基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计

马　晨，陈彦萍
（西安邮电大学 计算机学院，陕西 西安　710061）

摘要：针对现在 1394b 网络传输应用越来越广泛的特点，设计了一种基于 WDM 模型的 PCI Express 的 1394b 网络传输系统的驱动程序。重点讨论了基于自行研发的 PCI Express 总线 1394b 网络传输系统的 WDM 驱动程序开发过程，主要包括硬件访问、DMA 传输及中断通知等内容。经过反复测试验证，该传输系统工作稳定，并且此驱动程序可以方便地移植到其他传输系统中。

关键词：1394b；PCI Express 总线；WDM 驱动；DMA
中图分类号：TP316　　**文献标识码**：A　　**文章编号**：1000－8829(2013)03－0094－04

Driver Programming Design of 1394b Network Transmission System Based on PCI Express Bus

MA Chen, CHEN Yan-ping
(School of Computer Science, Xi' an University of Posts and Telecommunications, Xi' an 710061, China)

Abstract: Specific to current 1394b network transmission system used more widely, a WDM driver programming which for 1394b network transmission system based on PCI Express interface is designed. The development process of WDM driver based on PCI Express 1394b network transmission card is mainly discussed, which includes hardware access, DMA transfer and interrupt processing. After repeated testing, the system is stable and reliable, moreover, this driver program can be transplanted to other transmission systems.

Key words: 1394b; PCI Express bus; WDM driver; DMA

1394 网络是由 Apple 公司提出的一种简化计算机连接，并且为实时数据传输提供了一个高速接口，并命名为 FireWire(火线)。之后，随着 FireWire 的发展，在 1995 年，IEEE 组织正式认可了 FireWire 技术，并将其定位为 IEEE1394－1995 规范，开启了 FireWire 作为开放技术的发展的时代，并逐步完善，先后提出 1394a 以及 1394b 协议，并最终统一到 IEEE 1394－2008 协议规范。

1394 网络具有以下特点：

① 支持高速串行通信方式，具备简单易用的网络通信控制接口；

② 支持多余度通信接口，可以实现可靠的数据传输服务；

③ 支持多种通信速率模式，可以满足不同应用下的通信要求；

④ 采用串行通信接口，支持远距离通信传输；

⑤ 采用总线互联方式，最大支持 64 个通信节点，能够实现多设备互联通信；

⑥ 具备同行时间确定的特点，可以提供确定的点到点的通信服务，满足对通信可靠性有较高要求的系统通信；

⑦ 采用对等网络结构，不需要设置专门的服务器。

由于 1394 总线采取对等的总线结构，其一大优点是不用主机存在，在主机存在时，主机的身份也是作为普通的设备，这种平等性使一般设备之间的互连成为可能，这对于家电一体化之类的应用是大有好处的，所以未来对于 1394 设备的应用会越来越广泛。

PCI Express 总线[1]是用来连接如计算和通信平台应用中外围设备的第 3 代高性能 I/O 总线，能够应用于移动设备、台式电脑、工作站、服务器、嵌入式计算和通信平台等所有周边 I/O 设备的互连。第 1 代的

收稿日期：2012－05－10
作者简介：马晨(1985—)，男，山西祁县人，硕士研究生，主要研究方向为服务计算、系统内核；陈彦萍(1979—)，女，陕西铜川人，副教授，博士，主要研究方向为网络管理、服务计算。

PCI Express 1.0 接口传输速率达到 2.5 GHz，而 PCI Express 2.0 则在 1.0 版本基础上将接口速率提高到了 5 GHz，传输性能也提高了一倍。

1 1394b 通信网络方案设计

1.1 1394b 通信网络系统用途及组成基本框架

1394b 模块实现 CC/RN（CC 为控制计算机，RN 为远程节点）功能，构建 1394b 网络，为系统提供高速可靠的 1394b 通信卡，其基本构架如图 1 所示。

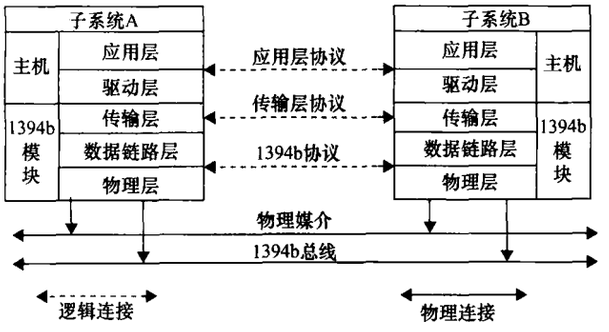

图 1 1394b 通信系统框架

其网络可分为 5 层：协议应用层、驱动层、传输层、数据链路层和物理层。其中物理层和数据链路层由模块的硬件实现，数据总线传输标准协议通过硬件逻辑实现，驱动软件和应用软件驻留在上位机（上位主机为 PC，运行 Windows XP 操作系统），应用软件通过调用 WDM 驱动实现子系统的功能需求。

1.2 1394b 通信传输子系统设计

本文所描述的驱动程序是基于自行研发的 1394b 通信传输子系统，该系统设计遵循数据总线传输标准协议，遵循系统提出的设计要求。完成符合协议所定义的通信调度和数据传输，同时提供网络管理硬件接口，能够实现网络通信的确定性调度和管理功能。

1394b 通信网络子系统的 FPGA 体系结构总体分为主机接口模块和数据总线传输标准协议处理模块，其示意图如图 2 所示。

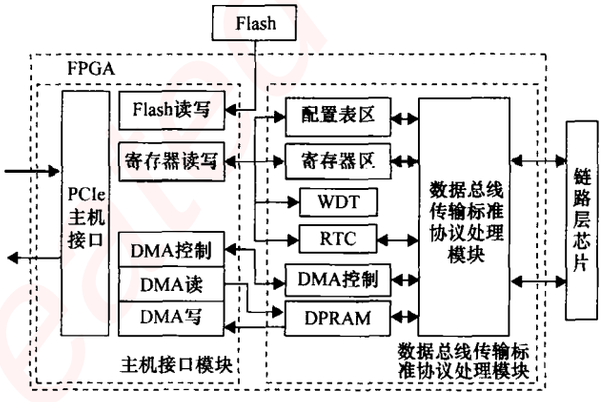

图 2 FPGA 体系结构示意图

逻辑设计功能需求特征如下：

① 支持数据总线传输标准协议定义的 STOF 包和异步流包传输；

② 支持数据总线传输标准协议定义的 CC/RN 功能，且 CC/RN 功能可配置；

③ 提供根节点切换功能硬件支持；

④ 主机接口支持 1 × 2.5 Gbit/s PCI-Express 接口；

⑤ 1394b 总线通信速率支持 S100B、S400B 模式；

⑥ 消息 Payload 长度可配置，S100B 模式下最大 512 B，S400 B 模式下最大 2 KB；

⑦ 硬件检查接收消息合法性。

2 基于 PCI Express 设计介绍

随着硬件的要求越来越高，PCI 总线已经不能满足越来越多的处理器和 I/O 设备所需要的带宽。为了提高总线性能，PCI SIG（PCI Special Interest Group）推出了取代 PCI 总线的新一代标准 PCI Express。高性能、高扩展性、高可靠性、更好的升级性以及更低廉的成本是 PCI Express 总线的设计理念。因此，1394b 网络通信子系统的 PCI Express 的主机接口设计如图 3 所示。

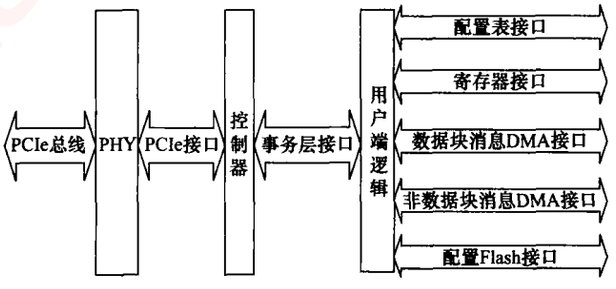

图 3 PCI Express 主机接口功能框图

本系统的 PCI Express 设计有如下特点：

① 兼容 PCI Express 1.1 规范；

② 最大支持 PCI Express 8Lanes，带宽可达双向 40 Gbit/s；

③ 支持消息中断方式（MSI）；

④ 管理控制，系统以及事务接口分离，可以较大提高设计灵活性；

⑤ 兼容传统的 PCI 软件构架，具有良好的软件兼容性；

⑥ 提供第三方 DMA 引擎设计参考，可实现 DirectSlave 的 DMA 读写访问操作；

⑦ 有可选的硬核用于设计，可有效降低器件逻辑规范；

⑧ 具有 32 bit、64 bit 的用户逻辑访问接口。

3 驱动程序设计

3.1 WDM 驱动模式介绍

WDM(Windows driver model)是 Microsoft 为 Windows 2000/XP 操作系统定做的新的驱动程序设计规范。在 WDM 模型中,完成一个设备操作,至少需要两个设备对象共同完成[2-3],即物理设备对象(PDO,physical device object)和功能设备对象(FDO,function device object)。在设备堆栈中 FDO 的上面和下面还会有一些过滤器设备对象(filter device object)。总线驱动程序检测到新的设备插入计算机后,操作系统的 PNP 管理器就会创建一个 PDO,PDO 创建完成后,PNP 管理器将装入并调用设备堆栈中的过滤驱动程序和功能驱动程序。

此外,WDM 还支持即插即用、电源管理、WMI(Windows management instrumentation)扩展等功能。

3.2 WDM 驱动程序实现

在 WDM 驱动模型中,驱动程序不仅要和硬件设备进行交互,而且还要实现与系统内核的交互。在交互过程中,很容易出错而使得整个系统崩溃,这就大大增加了驱动程序的开发难度,所以需要一个稳定的、可方便移植的驱动程序供开发人员参考。

WDM 驱动程序一般由几部分组成[4],包括初始化、创建和删除设备、处理 Win32 打开和关闭句柄要求、处理 Win32 I/O 请求、I/O 请求的撤销和超时处理、访问硬件资源、处理 Windows 的输入/输出请求、串行化设备访问、调用其他驱动程序、处理即插即用、电源管理、使用 WMI 和 NT 事件向系统报告等部分。

可以把一个完整的驱动程序看作是一个容器,它包含许多例程,当操作系统遇到一个 IRP 时,它就调用这个容器中的例程来执行该 IRP 的各种操作。表 1 表现了这个概念。有些例程,例如 DriverEntry 和 AddDevice,还有与几种 IRP 对应的派遣函数将出现在每一个这样的容器中。需要对 IRP 排队的驱动程序一般都有一个 StartIo 例程;执行 DMA 传输的驱动程序应有一个 AdapterControl 例程;大部分能生成硬件中断的设备,其驱动程序都有一个中断服务例程(ISR)和一个推迟过程调用(DPC)例程。驱动程序一般都有几个支持不同类型 IRP 的派遣函数,其中 3 个派遣函数是必须的。所以,WDM 驱动程序开发者的一个任务就是为这个容器选择所需要的例程[5]。

3.2.1 驱动程序初始化

DriverEntry 例程是每个驱动程序的入口函数,负责驱动程序的初始化,每个驱动程序都必须包含 DriverEntry 例程。它主要完成以下任务:设置各种分发例程的入口指针,使得 I/O 管理器知道调用哪些例程来处理用户打开、关闭、读写等请求。

extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDO,IN PUNICODE_STRING pRP)

表 1 WDM 驱动程序"容器"中的内容

基本驱动程序	I/O 控制程序	派遣服务程序
DriverEntry	StartIo	DispatchPnp
AddDevice	OnInterrupt	DispatchPower
	DpcForIsr	DispatchWmi
	AdapterControl	……
	……	

3.2.2 PCI Express 设备的硬件读写访问

在驱动程序初始化时,总线驱动会获取 PCI Express 设备的配置空间信息,并将配置空间中的基址存储器对应到计算机的内存空间。在基址寄存器中存放着所需映射的内存空间类型和大小,这样计算机就能分配一段相同类型和大小的内存空间,用于映射到逻辑中所设定的地址空间。BAR0 对应为片内寄存器,BAR1 对应为配置空间,BAR2 对应为片外 Flash。于是,当要访问上述空间时,只需访问映射好的计算机内存即可。驱动程序编写主要包括以下步骤:

(1) 声明相关资源的指针地址。

```
PVOID MemBar0;  //内存基地址 0
ULONG nMem0;  //基地址 BAR0 占用字节数
PVOID MemBar1;  //内存基地址 1
ULONG nMem1;  //基地址 BAR1 占用字节数
PVOID MemBar2;  //内存基地址 2
ULONG nMem2;  //基地址 BAR2 占用字节数
```

(2) 将 PCI Express 总线上的相关资源进行地址映射。下面例程仅介绍了 BAR1 的地址映射,其他 Bar2、Bar3 的映射以此类推。

```
PCM_PARTIAL_RESOURCE_DESCRIPTOR resource = &list->PartialDescriptors[0];
for (ULONG i = 0;i < nres; ++i, ++resource)
{ // 列举 PCIe 总线上每一个资源
  switch (resource->Type){// 切换不同的资源
  case CmResourceTypeMemory:
  if(resource->u.Memory.Length == 8*1024)
  { pdx->MemBar1 = (PVOID)MmMapIoSpace (resource->u.Memory.Start, resource->u.Memory.Length, MmNonCached);
    pdx->nMem1 = resource->u.Memory.Length;
  KdPrint (( "Bar1 Resource Translated Memory:(%x) Mapped:(%p) Length:(%d)\n",resource->u.Memory.Start.LowPart,pdx->MemBar1,resource->u.Memory.Length));}
  ……}
```

(3) 当需要访问映射地址时,可用 WDM 提供的函数。

```
READ_REGISTER_ULONG((PULONG)pdx->MemBar0 +
```

```
offset/sizeof(long));
    READ_REGISTER_USHORT((PUSHORT)pdx->MemBar0+offset/sizeof(short));
    READ_REGISTER_UCHAR((PUCHAR)pdx->MemBar0+offset);
    WRITE_REGISTER_ULONG((PULONG)pdx->MemBar0+offset/sizeof(long));
    WRITE_REGISTER_USHORT((PUSHORT)pdx->MemBar0+offset/sizeof(short));
    WRITE_REGISTER_UCHAR((PUCHAR)pdx->MemBar0+offset);
    //pdx->MemBar0 为对应的映射地址,offset 为对应的偏移
```

3.2.3 中断处理

在 1394b 网络传输过程中,会使用到很多的中断,例如当一个数据包发送完毕时或者接收完一个数据包时,都会产生中断,驱动程序收到中断后需要将此中断事件通知应用程序,以便应用程序对数据进行处理。本文设计的实现中断的方式包括以下几个步骤。

(1) 声明中断相关的变量。

```
ULONG          status;//中断状态
ULONG          param;//中断参数
PKINTERRUPT    InterruptObject;// 中断对象的地址
```

(2) 初始化中断事件和经行中断连接。

```
status = adEventInit(pdx);//对中断事件进行初始化
status = IoConnectInterrupt(&pdx->InterruptObject,(PKSERVICE_ROUTINE) OnInterrupt, (PVOID) pdx,NULL,vector,irql,irql,mode,TRUE,affinity,FALSE);//对中断进行挂接,相应的中断处理程序为 OnInterrupt 例程
```

(3) 响应中断,并在必要时进入中断延迟服务程序。中断延迟程序的目的是如果中断响应程序过大时,需要将一些代码放入中断延迟程序中处理,以保证中断响应程序的效率。

```
OnInterrupt(PKINTERRUPT InterruptObject,PDEVICE_EXTENSION pdx)//中断响应程序
{……
  IoRequestDpc(pdx->fdo,NULL,(PVOID)pdx);//挂接中断延迟程序
}
```

3.2.4 DMA 传输

由于在 FPGA 中已经实现了 DMA 控制器,所以在驱动程序无需控制 DMA 传输,只需要将 DMA 传输所需的内存开辟出来,然后赋值给 DMA 控制器即可。本设计中的 DMA 传输编程主要包括以下几个方面:

(1) 声明 DMA 传输的相关变量,并初始化 WDM 所提供的 DMA 传输适配函数。

```
typedef struct _DMA_STRUCT {
  struct _DMA_STRUCT * next;
  ULONG ulLen;
  PVOID pUserAddr;
  LARGE_INTEGER pPhyAddr;
  PVOID pDriverDmaAddr;
  PVOID pDriverMDL;
} DMA_STRUCT, * PDMA_STRUCT;
……
#if WINVER >= 0x0501 // windows XP 或者更新的操作系统
  DevDesc.Version = DEVICE_DESCRIPTION_VERSION2;
#else // windows 2000
  DevDesc.Version = DEVICE_DESCRIPTION_VERSION1;
#endif
  DevDesc.Master = TRUE;
  DevDesc.ScatterGather = FALSE;
  DevDesc.Dma32BitAddresses = TRUE;
  DevDesc.IgnoreCount = TRUE;
  DevDesc.InterfaceType = PCIBus;
  DevDesc.MaximumLength = 65535;
pdx->pDmaAdapter = IoGetDmaAdapter(pdx->pUnderlyingPDO,&DevDesc,&pdx->ulMapRegsGot);
```

(2) 申请用于 DMA 传输的公用缓冲区。

```
pUserStruct->pDriverDmaAddr = pdx->pDmaAdapter->DmaOperations->AllocateCommonBuffer(pdx->pDmaAdapter,pUserStruct->ulLen,&pUserStruct->pPhyAddr,FALSE);
```

(3) 获取用于 DMA 传输的用户地址,其部分代码:

```
pUserStruct->pDriverMDL = IoAllocateMdl(pUserStruct->pDriverDmaAddr,pUserStruct->ulLen,FALSE,FALSE,NULL);
pUserStruct->pDriverMDL = IoAllocateMdl(pUserStruct->pDriverDmaAddr,pUserStruct->ulLen,FALSE,FALSE,NULL);
MmBuildMdlForNonPagedPool((PMDL)pUserStruct->pDriverMDL);
pUserStruct->pUserAddr = MmMapLockedPagesSpecifyCache((PMDL)pUserStruct->pDriverMDL,UserMode,MmNonCached,NULL,FALSE,HighPagePriority);
```

(4) 将获取的用户地址传递给应用程序,并复制给硬件的 DMA 控制器,由 DMA 控制器控制 DMA 传输。传输完成后,将 DMA 所用资源释放。

4 结束语

驱动程序作为应用程序与硬件通信的桥梁,对系统性能的影响举足轻重。可靠的驱动程序是硬件稳定运行的保证。

本文探索了在 Windows 环境下 1394b 传输系统的 WDM 驱动开发,重点讨论了硬件访问、中断事件处理及 DMA 传输等大多数传输系统驱动程序都可能会涉及到的设计问题。经测试,其可以稳定、可靠地运行在 Windows 系统中。在此驱动程序的设计中,所有的参数都是由应用程序传递给驱动程序,使得该驱动程序具有很强的通用性,只需经过一些简单修改,便可用于其他基于 PCI 或基于 PCI Express 的数据传输标准接口的设备。

(下转第 101 页)

4 处理器关联技术在发动机检测系统中的应用

针对发动机检测系统数据速率快、实时性要求高的特点，特将多核处理器关联应用于该系统中。为了满足实时检测的要求，将检测系统任务进行了划分。除主进程外，其他实时性要求高的线程有数据接收线程和数据显示线程。

接收线程的功能是将 CAN 总线上的信号采集过来之后进行分析，提取有用的信息帧，根据所采集信号的帧 ID，将不同的数据存入所对应的数据队列中以便读取和显示。显示线程的功能是从不同的数据队列中读出相应的数据，进行数值的转换，实时显示在界面中并绘制各数值的曲线，处理器核心分配如图 4 所示。

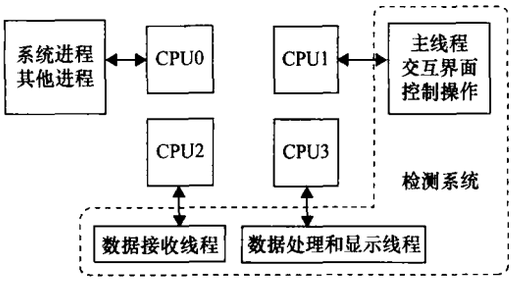

图 4 处理器核心分配

该检测系统运行时，首先进行处理器检测，进行处理器关联操作将接收线程和显示线程单独绑定至空闲的处理器，以保证接收数据的完整性和实时性。当系统结束时则恢复对计算机中所有进程的关联性操作，系统流程图如图 5 所示。

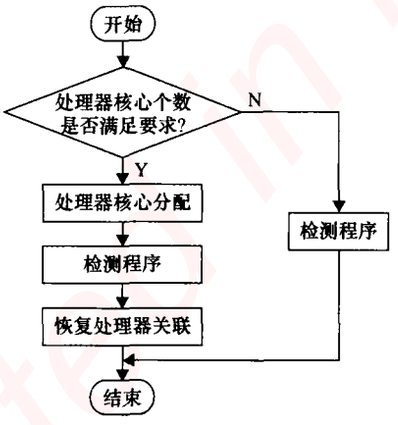

图 5 系统流程图

5 结束语

本文针对实时系统对实时性要求高的特点，利用处理器关联操作修改计算机中其他进程的处理器关联，以做到将实时性要求较高的进程绑定于某一处理器上运行。测试证明运用该方法后，系统性能得到了提升，并且不受其他进程和处理器状况的影响，即使系统处于繁忙阶段，独占处理器核的进程也可以运行得很顺畅。最终将该方法运用在一个数据采集系统中将重要进程绑定在单个处理器上，取得了良好的效果。

参考文献：

[1] Sebestyen G, Marfievici R. Real-time predictability on multi-processor and multi-core architectures[C]//IEEE 5th International Conferenceon on Intelligent Computer Communication and Processing. 2009:359 - 362.

[2] 多核系列教材编写组. 多核程序设计[M]. 北京:清华大学出版社,2007.

[3] Zhang C, Yuan X, Srinivasan A. Processor affinity and MPI performance on SMP-CMP clusters[C]//2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. 2010:1 - 8.

[4] 王晶,樊晓桠,张盛兵,等. 多核多线程结构线程调度策略研究[J]. 计算机科学,2007,34(9):256 - 258.

[5] Xie Y J, Loh G H. Thread-aware dynamic shared cache compression in multi-core processors[C]//2011 IEEE 29th International Conference on Computer Design. 2011.

[6] 杨磊,时磊,张铁军,等. 多核系统中共享 cache 的动态划分[J]. 微电子学与计算机,2009,26(5):56 - 59.

[7] Kazempour V, Fedorova A, Alagheband P. Performance implications of cache affinity on multicore prcoessors[C]// Proceedings of the 14th International Euro-Par Conference on Parallel Processing. 2008:151 - 161.

[8] 施惠丰,袁道华. 基于多核的多线程程序优化研究[J]. 计算机技术与发展,2010,20(6):70 - 73.

[9] 李骥,姜守达,邹昕光. Windows 操作系统多核 CPU 内核线程管理方法[J]. 自动化技术与应用,2010,29(1).

[10] Intel Corporation. Intel® V Tune™ Amplifier XE 2013[EB/OL]. http://software. intel. com/en-us/articles/intel-vtune-amplifier-xe/.

□

(上接第 97 页)

参考文献：

[1] Budruk R, Anderson D, Shanley T. PCI Express 系统体系结构标准教材[M]. 田玉梅,王崧,张波,译. 北京:电子工业出版社,2005.

[2] 武安河. Windows 2000/XP WDM 设备驱动程序开发[M]. 北京:电子工业出版社,2005.

[3] 张帆,史彩成,等. Windows 驱动开发技术详解[M]. 北京:电子工业出版社,2008.

[4] Cant C. Windows WDM 设备驱动程序开发指南[M]. 孙义,马莉波,国雪飞,等,译. 北京:机械工业出版社,2000.

[5] Oney W. Programming the Microsoft Windows Driver Model [M]. Microsoft Press,1999.

□

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

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
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发

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
10. 基于 COM Express 的回波预处理模块设计