PXI/PCI85XXB 高速数字化仪

驱动程序使用手册

北京阿尔泰科技发展有限公司

V6.05.01

阿尔泰科技
www.art-control.com

关于本手册

本手册为阿尔泰科技推出的 PXI8502B、PXI8504B、PXI8512B、PXI8514B、PCI8502B、PCI8504B、PCI8512B、PCI8514B 高速数字化仪驱动程序通用使用手册，以下简称 PXI/PCI85xxB，其中包括版权信息与命名约定、使用纲要、各功能操作流程介绍、设备操作函数接口介绍、上层用户函数接口应用实例、共用函数介绍、修改历史等。

文档版本：V6.05.01

目　录

1 版权信息与命名约定

1.1 版权信息

1.2 命名约定

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 PXI/PCIxxxx 则被省略，如 PCI8502B_CreateDevice 则写为 CreateDevice。

表 1-2-1：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。

2 使用纲要

2.1 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、ReadRegisterByte 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。

2.2 如何管理设备

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。

2.3 怎么进行 AD 数采操作

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）。具体执行流程请看图 2.1.2。

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。

2.4 哪些函数对您不是必须的

公共函数一般来说都是辅助性函数，这些函数您可完全不必理会，除非您是作为底层用户管理设备。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。对 PCI 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。

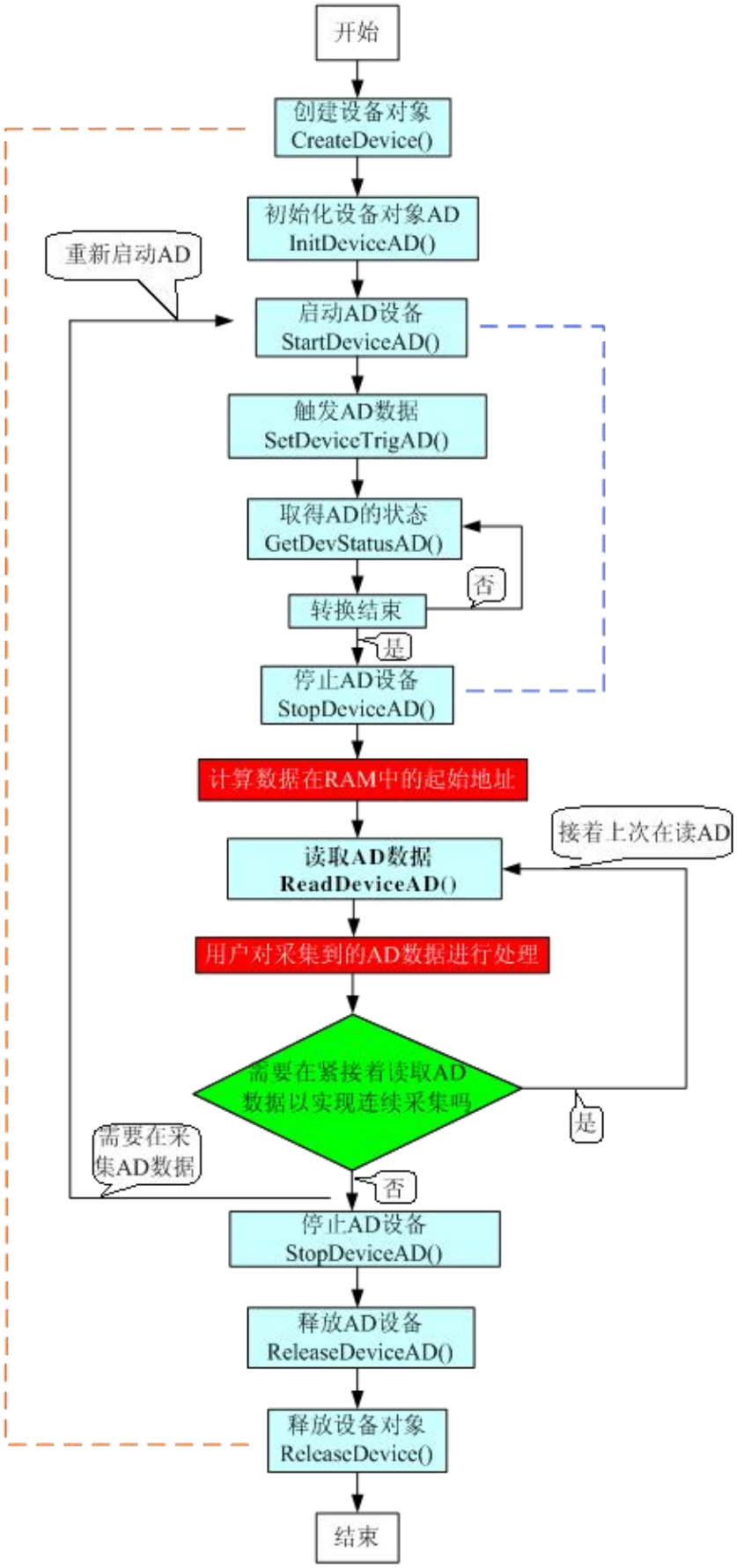


图 2.1.2　怎么进行 AD 数采操作

3 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 PCI 总线复杂的控制协议，同时还可以省掉您许多繁琐的工作。

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。

3.1 设备驱动接口函数总列表

表 3-1-1：驱动接口函数总列表（每个函数省略了前缀“PXI/PCI85xxB”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建设备对象	上层及底层用户
CreateDeviceEx	创建设备对象(该函数使用物理 ID 最大 255)	上层及底层用户
GetDeviceCount	取得设备总台数	上层及底层用户
SetDevicePhysID	设置当前设备的物理 ID 号[0:255]	上层及底层用户
GetDeviceCurrentID	取得当前设备的逻辑 ID 号和物理 ID 号	上层及底层用户
ListDeviceDlg	列表系统当中的所有的该 PCI 设备	上层及底层用户
ReleaseDevice	关闭设备，禁止传输，且释放资源	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 硬件参数操作函数		
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\PXI/PCI85xxB\INCLUDE\PXI/PCI85xxB.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 PXI/PCI85xxB.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。

3.2 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： CreateDeviceEx　GetDeviceCount　SetDevicePhysID　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例：

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

♦ **创建设备对象(该函数使用物理 ID 最大 255）**

函数原型：

Visual C++:

HANDLE CreateDeviceEx(LONG DevicePhysID)

功能：创建设备对象。

参数：

DevicePhysID： 物理设备 ID(Physic Device Identifier)标识号。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。

相关函数： CreateDevice GetDeviceCount SetDevicePhysID GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **取得设备总台数**

函数原型：

Visual C++:

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

功能：取得设备总台数。

参数：

hDevice： 设备对象句柄，它应由 CreateDevice 创建。

返回值：返回系统中设备的数量。

相关函数： CreateDevice CreateDeviceEx SetDevicePhysID GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **设置当前设备的物理 ID 号[0:255]**

函数原型：

Visual C++:

```
BOOL SetDevicePhysID (HANDLE hDevice,
                      LONG DevicePhysID)
```

功能：设置当前设备的物理 ID 号[0:255]

参数：

DevicePhysID： 物理设备 ID(Physic Device Identifier)标识号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReleaseDevice

- **取得当前设备的逻辑 ID 号和物理 ID 号**

函数原型：

Visual C++:

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

功能：取得当前设备的逻辑 ID 号和物理 ID 号

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： 物理设备 ID(Physic Device Identifier)标识号。

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

♦ **列表系统当中的所有的该 PCI 设备**

函数原型：

Visual C++:

BOOL ListDeviceDlg(HANDLE hDevice)

功能：列表系统当中的所有的该 PCI 设备

参数：

hDevice： 设备对象句柄，它指向要取得逻辑号的设备，它应由 CreateDevice 创建。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

相关函数： CreateDevice GetDeviceCount SetDevicePhysID GetDeviceCurrentID CreateDeviceEx ReleaseDevice

♦ **关闭设备，禁止传输，且释放资源**

函数原型：

Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

功能：关闭设备，禁止传输，且释放资源。

参数：

hDevice： 设备对象句柄，它指向要取得逻辑号的设备，它应由 CreateDevice 创建。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

相关函数： CreateDevice GetDeviceCount SetDevicePhysID GetDeviceCurrentID ListDeviceDlg CreateDeviceEx

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。

3.3 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。

pulDDR2Length： DDR2 的长度(单位：MB)

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,
PPARA_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,
                     PSTATUS_AD pADStatus);
```

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《AD 状态参数结构（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,
                  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。必须等于 M 加 N 的长度。

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 StopDeviceAD 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》。

3.4 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,
                PPARA_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 PARA_AD 的结构指针类型，它负责返回 PXI/PCI 硬件参数值，关于结构指针类型 PARA_AD 请参考 PXI/PCI85xxB.h 或 PXI/PCI85xxB.Bas 或 PXI/PCI85xxB.Pas 函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice 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,
                 PPARA_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 PARA_AD 的详细介绍请参考 PXI/PCI85xxB.h 或 PXI/PCI85xxB.Bas 或 PXI/PCI85xxB.Pas 函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD 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,
PPARA_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 PARA_AD 的详细介绍请参考 PXI/PCI85xxB.h 或 PXI/PCI85xxB.Bas 或 PXI/PCI85xxB.Pas 函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD
ReleaseDevice

4 硬件参数结构

4.1 AD 硬件参数介绍（PARA_AD）

Visual C++:

```
typedef struct _PARA_AD
{
        LONG Frequency;
        LONG bChannelArray[4];
        LONG InputRange[4];
        LONG CouplingType[4];
        LONG M_Length;
        LONG N_Length;
        LONG TriggerMode;
        LONG ATRTriggerChannel;
        LONG TriggerSource;
        LONG TriggerDir;
        LONG TrigLevelVolt;
        LONG TrigCount;
        LONG ClockSource;
        LONG bMasterEn;
        LONG SyncTrigSignal;
} PARA_AD, *PPARA_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， 其中 PXI/PCI 8502B/8504B 采样范围为[10, 40000000]；PXI/PCI 8512B/8514B 采样范围为[10, 80000000]。

bChannelArray 采样通道选择阵列，分别控制 4 个通道，=TRUE 表示该通道采样，否则不采样(只支持 3 种通道配置:0 01 0123)。

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义
INPUT_N1000_P1000mV	0x00	±1000mV
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
COUPLING_AC	0x00	交流耦合
COUPLING_DC	0x01	直流耦合

M_Length M 段长度(字)，延时触发 M 表示延时点数范围[0, 2147483647]。

N_Length N 段长度(字)，中间触发 M N 有效读取点数；

后触发 M 必须为 0，N 有效读取点数；

预触发 M 必须为 0，N 有效读取点数；

硬件延时触发 N 有效读取点数；

各种触发下有效读取点数相加范围[16/使能通道数， 内存大小(字节)/2/使能通道数]， 有效读取点数相加必须为(16/使能通道数)的整数倍。

TriggerMode 触发模式选择。

常量名	常量值	功能定义
TRIGMODE_MIDL	0x00	中间触发：M 为触发前采集点数，N 为触发后采集点数
TRIGMODE_POST	0x01	后触发：M 无效，N 为触发后采集点数
TRIGMODE_PRE	0x02	预触发：M 无效，N 为触发前采集点数
TRIGMODE_DELAY	0x03	硬件延时触发：M 为触发后延时点数，N 为延时后采集点数

ATRTriggerChannel ATR 通道选择。

常量名	常量值	功能定义
ATRTRIG_CH0	0x00	通道 0 信号作为 ATR 输入
ATRTRIG_CH1	0x01	通道 1 信号作为 ATR 输入
ATRTRIG_CH2	0x02	通道 2 信号作为 ATR 输入
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
TRIGMODE_SOFT	0x00	软件触发
TRIGSRC_DTR	0x01	选择 DTR 作为触发源
TRIGSRC_ATR	0x02	选择 ATR 作为触发源
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
TRIGDIR_NEGATIVE	0x00	下降沿触发
TRIGDIR_POSITIVE	0x01	上升沿触发
TRIGDIR_NEGAT_POSIT	0x02	上下边沿均触发

注明：TRIGDIR_NEGAT_POSIT 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(量程按模拟输入量程)。

TrigCount; 触发次数设置，默认为 1 次，此功能仅在后触发和硬件延时触发模式下有效

ClockSource 时钟源选择 (内/外时钟源/10M)。

常量名	常量值	功能定义
CLOCKSRC_IN	0x00	使用内部时钟

CLOKCSRC_10M	0x01	使用主卡 10M 时钟
CLOCKSRC_OUT	0x02	使用外部时钟

bMasterEn 主设备使能。

=0：从设备，通过 Trigger 信号接收主设备发送的同步触发信号

=1：主设备，通过 Trigger 信号为从设备发送自身的触发信号

注：在多模块同步系统中，只能设定其中一个设备为主设备，其余需设定为从设备，如果系统中只有一个设备或者有多个设备但是不要求同步，需将所有设备设定为从设备

SyncTrigSignal 同步触发信号

常量名	常量值	功能定义
STS_TRIGGER0	0x00	同步触发信号 TRIG0
STS_TRIGGER1	0x01	同步触发信号 TRIG1
STS_TRIGGER2	0x02	同步触发信号 TRIG2
STS_TRIGGER3	0x03	同步触发信号 TRIG3
STS_TRIGGER4	0x04	同步触发信号 TRIG4
STS_TRIGGER5	0x05	同步触发信号 TRIG5
STS_TRIGGER6	0x06	同步触发信号 TRIG6
STS_TRIGGER7	0x07	同步触发信号 TRIG7

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

4.2 AD 状态参数结构（STATUS_AD）

Visual C++：

```
typedef struct _STATUS_AD
{
    LONG bADEanble;
    LONG bTrigger;
    LONG bComplete;
    LONG bAheadTrig;
    LONG lCanReadPoint;
} STATUS_AD, *PSTATUS_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。

bADEanble AD 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。

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。

lCanReadPoint 可以读取的点数。

相关函数： CreateDevice GetDevStatusAD ReleaseDevice

5 数据格式转换与排列规则

5.1 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。

PXI/PCI 8502B/8512B 换算方法为：

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±5000mV	Volt = (10000.00/4096) * (ADBuffer[0]&0xFFF) – 5000.00	[-5000, +4997.55]
±1000mV	Volt = (2000.00/4096) * (ADBuffer[0]&0xFFF) – 1000.00	[-1000, +999.51]

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]&0xFFF;
    Volt = (10000.00/4096) * Lsb -5000.00;
```

PXI/PCI 8504B/8514B 换算方法为：

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±5000mV	Volt = (10000.00/16384) * (ADBuffer[0]&0x3FFF) – 5000.00	[-5000, +4999.38]
±1000mV	Volt = (2000.00/16384) * (ADBuffer[0]&0x3FFF) – 1000.00	[-1000, +999.87]

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]& 0x3FFF;
    Volt = (10000.00/16384) * Lsb -5000.00;
```

5.2 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	…

5.3 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;
    LONG FileType;
```

```
    LONG BusType;
    LONG DeviceNum;
    LONG HeadVersion;
    LONG VoltBottomRange;
    LONG VoltTopRange;
    LONG ChannelCount;
    LONG DataWidth;
    LONG bXorHighBit;
    PARA_AD ADPara;
    STATUS_AD ADStatus;
    LONG CrystalFreq;
    LONG ChannelNum;
    LONG HeadEndFlag;
} 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。

6 上层用户函数接口应用实例

6.1 简易程序演示说明

怎么进行 AD 数采操作

Visual C++:

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

[程序]丨**[阿尔泰测控演示系统]**丨**[PXI/PCI85xxB AD]**丨**[Microsoft　VS2005]**丨**[简易代码演示]**丨

6.2 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 PXI/PCI85xxB.h 和 ADDoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[PXI/PCI85xxB AD]**丨**[Microsoft VS2005]**丨**[高级代码演示]**

其默认存放路径为：系统盘\ART\PXI/PCI85xxB\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

7 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。

7.1 公用接口函数总列表

表 7-1-1：公用接口函数总列表（每个函数省略了前缀“PXI/PCI85xxB”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
GetDevVersion	获取设备固件及程序版本	底层用户
WriteRegisterByte	往设备的映射寄存器空间指定端口写入单节字数据	底层用户
WriteRegisterWord	写双字节数据（其余同上）	底层用户
WriteRegisterULong	写四节字数据（其余同上）	底层用户
ReadRegisterByte	读入单字节数据（其余同上）	底层用户
ReadRegisterWord	读入双字节数据（其余同上）	底层用户
ReadRegisterULong	读入四字节数据（其余同上）	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 线程操作函数		
CreateSystemEvent	创建系统内核事件对象，供 InitDeviceInt 和 VB 子线程等函数使用	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

7.2 PCI 内存映射寄存器操作函数原型说明

♦ **取得指定的指定设备寄存器组 BAR 地址**

函数原型：

Visual C++:

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

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 PCI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice 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

♦ **往设备的映射寄存器空间指定端口写入单节字数据**

函数原型:

Visual C++:

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

功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

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

Value ：　输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数:　CreateDevice　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
    :
```

♦ **写双字节数据（其余同上）**

函数原型:

Visual C++:

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

功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

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

Value ：　输出 16 位整型值。

返回值：无。

相关函数:　CreateDevice　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
    :
```

♦ **写四节字数据（其余同上）**

函数原型:

Visual C++:

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

功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

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

Value：　输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数:　CreateDevice　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
    :
```

♦ **读入单字节数据（其余同上）**

函数原型:

Visual C++:

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

功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

OffsetBytes：　相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 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　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ 程序举例:

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

◆ **读入双字节数据（其余同上）**

函数原型:

Visual C++:

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

功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

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

Visual C++程序举例:

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

◆ **读入四字节数据（其余同上）**

函数原型:

Visual C++:

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

功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice：　设备对象句柄，它应由 CreateDevice 创建。

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

OffsetBytes：　相对与 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 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　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

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

7.3　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 WritePortByte 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 WritePortWord
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
WritePortULong 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　ReadPortByte　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　ReadPortWord

- **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

7.4　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。

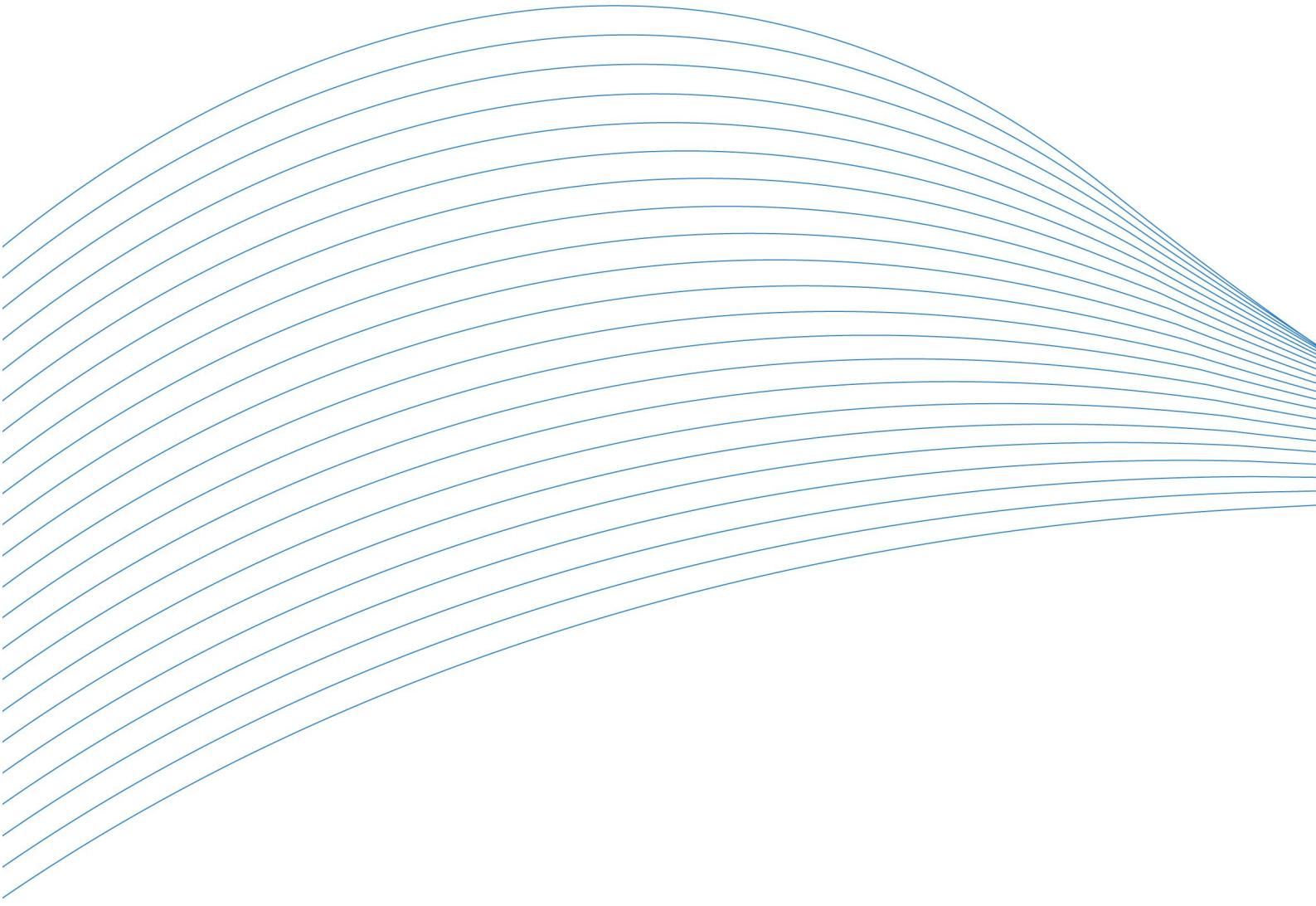
参数：

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。

8 修改历史

修改时间	版本号	修改内容
2016.11.30	V6.05.00	第一版
2017.1.12	V6.05.01	修改 trigger mode 注释，修改 M_Length\N_Length 注释



北京阿尔泰科技发展有限公司
服务热线：400-860-3335
邮编：100086
传真：010-62901157