PXI8996 数据采集卡

WIN2000/XP 驱动程序使用说明书

北京阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PXIxxxx_ 则被省略。如 PXI8996_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InitDeviceProAD、InitDeviceDmaAD、ReadDeviceProAD_Npt等。而底层用户函数如WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。

第二节、如何管理 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相应的驱动函数，如InitDeviceProAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD_Npt (或ReadDeviceProAD_Half)函数可以用hDevice句柄实现对AD数据的采样读取等。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何用非空查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD函数初始化AD部件，关于采样通道、频率等参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即

可实现所有硬件参数和设备状态的初始化。然后用StartDeviceProAD即可启动AD部件，开始AD采样，然后便可用ReadDeviceProAD_Npt反复读取AD数据以实现连续不间断采样。当您需要暂停设备时，执行StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceProAD_Npt虽然主要面对批量读取、高速连续采集而设计，但亦可用它以单点或几点的方式读取AD数据，以满足慢速、高实时性采集需要）。具体执行流程请看下面的图 2.1.1。

第四节、如何用半满查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD函数初始化AD部件，关于采样通道、频率等参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用StartDeviceProAD即可启动AD部件，开始AD采样，接着调用GetDevStatusProAD函数以查询AD的存储器FIFO的半满状态，如果达到半满状态，即可用ReadDeviceProAD_Half函数读取一批半满长度（或半满以下）的AD数据，然后接着再查询FIFO的半满状态，若有效再读取，就这样反复查询状态反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceProAD_Half函数在半满状态有效时也可以单点或几点的方式读取AD数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）具体执行流程请看下面的图 2.1.2。

第五节、如何用 Dma 直接内存方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceDmaAD函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hDmaEvent赋给InitDeviceDmaAD的相应参数，它将作为Dma事件的变量。然后用StartDeviceDmaAD即可启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hDmaEvent事件的发生，当当前缓冲段没有被DMA完成时，自动使所在线程进入睡眠状态（不消耗CPU时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用GetDevStatusDmaAD来确定哪一段缓冲是新的数据，即刻处理该数据，至到所有的缓冲段变为旧数据段。然后再回到WaitForSingleObject，就这样反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceDmaAD，当您需要关闭AD设备时，ReleaseDeviceDmaAD便可帮您实现（但设备对象hDevice依然存在）。具体执行流程请看图 2.1.3。

第六节、如何用中断方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceIntAD函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pPara参数结构体决定的。您只需要对这个pPara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。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hEvent赋给InitDeviceIntAD的相应参数，它将作为接受AD半满中断事件的变量。然后用StartDeviceIntAD即可启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hEvent中断事件的发生，在中断未到时，自动使所在线程进入睡眠状态（不消耗CPU时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用ReadDeviceIntAD函数一批半满长度（或半满以下）的AD数据，然后再接着再等待FIFO的半满中断事件，若有效再读取，就这样反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceIntAD，当您需要关闭AD设备时，ReleaseDeviceIntAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceIntAD函数在半满中断事件发生时可以单点或几点的方式读取AD数据，只是到下一次半满中断事件到来时的时间间隔会变得非常短，而不再是半满间隔，但它不同于半满查询方式读取，由于半满中断属于硬件中断，其优先级别高于所有软件，所以您单点或几点读取AD数据时，千万不能让中断间隔太短，否则，有可能使您的整个系统被半满中断事件吞没，就象死机一样，不能动弹。　切忌、切忌！）具体执行流程请看图 2.1.4。

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

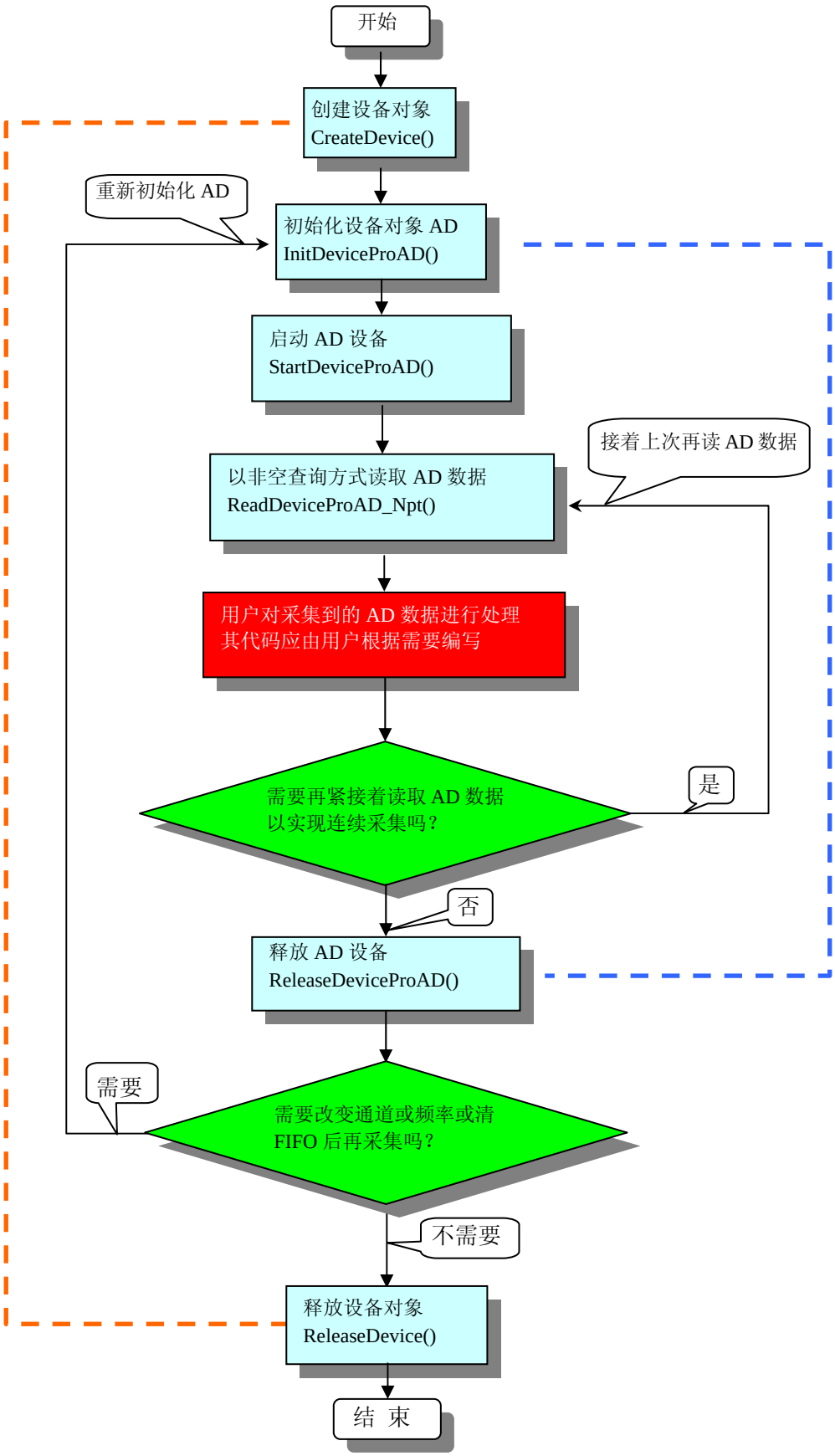


图 2.1.1 非空查询方式 AD 采集过程

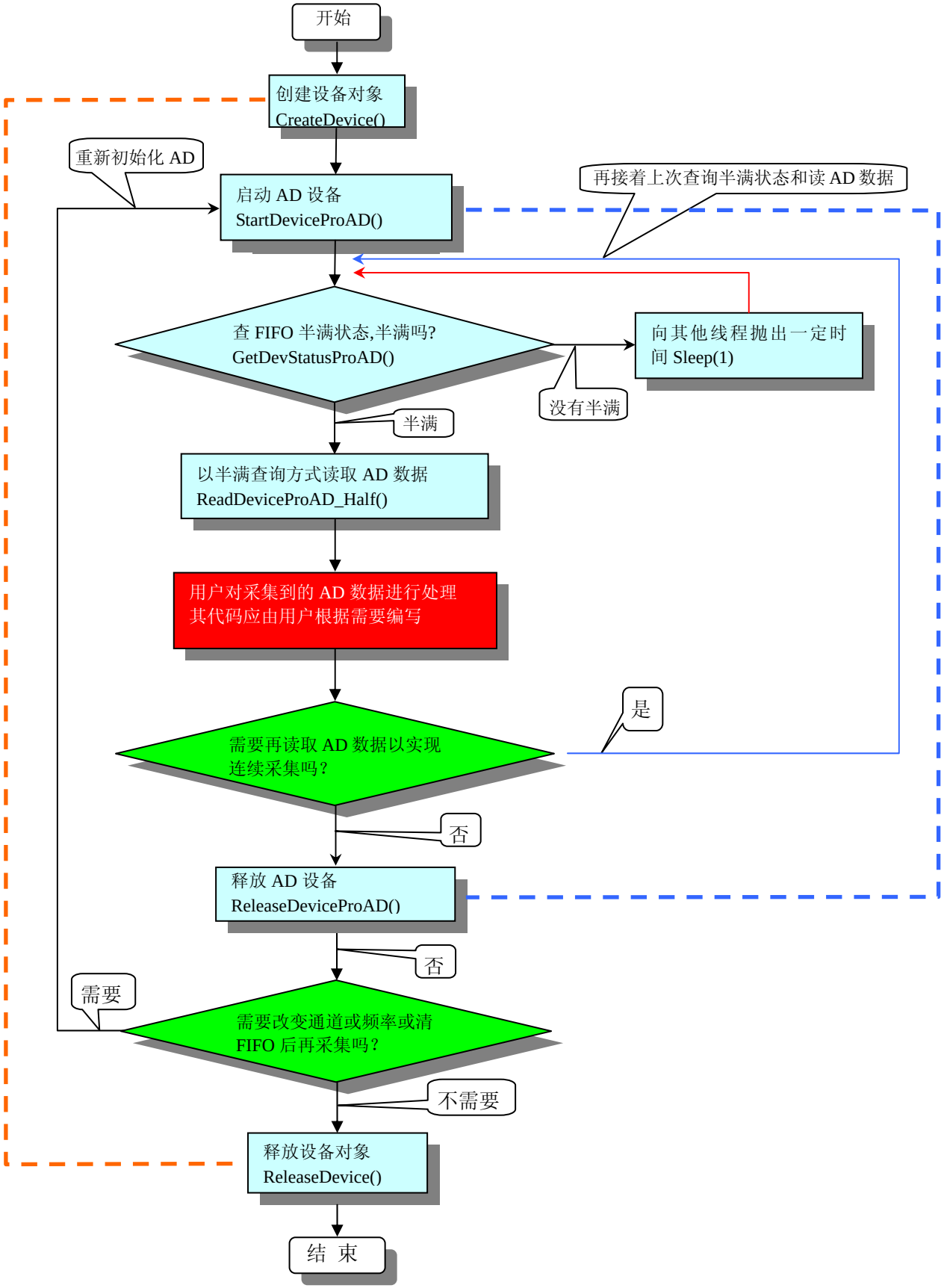


图 2.1.2 半满查询方式 AD 采集过程

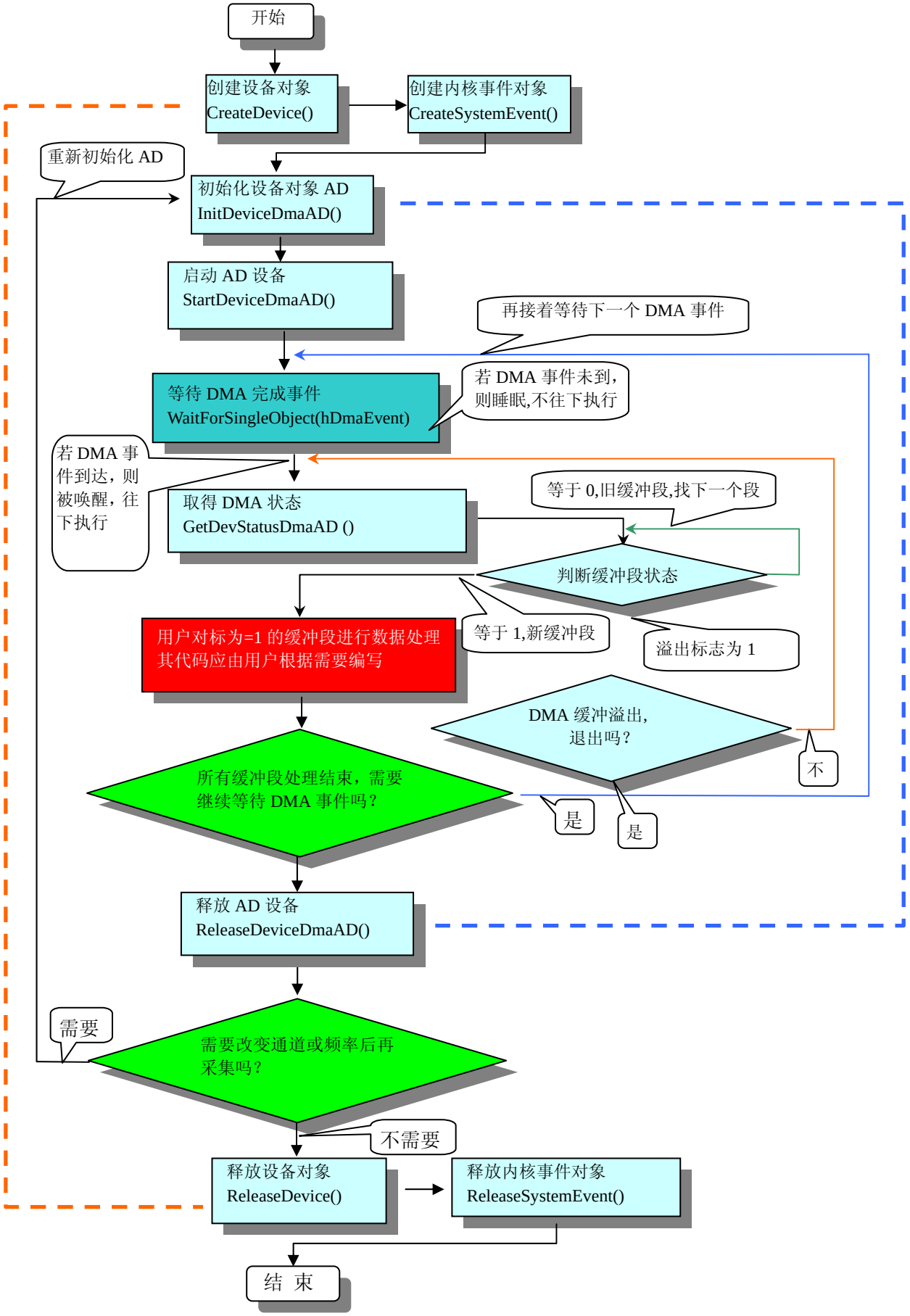


图 2.1.3 DMA 方式 AD 采集实现过程

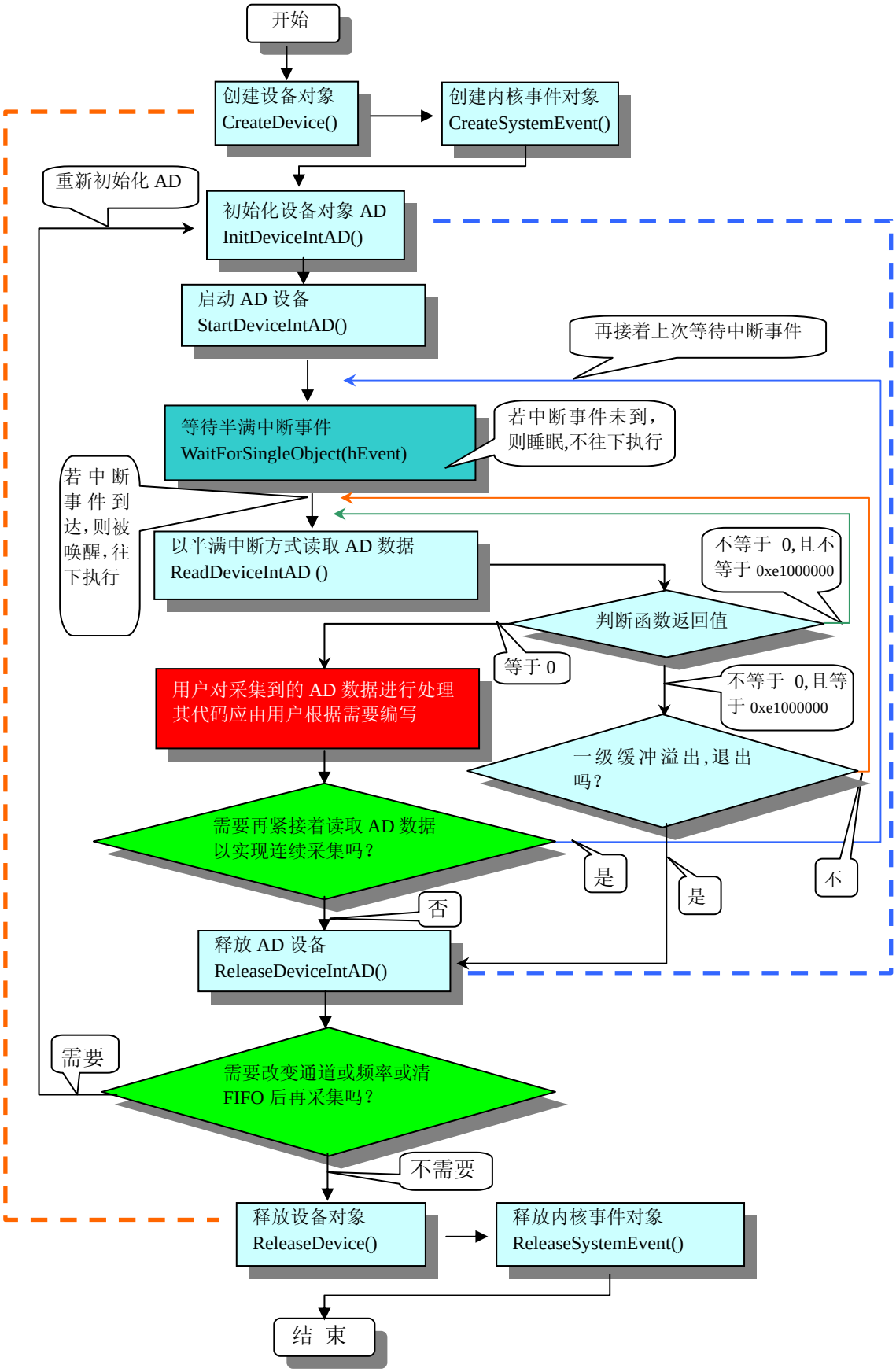


图 2.1.4 中断方式 AD 采集实现过程

第七节、哪些函数对您不是必须的

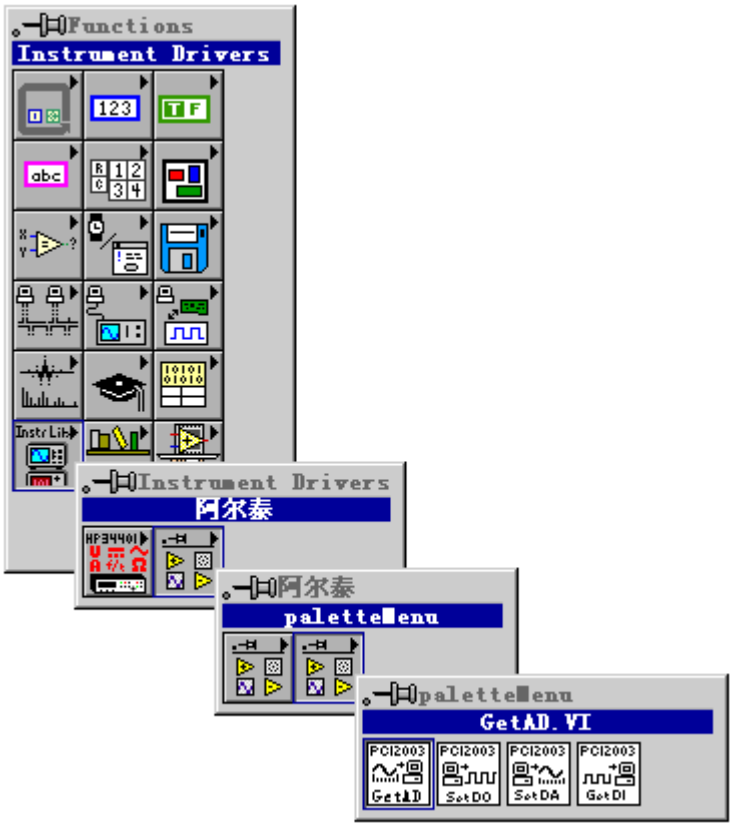
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， 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 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，只关心首末通道、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。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总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，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“PXI8996_”）

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
CreateDeviceEx	创建 PXI 设备对象(用设备物理号)	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
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	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 程序方式 AD 读取函数		
InitDeviceProAD	初始化 AD 部件准备传输	上层用户
SetDevFrequencyAD	可动态改变 AD 采样频率	上层用户
StartDeviceProAD	启动 AD 设备，开始转换	上层用户
ReadDeviceProAD_Npt	连续读取当前 PXI 设备上的 AD 数据	上层用户
GetDevStatusProAD	取得当前 PXI 设备 FIFO 半满状态	上层用户
ReadDeviceProAD_Half	连续批量读取 PXI 设备上的 AD 数据	上层用户
StopDeviceProAD	暂停 AD 设备	上层用户
ReleaseDeviceProAD	释放设备上的 AD 部件	上层用户
③ DMA 方式 AD 读取函数（唯有此种方式效率最高）		
InitDeviceDmaAD	初始化 AD 部件，如通道等	上层用户
StartDeviceDmaAD	启动 AD 采集	上层用户
GetDevStatusDmaAD	取得 DMA 的各种状态	上层用户
SetDevStatusDmaAD	清除 DMA 状态	
StopDeviceDmaAD	停止 AD 采集	上层用户
ReleaseDeviceDmaAD	释放设备上的 AD 部件	上层用户
④ 中断方式 AD 读取函数（唯有此种方式采用强制二级队列缓冲和动态链表技术）		

InitDeviceIntAD	初始化 PCI 设备 AD 部件，如通道等	上层用户
StartDeviceIntAD	启动 AD 采集	上层用户
ReadDeviceIntAD	连续批量读取 PCI 设备上的 AD 数据	上层用户
StopDeviceIntAD	停止 AD 采集	上层用户
ReleaseDeviceIntAD	释放设备上的 AD 部件	上层用户
⑤ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户

使用需知：

Visual C++ & C++Builder：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\Art\PXI8996\INCLUDE\PXI8996.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 PXI8996.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 “PXI8996.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。

C++ Builder:

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头文件(PXI8996.H)写进您的源程序头部。如：#include “\Art\PXI8996\Include\PXI8996.h”，然后再将 PXI8996.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”（Add to Project）命令，在弹出的对话框中分别选择文件类型：Library file (*.lib) ， 即可选择 PXI8996.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。

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 PXI8996.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

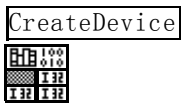
Delphi:

要使用如下函数一个关键的问题是首先必须将我们提供的单元模块文件（*.Pas）加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单，执行其中的"Project Manager"命令，在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PXI8996.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PXI8996”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PXI8996;  // 注意： 在此加入驱动程序接口单元 PXI8996
```

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 PXI8996.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

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

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

第二节、设备对象管理函数原型说明

◆ **创建设备对象函数（逻辑号）**

函数原型：

Visual C++ & C++Builder：

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

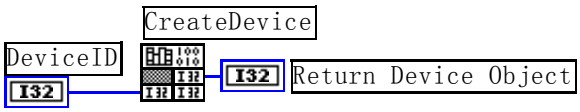
Visual Basic：

```
Declare Function CreateDevice Lib "PXI8996" (Optional ByVal DeviceLgcID As Integer = 0) As Long
```

Delphi：

```
Function CreateDevice(DeviceLgcID : Integer = 0) : Integer;
                    StdCall; External 'PXI8996' Name ' CreateDevice ';
```

LabVIEW：



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

参数：

DeviceLgcID 逻辑设备ID(Logic Device Identifier)标识号。当向同一个Windows系统中加入若干相同类型的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PXI8996 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PXI8996 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，调用CreateDeviceEx函数实现。

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

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++ & C++Builder 程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
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
```

：

♦ **创建设备对象函数（物理号）**

函数原型：

Visual C++ & C++Builder：

HANDLE CreateDeviceEx(int DevicePhysID = 0)

Visual Basic：

Declare Function CreateDeviceEx Lib "PXI8996" (Optional ByVal DevicePhysID As Integer = 0) As Long

Delphi：

Function CreateDeviceEx(DevicePhysID : Integer = 0) : Integer;
StdCall; External 'PXI8996' Name ' CreateDeviceEx ';

LabVIEW：

请参考相关演示程序。

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

参数：

DevicePhysID 物理设备ID(Physic Device Identifier)标识号。由CreateDevice函数的DeviceLgcID参数说明中可以看出，逻辑ID号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如A、B、C、D四个卡，构成 128 个通道（32*4），其通道序列为 0-127，每个通道接入不同物理意义的模拟信号，我们要求A卡位于 0-31 通道上，B卡位于 32-63 通道上，C卡位于 64-95 通道上，而D卡则位于 96-127 通道上，而其逻辑设备ID号在同一台计计算机上按不同顺序插入会发生变化，即便在不同计算机上按相同顺序插入也可能会因主板制造商的不同定义而发生变化，所以您可能由此无法确定 0-127 的通道分别接入了什么信号。那么如何将各个设备在设备链中的物理位置固定下来呢？那么物理设备ID的使用帮您解决了这个问题。它是在卡上提供了一个拔码器DID，可以由用户为各个设备手动设置不同的物理ID号，当调用CreateDeviceEx函数时，只需要指定该参数的值与您在拔码器上设定的值一样即可，驱动程序会自动跟踪拔码器值与此相等的设备。它的取值范围通常在[0, 15]之间。

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

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount
GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得本计算机系统中 PXI8996 设备的总数量**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

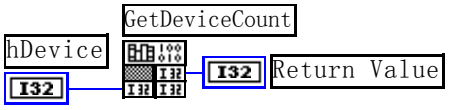
Visual Basic:

Declare Function GetDeviceCount Lib "PXI8996" (ByVal hDevice As Long) As Integer

Delphi：

Function GetDeviceCount (hDevice : Integer) : Integer;
StdCall; External 'PXI8996' Name ' GetDeviceCount ';

LabVIEW：



功能：取得 PXI8996 设备的数量。
参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
返回值：返回系统中 PXI8996 的数量。
相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount
GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

- **取得该设备当前逻辑 ID 和物理 ID**

函数原型：
Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PXI8996" (ByVal hDevice As Long,_
                                                   ByRef DeviceLgcID As Long,_
                                                   ByRef DevicePhysIDAs Long ) As Boolean
```

Delphi：

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DeviceLgcID : Pointer;
                             DevicePhysID : Pointer) : Boolean;
                             StdCall; External 'PXI8996' Name ' GetDeviceCurrentID ';
```

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或CreateDeviceEx创建。
DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。
DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拔码器 DID 决定。
返回值：如果初始化设备对象成功，则返回TRUE， 否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。
相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount
GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

- **用对话框控件列表计算机系统中所有 PXI8996 设备各种配置信息**

函数原型：
Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ListDeviceDlg Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ListDeviceDlg (hDevice : Integer) : Boolean;
                        StdCall; External 'PXI8996' Name ' ListDeviceDlg ';
```

LabVIEW：
请参考相关演示程序。

功能：列表系统中 PXI8996 的硬件配置信息。
参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
返回值：若成功，则弹出对话框控件列表所有 PXI8996 设备的配置情况。
相关函数： CreateDevice　ReleaseDevice

- **释放设备对象所占的系统资源及设备对象**

函数原型：
Visual C++ & C++Builder:

```
BOOL ReleaseDevice(HANDLE hDevice)
```

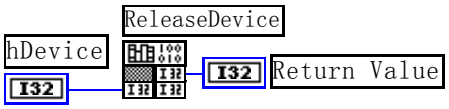
Visual Basic:

```
Declare Function ReleaseDevice Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function ReleaseDevice(hDevice : Integer) : Boolean;
                        StdCall; External 'PXI8996' Name ' ReleaseDevice ';
```

LabVIEW:



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

应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice后，再一次执行这些函数前，必须执行一次ReleaseDevice函数，以释放由CreateDevice占用的系统软硬件资源，如DMA控制器、系统内存等。只有这样，当您再次调用CreateDevice函数时，那些软硬件资源才可被再次使用。

第三节、AD 程序查询方式采样操作函数原型说明

♦ **动态改变采样频率**

函数原型：

Visual C++ & C++Builder:

```
LONG SetDevFrequencyAD ( HANDLE hDevice,
                         LONG nFrequency)
```

Visual Basic:

```
Declare Function SetDevFrequencyAD Lib "PXI8996" (ByVal hDevice As Long, _
                                                  ByVal nFrequency As Long) As Long
```

Delphi:

```
Function SetDevFrequencyAD (hDevice : Integer;
                            nFrequency : LongInt) : LongInt;
                            StdCall; External 'PXI8996' Name ' SetDevFrequencyAD ';
```

LabVIEW:

请参考相关演示程序。

功能：在 AD 转换过程中，动态改变采样频率,并返回实际频率值。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nFrequency 指定 AD 的当前采样频率。
返回值：如果动态改变采样频率成功，则返回实际频率值，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevFrequencyAD	InitDeviceProAD
StartDeviceProAD	ReadDeviceProAD_Npt	GetDevStatusProAD
ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
ReleaseDevice		

♦ **初始化设备**

函数原型：

Visual C++ & C++Builder:

```
BOOL InitDeviceProAD (HANDLE hDevice,
                      PPXI8996_PARA_AD pADPara);
```

Visual Basic:

```
Declare Function InitDeviceProAD Lib "PXI8996" (ByVal hDevice As Long, _
                                                ByRef pADPara As PXI8996_PARA_AD) As Boolean
```

Delphi:

```
Function InitDeviceProAD (hDevice : Integer;
                          pADPara: Pointer) : Boolean;
```

StdCall; External 'PXI8996' Name ' InitDeviceProAD';

LabVIEW：

请参考相关演示程序。

功能：初始化设备。

参数：

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

pADPara设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如AD采样通道、采样频率等。关于PXI8996_PARA_AD具体定义请参考PXI8996.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《AD硬件参数结构》章节。

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　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

- **启动 AD 设备(Start device AD for program mode)**

函数原型：

Visual C++ & C++Builder::

```
BOOL StartDeviceProAD ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StartDeviceProAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function StartDeviceProAD (hDevice : Integer ): Boolean;
                    StdCall; External 'PXI8996' Name ' StartDeviceProAD ';
```

LabVIEW

请参考相关演示程序。

功能：启动AD设备，它必须在调用InitDeviceProAD后才能调用此函数。该函数除了启动AD设备开始转换以外，不改变设备的其他任何状态。

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

返回值：如果调用成功，则返回TRUE，且AD立刻开始转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

- **读取 PXI 设备上的 AD 数据**

① 使用 FIFO 的非空标志读取 AD 数据

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ReadDeviceProAD_Npt Lib "PXI8996" (ByVal hDevice As Long, _
                                                    ByRef ADBuffer As Long,_
                                                    ByVal nReadSizeWords As Long,_
                                                    ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Npt(hDevice : Integer;
                             ADBuffer : Pointer;
                             nReadSizeWords : LongInt
                             nRetSizeWords : Pointer) : Boolean;
```

StdCall; External 'PXI8996' Name ' ReadDeviceProAD_Npt ';

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数读取设备上的AD数据。此函数使用FIFO的非空标志进行读取AD数据。

参数：

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

ADBuffer 接受AD数据的用户缓冲区，它可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次ReadDeviceProAD_Npt操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间。此参数值只与ADBuffer[]指定的缓冲区大小有效，而与FIFO存储器大小无效。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值： 其返回值表示所成功读取的数据点数(字)，也表示当前读操作在ADBuffer缓冲区中的有效数据量。通常情况下其返回值应与ReadSizeWords参数指定量的数据长度(字)相等，除非用户在这个读操作以外的其他线程中执行了ReleaseDeviceProAD函数中断了读操作，否则设备可能有问题。对于返回值不等于nReadSizeWords参数值的，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

当前错误码	功能定义
0xE1000000	其他不可预知的错误
0xE2000000	表示用户提前终止读操作

相关函数： CreateDevice, SetDevFrequencyAD, InitDeviceProAD, StartDeviceProAD, ReadDeviceProAD_Npt, GetDevStatusProAD, ReadDeviceProAD_Half, StopDeviceProAD, ReleaseDeviceProAD, ReleaseDevice

② 使用 FIFO 的半满标志读取 AD 数据

◆ **取得 FIFO 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDevStatusProAD ( HANDLE hDevice,
                         PPXI8996_STATUS_AD pADStatus);
```

Visual Basic:

```
Declare Function GetDevStatusProAD Lib "PXI8996" (ByVal hDevice As Long,_
                                                  ByRef pADStatus As PXI8996_STATUS_AD) As Boolean
```

Delphi:

```
Function GetDevStatusProAD (hDevice : Integer;
                            PADStatus : PPXI8996_STATUS_AD) : Boolean;
                            StdCall; External 'PXI8996' Name ' GetDevStatusProAD ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数查询FIFO存储器的状态（半满标志、非空标志、溢出标志）。我们通常用半满标志去同步半满读操作。当半满标志有效时，再紧接着用ReadDeviceProAD_Half读取FIFO中的半满有效AD数据。

参数：

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

pADStatus 获得AD的各种当前状态。它属于结构体，具体定义请参考《AD状态参数结构（PXI8996_STATUS_AD）》章节。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。若用户选择半满查询方式读取AD数据，则当GetDevStatusProAD函数取得的bHalf等于TRUE，应立即调用ReadDeviceProAD_Half读取FIFO中的半满数据。否则用户应继续循环轮询FIFO半满状态，直到有效为止。注意

在循环轮询期间，可以用Sleep函数抛出一定时间给其他应用程序(包括本应用程序的主程序和其他子线程)，以提高系统的整体数据处理效率。

相关函数： CreateDevice　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

♦ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "PXI8996" (ByVal hDevice As Long, _
                                                     ByRef ADBuffer As Long,_
                                                     ByVal nReadSizeWords As Long ,_
                                                     ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Half(hDevice : Integer;
                              ADBuffer : Pointer;
                              nReadSizeWords : LongInt;
                              nRetSizeWords : Pointer) : Boolean;
                              StdCall; External 'PXI8996' Name ' ReadDeviceProAD_Half ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用GetDevStatusProAD后取得的FIFO状态bHalf等于TRUE(即半满状态有效)时，应立即用此函数读取设备上FIFO中的半满AD数据。

参数：

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

ADBuffer 接受AD数据的用户缓冲区，通常可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次ReadDeviceProAD_Half操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间，而且应等于FIFO总容量的二分之一(如果用户有特殊需要可以小于FIFO的二分之一长)。比如设备上配置了 1K FIFO，即 1024 字，那么这个参数应指定为 512 或小于 512。

返回值：如果成功的读取由nReadSizeWords参数指定量的AD数据到用户缓冲区，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　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

♦ **暂停 AD 设备**

函数原型：

Visual C++ & C++Builder:

```
BOOL StopDeviceProAD ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StopDeviceProAD Lib "PXI8996" (ByVal hDevice As Long )As Boolean
```

Delphi:

```
Function StopDeviceProAD (hDevice : Integer ) : Boolean;
                          StdCall; External 'PXI8996' Name ' StopDeviceProAD ';
```

LabVIEW

请参考相关演示程序。

功能：暂停AD设备。它必须在调用StartDeviceProAD后才能调用此函数。该函数除了停止AD设备不再转换

以外，不改变设备的其他任何状态。此后您可再调用StartDeviceProAD函数重新启动AD，此时AD会按照暂停以前的状态（如FIFO存储器位置、通道位置）开始转换。

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

返回值：如果调用成功，则返回TRUE，且AD立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice SetDevFrequencyAD InitDeviceProAD StartDeviceProAD ReadDeviceProAD_Npt GetDevStatusProAD ReadDeviceProAD_Half StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

♦ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

BOOL ReleaseDeviceProAD(HANDLE hDevice)

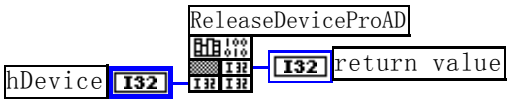
Visual Basic:

Declare Function ReleaseDeviceProAD Lib "PXI8996" (ByVal hDevice As Long) As Boolean

Delphi：

Function ReleaseDeviceProAD (hDevice : Integer) : Boolean;
StdCall; External 'PXI8996' Name ' ReleaseDeviceProAD ';

LabVIEW:



功能：释放设备上的 AD 部件。

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

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

应注意的是，InitDeviceProAD必须和ReleaseDeviceProAD函数一一对应，即当您执行了一次InitDeviceProAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceProAD函数，以释放由InitDeviceProAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用InitDeviceProAD函数时，那些软硬件资源才可被再次使用。

相关函数： CreateDevice InitDeviceProAD ReleaseDeviceProAD ReleaseDevice

♦ 程序查询方式采样函数一般调用顺序

非空查询方式:

① CreateDevice
② InitDeviceProAD
③ StartDeviceProAD
④ ReadDeviceProAD_Npt
⑤ StopDeviceProAD
⑥ ReleaseDeviceProAD
⑦ ReleaseDevice

注明：用户可以反复执行第④步，以实现高速连续不间断大容量采集。

半满查询方式:

① CreateDevice
② InitDeviceProAD
③ StartDeviceProAD
④ GetDevStatusProAD
⑤ ReadDeviceProAD_Half
⑥ StopDeviceProAD
⑦ ReleaseDeviceProAD
⑧ ReleaseDevice

注明：用户可以反复执行第④、⑤步，以实现高速连续不间断大容量采集。

关于两个过程的图形说明请参考《使用纲要》。

第四节、AD 直接内存存取 DMA 方式采样操作函数原型说明

（注：函数中的“Dma”字符是 Direct Memory Access 的缩写，标明以直接内存存取方式）

◆ **初始化设备上的 AD 对象**

函数原型：

Visual C++ & C++ Builder:

```
BOOL InitDeviceDmaAD( HANDLE hDevice,
                      HANDLE hDmaEvent,
                      LONG ADBuffer[ ],
                      LONG nReadSizeWords,
                      LONG nSegmentCount,
                      LONG nSegmentSizeWords,
                      PPXI8996_PARA_AD pADPara )
```

Visual Basic:

```
Declare Function InitDeviceDmaAD Lib "PXI8996" (ByVal hDevice As Long, _
                                                ByVal hDmaEvent As Long, _
                                                ByRef ADBuffer As Long,_
                                                ByVal nReadSizeWords As Long,_
                                                ByVal nSegmentCount As Long,_
                                                ByVal nSegmentSizeWords As Long,_
                                                ByRef pADPara As PXI8996_PARA_AD ) As Boolean
```

Delphi:

```
Function InitDeviceDmaAD(hDevice : Integer;
                         hDmaEvent: Integer;
                         ADBuffer : Pointer;
                         nReadSizeWords : LongInt;
                         nSegmentCount : LongInt;
                         nSegmentSizeWords : LongInt;
                         pADPara : PPXI8996_PARA_AD) : Boolean;
                         StdCall; External 'PXI8996' Name ' InitDeviceDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备操作及DMA传输就绪有关工作，如预置AD采集通道、采样频率等。且让设备上的AD部件以硬件DMA的方式工作，但它并不启动AD采样，而是需要在此函数被成功调用之后，再调用StartDeviceDmaAD函数即可启动AD采样。

参数：

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

hDmaEvent DMA 事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件每次DMA完一个指定段长(nSegmentSizeWords)的数据时这个内核系统事件被触发一次。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当该事件没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，因为它并不消耗CPU时间。当hDmaEvent事件被触发成发信号状态，那么WaitForSingleObject将复位该内核系统事件对象，使其处于不发信号状态，并立即唤醒所在线程，继而执行WaitForSingleObject其后的代码，比如移走ADBuffer中的数据、分析数据、显示数据等，待处理完数据后再循环调用WaitForSingleObject，让所在线程再次进入睡眠状态，重复以上过程。所以利用DMA方式采集数据，不仅等待AD转换指定数据不需要消耗CPU时间，同时将AD数据从卡上传输到计算机主存更是不需要花消CPU时间，其效率是最高的。其具体实现方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》。

ADBuffer 接受AD数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。关于如何将缓冲区中的这些AD数据转换成相应的电压值，请参考请第六章《数据格式转换与排列规则》。注意该缓冲区最好定义为两维缓冲或数组，以便DMA数据传输和缓冲区数据处理分时错开，以更好的达到AD转换、传输、处理等过程的并行工作。**注意：该缓冲区的生命周期必须跨越DMA的整个操作周期，我建议最好将期置为全局缓冲区，即整个应用程序的生命周期内存在。否则，可能会造成严重的存储区访问违反。**

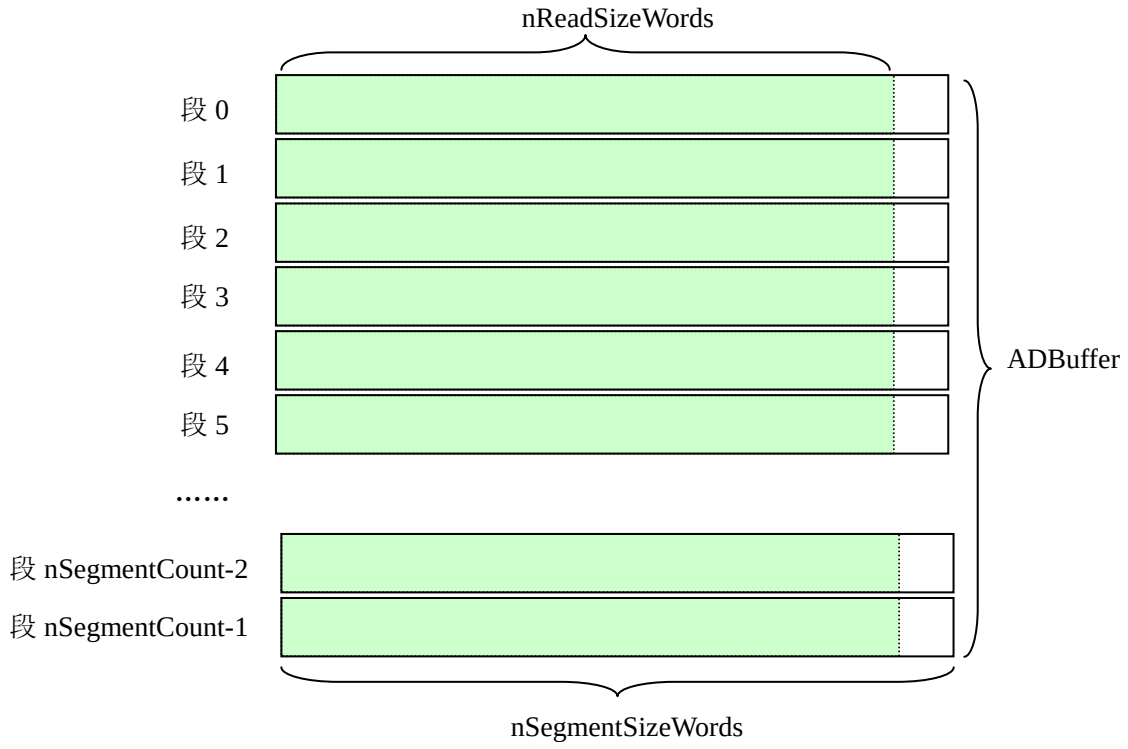
nReadSizeWords 在每个段缓冲中应 DMA 填充和用户读走的数据点数。它的取值范围不应小于 1，同时，不能大于段长 nSegmentSizeWords，其具体取值应根据采样通道数来确定其大小，通常应在段长范围内，取用为采

样通道数整数倍长，同时又最接近段长的读取长度来设置本参数。也就是说每当用户接受到 hDmaEvent 事件后，对相应段缓冲区作数据处理时只能从该段缓冲首单元开始往后共处理 nReadSizeWords 个数据采样点。

nSegmentCount 缓冲区段数。其取值范围为[2-64]。为了提高整体效率和性能，将用户缓冲区人为的划分为若干段，让 DMA 分段传输整个数据序列，以便用户能够实时并发的处理。而每段的长度由 nSegmentSizeWords 参数决定。

nSegmentSizeWords 缓冲区各段的长度(字或点)。其取值范围应等于或小于板载 FIFO 的半满空间。而段数由 nSegmentCount 决定。

pADPara 设备对象参数结构PXI8996_PARA_AD的指针， 它的各成员值决定了设备上的AD对象的各种状态及工作方式，如AD采样通道、采样频率等。具体定义请参考PXI8996.h(.Bas或.Pas或.VI)驱动接口文件和本文档中的《硬件参数结构》章节。



DMA 缓冲区结构图

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。

备注：DMA是直接内存存取的意思，其英文定义为：Direct Memory Access。它的技术含义可以顾名思义，就是数据传输在设备和内存之间直接进行，无需要CPU的参与。该项技术的使用大大提高了数据实时采集和处理的效率。但是为了更好的配合这样好的机制，我们需要将用户缓冲区分段，比如分为 32 段，每段的长度等于FIFO半满长度 4096，因此可以定义一个两维数组。如：SHORT ADBuffer[32][4096]，即nSegmentCount=32，nSegmentSizeWords=4096，然后开始启动设备后，ADBuffer[0]首先被DMA占用，当传输完成后，hDmaEvent即被触发，用户即可处理ADBuffer[0]，而DMA接着占用ADBuffer[1]，当传输完成后，hDmaEvent即再次被触发，用户即可处理ADBuffer[1]，而DMA接着占用ADBuffer[2]，就这样依次类推。至到ADBuffer[31]被传输完后DMA再回到始端，占用ADBuffer[0]，就这样周而复始的进行下去。除了hDmaEvent事件对象可以通知用户何时处理数据外，其GetDevStatusDmaAD函数也可以实时返回DMA各种状态，如DMA正在占用的缓冲段ID（iCurSegmentID），整个缓冲链各个段的更新状态(bSegmentSts[])，整个缓冲链是否溢出(bBufferOverflow)等，跟踪这些信息，可以使数据转换、传输和处理之间有更大的时间弹性，高度保证数据的连续性。

切记：在InitDeviceDmaAD函数被调用之后若想再调用它改变硬件的某些参数，那么必须在ReleaseDeviceDmaAD之后方可调用。即InitDeviceDmaAD和ReleaseDeviceDmaAD必须成对调用，且在应用程序被关闭前必须确保已调用ReleaseDeviceDmaAD释放了各种DMA资源，否则可能会引起系统严重错误。

相关函数： CreateDevice InitDeviceDmaAD StartDeviceDmaAD
GetDevStatusDmaAD SetDevStatusDmaAD StopDeviceDmaAD

ReleaseDeviceDmaAD　ReleaseDevice

♦ **启动设备上的 AD 部件**

函数原型：

Visual C++ & C++ Builder:

```
BOOL StartDeviceDmaAD(HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceDmaAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StartDeviceDmaAD (hDevice : Integer) : Boolean;
                          StdCall; External 'PXI8996' Name ' StartDeviceDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：在InitDeviceDmaAD被成功调用之后，调用此函数即可启动设备上的AD部件，让设备开始AD采样。

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

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

相关函数： CreateDevice　InitDeviceDmaAD　StartDeviceDmaAD
GetDevStatusDmaAD　SetDevStatusDmaAD　StopDeviceDmaAD
ReleaseDeviceDmaAD　ReleaseDevice

♦ **取得 DMA 的状态标志**

Visual C++ & C++Builder:

```
BOOL GetDevStatusDmaAD ( HANDLE hDevice,
                         PPXI8996_STATUS_DMA pDMAStatus )
```

Visual Basic:

```
Declare Function GetDevStatusDmaAD Lib "PXI8996" (ByVal hDevice As Long,_
                                                  ByRef pDMAStatus As PXI8996_STATUS_DMA) As Boolean
```

Delphi:

```
Function GetDevStatusDmaAD (hDevice : Integer;
                            pDMAStatus : PPXI8996_STATUS_DMA ) : Boolean;
                            StdCall; External 'PXI8996' Name ' GetDevStatusDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceDmaAD后，应立即用此函数查询DMA的状态（当前段缓冲ID、缓冲段新旧标志、DMA缓冲溢出标志）。我们通常用缓冲段新旧标志bSegmentSts[x]去同步缓冲区数据处理操作。当bSegmentSts[x]标志为 1 时表示其该段为新数据段，则可以处理x段数据，然后再执行SetDevStatusDmaAD函数将x段新旧标志置为 0，表示已处理完，该段变为旧数据。

参数：

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

pDMAStatus它属于PXI8996_STATUS_DMA的结构体指针。该参数实时返回DMA的当前状态。关于PXI8996_STATUS_DMA具体定义请参考PXI8996.h(.Bas或.Pas或.VI)驱动接口文件以及本文档中的《DMA状态参数结构（PXI8996_STATUS_DMA）》。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。

相关函数： CreateDevice　InitDeviceDmaAD　StartDeviceDmaAD
GetDevStatusDmaAD　SetDevStatusDmaAD　StopDeviceDmaAD
ReleaseDeviceDmaAD　ReleaseDevice

♦ **取得 DMA 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevStatusDmaAD ( HANDLE hDevice,
                         LONG iClrBufferID )
```

Visual Basic:

```
Declare Function SetDevStatusDmaAD Lib "PXI8996" (ByVal hDevice As Long,_
                                                   ByVal iClrBufferID As Long) As Boolean
```

Delphi:

```
Function SetDevStatusDmaAD (hDevice : Integer;
                            iClrBufferID : LongInt ) : Boolean;
                            StdCall; External 'PXI8996' Name ' SetDevStatusDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：当处理完 DMA 缓冲链中的某一段数据后，应该立即调用此函数将其缓冲段状态标志清除，使其复位至 0，表示该数据已被处理过，已变成了旧数据，以便在下一个 DMA 事件响应下，不会重复自理某一缓冲段的数据。同时也避免产生 DMA 缓冲区溢出的可能。

参数：

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

iClrBufferID 要被清除标志的缓冲段ID。当指定的缓冲段状态标志清除后，则从GetDevStatusDmaAD函数返回的bSegmentSts[x]则会为 0。只有待到DMA事件下，其相应的缓冲段状态标志才会被置 1。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。

相关函数： CreateDevice InitDeviceDmaAD StartDeviceDmaAD GetDevStatusDmaAD SetDevStatusDmaAD StopDeviceDmaAD ReleaseDeviceDmaAD ReleaseDevice

♦ 暂停设备上的 AD 采样工作

函数原型：

Visual C++ & C++ Builder:

```
BOOL StopDeviceDmaAD(HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceDmaAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StopDeviceDmaAD (hDevice : Integer) : Boolean;
                          StdCall; External 'PXI8996' Name ' StopDeviceDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：在StartDeviceDmaAD被成功调用之后，用户可以在任何时候调用此函数停止AD采样(必须在ReleaseDeviceDmaAD之间被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceDmaAD，那么设备会接着停止前的状态(如通道位置)继续开始正常的AD数据转换。

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

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

相关函数： CreateDevice InitDeviceDmaAD StartDeviceDmaAD GetDevStatusDmaAD SetDevStatusDmaAD StopDeviceDmaAD ReleaseDeviceDmaAD ReleaseDevice

♦ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseDeviceDmaAD(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDeviceDmaAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDeviceDmaAD(hDevice : Integer) : Boolean;
                            StdCall; External 'PXI8996' Name ' ReleaseDeviceDmaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的AD部件，如果AD没有被StopDeviceDmaAD函数停止，则此函数在释放AD部件之前先

停止AD部件。

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

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

相关函数： CreateDevice　InitDeviceDmaAD　StartDeviceDmaAD
GetDevStatusDmaAD　SetDevStatusDmaAD　StopDeviceDmaAD
ReleaseDeviceDmaAD　ReleaseDevice

应注意的是，InitDeviceDmaAD必须和ReleaseDeviceDmaAD函数一一对应，即当您执行了一次InitDeviceDmaAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceDmaAD函数，以释放先前由InitDeviceDmaAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用InitDeviceDmaAD函数时，那些软硬件资源才可被再次使用。

♦ **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceDmaAD
④ StartDeviceDmaAD
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ GetDevStatusDmaAD
⑦ SetDevStatusDmaAD
⑧ StopDeviceDmaAD
⑨ ReleaseDeviceDmaAD
⑩ ReleaseSystemEvent（公共函数）
⑩ ReleaseDevice

注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。

关于这个过程的图形说明请参考《使用纲要》。

注意：若成功初始化 DMA 后，要退出整个应用程序，切记应先释放 DMA 才能退出。

第五节、AD 中断方式采样操作函数原型说

♦ **初始化设备上的 AD 对象**

函数原型：

Visual C++ & C++ Builder:

```
BOOL InitDeviceIntAD(HANDLE hDevice,
                     HANDLE hEvent,
                     ULONG nFifoHalfLength,
                     PPXI8996_PARA_AD pPara)
```

Visual Basic:

```
Declare Function InitDeviceIntAD Lib "PXI8996" (ByVal hDevice As Long, _
                                                ByVal hEvent As Long, _
                                                ByVal nFifoHalfLength As Long,_
                                                ByRef pPara As PXI8996_PARA_AD ) As Boolean
```

Delphi:

```
Function InitDeviceIntAD (hDevice : Integer;
                          hEvent : Integer;
                          nFifoHalfLength : Longword;
                          pPara : PPXI8996_PARA_AD) : Boolean;
                          StdCall; External 'PXI8996' Name ' InitDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备操作就绪有关工作，如预置AD采集通道，采样频率等。且让设备上的AD部件以硬件中断的方式工作，其中断源信号由FIFO芯片半满管脚提供。但它并不启动AD采样，那么需要在此函数被成功调用之后，再调用StartDeviceIntAD函数即可启动AD采样。

参数：

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

hEvent 中断事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的

内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEvent事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。其具体实现方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》。

nFifoHalfLength 告诉设备对象，FIFO存储器半满长度大小。该参数很关键，因为不仅决定了设备对象每次产生半满中断时应读入AD数据的点数，同时，它也决定了一级缓冲队列中每个元素对应的缓冲区大小。比如，nFifoHalfLength等于 2048，则设备对象在系统空间中建立具有 64 个元素，且每个元素对应于 2048 个字长且物理连续的一级缓冲队列。但是该参数可以根据用户特殊需要，将其置成小于FIFO存储器实际的半满长度的值。比如用户要求在频率一定的情况下，提高FIFO半满中断事件的频率等，那么可以将此参数置成小于FIFO半满长度的值，但是绝不能大小半满长度。在工作期间，此队列的维护和管理完全由设备对象管理，与用户无关，用户只需要用ReadDeviceIntAD函数简单地读取AD数据，并注意检查其返回值即可。

pPara 设备对象参数结构指针， 它的各成员值决定了设备上的AD对象的各种状态及工作方式，如AD采样通道、采样频率等。请参考《硬件参数结构》章节。

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　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦ 启动设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL StartDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceIntAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StartDeviceIntAD (hDevice : Integer) : Boolean;
                    StdCall; External 'PXI8996' Name ' StartDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：在InitDeviceIntAD被成功调用之后，调用此函数即可启动设备上的AD部件，让设备开始AD采样。

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

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

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦读取 PCI 设备上的 AD 数据

Visual C++ & C++Builder:

```
DWORD ReadDeviceIntAD (HANDLE hDevice,
                       PULONG pADBuffer,
                       LONG nReadSizeWords,
                       PLONG nRetSizeWords )
```

Visual Basic:

```
Declare Function ReadDeviceIntAD Lib "PXI8996" (ByVal hDevice As Long,_
                                                ByRef pADBuffer As Long,_
                                                ByVal nReadSizeWords As Long,_
                                                ByRef nRetSizeWords As Long) As Long
```

Delphi:

```
Function ReadDeviceIntAD (hDevice : Integer;
                          pADBuffer : Pointer;
```

```
                                nReadSizeWords : LongInt;
                                nRetSizeWords : Pointer) : Longword;
                                StdCall; External 'PXI8996' Name ' ReadDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceIntAD后，应立即用WaitForSingleObject等待中断事件hIntEvent的发生，如果FIFO还没有达到半满状态，即中断事件还发生，则数据采集线程WaitForSingleObject的作用下自动进入睡眠状态(此状态下，数据采集线程或代码不消耗CPU时间)。当中断事件发生时线程被突发唤醒，即在WaitForSingleObject后的代码将被立即得到执行，因此为了提高数据吞吐率，在WaitForSingleObject之后，应紧接着用ReadDeviceIntAD函数读取FIFO半满数据。注意看演示程序如何处理这个问题。

参数：

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

pADBuffer接受AD数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。关于如何将缓冲区中的这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次ReadDeviceIntAD操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区pADBuffer的最大空间长度，且由于是半满读取数据，所以这个参数必须等于板上FIFO存储器总容量的二分之一，比如FIFO为 1K长度（即 1024 点），则此参数应为 512，若为 4K（即 4096 点）长度，则此参数应为 2048，其他情况以此类推。当然特殊情况下，比如用户不要求数据连续或不担心丢点问题，则可以将此参数设得比FIFO存储器的半满长度小。需要用户特别注意的是此参数必须与InitDeviceIntAD函数中的nFifoHalfLength参数相等，才能实现连续数据采集。如果大于nFifoHalfLength，此会造成缓冲访问异常，严重时可能会使整过Windows系统崩溃，如果小于nFifoHalfLength，则会丢失n个点的数据在一级缓冲内(n为nFifoHalfLength 减去nReadSizeWords的差值)。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值：如果失败，即一级缓冲队列溢出则返回 0xe1000000 码，如果成功，则返回一级缓冲队列中的未被ReadDeviceIntAD读空的缓冲队列元素数量。一个元素对应于一个由InitDeviceIntAD函数的nFifoHalfLength参数指定大小的系统物理缓冲区。

注释：由于设备对象在系统空间中维护两个当前指针，且这两个指针最初都指向缓冲队列中的第一个元素。为了便于说明，我们将这两个指针分别命名为：用户指针和系统指针。当每执行此函数一次，设备对象将用户指针指向的缓冲区中的数据映射到用户空间pADBuffer中，且将用户指针下移一个元素位置。而系统指针则不随用户的操作而改变。它是设备对象强制自动维护的指针。它的改变速度只与AD数据转换有关。可见，不管用户有没有读走当前指针指向的一级缓冲区中的数据，或者整个Windows系统有多忙，但这个系统指针每到一个半满状态时，它总会自动下移一个元素位置。设备对象根据某些状态信息和利用巧妙算法，统计出已经采集了但用户迟迟没有读走的缓冲区数量，这个数量便是ReadDeviceIntAD返回的正确值，且判断数据是否重叠或溢出，这个状态便是ReadDeviceIntAD返回的 0xe1000000 码。如果用户的处理速度与得到设备对象的传输速度一样，那么ReadDeviceIntAD的返回值应等于 0，如果某一次在WaitForSingleObject之后执行ReadDeviceIntAD所返回的值不为 0，且不为 0xe1000000，假如是 5，则视为一级缓冲区中的数据已有 5 个元素指向的数据是已采集的新数据，那么用户应接着用循环语句连读 5 次数据，直到ReadDeviceIntAD返回 0 为止。其他情况以此类推。此种方案的使用，让用户即便是在高速采集数据时，也能在很大程度上象往常一样随意进行窗口菜单等突发操作。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD　ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD　ReleaseDevice

♦ 暂停设备上的 AD 采样工作

函数原型：

Visual C++ & C++ Builder:

```
BOOL StopDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceIntAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StopDeviceIntAD (hDevice : Integer) : Boolean;
                                StdCall; External 'PXI8996' Name ' StopDeviceIntAD ';
```

LabVIEW:

请参考相关演示程序。

功能：在StartDeviceIntAD被成功调用之后，用户可以在任何时候调用此函数停止AD采样(必须在ReleaseDeviceDmaAD之间被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceDmaAD，那么设备会接着停止前的状态(如通道位置)继续开始正常的AD数据转换。

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

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

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD　ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD　ReleaseDevice

◆ **释放设备上的 AD 部件**

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDeviceIntAD Lib "PXI8996" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDeviceIntAD (hDevice : Integer) : Boolean;
                                    StdCall; External 'PXI8996' Name ' ReleaseDeviceIntAD ';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的AD部件，如果AD没有被StopDeviceIntAD函数停止，则此函数在释放AD部件之前先停止AD部件。

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

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

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD　ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD　ReleaseDevice

应注意的是，InitDeviceIntAD必须和ReleaseDeviceIntAD函数一一对应，即当您执行了一次InitDeviceIntAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceIntAD函数，以释放先前由InitDeviceIntAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用InitDeviceIntAD函数时，那些软硬件资源才可被再次使用。

◆ **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceIntAD
④ StartDeviceIntAD
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ ReadDeviceIntAD
⑦ StopDeviceIntAD
⑧ ReleaseDeviceDmaAD
⑨ ReleaseSystemEvent（公共函数）
⑩ ReleaseDevice

注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。

关于这个过程的图形说明请参考《使用纲要》。

第六节、AD 硬件参数保存与读取函数原型说明

◆ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++ & C++ Builder:

```
BOOL LoadParaAD(HANDLE hDevice,
```

PPXI8996_PARA_AD pADPara)

Visual Basic:

Declare Function LoadParaAD Lib "PXI8996" (ByVal hDevice As Long, _
ByRef pADPara As PXI8996_PARA_AD) As Boolean

Delphi:

Function LoadParaAD (hDevice : Integer;
pADPara : PPXI8996_PARA_AD) : Boolean;
StdCall; External 'PXI8996' Name ' LoadParaAD ';

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。

pADPara属于PPXI8996_PARA_AD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PPXI8996_PARA_AD请参考PXI8996.h或PXI8996.Bas或PXI8996.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++ & C++ Builder:

BOOL SaveParaAD (HANDLE hDevice,
PPXI8996_PARA_AD pADPara)

Visual Basic:

Declare Function SaveParaAD Lib "PXI8996" (ByVal hDevice As Long, _
ByRef pADPara As PXI8996_PARA_AD) As Boolean

Delphi:

Function SaveParaAD (hDevice : Integer;
pADPara : PPXI8996_PARA_AD) : Boolean;
StdCall; External 'PXI8996' Name ' SaveParaAD ';

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。

pADPara 设备硬件参数，关于PXI8996_PARA_AD的详细介绍请参考PXI8996.h或PXI8996.Bas或PXI8996.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

Viusal C++ & C++ Builder:

BOOL ResetParaAD (HANDLE hDevice,
PPXI8996_PARA_AD pADPara)

Visual Basic:

Declare Function ResetParaAD Lib "PXI8996" (ByVal hDevice As Long, _
ByRef pADPara As PXI8996_PARA_AD) As Boolean

Delphi:

Function ResetParaAD (hDevice : Integer;
pADPara : PPXI8996_PARA_AD) : Boolean;

StdCall; External 'PXI8996' Name ' ResetParaAD ';

LabVIEW:

请参考相关演示程序。

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

参数：

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

pADPara设备硬件参数，它负责在参数被复位后返回其复位后的值。关于PXI8996_PARA_AD的详细介绍请参考PXI8996.h或PXI8996.Bas或PXI8996.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

第四章 硬件参数结构

第一节、AD 硬件参数结构（PXI8996_PARA_AD）

Visual C++ & C++Builder：

```
typedef struct _PXI8996_PARA_AD
{
    LONG bChannelArray[8]; // 采样通道选择阵列，分别控制 8 个通道，=TRUE 表示该通道采样，否则不采样
    LONG InputRange;          // 模拟量输入量程选择
    LONG MultiFrequency;      // 主时钟与采样时钟的倍频模式
    LONG Frequency;           // 采集频率，  单位为 Hz
    LONG TriggerMode;         // 触发模式选择
    LONG TriggerSource;       // 触发源选择
    LONG TriggerType;         // 触发类型选择(边沿触发/脉冲触发)
    LONG TriggerDir;          // 触发方向选择(正向/负向触发)
    LONG ClockSource;         // 时钟源选择(内/外时钟源)
    LONG bTrigOutput;         // 是否将触发信号输出到 PXI 总线，=TRUE:允许输出，=FALSE:禁止输出
}PXI8996_PARA_AD, *PPXI8996_PARA_AD;
```

Visual Basic：

```
Private Type PXI8996_PARA_AD
    bChannelArray(0 to 7) As Long       ' 采样通道选择阵列，分别控制 8 个通道
    InputRange As Long                  ' 模拟量输入量程选择
    MultiFrequency As Long              ' 主时钟与采样时钟的倍频模式
    Frequency As Long                   ' 采集频率，单位为 Hz
    TriggerMode As Long                 ' 触发模式选择
    TriggerSource As Long               ' 触发源选择
    TriggerType As Long                 ' 触发类型选择(边沿触发/脉冲触发)
    TriggerDir As Long                  ' 触发方向选择(正向/负向触发)
    ClockSource As Long                 ' 时钟源选择(内/外时钟源)
    bTrigOutput As Long                 ' 是否将触发信号输出到 PXI 总线
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
  PPXI8996_PARA_AD = ^ PXI8996_PARA_AD;    // 指针类型结构
  PXI8996_PARA_AD = record                 // 标记为记录型
```

```
    bChannelArray : Array[0…7] of LongInt;          //采样通道选择阵列，分别控制 8 个通道
    InputRange : LongInt;                  // 模拟量输入量程选择
    MultiFrequency : LongInt;              // 主时钟与采样时钟的倍频模式
    Frequency : LongInt;                   // 采集频率，单位为 Hz
    TriggerMode : LongInt;                 // 触发模式选择
    TriggerSource : LongInt;               // 触发源选择
    TriggerType : LongInt;                 // 触发类型选择(边沿触发/脉冲触发)
    TriggerDir : LongInt;                  // 触发方向选择(正向/负向触发)
    ClockSource : LongInt;                 // 时钟源选择(内/外时钟源)
    bTrigOutput : LongInt;                 // 是否将触发信号输出到 PXI 总线
End;
```

LabVIEW:
请参考相关演示程序。

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

bChannelArray 采样通道选择阵列，分别控制 8 个通道，=TRUE 表示该通道采样，否则不采样。

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义
PXI8996_INPUT_N1000_P1000mV	0x00	±1000mV
PXI8996_INPUT_N10000_P10000mV	0x01	±10000mV

MultiFrequency 主时钟与采样时钟的倍频模式选择。

常量名	常量值	功能定义
PXI8996_MULTIFRE_256	0x00	256 倍
PXI8996_MULTIFRE_128	0x01	128 倍
PXI8996_MULTIFRE_64	0x02	64 倍

Frequency　AD 采样频率，本设备的频率取值范围为[1Hz, 192KHz]。

TriggerMode　AD 触发模式。

常量名	常量值	功能定义
PXI8996_TRIGMODE_SOFT	0x00	软件触发(属于内触发)
PXI8996_TRIGMODE_POST	0x01	硬件后触发(属于外触发)

TriggerSource AD 触发源选择。

常量名	常量值	功能定义
PXI8996_TRIGSRC_DTR	0x00	选择外部 DTR 触发源
PXI8996_TRIGSRC_PXI_TRIG0	0x01	选择 PXI 总线上的 TRIG0 触发源
PXI8996_TRIGSRC_PXI_TRIG1	0x02	选择 PXI 总线上的 TRIG1 触发源
PXI8996_TRIGSRC_PXI_TRIG7	0x03	选择 PXI 总线上的 TRIG7 触发源
PXI8996_TRIGSRC_PXI_STAR	0x04	选择 PXI 总线上的 STAR 触发源

TriggerType　AD 触发类型。

常量名	常量值	功能定义
PXI8996_TRIGTYPE_EDGE	0x00	边沿触发
PXI8996_TRIGTYPE_PULSE	0x01	脉冲触发(电平)

TriggerDir　AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义

PXI8996_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
PXI8996_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
PXI8996_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：PXI8996_TRIGDIR_POSIT_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。而在电平类型下，无论正电平还是负电平均触发。

ClockSource AD 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PXI8996_CLOCKSRC_IN	0x00	内部时钟
PXI8996_CLOCKSRC_OUT	0x01	外部时钟

bTrigOutput 是否将触发信号输出到 PXI 总线，=TRUE:允许输出， =FALSE:禁止输出。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第二节、AD 状态参数结构（PXI8996_STATUS_AD）

Visual C++ & C++Builder：

```
typedef struct _PXI8996_STATUS_AD
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bDynamic_Overflow;
    LONG bStatic_Overflow;
    LONG bConverting;
    LONG bTriggerFlag;
    } PXI8996_STATUS_AD, *PPXI8996_STATUS_AD;
```

Visual Basic：

```
Private Type    PXI8996_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bDynamic_Overflow As Long
    bStatic_Overflow As Long
    bConverting As Long
    bTriggerFlag As Long
End Type
```

Delphi:

```
Type    // 定义结构体数据类型
        PPXI8996_STATUS_AD = ^ PXI8996_STATUS_AD;   // 指针类型结构
        PXI8996_STATUS_AD = record          // 标记为记录型
        bNotEmpty : LongInt;
        bHalf : LongInt;
        bDynamic_Overflow : LongInt;
        bStatic_Overflow : LongInt;
        bConverting : LongInt;
        bTriggerFlag : LongInt;
End;
```

LabVIEW:

请参考相关演示程序。

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

bNotEmpty　AD 板载存储器 FIFO 的非空标志，=TRUE 表示存储器处在非空状态，即有可读数据，否则表示空。

bHalf　AD 板载存储器 FIFO 的半满标志，=TRUE 表示存储器处在半满状态，即有至少有半满以上数据可读，否则表示在半满以下，可能有小于半满的数据可读。

bDynamic_Overflow　AD 板载存储器 FIFO 的溢出标志，=TRUE 表示存储器处在全满或溢出状态，即全满的数据可读数据，但此时的数据很有可能已有丢点现象。否则表示满以下状态。该状态处于动态溢出状态，即 FIFO 随时溢出，它随时=TRUE，而随时不溢出，则随时=FALSE。

bStatic_Overflow　AD 板载存储器 FIFO 的溢出标志，=TRUE 表示存储器至少有过一次出现全满或溢出状态，然后永远为 TRUE，除非用户重新开始采集数据则会自动变为 FALSE。在启动采集过程中，只有一次全满或溢出状态都未发生过，则此标志恒等于 FALSE。所以用此标志可以确定在整过采集中是否有过溢出丢点现象。当然要避免丢点现象的发生，您需要考虑应用软件设计的合理性、效率性等各方因素，我们提供的高级演示程序（尤其是 VC）便很好的展示了此类思想。

bConverting　AD 是否被实际启动，=TRUE 表示已被用户启动，=FALSE 表示用户还没有启动设备。

bTriggerFlag　AD 触发标志，=TRUE 表示已被触发(即产生触发事件)，=FALSE 表示未产生触发事件。

相关函数：　CreateDevice　GetDevStatusProAD　ReleaseDevice

第三节、DMA 状态参数结构（PXI8996_STATUS_DMA）

```
const int MAX_SEGMENT_COUNT = 64;
```

Visual C++ & C++Builder：

```
typedef struct _PXI8996_STATUS_DMA
{
    LONG iCurSegmentID; //  当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    LONG bSegmentSts[MAX_SEGMENT_COUNT];
    LONG bBufferOverflow;    //  返回溢出状态
} PXI8996_STATUS_DMA, *PPXI8996_STATUS_DMA;
```

Visual Basic：

```
Private Type   PXI8996_STATUS_DMA
    iCurSegmentID As Long              ' 当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    bSegmentSts(MAX_SEGMENT_COUNT) As Long
    bBufferOverflow As Long          ' 返回溢出状态
End Type
```

Delphi:

```
Type   //  定义结构体数据类型
    P PXI8996_STATUS_DMA = ^ PXI8996_STATUS_DMA;   //  指针类型结构
    PXI8996_STATUS_DMA = record          //  标记为记录型
    iCurSegmentID : LongInt;    //  当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    bSegmentSts [MAX_SEGMENT_COUNT] : Array [0…63] of LongInt;
    bBufferOverflow : LongInt;   //  返回溢出状态
End;
```

LabVIEW:

请参考相关演示程序。

此结构体主要用于DMA传输时的状态监控，GetDevStatusDmaAD函数使用此结构体来实时取得DMA状态，以便同步各种数据处理过程。

iCurSegmentID　DMA正在传输的当前缓冲段ID号。该ID号返回值的最大范围为 0 至 127，但其具体的返回值范围为InitDeviceDmaAD中的nSegmentCount参数决定，它的返回值为 0 至nSegmentCount-1。注意，每次调用InitDeviceDmaAD初始化设备后，其值自动被复位至 0。

bSegmentSts[ ]　DMA缓冲区各段的状态。 如bSegmentSts[0]=0，表示缓冲区段 0 此时为旧数据段，若=1 则段 0 为新数据段，可以对其进行数据处理。同理，bSegmentSts[1]=0，表示缓冲区段 1 此时为旧数据段，若=1 则

段 1 为新数据段，可以对其进行数据处理。注意，每次调用InitDeviceDmaAD初始化设备后，其值自动被复位至 0。

bBufferOverflow 组缓冲区溢出标志。若等于 0，则表示整个DMA缓冲链未发生溢出，若等于 1，则表示整个DMA缓冲链已发生溢出。注意，每次调用InitDeviceDmaAD初始化设备后，其值自动被复位至 0。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/16777216) * ((ADBuffer[0]^0x800000) &0xFFFFFF) – 10000.00	[-10000, +9999.99]
±1000mV	Volt = (2000.00/16777216) * ((ADBuffer[0]^0x800000) &0xFFFFFF) – 1000.00	[-1000, +999.99]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++&C++Builder：

```
Lsb = (ADBuffer[0] ^ 0x800000)&0x0FFFFFF;
Volt = (20000.00/16777216) * Lsb -10000.00;
```

Visual Basic：

```
Lsb = (ADBuffer [0] Xor &H800000) And &HFFFFFF
Volt = (20000.00/16777216) * Lsb – 10000.00
```

Delphi:

```
Lsb：  = (ADBuffer[0] Xor $800000) And $FFFFFF;
Volt：= (200000.0/16777216) * Lsb – 10000.00;
```

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，当通道总数首末通道相等时，假如此时首末通道=5，其排放规则如下：

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

两通道采集(假如FirstChannel=0, LastChannel=1)：

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

四通道采集(假如FirstChannel=0, LastChannel=3)：

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

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细

地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效呢？还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

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

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	…
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	…
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	…
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;        // 文件头信息长度
    LONG FileType;
    // 该设备数据文件共有的成员
    LONG BusType;              // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;            // 该设备的编号(DEFAULT_DEVICE_NUM)

    LONG VoltBottomRange;      // 量程下限(mV)
    LONG VoltTopRange;         // 量程上限(mV)

    PXI8996_PARA_AD ADPara;    // 保存硬件参数

    LONG CrystalFreq;          // 晶振频率
    LONG HeadEndFlag;          // 头信息结束位
} FILE_HEADER,  *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第六章 上层用户函数接口应用实例

第一节、怎样使用ReadDeviceProAD_Npt函数直接取得AD数据

Visual C++ & C++Builder:

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[PXI8996 16 路同步 AD 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[AD 非空方式]

第二节、怎样使用ReadDeviceProAD_Half函数直接取得AD数据

Visual C++ & C++Builder:

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[PXI8996 16 路同步 AD 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[AD 半满方式]

第三节、怎样使用 DMA 方式取得 AD 数据

Visual C++ & C++Builder:

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[PXI8996 16 路同步 AD 卡] [Microsoft Visual C++]丨[简易代码演示]丨[AD DMA 方式]

第七章 高速大容量、连续不间断数据采集及存盘技术详解

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

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

可能用户还要问，既然数据处理线程是非工作者线程，那么如果用户移动窗口等操作堵塞了该线程，而数据采集线程则在不停地采集数据，那数据处理线程难道不会因此而丢失采集线程发来的某一段数据吗？如果不另加处理，这个情况肯定有发生的可能。但是，我们采用了一级缓冲队列和二级缓冲队列的设计方案，足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K 数据，那么我们就创建一个缓冲队列，在用户程序中最简单的办法就是开辟一个两维数组如 ADBuffer [SegmentCount][SegmentSize]， 我们将 SegmentSize 视为数据采集线程每次采集的数据长度，SegmentCount 则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32，则这个缓冲队列实际上就是数组 ADBuffer [32][8192]的形式。那么如

何使用这个缓冲队列呢？方法很简单，它跟一个普通的缓冲区如一维数组差不多，唯一不同是，两个线程首先要通过改变 SegmentCount 字段的值，即这个下标 Index 的值来填充和引用由 Index 下标指向某一段 SegmentSize 长度的数据缓冲区。需要注意的是两个线程不共用一个 Index 下标变量。具体情况是当数据采集线程在 AD 部件被 InitDeviceProAD 或 InitDeviceDmaAD 初始化之后，首次采集数据时，则将自己的 ReadIndex 下标置为 0，即用第一个缓冲区采集 AD 数据。当采集完后，则向数据处理线程发送消息，且两个线程的公共变量 SegmentCount 加 1，（注意 SegmentCount 变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了，但是却没被数据处理线程处理掉的缓冲区数量。）然后再接着将 ReadIndex 偏移至 1，再用第二个缓冲区采集数据。再将 SegmentCount 加 1，直到 ReadIndex 等于 31 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从 SegmentCount 变量中减去在所接受到的当前事件下所处理的缓冲区个数，具体处理哪个缓冲区由 CurrentIndex 指向。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对 SegmentCount 加以判断，观察其值是否大于了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。

图 7.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程填充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

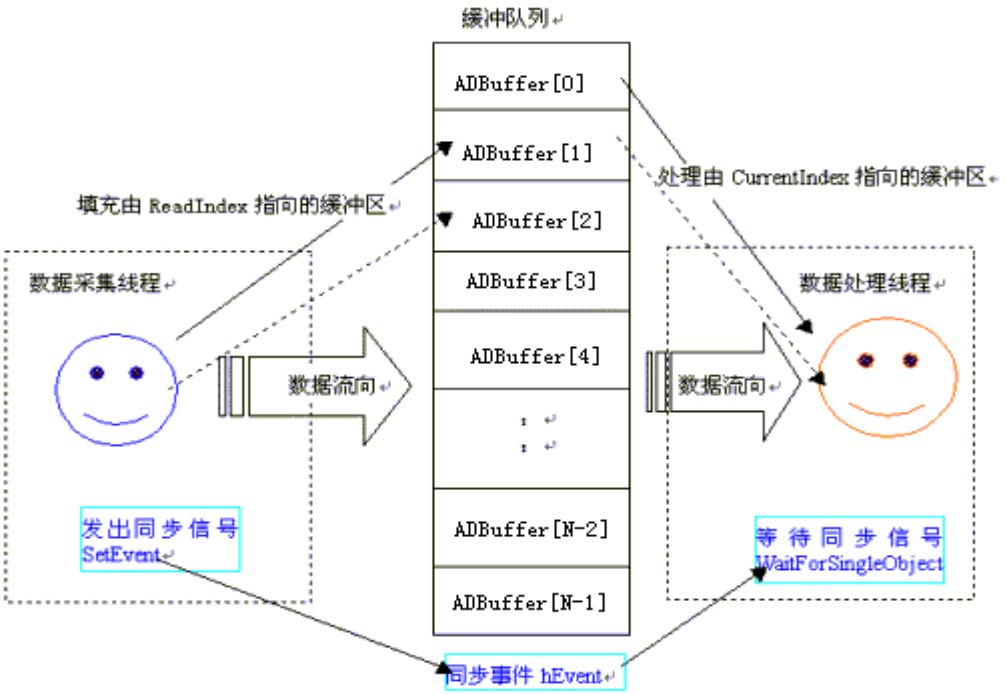


图 8.1

第一节、使用程序查询方式实现该功能

下面用 Visual C++程序举例说明。

一、使用ReadDeviceProAD_Npt函数读取设备上的AD数据（它使用FIFO的非空标志）

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

[程序]丨[阿尔泰测控演示系统]丨[PXI8996 24 位 8 路 AD 卡]丨[Microsoft Visual C++]丨[高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

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

BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD() // 终止采集函数

二、使用ReadDeviceProAD_Half函数读取设备上的AD数据（它使用FIFO的半满标志）

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

[程序]\|[阿尔泰测控演示系统]\|[PXI8996 24 位 8 路 AD 卡]\|[Microsoft Visual C++]\|[高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

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

当然用 FIFO 非空标志读取 AD 数据，能获得接近 FIFO 总容量的栈深度，这样用户在两批数据之间，便有更多的时间来处理某些数据。而用半满标志，则最多只能达到 FIFO 总容量的二分之一的栈深度，那么用户在两批数据之间处理数据的时间会相对短些，但是半满读取时，查询 AD 转换标志的时间则最少。当然究意那种方案最好，还得看用户的实际需要。

第二节、使用 DMA 方式实现该功能

DMA 方式是利用直接内存存取技术实现的数据传输技术，它基本上不占用 CPU 时间就可能很快的将数据从设备读到用户缓冲区中。所以利用 DMA 方式采集数据，其吞吐率要比程序方式高很多。

需要注意的是，由于DMA方式采用了多缓冲级链的方式，因此每次接受到DMA事件后，一定要注意GetDevStatusDmaAD函数返回的缓冲区状态，必须在该次事件之下，探测所有缓冲区段状态是否为新标志 1，直至所有标志为旧标志 0 后才能允许程序再去接管下一次DMA事件。

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

[程序]\|[阿尔泰测控演示系统]\|[PXI8996 24 位 8 路 AD 卡]\|[Microsoft Visual C++]\|[高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

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

第八章 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“PXI8996_”）

函数名	函数功能	备注
① PXI 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定 PXI 设备寄存器操作基地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户

ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateVBThread	在 VB 环境中建立子线程对象	在 VB 中可实现多线程
TerminateVBThread	终止 VB 的子线程	
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	
ReadFile	请求文件对象读数据到用户空间	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	
ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备
⑤ 各种参数保存和读取函数		
SaveParaInt	保存整型参数到注册表	
LoadParaInt	从注册表中读取整型参数值	
SaveParaString	保存字符参数到注册表	
LoadParaString	从注册表中读取字符参数值	
⑥ 其他函数		
kbhit	探测用户是否有击键动作	
getch	等待并获取用户击键值	
GetLastErrorEx	取得驱动函数错误信息	

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++ & C++ Builder:

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

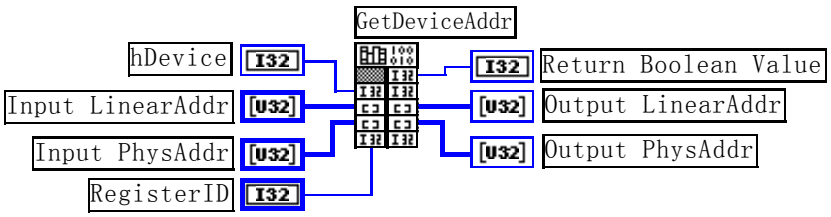
Visual Basic:

```
Declare Function GetDeviceAddr Lib "PXI8996" (ByVal hDevice As Long, _
                                              ByRef LinearAddr As Long, _
                                              ByRef PhysAddr As Long, _
                                              Optional ByVal RegisterID As Integer = 0) As Boolean
```

Delphi:

```
Function GetDeviceAddr(hDevice : Integer;
                    LinearAddr : Pointer;
                    PhysAddr : Pointer;
                    RegisterID : Integer = 0) : Boolean;
                    StdCall; External 'PXI8996' Name ' GetDeviceAddr ';
```

LabVIEW:



功能：取得 PXI 设备指定的内存映射寄存器的线性地址。

参数：

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

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

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

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

常量名	常量值	功能定义
PXI8996_REG_MEM_CPLD	0x0000	0 号寄存器对应板上控制单元所使用的内存模式基地址(使用 LinearAddr)
PXI8996_REG_IO_CPLD	0x0001	1 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr)

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

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

Visual C++ & C++ Builder 程序举例:

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

Visual Basic 程序举例:

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

◆ **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++ & C++ Builder:

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

BYTE Value)

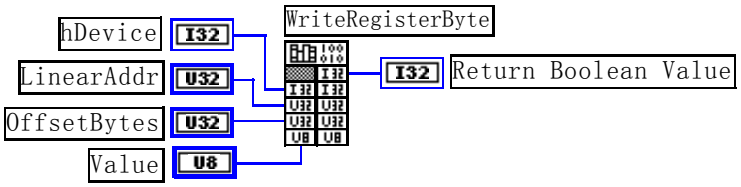
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PXI8996" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte ) As Boolean
```

Delphi:

```
Function WriteRegisterByte(  hDevice : Integer;
                             LinearAddr : LongWord;
                             OffsetBytes : LongWord;
                             Value : Byte) : Boolean;
                             StdCall; External 'PXI8996' Name ' WriteRegisterByte ';
```

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或CreateDeviceEx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WriteRegisterByte函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

Visual C++ & C++ Builder 程序举例:

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

Visual Basic 程序举例:

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

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

函数原型：

Visual C++ & C++ Builder:

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

```
                                ULONG OffsetBytes,
                                WORD Value)
```

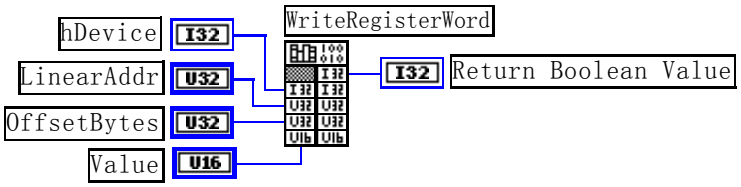
Visual Basic:

```
Declare Function WriteRegisterWord Lib "PXI8996" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WriteRegisterWord( hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : Word) : Boolean;
                            StdCall; External 'PXI8996' Name ' WriteRegisterWord ';
```

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或CreateDeviceEx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 GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

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

Visual Basic 程序举例:

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

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

函数原型：

Visual C++ & C++ Builder:

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

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

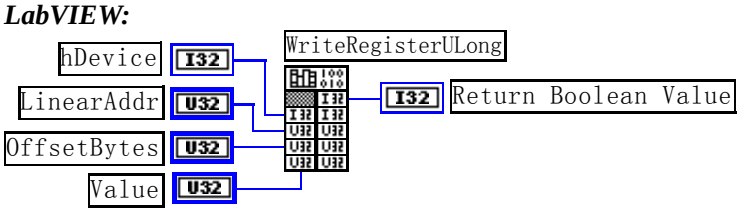
Visual Basic:

```
Declare Function WriteRegisterULong Lib "PXI8996" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long) As Boolean
```

Delphi:

```
Function WriteRegisterULong(hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : LongWord) : Boolean;
                            StdCall; External 'PXI8996' Name ' WriteRegisterULong ';
```

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或CreateDeviceEx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	GetDeviceAddr	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ & C++ Builder 程序举例:

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

Visual Basic 程序举例:

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

◆ **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

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

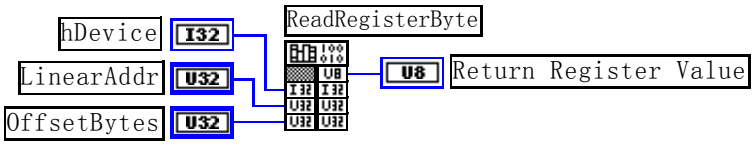
Visual Basic:

```
Declare Function ReadRegisterByte Lib "PXI8996" (ByVal hDevice As Long, _
                                                 ByVal LinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

Delphi:

```
Function ReadRegisterByte(hDevice : Integer;
                          LinearAddr : LongWord;
                          OffsetBytes : LongWord) : Byte;
                          StdCall; External 'PXI8996' Name ' ReadRegisterByte ';
```

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或CreateDeviceEx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　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,    PhysAddr, OffsetBytes;
BYTE    Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);     // 取得 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
    :
```

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 PXI 内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

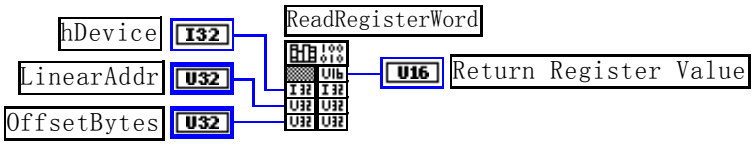
```
Declare Function ReadRegisterWord Lib "PXI8996" (  ByVal hDevice As Long, _
```

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

Delphi:

```
Function ReadRegisteWord(hDevice : Integer;
                         LinearAddr : LongWord;
                         OffsetBytes : LongWord) : Word;
                         StdCall; External 'PXI8996' Name ' ReadRegisterWord ';
```

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或CreateDeviceEx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 GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 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
    :
```

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 PXI 内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

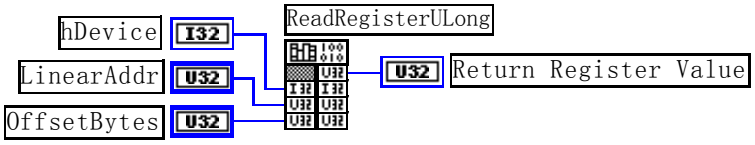
```
Declare Function ReadRegisterULong Lib "PXI8996" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```

Delphi:

```
Function ReadRegisterULong(hDevice : Integer;
                           LinearAddr : LongWord;
```

```
                                OffsetBytes : LongWord) : LongWord;
                                StdCall; External 'PXI8996' Name ' ReadRegisterULong ';
```

LabVIEW:



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或CreateDeviceEx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 GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
ULONG   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 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++ & C++ Builder:

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

Visual Basic:

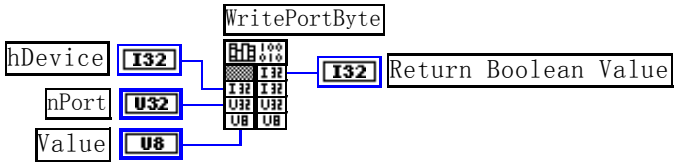
```
Declare Function WritePortByte Lib "PXI8996" ( ByVal hDevice As Long, _
                                                ByVal nPort As Long, _
                                                ByVal Value As Byte) As Boolean
```

Delphi:

```
Function WritePortByte(hDevice : Integer;
                       nPort : LongWord;
```

Value : Byte) : Boolean;
StdCall; External 'PXI8996' Name ' WritePortByte ';

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

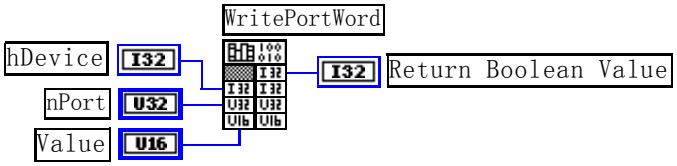
Visual Basic:

```
Declare Function WritePortWord Lib "PXI8996" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WritePortWord(hDevice : Integer;
                       nPort : LongWord;
                       Value : Word) : Boolean;
                       StdCall; External 'PXI8996' Name ' WritePortWord ';
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

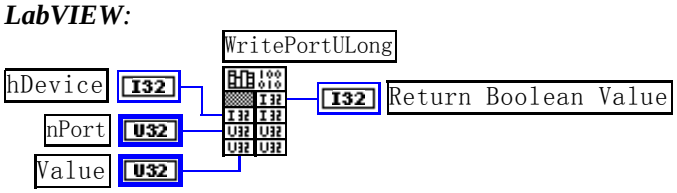
```
Declare Function WritePortULong Lib "PXI8996" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
```

ByVal Value As Long) As Boolean

Delphi:

```
Function WritePortULong(hDevice : Integer;
                        nPort : LongWord;
                        Value : LongWord) : Boolean;
                        StdCall; External 'PXI8996' Name ' WritePortULong ';
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。

参数：

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

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

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

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

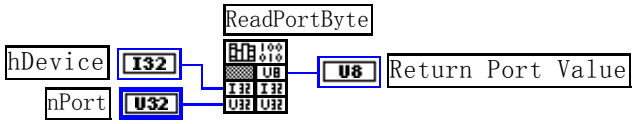
Visual Basic:

```
Declare Function ReadPortByte Lib "PXI8996" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

```
Function ReadPortByte(hDevice : Integer;
                      nPort : LongWord) : Byte;
                      StdCall; External 'PXI8996' Name ' ReadPortByte ';
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。

参数：

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

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

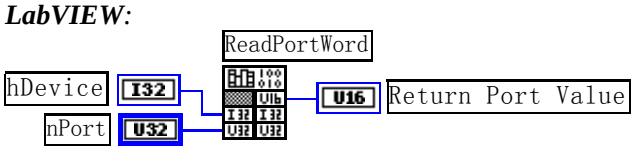
```
Declare Function ReadPortWord Lib "PXI8996" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

Delphi:

```
Function ReadPortWord(hDevice : Integer;
```

nPort : LongWord) : Word;
StdCall; External 'PXI8996' Name ' ReadPortWord ';

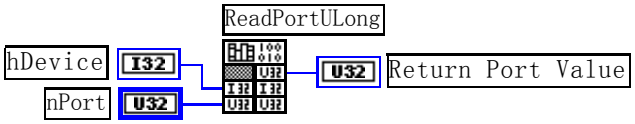
LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nPort　设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

♦ **以四字节(32Bit)方式读 I/O 端口**

函数原型：
Visual C++ & C++ Builder:
ULONG ReadPortULong(HANDLE hDevice,
UINT nPort)
Visual Basic:
Declare Function ReadPortULong Lib "PXI8996" (ByVal hDevice As Long, _
ByVal nPort As Long) As Long
Delphi:
Function ReadPortULong(hDevice : Integer;
nPort : LongWord) : LongWord;
StdCall; External 'PXI8996' Name ' ReadPortULong ';
LabVIEW:



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

第四节、线程操作函数原型说明

（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）

♦ **在 VB 环境中，创建子线程对象，以实现多线程操作**

函数原型：
Visual C++ & C++ Builder:
BOOL CreateVBThread(HANDLE *hThread,
LPTHREAD_START_ROUTINE StartThread);
Visual Basic:
Declare Function CreateVBThread Lib "PXI8996" (ByRef hThread As Long, _
ByVal StartThread As Long) As Boolean

功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题。通过该函数用户可以很轻松地实现多线程操作。
参数：

hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程、暂停线程以及删除线程等。

StartThread作为子线程运行的函数的地址，在实际使用时，请用AddressOf关键字取得该子线程函数的地址，再传递给CreateVBThread函数。

返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

注意: RoutineAddr 指向的函数或过程必须放在 VB 的模块文件中，如 PXI8996.Bas 文件中。

Visual Basic 程序举例:

```
' 在模块文件中定义子线程函数(注意参考实例)
Function NewRoutine() As Long        ' 定义子线程函数
        :                            ' 线程运行代码
NewRoutine = 1    ' 返回成功码
End Function
'
' 在窗体文件中创建子线程
        :
Dim hNewThread As Long
      If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
           MsgBox "创建子线程失败"
           Exit Sub
      End If
      ResumeThread (hNewThread)   '启动新线程          :
        :
```

◆ **在 VB 中，删除子线程对象**

函数原型：

Visual C++ & C++ Builder:

BOOL TerminateVBThread(HANDLE hThreadHandle)

Visual Basic:

Declare Function TerminateVBThread Lib "PXI8996" (ByVal hThreadHandleAs Long) As Boolean

功能：在VB中删除由CreateVBThread创建的子线程对象。

参数：hThreadHandle指向需要删除的子线程对象的句柄，它应由CreateVBThread创建。

返回值：当成功删除子线程对象时，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

Visual Basic 程序举例:

```
        :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
     MsgBox "创建子线程失败"
     Exit Sub
End If
        :
```

◆ **创建内核系统事件**

函数原型：

Visual C++ & C++ Builder:

HANDLE CreateSystemEvent(void)

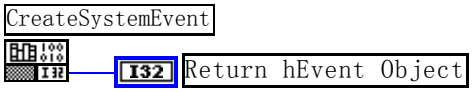
Visual Basic:

Declare Function CreateSystemEvent Lib " PXI8996 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'PXI8996' Name ' CreateSystemEvent ';

LabVIEW:



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。
参数：无任何参数。
返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

♦ **释放内核系统事件**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function ReleaseSystemEvent Lib " PXI8996 " (ByVal hEvent As Long) As Boolean
```

Delphi:

```
Function ReleaseSystemEvent(hEvent : Integer) : Integer;
                                    StdCall; External 'PXI8996' Name ' ReleaseSystemEvent ';
```

LabVIEW:

请参见相关演示程序。

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

♦ **高效高精度延时**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function DelayTimeUs Lib "PXI8996" (ByVal hDevice As Long, _
                                            ByVal nTimeUs As Long) As Boolean
```

Delphi:

```
Function DelayTimeUs (hDevice: Integer;
                      nTimeUs : LongInt) : Boolean;
                      StdCall; External 'PXI8996' Name ' DelayTimeUs ';
```

LabVIEW:

详见相关演示程序。

功能：微秒级延时函数。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nTimeUs 时间常数。单位 1 微秒。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获错误码。

第五节、文件对象操作函数原型说明

♦ **创建文件对象**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function CreateFileObject Lib "PXI8996" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer;
                           strFileName : string;
                           Mode : Integer) : Integer;
```

```
Stdcall; external 'PXI8996' Name ' CreateFileObject ';
```

LabVIEW:

请参见相关演示程序。

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

参数：

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

strFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：“C:\\PXI8996\\Data.Dat”，在 Basic 中，其语法格式如：“C:\PXI8996\Data.Dat”。

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

常量名	常量值	功能定义
PXI8996_modeRead	0x0000	只读文件方式
PXI8996_modeWrite	0x0001	只写文件方式
PXI8996_modeReadWrite	0x0002	既读又写文件方式
PXI8996_modeCreate	0x1000	如果文件不存在可以创建该文件， 如果存在，则重建此文件，且清 0
PXI8996_typeText	0x4000	以文本方式操作文件

返回值：若成功，则返回文件对象句柄。

相关函数： CreateDevice CreateFileObject WriteFile ReadFile ReleaseFile ReleaseDevice

- **通过设备对象，往指定磁盘上写入用户空间的采样数据**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function WriteFile Lib "PXI8996" ( ByVal hFileObject As Long,_
                                           ByRef pDataBuffer As Integer,_
                                           ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongInt) : Boolean;
                   Stdcall; external 'PXI8996' Name ' WriteFile ';
```

LabVIEW:

详见相关演示程序。

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

参数：

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

pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。

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

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

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **通过设备对象,从指定磁盘文件中读采样数据**

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReadFile( HANDLE hFileObject,
               PVOID pDataBuffer,
```

```
                    LONG nOffsetBytes,
                    LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PXI8996" (  ByVal hFileObject As Long, _
                                           ByRef pDataBuffer As Integer, _
                                           ByVal nOffsetBytes As Long, _
                                           ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile(hFileObject : Integer;
                  pDataBuffer : Pointer;
                  nOffsetBytes : LongInt;
                  nReadSizeBytes : LongInt) : Boolean;
                  Stdcall; external 'PXI8996' Name ' ReadFile ';
```

LabVIEW:

详见相关演示程序。

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

参数：

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

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

nOffsetBytes 指定从文件开始端所偏移的读位置。

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

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

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **设置文件偏移位置**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function SetFileOffset Lib "PXI8996" ( ByVal hFileObject As Long,
                                                ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset ( hFileObject : Integer;
                         nOffsetBytes : LongInt) : Boolean;
                         Stdcall; external 'PXI8996' Name ' SetFileOffset ';
```

LabVIEW:

详见相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

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

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

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function GetFileLength Lib "PXI8996" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject : Integer) : LongWord;
                        Stdcall; external 'PXI8996' Name ' GetFileLength ';
```

LabVIEW:

详见相关演示程序。

功能：取得文件长度。
参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。
返回值：若成功，则返回>1，否则返回 0，用户可以用GetLastErrorEx捕获错误码。
相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

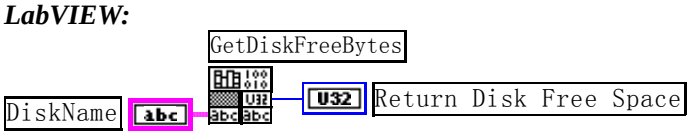
- **释放设备文件对象**

函数原型：
Visual C++ & C++ Builder:
```
BOOL ReleaseFile(HANDLE hFileObject)
```
Visual Basic:
```
Declare Function ReleaseFile Lib "PXI8996" (ByVal hFileObject As Long) As Boolean
```
Delphi:
```
Function ReleaseFile(hFileObject : Integer) : Boolean;
                    Stdcall; external 'PXI8996' Name ' ReleaseFile ';
```
LabVIEW:
详见相关演示程序。

功能：释放设备文件对象。
参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。
相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得指定磁盘的可用空间**

函数原型：
Visual C++ & C++ Builder:
```
ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName )
```
Visual Basic:
```
Declare Function GetDiskFreeBytes Lib "PXI8996" (ByVal strDiskName As String ) As Currency
```
Delphi:
```
Function GetDiskFreeBytes (strDiskName: String) : Currency;
                          Stdcall; external 'PXI8996' Name ' GetDiskFreeBytes ';
```
LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。
参数：strDiskName 需要访问的盘符，若为 C 盘为"C:\\"， D 盘为"D:\\"，以此类推。
返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用GetLastErrorEx捕获错误码。注意使用 64 位整型变量。

第六节、各种参数保存和读取函数原型说明

- **将整型变量的参数值保存在系统注册表中**

函数原型：
Visual C++ & C++ Builder:
```
BOOL SaveParaInt( HANDLE hDevice, LPCTSTR strParaName, int nValue)
```
Visual Basic:
```
Declare Function SaveParaInt Lib "PXI8996" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nValue As Integer) As Boolean
```
Delphi:
```
Function SaveParaInt( hDevice : Integer;
```

```
                    strParaName : String;
                    nValue : Integer) : Boolean;
                    Stdcall; external 'PXI8996' Name ' SaveParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PXI8996\Device-0\Others。

参数：

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

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

- **将整型变量的参数值从系统注册表中读出**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function LoadParaInt Lib "PXI8996" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nDefaultVal As Integer) As Long
```

Delphi:

```
Function LoadParaInt (hDevice : Integer;
                    strParaName : String;
                    nDefaultVal: Integer) : LongWord;
                    Stdcall; external 'PXI8996' Name ' LoadParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PXI8996\Device-0\Others。

参数：

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

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

- **将字符变量的参数值保存在系统注册表中**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function SaveParaString Lib "PXI8996" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String) As Boolean
```

Delphi:

```
Function SaveParaString ( hDevice : Integer;
                        strParaName : String;
                        strParaVal: String) : Boolean;
                        Stdcall; external 'PXI8996' Name ' SaveParaString';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PXI8996\Device-0\Others。

参数：

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

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

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

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

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

- **将字符变量的参数值从系统注册表中读出**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function LoadParaString Lib "PXI8996" (ByVal hDevice As Long,_
                                                ByVal strParaName As String,_
                                                ByVal strParaVal As String,_
                                                ByVal strDefaultVal As String) As Boolean
```

Delphi:

```
Function LoadParaString (hDevice : Integer;
                         strParaName : String;
                         strParaVal : String;
                         strDefaultVal : String) : Boolean;
                         Stdcall; external 'PXI8996' Name ' LoadParaString ';
```

LabVIEW:

详见相关演示程序。

功能：将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PXI8996\Device-0\Others。

参数：

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

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

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

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

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

第七节、其他函数原型说明

- **探测用户是否有按键动作**

函数原型：

Visual C++ & C++ Builder:

```
BOOL kbhit (void)
```

Visual Basic:

```
Declare Function kbhit Lib "PXI8996" () As Boolean
```

Delphi:

```
Function kbhit () : Boolean;
                Stdcall; external 'PXI8996' Name ' kbhit ';
```

LabVIEW:

详见相关演示程序。

功能：探测用户是否用键盘按键动作，主要应在基于 VB、DELPHI 等控制台应用程序中。
参数：无。
返回值：若自上次探测过后，若用户有键盘按键动作，则返回 TRUE，否则返回 FALSE。
相关函数： getch kbhit

- **等待按键动作并返回按键值**

函数原型：
Visual C++ & C++ Builder:
```
char getch (void)
```
Visual Basic:
```
Declare Function getch Lib "PXI8996" () As String
```
Delphi:
```
Function getch () : char;
                Stdcall; external 'PXI8996' Name 'getch';
```
LabVIEW:
详见相关演示程序。

功能：探等待用户键盘按键并以字符方式返回按键值，主要应在基于 VB、DELPHI 等控制台应用程序中。
参数：无。
返回值：若用户没有按键动作，此函数一直不返回，一旦用户有按键动作，便立即返回，且返回其当前按键值(ACII 码)。
相关函数： getch kbhit

- **怎样获取驱动函数错误信息**

函数原型：
Visual C++ & C++ Builder:
```
DWORD GetLastErrorEx (LPCTSTR strFuncName, LPCTSTR strErrorMsg)
```
Visual Basic:
```
Declare Function GetLastErrorEx Lib "PXI8996" (ByVal strFuncName As String,_
                                                ByVal strErrorMsg As String) As Long
```
Delphi:
```
Function GetLastErrorEx (strFuncName: String;
                         strErrorMsg: String) : LongWord;
                         Stdcall; external 'PXI8996' Name ' GetLastErrorEx ';
```
LabVIEW:
详见相关演示程序。

功能：将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字串。
参数：
strFuncName 出错函数的名称。注意此函数必须是完整名称，如 AD 初始化函数 PXI8996_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PXI8996_InitDeviceAD”，否则得不到相应信息。
strErrorMsg 取得指定函数的错误信息串。该串为字符数组，其分配空间最好不要小于 256 字节。
返回值：返回错误码。
相关函数：无。

Visual C++ & C++Builder 程序举例

```
        :
char strErrorMsg[256];     // 用于返回错误信息字串，要求其空间足够大
DWORD dwErrorCode;
int DeviceLgcID = 0;
hDevice = PXI8996_CreateDevice ( DeviceLgcID ); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
 {
     dwErrorCode = PXI8996_GetLastErrorEx(“PXI8996_CreateDevice”, strErrorMsg);
     AfxMessageBox(strErrorMsg); // 以对话框方式显示错误信息
     return;     // 退出该函数
 }
    :
```

Visual Basic 程序举例

```
    :
Dim strErrormsg As String   ' 用于返回错误信息字串，要求其空间足够大
Dim dwErrorCode As Long
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PXI8996_CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    dwErrorCode = PXI8996_GetLastErrorEx("PXI8996_CreateDevice", strErrorMsg)
    MsgBox strErrorMsg   ' 以对话框方式显示错误信息
    Exit Sub      ' 退出该过程
End If
    :
```