文章编号:1001 - 9081(2009)02 - 0577 - 03

基于 PCI 总线的高速数据采集设备驱动开发

傅志中,鲜海滢,陈友林
(电子科技大学 通信与信息工程学院,成都 610054)
(fuzz@ uestc. edu. cn)

摘　要:基于 PCI 总线的高速数据采集设备驱动开发,其核心问题是驱动程序必须要有快速的实时响应能力和灵活的数据接收结构。分析了限制驱动程序高速数据传输的瓶颈,提出了分片、分帧的数据传输结构,给出了基于该结构的设备驱动设计思路。数据传输速率可达 85 MBps,与计算机的配置无关。

关键词:PCI 总线;高速数据采集;设备驱动;数据传输结构
中图分类号: TP311　　**文献标志码**:A

Device driver development of high speed data capture based on PCI bus

FU Zhi-zhong, XIAN Hai-ying, CHEN You-lin
(*School of Communication and Information Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*)

Abstract: The device driver must have quick response and flexible data structure for the device driver development of high speed data capture based on PCI bus. A multi-slot and multi-frame data structure was presented in this paper after the bottleneck of high speed data capture was analyzed, and a guideline to the design of the device driver was also presented. The data capture speed can be 85MB per second and it is independent of the computers' configuration.

Key words: PCI bus; high speed data capture; device driver; data tranform structure

0　引言

数字信号处理技术已经广泛地应用于各个领域,并且随着数字信号处理技术的发展和应用环境及需求的不断提高,需要处理的数据量呈爆炸性增长。在实际应用中,常常需要将多个通道的高速信号同时采集并实时传输给计算机处理,高效的设备驱动程序是实现数据高速传输的关键。已有作者在 Windows 系统,分别基于 DriverStudio、DDK 或 WinDriver 环境下研究了相应的设备驱动程序[1-5],但其数据传输速率不超过 25 MBps[3-4],这对于实时的高速数据或多路图像数据采集是远远不够的。在某个要求驱动实时传输数据速率不低于 80 MBps 的数字接收机项目中,本文作者根据项目需求,在基于 PLX 公司提供的驱动程序开发框架上,对其核心部分进行改进,实现了在 PCI 总线上的多卡高速数据采集设备驱动的设计,实际数据传输速率达到 85 MBps。

1　驱动传输速率分析

该项目要求实现数字接收机功能的同时必须具备实时数据采集、侦察、测向功能。系统功能框图 1 所示。

AD 转换器的速率为 40 MBps,精度为每样本 14 bit。接收机功能由 AD、DDC 和 DSP 完成对输入信号任意分析带宽的提取及解调,该路数据流如图中虚线所示,其数据速率不超过 4 MBps;实时数据采集、侦察、测向功能将 AD 的原始数据实时传输给计算机,由计算机完成相关功能,如图中实线所示,该路数据流达到 80 MBps,远远超过已有文献报道的传输能力。

2　驱动程序设计

2.1　数据传输结构

本项目中 PCI 接口使用的是 PCI9054,该芯片生产厂家 PLX 公司提供了一个基于 DDK 的驱动框架,该驱动的数据传输结构如图 2 所示。

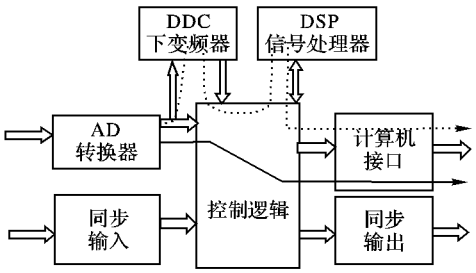

图 1　系统功能结构

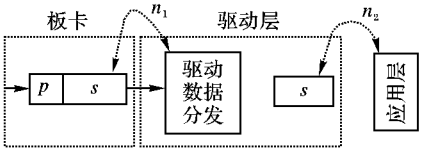

图 2　直通式数据传输结构

其中 s 是驱动层每次从板卡获取的数据量,也是每次驱动层传输给应用层的数据量,p 是缓冲区保护容量,n_1 为板卡中断驱动层频率,$n_2 = n_1$ 为应用层与驱动层的交互频率。从图 2 中可以看出,驱动层每次收到数据后,直接将数据传输给应用层。则该结构可以传输的数据量为:

$$C = n_2 \times s \tag{1}$$

该式表明,为了提高 C,必须增大 n_2 和 s。由该基本框架生成

收稿日期:2008 - 08 - 12;**修回日期**:2008 - 10 - 10。

作者简介:傅志中(1970 -),男,四川成都人,副教授,博士,主要研究方向:多媒体信息处理与通信、实时信号处理与实现;　鲜海滢(1974 -),男,四川成都人,博士研究生,主要研究方向:图像处理、模式识别、实时信号处理;　陈友林(1975 -),男,重庆人,工程师,主要研究方向:嵌入式系统。

的驱动程序,在板卡缓存为64 KB条件,40 KB数据满时触发数据传输,系统的传输速率不大于2 MBps,即 $C = 2$ MBps, $s = 40$ KB, 可得 $n_2 = 50$。测试结果表明:应用层与驱动层交互时间开销随着 n_2 的增大而增大,而与 s 和 n_1 关系不大。因此,降低应用层与驱动层的交互时间开销,是实现高速数据传输的关键。

如果增大板卡的缓存容量 s, 无疑是一个快速方便的解决方案。为了传输80 MBps的数据量,板卡缓存 s 满足:

$$s \geqslant \frac{C}{n} \times (1 + r) = 1.6 \times (1 + r) \tag{2}$$

r 是传输缓冲时间,通常取 $r = 1$, 即 $s = 3.2$ MB。该缓冲容量大大增加了板卡的体积和系统成本,其获得的好处是直接利用原始的驱动程序。为了在不增加板卡缓存容量的条件下实现高速的数据传输,本文采用如图3所示的数据传输结构。

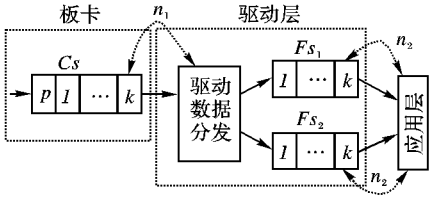

图3 分片、分帧的数据传输结构

该方案采用乒乓结构,其重点是将驱动层从板卡获取数据链路(频率为 n_1)与驱动层和应用层数据交互链路(频率为 $2 \times n_2$)分离。并设将板卡缓冲区容量 s 分割为 m 个片,每片的大小为 $s_i = s_0 (i = 1,2,\cdots,m)$, 剩余的记为 p, 驱动层中申请的缓冲区组织为乒乓结构,称为两帧, 分别标记为 Fs_1 和 Fs_2,每帧又分为 k 片,每片大小仍为 s_0,则有:

$$\begin{cases} s = p + m \times s_0 \\ Fs_1 = k \times s_0 \\ Fs_2 = k \times s_0 \end{cases} \tag{3}$$

数据传输速率为:

$$C = n_2 \times (Fs_1 + Fs_2) = 2ks_0n_2 \tag{4}$$

n_2 受限于计算机驱动层和应用层数据交互链路的响应速率, s_0 受限驱动程序响应硬件速度及板卡的缓冲容量, Fs_1 与计算机的内存容量有关,对于一般计算机来说,驱动程序申请几兆字节的内存是不受限的。因此,只要 n_2 和 s_0 满足系统硬件环境条件,数据传输速率 C 与 k 成线性关系,增加 k 就能提高系统的数据传输速率。

2.2 驱动软件设计

PLX公司提供了一个基于DDK的驱动框架,该框架实现了驱动程序的基本功能,其数据传输结构采用图2的直通式数据传输结构。该传输结构数据将板卡缓存与驱动层缓存、驱动层缓存与应用层之间的数据传输两个环节直接关联,导致驱动程序数据传输性能下降。图3将两个传输环节分开,数据传输速率 C 不受板卡缓存容量的影响,而与 k 成线性关系,从结构上提高了数据传输性能。为了实现图3的分片、分帧的数据传输结构,必须对原始的驱动程序进行相应的改动。

改进包括了以下几个重要的部分:缓存空间的申请与分配、驱动层数据分发、应用层的数据获取等。

2.2.1 缓存空间的申请与分配

该部分完成对驱动数据传输结构和控制变量的初始化工作。DriverBufferInit( ) 和 DriverBufferMalloc( ) 函数实现缓冲区的申请任务,应用层调用函数 PlxPciCommonBuffer-Get( ) 获取驱动分配的公共缓冲区,返回的缓冲区信息有:缓冲区物理地址、虚地址及缓冲区大小,单位为字节,之后根据板卡数量分配缓存容量,从而控制与使用该缓冲区。

缓冲区获取代码为:

```
// PCI Memory Structure
typedef struct _PCI_MEMORY
{
  U32 UserAddr;                    //虚地址
  U32 PhysicalAddr;                //物理地址
  U32 Size;                        //大小
}
PCI_MEMORY;
PCI_MEMORY PciMemory;
PlxPciCommonBufferGet( MyDeviceHandle, &PciMemory);
                                   //获取缓冲区属性
```

设有CARDNUM个PCI卡,不妨一般化,每卡平均分配该缓冲区。

缓冲区分配策略代码为:

```
// Each PCI Card Memory Structure
typedef struct _EACH_PCI_MEMORY
{
  U32 UserAddrPing;                //虚地址
  U32 UserAddrPang;                //虚地址
  U32 PhysicalAddrPing;            //物理地址
  U32 PhysicalAddrPang;            //物理地址
  U32 Size;                        //大小
}
EACH_PCI_MEMORY;
//定义每卡的缓冲区变量
EACH_PCI_MEMORY EachPciMemory[ CARDNUM];
```

则每卡的缓冲区大小为:

```
int BufLen = (PciMemory.Size / CARDNUM / 2) &0x0FFFFFFFC;
```

多卡CARDNUM的缓冲区属性初始化:

```
for( int i =0; i < CARDNUM; i + +){
EachPciMemory[ i]. UserAddrPing = PciMemory. UserAddr + 2 * i *
BufLen;
EachPciMemory[ i]. UserAddrPang = PciMemory. UserAddr + (2 *
i +1) * BufLen;
EachPciMemory[ i]. PhysicalAddrPing = PciMemory. PhysicalAddr +
2 * i * BufLen;
EachPciMemory[ i]. PhysicalAddrPang = PciMemory. PhysicalAddr +
(2 * i +1) * BufLen;
EachPciMemory[ i]. Size = BufLen;
}
```

其中缓冲区物理地址将通过 Dispatch_Io- Control( ) 传输给驱动,用于DMA传输,缓冲区虚地址用于应用层程序读取驱动接收的外部数据。

2.2.2 驱动层数据分发

实时响应外部中断,实现数据的收集与分发。中断控制、延迟处理是该部分的核心。中断处理函数框图如图4所示。

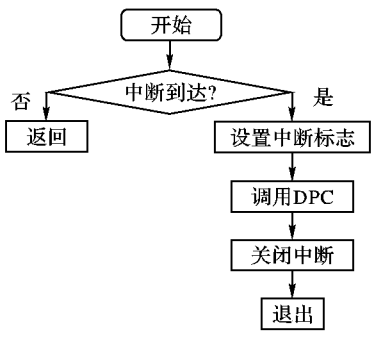

图4 中断服务程序流程

操作系统的中断由函数 OnInterrupt( ) 响应。为实现中断

响应快速返回,增设函数 void ProcessLocalIsr(PVOID pContext)。该函数在 DPC 例程中执行,处理外部中断,更新当前传输窗口帧信息,初始化 DMA0 通道,启动 DMA0 通道,根据当前传输帧数,判断是否已经有一个缓冲区已经传输完成,发送事件给应用层。最后,将外部中断打开。其实现框图如图 5 所示。

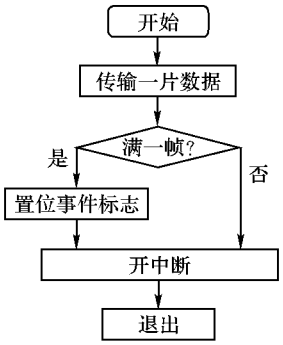

图 5　DPC 服务程序流程

2.2.3　*应用层的数据获取*

在应用层创建一个等待驱动事件的线程。当驱动将一帧数据准备后,驱动发送一个事件。线程响应后,获取当前数据帧号,读取对应的数据帧 $Fs_i(i=1,2)$。

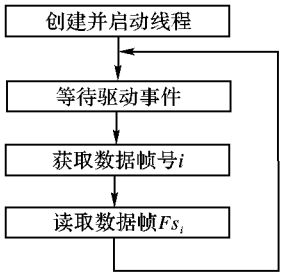

图 6　应用层数据获取流程

2.3　驱动程序性能测试

运行 CreateDriver.bat,将驱动源码生成最终的驱动程序 PCI9054.SYS 执行文件。Windows 2000 系统具有自动管理与加载多卡驱动能力,系统能根据板卡数量,自动加载驱动程序的多个副本。只需要在更改驱动程序的过程中,将改进的所有参变量与设备关联,就能实现驱动的多卡应用,不需要编写如文献[2]描述的特定驱动管理程序。该驱动程序的传输速率和存储速率在 Win2000/XP 操作系统下的测试结果如表 1 所示。

表 1　传输速率和存储速率测试结果　　MBps

硬件环境	传输速率	存储速率
研华工控机: 主频 2.8GHz,512MB 内存,SATA 硬盘	≤85	≤40
商用机: 主频 600MHz,128MB 内存,IDE 硬盘	≤85	≤10

测试结果表明,在 Windows 2000/XP 系统和两个不同的硬件环境,该驱动的数据传输速率超过了项目设计需求 80 MBps。该传输速率达到了 PCI 总线理论速率(132 MBps)的 64.4%,相比文献[3]的 18.3% 和文献[4]的 18.2%,极大地提高了驱动数据传输速率,分别提高了 46.1% 和 46.2%。

3　结语

本文在分析了现有基于 PCI 总线驱动程序的数据传输结构后,提出了一种基于分片、分帧的高效数据传输结构。基于该结构实现的驱动程序的数据传输速率可达 85 MBps,并与计算机配置无关,满足了本系统的要求。该驱动程序的实现,为充分利用计算机的高速计算能力提供了有力的数据获取能力保障。

参考文献:

[1] 季晓君, 王海. 同步多串口卡驱动程序设计[J]. 军事通信技术, 2003, 24(1): 44 - 47.

[2] 张耀中, 王安丽, 徐姣杰. Windows NT 下视频传输卡的多卡驱动程序开发方法[J]. 系统工程与电子技术, 2004, 26 (1): 91 - 94.

[3] 司玉美, 申会民, 耿爱辉, 等. 基于 PCI 总线数据通信卡 WDM 驱动程序设计[J]. 计算机测量与控制, 2006, 14(2): 259 - 261.

[4] 徐涛, 黄鲁, 王荣生. 一种多 PCI 卡组成的高速同步图像获取系统[J]. 计算机工程与应用, 2004, (2): 124 - 125.

[5] 贺骊. Windows NT 下多设备共用中断驱动程序设计[J]. 微电子学与计算机, 2002, (3): 59 - 63.

[6] PLX Technology. PLX SDK Programmer Reference Manual[EB/OL]. [2008 - 06 - 13]. http://www.eetop.com.cn/bbs/thread-44042-1-1.html.

(上接第 576 页)

参考文献:

[1] 葛哲学, 杨拥民, 胡政, 等. 基于知识的直升机自动驾驶仪故障融合诊断策略[J]. 中国机械工程, 2006, 17(4): 338 - 342.

[2] 胡志刚, 马好, 廖麟. 基于模糊神经 Petri 网的故障诊断模型[J]. 小型微型计算机系统, 2005, 26(11): 1978 - 1982.

[3] 李凯, 赵克, 许威. 概率粗糙集模型的机械故障诊断研究[J]. 机械科学与技术, 2005, , 24(12): 1437 - 1440.

[4] 谢强, 郑洪源, 周良, 等. 基于故障模式的外部质量控制及分析[J]. 中国机械工程, 2002, 13(18): 1570 - 1575.

[5] 周良, 谢强, 郑洪源等. 质量管理系统中故障模式的设计与实现[J]. 小型微型计算机系统, 2001, 22(12): 1494 - 1496.

[6] USCHOLD M, GRUNINGER M. Ontologies: Principles, methods, and applications[J]. Knowledge Engineering Review, 1996, 11 (2): 93 - 155.

[7] LEE S M, HONG S. An enterprise-wide knowledge management system infrastructure[J]. Industrial Management & Data Systems, 2002, 102(1): 17 - 25.

[8] EHRIG M, SURE Y. Ontology mapping an integrated approach [C]// Proceedings of ESWS. Heidelberg: Springer, 2004: 76 - 91.

[9] NATALYA F N, MUSEN M A. Anchor-prompt: Using non-local context for semantic matching[EB/OL]. [2008 - 06 - 23]. http://www.dit.unitn.it/~accord/RelatedWork/Matching/noy.pdf.

[10] GIUNCHIGLIA F, SHVAIKO P. Semantic matching [J]. The Knowledge Engineering Review Journal, 2004, 18(3): 265 - 280.

[11] GIUNCHIGLIA F, SHVAIKO P, YATSKEVICH M. S-match: an algorithm and an implementation of semantic matching[C]// Proceedings of the 1st European Semantic Web Symposium. Schloss Dagstuhl: IBFI, 2004: 61 - 75.

[12] MAEDCHE A, MOTIK B, SILVA N, *et al*. MAFRA-A mapping FRAmework for distributed ontologies[C]// 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW). London: Springer-Verlag, 2002: 235 - 250.