PXI9303 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 PXIxxxx_ 则被省略。如 PXI9303_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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			

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 PXI 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。当您需要暂停设备时，执行StopDeviceProDA，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

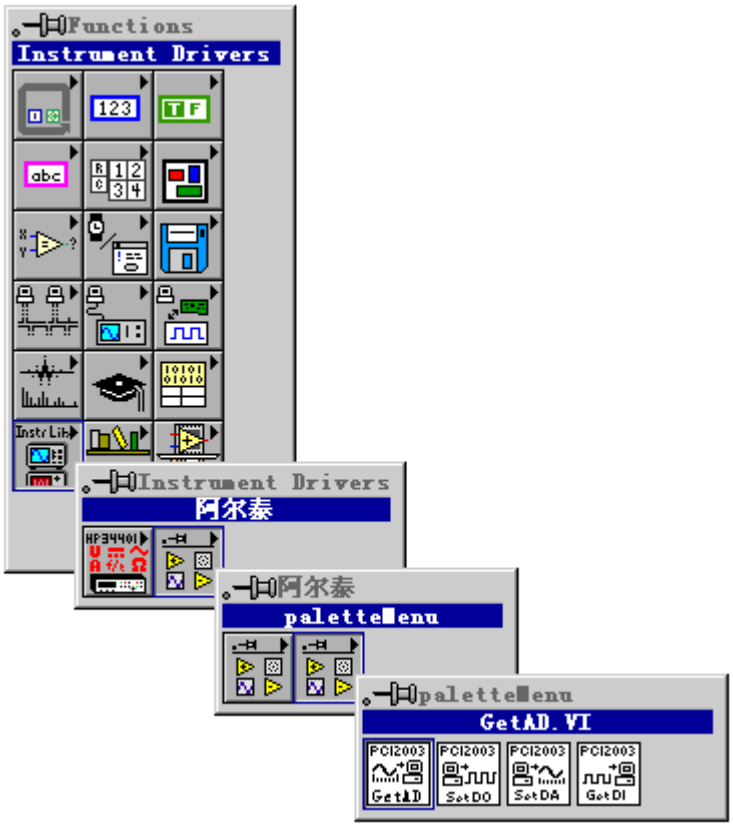
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord，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

第三章 PXI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的 Bit 位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉 PXI 总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解 PXI 的资源配置空间、PNP 即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用ReadRegisterULong和WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PXI9303_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PXI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PXI 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	上层及底层用户
CreateDeviceEx	用物理号创建设备对象	上层及底层用户
GetDeviceCurrentID	取得该设备当前逻辑 ID 和物理 ID	上层及底层用户
ReleaseDevice	关闭设备，且释放 PXI 总线设备对象	上层及底层用户
② 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	将注册表中的 DA 参数恢复至出厂默认值	上层用户
③ DA 数据读取函数		
StartDeviceProDA	启动某个 DA 通道开始 DA 转换	上层及底层用户
SetOutputRangeDA	设置 DA 的输出范围(只适用于 PXI9303)	上层及底层用户
InitDeviceProDA	初始化某个 DA 通道对象	上层及底层用户
StopDeviceProDA	暂停某个 DA 通道进行 DA 转换	上层及底层用户
GetDeviceStatusProDA	向 DA 的 FIFO 中写入批量数据(通常为 FIFO 的半满长度)	上层及底层用户
WriteDeviceProDA	向 DA 的 FIFO 中写入批量数据(通常为 FIFO 的半满长度)	上层及底层用户
WriteDeviceOneDA	向 DA 的 FIFO 中写入批量数据(通常为 FIFO 的半满长度)	上层及底层用户
④ DA 校准		
StartCalibration	启动 DA 校准	上层及底层用户

GetDACalibration	设备 DA 校准	上层及底层用户
SetDACalibration	设备 DA 校准	上层及底层用户
StopCalibration	停止 DA 校准	上层及底层用户
⑤ 计数器控制函数		
SelectFunction	功能选择函数	上层用户
EnableFunction	使能禁止 频率、周期、脉宽、计数器脉冲输出	上层用户
SetMeasureFre	设置设备测频模式	上层用户
GetMeasureFreSts	获取测频状态	上层用户
GetMeasureFre	获取被测频率值,返回被测频率值	上层用户
GetMeasureCycle	获取被测周期 uS	上层用户
GetPulseWidth	获取高低脉冲宽度	上层用户
SetPulseOutput	设置脉冲输出参数	上层用户

使用需知：

Visual C++：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\PXI\PXI9303\INCLUDE\PXI9303.H”

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PXI9303.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句：

#include “PXI9303.H”

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

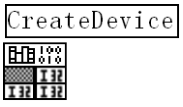
Visual Basic:

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 PXI9303.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：



一、在 LabView 中打开 PXI9303.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

二、根据 LabView 语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“［U16］”为无符号 16 位短整型数组或缓冲区或指针，“［U32］”与“[U16]”同理，只是位数不一样。

第二节、设备对象管理函数原型说明

♦ 创建设备对象函数（逻辑号）

函数原型：

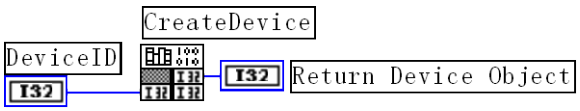
Visual C++：

HANDLE CreateDevice (int DeviceLgcID = 0)

Visual Basic:

Declare Function PXI9303_CreateDevice Lib "PXI9303_32" (ByVal DeviceLgcID As Long) As Long

LabVIEW：



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

参数：

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PXI 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 PXI9303 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 PXI9303 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 PXI 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。

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　ListDeviceDlgEx　ReleaseDevice

Visual C++程序举例

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

Visual Basic 程序举例

```
    :
    Dim hDevice As Long   ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    MsgBox “创建设备对象失败”
    Exit Sub     ' 退出该过程
End If
    :
```

♦ 取得本计算机系统中 PXI9303 设备的总数量

函数原型：

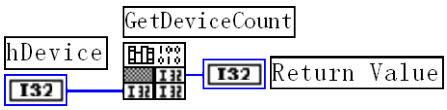
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function PXI9303_GetDeviceCount Lib "PXI9303_32" (ByVal hDevice As Long) As Long

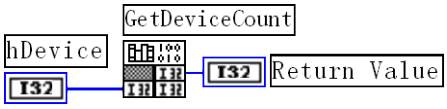
LabVIEW：



功能：取得 PXI9303 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 PXI9303 的数量。
相关函数：CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ListDeviceDlgEx　ReleaseDevice

- **用物理号创建设备对象**

函数原型：
Visual C++:
intCreateDeviceEx (int DevicePhysID = 0)
Visual Basic:
Declare Function PXI9303_CreateDeviceEx Lib "PXI9303_32" (ByVal DevicePhysID As Long) As Long
LabVIEW：



功能：取得 PXI9303 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 PXI9303 的数量。
相关函数：CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ListDeviceDlgEx　ReleaseDevice

- **用对话框控件列表计算机系统中所有 PXI9303 设备各种配置信息**

函数原型：
Visual C++:
BOOL ListDeviceDlg (HANDLE hDevice)
Visual Basic:
Declare Function PXI9303_ListDeviceDlg Lib "PXI9303_32" (ByVal hDevice As Long) As Boolean
LabVIEW：
请参考相关演示程序。

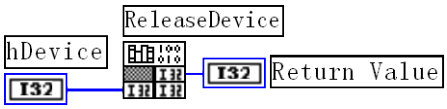
功能：列表系统中 PXI9303 的硬件配置信息。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则弹出对话框控件列表所有 PXI9303 设备的配置情况。
相关函数：CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ListDeviceDlgEx　ReleaseDevice

- **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 PXI9303_GetDeviceCurrentID Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef DeviceLgcID As Long,_
                                        ByRef DevicePhysID As Long) As Boolean
```
LabVIEW：
请参考相关演示程序。

功能：取得指定设备逻辑和物理 ID 号。
参数：
hDevice 设备对象句柄，它指向要取得逻辑和物理号的设备，它应由CreateDevice创建。
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 DID 决定。
DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。
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 GetDeviceCount GetDeviceCurrentID
ListDeviceDlg ListDeviceDlgEx 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 PXI9303_ReleaseDevice Lib "PXI9303_32" (ByVal hDevice As Long) As Boolean
LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则返回 TRUE， 否则返回 FALSE， 用户可以用GetLastErrorEx捕获错误码。
相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID
ListDeviceDlg ListDeviceDlgEx ReleaseDevice

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 数据读取函数原型说明

♦ **设置 DA 的输出范围(只适用于 PXI9303)**
函数原型：
Visual C++:
```
BOOL SetOutputRangeDA (HANDLE hDevice, _
                       LONG OutputRange
                       int nDAChannel )
```
Visual Basic:
```
Declare Function PXI9303_SetOutputRangeDA Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal OutputRange As Long,_
                                        ByVal nDAChannel As Long) As Boolean
```
LabVIEW：
请参考演示源程序。

功能：设置 DA 的输出范围(只适用于 PXI9303)。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
OutputRange 输出量程,分别控制两个通道
nDAChannel DA 输出通道,取值范围为：0-11
返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。
相关函数： StartDeviceProDA SetOutputRangeDA InitDeviceProDA

WriteDeviceOneDA　StopDeviceProDA　WriteDeviceProDA
GetDeviceStatusProDA

♦ **初始化设备对象**

函数原型：

Visual C++:

```
BOOL InitDeviceProDA (HANDLE hDevice,
                      PPXI9303_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function PXI9303_InitDeviceProDA Lib "PXI9303_32" (_
                                    ByVal hDevice As Long,_
                                    ByRef pDAPara As PXI9303_PARA_DA) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：初始化某个 DA 通道对象。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

PDAPara DA 输出的实际硬件参数

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 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartDeviceProDA　SetOutputRangeDA　InitDeviceProDA
WriteDeviceOneDA　StopDeviceProDA　WriteDeviceProDA
GetDeviceStatusProDA

♦ **启动某个 DA 通道开始 DA 转换**

函数原型：

Visual C++:

```
BOOL StartDeviceProDA (HANDLE hDevice)
```

Visual Basic:

```
Declare Function PXI9303_StartDeviceProDA Lib "PXI9303_32" (ByVal hDevice As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：启动某个 DA 通道开始 DA 转换。

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 否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartDeviceProDA　SetOutputRangeDA　InitDeviceProDA
WriteDeviceOneDA　StopDeviceProDA　WriteDeviceProDA
GetDeviceStatusProDA

♦ **暂停某个 DA 通道进行 DA 转换**

函数原型：

Visual C++:

```
BOOL StopDeviceProDA (HANDLE hDevice)
```

Visual Basic:

```
Declare Function PXI9303_StopDeviceProDA Lib "PXI9303_32" (ByVal hDevice As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：暂停某个 DA 通道进行 DA 转换。

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 否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartDeviceProDA SetOutputRangeDA InitDeviceProDA WriteDeviceOneDA StopDeviceProDA WriteDeviceProDA GetDeviceStatusProDA

- **在 AD 采样过程中取得 FIFO 的状态，通常用于半满方式读取**

函数原型：

Visual C++:

```
BOOL GetDeviceStatusProDA (HANDLE hDevice,
                           PPXI9303_STATUS_DA pDAStatus)
```

Visual Basic:

```
Declare Function PXI9303_GetDeviceStatusProDA Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByRef pDAStatus As PXI9303_STATUS_DA) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：在 AD 采样过程中取得 FIFO 的状态，通常用于半满方式读取。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pDAStatus 取得 FIFO 状态。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartDeviceProDA SetOutputRangeDA InitDeviceProDA WriteDeviceOneDA StopDeviceProDA WriteDeviceProDA GetDeviceStatusProDA

- **向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）**

函数原型：

Visual C++:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       PULONG pDABuffer,
                       ULONG nWriteSizeWords)
```

Visual Basic:

```
Declare Function PXI9303_WriteDeviceProDA Lib "PXI9303_32" (_
                                            ByVal hDevice As Long,_
                                            ByRef DABuffer As Long,_
                                            ByVal nWriteSizeWords As Long) As Boolean
```

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创建。

PDABuffer DA 数据用户缓冲区。

NWriteSizeWords 写入的点数（以字为单位）

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartDeviceProDA SetOutputRangeDA InitDeviceProDA WriteDeviceOneDA StopDeviceProDA WriteDeviceProDA GetDeviceStatusProDA

♦ **向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_WriteDeviceOneDA Lib "PXI9303_32" (_
                                            ByVal hDevice As Long,_
                                            ByRef DALSB As Long,_
                                            ByVal nDAChannel As Long) As Boolean
```

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创建。

DALSB DA 数据。

nDAChannelDA 输出通道,取值范围为：0-11

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartDeviceProDA　SetOutputRangeDA　InitDeviceProDA　WriteDeviceOneDA　StopDeviceProDA　WriteDeviceProDA　GetDeviceStatusProDA

第四节、DA 校准

♦**启动 DA 校准**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_StartCalibration Lib "PXI9303_32" (ByVal hDevice As Long) As Boolean
```

LabView：

请参考相关演示程序。

功能：启动 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，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartCalibration　GetDACalibration　SetDACalibration　StopCalibration

♦ **DA 设备校准**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_GetDACalibration Lib "PXI9303_32" (_
                                            ByVal hDevice As Long,_
                                            ByVal OutputRange As Boolean, _
                                            ByVal CalMode As Long,_
```

ByVal pCalData As Long,_
ByVal nDAChannel As Long) As Boolean

LabView：

请参考相关演示程序。

功能：设备 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

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

PCalData 校准值

nDAChannel 设备通道号，取值范围为[0, 11]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数: StartCalibration GetDACalibration
SetDACalibration StopCalibration

♦ **DA 设备校准**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_SetDACalibration Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByRef OutputRange As Long,_
                                                ByVal CalMode As Long,_
                                                ByVal CalData As Long,_
                                                ByVal nDAChannel As Long) As Boolean
```

LabView：

请参考相关演示程序。

功能：设备 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

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

PCalData 校准值

nDAChannel 设备通道号，取值范围为[0, 11]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartCalibration GetDACalibration
SetDACalibration StopCalibration

♦ **停止 DA 校准**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_StopCalibration Lib "PXI9303_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，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： StartCalibration GetDACalibration SetDACalibration StopCalibration

第五节、计数器控制函数

♦功能选择函数

函数原型：

Visual C++:

```
BOOL SelectFunction (HANDLE hDevice,
                     LONG lFunction,
                     ULONG nChannel)
```

Visual Basic:

```
Declare Function PXI9303_SelectFunction Lib "PXI9303_32" (_
                                          ByVal hDevice As Long,_
                                          ByRef lFunction As Long,_
                                          ByVal nChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：功能选择函数。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

lFunction 功能选择参数。

nChannel 计数器通道选择，取值为[0，1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

♦ 使能禁止 频率、周期、脉宽、计数器脉冲输出

函数原型：

Visual C++:

```
BOOL EnableFunction (HANDLE hDevice,
                     BOOL bEnableCNT,
                     ULONG nChannel)
```

Visual Basic:

```
Declare Function PXI9303_EnableFunction Lib "PXI9303_32" (_
                                          ByVal hDevice As Long,_
                                          ByRef bEnableCNT As Long,_
                                          ByVal nChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：使能禁止 频率、周期、脉宽、计数器脉冲输出。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

bEnableCNT 1:使能 0:禁止。

nChannel 计数器通道选择，取值为[0，1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

♦ **设置设备测频**

函数原型：

Visual C++:

```
BOOL SetMeasureFre (HANDLE hDevice,
                    ULONG ulDelayTime，
                    LONG lChannel)
```

Visual Basic:

```
Declare Function PXI9303_SetMeasureFre Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pCNTStatus As Long,_
                                        ByVal lChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：设置设备测频。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

ulDelayTime 确保测频准确度的定时高电平脉冲时间[10——2000ms]。

lChannel 计数器通道选择(0-1 通道)

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

♦ **获取测频状态**

函数原型：

Visual C++:

```
double GetMeasureFreSts (HANDLE hDevice,
                         PULONG pCNTStatus，
                         LONG lChannel)
```

Visual Basic:

```
Declare Function PXI9303_GetMeasureFreSts Lib "PXI9303_32" (_
                                           ByVal hDevice As Long,_
                                           ByRef ulDelayTime As Long,_
                                           ByVal lChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：获取测频状态。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pCNTStatus 测频状态：等于 0 表示测频结束，等于 1 表示正在测频计数。

lChannel 计数器通道选择(0-1 通道)

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

♦ **获取被测频率值,返回被测频率值**

函数原型：

Visual C++:

```
BOOL GetMeasureFre (HANDLE hDevice，
                    LONG lChannel)
```

Visual Basic:

```
Declare Function PXI9303_GetMeasureFre Lib "PXI9303_32" (_
                                        ByVal hDevice As Long, _
                                        ByVal lChannel As Long) As Long
```

LabVIEW:
请参考相关演示程序。

功能：获取被测频率值,返回被测频率值。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
lChannel 计数器通道选择(0-1 通道)
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

- **获取被测周期 uS**

函数原型：
Visual C++:
```
BOOL GetMeasureCycle (HANDLE hDevice，
                      LONG lChannel)
```
Visual Basic:

LabVIEW:
请参考相关演示程序。
```
Declare Function PXI9303_GetMeasureCycle Lib "PXI9303_32" (_
                                    ByVal hDevice As Long, _
                                    ByVal lChannel As Long) As Long
```
功能：获取被测周期 uS。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
lChannel 计数器通道选择(0-1 通道)
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

- **获取高低脉冲宽度**

函数原型：
Visual C++:
```
BOOL GetPulseWidth (HANDLE hDevice,
                    double* pPulseHigh，
                    double* pPulseLow，
                    LONG lChannel)
```
Visual Basic:
```
Declare Function PXI9303_GetPulseWidth Lib "PXI9303_32" (_
                                    ByVal hDevice As Long,_
                                    ByRef pPulseHigh As Double,_
                                    ByRef pPulseLow As Double, _
                                    ByVal lChannel As Long) As Long
```
LabVIEW:
请参考相关演示程序。

功能：获取高低脉冲宽度。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pPulseHigh 高脉冲宽度 nS
pPulseLow 低脉冲宽度 nS
lChannel) 测频通道选择(0-1 通道)
返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

♦ **设置脉冲输出参数**

函数原型：

Visual C++:

```
BOOL SetPulseOutput (HANDLE hDevice,
                     PPXI9303_PULSE_OUTPUT pPulsePara，
                     ULONG nChannel)
```

Visual Basic:

```
Declare Function PXI9303_SetPulseOutput Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal pPulsePara As PXI9303_PULSE_OUTPUT,_
                                        ByVal nChannel As Long) As Long
```

LabVIEW:

请参考相关演示程序。

功能：设置脉冲输出参数。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pPulsePara 脉冲输出参数

nChannel) 脉冲通道选择(0-1 通道)

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SelectFunction SetMeasureFre EnableFunction GetMeasureFreSts GetMeasureFre GetMeasureCycle GetPulseWidth SetPulseOutput ReleaseDevice

第六节、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,
                 PPXI9303_PARA_DA pDAPara）
```

Visual Basic:

```
Declare Function PXI9303_SaveParaDA Lib "PXI9303_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal pADPara As PXI9303_PARA_DA) As Long
```

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 PXI9303_PARA_DA 的详细介绍请参考 PXI9303.h 或 PXI9303.Bas 或 PXI9303.Pas 函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaDA SaveParaDA ReleaseDevice

♦ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaDA(HANDLE hDevice,
                PPXI9303_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function PXI9303_LoadParaDA Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal pADPara As PXI9303_PARA_DA) As Long
```

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 PXI9303_PARA_DA 的结构指针类型，它负责返回 PXI 硬件参数值，关于结构指针类型 PXI9303_PARA_DA 请参考 PXI9303.h 或 PXI9303.Bas 或 PXI9303.Pas 函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaDA SaveParaDA ReleaseDevice

♦ **将硬件参数结构体值复位为出厂默认值**

函数原型：

Visual C++:

```
BOOL ResetParaDA (HANDLE hDevice,
                  PPXI9303_PARA_DA pDAPara)
```

Visual Basic:

```
Declare Function PXI9303_ResetParaDA Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal pADPara As PXI9303_PARA_DA) As Long
```

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 PXI9303_PARA_DA 的详细介绍请参考 PXI9303.h 或 PXI9303.Bas 或 PXI9303.Pas 函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。调用此函数后，该参数指向的结构体成员将被复位至默认值。

nDAChannel DA 通道号，取值范围为[0, 1]。

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

相关函数： CreateDevice LoadParaDA SaveParaDA ResetParaDA ReleaseDevice

第四章 硬件参数结构

第一节、DA 的硬件参数

Visual C++：

```
typedef struct _PXI9303_PARA_DA
{
    LONG Frequency;          // 输出频率，单位为 Hz, [1,10000]
    LONG TriggerMode;     // 触发模式选择
    LONG TriggerSel;        // Trigger 触发选择
    LONG TriggerDir;        // 触发方向选择
    LONG ClockSource;          // 时钟源选择
    LONG SingleOutput;         // 是否单点输出 (1：单点输出 0：连续点输出）
} PXI9303_PARA_DA, *PPXI9303_PARA_DA;
```

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。

TriggerMode 触发类型所使用的选项。

常量名	常量值	功能定义
PXI9303_DA_TRIGMODE_SOFT	0x00	软件触发
PXI9303_DA_TRIGMODE_TRIG	0x01	Trig 触发

TTriggerSel 触发选择所使用的选项。

常量名	常量值	功能定义
PXI9303_DA_TRIG_PXI0	0x00	PXI0 触发
PXI9303_DA_TRIG_PXI1	0x01	PXI1 触发
PXI9303_DA_TRIG_PXI2	0x02	PXI2 触发
PXI9303_DA_TRIG_PXI3	0x03	PXI3 触发
PXI9303_DA_TRIG_PXI4	0x04	PXI4 触发
PXI9303_DA_TRIG_PXI5	0x05	PXI5 触发
PXI9303_DA_TRIG_PXI6	0x06	PXI6 触发
PXI9303_DA_TRIG_PXI7	0x07	PXI7 触发

TriggerDir 触发方向所使用的选项。

常量名	常量值	功能定义
PXI9303_DA_TRIGDIR_IN	0x00	输入
PXI9303_DA_TRIGDIR_OUT	0x01	输出

ClockSource 时钟源所使用的选项。

常量名	常量值	功能定义
PXI9303_DA_CLOCKSRC_IN	0x00	内部时钟
PXI9303_DA_CLOCKSRC_BACK	0x01	内部时钟
PXI9303_DA_CLOCKSRC_OUT	0x02	外部时钟

OutputRange 模拟量输入范围所使用的选项。

常量名	常量值	功能定义
PXI9303_OUTPUT_N10000_P10000mV	0x00	±10000mV

PXI9303_OUTPUT_N5000_P5000mV	0x01	±5000mV
PXI9303_OUTPUT_0_P10000mV	0x02	0～10000mV
PXI9303_OUTPUT_0_P5000mV	0x03	0～5000mV

SelectFunction 中的 lFunction 参数。

常量名	常量值	功能定义
PXI9303_FUNCTION_FREQUENCY	0x00	频率测量
PXI9303_FUNCTION_CYCLE	0x01	周期测量
PXI9303_FUNCTION_PULSEWIDTH	0x02	脉冲宽度测量
PXI9303_FUNCTION_PULSEOUTPUT	0x03	计数器脉冲输出

lOutputMode 使用的硬件选项。

常量名	常量值	功能定义
PXI9303_OPMODE_SINGLE	0x00	单次脉冲
PXI9303_OPMODE_N	0x01	N 脉冲
PXI9303_OPMODE_CONTINUOUS	0x02	连续脉冲

第二节、DA 状态参数结构（PXI9303_STATUS_DA）

Visual C++：

```
typedef struct _PXI9303_STATUS_DA
{
    ULONG bNotEmpty;    // 板载FIFO存储器的DA0非空标志，= TRUE非空，        = FALSE  空
    ULONG bHalf;        // 板载FIFO存储器的DA0半满标志，= TRUE半满以上，     = FALSE  半满以下
    ULONG bOverflow;// 板载FIFO存储器的DA0溢出标志，= TRUE已发生溢出，  = FALSE   未发生溢出
} PXI9303_STATUS_DA, *PPXI9303_STATUS_DA;
```

第三节、脉冲输出硬件参数

Visual C++：

```
typedef struct _PXI9303_PULSE_OUTPUT
{
    LONG lHPulseTime;           // 高脉冲时长(100~107374182400nS)
    LONG lLPulseTime;           // 低脉冲时长(100~107374182400nS)
    LONG lDelayTime;            // 延时时长(100~107374182400nS)
    LONG bOutputSel;        // 脉冲输出选择   TRUE空闲状态为高,FALSE空闲状态为低
    LONG lOutputMode;           // 输出方式
    LONG lOutputCount;          // N脉冲输出个数(输出方式为N脉冲有效)
} PXI9303_PULSE_OUTPUT, *PPXI9303_PULSE_OUTPUT;
```

第五章 数据格式转换与排列规则

第一节、DA 电压值转换成 LSB 原码数据的换算方法

量程(伏)	计算机语言换算公式(标准 C 语法)	Lsb 取值范围
0～5000mV	Lsb = Volt / (5000.00/65536)	[0，65535]
0～10000mV	Lsb = Volt / (10000.00/65536)	[0，65535]
±5000mV	Lsb = Volt / (10000.00/65536) + 32768	[0，65535]
±10000mV	Lsb = Volt / (20000.00/65536) + 32768	[0，65535]

第二节、DA 采集函数的 ADBuffer 缓冲区中的数据排放规则

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

第六章 上层用户函数接口应用实例

第一节、简易程序演示说明

一、怎样使用 WriteDeviceOneDA 向 DA 的 FIFO 中写入批量数据

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 PXI9303.h 和 Sys.cpp)。

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

其简易程序默认存放路径为：系统盘\PXI\PXI9303\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 PXI9303.h 和 DADoc.cpp)。

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

其默认存放路径为：系统盘\PXI\ PXI9303\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第七章 共用函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“PXI9303_”）

函数名	函数功能	备注
①PXI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
②ISA 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、PXI 内存映射寄存器操作函数原型说明

- **取得指定内存映射寄存器的线性地址和物理地址**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_GetDeviceBar Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pulPCIBar As Long) As Boolean
```

LabVIEW:

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pulPCIBar[6]) 返回 PCI BAR 所有地址,具体 PCI BAR 中有多少可用地址请看硬件说明书

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

- **以单字节（即 8 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型：

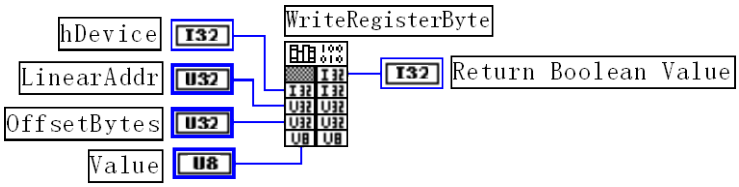
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_WriteRegisterByte Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal LinearAddr 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 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值

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 GetDeviceBar 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
    :
```

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

函数原型：

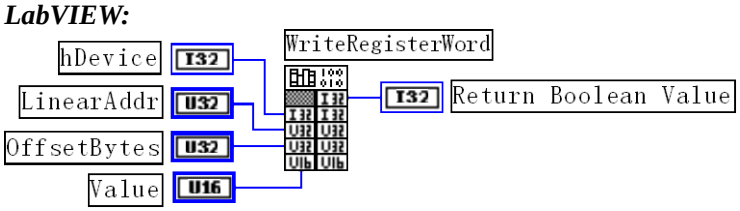
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_WriteRegisterWord Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByVal LinearAddr As Long,_
                                                ByVal OffsetBytes As Long,_
                                                ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr　PXI 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　GetDeviceBar　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

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

♦ **以四字节（即 32 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

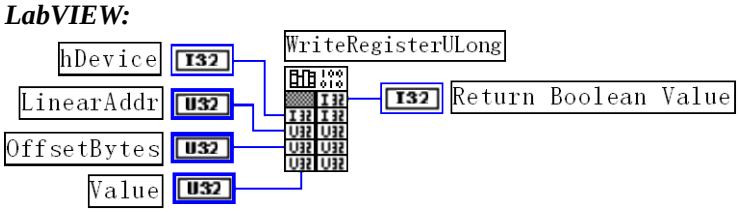
```
BOOL WriteRegisterULong( HANDLE hDevice,
```

```
                                __int64 LinearAddr,
                                ULONG OffsetBytes,
                                ULONG Value)
```

Visual Basic:

```
Declare Function PXI9303_WriteRegisterULong Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByVal LinearAddr As Long,_
                                                ByVal OffsetBytes As Long,_
                                                ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr PXI 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　GetDeviceBar　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

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

- **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型：

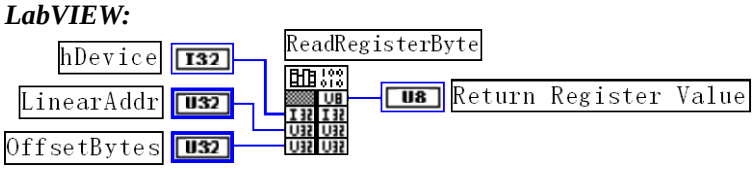
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadRegisterByte Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByVal LinearAddrAs As Long,_
                                                ByVal OffsetBytesAs As Long) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr PXI 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 GetDeviceBar 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 PXI 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
    :
```

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

函数原型：

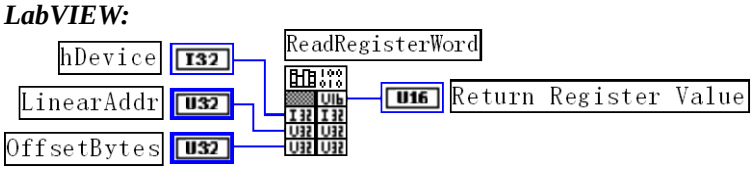
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadRegisterWord Lib "PXI9303_32" (_
                                                    ByVal hDevice As Long,_
                                                    ByVal LinearAddr As Long,_
                                                    ByVal OffsetBytes As Long) As Long
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr PXI 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 GetDeviceBar 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 PXI 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
    :
```

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

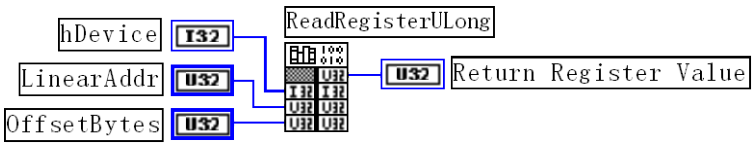
函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadRegisterULong Lib "PXI9303_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal LinearAddr As Long, _
                                    ByVal OffsetBytes As Long) As Long
```



功能：以四字节（即 32 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

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

Visual Basic 程序举例:

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

第三节、I0 端口读写函数原型说明

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

♦ **以单字节(8Bit)方式写 I/O 端口**

函数原型：

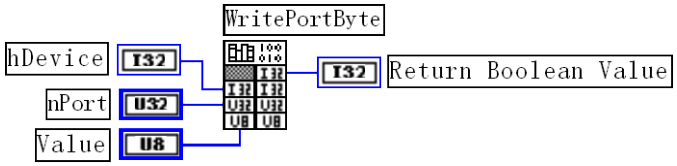
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_WritePortByte Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByVal nPort As Long,_
                                                ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pbPort 设备的 I/O 端口号。

Value 写入由 pbPort 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：

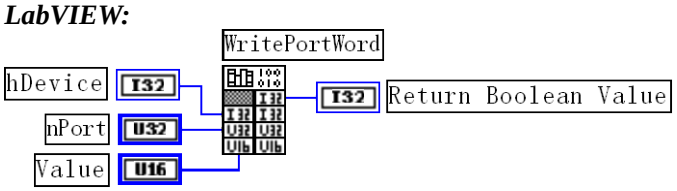
Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    __int64 pbPort,
                    WORD Value)
```

Visual Basic:

```
Declare Function PXI9303_WritePortWord Lib "PXI9303_32" (_
                                                ByVal hDevice As Long,_
                                                ByVal nPort As Long,_
                                                ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pbPort 设备的 I/O 端口号。

Value 写入由 pbPort 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：

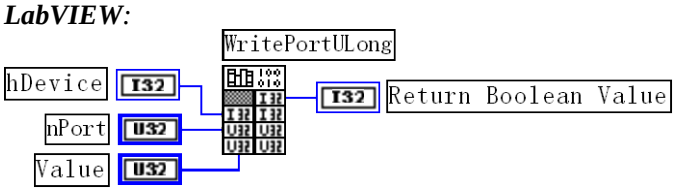
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_WritePortULong Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal nPort 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。

pbPort 设备的 I/O 端口号。

Value 写入由 pbPort 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：

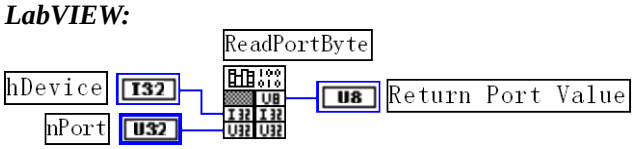
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadPortByte Lib "PXI9303_32" (_
                                      ByVal hDevice As Long,_
                                      ByVal nPort As Long) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pbPort 设备的 I/O 端口号。

返回值：返回由 pbPort 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：

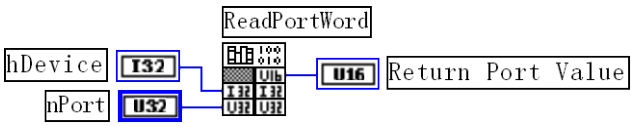
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadPortWord Lib "PXI9303_32" (ByVal hDevice As Long, ByVal nPort As Long)
As Long
```

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
返回值：返回由 pbPort 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**

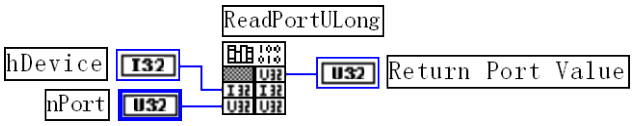
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function PXI9303_ReadPortULong Lib "PXI9303_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal nPort As Long) As Long
```

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
返回值：返回由 pbPort 指定端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

第四节、线程操作函数原型说明

♦ **创建内核系统事件**

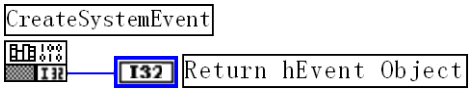
函数原型：
Visual C++:

```
HANDLE CreateSystemEvent(void)
```

Visual Basic:

```
Declare Function PXI9303_CreateSystemEvent Lib "PXI9303_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 PXI9303_ReleaseSystemEvent Lib "PXI9303_32" (ByVal hEvent As Long) As Boolean
```

LabVIEW:
请参见相关演示程序。

功能：释放系统内核事件对象。
参数：hEvent　被释放的内核事件对象。它应由CreateSystemEvent成功创建的对象。
返回值：若成功，则返回 TRUE。