PXI8506 同步采集卡

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》，它会使您事半功倍!

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 PXI8506_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 InitDeviceAD、ReadDeviceAD 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 InitDeviceAD 可以使用 hDevice 句柄以程序查询方式初始化设备的 AD 部件，ReadDeviceAD 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、如何取得 AD 数据

当您有了 hDevice 设备对象句柄后，便可用 InitDeviceAD 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。然后用 StartDeviceAD 即可启动 AD 部件，开始 AD 采样，然后便可用 ReadDeviceAD 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 StopDeviceAD，当您需要关闭 AD 设备时，ReleaseDeviceAD 便可帮您实现（但设备对象 hDevice 依然存在）。

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。

第四节、哪些函数对您不是必须的

如果您使用上层用户函数访问设备，那么 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。

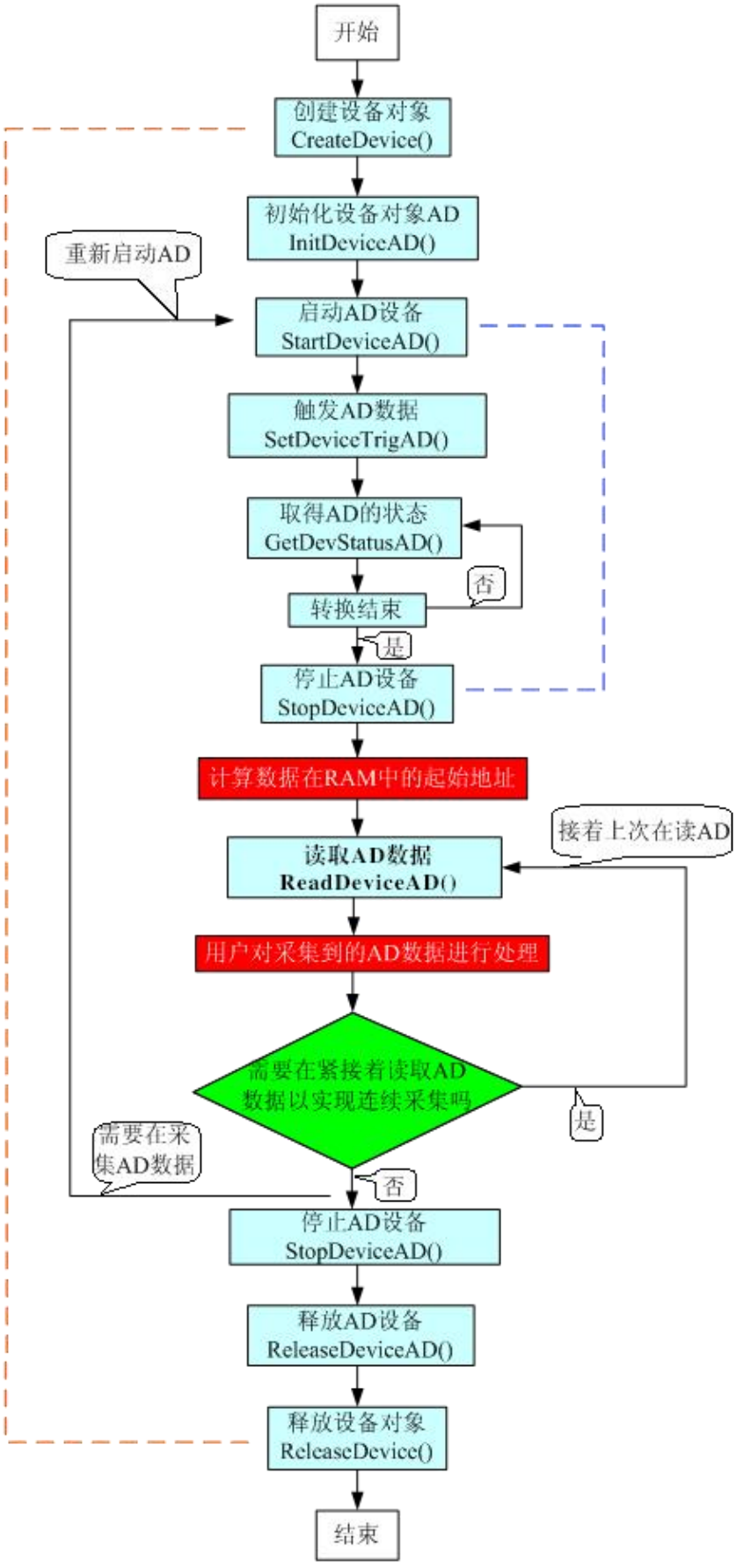

图 2.1.1　如何取得 AD 数据

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，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，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 InitDeviceAD）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 ReadDeviceAD 函数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 GetDeviceBar 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。

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PXI8506_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PXI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PXI 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PXI 总线设备对象	上层及底层用户
②AD 数据读取函数		
GetDDR2Length	返回板载 DDR2 大小，单位为 Mb	上层用户
ADCalibration	AD 校准	上层用户
InitDeviceAD	初始化设备，当返回 TRUE 后，设备即刻开始传输	上层用户
StartDeviceAD	在初始化之后，启动设备	上层用户
SetDeviceTrigAD	当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）	上层用户
GetDevStatusAD	取得当前设备状态	上层用户
ReadDeviceAD	DMA 方式读 AD 数据	上层用户
StopDeviceAD	在启动设备之后，暂停设备	上层用户
ReleaseDeviceAD	关闭 AD 设备，禁止传输，且释放资源	上层用户
③ AD 硬件参数系统保存、读取函数		
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
LoadParaAD	从 Windows 系统中读入硬件参数	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，必须在您的源程序中包含如下语句：

#include “C:\Art\PXI8506\INCLUDE\PXI8506.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 PXI8506.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。

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。

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 DeviceID = 0)

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：DeviceID 设备 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 DeviceID 标识值为名称后缀的标识符来确认和管理该设备。默认值为 0。

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： GetDeviceCount　　GetDeviceCurrentID
ListDeviceDlg　　ReleaseDevice

Visual C++ 程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PXI8506_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
 }
    :
```

♦ 取得本计算机系统中 PXI8506 设备的总数量

函数原型：

Visual C++ :

int GetDeviceCount (HANDLE hDevice)

功能：取得 PXI8506 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 PXI8506 的数量。

相关函数： CreateDevice　　GetDeviceCurrentID
ListDeviceDlg　　ReleaseDevice

♦ 用对话框控件列表计算机系统中所有 PXI8506 设备各种配置信息

函数原型：

Visual C++ :

BOOL ListDeviceDlg (HANDLE hDevice)

功能：列表系统中 PXI8506 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 PXI8506 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)

功能：释放设备对象所占用的系统资源及设备对象自身。

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

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

相关函数： CreateDevice

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 数据读取函数原型说明

♦ **返回板载 DDR2 大小，单位为 Mb**

函数原型：

Visual C++:

```
BOOL GetDDR2Length(HANDLE hDevice,
                   PULONG pulDDR2Length)
```

功能：返回板载 DDR2 的长度

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

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

相关函数：CreateDevice

♦ **AD 校准**

函数原型：

Visual C++:

```
BOOL ADCalibration(HANDLE hDevice)
```

功能：AD 校准

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

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

相关函数：CreateDevice

♦ **初始化设备对象**

函数原型：

Visual C++:

```
BOOL InitDeviceAD(HANDLE hDevice,
                  PPXI8506_PARA_AD pADPara)
```

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，然后启动 AD 设备开始 AD 采集，随后，用户便可以连续调用 ReadDeviceAD 读取设备上的 AD 数据以实现连续采集。

参数：

hDevice 设备对象句柄，它应由设备的 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。请参考《AD 硬件参数介绍》。

返回值：如果初始化设备对象成功，则返回 TRUE，且 AD 便被启动。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 StartDeviceAD SetDeviceTrigAD GetDevStatusAD ReadDeviceAD ReleaseDevice topDeviceAD ReleaseDeviceAD

注意：该函数将试图占用系统的某些资源，如系统内存区、DMA 资源等。所以当用户在反复进行数据采集之前，只须执行一次该函数即可，否则某些资源将会发生使用上的冲突，便会失败。除非用户执行了 ReleaseDeviceAD 函数后，再重新开始设备对象操作时，可以再执行该函数。所以该函数切忌不要单独放在循环语句中反复执行，除非和 ReleaseDeviceAD 配对。

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

函数原型：

Visual C++:

```
BOOL StartDeviceAD (HANDLE hDevice)
```

功能：启动 AD 设备，它必须在调用 InitDeviceAD 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 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 InitDeviceAD
SetDeviceTrigAD GetDevStatusAD ReadDeviceAD
ReleaseDevice topDeviceAD ReleaseDeviceAD

♦ 产生软件触发事件

函数原型：

Visual C++

```
BOOL SetDeviceTrigAD (HANDLE hDevice)
```

功能：当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）。

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

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

相关函数： CreateDevice InitDeviceAD StartDeviceAD
GetDevStatusAD ReadDeviceAD
ReleaseDevice topDeviceAD ReleaseDeviceAD

♦ 取得 AD 状态标志

函数原型：

Visual C++

```
BOOL GetDevStatusAD (HANDLE hDevice,
                     PPXI8506_STATUS_AD);
```

功能：一旦用户使用 StartDeviceAD 后，应立即用此函数查询存储器的状态。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 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《PXI8506_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。

相关函数： CreateDevice InitDeviceAD StartDeviceAD
SetDeviceTrigAD ReadDeviceAD
ReleaseDevice topDeviceAD ReleaseDeviceAD

♦DMA 方式读 AD 数据

函数原型：

Visual C++:

```
BOOL ReadDeviceAD(HANDLE hDevice,
                  PWORD pADBuffer,
                  ULONG nReadSizeWords,
                  ULONG ulStartAddr
                  PLONG nRetSizeWords);
```

功能：DMA 方式读 AD 数据

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pADBuffer 将用于接受数据的用户缓冲区(该区必须开辟大于 M 加 N 个字的空间)。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 ReadDeviceAD 操作应读取多少字数据到用户缓冲区。

ulStartAddr 数据在 RAM 中的起始地址

nRetSizeWords 返回实际读取的数据长度。

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 ReleaseDeviceAD 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。

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。

相关函数: CreateDevice InitDeviceAD StartDeviceAD SetDeviceTrigAD GetDevStatusAD ReleaseDevice topDeviceAD ReleaseDeviceAD

♦ **暂停 AD 设备**

函数原型：

Visual C++:

BOOL StopDeviceAD (HANDLE hDevice)

功能：在启动设备之后，暂停设备。它必须在调用 StartDeviceAD 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 StartDeviceAD 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 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 InitDeviceAD StartDeviceAD SetDeviceTrigAD GetDevStatusAD ReadDeviceAD ReleaseDevice ReleaseDeviceAD

♦ **释放设备上的 AD 部件**

函数原型：

Visual C++:

BOOL ReleaseDeviceAD(HANDLE hDevice)

功能：关闭 AD 设备，禁止传输，且释放资源。

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

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

应注意的是，InitDeviceAD 必须和 ReleaseDeviceAD 函数一一对应，即当您执行了一次 InitDeviceAD 后，再一次执行这些函数前，必须执行一次 ReleaseDeviceAD 函数，以释放由 InitDeviceAD 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 InitDeviceAD 函数时，那些软硬件资源才可被再次使用。

相关函数: CreateDevice InitDeviceAD ReleaseDevice

♦ **AD 读取函数一般调用顺序**

① CreateDevice
② InitDeviceAD
③ StartDeviceAD
④ GetDevStatusAD
⑤ StopDeviceAD
⑥ ReadDeviceAD
⑦ ReleaseDeviceAD
⑧ ReleaseDevice

注明：用户可以反复执行第⑥步，以实现采集。

关于两个过程的图形说明请参考《使用纲要》。

第四节、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,
                PXI8506_PARA_AD pADPara)
```

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 PXI8506_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 PXI8506_PARA_AD 请参考 PXI8506.h 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++ :

```
BOOL SaveParaAD (HANDLE hDevice,
                 PXI8506_PARA_AD pADPara)
```

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 PXI8506_PARA_AD 的详细介绍请参考 PXI8506.h 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++ :

```
BOOL ResetParaAD (HANDLE hDevice,
                  PXI8506_PARA_AD pADPara)
```

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 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 PXI8506_PARA_AD 的详细介绍请参考 PXI8506.h 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 硬件参数介绍（PXI8506_PARA_AD）

Visual C++ :

```
typedef struct _PXI8506_PARA_AD
{
    LONG Frequency;          // 采集频率,单位为 Hz, [10, 40000000]
    LONG InputRange[4];      // 模拟量输入量程选择
    LONG CouplingType;       // 耦合类型(直流耦合, 交流耦合)
    LONG M_Length;           // M 段长度(字)
    LONG N_Length;           // N 段长度(字)
    LONG TriggerMode;        // 触发模式选择
    LONG ATRTriggerChannel;  // ATR 通道选择
    LONG TriggerSource;      // 触发源选择
    LONG TriggerDir;         // 触发方向选择(正向/负向触发)
    LONG TrigLevelVolt;      // 触发电平(量程按模拟输入量程)
    LONG ClockSource;        // 时钟源选择(内/外时钟源)
```

LONG OutClockSource;// 时钟输入输出源
LONG bClockSourceDir; // 是否将时钟信号输出到 PXI 总线,=TRUE:允许输出, =FALSE:允许输入
} PXI8506_PARA_AD, *PPXI8506_PARA_AD;

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 InitDeviceAD 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。

Frequency 采集频率，单位为 Hz，取值范围为[10,40000000]。

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义
PXI8506_INPUT_N1000_P1000mV	0x00	±1000mV
PXI8506_INPUT_N5000_P5000mV	0x01	±5000mV

CouplingType 耦合类型(直流耦合, 交流耦合)。

常量名	常量值	功能定义
PXI8506_COUPLING_AC	0x00	交流耦合
PXI8506_COUPLING_DC	0x01	直流耦合

M_Length M 段长度(字)，总的取值范围 1-16383，但是 M 加 N 长度不能大于 16384。
N_Length N 段长度(字)，总的取值范围 1-16383，但是 M 加 N 长度不能大于 16384。

TriggerMode AD 触发模式。

常量名	常量值	功能定义
PXI8506_TRIGMODE_MIDL	0x00	中间触发
PXI8506_TRIGMODE_POST	0x01	后触发
PXI8506_TRIGMODE_PRE	0x02	预触发
PXI8506_TRIGMODE_DELAY	0x03	硬件延时触发

ATRTriggerChannel ATR 通道选择。

常量名	常量值	功能定义
PXI8506_ATRTRIG_CH0	0x00	通道 0 信号作为 ATR 输入
PXI8506_ATRTRIG_CH1	0x01	通道 1 信号作为 ATR 输入
PXI8506_ATRTRIG_CH2	0x02	通道 2 信号作为 ATR 输入
PXI8506_ATRTRIG_CH3	0x03	通道 3 信号作为 ATR 输入

TriggerSource AD 触发源。

常量名	常量值	功能定义
PXI8506_TRIGMODE_SOFT	0x00	软件触发
PXI8506_TRIGSRC_DTR	0x01	选择 DTR 作为触发源
PXI8506_TRIGSRC_ATR	0x02	选择 ATR 作为触发源
PXI8506_TRIGSRC_TRIGGER	0x03	Trigger 信号触发(用于多卡同步)

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
PXI8506_TRIGDIR_NEGATIVE	0x00	下降沿触发
PXI8506_TRIGDIR_POSITIVE	0x01	上升沿触发
PXI8506_TRIGDIR_NEGAT_POSIT	0x02	上下边沿均触发

注明：PXI8506_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。

TrigLevelVolt 触发电平，量程与模拟触发通道一致。

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
PXI8506_CLOCKSRC_IN	0x00	使用内部时钟
PXI8506_CLOKCSRC_PXI10M	0x01	使用用 PXI 背板 10M 时钟
PXI8506_CLOCKSRC_OUT	0x02	使用外部时钟

OutClockSource 时钟输入输出源。

常量名	常量值	功能定义
PXI8506_OUTCLOCKSRC_TRIGGER0	0x00	选择 PXI 总线上的 TRIG0 输入
PXI8506_OUTCLOCKSRC_TRIGGER1	0x01	选择 PXI 总线上的 TRIG1 输入
PXI8506_OUTCLOCKSRC_TRIGGER2	0x02	选择 PXI 总线上的 TRIG2 输入
PXI8506_OUTCLOCKSRC_TRIGGER3	0x03	选择 PXI 总线上的 TRIG3 输入
PXI8506_OUTCLOCKSRC_TRIGGER4	0x04	选择 PXI 总线上的 TRIG4 输入
PXI8506_OUTCLOCKSRC_TRIGGER5	0x05	选择 PXI 总线上的 TRIG5 输入
PXI8506_OUTCLOCKSRC_TRIGGER6	0x06	选择 PXI 总线上的 TRIG6 输入
PXI8506_OUTCLOCKSRC_TRIGGER7	0x07	选择 PXI 总线上的 TRIG7 输入
PXI8506_OUTCLOCKSRC_PXISTAR	0x08	PXI_STAR 信号触发

bClockSourceDir 是否将时钟信号输出到 PXI 总线，=TRUE：允许输出，=FALSE：允许输入。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第二节、AD 状态参数结构（PXI8506_STATUS_AD）

Visual C++ ：

```
typedef struct _PXI8506_STATUS_AD
    {
    LONG bADEanble;// AD 是否已经使能，=TRUE:表示已使能，=FALSE:表示未使能
    LONG bTrigger;   // AD 是否被触发，=TRUE:表示已被触发，=FALSE:表示未被触发
    LONG bComplete; // AD 是否整个转换过程是否结束，=TRUE:表示已结束，=FALSE:表示未结束
    LONG bAheadTrig;// AD 触发点是否提前，=TRUE:表示触发点提前，=FALSE:表示触发点未提前
    LONG lEndAddr;   // 数据完成的结束地址
    } PXI8506_STATUS_AD, *PPXI8506_STATUS_AD;
```

此结构体主要用于查询 AD 的各种状态，GetDevStatusAD 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。

bADEanbleAD 是否已经使能，=TRUE：表示已使能，=FALSE：表示未使能。
bTrigger AD 是否被触发，=TRUE：表示已被触发，=FALSE：表示未被触发。
bComplete AD 是否整个转换过程是否结束，=TRUE：表示已结束，=FALSE：表示未结束。
bAheadTrigAD 触发点是否提前，=TRUE：表示触发点提前，=FALSE：表示触发点未提前。

相关函数： CreateDevice GetDevStatusAD 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）
±5000mV	Volt=（10000.00/65536）*(ADBuffer[0]&0xFFFF)-5000.00	[-5000, +4999.85]
±1000mV	Volt=（2000.00/65536）*(ADBuffer[0]&0xFFFF)-1000.00	[-1000, +999.97]

下面举例说明各种语言的换算过程（以±5000mV 量程为例）

Visual C++:

```
Lsb = ADBuffer[0]&0xFFFF;
```

Volt = (10000.00/65536) * Lsb -5000.00;

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

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

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)

    LONG ChannelCount;                  // 通道总数
    LONG DataWidth;                     // 设备的精度(分辨率)
    LONG bXorHighBit;                   // 是否高位求反(为 1 则求反)
    PPXI8506_PARA_AD ADPara;            // 保存硬件参数
    PPXI8506_STATUS_AD ADStatus;        // 保存硬件参数
    LONG CrystalFreq;                   // 晶振频率
    LONG ChannelNum;                    // 通道号
    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、简易程序演示说明

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**[PXI8506 AD(V6.00.00)]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[AD 简易应用程序]**

第二节、高级程序演示说明

高级程序演示了本设备的所有功能，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PXI8506.h 和 ADDoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[PXI8506 AD(V6.00.00)]**丨**[Microsoft Visual C++]**丨[高级代码演示]

其默认存放路径为：系统盘\ART\PXI8506\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第七章 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，PXI 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 PXI 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 ReadDeviceAD 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 ReadDeviceAD 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 InitDeviceAD 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。

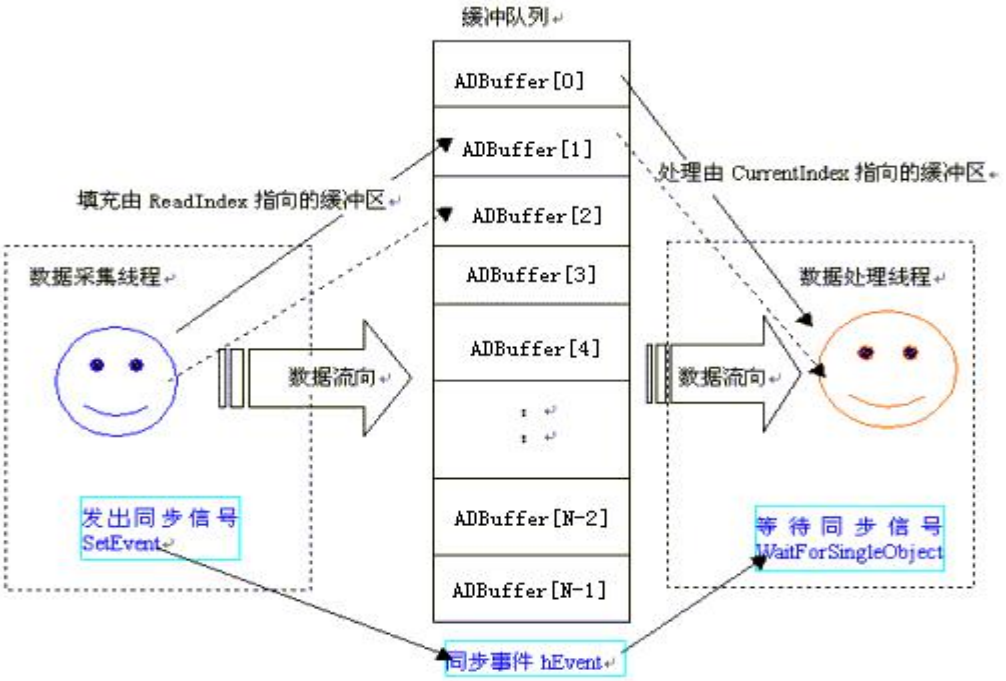

图 7.1

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“PXI8506_”）

函数名	函数功能	备注
① PXI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
GetDevVersion	获取设备固件及程序版本	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

- **取得指定的指定设备寄存器组 BAR 地址**

函数原型：

Visual C++ :

BOOL GetDeviceBar (HANDLE hDevice,

PUCHAR pulPXIBar[6])

功能：取得指定的指定设备寄存器组 BAR 地址。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pulPCIBar 返回 PXI BAR 所有地址。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice ReleaseDevice

♦ **获取设备固件及程序版本**

函数原型：
Visual C++ :

```
BOOL GetDevVersion ( HANDLE hDevice,
                     PULONG pulFmwVersion,
                     PULONG pulDriverVersion)
```

功能：获取设备固件及程序版本。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pulFmwVersion 指针参数，用于取得固件版本。
pulDriverVersion 指针参数，用于取得驱动版本。
返回值：如果执行成功，则返回 TRUE，否则会返回 FALSE。
相关函数： CreateDevice 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,
                        PUCHAR pbLinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

功能：以单字节（即 8 位）方式写 PXI 内存映射寄存器。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbLinearAddr 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 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
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

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

函数原型:

Visual C++ :

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

功能：以双字节（即 16 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 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, WriteRegisterByte, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

Visual C++ 程序举例:

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

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

函数原型:

Visual C++ :

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

功能：以四字节（即 32 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 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, WriteRegisterByte, WriteRegisterWord, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

Visual C++ 程序举例:

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

◆ **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++ :

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

功能：以单字节（即 8 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 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　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ 程序举例:

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

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

函数原型:

Visual C++ :

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

功能：以双字节（即 16 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 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 WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterULong ReleaseDevice

Visual C++ 程序举例:

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

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

函数原型:

Visual C++ :

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

功能：以四字节（即 32 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 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 WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord 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
:
```

第三节、I/O 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**

Visual C++ :

```
BOOL WritePortByte (HANDLE hDevice,
                    PUCHAR pbPort,
```

ULONG OffsetBytes,
BYTE Value)

功能：以单字节(8Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
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 WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

Visual C++ :

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

功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
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
WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

Visual C++ :

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

功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
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
ReadPortByte ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

Visual C++:

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

功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortWord

◆ **以双字节(16Bit)方式读 I/O 端口**

Visual C++ :

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

功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte

◆ **以四字节(32Bit)方式读 I/O 端口**

Visual C++ :

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

功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pbPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
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）

◆ **创建内核系统事件**

函数原型：

Visual C++ :

```
HANDLE CreateSystemEvent(void)
```

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)
```

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。