Nand Flash 启动模式下的 Uboot 移植

杨素秋

（常州信息职业技术学院，江苏 常州 213164）

摘　要：BootLoader 移植是嵌入式系统开发中的一个重要环节，而 Uboot 是一个支持多处理器多操作系统较为通用的 BootLoader。分析了 Uboot 的启动过程，并对 s3c2440A 如何支持驱动代码在外部 Nand Flash 上的执行，以及在 s3c2440A 平台上如何实现 Nand Flash 启动模式下的 Uboot 移植进行了介绍。具体操作和编译在 fedora9 操作系统和 arm-linux-gcc4.3.2 交叉编译下完成。

关键词：嵌入式系统；BootLoader；Nand Flash；Uboot；s3c2440A

中图分类号：TP301　　**文献标识码**：A　　**文章编号**：1672-7800(2013)003-0033-04

0　引言

BootLoader 是指在操作系统内核运行之前运行的一段程序，其功能在于完成硬件设备的初始化、建立内存空间的映射图、将系统的软硬环境带到一个合适的状态，为最终的内核调试做好准备。Uboot 是在 ppcboot 以及 ARMboot 的基础上发展而来的较为通用的 BootLoader，不仅仅支持 Linux 嵌入式操作系统的引导，还支持 VxWorks、QNX 等嵌入式操作系统，Uboot 除了支持 PowerPC 系列处理器，还能支持 ARM、x86、XScale、MIPS 等诸多常用的处理器。

目前，Uboot 的移植解决方案主要面向的是微处理器中的 Nor flash。但 Nor flash 存储器的价格比较昂贵且存储容量小，而 SDRAM 和 Nand flash 存储器的价格相对来说比较合适且容量大，如果能在 Nand Flash 中实现 Uboot 的启动并在 SDRAM 中执行主程序，则不仅能够有效地降低系统的成本，也能给实际应用带来极大的方便。

1　Uboot 启动过程分析

BootLoader 的启动通常有两个阶段，分为阶段 1（stage1）和阶段 2（stage2），Uboot 也不例外。在 Uboot 中依赖于 CPU 体系结构的代码（如 CPU 初始化代码等）通常都放在阶段 1 中且用汇编语言实现，而为了系统有更好的移植性和可读性，在阶段 2 中通常用 C 代码来实现。关于 CPU 的初始化文件一般放在第一阶段，其对应的文件目录为 cpu/arm920t/start.s，主要实现一些寄存器的配置以及初始化、内存和栈空间的配置。阶段 2 在 lib_arm/board.c 中，其基本由 C 语言实现，start_armboot 是整个启动代码中 C 语言的主函数，在其中通过调用各个硬件设备的初始化函数来完成本阶段要使用到的硬件设备的初始化，除此之外还涉及到内存的映射、代码的搬运、设置内核启动参数等功能。具体实现流程如图 1 所示。

[5] 刘瑞，许峰. 基于 ε 支配擂台赛法则的多目标遗传算法[J]. 软件导刊，2012(8).

[6] 邓金华，蒋浩，邝达，等. 用擂台赛排挤算法构造多目标最优解集的方法[J]. 软件学报，2007(6).

[7] 梁浩，林丹，马楠. 基于 ε 支配的自适应多目标进化算法[J]. 计算机工程与应用，2011(34).

（责任编辑：孙　娟）

MOGA Based On ε-Dominated Method for knapsack problems

Abstract: Multi-objective genetic algorithm NSGA-Ⅱ is an effective mean to solve 0/1 knapsack problem, but it has certain defects when scales of 0/1 knapsack problems is large and this method is hard to convergence to the pareto optimal set boundary and unsatisfactory distribution. In response to this problem, we propose ε dominated MOGA to solve 0/1 knapsack problem. The experiments demonstrate that this method is superior to compare to NSGA-Ⅱ.

Key Words: Multi-objective genetic algorithm; ε dominated; 0/1 knapsack problem

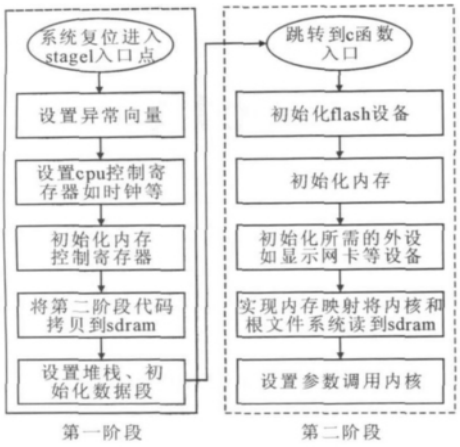

图 1　Uboot 启动流程

2　Uboot 在 Nand Flash 中的启动

Nand flash 的地址是非线性的，也就是说不能直接在芯片的内部执行程序。但为了满足这一要求，s3c2440A 配备了一个内部 SDRAM 缓冲器，叫做‘Steppingstone’，当系统启动时，Nand Flash 存储器的前 4k byte 数据将被自动载入到 steppingstone 中，并且装载到 steppingstone 中的代码会被执行。执行过程如图 2 所示。

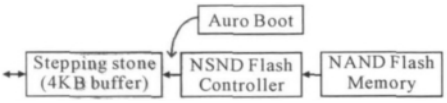

图 2　Nand Flash 的自动装载模式

由上图可知，Uboot 下载到 Nand Flash 中当系统加电时，Nand flash 存储空间的前 4kb 的数据也就是 Uboot 的第一阶段会自动加载到 SRAM 中并执行，运行到第二阶段时则把 Nand Flash 中的 Uboot 全部拷贝到 SDRAM 中继续执行。

3　Uboot 的移植及实现

Uboot 一直都没有支持 S3C2440，所以以 SBC2410 为蓝本进行修改和移植。2410 和 2440 基本上没有什么大的差异，主要区别是 2440 的主频变得更高，在接口方面增加了摄像头接口和 AC97 音频接口；在寄存器方面，除了新增模块的寄存器外，Nand Flash 的控制寄存器也有了较大的变化。下面对移植的过程进行介绍。

3.1　开发环境

宿主机的软件开发环境：Fedroa9. 0、arm-linux-gcc-4. 4. 3. tar。其安装路径为/usr/local/arm/4. 3. 2、uboot-2009-11。

目标开发板硬件环境：S3C2440A、SDRAM 为 K4S561632N-LC75、Nand Flash 为 256M 的 K9F2G08U0B。

3.2　建立属于自己的 board 平台

参照 board/sbc2410x 目录，在源码目录 board 下建立自己的开发平台 mine2440，步骤如下：

（1）这里将板子取名为 mine2440，建立 board/mine2440 目录，拷贝 board/sbc2410x 下的文件到 board/mine2440 目录，将 sbc2410x. c 更名为 mine2440. c.

```
mkdir-p mine2440 //进入 board 目录
cp-arf sbc2410x/* /board/mine2440
mv sbc2410x. c mine2440. c //进入 mine2440 目录
```

（2）修改 mine2440 目录下的 Makefile 文件在 Linux 中使用 gedit 命令，将 COBJS := sbc2410x. o flash. o 改为 COBJS := mine2440. o flash. o

```
gedit board/mine2440/Makefile
```

（3）修改 Uboot 根目录下的 Makefile 文件。查找到 smdk2410_config 的地方：

```
mine2440_config := unconfig
@ $（MKCONFIG） $（@: _config =） arm arm920t mine2440 NULL s3c24x0
```

，其各个参数的定义如表 1 所示。

表 1　各参数的定义

Arm	cpu 的架构
arm920t	cpu 的类型，对应 cpu/arm920t 目录
mine2440	开发板的型号，可自定义
NULL	开发者/或经销商
s3c24x0	片上系统定义

添加完开发板的定义，还要指定其交叉编译器。

（4）测试编译环境如下：

```
[root@localhost u-boot-2009.11]# make distclean
[root@localhost u-boot-2009.11]# make mine2440_config
Configuring for mine2440 board...
make
arm-linux-objcopy -O srec u-boot u-boot.srec
arm-linux-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
```

Make 成功则说明编译环境和开发板的创建添加都没有问题。下面针对代码执行流程对创建的开发板做出一些相应的修改。

3.3　在 Uboot 中添加对 S3C2440 一些寄存器的支持

由于 2410 和 2440 的寄存器及地址大部分是一致的，所以这里就直接在 2410 的基础上再加上对 2440 的支持即可，即在定义了 CONFIG_S3C2410 的地方加上 CONFIG_S3C2440，再根据两者的不同添加功能设置。

3.3.1　系统时钟配置

2440 的最高主频能达到 503MHz，但正常情况下让其运行在 400MHz，而 2410 只有 200MHz，所以需要对系统时钟部分做出更改。主要包括主时钟频率即 MPLL 与 USB 时钟频率 UPLL，下面分别给出具体的计算式：

```
MPLL=（2*m*Fin）/（p*2^s）/ * 主时钟频率
UPLL=（m*Fin）/（p*2^s）/ * USB 控制频率
```

其中 m、p、s 的值可以自己设置，用来控制各个模块的时钟输出频率。在 2440 系统中使用 12MHz 的晶振也就是公式里的 Fin。下面分别给 m、p、s 赋值具体实现如下：

```
#define S3C2440_MPLL_400 MHz（（0x5c<<12）|（0x01<<4）|（0x01））
#define S3C2440_UPLL_48 MHz（（0x38
```

<< 12) | (0x02 << 4) | (0x02))

#define S3C2440_CLKDIV 0x05 / * FCLK : HCLK : PCLK =1 : 4 : 8 , UCLK = UPLL

3.3.2 串口配置

2440 带有 3 个串口,在本系统中使用 uart0 来与上位机建立通信。串口时钟用 pclk 时钟,即 1/2hclk 时钟,1/8fclk 时钟。可以通过对寄存器 ULCON0 与 UFCON0 的配置,来对 uart0 的工作模式进行设置。串口通信中波特率的设置很重要,在此选择 115200 作为调制速率,具体实现可以通过给寄存器 UBRDIV0 赋值来实现,具体的设置值可以通过以下计算公式进行计算:

UBRDIVn=(int)(UARTclock/(buadrate * 16))-1。其中,UART clock 为 pclk 时钟,buard rate 为选择的波特率值。

3.4 实现 Nand Flash 启动

(1)在 Uboot 中添加对 mine2440 开发板 Nand Flash 的支持。目前,Uboot 中还没有对 2440 上 Nand Flash 的支持,也就是说想要 Uboot 从 Nand Flash 上启动就得自己去实现。首先在 include/configs/mine2440.h 头文件中定义 NF 要用到的宏和寄存器,主要包括如下内容:

```
#define CONFIG_S3C2440_NAND_BOOT //支持从 Nand Flash 中启动
#define CONFIG_UBOOT_SIZE 0x60000
//定义存放 uboot 代码段的大小
#define NAND_CTL_BASE 0x4E000000
//Nand Flash 配置寄存器基地址,查 2440 手册可得知
#define STACK_BASE 0x33F00000 // 定义堆栈的地址
#define STACK_SIZE 0x8000 //堆栈的长度大小
#define oNFCONF 0x00 // 相对 Nand 配置寄存器基地址的偏移量,还是配置寄存器的基地址
#define oNFCONT 0x04
// 相对 Nand 配置寄存器基地址的偏移量,可得到控制寄存器
的基地址(0x4E000004)
#define oNFADDR 0x0c
// 相对 Nand 配置寄存器基地址的偏移量,可得到地址寄存器
的基地址(0x4E00000c)
#define oNFDATA 0x10
// 相对 Nand 配置寄存器基地址的偏移量,可得到数据寄存器的基地址(0x4E000010)
#define oNFCMD 0x08
// 相对 Nand 配置寄存器基地址的偏移量,可得到指令寄存器
的基地址(0x4E000008)
#define oNFSTAT 0x20
//相对 Nand 配置寄存器基地址的偏移量,可得到状态寄存器的基地址(0x4E000020)
#define oNFECC 0x2c
//相对 Nand 配置寄存器基地址的偏移量,可得到 ECC 寄存器的基地址(0x4E00002c)
```

(2)修改 cpu/arm920t/start.S 文件,因为 u-boot 的入口程序是/cpu/arm920t/start.S,故需在该程序中添加 Nand Flash 的复位程序,以及实现从 Nand Flash 中把 u-boot 搬移到 RAM 中的功能程序。u-boot 默认是从 Nor Flash 启动的,所以首先屏蔽掉 u-boot 中的从 Nor Flash 启动部分。在添加 2440 中 u-boot 从 Nand Flash 启动的部分,代码如下:

```
#ifdef CONFIG_S3C2440_NAND_BOOT
//copy U-Boot to RAM
ldr r0, =TEXT_BASE
…… //这里初始化 nandflash 控制器,
代码省略
ldr sp, DW_STACK_START
//设置堆栈指针,为调用 C 语言函数作准备,在其后有定义
mov fp, #0 //初始化帧指针寄存器
//传递给 C 代码的第一个参数:u-boot 在 RAM 中的起始地址
mov r1, #0x0
// 传递给 C 代码的第二个参数:Nand Flash 的起始地址
mov r2, # CONFIG_UBOOT_SIZE
// 传递给 C 代码的第三个参数:u-boot 的长度大小(384k)
bl nand_read_ll
// 此处调用 C 代码中读 Nand 的函数,将整个 uboot 代码读到 ram
…….. //这里验证所读取的代码,代码省略
ok_nand_read:
// 检查搬移后的数据,如果前 4k 完全相同,表示搬移成功
notmatch:
loop3: b loop3 //当不匹配时在此循环
#endif
在__start_armboot: .word start_armboot
下面加上 DW_STACK_START 的定义
DW_STACK_START: .word STACK_BASE+STACK_SIZE-4
```

(3)在上面启动代码中添加对 Nand Flash 的支持中调用了 nand_read_ll 函数,它是跳入新增的 C 语言文件 nand_read.c,将其新建在 board/samsung/mine2440/目录下,这个文件参考的 vivi 代码在其基础进行了修改并添加了对本开发板所用的 Nand Flash 的支持。因为 S3C2440 和 S3C2410 之间的差别就是:S3C2410 的 Nand Flash 控制器只支持 512B+16B 的 Nand Flash,而 S3C2440 还支持 2KB+64B 的大容量 Nand Flash。所以在 Nand Flash 控制器上寄存器和控制流程方面的差别很明显。

首先根据用户手册在 nand_read.c 中定义 nandflash 控制器和各个寄存器的指针目标:

```
#define NF_BASE 0x4e000000
#if defined (CONFIG_S3C2440)
…… //根据数据手定义其偏移地址具体定义在此省略
#define NFSTAT_BUSY 1
#endif
```

再在 nand_read.c 中加入 nand_read_ll 函数的实现。这里给出详细的设计和伪程序，具体代码：

```
int nand_read_ll (unsigned char * buf, unsigned long start_addr, int size)
{
/* 参数说明 */
/*  buf:指向 ram 中的目标地址 */
/*  start_addr:所读内容在 nandflash 中的起始地址 */
/*  size:复制的总字节数 */
/* 主要操作步骤 */
/* 1:验证 nandflash 地址对齐 */
/* 2:打开 nandflash 芯片使能信号 */
/* 3:从 nandflash 首页开始(start_addr),逐页读出 size 个字节数并放置于 ram 的指定位置(buf)处 */
/* 4:关闭 nandflash 芯片使能信号 */
……
return 0;
}
```

最后添加我们使用的 nand flash 的 ID 识别程序代码，具体实现如下：

```
if (nand_id == 0xecda )
{ /* Samsung K9F2G08U0B 我们使用的型号大小为 256MB */
nand.page_size = 2048;
nand.block_size = 128 * 1024;
nand.bad_block_offset= nand.page_size;
}
```

在完成上面的工作后需要修改 board/my2410/Makefile，文件找到：OBJS := my2410.o flash.o 修改为 COBJS:=mini2440.o flash.o nand_read.o

(4)最后还有一个重要的地方需要修改，在 cpu/arm920t/u-boot.lds 中，这个 u-boot 启动连接脚本文件决定了 u-boot 运行的入口地址，以及各个段的存储位置，这也是链接定位的作用。添加下面两行代码的主要目的是防止编译器把添加的用于nandboot的子函数放到4K之后，否则无法启动。如下：

添加 cpu/arm920t/start.o (.text)

board/samsung/mini2440/lowlevel_init.o (.text)

board/samsung/mini2440/nand_read.o (.text)

在完成以上工作时再次通过 make mine2440_config 和 make 命令将 Uboot 编译成 uboot.bin 文件并将其下载到开发板中，通过串口可以看到如下信息，如图 3 所示。

```
U-Boot 2009.11

DRAM:  64 MB
Flash:  2 MB
NAND:  256 MiB
In:    serial
Out:   serial
Err:   serial
Net:   CS8900-0
```

图 3 串口调试信息

由调试信息可知，Uboot 在 Nand Flash 中的启动已经实现，并且 Uboot 成功地识别了 Nand Flash 的大小。

4 结语

Uboot 移植是一个非常复杂的任务，Uboot 对 Nand Flash 闪存的支持不仅体现在 Nand Flash 启动 Uboot 上，还体现在 Uboot 对 Nand Flash 进行常用操作的 Nand Flash 命令上。本文在 Uboot 完全移植中只是完成了一小部分，后续仍有许多工作有待完成。

参考文献：

[1] 刘淼. 嵌入式系统接口设计与 Linux 驱动程序开发[M]. 北京：北京航空航天大学出版社，2006.

[2] 黄卫军，苗放，郗诚. Uboot 在 S3C2410 系统上的移植与应用[J]. 铁路计算机应用，2008(8).

[3] 芦伟，潘炼. uboot 在 s3c2440 上的移植[J]. 微型机与应用，2010(4).

(责任编辑：孙 娟)

Porting of Uboot for Booting from NAND Flash Mode

Abstract: Porting of BootLoader is a very important part in the embedded system development process. Uboot is a general bootloader, it supports multiple processors and multiple operating system. This paper analyzes booting process of uboot, and introduce s3c2440A how to support drive codes implement in the external NAND Flash , the end , gives ideas and implement methods about achieving booting uboot from NAND Flash based on S3C2440A platform. The specific operation and compile are finished in fedora 9 and arm-linux-gcc 4..3.2 cross compiler environment.

Key Words: Embedded Systems; Bootloader; NAND Flash; Uboot; S3C2440A

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
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计

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
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

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
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版

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
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

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
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植

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
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现

12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计

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
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究

Programming:

1. 计算机软件基础数据结构 - 算法