基于设备树的 MPC8247 嵌入式 Linux 系统开发

张茂天[1*]，张　磊[1]，郭　晓[1]，孙　钧[2]
（1. 解放军理工大学 通信工程学院，南京 210007；　2. 中国人民解放军 94654 部队，南京 210000）
（* 通信作者电子邮箱 maotianzhang@gmail.com）

摘　要 针对基于 PowerPC 架构处理器的 MPC8247 目标系统，在分析 PowerPC 架构设备树原理的基础上，进行了嵌入式 Linux 的系统开发，包括 U-Boot、Linux 内核、设备树对象和 Ramdisk 根文件系统的移植和部署等。系统的实际运行情况表明，设备树文件编写正确，系统设计合理高效。

关键词 嵌入式 Linux 系统；Linux 内核；MPC8247；PowerPC；设备树对象；引导加载程序
中图分类号 TP311.52；TP316.81　　**文献标志码** A

Development of MPC8247 embedded Linux system based on device tree

ZHANG Maotian[1*], ZHANG Lei[1], GUO Xiao[1], SUN Jun[2]
(1. *College of Communications Engineering, PLA University of Science and Technology, Nanjing Jiangsu* 210007, *China*;
2. *Unit No.* 94654 *of PLA, Nanjing Jiangsu* 210000, *China*)

Abstract: Concerning the MPC8247 target system based on PowerPC, the device tree was discussed and an embedded Linux system was developed, including the transplant and deployment of U-Boot, Linux kernel, Device Tree Blob (DTB) and Ramdisk file system. The actual operation of the system shows that the device tree file is correct, and the system design is rational and efficient.

Key words: embedded Linux system; Linux kernel; MPC8247; PowerPC; Device Tree Blob (DTB); bootloader

0　引言

凭借经济和技术方面的诸多优势，Linux 正被越来越多的嵌入式设备所使用。同时，不断更新的 Linux 内核在支持嵌入式系统各方面做了很多改进，有利地促进了 Linux 在嵌入式系统中的应用。

PowerPC 架构是嵌入式领域最为流行和成功的架构之一，基于 PowerPC 架构的处理器已经应用于各种嵌入式产品[1-2]。但是，将 PowePC Linux 移植到新目标板上时，一个更具挑战的方面是要满足设备树对象 Device Tree Blob，DTB 的需求。之前的研究工作中，文献[3-6]只是详细介绍设备树的概念，文献[7]关注嵌入式 Linux 的系统开发过程而忽视了设备树的重要地位。

本文首先介绍了作者自己开发的 MPC8247 目标系统，接着应对设备树对象带来的挑战，分析了应用于 PowePC 架构的设备树概念、结构和存储方式，并实现了针对 MPC8247 目标系统的设备树配置和移植，最后，结合设备树移植，介绍了嵌入式 Linux 系统开发和部署过程。

1　MPC8247 目标系统

MPC8247 目标系统，以下称 MPC8247 开发板，是基于 MPC8247 处理器[8]的嵌入式开发平台。MPC8247 处理器属于 PowerQUICC Ⅱ通信处理器家族，集成 PowerPC RISC 微处理器，其核心是 MPC603e 变种的 G2_LE，CPU 频率为 266～400 MHz，通信处理器模块（CPM）包括多个网口和串口控制器。MPC8247 是面向高性能、低功耗、小体积的通信设备而开发的通用通信处理器。

MPC8247 开发板集成 MPC8247 系列处理器，128 MB 的 SDRAM 以及 16 MB 的 FLASH，为用户的软件研发提供了足够的空间。开发板提供非常丰富的外设接口，两个 10 Mb/s / 100 Mb/s Fast Eth 以太网接口、三个三线 RS-232 串口，一个标准 16 线 JTAG 接口，具有体积小、耗电低、处理能力强、网络功能强大等特点，能够装载和运行嵌入式 Linux 操作系统。

图 1 显示了 MPC8247 开发板的硬件组成。图 2 是 MPC8247 开发板的硬件框图，其中 SMC 作为调试串口，JTAG 主要用于连接 BDI2000 调试器和目标板，Fast Eth 主要用于 TFTP 文件传输和挂载根文件系统等。

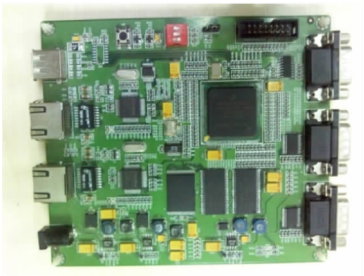

图 1　MPC8247 开发板实物

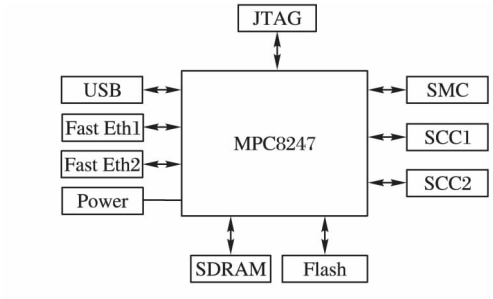

图 2　MPC8247 硬件结构

2 设备树

早期的嵌入式 PowerPC 使用 include/asm-ppc/ppcboot.h 定义的 bd_info 在引导加载程序和内核间传递数据。但是，改变 bd_info 将导致重新定制编译引导加载程序和内核，也就是说，内核只能针对单一的平台进行配置和编译。随着内核支持 32 位 arch/ppc 和 64 位 arch/ppc64 PowerPC，以及内核代码迁移到统一的 arch/powerpc 目录，旧的固件接口也得到清理，所有 PowerPC 平台为 Linux 内核提供一种新的 Open Firmware 规范的设备树[4]。内核通过读取解析设备树，确定平台的硬件配置。设备树的优点在于 规范化的硬件描述，多平台内核镜像，简化的板卡端口，更少的平台描述代码，以及更精简的设备驱动代码。

2.1 设备树描述

简单地说，设备树，也称为扁平设备树，是一种描述硬件配置的树形数据结构[5]。它包括有关 CPU、内存、总线以及外设等相关的信息。操作系统在启动时能够解析设备树，并决定怎样配置内核以及加载哪种设备驱动。设备树仅有一个根节点“/”，每个根节点有一个名字以及任意数量的子节点。设备树的数据格式继承了 IEEE standard 1275[6] 的定义。

设备树对象 .dtb 是操作系统所需的设备树二进制文件，由可读可编辑文本形式的设备树源码文件 Device Tree Source，DTS，.dts 经过设备树编译器 Device Tree Compiler，DTC 编译而成。

为了清楚地说明设备树的结构，以 MPC8247 开发板为例给出其设备树源码，其中省略了部分子节点的描述代码。在内核的 arch/powerpc 下没有 MPC8247 开发板的 DTS 文件，所以本文在 mgcoge.dts 基础上针对 MPC8247 开发板的硬件配置进行修改和编写。

```
/ {
    model = "MGCOGE";
    compatible = "keymile,km82xx";
    #address-cells = <1>;
    #size-cells = <1>;
    cpus {
            #address-cells = <1>;
            #size-cells = <0>;
            PowerPC,8247@0 {
               device_type = "cpu";
               reg = <0>;
               …
            };
    localbus@f0010100 {
      compatible = "fsl,mpc8247-localbus", "fsl,pq2-localbus",
        "simple-bus";
      #address-cells = <2>;
      #size-cells = <1>;
      reg = <0xf0010100 0x40>;
      ranges = <0x0 0x0 0xff000000 0x01000000>;
        /* 16MB 闪存* /
      flash@0,0 {
        compatible = "cfi-flash";
        reg = <0x0 0x0 0x1000000>;
        #address-cells = <1>;
        #size-cells = <1>;
        bank-width = <2>;
        device-width = <1>;
        partition@0x0 {
           label = "free space";
           reg = <0x00000000 0x00f00000>;
        partition@0x00f00000 {
           label = "u-boot";
           reg = <0x00f00000 0x00080000>;
           read-only;
        };
      };
    };
    memory {
       device_type = "memory";
       reg = <0 0>;
    };
    soc@f0000000 {
      #address-cells = <1>;
      #size-cells = <1>;
      compatible = "fsl,mpc8247-immr", "fsl,pq2-soc",
        "simple-bus";
      ranges = <0x00000000 0xf0000000 0x00053000>;
      device_type = "soc";
      cpm@119c0 {
        #address-cells = <1>;
        #size-cells = <1>;
        #interrupt-cells = <2>;
        compatible = "fsl,mpc8247-cpm",
          "fsl,cpm2", "simple-bus";
        reg = <0x119c0 0x30>;
        ranges;
        smc1: serial@11a80 {
           device_type = "serial";
           compatible = "fsl,mpc8248-smc-uart",
             "fsl,cpm2-smc-uart";
           reg = <0x11a80 0x20 0x87fc 2>;
           interrupts = <4 8>;
           interrupt-parent = <&PIC>;
        };
     <…省略部分节点代码…>
     chosen {
       linux,stdout-path = "/soc/cpm/serial@11a80";};
};
```

从设备树源码中可以看出，有一个根节点“/”，和许多子节点，每个节点都指定名称、属性和相应的属性值。其中，compatible 属性指明节点正在描述的设备名称，address-cells 和 size-cells 属性在节点中指定一个地址 或大小 所需 cell 32 位的字段 的数量，reg 属性指明设备地址，这里的设备地址是总线地址而非系统地址，ranges 属性可以将总线地址映射到父节点一级。

以上设备树源码列出了根节点、CPU 节点、总线节点、内存节点、片上系统节点和/chosen 节点。值得说明的是，根据 MPC8247 开发板的硬件配置，主要修改了 flash、smc1 和/chosen 子节点的描述代码，使得内核能够加载 flash、串口，取得环境变量和默认输入输出设备。

2.2 设备树对象存储格式

图 3 是设备树对象在内存中的存储布局，包括 3 个部分：内存保留表、字符串块和结构块[4]。一个简单的头部给出了设备树对象的大小和版本，3 个组成部分的位置以及早期启动时的重要参数。

内存保留表规定了内核不能使用的内存区域。字符串块包含很多有结束标志的属性名字符串，在设备树的结构块中存储了这些字符串的偏移地址，可以很容易地查找到属性名字符串，从而节省存储空间。结构块是设备树的主体部分，以节点形式保存目标板的设备信息。每个节点引入一个 32 位的 OF_DT_BEGIN_NODE 标志，紧跟着是有结束标示字符串

形式表示的节点名。其后是节点的所有属性，每个属性开头一个 OF_DT_PROP 标志，节点结束以 OF_DT_END_NODE 为标志。根节点结束后，出现 OF_DT_END 标志，表示整个设备树的结束。

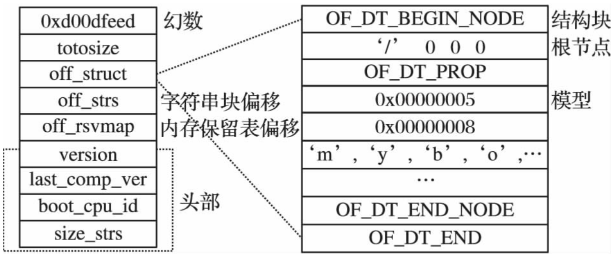

图 3 设备树对象存储格式图

2.3 Linux 内核中设备树的使用

设备树在内核中的使用分为两个阶段，一是早期启动阶段，二是设备初始化阶段[9]。在早期启动阶段，arch/powerpc 为所有 PowerPC 平台提供唯一的进入点。存在一个指向内存中存储的设备树对象的指针，在跳至进入点之前，内核需要该指针存放在 r3 寄存器中。每一个支持平台定义一个 machdep_calls 结构，内核调用 probe_machine（函数，遍历 machine_desc 表，并为每一个表调用.probe（钩子函数。每一次调用.probe（钩子函数都将测试设备树，如果平台代码支持设备树描述的板卡，就返回"真"，并继续系统启动过程。

在设备初始化阶段，存在多种方法使得平台代码获知设备描述信息并将其注册至内核。目前大部分使用设备树的嵌入式平台利用 of_platform 总线进行设备初始化。of_platform 总线是一种软件结构，而非硬件总线，它将设备注册至设备树中的设备模型。arch/powerpc/platforms 的平台代码可以利用 of_platform_bus_probe（为每一个设备注册一个 struct_of_device 函数，设备驱动依次注册一个 struct_of_platform_driver 函数，则 of_platform 可以将驱动加载至相应设备，最终实现设备初始化，内核可以正确地驱动硬件。

3 嵌入式 Linux 系统开发和部署

嵌入式系统及应用软件的开发一般采用连接式设置，即主机—目标板模式，通常主机和目标板的体系架构和操作系统不同。针对 PowerPC 架构的嵌入式 Linux 系统软件开发，主要有以下几个步骤：

1 构建交叉开发环境；

2 针对具体硬件平台，进行嵌入式 Linux 系统软件的移植和开发，包括 U-Boot、Linux 内核、设备树对象和根文件系统的移植开发；

3 目标板调试运行。

3.1 交叉开发环境的构建

开发主机运行 X86 架构下的 Linux Fedora 16 操作系统，目标平台运行 PowerPC 架构的嵌入式 Linux 操作系统。主机和目标板开发设置类型为连接式设置，接口连接如图 4 所示。

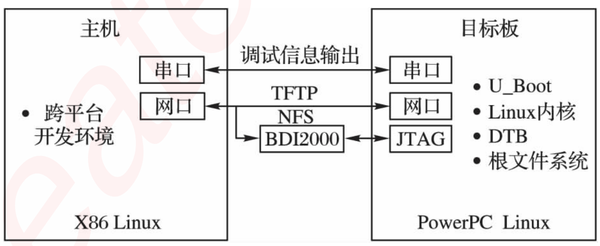

图 4 主机和目标板的接口连接

本文选择 DENX 软件中心的 ELDK（Embedded Linux Development Kit）开发工具……，在 Linux 主机上配置并安装支持 MPC8247 的嵌入式交叉开发工具链。

1 解压 eldk-5.2：

```
$/install.sh                //默认安装到/opt 下，默认安装 powerpc
```

2 在其提供的 environment-setup-powerpc-linux 文件中添加：

```
$export ARCH = powerpc
$export CROSS_COMPILE = powerpc-linux-
```

3 编写 hello.c，测试交叉编译，$powerpc-linux-gcc -o hello hello.c。

3.2 Bootloader 实现

Bootloader（引导加载程序）主要负责加载内核，尽管它在系统启动期间执行的时间非常短，不过它却是重要的系统组件。嵌入式 Linux 系统具有稍微独特的 Bootloader 需求，不仅必须将内核映像加载系统存储器，它还必须设定系统存储控制器，初始化处理器高速缓存区，启用各种硬件设备，直接实现网络引导基础设备的支持，以及从事各式各样的其他活动[10]。本文使用通用引导加载程序（Universal Bootloader，U-Boot）。U-Boot 的功能是在操作系统内核运行之前，初始化硬件设备，建立内存空间的映射图，为最终调用内核准备好正确的软硬件环境。

移植 U-Boot 的步骤分为：添加修改与 MPC8247 目标板相关的代码和配置选项，配置编译，烧写 U-Boot 至目标板。具体步骤如下：

1 在顶层 Makefile 中为开发板添加新的配置选项：

```
ICE8247_config: uconfig
@ $(MKCONFIG) $(@:_config=) powerpc mpc8260 mpc8247
```

2 在 u-boot-2012.04 开发包目录 board 下创建新的目标板文件夹 mpc8247，添加和编写 mpc8247.c，flash.c，Makefile，config.mk，u-boot.lds；

3 为开发板添加新的配置文件，include/configs/mpc8247.h，根据硬件配置 localbus，Flash，MAC，PHY 等参数的设置；

4 交叉编译 U-Boot，生成 u-boot.bin：

```
#make ARCH = powerpc
  CROSS_COMPILE = powerpc-linux- MPC8247_config
#make ARCH = powerpc
  CROSS_COMPILE = powerpc-linux-
```

5 将 u-boot.bin 放到/tftpboot 目录下，使用 BDI2000 烧写 u-boot.bin 至目标板；

6 调试 U-Boot 源代码，直至其在开发板上正常启动。

3.3 Linux 内核移植

本文使用的 Linux 内核版本为 3.4.4，可从 www.kernel.org 网站上下载源码。由于 Linux-3.4.4 支持 PowerPC 架构的多个处理器，所以移植的主要任务是硬件平台裁剪、编译和 DTS 的编译与移植。内核编译的主要步骤如下：

1 设置交叉编译环境变量：

```
ARCH = powerpc
CROSS_COMPILE = powerpc-linux-
```

2 本实验的开发板 MPC8247 与 Linux-3.4.4 内核自带的 mgcoge 开发板结构定义相似，但是需要修改 arch/powerpc/configs/mgcoge_defconfig，添加下列语句：

```
CONFIG_RD_GZIP = y                  //支持 gzip 解压
CONFIG_BLK_DEV_INITRD = y           //支持 initrd
```

3 编译内核：

```
#make ARCH = powerpc
  CROSS_COMPILE = powerpc-linux- mpc8247_defconfig
```

CROSS_COMPILE = powerpc-linux- uImage

3.4 DTS 文件编译和移植

设备树编译器 DTC 通过编译 DTS 文件生成设备树对象 DTB，其使用方法如下：

```
#dtc -O dtb -o mpc8247.dtb -b 0 mpc8247.dts
```

内核在 2.6.25 版本之后包含了 dtc 编译器，所以可以直接在 Linux-3.4.4 根目录下，使用 make 命令编译 DTS 文件：

```
#make ARCH = powerpc CROSS_COMPILE =
  powerpc-linux- mpc8247.dtb
```

最后，将 uImage 和 mpc8247.dtb 分别加载到内存中，可以看到一个典型的引导过程中，DTB 是如何得到使用的。

3.5 根文件系统制作

在系统调试期间使用网络文件系统（Network File System，NFS）部署期间使用 Ramdisk 根文件系统[13]。Ramdisk 用内存空间模拟出硬盘分区，通常用于保存经压缩的磁盘文件系统（例如 ext2）。我们建立一个基于 Ext2 的 Ramdisk 映像以供目标板使用。Ramdisk 根文件系统的制作及移植步骤如下（这里省略具体执行命令）：

1 创建根文件系统目录。

2 创建各种必要的系统文件和设备文件。

3 建立启动相关的配置文件。

4 拷贝工具链。

5 编译 Busybox，安装系统软件和应用。

6 将制作的根文件系统转换成 Ext2 映像，使用 mkfs.ext2 命令：

```
#losetup /dev/loop2 ramdisk.img
#mkfs.ext2 /dev/loop2
```

7 使用 mkimage 制作 U-Boot 映像文件。

```
#gzip -v9 ramdisk.img
#mkimage -T ramdisk -C gzip -n
  'Ramdisk Image' -d ramdisk.img.gz uRamdisk
```

8 将制作的 Ramdisk 映像加载至内存。

3.6 系统部署

在前面几节创建的 u-boot.bin、uImage、dtb 和 Ramdisk，经过系统部署后能够在目标板上运行嵌入式 Linux 操作系统。基本步骤如下：通过 BDI2000 及 JTAG 接口将 u-boot.bin 烧写到 Flash 位置，通过 U-Boot 和 TFTP，将 uImage、dtb 和 Ramdisk 烧写到 Flash 相应位置。设置 U-Boot 的相应启动参数，即可自动引导启动嵌入式 Linux 系统。

4 结语

基于自己开发的 MPC8247 嵌入式开发板，成功地实现了 PowerPC Linux 系统的开发和部署，并在硬件平台上稳定运行。通过本文，可以理解 PowerPC 架构、MPC8247 开发板及其硬件组成，掌握嵌入式 Linux 系统开发的整个流程。尤其是在理解扁平设备树基础上编写了 DTS 文件，使得整个系统能够顺利运行，这对基于 PowerPC 架构的嵌入式 Linux 系统开发有很好的借鉴作用。

开发的整个 PowerPC 架构 MPC8247 开发板为嵌入式 Linux 的研究提供了很好基础，接下来可以进行嵌入式实时性研究、文件系统研究和深入扁平设备树研究。一个重要的延伸是将 MPC8247 开发成通用的通信设备，例如小型交换机和路由器等。

参考文献：

[1] FU Y, DENG C, LI C. Optimal design of video decoder based on PowerPC 405[C]// Proceedings of International Conference on Consumer Electronics, Communications and Networks. Piscataway: IEEE, 2012: 3353 - 3357.

[2] 刘常清，黄文君，詹源. 基于 PowerPC 的车载通信系统设计[J]. 计算机工程，2012，38(7)：207 - 209.

[3] LIKELY G, BOYER J. A symphony of flavours: using the device tree to describe embedded hardware[C]// Proceedings of Linux Symposium. Ottawa, Canada: Linux Symposium, 2008: 26 - 37.

[4] GIBSON D, HERRENSCHMIDT B. Device trees everywhere[EB/OL]. (2008 - 01 - 09)[2012 - 10 - 11]. http://ozlabs.org/~dgibson/dtc/dtc.git.

[5] HERRENSCHMIDT B. Booting the Linux/PPC kernel without open firmware[EB/OL]. (2005 - 05 - 01)[2012 - 10 - 11]. http://ozlabs.org/pipermail/linuxppc64-dev/2005-May/004073.html.

[6] DENX. Flattened device tree blob[EB/OL]. [2012 - 10 - 11]. http://www.denx.de/wiki/DULG/LinuxFDTBlob.

[7] 王长清，蔡炜，蔡惠智. 基于 PowerPC 双核处理器嵌入式 Linux 系统开发[J]. 微计算机信息，2010，26(20)：5 - 7.

[8] Semiconductor Corporation. MPC8272 PowerQUICC Ⅱ family reference manual[EB/OL]. [2012 - 10 - 11]. http://cache.freescale.com/files/32bit/doc/ref_manual/MPC8272RM.pdf.

[9] IEEE Std. 1275-1994, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices[S]. [S.l.]: IEEE, 1994.

[10] DENX. The DENX U-Boot and Linux Guide (DULG) for TQM8272 [EB/OL]. [2012 - 10 - 11]. http://www.denx.de/wiki/publish/DULG/DULG-tqm8272.pdf.

[11] HALLINAN C. 嵌入式 Linux 基础教程[M]. 2 版. 周鹏，译. 北京：人民邮电出版社，2012.

[12] YAGHMOUR K. 构建嵌入式 Linux 系统 M]. 2 版. 秦云川，译. 北京：中国电力出版社，2010.

(上接第 1472 页)

[5] 李恒，徐自励，金立杰. 数据关联方法在多点定位系统中的应用[J]. 中国测试，2012，38(3)：69 - 71.

[6] 徐自励. 三站定位模糊、GDOP 与站点布局关系分析[J]. 通信技术，2012，45(3)：99 - 101.

[7] LU Y, WU H G, XU Z L, *et al*. An improvement on distributed detection in clock synchronization for MLAT sensor network[C]// 2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent System. Washington, DC: IEEE Computer Society, 2012: 81 - 85

[8] 张爱清 叶新荣 胡海峰. 无线传感器网络质心定位新算法及性能分析[J]. 计算机应用，2012，32(9)：2429 - 2431.

[9] WU C-H, SU W-H, HO Y-W. A study on GPS GDOP approximation using support-vector machines [J]. IEEE Transactions on Instrumentation and Measurement, 2011, 60(1): 137 - 145.

[10] LU Y, LIU C Z, WANG Z N, *et al*. TOA association based on an improved fuzzy clustering method in MLAT for A-SMGCS [C]// IEEE Eighth International Conference on Fuzzy Systems and Knowledge Discovery. Washington, DC: IEEE Computer Society, 2011: 573 - 577.

[11] ROMERO L A, MASON J. Evaluation of direct and iterative methods for overdetermined systems of TOA geolocation equations [J]. IEEE Transactions on Aerospace and Electronic Systems, 2011, 47(2): 1213 - 1229.

[12] CHAN Y T, HO K C. A simple and efficient estimator for hyperbolic location[J]. IEEE Transactions on Signal Processing, 1994, 42(8): 1905 - 1915.

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

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

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

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

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

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

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