第36卷 第20期 计 算 机 工 程 2010年10月
Vol.36 No.20 Computer Engineering October 2010
·工程应用技术与实现· 文章编号：1000—3428(2010)20—0211—03 文献标识码：A 中图分类号：TP316

RTX环境下PCI设备实时驱动的开发

黄 键，庞亚华，薛顺虎
(中航工业西安飞行自动控制研究所，西安 710065)

摘 要：掌握实时驱动的设计方法，能够从一定程度上解决由于设备实时驱动问题给实时应用在产品选型和功能实现等方面带来的不便。为此，介绍RTX实时环境下PCI总线设备的实时驱动的开发流程，以某PCI设备为例，给出实时驱动设计中包括设备定位、总线信息获取、地址映射、初始化配置以及伺服中断等功能的具体实现，将该驱动应用到某RTX实时系统中，经过实验验证取得了较好的应用效果。
关键词：RTX实时系统；实时驱动；PCI总线；地址映射

Development of Real-time Drive of PCI Device Under RTX Environment

HUANG Jian, PANG Ya-hua, XUE Shun-hu
(AVIC Xi'an Flight Automatic Control Research Institute, Xi'an 710065, China)

【Abstract】The inconvenience of product selection and function realization can be solved at a certain extern when the method of designing real-time drive is mastered. It introduces the developing flow of PCI bus drive in RTX real-time environment. The following functions of real-time drive design are explained including finding device, getting bus message, mapping address, initial configuration, serving interrupt and so on. The real-time drive application in a certain RTX real-time system proves its advantage.
【Key words】RTX real-time system; real-time drive; PCI bus; address mapping

1 概述

有些工业控制、航空航天等领域对实时性都有一定的要求，系统的事件响应如果不确定或超时，就可能会导致系统出错，甚至崩溃。RTX作为Windows的实时扩展子系统，不仅解决了Windows较差的实时性问题，而且也继承了Windows的所有优势，能为用户提供良好的实时控制性能、高效的扩展性和可靠的稳定性。作为实时平台，RTX只提供了部分外围硬件的驱动支持，如部分网卡、USB、串并口等，而大部分的第三方硬件也只是提供了Windows、Linux或VxWorks等操作系统的支持，对RTX的支持并不理想。因此，在构建基于RTX的实时应用系统时，需要考虑数据采集或总线通信等外围设备的RTX驱动问题。

对于硬件驱动支持，一般有2种选择：(1)从RTX代理商购买或委托其开发，这种选择成本较高。(2)选择开放性较好的硬件厂商进行自主开发，这种选择必须要求硬件厂商提供完整的硬件资料和技术方案书。

本文以某PCI数据采集设备为例，对其RTX实时环境下的驱动程序进行了设计开发，完成了对这些设备的初始化配置、启动以及中断挂接等操作。测试表明，驱动程序工作稳定，能够满足实时性要求，输出结果正确。将该设备驱动应用在某课题的RTX实时系统中，经过实验验证，达到了预期效果。

2 RTX简介[1-2]

RTX是美国Ardence公司开发的基于Windows操作系统的硬实时解决方案，可以为用户提供良好的实时控制性能、高效的扩展性以及稳定性。严格意义上来说，RTX并不是一个独立的操作系统，它是Windows上的一个实时扩展子系统。RTX作为一个完全的Windows扩展系统，并不对原有Windows系统架构作任何修改与封装，而是对Windows硬件抽象层的扩展。精确的时间执行机制对系统相当重要，RTX提供的定时器周期最低可以做到100 μs。通过应用高速的IPC信息和同步机制，RTX可以实现与Windows之间的数据通信以及进程间同步。RTX提供了对I/O以及内存的精确控制，为实时任务的执行提供了100%的可靠性。

RTX提供了对微软Visual Studio系列开发平台的全面支持，支持Win32 API标准函数的调用。开发人员可以最大程度地减少开发周期和成本，同时支持对IRQ、I/O、内存等硬件资源的直接访问，使得开发人员能够较为自主地进行应用系统的定制，便于系统的移植和扩展。

3 实时驱动设计[3-4]

所谓驱动程序，其实就是直接控制设备进行工作的那部分软件程序。如果要使用PCI总线设备上的某个功能，就需要CPU能通过某段范围的地址访问或内存访问的方式与该功能交互，当然也需要获知该功能的一些配置参数并可以修改。图1是PCI总线的配置空间结构示意图。对CPU而言，一般是通过操作PCI桥接芯片里面的定位寄存器和数据偏移寄存器，由桥接芯片通过PCI总线操作来完成配置。数据偏移寄存器则用来实现与位于设备上的内部配置寄存器进行数据交互，也就是图1中的Base Address Register。对于PCI总线设备来说，共有6个Base Address Register。

作者简介：黄 键(1980－)，男，工程师、硕士，主研方向：计算机仿真技术，实时操作系统应用；庞亚华，工程师、硕士；薛顺虎，研究员
收稿日期：2010-05-10 **E-mail：**beyond_huangjian@163.com

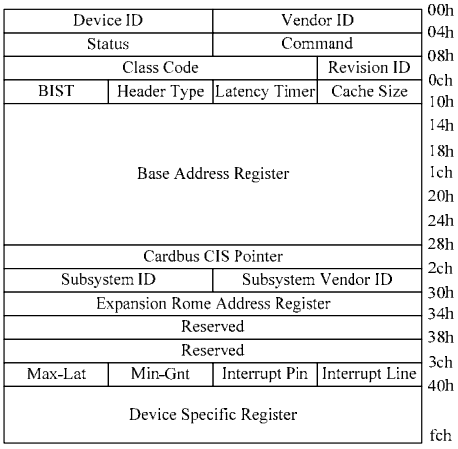

图 1　PCI 总线空间结构

一般情况下，PCI 定位寄存器配置由 BIOS 自动完成初始化，如分配总线号、中断向量、地址空间等，那么驱动程序只需要对数据偏移寄存器，也就是与之交互的设备内部寄存器进行配置。要访问内部寄存器，就必须获得 PCI 设备在 BIOS 上的映射基地址(存储于 Base Address Register 中)，然后根据设备的技术方案书里的寄存器偏移量和格式对寄存器进行访问。

3.1　开发流程

开发设备驱动程序首先要遍历目标平台上所有的 PCI 设备，根据 VendorID 和 DeviceID 确定需要驱动的具体设备，并得到该设备所在的 PCI 总线号、中断向量以及映射基地址等信息。为了能够访问设备内部寄存器，必须进行逻辑地址映射或端口使能。所有的信息获取到以后，就可以进行初始化配置、设备启动、停止等操作。当然，这些操作需要设备制造商提供详细的硬件内部寄存器资料。接着驱动程序根据软件的工作方式进入不同的处理流程：查询和中断，具体流程如图 2 所示。

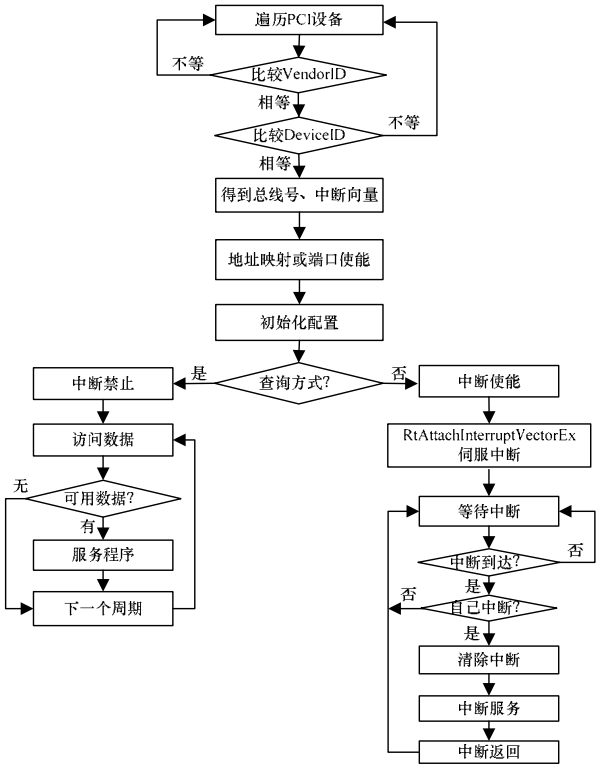

图 2　开发流程

3.2　设备定位

RTX 提供了 RtGetBusDataByOffset 接口函数，对所有 PCI 总线上的设备进行枚举，调用后返回所有 PCI 总线设备的信息，结构如图 1。通过比较 DeviceID 和 VendorID，就可以获取某 PCI 设备的总线号、中断向量、映射基地址以及空间大小等信息，部分程序流程如下：

```
for(bus)                    //总线循环
for(deviceNumber)     //设备循环
for(functionNumber)   //功能循环
{
 RtGetBusDataByOffset //返回 PciData 总线信息结构，指向图 1
 if(PciData->VendorID&& PciData->DeviceID) //找到设备
 {
  bus, InterruptLine…   //得到设备总线号和中断向量等信息
  if(true)   RtMapMemory //内存映射，返回逻辑地址
  else       RtEnablePortIO //IO 映射，使能 I/O 空间
 }
 PciData->Command = PCI_ENABLE_IO_SPACE |   //IO 使能
 PCI_ENABLE_MEMORY_SPACE |   //内存使能
 PCI_ENABLE_BUS_MASTER   //总线控制使能
 RtSetBusDataByOffset   //设置控制参数，使能控制寄存器访问
}
```

地址映射分为内存映射和 I/O 映射 2 种。如果 PCI 设备是内存映射，由于 RTX 不支持物理地址的直接访问，因此必须使用接口函数 RtMapMemory 对内存映射的地址进行逻辑映射，得到逻辑内存的基地址。如果 PCI 设备是 I/O 映射，则要使用 RtEnablePortIO 使能 I/O 地址空间。同时，为了确保内存地址或 I/O 空间能够正常访问，调用 RtSetBusDataByOffset 函数设置控制参数 Command，使能控制寄存器访问。

3.3　初始化配置

在获得设备的地址映射信息之后，就可以根据硬件技术方案书进行初始化配置。以某 32 路 AD 模拟量输入设备为例介绍初始化配置过程。该设备内部有 8 位、16 位和 32 位 3 种长度的寄存器，这里定义了 BYTE、WORD 和 DWORD 3 种类型的指针 pointer8、pointer16、pointer32，首指针都指向 Base Address Register 映射得到的 6 个逻辑内存基地址的第 2 个。当然，不同设备的内部寄存器映射到 Base Address Register 的位置也不同，具体情况要参见相应的硬件技术方案书。通过这 3 种类型的指针，加上设备不同功能的寄存器偏移量，就可以分别指向不同的寄存器完成初始化配置和状态监控。在完成初始化配置之后，驱动程序就可以启动设备工作。初始化配置过程如下：

```
*(pointer8+命令寄存器偏移)=0x80; //复位操作
*(pointer8+命令寄存器偏移)=0x81; //清复位操作，正常工作
*(pointer32+采样通道设置寄存器偏移)=0xffffffff;
//配置采样通道
*(pointer32+配置寄存器偏移)=0x02; //配置增益、信号方式等
*(pointer16+触发深度寄存器偏移)=0x64; //配置触发深度
*(pointer8+采样周期寄存器偏移)=0x04; //配置采样周期
*(pointer8+命令寄存器偏移)=0x89(0x88); //中断使能(禁止)
*(pointer8+命令寄存器偏移)=0x82; //AD 采样功能复位
*(pointer8+命令寄存器偏移)=0x83; //AD 清复位
```

3.4　设备启动

驱动程序工作方式一般有 2 种：查询方式和中断方式。在查询方式下，驱动程序在每个应用周期都对设备数据寄存器的数据进行访问，如果数据可用，软件将进行相应的处理；

否则，进入下一个应用周期。在中断方式下，首先使能设备中断，然后根据获取到的设备总线号和中断向量，调用 RtAttachInterruptVectorEx 函数挂接该中断。格式如下：

```
RtAttachInterruptVectorEx (
    NULL,             // 安全属性
    0,                // 堆栈大小
    DeviceIST,        // 指向中断服务程序
    NULL,             // 传递参数
    IST_PRIORITY,     // 中断级别
    PCIBus,           // 接口类型
    bus,              // 总线号
    irqLevel,         // 中断向量
    irqVector,        // 中断向量
    bShared,          //允许共享中断
    LevelSensitive,   // 中断模式
    NULL)
```

参数 DeviceIST 指向中断服务程序；接口类型为 PCIBus；bus 是设备总线号；irqLevel 与 irqVector 为中断向量；bShared 参数必须设为 TRUE，允许共享中断。

函数调用成功后，DeviceIST 中断服务程序将被挂起，等待中断。如果该中断到达，则进入 DeviceIST 中断服务程序处理中断事件。由于 PCI 总线是共享中断的，因此中断服务程序首先要读取该设备的中断寄存器来判断是否是自己发生了中断，是则继续处理，否则立即返回。

以该 32 路 AD 为例，查询方式下一般采用软件触发，启动采样，向启动寄存器写入任意值即可启动一次 AD 扫描，全部通道扫描完成以后就可以依次从 FIFO 中读取 AD 采样值，过程如下：

```
*(pointer16+启动寄存器偏移)=0xffff; //启动一次 AD 扫描
while((*(pointer16+状态寄存器偏移)&0x10)!=0x10) ;//等待完成
for(通道数)
AD_data=*(pointer16+FIFO 寄存器偏移); //读出所有 AD 数据
```

在中断方式下，一般采用内触发或外触发来启动采样，由外来事件控制 AD 工作。如果中断到达则启动一次 AD 扫描，同时进入中断服务程序处理 AD 采样数据。DeviceIST 中断服务程序如下：

```
if(*(pointer16+状态寄存器偏移)&0x20)!=0x20)
//是该设备的中断
{
  *(pointer8+命令寄存器偏移)=0x86; //清除中断
  for(通道数)
  AD_data=*(pointer16+FIFO 寄存器偏移); //读出所有 AD 数据
  }
else    return; //是其他设备的共享中断
```

3.5 设备关闭

在退出设备驱动时，首先要关闭中断，然后调用 RtReleaseInterruptVector 释放中断挂接、RtUnmapMemory 释放内存、RtDisablePortIo 关闭 I/O 空间。

4 应用

在驱动程序调试完成以后就可以发布。在实际应用中，驱动程序发布一般有 2 种方法：(1)将驱动源代码添加到 RTX 应用系统中，直接参与工程编译和链接。(2)将驱动代码打包生成 RTdll 链接库，通过 LoadLibrary 函数加载应用。

笔者使用这种实时驱动的设计方法，编写了一系列 PCI 总线数据采集和通信设备的 RTX 驱动，并应用到某型号导弹的半物理仿真实验中。到目前为止，已经成功地进行了若干轮次的控制律验证实验，取得了很好的效果。

5 结束语

由于设备驱动程序的限制，要使得实时操作系统在工业领域得到深入的应用，就有必要掌握实时环境下设备驱动的开发方法，以取得在系统设计、产品选型等行为上的主动。

本文以某 PCI 数据采集设备为例，介绍了 PCI 设备的 RTX 驱动程序的开发流程，并给出了具体的实现方法。RTX 实时驱动程序开发的成功不仅可实现硬件设备的功能，提供完善的接口函数，更重要的是它使得 RTX 实时应用中 I/O 设备有更多的选择，减少由于驱动带来的选型限制，而且有助于设计人员技术能力的提高。

参考文献

[1] Ardence 公司. RTX 技术白皮书[Z]. 北京航天捷越(美斯比)科技有限公司, 译. 2004.

[2] 黄 键, 薛顺虎, 宋 晓. RTX 平台下实时仿真系统的设计方法[J]. 计算机应用与软件, 2009, 26(4): 167-169.

[3] 周 静. VxWorks 下基于 PCI 总线的驱动程序开发[J]. 电子元器件应用, 2009, 11(7): 70-73.

[4] 宋有泉, 高小鹏, 龙 翔. 嵌入式 PCI 网卡驱动程序的设计与优化[J]. 计算机工程, 2007, 33(2): 264-266.

编辑 顾逸斐

~~~~~~

(上接第 205 页)

参考文献

[1] Coello C. Constraint Handling Using an Evolutionary Multiobjective Optimization Technique[J]. Civil Engineering and Environmental Systems, 2000, 17(4): 319-346.

[2] Deb K, Pratab A, Agrawal S, et al. A Fast and Elitist Nondominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA Ⅱ[J]. IEEE Trans. on Evolutionary Computation, 2002, 6(2): 182-197.

[3] Eberhart R, Kennedy J. A New Optimizer Using Particle Swarm Theory[C]//Proc. of the 6th Int'l Symposium on Micro Machine and Human Science. Piscataway, NJ, USA: IEEE Service Center, 1995: 39-43.

[4] Woldesenbet Y G, Yen G G, Tessema B G. Constraint Handling in Multiobjective Evolutionary Optimization[J]. IEEE Trans. on Evolutionary Computation, 2009, 13(2): 514-525.

[5] 雷德明, 吴智铭. 基于个体密集距离的多目标进化算法[J]. 计算机学报, 2005, 28(8): 1320-1326.

[6] 敖友云, 迟洪钦. 一种基于个体密度估算的多目标优化演化算法[J]. 计算机工程与应用, 2008, 44(15): 36-53.

编辑 张正兴
~~~~~~