基于 UEFI的 Application和 Driver的分析与开发

吴松青　王典洪
(中国地质大学机械与电子工程学院　湖北 武汉 430074)

摘　要　UEFI(Unified Extensible Firmware Interface,统一的可扩展固件接口)是由 Intel提出的下一代 BIOS构架。基于 UEFI2.0规范,借助 UEFI开发环境: Intel的 EDK(EFI Develop Kit),对 UEFI Application和 UEFI Driver作了一定的分析,并通过两个例子分别予以实现。

关键词　统一的可扩展固件接口　应用程序　驱动程序　基本输入输出系统

ANALYSIS AND DEVELOPMENT OF UEFI APPLICATION AND UEFI DRIVER

Wu Songqing　Wang Dianhong
(*Faculty of Mechanical & Electronic Engineering, China University of Geosciences, Wuhan Hubei 430074, China*)

Abstract　UEFI (Unified Extensible Firmware Interface) sponsored by Intel is the next generation of BIOS framework. This paper is based on UEFI 2.0 specification and Intel EDK environment. It analyses and realizes UEFI Application and UEFI Driver by giving two examples respectively.

Keywords　UEFI　Application　Driver　BIOS

1 引　言

UEFI是操作系统与硬件平台固件之间的新一代接口。它除了完成传统 BIOS的工作之外,还建立起高级语言执行环境,可以调用设备驱动,可以远程配置及引导,不用操作系统就可以进行磁盘管理及启动管理,以及具有脱离操作系统的管理工具等。UEFI的工作过程大致可以归纳为:启动系统,然后初始化标准固件,接着加载 EFI驱动程序库及执行相关应用程序,最后在系统启动菜单中选取所要进入的系统。UEFI为用户提供了一个交互环境:UEFI shell,用户可以通过 UEFI shell来导入自己编写的特定的 Application和 Driver。UEFI Application(下文中简称 App)可以是硬件检测或除错软件,引导管理设置软件,也可以是操作系统引导软件等。UEFI Driver提供一系列与系统设备通信的接口,它可以从任何支持 UEFI环境的设备中导入[1]。

2 UEFI Application

UEFI App和 UEFI库函数提供基本控制台 IO,基本磁盘 IO,内存管理以及字符串操作功能[1]。本文是通过 Intel的 EDK来开发基本的 UEFI App的。UEFI App以可执行程序 *.efi的形式存在,执行完后返回控制权,不会驻留内存,可以方便移植到不同的平台。目前有好几种编写 UEFI Apps的方法,分别是基于 UEFI的,基于 UEFI Library的,基于 C Library以及基于 C Standard library的[4]。文中对前两种方法做了分析。

2.1 基于 UEFI/UEFI Library写 Application

UEFI App可以添加到 UEFI源代码结构中去。建议把所有独立的 UEFI App都放在 EDK的 \efi\Apps目录下,因为它提供了一个方便的编译环境,当然也可以不这样做。下文中实现了一个名为 Hello的 App例子。当把新的 App加入到编译环境中去的时候,需要建立一个存放 App源代码的子目录,和一个与 App源代码相关的 make.inf。下面以 Hello App为例,对这个例子来说,App的文件放在 hello目录中。make.inf中包含了源文件列表,以及可执行 App映像的名字。Hello App代码比较简单,它不依赖于任何 UEFI库函数,所以 UEFI函数库不会链接到可执行程序中。该 APP使用被传递到代码运行入口点的系统表格来读写 EFI控制台设备[2]。控制台输出设备通过使用 IMPLE_TEXT_OUTPUT_INTERFACE协议的 Outputstring()函数来显示相关信息。然后,IMPLE_INPUT_INTERFACE协议中带 WaitForKey事件的 WaitForEvent()服务等待用户从控制台输入设备上按键[3]。一旦有按键,App就会退出。图 1就是 Hello App的大致执行流程,这个程序的工作就是打印一个字符串到终端设备上。类似于我们常见的 Hello World。

此外,还可以基于 UEFI Library写 UEFI App如果 UEFI App想使用 UEFI库函数的话,需要包含头文件 efilib.h,并且加入对 InitializeLib()的调用。这个 App利用 UEFI库把文本打印到控制台输出设备。

其中使用了全部变量如 ST而不是 SystemTable来做标准的

UEFI App调用。由于代码与基于 UEFI写 App的代码很相似，所以这里略去。

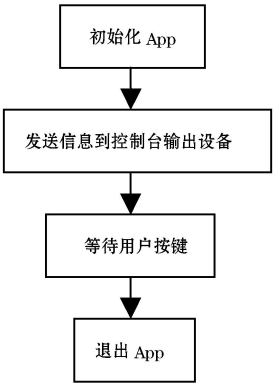

图 1 Hello App的执行流程

2.2 UEFI Application的编译和运行

UEFI App是带有修改过的头标识的 PE/COFF文件。头标识用来区分 UEFI映像和一般的 PE/COFF映像。Intel提供了相应的把 PE/COFF文件转换成 EFI映像文件的工具。在一个新的 UEFI App能够编译之前，每个编译终端（build tip）的 makefile文件还需要修改。这些文件包括 \efi\build\ia32-emb\makefile, efi\build\sal64\makefile和 \efi\build\bios32\makefile[2]。其中每个文件中都有一个标识为 everything to build的部分，并加入了对 makemaker的具体描述。这些步骤完成之后，当编译终端中运行 nmake时，新的 App就可以编译了。我们可以在 UEFI shell命令行敲入 App的名字来运行相关 UEFI App。

3 UEFI Driver

UEFI Driver可以在计算机预启动环境中访问启动设备，这些 Driver可以管理或控制平台上的硬件总线和设备，也可以提供基于平台的特定的软件服务。但是它不能取代操作系统中的 Driver。使用 UEFI Driver之前，必须先把它导入 UEFI Handle中。UEFI Driver容易更新，也容易增加对新硬件的支持。

3.1 UEFI协议及 Driver模型

UEFI协议是 UEFI对于硬件平台中各个设备的抽象，通过特定协议提供的接口可以对相应设备进行操作。UEFI协议接口包含很多函数指针，用户可以通过将这些指针指向不同的功能函数，使协议驱动不同的硬件。如图 2，首先通过 gBS->LoadImage()启动服务将设备驱动导入内存，然后通过 gBS->StartImage()调用该设备驱动。gBS->LoadImage()服务自动产生一个映像处理并且把 EFI_LOADED_IMAGE_PROTOCOL安装到该映像处理中去。EFI_LOADED_IMAGE_PROTOCOL描述了设备驱动的导入地址以及设备驱动在系统内存中的存放位置。EFI_LOADED_IMAGE_PROTOCOL的 Unload()服务由 gBS->LoadImage()初始化为 NULL，表示默认的情况下驱动是不可导入的。gBS->StartImage()服务把控制权传递到驱动映像的 PE/COFF头中所描述的驱动入口点。驱动入口点负责把 EFI_DRIVER_BINDING_PROTOCOL安装到驱动映像处理上[5]。图 2中显示了在一个设备驱动被导入之前，将要导入驱动之前，以及驱动入口点被执行之后系统的状态。

3.2 USB设备驱动开发

UEFI Driver的类型有很多，下文重点讲述了大容量 USB设备驱动。设备驱动符合 EFI驱动模型，它通过把一个或者多个 Driver Binding Protocol协议实例安装到 handle数据库，来产生一个或多个驱动 handle或驱动映像 handle。当 Driver binding协议的 start()被调用时，这种驱动不会像总线驱动那样产生子 handle。它只是会向已经存在的控制 handle中添加额外的 I/O协议。根据图 2所示的 UEFI Driver模型，开发 USB设备驱动程序，主要需实现的相关协议有 Driver Binding协议。Driver Binding协议中主要需实现 Supported()，Start()以及 Stop()。另外需实现的相关协议还有 Component Name协议、Driver Configuration协议和 Driver Diagnostics协议。不过这三种协议是可选的，不影响正常的设备功能。

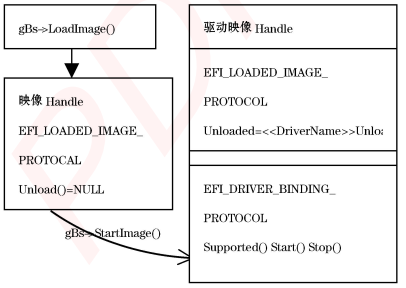

图 2 UEFI Driver模型

所有遵循 EFI Driver模型的 Driver程序都必须实现 Driver Binding协议。该协议提供了测试、开始和停止 Driver程序的服务，是 EFI能够管理硬件的前提。要实现该协议，必须实现三个接口：(1) supported()：它用于测试 Driver程序是否支持相应的硬件。Supported()服务用来检查相关控制器 handle是否是 USB设备的 handle。常见的做法是：检查 handle是否安装了 EFI_USB_IO_PROTOCOL。如果没有，该 handle就不是当前 USB总线上的 USB设备。获得从 USB_IO_DEVICE返回的 USB接口描述符。检查该设备的 InterfaceClass, InterfaceSubClass,和 InterfaceProtocol值是否与该 Driver能够处理的值一致。如果上述两步检查通过了的话，就表明 USB设备驱动能处理控制器 handle代表的设备。返回 EFI_SUCCESS。否则，返回 EFI_UNSUPPORTED。(2) start()：它用来启动 Driver程序。经过此步骤，硬件已经挂上系统，可以开始工作。该服务将打开 USB I/O协议 BY_DRIVER并安装 USB设备的 I/O抽象协议到安装了 EFI_USB_IO_PROTOCOL的 handle上去。这部分是具体怎么执行 USB设备驱动。

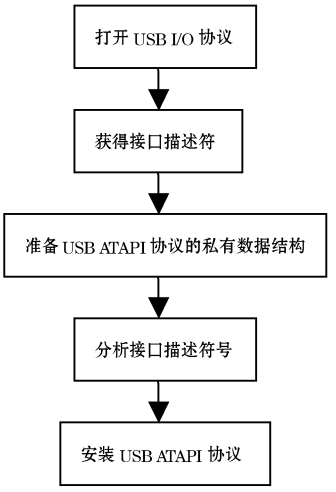

图 3 USB设备驱动 start()服务流程

这里使用 USB CBI大容量设备为例。图 3就讲述了怎样去执行 start()驱动 binding协议服务以及 USB ATAPI协议服务。(3) stop():它实际上就是 start()的反过程,用来销毁 start()中使用、创建的资源,把已经打开的协议关掉即可。关闭协议的顺序应该和 start()中打开协议的顺序相反[5]。

这里省略了 USB ATAPI协议的具体服务流程,数据传输方式以及 Driver的编译与运行方法。

4 结束语

如今,UEFI已经在某些计算机系统上得到了实现。对于计算机相关的 IC设计厂家以及 BIOS供应商来说,基于 UEFI的 App和 Driver的开发和测试工作也显得重要和紧迫起来。

参 考 文 献

[1] Intel Unified Extensible Firmware Interface Specification, Version 2.0, http://www.uefi.org/agreement.php, 2006-01-31/2006-07-18[S].

[2] Intel EFI Developer Kit (EDK) Getting Started Guide, Version 0.41, http://developer.intel.com/technology/efi/, 2005-01-31/2006-07-18[Z].

[3] Vincent Girard-Reydet, EDK Reference Manual, Version 0.3, http://developer.intel.com/technology/efi/, 2005-07-05/2006-07-18[Z].

[4] Intel, EFI Developer's Guide. Version 1.10, http://developer.intel.com/technology/efi/, 2004-01-05/2006-07-18[Z].

[5] Intel, EFI Driver Writer's Guide. Version 1.10, http://developer.intel.com/technology/efi/, 2004-07-20/2006-07-18[Z].

(上接第 13页)

果为 tag,则根据具体的 tag生成相应的对象节点,作为孩子添加到当前栈顶对象节点中,这里是 Page类对象,同时把当前 tag对应对象压入栈中作为栈顶对象,直到遇到相对应的 tag结束标示该对象被弹出。这样递归下去直到文档结束,此时将生成相应的文档对象树。对表 1的文档进行分析。生成的文档树见图 4。

表 1 示范代码

```
<html><table><tr>
<td>hello</td>
<td>world</td>
</tr></table></html>
```

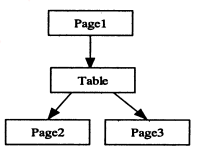

图 4 表 1文档对应的文档树

在生成文档树的同时,各个节点对象会把接口注册到相应的消息上。这些消息包括布局消息,绘图消息等。然后对接口用相应的布局和绘图函数进行赋值。这样可以通过定制这些函数实现定制浏览器行为。另外某些节点对象会根据上下文环境,注册自己和回调函数到特定的消息上。

当文档树构造完毕,数据处理模块会发出布局消息给布局模块。对于每个节点对象接到消息后,将首先根据注册的接口信息调用相应的布局函数。如果为非叶子节点,将继续发消息给自己的孩子对象。如果对象实例未注册该消息或为叶子节点,将直接返回。对于图 4布局模块将首先发消息给 Page1对象,Page1处理消息后,会判断是否有孩子,这里有一个孩子节点 Table,因此发消息给 Table,Table同理处理消息,会发消息给 Page2,Page3。而 Page2和 Page3由于是叶子节点,处理完消息后直接返回。这样构造的文档树完成一次遍历,对所有的文档进行了布局。

布局完成后,会发出显示消息给显示模块。显示模块对文档对象树处理与布局处理模块对文档对象树的处理一致,只是响应绘图消息注册的是,已注册的通过绘图接口映射过来的绘图函数。

在显示完成后,用户会点击超链接或按钮以及图片等,消息系统会首先完成由鼠标消息到文档对象可识别消息的映射,然后将消息激活。对于这些消息,由于在构造文档对象树的时候,直接注册的是对象实例和回调函数,因此可以直接调用处理函数对文档对象节点进行处理。

3 ElaScope

Elastos是 32位嵌入式操作系统,是完全面向构件技术的操作系统。操作系统提供的功能模块全部基于 CAR构件技术,因此是可拆卸的构件,应用系统可以按照需要剪裁组装,或在运行时动态加载必要的构件。Elastos体积小,速度快,适合网络时代的绝大部分嵌入式信息设备。

Atlas[6]是 ElastOS上的图形用户界面支持系统,提供和 windowsCE兼容。

我们基于该框架结构在 Elastos\Atlas上实现了 ElaScope浏览器。支持 HTML, XHTML, CSS, JPEG, GIF, HTTP1.1等。ElaScope完全用 C语言实现,编译后二进制代码共 304k,目前已经在 X86和 ARM上成功运行,并且可以访问主要的门户网站,布局和显示效果达到了桌面浏览器的水平。并且采用了多线程技术,可以多线程取数据,而且。ElaScope满足了目前的需求,并且由于基于该框架结构,可以很好地进行构件化和功能的扩展,比如添加对 JavaScript的支持。这也是我们下一步工作。

4 结 论

该框架在浏览器行为的定制、减少平台依赖性、良好的模块化和可扩展性方面有较大的优势。基于该框架实现的 ElaScope具有体积小、速度快和可灵活定制的特点。该框架不仅为嵌入式浏览器开发提供了一定的理论基础,而对嵌入式应用软件的开发具有一定参考价值。

参 考 文 献

[1] Document Object Model (DOM) Level1 Specification. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/[EB/OL]. W3C, 1998.

[2] Yoshinori Saida, Hiroshi Chishima, Satoshi Hieda, Naoki Sato, Yukikazu Nakamoto. An Extensible Browser Architecture for Mobile Terminals [C]. Proceedings of the 24th International Conference on Distributed Computing Systems Workshops 2004 IEEE.

[3] Embedding Gecko's website[EB/OL]. http://www.mozilla.org/projects/embedding/.

[4] Deepak Mulchandani Java for Embedded Systems[C]. IEEE Internet Computing, pp. 30～39, May 1998.

[5] D. Raggett HTML 4.01 Specification. W3C[M]. Dec. 1999.

[6] Elastos's WebSite[EB/OL]. http://www.elastos.com.cn.

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

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

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

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

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

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

Programming:

1. 计算机软件基础数据结构 - 算法