PXI8008 WIN2000/XP 驱动程序使用说明书

请您务必阅读《使用纲要》，他会使您事半功倍!

目　　录

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 PXI8008_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写规则

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

以上规则不局限于该产品。

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

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

第二节、如何管理 PXI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用CreateDevice函数创建一个设备对象句柄 hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，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，SetDeviceDO函数可用实现开关量的输出等。最后可以通过ReleaseDevice将 hDevice 释放掉。

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。

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。

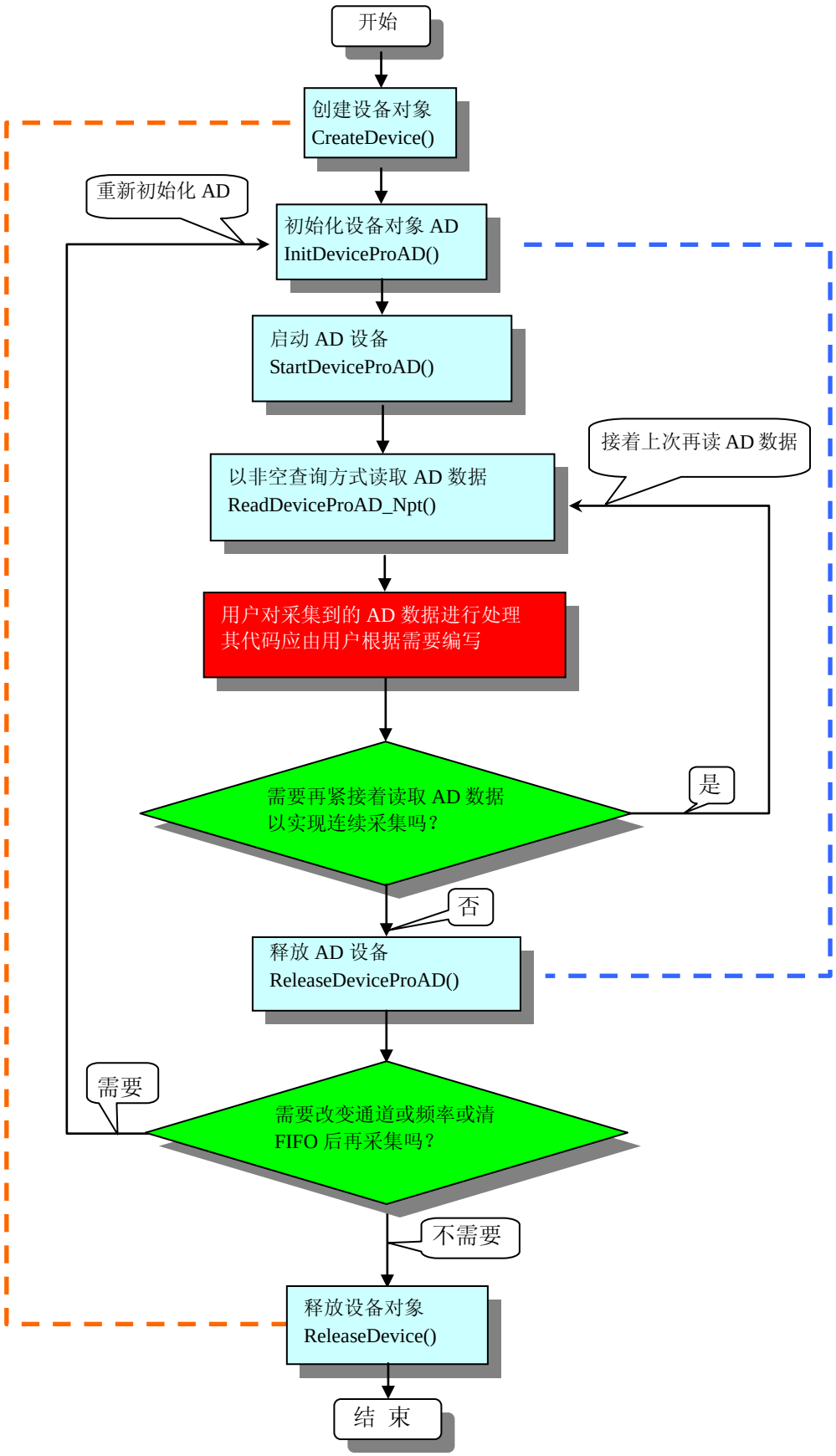


图 2.1.1 非空查询方式 AD 采集过程

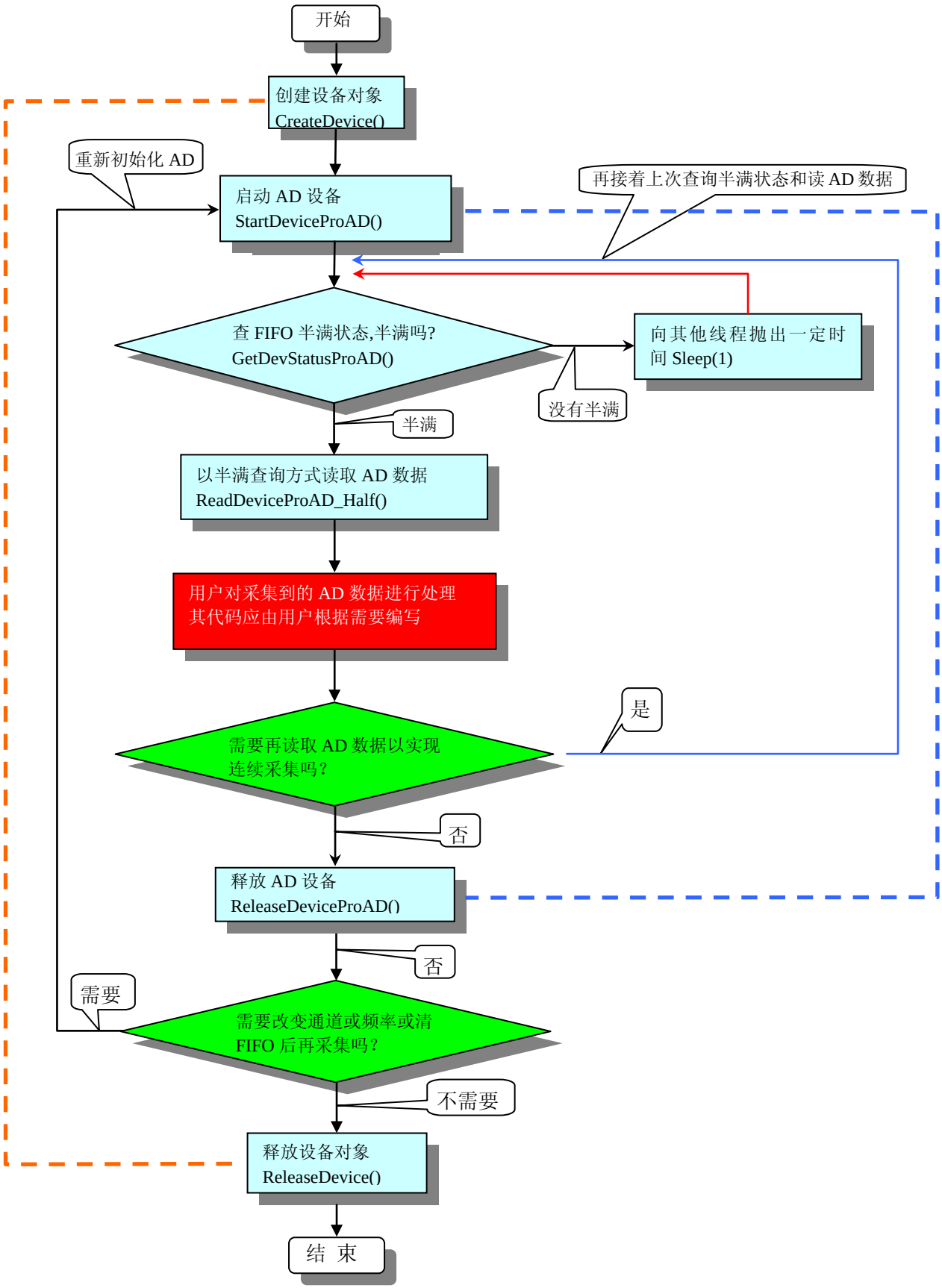


图 2.1.2 半满查询方式 AD 采集过程

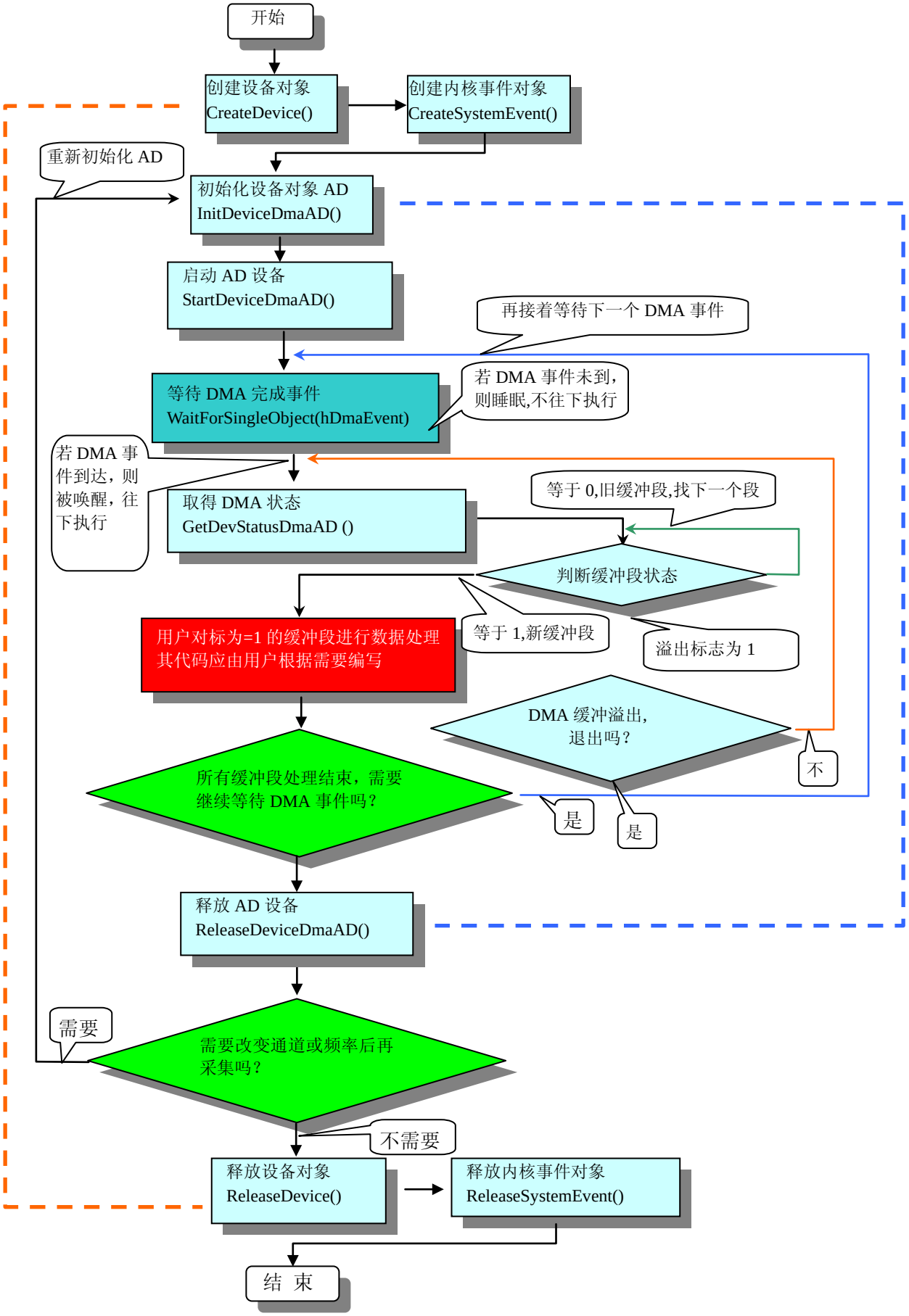


图 2.1.3 DMA 方式 AD 采集实现过程

第六节、哪些函数对您不是必须的

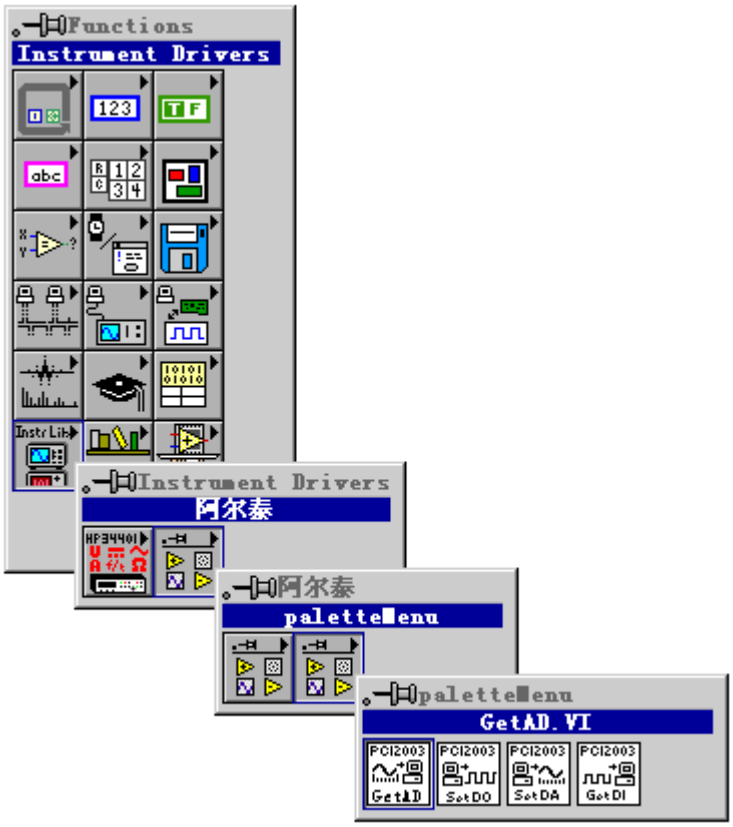
当公共函数如CreateFileObject, WriteFile, ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问备，那么GetDeviceAddr，WriteRegisterByte, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte, WritePortWord, WritePortULong, ReadPortByte, ReadPortWord, ReadPortULong则对 PXI 用户来说，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 Win2000、Win XP 等操作系统中实现对您原有传统设备如 ISA 卡、串口卡、并口卡的访问，而没有这些函数，您可能在新操作系统中无法继续使用您原有的老设备（除非您自己愿意去编写复杂的硬件驱动程序）。

第三章 PXI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节、只关心 AD 的首末通道、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 AD 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，我们则把这一群需要直接跟设备端口打交道的用户称之为底层用户。因此总的看来，上层用户要求简单，快捷，他们最希望他们在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如InitDeviceProAD）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用ReadDeviceProAD（或ReadDeviceDmaAD）函数只须指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的 Bit 位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉 PXI 总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解 PXI 的资源配置空间、PNP 即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用ReadRegisterULong和WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包极大的方便和满足了您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先得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。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、设备驱动接口函数列表（每个函数省略了前缀“PXI8008_”）

函数名	函数功能	备注
设备对象操作函数		
CreateDevice	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
CreateDeviceEx	创建 PCI 设备对象(用设备物理号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 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 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
AD 的程序方式读取函数		
InitDeviceProAD	初始化 AD 部件准备传输	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 PCI 设备上的 AD 数据	上层用户
GetDevStatusProAD	取得当前 PCI 设备 FIFO 半满状态	上层用户
ReadDeviceProAD_Half	连续批量读取 PCI 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
AD 的 DMA 方式读取函数（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
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++：

要使用如下函数关键的问题是：

首先，将 PXI8008.h 和 PXI8008.lib 文件从 Visual C++的源程序目录下的任意一个子目录下复制到您的源程序目录下（若有 Advanced 高级源程序目录，则最好选择它），然后在您的源程序中包含如下语句（若想在整个工程的所有源代码文件中使用本驱动，请您最好在 StdAfx.h 全局头文件中包含如下语句）：

#include “PXI8008.H”

那么对于导入库 PXI8008.lib 文件您则可以不必再加入您的工程，因为 PXI8008.h 头文件已帮助自动完成了。

Visual Basic:

要使用如下函数一个关键的问题是：

首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单,执行其中的"添加模块"(Add Module)命令,在弹出的对话中选择 PXI8008.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

LabVIEW/CVI :

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

一、在 LabVIEW 中打开 PXI8008.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续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竖线为中心，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_NotEmpty 接口单元，左边为设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

♦ 创建设备对象函数（逻辑号）

函数原型：

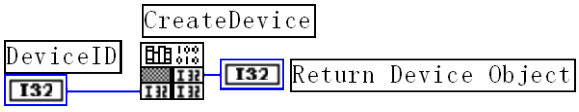
Visual C++：

HANDLE CreateDevice (int DeviceLgcID=0)

Visual Basic：

Declare Function PXI8008_CreateDevice Lib "PXI8008_32" (ByVal DeviceID As Long) As Long

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能

实现对该设备所有功能的访问。

参数：

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PCI 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 PXI8008 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 PXI8008 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 PCI 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++ 程序举例

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

 }
   :
```

Visual Basic 程序举例

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

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

♦ **创建设备对象函数（物理号）**

函数原型：

Visual C++：

HANDLE CreateDeviceEx(int DevicePhysID=0)

Visual Basic：

Declare Function PXI8008_CreateDeviceEx Lib "PXI8008_32" (ByVal DevicePhysID As Long) As Long

LabVIEW：

请参考相关演示程序。

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

参数：

DevicePhysID 物理设备 ID(Physic Device Identifier)标识号。由CreateDevice函数的 DevieLgcID 参数说明中可以看出，逻辑 ID 号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如 A、B、C、D 四个卡，构成 256 个通道（64*4），其通道序列为 0-255，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 0-63 通道上，B 卡位于 63-127 通道上，C 卡位于 128-191 通道上，而 D 卡则位于 192-255 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 0-255 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。

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 PXI8008 设备的总数量**

函数原型：

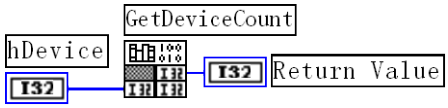
Visual C++:

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

Visual Basic:

```
Declare Function PXI8008_GetDeviceCount Lib "PXI8008_32" (ByVal hDevice As Long) As Long
```

LabVIEW：



功能：取得 PXI8008 设备的数量。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：返回系统中 PXI8008 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++:

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

Visual Basic:

```
Declare Function PXI8008_GetDeviceCurrentID Lib "PXI8008_32" (_
                                    ByVal hDevice As Long, _
                                    ByRef DeviceLgcID As Long, _
                                    ByRef DevicePhysID As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：取得 PXI8008 设备的数量。

参数：

hDevice 设备对象句柄，它应由CreateDevice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 GetLastError 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 PXI8008 设备各种配置信息**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI8008_ListDeviceDlg Lib "PXI8008_32" (ByVal hDevice As Long) As Boolean
```

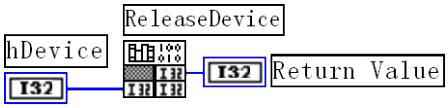
LabVIEW：

请参考相关演示程序。

功能：列表系统中 PXI8008 的硬件配置信息。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则弹出对话框控件列表所有 PXI8008 设备的配置情况。
相关函数：CreateDevice ReleaseDevice

♦ **释放设备对象所占的系统资源及设备对象**

函数原型：
Visual C++:
BOOL ReleaseDevice(HANDLE hDevice)
Visual Basic:
Declare Function PXI8008_ReleaseDevice Lib "PXI8008_32" (ByVal hDevice As Long) As Boolean
LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则返回 TRUE，否则返回 FALSE，用户可以用 GetLastError 捕获错误码。
相关函数：CreateDevice
应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice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 数据采样操作函数原型说明

♦ **初始化设备对象**

函数原型：
Visual C++:
BOOL InitDeviceProAD(HANDLE hDevice,
PPXI8008_PARA_AD pADPara)
Visual Basic:
Declare Function PXI8008_InitDeviceProAD Lib "PXI8008_32" (_
ByVal hDevice As Long, _
ByRef pADPara As PXI8008_PARA_AD) As Boolean

LabVIEW：
请参考相关演示程序。

功能：它负责初始化设备对象中的 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。但它并不启动 AD 设备，若要启动 AD 设备，须在调用此函数之后再调用StartDeviceProAD 。
参数：
hDevice 设备对象句柄,它应由CreateDevice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,如采样频率等。关于具体操作请参考第四章《硬件参数结构》中的《AD 硬件参数结构》。
返回值：如果初始化设备对象成功,则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码,并加以分析
相关函数： CreateDevice InitDeviceProAD SetDevFrequencyAD
StartDeviceProAD GetDevStatusProAD ReadDeviceProAD
ReadDeviceDmaAD StopDeviceProAD ReleaseDevice

♦ **启动 AD 设备**

函数原型：
Visual C++:
BOOL StartDeviceProAD (HANDLE hDevice)

Visual Basic:

```
Declare Function PXI8008_StartDeviceProAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long) As Boolean
```

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。调用该函数后它立即启动。

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

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

相关函数： CreateDevice　InitDeviceProAD　SetDevFrequencyAD
StartDeviceProAD　GetDevStatusProAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceProAD　ReleaseDevice

♦ **读取设备上的 AD 数据**

① 使用 FIFO 的非空标志读取 AD 数据

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function PXI8008_ReadDeviceProAD_Npt Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByRef ADBuffer As Long, _
                                        ByVal nReadSizeWords As Long, _
                                        ByRef nRetSizeWords As Long) As Long
```

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 GetLastError 捕获当前错误码，并加以分析。

注释：此函数也可用于单点读取和几个点的读取，只需要将 nReadSizeWords 设置成 1 或相应值即可。其使用方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》章节。

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 AD 的状态标志**

函数原型：

Visual C++:

```
BOOL GetDevStatusProAD ( HANDLE hDevice, PPXI8008_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function PXI8008_GetDevStatusProAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByRef pADStatus As PXI8008_STATUS_AD) As Boolean
```

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 AD 状态去同步 RAM 读操作。ADStatus. bRamSwitch 等于 TRUE 时，板载 RAM 已发生乒乓切换，用户便可调用ReadDeviceProAD或ReadDeviceDmaAD 函数读取 RAM 中的 AD 数据。由于每个通道具有 64K 点（字）的存储器深度，因此当调用StartDeviceProAD 启动 AD 时，写满 RAM 需要一定时间，所以当该函数取回的 ADStatus. bRamSwitch 值为 FALSE，则用户应继续等待，直到等于 TRUE 时才能读取数据。

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

pADStatus 设备状态参数结构，它返回设备当前的各种状态，如板载 RAM 是否发生切换、重写、触发点是否产生等信息。关于具体操作请参考第四章《硬件参数结构》中的《AD 硬件参数结构》。

返回值：若 AD 成功取标状态，则返回 TRUE，否则返回 FALSE。注意在 Win2K 以上系统中，在循环轮询期间，尽可能使用 WaitForSingleObject 来等待切换中断事件，且使用 DMA 方式得到更高的数据采样率和系统的整体性能。

相关函数： CreateDevice　InitDeviceProAD　SetDevFrequencyAD
StartDeviceProAD　GetDevStatusProAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceProAD　ReleaseDevice

♦ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function PXI8008_ReadDeviceProAD_Half Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByRef ADBuffer As Long, _
                                        ByVal nReadSizeWords As Long, _
                                        ByRef nRetSizeWords As Long) As Boolean
```

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 GetLastError 捕获当前错误码，并加以分析。

其使用方法请参考本部分第十章 《高速大容量、连续不间断数据采集及存盘技术详解》。

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++:

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

Visual Basic:

```
Declare Function PXI8008_StopDeviceProAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long) As Boolean
```

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。

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

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

相关函数： CreateDevice　InitDeviceProAD　SetDevFrequencyAD
StartDeviceProAD　GetDevStatusProAD　ReadDeviceProAD
ReadDeviceDmaAD　StopDeviceProAD　ReleaseDevice

♦ **释放设备上的 AD 部件**

函数原型：

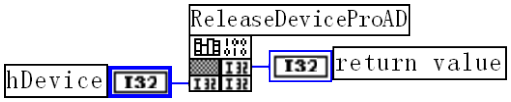
Visual C++:

```
BOOL ReleaseDeviceProAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function PXI8008_ReleaseDeviceProAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long) As Boolean
```

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 GetLastError 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++:

```
BOOL InitDeviceDmaAD( HANDLE hDevice,
                      HANDLE hDmaEvent,
                      LONG ADBuffer[ ],
                      LONG nReadSizeWords,
                      LONG nSegmentCount,
                      LONG nSegmentSizeWords,
                      PPXI8008_PARA_AD pADPara )
```

Visual Basic:

```
Declare Function PXI8008_InitDeviceDmaAD Lib "PXI8008_32" ( _
                                        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 PXI8008_PARA_AD) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的 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

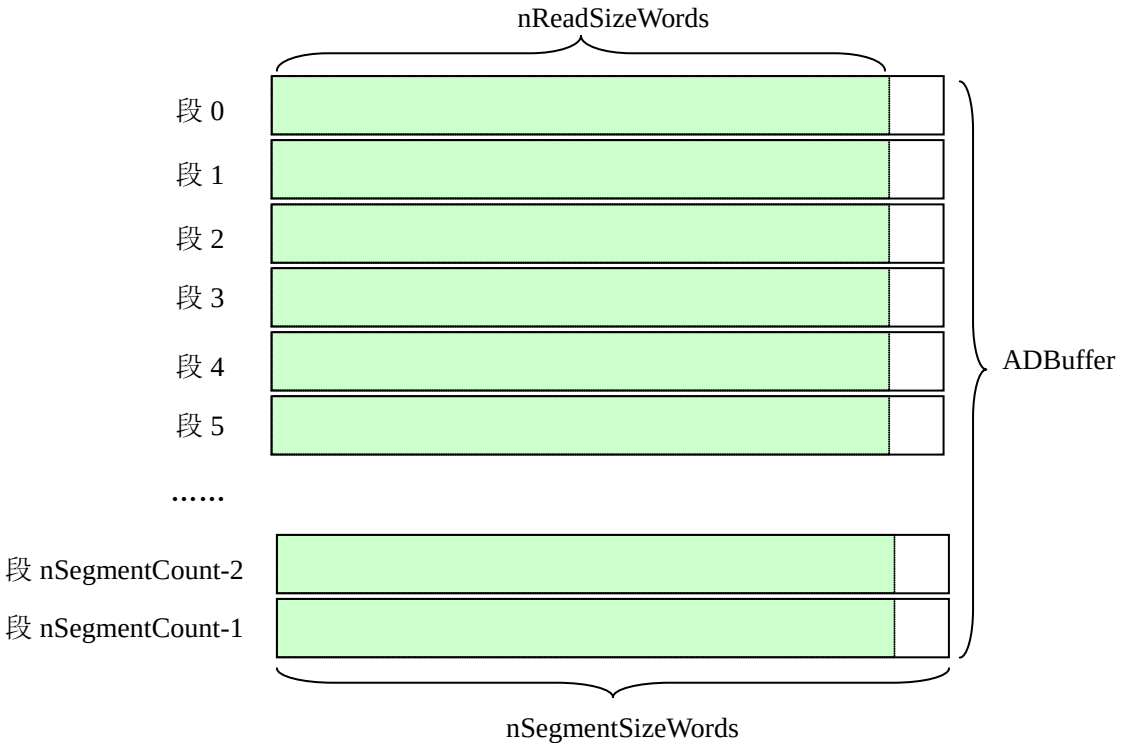
配函数分配的内存空间。关于如何将缓冲区中的这些 AD 数据转换成相应的电压值，请参考请第六章《数据格式转换与排列规则》。注意该缓冲区最好定义为两维缓冲或数组，以便 DMA 数据传输和缓冲区数据处理分时错开，以更好的达到 AD 转换、传输、处理等过程的并行工作。**注意：该缓冲区的生命周期必须跨越 DMA 的整个操作周期，我建议最好将期置为全局缓冲区，即整个应用程序的生命周期内存在。否则，可能会造成严重的存储区访问违反**。

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 PXI8008_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 PXI8008.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 GetLastError 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++:

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

Visual Basic:

```
Declare Function PXI8008_StartDeviceDmaAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long) As Boolean
```

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 GetLastError 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**

函数原型：

Visual C++:

```
BOOL GetDevStatusDmaAD ( HANDLE hDevice,
                         PPXI8008_STATUS_DMA pDMAStatus )
```

Visual Basic:

```
Declare Function PXI8008_GetDevStatusDmaAD Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByRef pDMAStatus As PXI8008_STATUS_DMA) As Boolean
```

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 PXI8008_STATUS_DMA 的结构体指针。该参数实时返回 DMA 的当前状态。关于 PXI8008_STATUS_DMA 具体定义请参考 PXI8008.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（PXI8008_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 GetLastError 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++:

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

Visual Basic:

```
Declare Function PXI8008_SetDevStatusDmaAD Lib "PXI8008_32" ( _
                                    ByVal hDevice As Long, _
                                    ByVal iClrBufferID As Long) As Boolean
```

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 GetLastError 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++:

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

Visual Basic:

```
Declare Function PXI8008_StopDeviceDmaAD Lib "PXI8008_32" ( _
                                    ByVal hDevice As Long) As Boolean
```

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 GetLastError 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++:

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

Visual Basic:

```
Declare Function PXI8008_ReleaseDeviceDmaAD Lib "PXI8008_32" ( _
                                    ByVal hDevice As Long) As Boolean
```

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 GetLastError 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 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++:
BOOL LoadParaAD(HANDLE hDevice, PPXI8008_PARA_AD pADPara)
Visual Basic:
```
Declare Function PXI8008_LoadParaAD Lib "PXI8008_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pADPara As PXI8008_PARA_AD) As Boolean
```
LabVIEW:
请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。
参数：
hDevice 设备对象句柄,它应由CreateDevice创建。
pADPara 属于 PPXI8008_PARA_AD 的结构指针类型，它负责返回 PCI 硬件参数值，关于结构指针类型 PPXI8008_PARA_AD 请参考 PXI8008.h 或 PXI8008.Bas 或 PXI8008.Pas 函数原型定义文件，也可参考本文第四章《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

♦ **写设备硬件参数函数到 Windows 系统中**

函数原型：
Viusal C++:
BOOL SaveParaAD (HANDLE hDevice, PPXI8008_PARA_AD pADPara)
Visual Basic:
```
Declare Function PXI8008_SaveParaAD Lib "PXI8008_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pADPara As PXI8008_PARA_AD) As Boolean
```

LabVIEW:
请参考相关演示程序。

功能：负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。
参数：
hDevice 设备对象句柄,它应由CreateDevice创建。
pADPara 设备硬件参数，关于 PXI8008_PARA_AD 的详细介绍请参考 PXI8008.h 或 PXI8008.Bas 或 PXI8008.Pas 函数原型定义文件，也可参考本文第四章《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

◆ **将硬件参数结构体值复位为出厂默认值**
函数原型：
Viusal C++:
BOOL ResetParaAD (HANDLE hDevice, PPXI8008_PARA_AD pADPara)
Visual Basic:
Declare Function PXI8008_ResetParaAD Lib "PXI8008_32" (ByVal hDevice As Long,_
ByRef pADPara As PXI8008_PARA_AD) As Boolean
LabVIEW:
请参考相关演示程序。

功能：负责将硬件参数的值复位至出厂默认值，不仅会将 pADPara 指向的结构体成员值更新为默认值，同时会将系统中保存的参数更新为默认值。这些默认值在产品驱动第一次被安装时会出现。而且这些默认值的设定是充分的考虑到用户的实际情况，确保用户不用外部任何条件，只要开始采集数据，即可获得相应的结果。
参数：
hDevice 设备对象句柄,它应由CreateDevice创建。
pADPara 设备硬件参数，关于 PXI8008_PARA_AD 的详细介绍请参考 PXI8008.h 或 PXI8008.Bas 或 PXI8008.Pas 函数原型定义文件，也可参考本文第四章《硬件参数结构》关于该结构的有关说明。调用此函数后，该参数指向的结构体成员将被复位至默认值。
返回值：若成功，返回 TRUE，它表明已成功将系统中的 AD 参数复位至默认值，同时更新了 pADPara 指向的结构体。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（PXI8008_PARA_AD）

Visual C++：

```
typedef struct _PXI8008_PARA_AD
{
LONG bChannelArray[16]; // 采样通道选择阵列，分别控制 16 个通道，=TRUE 表示该通道采样，否则不采样
LONG Gains[16];          // 模拟量增益选择阵列，分别控制 16 个通道
LONG Frequency;          // 采集频率，单位为 Hz, [3, 80000]
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 TrigLevelVolt;      // 触发电平(±5000mV)
LONG TrigWindow;         // 触发灵敏窗[1, 65535]，单位 25 纳秒
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
```

} PXI8008_PARA_AD, *PPXI8008_PARA_AD;

Visual Basic：

```
Type PXI8008_PARA_AD
    bChannelArray(0 To 15) As Long    ' 采样通道选择阵列，分别控制 16 个通道，=TRUE 表示该通道采样，否则不采样
    Gains(0 To 15) As Long        ' 模拟量增益选择阵列，分别控制 16 个通道
    Frequency As Long             ' 采集频率，单位为 Hz, [3, 80000]
    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)
    TrigLevelVolt   As Long       ' 触发电平(±5000mV)
    TrigWindow    As Long         ' 触发灵敏窗[1, 65535], 单位 25 纳秒
    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,=TRUE:允许输出, =FALSE:禁止输出
End Type
```

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 InitDeviceIntAD 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。

Gains[x] 模拟量输入范围所使用的选项，取值如下表：

常量名	常量值	功能定义
PXI8008_GAINS_1MULT	0x0000	1 倍增益(使用 AD8251 放大器)
PXI8008_GAINS_2MULT	0x0001	2 倍增益(使用 AD8251 放大器)
PXI8008_GAINS_4MULT	0x0002	4 倍增益(使用 AD8251 放大器)
PXI8008_GAINS_8MULT	0x0003	8 倍增益(使用 AD8251 放大器)

Frequency AD 采样率(Hz)，其取值范围为[3Hz, 80KHz]。

TriggerMode AD 触发模式选择，其的取值如下表：

常量名	常量值	功能定义
PXI8008_TRIGMODE_SOFT	0x0000	软件内触发
PXI8008_TRIGMODE_POST	0x0001	硬件后触发

TriggerSource AD 触发源选择，其选项值如下表：

常量名	常量值	功能定义
PXI8008_TRIGSRC_ATR	0x0000	选择外部 ATR 触发源
PXI8008_TRIGSRC_DTR	0x0001	选择外部 DTR 触发源
PXI8008_TRIGSRC_PXI_TRIG0	0x0002	选择 PXI 总线上的 TRIG0 触发源
PXI8008_TRIGSRC_PXI_TRIG1	0x0003	选择 PXI 总线上的 TRIG1 触发源
PXI8008_TRIGSRC_PXI_TRIG7	0x0004	选择 PXI 总线上的 TRIG7 触发源
PXI8008_TRIGSRC_PXI_STAR	0x0005	选择 PXI 总线上的 STAR 触发源

TriggerType AD 触发类型选择。选项值如下表：

常量名	常量值	功能定义
PXI8008_TRIGTYPE_EDGE	0x0000	边沿触发
PXI8008_TRIGTYPE_PULSE	0x0001	脉冲触发(电平方式),适用于采集馒头波信号

TriggerDir AD 触发方向选择。选项值如下表：

常量名	常量值	功能定义

PXI8008_TRIGDIR_NEGATIVE	0x0000	负向触发(低脉冲/下降沿触发)
PXI8008_TRIGDIR_POSITIVE	0x0001	正向触发(高脉冲/上升沿触发)
PXI8008_TRIGDIR_POSIT_NEGAT	0x0002	正负向触发(高/低脉冲或上升/下降沿触发)

当 TriggerType= PXI8008_TRIGTYPE_EDGE **时（此种方式下，要求触发信号相对于门坎电压必须有上下跳变）：**

当 TriggerDir= PXI8008_TRIGDIR_NEGATIVE 时，表示外部触发信号须由大于 AO0 变成小于 AO0 时产生触发（即下降沿触发）。一旦触发产生后，其后续的触发信号变化均无效。

当 TriggerDir= PXI8008_TRIGDIR_POSITIVE 时，表示外部触发信号由小于 AO0 变成大于 AO0 时触发产生(即上升沿触发)。一旦触发产生后，其后续的触发信号变化均无效。

当 TriggerDir= PXI8008_TRIGDIR_POSIT_NEGAT 时，表示外部触发信号由小于 AO0 变成大于 AO0 以及外部触发信号由大于 AO0 变成小于 AO0 时均产生触发((即上下沿均触发))。一旦触发产生后，其后续的触发信号变化均无效。

当 TriggerType= PXI8008_TRIGTYPE_PULSE **时（此种方式下，不要求触发信号必须有大跳变，而是相对稳定）：**

当 TriggerDir=PXI8008_TRIGDIR_NEGATIVE 时，表示外部触发信号若小于触发电平(TriggerLevelLsb)，则触发产生，AD 开始工作，但一旦触发信号大于触发电平时，则 AD 自动停止工作，直到触发信号再次小于触发电平时 AD 便会又自动开始工作，即只采集触发电平下方的波形。

当 TriggerlDir= PXI8008_TRIGDIR_POSITIVE 时，表示外部触发信号若大于触发电平(TriggerLevelLsb)，则触发产生，AD 开始工作，但一旦触发信号小于触发电平时，则 AD 自动停止工作，直到触发信号再次大于触发电平时 AD 便会又自动开始工作，即只采集触发电平上方的波形。

当 TriggerDir= PXI8008_POSIT_NEGAT_DIR 时，不管触发信号（ATR）大于还是小于触发电平均触发，AD 开始工作，此种情况相当于没有使用内触发方式，这里只是为了和边沿触发功能对齐。

TrigLevelVolt 触发电平取值。它的取值范围为[-10000，+10000]，单位为 mV。

TrigWindow 触发灵敏窗[1, 65535], 单位 25 纳秒。

ClockSource AD 时钟源选择。选项值如下表：

常量名	常量值	功能定义
PXI8008_CLOCKSRC_IN	0x0000	内部时钟
PXI8008_CLOCKSRC_OUT	0x0001	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。

第二节、AD 状态参数结构（PXI8008_STATUS_AD）

Visual C++：

```
typedef struct _PXI8008_STATUS_AD
{
LONG bNotEmpty;              // 板载 FIFO 存储器的非空标志，=TRUE 非空， = FALSE 空
LONG bHalf;                  // 板载 FIFO 存储器的半满标志，=TRUE 半满以上， = FALSE 半满以下
LONG bDynamic_Overflow; // 板载 FIFO 存储器的动态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
LONG bStatic_Overflow;  // 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
LONG bConverting;            // AD 是否正在转换， =TRUE:表示正在转换， =FALS 表示转换完成
LONG bTriggerFlag;           // 触发标志, =TRUE 表示触发事件发生， =FALSE 表示触发事件未发生
LONG nTriggerPos;           // 触发位置
} PXI8008_STATUS_AD, *PPXI8008_STATUS_AD;
```

Visual Basic：

```
Type PXI8008_STATUS_AD
    bNotEmpty   As Long    ' 板载 FIFO 存储器的非空标志，=TRUE 非空， = FALSE 空
    bHalf   As Long    ' 板载 FIFO 存储器的半满标志，=TRUE 半满以上， = FALSE 半满以下
```

```
    bDynamic_Overflow   As Long ' 板载 FIFO 存储器的动态溢出标志,= TRUE 已发生溢出，= FALSE
                                  未发生溢出
    bStatic_Overflow    As Long  ' 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出， = FALSE
                                  未发生溢出
    bConverting    As Long      'AD 是否正在转换， =TRUE:表示正在转换， =FALS 表示转换完成
    bTriggerFlag    As Long     ' 触发标志, =TRUE 表示触发事件发生， =FALSE 表示触发事件未发生
    nTriggerPos    As Long       ' 触发位置
  End Type
```

LabVIEW:

请参考相关演示程序。

此结构主要用于GetDevStatusProAD函数返回设备各状态。

bNotEmpty 板载 FIFO 存储器的非空标志，=TRUE 非空，= FALSE 空。

bHalf 板载 FIFO 存储器的半满标志，=TRUE 半满以上， = FALSE 半满以下。

bDynamic_Overflow 板载 FIFO 存储器的动态溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出。

bStatic_Overflow 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出。

bConverting 表示 AD 是否还在转换，等于 TRUE， 表示还在转换，等于 FALSE 表示停止转换。

bTriggerFlag 表示是否产生了触发点。等于 TRUE，表示产生了触发点，等于 FALSE 表示没有。

nTriggerPos 触发点位置。

第三节、DMA 状态参数结构（PXI8008_STATUS_DMA）

Visual C++：

```
  const int MAX_SEGMENT_COUNT = 128;
  typedef struct _PXI8008_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];   // 各个缓冲区的新旧状态,=1 表示该相应缓冲区数
                                                据为新,否则为旧
LONG bBufferOverflow;                       // 返回溢出状态
} PXI8008_STATUS_DMA, *PPXI8008_STATUS_DMA;
```

Visual Basic：

```
  Public Const MAX_SEGMENT_COUNT = 128
  Type PXI8008_STATUS_DMA
    iCurSegmentID    As Long          ' 当前段缓冲 ID,表示 DMA 正在传输的缓冲区段
    bSegmentSts(0 To 217) As Long    ' 各个缓冲区的新旧状态,=1 表示该相应缓冲区数据为新,否则为旧
    bBufferOverflow    As Long       ' 返回溢出状态
  End Type
```

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 63，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

在换算过程中弄清模板精度（即 Bit 位数）是很关键的，因为它决定了 LSB 数码的总宽度 CountLSB。比如 8 位的模板 CountLSB 为 256。而本设备的 AD 为 14 位，则为 16384。其他类型同理均按 2^n=LSB 总数（n 为 Bit 位数）换算即可。

设从设备上读入的某个 AD 原码数据经高位求补后为变量 Lsb，其对应的电压为变量 Volt（单位 mV）。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000mV	Volt = (20000.00/16384) * (ADBuffer[0]^0x2000) &0x3FFF) – 10000.00	[-10000，+9998.77]
±5000mV	Volt = (10000.00/16384) * (ADBuffer[0]^0x2000) &0x3FFF) – 5000.00	[-5000，+4999.38]

注意：以上所列 ADBuffer[0]必须是将从设备上读入的 AD 数据（注意在上层用户接口中的 AD 数据读取函数的 ADBuffer 参数指向的用户缓冲区存放的就是这样的最原始数据），且存放这些数据的变量也应该是 14 位整型变量。举列说明如何将取得的 AD 值转换成相应量程的电压值：（此处只转换用户缓冲区中的第一个点，其它同理，且按±10000mV 量程计算）

Visual C++：

```
Lsb = (ADBuffer[0]^0x2000) &0x3FFF;
Volt = (20000.00/16384) * Lsb – 10000.00;
```

Visual Basic：

```
Dim Lsb As Long
Lsb = ADBuffer (0)    XOR &H2000 And &H3FFF
Volt = (20000.00/16384) * Lsb – 10000.00
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

由于各个通道除了共享采样时钟和触发条件外，均独立工作，分别具有自己独立的存储器，因此其 AD 数据排列非常简单。在各自读回的数据中全为自身的数据。只有在异步模式下，其数据序列应由几个通道穿叉排列形成，即把读回的数据按下列顺序排放，以 4 个通道为例：

数据序列	通道序列
0	AI0[0]
1	AI1[0]
2	AI2[0]
3	AI3[0]
4	AI0[1]
5	AI1[1]
6	AI2[1]
7	AI3[1]
8	AI0[2]
9	AI1[2]
10	AI2[2]
11	AI3[2]
12	AI0[3]
13	AI1[3]
14	AI2[3]
15	AI3[3]
16	AI0[4]
17	AI1[4]
18	AI2[4]
19	AI3[4]
:	:
N	AI0[N/4]

N+1	AI1[N/4]
N+2	AI2[N/4]
N+3	AI3[N/4]

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)

    PCI8008_PARA_AD ADPara;    // 保存硬件参数

    LONG HeadEndFlag;          // 头信息结束位
} FILE_HEADER,  *PFILE_HEADER;
```

AD 数据的格式为 14 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 14 位二进制(字)数据对应一个 14 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++:

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**[PXI8008 同步采集卡]**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++:

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**[PXI8008 同步采集卡]**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++:

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**[PXI8008 同步采集卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[AD**

DMA 方式]

第七章 共用函数介绍

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

第一节、公用接口函数列表（每个函数省略了前缀“PXI8008_”）

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
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
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		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

第二节、PXI 内存映射寄存器操作函数原型说明

- **取得指定内存映射寄存器的线性地址和物理地址**

函数原型:

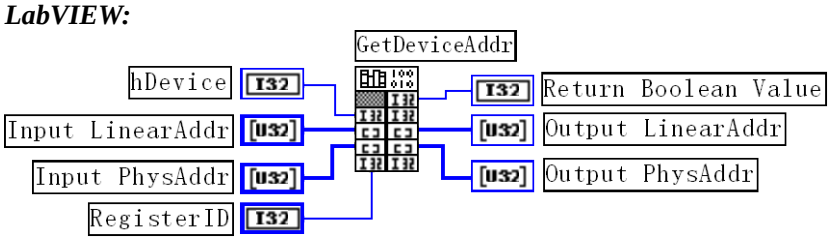
Visual C++:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    __int64 pbLinearAddr,
                    __int64 pbPhysAddr,
                    int RegisterID = 0)
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib    "PXI8008_32" ( _
                                          ByVal hDevice As Long, _
                                          ByVal pbLinearAddr As Long, _
                                          ByVal pbPhysAddr As Long, _
                                          ByVal RegisterID As Long) As Boolean
```

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
PXI8008_REG_MEM_PLXCHIP	0x0000	0 号寄存器对应 PLX 芯片所使用的内存模式基地址(使用 LinearAddr)
PXI8008_REG_IO_PLXCHIP	0x0001	1 号寄存器对应 PLX 芯片所使用的 IO 模式基地址(使用 PhysAddr)
PXI8008_REG_MEM_CPLD	0x0002	2 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)
PXI8008_REG_IO_CPLD	0x0003	3 号寄存器对应板上 AD 缓冲区所使用的 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 GetLastError 捕获当前错误码，并加以分析。

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

Visual C++程序举例:

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

Visual Basic 程序举例:

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

- **取得指定的指定设备寄存器组 BAR 地址**

函数原型:

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceBar Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByRef pbPCIBar As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

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

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

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

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

- **以单字节（即 8 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型:

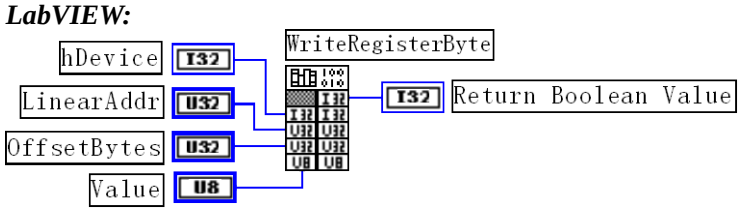
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterByte Lib
    Declare Function PXI8008_WriteRegisterByte Lib "PXI8008_32" ( _
                                                    ByVal hDevice As Long, _
                                                    ByVal pbLinearAddr As Long, _
                                                    ByVal OffsetBytes As Long, _
                                                    ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节（即 8 位）方式写 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 GetDeviceBar WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

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

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:

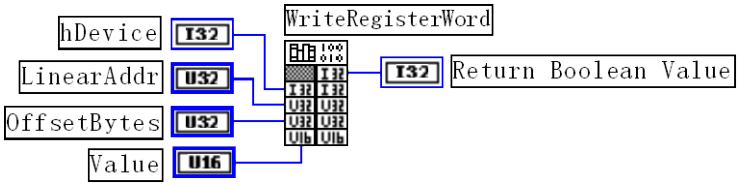
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByVal pbLinearAddr As Long, _
                                        ByVal OffsetBytes As Long, _
                                        ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 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 GetDeviceBar WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

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

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:

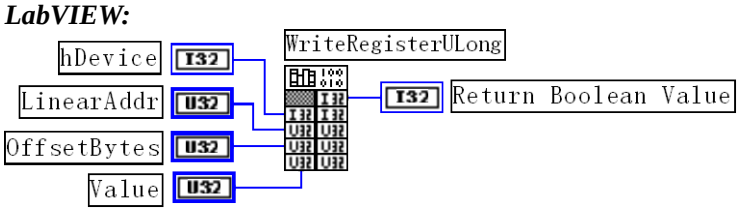
Visual C++:

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

Visual Basic:

```
Declare Function WriteRegisterULong Lib "PXI8008_32" ( _
                                                        ByVal hDevice As Long, _
                                                        ByVal pbLinearAddr As Long, _
                                                        ByVal OffsetBytes As Long, _
                                                        ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由CreateDevice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 GetDeviceBar WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

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

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:

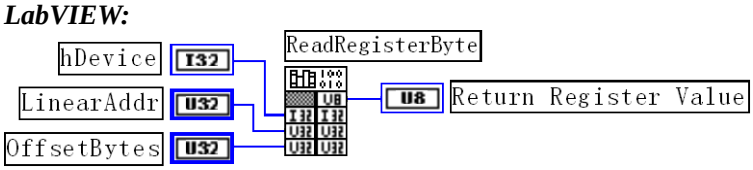
Visual C++:

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

Visual Basic:

```
Declare Function ReadRegisterByte Lib"PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByVal pbLinearAddr As Long, _
                                        ByVal OffsetBytes As Long) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 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 GetDeviceBar
WriteRegisterByte WriteRegisterWord WriteRegisterULong
ReadRegisterByte ReadRegisterWord ReadRegisterULong
ReleaseDevice

Visual C++程序举例:

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

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++:

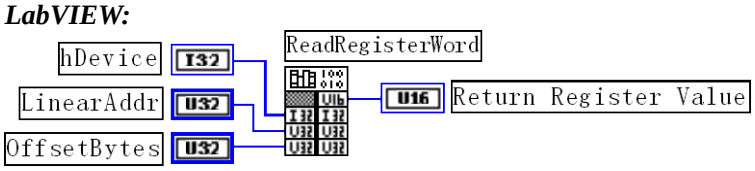
```
WORD ReadRegisterWord( HANDLE hDevice,
                       __int64 pbLinearAddr,
```

ULONG OffsetBytes)

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByVal pbLinearAddr As Long, _
                                        ByVal OffsetBytes As Long) As
```

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 GetDeviceBar WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

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

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:

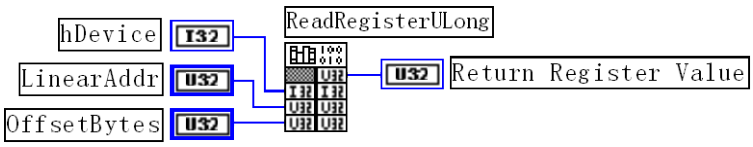
Visual C++:

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

Visual Basic:

```
Declare Function ReadRegisterULong Lib"PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByVal pbLinearAddr As Long, _
                                        ByVal OffsetBytes As Long) As Long
```

LabVIEW:



功能：以四字节（即 32 位）方式读 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　GetDeviceBar　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

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

Visual Basic 程序举例:

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

第三节、I0 端口读写函数原型说明

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

♦ **以单字节(8Bit)方式写 I/O 端口**

函数原型:

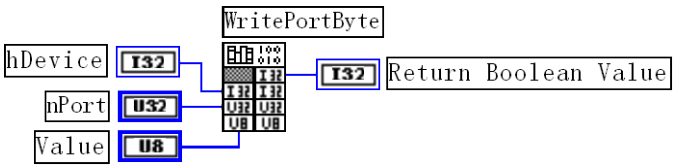
Visual C++:

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

Visual Basic:

```
Declare Function WritePortByte Lib "PXI8008_32" ( _
                                        ByVal hDevice As Long, _
                                        ByVal pbPort As Long, _
                                        ByVal OffsetBytes As Long, _
                                        ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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 GetLastError 捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

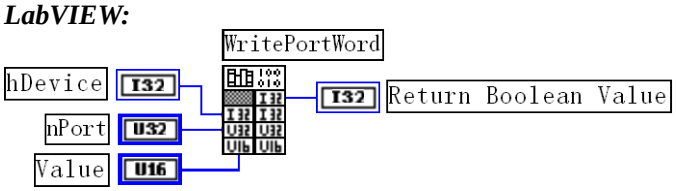
函数原型:
Visual C++:

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

Visual Basic:

```
Declare Function WritePortWord Lib"PXI8008_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal pbPort As Long,_
                                    ByVal OffsetBytes As Long,_
                                    ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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 GetLastError 捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

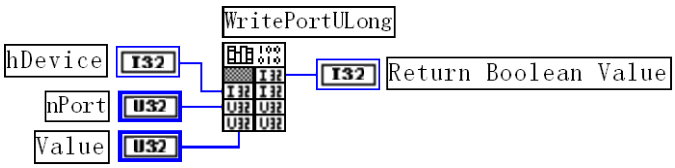
函数原型:
Visual C++:

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

Visual Basic:

```
Declare Function WritePortULong Lib "PXI8008_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal pbPort As Long,_
                                    ByVal OffsetBytes As Long, _
                                    ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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 GetLastError 捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

♦ **以单字节(8Bit)方式读 I/O 端口**

函数原型:

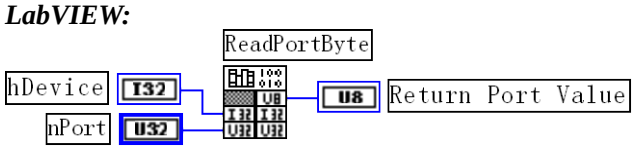
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortByte Lib "PXI8008_32" (_
                                   ByVal hDevice As Long,_
                                   ByVal pbPort As Long,_
                                   ByVal OffsetBytes As Long) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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:

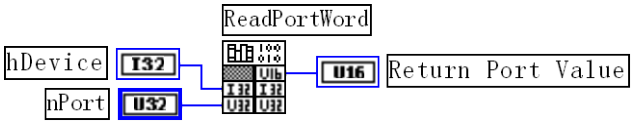
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortWord Lib "PXI8008_32" (_
                                   ByVal hDevice As Long,_
                                   ByVal pbPort As Long,_
                                   ByVal OffsetBytes As Long) As Integer
```

LabVIEW:



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

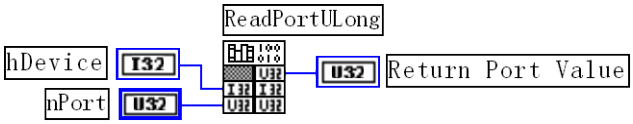
函数原型:
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortULong Lib"PXI8008_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal pbPort As Long,_
                                        ByVal OffsetBytes As Long) As Long
```

LabVIEW:



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

第四节、线程操作函数原型说明

- **创建内核系统事件**

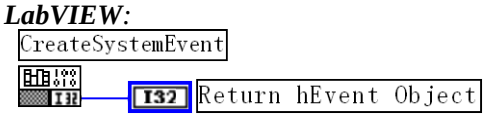
函数原型：
Visual C++:

```
HANDLE CreateSystemEvent(void)
```

Visual Basic:

```
Declare Function CreateSystemEvent Lib " PXI8008 " () As Long
```

LabVIEW:



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

- **释放内核系统事件**

函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function ReleaseSystemEvent Lib " PXI8008 " (ByVal hEvent As Long) As Boolean
```

LabVIEW:
请参见相关演示程序。

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