基于 EFI系统的多文件系统解决方案

黄海彬，金　晶

(上海交通大学微电子学院，上海 200240)

摘　要：EFI (Extensible Firmware Interface) 是由 Intel公司研发并推广的旨在取代传统 BIOS的固件解决方案。EFI架构拥有传统 BIOS无法比拟的优点，譬如：C语言代码编写，设备驱动模型，文件系统等高级特性。在介绍 EFI固件文件系统的基础上，提出了一种支持 EFI多文件系统的解决方案。

关键词：EFI；BIOS；固件文件系统

A method to support multi-FFS for EFI system

HUANG Hai-bin, JIN Jing

(Microelectronics School, Shanghai Jiaotong University, Shanghai 200240, China)

Abstract: EFI (Extensible Firmware Interface) framework is initiated and promoted by Intel in PC industry to replace legacy BIOS. EFI specification defines a new model for the interface between operating systems and computing platform firmware and provides a standard environment for booting an operating system and running pre-boot applications. Beyond BIOS, EFI has some advanced features: implementation in C, driver model, and firmware file system and etc. Based on EFI FFS (Firmware File System), this paper introduces a new method to support multi-FFS.

Key words: EFI; BIOS; FFS

0　引言

众所周知，BIOS (Basic Input/Output System，基本输入/输出系统)已经伴随着计算机走过了 20多年的漫漫长路。自从 IBM PC诞生以来，虽然计算机软硬件得到了日新月异的发展，但 BIOS却一直停滞不前。传统 BIOS是由汇编代码编写，并运行在 IA架构 16位模式下，只使用 1M内存，这些缺点使得传统 BIOS开发周期长，维护成本高，无法全面地使用某些硬件最新功能。而且对于 IA64安腾 (Itanium)平台，ARM的 Xscale平台，传统 BIOS都不支持。为此，英特尔公司开发了 EFI架构 (EFI Framework)，并将其内核代码 EDK (EFI Develop Kit)开源在网站 www. tianocore. org。同时英特尔联合微软，HP，DELL等业界各大公司一起制定了 UEFI(Unified EFI)规范来推广 EFI技术。

相比传统 BIOS，EFI架构具有很多优势。譬如：C语言的编程，模块化的设计，文件系统的支持等这些只有操作系统才有的特性。实质上，EFI架构本身就是个嵌入式操作系统。为了支持模块化设计，EFI定义了驱动模型 EDM (EFI Driver Model)。这样每个硬件厂商可以按照 EDM标准开发出自己设备的驱动，从而减少了厂商之间彼此的依赖性。作为一种新型的 BIOS，EFI的映像文件 (Image)也是需要固化在主板上的 Flash芯片内的。为了在有限的 Flash空间内能更有效地存储模块化驱动文件，EFI架构提出了固件文件系统 FFS(Firmware File System)。

1　EFI编译系统

EFI支持模块化编译，每个模块都由源文件 *. c, *. h文件和 meta-data文件 *. inf构成。以开

收稿日期：2009 - 12 - 29

作者简介：黄海彬 (1980 -)，男，2003年获电子科技大学电子工程学士学位。上海交通大学微电子学院 2008级在职工程硕士，研究方向为微处理器系统设计。目前任职于英特尔亚太研发有限公司 EFI项目开发小组。

Bus. inf为例:

```
[defines]
    BASE_NAME           = ScsiBus
    FILE_GUIDP          = 0167CCC4-D0F7-4f21
                          -A3EF-9E64B7CDCE8B
    COMPONENT_TYPE      =BS_DRIVER
    [sources.common]
    ScsiBus.h
    ScsiBus.c
    ComponentName.c
    [libraries.common]
    EdkProtocolLib
    EfiProtocolLib
    EfiDriverLib
    ScsiLib.Lib
    [nmake.common]
    IMAGE_ENTRY_POINT = ScsiBusControllerDriv-
erEntryPoint
```

1.1 生成 ScsiBus.efi文件

在编译该模块时,Build tool会分析 ScsiBus.inf里的每个字段。先编译[Source.common]字段指示的源文件,并与[libraries.common]字段指示的库文件进行静态链接,从而生成可执行的二进制文件 ScsiBus.efi。*.efi文件采用 Windows的 PE/COFF格式,PE/COFF文件需要指示该代码段执行的入口地址,这里 IMAGE_ENTRY_POINT描述的 ScsiBusControllerDriverEntryPoint函数就是该模块的执行入口。

1.2 生成 0167CCC4-D0F7-4f21-A3EF-9E64B7CDCE8B-ScsiBus.ffs文件

当生成 ScsiBus.efi文件后,Build tool会根据[defines]中 COMPONENT_TYPE字段来判断该模块是 DRIVER还是 APPLICATION。BS_DRIVER是代表 DXE DRIVER。同时 Build tool会将 ScsiBus.efi封装在一个 PE32 Section段内,PE32 Section是专门用来封装 PE/COFF格式 *.efi文件的。最后将该 PE32 Section封装在一个 FF(Firmware File)内,该 FF名称是[defines]中 FILE_GUID,FF类型是 DXE DRIVER。最后产生 0167CCC4-D0F7-4f21-A3EF-9E64B7CDCE8B-ScsiBus.ff文件。

1.3 生成平台的 FD文件

当为某个平台生成 Image时,需要定义该平台 Nt32\Build\Nt32.dsc为例:

```
DEFINE FV = FvRecovery
Foundation\Core\$(PI_PREFIX)Pei\PeiMain.inf
Sample\Universal\Variable\Pei\Variable.inf
Sample\Bus\Scsi\ScsiBus\Dxe\ScsiBus.inf
Sample\Platform\Nt32\Pei\Autoscan\WinNtAu-
toScan.inf
……
```

FV = FvRecover指示 Build tool创建名称为 FvRecovery的 FV(Firmware Volume)。该 FV包含若干的模块:PeiMain、Variable、ScsiBus、WinNtAutoScan等。Build tool将逐个编译各个模块,并生成各自的 FF(Firmware File),如 0167CCC4-D0F7-4f21-A3EF-9E64B7CDCE8B-ScsiBus.ffs。然后将各个 FF排列在当前的 FV中。往往一个平台的 DSC文件会定义若干个 FV,这些 FV最后组合而成 FD(Firmware Device)文件。

2 EFI固件文件系统 FFS

固化在主板 Flash芯片里的 EFI的二进制文件称为 FD(Firmware Device)文件。每个 FD文件是由若干个 FV(Firmware Volume)文件组合而成,FV文件可以看做是逻辑上的 FD文件。在 EFI架构中,FV是用来存储代码和数据的单元。原则上,每个 FV都可以拥有自己的固件文件系统。因此,EFI固件文件系统分为三个层面:FV(Firmware Volume);FF(Firmware File);Section(Firmware File Section),三者关系如图 1所示。

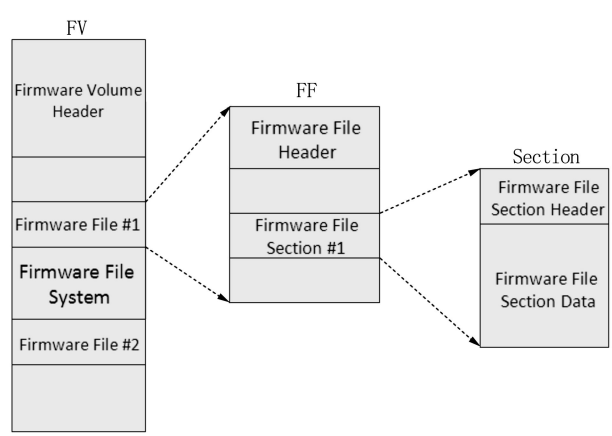

图 1 FV, FF, Section关系图

2.1 FV(Firmware Volume)

FV是存储代码和数据的基本单元,其内部遵循

Body构成。FV Header描述了 FV Body的属性。FV Body可以是 Raw Data, 也可以包含若干个 FF (Firmware File)。在 EFI架构中, FV Header有标准的数据结构定义。

```
typedef struct {
UINT8                       ZeroVector[16];
EFI_GUID                    FileSystemGuid;
UINT64                      FvLength;
UINT32                      Signature;
EFI_FVB_ATTRIBUTES          Attributes;
UINT16                      HeaderLength;
UINT16                      Checksum;
UINT16                      ExtHeaderOffset;
UINT8                       Reserved[1];
UINT8                       Revision;
EFI_FV_BLOCK_MAP_ENTRY      BlockMap[];
} EFI_FIRMWARE_VOLUME_HEADER;
```

①FileSystemGuid成员是用来定义该 FV所遵循的文件系统名称 (GUID, Globally Unique Identifier)。

②FvLength成员是该 FV大小的字节数。

③Signature成员是四个标识字符 '_', 'F', 'V', 'H'的 ASCII码。

④Attribute成员是 32 bit的数值,每个 bit都代表一种属性。譬如,该 FV是否支持 Read, Write, Lock以及 Alignment等特性要求。

⑤Checksum成员使得 FV Header所有数据做加法归零。通过这个域,可以验证这个 FV是否有效。

2.2 FF (Firmware File)

FF是构成 FV的主体,在 FV Body里按照某种文件系统格式 (由 FvHeader. FileSystemGuid成员定义)存放着。每个 FF都由 FF Header和 FF Body构成。FF Header描述了 FF Body的属性。FF Body可以是 Raw Data,也可以包含若干个 Section (Firmware File Section)。在 EFI架构中, FF Header有着标准的数据结构定义。

```
typedef struct {
EFI_GUID                    Name;
EFI_FFS_INTEGRITY_CHECK     IntegrityCheck;
EFI_FV_FILETYPE             Type;
EFI_FFS_FILE_ATTRIBUTES     Attributes;
UINT8                       Size[3];
EFI_FFS_FILE_STATE          State;
```

①Name成员为该 FF文件的名称 (GUID, Globally Unique Identifier)。

②IntegrityCheck成员使得 FF文件所有数据做加法归零。通过这个域,可以验证这个 FF是否有效。

③Type成员描述该文件是 DRIVER还是 APPLICATION还是 Foundation。EFI架构将所有文件大致分为三类: Foundation是内核程序, DRIVER是驱动程序, APPLICATION是应用程序。内核负责遍历系统中所有的 FV并搜索其中所有的 FF并运行它们。

④Attributes成员是 8 bit数值,每个 bit表示一种属性。譬如,该文件是否有对其要求,能否移动等。

⑤Size成员是该 FF文件大小的字节数。

⑥State成员表示该 FF当前处于什么状态。譬如, VALID, CONSTRUCTION, FOR_UPDATE, DELETED, INVALID等。通过操作这个域,可以表示更新,删除等文件操作状态。

2.3 Section (Firmware File Section)

Section是固件文件系统中的最小单元。每个 Section都由 Section Header和 Section Body构成。在 EFI架构中, Section Header有着标准的数据结构定义。

```
typedef struct {
UINT8                       Size[3];
EFI_SECTION_TYPE            Type;
} EFI_COMMON_SECTION_HEADER;
```

①Size成员表示 Section大小字节数。

②Type成员表示 Section Body里数据类型。所有的 Section类型都可以分成两类: Encapsulation sections and leaf sections。如图 2所示。Encapsulation sections含有其他 sections,故被称作 parent section。Leaf section直接含有数据而并不含有其他 section,故被称作 child section。因而 Section的遍历需要支持嵌套递归。通常有 PE32, TE, RAW, DEPEX, COMPRESSION, GUID_DEFINED等类型。

3 目前 EFI内核对固件文件系统的支持

EFI内核按照执行先后顺序分为两个阶段: PEI阶段和 DXE阶段。PEI内核的主要任务是遍历所有 FV将其认识的 DRIVER (PEIM, PEI阶段执行的 DRIVER)进行 Dispatch,并最终检测主板上物理内存。DXE内核的主要任务是遍历所有的 FV将其认识的 DRIVER (DXE DRIVER, DXE阶段执行的 DRIVER)进行 Dispatch, DXE阶段是 EFI执行的主

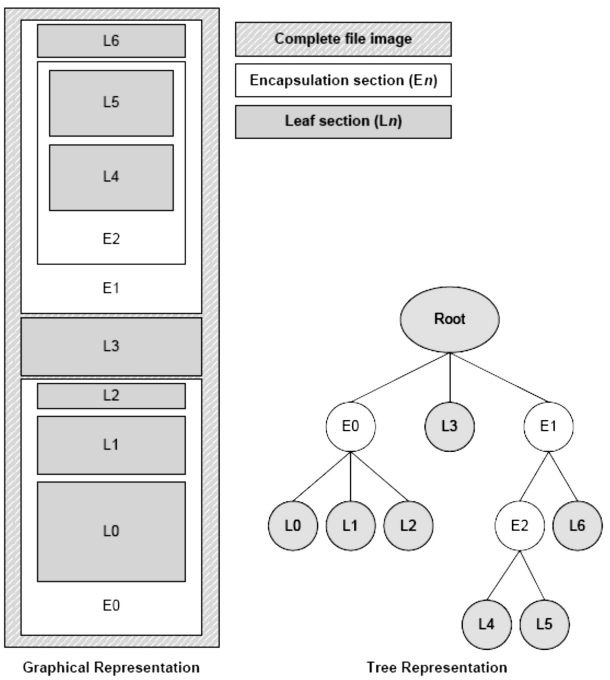

图 2 Encapsulation and leaf section

PEI内核会创建并管理 PEIDatabase,各个 PEI Driver都可以安装自己的 PPI(PEIM - to - PEIM Interface)到 PEIDatabase,同时也可以调用别的 Driver安装的 PPI。同样,DXE 内核会创建并管理 DXE Database,各个 DXE Driver都可以安装自己的 Protocol到 DXE Database,也可以调用别的 Driver安装的 Protocol。实际上,PPI和 Protocol类似 C + +中的类,封装了数据成员和成员函数。EFI正是通过 Database来对 PPI/Protocol进行管理,从而为模块化的设计提供了支持。

由此看出,无论是 PEI内核还是 DXE内核都需要事先知道各个 FV 内部的固件文件系统,这样才能对其内部的 FF进行遍历搜索并将它们 Dispatch。事实上,目前 EFI架构规定各个 FV 都采用 EFI标准的固件文件系统。这样虽然 PEI内核和 DXE内核对 FV的解析得到了简化,但是对于追求模块化设计的 EFI架构来说并不灵活。

3.1 PEI内核

PEI内核对 Flash上各个 FV的遍历搜索是基于几个 PEI Foundation Services: FfsFindNextVolume、FfsFindNextFile、FfsFindSectionData、FfsFindFileByName、FfsGetFileInfo、FfsGetVolumeInfo。这些 Services的实现无一例外的都是基于 EFI标准的固件文件系统。譬如,FfsFindNextVolume假定每个 FV 析知道 FV Body的情况。FfsFindNextFile假定每个 FF都有标准的 FF Header,通过对 FF Header进行分析知道 FF Body的情况。FfsFindSectionData则假定每个 Section都有标准的 Section Header,通过对 Section Header进行分析知道 Section Body的情况。

3.2 DXE内核

DXE内核对 Flash上各个 FV的遍历是基于 FV Protocol的。FV Protocol有若干个成员函数: FvGetVolumeAttributes, FvSetVolumeAttributes, FvReadFile, FvWriteFile, FvGetNextFile, FvReadFileSection。DXE内核自带的 FV Protocol也是基于 EFI标准的固件文件系统。

4 EFI内核对多文件系统支持的方案

事实上,为了追求 Flash存储空间利用率的最大化,厂商们可能会提出自己的固件文件系统,也就是其 FV内部并不遵照 EFI标准的固件文件系统来存放各个驱动文件。当遍历这样的 FV时,PEI、DXE内核是无法识别的。这对于追求模块化设计的 EFI架构来讲,不得不说是个遗憾。事实上,在 FV粒度上,EFI架构也应该可以提供相应的灵活性。这样,厂商们就可以发布自己的 FV文件,而不是单一的驱动文件。为此,本文提出一种支持多文件系统的解决方案。

4.1 FV布局要求

支持多文件系统,也就意味着 FV上的文件分布可以是任意形式,不必再遵循 EFI标准的固件文件系统。这样,各个厂商可以去设计各自的文件系统来提高存储空间利用率,更甚至,可以没有文件系统的概念,以 RAW Data的形式存在。那么 EFI内核如何认识各式各样的文件系统。一种简易可行的方案是要求各个 FV具有“Self - Description”功能。也只有当每个 FV具有“自描述”功能时,EFI内核才能认识形式多样的 FV。为此,该方案对 FV布局上提出一个也是唯一要求。各个 FV的起始位置存放一个“自描述”的可执行 EFI文件。该可执行 EFI文件能识别 PEI环境调用和 DXE环境调用。在被 PEI内核调用时,安装“自描述”的 FV PPI。在被 DXE内核调用时,安装“自描述的”FV Protocol。如图 3所示。

4.2 PEI内核

当处理 FV时,PEI内核首先调用该 FV开头的“自描述”EFI文件。该 EFI文件会安装 FV PPI(负责解释当前 FV的 PPI)到 PEIDatabase。FV PPI的

tVolume、FfsFindNextFile、FfsFindSectionData、FfsFindFileByName、FfsGetFileInfo、FfsGetVolumeInfo接口一致,提供了访问该FV内部文件的所有方法。这样,一种文件系统的FV需要安装一个对应的FV PPI到PEIDatabase中。PEI内核无需再对各个FV内部文件系统做任何假定。当PEI内核处理某个FV时,可以调用相应的FV PPI来遍历该FV内部的所有文件。

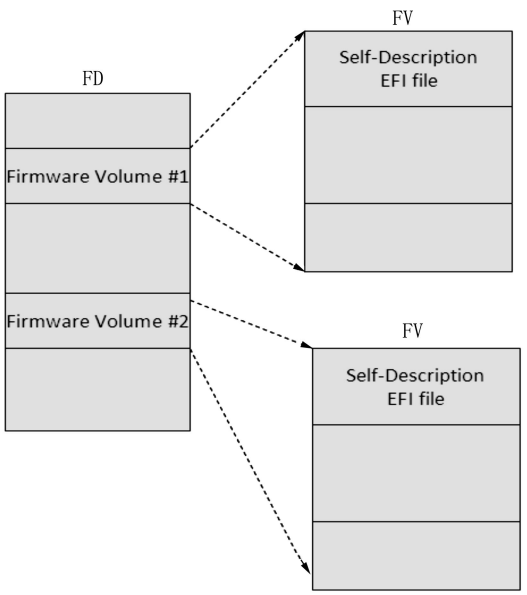

图3 FV布局

4.3 DXE内核

与PEI内核一样,DXE内核会首先调用各个FV开头的"自描述"EFI文件。该EFI文件会安装Database。DXE内核无需再对FV内部文件系统做任何假定。每个FV都安装自己的FV Protocol到DXE Database中。当DXE内核处理某个FV时,可以调用相应的FV Protocol来遍历该FV内部的所有文件。

5 结束语

本文讨论了EFI固件文件系统以及EFI内核对固件文件系统的支持。并在此基础上,指出EFI架构对多文件系统支持的不足,同时提出了一种解决多文件系统支持的方案。本文所提出的解决方案已在EFI多文件系统支持产品得到完全实现并获得满意的效果。

参考文献:

[1] Vincent Zimmer, Michael Rothman, Robert Hale. Beyond BIOS: Implementing the Unified Extensible Firmware Interface with Intel's Framework[M]. 2006: 121 - 140.

[2] UEFI, Unified Extensible Firmware Interface Specification[S]. Version 2.3, 2009: 15 - 19.

[3] UEFI, Platform Initialization Specification[S]. Version 1.2, 2009: 5 - 13.

[4] Intel Corporation, Intel Platform Innovation Framework for EFI Architecture Specification[S]. Version 0.9, 2003: 10 - 18.

[5] Intel Corporation, Intel Platform Innovation Framework for EFI Firmware Volume Specification[S]. Version 0.9, September 16, 2003: 15 - 20.

[6] Intel Corporation, Intel Platform Innovation Framework for EFI Firmware File System Specification[S]. Version 0.9, September 16, 2003: 11 - 24.

责任编辑:刘新影

(上接第102页)

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.10.1.0/24     172.15.1.1               0             0 100 i
*> 10.10.2.0/24     172.15.1.1               0             0 100 i
*> 168.15.2.0/24    0.0.0.0                  0         32768 i
*> 168.15.3.0/24    172.15.1.1                             0 100 300 i
*> 168.15.4.0/24    172.15.1.1                             0 100 300 400 i
*> 168.15.5.0/24    172.15.1.1                             0 100 500 i
*> 168.15.6.0/24    172.15.1.1                             0 100 500 600 i
*> 168.15.7.0/24    172.15.1.1                             0 100 500 600 700 i
*  172.15.1.0/24    172.15.1.1               0             0 100 i
*>                  0.0.0.0                  0         32768 i
*> 172.16.4.0/24    172.15.1.1               0             0 100 i
*> 172.16.5.0/24    172.15.1.1               0             0 100 i
*> 172.16.6.0/24    172.15.1.1                             0 100 300 i
*> 172.16.7.0/24    172.15.1.1                             0 100 500 i
*> 172.16.8.0/24    172.15.1.1                             0 100 500 i
*> 172.16.10.0/24   172.15.1.1                             0 100 500 600 i

Total number of prefixes 15
R bgpd#
```

图2 BGP路由信息

4 结束语

由图2的测试结果可见,设计的嵌入式路由协议系统能够方便地对路由软件进行配置。配置好的路由协议软件工作状态良好,可以利用设计的系统对BGP路由协议进行更进一步的研究和完善。

另外,由于篇幅所限,只以常用路由器设置为例,介绍了配置后的实验结果。实际上,根据路由协议的标准,本系统也可以进行更详细的路由协议的功能实现。

参考文献:

[1] Sam. Haibi, Danny McPherson. Internet Routing Architectures[M]. Cisco-Press 2000.

[2] 李超,肖键. 嵌入式Linux开发技术与应用[M]. 北京:电子工业出版社, 2008.

[3] Dianel P. Bovet Marco Cesate. Understanding the Linux Kernel [M]. OReilly Media, Inc. 2003.

[4] 卢泽新,白建军,等. IP路由协议疑难解析[M]. 北京:人民邮电出版社, 2008.

[5] 葛建立,吴剑章. TCP/IP路由技术[M]. 北京:人民邮电出版社, 2007.

[6] 白建军. 核心路由器边界网关协议BGP-4实现技术的研究[D]. 长沙:国防科学技术大学, 2002.

[7] Randy Zhang, Micah Bartell. BGP Design and implementation[M]. Ciso-press 2003.

责任编辑:李光辉

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
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究
65. 具有 MVB 接口的输入输出设备的分析
66. 基于 STM32 的 GSM 模块综合应用
67. 基于 ARM7 的 MVB CAN 网关设计
68. 机车车辆的 MVB CAN 总线网关设计
69. 智能变电站冗余网络中 IEEE1588 协议的应用
70. CAN 总线的浅析 CANopen 协议
71. 基于 CANopen 协议实现多电机系统实时控制
72. 以太网时钟同步协议的研究
73. 基于 CANopen 的列车通信网络实现研究
74. 基于 SJA1000 的 CAN 总线智能控制系统设计
75. 基于 CANopen 的运动控制单元的设计
76. 基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现

77. 分布式控制系统精确时钟同步技术
78. 基于 IEEE 1588 的时钟同步技术在分布式系统中应用
79. 基于 SJA1000 的 CAN 总线通讯模块的实现
80. 嵌入式设备的精确时钟同步技术的研究与实现
81. 基于 SJA1000 的 CAN 网桥设计
82. 基于 CAN 总线分布式温室监控系统的设计与实现
83. 基于 DSP 的 CANopen 通讯协议的实现
84. 基于 PCI9656 控制芯片的高速网卡 DMA 设计
85. 基于以太网及串口的数据采集模块设计
86. MVB1 类设备控制器的 FPGA 设计
87. MVB 接口彩色液晶显示诊断单元的显示应用软件设计
88. IPv6 新型套接字的网络编程剖析
89. 基于规则的 IPv4 源程序到 IPv6 源程序的移植方法
90. MVB 网络接口单元的 SOC 解决方案
91. 基于 IPSec 协议的 IPv6 安全研究
92. 具有 VME 总线的车载安全计算机 MVB 通信板卡
93. SD 卡的传输协议和读写程序
94. 基于 SCTP 的 TLS 应用
95. 基于 IPv6 的静态路由实验设计
96. 基于 MVB 的地铁列车司机显示系统研究
97. 基于参数优化批处理的 TLS 协议
98. SSD 数据结构与算法综述
99. 大容量 NAND Flash 文件系统中的地址映射算法研究
100. 基于 MVB 总线的动车组门控系统的设计与仿真研究

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
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现
43. 实时操作系统 VxWorks 在微机保护中的应用
44. 基于 VxWorks 的多任务程序设计及通信管理
45. 基于 Tilcon 的 VxWorks 图形界面开发技术
46. 嵌入式图形系统 Tilcon 及应用研究
47. 基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现
48. 基于嵌入式的 Tilcon 用户图形界面设计与开发
49. 基于 Tilcon 的交互式多页面的设计
50. 基于 Tilcon 的嵌入式系统人机界面开发技术
51. 基于 Tilcon 的指控系统多任务人机交互软件设计
52. 基于 Tilcon 航海标绘台界面设计
53. 基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发

54. VxWorks 环境下内存文件系统的应用
55. VxWorks 下的多重定时器设计
56. Freescale 的 MPC8641D 的 VxWorks BSP
57. VxWorks 实验五[时间片轮转调度]
58. 解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题
59. 基于 VxWorks 系统的 MiniGUI 图形界面开发
60. VxWorks BSP 开发中的 PCI 配置方法
61. VxWorks 在 S3C2410 上的 BSP 设计
62. VxWorks 操作系统中 PCI 总线驱动程序的设计与实现
63. VxWorks 概述
64. 基于 AT91RM9200 的 VxWorks END 网络驱动开发
65. 基于 EBD9200 的 VxWorks BSP 设计和实现
66. 基于 VxWorks 的 BSP 技术分析
67. ARM LPC2210 的 VxWorks BSP 源码
68. 基于 LPC2210 的 VxWorks BSP 移植
69. 基于 VxWorks 平台的 SCTP 协议软件设计实现
70. VxWorks 快速启动的实现方法[上电到应用程序 1 秒]

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
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略
52. Linux 基于 W83697 和 W83977 的 UART 串口驱动开发文档
53. 基于 AT91RM9200 的嵌入式 Linux 系统的移植与实现
54. 路由信息协议在 Linux 平台上的实现
55. Linux 下 IPv6 高级路由器的实现
56. 基于 Android 平台的嵌入式视频监控系统设计

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
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现
28. 基于 XPE 的数字残币兑换工具开发
29. Windows CENET 下 ADC 驱动开发设计
30. Windows CE 下 USB 设备流驱动开发与设计
31. Windows 驱动程序设计
32. 基于 Windows CE 的 GPS 应用
33. 基于 Windows CE 下大像素图像分块显示算法的研究
34. 基于 Windows CE 的数控软件开发与实现
35. NAND FLASH 在 WINCENET 系统中的应用设计

PowerPC：

WeChat ID: kontronn

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
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现
27. 基于 PowerPC 和 DSP 的电能质量在线监测装置的研制
28. 基于 PowerPC 架构多核处理器嵌入式系统硬件设计
29. 基于 PowerPC 的多屏系统设计
30. 基于 PowerPC 的嵌入式 SMP 系统设计
31. 基于 MPC850 的多功能通信管理器
32. 基于 MPC8640D 处理系统的技术研究
33. 基于双核 MPC8641D 处理器的计算机模块设计
34. 基于 MPC8641D 处理器的对称多处理技术研究

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-OS II 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
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现
36. UBoot 在 AT91RM9200 上的移植简析
37. 基于工控级 AT91RM9200 开发板的 UBoot 移植分析
38. 基于 ARM11 和 Zigbee 的人员定位防丢器
39. 基于 NAND FLASH 的嵌入式系统启动速度的研究

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
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法
34. 基于 GPU 的瑕疵检测
35. 基于 GPU 通用计算的分析与研究
36. 面向 OpenCL 架构的 GPGPU 量化性能模型
37. 基于 OpenCL 的图像积分图算法优化研究
38. 基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究

39. 基于 OpenCL 的异构系统并行编程
40. 嵌入式系统中热备份双机切换技术研究
41. EFI-Tiano 环境下的 AES 算法应用模型
42. EFI 及其安全性研究
43. 基于 UEFI Shell 的 PreOS Application 的开发与研究
44. UEFI Bootkit 模型与分析
45. UEFI 计算机系统快速调试方法的实现

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
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案
10. 基于小波变换与偏微分方程的图像分解及边缘检测
11. 基于图像能量的布匹瑕疵检测方法
12. 基于 OpenCL 的拉普拉斯图像增强算法优化研究
13. 异构平台上基于 OpenCL 的 FFT 实现与优化
14. 数据结构考题 - 第 4 章 串
15. 数据结构考题 - 第 4 章 串答案
16. 用 IPv6 编程接口实现有连接通信的方法

FPGA / CPLD:

1. 一种基于并行处理器的快速车道线检测系统及 FPGA 实现
2. 基于 FPGA 和 DSP 的 DBF 实现
3. 高速浮点运算单元的 FPGA 实现
4. DLMS 算法的脉动阵结构设计及 FPGA 实现
5. 一种基于 FPGA 的 3DES 加密算法实现
6. 可编程 FIR 滤波器的 FPGA 实现
7. 基于 FPGA 的 AES 加密算法的高速实现

8. 基于 FPGA 的精确时钟同步方法
9. 应用分布式算法在 FPGA 平台实现 FIR 低通滤波器
10. 流水线技术在用 FPGA 实现高速 DSP 运算中的应用
11. 基于 FPGA 的 CAN 总线通信节点设计
12. 基于 FPGA 的高速时钟数据恢复电路的实现
13. 基于 FPGA 的高阶高速 FIR 滤波器设计与实现
14. 基于 FPGA 高效实现 FIR 滤波器的研究
15. FPGA 的 VHDL 设计策略
16. 用 FPGA 实现串口通信的设计
17. GPIB 接口的 FPGA 实现
18. 一种基于 FPGA 的 FFT 阵列处理器
19. 基于 FPGA 的 FFT 信号处理器的硬件实现