PCI8214 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

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 PCI8613_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写规则

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			

以上规则不局限于该产品。

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、ReadDeviceProAD 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，您基本上可以必参考硬件说明书，除非您需要知道板上 D 型插座等管脚分配情况。因为上层函数的命名、参数的命名极其规范。

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其他函数，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，ReadDeviceProAD 函数可以用 hDevice 句柄实现对 AD 数据的采样读取。最后可以通过 ReleaseDevice 将 hDevice 释放掉。

第三节、如何取得 AD 数据

当您有了 hDevice 设备对象句柄后，便可用 InitDeviceAD 函数初始化 AD 部件，关于频率等参数的设置是由这个函数的 pADPara 参数结构体决定的。您只需要对这个 pADPara 参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。接着用 StartDeviceAD 即可启动 AD 部件，开始 AD 采样，GetDevStatusAD 函数以查询 AD 的状态，若状态表明所有数据转换完成，便可用 ReadDeviceProAD 读取 AD 数据。如果要反复采样，则重复前面的过程。当您需要暂停设备时，执行 StopDeviceAD，当您需要关闭 AD 设备时，ReleaseDevice 便可帮您实现（但设备对象 hDevice 依然存在）具体执行流程请看下面的图 2.1.1。

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，最初执行一次 CreateDevice，在结束是就须执行一次 ReleaseDevice。

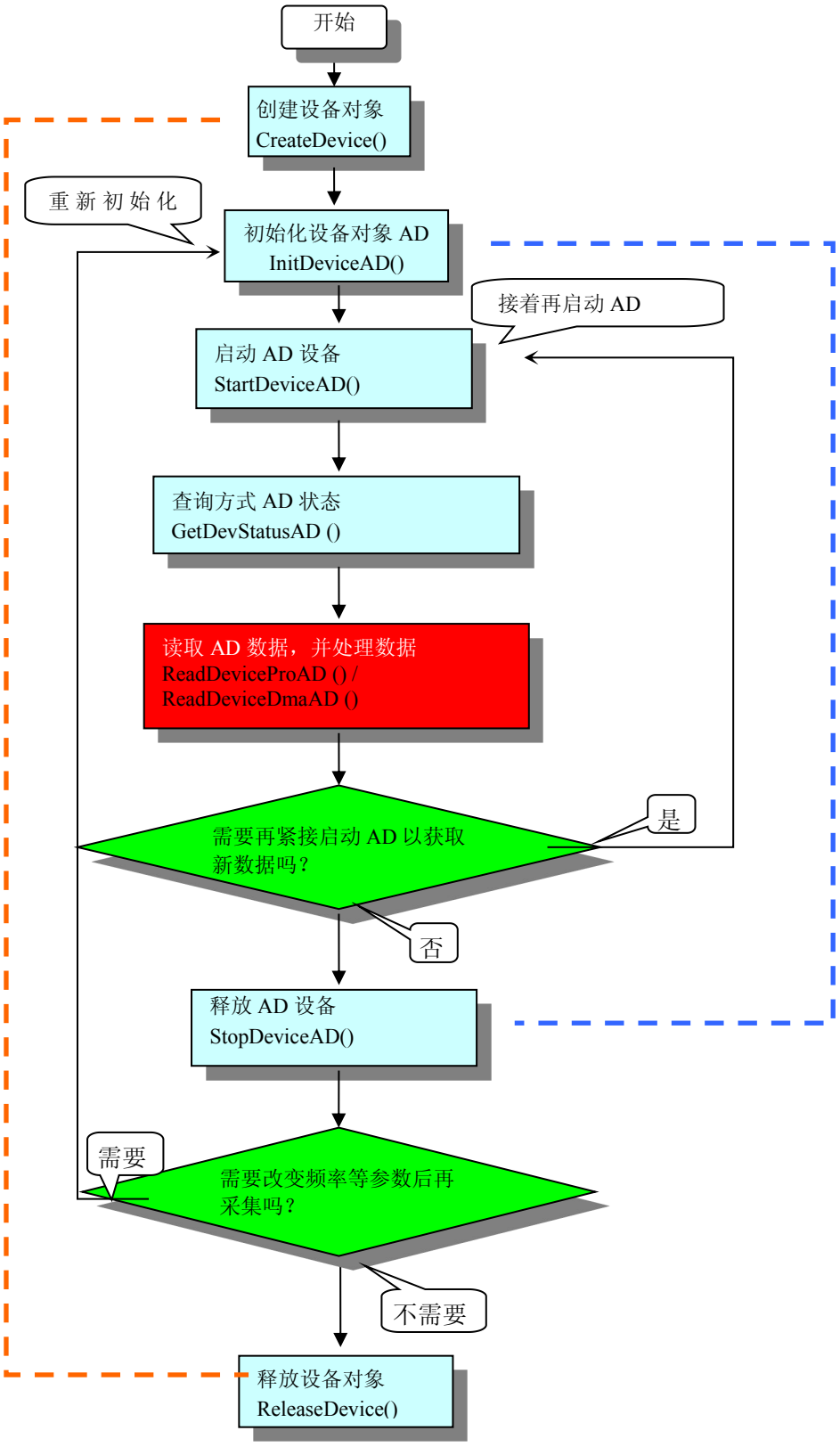


图 2.1.1

第四节、如何实现 DA 的简便输出

当您有了 hDevice 设备对象句柄后，首先用函数实现 DA 的复位操作，然后反复调用 WriteDeviceProDA 函数输出每一个 DA 数据。

第五节、哪些函数对您不是必须的

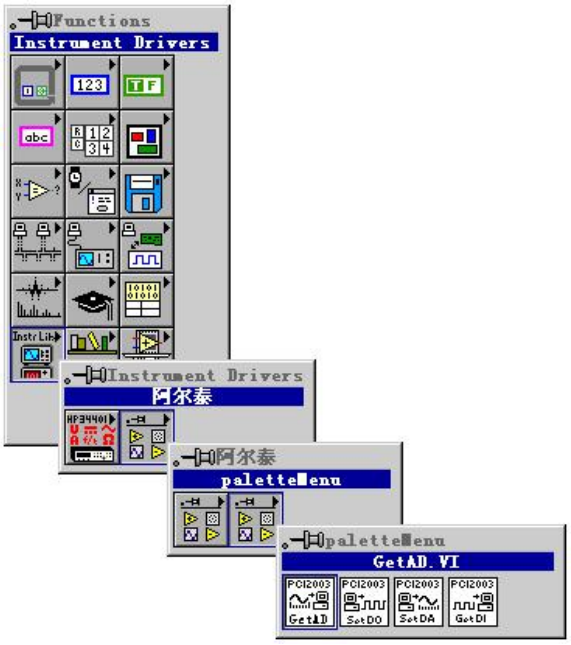
当公共函数如 CreateFileObject, WriteFile, ReadFile 等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问备，那么 GetDeviceAddr， WriteRegisterByte, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong 等函数您可完全不必理会，除非您是作为底层用户管理设备。而 WritePortByte, WritePortWord, WritePortULong, ReadPortByte, ReadPortWord, ReadPortULong 则对 PCI 用户来说，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 Win2000、Win XP 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，您可能在新操作系统中无法继续使用您原有的老设备（除非您自己愿意去编写复杂的硬件驱动程序）。

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，我们则把这一群需要直接跟设备端口打交道的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，他们最希望他们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 ReadDeviceProAD（或 ReadDeviceDmaAD）函数只须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极大的方便和满足了您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先得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。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、设备驱动接口函数列表（每个函数省略了前缀“PCI8214_”）

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	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层
GetDeviceCurrentID	取得指定设备句柄指向的设备 ID 号	上层
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
AD 数据采样操作函数		
InitDeviceAD	初始化 PCI 设备上的 AD 部件准备传输	上层用户
SetDevFreqencyAD	可动态改变 AD 采样频率	上层用户
StartDeviceAD	启动 AD 设备，开始转换	上层用户
SetDevTrigSrcAD	设置触发源	上层用户
GetDevTrigPosAD	取得触发点位置(相对于当前采集的所有数据的起点)	上层用户
ClearDevStatusAD	清除指定标志	上层用户
GetDevStatusAD	取得当前 AD 状态	上层用户
ReleaseDeviceAD	关闭 AD 设备,禁止传输,且释放资源	
StopDeviceAD	暂停 AD 设备	上层用户
AD 数据传输函数		
ReadDeviceDmaAD	以 DMA 方式读取 PCI 设备上的 AD 数据	上层用户
ReadDeviceProAD	以程序方式读取 PCI 设备上的 AD 数据	上层用户
模拟信号频率测量函数		
InitDevicePulse0	初始化(ATR)	上层用户
GetDevPulse0	取得模拟触发源的脉冲宽度	上层用户
InitDevicePulse1	测定 PULSE 信号输入的频率	上层用户
GetDevPulse1	取得状态(PULSE)	上层用户
数字 I/O 输入输出函数		
GetDeviceDI	取得开关量状态	上层用户
SetDeviceDO	输出开关量状态	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		上层用户
程序方式 DA 输出函数		
WriteDeviceProDA	单点输出 DA 数据	上层用户
TransVoltToLsbDA	将电压值转换成相应码值	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，将 PCI8214.h 和 PCI8214.lib 文件从 Visual C++的源程序目录下的任意一个子目录下复制到您的源程序目录下（若有 Advanced 高级源程序目录，则最好选择它），然后在您的源程序中包含如下语句（若想在整个工程的所有源代码文件中使用本驱动，请您最好在 StdAfx.h 全局头文件中包含如下语句）：

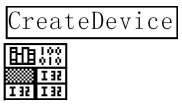
#include “PCI8214.H”

那么对于导入库 PCI8214.lib 文件您则可以不必再加入您的工程，因为 PCI8214.h 头文件已帮助自动完成了。

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。



一、在 LabVIEW 中打开 PCI8214.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续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竖线为中心，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 ReadDeviceProAD_NotEmpty 接口单元，左边为设备对象句柄、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：

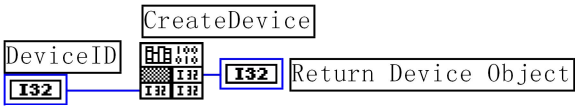
Visual C++：

HANDLE CreateDevice (int DeviceLgcID=0)

Visual Basic：

Declare Function CreateDevice Lib “PCI8214”(Optional ByVal DeviceLgcID As long = 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 PCI8214 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 PCI8214 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 CreateDeviceEx 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　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
 hDevice=CreateDevice ( 0 ); // 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
```

```
hDevice = CreateDevice ( 0 )  ' 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

Else
  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 “PCI8214”(Optional ByVal DevicePhysID As long = 0) As long

LabVIEW：

请参考相关演示程序。

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：

DevicePhysID 物理设备 ID(Physic Device Identifier)标识号。由 CreateDevice 函数的 DevieLgcID 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 256 个通道（64*4），其通道序列为 0-255，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 0-63 通道上，B 卡位于 63-127 通道上，C 卡位于 128-191 通道上，而 D 卡则位于 192-255 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 0-255 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。

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 PCI8214 设备的总数量**

函数原型：

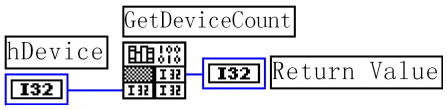
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "PCI8214" (ByVal hDevice As Long) As Long

LabVIEW：



功能：取得 PCI8214 设备的数量。

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

返回值：返回系统中 PCI8214 的数量。

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得该设备当前逻辑 ID 和物理 ID**

函数原型：

Visual C++:

int???(类型 BOOL) GetDeviceCurrentID (HANDLE hDevice，
　　PLONG DeviceLgcID，

PLONG DevicePhysID)

Visual Basic:

Declare Function GetDeviceCurrentID Lib "PCI8214" (ByVal hDevice As Long,_
ByRef DeviceLgcID As Long,_
ByRef DevicePhysIDAs Long) As Long???

LabVIEW：

请参考相关演示程序。

功能：取得 PCI8214 设备的数量。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 7]。

DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 7]，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 PCI8214 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib "PCI8214" (ByVal hDevice As Long) As Boolean

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI8214 的硬件配置信息。

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

返回值：若成功，则弹出对话框控件列表所有 PCI8214 设备的配置情况。

相关函数： CreateDevice ReleaseDevice

- **释放设备对象所占的系统资源及设备对象**

函数原型：

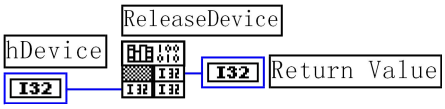
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "PCI8214" (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 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

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 数据采样操作函数原型说明

- **初始化设备对象**

函数原型：

Visual C++:

```
BOOL InitDeviceAD( HANDLE hDevice,
                   HANDLE hEventInt ???参数名 hDmaEvent,
                   PPCI8214_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceAD Lib "PCI8214" (ByVal hDevice As Long, _
                                             ByVal hEventInt??? As Long,_
                                             ByRef pADPara As PPCI8214_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,为设备操作就绪有关工作,如预置 AD 采集通道,采样频率等。但它并不启动 AD 设备，若要启动 AD 设备，须在调用此函数之后再调用 StartDeviceAD。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

hEventInt??? 中断事件句柄。当板载 RAM 发生乒乓切换时设备对象会立即触发该事件。用户可调用 WIN32 API 函数 WaitForSingleObject 等待该事件的发生以同步 AD 数据的读操作。该参数在 Win2K 以上系统有效。

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,如采样频率等。关于具体操作请参考第四章《硬件参数结构》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　SetDevFreqencyAD
StartDeviceAD　GetDevStatusAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceAD　ReleaseDevice

♦ **动态改变采样频率**

函数原型：

Visual C++:

```
BOOL SetDevFreqencyAD ( HANDLE hDevice,   LONG nFrequency)
```

Visual Basic:

```
Declare Function SetDevFreqencyAD Lib "PCI8214" (ByVal hDevice As Long, _
                                                 ByVal nFrequency As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：在 AD 采样过程中，可动态改变采样频率。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

nFrequency 采样频率，单位 Hz。当 ADMode 设为同步采样时，其取值范围由 1Hz 至 2MHz，当 ADMode 设为异步采样时，其取值范围由 1Hz 至 10MHz。

返回值：如果调用成功,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　SetDevFreqencyAD
StartDeviceAD　GetDevStatusAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceAD　ReleaseDevice

♦ **启动 AD 设备**

函数原型：

Visual C++:

```
BOOL StartDeviceAD ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StartDeviceAD Lib "PCI8214" (ByVal hDevice As Long )
```

LabVIEW：

请参考相关演示程序。

功能：启动 AD 设备，它必须在调用 InitDeviceAD 后才能调用此函数。调用该函数后它立即启动。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

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。

相关函数： CreateDevice InitDeviceAD SetDevFreqencyAD
StartDeviceAD GetDevStatusAD ReadDeviceProAD
ReadDeviceDmaAD StopDeviceAD ReleaseDevice

♦ 取得 AD 的状态标志

函数原型：

Visual C++:

```
BOOL GetDevStatusAD ( HANDLE hDevice, PPCI8214_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "PCI8214" (ByVal hDevice As Long,_
                                               ByRef pADStatus As PCI8214_STATUS_AD) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：

一旦用户使用 StartDeviceAD 后，应立即用此函数查询 AD 状态去同步 RAM 读操作。ADStatus. bRamSwitch 等于 TRUE 时，板载 RAM 已发生乒乓切换，用户便可调用 ReadDeviceProAD 或 ReadDeviceDmaAD 函数读取 RAM 中的 AD 数据。由于每个通道具有 64K 点（字）的存储器深度，因此当调用 StartDeviceAD 启动 AD 时，写满 RAM 需要一定时间，所以当该函数取回的 ADStatus. bRamSwitch 值为 FALSE，则用户应继续等待，直到等于 TRUE 时才能读取数据。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pADStatus 设备状态参数结构，它返回设备当前的各种状态，如板载 RAM 是否发生切换、重写、触发点是否产生等信息。关于具体操作请参考第四章《硬件参数结构》中的《AD 硬件参数结构》。

返回值：若 AD 成功取标状态，则返回 TRUE，否则返回 FALSE。注意在 Win2K 以上系统中，在循环轮询期间，尽可能使用 WaitForSingleObject 来等待切换中断事件，且使用 DMA 方式得到更高的数据采样率和系统的整体性能。

相关函数： CreateDevice InitDeviceAD SetDevFreqencyAD
StartDeviceAD GetDevStatusAD ReadDeviceProAD
ReadDeviceDmaAD StopDeviceAD ReleaseDevice

♦ 取得触发点的位置

函数原型：

Visual C++:

```
BOOL GetDevTrigPosAD ( HANDLE hDevice, PLONGLONG nTriggerPos)
```

Visual Basic:

```
Declare Function GetDevTrigPosAD Lib "PCI8214" (ByVal hDevice As Long,_
                                                ByRef nTriggerPos As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：取得触发点的位置，该位置相对于第一个点的距离长度。若转换一个 AD 数据则该计数加 1，直到触发点产生，则计数保持不变。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

nTriggerPos 返回触发点位置，尽可能使用 64 位整型数据类型。

返回值：如果调用成功,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 SetDevFreqencyAD
StartDeviceAD GetDevStatusAD ReadDeviceProAD
ReadDeviceDmaAD StopDeviceAD ReleaseDevice

♦ 清除各状态标志

函数原型：

Visual C++:

```
BOOL ClearDevStatusAD ( HANDLE hDevice,
                        BOOL bClrRamSwitch = FALSE，
```

```
                                    BOOL bClrOverflow = FALSE，
                                    BOOL bClrTrigFlag = FALSE)
```

Visual Basic:

```
Declare Function ClearDevStatusAD Lib "PCI8214" ( ByVal hDevice As Long,_
                                                   bClrRamSwitch As Boolean,_
                                                   bClrOverflow As Boolean,_
                                                   bClrTrigFlag As Boolean,_
                                                   ) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：清除各状态标志。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

bClrRamSwitch 是否清除 RAM 切换标志，=TRUE 表示清除， = FALSE 表示不清除。

bClrOverflow 是否清除 RAM 重写标志，=TRUE 表示清除， = FALSE 表示不清除。

bClrTrigFlag 是否清除触发点标志，=TRUE 表示清除， = FALSE 表示不清除。

返回值：如果调用成功,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	SetDevFreqencyAD
StartDeviceAD	GetDevStatusAD	ReadDeviceProAD
ReadDeviceDmaAD	StopDeviceAD	ReleaseDevice

- **暂停 AD 设备**

函数原型：

Visual C++:

```
BOOL StopDeviceAD ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StopDeviceAD Lib "PCI8214" (ByVal hDevice As Long )
```

LabVIEW：

请参考相关演示程序。

功能：

暂停 AD 设备。它必须在调用 StartDeviceAD 后才能调用此函数。该函数除了停止 AD 设备不再转换以外，不改变设备的其他任何状态。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

返回值：如果调用成功,则返回 TRUE,且 AD 立刻停止转换，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	SetDevFreqencyAD
StartDeviceAD	GetDevStatusAD	ReadDeviceProAD
ReadDeviceDmaAD	StopDeviceAD	ReleaseDevice

- **采样和和传输函数一般调用顺序**

① CreateDevice
② InitDeviceAD
③ StartDeviceAD
④ GetDevStatusAD（WaitForSingleObject()）（循环查询 AD 状态）
⑤ ReadDeviceProAD (ReadDeviceDmaAD)
⑥ StopDeviceAD
⑦ ReleaseDevice

关于过程的图形说明请参考第二章《使用纲要》的第三节《如何取得 AD 数据》。

第四节、AD 数据传输函数原型说明

- **以程序方式将 PCI 设备上 RAM 中的 AD 数据传输至主机**

函数原型：

Visual C++:

```
BOOL ReadDeviceProAD (  HANDLE hDevice,
                        PSHORT ADBuffer,??? USHORT ADBuffer[]
                        LONG nReadOffsetWords,
                        LONG nReadSizeWords,
                        PLONG nRetSizeWords,
                        LONG???类型 int nADChannel)
```

Visual Basic:

```
Declare Function ReadDeviceProAD Lib "PCI8214" ( _
                        ByVal hDevice As Long, _
                        ByRef ADBuffer As Integer,_
                        ByVal nReadOffsetWords As Long,_
                        ByVal nReadSizeWords As Long,_
                        ByRef nRetSizeWords As Long,_
                        ByVal nADChannel As Long) As Long
```

LabVIEW：

请参考相关演示程序。

功能：用户随时可以用此函数读取设备上 RAM 中的 AD 数据。一般用户使用 GetDevStatusAD 查询到完成标志后，才用此函数读取设备上的 AD 数据。(它用软件方式即 CPU 指令读取 RAM 中的 AD 数据)

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

ADBuffer 接受 AD 数据的用户缓冲区地址，它可以是一个 16Bit 整型数组，也可以是由其他方式分配的 16Bit 整型缓冲区。关于如何将这些 AD 数据转换成相应的电压值，请参考请《数据格式转换与排列规则》。

nReadOffsetWords 指定当前从物理缓冲区即板上 RAM 中的起始位置(以点/字为单位)。此参数不能超过 RAM 的最大长度(字/点)。

nReadSizeWords 指定一次从物理缓冲区 RAM 中由 nReadOffsetWords 指定偏移位置开始读取的长度。注意此参数的值与 nReadOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。此参数值不能大于 ADBuffer 指定的缓冲区的长度。

nRetSizeWords 返回当前读操作实际实现的数据长度。它表明该函数调用后，在 ADBuffer 中有多少数据是有效的。

nADChannel 指定 AD 通道号（0-3）。

返回值：如果调用成功,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, SetDevFreqencyAD, StartDeviceAD, GetDevStatusAD, ReadDeviceProAD, ReadDeviceDmaAD, StopDeviceAD, ReleaseDevice

♦ **以 DMA 直接内存存取方式将 PCI 设备上 RAM 中的 AD 数据传输至主机**

函数原型：

Visual C++:

```
LONG ReadDeviceDmaAD ( HANDLE hDevice,
                        PSHORT ADBuffer,??? USHORT ADBuffer[]
                        LONG nReadOffsetWords,
                        LONG nReadSizeWords,
                        PLONG nRetSizeWords,
                        LONG???类型 int nADChannel)
```

Visual Basic:

```
Declare Function ReadDeviceDmaAD Lib "PCI8214" ( _
                        ByVal hDevice As Long, _
                        ByRef ADBuffer As Integer,_
                        ByVal nReadOffsetWords As Long,_
                        ByVal nReadSizeWords As Long,_
                        ByRef nRetSizeWords As Long,_
                        ByVal nADChannel As Long) As Long
```

LabVIEW：

请参考相关演示程序。

功能：

用户随时可以用此函数读取设备上 RAM 中的 AD 数据。一般用户使用 GetDevStatusAD 查询到完成标志后，才

用此函数读取设备上的 AD 数据。(它用软件方式即 CPU 指令读取 RAM 中的 AD 数据)

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

ADBuffer 接受 AD 数据的用户缓冲区地址，它可以是一个 16Bit 整型数组，也可以是由其他方式分配的 16Bit 整型缓冲区。关于如何将这些 AD 数据转换成相应的电压值，请参考请《数据格式转换与排列规则》。

nReadOffsetWords 指定当前从物理缓冲区即板上 RAM 中的起始位置(以点/字为单位)。此参数不能超过 RAM 的最大长度(字/点)。

nReadSizeWords 指定一次从物理缓冲区 RAM 中由 nReadOffsetWords 指定偏移位置开始读取的长度。注意此参数的值与 nReadOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。此参数值不能大于 ADBuffer 指定的缓冲区的长度。

nRetSizeWords 返回当前读操作实际实现的数据长度。它表明该函数调用后，在 ADBuffer 中有多少数据是有效的。

nADChannel 指定 AD 通道号（0-3）。

返回值：如果调用成功,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。

该方式是最高效的方式，但只在 Win2K 以上系统实现。

相关函数： CreateDevice　InitDeviceAD　SetDevFreqencyAD
StartDeviceAD　GetDevStatusAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceAD　ReleaseDevice

第五节、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, PPCI8214_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PCI8214" (ByVal hDevice As Long, _
                                           pADPara As PCI8214_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 创建。

pADPara 属于 PPCI8214_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 PPCI8214_PARA_AD 请参考 PCI8214.h 或 PCI8214.Bas 或 PCI8214.Pas 函数原型定义文件，也可参考本文第四章《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

◆ 写设备硬件参数函数到 Windows 系统中

函数原型：

Viusal C++:

```
BOOL SaveParaAD (HANDLE hDevice, PPCI8214_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI8214" (ByVal hDevice As Long, _
                                           pADPara As PCI8214_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 创建。

pADPara 设备硬件参数，关于 PCI8214_PARA_AD 的详细介绍请参考 PCI8214.h 或 PCI8214.Bas 或 PCI8214.Pas

函数原型定义文件，也可参考本文第四章《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

◆ **将硬件参数结构体值复位为出厂默认值**

函数原型：

Viusal C++:

```
BOOL ResetParaAD (HANDLE hDevice, PPCI8214_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "PCI8214" (ByVal hDevice As Long, _
                                            pADPara As PCI8214_PARA_AD) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责将硬件参数的值复位至出厂默认值，不仅会将 pADPara 指向的结构体成员值更新为默认值，同时会将系统中保存的参数更新为默认值。这些默认值在产品驱动第一次被安装时会出现。而且这些默认值的设定是充分的考虑到用户的实际情况，确保用户不用外部任何条件，只要开始采集数据，即可获得相应的结果。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pADPara 设备硬件参数，关于 PCI8214_PARA_AD 的详细介绍请参考 PCI8214.h 或 PCI8214.Bas 或 PCI8214.Pas 函数原型定义文件，也可参考本文第四章《硬件参数结构》关于该结构的有关说明。调用此函数后，该参数指向的结构体成员将被复位至默认值。

返回值：若成功，返回 TRUE，它表明已成功将系统中的 AD 参数复位至默认值，同时更新了 pADPara 指向的结构体。否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

第六节、DA 程序方式输出函数原型说明

（注：函数中的“Pro”字符是 Program 的缩字，标明以程序查询方式）

◆ **往设备上的 DA 部件输出 DA 数据**

函数原型：

Visual C++:

```
BOOL WriteDeviceProDA (HANDLE hDevice, WORD DAData??? SHORT nDALsb）
```

Visual Basic:

```
Declare Function WriteDeviceProDA   Lib "PCI8214" (ByVal hDevice As Long, _
                                                   ByVal DAData As Integer ) As Boolean
```

功能：在 DA 部件上输出一个点的 DA 数据，它决定了模拟触发时内部的门坎电压。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

DAData DA 数据，它存放的是 DA 数据的原码 Lsb，关于如何将电压数据转换成相应 Lsb 原码，请参考第五章《数据格式转换与排列规则》。

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 WriteDeviceProDA ReleaseDevice

◆ 函数一般调用顺序

① CreateDevice

② WriteDeviceProDA

③ ReleaseDevice

注明：用户可以反复执行第②步，以实现高速连续 DA 输出。

第七节、DIO 数字量输入输出开关量操作函数原型说明

◆ **开关量输入**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI8214" ( 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 CreateDeviceEx 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 "PCI8214" (ByVal hDevice As Long, _
                                            ByVal bDOSts(0 to 15) As Byte) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer;
                      bDOSts : Pointer) : Boolean;
                      StdCall; External 'PCI8214' Name ' SetDeviceDO ';
```

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 CreateDeviceEx 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

第一节、AD 硬件参数结构（PCI8214_PARA_AD）

Visual C++:

```
typedef struct _PCI8214_PARA_AD
{
    LONG Gains[4];              // AD 程控增益,分别控制四个通道
    LONG SwitchLenCtrl;         //  存储器切换长度控制
```

```
    LONG ADMode;            // 同步/异步方式选择
    LONG Frequency;         // 采集频率 Hz(大于 3???0.01Hz)
    LONG TriggerMode;       // 触发模式(软件内触发和硬件外触发)???
    LONG TriggerSource;     // 触发源信号选择(AI0, AI1, AI2, AI3, ATR)
    LONG TriggerType;       // 脉冲触发/边沿触发
    LONG TriggerDir;        // 正向/负向触发选择
    LONG bSetTrigLevel;     // 是否置触发电平(=TRUE:表示设置触发电平，=FALSE 表示不设置触发电平)???
    LONG TrigLevelVolt;     // 触发电平,单位 mV(-10000- +10000mV)???
    LONG ClockSource;       // 内/外时钟源选择
    LONG bClockOutput;      // 时钟输出控制, =TRUE:允许本卡上的自带时钟向外输出, =FALSE:禁止本卡上的自带时钟向外输出???
} PCI8214_PARA_AD,  *PPCI8214_PARA_AD;
```

Visual Basic：

```
Type PCI8214_PARA_AD
    Gains (0 to 3) As Long      ' AD 程控增益,分别控制四个通道
    SwitchLenCtrl As Long;      ' 存储器切换长度控制
    ADMode As Long              ' 同步/异步方式选择
    Frequency As Long           ' 采集频率 Hz(大于 3???0.01Hz)
    TriggerMode As Long         ' 触发模式(软件内触发和硬件外触发)???
    TriggerSource As Long       ' 触发源信号选择(AI0, AI1, AI2, AI3, ATR)
    TriggerType As Long         ' 脉冲触发/边沿触发
    TriggerDir As Long          ' 正向/负向触发选择
    bSetTrigLevel As Long       ' 是否置触发电平(=TRUE:表示设置触发电平,=FALSE 表示不设置触发电平)???
    TrigLevelVolt As Long       ' 触发电平,单位 mV(-10000- +10000mV)???
    ClockSource As Long         ' 内/外时钟源选择
    bClockOutput As Long        ' 时钟输出控制, =TRUE:允许本卡上的自带时钟向外输出, =FALSE:禁止本卡上的自带时钟向外输出???
End Type
```

LabVIEW:

请参考相关演示程序。

首先请您关注一下这个结构与前面 ISA 总线部分中的两个结构 PARA、PARAEX 比较，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，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 InitDeviceProAD 或 InitDeviceIntAD 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。

Gains[4] 程控增益，它表示输入的模拟信号被放大多少倍后才被采样。取值如下表：

常量名	常量值	功能定义
PCI8214_GAINS_1MULT	0x00	1 倍增益(使用 AD8250 或 AD8251 放大器)
PCI8214_GAINS_2MULT	0x01	2 倍增益(使用 AD8250 或 AD8251 放大器)
PCI8214_GAINS_5MULT	0x02	5 倍增益(使用 AD8250 放大器)
PCI8214_GAINS_10MULT	0x03	10 倍增益(使用 AD8250 放大器
PCI8214_GAINS_2MULT	0x01	2 倍增益(使用 AD8251 放大器)
PCI8214_GAINS_4MULT	0x02	4 倍增益(使用 AD8251 放大器)
PCI8214_GAINS_8MULT	0x03	8 倍增益(使用 AD8251 放大器)

SwitchLenCtrl 存储器切换长度控制，是指在 RAM 乒乓切换机制中，设备每采集多少数据点数便切换到另一片 RAM 存储器中继续采集，取值如下表：

常量名	常量值	功能定义
PCI8214_SWITCH_LEN_64K	0x00	每采集 64K 点切换一次

PCI8214_SWITCH_LEN_32K	0x01	每采集 32K 点切换一次
PCI8214_SWITCH_LEN_16K	0x02	每采集 16K 点切换一次
PCI8214_SWITCH_LEN_8K	0x03	每采集 8K 点切换一次
PCI8214_SWITCH_LEN_4K	0x04	每采集 4K 点切换一次
PCI8214_SWITCH_LEN_2K	0x05	每采集 2K 点切换一次
PCI8214_SWITCH_LEN_1K	0x06	每采集 1K 点切换一次

ADMode AD 采集方式，取值如下表：

常量名	常量值	功能定义
PCI8214_ADMODE_SYNC	0x00	同步方式
PCI8214_ADMODE_ASYN	0x01	异步方式

当为同步方式时，四个 AD 芯片按照相同的频率（由 Frequency 参数设定）同时转换各自的所选择的模拟信号源，它们在时序上是完全同步的（即不存在相位差）。而异步方式则是四个 AD 芯片错位轮流转换同一模拟信号源，以实现可高于每个 AD 芯片的最高转换率的采样频率（最高可达四倍于 AD 芯片的采样率即 8MHz 以上）。

需要注意的是：同步方式时，四个芯片转换的 AD 数据是独立的（它们表示各四个对应通道的数据），而异步方式时，则四个芯片转换的 AD 数据则需要软件依次合成一个相应通道的数据。合成方法为：

数据索引号	0	1	2	3	4	5	6	7	8	9	10	11	……
芯片编号	0	1	2	3	0	1	2	3	0	1	2	3	……

其合成的工作要由用户完成。

Frequency　AD 采样率(Hz)，AD 采样频率的实际取值是 ADMode 参数所选择的采集方式。当为同步采集时，则其取值范围为[3Hz, 400KHz]，而为异步采集时，则取值范围为[3Hz, 1.6MHz]。

TriggerMode AD 触发模式，其的取值如下表：???

常量名	常量值	功能定义
PCI8214_TRIGMODE_SOFT	0x0000	内触发方式
PCI8214_TRIGMODE_POST	0x0001	外触发方式

TriggerSource AD 触发源选择，其选项值如下表：

常量名	常量值	功能定义
PCI8214_TRIGSRC_AI0	0x0000	选择 AI0 作为触发源
PCI8214_TRIGSRC_AI1	0x0001	选择 AI1 作为触发源
PCI8214_TRIGSRC_AI2	0x0002	选择 AI2 作为触发源
PCI8214_TRIGSRC_AI3	0x0003	选择 AI3 作为触发源
PCI8214_TRIGSRC_ATR	0x0004	选择 ATR 作为触发源

TriggerType AD 外触发方式类型。选项值如下表：

常量名	常量值	功能定义
PCI8214_TRIGTYPE_EDGE	0x0000	边沿触发
PCI8214_TRIGTYPE_PULSE	0x0001	脉冲触发(电平方式),适用于采集馒头波信号

TriggerDir AD 外触发方式使用信号方向。选项值如下表：

常量名	常量值	功能定义
PCI8214_TRIGDIR_NEGATIVE	0x0000	下降沿/低脉冲外触发(属于负向)
PCI8214_TRIGDIR_POSITIVE	0x0001	上升沿/高脉冲外触发(属于正向)
PCI8214_TRIGDIR_POSIT_NEGAT	0x0002	正负向触发(高/低脉冲或上升/下降沿触发)

当 TriggerType= PCI8214_TRIGTYPE_EDGE **时（此种方式下，要求触发信号相对于门坎电压必须有上下跳变）：**

当 TriggerDir= PCI8214_TRIGDIR_NEGATIVE 时，表示外部触发信号须由大于 AO0 变成小于 AO0 时产生触发（即下降沿触发）。一旦触发产生后，其后续的触发信号变化均无效。

当 TriggerDir= PCI8214_TRIGDIR_POSITIVE 时，表示外部触发信号由小于 AO0 变成大于 AO0 时触发产生(即上升沿触发)。一旦触发产生后，其后续的触发信号变化均无效。

当 TriggerDir= PCI8214_TRIGDIR_POSIT_NEGAT 时，表示外部触发信号由小于 AO0 变成大于 AO0 以及外部触发信号由大于 AO0 变成小于 AO0 时均产生触发((即上下沿均触发))。一旦触发产生后，其后续的触发信号变化均无

效。

当 TriggerType= PCI8214_TRIGTYPE_PULSE 时（此种方式下，不要求触发信号必须有大跳变，而是相对稳定）：

当 TriggerDir=PCI8214_TRIGDIR_NEGATIVE 时，表示外部触发信号若小于触发电平(TriggerLevelLsb)，则触发产生，AD 开始工作，但一旦触发信号大于触发电平时，则 AD 自动停止工作，直到触发信号再次小于触发电平时 AD 便会又自动开始工作，即只采集触发电平下方的波形。

当 TriggerlDir= PCI8214_TRIGDIR_POSITIVE 时，表示外部触发信号若大于触发电平(TriggerLevelLsb)，则触发产生，AD 开始工作，但一旦触发信号小于触发电平时，则 AD 自动停止工作，直到触发信号再次大于触发电平时 AD 便会又自动开始工作，即只采集触发电平上方的波形。

当 TriggerDir= PCI8214_POSIT_NEGAT_DIR 时，不管触发信号（ATR）大于还是小于触发电平均触发，AD 开始工作，此种情况相当于没有使用内触发方式，这里只是为了和边沿触发功能对齐。

在电平触发模式下，当选择外触发或需要实现触发点时，请用 WriteDeviceProDA 输出阈值电压，然后从外接管脚输入触发信号，与阈值电压相比较，上述任意一个条件成立时就触发。

bSetTrigLevel 是否重置触发电平，＝TRUE：表示重新设置触发电压 AO0。

TrigLevelVolt 触发电平取值。它的取值范围为[-10000，+10000]，单位为 mV。

ClockSource AD 外时钟选择：

常量名	常量值	功能定义
PCI8214_CLOCKSRC_IN	0x0000	内部时钟定时触发
PCI8214_CLOCKSRC_OUT	0x0001	外部时钟定时触发

bClockOutput AD 允许时钟输出。

常量名	常量值	功能定义
PCI8214_CLOCKOUT_ENABLE	0x0000	允许内部时钟向外输出
PCI8214_CLOCKOUT_DISABLE	0x0001	禁止内部时钟向外输出

第二节、AD 状态参数结构（PCI8214_STATUS_AD）

Visual C++：

```
typedef struct _PCI8214_STATUS_AD
{
    LONG bRamSwitch;   // 板载 RAM 是否切换，=TRUE 表示切换， =FALSE 表示未切换
    LONG bOverflow;    // 板载 RAM 是否发生重写，=TRUE 表示已重写， =FALSE 表示未重写
    LONG bTrigFlag??? (bTriggerFlag);      // 触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    LONGLONG nTriggerPos;   // 触发点位置???(类型)
    LONG bConverting;   // AD 是否正在转换， =TRUE:表示正在转换， =FALS 表示转换完成
    LONG nCurRamNum;        // 当前可读取的 RAM 编号，取值为[0, 1]
} PCI8214_STATUS_AD, *PPCI8214_STATUS_AD;
```

Visual Basic：

```
Type PCI8214_STATUS_AD
    bRamSwitch As Long    ' 板载 RAM 是否切换，=TRUE 表示切换， =FALSE 表示未切换
    bOverflow As Long     ' 板载 RAM 是否发生重写，=TRUE 表示已重写， =FALSE 表示未重写
    bTrigFlag??? (bTriggerFlag) As Long      ' 触发标志是否有效,=TRUE 表示触点标有效, = FALSE 表示无效（即触发点未到）
    nTriggerPos As LONGLONG;     // 触发点位置??? (类型)
    bConverting As Long    ' AD 是否正在转换， =TRUE:表示正在转换， =FALS 表示转换完成
    nCurRamNum As Long      ' 当前可读取的 RAM 编号，取值为[0, 1]
End Type
```

LabVIEW:

请参考相关演示程序。

此结构主要用于 GetDevStatusAD 函数返回设备各状态。

bRamSwitch RAM 切换标志。由于板载 RAM 是单口 RAM，读和写操作不能同时进行，因此将其分割成两块

RAM0、RAM1， 当 AD 部件写 RAM0 时，主机便可以读 RAM1，当 AD 部件写 RAM1 时，主机便可以读 RAM0，因此该切换标志=TRUE 时表示 AD 部件已写满了某一块 RAM，正在写另一块 RAM，此时，主机便可能读取前一块 RAM 的数据。若该标志=FALSE，表示未发生切换，主机则继续等待。当标志等于 TRUE 时，主机须立即读走数据，并且尽可能快地调用 ClearDevStatusAD 函数清除此标志，否则，当 AD 转回来时，发现此标志还等于 TRUE，则会认为已发生重写错误。

bOverflow 当 RAM 的数据未及时读走，且没有及时清除 RAM 切换标志 bRamSwitch，则该状态会变成 TRUE，表示 RAM 发生重写错误。

bTrigFlag??? (bTriggerFlag) 表示是否产生了触发点。等于 TRUE，表示产生了触发点，等于 FALSE 表示没有。

nTriggerPos 触发点位置。

bConverting 表示 AD 是否还在转换，等于 TRUE， 表示还在转换，等于 FALSE 表示停止转换。

nCurRamNum 表示当前可读取的 RAM 块号。取值为[0, 1]。

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

在换算过程中弄清模板精度（即 Bit 位数）是很关键的，因为它决定了 LSB 数码的总宽度 CountLSB。比如 8 位的模板 CountLSB 为 256。而本设备的 AD 为 16 位，则为 65536。其他类型同理均按 2^n=LSB 总数（n 为 Bit 位数）换算即可。

设从设备上读入的某个 AD 原码数据经高位求补后为变量 Lsb，其对应的电压为变量 Volt（单位 mV）。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000mV	Volt = (20000.00 / 65536) * ADBuffer[0]-10000.00	[-10000，+9998.78]
±5000mV	Volt = (10000.00 / 65536) * ADBuffer[0]-5000.00	[-5000，+4999.38]
±2500mV	Volt = (5000.00 / 65536) * ADBuffer[0]-2500.00	[-2500，+2499.69]
0～10000mV	Volt = (10000.00 / 65536) * ADBuffer[0]	[0，+9999.84]
0～5000mV	Volt = (5000.00 / 65536) * ADBuffer[0]	[0，+4999.92]

注意：以上所列 ADBuffer[0]必须是将从设备上读入的 AD 数据（注意在上层用户接口中的 AD 数据读取函数的 ADBuffer 参数指向的用户缓冲区存放的就是这样的最原始数据），且存放这些数据的变量也应该是 16 位整型变量。举列说明如何将取得的 AD 值转换成相应量程的电压值：（此处只转换用户缓冲区中的第一个点，其它同理，且按±10000mV 量程计算）

Visual C++：

```
Lsb = ADBuffer[0];
Volt = (20000.00/65536) * Lsb – 10000.00;
```

Visual Basic：

```
Dim Lsb As Long      ‘ 注意 Visual Basic 中没有无符号 16 位整型数据的定义，所以用 32 位有符号数
Lsb = ADBuffer (0) And 65535    ‘ 通过此方法将有符号 16 位整型数据转换成无符号 16 位有效数据
Volt = (20000.00/65536) * Lsb – 10000.00
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

由于各个通道除了共享采样时钟和触发条件外，均独立工作，分别具有自己独立的存储器，因此其 AD 数据排列非常简单。在各自读回的数据中全为自身的数据。只有在异步模式下，其数据序列应由四个通道穿叉排列形成，即把读回的数据按下列顺序排放(4 个通道)：

数据序列	通道序列
0	AI0[0]
1	AI1[0]
2	AI2[0]
3	AI3[0]
4	AI0[1]
5	AI1[1]
6	AI2[1]
7	AI3[1]
8	AI0[2]
9	AI1[2]
10	AI2[2]
11	AI3[2]

12	AI0[3]
13	AI1[3]
14	AI2[3]
15	AI3[3]
16	AI0[4]
17	AI1[4]
18	AI2[4]
19	AI3[4]
:	:
N	AI0[N/4]
N+1	AI1[N/4]
N+2	AI2[N/4]
N+3	AI3[N/4]

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 n(n 等于头信息的大小)字节位置属于文件头信息，而从 n 字节开始才是真正的 AD 数据。文件头信息包含的内容如下结构体所示。

```
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 DeviceID;                 // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;

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

    PCI8214_PARA_AD ADPara;     // 保存硬件参数
    LONG HeadEndFlag;           // 头信息结束标志

} 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。

注意：黑体字部分为我公司所有数据文件的标准部分，必须一致。

第四节、DA 电压值转换成 LSB 原码数据的换算方法

量程(伏)	计算机语言换算公式(标准 C 语法)	Lsb 取值范围
0～10000mV	Lsb = Volt/ (10000 / 4096)	[0，4095]
±5000mV	Lsb = Volt / (10000 / 4096) + 2048	[0，4095]
±10000mV	Lsb = Volt/ (20000 / 4096) + 2048	[0，4095]

请注意这里求得的 LSB 数据就是用于 WriteDeviceProDA 中的 DAData 参数的。在换算时请注意上下边界。

第六章 上层用户函数接口应用实例

第一节、怎样使用 ReadDeviceProAD 函数直接取得 AD 数据

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，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(主要参考 PCI8214.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨[简易代码演示]丨[查询方式演示源程序]

其他语言的演示可以用上面类似的方法找到。

第二节、怎样使用 ReadDeviceDmaAD 函数直接取得 AD 数据

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，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(主要参考 PCI8214.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨[简易代码演示]丨[DMA 方式演示源程序]

其他语言的演示可以用上面类似的方法找到。

第七章 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、公用接口函数列表（每个函数省略了前缀“PCI8214_”）

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线程操作函数		
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	

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：

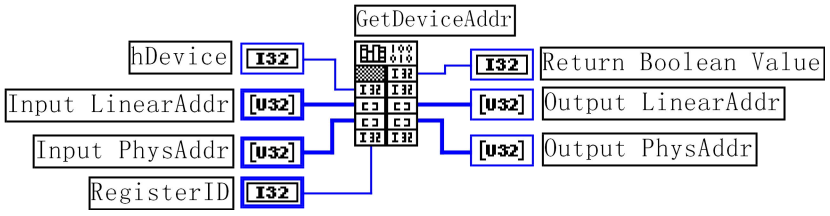
Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCI8214" (ByVal hDevice As Long, _
                                              ByRef LinearAddr As Long, _???
                                              ByRef PhysAddr As Long, _???
                                              ByVal RegisterID As Long, _???
                                              ) 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 创建。

LinearAddr 指针参数，用于返回所取得的映射寄存器指向的线性地址，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。

PhysAddr 所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。

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。

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：

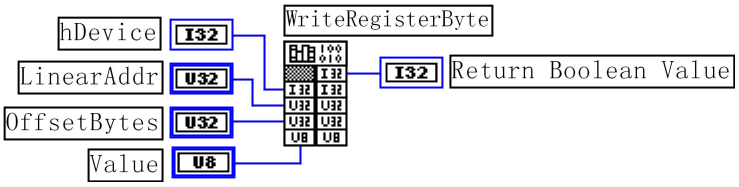
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI8214" ( 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 决定。

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：

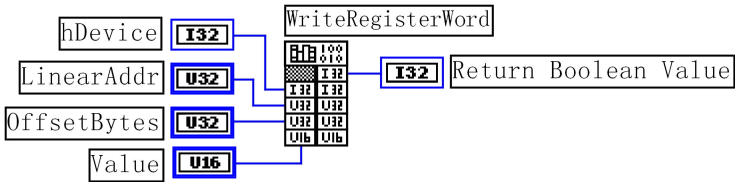
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI8214" (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 确定。

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 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 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
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：

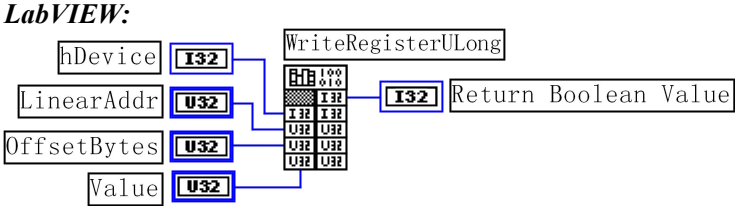
Visual C++:

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

Visual Basic:

```
Declare Sub WriteRegisterULong Lib "PCI8214" (ByVal hDevice As Long, _
                                              ByVal LinearAddr As Long, _
                                              ByVal OffsetBytes As Long, _
                                              ByVal Value As Long)
```

LabVIEW:



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 决定。

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 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 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
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：

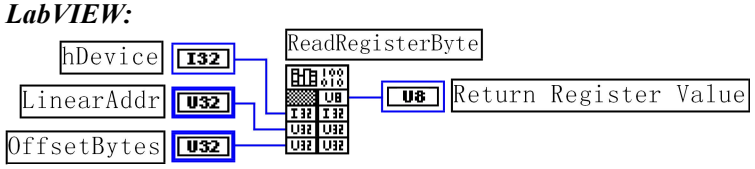
Visual C++:

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

Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI8214" (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 决定。

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：

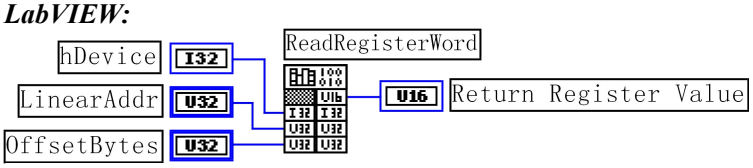
Visual C++:

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

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI8214" (  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 决定。

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：

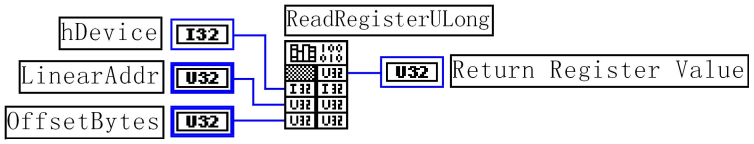
Visual C++:

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

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI8214" (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 决定。

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

函数原型：

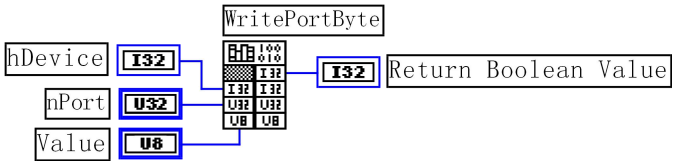
Visual C++:

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

Visual Basic:

```
Declare Function WritePortByte Lib "PCI8214" (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 CreateDeviceEx 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**

函数原型：

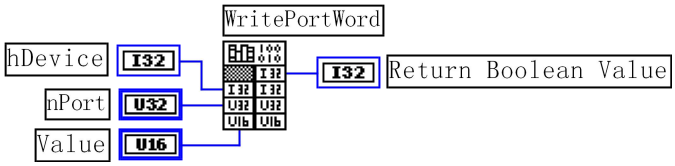
Visual C++:

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

Visual Basic:

```
Declare Function WritePortWord Lib "PCI8214" (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 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**

函数原型：

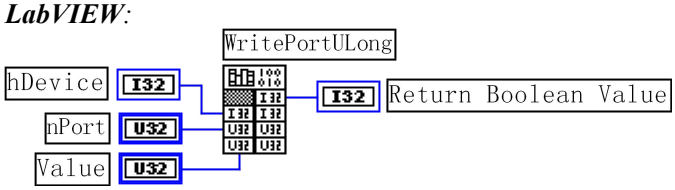
Visual C++:

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

Visual Basic:

```
Declare Function WritePortULong Lib "PCI8214" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

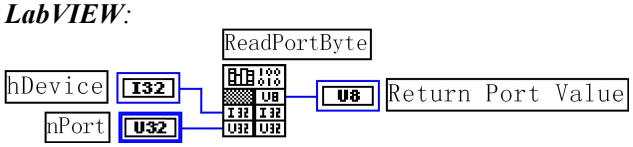
LabVIEW:



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 CreateDeviceEx 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**

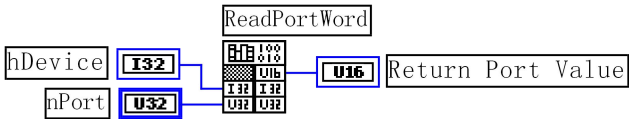
函数原型：
Visual C++:
BYTE ReadPortByte(HANDLE hDevice, UINT nPort)
Visual Basic:
Declare Function ReadPortByte Lib "PCI8214" (ByVal hDevice As Long, _
ByVal nPort As Long) As Byte
LabVIEW:



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 CreateDeviceEx 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**

函数原型：
Visual C++:
WORD ReadPortWord(HANDLE hDevice, UINT nPort)
Visual Basic:
Declare Function ReadPortWord Lib "PCI8214" (ByVal hDevice As Long, _
ByVal nPort As Long) As Integer
LabVIEW:

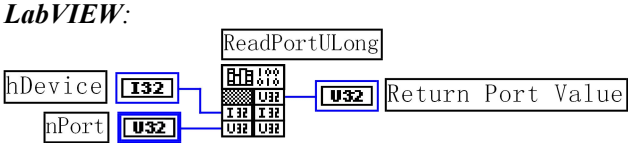


功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄,它应由 CreateDevice 或 CreateDeviceEx 创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式读 I/O 端口**

函数原型：
Visual C++:
ULONG ReadPortULong(HANDLE hDevice, UINT nPort)
Visual Basic:
Declare Function ReadPortULong Lib "PCI8214" (ByVal hDevice As Long, _
ByVal nPort As Long) As Long

LabVIEW:



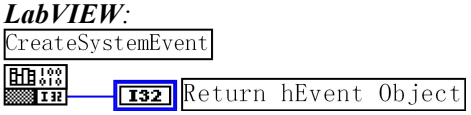
功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄,它应由 CreateDevice 创建。
nPort　设备的 I/O 端口号。
返回值：返回由 nPort 指定端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

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）

♦ 创建内核系统事件

函数原型：
Visual C++:
HANDLE CreateSystemEvent(void);
Visual Basic:
Declare Function CreateSystemEvent Lib " PCI8214 " () 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 " PCI8214 " (ByVal hEvent As Long) As Boolean
LabVIEW:
请参见演示程序。

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。