ACTS1001 同步采集卡

驱动程序使用手册

北京阿尔泰科技发展有限公司

V6.00.00

ART Technology

前言

免责说明

订购产品前，请向厂家或经销商详细了解产品性能是否符合您的需求。

正确的运输、储存、组装、装配、安装、调试、操作和维护是产品安全、正常运行的前提。本公司对于任何因安装、使用不当而导致的直接、间接、有意或无意的损坏及隐患概不负责。

安全使用小常识

1.在使用产品前，请务必仔细阅读产品使用手册；

2.对未准备安装使用的产品，应做好防静电保护工作(最好放置在防静电保护袋中，不要将其取出)；

3.在拿出产品前，应将手先置于接地金属物体上，以释放身体及手中的静电，并佩戴静电手套和手环，要养成只触及其边缘部分的习惯；

4.为避免人体被电击或产品被损坏，在每次对产品进行拔插或重新配置时，须断电；

5.在需对产品进行搬动前，务必先拔掉电源；

6.对整机产品，需增加/减少板卡时，务必断电；

7.当您需连接或拔除任何设备前，须确定所有的电源线事先已被拔掉；

8.为避免频繁开关机对产品造成不必要的损伤，关机后，应至少等待 30 秒后再开机。

目 录

1 版权信息与命名约定

1.1 版权信息

本软件产品及相关套件均属北京阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位、我公司授权的代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，我们将热情接待。

1.2 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 PCIexxxx_则被省略。如 ACTS1001_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。

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、StartDeviceAD 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。

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 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 InitDeviceAD 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 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：ReadDeviceAD 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。

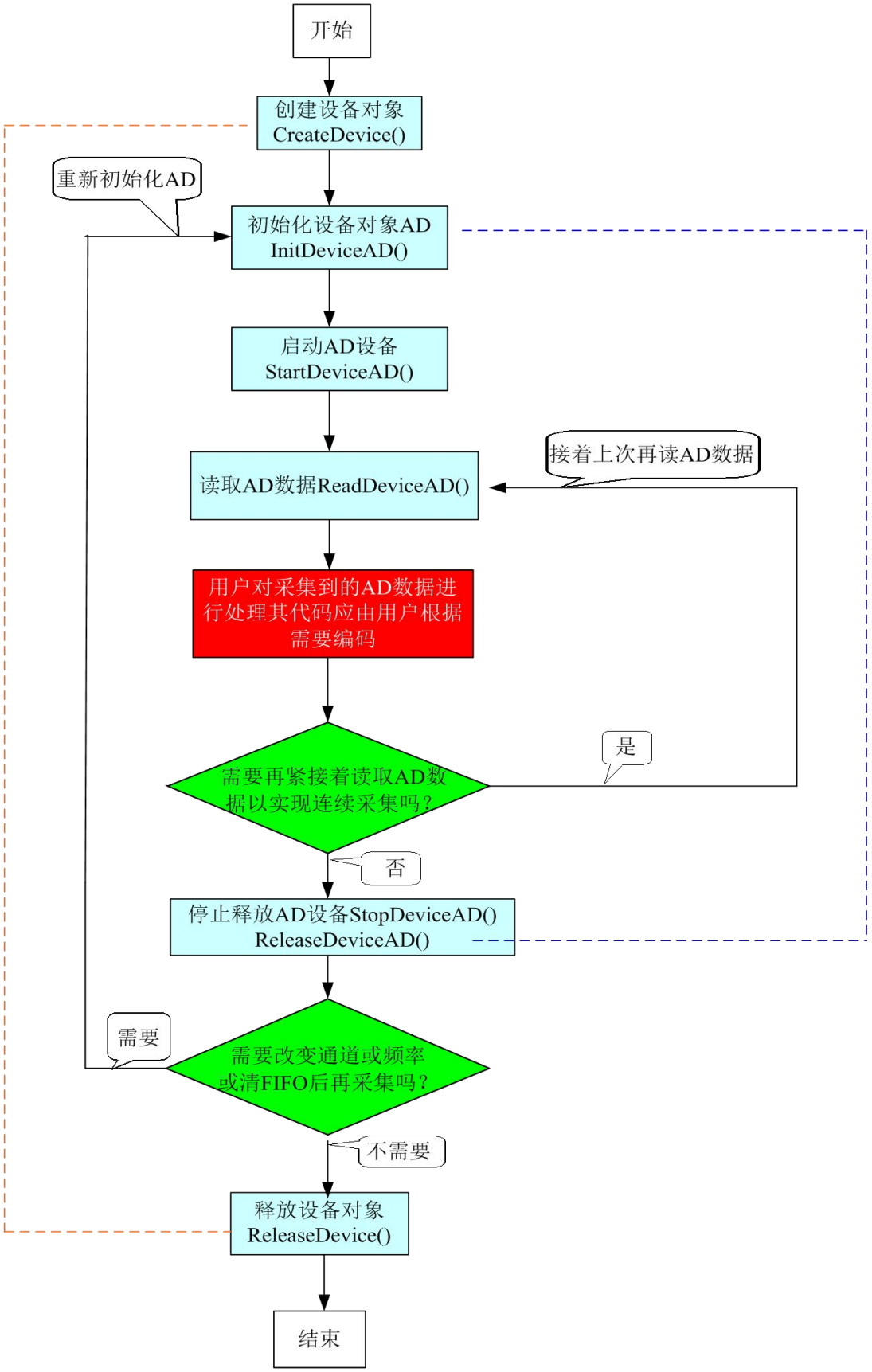


图 2.1.1 DMA 方式 AD 采样过程

2.4 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 GetDeviceBar，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 PCIe 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。

3 ACTS 设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心通道使能、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 PCIe 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 PCIe 的资源配置空间、PNP 即插即用管理，而只须 GetDeviceBar 函数便可以同时取得指定设备的地址。这个时候您便可以用这个地址，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。

3.1 设备驱动接口函数总列表（每个函数省略了前缀“ACTS1001_”）

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	取得同一种设备的总台数	上层及底层用户
SetDevicePhysID	获取设备的物理 ID 号	上层及底层用户
GetDeviceCurrentID	获取设备的 ID 号	上层及底层用户
ListDeviceDlg	列表所有同一种设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放总线设备对象	上层及底层用户
② AD 读取函数		
GetMainInfo	获取板卡信息	上层用户
ADCalibration	设备校准	上层用户
InitDeviceAD	初始化 AD 部件准备传输	上层用户
StartDeviceAD	启动 AD 设备，开始转换	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
GetDeviStatusAD	取得当前设备状态	上层用户
ReadDeviceAD	DMA 方式读取设备上的 AD 数据	上层用户

StopDeviceAD	暂停 AD 设备	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
AllocateBuffer	申请内存	上层用户
FreeBuffer	释放申请的内存	上层用户
④ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读出硬件参数	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，必须在您的源程序中包含如下语句：

#include “C:\Art\ACTS1001\INCLUDE\ACTS1001.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 ACTS1001.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。

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 DeviceD = 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。

相关函数： GetDeviceCount　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例

```
        :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
    hDevice = ACTS1001_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 ACTS1001 设备的总数量**

函数原型：

Visual C++:

int GetDeviceCount (HANDLE hDevice)

功能：取得 ACTS1001 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 ACTS1001 的数量。

相关函数： GetDeviceCount　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦**取得该设备当前相应的 ID 号**

函数原型：

Visual C++:

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

功能：取得指定设备相应 ID 号。

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。

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 号。

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： GetDeviceCount　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 ACTS1001 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 ACTS1001 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 ACTS1001 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。

3.3 AD 程序查询方式采样操作函数原型说明

♦ **设备校准函数**

函数原型：

Visual C++:

BOOL GetMainInfo(HANDLE hDevice
PACTS1001_AD_MAIN_INFO pMainInfo)

功能：获取板卡信息。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pMainInfo 板卡信息，此程序支持 8533B、8534B、8543B、8544B、8531B、8532B 板卡，此结构本主要返回当前板卡的型号、通道数、分辨率等信息。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice

♦ **设备校准函数**

函数原型：

Visual C++:

BOOL ADCalibration (HANDLE hDevice)

功能：设备校准函数。

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,
PACTS1001_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，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 StartDeviceAD。

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　InitDeviceAD　StartDeviceAD
SetDeviceTrigAD　GetDeviStatusAD　ReadDeviceAD
StopDeviceAD　ReleaseDeviceAD　ReleaseDevice

♦ **启动 AD 设备**

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　StartDeviceAD
SetDeviceTrigAD　GetDeviStatusAD　ReadDeviceAD
StopDeviceAD　ReleaseDeviceAD　ReleaseDevice

♦ **产生软件触发事件**

函数原型：

Visual C++:

BOOL SetDeviceTrigAD(HANDLE hDevice);

功能：当前设备使能允许后，产生软件触发事件（只有触发源位软件触发时有效）。

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
SetDeviceTrigAD　GetDeviStatusAD　ReadDeviceAD
StopDeviceAD　ReleaseDeviceAD　ReleaseDevice

- **取得 AD 状态标志**

函数原型：

Visual C++:

```
BOOL GetDevStatusAD( HANDLE hDevice,
                     ACTS1001_STATUS_AD pADStatus);
```

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

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（ACTS1001_STATUS_AD）》章节。

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。

相关函数: CreateDevice　InitDeviceAD　StartDeviceAD
SetDeviceTrigAD　GetDeviStatusAD　ReadDeviceAD
StopDeviceAD　ReleaseDeviceAD　ReleaseDevice

- **DMA 方式读取设备上的 AD 数据**

函数原型：

Visual C++:

```
BOOL ReadDeviceAD( HANDLE hDevice,
                   PWORD pADBuffer,
                   ULONG nReadSizeWords,
                   PULONG nRetSizeWords,
                   PULONG pbFlag,
                   double fTimeout);
```

功能：DMA 方式读取 AD 数据。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pADBuffer　接受 AD 数据的用户缓冲区，如果缓冲区首地址不是 4096 整数倍，请将缓冲区多分配 2048 点，也可用 AllocateBuffer 分配首地址是 4096 整数倍的缓冲区。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。必须为 2048 整数倍。

nRetSizeWords 返回实际读取点数，=NULL,表示无须返回。

pbFlag 此参数有两个意义：最高位为 1 表示溢出，低 16 位为有效数据相对于首地址的偏移位置，如果 pADBuffer 首地址为 4096 整数位则低 16 位返回 0。

fTimeout 超时时间单位:秒(S)

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 SetDeviceTrigAD GetDeviStatusAD ReadDeviceAD StopDeviceAD ReleaseDeviceAD ReleaseDevice

♦ **暂停 AD 设备**

函数原型：

Visual C++:

BOOL StopDeviceAD (HANDLE hDevice)

功能：暂停 AD 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 ReadDeviceAD StopDeviceAD ReleaseDeviceAD ReleaseDevice

♦ **释放设备上的 AD 部件**

函数原型：

Visual C++:

BOOL ReleaseDeviceAD (HANDLE hDevice)

功能：释放设备上的 AD 部件。

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 ReleaseDeviceAD ReleaseDevice

♦ **申请内存**

函数原型：

Visual C++:

BOOL AllocateBuffer (HANDLE hDevice)

功能：申请内存。

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。

注意:此函数申请首地址能被 4096 整除的内存,内存做为 ReadDeviceAD 存放码值的缓冲。

♦ **释放内存**

函数原型：

Visual C++:

BOOL FreeBuffer (HANDLE hDevice)

功能：释放申请的内存。

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。

♦ **AD 读取函数一般调用顺序**

① CreateDevice

② InitDeviceAD

③ StartDeviceAD

④ ReadDeviceAD

⑤ StopDeviceAD

⑥ ReleaseDeviceAD

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。

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

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++:

BOOL SaveParaAD (HANDLE hDevice,
PACTS1001_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 ACTS1001_PARA_AD 的详细介绍请参考 ACTS1001.h 或 ACTS1001.Bas 或 ACTS1001.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

♦ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaAD (HANDLE hDevice,
                 PACTS1001_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 ACTS1001_PARA_AD 的结构指针类型，它负责返回 PCIe 硬件参数值，关于结构指针类型 ACTS1001_PARA_AD 请参考 ACTS1001.h 或 ACTS1001.Bas 或 ACTS1001.Pas 函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

Viusal C++:

```
BOOL ResetParaAD (HANDLE hDevice,
                  PACTS1001_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 ACTS1001_PARA_AD 的详细介绍请参考 ACTS1001.h 或 ACTS1001.Bas 或 ACTS1001.Pas 函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ResetParaAD　ReleaseDevice

4 硬件参数结构

4.1 AD 硬件参数介绍（ACTS1001_PARA_AD）

Visual C++:

```
typedef struct _ACTS1001_PARA_AD
{
    LONG bChannelArray[4];      // 采样通道选择阵列,分别控制个通道,=TRUE 表示该通道采样,否则不采样(只支持种通道配置:0 01 0123)
    LONG InputRange[4];         // 模拟量输入量程选择
    LONG CouplingType[4]; //  耦合类型(直流耦合, 交流耦合)
    LONG FreqDivision;          // 分频数[1, 2147483647],外时钟:采样频率=外部时钟频率/分频数;其它时钟:采样频率=基准频率/分频数;
    LONG DelaySamps;              // M 段长度(字),延时触发下延时点数范围[0, 2147483647]
    LONG TriggerMode;           // 触发模式选择
    LONG TriggerSource;         // 触发源选择
    LONG TriggerType;       // 触发类型
    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 TrigWindow;        // 触发灵敏窗单位 nS,步进为本卡最高采样率的采样周期;例如最高 M,步进即为.5nS
    LONG ReferenceClock;  // 参考时钟选择
    LONG TimeBaseClock;         // 采样时基选择
    LONG bMasterEn;             // 主设备使能
                                // =0：从设备,接收主设备发送的同步触发信号
                                // =1：主设备,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
    LONG SyncTrigSignal;  // 同步触发源
    LONG bClkOutEn;             // 时钟是否输出,TRUE 输出,(仅板卡 b 8532b 支持)
    LONG ClkOutSel;             // 时钟输出选择,(仅板卡 b 8532b 支持)
    LONG bTrigOutEn;        // 触发是否输出,TRUE 输出,(仅板卡 b 8532b 支持)
    LONG TrigOutPolarity;  // 触发输出极性,(仅板卡 b 8532b 支持)
    LONG TrigOutWidth;          // 触发脉冲输出宽度单位 nS,[50, 50000]步进 nS
} ACTS1001_PARA_AD, *PACTS1001_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。

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。

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义

ACTS1001_INPUT_N5000_P5000mV	0x00	±5000mV
ACTS1001_INPUT_N1000_P1000mV	0x01	±1000mV

CouplingType 耦合类型所使用的选项。

常量名	常量值	功能定义
ACTS1001_COUPLING_DC	0x00	直流耦合
ACTS1001_COUPLING_DC	0x01	交流耦合

FreqDivision 分频数[1,2147183647]，外时钟：采样频率=外部时钟频率/分频数；其他时钟：采样频率=基准频率频率/分频数；

TriggerMode AD 触发模式。

常量名	常量值	功能定义
ACTS1001_TRIGMODE_POST	0x00	后触发
ACTS1001_TRIGMODE_DELAY	0x01	硬件延时触发

TriggerType AD 触发类型。

常量名	常量值	功能定义
ACTS1001_TRIGTYPE_EDGE	0x00	边沿触发

TriggerSource 触发源信号所使用的选项。

常量名	常量值	功能定义
ACTS1001_TRIGMODE_SOFT	0x00	软件触发
ACTS1001_TRIGSRC_DTR	0x01	外部数字触发(DTR)
ACTS1001_TRIGSRC_TRIGGER	0x02	同步信号触发(用于多卡同步,数字信号,直接捕捉边沿)
ACTS1001_TRIGSRC_CH0	0x03	通道 0 触发
ACTS1001_TRIGSRC_CH1	0x04	通道 1 触发
ACTS1001_TRIGSRC_CH2	0x05	通道 2 触发
ACTS1001_TRIGSRC_CH3	0x06	通道 3 触发

TrigLevelVolt 触发电平模拟输入量程。

TrigWindow 触发灵敏窗时间值，单位为 25 纳秒。

TriggerDir 触发方向所使用触发模式。它的选项值如下表：

常量名	常量值	功能定义
ACTS1001_TRIGDIR_NEGATIVEL	0x00	下降沿触发
ACTS1001_TRIGDIR_POSITIVE	0x01	上升沿触发
ACTS1001_TRIGDIR_NEGAT_POSIT	0x02	上下边沿均触发

ReferenceClock 参考时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
ACTS1001_RECLK_ONBOARD	0x00	板载时钟(默认)
ACTS1001_RECLK_EXT_10M	0x01	外部 M（CLK_IN
ACTS1001_RECLK_PXI_10M	0x02	PXI_CLK10M

TimeBaseClock　采样时基选择。它的选项值如下表：

常量名	常量值	功能定义
ACTS1001_TBCLK_IN	0x00	内部时钟(默认)
ACTS1001_RECLK_EXT	0x01	外部时钟（CLK_IN）

SyncTrigSignal 同步触发源所使用的选项

常量名	常量值	功能定义
ACTS1001_STS_TRIGGER0	0x00	选择同步 TRIG0 信号触发
ACTS1001_STS_TRIGGER1	0x01	选择同步 TRIG1 信号触发
ACTS1001_STS_TRIGGER2	0x02	选择同步 TRIG2 信号触发
ACTS1001_STS_TRIGGER3	0x03	选择同步 TRIG3 信号触发
ACTS1001_STS_TRIGGER4	0x04	选择同步 TRIG4 信号触发
ACTS1001_STS_TRIGGER5	0x05	选择同步 TRIG5 信号触发
ACTS1001_STS_TRIGGER6	0x06	选择同步 TRIG6 信号触发
ACTS1001_STS_TRIGGER7	0x07	选择同步 TRIG7 信号触发

ClkOutSel 时钟输出选择所使用的选项

常量名	常量值	功能定义
ACTS1001_CLKOUT_REFERENCE	0x00	输出参考时钟
ACTS1001_CLKOUT_TIMEBASE	0x01	输出基准时钟

TrigOutPolarity 时钟输出选择所使用的选项

常量名	常量值	功能定义
ACTS1001_TOP_NEGATIVE	0x00	负脉冲输出
ACTS1001_TOP_POSITIVE	0x01	正脉冲输出

TrigOutWidth 触发脉冲输出宽度单位 nS,[50, 50000]步进 nS

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ReleaseDevice

4.2 AD 主要信息结构体(ACTS1001_AD_MAIN_INFO)

Visual C++:

```
typedef struct _ACTS1001_AD_MAIN_INFO
{
  LONG nDeviceType;          // 总线类型\设备类型 x20128514 20118512...2012(PXIE)2011(PCIE)
```

```
    LONG nChannelCount;         // AD 通道数量
    LONG nDepthOfMemory;        // AD 板载存储器深度(MB)
    LONG nSampResolution;// AD 采样分辨率(如=8 表示 Bit; =12 表示 Bit; =14 表示 Bit; =16 表示 Bit)
    LONG nSampCodeCount;        // AD 采样编码数量(如, 4096, 16384, 65536)
    LONG nTrigLvlResolution;    //  触发电平分辨率(如=8 表示 Bit; =12 表示 Bit; =16 表示 Bit)
    LONG nTrigLvlCodeCount;     //  触发电平编码数量(如, 4096)
    LONG nBaseRate;             //  基准频率 Hz
    LONG nMaxRate;              //  最大频率 Hz
    LONG nMinFreqDivision;      //  最小分频数
    LONG nSupportImped;         //  是否支持输入阻抗(0:不支持 1:支持)
    LONG nSupportPFI;           //  是否支持 PFI 触发输入输出选择(0:不支持 1:支持)
    LONG nSupportExt10M;        //  是否支持外部 M(0:不支持 1:支持)
    LONG nSupportExtClk;        //  是否支持外时钟(0:不支持 1:支持)
    LONG nSupportPXIE100M;      //  是否支持 PXIE100M(0:不支持 1:支持)
    LONG nSupportClkOut;        //  是否支持时钟输出(0:不支持 1:支持)
    LONG nSupportTrigOut;       //  是否支持触发输出(0:不支持 1:支持)
    LONG nReserved0;            //  保留字段(暂未定义)
    LONG nReserved1;            //  保留字段(暂未定义)
    LONG nReserved2;            //  保留字段(暂未定义)
} ACTS1001_AD_MAIN_INFO, *PACTS1001_AD_MAIN_INFO;
```

此结构体主要返回板卡信息，各个板卡返回的信息不同。

相关函数： CreateDevice　　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。

以 14 位 AD 板卡为例

量程	计算机语言换算公式（ANSIC语法）	Volt取值范围（mV）
±5000mV	Volt=(10000.00/16384)*(ADBuffer[0]&0x3FFF)-5000	[-5000,+4999.39]
±1000mV	Volt=(2000.00/16384)*(ADBuffer[0]&0x3FFF)-1000	[-1000,+999.88]

以 12 位 AD 板卡为例

量程	计算机语言换算公式（ANSIC语法）	Volt取值范围（mV）
±5000mV	Volt=(10000.00/4096)*(ADBuffer[0]&0xFFF)-5000	[-5000,+4997.56]
±1000mV	Volt=(2000.00/4096)*(ADBuffer[0]&0xFFF)-1000	[-1000,+999.51]

下面举例说明各种语言的换算过程（以±10V 量程为例）

Visual C++:

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

Visual Bacic:

```
Lsb= ADBuffer[0] And H3FFF;
Volt=(20000.00/16384)*Lsb-1000.00;
```

LabVIEW:

请参考相关演示程序。

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	…

6 上层用户函数接口应用实例

6.1 简易程序演示说明

怎样使用 ReadDeviceAD 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**[ACTS1001 AD (V6.00.02)]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[非空方式]**

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 ACTS1001.h 和 ADDoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[ACTS1001 AD (V6.00.02)]**丨**[Microsoft Visual C++]**丨[高级代码演示]

其默认存放路径为：系统盘\ART\ACTS1001\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（每个函数省略了前缀“ACTS1001_”）

函数名	函数功能	备注
① PCIe 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的设备寄存器组 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 附加操作函数		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

7.2 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。

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

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数;：CreateDevice　　GetDeviceBar　　WriteRegisterByte

WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice
GetDevVersion

♦ **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 GetDeviceBar WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice
GetDevVersion

♦ **以单字节（即 8 位）方式写 PCIe 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)
```

功能：以单字节（即 8 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr PCIe 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 GetDeviceBar WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice
GetDevVersion

Visual C++程序举例:

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

♦ **以双字节（即 16 位）方式写 PCIe 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)
```

功能：以双字节（即 16 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

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

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

Value 输出 16 位整型值。

返回值：无。

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

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 PCIe 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)
```

功能：以四字节（即 32 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

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

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

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

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 PCIe 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)
```

功能：以单字节（即 8 位）方式读 PCIe 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　PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

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

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

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

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 PCIe 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 PCIe 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)
```

功能：以双字节（即 16 位）方式读 PCIe 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　PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

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

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

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

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 PCIe 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 PCIe 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)
```

功能：以四字节（即 32 位）方式读 PCIe 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　PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

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

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

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

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 PCIe 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,
                    __int64 pPort,
                    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。

pPort 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,
                    __int64 pPort,
                    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。

pPort 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,
                     __int64 pPort,
                     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。

pPort 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,
                   __int64 pPort)
```

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。

pPort 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,
                   __int64 pPort)
```

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。

pPort 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,
                     __int64 pPort)
```

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。

pPort 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：

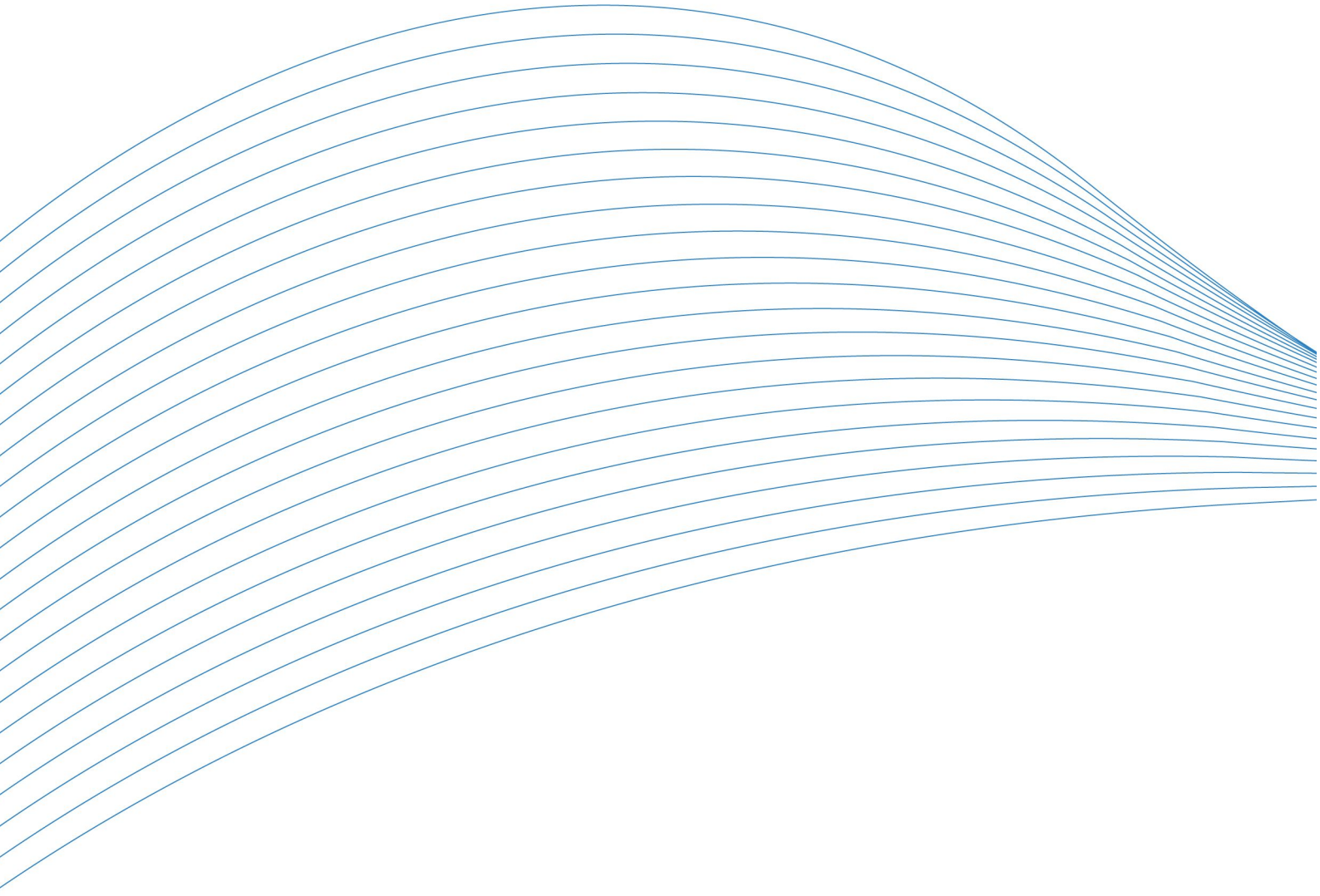
Visual C++:

```
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。



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