基于龙芯平台的 PMON 研究与开发

吴亚杰[1]，刘卫东[1,2]，曾小光[2]

（1. 中国海洋大学 信息科学与工程学院，山东 青岛 266100；2. 海信电器股份有限公司 山东 青岛 200071）

摘要：在嵌入式软件系统开发中，为了能够快速正确地引导操作系统，需要有一个功能强大的 Bootloader 来支持。PMON 是一种针对嵌入式系统而开发的操作系统引导程序，其作用在于初始化嵌入式硬件系统的外设以便能够正确引导、启动操作系统。基于 PMON 开发移植的目的，从研究 PMON 的架构出发，深入研究 PMON 的编译环境及其执行流程，在此基础上，通过向 PMON 中添加网卡驱动代码，并验证了网卡驱动工作的正确性，最终实现了 PMON 下的网卡驱动移植。

关键词：龙芯；PMON；网卡驱动；嵌入式系统

中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1674-6236(2011)17-0140-03

Research and development of PMON based on Loongson platform

WU Ya-jie[1], LIU Wei-dong[1,2], ZENG Xiao-guang[2]

(1. *Collegel of Information Science and Engineering*, *Ocean University of China*, *Qingdao* 266100, *China*;
2. *Hisense Electric Appliance Limited Company*, *Qingdao* 200071, *China*)

Abstract: To boot the operating system quickly and correctly, a powerful bootloader is needed during the development of the embedded software system. PMON is a bootstrap program designed for embedded system, and it's used to initialize the devices on the hardware system, after that, the operating system could boot validly. Based on the purpose of developing and transplanting of the PMON , the compile environment and executing process of PMON are researched deeply from the view of the basic PMON's framework . On this basis, add the Network driver to PMON and verify the correctness of the network driver. Ultimately, the transplant of the network device is finished correctly.

Key words: Loongson; PMON; network driver; embedded system

龙芯是中国科学院计算所研制的通用 CPU，已获得 MIPS 科技公司的 MIPS 指令集的专利授权。龙芯 1 号的 CPU 主频是 266 MHz，最早在 2002 年开始产业化应用。龙芯 2 号主频最高为 1 GHz。龙芯 3 号于 2010 年推出成品，其设计的目标则在多核心的设计[1]。随着龙芯的发展，龙芯 CPU 已不仅仅局限于个人桌面计算机领域，在嵌入式开发领域，龙芯 CPU 也同样发展迅速，从 2000 年以来，越来越多的产品开始采用龙芯 CPU。在计算机的体系结构中，无论是个人计算机、服务器还是嵌入式领域，基本输入输出系统 BIOS 是必不可少的，因为 BIOS 负责计算机系统的开机自检、板级硬件初始化、加载操作系统内核以及基本 I/O 功能。

PMON 是一款 ROM-Monitor 型的开源软件，最初是为了 LSI Logic MIPS R3000 评估板的功能需求而开发的。经过多年发展，目前已经能够支持 MIPS、ARM、PPC 和 X86 等 CPU 体系[2]。PMON 具有强大而丰富的功能，除基本的 I/O 功能外，还包括 CPU 初始化、板级外设初始化与检测、操作系统引导和调试等功能，并且 PMON 支持从 Flash、IDE、TFTP 以及 USB 来启动操作系统。

1　PMON 框架分析

1.1　PMON 目录结构说明

PMON 源代码的目录结构如图 1 所示，对于图示中关键模块说明如下：

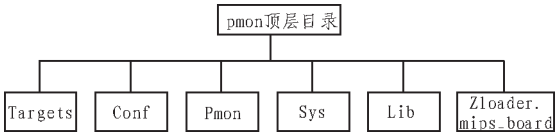

图 1　PMON 目录结构

Fig. 1　PMON composition

1)Targets 目录　Targets 目录下存放的是与板级相关的代码，该目录下的每个子目录都对应着某一个具体的开发板，当要将 PMON 移植到一个新的开发板时，就需要在该目录新建一个子目录，并向新建的子目录中添加开发板相关的代码，其中主要有以下几个重要文件：start.S 文件位于 Targets/mips_board/mips_board 目录下，是整个 PMON 运行的起点；tgt_machdep.c 文件位于 Targets/mips_board/mips_board 目录下，完成大部分板级外设的初始化工作；Targets/

mips_board/dev 目录存放板级外设的驱动程序，所需移植的网卡驱动文件即存放于此目录中；Targets/mips_board/conf 目录主要存放与硬件板相关的配置文件。大部分的文件都在 Targets/mips_board/compile/mips_board 目录中完成编译，调试用的 pmon.gdb 文件即位于此目录。

2）conf 目录　Conf 目录下存放的是整个 PMON 系统的配置文件。

3）pmon 目录　该目录下存放的是 PMON 公用的代码，包括 PMON 所支持的各种命令，与 CPU 相关的代码以及文件系统相关的代码，主要有以下几个子目录：arch 目录下的子目录存放的是与 CPU 相关的代码；cmds 目录下存放的是各种在 PMON 中可以使用的命令文件，比如：ifup、devcp、g 等命令，如果要想向 PMON 中添加新的命令，需要在此目录下添加源文件、实现该命令功能即可；fs 目录下存放的是与各文件系统相关的代码；common 目录下存放的是一些通用代码，比如：命令解析、调试接口、异常处理、环境变量设置程序等部分；netio 目录下存放的是与网络相关的命令的实现代码。

4）Sys 目录　Sys 目录存放的是系统支持文件。

5）Lib 目录　Lib 目录存放的是库的实现代码。

6）zloader.mips_board 目录　最终烧写到 Nand Flash 中的的 gzrom.bin 文件就是在该目录下经过链接而生成的。

1.2　PMON 初始化流程

当开发板上电之后，CPU 即从 0xBFC00000 处取指令执行，整个 PMON 的入口位于 start.S 文件。该汇编程序主要完成 CPU 的初始化工作，设置异常向量入口、设置栈、初始化 UART、初始化内存、初始化 CACHE，并完成对 PMON 的代码拷贝工作，即由 Nor Flash 搬运到 SDRAM，以提高代码执行速度[3]。最后 PC 指针跳转到 PMON 的 C 入口 initmips 函数处继续执行，从此进入 C 语言的执行环境。整个执行流程如图 2 所示。

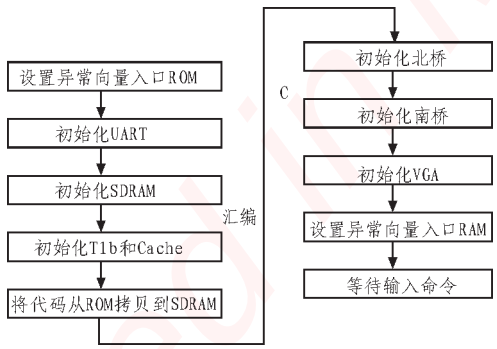

图 2　PMON 初始化流程

Fig. 2　Flow chart of PMON initializing

在 initmips 函数中主要通过 dbginit 这个函数来完成大部分的初始化工作，主要有以下几个函数来实现初始化工作：

1）__init 函数：初始化带有 __attribute__ ((constructor)) 属性的函数。

2）envinit 函数：环境变量初始化。

3）init_net 函数：网络初始化，网卡设备的部分初始化也在这个函数中完成。

4）histinit 函数：初始化历史命令记录。

在 initmips 函数完成初始化任务后，即跳转到 pmon/common/main.c 中的 main 函数执行，在 main 函数中设置完一些参数后，即进入一个 while 循环，等待用户输入命令，while 循环内部主要有两个函数 get_line 和 do_cmd 函数。get_line 函数一直试图获取用户输入的命令，而 do_cmd 函数负责解析命令，解析成功后，则分派相应的命令函数去执行；解析失败则返回到 while 循环，继续等待用户输入命令。执行到这里 PMON 已经完全运行起来了。此时如果需要加载内核，用 load 命令将内核加载到内存中，接着用 g 命令则传递参数给内核，并开始启动操作系统。

1.3　PMON 中调试命令介绍

PMON 与其他 Bootloader 相比，其优势在于 PMON 的调试功能强大。PMON 本身能支持设置断点命令 b、查看/设置寄存器命令 r、单步执行命令 t、查看堆栈信息命令 bt 以及继续执行命令 c 等调试相关的命令。b 命令用于设置断点，需要注意的是在 PMON 中最多可以支持 32 个断点。r 命令用于显示/设置 CPU 寄存器，直接输入 r 后会打印所有寄存器的信息。t 命令用于单步执行。bt 命令用于显示当前堆栈信息。c 命令用于继续执行，即从当前断点处继续往下执行，相当于 gdb 的 continue 命令。除了上面列出的调试命令外，PMON 还支持很多其它命令，比如：用于烧写 Nor Flash 的 devcp 命令、显示设备的 devls 命令、设置环境变量的 set 命令、显示环境变量的 env 命令、加载文件的 load 命令、运行程序的 g 命令等。

2　PMON 中网卡驱动的实现

2.1　PMON 查找设备

以上重点描述了 PMON 的整体执行过程，接下来就要具体实现在 PMON 中的网卡移植过程。首先要在配置文件 Targets/Hiview/conf/file.Hiview 中添加如下部分：

```
device    fxp: ether, ifnet, ifmedia, mii
attach    fxp at localbus
file  Targets/Hiview/dev/net_fxp.c  fxp
```

上面这部分内容定义了网卡挂载的总线，以及需要编译的网卡驱动的源代码文件等，在重新编译 PMON 时需要执行 make cfg 这个命令，此时会读取配置文件，从而生成一个名为 cfdata 的数组，在 PMON 的启动过程中会通过 configure 函数去配置已知的各个设备，并通过扫描有哪些设备挂在了总线上，PMON 根据 cfdata 数组依次扫描设备。PMON 首先通过 config_rootfound 函数来查找根设备，查找成功后再通过 config_rootsearch 函数来查找根设备上的子设备，子设备查找成功后则执行相应的子设备的挂载函数，通知 PMON 该子设备已找到，并将相应的子设备操作函数注册到 PMON 中。

2.2　网卡初始化

若网卡设备查找成功，则执行网卡的挂载函数，即 fxp_attach 函数，在 fxp_attach 函数中完成中断处理函数

fxp_intr 的注册，调用 tgt_poll_register 函数将中断处理函数 fxp_intr 注册到查询列表 poll_list 上。在 fxp_attach 函数中完成的另外一个重要工作是将网卡驱动的函数添加到 PMON 中，以便 PMON 的上层接口能够正确调用到网卡设备的下层驱动函数来实现功能，这里通过填充 net_device 结构体来实现，如下代码即实现了该工作：

```
struct lakers_priv  *priv;
priv = netdev_priv(ndev);
priv->ndev = ndev;
……
ndev->open = net_fxp_open;
ndev->stop = net_fxp_close;
ndev->hard_start_xmit = net_fxp_hard_start_xmit;
……
```

其中打开网络设备通过 net_fxp_open 函数来完成，net_fxp_open 主要工作是初始化网卡设备的相关寄存器，并分配用于接收、发送数据的缓冲区，设置好缓冲区的状态。net_fxp_close 函数则是在关闭网络设备时调用，主要完成清除发送队列，关闭网卡的发送、接收使能等工作。net_fxp_hard_start_xmit 则负责启动网卡发送数据[4]。

2.3 网卡发送与接收数据过程

当网卡设备接口处有数据传进来时就会触发一个中断，然后调用网卡接收程序 net_fxp_rx 函数进行处理。当网卡接收程序 net_fxp_rx 接收完数据或者网卡发送程序 net_fxp_hard_start_xmit 发送完数据后，也会触发一个中断，fxp_intr 对接收到的中断进行检测，扫描网卡设备的中断寄存器，判断是接收中断还是发送完毕中断，然后根据检测结果跳转到不同的处理函数去执行，如果是接收中断，则转到 net_fxp_rx_poll 函数中去处理传过来的数据，并将其传递给上层协议。如果是包发送完毕中断，则跳转到 net_fxp_tx_done 函数，通过该函数检查网卡的发送状态并记录下发送数据的字节数等信息，检查发送队列判断是否要接着发送数据，还是发送数据任务已经全部完成。若是已完成数据的发送，则更新缓冲区状态，然后返回到中断处理函数[5]。

2.4 编译 PMON

龙芯 LS232 CPU 是兼容 MIPS 指令集的，故在该 CPU 平台下可采用 MIPS 的工具链。本文的开发环境是 REHL 5.5 操作系统，gcc 编译器采用的是 gcc-3.4.6 版本，在制作交叉工具链时需要加上- -target=mipsel-linux 参数[6]。添加完网卡驱动后，需要重新编译 PMON，依次执行以下命令：

```
# 进入 pmon/tools/pmoncfg 目录执行 make 命令，生成配置工具 pmoncfg
make
# 将 pmoncfg 拷贝到/usr/bin 目录下
cp pmoncfg /usr/bin
# 进入 pmon/zloader.mips_boadr 目录（该目录是 zloader 的一个软连接），根据配置文件，重新生成 makefile
make cfg
# 生成带调试信息的 rom bin 文件
make tgt=rom CROSS_COMPILE=mipsel-linux- ARCH=mips DEBUG=-g
```

编译成功后，会在此目录下生成一个 gzrom.bin 文件，将其烧入 Nor Flash 的 0xbfc00000 地址处即可[7]。

2.5 测试结果

在添加完网卡驱动后，PMON 重新编译成功。启动 PMON 后，通过使用 ping 程序测试（如图 3 所示），测试结果表明网卡驱动功能正常。

```
PMON> ping 172.16.14.27
PING 172.16.14.27 (172.16.14.27): 56 data bytes
64 bytes from 172.16.14.27: icmp_seq=0 ttl=64 time=0.625 ms
64 bytes from 172.16.14.27: icmp_seq=1 ttl=64 time=0.372 ms
64 bytes from 172.16.14.27: icmp_seq=2 ttl=64 time=0.373 ms
64 bytes from 172.16.14.27: icmp_seq=3 ttl=64 time=0.368 ms
64 bytes from 172.16.14.27: icmp_seq=4 ttl=64 time=0.362 ms
64 bytes from 172.16.14.27: icmp_seq=5 ttl=64 time=0.378 ms
64 bytes from 172.16.14.27: icmp_seq=6 ttl=64 time=0.364 ms
64 bytes from 172.16.14.27: icmp_seq=7 ttl=64 time=0.377 ms

--- 172.16.14.27 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.362/0.390/0.625 ms
```

图 3 ping 测试

Fig. 3 Ping testing

3 结束语

本文是研究基于龙芯平台下的一种 Bootloader（PMON）的实现。分别分析了 PMON 的整体框架、初始化流程、PMON 的源码，在此基础上进行了网卡驱动的移植工作。代码编写及网卡移植后，完成单元测试、功能验证，PMON 及网卡模块功能正常、运行稳定。

参考文献：

[1] 龙芯官方论坛. 龙芯的历程[EB/OL]. (2011-05-04)[2011-06-09].http://www.loongson.cn/about_two.php?id=10&sub=龙芯的历程.

[2] PMON-LinuxMIPS. PMON [EB/OL]. (2010-02-08)[2011-06-10].http://www.linux-mips.org/wiki/PMON.

[3] aaaaatiger. PMON启动流程[EB/OL]. (2007-06-04)[2011-06-12].http://blog.csdn.net/aaaaatiger/article/details/1638182.

[4] 宋宝华. Linux设备驱动开发详解[M]. 2版. 北京：人民邮电出版社，2010.

[5] CorbetJ. LINUX设备驱动程序[M]. 魏永明，耿岳，钟书毅，译.北京：中国电力出版社，2006.

[6] STRONGCHINA. Loongson GCC 安装和发布事项 2.2[EB/OL]. (2008 -10 -07) [2011 -06 -23].http://bbs.lemote.com/viewthread.php?tid=18816&extra=page%3D1.

[7] CAIMOUSE. 编译PMON指南 [EB/OL]. (2006-12-24)[2011 -06 -23]. http://www.lemote.com/bbs/viewthread.php?tid=3147&extra=page%3D1%26filter%3Ddigest.

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

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

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

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
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

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