电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER THESIS

（电子科技大学图标）

论文题目 **基于 S5PV210 的频谱监测设备嵌入式系统设计与实现**

学科专业 **导航、制导与控制**

学　　号 **201321190210**

作者姓名 **牛　崇**

指导教师 **秦开宇　教　授**

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 S5PV210 的频谱监测测设备嵌入式系统设计与实现

（题名和副题名）

牛 崇

（作者姓名）

指导教师 秦开宇 教 授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 学科专业 导航、制导与控制

提交论文日期 2016 年 3 月 1 日 论文答辩日期 2016 年 5 月 20 日

学位授予单位和日期 电子科技大学 2016 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

The Design and Implementation of The Embedded Systems of the Spectrum Monitoring Equipment Based on S5PV210

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Navigation, Guidance and Control**

Author: **Niu Chong**

Advisor: **Prof. Qin Kaiyu**

School: **School of Areonatics & Astronatics**

独创性声明

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

作者签名：_____ 日期： 年 月 日

论文使用授权

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

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期： 年 月 日

摘 要

随着无线电应用的越来越广，加上更加复杂的无线电环境，频谱检测设备的需求和应用愈加广泛。嵌入式设备功能为我们提供了非常多性能强，体积小，可控资源多的控制器，促使其在许多方面代替了传统计算机系统，所以被广泛应用于仪器制造中。

本论文旨在从硬件到软件实现基于高性能 ARM 处理器 S5PV210 的频谱监测设备的嵌入式系统。该系统不仅满足频谱监测设备频谱监测功能，同时也能提供各种智能设备的扩展功能，要具有良好的平台通用性。主要的研究内容有：

1. 根据需求设计系统的完整通用性方案，包括主控单元最小系统，着重设计了 ARM 端与中频信号处理单元的 HPI 通信总线和配置数据下发高速总线，频率合成模块的通信通道设计，与并设计了智能功能如网卡，USB，串口等相关功能硬件电路。

2. 分四层规划嵌入式操作系统的四层结构，针对每一层结构进行详细条理化的分析设计，同时针对项目中的难点驱动设计实现提出分层设计思想，极大程度保证系统和驱动可继承性。

3. 根据 Linux 内核 2.6 版本下的驱动体系结构，以分层思想为指导实现以微处理器为核心的频谱监测设备的信号处理系统总线驱动设计，以及中频信号处理模块的协议和驱动；针对频率合成模块的主控 FPGA 控制，运用隔离设计思想实现该设备 GPIO 和 DMA 两种控制方法；设计实现了温度检测驱动，显示系统 LCD 驱动，触控屏幕，同时提供控制应用程序供应用软件开发者参考调用。

4. 针对产品进行了软件上包括 U-boot 的网络和 LCD 驱动的智能功能实现，以及其和内核商业化定制；并针对频谱监测设备进行了精简，降低设备的功耗。

本论文设计的系统经过了完整联调验证，运行稳定，实现了所有预期目标，最终的频谱监测设备达到了产品化的要求，并提供了完整的实现方案和可继承性软件系统。

关键词：频谱监测设备，智能，高性能，通用平台，DMA，分层，隔离

ABSTRACT

With the wider radio application, plus the more complex radio environment, the demand for the spectrum monitoring equipment has become even more widespread. Because of the enhancement of the capabilities of the embedded device, it has displaced conventional computer system in many aspects, which also provides us with stronger performance, smaller size, and more controllable resource, so it is widely used in instrument manufacture.

This thesis aims to fully implement an embedded system of monitoring equipment which is based on the high performance ARM processor S5PV210. It not only meets the requirement of the spectrum monitoring equipment spectrum monitoring device functions, but also provides various extended functions which the smart devices should equipped with and an excellent universal platform. Here are the main contents of the research:

a. Accroding to the requirement of the hardware, it designs the framework of the hardware system and the master chip power supply circuit, emphatically design the HPI communication bus and high-speed data bus between the ARM and the signal processing unit, also design the intelligent devices such as network card, USB interface, serial hardware circuits and other related functions.

b. Based on the software development requirements, we divide the embedded operating system into four layers, then make detailed structured analyses and designs for each layer. For the most imoportant part of the driver design and implementation, we propose the hierarchical designing idea, which assure the inheritance of the systems and driver.

c. According to the driver architecture of the version 2.6 Linux kernel, this thesis completes the design and implementation of the bus drivers of the spectrum monitoring equipment signal processing system which is connected to the microprocessor core under the guidance of hierarchical idea. With the division thought, the thesis achevie the goal of controlling the FPGA of the frequency synthesis module by DMA access. It also equips with the temperature detecting driver, the LCD display driver, and the touchscreen driver, which also provides the control applications programs for software developers.

d. It finishes the network and LCD driver features of the U-boot, also customizes the kernel software in commercialization. To lower the power consumption of the devices, some simplification for the kernel structure have been done.

The whole system designed by this thesis has been verified by the actual experiments, it turns out to be quite stable, and achieves all the desired goals, the final spectrum monitoring equipment fulfills the requirements of a mature product, and provides a complete universal spectrum monitoring device embedded system plan and a inheritablthe software system.

Keywords: spectrum monitoring equipment, smart, high performance, universal platform, DMA, hierarchical design

目 录

第一章 绪 论

1.1 研究工作的背景与意义

频谱分析仪在电子工业界被广泛用于分析射频、音频等信号的频谱。频谱分析仪多用来研究电信号频谱结构，包括信号的频率，失真度，调制度等多种信号特征，还可以用于放大器，滤波器电路的测试，用途广泛。

现代频谱分析仪多以模拟或者数字的方式显示测量结果，其测量对象可以是1hz 以下的，严格到亚毫米波段所有无线电信号。通过分析信号的频谱，可以推演出生成这些信号的原件和电路，而且往往除了频谱分析仪其他的方法都是无法做到的。频谱分析仪可做各式各样的测量，所以在 RF 设计和测试实验室，包括许多其他专业方向的应用，都是非常有用的工具[1]。

现代频谱分析仪，基于 FFT（快速傅立叶）变换[2]，通过傅立叶变换将信号由时域转化为频域，这样就得到了频谱的状态。这种方法，需要进行数学运算，所以我们要通过模数转换器将信号从模拟信号采样转化为数字信号，再经 FFT 变换，得到频谱状况图。通过可见的在时域中的频谱，我们可以对一个信号的谐波和杂波进行分析，同时可以看到调制信号的宽度也是相当有用。这些对于任何形式的发射机，包括蜂窝网络，无线网络，和其他任何无线电或者相关的无线电应用，都是极为有用的。

频谱图包含了这个信号的许多相关特征，可以方便推论出电磁信号原本的调制类型和发射端[3]。频谱监测设备可以协助推断产生这些干扰的可能设备，从而快速判断该设备的存在范围。比如现在极为隐蔽的非法基站，这些非法基站盗用了运营商的无线频率，进行非法的活动，靠频谱监测设备不仅可以判断信号产生的设备，同时也可以通过类似场强仪的功能定位非法基站。

对频谱监测设备的研制国内外也一直在不断进步，不断提高相关设备的参数，包括分析仪的频率范围、分辨力、分析谱宽、分析时间、扫频速度、灵敏度、显示方式和假响应等等各项参数[4]。频谱监测设备往往包含有前端的接收机，信号处理，图象显示，控制输入，外围接口等等各个方面，主控和显示部分一开始往往采用的都是 PC 等量级的控制模块，供电部分也需要相当的功率，这就意味着频谱监测设备是一个相当复杂的整体系统，所以往往频谱监测设备的产品都相对来说比较庞大，便携性移动性比较差，甚至需要专门的移动拖车之类的设备辅助其移动。

正如前面所说，现在无线电环境复杂，相应的测量环境也会有不同的变化。我们不再是单一的实验室测量或者是室内等小范围的测量，需要机动性的变化测量。比如在进行非法基站检测时，我们需要移动性地定位非法基站的信号来源和测量其频谱范围，这时如果是传统的实验室仪器的话，在便携性较差的情况下，某些复杂的地形中这种仪器就成了最大的障碍。我们亟待有便携式的频谱监测设备，这样就可以实施随地地进行信号监测[5]。

用户在使用便携式仪表的过程中，也不仅仅是追求便携性，同时也希望在便携的同时也能有良好的测量精度和准确性。

后期随着技术的成熟和进步，逐渐规范化了便携式频谱仪的定义，其应用场景主要是某些在户外进行测量，或者需要可自由移动的场景等，这就特别需要频谱仪具有便携的特性[6]。一个便携式频谱分析仪需要包含以下几个特性：

1.可选电池供电方式，使得用户可以自由在户外移动；

2.可视显示器，在阳光下，包括黑暗或能见度较低的情况可轻松读取；

3.重量较轻（一般小于 7 公斤）。

符合相关条件的设备才可称为标准的便携式频谱监测设备[7]。

随着嵌入式设备功能的增强，嵌入式系统可以做许多原来高性能处理器才可以做的事情，由于嵌入式系统往往都体积较小，可扩展性，可定制性也极强，所以被广泛应用于测量仪器中。

频谱监测设备对便携性的要求越来越强烈，嵌入式系统为频谱监测设备的便携性提供了可能。同时，由于嵌入式系统具有极强的可扩展性和硬件可裁剪性，频谱监测设备也被添加了众多功能，比如触摸屏控制，网口转发数据，USB 可扩展接口等等[7]，这就是现代仪器的便携式频谱仪和手持式频谱仪，也正是本项目的研究意义所在。

1.2 国内外研究历史与现状

频谱分析仪经历模拟电路、单片机频谱仪、数字化频谱仪等多种类型[8]。集成电路和数字电路的发展极大地促进了频谱仪的小型化和高精度化。早期的频谱仪体积巨大，如今已经进化到掌上设备那么大，整体上频谱监测设备都向小型化，轻量化，高精度化发展[9]。

1.2.1 微型频谱监测设备国内外发展状况

根据可查的行业报告显示预计在 2019 年将会达到 150 亿美元（约合人民币 1

000 亿)，而在 2008 年中国频谱分析仪市场销售额已经达到 6.28 亿元，而且近几年无线电应用市场更是蓬勃发展，测量仪器的商业市场基本只有几个行业举头，诸如美国 Agilent、Tektronix、Aeroflex，德国的 Rohde Schwarz，日本的 Anritsu 公司等公司[10]，所以低价位低成本的市场很大，潜力无限。

随着便携类的频谱监测设备和传统台式设备的测量精度和功能上差别的减小，微型频谱监测设备更是具有极大的发展潜力，同时由于业界频谱监测设备的便携性需求量是巨大的，而且因此世界上的顶级测量设备生产厂商在已经拥有了大型台式频谱分析设备的相对成熟的生产技术之后，便携性的频谱监测设备市场更是成为各大厂家争相分食的饕餮大餐，都相继开发了便携类的频谱监测设备，来满足不同的场景需求。

Rohde Schwarz 的 R&S FSW 系列是顶级的信号分析仪[11]，利用实时频谱分析功能查看、捕获并分析最罕见的信号，包括已知和未知信号，实施检测带宽可达 510MHz ，无杂散动态范围也达到了 75 dBc ，查看以往难以观测的信号，使用本底噪声扩展（NFE）技术将测量噪声降低最多 10 dB，可以提高低电平信号测量的精度。

图 1-1 R&S○R FSW 频谱与信号分析仪

Agilent PSA 和 RS FSU 系列具有高端频谱仪的各项测量指标非常先进，非常适合高端科研和航天领域。 Agilent 和 R&S 生产了许多价格始终的监测设备，尽管这些设备价格并不高，但其功能指标叶然国内的仪器生产厂商也望其项背。

图 1-2 Agilent X-Series N9040B UXA 频谱与信号分析仪

Rohde Schwarz 推出了顶级的顶级 ZVH 系列手持式设备，还有覆盖较广的 R&S○R FSH 系列，在各种移动测试场景下非常适用。

图 1-3 R&S ZVH 手持式频谱与信号分析仪

R&S○R ZVH 集快速的天馈线分析与高性能的频谱测试于一身，发射机的安装、无线干扰查找都是非常好的应用场景。该仪器频率范围：100 kHz～3.6/8 GHz 传输测量动态范围高达 100 dB，内置直流偏置供电，方便有源器件测试，分辨率带宽最低可达 1Hz，带电池重量只有 3 kg，支持的工作时间可达 4.5 小时，还包含有用于远程控制和测量数据传送的 LAN 口和 USB 接口，测量结果可保存在 SD 卡中。而应用更广泛的 R&S FSH 系列，是罗德与施瓦茨公司全新推出手持式频谱分析仪。它集多种测试功能于一身，达到了中高档台式频谱仪测量指标，是行业内顶尖的手持式射频仪器。同样满足手持式频谱仪的所有特性，重量轻，易于手持，电池供电，具有可用于远程控制和测量数据传送的 LAN 口和 USB 接口，测量结果保存在 SD 卡中以便后续提取。

图 1-4 R&S FSH 手持式频谱分析仪

相比之下国内的频谱监测设备的起步较晚，在国际上频谱监测设备做的精良的也只是上述的几个公司，余下基本都是是技术上相对落后的。随着中国频谱分析仪市场的火热，而且近几年无线电应用市场更是蓬勃发展，国内的频谱监测设备研发也是呈现一片欣欣向荣的状况。中国电子科技集团第 41 所，致力与测试测量仪器的设计制造[1]。该机构推出的 AV4051，如图 1-5，系列信号分析仪具有

优良的测试动态范围、相位噪声、幅度精度和测试速度，具备高灵敏度频谱分析，具有良好的扩展能力，可通过灵活配置选件进一步提升测试性能，也可通过各种数字和模拟信号输出接口构建测试系统或进行二次开发。

图 1-5 中电 41 所推出的 AV4051 频谱信号分析仪

针对手持式频谱分析仪，中电 41 所也推出了一款相对成熟的设备，如图 1-6 所示的 AV4022。该设备频率范围在 9kHz 3GHz，测噪声边带小于-110dBc/Hz（载波 1GHz，频偏 1MHz），同时该设备也满足手持式频谱监测设备的所有特性，功耗小（低于 15W)，内涵锂电池，工作时长不低于 4 小时。该设备可通过串口与 PC 机通信进行测试数据的传输与通信。AV4022 的出现，一定程度上填补了国内手持式频谱监测设备研发的空白，但是国内相关领域的发展仍然是任重道远。

图 1-6 中电 41 所推出的 AV4022 手持式频谱信号分析仪

1.2.2 嵌入式系统在频谱监测设备的应用

嵌入式系统一般是在一个大型的机械或者电器系统中为了完成一个专用的功能而设计的计算机系统，该系统往往是实时计算的操作系统[12]。

嵌入式计算机在应用数量上远远超过了传统的 PC，PC 机的外部设备中其实就包含多个嵌入式系统，比如常见的读卡器，显卡、打印机等等，这些设备均配备有微型的嵌入式处理器。在制造业、通信领域、仪器仪表、消费电子行业都具有广泛的应用。作为嵌入式系统发展最为迅猛的 ARM 嵌入式系统，更是在现代技术的发展中承担了不可磨灭的贡献和角色[13]。基于 ARM 核的微控制器芯片在高度微控制器市场不仅独领风骚，同时像低端微控制器应用领域展开攻势，AR

M 微控制器的低功耗、高性价比更是对传统的微控制器构成了巨大威胁。

现有的嵌入式控制器特别是 ARM 系列的飞速发展，更是满足了频谱监测设备的需求。ARM 架构从 V1 版架构陆续在吧不到 30 年内升级到 ARMV8 系列，内核升级到 Cortex 系列[14]，ARM 处理器的性能越来越强悍，能效比越来越高，应用范围越来越广。

根据以下微型频谱监测设备的要求，现有的高性能 ARM 嵌入式系统，供电方式多是锂离子电池的方式供电，最新的 ARM Cortex-A32 构架的功耗更是低到 0.004W；众多的 ARM 嵌入式设备都可配备 LCD 的显示屏幕，现如今的手机便是 ARM 嵌入式设备最好的证明[15]；根据嵌入式设备的特点，更是知道其体积极小，就手机来说，其上集成了摄像头，重力传感器，温度传感器，语音传感器，外设存储等等相关设备之后还是可以如此小巧，那么针对微型频谱监测设备来说体积上就更不成问题。针对这些方面，在微型频谱监测设备上采用嵌入式系统作为主控设备便是不二选择。

根据对频谱监测设备的分析我们可以知道，微型频谱仪的最主要的特性就是便携，低功耗，扩展性强，同时也需要有强大的运算能力，而 ARM 处理器控制核心正是呼应了这样的要求，也就注定了 ARM 会在仪器的设计研发上大显身手。

1.3 本文的研究内容与贡献

本项目系频谱监测设备首次采用高性能 ARM 处理器 SAMSUNG 的 S5PV210，最终的产品实现了智能频谱检测设备的嵌入式系统，作为频谱监测设备在便携和手持上的通用平台，经各项验证，其功能丰富，处理能力极强，可扩展能力也极强；提炼的驱动设计分层隔离的驱动设计理念使得该嵌入式系统具备较强的通用性，为以后产品随着时代的设计提供了便捷有利的条件。

本文主要的研究内容为：

·第一章绪论

首先介绍了频谱检测设备研究意义和技术发展历程，之后说明了国内外频谱监测设备的的研究状况以及未来的市场前景；接着介绍了嵌入式系统的优良特性，并指出嵌入式系统在测量设备中的应用是未来测量仪器设备的重要发展方向，并简要概括了现阶段的嵌入式系统在频谱监测设备中的应用状态；最终概括了本设计的成果和研究意义，以及本文的结构安排。

·第二章频谱监测设备嵌入式系统需求分析

首先分析了频谱检测设备的信号处理部分对主控平台嵌入式系统的性能需求，

包括处理器的工作主频和内存需求，之后针对频谱检测设备的功能性需求提出了详细的要求，包括控制射频输入前端和中频数字信号处理单元的通信需求等等，还提出了实现智能设备所需的如显示，触控，网络，监察，温度监测等各项详细功能；最后针对嵌入式系统的软件，从底层到上层也提出了各项需求。

·第三章嵌入式系统设计

根据频第二章频谱检测设备的嵌入式系统的需求，有针对性地设计嵌入式系统；硬件系统上选取高性能主控芯片 S5PV210，首先搭建 ARM 最小系统，设计其电源，供电，调试等接口电路，并着重设计与数字化中频信号处理单元的高速通信总线 HPI 总线以及 FPGA 重配总线，并针对 FPGA 重配总线进行两种设计，第一是为了兼容前代版本的信号处理模块，第二种方案着重提高重配速度，提升设备性能和交互性，同时与上层应用开发人员协商相应的通信数据格式；接着设计了与射频输入前端的通信总线，并在协商与其通信的数据格式；设计 LCD 的显示功能，同时为了实现良好的交互设计了屏幕的触控功能，详细分析了触摸功能所需的各项环境；同时为实现频谱监测设备的远程监控和数据传输功能设计具体的网络接口方案；以及设计了温度监测，电量监测等等。这些设计均包含完整的方案设计和原理图绘制。

软件设计上，首先根据嵌入式系统主控平台选择优秀合理的开发工具并在硬件搭建完毕的基础上搭建软件开发环境，继而因嵌入式系统的软件设计从底层到上层分为四层，从底层的 Bootloader 选择到设计，到 Linux 操作系统的内核选择与移植，内核中与硬件相关的核心部分驱动设计体系的深入分析理解，再到上层应用的运行环境搭建针对嵌入式系统的软件系统与驱动的交互方法，归纳出内核驱动的设计理念–面向对象和分层隔离思想，该理念好可保证该设计具有更好的继承性和迭代更新特性，并在软件系统设计中履行该设计理念。

·第四章基于分层隔离设计理念的驱动实现

Linux 驱动设计需要了解硬件的工作原理，同时也要匹配 Linux 操作系统的软件文件操作接口。FPGA 作为数字化中频处理的主要组件，其工作有滤波，数字下变频等。FPGA 的内部资源是宝贵而且一定的，如果需要切换其工作模式，需要我们能够动态重配 FPGA，此举在某种成程度上还降低了 FPGA 的功耗。而 FPGA 的重配是章程化的，有相应的硬件控制逻辑，需要按照其控制逻辑和配置过程实现电路和驱动的设计，并尝试采用 DMA 方式快速重配，使其能够在设计的系统板上正常工作，并且需要保证实现 FPGA 的高速重配。

DSP 与系统处理器传输了大量的经过 FFT 变换的频谱数据，对总线的传输速度有相当的要求，DSP 配备了高速的 HPI 高速的数据总线，利用该总线在

ARM 端实现其控制逻辑与控制时序，完成数据通信。S5PV210 芯片提供 USB-OTG 的引脚功能，但是据目前该芯片在各个厂商的测试板 Linux 内核中都是不含有该功能模块的，但内核中集成了 OTG 的控制器，我们这就需要我们主动去设计移植驱动，最终在微型频谱监测设备上测试并实现该功能；接着分析了 LCD 显示器的 Framebuffer 机制及驱动实现，并着重通过触摸屏提高了屏幕大的交互功能，同时完成触控屏幕 Qt 库移植，保证上层应用可正常调用触摸屏并作出快速响应。针对每个总线以及驱动都进行了稳定性测试和性能测试，保证每个驱动都可圆满完成任务并符合需求。

·第五章系统优化与个性化定制

手持式频谱监测设备主要与信号处理单元的功能实现外，还有额外的扩展功能，比如网络系统，电量温度，触屏控制等，分别分析实现了各类外设的扩展功能；然后针对 Linux 的内核，在移植过程中还需要对齐进行合理的精简，提高其效率并降低资源消耗；同时在 U-boot 的移植过程中还需要对 U-boot 进行功能性的定制，比如网卡驱动，LCD 显示驱动，以及其环境变量，包括需要下发给内核的相关参数的调整等；同时仪器在上电后即需要显示出相关的交互信息，所以在 U-boot 中移植 LCD 显示器的驱动，并进行相关测试。

U-boot 的定制设计到仪器在上电时提供相关的交互信息，使得产品具有现代感，个性化，这就涉及到 U-boot 启动 logo 和输出信息的个性化定制。内核需要针对频谱监测设备的生产商进行个性化的定制，包括内核启动 logo，启动应用程序的过度动画等等的相关定制都可以在内核中完成。

第六章 全文总结与展望

总结了本论文完成的贡献和创新点，并对该系统的其他应用场景进行了分析，另外提出了未来的改进方向。

第二章 频谱监测设备嵌入式系统需求分析

频谱监测设备，除了测量功能，也需要其兼具监视监察的功能，这就要求设备需要具有一定的智能性。理想的状态就是该设备可以用在信号的测量上，也能够及时通过网络之类的接口向服务器反馈相应的信息，同时我们也希望该设备可以储存监测的数据以供后续的排查检测。

针对此类智能型设备的实现，选取嵌入式系统作为其智能主控平台的核心控制系统，而本文的研究目标就是构建一个完整的嵌入式软硬平台来实现其所有的包括频谱仪具备的测量功能，以及智能型的相关功能。

同时该嵌入式系统可以具有良好的通用性，比如如果我们更换相应的信号处理模块，该嵌入式系统同样能够提供良好的兼容性，能广泛应用于各类频谱检测平台，包括传统的台式仪器平台，便携频谱检测平台以及手持式频谱监测平台等等。

2.1 频谱监测设备结构

本项目旨在设计的现代实时频谱监测设备，实时频谱仪最大的优点就是能够充分利用从时域得到的采样数据，实现完全的频谱测量，并能够按特定频谱的分量进行触发，其中利用快速傅立叶变换，对数字信号进行处理分析[16]。

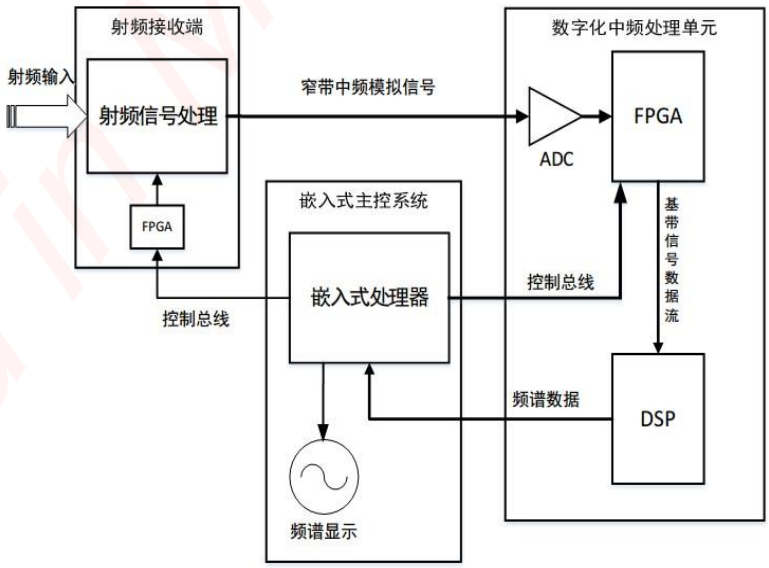

图 2-1 频谱检测设备整体结构

如图 2-1，信号的输入端被称为射频接收端，射频接收端对射频信号进行一定的处理，诸如滤波，下变频等等；处理完成的信号进行模数转换，经高速 ADC 采样，并将其传输至数字中频信号处理单元；数字中频信号处理单元在接收到数字信号后，由中频数字处理模块的 FPGA 对数字中频信号进行存储，滤波等等处理，包括数字下变频为基带信号；处理完成的基带信号再交由数字中频信号处理单元的数字信号处理器进行快速 FFT 变换，将时域数字信号转换成频谱信息；经数字信号处理器处理完毕的频谱数据经总线传输至嵌入式主控平台，嵌入式主控平台对谱数据进行格式转换，特征提取，包括频谱数据分析，显示，需要具备时间频率幅度的三维显示，甚至包括频谱密度的显示，另外还有存储，转发等等；嵌入式主控平台还涉及到射频接收前端和数字中频信号处理单元的相关控制和模式转换等等。

一个完整功能的智能化频谱监测设备，需要实现如下几项要求：

1.信号频谱的实时分析显示

2.显示设备的良好支持（LCD 或者 VGA）

3.可实现谱数据的存储与转发

4.逻辑清晰，响应迅速的交互界面

5.触控屏幕的操作支持

6.仪器标准接口 GPIB

7.智能设备接口以太网接口，USB 接口，OTG 功能

8.低功耗，高性能，个性化，定制化这其中许多重要的功能实现都需要嵌入式主控系统的良好支持才能得以实现。

2.1.1 射频接收前端与嵌入式单元交互需求

在工程应用中，这些信号就是射频信号，这就意味着一个频谱监测设备必然不能离开信号的接收端[17]，该接收机前端主要完成的工作就是：

输入衰减：提高匹配能力和打信号测量功能，保护混频器、放大器等因为信号的输入功率过大，造成元器件的烧毁或者出现信号的功率损失等等。

波段滤波：是当地进行滤波，同时提供波段的切换功能，可以提高仪器的测量带宽变频：将射频的宽频带信号转成窄带中频信号，方便后续的信号处理还包括后续对中频信号进行相关的优化处理等等。

射频信号进入之后经过射频接收前端的处理变成后续单元可以进行处理的中频信号，该单元也包含了许多相应硬件单元，包括一些可调控的单元，在频谱监测设备进行相关的测量模式调整时也需要此类单元进行相应的参数调整。

这就需要控制系统进行相关的控制参数的下发并设计相应的通信通道对射频接收前端进行控制，这也对控制系统提出了控制需求，这个通道的数据量要求不大，但是响应速度要快，而且要保证该总线设计简洁，在小型低性能 FPGA 上也能够很好地实现其控制时序。

2.1.2 数字化中频信号处理单元

数字化中频信号处理单元的输入信号显然就是射频接收前端接收外界输入信号后经过处理的中输出，也就是中频信号。该单元需要对信号转化处理，用的都是数字化的信号处理器，这也就意味着需要将模拟信号数字化，进行时域到频域的变换等等，那么该单元具体的功能如下：

1.中频信号数字化：采用高性能 ADC 对射频信号进行采用并进行模数转换，将其转化为数字中频信号。

2.数字信号下变频：中频信号需要转换为亟待数字信号才可用于后续的数字信号处理器的处理。

3.频域频域变换：将采样的到时域数字基带信号经过 FFT 变换转化为频域信息，得到信号的幅度，频率，相位等等各项测量信息。

4.数字检波：通过多重采样技术，提高或者降低采样率进行更宽的带宽检测，在检波模式下，采样率降低，但是采样带宽提高，通过包络检测可以判断目标所处带宽，进行检波后自动变换为目标的单点模式。

这里主要涉及到两个主要的信号处理芯片，其一是 Xlinx 的 A7 FPGA，该可编程逻辑阵列具有强大的性能，在此单元主要承担的是数字信号下变频，以及一些数字信号的优化处理，诸如滤波优化等等[18] 。该信号处理模块也决定了频谱检测设备数字频谱仪的工作模式。在进行数字检波到单点检波模式切换的时候，由单点模式切换到检波模式是需要切换采样率的，这个时候需要数字中频信号下变频单元的处理其 FPGA 进行相应的处理模式切换，而切换需要进行 FPGA 的重配，而这个过程可能在频谱监测设备的使用过程中随时可能由于用户的模式切换而发生，这就需要嵌入式主控平台能够主动去重配 FPGA，这就需要主控平台同数字信号下边拍照单元同样需要有通信控制总线；同时我们也希望有检波模式到单点模式切换，这个场景可能发生在用户主动切换模式或者是检波模式自动检测目标信号的所处带宽进行主动切换。

在切换相应的模式时，如果切换时间过长，就会发生数字信号处理器这一端送出空数据或者错误数据的情况，同时明显感觉到上层的界面应用程序的卡顿感，所以切换时长应该控制在 1-2s 左右，这样相应的操作才会更流畅，交互性更好，

这就是动态重配 FPGA 的要求。

动态重配 FPGA 的的过程中涉及到一系列的逻辑控制，需要调动一定的控制时序才可启动 FPGA 的重配过程，而这些必然是交由嵌入式主控平台中的 ARM 处理器完成，而且需要 ARM 主控平台进行配置文件的下发烧写，这也就以为这 ARM 和该 FPGA 之间除了控制逻辑意外还需要有数据通信总线。A7 的配置文件是编译完成的二进制可烧写文件，文件大小约在 9MB-12MB，根据转换时间的要求，这条数据总线的传输速率不能低于 6MB/s，也就是在 50Mbps 的带宽以上才可保证良好地体验。

该中频数字信号处理单元还有很重要的一个部分便是数字信号处理器，根据现有的信号处理单元方案设计，该处理器采用的是 6416 数字信号处理器（DSP），DSP 芯片由于其读点运算能力强大，对数字信号处理算法有这不可比拟的优势。本论文中，该处理器收到数字化处理后的中频信号进行后续的频谱数据处理。

在本单元中，其主要完成的工作便是将已经经过数字下变频的基带信号经过快速 FFT 变换，得到谱信息，同时计算得到信号的幅度，相位等测量信息[19]。该数字信号处理器会将相应的谱数据传输至嵌入式主控平台的 ARM 处理器，ARM 完成数据接收之后，将其转换为可视的图像和数字特征呈献给用户，同时也可将相应的数据进行保存或者转发，比如将数据保存至存储器，定期刷新或者删除旧的数据，另外还有可能根据用户的远程操控需要将数据通过网络协议进行远距离的传输。

这里所设计的许多功能都需要嵌入式主控平台进行相应的控制和参数下发等等：频域变换后的谱数据经过数字信号处理器的处理之后需要交由主控系统进行分析和图像转化，将其显示为直观的频谱信息，这就需要数字信号处理器和主控系统之间有数据通信通道，根据该频谱检测数据的数据量，计划采样点在 800 个点到 3200 个数据点进行转换，每个点为 4 个字节，主控系统显示平台上，良好视觉的频谱刷新频率为 24HZ，由此知道数据通信总线的带宽要达到 2500kbps 以上。

以上是中频数字信号处理单元对嵌入式主控系统的需求。

2.2 嵌入式系统主控单元需求

嵌入式系统主控单元是嵌入式系统的核心，也是主控控制系统的核心，是与用户有直接交互的系统单元[20]。实时频谱仪需要大量的数据处理，同时还要完成多种格式的数据的显示和刷新，实时频谱分析仪对数字化中频信号处理输出的数字信号进行数据处理，根据得到的频谱数据，进行格式的封装和转换，交由主控

系统进行数据格式处理，分析数据，转化为图像显示，该设备对嵌入式处理器多任务性能要求较高，同时也需要优秀的 2D 图像处理引擎来完成图像的显示。上层应用软件运行于该嵌入式中控平台，该应用软件的运行消耗在主频为最高为 180Mhz 的 arm9 上，一旦有大量的数据处理和控制交互时，处理器出现长时间占用 100%，软件会出现严重卡顿甚至不能响应，交互性大大变差；而在主频为 533Mhz 的 ARM11 处理器上，同样在出现 80%甚至 100%的占用，出现界面谱数据刷新卡顿，鼠标移动出现严重掉帧现象，交互性极差的情况。该软件的内存占用大约是在，一般来说系统本身其他服务的占用约在，这也就意味着该嵌入式中控平台的内存至少在以上，当然是在合理范围内越大越好。此处为嵌入式中控平台的处理器性能提出了两点要求，第一是其多任务处理性能要足够，主频在 1Ghz 以上最好，同时主设备的内存在 500MB 左右。

主控平台除了谱数据的处理和应用软件的运行性能需求等等，还需要更多的功能性的要求，比如液晶显示功能，网络传输，数据存储，串口调试功能，温度电量检测等等功能。细化具体的主控系统的功能性需求，具体有：

·提供优秀的操作系统运行平台并提供上层应用软件的运行环境

·数字中频处理单元处理完成的谱数据的接收和处理，能够适应各种不同的分析技术需求

·数字化中频信号处理单元通信通道，用来完成该单元数字下变频处理器的切换及配置文件下发

·射频接收端单元硬件控制命令下发，硬件状态检测功能

·LCD 液晶显示功能，触控操作功能

·物理按键，包括复位功能等

·USB 2.0 HOST 和 USB 2.0 OTG 接口功能

·两路 UART 接口

·10M/100M 网络接口，网络传输等功能

·SPI 接口功能

·micro-SD 卡接口

·温度测量功能

·电量监测功能

综合以上细化的功能需求，设计嵌入式系统的框架如下图 2-2 所示

频谱监测设备的嵌入式系统硬件单元设计是设计的重中之重，有了该主控系统，其他模块才可以正常组合，整个频谱监测设备才可以成为完善成熟度的设备。硬件系统设计方案的好坏对于后续的软件设计也有莫大的影响，优秀的硬件设计

可保证软件系统的正常运行，配合软件系统达到整个系统的完美协作。

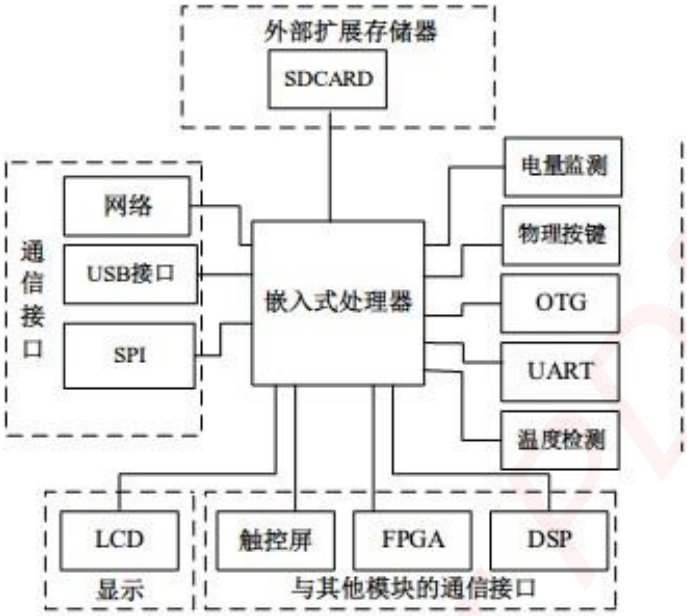

图 2-2 嵌入式框架图

在硬件设计需求提出之后，更多地是需要软件系统对硬件的配合实现，如果没有软件的实现，所有的硬件也不可能实现其应有的功能，所以硬件是软件设计的基础，特别是针对频谱监测设备来说，任何一个硬件上的需求都从下到上需要配合设计才能发挥其最大功效。

嵌入式操作系完成嵌入式系统的软、硬件资源分配、任务调度，控制、协调等各类操作[21]。

对本论文来说，对操作系统的需求有：

·操作系统内核精悍

本频谱监测设备系统主打精悍，高效，低功耗，所以系统资源相对有限，操作系统的内核决定了操作系统的性能，所以要求内核小而精。

·多任务

频谱监测设备在处理频谱数据的同时还需要多项其他的任务能够同时处理，比如在显示频谱的同时，能够完成温度，电量数据的采集，完成数据的存储和通过网口的转发，包括可以自由的进行数字化中频信号处理单元的调配等等。所以针对此，提出对嵌入式系统的有较强的多任务处理功能，可以完成操作系统的多进程多任务的协调控制。

·丰富的开发工具

进行频谱设备的嵌入式软件系统的开发时，由于其设计到多项功能的开发，

所以往往繁琐，底层硬件和软件需要良好配合，这个时候就对软件开发工具提出了一些列的需求。嵌入式系统本身并不具备直接开发的能力，设计完成之后，开发者是不可能在嵌入式系统上直接对开发完毕的软件直接在嵌入式系统上进行再度修改编译，所以需要有一套高效易用的开发工具，这些工具不仅包括软件的开发环境，比如 Linux 嵌入式常用的 Linux PC，还需要包括硬件的测试和调试工具，比如示波器，逻辑分析仪，信号发生器等等。

·灵活性

在开发嵌入式软件系统时，很重要的一部分是硬件功能的软件实现，这里就设计到驱动的设计。一套良好的驱动设计体系决定了开发的难度和灵活性。本设计希望该嵌入式软件系统的驱动开发能够更具条理性，更加结构化，同时在进行相应的功能添加删除时能够更加灵活，呈现模块效果，开发者可以针对功能进行合理的添加和删除以及修改。

·继承性

本设计中的嵌入式软件系统，希望其可以在多代设备中得到良好的继承和实现，在未来的产品更新换代的时候也希望其中的设计可以经过简单的修改和移植实现。这就需要我们尽量减少其对硬件差异化的依赖，按照更加优秀的设计理念对软件系统进行设计，使得软件和硬件可以相对独立化层次化，使得其具有更好的继承性。

2.3 本章小结

本章简要分析了频谱检测设备的主要框架，然后深入介绍了该项目中嵌入式系统的硬件设计框架，包括所有的外设等等；接着根据项目的软件需求设计了系统的软件框架，包括其从底层到上层的四层结构，并对每层的需求进行分析，最终确定各类软件平台。

第三章 嵌入式系统方案设计

一个完整的嵌入式系统的设计包含硬件和软件的两部分设计，硬件的选型和设计决定了软件的最优性能和配合程度，软件的设计需要依赖硬件的框架结构，两者相辅相成组成一个整体。硬件设计其中包含了核心的嵌入主控芯片，闪存以及 RAM 模块，电源时钟电路，调试接口，外设功能性接口如网口，USB 接口，LCD 模块，触控屏模块，OTG 接口，以及与数字化中频信号处理单元 DSP 和 FPGA 的总线接口；软件设计主要包含了引导启动程序的移植与设计，以及操作系统的移植与设计，核心部分就是硬件功能在软件上的实现，各个功能模块在驱动上的设计与实现。

3.1 硬件平台设计

首先根据频谱监测设备的需求进行嵌入式主控芯片的选型，根据该处理完成完成整体系统的硬件搭建与设计。包括现代频谱检测设备的一些特殊外设的设计与开发，比如：与数字信号处理器的通信接口电路设计，对 FPGA 的动态配置电路设计，射频单元控制系统设计，显示子系统与触控屏幕，USB-OTG，网卡，电池电量检测等硬件电路的匹配性设计等。

3.1.1 频谱检测设备嵌入式系统的硬件框架

由第二章的性能需求分析，本主控系统采用 ARM 三星 S5PV210 嵌入式硬件平台。S5PV210 是三星推出的名为“蜂鸟”的处理器，被广泛用于平板设备和智能手机等设备。对嵌入式各类系统的支持也相当广泛，最重要的是其性价比极高，价格低廉，但是其最高主频可达 1Ghz，数据/指令一级缓存 32/32KB，二级缓存 512KB，最高可达 20 亿条指令集的单位时间运算速度，这使得其在运行该频谱监测软件时响应速度得到大大提升，免除卡顿的烦恼；同时设计采用 4 片 K4T1G164QF 的 128M Byte DDR2 组成 512M 的运行内存，足以满足各项处理任务的内存需求；同时其具有优秀的 2D/3D 图像加速，可保证频谱数据转化为图像后良好的展现。

S5PV210 提供包括：高带宽可复用的 AHB 总线，可保证高速数据的交互；24 位 LCD 控制器，最大支持 1024*768 分辨率，4 路 UART，24 通道的 DMA 控制器，一路 USB HOST 2.0 和 USB-OTG 可实现 480Mbps 的通讯速率，两路 SPI，2

37 个可支配的 GPIO 等等外围接口[22]，这也是我们在本项目中的频谱监测设备所需要用到几个重要的接口。

从性能和功能上来看，该处理其完全满足所有的频谱检测设备的需求，设计的框架结构如图 3-1。

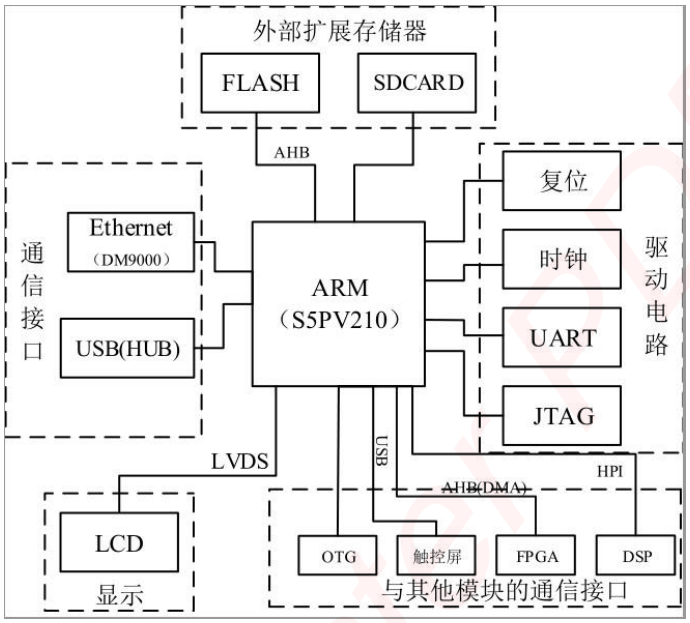

图 3-1 嵌入式系统

3.1.2 供电设计

主控芯片 S5PV210 核心板需要 5V 和 12V 的直流电源输入，另外设计板上需 LCD 和 USB 的供电供应，所以需要设计这两块的供电电路，具体的设计如下，J20 是核心板的底座。采用 NFE31PT222Z1E9 静噪滤波，保护相关设备免于静电的危险。

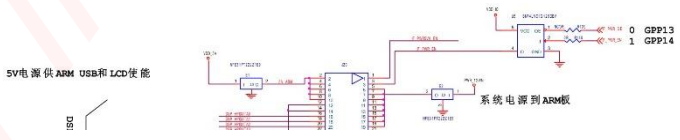

图 3-2 嵌入式系统框架图

上电启动各项芯片以及 LCD 模块等等是有一定的时间逻辑的，首先面板处于待机状态，点击开关后，ARM 进入启动状态，同时中频单元的 5V 电源启动，LCD 模块被点亮，显示屏进入显示阶段，接着如果 ARM 端启动进入内核状态，ARM 启动完成后通过负载开关启动 FPGA 和 DSP 的上电顺序控制芯片，从而按

照上电顺序启动此两个芯片。上电顺序如下图 3-3 所示

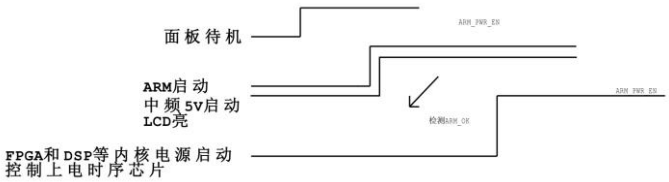

图 3-3 上电时序逻辑

3.1.3 与数字中频处理单元的通信总线设计

据第二章的需求分析知道，嵌入式主控系统与中频信号处理单元需要有两个重要的数据交互总线，其一是与数字信号处理器 DSP 的交互总线，主要完成谱数据的交互，其二是与中频数字下变频的 FPGA A7 的数据交互总线，主要完成配置时序的启动和配置文件的下发。具体到详细的设计，我们需要根据 DSP 和 FPGA 现有的硬件资源进行合理的分析。

3.1.3.1 HPI 总线设计

数字信号处理器（DSP）是一种专门的微处理器，其架构针对数字信号处理的业务需求进行了优化。DSP 的目标通常是测量，过滤器或压缩连续现实世界的模拟信号。大多数通用微处理器也能成功执行数字信号处理算法，但专用的 DSP 通常有更好的电源效率，因此它们更适合在便携式设备，比如如功耗限制较强额设备上。DSP 通常使用特殊内存架构，能够同时获取多个数据指令[23]。

主机接口即 HPI（Host Port Interface）是 DSP 芯片与主机通信的并行数据通信接口，它的主要应用场景多实在 DSP 与其他的额设备进行高速数据通信的时候[24]。一般来说有 16 或 32 位的数据宽度，时钟是由高速主机来提供，所以速度往往很快。HPI 在 DSP 上的体现就主要是一些控制寄存器，针对主机来说，主机可以控制 DSP 的所有寄存器，可以通过 HPI 的协议来访问所有的存储区域；如果 DSP 的外围设备在 DSP 的存储上有映射，那么主机同样也可以访问这些被映射的存储区域而达到访问外设的目的。

本论文针对 S5PV210 采用数据和地址复用模式进行总线的连接设计，在复用模式下没有直接的地址线，主机访问 DSP 的地址信息是以数据方式送到 HPIA(HPI 地址寄存)。从硬件信号的角度，地址，数据信号是由同一组数据线传递，所以称为复用模式。首先我们需要了解 HPI 的在 DSP 上的结构

DSP 的接口结构如图 3-4。具体每个控制线的意义如下：

(1) HD[15：0]：数据总线

(2) HCNTL[1：0](控制 HPI 访问类型)

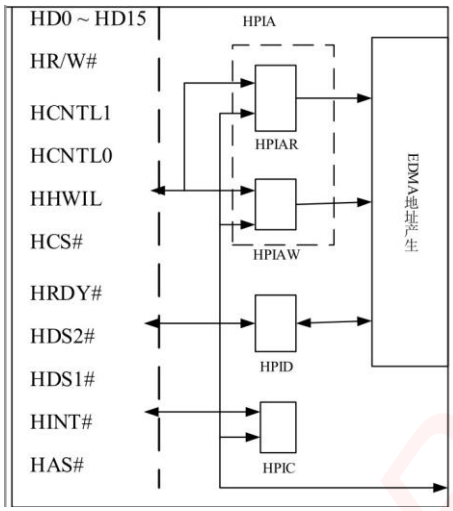

图 3-4 DSP 的 HPI 接口

如前所述，对 HPI 的访问需要通过三个寄存器，即 HPI 地址寄存器(HPIA)，HPI 数据寄存器(HPID)和 HPI 控制寄存器(HPIC)来实现。HCNTL[1：0]就是用于选择这三个寄存器的专用引脚。

(3) HHWIL ：半字指示选择

HHWIL 指示当前的为第一个或是第二个半字传输，但是 HWOB 决定了该数据是最高有效还是最低有效，在 HPI32 模式下，不使用此信号。

(4) HR/W ：读/写操作指示

(5) HRDY：输出准备好指示引脚

(6) HCS,HDS1,HDS2：选通信号

(7) HAS ：地址输入选通，这里并没有使用，所以直接拉高。

(8) HINT：HPI 输出中断

SROM 控制器的控制逻辑与 HPI 即为相似，利用 SROM 控制器，方便地实现 DSP 与 S5PV210 芯片数据交流，S5PV210 芯片的 SROM 控制器提供了 Bank0-Bank5 接口，数据线地址线复用，每一个 Bank 有一根使能控制线，这也决定了设备到底是挂载在那个 Bank 。每个 Bank 在 CPU 中根据内存分配表都有固定起始内存地址，该控制器还包括 WAIT 信号，这个信号是总线的忙碌控制，同时也支持半字访问。

这样分析来看我们可以将 HPI 映射在 SROM 的一个 Band 上，这里我们选取系统空闲 Bank，并要保证与其他 Band 进行区分。具体到连接上，设计 HPI 的 HCNTL1、HCNTL0，这里选用 ADDR2 和 ADDR3。HPI 的选通状态由 HCS、HDS

1、HDS2 三根线共同决定，产生逻辑为 HDS1 异或 HDS2 后，再与 HCS 与非。将 HPI 接口映射在主机的 Bank0 区域，则直接将 S5PV210 主控芯片的的片选信号 Xm0CSn0 接到 HCS，而将读写使能 Xm0OEn、Xm0WEn 分别接到 HDS1、HDS2。对于 HR/W 信号，由地址线 ADDR4 控制，当 ADDR4 高电平，读操作，低电平进行写操作。

本设计 HPI 采用 16 位并行数据总线，可以使用主机的地址线 ADDR11 接 HHWIL 来完成高低字节的识别：当为 0 时，表示为第一字节；为 1 时表示第二字节。要实现这个功能就需要将 32 位的数据截取成两个 16 位的数据，区分高低半字进行传输，针对接收方来说，监测 HPI 的 16 位数据总线上的是高 16 位还是低 16 位由 HBEO 和 HBEI 引脚来决定，该两个引脚最终的表现就是在 HPI 总线中的 HHWIL 引脚；收到两个 16 位数据后，按照高低半字对这 32 位数据进行合成，即可得到 32 位数据。

由于主机的 ARDY 信号和 C6416 HPI 接口的 HRDY 信号逻辑刚好相反，因此要将 HRDY 信号经过反相后再接 ARDY 信号。 HPI 的 HINT 信号由主机外部中断 EXNT1 接收，实现 HPI 与主机的中断握手。HAS 是地址选通信号线，用于主机数据和地址线的复用情况，这里就直接对其进行拉高。

我们采取如下的连接方式，如图 3-5

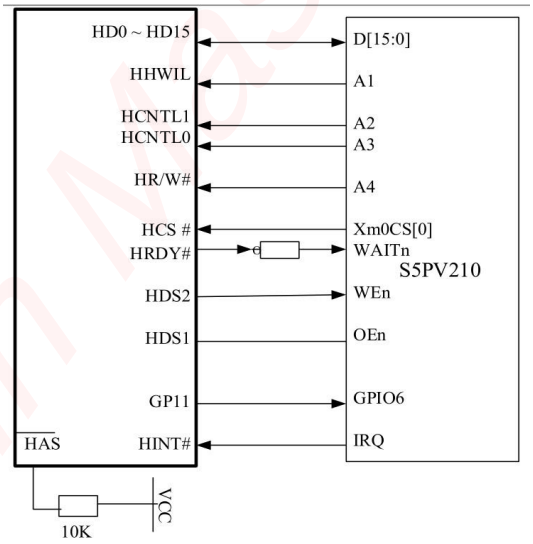

图 3-5 HPI 与 SROM 控制器的硬件连接图

3.1.3.2 与 FPGA 通信总线设计

在整个 FPGA 配置接口驱动实现过程中采用的是模拟通信协议的方式实现。所以在硬件实现的时候就要按照需求进行一定的设计，我们采用了 GPIO 控制通

信的方式，因此 FPGA 配置接口的控制引脚都与 ARM 的通用 GPIO 引脚相连接，数据端口采用一组 GPIO 组来模拟。采用一组 GPIO 端口模拟数据通道，可以在保证最大限度的提高数据传输速率同时，使整个方案的效率以及设计更加紧凑。本项目高性能 FPGA 采用的 Airtex-7 的微处理器配置方案主要有两种配置模式：Slave Serial 模式和 Slave SelectMAP 模式，分别是串行数据发送模式和并行发送模式[25]。

M[2:0]这三个引脚决定决定了 FPGA 的配置模式。M[2:0]三个引脚的状态对应的配置模式如下： M[2:0]=110，Slave Serial 模式；M[2:0]=111，Slave SelectMAP 模式。

本论文中采用 M[2:0]=110 即 Slave Serial 的数据下发的固定模式，所以这里需要对这三个引脚进行直接地上拉或者下拉。

FPGA 通电后 PROGRAM_B 会主动保持有效状态，如果需要主动重配，主机设置 PROGRAM_B 引脚有效，启动重配过程，所以这里需要 GPIO 进行 PROGRAM_B 引脚的控制，本设计中采用 GPE1[1]引脚进行控制。

在初始化配置过程中，INIT_B 处于低电平状态，初始化完成，该引脚会被置高通知主机初始化完成。这就意味着 FPGA 的配置文件的下发可以开始了，所以同样需要一个 GPIO 对该引脚进行监测。

FPGA 的配置文件接收完成之后，引脚 DONE 会被置为高阻态，同样是可以用来告知主机配置完成或者配置成功，这里需要另外一枝 GPIO 对引脚进行状态监测，同时设计 LED 指示灯连接在此引脚，继而可以明确指示 FPGA 是否配置完成。

针对配置文件的接收和发送，需要设计数据通信总线。FPGA 并没有提供直接的数据交互总线，同时由于其可编程性，可以在 ARM 端设计相应的数据发送逻辑进行协商实现。计划采用一纵 GPIO 模拟数据总线的并行数据通道，由于 S5PV210 的一组 GPIO 最大为 8 位，所以数据位宽设置为 8 位这里采用 GPE0 组 GPIO；同时总线需要有时钟，时钟也采用 GPIO 进行模拟，这里采用 GPE1[3]；总线的选通信号采用 GPE1[2]，读写控制信号线采用 GPE1[0] 。

针对以上的详细设置，具体接口结构如图所示：

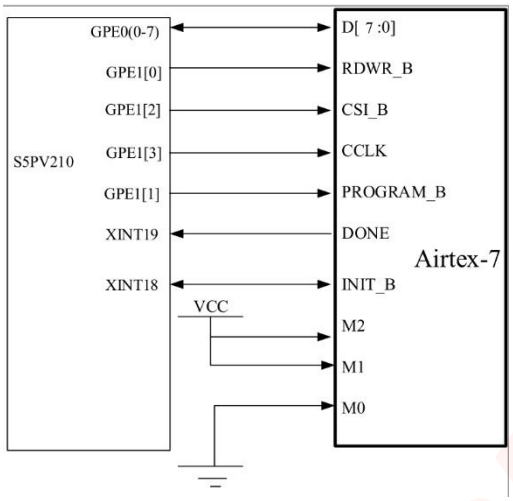

图 3-6 FPGA 与 S5PV210 的接口原理图设计

3.1.4 功能性外设设计

3.1.4.1 温度传感器设计

本项目中温度传感器采 ds18b20，是常用的温度传感器，具有体积小，硬件开销低，抗干扰能力强，精度高的特点[26]。它的工作电平在 3V5.5V 的区间，静态功耗下电流在 3uA 以下，温度测量范围为-55°C 至 125°C。

温度传感器可编程的分辨率为 9-12 位，温度转换为 12 位数字格式最大值为 750 毫秒，DS18B20 可以不需接外接电源而直接从数据线获取电量，也可以设计外接电源。本论文设计了外接电源，相对来说更稳定，兼容性更好，其只需三根线进行设计连接。本设计中机箱的温度传感器集成在嵌入式系统的设计板上，而其他半字的温度检测装置是可以通过相应的接口传输过来，这里只设计了一个插口。温度传感器的数据线最终输入进入 ARM 的 GPD1_2 的 GPIO 作为 I/O 与 ARM 主机进行数据交互。

这两部分设计如下：

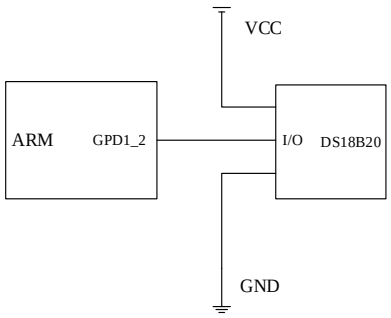

图 3-7 温度传感器设计

3.1.4.2 网络子系统设计

网络子系统网卡选取 DM9000，DM9000 应用广泛，性价比极高。其具有 10/100M 自使用的物理层结构，同时具有 4K 双字节的 SRAM 缓存空间，支持 8 位，16 位和 32 为的接口访问[27]。S5PV210 用于 8 个 Bank 的 SROM 控制器，本项目中设计将其挂载在 Bank1 上，值得注意的是 HPI 也被映射在 SROM 控制器上，其被映射至 SROM 的 Bank0，具体到内核和驱动设计时，需要加以区分，以免驱动设计出现问题。

从硬件上来说与介质无关的功能引脚定义如图 3-9。

引脚号	引脚名	引脚类型	功能描述
37	LINK_I	I	外部介质无关接口器件连接状态
38、39、40、41	RXD [3:0]	I	外部介质无关接口接收数据 4 位 半字节输入(同步于接收时钟)
43	CRS	I/O	外部介质无关接口的载波检测
44	COL	I/O	外部介质无关接口的冲突检测，输出到外部设备
45	RX_DV	I	外部介质无关接口数据有效信号
46	RX_ER	I	外部介质无关接口接收错误
47	RX_CLK	I	外部介质无关接口接收时钟
49	TX_CLK	I/O	外部介质无关接口发送时钟
50～53	TXD[3:0]	O	外部介质无关接口发送数据低 4 位输出 TXD[2:0]决定内部存储空间基址：TXD [2:0]) * 10H + 300H
54	MDIO	I/O	外部介质无关接口串行数据通信
57	MDC	O	外部介质无关串行数据通信口时钟，且与中断引脚有关 该引脚高电平时候，中断引脚低电平有效；否则高有效

图 3-9 介质无关接口引脚

这里所有的引脚在 DM9000 中都内含 60K 的上拉电阻。具体到 DM9000 的处理器引脚意义如下图 3-10 所示： 这里涉及到 CMD 引脚的控制，也决定了是访问其地址还是数据。首先将其映射到 Band1 上后，其在 S5PV210 中以固定的起始物理地址，由于 CMD 决定了其地址还是数据的访问，这里将其映射到不同的地址偏移量上，这里将其设置为 ADDR5，也就意味着其物理地址的偏移量反映在数字上是 25^，也就是 32。同时设置其中断为外部中断 9。这些都是后续的驱动开发和移植需要严格注意的问题。

引脚号	引脚名	引脚类型	功能描述
1	IOR#	I	处理器读命令 低电平有效,极性能够被 EEPROM 修改,详细请参考对 EEPROM 内容的描述
2	IOW#	I	处理器写命令 低电平有效，同样能修改极性
3	AEN#	I	芯片选择，低电平有效
4	IOWAIT	O	处理器命令就绪 当上一指令没有结束，该引脚电平拉低表示当前指令需要等待
14	RST	I	硬件复位信号，高电平有效复位
1~6 82~89	SD0~15	I/O	0~15 位的数据地址复用总线,由 CMD 引脚决定当期访问类型
93~98	SA4~9	I	地址线 4~9；仅作芯片选择信号 SA4~9：TXD0~2 ，011)被选中
92	CMD	I	访问类型 高电平是访问数据端口；低电平是访问地址端口
91	IO16	O	字命令标志，默认低电平有效 当访问外部数据存储器是字或双字宽度时，被置位
100	INT	O	中断请求信号 高电平有效，极性能修改
37~53 56	SD31~16	I/O	双字模式，高 16 位数据引脚
57	IO32	O	双字命令标志，默认低电平有效

图 3-10 DM9000 处理器接口引脚

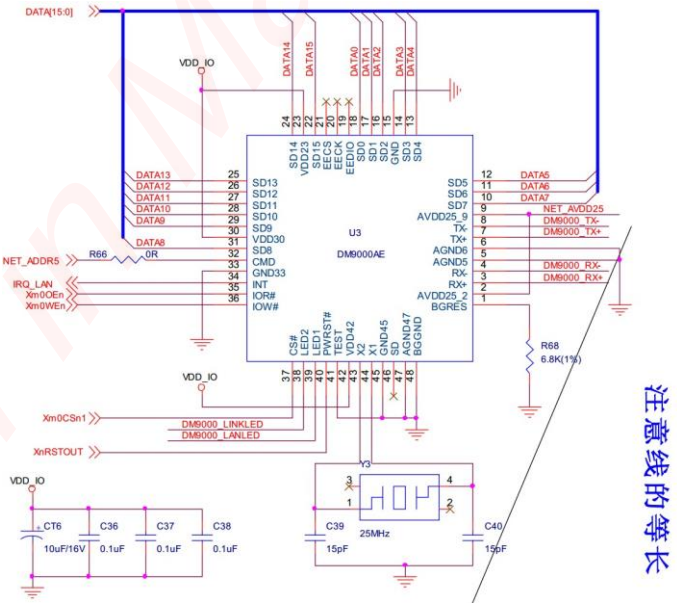

图 3-11 DM9000 电路设计

3.1.4.3 射频控制通道电路设计

嵌入式主控芯片与射频接收前端的 FPGA 之间的交互数据量比较小，但是对响应速度有一定的要求，这里采用 SPI 总线对其进行控制。

SPI 总线串行外设接口（Serial Peripheral Interface）的缩写，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，同时如果设计了该总线，该总线的复用也非常方便，只需额外再占用一个引脚作为片选线即可。所有基于 SPI 的设备共有的四根线分别是 SDI（数据输入）、SDO（数据输出）、SCLK（时钟）、CS（片选）[28]。

S5PV210 提供 2 路高速 SPI，而 SPI 也可以通过 GPIO 自行模式实现。这里直接选用芯片集成的 SPI 控制器，只用设计定义的接口即可，难点是 SPI 的驱动设计与实现。针对其在核心板上的连接设计如下，为了方便调试和连接，我们将其设计为插孔的形式。

3.1.5 扩展接口设计

3.1.5.1 USB 相关设计

S5PV210 芯片提供 1 路 USB HOST 2.0 的输出，在实际的频谱检测设备中有多处需要用到 USB 的接口：首先触控屏幕的输入接口采用 USB 的接口，第二键盘或者鼠标设备也同样需要 USB 的接口输入，第三可移动存储设备的接入也是通过 USB 接口与主控设备进行交互，只有一路 USB 的输入显然是不足的，所以需要采用集线器对其进行扩展。

本论文采用 AU9254A21 集线器，该集线器可实现一路输入扩展为 4 路输出，该芯片兼容 Universal Serial Bus Version 1.1 的规范，而且支持总线供电和自供电两种供电模式，其功耗很低；其内建 3.3V 电压校准器，支持 5V 的运行电压，最大运行频率在 12Mhz。满足频谱监测设备的 USB 扩展要求。具体的集线器电路设计如下图 3-8 所示其输入为 USB DP 和 USB DM，四路输出分别为 USB1，USB2，USB3，USB4，其中 USB4 用于触控屏幕的触控 USB 输入，为了方便调试，设计了支持热插拔接口。

另外还需要设计 USB OTG 模块的电路，OTG 是近几年才发展起来的额技术，随着 PDA 以及个人数码设备的小型化和普及化，各个设备之前不希望受限于传统 USB 一定需要 HOST 才能进行数据交互，OTG 正是提供了这样一种便捷的功能，它本身集成有电源管理的功能，模糊了主机和从机的绝对区别，使其可以作为外设也可作为主机进行数据交互[29]。频谱监测设备的嵌入式系统可通过 OTG

极为方便地与其他主机进行数据交互，其他主机可以主动通过 OTG 进行数据的读写，这样就提高了该嵌入式系统的可移植性同时也提供了另外一种数据交互的方式。由于 S5PV210 提供一路 OTG 输出，核心内容是其在操作系统中的驱动实现。

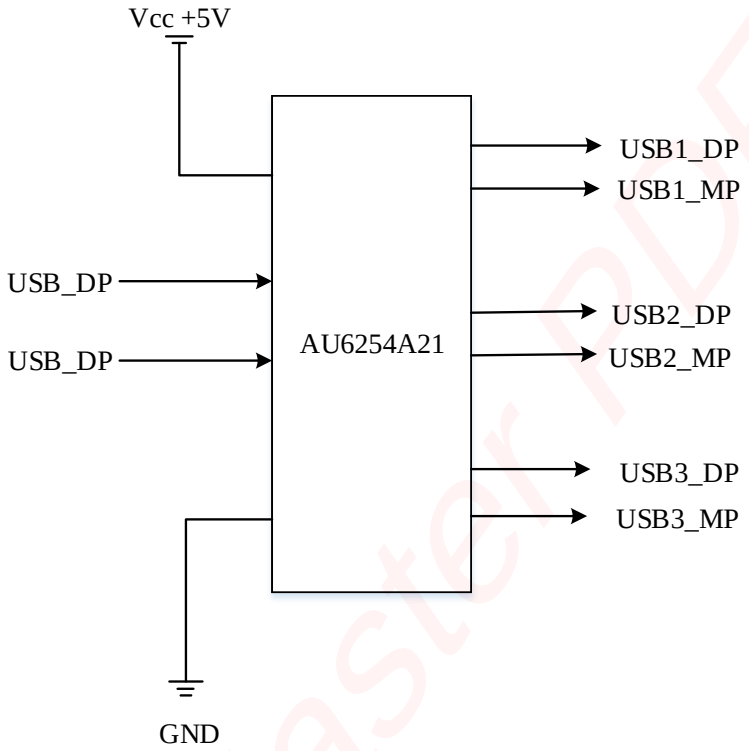

图 3-8 USB 集线器设计框图

3.1.5.2 TFCARD 卡设计

本项目的频谱检测设备中还涉及到外界存储器 TFCARD 卡的设计，S5PV210 本省提供了 MMC 卡的数据接口，这里的设计难点是在软件系统中的驱动实现。

3.1.5.3 UART 设计

S5PV210 提供了 4 路 UART 输出，针对本项目，串口 0 设计用来进行调试所用的 debug 口，串口 1 设计用来外接 GPS 芯片进行位置获取；串口 2 用于外接键盘或者鼠标；外加一个闲置串口以防其他串口出现状况以及用于功能性的补充等。具体的设计如下。

3.2 嵌入式操作系统设计

通常情况下，操作系统重要的组成部分操作系统内核，文件操作接口，网络协议，可视化界面等，该部分需要负责嵌入式系统资源分配，以及相应的任务调度控制协调，包括多任务的并发执行，以及人机交互相关的功能等，系统可以根据相关设计的硬件通过模块的添加删除来实现功能的自由定制。

目前在嵌入式系统领域广泛应用的操作系统有多中供我们选择，嵌入式 Linux、Windows Embedded(Windows Mobile)、VxWorks 等，以及我们广泛接触的的 Android、iOS 等。

VxWorks 的实时操作系统。该系统具有良好稳定性和安全性，其内核效率高[30]，用户开发环境也成熟友好，在嵌入式领域一直占有一席之地。然而此系统由于是商业公司研发，价格极其昂贵，包括其开发环境和操作系统都是需要授权的，在十万人民币左右才可搭建可用的开发环境。每一个额外支持的应用需要另加授权费。而且系统并不是开源的，风河只提供编译后的代码[30]。由于系统的专用性，需要掌握其特殊的技术才可进行开发，这就导致软件的开发和维护成本相对比昂贵。

Windows CE 操作系统继承了 Windows 的优良特性，多线程、完整优先权、多任务特性优异[31]，但是内核相对比较大，基本内核就需要 200KB 的存储空间。同时 Windows CE 也没有开放源代码，对 Windows CE 的定制性也就变得比较差，而且 Windows CE 的效率和功耗上比起其他操作系统没有优势，应用程序占用内存过大，而且开发出的应用程序往往也体积庞大，Windows CE 同时也需要一定的授权费用。

嵌入式 Linux 操作系统，遵循 GPL 协议源代码公开，可供开发者自由根据需要修改，以满足特定场景的应用[32]。该操作系统遵从 GPL 协议，无须许可证费用。由于 Linux 操作系统起源于互联网，有大量的免费开源软件，经过移植就可以用于自己的开发。同时 Linux 的开发也有很多遵从 GPL 协议，开放源代码免费的优秀的开发工具。同时由于互联网共享的精神，该系统有庞大的开发人员群体，所以软件的开发和维护成本很低。该系统具有优秀的网络功能，可扩展性非常强，同时也具有极佳的稳定性。内核精悍，可供开发者自由精简，所以运行所需资源少，是很多场景下嵌入式系统的最好选择。

嵌入式 Linux 和普通 Linux 本质上是相同或者说类似的，嵌入式 Linux 大都可良好支持 PC 上的硬件。由于源码开放，所以包括操作系统的移植，包括开发驱动程序难度就降低了。在移动电话、消费性电子产品以及航空航天等相关领域中的嵌入式操作系统，现如今最常见的大多都是基于 Linux 为基础的，比如广泛应用的 IOS，以及 Android OS 都是以 Linux 为基础的嵌入式操作系统，随着 Linu

x 的流行，现如今市面上 80%以上的嵌入式设备均采用 Linux 系统。

综合以上流行的嵌入式系统来看，在频谱监测设备上，我们需要的是成本低廉，可靠性强，可扩展性强的操作系统，所以嵌入式系统采用 Linux 也是不二之选。在确定操作系统内核后，将嵌入式系统软件分为四个层次进行设计。

3.2.1 Bootloader（引导加载程序）设计

引导加载程序运行在嵌入式操作系统之前，负责初始化硬件设备、建立内存空间等各项操作，提供操作系统可运行的环境，这也是一种分层结构的思想，可以保证操作系统良好的移植性，而不必理会最底层的硬件初始化相关内容。

固态存储设备(比如：ROM、EEPROM 或 FLASH 等)是处理器构建的嵌入式系统的必须部分，这些存储系统被映射到这个预先安排的地址上。因此在系统加电后，CPU 将首先执行存储在这些固态存储设备中的引导加载程序。

通常，Bootloader 的设计与硬件相关性很强，其选择也依赖于处理器体系结构，因为嵌入式中硬件多种多样，编写一个通用的 Bootloader 非常困难[33]。现有的研究只能综合出一些 Bootloader 通用的参数和概念，开发者通过这些参数和概念实现特定的功能。

针对本项目，我们为 S5PV210 选用 U-boot 1.5.1 作为其 Bootloader，除了与处理器的体系结构相关，U-boot 与嵌入式系统设计班的配置也息息相关。两块相同的 CPU 的设计板，但是外设的设置不同，如果想让同一个 U-boot 运行在这两块板子上，通常也是不现实的，还是需要修改的匹配的。

针对 S5PV210 以及本论文中的相关器件和电路的选型及设计，该版本的 U-boot 的内核的启动参数、内核映像和根文件系统映像的固态存储设备的存储分配结构图如下图 3-12 所示，在 U-boot 的启动阶段，这里全部设计在该引导启动程序中的起始阶段的一段初始化程序中，这里进行如下的设计。

类别	起始和中止地址	空间大小
U-boot	0x0~0100000	1024KB
Kernel	0x100000~0x600000	5120KB
Logo	0x600000~0x900000	3072KB
System	0x900000~END	1015MB

图 3-12 本嵌入式设备中 S5PV210 的固态存储分配表

我们也需要在一定程度上控制设置相应的 U-boot 参数，这些通信往往都是通过 U-boot 启动后立即通过串口简历主机和目标机之间的链接。当然随着 U-boot 的功能越来也完善和强大，U-boot 在启动时也会初始化网卡，加载网卡的驱动程

序，这就意味着主机可以通过网口与目标机进行通信，网口的速度更快，操作起来也一定程度上更方便，甚至可以通过网卡进行内核的加载，但是由于不同的设计板上集成的网卡类型可能不尽相同，所以 U-boot 如果想要通过网卡进行交互还需要根据设计板上的网卡匹配相应的驱动程序。

3.2.2 Linux 内核移植设计

操作系统的组成部分包括有内核，设备驱动，命令行 shell 等等。用户的界面只是操作系统体现给用户的交互现象，内核才是操作系统的本质所在。操作系统的正常运行必须依赖内核提供的的服务和接口，内核才是操作系统灵魂。内核实现了操作系统的机制，比如中断、任务调度等等。对于现代操作系统内核运行与用户应用程序是分开的，有独立的内存空间，又有直接的硬件控制权限。而应用程序是运行在用户空间，只能使用被规划的系统资源，不能直接访问硬件。图 3-13 说明了 Linux 应用程序和内核以及底层硬件的层次和调用关系。

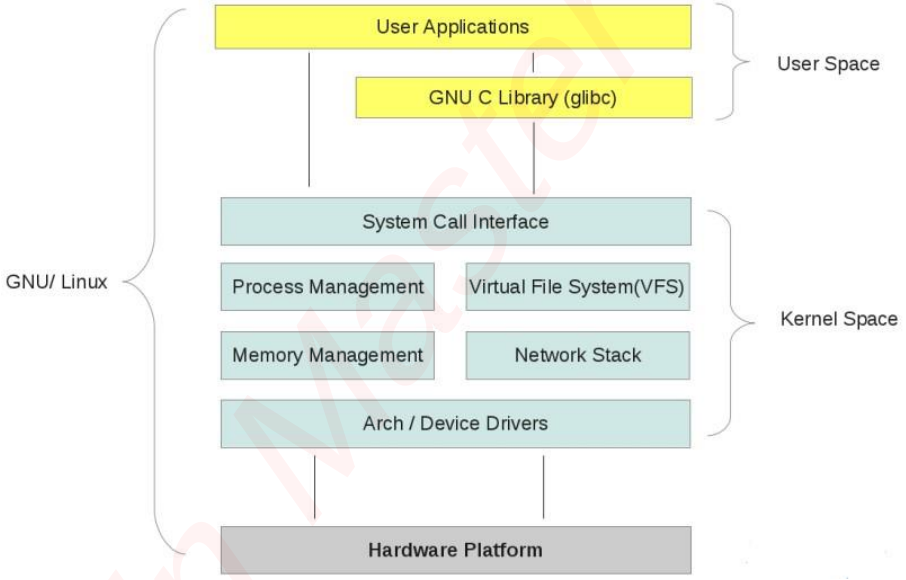

图 3-13 Linux 应用程序，内核，硬件层次关系

应用程序运行在用户空间，通过直接地系统调用接口（System Call Interface），或者调用库函数（GUN C Library），函数库再调用系统调用接口进入内核。

现有的所有处理器的体系结构，几乎都提供了中断机制。如果硬件想要和操作系统进行通信时，需要由硬件发出异步中断来通知处理器，内核将会暂停来源的执行工作。中断通过中断号来区分不同的中断，不同的中断号对应不同的中断服务程序，相应的中断服务程序响应这个中断。这样内核在中断上下文就完成了用户空间和底层硬件空间的交互。

本论文中嵌入式系统平台内核采用 Linux 2.6， Linux 2.6 在原有基础上有了相当大的改进。新特性针对性能、扩展能力、吞吐能力，开启对 SMP 和 NUMA 的支持。

Linux 内核工作于虚拟内存模式：每一个虚拟页对应一个相应的系统内存的物理页[35]。针对驱动程序移植，2.6 内核提供了更完整的体系，首先，相对于 2.4 来说，改进了内核编译系统，从而获得更快的编译速度。加入了改进的图形化工具：make xconfig（需要 Qt 库）和 make gconfig（需要 GTK 库）。

Linux 2.6 内核模块加载器，使用内核编译机制，产生一个 .ko(内核目标文件，kernel object)模块目标文件而不是一个 .o 模块目标文件。内核编译系统首先编译这些模块，并将其连接成为 vermagic.o。这一过程在目标模块创建了一个特定部分，以记录使用的编译器版本号，内核版本号，是否使用内核抢占等信息。

makefile 文件使用内核编译机制来编译模块，以前的 Makefile 不再被使用，按照新制定的规则进行 Makefile 的开发。

2.6 内核针对硬件设备拥有统一的设备模型，这个设备模型通过维持大量的数据结构囊括了几乎所有的设备结构和系统[37]。由于这个特性，根据该统一的设备模型进行设计，包括根据其所连接的总线类型，特定情形下设备的电源状态，系统清楚设备的驱动程序，并清楚哪些设备受其控制，设备在系统中的类别描述（类别包括磁盘，分区等等）。

针对 linux2.6 的新特性以及驱动新模型，在进行相应的开发时需要依靠其进行合理地利用。

3.2.3 根文件系统制作

文件系统是操作系统的主要组件之一，它表明了文在在存储设备上组织文件的方法，也是可与用户有直接交互的部分。文件系统通常是由文件系统的操作接口，文件操作和管理软件集合，文件对象和属性在嵌入式 Linux 下，MTD 建立了一个抽象接口，这个接口用来负责底层硬件和文件系统的交互，也就是说 MTD 驱动层是 FLASH 存储文件系统的基础，这一点我们可以从下面的 Linux 文件系统结构图可以看到。MTD 驱动程序设计的初衷就是为非易失性存储器提供更好的支持、管理和提供更方便的基于扇区的擦除、读/写操作接口。

针对闪存芯片，现已有多种文件系统对 NOR 或者 NAND FLASH 芯片提供了良好的支持，如 jffs2，Yaffs2 等等[38]。针对本项目的频谱监测设备中的根文件系统系统，需要从这两中文件系统中进行梳理和选择：

1.Jffs2: 日志闪存文件系统版本 2 (Journalling Flash FileSystem v2)

JFFS 文件系统最早是由瑞典 Axis Communications 公司开发的文件系统。JF FS2 是 Red Hat 公司在 JFFS 基础上的升级开发，最初只针对自身开发的 eCos 嵌入式系统，JFFS2 同时也可以被 Linux, uCLinux 等多种系统中[39]。其优点有：支持数据压缩、日志文件基于哈希表以及提供崩溃/掉电安全保护等。最大的缺点就是文件系统快满时，该文件系统会调用垃圾收集机制，其运行速度降低非常多。

Jffs2 文件系统的日志基于哈希表，那么在其挂载时会花相当的时间扫描整个存储设备建立日志节点，容量越大耗时越久，这就决定了该文件系统不适合额用于大容量的闪存。

2.Yaffs yaffs/yaffs2 是是一种日志型文件系统，针对 NAND 闪存开发。速度更快，挂载迅速，资源占用少[39]。对不同的平台支持也很广泛，比如其支持 Linux 以及 WinCE，ThreadX 等等。

Yaffs2 文件系统对 NAND Flash 支持非常好，其日志文件系统的设计的初衷是追踪文件系统变化，与文件内容变化无关，而且自带有文件操作接口，同时也兼容 MTD 与 VFS 提供的文件操作接口；该文件系统提供了掉电保护机制，如果在操作文件的过程中发生意外，数据可以得到有效保护。该文件系统最大支持 2 KB 的大页内存，垃圾回收速度和读写速率都有优秀的表现。

本平台硬件存储采用 1GB 的 NAND 闪存芯片，且对文件的读取和操作较为频繁，综合以上的分析，YAFFS2 文件系统性能优越且易于移植，加上其优良的特性，所以采用 YAFFS2 文件系统。

Linux 初始内核是不支持 Yaffs2 文件系统的，需要进行 yaff2 补丁安装才能提供支持。由于 NAND FLASH 出现位反转的可能性比较大，所以我们在对文件进行操作的时候需要进行 ECC 错误检查，并在出错的时候进行数据恢复。在设计 YAFFS2 文件系统的时候，利用 YAFFS2 的前 528B 数据，其中后 16B 包含的 OOB 数据进行错误检测和坏块处理，使用 mkyaffs2image 工具制作 yafs2 文件系统的镜像，然后对 FLASH 进行相应的烧写。

3.3 内核驱动设计思想

软件和硬件的开发者都寄希望于两方可以互相独立，软件工程师也不用处处都需要关心硬件上的时序和逻辑问题，而硬件工程师也不需要关心软件这方面的功能化定制。例如，本项目中应用软件工程师在调用 HPI 的操作接口的时候，完全不需知晓 DSP 所提供的中断，以及 DSP 硬件寄存器和存储器的物理存储，只需要调用读写函数的接口，就可以直接得到相应的数据信息，这些数据信息就包

含了上层软件所需的所有有用的信息；当软件工程师希望将图像输出到屏幕的时候，也是直接调用绘图控件，不必理会图像是怎么样一个像素一个像素地控制，怎样一行一行第扫描。

这就意味着软件工程师看到的只有软件的接口，所谓的硬件功能对于软件来说也只是一个软件接口函数。那么如何将硬件编程软件工程师可以直接实用的软件接口，这就是驱动的根本精髓。

驱动程序就是驱使硬件行动的软件程序，形象地被称为硬件和软件之间的“桥梁”[37]。驱动需要操作控制硬件，所以驱动程序要运行在内核空间，而内核是可以与硬件进行交互，这也就是说驱动也可以直接操作硬件。驱动程序依照硬件的操作逻辑，中断控制等等实现对硬件的通信，而用户空间的软件可以与驱动进行交互，间接地就进行了硬件上的操作。

显而易见，驱动程序担当了硬件和应用软件之间联络人的角色，它提供的编程接口可以让应用软件方便地进行调用。在逻辑运行程序的时候，也需要对硬件的功能和接口进行封装，比如串口的操作 SerialSend()、SerialRecv()；对 Flash 的操作 FlashWrite()、FlashRead()等。那么在处理器运行了操作系统的情况下，操作系统所在设计之初便需要协定这个操作系统下的驱动结构。驱动工程师需要按照驱动设计的协定进行实现，当然这也很好地解决了不同平台相同操作系统的驱动开发的移植问题，只需要了解这个操作系统的驱动体系，那么即使换了平台，驱动整体的架构是不会变的，使得驱动的统一度大大提升。

针对本项目中采用的 Linux 操作系统，该操作系统将设备规划为三大类: 以字节为传输基本单位只能顺序访问的字符设备驱动类型，以块为传输基本单位可以随机传输的块设备驱动类型和通过套接字进行数据包传输的网络设备驱动类型[36]。网络设备的访问是调用套接字的方式进行的，而针对字符设备和块设备，Linux 设计了一种虚拟文件系统（VFS）。VFS 可以很好地完成在不同存储介质上不同文件系统下的文件操作。

Linux 的设计思想就是所有的外围设备在操作系统送看来都是一个标准的文件，那么一个标准的文件可以进行的所有操作，在外围设备上都有相应的映射，而这些映射就是外围硬件的不同功能[40]。应用程序看到的只有像 open()、read()、write() 这样的函数接口，在希望操作硬件的时候就像打开一个文件操作一个文件一样，不需要因为不同的存储介质而去调用不同的 API 进行驱动的操作。

这就是意味着字符驱动和块设备驱动的作用就是完成 VFS 系统调用的对接，在内核空间完成对于硬件的操作。

我们知道，解决一个问题的方法一般是先了解这个事件本身，然后分析解决

这个事件需要哪些步骤，然后按照步骤一步一步将其解决。这也是程序语言最初的最原始的设计思想。这样的思想也占据了 Linux 编程的绝大部分，包括 Linux 内核中是被广泛应用的。

本身针对驱动来说，我们需要在驱动中去实现硬件的相关操作，比如一个硬件上的功能实现，像本项目中涉及到的 FPGA 的配置，我们需要按照 FPGA 的控制逻辑，比如按照时序拉低拉高某些引脚的电平，使 FPGA 获得重配初始化的信息，这个过程的实现必然要在驱动中去实现这个过程一一去解决相关的问题，这也是面向过程的典型所在。

因为 C 语言以及驱动程序设计本身对面向对象思想的支持并不良好，但是这并不意味着我们在进行 C 语言的相关开发时不可以利用面向对象的思想进行程序设计。

3.3.1 内核中面向对象设计思想

在软硬件环境繁复多样的状态下，面向对象程序设计通过重视代码的可复用性增强，软件的维护和更新也就变得相对容易。Smalltalk 语言在面向对象上提供的设计思想也成了后续面向对象设计语言的奠基，它的理念就是将程序和数据封装其中在对象中，进而使得软件的复用性和灵活性增强。面向对象，其实从深层意义上说是程序开发方式，不只是程序设计方法。

Linux 内核完全由 C 语言和汇编语言，显然不是面向对象的语言，但驱动设计的时候，同类的设备在 Linux 下都有一个相应的框架，这个框架的核心实现了此类设备操作的一些通用的接口。如果特别一点的设备，属于这个类，但是并不想使用这个框架内的核心函数，同样也可以自己编写实现类似的功能函数。例如在此项目中 HPI 的操作函数中，如果需要向 DSP 的某个寄存器写或者读的化，可以按照字符型设备的写或者读函数，指定要写的内容和映射的虚拟地址，通过 read 或者 write 可以直接操作；当然，我们不像如此做的化，也可以自己建立一个函数，直接通过 IO 进行操作，可以将其规划为一个函数。

Linux 设备模型其实归纳起来就是一个较为繁杂的数据结构。例如如图 3-18，这是一个简化的 USB 驱动模型[41]。一个含有 USB 芯片的鼠标驱动结构，显示鼠标连接到系统的过程。总线的树状结构可以是系统追踪到那些设备连接到这个总线上，而在 classes 的分支已经确定了这类设备的功能，不用去关心设备的连接方式。

任意的设备模型也有许多的节点，这也就意味着你不可能把一个这么复杂的数据结构抽象成一个驱动。看到 buses 和 classes，就可以看到面向对象在驱动编

写的体现。这里将所有的实现统一特性譬如鼠标特性的设备归为一个类，也就是一个 class，至于这些鼠标是通过什么总线连接上来的，class 并不关心，只认为这是一个鼠标，那么鼠标具有什么特性，在这个类的子树里已经有明确的规定，上层应用在调用鼠标的时候只用调用鼠标调用的通用接口就可以了。那么对于应用程序来说，鼠标本身和它的操作逻辑就被抽象为一个基本单元，也就是一个对象。

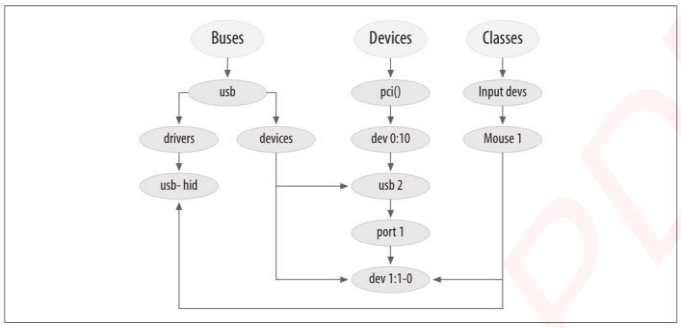

图 3-14 USB 设备模型

在内核的实际操作中，已注册 kobject 对象反映在 sysfs 文件系统中，就是一个目录。这些与面向对象语言中的类是共通的，而且 kobject 一般都会被嵌入到其他的结构（面向对象的语言中的容器）中。比如 bus，devices，drivers 等等，这些“容器”通过 kobject 链接起来，组成某种树状结构[42]。

kobject 的是构成设备模型的基本结构，设计初衷是用来做引用计数结构，但其职责随着时间的推移超出了它原来的领域。现在由 kobject 的结构支持处理的任务包括：通常情况下，在创建内核对象时，不可得到它的存在时长。当在内核没有代码保持对给定对象的引用，该对象已完成了它的使用寿命，可以被消除掉了，而 kobject 便可以完成这种引用计数的功能；2）kobject 也完成了 sysfs 的文件表示，毕竟每一个对象都有也 kobject，而已注册 kobject 对象反映在 sysfs 文件系统中，就是一个目录。3）设备模型一个相当繁杂的复杂数据结构制成，而这些复杂的数据结构之间也是多层次，内部也有极为复杂的关联，而这些关联就是通过 kobject 这种结构联系在一起。4）kobject 的子系统处理该通知用户空间有关系统上硬件事件包括硬件的交互中断等等时间的发生状态，也就完成了热插拔事件的处理。

有关 struct kobject 的类型的定义，我们可以在 linux / kobject.h 看到，这个文件中不仅定义了 kobjcet 的结构，同时也定义了一系列 kobject 的操作方法。

kobject 的结构经常被用来一起对象通过指针父指针和 ksets 建立到某一个分层结构的连接，一个 kobject 的父字段是一个指向另一 kobject，而这个 kobject 代

表了层次结构中的一下一层。例如，一个 kobject 表示 USB 设备，其父指针可能链接到其所在 HUB 的 kobject。通过父指针可以在 sysfs 层次结构以定位该对象。kset 在建立和加入系统之后，sysfs 系统中就会产生对应的目录，其中内含的 kobject 都后再 sysfs 中有有明确的对应关系。

3.3.2 总线、设备和驱动的分层结构设计

到目前为止，我们已经看到了底层的数据结构中的面向对象的设计思想，进入更高层次的 Linux 设备模型。在这个层次的操作上来说，驱动开发者很少需要添加一个新的总线类型。但这也正式驱动分层结构的体现，也正是本项目驱动设计遵循的最终思想[43]。开发一个驱动，遵从分层结构的思想，将整个驱动通过分层的结构体系注册到内核中去。

驱动最内部的三个组成部分为：总线、设备和驱动。操作系统启动时，在板极信息信息文件中会初始化总线设备，为每一个被检测到连接在这个总线上的设备建立 struct device 结构体；相应的驱动程序也有一个 struct device driver 结构体。内核中也会建立对应这个设备和驱动的链表，将设备加入设备 struct devices 链表，驱动插入 struct drivers 链表。这样总线通过索引这两个链表，就可以所引到每个设备和驱动。

驱动程序在加载的时候会执行一段初始化程序，这部分初始化程序中会调用注册函数 driver register 向内核注册 strcut dev driver，系统的总线会在设备的 devices 连表中依次查询是否有未关联驱动的设备，如果有的话，接着会监测该设备的标识，如果其中指定的特征与 device driver 结构体的标识相符合，内核函数 device bind driver 即会将两个结构体进行关联，关联的方法就是 struct device 中的 device driver 指针指向这个该驱动结构体，同时将这个 struct device driver 的 diver 也会将次 device 加入到这个驱动的 device 链表中，这样就完成了设备和驱动的双向关联。总线探测到设备之后会在驱动目录下主动创建相应的设备文件。操作系统像内核注册某个设备时，会主动寻找与该设备匹配的驱动；相反的，在系统安装某个驱动模块的时候，也会通过驱动匹配相应的，这个匹配过程是由总线来完成。

Linux 操作系统下设备和驱动挂载在同一种总线上。设备与驱动的关联会通过总线操作方法 match()进行匹配，驱动往总线上挂载时，会与所有的设备进行匹配；而设备往总线上挂载的时候，这个总线上所有的驱动也同样会进行配对，如多配对成功，驱动中的操作方法 probe()方法会被调用。在开发驱动程序时只需考

虑相应的设备和驱动程序，不需要靠考虑设计总线的类型，毕竟 Linux 中已经有足够的总线类型，不需要在开发定义新的总线类型。

3.3.3 控制器与外设驱动隔离设计

Linux 驱动开发过程中，还有隔离的思想也是一个非常有创造力的思想。简单来说，比如我们假设我们要通过 I2C 总线访问某个 I2C 的设备，要通过访问 CPU S5PV210 上的 I2C 控制器的寄存器，进而通过控制器再访问 I2C 外设[45]。最直观的驱动编写方法就是，先控制器的寄存器中写控制命令，再写数据，最后检查状态。

这每一步操作都需要一个与 CPU 直接相关的 IO 操作函数。如果存在另外一个 CPU 上，同样一个 I2C 设备，就意味着需要另外一个 CPU 相关的函数，这就使得驱动的通用性大大变差。这个时候，引入控制器和外设驱动分离的设计思想。如图 3-15 所示，假设存在主控制器 1、2、3 和外设 a、b、c。按照控制器和外设不分离的思想，那么需要 3*3=9 个驱动的开发才能完成访问。主控制的多样化，就意味着如果没有隔离机制，驱动开发就极为复杂。隔离机制将控制器驱动和外设的驱动分开来，外设不需要关心主机，主机也不需要关心外设的类型，外设的访问只是通过通用的 API 进行，主机和外设之间可以进行任意的匹配，而不需要修改驱动。例如本项目中的 SPI 驱动就采用了这样的隔离思想：所有的 I2C 设备的驱动操作函数在 Linux 下，无论主控制器是什么，操作函数是一样的。这样就简化了 SPI 驱动的设计。

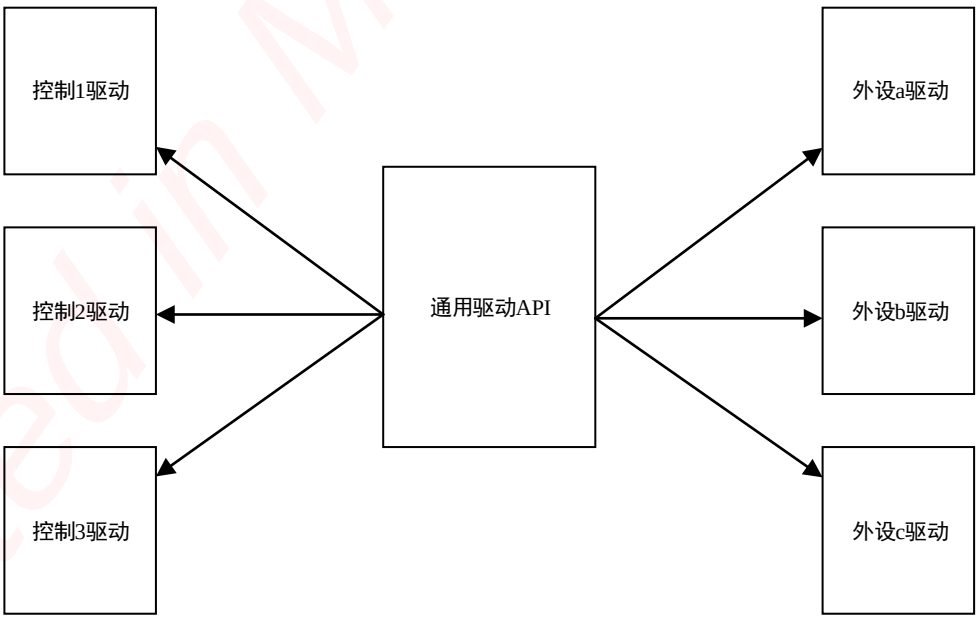

图 3-15 控制器与外设分离

3.4 本章小结

本章在根据前面一章提出的需求分析，首先根据其性能需求选定了嵌入式系统的核心为 S5PV210 处理器，同时选定 512MB 的内存，以此为基础进行了嵌入式系统的其余硬件设计，特别根据嵌入式系统与中频化数字信号处理单元的 DSP 和 FPGA 的通信总线进行了详细的设计和分析，然后根据功能需求设计了 USB 接口，网卡电路，射频控制接口，TF 卡卡槽，UART 接口，以及温度传感器接口等；之后在嵌入式系统设计实现中最重要的部分就是要设计相应的设备驱动，首先分析了 Linux 的驱动设计体系，然后根据 Linux 的驱动体系确定了设计理念，提炼出内核中的面向对象思想和分层隔离的思想，接下来会将这种设计思想应用到后续的驱动设计与实现中。

第四章 基于分层隔离思想的频谱监测设备驱动核心设计实现

第三章设计了频谱监测设备的嵌入式系统的硬件框架和软件设计框架，并确定了驱动的设计思想，根据 Linux 的驱动框架体系，接下来就需要在 Linux 操作系统下实现相应的驱动程序。这里涉及到的有 FPGA 的重配驱动以及其配置应用程序的实现，其二就是 DSP 的 HPI 驱动设计与实现，以及 S5PV210 与频率合成模块的射频接收前端的 FPGA 的驱动程序设计与实现，以及频谱监测设备的 LCD 显示驱动程序的实现，并在此基础上需要实现触控屏的功能。

4.1 FPGA 的配置启动驱动以及重配应用程序封装

FPGA（Field－Programmable Gate Array），又名现场可编程门阵列，它可以以硬件描述语言进行定制化的电路设计和综合布局，之后将生成的可烧写配置文件快速的烧录至 FPGA，进行各个场景的应用与测试。从配置文件的存放来说，

FPGA 的重配方法有静态重配和动态重配两种配制方法。静态重配试讲可重配的程序放在 FPGA 的外部存储器中，FPGA 固定地从某个非易失性存储器获得配置文件，如果要重配需要修改或擦数存储器中的文件；动态重配是指 FPGA 的运行配置时可以在启动的时候由控制设备主动下发，这意味着主控系统可以随时重新下发不同类型的配置文件。

4.1.1 FPGA 配置流程

A7 的配置过程基本如图 4-1 所示上电流程，这个过程中需要调配 FPGA 的各个引脚，才能在正确的时候按照相应的控制时序来进行。

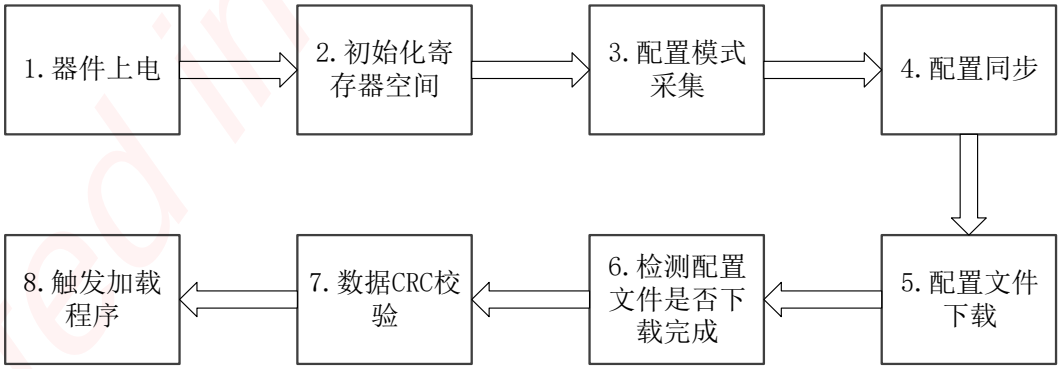

图 4-1 FPGA 的上电启动流程

重配：FPGA 通电后 PROGRAM B 会主动保持有效状态，如果需要主动重配，

主机设置 PROGRAM B 引脚有效，启动重配过程。初始化：重配引脚有效，FPGA 进行内存初始化操作。初始化过程中，INT_B 被至于低电平状态，FPGA 的配置空间会被擦除为空。一旦初始化结束，等待被设置为高阻的 INIT B 引脚重新设置为高电平，这需要在电路设计的上班将 INIT B 引脚上拉电阻，在 INIT_B 上升沿触发采样 M[0:2]的状态。采样结束后，FPGA 就知道采用的是那种配置模式，接着就进行配置文件的下载。

配置文件下载：针对有 S5PV210 的嵌入式主控系统，本项目采用的是 Slave SelectMAP 模式配置 FPGA，该模式相对串行数据模式速率更快，配置时间更短，考虑到频谱监测设备在切换模式时不能出现长时间的等待，所以配置当然是越快越好。

在使能引脚 CS 引脚和重配 PROGRAM B 引脚同时为低时，在每个时钟的上升沿将通过 FPGA 的与 ARM 连接设计的数据接口传输的数据下载至 FPGA。涉及到其相应配置引脚的电时序如下图 4-2 所示，其中 t1 和 t2 的大小有 FPGA 的特性决定，t1 为 5~30ms，t2 最大 4ms。

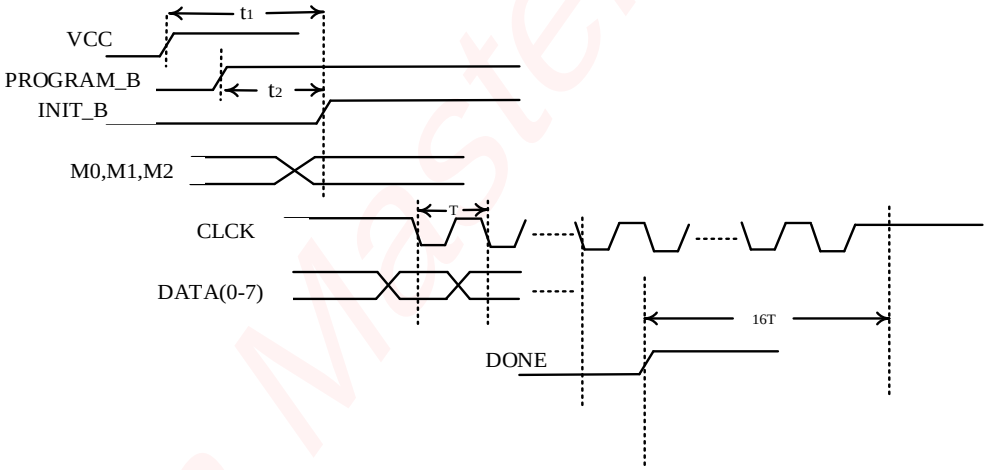

图 4-2 FPGA 配置时序

FPGA 收到所有的数据之后还需要进行 CRC 校验，配置文件的的最后部分会促使 FPGA 开始加载配置数据。这时 FPGA 会将添加有上拉电阻的 DONE 引脚设置为高阻态，这个时候我们就可以通过这个引脚的电平检测 FPGA 是否配置成功。不过配置文件最后触发的 FPGA 加载程序的完成与否与 DONE 引脚没有直接的关联，所以我们要保证触发加载的程序完全正常下发之后才可以取消掉时钟，也就是起动程序到达 EOS 状态。这个时候不仅要保证时钟在 DONE 引脚拉高之前一直有效外，还要在 DONE 引脚拉高之后在保持大于引导加载触发程序的数据量发送所需的时钟周期，这里设置为 16 个。

4.1.2 FPGA 动态配置驱动实现

在 Linux 系统中，没有标准的驱动系统与 FPGA 的配置时序逻辑相同或者相仿的，因此本设计采用模拟 FPGA 配置时序的方式实现 FPGA 动态配置接口驱动，同样此驱动依然是字符设备驱动类型，具体流程与图 4-1 流程图一致。整个 FPGA 配置接口驱动的整体架构如图 4-3，整个配置接口驱动依然以字符类设备驱动的方式实现。

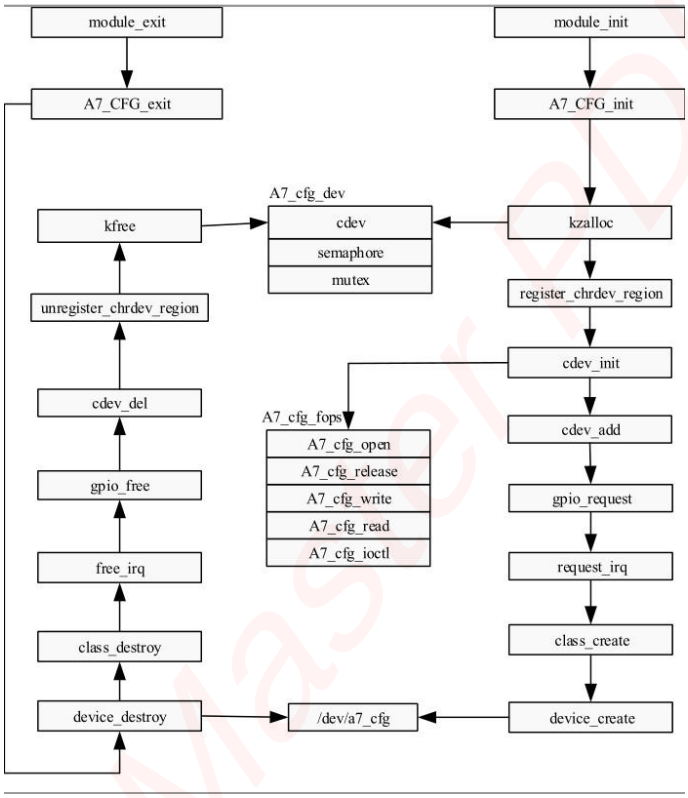

图 4-3 A7 FPGA 的驱动设计框架

4.1.2.1 GPIO 并行设计总线实现

A7cfg dev 是定义好的设备结构，内含标准字符设备结构体 cdev，同时将信号量 semaphore 以及互斥量 mutex 也定义在这个结构体中 。互斥量的作用主要是保证在操作临界区的独享；设置 GPIO 组的配置寄存器使得利用的那组 GPIO 被设置成驱动需要的工作模式；FPGA 和主机的交互采用中断的方式实现握手通信的机制，由于数据下发涉及到内存分配等，这些不能在中断处理程序中完成，所以通过中断更改信号量来完成数据握手。

在确定设备驱动的整体架构之后，我们就要完善字符操作结构体 A7cfg_fops，接着开发如打开，释放，读写等操作，io 控制函数，这些函数实现了 ARM 和

FPGA 的交互，包括初始化时序是需要与 FPGA 完成的交互函数的接口，因为初始化的时序流程主要是 GPIO 的电平按照控制逻辑的时序操作，我们将其封装在 ioctl 函数里面，定义其操作魔数为#define A7CFG _IOC MAGIC 'k'，需要和驱动中的操作魔数相匹配驱动才可进行操作，这个机制可以保证函数得额接口不会随意被其他进程所操作。

具体到 ioctl 的操作按照图 3-21 Airtex-7 配置过程体现如下图 4-4。

图 4-4 FPGA 重配驱动 IOCTL 函数中重配过程的体现

执行这个控制命令后，硬件上 FPGA 才进入重配文件接收阶段。FPGA 重配的初始化函数会在加载模块时调用，即执行 insmod 时调用，而初始化时需要做的就是包括资源的申请，向内核注册该设备，创建设备文件节点等等。

编写 Makefile 文件，如图 4-5，同样编译成模块。

图 4-5 Makefile 文件

执行 make 命令，make 会按照 Makefile 和指定的内核目录下的 rules 编译驱动文件，生成驱动的模块文件，拷贝到目标板上，执行 insmod，可以看到目标版输出驱动加载成功的打印信息，如图 4-6：

```
root@FORLINX210]#
root@FORLINX210]# insmod fpga_cfg.ko
  109.624731] register the A7 success.
```

图 4-6 FPGA 重配驱动加载成功输出信息

频谱仪的 FPGA 重配目标文件是由 FPGA 开发小组的固化程序生成，我们的重配工作就是需要把这个二进制文件写入 FPGA 并检测是否重配成功。

```
fd=open(”/dev/a7cfg”,O RDWR);
……
if((fd2=open(”led.bin”,O RDONLY))¡= 0)
//写初始化信息到 A7
……
```

在进行重配时，主要的操作就是向 FPGA 写入重配文件，主要的调用函数是驱动里面的写函数。进行切换操作需要将测试程序跨平台编译，在 x86 平台调用交叉编译工具，编译成 ARM 可用的可执行文件。拷贝到目标板上，同时将需要写入 FPGA 的二进制文件也拷贝到目标版与测试程序同目录下，执行该模式切换测试程序，发现配制成功，FPGA 的配置完成引脚 FPGA DONE 拉高，如图 4-7 指示灯点亮。

图 4-7 FPGA 重配驱动改进 DMA 方式连接方式设计图

4.1.2.2 DMA 方式改进

通过次驱动进行 FPGA 重配的时间大概是在 9-10s 左右，在界面上切换 SPAN 参数额同时会切换 A7 的配置数据，相当与重新将 FPGA 重新初始化之后，再将新的二进制 bin 文件重新写入，由于上述驱动设计的数据总线位宽是 8 位，时钟是靠 GPIO 实现的，时钟频率较慢，所以后续有设计了新的方案采用 DMA 的方式将 FPGA 挂载在 AHB 总线上。具体的接口原理图设计如图 4-8 在 BANK3 上。

DMA 是直接存储器的数据传送过程中，调用 DMA 控制器进行数据传输的控制[46]，不需要耗费中央处理器处理器的资源，这也就导致中断是 DMA 传输实现的时候所必需的一个功能，即使数据传输中央处理器不参与，但也需要在适当的时候告知处理器数据传输完毕，可以进行下一步操作。 CPU 要负责最开始 DMA 控制器的相关初始化工作，比如设置通道，源地址，目的地址，所传输的数据大小等等。初始化完成，就不需要 CPU 在进行参与，可以被内核调度去进行其他的进程处理。DMA 数据传输结束，控制器向处理器发送传输结束的中断，DMA 控制器将目标地址的数据提取完成。

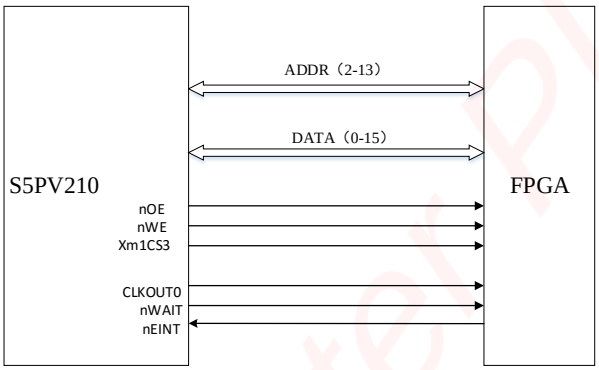

图 4-8 FPGA 重配驱动改进 DMA 方式连接方式设计图

在本项目中, FPGA 的驱动程序是一个字符设备驱动，数据传输的过程需要配合 DMA 控制器来使用， dma _ read , dma _ write , dma _ ioctl , dma _ init , dma _ exit 等多项函数都需要在驱动中实现。驱动的设计流程如图 4-9 所示:

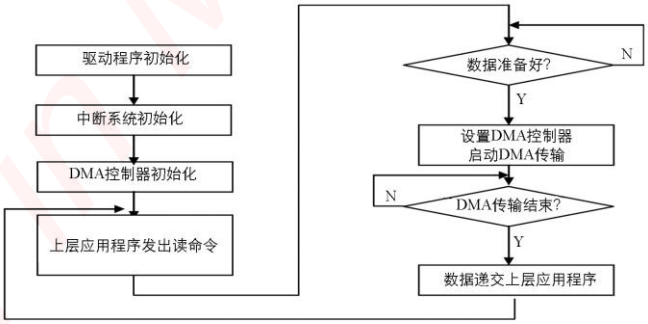

图 4-9 Linux 下 DMA 方式交互流程

系统加载后，驱动程序执行初始化代码，该阶段会配置相关寄存器，以及存储区域的物理内存到虚拟内存的映射，接下来进行中断申请，配置外部中断。接着初始化 DMA 控制器，我们知道 S5PV210 支持 24 通道的 DMA，所以要在这里

指定 DMA 通道号，由于 DMA 传输的是连续的地址的一段数据，所以需要做诸如指定其起始地址，和目标地址等待初始化操作。

应用程序发动写入 FPGA 的命令，启动 DMA 传输传输结束中断到来，FPGA 将目标地址中的数据取走后回复一个中断，操作结束，我们甚至可以一次性将所有的配置文件全部下发。

4.1.2.3 改进结果分析

先进性简单的数据读写测试：

```
While ( 1 )
nreadfromfpga=read ( fd _ fpga ,( char* ) buf , nToRead );//读取数据到 buf
```

可以感受到这个速度非常快，如果 FPGA 的时钟与 DMA 控制器的时钟匹配的时候，重配文件的下发也会达到这个速度而且整个过程不需要 CPU S5PV210 的参与，可以让在传输数据的同时 CPU 去处理其他的事情，降低了资源的消耗，同时也不会造成 CPU 进程的阻塞。

同样按照驱动的测试逻辑，先进行模块化的测试，生成 FPGA DMA drv.ko, 执行 insmod，进行实际测试，发现在实际的频谱监测设备中，FPGA 的配置速度不到一秒的时间就完成了，如图 4-10。

```
[root@FORLINX210]# time ./a7CfgToDetect.o
execute the open [  187.083133] Detect started!!
a7 device.
init the a7 configuration.
[  187.287181] A7_CFG_CLK==1
[  187.288320] A7_CFG_RDWR_B==0
[  187.291195] A7_CFG_CS==1
[  187.394254] A7_CFG_PROGRAM_B==0
[  187.509376] A7_CFG_PROGRAM_B==1
[  187.611591] A7_CFG_CLK==0
[  187.612754] A7 Configuration Success!
whether the operation is success?
ioctl: Success
Command exited with non-zero status 242
real    0m 0.53s
user    0m 0.00s
sys     0m 0.51s
[root@FORLINX210]#
```

图 4-10 DMA 配置速度测试

4.2 DSP 与 S5PV210 的 HPI 接口的驱动设计与实现

4.2.1 HPI 控制时序

主控对 HPI 的访问如下三个步骤，首先主机发起访问，发起访问，响应，收到回应结束访问。

主机输出硬件信号为 HSTROBE，HR/W，HCNTL0/1，HWIL，以及数据线 HD[0:n]。总线在 HSTROBE 的电平下降沿进行控制信号的采样，同时在做出对应寄存在控制寄存器内的控制命令的相应。

有 HR/W 控制线得到进行的使读还是写操作，如果是写操作，总线在 HSTROBE 的上升沿将数据线上的数据直接锁存到有 HCNTL0/1 和 HWIL 这三个控制线所确定的寄存器。如果是读 HPIC 或 HPIA 寄存器，HPI 总线会将将寄存器内存储的数据锁存在数据总线上；如果 HPID 寄存器的话，HPI 先将 HRDY 置为忙状态，继而将数据从 HPIA 指向的内存单元送入 HPID，再传输至数据线，并清除 HRDY 忙状态。

针对 S5PV210 并没有成熟的 HPI 驱动控制器或者总线协议，这就意味着我们需要在驱动里面将其按照 HPI 的逻辑时序进行详细的实现。这个时候就需要遵循 Lin ux 下驱动的设计体系和框架结构进行设计，如果想要开发的 HPI 驱动有良好的一致性和可继承性，就要按照驱动的面向对象和分层设计思想，使 HPI 的操作方法独立于 CPU。如果将来进行产品的更新迭代或者是架构更换，HPI 驱动的架构不需要进行变化，只需修改其中方法函数的映射即可。

DSP 的片内 RAM 主机方和 DSP 都可以获得控制权。保证主机访问时主机与 DSP 的时钟同步，同一个存储地址如果收到主机和 DSP 的同时访问，两者生冲突解决策略是主机具有优先访问权，一个时钟周期后 D SP 才能接着去访问。

HPI 总线控制器在 DSP 内部有三个主要的寄存器： HPIC (控制寄存器)， HPIA(地址寄存器)， HPID (数据寄存器)。控制寄存器 HPIC 存放的就是 HPI 的控制命令，

HP IC 一共 32 位每一位代表相应的控制命令，该寄存器无论是主机还是 DSP，都可以进行执行的访问，具体的 HPIC 寄存器对应的控制位的功能如下图 4-11 和 4-12。而 HPIA 是地址寄存器，只能有主机进行直接的访问，该寄存器也是 32 位其中存放的就是当前 HPI 寻址的存储单元。HPID 是数据锁存寄存器，也只有主机可以对齐进行访问，无论是读还是写，该寄存器中存放的就是对应地址的数据，这些数据会通过 HPI 的数据总线进行传输。针对每一位其意义分别是如下图 4-9 所示

31 21	20	19	18	17	16
Rsvd	FETCH	HRDY	HINT	DSPINT	HWOB
15 5	4	3	2	1	0
Rsvd	FETCH	HRDY	HINT	DSPINT	HWOB

图 4-11 HPIC 寄存器位描述

位	主机	DSP	功能描述
HWOB	RW	R	地址和数据寄存器半字顺序选择位该位为 0，低字节首先传输；该位为 1，高字节首先传输。对于 32 位的 HPI 无意义。
DSPINT	RW	R	主机到 DSP 的中断，主机向 DSPINT 位写 1，可以中断 DSP。主机和 DSO 读该位总是为 0，DSP 写该位无效。
HINT	R	RW	DSP 到主机中断。DSP 向 HINT 写 1，产生一个主机中断，HINT 位是 HINT 引脚的反向逻辑电平。主机或者 DSP 向该位写 0 时，无效。
HRDY	R	R	HRDY#引脚的逻辑状态位。主机和 DSP 可以读取该位来查询 HRDY#引脚的状态，如果 HRDY=0，则表示 HPI 还没有完成当前的数据传输。
FETCH	RW	R	主机数据获取请求。主机向该位写 1 时，HPIA 寄存器所指定地址的数据被获取，并加载到 HPID。主机和 DSP 读该位的地址总为 0.
Rsvd	——	——	预留位，对 C64X 可以对第 7、15、23 和 31 位写入 0

图 4-12 HPIC 寄存器功能表述

HPI 接口是通过控制引脚（HCNTL0，HCNTL1）进行这三个寄存器访问的选择，具体的对应方法如下图 4-13 所示：

HCNTL0	HCNTL0	功能描述
0	0	主机对 HPIC 寄存进行读写操作
0	1	主机对 HPIA 寄存器进行读写操作
1	0	主机以地址自增的方式对 HPID 寄存器进行读写操作，每读取一次 HPID，HPI 寄存器地址增加一个字地址
1	1	主机以固定的地址模式对 HPID 寄存器进行读写操作，HPIA 寄存器地址保持不变

图 4-13 HCNTL0，HCNTL1 控制引脚对 HPI 功能寄存器功能选择

三个输入信号 HCS，HDS1，和 HDS2 产生 HSTROBE，HCNTL1、HCNTL0、HR/W 在在 HSTROBE 信号的下降沿被锁存，使能 HCS 需要在这和阶段被设置为有效。HPI 在此时获得了控制信息，包括半字，读写，控制命令。DSP 根据 HPIC 中的控制状态作出相应的回应，由于 DSP 并不是一直在等待 HPI 的所有相应，这就意味着可能 DSP 处于忙碌状态，需要通知主机有一定的延时，这个功能是通过 HRDY 信号线实现。如果 DSP 准备未完成，则 HRDY 一直处于高电平状态，主机会在 DRDY 变为低电平一段时间后自动将数据读走或者送出，而 HSTROBE

信号也会在 HRDY 的低电平之后短暂时间内恢复高电平，接着进行下一个处理流程，产生下一个下降沿进行数据锁存，这便是一个完整的 HPI 读写周期。具体的时序图如下图 4-14 所示。

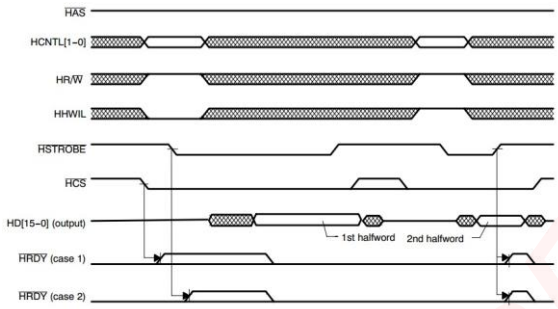

图 4-14 HPI 读写时序

4.2.2 HPI 驱动设计与实现

DSP 上的 HPI 只是一个总线，本身不是存储器，所以不存在寻址的概念。HCN TL0，HCNTL1 的组合决定了 HPIA、HPIC 和 HPID 寄存器访问控制，我们可以通过 SROM 的地址映射完成次三个信号线的控制，HR/W 是读写信号线，同样 S ROM 控制器也有单独的读写逻辑，以及地址线，还有忙碌等待线与 HRDY 对应。

方案设计中已经分析过，将 HPI 总线应设在 SROM 的 Band1 上，那么 HPI 的三个寄存器也需要进行地址的映射，使得其在 S5PV210 的 SROM 控制器看来就是一个存储器，这三个控制器就是不同地址的三个控制单元，这些都需要在驱动设计中进行实现，才可以实现相应的逻辑操作。这时我们就需要将三个寄存器进行地址的映射，使得其在 S5PV210 的 SROM 控制器看来就是一个存储器，这三个控制器就是不同地址的三个控制单元。

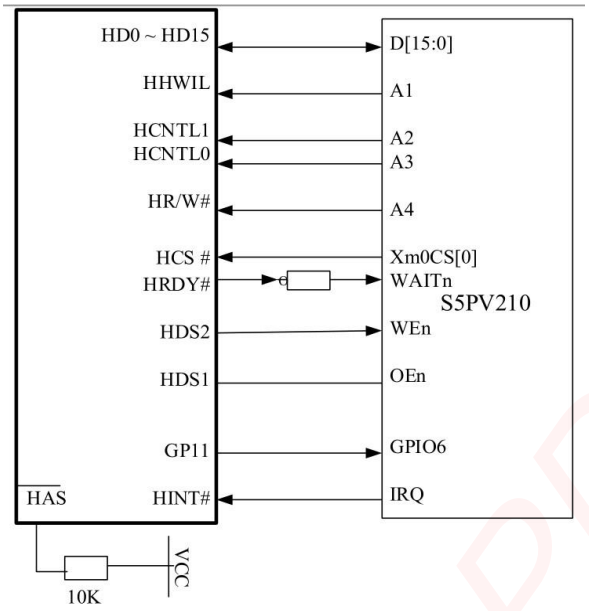

图 4-15 HPI 与 SROM 控制器的硬件连接图

那我们就需要对其不同的控制逻辑进行地址的编码，如下图 4-16 所示

A4	A3	A2	A1	功能描述
0	0	0	0	将 DATA[15:0]写入 HPI 的 HPIC 寄存器，并标记为低半字
0	0	0	1	将 DATA[15:0]写入 HPI 的 HPIC 寄存器，并标记为高半字
0	0	1	0	将 DATA[15:0]写入 HPI 的 HPIA 寄存器，并标记为低半字
0	0	1	1	将 DATA[15:0]写入 HPI 的 HPIA 寄存器，并标记为高半字
0	1	0	0	以 HPIA 自增的方式，将 DATA[15:0]写入 HPI 的 HPID 寄存器，并标记为低半字
0	1	0	1	以 HPIA 自增的方式，将 DATA[15:0]写入 HPI 的 HPID 寄存器，并标记为高半字
0	1	1	0	保持 HPIA 不变的方式，将 DATA[15:0]写入 HPI 的 HPID 寄存器，并标记为低半字
0	1	1	1	保持 HPIA 不变的方式，将 DATA[15:0]写入 HPI 的 HPID 寄存器，并标记为高半字
1	0	0	0	读取 HPI 的 HPIC 寄存器的低半字
1	0	0	1	读取 HPI 的 HPIC 寄存器的高半字
1	0	1	0	读取 HPI 的 HPIA 寄存器的低半字
1	0	1	1	读取 HPI 的 HPID 寄存器的低半字
1	1	0	0	以 HPIA 自增的方式，读取 HPI 的 HPID 寄存器的低半字
1	1	0	1	以 HPIA 自增的方式，读取 HPI 的 HPID 寄存器的高半字
1	1	1	0	保持 HPIA 不变的方式，读取 HPI 的 HPID 寄存器的低半字
1	1	1	1	保持 HPIA 不变的方式，读取 HPI 的 HPID 寄存器的高半字

图 4-16 编码表

HPI 总线并没有在 Linux 中有过规定，其实一个特殊的总线，不像 SPI 总线 CPU 内部实现了控制器，而且 SPI 是标准的 Linux 分层设备模型，符合总线、设

备和驱动的三层结构，但是 HPI 并没有，我们进行设进行设的时候，现将将其看成字符设备进行驱动的开发，是整个频谱监测设备平台可以运行验证，后续为了驱动额继承和移植，将其也利用分层的思想实现。

驱动程序整体架构见框图 4-17：连续内存空间的分配调用 kzalloc，该内存分配函数在所需内存不能获得的时候可以进入睡眠；mutex init 在定义互斥量之后，对齐进行初始化，主设备号和次设备号通过 MAKEDEV 宏操作生成；字符设备的注册有两种方式一种是动态注册使用 alloc chrdev_region 自动分配可用的主设备号，该方法多用在没有指定主设备号或者主设备号已经被占用的情况下，而静态分配调用 register _chrdev region 注册，多用于在驱动程序中直接指定了主设备号，一般驱动两种注册方法条件使用，条件就是主设备号的定义是否为真；cdev add 向系统中注册字符设备，Linux 中实现了一个名为 udev 的产生设备文件节点的机制，该机制主要集成函数 class create、device create，前面一个负责生成设备类，第二个创建设备节点，设备节点会被创建在/dev 的目录下。驱动中并没有 C 语言库的中的输入输出等函数，如果需要调试驱动家需要调用 printk 将需要输出的调试信息显示成内核信息，输出到终端，通过终端进行查看和调试。

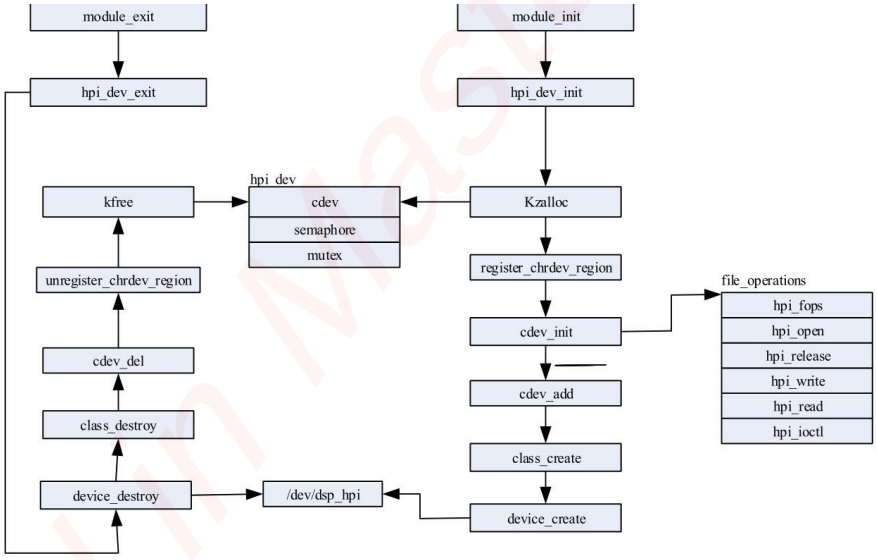

图 4-17 HPI 驱动设计流程

完成驱动的编写后，进行编译。当前工程目录下应有的文件有，dsp hpi.c，Makefile。其中，Makefile 文件为 make 命令的执行方式，一般情况下 make 命令需要制定编译环境，包括目标平台，源码目录等等，正确的执行方式为 make –C / hom e/chase l/android-kernel-samsung-dev/ M=’pwd’ modules. 该语句说明的是，目标文件编译为 modules，指定的内核目录为 KERN DIR,生成的目标代码存置于 pw

d（当前目录下），obj-m 指示编译工具将其编译为模块.ko 文件。

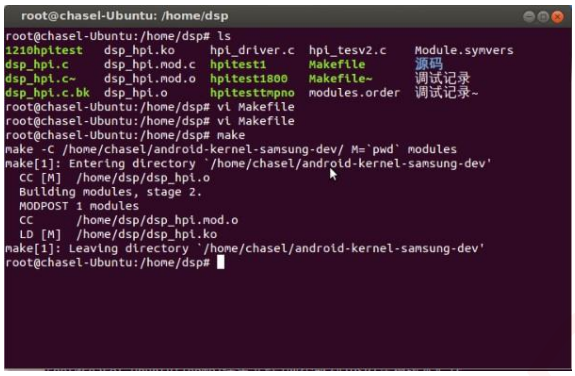

图 4-18 HPI 生成模块文件

在此目录下执行 make，生成模块文件，dsp hpi.ko，将其拷贝入目标板，进行相应的逻辑测试。

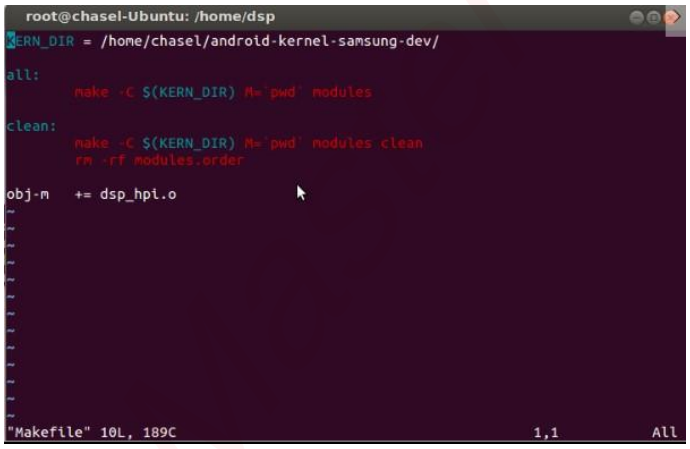

图 4-19 执行 make 命令

首先在目标板上执行 insmod dsp hpi.ko，这样，一个插入模块的过程就完成了。调用驱动编写时的初始化入口，也就是 module init 宏所引导的 hpi dev _init 函数，根据 hpi dev init 函数的实际操作过程我们可以看到初始化成功提示。

register DSP HPI device success!!!

4.2.3 HPI 驱动测试结果

接下来进行实际的读写操作，针对 ARM 与 DSP 的数据交互，我们知道，实际的数据流行是由 DSP 流向，我们知道需通过 HPI 向 DSP 取得大量的频谱数据，将此操作集成到驱动的读函数中，上层在进行数据调用时只需循环调用读函数（s

ysfs 文件系统的标准 read 接口即可）。写简单的测试程序如下，循环读 15 次：

```
int main()
……
/*读十五次次*/
for(i=0;i¡15;i++)
read(fd,pReadBuffer,1800); for(i=0;i¡18
00;i++)
printf(”read from the file is
pReadBuffer[i]=’0’;
……
```

我们需要按照 HPI 总线的设计时序逻辑，使用逻辑分析仪工具进行测试，由 HPI 的控制逻辑来看，我们需要特别关注如下几条控制线，HCS，HINT，HCNTL0，HCNTL1，HWIL，外加数据线，采用逻辑分析仪进行时序的实际测试，如图 4-20：

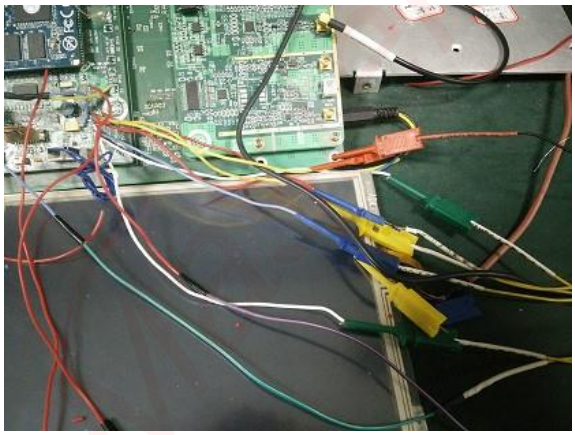

图 4-20 应用过程中逻辑分析仪检测到的时序信息

先启动逻辑分析仪采样，根据标准时序图，HPI 在 HCS 控制线低电平时才可进行针对 DSP 的读写，所以出发电平选择 HCS 的下降沿触发，由此得到的测试时序图可以看到，十五次均有体现，具体到每一次的读写逻辑，如下图， 经过与 HPI 总线的时序进行对比分析，时序完全正确，这是只写 HPI 寄存器的状态。联合 DSP 调试端进行读取数据的联合判断，得出结论驱动工作正常无误。

得到的时序逻辑图，如图 4-21 所示 可以看到读写逻辑线，HR/W 读写控制线 R 高电平有效，W 地点凭有效；按照读函数的过程，先写 HPID 寄存器，对应的 HCNTL 控制线编码为 00，即 HCNTL0，HCNTL1 均为低电平；HWIL 半字信号发送正确。

图中，我们看到 H RDY 有脉冲，因为 HPI 连接 ARM 的 AHB 总线，该总线是复用的；如果总线上的数据没有准备好，HRDY 信号呈现高电平，连接反相器到 ARM 检测到就是低电平，这时根据 SROM 控制器协议，我们知道，总线会等待 HPI 总线 Tcos = 2-cycle ，数据准备好之后，HRDY 在 DSP 端拉低，ARM 端拉高，读取数据继续。

图 4-21 应用过程中逻辑分析仪检测到的时序信息

图 4-22 应用过程中逻辑分析仪检测到的时序信息

在实际测试过程中我们也可以感受到 HPI 的读取速度也是极快的，在 15 个读写周期，每个点 4 个字节，每次 3200 个点的数据，总共耗时在毫秒级以下，自带的 time 命令已经监测不到 15 次的运行时间，如图 4-23。

```
read from the file is 30
read from the file is 30
read from the file is 30
Command exited with non-zero status 25
real    0m 0.00s
user    0m 0.00s
sys     0m 0.00s
[root@FORLINX210]#
```

图 4-23 HPI 的 15 个读写周期耗时

4.3 S5PV210 与频率合成模块控制总线驱动设计与实现

本项目中频率合成模块的参数主控芯片是靠一款低性能 FPGA Spartan 3 实现的，其与 ARM 之间的交互采用的是 SPI 的总线连接，我们需要在 S5PV210 中设计并实现这个 SPI 设备的驱动。

4.3.1 SPI 硬件控制逻辑

SPI 总线硬件上有四根控制线，每根控制线的代表分别是

1.SDO – 数据主入从出线

2.SDI – 数据主出从入线；

3.SCLK – 时钟；

4.CS – 使能信号线。其中，CS 是芯片的是能信号线，每个 SPI 外设通过 CS 使能才会被选中，如果本设备连接的 CS 无效（电平检测），就没有被选中，这也是同一个 SPI 总线上可以挂在多个 SPI 设备的基础。

SPI 常用四种数据传输模式，串行同步时钟极性 CPOL= 0 或 1 决定了串行同步时钟的空闲状态为低电平还是高电平，相位 CPHA= 0 或 1 决定了数据被踩样式在同步时钟的前沿或者后沿。

不同的传输模式时序图如下图 4-24，我们之举 CPOL 和 CPHA 分别为 00 和 01 的时序图做例子。这四种模式中究竟选择哪种模式取决于设备。

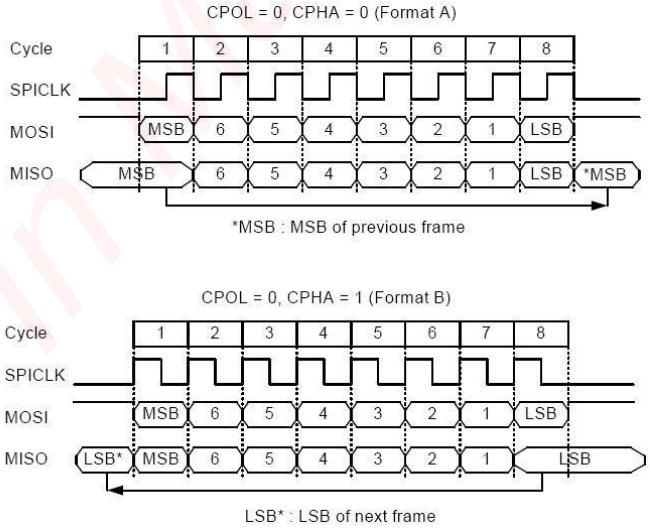

图 4-24 SPI 的其中两种传输模式

我们这里针对配置射频端的 FPGA 采用的 SPAN3，由 SPI 通道进行参数的下发。根 FPGA 开发者协商之后，这里 FPGA 采用为 0 0 的模式，即串行同步时钟

的低电平空闲状态，前沿采样。

硬件电路连接如图 4-25 所示，

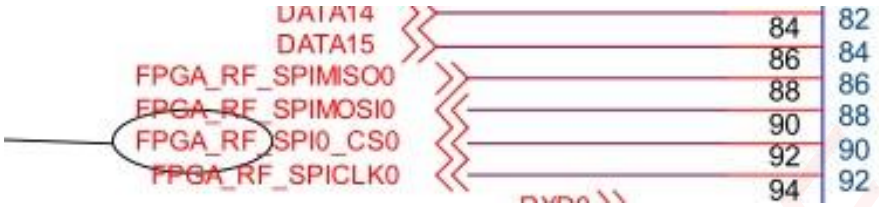

图 4-25 S5Pv210 与 Spartan 3 的 SPI 总线硬件原理图

4.3.2 基于分层和隔离思想的 SPI 驱动设计实现

Linux 中的 spi 驱动主要是由 spi 子系统来管理的，其核心代码位于 kernel /drivers/spi/spi.c 中。具体的 spi 控制器驱动也在 kernel/drivers/spi/目录中。pi 通常是 spi 控制器、spi 总线和连接在 spi 总线上的设备构成：这里的总线只是指物理的 spi 连线，并不是指 Li nux 设备模型中逻辑上的总线概念。可以把 spi 控制器和 spi 总线看成是一体的，spi 总线就是 spi 控制器加上和 spi 设备的连接线。spi 设备包含很多种，它可以是一个 spi 接口的 nor flash，比如本项目实用的 SPAN3 的与 ARM 的通信 SPI 通信总线。

4.3.2.1 SPI 控制器

Linux 的 spi 子系统对 spi 控制器的描述使用的是 struct spi master 这个数据结构，所以在内核中，一个 spi master 结构就代表了一个 spi 控制器，或者说代表一个 spi 主机，其结构体含有 spi 设备指针，spi 信息指针等。基于 Linux 的设备模型概念，设备和驱动都需要依附于总线，针对 spi 的总线类型被定义为 spi bus type。

4.3.2.2 SPI 设备

Linux spi 子系统对 spi 总线上的设备用 struct spi device 结构来描述，运行的内核中，通常一个 struct spi device 对象对应一个 spi 设备，该设备结构体描述了这个设备控制器的归属，以及设备的最大时钟速度，片选信号线，工作模式等等。struct spi device 主要用来描述连接在 spi 总线上的一个 spi 设备的一些电气信息的。在 2.6.xx 版本的内核中可能会经常看到平台代码中用 struct spi board info 来描述 spi 设备的电气信息，该结构体包含外设的片选号、总线号、模式以及传输速率等信息。

在板级初始化函数中会用 struct spi board info 来描述系统中的 spi 设备信息，然后调用 spi register board info 函数将这些设备信息注册到系统中。

4.3.2.3 SPI 设备驱动

spi 设备的驱动都会有一个 struc spi driver 结构体来描述，结构体中定义对应的操作函数指针，用来管理依附于总线上的相关设备[48]，spi 设备的驱动主要就是实现这个结构体中的各个接口，并将之注册到 spi 子系统中去。第一个与数据传输相关的数据结构为 struct spi transfer，一个 struct spi transfer 对象代表了一次单段的 spi 数据传输。struct spi transfer 结构体中记录了本段传输需要交换的数据和长度，传输的速度，传输时片选信号的变化情况。

另外一个与数据交换相关的数据结构为 struct spi message，一个 struct spi _me ssage 代表对一个设备进行一个多段 spi 数据传输。每一段传输其实就是使用上面提到的 struct spi _trasnfer 对象完成的。struct spi message 主要记录了这次传输针对的设备。上面提到的 struct spi transfer 对象会被链接到 struct spi message 对象中。

在板级代码中注册 struct platform device 来描述各种控制器，例如 spi 控制器，i2c 控制器等等。然后再向系统中注册 struct platform driver 来管理和驱动对应的平台设备。spi 控制器驱动也是同样的做法。

明确驱动的架构后，进行针对本设备的驱动开发，由于 S5PV210 的 Linux 系统已经集成了 SPI 控制器和总线的驱动结构，我们需要在对应自己设备所属的控制器和总线上添加自己的设备。

针对此设备，我们首先在板极信息里添加自己的 spi 设备的硬件信息，首先是 spi 的片选信号线，板极信息在文件，arch/arm/mach-s6pv210/mach-smdkc110.c 中，找到 spi 专属代码区域，添加以下信息在进行板极信息的注册，包括该设备的名称，工作模式，工作频率，以及对应的控制器。

```
static struct S5PV210_spi_csinfo smdk_spi0_csi[] = {
    [SMDK_FPGASPI_CS] = {
        .line = S5PV210_GPB(1),
        .set_level = gpio_set_value,
        .fb_delay = 0x0,
    },
};
```

图 4-26 SPI 板极信息

```
static struct spi_board_info s3c_spi_devs[] __initdata = {
//niuchong  2015.05.25

    [0] = {
        .modalias        = "spidev",    /* device node name */
        .mode            = SPI_MODE_0,  /* CPOL=0, CPHA=0 */
        .max_speed_hz    = 10000000,
        /* Connected to SPI-0 as 1st Slave */
        .bus_num         = 0,
        .irq             = IRQ_SPI0,
        .chip_select     = 0,
        .controller_data = &smdk_spi0_csi[SMDK_FPGASPI_CS],
    },
```

图 4-27 SPI 设备初始化数据

板极信息中，如上图所示， devs[0]定义该 spi 设备的隶属 0 号控制器上，属于 0 号控制器的 0 号设备，其片选信号线定义在 smdk spio csi[SMDK_FPGASPI CS]结构体中。板极信息中的设备信息，包括其定义信息完成之后，根据 SPI 驱动的三层结构，需要编写该设备的设备驱动。设备驱动同样需要初始化和卸载函数，驱动的入口和出口函数位需要将 spi driver 的设备结构体中的 driver.name 与板极信息中的 name 相匹配，这时 spi 的设备在班级信息初始化时会将该设备注册到内核，

spi 设备的在总线检测到设备的时候，内核会调用 probe 函数，而 probe 函数就会根据 spi 设备的特点设置相应的参数。而 spi 的设备文件操作结构体需要符合字符型设备的设备文件操作结构，如下 struct file operations spi fpga fops ，其结构与一般字符型的文件操作结构体类似我们这里 SPI 设备采用全双工的工作模式，所以我们将操作函数封装在函数 spidev message 函数中，该函数会根据 spi messag e 结构体中的 rx 和 tx 是否为空进行接收和发送数据的操作。我们要和上层软件协商相应的收发模式，以及首发参数。针对常用的而且将该设备的硬件信息注册在板极信息里面的，我们需要将其编译至内核中，而不能编译成模块。需要在对应的 kconfig 和.config 以及 Makefile 指定该设备驱动的编译形式。在/drivers/spi/Kco nfig 中指定 spi 的 menu：

```
config SPI_S3C64XX
        tristate "Samsung S3C64XX series type SPI"
        depends on (ARCH_S3C64XX || ARCH_S5PV210 || ARCH_S5P6450 || ARCH_S5PV310) && EXPERI
MENTAL
        select S3C64XX_DMA
        select S3C64XX_DEV_SPI
        help
          SPI driver for Samsung S3C64XX and newer SoCs.

config S3C64XX_DEV_SPI
        depends on SPI_S3C64XX
```

图 4-28 SPI 的 menu 目录结构

最后需要在内核的项目录下执行 make menuconfig 来配置驱动编译入内核，勾选 SPI 主控制器的驱动，位与 menuconfig 的 Device Drivers—>SPI support—>Samsung S3C64xx SPI Master,再勾选我们的设备驱动，spidev For FPGA,执行 make，将该驱动编译入内核。

```
      *** SPI Master Controller Drivers ***
<*>   Utilities for Bitbanging SPI masters
< >   GPIO-based bitbanging SPI Master
<*>   Samsung S3C64XX series type SPI
< >   Xilinx SPI controller common module
<*>   DesignWare SPI controller core support
<*>     Memory-mapped io interface driver for DW SPI core
      *** SPI Protocol Masters ***
<*>   spidev For FPGA
<*>   Infineon TLE62X0 (for power switching)
```

图 4-29 S5Pv210 与 Spartan 3 的 SPI 内核配置

4.3.3 SPI 驱动调试测试结果

将内核拷贝至开发板的 sd 卡，烧写完成进行测试。内核启动后，可以看到在 /dev 目录下由我们设备的名字，spidev0.0，该命名方式意味调用 0 号控制器的总线上的第 0 个设备。

[root@FORLINX210] #ls /dev/spi*

/dev/spidev0.0

连接上逻辑分析仪，具体的连接如下图 4-30 经过测试，调用测试程序，用逻辑分析仪检测 spi 的控制时序，得到时序图如下可以看到时序是正确的，SPI 工作正常。

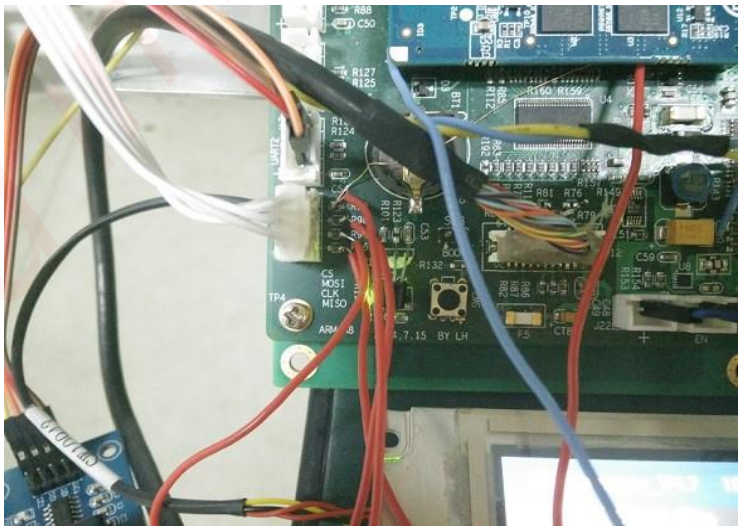

图 4-30 SPI 逻辑测试仪连接

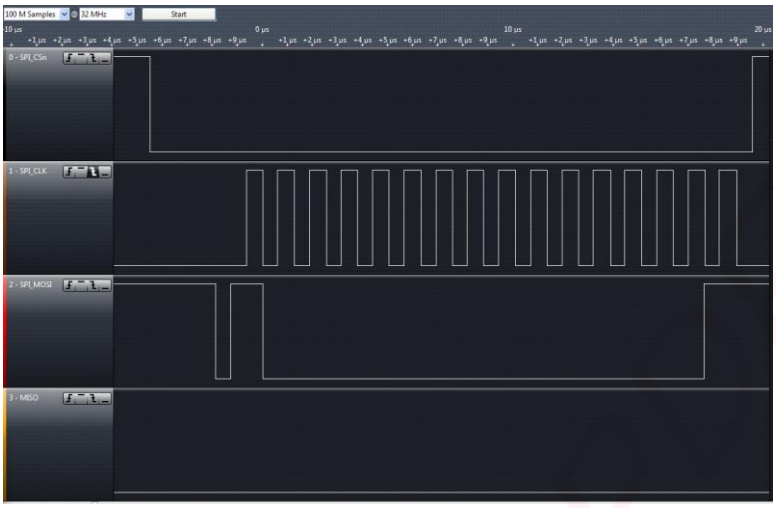

图 4-31 SPI 逻辑测试仪连接

4.4 显示子系统驱动移植及触摸屏驱动设计与实现

4.4.1 显示子系统驱动移植

LCD 的正常工作需要需要 LCD 驱动器，还需要匹配的 LCD 控制器。一般来说，LCD 驱动器会与玻璃基板集成，控制器需要附加的设计电路来实现。S5PV210 集成了 LCD 的控制器，所以我们要根据屏幕的定制参数来一直修改 LCD 控制器驱动以实现 LCD 的正常工作。 LCD 的外部接口信号，我们需要关心的几个参数的意义分别为 VSYNC 为垂直同步信号，HSYNC 为水平同步信号，VCLK 为象素时钟信号。

VBPD(vertical back porch) 是一帧图像开始时垂直同步信号以后无效的行数，VFBD(vertical front porch）是一帧图像结束后垂直同步信号以前的无效行数，VSPW(vertical sync pulse width) 表示垂直同步脉冲的宽度。HBPD(horizontal back porch)是水平同步信号开始到一行的有效数据开始之间的像素时钟个数，HFPD(horizontal front porth)表示有效数据结束到下一个水平同步信号开始之间的像素时钟的个数，HSPW(horizontal sync pulse width)表示水平同步信号的宽度，用 VCLK 计算。

根据 menuconfig，我们索引到原来的 LCD 控制器匹配的驱动是* /drivers/video/samsung/s3cfb.c 这些参数的值是根据屏幕本身而定的，具体屏幕的参数（每一个参数都有相应的意义）如下图，按照典型值进行修改 G084SN03 显示屏接口帧

时序如图 4-32，其参数规定如图 4-32 在显示子系统参数设定时，本方案采用了典型值进行设定，最终输出分辨率为 800*600。

项目	符号	最小值	典型值	最大值	单位
时钟频率	Fck	38	40	48	MHz
水平周期	Th	850	1056	1300	Clk
水平脉冲宽度	Thw	10	128	--	Clk
水平前帘	Thf	15	40	--	Clk
水平后帘	Thb	10	88	--	Clk
垂直周期	Tv	610	628	750	CLK
垂直脉冲宽度	Tvw	1	4	--	Th
垂直前帘	Tvf	0	1	--	Th
垂直后帘	Tvb	10	28	150	Th

图 4-32 本项目中的 LCD 控制器参数表

针对本项目按照参数修改 s3cfb.c 如图

```
44 int lcdsize = 4;
/***************/
1032        else if(lcdsize == 4)
1033            {
1034            printk("now execute here. niuchong");
1035                    fbdev->ltd.->freq=40;//60
1036                    fbdev->lcd->width=800;
1037                    fbdev->lcd->height=600;//600
1038                    //fbdev->lcd->p_width=800; //bs android error
1039                    //fbdev->lcd->p_height=600;
1040                    fbdev->lcd->timing.h_sw=128;//128
1041                    fbdev->lcd->timing.v_sw=4;//4
1042                    fbdev->lcd->timing.h_fp=40;//40
1043                    fbdev->lcd->timing.h_bp=88;
1044                    fbdev->lcd->timing.v_fp=1;
1045                    fbdev->lcd->timing.v_bp=23;
1046            }
```

图 4-33 LCD 控制器参数表设置

可以看到最终屏幕正常显示不再出现花屏错位现象，如图 4-34 所示的对比结

果。

(a) 驱动移植前　　(b) 驱动移植后

图 4-34 显示子系统移植对比测试

4.4.2 触控屏功能实现

本身含有 USB 接口，如果接上 USB 的鼠标，会自动识别为 USB HID 的设备，鼠标可以直接使用。如下图所示，可以看到该鼠标设备所在的设备目录为 /devices/input/event1 鼠标在界面中是有响应的，但是本项目是便携式频谱监测设备，所以考虑到便携性，我们设计 USB 接口的主要用途会用于可移动存储器的方便接入，针对界面的输入和控制，我们将采用触摸屏的形式，集成在屏幕上，便携性大大提升，而且，本身触控屏的操作逻辑更直观，更方便，学习成本低，对频谱仪界面的操作提升很明显。

本项目中，触摸屏采用四线电阻屏。四线电阻触摸屏完全密封处理，抗尘土，抗水，而且功耗小，非常适合用于便携式仪器[49]，本项目中的频谱监测设备就采用此种触摸屏控制器。本项目采用 PenMount 6500 的 USB 接口触摸屏控制器，它是一个集成功能和自动检测 USB 和 RS232 串口功能接口，支持本项目中的 4 线电阻式触摸屏的控制板，还有一个 4 针 USB A 型线缆。因为项目嵌入式系统板设计的有 USB 的接口，所以本项目选用其 4 针 USB 型接口。本控制器的控制板如下图 4-35 所示：

图 4-35 触摸屏控制板

与嵌入式系统，USB 连线固定在 USB 接口的第 1 个输入设备接口，这里直接采用 USB HID 的设备驱动，针对 USB HID 驱动，需要按照 USB HID 驱动的标准开发流程。

HID 设备也是在原内核中集成了通用的控制器。HID 是人机交互设备的统称，包括鼠标，键盘等等可以与人类进行直接的输入输出交互的都可以被称为人机交互设备的大类。

针对 USB HID 设备，我们需要在移植对应的 HID 设备驱动并将其编译入内核。在 make menuconfig 中，选中 USB Human Interface Device(full HID) support，支持 USB HID 设备，提供 USB HID 的 API，再针对特殊的硬件移植相应的设备驱动即可。在 drivers / hid / usbhid / Kconfig 看到这项对应的为：CONFIG_USB_HID 又在 drivers / hid / usbhid / Makefile 中看到：obj-$(CONFIG_USB_HID) += usbhid.o 在/driver/ hid /

Makefile 添加 PenMount 的编译规则，如下图 4-36 所示

```
obj-$(CONFIG_HID_ZYDACRON)      += hid-zydacron.o
obj-$(CONFIG_HID_WACOM)         += hid-wacom.o
obj-$(CONFIG_HID_PENMOUNT)      += hid-penmount.o

obj-$(CONFIG_USB_HID)           += usbhid/
obj-$(CONFIG_USB_MOUSE)         += usbhid/
obj-$(CONFIG_USB_KBD)           += usbhid/
```

图 4-36 USB PenMount 的编译规则

再在 hid 的目录下的 Kconfig 中添加 menu 菜单，Device Drivers—HID Devices—Special HID drivers— TouchScreen PenMount，勾选为编译入内核。

这样，设备驱动已经移植完毕，在将其插入 USB 接口之后，可以在 Input 目录下看到 event1 设备，即为该触摸屏设备的设备节点。为了使触摸屏能够在 Qt 程序中可以正常调用，我们需要一直 Tslib 至目标板。 tslib 是是一个电阻屏触摸校准的开源程序库，其针对驱动的触摸屏采样数据去抖、校准等，这个库为上层的应用提供统一调用接口。因此这里先编译安装 tslib，这样在后面编译 Qt 的时候才能打包编译进去。

复制编译结束的 Tslib 到目标板上，然后添加相应的库文件所在的目录到环境变量中，这样在应用程序运行的时候才可以正确地找到相应的库进行正确地调用。按照要求将毕业的文件放置在对应的目录下，比如库文件放在开发版的 lib 文件下，配置文件在放在 etc 目录下。最后添加环境变量，通过超级终端在开发

板系统中添加环境变量，将其添加到/bin/目录下的 qt4 脚本文件中，本项目中许多对该开发板的配置都设置在此脚本中，最后在/etc/init.d/rcS 初始化脚本文件中添加“source qt4”使得相应的环境变量定制启用。

环境变量中的设置文件设置如下：

```
export TSLIB\_TSDEVICE=/dev/input/enent1
export TSLIB\_CONFFILE=/etc/ts.conf
export TSLIB\_PLUGINDIR=/usr/local/tslib/lib
export TSLIB\_CALIBFILE=/etc/pointercal
export TSLIB\_CONSOLEDEVICE=none
export TSLIB\_FBDEVICE=/dev/fb0
if [ -c /dev/input/event1 ]; then
if [ -f /proc/driver/ft5x06\_ts ]; then
echo "capacitor panel touch..."
export QWS\_MOUSE\_PROTO='Tslib:/dev/input/event1'
else
echo "resistance panel touch..."
export QWS\_MOUSE\_PROTO='Tslib:/dev/input/event2'
fi

if [ -e /etc/pointercal -a ! -s /etc/pointercal ] ; then
rm /etc/pointercal
/root/tslib/build/bin/ts\_calibrate
fi
else
```

以上环境变量设置完成后需要检测是否与设计板上的文件节点对应。环境变量，QWS MOUSE PROTO 制定了该输入设备的鼠标协议，如果不进行正确的指定，QT 程序不能正确接收触摸屏的检测。触摸屏对应的设备节点设置为/dev/input/event，这个要与环境变量 TSLIB_TSDEVICE 一致，毕竟应用程序通过环境变量来读取这些。

图 4-37 触摸屏控制板

图 4-38 触摸屏控制板

最后运行屏幕校准程序，ts calibrate,该程序取左上，左下，右上，右下和中心点五个点为基准点，根据手动校准的点击情况，进行校准，触摸屏的校准差值保存于/etc/pointercal。这样 Qt 程序会在启动时读取 TSLIB CALIBFILE 环境变量中校准插值文件中的数据，自动换算出准确的触点位置。运行测试程序，结果如下图 4-37 所示可以看到，触控屏工作正常，可用手指在上面写，拖拽，如图 4-38。

4.5 其他外设子系统驱动实现

4.5.1 网络子系统驱动实现

手持式频谱监测设备在嵌入式系统设计时，采用地址线 ADDR5 控制操作 CMD，外部中断 5 引脚接收中断。

Linux 网络设备驱动的体系结构依次是网络协议接口层、网络设备接口层、提供实际功能的设备驱动功能层以及网络设计与媒介层，这四层的作用如下图 3-39 所示：

1.网络协议接口层向网络层协议提供标准的数据包收发接口，同意使用 dev_queue_xmit( )函数发送数据， netif_rx( )接收。使得上层协议与下层具体的设备处于隔离状态。

2. net_device 是网络设备接口层向协议提供的同意设备的结构体，该结构体是设备驱动功能层中各函数的容器。

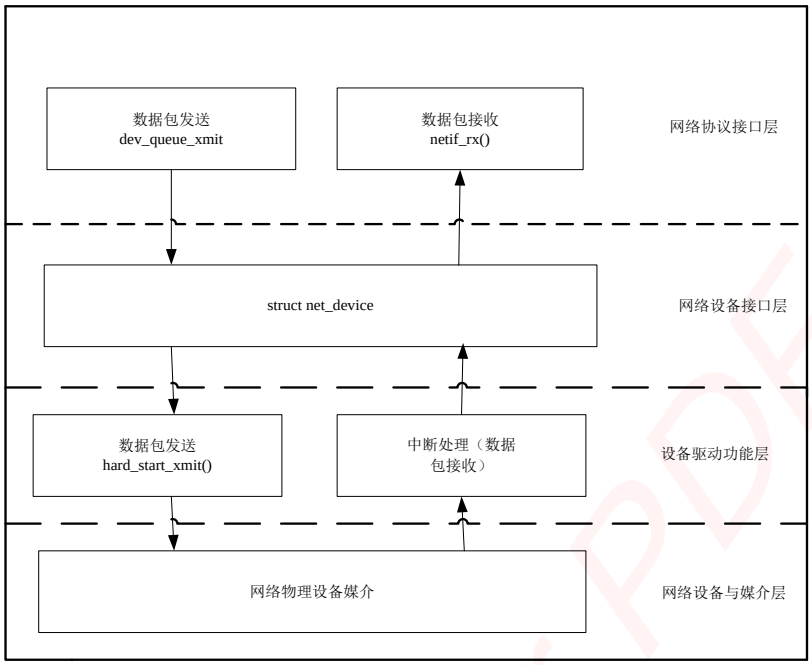

图 4-39 Linux 网络设备驱动的体系结构

3. net_device 数据结构的具体成员包含设备驱动功能层的各个函数，驱使网络设备完成指定的操作， hard_start_xmit( )函数在设备进行发送数据前进行初始化，再通过设备中断进行接收操作。

DM9000 网卡驱动实现通用网络设备驱动体系中的 net_device 结构体，重要的函数如 hard_start_xmit、open、stopset_multicast_list、do_ioctl、tx_timeout 等 [50]。

DM9000 在移植驱动的时候，只需要在板级文件中定义其对应的硬件资源，比如其寄存器和数据地址，并指定正确的 IRQ 资源，在平台文件里面设置 dm9000 的驱动的初始化状态。

再定义该设备的 device 结构体的注册结构体，使得设备在初始化时被注册到系统虚拟总线上，在 Kconfig 中添加相应的目录结构，在默认配置文件中添加该设备

具体的目录 Device Drivers--->Network device support--->Ethernet(10 or 100Mbit)--->目录下，勾选 DM9000 support，DM9000 16 bit 等目录，执行 make，将驱动编译入内核，拷贝人开发板进行测试，得到测试结果如下。

执行 ifconfig 查看网卡的 IP 分配情况，如下图可以看到初始化状态下网卡未分配合理的 ip 地址和网关地址。用 udhcpc 命令利用路由器的额 dhcp 服务器获得目标版的局域网 IP 地址和网关等信息。

```
[root@FORLINX210]# udhcpc
udhcpc (v1.13.3) started
Sending discover...
Sending select for 192.168.1.101...
Lease of 192.168.1.101 obtained, lease time 7200
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.1.1
[root@FORLINX210]#
```

图 4-40 dhcpc 结果

利用 ifconfig 查看到网卡的具体网络配置：

```
[root@FORLINX210]# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:90:90:90:90:90
          inet addr:192.168.0.232  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:37 Base address:0xa300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

图 4-41 ifconfig 的结果

利用 ping 命令检测网卡与局域网直接 ip 的互通性，可以看到网卡是可以正常工作的，局域网的主机之间是互通的，利用 ftp 和 http 浏览器可实现网络传输功能。

```
[root@FORLINX210]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=4.124 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=0.762 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=0.902 ms
64 bytes from 192.168.1.1: seq=3 ttl=64 time=0.782 ms
^C
--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.762/1.642/4.124 ms
```

图 4-42 与主机间的互 ping 结果

4.5.2 温度传感系统实现

本项目采用的温度传感器 DS18B20 的温度数据分辨率为 9~12 位，温度转换为 12 位数字转换时间需要 750ms[51]。

由于每个 DS18B20 的特定序列号，使得 ds18b20s 可以同时挂载在同一条总线。

因为 DS18B20 只有以条线的 I/O 接口，先进行 ROM 设定，才能进行控制设定等等。这些 ROM 设定有如读 ROM， ROM 匹配，跳过 ROM 等各项命令，具体的 ROM 指令表如图 4-44 中的 ROM 指令表所示。

若操作成功地使 DS18B20 完成温度测量，数据存储在 DS18B20 的存储器。

测量结果将被放置在 DS18B20 内存中，并有温度记忆功能，读取片上存储器储存的数据。

根据 DS18B20 的通讯协议如向下：每一次读写之前首先复位，复位完成后发送 ROM 匹配指令，接着将 RAM 指令外送进行指定的功能性操作，操作流程如图 4-31。而其中约定的指令表如表格 4-43 所示。复位的具体电平操作是，通过主机将传感器将数据线下拉 500 微秒，然后释放，收到此操作之后 DS18B20 等待 16-60 微秒，接着将 60-240 微秒的低脉冲反馈给主机，主机收到此响应说明复位完成。

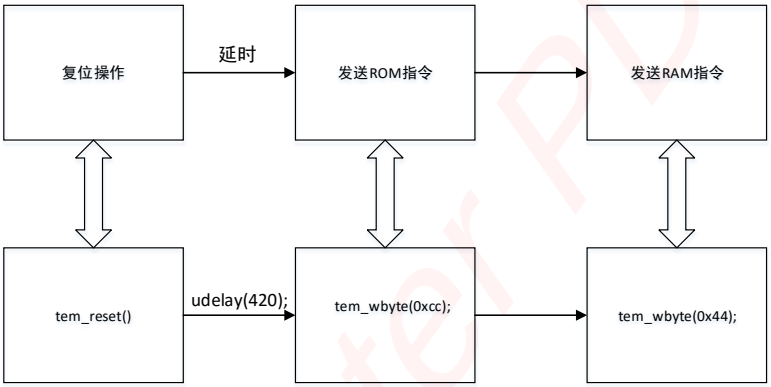

图 4-43 DS18B20 操作流程

DS18B20 的 TEMP2 数据线连接到 S5PV210 的 GPD1(2)，本驱动采用的是 9 位分辨率。之所以采用 9 位分辨率，是因为 DS18B20 的温度转换时间我们在上表中可以看到，9 位分边率需要 100ms 左右，而且因为该芯片没有握手机制，所以只能设置主机在等待温度转换完毕后从芯片寄存器中读取相应的数值；同时由于是单片温度传感器所有驱动编写时直接发送跳过 ROM 指令。

表 4-44 指令表

a.ROM 指令表

指 令	代码	功 能
读	33H	读 DS1820 温度传感器 ROM 中的编码（即 64 位地址）
符合	55H	发出此命令之后，接着发出 64 位 ROM 编码，访问单总线上与该编码相对应的 DS1820 使之作出响应，
搜索	0FOH	用于确定挂接在同一总线上 DS1820 的个数和识别 64 位 ROM 地址。为操作各器件作好准备
跳过	0CCH	忽略 64 位 ROM 地址，直接向 DS1820 发温度变换命令。适用于单片工作

告警	0ECH	执行后只有温度超过设定值上限或下限的片子才做出响应

b.RAM 指令表

指 令	代码	功 能
转换	44H	启动 DS1820 进行温度转换，12 位转换时最长为 750ms（9 位为 93.75ms）
读	0BEH	读内部 RAM 中 9 字节的内容
写	4EH	发出向内部 RAM 的 2、3、4 字节写上、下限温度数据命令以及配置寄存器的命令，紧跟该命令之后，是传送三字节的数据
复制	48H	将 RAM 中第 2、3 、4 字节的内容复制到 EEPROM 中
重调	0B8H	将 EEPROM 中内容恢复到 RAM 中的第 2、3 字节

如果采用过高的分辨率，等待时间过久，上层软件在读取温度时需要一直等待，这时因为界面刷新的原因会显得过于卡顿。所以我们采取 9 位分辨率。

我们将该温度检测接口放在 read 函数中，这样上层在调用 read 后可以直接收到转化过得温度参数，进行上层显示。

将该驱动添加到内核，编写 Kconfig 和 Makefile。

将生成的内核烧写至目标版，可以在设备目录下看到该温度检测设备，

```
[root@FORLINX210]# ls -l /dev/
crw-rw----    1 root     root     139,   0 Jan  1 08:00 DSP_HPI
crw-rw----    1 root     root     243,   0 Jan  1 08:00 TEM0
crw-rw----    1 root     root     158,   0 Jan  1 08:00 a7CfgToDetect
crw-rw----    1 root     root      10, 131 Jan  1 08:00 adc
crw-rw----    1 root     root      10, 134 Jan  1 08:00 apm_bios
crw-rw----    1 root     root      10,  62 Jan  1 08:00 ashmem
```

图 4-45 设备结构目录

编写测试程序测试温度检测设备是否工作正常，

测试结果如下

```
[root@FORLINX210]# ./temp_test
Open Device DS18B20 successed.
251.93C
251.93C
22.50C
22.50C
22.50C
22.50C
22.50C
22.50C
22.50C
22.50C
```

图 4-40 温度测试结果

当前室内温度为 21 摄氏度，与室温差在 1.5 摄氏度，温度传感器所处位置在

设备上，周围有发热设备，所以温度比室温稍高。具体在 Qt 界面中我们可以看到温度额可以正常读取，如途中红色圈框区域即为温度显示区域

4.6 本章小结

本章首先分析了 Linux 的驱动体系，并提炼了分层和隔离思想在驱动设计中的重要指导意义，最终按照该思想设计实现了重配驱动，HPI 总线驱动，以及与频率和成模块的 SPI 总线驱动，针对各项驱动用逻辑分析仪进行了详细的测试验证，保管其稳定地工作；在完成显示系统的移植之后，还进行了人机交互设备触控屏幕的驱动设计，并进行了库函数的移植，最终对触控模块进行了详细的测试。

第五章 软件平台定制与优化

在实际的运行过程中，该频谱监测软件的实际消耗如图 5-1。

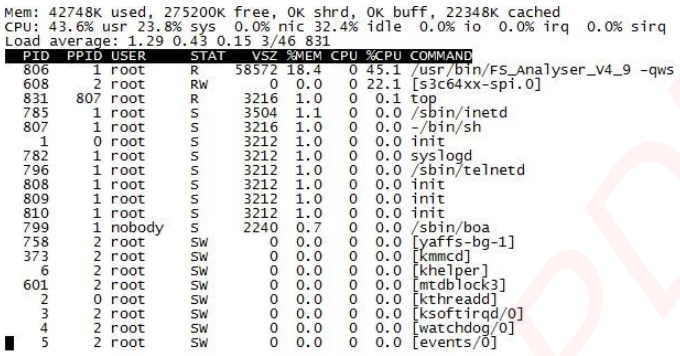

图 5-1 内存占用图

可见其资源提供达标，当然可以明显看到内核中还有其他额外的多任务其他程序，同时在主要的任务程序之外，用户占用 43.6%，系统占用 23.2%。这就设计到该嵌入式软件系统的相关优化和定制。根据之前的设计我们知道嵌入式软件系统从结构上看分为三层，驱动层包含在内核层，针对每一层，都需要有优的设计和优化，以及针对频谱监测设备的定制和处理。

5.1 U-boot 设计与定制

U-boot 是一个开源的引导程序，如果需要对其进行详细的定制，我们需要了解 U-boot 的结构。从某种意义上来说 U-boot 更像是一个简约的 Linux 内核，其中甚至包含一些硬件设备的驱动。 U-boot 含有一套完整的目录结构，每部分目录也指定了不同的应用功能 u-boot 目录结构

1.board 存放于设计板的配置文件，不同款的设计板可指定不同的目录以加以区分；

2.Commom 存放 U-boot 支持的命令

3.cpu 中存放 u-boot 支持的某些 cpu 架构目录；

4.Doc 是说明文档目

5.Drivers 存放的就是所支持设备的驱动程序；

6.Fs 是支持的文件系统；

7.Include 各类头文件；

8.Net 存放网络协议；

9.Tooles 是生成 U-boot 开发工具。

再次项目中涉及到的装对/board 和/drivers 较多，这也是最核心的两个目录之一，这些目录下的文件在移植 U-boot 及实现 U-boot 的相应功能时非常重要。

5.1.1 U-boot 启动设计

U-boot 的启动过程分成了两个分工明确的阶段，多阶段的 U-boot 可集成更加多样化的功能，也会带来良好的可移植性。

5.1.1.1 阶段一

为了阶段二和内核的载入运行，阶段一通常要完成硬件的初始化等等工作，更详细的工作要包括

1.屏蔽中断；

2.设置处理器运行速度和时钟频率；

3.内存初始化；

4.初始化 LED。一般会通过 GPIO 调节可以表示系统状态的 LED；

5.关闭 CPU 内部指令 / 数据 cache；

6.准备阶段二执行所需要的 RAM 空间，阶段二的代码一般是会被加载到 RAM 以便更快地执行，所以需要准备好一段 RAM 空间；

7.拷贝 stage2 到 RAM 中；

8.设置堆栈指针 sp ,阶段二的 C 语言代码的运行自然离不开堆栈指针；

9.跳转到阶段二的 C 执行程序入口。

完成上述帮步骤，U-boot 进入阶段二。

5.1.1.2 阶段二

阶段一已经为阶段二的运行提供了良好的环境，阶段而更多地是一些功能型和内核载入前的准备工作。

1.初始化阶段二需要的硬件设备为了方便调试和交互，需要开启一个串口与终端通信，计时器等。方便起见，点亮 LED 表明阶段二进入执行。当交互串口初始化结束，可以通过这些交互串口打印部分的信息，比如硬件的型号或者软件的版本号等等。

2.检测系统内存映射(memory map)

3.将内核映像从 flash 上读到 RAM 空间中 S5PV210 的设置是将内核拷贝至

MEM _ START 偏移 0x8000 开始，大约几兆字节大小的内存（本项目内核 4 MB 左右）。

4.为内核设置启动参数 Linux 会将内核启动参数和内核页表等启动参数存储于从 MEM START 开始大约 32KB 的存储空间内。

5.调用内核

这两个阶段完成之后，内核就开始正常的执行，进入操作系统的启动阶段，继而整个操作系统运行起来。

5.1.2 针对 U-boot 的 LCD 驱动移植

针对 U-boot 的 LCD 控制，其驱动所在文件是 uboot smdkv210 / common / lcd.c，按照目标 LCD 的参数控制进行相应的参数设置。我们同样按照如表格 4-25，在 Uboot 的内核中设置相应的参数，如下图，位置是 uboot smdkv210 / common / lcd.c。

```
71 #else
72 #define S3CFB_HFP               40          /* front porch */
73 #define S3CFB_HSW               128 //changed by niuchong 48 to 28          /* hsync
width */
74 #define S3CFB_HBP               88 //changed by niuchong 40 to 88          /* back
porch */
75
76 #define S3CFB_VFP               1 //changed by niuchong 13 to 1 /* front porch */
77 #define S3CFB_VSW               4 //changed by niuchong 3 to 4    /* vsync width */
78 #define S3CFB_VBP               23//changed by niuchong 29 to 23          /* back
porch */
79
80 #define S3CFB_HRES              800         /* horizon pixel    x resolition */
81 #define S3CFB_VRES              600//changed by niuchong 480 to 600         /* line
cnt          y resolution */
82 #endif
83
84 #define S3CFB_VFRAME_FREQ          60
```

图 5-2 U-boo 中 LCD 参数控制

最终可以看到 u-boot 下，LCD 屏幕正常输出 logo 并不再出现雪花屏幕。

5.1.3 开启 U-boot 网络支持

为了使内核调试更方便，在服务器上搭建 tftp 环境，同时移植 uboot 支持网卡，安装 tftp 工具，在 uboot 启动内核时调用 tftp 工具将内核从服务器下载至目标板的对应内存区域，完成内核启动。这样每次在内核有新的变化的时候就不需要重新拷贝，烧写，编译完成的内核可以直接在目标板上被解压执行。

U-boot 本身并不支持中断机制，所以网卡也没有中断控制。如果要控制网卡

可以正常地传输数据，我们需要做的就是其控制线 cmd 引脚的连接，有硬件原理图我们看到，本项目中的网卡控制线有 addr5 地址线控制。所以其数据寄存器相对网卡映射地址的偏移量九尾 25，即 32. 针对 210 的手持频谱仪 uboot 网卡修改，我们只需要修改 uboot 目录下的 uboot / uboot_smdkv210/ include/ configs/ smdkv210single.h，将其中的 io 和数据的地址范围按照要求修改即可。

```
122 #define CONFIG_DRIVER_DM9000      1
123 #define CONFIG_DM9000_BASE        (0x88000300)//
124 #define CONFIG_DM9000_USE_16BIT
125 #define DM9000_IO                 CONFIG_DM9000_BASE
126 #define DM9000_DATA               (CONFIG_DM9000_BASE+32) //niuchongchanged from
(CONFIG_DM9000_BASE+0x4) to 32
127 #define DM9000_16BIT_DATA
```

同时我们需要设置 U-boot 的环境变量（调用 printenv，setenv，savenv），包含网络的网关地址，掩码，以及本目标版在启动 U-boot 的局域网 IP 地址等等，具体针对此项目，进行具体的设置。

这样在 uboot 启动的时候网卡也是可以用的，设置好 tftp 的环境之后，可以用 tftp 的方式下载内核文件，方便调试。

5.1.4 U-boot 启动定制

由本项目中 S5PV210 的操作系统设计我们知道，存储区域的划分如图 3-16，其中有一项名为 LOGO，这部分区域存储的便是已经固化后的 logo 文件。而这个 logo 文件十一二进制的形式进行存储的，我们需要做的就是将 U-boot 的图片，转化为二进制文件烧写到该存储区域。

图 5-2 LOGO 制作

首先需位图格式个图片源文件，此项目采用 bmp 文件格式位源文件，利用 Image2LCD 软件进行制作。选择灰度为 16 位真彩色，图片的分辨率不能超过屏幕的最大分辨率。

拷贝生成的 logo.bin 文件至 sd 卡的 sdfuse 目录下，进入 U-boo t 的等待时间点击空格键，执行 sdfuse flash logo logo.bin 将 logo 烧写至固定存储区域内，烧写完毕后，执行 reset 命令重启系统。重启系统发现 logo 已经被修改，这里需要注意，在启动后，u-boot 会直接读取该区域额内存显现 logo 来，我们的烧写也是在 uboot 的命令行，所以一旦重新烧写，会出现花屏的现象，这时正常的，重新启动花屏消失，同时新烧写得的 logo 已经正常显现，并且不会花屏。

5.2 内核个性化定制

内核启动也是有动画的，每次内核的启动会加载一个图片。现有的开源内核多是一个小企鹅，针对定制化产品化设备，这些也是需要定制的。

内核只支持 ascii 格式的 ppm 图片文件，我们选取对应的期望的图片，在 Ub untu 主机上利用 PPM 转换工具，具体步骤如下 ubuntu 下转换 PPM：将生成的 lo go.ppm 重命名覆盖 logo2 android clut224 .ppm 。

图 5-3 格式处理

执行 make menuconfig 的时候选中替换文件，执行 make 编译内核文件，生成 zImage。 Device Drivers — Graphicsupport — Bootup logo — 选中 Standard 224-col or Android logo2 logo forlinx mod ，去掉其他的多于的选择项降低可缩减内核文件的体积。

内核中 logo 的具体位置的定制可以在具体的 logo 图片的位置定制文件中 driv ers/ video/ fbmem.c 进行修改如果想要 logo 放置在居中的位置，修改文件 drivers/ video/ fbmem.c 找到 ”fb show logo line” 函数，把 image.dx 的参数设置为 (info-var. xres/2)-(800/2)，意为水平位置相对正中央的偏移量；image.dy 的参数设置为 (info -var.yres/2)-(206/2); 意为垂直位置相对正中央的偏移量；这里 info-var.xres 和 info-

var.yres 是屏幕分辨率大小，800 和 206 是 logo 图片的和宽的像素计数。

这里的结果会使 logo 位于屏幕的上下左右的正中央，如果只是想靠上的中间，只需要修改 x 方向的偏移量。如果只需要将 logo 放在屏幕上方的正中间，跟 uboot 的启动 logo 产生一定的偏移，可以使仪器启动的过程中，这里就不需要进行定制。最终的定制结果如下：

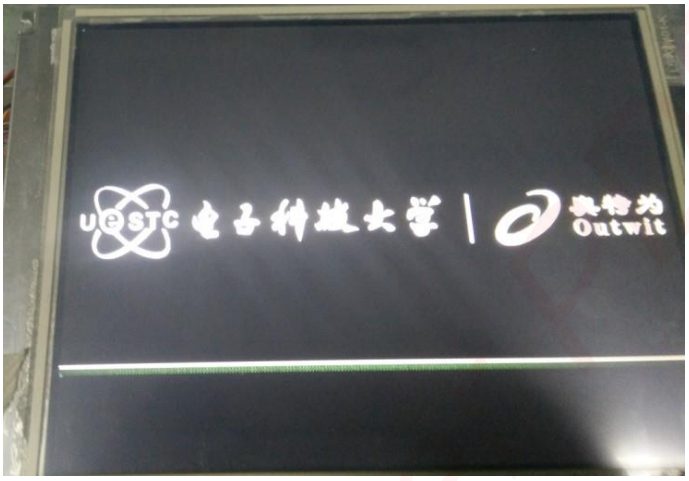

图 5-4 开机 U-boot 界面

5.3 本章小结

本章首先分析了 U-boot 的源码，并对 u-boot 进行了针对频谱检测项目的设计和定制，包括网卡，显示器，定制等等；接着对内核个各类项目也进行了详细的定制和优化，最终进行了联合的测试，达到了商品化的定制结果。

第六章 全文总结与展望

6.1 全文总结

本设计系实验室首次采用 A8 高性能 ARM 处理器 SAMSUNG 的 S5PV210 的频谱检测设备的设计与实现。

频谱监测设备的嵌入式系统与三个重要外围单元的交互：一是与 DSP 信号处理单元的 HPI 总线通信交互， Artix -7 FPGA 通信总线，三是射频控制单元通过 SPI 总线。智能嵌入式频谱检测设备的智能性硬件设计，以及电量监测，网络功能 DM9000 网卡设计，DS18b20 温度检测，串口，USB 以及 USB-OTG 的等等外围子系统的设计。在硬件平台设计搭建完毕之后，需要进行软件系统包括引导加载程序，内核和文件系统的针对性地选择规划，包括移植 U-boot 与内核，制作文件系统。

依托于 Linux 下的驱动体系结构，设计出与此三个单元交流的驱动，并在驱动设计的时候，要创新地应用面向对象的思想和驱动分层以及隔离思想进行驱动设计，最终完成三个驱动的设计与实现。同时，LCD 显示子系统的移植，以及触控屏幕的实现也是重要的工作之一，触摸屏的 USB 驱动设计与实现也是难点和亮点，该特色使得整个系统更加完善，人机交互也更完整。

另外还有功能性外设的驱动设计与实现，针对每一项外设都做了最为仔细和章程化的设计。驱动设计完成之后，还须负责对该硬件调控的上层应用小样的编写测试，将一些应用小样直接集成在文件系统中，不再需要重复拷贝或者上层应用开发者再度集成，并于上层软件开发者联合完成频谱检测软件的设计与实现。

由于本项目的频谱检测设备是一款产品级的设备，所以需要中 U-boot 到内核，再到文件系统进行详细的定制和优化，包括针对性的产品定制，内核针对性地精简优化等等。

6.2 后续工作展望

嵌入式处理器的设计架构是在不断进化的，迭代速度惊人。频谱检测设备中是离不开数字信号处理芯片的，如今的高性能 ARM 处理器往往也会在片内同时集成优秀的 DSP 模块，本设计中的 DSP 和主控芯片 ARM 是独立开来的，并且还须设计者在硬件上在读建立通道。Ti 的达芬奇架构微处理器集成一个高性能的 A

RM 核和 DSP 核心[52]，使其在信号处理领域可以将这两个模块更加集成化。会大大缩减硬件设计的难度，同时也缩小了设备的体积。

出于成本的考虑嵌入式系统平台的触控芯片选择和价位合理的性能最好的处理器，但是如果想要获得更流畅和快速的体验需要硬件上进一步的提升。加上选用集成了 DSP 的 ARM 处理器进行开发，也可以将射频前端用更加集成化的集成电路代替，频谱检测设备将会更加小型化，便携化，性能也会更好。

致谢

七年的时光，在成电成长了许多也收获了许多。从七年前的懵懵懂懂，到如今对未来的无限憧憬和充满信心，成电给予我太多太多。

在成电的三年研究生学习生活中，不仅培养了良好的科研素质和研究方法，同时养成了良好的工作作风和团队合作意识。在此，我非常感谢我的导师秦开宇教授，感谢他对我的辛勤栽培以及对我从大四以来将近四年的欢爱和关怀，他严谨的科研作风，认真的工作态度都是我学习的榜样。

特别感谢唐博老师三年多来对我的无论是在生活上，还是科研上的热心帮助和悉心指导，唐老师在整个毕业设计的选题过程中倾注了大量心血，并且提供了优越的科研环境，在整个毕业设计实现过程中提供了大量的建议和帮助，使得我能够顺利完成毕业设计。感谢张鑫师兄、周雷师兄和梁浩师兄没有与他们的共同合作，感谢师弟刘东，陈其樊在项目上的帮助，没有他们的帮助，整个项目不会进行的这么顺利；感谢林伯先老师在给我的研究生学习生涯的提点与指引，让我在空天收获了许多，成长了许多；最后要特别感谢空天 8+2 的那一帮好朋友们，这份友谊，值得珍藏和怀念一生。

特别感谢我的家人，他们不仅给予我生命，还将含辛茹苦将我们姐弟三人养大，并一一供读至硕士，博士学位，也促成了我更美好的明天，是家人的无条件支持，才让我顺利完成了学业，在此我深深地祝福我的家人，感谢我的家人。

最后，由衷的感谢各位评阅老师。

参考文献

[1] V. Lavielle, I. Lorgere, J.-L. Le Gou´ et, et al. Wideband versatile radio-frequency spectrum ana-¨ lyzer[J]. Optics letters, 2003, 28(6):384–386

[2] 班万荣, et al. 频谱分析仪的原理和发展 [J][J]. 现代电子技术, 2005, 28(7):101–102

[3] G. R. Basawapatna, V. Basawapatna, A. G. Basawapatna, et al. Apparatus for very high speed adaptive spectrum analysis[M]. Google Patents, 2015

[4] 宿绍莹, 刘平, 陈曾平. 宽带实时频谱分析技术研究与实现[J]. 电子测量与仪器学报, 2007,21(5):113–117

[5] R. A. Brown, C. Robinson. Hand-held microwave spectrum analyzer with operation range from 9 KHz to over 20 GHz[M]. Google Patents, 2012

[6] J. Crooks. Method and Apparatus for a SuperSpeed USB Bus Powered Real-Time Spectrum Analyzer[M]. 2016

[7] T. Zhang, A. Patro, N. Leng, et al. A Wireless Spectrum Analyzer in Your Pocket[M]. 2015,

[8] 魏竹, 王建忠, 邓晓莉, et al. 数字存储式频谱分析仪计量方法的优化[J]. 信息与电子工程,2012, 4:018

[9] Yepez, et al. FPGA-based multiple-channel vibration analyzer for industrial applications in induction motor failure detection[J]. Instrumentation and Measurement, IEEE Transactions on, 2010, 59(1):63–72

[10] 刘宝元, 张玉虹, 姜旭, 等. 基于单片机的温湿度监控系统设计[J]. 国外电子测量技术, 2009 (12): 77-80.

[11] 杨晖. 中国 (成都) 电子展聚焦微波射频新技术[J]. 电子技术应用, 2015, 8: 046.

[12] 王福刚, 杨文君, 葛良全. 嵌入式系统的发展与展望[J]. 计算机测量与控制, 2014, 22(12): 3843-3847.

[13] 贾天卓. ARM 系列芯片开发的前景[J]. 通讯世界, 2015 (9): 256-256.

[14] Langbridge J A. Professional embedded ARM development[M]. John Wiley & Sons, 2013.

[15] 马成. 基于 Linux 的嵌入式智能家居服务器的研究与设计[D]. 硕士学位论文]. 镇江: 江苏科技大学, 2013.

[16] 江炜宁. 数字下变频 FFT 及其在频谱分析仪中的实现[J]. 电子测试技术国家重点试验室,2007, 26(5):6–8

[17] S. Gupta, S. Abielmona, C. Caloz. Microwave analog real-time spectrum analyzer (RTSA) based on the spectral–spatial decomposition property of leaky-wave structures[J]. Microwave

Theory and Techniques, IEEE Transactions on, 2009, 57(12):2989–2999

[18] L. De Vito, S. Rapuano, M. Villanacci. Prototype of an automatic digital modulation classifier embedded in a real-time spectrum analyzer[J]. Instrumentation and Measurement, IEEE Transactions on, 2010, 59(10):2639–2651

[19] L. Q. M. Sien. Implementation of the Communication between ARM and DSP Based on HPI [J][J]. Electronic Technology, 2010, 3:023

[20] 何宗键, C. Windows. 嵌入式系统[M]. 北京航空航天大学出版社, 2006

[21] 王琼. 基于嵌入式 Linux 数据采集系统设计与实现 [D][M]. 2013

[22] XIONG J, WANG Q. Design of Tablet PC Based on S5PV210[J]. Computer and Modernization, 2012, 5: 053.

[23] L. Man, J. Liubing, T. Shiqiang, et al. Construction of radar and radar countermeasure demonstration experiment system based on DSP+ FPGA+ ARM [J][J]. Journal of Guilin University of Electronic Technology, 2011, 6:008

[24] Li J, He Y, Zou Z, et al. Design of an Embedded Multi-biometric Recognition Platform Based on DSP and ARM[M] Biometric Recognition. Springer International Publishing, 2014: 426-433.

[25] X.-P. Zhu, T.-J. Xiao, H. Zhao. System design of real-time data acquisition based on ARM and FPGA [J][J]. Computer Engineering and Design, 2009, 13:018

[26] 何健. 基于 DS18B20 传感器测温系统的设计与实现[J]. 湖南农机: 学术版, 2013 (2): 66-67.

[27] Fan C, Li Z, Ding Q, et al. Design of Embedded Ethernet Interface Based on ARM11 and Implementation of Data Encryption[M]//Intelligent Data analysis and its Applications, Volume I. Springer International Publishing, 2014: 431-439.

[28] FU X W, ZHANG J, WANG Y. Char Drive Procedures Based on SPI Bus Protocol [J][J]. Computer Systems & Applications, 2013, 2: 035.

[29] 郭晓博, 赵敏, 乐珺. 基于 USB-OTG 的能谱仪通讯系统设计[J]. 机械制造与自动化, 2015, 44(1): 155-157.

[30] Weiyan X, Dong L, Ming L, et al. Autonomous Recovery Technique of Software Bus Based on VxWorks Operating System[C]//Proceedings of the World Congress on Engineering. 2014, 1.

[31] 王翔. 基于 WINCE 平台的便携式 429 总线测试仪的设计[J]. 机电产品开发与创新, 2014 (3): 137-138.

[32] 张童. 关于 Linux 发展历程与相关应用的研究[J]. 移动信息, 2015 (2): 65-65.

[33] 原尧燊, 原峰山, 姜充. 嵌入式系统开发中 U—boot 移植中两个关键参数的配置[J]. 重庆工商大学学报: 自然科学版, 2013, 30(10):73–77

[34] 朱永华, 沈熠, 刘玲. Linux 内核完全公平调度器改进的研究[J]. Computer Engineering and

Applications, 2014, 50(21).

[35] 王兆文, 蒋泽军, 陈进朝. 一种提高 Linux 内存管理实时性的设计方案[J]. 计算机工程, 2014, 40(9): 291-294,299

[36] 戴明华, 李长云, 曾志浩, 等. 嵌入式 Linux 驱动程序框架研究综述[J]. 长沙大学学报, 2012, 26(2): 52-53.

[37] 宁玉玲, 陈琼, 马扬龙. Linux 设备驱动模型框架的分类研究[J]. 现代电子技术, 2013, 36(4): 5-8.

[38] 秦晓康, 徐惠民. 嵌入式设备 NAND Flash 存储系统的设计与实现[J]. 计算机工程与设计,2010(3):514–517

[39] 曹璐. 基于 NAND FLASH 的文件系统设计与实现[D]. 华东师范大学, 2012.

[40] 姜远志. Linux 的驱动开发分析[J]. 无线互联科技, 2014 (1): 103-103.

[41] 李全. 基于 Linux 的 USB 网络设备驱动程序的研究 [D][M]. 2010

[42] 董国通, 周子健. 基于嵌入式 Linux 的视频采集系统设计[J]. 仪表技术, 2015, 11: 004.

[43] 董磊, 马桂芳, 李清东. 卫星电源系统的有向图分层诊断方法[J]. 计算机应用, 2013, 32(A02): 38-40.

[44] 聂和平. 基于 ARM9 的嵌入式 Linux 系统移植与驱动开发 [D][D]. 南京邮电大学, 2013.

[45] 高非非, 刘辛国. ARM—Linux 中 I^ 2C 总线驱动开发[J]. 微型机与应用, 2012, 31(5): 57-58.

[46] H. Kavianipour, S. Muschter, C. Bohm. High performance FPGA-based DMA interface for PCIe[J]. Nuclear Science, IEEE Transactions on, 2014, 61(2):745–749

[47] Y. F. Z. Yongqiang. The Development of Ethernet Chip Driver Based on U-Boot [J][J]. ElectronicTechnology, 2008, 3:013

[48] FU X W, ZHANG J, WANG Y. Char Drive Procedures Based on SPI Bus Protocol [J][J]. Computer Systems & Applications, 2013, 2: 035.

[49] 曾小波, 黄建华, 李竞雄. 基于 ARM 微处理器的触摸屏控制器设计及实现[J]. 湖南工程学院学报: 自然科学版, 2015, 25(1): 37-40.

[50] 高嵩, 纪超, 陈超波. 基于嵌入式 Linux 的 DM9000 网络驱动设计[J]. 计算机与数字工程, 2013, 41(2): 304-306.

[51] J 吕建波. 基于单总线数字温度传感器 DS18B20 的测温系统设计[J]. 现代电子技术, 2012, 35(19): 117-119.

[52] 刘慧念. TI 达芬奇软件框架技术的研究与改进[J]. 电信科学, 2009, 6: 51-54.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计

12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备