PCI8735 WIN2000/XP 驱动程序使用说明书

请您务必阅读《使用纲要》，他会使您事半功倍!

目　　录

PCI8735 WIN2000/XP 驱动程序使用说明书 1
第一章 版权信息与命名约定 2
第一节、版权信息 2
第二节、命名约定 2
第二章 使用纲要 2
第一节、使用上层用户函数，高效、简单 2
第二节、如何管理PCI设备 2
第三节、如何实现开关量的简便操作 3
第四节、哪些函数对您不是必须的 3
第三章 PCI即插即用设备操作函数接口介绍 3
第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI8735_”） 4
第二节、设备对象管理函数原型说明 5
第三节、AD采样操作函数原型说明 8
第四节、AD硬件参数保存与读取函数原型说明 9
第五节、DIO数字量输入输出操作函数原型说明 11
第四章 硬件参数结构 12
第五章 数据格式转换与排列规则 13
第一节、AD原码LSB数据转换成电压值的换算方法 13
第二节、AD采集函数的ADBuffer缓冲区中的数据排放规则 13
第三节、AD测试应用程序创建并形成的数据文件格式 14
第六章 上层用户函数接口应用实例 15
第一节、怎样使用ReadDeviceAD函数直接取得AD数据 15
第二节、怎样使用GetDeviceDI函数进行更便捷的数字开关量输入操作 15
第三节、怎样使用SetDeviceDO函数进行更便捷的数字开关量输出操作 15
第七章 AD高速大容量、连续不间断数据采集及存盘技术详解 15
第八章 共用函数介绍 17
第一节、公用接口函数总列表（每个函数省略了前缀“PCI8735_”） 17
第二节、PCI内存映射寄存器操作函数原型说明 18
第三节、IO端口读写函数原型说明 25
第四节、线程操作函数原型说明 27
第五节、文件对象操作函数原型说明 29
第六节、各种参数保存和读取函数原型说明 31
第七节、其他函数原型说明 33

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_ 则被省略。如 PCI8735_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。诸如InitDeviceAD、ReadDeviceAD、SetDeviceDO等。而底层用户函数如 WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。

第二节、如何管理 PCI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 CreateDevice函数创建一个设备对象句柄hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如 InitDeviceAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceAD函数可以用hDevice句柄实现对AD数据的采样读取，SetDeviceDO函数可用实现开关量的输出等。最后可以通过 ReleaseDevice将hDevice释放掉。

注意：图中较粗的虚线表示对称关系。如红色虚线表示 CreateDevice和 ReleaseDevice两个函数的关系是：最初执行一次 CreateDevice，在结束是就须执行一次 ReleaseDevice。

第三节、如何实现开关量的简便操作

当您有了hDevice设备对象句柄后，便可用 SetDeviceDO函数实现开关量的输出操作，其各路开关量的输出状态由其bDOSts[16]中的相应元素决定。由 GetDeviceDI函数实现开关量的输入操作，其各路开关量的输入状态由其bDISts[16]中的相应元素决定。

第四节、哪些函数对您不是必须的

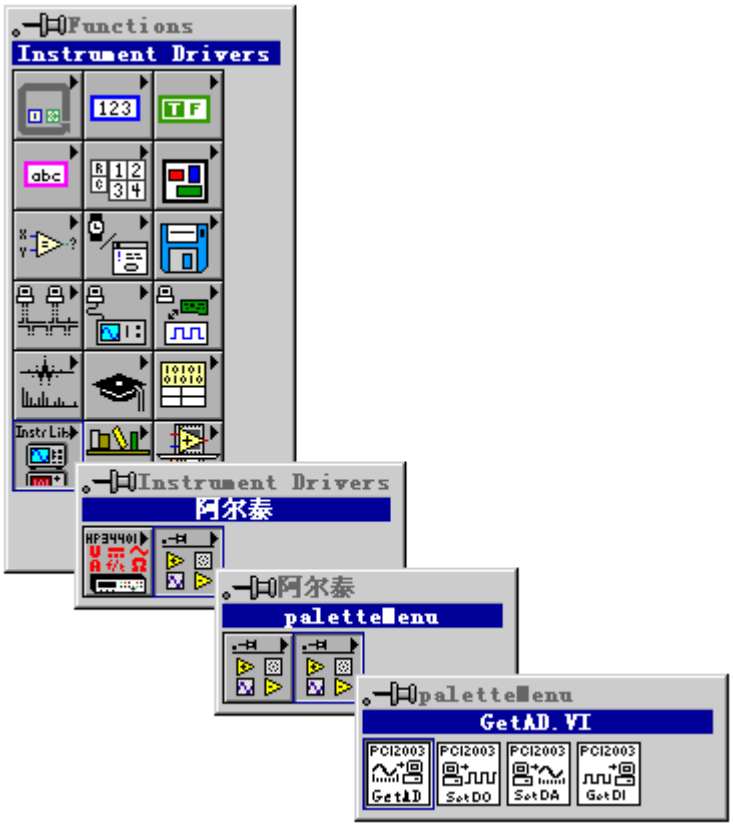
公共函数如 CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么 GetDeviceAddr， WriteRegisterByte， WriteRegisterWord， WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而 WritePortByte， WritePortWord， WritePortULong， ReadPortByte， ReadPortWord， ReadPortULong则对PCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PCI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如 InitDeviceAD）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用 ReadDeviceAD函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉PCI总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解PCI的资源配置空间、PNP即插即用管理，而只须用 GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用 ReadRegisterULong和 WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI8735_”）

函数名	函数功能	备注
设备对象操作函数		
CreateDevice	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
CreateDevicEx	创建 PCI 设备对象(用设备物理号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
程序方式 AD 读取函数		
InitDeviceAD	初始化 AD 部件准备传输	上层用户
ReadDeviceAD	连续读取当前 PCI 设备上的 AD 数据	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	上层用户
RetDeviceDO	回读开关量输出状态	上层用户

使用需知：

Visual C++：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:PCI8735\Samples\ PCI8735.H”

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PCI8735.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话中选择 PCI8735.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

请注意，因考虑 Visual C++和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所举的 Visual Basic 程序均是需要编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不能保证完全顺利运行。

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：

CreateDevice 

一、在 LabView 中打开 PCI8735.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

二、根据LabView语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如 ReadDeviceAD接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“［U16］”为无符号 16 位短整型数组或缓冲区或指针，“［U32］”与“［U16］”同理，只是位数不一样。

第二节、设备对象管理函数原型说明

◆ **创建设备对象函数（逻辑号）**

函数原型：

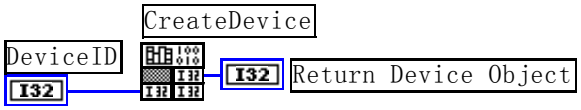
Visual C++：

HANDLE CreateDevice (int DeviceLgcID = 0)

Visual Basic：

Declare Function CreateDevice Lib "PCI8735" (Optional ByVal DeviceLgcID As Integer = 0) As Long

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PCI 设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 IOC1320 模板时，驱动程序逻辑号为“0”来确认和管理第一个设备，若用户接着再添加第二个 IOC1320 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，

若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 PCI 设备时，DeviceLgcID 应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由 BIOS 和操作系统加载设备时，依据主板总线编号等信息进行这个设备 ID 号分配，说得简单点，就是加载设备的顺序编号，编号的递增顺序为 0、1、2、3……。所以用户无法直接固定某一个设备的在设备列表中的物理位置，若想固定，则必须须使用物理 ID 号，调用 CreateDevicEx 函数实现。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice　CreateDevicEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI8735_CreateDevice (DeviceLgcID); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
 {
     return;     // 退出该函数
 }
    :
```

Visual Basic 程序举例

```
    :
Dim hDevice As Long   ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI8735_CreateDevice (DeviceLgcID)   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
     MsgBox “创建设备对象失败”
     Exit Sub      ' 退出该过程
End If
     :
```

- **创建设备对象函数（物理号）**

函数原型：

Visual C++：

HANDLE CreateDeviceEx(int DevicePhysID = 0)

Visual Basic：

Declare Function CreateDeviceEx Lib "PCI8735" (Optional ByVal DevicePhysID As Integer = 0) As Long

LabVIEW：

请参考相关演示程序。

功能：该函数使用物理 ID 号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：

DevicePhysID 物理设备ID(Physic Device Identifier)标识号。由 CreateDevice函数的DeviceLgcID参数说明中可以看出，逻辑ID号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如A、B、C、D四个卡，构成 128 个通道（32*4），其通道序列为 0-127，每个通道接入不同物理意义的模拟信号，我们要求A卡位于 0-31 通道上，B卡位于 32-63 通道上，C卡位于 64-95 通道上，而D卡则位于 96-127 通道上，而其逻辑设备ID号在同一台计计算机上按不同顺序插入会发生变化，即便在不同计算机上按相同顺序插入也可能会因主板制造商的不同定义而发生变化，所以您可能由此无法确定 0-127 的通道分别接入了什么信号。那么如何将各个设备在设备链中的物理位置固定下来呢？那么物理设备ID的使用帮您解决了这个问题。它是在卡上提供了一个拔码器DID，可以由用户为各个设备手动设置不同的物理ID号，当调用 CreateDeviceEx函数时，只需要指定该参数的值与您在拔码器上设定的值一样即可，驱动程序会自动跟踪拔码器值与此相等的设备。它的取值范围通常在[0, 15]之间。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **取得本计算机系统中 PCI8735 设备的总数量**

函数原型：

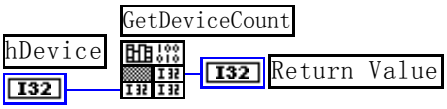
Visual C++:

```
int GetDeviceCount (HANDLE hDevice)
```

Visual Basic:

```
Declare Function GetDeviceCount Lib "PCI8735" (ByVal hDevice As Long ) As Integer
```

LabVIEW：



功能：取得 PCI8735 设备的数量。

参数：hDevice设备对象句柄，它应由 CreateDevice或CreateDevicEx创建。

返回值：返回系统中 PCI8735 的数量。

相关函数： CreateDevice CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **取得该设备当前逻辑 ID**

函数原型：

Visual C++:

```
BOOL GetDeviceCurrentID (HANDLE hDevice，
                         PLONG DeviceLgcID)
```

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib " PCI8735" (ByVal hDevice As Long,_
                                                    ByRef DeviceLgcID As Long,_
                                                    ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：取得指定设备逻辑和物理 ID 号。

参数：

hDevice 设备对象句柄，它指向要取得逻辑和物理号的设备，它应由 CreateDevice或CreateDevicEx创建。

DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。

DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拔码器 DID 决定。

返回值：如果初始化设备对象成功，则返回TRUE， 否则返回FALSE， 用户可用 GetLastError捕获当前错误码，并加以分析。

相关函数： CreateDevice CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **用对话框控件列表计算机系统中所有 PCI8735 设备各种配置信息**

函数原型：

Visual C++:

```
BOOL ListDeviceDlg (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ListDeviceDlg Lib " PCI8735" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI8735 的硬件配置信息。

参数：hDevice设备对象句柄，它应由 CreateDevice或CreateDevicEx创建。

返回值：若成功，则弹出对话框控件列表所有 PCI8735 设备的配置情况。

相关函数： CreateDevice CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦ **释放设备对象所占的系统资源及设备对象**

函数原型：

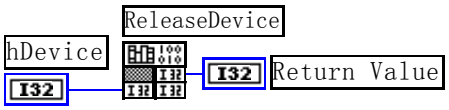
Visual C++:

```
BOOL ReleaseDevice(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDevice Lib " PCI8735" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice设备对象句柄，它应由 CreateDevice或CreateDevicEx创建。

返回值：若成功，则返回TRUE， 否则返回FALSE， 用户可以用 GetLastError捕获错误码。

相关函数： CreateDevice CreateDevicEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

应注意的是，CreateDevice必须和 ReleaseDevice函数一一对应，即当您执行了一次 CreateDevice后，再一次执行这些函数前，必须执行一次 ReleaseDevice函数，以释放由 CreateDevice占用的系统软硬件资源，如DMA控制器、系统内存等。只有这样，当您再次调用 CreateDevice函数时，那些软硬件资源才可被再次使用。

第三节、AD 采样操作函数原型说明

♦ **初始化 AD 设备（ Initlize device AD for program mode）**

函数原型

Visual C++:

```
BOOL InitDeviceAD ( HANDLE hDevice,
                    PPCI8735_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceAD Lib "PCI8735" (ByVal hDevice As Long, _
                                             ByRef pADPara As PCI8735_PARA_AD) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：它负责初始化设备对象中的 AD 部件，为设备的操作就绪做有关准备工作，如预置 AD 采集通道、采样频率等。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

pADPara 设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如AD采样通道、采样频率等。关于PCI8735_PARA_AD具体定义请参考PCI8735.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《AD硬件参数结构》。

返回值：如果初始化设备对象成功，则返回TRUE， 否则返回FALSE， 用户可用 GetLastError捕获当前错误码，并加以分析。

相关函数： CreateDevice InitDeviceAD ReadDeviceAD ReleaseDeviceAD ReleaseDevice

♦ **读取 PCI 设备上的 AD 数据**

函数原型:

Visual C++:

```
BOOL ReadDeviceAD(HANDLE hDevice,
                  USHORT ADBuffer[],
                  LONG nReadSizeWords,
                  PLONG nRetSizeWords = NULL)
```

Visual Basic:

```
Declare Function ReadDeviceAD Lib "PCI8735" (ByVal hDevice As Long, _
                                             ByRef ADBuffer As Integer,_
                                             ByVal nReadSizeWords As Long,_
```

Optional ByRef nRetSizeWords As Long = 0) As Boolean

LabVIEW：

请参考相关演示程序。

功能：读取PCI设备AD部件上的批量数据。它不负责初始化AD部件，待读完整过指定长度的数据才返回。它必须在 InitDeviceAD之后， ReleaseDeviceAD之前调用。

参数：

hDevice设备对象句柄，它应由 CreateDevice或CreateDevicex创建。

ADBuffer 接受AD数据的用户缓冲区，它可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords指定一次 ReadDeviceAD操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间。此参数值只与ADBuffer[]指定的缓冲区大小有效，而与FIFO存储器大小无效。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值：如果调用成功，则返回TRUE，且AD立刻开始转换，否则返回FALSE，用户可用 GetLastError捕获当前错误码，并加以分析。

注释：此函数也可用于单点读取和几个点的读取，只需要将nReadSizeWords设置成 1 或相应值即可。其使用方法请参考《AD高速大容量、连续不间断数据采集及存盘技术详解》章节。

相关函数： CreateDevice InitDeviceAD ReadDeviceAD ReleaseDeviceAD ReleaseDevice

- **释放设备上的 AD 部件**

函数原型：

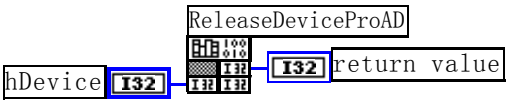
Visual C++:

BOOL ReleaseDeviceAD(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceAD Lib "PCI8735" (ByVal hDevice As Long) As Boolean

LabVIEW:



功能：释放设备上的 AD 部件。

参数：hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

返回值：若成功，则返回TRUE， 否则返回FALSE， 用户可以用 GetLastError捕获错误码。

应注意的是，InitDeviceAD必须和 ReleaseDeviceAD函数一一对应，即当您执行了一次 InitDeviceAD后，再一次执行这些函数前，必须执行一次 ReleaseDeviceAD函数，以释放由 InitDeviceAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用 InitDeviceAD函数时，那些软硬件资源才可被再次使用。

相关函数： CreateDevice InitDeviceAD ReadDeviceAD ReleaseDeviceAD ReleaseDevice

- **程序查询方式采样函数一般调用顺序**

① CreateDevice
② InitDeviceAD
③ ReadDeviceAD
④ ReleaseDeviceAD
⑤ ReleaseDevice

注明：用户可以反复执行第③步，以实现高速连续不间断大容量采集。

第四节、AD 硬件参数保存与读取函数原型说明

- **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

BOOL LoadParaAD(HANDLE hDevice,

PPCI8735_PARA_AD pADPara)

Visual Basic:

Declare Function LoadParaAD Lib "PCI8735" (ByVal hDevice As Long, _
ByRef pADPara As PCI8735_PARA_AD) As Boolean

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

pADPara属于PPCI8735_PARA_AD的结构指针类型，它负责返回PCI硬件参数值，关于结构指针类型PPCI8735_PARA_AD请参考PCI8735.h或PCI8735.Bas或PCI8735.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++:

BOOL SaveParaAD (HANDLE hDevice,
PPCI8735_PARA_AD pADPara)

Visual Basic:

Declare Function SaveParaAD Lib "PCI8735" (ByVal hDevice As Long, _
ByRef pADPara As PCI8735_PARA_AD) As Boolean

LabVIEW:

请参考相关演示程序。

功能：负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

pADPara设备硬件参数，关于PCI8735_PARA_AD的详细介绍请参考PCI8735.h或PCI8735.Bas或PCI8735.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

Viusal C++:

BOOL ResetParaAD (HANDLE hDevice,
PPCI8735_PARA_AD pADPara)

Visual Basic:

Declare Function ResetParaAD Lib "PCI8735" (ByVal hDevice As Long, _
ByRef pADPara As PCI8735_PARA_AD) As Boolean

LabVIEW:

请参考相关演示程序。

功能：将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无法确定错误原因的后果。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

pADPara设备硬件参数，它负责在参数被复位后返回其复位后的值。关于PCI8735_PARA_AD的详细介绍请参考PCI8735.h或PCI8735.Bas或PCI8735.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构

的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

注意：在您编写工程应用软件时，若要更方便的保存和读取您特有的软件参数，请不防使用我们为您提供的辅助函数：SaveParaInt、 LoadParaInt、SaveParaString、LoadParaString，详细说明请参考共用函数介绍章节中的《各种参数保存和读取函数原型说明》。

第五节、DIO 数字量输入输出操作函数原型说明

- **开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   BYTE bDISts[16])
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI8735" ( ByVal hDevice As Long, _
                                             ByVal bDISts(0 to 15) As Byte) As Boolean
```

LabVIEW

请参考相关演示程序。

功能：负责将 PCI 设备上的输入开关量状态读入到 bDISts[x]数组参数中。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

bDISts 十六路开关量输入状态的参数结构，共有 16 个元素，分别对应于 DI0-DI15 路开关量输入状态位。如果 bDISts[0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

返回值：若成功，返回 TRUE，其 bDISts[x]中的值有效；否则返回 FALSE，其 bDISts[x]中的值无效。

相关函数: CreateDevice SetDeviceDO ReleaseDevice

- **开关量输出**

函数原型：

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[16])
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI8735" (ByVal hDevice As Long, _
                                            ByVal bDOSts(0 to 15) As Byte) As Boolean
```

LabVIEW

请参考相关演示程序。

功能：负责将 PCI 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

bDOSts 十六路开关量输出状态的参数结构，共有 16 个元素，分别对应于 DO0-DO15 路开关量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice GetDeviceDI ReleaseDevice

- **回读数字量输出状态**

函数原型：

Visual C++:

```
BOOL RetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts [16])
```

Visual Basic:

```
Declare Function RetDeviceDOLib "PCI8735" (ByVal hDevice As Long, _
                                            ByVal bDOSts (0 to 15) As Byte) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

bDOSts 十六路开关量输出状态的参数结构，共有 16 个元素，分别对应于 DO0-DO15 路开关量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice GetDeviceDI ReleaseDevice

◆ **以上函数调用一般顺序**

① CreateDevice

② SetDeviceDO(或 GetDeviceDI，当然这两个函数也可同时进行)

③ ReleaseDevice

用户可以反复执行第②步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及 AD 采样可以同时进行，互不影响）。

第四章 硬件参数结构

AD 硬件参数结构（PCI8735_PARA_AD）

Visual C++：

```
typedef struct _PCI8735_PARA_AD
{
    LONG FirstChannel;          // 首通道[0, 31]
    LONG LastChannel;           // 末通道[0, 31],要求末通道必须大于或等于首通道
    LONG InputRange;            // 模拟量输入量程范围
    LONG GroundingMode;         // 接地方式(单端或双端选择)
    LONG Gains;                 // 程控增益
} PCI8735_PARA_AD, *PPCI8735_PARA_AD;
```

Visual Basic：

```
Private Type PCI8735_PARA_AD
    FirstChannel As Long            ' 首通道[0, 31]
    LastChannel As Long             ' 末通道[0, 31]，要求末通道必须大于或等于首通道
    InputRange As Long              ' 模拟量输入量程范围
    GroundingMode As Long           ' 接地方式(单端或双端选择)
    Gains As Long                   ' 程控增益
End Type
```

LabVIEW:

请参考相关演示程序。

该结构实在太简易了，其原因就是 PCI 设备是系统全自动管理的设备，再加上驱动程序的合理设计与封装，什么端口地址、中断号、DMA 等将与 PCI 设备的用户永远告别，一句话 PCI 设备是一种更易于管理和使用的设备。

此结构主要用于设定设备AD硬件参数值，用这个参数结构对设备进行硬件配置完全由 InitDeviceAD函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

FirstChannel AD采样首通道，其取值范围为[0, 31]，它应等于或小于 LastChannel参数。

LastChannel AD采样末通道，其取值范围为[0, 31]，它应等于或大于 FirstChannel参数。

InputRange 被测模拟信号输入范围，取值如下表：

常量名	常量值	功能定义
PCI8735_INPUT_N10000_P10000mV	0x00	±10000mV
PCI8735_INPUT_N5000_P5000mV	0x01	±5000mV
PCI8735_INPUT_N2500_P2500mV	0x02	±2500mV
PCI8735_INPUT_0_P10000mV	0x03	0～10000mV

关于各个量程下采集的数据ADBuffer[]如何换算成相应的电压值，请参考《AD原码LSB数据转换成电压值的换算方法》章节。

GroundingMode AD 接地方式选择。它的选项值如下表：

常量名	常量值	功能定义
PCI8735_ GNDMODE_SE	0x00	单端方式(SE:Single end)
PCI8735_ GNDMODE_DI	0x01	双端方式(DI:Differential)

Gains 程控增益，即将模拟量输入放大指定倍数后再给 AD 转换器转换。其取值范围如下表：

常量名	常量值	功能定义
PCI8735_GAINS_1MULT	0x00	1 倍增益
PCI8735_GAINS_2MULT	0x01	2 倍增益
PCI8735_GAINS_4MULT	0x02	4 倍增益
PCI8735_GAINS_8MULT	0x03	8 倍增益

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.0 / 8192) * (ADBuffer[0] & 0x1FFF) -10000.0	[-10000, +9995.11]
±5000mV	Volt = (10000.0 / 8192) * (ADBuffer[0] & 0x1FFF) -5000.0	[-5000, +4997.55]
±2500mV	Volt = (5000.0 / 8192) * (ADBuffer[0] & 0x1FFF) -2500.0	[-2500，+2498.77]
0~10000mV	Volt = (10000.0 /8192) * (ADBuffer[0] & 0x1FFF)	[0，+9997.55]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++：

```
Lsb = (ADBuffer[0])&0x1FFF;
Volt = (20000.00/8192) * Lsb -10000.00;
```

Visual Basic：

```
Lsb = (ADBuffer [0]) And &H1FFF
Volt = (20000.00/8192) * Lsb – 10000.00
```

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，当通道总数首末通道相等时，假如此时首末通道=5，其排放规则如下：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	…

两通道采集(假如 FirstChannel=0, LastChannel=1)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	…

四通道采集(假如 FirstChannel=0, LastChannel=3)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	…

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceAD 函数读回，即便不考虑是否能一次读完的问题，仅对于用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效呢? 还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	…
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	…
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	…
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;        // 文件头信息长度
    LONG FileType;
    // 该设备数据文件共有的成员
    LONG BusType;              // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;            // 该设备的编号(DEFAULT_DEVICE_NUM)
```

```
    LONG VoltBottomRange;          // 量程下限(mV)
    LONG VoltTopRange;             // 量程上限(mV)

    PCI8735_PARA_AD ADPara;        // 保存硬件参数

    LONG CrystalFreq;              // 晶振频率
    LONG HeadEndFlag;              // 头信息结束位
} FILE_HEADER,  *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第六章 上层用户函数接口应用实例

第一节、怎样使用 ReadDeviceAD函数直接取得AD数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]\|**[阿尔泰测控演示系统]**\| **[PCI8735 32 路 AD 和 16 路开关量卡]**\| **[Microsoft Visual C++]**\|**[简易代码演示]**\|**[AD 采集简易应用程序]**

第二节、怎样使用 GetDeviceDI函数进行更便捷的数字开关量输入操作

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]\|**[阿尔泰测控演示系统]**\| **[PCI8735 32 路 AD 和 16 路开关量卡]**\| **[Microsoft Visual C++]**\|**[简易代码演示]**\|**[DIO 开关量应用程序]**

第三节、怎样使用 SetDeviceDO函数进行更便捷的数字开关量输出操作

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]\|**[阿尔泰测控演示系统]**\| **[PCI8735 32 路 AD 和 16 路开关量卡]**\| **[Microsoft Visual C++]**\|**[简易代码演示]**\|**[DIO 开关量应用程序]**

第七章 AD 高速大容量、连续不间断数据采集及存盘技术详解

与 ISA、USB 设备同理，使用子线程跟踪 AD 转换进度，并进行数据采集是保持数据连续不间断的最佳方案。但是与 ISA 总线设备不同的是，PCI 设备在这里不使用动态指针去同步 AD 转换进度，因为 ISA 设备环形内存池的动态指针操作是一种软件化的同步，而 PCI 设备不再有软件化的同步，而完全由硬件和驱动程序自动完成。这样一来，用户要用程序方式实现连续数据采集，其软件实现就显得极为容易。每次用 ReadDeviceAD 函数读取 AD 数据时，那么设备驱动程序会按照 AD 转换进度将 AD 数据一一放进用户数据缓冲区，当完成该次所指定的点数时，它便会返回，当您再次用这个函数读取数据时，它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次 ReadDeviceAD 之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中，由于任务之间的调度切换非常平凡，特别是当用户移动窗口、或弹出对话框等，则会使当前线程猛地花掉大量的时间去处理这些图形操作，因此如果

处理不当，则将无法实现高速连续不间断采集，那么如何更好的克服这些问题呢？用子线程则是必须的（在这里我们称之为数据采集线程），但这还不够，必须要求这个线程是绝对的工作者线程，即这个线程在正常采集中不能有任何窗口等图形操作。只有这样，当用户进行任何窗口操作时，这个线程才不会被堵塞，因此可以保证其正常连续的数据采集。但是用户可能要问，不能进行任何窗口操作，那么我如何将采集的数据显示在屏幕上呢？其实很简单，再开辟一个子线程，我们称之数据处理线程，也叫用户界面线程。最初，数据处理线程不做任何工作，而是在 Win32 API 函数 WaitForSingleObject 的作用下进入睡眠状态，此时它基本不消耗 CPU 时间，即可保证其他线程代码有充分的运行机会（这里当然主要指数据采集线程），当数据采集线程取得指定长度的数据到用户空间时，则再用 Win32 API 函数 SetEvent 将指定事件消息发送给数据处理线程，则数据处理线程即刻恢复运行状态，迅速对这批数据进行处理，如计算、在窗口绘制波形、存盘等操作。

可能用户还要问，既然数据处理线程是非工作者线程，那么如果用户移动窗口等操作堵塞了该线程，而数据采集线程则在不停地采集数据，那数据处理线程难道不会因此而丢失采集线程发来的某一段数据吗？如果不另加处理，这个情况肯定有发生的可能。但是，我们采用了一级缓冲队列和二级缓冲队列的设计方案，足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K数据，那么我们就创建一个缓冲队列，在用户程序中最简单的办法就是开辟一个两维数组如ADBuffer [SegmentCount][SegmentSize]， 我们将SegmentSize视为数据采集线程每次采集的数据长度，SegmentCount则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32，则这个缓冲队列实际上就是数组ADBuffer [32][8192]的形式。那么如何使用这个缓冲队列呢？方法很简单，它跟一个普通的缓冲区如一维数组差不多，唯一不同是，两个线程首先要通过改变SegmentCount字段的值，即这个下标Index的值来填充和引用由Index下标指向某一段SegmentSize长度的数据缓冲区。需要注意的是两个线程不共用一个Index下标变量。具体情况是当数据采集线程在AD部件被 InitDeviceAD初始化之后，首次采集数据时，则将自己的ReadIndex下标置为 0，即用第一个缓冲区采集AD数据。当采集完后，则向数据处理线程发送消息，且两个线程的公共变量SegmentCount加 1，（注意SegmentCount变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了，但是却没被数据处理线程处理掉的缓冲区数量。）然后再接着将ReadIndex偏移至 1，再用第二个缓冲区采集数据。再将SegmentCount加 1，直到ReadIndex等于 31 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从SegmentCount变量中减去在所接受到的当前事件下所处理的缓冲区个数，具体处理哪个缓冲区由CurrentIndex指向。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对SegmentCount加以判断，观察其值是否大于了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。

图 8.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程填充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

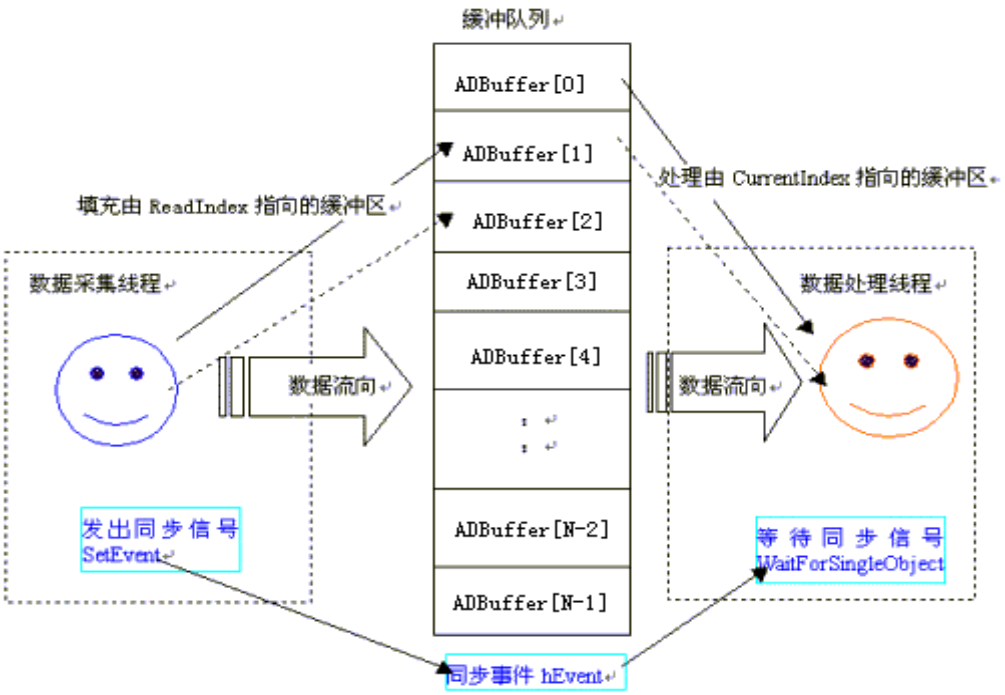


图 8.1

下面用 Visual C++程序举例说明。

使用 ReadDeviceAD函数读取设备上的AD数据

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.h 和 ADDoc.cpp, ADThread.h 和 ADThread.cpp)。

[程序]\[阿尔泰测控演示系统]\[PCI8735 32 路 AD 和 16 路开关量卡]\[Microsoft Visual C++]\[高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc::StartDeviceAD()            // 启动线程函数
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Npt (PVOID pThreadPara)   // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)   // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD()          // 终止采集函数
```

第八章 共用函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“PCI8735_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定 PCI 设备寄存器操作基地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户

ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateVBThread	在 VB 环境中建立子线程对象	在 VB 中可实现多线程
TerminateVBThread	终止 VB 的子线程	
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	
ReadFile	请求文件对象读数据到用户空间	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	
ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备
⑤ 各种参数保存和读取函数		
SaveParaInt	保存整型参数到注册表	
LoadParaInt	从注册表中读取整型参数值	
SaveParaString	保存字符参数到注册表	
LoadParaString	从注册表中读取字符参数值	
⑥ 其他函数		
kbhit	探测用户是否有击键动作	
getch	等待并获取用户击键值	
GetLastError	取得驱动函数错误信息	

第二节、PCI 内存映射寄存器操作函数原型说明

♦ **取得指定内存映射寄存器的线性地址和物理地址**

函数原型:

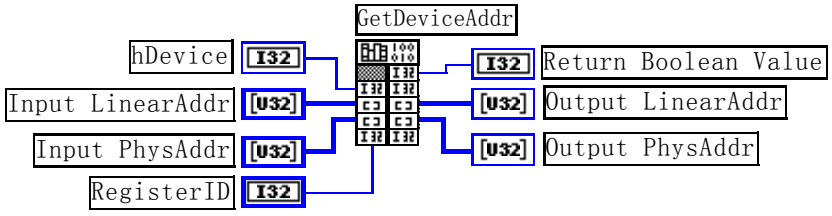
Visual C++:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PULONG LinearAddr,
                    PULONG PhysAddr,
                    int RegisterID = 0)
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCI8735" (ByVal hDevice As Long, _
                                              ByRef LinearAddr As Long, _
                                              ByRef PhysAddr As Long, _
                                              Optional ByVal RegisterID As Integer = 0) As Boolean
```

LabVIEW:



功能：取得 PCI 设备指定的内存映射寄存器的线性地址。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr 指针参数，用于取得的映射寄存器指向的线性地址，RegisterID 指定的寄存器组属于 MEM 模式时该值不应为零，也就是说它可用于 WriteRegisterX 或 ReadRegisterX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。但如果 RegisterID 指定的寄存器组属于 I/O 模式时该值通常为零，您不能通过以上函数访问设备。

PhysAddr 指针参数，用于取得的映射寄存器指向的物理地址，它指明该设备位于系统空间的物理位置。如果由 RegisterID 指定的寄存器组属于 I/O 模式，则可用于 WritePortX 或 ReadPortX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。

RegisterID 指定映射寄存器的 ID 号，其取值范围为[0，5]，通常情况下，用户应使用 0 号映射寄存器，特殊情况下，我们为用户加以申明。本设备的寄存器组 ID 定义如下：

常量名	常量值	功能定义
PCI8735_REG_MEM_CPLD	0x0000	0 号寄存器对应板上控制单元所使用的内存模式基地址(使用 LinearAddr)
PCI8735_REG_IO_CPLD	0x0001	1 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr)

返回值：如果执行成功，则返回TRUE，它表明由RegisterID指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回，否则会返回FALSE，同时还要检查其LinearAddr和PhysAddr是否为 0，若为 0 则依然视为失败。用户可用 GetLastError捕获当前错误码，并加以分析。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr;
hDevice = CreateDevice(0);
if(!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox("取得设备地址失败…");
}
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr    As Long
hDevice = CreateDevice(0)
if Not GetDeviceAddr(hDevice, LinearAddr,    PhysAddr, 0) then
    MsgBox "取得设备地址失败…"
End If
    :
```

♦ **以单字节（即 8 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型:

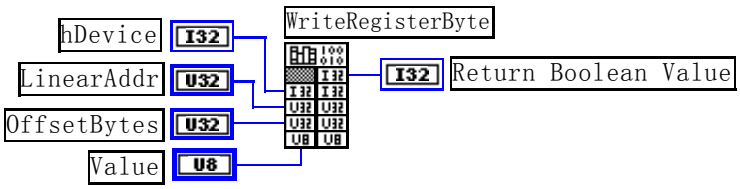
Visual C++:

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI8735" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte ) As Boolean
```

LabVIEW:



功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterByte函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,   &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);   // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

♦ **以双字节（即 16 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型:

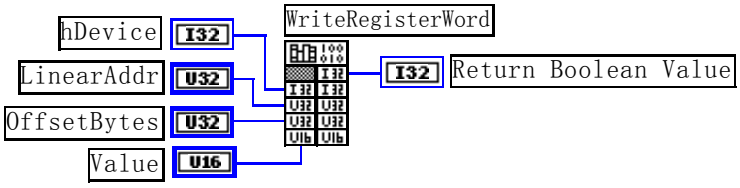
Visual C++:

```
BOOL WriteRegisterWord(HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes,
                       WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI8735" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr　PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes　相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterWord函数所访问的映射寄存器的内存单元。

Value　输出 16 位整型值。

返回值：无。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,    &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);   // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

◆ **以四字节（即 32 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型:

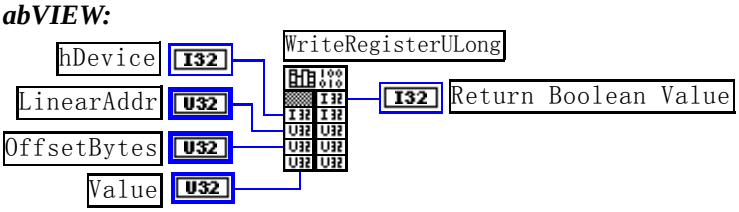
Visual C++:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib "PCI8735" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long)As Boolean
```

abVIEW:



功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox "取得设备地址失败…";
}
OffsetBytes=100;// 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);  // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
    :
```

♦ **以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

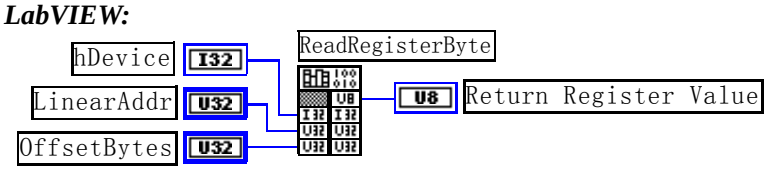
Visual C++:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI8735" (ByVal hDevice As Long, _
                                                 ByVal LinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterByte函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
BYTE   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

- **以双字节（即 16 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

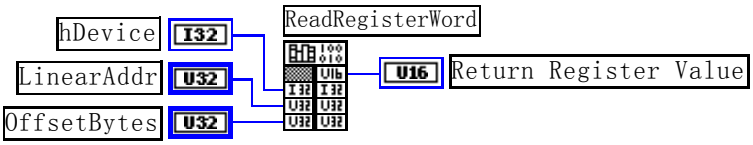
Visual C++:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI8735" (  ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long) As Integer
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterWord函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );   // 释放设备对象
```

：

Visual Basic 程序举例:

```
：
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
：
```

♦ **以四字节（即 32 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

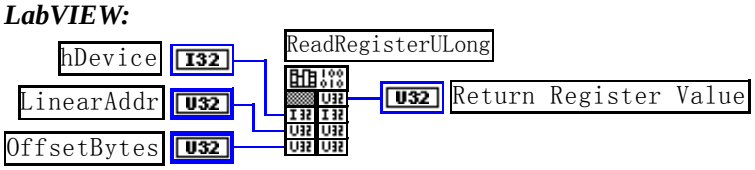
Visual C++:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI8735" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```

LabVIEW:



功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr确定。

OffsetBytes 相对与LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
：
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
ULONG   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);     // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );   // 释放设备对象
：
```

Visual Basic 程序举例:

```
：
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
```

：

第三节、I0 端口读写函数原型说明

注意：若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

- **以单字节(8Bit)方式写 I/O 端口**

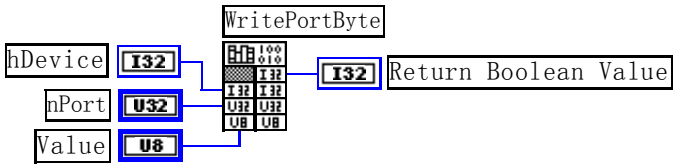
Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    UINT nPort,
                    BYTE Value)
```

Visual Basic:

```
Declare Function WritePortByte Lib "PCI8735" ( ByVal hDevice As Long, _
                                                ByVal nPort As Long, _
                                                ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用 GetLastError捕获当前错误码。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

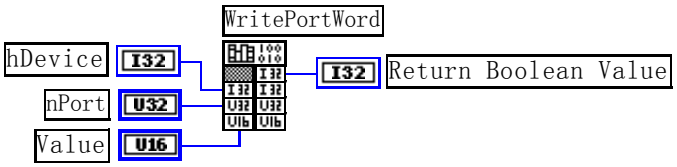
Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    UINT nPort,
                    WORD Value)
```

Visual Basic:

```
Declare Function WritePortWord Lib "PCI8735" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用 GetLastError捕获当前错误码。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

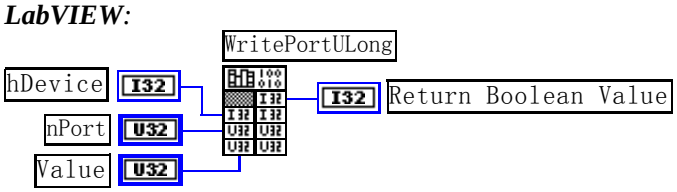
Visual C++:

```
BOOL WritePortULong(HANDLE hDevice,
                    UINT nPort,
                    ULONG Value)
```

Visual Basic:

```
Declare Function WritePortULong Lib "PCI8735" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用 GetLastError捕获当前错误码。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

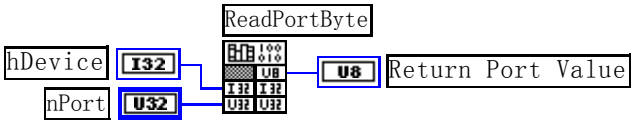
Visual C++:

```
BYTE ReadPortByte( HANDLE hDevice,
                   UINT nPort)
```

Visual Basic:

```
Declare Function ReadPortByte Lib "PCI8735" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

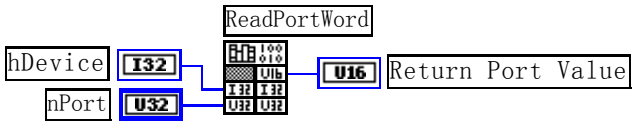
Visual C++:

```
WORD ReadPortWord(HANDLE hDevice,
                  UINT nPort)
```

Visual Basic:

```
Declare Function ReadPortWord Lib "PCI8735" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式读 I/O 端口**

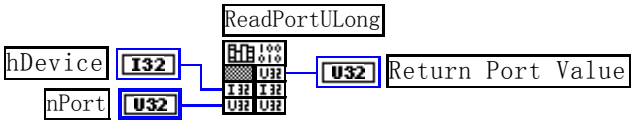
Visual C++:

```
ULONG ReadPortULong(HANDLE hDevice,
                    UINT nPort)
```

Visual Basic:

```
Declare Function ReadPortULong Lib "PCI8735" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long ) As Long
```

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

第四节、线程操作函数原型说明

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

- **在 VB 环境中，创建子线程对象，以实现多线程操作**

函数原型：

Visual C++:

```
BOOL CreateVBThread(HANDLE *hThread,
                    LPTHREAD_START_ROUTINE RoutineAddr)
```

Visual Basic

```
Declare Function CreateVBThread Lib "PCI8735" ( ByRef hThread As Long, _
                                                ByVal RoutineAddr As Long ) As Boolean
```

功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题。通过该函数用户可以很轻松地实现多线程操作。

参数：

hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程、暂停线程以及删除线程等。

RoutineAddr 作为子线程运行的函数的地址，在实际使用时，请用AddressOf关键字取得该子线程函数的地址，再传递给 CreateVBThread函数。

返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用 GetLastError捕获当前错误码。

相关函数： CreateVBThread　TerminateVBThread

注意: RoutineAddr 指向的函数或过程必须放在 VB 的模块文件中，如 PCI8735.Bas 文件中。

Visual Basic 程序举例:

```
' 在模块文件中定义子线程函数(注意参考实例)
Function NewRoutine() As Long        ' 定义子线程函数
```

```
        :                    ' 线程运行代码
NewRoutine = 1   ' 返回成功码
End Function
'-------------------------------------------------------------------
' 在窗体文件中创建子线程
      :
Dim hNewThread As Long
    If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
        MsgBox "创建子线程失败"
        Exit Sub
    End If
    ResumeThread (hNewThread)   '启动新线程          :
     :
```

♦ **在 VB 中，删除子线程对象**

函数原型：

Visual C++:

BOOL TerminateVBThread(HANDLE hThread)

Visual Basic:

Declare Function TerminateVBThread Lib "PCI8735" (ByVal hThread As Long) As Boolean

功能：在VB中删除由 CreateVBThread创建的子线程对象。

参数：hThread 指向需要删除的子线程对象的句柄，它应由 CreateVBThread创建。

返回值：当成功删除子线程对象时，返回TRUE，否则返回FALSE，用户可用 GetLastError捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

Visual Basic 程序举例:

```
      :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
      :
```

♦ **创建内核系统事件**

函数原型：

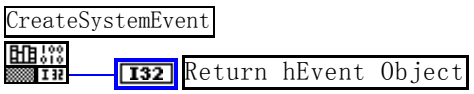
Visual C++:

HANDLE CreateSystemEvent(void)

Visual Basic:

Declare Function CreateSystemEvent Lib " PCI8735 " () As Long

LabVIEW:



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

♦ **释放内核系统事件**

函数原型：

Visual C++:

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function ReleaseSystemEvent Lib " PCI8735 " (ByVal hEvent As Long) As Boolean

LabVIEW:

请参见相关演示程序。

功能：释放系统内核事件对象。

参数：hEvent 被释放的内核事件对象。它应由 CreateSystemEvent成功创建的对象。

返回值：若成功，则返回 TRUE。

- **高效高精度延时**

函数原型：

Visual C++:

```
BOOL DelayTimeUs (HANDLE hDevice,
                  LONG nTimeUs)
```

Visual Basic:

```
Declare Function DelayTimeUs Lib "PCI8735" (ByVal hDevice As Long, _
                                            ByVal nTimeUs As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：微秒级延时函数。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

nTimeUs 时间常数。单位 1 微秒。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用 GetLastError捕获错误码。

第五节、文件对象操作函数原型说明

- **创建文件对象**

函数原型：

Visual C++:

```
HANDLE CreateFileObject ( HANDLE hDevice,
                          LPCTSTR strFileName,
                          int Mode)
```

Visual Basic:

```
Declare Function CreateFileObject Lib "PCI8735" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

LabVIEW:

请参见相关演示程序。

功能：初始化设备文件对象， 以期待 WriteFile 请求准备文件对象进行文件操作。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

strFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：“C:\\PCI8735\\Data.Dat”，在 Basic 中，其语法格式如：“C:\PCI8735\Data.Dat”。

Mode 文件操作方式，所用的文件操作方式控制字定义如下(可通过或指令实现多种方式并操作)：

常量名	常量值	功能定义
PCI8735_modeRead	0x0000	只读文件方式
PCI8735_modeWrite	0x0001	只写文件方式
PCI8735_modeReadWrite	0x0002	既读又写文件方式
PCI8735_modeCreate	0x1000	如果文件不存在可以创建该文件， 如果存在，则重建此文件，且清 0
PCI8735_typeText	0x4000	以文本方式操作文件

返回值：若成功，则返回文件对象句柄。

相关函数： CreateDevice CreateFileObject WriteFile ReadFile ReleaseFile ReleaseDevice

- **通过设备对象，往指定磁盘上写入用户空间的采样数据**

函数原型：

Visual C++:

```
BOOL WriteFile(HANDLE hFileObject,
               PVOID pDataBuffer,
```

```
                    LONG nWriteSizeBytes)
```

Visual Basic:

```
Declare Function WriteFile Lib "PCI8735" ( ByVal hFileObject As Long,_
                                           ByRef pDataBuffer As Byte,_
                                           ByVal nWriteSizeBytes As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：通过向设备对象发送“写磁盘消息”，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由 CreateFileObject函数中的strFileName指定。

参数：

hFileObject 设备对象句柄，它应由 CreateFileObject创建。

pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。

nWriteSizeBytes 告诉设备对象往磁盘上一次写入数据的长度(以字节为单位)。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **通过设备对象,从指定磁盘文件中读采样数据**

函数原型：

Visual C++:

```
BOOL ReadFile(HANDLE hFileObject,
              PVOID pDataBuffer,
              LONG nOffsetBytes,
              LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PCI8735" (  ByVal hFileObject As Long, _
                                           ByRef pDataBuffer As Integer, _
                                           ByVal nOffsetBytes As Long, _
                                           ByVal nReadSizeBytes As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将磁盘数据从指定文件中读入用户内存空间中，其访问方式可由用户在创建文件对象时指定。

参数：

hFileObject 设备对象句柄，它应由 CreateFileObject创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针，可以是用户分配的数组空间。

nOffsetBytes 指定从文件开始端所偏移的读位置。

nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **设置文件偏移位置**

函数原型：

Visual C++:

```
BOOL SetFileOffset (HANDLE hFileObject,
                    LONG nOffsetBytes)
```

Visual Basic:

```
Declare Function SetFileOffset Lib "PCI8735" ( ByVal hFileObject As Long,
                                               ByVal nOffsetBytes As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由 CreateFileObject创建。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。
相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：
Visual C++:
ULONG GetFileLength (HANDLE hFileObject);
Visual Basic:
Declare Function GetFileLength Lib "PCI8735" (ByVal hFileObject As Long) As Long
LabVIEW:
请参考相关演示程序。

功能：取得文件长度。
参数：hFileObject 设备对象句柄，它应由 CreateFileObject创建。
返回值：若成功，则返回>1，否则返回 0，用户可以用 GetLastError捕获错误码。
相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

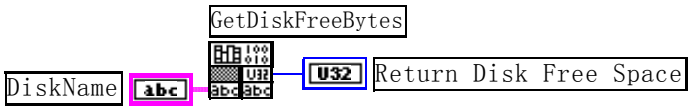
- **释放设备文件对象**

函数原型：
Visual C++:
BOOL ReleaseFile(HANDLE hFileObject)
Visual Basic:
Declare Function ReleaseFile Lib "PCI8735" (ByVal hFileObject As Long) As Boolean
LabVIEW:
请参考相关演示程序。

功能：释放设备文件对象。
参数：hFileObject 设备对象句柄，它应由 CreateFileObject创建。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。
相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得指定磁盘的可用空间**

Visual C++:
ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName)
Visual Basic:
Declare Function GetDiskFreeBytes Lib "PCI8735" (ByVal strDiskName As String) As Currency
LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。
参数：strDiskName 需要访问的盘符，若为 C 盘为"C:\\"， D 盘为"D:\\"，以此类推。
返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用 GetLastError捕获错误码。注意使用 64 位整型变量。

第六节、各种参数保存和读取函数原型说明

- **将整型变量的参数值保存在系统注册表中**

函数原型：
Visual C++:
BOOL SaveParaInt(HANDLE hDevice, LPCTSTR strParaName, int nValue)

Visual Basic:

```
Declare Function SaveParaInt Lib "PCI8735" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nValue As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8735\Device-0\Others。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nValue 整型参数值。它保存在由 strParaName 命名的键项里。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

◆ **将整型变量的参数值从系统注册表中读出**

函数原型：

Visual C++:

```
UINT LoadParaInt( HANDLE hDevice, LPCTSTR strParaName, int nDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaInt Lib "PCI8735" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nDefaultVal As Integer) As Long
```

LabVIEW:

请参考相关演示程序。

功能：将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8735\Device-0\Others。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

返回值：若指定的整型参数项存在，则返回其整型值。否则返回由 nDefaultVal 指定的默认值。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

◆ **将字符变量的参数值保存在系统注册表中**

函数原型：

Visual C++:

```
BOOL SaveParaString ( HANDLE hDevice, LPCTSTR strParaName, LPCTSTR strParaVal)
```

Visual Basic:

```
Declare Function SaveParaString Lib "PCI8735" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8735\Device-0\Others。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 字符参数值。它保存在由 strParaName 命名的键项里。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

- **将字符变量的参数值从系统注册表中读出**

函数原型：

Visual C++:

```
BOOL LoadParaString ( HANDLE hDevice,
                      LPCTSTR strParaName,
                      LPCTSTR strParaVal,
                      LPCTSTR strDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaString Lib "PCI8735" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String,_
                                               ByVal strDefaultVal As String) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8735\Device-0\Others。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDevice创建。

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

strDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastError捕获错误码。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

第七节、其他函数原型说明

- **探测用户是否有按键动作**

函数原型：

Visual C++:

```
BOOL kbhit (void)
```

Visual Basic:

```
Declare Function kbhit Lib "PCI8735" () As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：探测用户是否用键盘按键动作，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若自上次探测过后，若用户有键盘按键动作，则返回 TRUE，否则返回 FALSE。

相关函数： getch kbhit

- **等待按键动作并返回按键值**

函数原型：

Visual C++:

```
char getch (void)
```

Visual Basic:

```
Declare Function getch Lib "PCI8735" () As String
```

LabVIEW:

请参考相关演示程序。

功能：探等待用户键盘按键并以字符方式返回按键值，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若用户没有按键动作，此函数一直不返回，一旦用户有按键动作，便立即返回，且返回其当前按键值(ACII 码)。

相关函数： getch kbhit

◆ **怎样获取驱动函数错误信息**

函数原型：

Visual C++:

```
DWORD GetLastErrorEx (LPCTSTR strFuncName, LPCTSTR strErrorMsg)
```

Visual Basic:

```
Declare Function GetLastErrorEx Lib "PCI8735" (ByVal strFuncName As String,_
                                                ByVal strErrorMsg As String) As Long
```

LabVIEW:

请参考相关演示程序。

功能：将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字串。

参数：

strFuncName 出错函数的名称。注意此函数必须是完整名称，如 AD 初始化函数 PCI8735_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8735_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串。该串为字符数组，其分配空间最好不要小于 256 字节。

返回值：返回错误码。

相关函数： 无。

Visual C++程序举例

```
    :
char strErrorMsg[256];     // 用于返回错误信息字串，要求其空间足够大
DWORD dwErrorCode;
int DeviceLgcID = 0;
hDevice = PCI8735_CreateDevice ( DeviceLgcID ); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
 {
     dwErrorCode = PCI8735_GetLastError(“PCI8735_CreateDevice”, strErrorMsg);
     AfxMessageBox(strErrorMsg); // 以对话框方式显示错误信息
     return;     // 退出该函数
 }
    :
```

Visual Basic 程序举例

```
    :
Dim strErrormsg As String   ' 用于返回错误信息字串，要求其空间足够大
Dim dwErrorCode As Long
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI8735_CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
     dwErrorCode = PCI8735_GetLastError(“PCI8735_CreateDevice”, strErrorMsg)
     MsgBox strErrorMsg   ' 以对话框方式显示错误信息
     Exit Sub      ' 退出该过程
End If
    :
```