基于 VxBus 和 MPC8569E 千兆网驱动开发和实现

李　丹
（广东轻工职业技术学院 计算机系，广州　510300）

摘要：随着嵌入式操作系统 VxWorks 在国内外各种领域越来越广泛的应用，如何让 VxWorks 支持各种硬件平台成了亟待解决的问题；特别是随着大量网络应用的出现，嵌入式系统下的千兆网络驱动是研究开发的热点；文章介绍了设备驱动相关的一些基础知识，分析了 VxWorks 6.X 最新引进的 VxBus 驱动程序框架，并对 MUX 在 VxWorks END 网络驱动软件中的作用进行了详细的研究和论述；最后，阐述了如何在 VxBus 中实现 MPC8569E 的千兆以太网 MAC 控制器（UCC）的驱动，给出了 UCC 的 END 驱动程序开发流程，对所有基于 VxBus 的框架下的网络驱动软件开发具有参考意义。

关键词：VxWorks；VxBus；MPC8569E；千兆以太网；UCC；END

Development and Realization of Gigabit Network Driver Based on VxBus and MPC8569E

Li Dan
(Guangdong Industry Technical College, Guangzhou　510300, China)

Abstract: With more and more extensive application of VxWorks, a popular embedded operating system, in many areas, how to make VxWorks support hardware platforms is an urgent problem we must solve. Especially, the network technology has found extensive application in various fields and the development of gigabit network driver software becomes a hotspot. This paper first introduces basic knowledge related to device driver. Then the VxBus is analyzed, which is the new driver framework introduced from VxWorks 6.X; And also, This paper gives a detailed research and discussion of MUX's function in the END Network Driver in VxWorks environment. Finally, it describes how to implement the Gigabit Ethernet driver for MAC controller (UCC) of MPC8569E processor and lays out development process of END driver for UCC, which may have a reference value for other network device driver developers.

Key words: VxWorks; VxBus; MPC8569E; gigabit ethernet; UCC; END

0　引言

VxWorks 实时操作系统是由美国 WindRiver 公司推出的一种嵌入式实时操作系统。作为被业界公认的最优秀的实时操作系统之一，VxWorks 操作系统被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中。VxWorks 操作系统自从 6.X 版本以来采用全新的 VxBus 模式进行驱动程序的设计。MPC8569E 是飞思卡尔公司推出的最新一代 32 位嵌入式处理器，它最主要的特点

是集成了高性能的网络处理模块，不需要额外的网卡支持。本文的千兆以太网嵌入式实验平台选用了 MPC8569E 处理器和 VxWorks 嵌入式操作系统，研究 MPC8569E 网络处理模块在 VxBus 新驱动框架下的驱动实现对实验平台的成功搭建具有非常重要的意义。

收稿日期：2013-07-30；　**修回日期**：2013-09-29。

基金项目：国家自然科学基金(61272415,61133014)；广东省自然科学基金项目(S2011010002708)。

作者简介：李　丹(1979-)，女，吉林人，工学硕士，主要从事计算机网络，计算机体系结构方向的研究。

1　MPC8569E 网络模块的硬件结构

千兆以太网 MAC 控制器对应于 OSI 七层协议中数据链路层，主要负责控制与连接物理层的物理介质。PHY 设备对应于 OSI 七层协议中的物理层。千兆以太网的接口实质就是 MAC 通过 GMII 总线控制 PHY 的过程。GMII 总线包括数据接口和管理接口；数据接口使用的是 GMII（或者 RGMII）协议；而管理接口使用的是 MDC/MDIO 协议[1]。MPC8569E CPU 把 MAC 功能集成在 QUICC Engine 模块中，在 QUICC Engine 模块中有 8 个 UCC（Unified Communication Controller），这些 UCC 可以配配置成 10M，100M，1000M 以太网控制器（UEC），实现 MAC 的功能[2]。如图 1 所示。

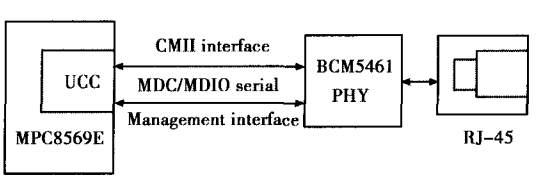

图 1　MAC 和 PHY 的结构图

因为 PHY 是物理层的实现，收发数据由硬件自动完成，软件对它的干预比较少，PHY 的驱动一般只做了初始配置

PHY 寄存器和监控其状态的工作，所以本论文重点对 MAC 控制器（UCC）的驱动做分析论述。

2 VxWorks 网络驱动程序分析。

2.1 VxBus 驱动程序框架简述

操作系统 VxWorks 从 6.x 版本开始为驱动开发提出了一个新的概念：VxBus 虚拟总线。VxBus 驱动模式提供了对设备驱动的管理，提高了 VxWorks 开发的可扩展性，可升级等性能。一方面，VxBus 将各个驱动有效组织，并将驱动对应到实际的设备上，形成实例；另一方面，VxBus 给上层软件提供了友好的接口来调用设备，使底层驱动对上层用户更加透明。另外，VxBus 将设备驱动模块化，这些模块能在 WindRiver 的开发环境 Workbench 里方便的组织和配置，这就意味着驱动可以加入一个工程中，也可以进行配置和删除，不需要对 BSP 进行修改，简化了驱动的开发流程[3]。

2.2 VxWorks 网络驱动程序结构

如图 2 所示，VxWorks 在网络协议栈和网络驱动程序之间增加了一层 MUX 层。MUX 层是 VxWorks 为方便在网络接口硬件上实现多种协议而增加的一层。它主要用于管理底层的多种硬件的设备驱动，向上层不同协议提供统一的接口，降低了上层协议和底层物理硬件的耦合，使得网络驱动和上层协议彼此保持独立，既方便在现有硬件基础上实现新的上层协议，也利于用新的硬件支持原有的上层协议[4]。

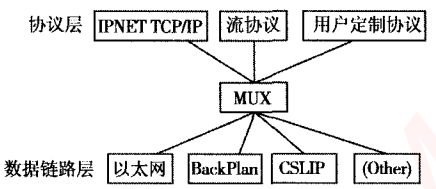

图 2 MUX 协议层关系图

2.3 MUX 与以太网驱动。

目前，VxWorks 支持两种以太网设备驱动程序：增强型的网络驱动程序（END）及 IPNET－Native 网络驱动程序；本文重点以 END 驱动程序为例做论述。MUX 与底层驱动的交互是通过提供一套可供底层调用的接口服务来实现的，实现 END 或者 IPNET－NATIVE 驱动必须遵循这套接口关系。如图 3 所示。

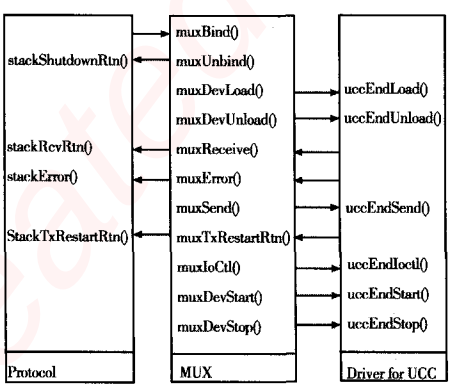

图 3 MUX 接口函数

上层网络协议可以通过 MUX 层调用函数 muxBind（）和指定的一个和多个 END 驱动程序进行绑定，绑定后上层网络协议可以通过 MUX 层将数据包传输给底层 END 驱动程序，相反，底层驱动程序也通过 MUX 层将数据包传输给上层网络协议，而数据传输的细节问题都交给 MUX 层的 API 函数来处理，这也使得 END 驱动程序和网络协议的通用性和可移植性得到了提高。如果要取消绑定可以调用函数 muxUnbind（）来实现。

2.4 缓冲池数据结构

网络设备驱动与上层协议之间需要进行数据交换，所以需要相应的内存缓冲，并且管理这些缓冲也需要相应的函数。VxWorks 提供了 netBufLib 函数库用于创建和管理网络设备用到的内存缓冲池，网络设备驱动可以直接使用也可以在此基础上设计自己特定的内存缓冲池。数据以簇（cluster）的形式保存，数据结构 mBlks（内存块）和 clBlks（簇块）形成的数据链结构则用于指定各个簇。

在 clBlk 之上是 mBlk 结构。该结构存储一个到 clBlk 的连接，也可以存储一个到另一个 mBlk 的连接。通过 mBlk 的连接，可以引用任意数量的数据，如图 4 所示[5]。

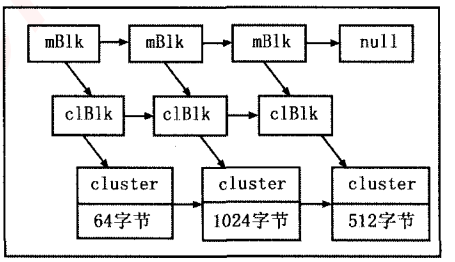

图 4 网络数据包结构

3 千兆以太网驱动在 VxBus 模式下的实现

3.1 UCC 驱动初始化

在 VxBus 架构中需要驱动程序匹配上具体的设备，从而形成一个实例。网络设备初始化过程中的首要任务就是为网络设备匹配上相应的网络驱动程序。所以首先需要注册 MAC 控制器（UCC）的驱动程序，uccRegister（）就是网络驱动程序的注册函数，它在 BSP 的初始化函数 hardWareInterFaceInit（）中被调用；注册的目的就是通知 VxBus 驱动程序框架该驱动可用，并提供关于该驱动的必要信息，包括设备 ID、挂载总线类型、驱动版本、驱动名以及初始化函数集。在 hwconf.c 文件中配置的网络设备与上述注册的网络驱动程序匹配，从而形成一个实例，挂在 PLB（Processor Local Bus）处理器局部总线上。

就像所有的 VxBus 实例一样，网络实例创建之后，需要执行网络设备的三阶段初始化，它分别由函数 devInstanceInit（），devInstanceInit2（）和 devInstanceConnect（）完成[6]。

1）在第一阶段，由于缺乏操作系统的支持，特别是因为系统内存池还没有初始化，许多设备的初始化都比较简单。对网络设备而言，唯一的任务就是设置网络设备的单元编号。

2）在第二阶段，这一阶段系统内存池已经初始化，驱动可以分配内存。所以，sysHwInit（）调用 devInstanceInit2（）

函数分配驱动所需的数据结构体（UCC _ DRV _ CTRL）和环形缓冲区内存（DMA ring buffer），这个环形缓冲区用于网络驱动与网络设备间发送和接收网络数据。

3）第三阶段，这一阶段主要的任务是把驱动程序与上层软件实体作连接；对以太网来说，就是把驱动装载进 MUX；但是因为这时 MUX 层还没有初始化，所以从 sysHwInit2（）调用的 devInstanceConnect（）没有做具体的任务，与 MUX 层的连接在系统启动的后期阶段再完成。

如上文所述，在 MUX 层初始化完成之后，要装载和启动网络设备，这需要从 usrNetEndLibInit（）函数中调用 muxDevConnect 方法来完成；muxDevConnect 是在 vxBus 框架中为网络设备注册的驱动方法，其实，它只是一个方法的标示，实际中具体要为 UCC 执行的函数为 uccMuxConnect（）；它的主要功能就是装载和启动网络设备，这是通过顺序调用 muxDevLoad（）和 muxDevStart（）来完成的。

3.2 UCC 的启动和关闭

UCC MAC 控制器利用 DMA 数据传输模式。DMA 数据传输模式有效地提高了系统的工作效率，它可以使 CPU 不必参与数据的传输直接由 DMA 控制器进行管理。为了利用 DMA，驱动软件必须首先完成必要的内存，寄存器等资源分配后，UCC MAC 控制器才能开始工作。在这其中，缓冲描述符（buffer descriptor）对 DMA 的工作起到非常重要作用。缓冲描述符必须在网口初始化的时候初始化（如 3.1 节所述），并将自己的地址赋给 MPC8569E 的寄存器。缓冲描述符可分为发送缓冲描述符（txBD）和接收缓冲描述符（rxBD）。

如 3.1 节所述，网络驱动由 muxDevLoad 加载并初始化完成后，但这时 UCC 还不能开始工作，这需要通过 MuxDevStart（）函数（实际执行的是 uccEndStart（））来启动 UCC MAC 控制器；它做的主要工作如下：

1）因为引入了网络作业线程（tNetTask），所以需要分配并初始化网络作业队列（network job queue），并指定各个作业队列的优先级。

2）分配接送和发送的 mBlk 结构，并建立起在 RxBDs 和 mBlk 结构之间的初始映射。

3）继续初始化 MPC8569E 所特有的诸如流量控制，TCP/IP 减负载等其他功能。

4）设置运作模式，初始化中断资源，并激活中断，最后，激活发送和接收功能[6]。

关闭 UCC MAC 控制器的工作与启动 UCC MAC 控制器的工作正好相反。但是从 MUX 调用中通过 uccEndStop（）函数关闭 UCC 之前，应该先完成网络任务的处理以避免系统崩溃。

3.3 UCC 数据的接受和发送

在 VxWorks 系统，发送数据的流程如下图 5 所示：

通过对图 5 的分析，主要包括以下几个处理：

1）用户调用 write（）函数，通过套接字访问网络。

2）网络协议层首先调用 netTupleGet（）从内存缓冲池中取得一个 mBlk 结构，再将待发送的数据通过 memcpy 拷贝到 mBlk 结构的 mBlkHdr. mData 中，mBlkHdr. mLen 存放待发送数据的字节数；并调用协议驱动程序的发送程序。

3）协议驱动程序调用 MUX 的 muxSend（）启动发送循环。

4）muxSend（）通过调用 send（）回调函数（对 UCC END 就是函数 uccEndSend（）），把缓冲区传递给 END 驱动程序；

5）数据发送程序把数据拷贝到设备缓冲区中，并把他放置到设备的发送队列中（FIFO 队列）。

6）当产生发送中断时，驱动程序的中断服务调度程序丢弃已发送的数据包，彻底清理发送队列。

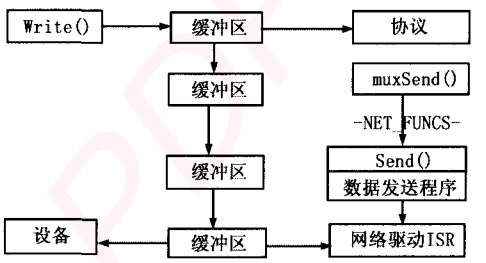

图 5 数据发送

在 VxWorks 网络系统中，接收数据的流程如图 6 所示。

通过对图 6 进行分析，它主要包含以下几个处理。

1）设备接收到数据包后利用 DMA 技术直接把数据存放到预先分配的缓冲区中（BD：Buffer Descriptor）。

2）当接收到中断时，驱动程序的中断服务程序把任务放置在网络线程作业线程的队列上，并调度网络作业线程（tNetTask）

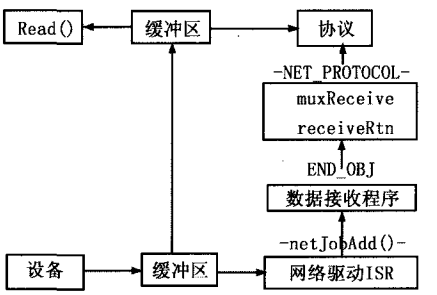

图 6 数据接收

3）网络作业线程的相应处理函数进行如下操作。

①clBlk 结构和簇连接；mBlk 和 clBlk 连接；最后构成缓冲区。

②通过调用 receiveRtn（）函数，把缓冲区传递给更高级别的协议（实际执行的是 muxReceive（）回调函数）。

③muxReceive（）调用协议的 stackRcvRtn（）函数，把成列的缓冲区传递给应用。用户使用 read（）函数，通过套接字访问网络中的成列缓冲区[7]。

3.4 中断处理和网络作业线程.

中断处理函数 uccEndInt（）处理设备中断。根据中断状态调用相应的中断处理程序，如接收终端程序、发送中断程序等。当网络接口产生中断时，系统调用中断服务程序。中断驱动程序只处理那些需要最小时间的工作，把其他耗时的任务排列到网络作业线程（如 3.2 节所述（tNetTask）的工作队列

中。VxWorks 通过 tNetTask 处理任务级的网络处理。tNetTask 调用队列中处理程序如下。

①包接收程序：把接收到的数据包上传到网络缓冲区的堆栈中，通过一个调用上传给 MUX。

②释放所有发送帧程序：程序调用 netClFree（）函数释放发送缓冲区中所有已经发送的数据帧。

3.5 千兆以太网的模式设置

为了网络接口的正常工作，MAC 一般需要被编程（Programmed）来匹配 PHY 的传输速度（10，100，or 1 000Mb/s）和双工模式（全双工或者半双工）；所以当 PHY 的状态有变化时，MAC 驱动应该得到通知，从而同步上 PHY 的最新状态（传输速度和模式）。为了做到这些，MAC 驱动必须注册一个专门负责处理 PHY 状态更新的方法｛miiLinkUpdate｝，以至于 link 状态监控线程（miiBusMonitor Task）可以调用这个方法通知 MAC 驱动 PHY 状态的变化[8]。

uccLinkUpdate（）方法就是 VxBus 中为 UCC 驱动实例注册的方法，他会根据 PHY 的状态写 MAC 的寄存器；具体在 MPC8569E 处理器中，MAC 寄存器 MACCFG2 的 IFM 域应该写成 byte Mode，FDX 域应该根据 PHY 的模式写成全双工或半双工模式。

4 软件调试与验证

4.1 调试步骤和测试环境

在调试过程中，首先建立可下载的包括了网络驱动程序的镜像文件。在目标板上的 BootFlash 里先烧录 bootloader。通过 bootloader 将内核影像下载到目标板上运行。然后，按着下列的调试步骤验证驱动程序是否加载成功：

1）检查 UCC MAC 控制器的相关寄存器是否设置正确的值。

2）用 vxBus 提供的相关命令（诸如 vxBusShow）检查 UCC 驱动是否注册成功。

3）检查 PHY 芯片模块信号灯的状态。

具体的测试环境如下：

1）主机环境。主机环境是 Linux 的集成交叉开发环境及相关的测试用软件。

2）目标机环境。目标机的硬件环境是 MPC8569E 开发板，软件环境是 VxWorks6.9 操作系统。

由主机和目标机共同组成了网络驱动程序的测试环境，两者通过网线相连，在各自系统启动完毕之后就可以针对以太网接口进行相关测试工作。

4.2 驱动程序测试

测试过程中在目标机上编写基于 TCP 协议的简单测试用程序，其功能是实现目标机和主机之间的组成 TCP 的 Client－Server 系统进行网络接口速度测试和可靠性测试，经测试，证明已经 PING 通实验板，可以进行数据传输，证明所开发的网络驱动程序实现了所有上文的设计功能，并在速度上得到了初步的验证（接近 1Gb/s），能够满足大批量数据传输工作。至此，初步调试测试过程结束，本方案取得了令人满意的结果。

5 结束语

本文分析了基于 VxBus 最新驱动框架下的 MPC8569E 硬件平台千兆以太网驱动的开发流程。分析过程中更多的侧重于基于 MUX 机制下的 MPC8569E UCC END 驱动程序在 VxWorks 下实现的共性，包括必须的步骤和配置，并在实践中成功实现了 VxWorks 下 MPC8569E MAC 控制器（UCC）的驱动；对其他未集成 MAC 控制器功能的嵌入式处理器，需要独立的外置网卡。由于网卡的主控制器的内部差异，其他网卡在初始化，接收数据和发送数据方面会有所不同，但是在其他方面可以参考本文所给 MPC8569E MAC 控制器（UCC）驱动程序在 VxBus 新驱动框架下的实现。

参考文献：

[1] Jason Liu. 千兆网口 FREESCALE ETSEC + Marvell 88E1111 uboot Linux 驱动分析 [EB/OL]. 2012－09－16 http://blog.csdn.net/jasonliu001/article/details/7983711.

[2] FREESCALE. MPC8569E PowerQUICC III Integrated Processor Reference Manual [EB/OL]. 2012－11－01. http://cache.freescale.com/files/32bit/doc/ref_manual/MPC8569ERMAD.pdf.

[3] 贺小琳，张善从. 基于 VxWorks 的 SD 卡驱动程序的设计与实现 [J]. 计算机工程和设计，2010，31（16）：3573－3574.

[4] 寇云林，陈怀民，等. VxWorks END 网络驱动软件的开发与实现 [J]. 计算机测量与控制，2009，17（1）：218－220.

[5] WindRiver Sytem. Vxbus device driver developers guide 6.9 [EB/OL] 2011－02－11. www.windriver.com/support.

[6] Yu－lin Zhang，Fang－yong Lu etc. Development and Realization of Gigabit Network Driver for VxWorks based on MPC8313E [A]. 2010 Internal Conference on Computer Application and System Modeling [C].

[7] 王景刚，邓如玉，等. 基于 VxWorks 的网卡驱动设计 [J]. 中国新技术新产品，2010，14：28－29.

[8] FREESCALE. QUICC Engine™ BLOCK Reference Manual with Protocol Interworking [EB/OL]. 2010－11－01. http://sutras.googlecode.com/svn－history/r5/trunk/docs/book.pdf

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

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

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）

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

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

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