

PXI9053 数据采集卡

驱动程序使用手册

北京阿尔泰科技发展有限公司

V6..02.00



■ 关于本手册

本手册为阿尔泰科技推出的 PXI9053 数据采集卡的驱动程序使用手册，其中包括版权信息与命名约定、使用纲要、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：V6.02.00

目 录

■ 关于本手册.....	1
■ 1 版权信息与命名约定.....	3
1.1 版权信息.....	3
1.2 命名约定.....	3
■ 2 使用纲要.....	4
2.1 使用上层用户函数，高效、简单.....	4
2.2 如何管理设备.....	4
2.3 哪些函数对您不是必须的.....	4
■ 3 PXI 设备操作函数接口介绍.....	5
3.1 设备驱动接口函数总列表.....	5
3.2 设备对象管理函数原型说明.....	6
3.3 AD 数据读取函数原型说明.....	8
3.4 AD 程序查询方式函数原型说明.....	8
3.5 AD 数据读取函数(DMA 方式)原型说明.....	11
3.6 AD 硬件参数操作函数原型说明.....	13
3.7 DA 数据输出函数原型说明.....	14
3.8 DA 校准函数原型说明.....	16
3.9 数字 I/O 输入输出函数原型说明.....	18
■ 4 硬件参数结构介绍.....	19
■ 5 上层用户函数接口应用实例.....	23
5.1 简易程序演示说明.....	23
5.2 高级程序演示说明.....	23
■ 6 共用函数介绍.....	24
6.1 公用接口函数总列表.....	24
6.2 PXI 内存映射寄存器操作函数原型说明.....	24
6.3 I/O 端口读写函数原型说明.....	30
6.4 线程操作函数原型说明.....	32
■ 7 修改历史.....	34

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

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 PXI9053_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 [WriteDeviceDA](#)。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，您基本上可以不必参考硬件说明书，除非您需要知道板上 D 型插座等管脚分配情况。因为上层函数的命名、参数的命名极其规范。

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其他函数，如 [WriteDeviceDA](#) 函数可以用 `hDevice` 句柄实现对 DA 数据的采样读取。最后可以通过 [ReleaseDevice](#) 将 `hDevice` 释放掉。

2.3 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，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么 [GetDeviceAddr](#)，[WriteRegisterByte](#)，[WriteRegisterWord](#)，[WriteRegisterULong](#)，[ReadRegisterByte](#)，[ReadRegisterWord](#)，[ReadRegisterULong](#) 等函数您可完全不必理会，除非您是作为底层用户管理设备。而 [WritePortByte](#)，[WritePortWord](#)，[WritePortULong](#)，[ReadPortByte](#)，[ReadPortWord](#)，[ReadPortULong](#) 则对 PXI 用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在 NT、Win2000 等操作系统中实现对您原有传统设备如 ISA 卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于 Windows NT 架构的操作系统中无法继续使用您原有的老设备。

3 PXI 设备操作函数接口介绍

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“PXI9053_”)

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
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	以对话框窗体方式列表系统当中的所有的该 PXI 设备	上层及底层用户
ReleaseDevice	关闭设备, 且释放 PXI 总线设备对象	上层及底层用户
② AD 数据读取函数		
ADCalibration	AD 自动校准函数	上层用户
③AD 程序查询方式函数		
InitDeviceProAD	初始化设备, 当返回 TRUE 后, 设备即准备就绪.	上层用户
StartDeviceProAD	在初始化之后, 启动设备	上层用户
ReadDeviceProAD_Npt	当 AD 标志有效时, 用此函数读取设备上的 AD 数据(程序非空方式)	上层用户
GetDevStatusProAD	在 AD 采样过程中取得设备的各种状态, 返回值表示函数是否成功	上层用户
ReadDeviceProAD_Half	当 AD 标志有效时, 用此函数读取设备上的 AD 数据(程序半满方式)	上层用户
StopDeviceProAD	在启动设备之后, 暂停设备	上层用户
ReleaseDeviceProAD	关闭 AD 设备, 禁止传输, 且释放资源	上层用户
④ AD 数据读取函数(DMA 方式)		
InitDeviceDmaAD	初始化设备, 当返回 TRUE 后, 设备即准备就绪.	上层用户
StartDeviceDmaAD	在初始化之后, 启动设备	上层用户
GetDevStatusDmaAD	在 AD 采样过程中取得 DMA 的有关状态, 返回值表示函数是否成功	上层用户
SetDevStatusDmaAD	在 AD 采样过程中设置 DMA 的有关状态, 返回值表示函数是否成功	上层用户
StopDeviceDmaAD	在启动设备之后, 暂停设备	上层用户
ReleaseDeviceDmaAD	关闭 AD 设备, 禁止传输, 且释放资源	上层用户
⑤AD 的硬件参数操作函数		
SaveParaAD	将当前的 AD 采样参数保存至系统中	上层用户
LoadParaAD	将 AD 采样参数从系统中读出	上层用户
ResetParaAD	将 AD 采样参数恢复至出厂默认值	上层用户
⑥DA 数据输出函数		
InitDeviceProDA	初始化 DA 对象	上层用户

ClearFIFODA	清除 FIFO 中的数据	上层用户
SetDevFreqDA	动态设置 DA 的输出频率	上层用户
StartDeviceDA	启动 DA 设备开始 DA 转换	上层用户
GetDevStatusDA	在 DA 输出过程中取得 FIFO 的状态，通常用于半满方式读取	上层用户
WriteDeviceDA	向 DA 的 FIFO 中写入批量数据(通常为 FIFO 的半满长度)	上层用户
StopDeviceDA	暂停 DA 设备进行 DA 转换	上层用户
ReleaseDeviceDA	停止 DA 转换，且释放 DA 资源	上层用户
⑦DA 校准函数		
StartCalibration	启动 DA 校准	上层用户
GetDACalibration	设备 DA 校准	上层用户
SetDACalibration	设备 DA 校准	上层用户
StopCalibration	停止 DA 校准	上层用户
⑧数字 I/O 输入输出函数		
SetDeviceDO	输出开关量状态	上层用户
GetDeviceDI	取得开关量状态	上层用户
RetDeviceDO	回读开关量输出状态	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\PXI9053\INCLUDE\PXI9053.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 PXI9053.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。#include "PXI9053.H"

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 DeviceLgcID = 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，您才能实现对设备所有功能的访问。

参数:

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PXI 设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PXI9053 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 PXI9053 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 PXI 设备时，DeviceLgcID 应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由 BIOS 和操作系统加载设备时，依据主板总线编号等信息进行这个设备 ID 号分配，说得简单点，就是加载

- ◆ 以对话框窗体方式列表系统当中的所有的该 PXI 设备

函数原型:

Visual C++:

[BOOL ListDeviceDlg \(HANDLE hDevice\)](#)

功能: 列表系统中 PXI9053 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 PXI9053 设备的配置情况。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

- ◆ 释放设备对象

函数原型:

Visual C++:

[BOOL ReleaseDevice\(HANDLE hDevice\)](#)

功能: 释放设备对象。

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 数据读取函数原型说明

- ◆ AD 自动校准函数

函数原型:

Visual C++:

[BOOL_ADCalibration\(HANDLE hDevice\);](#)

功能: AD 自动校准函数

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

相关函数: [CreateDevice](#) [ReleaseDevice](#)

3.4 AD 程序查询方式函数原型说明

- ◆ 初始化设备, 当返回 TRUE 后, 设备即准备就绪

函数原型:

Visual C++:

[BOOL_InitDeviceProAD\(HANDLE hDevice](#)


```
BOOL_GetDevStatusProAD(HANDLE hDevice
                        PPXI9053_STATUS_AD pADStatus);
```

功能: 初始化设备, 当返回 TRUE 后, 设备即准备就绪。

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

PSTATUS_AD pADStatus AD 的各种信息结构体。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceProAD](#) [StartDeviceProAD](#) [ReadDeviceProAD_Npt](#)
[ReadDeviceProAD_Half](#) [StopDeviceProAD](#) [ReleaseDeviceProAD](#)
[ReleaseDevice](#)

◆当 AD 标志有效时, 用此函数读取设备上的 AD 数据(程序半满方式)

函数原型:

Visual C++:

```
BOOL_ReadDeviceProAD_Half(HANDLE hDevice
                           ULONG ADBuffer[]
                           LONG nReadSizeWords
                           PLONG nRetSizeWords);
```

功能: 当 AD 标志有效时, 用此函数读取设备上的 AD 数据(程序半满方式)

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

ADBuffer[] 接受原始 AD 数据的用户缓冲区。

nReadSizeWords 相对于偏位点后读入的数据长度(字)。

nRetSizeWords 返回实际读取的长度(字)。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceProAD](#) [StartDeviceProAD](#) [ReadDeviceProAD_Npt](#)
[GetDevStatusProAD](#) [StopDeviceProAD](#) [ReleaseDeviceProAD](#)
[ReleaseDevice](#)

◆在启动设备之后, 暂停设备

函数原型:

Visual C++:

```
BOOL_StopDeviceProAD(HANDLE hDevice);
```

功能: 在启动设备之后, 暂停设备。

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

相关函数: [CreateDevice](#) [InitDeviceProAD](#) [StartDeviceProAD](#) [ReadDeviceProAD_Npt](#)
[GetDevStatusProAD](#) [ReadDeviceProAD_Half](#) [ReleaseDeviceProAD](#)
[ReleaseDevice](#)

◆关闭 AD 设备, 禁止传输, 且释放资源

函数原型:

Visual C++:

`BOOL_ReleaseDeviceProAD(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: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceProAD](#) [StartDeviceProAD](#) [ReadDeviceProAD_Npt](#)
[GetDevStatusProAD](#) [ReadDeviceProAD_Half](#) [StopDeviceProAD](#) [ReleaseDevice](#)

3.5 AD 数据读取函数 (DMA 方式) 原型说明

◆初始化设备, 当返回 TRUE 后,设备即准备就绪

函数原型:

Visual C++:

```
BOOL_InitDeviceDmaAD(HANDLE hDevice
                    HANDLE hDmaEvent,
                    ULONG ADBuffer[],
                    LONG nReadSizeWords,
                    LONG nSegmentCount,
                    LONG nSegmentSizeWords,
                    PPXI9053_PARA_AD pADPara);
```

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。

`hDmaEvent` DMA 事件句柄,它由 [CreateSystemEvent](#) 创建

`ADBuffer[]` 用户缓冲区,最好为两维数组

`nReadSizeWords` 每次 DMA 时,用户从指定缓冲应读取的实际长度(要小于或等于 `nSegmentSizeWords`)

`nSegmentCount` 缓冲分段的数量,取值范围为 2-128

`nSegmentSizeWords` 缓冲区分段的段长(必须等于 FIFO 半满长度)

`PPARA_AD pADPara`) 硬件参数, 它仅在此函数中决定硬件状态

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [StartDeviceDmaAD](#) [GetDevStatusDmaAD](#)
[SetDevStatusDmaAD](#) [StopDeviceDmaAD](#) [ReleaseDeviceDmaAD](#) [ReleaseDevice](#)

◆在初始化之后, 启动设备

函数原型:

Visual C++:

```
BOOL_StartDeviceDmaAD(HANDLE hDevice);
```

功能: 在初始化之后, 启动设备

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

相关函数: [CreateDevice](#) [InitDeviceDmaAD](#) [GetDevStatusDmaAD](#)
[SetDevStatusDmaAD](#) [StopDeviceDmaAD](#) [ReleaseDeviceDmaAD](#) [ReleaseDevice](#)

◆在 AD 采样示函数过程中取得 DMA 的有关状态,返回值表示是否成功

函数原型:

Visual C++:

```
BOOL_GetDevStatusDmaAD(HANDLE hDevice
                        PPXI9053_STATUS_DMA pDMAStatus);
```

功能: 在 AD 采样示函数过程中取得 DMA 的有关状态,返回值表示是否成功

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

PSTATUS_DMA pDMAStatus 获得的 DMA 工作状态

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceDmaAD](#) [StartDeviceDmaAD](#)
[SetDevStatusDmaAD](#) [StopDeviceDmaAD](#) [ReleaseDeviceDmaAD](#) [ReleaseDevice](#)

◆在 AD 采样过程中设置 DMA 的有关状态,返回值表示函数是否成功

函数原型:

Visual C++:

```
BOOL_SetDevStatusDmaAD(HANDLE hDevice
                        LONG iClrBufferID);
```

功能: 在 AD 采样过程中设置 DMA 的有关状态,返回值表示函数是否成功

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

iClrBufferID 要清除的缓冲区 ID,将其置为 0。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceDmaAD](#) [StartDeviceDmaAD](#) [GetDevStatusDmaAD](#)
[StopDeviceDmaAD](#) [ReleaseDeviceDmaAD](#) [ReleaseDevice](#)

◆在启动设备之后, 暂停设备

函数原型:

Visual C++:

```
BOOL_StopDeviceDmaAD(HANDLE hDevice);
```

功能: 在启动设备之后, 暂停设备

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

相关函数: [CreateDevice](#) [InitDeviceDmaAD](#) [StartDeviceDmaAD](#) [GetDevStatusDmaAD](#)
[SetDevStatusDmaAD](#) [ReleaseDeviceDmaAD](#) [ReleaseDevice](#)

◆关闭 AD 设备,禁止传输,且释放资源

函数原型:

Visual C++:

```
BOOL_ReleaseDeviceDmaAD(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: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceDmaAD](#) [StartDeviceDmaAD](#) [GetDevStatusDmaAD](#)
[SetDevStatusDmaAD](#) [StopDeviceDmaAD](#) [ReleaseDevice](#)

3.6 AD 硬件参数操作函数原型说明

◆将当前的 AD 采样参数保存至系统中

函数原型:

Visual C++:

```
BOOL_SaveParaAD(HANDLE hDevice
                PPARA_AD pADPara);
```

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。

pADPara 硬件参数, 它仅在此函数中决定硬件状态。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [LoadParaAD](#) [ResetParaAD](#) [ReleaseDevice](#)

◆将 AD 采样参数从系统中读出

函数原型:

Visual C++:

```
BOOL_LoadParaAD(HANDLE hDevice
                PPARA_AD pADPara);
```

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。

pADPara 硬件参数, 它仅在此函数中决定硬件状态。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [SaveParaAD](#) [ResetParaAD](#) [ReleaseDevice](#)

◆将 AD 采样参数恢复至出厂默认值

函数原型:

Visual C++:

```
BOOL_ResetParaAD(HANDLE hDevice
                PPARA_AD pADPara);
```

功能: 将 AD 采样参数恢复至出厂默认值

参数:

hDevice 设备对象句柄, 它应由 [CreateDevice](#) 创建。

pADPara 硬件参数, 它仅在此函数中决定硬件状态。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [SaveParaAD](#) [LoadParaAD](#) [ReleaseDevice](#)

3.7 DA 数据输出函数原型说明

◆初始化 DA 对象

函数原型:

Visual C++:

```
BOOL_InitDeviceProDA(HANDLE hDevice
                    DWORD DAFrequency,
                    LONG OutputRange);
```

功能: 初始化 DA 对象

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

DAFrequency DA 的输出频率 Hz (与 [PXI9053_SetDeviceFrequencyProDA](#) 的函数功能相同)。

OutputRange 输出量程, 具体定义请参考上面的常量定义部分

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [ClearFIFODA](#) [SetDevFreqDA](#) [StartDeviceDA](#)
[GetDevStatusDA](#) [WriteDeviceDA](#) [StopDeviceDA](#) [ReleaseDeviceDA](#) [ReleaseDevice](#)

◆清除 FIFO 中的数据

函数原型:

Visual C++:

```
BOOL_ClearFIFODA(HANDLE hDevice);
```

功能: 初始化 DA 对象

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceProDA](#) [SetDevFreqDA](#) [StartDeviceDA](#)
[GetDevStatusDA](#) [WriteDeviceDA](#) [StopDeviceDA](#) [ReleaseDeviceDA](#) [ReleaseDevice](#)

◆动态设置 DA 的输出频率

函数原型:

Visual C++:

```
BOOL_SetDevFreqDA(HANDLE hDevice
                  DWORD DAFrequency);
```

功能: 动态设置 DA 的输出频率

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

DAFrequency DA 的输出频率 Hz

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [InitDeviceProDA](#) [ClearFIFODA](#) [StartDeviceDA](#)
[GetDevStatusDA](#) [WriteDeviceDA](#) [StopDeviceDA](#) [ReleaseDeviceDA](#) [ReleaseDevice](#)

◆启动 DA 设备开始 DA 转换

函数原型:

Visual C++:

pCalData 校准值

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [StartCalibration](#) [StopCalibration](#) [ReleaseDevice](#)

◆设备 DA 校准

函数原型:

Visual C++:

```
BOOL_SetDACalibration(HANDLE hDevice
                      LONG OutputRange,
                      LONG CalMode,
                      PLONG pCalData);
```

功能: 启动 DA 校准

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

OutputRange 输出量程, 分别控制两个通道

CalMode 0 为零点校准, 1 为满度校准

pCalData 校准值

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [StartCalibration](#) [GetDACalibration](#) [StopCalibration](#)
[ReleaseDevice](#)

◆停止 DA 校准

函数原型:

Visual C++:

```
BOOL_StopCalibration(HANDLE hDevice);
```

功能: 停止 DA 校准

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [StartCalibration](#) [GetDACalibration](#) [StopCalibration](#)
[ReleaseDevice](#)

3.9 数字 I/O 输入输出函数原型说明

◆输出开关量状态

函数原型:

Visual C++:

```
BOOL_SetDeviceDO(HANDLE hDevice
                 BYTE bDOSets[16]);
```

功能: 输出开关量状态

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

bDOSets[16] 开关状态

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [GetDeviceDI](#) [RetDeviceDO](#) [ReleaseDevice](#)

◆取得开关量状态

函数原型:

Visual C++:

```
BOOL_GetDeviceDI(HANDLE hDevice
                 BYTE bDOSets[16]);
```

功能: 取得开关量状态

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

bDOSets[16] 开关状态

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [SetDeviceDO](#) [RetDeviceDO](#) [ReleaseDevice](#)

◆回读开关量状态

函数原型:

Visual C++:

```
BOOL_RetDeviceDO(HANDLE hDevice
                 BYTE bDOSets[16]);
```

功能: 输出开关量状态

参数:

hDevice 设备对象句柄, 管理 DA 对象的设备对象, 它应由 [CreateDevice](#) 创建。

bDOSets[16] 开关状态

返回值: 如果调用成功, 则返回 TRUE, 否则返回 FALSE

相关函数: [CreateDevice](#) [SetDeviceDO](#) [GetDeviceDI](#) [ReleaseDevice](#)

4 硬件参数结构介绍

4.1 用于 AD 采样的实际硬件参数

```

typedef struct _CHANNEL_ARRAY_AD
{
    ULONG ADChannel;
    ULONG ADGains;
} CHANNEL_ARRAY_AD;

typedef struct _PXI9053_PARA_AD
{
    ULONG ChannelCount;
    CHANNEL_ARRAY_AD ChannelArray[16];
    ULONG ADMode;
    ULONG Frequency;
    ULONG GroupInterval;
    ULONG LoopsOfGroup;
    ULONG InputRange;
    ULONG TriggerMode;
    ULONG TriggerSource;
    ULONG TriggerType;
    ULONG TriggerDir;
    ULONG ClockSource;
    ULONG bClockOutput;
    ULONG ClockSourceDir;
    ULONG OutClockSource;
    ULONG GroundingMode;
}PXI9053_PARA_AD,*PPXI9053_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。

ADChannel; AD 通道号
ADGains; AD 增益
ChannelCount; 通道总数(1-16)
ChannelArray[16]; 采样阵列(包括通道和增益)

ADMode 所使用工作模式

常量名	常量值	功能定义
ADMODE_SEQUENCE	0x00	连续采样
ADMODE_GROUP	0x01	分组采样

InputRange 模拟量输入范围

常量名	常量值	功能定义
INPUT_N10000_P10000mV	0x00	±1000mV
INPUT_N5000_P5000mV	0x01	±5000mV
INPUT_N5000_P2500mV	0x02	±2500mV
INPUT_0_P10000mV	0x03	0~10000mV
INPUT_0_P5000mV	0x04	0~5000mV

Gains 使用的硬件增益

常量名	常量值	功能定义
GAINS_1MULT	0x00	1 倍增益
GAINS_2MULT	0x01	2 倍增益
GAINS_4MULT	0x02	4 倍增益
GAINS_8MULT	0x03	8 倍增益

TriggerMode 成员变量所使用触发模式

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

TriggerSource 触发源信号

常量名	常量值	功能定义
TRIGSRC_ATR	0x00	选择外部 ATR 作为触发源
TRIGSRC_DTR	0x01	选择外部 DTR 作为触发源

TriggerType 触发类型

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

TriggerDir 触发方向

常量名	常量值	功能定义
TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低脉冲或上升/下降沿触发)

ClockSource 时钟源

常量名	常量值	功能定义
CLOCKSRC_IN	0x00	内部时钟
CLOCKSRC_OUT	0x01	外部时钟(CLKIN)

ClockSourceDir 时钟源方向

常量名	常量值	功能定义
CLOCKSRC_DIR_IN	0x00	方向输入
CLOCKSRC_DIR_OUT	0x01	方向输出

OutClockSource 外部时钟源

常量名	常量值	功能定义
OUTCLOCKSRC_TRIGGER0	0x00	选择 PXI 总线上的 TRIG0 输入/输出时钟
OUTCLOCKSRC_TRIGGER1	0x01	选择 PXI 总线上的 TRIG1 输入/输出时钟
OUTCLOCKSRC_TRIGGER2	0x02	选择 PXI 总线上的 TRIG2 输入/输出时钟
OUTCLOCKSRC_TRIGGER3	0x03	选择 PXI 总线上的 TRIG3 输入/输出时钟
OUTCLOCKSRC_TRIGGER4	0x04	选择 PXI 总线上的 TRIG4 输入/输出时钟
OUTCLOCKSRC_TRIGGER5	0x05	选择 PXI 总线上的 TRIG5 输入/输出时钟
OUTCLOCKSRC_TRIGGER6	0x06	选择 PXI 总线上的 TRIG6 输入/输出时钟
OUTCLOCKSRC_EXTERN	0x07	选择外时钟输出线

bClockOutput 成员变量

常量名	常量值	功能定义
CLOCKOUT_DISABLE	0x00	禁止本卡上的自带时钟向外输出
CLOCKOUT_ENABLE	0x01	允许本卡上的自带时钟向外输出

GroundingMode 使用的模拟信号接地方式

常量名	常量值	功能定义
GNDMODE_SE	0x00	单端方式(SE:Single end)
GNDMODE_DI	0x01	双端方式(DI:Differential)

```
typedef struct _PXI9053_STATUS_AD
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bOverflow;
} PXI9053_STATUS_AD, *PPXI9053_STATUS_AD;
```

OutputRange 模拟量输出范围

常量名	常量值	功能定义
OUTPUT_0_P5000mV	0x00	0~5000mV
OUTPUT_0_P10000mV	0x01	0~10000mV

OUTPUT_N5000_P5000mV	0x02	±5000mV
OUTPUT_N10000_P10000mV	0x03	±10000mV

4.2 用于 DA 采样的实际硬件参数

```
typedef struct _PXI9053_STATUS_DA
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bOverflow;
} PXI9053_STATUS_DA, *PPXI9053_STATUS_DA;
const int MAX_SEGMENT_COUNT = 64;
```

```
typedef struct _PXI9053_STATUS_DMA
{
    LONG iCurSegmentID;
    LONG bSegmentSts[MAX_SEGMENT_COUNT];
    LONG bBufferOverflow;
} PXI9053_STATUS_DMA, *PPXI9053_STATUS_DMA;
```

CreateFileObject 用的文件操作方式控制字(可通过或指令实现多种方式并操作)

常量名	常量值	功能定义
modeRead	0x0000	只读文件方式
modeWrite	0x0001	只写文件方式
modeReadWrite	0x0002	既读又写文件方式
modeCreate	0x1000	如果文件不存可以创建该文件，如果存在，则重建此文件，并清 0
typeText	0x4000	以文本方式操作文件

5 上层用户函数接口应用实例

5.1 简易程序演示说明

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] | [阿尔泰高级测控演示系统] | [PXI9053AD DA DIO Card] | [Microsoft VS2005] | [简易代码演示]

5.2 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 PXI9035.h 和 ADDoc.cpp)。

[程序] | [阿尔泰高级测控演示系统] | [PXI9053AD DA DIO Card] | [Microsoft VS2005] | [高级代码演示]

其默认存放路径为:系统盘\ART\PXI9035\SAMPLES\VC\ADVANCED
其他语言的演示可以用上面类似的方法找到。

6 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。

6.1 公用接口函数总列表

表 5-1-1: 公用接口函数总列表（每个函数省略了前缀“PXI9053_”）

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
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 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	

6.2 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,
                   PULONG LinearAddr,
                   PULONG PhysAddr,
                   int RegisterID = 0)
```

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](#) 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
PXI9053_REG_MEM_CPLD	0x0000	0 号寄存器对应板上控制单元所使用的内存模式基地址(使用 LinearAddr)
PXI9053_REG_IO_CPLD	0x0001	1 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr)

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

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
[ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 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...");
}
:

```

◆ **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,
 ULONG 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。

pulPXIBar 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](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)

◆ 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: 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。

pulFmwVersion 固件版本

pulDriverVersion 驱动版本

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
 [WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)

◆ 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,
                       ULONG LinearAddr,
                       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。

LinearAddr PXI 设备内存映射寄存器的线性基地址, 它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数, 它与 **LinearAddr** 两个参数共同确定

[WriteRegisterByte](#) 函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

相关函数: [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
 [WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
 [ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 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 PXI 内存映射寄存器的某个单元

函数原型：

Visual C++ :

```

BOOL WriteRegisterWord(HANDLE hDevice,
                       PCHAR LinearAddr,
                       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。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定

[WriteRegisterWord](#) 函数所访问的映射寄存器的内存单元。

Value 输出 16 位整型值。

返回值：无。

相关函数：

CreateDevice	GetDeviceAddr	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ 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,
                        PCHAR LinearAddr,
                        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。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定

[WriteRegisterULong](#) 函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数:

CreateDevice	GetDeviceAddr	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ 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,
                      PCHAR LinearAddr,
                      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。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定

[ReadRegisterByte](#) 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数： [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
 [WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
 [ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 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\(HANDLE hDevice,
 PUCHAR LinearAddr,
 ULONG OffsetBytes\)](#)

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。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

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

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数： [CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
 [WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
 [ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 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,
                        PCHAR LinearAddr,
                        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。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

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

[WriteRegisterULong](#) 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数：[CreateDevice](#) [GetDeviceAddr](#) [WriteRegisterByte](#)
[WriteRegisterWord](#) [WriteRegisterULong](#) [ReadRegisterByte](#)
[ReadRegisterWord](#) [ReadRegisterULong](#) [ReleaseDevice](#)

Visual C++ 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
:

```

6.3 I/O 端口读写函数原型说明

◆ 以单字节(8Bit)方式写 I/O 端口

函数原型：

Visual C++:

```
BOOL WritePortByte( HANDLE hDevice,
                   PCHAR nPort,
                   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。

nPort 设备的 I/O 端口号。

Value 写入由 **nPort** 指定端口的值。

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

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
[WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以双字(16Bit)方式写 I/O 端口

函数原型:

Visual C++:

[BOOL WritePortWord](#) ([HANDLE hDevice](#),
[PUCHAR nPort](#),
[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。

[nPort](#) 设备的 I/O 端口号。

[Value](#) 写入由 [nPort](#) 指定端口的值。

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

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
[WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以四字节(32Bit)方式写 I/O 端口

函数原型:

Visual C++:

[BOOL WritePortULong](#)([HANDLE hDevice](#),
[PUCHAR nPort](#),
[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。

[nPort](#) 设备的 I/O 端口号。

[Value](#) 写入由 [nPort](#) 指定端口的值。

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

相关函数: [CreateDevice](#) [WritePortByte](#) [WritePortWord](#)
[WritePortULong](#) [ReadPortByte](#) [ReadPortWord](#)

◆ 以单字节(8Bit)方式读 I/O 端口

函数原型:

Visual C++:

[BYTE ReadPortByte](#)([HANDLE hDevice](#),
[PUCHAR nPort](#))

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。

[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,
PUCHAR nPort)`

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。

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,
PUCHAR nPort)`

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。

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](#)

6.4 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)`

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。

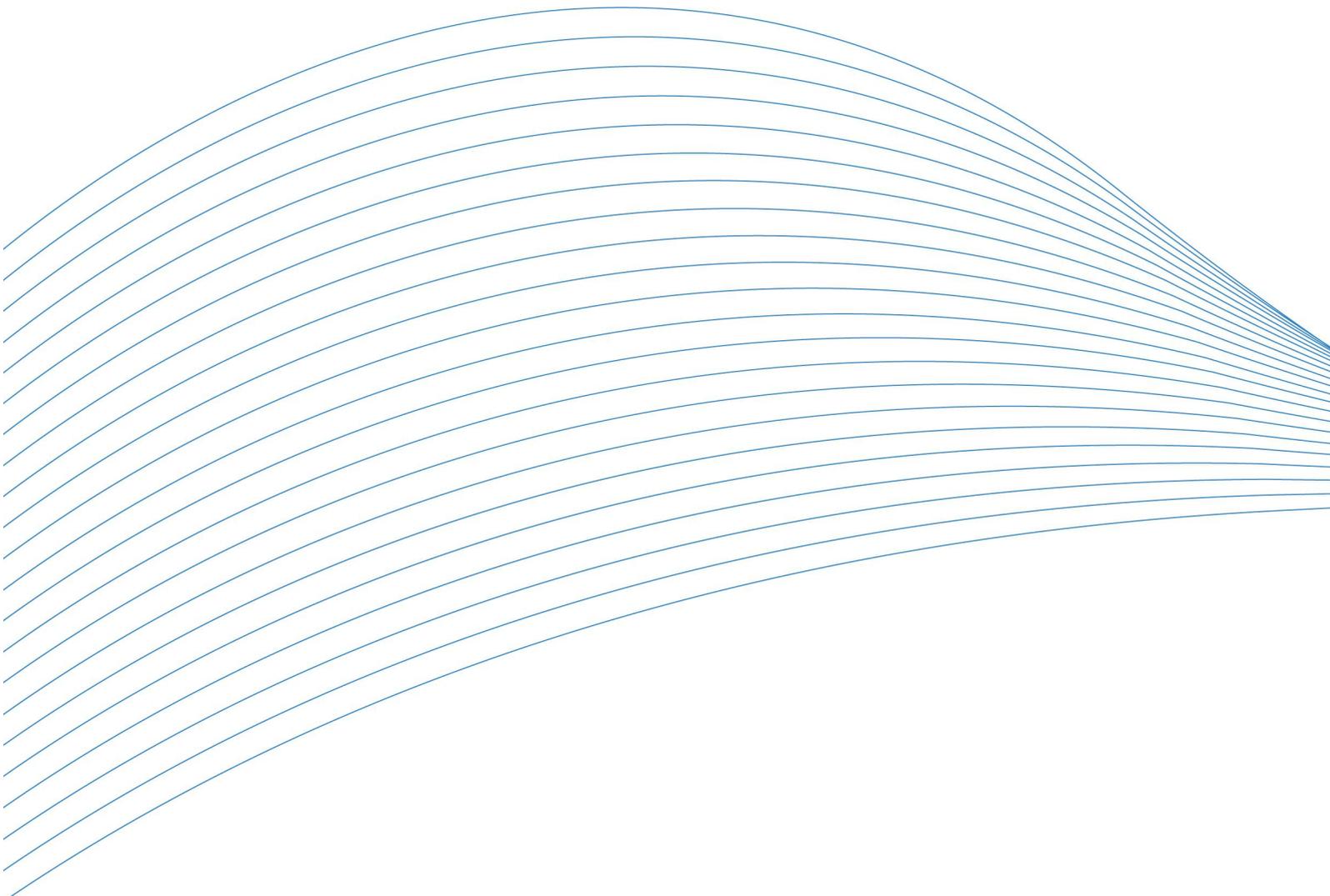
参数:

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。

7 修改历史

修改时间	版本号	修改内容
2017.8.31	V6.00.00	第一版



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