PCI2366 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司

产品研发部修订

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 PCI2366_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、InitDeviceProAD 、 InitDeviceDmaAD 、 ReadDeviceProAD-Npt 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插座等管脚分配情况。

第二节、如何管理设备

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InitDeviceProAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD Npt函数可以用hDevice句柄实现对AD数据的采样读取等。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何实现数字量的简便操作

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

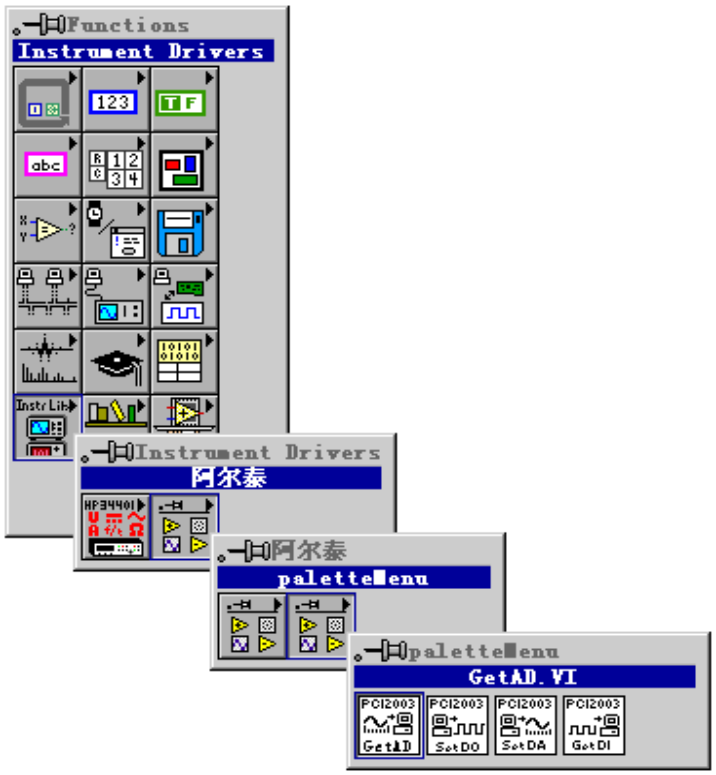
于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PCI 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InitDeviceProAD）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 Npt（或ReadDeviceProAD Half）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，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“PCI2366_”）

函数名	函数功能	备注
① 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
GetDeviceCount	取得同一种 PCI 设备的总台数	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
② AD 数据读取函数		
ReadDevOneAD	从设备上读取一个点的 AD 数据	上层用户
ReadDevBulkAD	从设备上批量读取 AD 数据(也可单点)	上层用户
③ 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
④ DA 模拟量输出操作函数		
WriteDeviceProDA	DA 数据输出	上层用户
⑤ 计数器操作函数		
InitDevCounter	初始化各路计数器	上层用户
GetDevCounter	取得计数器值	上层用户
⑥ 数字 I/O 输入输出函数		
SetDeviceDO	输出开关量状态(控制继电器)	上层用户
GetDeviceDI	取得开关量状态	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:\Art\PCI2366\INCLUDE\PCI2366.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 PCI2366.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。

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 PCI2366.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：

一、在 LabView 中打开 PCI2366.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ReadDeviceProAD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：

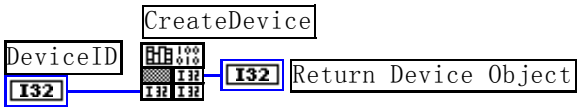
Visual C++：

HANDLE CreateDevice (int DeviceID = 0)

Visual Basic：

Declare Function CreateDevice Lib "PCI2366_32" (ByVal DeviceID As Integer) 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：

DeviceID 设备 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 DeviceID 标识值为名称后缀的标识符来确认和管理该设备。默认值为 0。

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　　GetDeviceCount　　ListDeviceDlg　　ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI2366_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 = PCI2366_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
    :
```

♦ **取得本计算机系统中 PCI2366 设备的总数量**

函数原型：

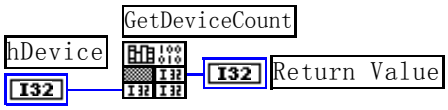
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib"PCI2366_32" (ByVal hDevice As Long) As Integer

LabVIEW：



功能：取得 PCI2366 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 PCI2366 的数量。

相关函数： CreateDevice GetDeviceCount ListDeviceDlg ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PCI2366 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib "PCI2366_32" (ByVal hDevice As Long) As Boolean

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI2366 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 PCI2366 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：

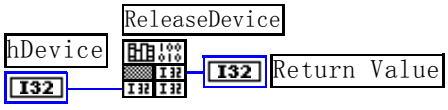
Visual C++:

BOOL ReleaseDevice (HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "PCI2366_32" (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GetLastErrorEx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 数据读取函数说明

- **从设备上读取一个点的 AD 数据**

函数原型：

Visual C++:

```
WORD ReadDevOneAD (HANDLE hDevice
                   int nADChannel)
```

Visual Basic:

```
Declare Function ReadDevOneAD Lib "PCI2366_32" (_
                                ByVal hDevice As Long,_
                                ByVal nADChannel As Integer) As Integer
```

LabVIEW：

请参考相关演示程序。

功能：从设备上读取一个点的 AD 数据。

参数：

hDevice 设备对象句柄。

nADChannel AD 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice ReadDevOneAD ReadDevBulkAD ReleaseDevice

- **从设备上批量读取 AD 数据(也可单点)**

函数原型：

Visual C++：

```
BOOL ReadDevBulkAD (HANDLE hDevice,
                    PWORD pADBuffer,
                    ULONG nReadSizeWords,
                    PPCI2366_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ReadDevBulkAD Lib "PCI2366_32" (ByVal hDevice As Long, _
                                ByRef pADBuffer As Integer, _
                                ByVal nReadSizeWords As Long, _
                                ByRef pADPara As PCI2366_PARA_AD) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：从设备上批量读取 AD 数据(也可单点)

参数：

hDevice 设备对象句柄，它应由设备的CreateDevice创建。

PADBuffer AD 数据缓冲区

NReadSizeWords 读取 AD 的点数

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。请参考《AD硬件参数介绍》。

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

相关函数: CreateDevice ReadDevOneAD ReadDevBulkAD ReleaseDevice

第四节、计数器操作函数原型说明

♦ 初始化各路计数器

函数原型：

Visual C++:

```
BOOL InitDevCounter (HANDLE hDevice,
                     PPCI2366_PARA_COUNTER_CTRL pCtrlPara,
                     int InitCntrVal,
                     int nCntrChannel)
```

Visual Basic:

```
Declare Function InitDevCounter Lib "PCI2366_32" (_
                                 ByVal hDevice As Long, _
                                 ByRef pCtrlPara As PCI2366_PARA_COUNTER_CTRL, _
                                 ByVal InitCntrVal As Integer, _
                                 ByVal nCntrChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：初始化各路计数器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pCtrlPara 计数器控制字

InitCntrVal 计数器的 16 位值

nCntrChannel 计数器通道号（0-2）

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GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice InitDevCounter GetDevCounter ReleaseDevice

♦ 取得计数器值

函数原型：

Visual C++:

```
BOOL GetDevCounter (HANDLE hDevice
                    PLONG pCntrValue,
                    int nCntrChannel)
```

Visual Basic:

```
Declare Function GetDevCounter Lib "PCI2366_32" (_
                               ByVal hDevice As Long,_
                               ByRef pCntrValue As Long, _
                               ByVal nCntrChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：取得计数器的值。

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，意味着AD被启动，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice InitDevCounter GetDevCounter 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,
                 PPCI2366_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PCI2366" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCI2366_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 PCI2366_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 PCI2366_PARA_AD请参考PCI2366.h或PCI2366.Bas或PCI2366.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

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPCI2366_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI2366" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCI2366_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PCI2366_PARA_AD的详细介绍请参考PCI2366.h或PCI2366.Bas或PCI2366.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

第六节、DA 模拟量输出操作函数原型说明

♦ **DA 数据输出**

函数原型：

Visual C++:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       SHORT nDAData,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceOneDA Lib "PCI2366"(ByVal hDevice As Long, _
                                                 ByVal ulDataCode As Long, _
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：DA 输出。

参数：

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

ulDataCode 准备输出的 12 位 DA 数据 LSB 原码。

nDAChannel 指定输出的 DA 通道。

返回值：若 DA 成功单点输出，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice WriteDeviceProDA ReleaseDevice

第七节、DIO 数字量输入输出操作函数原型说明

- **数字量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI (HANDLE hDevice,
                  PPCI2366_PARA_DI pDIPara）
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI2366_32" (_
                                 ByVal hDevice As Long,_
                                 ByRef pDIPara As PCI2366_PARA_DI) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pDIPara 开关状态

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

- **数字量输出**

函数原型：

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  PPCI2366_PARA_DO pDOPara)
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI2366_32" (_
                                 ByVal hDevice As Long,_
                                 ByRef pDOPara As PCI2366_PARA_DO) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pDOPara 开关状态。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

第四章 硬件参数结构

第一节、AD 硬件参数介绍（PCI2366_PARA_AD）

Visual C++:

```
typedef struct _PCI2366_PARA_AD
{
    LONG FirstChannel;          // 首通道
    LONG LastChannel;           // 末通道
} PCI2366_PARA_AD, *PPCI2366_PARA_AD;
```

Visual Basic：

```
Type PCI2366_PARA_AD        ' 用于 AD 转换的硬件参数
    FirstChannel As Long        ' 首通道
    LastChannel As Long         ' 末通道
End Type
```

LabVIEW:

请参考相关演示程序。

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。

第二节、计数器控制字(CONTROL)

Visual C++:

```
typedef struct _PCI2366_PARA_COUNTER_CTRL       // 计数器控制字(CONTROL)
{
    // 计数器控制字
    BYTE OperateType;   // 操作类型(D5D4)
    BYTE CountMode;     // 计数方式(D3-D1)
    BYTE BCD;           // 计数类型(D0)
    // 信号源控制
    BYTE ClockSource;   // 时钟基准源选择
    BYTE GateSource;    //  GATE 门信号源选择
} PCI2366_PARA_COUNTER_CTRL,*PPCI2366_PARA_COUNTER_CTRL;
```

Visual Basic：

```
Type PCI2366_PARA_COUNTER_CTRL      ' 计数器控制字(CONTROL)

    ' 计数器控制字
    OperateType   As Byte       ' 操作类型(D5D4)
    CountMode     As Byte       ' 计数方式(D3-D1)
    BCD           As Byte       ' 计数类型(D0)
    ' 信号源控制
    ClockSource   As Byte       ' 时钟基准源选择
    GateSource    As Byte       '  GATE 门信号源选择
End Type
```

LabVIEW:

请参考相关演示程序。

OperateType 操作类型选择。

常量名	常量值	功能定义
PCI2366_OperateType_0	0x00	计数器锁存操作
PCI2366_OperateType_1	0x01	只读/写低字节
PCI2366_OperateType_2	0x02	只读/写高字节
PCI2366_OperateType_3	0x03	先读/写低字节，后读/写高字节

CountMode CountMode 成员所使用选项。

常量名	常量值	功能定义
PCI2366_CountMode_0	0x00	计数方式 0，计数器结束中断方式
PCI2366_CountMode_1	0x01	计数方式 1，可编程单次脉冲方式
PCI2366_CountMode_2	0x02	计数方式 2，频率发生器方式
PCI2366_CountMode_3	0x03	计数方式 3，方波频率发生器方式
PCI2366_CountMode_4	0x04	计数方式 4，软件触发选通方式
PCI2366_CountMode_5	0x05	计数方式 5，硬件触发选通方式

BCD 成员所使用选项。

常量名	常量值	功能定义
PCI2366_BCD_0	0x00	计数类型 0，二进制计数
PCI2366_BCD_1	0x01	计数类型 1，BCD 码计数

中断硬件参数 PCI2366_PARA_INT 中的 bDI0 和 bOutTrigger 成员所使用选项

常量名	常量值	功能定义
PCI2366_RISING_EDGE	0x01	上升沿触发中断
PCI2366_FALLING_EDGE	0x02	下降沿触发中断
PCI2366_ALL_EDGE	0x03	上下降沿触发中断

注明：PCI2366_TRIGDIR_POSIT_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。而在电平类型下，无论正电平还是负电平均触发。

ClockSource AD 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PCI2366_IN_CLOCK	0x00	内部时钟定时触发(板上 10MHz)
PCI2366_OUT_CLOCK	0x01	外部时钟定时触发
PCI2366_OTHER_CLOCK	0x02	其他计数器的 OUT 输出提供的 Clock(CLK2 接 OUT1,CLK1 接 OUT0, CLK0 接外部 CLK)

GateSource 成员所使用选项。

常量名	常量值	功能定义
PCI2366_LOW_VOLT_GATE	0x00	低电平(内部软件自动实现)
PCI2366_HIGH_VOLT_GATE	0x01	高电平(内部软件自动实现)
PCI2366_OUTSIDE_GATE	0x02	外部 GATE 信号输入(D 型头上)
PCI2366_NOUTSIDE_GATE	0x03	外部 GATE 信号反向输入(D 型头上)

第三节、开关量参数结构（PCI2366_STATUS_AD）

Visual C++：

```
typedef struct _PCI2366_PARA_DO          // 数字量输出参数
{
    BYTE DO0;          // 0 通道
    BYTE DO1;          // 1 通道
    BYTE DO2;          // 2 通道
    BYTE DO3;          // 3 通道
    BYTE DO4;          // 4 通道
    BYTE DO5;          // 5 通道
    BYTE DO6;          // 6 通道
    BYTE DO7;          // 7 通道
    BYTE DO8;          // 8 通道
    BYTE DO9;          // 9 通道
    BYTE DO10;          // 10 通道
    BYTE DO11;          // 11 通道
    BYTE DO12;          // 12 通道
    BYTE DO13;          // 13 通道
    BYTE DO14;          // 14 通道
    BYTE DO15;          // 15 通道
} PCI2366_PARA_DO,*PPCI2366_PARA_DO;
typedef struct _PCI2366_PARA_DI          // 数字量输入参数
{
    BYTE DI0;          // 0 通道
    BYTE DI1;          // 1 通道
    BYTE DI2;          // 2 通道
    BYTE DI3;          // 3 通道
    BYTE DI4;          // 4 通道
    BYTE DI5;          // 5 通道
    BYTE DI6;          // 6 通道
    BYTE DI7;          // 7 通道
    BYTE DI8;          // 8 通道
    BYTE DI9;          // 9 通道
    BYTE DI10;          // 10 通道
    BYTE DI11;          // 11 通道
    BYTE DI12;          // 12 通道
    BYTE DI13;          // 13 通道
    BYTE DI14;          // 14 通道
    BYTE DI15;          // 15 通道
} PCI2366_PARA_DI,*PPCI2366_PARA_DI;
```

Visual Basic：

```
Type PCI2366_PARA_DO        ' 数字量输出参数
    DO0     As Byte       ' 0 通道
    DO1     As Byte       ' 1 通道
    DO2     As Byte       ' 2 通道
    DO3     As Byte       ' 3 通道
    DO4     As Byte       ' 4 通道
    DO5     As Byte       ' 5 通道
    DO6     As Byte       ' 6 通道
```

```
    DO7     As Byte      ' 7 通道
    DO8     As Byte      ' 8 通道
    DO9     As Byte      ' 9 通道
    DO10    As Byte      ' 10 通道
    DO11    As Byte      ' 11 通道
    DO12    As Byte      ' 12 通道
    DO13    As Byte      ' 13 通道
    DO14    As Byte      ' 14 通道
    DO15    As Byte      ' 15 通道
End Type

Type PCI2366_PARA_DI        ' 数字量输入参数

    DI0     As Byte      ' 0 通道
    DI1     As Byte      ' 1 通道
    DI2     As Byte      ' 2 通道
    DI3     As Byte      ' 3 通道
    DI4     As Byte      ' 4 通道
    DI5     As Byte      ' 5 通道
    DI6     As Byte      ' 6 通道
    DI7     As Byte      ' 7 通道
    DI8     As Byte      ' 8 通道
    DI9     As Byte      ' 9 通道
    DI10    As Byte      ' 10 通道
    DI11    As Byte      ' 11 通道
    DI12    As Byte      ' 12 通道
    DI13    As Byte      ' 13 通道
    DI14    As Byte      ' 14 通道
    DI15    As Byte      ' 15 通道
End Type
```

LabVIEW:

请参考相关演示程序。

第四节、中断信息控制参数

Visual C++:

```
typedef struct _PCI2366_PARA_INT     // 中断信息控制参数
{
    LONG bDI0;              // 开关输入 0 通道跳变触发中断(信号类型可选, =0 禁止)
    LONG bOutTrigger;     // D 型头上的外部信号触发中断 INTIN(信号类型可选, =0 禁止)
    LONG bADTrigger;     // 允许 AD 转换结束信号触发中断(=1:允许; =0:禁止)
    LONG bCounter0Out; // 允许计数器 0 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
    LONG bCounter1Out; // 允许计数器 1 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
    LONG bCounter2Out; // 允许计数器 2 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
} PCI2366_PARA_INT, *PPCI2366_PARA_INT;
```

Visual Basic:

```
Type PCI2366_PARA_INT     ' 中断信息控制参数

    bDI0 As Long            ' 开关输入 0 通道跳变触发中断(信号类型可选, =0 禁止)
    bOutTrigger    As Long     ' D 型头上的外部信号触发中断 INTIN(信号类型可选, =0 禁止)
    bADTrigger    As Long     ' 允许 AD 转换结束信号触发中断(=1:允许  =0:禁止)
```

```
    bCounter0Out    As Long ' 允许计数器 0 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
    bCounter1Out    As Long ' 允许计数器 1 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
    bCounter2Out As Long    ' 允许计数器 2 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
End Type
```

LabVIEW:

请参考相关演示程序。

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）
±10V	Volt = (20000.00/4096) * (ADBuffer[0]&0xFFF) – 10000.00	[-10000, +9997.55]
±5V	Volt = (10000.00/4096) * (ADBuffer[0]&0xFFF) – 5000.00	[-5000, +4998.77]
0～10V	Volt = (10000.00/4096) * (ADBuffer[0]&0xFFF)	[0, +9998.77]

下面举例说明各种语言的换算过程（以±5000mV 量程为例）

Visual C++：

```
Lsb = ADBuffer[0]&0xFFF;
Volt = (10000.00/8192) * Lsb -5000.00;
```

Visual Basic：

```
Lsb = ADBuffer [0] And &HFFF
Volt = (10000.00/8192) * Lsb – 5000.00
```

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

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	…

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 HeadVersion;                   // 头信息版本(D15-D8=Major,D7-D0=Minijor) = 1.0
    LONG VoltBottomRange;               // 量程下限(mV)
    LONG VoltTopRange;                  // 量程上限(mV)
    LONG StaticOverFlow;                // 同批文件识别码
    PCI2366_PARA_AD ADPara;             // 保存硬件参数
    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	Lsb 取值范围
0～5000mV	Lsb=Volt/(5000.00/4096)	[0，4095]
0～10000mV	Lsb=Volt/(10000.00/4096)	[0，4095]
±5000mV	Lsb=Volt/(10000.00/4096)+2048	[0，4095]
±10000mV	Lsb=Volt/(20000.00/4096)+2048	[0，4095]

第六章 上层用户函数接口应用实例

第一节、简易程序演示说明

一、怎样使用ReadDevOneAD函数读取一个点的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]丨[阿尔泰测控演示系统]丨[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[AD 方式]

二、怎样使用WriteDeviceProDA函数连续的DA输出数据

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]丨[阿尔泰测控演示系统]丨[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[DA 方式]

三、怎样使用InitDevCounter函数初始化各路计数器数据

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]丨[阿尔泰测控演示系统]丨[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨

四、怎样使用GetDeviceDI函数进行更便捷式数字量输入操作

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]丨[阿尔泰测控演示系统]丨[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[DIO...]

五、怎样使用SetDeviceDO函数进行更便捷式数字量输出操作

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]丨[阿尔泰测控演示系统]丨[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[DIO...]

第二节、高级程序演示说明

高级程序演示了本设备的所有功能，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 PCI2366.h 和 ADDoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2366 16 路 AD、4 路 DA、3 路 CNT 和 16 路 DIO 卡]**丨**[Microsoft Visual C++]**丨**[高级代码演示]**

其默认存放路径为：系统盘\ART\PCI2366\SAMPLES\VC\ADVANCED

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、公用接口函数总列表（每个函数省略了前缀“PCI2366_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 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		
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

- **取得指定的指定设备寄存器组 BAR 地址**

函数原型：

Visual C++:

```
BOOL GetDeviceBar (HANDLE hDevice,
                   PUCHAR pbPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib"PCI2366_32" (_
                              ByVal hDevice As Long,_
                              ByRef pbPCIBar() As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbPCIBar 返回 PCI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReleaseDevice

♦ **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:

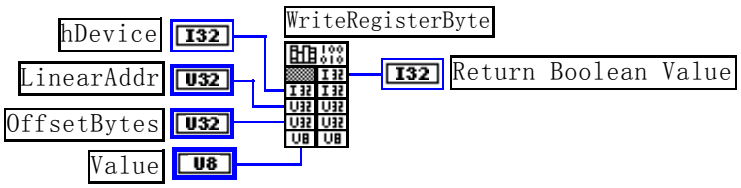
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterByte Lib"PCI2366_32" (_
                                    ByVal hDevice As Long,_
                                    ByRef pbLinearAddr 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创建。

pbLinearAddr 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:

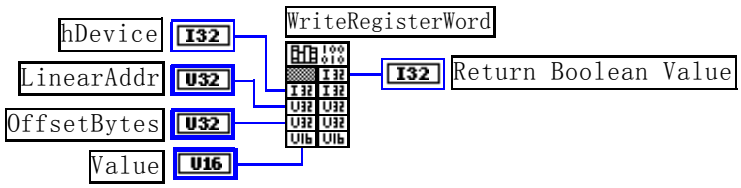
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI2366_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pbLinearAddr As Long,_
                                        ByVal OffsetBytes As Long,_
                                        ByVal Value As Long) 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创建。

pbLinearAddr 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:

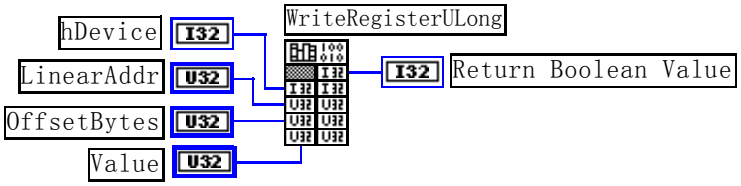
Visual C++:

```
BOOL WriteRegisterULong (HANDLE hDevice,
                         PUCHAR pbLinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib"PCI2366_32" (_
                                    ByVal hDevice As Long,_
                                    ByRef pbLinearAddr As Long,_
                                    ByVal OffsetBytes As Long, _
                                    ByVal Value As Long) As Boolean
```

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创建。

pbLinearAddr 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:

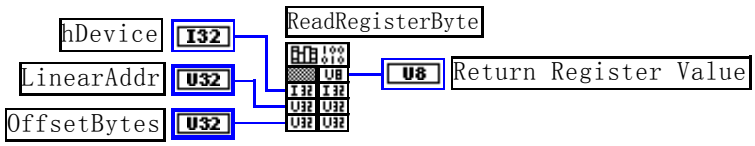
Visual C++:

```
BYTE ReadRegisterByte (HANDLE hDevice,
                       PUCHAR pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib"PCI2366_32" (_
                                    ByVal hDevice As Long,_
                                    ByRef pbLinearAddr 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创建。

pbLinearAddr 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:

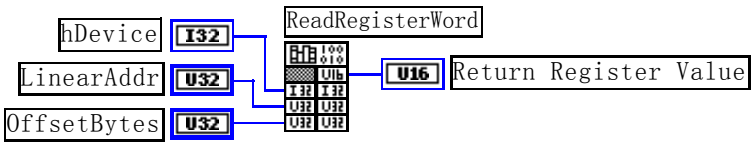
Visual C++:

```
WORD ReadRegisterWord (HANDLE hDevice,
                       PUCHAR pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib"PCI2366_32" (_
                                   ByVal hDevice As Long,_
                                   ByRef pbLinearAddr As Long,_
                                   ByVal OffsetBytes As Long) As Long
```

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创建。

pbLinearAddr 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:

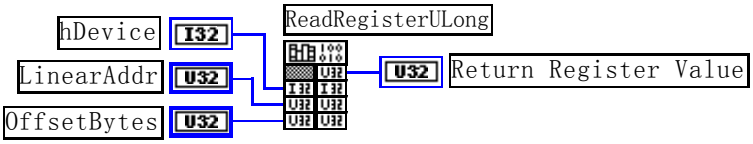
Visual C++:

```
ULONG ReadRegisterULong (HANDLE hDevice,
                         PUCHAR pbLinearAddr,
                         ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib"PCI2366_32" (_
                                    ByVal hDevice As Long, _
                                    ByRef pbLinearAddr 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创建。

pbLinearAddr 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

- **以单字节(8Bit)方式写 I/O 端口**

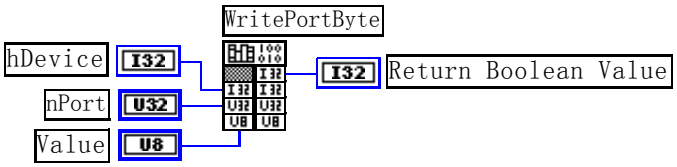
Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    PUCHAR pPort,
                    BYTE Value)
```

Visual Basic:

```
Declare Function WritePortByte Lib "PCI2366_32" (_
                                ByVal hDevice As Long, _
                                ByRef pbLinearAddr As Long, _
                                ByVal OffsetBytes As Long) As Long
```

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创建。

pPort 指定寄存器的物理基地址。

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GetLastErrorEx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**

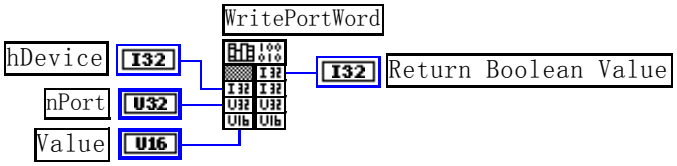
Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    PUCHAR pPort,
                    WORD Value)
```

Visual Basic:

```
Declare Function WritePortWord Lib "PCI2366_32" (_
                                ByVal hDevice As Long,_
                                ByRef pPort As Long,_
                                ByVal Value As Long) 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。
pPort 指定寄存器的物理基地址。
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GetLastErrorEx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**

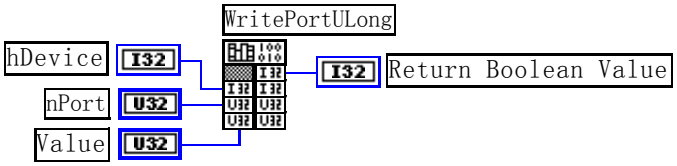
Visual C++:

```
BOOL WritePortULong (HANDLE hDevice,
                     PUCHAR pPort,
                     ULONG Value)
```

Visual Basic:

```
Declare Function WritePortULong Lib "PCI2366_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pPort 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创建。
pPort 指定寄存器的物理基地址。
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GetLastErrorEx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**

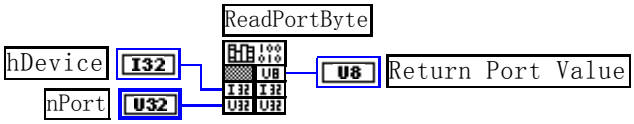
Visual C++:

```
BYTE ReadPortByte (HANDLE hDevice,
                   PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortByte Lib "PCI2366_32" (_
                                        ByVal hDevice As Long, _
                                        ByRef pPort 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创建。
pPort 指定寄存器的物理基地址。
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**

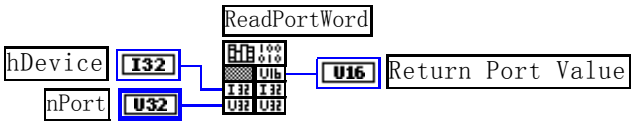
Visual C++:

```
WORD ReadPortWord (HANDLE hDevice,
                   PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortWord Lib"PCI2366_32" (_
                                ByVal hDevice As Long,_
                                ByRef pPort As Long) As Long
```

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创建。
pPort 指定寄存器的物理基地址。
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**

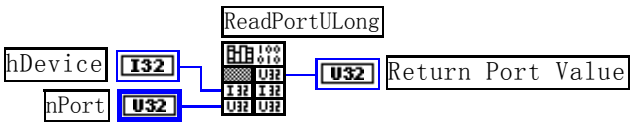
Visual C++:

```
ULONG ReadPortULong (HANDLE hDevice,
                     PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortULong Lib"PCI2366_32" (_
                                ByVal hDevice As Long, _
                                ByRef pPort 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。

pPort 指定寄存器的物理基地址。

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

- **创建内核系统事件**

函数原型：

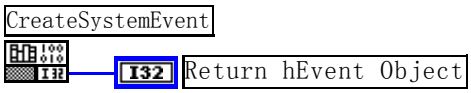
Visual C++:

HANDLE CreateSystemEvent (void)

Visual Basic:

Declare Function CreateSystemEvent Lib "PCI2366_32" () 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"PCI2366_32" (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