PCI8252 数据采集卡

WIN2000/XP 驱动程序使用说明书

北京阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 xxxx_则被省略。如 PCI8252_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InitDeviceProDA、WriteDeviceProDA等。而底层用户函数如WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上可以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。因为上层函数的命名、参数的命名极其规范。

第二节、如何管理设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用CreateDevice函数创建一个设备对象句柄hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给其他函数，如InitDeviceProDA可以使用hDevice句柄以程序查询方式初始化设备的DA部件，WriteDeviceProDA函数可以用hDevice句柄实现对DA数据的采样读取。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InitDeviceProDA函数初始化DA部件，关于频率等参数的设置是由这个函数的pDAPara参数结构体决定的。您只需要对这个pDAPara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。然后调用WriteDeviceProDA将准备好的DA数据写入板载RAM中，接着用StartDeviceProDA即可启动DA部件，开始DA输出，GetDeviceStatusProDA函数以查询DA的状态，用户可以根据其状态作出相应的处理。当您需要暂停设备时，执行StopDeviceProDA，当您需要关闭DA设备时，ReleaseDeviceProDA 便可帮您实现（但设备对象hDevice依然存在）具体执行流程请看下面的图 2. 1。

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。（图中的线）

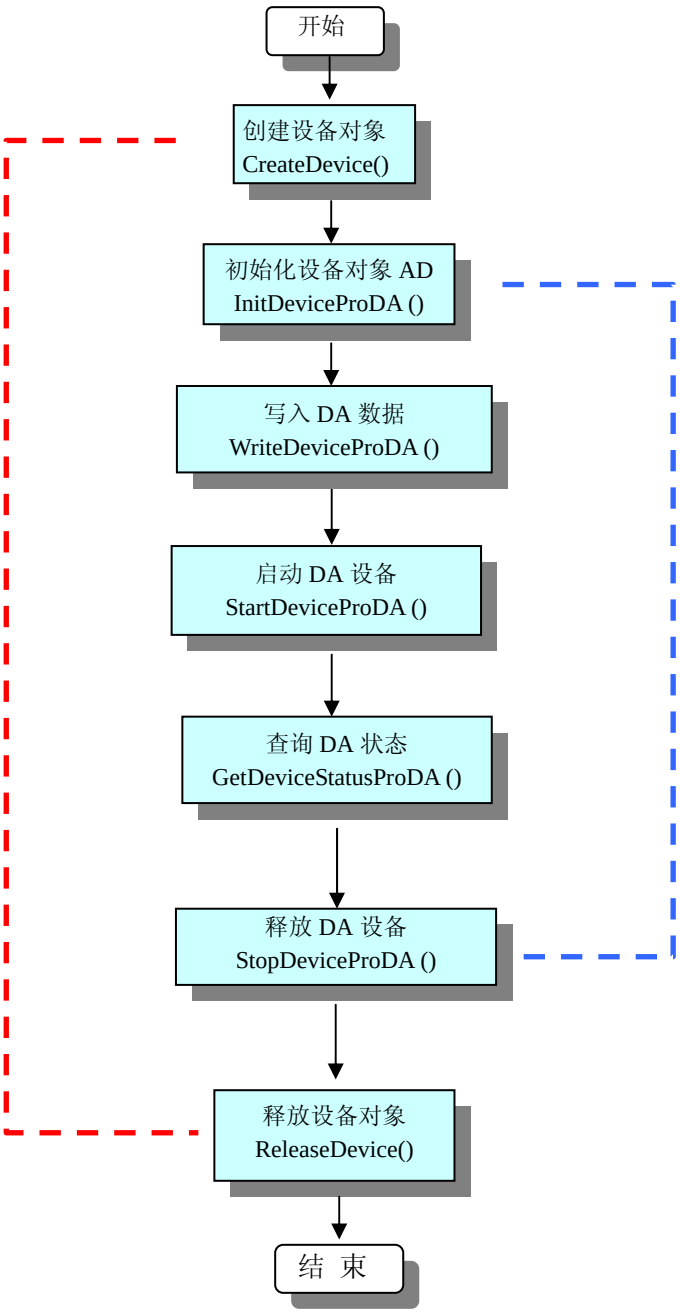


图 2.1　DA 输出实现过程

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则对用户来讲，可以说完全是辅助性的，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，您可能在新操作系统中无法继续使用您原有的老设备（除非您自己愿意去编写复杂的硬件驱动程序）。

第三章 设备操作函数接口介绍

第一节、设备驱动接口函数列表

（每个函数省略了前缀“PCI8252_”）

函数名	函数功能	备注
设备对象操作函数		
CreateDevice	创建设备对象	上层及底层用户
CreateDeviceEx	创建设备对象(用设备物理号)	上层及底层用户
GetDeviceCount	取得同一种设备的总台数	上层用户
GetDeviceCurrentID	取得指定设备句柄指向的设备 ID 号	上层用户
ListDeviceDlg	列表所有同一种设备的各种配置	上层用户
ReleaseDevice	关闭设备，且释放总线设备对象	上层及底层用户
DA 数据采样操作函数		
ResetDeviceDA	复位整个 DA 设备状态	上层用户
SetDeviceFreqDA	可动态改变 DA 采样频率	上层用户
SetOutputRangeDA	设置 DA 的输出范围	上层用户
ClearFIFODA	清除 FIFO 中的数据	上层用户
InitDeviceProDA	初始化设备上的 DA 部件准备传输	上层用户
StartDeviceProDA	启动 DA 设备，开始转换	上层用户
StopDeviceProDA	暂停 DA 设备	上层用户
GetDeviceStatusProDA	取得当前 DA 状态	上层用户
WriteDeviceProDA	用程序查询方式向 DA 的 FIFO 中写入批量数据	上层用户
ReleaseDeviceProDA	释放 DA 设备	上层用户
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

使用需知：

要使用如下函数关键的问题是：

Visual C++ & C++Builder：

首先，将 PCI8252.h 和 PCI8252.lib 文件从 Visual C++的源程序目录下的任意一个子目录下复制到您的源程序目录下（若有 Advanced 高级源程序目录，则最好选择它），然后在您的源程序中包含如下语句（若想在整个工程的所有源代码文件中使用本驱动，请您最好在 StdAfx.h 全局头文件中包含如下语句）：

#include “PCI8252.H”

那么对于导入库 PCI8252.lib 文件您则可以不必再加入您的工程，因为 PCI8252.h 头文件已帮助自动完成了。

C++ Builder:

首先，将 PCI8252.h 和 PCI8252.lib 文件从 C++Builder 的源程序目录任意一个子目录下复制到您的源程序目录下（若有 Advanced 高级源程序目录，则最好选择它），然后在您的源程序中包含如下语句：

#include “PCI8252.H”

那么对于导入库 PCI8252.lib 文件您则可以不必再加入您的工程，因为 PCI8252.h 头文件已帮助自动完成了。

Visual Basic:

要使用如下函数一个关键的问题是：

首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话中选择 PCI8252.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

Delphi:

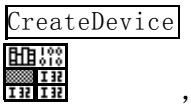
要使用如下函数一个关键的问题是：

首先必须将我们提供的单元模块文件（*.Pas）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 PCI8252.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。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。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：“PCI8252”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PCI8252;  // 注意：  在此加入驱动程序接口单元 PCI8252
```

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。其驱动程序接口单元模块的使用方法如下：



1. 在LabVIEW中打开PCI8252.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。
2. 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WriteDeviceProDA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。
3. 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++ & C++Builder：

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

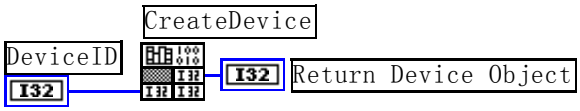
Visual Basic：

```
Declare Function CreateDevice Lib "PCI8252" (Optional ByVal DeviceLgcID As Integer = 0) As Long
```

Delphi：

```
Function CreateDevice(DeviceLgcID:Integer = 0):Integer;
                    StdCall; External 'PCI8252' Name ' CreateDevice';
```

LabVIEW：



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

参数：DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PCI8252 模板时，驱动程序逻辑号为“0”来确认和管理第一个设备，若用户接着再添加第二个 PCI8252 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则

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

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

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++ & 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
If hDevice = INVALID_HANDLE_VALUE Then    ' 判断设备对象句柄是否有效

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

◆ **创建设备对象函数（物理号）**

函数原型：

Visual C++ & C++Builder：

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

Visual Basic：

```
Declare Function CreateDeviceEx Lib "PCI8252" (Optional ByVal DevicePhysID As Integer = 0) As Long
```

Delphi：

```
Function CreateDeviceEx(DevicePhysID:Integer = 0):Integer;
                        StdCall; External 'PCI8252' Name ' CreateDeviceEx';
```

LabVIEW：

请参考相关演示程序。

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

参数：DevicePhysID 物理设备ID(Physic Device Identifier)标识号。由CreateDevice函数的DevieLgcID参数说明中可以看出，逻辑ID号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如A、B、C、D四个卡，构成 8 个通道（2*4），其通道序列为 0-15，每个通道驱动不同物理设备，我们要求A卡位于 0-1 通道上，B卡位于 2-3 通道上，C卡位于 4-5 通道上，而D卡则位于 6-7 通道上，而其逻辑设备ID号在同一台计算机上按不同顺序插入会发生变化，即便在不同计算机上按相同顺序插入也可能会因主板制造商的不同定义而发生变化，所以您可能由此无法确定 0-15 的通道所驱动的外部设备。那么如何将各个设备在设备链中的物理位置固定下来呢？那么物理设备ID的使用帮您解决了这个问题。它是在卡上提供了一个拔码器DID，可以由用户为各个设备手动设置不同的物理ID号，当调用CreateDeviceEx函数时，只需要指定该参数的值与您在拔码器上设定的值一样即可，驱动程序会自动跟踪拔码器值与此相等的设备。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。

由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得本计算机系统中 PCI8252 设备的总数量**

函数原型：

Visual C++ & C++Builder:

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

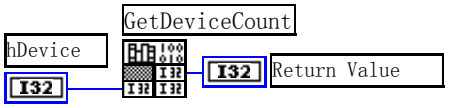
Visual Basic:

```
Declare Function GetDeviceCount Lib "PCI8252" (ByVal hDevice As Long ) As Integer
```

Delphi：

```
Function PCI8252_GetDeviceCount (hDevice : Integer):Integer;
                                 StdCall; External 'PCI8252' Name 'GetDeviceCount';
```

LabVIEW：



功能：取得 PCI8252 设备的数量。

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

返回值：返回系统中 PCI8252 的数量。

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得该设备当前逻辑 ID 和物理 ID**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PCI8252" (ByVal hDevice As Long, _
                                                   ByRef DeviceLgcID As Long, _
                                                   ByRef DevicePhysID As Long ) As Boolean
```

Delphi:

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DeviceLgcID: Pointer;
                             DevicePhysID: Pointer;): Boolean;
                             StdCall; External 'PCI8252' Name ' GetDeviceCurrentID ';
```

LabVIEW：

请参考相关演示程序。

功能：取得 PCI8252 设备的数量。

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 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拔码器 DID1 决定。

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

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PCI8252 设备各种配置信息**

函数原型：

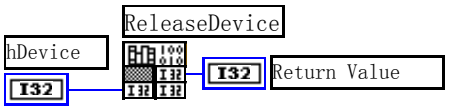
Visual C++ & C++Builder:

BOOL ListDeviceDlg (HANDLE hDevice)
Visual Basic:
Declare Function ListDeviceDlg Lib "PCI8252" (ByVal hDevice As Long) As Boolean
Delphi：
Function ListDeviceDlg (hDevice : Integer):Boolean;
StdCall; External 'PCI8252' Name ' ListDeviceDlg ';
LabVIEW：
请参考相关演示程序。

功能：列表系统中 PCI8252 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 PCI8252 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 "PCI8252" (ByVal hDevice As Long) As Boolean
Delphi：
Function ReleaseDevice(hDevice : Integer):Boolean;
StdCall; External 'PCI8252' 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 数据采样操作函数原型说明

♦ **复位整个 DA 设备状态**

函数原型：
Visual C++ & C++Builder:
BOOL ResetDeviceDA (HANDLE hDevice,
PPCI8252_PARA_DA pDAPara)
Visual Basic:
Declare Function ResetDeviceDA Lib "PCI8252" (ByVal hDevice As Long,_
ByRef pDAPara As PCI8252_PARA_DA) As Boolean
Delphi:
Function ResetDeviceDA (hDevice : Integer;
pDAPara : PCI8252_PARA_DA):Boolean;
StdCall; External 'PCI8252' Name ' ResetDeviceDA ';
LabView:
请参考相关演示程序。

功能：复位整个 DA 设备状态。
参数：
hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pDAPara 属于 PPCI8252_PARA 的结构指针型，它负责返回硬件参数值，关于结构指针类型 PPCI8252_PARA 请参考相应 PCI8252.h 或该结构的帮助文档的有关说明。

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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

- **动态设置 DA 的输出频率**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceFreqDA (HANDLE hDevice,
                      ULONG Frequency,
                      int nTimerChannel)
```

Visual Basic:

```
Declare Function SetDeviceFreqDA Lib "PCI8252" (ByVal hDevice As Long, _
                                                ByVal Frequency As Long,_
                                                ByVal nTimerChannel As Integer) As Boolean
```

Delphi:

```
Function SetDeviceFreqDA (hDevice : Integer;
                          Frequency : LongWord;
                          nTimerChannel : Integer):Boolean;
                          StdCall; External 'PCI8252' Name ' SetDeviceFreqDA ';
```

LabVIEW：

请参考演示源程序。

功能：在 DA 采样过程中，可动态设置 DA 的输出频率。

参数：

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

Frequency 时钟输出频率（Hz）。

nTimerChannel 时钟通道号(0 或 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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

- **设置 DA 的输出范围**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetOutputRangeDA (HANDLE hDevice,
                       ULONG OutputRange[4])
```

Visual Basic:

```
Declare Function SetOutputRangeDA Lib "PCI8252" (ByVal hDevice As Long, _
                                                 ByVal OutputRange (0 to 3)As Long) As Boolean
```

Delphi:

```
Function SetOutputRangeDA (hDevice : Integer;
                           OutputRange: Pointer):Boolean;
                           StdCall; External 'PCI8252' Name ' SetOutputRangeDA ';
```

LabVIEW：

请参考演示源程序。

功能：在 DA 采样过程中，设置 DA 的输出范围。

参数：

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

OutputRange 输出量程，分别控制四个通道。

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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

♦ 清除 FIFO 中的数据

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ClearFIFODA Lib "PCI8252" (ByVal hDevice As Long, _
                                            ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function ClearFIFODA (hDevice : Integer;
                      nDAChannel : Integer):Boolean;
                      StdCall; External 'PCI8252' Name ' ClearFIFODA';
```

LabVIEW：

请参考演示源程序。

功能：清除 FIFO 中的数据。

参数：

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

nDAChannel DA 输出通道，取值范围为[0, 3]。

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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

♦ 初始化设备对象

函数原型：

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

```
BOOL InitDeviceProDA (HANDLE hDevice,
                      LONG ClockSource,
                      int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceProDA Lib "PCI8252" (ByVal hDevice As Long, _
                                                ByVal ClockSource As Long,_
                                                ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function InitDeviceProDA (hDevice : Integer;
                          ClockSource : LongInt;
                          nDAChannel: Integer):Boolean;
                          StdCall; External 'PCI8252' Name ' InitDeviceProDA ';
```

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StartDeviceProDA (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。

ClockSource 时钟源选择，其取值为：

常量名	常量值	功能定义
PCI8252_IN_CLOCK0	0x0000	使用本设备上的内部时钟 0
PCI8252_IN_CLOCK1	0x0001	使用本设备上的内部时钟 1
PCI8252_OUT_CLOCK	0x0002	使用外部时钟

nDAChannel DA 输出通道，取值范围为[0, 3]。

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　ResetDeviceDA　SetDeviceFreqDA
SetOutputRangeDA　ClearFIFODA　InitDeviceProDA
StartDeviceProDA　StopDeviceProDA　GetDeviceStatusProDA
WriteDeviceProDA　ReleaseDeviceProDA　ReleaseDevice

◆ **启动 DA 设备**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StartDeviceProDA Lib "PCI8252" (ByVal hDevice As Long,_
                                                 ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function StartDeviceProDA (hDevice : Integer;
                           nDAChannel : Integer ): Boolean;
                           StdCall; External 'PCI8252' Name ' StartDeviceProDA ';
```

LabVIEW：

请参考相关演示程序。

功能：启动DA设备，它必须在调用InitDeviceProDA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, 3]。

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　ResetDeviceDA　SetDeviceFreqDA
SetOutputRangeDA　ClearFIFODA　InitDeviceProDA
StartDeviceProDA　StopDeviceProDA　GetDeviceStatusProDA
WriteDeviceProDA　ReleaseDeviceProDA　ReleaseDevice

◆ **暂停 DA 转换**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StopDeviceProDA Lib "PCI8252" (ByVal hDevice As Long,_
                                                ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function StopDeviceProDA (hDevice : Integer;
                          nDAChannel : Integer ): Boolean;
                          StdCall; External 'PCI8252' Name ' StopDeviceProDA ';
```

LabVIEW：

请参考相关演示程序。

功能：暂停DA设备。它必须在调用StartDeviceProDA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, 3]。

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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

♦ **取得 DA 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceStatusProDA (HANDLE hDevice,
                           PPCI8252_STATUS_DA pDAStatus)
```

Visual Basic:

```
Declare Function GetDeviceStatusProDA Lib "PCI8252" (ByVal hDevice As Long,_
                                                     ByRef pDAStatus As PCI8252_STATUS_DA) As Boolean
```

Delphi:

```
Function GetDeviceStatusProDA (hDevice : Integer;
                               pDAStatus : PPCI8252_STATUS_DA):Boolean;
                               StdCall; External 'PCI8252' Name ' GetDeviceStatusProDA ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProDA后，可以用此函数却查询FIFO状态，如是否溢出、半满、非空等信息。

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》。

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, ResetDeviceDA, SetDeviceFreqDA, SetOutputRangeDA, ClearFIFODA, InitDeviceProDA, StartDeviceProDA, StopDeviceProDA, GetDeviceStatusProDA, WriteDeviceProDA, ReleaseDeviceProDA, ReleaseDevice

♦ **用程序查询方式向 DA 的 FIFO 中写入批量数据**

函数原型：

Visual C++ & C++Builder:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       PLONG pDABuffer,
                       ULONG nWriteSizeWords,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceProDA Lib "PCI8252" (ByVal hDevice As Long,_
                                                 ByRef pDABuffer As Long,_
                                                 ByVal nWriteSizeWords As Long,_
                                                 ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceProDA (hDevice : Integer;
                           pDABuffer: Pointer;
                           nWriteSizeWords : LongWord;
                           nDAChannel: Integer):Boolean;
                           StdCall; External 'PCI8252' Name ' WriteDeviceProDA ';
```

LabVIEW:

请参考相关演示程序。

功能：向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）。
参数：
hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
pDABuffer 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 写入的点数（以字为单位）。
nDAChannel 通道号，取值范围为[0, 3]。
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　ResetDeviceDA　SetDeviceFreqDA
SetOutputRangeDA　ClearFIFODA　InitDeviceProDA
StartDeviceProDA　StopDeviceProDA　GetDeviceStatusProDA
WriteDeviceProDA　ReleaseDeviceProDA　ReleaseDevice

◆ **释放 DA 设备**
函数原型：
***Visual C++ & C++Builder*:**
```
BOOL ReleaseDeviceProDA (HANDLE hDevice,
                         int nDAChannel)
```
Visual Basic:
```
Declare Function ReleaseDeviceProDA Lib "PCI8252" (ByVal hDevice As Long, _
                                                   ByVal nDAChannel As Integer) As Boolean
```
Delphi:
```
Function ReleaseDeviceProDA (hDevice : Integer ;
                             nDAChannel As Integer) : Boolean;
                             StdCall; External 'PCI8252' Name ' ReleaseDeviceProDA ';
```
LabView：
请参考相关演示程序。

功能：释放DA设备。它必须在调用InitDeviceProDA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, 3]。
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　ResetDeviceDA　SetDeviceFreqDA
SetOutputRangeDA　ClearFIFODA　InitDeviceProDA
StartDeviceProDA　StopDeviceProDA　GetDeviceStatusProDA
WriteDeviceProDA　ReleaseDeviceProDA　ReleaseDevice

◆ **采样和和传输函数一般调用顺序**
①CreateDevice (创建设备对象)
②InitDeviceProDA (初始化设备)
③WriteDeviceProDA
④StartDeviceProDA (启动DA设备)
⑤GetDeviceStatusProDA（循环查询DA状态）
⑥StopDeviceProDA
⑦ReleaseDeviceProDA
⑧ReleaseDevice

关于调用过程的图形说明请参考《使用纲要》。

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：

Viusal C++ & C++ Builder:

```
BOOL SaveParaDA (HANDLE hDevice,
                 PPCI8252_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function SaveParaDA Lib "PCI8252" (ByVal hDevice As Long, _
                                           ByRef pDAPara As PCI8252_PARA_DA) As Boolean
```

Delphi:

```
Function SaveParaDA (hDevice : Integer;
                     pDAPara:PPCI8252_PARA_DA):Boolean;
                     StdCall; External 'PCI8252' Name ' SaveParaDA ';
```

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PPCI8252_PARA_DA的详细介绍请参考PCI8252.h或PCI8252.Bas或PCI8252.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。

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++ & C++ Builder:

```
BOOL LoadParaDA(HANDLE hDevice,
                PPCI8252_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function LoadParaDA Lib "PCI8252" (ByVal hDevice As Long, _
                                           ByRef pDAPara As PCI8252_PARA_DA) As Boolean
```

Delphi:

```
Function LoadParaDA (hDevice : Integer;
                     pDAPara:PPCI8252_PARA_DA):Boolean;
                     StdCall; External 'PCI8252' Name ' LoadParaDA ';
```

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PPCI8252_PARA_DA的结构指针类型，它负责返回硬件参数值，关于结构指针类型PPCI8252_PARA_DA请参考PCI8252.h或PCI8252.Bas或PCI8252.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。

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++ & C++ Builder:

```
BOOL ResetParaDA (HANDLE hDevice,
                  PPCI8252_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function ResetParaDA Lib "PCI8252" (ByVal hDevice As Long, _
                                            ByRef pDAPara As PCI8252_PARA_DA) As Boolean
```

Delphi:

```
Function ResetParaDA ( hDevice : Integer;
                       pDAPara:PPCI8252_PARA_DA):Boolean;
                       StdCall; External 'PCI8252' Name ' ResetParaDA ';
```

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PPCI8252_PARA_DA的详细介绍请参考PCI8252.h或PCI8252.Bas或PCI8252.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。

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

第四章 硬件参数结构

第一节、DA 硬件参数结构（PCI8252_PARA_DA）

Visual C++ & C++Builder：

```
typedef struct _PCI8252_PARA_DA
{
    LONG bEnableTimer0;      // 1:允许时钟 0 工作; 0:禁止时钟 0 工作;
    LONG Frequency0;         // 时钟 0 输出频率
    LONG bEnableTimer1;      // 1:允许时钟 1 工作; 0:禁止时钟 1 工作;
    LONG Frequency1;         // 时钟 1 输出频率
    LONG TriggerSource;      // 内触发和外触发方式选择
    LONG OutTriggerEdge;     // 外触发上升沿和下降沿类型选择
    LONG bClockOutput;       // 是否允许时钟输出
    LONG bImmediateClkOut;   // 是否立即启动定时器输出
    LONG ClockOutputNum;     // 选择哪一路时钟源作为输出(0:Timer0; 1:Timer1)
} PCI8252_PARA_DA, *PPCI8252_PARA_DA;
```

Visual Basic：

```
Type PCI8252_PARA_DA
    bEnableTimer0 As Long          ' 1:允许时钟 0 工作; 0:禁止时钟 0 工作
    Frequency0 As Long             ' 时钟 0 输出频率
    bEnableTimer1 As Long          ' 1:允许时钟 1 工作; 0:禁止时钟 1 工作
    Frequency1 As Long             ' 时钟 1 输出频率
    TriggerSource As Long          ' 内触发和外触发方式选择
    OutTriggerEdge As Long         ' 外触发上升沿和下降沿类型选择
    bClockOutput As Long           ' 是否允许时钟输出
    bImmediateClkOut As Long       ' 是否立即启动定时器输出
    ClockOutputNum As Long         ' 选择哪一路时钟源作为输出(0:Timer0; 1:Timer1)
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PPCI8252_PARA_DA = ^ PCI8252_PARA_DA;   // 指针类型结构
    PCI8252_PARA_DA = record      // 标记为记录型
```

```
    bEnableTimer0 : LongInt;        // 1:允许时钟 0 工作; 0:禁止时钟 0 工作
    Frequency0 : LongInt;           // 时钟 0 输出频率
    bEnableTimer1: LongInt;         // 1:允许时钟 1 工作; 0:禁止时钟 1 工作
    Frequency1 : LongInt;           // 时钟 1 输出频率
    TriggerSource : LongInt;        // 内触发和外触发方式选择
    OutTriggerEdge : LongInt;       // 外触发上升沿和下降沿类型选择
    bClockOutput : LongInt;         // 是否允许时钟输出
    bImmediateClkOut : LongInt;     // 是否立即启动定时器输出
    ClockOutputNum : LongInt;       // 选择哪一路时钟源作为输出(0:Timer0; 1:Timer1)
End;
```

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InitDeviceProDA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。

bEnableTimer0 = 1:允许时钟 0 工作; = 0:禁止时钟 0 工作。
Frequency0 时钟 0 输出频率。
bEnableTimer1 = 1:允许时钟 1 工作; = 0:禁止时钟 1 工作。
Frequency1 时钟 1 输出频率。

TriggerSource DA 触发源选择。选项值如下表：

常量名	常量值	功能定义
PCI8252_IN_TRIGGER	0x0000	内触发启动方式
PCI8252_OUT_TRIGGER	0x0001	外触发启动方式

OutTriggerEdge 外触发上升沿和下降沿类型选择。选项值如下表：

常量名	常量值	功能定义
PCI8252_FALLING_EDGE	0x0000	外触发上升沿触发方式
PCI8252_RISING_EDGE	0x0001	外触发下降沿触发方式

bClockOutput 是否允许时钟输出。
bImmediateClkOut 是否立即启动定时器输出。
ClockOutputNum 选择哪一路时钟源作为输出(0:Timer0; 1:Timer1)。

第二节、DA 状态参数结构（PCI8252_STATUS_DA）

Visual C++ & C++Builder：

```
typedef struct _PCI8252_STATUS_DA
{
    ULONG bOverflow0;   // 板载 FIFO 存储器的 DA0 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    ULONG bHalf0;       // 板载 FIFO 存储器的 DA0 半满标志，= TRUE 半满以上，= FALSE 半满以下
    ULONG bNotEmpty0;   // 板载 FIFO 存储器的 DA0 非空标志，= TRUE 非空，= FALSE 空
    ULONG bOverflow1;   // 板载 FIFO 存储器的 DA1 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    ULONG bHalf1;       // 板载 FIFO 存储器的 DA1 半满标志，= TRUE 半满以上，= FALSE 半满以下
    ULONG bNotEmpty1;   // 板载 FIFO 存储器的 DA1 非空标志，= TRUE 非空，= FALSE 空
    ULONG bOverflow2;   // 板载 FIFO 存储器的 DA2 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    ULONG bHalf2;       // 板载 FIFO 存储器的 DA2 半满标志，= TRUE 半满以上，= FALSE 半满以下
    ULONG bNotEmpty2;   // 板载 FIFO 存储器的 DA2 非空标志，= TRUE 非空，= FALSE 空
    ULONG bOverflow3;   // 板载 FIFO 存储器的 DA3 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢
```

出
    ULONG bHalf3;   // 板载 FIFO 存储器的 DA3 半满标志，= TRUE 半满以上，= FALSE 半满以下
    ULONG bNotEmpty3; // 板载 FIFO 存储器的 DA3 非空标志，= TRUE 非空，= FALSE 空
} PCI8252_STATUS_DA, *PPCI8252_STATUS_DA;

Visual Basic：

```
Type PCI8252_ STATUS _DA
    bOverflow0 As Long     '板载 FIFO 存储器的 DA0 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    bHalf0 As Long         '板载 FIFO 存储器的 DA0 半满标志，= TRUE 半满以上，= FALSE 半满以下
    bNotEmpty0 As Long     '板载 FIFO 存储器的 DA0 非空标志，= TRUE 非空，= FALSE 空
    bOverflow1 As Long     '板载 FIFO 存储器的 DA1 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    bHalf1 As Long         '板载 FIFO 存储器的 DA1 半满标志，= TRUE 半满以上，= FALSE 半满以下
    bNotEmpty1 As Long     '板载 FIFO 存储器的 DA1 非空标志，= TRUE 非空，= FALSE 空
    bOverflow2 As Long     '板载 FIFO 存储器的 DA2 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    bHalf2 As Long         '板载 FIFO 存储器的 DA2 半满标志，= TRUE 半满以上，= FALSE 半满以下
    bNotEmpty2 As Long     '板载 FIFO 存储器的 DA2 非空标志，= TRUE 非空，= FALSE 空
    bOverflow3As Long      '板载 FIFO 存储器的 DA3 溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    bHalf3 As Long         '板载 FIFO 存储器的 DA3 半满标志，= TRUE 半满以上，= FALSE 半满以下
    bNotEmpty3 As Long     '板载 FIFO 存储器的 DA3 非空标志，= TRUE 非空，= FALSE 空
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PPCI8252_ STATUS _DA = ^ PCI8252_ STATUS _DA;   // 指针类型结构
    PCI8252_ STATUS _DA = record          // 标记为记录型
        bOverflow0 : LongInt; //板载 FIFO 存储器的 DA0 溢出标志，= TRUE 已发生溢出，= FALSE 未发生
    溢出
        bHalf0 : LongInt;        //板载 FIFO 存储器的 DA0 半满标志，= TRUE 半满以上，= FALSE 半满以下
        bNotEmpty0 : LongInt;//板载 FIFO 存储器的 DA0 非空标志，= TRUE 非空，= FALSE 空
        bOverflow1 : LongInt;//板载 FIFO 存储器的 DA1 溢出标志，= TRUE 已发生溢出，= FALSE 未发生
    溢出
        bHalf1 : LongInt;        //板载 FIFO 存储器的 DA1 半满标志，= TRUE 半满以上，= FALSE 半满以下
        bNotEmpty1 : LongInt;//板载 FIFO 存储器的 DA1 非空标志，= TRUE 非空，= FALSE 空
        bOverflow2 : LongInt; //板载 FIFO 存储器的 DA2 溢出标志，= TRUE 已发生溢出，= FALSE 未发生
    溢出
        bHalf2 : LongInt;        //板载 FIFO 存储器的 DA2 半满标志，= TRUE 半满以上，= FALSE 半满以下
        bNotEmpty2 : LongInt;//板载 FIFO 存储器的 DA2 非空标志，= TRUE 非空，= FALSE 空
        bOverflow3 : LongInt;//板载 FIFO 存储器的 DA3 溢出标志，= TRUE 已发生溢出，= FALSE 未发生
    溢出
        bHalf3 : LongInt;        //板载 FIFO 存储器的 DA3 半满标志，= TRUE 半满以上，= FALSE 半满以下
        bNotEmpty3 : LongInt;//板载 FIFO 存储器的 DA3 非空标志，= TRUE 非空，= FALSE 空
End;
```

LabVIEW:

请参考相关演示程序。

bOverflow0 板载 FIFO 存储器的 DA0 溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出。
bHalf0 板载 FIFO 存储器的 DA0 半满标志，= TRUE 半满以上，= FALSE 半满以下。
bNotEmpty0 板载 FIFO 存储器的 DA0 非空标志，= TRUE 非空，= FALSE 空。
bOverflow1 板载 FIFO 存储器的 DA1 溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出。
bHalf1 板载 FIFO 存储器的 DA1 半满标志，= TRUE 半满以上，= FALSE 半满以下。
bNotEmpty1 板载 FIFO 存储器的 DA1 非空标志，= TRUE 非空，= FALSE 空。
bOverflow2 板载 FIFO 存储器的 DA2 溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出。
bHalf2 板载 FIFO 存储器的 DA2 半满标志，= TRUE 半满以上，= FALSE 半满以下。

bNotEmpty2 板载 FIFO 存储器的 DA2 非空标志，= TRUE 非空，= FALSE 空。
bOverflow3 板载 FIFO 存储器的 DA3 溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出。
bHalf3 板载 FIFO 存储器的 DA3 半满标志，= TRUE 半满以上，= FALSE 半满以下。
bNotEmpty3 板载 FIFO 存储器的 DA3 非空标志，= TRUE 非空，= FALSE 空。

第五章 数据格式转换与排列规则

第一节、DA 的电压值如何转换成输出到 DA 转换器的 LSB 原码数据？

量程(伏)	计算机语言换算公式	Lsb 取值范围
0～5000mV	Lsb = Volt / (5000.00 / 65536)	[0，65535]
0～10000mV	Lsb = Volt / (10000.00 / 65536)	[0，65535]
±2500mV	Lsb = Volt / (5000.00 / 65536) + 32768	[0，65535]
±5000mV	Lsb = Volt / (10000.00 / 65536) + 32768	[0，65535]

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

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，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)	768K 至(1024K-1)

图 5.1

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

图 5.2

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 PCI8252_PARA_DA 中的 SegmentInfo 决定的。

表 5.2.1

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

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WriteDeviceProDA函数进行批量DA数据输出

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 PCI8252.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DA 简易源程序]**

其简易程序默认存放路径为：系统盘\ART\PCI8252\SAMPLES\VC\SIMPLE\DA\BULK

其他语言的演示可以用上面类似的方法找到。

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 8100.h 和 DADoc.cpp)。

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

其默认存放路径为：系统盘\ART\8100\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，它只是您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数列表

（每个函数省略了前缀“PCI8252_”）

函数名	函数功能	备注
① 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定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	
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	
④ 文件对象操作函数		
CreateFileObject	创建文件对象	
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	不消耗 CPU 时间
getch	等待并获取用户击键值	
GetLastErrorEx	获取错误信息	
RemoveLastErrorEx	移除错误信息	

第二节、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)
```

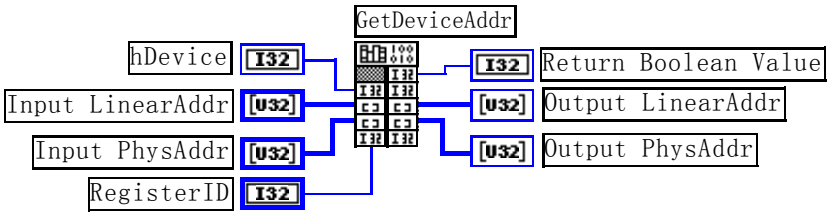
Visual Basic:.

```
Declare Function GetDeviceAddr Lib "PCI8252" (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 'PCI8252' Name 'GetDeviceAddr';
```

LabVIEW:



功能：取得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。

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　GetDeviceBar
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 pulBar[6])
```

Visual Basic:.

```
Declare Function GetDeviceBar Lib "PCI8252" (ByVal hDevice As Long, _
                                             ByRef pulBar(0 to 5) As Long) As Boolean
```

Delphi:

```
Function GetDeviceBar (hDevice : Integer;
                       pulBar : Pointer):Boolean;
                       StdCall; External 'PCI8252' Name ' GetDeviceBar ';
```

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。

pulBar 返回 BAR 所有地址，具体 BAR 中有多少可用地址请看硬件说明书。

返回值：如果执行成功，则返回TRUE，否则会返回FALSE。用户可用GetLastErrorEx捕获当前错误码，并加以分析。

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

- **以单字节（即 8 位）方式写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)
```

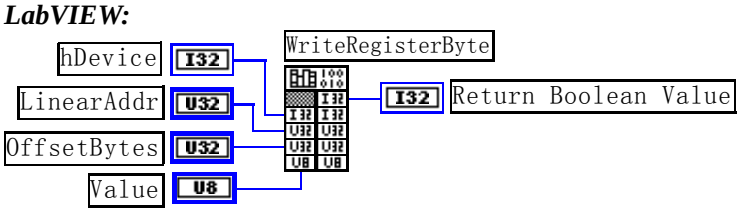
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI8252" (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 'PCI8252' Name 'WriteRegisterByte';
```

LabVIEW:



功能：以单字节（即 8 位）方式写内存映射寄存器。

参数：

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

LinearAddr 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　GetDeviceBar　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)
```

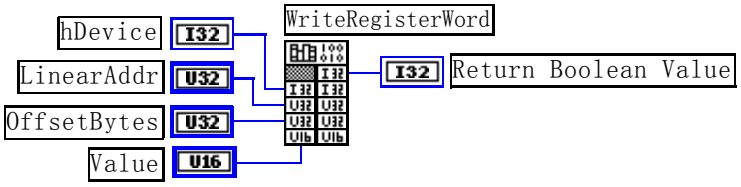
Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI8252" ( 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 'PCI8252' Name 'WriteRegisterWord';
```

LabVIEW:



功能：以双字节（即 16 位）方式写内存映射寄存器。

参数：

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

LinearAddr 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　GetDeviceBar　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 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
```

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

◆ **以四字节（即 32 位）方式写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)
```

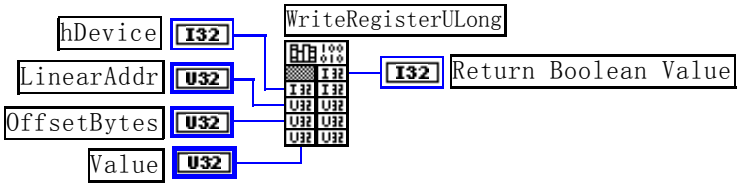
Visual Basic:

```
Declare Sub WriteRegisterULong Lib "PCI8252" (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 'PCI8252' Name 'WriteRegisterULong';
```

LabVIEW:



功能：以四字节（即 32 位）方式写内存映射寄存器。

参数：

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

LinearAddr 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　GetDeviceBar
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 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
```

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

◆ **以单字节（即 8 位）方式读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)
```

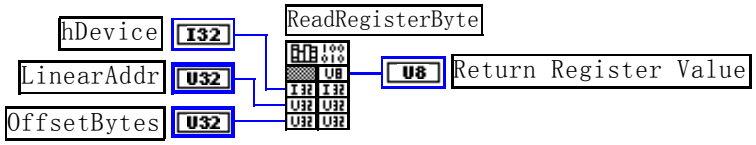
Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI8252" (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 'PCI8252' Name 'ReadRegisterByte';
```

LabVIEW:



功能：以单字节（即 8 位）方式读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决定。

LinearAddr 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	GetDeviceBar
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)
```

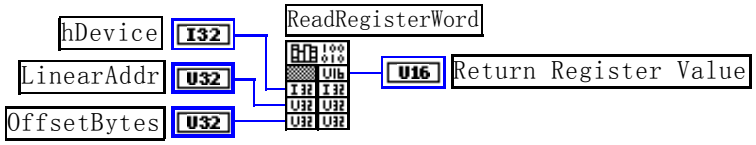
Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI8252" ( 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 'PCI8252' Name 'ReadRegisterWord';
```

LabVIEW:



功能：以双字节（即 16 位）方式读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决定。

LinearAddr 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, GetDeviceBar, 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)
```

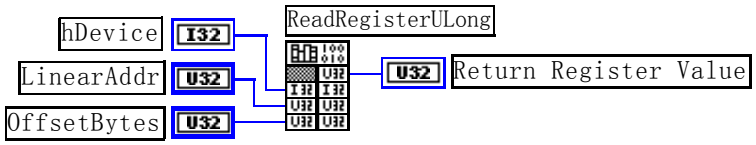
Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI8252" (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 'PCI8252' Name 'ReadRegisterULong';
```

LabVIEW:



功能：以四字节（即 32 位）方式读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决定。

LinearAddr 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 ReadRegisterULong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	GetDeviceBar
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、IO 端口读写函数

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

- **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

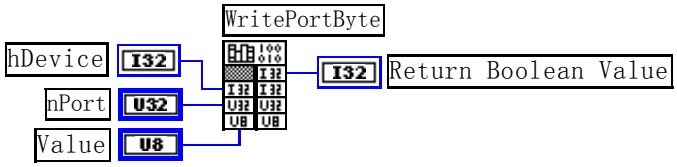
Visual Basic:

```
Declare Function WritePortByte Lib "PCI8252" (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 'PCI8252' 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**

函数原型：

Visual C++ & C++ Builder:

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

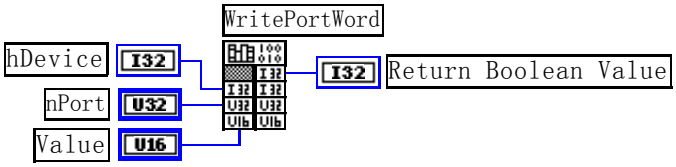
Visual Basic:

```
Declare Function WritePortWord Lib "PCI8252" (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 'PCI8252' 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**

函数原型：
Visual C++ & C++ Builder:

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

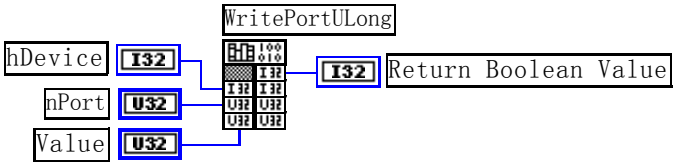
Visual Basic:

```
Declare Function WritePortULong Lib "PCI8252" (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 'PCI8252' 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**

函数原型：
Visual C++ & C++ Builder:

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

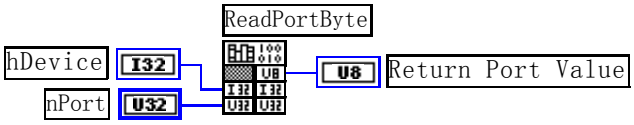
Visual Basic:

```
Declare Function ReadPortByte Lib "PCI8252" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

```
Function ReadPortByte(hDevice : Integer;
                      nPort:LongWord):Byte;
                      StdCall; External 'PCI8252' 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**

函数原型：
Visual C++ & C++ Builder:

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

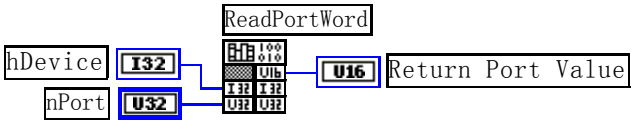
Visual Basic:

```
Declare Function ReadPortWord Lib "PCI8252" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Integer
```

Delphi:

```
Function ReadPortWord(hDevice : Integer;
                      nPort:LongWord):Word;
                      StdCall; External 'PCI8252' 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**

函数原型：
Visual C++ & C++ Builder:

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

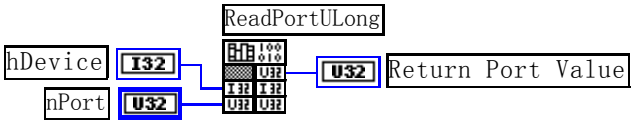
Visual Basic:

```
Declare Function ReadPortULong Lib "PCI8252" (ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                       nPort:LongWord):LongWord;
                       StdCall; External 'PCI8252' 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

♦ **创建内核系统事件**

函数原型：

Visual C++ & C++ Builder

```
HANDLE CreateSystemEvent(void)
```

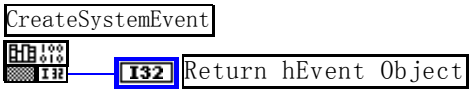
Visual Basic:

```
Declare Function CreateSystemEvent Lib " PCI8252 " () As Long
```

Delphi:

```
Function CreateSystemEvent():Integer;
                    StdCall; External 'PCI8252' 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**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function ReleaseSystemEvent Lib " PCI8252 " (ByVal hEvent As Long) As Boolean
```

Delphi:

```
Function ReleaseSystemEvent(hEvent : Integer):Boolean;
                    StdCall; External 'PCI8252' Name ' ReleaseSystemEvent ';
```

LabVIEW:

请参考相关演示程序。

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 创建。

返回值：若成功，则返回 TRUE。

第五节、文件对象操作函数

♦ **初始化设备文件对象**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function CreateFileObject Lib "PCI8252" (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 'PCI8252' name 'CreateFileObject';
```

LabVIEW:

请参考相关演示程序。

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:\\PCI8252\\Data.Dat"，在 Basic 中，其语法格式如："C:\PCI8252\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表 (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
PCI8252_modeRead	0x0000	只读文件方式
PCI8252_modeWrite	0x0001	只写文件方式
PCI8252_modeReadWrite	0x0002	既读又写文件方式
PCI8252_modeCreate	0x1000	如果文件不存在可以创建该文件，否则重建此文件，并清 0
PCI8252_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 "PCI8252" (ByRef hFileObject As Long,_
                                          ByVal pDataBuffer As Integer,_
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer:PWordArray;
                   nWriteSizeBytes: LongInt):Boolean;
                   Stdcall; external 'PCI8252' name 'WriteFile';
```

LabVIEW:

请参考相关演示程序。

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

参数：

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

pDataBuffer 用户数据空间地址。

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++ & C++ Builder:

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

Visual Basic:

```
Declare Function ReadFile Lib "PCI8252" ( ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal nOffsetBytes As Long, _
```

ByVal nReadSizeBytes As Long) As Boolean

Delphi:

```
Function ReadFile(hFileObject: Integer;
                  pDataBuffer:PWordArray;
                  nOffsetBytes: LongInt;
                  nReadSizeBytes: LongInt):Boolean;
                  Stdcall; external 'PCI8252' name 'ReadFile';
```

LabVIEW:

请参考相关演示程序。

功能：将磁盘数据从指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。

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++ & C++ Builder:

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

Visual Basic:

```
Declare Function SetFileOffset Lib "PCI8252" (ByVal hFileObject As Long,_
                                              ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset (hFileObject: Integer;
                        nOffsetBytes: LongInt):Boolean;
                        Stdcall; external 'PCI8252' Name ' SetFileOffset ';
```

LabVIEW:

请参考相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由CreateFileObject创建。

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

```
ULONG GetFileLength (HANDLE hFileObject)
```

Visual Basic:

```
Declare Function GetFileLength Lib "PCI8252" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject: Integer):LongWord;
                        Stdcall; external 'PCI8252' Name ' GetFileLength ';
```

LabVIEW:

请参考相关演示程序。

功能：取得文件长度（字节）。

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

返回值：若成功，则实际长度（字节），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++ & C++ Builder:

```
BOOL ReleaseFile(HANDLE hFileObject)
```

Visual Basic:

```
Declare Function ReleaseFile Lib "PCI8252" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
Function ReleaseFile(hFileObject: Integer):Boolean;
                    Stdcall; external 'PCI8252' Name 'ReleaseFile';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备文件对象。

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

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用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**

函数原型：

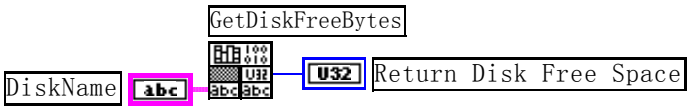
Visual C++ & C++ Builder:

```
ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName )
```

Visual Basic:

```
Declare Function GetDiskFreeBytes Lib "PCI8252" (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++ & C++ Builder:

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

Visual Basic:

```
Declare Function SaveParaInt Lib "PCI8252" (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 'PCI8252' 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\PCI8252\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++ & C++ Builder:

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

Visual Basic:

```
Declare Function LoadParaInt Lib "PCI8252" (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 'PCI8252' 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\PCI8252\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 "PCI8252" (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 'PCI8252' 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\PCI8252\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++ & C++ Builder:

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

Visual Basic:

```
Declare Function LoadParaString Lib "PCI8252" (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 'PCI8252' 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\PCI8252\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++ & C++ Builder:

```
BOOL kbhit (void)
```

Visual Basic:

```
Declare Function kbhit Lib "PCI8252" () As Boolean
```

Delphi:

```
Function kbhit () : Boolean;
                  Stdcall; external 'PCI8252' 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 "PCI8252" () As String
```

Delphi:

```
Function getch () : char;
                    Stdcall; external 'PCI8252' 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 "PCI8252" (ByVal strFuncName As String,_
                                               ByVal strErrorMsg As String) As Long
```

Delphi:

```
Function GetLastErrorEx (strFuncName: String;
                         strErrorMsg: String) : LongWord;
                         Stdcall; external 'PCI8252' 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 PCI8252_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8252_InitDeviceAD”，否则得不到相应信息。

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

返回值：返回错误码。

相关函数：无。

Visual C++ & 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 = PCI8252_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 = PCI8252_GetLastErrorEx(“PCI8252_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 = PCI8252_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 = PCI8252_GetLastErrorEx("PCI8252_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
    :
```

♦ **移除指定函数的最后一次错误信息**

函数原型：

Visual C++ & C++ Builder:

```
BOOL RemoveLastErrorEx (LPCTSTR strFuncName)
```

Visual Basic:

```
Declare Function RemoveLastErrorEx Lib "PCI8252" (ByVal strFuncName As String) As Long
```

Delphi:

```
Function RemoveLastErrorEx (strFuncName: String) : LongWord;
                                Stdcall; external 'PCI8252' Name ' RemoveLastErrorEx ';
```

LabVIEW:

详见相关演示程序。

功能：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 PCI8252_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8252_InitDeviceAD”，否则得不到相应信息。

返回值：返回错误码。

相关函数：无。