基于嵌入式 Linux 的 SD卡驱动程序的设计与实现

邵自然 吕格莉

中南大学信息科学与工程学院 湖南 长沙 410083)

摘 要：SD卡和 Linux 操作系统以其优越的性能在嵌入式设备上得到了越来越广泛的应用。文章介绍了 Linux下的设备驱动程序和 SD卡的操作方法，设计和实现了 Linux下基于uP X 处理器的SD卡驱动程序。

关键词：Linux驱动程序；SD卡；嵌入式系统

引言

SD存储卡（Secure Digital Memory Card）是由日本松下公司、东芝公司和美国 SANDISK公司在 MMC卡的基础上共同开发研制的一款具有大容量、高性能、安全性好等特点的多功能存储卡。它在数码相机、MP3、掌上电脑和手机等便携式设备上使用非常广泛。随着处理器能力的提升以及系统复杂度的提高，更多嵌入式系统选择使用操作系统来提高开发速度、降低开发风险和提高系统的稳定性。Linux操作系统因为其内核小、开源和可裁剪等优点，在嵌入式设备中得到了广泛的应用。在 Linux下使用 SD卡，需要开发和加载 SD卡的设备驱动程序。SD卡作为 Linux下面一个目录，从而用户和应用程序可以不用关心具体细节对 SD卡进行存取操作。本文分析了 Linux驱动程序机制和 SD卡协议，并在针对 PXA255处理器的 linux-2.4.19内核基础上，实现了 SD卡的驱动程序。

1 Linux的设备驱动程序

驱动程序位于操作系统的最底层，它屏蔽了具体的硬件细节，是整个操作系统的基础。在 Linux中，驱动程序实际上是内核的一部分。驱动程序可以在编译内核的时候，静态地链接进内核，在操作系统初始化的同时完成驱动程序的初始化工作；也可以在系统运行的时候，以可加载模块的方式加载进内核，由模块的初始化函数来完成初始化工作。

由于设备种类繁多，为了管理方便，Linux把它的驱动程序分成三大类：字符设备、块设备和网络设备。其中网络设备是通过 BSD socket接口访问的设备，它负责数据包的发送和接收，通常不作为存储设备。

字符设备能够存储或者传输不定长的数据。内核把字符设备看成是可访问的连续字节流，它在单个字符的基础上接受和发送数据。字符设备不能够随机访问，也不允许查找操作。常见的字符设备有终端、打印机、串口、鼠标、声卡等等。

块设备中通常存储的是定长且可随机访问的数据块，对它的 I/O操作只能以块为单位进行。Linux的文件系统通常在块设备之上。

2 SD卡的操作

2.1 SD卡的总线协议

SD存储卡的协议是一种问答式的协议。首先主端发送 CMD，卡端接收回应 URXE2S。如果有数据需要传送就会在 DATA线出现数据。SD卡有 34个命令（不包含对于版权保护的几个命令），其中 26个基本命令和 8个专用命令。在 CMD线上发送的 CMD命令格式是确定的，图 1对命令格式进行了描述。

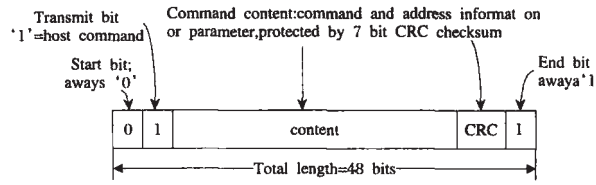

图 1 SD的命令格式

SD卡的回应格式分为 4种，分别是 R1、R2、R3和 R

呼、群呼等功能并具备对讲的图像显示、对讲呼叫优先显示功能。对讲分机前端营业网点实行对讲呼叫联动功能，对讲双方一旦呼叫就会立即显示对讲分机前端的图像。

网点将通过光端机的专用音频通道实现双向对讲。对于 2M SDH组网的网点需在对讲主机和分机端分别增设音频编解码器 OB9966D，负责将发送的对讲音频信号转换成 IP语音信号并将接收到的 IP语音信号还原成对讲音频信号，从而实现双向对讲。

结束语

银行远程数字集中监控系统设计方案综合了国内外的先进技术，实现了银行安保部门对各储蓄所、分理处、支行等营业场所防护区域实时图像、声音、报警数据的远程监视、监测、控制功能，完全达到了银行与公安系统建立联防、联建体制的目标，并能够最大程度地保障用户原来的设备投入。系统定能增强银行系统自身的安全防范能力，提高银行系统的服务品质，使银行安全防范体系更加完善。

参考文献：

[1] 中国人民共和国国家标准．银行营业场所安全防范工程设计规范．GB/T16677-1996．

[2] 中国人民共和国国家标准．视频安防系统技术要求．GB/T367-2001．

[3] 通信行业标准．电信网光纤数字传输系统工程施工及验收暂行技术规定．YDJ44-1989．

[4] 中国人民共和国国家标准．报警图像信号有线传输装置．GB/T6677-1996．

了命令 CMD0(GO_IDLE_STATE)以外,其他的每个命令都会有一个特定的回应。如 CMD17(READ_SINGLE_BLOCK)使用 R1回应,该回应格式如图 所示,它包含一个起始位、传输位、命令号、卡的状态信息、循环冗余位和结束位。其中命令号对应它所回应的命令,状态信息表示卡的当前状态。其它几个回应的格式与 R1相似,区别在于长度和携带的信息有所不同。

位置	47	46	[45:40]	[39:08]	[07:01]	00
宽度	1	1	6	32	7	1
值	"0"	"0"	X	X	X	"1"
说明	Start bit	Transmission bit	Command index	Card status	CRC7	End bit

图 2 R1回应格式

2.2 SPI寄存器

对 SD卡的配置过程,实际上是对 SD卡寄存器的读写过程。其中, CID(Card Identification) CSD(Card-Specific Data)和 OCR(OCR operation conditions register)寄存器是在操作过程中非常重要的寄存器。

CID寄存器包含了卡的厂商和版本等信息,驱动程序可以通过这个寄存器来判别是否能够对该卡进行支持。

CSD寄存器则包含了卡的特殊信息,包括卡的容量、块大小以及是否写保护等信息。只有取得了该寄存器的值,驱动程序才有足够的信息注册驱动程序。

OCR寄存器主要包含了卡的操作电压信息,通过读写该寄存器,驱动程序可以设置卡的工作电压。

2.3 SPI操作过程

通过向卡发送不同的命令,可以将卡从一个状态转移到另外一个状态,同时获取我们需要的数据。图 描述了卡初始化的状态转移关系图。其中 CMD0(GO_IDLE_STATE)没有返回数据, ACMD41(SD_SEND_OP_COND)会通过 R1回应返回 OCR信息, CMD2(ALL_SEND_CID)通过 R1回应来返回 CID信息, CMD3(SEND_RELATIVE_ADDR)则通过 R1回应卡的 RCA(Card Address)信息。

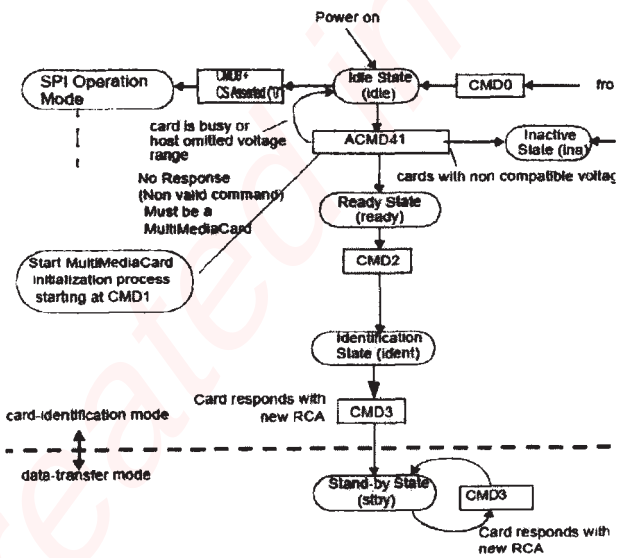

图 3 SD卡初始化的状态图

初始化结束之后,卡进入 standby状态,这时,可以发送 CMD9(SEND_CSD)来得到包含 CSD信息的 R1回应,之后通过发送 CMD7(SELECT/DESELECT_CARD)来进入读写状态。在读写状态,驱动程序通过发送 CMD17(READ_SINGLE_BLOCK)、 CMD18(READ_MULTIPLE_BLOCK)以及 CMD24(WRITE_BLOCK)和 CMD25(WRITE_MULTIPLE_BLOCK)来进行读写。

3 Linux的 SPI驱动程序实现

3.1 SPI控制器的操作

由于 SD卡的时序兼容 MMC卡,我们采用带 MMC卡控制器的 PXA255芯片作为主控处理器。 PXA255是 Intel公司生产的基于 XScale微架构的处理器,属于 ARM v5TE体系结构。

驱动程序可以通过读写 PXA255内置的 MMC卡控制器的寄存器和 FIFO启动一次与卡的通信。通过设置寄存器,可以发送命令和参数、设置接受的回应格式、是否发送/接受数据以及是否产生同步时钟等 SPI卡操作所需要的功能。

该控制器还可以进行 DMA操作,将控制器的 FIFO作为 DMA控制器的目标或源,可以实现后台的数据传输,从而提高系统效率。

3.驱动程序实现

SD卡的读写必须以块为单位。本文将驱动程序实现为块设备,其上挂载文件系统。

在块设备被使用之前,它必须向操作系统内核注册,告诉内核自己实现的文件操作集。为此, Linux为每个块设备准备了一个 device_struct结构,允许块设备驱动程序在其中登记自己实现的文件操作集。

文件操作集包含设备的打开、关闭、 I/O控制、请求操作。其中打开和关闭函数用来打开和关闭电源,准备数据结构等等; I/O控制函数则实现用户程序对设备的直接 I/O操作例程;请求操作 (request)函数则完成数据在操作系统的块设备缓冲区到设备之间的数据传输。

SD卡是可拔插的设备。在系统启动的时候,卡片可能并不在系统当中,因此,本文在实现驱动程序的时候,将它分为总线检测模块和 SD卡操作模块。总线检测模块静态编译进内核,在系统初始化时就注册中断服务程序,当 SD卡插入中断发生之后,就把 SD卡操作模块加载进内核。 SD卡操作模块则以可加载模块的方式存在,主要需要完成驱动程序的注册、卸载和 Request函数。

总线检测模块检测到 SD卡插入系统后,会调用 SD卡操作模块的模块初始化函数,从而完成驱动的注册。下面是初始化的部分代码:

```
static int sd_blk_init(void)
{ int *sd_sizes;
  int *sd_blk_sizes;
  int *sd_hard_sects;
  sd_card_go_idle(); 重置 SD卡
```

```
sd_op(); 获取并设置 SD卡操作电压信息
sd_card_get_cid(); 获取 SD卡的 CID
sd_card_get_rca(); 取得 SD卡的 RCA寄存器信息
sd_card_get_csd(); 取得 SD卡的 CSD
sd_card_select(); 使卡处于待命状态
major = register_blkdev (0,"mmc_ disk", &mmc_ops);
if(major < 0)
  return -1;
  注册块设备驱动程序
blk_init_queue(BLK_DEFAULT_QUEUE(major),sd_request);
blk_dev[major].request_queue.request_fn = sd_request;
  设置块设备的请求传输函数
read_ahead[major] = 10;
sd_sizes = kmalloc(3*sizeof(int),GFP_KERNEL);
mmc_blk_sizes = mmc_sizes + 1;
mmc_hard_sects = mmc_sizes + 2;
*mmc_sizes = sd_info.size/1024;
*sd_blk_sizes = 1024;
*sd_hard_sects = sd_info.hard_sect;
blk_size[major] = sd_sizes;
blksize_size[major] = sd_blk_sizes;
hardsect_size[major] = sd_hard_sects;
  设置设备管理的设备大小、扇区大小信息
return 0;
}
```

注册的主要工作是初始化 SD卡，向内核注册块设备以及设置块设备的信息。注册之后， Linux在有数据要进行读写的时候，会调用驱动程序的 request函数。下面是该函数的伪代码：

```
static void mmc_request(request_queue_t *q)
{ unsigned long nr;
  int blk;
  while(1)
  { INIT_REQUEST; 检查缓冲队列是否为空
    switch(CURRENT->cmd) {
    case READ:
      for(nr=CURRENT->current _nr_sectors,blk=0;nr>0;nr--)
      { mmc_read_singl_block(CURRENT->buffer+blk*512,
          CURRENT->sector+blk);
        blk++;
      }
      break;
    case WRITE:
      for(nr=CURRENT->current_nr_sectors,blk=0;nr>0;nr--)
      { mmc_write_singl_block(CURRENT->buffer+blk*512,
          CURRENT->sector+blk);
        blk++;
      }
      break;
    default:break;
    }
    end_request(1);
      完成一个缓冲块的传输
  }
  return;
}
```

当用户要拔出 SD卡的时候，模块的卸载函数则会被调用，由该函数来完成驱动程序的卸载和资源的释放。下面是卸载部分的代码：

```
static void mmc_blk_exit(void)
{ fsync_dev(MKDEV(major,0));
  unregister_blkdev(major,"mmc_disk");
    卸载驱动程序
  blk_cleanup_queue(BLK_DEFAULT_QUEUE(major));
    清楚队列然后释放申请的资源
  kfree(blk_size[major]);
  blk_size[major] = NULL;
  blksize_size[major] = NULL;
  hardsect_size[major] =NULL;
  return;
}
```

将驱动程序加载进内核之后，将 SD卡插入系统，驱动程序就会自动将它初始化。用户可以使用 mkfs命令来将其格式化，然后使用 mount命令将其加载到指定目录。也可以建立一个专门的守护进程，发现卡插入之后，自动地 mount到指定的目录，从而实现即插即用的功能。

结束语

本文对 Linux驱动程序和与其相关的 SD卡底层操作进行了介绍，并在此基础之上，开发了嵌入式 Linux下的 SD卡驱动程序。用户程序可以通过 Linux本身提供的文件系统对 SD卡进行访问，并实现了 SD卡的即插即用。本文没有在 SD卡的文件系统、版权保护等高级应用上进行分析。要实现 SD卡的全部功能，还需要在文件系统和用户程序上进行支持。

参考文献：

[1] Alessandro Rubini. Linux Device Drivers[M]. O Reilly & Assocoates, Inc. 1998.

[2] HANBIT Electronics Co., Ltd. Secure Digital Card Datasheet [DB/OL]. Rev. 1.1 http://www.hbe.co.kr/english/main/e_main.htm. May, 2003.

[3] Intel. Intel (r) PXA255 Processor Developer's Manual[DB/OL]. http://www.intel.com. March, 2003.

[4] Maurer, S.S. A survey of embedded systems programming languages [J]. IEEE Potentials, Volume: 21, Issue: 2, April-May 2002: 30~34

[5] Rui Wang, Shiyuan Yang. The design of a rapid prototype platform for ARM based embedded system [J]. Consumer Electronics IEEE Transactions on, Volume: 50, Issue: 2, 2004.5: 746~751

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
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化

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
23. Vxworks 下的冗余 CAN 通讯模块设计

24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析

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
30. 基于嵌入式 Linux 的网络设备驱动设计与实现

31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发

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
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统

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
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计

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
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析

24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计

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
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现