嵌入式下 Linux系统设备驱动程序的开发

郑静菡，唱江华，刘树东，戴学丰

（齐齐哈尔大学 计算机控制与工程学院，黑龙江 齐齐哈尔 161006）

摘要：介绍了 Linux 操作系统的特点，分析了嵌入式下设备驱动程序的种类，探讨了基于 Linux 下设备驱动程序具体的开发过程，并提出了几点提高执行效率的方法。

关键词：嵌入式系统；Linux；驱动程序

中图分类号：TP316.81 文献标识码：A 文章编号：1007-984X(2009)01-0022-04

嵌入式 Linux 系统以其可应用于多种硬件平台、内核高效稳定、源码开放、软件丰富、网络通信和文件管理机制完善等优良特性而正被作为研究热点，越来越多的研究人员采用 Linux 平台来开发自己的产品。Linux 设备驱动程序在 Linux 内核源代码中占有很大比例，从 2.0、2.2、2.4 到 2.6 版本的内核，源代码的长度日益增加，其实主要是设备驱动程序在增加。

Linux 内核 2.6 有了更大突破：新的调度器，内核任务可以被抢占，改进了线程模型，融合了反向映射技术，改善了虚拟内存在一定程度负载下的性能，增加了对日志文件系统功能的支持，新的声音体系结构支持 USB 音频和 MIDI 设备，并支持全双工重放等功能，总线问题，电源 ACPI，用于调整 CPU 在不同的负载下工作于不同的时钟频率已降低功效，联网和 IPSec，重写了帧缓冲/控制台层，人机界面层还加入了对大多数接口设备的支持。在设备驱动程序的方面，比以前也有较大改动，表现在内核 API 中增加了不少新功能：sysfs 文件系统、内核模块从.o 变为.ko、驱动模块编译方式、模块实用计数、模块加载和卸载函数的定义等方面。

1 设备驱动程序的编写

Linux 有两种方式使用设备驱动程序：直接编译到内核中，在运行时侯加载。直接将硬件驱动程序写入内核的优点在于，用户可随时对它进行调用而无需安装，但是这样大大增加了内核占用的空间。将硬件驱动程序写成一种可以加载的内核模块，可谓软件开发提供许多便利。也可由系统管理员把已加载的的模块动态地卸载下来。

驱动程序是内核的一部分，是操作系统内核和机器硬件之间的接口，它由一组函数和一些私有数据组成，是连接应用程序与具体硬件的桥梁。它向下负责和硬件设备的交互，向上通过一个通用的接口挂到文件系统上，从而和系统的内核等联系起来。

1）设备驱动的特点和分类。Linux 的一个基本特点是它对硬件设备的管理抽象化，系统中的每一个设备都用一个特殊的文件来表示，所有的硬件设备都像普通的文件一样看待，使用与操作系统相同的标准系统来进行打开（open）、读写(read/write)和关闭（close）。在 Linux 操作系统下有 3 类主要的设备文件类型：块设备、字符设备、网络设备。字符设备是指存取时没有缓存的设备，可像文件一样访问字符设备，字符设备驱动程序负责实现这些行为。系统的控制台和并口就是字符设备的例子，它们可以很好地用“流”来描述。块设备是文件系统的宿主，如磁盘。Linux 允许像字符设备那样读取块设备——允许一次传输任意数目的字节。结果是，字符设备和块设备读取数方式一致。而网络设备不同于字符设备和块设备，它面向的上一层不是文件系统而是网络协议层，是通过 BSD 套接口访问数据。与设备相对应的是三类设备驱动程序,字符设备驱动程序、块设备驱动程序、网络设备驱动程序。

2）设备驱动程序的结构体。 设备驱动程序、块设备驱动程序与网络设备驱动程序的结构体是不同的。在 Linux 源代码 linux/include/linux/fs.h 中定义了字符设备和块设备驱动程序中必须使用的 file operations 结构，每个设备驱动都实现这个接口所定义的部分或全部函数。随着内核的不断升级，file_operations 结构也越来越大，不同的版本的内核会稍有不同。

file-operations 定义如下。

```
struct file_operations{
int （*lseek）(struct include*,struct file*,off_t,int);
int (*read) (struct include*,struct file*,char*,int);
int (*fsync) (struct inode*,struct file*,int);
int (*write) (struct inode*,struct file*, const char*,int);
int (*fasync) (struct inode*, struct file* ,int);
int (*readdir) (struct inode, struct file, void*, filldir_t);
int (*check-media-change) (kdev_tdev);
int (*select) (struct inode*, struct file*,int,select_table*);
int (*revalidate) (kdev_t dev);
int (*ioctl) (struct inode*, struct file*, unsigned int, unsigned long);
int (*mmap) (struct file*,struct vm_area_struct*);
int (*open) (struct inode*,struct file*);
int ( *release) (struct include*,struct file*);
}；
```

应用程序只有通过对设备文件的 open、release、read、write、ioctl 等才能访问字符设备和块设备。用户自己定义好 file_operations 结构后，编写出设备实际所需要的各操作函数，对于不需要的操作函数用 NULL 初始化，这些操作函数将被注册到内核，当应用程序对设备相应的设备文件进行文件操作时，内核会找到相应的操作函数，并进行调用。如果操作函数使用 NULL，操作函数就进行默认处理。

对于字符设备而言，llseek()，read()，write()，ioctl()，open()，release()这些函数是不可缺的；对于块设备，open()，release()，ioctl()，check_media_change()，revalidate()是不可缺少的。

网络设备结构体 net_device 定义在 include\linux\netdevice.h 里，如下所示。

```
struct net_device
    {
     Char name ;
     int (*init)(struct net-device dev);
     unsigned short flags;
     int (*open)(struct net_device*dev);
     unsigned long base_addr;
     int (*stop)(struct net_device*dev)unsigned int irq;
     int (*hard-start-xmit)(struct sk_buff*skb, unsigned char dev_addr;
     struct net_device* dev);
     unsigned char addr_len;
     int (*set_mac_address)(struct net_deviceunsignedlong trans_start; *dev,void* addr);
          ……
    }
```

定义好 net_device 结构体后，根据实际情况编写操作函数，其中 hard_start_xmit()函数是用来发送数据的，set_mac_address()是进行网络参数设置的。当 Linux 初始化时将调用初始化函数 int device_init()，该函数包括以下内容：注册所用设备。Linux 用设备号来标识字符设备和块设备。设备号分为主设备号和从设备号，最终形成设备接点。设备节点在访问字符设备和块设备的设备驱动程序时将使用。通常主设备号标识设备对应的驱动程序，大多数设备是“ 一个主设备号对应一个驱动程序 ”，如：虚拟控制台和串口终端由驱动程序管理。次设备号由内核使用，用于确定设备文件所指的设备。字符设备和块设备注册时必须先定义好设备号。

字符设备注册函数如下：

int register_chrdev(unsigned int major, const char*name, struct file_oprations* fops);

其中，major 是主设备号。

由于对网络设备驱动程序的访问不需要设备节点，它的注册函数如下：

int register_netdev(struct net_device *dev)

注册设备所用的中断。中断在现代计算机结构中有重要的地位，操作系统必须提供程序响应中断的能力。一般是把一个中断处理程序注册到系统中去。操作系统在硬件中断发生后调用驱动程序的处理程序。

注册中断所用的函数如下：

int request_irq(unsigned irq,void(*handler)(int,void*,struct pt_regs*),unsigned long flags,const char*device, void*dev_id);

其中，irq 是中断向量；handler 是中断处理函数；flags 是中断处理中的掩码；devices 是设备名；dev_id 是在中断共享使用的 id。

当 Linux 使用设备时，就要调用清除函数 void device-exit(), 它同初始化函数相对应的，主要是：注销设备,字符设备注销函数如下：

int unregister_chrdev(unsigned int maior,const char *name, struct file_oprations* fops)；

注销中断，注销中断所用的函数如下：

int free_irq(unsigned irq ,void(*handler)(int，void*，struct pt_regs*)，unsigned long flags，const char*device，void*dev_id);

释放资源，模块初始化和清除函数采用 module_init (device_init)，module_exit(device_exit)形式

3）编写服务子程序

服务于 I/O 请求的子程序，又称为驱动程序的上半部分。调用这部分是由于系统调用的结果。这部分程序在执行的时候，系统仍认为是和进行调用的进程属于同一个进程。只是用户态变成了核心态，具有进行此系统调用的用户程序的运行环境，因此可以在其中调用 sleep 等与进程运行环境有关的函数。

中断服务子程序，又称为驱动程序的下半部分。在 Linux 系统中，并不是直接从中断向量表中调用设备驱动程序的中断服务子程序，而是由 Linux 系统来接收硬件中断，再由系统调用中断服务子程序。中断可以产生在任何一个进程运行的时候，因此在中断服务程序被调用的时候，不能依赖于任何进程的状态，也就不能调用任何与进程运行环境相关的函数。因为设备驱动程序一般支持同一类型的若干设备，所以一般在系统调用中断服务程序的时候，都带有一个或多个参数，以唯一标识请求服务的设备。

2 提高嵌入式 Linux 驱动执行效率的几个方法

对于运行的在 ARM、PowerPC 等一般速率不高的嵌入式处理器上，但又要求实时处理的，提高运行效率的情况十分必要。首先，执行时间比较长的函数，应该考虑尽量避免动态重复调用执行。其次，尽量设用内核提供的函数。系统中提供的内核函数是经过精心编写的 ，不管是执行效率，还是稳定性都是无可挑剔。最后，为驱动与上层应用设计良好的接口。设计良好的接口将驱动和应用程序实现的功能合理划分开来，可以简化驱动的实现过程，提高驱动的执行效率。

3 结束语

设备驱动程序的开发是在 Linux 环境中最复杂的编程任务之一。驱动程序沟通着硬件和应用软件，而驱动工程师则沟通着硬件工程师和应用软件工程师。随着通信、电子行业的飞速发展，全世界每天都会有大量的芯片得生产，大量的新电路板被设计，因此，也会有大量设备驱动需要开发，设备驱动设计的好坏将直接影响整个系统的性能表现，这些设备驱动发挥着不可替代的作用。

参考文献

[1] 周立功，陈明记，陈渝. ARM 嵌入式 Linux 系统构建与驱动开发[M]. 北京：北京航空航天大学出版社，2006.

[2] AlessandroRubmi，JonathanCorbet．Linux 设备驱动程序[M]. 3 版，北京：中国电力出版社，2006.

[3] 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.

[4] 王学龙. 嵌入式 Linux 系统设计与应用[M]. 北京：清华大学出版社，2001.

The development of device driver on embedded Linux systems

ZHENG Jing-han，CHANG Jiang-hua，LIU Shu-dong，DAI Xue-feng
(College of Computer and Control Engineering，Qiqihar University，Heilongjiang Qiqihar 161006，China)

Abstract: This paper describes the feature of development of Linux operation system, introduces the type of device driver on the embedded Linux, discuses the development process of the device driver on Linux-based, proposes the methods of improving the efficiency.

Key words: embedded system；Linux；device driver

偏心工件的精确配重计算

我厂生产的 SMVTM1250×55/400L-NC 数控单柱移动立式车铣床中，刀夹体 01203 采用直径为 ϕ150h6 的轴与 400×400 方滑枕定位连接。轴 ϕ150h6 与方 400×400（mm）偏心 32.5 mm，由离心力引起的工件不平衡部分的力矩使外圆磨削圆柱度达不到要求。为解决这个问题，磨削时应采取平衡措施，以减少由离心力产生的振动和主轴轴承的磨损,保证圆柱度要求。

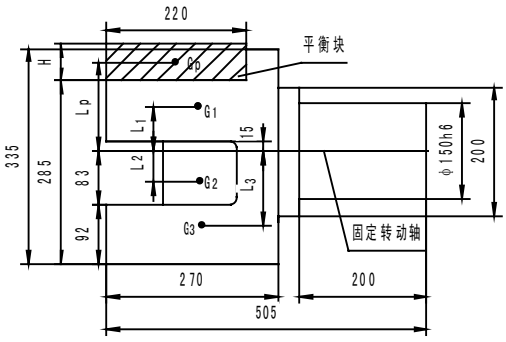
1　加工原理

在实际加工中，用常规重量进行配重，很不准确。我们在工作中总结出利用力矩平衡方法计算刀夹配重。磨削时用顶尖顶住刀夹两端中心孔，因此轴向力不必计算，利用各区离心力引起的力矩，计算径向力距平衡就可以了，用此方法计算结果非常准确。使外圆磨削的圆柱度达到技术要求。有固定转动轴物体的平衡条件是合力矩为零，即 ΣM=0，如果一个物体所受到的力的合力矩的代数和是 0，那么就说这个物体处于力矩平衡状态。

2　计算过程

如右图，刀夹体绕固定转动轴线匀速转动，以轴线划分，则存在不平衡力矩。相对于轴线对称的部分离心力矩相互抵消，将相对于轴线不对称的部分进行区域划分，分区域进行力矩计算。去掉平衡区域后，可将工件划分为 G_1、G_2、G_3 三个不平衡区域。预设增加的平衡块重量为 G_p，厚度为 H，长和宽与刀夹尺寸相适应。则转动时轴线上部所产生的离心力力矩为：$M_{上}=G_pL_p+G_1L_1$，转动时轴线下部所产生的离心力力矩为

$$M_{下}=G_2L_2+G_3L_3$$



刀夹体

G_p：平衡块的重量（kg）；G_1、G_2、G_3：工件不平衡部分的合成重量（kg）；L_p：平衡块重心至回转中心的距离（m）；L_1，L_2，L_3：工件不平衡部分的合成质量重心至回转中心的距离（m）。

考虑在平衡位置情况下，满足平衡关系式：$M_{上}+M_{下}=0$ 即：$G_pL_p+G_1L_1+(G_2L_2+G_3L_3)=0$

$m_pgL_p+m_1gL_1+(m_2gL_2+m_3gL_3)=0$ 化简后为

$$V_pL_p+V_1L_1+(V_2L_2+V_3L_3)=0$$ ，V_p，V_1，V_2，V_3：各区域的体积（m^3）

则代入数字计算如下

$$400\times220\times H(H/2+110)+28150\times400\times69=7798.54\times220\times49+29281.46\times400\times117$$

经过计算得出：H=55.82 mm。即在偏心上侧安装一块长 400 mm、宽 220 mm、厚 55.82 mm 的钢板，即能使刀夹在磨削时处于力矩平衡状态，使外圆磨削的圆柱度达到技术要求。

3　结论

通过以上例证可知，用此方法可以对有固定转动轴的偏心工件的配重进行精确的计算，可适用于磨削和车削，通过生产实践证明，用此方法计算结果非常准确，使零件的各项精度达到技术要求。

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

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
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

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