

武汉科技大学

硕士学位论文

基于ARM的WiFi无线通信终端的研究与实现

姓名：刘芳华

申请学位级别：硕士

专业：电路与系统

指导教师：周凤星

2010-10-20

摘 要

随着“无线城市”概念的提出，WiFi 无线通信技术得到了迅速发展，WiFi 已成为当今无线网络接入的主流标准。国内外许多地区都提供了 WiFi 信号覆盖域，只要随身携带的电子产品上有 WiFi 终端，便可接入互联网。ARM 是目前进行便携式电子产品开发的主流芯片，因此，对 ARM 架构下 WiFi 无线通信终端的研究具有非常重要的意义。

本文基于三星公司的 S3C2440 ARM 处理器和 Linux 嵌入式操作系统，对 Marvel I 88w8686 WiFi 无线网络控制器的应用进行研究，实现一个可以在 WiFi 热点区域接入互联网、功耗小的无线通信终端。在软件方面，使用基于 ADS 集成开发平台的 Bootloader，采用嵌入式 Linux 系统作为操作系统，使用 Qtopia 桌面环境，并采用基于 Qt 的 Konqueror 作为嵌入式浏览器。

本文阐述了无线通信终端软件部分从底层到顶层的实现。包括 Bootloader 的移植，Linux 内核的移植，根文件系统的制作，WiFi 无线网卡、LCD、触摸屏等设备驱动的移植，Qtopia 桌面环境的移植以及 Konqueror 浏览器的移植，最后还简单介绍了人机交互界面和电源管理等应用程序的实现。

经测试表明，该无线通信终端可以在 WiFi 信号覆盖区域接入因特网，能通过鼠标、键盘和触摸屏进行操作。在电源管理功能上，系统能在正常状态、低功耗状态和休眠状态之间进行正常切换，能正确进行休眠和唤醒。

关键词：无线通信；ARM；WiFi；嵌入式

Abstract

With the "wireless city" concept put forward, WiFi wireless communication technology developed rapidly, WiFi has become the mainstream standard of wireless Internet access technology. Since many areas at home and abroad has provided WiFi coverage domain, you can access the Internet as long as portable electronic products has WiFi terminals. At present, ARM is widely used in portable electronic product development, so it is of great value to do deep research on WiFi wireless communication terminal under ARM.

Based on Samsung's S3C2440 ARM processor and embedded Linux operating system, this paper has a research on the application of the Marvell 88w8686 WiFi wireless network controller, and implements a low-power wireless communication terminal which can access the Internet in the WiFi hotspot. On the software side, the system uses ADS-based integrated development platform Bootloader, the Qtopia desktop environment, and adopts embedded Linux system as the operating system, the Qt-based Konqueror as a browser.

This paper describes the software implementation of wireless communication terminal from the bottom to the top, including porting Bootloader, porting Linux kernel, making root file system, porting device drivers such as WiFi wireless network card, LCD and touch screen, transplanting Qtopia and Konqueror. Finally, the implementation of the interactive interface and power management is introduced.

The result of the test shows that the wireless communication terminal that we build can visit the Internet in WiFi coverage area. We Call also operate the terminal using mouse, keyboard and touch screen. In the power management features, the system can switch normally among the normal state, low-power state and sleep state, can sleep and wake up properly.

Keywords: Wireless Communication; ARM; WiFi; Embedded

武汉科技大学

研究生学位论文创新性声明

本人郑重声明：所呈交的学位论文是本人在导师指导下，独立进行研究所取得的成果。除了文中已经注明引用的内容或属合作研究共同完成的工作外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

论文作者签名： 刘芳华 日期： 2010.12.2

研究生学位论文版权使用授权声明

本论文的研究成果归武汉科技大学所有，其研究内容不得以其它单位的名义发表。本人完全了解武汉科技大学有关保留、使用学位论文的规定，同意学校保留并向有关部门(按照《武汉科技大学关于研究生学位论文收录工作的规定》执行)送交论文的复印件和电子版本，允许论文被查阅和借阅，同意学校将本论文的全部或部分内容编入学校认可的国家相关数据库进行检索和对外服务。

论文作者签名： 刘芳华

指导教师签名： 刘芳华

日 期： 2010.12.2

第一章 绪论

1.1 研究背景和意义

随着互联网越来越深入的走进人们的生活,用户对能够随时随地上网的需求越来越迫切。目前已经出现很多便携式电子设备,如智能手机、PDA、UMPC 和 MID 等,但它们大多通过 GPRS 或 3G 入网,费用较贵。Wi Fi (Wireless Fidelity) 是一种无线传输技术,工作在世界范围内无需任何电信运营执照的不收费频段,能够为其网络覆盖范围内的用户提供免费高速的无线宽带互联网访问。虽然很多 Wi Fi “热点”是免费接入的,但也有很多是由私人互联网服务提供商(ISP)提供的,会在用户接入互联网的时候收取一定的费用,但与在速度方面落后 Wi Fi 很多的而且费用昂贵的 WCDMA 或 TDCDMA 相比,Wi Fi 要便宜很多。因此,Wi Fi 将成为无线接入的首选,只要随身携带的电子设备集成了 Wi Fi 无线通信终端,用户就可以在 Wi Fi “热点”区域内随时拨打或接听电话、快速浏览网页、下载或上传音视频文件、收发电子邮件,而无需担心花费太高和网速太慢等问题。

为了响应以建设 Wi Fi 无线局域接入网络为主要标志的“无线城市”的号召,国内外许多地区都积极的拿出了 Wi Fi 网络覆盖方案。2008 年,香港特别行政区提出了“香港政府 Wi Fi 通”计划,在指定场所设置 AP (Access Point),方便市民免费的接入互联网;台湾也一直致力于无线城市的推广,目前台北市的 Wi Fi 网络覆盖率已经超过了 90%;美国的休斯顿为建设城市 Wi Fi 网络投资了 350 万美元,为各非赢利性团体,包括社区中心和学校等,提供免费而高速的无线网络连接;旧金山的新型公交车站计划最近也已经开启了,在每一个公交站都提供了一个小型的 Wi Fi 基站,在公交车站的周围即可以进行 Wi Fi 连接。到目前为止,日本和美国等发达国家依旧还是 Wi Fi 用户最多的地区,我国的许多大中城市的机场、车站、咖啡厅、酒店、图书馆等公共场也逐渐被 Wi Fi 信号所覆盖。随着城市建设的发展,Wi Fi 服务今后将变成城市基础设施建设的一部分,逐渐成为一种公共服务。

Wi Fi 已成为当今无线网络接入的主流标准,随着 Wi Fi 信号覆盖范围越来越广,Wi Fi 无线通信技术在各种便携式产品上的应用也将变得越来越多。目前已经出现了具备 Wi Fi 连接功能的智能手机和上网本,它们除了可以通过 GSM/CDMA 移动通信网通话以外,还可在 Wi Fi 无线局域网信号覆盖区域内接入因特网或进行 VoIP 通话,随着技术的进一步发展,各种 PDA、PSP、数码相机、投影仪和电视机等电子产品都将提供 Wi Fi 无线接入功能^[1]。

本文讨论的是一个基于 ARM 的 Wi Fi 无线通信终端的研究与实现,该无线通信终端可以通过 Wi Fi 无线网卡接入互联网,通过浏览器上网冲浪,并提供了控制系统功耗的电源管理功能。该 Wi Fi 无线通信终端是 Wi Fi 无线通信技术的一种实现,它不仅为便携式电子产品上实现 Wi Fi 无线接入功能提供了一种解决方案,而且,在其基础上还能扩展开发出具备 Wi Fi 无线接入功能的各种电子产品。

1.2 研究现状及发展方向

Wi Fi 是一种可以将个人电脑、手持设备（如 PDA、手机）等终端以无线方式互相连接的技术，它基于 IEEE 802.11 系列标准，该标准包括 IEEE 802.11、802.11a、802.11b 和 802.11g。其中 802.11a 标准使用 5GHz 频段，支持的最大速度为 54 Mb/s；802.11b 和 802.11g 标准都使用 2.4 GHz 频段，802.11g 提供最高 54Mbps 的数据传输速率，而 802.11b 是使用历史最长的 Wi Fi 技术，采用直接序列扩频 DSSS(Direct Sequence Spread Spectrum)技术和补偿码键控 CCK 调制方式，可提供 11Mbps 的无线传输速率，在有干扰或信号弱的情况下，可将带宽调整为 5.5 Mbps、2 Mbps、1 Mbps，信号覆盖范围为 80~100m^[2]。

随着主流应用的需求越来越高，只具备最高 11Mbps 的 802.11b 无线产品在市场上慢慢淡出，做为 802.11b 的继任者，802.11g 产品渐渐占据主导地位，在 802.11g 基础上还出现了众多的 802.11g 增强版无线产品，最高传输速率可达 125Mbps。为了使无线局域网达到以太网的性能水平，能够给用户 提供高带宽、高质量的 WLAN 服务，Wi Fi 联盟推出 802.11n 无线传输标准协议，该标准使用了将 MIMO 与 OFDM 技术相结合的 MIMO OFDM 技术，无线传输质量和传输速率得到了提高，802.11n 可以提供 300Mbps 甚至高达 600Mbps 的传输速率，覆盖范围扩大到好几平方公里，不仅能够支持向前后兼容，而且可以实现 WLAN 与无线广域网的结合。

Wi Fi 可移动性强和价格低廉的特点使它从一开始就被定位为高速有限接入网络的补充，用于有线网络不方便接入的领域，如临时会场等。蜂窝移动通信具有覆盖面积广、移动性强和中低等数据传输速率等特点，通过与传输速率高的 Wi Fi 技术结合，它可以弥补自身数据传输速率受限的不足，与此同时，Wi Fi 也可结合蜂窝移动通信网络覆盖面广的特点实现多接入切换功能，还可以借助蜂窝移动通信网络完善的计费机制与鉴权规范自身的体制。因此，逐渐实现 Wi Fi 对蜂窝移动通信的补充，实现 Wi Fi 与蜂窝移动通信的融合，对蜂窝移动通信和 Wi Fi 都非常有好处。

第三代蜂窝移动通信（3G）技术具有较强的业务能力、较高的技术先进性和相当广泛的应用，是一个比较完美的系统。但是，3G 在室内的数据速率只有 2Mbps，此时 Wi Fi 就具有绝对的优势；而且，3G 分配的频率资源有限，并且数据业务对信道的占用率非常高，同时接入的语音用户数量很容易受到影响，如果在特定区域内把数据业务转移到 Wi Fi 公共数据通道，3G 无线网络资源利用率势必将大大提高；另外，3G 技术主要针对大话务量和多用户数的情况，各种广播式的语音业务更适合使用基于 IP 技术的 WLAN 网络。由此可以看出，在特定的区域和范围内 Wi Fi 技术可以对 3G 技术进行补充，而且，Wi Fi 与 3G 技术的结合一定具有非常广阔的发展前景。目前，结合 Wi Fi 技术的 3G 网络正进入部署阶段，Wi Fi /3G 双模技术很快就会得到应用。

结合上述介绍，归纳 Wi Fi 技术的发展方向如下：

- （1）扩大 Wi Fi 网络的覆盖范围，从热点到热区再到整个城市；
- （2）实现 Wi Fi 手持终端和 VoWLAN 业务的大量应用；

(3) 应用基于 IP 的 Wi Fi 技术和开放性的业务平台, 使 WLAN 网络变得更智能、更易管理;

(4) 实现不同等级安全方案的提供, 使企业或个人用户可以按照不同的性价比来选择适合自己的安全策略;

(5) 实现 Wi Fi 技术与 3G 技术的融合。

1.3 研究内容及论文结构

本课题基于三星公司的 S3C2440 ARM 处理器和 Linux 嵌入式操作系统, 对 Marvell 88w8686 Wi Fi 无线网络控制器的应用进行研究, 实现一个可以在 Wi Fi 热点区域接入互联网、功耗小的 Wi Fi 嵌入式无线通信终端。嵌入式系统开发需要软件和硬件的协同进行, 整个过程可分为硬件开发和软件开发两个部分, 本文主要介绍软件开发部分, 主要的研究内容包括: 软件开发环境搭建、设备驱动程序开发、电源管理功能实现和应用程序开发。

根据上述研究内容, 本论文共分为七章, 结构安排如下:

第一章 绪论。介绍本课题研究背景及意义、研究现状及未来发展、论文主要内容。

第二章 Wi Fi 无线通信终端的实现原理。介绍 Wi Fi 无线网络的基本结构和通信原理、嵌入式系统开发 Wi Fi 无线通信终端的总体方案。

第三章 Wi Fi 无线通信终端软件平台的搭建。包括交叉编译环境建立、Bootloader 移植、内核移植和根文件系统制作。

第四章 Linux 设备驱动实现与移植。介绍 Wi Fi、LCD 和触摸屏等设备驱动的实现与移植。

第五章 应用程序的开发与移植。主要包括: Wi Fi 无线通信人机交互界面程序开发、嵌入式浏览器的移植、电源管理应用程序开发。

第六章 系统测试。测试 Wi Fi 无线网络接入、上网浏览、电源管理等功能是否正常使用。

第七章 结束语。对本文所做的工作进行总结, 指出存在的不足, 提出将来可改进的方向。

第二章 Wi Fi 无线通信终端的实现原理

2.1 Wi Fi 无线网络技术

Wi Fi 实质上是一种商业认证,它是目前应用最广泛的 WLAN (Wireless Local Area Network, 无线局域网) 标准,具有 Wi Fi 标准认证的产品都符合 IEEE 802.11b 无线局域网规范。下面首先介绍 WLAN 技术和 IEEE 802.11b 无线局域网规范,在此基础上,介绍 Wi Fi 无线网络工作原理、组网模式以及 Wi Fi 与其它 WLAN 技术标准的比较。

2.1.1 WLAN 技术介绍

WLAN 是指通过无线通信技术将分布在一定范围内的计算机设备或者其它智能终端设备互联起来,构成可以实现资源共享和互相通信的网络体系。WLAN 最大的特点是不再使用网络电缆将计算机与网络终端连接起来,而是使用无线的连接方式,使得网络的组建和终端的移动更加方便灵活。

2.1.1.1 WLAN 的组成单元

一个无线局域网通常由工作站 (STA, Station)、无线介质 (Wireless Medium, WM)、无线接入点和主干分布式系统 (DS, Distribution System) 等几部分组成。现将各部分的功能与特点描述如下^[3]:

(1) 工作站 (STA): 它是无线局域网最基本的组成单元,是集成了无线网络设备的计算机或智能设备终端。其无线网络设备的作用是接收无线信号,连接到无线接入点,实现计算机或智能终端之间的无线连接。根据应用的不同,无线网络设备可以分为无线局域网卡、无线上网卡和蓝牙适配器等。

(2) 无线接入点: 无线接入点可以是无线 AP (Access Point), 也可以是无线路由器,主要负责连接所有无线工作站进行集中管理、收发无线信号实现数据交换、实现无线工作站和有线局域网之间的互联等工作,具有有线网络中交换机的作用。

(3) 无线介质 (WM): 无线介质是 WLAN 中 STA 和 STA、STA 和 AP 之间通信时发送的无线电波的传输媒质。WLAN 中的无线介质由无线局域网物理层标准定义,最常见的是空气,用来传输无线电波。

(4) 主干分布式系统 (DS): 一个 WLAN 所能覆盖的区域称为基本服务区域 (Basic Service Set, BSS), 它是构成 WLAN 的最小单元。为了使无线局域网覆盖的区域更大,需要把多个 BSS 连接起来,形成一个扩展服务区 (Extended Service Area, ESA), 分布式系统用来连接不同的 BSS 形成 ESA。

2.1.1.2 WLAN 的组成结构

WLAN 网络主要分为无中心网络和有中心网络两种^[4], 组建这两种类型的无线局域网所需的设备不同,而且网络结构也很不一样。

(1) 无中心网络

无中心网络又称 Ad-hoc 网络，用于多台无线工作站之间的直接通讯。一个 Ad-hoc 网络由一组具有无线网络设备的计算机组成，这些计算机具有相同的工作组名、密码和 SSID，只要互相都在彼此的有效范围之内，任意两台或多台计算机都可以建立一个独立的局域网络。该网络不能接入有线网，是最简单的 WLAN 网络结构，如图 2.1 所示。

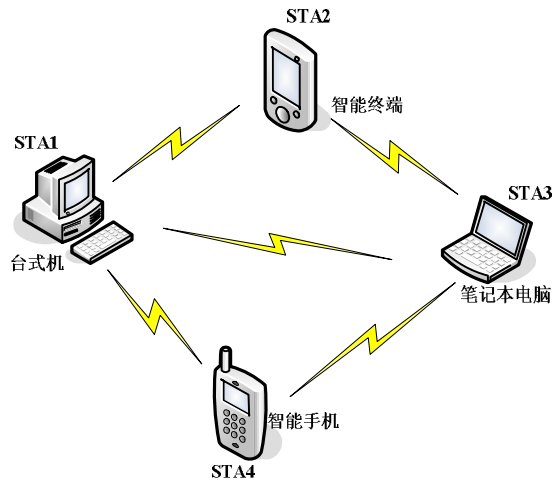


图2.1 无中心网络结构图

(2) 有中心网络

有中心网络又称结构化网络，它由 STA、WM 和 AP 组成，结构如图 2.2 所示。

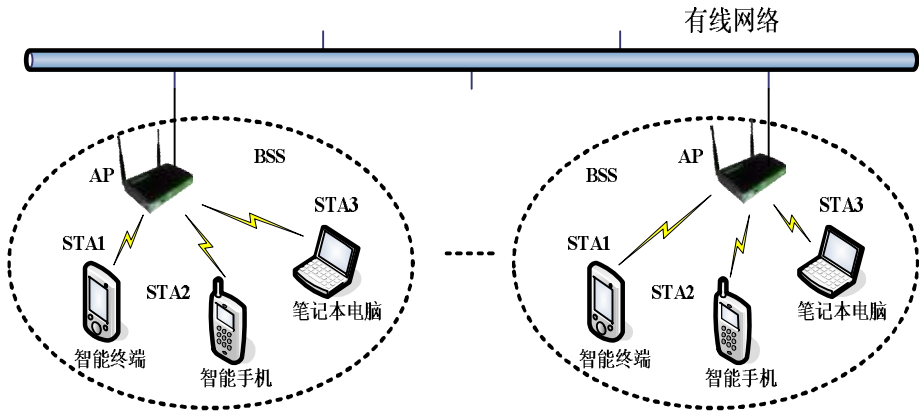


图2.2 有中心网络结构图

所有的工作站在本 BSS 以内都可以直接通信，但在和本 BSS 以外的工作站通信时都要通过本 BSS 的 AP 连接到有线网络来实现。WLAN 可以使通过无线设备联网的用户充分共享有线网络中的所有资源。

2.1.1.3 WLAN 技术标准

无线接入技术与有线接入技术的一个很大的不同点体现在无线接入技术标准不统一，不同的标准有不同的应用。目前比较常见的 WLAN 技术标准有 802.11 系列标准、蓝牙 (Bluetooth) 标准、IrDA(Infrared Data Association)、家用射频 HomeRF 标准以及新近兴起的 UWB 标准等。下表 2.1 是当前常用的 WLAN 协议标准的比较。

表2.1 各WLAN协议标准的比较

协议标准	射频频段	最高传输率(物理层)	备注
蓝牙 (Bluetooth)	2.4~2.485GHZ	2.1+EDR:3Mbps	实现语音和数据无线传输的开放性规范, 可方便快捷地实现各类数据及语音设备之间的通信
红外 (IrDA)	红外线	16 Mbps	相应软、硬件技术比较成熟; 体积小、功率低、抗干扰性强; 不适合障碍较多的地方及多个设备之间的数据传输, 传输距离一般在 2m 左右
家用射频 (HomeRF)	2.4~2.485GHZ	HomeRF2.x:10 Mbps	主要针对家庭网络设计: 在数据传输时, 采用 IEEE802.11 标准中的 TCP/IP 传输协议, 语音通信时则采用无绳电话 DECT 技术; 过于局限家庭应用, 前景不明
HiperLAN	HiperLAN1:5.3GHZ HiperLAN2:5GHZ	HiperLAN1:23.5 Mbps HiperLAN2:54 Mbps	在欧洲被广泛支持和应用, 主要是为集团消费者、公共和家庭提供无线接入和实时视频服务 HiperLAN/1 主要采用高斯滤波最小频移键控 GMSK 调制方式, HiperLAN/2 采用正交频分复用 (OFDM) 调制方式
超宽带 (UWB)	3.1~10.6GHz	600Mbps	新兴的无线局域网技术, 非常适合传输多媒体及其它的高带宽应用, 目前主要应用于短程、高分辨率、高带宽的雷达和成像系统, 初期产品传输距离约 10m, 数据传输率接近 100Mbps
IEEE802.11 系列	802.11:IrDA 或 2.4GHz 802.11a:5GHz 802.11:IrDA 或 2.4GHz 802.11b/g:2.4~2.485GHz 802.11n:2.4GHz,5GHz	802.11:2Mbps 802.11a:54Mbps 802.11b:11Mbps 802.11g:54Mbps 802.11n:300Mbps	由 IEEE802 工作组定义的无线局域网主流标准, 应用最为广泛

总的来说, HomeRF 比较适合用于家庭中语音设备或移动数据之间的通信; HiperLAN 标准主要在欧洲得到广泛的使用, 常作为蓝牙或 IrDA 应用场合的有益补充; UWB 是一种新型的无线传输标准, 发展前景很广阔; IEEE802.11 系列标准则更适用于校园网、企业构建内联网或者企业互联网, 应用最广泛, 是当前 WLAN 领域的主流标准, 也是 Wi-Fi 技术的认证标准, 下面将重点介绍。

2.1.2 IEEE802.11 系列协议

IEEE 制订的第一个 WLAN 标准就是 802.11, 该标准主要用于解决校园网中用户终端的无线接入和办公室的无线局域网。IEEE 802.11 应用到的关键技术主要有扩频 (Spread Spectrum, SS) 技术、红外 (Infrared) 技术、正交频分复用技术 OFDM (Orthogonal Frequency Division Multiplexing) 等, 其中, 扩频技术又分为直序扩频 (Direct Sequence Spread

Spectrum，DSSS）和跳频扩频（Frequency Hopping Spread Spectrum，FHSS）两种。

局域网的协议标准主要由物理层(Physics Layer ，PHY)、数据链路层(Data-Link Layer，DDL)和网络层三层组成。因为无线局域网不存在路由问题，所以没有单独设立网络层。数据链路层分为逻辑链路控制(Logical Link Control ，LLC)和媒体访问控制(Media Access Control ，MAC)两个子层^[5]。在 IEEE 802.11 标准中规定了 Wi Fi 网络的基本结构，包括物理层、MAC 层和逻辑链路控制层，图 2.3 显示的是 IEEE 802.11 标准中这三层的结构。

802.2 LLC(Logical Link Control)				
802.11 MAC(Media Access Control)				
802.11 PHY FHSS	802.11 PHY DSSS	802.11 PHY IR/DSSS	802.11 PHY OFDM	802.11 PHY DSSS/OFDM
802.11b 11Mbit/s 2.4GHz			802.11a 54Mbit/s 5GHz	802.11g 54Mbit/s 5GHz

图2.3 IEEE 802.11标准的三层结构

物理层定义了网络设备之间进行实际连接时的电气特性，该层直接与传输介质相连接，并且向上服务于数据链路层。它在各数据链路实体之间提供所需的物理连接，按特定的顺序传输数据位和进行差错检查，一旦发现错误，立即向数据链路层报告。从图 1.3 可以看出，Wi Fi 协议标准 IEEE 802.11b 工作在 2.4GHz 频段，主要用到 FHSS、DSSS 和 IR 等关键技术；标准 IEEE 802.11a 工作在 5GHz 频段，主要用到 OFDM 技术；IEEE 802.11g 工作在 5GHz 频段，主要用到 DSSS/OFDM 技术。

数据链路层最基本的服务是将本机网络层的数据无差错地传输到相邻节点的目标机网络层，主要功能有：将数据组合成数据块；控制帧在物理信道上的传输；帧同步功能；差错控制功能；流量控制功能和链路管理功能。IEEE 802.11b 标准规定的数据链路层的 MAC 子层使用载波侦听多路访问/冲突避免（CSMA/CA，Carrier Sense Multiple Access/Collision Avoidance）媒体访问控制协议来实现冲突检测和避免。

2.1.3 Wi Fi 网络工作原理

Wi Fi 无线上网原理如下图所示：

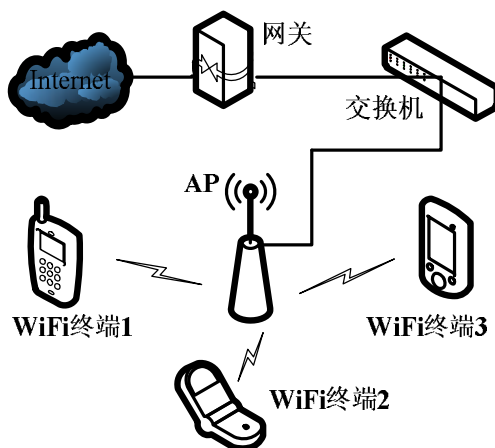


图2.4 Wi Fi无线上网图

Wi Fi 终端与其有效范围内的 AP 建立连接，AP 通过有线局域网络连接到交换机，交换机经由网关连接到互联网上，从而实现了 Wi Fi 无线上网。

AP 每 100ms 将服务单元标识 (Service Set Identifier, SSID) 经由信号台 (beacons) 封包广播一次，信号有效范围内的 Wi Fi 终端收到这个广播后，选择是否和这一个 SSID 的 AP 建立连接。由于信号台封包的传输速度和长度都很短，所以这个广播动作没有对网络的性能产生多大的影响，而且可以确保信号有效范围内所有的 Wi Fi 终端都能收到这个广播封包。如果某个 Wi Fi 终端的有效信号范围内有多个 AP，那么它将收到多个不同的 SSID 封包广播，这时可以选择其中信号好的 AP 来建立连接。

Wi Fi 无线网络的组网模式分为 Ad hoc 自组网络模式和基础结构 (Infrastructure) 组网模式，其中 Ad hoc 自组网络对应于 WLAN 的无中心网络，Infrastructure 组网对应于 WLAN 的有中心网络，可参见前面 WLAN 网络结构的介绍部分。

2.2 嵌入式系统开发介绍

2.2.1 嵌入式系统的组成

嵌入式系统一般由嵌入式硬件和软件组成。嵌入式硬件通常由微处理器和外围设备组成，而嵌入式软件则由实时多任务操作系统、各种专用软件 and 应用程序组成，具体组成结构如图 2.5 所示。

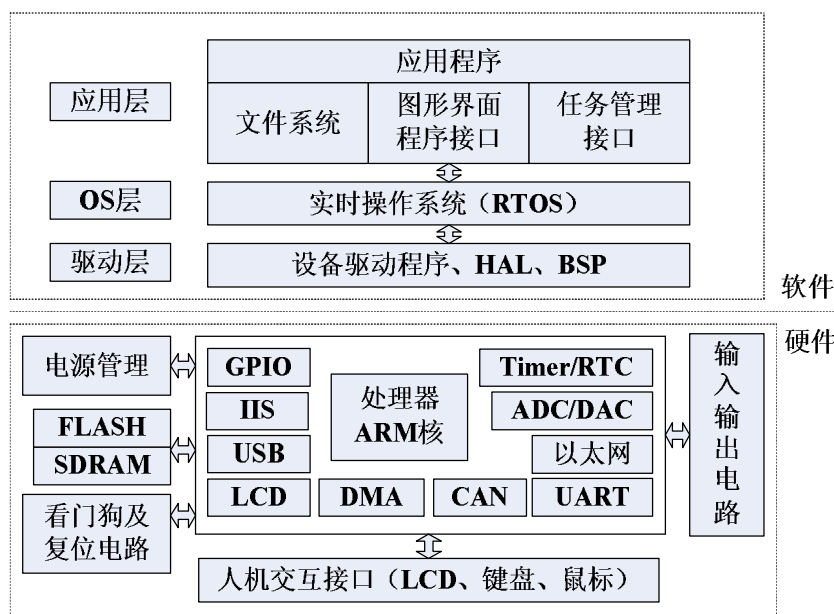


图2.5 嵌入式系统软硬件基本组成结构

嵌入式系统硬件分为嵌入式处理器、存储设备、电源电路、通信接口以及外围设备等几个部分。嵌入式处理器可以分为嵌入式微控制器（MCU）、嵌入式 DSP 处理器（DSP）、嵌入式微处理器（MPU）和嵌入式片上可编程系统（SOC）等^[6]，其中 MPU 主要包括：Motorola PowerPC、Motorola 68000、Intel Pentium、Strong ARM、MIPS、AMD X86 系列等，MCU 主要包括：MCS-51、MCS-96/196/296、MCS-251、C166/167、P51XA、68K、C540、Z8、AVR、PIC 等系列，目前广泛使用的 DSP 是 TI 产品系列以及 Intel 和 Siemens 的相应产品。常用的存储设备有 NAND FLASH、NOR FLASH、SDRAM 等^[7]。外围设备通常包括 LCD、键盘、鼠标、USB、UART、摄像头、蓝牙、Wi Fi、GPRS 等。

嵌入式系统软件可分为应用层、OS 层和 BSP 三层结构。板级支持包(BSP, Board Support Packet)主要完成底层硬件相关的初始化以及加载实时操作系统等工作，包含了 Bootloader 和系统硬件正常工作所需的部分驱动。目前常用的实时操作系统主要有 Linux、WinCE、ucos、Symbian、VxWorks 等^[8]，开发者根据开发需要选择合适的操作系统，然后对所选择的嵌入式操作系统进行裁剪、移植。应用层软件是针对用户特定的应用而制定的，其中文件系统必须要有，图形界面程序和其它应用程序依具体开发而定。

2.2.2 嵌入式系统开发流程

从嵌入式系统的组成结构可以看出，嵌入式系统开发分为硬件开发和软件开发两部分。嵌入式系统开发通常是利用 PC 机上良好的软硬件资源和开发环境来开发目标板上的软件，然后用交叉编译环境编译生成可执行文件和目标代码，通过以太网、串口或 USB 等方式下载到目标板上，分析调试，最后将调试好的目标文件下载固化到目标机上。从整体上来看，嵌入式系统开发流程如图 2.6 所示，主要包括用户需求分析、体系结构设计、软

硬件开发、系统集成、系统测试，直到得到最终产品^[9]。

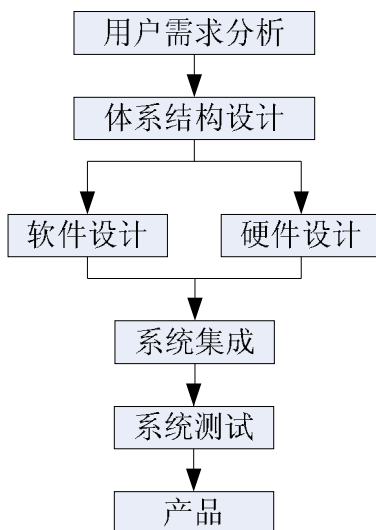


图2.6 嵌入式系统开发流程

(1) 用户需求分析。按照用户需求确定开发目标和开发任务，制定开发规格说明书，作为开发指导和验收标准。需求又分为功能性需求和非功能性需求，功能性需求主要是系统功能方面的需求，如操作方式、输入输出信号等；非功能需求包括所需成本、系统性能、功耗、重量、体积等因素^[10]。

(2) 体系结构设计。对系统如何实现功能和非功能需求的方法进行描述，包括对软硬件的选型、功能划分以及软硬件整体设计方案等。它是系统开发成功与否的关键。

(3) 软硬件协同设计。按照体系结构的设计方案，对软件、硬件进行详细设计并投入开发。嵌入式系统软硬件的开发往往是并行的，而且大部分工作都集中在软件开发上。

(4) 系统集成。把按照设计方案开发的软件、硬件集成在一起，进行调试，发现单元设计过程中的错误，并改进。

(5) 系统测试。对集成好的系统进行测试，看其是否满足先期制定的规格说明书中的各项功能要求。

2.2.3 系统开发平台的选择

本嵌入式 Wi Fi 无线通信终端的设计与实现所依赖的软硬件开发平台选择如下：

(1) 嵌入式处理器选择 S3C2440。开发板选用深圳优龙公司的 YLP2440，核心处理器是 Samsung S3C2440A，主频 400MHz，最高 533Mhz，内部集成 ARM 公司的 ARM920T 微处理器，扩展了一片 64M、32 位 NAND FLASH 和一片 64M、32 位 SDRAM，还扩展了很多其它的外围设备和引脚。

(2) 嵌入式操作系统选择 Linux。选择 Linux 实时操作系统最主要的原因是它的开源性，不仅可以从网上免费获取内核源码，还能获取很丰富的驱动代码，降低了开发难度。

(3) 图形界面程序选择用 QT 来开发。QT 是现在应用很广泛的图形界面程序开发工具，基于 C++ 编程语言，上手快，代码效率高，扩展性好。

2.3 Wi Fi 无线通信终端总体设计方案

按照系统需求，对本 Wi Fi 无线通信终端的总体方案进行了设计，其软硬件结构如图 2.7 所示。

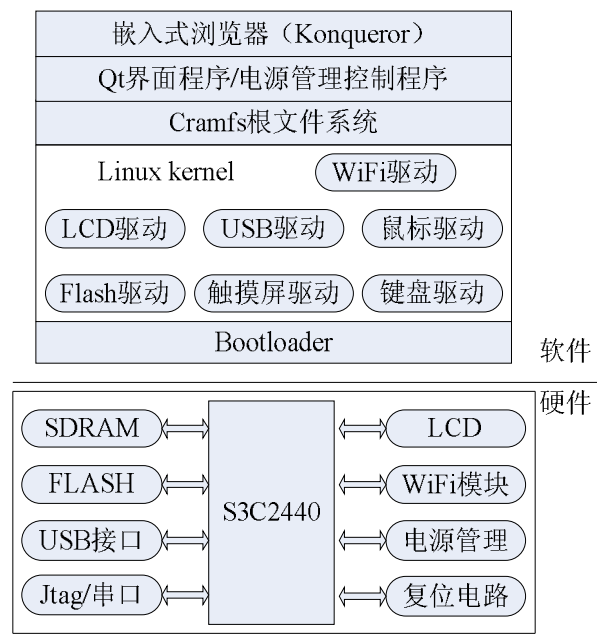


图2.7 Wi Fi 无线通信终端软硬件结构图

Wi Fi 无线通信终端硬件部分在 S3C2440 微处理器上扩展了一片 64M、32 位 SDRAM 用作系统内存，供程序运行使用；一片 64M、32 位 NAND FLASH 用来存放系统运行所需的软件；两个 USB 接口，分别用来接 USB 鼠标和 USB 键盘；一个 JTAG 接口和一个串口，用来调试和下载程序使用；一个带触摸屏的 LCD，用来显示人机交互界面；Wi Fi 功能模块，用来实现无线通信；电源管理模块，用来控制系统的功耗，节省电源消耗；最后是复位电路，复位时使用。

软件部分主要由 Boot loader、Li nux Kernel 、根文件系统和应用程序组成。Boot loader 由 ADS 集成开发环境开发，完成硬件设备初始化、下载文件到目标板、对 FLASH 进行擦写以及引导和加载内核镜像等工作；Li nux Kernel 是 Li nux 操作系统的基础，提供进程调度及进程通信、内存管理、文件系统控制、硬件层驱动和多任务等功能，在本系统中 Li nux 内核中需加载的主要驱动包括：Wi Fi 驱动、LCD 驱动、触摸屏驱动、Flash 驱动、USB 驱动、键盘和鼠标驱动等；根文件系统是 Li nux 内核启动后挂载的第一个文件系统，它限定了文件在 NAND FLASH 上的存储和组织方式，本系统采用 Cramfs 格式的文件系统；本系统所需的应用程序包括图形界面程序、电源管理控制程序和嵌入式浏览器，用 QT 开发环境开发 Wi Fi 无线上网参数配置等人机交互界面程序，在连接好无线网络后，启动嵌入式浏览器 Konqueror，即可上网浏览网页，同时电源管理应用程序在后台运行，控制系统功耗。

第三章 Wi Fi 无线通信终端软件平台搭建

3.1 交叉编译环境的建立

嵌入式系统开发涉及到体系结构 (Architecture) 和操作系统 (Operating System, OS) 两个概念, 同一体系结构上可以运行不同的操作系统, 同一操作系统也可以运行在不同的体系结构上, 但是, 在一种体系结构和操作系统下编译开发的程序只能直接在同种体系结构和操作系统上运行^[11]。举例来说, 常说的 x86 Linux 平台是指支持 X86 架构的 Linux 操作系统运行在 Intel x86 体系结构上; x86 WinNT 平台是指支持 X86 架构的 Windows NT 操作系统运行在 Intel x86 体系结构上; ARM Linux 平台则由 ARM 体系结构和支持 ARM 架构的 Linux 操作系统组成。x86 Linux 平台上编译生成的程序不能直接运行在 ARM Linux 平台上。

通常情况下, 嵌入式系统的硬件资源是很有限的, 内存比较小, 只有 FLASH 而没有类似于硬盘这样的大容量存储设备, 如 ARM 平台的 NAND FLASH 通常只有几十兆到几百兆, 存储空间和运算能力都很小, 这种限制使开发者很难将软件开发工具 (编译器等) 安装在嵌入式设备中。另一方面, 嵌入式系统开发初期, 目标架构上的软件平台还没有建立, 上面连操作系统都还没有, 更谈不上安装运行编译器了。在这种情况下, 要进行嵌入式开发, 就要使用交叉编译工具, 使在一个平台上编译生成的程序可以在另外一个平台上运行, 如, 在 CPU 运算能力和存储空间都很强大的 86 Linux 平台上编译生成可以在 ARM Linux 平台上运行的程序。用来进行交叉编译的平台叫宿主机 (host), 相对应的运行目标程序的平台叫目标机 (target), 本系统中, host 是 x86 Linux 平台, target 是 ARM Linux 平台^[12]。

交叉编译工具可以自己编译生成, 也可以从网上获取, 如果在网上可以找到版本匹配的交叉编译工具, 直接从网上下载会比较方便, 因为在没有经验的情况下自己编译会遇到很多挫折, 延长了开发周期。

在宿主机上建立交叉编译环境, 按以下步骤进行:

(1) 获取交叉编译工具包。从网上下载交叉编译工具包 arm-linux-gcc-4.3.2.tgz。

(2) 将 arm-linux-gcc-4.3.2.tgz 复制到 Linux 下某个目录, 然后进入该目录, 执行解压命令:

```
#sudo tar zxvf arm-linux-gcc-4.3.2.tgz -C /
```

该命令执行完后会将 arm-linux-gcc 安装到 /usr/local/arm/4.3.2 目录。

(3) 把安装后的交叉编译器路径添加到系统环境变量。运行命令:

```
#sudo vim /etc/environment
```

将其内容改为:

```
PATH="/usr/local/arm/4.3.2/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"
```


保存退出。

(4) 使以上设置生效。运行命令：

```
#source /etc/environment
```

(5) 确认交叉编译环境已经建立，运行指令：

```
#arm-linux-gcc -v
```

运行结果如下，则表明交叉编译环境已经成功安装。

```
Using built-in specs.
```

```
Target: arm-none-linux-gnueabi
```

```
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure
--build=i686-pc-linux-gnu
```

```
.....
```

```
--with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-
ne-linux-gnueabi/bin
```

```
Thread model: posix
```

```
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
```

3.2 Bootloader 的移植

3.2.1 Bootloader 介绍

对于嵌入式 Linux 系统来说，系统从开机上电到 Linux 操作系统启动需要一个引导程序，通常把这个引导程序叫作 Bootloader。Bootloader 是一段小程序，这段程序完成初始化硬件设备、设置堆栈、建立系统内存映射、设置内核启动参数以及启动 Linux 内核等工作，其将系统的软件和硬件配置成一种合适的状态，为加载并启动 Linux 内核做好准备，并最终启动内核^[13]。

ARM 嵌入式系统常用的 Bootloader 有：支持大多数嵌入式处理器架构的 Uboot、Mizi 公司针对三星公司的 ARM 微处理器架构专门设计 ViVi 引导程序、WinCE 操作系统下 PB 开发环境开发的 eboot 和 Windows 下 ADS 集成开发环境开发的 Bootloader 等。其中 ViVi 和 Uboot 都在 Linux 环境下开发，启动速度一般且不易调试，而 eboot 在 PB 环境下开发，启动速度比较慢，本文采用在 Windows 下 ADS 集成开发环境上开发的 Bootloader，这种 Bootloader 不仅开发环境容易熟悉、代码修改方便，而且可以使用 jlink 工具仿真下载、单步调试，启动速度也非常快。

3.2.2 Bootloader 工作流程

本系统的 Bootloader 工作流程如图 3.1 所示：

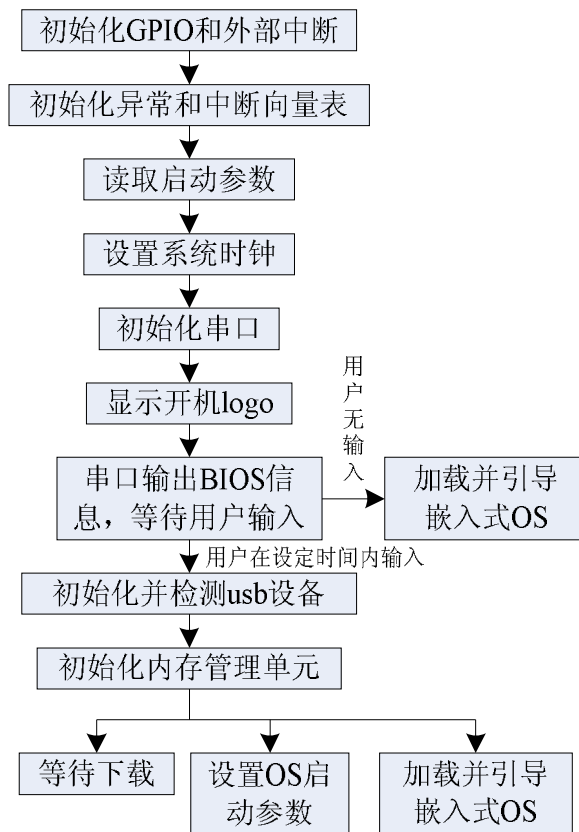


图3.1 Bootloader工作流程

Bootloader 先对硬件设备作必要的初始化，读入 Linux 操作系统启动参数，显示开机 Logo，然后通过串口打印程序运行输出，并等待三秒钟，在这三秒钟内用户按键盘上的任意键，Bootloader 进入下载模式，若用户无输入，Bootloader 加载并启动 Linux 内核。

3.2.3 Bootloader 的移植

Bootloader 的实现严重依赖于硬件，不同的微处理器体系架构的 Bootloader 不同；相同的微处理器体系架构，开发板上的其它设备不同，Bootloader 也不同。Bootloader 移植指的是参照 CPU 架构相同、其它设备也尽可能相同的嵌入式开发板的 Bootloader 代码，修改出与目标板相匹配的 Bootloader，编译并下载到目标板。YLP2440 开发板有自带的 Bootloader 代码，由于外围硬件设备有所变化，该代码不能直接用于本系统的开发，需要进行部分修改，具体的修改过程如下。

(1) 原 Bootloader 代码支持的液晶屏是夏普的 3.5 寸屏，而本系统采用的是群创的 8 寸屏，修改 lcd_type_choose.h 文件，使该 Bootloader 支持群创 8 寸屏，具体修改如下：

修改 #define lcd_type_choose CT35TF05 为：

define lcd_type_choose AT080TN52

(2) 显示自己的开机画面。原始开机画面显示的是优龙公司的 LOGO，整体画面也不是很美观，要显示自己的开机画面，需要修改 lcdinit.c 文件里的 LcdDisplay 函数。将

需要显示的图片 bmp 图片用画图打开，然后保存为 24 位格式，再用 bmp2h.exe 软件生成该图片对应的数组 c 文件，用该生成的 c 文件里数组的数据部分内容替换 uCdragon_logo.c 文件里数组的数据部分。对 lcdinit.c 文件里的 LcdDisplay 函数的修改如下：

将：

```
#elif(Lcd_type_choose == AT080TN52)
Lcd_ClearScr(0x0);
Glib_FilledRectangle( 0, 0, 800, 10, 0xf800); //Red, top
Glib_FilledRectangle(10, 10, 790, 20, 0x07e0); //Green, top
Glib_FilledRectangle(20, 20, 780, 30, 0x001f); //Blue, top
Glib_FilledRectangle( 0, 0, 10, 600, 0xf800); //Red, left
Glib_FilledRectangle(10, 10, 20, 590, 0x07e0); //Green, left
Glib_FilledRectangle(20, 20, 30, 580, 0x001f); //Blue, left
Glib_FilledRectangle(790, 0, 800, 600, 0xf800); //Red, right
Glib_FilledRectangle(780, 10, 790, 590, 0x07e0); //Green, right
Glib_FilledRectangle(770, 20, 780, 580, 0x001f); //Blue, right
Glib_FilledRectangle( 0, 590, 800, 600, 0xf800); //Red, button
Glib_FilledRectangle( 10, 580, 790, 590, 0x07e0); //Green, button
Glib_FilledRectangle( 20, 570, 780, 580, 0x001f); //Blue, button
Paint_Bmp(270, 230, 280, 140, uCdragon_logo); //picture
```

修改为：

```
#elif(Lcd_type_choose == AT080TN52)
Lcd_ClearScr(0x0);
Paint_Bmp(231, 248, 338, 104, uCdragon_logo);
```

(3) 原 Bootloader 所支持显示的开机 Logo 只能是小图片，因为如果用大图片，Bootloader 编译生成的目标文件就会有 256k 以上，而 Nand Flash 分区里设置的 boot 大小最大只有 256k，将 256k 以上的 Bootloader 目标文件下载到目标板后，256k 以外的数据就会覆盖 Nand Flash 上的其他分区的内容，系统会损失一部分数据；而且 bootloader 启动的时候，在汇编里面有一部分代码是用来拷贝程序数据到 SDRAM 里运行的，如果用大的图片生成的文件超出这个拷贝大小，最后导致拷贝出来的数据不完整，从而导致 bootloader 无法启动。如果要用大图片，解决方法是修改 Nand Flash 的分区大小，并修改启动文件里的拷贝动作对应的地址大小。

完成上述修改后，将 Bootloader 重新编译后下载到目标板，上电后目标板能正常启动，液晶屏显示正常开机画面。

3.3 Linux 内核移植

3.3.1 Linux 内核介绍

Linux 内核主要由系统调用接口、内存管理、虚拟文件系统、进程管理、网络接口、设备驱动程序、硬件架构相关代码几部分组成^[14]，如下图 3.3 所示：

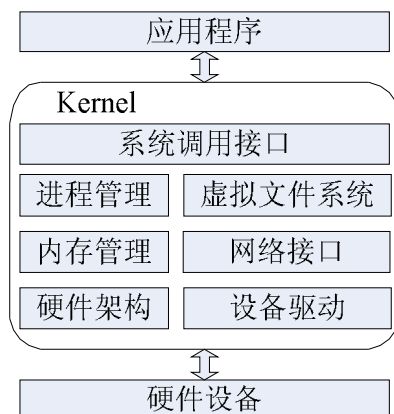


图3.3 Linux内核组成结构

系统调用接口：通常情况下用户进程不能调用内核函数，也不允许访问内核数据，它们只能调用用户空间的函数，访问用户空间的数据^[15]。系统调用接口是一组特殊的函数，用户进程可以通过组函数来获得内核提供的服务，访问内核空间的数据。

内存管理：Linux 操作系统的高效性和稳定性取决于它的内存管理机制。Linux 虚拟内存机制把正在使用的程序块存放在内存中，把其它的程序块存放在磁盘中，当需要用到其它程序块时，交换内存和磁盘上的程序块^[16]。虚拟内存机制使得所有进程都认为自己占有了全部内存，实现了内存的“无穷大”和快速访问。虚拟内存是通过地址映射、内存的分配与回收、缓存、分页、页面交换以及内存共享等机制来实现的。分页机制实现了分时多任务；缓存和交换机制实现了快速访问，使存储系统在速度上接近 cache^[17]。

虚拟文件系统：Linux 操作系统支持多种物理文件系统，不同的物理文件系统对文件的组织结构和处理方式不同，虚拟文件系统是对所有不同的物理文件系统的一种抽象，它隐藏了各种物理文件系统的具体细节，提供了一个公共接口，使得用户对所有物理文件系统的操作都一样^[18]。虚拟文件系统并不是物理的文件系统，它只是一套把各种不同物理文件系统转换为具有统一共性的虚拟文件系统转换机制，它在操作系统启动时建立，在系统关闭时撤销，而且仅仅存在于内存空间。

进程管理：进程管理是文件管理、设备管理以及存储管理的基础，由进程控制、进程调度、任务队列、定时器、中断处理、下半部分队列、进程通信等部分组成。进程调度主要负责控制进程有序的使用 CPU 资源，进程通信则主要协调各进程间数据和信息交互工作，常用的进程间通信手段有管道、信号、报文、共享内存、信号量、套接字等^[19]。

网络接口：网络接口提供对各种网络标准的实现和网络硬件设备的支持。网络接口的内核代码主要分为网络设备接口、网络接口核心、网络协议族和网络接口 socket 四个

部分。其中网络设备接口负责通过物理设备接收和发送数据；网络接口核心为各种网络协议提供统一的收发接口，并负责把来自下层的数据包配送给合适的协议；网络协议族部分实现了各种具体协议；网络接口 Socket 层为用户层的网络服务提供编程接口。

设备驱动程序：应用软件可以通过设备驱动程序安全高效的访问硬件，驱动程序直接操控硬件设备，如收发通讯数据、读写存储介质、操作输出设备等。作为一个中间层软件，设备驱动程序隐藏了底层细节，是上层对硬件的操作简单化。

硬件架构相关代码：Linux 支持不同的硬件体系架构，在 Linux 内核代码的 arch 文件夹下，不同的目录对应不同的硬件架构代码，目前 Linux 支持的硬件架构主要有 ARM 系列、Mips、PowerPC、SPARC 和 UltraSPARC、Acorn 的 Archimedes、RiscPC 系列、康柏的 Alpha、惠普的 PA-RISC、IA64、IBM 的 S/390 和 AS/400、英特尔 80386 及之后的兼容处理器、托罗拉 68020 及以上处理器、Hitachi SuperH 的 SEGA Dreamcast、索尼的 PlayStation 2、微软的 Xbox 等。

3.3.2 Linux 内核移植

Linux 内核移植指的是针对目标板的核心处理器架构和外围设备，对 Linux 内核源代码进行配置和裁剪，制定出适合自己的目标板的 Linux 内核。下面将介绍 Linux 2.6.26 内核到 YLP2440 开发板的移植过程，这里移植的是使 YLP2440 开发板可以正常启动的最小内核，许多外围设备驱动在此还未添加，这些驱动的移植将在后续内容详细讨论。

(1) 获取 Linux 源码

从 <http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.tar.bz2> 下载 Linux 内核源代码，并解压到工作目录下：

```
#tar jxvf linux-2.6.26.tar.bz2
#cd linux-2.6.26
```

此目录下是 Linux 源码的所有内容，如无特殊交代，下文的指令都在这个目录下运行。

(2) 修改配置文件

① 修改最上层的 Makefile

修改目标架构和编译器，此修改一定要在配置内核之前进行，否则在下面进行内核配置的时候，系统只显示 Makefile 里默认的硬件架构所对应的配置信息。运行指令：

```
#vim Makefile
修改 ARCH ?= 为 ARCH ?= arm
修改 CROSS_COMPILE ?= 为 CROSS_COMPILE?= /usr/local/arm/4.3.2/bin/arm-linux-
将 CC = $(CROSS_COMPILE)gcc
改为：CC = $(CROSS_COMPILE)gcc -march=armv4t
```

因为 arm-linux-gcc 4.3.2 默认编译的是 armv5t，而 s3c2440 支持的是 armv4t，所以这里必须明确指出按 armv4t 编译，否则，按此配置移植后的内核很有可能无法正常加

载根文件系统。

② 修改 fs/Kconfig

为了使内核支持 devfs，并支持在/sbin/init 运行之前能自动挂载/dev 为 devfs 文件系统，需修改 fs/Kconfig 文件。

```
#vim fs/Kconfig
```

在 menu "Pseudo filesystems"下面添加如下代码：

```
config DEVFS_FS
```

```
bool "/dev file system support (OBSOLETE)"
```

```
default y
```

```
config DEVFS_MOUNT
```

```
bool "Automatically mount at boot"
```

```
default y
```

```
depends on DEVFS_FS
```

(3) 修改源码

① 修改 NandFlash 分区信息

NandFlash 的分区信息必须跟 Bootloader 里的一致，Linux 内核才能正确的对 NandFlash 进行操作。YLP2440 的 Bootloader 里的 NandFlash 的分区信息为（在 src/nand.c 里）：

```
static struct Partition NandPart[] = {
    {0,          0x00030000, "boot"},      //256K
    {0x00030000, 0x001d0000, "kernel"},
    {0x00200000, 0x01e00000, "rootfs"},    //30M
    {0x02000000, 0x02000000, "ext-fs1"},  //32M
    {0,          0,          , 0}
};
```

所以应该将内核源码里的 NandFlash 分区信息修改如下：

在 arch/arm/plat-s3c24xx/common-smdk.c 文件中修改 smdk_default_nand_part 为：

```
static struct mtd_partition smdk_default_nand_part[] = {
    [0] = {
        .name    = "Boot",
        .size    = SZ_128K + SZ_64K,
        .offset  = 0,
    },
    [1] = {
        .name    = "Kernel",
```

```

        .offset = SZ_16K*12,
        .size   = SZ_16K*116,
    },
    [2] = {
        .name    = "Rootfs",
        .offset  = SZ_2M,
        .size    = SZ_2M*15,
    },
    [3] = {
        .name    = "User",
        .offset  = SZ_32M,
        .size    = SZ_32M,
    }
};

```

另外这个文件还要将 `smdk_nand_info` 修改为:

```

static struct s3c2410_platform_nand smdk_nand_info = {
    .tacls          = 0,           //default is 20
    .twrph0         = 30,         //default is 60
    .twrph1         = 0,         //default is 20 changed by yangdk
    .nr_sets        = ARRAY_SIZE(smdk_nand_sets),
    .sets           = smdk_nand_sets,
};

```

② 修改时钟

在 `arch/arm/mach-s3c2440/mach-smdk2440.c` 中 `smdk2440_map_io` 修改为:

```

static void __init smdk2440_map_io(void)
{
    s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
    s3c24xx_init_clocks(12000000); //default is 16934400
    s3c24xx_init_ucts(smdk2440_uartcfgs, ARRAY_SIZE(smdk2440_uartcfgs));
}

```

③ 修改 nand Flash 的校验方式, 去掉 ECC 校验。

在内核中明确说明不建议去掉 ECC 校验, 因为这样做会忽略了对 NAND FLASH 坏块的检测。但实验证明, 如果不去掉, 在编译 Linux 内核时会报错。

```
#vim drivers/mtd/nand/s3c2410.c
```

将 `chip->ecc.mode = NAND_ECC_SOFT;`

改为 `chip->ecc.mode = NAND_ECC_NONE;`

(4) 配置及编译内核

先把默认配置文件拷贝过来：

```
#cp arch/arm/configs/s3c2410_defconfig .config
```

```
#make menuconfig
```

如下所述进行配置：

[*] Enable loadable module support --->

 [*] Module unloading

 [*] Automatic kernel module loading

选择这两个，剩下的可以去掉

System Type

 [*] S3C2410 DMA support

S3C UART to use for low-level messages

这里应该选串口 1，应为我们是用板子上的 uart1 来传出消息的。

S3C2410 Machines

 [*] SMDK2410/A9M2410

S3C2440 Machines

 [*] SMDK2440

 [*] SMDK2440 with S3C2440 CPU module

 [*] Support ARM920T processor

 [*] Support Thumb user binaries

System Type 中只选这些，其他的都不选。

Kernel Features --->

 [*] Use the ARM EABI to compile the kernel

 [*] Allow old ABI binaries to run with this kernel (EXPERIMENTAL) (NEW)

用 arm-linux-gcc 4.3.2 编译时，最好选上 EABI，否则，很有可能导致根文件系统无法正常启动，报错：Kernel panic - not syncing: Attempted to kill init!。因为 arm-linux-gcc 4.3.2 工具链是支持 EABI 的，所以内核编译时也要选上，否则用这个编译器编出来的用户程序无法正常运行，最典型的错误是 busybox 无法运行，也会报错：Kernel panic - not syncing: Attempted to kill init!。

Boot options

将启动参数设置为：

```
noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC1,115200
```

Userspace binary formats

< > Kernel support for a.out and ECOFF binaries

Power management options

 [] Power Management support

Networking options

Networking options --->

< > IP: IPsec transport mode

< > IP: IPsec tunnel mode

< > IP: IPsec BEET mode

<*> Bluetooth subsystem support --->

<*> L2CAP protocol support

<*> SCO links support

<*> RFCOMM protocol support

[*] RFCOMM TTY support

<*> BNEP protocol support

[*] Multicast filter support

[*] Protocol filter support

<*> HIDP protocol support

Bluetooth device drivers --->

<*> HCI UART driver

[*] UART (H4) protocol support

[*] BCSP protocol support

[*] HCI LL protocol support

Device Drivers

<*> Memory Technology Device (MTD) support --->

< > RedBoot partition table parsing

[] Command line partition table parsing

RAM/ROM/Flash chip drivers --->

< > Detect non-CFI AMD/JEDEC-compatible flash chips

< > Support for AMD/Fujitsu flash chips

< > Parallel port support --->

[*] Block devices (NEW) --->

<*> Low Performance USB Block driver

< > ATA over Ethernet support

[*] Network device support --->

[] Ethernet (1000 Mbit) --->

[] Ethernet (10000 Mbit) --->

< > Real Time Clock --->

如果不去掉，启动内核是会打印出一条错误信息。

File systems --->

```

[*] Ext2 extended attributes
[*] Ext2 POSIX Access Control Lists
DOS/FAT/NT Filesystems --->
    <*> NTFS file system support
    [*] NTFS write support
Pseudo filesystems --->
[*] Virtual memory file system support (former shm fs)
Partition Types --->
    [ ] BSD disklabel (FreeBSD partition tables) support
    [*] Minix subpartition support
    [ ] Solaris (x86) partition table support
    [*] Windows Logical Disk Manager (Dynamic Disk) support
    [*] Windows LDM extra logging
Library routines --->
    <*> CRC32c (Castagnoli, et al) Cyclic Redundancy-Check

```

除上述配置外，其他的都用默认配置。到此为止，一个最基本的 s3c2440 的 Linux 内核已经配置完毕。

(5) 编译内核

运行指令：#make zImage

内核编译成功后会在 arch/arm/boot 目录下生成 zImage 文件，将该文件下载到目标板，终端上的输出信息表明 Linux 内核移植成功。

3.4 根文件系统的制作

3.4.1 根文件系统介绍

在 Linux 操作系统中，所有的设备都以文件的形式存在，系统提供各种设备所对应的文件与内核的交互接口，并将这些文件进行分类存放和组织，就会形成一定的目录结构，这种目录结构就是文件系统^[20]。根文件系统是 Linux 内核启动时最先挂载的文件系统，包括 Linux 系统启动所必须的目录和配置文件，如配置环境变量的/etc/profile 文件，挂载分区的/etc/fstab 文件，包含应用程序的/bin 目录等。

Linux 操作系统支持多种不同的文件系统，各文件系统所对应的存储设备的硬件特性和系统需求都不相同，Linux 通过虚拟文件系统对这些不同的文件系统进行统一管理。嵌入式 Linux 系统常用的存储设备有 RAM (SDRAM 等) 和 ROM (FLASH 等)，常见的基于这两种存储设备的文件系统有：jffs2、romfs、yaffs、cramfs、ramfs、ramdisk 等。虚拟文件系统、各物理文件系统、存储技术设备 (MTD, Memory Technology Device) 和硬件设备间的关系如图 3.4 所示：

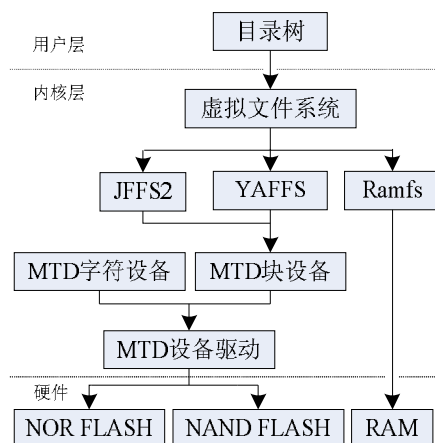


图3.4 文件系统存储结构图

基于 FLASH 存储设备的文件系统有 jffs2、yaffs、cramfs 和 romfs 等，在向 Flash 中的存储区域写入时，先要对该存储区域进行擦除，擦除操作通常以块为单位进行。jffs2 文件系统的特点是可读写、基于 ROM 型闪存、支持数据压缩、提供崩溃安全保护、支持写平衡等，不足是文件系统接近满时运行速度会变慢很多；yaffs 文件系统专为嵌入式系统使用 NAND FLASH 而设计，内存占用少，挂载时间短，运行速度快；cramfs 是一种压缩的只读文件系统运行时解压缩，并从 FLASH 里拷到 RAM 里运行，执行速度快，运行效率高，不容易被破坏，安全可靠；romfs 文件系统按顺序存放数据，很紧凑，是最简单的只读文件系统，很节约 RAM 空间。

基于 RAM 存储设备的文件系统有 ramdisk 和 ramfs 等，需要将一部分内存分区出来存放数据。ramdisk 文件系统是将经常被用到而又不用更改的文件存放的内存的分区中，可以提高系统性能；ramfs 是基于内存的，不能格式化，所有文件都存放在内存中，其大小可随所含文件的大小而调整。

3.4.2 根文件系统的制作

本系统所采用的是 cramfs 格式的根文件系统，制作根文件系统需要用到 BusyBox 工具包，BusyBox 的版本选择很重要，如果所选择的 BusyBox 版本跟内核版本或交叉编译器版本不配套，经常会出现错误。开始制作根文件系统时用的是 busybox1.13.3，结果报如下错误：

.....

VFS: Mounted root (cramfs filesystem) readonly.

Freeing init memory: 132K

Kernel panic - not syncing: Attempted to kill init!

反复试验未能解决，最后在优龙工程师的帮助下，换了 busybox1.13.0 的版本来试，错误消失了。而且需要注意的是，在执行下面的所有操作候，都使用 root 用户权限，否则，内核挂载根文件系统是会报错：mkdir: cannot create directory '/var/lock': Read-only file system。下面介绍 cramfs 根文件系统的制作步骤：

(1) 获取 BusyBox 源码

下载 BusyBox 源代码: <http://busybox.net/downloads/busybox-1.13.0.tar.bz2>

```
#tar jxvf busybox-1.13.0.tar.bz2
```

```
#cd busybox-1.13.0
```

此文件夹下的即为 BusyBox 的所有源代码。

(2) 修改 Makefile

此处操作与上面内核 Makefile 的修改类似, 主要是修改架构类型和编译器。这一步一定要在配置 BusyBox 之前进行, 否则会按照 Makefile 里默认的硬件架构输出配置信息。

```
#vim Makefile
```

修改 ARCH ?= 为 ARCH ?= arm

修改 CROSS_COMPILE?=为 CROSS_COMPILE ?= /usr/local/arm/4.3.2/bin/arm-linux-

将 CC=\$(CROSS_COMPILE)gcc

改为: CC=\$(CROSS_COMPILE)gcc -march=armv4t

(3) 配置及编译 BusyBox

配置 BusyBox 很重要, 如果配置得不对, 很可能导致文件系统运行不起来。

```
#make menuconfig
```

Busybox Settings ---->

Build Options ---->

☐ Build BusyBox as a static binary (no shared libs)

☒ Build shared libbusybox

☐ Produce a binary for each applet, linked against libbusybox

☐ Produce additional busybox binary linked against libbusybox

BusyBox 采用动态链接。如果采用静态编译的方式, 不需要 lib 的支持, 这种办法比较简单, 但是不能运行自己的程序, 所以采用动态链接方式, 动态链接方式还需要手动把运行时所需的库文件复制到/lib 中。

Busybox Library Tuning ---->

☒ vi-style line editing commands

☒ Fancy shell prompts

为了设置 PS1(终端提示符), 使/u,/h,/w 等特殊字符生效, 必须选上

Linux Module Utilities ---->

☐ Simplified modutils

☒ insmod

☒ rmmod

☒ lsmod

☒ modprobe

其他的都用默认配置。配置完成后, 运行如下指令进行编译:

```
#make install
```

编译完后会自动安装到_install 文件夹下。

(4) 修改_install/bin/busybox 的属性为 4755

```
chmod 4755 ./_install/bin/busybox
```

必须要修改属性，否则在 BusyBox 中很多命令的使用会受限制。

(5) 创建文件系统相关的文件夹

```
#mkdir rootfs
```

```
#mkdir bin dev etc home lib mnt opt proc root sbin sys tmp usr var
```

(6) 复制 BusyBox 里生成的相关文件到 rootfs

因为有很多软链接，所以建议先打包在直接解压。

```
#cd _install
```

```
#tar zcvf lib.tar.gz bin sbin usr
```

```
#cp lib.tar.gz ../rootfs
```

```
#cd ../rootfs
```

```
#tar zxvf lib.tar.gz
```

这样 busybox 生成的指令文件就拷贝到 rootfs 里了。

(7) 复制库文件到 rootfs/lib

因为 kernel 是用的 arm-linux-gcc 4.3.2 编译的，它支持 EABI，所以编译 BusyBox 也必须用支持 EABI 的编译器，对于 arm-linux-gcc-4.3.2 工具链，其 lib 目录下包含三个不同用途的 glibc 库（不知为什么要这样划分，而以前的非 EABI 工具链只有一个），对于 Samsung S3C2440A，必须使用 lib/armv4t 下的 glibc 库。否则就会无法启动文件系统，并报错：Kernel panic - not syncing: Attempted to kill init!

所应该拷贝的库文件在/usr/local/arm/4.3.2/arm-none-linux-gnueabi/lib/armv4t/lib 文件里，因为有很多用不上，只需要复制常用的即可。也可以运行 arm-linux-readelf -d busybox 看看运行 busybox 需要哪些库，就拷哪些库。

(8) 在 dev 下建立节点文件

```
sudo mknod -m 600 console c 5 1
```

```
sudo mknod -m 666 null c 1 3
```

(9) 编写配置文件 rootfs/linuxrc

Linux 内核启动完后，首先执行根目录下的 linuxrc

```
#!/bin/sh
```

```
echo "beginning linuxrc....."
```

```
#echo "mount /etc as ramfs"
```

```
#/bin/mount -n -t ramfs ramfs /etc
```

```
#/bin/cp -a /mnt/etc/* /etc
```

```
#/sbin/insmod /usr/sd_mod.ko
```

```

#/sbin/insmod /usr/usb-storage.ko
/bin/mount -f -t cramfs -o remount,ro /dev/bon/2 /
#/bin/mount -f -t ramfs ramfs /etc
/bin/mount -t ramfs ramfs /var
/bin/mkdir -p /var/tmp
/bin/mkdir -p /var/run
/bin/mkdir -p /var/log
/bin/mkdir -p /var/lock
/bin/mkdir -p /var/empty
/bin/mount -t usbdevfs none /proc/bus/usb
exec /sbin/init

```

(10) 编写 rootfs/etc/fstab 文件，它用于指定要挂载的文件系统，在 rcS 中调用。

# device	mount-point	type	options	dump	fsck order
proc	/proc	proc	defaults	0	0
tmpfs	/tmp	tmpfs	defaults	0	0
sysfs	/sys	sysfs	defaults	0	0
tmpfs	/dev	tmpfs	defaults	0	0
var	/dev	tmpfs	defaults	0	0

(11)编写 rootfs/etc/inittab 文件，它用于控制 init 进程

```

# /etc/inittab
::sysinit:/etc/init.d/rcS
console::askfirst:/bin/sh
::once:/usr/sbin/telnetd -l /bin/login
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r

```

(12) 编写 rootfs/etc/init.d/rcS 文件，它在/etc/inittab 脚本中被调用。

```

#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel
mount -a
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug

```

```
mdev -s
```

```
ifconfig lo 127.0.0.1
```

(13) 编写 rootfs/etc/profile 文件，用户获得一个 shell 后，sh 就会根据这个文件配置用户的登陆环境，这个文件初始化各种环境变量。

```
USER="`id -un`"
```

```
LOGNAME=$USER
```

```
HOSTNAME=`cat /etc/host`
```

```
PATH=$PATH
```

```
PS1="[\u@$HOSTNAME \w]\$"
```

```
export USER LOGNAME HOSTNAME PS1 PATH
```

(14) 编写 rootfs/etc/host，用于存放主机名：

```
Hostname: s3c2440
```

(15) 编写 rootfs/etc/passwd 文件：

```
root:x:0:0:root:/root:/bin/bash
```

(16) 编写 rootfs/etc/group 文件：

```
root:x:0:
```

(17) 生成 cramfs 文件系统

```
#mkcramfs rootfs rootfs.cramfs
```

第四章 Linux 设备驱动实现与移植

4.1 Linux 设备驱动程序

系统调用是操作系统内核和应用程序之间的接口，而设备驱动程序则是操作系统内核和硬件设备之间的接口。Linux 操作系统把硬件设备抽象为一个设备文件，应用程序可以像操作普通文件一样对硬件设备进行操作，Linux 应用程序独立于底层硬件运行，用户无需关心硬件问题，只需要通过一组标准化的接口调用来完成相关操作，把这些调用映射到设备特定的操作上，是设备驱动程序的主要任务^[21]。Linux 操作系统将设备分成三种基本类型：字符设备、块设备和网络设备。本系统所涉及到的主要驱动有：Wi-Fi 驱动、LCD 驱动、触摸屏驱动、Flash 驱动、USB 驱动、键盘和鼠标驱动等，由于 Flash 驱动、USB 驱动、键盘和鼠标驱动应用得比较成熟，移植比较简单，本文只详细介绍 Wi-Fi 驱动、LCD 和触摸屏驱动的实现和移植，其中 Wi-Fi 是网络设备，LCD 和触摸屏都是字符设备。

4.2 Wi-Fi 驱动的实现和移植

连接 ARM 与 Wi-Fi 模块常用的硬件接口有 SPI、SDIO 和 USB，其中 SPI 是大多数微处理器都集成的一种接口，具有硬件连接方便，软件设计简单，节省系统资源等特点。本系统采用的 Wi-Fi 模块是当今广泛使用的 Marvell 88w8686，该模块支持 SPI 和 SDIO 连接，本系统采用 SPI 接口来连接 S3C2440 处理器。

4.2.1 Wi-Fi 与 S3C2440 的硬件连接

88w8686 是 Marvell 推出的一款面向移动电话、PDA、数字摄像机等移动设备的高整合 Wi-Fi 芯片。在单一的芯片上集成了可以工作于 2.4 和 5GHz 的双频射频无线收发器、物质层、媒介接入控制器和一个 ARM 处理器。使用了 DSSS、OFDM、DBPSK、DQPSK、CCK 和 QAM 等技术，实现了无线局域网通信、电源管理和加密等功能，支持视频、语音和多媒体应用。

88w8686 向用户提供了 SDIO 和 SPI 数据传输接口。该模块会将 SDIO 或 SPI 接口传过来的用户数据封装成数据帧，通过 WLAN 传送给远程的客户端。

S3C2440A 通过 SPI0 接口与 Marvell 88w8686 的 SPI 接口连接实现数据的收发，连接如图 4.1 所示。其中，S3C2440A 是数据的发送源，所以将其配置为主设备，88w8686 则配置为从设备。SPI 采用全双工通信模式，主设备的 MISO、MOSI 和 CLK 引脚分别与从设备的 SDO、SDI 和 CLK 引脚相连接；88w8686 的 SPI_SINTn 是低电平中断输出引脚^[22]，与 S3C2440 的 EINT1 引脚相连，用于在通信过程中检测 Wi-Fi 的状态；S3C2440A 的 nSS0 是片选引脚，低电平有效，用于激活从设备。

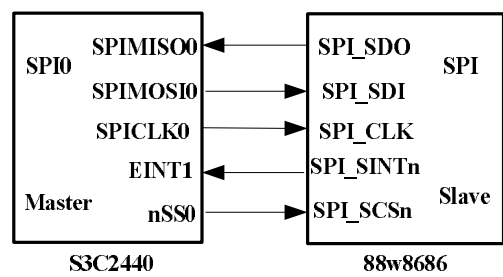


图4.1 SPI连接图

4.2.2 Wi Fi 驱动体系结构

Marvell 88w8686 Wi Fi 设备在 Linux 下的软件结构层次如图 4.2 所示。Wi Fi 模块正常工作所需的 Host 驱动包括 WLAN 和 SPI 接口驱动两部分。WLAN 驱动在整个数据收发过程中充当着数据中转的角色，接收上层用户应用程序的数据流，通过 SPI 口转发到 Wi Fi 硬件，或者，响应 Wi Fi 硬件中断，从硬件的缓冲区读取数据流，通过驱动程序注册的接口函数，发送到上层应用程序。

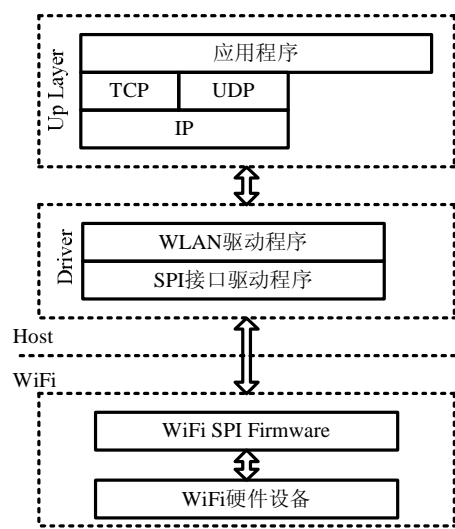


图4.2 Wi Fi 软件结构层次图

Firmware 是在 Wi Fi 设备硬件中执行的一段程序，系统上电后由 WLAN 驱动将其下载到 Wi Fi 模块中，实现 Wi Fi 硬件接口控制、数据缓冲、802.11 与 802.3 帧类型转换、802.11 MAC 层管理、WLAN MAC 中断管理、硬件控制等功能。发送数据时，Host 驱动程序将从上层接收到的标准 802.3 帧发送给 Firmware，Firmware 将收到的数据帧转换成 802.11 帧，再通过无线连接传输出去；接收数据时，Firmware 将接收到的所有 802.11 帧转换成 802.3 帧后，通过 SPI 口发送给 WLAN 驱动。由此可以看出，Wi Fi 无线网卡设备在 Linux 中是当作普通的以太网设备对待的，Wi Fi 驱动程序中无需实现 802.11 帧与 802.3 帧之间的类型转换。

4.2.3 WLAN 驱动程序

4.2.3.1 WLAN 设备初始化

WLAN 模块初始化函数 `wlan_init_module()` 的主要工作是注册回调函数，将完成设备添加和删除功能的回调函数分别注册为 `wlan_add_card()` 和 `wlan_remove_card()`。`wlan_add_card()` 中实现了 WLAN 设备初始化的所有操作，依次完成以下几部分工作^[23]：

① 通过 `sbi_probe_card()` 检测设备，探测到无线网卡；然后，和一般 Linux 下网络设备驱动程序一样，调用 `alloc_etherdev()` 来分配代表网络设备的 `struct net_device` 结构，并在该结构中初始化内核对网卡操作的接口函数，主要接口初始化如下

```
dev->open = wlan_open;
dev->hard_start_xmit = wlan_hard_start_xmit;
dev->stop = wlan_close;
dev->do_ioctl = wlan_do_ioctl;
dev->set_mac_address = wlan_set_mac_address;
```

② 调用 `wlan_create_thread()` 创建主线程 `wlan_service_main_thread`，该线程处理 wlan 驱动的主要工作，包括处理 firmware 产生的事件，接收从 firmware 发送过来的数据，发送从内核传递过来的数据。

③ 通过 `wlan_create_thread()` 创建 `wlan_reassociation_thread` 线程，该线程在连接自动断开的时候，负责重新连接 AP。

④ 通过 `sbi_register_dev()` 注册 WLAN 设备，填写代表网卡设备的私有结构体中网卡硬件设备相关信息和 IRQ 请求。

⑤ 调用 `wlan_init_fw()` 初始化 firmware 并下载 firmware 到 Wi-Fi。

⑥ 调用 `register_netdev()` 注册网络设备，用来供上层访问。该函数返回设备的主设备号，之后对网络设备的所有调用都通过这个设备号来实现。

4.2.3.2 WLAN 数据包发送

WLAN 驱动程序的数据发送函数是 `wlan_hard_start_xmit()`，这一点已经在 `wlan_add_card()` 中设置。WLAN 驱动程序中还引入 WMM (Wi-Fi Multimedia) 机制来规范数据流量优先权，实现无线网络流量的优先级管理。跟一般的网络设备驱动程序一样，`sk_buff` 数据结构用来存放从上层应用程序接收到的数据。发送函数调用 `wlan_tx_packet()` 把接收到的 `sk_buff` 结构数据添加到 WMM 队列，然后唤醒主线程来处理数据的发送。

主线程首先为数据传输做了一些准备工作，包括初始化 WMM 状态信息和数据队列、检查当前设备是否可用、唤醒网络设备等，然后依照当前设备的可用性做不同的处理：如果当前设备不可用，即设备忙，则等待，等到设备可用再发送；如果当前设备可用，就通过 `wmm_process_tx()` 传输 WMM 等待队列中优先级最高的数据包。`wmm_process_tx()` 先进行模式判断，若处于 WMM PS (Power Save) 模式，则不发送数据，若处于 Active 模式，则调用 `wlan_process_tx()` 检查数据发送条件，准备就绪后将数据传送给 `SendSinglePacket()` 进行单个数据包的发送。`SendSinglePacket()` 先对数据包做一些

检查,然后把数据包复制到 TxPD 类型区域中,最后调用 `sbi_host_to_card()` 来将数据包发送到 Wi Fi 模块硬件,由固化在 Wi Fi 模块硬件中的 Firmware 把数据发送出去。

数据成功发送到 Wi Fi 模块硬件后将产生中断,该中断对应的中断处理函数所完成的主要工作包括清除相应状态位、释放 `sk_buff` 结构、通知系统可以再次发送。如果数据发送不成功, `dev->busy` 置位,驱动程序会不断尝试重传,若重传超时(如设备严重堵塞时),驱动程序会将该数据包丢弃,同时释放 `sk_buff` 结构。

4.2.3.3 WLAN 数据包接收

WLAN 驱动程序并没有提供专门的数据接收函数,网络设备收到数据后会产生一个中断,中断处理程序通过 `sbi_get_int_status()` 读取当前状态寄存器,判断出数据接收状态,申请一块大小合适的 `sk_buff` 缓冲区,再通过 `sbi_card_to_host()` 把 Wi Fi 模块硬件中的网络数据包读取到该 `sk_buff` 缓冲区,并将该 `sk_buff` 添加到接收数据队列。从硬件中成功把数据读进来后, `wlan_send_rxskbQ()` 调用 `ProcessRxedPacket()` 处理接收到的数据包,将其转发到协议层。

4.2.4 SPI 驱动程序

Host 和 Wi Fi 模块之间的数据传输最终是通过 SPI 实现的, SPI 属于字符型设备,应用程序通过字符设备文件来对 SPI 硬件进行操作^[24]。 SPI 设备驱动程序的结构与大多数字符型设备驱动相似,主要包含设备初始化和注销、打开关闭和数据读写等操作。

SPI 设备初始化通过 `gsplhost_module_init()` 实现,该函数初始化 S3C2440 中与 SPI 相关的寄存器,调用 `gsplhost_init_hw()` 配置 SPI0MISO、SPI0MOSI、SPI0CLK、SPI0CS 和 EINT1 引脚所对应的寄存器,并通过 `register_chrdev` 将 SPI 注册为字符设备。注销函数则执行相反的操作。

SPI 设备打开和关闭通过 `gsplhost_open()` 和 `gsplhost_release()` 实现,数据读写通过 `gspl_read_data()` 和 `gspl_write_data()` 实现。数据读写最终分别会调用 `gspl_read_data_direct()` 和 `gspl_write_data_direct()` 按照 SPI 接口时序直接对寄存器进行读写来实现数据的收发,而且,这两个函数还会在 WLAN 数据收发中断处理函数中被调用。

4.2.5 Wi Fi 驱动的移植

嵌入式 Linux 设备驱动有静态加载和动态加载两种加载方式,本 Wi Fi 设备驱动采用动态加载方式。该 Wi Fi 驱动程序编译后生成 `gspl.ko` 和 `gspl8xxx.ko` 两个文件,加载过程如下:

加载 SPI 驱动,运行指令: `# insmod gspl.ko`

加载 WLAN 驱动,运行指令:

```
# insmod gspl8686.ko  helper_name=./helper_gspl.bin fw_name=./gspl8686.bin
mfdmode=1
```

成功加载 Wi Fi 驱动后, 对该驱动进行测试, 具体操作如下:

- ① 查看是否检测到设备, 运行 `iwconfig` 指令, 结果显示可以检测到。
- ② 配置设备接口, 运行指令: `# ifconfig eth1 up`, 运行结果显示设置成功。
- ③ 扫描可用 AP, 并连接到其中一个 AP。运行指令及运行结果如下图 4. 3:

```
#iwlist eth1 scan
Cell 04 - Address: 00:18:84:81:46:E2 ESSID:"MyPlace"
        Mode: Managed Frequency: 2.427 GHz (Channel 4)
        Quality=100/100 Signal level=-39 dBm Noise level=-96dBm
        Encryption key: off
        Bit Rates: 1 Mb/s; 2 Mb/s; 5.5 Mb/s; 6 Mb/s; 9 Mb/s
                  11 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                  48 Mb/s; 54 Mb/s
#iwconfig eth1 txpower auto essid MyPlace channel 4
eth1 IEEE 802.11b/g ESSID:"MyPlace"
        Mode: Managed Frequency:2.427GHz
        Access Point: 00:18:84:81:46:E2 Bit Rate:0 kb/s Tx-Power=13dBm
        Retry short limit:8 RTS thr=2347 B Fragment thr=2346 B
        Encryption key: off Power Management: off
        Link Quality=97/100 Signal level=-43 dBm Noise level=-94dBm
        Rx invalid nwid:0 Rx invalid crypt: 3109 Rx invalid frag: 0
        Tx excessive retries:13 Invalid misc:3315 Missed beacon:0
```

图4. 3 AP扫描和连接指令运行结果

- ④ 获取 IP 地址, 运行指令: `#udhcpc -l eth1`, 获取 IP 为 192. 168. 10. 216。
- ⑤ 运行指令: `#ping -c 192. 168`, 运行结果表明可以 ping 通。

测试结果表明, 该 Wi Fi 驱动能够成功加载固件、可以扫描并连接到可用 AP、可以和网内计算机互通信息, 实现了预期目标, 确实可用。

4. 3 LCD 驱动的实现和移植

4. 3. 1 LCD 显示原理

LCD 正常的显示文字或图像, 需要相应的 LCD 控制器。S3C2440 的 MCU 内部集成了 LCD 控制器, 其作用是将定位在系统存储器 (SDRAM) 中的显示缓冲区中的 LCD 图像数据传送到外部 LCD 驱动器, 并产生控制信号来控制 TFT 屏。嵌入式设备驱动 LCD 不同于 PC 机, 没有显卡, 通用的方式是在内存区中划出一段空间作为画面缓冲区 (Frame Buffer), 其组织与所设定的 LCD 规格相对应, 形成一个虚拟的显示器^[25]。显示缓冲和系统共享内存空间, 处理器可以直接读写缓存, 这样, LCD 显示的每一个点的信息都被存放在显示缓冲区中, LCD 控制器在扫描时, 通过读取显示缓冲区数据信息, 按相应的时序输出, 实现了液晶屏的显示。

4. 3. 2 Frame Buffer 设备驱动

帧缓冲 (Frame Buffer) 设备将显示内存和显示芯片寄存器从物理内存映射到进程地址空间, 是 Linux 为图形设备提供的一个抽象接口, 它将显示设备抽象为帧缓冲区。下面是 Frame Buffer 设备驱动在 Linux 系统中的结构图。

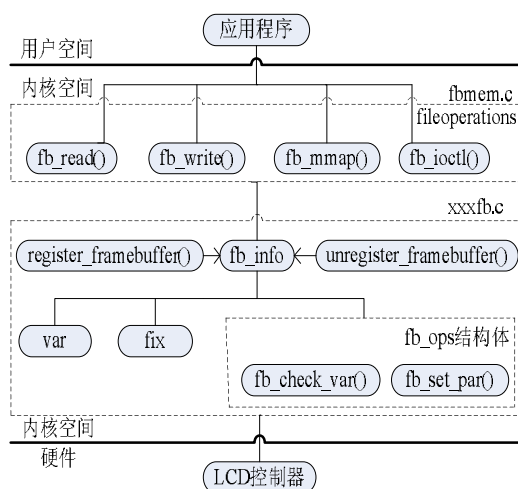


图4.4 Frame Buffer设备驱动程序结构图

从图中可以看出，Frame Buffer 设备驱动在 Linux 中可以看作为一个完整的子系统，主要由 fbmem.c 和 xxxfb.c 文件组成。fbmem.c 中实现了上层应用程序对 Frame Buffer 设备进行 read、write、ioctl 等操作的操作接口；xxxfb.c 提供了对硬件的操作接口，这些接口要根据具体的 LCD 控制器硬件进行设计实现。

Frame Buffer 设备驱动中一个重要的结构体是 fb_info，该结构体记录了帧缓冲设备的属性和操作的全部信息，包括设备的参数设置、状态设置以及对底层硬件操作的函数指针。该结构中比较重要的结构体成员有 struct fb_var_screeninfo var、struct fb_fix_screeninfo fix 和 struct fb_ops *fbops^[26]。其中 fb_var_screeninfo 结构体主要记录用户可以修改的控制器的参数，比如屏幕的分辨率和每个像素的比特数等；fb_fix_screeninfo 结构体主要记录用户不可以修改的控制器的参数，比如屏幕缓冲区的物理地址和长度等；fb_ops 结构体是指向帧缓冲设备操作的函数指针，该结构体定义了对帧缓冲设备的所有操作，这些操作接口的具体实现都在 fbmem.c 文件中。

4.3.3 LCD 设备注册为平台设备

由于 LCD 控制器已经集成在 S3C2440 里，Linux2.6.26 内核把它看作一个平台设备，并在 s3c_lcd_resource 和 s3c_device_lcd 结构体中定义了与 LCD 相关的平台设备及资源。Linux 内核还为 LCD 平台设备定义了 s3c2410fb_mach_info 结构体，该结构体主要记录 LCD 的硬件参数信息。其 s3c2410fb_display 成员结构用于记录 LCD 的屏幕尺寸、屏幕信息、可变的屏幕参数、LCD 配置寄存器等。修改这两个结构体里的参数信息，可以将不同的 LCD 注册为平台设备。

本系统使用的是群创公司的 AT080TN52 V3 液晶屏，将该 LCD 注册为平台设备的方法如下。AT080TN52 V3 液晶屏的分辨率为 800*600，接口为 24bits/pixel。

(1) 定义一个 s3c2410fb_display 结构体，记录 LCD 硬件信息。

```
static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {
    .lcdcon5=(1<<11)|(0<<10)|(1<<9)|(1<<8)|(0<<7)|(0<<5)|(1<<3)|(0<<1)|(1),
```

```

.type = S3C2410_LCDCON1_TFT, //TFT 类型
.width = 800, //屏幕宽度
.height = 600, //屏幕高度
.pixclock = 250000, //像素时钟
.xres = 800, //水平可见的有效像素
.yres = 600, //垂直可见的有效像素
.bpp = 16, //色位模式
.left_margin = 209, //HFPD, 行切换, 从同步到绘图之间的延迟
.right_margin = 45, //HBP, 行切换, 从绘图到同步之间的延迟
.hsycn_len = 20, //HSPW, 水平同步的长度
.upper_margin = 11, //VFPD, 帧切换, 从同步到绘图之间的延迟
.lower_margin = 22, //VBPD, 帧切换, 从绘图到同步之间的延迟
.vsync_len = 10, //VSPW, 垂直同步的长度
};

```

上述配置信息根据群创 AT080TN52 V3 液晶屏数据手册分析得出。`pixclock` 的值是像素时钟, `left_margin` 和其它六个参数对应于手册上时序图上的各时钟延时参数, 他们的值通过分析时序图得出。

(2) 定义一个 `s3c2410fb_mach_info` 结构体, 注册硬件信息, 初始化 GPIO。

```

static struct s3c2410fb_mach_info smdk2440_fb_info __initdata = {
    .displays = &smdk2440_lcd_cfg,
    .num_displays = 1,
    .default_display = 0,
    .gpccon = 0xaaaaaaaa,
    .gpccon_mask = 0xffffffff,
    .gpcup = 0xffffffff,
    .gpcup_mask = 0xffffffff,
    .gpdcon = 0xaaaaaaaa,
    .gpdcon_mask = 0xffffffff,
    .gpdup = 0xffffffff,
    .gpdup_mask = 0xffffffff,
    .lpcsel = 0x00,
};

```

其中 `display` 的值是将上面定义的 `smdk2440_lcd_cfg` 结构体添加到新定义的 `smdk2440_fb_info` 结构体中。`gpccon` 到 `gpdup_mask` 的是在初始化 LCD 相关的寄存器的值。因为用的不是三星的屏, 所以将 `lpcsel` 的值设置为 0。

(3) 将 LCD 硬件信息注册到平台数据中。

```
static void __init smdk2440_init(void)
{
    s3c24xx_fb_set_platdata(&smdk2440_fb_info); //注册到平台数据中
    platform_add_devices(smdk2410_devices, ARRAY_SIZE(smdk2410_devices));
    smdk_machine_init();
}
```

至此，AT080TN52 LCD 的硬件信息就成功的注册到平台设备数据中。

4.3.4 LCD 驱动的移植

Linux2.6.26 内核本来就支持 LCD 驱动，但原驱动代码只将三星 3.5 寸液晶屏的设备信息注册到平台设备，故需修改相关的文件，注册本系统所用的 LCD 到平台设备，具体移植按如下步骤进行。

(1) 重新配置内核，运行指令：

```
#make menuconfig
```

选择如下几项：

```
Device Drivers --->
```

```
Graphics support --->
```

```
<*> Support for frame buffer devices --->
```

```
[*] Enable firmware EDID
```

```
<*> S3C2410 LCD framebuffer support
```

```
<*> Virtual Frame Buffer support (ONLY FOR TESTING!)
```

```
Console display driver support --->
```

```
<*> Framebuffer Console support
```

```
[*] Select compiled-in fonts
```

```
[*] VGA 8x8 font
```

```
[*] VGA 8x16 font
```

```
[*] Bootup logo --->
```

```
[*] Standard 16-color Linux logo
```

```
[*] Standard 224-color Linux logo
```

(2) 其次是修改驱动源程序。

修改 arch/arm/mach-s3c2410/mach-smdk2410.c 文件，首先在头文件出添加：

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

然后，按照上节将 AT080TN52 V3 注册到平台设备的方法将上述三段代码添加进来。

(3) 重新编译内核并下载到开发板上，系统启动后可以在 LCD 上看到一个小企鹅图标，说明移植成功。

4.4 触摸屏驱动实现和移植

4.4.1 触摸屏硬件原理

触摸屏附着在 LCD 显示器的表面，本处所用的 LCD 表面附着的是电阻式的触摸屏，由 4 层透明的复合薄膜组成，中间两层是具有相同表面电阻的金属导电层，由透明弹性材料把它们隔开，当手指触摸屏幕时，两导电层在触摸点处接触^[27]。采用分压器原理来产生代表 X、Y 坐标的电压，可以计算出 x、y 的绝对坐标值。

S3C2440 的触摸屏接口与 ADC 接口结合在一起。触摸屏接口的模式有普通 ADC 转换模式、独立 X/Y 位置转换模式、自动 X/Y 位置转换模式、等待中断模式四种。在驱动程序中主要采用自动 X/Y 位置转换模式和等待中断模式^[28]。自动转换模式是触摸屏控制器自动转换 X、Y 的触摸位置，转换完毕后将数据分别存放在寄存器 ADCDAT0 和 ADCDAT1，并产生 INT_ADC 中断通知转换完毕；等待中断模式是等待触点信号，触摸后，触摸屏控制器产生 INT_TC 中断，通过自动 X/Y 位置转换模式来获取 X、Y 位置数据。

4.4.2 Linux 中 Input 子系统

触摸屏设备的驱动可以利用内核提供的 input 子系统来实现。输入子系统的结构如下图所示：

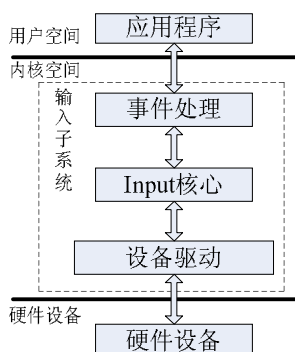


图4.5 输入子系统的结构图

在 Input 子系统中，事件处理模块用来处理来自硬件的输入事件，并向上层提供文件操作的接口；Input 核心模块的作用是负责设备的添加和删除，通知事件处理模块对事件进行处理，并向下层设备驱动提供内核接口；设备驱动的核心工作是将底层的硬件输入转化为事件形式，向 Input 核心汇^[29]。事件处理模块和 Input 核心模块在 linux 内核中已经很完整的实现，所以只需要编写 s3c2440 触摸屏设备驱动程序，提供将触摸屏采集到的数据上报给 Input 核心模块的方法，便可以在 Linux 内核中实现对 s3c2440 触摸屏的支持。

4.4.3 触摸屏设备驱动实现

实现触摸屏设备驱动所要完成的主要工作有^[30]：

- (1) 实现驱动模块的加载和卸载；

(2) 实现触摸事件的中断服务程序;

(3) 实现 ADC 转换的中断服务程序。

首先, 驱动模块的加载和卸载。加载部分主要完成如下工作: 获取并启用 ADC 所需要的时钟、映射 ADC 的 I/O 端口到内存的虚拟地址、初始化 ADC 和触摸屏控制寄存器、申请 ADC 和触摸屏中断、初始化触摸屏输入设备、将输入设备注册到输入子系统^[31]。卸载部分主要完成释放中断和虚拟地址空间、销毁时钟、将触摸屏设备从输入子系统中注销等工作。这些工作主要在 `s3c2440ts_init()` 和 `s3c2440ts_exit()` 两个函数里实现。用到一个重要的结构体 `input_dev`, 其定义一个输入设备来表示触摸屏设备, 定义语句为: `static struct input_dev *ts_dev`。向输入子系统中加载和卸载 `ts_dev` 触摸屏设备是通过调用 Input 子系统的函数接口实现的, 具体实现为:

```
input_register_device(ts_dev);
```

```
input_unregister_device(ts_dev);
```

其次, 触摸事件中断服务程序的实现。对触摸屏的操作一般包括提起、按下和拖动, 当触摸事件发生时将触发中断, 捕捉到该中断后交由中断处理函数 `stylus_updown()` 函数来处理, 该函数定义了 `updown` 变量用于记录触摸屏状态是按下还是抬起, 是抬起时释放资源, 函数返回到等待状态, 是按下时调用 `touch_timer_fire()` 函数启动 ADC 转换。其中 `touch_timer_fire()` 函数是实现把采集到的数据传递给 Input 核心模块的主要函数, 下面是该函数的流程图:

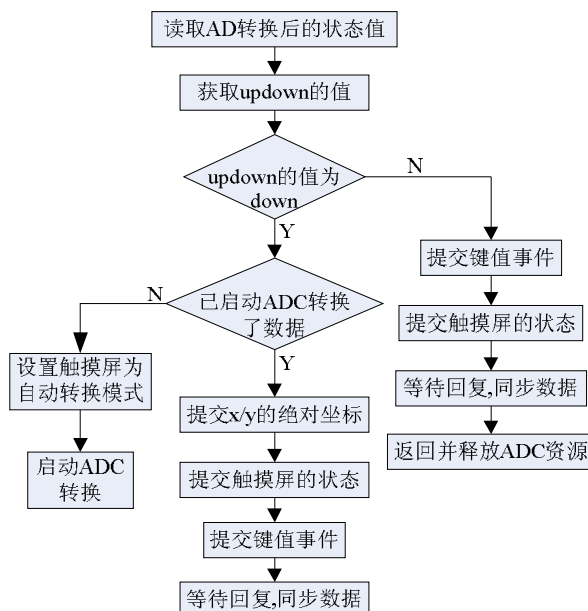


图4.6 touch_timer_fire()函数流程图

从流程图可以看出, 当缓冲区没有数据时, 设置 AD 转换的模式后, 开启 AD 转换; 当 AD 转换把缓冲区填满后, 就将采集到的数据提交给 Input 子系统。该函数中将采集到的数据提交给输入子系统的所用到的接口函数有:

提交按键事件的函数:

```
void input_report_key(struct input_dev *dev, unsigned int code, int value);
```

提交绝对坐标事件的函数：

```
void input_report_abs(struct input_dev *dev, unsigned int code, int value);
```

事件同步函数，用于等待 Input 核心的回复：

```
void input_sync(struct input_dev *dev);
```

最后，ADC 转换的中断服务程序的实现。ADC 中断服务程序在 AD 转换完成后触发执行。如果当前 AD 转换的次数小于 4，则重新启动 ADC 进行转换，4 次转换完毕后，启动 1 个时间滴答的定时器 touch_timer，该定时器的服务程序为 touch_timer_fire，如果 1 个时间滴答到来则进入定时器服务程序，将数据上报 Input 核心模块。在这个时间滴答内，ADC 转换是停止的。

4.4.4 触摸屏驱动移植

由于在 linux-2.6.26 的内核源码中，对 S3C2440 架构还没有支持触摸屏驱动代码，所以需要自己编写触摸屏驱动。这里主要说明将写好的触摸屏驱动添加进内核的方法。

(1) 将驱动代码添加进内核

由于 s3c2440 上的接的触摸屏不需要 adc 支持就可以直接驱动，所以只需要编写出触摸屏的驱动，而不需要再添加 adc 驱动。触摸屏属于字符设备，将编写的驱动程序放到字符设备驱动文件夹下：

```
cp s3c2410_ts.c linux-2.6.26/drivers/char
```

```
cp s3c2410ts.h linux-2.6.26/include/asm-arm/arch-s3c2410
```

(2) 修改字符设备驱动程序的 Makefile 和配置文件

```
cd linux-2.6.26/drivers/char
```

```
vim Makefile
```

```
添加: obj-$(CONFIG_S3C2410_TS) += s3c2410ts.o
```

```
vim Kconfig
```

添加：

```
choice
```

```
    prompt "S3C2410 touchscreen driver"
```

```
    default S3C2410_TS
```

```
    depends on ARCH_S3C2410
```

```
    config S3C2410_TS
```

```
        tristate "normal touchscreen driver add by liufanghua"
```

```
endchoice
```

(3) 将触摸屏设备信息添加到平台设备初始化表

```
vim arch/arm/mach-s3c2410/ mach-smdk2410.c
```

添加：

```
#include <asm/arch/s3c2410ts.h>
```

```
static struct s3c2410_ts_mach_info ylp2410_ts_cfg __initdata = {
    .delay = 10000,
    .presc = 49,
    .oversampling_shift = 2,
};
```

在结构指针 `smdk2410_devices[]` 中添加: `&s3c_device_ts`,

在函数 `smdk2410_map_io` 中添加: `set_s3c2410ts_info(&ylp2410_ts_cfg)`;

(4) 添加触摸屏设备资源

`vim linux-2.6.26/include/asm-arm/plat-s3c24xx/devs.h`

添加: `extern struct platform_device s3c_device_ts;`

`vim linux-2.6.26/arch/arm/plat-s3c24xx/devs.c`

添加:

```
#include <asm/arch/s3c2410ts.h>
static struct s3c2410_ts_mach_info s3c2410ts_info;
void __init set_s3c2410ts_info(struct s3c2410_ts_mach_info
    *hard_s3c2410ts_info)
{
    memcpy(&s3c2410ts_info, hard_s3c2410ts_info, sizeof(struct
        s3c2410_ts_mach_info));
}
EXPORT_SYMBOL(set_s3c2410ts_info);
struct platform_device s3c_device_ts = {
    .name      = "s3c2410-ts",
    .id        = -1,
    .dev       = {.platform_data = &s3c2410ts_info, }
};
EXPORT_SYMBOL(s3c_device_ts);
```

重新编译内核，下载到板子上去，触摸屏幕，光标可以移动，说明触摸屏驱动好了。此时可能还存在光标不准的情况，需要移植校准程序才行，校准程序的移植将在 `Qtopia` 移植部分介绍。

第五章 应用程序的开发与移植

5.1 Qtopia 平台的搭建与移植

Qt 是一个 C++ 应用程序开发框架，在所有的平台上它的 API 都是相同的，因此 Qt 应用程序开发与平台无关。通常意义上，Qt 泛指其所有版本的图形界面库，其中用于 Linux 系统的是 Qt/X11，用于嵌入式 Linux 系统的是 Qt/Embedded，简称 Qt/E^[32]。Qtopia 是基于 Qt/E 库开发出来的应用程序，可用于桌面系统开发，包含 PDA 和 Phone 两种版本，PDA 版 Qtopia 的最高版本是 Qtopia-2.2.0。本系统需要开发带有桌面系统的图形界面程序，采用的是 Qtopia-2.2.0。

5.1.1 Qtopia 开发环境的搭建

在 3.1 节介绍过，x86 和 ARM 是两种不同的硬件体系架构，在 x86 系统上编译出来的程序只能在 x86 系统上运行，要想其能运行于 ARM 系统，需要进行交叉编译。同样的道理，基于 x86 编译器编译出来的 Qtopia 开发环境开发的应用程序只能运行于 x86 系统，要想 Qtopia 开发环境能开发出运行于 ARM 系统的应用程序，需要建立由 ARM 交叉编译器编译出来的 Qtopia 开发环境。由于 Qtopia-2.2.0 的源码中的 qt 目录可编译生成 Qt/E 库和 Qt 工具，所以建立 Qtopia 后，Qt/E 库和 Qt 开发环境均被建立。下面详细介绍 Linux 下这两种 Qtopia 开发环境的建立过程。

(1) 获取源码

下载 qtopia-free-src-2.2.0.tar.gz 源码包，运行解包指令：

```
#tar zxvf qtopia-free-src-2.2.0.tar.gz
```

运行指令：

```
#cp qtopia-free-src-2.2.0 qtopia-2.2.0-x86
```

```
#mv qtopia-free-src-2.2.0 qtopia-2.2.0-arm
```

此时，生成两份源码，qtopia-2.2.0-x86 用于建立 x86 系统的 Qtopia 开发环境，qtopia-2.2.0-arm 用于建立 arm 系统的 Qtopia 开发环境。

(2) 建立 x86 系统的 Qtopia 开发环境

```
#cd qtopia-2.2.0-x86
```

运行下列指令进行配置：

```
#./configure -qte '-embedded -no-xft -qconfigpde -depths16,32 -system-jpeg -gif' -qpe '-edition pda -displaysize 800x600 -fontfamilies "elvetica fixed micro smallsmooth smoothtimes"' -qt2 '-no-opengl -no-xft' -dqt '-no-xft -thread'
```

指令运行成功后，会生成 Makefile 文件，运行编译指令：#make

运行安装指令: #make install

(3) 建立 arm 系统的 Qtopia 开发环境

为了实现触摸屏的校准,在此先要编译校准程序。下载源码 tslib-1.4.1.tar.bz2 ,
解压:

```
#tar jxvf tslib-1.4.1.tar.bz2 -C qtopia-2.2.0-arm
```

```
#cd qtopia-2.2.0-arm
```

```
# cd tslib-1.4.1
```

先运行:

```
#export CC=/usr/local/arm/4.3.2/bin/arm-linux-gcc
```

```
#export CXX=/usr/local/arm/4.3.2/bin/arm-linux-g++
```

```
#./autogen.sh
```

```
#echo "ac_cv_func_malloc_0_nonnull=yes" >arm-linux.cache
```

再开始配置:

```
#./configure - host=arm-linux - cache-file=arm-linux.cache
```

```
- prefix=$PWD/install - enable-inputapi=no ac_cv_func_malloc_0_nonnull=yes
```

生成 Makefile 后, 编译:

```
#make
```

```
#make install
```

编译安装完成后, 在当前目录的 install 目录下可以看到相应的工具、共享库和配置文件。然后退出到上级目录进行 Qtopia 的配置:

```
#cd ..
```

运行下列指令进行配置:

```
./configure -qte '-embedded -no-xft -qconfig qpe -depths 16,32  
-system-jpeg -qt-zlib -qt-libpng -gif -no-g++-exceptions -no-qvfb  
-xplatform linux-arm-g++ -tslib' -qpe 'edition pda -display-size 800x600  
-fontfamilies "elvetica fixed micro smooth smoothtimes uni font"  
-xplatform linux-arm-g++ -luuid' -qt2 '-no-opengl -no-xft' -dqt '-no-xft  
-thread'
```

指令运行成功后, 会生成 Makefile 文件, 运行编译指令: #make

运行安装指令: #make install

5.1.2 Qtopia 的移植

要使 Qtopia 桌面系统程序运行于 ARM 架构的目标板, 需要将建立 arm 系统的 Qtopia 开发环境时生成的工具和库文件放进文件系统, 并添加相应的配置, 这个过程就是 Qtopia 的移植。具体过程如下:

(1) 添加对 Qtopia 的配置

编写 shell 脚本 qtopia，并将其置于文件系统的 bin/目录下：

```
#vim qtopia
```

添加如下内容：

```
#!/bin/sh
#-----Qtopia environment
echo "set Qtopia environment....."
export set HOME=/tmp
export set QTDIR=/opt/Qtopia
export set QPDIR=/opt/Qtopia
export set KDEDIR=/opt/kde
export set QWS_SIZE=800x600
export set QWS_KEYBOARD="USB:/dev/event1"
export set QWS_MOUSE_PROTO="TPanel:/dev/event0"
export set PATH=$QPDIR/bin:$PATH
export set LD_LIBRARY_PATH=$QTDIR/lib:$QPDIR/lib
        mknod -m 666 /tmp/null c 1 3
if [ -f /tmp/pointercal ] ; then
        $QPDIR/bin/qpe > /tmp/null 2>/tmp/null
else
        ts_calibrate
        $QPDIR/bin/qpe -qws > /tmp/null 2>/tmp/null
#        $QPDIR/bin/qpe -qws
fi
```

保存退出，运行指令：

```
#chmod a+x qtopia
```

(2) 添加对触摸屏校准的配置

在上述 qtopia 脚本中的#!/bin/sh 下面添加：

```
#-----touch screen environment
echo "set touch screen environment"
export set TSLIB_TSDEVICE=/dev/event0
export set TSLIB_CONFFILE=/etc/ts.conf
export set TSLIB_PLUGINDIR=/lib/ts
export set TSLIB_CALIBFILE=/tmp/pointercal
```

(3) 添加启动 Qtopia 桌面环境的配置

在文件系统的 etc/init.d/rcS 文件中添加：

```
qtopia &
```

(4) 添加 Qtopia 文件到根文件系统

将 qtopia-2.2.0-arm/qtopia/image/opt 目录下的 Qtopia 和 kde 文件夹拷贝到文件系统的 opt 目录下:

```
#cp -rf qtopia-2.2.0-arm/qtopia/image/opt/Qtopia rootfs/opt
#cp -rf qtopia-2.2.0-arm/qtopia/image/opt/kde rootfs/opt
```

(5) 添加触摸屏校准文件到根文件系统

```
# cp qtopia-2.2.0-arm/tslib-1.4.1/install/bin/ts_calibrate rootfs/sbin
# cp qtopia-2.2.0-arm/tslib-1.4.1/install/etc/ts.conf rootfs/etc
```

并修改该配置文件:

```
#cd rootfs/etc
#vim ts.conf
```

将内容修改为:

```
module_raw input
module pthres pmin=1
module variance delta=30
module dejitter delta=100
module linear
```

修改完后保存退出。

```
# cp -rf qtopia-2.2.0-arm/tslib-1.4.1/install/lib/ts rootfs/lib
```

(6) 将 rootfs 文件转换成 cramfs 文件系统格式

```
#mkcramfs rootfs rootfs.cramfs
```

将 rootfs.cramfs 下载到目标板, 目标板上电后, 可以显示 Qtopia 桌面系统, 触摸屏校准可用, 说明 Qtopia 移植成功。

5.2 嵌入式浏览器的移植

常见的开源嵌入式浏览器有 kconqueror embedded、minimo(mini mozilla)等, 本系统所采用 kconqueror。将 kconqueror 移植到 ARM 平台上的步骤如下:

(1) 获取 kconqueror 源代码。从如下网址获取 kconqueror 源代码包并解压。

<http://developer.kde.org/~hausmann/snapshots/Attic/konqueror-embedded-snapshot-20030705.tar.gz>

```
#tar zxvf konqueror-embedded-snapshot-20030705.tar.gz
#cd ~/konqueror-embedded
```

(2) 设置编译时环境变量

```
export CC=arm-linux-gcc
export CXX=arm-linux-g++
export LDFLAGS=-ldl
```

```

export CROSS_COMPILE=1
export QPEDI R=$HOME/Qttopia-2.2.0-arm/Qttopia
export QTOPIA_DEPOT_PATH=$HOME /Qttopia-2.2.0-arm/Qttopia
export QTDI R=$HOME/Qttopia-2.2.0-arm/Qt2
export DQTDI R=$HOME /Qttopia-2.2.0-arm/dQt
export TMAKEDI R=$HOME /Qttopia-2.2.0-arm/tmake
export TMAKEPATH=$TMAKEDI R/lib/qws/linux-arm-g++
export PATH=$QPEDI R/bin:$QTDI R/bin:$DQTDI R/bin:$PATH
export
LD_LIBRARY_PATH=$QPEDI R/lib:$QTDI R/lib:$DQTDI R/lib:$LD_LIBRARY_PATH

```

(3) 运行下列指令进行配置

```

#. /configure --enable-embedded --enable-qt-embedded --enable-qpe
--disable-debug --enable-static --disable-shared --disable-mt --without-ssl
--with-qt-dir=$QTDI R --with-qt-includes=$QTDI R/include --with-qt-libraries
=$QPEDI R/lib --with-Qttopia-dir=$QPEDI R --with-gui=qpe --host=arm-linux
--target=arm-linux

```

(4) 上述配置完成后，会生成 Makefile 文件，运行编译指令：#make

(5) 将编译成功后生成的文件复制到根文件系统，运行下列指令：

```

#cd ~/konqueror-embedded/konq-embed
#cp src/konqueror ~/rootfs/opt/Qttopia/bin
#mkdir ~/rootfs/opt/Qttopia/pics/konqueror
#cp src/konqueror.png ~/rootfs/opt/Qttopia/pics/konqueror
#mkdir -p ~/rootfs/opt/kde/share/apps/khtml/css
#mkdir -p ~/rootfs/opt/kde/share/config
#cp kdesrc/khtml/css/html4.css ~/rootfs/opt/kde/share/apps/khtml/css
#cp kdesrc/kdecore/charsets . ~/rootfs/opt/kde/share/config
#mkdir opt/Qttopia/apps/Desktop
#cp src/konqueror.desktop ~/rootfs/opt/Qttopia/apps/Desktop

```

(6) 将根文件系统重新打包、下载到目标板上，系统起来后可以再桌面上看到 konqueror 图标，在网络正常连接后点击该图标，可启动嵌入式浏览器，顺利接入因特网。

5.3 Qt 界面程序开发

Qt 是一个跨平台的 C++图形用户界面库和 UI 开发框架，与 X Window 上的 GTK、Motif、Openwin 等图形界面库和 Windows 平台上的 MFC、OWL、VCL、ATL 属于同种类型，具备下列优点：优良的跨平台特性、面向对象、丰富的 API、支持 2D/3D 图形渲染、支持 OpenGL、支持 XML 等。常用于 Embedded Linux、Mac OS X、Windows、Linux/X11、Windows

CE/Mobile、Symbian、MeeGo 等系统^[34]。使用 Qt，只需一次性开发应用程序，不用重新编写源代码，便可跨不同桌面和嵌入式操作系统部署这些应用程序。

QT 的核心机制是信号和槽机制，该机制是 QT 自行定义的一种通信机制，应用于对象之间的通信。一个槽就是一个函数，当某个事件出现时，通过发送信号，可以将与之相关联的槽函数激活，执行槽函数代码。一个信号可以和一个或多个槽相关联，多个信号也可以和一个槽相关联^[33]。

开发 Qt 界面程序的流程比较简单，首先通过 Qt designer 搭建出所需要的界面，用布局管理器对控件进行布局，并用信号和槽连接相应控件，保存为*.ui 工程文件，然后运用 uic 和 moc 工具将该文件编译成*.cpp 文件，添加添加主函数、槽函数、中间函数等。

本系统主要开发的界面程序有 WiFi 网络配置界面和电源管理参数设置界面，下面分别介绍。

5.3.1 Wi Fi 网络配置界面

通常在控制台对 Wi Fi 网络进行参数配置的方法为：使用 iwlist 扫描可用 AP，用 iwconfig 设置所要登录的无线网络名称和登录密码，用 ifconfig 和 route 设置 IP 等参数。而在配置界面中这些配置工作则需要通过点击按钮来进行，实现方法为，在所制作的 Qt 配置界面中设定一些按钮来扫描可用网络和设置网络参数，这些按钮的响应函数中执行 iwlist、iwconfig、ifconfig 等配置命令，每个命令所需要用到的参数保存在文本文件中，每次执行命令就从对应的文本文件中读取。

该界面的控件选取、布局、信号与槽的连接在 qt designer 中完成，接下来的主要工作就是编写主函数和事件响应槽函数。

该 Wi Fi 网络配置界面在 Qtopia 桌面环境中相当一个 Quick Luncher 插件，需要在主函数中通过 QTOPIA_ADD_APPLICATION 产生主窗口，并通过 QTOPIA_MAIN 注册到 qtopia 中。进行注册之后的应用程序不能执行应用程序及操作，所有的操作由 QPEApplication 处理。因为需要将配置指令运行所需的参数读入并保存到脚本文件中，还需要将脚本文件中的参数读出来供指令调用时使用，故需要用到 QFile 类来定义文件。执行 ifconfig 等 shell 命令是通过调用 system 函数来实现的，该函数的参数是一个脚本文件名。

图 5.1 是所实现的网络配置界面，其中 NetWork 页面上是网络的总体配置，configure 页面上是网络连接选择，用以选择无线网、有线网或其他形式的连接，Wi Fi 页面上是在选择了一种无线网后的登陆框，IP、DNS/MAC 页面上分别是 IP 地址配置和 DNS/MAC 配置。



图5.1 网络配置界面

5.3.2 电源管理设置界面

电源管理的主要功能是控制整个系统的功耗，以起到省电的作用。电源管理使系统处于三种状态：正常状态、二级背光、休眠状态。这三种状态的具体实现会在后面章节讲述，这里仅介绍这三种状态的转换。正常状态是指系统在正常工作，与用户之间存在交互状态，所有正在使用的外围设备都在耗电，这种状态下系统的功耗最高；二级背光状态是指闲置一段时间后，LCD 的亮度自动降低，LCD 的功耗也得到相应的降低，在这种状态下敲击键盘或移动鼠标都可以使系统恢复到正常状态；休眠状态是指在保证 CPU 和整个系统能够从内存中快速恢复的前提下，系统可能进入的最低功耗状态，在系统长时间闲置后，系统将关闭 CPU 核心电压和时钟、外设时钟、大部分外围设备（包括鼠标、键盘、Wi Fi、USB 等），外部 SDRAM 保持自刷新状态，内部 SRAM 维持刷新，这种状态下，通过按系统软件设定的唤醒键或其它预定操作来唤醒系统，使其恢复到正常状态。

电源管理的设置界面主要用来设置系统从正常状态进入二级背光的时间、从二级背光状态进入休眠状态的时间、LCD 的背光值等。这些时间值的设定可以通过 slider 滑动条控件和 ComboBox 按钮下拉式选择控件来完成。图 5.2 即为本系统所实现的电源管理设置界面。



图5.2 电源管理设置界面

5.4 电源管理程序开发

嵌入式移动设备中电池能量有限，而大部分常用的 Wi Fi 模块功率消耗相比其它外设要高很多，因此，降低功率消耗对提高嵌入式 Wi Fi 移动设备的可用性具有非常重要意义。在 5.3 节中已经介绍过电源管理的作用和相关概念，可知电源管理主要用于降低系统功耗。故，在嵌入式 Wi Fi 移动终端中添加电源管理功能必不可少。

电源管理模块涉及底层硬件设备、设备驱动、Linux 内核程序和应用程序四个部分，硬件部分有专门的电源管理芯片，并提供了相应的驱动程序，软件部分由 daemon 进程和用户交互应用程序组成。本文主要介绍软件部分的实现，由于篇幅有限，只介绍实现原理。

5.4.1 电源管理的工作内容

常见的电源管理策略有多种低功耗模式和动态频率切换，嵌入式设备上电源管理所要做的工作主要包括以下几点^[35]：

（1）系统状态切换

嵌入式系统中 CPU 和系统总线的状态可分为：

- ① 全速工作状态：CPU 运行在手册推荐的最大工作频率。
- ② 降频工作模式：通过 ARM 协处理器指令降低 CPU、总线和外设的频率变换工作频率，降低功耗。
- ③ 低功耗模式：系统处于低功耗状态，涉及 CPU 时钟和电压、总线时钟、LCD、SDRAM、SRAM、RTC 时钟等在进入休眠后是否保持刷新等问题。
- ④ 深度休眠模式：也就是系统的关机状态，只有 RTC 时钟相关的一小部分电路处于工作状态。

电源管理需要控制系统在上述四种状态之间转换。

（2）控制系统底电流

当系统工作在低功耗状态，进入休眠模式后，CPU 自身功耗被降到最低，但系统总底电流和外设状态相关，需要根据系统状态适时的控制外设供电和工作，以保证总体电流的最优化。该工作通常是通过调用外围设备驱动程序注册的 Suspend 和 Resume 函数来实现。

（3）控制系统运行功耗

降低系统运行功耗通常有下面几种方法：调节系统背光亮度，或在适当场合关闭系统背光；尽量关闭不必要的外设，在需要使用的时候再上电；适时降低 CPU 频率；让不能关闭的外设在不工作时处于休眠状态；优化软件性能。

（4）与用户交互

用户可以根据自身需要设置背光亮度、电源管理进入休眠的时间等，这一部分通常是用户界面程序。

5.4.2 电源管理的软件结构

(1) 内核态程序

这部分代码的实现基于 Linux 内核中的电源管理程序，主要包括进入、退出系统低功耗模式过程的控制程序；监控硬件负载程序；对外围设备驱动程序 Suspend/Resume 函数的调用；变频操作相关过程的控制程序，如描述并控制 CPU 核心频率、电压，各种总线频率等参数；为设备驱动程序提供的电源管理接口程序。

电源管理对外围设备的功耗进行控制的时候都是通过对应设备的驱动程序来实现的，它对驱动程序提供了统一的电源管理接口，设备驱动程序只要实现了这些接口，就可以实现电源管理的功能^[36]。通常，在设备驱动中需要实现 suspend/resume 函数。suspend 用来关闭一个设备，需要两个调用，一个用于保存状态，另外一个用于关闭设备电源；resume 用来唤醒设备，也需要两个调用，一个用于给设备供电，另一个用于恢复设备的状态。关闭一个总线设备，必须关闭其所有子设备，而重新使能总线设备时，必须先使能根设备，然后使能子设备。

(2) 用户态电源管理 daemon 进程

用户态电源管理 daemon 进程是用户空间中的一套电源管理策略，负责实时获取内核中上述各部分的相关信息并进行调度控制，它为其它用户层应用程序提供了统一的接口，用来通知电源状态变化，发送休眠和唤醒消息，接受并处理各种用户限制条件，以正确进行系统状态切换。这一部分的代码需要用到大量的系统调用和接口函数^[37]。

(3) 用户界面程序

用户看到的运行有电源管理底层程序的系统表现如下：

- ① 正常工作：各设备可以正常使用，背光正常亮度。
- ② 背光变暗：一段时间没有交互，系统背光变暗。
- ③ 背光熄灭：一段时间后仍然没有交互，系统背光熄灭，部分功能可以正常工作。
- ④ 休眠状态：系统背光熄灭，所有功能关闭，可通过一些方式唤醒，如触摸屏幕等。
- ⑤ 关机状态。

系统的上述不同表现是经历不同长度时间后出现的，每个用户对每个阶段的时间要求可能不同，对不同状态背光的亮度喜好也不同，因此需要开发与用户交互的界面程序，让用户按照自身需要来设定，并随时了解电源状态。这一部分已在 5.3.2 节中实现。

第六章 系统测试

6.1 Wi Fi 无线上网测试

实现 Wi Fi 无线上网有 Wi Fi 无线网搜索与配置、Wi Fi 无线网络连接、启动嵌入式浏览器上网浏览几个步骤。下面据此展开测试。

测试之前的预备工作：将本文实现的 Wi Fi 无线通信终端置于 Wi Fi 热点区域；获取该 Wi Fi 热点区域的用户名的密码。

6.1.1 Wi Fi 无线网络连接测试

(1) 启动系统。


(2) 在 Qtopia 桌面环境中打开 Wi Fi 网络配置界面，点击 Configure Wi Fi Wi rel ess，将进行 Wi Fi 无线网络搜索，下面是搜索到 Wi Fi 网络的界面。



图6.1 搜索到的Wi Fi 网络

从图中可以看出，可以扫描到两个可用 Wi Fi 无线网络。

(3) 选择 Beacon，点击 Connect，出现 Wi Fi 连接界面，输入用户名和密码，点击 set 即开始连接。此设置界面见图 5.1。

(4) 连接后，在 Qtopia 桌面环境的右下角出现  图标，标示 Wi Fi 连接成功。关于连接成功的网络的详细信息可通过点击 NetWork 页面查看。

6.1.2 嵌入式浏览器上网测试

(1) 启动系统。

(2) 在 Qtopia 桌面环境中双击 Konqueror 浏览器图标，Konqueror 嵌入式浏览器能正常启动。

(3) 输入网址，点击连接，能正常接入因特网。浏览一些中文网站，如新浪、搜狐

等，能正常显示中文汉字和彩色图片。

6.2 电源管理测试

电源管理需要对电池电量显示、背光亮度、进入二级背光的时间、关闭背光的时间、进入休眠的时间、休眠后 Wi Fi 是否关闭、休眠能否被唤醒、唤醒后 Wi Fi 能否自动连接以及系统功耗进行测试。限于时间和篇幅，本文只介绍前面的七种。

测试之前先对各时间值和背光亮度进行设置，设置结果如图 6.2 所示：



图6.2 电源管理测试设置

从图中可以看出，当使用电池的时候，进入二级背光的时间被设置为 15s，进入休眠的时间被设置为 5min；使用交流电源时，进入二级背光的时间被设置为 30s，进入休眠的时间被设置为 5min；正常情况下的系统背光亮度为 8。默认的，在系统进入二级背光后，若还被闲置，则 15s 后背光熄灭。下表 6.1 是在上述设置条件下对系统电源管理进行测试所得到的数据。

表6.1 电源管理测试数据

待测		背光亮度	二级背光	背光熄灭	休眠	能否唤醒	Wi Fi 关闭	Wi Fi 自动连接
电 池	1	8	14s	29s	5'	鼠标唤醒	是	是
	2	8	15s	30s	4' 55' '	触屏唤醒	是	是
	3	8	15s	30s	4' 58' '	键盘唤醒	是	是
电 源	1	8	30s	45s	4' 59' '	鼠标唤醒	是	是
	2	8	29s	46s	4' 57' '	触屏唤醒	是	是
	3	8	30s	45s	5'	键盘唤醒	是	是

从测试数据可以看出，系统的电源管理程序运行正常，进入二级背光和休眠的时间有些不准，原因是在测试时存在计时误差。

第七章 结束语

本文在讨论了无线通信和嵌入式系统开发技术的基础上,提出了基于 ARM 和 Linux 的 WiFi 无线通信终端的实现方案,并完成了该无线通信终端软件部分的设计与实现。由于本系统采用的 S3C2440 微处理器具有丰富的外围设备、易于扩展,因此,在该无线通信终端的基础上可以扩展开发出各种便携式电子仪表。

该无线通信终端系统属于一个嵌入式系统,嵌入式系统的软件开发涉及从底层到上层的全过程,包括底层环境的建立、内核层软件的开发、用户层应用软件的开发。现将本文所做的工作总结如下:

(1) 建立 WiFi 无线通信终端软件开发平台。软件开发平台的搭建主要包括建立交叉编译环境、移植 BootLoader、移植 Linux 内核、制作根文件系统等。交叉编译工具链将 X86 架构下开发的软件交叉编译成 ARM 架构的可执行程序; BootLoader 主要完成硬件初始化和系统引导工作; Linux 内核控制和管理整个系统的软硬件; 根文件系统规定文件在存储设备上的存放和组织形式。

(2) 实现并移植设备驱动。本系统所涉及到的驱动程序比较多,本文详细讨论了 WiFi、LCD 以及触摸屏设备驱动的实现和移植。

(3) 开发应用程序。本系统所需要开发的应用程序主要包括与用户进行交互的界面程序和电源管理程序。其中界面程序通过 Qtopia 桌面环境和 Qt 共同实现,电源管理程序涉及底层内核编程和上层用户程序。

(4) 系统测试。测试 WiFi 无线网络接入和网页浏览功能,测试电源管理各项指标是否达标。测试结果显示,系统启动和运行正常,各种应用程序可以正常使用。

由于时间有限,在本系统中没有实现 WiFi 的可移动性功能,可移动性功能可以使 WiFi 无线设备在移动过程中正常的在两个不同的 WiFi 子网之间切换,不掉线。该功能的实现基于移动 IPv6 技术,将在下一步的工作中进行。

参考文献

- [1] 范皖勇. 嵌入式 Wi-Fi 技术应用与研究[D]. 上海: 华东师范大学, 2008: 28-30.
- [2] 黄猛, 杜红彬. 移动机器车的 Wi-Fi 接口设计[J]. 自动化仪表, 2010, 31 (3): 50-56.
- [3] 叶正鑫. 基于 802.11b 标准的无线移动终端研制[D]. 南京: 南京航空航天大学, 2007: 12-24.
- [4] 沈韬, 李绍荣. 无线网卡驱动分析与 WLAN 性能测试[J]. 通信技术, 2009, 10 (42): 105-110.
- [5] 赵义平. 802.11n 协议的研究与无线接入点功能的实现[D]. 四川: 西南交通大学, 2009.
- [6] 孙纪坤, 张小全. 嵌入式 Linux 系统开发技术详解—基于 ARM[M]. 河北: 人民邮电出版社, 2007: 180-183.
- [7] 杜春雷. ARM 体系结构与编程[M]. 北京: 清华大学出版社, 2004.
- [8] Daniel W. Lewis. FUNDAMENTALS OF EMBEDDED SOFTWARE Where C and Assembly Meet[M]. 北京: 高等教育出版社, 2004.
- [9] 吴军, 周转运. 嵌入式 Linux 系统应用基础与开发范例[M]. 北京: 人民邮电出版社, 2007.
- [10] 林晓飞, 刘彬, 张辉. 基于 ARM 嵌入式 Linux 应用开发与实例教程[M]. 北京: 清华大学出版社, 2007.
- [11] 田泽. ARM9 嵌入式 Linux 开发实验与实践[M]. 北京: 北京航空航天大学出版社, 2006.
- [12] 张晓林. 嵌入式系统设计与实践[M]. 北京: 航空航天大学出版社, 2006.
- [13] 俞用昌. Linux 设备驱动开发技术及应用[M]. 北京: 人民邮电出版社, 2008.
- [14] Robert Love. Linux 内核设计与实现[M]. 北京: 机械工业出版社, 2008.
- [15] 吴国伟, 李张, 任广臣. Linux 内核分析及高级编程[M]. 北京: 电子工业出版社, 2008: 35-44.
- [16] 倪继利. Linux 内核分析及编程[M]. 北京: 电子工业出版社, 2005.
- [17] DANIEL P. BOVET, MARCO CESATI 著. 陈莉君, 张琼声, 张宏伟译. 深入理解 Linux 内核(第三版)[M]. 北京: 中国电力出版社, 2007.
- [18] 赵炯. Linux 内核完全注释[M]. 北京: 机械工业出版社, 2004.
- [19] 毛德操, 胡希明. Linux 内核源代码情景分析. 杭州: 浙江大学出版社, 2001.
- [20] W. Richard Stevens, Stephen A. Rago 著. 尤晋元, 张亚英, 戚正伟译. UNIX 环境高级编程 [M]. 北京: 人民邮电出版社, 2006.
- [21] 刘淼. 嵌入式系统接口设计与 Linux 驱动程序开发. 北京: 北京航空航天大学出版社, 2006.
- [22] Marvell Technology, 88W8686 Datasheet Rev D, February 2007.
- [23] 索炜. 基于 S3C2440+Linux 的无线射频模块的驱动程序设计[D]. 北京: 北京邮电大学, 2008.
- [24] 董志国, 李式巨. 嵌入式 Linux 设备驱动程序开发[J]. 计算机工程与设计, 2006, 27(20): 3737-3740.
- [25] 赵孔新, 王晓红, 刘丽伟. 基于 S3C2440A 的彩色液晶显示系统设计[J]. 微计算机信息, 2007, 11(2): 163-165.
- [26] 孙天泽, 袁文菊, 张海峰. 嵌入式设计及 Linux 驱动开发指南—基于 ARM 处理器[M].
- [27] 於琪建, 张海峰. Linux 输入子系统在触摸屏驱动上的实现[J]. 机电工程, 2009, 3: 32-101.
- [28] 林学祥, 李伟鹏. 基于 MiniGUI 的 IAL 定制及触摸屏驱动程序开发[J]. 计算机工程与设

计, 2008, 6(29), 3088-3090.

- [29] 蔡海蛟, 危峻. 便携式红外相机中触摸屏原理与应用[J]. 计算机工程与设计, 2008, 4, 29(7).
- [30] 宋宝华. Linux 设备驱动开发详解[M]. 北京: 人民邮电出版社, 2008: 248-250.
- [31] Jonathan Corbet, Alessandro Rubini, Greg Kroah-hartman 著. 魏永明, 耿岳, 钟书毅译. LINUX 设备驱动程序[M]. 北京: 中国电力出版社. 2006.
- [32] Jasmin Blanchette, Mark Summerfield .C++ GUI Qt4 编程(第二版)[M]. 北京: 电子工业出版社, 2009.
- [33] 蔡志明, 卢传富, 李立夏. 精通 Qt4 编程[M]. 北京: 电子工业出版社, 2009: 87-183.
- [34] Stanley B. Lippman, Josée LaJoie, Barbara E. Moo 著. 李师贤, 蒋爱军, 梅晓勇译. C++ Primer 中文版[M]. 北京: 人民邮电出版社, 2006.
- [35] 周志敏, 周纪海, 纪爱华. 便携式电子设备电源设计与应用[M]. 北京: 人民邮电出版社, 2007.
- [36] 周志敏, 纪爱华. 绿色电源: 电子设备电源管理技术与解决方案[M]. 北京: 人民邮电出版社, 2010.
- [37] Robert Love. Linux 系统编程[M]. 南京: 东南大学出版社. 2008.

致 谢

研究生阶段即将结束，回想这两年多的学习和生活，不但学到了很多有用的知识，还参加了实际项目，使动手能力有了很大的提高。在此期间一直得到各位老师、同学和朋友的指导、关心和帮助，在此我谨向他们表示衷心的感谢！

特别要感谢我的导师周凤星老师，感谢周老师一直以来对我在学业上的严格要求和在论文上的悉心指导；感谢林老师在生活上对我无微不至的关怀和帮助；更感谢林老师以他高尚的品格情操、脚踏实地的工作态度、科学创新的探索精神言传身教，使我收益良多、受益一生。

其次，感谢实验室的兄弟姐妹以及所有帮助过我的朋友，是你们的关心与鼓励让我一路前行，辛苦并快乐着；同时，感谢联想研究院曾经的同事们，感谢他们在这篇论文研究过程中所给予的热情的指导、大力的支持和无私的帮助。

攻读硕士期间发表的论文

- [1] 刘芳华, 周凤星. 《Linux 下 Wi Fi 驱动程序设计与实现》. 自动化仪表, 已录用.

基于ARM的WiFi无线通信终端的研究与实现

作者：[刘芳华](#)
学位授予单位：[武汉科技大学](#)

本文链接：http://d.g.wanfangdata.com.cn/Thesis_D136258.aspx