# PCI8932　WIN2000/XP 驱动程序使用说明书

**请您务必阅读《使用纲要》，他会使您事半功倍!**

## 目　　录



**提醒用户：**

通常情况下，WINDOWS 系统在安装时自带的 DLL 库和驱动不全，所以您不管使用那种语言编程，请您最好先安装上 Visual C++6.0 版本的软件，方可使我们的驱动程序有更完备的运行环境。

**有关设备驱动安装和产品二次发行请参考 PCI8932Inst.doc 文档。**

# 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市阿尔泰科贸有限公司所有，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 PCI2815_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、ReadDeviceAD、SetDeviceDO等。而底层用户函数如WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。

## 第二节、如何管理 PCI 设备

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

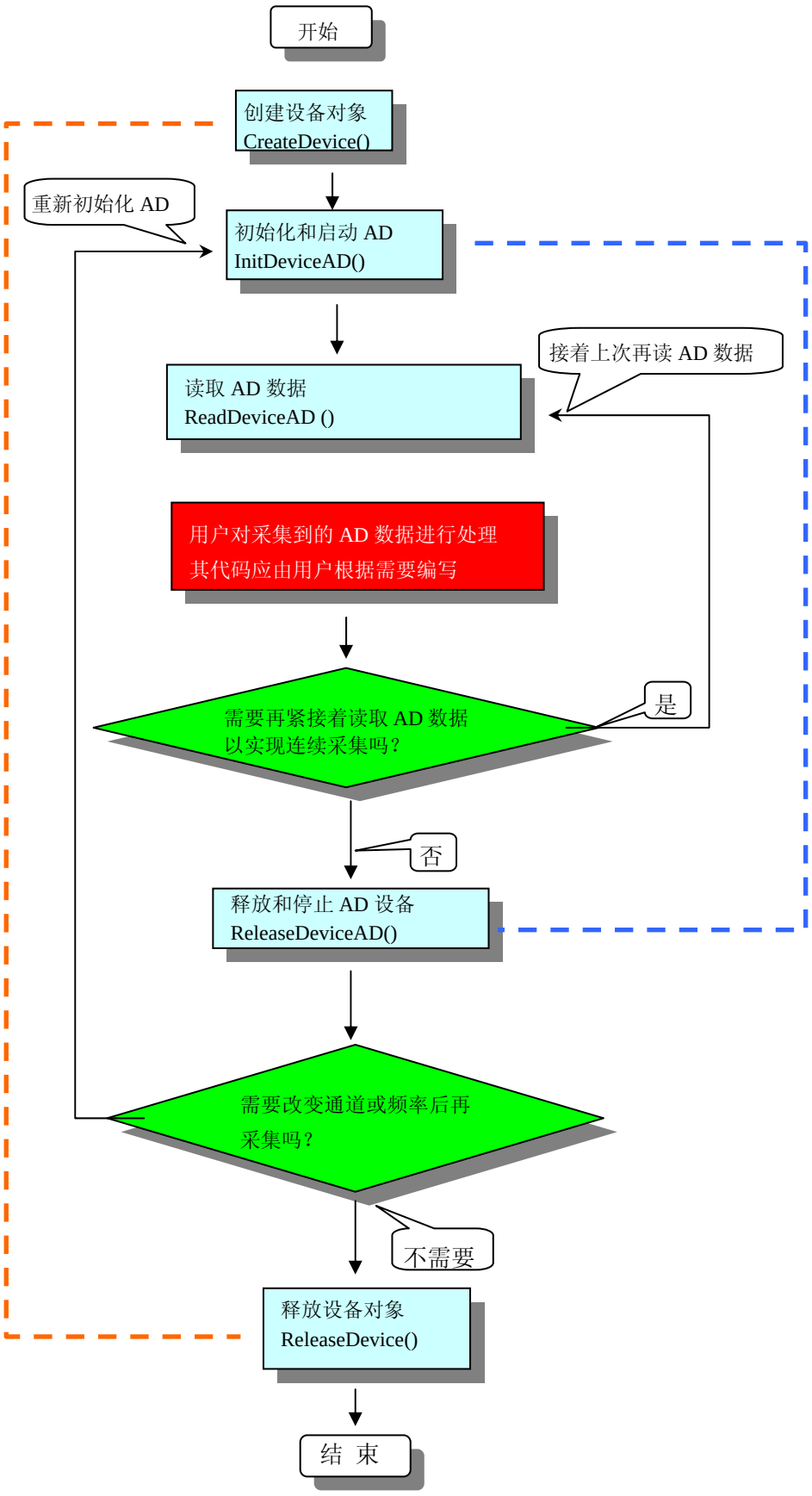
注意：图中较粗的虚线表示对称关系。如红色虚线表示CreateDevice和ReleaseDevice两个函数的关系是：最初执行一次CreateDevice，在结束是就须执行一次ReleaseDevice。

## 第三节、如何实现开关量的简便操作

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

## 第四节、哪些函数对您不是必须的

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


开始
创建设备对象
CreateDevice()
重新初始化 AD
初始化和启动 AD
InitDeviceAD()
接着上次再读 AD 数据
读取 AD 数据
ReadDeviceAD ()
用户对采集到的 AD 数据进行处理
其代码应由用户根据需要编写
需要再紧接着读取 AD 数据
以实现连续采集吗？
是
否
释放和停止 AD 设备
ReleaseDeviceAD()
需要改变通道或频率后再
采集吗？
不需要
释放设备对象
ReleaseDevice()
结 束

AD 采集实现过程

# 第三章　PCI 设备专用函数接口介绍

## 第一节、设备驱动接口函数列表（每个函数省略了前缀“PCI8932_”）

**本章函数是设备使用 PCI 方式传输时所使用的。**

| 函数名 | 函数功能 | 备注 |
|---|---|---|
| ***① 设备对象操作函数*** | | |
| CreateDevice | 创建 PCI 对象(用设备逻辑号) | |
| CreateDeviceEx | 创建 PCI 对象(用设备物理号) | 上层及底层用户 |
| GetDeviceCount | 取得设备总数 | |
| GetDeviceCurrentID | 取得设备当前 ID 号 | |
| ListDeviceDlg | 列表所有同一种 PCI 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 | |
| ***② AD 采样操作函数*** | | |
| InitDeviceAD | 初始化 PCI 设备 AD 部件，准备传数 | |
| ReadDeviceAD | 连续批量读取 PCI 设备上的 AD 数据 | |
| ReleaseDeviceAD | 释放 PCI 设备对象中的 AD 部件 | |
| ***③ 辅助函数（硬件参数设置、保存、读取函数）*** | | |
| LoadParaAD | 从 Windows 系统中读取硬件参数 | |
| SaveParaAD | 往 Windows 系统保存硬件参数 | |
| ResetParaAD | 将注册表中的 AD 参数恢复至出厂默认值 | 上层用户 |
| ***④ DA 输出操作函数*** | | |
| WriteDeviceDA | 输出模拟信号到指定通道 | |
| ***⑤计数器操作函数*** | | |
| SetDeviceCNT | 设置计数器的初值 | |
| GetDeviceCNT | 取得各路计数器的当前计数值 | |
| ***⑥开关量函数*** | | |
| GetDeviceDI | 开关输入函数 | |
| SetDeviceDO | 开关输出函数 | |
| RetDeviceDO | 回读开关量输出状态 | |

**使用需知**

***Visual C++ & C++Builder：***

首先将 PCI8932.h 和 PCI8932.lib 两个驱动库文件从相应的演示程序文件夹下复制到您的源程序文件夹中，然后在您的源程序头部添加如下语句，以便将驱动库函数接口的原型定义信息和驱动接口导入库(PCI8932.lib)加入到您的工程中。

#include “PCI8932.H”

在 VC 中，为了使用方便，避免重复定义和包含，您最好将以上语句放在 StdAfx.h 文件。一旦完成了以上工作，那么使用设备的驱动程序接口就跟使用 VC/C++Builder 自身的各种函数，其方法一样简单，毫无二别。

关于 PCI8932.h 和 PCI8932.lib 两个文件均可在演示程序文件夹下面找到。

***Visual Basic:***

首先将 PCI8932.Bas 驱动模块头文件从 VB 的演示程序文件夹下复制到您的源程序文件夹中，然后将

此模块文件加入到您的 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 PCI8932.Bas 模块文件即可，一旦完成以上工作后，那么使用设备的驱动程序接口就跟使用 VB 自身的各种函数，其方法一样简单，毫无二别。

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，我们不保证能完全顺利运行。

***Delphi:***

首先将 PCI8932.Pas 驱动模块头文件从 Delphi 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单,执行其中的"Project Manager"命令,在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PCI8932.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCI8932”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PCI8932;  // 注意： 在此加入驱动程序接口单元 PCI8932
```

***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 的驱动程序接口的详细说明请参考其演示源程序。

## 第二节、设备对象管理函数原型说明

### ♦ 创建设备对象函数

函数原型：

***Visual C++ & C++ Builder：***

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

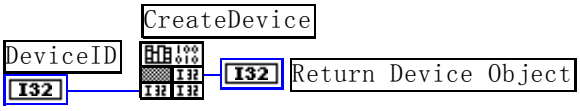
***Visual Basic：***

```
Declare Function CreateDevice Lib "PCI8932" (Optional ByVal DeviceLgcID As Long = 0) As Long
```

***Delphi：***

```
Function CreateDevice(DeviceLgcID:Integer = 0):Integer; StdCall; External 'PCI8932' Name 'CreateDevice';
```

***LabView:***




功能：该函数负责创建设备对象，并返回其设备对象句柄。

参数：

DeviceLgcID 设备逻辑 ID( Identifier )标识号。当向同一个 Windows 系统中加入若干相同类型的 PCI 设备时，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。比如若用户往 Windows 系统中加入第一个 PCI8932 AD 模板时，系统则以“PCI8932”作为基本名称，再以 DeviceLgcID 的初值组合成该设备的标识符“PCI8932-0”来确认和管理这第一个设备，若用户接着再添加第二个 PCI8932 AD

模板时，则系统将以“PCI8932-1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 PCI 设备时，DeviceLgcID 应置 0，第二应置 1，也以此类推。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：**ReleaseDevice

- **创建设备对象函数(扩展函数)**

函数原型：

***Visual C++ & C++ Builder：***

```
HANDLE CreateDeviceEx(int DevicePhysID = 0)
```

***Visual Basic：***

```
Declare Function CreateDeviceEx Lib "PCI8932" (ByVal DevicePhysID As Long = 0) As Long
```

***Delphi：***

```
Function CreateDeviceEx(DevicePhysID:Integer = 0):Integer;
                    StdCall; External 'PCI8932' Name 'CreateDeviceEx';
```

***LabView:***

请参考相关演示程序。

功能：该函数负责创建设备对象，并返回其设备对象句柄。设备对象句柄是访问某一台设备的唯一依据。不同 DevicePhysID 创建的设备对象句柄用于访问不同的设备。只有成功创建 DevicePhysID 指定设备的句柄后，设备对用户来讲才是可用的。因为每一个访问设备的驱动函数接口都需要 hDevice 这个设备句柄参数。

参数：

DevicePhysID 设备物理 ID( Identifier )标识号。如果您使用单个 PCI8932 设备，该参数等于 0 即可，且下面的内容您不必阅读。但如果您需要多卡工作时，此参数便大有用武之地，请阅读下以内容。

假如您所选用的产品只有 32 个 AD 通道，而您需要 128 路信号，那么您就需要同时使用四块卡来实现此功能。具体办法是您需要将 128 路信号依次分配到四个卡上，如下表：

| 卡号 | 信号通道 | 物理 ID 号 | 逻辑 ID 号 |
|---|---|---|---|
| 第一块 | 1～32 | 0 | 若第二个加载此卡，则为 1，否则为其他号 |
| 第二块 | 33～64 | 1 | 若最先加载此卡，则为 0，否则为其他号 |
| 第三块 | 65～96 | 2 | 若第四个加载此卡，则为 3，否则为其他号 |
| 第四块 | 97～128 | 3 | 若第三个加载此卡，则为 2，否则为其他号 |

从上表可知，如果使用您物理ID号，那么不管怎么安装您的硬件设备，其卡的物理编址不会发生任何变化，在软件上您用相应的物理ID号创建的设备对象所访问设备在物理顺序上不会发生变化，它始终对应于您的事先分配的信号通道。而使用逻辑ID号则不然，同样的逻辑ID值可能在不同的拔插顺序下会指向不同的物理设备而使软件的通道序列与硬件上无法一一对应。为了更好的保证物理通道序列与软件通道序列的对应关系，请务必保证板上的物理ID设置不重号。否则，CreateDevice ()只能创建重号中逻辑号最小的一个设备的对象，且具体创建的是哪一个设备也只能由用户根据采样信号特征等大概地确定。需要注意的是，物理ID号的设置有限的。如果实际设备数量超过这个有限值，那么超过的部分的物理ID号均设置为最大物理ID号，在创建设备对象时自动按逻辑ID处理。比如该产品的物理ID最多只能管理 16 个设备(ID值为 0～15)，如果您要使用 18 个设备，那么为第 17、18 个设备创建设备对象时，其DevicePhysID应分别等于 16、17。

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：**ReleaseDevice

***Visual C++ & C++Builder 程序举例***

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

***Visual Basic 程序举例***

```
        :

Dim hDevice As Long   ' 定义设备对象句柄
hDevice = CreateDevice(0)   ' 创建设备对象,并取得设备对象句柄，管理第一个 PCI 设备
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效

Else

   Exit Sub       ' 退出该过程
End If
        :
```

◆ **取得在系统中的设备总台数**

函数原型：

***Visual C++ & C++Builder:***

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

***Visual Basic:***

```
Declare Function GetDeviceCount Lib "PCI8932" (ByVal hDevice As Long ) As Integer
```

***Delphi:***

```
Function GetDeviceCount (hDevice : Integer): Integer;
                                StdCall; External 'PCI8932' Name ' GetDeviceCount ';
```

***LabView:***

请参考相关演示程序。

功能：取得在系统中物理设备的总台数。

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

返回值：若成功，则返回实际设备台数，否则返回 0，用户可以用 GetLastError 捕获错误码。

**相关函数**：CreateDevice　　ReleaseDevice

◆ **取得当前设备对象句柄指向的设备所在的设备 ID**

函数原型：

***Visual C++ & C++Builder:***

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

***Visual Basic:***

```
Declare Function GetDeviceCurrentID Lib "PCI8932" (ByVal hDevice As Long,_
                                                   ByRef DeviceLgcID As Long,_
                                                   ByRef DevicePhysIDAs Long ) As Boolean
```

***Delphi：***

```
Function GetDeviceCurrentID (hDevice : Integer;
                                DeviceLgcID : Pointer;
                                DevicePhysID : Pointer) : Boolean;
```

StdCall; External 'PCI8932' Name ' GetDeviceCurrentID ';

***LabView:***

请参考相关演示程序。

功能：取得指定设备对象所代表的设备在设备链中的物理设备 ID 号和逻辑 ID 号。

参数：

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

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

DevicePhysID 指针参数，取得指定设备的物理 ID。该号可以由用户设置板上的拔码器 DID1 获得。当多卡工作时，该物理 ID 号能让用户准确的辨别每一个卡所处的编址。该编址除了用户能手动改变以外，不会随着系统加载卸载设备的顺序或者是用户插拔设备的顺序而改变其相应的物理编址，它只会改变其逻辑编址，即 DeviceLgcID 的值。比如您使用某一产品，该产品只有 32 个 AD 通道，而您需要 128 个通道采样，那么您就需要四块卡来实现此功能。在实际采样中，您需要对将 128 路信号依次分配到四个卡上，如下表：

| 卡号 | 信号通道 | 物理 ID 号 | 逻辑 ID 号 |
|---|---|---|---|
| 第一块 | 1～32 | 0 | 若第二个加载此卡，则为 1，否则为其他号 |
| 第二块 | 33～64 | 1 | 若最先加载此卡，则为 0，否则为其他号 |
| 第三块 | 65～96 | 2 | 若第四个加载此卡，则为 3，否则为其他号 |
| 第四块 | 97～128 | 3 | 若第三个加载此卡，则为 2，否则为其他号 |

从上表可知，如果使用您物理 ID 号，那么不管怎么安装您的硬件设备，那么其卡的物理编址不会发生任何变化，在软件上您用相应的物理 ID 号创建的设备对象所访问设备在物理上不会发生变化，它始终对应于您的事先分配的信号通道。而使用逻辑 ID 号则不然，同样的逻辑 ID 值可能在不同的拔插顺序下会指向不同的物理设备而使软件的通道序列与硬件上无法一一对应。

返回值：若成功，则返回由hDevice参数代表的设备在设备链中的设备ID， 否则返回-1，用户可以用GetLastError捕获错误码。注意其返回的ID是一定与在CreateDevice函数中指定的DeviceLgcID参数值相等。

**相关函数：**CreateDevice　　ReleaseDevice

- **用对话框控件列表计算机系统中所有 PCI8932 设备各种配置信息**

函数原型：

***Visual C++ & C++Builder:***

BOOL ListDeviceDlg (HANDLE hDevice)

***Visual Basic***:

Declare Function ListDeviceDlg Lib "PCI8932" (ByVal hDevice As Long ) As Boolean

***Delphi：***

Function ListDeviceDlg (hDevice : Integer) : Boolean;

StdCall; External 'PCI8932' Name ' ListDeviceDlg ';

***LabVIEW：***

请参考相关演示程序。

功能：列表系统中 PCI8932 的硬件配置信息。

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

返回值：若成功，则弹出对话框控件列表所有 PCI8932 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++ & C++Builder:***

BOOL ReleaseDevice(HANDLE hDevice)

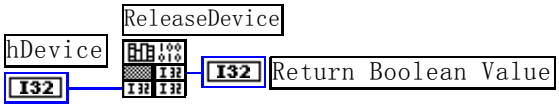
***Visual Basic:***

Declare Function ReleaseDevice Lib "PCI8932" (ByVal hDevice As Long ) As Boolean

***Delphi:***

Function ReleaseDevice(hDevice : Integer):Boolean; StdCall; External 'PCI8932' Name 'ReleaseDevice';

***LabView:***




功能：释放设备对象所占用的系统资源及设备对象自身。

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

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

**相关函数：**CreateDevice

应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice，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，如系统内存等。只有这样，当您再次调用CreateDevice函数时，那些软硬件资源才可被再次使用。

## 第三节、AD 采样操作函数原型说明

◆ **初始化设备对象**

函数原型:

***Visual C++ & C++Builder：***

BOOL InitDeviceAD( HANDLE hDevice,
PPCI8932_PARA_AD pADPara )

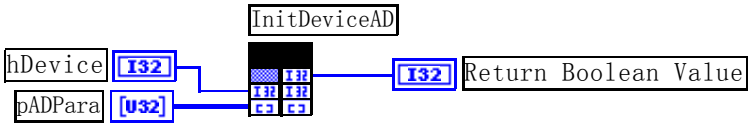
***Visual Basic:***

Declare Function InitDeviceAD Lib "PCI8932" (ByVal hDevice As Long, _
ByRef pADPara As PCI8932_PARA _AD) As Boolean

***Delphi:***

Function InitDeviceAD( hDevice : Integer; pADPara:PPCI8932_PARA_AD):Boolean;
StdCall; External 'PCI8932' Name 'InitDeviceAD';

***LabView:***




功能：它负责初始化设备对象中的AD部件,为设备操作就绪有关工作,如预置AD采集通道,采样频率等,然后启动AD设备开始AD采集，随后，用户便可以连续调用ReadDeviceAD读取PCI设备上的AD数据以实现连续采集。

参数：

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

pADPara 设备对象参数结构，它决定了设备对象的各种状态及工作方式,如AD采样通道、采样频率等。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GetLastError捕获当前错误码,并加以分析。

**相关函数：** CreateDevice ReadDeviceAD ReleaseDevice

***注意：该函数将试图占用系统的某些资源，如系统内存区、DMA资源等。所以当用户在反复进行数据采集之前，只须执行一次该函数即可，否则某些资源将会发生使用上的冲突，便会失败。除非用户执行了ReleaseDeviceAD函数后，再重新开始设备对象操作时，可以再执行该函数。所以该函数切忌不要单独放在循***

*环语句中反复执行，除非和ReleaseDeviceAD配对。*

♦ **批量读取 PCI 设备上的 AD 数据**

函数原型：

***Visual C++ & C++Builder:***

```
BOOL ReadDeviceAD (HANDLE hDevice,
                   WORD ADBuffer[],
                   LONG nReadSizeWords,
                   PLONG nRetSizeWords)
```

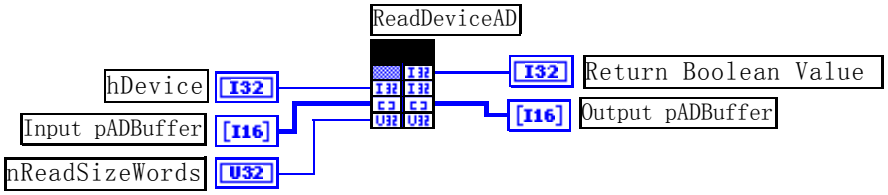
***Visual Basic:***

```
Declare Function ReadDeviceAD Lib "PCI8932" (ByVal hDevice As Long, _
                                             ByRef ADBuffer As Integer, _
                                             ByVal nReadSizeWords As Long, _
                                             Optional ByRef nRetSizeWords As Long) As Boolean
```

***Delphi:***

```
 Function ReadDeviceAD( hDevice : Integer;
                        ADBuffer : Word;
                        nReadSizeBytes:LongWord;
                        nRetSizeWords : Pointer) : Boolean;
                        StdCall; External 'PCI8932' Name 'ReadDeviceAD';
```

***LabView:***




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

参数：

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

ADBuffer 用户数据缓冲区地址。接受的是从设备上采集的LSB原码数据，关于如何将LSB原码数据转换成电压值，请参考《数据格式转换与排列规则》章节。

nReadSizeWords 读取数据的长度（以字为单位），为了提高读取速率，根据特定要求，其长度必须指定为 256 字的整数倍长，如 256、512、1024 …… 8192 等字长，同时，数据长度也要为采样通道数的整数倍，以便于通道数据对齐处理，所以 nReadSizeWords 为(256*(LastChannel － FirstChannel + 1))的整数倍。否则，PCI 设备对象将失败该读操作。

nRetSizeWords 在当前操作中该函数实际读取的点数。只有当函数成功返回时该参数值才有意义，而当函数返回失败时，则该参数的值与调用此函数前的值相等，不会因为函数被调用而改变，因此最好在读取 AD 数据前，将此参数值赋初值 0。需要注意的是在函数成功返回后，若此参数值等于 0，则需要重新调用此函数读取 AD 数据，直到此参数的值不等于 0 为止。

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

**相关函数:** CreateDevice InitDeviceAD ReleaseDevice

♦ **释放设备对象中的 AD 部件**

函数原型：

***Visual C++ & C++Builder:***

BOOL ReleaseDeviceAD(HANDLE hDevice)

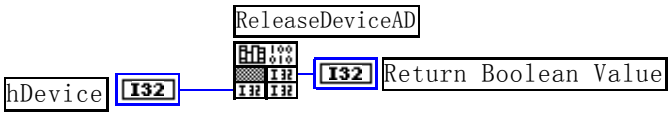
***Visual Basic:***

Declare Function ReleaseDeviceAD Lib "PCI8932" (ByVal hDevice As Long ) As Boolean

***Delphi:***

Function ReleaseDeviceAD(hDevice : Integer):Boolean;
StdCall; External 'PCI8932' Name 'ReleaseDeviceAD';

***LabView:***




功能：释放设备对象中的 AD 部件所占用的系统资源。

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

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

**相关函数**：CreateDevice InitDeviceAD ReleaseDevice

应注意的是，InitDeviceAD必须和ReleaseDeviceAD函数一一对应，即当您执行了一次InitDeviceAD，再一次执行这些函数前，必须执行一次ReleaseDeviceAD函数，以释放由InitDeviceAD占用的系统软硬件资源，如系统内存等。只有这样，当您再次调用InitDeviceAD函数时，那些软硬件资源才可被再次使用。这个对应关系对于非连续采样的场合特别适用。比如用户先采集一定长度的数据后，然后对根据这些数据或其他条件，需要改变采样通道或采样频率等配置时，则可以先用ReleaseDeviceAD释放先已由InitDeviceAD占用的资源，然后再用InitDeviceAD重新分配资源和初始化设备状态，即可实现所提到的功能。

❖ **以上函数调用一般顺序**

① CreateDevice
② InitDeviceAD
③ ReadDeviceAD
④ ReleaseDeviceAD
⑤ ReleaseDevice

用户可以反复执行第③步，以实现高速连续不间断数据采集。如果在采集过程中要改变设备状态信息，如采样通道等，则执行到第④步后再回到第②步用新的状态信息重新初始设备。

注意在第③步中，若其ReadDeviceAD函数成功返回，且nRetSizeWords参数值等于 0，则需要重新执行第③步，直到不等于 0 为止。

## 第四节、AD 硬件参数系统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++ & C++Builder:***

BOOL LoadParaAD(HANDLE hDevice, PPCI8932_PARA_AD pADPara)

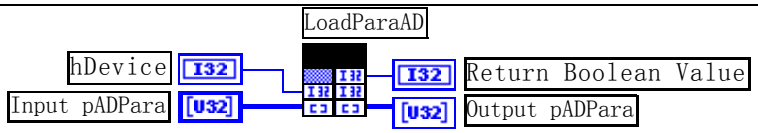
***Visual Basic:***

Declare Function LoadParaAD Lib "PCI8932" (ByVal hDevice As Long, _
ByRef pADPara As PCI8932_PARA_AD) As Boolean

***Delphi:***

Function LoadParaAD(hDevice : Integer; pADPara:PPCI8932_PARA_AD):Boolean;
StdCall; External 'PCI8932' Name 'LoadParaAD';

***LabView:***




功能：负责从 Windows 系统中读取设备硬件参数。

参数：

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

pADPara 属于 PPCI8932_PARA 的结构指针型，它负责返回 PCI 硬件参数值，关于结构指针类型 PPCI8932_PARA 请参考相应 PCI8932.h 或该结构的帮助文档的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数:** CreateDevice SaveParaAD ReleaseDevice

***Visual C++ & C++Builder 举例:***

```
        :
PCI8932_PARA_AD ADPara;
HANDLE   hDevice;
HDevice = CreateDevice(0) ;      // 管理秕一个 PCI 设备
if(!LoadParaAD(hDevice, &ADPara))
{
      AfxMessageBox("读入硬件参数失败，请确认您的驱动程序是否正确安装");
      Return;    // 若错误，则退出该过程
}
        :
```

***Visual Basic 举例:***

```
        :
Dim ADPara As PCI8932_PARA_AD
Dim hDevice As Long   :
 hDevice = CreateDevice(0)    ' 管理第一个 PCI 设备
If Not LoadParaAD(hDevice, ADPara) Then
      MsgBox "读入硬件参数失败，请确认您的驱动程序是否正确安装"
      Exit Sub     ' 若错误，则退出该过程
End If
        :
```

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

***Viusal C++ & C++Builder:***

BOOL SaveParaAD(HANDLE hDevice, PPCI8932_PARA_AD pADPara)

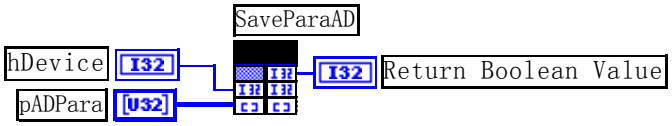
***Visual Basic:***

```
Declare Function SaveParaAD Lib "PCI8932" (ByVal hDevice As Long, _
                                          ByRef pADPara As PCI8932_PARA_AD) As Boolean
```

***Delphi:***

```
Function SaveParaAD (hDevice : Integer; pADPara:PPCI8932_PARA_AD):Boolean;
                    StdCall; External 'PCI8932' Name 'SaveParaAD';
```

***LabView:***




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。

pADPara AD设备硬件参数，请参考《硬件参数结构》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数:** CreateDevice LoadParaAD ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

***Viusal C++ & C++ Builder:***

```
BOOL ResetParaAD (HANDLE hDevice,
                  PPCI8932_PARA_AD pADPara)
```

***Visual Basic:***

```
Declare Function ResetParaAD Lib "PCI8932" (ByVal hDevice As Long, _
                                            ByRef pADPara As PCI8932_PARA_AD) As Boolean
```

***Delphi:***

```
Function ResetParaAD ( hDevice : Integer;
                       pADPara : PPCI8932_PARA_AD) : Boolean;
                       StdCall; External 'PCI8932' Name ' ResetParaAD ';
```

***LabVIEW:***

请参考相关演示程序。

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

参数：

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

pADPara设备硬件参数，它负责在参数被复位后返回其复位后的值。关于PCI8932_PARA_AD的详细介绍请参考PCI8932.h或PCI8932.Bas或PCI8932.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice LoadParaAD SaveParaAD
ResetParaAD ReleaseDevice

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

## 第五节、DA 模拟量输出操作函数原型说明

♦ **输出模拟信号到指定通道**

函数原型：

***Visual C++ & C++Builder:***

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

***Visual Basic:***

```
Declare Function WriteDeviceDA Lib "PCI8932" (ByVal hDevice As Long, _
                                              ByVal nDAData As Integer,_
                                              ByVal nDAChannel As Integer) As Boolean
```

***Delphi:***

```
Function WriteDeviceDA ( hDevice : Integer;
```

```
                                nDAData : SmallInt;
                                nDAChannel : Integer) : Boolean;
                                StdCall; External 'PCI8932' Name ' WriteDeviceDA ';
```

***LabVIEW：***

请参考相关演示程序。

功能：输出 DA 原始数据到 CN1 上的 VOUT0 上。

参数：

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

nDAData 指输出的DA原始码数据，它的取值范围为[0, 4096]，它与实际输出模拟量值的对应关系请参考《DA电压值转换成LSB原码数据的换算方法》章节。

nDAChannel 需要指定的模拟量通道号，其取值范围为[0, 3]。

返回值：若成功，返回TRUE，输出DA原始数据到CN1 上的VOUT0 上；否则返回FALSE，您可以调用GetLastErrorEx函数取得错误或错误字符信息。

**相关函数**: CreateDevice ReleaseDevice

- **以上函数调用一般顺序**

① CreateDevice

② WriteDeviceDA

③ ReleaseDevice

用户可以反复执行第②步，以进行模拟量输出不断变化（可以和 AD 采样同时进行，互不影响）。

## 第六节、CNT 计数与定时器操作函数原型说明

- **设置计数器的初值**

函数原型：

***Visual C++ & C++Builder:***

```
BOOL SetDeviceCNT(HANDLE hDevice,
                  ULONG ContrlMode,
                  ULONG CNTVal,
                  ULONG ulChannel)
```

***Visual Basic:***

```
Declare Function SetDeviceCNT Lib "PCI8932" (ByVal hDevice As Long, _
                                             ByVal ContrlMode As Long, _
                                               ByVal CNTVal As Long, _
                                               ByVal ulChannel As Long) As Boolean
```

***Delphi:***

```
Function SetDeviceCNT (hDevice : Integer;
                       ContrlMode : LongWord;
                       CNTVal : LongWord;
                       ulChannel : LongWord) : Boolean;
                       StdCall; External 'PCI8932' Name ' SetDeviceCNT ';
```

***LabVIEW:***

请参考相关演示程序。

功能：设置计数器的初值。

参数：

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

ContrlMode 方式控制字。其选项值如下表：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| PCI8932_GATEMODE_POSITIVE_0 | 0x0000 | 计数结束产生中断 |
| PCI8932_GATEMODE_RISING_1 | 0x0001 | 可编程单拍脉冲 |
| PCI8932_GATEMODE_POSITIVE_2 | 0x0002 | 频率发生器 |
| PCI8932_GATEMODE_POSITIVE_3 | 0x0003 | 方波发生器 |
| PCI8932_GATEMODE_POSITIVE_4 | 0x0004 | 软件触发选通 |
| PCI8932_GATEMODE_RISING_5 | 0x0005 | 硬件触发选通 |

CNTVal 计数初值。

ulChannel 计数器通道选择，取值为[0，2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数**: CreateDevice GetDeviceCNT ReleaseDevice

◆ **取得各路计数器的当前计数值**

函数原型：

***Visual C++ & C++Builder:***

```
BOOL GetDeviceCNT (HANDLE hDevice,
                   PULONG pCNTVal,
                   ULONG  ulChannel)
```

***Visual Basic:***

```
Declare Function GetDeviceCNT Lib "PCI8932" (ByVal hDevice As Long, _
                                             ByRef pCNTVal As Integer, _
                                             ByVal ulChannel As Long) As Boolean
```

***Delphi:***

```
Function GetDeviceCNT ( hDevice : Integer;
                        pCNTVal: Pointer;
                        pControlMode: Pointer;
                        ulChannel: LongWord) : Boolean;
                        StdCall; External 'PCI8932' Name ' GetDeviceCNT ';
```

***LabVIEW:***

请参考相关演示程序。

功能：取得各路计数器的当前计数值。

参数：

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

pCNTVal 取得计数初值。

ulChannel 计数器通道选择，取值为[0，2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数**: CreateDevice SetDeviceCNT ReleaseDevice

## 第七节、DIO 数字开关量输入输出简易操作函数原型说明

◆ **十六路开关量输入**

函数原型：

***Visual C++ & C++Builder:***

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

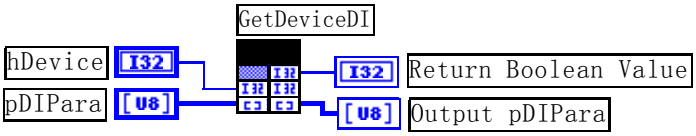
***Visual Basic:***

Declare Function GetDeviceDI Lib "PCI8932" (ByVal hDevice As Long, _
ByVal bDISts(0 to 15 ) As Long) As Boolean

***Delphi:***

Function GetDeviceDI ( hDevice : Integer; bDISts:Pointer):Boolean;
StdCall; External 'PCI8932' Name ' GetDeviceDI ';

***LabView：***




功能：负责将 PCI 设备上的输入开关量状态读入内存。

参数：

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

bDISts十六路开关量输入状态的参数结构，共有 16 个成员变量，分别对应于DI0-DI15 路开关量输入状态位。如果bDISts[0]为“1”则使 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。具体定义请参考《DI数字开关量输入参数介绍》章节。

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

♦ **十六路开关量输出**

函数原型：

***Visual C++ & C++Builder:***

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

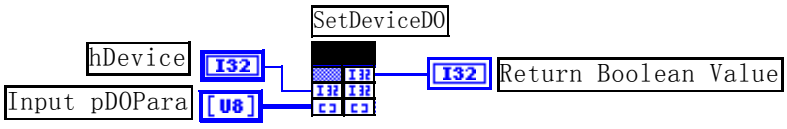
***Visual Basic:***

Declare Function SetDeviceDO Lib "PCI8932" ( ByVal hDevice As Long, _
ByVal bDOSts(0 to 15 )As Long) As Boolean

***Delphi:***

Function SetDeviceDO (hDevice : Integer; bDOSts:Pointer):Boolean;
StdCall; External 'PCI8932' Name ' SetDeviceDO ';

***LabView：***




功能：负责将 PCI 设备上的输出开关量置成相应的状态。

参数：

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

bDOSts 十六路开关量输出状态的参数结构，共有 16 个成员变量，分别对应于DO0-DO15 路开关量输出状态位。比如置bDOSts[0]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，必须对这个参数结构的DO0 至DO15 共 16 个成员变量赋初值，其值必须为“1”或“0”。具体定义请参考《DO数字开关量输出参数介绍》。

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++ & C++Builder:***

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

***Visual Basic:***

```
Declare Function RetDeviceDOLib "PCI8932" (ByVal hDevice As Long, _
                                           ByVal bDOSts(0 to 15) As Byte) As Boolean
```

***Delphi:***

```
Function RetDeviceDO (hDevice : Integer;
                      bDOSts : Pointer) : Boolean;
                      StdCall; External 'PCI8932' Name ' RetDeviceDO';
```

***LabVIEW:***

请参考相关演示程序。

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

参数：

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

bDOSts 获得开关输出状态(注意：必须定义为 16 个字节元素的数组)。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice GetDeviceDI ReleaseDevice

❖ **以上函数调用一般顺序**

① CreateDevice

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

③ ReleaseDevice

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

# 第四章 硬件参数结构

**AD 硬件参数介绍（PCI8932_PARA_AD）**

***Visual C++ & C++Builder:***

```
typedef struct _PCI8932_PARA_AD        // 板卡各参数值
{
    LONG FirstChannel;          // 首通道[0, 15]
    LONG LastChannel;           // 末通道[0, 15],要求末通道必须大于或等于首通道
    LONG InputRange;            // 模拟量输入量程范围
    LONG GroundingMode;         // 接地方式(单端或双端选择)
    LONG Gains;                 // 程控增益
} PCI8932_PARA_AD, *PPCI8932_PARA_AD;
```

***Visual Basic：***

```
Private Type PCI8932_PARA_AD
```

```
    FirstChannel As Long          ' 首通道[0, 15]
    LastChannel As Long           ' 末通道[0, 15]，要求末通道必须大于或等于首通道
    InputRange As Long            ' 模拟量输入量程范围
    GroundingMode As Long         ' 接地方式(单端或双端选择)
    Gains As Long                 ' 程控增益
End Type
```

***Delphi:***

```
Type  // 定义结构体数据类型
PPCI8932_PARA_AD = ^PCI8932_PARA_AD;  // 指针类型结构
PCI8932_PARA_AD = record          // 标记为记录型
    FirstChannel : LontInt;       // 首通道[0, 15]
    LastChannel : LontInt;        // 末通道[0, 15]，要求末通道必须大于或等于首通道
    InputRange : LontInt;         // 模拟量输入量程范围
    GroundingMode : LontInt;      // 接地方式(单端或双端选择)
    Gains As Long;                // 程控增益
End;
```

***LabView:***

请参考相关演示程序。

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

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

FirstChannel　AD采样首通道，其取值范围为[0, 15]，它应等于或小于LastChannel参数。
LastChannel　AD采样末通道，其取值范围为[0, 15]，它应等于或大于FirstChannel参数。

InputRange 被测模拟信号输入范围，取值如下表：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| PCI8932_INPUT_N10000_P10000mV | 0x00 | ±10000mV |
| PCI8932_INPUT_N5000_P5000mV | 0x01 | ±5000mV |
| PCI8932_INPUT_N2500_P2500mV | 0x02 | ±2500mV |
| PCI8932_INPUT_0_P10000mV | 0x03 | 0～10000mV |

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

GroundingMode AD 接地方式选择。它的选项值如下表：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| PCI8932_ GNDMODE_SE | 0x00 | 单端方式(SE:Single end) |
| PCI8932_ GNDMODE_DI | 0x01 | 双端方式(DI:Differential) |

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

| 常量名 | 常量值 | 功能定义 |
|---|---|---|

| | | |
|---|---|---|
| PCI8932_GAINS_1MULT | 0x00 | 1 倍增益 |
| PCI8932_GAINS_2MULT | 0x01 | 2 倍增益 |
| PCI8932_GAINS_4MULT | 0x02 | 4 倍增益 |
| PCI8932_GAINS_8MULT | 0x03 | 8 倍增益 |

**相关函数：** CreateDevice LoadParaAD SaveParaAD ReleaseDevice

# 第五章 数据格式转换与排列规则

## 第一节、AD 原始数据 LSB 转换成电压值 Volt 的换算方法

在换算过程中弄清模板精度（即 Bit 位数）是很关键的，因为它决定了 LSB 数码的总宽度 CountLSB。比如 12 位的模板 CountLSB 为 4096。其他类型同理均按 $2^n$=LSB 总数（n 为 Bit 位数）换算即可。

| 量程(毫伏) | 计算机语言换算公式(标准 C 语法) | Volt 取值范围 mV |
|---|---|---|
| ±10000 | Volt = (20000.00/4096) * ((ADBuffer[0] ^ 0x0800) &0x0FFF) – 10000.00 | [-10000.00，+ 9995.11] |
| ±5000 | Volt = (10000.00/4096) * ((ADBuffer[0] ^ 0x0800) &0x0FFF) – 5000.00 | [-5000.00， + 4997.55] |
| ±2500 | Volt = (5000.00/4096) * ((ADBuffer[0] ^ 0x0800) &0x0FFF) – 2500.00 | [-2500.00， + 2498.77] |
| 0～10000 | Volt = (10000.00/4096) * ((ADBuffer[0] ^ 0x0800) &0x0FFF) | [0.00， + 9997.55] |

换算举例：（设量程为±10000mV，这里只转换第一个点）

***Visual C++&C++Builder：***

```
USHORT Lsb; // 定义存放标准 LSB 原码的变量
float Volt;    // 定义存放转换后的电压值的变量
float PerLsbVolt = 20000.00/4096; // 求出每个 LSB 原码单位电压值
Lsb = (ADBuffer[0] ^ 0x0800) &0x0FFF;
Volt = PerLsbVolt * Lsb – 10000.00;  // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

***Visual Basic:***

```
Dim Lsb As Long   ' 定义存放标准 LSB 原码的变量
Dim Volt As Single   ' 定义存放转换后的电压值的变量
Dim PerLsbVolt As Single
PerLsbVolt = 20000.00/4096   ' 求出每个 LSB 原码单位电压值
Lsb = (ADBuffer(0) XOR 2048) AND 4096    ' 将其转换成无符号 16 位有效数据
Volt = PerLsbVolt * Lsb-10000.00 ' 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

***Delphi:***

```
Lsb : Word;   // 定义存放标准 LSB 原码的变量
Volt ： Single;   // 定义存放转换后的电压值的变量
PerLsbVolt : Single;
PerLsbVolt := 20000.00/4096;   // 求出每个 LSB 原码单位电压值
```

Lsb := (ADBuffer[0] ^ 0x0800) &0x0FFF;

Volt ：= PerLsbVolt * Lsb-10000.00; // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值

## 第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

当首末通道相等时，即为单通道采集，假如FirstChannel=5, LastChannel=5,其排放规则如下：

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

两通道采集(CH0 – CH1)

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

四通道采集(CH0 - CH3)

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

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取AD数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐问题，尤其任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长（在PCI设备上同时也应 32 的整数倍），这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个AD通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n为每个通道的点数)的长度采集。为了更加详细地说明问题民，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如ReadDeviceAD函数读回，即便不考虑是否能一次读完的问题，但对用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

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

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


## 第三节、DA 电压值转换成 LSB 原码数据的换算方法

| 量程(伏) | 计算机语言换算公式(标准 C 语法) | Lsb 取值范围 |
|---|---|---|
| 0～5000mV | Lsb = Volt / (5000.00 / 4096 ) | [0，4095] |
| 0～10000mV | Lsb = Vol t/ (10000.00 / 4096 ) | [0，4095] |
| ±5000mV | Lsb = Volt / (10000.00 / 4096) + 2048 | [0，4095] |
| ±10000mV | Lsb = Volt / (20000.00 / 4096 ) + 2048 | [0，4095] |

请注意这里求得的LSB数据就是用于WriteDeviceDA中的nDAData参数的。

# 第六章　上层用户函数接口应用实例

如果您想快速的了解驱动程序的使用方法和调用流程，以最短的时间建立自己的应用程序，那么我们强烈建议您参考相应的简易程序。此种程序属于工程级代码，可以直接打开不用作任何配置和代码修改即可编译通过，运行编译链接后的可执行程序，即可看到预期效果。

如果您想了解硬件的整体性能、精度、采样连续性等指标以及波形显示、数据存盘与分析、历史数据回放等功能，那么请参考高级演示程序。特别是许多不愿意编写任何程序代码的用户，您可以使用高级程序进行采集、显示、存盘等功能来满足您的要求。甚至可以用我们提供的专用转换程序将高级程序采集的存盘文件转换成相应格式，即可在 Excel、MatLab 第三方软件中分析数据（此类用户请最好选用通过 Visual C++制作的高级演示系统）。

## 第一节、简易程序演示说明

**一、怎样使用ReadDeviceAD函数进行AD连续数据采集**

其详细应用实例及工程级代码请参考 Visual C++简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PCI8932.h 和 Sys.cpp)。

**[程序]**\|**[阿尔泰测控演示系统]**\|**[Microsoft Visual C++]**\|[简易代码演示]\|[AD 采集演示源程序]

**其简易程序默认存放路径为：系统盘\ART\PCI8932\SAMPLES\VC\SIMPLE\AD**

**二、怎样使用SetDeviceDO和GetDeviceDI函数进行开关量输入输出操作**

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PCI8932.h 和 Sys.cpp)。

**[程序]**\|**[阿尔泰测控演示系统]**\|**[Microsoft Visual C++]**\|[简易代码演示]\|[开关量演示源程序]

**其默认存放路径为：系统盘\ART\PCI8932\SAMPLES\VC\SIMPLE\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 PCI8932.h 和 DADoc.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\PCI8932\SAMPLES\VC\ADVANCED**

**其他语言的演示可以用上面类似的方法找到。**

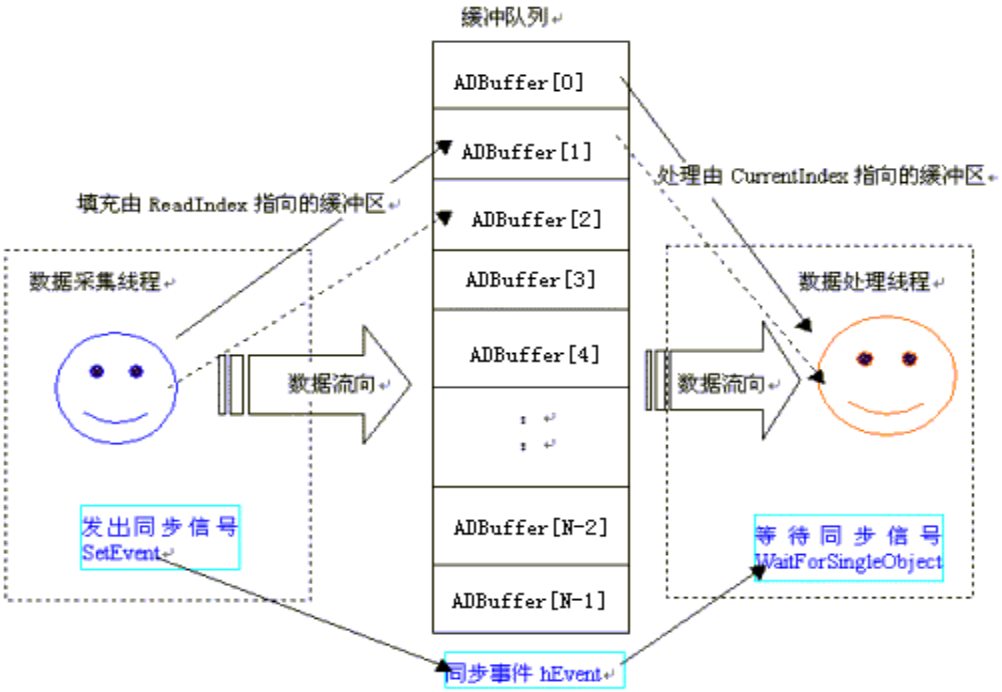
# 第七章　基于 PCI 总线的大容量连续数据采集详述

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

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中，由于任务之间的调度切换非常平凡，特别是当用户移动窗口、或弹出对话框等，则会使当前线程猛地花掉大量的时间去处理这些图形操作，因此如果处理不当，则将无法实现高速连续不间断采集，那么如何更好的克服这些问题呢？用子线程则是必须的（在这里我们称之为数据采集线程），但这还不够，必须要求这个线程是绝对的工作者线程，即这个线程在正常采集中不能有任何窗口等图形操作。只有这样，当用户进行任何窗口操作时，这个线程才不会被堵塞，因此可以保证其正常连续的数据采集。但是用户可能要问，不能进行任何窗口操作，那么我如何将采集的数据显示在屏幕上呢？其实很简单，再开辟一个子线程，我们称之数据处理线程，也叫用户界面线程。最初，数据处理线程不做任何工作，而是在 Win32 API 函数 WaitForSingleObject 的作用下进入睡眠状态，此时它基本不消耗 CPU 时间，即可保证其他线程代码有充分的运行机会（这里当然主要指数据采集线程），当数据采集线程取得指定长度的数据到用户空间时，则再用 Win32 API 函数 SetEvent 将指定事件消息发送给数据处理线程，则数据处理线程即刻恢复运行状态，迅速对这批数据进行处理，如计算、在窗口绘制波形、存盘等操作。

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

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




**下面用 Visual C++程序举例说明。**

**使用ReadDeviceAD函数读取设备上的AD数据**

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

**[程序]**丨**[阿尔泰测控演示系统]**丨**[PCI8932 32 路 AD 和 16 路开关量卡]**丨**[Microsoft Visual C++]**丨**[高级演示程序]**

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

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

# 第八章　公共接口函数介绍

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“PCI8932_”）

| 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 | |
| GetDevVersion | 获取设备固件及程序版本 | |
| WriteRegisterByte | 以字节(8Bit)方式写寄存器端口 | 底层用户 |
| WriteRegisterWord | 以字(16Bit)方式写寄存器端口 | 底层用户 |
| WriteRegisterULong | 以双字(32Bit)方式写寄存器端口 | 底层用户 |
| ReadRegisterByte | 以字节(8Bit)方式读寄存器端口 | 底层用户 |
| ReadRegisterWord | 以字(16Bit)方式读寄存器端口 | 底层用户 |
| ReadRegisterULong | 以双字(32Bit)方式读寄存器端口 | 底层用户 |
| **② ISA 总线 I/O 端口操作函数** | | |
| WritePortByte | 以字节(8Bit)方式写 I/O 端口 | 用户程序操作端口 |
| WritePortWord | 以字(16Bit)方式写 I/O 端口 | 用户程序操作端口 |
| WritePortULong | 以无符号双字(32Bit)方式写 I/O 端口 | 用户程序操作端口 |
| ReadPortByte | 以字节(8Bit)方式读 I/O 端口 | 用户程序操作端口 |
| ReadPortWord | 以字(16Bit)方式读 I/O 端口 | 用户程序操作端口 |
| ReadPortULong | 以无符号双字(32Bit)方式读 I/O 端口 | 用户程序操作端口 |
| **③ 线程操作函数** | | |
| CreateSystemEvent | 创建系统内核事件对象 | 用于线程同步或中断 |
| ReleaseSystemEvent | 释放系统内核事件对象 | |
| DelayTimeUs | 高效高精度延时函数 | 不消耗 CPU 时间 |
| **④ 文件对象操作函数** | | |
| CreateFileObject | 初始设备文件对象 | |
| WriteFile | 请求文件对象写用户数据到磁盘文件 | |
| ReadFile | 请求文件对象读数据到用户空间 | |
| SetFileOffset | 设置文件指针偏移 | |
| GetFileLength | 取得文件长度 | |
| ReleaseFile | 释放已有的文件对象 | |
| GetDiskFreeBytes | 取得指定磁盘的可用空间(字节) | 适用于所有设备 |
| **⑤ 各种参数保存和读取函数** | | |
| SaveParaInt | 保存整型参数到注册表 | |
| LoadParaInt | 从注册表中读取整型参数值 | |
| SaveParaString | 保存字符参数到注册表 | |
| LoadParaString | 从注册表中读取字符参数值 | |
| **⑥ 其他函数** | | |
| kbhit | 探测用户是否有击键动作 | |
| getch | 等待并获取用户击键值 | |
| 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++ & C++ Builder:***

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

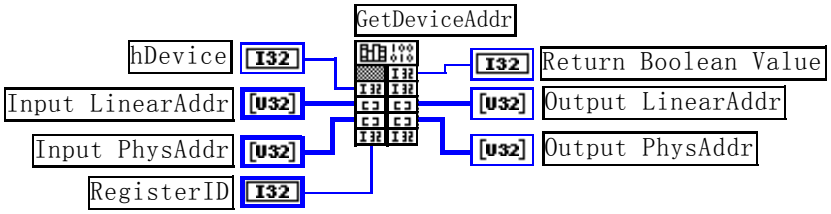
***Visual Basic:***

```
Declare Function GetDeviceAddr Lib "PCI8932" (ByVal hDevice As Long, _
                                              ByRef LinearAddr As Long, _
                                              ByRef PhysAddr As Long, _
                                              Optional ByVal RegisterID As Integer = 0) As Boolean
```

***Delphi:***

```
Function GetDeviceAddr(hDevice : Integer;
                    LinearAddr : Pointer;
                    PhysAddr : Pointer;
                    RegisterID : Integer = 0) : Boolean;
                    StdCall; External 'PCI8932' Name ' GetDeviceAddr ';
```

***LabVIEW:***




功能：取得 PCI 设备指定的内存映射寄存器的线性地址。

参数：

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

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

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

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

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| PCI8932_REG_MEM_PLXCHIP | 0x0000 | 0 号寄存器对应 PLX 芯片所使用的内存模式基地址(使用 LinearAddr) |
| PCI8932_REG_IO_CPLD | 0x0001 | 1 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr) |

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

***Visual C++ & C++ Builder 程序举例:***

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

***Visual Basic 程序举例:***

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

♦ **取得指定的指定设备寄存器组 BAR 地址**

函数原型:

***Visual C++ & C++ Builder:***

```
BOOL GetDeviceBar ( HANDLE hDevice,
                    ULONG    pulPCIBar[6])
```

***Visual Basic:***

```
Declare Function GetDeviceBar Lib "PCI8932" (ByVal hDevice As Long, _
                                             ByVal pulPCIBar (0 to 5) As Long) As Boolean
```

***Delphi:***

```
Function GetDeviceBar (hDevice : Integer;
                       pulPCIBar : Pointer) : Boolean;
                       StdCall; External 'PCI8932' Name ' GetDeviceBar';
```

***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或CreateDeviceEx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, GetDeviceAddr, WriteRegisterByte, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

♦**获取设备固件及程序版本**

函数原型:

***Visual C++ & C++ Builder:***

```
BOOL GetDevVersion ( HANDLE hDevice,
                     PULONG pulFmwVersion,
                     PULONG pulDriverVersion)
```

***Visual Basic:***

```
Declare Function GetDevVersion Lib "PCI8932" (ByVal hDevice As Long, _
                                              ByRef pulFmwVersion As Long, _
                                              ByRef pulDriverVersion As Long) As Boolean
```

***Delphi:***

```
Function GetDevVersion (hDevice : Integer;
                        pulFmwVersion: Pointer;
                        pulDriverVersion: Pointer) : Boolean;
                        StdCall; External 'PCI8932' Name ' GetDevVersion ';
```

***LabVIEW:***

请参见相关演示程序。

功能：获取设备固件及程序版本。

参数：

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

pulFmwVersion 指针参数，用于取得固件版本。

pulDriverVersion 指针参数，用于取得驱动版本。

返回值：如果执行成功，则返回 TRUE，否则会返回 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++ & C++ Builder:***

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

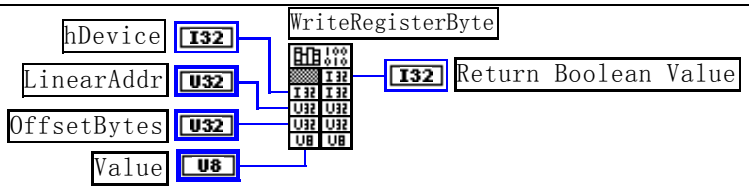
***Visual Basic:***

```
Declare Function WriteRegisterByte Lib "PCI8932" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte ) As Boolean
```

***Delphi:***

```
Function WriteRegisterByte(  hDevice : Integer;
                             LinearAddr : LongWord;
                             OffsetBytes : LongWord;
                             Value : Byte) : Boolean;
                             StdCall; External 'PCI8932' Name ' WriteRegisterByte ';
```

***LabVIEW:***




功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:***

```
BOOL WriteRegisterWord(HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes,
```

WORD Value)

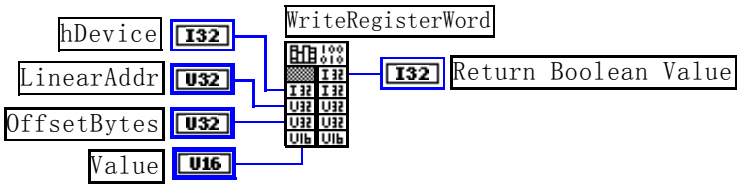
***Visual Basic:***

```
Declare Function WriteRegisterWord Lib "PCI8932" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

***Delphi:***

```
Function WriteRegisterWord( hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : Word) : Boolean;
                            StdCall; External 'PCI8932' Name ' WriteRegisterWord ';
```

***LabVIEW:***




功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:***

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

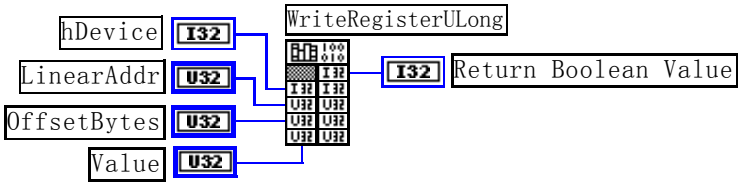
***Visual Basic:***

```
Declare Function WriteRegisterULong Lib "PCI8932" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long)As Boolean
```

***Delphi:***

```
Function WriteRegisterULong(hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : LongWord) : Boolean;
                            StdCall; External 'PCI8932' Name ' WriteRegisterULong ';
```

***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或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:***

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

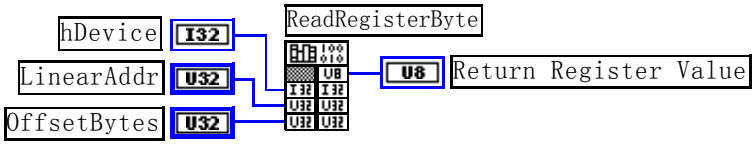
***Visual Basic:***

```
Declare Function ReadRegisterByte Lib "PCI8932" (ByVal hDevice As Long, _
                                                 ByVal LinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

***Delphi:***

```
Function ReadRegisterByte(hDevice : Integer;
                          LinearAddr : LongWord;
                          OffsetBytes : LongWord) : Byte;
                          StdCall; External 'PCI8932' Name ' ReadRegisterByte ';
```

***LabVIEW:***




功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:***

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

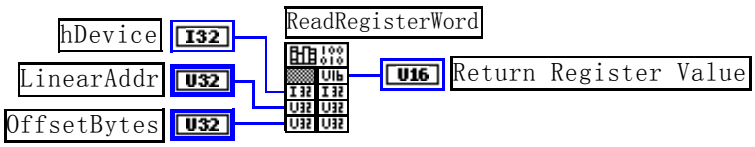
***Visual Basic:***

```
Declare Function ReadRegisterWord Lib "PCI8932" (  ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long) As Integer
```

***Delphi:***

```
Function ReadRegisteWord(hDevice : Integer;
                         LinearAddr : LongWord;
                         OffsetBytes : LongWord) : Word;
                         StdCall; External 'PCI8932' Name ' ReadRegisterWord ';
```

***LabVIEW:***




功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:***

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

***Visual Basic:***

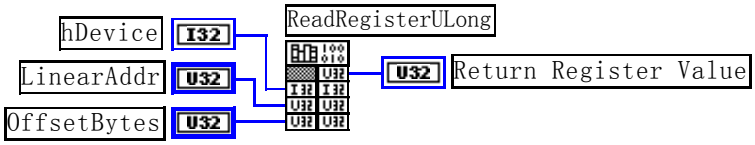
```
Declare Function ReadRegisterULong Lib "PCI8932" (ByVal hDevice As Long, _
```

```
                                        ByVal LinearAddr As Long, _
                                        ByVal OffsetBytes As Long) As Long
```

***Delphi:***

```
Function ReadRegisterULong(hDevice : Integer;
                           LinearAddr : LongWord;
                           OffsetBytes : LongWord) : LongWord;
                           StdCall; External 'PCI8932' Name ' ReadRegisterULong ';
```

***LabVIEW:***




功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder 程序举例:***

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

***Visual Basic 程序举例:***

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

## 第三节、I0 端口读写函数原型说明

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

♦ **以单字节(8Bit)方式写 I/O 端口**

***Visual C++ & C++ Builder:***

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

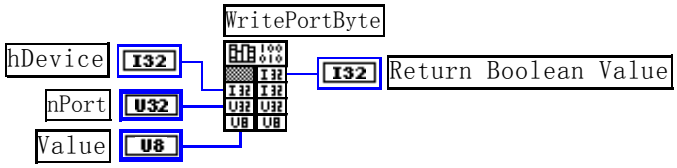
***Visual Basic:***

```
Declare Function WritePortByte Lib "PCI8932" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Byte) As Boolean
```

***Delphi:***

```
Function WritePortByte(hDevice : Integer;
                       nPort : LongWord;
                       Value : Byte) : Boolean;
                       StdCall; External 'PCI8932' Name ' WritePortByte ';
```

***LabVIEW:***




功能：以单字节(8Bit)方式写 I/O 端口。

参数：

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

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用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++ & C++ Builder:***

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

***Visual Basic:***

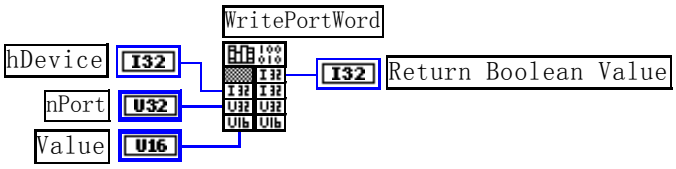
```
Declare Function WritePortWord Lib "PCI8932" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

***Delphi:***

```
Function WritePortWord(hDevice : Integer;
                       nPort : LongWord;
                       Value : Word) : Boolean;
```

StdCall; External 'PCI8932' Name ' WritePortWord ';

***LabVIEW:***




功能：以双字(16Bit)方式写 I/O 端口。

参数：

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

nPort　设备的 I/O 端口号。

Value　写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用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++ & C++ Builder:***

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

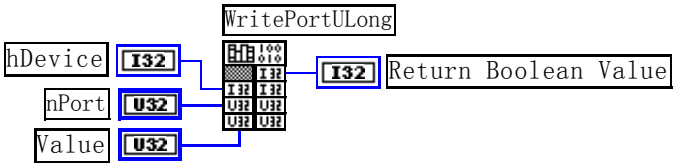
***Visual Basic:***

```
Declare Function WritePortULong Lib "PCI8932" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

***Delphi:***

```
Function WritePortULong(hDevice : Integer;
                        nPort : LongWord;
                        Value : LongWord) : Boolean;
                        StdCall; External 'PCI8932' Name ' WritePortULong ';
```

***LabVIEW:***




功能：以四字节(32Bit)方式写 I/O 端口。

参数：

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

nPort　设备的 I/O 端口号。

Value　写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用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++ & C++ Builder:***

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

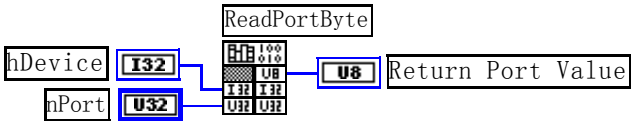
***Visual Basic:***

```
Declare Function ReadPortByte Lib "PCI8932" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

***Delphi:***

```
Function ReadPortByte(hDevice : Integer;
                      nPort : LongWord) : Byte;
                      StdCall; External 'PCI8932' Name ' ReadPortByte ';
```

***LabVIEW:***




功能：以单字节(8Bit)方式读 I/O 端口。

参数：

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

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

**相关函数：** CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

***Visual C++ & C++ Builder:***

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

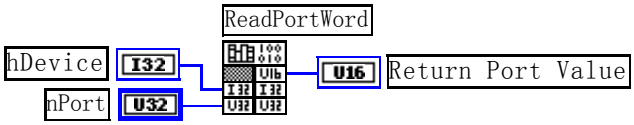
***Visual Basic:***

```
Declare Function ReadPortWord Lib "PCI8932" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

***Delphi:***

```
Function ReadPortWord(hDevice : Integer;
                      nPort : LongWord) : Word;
                      StdCall; External 'PCI8932' Name ' ReadPortWord ';
```

***LabVIEW:***




功能：以双字节(16Bit)方式读 I/O 端口。

参数：

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

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

**相关函数：** CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式读 I/O 端口**

***Visual C++ & C++ Builder:***

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

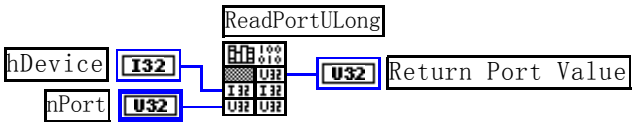
***Visual Basic:***

```
Declare Function ReadPortULong Lib "PCI8932" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long ) As Long
```

***Delphi:***

```
Function ReadPortULong(hDevice : Integer;
                       nPort : LongWord) : LongWord;
                       StdCall; External 'PCI8932' Name ' ReadPortULong ';
```

***LabVIEW:***




功能：以四字节(32Bit)方式读 I/O 端口。

参数：

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

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

**相关函数：** CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

## 第四节、线程操作函数原型说明

♦ **创建内核系统事件**

***Visual C++ & C++ Builder:***

```
HANDLE CreateSystemEvent(void)
```

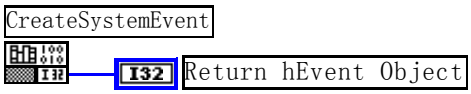
***Visual Basic:***

```
Declare Function CreateSystemEvent Lib " PCI8932 " ( ) As Long
```

***Delphi:***

```
Function CreateSystemEvent( ) : Integer;
                    StdCall; External 'PCI8932' Name ' CreateSystemEvent ';
```

***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**

***Visual C++ & C++ Builder:***

```
BOOL ReleaseSystemEvent(HANDLE hEvent)
```

***Visual Basic:***

```
Declare Function ReleaseSystemEvent Lib " PCI8932 " (ByVal hEvent As Long) As Boolean
```

***Delphi:***

```
Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
                                StdCall; External 'PCI8932' Name ' ReleaseSystemEvent ';
```

***LabVIEW:***

请参见相关演示程序。

功能：释放系统内核事件对象。

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

返回值：若成功，则返回 TRUE。

- **高效高精度延时**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function DelayTimeUs Lib "PCI8932" (ByVal hDevice As Long, _
                                            ByVal nTimeUs As Long) As Boolean
```

***Delphi:***

```
Function DelayTimeUs (hDevice: Integer;
                      nTimeUs : LongInt) : Boolean;
                      StdCall; External 'PCI8932' Name ' DelayTimeUs ';
```

***LabVIEW:***

请参考相关演示程序。

功能：微秒级延时函数。

参数：

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

nTimeUs 时间常数。单位 1 微秒。

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

## 第五节、文件对象操作函数原型说明

- **创建文件对象**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function CreateFileObject Lib "PCI8932" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

***Delphi:***

```
Function CreateFileObject ( hDevice : Integer;
                            strFileName : string;
                            Mode : Integer) : Integer;
                          Stdcall; external 'PCI8932' Name ' CreateFileObject ';
```

***LabVIEW:***

请参见相关演示程序。

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

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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:\\PCI8932\\Data.Dat”，在 Basic 中，其语法格式如：“C:\PCI8932\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 |
|---|---|---|
| PCI8932_modeRead | 0x0000 | 只读文件方式 |
| PCI8932_modeWrite | 0x0001 | 只写文件方式 |
| PCI8932_modeReadWrite | 0x0002 | 既读又写文件方式 |
| PCI8932_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 |
| PCI8932_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++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function WriteFile Lib "PCI8932" (ByVal hFileObject As Long,_
                                          ByRef pDataBuffer As Byte,_
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

***Delphi:***

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : Integer) : Boolean;
                   Stdcall; external 'PCI8932' Name ' WriteFile ';
```

***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 GetLastError 捕获错误码。

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

♦ **通过设备对象,从指定磁盘文件中读采样数据**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function ReadFile Lib "PCI8932" (  ByVal hFileObject As Long, _
                                           ByRef pDataBuffer As Integer, _
                                           ByVal nOffsetBytes As Long, _
                                           ByVal nReadSizeBytes As Long) As Boolean
```

***Delphi:***

```
Function ReadFile( hFileObject : Integer;
                   pDataBuffer : Pointer;
                   nOffsetBytes : Integer;
                   nReadSizeBytes : Integer) : Boolean;
                   Stdcall; external 'PCI8932' Name ' ReadFile ';
```

***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。

nOffsetBytes 指定从文件开始端所偏移的读位置。

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

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

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

♦ **设置文件偏移位置**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function SetFileOffset Lib "PCI8932" (ByVal hFileObject As Long,
                                              ByVal nOffsetBytes As Long) As Boolean
```

***Delphi:***

```
Function SetFileOffset ( hFileObject : Integer;
                         nOffsetBytes : LongInt) : Boolean;
```

Stdcall; external 'PCI8932' Name ' SetFileOffset ';

***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。

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

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：

***Visual C++ & C++ Builder:***

ULONG GetFileLength (HANDLE hFileObject)

***Visual Basic:***

Declare Function GetFileLength Lib "PCI8932" (ByVal hFileObject As Long) As Long

***Delphi:***

Function GetFileLength (hFileObject : Integer) : LongWord;
Stdcall; external 'PCI8932' Name ' GetFileLength ';

***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 GetLastError 捕获错误码。

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

- **释放设备文件对象**

函数原型：

***Visual C++ & C++ Builder:***

BOOL ReleaseFile(HANDLE hFileObject)

***Visual Basic:***

Declare Function ReleaseFile Lib "PCI8932" (ByVal hFileObject As Long) As Boolean

***Delphi:***

Function ReleaseFile(hFileObject : Integer) : Boolean;
Stdcall; external 'PCI8932' Name ' ReleaseFile ';

***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 GetLastError 捕获错误码。

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

♦ **取得指定磁盘的可用空间**

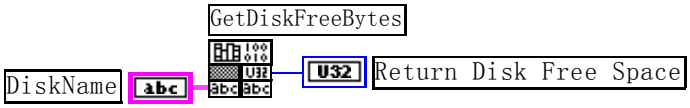
***Visual C++ & C++ Builder:***

LONGLONG GetDiskFreeBytes(LPCTSTR strDiskName )

***Visual Basic:***

Declare Function GetDiskFreeBytes Lib "PCI8932" (ByVal strDiskName As String ) As Currency

***LabVIEW:***




功能：取得指定磁盘的可用剩余空间(以字为单位)。

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

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

## 第六节、各种参数保存和读取函数原型说明

♦ **将整型变量的参数值保存在系统注册表中**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function SaveParaInt Lib "PCI8932" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nValue As Integer) As Boolean
```

***Delphi:***

```
Function SaveParaInt( hDevice : Integer;
                      strParaName : String;
                      nValue : Integer) : Boolean;
                     Stdcall; external 'PCI8932' Name ' SaveParaInt ';
```

***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\PCI8932\Device-0\Others。

参数：

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

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

**相关函数：** SaveParaInt LoadParaInt SaveParaString LoadParaString

♦ **将整型变量的参数值从系统注册表中读出**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function LoadParaInt Lib "PCI8932" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nDefaultVal As Integer) As Long
```

***Delphi:***

```
Function LoadParaInt ( hDevice : Integer;
                       strParaName : String;
                       nDefaultVal: Integer) : Longword;
                      Stdcall; external 'PCI8932' Name ' LoadParaInt ';
```

***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\PCI8932\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx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++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function SaveParaString Lib "PCI8932" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String) As Boolean
```

***Delphi:***

```
Function SaveParaString (hDevice : Integer;
                         strParaName : String;
                         strParaVal: String) : Boolean;
                         Stdcall; external 'PCI8932' Name ' SaveParaString';
```

***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\PCI8932\Device-0\Others。

参数：

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

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

**相关函数：** SaveParaInt LoadParaInt SaveParaString

LoadParaString

♦将字符变量的参数值从系统注册表中读出

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function LoadParaString Lib "PCI8932" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String,_
                                               ByVal strDefaultVal As String) As Boolean
```

***Delphi:***

```
Function LoadParaString ( hDevice : Integer;
                          strParaName : String;
                          strParaVal : String;
                          strDefaultVal : String) : Boolean;
                          Stdcall; external 'PCI8932' Name ' LoadParaString ';
```

***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\PCI8932\Device-0\Others。

参数：

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

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

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

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

**相关函数：** SaveParaInt LoadParaInt SaveParaString LoadParaString

## 第七节、其他函数原型说明

♦ 探测用户是否有按键动作

函数原型：

***Visual C++ & C++ Builder:***

```
BOOL kbhit (void)
```

***Visual Basic:***

```
Declare Function kbhit Lib "PCI8932" () As Boolean
```

***Delphi:***

```
Function kbhit () : Boolean;
                    Stdcall; external 'PCI8932' Name ' kbhit ';
```

***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++ & C++ Builder:***

```
char getch (void)
```

***Visual Basic:***

```
Declare Function getch Lib "PCI8932" () As String
```

***Delphi:***

```
Function getch () : char;
                Stdcall; external 'PCI8932' Name 'getch';
```

***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(ACII 码)。

**相关函数：** getch kbhit

◆**怎样获取驱动函数错误信息**

函数原型：

***Visual C++ & C++ Builder:***

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

***Visual Basic:***

```
Declare Function GetLastErrorEx Lib "PCI8932" (ByVal strFuncName As String,_
                                                ByVal strErrorMsg As String) As Long
```

***Delphi:***

```
Function GetLastErrorEx (strFuncName: String;
                         strErrorMsg: String) : LongWord;
                         Stdcall; external 'PCI8932' Name ' GetLastErrorEx ';
```

***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 PCI8932_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8932_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串。

返回值：返回错误码。

**相关函数：** 无。

***Visual C++ & C++Builder 程序举例***

```
        :
char strErrorMsg[256];     // 用于返回错误信息字串，要求其空间足够大
DWORD dwErrorCode;
int DeviceLgcID = 0;
hDevice = PCI8932_CreateDevice ( DeviceLgcID ); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
 {
      dwErrorCode = PCI8932_GetLastErrorEx("PCI8932_CreateDevice", strErrorMsg);
      AfxMessageBox(strErrorMsg); // 以对话框方式显示错误信息
      return;     // 退出该函数
 }
     :
```

***Visual Basic 程序举例***

```
      :
Dim strErrormsg As String   ' 用于返回错误信息字串，要求其空间足够大
Dim dwErrorCode As Long
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI8932_CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
      dwErrorCode = PCI8932_GetLastErrorEx("PCI8932_CreateDevice", strErrorMsg)
      MsgBox strErrorMsg   ' 以对话框方式显示错误信息
      Exit Sub      ' 退出该过程
End If
      :
```

◆ **移除驱动函数错误信息**

函数原型：

***Visual C++ & C++ Builder:***

BOOL RemoveLastErrorEx (LPCTSTR strFuncName)

***Visual Basic:***

Declare Function RemoveLastErrorEx Lib "PCI8932" (ByVal strFuncName As String) As Boolean

***Delphi:***

Function RemoveLastErrorEx (strFuncName: String) : Boolean;
                Stdcall; external 'PCI8932' Name ' RemoveLastErrorEx';

***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 PCI8932_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8932_InitDeviceAD”，否则得不到相应信息。

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**：无。