火龙果•整理
uml.org.cn

哈爾濱工業大學

毕业设计（论文）

班级： 0402301

姓名： 张立强

2008 年 6 月

火龙果•整理
uml.org.cn

哈爾濱工業大學

毕业设计（论文）

题 目：基于 ARM9 的嵌入式导航计算机 Linux 系统移植

院 （系） 信息科学与工程学院
专 业 电气工程及其自动化
学 生 张立强
学 号 040230125
指导教师 曲延滨

哈尔滨工业大学教务处制

2008 年 6 月

火龙果•整理 uml.org.cn

毕业设计（论文）任务书

<table>
<tr><td>专　　业</td><td colspan="2">电气工程及其自动化</td><td>班　　级</td><td colspan="2">0402301</td></tr>
<tr><td>学　生</td><td>张立强</td><td>指导教师</td><td>曲延滨</td><td>指导教师
职　称</td><td>教授</td></tr>
<tr><td>设计题目</td><td colspan="5">基于 ARM9 的嵌入式导航计算机 Linux 系统移植</td></tr>
<tr><td>题目类型</td><td colspan="5">实验研究型</td></tr>
<tr><td>设计时间</td><td colspan="5">2008 年 4 月 9 日　至　2008 年 6 月 15 日　共 12 周</td></tr>
<tr><td rowspan="3">设
计
要
求</td><td colspan="5">设计的专业方向、基本理论及设计内容：
专业方向：电气工程及其自动化
基本理论：基于 ARM 处理器的嵌入式操作系统移植
设计内容：本设计主要研究嵌入式开发的操作系统移植。需要使用 PC 机建立编译环境，按步进行操作系统各个部分的修改定制、编译和移植。设计中需要在开发板上进行大量的实验，通过实验中发现问题，逐步进行修改，工作完成后所要达到的目标是使 Linux 操作系统在 ARM 开发板上正常运行，并实现一定的功能。</td></tr>
<tr><td colspan="5">学生本人在该设计中具体完成的工作：
1， 在 PC 机上建立交叉编译工具链
2， 移植 Bootloader
3， 精简 Linux 内核，并将其移植到开发板
4， 建立根文件系统</td></tr>
<tr><td colspan="5">主要参考文献、资料：
[1] 李亚锋等《ARM 嵌入式 Linux 系统开发》 清华大学出版社 2007.8
[2] 孙纪坤等《嵌入式 Linux 系统开发技术详解—基于 ARM》人民邮电出版社 2006.8
[3] 孙天泽等《嵌入式设计及 Linux 驱动开发指南—基于 ARM9》电子工业出版社 2005
[4] 郭秋平《基于 ARM 系统的 Linux 平台移植研究》浙江大学硕士论文 2006.1</td></tr>
<tr><td>系
审
批
意
见</td><td colspan="5">负责人签字

年　　月　　日</td></tr>
<tr><td>院
审
批
意
见</td><td colspan="5">负责人签字

年　　月　　日</td></tr>
</table>

火龙果•整理
uml.org.cn

摘要

嵌入式导航计算机是飞机，车辆，导弹和船舶等运载体上的重要设备。其主要任务是按照原定的计划和任务，以要求的精度，在一定时间内将载体引导至目的地。本课题的目的就是针对其硬件环境，搭建起一个高效、稳定的嵌入式操作系统的平台。Linux 操作系统具有开放源代码、功能强大且易于移植等特点而成为嵌入式操作系统的首选。

本文首先总结了 ARM 的体系结构和特点，并选择了基于 ARM920T 内核的 LJD2410 开发板作为移植平台。然后介绍了搭建交叉开发环境的一般方法。之后介绍了嵌入式 Linux 系统的启动程序的实现原理，并实现了 U-Boot 的移植。在对 Linux 内核结构进行了分析后，结合嵌入式导航计算机的系统需求，给出了 Linux 内核的移植和裁剪方案。最后详细介绍了如何构建根文件系统以及系统部署的方法。

关键词： 嵌入式系统；ARM；Linux；移植；系统部署

火龙果•整理 uml.org.cn

Abstract

Embedded navigation computer is an important equipment in aircrafts, vehicles, missiles, ships and other transporters. Its main task is to lead carriers to the destination as planed with the required precision and in a certain period of time. The purpose of this subject is building an efficient and stable embedded operating system platform according to its hardware environment. Linux operating system has became the first choice in building an embedded operating system for its openness in source code, its powerful function and easiness in transplanting.

Firstly, the paper summarizes the features of the ARM architecture. The LJD2410 board that based on ARM920T is choosen as the target platform. Afterward the way to building a cross-development environment is expounded. Then comes the introduction of the methods in building a cross-development environment and the bootloader of embedded Linux system is expounded, along with the transplant of U-Boot. After the analysis of the kernel in Linux, according to the system requirements of the embedded navigation computer, the scheme for its transplant and reducing is given. Finally the paper described in detail how to build a root file system and the system deployment methods.

Keywords: Embedded system; ARM; Linux; Transplant; System deployment

火龙果•整理
uml.org.cn

目录

摘要 I
Abstract II

第 1 章 绪论 1
1.1 嵌入式系统概述 1
1.1.1 嵌入式系统的定义 1
1.1.2 嵌入式系统的组成 1
1.1.3 嵌入式系统的开发 1
1.2 嵌入式操作系统概述 2
1.2.1 嵌入式操作系统 2
1.2.2 嵌入式 Linux 操作系统 2
1.3 ARM 体系结构与硬件平台 3
1.3.1 ARM 体系结构的历史与技术特征 3
1.3.2 Samsung S3c2410 处理器简介 4
1.3.3 LJD2410 开发板简介 5
1.4 本课题的背景和意义 6
1.5 本课题的主要工作和研究内容 6
第 2 章 交叉编译环境的建立 8
2.1 上位机的软硬件配置 8
2.1.1 上位机硬件配置 8
2.1.2 上位机操作系统及软件 8
2.2 硬件连接与调试 9
2.2.1 硬件连接方式 9
2.2.2 串口调试 10
2.3 配置 TFTP 及 NFS 服务 11
2.3.1 TFTP 服务简介 11
2.3.2 TFTP 服务安装与配置 12
2.3.3 NFS 服务简介 12
2.3.4 NFS 服务安装与配置 13
2.4 安装交叉编译工具 13
2.4.1 交叉编译简介 13

火龙果•整理
uml.org.cn

2.4.2 交叉编译器的安装及配置......14
2.4.3 测试交叉编译器......14
2.5 本章小结......15
第 3 章 移植 Bootloader......16
3.1 Bootloader 概述......16
3.2 U-boot 简介......16
3.2.1 U-boot 的获取......16
3.2.2 U-boot 目录结构......16
3.3 U-boot 的启动过程及工作原理......17
3.3.1 启动模式介绍......17
3.3.2 启动阶段 1 分析......19
3.3.3 启动阶段 2 分析......20
3.4 U-boot 的移植过程......20
3.4.1 准备工作......20
3.4.2 添加支持 NAND Flash 启动功能......21
3.4.3 添加 NAND Flash 读写功能......22
3.4.4 修改 U-boot 环境变量保存方式......22
3.4.5 加入 NAND Flash 闪存型号支持......22
3.4.6 编译 U-boot......23
3.5 U-boot 的烧写及测试......23
3.6 设置 U-boot 环境变量......25
3.7 本章小结......26
第 4 章 Linux 内核的移植......27
4.1 Linux 内核的结构......27
4.2 Linux 启动过程简析......28
4.3 Linux 内核的移植过程......28
4.3.1 选择参考板......28
4.3.2 修改 NAND Flash 分区信息......28
4.3.3 关闭 ECC 校验......30
4.4 CS8900a 网卡的移植过程......30
4.4.1 修改硬件地址映射......30
4.4.2 添加 CS8900A 内核编译项......31
4.5 Linux 内核的剪裁配置......31

火龙果•整理
uml.org.cn

4.5.1 使用配置菜单......31
4.5.2 基本配置选项......32
4.5.3 驱动程序配置选项......33
4.5.4 保存配置文件......34
4.5.5 编译 Linux 内核......34
4.6 内核的下载及启动......34
4.6.1 将引导信息加入内核映像......34
4.6.2 内核映像的下载及运行......35
4.7 本章小结......36
第 5 章 建立根文件系统......36
5.1 根文件系统概述......36
5.1.1 根文件系统简介......36
5.1.2 NFS 文件系统与 Cramfs 文件系统......36
5.2 建立 Linux 根文件系统目录......36
5.3 移植 Busybox 工具......37
5.3.1 Busybox 工具简介......37
5.3.2 Busybox 的配置......37
5.3.3 编译并安装 Busybox......38
5.4 移植 Tinylogin 工具......38
5.4.1 编译 Tinylogin 工具......38
5.4.2 建立登陆密码文件......38
5.5 建立初始化文件......39
5.5.1 Inittab 文件......39
5.5.2 Fstab 文件......39
5.5.3 Profile 文件......40
5.6 建立启动脚本文件......40
5.7 应用程序的建立......40
5.8 将 Linux 内核及根文件系统部署到开发板......42
5.8.1 重新修改、编译内核及启动脚本文件......42
5.8.2 内核的烧写固化过程......42
5.8.3 制作 Cramfs 根文件系统映像......42
5.8.4 根文件系统的固化过程......43
5.9 本章小结......43

火龙果•整理
uml.org.cn

结论......44
致谢......45
参考文献......46
附录......48

火龙果•整理
uml.org.cn

第 1 章 绪论

1.1 嵌入式系统概述

1.1.1 嵌入式系统的定义

所谓嵌入式系统（Embedded System），实际上是“嵌入式计算机系统”的简称，它是相对于通用计算机系统而言的。国际电气与电子工程师协会（IEEE）对于嵌入式系统的定义：嵌入式系统是用来控制或监视机器、装置或工厂等大规模系统的设备。可以看出此定义是从应用方面考虑的。国内对嵌入式系统的一般定义：嵌入式系统是以应用为中心、以计算机技术为基础、软件硬件可剪裁、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统[1]。

嵌入式系统是软件和硬件的综合体，其涵盖范围和领域都十分广泛，几乎包括了我们周围所有的电器设备，比如：电视机机顶盒、掌上 PDA，甚至路由器、交换机等。

1.1.2 嵌入式系统的组成

嵌入式系统一般可以分为四个部分：嵌入式处理器、嵌入式外围设备、嵌入式操作系统和嵌入式应用软件[2]，如图 1-1 所示。

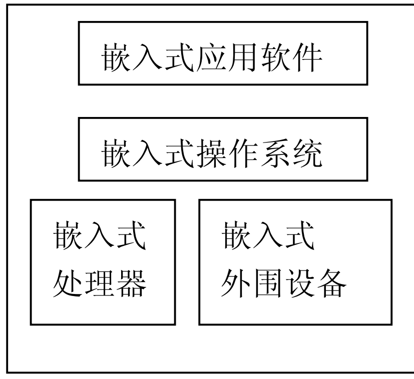

图 1-1 嵌入式系统的组成

1.1.3 嵌入式系统的开发

嵌入式开发的流程可以简述如下：

火龙果•整理
uml.org.cn

（1） 系统需求分析
（2） 硬件平台的选择和设计
（3） 移植嵌入式操作系统
（4） 驱动程序的开发
（5） 应用软件的开发

本课题主要研究的是移植嵌入式操作系统，其中主要包括以下几个方面：

（1） 建立开发环境
（2） 引导程序的移植
（3） 内核剪裁移植
（4） 建立根文件系统

1.2 嵌入式操作系统概述

1.2.1 嵌入式操作系统

嵌入式操作系统是一种支持嵌入式系统应用的操作系统软件，它是嵌入式系统(包括硬、软件系统)极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等 Browser[3]。

一般情况下，嵌入式操作系统可以分为两类，一类是面向控制、通信等领域的实时操作系统，如 WindRiver 公司的 VxWorks、ISI 的 pSOS、QNX 系统软件公司的 QNX、ATI 的 Nucleus 等；另一类是面向消费电子产品的非实时操作系统，这类产品包括个人数字助理(PDA)、移动电话、机顶盒等。

1.2.2 嵌入式 Linux 操作系统

Linux 的嵌入式改造主要围绕体积和实时性展开，目前已经有很多公司在进行这方面的工作，其中包括 RT-Linux，uClinux，Embedix，Xlinux，MidoriLinux 和红旗嵌入式 Linux 等等。

与目前市场上的众多商业的实时系统相比，嵌入式 Linux 除具有内核稳定，功能强大，支持多种硬件平台，兼容性好的优势外，还拥有以下的特点：

(1)完全开放源代码

嵌入式 Linux 完全开放其源代码，这使得修改，裁剪 Linux 成为可能，开发者可以根据实际需要优化操作系统代码，降低整个系统的开销与能耗。

(2)成本低

火龙果•整理 uml.org.cn

GPL 协议保证了源自 Linux 的嵌入式 Linux 也是开放源代码的自由软件。而大多数嵌入式 Linux 使用的开发工具也是遵守 GPL 协议的，同样也可以免费获得。

(3)丰富的实用软件支持

Linux 提供了大量的实用程序和各种应用软件。这些软件的正确性和有效性都经过了实际检验，可以根据需要合理利用他们迅速构建嵌入式应用的软件环境。这样可以极大地减小嵌入式软件开发的时间和费用，提高系统可靠性。而商用的实时操作系统也试图提供各种常用软件工具包，但其数量是无法和 Linux 操作系统匹敌的。

由此可见，选择嵌入式 Linux 操作系统，就有了丰富的资源保障，在节省成本的同时，提高了开发效率。

1.3 ARM 体系结构与硬件平台

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。ARM 处理器是一种低功耗刚性能的 32 位 RISC 处理器[4]。作为 SOC（System On Chip）的典型应用，目前，基于 ARM 的处理器以其高速度、低功耗等诸多优异的性能而得到非常广泛的应用。

1.3.1 ARM 体系结构的历史与技术特征

1983 年 10 月至 1985 年 4 月间，第一片 ARM 处理器在位于英国剑桥的 Acorn Computer 公司开发。在 20 世纪 80 年代后期，ARM 处理器已经发展为可支持 Acorn 公司的台式计算机产品，这些产品奠定了英国教育界计算机技术的基础。1990 年 11 月，为广泛推广 ARM 技术，由苹果电脑、Acorn 集团等合资组建新公司：Advanced RISC Machine Limited（简写为 ARM Limited），此后 ARM 代表着“Advanced RISC Machine”。

采用 RISC 构架的 ARM 处理器一般具有如下特点[5]：

（1）体积小、低功耗、低成本、高性能

（2）支持 Thumb（16 位）/ARM（32 位）双指令集，能很好地兼容 8 位/16 位器件。

（3）大量使用寄存器，指令执行速度更快。

（4）ARM9 系列处理器采用 5 级流水线，还带有 MMU（Memory Management Unit）功能。

火龙果•整理
uml.org.cn

（5）ARM9 系列处理器支持指令 Cache 和数据 Cache，具有更高的数据处理能力。

1.3.2 Samsung S3c2410 处理器简介

S3c2410 是著名半导体公司 Samsung 推出的一款 32 位 RISC 处理器。S3c2410 的内核基于 ARM920T，带有 MMU 功能，主频高达 203MHz，可以支持 Linux、WinCE 等主流嵌入式操作系统。同时它还采用了一种叫做 Advanced Microcontroller Bus Architecture（AMBA）的新型总线结构。

此外 S3c2410 还集成了以下片上功能：

（1）16KB 指令 Cache 和 16KB 的数据 Cache；

（2）LCD 控制器（支持 STN 和 TFT）；

（3）4 通道 DMA；

（4）3 通道 UART；

（5）2 通道 USB；

（6）4 路 PWM 和 1 个内部时钟控制器；

（7）117 个通用 IO，24 路外部中断；

（8）16 位看门狗定时器；

（9）RTC（实时时钟）；

（10）1 通道 IIC/IIS 控制器；

（11）NAND Flash 控制器；

（12）PLL 数字锁相环。

S3c2410 将系统的存储空间分为 8 组（bank），每组大小是 128MB，共 1GB。Bank0 到 Bank6 都采用固定 Bank 起始寻址，用于 ROM 或 SRAM。Bank7 具有可编程的 Bank 的起始地址和大小，用于 ROM、RAM 或 SDRAM。S3c2410 还支持从 NAND Flash 启动，NAND Flash 具有容量大、比 NOR Flash 价格低等特点[6]。系统采用 NAND Flash 与 SDRAM 相结合的方式，可以获得非常高的性价比。系统使用 NAND Flash 时地址空间分配如图 1-2 所示：

火龙果•整理 uml.org.cn

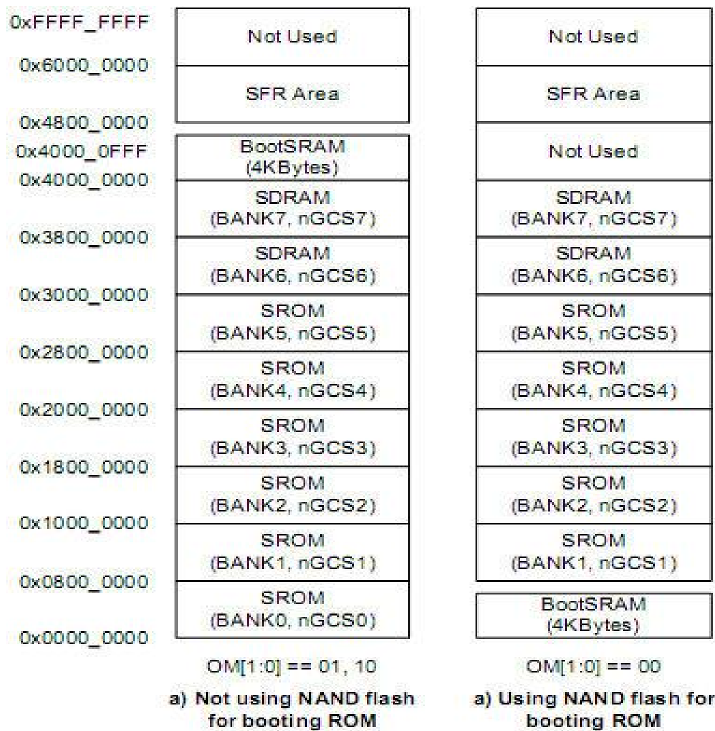

图 1-2 NAND Flash 地址分配表

1.3.3 LJD2410 开发板简介

本课题使用北京蓝海微芯科技有限公司的 ARM9 开发套件 LJD-2410-DVK-I 开发板，该开发板基于 Samsung S3c2410 处理器，采用了核心板加底板的设计方式，提供了完备的软硬件平台。

核心板的基本配置如下：

CPU：Samsung S3c2410A

内存：hynix HY57v561620FTP-h 两块共 64MB

NAND Flash：Samsung K9F1208U0B 64MB

网卡：Crystal CS8900A-CQ3Z

底板上与本课题相关的接口有：

两个 RS-232 串口

一个 RJ-45 网卡接口，带连接和传输指示灯

一个 JTAG 调试接口

火龙果•整理 uml.org.cn

1.4 本课题的背景和意义

嵌入式导航计算机是飞机，车辆，导弹和船舶等运载体上的重要设备，主要任务是按照原定的计划和任务，以要求的精度，在一定时间内将载体引导至目的地。嵌入式导航计算机主要分为两部分：硬件电路，嵌入式操作系统。

本课题的目的就是针对其硬件环境，搭建起一个高效、稳定的嵌入式操作系统的平台。它具有通用操作系统的基本特点，能够有效管理复杂的系统资源；能够快速的处理大量的信息；能够提供库函数、驱动程序、工具集以及部分应用程序。在这个系统平台上可以运行导航程序，接受传感器的数据，经处理后得到任务所需要的信息，从而实施导航任务。

嵌入式 Linux 有着嵌入式导航计算机操作系统需要的很多特色：支持多任务处理、中断处理及任务间通信，性能稳定，剪裁性好，开发与使用都很方便。因此，本设计选用嵌入式 Linux 作为嵌入式导航计算机的操作系统，这对于实现导航计算机的高效率、低功耗具有现实意义。

嵌入式导航计算机硬件平台结构如图 1-3 所示：

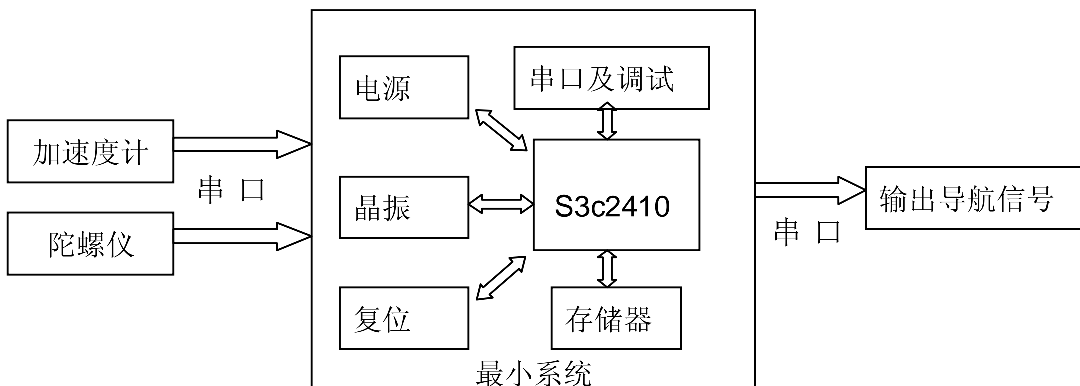

图 1-3 嵌入式导航计算机硬件平台结构图

1.5 本课题的主要工作和研究内容

本课题的最重目标是为嵌入式导航计算机移植 Linux 操作系统。通过参阅大量文献，学习嵌入式 Linux 系统和 ARM 体系微处理芯片 S3C2410 的相关知识，研究启动下载程序 Bootloader 和 Linux 内核的基本工作原理，并且搭建交叉编译平台，重点是移植 Bootloader 和内核，以及制作根文件系统。具体工作内容包括：

火龙果•整理
uml.org.cn

（1） 学习 Linux 操作系统的知识。

（2） 了解 ARM 的体系结构和 S3C2410 芯片硬件结构。

（3） 完成交叉编译环境的建立。

（4） 修改并移植 U-boot 1.2.0 。

（5） 修改和裁剪 Linux 2.6.24.4 内核，移植网卡驱动程序。

（6） 制作根文件系统。

（7） 编写应用程序进行测试。

（8） 将内核和根文件系统部署到开发板。

火龙果•整理
uml.org.cn

第 2 章 交叉编译环境的建立

采用交叉开发环境（Cross Development Environment）是嵌入式应用软件开发时的一个显著特点，通常在通用计算机上编写程序，然后通过交叉编译生成目标平台上可运行的二进制代码格式，最后再下载到目标平台上的特定位置运行，交叉开发环境是指编译、链接和调试嵌入式应用软件的环境，它与运行嵌入式应用软件的环境有所不同，通常采用主机/目标及模式[7]。交叉开发模型如图 2-1 所示：

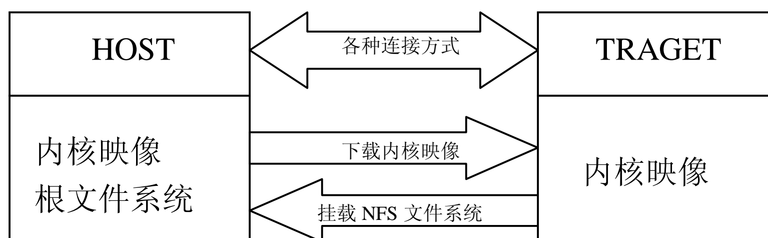

图 2-1 交叉开发模型

2.1 上位机的软硬件配置

2.1.1 上位机硬件配置

本课题用到一台通用 PC 机和一台笔记本电脑，其硬件配置如下：

PC 机：

CPU：P IV 2.0G

RAM：256MB

串口：RS-232

并口：25 针母头

笔记本电脑：

CPU：PM 705

RAM：768MB

网卡：10/100MBps 自适应网卡

2.1.2 上位机操作系统及软件

PC 机的操作系统为 Windows XP，装有 DNW 串口调试工具以及 SJF2410 三星 Flash 烧写工具。前者用于串口调试，后者用于烧写 Bootloader。

火龙果•整理
uml.org.cn

笔记本电脑的操作系统为 Ubuntu7.10，装有 GCC 等编译工具以及 arm-linux-gcc 交叉编译工具，并开启 TFTP 和 NFS 服务。用于 Linux 内核编译和软件开发，并作为 TFTP 服务器和 NFS 主机。

其中，Ubuntu7.10 是 Linux 的桌面发行版之一，是当今最为流行的桌面 Linux 系统[8]。使用 Linux 操作系统及其自带的工具，是目前最权威的嵌入式 Linux 系统开发方式，但是许多操作都是基于命令行的，所以需要扎实的 Linux 基础知识。

在 Ubuntu 中建立 arm 用户，专门用于 ARM 开发。在 home 目录中建立下列几个子目录：

Boot：用于存放 bootloader 相关程序。

Kernel：用于存放 Linux 内核源码。

FS：用于存放根文件系统相关的程序。

Program：用于存放用户程序。

2.2 硬件连接与调试

2.2.1 硬件连接方式

如图 2-2 所示：

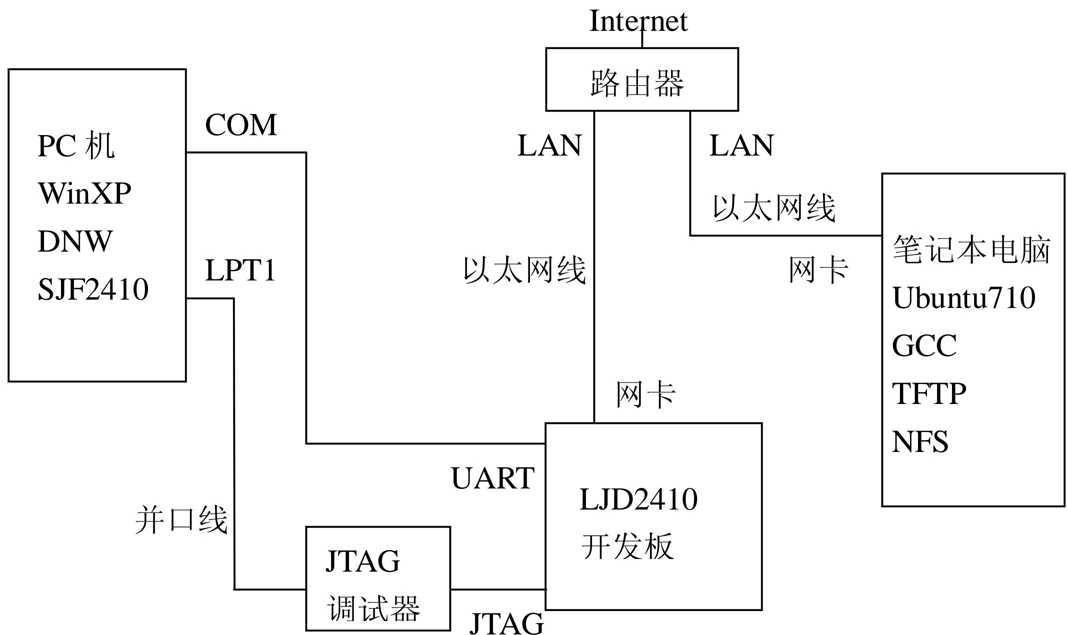

图 2-2 硬件连接图

火龙果•整理 uml.org.cn

（1）开发板串口 UART0 通过交叉串口线与 PC 主机的 COM1 口相连。
（2）开发板的 JTAG 口通过 20PIN 排线与 SUPER JTAG 调试头相连，再通过 25PIN 并口线连接到主机的 LPT1 口。
（3）开发板的网卡接口通过以太网线连接到路由器的 LAN1 口。
（4）笔记本的网卡接口通过以太网线连接到路由器的 LAN2 口。
（5）路由器的 WAN 口连接到 INTERNET。

2.2.2 串口调试

在本课题嵌入式系统中的目标开发板，采用串口调试的方法，即把串口当作目标开发板的显示终端，无论是打印输出，还是管理配置输入，都使用串口，这就需要主机系统装有串口调试工具。

PC 机中安装有 DNW 串口调试工具，在使用 DNW 之前，应当对 PC 机的串口进行设置。方法如下：
（1）XP 系统中，右键单击“我的电脑”，选择“属性”。
（2）选择“硬件”——“设备管理器。
（3）选择“端口”——“串口（COM1）”，打开的对话框按图 2-3 设置。

图 2-3 串口属性设置

火龙果•整理 uml.org.cn

运行 DNW 工具，选择菜单“Configuration”——“Options”，按图 2-4 所示进行设置。

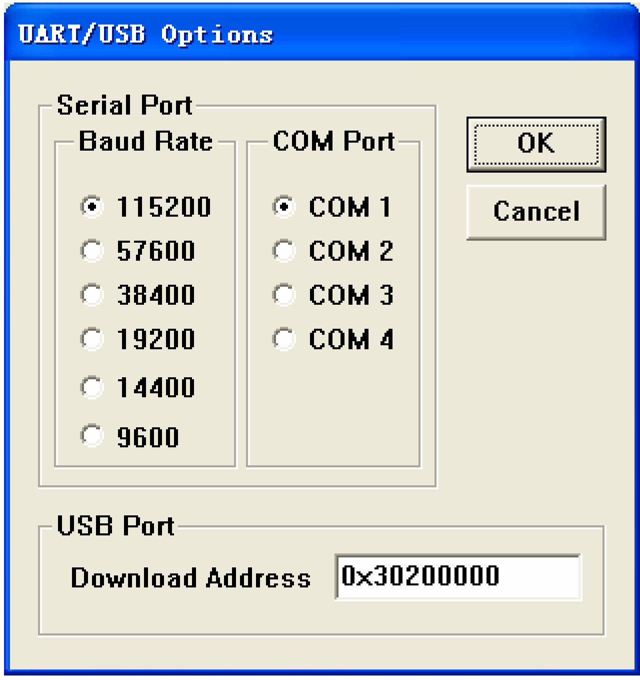

图 2-4　DNW 设置

每次使用 DNW 之前，应当设置 DNW 连接到串口。点击菜单中的“Serial Port”——“Connect”，当 DNW 的标题栏出现[COM1,115200bps]的提示后，表明已经连接好，此时才可以使用 DNW 工具。

2.3　配置 TFTP 及 NFS 服务

2.3.1　TFTP 服务简介

TFTP（Trivial File Transfer Protocol）协议即简单文件传输协议，是 TCP/IP 协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。TFTP 承载在 UDP 上，提供不可靠的数据流传输服务，不提供存取授权与认证机制，使用超时重传方式来保证数据的到达。

TFTP 服务在 Linux 系统中有客户端和服务器两个软件包。配置 TFTP 服务，必须都安装好。

火龙果•整理
uml.org.cn

2.3.2 TFTP 服务安装与配置

（1）Ubuntu 中安装 tftp 工具只需在终端中键入命令：

$ sudo apt-get install tftp tftpd

其中，前者是客户端，后者是服务器。

（2）Ubuntu 是 debian 类的系统，默认是没有安装 inetd 的，安装命令如下：

$ sudo apt-get install netkit-inetd

（3）在 home 目录里建立 tftpboot 文件夹，命令如下[9]：

$ cd ~

$ sudo mkdir tftpboot

$ sudo chmod 777 tftpboot

其中，参数 777 的意义是：根管理员、组和其他用户对 tftpboot 文件夹的权限均为“可读、可写、可以执行”

（4）修改/etc/inetd.conf，添加如下语句：

tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /home/arm/tftpboot

目的是指定 tftp 服务的根目录为/home/arm/tftpboot，修改/etc/inetd.conf 文件后应当重启 inted 进程，命令如下：

$ sudo /etc/init.d/inetd reload

（5）重启 inted 进程后，配置即可生效，在 tftpboot 中建立文件 test 后，用下列命令可以进行测试：

$ cd ~

$ tftp 127.0.0.1

Tftp> get test

若可下载 test 文件，则证明 TFTP 服务配置正确。

2.3.3 NFS 服务简介

NFS 就是 **N**etwork **F**ile **S**ystem 的缩写，最早之前是由 Sun 这家公司所开发的。最大的功能就是可以透过网络，让不同的机器、不同的操作系统、可以彼此分享个别的档案 (share files)[10]。所以，可以简单的将它看做是一个文件服务器 (file server) 。通过 NFS 服务器可以让开发板将网络远端的 NFS 主机分享的目录，挂载到开发板当中，在开发板看起来，那个远端主机的目录就好像是自己的根目录一样，可以方便的进行开发调试。

火龙果•整理 uml.org.cn

2.3.4 NFS 服务安装与配置

（1）Ubuntu 上默认是没有安装 NFS 服务器的，首先要安装 NFS 服务程序：

$ sudo apt-get install nfs-kernel-server

在安装 nfs-kernel-server 时，apt 会自动安装 nfs-common 和 portmap。这样，宿主机就相当于 NFS Server。

（2）配置/etc/exports：

NFS 挂载目录及权限由/etc/exports 文件定义。本课题要将 home 目录中的/home/zp/share 目录让 192.168.0.*的 IP 共享, 则在该文件末尾添加下列语句：

/home/arm/FS/myrootfs 192.168.0.2/10 (rw,sync,no_root_squash)

配置参数说明：

rw：具有可擦写的权限。

sync：文件同步写入到内存和硬盘当中。

no_root_squash：若登陆共享目录的使用者是 root 的话，则他的权限将被限制为匿名使用者，通常他的 UID 和 GID 都会变为 nobody。

（3）本地测试 NFS：

输入以下命令可以将 NFS 根目录挂载到本地的/mnt 目录中：

$ sudo mount 192.168.0.2:/home/arm/FS/myroot /mnt

此时/mnt 中的内容应当与 NFS 根目录中的内容一致。

2.4 安装交叉编译工具

2.4.1 交叉编译简介

所谓交叉编译，简单的说，就是在一个平台上生成另一个平台上的可执行代码，比如在 PC 平台上（X86 CPU）编译出能运行在以 ARM 为内核的 CPU 平台上的程序，一般选择 GNU 开发工具 gcc。GNU 的开发工具都是免费的，遵循 GPL 协议，任何人可以从网上获取。GNU 提供的编译工具包括汇编器 as、c 编译器 gcc、c++编译器 g++、连接器 ld 和二进制转换工具 objcopy。出于兼容性和稳定性考虑，本课题选择目前比较稳定的版本 Cross-3.3.2 和 Cross-3.4.1。

火龙果•整理
uml.org.cn

2.4.2 交叉编译器的安装及配置

（1）获取 arm-linux 交叉编译工具：

登陆 arm-linux 项目组的 FTP 服务器：

ftp.arm.linux.org.uk/pub/armlinux/toolchain/

下载 cross-3.3.2.tar.bz2 和 cross-3.4.1.tar.bz2。

（2）通过下列命令可以安装 arm-linux 交叉编译工具：

```
$ cp cross-3.4.1.tar.bz2 /
$ cd /
$ tar jxvf cross-3.4.1.tar.bz2
```

这样，交叉编译工具就被安装到了/usr/local/arm/3.4.1 中。用同样的方法可以安装 cross-3.3.2 版的交叉编译工具。

（3）设置环境变量：

修改 home 目录下的 profile 文件，加入如下代码，指明交叉编译工具的目录。

```
# User specific environment and startup programs
export TARGET=arm-linux
export PRJROOT=/home/arm
export
PATH=$PATH:$HOME/bin:$PREFIX/bin:/usr/local/arm/3.4.1/bin:/sbin:/usr/sbin:/usr/local/sbin
```

2.4.3 测试交叉编译器

可以通过一个简单的程序测试安装好的交叉编译工具，看其能否正常工作。编写一个 hello.c 源文件，通过以下命令进行编译，编译后生成名为 Hello 的可执行文件，通过 file 命令可以查看文件的类型。当显示以下信息是表明交叉编译工具正常安装了，通过编译生成了 ARM 体系可执行的文件。注意，通过该交叉编译器编译的可执行文件只能在 ARM 体系下执行，不能在基于 X86 的普通 PC 上执行。

```
$ arm-linux-gcc –o Hello hello.c
$ file Hello
Hello:ELF 32-bit LSB executable ,ARM, version 1 (ARM), for GNU/Linux 2.4.3, dynamically linked (uses shared libs), not stripped
```

火龙果•整理
uml.org.cn

2.5 本章小结

本章主要论述了嵌入式开发的基础——交叉编译环境的建立方法，其中包括两个方面：硬件环境和软件环境。本课题采用 NFS 交叉开发方法，所以需要网络连接；arm 开发板的输出由串口来管理；所需的两台 PC 机分别安装 Windows 和 Linux 两种操作系统，前者用于串口调试、后者用于软件的开发编译。然后介绍了 TFTP 和 NFS 服务的安装配置，这是交叉开发的两个必须服务。最后介绍了交叉编译器的安装方法，正确的安装交叉编译器是后续工作的基础。

火龙果•整理
uml.org.cn

第 3 章 移植 Bootloader

3.1 Bootloader 概述

简单地说，Bootloader 就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，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的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。

通常，Bootloader 是严重地依赖于硬件而实现的，特别是在嵌入式世界。因此，在嵌入式世界里建立一个通用的 Bootloader 几乎是不可能的，不同处理器构架都有不同的 Bootloader。Bootloader 不但依赖于 CPU 的体系结构，而且依赖于嵌入式板级设备的配置。对于不同的嵌入式板而言，即使它们使用同一种处理器，要想让运行在一块板子上的 Bootloader 运行在另一块板子上，一般也要修改其源代码。

目前常用的 Bootloader 程序有以下几种：U-boot、VIVI、Blob 和 RedBoot。其中，U-boot 功能丰富，且对于 ARM 体系支持良好，事实上，它已成为 ARM 平台上标准 Bootloader[11]。因此，本课题选用 U-boot 作为移植对象。

3.2 U-boot 简介

U-boot 是德国 DENX 小组的开发用于多种嵌入式 CPU 的 Bootloader 程序，U-boot 不仅仅支持嵌入式 Linux 系统的引导，还支 NetBSD，VxWorks，QNX，ARTOS，LynxOS 等嵌入式操作系统。U-boot 除了支持 ARM 系列的处理器外，还能支持 MIPS、 x86、PowerPC、NIOS、XScale 等诸多常用系列的处理器。

3.2.1 U-boot 的获取

U-boot 的源码可以从 sourceforge 网站下载，网址为：

http://sourceforge.net/project/u-boot。

下载的文件为 u-boot-1.2.0.tar.bz2，用以下命令将其解压。

$ tar jcvf u-boot-1.2.0.tar.bz2 /home/arm/boot/

3.2.2 U-boot 目录结构

解压后，在 U-boot 顶层目录下有 18 个子目录，分别存放和管理不同的

火龙果•整理 uml.org.cn

源码。这些目录中所要存放的文件有其规则，可以分为 3 类，如表 3-1 所示：

第一类目录与处理器体系结构或开发板硬件直接相关。

第二类目录是一些通用的函数或者驱动程序。

第三类目录是 U-boot 的应用程序、工具或者文档。

表 3-1 U-boot 顶层目录下部分目录的存放规则【12】

目录	特性	解释说明
Board	平台依赖	存放电路板相关的目录文件，如 smdk2410。
Cpu	平台依赖	存放 cpu 相关的目录文件，如 arm924t。
Lib_arm	平台依赖	存放 ARM 体系结构通用的文件。
Include	通用	头文件和开发板配置文件，所有开发板配置文件都在 config 目录下。
Common	通用	通用的多功能函数实现。
Driver	通用	通用的设备驱动程序，主要有以太网的驱动。
Examples	应用例程	一些独立运行的应用程序例子。
Tools	工具	存放制作 S-Record 或 U-boot 格式映像的工具，如 mkimage。
Doc	文档	开发使用文档。

3.3 U-boot 的启动过程及工作原理

3.3.1 启动模式介绍

Bootloader 都包含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【13】。

启动加载（Bootloading）模式：这种模式也称为"自主"（Autonomous）模式。也即 Bootloader 从目标机上的某个固态存储设备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，因此在嵌入式产品发布的时侯，Bootloader 显然必须工作在这种模式下。

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

火龙果•整理
uml.org.cn

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 Bootloader 保存到目标机的 RAM 中，然后再被 Bootloader 写到目标机上的 FLASH 类固态存储设备中 。Bootloader 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 Bootloader 的这种工作模式。工作于这种模式下的 Bootloader 通常都会向它的终端用户提供一个简单的命令行接口。

U-boot 这样功能强大的 Bootloader 同时支持这两种工作模式，而且允许用户在这两种工作模式之间进行切换。

大多数 Bootloader 都分为阶段 1(stage1)和阶段 2(stage2)两大部分，U-boot 也不例外。依赖于 CPU 体系结构的代码（如 CPU 初始化代码等）通常都放在阶段 1 中且通常用汇编语言实现，而阶段 2 则通常用 C 语言来实现，这样可以实现复杂的功能，而且有更好的可读性和移植性。

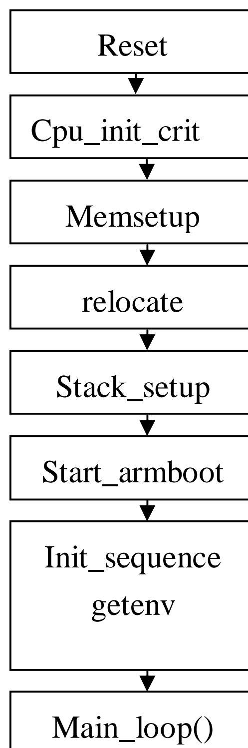

图 3-1 U-boot 启动代码流程图

火龙果•整理
uml.org.cn

3.3.2 启动阶段 1 分析

如果 S3C2410 被配置成从 NAND 闪存启动，上电后，S3C2410 的 NAND 闪存控制器会自动把 NAND 闪存中的前 4K 数据搬移到内部 RAM 中， 并把 0x00000000 设置为内部 RAM 的起始地址，CPU 从内部 RAM 的 0x00000000 位置开始启动。因此要把最核心的启动程序放在 NAND 闪存的前 4K 中。由于 NAND 闪存控制器从 NAND 闪存中搬移到内部 RAM 的代码是有限的，所以，在启动代码的前 4K 里，必须完成 S3C2410 的核心配置，并把启动代码的剩余部分搬到 RAM 中运行。这前 4K 完成的主要工作就是 U-boot 启动的第一个阶段(stage1)。

U-boot 的 stage1 代码通常放在 start.s 文件中，它用汇编语言写成。此阶段要完成的主要工作如下：

（1）设置异常向量，当发生异常时，执行 cpu/arm920t/interrupts.c 中定义的中断处理函数。

（2）设置 CPU 的模式为 SVC（管理模式，操作系统使用的保护模式）

（3）关闭看门狗。

（4）禁掉所有中断。

（5）设置 cpu 频率，默认频率比为 FCLK:HCLK:PCLK = 1:2:4，默认 FCLK 的值为 120 Mhz，该值为 S3C2410 手册的推荐值。

（6）调用 cpu 初始化函数 cpu_init_crit。其中一个功能是设置 CP15 寄存器，失效指令(I)Cache 和数据(D)Cache 后，禁止 MMU 与 Cache。

（7）重定向，将 NAND Flash 代码复制到 RAM，其中有以下两个个步骤：

（a）通过 copy_myself 子程序，把数据从 Nand Flash 中拷贝到 RAM。

（b）配置栈空间，配置代码段的开始地址、动态内存区长度、全局数据大小以及分配 IRQ 和 FRQ 的栈空间。

（8）BSS（Block Started by Symbol）段清零。

（9）进入 C 代码：

```
ldr   pc, _start_armboot
_start_armboot:   .word start_armboot
```

其中 start_armboot 是 U-boot 运行的第一个 C 程序，在 lib_arm/board.c 文件中定义。随后进入第二阶段。

火龙果•整理 uml.org.cn

3.3.3 启动阶段 2 分析

lib_arm/board.c 中的 start armboot 是 C 语言开始的函数，也是整个启动代码中 C 语言的主函数，同时还是整个 U-boot(armboot）的主函数，该函数主要完成如下操作：

（1）定义初始化函数表。

（2）NAND Flash 初始化，利用 nand_init()函数实现。

（3）环境变量初始化，利用 env_relocate ()函数实现。

（4）外围设备初始化，利用 devices_init()函数实现。

（5）使能中断，利用 enable_interrupts ()函数实现。

（6）初始化网络设备。

（7）进入 U－boot 的命令循环，接受用户输入的命令，执行相应的工作。

3.4 U-boot 的移植过程

移植 U-boot 的主要工作就是添加开发板硬件相关的文件、配置选项，然后进行编译。

3.4.1 准备工作

（1）建立开发板编译项，在顶层 Makefile 中加入如下两行：

```
LJD2410_config :    unconfig
@$(MKCONFIG) $(@:_config=) arm arm920t LJD2410 NULL s3c24x0
```

各项意义如下：

arm：CPU 的架构(ARCH)

arm920t：CPU 的类型(CPU),其对应于 cpu/arm920t 子目录。

LJD2410：开发板的型号(BOARD),对应于 board/crane2410 目录。

NULL：开发者/或经销商(vender)。

s3c24x0：片上系统(SOC)。

（2）在 board 子目录中建立 LJD2410 开发板目录：

```
$ cp rf board/smdk2410 board/LJD2410
$ cd board/LJD2410
$ mv smdk2410.c LJD2410.c
```

（3）在 include/configs/中建立配置头文件：

```
$ cp include/configs/smdk2410.h include/configs/LJD2410.h
```

火龙果•整理
uml.org.cn

（4）测试编译能否成功：

```
$ make distclean
$ make LJD2410_config
$ make CROSS_COMPILE=arm-linux-
```

如果编译成功，证明已经建立好了 LJD2410 的编译项，但是还要进行进一步的修改，因为现在的代码是完全拷贝 smdk2410 开发板的，还不能工作在 LJD2410 板上。接下来要按照 LJD2410 板的硬件配置来进一步移植。

（5）调整 SDRAM 的刷新率，修改 lowlevel_init.S：

```
#define REFCNT 1268
```

在 smdk2410.c 中调整 HCLK 为 100MHz：

```
/*Fout = 200MHz */
#define M_MDIV 0x5C
#define M_PDIV 0x4
#define M_SDIV 0x0
```

3.4.2 添加支持 NAND Flash 启动功能

由于 U-boot 不支持从 NAND Flash 启动，所以将程序复制到 RAM 里面去需要新加代码实现，一般通过 copy_myself 函数实现。这可以参考 VIVI 的 copy_muself 代码将其添加到 Start.S 中，详见附录 A-1。

在 Start.S 中调用了 nand_reak_ll 函数，该函数用于 NAND Flash 读操作，在 U-boot 中没有定义，需要新加代码实现，该函数的实现可以参考 VIVI 源代码。将 VIVI/s3c2410/nand_read.c 复制到 LJD2410 目录内即可。

由于使用了新的 Flash 读函数，在编译时需要重新链接，修改 LJD2410 目录中的 Makefile 文件，将原先的 OBJS := myboard.o flash.o 改为：OBJS := myboard.o nand_read.o。

S3c2410 处理器带有 NAND Flash 控制器，但是 U-boot 没有定义其寄存器地址，修改 include/s3c2410.h 文件，加入如下代码：

```
#define oNFCONF 0x00
#define oNFCMD 0x04
#define oNFADDR 0x08
#define oNFDATA 0x0C
#define oNFSTAT 0x10
#define oNFECC 0x14
```

火龙果•整理
uml.org.cn

3.4.3 添加 NAND Flash 读写功能

U-boot 运行至第二阶段进入 start_armboot()函数。其中 nand_init()函数是对 NAND Flash 的最初初始化函数。其调用与 CFG_NAND_LEGACY 宏有关，如果没定义 CFG_NAND_LEGACY 这个宏，就按照 start_armboot()调用 drivers/nand/nand.c 中的 nand_init 函数(该函数在 1.2.0 已经被实现)默认规定，但还有个 board_nand_init()函数没实现，需自己添加。如果定义 CFG_NAND_LEGACY，就不使用默认的 nand_init，而调用自己写的 nand_init 函数了，本课题选择第二种方式。

在/drivers/nand_legacy/nand_legacy.c 中添加 NAND Flash 初始化函数 nand_init，详见附录 A-2。

可以看到 nand_init()调用 NF_Init()函数，使能 NAND Flash 控制器和 NAND Flash；调用 NF_Reset()函数置位，NF_WaitRB()查询 NAND Flash 的状态，最后再调用 nand_probe((ulong)nand)函数探测 NAND Flash。

在 include/configs/smdk2410.h 文件的后半部原先有 Flash 的参数，删除它，并加入 NAND Flash 的参数，并且开启一些命令宏，详见附录 A-3。

3.4.4 修改 U-boot 环境变量保存方式

由于本课题使用 NAND Flash 作为外存储器，所以 U-boot 的参数存储函数应当进行适当的修改。

在/common/env_common.c 里添加 default_env 函数（详见附录 A-4），此函数的作用是对环境变量保存方式的简单初始化。这个文件中还定义了 U-boot 保存环境变量的底层函数。其中/* Environment not changable */行下面的部分应当用 default_env 函数代替。这样，就可以在 U-boot 命令行中实现对环境变量的设置与保存。

文件/common/env_nand.c 中包含了 Flash 擦写函数，结合 CFG_NAND_LEGACY 这个宏，添加代码实现 NAND Flash 的擦写功能。初始化环境仍用 default_env 函数替换。

3.4.5 加入 NAND Flash 闪存型号支持

在/include/linux/mtd/ nand_ids.h 中对 nand_flash_dev nand_flash_ids 结构体的赋值进行修改，加入下列代码：

{"Samsung K9F1208U0B", NAND_MFR_SAMSUNG, 0x76, 26, 0, 3, 0x4000,

火龙果•整理 uml.org.cn

0},

这样，U-boot 就可以正确识别此款 NAND Flash 芯片。

3.4.6 编译 U-boot

确认以上工作准确无误后，可以用下面的命令重新编译 U-boot：

```
$ make disrclean
$ make LJD2410_config
$ make
```

如果编译正确，将在 U-boot 顶层目录下生成 u-boot、u-boot.bin 和 u-boot.srec 三个映像文件。其中 u-boot 是 ELF 格式二进制的 image 文件，u-boot.bin 是原始的二进制 image 文件，u-boot.srec 是 Motorola S-Record 格式的 image 文件[14]。

本课题只使用 u-boot.bin，此文件没有保存 ELF 信息和调试信息。下一节将详细介绍使用 sjf2410 工具烧写映像文件到 NAND Flash 的过程。

3.5 U-boot 的烧写及测试

若开发板中没有任何程序，则不能启动，需要先将 U-boot 烧写到 Flash 中。常用的烧写方法有如下几种：

（1） 将 Flash 取下，用编程器烧写。

（2） 通过串口线烧写。

（3） 通过 JTAG 调试接口烧写。

本课题采用第三种方法。通过 JTAG 接口烧写的优点是操作简单，但是烧写速度较慢，总体来说是一种非常经济实用的方法。具体操作如下：

（1） 连接好开发板和 PC 主机，主机安装并口设备驱动程序。

（2） 将 u-boot.bin 拷贝至 sjf2410 目录下，用以下命令运行 sjf2410：
sjf2410 /f:u-boot.bin

（3） sjf2410 程序启动后，会有三个选项，依次为：

（a）选择 Flash 芯片型号，

（b）选择程序类型，

（c）选择烧写起始地址。

本课题全部选择“0”即可，烧写过程如图 3-2 所示：

火龙果•整理
uml.org.cn


```
C:\WINDOWS\system32\cmd.exe
+------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.4      |
|     (S3C2410X & SMDK2410 B/D)      |
+------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2410X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S12(56)08 prog     1:28F128J3A prog   2:AM29LV800 Prog   3:SST39VF160 Prog

 4:Memory Rd/Wr      5:Exit
Select the function to test:0

[K9S12(56)08 NAND Flash JTAG Programmer]
K9S12(56)08 is detected. ID=0xec76
 0:K9S12(56)08 Program      1:K9S12(56)08 Pr BlkPage   2:Exit
Select the function to test :0

[SMC(K9S1208V0M) NAND Flash Writing Program]

Source size:0h~18b9fh

Available target block number: 0~4095
Input target block number:0
target start block number     =0
target size         (0x4000*n) =0x1c000
STATUS:Epppppppppppppppppppppppppppp
```

图 3-2 sjf2410 烧写 U-boot 的过程

（4） 烧写完毕后选择“2”退出 sjf2410。

烧写完成后，断开 JTAG 线，PC 机运行 DNW 串口调试软件。重启开发板后，DNW 中会输出以下信息，表明 U-boot 可以正常启动：

```
U-Boot 1.2.0 (Apr 28 2008 - 15:13:44)

RAM Configuration:
Bank #0: 30000000 64 MB
NAND:       64 MB

In:      serial
Out:     serial
Err:     serial
```

其中，“LJD2410 >”即系统提示符，在此可以输入 U-boot 的命令并执

火龙果•整理 uml.org.cn

行。U-boot 提供了几十个常用的命令，通过这些命令，可以对开发板进行调试，可以引导 Linux 内核，还可以擦写 Flash 完成系统部署等功能。

输入“help”命令，可以看到 U-boot 当前的所有命令列表，如表 3-2 所示，每一条命令后面是简单的说明。

表 3-2　U-boot 中几个常用命令及其说明

命令	说明
bootm	从内核的入口地址引导内核。
tftp	利用 tftp 协议将主机上的映像文件下载到指定的内存地址中。
printenv	打印环境变量信息。
setenv	设置环境变量。
saveenv	保存环境变量到 Flash。
nand erase	擦除 Flash 中指定地址的数据。
nand write	将内存中指定地址的数据写到 Flash 上的指定地址。
nand read	将 Flash 中指定地址的数据读到内存指定地址中。

3.6　设置 U-boot 环境变量

U-boot 的环境变量存储在 NAND Flash 中 U-boot 程序映像后面的 128Kb 字节中，这部分被称为“变量区”。

本课题中，设置 U-boot 环境变量共有两种方法：

（1）在板级头文件 LJD2410.h 中定义有相关的环境变量宏。

这类的宏名称中以“CONFIG_”开头，区别于以“CFG_”开头的内部变量宏。以开发板的 IP 地址为例，LJD2410.h 中有如下代码：

```
#define CONFIG_IPADDR        192.168.0.10
```

这种方法定义简便，但是每次更改环境变量必须重新编译、烧写 U-boot 程序，操作复杂，不方便调试。

（2）使用命令设置环境变量。

这种方法得益于先前所做的移植工作，优点是操作简便，可以在线设置，重启开发板即可生效。还是以设置开发板 IP 地址为例，U-boot 提示符下输入以下命令：

```
LJD2410 > setenv ipaddr 192.168.0.10
LJD2410 > saveenv
```

系统显示：

火龙果•整理
uml.org.cn

Saving Environment to NAND...

Erasing Nand...Writing to Nand... done

表明新设置的环境变量已保存至 Flash 中的变量区。

3.7 本章小结

本章首先简单介绍了 Bootloader 的基本概念，然后结合 S3c2410 处理器对 U-boot 的启动过程做了简要分析，接着介绍了移植 U-boot 到 LJD2410 开发板的流程和几个关键步骤，最后介绍了 U-boot 的烧写过程和环境变量的设置方法。本章工作结束后，开发板可以正常启动，并且能够实现下载和启动内核的基本功能，为 Linux 内核移植打下了基础。

火龙果•整理
uml.org.cn

第 4 章 Linux 内核的移植

4.1 Linux 内核的结构

在对 Linux 内核移植之前，首先要明确内核源码的基本组织情况，只有了解了各目录级代码的功能才能准确找到需要修改和改进的地方。

Linux 内核主要由 5 个子系统组成：进程调度、内存管理、虚拟文件系统、网络接口、进程间通信[15]。

Linux 内核源码中几个主要的目录说明如下：

（1）/arch 包含了所有硬件结构特定的内核代码。

Linux 系统能支持如此多平台的部分原因是因为内核把原程序代码清晰的划分为体系结构无关部分和体系结构相关部分。对于任何平台，都必须包含以下几个目录：

（a）boot:包括启动内核所使用的部分或全部平台特有代码。

（b）kernel:存放支持体系结构特有的(如信号处理和 SMP)特征的实现。

（c）lib:存放高速体系结构特有的(如 strlen 和 memcpy)通用函数的实现。

（d）mm:存放体系结构特有的内存管理程序的实现。

（e）math-emu:模拟 FPU 的代码。对于 ARM 处理器来说，此目录用 mach-xxx 代替。

（2）/drivers 包含了内核中所有的设备驱动程序。

（3）/fs 包含了所有的文件系统的代码。

（4）/include 包含了建立内核代码时所需的大部分库文件。

该目录也包含了不同平台需要的库文件。比如，asm-arm 是 arm 平台需要的库文件。

（5）/init 包含了内核的初始化代码，内核从此处工作。这是研究核心如何工作的好起点。

（6）/ipc 包含了进程间通信代码。

（7）/kernel 包含了主内核代码。

（8）/mm 包含了所有内存管理代码。

（9）/net 包含了和网络相关的代码。

（10）/documents 包含了内核源码各个部分的说明文件。

通常，在每个目录下，都有一个 Kconfig 文件和一个 Makefile 文件，这两个文件都是编译时使用的辅助文件，仔细阅读这两个文件对弄清各个文件

火龙果•整理
uml.org.cn

之间的联系和依托关系很有帮助；而且在有的目录下还有 Readme 文件，它是对该目录下的文件的一些说明，同样有利于我们对内核源码的理解。

显然，移植工作的重点就是移植 arch 目录下的文件。

4.2 Linux 启动过程简析

Linux 内核启动就是引导内核映像启动的过程。典型的内核映像是 zImage，包含自引导程序和压缩的 vmlinux 两部分。

启动过程从内核映像入口开始执行，解压 vmlinux 并转到虚拟地址空间；再调用统一的内核启动函数 start_kernel()，完成一系列基本初始化；随后启动一个叫做 init 的内核线程，完成挂载文件系统、初始化设备驱动和启动用户空间 init 进程等工作。

4.3 Linux 内核的移植过程

4.3.1 选择参考板

内核的移植工作主要是修改跟硬件平台相关的代码，一般不涉及 Linux 内核通用程序。移植的难度也取决于两种硬件平台的差异。Linux 对于特定硬件平台的软件叫做 BSP（Board Support Package）.

Linux 内核已经支持了各种体系的多种开发板，我们很容易从中找到与本课题类似的目标板，参考该目标板并做一定的修改，即可完成移植工作。选择参考板的原则如下：

（1）参考板与开发板具有相同的处理器，至少类似的处理器；

（2）参考板与开发板具有相同的外围接口电路，至少基本接口相同；

（3）Linux 内核已经支持参考板，至少有非官方的补丁或者 BSP；

（4）参考板 Linux 设备驱动工作正常，至少已经驱动基本接口。

根据以上原则，本课题选择 SMDK2410 作为参考板。修改顶层 Makefile 文件，指定体系结构和编译器地址：

```
ARCH := arm
CROSS_COMPILE := /usr/local/arm/3.4.1/bin/arm-linux-
```

4.3.2 修改 NAND Flash 分区信息

本课题中，NAND Flash 应按照功能分为 4 个分区，如图 4-1 所示：

火龙果•整理 uml.org.cn

U-boot	kernel	Rootfs	user

0x0　0x100000　0x400000　0x2d00000

图 4-1 NAND Flash 分区示意图

Linux 内核对于 Flash 分区由 arch/arm/plat-s3c24xx/common-smdk.c 文件中的 mtd_partition smdk_default_nand_part 结构体定义，默认已经分为了 8 个区。按照图 4-1 的分区信息，修改该结构体为：

```
static struct mtd_partition smdk_default_nand_part[] = {
[0] = {
    .name   = "U-boot",
    .size   = 0x00100000,
    .offset = 0x0,
},
[1] = {
    .name   = "Kernel",
    .offset = 0x00100000,
    .size   = 0x00300000,
},
[2] = {
    .name   = "RootFS",
    .offset = 0x00400000,
    .size   = 0x02800000,
},
[3] = {
    .name   = "User",
    .offset = 0x02d00000,
    .size   = 0x00f00000,
},
```

同时还应根据 CPU 手册修改 NAND Flash 的读写时序：

```
static struct s3c2410_platform_nand smdk_nand_info = {
 .tacls    = 0,
 .twrph0   = 30,
```

火龙果•整理 uml.org.cn


```
.twrph1        = 0,
};
```

4.3.3 关闭 ECC 校验

本设计中，内核都是通过 U-boot 写到 Nand Flash 的，U-boot 通过的软件 ECC 算法产生 ECC 校验码，这与内核校验的 ECC 码不一样，而内核中的 ECC 码是由 S3C2410 中 Nand Flash 控制器产生的。所以，我们在这里选择禁止内核 ECC 校验.，具体操作如下：

文件 drivers/mtd/nand/s3c2410.c 中，找到 s3c2410_nand_init_chip()函数，将最后一行的

```
chip-->eccmode          = NAND_ECC_SOFT
```

改为：

```
chip-->eccmode          = NAND_ECC_NONE
```

4.4 CS8900a 网卡的移植过程

本课题中使用的 LJD2410 开发板带有 CS8900A 网卡芯片，并提供 RJ-45 网络接口。Linux 内核中并没有为 ARM 体系配置 CS8900A 的网卡驱动，需要自己添加。CS8900A 的驱动文件有两个：CS8900A.h 和 CS8900A.c，这两个文件可以由网络获得，将其拷贝至 drivers/net/arm 文件夹下，但这样并不能使驱动程序正常工作，还应对内核源文件做些修改。

4.4.1 修改硬件地址映射

（1）在/arch/arm/mach-s2410 文件夹里建立文件 smdk2410.h，添加如下代码：

```
#define pSMDK2410_ETH_IO       __phys_to_pfn(0x19000000)
#define vSMDK2410_ETH_IO       0xE0000000
#define SMDK2410_EHT_IRQ       IRQ_EINT9
```

这三个宏分别定义了网卡的物理地址、虚拟地址和占用的中断号。

（2）修改/arch/arm/mach-s2410/mach-smdk2410.c，添加如下代码：

```
#include <asm/arch/smdk2410.h>
```

（3）在 map_desc smdk2410_iodesc[]结构题中添加 CS8900A 对于的 io 空间映射：

```
static struct map_desc smdk2410_iodesc[] __initdata = {
{ vSMDK2410_ETH_IO , pSMDK2410_ETH_IO, SZ_, MT_DEVICE },
```

火龙果•整理 uml.org.cn


```
};
```

4.4.2 添加 CS8900A 内核编译项

Kconfig 文件是 Linux2.6 内核引入的配置文件，是内核配置选项的源文件。只有在这个文件里加入相应代码，才能在编译选项中出现菜单项。

在/drivers/net/arm/Kconfig 中增加 CS8900A 的编译项代码：

```
config   ARM_CS8900
      tristate "CS8900 support"
      depends on NET_ETHERNET && ARM && ARCH_SMDK2410
      help …
```

最后应在/drivers/net/arm/Makefile 中添加：

```
obj-$(CONFIG_ARM_CS8900)        += cs8900.o
```

以上工作完成后，新移植的 CS8900A 驱动就可以编译进内核里了。

4.5 Linux 内核的剪裁配置

配置内核选项是整个移植过程中很重要的一步，本设计使用 SMDK2410 作为参考开发板，所以可以参考内核中 SMDK2410 开发板的配置文件，通过以下命令将其复制到内核根文件夹下：

```
$ cp arch/arm/config/smdk2410_defconfig .config
```

在此基础上，根据本课题的实际需求进行配置增减。

4.5.1 使用配置菜单

配置内核可以选择不同的配置界面，图形界面或者光标界面。由于光标菜单运行时不依赖于 X11 图形软件环境，可以运行在字符终端上，所以光标菜单界面比较通用[16]。图 4-2 所示就是执行 make menuconfig 出现的配置菜单。

在各级子菜单中，选择相应的配置时，有 3 种选择，它们代表的含义分别如下：

Y—将该功能编译进内核。

N—不将该功能编译进内核。

M—将该功能编译成可以在需要时动态插入到内核中的模块。

火龙果•整理
uml.org.cn

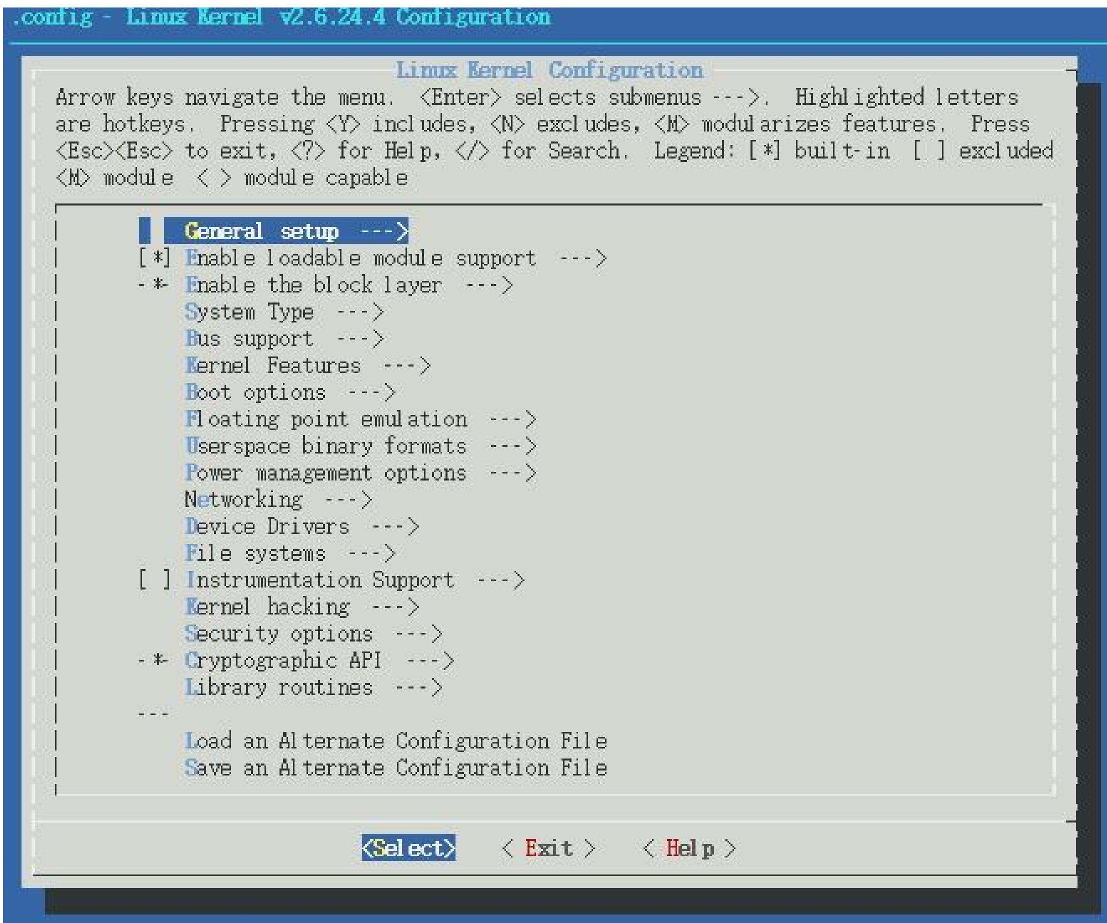

图 4-2 内核配置主菜单

内核配置原则是：将与内核其他部分关系较远且不经常且不经常使用的部分功能代码编译成可加载模块，有利于减少内核长度，减小内核消耗的内存，简化该功能相应的环境改变时对内核的影响；不需要的功能就不选；与内核关系紧密而且经常使用的部分功能代码直接编译到内核中。

4.5.2 基本配置选项

Linux 内核的各个版本配置餐单各不相同，下面以本课题使用的 2.6.24.4 版为例，结合本课题的实际需求，简介下内核的基本配置选项。

（1）General setup：包含通用的一些配置选项，保持默认即可。

（2）Enable loadable module supple：包含支持动态模块的配置选项，保持默认。

（3）System Type：包含系统平台列表及其相关的配置，去掉 SMDK2410

火龙果•整理 uml.org.cn

以外所有开发板的支持、开启 s3c2410 DMA 支持。

（4）Bus support：包含各种总线配置选项，全部去掉。

（5）Kernel Features：包含内核特性相关选项，保持默认。

（6）Boot options：包含内核启动相关选项，其中内核启动参数设置为："noinitrd console=ttySAC0,115200 root=/dev/nfs init=linuxrc nfsroot=192.168.0.2:/home/arm/FS/myrootfs mem=64M ip=192.168.0.10:192.168.0.2:192.168.0.1:255.255.255.0:LJD2410:eth0:off"，支持 NFS 文件系统。

（7）Floating point emulation：包含浮点数运算仿真功能，需要开启"NWFPE"选项。

（8）Userapace binary formats：包含支持的应用程序格式，仅保留"ELF"格式支持，去掉其它。

（9）Power management options：包含电源管理功能，保持默认。

（10）Networking：包含网络功能：需要开启基本功能选项。

（11）Device Drivers：包含设备驱动选项，下一小节将详细介绍。

（12）File systems：包含各种文件系统的支持选项，去掉"EX2"等选项，仅保留 ROM 文件系统支持，在"Miscellaneous filesystems"子菜单中近保留"cramfs"文件系统支持，并且开启 NFS 文件系统支持，去掉其它选项。

（13）Kernel hacking：包含各种内核调试选项，保持默认。

（14）Security options：包含安全性有关选项，保持默认。

（15）Cryptographic API：包含加密算法，保持默认。

（16）Library routines：包含几种压缩和校验函数，保持默认。

4.5.3 驱动程序配置选项

几乎所有 Linux 的设备驱动都在"Device Drivers"菜单下，它对设备驱动程序加以归类，放在子菜单下。本课题对于设备驱动的裁剪较多，具体如下：

（1）MTD support：MTD 设备驱动，应添选 NAND Flash 驱动支持。

（2）Network debice support：网络设备支持，在子菜单"Ethernet（10 or 100Mbits）"中可以看到 CS8900A 网卡的配置项，这正是 4.4 节工作的结果。

（3）Real Time Clock：时钟驱动选项，应选上"Samsung S3C series SoC RTC"，这样系统时钟才能正常运行。

（4）由于嵌入式导航计算机只使用串口作为输入输出接口，所以应该剪裁掉那些无用的驱动，包括：并口、ATA 及 SATA 驱动、RAID 驱动、ISDN

火龙果•整理
uml.org.cn

支持、输入设备驱动、多媒体设备支持、USB 支持以及 MMC/SD 卡支持。

（5）其它驱动支持保持默认即可。

4.5.4 保存配置文件

内核配置主菜单中选择“Save an Alternate Configuration File”即可将目前的配置状态保存成文件。程序默认保存为“.config”，此文件位于内核根目录内，可以直接修改。

4.5.5 编译 Linux 内核

正式编译 Linux 内核之前，应当清理一下内核树，命令如下：

$ make mrproper

此命令会清除掉.config 文件，所以应当在配置内核之前做。

Linux 2.6 版本的编译已经简化，使用一个 make 命令就可以完成诸如建立文件依赖、生成 zImage、编译模块、安装模块等一系列功能。内核编译完成后，将在/arch/arm/boot 目录中生成 image 和 zIamge 两个内核映像文件，其中 image 为正常大小的映像文件，而 zImage 为压缩后的映像文件。此时编译好的可加载模块也被安装到预定位置，默认为/lib/modules。

4.6 内核的下载及启动

4.6.1 将引导信息加入内核映像

U-boot 引导内核时需要检查一个 64byte 的头信息，其中包含了入口地址、映像类型等基本信息。这个引导头可以用 U-boot 附带的 mkimage 工具生成，命令如下：

$ mkimage -n 'linux-2.6.24' -A arm -O linux -T kernel -C none -a 0x30008000 -e 0x30008040 -d zImage zImage.img

各个参数的含义[17]：

-n：设置映像名

-A：设置体系信息

-O：设置操作系统信息

-T：设置映像类型

-c：压缩类型

-a：读入地址

火龙果•整理
uml.org.cn

-e：入口地址

-d：源映像文件

该命令生成的 zImage.img 文件就可以下载到开发板运行了。

4.6.2 内核映像的下载及运行

将上一小节中生成的 zImage.img 文件拷贝到主机 tftpboot 文件夹内。启动开发板，进入 U-boot 提示符。使用 tftp 命令将内核映像下载到开发板内存中：

```
LJD2410> tftp 0x30008000 zImage.img
TFTP from server 192.168.0.2; our IP is 192.168.0.10
Filename ‘zImage.img’
Load address : 0x30008000
Loading: ##################################################################
Done
```

其中 0x30008000 为指定的下载到内存的地址，zImge.img 就是带有引导头的内核映像。当内核下载完成后，可以通过 bootm 命令启动内核：

```
LJD2410> bootm 0x30008000
```

4.7 本章小结

本章主要是 Linux 内核进行分析移植。首先分析了 Linux 内核的结构和启动流程；然后介绍将 Linux 内核以及 CS8900a 网卡驱动移植到 LJD2410 开发板的具体方法和剪裁配置；最后完成了 Linux 在开发板上的正常启动。

火龙果•整理
uml.org.cn

第 5 章 建立根文件系统

5.1 根文件系统概述

5.1.1 根文件系统简介

对于嵌入式操作系统而言，仅包含内核是不够的，还必须有文件系统的支持。跟文件系统（root filesystem）是 Linux 系统的核心部分，包含系统使用的软件和库，以及无偶有用来为用户提供支持架构和用户使用的应用软件，并作为存储数据读写结果的区域。在 Linux 系统启动时，首先完成内核安装及环境初始化，最后会寻找一个文件系统作为根文件系统被加载。Linux 系统中使用“/”来唯一表示根文件系统的安装路径。嵌入式系统中通常可以选择的根文件系统有:Romfs, CRAMFS, RAMFS,JFFS2, EXT2 等，甚至还可以使用 NFS(网络文件系统)作为根文件系统[18]。

5.1.2 NFS 文件系统与 Cramfs 文件系统

NFS（Network File System）是由 SUN 公司发展，并于 1984 年推出的一种文件系统。它可以让开发者通过网络连接，使开发板可以直接挂载主机的某一个指定文件夹作为根文件系统。在嵌入式开发过程中，通常使用这种文件系统搭建交叉编译环境。

cramfs (Compressed ROM File System)是 Linux 创始人 Linus Torvalds 开发的一个适用于嵌入式系统的文件系统。cramfs 是一个只读文件系统，采用了 zlib 压缩，压缩比一般可以达到 1:2，但仍可以做到高效的随机读取。Linux 系统中，通常把不需要经常修改的目录压缩存放，并在系统引导的时候再将压缩文件解开[19]。因为 cramfs 不会影响系统读取文件的速度，而且是一个高度压缩的文件系统，因此本课题最终选用 cramfs 作为根文件系统部署到开发板。

5.2 建立 Linux 根文件系统目录

嵌入式 Linux 根文件系统必须包含一些必须的目录，比如设备目录/dev、命令目录/bin、库目录/lib 等等。

本课题构建根文件系统的工作目录是 myrootfs，通过下列命令可以在 myrootfs 中创建所需的子目录：

火龙果•整理 uml.org.cn


```
$ mkdir bin dev etc lib proc sbin sys usr
$ mkdir usr/bin usr/lib usr/sbin lib/modules
$ mkdir mnt tmp var
$ chmod 1777 tmp
$ mkdir var/lib var/lock var/log var/run var/tmp
$ chmod 1777 var/tmp
$ mkdir home root boot
```

这样，一个基本的根文件系统就建立起来了，但是各个目录都是空的，缺少各种程序和命令工具，需要进一步完善。

5.3 移植 Busybox 工具

5.3.1 Busybox 工具简介

Busybox 使用一个文件整合很多微型版本的 UNIX 工具。它可提供多数 GNU 文件工具、shell 脚本工具，如 cp、sh、ls、mv 等，被形象的称为嵌入式 Linux 系统中的“瑞士军刀”。Busybox 的特色是所有命令都编译成一个文件——busybox，其他命令都是指向它的连接。在使用 Busybox 生成的工具时，会根据工具的文件名转到特定的处理程序。这样，所有这些程序只需被加载一次，而所有的 Busybox 工具组件都可以共享相同的代码段，这在很大程度上节省了系统的内存资源和提高了应用程序的执行速度。Busybox 仅需用几百 Kb 的空间就可以运行，这使得 Buxybox 很适合嵌入式系统使用[20]。

5.3.2 Busybox 的配置

Busybox 源码开放，可以从 http://www.busybox.net 下载源码包 busybox-1.9.1.tar.bz2。

解压源码后，首先应当修改 makefile 文件，指定开发板体系和编译器地址。Busybox 的配置方法与 Linux 内核配置相似，也是通过 make menuconfig 命令使用光标菜单，在默认的基础上，本课题所做的修改如下：

（1）Busybox Settings——General Configuration——Build Busybox as a static binary，设置成静态编译方式，这样可以省去共享库文件。

（2）Login/Password Management Utilities，登陆管理功能，去掉所有配置项，这部分功能将使用 TinyLogin 工具代替。

（3）Shells——ash，本课题使用 ash 作为默认的操作界面。

火龙果•整理
uml.org.cn

5.3.3 编译并安装 Busybox

确保以上工作无误后，使用 make 命令编译 Busybox 工具[21]。

编译完成之后，再通过 make install 生成可以用于根文件系统的工具集，默认安装位置是 busybox/_install，可以通过修改 makefile 文件的“PREFIX=”这一项重新指定安装位置。

最后将生成的文件复制到 myrootfs 中，并覆盖相应的空目录。

5.4 移植 Tinylogin 工具

Tinylogin 是 Busybox 的一个插件，用于用户登陆管理。其源码包可以从 http://tinylogin.busybox.net/downloads/tinylogin1.4.tar.bz2 下载。

5.4.1 编译 Tinylogin 工具

首先应该修改 Makefile 文件，具体如下：

（1）指明静态编译，不连接动态库：

DOSTATIC = true

（2）指明 TinyLogin 使用自己的算法来处理用户密码：

USE_SYSTEM_PWD_GRP = false

USE_SYSTEM_SHADOW = false

（3）指明交叉编译器：

CROSS=arm-linux-

（4）指明安装目录：

PREFIX=/home/arm/FS/myrootfs/

以上修改无误后，可以用 make all install 命令安装 Tinylogin 工具。

5.4.2 建立登陆密码文件

与 Linux 系统登陆密码相关的文件共有 3 个：passwd、shadow 和 group。本课题建立这三个文件的策略是：拷贝主机相应文件并在其之上修改。这三个文件的位置在/ect，使用 cp 命令可以完成拷贝工作：

```
$ cd /home/arm/FS/myrootfs/
$ cp /etc/passwd .
$ cp /etc/shadow .
$ cp /etc/group .
```

火龙果•整理 uml.org.cn

修改文件，仅保留 root 帐户相关的信息。修改后的文件详见附录 B。

5.5 建立初始化文件

5.5.1 Inittab 文件

内核加载完成后，Busybox 会转到 init 进程开始执行。当 init 进程对控制台的初始化完成之后，Busybox 会去解析/etc/inittab 文件并执行相应的运行级。Busybox 支持的 inittab 文件格式如下所示：

Id：runlevel：action：process

其中，action 是动作类型，包含以下几种：

（1）Sysinit：为 init 提供初始化命令行的路径。

（2）Respawn：在相应的程序结束时就重启。

（3）Wait：wait 动作会通知 init 必须等到相应的进程执行完之后才能继续执行其他的动作。

（4）Once：进程只执行一次，而且不会等待它完成。

（5）Ctrlaltdel：当按下 Ctrl+Alt+Delete 组合键时运行的进程。

（6）Shutdown：当系统关机时运行的进程。

（7）Restart：当 init 进程重启时运行的进程，事实上就是 init 本身。

5.5.2 Fstab 文件

文件/etc/fstab 存放的是系统中的文件系统信息。当正确的设置了该文件，则可以通过"mount /directoryname"命令来加载一个文件系统，每种文件系统都对应一个独立的行，每行中的字段都有空格或tab 键分开。同时 fsck、mount、umount 的等命令都利用该程序。

下面是/etc/fatab 文件的一个示例行：

fs_spec fs_file fs_type fs_options fs_dump fs_pass

其中各项的含义为：

（1）fs_spec：该字段定义希望加载的文件系统所在的设备或远程文件系统。

（2）fs_file：该字段描述希望的文件系统加载的目录点。

（3）fs_type：定义了该设备上的文件系统类型。

（4）fs_options：指定加载该设备的文件系统是需要使用的特定参数选项，多个参数是由逗号分隔开来。

火龙果•整理
uml.org.cn

（5）fs_dump：该选项被"dump"命令使用来检查一个文件系统应该以多快频率进行转储，若不需要转储就设置该字段为 0。

（6）fs_pass：该字段被 fsck 命令用来决定在启动时需要被扫描的文件系统的顺序，根文件系统"/"对应该字段的值应该为 1，其他文件系统应该为 2。若该文件系统无需在启动时扫描则设置该字段为 0。

5.5.3 Profile 文件

Profile 文件中存放了系统的环境变量。一般包括 Shell 命令工具的地址和库文件地址，还可能包含 Shell 相关的配置信息。

附录 C 中列出了 inittab、fstab 和 profile 这三个文件的具体内容。

5.6 建立启动脚本文件

系统初始化过后，将执行/etc/init.d/rcS 脚本文件，用户可以在这个文件中添加一些需要在登陆之前完成的命令。本课题中的 rcS 脚本主要包括下列几个命令：

（1）按照 fstab 中的定义挂载文件系统：

```
/bin/mount -a
```

（2）使用 Busybox 的 mdev 命令挂载设备文件：

```
mdev -s
/bin/echo /sbin/mdev > /proc/sys/kernel/hotplug
```

（3）设置开发板的 IP 和网络名：

```
/sbin/ifconfig eth0 192.168.0.10
/bin/hostname LJD2410
```

（4）从开发板 RTC 中读取日期时间信息：

```
/sbin/hwclock -s
```

（5）挂载 Tmpfs 文件系统：

```
/bin/mkdir /dev/shm
/bin/chmod 777 /dev/shm
/bin/mount tmpfs /dev/shm -t tmpfs
```

5.7 应用程序的建立

通过以上的工作，我们可以在开发板上运行起嵌入式 Linux 系统，并将主机上的 myrootfs 文件夹当作根文件系统挂载。在此基础之上，可以进行应

火龙果•整理 uml.org.cn

用程序的开发和调试，下面以一个最简单的 Hello World 程序为例，简单介绍下嵌入式应用程序的开发调试过程。

（1）建立 Hello World 程序。

创建一个简单的应用程序，一般需要三个文件，Makefile，.h 头文件、.c 主程序文件，另外，根据程序的架构的大小，也可能会有多个头文件和程序文件，这个需要随程序的大小而定。本节创建的 Hello World 程序源码详见附录 D。使用 make 命令就可以编译出能运行于 ARM 构架 CPU 上的 Hello World 程序，将此程序拷贝至跟文件系统中。

（2）添加所需的库文件。

由于使用了动态编译的方法，程序中需包含一些程序共享库，可以通过以下命令查看：

```
$ arm-linux-readelf helloword –a
Dynamic segment at offset 0x4fc contains 21 entries:
标记          类型                          名称/值
0x00000001 (NEEDED)                        共享库：[libpthread.so.0]
0x00000001 (NEEDED)                        共享库：[libc.so.6]
0x0000000c (INIT)                          0x8294
…
```

从交叉编译器的库目录中拷贝相关的库文件：

```
$ cd /home/arm/FS/myrootfs/lib
$ cp   /usr/local/arm/3.4.1/arm-linux/lib/ld* .
$ cp /usr/local/arm/3.4.1/arm-linux/lib/libc-2.3.2.so .
$ cp /usr/local/arm/3.4.1/arm-linux/lib/libc.so.6 .
$ cp /usr/local/arm/3.4.1/arm-linux/lib/libpthread* .
```

（3）运行程序。

启动开发板，进入 Linux，通过以下命令就可以运行 HelloWorld 程序了：

```
# ./helloworld
Hello World!
```

这表明 HelloWorld 程序运行成功。

火龙果•整理
uml.org.cn

5.8 将 Linux 内核及根文件系统部署到开发板

5.8.1 重新修改、编译内核及启动脚本文件

当开发工作完成后，需要将 Linux 内核和根文件系统部署到开发板，这样就可以脱离交叉开发环境而独立工作。由于本课题的目标系统不需要网络支持，所以要重新修改内核，去除网络功能，主要修改如下：

（1） 内核配置主菜单中，Networking 子菜单以内的选项全部去掉。

（2） Device——Network debice support 子菜单内的选项全部去掉。

（3）修改 Boot options 菜单内的内核启动参数为：

“noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0,115200 mem=64M rootfs=cramfs”。

重新编译内核，得到 zImage 映像文件，再利用 mkimage 工具添加引导头信息，生成 zImage.img 文件。

根文件系统中，涉及到网络功能的也应该全部去掉，具体是要注释掉 /etc/init.d/rcS 脚本文件中的网卡初始化命令行。

5.8.2 内核的烧写固化过程

将内核固化到 NAND Flash 可以用 U-boot 的 nand 擦写命令。但需注意，写入 Flash 之前，必须先将待写区域擦除干净。启动开发板，进入 U-boot 提示符。

（1）将新编译的内核映像下载到开发板内存：

LJD2410>tftp 0x30008000 zImage.img

（2）将内存中的内核固化到 Flash 中：

LJD2410>nand erase 0x100000 0x200000

LJD2410>nand write 0x30008000 0x100000 0x200000

5.8.3 制作 Cramfs 根文件系统映像

制作 Cramfs 根文件系统映像需要使用 mkcranfs 工具，mkcramfs 的命令格式为：

Mkcramfs [-h] [-e edition] [-i file] [-n name] dirname outfile

其中，-h 为显示帮助信息；-e edition 为生成的文件系统中的版本号；-i file 是将一个文件映像插入这个文件系统之中；-n name 设定了 cramfs 文件系统

火龙果•整理
uml.org.cn

的名字；dirname 指明需要被压缩的整个目录树；outfile 为最终输出的文件。

执行以下命令可以将先前构建好的文件系统压缩为映像文件：

$ mkcramfs /home/arm/FS/myrootfs myrootfs.cramfs

5.8.4 根文件系统的固化过程

根文件系统的固化过程与内核固化过程相似，仍使用 U-boot 的 nand 擦写命令。

（1）将根文件系统下载到开发板内存：

LJD2410>tftp 0x30008000 myrootfs.cramfs

（2）将根文件系统固化到 Flash：

LJD2410>nand erase 0x400000 0x700000

LJD2410>nand write 0x30008000 0x400000 0x700000

5.9 本章小结

本章介绍了构建嵌入式 Linux 根文件系统的一般步骤，并介绍了系统部署的方法。构建根文件系统要用到 Busybox 和 Tinylogin 这两个工具，并且需要根据需求自己编写初始化文件和启动脚本文件。构建好的根文件系统可以通过 NFS 挂载到开发板上，实现交叉开发。当开发结束后，应当将根文件系统做成 Cramfs 格式的映像文件，与内核映像一起部署到开发板。这样，开发板就可以脱离交叉开发环境而独立工作，开发板的启动过程详见附录 E。

火龙果•整理
uml.org.cn

结论

本课题的目标是为基于ARM9处理器的导航计算机移植Linux操作系统。研究过程中，使用了LJD2410型开发板，此开发板的处理器是基于ARM920T的Samsung S3c2410，能够满足嵌入式导航计算机的硬件需求。本课题所做的工作简要总结如下：

首先，本文对嵌入式系统、嵌入式Linux操作系统和ARM体系处理器做了简单介绍，并且分析了嵌入式导航计算机的操作系统需求。

其次，介绍了交叉开发环境的建立。本课题两台主机连接开发板的方法，主机分别安装不同的操作系统，在开发过程中完成不同的工作。通过TFTP和NFS等网络服务，实现高效连接，有利于提高开发效率。这部分是整个课题的基础，之后的所有工作都是在这个基础上完成的。

第三，本文重点介绍了Linux系统的移植过程。Linux系统移植包括三个方面：启动加载程序（Bootloader）的移植，Linux内核的移植和根文件系统的建立。本课题选用功能强大的U-boot作为启动加载程序，通过对其源代码进行修改，使其可以正常运行于开发板，并且实现下载、烧写等功能。内核则采用了2008年4月发布的2.6.24.4版本，移植了网卡驱动，并针对课题需求，进行了修改和裁剪，使得内核加载更快，运行更稳定。根文件系统选用了Cramfs文件系统，这种文件系统采用压缩格式，存储空间需求小，但是不影响读取速度，非常适合与嵌入式Linux系统。这三个方面的工作有前后继承关系，但是又有一定独立性，移植过程中应多调试，多实验。

最后，简单介绍了系统部署的方法。将Linux内核和根文件系统部署到开发板后，开发板就可以脱离交叉开发环境而独立运行，最终达到设计需求。

本课题充分利用前人积累的经验，结合最新的软件版本进行移植工作。在移植过程中遇到了许多困难和问题，主要靠查阅文献和自己的试探性试验来研究问题，通过多次的实践，最终得到明确的解决方法。

虽然移植后的Linux系统可以正常运行在开发板上，能满足设计需求。但是由于时间仓促，有许多问题没有深入研究，难免会出现一定的疏漏和瑕疵，这需要我在今后的学习中不断努力，加以改进。

火龙果•整理
uml.org.cn

致谢

本论文得到了导师曲延滨教授的悉心指导，在选题、开题和研究方法方面，都给予了支持和关心。当遇到挫折的时候，也是曲老师给予热情的鼓励和肯定。曲老师严谨的治学态度、开明的思想以及对学生无微不至的关怀和爱护，是我大学生涯中一份珍贵的回忆。在此谨致真挚的敬意和感谢！

还要感谢哈尔滨工业大学（威海）所有的老师，你们是我学习的榜样。

感谢我的父母，没有你们，就没有我的今天。你们的支持与鼓励，永远是支撑我前进的强大动力。

感谢实验室的几位同学，尤其是黄正仙师兄，你们给予了我技术上的支持，并不吝把研究成果与我分享。

最后，我要感谢这四年中与我共同奋斗的所有同学们，谢谢你们！

火龙果•整理
uml.org.cn

参考文献

1．孙天泽、袁文菊、张海峰.《嵌入式设计及 Linux 驱动开发指南——基于 ARM9 处理器》，电子工业出版社，2005。

2．郭秋平.《基于 ARM 系统的 Linux 平台移植研究》，浙江大学硕士学位论文，2006。

3．赵巧宁.《基于 ARM9 的嵌入式 Linux 开发研究》，西安电子科技大学硕士学位论文，2007。

4．李亚峰、欧文盛等.《ARM 嵌入式 Linux 开发从入门到精通》，清华大学出版社，2007。

5．ARM Limited Corporation. *ARM Architecture Reference Manual*, ARM Limited Corporation, 1996-2000

6．Samsung Electronics.《S3C2410X USER'S MANUAL Revision 1.2》，2003。

7．孙纪坤、张小全等.《嵌入式 Linux 系统开发技术详解——基于 ARM》，人民邮电出版社，2006。

8．(美) Benjamin Mako Hill.《Ubuntu 官方指南》，宋吉广译，人民邮电出版社，2007。

9．邓士昌.《Linux 指令语法词典》，中国铁道出版社，2006。

10．孙雨.《基于 ARM 的嵌入式系统移植设计》，长春理工大学硕士学位论文，2007。

11．刘红丹.《基于 ARM 的嵌入式 Linux 移植和裁剪研究》，哈尔滨工程大学硕士学位论文，2007。

12．曹程远.《U- Boot 在 S3C2410 上的移植》，《微型电脑应用》2005 年第 21 卷第 7 期。

13．宋凯、甘岚、严丽平.《U - Boot 在 S3C2410 上的移植分析》，《华东交通大学学报》2005 年第 22 卷第 5 期。

14．李园园等.《U-Boot 的分析及其在 S3C2410 上的移植》，《青岛科技大学学报(自然科学版)》2007 年第 28 卷第 3 期。

15．秦蔚.《ARM 平台下 Linux 内核移植技术的分析研究与应用》，昆明理工大学硕士学位论文，2004。

16．张维维等.《基于 S3C2410X 的 Linux 移植研究》，《大连民族学院学报》，2007.9。

17．《U-BOOT 下使用 bootm 引导内核方法》，http://www.cnitblog.com/

火龙果•整理
uml.org.cn

18．李庆诚等.《基于 NAND 型闪存的嵌入式文件系统设计》,《计算机应用研究》2006 年第 4 期。

19．张明磊等.《几种源码开放的嵌入式文件系统分析与比较》,《单片机与嵌入式系统应用》2007 年第 11 期。

20．熊伟、董金明.《嵌入式 Linux 中根文件系统的实现》,《电子测量技术》2007 年第 30 卷第 7 期。

21．邵长彬等.《用 Busybox 制作嵌入式 Linux 根文件系统》,《微计算机信息》（嵌入式与 SOC）2007 年第 23 卷第 10-2 期。

22．马忠梅等.《ARM & Linux 嵌入式系统教程》，北京航空航天大学出版社，2004。

23．谭浩强.《C 程序设计（第二版）》，清华大学出版社，2002。

24．Craig Hollabaugh. *Embedded Linux Hardware, software and Interface*. Addison-Wesley, 2002.

25．Gajski D D. *Specification and design of embedded software-hardware System*, IEEE Design & Test of Computers,1995,12(1)

火龙果•整理
uml.org.cn

附录 A 移植 U-boot 过程中涉及的文件

A-1 start.s

```
relocate:
copy_myself:
	@ reset NAND
	mov	r1, #S3C2410_NAND_BASE
	ldr	r2, =0xf830	@ initial value enable
	str	r2, [r1, #oNFCONF]
	ldr	r2, [r1, #oNFCONF]
	bic	r2, r2, #0x800		@ enable chip
	str	r2, [r1, #oNFCONF]
	mov	r2, #0xff	@ RESET command
	strb r2, [r1, #oNFCMD]
	mov	r3, #0			@ wait
1:	add r3, r3, #0x1
	cmpr3, #0xa
	blt	1b
2:	ldr	r2, [r1, #oNFSTAT]	@ wait ready
	tst	r2, #0x1
	beq 2b
	ldr	r2, [r1, #oNFCONF]
	orr	r2, r2, #0x800		@ disable chip
	str	r2, [r1, #oNFCONF]

	/* stack setup */
	ldr	r0, _TEXT_BASE		/* upper 128 KiB: relocated uboot	*/
	sub	r0, r0, #CFG_MALLOC_LEN		/*malloc area*/
	sub	r0, r0, #CFG_GBL_DATA_SIZE	/*bdinfo*/
#ifdef CONFIG_USE_IRQ
	sub	r0, r0, #(CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ)
#endif
```

火龙果•整理
uml.org.cn


```
	sub	sp, r0, #12		/* leave 3 words for abort-stack	*/

	@ copy u-boot to RAM
	ldr	r0, _TEXT_BASE
	mov		r1, #0x0
	mov	r2, #CFG_UBOOT_SIZE
	bl	nand_read_ll

	tst	r0, #0x0
	beq	ok_nand_read

ok_nand_read:
	@ verify
	mov	r0, #0
	ldr	r1, _TEXT_BASE
	mov	r2, #0x400	@ 4 bytes * 1024 = 4K-bytes
go_next:
	ldr	r3, [r0], #4
	ldr	r4, [r1], #4
	teq	r3, r4
	bne	notmatch
	subs	r2, r2, #4
	beq	done_nand_read
	bne	go_next
notmatch:
1:	b	1b
done_nand_read:
```

A-2 /drivers/nand_legacy/nand_legacy.c

```
……
#include <s3c2410.h>
……
/*
```

火龙果•整理 uml.org.cn


```
 * NAND flash initialization.
 */
typedef enum {
	NFCE_LOW,
	NFCE_HIGH
} NFCE_STATE;

static inline void NF_Reset(void);
static inline void NF_Init(void);
static void NF_Conf(u16 conf);

static void NF_Cmd(u8 cmd);
static void NF_CmdW(u8 cmd);
static void NF_Addr(u8 addr);
static void NF_SetCE(NFCE_STATE s);
static void NF_WaitRB(void);
static void NF_Write(u8 data);
static u8 NF_Read(void);
static void NF_Init_ECC(void);
static u32 NF_Read_ECC(void);

static inline void NF_Reset(void)
{
	int i;

	NF_SetCE(NFCE_LOW);
	NF_Cmd(0xFF);			// reset command
	for(i = 0; i < 10; i++);	// tWB = 100ns.
	NF_WaitRB();			// wait 200~500us;
	NF_SetCE(NFCE_HIGH);
}

void nand_init(void)
```

火龙果•整理 uml.org.cn


```
{
        S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
        NF_Init();
        printf ("%4lu MB\n", nand_probe((ulong)nand) >> 20);
}

static inline void NF_Init(void)
{
    #if 0           // a little bit too optimistic
        #define TACLS 0
        #define TWRPH0 3
        #define TWRPH1 0
    #else
        #define TACLS 0
        #define TWRPH0 4
        #define TWRPH1 2
    #endif

NF_Conf((1<<15)|(0<<14)|(0<<13)|(1<<12)|(1<<11)|(TACLS<<8)|(TWRPH0<<
4)|(TWRPH1<<0));
        // 1    1      1       1,     1          xxx,    r xxx, r xxx
        // En 512B 4step ECCR nFCE=H tACLS     tWRPH0 tWRPH1
        NF_Reset();
}

static    void NF_Conf(u16 conf)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    nand->NFCONF = conf;
}

static void NF_Cmd(u8 cmd)
{
```

火龙果•整理 uml.org.cn


```
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    nand->NFCMD = cmd;
}

static void NF_CmdW(u8 cmd)
{
    NF_Cmd(cmd);
    udelay(1);
}

static void NF_Addr(u8 addr)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    nand->NFADDR = addr;
}

static void NF_SetCE(NFCE_STATE s)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    switch (s) {
        case NFCE_LOW:
            nand->NFCONF &= ~(1<<11);
            break;
        case NFCE_HIGH:
            nand->NFCONF |= (1<<11);
            break;
    }
}

static void NF_WaitRB(void)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    while (!(nand->NFSTAT & (1<<0)));
```

火龙果•整理
uml.org.cn


```
}

static void NF_Write(u8 data)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    nand->NFDATA = data;
}

static u8 NF_Read(void)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    return(nand->NFDATA);
}

static void NF_Init_ECC(void)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    nand->NFCONF |= (1<<12);
}

static u32 NF_Read_ECC(void)
{
    S3C2410_NAND * const nand = S3C2410_GetBase_NAND();
    return(nand->NFECC);
}
……
```

A-3 include/configs/smdk2410.h

```
……
#define CONFIG_COMMANDS \
            (CONFIG_CMD_DFL     | \
            CFG_CMD_CACHE  | \
            CFG_CMD_NAND    | \
```

火龙果•整理
uml.org.cn


```
            CFG_CMD_ENV       | \
            /*CFG_CMD_EEPROM |*/ \
            /*CFG_CMD_I2C     |*/ \
            /*CFG_CMD_USB     |*/ \
            CFG_CMD_NET       | \
            CFG_CMD_PING      | \
            CFG_CMD_REGINFO   | \
            CFG_CMD_DATE      | \
            CFG_CMD_ELF) & ~CFG_CMD_IMLS & ~CFG_CMD_FLASH
……
/*-----------------------------------------------------------------------
 * Physical Memory Map
 */
#define CONFIG_NR_DRAM_BANKS     1        /* we have 1 bank of DRAM
*/
#define PHYS_SDRAM_1            0x30000000 /* SDRAM Bank #1 */
#define PHYS_SDRAM_1_SIZE  0x04000000 /* 64 MB */

#define PHYS_FLASH_1       0x00000000 /* Flash Bank #1 */

#define CFG_FLASH_BASE        PHYS_FLASH_1
#define CFG_NO_FLASH      1

/* Nand Flash Setting */
#define CONFIG_S3C2410_NAND_BOOT

#define CFG_ENV_IS_IN_NAND       1
#define CMD_SAVEENV            1
#define CFG_ENV_OFFSET       0x20000
#define CFG_ENV_SIZE       0x20000                    /*128kB*/

#define CFG_UBOOT_SIZE       0x40000           /*256kB*/
```

火龙果•整理 uml.org.cn


```
#if (CONFIG_COMMANDS & CFG_CMD_NAND)
#define CFG_NAND_LEGACY    1
#define CFG_MAX_NAND_DEVICE 1     /* Max number of NAND devices */
#define SECTORSIZE          512
#define ADDR_COLUMN          1
#define ADDR_PAGE           2
#define ADDR_COLUMN_PAGE 3
#define NAND_ChipID_UNKNOWN       0x00
#define NAND_MAX_FLOORS         1
#define NAND_MAX_CHIPS          1

#define NAND_WAIT_READY(nand)NF_WaitRB()

#define NAND_DISABLE_CE(nand)  NF_SetCE(NFCE_HIGH)
#define NAND_ENABLE_CE(nand)   NF_SetCE(NFCE_LOW)

#define WRITE_NAND_COMMAND(d, adr)  NF_Cmd(d)
#define WRITE_NAND_COMMANDW(d, adr)     NF_CmdW(d)
#define WRITE_NAND_ADDRESS(d, adr)     NF_Addr(d)
#define WRITE_NAND(d, adr)          NF_Write(d)
#define READ_NAND(adr)              NF_Read()
/* the following functions are NOP's because S3C24X0 handles this in hardware
*/
#define NAND_CTL_CLRALE(nandptr)
#define NAND_CTL_SETALE(nandptr)
#define NAND_CTL_CLRCLE(nandptr)
#define NAND_CTL_SETCLE(nandptr)
```

A-4 default_env 函数

```
void default_env(void)
{
if (sizeof(default_environment) > ENV_SIZE)
   {
```

火龙果•整理
uml.org.cn


```
    puts ("*** Error - default environment is too large\n\n");
    return;
    }

memset (env_ptr, 0, sizeof(env_t))
memcpy (env_ptr->data,

default_environment,
sizeof(default_environment));

#ifdef CFG_REDUNDAND_ENVIRONMENT
env_ptr->flags = 0xFF;
#endif

env_crc_update ();
gd->env_valid = 1;
}
```

火龙果•整理
uml.org.cn

附录 B 登录密码相关的文件

B-1 password

root:x:0:0:root:/root:/bin/sh

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

B-2 shadow

root:1rNgSX$RKJSUU25.p84kZhRKbjX51:14021:0:99999:7:::

daemon:*:13801:0:99999:7:::

bin:*:13801:0:99999:7:::

B-3 group

root:x:0:

daemon:x:1:

bin:x:2:

火龙果•整理
uml.org.cn

附录 C 初始化文件

C-1 inittab

```
::sysinit:/etc/init.d/rcS
::respawn:-/bin/login
::restart:/sbin/init
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::shutdown:/sbin/swapoff -a
```

C-2 fstab

```
proc        /proc     proc     defaults     0     0
none        /tmp ramfs size=32m     0     0
mdev    /dev ramfs defaults     0     0
sysfs    /sys sysfs defaults     0     0
```

C-3 profile

```
# /etc/profile: system-wide .profile file for the Bourne shells
echo "Processing /etc/profile... "
# no-op
# Set search library path
echo "Set search library path in /etc/profile"
export LD_LIBRARY_PATH=/lib:/usr/lib
# Set user path
echo "Set user path in /etc/profile"
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
# Set PS1
echo "Set PS1 in /etc/profile"
export PS1="[$USER@LJD2410 \\w]\\$"
echo "Done"
```

火龙果•整理
uml.org.cn

附录 D Hello Wrold 程序源文件

D-1 helloworld.c

```
#include <stdio.h>
#include "helloworld.h"
int main(void)
{
	printf("hello,world!\n");
	return 1;
}
```

D-2 helloworld.h

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H
int main(void);
#endif
```

D-3 Makefile

```
TARGET = helloworld
DEP_FILES = helloworld.c
CC=/usr/local/arm/3.4.1/bin/arm-linux-gcc
CFLAGS =-I./\
	-Wstrict-prototypes\
	-pipe -lpthread	\
	-g -O2 -Wall
all:$(TARGET)
$(TARGET):$(DEP_FILES) helloworld.h
	$(CC) $(CFLAGS) -o $@ $(DEP_FILES)
clean:
	rm -f $(TARGET) *.o
```

火龙果•整理
uml.org.cn

附录 E 启动过程

U-Boot 1.2.0 (May 21 2008 - 10:02:23)
DRAM: 64 MB
NAND: 64 MB
In: serial
Out: serial
Err: serial

Hit any key to stop autoboot: 0

NAND read: device 0 offset 1048576, size 2097152 ...
2097152 bytes read: OK
Booting image at 30008000 ...

Image Name: 2.6.24.4
Created: 2008-05-26 6:53:37 UTC
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 915724 Bytes = 894.3 kB
Load Address: 30008000
Entry Point: 30008040
Verifying Checksum ... OK
XIP Kernel Image ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Linux version 2.6.24.4 (arm@Haier) (gcc version 3.4.1) #9 Mon May 26 14:50:48 CST 2008
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=00007177
Machine: LJD2410

火龙果•整理
uml.org.cn

Memory policy: ECC disabled, Data cache writeback
CPU S3C2410A (id 0x32410002)
S3C2410: core 200.000 MHz, memory 100.000 MHz, peripheral 50.000 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: noinitrd root=/dev/mtdblock2 init=linuxrc
console=ttySAC0,115200 mem=64M rootfstype=cramfs
irq: clearing subpending status 00000002
PID hash table entries: 256 (order: 8, 1024 bytes)
timer tcon=00500000, tcnt a2c1, tcfg 00000200,00000000, usec 00001eb8
Console: colour dummy device 80x30
console [ttySAC0] enabled
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 62976KB available (1624K code, 220K data, 84K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
S3C2410 Power Management, (c) 2004 Simtec Electronics
S3C2410: Initialising architecture
S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
DMA channel 0 at c4800000, irq 33
DMA channel 1 at c4800040, irq 34
DMA channel 2 at c4800080, irq 35
DMA channel 3 at c48000c0, irq 36
NetWinder Floating Point Emulator V0.97 (double precision)
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered

火龙果•整理
uml.org.cn

io scheduler cfq registered
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
s3c2410-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2410
s3c2410-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2410
s3c2410-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2410
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: module loaded
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c2410-nand s3c2410-nand: Tacls=1, 10ns Twrph0=4 40ns, Twrph1=1 10ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
NAND_ECC_NONE selected by board driver. This is not recommended !!
Scanning device for bad blocks
Creating 4 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00000000-0x00100000 : "U-boot"
0x00100000-0x00400000 : "Kernel"
0x00400000-0x02c00000 : "RootFS"
0x02d00000-0x03c00000 : "User"
mice: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2410-rtc s3c2410-rtc: rtc disabled, re-enabling
s3c2410-rtc s3c2410-rtc: rtc core: registered s3c as rtc0
s3c2410-i2c s3c2410-i2c: slave address 0x10
s3c2410-i2c s3c2410-i2c: bus frequency set to 390 KHz
s3c2410-i2c s3c2410-i2c: i2c-0: S3C I2C adapter
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
s3c2410-rtc s3c2410-rtc: setting system clock to 2008-06-13 10:12:31 UTC (1224915151)
VFS: Mounted root (cramfs filesystem) readonly.
Freeing init memory: 84K

init started: BusyBox v1.9.1 (2008-05-14 18:28:29 CST)

火龙果•整理
uml.org.cn


```
starting pid 718, tty '': '/etc/init.d/rcS'
----------Mount all......
----------Starting mdev......
----------Setting date & time......
----------Mounting Tmpfs......

******************************************
          Zhang.L.Q 's LJD2410
     Linux kernel 2.6.24.4    Cramfs
          040230125    WHHIT
                2008.5
******************************************

******************************************
    Login name : root
    Password : 040230125
******************************************

starting pid 726, tty '': '/bin/login'

LJD2410 login: root
Password:
login[726]: root login    on `console'

Processing /etc/profile...
Set search library path in /etc/profile
Set user path in /etc/profile
Set PS1 in /etc/profile
Done

[root@LJD2410 /root]#
```