文章编号:1671-4598(2020)03-0159-04 DOI:10.16526/j.cnki.11—4762/tp.2020.03.033 中图分类号:TP316.2 文献标识码:A

RTX 系统下并行 I/O 卡驱动程序的开发

刘士勋[1]，唐同斌[1]，喻 戈[1]，路 意[1]，闫小东[2]
(1. 西安现代控制技术研究所，西安 710065；
2. 中国人民解放军 93811 部队 保障部装备质量控制与安全监察中心)

摘要：研华 PCI—1751 板卡作为众多接口卡的一员被广泛使用，而出厂时该板卡并不携带 RTX 系统下的驱动程序，在实时性要求较高的环境下，该板卡的使用是不太方便的；因此，开发 RTX 操作系统下 PCI—1751 板卡驱动程序成为了一个急需解决的问题；文章利用 RTX 提供的接口，使用 I/O 映射的方法操作板卡寄存器，讨论 PCI—1751 板卡的驱动开发流程，最后通过一个自发自收闭环实验，对比 Windows 驱动与 RTX 驱动的实时性差异。

关键词：RTX；DIO；驱动程序

Parallel I/O Card Driver Development Based on RTX System

Liu Shixun[1], Tang Tongbin[1], Yu Ge[1], Lu Yi[1], Yan Xiaodong[2]
(1. Xi'an Institute of Modern Control Technology, Xi'an 710065, China;
2. Unit 93811 of PLA, China)

Abstract: Advantech PCI—1751 card, as an important interface card, is widely used in lots of universities and institutes. However, this card doesn't provide the user with the driver in RTX system after manufactured. For users who want to use this card in RTX environment, maybe it is not that convenient. This problem is very urgent to be solved. This paper, using the RTX API and I/O map functions, introduces the RTX driver developing process and finally shows the driver performance between Windows and RTX.

Keywords: RTX; DIO; driver

0 引言

PCI—1751 卡是一块基于 PCI 总线的拥有 48 路并行 I/O 的板卡。由于可以同时控制多路电平输入输出，该板卡广泛于工业交流/直流 I/O 设备监控、继电器和开关控制、并行数据传输、感应 TTL 信号逻辑、驱动 LED 指示器等环境。同时 PCI—1751 板卡上也集成了 2 个 8254 定时器/计数器，也可用于一些高精度定时计数的功能场景。

RTX 操作系统作为 Windows 系统的扩展系统，受到许多高校研究单位的青睐。而很多板卡在出厂时都不提供 RTX 驱动程序，PCI—1751 板卡也不例外。因此，本文结合研华科技公司的 PCI—1751 板卡，介绍 RTX 系统下板卡驱动的编写调试方法及一些经验，以求 RTX 驱动程序的开发被更多探索。

1 RTX 驱动程序机理

1.1 RTX 下 PCI 驱动程序开发实质

寄存器是板卡上的具有特定功能的内存存储。用户可以不了解板卡内部的具体硬件实现，但只要能理解其意义并通过地址访问到寄存器，即可实现板卡功能，故称为板卡与用户之间的软件接口。

因此，RTX 下 PCI 板卡驱动的开发实质就是利用 RTX 系统函数操纵板卡上的寄存器。

1.2 PCI 板卡寄存器的分类

开发驱动的用户需要关心两类寄存器，即 PCI 配置寄存器与板卡功能寄存器。

1.2.1 PCI 配置寄存器

PCI 配置寄存器是每块板卡寄存器的“目录”，是 PCI 协议预定义的 256 字节的内存[3]。在该寄存器中，标识了该板卡的所有有用信息，其具体内容如表 1 所示。

表 1 PCI 配置空间

地址	定义	
0x00	DeviceID	VendorID
0x04	状态寄存器	控制寄存器
…	…	
0x10	基地址寄存器 1	
0x14	基地址寄存器 2	
0x18	基地址寄存器 3	
0x1C	基地址寄存器 4	
0x20	基地址寄存器 5	
…	…	

1) DeviceID 与 VendorID：

每类板卡独一无二的属性[1]，用户在遍历计算机系统中的所有板卡时，可根据这两个值，来判断该板卡是否存在于该计算机系统。

收稿日期:2019-07-28； **修回日期**:2019-08-29。

作者简介:刘士勋(1992-)，男，陕西西安人，硕士研究生，助理工程师，主要从事半实物仿真技术方向的研究。

2）基地址寄存器：

用于存放寄存器映射的基地址。基地址是板卡功能寄存器的起始地址，用户可以根据基地址和偏移地址计算板卡上所有功能寄存器的地址。

1.2.2 板卡功能寄存器

板卡功能寄存器是板卡功能的软件接口，用户只需对这些寄存器置数取数，即可完成与之对应的功能。

板卡在出厂硬件手册都会附带寄存器功能说明及地址分布，这些地址都是从基地址开始有规律累加的，每个寄存器相对基地址的累加量称作偏移地址。因此，只要找到了板卡 I/O 基地址，所有寄存器的地址都可以很容易的推算出来。

在图 1 所示的计算机环境中，PCI－1751 的内存基地址为 0xFEBFF400，这与板卡 PCI 配置空间的基地址寄存器 2 中的值所吻合，我们可得到其基地址存放在基地址寄存器 2 中。感兴趣的读者可以通过该方法自己动手验证基地址是否存放于 PCI 配置寄存器中的基地址寄存器 2 中。

图 1 板卡资源对话框

2 PCI－1751 板卡简介

2.1 板卡功能

研华 PCI－1751 接口卡是一块具有 48 路并行 DI/O 输入输出卡，同时该板卡也携带 3 个定时器/计数器，可以完成高精度的定时计数功能。

该板卡借鉴了 8255 芯片的设计思路，实现了两块 8255 芯片的 mode 0 模式，共具有 24x2＝48 路 DI/O 通道。同时该板卡的 I/O 驱动能力远超出于普通的 8255 芯片。

同时，PCI－1751 板卡提供了断电保护功能，当所在机器遭遇突发断电又瞬时恢复的情况，板卡可以保持之前保存的通道输出值。

2.2 板卡 DI/O 通道分组

PCI－1751 板卡上两个增强的 8255 芯片的 48 路 DI/O 通道被分为 6 个组，分别为 PA0、PB0、PC0（PC0H、PC0L）、PA1、PB1、PC1（PC1H、PC1L）。每个通道组可以单独配置输入输出方向，PC0 和 PC1 组高低字节也可单独配置，互不影响。

2.3 相关寄存器配置

PCI－1751 的寄存器地址列表如表 2 所示。

板卡上的硬件跳线可以强制配置 I/O 口输入输出方向。当跳线配置为软件配置模式时，需要在使用前先写入控制字。控制寄存器的偏移地址为 0 和 7，对应 Port0 和 Port1，其内容格式如表 3 所示。

表 2 PCI－1751 寄存器地址

偏移地址	功能	
	读	写
0	PortA0	PortA0
1	PortB0	PortB0
2	PortC0	PortC0
3		Port0 配置寄存器
4	PortA1	PortA1
5	PortB1	PortB1
6	PortC1	PortC1
7		Port1 配置寄存器
8－23	预留	
24	8254 计数器 0	8254 计数器 0
25	8254 计数器 1	8254 计数器 1
26	8254 计数器 2	8254 计数器 2
27		8254 控制寄存器
28－31	预留	
32	中断状态寄存器	中断控制寄存器

表 3 Port0、Port1 配置寄存器

7	6	5	4	3	2	1	0
			PortA	PortCH		PortB	PortCL

对于 Port0、Port1 配置寄存器，写 1 为输入方向，写 0 为输出方向。例如，只想配置 PC0 通道组为输入通道，其他通道均为输出通道，则应将控制字 0x09（00001001B）写入偏移地址为 3 的寄存器中。

配置好输入/输出方向后，对相应的通道寄存器进行读/写即可完成输入/输出操作。例如读取 PC0 通道组，只需读取 base＋2 的寄存器值即可。

2.4 板卡定时器/计数器

PCI－1751 板卡上携带三块 8254 计数器芯片，定时器连接关系如图 2 所示。

板卡在设计时，为了提供更多的灵活性，Timer1 的 CLK 引脚可以通过跳线连接到外部信号源 CLK1，亦可连接到 Timer0 的输出端。当 Timer1 的时钟源连接到 Timer0 的输出端时，相当于 Timer0 与 Timer1 串联形成一个 32 位的计数器。

板卡内部的定时器晶振频率为 10 MHz，使用 Timer0 与 Timer1 进行定时，最大定时频率为 10 MHz/2＝5 MHz；最小定时频率为 10 MHz/65 536/65 536＝0.002 328 Hz。

2.5 板卡中断寄存器

在表 2 所示的寄存器列表中，特别值得关注的是偏移地址为 32 的中断控制/状态寄存器。该寄存器在写入时作为控制寄存器，读取时作为状态寄存器，他们使用相同的

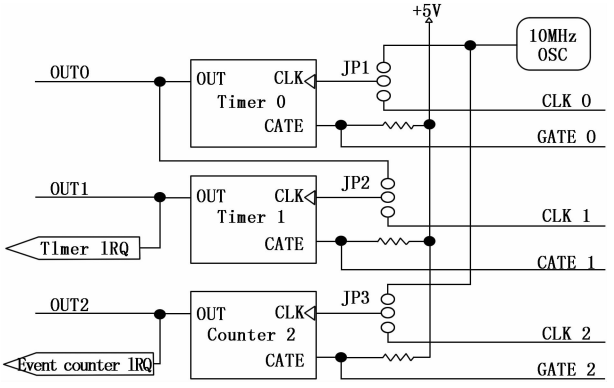

图 2　定时器/计数器结构图

偏移地址。

1751 板卡将 PC00、PC04、Timer1、PC10、PC14、Timer2 的输出引入到板卡的中断电路中。中断控制寄存器决定了中断源的选择、中断触发模式等设置，中断状态寄存器显示当前中断配置与触发状态，其定义如表 4 所示。

表 4　中断控制/状态寄存器

Port #	Port 1				Port 0			
位数	76	5	4	3	2	1	0	
缩写	F	E	M1	M0	F	E	M1	M0

其中：F 是中断标志，作为状态寄存器时，该位表示中断是否发生；作为控制寄存器，写 0 是对中断标志的清除。E 是上升沿/下降沿的配置，1 为上升沿，0 为下降沿。M1M0 是中断源的选择，具体示意如表 5 所示。

表 5　中断模式配置

Port 1			Port 0		
M1	M0	描述	M1	M0	描述
0	0	关闭中断	0	0	关闭中断
0	1	PC10	0	1	PC00
1	0	PC10&PC14	1	0	PC00&PC04
1	1	计数器 2	1	1	计数器 1

例如，在本文的第四章节所介绍的实验中，欲检测 PC00 的上升沿中断，需将 0x05 写入中断控制寄存器中即可完成配置；中断服务函数中，就是通过该寄存器的 D3 位即可检测 PC00 中断是否来临。

有了这些知识储备，即可开始进行板卡驱动的开发。

3　RTX 驱动程序开发示例

在本文示例的驱动程序中，主要提供关于 DI/O 操作的几个重要函数，分别为打开板卡函数、中断配置函数、配置通道组函数、读通道组函数、写通道组函数、等待中断函数。通过这些函数，该板卡可以完成多路电平的输入输出以及上升沿/下降沿中断采集的功能。

3.1　打开板卡函数—OpenCard _ PCI1751

PC 机可能存在很多板卡，因此在打开板卡函数的实现中，主要操作为根据 DeviceID 和 VendorID 搜索 PCI－1751 板卡是否存在。如果搜寻到板卡，则保存板卡的 I/O 映射基地址，方便后续读写板卡内部寄存器时使用。

示例代码如下：

```
for ( bus=0; bFlag; bus++ )
for ( deviceNumber = 0; deviceNumber < PCI _ MAX _ DEVICES&&bFlag;deviceNumber++ )
for(functionNumber=0;functionNumber<PCI_MAX_FUNCTION; functionNumber++ )
{
bytesWritten = RtGetBusDataByOffset(…)
if(( PciData->VendorID == vendorID ) && ( PciData->DeviceID == deviceID ))
base_add = base_add_register[2];//get add
}
```

这三层循环会遍历所在计算机系统中的所有板卡。通过 RTX 系统提供的接口 RtGetBusDataByOffset，可以获得 PCI 配置空间的内存指针，即表 1 所示的内存区域，将该内存中的 DeviceID 和 VendorID 成员与 PCI－1751 板卡的进行对比，即可验证当前所遍历板卡是否为 1751 板卡。如果找到，保存 I/O 映射基地址。

对于 PCI－1751 板卡而言，DeviceID 为 0x1751，VendorID 为 0x13FE。

3.2　打开中断——EnableInterrupt _ 1751

打开中断函数内部完成两个操作。

首先根据用户需求，对中断控制寄存器进行配置，其次使用 RTX 提供的 API 函数 RtAttachInterruptVector 对 PCI 中断进行挂接响应。

完成上述两个设置之后，板卡上被使能的中断就可以触发中断服务函数。

中断寄存器的配置示例代码如下：

```
IntCmd = IntMode<<(port*4);
RtWritePortUchar(BaseAdd+32,(UCHAR)IntCmd);
```

其中，IntMode 对应表 5 中的中断模式选择，取值 0～3；port 定义为 Port 口编号，Port0 为 0，Port1 为 1。

3.3　配置通道组——SetPortDirection _ 1751

在配置通道组函数中，主要操作就是对欲使用的通道组的控制字进行设置，然后将控制字写入对应的寄存器中。

示例代码如下：

```
dirsetting=PA<<4+PCH<<3+PB<<1+PCL;
RtWritePortUchar(BaseAdd+(port+1)*4-1,(UCHAR)dirsetting);
```

在形参列表中，PA、PCH、PB、PCL 是通道组输入输出方向，定义为输出传 0，输入传 1；port 定义为 Port 口编号，Port0 为 0，Port1 为 1。

3.4　读通道组——ReadPort _ 1751

读取通道组，就是读取指定通道对应的寄存器。

示例代码如下：

```
if (channel >= 3) channel += 1;
```

```
cResult=RtReadPortUchar(BaseAdd+channel);
```

在形参列表中，channel 代表 I/O 口编号，定义为 PortA0、PortB0、PortC0、PortA1、PortB1、PortC1 依次为 0～5。

3.5 写通道组——WritePort _ 1751

输出通道组，就是向指定通道对应的寄存器上写值。

示例代码如下：

```
if (channel >= 3) channel += 1;
RtWritePortUchar(BaseAdd+channel, (UCHAR)value);
```

在形参列表中，channel 代表 I/O 口编号，定义为 PortA0、PortB0、PortC0、PortA1、PortB1、PortC1 依次为 0～5。

3.6 等待中断——WaitPortInterrupt_1751

3.6.1 原理解析

对于像 Windows、RTX 这样的多任务操作系统，每个任务对应一个运行的进程，每个运行的进程中又可以包含很多线程。如果没有同步机制，所有的线程会任意运行。然而，多个线程可能会要求同一个资源，这就需要同步处理。

等待中断函数就使用到了同步机制。调用等待函数后，其内部的等待同步对象的函数，例如 WaitForSingleObject 函数，就会处于等待状态，对于用户，其表征为“卡死”状态，只有当中断触发后，中断服务函数内部对该同步对象使能后，等待同步对象的函数才会释放线程占有权，等待中断函数才能继续运行下去。

RTX 操作系统提供的等待信号量的函数为 RtWaitForSingleObject，形参和用法兼容 Windows 操作系统函数。形参 1 是信号量的句柄，形参 2 是等待时间，当形参 2 传入 INFINITE 时，永久等待，直至信号量有效。等待中断函数就是利用永久等待信号量来实现的。

3.6.2 函数实现

等待中断函数内部对两个 port 口，3 类中断进行等待。当用户调用该函数时，先清空对应信号量，然后等待信号量，此时该函数处于阻塞状态。

中断服务函数检测到中断触发后，将对应信号量激活。等待中断函数才能继续进行，达到了“卡死”等待的作用。

这里对 port0 口的 PC00 中断进行示意。

```
IntCmd = 0x01;//中断源，对应表 5
RtWritePortUchar(BaseAdd+32, (UCHAR) IntCmd);//写中断控制寄存器
RtWaitForSingleObject(hInterHandle[0], INFINITE);
Printf("PC00 Int found/n");//中断到达了
return 0;
```

在中断服务函数内部，其核心代码如下：

```
temp1=RtReadPortUchar(base+32);//得到中断状态寄存器
if (temp1 & 0x08)//对比表 4 中的 D3 位
  RtSetEvent(hInterHandle[0]);
```

4 RTX 驱动程序测试

4.1 测试原理

对于板卡驱动性能的测试，这里使用了一个“自发自收”的闭环测试模型，即板卡 PA00 自己产生上升沿，板卡 PC00 采集该上升沿，通过对比上升沿产生前后的时间间隔来衡量驱动程序的性能。测试流程如图 3 所示。

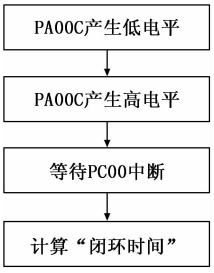

图 3 驱动测试流程

4.2 测试方法

板卡硬件上用导线连接引脚 1 与引脚 19，即 PA00 引脚与 PC00 引脚。

软件上将 PA 口配置为输出方向，PC 口设置为输入方向，这样 PA00 的电压会被 PC00 实时采集。

I/O 口方向设置好后，使 PA00 口先输出低电平，再输出高电平，等待 PC00 口检测到该上升沿触发中断。

Windows 与 RTX 的测试程序均按照图 3 所示流程进行编写，具体流程如下：

1）打开板卡，配置 PA 口为输出方向，PC 口为输入方向；

2）配置中断控制寄存器，使能 PC00 上升沿中断；

3）记录当前时刻 t_1；

4）PA00 输出低电平；

5）PA00 输出高电平；

6）等待 PC00 上升沿中断，记录中断触发时刻 t_2；

7）计算“闭环”时间 t_2-t_1；

8）程序结束。

4.3 测试环境与考核指标

本次试验使用研华 610L 原装机箱作为测试硬件环境，系统环境为 Windows XP SP3+RTX8.1，编译器使用 Visual Studio 6.0。

Windows 与 RTX 实验程序均按照 4.2 节中的流程开发，t_1 和 t_2 通过系统函数获取，t_2 与 t_1 的差值作为最终考核指标。

4.4 测试结果与分析

实验 100 次取平均值作为测试最终结果，Windows 驱动与 RTX 驱动的“闭环”测试结果如表 6 所示。

表 6 驱动测试结果 （单位：ms）

	闭环时间	最大值	最小值
Windows 驱动	0.046	0.051	0.043
RTX 驱动	0.015	0.016	0.013

通过平均值的对比可以看到，RTX 驱动程序相比 Windows 驱动，响应时间缩短了 68%，性能提升相当明显。

（下转第 167 页）