北京交通大学

硕士学位论文

基于2.6内核的嵌入式LINUX研究

姓名：刘锋

申请学位级别：硕士

专业：电路与系统

指导教师：路勇

20060301

摘 要

LINUX 操作系统凭借着优异的可靠性、良好的可裁减性、完全的代码开源性，在许多产品中得到了大量地使用。现在它已经支持几乎所有主流的 32 位 CPU，新的 2.6 版内核更提供了许多针对嵌入式应用的支持，并且改进了进程调试算法。使得 LINUX 在嵌入式系统中的应用备受关注。但是嵌入式系统之间差别很大，掌上电脑(PDA)、机顶盒、手机、数码相机、数字电视、家用电器、工业控制设备，等等，都是典型的嵌入式应用。和桌面操作系统相比，由于嵌入式应用环境林林总总，难于开发出适应于各种嵌入式应用环境的通用嵌入式操作系统。当前流行的各种嵌入式操作系统，仅仅在某些特定领域获得成功，其原因就在于此。嵌入式 LINUX 操作系统也存在这方面的问题。

本文在论述嵌入式技术特点与应用发展、ARM 微处理器架构以及嵌入式操作系统 LINUX 的技术特点的基础上，以 ARM920T 为内核的 S3C2410X 微处理器为例，详细阐述了基于 ARM 微处理器的嵌入式控制器硬件开发平台的设计和基于嵌入式 LINUX 的软件开发平台的构建。其中主要涉及硬件电路结构、嵌入式操作系统的移植、文件系统的组织创建、网卡驱动程序的编写等方方面面的内容。希望读者通过本文的介绍，能够对嵌入式开发平台以及嵌入式 LINUX 技术有一个更见全面和深刻的认识。

关键字：LINUX 2.6 内核，嵌入式 LINUX，S3C2410
设备驱动，YAFFS 文件系统

Abstract

PDA, TV-cable, cell phone, digital camera, HDTV, industry controller, and so on, are representative embedded applications. Comparing with PC, every embedded system has its own characteristic. It is impossible to build a universal embedded operating system for each embedded applications. So, we must build a "special" embedded operating system for a "special" embedded system. The LINUX operating system, with good reliability, great flexible and open source, has been very popular for many years in our life. The new coming 2.6 kernel has a lot of specialities for embedded applications, and also has some improving in arithmetic of scheduling. All these make LINUX to be a HOT-POINT in embedded system.

The paper's research centre is a model of device driver for 2.6 kernel of Linux. The research platform is based on S3C2410, a chip from ARM920T series, which has a MMU that is needed by modern operating systems such as Linux, Wince. As the development of semiconductor is fast, a CPU is more power, which is the "heart" of an embedded system. So, a study of modern embedded operating system is coming. The main parts of the paper are structure of hardware, porting of embedded Linux, building of file system, and driver of ether card.

Keyword: Linux 2.6 kernels, embedded Linux, S3C2410, device driver, YAFFS files system.

第 1 章 绪论

对于大多数人来说，“嵌入式系统”是一个新鲜的词汇。其实，嵌入式系统早已融入到人们日常生活的方方面面了。手机、手表、电子游戏机、PDA、电视、冰箱等民用电子与通信产品无不与嵌入式系统息息相关。在后 PC 时代的来临，家电、玩具、汽车、3G 手机、数码相机、先进的医疗仪器乃至即将到来的智能家居、智能 office、与其他给电相关的器材设备更是缺少不了嵌入式系统整个核心技术。嵌入式系统的兴起是在 1971 年由 INTEL 公司推出有史以来第一粒微处理器 4004 开始，而微处理器的成功也让接下来的几十年改变了人类的生活，典型的嵌入式系统几乎让人感觉不到她的存在！

1.1 嵌入式系统的定义

嵌入式系统是指用于执行独立功能的专用计算机系统。它由包括微处理器、定时器、微控制器、存储器、传感器等一系列微电子芯片与器件，和嵌入在存储器中的微型操作系统、控制应用软件组成，共同实现诸如实时控制、监视、管理、移动计算、数据处理等各种自动化处理任务。嵌入式系统以应用为中心，以微电子技术、控制技术、计算机技术和通讯技术为基础，强调硬件软件的协同性与整合性，软件与硬件可剪裁，以满足系统对功能、成本、体积和功耗等要求。

简单来说，一个嵌入式系统就是一个计算机硬件和软件的集合体，也许还包括其它一些机械部件，它是为完成某种特定功能而设计的。

嵌入式系统通常是一个大系统或大的电子设备中的一部分，工作在一个与外界发生交互并受到时间约束的环境中，在没有人工干预的情况下进行人工控制。其中，软件用以实现有关功能并使其系统具有适应性和灵活性；硬件(处理器、ASIC、存储器等)用以满足性能甚至安全方面的需要。

1.2 嵌入式系统的特征

1.完成单一或一组紧密相关的特定功能

嵌入式系统产生某种动作，以响应外部事件的要求。为了完成这个功能，嵌入式系统在软件的控制下通过硬件来高速地获取数据，并进行处理，而后产生响应动作。整个过程是在严格的时间和可靠性的约束下进行的。由于这些约束相当苛刻，嵌入式系统通常只用于满足单方面的应用。

2.具有高性能和实时的要求，甚至这些需求要放到第一位

实时特征是嵌入式系统的主要特征。与实时系统一样，可以根据对响应时间要求的不同分为硬实时和软实时。硬实时要求响应时间范围很严格，而软实时的时间限制稍宽。

3.软/硬件紧密融合

在嵌入式系统中，硬件操作需要特定软件控制，软件运行需要特定硬件环境支持。在嵌入式系统软件，硬件体系结构内部，各层次模块之间的耦合度也要比通用计算机系统强。与此构成鲜明对比的是，通用计算机系统更强调系统的模块化，标准化，层次化。

4.系统的可靠性和安全性

尽管所有的系统都要求可靠，但嵌入式系统在可靠性、重启动和故障恢复方面有更特殊的要求。对于可靠性要求特别高的场合，往往采用冗余备份方式。

5.效率优先

嵌入式系统的硬件和软件都必须高效率的设计，在保证稳定安全可靠的基础上量体裁衣，取出冗余，力争在同样的硅片面积上实现更高的性能。

6.功率消耗比较少

在嵌入式系统中，能耗起着决定性作用。耗能大的元器件一般发热量比较大，那么就需要风扇之类的装置来降温，而使用风扇将降低这个系统的使用寿命。并且，嵌入式系统一般使用电池作为主要电源，如果能耗很大的话，很快就会将电池耗尽。

7.自身开发能力差

嵌入式系统本身不具有自举开发能力。即使设计完成以后，用户通常也不能对其中的程序功能进行修改，必须有一套交叉开发工具和交叉开发环境才能进行开发。

1.3 嵌入式系统的发展趋势

未来嵌入式系统的发展趋势将趋向软硬件系统整合，SOC 设计，应用程序研发以及内容服务几个方面发展：

1.系统：嵌入式操作系统

嵌入式操作系统并未要求全面，但是必须能够依据系统设计规格，有效地发挥出硬件的运算能力，使得产品达到性价比的最大化。现在，虽然已经出现了许多比较优秀的嵌入式操作系统，比如 VXWORK，QNX，WINCE。但是任何系统都有一个平台移植问题。如何才能最大化的发挥硬件设备的潜能是个值得研究的方向。

2. SOC

嵌入式产品所需的处理器及芯片组要求体积小、散热佳、省电，以此多采用高整合度的 SOC（system-on-chip）为其处理器核心，为了尽快缩小制成技术进步与设计生产力间的差距，并加速 SOC 的实现，SIP（silicon intellectual property）的重复使用成为各个方面瞩目的焦点。

3.应用软件

嵌入式软件可区分为用户端的应用程序及服务器端的整合软件，服务器端的程序是以嵌入式操作系统为核心，并搭配各种数据库系统；用户端由于各个产品种类繁多，可开发出的软件也相对增加。有一个裁减得当的操作系统固然重要，但是如果没有完善的应用程序，这个系统的功效也不能得到很好的发挥。

4.服务

由于嵌入式产品必须能随身携带或走入家居生活，故其体积上要求轻薄短小、造型及颜色必须任性化、输入必须自然化、输出必须多媒体化才能吸引消费者；另一方面，由于嵌入式产品多与网络结合，所以与网络服务提供商或电子商务平台商极易结合。

第 2 章 嵌入式系统的基本原理

2.1 一个基本的嵌入式系统

与为通用计算机设计的软件不同，嵌入式软件通常无法在不做显著修改的情况下在其它嵌入式系统中运行。这主要是由于嵌入式系统中底层硬件之间的明显不同所致。每个嵌入式系统的硬件都是为特定的应用专门调整过的，这样才能使系统的成本保持很低。所以，不必要的电路就被省去了，硬件资源也尽可能地共享使用。

所有的嵌入式系统都包含处理器和软件，要想执行软件，就必须有存储执行代码的地方和管理运行时数据的临时存储区，这就分别要用到ROM 和RAM。而且，都必须包含某种输入和输出，其输出几乎总是它的输入和其它一些因素的函数。

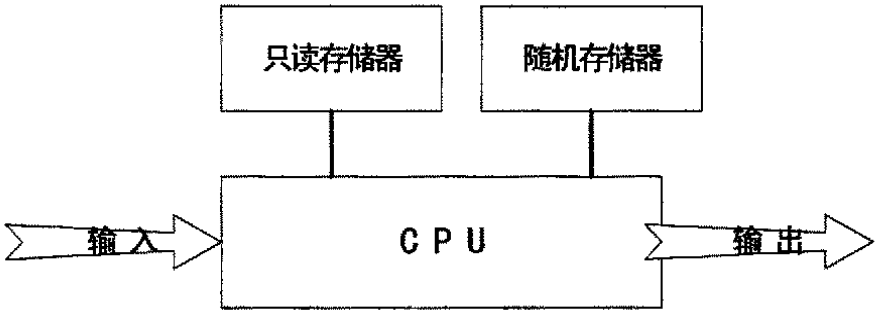


图2.1 基本的嵌入式系统结构

2.2 典型嵌入式系统的基本结构

嵌入式系统通常由嵌入式处理器，外围电路，嵌入式操作系统和应用软件等几大部分组成。

2.2.1 嵌入式处理器

嵌入式处理器是嵌入式系统的核心部件。它通常把通用计算机中许多由板卡完成的任务集成在芯片内部，从而有利于嵌入式系统设计趋于小型化，并具有高效率，高可靠性等特征。

2.2.2 外围设备

外围设备是指一个嵌入式系统中，除了嵌入式处理器以外，用于完成存储，通信，调试，显示等辅助功能的其他部件。

- 存储器：静态易失性存储器（RAM/SRAM），动态存储器（DRAM）和非易失性存储器（闪存）。其中，闪存具有可擦写次数多，存储速度快，容量大等特点，在嵌入式系统中广泛使用。
- 输入输出设备：嵌入式系统中输入设备一般包括触摸屏，语音识别，按键，键盘等。输出设备主要是LCD显示和语音输出。
- 接口：主要用于CPU和外设，存储器的连接和数据交换。主要有并口，串口，USB，PCMCIA，红外线接口，SPI串行外围设备接口，I^2C总线接口，以太网口等，CAN总线接口等。

2.2.3 嵌入式操作系统

在大型嵌入式应用系统中，为了使嵌入式开发更方便快捷，需要具备一种稳定的安全的软件模块集合，用以管理存储器分配，中断处理，任务间通信和定时器相应，以及提供多任务处理等，即嵌入式操作系统。嵌入式系统的引入，大大提高了嵌入式系统的功能，方便了应用软件的设计，但同时也占用了宝贵的嵌入式系统资源。一般在比较大型或需要多任务的应用场所，才考虑使用嵌入式操作系统。

2.2.4 应用软件

嵌入式系统的应用软件是针对特定的实际专业领域，基于相应的

嵌入式硬件平台，并能完成用户和其任务的计算机软件。用户的任务可能有时间和精度的要求。有些应用软件需要嵌入式操作系统的支持，但在简单的应用场合下，不需要专门的操作系统。

由于嵌入式应用对成本十分敏感，因此，为减少系统成本，除了精简每个硬件单元成本外，应尽可能地减少应用软件的资源消耗，尽可能的优化。应用软件是实现嵌入式系统功能的关键，对嵌入式系统软件和应用软件的要求也与通用计算机有所不同。

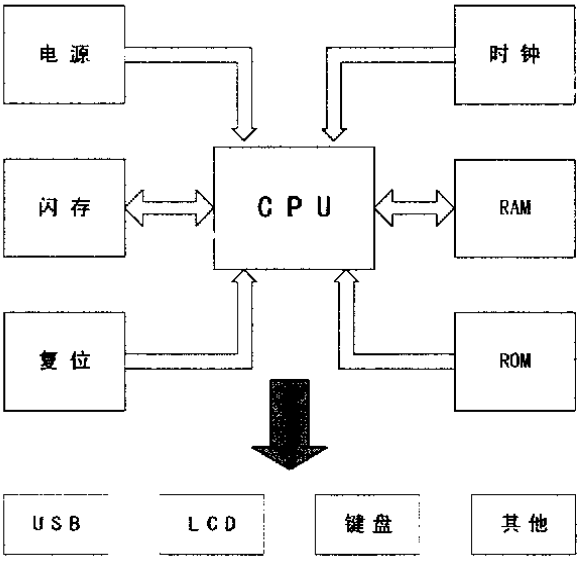


图2.2 典型嵌入式系统的硬件组成

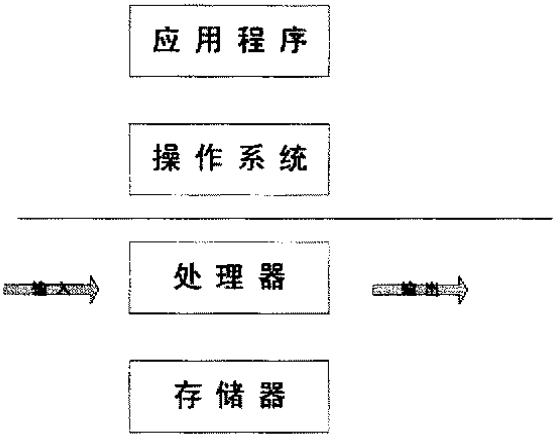


图2.3 典型嵌入式系统的软件基本组成

2.3 嵌入式系统常见的设计需求

嵌入式系统的其它部分通常是互不相同的。实现之间的差异是由不同的设计侧重导致的。每个系统都是面向完全不同的一整套需求，这些需求的综合考虑直接影响了产品的开发过程。

需要考虑的设计需求包括：

- 处理能力

  要完成目标所需的运算能力，一个常用的衡量运算能力的指标是MIPS（每秒可执行的百万条指令数）。但是，还要考虑处理器的其它一些重要特性，例如：寄存器字长，一般都会是8 到64 位。

- 存储能力

  用来保存执行代码和数据的存储器的容量。存储容量也会影响处理器的选择。

- 开发费用

  硬件软件开发过程所需的费用，这是一个确定的，一次性的花费。通常对于大批量产品，这也许是无关紧要的，如果只生产少量产品，可能需要仔细衡量。

- 批量

  生产费用和开发费用的折中考虑主要由期望的生产批量和销售量所决定的。

- 预计的生命周期

  系统必须延续多久，这影响到从硬件的选择到开发和生产费用方面的各种设计决策。

- 可靠性

  最终产品应具有什么程度的可靠性。可靠性标准通常用系统无故障运行时间，即平均无故障间隔时间来衡量。

2.4 嵌入式系统应用软件开发过程

嵌入式应用软件的开发方式一般是：在宿主机上建立开发环境，

进行应用程序编码和交叉编译，然后宿主机通过网口或串口将交叉编译生成的可执行目标代码装载到目标机上，并用交叉调试器进行调试。应用程序经过调试和优化，最后将应用程序固化到目标机中实际运行。

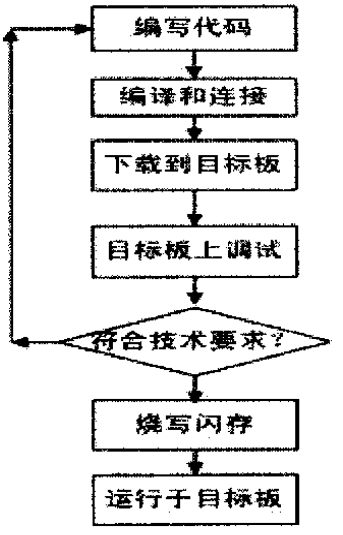


图 2.4 典型应用软件开发流程

嵌入式软件开发过程较为复杂，主要表现为以下几点：

1.软件要求固态化存储

为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存贮于磁盘等载体中。

2.软件代码高质量、高可靠性

尽管半导体技术的发展使处理器速度不断提高、片上存储器容量不断增加，但在大多数应用中，存储空间仍然是宝贵的，还存在实时性的要求。为此要求程序编写和编译工具的质量要高，以减少程序二进制代码长度、提高执行速度。

3.操作系统软件(OS)的调度性是基本要求

在多任务嵌入式系统中，对重要性各不相同的任务进行统筹兼顾的合理调度是保证每个任务及时执行的关键，单纯通过提高处理器速度是无法完成和没有效率的，这种任务调度只能由优化编写的系统软件来完成，因此系统软件的优化调度性是基本要求。

第3章 嵌入式操作系统
——LINUX操作系统

3.1 操作系统概述

在计算机技术发展的初期阶段，计算机系统中没有操作系统(Operating System)这个概念。应用程序开发人员都要对处理器和硬件进行彻头彻尾的控制。实际上，第一个操作系统的诞生，就是为了提供一个虚拟的硬件平台，以方便程序员开发，同时提高计算机的资源利用率。为实现这个目标，操作系统只需提供一些较为松散的函数、例程——就好象现在的软件库一样——以便于对硬件设备进行重置、读取状态、写入指令之类的操作。这是操作系统的雏形：计算机监控程序(Monitor)，使用户能通过监控程序来使用计算机。

随着计算机技术的发展，计算机系统的硬件、软件资源也愈来愈丰富，监控程序已不能适应计算机应用的要求。于是在六十年代中期，监控程序又进一步发展，形成了操作系统(Operating System)。嵌入式操作系统是一种支持嵌入式系统应用的操作系统，它是嵌入式系统（包括软件、硬件系统）极为重要的组成部分，负责这个系统的软、硬件资源的分配、调度、控制、协调并发活动。与通用操作系统相比较，嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固化性、以及应用的专用性等方面具有较为突出的特点。

3.1.1 操作系统的概念和历史

操作系统是一组计算机程序的组合，用来有效的控制和管理计算机的硬件和软件环境，即合理的对资源进行调度，并为用户提供方便的应用接口。它主要充当计算机用户和计算机硬件之间的一个中介，并用于管理计算机资源和控制应用程序运行的计算机程序。

1.操作系统提供的相关服务

◆ 程序运行

一个程序的运行离不开操作系统的配合，其中包括指令和数据载入内存，I/O 设备和文件系统的初始化等等。

◆ I/O

每种I/O设备的管理和使用都有自己的特点。而操作系统接管了这些工作，从而使用户在使用这些I/O 设备的过程中会感觉更方便。

◆ 文件访问

文件访问不仅需要熟悉相关I/O 设备（磁盘驱动器等）的特点，而且还要熟悉相关的文件格式。另外，对于多用户操作系统或者网络操作系统，从计算机安全角度考虑，需要对文件的访问权限做出相应的规定和处理。这些都是操作系统所要完成的工作。

◆ 系统访问

对于一个多用户或者网络操作系统而言，操作系统需要对用户系统访问权限做出相应的规定和处理。

◆ 错误检测和反馈

当操作系统运行时，会出现这样那样的问题。操作系统应当提供相应的机制来检测这些信息，并且能对某些问题给出合理的处理或者报告用户。

◆ 系统使用纪录

在一些现代操作系统中，出于系统性能优化或者系统安全角度考虑，操作系统会对用户使用过程纪录相关信息。

◆ 程序开发

一般操作系统都会提供丰富的API 供程序员开发应用程序，并且很多程序编辑工具，集成开发环境等等也都是通过操作系统提供的。而计算机有很多资源，它们分别用于数据的传输、处理或存储以及这些操作的控制。这些资源的管理工作就交给了操作系统。

2.整个操作系统的发展

◆ 串行处理系统

早期没有操作系统的概念，人们通过显示灯、跳线、某些输

入输出装置设备同计算机打交道。当需要执行某个计算机程序时，人们通过输入设备将程序灌入计算机，然后等待运行结果。这种工作方式为串行处理方式。

- 简单批处理系统

  核心思想是借助某个称为监视器的软件，用户不需要直接和计算机硬件打交道，而只需要将自己所要完成的计算任务提交给计算机操作员。在操作员那里，所有计算任务按照一定的顺序被成批输入计算机中。当某个计算任务结束后，监视器会自动开始执行下一个计算任务。

- 多任务批处理系统

  多任务程序设计思想主要是允许某个计算任务在等待I/O操作的时候，计算机可以转而执行其他计算任务，从而提高处理器的利用率。

- 分时系统

  在分时系统中，处理器时间按照一定的分配策略在多个用户中间共享。在实际的单处理器系统中，是多个任务交替获取处理器控制权，交替执行，从而提供更好的交互性能。

- 现代操作系统

  现代操作系统技术是在综合了以上四种典型的操作系统技术的基础上提出的操作系统实现方式，它适应了现代计算机系统管理和使用要求。其主要特征是多任务、分时、多用户。现代操作系统一般包括：进程和进程管理；内存和虚拟管理；信息保护和安全；调度和资源管理；模块化系统设计。

3.2 嵌入式操作系统的特点

嵌入式操作系统是相对于一般的操作系统而言的，它除了具备一般操作系统最基本的功能，如任务调度、同步机制、终端处理、文件处理等，还具有以下特点：

- 可装卸性。具有开放性、可伸缩性的体系结构。
- 强实时性。嵌入式操作系统的实时性一般较强，可用于这个设备控制当中。

- 统一的接口。提供各种设备的驱动接口。
- 操作方便、简单，提供友好的图像界面。
- 提供强大的网络功能。支持TCP/IP协议以及其他协议，提供TCP/UDP/IP/PPP协议支持，统一的MAC访问层接口，为各种移动计算设备预留接口。
- 强稳定性，弱交换性。嵌入式系统一旦开始运行，就不需要用户过多的干预，这就要求负责系统管理的嵌入式操作系统具有较强的稳定性。嵌入式操作系统的用户接口一般不提供操作命令，它通过系统的调用命令，向用户程序提供服务。
- 固化代码。在嵌入式系统中，嵌入式操作系统和应用软件被固化在嵌入式系统计算机中的闪存中。辅助存储器在嵌入式系统中很少使用，因此，嵌入式操作系统的文件管理功能应该能够容易的拆卸，而采用各种内存文件系统。
- 更好的硬件适应性，即良好的移植性。

3.3 LINUX 和嵌入式 LINUX

3.3.1 LINUX 的简介

LINUX 是一个免费的开源的操作系统，它具有完全的多任务，X Windows 系统，TCP/IP 网络等性质。它脱胎于 UNIX 系统，许多特性与 UNIX 非常相似。更重要的是，许多 UNIX 下的程序经过简单移植，甚至不需要改动就完全在 LINUX 系统运行。狭义的说，LINUX 只是操作系统的内核，主要完成进程调度，虚拟内存，文件管理，设备 IO 管理等功能，也就说 LINUX 只是操作系统的最底层部分。但是，大多数人们还是认为"LINUX"是一个完全系统——内核以及运行在内核之上的各种应用：完全的开发环境，图形界面，文本编辑，游戏等等。

虽然，LINUX 产生于一个学生的"心血来潮"，就是现在，内核源码的发布与变化也由这个学生控制，但是，由于 LINUX 完全遵从 GNU/GPL 规定，所以世界上有成千上万的人们一起推动着 LINUX 内核的发展，使得 LINUX 成为一个性能优越，超级稳定的多用户系统。

3.3.2 嵌入式 LINUX

嵌入式LINUX 是一种开放源码、软实时、多任务的嵌入式操作系统，是标准LINUX 系统的嵌入式移植版。它具备了LINUX 的所有功能和优点：支持多用户，多进程，多线程，实时性较好。它本身内置网络支持，是功能强大而稳定的操作系统。

嵌入式LINUX 的应用范围非常广泛：从移动计算平台（PDA，掌上电脑）、 信息家电（机顶盒，数字电视，一切和Internet 相连接的设备）、无线通讯设备（包括WAP 手机，股票接收机等）、 到工业/商业控制（智能工控设备，POS/ATM 机）、 电子商务平台、甚至军事应用等等。

嵌入式LINUX 主要可以分为两类：第一类是在利用LINUX 强大功能的前提下，使它尽可能的小，以满足许多嵌入式系统对体积的要求，如uCLINUX；第二类是将LINUX 开发成实时系统尤其是硬（firm）实时系统，应用于一些关键的控制场合，如RTLINUX、Hard HatLINUX 等。

3.3.3 嵌入式 LINUX 基本构成元素

一个小型的嵌入式LINUX系统只需要下面六个基本元素：

- 引导工具
- LINUX微内核
- 初始化进程
- 一个文件系统
- 硬件驱动程序
- 提供所需功能的应用程序

3.3.4 LINUX 和嵌入式 LINUX 的一些比较

嵌入式 LINUX 脱胎于标准 LINUX，但是他们之间还是有很多不同。

1.内核改变

为了满足嵌入式系统的某些性能要求（比如实时性、最小内存开销等），不同嵌入式LINUX系统均对内核作了不同程度的修改，以满足需要。比如，为了满足实时性要求，可能会采用优先级调度策略替换时间片轮转调度策略，将内核由不可抢占调度方式改为可抢占调度方式；采用某种改进方法将最大中断禁止时间、任务切换时间减低到最小。

2.内存管理

标准的LINUX 还有一个特点是使用虚拟内存，程序过大，可以交换到虚拟存储器上。然而在嵌入式系统中特别要求实时性很强的系统来说，这个功能的确不那么重要，因为这个机制会浪费时间。所以，嵌入式系统的应用程序还是在固定的地方运行比较好。考虑到一些CPU 有这方面的特点，建议保留虚拟存储器的代码，这样能够使得不同进程使用相同代码 。如果没有了这个功能，每个程序都需要自己的运行库，在内存中就会有库的很多拷贝。其实只需把交换空间的长度设置为零，就可以关闭虚拟内存的页面换入和换出功能。

3.用户界面

在LINUX 中使用的是X-Window，而嵌入式LINUX 多使用MiniGUI、MicroWindows、TinyX 等小型GUI。

3.4.当前流行的嵌入式 LINUX 操作系统

3.4.1 RT-LINUX

这是由美国墨西哥理工学院开发的嵌入式LINUX操作系统。到目前为止，RT-LINUX已经成功地应用于航天飞机的空间数据采集、科学仪器测控和电影特技图像处理等广泛领域。RT-LINUX开发者并没有针对实时操作系统的特性而重写LINUX的内核，因为这样做的工作量非常大，而且要保证兼容性也非常困难。为此，RT-LINUX提出了

精巧的内核，并把标准的LINUX核心作为实时核心的一个进程，同用户的实时进程一起调度。这样对LINUX内核的改动非常小，并且充分利用了LINUX下现有的丰富的软件资源。

3.4.2 UCLINUX

uCLINUX是Lineo公司的主打产品，同时也是开放源码的嵌入式LINUX的典范之作。uCLINUX主要是针对目标处理器没有存储管理单元MMU （Memory Management Unit）的嵌入式系统而设计的。它已经被成功地移植到了很多平台上。由于没有MMU，其多任务的实现需要一定技巧。uCLINUX是一种优秀的嵌入式 LINUX版本，是micro-Conrol-LINUX的缩写。它秉承了标准LINUX的优良特性，经过各方面的小型化改造，形成了一个高度优化的、代码紧凑的嵌入式LINUX。虽然它的体积很小，却仍然保留了LINUX的大多数的优点：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和标准丰富的API。它专为嵌入式系统做了许多小型化的工作，目前已支持多款CPU。其编译后目标文件可控制在几百KB数量级，并已经被成功地移植到很多平台上。

3.4.3 Embedix

Embedix是由嵌入式LINUX行业主要厂商之一Luneo推出的，是根据嵌入式应用系统的特点重新设计的LINUX发行版本。Embedix提供了超过25种的LINUX系统服务，包括Web服务器等。系统需要最小8MB内存，3MB ROM或快速闪存。Embedix基于LINUX 2.2内核，并已经成功地移植到了Intel x86和PowerPC处理器系列上。像其它的LINUX版本一样，Embedix可以免费获得。Luneo还发布了另一个重要的软件产品，它可以让在 Windows CE上运行的程序能够在Embedix上运行。Luneo还将计划推出Embedix的开发调试工具包、基于图形界面的浏览器等。可以说，Embedix 是一种完整的嵌入式LINUX解决方案。

3.4.4 XLINUX

XLINUX是由美国网虎公司推出，主要开发者是陈盈豪。他在加盟网虎几个月后便开发出了基于XLINUX的、号称是世界上最小的嵌入式LINUX系统，内核只有143KB，而且还在不断减小。XLINUX核心采用了"超字元集"专利技术，让LINUX核心不仅可能与标准字符集相容，还含盖了12个国家和地区的字符集。

3.4.5 PoketLINUX

PoketLINUX由Agenda公司采用、作为其新产品"VR3 PDA"的嵌入式LINUX操作系统。它可以提供跨操作系统构造统一的、标准化的和开放的信息通信基础结构，在此结构上实现端到端方案的完整平台。 PoketLINUX资源框架开放，使普通的软件结构可以为所有用户提供一致的服务。PoketLINUX平台使用户的视线从设备、平台和网络上移开，由此引发了信息技术新时代的产生。在PoketLINUX中，称之为用户化信息交换（CIE），也就是提供和访问为每个用户需求而定制的"主题"信息的能力，而不管正在使用的设备是什么。

3.4.6 ETLINUX

EtLINUX 是适用于小型工业计算机(pc/104)上的完全基于LINUX 的操作系统。他具有全 UNIX 功能：抢占式多任务，多进程，内存保护，快速 I/O，增强的稳定性，速度等等。并且他还具有一些自己的特性，例如:嵌入式 web 服务器，telnet 服务器，email 服务器(可以执行通过 email 传递而来的远程命令)，方便的远程文件管理，灵活的包选择机制(允许系统的快速定制)，源代码公开。

3.4.7 红旗嵌入式 LINUX

由北京中科院红旗软件公司推出的嵌入式LINUX是国内做得较

好的一款嵌入式操作系统。目前，中科院计算所自行开发的开放源码的嵌入式操作系统——Easy Embedded OS(EEOS)也已经开始进入实用阶段了。该款嵌入式操作系统重点支持p-Java。系统目标一方面是小型化，另一方面能重用LINUX的驱动和其它模块。由于有中科院计算所的强大科研力量做后盾，EEOS有望发展成为功能完善、稳定、可靠的国产嵌入式操作系统平台。

3.5 LINUX 的内核模块编程

LINUX 有一个很好的特性:内核提供的特性可在运行时进行扩展。这意味着当系统启动并运行时，我们可以向内核添加功能。

可在运行时添加到内核中的代码被称为"内核模块"。内核模块是一些可以让操作系统内核在需要时载入和执行的代码，这同样意味着它可以在不需要时由操作系统卸载。它们扩展了操作系统内核的功能却不需要重新启动系统。一旦装入一个 LINUX 内核模块，那么他就像任何标准的内核代码一样成为内核的一部分，具有相同的权限和职责。一方面凡是由内核移出的所有符号都可以在模块中引用;另一方面，除了这些特意移出的符号及系统调用外，应用程序别无其他途径直接访问内核中的资源。

在应用程序界面上，内核通过 4 个系统调用支持可安装模块的动态安装和拆卸，他们是 creat_module()，init_module，query_module，以及 delete_module。通常情况下，用户不需要直接跟这些系统调用打交道，而是使用系统提供的工具 insmod，rmmod，modprobe 来安装和卸载模块。当然，这几个工具最终还是通过这些系统调用完成有关操作。

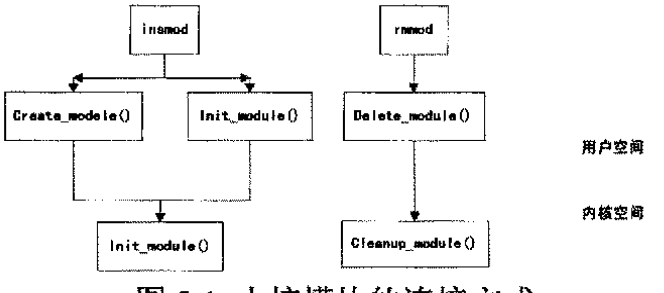


图 3.1 内核模块的连接方式

要想为 2.6.X 内核构造模块，必须在自己的系统中配置并构造好内核树，这一要求和先前版本的内核不同，先前的内核只需要有一套内核头文件就够了。但因为 2.6 内核的模块要和内核源代码树中的目标文件连接，通过这种方式，可得到一个更加健壮的模块装载器，但也需要这些目标文件存在于内核目录树中。

3.5.1 内核模块和应用程序的对比

大多数小规模以及中规模应用程序是从头到尾执行单个任务，而模块却只是预先注册自己以便于将来的某个请求，然后他的初始化函数就立即结束。模块初始化函数的任务就是为以后的调用模块函数预先作准备。模块的退出函数将在模块被卸载之前调用。这种编程方式和事件驱动的编程有些类似，但并不是所有的应用程序都是事件驱动的，而每个内核模块都是这样的。事件驱动的应用程序和内核代码之间的另一个主要不同是:应用程序在退出时，可以不管资源的释放或者其他的清除工作，但模块的退出函数却必须仔细撤销初始化函数所作的一切，否则，在系统重新引导之前某些东西会残留在系统中。

内核模块只能使用作为内核一部分的函数，和内核相关的任何内容都在安装和配置好的内核源代码树的头文件中声明，其中，大多数相关头文件保存在 include/linux 和 include/asm 目录中，但 include/的其他子目录中保存有和特定内核子系统相关的头文件。

内核编程和应用程序编程的另一个重要不同之处在于各环境下处理错误的方式不同:应同程序开发过程中的错误是无害的，并且总是可以使用调试器跟踪到源代码中的问题所在，而一个内核错误即使不影响整个系统，也至少会杀死当前程。

	C 语言普通程序	C 语言内核程序
入口	Main()	Init_module()
出口	无	Cleanup_module()
编译	gcc -c	gcc –c –D _KERNEL_ -D MODULE
连接	gcc	insmod
运行	直接运新	insmod
调试	gdb	Kgdb，kdebug，kdb 等内核调试工具

3.5.2 用户空间和内核空间

模块运行在所谓的内核空间中，应用程序运行在所谓的用户空间中。这个概念是操作系统理论的基础之一。

1.处理器保护

实际上，操作系统的作用是为应用程序提供一个对计算机硬件的一致视图。除此之外，操作系统必须负责程序的独立操作并保护资源不受非法访问。这个重要任务只能在 CPU 能够保护系统软件不受应用程序破坏时才能完成。

所有的现代处理器都具备这个功能。人们选择的方法是在 CPU 中实现不同的操作模式。不同的级别具有不同功能，在较低的级别中将禁止某些操作。程序代码只能通过有限数目的"门"来从一个级别切换到另一级别。UNIX 系统设计时利用了这种硬件特些，使用了两个这样的级别。当前所有的处理器都至少具有两个保护级别，而其他的一些处理器，比如 X86，ARM 系列，则有更多的级别。当处理器存在多个级别时，UNIX 使用最高级别和最低级别。在 UNIX 中，内核运行在最高级别(也称为超级用户级别)，在这个级别中可以进行所有的操作。而应用程序运行在最低级别(也称为用户级别)，在这个级别中，处理器控制着对硬件的直接访问以及对内存的非授权访问。

2.用户空间和内核空间权限

对应于在超级用户级的内核程序，他所在的内存空间是内核空间;对应于在用户级运行的应用程序，他所在的内存空间是用户空间。这两个术语不仅说明两种模式具有不同的优先级，并且还说明每个模式都有自己的内存映射，也即自己的地址空间。每当应用程序执行系统调用或者被硬件中断挂起时，UNIX 将执行模式从用户空间切换到内核空间。执行系统调用的内核代码运行在进程上下文中，他代表调用进程执行操作，因此能够访问进程地址空间的所有数据。而处理硬件中断的内核代码和进程是异步的，与任何一个特定进程无关。对中断来说，他并不存在于任何进程的上下文中，而是有内核来运行的。对模块来说，他就是在系统的用户空间定义的，然后通过用户空间的程

序 insmod 将他插入到内核空间中运行，可以像真正的内核一样充分利用超级用户级的功能执行程序。

3. 用户空间和内核空间的范围及函数参数传递

运行内核代码的时候用内核的段描述符号可以直接访问用户空间，但运行用户代码的时候用户描述符号不能访问内核空间，这使用了保护模式的一些机制。在内核中的函数代码里面的用户空间和内核空间参数传递是没有直接拷贝的，用户参数提供的指针等不能指向系统空间。

LINUX 的 task_struct{}结构中有一个成员 addr_limit，该成员规定了进程有用户态和内核态情况下的虚拟空间地址访问范围。在用户态，addr_limit 成员值是 0XBFFFFFFF，也就是有 3G 的虚拟地址空间；在内核态，addr_limit 的值是 0XFFFFFFFF，范围扩展了 1GB。进程运行时会检测用户传递参数访问的空间是不是小于等于这个范围，因为内核空间在用户空间之上。

内核对用户空间的数据访问存在两种情况，第一种是把内核空间的数据放到用户空间的内存块或把用户空间中的数据拷贝到内核空间的内存块中。第二种是内核空间调用的函数中的参数含有用户空间的内存指针。对于第一种情况常用函数 copy_from_user/copy_to_user 来解决。对于第二种情况，使用函数 set_fs(x)临时设置用户空间限制为内核空间的范围，调用完了过后恢复就可以解决了。

4. 内核态和用户态之间的数据传递

在内核空间和用户空间存在数据的传递，下面的函数和宏在内核中对这种传递进行操作。

函数 copy_to_user 从内核空降将数据拷贝到用户空间，拷贝成功返回 0，否则，返回不能被拷贝的字节数。其参数 to 表示在用户空间的目标地址，参数 from 是在内核空间的源地址，参数 n 是需要拷贝数据的长度。函数原型如下：

Unsigned long copy_to_user (void __user *to， const void *from，unsigned long n);

宏定义 put_user 写一个简单的数值从内核空间 x 到用户空间 ptr，

他支持简单的类型如 char 和 int，但不是像结构或数组的大数据类型，拷贝成功返回 0，失败时返回错误-EFAUL。

函数 copy_from_user 从用户空间拷贝一块数据，参数 to 表示在内核空间的目标地址，参数 from 是在用户空间的源地址，参数 n 是需要拷贝的数据长度。拷贝成功返回 0，否则，返回不能被拷贝的字节数。如果有些数据没拷贝，函数 copy_from_user 将填充 0 到请求的长度 0，函数 copy_from_user 类型如下:

Unsigned long copy_from_user (void *to, const __user *from, unsigned long n);

宏定义 get_user 从用户空间 ptr 中获取数据并存放在内核数据地址 x 中。x 变量用来存储结构，ptr 指向用户空间里的源地址。这个宏地址从用户空间拷贝单个简单的变量到内核空间。但不是像结构或数组的大数据类型，拷贝成功返回 0，失败时返回错误-EFAUL。

3.5.3 内核模块的编译，装载和使用

内核模块有其自己独特的创建，装载和使用过程。

1. 编译模块

模块的构造过程和用户空间应用程序的构造构成有很大的不同。内核是一个大的，独立的程序，为了将他的各个片断放在一起，要满足许多详细而明确的要求。在构造内核模块之前，有一下先决条件必须满足:首先确保正确版本的编译器，模块工具和其他必要工具，内核文档目录中的 documentation/changes 文件列出了需要的工具版本。其次，在自己的文件系统中要安装 2.6 内核树，否则，无法构造可装载模块。

如果我们要构造的模块名称为 module.ko，并由两个源文件生成(比如 file1.c 和 file2.c)，则正确的 makefile 可如下编写:

```
Obj-m := module.o
Module-objs := file1.o file2.o
```

为了让上面这种类型的 makefile 文件正常工作，必须在大的内核构造系统环境中调用他们。例如我们的内核源代码树保存在

~/kernel-2.6 目录中，则用来构造模块的 make 命令应该是(在包含模块源代码和 makefile 的目录中键入):

```
Make –C ~/kernel-2.6 M='pwd' modules
```

上述命令首先改变目录到-C 指定的位置(即内核源代码目录)，其中保存由内核的顶层 makefile 文件。M=选项让该 makefile 在构造 modules 目标之前返回到模块源代码目录。然后，modules 目标指向 obj-m 变量中设定的模块;在上面的例子中，我们将该变量设置成 module.o。

2. 装载和卸载模块

在构造模块后，下一步就是将模块装入内核中。insmod 为我们完成这项工作。insmod 程序和 ld 有些相似，他将模块代码和数据装入内核，然后使用内核的符号表解析模块中任何未解析的符号。然而，与连接器不同，内核不会修改模块的磁盘文件，而仅仅修改内存中的副本。Insmod 可以接受一下命令行选项，并且可以在模块连接到内核之前给模块中的整型和字符串型变量赋值。因此，一个良好设计的模块可以在装载时进行配置，这比编译时的配置为用户提供更多的灵活性，但有些情况下仍然要使用编译时的配置。

与 insmod 类似，modprobe 也用来将模块装载到内核中。他和 insmod 的区别在于，他会考虑要装载的模块是否引用了一些当前内核不存在的符号。如果有这类引用，modprobe 会在当前模块搜索路径中查找定义了这些符号的其他模块。如果 modprobe 找到了这些模块(即要装载的模块所依赖的模块)，他会同时将这些模块装载到内核。如果在这种情况下使用 insmod，给命令会失败，并在系统日志文件中记录"unresolved symbols(未解析的符号)"消息。

我们可以使用 rmmod 工具从内核中移出模块。注意，如果内核认为模块仍然在使用状态(例如，某个程序正打开由该模块导出的设备文件)，或者内核被配置为禁止移出模块，则无法移出该模块。配置内核并使得内核在模块忙的时候仍能"强制"移出模块也是可能的。但是，使用这种方式，还不如重新引导系统更好。

Lsmod 程序列出当前装载到内核中的所有模块，还提供了一下信息，比如其他模块是不是在使用某个特定模块等。lsmod 通过读取

/proc/kallmodules 虚拟文件来获得这些信息。有关当前已装载模块的信息也可以在 sysfs 虚拟文件系统的 sys/module 下找到。

3.6 更适用于嵌入式系统的新内核

LINUX 内核的开发已经经历了一个漫长的过程，最初是 Linus Torvalds 于 1991 年发布的原始的 0.1 版本，这个版本中包括一个基本的调度器、IPC（进程间通信）和内存管理算法。而现在它已经是一个以往操作系统的实用的替代品，在市场上表现出了强大的竞争力。越来越多的政府机构和 IT 巨头的注意力正在转向 LINUX。从最小的嵌入式设备到 S/390，从手表到大型企业服务器，LINUX 现在几乎可以用于所有的地方。

3.6.1 2.6 内核的新特性

LINUX 2.6 是 LINUX 开发周期中的下一个主要版本，它包括了一些强有力的特性，这些特性旨在改进高端企业服务器的性能和支持越来越多的嵌入式设备。下面主要介绍一下 2.6 内核针对于嵌入式系统的新亮点。

1. 速度更快

1）支持 NUMA（非均衡内存访问）服务—新的任务调度器优化了诸如 I/O 设备，多 CPU，和内存等相关系统资源。这样在 LINUX 上运行大型服务的速度将有显著提高。

2）改进的同步多线程技术提高了任务调度器对虚拟处理器和实际处理器之间的负载均衡优化能力。这样，LINUX 在 P4 类具有超线程技术的 CPU 上的表现更加优越。

3）新的任务调度器。2.6 内核的任务调度器对系统资源具有更好的理解力和优化能力。可以在 16 个以上的 CPU 间进行无缝的进程切换。

2. 系统更大

2.6支持更多的设备。2.4内核支持255个主设备和255个子设备。新的2.6内核支持4096个主设备和上千万个子设备。

3. 性能更优异

1）LINUX的NFS（网络文件系统）的速度，安全性上有了很大改善

- 支持NFSv4协议
- 使用新的密码方式提供更安全的授权
- NFS服务器同时最多可支持用户是2.4内核的64倍

2）内核中的EXT3文件系统提供更安全的数据存储和更快捷的数据恢复。

4. 多媒体功能

1）内核包含了ALSA（高级LINUX声音结构）。ALSA完全基于线程，并且是SMP可靠的。使得LINUX具有更好的多声卡，硬件混音能力。

2）增强了游戏手柄支持。

5.无线设备

1）将无线设备的各个子系统合并成一个集中式无线API接口，这样使得无线工具可以轻松的应用于所有的已支持的无线设备。

2）内核包含蓝牙技术所需协议，可以对蓝牙设备进行更好的支持。

6.电池保护

支持新处理器所具有的根据电源状况调节运行频率的能力。

7.设备支持

1）统一的设备模型集中控制了系统资源，对热插拔，PC卡，USB和火线设备有了更好的支持。

2）新内核是真正的全即插即用操作系统，并且可以在系统的BIOS中进行配置。

3）驱动已经被模块化，这样就对多声卡系统有了清晰支持。

4）内核支持 USB2.0。基于 2.6 内核的系统可以同时成为 USB 主机和 USB 设备。这样，可以方便的连接和同步诸如 PDA 的手持设备。

8.安全性

新内核支持 IPsec（IP 安全）网络协议（例如 IPV6），这样就提供了网络协议层的密码保护。

9.嵌入式应用

1）支持 uCLINUX。大多数的 uCLINUX 代码已经包含在内核中。在桌面 LINUX 和嵌入式 LINUX 之间第一次统一了开发环境。

2）支持更多没有 MMU 的处理器，这些处理器广泛应用于 PDA 等设备上。

3）嵌入式特性支持。允许新内核能方便的运用于嵌入式设备和消费电子类设备上。

3.6.2 2.6 内核在驱动程序上新变化

1. 使用新的入口

必须包含 <linux/init.h>

module_init(your_init_func);

module_exit(your_exit_func);

老版本：int init_module(void)； void cleanup_module(voi);

2.4 中两种都可以用，对如后面的入口函数不必要显示包含任何头文件。

2. GPL

MODULE_LICENSE("Dual BSD/GPL");

老版本：MODULE_LICENSE("GPL");

3. 模块参数

必须显式包含<linux/moduleparam.h>

module_param(name， type， perm);

module_param_named(name，value， type， perm);

参数定义

module_param_string(name， string， len， perm);

module_param_array(name， type， num， perm);

老版本：MODULE_PARM(variable，type);

MODULE_PARM_DESC(variable，type);

4. 模块别名

MODULE_ALIAS("alias-name");

这是新增的，在老版本中需在/etc/modules.conf 配置，现在在代码中就可以实现。

5. 模块计数

int try_module_get(&module);

imodule_put();

老版本：MOD_INC_USE_COUNT 和 MOD_DEC_USE_COUNT

6. 符号导出

只有显示的导出符号才能被其他模块使用，默认不导出所有的符号，不必使用 EXPORT_NO_SYMBOLS。

老板本：默认导出所有的符号，除非使用 EXPORT_NO_SYMBOLS

7. 内核版本检查

需要在多个文件中包含<linux/module.h>时，不必定义__NO_VERSION__。

老版本：在多个文件中包含<linux/module.h>时，除在主文件外的其他文件中必须定义__NO_VERSION__，防止版本重复定义。

8. 设备号

kdev_t 被废除不可用，新的 dev_t 拓展到了 32 位，12 位主设备

号，20位次设备号。

unsigned int iminor(struct inode *inode);

unsigned int imajor(struct inode *inode);

老版本：8位主设备号，8位次设备号

int MAJOR(kdev_t dev); int MINOR(kdev_t dev);

9. 内存分配头文件变更

所有的内存分配函数包含在头文件<linux/slab.h>。

老版本：内存分配函数包含在头文件<linux/malloc.h>

10. 结构体的初试化

gcc开始采用ANSI C的struct结构体的初始化形式：

```
static struct some_structure = {
.field1 = value,
.field2 = value,
};
```

老版本：非标准的初试化形式

```
static struct some_structure = {
field1: value,
field2: value,
};
```

11. 用户模式帮助器

int call_usermodehelper(char *path， char **argv， char **envp，int wait);

新增wait参数。

12. Request_module ()

request_module("foo-device-%d"， number);

老版本：

```
char module_name[32];
printf(module_name， "foo-device-%d"， number);
```

request_module (module_name)

13. dev_t 引发的字符设备的变化

1）取主次设备号

unsigned iminor(struct inode *inode);

unsigned imajor(struct inode *inode);

2）老的 register_chrdev()用法没变，保持向后兼容，但不能访问设备号大于 256 的设备。

3）新的接口为：

a）注册字符设备范围

int register_chrdev_region(dev_t from， unsigned count， char *name);

b）动态申请主设备号

int alloc_chrdev_region(dev_t *dev， unsigned baseminor，unsigned count， char *name);

c）包含 <linux/cdev.h>，利用 struct cdev 和 file_operations 连接

struct cdev *cdev_alloc(void);

void cdev_init(struct cdev *cdev， struct file_operations *fops);

int cdev_add(struct cdev *cdev， dev_t dev， unsigned count);

分别为，申请 cdev 结构，fops 连接和将设备加入到系统中

d) void cdev_del(struct cdev *cdev)

只有在 cdev_add 执行成功才可运行。

e）辅助函数：这一部分变化和新增的/sys/dev 有一定的关联

kobject_put(&cdev->kobj);

struct kobject *cdev_get(struct cdev *cdev);

void cdev_put(struct cdev *cdev);

14. 新增对/proc 的访问操作 <linux/seq_file.h>

以前的/proc 中只能得到 string， seq_file 操作能得到如 long 等多种数据。相关函数：

static struct seq_operations 必须实现这个类似 file_operations 得数据中得各个成员函数。

seq_printf();

int seq_putc(struct seq_file *m， char c);

int seq_puts(struct seq_file *m， const char *s);

int seq_escape(struct seq_file *m， const char *s， const char *esc);

int seq_path(struct seq_file *m， struct vfsmount *mnt，struct dentry *dentry， char *esc);

seq_open(file， &ct_seq_ops);等等

15.底层内存分配

1）<linux/malloc.h>头文件改为<linux/slab.h>

2）分配标志 GFP_BUFFER 被取消，取而代之的是 GFP_NOIO 和 GFP_NOFS

3）新增__GFP_REPEAT，__GFP_NOFAIL，__GFP_NORETRY 分配标志

4）页面分配函数 alloc_pages()，get_free_page()被包含在 <linux/gfp.h>中

5）新增 Memory pools <linux/mempool.h>

mempool_t *mempool_create(int min_nr，

mempool_alloc_t *alloc_fn，

mempool_free_t *free_fn，

void *pool_data);

void *mempool_alloc(mempool_t *pool， int gfp_mask);

void mempool_free(void *element， mempool_t *pool);

int mempool_resize(mempool_t *pool， int new_min_nr， int gfp_mask);

16. 内核时间变化

1）现在的各个平台的 HZ 为

Alpha:1024/1200;ARM:100/128/200/1000;CRIS:100;i386：1000;IA-64:1024;M68K: 100; M68K-nommu: 50-1000; MIPS: 100/128/1000; MIPS64: 100; PA-RISC: 100/1000; PowerPC32: 100; PowerPC64: 1000; S/390: 100; SPARC32: 100; SPARC64:100; SuperH:

100/1000; UML: 100; v850: 24-100; x86-64: 1000。

2）由于 HZ 的变化，原来的 jiffies 计数器很快就溢出了，引入了新的计数器 jiffies_64

3） #include <linux/jiffies.h>

u64 my_time = get_jiffies_64();

4）新的时间结构增加了纳秒成员变量

struct timespec current_kernel_time(void);

5）timer 函数没变，新增

void add_timer_on(struct timer_list *timer， int cpu);

6）新增纳秒级延时函数 ndelay();

17. 工作队列（workqueue）

1）任务队列（task queue ）接口函数都被取消，新增了 workqueue 接口函数

struct workqueue_struct *create_workqueue(const char *name);

DECLARE_WORK(name， void (*function)(void *)， void *data);

INIT_WORK(struct work_struct *work，

void (*function)(void *)， void *data);

PREPARE_WORK(struct work_struct *work，

void (*function)(void *)， void *data);

2）申明 struct work_struct 结构

int queue_work(struct workqueue_struct *queue，struct work_struct *work);

int queue_delayed_work(struct workqueue_struct *queue，struct work_struct *work，unsigned long delay);

int cancel_delayed_work(struct work_struct *work);

void flush_workqueue(struct workqueue_struct *queue);

void destroy_workqueue(struct workqueue_struct *queue);

int schedule_work(struct work_struct *work);

int schedule_delayed_work(struct work_struct *work， unsigned long delay);

18. DMA 的变化

未变化的有：

void *pci_alloc_consistent(struct pci_dev *dev， size_t size，dma_addr_t *dma_handle);

void pci_free_consistent(struct pci_dev *dev， size_t size，void *cpu_addr， dma_addr_t dma_handle);

变化的有：

1) void *dma_alloc_coherent(struct device *dev， size_t size，dma_addr_t *dma_handle， int flag);

void dma_free_coherent(struct device *dev， size_t size，void *cpu_addr， dma_addr_t dma_handle);

2) 列举了映射方向：

enum dma_data_direction {

DMA_BIDIRECTIONAL = 0，

DMA_TO_DEVICE = 1，

DMA_FROM_DEVICE = 2，

DMA_NONE = 3，

};

19.互斥 <linux/seqlock.h>

新增 seqlock 主要用于：

- 少量的数据保护
- 数据比较简单(没有指针)，并且使用频率很高
- 对不产生任何副作用的数据的访问
- 访问时写者不被饿死

20.内核可剥夺 <linux/preempt.h>

preempt_disable()； preempt_enable_no_resched()

preempt_enable_noresched()； preempt_check_resched()；

21.睡眠和唤醒

1）原来的函数可用，新增下列函数：

prepare_to_wait_exclusive()；prepare_to_wait()；

2）等待队列的变化

typedef int (*wait_queue_func_t)(wait_queue_t *wait，unsigned mode， int sync)；

void init_waitqueue_func_entry(wait_queue_t，*queue，wait_queue_func_t func)；

22.中断处理

1）中断处理有返回值了。IRQ_RETVAL(handled)；

2）cli()， sti()， save_flags()， 和 restore_flags()不再有效，应该使用 local_save_flags() 或 local_irq_disable()。

3）synchronize_irq()函数有改动

4）新增 int can_request_irq(unsigned int irq，unsigned long flags)；

5）request_irq() 和 free_irq() 从 <linux/sched.h>改到了 <linux/interrupt.h>

23.异步 I/O(AIO)<linux/aio.h>

ssize_t (*aio_read) (struct kiocb *iocb， char __user *buffer，size_t count， loff_t pos)；

ssize_t (*aio_write) (struct kiocb *iocb， const char __user *buffer，size_t count， loff_t pos)；

int (*aio_fsync) (struct kiocb *， int datasync)；

新增到了 file_operation 结构中。

is_sync_kiocb(struct kiocb *iocb)；

int aio_complete(struct kiocb *iocb， long res， long res2)；

24. 网络驱动

1）struct net_device *alloc_netdev(int sizeof_priv， const char *name，void (*setup)(struct net_device *))；

struct net_device *alloc_etherdev(int sizeof_priv);

2）新增 NAPI(New API)

void netif_rx_schedule(struct net_device *dev);

void netif_rx_complete(struct net_device *dev);

int netif_rx_ni(struct sk_buff *skb);(老版本为 netif_rx())

25. USB 驱动

老版本 struct usb_driver 取消了，新的结构体为

struct usb_class_driver {

char *name; struct file_operations *fops;

mode_t mode; int minor_base;

};

int usb_submit_urb(struct urb *urb， int mem_flags);

int (*probe) (struct usb_interface *intf，const struct usb_device_id *id);

26. mmap ()

int remap_page_range(struct vm_area_struct *vma， unsigned long from，unsigned long to， unsigned long size，pgprot_t prot);

int io_remap_page_range(struct vm_area_struct *vma， unsigned long from，unsigned long to， unsigned long size，pgprot_t prot);

struct page *(*nopage)(struct vm_area_struct *area，unsigned long address，int *type);

int (*populate)(struct vm_area_struct *area，unsigned long address，unsigned long len， pgprot_t prot， unsigned long pgoff，int nonblock);

int install_page(struct mm_struct *mm， struct vm_area_struct *vma，unsigned long addr， struct page *page，pgprot_t prot);

struct page *vmalloc_to_page(void *address);

27. 高端内存操作 kmaps

void *kmap_atomic(struct page *page， enum km_type type);

void kunmap_atomic(void *address， enum km_type type);

```
struct page *kmap_atomic_to_page(void *address);
```

第4章 基于S3C2410的嵌入式平台的设计

4.1 概 述

当前投入使用的嵌入式LINUX系统都是以2.4内核为基础，运行于ARM7系列的芯片之上。虽然，2.6内核已经面世很长时间了，但是由于内核变化比较大，且了解新内核具体细节的技术人员比较少。所以，2.6内核并没有马上在业界流行起来。然而，凭借着对嵌入式系统的种种优化，我相信基于2.6内核的嵌入式系统一定会在未来嵌入式应用中占领先地位。

嵌入式应用中主要使用的ARM7系列芯片，这其中的主要原因是ARM9系列芯片价格比较昂贵。但是，随着半导体技术的不断进步，ARM9系列芯片已经越来越便宜，致使ARM7系列芯片已经停产。因此，为了在将来激烈的市场竞争中处于领导者地位，目前许多嵌入式系统开发商已经着手进行硬件系统更新换代，软件系统升级。

嵌入式LINUX的未来发展方向已经比较明确，即运行于ARM9系列CPU上的以2.6内核为核心的嵌入式LINUX。目前许多知名厂商已经投入大量人力物力，以实现基于2.6内核的手机，路由器，工业以外网控制器等设备。

4.2 设计要求

1. 实现2.6内核在S3C2410上的移植。

2. 为了发挥LINUX的最大优势—网络功能强大，实现基于2.6内核的CS8900A以太网卡驱动程序。

3. 由于大容量闪存设备的广泛使用，在NAND闪存上移植YAFFS

3. 由于大容量闪存设备的广泛使用，在NAND闪存上移植YAFFS
文件系统。

4.3 硬件平台设计

4.3.1 ARM 系列 CPU 的特性

ARM（Advanced RISC Machines），既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通称，还可以认为是一种技术的名字。1991年ARM公司成立于英国剑桥，主要出售芯片设计技术的授权。目前，采用ARM技术知识产权（IP）核的微处理器，即我们通常所说的ARM微处理器，已遍及工业控制、消费类电子产品、通信系统、网络系统、无线系统等各类产品市场，基于ARM技术的微处理器应用约占据了32位RISC微处理器75%以上的市场份额，ARM技术正在逐步渗入到我们生活的各个方面。ARM公司是专门从事基于RISC技术芯片设计开发的公司，作为知识产权供应商，本身不直接从事芯片生产，靠转让设计许可由合作公司生产各具特色的芯片，世界各大半导体生产商从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微处理器芯片进入市场。

1.ARM 微处理器的主要应用

- 工业控制领域：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。
- 无线通讯领域：目前已有超过85%的无线通讯设备采用了ARM技术，ARM以其高性能和低成本，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。
- 消费类电子产品：ARM技术在目前流行的数字音频播放器、数字机顶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.ARM 微处理器的特点

ARM芯片具有RSIC体系的一般特点，如：

- 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体系还采用了一些特别的技术，在保证高性能的同时尽量减小芯片体积，减低芯片的功耗。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移位处理
- 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

3.ARM 微处理器系列

- ARM7系列
- ARM9系列
- ARM9E系列
- ARM10E系列
- SecurCore系列
- Intel的Xscale
- Intel的StrongARM

其中，ARM7、ARM9、ARM9E和ARM10为4个通用处理器系列，每一个系列提供一套相对独特的性能来满足不同应用领域的需求。

4.ARM 微处理器结构

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并非只是简单地去减少指令，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；以控制逻辑为主，不用或少用微码控制等措施来达到上述目的。

到目前为止，RISC 体系结构也还没有严格的定义，一般认为，RISC 体系结构应具有如下特点：

- 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。
- 使用单周期指令，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效率。

现代的 CPU 多采用 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。

4.3.2 ARM920T 处理器的特点

ARM920T 高速缓存处理器，是 ARM9Thmub 高性能 32 位片上系统处理器系列中的一款。他提供了一个完全的高性能 CPU 系统。包括：

- ARM9TDMI RISC 整型 CPU
- 16K 字节的指令缓存和 16K 字节的数据缓存
- 指令和数据存储管理单元
- 写缓冲功能
- 高级微处理总线结构（AMBA）总线接口

◆ 嵌入式跟踪宏单元（ETM）接口

ARM920T 内核的 ARM920TDMI 可以执行 32 位的 ARM 指令集和 16 位的 Thumb 指令集。ARM920TDMI 处理器是哈佛结构的处理器。

ARM920T 处理器的主要特点是：

◆ 基于 ARM920TDMI，ARMv4T 结构
◆ 具有两种指令集，包括 ARM 高性能 32 位指令集和 Thumb 高代码率的 16 位指令集
◆ 5 级流水线指令结构：取指令（F），指令译码（D），执行（E），数据存储器访问（M）和写寄存器（W）。
◆ 16K 字节的数据缓存，16K 字节的指令缓存
◆ 具有写缓冲器，可以实现 16 字的数据缓冲，以及 4 字地址的地址缓冲
◆ 标准的 ARMv4 存储单元管理（MMU），支持按区访问
◆ 8 位，16 位和 32 位的数据总线，支持指令和数据传输

4.3.3 ARM9 系列内核的优势

当前的嵌入式应用中主要使用的 ARM7 系列芯片，这其中的主要原因是 ARM9 系列芯片价格比较昂贵。但是，随着半导体技术的不断进步，ARM9 系列芯片已经越来越便宜，致使 ARM7 系列芯片已经停产。

1.工作频率

ARM7 系列微处理器的典型速度为 0.9MIPS/MHz，常见的 ARM7 芯片主时钟为 20MHz-133MHz。而 ARM9 系列微处理器的典型处理速度是 1.1MIPS/MHz。常见的 ARM9 芯片主时钟为 100MHz-233MHz。对于嵌入式应用中，CPU 主时钟越快，单位时间中所能完成的任务越多，则系统的集成度将越高。

2.操作系统的选择

以 ARM7TDMI 为内核的芯片，都没有 MMU，只能使用

UCLINUX，而不能使用更加稳定，支持更多的标准 LINUX。ARM9 系列芯片相对于 ARM7 系列芯片的最大优势就是配备了 MMU。这个在 ARM9 系列芯片上可以移植标准 LINUX。使用标准 LINUX 的优势在于，当前在 PC 上使用的应用程序可以方便的移植到嵌入式系统中。而使用 UCLINUX 的嵌入式系统只能重新进行应用程序的编写。

3.ARM9 的 5 级流水线

ARM9TDMI 内核将 ARM7TDMI 的功能显著提高到更高，更强的水平。最显著的区别是流水线从 3 级增加到 5 级。同时具有分开的指令和数据存储器，减少了在每个时钟周期内必须完成的最大工作，进而允许使用更高的时钟频率。

5 级流水线具体如下：

- 取指：从存储器中取出指令，并将其放入指令流水线
- 译码：对指令进行译码
- 执行：把一个操作数移位，产生 ALU 的结果
- 缓冲/数据：如果需要，则访问数据存储器；否则 ALU 的结果只是简单的缓冲一个时钟周期，以便于所有的指令具有同样的流水线流程
- 回写：将指令产生的结果回写到寄存器堆，包括任何从存储中读取的数

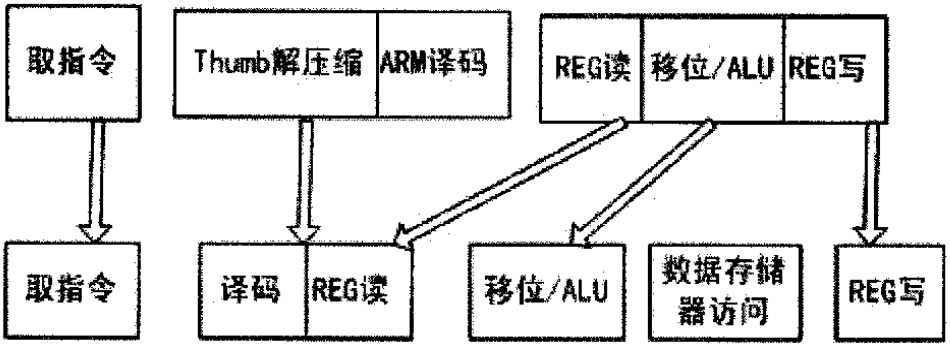


图 4.1 3 级流水线和 5 级流水线的对比

4.3.4 S3C2410 微控制器的特点

三星公司推出的 16/32 位 RSIC 处理器 S3C2410X，为手持设备和一般类型应用提供了低价格，低功耗，高性能小型微处理控制器的解决方案。为了降低整个系统的成本，S3C2410A 提供了一下丰富的内部设备：

- 一个 LCD 控制器（支持 STN 和 TFT 带有触摸屏的液晶显示器）
- SDRAM 控制器
- 3 个通道的 UART
- 4 个通道的 DMA
- 4 个具有 PWM 功能的计时器和 1 个内部时钟
- 8 通道的 10 位 ADC
- 触摸屏接口
- I2S 总线接口
- 2 个 USB 主机接口。1 个 USB 设备接口
- 2 个 SPI 接口
- SD 接口和 MMC 卡接口
- 看门狗计数器
- 117 位通用 I/0 和 24 位外部中断源
- 8 通道 10 位 AD 控制器

在时钟方面 S3C2410 也有突出的特点，该芯片集成了一个具有日历功能的 RTC 和具有 PLL（MPLL 和 UPLL）的芯片时钟发生器。MPLL 产生主时钟，能够使处理器工作频率最高达到 203MHZ。这个工作频率能够使处理器轻松的运行于 Windows CE，linux 等操作系统以及进行复杂的信息处理。UPLL 产生主从 USB 功能的时钟。

S3C2410 将系统的存储空间分成 8 组（Bank），每组的大小位 128MB，共 1GB。Bank0 到 Bank5 的开始地址是固定的，用于 ROM 说是 SRAM。Bank6 和 Bank7 用于 ROM，SRAM 或是 SDRAM，这两组可编程且大小相同。Bank7 的开始地址是 Bank6 的结束地址，灵活可变。所有内存块的访问周期都可编程。S3C2410 采用 Ngcs[7:0]8

个通用片选信号选择这些组。

S3C2410 支持从 NAND 闪存启动，系统采用 NAND 闪存和 SDRAM 组合，可以获得非常高的性价比。S3C2410 具有三种启动方式，可通过 OM[1:0]管脚进行选择：

- OM[1:0]=00 时，处理器从 NAND 闪存启动
- OM[1:0]=01 时，处理器从 16 位宽的 ROM 启动
- OM[1:0]=10 时，处理器从 32 位宽的 ROM 启动

用户可以将引导代码和操作系统镜像存放在外部的 NAND 闪存中，并从 NAND 闪存启动。当处理器在这种模式下电复位时，内置的 NAND 闪存将访问控制接口，并将引导代码自动加载到内部 SRAM（此时刻该 SRAM 定位于起始地址空间 0X00000000，容量为 4KB）并且运行。之后，SRAM 中的引导程序将操作系统镜像加载到 SDRAM 中，操作系统就能够在 SDRAM 中运行。启动完毕后，4KB 的启动 SRAM 就可以用于其他用途。如果从其他方式启动，启动 ROM 就要定位于内存的起始地址空间 0X00000000，处理器直接在 ROM 上运行启动程序，而 4KB 的启动 SRAM 被定位于内存地址的 0X40000000 处。

S3C2410 对于片内的各个部件采用了独立的电源供电方式：

- 内核采用 1.8v 供电
- 存储单元采用 3.3V 独立供电，对于一般 SDRAM 可以采用 3.3V
- V_{DDQ} 等于 3.0/3.3V
- I/O 采用独立 3.3V 供电

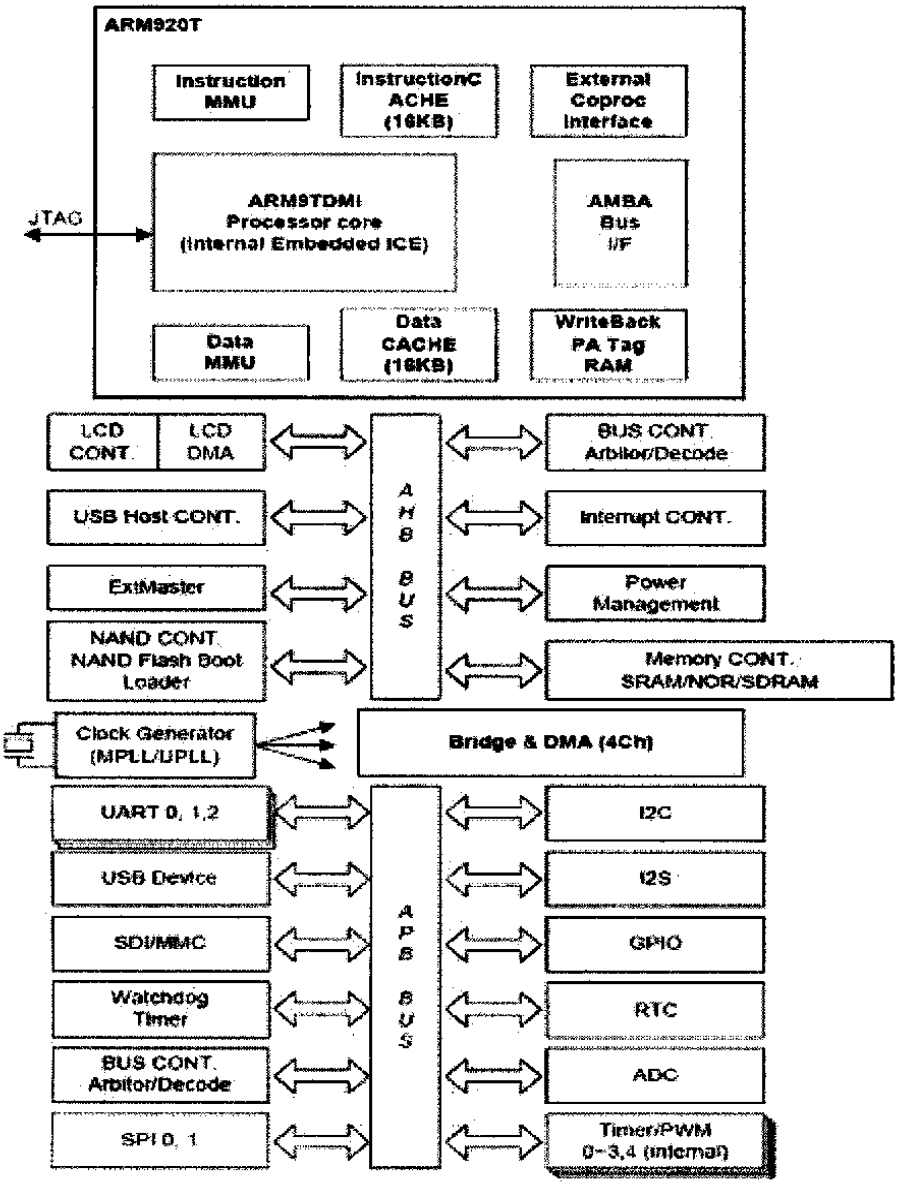


图 4.2 S3C2410 结构框图

4.3.5 硬件平台总体框架

三星公司的 S3C2410 微控制器具有强大的功能，支持丰富的外围设备，因此，所需的外围器件很少。基于 S3C2410 微控制器的嵌入式

平台包括以下功能模块：

1）存储器

- 1MB 的 AMD NOR 闪存
- 64MB 的三星 NAND 闪存
- 32M×2 的现代 SDRAM

2）局域网

- CS8900A 的 10M 以太网控制芯片
- RJ45 接口

3）USB（遵从 1.1 协议）

- 1 个 USB Host A 型接口
- 1 个 USB Device B 型接口

4）音频

- UDA1341 音频控制芯片
- 一个音频输出接口（双声道）
- 一个音频输入接口

5）时钟电路

- 32.768kHZ 标准晶振，用于实时时钟
- 20MHZ 标准晶振，用于以外网物理层收发器
- 12MHZ 标准晶振，用于微处理器的主时钟，或是 USB 设备运行时钟

6）复位电路

- 复位控制器，提供上电复位和手动复位

7）LED

- 4 个可编程用户 LED

9）键盘

- 4 个可编程用户按键

10）电源

- 一个开关电源＋5V 供电

11）通信

- 用于串行调试的 RS232 DB9 两线串口，并且可直接引出 CPU 内部三串口

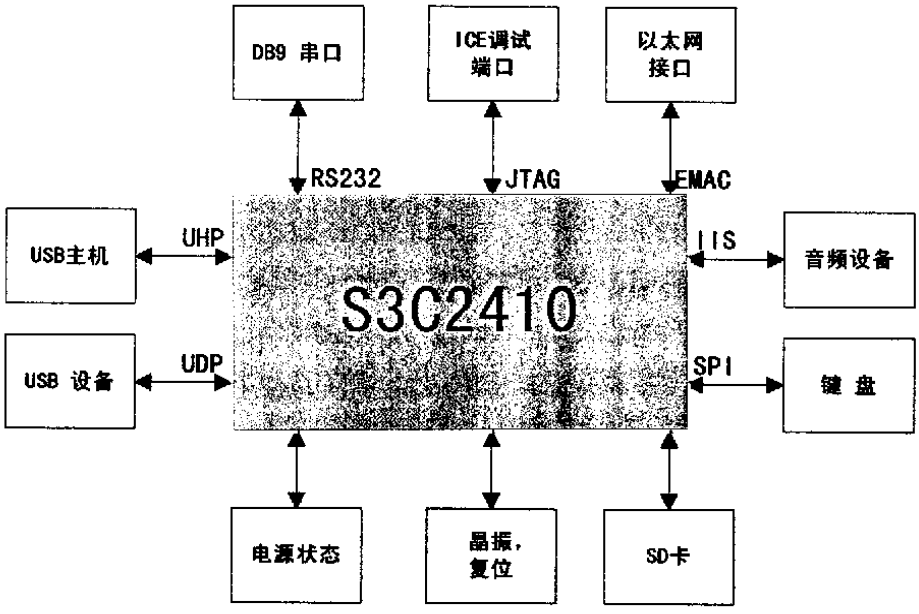


图 4.3 3C2410 嵌入式平台的原理框图

4.3.6 功能组件简介

复位逻辑

nRESET（系统复位信号）必须保持低电平 4 个 CLKS，以保证信号的复位信号的正确识别。在 nRESET 信号和内部 nRESET 信号之间要花费 128CLKS。nRESET 和 nTRST(JTAG 复位信号)之间通过 4.7K 的电阻相连。

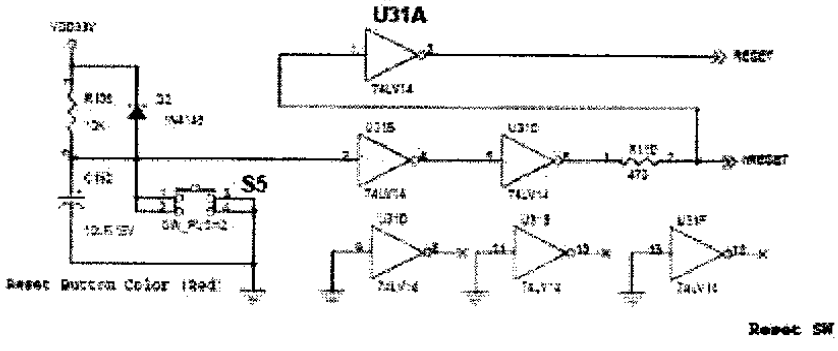


图 4.4 复位逻辑电路

电源供应

S3C2410微处理器的嵌入式平台需要两种供电电源：1.8V，3.3V。其中，1.8V供ARM内核使用，3.3V供I/O端口和外设使用。本系统中采用，将5V电压通过LM1117-33转化为3.3V，以及通过LM1117-18转换为1.8V的方式实现电压变化。

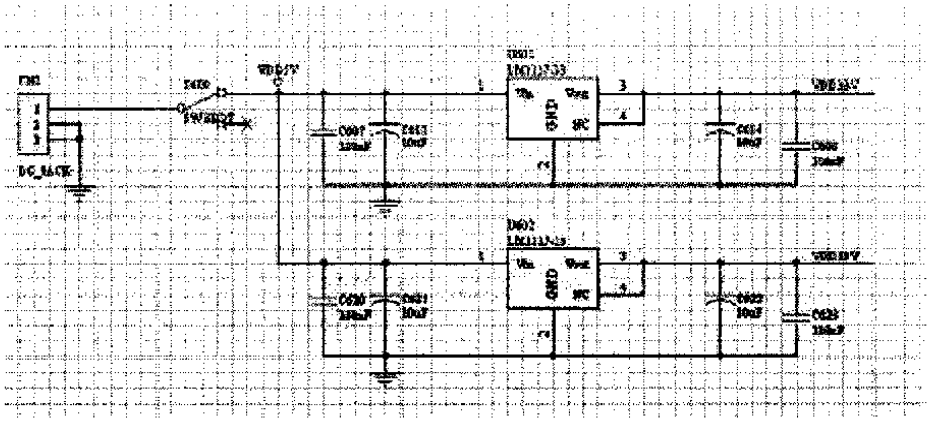


图 4.5 电源供应电路

网络设备

CS8900A是CIRRUS LOGIC公司生产的16位以太网控制器，芯片内嵌片内RAM，10BASE-T收发滤波器，直接ISA总线接口。特点是使用灵活，其物理层接口，数据传输模式和工作模式等都能根据需要动态调整，通过内部寄存器的设置来使用不同的应用环境。

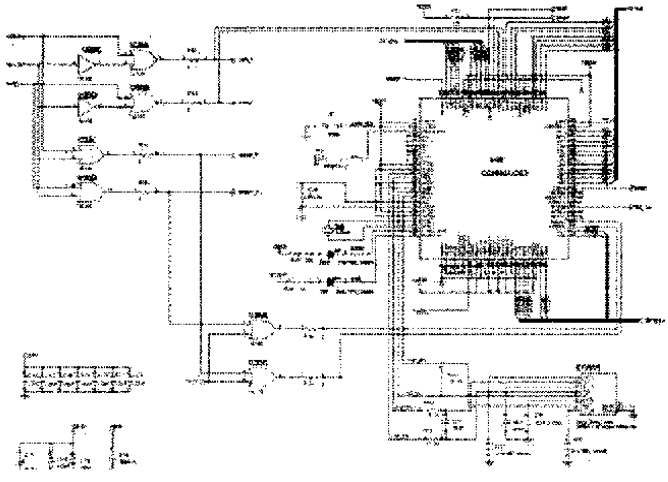


图 4.6 CS8900A 电路连接图

4.4 软件平台构建

由于大量软件精英的不懈努力，LINUX 在嵌入式系统中的应用越来越广泛。2.6 内核中的大量新特性，使其更加适合于嵌入式系统。正是基于尝试和创新的考虑，本系统采用了 2.6.11 内核以及 YAFFS 文件系统。本系统的硬件电路是基于 SMDK2410 的，其相关知识点，已在其他详细介绍。

下面主要介绍软件环境的移植工作。

1.获得 LINUX 内核源文件以及其相应补丁

LINUX 内核的官方网站是 www.kernel.org。有关于内核的任何变化，更新都已此网站为准。

2.获得交叉编译工具

交叉编译工具主要用来从PC机上产生可以在ARM架构上运行的二进制代码。我们可以从 www.handheld.org 找到与 2.6.11 内核向适应的交叉编译工具。本系统使用 arm-linux-gcc-3.4.1.tar.bz2 作为我们的交叉编译器。

3.获得 YAFFS 源代码

有关于 YAFFS 的技术细节可参考相关章节。可以在 http://husaberg.toby-churchill.com/balloon/releases/v0.7/base/armv4l/ 处获得所需的源代码。

4.开始移植

解压缩所获得源文件，并打上相应的补丁。我们假设所有的文件都保存在/home/yourname 目录下。

```
#cd /home/yourname
#tar –zxvf linux-2.6.11.12.tar.gz
#cp –a ./linux-2.6.11.12 /usr/src
#cp patch-2.6.11.12.tar.gz /usr/src/
```

#mkdir /usr/local/arm/

#tar –jxvf arm-linux-gcc-3.4.1.tar.bz2

#cp –a ./3.4.1 /usr/local/arm/

#tar –zxvf yaffs.tar.gz

#cp –a ./yaffs /usr/src/linux-2.6.11.12/fs/yaffs

#cd /usr/src/linux-2.6.11.12/

#zcat../patch-2.6.11.12.tar.gz | patch –p1 –f

5.修改 2.6.11 内核相关文件内容

进入 2.6.11 根目录，然后修改内核源文件。

1）修改 usr/src/linux-2.6.11.12/arch/arm/mach-s3c2410/devs.c 文件

#cd usr/src/linux-2.6.11.12/arch/arm/mach-s3c2410/

#vi devs.c

在此文件中加入你的 nand 闪存分区信息。

2）修改 arch/arm/mach-s3c2410/mach-smdk2410.c

在此文件的__initdata 中，添加“&s3c_device_nand”相关内容

3）关闭 ECC。

修改 driver/mtd/nand/s3c2410.c，将其中的 chip->eccmode = NAND_ECC_SOFT，改为 NAND_ECC_NONE.

6.修改 YAFFS 源文件中的相关内容

#cd /usr/src/linux-2.6.11.12/fs/yaffs/linux-kernel/

./patch-ker.sh

#cd ./fs/yaffs

#cp. /Makefile /usr/src/linux-2.6.11.12/fs/yaffs/

在 yaffs_guts.c， yaffs_mtdif.c， yaffs_ecc-c 以及 yaffs_fs.c 中添加“#include <linux/config.h>”

7.配置内核

#cd /usr/src/linux-2.6.11.12/

#vi Makefile

我们必须修改“ARCH=”和“CROSS_COMPILE=”的相关内容。

把“SUBARCH:=”改写成“SUBARCH:=arm”，“CROSS_COMPILE”改写成“CROSS_COMPILE :={你的交叉编译器安装路径}”。使用#make menuconfig 命令根据硬件条件配置内核。一般情况下，使用默认配置即可。

8.编译内核

使用#make zImage 命令编译内核。如果编译成功的话，可以在/usr/src/linux-2.6.11.12/arch/arm/boot/处获得新编译成功的内核。

9.修改 VIVI bootloader

由于本系统不是 PC 机，不具有 BIOS，所以需要使用 bootloader 来初始化 2410 和闪存，并将 LINUX 内核从闪存中导引至内存中。

1）从 MIZI 公司网站获得 VIVI 源文件，并将其放入/home/yourname 子目录中。执行下列命令：

#cd vivi

#vi Makefile

2）将“LINUX-INCLUDE-DIR =”修改成“LINUX-INCLUDE-DIR = {本系统交叉编译器的 include 路径}”。注意，此时的交叉编译器版本与内核源代码不同，需要根据 VIVI 的要求选择合适的编译器。本系统为“LINUX-INCLUDE-DIR = /usr/local/arm/2.95.3/include”

3）将“CROSS-COMPILE =”修改成“CROSS-COMPILE = usr/local/arm/2.95.3/bin/arm-linux-”。

4）修改内核启动时使用的命令行初始参数。

本系统使用的参数是：“char linux_cmd[] = "noinitrd root=/dev/mtdblock/3 init=/linuxrc console=ttySAC0 , 115200 mem=64M。

5）返回到 VIVI 的根目录，根据硬件情况运行#make menuconfig 命令配置 VIVI。

6）使用#make vivi 命令编译 VIVI 源文件。如果一切顺利，在根目录处可以获得编译成功的 VIVI 二进制代码。将 VIVI 烧写在 NAND 闪存的地址 0 处。

第 5 章 LINUX2.6 内核的网络设备驱动程序

5.1. 网络设备驱动程序简介

LINUX 系统将设备分成三种基本类型：字符设备，块设备，网络设备。字符设备是个能像字符流一样被访问的设备，主要特点是不支持随机访问。块设备是个能够容纳文件系统的大容量存储设备，主要特点是一次可随机输入输出大量数据。网络设备是个能和其他主机交换数据的设备，主要处理各种网络事务。

设备驱动程序是内核控制硬件设备，硬件设备向内核反馈信息的媒介。设备驱动程序在 LINUX 内核中扮演着特殊的角色。他们是一个个独立的"黑盒子"，使某个特定硬件响应一个定义良好的内部编程接口，这些接口完全隐藏了设备的工作细节。用户的操作通过一组标准化的调用执行，而这些调用独立于特定的驱动程序。将这些调用映射到作用于实际硬件设备的特有操作上，则是设备驱动程序的任务。

区分机制和策略是 UNIX 设计背后隐含的最好思想之一。大多数编程问题实际上都可以分成两部分："需要提供什么功能"（机制）和"如何使用这些功能"（策略）。如果这里两个问题由程序的不同部分来处理，或者甚至由不同程序来处理，则这个软件包更易开发，也更容易根据需要来调整。一个典型的例子就是具有分层结构的 TCO/IP 网络：位于下层的操作系统负责提供套接字抽象层，但在所传输的数据上则没有附加任何策略；上面各层的服务器则分别提供不同的服务（以及相关策略）。

驱动程序同样存在机制和策略的分离问题。例如，软驱的驱动程序不带策略，它的作用就是将磁盘表示成一个连续的数据块阵列。系统高层负责策略，比如谁有权力访问软盘驱动器，是直接访问驱动器还是通过文件系统，以及用户是否可以在驱动器上挂装文件系统等等。既然不同的环境通常需要不同的方式来使用硬件，我们应该尽可能做到让驱动程序不带策略。所以，在编写驱动程序时，程序员要特别注意这个基本概念：编写访问硬件的内核代码时，不要给用户强加

任何特定策略。

从另外一个角度来看驱动程序，它可以看成是应用程序和实际设备之间的一个软件层。驱动程序的这种特权角色可让编写者选择如何展现设备特性，也就是说，即使对于相同的设备，不同的驱动程序可能提供不同的功能。

总的来说，驱动程序设计主要还是考虑下面三个方面的因素：提供给用户尽量多的选项，编写驱动程序要占用的时间以及尽量保持程序简单而不至于错误丛生。

5.2. 网络设备驱动程序

5.2.1 网络设备驱动程序的概念及其作用

网络设备由内核中的网络子系统驱动，负责发送和接受数据包，但它不需要了解每个事务如何映射到实际传送的数据包。许多网络连接是面向流的，但网络设备却是围绕数据包的传输和接受而设计的。

系统中网络设备的角色，和一个已挂装的块设备类似。一个块设备向内核注册其磁盘和函数，然后使用其请求函数，根据请求“发送”和“接受”块数据，与此类似，网络接口也必须使用特定的内核数据结构注册自己，以备与外界进行数据包交换时调用。但是这两种设备之间最重要的不同是：块设备只响应来自内核的请求，而网络设备异步地接受来自外部世界数据包，因此当内核要求一个块设备驱动程序向其发送缓冲区数据时，而网络设备则向内核请求把外部获得地数据包发送到内核。

LINUX 网络设备驱动程序是 LINUX 网络应用的重要组成部分，所有的 LINUX 网络驱动程序遵循通用的接口。在内核启动时，系统通过网络设备驱动程序登记已经存在的网络设备。设备用标准的支持网络的机制来把收到的数据转送到相应的网络层。所有被发送和接受的数据包都用数据结构 sk_buff 表示。这是一个具有良好灵活性的数据结构，可以很容易的增加或删除网络协议数据包的首部。

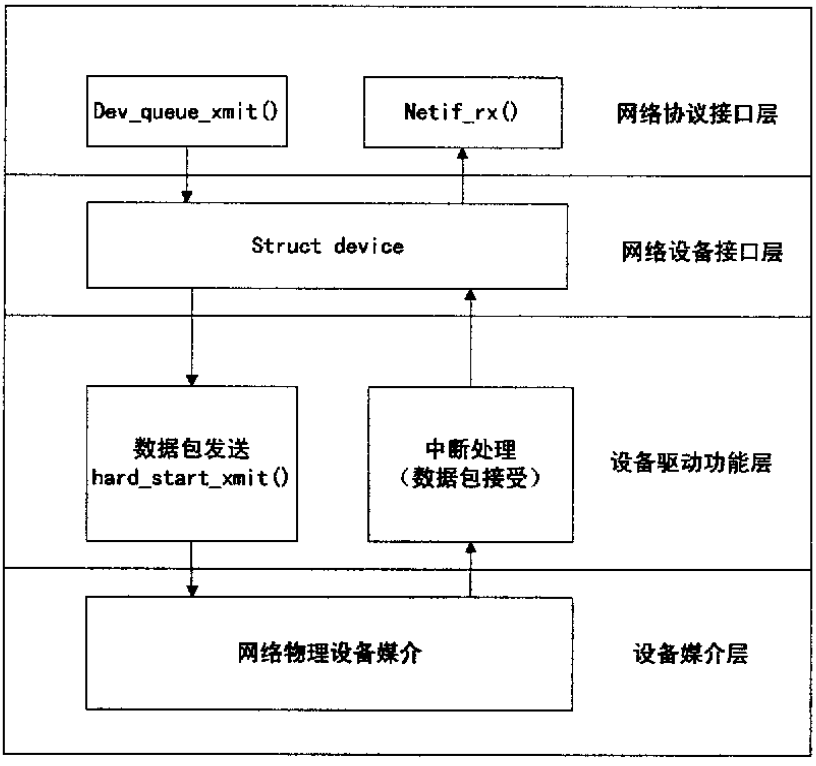


图 5.1 LINUX 网络设备驱动程序的体系结构

5.2.2 网络设备驱动程序的结构

1.设备注册

驱动程序对每个新检测到的接口，向全局网络设备链表中插入一个数据结构。每个接口由一个 net_device 结构描述，其定义在<linux/netdevice.h>。和其他所有内核数据结构一样，net_device 包含了一个 kobject 和引用计数，并且通过 sysfs 导出信息。由于与其他结构相关，因此它必须被动态分配。用来执行分配的内核函数是 alloc_netdev，原型如下：

```
struct net_device *alloc_netdev(int sizeof_priv,  const char *name,
                                void (*setup) (struct net_device *));
```

这里 sizeof_priv 是驱动程序的"私有数据"区的大小；这个区成员和 net_device 结构一同分配给网络设备。Name 是接口的名字，其在

用户空间可见。Setup 是一个初始化函数，用于设置 net_device 结构剩余的部分。

网络子系统针对 alloc_netdev 函数，为不同种类的接口封装了许多函数。最常用的是 alloc_etherdev，它在<linux/etherdevice.h>中定义：

Struct net_device *alloc_etherdev(int sizeof_priv)；此函数使用 eth%d 的形式制定分配的网络设备的名字。它提供了自己的初始化函数（ether_setup），用正确的值为以太网设备设置 net_device 中的许多成员。因此在驱动程序中没有为 alloc_etherdev 提供初始化函数；驱动程序只是在成功分配"私有数据"区后，直接作一些必须的初始化工作。

一旦 net_device 结构被初始化后，剩余的工作就是将给结构传递给 register_netdev 函数。

2.模块卸载

模块的清除函数只是注销接口，完成一些需要在清除函数中完成的事情：释放申请的 DMA，IRQ，内存等系统资源；然后释放 net_device 给系统。Unregister_netdev 函数从系统中删除了接口，free_netdev 函数将 net_device 结构返还给了系统。

3.设备的打开和关闭

驱动程序可在装载阶段或内核引导阶段探测接口。但是在接口能够传送数据包之前，内核必须打开接口并赋予其地址。内核可在响应 ifconfig 命令时打开或关闭一个接口。对实际代码来说，驱动程序必须提供一些基本的可执行函数：open 打开接口，stop 停止接口，hard_start_xmit 初始化数据包的传输，hard_header 根据先前检索到的源和目标硬件地址建立硬件头，rebuild_header 用于在传输数据包之前，完成 ARP 解析之后，重新建立硬件头，tx_timeout 负责解决丢包故障并重新建立数据包，get_stats 当应用程序需要获得接口的统计信息时，将调用该函数，set_config 改变接口配置，此函数是配置驱动程序的入口点。但是除此之外，还要执行其他一些步骤。

首先，在接口能够和外界通讯之前，要将硬件地址（MAC）从硬件设备复制到 dev->dev_addr。硬件地址可在打开期间拷贝到设备

中。一旦准备好开始发送数据后，open 函数还应该启动接口的传输队伍，内核提供如下函数可启动给队伍：

Void netif_start_queue(struct net_device *dev)

相对应，子阿接口被关闭时，调用 net_stop_queue（）函数来标示设备不能传输其他数据包。

4.数据包的传输

传输指的是将数据包通过网络连接发送出去。无论何时内核要传输一个数据包，它都会调用驱动程序的 hard_start_transmit 函数将数据放入外发队伍。内核处理的每个数据包位于一个套接字缓冲区结构(sk_buff)中，该结构定义在<linux/skbuff.h>中。Sk_buff 包含了物理数据包，并拥有完整的传输层数据包头。接口无需修改传输的数据。Skb->data 指向要传输的数据包，sk->len 是已 8 字节为单位的长度。

5.数据包的接受

从网络上接受数据要比传输数据复杂一点，因为必须在原子上下文中分配一个 sk_buff 并传递给上层处理。网络驱动程序实现了两种模式接受数据包：中断驱动方式&轮询方式。大多数驱动程序使用了中断驱动技术。

当硬件接受到数据包之后，会通过中断服务程序来调用数据包接受处理函数（rx）。Rx 函数接受一个指向数据的指针，以及数据包的长度，它负责将数据包以及其他附加信息发送到上层的网络程序。这个 rx 函数具有一个比较通用的模式。

第一步是分配一个保存数据包的缓冲区。Dev_alloc_skb 以原子的优先权调用 kmalloc，因此可在中断期间安全使用，它需要知道数据长度。一旦拥有一个合法的 skb 指针，则调用 memcpy 将数据包数据拷贝到缓冲区内。Skb_put 函数刷新缓冲区内的数据末尾指针，并且返回新创建数据区的指针。在能够处理数据包之前，网络层必须知道数据包的一些信息。为此必须在将缓冲区传递到上层之前，对 dev 和 protocol 成员正确赋值。以太网支持代码导出了辅助函数 eth_type_trans，用来查找填入 protocol 中的正确值。然后需要指定如何求得校验和，或者已经在数据包上求得了校验和。最后，驱动程序

更新其统计计数器，以记录已接受到的每个数据包。接受数据包过程中的最后一个步骤由 netif_rx 执行，它将套接字缓冲区传递给上层软件处理。

5.3.以太网控制器 CS8900A

CS8900A 是 CIRRUS LOGIC 公司生产的 16 位以太网控制器。是真正的单芯片，全双工以太网解决方案，集成了以太网电路所需的全部模拟数字器件。主要功能部件包括：ISA 总线接口，802.3MAC，内嵌 RAM 缓冲区，串行 EEPROM 接口，RJ45 接口。

5.3.1 主要特性：

- 最大工作电流 55mA
- 边沿扫描和回环测试
- 3V 供电电压
- 工业级温度范围
- 待机&睡眠模式
- 可编程发送功能
- 数据碰撞自动重发
- 自动打包及生成 CRC 校验码
- 可编程接受功能
- 数据流降低 CPU 消耗
- 自动切换于 DMA 和片内 RAM
- 提前产生中断便于数据帧处理
- 自动阻断错误包
- 可跳线控制 EEPROM 功能
- LED 驱动器用于指示连接状态和网络活动情况

5.3.2 工作原理

收到由主机发过来的数据包（从目的地址域到数据域）后，侦听网络线路。如果线路忙，它就等到线路空闲为止。发送过程中，首先添加以太网帧头（包括先导字段和帧开始标志），然后产生 CRC 校验码，最后将此数据帧发送到以太网上。接收时，它将以太网收到的数据帧在经过解码，去掉帧头和地址检验等步骤后缓存在片内。通过 CRC 校验后，它会根据初始化配置情况，通知主机 CS8900A 收到了数据帧，最后用上面介绍的某种传输模式传到主机的存储区中。

CS8900A 可以在内存模式和 I/O 模式下操作。当配置成内存模式操作时，CS8900A 的内部寄存器和帧缓冲区映射到主机内存中连续的 4KB 的块中，主机可以通过这个块之间访问 CS8900A 的内部寄存器和帧缓冲区。

第6章 适用于闪存的YAFFS文件系统

随着闪存技术的不断进步，闪存的价格越来越低。闪存这个曾经高高在上的“贵族”，现在也活跃在我们的日常生活中。正是由于闪存的存在，才真正实现了便携设备的便携性。

本课题的试验板上配备了两块闪存，分别是NOR闪存和NAND闪存。其中NAND闪存有64MB容量，完全可以胜任各种嵌入式应用的存储要求。如此大的容量，必须移植文件系统，只有这样才能最大化的发挥闪存的存储能力。但是，由于闪存本身的技术特点所限制，普通PC上所使用的FAT32等文件系统并不适用。本章将对适用于闪存的YAFFS文件系统进行讨论。

6.1 闪存简介

闪存是一种能够长期存储数据的设备。即使在不加电的情况下，数据也不会丢失。和磁设备相比，闪存在体积，抗震性，耗电量等几个方面都具有很大优势，因此成为嵌入式系统的首选存储设备。市面上的闪存芯片一般有以下两种:传统的NOR型闪存，可以直接读取其芯片内存储的数据，因而速度比较快，但是价格比较高;NAND 型闪存，这种闪存也称为固态硬盘，他内部数据以块为单位进行存储，地址线和数据线共用，使用控制信号选择。

当前已经存在了许多闪存文件系统或是适用于闪存的块驱动程序。闪存具有很多技术上的限制，所有任何一个运用在闪存上的文件系统只能在这些限制内完成本职工作。更重要的是，应该清楚"flash"具有两种类型NOR和NAND。这两种类型的闪存具有不同的技术特点。由于都使用了比较通用的"flash"术语，所以很多人想当然的认为适用于NOR的驱动程序也可以适用于NAND闪存。

基于闪存的文件系统主要采取以下架构：

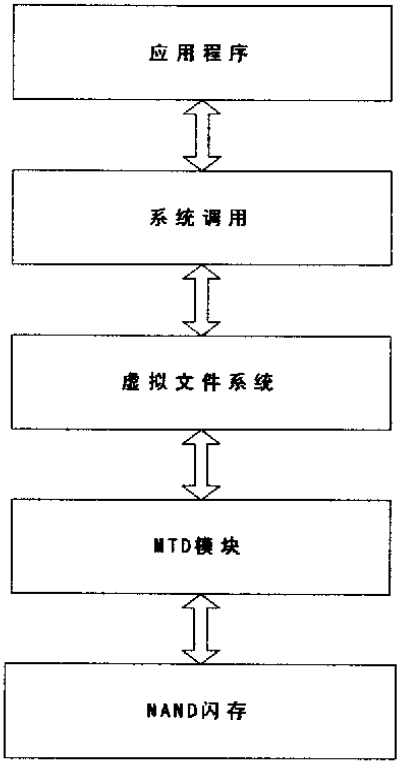


图 6.1 典型闪存文件系统架构

6.2 闪存的技术特点:

6.2.1 两种闪存的共同点

1.闪存的最小寻址单位是字节(byte)，而不是磁盘上的扇区(sector)。这意味着我们可以从一块闪存的任意偏移(offset)读数据，但并不表明对闪存写操作也是以字节为单位进行的。

2.当一块闪存处在干净的状态时（被擦写过，但是还没有写操作发生），在这块 flash 上的每一位(bit)都是逻辑 1。

3.闪存上的每一位(bit)可以被写操作置成逻辑 0。 可是把逻辑 0 置成逻辑 1 却不能按位(bit)来操作，而只能按擦写块（erase block）为单位进行擦写操作。擦写块的大小从 4K 到 128K 不等。从上层来

看，擦写所完成的功能就是把擦写块内的每一位都重设置（reset）成逻辑 1。

4.闪存的使用寿命是有限的。具体来说，闪存的使用寿命是由擦写块的最大可擦写次数来决定的。超过了最大可擦写次数，这个擦写块就成为坏块(bad block)了。因此为了避免某个擦写块被过度擦写，以至于它先于其他的擦写块达到最大可擦写次数，我们应该在尽量小的影响性能的前提下，使擦写操作均匀的分布在每个擦写块上。这个过程叫做磨损平衡（wear leveling）。

5.向闪存中写数据必须先将芯片中对应得内容清空，然后再写入，也就是通常说得"先擦后写"。

6.2.2 NOR 型闪存与 NAND 型闪存的不同之处：

1.NOR 型闪存读/写操作的基本单位是字节；而 NAND 型闪存又把擦写块分成页(page)， 页是写操作的基本单位，一般一个页的大小是 512 或 2K 个字节。对于一个页的重复写操作次数是有限制的，不同厂商生产的 NAND 型闪存有不同的限制，有些是一次，有些是四次，六次或十次。

2.按照现在的技术水平，一般来说 NOR 型闪存擦写块的最大可擦写次数在十万次左右，NAND 型闪存擦写块的最大可擦写次数在百万次左右。但是由于 NAND 型闪存擦写整个页面(或块)，块内得页面中如果有一位失效整个页面就会失效，而且由于擦写过程复杂，失败的概率相对较高，所以从整体上来说 NOR 型闪存的寿命较长。

3.NOR 型闪存是随机存储介质，适合存放小文件;NAND 型闪存是连续存储介质，适合存放大文件。

4.NOR 型闪存的地址线和数据线是分开的，可以像 SRAM 一样连在数据线上， 所以他的传输效率很高，可执行程序可以在芯片内执行，不必再将代码读到 RAM 中。因此，嵌入式系统中经常将 NOR 型闪存作为启动芯片。而 NAND 共用地址和数据总线，需要额外联结一些控制引脚，直接将 NAND 型闪存作启动芯片比较难。

5.NOR 型闪存可以对字进行操作，处理小数据量 I/O 时其速度快

于NAND型闪存;反之，NAND型闪存共用地址和数据总线，只能对一个固定大小的区域进行清零操作，处理大数据量时速度快于前者。

6.NOR型闪存的可靠性高于NAND内存。这是因为NOR型闪存的接口简单，数据操作少，位交换操作少，极少出现坏区块。相反，NAND型闪存接口复杂，操作繁琐，位交换也很多，出现问题的几率要大得多。

6.3 当前使用的闪存文件系统

不同类型的闪存设备根据其自身特点使用不同的文件系统。

1.NAND设备:一般采用FAT16作为其文件系统。此文件系统的鲁棒性不是很好。这种块驱动程序提供了一个逻辑到物理的映射层，以模拟可会写块使其实现硬盘扇区的相应功能。但是像所有的以FAT为基础的系统一样，他们比较容易崩溃。

2.NOR设备:基于NOR设备的LINUX多采用JFFS和JFFS2。这两种文件系统都是居于日志文件原理的，所以都显著的提高了系统的鲁棒性。但是，他们在系统启动时间和RAM消耗率的表现都不是很好。JFFS要求在RAM中为每一个闪存中的日志点都保存一个jffs_node数据结构，这个数据结构占有48个比特。JFFS2对这个数据结构进行了改进，使用16比特的jffs2_raw_node_ref来代替前者。然而，就是按照平均一个日志节点大小为512比特，一个128MB的NOR闪存也要使用4MB的RAM。

JFFS和JFFS2均要求在启动时候扫描整个闪存矩阵以寻找日志节点和决定文件结构。由于NAND具有容量大，读写缓慢，串行处理等性质，所以大大延长了整个系统的启动时间。一般的，扫描一个128MB的NAND型闪存大约使用25秒。

6.4 YAFFS的简介:

YAFFS意义为'yet another flash file system'，也是一个开源的文件系统。YAFFS是目前为止唯一一个专门为NAND flash设计的应用于

嵌入式系统的文件系统，具有很好的可移植性，能够在linux，uclinux，和wince等嵌入式操作系统中。

6.4.1 使用YAFFS的现实意义:

1.相对于NOR闪存，NAND闪存比较便宜，具有更快的擦写速度。

2.NAND的物理界面非常简单。

NAND外形小巧，并且工作电流很小，很适合于在嵌入式系统中使用。然而，在嵌入式系统中，电源可能被随时切断，这将导致数据丢失，甚至文件系统崩溃。所以应该使用YAFFS文件系统，由他来处理坏块和使用格式，从而防止整个系统崩溃。

YAFFS/YAFFS与JFFS2相比减少了一些功能，所以速度更快，并且对内存的占用较小。YAFFS自带NAND芯片驱动，而且为嵌入式系统提供了直接访问文件系统的API，用户可以不使用LINUX的MTD和VFS，直接去文件进行操作。YAFFS2是YAFFS的改进版本，在速度。内存使用上，对NAND设备的支持都有所改善。YAFFS2还支持大页面的NAND设备，并且对大页面的NAND设备作了优化。

6.4.2 YAFFS的技术特点:

1.YAFFS是为闪存设备设计的文件系统，提供了写平衡，垃圾收集等底层操作。

2.YAFFS借助了日志系统的思想，不提供日志机能，所以资源占用少。

3.YAFFS是NAND闪存专用的文件系统，充分利用了NAND设备的特性，所以在NAND设备上速度比较快。

4.YAFFS2不支持压缩功能，更适合存储容量大的系统。

5.YAFFS 中的文件是以页(chunk)为单位的，2n 个页面组成块(block)，其中chunk的大小和NAND闪存中页的大小相同。在文件系统加载的时候，只需要加载文件ID和页面号并且为两者建立映射表。

每个页面平均需要大约5B存放页面映射关系，在内存使用只有JFFS2的一般。

6.YAFFS不需要读取全部文件节点的信息，而是在需要读取某个文件的时候，找文件对应的ID号。在文件映射表中保存了从文件ID到页面索引的哈希表。在文件系统加载时，只要将哈希索引表加载到内存中就可以了。所以在文件系统被加载的时候，为了加载哈希表，需要读取页面的映射表。为了加快扫描时间并节约空间，页面的映射表存放在闪存芯片的OOB空间内。这样文件系统加载的时候只需扫描OOB空间(out-of--band空间)即可。读取速度大大加快。

7.YAFFS中因为使用了页面的映射表，所以如果文件被替换，则数据内容写入新的chunk中，并且修改映射关系，原来的页面被标记为"废弃"。

8.YAFFS使用多级链表管理需要回收的脏块，并且使用系统时间生成伪随机数决定要回收的块，通过这个方法能提供较好的写平衡。

总之，YAFFS是一个非常简单的文件系统，而且由于其耗费校对较少的系统RAM和较短的启动时间，所以他非常使用于在NAND型闪存上使用。当然YAFFS也是一种新的文件系统，所以他的很多代码都借鉴了JFFS文件系统。也正因为如此他的鲁棒型远远超过了FAT型文件系统。

6.5 将YAFFS作为内核文件系统

NAND型闪存多使用mtd+yaffs，所以我们将YAFFS装载成本项目的文件系统。

YAFFS官方网址http://www.aleph1.co.hk/armlinux/projects/。你可以从此处获得YAFFS的源代码。

按照下列步骤完成此项工作:

1.因为YAFFS需要内核的MTD支持，所以确保下载安装最新的MTD包。

2.在内核源代码树中创建[linux]/fs/yaffs目录。

3.将devextras.h，yaffs_fs.c，yaffs_guts.c，yaffs_guts.h，

yaffs_mtddif.c，　yaffs_mtdif.h，　yaffsinteface.h，　yportenv.h，yaffs_config.h，　Makefile 拷贝到上一步创建的目录中。

4.修改 Makefile 文件，修改为

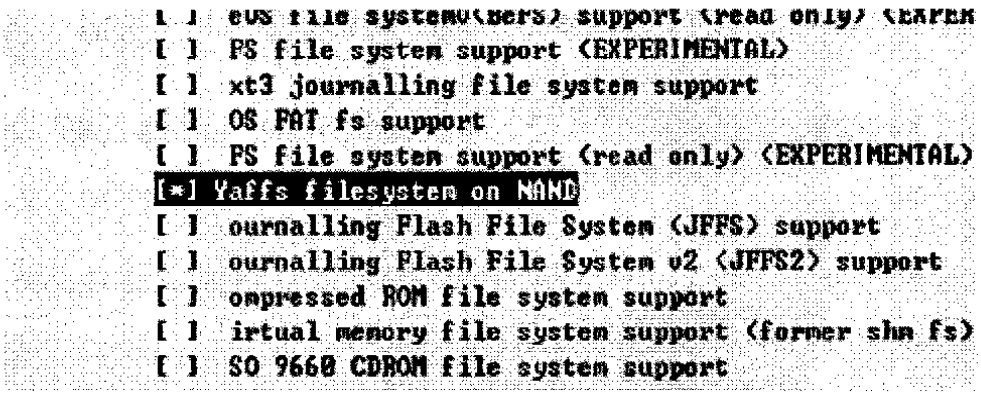
```
O_TARGET:= yaffs.o
Obj-y: =yaffs_fs.o yaffs_guts.o yaffs_mtdif.o yaffs_ecc.o
obj-m :=$(O_TARGET)
include $(TOPDIR)/Rules.make
```

5.修改[linux]/fs/Config.in 文件

```
if ["CONFIG_MTD_NAND" = "y" ]; then
Tristate "Yaffs filesystem on NAND" CONFIG_YAFFS_FS
Fi
```

6.修改[linux]/fs/Makefile 文件

```
subdir-$(CONFIG_YAFFS_FS)        +=yaffs
```



7.在控制台键入 make menuconfig 并且选择 Yaffs。

8.在控制台键入 make dep 编译内核，并将新内核当入 NAND 型闪存中。

9.使用新内核启动，运行 MTD 包中的 util 目录下的 MAKEDEV 程序建立/dev/mtd 目录。

10.装载 NAND 的 MTD 驱动程序:

```
#sbin/insmod    mtdemul/nandemul
```

11.装载 YAFFS 的驱动程序

```
#sbin/insmod    mtdemul/yaffs.
```

结束语

LINUX 从诞生到现今只经历了短短十几年，但其影响力是惊人的！现在，从桌面系统到手持设备，处处都有他的身影。这都归功于他的源码开放性。正是由于世界各地的技术精英对 LINUX 的维护和改进，才使得他能够有如此高的稳定性。更重要的是，LINUX 的成功大大促进了开源事业的发展！嵌入式 LINUX 是嵌入式操作系统的一个新成员，其最大的特点就是源代码公开并且遵循 GPL 协议，在近几年以来成为研究热点。

本文主要讲述了嵌入式系统特点，嵌入式系统软件的开发，嵌入式 LINUX 的结构。并在此理论基础上，介绍了 LINUX2.6 内核以及网卡驱动程序和 YAFFS 文件系统在 S3C2410 开发板上的移植过程。

硬件开发平台的 CPU 是三星公司的 S3C2410，他是 ARM920T 核的一款内核芯片，主要应用于高端手持设备，具有高性能、低功耗、体积小等特点。随着半导体技术的飞速进步，SOC 将成为嵌入式应用的发展趋势，操作系统的使用也越来越普及。LINUX 内核小巧灵活，易于裁减，使其很适合嵌入式系统应用。

嵌入式操作系统移植过程可分为两个过程，第一步是先移植内核到 ARM 平台上，使内核可以在他上面运行起来。这步一般多由 CPU 生产厂商来完成；第二步是内核运转正常的情况下移植外设驱动程序和应用程序，这步一般是由嵌入式系统生产商提供。本文着眼于应用最广泛的网卡驱动进行研究，并取得一定成果。

现代操作系统推动了嵌入式开发的新理念，虽然说有许多优秀的商业软件具有开发周期短、性能优越等优点。但是这些软件大都采用黑盒子方式，即源代码不公开。本文通过移植源码开放的 LINUX 操作系统到 S3C2410 平台上，使平台具有系统软件的支持，为以后的应用程序开发提供了操作系统支持。

致谢

终于到了写致谢的时候，为论文的最终定稿舒了一口气。忽然意识到两年半研究生生涯将就此结束，心中难免不舍。回顾这一程求学路，记忆里满是老师的悉心指导和同学的快乐相伴，他们让我的生活充实而富有活力，让我在生命的又一里程碑上刻下了重要的篇章，在此我要向他们表达最诚挚的感谢。

首先要感谢我的导师路勇老师，她在学习和科研方面给了我大量的指导，并为我提供了良好的科研环境，让我学到了知识，掌握了科研的方法，也获得了实践锻炼的机会。她严谨的治学态度、对我的严格要求以及为人处世的坦荡将使我终身受益。在本论文的撰写过程中，路老师从选题直至成稿一直给予我重要的指导和帮助，为我解开了无数的困惑，提供了很多关键性的建议。另外还要感谢李哲英，陈后金，李晓光以及电信学院的所有的老师，他们让我在课堂上学到了丰富的知识，让我见识了众多研究领域的精华。

研究生期间朝夕相处的同学也是宝贵的财富，感谢王东、李晓光、刘文才，李兵等好友让我得到了日积月累的真挚友情，不论是有形的图文还是无形的记忆，都会珍藏着我们的欢声笑语，永不磨灭。感谢电路与系统班所有同学，感谢所有的朋友，很幸运能够认识你们，我的学习生活因你们而更加丰富多彩。

感谢生我养我的父母以及温柔可人的妻子，他们给了我无私的爱，我深知他们为我求学所付出的巨大牺牲和努力，而我至今仍无以为报。祝福他们，以及那些给予我关爱的长辈，祝他们幸福、安康！

还有很多我无法一一列举姓名的师长和友人给了我指导和帮助，在此衷心的表示感谢，他们的名字我一直铭记在心！

最后，衷心感谢在百忙之中抽出时间审阅本论文的专家教授。

参考文献

[1] 作者：Daniel P.Bovet & Marco Cesati 译者：陈莉君 冯锐 牛欣源《深入理解 LINUX 内核（第二版）》 2004.6 O`REILLY&中国电力出版社

[2] 作者：Alessandro Rubini & Jonathan Corbet 译者：魏永明 骆刚 姜君 《LINUX 设备驱动程序（第二版）》 2002.11 O`REILLY&中国电力出版社

[3] 作者：Alessandro Rubini ，Jonathan Corbet &Greg Kroab-Hartman 译者：魏永明 耿岳 钟书毅《LINUX 设备驱动程序（第三版）》2006.1 O`REILLY&中国电力出版社

[4] 倪继利 《LINUX 内核分析及编程》 2005.9 电子工业出版社

[5] 李善平 陈文智等编著 《边干边学－LINUX 内核指导》2002.8 浙江大学出版社

[6] 杜春雷编著 《ARM 体系结构与编程》2003.2 清华大学出版社

[7] 作者：Craig Hollabaugh 译者：陈雷 钟书毅等 《嵌入式 LINUX－硬件，软件和接口》 2004.4 电子工业出版社

[8] 吴明晖主编 《基于 ARM 嵌入式系统开发与应用》 2004.6 人民邮电出版社

[9] 周立功等编著 《ARM 微控制器基础和实战》 2003.11 北京航天航空大学出版社

[10] 孙天泽 袁文菊 张海峰编著 《嵌入式设计及 LINUX 驱动程序开发指南》 2005.2 电子工业出版社

[11] 刘乐善主编 《微型计算机接口技术及应用》2000.4 华中科技大学出版社

[12] 任永铮编著 《LINUX C 程序员指南》 2000.9 国防工业出版社

[13] 作者：Jobn Catsoulis 译者：徐君明 许铁军 黄年松等 《嵌入式硬件设计》 2004.6 O`REILLY&中国电力出版社

[14] 作者：Cameron Newbam & Bill Rosenblatt 译者：徐炎 查石祥等 《学习 BASH（第二版）》 2003.1 O`REILLY&中国电力出版社

[15] 作者：W.Richard Stevens 译者：范建华 张涛等《TCP/IP 详解 卷 1：协议》 2004.12 机械工业出版社 & Addison-Wesley.

[16] W.Richard Stevens， Bill Fenner， Andrew M.Rudoff 著 《UNIX Network Programming Volume 1 The socket Networking API》机械工业出版社

[17] Rafeeq Ur Rehman， Christopher Paul Linux Development Platform 2002.11 Prentice Hall PTR

[18] SamSung s3c2410x User`manual Revision 1.2 2003

[19] Cirrus Logic CS8900A Product Data Sheet 2004.9

[20] MIZI Research Inc MIZI linux 1.5 SDK for S3C2410 Rev 2.1 2004-07

[21] Gerard Beekmans Linux From Scratch Rev 5.0 2003

[22] Alan Cox Begging Linux Programming 2nd Edition 2001 Wrox Press Ltd

[23] Michel Barr Programming Embeded Systems in C and C++ 1999 O`Reilly

[24] Matthias Kalle Dalheimer Running Linux 4th Edition 2002 O`Reilly

[25] GNU The GNU C Library

[26] www.keranl.org

[27] www.arm.com.uk

[28] www.sourceforge.net