LINUX+ARM下的USB驱动开发*

USB Driver Development Under LINUX and ARM

张　鹏，孙世磊，刘瑞北，何明聪

ZHANG Peng, SUN Shi-lei, LIU Rui-bei, HE Ming-cong

(武汉大学计算机学院，湖北 武汉 430072)

(School of Computer Science and Technology, Wuhan University, Wuhan 430072, China)

摘　要：以Linux为OS的嵌入式系统已大量普及，在上面开发USB驱动的需求也越来越大。本文在开发成功的具体案例上，讨论了Linux下USB驱动的编写技术。

Abstract: Embedded systems based on Linux are more popular now, and they need more USB drivers. This paper discusses how to write a Linux USB driver based on a successful development case.

关键词：Linux;嵌入式系统；USB设备；USB海量存储设备类

Key words: Linux; embedded system; USB device; USB mass storage

中图分类号：TP316　　**文献标识码**：A

1　引言

USB是应用广泛的一种串行总线技术，通常HOST端控制所有的传输，而外设（如数码相机等）作为DEVICE端实现不同的功能。在笔者一项基于S3C2410芯片（ARM920T内核）的开发中，要求同时提供HOST和DEVICE两种接口。操作系统是LINUX 2.4.18，而S3C2410本身带有支持USB1.1的HOST和DEVICE硬件接口。本文将首先说明如何实现S3C2410的USB HOST总线和USB DEVICE总线驱动，然后说明如何在USB DEVICE总线驱动上实现USB海量存储设备类的功能驱动。

2　USB HOST总线驱动

USB协议没有明确定义软件、硬件的分界面，其中HOST在发展过程中形成了OHCI(Open Host Controller Interface,简称OHCI)和UHCI(Universal Host Controller Interface,简称UHCI)两种标准。OHCI由Compaq等提出，大多数便携式计算机使用OHCI标准芯片；UHCI由Intel公司开发，大多数台式PC使用这种接口标准。

S3C2410的USB HOST支持OHCI标准。OHCI规定了一系列连续的寄存器，作为硬件与软件交互的界面。虽然USB OHCI的标准在软件上实现非常复杂，但LINUX内核已经包括了OHCI的机制实现部分。针对S3C2410芯片，唯一需要对内核修改的就是指定S3C2410的OHCI寄存器基地址。

3　USB DEVICE总线驱动

在标准的LINUX内核里面不支持USB DEVICE总线驱动，针对这款芯片，必须开发全新的驱动模块。

S3C2410的USB DEVICE接口支持一个16字节的双向控制端点（0号端点）和四个64字节的双向批量（中断）传输端点。为了实现控制功能，它提供了大量的寄存器。对于四个批量（中断）传输端点，提供的寄存器都是一样的；而对于0号端点，除了不支持DMA传输以外，与其他端点的寄存器也基本一样。这些寄存器主要分为下面几类：

(1) 控制状态寄存器。用于设定端点的各种属性，包括工作模式（批量还是中断），是否采用DMA传输以及其他在传输过程中的细节设定。读取这类寄存器就可以得到当前端点传输的各种状态。

(2) 中断设定（使能、屏蔽、等待），也可获知究竟是哪个端点发生了中断。

(3) FIFO数据寄存器。需要传输的数据将连续写入该寄存器，实际上被保存到了一个FIFO（先进先出）的缓冲区中，将

被自动发送。同时,读取该寄存器将获得接受到的数据。

(4) DMA传输寄存器。主要设定是否工作在DMA模式,以及相应的细节。但是,具体DMA需要在系统的DMA控制器驱动里实现。

同时,还提供了一条中断请求线,用于在接受到数据或者发送完数据(以及其他定义需要中断的情况)时向系统发出中断请求。

在能够使用各个端点以前,必须通过0号端点进行配置,这也是USB DEVICE接口与PC机的USB HOST接口连接后的第一个动作。通常,我们使用有限状态自动机来完成这项配置工作(具体配置过程可以参见USB1.1规范)。

配置好通道以后,一个典型的数据发送过程如下:

S3C2410的1号通道FIFO缓冲区寄存器为EP1_FIFO,长度设为最大值64字节,等待发送的数据开始地址为*buf*,长度为*len*。假设*len*>64(*len*≤64可当成以下过程的特例)。

(1) 将*buf*开始的64字节依次赋值给EP1_FIFO。

(2) 设定控制寄存器开始发送。

(3) 本次硬件发送完毕以后会发出中断,在中断处理函数里面,检查(*len* -=64)是否大于0。如果大于0(未发送完),则发送下面的不超过64字节的数据,并返回(这会引起下一次中断);如果已经发送完毕,那么调用用户给予的一个回调函数来完成用户指定的工作(如开始接受回应的数据)。

通道已经建立,DEVICE总线驱动也就完成了。

4 USB海量存储设备类

在确定USB规范的时候,确立了设备类的概念。所谓设备类就是共用一个主机端驱动的一类设备。在该应用中,需要通过USB DEVICE接口直接读写硬盘,那么就要遵循MASS STORAGE类(海量存储设备类)的协议,主要是下列规范:

(1) USB MASS STORAGE Class Bulk-Only Transport。仅仅使用Bulk端点传送数据/命令/状态。

(2) USB MASS STORAGE Class UFI Command Specification。需要DEVICE完成的UFI命令。

Bulk-Only的传输规范通过三个步骤来完成:

(1) 主机向客户端发送一个命令块包CBW(Command Block Wrapper,简称CWB)。

(2) 解析CBW里包装的UFI命令,主机和客户端之间完成一定的数据传输。关键的UFI命令有INQUIRY、TEST UNIT READY、MODE SENSE(6)、READ CAPACITY、READ(10)、WRITE(10)。

(3) 客户向主机发送一个命令状态包CSW(Command Status Wrapper,简称CSW),表示操作的结果(成功或者失败)。

5 在S3C2410上实现USB海量存储设备类DEVICE端驱动

具体实现遵循了上述的流程,如图1所示。

为了提高效率,采取了内核线程加等待队列的方法来完成异步处理。

将接受CBW、处理UFI命令和发送CSW这样一个循环作为一个内核线程,在外设插入主机完成配置以后开始自动运行,在拔出时自动停止。

UFI命令的难点在于READ(10)与WRITE(10)的处理,其关键参数就是读写块的开始地址和长度,涉及到与硬盘交互的问题。有两种方法:

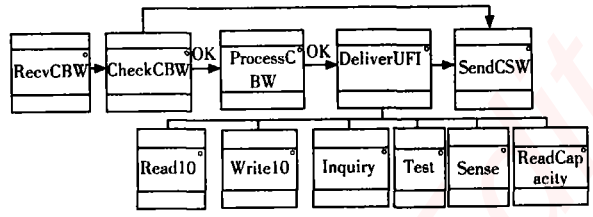

图1 具体实现流程

(1) 自己写一个简化的硬盘驱动程序,完成读写的操作。其好处是可以直接向硬盘发出指令,实际上系统起了一个USB-IDE桥的作用,效率很高。不好的地方是实现起来难度较大,而且会导致系统数据不一致(缓冲区、中断处理等)的情况。

(2) 利用LINUX的block设备接口函数来完成对硬盘的读写。好处是难度较小,而且稳定性会更高;缺点是效率会降低,比如写硬盘,必须先将硬盘数据读入内存,修改后再写入硬盘。

综合考虑各种情况,采用了LINUX的block设备接口来完成对数据的读写。LINUX的block接口主要提供了两个函数:

```
struct buffer_head * getblk(kdev_t dev, int block, int size);
```

来获取硬盘上一个块在内存中的缓冲区。如果这块已经被缓存,那么直接返回缓存的buffer_head;如果还没有缓存,那么创建一个新的缓冲区,返回其buffer_head。

```
void ll_rw_block(int rw, int nr, struct buffer_head * bhs[]);
```

可将一组"脏"(已经被修改的,dirty)的缓冲区块回写到硬盘中;或是将一组"没有更新"(还没有从硬盘读入,没有uptodate)的缓冲区块从硬盘读入。

对于READ10,其处理过程如下:

(1) 从UFI命令中取得要读的起始块和数量;

(2) 对于每一个块,采用getblk函数获取一个缓冲区;

(3) 将其中"没有更新"的缓冲区组成一个缓冲区数组,调用ll_rw_block函数从硬盘上读取数据到这些缓冲区中;

(4) 将内核线程依次等待在每一个缓冲区块上(使用wait_on_buffer函数),一旦线程被唤醒,意味着这块已经"更新",于是申请USB DEVICE总线驱动将这块发送出去,直到所有的缓冲区块都完成了处理;

(5) 内核线程等待在USB DEVICE总线驱动上,直到所有的数据块都被发送完毕。

现在处理完毕,可以发送CSW。从以上可以看到,对于硬盘的读写和对于USB DEVICE总线的读写是并行的,这样有效地提高了效率。

对于WRITE10,实际上是先读后写,其处理过程如下:

(1) 从UFI命令中取得要写的起始块和数量。

(2) 开辟一个临时缓冲区,向USB DEVICE总线申请接受所有需要写入的数据(按照经验,每次通常不会超过128块,也就是64K)。

(3) 对于每一个块,采用getblk函数获取一个缓冲区。

(4) 将其中"没有更新"的缓冲区组成一个缓冲区数组,调用ll_rw_block函数从硬盘上读取数据到这些缓冲区中。

(5) 将内核线程依次等待在每一个缓冲区块上(使用wait_on_buffer函数),直到所有的缓冲区都已经读入;然后等待在USB DEVICE总线上,直到所有从USB DEVICE总线上来的数据都写入了临时缓冲区。

(6) 将临时缓冲区中的数据依次写入上面的缓冲区块中,并将这些缓冲区设置为dirty,调用brelse函数释放这些缓冲区(LINUX块设备驱动将在后台回写这些数据)。

(下转第133页)

库中的知识(含灰运算规则)推断出结论,原理如图 3所示。推理过程伪码描述如下:

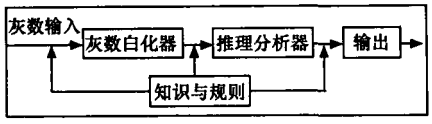

图 3 灰推理机原理图

```
procedure forwardref
  while  S非空且问题未求解 do
  begin
    调用   select_rule(S)
    调用   forwardref
  end
```

其中,S为可用规则集,$select_rule(S)$按设计的冲突消解策略选择规则。

4.2 实例数据

依据参考文献[2],以可靠性 Re、可维性 Ma、正确性 Co、可使用性 Us、效率 Ef和可移植性 Po等作为质量评价因子,四个被评价软件(用 A、B、C、D表示)的质量评价因子值如表 1所示。

表 1 软件质量评价因子白化值

因子	Re	Ma	Co	Us	Ef	Po
A($x1$)	0.93	0.79	0.91	0.76	0.78	0.83
B($x2$)	0.89	0.86	0.87	0.81	0.84	0.90
C($x3$)	0.86	0.84	0.93	0.78	0.75	0.87
D($x4$)	0.89	0.87	0.78	0.82	0.85	0.89

软件质量最优参考数据为 $x0=(1,1,1,1,1,1)$。评价者侧重于 (Re, Co, Us, Ef)等指标,取权重集 $\beta=(0.18, 0.1, 0.2, 0.2, 0.25, 0.07)$,将上述各值输入系统得 $\gamma=(0.715, 0.730, 0.698, 0.703)$。

4.3 结论分析

四个被评价对象以 (Re, Ma, Co, Us, Ef, Po)等作为质量评价要素因子、以 β为权重集进行评价时,软件质量优劣依次为 B→A→D→C。

5 结束语

本文提出了灰色推理的技术模型,介绍了软件质量智能灰色评价系统(SQIGES)的基本结构和实现方法。研究结果表明,灰色推理模型是对模糊集合和 Cantor集合的扩展。SQIGES系统理论上先进,技术上可行。为了完善灰色推理模型和提高软件质量评价准确性,运用可信度的不确定推理、模糊推理与灰色聚类分析的相互校验等问题尚待进一步探究。

参考文献:

[1] 廉师友. 人工智能技术导论[M]. 西安:西安电子科技大学出版社, 2002.
[2] 尹朝庆,尹 皓. 人工智能与专家系统[M]. 北京:中国水利水电出版社, 2002.
[3] 刘思峰. 灰色系统理论及其应用[M]. 北京::科学出版社, 1999.
[4] 罗佑新,张龙庭,李敏. 灰色系统理论及其在机械工程中的应用[M]. 长沙:国防科技大学出版社, 2001.
[5] 王清印,王峰松,左其亭,等. 灰色数学基础[M]. 武汉:华中理工大学出版社, 1996.
[6] 齐治昌,谭庆平,宁洪. 软件工程[M]. 北京:高等教育出版社, 2001.
[7] 冯建湘,唐嵘,高利. 基于模糊逻辑的软件质量评价方法[J]. 安徽理工大学学报, 2003, 23(4): 40-42.
[8] 冯建湘,唐嵘,王双维. 软件质量要素的灰色关联分析及其应用[J]. 计算机工程, 2004, 30(18): 91-92.
[9] 郑人杰,殷人昆,陶永雷. 实用软件工程[M]. 北京:清华大学出版社, 2002.
[10] 黄梯云. 智能决策支持系统[M]. 北京:电子工业出版社, 2001.

(上接第 107页)

现在可以发送 CSW了。上述过程可以通过测量硬盘的读写时间和 USB DEVICE的读写时间做一定的修改,以提高效率。

其他 UFI的命令处理比较简单,按照 UFI规范处理就可以了。

6 结束语

上述驱动使整个系统在不加入新硬件的情况下模拟成一个移动硬盘,较好地完成了预定目标。经过测试,可以通过 USB接口在 PC机上流畅地播放位于本开发板硬盘内的多媒体文件。大文件拷贝测试速度为:读平均 660K/s,写平均 440K/s。相对于 S3C2410 USB接口 1.2M/s的速度和硬盘工作于 PIO大约 3M/s的速度,这个结果是比较令人满意的。

参考文献:

[1] 毛德操,胡希明. Linux内核源代码情景分析[M]. 杭州:浙江大学出版社, 2001.

(上接第 125页)

的 J2EE三层体系框架,结合 MVC设计模式,并遵循 WfMC的相关标准,使整个工作流管理系统具有良好的结构、高度的可配置性和构件的可重用性,可将工作流平台作为中间件用于信息系统的开发中,节省了开发时间,提高了开发效率和效益。

参考文献:

[1] 何清法,李国杰,焦丽梅,等. 基于关系结构的轻量级工作流引擎[J]. 计算机研究与发展, 2001, 38(2): 129-137.
[2] Martin Fowler. Analysis Patterns-Reusable Object Models[M]. Boston: Addison-Wesley Publisher, 1997.
[3] 阎宏. Java与模式[M]. 北京:电子工业出版社, 2002.
[4] 罗海滨,范玉顺,吴澄. 工作流技术综述[J]. 软件学报, 2000, 11(7): 899-907.
[5] David Hollingsworth. The Workflow Reference Model[EB/OL]. http://www.wfmc.org/standards/docs/tc003v11.pdf, 2003-06.

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

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

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

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

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

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

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