

PCI9100 任意波形发生器

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

目 录

目 录	1
第一章 版权信息与命名约定	2
第一节、版权信息	2
第二节、命名约定	2
第二章 使用纲要	2
第一节、使用上层用户函数，高效、简单	2
第二节、如何管理设备	2
第三节、如何实现DA波形数据输出	3
第四节、如何实现数字量的简便操作	3
第五节、哪些函数对您不是必须的	3
第三章 PCI设备操作函数接口介绍	3
第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI9100_”）	4
第二节、设备对象管理函数原型说明	5
第三节、DA模拟量输出操作函数原型说明	8
第四节、DA硬件参数系统保存与读取函数原型说明	15
第五节、DIO数字量输入输出操作函数原型说明	16
第四章 硬件参数结构	18
第一节、DA硬件参数介绍（PCI9100_PARA_DA）	18
第二节、DA状态参数结构（PCI9100_STATUS_DA）	19
第三节、DA各段信息参数结构（PCI9100_SEGMENT_INFO）	20
第五章 数据格式转换与排列规则	20
第一节、DA电压值转换成LSB原码数据的换算方法	20
第二节、关于DA数据DABuffer缓冲区中的数据排放规则	20
第六章 上层用户函数接口应用实例	21
第一节、简易程序演示说明	21
第二节、高级程序演示说明	21
第七章 共用函数介绍	21
第一节、公用接口函数总列表（每个函数省略了前缀“PCI9100_”）	22
第二节、PCI内存映射寄存器操作函数原型说明	22
第三节、IO端口读写函数原型说明	30
第四节、线程操作函数原型说明	32
第五节、文件对象操作函数原型说明	33
第六节、各种参数保存和读取函数原型说明	36
第七节、其他函数原型说明	38

第一章 版权信息与命名约定

第一节、版权信息

本软件产品及相关套件均属北京阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位、我公司授权的代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，我们将热情接待。

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 PCI9100_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 [InitDeviceDA](#)、[WriteDeviceBulkDA](#) 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 [InitDeviceDA](#) 可以使用hDevice句柄以程序查询方式初始化设备的DA部件，[WriteDeviceBulkDA](#) 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 DA 波形数据输出

当您有了 hDevice 设备对象句柄后，便可用 [InitDeviceDA](#) 函数初始化 DA 部件，关于采样通道、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。然后用 [WriteDeviceBulkDA](#) 将准备好的 DA 数据写入板载 RAM 中，接着用 [EnableDeviceDA](#) 即可启动 DA 部件，开始 DA 输出。当您需要暂停设备时，执行 [DisableDeviceDA](#)，当您需要关闭 DA 设备时，[ReleaseDeviceDA](#) 便可帮您实现（但设备对象 hDevice 依然存在）。

第四节、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[8] 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[8] 中的相应元素决定。

第五节、哪些函数对您不是必须的

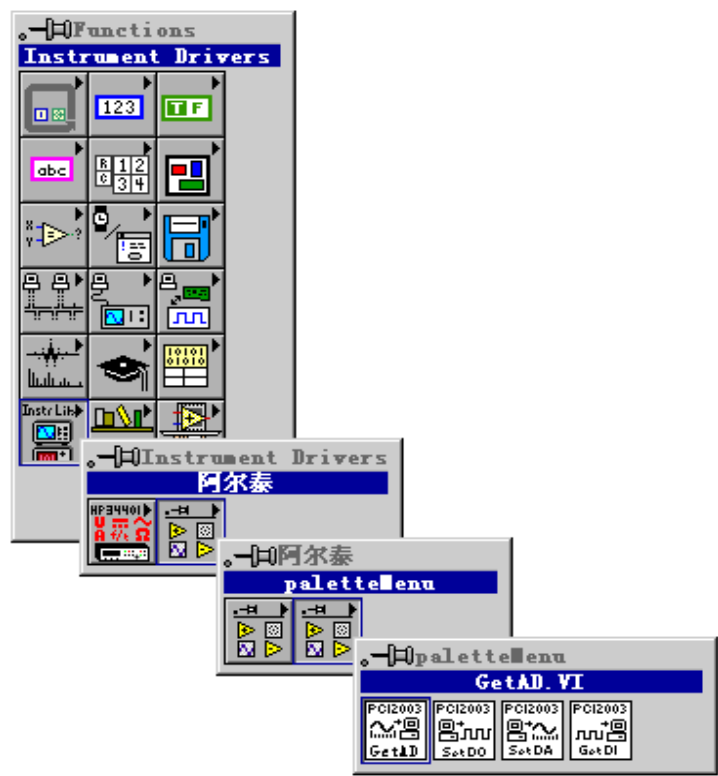
公共函数如 [CreateFileObject](#)，[WriteFile](#)，[ReadFile](#) 等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么 [GetDeviceAddr](#)，[WriteRegisterByte](#)，[WriteRegisterWord](#)，[WriteRegisterULong](#)，[ReadRegisterByte](#)，[ReadRegisterWord](#)，[ReadRegisterULong](#) 等函数您可完全不必理会，除非您是作为底层用户管理设备。而 [WritePortByte](#)，[WritePortWord](#)，[WritePortULong](#)，[ReadPortByte](#)，[ReadPortWord](#)，[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，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 Library 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“PCI9100_”）

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
CreatDeviceEx	创建 PCI 设备对象(用设备物理号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
② DA 模拟量输出操作函数		
SetDevTrigLevelDA	设置 DA 的触发电平	上层用户
SetDevFrequencyDA	可动态改变 DA 采样频率	上层用户
ReadSegmentInfo	读段信息	上层用户
InitDeviceDA	初始化 PCI 设备上的 DA 部件准备传输	上层用户
WriteDeviceOneDA	DA 单点输出	上层用户
WriteDeviceBulkDA	批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中	上层用户
ReadDeviceBulkDA	以批量方式将板载 RAM 中的 DA 数据回读至主机的用户缓冲区	上层用户
EnableDeviceDA	启动 DA 设备，开始转换	上层用户
SetDeviceTrigDA	软件产生触发事件	上层用户
GetDevStatusDA	取得当前 DA 状态	上层用户
DisableDeviceDA	暂停 DA 设备	上层用户
ReleaseDeviceDA	释放 DA 设备	上层用户
③ DA 硬件参数系统保存、读取函数		
LoadParaDA	从 Windows 系统中读入硬件参数	上层用户
SaveParaDA	往 Windows 系统写入设备硬件参数	上层用户
ResetParaDA	将硬件参数结构体值复位为出厂默认值	上层用户
④ 数字量简易操作函数		

GetDeviceDI	数字输入函数	上层用户
SetDeviceDO	数字输出函数	上层用户
RetDeviceDO	数字量输出状态	

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\PCI9100\INCLUDE\PCI9100.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 PCI9100.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。

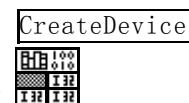
Visual Basic:

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, 在弹出的对话框中选择 PCI9100.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 PCI9100.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:

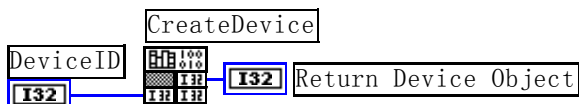
Visual C++:

```
HANDLE CreateDevice (int DeviceLgcID = 0)
```

Visual Basic:

```
Declare Function CreateDevice Lib "PCI9100" (Optional ByVal DeviceLgcID As Integer = 0) As Long
```

LabVIEW:



功能: 该函数使用逻辑号创建设备对象, 并返回其设备对象句柄 hDevice。只有成功获取 hDevice, 您才能实现

对该设备所有功能的访问。

参数:

DeviceLgcID 逻辑设备 ID(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 DeviceLgcID 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](#) [CreatDeviceEx](#) [GetDeviceCount](#)
 [GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

Visual C++ 程序举例

```

:
HANDLE hDevice;    // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI9100_CreateDevice (DeviceLgcID); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALID_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 = PCI9100_CreateDevice (DeviceLgcID) ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub ' 退出该过程
End If
:

```

◆ **创建设备对象函数 (物理号)**

函数原型:

Visual C++:

[HANDLE CreateDeviceEx \(int DevicePhysID = 0\)](#)

Visual Basic:

[Declare Function CreateDeviceEx Lib "PCI9100" \(Optional ByVal DevicePhysID As Integer = 0\) As Long](#)

LabVIEW:

请参考相关演示程序。

功能: 该函数使用物理号创建设备对象, 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(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 DevicePhysID 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](#) [CreatDeviceEx](#) [GetDeviceCount](#)
 [GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

◆ **取得本计算机系统中 PCI9100 设备的总数量**

函数原型:

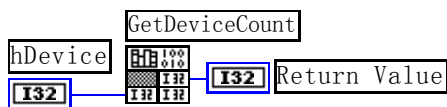
Visual C++:

`int GetDeviceCount (HANDLE hDevice)`

Visual Basic:

`Declare Function GetDeviceCount Lib "PCI9100" (ByVal hDevice As Long) As Integer`

LabVIEW:



功能: 取得 PCI9100 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 PCI9100 的数量。

相关函数: [CreateDevice](#) [CreatDeviceEx](#) [GetDeviceCount](#)
 [GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

◆ 取得该设备当前相应的 ID 号

函数原型:

Visual C++:

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

Visual Basic:

`Declare Function GetDeviceCurrentID Lib "PCI9100" (ByVal hDevice As Long, _
 ByRef DeviceLgcID As Long, _
 ByRef DevicePhysID As Long) As Boolean`

LabVIEW:

请参考相关演示程序。

功能: 取得指定设备相应 ID 号。

参数:

hDevice 设备对象句柄, 它指向要取得逻辑号的设备, 它应由 [CreateDevice](#) 创建。

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

DevicePhysID 返回设备的物理 ID 号。

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](#) [CreatDeviceEx](#) [GetDeviceCount](#)
 [GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

◆ 用对话框控件列表计算机系统中所有 PCI9100 设备各种配置信息

函数原型:

Visual C++:

`BOOL ListDeviceDlg (HANDLE hDevice)`

Visual Basic:

`Declare Function ListDeviceDlg Lib "PCI9100" (ByVal hDevice As Long) As Boolean`

LabVIEW:

请参考相关演示程序。

功能: 列表系统中 PCI9100 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 PCI9100 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:

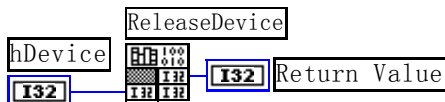
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "PCI9100" (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、DA 模拟量输出操作函数原型说明

◆ 设置触发电平

函数原型:

Visual C++:

BOOL SetDevTrigLevelDA (HANDLE hDevice,
float fTrigLevelVolt)

Visual Basic:

Declare Function SetDevTrigLevelDA Lib "PCI9100" (ByVal hDevice As Long, _
ByVal fTrigLevelVolt As Single) As Boolean

LabView:

请参考相关演示程序。

功能: 设置触发电平, 该触发电平对所有 DA 通道同时有效。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

nTrigLevelVolt 触发电平值, 单位为 mV, 其取值范围为[-10000mV, +10000mV]。

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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 动态改变采样频率

函数原型:

Visual C++:

```
BOOL SetDevFrequencyDA (HANDLE hDevice,  
                        LONG nFrequency,  
                        int nDAChannel)
```

Visual Basic:

```
Declare Function SetDevFrequencyDA Lib "PCI9100" (ByVal hDevice as Long, _  
                                                ByVal nFrequency As Long, _  
                                                ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考演示源程序。

功能: 在 DA 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](#)或 [CreateDeviceEx](#)创建。

nFrequency 采样频率 (Hz)。

nDAChannel DA 通道号, 取值范围为[0, 1]。

返回值: 如果调用成功, 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **读段信息**

函数原型:

Visual C++:

```
BOOL ReadSegmentInfo(HANDLE hDevice,  
                    LONG SegmentCount,  
                    PCI9100_SEGMENT_INFO SegmentInfo[],  
                    int nDAChannel)
```

Visual Basic:

```
Declare Function ReadSegmentInfo Lib "PCI9100" (ByVal hDevice as Long, _  
                                                ByVal SegmentCount As Long, _  
                                                ByRef SegmentInfo() As PCI9100_SEGMENT_INFO, _  
                                                ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考演示源程序。

功能: 读段信息。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

SegmentCount 分段总数, 它的理论取值范围为[1, 666], 但实际工作时, 它要受到板载 RAM 大小、各段数据长度大小及其他通道对 RAM 的使用情况等因素的影响。在该通道分配的板载 RAM 空间一定的情况下, 其各段数据长度越短, 则可分配的段数越多。

SegmentInfo[] 段信息集合, 属于 PCI9100_SEGMENT_INFO 的结构数据类型, 它的有效元素个数由 **SegmentCount** 参数决定。因此用户分配的段信息集合不应小于 **SegmentCount** 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中, 它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

nDAChannel DA 通道号, 取值范围为[0, 1]。

返回值: 如果调用成功, 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA

[WriteDeviceBulkDA](#)[ReadDeviceBulkDA](#)[EnableDeviceDA](#)[SetDeviceTrigDA](#)[GetDevStatusDA](#)[DisableDeviceDA](#)[ReleaseDeviceDA](#)[ReleaseDevice](#)

◆ 初始化设备对象

函数原型

Visual C++:

```
BOOL InitDeviceDA(HANDLE hDevice,
                  LONG SegmentCount,
                  PCI9100_SEGMENT_INFO SegmentInfo[],
                  PCI9100_PARA_DA pDAPara,
                  int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceDA Lib "PCI9100" ( ByVal hDevice As Long, _
                                             ByVal SegmentCount As Long, _
                                             ByRef SegmentInfo() As PCI9100_SEGMENT_INFO, _
                                             ByRef pDAPara As PCI9100_PARA_DA, _
                                             ByVal nDAChannel As Integer) 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DA部件, 为设备操作就绪有关工作做准备, 如预置DA采集通道、采样频率等。但它并不启动DA设备, 若要启动DA设备, 须在调用此函数之后再调用 [EnableDeviceDA](#) (但DA要实际输出波形, 则一般要等待某种触发事件的到来)。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

SegmentCount 分段总数, 它的理论取值范围为[1, 65536], 但实际工作时, 它要受到板载 RAM 大小、各段数据长度大小及其他通道对 RAM 的使用情况等因素的影响。在该通道分配的板载 RAM 空间一定的情况下, 其各段数据长度越短, 则可分配的段数越多。

SegmentInfo[] 段信息集合, 属于 PCI9100_SEGMENT_INFO 的结构数据类型, 它的有效元素个数由 SegmentCount 参数决定。因此用户分配的段信息集合不应小于 SegmentCount 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中, 它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

pDAPara 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《[DA硬件参数结构](#)》。

nDAChannel DA 通道号, 取值范围为[0,1]。

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](#)[SetDevTrigLevelDA](#)[SetDevFrequencyDA](#)[ReadSegmentInfo](#)[InitDeviceDA](#)[WriteDeviceOneDA](#)[WriteDeviceBulkDA](#)[ReadDeviceBulkDA](#)[EnableDeviceDA](#)[SetDeviceTrigDA](#)[GetDevStatusDA](#)[DisableDeviceDA](#)[ReleaseDeviceDA](#)[ReleaseDevice](#)

◆ DA 单点输出

函数原型:

Visual C++:

```
BOOL WriteDeviceOneDA (HANDLE hDevice,
                       ULONG ulDataCode,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceOneDA Lib "PCI9100" (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 单点输出。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

ulDataCode 输出码值 (0—4095)。

nDAChannel 通道号, 取值范围为[0,1]。

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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中

函数原型:

Visual C++:

```
BOOL WriteDeviceBulkDA (HANDLE hDevice,  
                        SHORT DABuffer[],  
                        LONG nWriteSizeWords,  
                        PLONG nRetSizeWords,  
                        int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceBulkDA Lib "PCI9100" (ByVal hDevice as Long, _  
                                                ByRef DABuffer() As Integer, _  
                                                ByVal nWriteSizeWords As Long, _  
                                                ByRef nRetSizeWords As Long, _  
                                                ByVal nDAChannel As Integer) As Boolean
```

LabView:

请参考相关演示程序。

功能: 往指定通道的板载 RAM 中写入批量 DA 数据。在初始化设备之后, 启动 DA 之前, 便可以用此函数将 DA 数据写入板载 RAM 以供输出。但是在启动之后 (即在输出过程中, 不能对 RAM 进行写操作, 包括读操作)。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

DABuffer[] 接受 DA 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 DA 数据转换成相应的电压值, 请参考《[数据格式转换与排列规则](#)》。

nWriteSizeWords 指定一次往物理缓冲区由 nWriteOffsetWords 参数指定偏移位置开始写入的数据长度。注意此参数的值与 nWriteOffsetWords 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。同时此参数值不能大于 DABuffer[] 指定的缓冲区的长度。

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

nDAChannel DA 通道号, 取值范围为[0, 1]。

返回值: 如果调用成功, 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 以批量方式将板载 RAM 中的 DA 数据回读至主机的用户缓冲区

函数原型:

Visual C++:

```
BOOL ReadDeviceBulkDA (HANDLE hDevice,
                        SHORT DABuffer[],
                        LONG nReadSizeWords,
                        PLONG nRetSizeWords,
                        int nDAChannel)
```

Visual Basic:

```
Declare Function ReadDeviceBulkDA Lib "PCI9100" (ByVal hDevice as Long, _
                                                ByRef DABuffer() As Integer, _
                                                ByVal nReadSizeWords As Long, _
                                                ByRef nRetSizeWords As Long, _
                                                ByVal nDAChannel As Integer) As Boolean
```

LabView:

请参考相关演示程序。

功能: 从指定通道的 RAM 中的指定段以及指定段内偏移位置开始将 DA 数据从板载 RAM 中读回至主机的用户缓冲区。但是在启动之后（即在输出过程中），不能对 RAM 进行读操作，包括写操作。该函数的作用是为了验证写入的数据是否正确而提供的。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

DABuffer[] 接受 DA 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 DA 数据转换成相应的电压值，请参考《[数据格式转换与排列规则](#)》。

nReadSizeWords 指定一次从物理缓冲区由 nReadOffsetWords 参数指定偏移位置开始读入的数据长度。注意此参数的值与 nWriteOffsetWords 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。同时此参数值不能大于 DABuffer[] 指定的缓冲区的长度。

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

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值: 如果调用成功，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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 启动 DA 设备

函数原型

Visual C++:

```
BOOL EnableDeviceDA (HANDLE hDevice,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function EnableDeviceDA Lib "PCI9100" (ByVal hDevice As Long, _
                                                ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能: 启动 DA 设备，它必须在调用 [InitDeviceDA](#) 后才能调用此函数。调用该函数后它可能立即启动，这就要取决您选择的触发方式或触发源。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

nDAChannel 通道号，取值范围为[0,1]。

返回值：如果调用成功，则返回TRUE，且DA准备就绪，等待触发事件的到来就开始实际的DA输出， 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆软件产生触发事件

函数原型：

Visual C++:

```
BOOL SetDeviceTrigDA (HANDLE hDevice,  
                      BOOL bSetSyncTrig,  
                      int nDAChannel)
```

Visual Basic:

```
Declare Function SetDeviceTrigDA Lib "PCI9100" (ByVal hDevice As Long, _  
                                              ByVal bSetSyncTrig As Boolean, _  
                                              ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：在指定通道被启动后，可由此函数以软件产生触发事件。当然只有当用户选择触发源为软件触发时方有效。

参数：

hDevice 设备对象句柄，它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

bSetSyncTrig 是否置同步触发。

nDAChannel 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，则产生一个软件触发事件，在某些触发模式下会直接使DA输出波形， 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆取得 DA 的状态标志

函数原型：

Visual C++:

```
BOOL GetDevStatusDA (HANDLE hDevice,  
                     PCI9100_STATUS_DA pDAStatus,  
                     int nDAChannel)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PCI9100" (ByVal hDevice As Long, _  
                                              ByRef pDAStatus As PCI9100_STATUS_DA, _  
                                              ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用 [EnableDeviceDA](#)后，可以用此函数查询DA状态，如是否被启动（bConverting），触发标志是否有效(bTrigFlag)，当前段循环次数(nCurSegLoopCount)等信息。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

pDAStatus 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《[DA硬件参数结构](#)》。

nDAChannel 通道号, 取值范围为[0, 1]。

返回值: 若 DA 成功取回标状态, 则返回 TRUE, 否则返回 FALSE。

相关函数:

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 暂停 DA 设备

函数原型

Visual C++:

```
BOOL DisableDeviceDA (HANDLE hDevice,
                      int nDAChannel )
```

Visual Basic:

```
Declare Function DisableDeviceDA Lib "PCI9100" (ByVal hDevice as Long, _
                                                ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能: 暂停DA设备。它必须在调用 [EnableDeviceDA](#)后才能调用此函数。该函数除了停止DA设备不再转换以外, 不改变设备的其他任何工作参数。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

nDAChannel 通道号, 取值范围为[0,1]。

返回值: 如果调用成功, 则返回TRUE, 且DA立刻停止转换, 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 释放 DA 设备

函数原型

Visual C++:

```
BOOL ReleaseDeviceDA (HANDLE hDevice,
                      int nDAChannel)
```

Visual Basic:

```
Declare Function ReleaseDeviceDA Lib "PCI9100" (ByVal hDevice as Long, _
                                                ByVal nDAChannel As Integer) As Boolean
```

LabView:

请参考相关演示程序。

功能: 释放DA设备。它必须在调用 [InitDeviceDA](#)后的某个时刻调用此函数。该函数除了停止DA设备, 还释放掉所占用的各种资源。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

nDAChannel 设备通道号, 取值范围为[0,1]。

返回值: 如果调用成功, 则返回TRUE, 且DA立刻停止转换, 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	SetDevTrigLevelDA	SetDevFrequencyDA
	ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
	WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
	SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
	ReleaseDeviceDA	ReleaseDevice	

◆ 采样和传输函数一般调用顺序

- ① [CreateDevice](#) (创建设备对象)
- ② [InitDeviceDA](#) (初始化设备)
- ③ [WriteDeviceBulkDA](#) (批量写入DA数据到板载RAM)
- ④ [EnableDeviceDA](#) (启动DA设备)
- ⑤ [SetDevTrigLevelDA](#) (若为软件触发源，则置软件触发事件)
- ⑥ [GetDevStatusDA](#) (循环查询DA状态)
- ⑦ [DisableDeviceDA](#)
- ⑧ [ReleaseDevice](#)

关于调用过程的图形说明请参考《[绪论](#)》。

第四节、DA 硬件参数系统保存与读取函数原型说明

◆ 写设备硬件参数函数到 Windows 系统中

函数原型:

Visual C++:

```
BOOL SaveParaDA (HANDLE hDevice,
                  PCI9100_PARA_DA pDAPara,
                  int nDAChannel)
```

Visual Basic:

```
Declare Function SaveParaDA Lib "PCI9100" (ByVal hDevice As Long, _
                                           ByRef pDAPara As PCI9100_PARA_DA, _
                                           ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

pDAPara设备硬件参数，关于PCI9100_PARA_DA的详细介绍请参考PCI9100.h或PCI9100.Bas或PCI9100.Pas函数原型定义文件，也可参考《[硬件参数结构](#)》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaDA](#) [SaveParaDA](#)
[ReleaseDevice](#)

◆ 从 Windows 系统中读入硬件参数函数

函数原型:

Visual C++:

```
BOOL LoadParaDA(HANDLE hDevice,
                 PCI9100_PARA_DA pDAPara,
                 int nDAChannel)
```

Visual Basic:

```
Declare Function LoadParaDA Lib "PCI9100" (ByVal hDevice As Long, _
                                           ByRef pDAPara As PCI9100_PARA_DA, _
```

ByVal nDACHannel As Integer) As Boolean

LabVIEW:

请参考相关演示程序。

功能: 负责从 Windows 系统中读取设备的硬件参数。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#)或 [CreateDeviceEx](#)创建。

pDAPara 属于 PCI9100_PARA_DA 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 PCI9100_PARA_DA 请参考 PCI9100.h 或 PCI9100.Bas 或 PCI9100.Pas 函数原型定义文件, 也可参考《[硬件参数结构](#)》关于该结构的有关说明。

nDACHannel DA 通道号, 取值范围为[0, 1]。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaDA](#) [SaveParaDA](#)
[ReleaseDevice](#)

◆ 将硬件参数结构体值复位为出厂默认值

函数原型:

Visual C++:

```
BOOL ResetParaDA (HANDLE hDevice,  
                  PCI9100_PARA_DA pDAPara,  
                  int nDACHannel)
```

Visual Basic:

```
Declare Function ResetParaDA Lib "PCI9100" (ByVal hDevice As Long, _  
                                             ByRef pDAPara As PCI9100_PARA_DA, _  
                                             ByVal nDACHannel As Integer) 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 pDAPara 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](#)或 [CreateDeviceEx](#)创建。

pDAPara 设备硬件参数, 关于 PCI9100_PARA_DA 的详细介绍请参考 PCI9100.h 或 PCI9100.Bas 或 PCI9100.Pas 函数原型定义文件, 也可参考《[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。

nDACHannel DA 通道号, 取值范围为[0, 1]。

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

相关函数: [CreateDevice](#) [LoadParaDA](#) [SaveParaDA](#)
[ResetParaDA](#) [ReleaseDevice](#)

第五节、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,  
                  BYTE bDISTs[8])
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI9100" (ByVal hDevice As Long, _  
                                             ByVal bDISTs (0 to 7) 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。

bDISts 八路数字量输入状态的参数结构，共有 8 个元素，分别对应于 DI0~DI7 路数字量输入状态位。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](#)

◆ **数字量输出**

函数原型:

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,  
                  BYTE bDOSts[8])
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI9100" (ByVal hDevice As Long, _  
                                           ByVal bDOSts(0 to 7) As Byte) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

bDOSts 八路数字量输出状态的参数结构，共有 8 个元素，分别对应于 DO0~DO7 路数字量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [GetDeviceDI](#) [ReleaseDevice](#)

◆ **回读数字量输出状态**

函数原型:

Visual C++:

```
BOOL RetDeviceDO (HANDLE hDevice,  
                  BYTE bDOSts[8])
```

Visual Basic:

```
Declare Function RetDeviceDO Lib "PCI9100" (ByVal hDevice As Long, _  
                                           ByVal bDOSts(0 to 7) As Byte) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。

bDISts 获得开关量输出状态。

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、DA 硬件参数介绍（PCI9100_PARA_DA）

Visual C++ & C++Builder:

```
typedef struct _PCI9100_PARA_DA
```

```
{
    LONG OutputRange;           // 输出量程控制
    LONG LoopCout;              // 整过 RAM 的大循环次数，=0 无限循环，=n 表示 n 次循环
    LONG TriggerMode;           // 触发模式选择
    LONG TriggerSource;         // 触发源选择
    LONG TriggerDir;            // 触发方向选择
    LONG bSingleOut;            // 是否单点输出
    LONG Frequency;             // 采集频率,单位为 Hz, [0.01, 80000000]
    LONG ClockSource;           // 时钟源选择
} PCI9100_PARA_DA, *PCI9100_PARA_DA;
```

Visual Basic:

Private Type PCI9100_PARA_DA

OutputRange As Long ' 输出量程控制

LoopCout As Long ' 整过 RAM 的大循环次数，=0 无限循环，=n 表示 n 次循环

TriggerMode As Long ' 触发模式选择

TriggerSource As Long ' 触发源选择

TriggerDir As Long ' 触发方向选择

bSingleOut As Long ' 是否单点输出

Frequency As Long ' 采集频率,单位为 Hz, [0.01, 80000000]

ClockSource As Long ' 时钟源选择

End Type

LabVIEW:

请参考相关演示程序。

此结构主要用于设定设备DA硬件参数值，用这个参数结构对设备进行硬件配置完全由 [InitDeviceDA](#)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。

[OutputRange](#) 模拟量输出量程选择。

常量名	常量值	功能定义
PCI9100_OUTPUT_N5000_P5000mV	0x00	± 5000mV
PCI9100_OUTPUT_N10000_P10000mV	0x01	± 10000mV

[TriggerMode](#) DA 触发模式选择。

常量名	常量值	功能定义
PCI9100_TRIGMODE_SINGLE	0x00	单次触发
PCI9100_TRIGMODE_CONTINUOUS	0x01	连续触发
PCI9100_TRIGMODE_STEPPED	0x02	单步触发
PCI9100_TRIGMODE_BURST	0x03	紧急触发

[TriggerSource](#) DA 触发源选择。

常量名	常量值	功能定义
PCI9100_TRIGSRC_SOFT_P5000mV	0x00	软件触发
PCI9100_TRIGSRCATR_DA	0x01	ATR 硬件模拟触发

bSingleOut DA 是否单点输出，=TRUE 表示单点输出，=FALSE 表示连续输出。

Frequency DA 的采样频率，其取值范围为[0.01, 80000000Hz]。

TriggerDir DA 触发方向。它的选项值如下表：

常量名	常量值	功能定义
PCI9100_TRIGDIR_NEGATIVE	0x00	负向触发（低脉冲/下降沿触发）
PCI9100_TRIGDIR_POSITIVE	0x01	正向触发（高脉冲/上升沿触发）
PCI9100_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：PCI9100_TRIGDIR_POSIT_NEGAT 在边沿类型下，则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 DA 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PCI9100_CLOCKSRC_IN	0x00	内部时钟
PCI9100_CLOCKSRC_OUT	0x01	外部时钟

第二节、DA 状态参数结构（PCI9100_STATUS_DA）

Visual C++:

```
typedef struct _PCI9100_STATUS_DA
{
    LONG bEnable;
    LONG bTrigFlag;
    LONG bConverting;
    LONG nCurSegNum;
    LONG nCurSegAddr;
    LONG nCurLoopCount;
    LONG nCurSegLoopCount;
} PCI9100__STATUS_DA, *PCI9100__STATUS_DA;
```

Visual Basic:

Private Type PCI9100__STATUS_DA

bEnable As Long

bTrigFlag As Long

bConverting As Long

nCurSegNum As Long

nCurSegAddr As Long

nCurLoopCount As Long

nCurSegLoopCount As Long

End Type

LabVIEW:

请参考相关演示程序。

此结构体主要用于查询DA的各种状态，[GetDevStatusDA](#)函数使用此结构体来实时取得DA状态，以便同步各种数据采集和处理过程。

bEnable DA 使能启动标志，=TRUE 表示 DA 已被使能，=FALSE 表示 DA 被禁止。

bTrigFlag 触发标志是否有效，=TRUE 表示触点标志有效，=FALSE 表示无效（即触发点未到）。

bConverting DA 是否正在转换，=TRUE 表示正在转换，=FALSE 表示转换完成。

nCurSegNum 可读取的 RAM 段号。

nCurSegAddr 可读取的 RAM 段地址。

nCurLoopCount 当前总循环次数。

nCurSegLoopCount 当前段循环次数。

相关函数: [CreateDevice](#) [GetDevStatusProDA](#) [ReleaseDevice](#)

第三节、DA 各段信息参数结构（PCI9100_SEGMENT_INFO）

```
Visual C++:  
typedef struct _PCI9100_SEGMENT_INFO  
{  
    LONG SegLoopCount;  
    LONG SegmentSize;  
} PCI9100__ SEGMENT_INFO, *PCI9100__ SEGMENT_INFO;
```

```
Visual Basic:  
Private Type PCI9100__ SEGMENT_INFO  
    SegLoopCount As Long  
    SegmentSize As Long  
End Type
```

LabVIEW:
请参考相关演示程序。

此结构体主要用于建立段信息，它包括段长、段循环等，它主要用于函数 [InitDeviceDA\(\)](#)的 SegmentInfo[]参数。

SegLoopCount 段循环次数，如果是多段输出，则首先循环输出段 0 的波形数据，直到段 0 循环到该参数指定的次数再跳转到段 1 直至循环结束再输出下一段，当然实际情况要看用户选择的触发模式。其取值范围为 20 位，即 [1,16777215]。

SegmentSize 段长，指的是该段的波形数据长度，单位点。其取值范围受该通道分配的板载 RAM 空间大小、总有效输出段数及各段长度决定。

第五章 数据格式转换与排列规则

第一节、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
±5000mV	Lsb=Volt/(10000.00/4096)+2048	[0, 4095]
±10000mV	Lsb=Volt/(20000.00/4096)+2048	[0, 4095]

第二节、关于 DA 数据 DABuffer 缓冲区中的数据排放规则

由于各个通道的段信息与波形数据均共享一个板载物理 RAM，它们的排放顺序如图 5.1，但各个通道所占 RAM 空间大小均可以通过函数 SetDevRamSizeDA()调整，系统默认值为四个通道均分整个 RAM 空间，即默认每通道 RAM 空间为 256K 点。从下图可以看出，各个通道所占 RAM 空间不一定相等，可大可小，只是四个通道的总空间不能大于板载物理 RAM 空间即可。

通道 0	通道 1	通道 2	通道 3
0 至(256K-1)	256 至（512K-1）	512 至（768K-1）	768 至（1024K-1）

关于每个通道 RAM 空间的内部分配是这样的，其空间首部存放的是所有段的段信息数据，其后才是各个段的波

形数据，再其后可能还有未用空间。假如有三个分段，如图 5.2:

段 0 信息	段 1 信息	段 2 信息	段 0 波形数据	段 1 波形数据	段 2 波形数据	未用空间
-----------	-----------	-----------	----------	----------	----------	------

关于每个段的段信息包括的内容有：该段波形数据在 RAM 中的起始地址、终止地址、段循环次数，如表 5.2.1，注意其段起始地址和终止地址是由 PCI9100_PARA_DA 中的 SegmentInfo 决定的。

板载 RAM 内存单元 (16Bit)	各单元定义	有效位
0	段 0 波形数据起始地址低 12 位	D11: D0
1	段 0 波形数据起始地址高 8 位	D7: D0
2	段 0 波形数据终止地址低 12 位	D11: D0
3	段 0 波形数据终止地址高 8 位	D7: D0
4	段 0 循环次数低 12 位	D11: D0
5	段 0 循环次数高 8 位	D7: D0
6	段 1 波形数据起始地址低 12 位	D11: D0
7	段 1 波形数据起始地址高 8 位	D7: D0
8	段 1 波形数据终止地址低 12 位	D11: D0
9	段 1 波形数据终止地址高 8 位	D7: D0
10	段 1 循环次数低 12 位	D11: D0
11	段 1 循环次数高 8 位	D7: D0
:	:	:
段信息结束后便是波形数据	段信息结束后便是波形数据	D11: D0

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

怎样使用 [WriteDeviceBulkDA](#) 函数取得 AD 数据

Visual C++:

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

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++] | [简易代码演示] | [DA 批量输出演示源程序]

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 PCI9100.h 和 ADDoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++] | [高级代码演示]

其默认存放路径为：系统盘\ART\PCI9100\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“PCI9100_”）

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
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	
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	
ReadFile	请求文件对象读数据到用户空间	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	
ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备
⑤ 各种参数保存和读取函数		
SaveParaInt	保存整型参数到注册表	
LoadParaInt	从注册表中读取整型参数值	
SaveParaString	保存字符参数到注册表	
LoadParaString	从注册表中读取字符参数值	
⑥ 其他函数		
kbhit	探测用户是否有击键动作	
getch	等待并获取用户击键值	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间
GetLastErrorEx	取得驱动函数错误信息	
RemoveLastErrorEx	移除函数错误信息	

第二节、PCI 内存映射寄存器操作函数原型说明

◆ 取得指定内存映射寄存器的线性地址和物理地址

函数原型:

Visual C++:

BOOL GetDeviceAddr (HANDLE hDevice,

PULONG MemCPLDBase,
PULONG MemDA0Buffer,
PULONG MemDA1Buffer)

Visual Basic:

Declare Function GetDeviceAddr Lib "PCI9100" (ByVal hDevice As Long, _
ByRef MemCPLDBase r As Long, _
ByRef MemDA0Buffer As Long, _
ByRef MemDA1Buffer As Long) As Boolean

LabVIEW:

请参考相关演示程序。

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。

MemCPLDBase 返回指定映射寄存器的线性地址。

MemDA0Buffer 返回指定映射寄存器的线性地址。

MemDA1Buffer 返回指定映射寄存器的线性地址。

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 [GetLastErrorEx](#) 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](#)

◆ 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,
 ULONG pulPCIBar[6])

Visual Basic:

Declare Function GetDeviceBar Lib "PCI9100" (ByVal hDevice As Long, _
 ByVal pulPCIBar (0 to 5) 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。

pulPCIBar 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++:

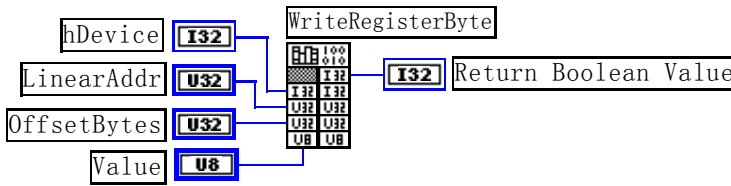
BOOL WriteRegisterByte (HANDLE hDevice,
 ULONG LinearAddr,
 ULONG OffsetBytes,
 BYTE Value)

Visual Basic:

Declare Function WriteRegisterByte Lib "PCI9100" (ByVal hDevice As Long, _
 ByVal LinearAddr As String, _

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。

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++:

```

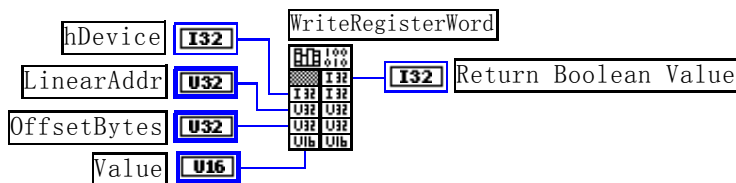
BOOL WriteRegisterWord (HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        WORD Value)

```

Visual Basic:

Declare Function WriteRegisterWord Lib "PCI9100" (ByVal hDevice As Long, _
ByVal LinearAddr As String, _
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](#) 创建。

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

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

Value 输出 16 位整型值。

返回值: 无。

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
[ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 程序举例:

```

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

```

Visual Basic 程序举例:

```

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

```

◆ 以四字节（即 32 位）方式写 PCI 内存映射寄存器的某个单元

函数原型:

Visual C++:

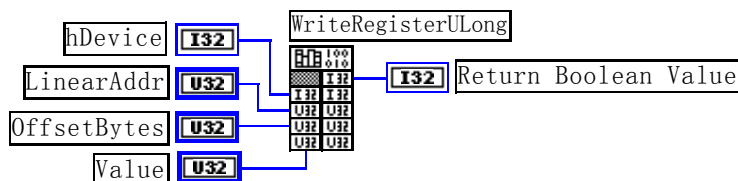
BOOL WriteRegisterULong (HANDLE hDevice,
ULONG LinearAddr,

ULONG OffsetBytes,
ULONG Value)

Visual Basic:

Declare Function WriteRegisterULong Lib "PCI9100" (ByVal hDevice As Long, _
ByVal LinearAddr As String, _
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。

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

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

Value 输出 32 位整型值。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
[ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 程序举例:

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

Visual Basic 程序举例:

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

◆ 以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元

函数原型:

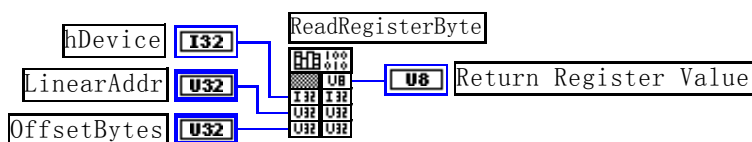
Visual C++:

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

Visual Basic:

Declare Function ReadRegisterByte Lib "PCI9100" (ByVal hDevice As Long, _
ByVal LinearAddr As String, _
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。

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, LinearAddr, 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:

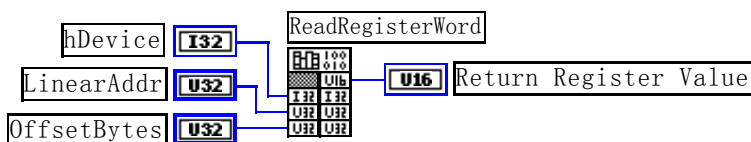
Visual C++:

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

Visual Basic:

**Declare Function ReadRegisterWord Lib "PCI9100" (ByVal hDevice As Long, _
ByVal LinearAddr As String, _
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](#) 创建。

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

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

返回值: 返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
[ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 程序举例:

```

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

```

Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr(hDevice, LinearAddr, 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:

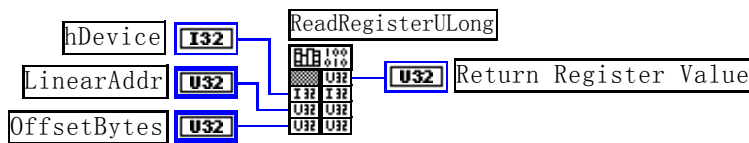
Visual C++:

**ULONG ReadRegisterULong (HANDLE hDevice,
ULONG LinearAddr,**

Visual Basic:

Declare Function ReadRegisterULong Lib "PCI9100" (ByVal hDevice As Long, _
ByVal LinearAddr As String, _
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。

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, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:
```

第三节、IO 端口读写函数原型说明

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

◆ 以单字节(8Bit)方式写 I/O 端口

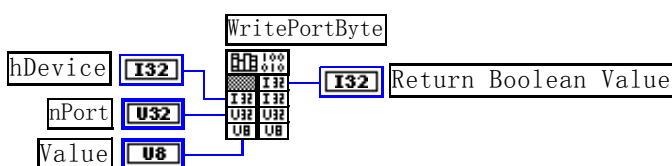
Visual C++:

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

Visual Basic:

```
Declare Function WritePortByte Lib "PCI9100" (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](#) 创建。

nPort 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

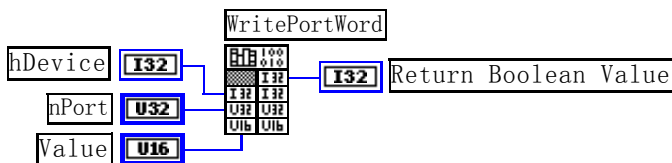
Visual C++:

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

Visual Basic:

```
Declare Function WritePortWord Lib "PCI9100" (ByVal hDevice As Long, _
                                             ByVal nPort As String, _
                                             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 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

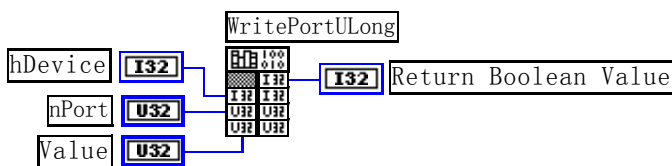
Visual C++:

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

Visual Basic:

```
Declare Function WritePortULong Lib "PCI9100" (ByVal hDevice As Long, _
                                             ByVal nPort As String, _
                                             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。

nPort 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

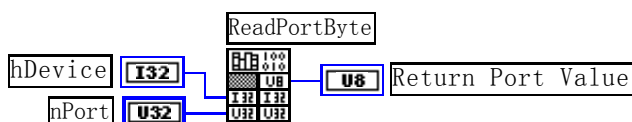
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortByte Lib "PCI9100" (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](#)创建。

nPort 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

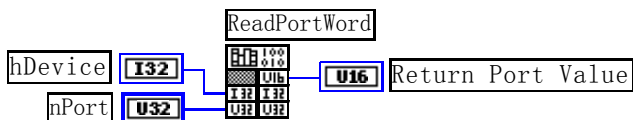
Visual C++:

WORD ReadPortWord (HANDLE hDevice,
UNIT nPort)

Visual Basic:

Declare Function ReadPortWord Lib "PCI9100" (ByVal hDevice As Long, _
ByVal nPort As LongWord) 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](#) 创建。

nPort 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

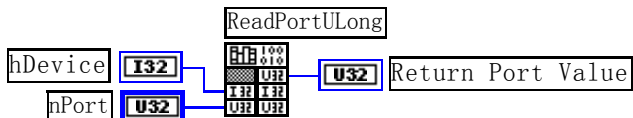
Visual C++:

ULONG ReadPortULong (HANDLE hDevice,
UNIT nPort)

Visual Basic:

Declare Function ReadPortULong Lib "PCI9100" (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 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:

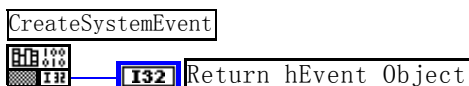
Visual C++:

HANDLE CreateSystemEvent (void)

Visual Basic:

Declare Function CreateSystemEvent Lib "PCI9100" () 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 "PCI9100" (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。

第五节、文件对象操作函数原型说明

◆ 创建文件对象

函数原型：

Visual C++:

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

Visual Basic:

Declare Function CreateFileObject Lib "PCI9100" (ByVal hDevice As Long, _
ByVal strFileName As String, _
ByVal Mode As Integer) As Long

LabVIEW:

请参见相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

strFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：“C:\PCI9100\Data.Dat”，在 Basic 中，其语法格式如：“C:\PCI9100\Data.Dat”。

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

常量名	常量值	功能定义
PCI9100_modeRead	0x0000	只读文件方式
PCI9100_modeWrite	0x0001	只写文件方式
PCI9100_modeReadWrite	0x0002	既读又写文件方式
PCI9100_modeCreate	0x1000	如果文件不存在可以创建该文件，如果存在，则重建此文件，且清 0
PCI9100_typeText	0x4000	以文本方式操作文件

返回值：若成功，则返回文件对象句柄。

相关函数：[CreateDevice](#) [CreateFileObject](#) [WriteFile](#)
[ReadFile](#) [ReleaseFile](#) [ReleaseDevice](#)

◆ 通过设备对象, 往指定磁盘上写入用户空间的采样数据

函数原型:

Visual C++:

```
BOOL WriteFile(HANDLE hFileObject,
               PVOID pDataBuffer,
               LONG nWriteSizeBytes)
```

Visual Basic:

```
Declare Function WriteFile Lib "PCI9100" (ByVal hFileObject As Long, _
                                         ByRef pDataBuffer As Byte, _
                                         ByVal nWriteSizeBytes As Long) As Boolean
```

LabVIEW:

详见相关演示程序。

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

参数:

hFileObject 设备对象句柄, 它应由 [CreateFileObject](#) 创建。

pDataBuffer 用户数据空间地址, 可以是用户分配的数组空间。

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

返回值: 若成功, 则返回TRUE, 否则返回FALSE, 用户可以用 [GetLastErrorEx](#) 捕获错误码。

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ 通过设备对象, 从指定磁盘文件中读采样数据

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function ReadFile Lib "PCI9100" ( ByVal hFileObject As Long, _
                                         ByRef pDataBuffer As Integer, _
                                         ByVal OffsetBytes As Long, _
                                         ByVal nReadSizeBytes As Long) As Boolean
```

LabVIEW:

详见相关演示程序。

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

参数:

hFileObject 设备对象句柄, 它应由 [CreateFileObject](#) 创建。

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

OffsetBytes 指定从文件开始端所偏移的读位置。

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

返回值: 若成功, 则返回TRUE, 否则返回FALSE, 用户可以用 [GetLastErrorEx](#) 捕获错误码。

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ 设置文件偏移位置

函数原型:

Visual C++:

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

Visual Basic:

**Declare Function SetFileOffset Lib "PCI9100" (ByVal hFileObject As Long, _
ByVal nOffsetBytes As Long) As Boolean**

LabVIEW:

详见相关演示程序。

功能: 设置文件偏移位置，用它可以定位读写起点。

参数:

hFileObject 文件对象句柄，它应由 [CreateFileObject](#) 创建。

nOffsetBytes 文件偏移位置（以字为单位）。

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

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ **取得文件长度（字节）**

函数原型:

Visual C++:

ULONG GetFileLength (HANDLE hFileObject)

Visual Basic:

Declare Function GetFileLength Lib "PCI9100" (ByVal hFileObject As Long) As Long

LabVIEW:

详见相关演示程序。

功能: 取得文件长度。

参数:

hFileObject 设备对象句柄，它应由 [CreateFileObject](#) 创建。

返回值: 若成功，则返回>1，否则返回 0，用户可以用 GetLastErrorEx捕获错误码。

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ **释放设备文件对象**

函数原型:

Visual C++:

BOOL ReleaseFile (HANDLE hFileObject)

Visual Basic:

Declare Function ReleaseFile Lib "PCI9100" (ByVal hFileObject As Long) As Boolean

LabVIEW:

详见相关演示程序。

功能: 释放设备文件对象。

参数:

hFileObject 设备对象句柄，它应由 [CreateFileObject](#) 创建。

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

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ **取得指定磁盘的可用空间**

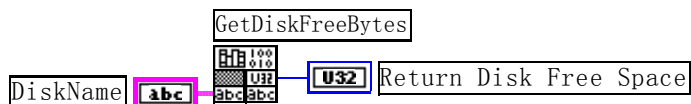
Visual C++:

ULONGLONG GetDiskFreeBytes (LPCTSTR DiskName)

Visual Basic:

Declare Function GetDiskFreeBytes Lib "PCI9100" (ByVal DiskName As String) As Currency

LabVIEW:



功能: 取得指定磁盘的可用剩余空间(以字为单位)。

参数:

DiskName 需要访问的盘符, 若为 C 盘为"C:\", D 盘为"D:\", 以此类推。

返回值: 若成功, 返回大于或等于 0 的长整型值, 否则返回零值, 用户可用 [GetLastErrorEx](#)捕获错误码。注意使用 64 位整型变量。

第六节、各种参数保存和读取函数原型说明

◆ 将整型变量的参数值保存在系统注册表中

函数原型:

Visual C++:

```
BOOL SaveParaInt (HANDLE hDevice,
                  LPCTSTR strParaName,
                  int nValue)
```

Visual Basic:

```
Declare Function SaveParaInt Lib "PCI9100" (ByVal hDevice As Long, _
                                           ByVal strParaName As String, _
                                           ByVal nValue As Integer) As Boolean
```

LabVIEW:

详见相关演示程序。

功能: 将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为: HKEY_CURRENT_USER\Software\Art\PCI9100\Device-0\Others。

参数:

hDevice设备对象句柄, 它应由 [CreateDevice](#)创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

返回值: 若成功, 则返回TRUE, 否则返回FALSE, 用户可以用 [GetLastErrorEx](#)捕获错误码。

相关函数: [SaveParaInt](#) [LoadParaInt](#) [SaveParaString](#)
[LoadParaString](#)

◆ 将整型变量的参数值从系统注册表中读出

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function LoadParaInt Lib "PCI9100" (ByVal hDevice As Long, _
                                           ByVal strParaName As String, _
                                           ByVal nDefaultVal As Integer) As Long
```

LabVIEW:

详见相关演示程序。

功能: 将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为: HKEY_CURRENT_USER\Software\Art\PCI9100\Device-0\Others。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

相关函数: [SaveParaInt](#) [LoadParaInt](#) [SaveParaString](#)
 [LoadParaString](#)

◆ 将字符变量的参数值保存在系统注册表中

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function SaveParaString Lib "PCI9100" (ByVal hDevice As Long, _  
                                              ByVal strParaName As String, _  
                                              ByVal strParaVal As String) As Boolean
```

LabVIEW:

详见相关演示程序。

功能: 将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为: HKEY_CURRENT_USER\Software\Art\PCI9100\Device-0\Others。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

相关函数: [SaveParaInt](#) [LoadParaInt](#) [SaveParaString](#)
 [LoadParaString](#)

◆ 将字符变量的参数值从系统注册表中读出

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function LoadParaString Lib "PCI9100" (ByVal hDevice As Long, _  
                                              ByVal strParaName As String, _  
                                              ByVal strParaVal As String, _  
                                              ByVal strDefaultVal As String) As Boolean
```

LabVIEW:

详见相关演示程序。

功能: 将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为: HKEY_CURRENT_USER\Software\Art\PCI9100\Device-0\Others。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

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

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

相关函数: [SaveParaInt](#) [LoadParaInt](#) [SaveParaString](#)
[LoadParaString](#)

第七节、其他函数原型说明

◆ 探测用户是否有按键动作

函数原型:

Visual C++:

BOOL kbhit (void)

Visual Basic:

Declare Function kbhit Lib "PCI9100" () As Boolean

LabVIEW:

详见相关演示程序。

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

参数: 无。

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

相关函数: [getch](#) [kbhit](#)

◆ 等待按键动作并返回按键值

函数原型:

Visual C++:

char getch (void)

Visual Basic:

Declare Function getch Lib "PCI9100" () As String

LabVIEW:

详见相关演示程序。

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

参数: 无。

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

相关函数: [getch](#) [kbhit](#)

◆ 高效高精度延时

函数原型:

Visual C++:

BOOL DalayTimeUs (HANDLE hDevice,
LONG nTimeUs)

Visual Basic:

Declare Function DalayTimeUs Lib "PCI9100" (ByVal hDevice As Long, _
ByVal nTimeUs As Long) As Boolean

LabVIEW:

详见相关演示程序。

功能: 将当某个驱动函数出错时, 可以调用此函数获得具体的错误和错误信息字符串。

参数:

hDevice 设备对象句柄。

nTimeUs 时间常数, 单位 1 微秒。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE, 用户可用 [GetLastErrorEx](#) 捕获错误码。

◆ 怎样获取驱动函数错误信息

函数原型:

Visual C++:

DWORD GetLastErrorEx (LPCTSTR strFuncName ,
LPTSTR strErrorMsg)

Visual Basic:

Declare Function GetLastErrorEx Lib "PCI9100" (ByVal strFuncName As String, _
ByVal strErrorMsg As String) As Long

LabVIEW:

详见相关演示程序。

功能: 将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字符串。

参数:

strFuncName 出错函数的名称。注意此函数必须是完整名称。如 AD 初始化函数 PCI9100_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI9100_InitDeviceAD”，否则得不到相应信息。

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

返回值: 返回错误码。

相关函数: 无。

◆ 移除驱动函数错误信息

函数原型:

Visual C++:

BOOL RemoveLastErrorEx (LPCTSTR strFuncName)

Visual Basic:

Declare Function RemoveLastErrorEx Lib "PCI9100" (ByVal strFuncName As String) As Boolean

LabVIEW:

详见相关演示程序。

功能: 从错误信息库中移除指定函数的最后一次错误信息。

参数:

strFuncName 出错函数的名称。注意此函数必须是完整名称。如 AD 初始化函数 PCI9100_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI9100_InitDeviceAD”，否则得不到相应信息。

返回值: 若成功，返回 TRUE，否则返回 FALSE，用户可用 [GetLastErrorEx](#) 捕获错误码。

相关函数: 无。