PCI2600 光纤通讯卡

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 PCI2600_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InitDeviceProAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD_ NotEmpty (或ReadDeviceProAD_Half)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、如何用非空查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD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StartDeviceProAD即可启动AD部件，开始AD采样，然后便可用ReadDeviceProAD_ NotEmpty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StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。

（注：ReadDeviceProAD_ NotEmpty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 2.1.1。

第四节、如何用半满查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD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StartDeviceProAD即可启动AD部件，开始AD采样，接着调用GetDevStatusProAD函数以查询AD的存储器FIFO的半满状态，如果达到半满状态，即可用ReadDeviceProAD_Half函数读取一批半满长度（或半满以下）的AD数据，然后接着再查询FIFO的半满状态，若有效再读取，就这样反复查询状态反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceProAD_Half函数在半满状态有效时也可以单点或几点的方式读取AD数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）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InitDeviceIntAD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InitDeviceIntAD的相应参数，它将作为接受AD半满中断事件的变量。然后用StartDeviceIntAD即可启动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ReadDeviceIntAD函数一批半满长度（或半满以下）的AD数据，然后再接着再等待FIFO的半满中断事件，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。当您需要暂停设备时，执行StopDeviceIntAD，当您需要关闭AD设备时，ReleaseDeviceIntAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceIntAD函数在半满中断事件发生时可以单点或几点的方式读取AD数据，只是到下一次半满中断事件到来时的时间间隔会变得非常短，而不再是半满间隔，但它不同于半满查询方式读取，由于半满中断属于硬件中断，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。　切忌、切忌！）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。

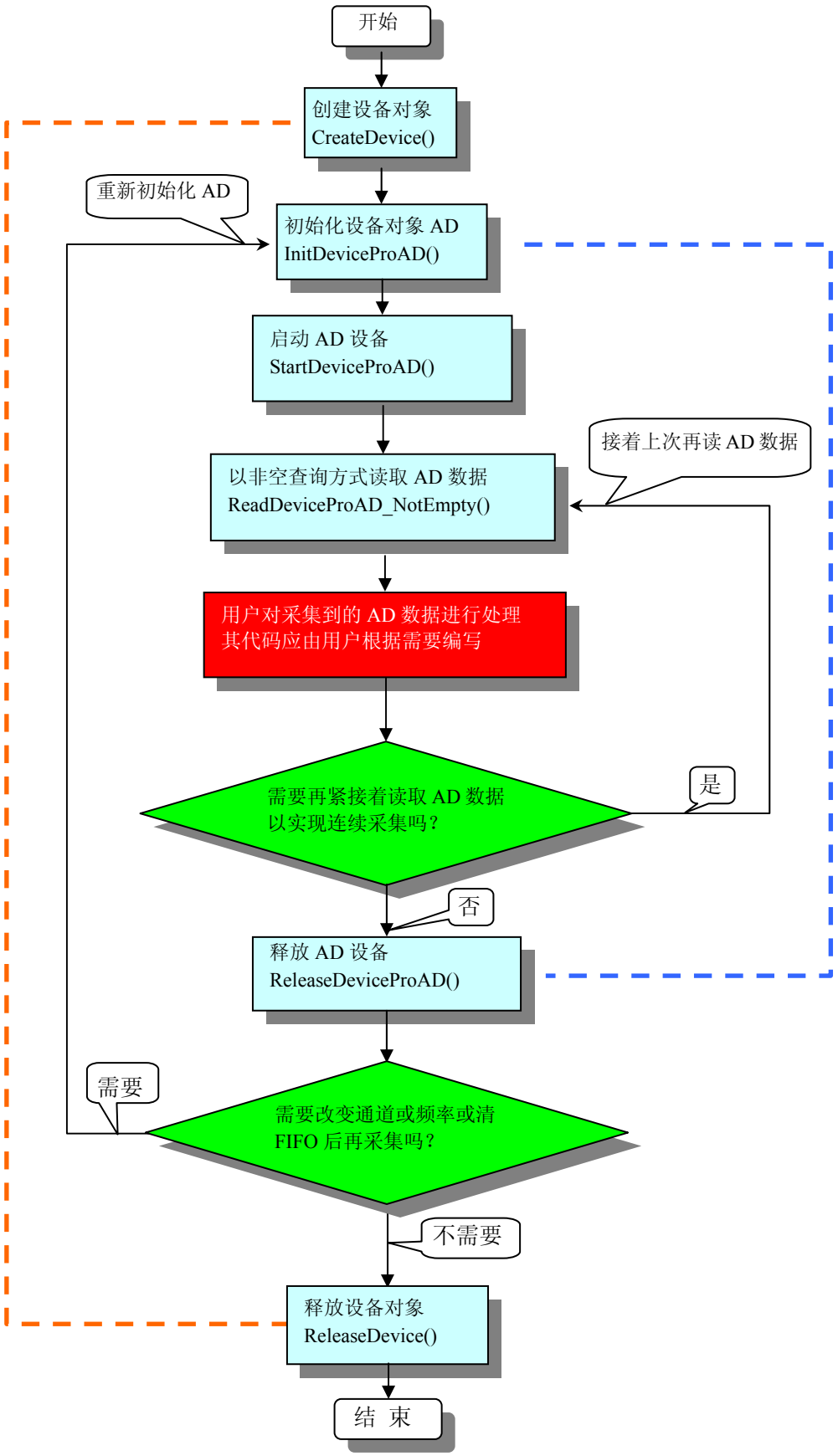


图 2.1.1 非空查询方式 AD 采集过程

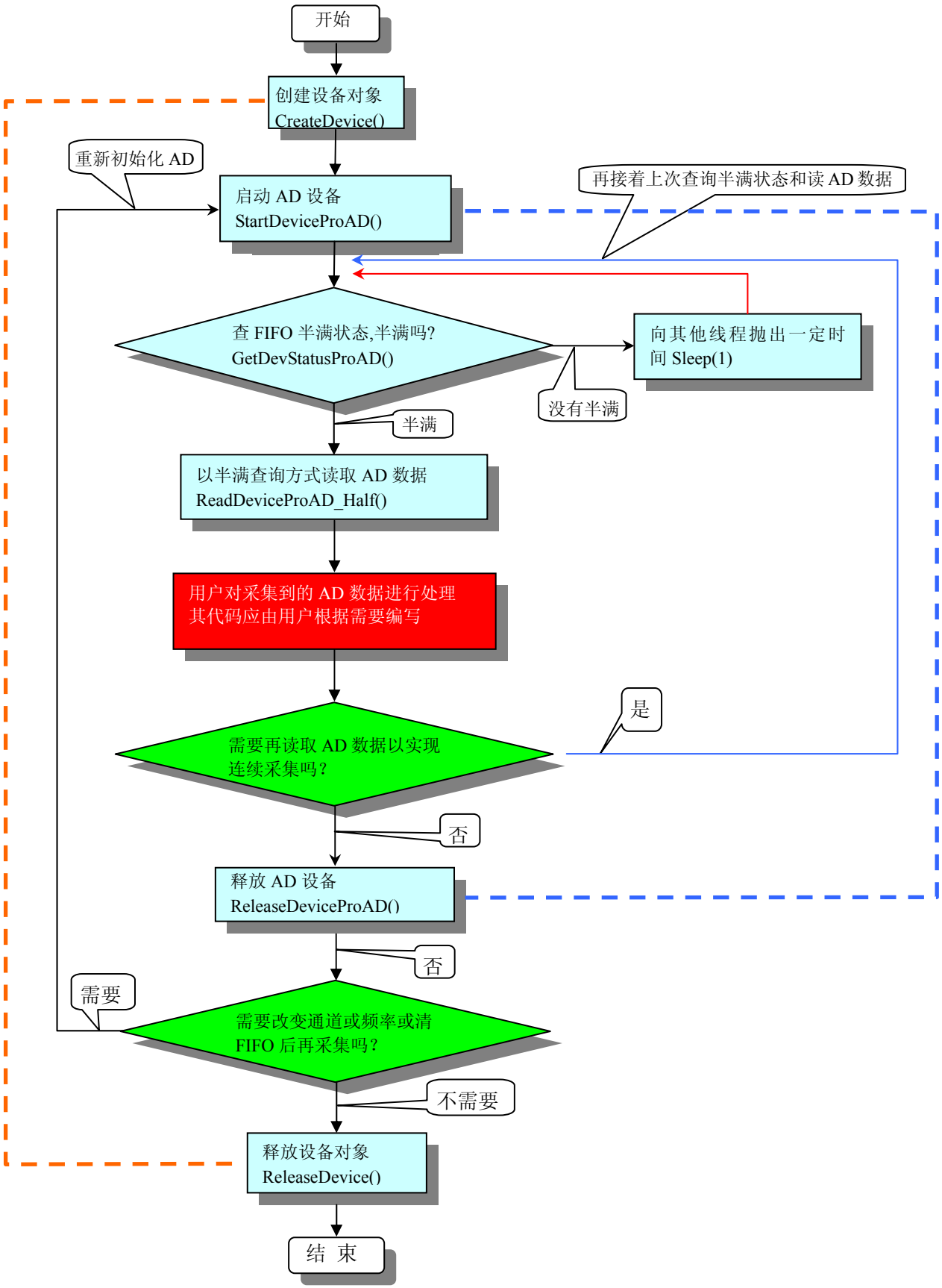


图 2.1.2 半满查询方式 AD 采集过程

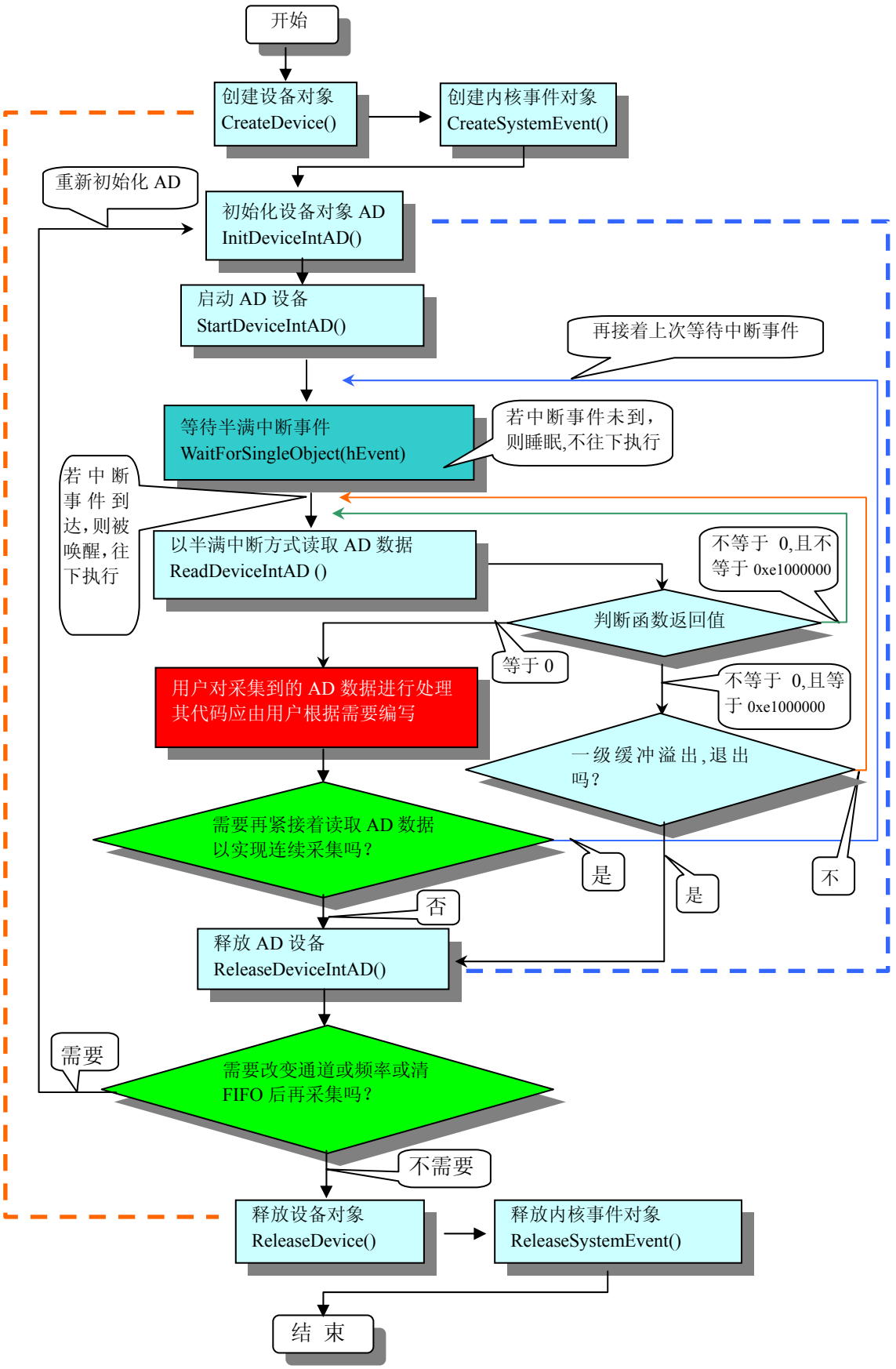


图 2.1.3 中断方式 AD 采集实现过程

第六节、哪些函数对您不是必须的

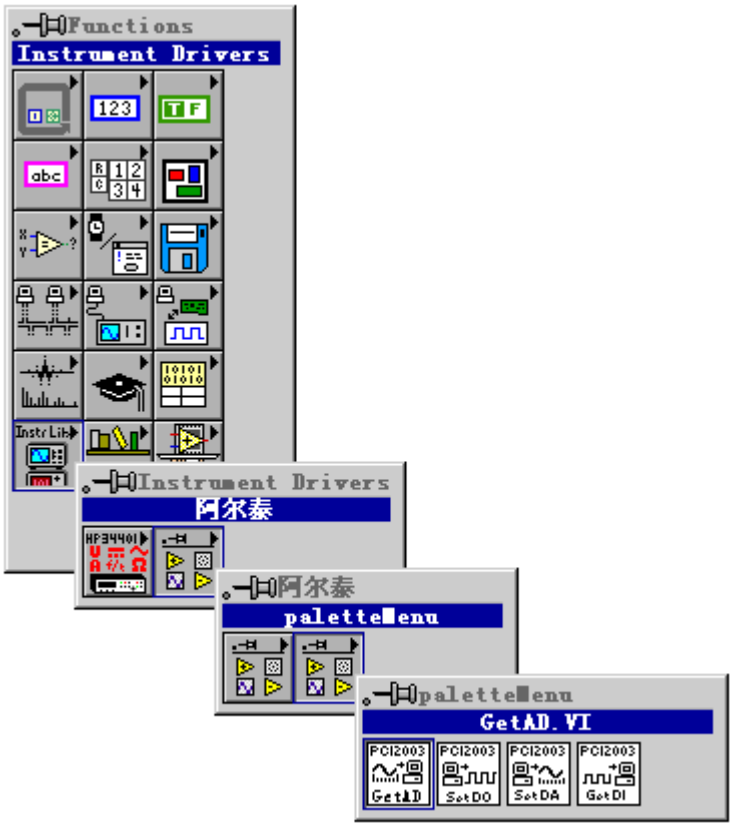
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， 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（如InitDeviceProAD）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ReadDeviceProAD_NotEmpty（或ReadDeviceProAD_Half）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。

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“PCI2600_”）

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
CreateDeviceEx	用物理号创建设备对象	上层及底层用户
GetDeviceCurrentID	取得指定设备句柄指向的设备 ID 号	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 发送器传输控制函数		
StartDeviceSend	使能发送器设备	上层用户
ClrDeviceSendFifo	清发送器 FIFO	上层用户
GetDevStatusSend	获得发送状态	上层用户
WriteDeviceData_Pro	用查询方式写入数据	
WriteDeviceData_Dma	用 DMA 方式写入数据	上层用户
StopDeviceSend	禁止发送器设备	上层用户
③ 接收器传输控制函数		
StartDeviceReceive	使能接收器设备	上层用户
ClrDeviceReceiveFifo	清接收器 FIFO	上层用户
ReadDevicePro_Npt	非空标志读取设备上的 AD 数据	上层用户
GetDevStatusReceive	获得接收状态	上层用户
ReadDevicePro_Half	FIFO 半满读 AD 数据	上层用户
ReadReceiveDeviceDma	直接内存(DMA)方式函数	上层用户
StopDeviceReceive	禁止接收器设备	上层用户
④ 中断方式 AD 读取函数		
InitDeviceInt	初始化 PCI 设备 AD 部件	上层用户
GetDeviceIntCount	得到中断次数	上层用户

ReleaseDeviceInt	释放中断	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\PCI2600\INCLUDE\PCI2600.H”

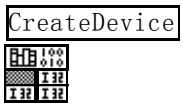
注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PCI2600.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句：

#include “PCI2600.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。

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 PCI2600.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ReadDeviceProAD_ NotEmpty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

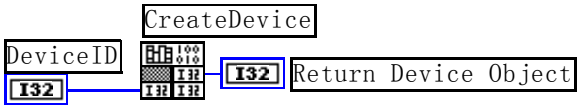
◆ **创建设备对象函数（逻辑号）**

函数原型：

Visual C++：

HANDLE CreateDevice (int DeviceID = 0)

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　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例：

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

♦ **创建设备对象函数（物理号）**

函数原型：

Visual C++：

HANDLE CreateDeviceEx (int DevicePhysID = 0)

LabVIEW：

请参考相关演示程序。

功能：该函数使用物理 ID 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：

DevicePhysID 物理设备ID(Physic Device Identifier)标识号。由CreateDevice函数的DeviceLgcID参数说明中可以看出，逻辑ID号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如A、B、C、D四个卡，构成 128 个通道（32*4），其通道序列为 0-127，每个通道接入不同物理意义的模拟信号，我们要求A卡位于 0-31 通道上，B卡位于 32-63 通道上，C卡位于 64-95 通道上，而D卡则位于 96-127 通道上，而其逻辑设备ID号在同一台计计算机上按不同顺序插入会发生变化，即便在不同计算机上按相同顺序插入也可能会因主板制造商的不同定义而发生变化，所以您可能由此无法确定 0-127 的通道分别接入了什么信号。那么如何将各个设备在设备链中的物理位置固定下来呢？那么物理设备ID的使用帮您解决了这个问题。它是在卡上提供了一个拔码器DID，可以由用户为各个设备手动设置不同的物理ID号，当调用CreateDeviceEx函数时，只需要指定该参数的值与您在拔码器上设定的值一样即可，驱动程序会自动跟踪拔码器值与此相等的设备。它的取值范围通常在[0, 15]之间。

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。

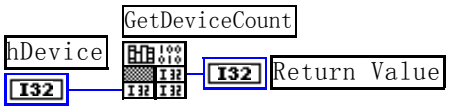
相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得本计算机系统中 PCI2600 设备的总数量**

函数原型：

Visual C++:

int GetDeviceCount (HANDLE hDevice)



功能：取得 PCI2600 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 PCI2600 的数量。

相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得该设备当前 ID**

函数原型：

Visual C++:

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

LabVIEW：

请参考相关演示程序。

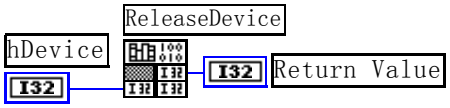
功能：取得 PCI2600 设备的数量。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：返回系统中当前设备相应的 ID 号。
相关函数： CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

- **用对话框控件列表计算机系统中所有 PCI2600 设备各种配置信息**

函数原型：
Visual C++:
BOOL ListDeviceDlg (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。
功能：列表系统中 PCI2600 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 PCI2600 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)
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、发送器传输控制函数

- **使能发送器设备**

函数原型：
Visual C++:
BOOL StartDeviceSend (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。

功能：使能发送器设备。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果发送器连接成功返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice　StartDeviceSend　ClrDeviceSendFifo　GetDevStatusSend　WriteDeviceData_Dma　WriteDeviceData_Pro　StopDeviceSend　ReleaseDevice

- **清发送器 FIFO**

函数原型：
Visual C++:
BOOL ClrDeviceSendFifo (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。

功能：清发送器 FIFO。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceSend ClrDeviceSendFifo
GetDevStatusSend WriteDeviceData_Dma WriteDeviceData_Pro
StopDeviceSend ReleaseDevice

◆ **获得发送器状态**
函数原型：
Visual C++:
BOOL GetDevStatusSend (HANDLE hDevice，
PPCI2600_STATUS_SEND pSendStatus)
LabVIEW:
请参考相关演示程序。

功能：获得发送器状态。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
pSendStatus 发送器状态
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceSend ClrDeviceSendFifo
GetDevStatusSend WriteDeviceData_Dma WriteDeviceData_Pro
StopDeviceSend ReleaseDevice

◆ **用查询方式写入数据**
函数原型：
Visual C++:
BOOL WriteDeviceData_Pro (HANDLE hDevice,
PULONG pDataBuffer,
ULONG nWriteSizeWords)
LabVIEW:
请参考相关演示程序。

功能：用查询方式写入数据。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
pDataBuffer 数据用户缓冲区
nWriteSizeWords 写入的点数（以字为单位）
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceSend ClrDeviceSendFifo
GetDevStatusSend WriteDeviceData_Dma WriteDeviceData_Pro
StopDeviceSend ReleaseDevice

◆ **用 DMA 方式写入数据**
函数原型：
Visual C++:
BOOL WriteDeviceData_Dma (HANDLE hDevice,
PULONG pDataBuffer,
ULONG nWriteSizeWords)
LabVIEW:
请参考相关演示程序。

功能：用 DMA 方式写入数据。

参数：hDevice设备对象句柄，它应由CreateDevice创建。
pDataBuffer 数据用户缓冲区
nWriteSizeWords 写入的点数（以字为单位）
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceSend ClrDeviceSendFifo GetDevStatusSend WriteDeviceData_Dma WriteDeviceData_Pro StopDeviceSend ReleaseDevice

- **禁止发送器设备**

函数原型：
Visual C++:
BOOL StopDeviceSend (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。

功能：禁止发送器设备。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceSend ClrDeviceSendFifo GetDevStatusSend WriteDeviceData_Dma WriteDeviceData_Pro StopDeviceSend ReleaseDevice

第四节、接收器传输控制函数

- **使能接收器设备**

函数原型：
Visual C++:
BOOL StartDeviceReceive (HANDLE hDevice)
LabVIEW：
请参考相关演示程序。

功能：使能接收器设备。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果发送器连接成功返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

- **清接收器 FIFO**

函数原型：
Visual C++:
BOOL ClrDeviceReceiveFifo (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。

功能：清接收器 FIFO。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

- **获得接收器状态**

函数原型：
Visual C++:
BOOL GetDevStatusReceive (HANDLE hDevice，

PPCI2600_STATUS_Receive pReceiveStatus)

LabVIEW:

请参考相关演示程序。

功能：获得接收器状态。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

pReceiveStatus 接收器状态

返回值：如果成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

♦ **用非空标志读取设备上的 AD 数据**

函数原型：

Visual C++:

```
BOOL ReadDevicePro_Npt (HANDLE hDevice,
                        PULONG pDataBuffer,
                        ULONG nReadSizeWords,
                        PLONG nRetSizeWords)
```

LabVIEW:

请参考相关演示程序。

功能：用非空标志读取设备上的 AD 数据。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

pDataBuffer 数据用户缓冲区

nReadSizeWords 读入的数据长度

nRetSizeWords 返回实际读取的数据长度

返回值：如果成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

♦ **FIFO 半满读 AD 数据**

函数原型：

Visual C++:

```
BOOL ReadDevicePro_Half (HANDLE hDevice,
                         PULONG pDataBuffer,
                         ULONG nReadSizeWords,
                         PLONG nRetSizeWords)
```

LabVIEW:

请参考相关演示程序。

功能：FIFO 半满读 AD 数据。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

pDataBuffer 数据用户缓冲区

nReadSizeWords 读入的数据长度

nRetSizeWords 返回实际读取的数据长度

返回值：如果成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

♦ **直接内存(DMA)方式函数**

函数原型：

Visual C++:

```
BOOL ReadReceiveDeviceDma (HANDLE hDevice,
                           PULONG pDataBuffer,
```

LONG nReadSizeWords,
PLONG nRetSizeWords)

LabVIEW:
请参考相关演示程序。

功能：直接内存(DMA)方式函数。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
pDataBuffer 数据用户缓冲区
nReadSizeWords 每次 DMA 时,用户从指定缓冲应读取的实际长度
nRetSizeWords 返回实际读取的数据长度
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

- **禁止接收器设备**

函数原型：
Visual C++:
BOOL StopDeviceReceive (HANDLE hDevice)
LabVIEW:
请参考相关演示程序。
功能：禁止接收器设备。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice StartDeviceReceive ClrDeviceReceiveFifo ReadDevicePro_Npt GetDevStatusReceive ReadDevicePro_Half ReadReceiveDeviceDma StopDeviceReceive ReleaseDevice

第五节、中断实现计数器控制函数

- **初始化设备上的 AD 对象**

函数原型：
Visual C++:
BOOL InitDeviceInt (HANDLE hDevice,
HANDLE hEventInt)
LabVIEW:
请参考相关演示程序。

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。且让设备上的AD部件以硬件中断的方式工作，其中断源信号由FIFO芯片半满管脚提供。但它并不启动AD采样，那么需要在此函数被成功调用之后，再调用StartDeviceIntAD函数即可启动AD采样。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
hEvent 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hEvent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。其具体实现方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》。
nFifoHalfLength 告诉设备对象，FIFO 存储器半满长度大小。该参数很关键，因为不仅决定了设备对象每次产生半满中断时应读入 AD 数据的点数，同时，它也决定了一级缓冲队列中每个元素对应的缓冲区大小。比如，nFifoHalfLength 等于 2048，则设备对象在系统空间中建立具有 64 个元素，且每个元素对应于 2048 个字长且物理连续的一级缓冲队列。但是该参数可以根据用户特殊需要，将其置成小于 FIFO 存储器实际的半满长度的值。比如用户要求在频率一定的情况下，提高 FIFO 半满中断事件的频率等，那么可以将此参数置成小于 FIFO

半满长度的值，但是绝不能大小半满长度。在工作期间，此队列的维护和管理完全由设备对象管理，与用户无关，用户只需要用 ReadDeviceIntAD 函数简单地读取 AD 数据，并注意检查其返回值即可。

pADPara设备对象参数结构指针， 它的各成员值决定了设备上的AD对象的各种状态及工作方式，如AD采样通道、采样频率等。请参考《硬件参数结构》章节。

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

相关函数： CreateDevice　InitDeviceInt　GetDeviceIntCount　ReleaseDeviceInt　ReleaseDevice

◆ **获得中断次数**

函数原型：

Visual C++:

```
BOOL GetDeviceIntCount (HANDLE hDevice
                        PULONG pIntSrc)
```

LabVIEW:

请参考相关演示程序。

功能：在InitDeviceIntAD被成功调用之后，调用此函数即可启动设备上的AD部件，让设备开始AD采样。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

pIntSrc　得到中断次数

返回值：若成功，则返回 TRUE，意味着 AD 被启动，否则返回 FALSE。

相关函数： CreateDevice　InitDeviceInt　GetDeviceIntCount　ReleaseDeviceInt　ReleaseDevice

◆ **释放设备上的 AD 部件**

函数原型：

Visual C++:

```
BOOL ReleaseDeviceInt (HANDLE hDevice)
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的AD部件，如果AD没有被StopDeviceIntAD函数停止，则此函数在释放AD部件之前先停止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。

相关函数： CreateDevice　InitDeviceInt　GetDeviceIntCount　ReleaseDeviceInt　ReleaseDevice

第四章 硬件参数结构

第一节、用于传输的实际硬件参数结构（PCI2600_STATUS_SEND）

Visual C++:

```
typedef struct _PCI2600_STATUS_SEND
{
    LONG bNotEmpty;             // 板载 FIFO 存储器的非空标志，=TRUE 非空， =FALSE 空
    LONG bHalf;                 // 板载 FIFO 存储器的半满标志，=TRUE 半满以上， =FALSE 半满以下
    LONG bDynamic_Overflow; // 板载 FIFO 存储器的动态溢出标志，= TRUE 已发生溢出，= FALSE 未发生溢出
    LONG bStatic_Overflow;  // 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
} PCI2600_STATUS_SEND, *PPCI2600_STATUS_SEND;

typedef struct _PCI2600_STATUS_RECEIVE
{
LONG bNotEmpty;             // 板载 FIFO 存储器的非空标志，=TRUE 非空， =FALSE 空
LONG bHalf;                 // 板载 FIFO 存储器的半满标志，=TRUE 半满以上， =FALSE 半满以下
LONG bDynamic_Overflow; // 板载 FIFO 存储器的动态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
LONG bStatic_Overflow;  // 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
LONG ulRemainCount;         // 板载 FIFO 存储器里剩余数据个数
} PCI2600_STATUS_RECEIVE, *PPCI2600_STATUS_RECEIVE;
```

第五章 上层用户函数接口应用实例

第一节、怎样使用 StartDeviceReceive 函数直接接收数据

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]**｜**[PCI2600 光纤通讯卡]**｜**[Microsoft Visual C++]**｜**[简易代码演示]**｜**[使能接收方式]**

第二节、怎样使用ReadDevicePro_Half函数直取得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]**｜**[PCI2600 光纤通讯卡]**｜**[Microsoft Visual C++]**｜**[简易代码演示]**｜**[AD 半满方式]**

第三节、怎样使用中断实现计数器控制函数

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]**｜**[PCI2600 光纤通讯卡]**｜**[Microsoft Visual C++]**｜**[简易代码演示]**｜**[AD 中断方式]**

第四节、怎样使用 ReadDevicePro_Npt 读取设备上的 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]** | **[PCI2600 光纤通讯卡]** | **[Microsoft Visual C++]** | **[简易代码演示]** | **[DA 输出**

第六章 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ReadDeviceProAD_X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ReadDeviceProAD_NotEmpty(或者ReadDeviceProAD_Half)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InitDeviceProAD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。

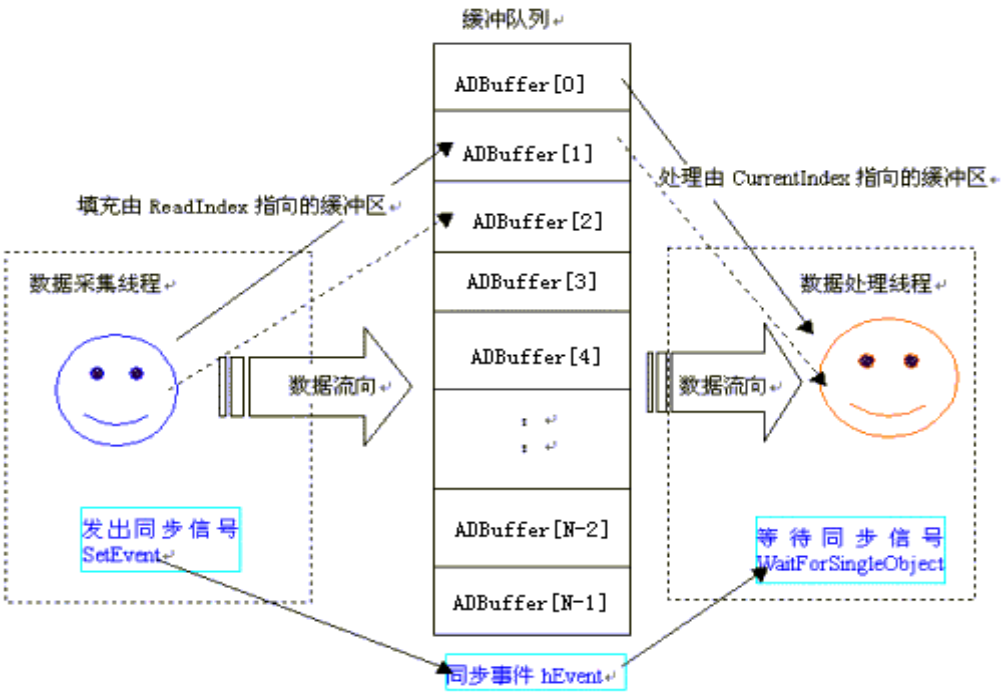


图 7.1

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 ReadDevicePro_Npt 函数读取设备上的 AD 数据（它使用 FIFO 的非空标志）

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]**｜**[PCI2600 光纤通讯卡]**｜**[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
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Npt (PVOID pThreadPara)   // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)   // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD()          // 终止采集函数
```

二、使用ReadDevicePro_Half函数读取设备上的AD数据（它使用FIFO的半满标志）

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]**｜**[PCI2600 光纤通讯卡]**｜**[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
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Half (PVOID pThreadPara)   // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)   // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
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]｜[PCI2600 光纤通讯卡]｜[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_Int()        // 采集线程函数
BOOL StopDeviceAD_Int()          // 采集线程的终止函数
UINT DrawWindowProc ()                   // 绘制数据线程
void CADDoc:: OnStopDeviceAD ()         // 终止采集函数
```

第七章 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“PCI2600_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定 PCI 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 ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	用于线程同步或中断
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	适用于所有设备
WriteFile	请求文件对象写用户数据到磁盘文件	适用于所有设备
ReadFile	请求文件对象读数据到用户空间	适用于所有设备
SetFileOffset	设置文件指针偏移	适用于所有设备
GetFileLength	取得文件长度	适用于所有设备
ReleaseFile	释放已有的文件对象	适用于所有设备

GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备
⑤ 辅助函数		
GetLastErrorEx	从错误信息库中获得指定函数的最后一次错误信息	
RemoveLastErrorEx	从错误信息库中移除指定函数的最后一次错误信息	

第二节、PCI 内存映射寄存器操作函数原型说明

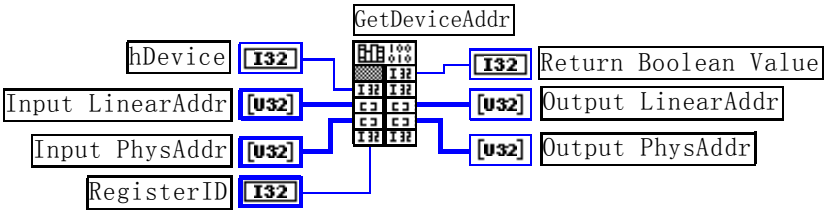
♦ 取得指定内存映射寄存器的线性地址和物理地址

函数原型：

Visual C++:

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

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。本设备的寄存器组 ID 定义如下：

常量名	常量值	功能定义
PCI2600_REG_MEM_PLXCHIP	0x0000	0 号寄存器对应 PLX 芯片所使用的内存模式基地址(使用 LinearAddr)
PCI2600_REG_IO_PLXCHIP	0x0001	1 号寄存器对应 PLX 芯片所使用的 IO 模式基地址(使用 PhysAddr)
PCI2600_REG_IO_CPLD	0x0002	2 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)
PCI2600_REG_IO_ADFIFO	0x0003	3 号寄存器对应板上 AD FIFO 缓冲区所使用的 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 pulPCIBar[6])
```

LabVIEW:

请参考相关演示程序。

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pulPCIBar[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 ReleaseDevice

- **获取设备固件及程序版本**

函数原型：

Visual C++:

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

LabVIEW:

请参见相关演示程序。

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 GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

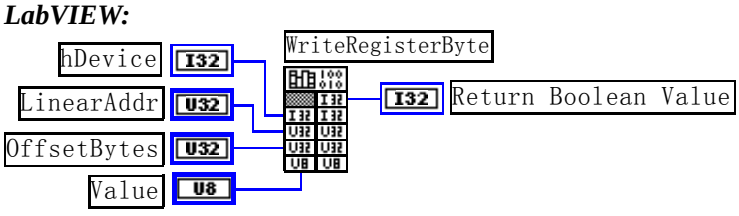
- **以单字节（即 8 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

Value 输出8位整数。

返回值：若成功，返回TRUE，否则返回FALSE。

相关函数： CreateDevice 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
    :
```

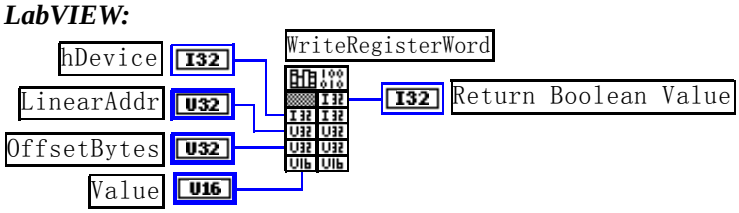
- **以双字节（即16位）方式写PCI内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以双字节（即16位）方式写PCI内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

Value 输出16位整型值。

返回值：无。

相关函数： CreateDevice 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
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

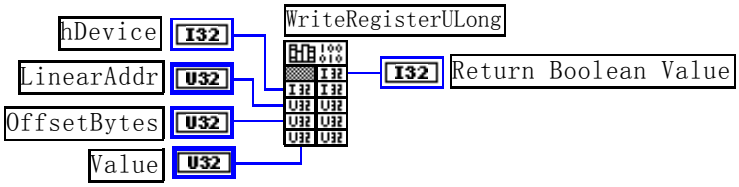
◆ **以四字节（即 32 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　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
    :
```

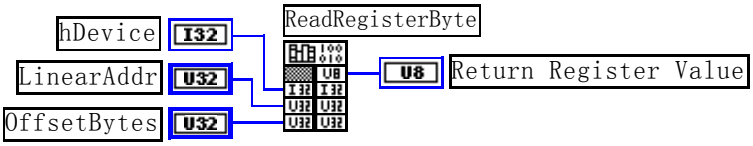
◆ **以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

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

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

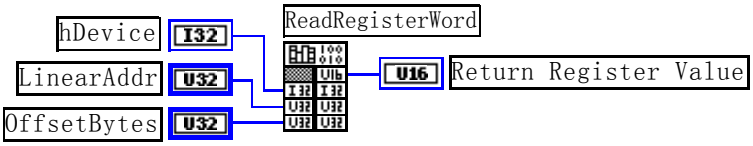
◆ **以双字节（即 16 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

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

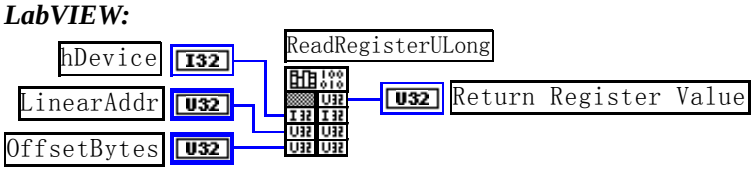
◆ **以四字节（即 32 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes)
```

LabVIEW:



功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

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

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
```

ReleaseDevice(hDevice)

:

第三节、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。

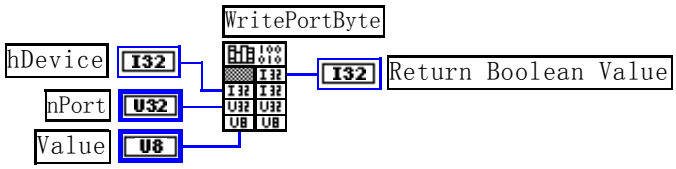
- **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    UINT nPort,
                    BYTE Value)
```

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 WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

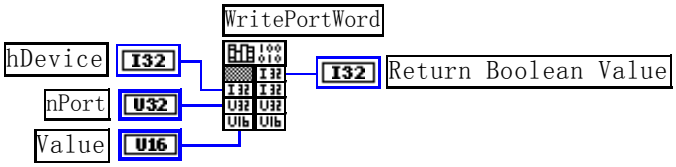
- **以双字(16Bit)方式写 I/O 端口**

函数原型：

Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    UINT nPort,
                    WORD Value)
```

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 WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

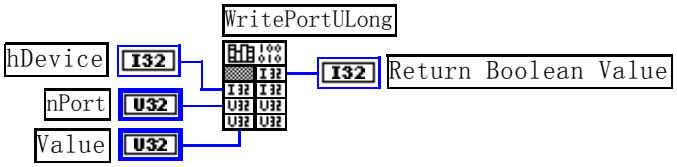
- **以四字节(32Bit)方式写 I/O 端口**

函数原型：

Visual C++:

```
BOOL WritePortULong(HANDLE hDevice,
                    UINT nPort,
                    ULONG Value)
```

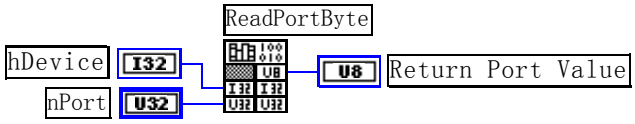
LabVIEW:



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 WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

◆ **以单字节(8Bit)方式读 I/O 端口**

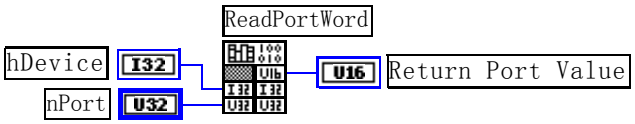
函数原型：
Visual C++:
BYTE ReadPortByte(HANDLE hDevice,
UINT nPort)
LabVIEW:



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 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,
UINT nPort)
LabVIEW:



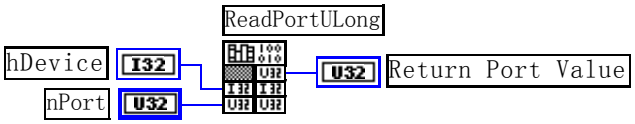
功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

◆ **以四字节(32Bit)方式读 I/O 端口**

函数原型：
Visual C++:
ULONG ReadPortULong (HANDLE hDevice,

UINT nPort)

LabVIEW:

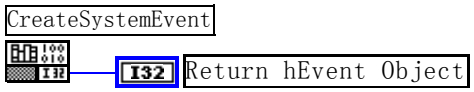


功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort　设备的 I/O 端口号。
返回值：返回由 nPort 指定端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

第四节、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)
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)
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。

第五节、文件对象操作函数原型说明

- **创建文件对象**

函数原型：
Visual C++:
HANDLE CreateFileObject (HANDLE hDevice,
LPCTSTR NewFileName,
int Mode)
LabVIEW:
请参见相关演示程序。

功能：初始化设备文件对象，以期待 WriteFile 请求准备文件对象进行文件操作。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
szNewFileName 新文件名。

Mode 文件操作方式，所用的文件操作方式控制字定义如下
(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
PCI2600_modeRead	0x0000	只读文件方式
PCI2600_modeWrite	0x0001	只写文件方式
PCI2600_modeReadWrite	0x0002	既读又写文件方式
PCI2600_modeCreate	0x1000	如果文件不存在可以创建该文件， 如果存在，则重建此文件，且清 0
PCI2600_typeText	0x4000	以文本方式操作文件

返回值：若成功，则返回文件对象句柄。
相关函数： CreateDevice CreateFileObject WriteFile ReadFile ReleaseFile ReleaseDevice

◆ **通过设备对象，往指定磁盘上写入用户空间的采样数据**

函数原型：
Visual C++:

```
BOOL WriteFile(HANDLE hFileObject,
               PVOID pDataBuffer,
               ULONG nWriteSizeBytes)
```

LabVIEW:
详见相关演示程序。

功能：通过向设备对象发送“写磁盘消息”，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由CreateFileObject函数中的strFileName指定。

参数：
hFileObject 设备对象句柄，它应由CreateFileObject创建。
pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。
nWriteSizeBytes 告诉设备对象往磁盘上一次写入数据的长度(以字节为单位)。
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： CreateFileObject WriteFile ReadFile ReleaseFile

◆ **通过设备对象,从指定磁盘文件中读采样数据**

函数原型：
Visual C++:

```
BOOL ReadFile ( HANDLE hFileObject,
                PVOID pDataBuffer,
                ULONG OffsetBytes,
                ULONG nReadSizeBytes)
```

LabVIEW:
详见相关演示程序。

功能：将磁盘数据从指定文件中读入用户内存空间中，其访问方式可由用户在创建文件对象时指定。
参数：
hFileObject 设备对象句柄，它应由CreateFileObject创建。
pDataBuffer 用于接受文件数据的用户缓冲区指针，可以是用户分配的数组空间。
OffsetBytes 指定从文件开始端所偏移的读位置。
nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。
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： CreateFileObject WriteFile ReadFile ReleaseFile

◆ **设置文件偏移位置**

函数原型：

Visual C++：

BOOL SetFileOffset (HANDLE hFileObject,
ULONG nOffsetBytes)

LabVIEW:

请参考相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由CreateFileObject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： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：

Visual C++:

ULONG GetFileLength (HANDLE hFileObject)

LabVIEW:

详见相关演示程序。

功能：取得文件长度。

参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。

返回值：若成功，则返回>1，否则返回 0，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **释放设备文件对象**

函数原型：

Visual C++:

BOOL ReleaseFile(HANDLE hFileObject)

LabVIEW:

详见相关演示程序。

功能：释放设备文件对象。

参数：hFileObject 设备对象句柄，它应由CreateFileObject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： CreateFileObject WriteFile ReadFile ReleaseFile

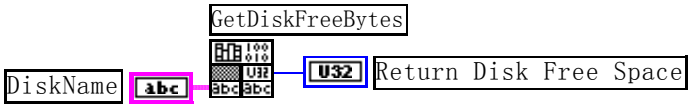
- **取得指定磁盘的可用空间**

函数原型：

Visual C++:

ULONGLONG GetDiskFreeBytes(LPCTSTR DiskName)

LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数： DiskName 需要访问的盘符，若为 C 盘为"C:\\"， D 盘为"D:\\"，以此类推。

返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用 GetLastErrorEx 捕获错误码。注意使用 64 位整型变量。

第六节、辅助函数

- **从错误信息库中获得指定函数的最后一次错误信息**

函数原型：
Visual C++:
BOOL GetLastErrorEx (LPCTSTR strFuncName,
LPTSTR strErrorMsg)
LabVIEW:
详见相关演示程序。

功能：从错误信息库中获得指定函数的最后一次错误信息。
参数：strFuncName 出错的函数名，注意大小写。
strErrorMsg 返回的错误信息
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： GetLastErrorEx RemoveLastErrorEx

- **从错误信息库中移除指定函数的最后一次错误信息**

函数原型：
Visual C++:
BOOL RemoveLastErrorEx (LPCTSTR strFuncName)
LabVIEW:
详见相关演示程序。

功能：从错误信息库中移除指定函数的最后一次错误信息。
参数：strFuncName 出错的函数名，注意大小写。
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： GetLastErrorEx RemoveLastErrorEx