);
return (len);
}

int readcom(char *buffer, int n, int datalen)    /*读串口函数, buffer
                                                 为输入缓冲区,n为串
                                                 口号,datalen为读入
                                                 长度*/
{
int len=0;
int fd;
  if(n= =0)
  fd=open("/dev/ttyS0",O_RDWR|O_NOCTTY|O_NDELAY);
```

```
    else
    fd=open("/dev/ttyS1",O_RDWR|O_NOCTTY|O_NDELAY);
len=read(fd,buffer,datalen);/*将串口的数据读入到buffer,读入长度为
                                                buffer的大小.返回实际读入的长度
                                                */
    if(len<0)
      printf("Read com failed!\n");
close(fd);
return (len);
}

void SMS_send()/*短信发送程序*/
{
int len=0;
int phone;
char ans[12];/*应答串*/
char content[128];/*短信内容串*/
char cmd[16];/*命令串*/
printf("The mobilephone NO.:\n");
readcom(ans,0,12);
phone=char2int(ans);/*将字符串变为整数*/
sprintf(cmd,"AT+CMGS=%d\r",phone);/*生成AT指令*/
writecom(cmd,1,16);/*发送AT指令*/
len=readcom(ans,1,12);
  if(len==4&&strncmp(ans,"\r\n>",4)==0) /*系统回复是否继续输入短
                                          信的内容*/
    {
     printf("The message content:\n");
readcom(content,0,128));
strcat(content,"\x01a");/*信息以crtl-z结束*/
writecom(content,1,128);/*发送短信内容到串口*/
len=readcom(ans,1,12);
if(len>0&&strncmp(ans,"+CMS ERROR",10)!=0)/*系统回复，是否发送
                                            成功*/
```

```
    printf("Send message success!\n");
  else
    printf("Send message failed!\n");
 }
else
   printf("error!\n");
}
void SMS_read(int index)/*读短信函数*/
{
char cmd[16];/*命令段*/
char content[128];/*短信内容段*/
int len=0;
sprintf(cmd,"AT+CMGR=%d\r",index);/*生成AT指令*/
writecom(cmd,1,16);/*发送指令*/;
len=readcom(content,1,128);
 if(len<0)
 printf("Read message failed!\n");
printf("%s\n",content);/*输出短信内容*/
}

void SMS_delet()/*删除短信函数*/
{
char cmd[16];/*命令串*/
char ans[12];/*应答串*/
int len=0;
int index;
printf("The index of SMS:\n");
readcom(ans,0,12);
index=char2int(ans);/*将字符串变为整型*/
sprintf(cmd,"AT+CMGD=%d\r",index);/*生产AT指令*/
writecom(cmd,1,16);/*发送命令*/
len=readcom(ans,1,12);
 if(len>0 && strncmp(ans,"+CMS ERROR",10)!=0)/*判断操作是否成功*/
  printf("Delet success!\n");
```

```
else
  printf("Delet failed!\n");
}
```

函数char2int () 代码如下:

```
int char2int(char *ans)
{
char *hd;
int len=0;
int i,real;
hd=ans;
  while(*hd!=' \0' )/*统计字符串的长度*/
  {
  len+=1;
  hd++;
  }
  for(i=0;i<len;i++)
  {
  real+=(ans[i]-48)*pow(len-i-1);/*字符相应的ASCII值与字符0对应的值
                                  相减，最终得十进制数值*/
  }
}
int pow(int n)/*以10为底n为幂的幂函数*/
{
int res=1;
int l;
  for(l=0,l<n,l++)
  res*=10;
return (res);
}
```

由于语音服务相关的程序代码和短信删除功能模块的代码除发送的AT指令不同外基本相同，这里就不再列出。

至此该系统的各个功能函数都已经编写完毕，主函数只需要监听串口的不同输入来执行相关的功能函数即可。功能函数执行完成后则返回主函数，或等待或执行其他的功能。

最后先使用gcc进行编译，在编译时使用-g选项在文件种生成调试信息，以便使用gdb对程序进行调试。调试好后再使用arm-elf-gcc编译连接，生成最终开发板可运行的文件。

4.3.2.3 应用程序的调试

在调试应用程序的时候，如果每做一点修改都要重新编译内核并烧写入FLASH运行，工作量是巨大的。而uClinux操作系统本身有强大的网络功能，我们的开发板也提供了网口。所以通过局域网在目标系统和主机间传输文件，进行调试将是一个不错的选择[2][4][19]。

目前uClinux内核采用ROMFS为其根文件系统，当目标系统的uClinux启动运行后，其目录大多数是建立在FLASH中的，因而是不可写的。只有/var、/tmp等少数几个目录建立在SDRAM中，可读写的。所以我们可以使用FTP将我们的应用程序传输到上面提到的两个目录中，在开发板上运行，调试。

首先我们要先将主机的IP设为：192.168.3.22。

接着启动开发板上的uClinux操作系统，通过串口工具（DNW）输入以下命令：

ifconfig eth0 192.168.3.100

ifconfig命令用于显示几设置目标系统的网卡配置。设置好目标系统的IP后，使用如下命令测试与主机的连接。

ping 192.168.3.22

得到主机系统的应答信息，证明已和主机建立了连接。接着使用如下命令进入目标系统的可写目录并登陆到主机。

cd tmp

ftp 192.168.3.22

此时输入宿主机的用户名及密码，便和宿主机建立了FTP连接。为传输已编译好的二进制可执行文件（SMS），输入命令：

ftp>binary

200 Type set to I

接着传输文件并退出FTP

ftp>get SMS

ftp>bye

至此可执行文件SMS已传输到目录/tmp下，但文件的可执行属性未被设置，通过命令：

chmod 755 SMS

修改文件的可执行属性.

然后执行

./SMS

即可在目标系统上运行调试程序.大大节省了时间.

一切调试完成后,再从新编译内核,将应用程序放入FLASH存储器中,完成整个短信和语音应用程序设计实验.

4.3.3 实验结论

在宿主机终端上以输入命令行的方式，完成了短信收发、阅读、删除及语音通信功能，实现了宿主机与实验开发板，实验开发板与 GPRS 终端的数据交换与处理。熟练运用 AT 指令控制 GPRS 终端，同时深入了解并掌握了 Linux 环境下从程序编写，交叉编译连接到调试的一整套应用程序的开发方法。

4.3.4 扩展实验项目

开发板配备了LCD接口,通过移植miniGUI,及在uClinux下填加LCD驱动程序,配合网口及uClinux内建的TCP/IP协议栈可以开发与internet相关的实验项目。如果使用GPRS模块，则可以通过编写uClinux下的ppp链接脚本文件，使实验系统自动通过GPRS网接入internet，实现无线接入的相关实验项目。另外还可以进行如USB 驱动程序开发，嵌入式系统的存储扩展等实验。

第五章 结论

目前对嵌入式系统和无线通信的研究和应用在国内外发展地如火如荼，但在大学校园中，将两者相结合的实验项目很少。为了改变这种状况，适应电子科技发展的潮流，本文提出了一套基于ARM的嵌入式无线通信实验方案。该方案包括从硬件选型，嵌入式实验平台的搭建到嵌入式操作系统下应用程序的编写的一整套内容。在具体实验方案的设计上突出了开放性和创新性的特点，针对业务能力及动手能力强的学生设计了连贯的3个实验内容：Boot Loader的设计，uClinux内核及ROMFS根文件系统映象文件的制作，短消息及语音通信应用程序的编写。并且每个实验项目都提供充分的理论依据或是准备知识，但并不给出具体的实现方法，换之给了实验参与者更大的发挥空间，使对能力的培养不只局限于技术层面而是延伸到了项目管理与组织能力的培养，较以往的验证型实验很大的进步。

在硬件上充分利用ARM7TDMI微处理器经济实用，支持接口丰富，支持多种操作系统，可扩展性强等优点。搭建了一个能开展各种嵌入式实验的硬件平台，而不仅仅局限于网络相关的应用实验。在无线网络的接入设备上，我们选择GPRS终端接入属于2.5G的GPRS无线网。该硬件平台搭配GPRS终端能在未来比较长的一段时间内适应无线技术和相关业务的发展，完成更多更复杂的嵌入式无线通信实验任务。从而真正使这套实验系统体现出性价比高，适应性强，可扩展能力强的优点。

该方案连贯的3个实验实例：Boot Loader的设计，uClinux内核及ROMFS根文件系统映象文件的制作，短消息及语音通信应用程序的设计中，前两个实验实例尤其是Boot Loader的设计可以说是一切嵌入式实验的准备实验。Boot Loader的设计有很强的硬件相关性，而uClinux内核及ROMFS根文件系统映象文件的制作又涉及到一部分嵌入式操作系统的知识。这两个实验项目的完成，学生能对ARM7TDMI的架构，启动流程，Linux操作系统有深刻的了解，并熟练掌握Thumb指令、C语言编程及在Linux环境下文件编译的方法，为其后运用模块化思想编写应用程序打下坚实的基础。在实验项目的设计上贯穿了层层递近的思路，并亲手完成了整个的实验具体内容的设计，作为参考。

相信随着嵌入式系统的不断发展，将会有更多的实验项目被开发出来。使学生在实验过程中领略科技之美提高实践动手能力的同时也走在科技发展的浪尖。而该套实验系统也将发挥越来越大的作用。

致谢

在整个开放实验项目设计的过程中得到了国家电子技术实验基地主任，朱红副教授的悉心指导和宝贵意见，使开放实验项目从设计思路上发生了根本的转变，实验内容和形式焕然一新。在此表示衷心的感谢。

参考文献

[1].陆力，朱红. 一种开放式实验教学新模式:IDPM 模式研究.实验技术与管理.2004.vol 5

[2].李驹光，聂雪媛，江泽明等.ARM 应用系统开发详解.北京：清华大学出版社.2003.202~282

[3].Sansumg electronic.S3C44B0X RISC Microprocessor Data Sheet.2002

[4].陈章龙，唐志强，涂时亮..嵌入式技术与系统—Intel XScale 结构与开发.北京：北京航空航天大学出版社.2004.94~142

[5].钟章队.GPRS 通用分组无线业务.北京：人民邮电出版社.2001.20~100

[6].李华.现代移动通信新技术 GPRS.广州：华南理工大学出版社.2001.155~210

[7].Siemens mobile.MC35_MC35i_MG_01_v01.01.Migration Guideline MC35->MC35i.2003

[8].Siemens mobile.MC35_MC35i_MG_01_v01.03.Migration from MC35 to MC35i.2003

[9].Siemens mobile.MC35i_HD_V01.02.Hardware Interface Description.2003

[10].Silicon Storage Technology.Inc.S71145-02-000.16Mbit(× 16) Multi-Purpose Flash SST39LF160/SST39VF160 Data Sheet.2002.2

[11].郝玉洁，袁平，常征等.C 语言程序设计.北京：机械工业出版社.2000

[12].赫伯特·希尔特.C 语言大全（王子恢 戴健鹏等译）.北京：电子工业出版社.2001

[13].邹思轶.嵌入式 Linux 设计与应用。北京：清华大学出版社.2002.43~62

[14].刘峥嵘，张智超，许振山等.嵌入式 Linux 应用开发详解.北京：机械工业出版社.2004.192~221

[15].Siemens mobile.MC35i_ATC_V00.01.AT Command Set.2003

[16].李广军，王厚军.实用接口技术.成都：电子科技大学出版社.1997.190~220

[17]Alessandro Rubini，Jonathan Corbet.Linux 设备驱动程序（魏永明，骆刚，姜君 译）.北京：中国电力出版社.2002.65~195

[18].李善平，陈文智等.边学边干—Linux 内核指导.杭州：浙江大学出版社.2002.437~456

[19].Craig Hollabaugh.嵌入式 Linux 硬件、软件与接口（陈雷，钟书毅等译）.北京：电子工业出版社.2003.102~115

附录

附录 1. 责任矩阵

参与者	工作 1	工作 2	工作 3	工作 4	工作 5	...	工作 M
人员 1							
人员 2							
人员 N							

表中使用如下符号填写

R：负责人

P：参与者

T：审查者

整个项目的参与者可以在不同的工作分组中担任不同的角色

附录 2 Boot Loader 及 uClinux 根文件系统实验报告

1.实验题目

Boot Loader 设计及 uClinux 根文件系统制作

2.实验任务及要求

Boot Loader 设计的实验任务：

（1）ARM 各寄存器、堆栈、SDRAM 的初始化

（2）串口、I/O 口、网口等外围设备的初始化

（3）时钟频率的初始化

（4）操作系统的下载，FLASH 与 SDRAM 间数据的搬运，引导进入操作系统

uClinux 根文件系统制作的实验任务：

在Linux环境下完成内核映象文件zimage和根文件系统映象文件romfs.img的制作

要求实验参与者引入项目管理内容，将项目分工，提出进度计划表，并建立责任矩阵，按期评估。

3.实验总体方案设计及框图

由于 Boot Loader 的实现依赖于 CPU 的体系结构，针对不同的 CPU 需要编写不同的代码，如果全部用汇编来写那么代码将不能移植到其它结构不同的 CPU 上。因此我们将 Boot Loader 设计为 stage1 和 stage2 两大部分。依赖于 CPU 体系结构的代码，比如设备初始化代码等，都放在 stage1 中，并且都用汇编语言来实现，以达到短小精悍的目的。而 stage2 则用 C 语言来实现。这样在后续开发中可以不断对功能进行扩展和加强，而且代码会具有更好的可读性和可移植性。

Boot Loader 的 stage1 完成以下功能：

（1）硬件设备初始化。

（2）设置堆栈。

（3）为加载 Boot Loader 的 stage2 准备 RAM 空间。

（4）拷贝 Boot Loader 的 stage2 到 RAM 空间中。

（5）跳转到 stage2 的 C 入口点。

Boot Loader 的 stage2 必须完成以下功能：

（1）初始化本阶段要使用到的硬件设备。

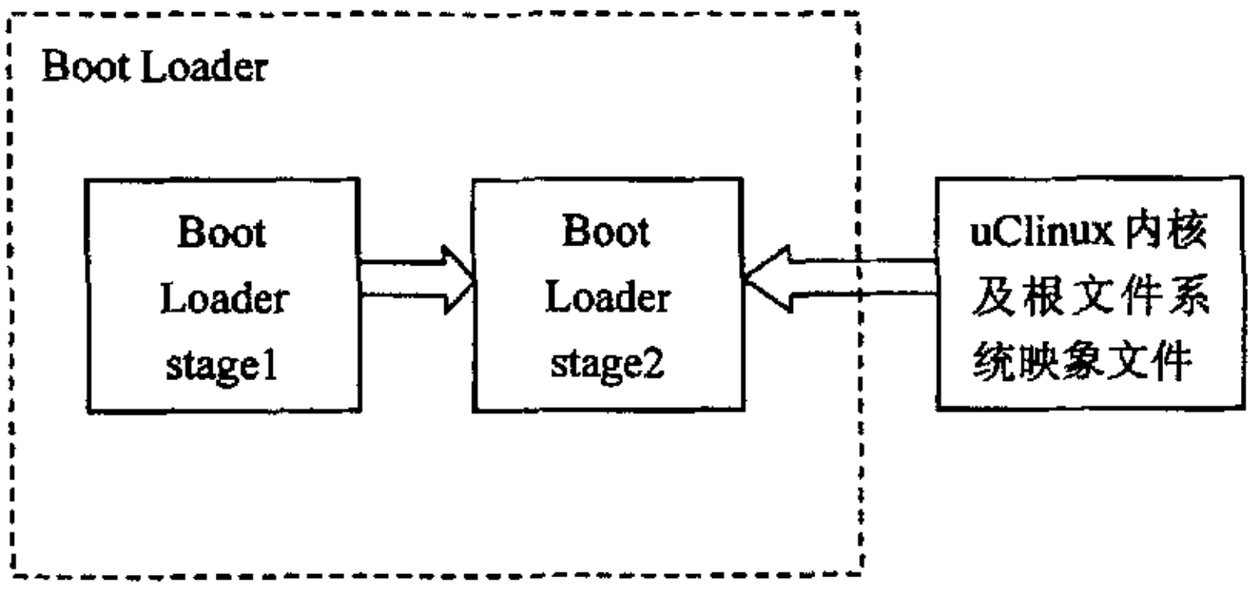
（2）将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中。

（3）为内核设置启动参数。

（4）调用内核。

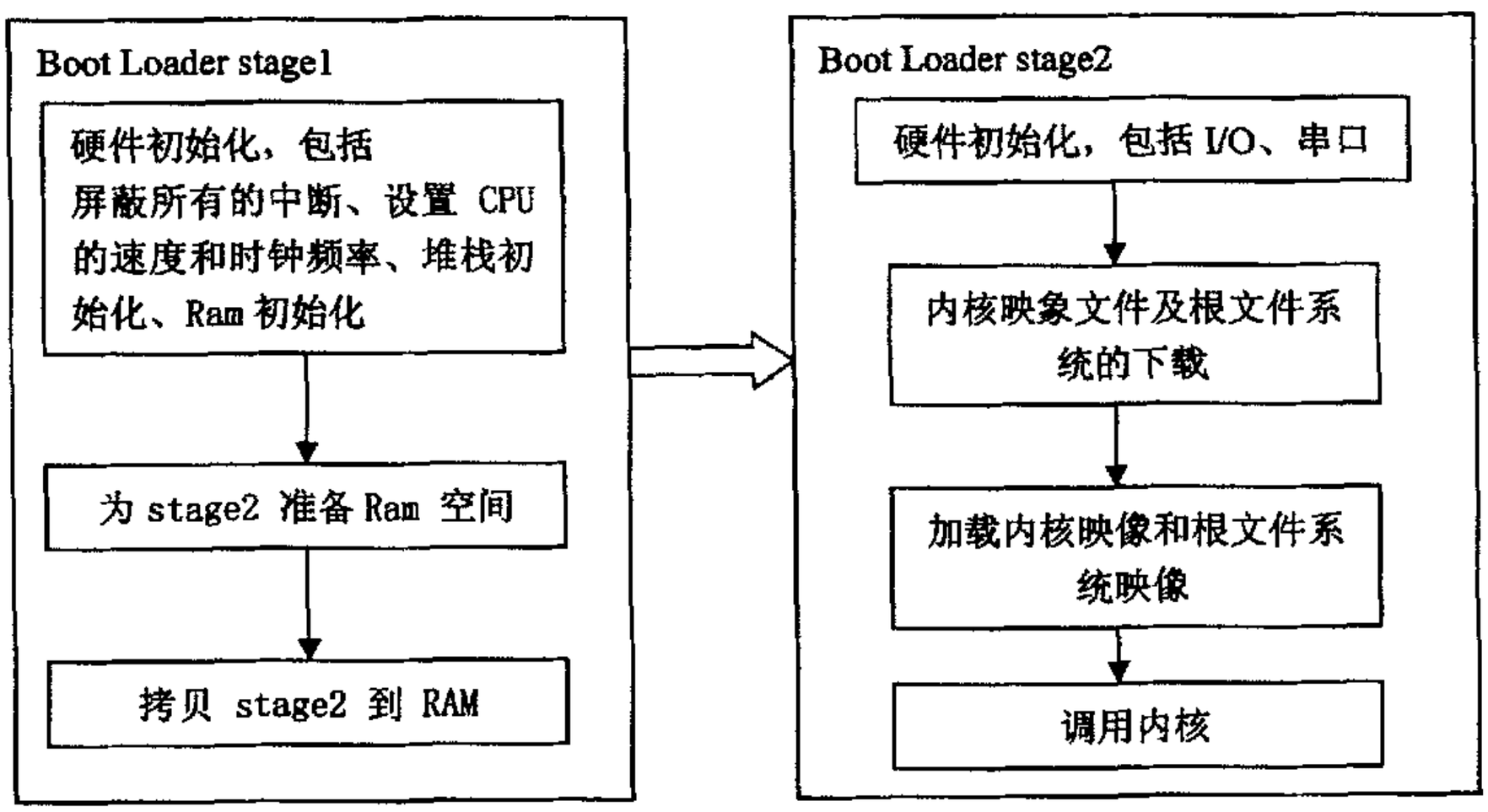
对于 uClinux 根文件系统的制作则使用 uClinux 的 dist 包，在宿主机的 Linux 环境下通过 make menuconfig 命令来完成对根文件系统的配置，编译，连接等一系列工作，生成所需要的根文件系统映象文件。

根据实验设计的功能要求，将实验分为 Boot Loader stage1、Boot Loader stage2、uClinux 根文件系统制作三个大的模块。其总体关系框图如下：



4 具体模块设计

根据 Boot Loader 每个大的功能模块所要完成的任务，将它们根据完成的具体任务的不同细分为更小的任务模块，其框图及流程如下：



5. 调试方法

在 ADS 环境下使用 AXD 调试工具进行调试。

6. 实验结论

根据上述框图及模块的功能设计，完成了全部的实验内容。设计的 Boot Loader 程序成功实现了对实验开发板的启动及初始化，并且完成了对 uClinux

操作系统的引导，使操作系统成功地运行起来。完全达到了实验项目的功能设计要求。

7.实验收获及体会

通过上述两个实验项目，掌握了 ARM7 微处理器的结构和启动的具体过程，同时熟悉了 ARM 编程开发过程及在 uClinux 下文件的编译过程。有了一定的嵌入式开发基础，可以进一步开始应用程序开发的实验。

个人简历

1980 年 3 月 21 日出生于重庆市。

1995 年 9 月—1998 年 7 月 成都七中

1998 年 9 月—2002 年 7 月 电子科技大学 电子工程学院 学士

2002 年 9 月—至今 电子科技大学 信号与信号处理专业 硕士

在学期间参与已正式发表学术论文及获奖情况如下：

廖晓维，朱红. 应用 CPLD 实现 ASI—SPI 接口转换的方法. 中国有线电视. 2004. vol18

2004 年获校优秀研究生二等奖学金。