PCI9103 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司

产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_ 则被省略。如 PCI9103_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 绪 论

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。诸如InitDeviceDA、WriteDeviceBulkDA等。而底层用户函数如 WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上可以不 InitDeviceDA必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。因为上层函数的命名、参数的命名极其规范。

第二节、如何管理 PCI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 CreateDevice函数创建一个设备对象句柄hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给其他函数，如 InitDeviceDA可以使用hDevice句柄以程序查询方式初始化设备的DA部件，WriteDeviceBulkDA函数可以用hDevice句柄实现对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 InitDeviceDA函数初始化DA部件，关于频率等参数 InitDeviceDA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 WriteDeviceBulkDA将准备好的DA数据写入板载RAM中，接着用 EnableDeviceDA即可启动DA部件，开始DA输出，GetDevStatusDA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 DisableDeviceDA，当您需要关闭DA设备时，ReleaseDeviceDA便可帮您实现（但设备对象hDevice依然存在）。

第四节、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　PCI 即插即用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“PCI9103_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PCI 设备对象	上层及底层用户
GetDeviceCount	取得同一种 PCI 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 PCI 设备的各种配置	上层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
② DA 数据采样操作函数		
SetDevTrigLevelDA	设置 DA 的触发电平	上层用户
SetDevImpedanceDA	设置输出阻抗	
SetDevFrequencyDA	可动态改变 DA 采样频率	上层用户
ReadSegmentInfo	读段信息	
InitDeviceDA	初始化 PCI 设备上的 DA 部件准备传输	上层用户
WriteDeviceOneDA	DA 单点输出	
WriteDeviceBulkDA	批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中	

ReadDeviceBulkDA	以批量方式将板载 RAM 中的 DA 数据回读至主机的用户缓冲区	
EnableDeviceDA	启动 DA 设备，开始转换	上层用户
SetDeviceTrigDA	软件产生触发事件	
GetDevStatusDA	取得当前 DA 状态	上层用户
DisableDeviceDA	暂停 DA 设备	上层用户
ReleaseDeviceDA	释放 DA 设备	上层用户
StartCalibration	启动 DA 校准	
GetDACalibration	设备 DA 校准	
SetDACalibration	设备 DA 校准	
StopCalibration	停止 DA 校准	
③ DA 硬件参数系统保存、读取函数		
LoadParaDA	从 Windows 系统中读入硬件参数	上层用户
SaveParaDA	往 Windows 系统写入设备硬件参数	上层用户
ResetParaDA	将硬件参数结构体值复位为出厂默认值	上层用户
④ 开关量函数		
GetDeviceDI	开关输入函数	
SetDeviceDO	开关输出函数	

使用需知：

要使用如下函数关键的问题是：

Visual C++：

首先，将 PCI9103.h 和 PCI9103.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 “PCI9103.H”

那么对于导入库 PCI9103.lib 文件您则可以不必再加入您的工程，因为 PCI9103.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 PCI9103.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。其驱动程序接口单元模块的使用方法如下：

1. 在LabVIEW中打开PCI9103.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 WriteDeviceBulkDA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：

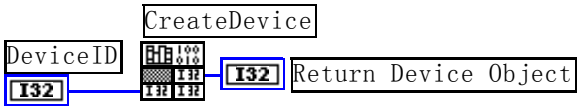
Visual C++：

HANDLE CreateDevice (int DeviceLgcID=0)

Visual Basic：

Declare Function CreateDevice Lib "PCI9103_32" (Optional ByVal DeviceLgcID As Integer = 0) As long

LabVIEW：



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

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

相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例:

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

Visual Basic 程序举例:

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


Else
```

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

♦ **取得本计算机系统中 PCI9103 设备的总数量**

函数原型：

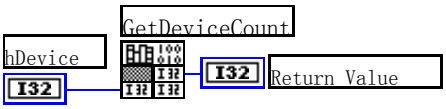
Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceCount Lib "PCI9103_32" (ByVal hDevice As Long ) As Integer
```

LabVIEW：



功能：取得 PCI9103 设备的数量。

参数：hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：返回系统中 PCI9103 的数量。

相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦ **取得该设备当前逻辑 ID 和物理 ID**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PCI9103_32" (ByVal hDevice As Long
                                                      ByVal DeviceLgcID As Long
                                                      ByVal DevicePhysID As Long)As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：取得 PCI9103 设备的数量。

参数：hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE

相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PCI9103 设备各种配置信息**

函数原型：

Visual C++:

```
BOOL ListDeviceDlg (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ListDeviceDlg Lib "PCI9103_32" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI9103 的硬件配置信息。
参数：hDevice 设备对象句柄，它应由 CreateDevice创建。
返回值：若成功，则弹出对话框控件列表所有 PCI9103 设备的配置情况。
相关函数： CreateDevice ReleaseDevice

- **释放设备对象所占的系统资源及设备对象**

函数原型：

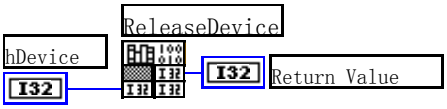
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "PCI9103_32" (ByVal hDevice As Long) As Boolean

LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。
参数：hDevice 设备对象句柄，它应由 CreateDevice创建。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
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

- **设置触发电平**

函数原型：

Visual C++：

BOOL SetDevTrigLevelDA (HANDLE hDevice, float fTrigLevelVolt)

Visual Basic:

```
Declare Function SetDevTrigLevelDA Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                     ByVal fTrigLevelVolt As Single ) As Boolean
```

LabView：

请参考相关演示程序。

功能：设置触发电平，该触发电平对所有 DA 通道同时有效。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
nTrigLevelVolt 触发电平值，单位为 mV，其取值范围为[0, +10000.0]，注意本设备的触发电平在其取值范围中可分为 256 个分辨率。也就是说触发电平的设置精度为 39.0625 毫伏(10000.0/256)。
返回值：如果初始化设备对象成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA

WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

♦ **动态改变采样频率**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function SetDevFrequencyDA Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                     ByVal nFrequency As Long,_
                                                     ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考演示源程序。

功能：在 DA 采样过程中，可动态改变采样频率。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

nFrequency 采样频率，取值范围为[1Hz, 1.7MHz]，其单位为 Hz。

nDAChannel DA 通道号，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。

相关函数：

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

♦ **读段信息**

函数原型：

Visual C++:

```
BOOL ReadSegmentInfo(HANDLE hDevice,
                     LONG SegmentCount,
                     PCI9103_SEGMENT_INFO SegmentInfo[],
                     int nDAChannel)
```

Visual Basic:

```
Declare Function ReadSegmentInfo Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                   ByVal SegmentCount As Long,_
                                                   ByRef SegmentInfo() As PCI9103_SEGMENT_INFO,_
                                                   ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考演示源程序。

功能： 读段信息。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

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

SegmentInfo[] 段信息集合，属于 PCI9103_SEGMENT_INFO 的结构数据类型，它的有效元素个数由 SegmentCount 参数决定。因此用户分配的段信息集合不应小于 SegmentCount 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中，它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

nDAChannel DA 通道号，取值范围为[0, 3]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA, WriteDeviceBulkDA, ReadDeviceBulkDA, EnableDeviceDA, SetDeviceTrigDA, GetDevStatusDA, DisableDeviceDA, ReleaseDeviceDA, ReleaseDevice

◆ **初始化设备对象**

函数原型

Visual C++：

```
BOOL InitDeviceDA(HANDLE hDevice,
                  LONG SegmentCount,
                  PCI9103_SEGMENT_INFO SegmentInfo[],
                  PPCI9103_PARA_DA pDAPara,
                  int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceDA Lib "PCI9103_32" (   ByVal hDevice As Long, _
                                                   ByVal SegmentCount As Long,_
                                                   ByRef SegmentInfo() As PCI9103_SEGMENT_INFO,_
                                                   ByRef pDAPara As PCI9103_PARA_DA,_
                                                   ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：它负责初始化设备对象中的DA部件，为设备操作就绪有关工作做准备，如预置DA采集通道、采样频率等。但它并不启动DA设备，若要启动DA设备，须在调用此函数之后再调用 EnableDeviceDA(但DA要实际输出波形，则一般要等待某种触发事件的到来)。

参数：

hDevice 设备对象句柄，它应由 CreateDevice。

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

SegmentInfo[] 段信息集合，属于 PCI9103_SEGMENT_INFO 的结构数据类型，它的有效元素个数由 SegmentCount 参数决定。因此用户分配的段信息集合不应小于 SegmentCount 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中，它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

pDAPara 设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如采样频率等。关于具体操作请参考《DA硬件参数结构》。

nDAChannel DA 通道号，取值范围为[0, 3]。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA, WriteDeviceBulkDA, ReadDeviceBulkDA, EnableDeviceDA, SetDeviceTrigDA, GetDevStatusDA, DisableDeviceDA, ReleaseDeviceDA, ReleaseDevice

♦ **DA 单点输出**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function WriteDeviceOneDA Lib "PCI9103_32" (ByVal hDevice As Long,_
                                                    ByVal ulDataCode As Long,_
                                                    ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：DA 单点输出。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

ulDataCode 输出码值（0—4095）。

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

♦ **批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中**

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function WriteDeviceBulkDA Lib "PCI9103_32" (
                                   ByVal hDevice as Long, _
                                   ByRef DABuffer() As Integer, _
                                   ByVal nWriteSizeWords As Long, _
                                   ByRef nRetSizeWords As Long, _
```

ByVal nDAChannel As Integer) As Boolean

LabView：

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄,它应由 CreateDevice创建。

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

nWriteOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)。此参数不能超过 RAM 的最大长度(字/点)。

nWriteSizeWords 指定一次往物理缓冲区由 nWriteOffsetWords 参数指定偏移位置开始写入的数据长度。注意此参数的值与 nWriteOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。同时此参数值不能大于 DABuffer[]指定的缓冲区的长度。

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

nDAChannel DA 通道号，取值范围为[0, 3]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

相关函数：

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

♦ **用此函数将板载 RAM 中的数据读回计算机(程序方式)**

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function ReadDeviceBulkDA Lib "PCI9103_32" (
                                ByVal hDevice as Long, _
                                ByRef DABuffer() As Integer, _
                                ByVal nReadSizeWords As Long, _
                                ByRef nRetSizeWords As Long, _
                                 ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

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

nReadOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)。此参数不能超过 RAM 的最大长度(字/点)。

nReadSizeWords 指定一次从物理缓冲区由 nReadOffsetWords 参数指定偏移位置开始读入的数据长度。注意此参数的值与 nWriteOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。同时此参数值不能大于 DABuffer[]指定的缓冲区的长度。

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

nDAChannel DA 通道号，取值范围为[0, 3]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA, WriteDeviceBulkDA, ReadDeviceBulkDA, EnableDeviceDA, SetDeviceTrigDA, GetDevStatusDA, DisableDeviceDA, ReleaseDeviceDA, ReleaseDevice

♦ 启动 DA 设备

函数原型

Visual C++:

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

Visual Basic:

```
Declare Function EnableDeviceDA Lib "PCI9103_32" (ByVal hDevice As Long,_
                                                  ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：启动DA设备，它必须在调用 InitDeviceDA后才能调用此函数。调用该函数后它可能立即启动，这就要取决您选择的触发方式或触发源，详细请参考后面的《触发功能详述》。

参数：

hDevice 设备对象句柄，它应由 CreateDevice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。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA, WriteDeviceBulkDA, ReadDeviceBulkDA, EnableDeviceDA, SetDeviceTrigDA, GetDevStatusDA, DisableDeviceDA, ReleaseDeviceDA, ReleaseDevice

♦软件产生触发事件

函数原型：

Visual C++：

```
BOOL SetDeviceTrigDA ( HANDLE hDevice ,
```

```
                    BOOL bSetSyncTrig,
                    int nDAChannel)
```

Visual Basic:

```
Declare Function SetDeviceTrigDA Lib "PCI9103_32" (ByVal hDevice As Long,_
                                                   ByVal bSetSyncTrig As Boolean,_
                                                   ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

bSetSyncTrig 是否置同步触发。

nDAChannel 通道号，取值范围为[0, 3]。

返回值：如果调用成功，则返回 TRUE，则产生一个软件触发事件，在某些触发模式下会直接使 DA 输出波形， 否则返回 FALSE。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA, WriteDeviceBulkDA, ReadDeviceBulkDA, EnableDeviceDA, SetDeviceTrigDA, GetDevStatusDA, DisableDeviceDA, ReleaseDeviceDA, ReleaseDevice

♦ **取得 DA 的状态标志**

函数原型：

Visual C++:

```
BOOL GetDevStatusDA (HANDLE hDevice,
                     PPCI9103_STATUS_DA pDAStatus,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PCI9103_32" (ByVal hDevice As Long,_
                                                  ByRef pDAStatus As PCI9103_STATUS_DA,_
                                                  ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

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

nDAChannel 通道号，取值范围为[0, 3]。

返回值：若 DA 成功取回标状态，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetDevTrigLevelDA, SetDevFrequencyDA, ReadSegmentInfo, InitDeviceDA, WriteDeviceOneDA

WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA ReleaseDevice

♦ **暂停 DA 设备**

函数原型

Visual C++：

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

Visual Basic:

```
Declare Function DisableDeviceDA Lib "PCI9103_32" (ByVal hDevice as Long,_
                                                   ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice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。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA ReleaseDevice

♦ **释放 DA 设备**

函数原型

Visual C++：

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

Visual Basic:

```
Declare Function ReleaseDeviceDA Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                   ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice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。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA

SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA ReleaseDevice

◆ **启动 DA 校准**

函数原型

Visual C++:

```
BOOL StartCalibration ( HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartCalibration Lib "PCI9103_32" (ByVal hDevice as Long
                                                    ) As Boolean
```

LabView:

请参考相关演示程序。

功能：启动 DA 设备。

hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA ReleaseDevice

◆ **设备 DA 校准**

函数原型

Visual C++:

```
BOOL GetDACalibration ( HANDLE hDevice,
                        LONG OutputRange,
                        LONG CalMode,
                        PLONG pCalData,
                        int nDAChannel);
```

Visual Basic:

```
Declare Function GetDACalibration Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                    ByRef OutputRange As Integer, _
                                                    ByVal CalMode As Long, _
                                                    ByRef pCalData As Long, _
                                                     ByVal nDAChannel As Integer) As Boolean
```

LabView:

请参考相关演示程序。

功能：设备 DA 校准

参数：OutputRange,输出量程,分别控制两个通道
CalMode，0 为零点校准，1 为满度校准
nDAChannel，DA 输出通道,取值范围为：0-1
hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA

ReleaseDeviceDA ReleaseDevice

- **设备 DA 校准**

函数原型

Visual C++：

```
BOOL SetDACalibration ( HANDLE hDevice,
                        LONG OutputRange,
                        LONG CalMode,
                        LONG CalData,
                        int nDAChannel);
```

Visual Basic:

```
Declare Function SetDACalibration Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                    ByRef OutputRange As Integer, _
                                                    ByVal CalMode As Long, _
                                                    ByRef CalData As Long, _
                                                     ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：设备 DA 校准

参数：OutputRange,输出量程,分别控制两个通道

CalMode，0 为零点校准，1 为满度校准

nDAChannel，DA 输出通道,取值范围为：0-1

hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA ReleaseDevice

- **停止 DA 校准**

函数原型

Visual C++：

```
BOOL StopCalibration ( HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopCalibration Lib "PCI9103_32" (ByVal hDevice as Long, _ ) As Boolean
```

LabView：

请参考相关演示程序。

功能：停止 DA 校准。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE。

相关函数： CreateDevice SetDevTrigLevelDA SetDevFrequencyDA
ReadSegmentInfo InitDeviceDA WriteDeviceOneDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA
SetDeviceTrigDA GetDevStatusDA DisableDeviceDA

ReleaseDeviceDA　　ReleaseDevice

♦ **采样和和传输函数一般调用顺序**

①CreateDevice (创建设备对象)
②InitDeviceDA (初始化设备)
③ WriteDeviceBulkDA(批量写入DA数据到板载RAM)
④EnableDeviceDA (启动DA设备)
⑤SetDevTrigLevelDA (若为软件触发源，则置软件触发事件)
⑥ GetDevStatusDA（循环查询DA状态）
⑦ DisableDeviceDA
⑧ ReleaseDevice
关于调用过程的图形说明请参考《绪论》。

第四节、DA 硬件参数系统保存与读取函数原型说明

♦ **写设备硬件参数函数到 Windows 系统中**

函数原型：

Viusal C++:

```
BOOL SaveParaDA (HANDLE hDevice,
                 PPCI9103_PARA_DA pDAPara,
                 int nDAChannel)
```

Visual Basic:

```
Declare Function SaveParaDA Lib "PCI9103_32" (ByVal hDevice As Long, _
                                              ByRef pDAPara As PCI9103_PARA_DA,_
                                              ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。
参数：
hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
pDAPara设备硬件参数，关于PPCI9103_PARA_DA的详细介绍请参考PCI9103.h或PCI9103.Bas或PCI9103.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。
nDAChannel　DA 通道号，取值范围为[0, 3]。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice　　LoadParaDA　　SaveParaDA
ReleaseDevice

♦ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaDA(HANDLE hDevice,
                PPCI9103_PARA_DA pDAPara,
                int nDAChannel)
```

Visual Basic:

```
Declare Function LoadParaDA Lib "PCI9103_32" (ByVal hDevice As Long, _
                                              ByRef pDAPara As PCI9103_PARA_DA, _
```

ByVal nDAChannel As Integer) As Boolean

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pDAPara属于PPCI9103_PARA_DA的结构指针类型，它负责返回PCI硬件参数值，关于结构指针类型PPCI9103_PARA_DA请参考PCI9103.h或PCI9103.Bas或PCI9103.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 3]。

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：

Viusal C++ :

```
BOOL ResetParaDA (HANDLE hDevice,
                  PPCI9103_PARA_DA pDAPara,
                  int nDAChannel)
```

Visual Basic:

```
Declare Function ResetParaDA Lib "PCI9103_32" (ByVal hDevice As Long, _
                                               ByRef pDAPara As PCI9103_PARA_DA,_
                                               ByVal nDAChannel As Integer) As Boolean
```

LabVIEW:

请参考相关演示程序。

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

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pDAPara设备硬件参数，关于PPCI9103_PARA_DA的详细介绍请参考PCI9103.h或PCI9103.Bas或PCI9103.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。调用此函数后，该参数指向的结构体成员将被复位至默认值。

nDAChannel DA 通道号，取值范围为[0, 3]。

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

相关函数： CreateDevice LoadParaDA SaveParaDA ResetParaDA ReleaseDevice

第五节、DIO 数字量输入输出开关量操作函数原型说明

◆ **开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   BYTE bDISts[4])
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI9103_32" (ByVal hDevice As Long, _
                                               ByVal bDISts(0 to 3) As Byte) As Boolean
```

LabVIEW：

请参考相关演示程序。

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

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

bDISts 四路开关量输入状态的参数结构，共有 4 个元素，分别对应于 DI0-DI3 路开关量输入状态位。如果 bDISts[0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

返回值：若成功，返回 TRUE，其 bDISts[x]中的值有效；否则返回 FALSE，其 bDISts[x]中的值无效。

相关函数: CreateDevice SetDeviceDO ReleaseDevice

◆ **开关量输出**

函数原型：

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[4])
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI9103_32" (ByVal hDevice As Long, _
                                               ByVal bDOSts(0 to 3) As Byte) As Boolean
```

LabVIEW

请参考相关演示程序。

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

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

bDOSts 四路开关量输出状态的参数结构，共有 4 个元素，分别对应于 DO0-DO3 路开关量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice GetDeviceDI ReleaseDevice

第四章 硬件参数结构

第一节、DA 各段信息参数结构（PCI9103_SEGMENT_INFO）

Visual C++：

```
typedef struct _PCI9103_SEGMENT_INFO
{
    LONG SegLoopCount;      // 每个段在大循环中的小循环次数，取值为[1, 16777215]
    LONG SegmentSize;       // 每个段在 RAM 中的长度(单位：字/点)
```

```
} PCI9103_SEGMENT_INFO, *PPCI9103_SEGMENT_INFO;
```

Visual Basic：

```
Type PCI9103_SEGMENT_INFO
    SegLoopCount As Long        ' 每个段在大循环中的小循环次数，取值为[1, 16777215]
    SegmentSize As Long         ' 每个段在 RAM 中的长度(单位：字/点)
End Type
```

LabVIEW:

请参考相关演示程序。

此参数结构主要用于建立段信息，它包括段长、段循环等，它主要用于函数 InitDeviceDA（）的SegmentInfo[]参数。

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

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

第二节、DA 硬件参数结构（PCI9103_PARA_DA）

Visual C++：

```
typedef struct _PCI9103_PARA_DA
{
    LONG OutputRange;           // 输出量程
    LONG Frequency;             // 采集频率[1Hz, 1.7MHz]，  为正数时单位 Hz
    LONG LoopCount;             // 整过 RAM 的大循环次数，=0:无限循环，  =n:表示 n 次循环(1<n<32768)
    LONG TriggerMode;           // 触发模式选择
    LONG TriggerSource;         // 触发源选择
    LONG TriggerDir;            // 触发方向选择
    LONG bSingleOut;            // 是否单点输出
    LONG ClockSource;           // 时钟源选择
} PCI9103_PARA_DA, *PPCI9103_PARA_DA;
```

Visual Basic：

```
Type PCI9103_PARA_DA
    OutputRange As Long         ' 输出量程
    Frequency As Long           ' 采集频率[1Hz, 1.7MHz]，  为正数时单位 Hz
    LoopCount As Long           ' 整过 RAM 的大循环次数，=0:无限循环，  =n:表示 n 次循环(1<n<32768)
    TriggerMode As Long         ' 触发模式选择
    TriggerSource As Long       ' 触发源选择
    TriggerDir As Long          ' 触发源选择
    bSingleOut As Long          ' 是否单点输出
    ClockSource As Long         ' 时钟源选择
End Type
```

LabVIEW:

请参考相关演示程序。

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

OutputRange 输出信号的量程范围，取值如下表：

常量名	常量值	功能定义
PCI9103_OUTPUT_0_P5000mV	0x00	0～5000mV
PCI9103_OUTPUT_0_P10000mV	0x01	0～10000mV
PCI9103_OUTPUT_N5000_P5000mV	0x02	±5000mV
PCI9103_OUTPUT_N2500_P2500mV	0x03	±2500mV
PCI9103_OUTPUT_N10000_P10000mV	0x04	±10000mV

关于电压值到设备原始码之间的换算公式请参考《DA的电压值如何转换成输出到DA转换器的LSB原码数据》。

Frequency DA 输出时的点频率，即刷新频率，单位为 Hz，如为 120 时表示其点频为 120Hz。注意实际输出的信号频率是由该点频率与每周期 DA 数据点数共同作用的结果。如点频率是 100KHz，每周期的正弦波数据点数是 256 个点，则输出正弦波信号的频率为 0.390625KHz(即 100 / 256 所得)。

LoopCount 总循环次数，=0：无限循环, =n：表示 n 次循环(1<n<32768)。该参数只有触发模式为连续触发时有效。它表示指定通道指定有效段输出的次数。比如有效段数 SegmentCount 为 3 时，该参数为 4 时，则实际输出的段序列为：

循环第 1 次			循环第 2 次			循环第 3 次			循环第 4 次		
段 0	段 1	段 2	段 0	段 1	段 2	段 0	段 1	段 2	段 0	段 1	段 2

TriggerMode DA 触发模式选择。选项值如下表：

常量名	常量值	功能定义
PCI9103_TRIGMODE_SINGLE	0x00	单次触发
PCI9103_TRIGMODE_CONTINUOUS	0x01	连续触发
PCI9103_TRIGMODE_STEPED	0x02	单步触发
PCI9103_TRIGMODE_BURST	0x03	紧急触发

TriggerSource DA 转换触发源。选项值如下表：

常量名	常量值	功能定义
PCI9103_TRIGSRC_SOFT_DA	0x00	软件触发
PCI9103_TRIGSRC_ATR_DA	0x01	ATR 硬件模拟触发
PCI9103_TRIGSRC_DTR_DA	0x02	DTR 硬件数字触发

TriggerDir DA 外触发方式使用信号方向，只对硬件模拟 ATR 触发源和硬件 DTR 数字触发源有效。选项值如下表：

常量名	常量值	功能定义
PCI9103_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
PCI9103_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
PCI9103_TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低脉冲或上升/下降沿触发)

当 TriggerDir = PCI9103_TRIGDIR_NEGATIVE 时，表示外部触发信号须由大于 TrigLevelVolt 变成小于 TrigLevelVolt 时产生触发事件（即下降沿触发）。

当 TriggerDir = PCI9103_TRIGDIR_POSITIVE 时，表示外部触发信号由小于 TrigLevelVolt 变成大于 TrigLevelVolt 时产生触发事件 (即上升沿触发)。

当 TriggerDir = PCI9103_TRIGDIR_POSIT_NEGAT 时，凡外部触发信号发生以上两种情况中的任意一种则产生触发事件。

bSingleOut 是否单点输出。

ClockSource DA 外时钟选择，选项值如下表：

常量名	常量值	功能定义
PCI9103_CLOCKSRC_IN	0x00	内部时钟
PCI9103_CLOCKSRC_OUT	0x01	外部时钟

第三节、DA 状态参数结构（PCI9103_STATUS_DA）

Visual C++：

```
typedef struct _PCI9103_STATUS_DA
{
    LONG bEnable;          // DA 使能启动标志，＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止
    LONG bTrigFlag;        // 触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    LONG bConverting;      // DA 是否正在转换， =TRUE:表示正在转换， = FALS 表示转换完成
    LONG nCurSegNum;       // 可读取的 RAM 段号，取值为[0, SegmentCount-1], (注 SegmentCount 为 InitDeviceDA 函数的参数)
    LONG nCurSegAddr;      // 可读取的 RAM 段地址
    LONG nCurLoopCount; // 当前总循环次数
    LONG nCurSegLoopCount; // 当前段循环次数
} PCI9103_STATUS_DA, *PPCI9103_STATUS_DA;
```

Visual Basic：

```
Type PCI9103_ STATUS _DA
    bEnable As Long          ' DA 使能启动标志，＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止
    bTrigFlag As Long            '触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    bConverting As Long          ' DA 是否正在转换， =TRUE:表示正在转换， = FALS 表示转换完成
    nCurSegNum As Long           '可读取的 RAM 段号
    nCurSegAddr As Long          ' 可读取的 RAM 段地址
    nCurLoopCount As Long                ' 当前总循环次数
    nCurSegLoopCount As Long             ' 当前段循环次数
End Type
```

LabVIEW:

请参考相关演示程序。

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

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

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

nCurSegNum 可读取的 RAM 段号。

nCurSegAddr 可读取的 RAM 段地址。

nCurLoopCount 当前总循环次数。

nCurSegLoopCount 当前段循环次数。

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 / 4096)	[0，4095]
0～10000mV	Lsb = Volt / (10000.00 / 4096)	[0，4095]
±2500mV	Lsb = Volt / (5000.00 / 4096) + 2048	[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 WriteDeviceBulkDA中的DABuffer[]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 PCI9103_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　触发功能详述

第一节、段结构和排列序列

要讨论触发功能，首先要关心输出的波形数据在板载 RAM 中的排列规则和段结构。总体上整个输出的波形序列通常是由若干段组成的，各段以若干循环输出形成的。而不同的触发模式就是对段序列进行不同模式的控制，以实现某些特定的功能。

一、波形长度

要确保每一段下载到RAM中的数据的长度至少要等于波形实际采样点数，而且正好是波形周期的整数倍长，否则可能会出现波形在循环输出过程中出现不连续的现象，即断波。而正常的段间循环和连续需要总段数、各段起始地址、各段长度、各段循环次数的控制。

二、波形的顺序

多个波形和波形的顺序都将被下载到信号发生器的板载 RAM 中。循环波形就是重复单个波形段若干次。波形连续就是将若干个段循环输出后连起来形成一个更复杂的信号。通过循环和连接可以极大的扩展板载 RAM 的空间和增加波形输出的时间。

下面的图形展示了波形和段顺序的一些概念。

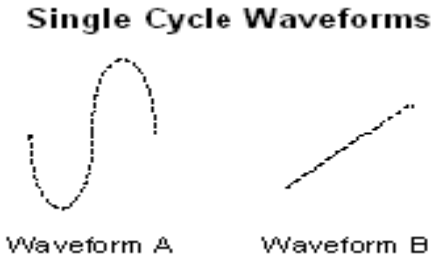


图 6.1.1 单周期波形

图 6.1.1 中波形 A 表现的是一个下载到板 RAM 中的单周期正弦波，波形 B 表现的是一个下载到板载 RAM 中的单周期斜波。通常一个周期的波形数据即可构成一个段(Segment)，段的循环就是对一个单周期波形的循环。可见通过段的循环可以将一个相对较短的单周期的波形数据实现 N 倍长的波形输出，相当于将板载 RAM 空间扩展了 N 倍。当然这里的 N 指的是段循环次数或总循环次数。犹如下面：

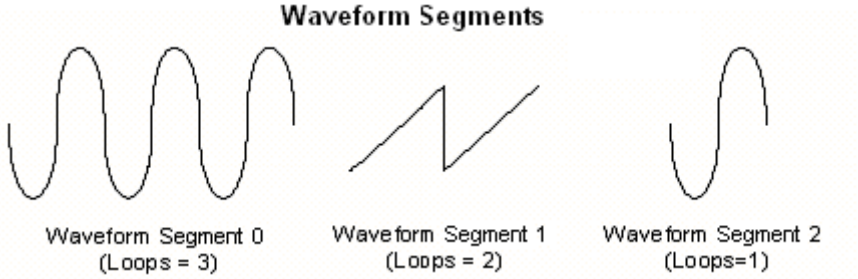


图 6.1.2

图 6.1.2 中的段 0 则是由一个周期的波形数据 A（正弦波）循环三次形成的。段 1 则是由一个周期的波形数据 B（斜波）循环两次形成的，而段 2 则是由一个周期的波形数据 A（正弦波）循环一次形成的。

那么按以上顺序和段循环次数最后连续起来的波形如图 6.1.3：

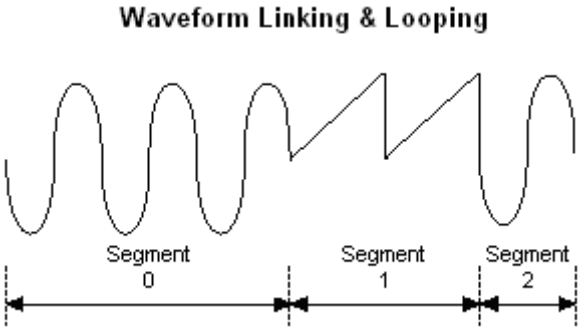


图 6. 1.3

第二节、触发功能

一、触发源(TriggerSource)

触发源包括软件触发、ATR 模拟量触发、DTR 数字量触发，可以通过软件选择。它的作用是决定由什么信号产生触发事件。注意软件触发源是由软件指令实现，即调用一次 SetDeviceTirgDA()即产生一次触发事件。而 ATR 和 DTR 需要由板外输入硬件信号。

二、触发方向(TriggerDir)

触发方向包括负向触发、正向触发、正负向触发。具体定义：

负向触发指的是触发源信号产生一个由高到低的瞬间跳变信号（即下边沿信号）时产生触发事件。比如触发源选择 ATR 时表示 ATR 输入信号由大于触发电平状态瞬间变化到小于触发电平状态时产生触发事件。对于 DTR 则是直接产生一个下边沿即可产生触发事件。而软件触发源则不必关心触发方向。

正向触发指的是触发源信号产生一个由低到高的瞬间跳变信号（即上边沿信号）时产生触发事件。比如触发源选择 ATR 时表示 ATR 输入信号由小于触发电平状态瞬间变化到大于触发电平状态时产生触发事件。对于 DTR 则是直接产生一个上边沿即可产生触发事件。而软件触发源则不必关心触发方向。

正负向触发指的是一旦发生以上两种情况中的一种即可产生触发事件。

三、触发模式（TriggerMode）

触发模式包括单次触发(Single Trigger Mode)、连续触发（Continuous Trigger Mode）、单步触发（Stepped Trigger Mode）、紧急触发(Burst Trigger Mode)，具体定义：

1、单次触发(Single)

总体上是指定的段序列被依次循环输出一次。具体是当调用 EnableDeviceDA ()时DA不输出波形，只保持在原始状态下，只有当产生一个触发事件时设备才开始从段 0 输出波形。当段 0 循环指定次数后，自动转入下一段继续循环输出，待循环结束后再继续下一段。当输出到最后一段时，则最后一段自动进入无限循环，直到用户强行停止，停止后保持在被停止时的当前数据点状态下。如图 6.2.1，使用段 0、1、2 去创建单次触发的任意波形。当信号发生器收到一个触发事件，则开始产生段 0 数据波形，待段 0 数据按照指定次数循环结束后，紧接着循环段 1，以至到段 2。此时段 2 总是作无穷循环直到您停止它。因此，其总循环次数和最后段的循环次数不再有效。一旦触发后，不再接受新的触发事件。

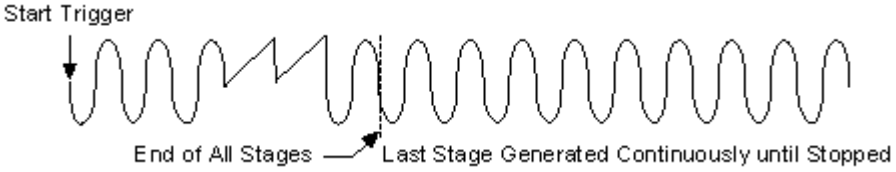


图 6.2.1

2、连续触发(Continuous)

总体上是指定的段序列被依次整个有限或无限循环输出。具体是当调用 EnableDeviceDA ()时DA不输出波形，只保持在原始状态下，只有当产生一个触发事件时设备才开始从段 0 输出波形。当段 0 循环指定次数后，自动转

入下一段继续循环输出，待循环结束后再继续下一段。当输出到最后一段时，则最后一段不像单次触发那样，而是循环指定次数自动转入段 0 继续循环，当再次循环完最后段时再回到段 0。若总循环次数设为 0 时，则表示总循环为无穷循环，一旦到最后段结束，便总是回到段 0 继续输出，直到用户强行停止。若总循环次数（LoopCount）不等于 0，则表示总循环为有限循环，即从最后段循环完后回到段 0 的次数。当回到段 0 的次数达到LoopCount 指定的值，则输出到最后段时会自动停止。停止后的状态保持在停止当时的点数据状态下。如图 6.2.2，使用段 0、1、2 去创建连续触发的任意波形。当信号发生器收到一个触发事件，则开始产生段 0 数据波形，待段 0 数据按照指定次数循环结束后，紧接着循环段 1，以至到最后段 2。待段 2 结束后自动回到段 0。因此，其总循环次数和段循环次数均有效。一旦触发后，不再接受新的触发事件。

当收到触发事件后，发生器从段 0 开始循环，直至最后一段，然后回到段 0，继续输出，除非用户停止，否则一直循环到指定的总循环次数。在输出过程中，不再接受新的触发事件。当一旦触发时，置触发标志为 1，而触发之前则为 0。且用户停止输出时自动复位为 0。

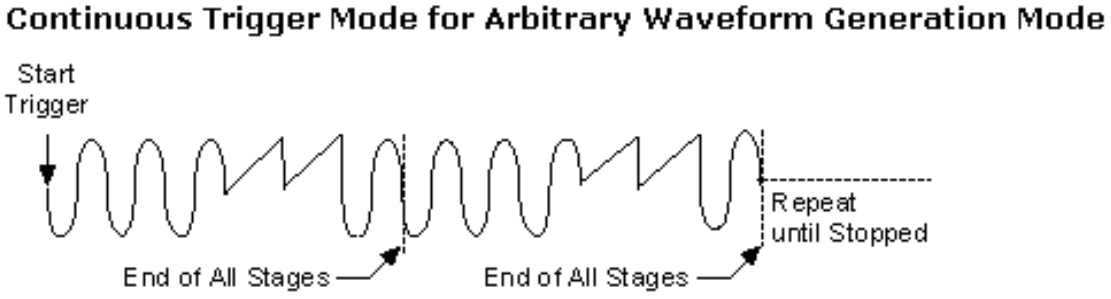


图 6.2.2

3、单步触发(Stepped)

总体上是指定的段序列中每个段均要接受一个独立触发事件才被依次有限循环输出。具体是当调用 EnableDeviceDA ()时DA不输出波形，只保持在原始状态下，只有当产生第一个触发事件时设备才开始从段 0 输出波形，待指定次数循环完后，自动停止，且保持在该段最后一个点状态下。此时若再产生一个触发事件，则自动转入下一段继续循环输出，待循环结束后又自动停止，且保持在该段最后一个点状态下，直到最后一个结束后又再产生触发事件，则又自动回到段 0。注意在各个段循环输出过程中新的触发事件无效，只有在某个段循环结束处于最后点保持状态下时产生的触发事件才有效。如图 6.2.3，使用段 0、1、2 去创建单步触发的任意波形。当信号发生器收到一个触发事件，则开始产生段 0 数据波形，待段 0 数据按照指定次数循环结束后，自动停止，且保持在最一个点状态下，当再来一个触发事件时，则自动循环输出段 1，以至到最后段 2。待段 2 结束后自动再来新的触发则又回到段 0。因此，只有段循环次数有效。

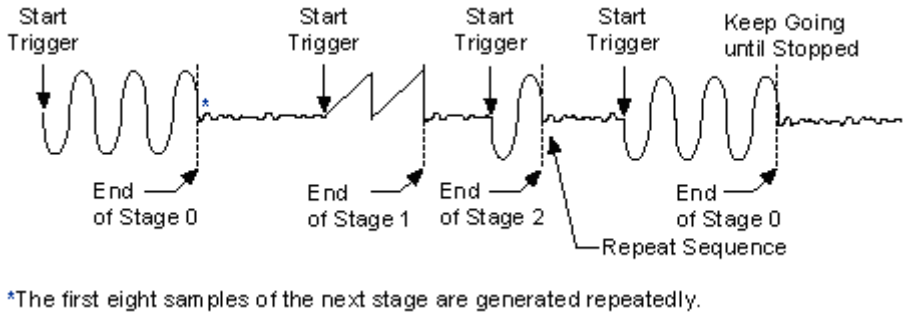


图 6.2.3

4、紧急触发(Burst)

总体上是指定的段序列中每个段均要接受一个独立触发事件才被依次无限循环输出。具体是当调用 EnableDeviceDA ()时DA不输出波形，只保持在原始状态下，只有当产生第一个触发事件时设备才开始从段 0 输出波形，此时段 0 作无限循环输出，此时若再产生一个触发事件，则自动转入下一段继续作无限循环输出，直至

到最后段作无限循环输出，若再来一个触发事件，则又自动回到段 0 继续作无限循环输出。注意在各个段循环输出过程中新的触发事件总是立即有效。待用户停止后保持在最后一个点上。如图 6.2.4，使用段 0、1、2 去创建紧急触发的任意波形。当信号发生器收到一个触发事件，则开始产生段 0 数据波形，此时段 0 则作无限循环输出，若此时再来一个触发事件，则自动开始段 1 的无限循环输出，当再来触发事件，则自动开始段 2 的无限循环输出，当再来触发事件时则又回到段 0。因此，总循环次数和段循环次数均无效。

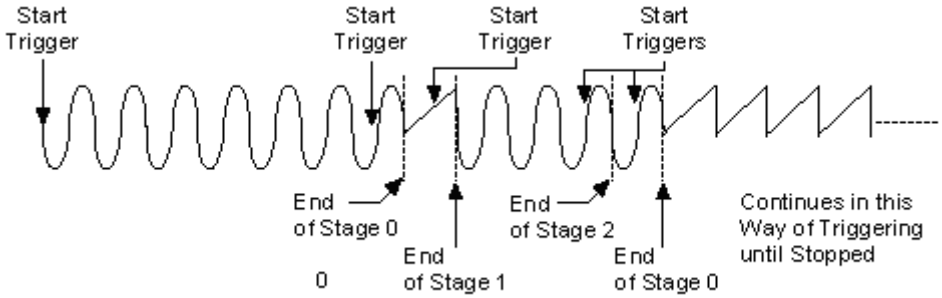


图 6.2.4

注：所谓原始状态即指上电最初状态或输出停止后保持的状态也可指软件复位后的状态。

第七章　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 WriteDeviceBulkDA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 PCI9103.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI9103 DA DIO 卡]**丨**[Microsoft VS2005 C++]**丨**[简易代码演示]**丨**[DIO…]**

其简易程序默认存放路径为：系统盘\ART\PCI9103\SAMPLES\VC\SIMPLE\DA\BULK

其他语言的演示可以用上面类似的方法找到。

二、怎样使用 GetDeviceDI函数进行更便捷的数字开关量输入操作

Visual C++:

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

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI9103 DA DIO 卡]**丨**[Microsoft VS2005 C++]**丨**[简易代码演示]**丨**[DIO…]**

三、怎样使用 SetDeviceDO函数进行更便捷的数字开关量输出操作

Visual C++:

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

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI9103 DA DIO 卡]**丨**[Microsoft VS2005 C++]**丨**[简易代码演示]**丨**[DIO…]**

第二节、高级程序演示说明

高级程序演示了本设备的所有功能，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PCI8100.h 和 DADoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI9103 DA DIO 卡]**丨**[Microsoft VS2005 C++]**丨**[高级代码演示]**

其默认存放路径为：系统盘\ART\PCI9103\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第八章　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“PCI9103_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 线程操作函数		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

第二节、PCI 内存映射寄存器操作函数原型说明

- **取得指定的指定设备寄存器组 BAR 地址**

函数原型:

Visual C++:

```
BOOL GetDeviceBar ( HANDLE hDevice,
                    __int64 pbPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib "PCI9103_32" (ByVal hDevice As Long, _
```

ByVal pbPCIBar (0 to 5) As Long) As Boolean

LabVIEW:

请参考相关演示程序。

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

pbPCIBar 返回 PCI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

- **以单字节（即 8 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型:

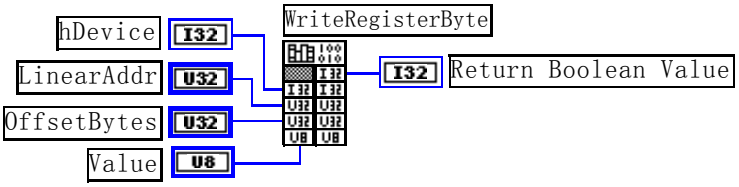
Visual C++:

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        __int64 pbLinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                     ByVal pbLinearAddr As Long, _
                                                     ByVal OffsetBytes As Long, _
                                                     ByVal Value As Byte ) As Boolean
```

LabVIEW:



功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice决定。

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

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

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

Visual C++程序举例:

：

```
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);  // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

♦ **以双字节（即 16 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型:

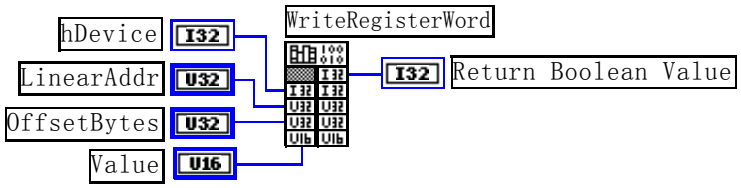
Visual C++:

```
BOOL WriteRegisterWord(HANDLE hDevice,
                       __int64 pbLinearAddr,
                       ULONG OffsetBytes,
                       WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI9103_32" ( ByVal hDevice as Long, _
                                                      ByVal pbLinearAddr As Long, _
                                                      ByVal OffsetBytes As Long, _
                                                      ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice确定。

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

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

Value 输出16位整型值。

返回值：无。

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

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
   AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);  // 往指定映射寄存器单元写入 16 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

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

♦ **以四字节（即32位）方式写PCI内存映射寄存器的某个单元**

函数原型:

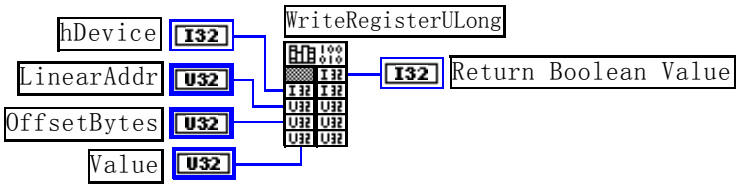
Visual C++:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         __int64 pbLinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

Visual Basic:

```
Declare Sub WriteRegisterULong Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                 ByVal pbLinearAddrAs Long, _
                                                 ByVal OffsetBytes As Long, _
                                                 ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice决定。

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

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

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);  // 往指定映射寄存器单元写入 32 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

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

◆ **以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

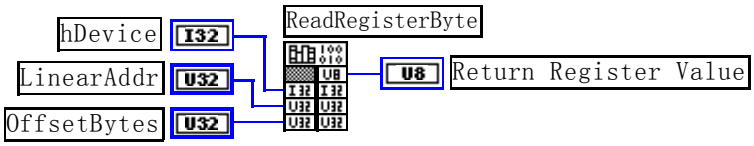
Visual C++:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       __int64 pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI9103_32" ( ByVal hDevice as Long, _
                                                     ByVal pbLinearAddr As Long, _
                                                     ByVal OffsetBytes As Long ) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice决定。

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

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数 。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

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

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
BYTE   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :

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

♦ **以双字节（即 16 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

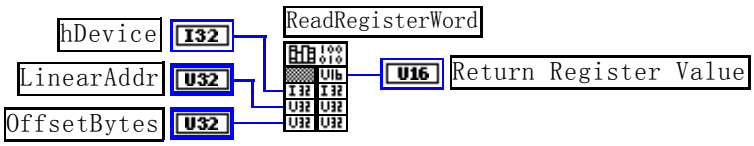
Visual C++:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       __int64 pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI9103_32" (   ByVal hDevice as Long, _
                                                       ByVal pbLinearAddr As Long, _
                                                       ByVal OffsetBytes As Long) As Integer
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice决定。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

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

Visual C++程序举例:

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

Visual Basic 程序举例:

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

```
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

◆ **以四字节（即 32 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

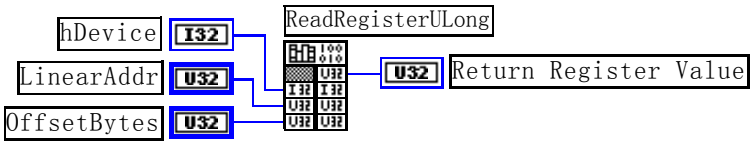
Visual C++:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         __int64 pbLinearAddr,
                         ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI9103_32" (ByVal hDevice as Long, _
                                                     ByVal pbLinearAddr As Long, _
                                                     ByVal OffsetBytes As Long) As Long
```

LabVIEW:



功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice。

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 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	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++程序举例:

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

Visual Basic 程序举例:

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

第三节、I0 端口读写函数

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

- **以单字节(8Bit)方式写 I/O 端口**

函数原型:

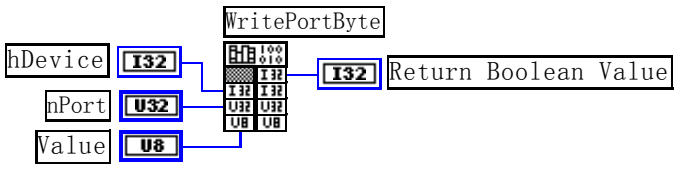
Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    __int64 pPort,
                    BYTE Value)
```

Visual Basic:

```
Declare Function WritePortByte Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                 ByVal pPort As Long, _
                                                 ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pPort 设备的 I/O 端口号。

Value 写入由 pPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

函数原型:

Visual C++:

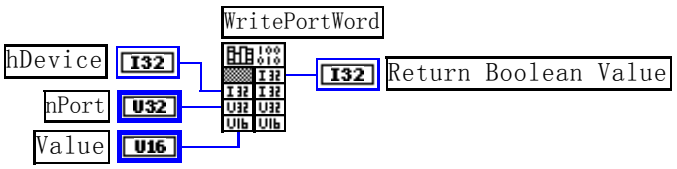
```
BOOL WritePortWord (HANDLE hDevice,
                    __int64 pPort,
```

WORD Value)

Visual Basic:

```
Declare Function WritePortWord Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                 ByVal pPort As Long, _
                                                 ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

pPort　设备的 I/O 端口号。

Value　写入由 pPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

函数原型:

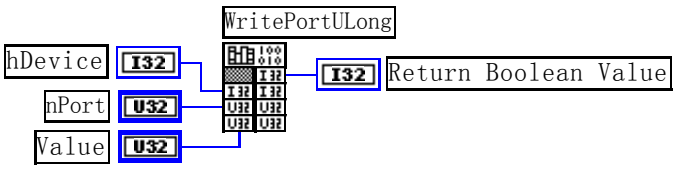
Visual C++:

```
BOOL WritePortULong(HANDLE hDevice,
                    __int64 pPort,
                    ULONG Value)
```

Visual Basic:

```
Declare Function WritePortULong Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                  ByVal pPort As Long, _
                                                  ByVal Value As Long ) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pPort　设备的 I/O 端口号。

Value　写入由 pPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

函数原型:

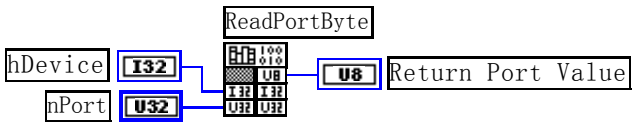
Visual C++:

```
BYTE ReadPortByte(HANDLE hDevice,
                  __int64 pPort)
```

Visual Basic:

```
Declare Function ReadPortByte Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                ByVal pPort As Long ) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice。

pPort 设备的 I/O 端口号。

返回值：返回由 pPort 指定的端口的值。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

◆ **以双字节(16Bit)方式读 I/O 端口**

函数原型:

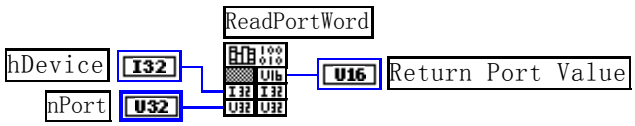
Visual C++:

```
WORD ReadPortWord(HANDLE hDevice,
                  __int64 pPort)
```

Visual Basic:

```
Declare Function ReadPortWord Lib "PCI9103_32" (ByVal hDevice As Long, _
                                                ByVal pPort As Long ) As Integer
```

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pPort 设备的 I/O 端口号。

返回值：返回由 pPort 指定的端口的值。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

◆ **以四字节(32Bit)方式读 I/O 端口**

函数原型:

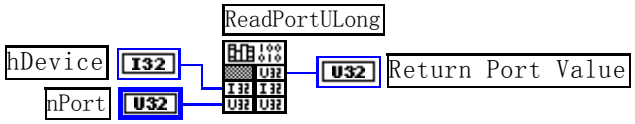
Visual C++:

```
ULONG ReadPortULong(HANDLE hDevice,
                    __int64 pPort)
```

Visual Basic:

Declare Function ReadPortULong Lib "PCI9103_32" (ByVal hDevice As Long, _
ByVal pPort As Long) As Long

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

pPort　设备的 I/O 端口号。

返回值：返回由 pPort 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

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

- **创建内核系统事件**

函数原型:

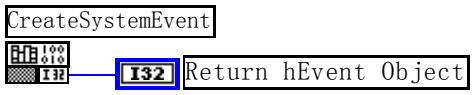
Visual C++：

HANDLE CreateSystemEvent(void);

Visual Basic:

Declare Function CreateSystemEvent Lib " PCI9103_32 " () As Long

LabVIEW:



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

- **释放内核系统事件**

函数原型:

Visual C++ :

BOOL ReleaseSystemEvent(HANDLE hEvent);

Visual Basic:

Declare Function ReleaseSystemEvent Lib " PCI9103_32 " (ByVal hEvent As Long) As Boolean

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。