PCI2306 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_ 则被省略。如 PCI2306_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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			

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。诸如InitDeviceProAD、ReadDeviceProAD_NotEmpty、SetDeviceDO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D型插座等管脚分配情况。

第二节、如何管理 PCI 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ReadDevOneAD (或ReadDevBulkAD)函数可以用hDevice句柄实现对AD数据的采样读取，SetDeviceDO函数可用实现开关量的输出等。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何用单点方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用初始化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。然后启动AD部件，开始AD采样，然后便可用ReadDevOneAD反复读取AD数据以实现连续不间断采样。具体执行流程请看下面的图 2.1.1。

第四节、如何用批量方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用初始化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。然后启动AD部件，开始AD采样，然后便可用ReadDevBulkAD反复读取AD数据以实现连续不间断采样。（注：ReadDevBulkAD虽然主要面对批量读取、高速连续采集而设计，但亦可用它以单点或几点的方式读取AD数据，以满足慢速、高实时性采集需要）。具体执行流程请看下面的图 2.1.2。

第五节、如何用中断方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceInt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pPara参数结构体决定的。您只需要对这个pPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hEvent赋给InitDeviceInt的相应参数，它将作为接受AD中断事件的变量。然后启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hEvent中断事件的发生，在中断未到时，自动使所在线程进入睡眠状态（不消耗CPU时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用函数读一批AD数据，然后再接着再等待FIFO的中断事件，若有效再读取，就这样反复读取AD数据即可实现连续不间断采样。当您需要关闭AD设备时，ReleaseDeviceInt便可帮您实现（但设备对象hDevice依然存在）。（注：ReleaseDeviceInt函数在中断事件发生时可以单点或几点的方式读取AD数据，只是到下一次中断事件到来时的时间间隔会变得非常短，由于属于硬件中断，其优先级别高于所有软件，所以您单点或几点读取AD数据时，千万不能让中断间隔太短，否则，有可能使您的整个系统被半满中断事件吞没，就象死机一样，不能动弹。）具体执行流程请看图 2.1.3。

注意：图中较粗的虚线表示对称关系。如红色虚线表示CreateDevice和ReleaseDevice两个函数的关系是：最初执行一次CreateDevice，在结束是就须执行一次ReleaseDevice。

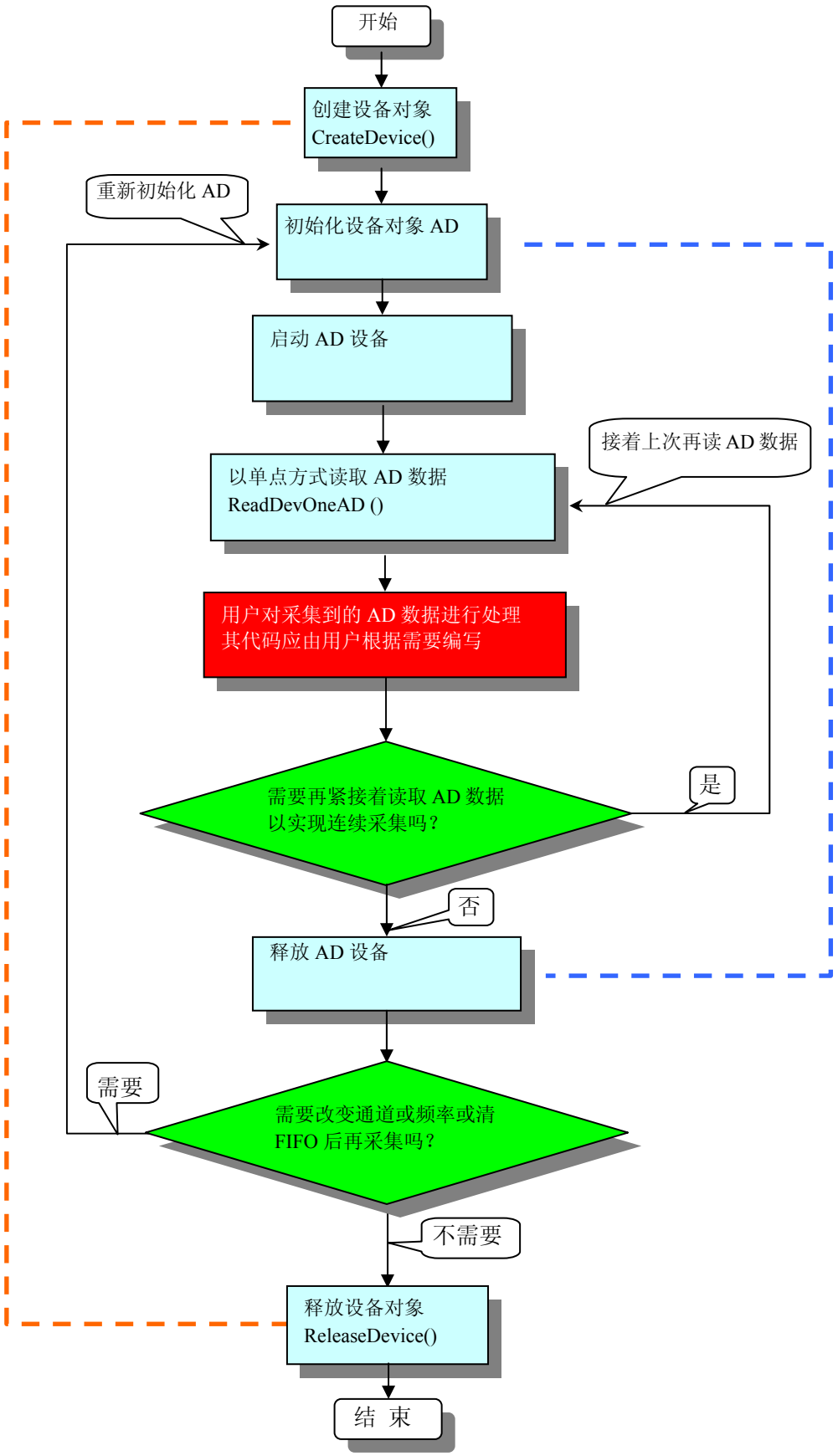


图 2.1.1 单点方式 AD 采集过程

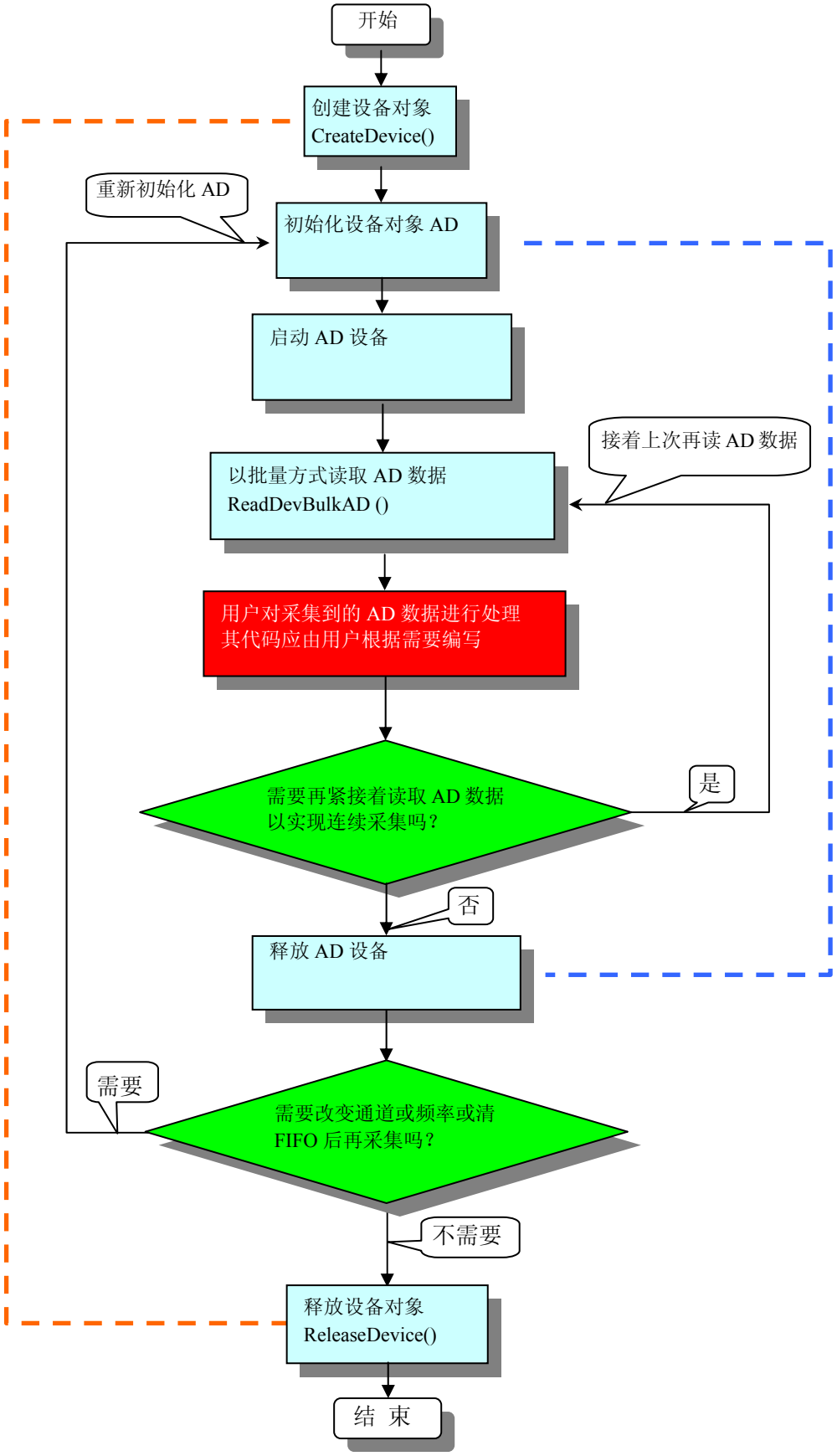


图 2.1.2 批量方式 AD 采集过程

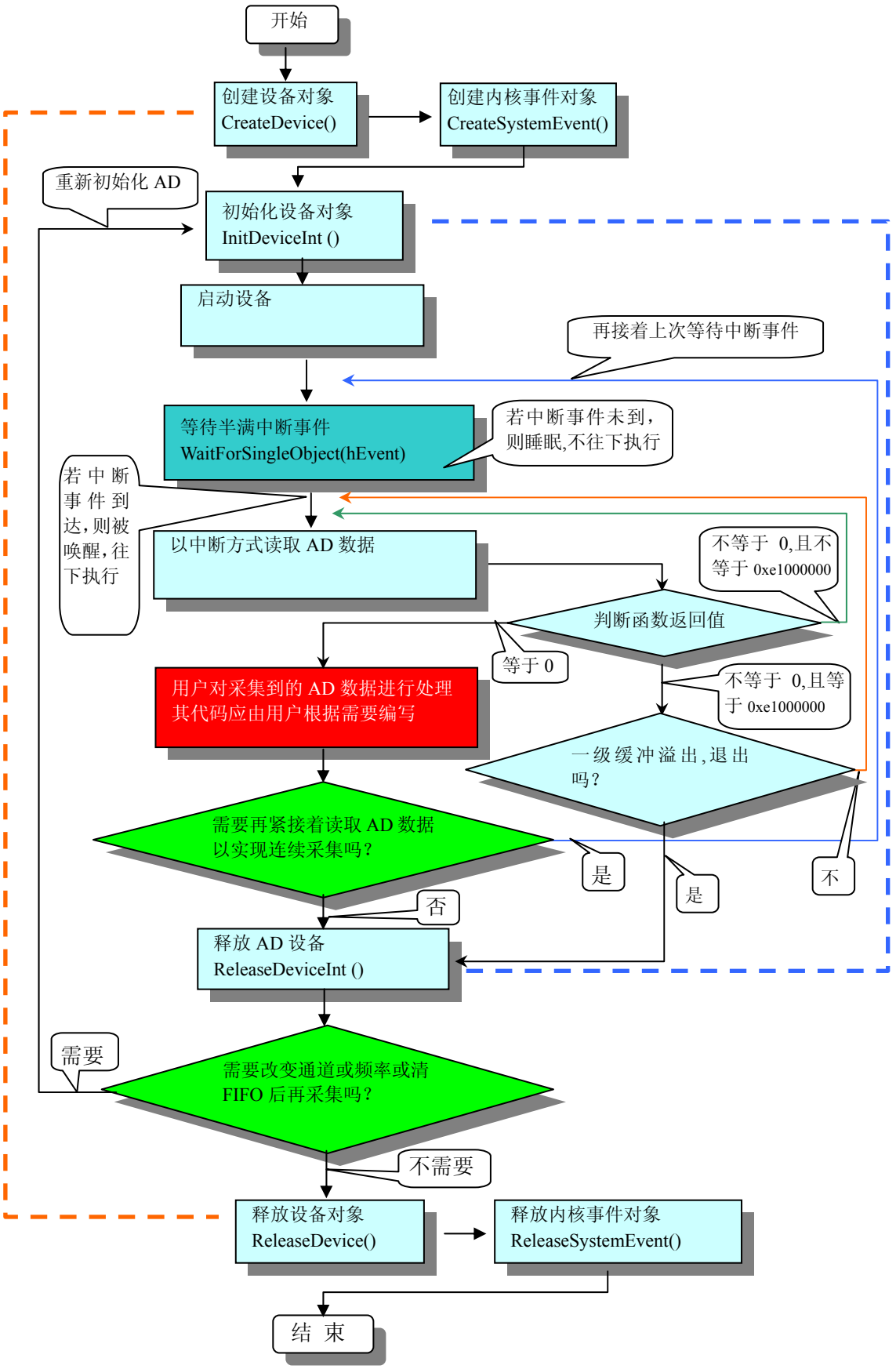


图 2.1.3 中断方式采集实现过程

第六节、如何实现开关量的简便操作

当您有了hDevice设备对象句柄后，便可用SetDeviceDO函数实现开关量的输出操作，其各路开关量的输出状态由其bDOSts[16]中的相应元素决定。由GetDeviceDI函数实现开关量的输入操作，其各路开关量的输入状态由其bDISts[16]中的相应元素决定。

第七节、哪些函数对您不是必须的

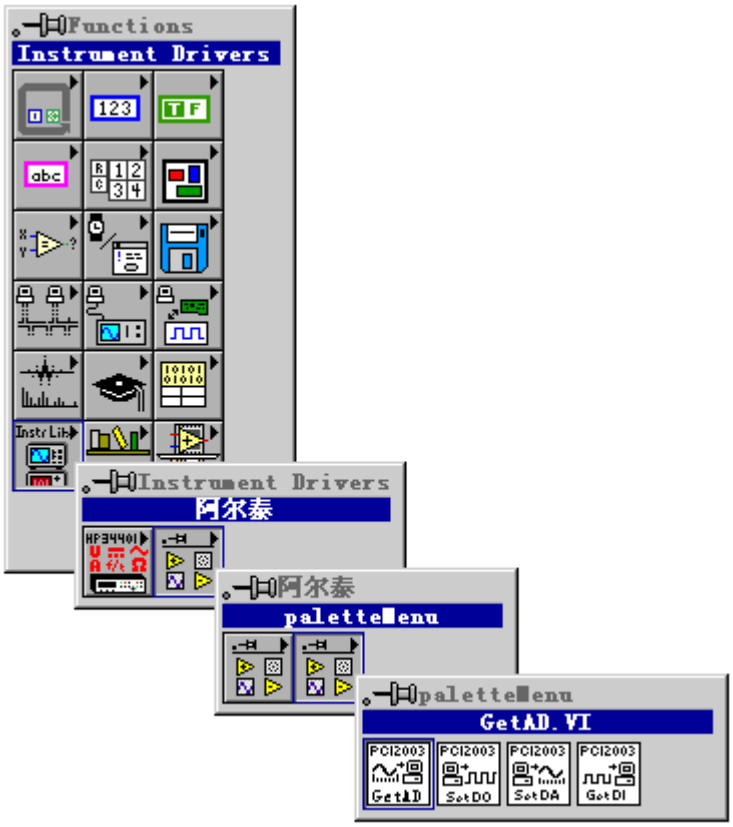
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对PCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、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。

第三章 PCI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（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ReadDevOneAD (或ReadDevBulkAD)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，同是还可以省掉您许多繁琐的工作，比如您不用去了解PCI的资源配置空间、PNP即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后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。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI2306_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
② AD 的程序方式读取函数		
ReadDevOneAD	读取单点 AD 数据	上层用户
ReadDevBulkAD	连续批量读取 PCI 设备上的 AD 数据	上层用户
③ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
④ DA 模拟量输出操作函数		
WriteDeviceProDA	输出 DA 数据	上层用户
⑤CNT 计数定时器操作函数		
InitDevCounter	初始化各路计数器	上层用户
GetDevCounter	取得计数器值	上层用户
⑥ 中断操作函数		
InitDeviceInt	初始化中断	上层用户
GetDeviceIntCount	在中断初始化后，取得中断服务程序产生的次数	上层用户
GetDeviceIntSrc	在中断初始化后，取得中断源	上层用户
ReleaseDeviceInt	释放中断资源	上层用户
⑦ DIO 开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	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\PCI2306\INCLUDE\PCI2306.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 PCI2306.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。然后加入如下语句：

#include “PCI2306.H”

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。

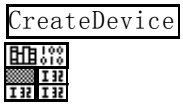
Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话中选择 PCI2306.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：



一、在 LabView 中打开 PCI2306.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

二、根据LabView语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如ReadDevOneAD (或ReadDevBulkAD)接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

◆ **创建设备对象函数（逻辑号）**

函数原型：

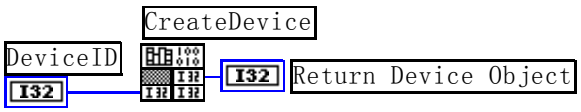
Visual C++：

HANDLE CreateDevice (int DeviceID = 0)

Visual Basic：

Declare Function PCI2306_CreateDevice Lib "PCI2306_32" (ByVal DeviceID As Long) As Long

LabVIEW：



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

实现对该设备所有功能的访问。

参数：

DeviceID 设备 ID 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。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice GetDeviceCount 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
}
    :
```

Visual Basic 程序举例：

```
    :
Dim hDevice As Long   ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub      ' 退出该过程
End If
    :
```

- **取得本计算机系统中 PCI2306 设备的总数量**

函数原型：

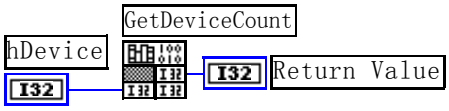
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function PCI2306_GetDeviceCount Lib "PCI2306_32" (ByVal hDevice As Long) As Long

LabVIEW：



功能：取得 PCI2306 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 PCI2306 的数量。

相关函数： CreateDevice GetDeviceCount ListDeviceDlg ReleaseDevice

- **用对话框控件列表计算机系统中所有 PCI2306 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function PCI2306_ListDeviceDlg Lib "PCI2306_32" (ByVal hDevice As Long) As Long

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI2306 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 PCI2306 设备的配置情况。

相关函数： CreateDevice ReleaseDevice

◆ **释放设备对象所占的系统资源及设备对象**

函数原型：

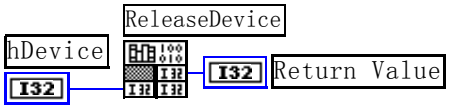
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function PCI2306_ReleaseDevice Lib "PCI2306_32" (ByVal hDevice As Long) As Long

LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

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

相关函数：CreateDevice

应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice后，再一次执行这些函数前，必须执行一次ReleaseDevice函数，以释放由CreateDevice占用的系统软硬件资源，如DMA控制器、系统内存等。只有这样，当您再次调用CreateDevice函数时，那些软硬件资源才可被再次使用。

第三节、AD 程序查询方式采样操作函数原型说明

◆ **读取 PCI 设备上的 AD 数据**

函数原型：

Visual C++:

```
USHORT ReadDevOneAD ( HANDLE hDevice,
                      int nADChannel)
```

Visual Basic:

```
Declare Function PCI2306_ReadDevOneAD Lib "PCI2306_32" (ByVal hDevice As Long,_
                                                        ByVal nADChannel As Long) As Long
```

LabVIEW：

请参考相关演示程序。

功能：从设备上读取一个点的 AD 数据。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

nADChannel AD 通道号，取值为[0~15]。

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nReadSizeWords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。

相关函数： CreateDevice ReadDevOneAD ReadDevBulkAD ReleaseDevice

◆ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型：

Visual C++:

```
BOOL ReadDevBulkAD ( HANDLE hDevice,
                     USHORT ADBuffer,
                     ULONG nReadSizeWords,
                     PPCI2306_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function PCI2306_ReadDevBulkAD Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pADBuffer As Integer,_
                                        ByVal ReadSizeWords As Long, _
```

ByRef pADPara As PCI2306_PARA_AD) As Long

LabVIEW:

请参考相关演示程序。

功能：从设备上批量读取 AD 数据(也可单点)。

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，通常可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区 ADBuffer 的最大空间。

pADPara 设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如AD采样通道、采样频率等。关于PCI2306_PARA_AD具体定义请参考PCI2306.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《AD硬件参数结构》章节。

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

返回值：如果成功的读取由nReadSizeWords参数指定量的AD数据到用户缓冲区，则返回TRUE，否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice ReadDevOneAD ReadDevBulkAD ReleaseDevice

◆ **程序查询方式采样函数一般调用顺序**

① CreateDevice

② ReadDevOneAD或ReadDevBulkAD

③ ReleaseDevice

注明：用户可以反复执行第②步，以实现高速连续不间断大容量采集。

关于这个过程的图形说明请参考《使用纲要》。

第四节、AD 硬件参数保存与读取函数原型说明

◆ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaAD(HANDLE hDevice,
                PPCI2306_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function PCI2306_LoadParaAD Lib "PCI2306_32" (_
                                ByVal hDevice As Long, _
                                ByRef pADPara As PCI2306_PARA_AD) As Long
```

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara属于PPCI2306_PARA_AD的结构指针类型，它负责返回PCI硬件参数值，关于结构指针类型PPCI2306_PARA_AD请参考PCI2306.h或PCI2306.Bas或PCI2306.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

◆ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPCI2306_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function PCI2306_SaveParaAD Lib "PCI2306_32" (_
                                    ByVal hDevice As Long, _
                                    ByRef pADPara As PCI2306_PARA_AD) As Long
```

LabVIEW:

请参考相关演示程序。

功能：负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara设备硬件参数，关于PCI2306_PARA_AD的详细介绍请参考PCI2306.h或PCI2306.Bas或PCI2306.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

第五节、DA 模拟量输出操作函数原型说明

◆ **向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）**

函数原型：

Visual C++:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       SHORT nDAData,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function PCI2306_WriteDeviceProDA Lib "PCI2306_32" (_
                                          ByVal hDevice As Long, _
                                          ByVal nDAData As Integer,_
                                          ByVal nDAChannel As Long) As Long
```

LabVIEW:

请参考相关演示程序。

功能　向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

nDAData准备输出的DA数据LSB原码，关于如何将电压数据转换成相应Lsb原码，请参考《DA电压值转换成LSB原码数据的换算方法》章节。

nDAChannel DA 通道，取值为(0，3)。

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

相关函数： CreateDevice　　WriteDeviceProDA　　ReleaseDevice

◆ **以上函数调用一般顺序**

① CreateDevice

② WriteDeviceProDA（往FIFO存储器中写入全满数据）

③ ReleaseDevice

第六节、CNT 计数与定时器操作函数原型说明

◆ **初始化计数与定时器**

函数原型：

Visual C++:

```
BOOL InitDevCounter ( HANDLE hDevice,
                      PPCI2306_PARA_COUNTER_CTRL pCtrlPara,
                      int InitCntrVal,
                      int nCntrChannel)
```

Visual Basic:

```
Declare Function PCI2306_InitDevCounter Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef pCntrCtrlPara As PCI2306_PARA_COUNTER_CTRL,_
                                        ByVal CounterValue As Long,_
                                        ByVal CounterChannel As Long) As Long
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的 CNT 部件，为设备的操作就绪做有关准备工作，初始化各路计数/定时器

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pCtrlPara设备对象的计数器控制字参数结构，它决定了设备对象的各种状态及工作方式，如功能模式、时钟源等。关于PARA_COUNTER_CTRL具体定义请参考PCI2306.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《CNT计数器参数结构》章节。

InitCntrVal 计数器的 16 位值，为 8253（或 8254）计数器控制字。它决定计数方式、工作模式等。

nCntrChannel 为 8253 计数器通道号，取值范围为(0-2)。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice InitDevCounter GetDevCounter ReleaseDevice

◆ **取得各路计数器的当前计数值**

函数原型：

Visual C++:

```
BOOL GetDevCounter (HANDLE hDevice,
                    PLONG pCntrValue,
                    int nCntrChannel)
```

Visual Basic:

```
Declare Function PCI2306_GetDevCounter Lib "PCI2306_32" (ByVal hDevice As Long,_
                                        ByRef pCntrValue As Long, _
                                        ByVal nCntrChannel As Long) As Long
```

LabVIEW:

请参考相关演示程序。

功能：取得各路计数器的当前计数值。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pCntrValue 返回计数值，取得 16 位计数器值。

nCntrChannel 计数器通道号，取值为(0-2)。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice InitDevCounter GetDevCounter ReleaseDevice

第七节、中断操作函数原型说明

◆ **初始化中断**

函数原型：

Visual C++:

```
BOOL InitDeviceInt (HANDLE hDevice,
                    HANDLE hEventInt,
                    PPCI2306_PARA_INT pCtrlPara)
```

Visual Basic:

```
Declare Function PCI2306_InitDeviceInt Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal hEventInt As Long,_
                                        ByVal pCtrlPara As PCI2306_PARA_INT) As Long
```

LabVIEW:
请参考相关演示程序。

功能：它负责初始化设备对象的硬件中断的方式工作。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
hEventInt中断事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEventInt事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。
pCtrlPara设备对象的中断参数结构，它决定了设备对象的各种状态及工作方式。关于PCI2306_PARA_INT具体定义请参考PCI2306.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《中断信息控制参数》章节。
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，并加以分析。
相关函数： CreateDevice InitDeviceInt GetDeviceIntCount GetDeviceIntSrc ReleaseDeviceInt ReleaseDevice

◆ **取得中断服务程序产生的次数**
函数原型：
Visual C++:
```
LONG GetDeviceIntCount (HANDLE hDevice)
```
Visual Basic:
```
Declare Function PCI2306_GetDeviceIntCount Lib "PCI2306_32" (ByVal hDevice As Long) As Long
```
LabVIEW:
请参考相关演示程序。

功能：在中断初始化后，用它取得中断服务程序产生的次数。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，返回取得中断服务程序产生的次数，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 InitDeviceInt GetDeviceIntCount GetDeviceIntSrc ReleaseDeviceInt ReleaseDevice

◆ **取得中断源**
函数原型：
Visual C++:
```
LONG GetDeviceIntSrc (HANDLE hDevice,
                      PPCI2306_PARA_INT pSrcPara)
```
Visual Basic:
```
Declare Function PCI2306_GetDeviceIntSrc Lib "PCI2306_32" (_
                                        ByVal hDevice As Long, _
                                        ByVal pSrcPara As PCI2306_PARA_INT) As Long
```
LabVIEW:
请参考相关演示程序。

功能：在中断初始化后，用它取得中断源。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pSrcPara设备对象的中断参数结构，它决定了设备对象的各种状态及工作方式。关于PCI2306_PARA_INT具体定义请参考PCI2306.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《中断信息控制参数》章节。
返回值：若成功，则返回取得的中断源，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 InitDeviceInt GetDeviceIntCount

GetDeviceIntSrc ReleaseDeviceInt ReleaseDevice

◆ **释放中断资源**

函数原型：

Visual C++:

BOOL ReleaseDeviceInt (HANDLE hDevice)

Visual Basic:

Declare Function PCI2306_ReleaseDeviceInt Lib "PCI2306_32" (ByVal hDevice As Long) As Long

LabVIEW:

请参考相关演示程序。

功能：释放中断资源。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

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，并加以分析。

相关函数： CreateDevice InitDeviceInt GetDeviceIntCount
GetDeviceIntSrc ReleaseDeviceInt ReleaseDevice

第八节、DIO 数字量输入输出开关量操作函数原型说明

◆ **开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   PPCI2306_PARA_DI pDIPara)
```

Visual Basic:

```
Declare Function PCI2306_GetDeviceDI Lib "PCI2306_32" (ByVal hDevice As Long,_
                                ByRef pDIPara As PCI2306_PARA_DI) As Long
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的输入开关量状态读入内存。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pDIPara 十六路开关量输入状态的参数结构，共有 16 个元素，分别对应于 DI0~DI15 路开关量输入状态位。如果 bDISts[0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

返回值：若成功，返回 TRUE，其 pPara 中的值有效；否则返回 FALSE，其 pPara 中的值无效。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

◆ **开关量输出**

函数原型：

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  PPCI2306_PARA_DO pDOPara)
```

Visual Basic:

```
Declare Function PCI2306_SetDeviceDO Lib "PCI2306_32" (ByVal hDevice As Long,_
                                ByRef pDOPara As PCI2306_PARA_DO) As Long
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的输出开关量置成相应的状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pDOPara 十六路开关量输出状态的参数结构，共有 16 个成员变量，分别对应于 DO0~DO15 路开关量输出状态位。比如置 pPara->DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO15 共 16 个成员变量赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

◆ **以上函数调用一般顺序**

① CreateDevice
② SetDeviceDO(或GetDeviceDI，当然这两个函数也可同时进行)
③ ReleaseDevice

用户可以反复执行第②步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及 AD 采样可以同时进行，互不影响）。

第四章 硬件参数结构

第一节、AD 硬件参数结构（PCI2306_PARA_AD）

Visual C++：

```
typedef struct _PCI2306_PARA_AD
{
    LONG FirstChannel;          // 首通道
    LONG LastChannel;           // 末通道
} PCI2306_PARA_AD, *PPCI2306_PARA_AD;
```

Visual Basic：

```
Type PCI2306_PARA_AD        ' 板卡各参数值
    FirstChannel As Long ' 首通道
    LastChannel As Long   ' 末通道
End Type
```

LabVIEW:

请参考相关演示程序。

该结构实在太简易了，其原因就是 PCI 设备是系统全自动管理的设备，再加上驱动程序的合理设计与封装，什么端口地址、中断号等将与 PCI 设备的用户永远告别，一句话 PCI 设备是一种更易于管理和使用的设备。

此结构主要用于设定设备AD硬件参数值，用这个参数结构对设备进行硬件配置完全由InitDeviceProAD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。

硬件参数说明:

FirstChannel AD采样首通道，其取值范围为[0, 31]，它应等于或小于LastChannel参数。
LastChannel AD采样末通道，其取值范围为[0, 31]，它应等于或大于FirstChannel参数。

相关函数： CreateDevice LoadParaAD SaveParaAD
ReleaseDevice

第二节、CNT 计数器参数结构（PCI2306_PARA_COUNTER_CTRL）

Visual C++:

```
typedef struct _PCI2306_PARA_COUNTER_CTRL
{
    BYTE OperateType;     // 操作类型(D5D4)
    BYTE CountMode;       // 计数方式(D3-D1)
    BYTE BCD;             // 计数类型(D0)
    BYTE ClockSource;     // 时钟基准源选择
    BYTE GateSource;      // GATE 门信号源选择
} PCI2306_PARA_COUNTER_CTRL, *PPCI2306_PARA_COUNTER_CTRL;
```

Visual Basic：

```
Type PCI2306_PARA_COUNTER_CTRL
```

```
        OperateType As Byte
        CountMode As Byte
        BCD As Byte
        ClockSource As Byte
        GateSource As Byte
    End Type
```

LabVIEW:

请参考相关演示程序。

OperateType 操作类型(D5D4)。

常量名	常量值	功能定义
PCI2306_OperateType_0	0x0000	计数器锁存操作
PCI2306_OperateType_1	0x0001	只读/写低字节
PCI2306_OperateType_2	0x0002	只读/写高字节
PCI2306_OperateType_3	0x0003	先读/写低字节，后读/写高字节

CountMode 计数方式(D3-D1)。

常量名	常量值	功能定义
PCI2306_CountMode_0	0x0000	计数方式 0，计数器结束中断方式
PCI2306_CountMode_1	0x0001	计数方式 1，可编程单次脉冲方式
PCI2306_CountMode_2	0x0002	计数方式 2，频率发生器方式
PCI2306_CountMode_3	0x0003	计数方式 3，方波频率发生器方式
PCI2306_CountMode_4	0x0004	计数方式 4，软件触发选通方式
PCI2306_CountMode_5	0x0005	计数方式 5，硬件触发选通方式

BCD 计数类型(D0)。

常量名	常量值	功能定义
PCI2306_BCD_0	0x0000	计数类型 0，二进制计数
PCI2306_BCD_1	0x0001	计数类型 1，BCD 码计数

ClockSource 时钟基准源选择，其选项如下：

常量名	常量值	功能定义
PCI2306_IN_CLOCK	0x0000	内部时钟定时触发(板上 10MHz)
PCI2306_OUT_CLOCK	0x0001	外部时钟定时触发
PCI2306_OTHER_CLOCK	0x0002	其他计数器的 OUT 输出提供的 Clock(CLK2 接 OUT1,CLK1 接 OUT0, CLK0 接外部 CLK)

GateSource GATE 门信号源选择，其选项如下：

常量名	常量值	功能定义
PCI2306_LOW_VOLT_GATE	0x0000	低电平(内部软件自动实现)
PCI2306_HIGH_VOLT_GATE	0x0001	高电平(内部软件自动实现)
PCI2306_OUTSIDE_GATE	0x0002	外部 GATE 信号输入(D 型头上)
PCI2306_NOUTSIDE_GATE	0x0003	外部 GATE 信号反向输入(D 型头上)

相关函数： CreateDevice InitDevCounter GetDevCounter ReleaseDevice

第三节、中断信息控制参数（PCI2306_PARA_INT）

Visual C++:

```
typedef struct _PCI2306_PARA_INT     // 中断信息控制参数
{
    LONG bDI0;              // 开关输入 0 通道跳变触发中断(信号类型可选, =0 禁止)
```

```
    LONG bOutTrigger;     // D 型头上的外部信号触发中断(信号类型可选, =0 禁止)
    LONG bADTrigger;      //  允许 AD 转换结束信号触发中断(=1:允许; =0:禁止)
    LONG bCounter0Out; //  允许计数器 0 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
    LONG bCounter1Out; //  允许计数器 1 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
    LONG bCounter2Out; //  允许计数器 2 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)
} PCI2306_PARA_INT, *PPCI2306_PARA_INT;
```

Visual Basic：

```
  Type PCI2306_PARA_INT      ' 中断信息控制参数
       bDI0 As Long          ' 开关输入 0 通道跳变触发中断(信号类型可选, =0 禁止)
       bOutTrigger As Long     'D 型头上的外部信号触发中断(信号类型可选, =0 禁止)
       bADTrigger As Long      ' 允许 AD 转换结束信号触发中断(=1:允许  =0:禁止)
       bCounter0Out As Long      ' 允许计数器 0 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
       bCounter1Out As Long    ' 允许计数器 1 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
      bCounter2Out As Long       ' 允许计数器 2 的输出信号触发中断(=TRUE/1:允许  =FALSE/0:禁止)
End Type
```

LabVIEW:

请参考相关演示程序。

bDI0 开关输入 0 通道跳变触发中断(信号类型可选, =0 禁止)。

常量名	常量值	功能定义
PCI2306_RISING_EDGE	0x0000	上升沿触发中断
PCI2306_FALLING_EDGE	0x0001	下降沿触发中断
PCI2306_ALL_EDGE	0x0002	上下降沿触发中断

bOutTrigger D 型头上的外部信号触发中断(信号类型可选, =0 禁止)。

常量名	常量值	功能定义
PCI2306_RISING_EDGE	0x0000	上升沿触发中断
PCI2306_FALLING_EDGE	0x0001	下降沿触发中断
PCI2306_ALL_EDGE	0x0002	上下降沿触发中断

bADTrigger 允许 AD 转换结束信号触发中断(=1:允许; =0:禁止)。
bCounter0Out 允许计数器 0 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)。
bCounter1Out 允许计数器 1 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)。
bCounter2Out 允许计数器 2 的输出信号触发中断(=TRUE/1:允许; =FALSE/0:禁止)。

相关函数： CreateDevice　InitDeviceInt　GetDeviceIntCount
GetDeviceIntSrc　ReleaseDeviceInt　ReleaseDevice

第四节、数字量输入参数（PCI2306_PARA_DI）

Visual C++:

```
typedef struct _PCI2306_PARA_DI        // 数字量输入参数
{
    BYTE DI0;          // 0 通道
    BYTE DI1;          // 1 通道
    BYTE DI2;          // 2 通道
    BYTE DI3;          // 3 通道
    BYTE DI4;          // 4 通道
    BYTE DI5;          // 5 通道
    BYTE DI6;          // 6 通道
    BYTE DI7;          // 7 通道
    BYTE DI8;          // 8 通道
    BYTE DI9;          // 9 通道
```

```
    BYTE DI10;          // 10 通道
    BYTE DI11;          // 11 通道
    BYTE DI12;          // 12 通道
    BYTE DI13;          // 13 通道
    BYTE DI14;          // 14 通道
    BYTE DI15;          // 15 通道
} PCI2306_PARA_DI,*PPCI2306_PARA_DI;
```

Visual Basic：

```
Type PCI2306_PARA_DI
    DI0 As Byte   ' 0 通道
    DI1 As Byte   ' 1 通道
    DI2 As Byte   ' 2 通道
    DI3 As Byte   ' 3 通道
    DI4 As Byte   ' 4 通道
    DI5 As Byte   ' 5 通道
    DI6 As Byte   ' 6 通道
    DI7 As Byte   ' 7 通道
    DI8 As Byte   ' 8 通道
    DI9 As Byte   ' 9 通道
    DI10 As Byte   ' 10 通道
    DI11 As Byte   ' 11 通道
    DI12 As Byte   ' 12 通道
    DI13 As Byte   ' 13 通道
    DI14 As Byte   ' 14 通道
    DI15 As Byte   ' 15 通道
End Type
```

该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要执行GetDeviceDI即可完成数字量输入操作。然后象Visual Basic中的属性操作那样，简单的进行属性成员分析即可确定各路状态。

关于LabView的参数，由于需要的是返回值，因此根据LabView的特点，应分配一个 16 字节的内存单元，每一个字节的内存单元对应相应位置上的开关量输入状态。要使用这些状态，则应在GetDeviceDI之后，将存放实际的当前开关量状态的内存单元用Index Array数组操作控件将其每一路开关量状态分离出来，即可确定每一路开关输入状态。详见开关量输入输出LabView演示部分。

其每一个成员变量对应于相应的 DI 通道，即 DI0-DI15 分别对应于 DI 通道 0-15。且这些成员变量只能是“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

第五节、数字量输出参数（PCI2306_PARA_DO）

Visual C++:

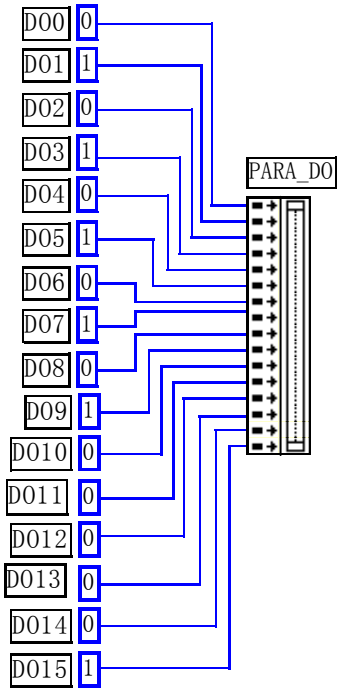
```
typedef struct _PCI2306_PARA_DO        // 数字量输出参数
{
    BYTE DO0;          // 0 通道
    BYTE DO1;          // 1 通道
    BYTE DO2;          // 2 通道
    BYTE DO3;          // 3 通道
    BYTE DO4;          // 4 通道
    BYTE DO5;          // 5 通道
    BYTE DO6;          // 6 通道
    BYTE DO7;          // 7 通道
    BYTE DO8;          // 8 通道
```

```
    BYTE DO9;          // 9 通道
    BYTE DO10;         // 10 通道
    BYTE DO11;         // 11 通道
    BYTE DO12;         // 12 通道
    BYTE DO13;         // 13 通道
    BYTE DO14;         // 14 通道
    BYTE DO15;         // 15 通道
} PCI2306_PARA_DO,*PPCI2306_PARA_DO;
```

Visual Basic：

```
Type PCI2306_PARA_DO
    DO0 As Byte   ‘ 0 通道
    DO1 As Byte   ‘ 1 通道
    DO2 As Byte   ‘ 2 通道
    DO3 As Byte   ‘ 3 通道
    DO4 As Byte   ‘ 4 通道
    DO5 As Byte   ‘ 5 通道
    DO6 As Byte   ‘ 6 通道
    DO7 As Byte   ‘ 7 通道
    DO8 As Byte   ‘ 8 通道
    DO9 As Byte   ‘ 9 通道
    DO10 As Byte   ‘ 10 通道
    DO11 As Byte   ‘ 11 通道
    DO12 As Byte   ‘ 12 通道
    DO13 As Byte   ‘ 13 通道
    DO14 As Byte   ‘ 14 通道
    DO15 As Byte   ‘ 15 通道
End Type
```

LabView:



该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要象Visual Basic中的属性操作那样，只需要有简单的进行属性赋值，然后执行SetDeviceDO即可完成数字量输出。注意关于LabView的参数定义，他最主要表达了在LabView环境中怎样使用SetDeviceDO实现开关量输出操作的基本实现方法。在用户实际使用中，您可以将左边的常量图标换成开关控件图标等，以实现动态改变开关量输出状态。但需要注意的是开关控件图标（xxx Switch）输出的值是布尔变量，因此在开关控件图

标与PPCI2306_PARA_DO之间，应使用Boolean To (0,1)逻辑转换控件，即先将布尔变量转换成 0 或 1 的整型值，再将这个整型值传递给PPCI2306_PARA_DO，详见开关量输入输出LabView演示部分。

其每一个成员变量对应于相应的 DO 通道，即 DO0-DO15 分别对应于 DO 通道 0-15。且这些成员变量只能被赋值为“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/4096)*(ADBuffer[0] &0x0FFF)–10000.00	[-10000, +9995.11]
±5000mV	Volt = (10000.00/4096)*(ADBuffer[0] &0x0FFF)–5000.00	[-5000, +4997.55]
0～10000mV	Volt = (10000.00/4096)*(ADBuffer[0] &0x0FFF)	[0, +9997.55]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++：

```
Lsb = (ADBuffer[0]) & 0x0FFF;
Volt = (20000.00/4096) * Lsb -10000.00;
```

Visual Basic：

```
Lsb = (ADBuffer [0]) And &0H0FFF
Volt = (20000.00/4096) * Lsb – 10000.00
```

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，假如此时选择通道 5，其排放规则如下：

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	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	…

两通道采集(假如通道 0 和 1)：

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	0	1	0	1	0	1	0	1	0	1	0	1	0	…

四通道采集(假如通道 0~通道 3)：

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	…

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环

处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效呢？还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	…
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	…
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	…
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeaderSizeBytes;          // 文件头信息长度
    LONG FileType;                 // 指文件用处和类型

    // 该设备数据文件共有的成员
    LONG BusType;                  // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;                // 该设备的 ID 编号(DEFAULT_DEVICE_NUM)

    LONG VoltBottomRange;          // 量程下限(mV)
    LONG VoltTopRange;             // 量程上限(mV)
    LONG FirstChannel;             // 首通道
    LONG LastChannel;              // 末通道
    // 该设备具体参数
    PCI2306_PARA_AD ADPara;        // 保存硬件参数
    LONG FileEndFlag;              // 文件结束标志(等于 FILE_END_FLAG)
} FILE_HEADER, *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第四节、DA 电压值转换成 LSB 原码数据的换算方法

量程(伏)	计算机语言换算公式(标准 C 语法)	Lsb 取值范围
0～5000mV	Lsb = Volt / (5000.00 / 4096)	[0，4095]
0～10000mV	Lsb = Vol t/ (10000.00 / 4096)	[0，4095]
±5000mV	Lsb = Volt / (10000.00 / 4096) +2048	[0，4095]
±10000mV	Lsb = Volt / (20000.00 / 4096) + 2048	[0，4095]

第六章 上层用户函数接口应用实例

第一节、怎样使用ReadDevOneAD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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[简易代码演示]｜[AD 单点采集]

第二节、怎样使用ReadDevBulkAD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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[简易代码演示]｜[AD 批量采集]

第三节、怎样使用WriteDeviceDA函数取得DA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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[简易代码演示]｜[DA 输出]

第四节、怎样使用计数器取得数据

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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[简易代码演示]｜[Counter 方式]

第五节、怎样使用GetDeviceDI函数进行更便捷的数字开关量输入操作

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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[简易代码演示]｜[DIO…]

第六节、怎样使用SetDeviceDO函数进行更便捷的数字开关量输出操作

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]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft

第七章 高速大容量、连续不间断数据采集及存盘技术详解

与ISA、USB设备同理，使用子线程跟踪AD转换进度，并进行数据采集是保持数据连续不间断的最佳方案。但是与ISA总线设备不同的是，PCI设备在这里不使用动态指针去同步AD转换进度，因为ISA设备环形内存池的动态指针操作是一种软件化的同步，而PCI设备不再有软件化的同步，而完全由硬件和驱动程序自动完成。这样一来，用户要用程序方式实现连续数据采集，其软件实现就显得极为容易。每次用ReadDevOneAD (或ReadDevBulkAD)函数读取AD数据时，那么设备驱动程序会按照AD转换进度将AD数据一一放进用户数据缓冲区，当完成该次所指定的点数时，它便会返回，当您再次用这个函数读取数据时，它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次ReadDevOneAD (或ReadDevBulkAD)之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中，由于任务之间的调度切换非常平凡，特别是当用户移动窗口、或弹出对话框等，则会使当前线程猛地花掉大量的时间去处理这些图形操作，因此如果处理不当，则将无法实现高速连续不间断采集，那么如何更好的克服这些问题呢？用子线程则是必须的（在这里我们称之为数据采集线程），但这还不够，必须要求这个线程是绝对的工作者线程，即这个线程在正常采集中不能有任何窗口等图形操作。只有这样，当用户进行任何窗口操作时，这个线程才不会被堵塞，因此可以保证其正常连续的数据采集。但是用户可能要问，不能进行任何窗口操作，那么我如何将采集的数据显示在屏幕上呢？其实很简单，再开辟一个子线程，我们称之数据处理线程，也叫用户界面线程。最初，数据处理线程不做任何工作，而是在 Win32 API 函数 WaitForSingleObject 的作用下进入睡眠状态，此时它基本不消耗 CPU 时间，即可保证其他线程代码有充分的运行机会（这里当然主要指数据采集线程），当数据采集线程取得指定长度的数据到用户空间时，则再用 Win32 API 函数 SetEvent 将指定事件消息发送给数据处理线程，则数据处理线程即刻恢复运行状态，迅速对这批数据进行处理，如计算、在窗口绘制波形、存盘等操作。

可能用户还要问，既然数据处理线程是非工作者线程，那么如果用户移动窗口等操作堵塞了该线程，而数据采集线程则在不停地采集数据，那数据处理线程难道不会因此而丢失采集线程发来的某一段数据吗？如果不另加处理，这个情况肯定有发生的可能。但是，我们采用了一级缓冲队列和二级缓冲队列的设计方案，足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K 数据，那么我们就创建一个缓冲队列，在用户程序中最简单的办法就是开辟一个两维数组如 ADBuffer [SegmentCount][SegmentSize]，我们将 SegmentSize 视为数据采集线程每次采集的数据长度，SegmentCount 则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32，则这个缓冲队列实际上就是数组 ADBuffer [32][8192]的形式。那么如何使用这个缓冲队列呢？方法很简单，它跟一个普通的缓冲区如一维数组差不多，唯一不同是，两个线程首先要通过改变 SegmentCount 字段的值，即这个下标 Index 的值来填充和引用由 Index 下标指向某一段 SegmentSize 长度的数据缓冲区。需要注意的是两个线程不共用一个 Index 下标变量。具体情况是当数据采集线程在 AD 部件被初始化之后，首次采集数据时，则将自己的 ReadIndex 下标置为 0，即用第一个缓冲区采集 AD 数据。当采集完后，则向数据处理线程发送消息，且两个线程的公共变量 SegmentCount 加 1，（注意 SegmentCount 变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了，但是却没被数据处理线程处理掉的缓冲区数量。）然后再接着将 ReadIndex 偏移至 1，再用第二个缓冲区采集数据。再将 SegmentCount 加 1，直到 ReadIndex 等于 31 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从 SegmentCount 变量中减去在所接受到的当前事件下所处理的缓冲区个数，具体处理哪个缓冲区由 CurrentIndex 指向。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对 SegmentCount 加以判断，观察其值是否大于了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。

图 7.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程填充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

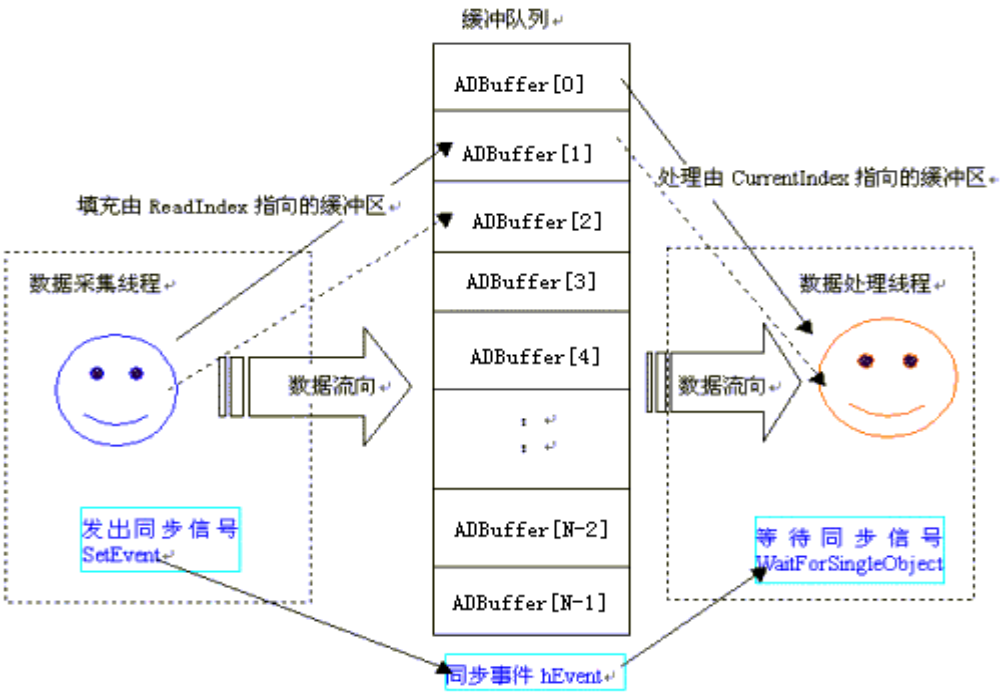


图 7.1

第一节、使用程序查询方式实现该功能

下面用 Visual C++程序举例说明。

一、使用ReadDevOneAD函数单点读取设备上的AD数据

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(ADDoc.h 和 ADDoc.cpp， ADThread.h 和 ADThread.cpp)。

[程序]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[高级代码演示]｜[演示源程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc::StartDeviceAD()          // 启动线程函数
UINT ReadDataThread (PVOID hWnd)   // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)   // 绘制数据线程
void CADDoc::StopDeviceAD()          // 终止采集函数
```

二、使用ReadDevBulkAD函数批量读取设备上的AD数据

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(ADDoc.h 和 ADDoc.cpp， ADThread.h 和 ADThread.cpp)。

[程序]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[高级代码演示]｜[演示源程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc::StartDeviceAD()          // 启动线程函数
UINT ReadDataThread (PVOID hWnd)   // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)   // 绘制数据线程
void CADDoc::StopDeviceAD()          // 终止采集函数
```

当然用 FIFO 非空标志读取 AD 数据，能获得接近 FIFO 总容量的栈深度，这样用户在两批数据之间，便有更多的时间来处理某些数据。而用半满标志，则最多只能达到 FIFO 总容量的二分之一的栈深度，那么用户在两批数据之间处理数据的时间会相对短些，但是半满读取时，查询 AD 转换标志的时间则最少。当然究意那种方案最好，还得看用户的实际需要。

第二节、使用中断方式实现该功能

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(ADDoc.cpp 和 ADThread.cpp)。

[程序]｜[阿尔泰测控演示系统]｜[PCI2306 16 路 100KhzAD、4 路 DA、16 路 DIO 和计数器卡]｜[Microsoft Visual C++]｜[高级代码演示]｜[演示源程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc:: OnStartDeviceAD ()        // 采集线程和处理线程的启动函数
BOOL StartDeviceAD_Int ()        // 启动采集线程函数
UINT ReadDataThread ()        // 采集线程函数
BOOL StopDeviceAD_Int()        // 采集线程的终止函数
void DrawWindowProc ()                // 绘制数据线程
void CADDoc:: OnStopDeviceAD ()        // 终止采集函数
```

第八章 共用函数介绍

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

第一节、公用接口函数总列表（每个函数省略了前缀“PCI2306_”）

函数名	函数功能	备注
① ISA 总线 I/O 端口操作函数		
GetDeviceAddr	取得指定 PCI 设备寄存器操作基地址	底层用户
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
② 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateVBThread	在 VB 环境中建立子线程对象	
TerminateVBThread	终止 VB 的子线程	
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	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间

第二节、I0 端口读写函数原型说明

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

◆ **取得指定内存映射寄存器的线性地址和物理地址**

函数原型：

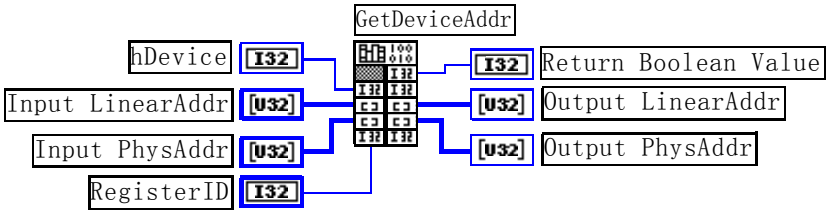
Visual C++:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    __int64 LinearAddr,
                    __int64 PhysAddr,
                    int RegisterID)
```

Visual Basic:

```
Declare Function PCI2306_GetDeviceAddr Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByRef LinearAddr As Long,_
                                        ByRef PhysAddr As Long,_
                                        ByVal RegisterID As Long) As Boolean
```

LabVIEW:



功能：取得 PCI 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。

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　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord　ReadPortULong　ReleaseDevice

Visual C++程序举例:

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

Visual Basic 程序举例:

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

◆ **以单字节(8Bit)方式写 I/O 端口**

函数原型：

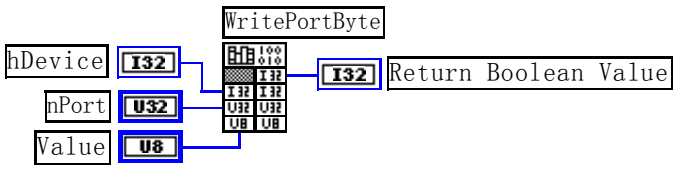
Visual C++:

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

Visual Basic:

```
Declare Function PCI2306_WritePortByte Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal PbPort As Long, _
                                        ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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 GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以双字(16Bit)方式写 I/O 端口**

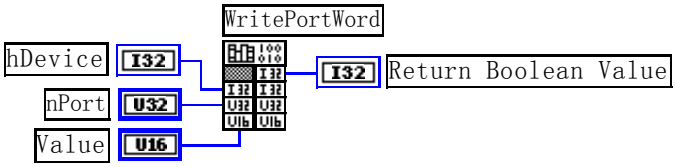
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function PCI2306_WritePortWord Lib "PCI2306_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal PbPort As Long,_
                                    ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice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 GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以四字节(32Bit)方式写 I/O 端口**

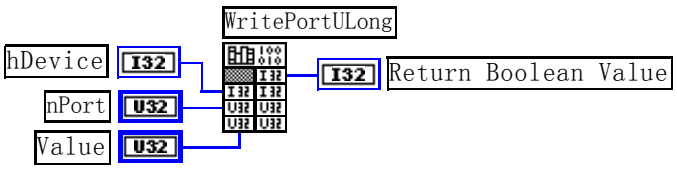
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function PCI2306_WritePortUlong Lib "PCI2306_32" (_
                                    ByVal hDevice As Long,_
                                    ByVal PbPort As Long,_
                                    ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice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 GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以单字节(8Bit)方式读 I/O 端口**

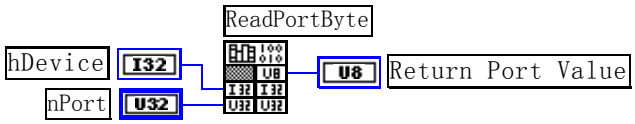
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function PCI2306_ReadPortByte Lib "PCI2306_32" (_
                                        ByVal hDevice As Long, _
                                        ByVal PbPort As Long) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice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 GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以双字节(16Bit)方式读 I/O 端口**

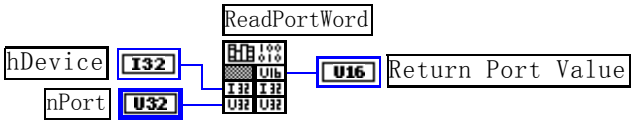
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function PCI2306_ReadPortWord Lib "PCI2306_32" (_
                                        ByVal hDevice As Long, _
                                        ByVal PbPort As Long) As Integer
```

LabVIEW:

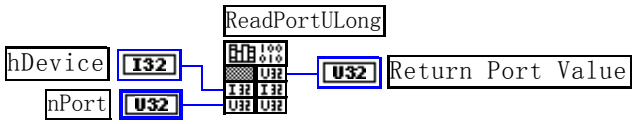


功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。

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　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

◆ **以四字节(32Bit)方式读 I/O 端口**

函数原型：
Visual C++:
```
ULONG ReadPortULong(HANDLE hDevice,
                    __int64 pbPort)
```
Visual Basic:
```
Declare Function PCI2306_ReadPortUlong Lib "PCI2306_32" (_
                                        ByVal hDevice As Long,_
                                        ByVal PbPort As Long) As Long
```
LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice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　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

第三节、线程操作函数原型说明

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

◆ **在 VB 环境中，创建子线程对象，以实现多线程操作**

函数原型：
Visual C++:
```
BOOL CreateVBThread(HANDLE *hThread,
                    LPTHREAD_START_ROUTINE StartThread)
```
Visual Basic：
```
Declare Function PCI2306_CreateVBThread Lib "PCI2306_32" (hThread As Long,_
                                        ByVal pStartThread As Long) As Long
```
功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题。通过该函数用户可以很轻松地实现多线程操作。
参数：
hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程、暂停线程以及删除线程等。
StartThread作为子线程运行的函数的地址，在实际使用时，请用AddressOf关键字取得该子线程函数的地址，再传递给CreateVBThread函数。
返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateVBThread　TerminateVBThread

注意: StartThread 指向的函数或过程必须放在 VB 的模块文件中，如 PCI2306.Bas 文件中。

Visual Basic 程序举例:
```
' 在模块文件中定义子线程函数(注意参考实例)
```

```
Function NewRoutine() As Long        ' 定义子线程函数
        :                            ' 线程运行代码
NewRoutine = 1    ' 返回成功码
End Function
'
' 在窗体文件中创建子线程
        :
Dim hNewThread As Long
    If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
        MsgBox "创建子线程失败"
        Exit Sub
    End If
    ResumeThread (hNewThread)   '启动新线程          :
        :
```

◆ **在 VB 中，删除子线程对象**

函数原型：

Visual C++:

BOOL TerminateVBThread (HANDLE hThreadHandle)

Visual Basic:

Declare Function PCI2306_TerminateVBThread Lib "PCI2306_32" (ByVal hThreadHandle As Long) As Long

功能：在VB中删除由CreateVBThread创建的子线程对象。

参数：hThreadHandle 指向需要删除的子线程对象的句柄，它应由CreateVBThread创建。

返回值：当成功删除子线程对象时，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

Visual Basic 程序举例:

```
        :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
        :
```

◆ **创建内核系统事件**

函数原型：

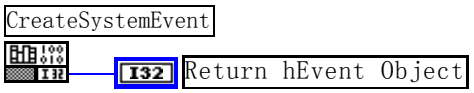
Visual C++:

HANDLE CreateSystemEvent(void)

Visual Basic:

Declare Function PCI2306_CreateSystemEvent Lib "PCI2306_32" () As Long

LabVIEW：



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

◆ **释放内核系统事件**

函数原型：

Visual C++：

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function PCI2306_ReleaseSystemEvent Lib "PCI2306_32" (ByVal hEvent As Long) As Long

LabVIEW：

请参见相关演示程序。

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

- **高效高精度延时**

函数原型：

Visual C++：

```
BOOL DelayTimeUs (HANDLE hDevice,
                  LONG nTimeUs)
```

Visual Basic：

```
Declare Function DelayTime Lib "PCI2306" (ByVal hDevice As Long, _
                                          ByVal nTimeUs As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：微秒级延时函数。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nTimeUs 时间常数。单位 1 微秒。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获错误码。
相关函数：无。