PCI8603 数据采集卡

WIN2000/XP 驱动程序使用说明书

北京阿尔泰科技发展有限公司
产品研发部修订

目 录

目 录 1
第一章 版权信息与命名约定 2
第一节、版权信息 2
第二节、命名约定 2
第二章 使用纲要 2
第一节、使用上层用户函数，高效、简单 2
第二节、如何管理 PCI 设备 2
第三节、如何用非空查询方式取得 AD 数据 2
第四节、如何用半满查询方式取得 AD 数据 3
第四节、如何用 DMA 方式取得 AD 数据 3
第五节、如何实现开关量的简便操作 6
第六节、哪些函数对您不是必须的 7
第三章 PCI 即插即用设备操作函数接口介绍 7
第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI8603_”） 8
第二节、设备对象管理函数原型说明 10
第三节、AD 程序查询方式采样操作函数原型说明 13
第四节、AD 硬件参数保存与读取函数原型说明 19
第五节、DA 数据采样操作函数原型说明 20
第六节、DA 硬件参数系统保存与读取函数原型说明 28
第七节、DIO 数字量输入输出开关量操作函数原型说明 29
第四章 硬件参数结构 31
第一节、AD 硬件参数结构（PCI8603_PARA_AD） 31
第二节、AD 状态参数结构（PCI8603_STATUS_AD） 34
第三节、DMA 状态参数结构（PCI8603_STATUS_DMA） 35
第四节、DA 各段信息参数结构（PCI8603_SEGMENT_INFO） 36
第五节、DA 硬件参数结构（PCI8603_PARA_DA） 36
第六节、DA 状态参数结构（PCI8603_STATUS_DA） 38
第五章 数据格式转换与排列规则 39
第一节、AD 原码 LSB 数据转换成电压值的换算方法 39
第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则 40
第三节、AD 测试应用程序创建并形成的数据文件格式 40
第四节、DA 的电压值如何转换成输出到 DA 转换器的 LSB 原码数据？ 41
第五节、关于 DA 数据 DABuffer 缓冲区中的数据排放规则 41
第六章 上层用户函数接口应用实例 42
第一节、怎样使用 ReadDeviceProAD_Npt 函数直接取得 AD 数据 42
第二节、怎样使用 ReadDeviceProAD_Half 函数直接取得 AD 数据 42
第三节、怎样使用 ReadDeviceDmaAD 函数直接取得 AD 数据 42
第四节、怎样使用 WriteDeviceBulkDA 函数取得 DA 数据 42
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 42
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 43
第七章 高速大容量、连续不间断数据采集及存盘技术详解 43
第一节、使用程序查询方式实现该功能 44
第二节、使用 DMA 方式实现该功能 45
第八章 共用函数介绍 45
第一节、公用接口函数总列表（每个函数省略了前缀“PCI8603_”） 45
第二节、PCI 内存映射寄存器操作函数原型说明 46
第三节、IO 端口读写函数原型说明 53
第四节、线程操作函数原型说明 56
第五节、文件对象操作函数原型说明 57
第六节、各种参数保存和读取函数原型说明 60
第六节、其他函数原型说明 62

第一章 版权信息与命名约定

第一节、版权信息

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

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 PCI8603_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InitDeviceAD、ReadDeviceProAD_Npt、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InitDeviceAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD_Npt、ReadDeviceProAD_Half (或ReadDeviceDmaAD)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InitDeviceAD函数初始化AD部件，关于采样通道、频率等参数

的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用StartDeviceAD即可启动AD部件，开始AD采样，然后便可用ReadDeviceProAD_Npt反复读取AD数据以实现连续不间断采样。当您需要暂停设备时，执行StopDeviceAD，当您需要关闭AD设备时，ReleaseDeviceAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceProAD_Npt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InitDeviceAD函数初始化AD部件，关于采样通道、频率等参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用StartDeviceAD即可启动AD部件，开始AD采样，接着调用GetDevStatusAD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StopDeviceAD，当您需要关闭AD设备时，ReleaseDeviceAD便可帮您实现（但设备对象hDevice依然存在）。（注：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。

第四节、如何用 DMA 方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceAD函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hDmaEvent赋给ReadDeviceDmaAD的相应参数，它将作为Dma事件的变量。然后用StartDeviceAD即可启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hDmaEvent事件的发生，当当前缓冲段没有被DMA完成时，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GetDevStatusAD来确定哪一段缓冲是新的数据，即刻处理该数据，至到所有的缓冲段变为旧数据段。然后再回到WaitForSingleObject，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StopDeviceAD，当您需要关闭AD设备时，ReleaseDeviceAD便可帮您实现（但设备对象hDevice依然存在）。具体执行流程请看图 2.1.3。

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。

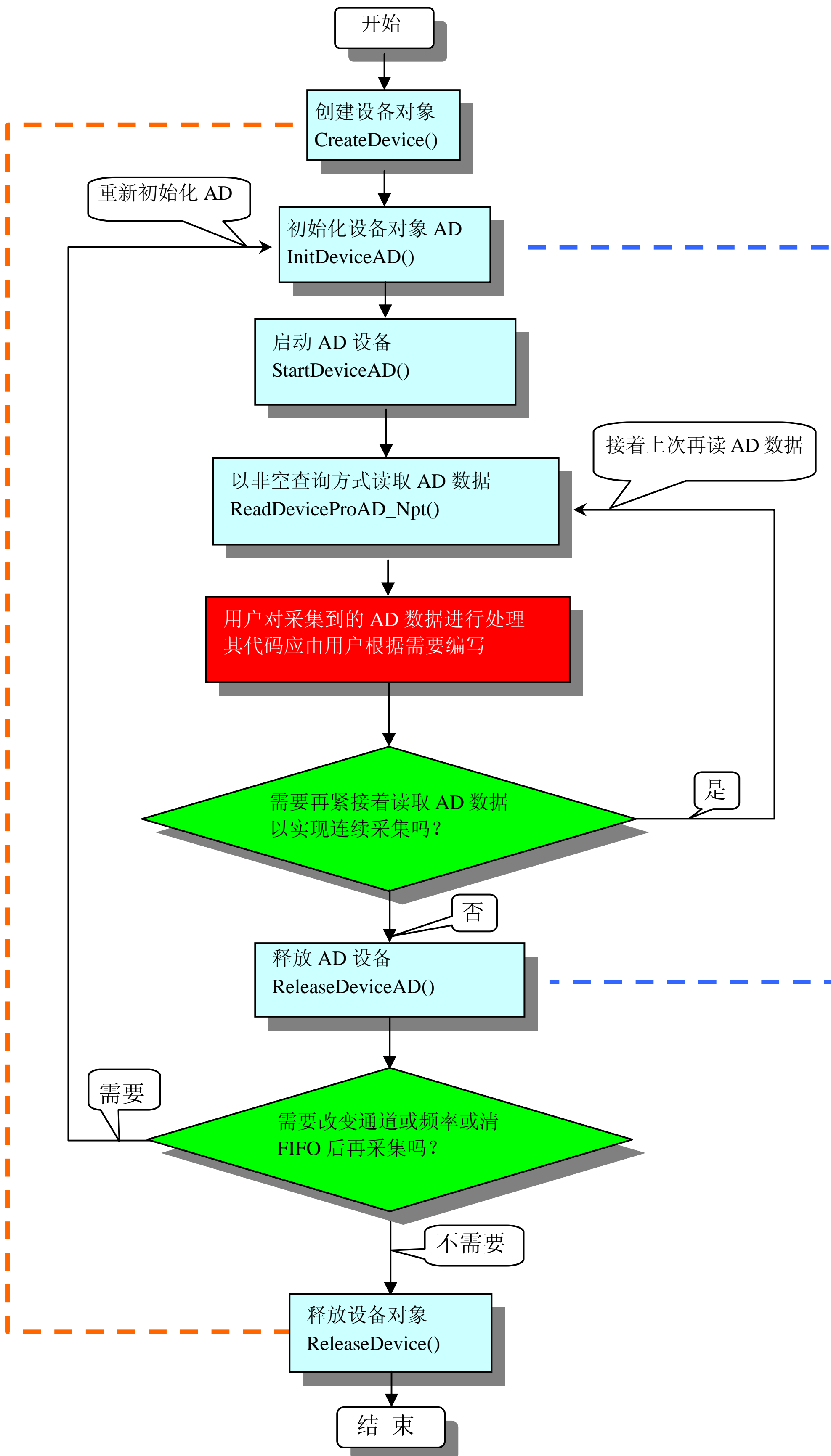


图 2.1.1 非空查询方式 AD 采集过程

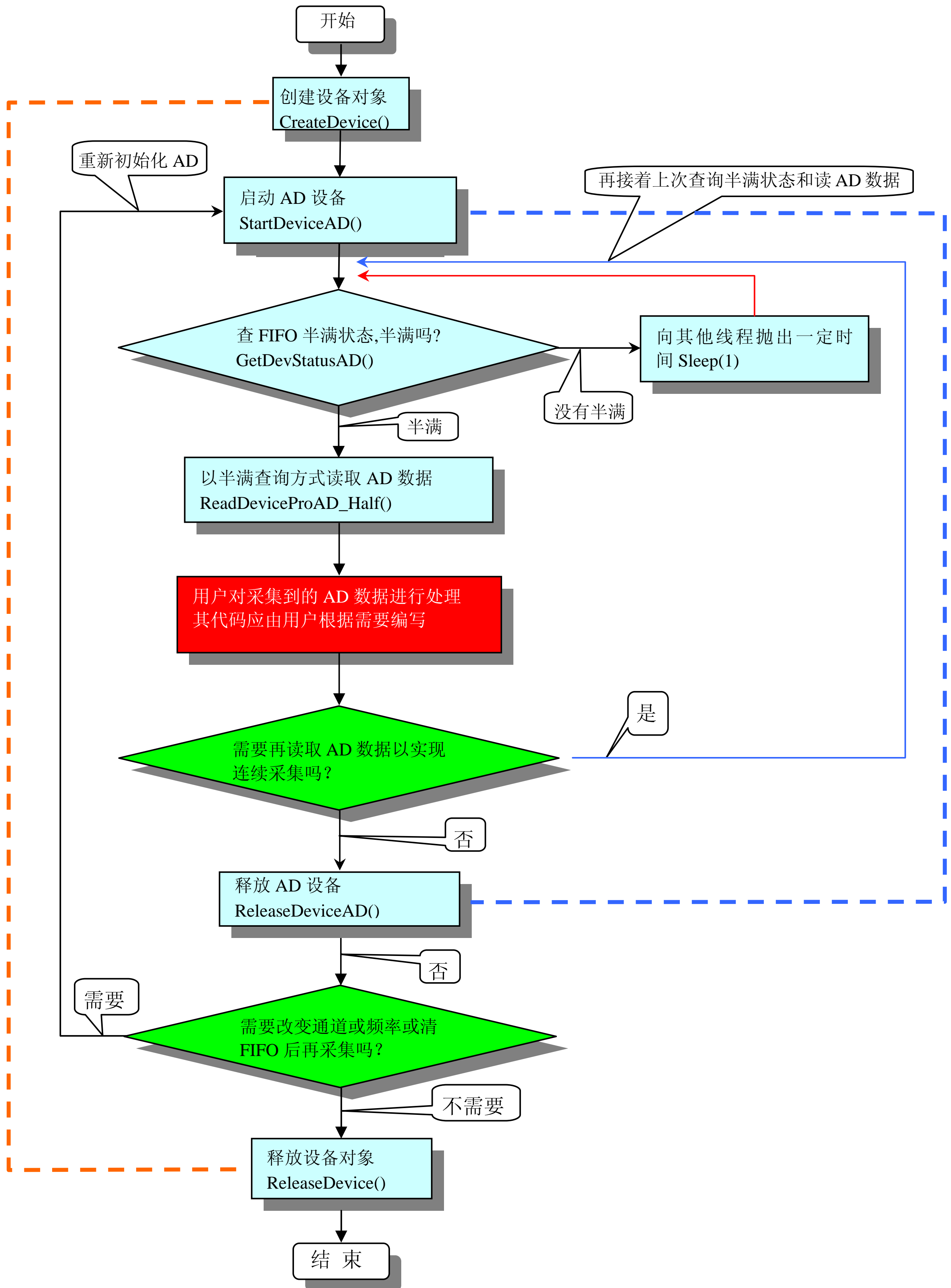


图 2.1.2 半满查询方式 AD 采集过程

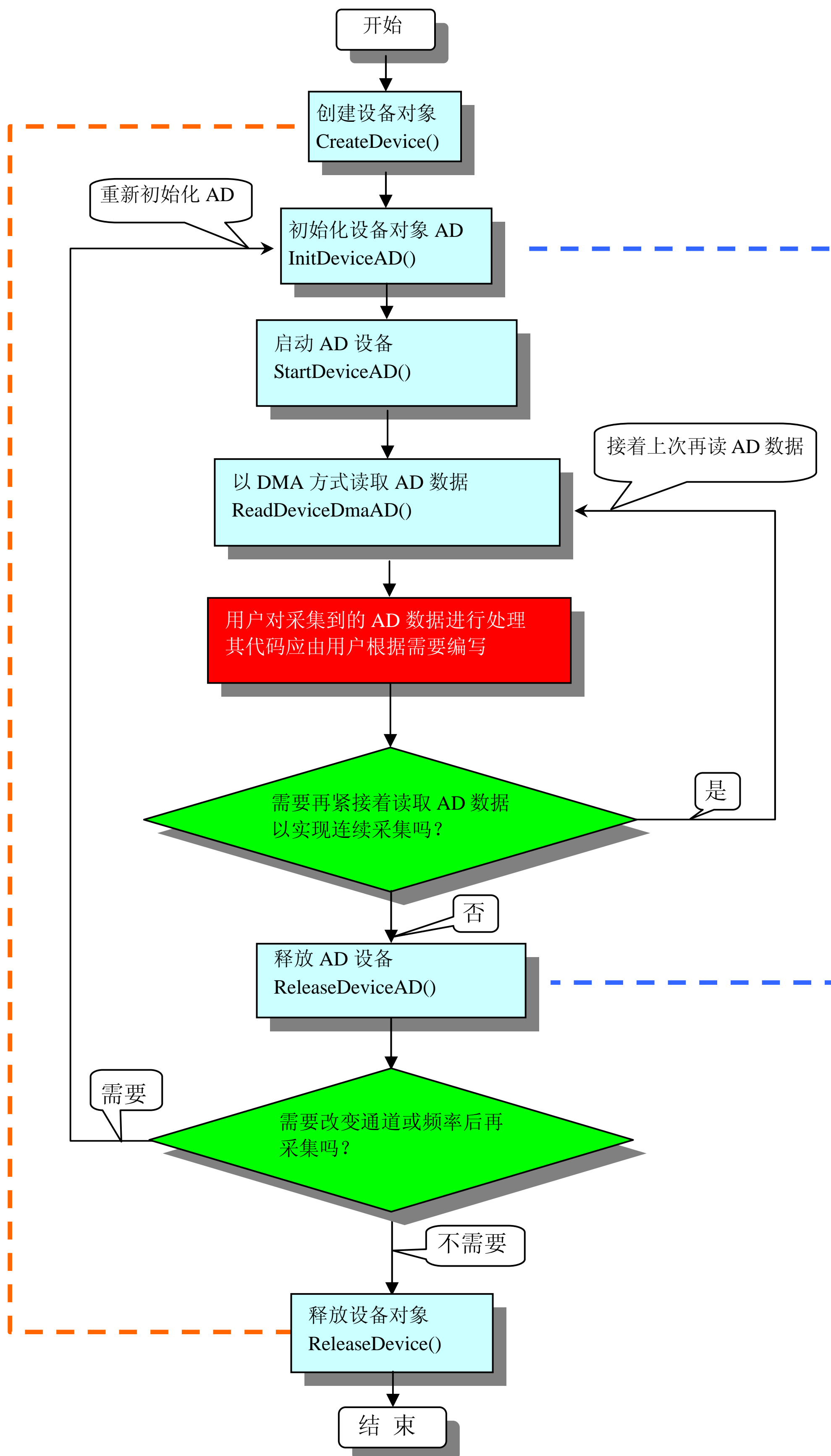


图 2.1.3 DMA 方式 AD 采集过程

第五节、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[8]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[8]中的相应元素决定。

第六节、哪些函数对您不是必须的

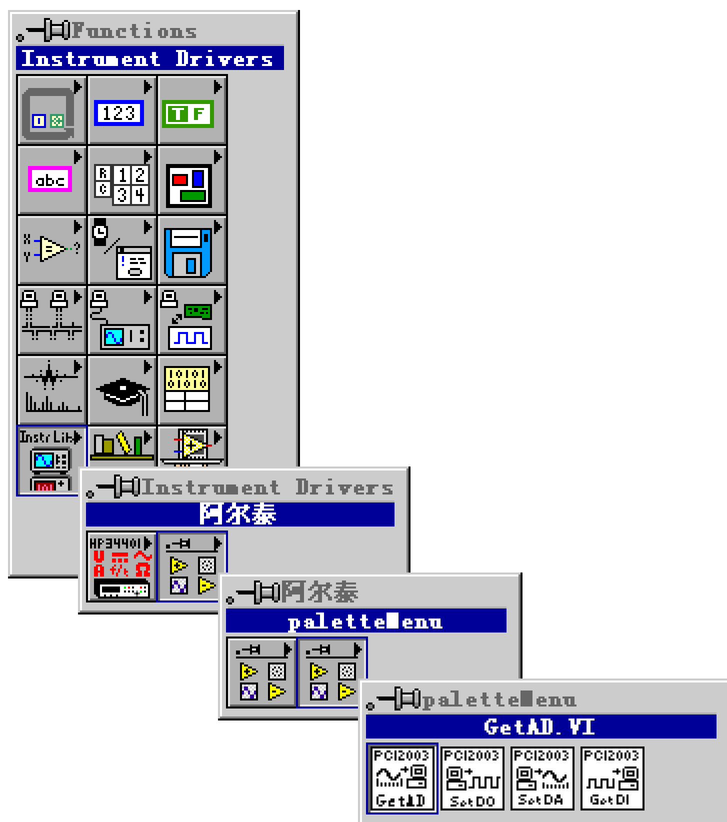
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr，WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， 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InitDeviceAD）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用ReadDeviceProAD_Npt 、ReadDeviceProAD_Half（或ReadDeviceDmaAD）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“PCI8603_”）

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
CreateDeviceEx	创建 PCI 设备对象(用设备物理号)	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
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	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**		
GetDevTrigPosAD	取得触发点位置	上层用户
InitDeviceAD	初始化 AD 部件准备传输	上层用户
StartDeviceAD	启动 AD 设备，开始转换	上层用户
ReadDeviceProAD_Npt	连续读取当前 PCI 设备上的 AD 数据	上层用户
GetDevStatusAD	取得当前 PCI 设备 FIFO 半满状态	
ReadDeviceProAD_Half	连续批量读取 PCI 设备上的 AD 数据	
ReadDeviceDmaAD	DMA 方式读取当前 PCI 设备上的 AD 数据	上层用户
StopDeviceAD	暂停 AD 设备	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
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
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
④ **DA 模拟量输出操作函数**		
SetDevTrigLevelDA	设置 DA 的触发电平	上层用户
SetDevFrequencyDA	可动态改变 DA 采样频率	上层用户
ReadSegmentInfo	读段信息	

InitDeviceDA	初始化 PCI 设备上的 DA 部件准备传输	上层用户
WriteDeviceOneDA	DA 单点输出	
WriteDeviceBulkDA	批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中	
ReadDeviceBulkDA	以批量方式将板载 RAM 中的 DA 数据回读至主机的用户缓冲区	
EnableDeviceDA	启动 DA 设备，开始转换	上层用户
SetDeviceTrigDA	软件产生触发事件	
GetDevStatusDA	取得当前 DA 状态	上层用户
DisableDeviceDA	暂停 DA 设备	上层用户
ReleaseDeviceDA	释放 DA 设备	上层用户
⑤ DA 硬件参数系统保存、读取函数		
LoadParaDA	从 Windows 系统中读入硬件参数	上层用户
SaveParaDA	往 Windows 系统写入设备硬件参数	上层用户
ResetParaDA	将硬件参数结构体值复位为出厂默认值	上层用户
⑥ 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
RetDeviceDO	回读输出开关量状态	上层用户

使用需知：

Visual C++ & C++Builder：

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\PCI8603\INCLUDE\PCI8603.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 PCI8603.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 “PCI8603.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。

C++ Builder:

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头文件(PCI8603.H)写进您的源程序头部。如：#include “\Art\PCI8603\Include\PCI8603.h”，然后再将 PCI8603.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”（Add to Project）命令，在弹出的对话框中分别选择文件类型：Library file (*.lib) ， 即可选择 PCI8603.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。

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 PCI8603.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。

Delphi:

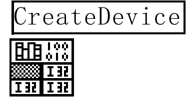
要使用如下函数一个关键的问题是首先必须将我们提供的单元模块文件（*.Pas）加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单，执行其中的"Project Manager"命令，在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PCI8603.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCI8603”。如：

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
PCI8603; // 注意： 在此加入驱动程序接口单元 PCI8603

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 PCI8603.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++ & C++Builder :

HANDLE CreateDevice (int DeviceLgcID = 0)

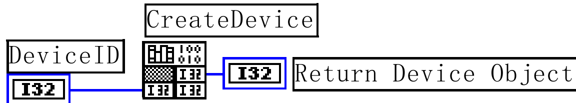
Visual Basic :

Declare Function CreateDevice Lib "PCI8603" (ByVal DeviceLgcID As Integer = 0) As Long

Delphi :

```
Function CreateDevice(DeviceLgcID : Integer = 0) : Integer;
                        StdCall; External 'PCI8603' Name ' CreateDevice ';
```

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：

DeviceLgcID 逻辑设备ID(Logic Device Identifier)标识号。当向同一个Windows系统中加入若干相同类型的PCI设备时，我们的驱动程序将以该设备的“基本名称”与DeviceLgcID标识值为后缀的标识符来确认和管理该设备。比如若用户往Windows系统中加入第一个PCI8603 模板时，驱动程序逻辑号为“0”来确认和管理第一个设备，若用户接着再添加第二个PCI8603 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个PCI设备时，DeviceLgcID应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由BIOS和操作系统加载设备时，依据主板总线编号等信息进行这个设备ID号分配，说得简单点，就是加载设备的顺序编号，编号的递增顺序为 0、1、2、3……。所以用户无法直接固定某一个设备的在设备列表中的物理位置，若想固定，则必须须使用物理ID号，调用CreateDeviceEx函数实现。

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++ & C++Builder 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
    :
```

♦ 创建设备对象函数（物理号）

函数原型：

Visual C++ & C++Builder：

HANDLE CreateDeviceEx(int DevicePhysID = 0)

Visual Basic：

Declare Function CreateDeviceEx Lib "PCI8603" (Optional ByVal DevicePhysID As Integer = 0) As Long

Delphi：

Function CreateDeviceEx(DevicePhysID : Integer = 0) : Integer;
StdCall; External 'PCI8603' Name ' CreateDeviceEx ';

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 PCI8603 设备的总数量**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

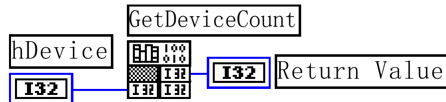
Visual Basic:

Declare Function GetDeviceCount Lib "PCI8603" (ByVal hDevice As Long) As Integer

Delphi:

Function GetDeviceCount (hDevice : Integer) : Integer;
StdCall; External 'PCI8603' Name ' GetDeviceCount ';

LabVIEW:



功能：取得 PCI8603 设备的数量。

参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

返回值：返回系统中 PCI8603 的数量。

相关函数： CreateDevice CreateDeviceEx GetDeviceCount
GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **取得该设备当前逻辑 ID 和物理 ID**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

Declare Function GetDeviceCurrentID Lib "PCI8603" (ByVal hDevice As Long,_
ByRef DeviceLgcID As Long,_
ByRef DevicePhysIDAs Long) As Boolean

Delphi:

Function GetDeviceCurrentID (hDevice : Integer;
DeviceLgcID : Pointer;
DevicePhysID : Pointer) : Boolean;
StdCall; External 'PCI8603' Name ' GetDeviceCurrentID ';

LabVIEW:

请参考相关演示程序。

功能：取得指定设备 ID 号。

参数：

hDevice 设备对象句柄，它指向要取得逻辑和物理号的设备，它应由CreateDevice或CreateDeviceEx创建。

DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。

DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拔码器 DID 决定。

返回值：返回设备 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 PCI8603 设备各种配置信息**

函数原型：

Visual C++ & C++Builder:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib "PCI8603" (ByVal hDevice As Long) As Boolean

Delphi:

Function ListDeviceDlg (hDevice : Integer) : Boolean;
StdCall; External 'PCI8603' Name ' ListDeviceDlg ';

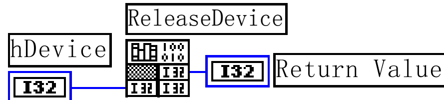
LabVIEW:

请参考相关演示程序。

功能：列表系统中 PCI8603 的硬件配置信息。
参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
返回值：若成功，则弹出对话框控件列表所有 PCI8603 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++ & C++Builder:
BOOL ReleaseDevice(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDevice Lib "PCI8603" (ByVal hDevice As Long) As Boolean
Delphi：
Function ReleaseDevice(hDevice : Integer) : Boolean;
　　　　StdCall; External 'PCI8603' Name ' ReleaseDevice ';
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或CreateDeviceEx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

◆ 取得触发位置

函数原型：
Visual C++ & C++Builder:
BOOL GetDevTriggerPos(HANDLE hDevice,
　　　　PULONG nTriggerPos)
Visual Basic:
Declare Function GetDevTriggerPos Lib "PCI8603" (ByVal hDevice As Long,_
　　　　ByRef nTriggerPos As Long) As Boolean
Delphi:
Function GetDevTriggerPos (hDevice : Integer;
　　　　nTriggerPos : Pointer) : Boolean;
　　　　StdCall; External 'PCI8603' Name ' GetDevTriggerPos ';
LabVIEW
请参考相关演示程序。

功能：在 AD 内触发采样过程中，在外触发处加一个触发信号，此函数可取得触发位置。
参数：
hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
nTriggerPos 触发位置值。
返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。
相关函数：

CreateDevice	InitDeviceAD	StartDeviceAD
ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
ReleaseDevice		

◆ 初始化 AD 设备（ **Initlize device AD for program mode**）

函数原型：

Visual C++ & C++Builder:

```
BOOL InitDeviceAD( HANDLE hDevice,
                   PPCI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceAD Lib "PCI8603" (ByVal hDevice As Long, _
                              ByRef pADPara As PCI8603_PARA_AD) As Boolean
```

Delphi:

```
Function InitDeviceAD(hDevice : Integer;
                      pADPara : PPCI8603_PARA_AD) : Boolean;
                      StdCall; External 'PCI8603' Name ' InitDeviceAD ';
```

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，为设备的操作就绪做有关准备工作，如预置AD采集通道、采样频率等。但它并不启动AD设备，若要启动AD设备，须在调用此函数之后再调用StartDeviceAD。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pADPara 设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如AD采样通道、采样频率等。关于PCI8603_PARA_AD具体定义请参考PCI8603.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。

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	InitDeviceAD	StartDeviceAD
	ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
	ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
	ReleaseDevice		

◆ **启动 AD 设备(Start device AD for program mode)**

函数原型：

Visual C++ & C++Builder:

```
BOOL StartDeviceAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceAD Lib "PCI8603" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function StartDeviceAD (hDevice : Integer ): Boolean;
                        StdCall; External 'PCI8603' Name ' StartDeviceAD ';
```

LabVIEW:

请参考相关演示程序。

功能：启动AD设备，它必须在调用InitDeviceAD后才能调用此函数。该函数除了启动AD设备开始转换以外，不改变设备的其他任何状态。

参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

返回值：如果调用成功，则返回TRUE，且AD立刻开始转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：	CreateDevice	InitDeviceAD	StartDeviceAD
	ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
	ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
	ReleaseDevice		

◆ **读取 PCI 设备上的 AD 数据**

① 使用 FIFO 的非空标志读取 AD 数据

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceProAD_Npt( HANDLE hDevice,
                          ULONG ADBuffer[],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Npt Lib "PCI8603" (ByVal hDevice As Long, _
                                                     ByVal ADBuffer As Long,_
                                                     ByVal nReadSizeWords As Long,_
                                                     ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Npt(hDevice : Integer;
                             ADBuffer: LongWord;
                             nReadSizeWords : LongInt;
                             nRetSizeWords : Pointer) : Boolean;
                             StdCall; External 'PCI8603' Name ' ReadDeviceProAD_Npt ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数读取设备上的AD数据。此函数使用FIFO的非空标志进行读取AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

ADBuffer接受AD数据的用户缓冲区，它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ReadDeviceProAD_Npt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。此参数值只与ADBuffer[]指定的缓冲区大小有效，而与FIFO存储器大小无效。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值：其返回值表示所成功读取的数据点数(字)，也表示当前读操作在ADBuffer缓冲区中的有效数据量。通常情况下其返回值应与ReadSizeWords参数指定量的数据长度(字)相等，除非用户在这个读操作以外的其他线程中执行了ReleaseDeviceProAD函数中断了读操作，否则设备可能有问题。对于返回值不等于nReadSizeWords参数值的，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

注释：此函数也可用于单点读取和几个点的读取，只需要将nReadSizeWords设置成 1 或相应值即可。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。

相关函数：	CreateDevice	InitDeviceAD	StartDeviceAD
	ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
	ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
	ReleaseDevice		

② 使用 FIFO 的半满标志读取 AD 数据

- **取得 FIFO 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDevStatusAD ( HANDLE hDevice,
                      PPCI8603_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "PCI8603" (ByVal hDevice As Long,_
                                    ByRef pADStatus As PCI8603_STATUS_AD)As Boolean
```

Delphi:

```
Function GetDevStatusAD (hDevice : Integer;
                         PADStatus : PPCI8603_STATUS_AD) : Boolean;
                         StdCall; External 'PCI8603' Name ' GetDevStatusAD ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数查询FIFO存储器的状态（半满标志、非空标志、溢出标志）。我们通常用半满标志去同步半满读操作。当半满标志有效时，再紧接着用ReadDeviceProAD_Half读取FIFO中的半满有效AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pADStatus 获得AD的各种当前状态。它属于结构体，具体定义请参考《AD状态参数结构（PCI8603_STATUS_AD）》章节。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。若用户选择半满查询方式读取AD数据，则当GetDevStatusDA函数取得的bHalf等于TRUE，应立即调用ReadDeviceProAD_Half读取FIFO中的半满数据。否则用户应继续循环轮询FIFO半满状态，直到有效为止。注意在循环轮询期间，可以用Sleep函数抛出一定时间给其他应用程序(包括本应用程序的主程序和其他子线程)，以提高系统的整体数据处理效率。

其使用方法请参考本文档的《高速大容量、连续不间断数据采集及存盘技术详解》章节。

相关函数：

CreateDevice	InitDeviceAD	StartDeviceAD
ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
ReleaseDevice		

◆ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceProAD_Half( HANDLE hDevice,
                           ULONG ADBuffer[],
                           LONG nReadSizeWords,
                           PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "PCI8603" ( ByVal hDevice As Long, _
                                                      ByVal ADBuffer As Long,_
                                                      ByVal nReadSizeWords As Long ,_
                                                      ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Half(hDevice : Integer;
                              ADBuffer : LongWord;
                              nReadSizeWords : LongInt;
                              nRetSizeWords : Pointer) : Boolean;
                              StdCall; External 'PCI8603' Name ' ReadDeviceProAD_Half ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用GetDevStatusDA后取得的FIFO状态bHalf等于TRUE(即半满状态有效)时，应立即用此函数读取设备上FIFO中的半满AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

ADBuffer 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ReadDeviceProAD_Half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，而且应等于FIFO总容量的二分之一(如果用户有特殊需要可以小于FIFO的二分之一长)。比如设备上配置了 1K FIFO，即 1024 字，那么这个参数应指定为 512 或小于 512。

nRetSizeWords 返回实际读取的点数(或字数)。

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。

其使用方法请参考本部分《高速大容量、连续不间断数据采集及存盘技术详解》。

相关函数：

CreateDevice	InitDeviceAD	StartDeviceAD
ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
ReleaseDevice		

◆ **DMA 方式读取 PCI 设备上的 AD 数据**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceDmaAD (HANDLE hDevice,
                      PULONG pADBuffer,
                      ULONG nReadSizeWords,
                      PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceDmaAD Lib "PCI8603" (ByVal hDevice As Long, _
                                                ByRef pADBuffer As Long,_
                                                ByVal nReadSizeWords As Long,_
                                                ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceDmaAD (hDevice : Integer;
                          pADBuffer: Pointer;
                          nReadSizeWords : LongWord;
                          nRetSizeWords : Pointer) : Boolean;
                          StdCall; External 'PCI8603' Name ' ReadDeviceDmaAD ';
```

LabVIEW：

请参考相关演示程序。

功能：当 AD 标志有效时，用此函数读取设备上的 AD 数据(DMA 方式)。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pADBuffer接受AD数据的用户缓冲区，它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ReadDeviceDmaAD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。此参数值只与ADBuffer[]指定的缓冲区大小有效，而与FIFO存储器大小无效。

nRetSizeWords指定在读取AD数据的过程中，是否对溢出标志进行检测。默认值为FALSE。若指定为TRUE，表示对溢出标志进行监控，若发生溢出，则该函数立即返回，其返回值表示在溢出前已成功读取的AD数据点数，但这个返回值必将小于nReadSizeWords参数的值。若指定为FALSE，则表示不对FIFO存储器的溢出标志进行监控，即便溢出已发生，也始终返回由nReadSizeWords参数指定长度的数据，其返回值也必将等于nReadSizeWords参数值，除非用户在这个函数返回前，就提前调用了ReleaseDeviceAD函数要释放AD设备，那么返回值也可能小于nReadSizeWords参数值。究竟是溢出还是提前释放AD引起的返回值小于nReadSizeWords参数值，用户可以在这种情况下，调用GetLastErrorEx来判断。

返回值： 其返回值表示所成功读取的数据点数(字)，也表示当前读操作在ADBuffer[]缓冲区中的有效数据量。通常情况下其返回值应与ReadSizeWords参数指定量的数据长度(字)相等，除非用户在这个读操作以外的其他线程中执行了ReleaseDeviceAD函数中断了读操作，否则设备可能有问题。对于返回值不等于nReadSizeWords参数值的，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	InitDeviceAD	StartDeviceAD
ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
ReleaseDevice		

- **暂停 AD 设备**

函数原型：

Visual C++ & C++Builder:

```
BOOL StopDeviceAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceAD Lib "PCI8603" (ByVal hDevice As Long )As Boolean
```

Delphi:

```
Function StopDeviceAD (hDevice : Integer ) : Boolean;
                       StdCall; External 'PCI8603' Name ' StopDeviceAD ';
```

LabVIEW:

请参考相关演示程序。

功能：暂停AD设备。它必须在调用StartDeviceAD后才能调用此函数。该函数除了停止AD设备不再转换以外，不改变设备的其他任何状态。此后您可再调用StartDeviceAD函数重新启动AD，此时AD会按照暂停以前的

状态（如FIFO存储器位置、通道位置）开始转换。

参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

返回值：如果调用成功，则返回TRUE，且AD立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	InitDeviceAD	StartDeviceAD
ReadDeviceProAD_Npt	GetDevStatusAD	ReadDeviceProAD_Half
ReadDeviceDmaAD	StopDeviceAD	ReleaseDeviceAD
ReleaseDevice		

♦ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

BOOL ReleaseDeviceAD(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceAD Lib "PCI8603" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDeviceAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI8603' Name ' ReleaseDeviceAD ';

LabVIEW:

请参考相关演示程序。

功能：释放设备上的 AD 部件。

参数：hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

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

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

相关函数：

CreateDevice	InitDeviceAD	ReleaseDeviceAD
ReleaseDevice		

♦ 程序查询方式采样函数一般调用顺序

非空查询方式:

① CreateDevice
② InitDeviceAD
③ StartDeviceProAD
④ ReadDeviceProAD_Npt
⑤ StopDeviceProAD
⑥ ReleaseDeviceProAD
⑦ ReleaseDevice

注明：用户可以反复执行第④步，以实现高速连续不间断大容量采集。

半满查询方式:

① CreateDevice
② InitDeviceAD
③ StartDeviceProAD
④ GetDevStatusDA
⑤ ReadDeviceProAD_Half
⑥ StopDeviceProAD
⑦ ReleaseDeviceProAD
⑧ ReleaseDevice

注明：用户可以反复执行第④、⑤步，以实现高速连续不间断大容量采集。

关于两个过程的图形说明请参考《使用纲要》。

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++ & C++ Builder:

```
BOOL LoadParaAD(HANDLE hDevice,
                PPCI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PCI8603" (ByVal hDevice As Long, _
                    ByRef pADPara As PCI8603_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD (hDevice : Integer;
                     pADPara : PPCI8603_PARA_AD) : Boolean;
                     StdCall; External 'PCI8603' Name ' LoadParaAD ';
```

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或CreateDeviceEx创建。

pADPara属于PPCI8603_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PPCI8603_PARA_AD请参考PCI8603.h或PCI8603.Bas或PCI8603.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：

Visual C++ & C++ Builder:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPCI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI8603" (ByVal hDevice As Long, _
                    ByRef pADPara As PCI8603_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
                     pADPara : PPCI8603_PARA_AD) : Boolean;
                     StdCall; External 'PCI8603' Name ' SaveParaAD ';
```

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或CreateDeviceEx创建。

pADPara设备硬件参数，关于PCI8603_PARA_AD的详细介绍请参考PCI8603.h或PCI8603.Bas或PCI8603.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　　LoadParaAD　　SaveParaAD
ReleaseDevice

♦ AD 采样参数复位至出厂默认值函数

函数原型：

Visual C++ & C++ Builder:

```
BOOL ResetParaAD (HANDLE hDevice,
                  PPCI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "PCI8603" ( ByVal hDevice As Long, _
                                              ByRef pADPara As PCI8603_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD ( hDevice : Integer;
                       pADPara : PPCI8603_PARA_AD) : Boolean;
                       StdCall; External 'PCI8603' Name ' ResetParaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无法确定错误原因的后果。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pADPara设备硬件参数，它负责在参数被复位后返回其复位后的值。关于PCI8603_PARA_AD的详细介绍请参考PCI8603.h或PCI8603.Bas或PCI8603.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD
ResetParaAD　ReleaseDevice

第五节、DA 数据采样操作函数原型说明

◆ 设置触发电平

函数原型：

***Visual C++ & C++Builder*:**

```
BOOL SetDevTrigLevelDA ( HANDLE hDevice,
                         float fTrigLevelVolt)
```

Visual Basic:

```
Declare Function SetDevTrigLevelDA Lib "PCI8603" (ByVal hDevice As Long, _
                                                  ByVal fTrigLevelVolt As Single ) As Boolean
```

Delphi:

```
Function SetDevTrigLevelDA (hDevice : Integer;
                            fTrigLevelVolt: Single) : Boolean;
                            StdCall; External 'PCI8603' Name ' SetDevTrigLevelDA';
```

***LabView*:**

请参考相关演示程序。

功能：设置触发电平，该触发电平对所有 DA 通道同时有效。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

nTrigLevelVolt 触发电平值，单位为 mV，其取值范围为[-10000, +10000.0]。

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　SetDevTrigLevelDA　SetDevFrequencyDA
ReadSegmentInfo　InitDeviceDA　WriteDeviceOneDA
WriteDeviceBulkDA　ReadDeviceBulkDA　EnableDeviceDA
SetDeviceTrigDA　GetDevStatusDA　DisableDeviceDA
ReleaseDeviceDA　ReleaseDevice

◆ 动态改变采样频率

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevFrequencyDA (HANDLE hDevice,
                        LONG nFrequency,
                        int nDAChannel)
```

Visual Basic:

```
Declare Function SetDevFrequencyDA Lib "PCI8603" (ByVal hDevice as Long, _
```

```
                                  ByVal nFrequency As Long,_
                                  ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function SetDevFrequencyDA (hDevice : Integer;
                      nFrequency: LongInt;
                      nDAChannel : Integer):Boolean;
                      StdCall; External 'PCI8603' Name ' SetDevFrequencyDA ';
```

LabVIEW:

请参考演示源程序。

功能：在 DA 采样过程中，可动态改变采样频率。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

nFrequency 采样频率，取值范围为[0.010Hz, 1MHz]，当为正数时，其单位为 Hz。当为负数时，其单位为 0.001Hz，比如该参数等于 25000，则表示其频率为 25000Hz， 再如该参数等于-150， 则表示其频率为 0.150Hz。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 读段信息

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadSegmentInfo(HANDLE hDevice,
                     LONG SegmentCount,
                     PCI8603_SEGMENT_INFO SegmentInfo[],
                     int nDAChannel)
```

Visual Basic:

```
Declare Function ReadSegmentInfo Lib "PCI8603" (ByVal hDevice as Long, _
                              ByVal SegmentCount As Long,_
                              ByRef SegmentInfo() As PCI8603_SEGMENT_INFO,_
                              ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function ReadSegmentInfo (hDevice : Integer;
                     SegmentCount : LongInt;
                     SegmentInfo[] : PCI8603_SEGMENT_INFO;
                     nDAChannel : Integer):Boolean;
                     StdCall; External 'PCI8603' Name ' ReadSegmentInfo';
```

LabVIEW:

请参考演示源程序。

功能： 读段信息。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

SegmentCount 分段总数，它的理论取值范围为[1, 65536]，但实际工作时，它要受到板载 RAM 大小、各段数据长度大小及其他通道对 RAM 的使用情况等因素的影响。在该通道分配的板载 RAM 空间一定的情况下，其各段数据长度越短，则可分配的段数越多。

SegmentInfo[] 段信息集合，属于 PCI8603_SEGMENT_INFO 的结构数据类型，它的有效元素个数由 SegmentCount 参数决定。因此用户分配的段信息集合不应小于 SegmentCount 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中，它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并

加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ 初始化设备对象

函数原型

***Visual C++ & C++Builder*：**

```
BOOL InitDeviceDA(HANDLE hDevice,
                  LONG SegmentCount,
                  PCI8603_SEGMENT_INFO SegmentInfo[],
                  PPCI8603_PARA_DA pDAPara,
                  int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceDA Lib "PCI8603" (  ByVal hDevice As Long, _
                                    ByVal SegmentCount As Long,_
                                        ByRef SegmentInfo() As PCI8603_SEGMENT_INFO,_
                                        ByRef pDAPara As PCI8603_PARA_DA,_
                                        ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function InitDeviceDA(hDevice : Integer;
                   SegmentCount: LongInt;
                   SegmentInfo[] : PCI8603_SEGMENT_INFO;
                       pDAPara:PPCI8603_PARA_DA;
                       nDAChannel: Integer):Boolean;
                       StdCall; External 'PCI8603' Name 'InitDeviceDA';
```

***LabVIEW*：**

请参考相关演示程序。

功能：它负责初始化设备对象中的DA部件，为设备操作就绪有关工作做准备，如预置DA采集通道、采样频率等。但它并不启动DA设备，若要启动DA设备，须在调用此函数之后再调用EnableDeviceDA(但DA要实际输出波形，则一般要等待某种触发事件的到来)。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

SegmentCount 分段总数，它的理论取值范围为[1, 65536]，但实际工作时，它要受到板载 RAM 大小、各段数据长度大小及其他通道对 RAM 的使用情况等因素的影响。在该通道分配的板载 RAM 空间一定的情况下，其各段数据长度越短，则可分配的段数越多。

SegmentInfo[] 段信息集合，属于 PCI8603_SEGMENT_INFO 的结构数据类型，它的有效元素个数由 SegmentCount 参数决定。因此用户分配的段信息集合不应小于 SegmentCount 的指定的大小。注意此段信息集合将被此函数写入板载 RAM 中，它与后面的 DA 数据区共享该通道指定的板载 RAM 区域。

pDAPara 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，如采样频率等。关于具体操作请参考《DA硬件参数结构》。

nDAChannel DA 通道号，取值范围为[0, 1]。

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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **DA 单点输出**

函数原型：

Visual C++ & C++Builder:

```
BOOL WriteDeviceOneDA (HANDLE hDevice,
                       ULONG ulDataCode,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceOneDA Lib "PCI8603" (ByVal hDevice As Long,_
                                                 ByVal ulDataCode As Long,_
                                                 ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceOneDA (hDevice : Integer;
                           ulDataCode : LongWord;
                           nDAChannel: Integer):Boolean;
                           StdCall; External 'PCI8603' Name ' WriteDeviceOneDA';
```

LabVIEW:

请参考相关演示程序。

功能：DA 单点输出。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

ulDataCode 输出码值（0—4095）。

nDAChannel 通道号，取值范围为[0,1]。

返回值：若 DA 成功单点输出，则返回 TRUE，否则返回 FALSE。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

- **批量方式将用户缓冲区中的 DA 数据传输至板载 RAM 中**

函数原型:

Visual C++ & C++Builder:

```
BOOL WriteDeviceBulkDA ( HANDLE hDevice,
                         SHORT DABuffer[],
                         LONG nWriteOffsetWords,
                         LONG nWriteSizeWords,
                         PLONG nRetSizeWords,
                         int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceBulkDA Lib "PCI8603" (ByVal hDevice as Long, _
                                                  ByRef DABuffer() As Integer, _
                                                  ByVal nWriteOffsetWords As Long, _
                                                  ByVal nWriteSizeWords As Long, _
                                                  ByRef nRetSizeWords As Long, _
                                                  ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceBulkDA (hDevice : Integer;
                            DABuffer[] : SmallInt;
                            nWriteOffsetWords: LongInt;
                            nWriteSizeWords:LongInt;
                            nRetSizeWords : Pointer;
                            nDAChannel : Integer):Boolean;
                            StdCall; External 'PCI8603' Name ' WriteDeviceBulkDA ';
```

LabView:

请参考相关演示程序。

功能：往指定通道的板载 RAM 中写入批量 DA 数据。在初始化设备之后，启动 DA 之前，便可以用此函数将 DA 数据写入板载 RAM 以供输出。但是在启动之后（即在输出过程中，不能对 RAM 进行写操作，包括读操作）。

参数：

hDevice 设备对象句柄,它应由CreateDevice或CreateDeviceEx创建。

DABuffer[] 接受DA数据的用户缓冲区地址，它可以是一个 16Bit整型数组，也可以是由其他方式分配的16Bit整型缓冲区。关于如何将这些DA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》。

nWriteOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)。此参数不能超过 RAM 的最大长度(字/点)。

nWriteSizeWords 指定一次往物理缓冲区由 nWriteOffsetWords 参数指定偏移位置开始写入的数据长度。注意此参数的值与 nWriteOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。同时此参数值不能大于 DABuffer[]指定的缓冲区的长度。

nRetSizeWords 返回当前写操作实际实现的数据长度。它表明该函数调用后，在 DABuffer[]中有多少数据是有效的。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **以批量方式将板载 RAM 中的 DA 数据回读至主机的用户缓冲区**

函数原型:

Visual C++ & C++Builder:

```
BOOL ReadDeviceBulkDA ( HANDLE hDevice,
                        SHORT DABuffer[],
                        LONG nReadOffsetWords,
                        LONG nReadSizeWords,
                        PLONG nRetSizeWords,
                        int nDAChannel)
```

Visual Basic:

```
Declare Function ReadDeviceBulkDA Lib "PCI8603" ( ByVal hDevice as Long, _
                                                  ByRef DABuffer() As Integer, _
                                                  ByVal nReadOffsetWords As Long, _
                                                  ByVal nReadSizeWords As Long, _
                                                  ByRef nRetSizeWords As Long, _
                                                  ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function ReadDeviceBulkDA (hDevice : Integer;
                           DABuffer[] : SmallInt;
                           nReadOffsetWords : LongInt;
                           nReadSizeWords:LongInt;
                           nRetSizeWords : Pointer
                           nDAChannel : Integer): Boolean;
                           StdCall; External 'PCI8603' Name ' ReadDeviceBulkDA';
```

LabView：

请参考相关演示程序。

功能：从指定通道的 RAM 中的指定段以及指定段内偏移位置开始将 DA 数据从板载 RAM 中读回至主机的用户缓冲区。但是在启动之后（即在输出过程中），不能对 RAM 进行读操作，包括写操作。该函数的作用是为了验证写入的数据是否正确而提供的。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

DABuffer[] 接受DA数据的用户缓冲区地址，它可以是一个 16Bit整型数组，也可以是由其他方式分配的16Bit整型缓冲区。关于如何将这些DA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》。

nReadOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)。此参数不能超过 RAM 的最大长度(字/点)。

nReadSizeWords 指定一次从物理缓冲区由 nReadOffsetWords 参数指定偏移位置开始读入的数据长度。注意此参数的值与 nWriteOffsetWords 参数值之和不能大于指定通道的物理缓冲区即板上 RAM 的最大长度。同时此参数值不能大于 DABuffer[]指定的缓冲区的长度。

nRetSizeWords 返回当前写操作实际实现的数据长度。它表明该函数调用后，在 DABuffer[]中有多少数据是有效的。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

♦ **启动 DA 设备**

函数原型

Visual C++ & C++Builder:

```
BOOL EnableDeviceDA (HANDLE hDevice,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function EnableDeviceDA Lib "PCI8603" (ByVal hDevice As Long,_
                                    ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function EnableDeviceDA (hDevice : Integer;
                     nDAChannel : Integer ): Boolean;
                     StdCall; External 'PCI8603' Name ' EnableDeviceDA ';
```

LabVIEW:

请参考相关演示程序。

功能：启动DA设备，它必须在调用InitDeviceDA后才能调用此函数。调用该函数后它可能立即启动，这就要取决您选择的触发方式或触发源，详细请参考后面的《触发功能详述》。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

nDAChannel 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，且DA准备就绪，等待触发事件的到来就开始实际的DA输出， 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

♦**软件产生触发事件**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceTrigDA ( HANDLE hDevice ,
                     BOOL bSetSyncTrig,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function SetDeviceTrigDA Lib "PCI8603" (ByVal hDevice As Long,_
                                    ByVal bSetSyncTrig As Boolean,_
                                         ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function SetDeviceTrigDA (hDevice : Integer;
                   bSetSyncTrig : Boolean;
                       nDAChannel : Integer ): Boolean;
                       StdCall; External 'PCI8603' Name ' SetDeviceTrigDA ';
```

LabVIEW:

请参考相关演示程序。

功能：在指定通道被启动后，可由此函数以软件产生触发事件。当然只有当用户选择触发源为软件触发时方有效。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

bSetSyncTrig 是否置同步触发。

nDAChannel 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，则产生一个软件触发事件，在某些触发模式下会直接使DA输出波形， 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	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **取得 DA 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDevStatusDA (HANDLE hDevice,
                     PPCI8603_STATUS_DA pDAStatus,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PCI8603" (ByVal hDevice As Long,_
                                    ByRef pDAStatus As PCI8603_STATUS_DA,_
                                    ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function GetDevStatusDA (hDevice : Integer;
                     pDAStatus : PPCI8603_STATUS_DA;
                     nDAChannel: Integer):Boolean;
                     StdCall; External 'PCI8603' Name ' GetDevStatusDA ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用EnableDeviceDA后，可以用此函数却查询DA状态，如是否被启动（bConverting)，触发标志是否有效(bTrigFlag)，当前段循环次数(nCurSegLoopCount)等信息。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pDAStatus 设备状态参数结构，它返回设备当前的各种状态，如板载RAM是否发生切换、重写、触发点是否产生等信息。关于具体操作请参考《DA硬件参数结构》。

nDAChannel 通道号，取值范围为[0, 1]。

返回值：若 DA 成功取回标状态，则返回 TRUE，否则返回 FALSE。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **暂停 DA 设备**

函数原型

Visual C++ & C++Builder：

```
BOOL DisableDeviceDA (HANDLE hDevice,
                      int nDAChannel )
```

Visual Basic:

```
Declare Function DisableDeviceDA Lib "PCI8603" (ByVal hDevice as Long,_
                                    ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function DisableDeviceDA (hDevice : Integer;
                      nDAChannel As Integer) : Boolean;
```

StdCall; External 'PCI8603' Name ' DisableDeviceDA ';

LabVIEW：

请参考相关演示程序。

功能：暂停DA设备。它必须在调用EnableDeviceDA后才能调用此函数。该函数除了停止DA设备不再转换以外，不改变设备的其他任何工作参数。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

nDAChannel 通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，且DA立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **释放 DA 设备**

函数原型

Visual C++ & C++Builder：

```
BOOL ReleaseDeviceDA ( HANDLE hDevice,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function ReleaseDeviceDA Lib "PCI8603" (ByVal hDevice as Long, _
                                    ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function ReleaseDeviceDA (hDevice : Integer ;
                          nDAChannel As Integer) : Boolean;
                          StdCall; External 'PCI8603' Name ' ReleaseDeviceDA';
```

LabView：

请参考相关演示程序。

功能：释放DA设备。它必须在调用InitDeviceDA后的某个时刻调用此函数。该函数除了停止DA设备，还释放掉所占用的各种资源。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回TRUE，且DA立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevTrigLevelDA	SetDevFrequencyDA
ReadSegmentInfo	InitDeviceDA	WriteDeviceOneDA
WriteDeviceBulkDA	ReadDeviceBulkDA	EnableDeviceDA
SetDeviceTrigDA	GetDevStatusDA	DisableDeviceDA
ReleaseDeviceDA	ReleaseDevice	

◆ **采样和和传输函数一般调用顺序**

①CreateDevice (创建设备对象)

②InitDeviceDA (初始化设备)

③ WriteDeviceBulkDA(批量写入DA数据到板载RAM)

④EnableDeviceDA (启动DA设备)

⑤SetDevTrigLevelDA (若为软件触发源，则置软件触发事件)

⑥ GetDevStatusDA（循环查询DA状态）

⑦ DisableDeviceDA

⑧ ReleaseDevice

关于调用过程的图形说明请参考《绪论》。

第六节、DA 硬件参数系统保存与读取函数原型说明

◆ **写设备硬件参数函数到 Windows 系统中**

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaDA (HANDLE hDevice,
                 PPCI8603_PARA_DA pDAPara,
                 int nDAChannel)
```

Visual Basic:

```
Declare Function SaveParaDA Lib "PCI8603" (ByVal hDevice As Long, _
                                  ByRef pDAPara As PCI8603_PARA_DA,_
                                  ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function SaveParaDA (hDevice : Integer;
                     pDAPara:PPCI8603_PARA_DA;
                     nDAChannel : Integer):Boolean;
                     StdCall; External 'PCI8603' Name ' SaveParaDA ';
```

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或CreateDeviceEx创建。

pDAPara 设备硬件参数，关于 PPCI8603_PARA_DA 的详细介绍请参考 PCI8603.h 或 PCI8603.Bas 或 PCI8603.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaDA SaveParaDA ReleaseDevice

◆ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++ & C++ Builder:

```
BOOL LoadParaDA(HANDLE hDevice,
                PPCI8603_PARA_DA pDAPara,
                int nDAChannel)
```

Visual Basic:

```
Declare Function LoadParaDA Lib "PCI8603" (ByVal hDevice As Long, _
                                  ByRef pDAPara As PCI8603_PARA_DA, _
                                  ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function LoadParaDA (hDevice : Integer;
                     pDAPara:PPCI8603_PARA_DA;
                     nDAChannel : Integer):Boolean;
                     StdCall; External 'PCI8603' Name ' LoadParaDA ';
```

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或CreateDeviceEx创建。

pDAPara属于PPCI8603_PARA_DA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PPCI8603_PARA_DA请参考PCI8603.h或PCI8603.Bas或PCI8603.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaDA SaveParaDA

ReleaseDevice

◆ 将硬件参数结构体值复位为出厂默认值

函数原型：

Viusal C++ & C++ Builder:

BOOL ResetParaDA (HANDLE hDevice,
PPCI8603_PARA_DA pDAPara,
int nDAChannel)

Visual Basic:

Declare Function ResetParaDA Lib "PCI8603" (ByVal hDevice As Long, _
ByRef pDAPara As PCI8603_PARA_DA,_
ByVal nDAChannel As Integer) As Boolean

Delphi:

Function ResetParaDA (hDevice : Integer;
pDAPara:PPCI8603_PARA_DA;
nDAChannel As Integer):Boolean;
StdCall; External 'PCI8603' Name ' ResetParaDA ';

LabVIEW:

请参考相关演示程序。

功能：负责将硬件参数的值复位至出厂默认值，不仅会将 pDAPara 指向的结构体成员值更新为默认值，同时会将系统中保存的参数更新为默认值。这些默认值在产品驱动第一次被安装时会出现。而且这些默认值的设定是充分的考虑到用户的实际情况，确保用户不用外部任何条件，只要开始采集数据，即可获得相应的结果。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

pDAPara 设备硬件参数，关于 PPCI8603_PARA_DA 的详细介绍请参考 PCI8603.h 或 PCI8603.Bas 或 PCI8603.Pas函数原型定义文件，也可参考《硬件参数结构》关于该结构的有关说明。调用此函数后，该参数指向的结构体成员将被复位至默认值。

nDAChannel DA 通道号，取值范围为[0, 1]。

返回值：若成功，返回 TRUE，它表明已成功将系统中的 DA 参数复位至默认值，同时更新了 pDAPara 指向的结构体。否则返回 FALSE。

相关函数： CreateDevice LoadParaDA SaveParaDA
ResetParaDA ReleaseDevice

第七节、DIO 数字量输入输出开关量操作函数原型说明

◆ 八路开关量输入

函数原型：

Visual C++ & C++Builder:

BOOL GetDeviceDI (HANDLE hDevice,
BYTE bDISts[8])

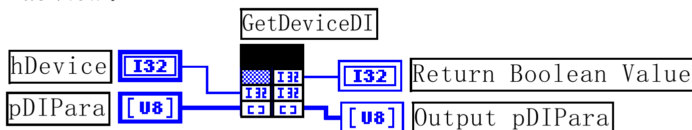
Visual Basic:

Declare Function GetDeviceDI Lib "PCI8603" (ByVal hDevice As Long, _
ByVal bDISts(0 to 7) As Byte) As Boolean

Delphi:

Function GetDeviceDI (hDevice : Integer;
bDISts : Pointer):Boolean;
StdCall; External 'PCI8603' Name ' GetDeviceDI ';m

LabView：



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或CreateDeviceEx创建。

bDISts八路开关量输入状态的参数结构，共有 8 个成员变量，分别对应于DI0～DI7 路开关量输入状态位。如果bDISts[0]为“1”则使 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。具体定义请参考《DI数字开关量输入参数介绍》章节。

返回值：若成功，返回 TRUE，其 bDISts[x]中的值有效；否则返回 FALSE，其 bDISts[x]中的值无效。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

◆ 八路开关量输出

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[8])
```

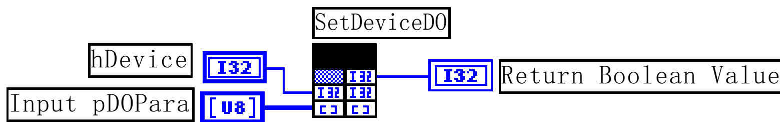
Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI8603" ( ByVal hDevice As Long, _
                                             ByVal bDOSts(0 to 7 )As Byte) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer;
                      bDOSts : Pointer):Boolean;
                      StdCall; External 'PCI8603' Name ' SetDeviceDO ';
```

LabView:



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或CreateDeviceEx创建。

bDOSts 八路开关量输出状态的参数结构，共有 8 个成员变量，分别对应于DO0～DO7 路开关量输出状态位。比如置bDOSts[0]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DO7 共 8 个成员变量赋初值，其值必须为“1”或“0”。具体定义请参考《DO数字开关量输出参数介绍》。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

◆ 回读数字量输出状态

函数原型：

Visual C++ & C++Builder:

```
BOOL RetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[8])
```

Visual Basic:

```
Declare Function RetDeviceDOLib "PCI8603" (ByVal hDevice As Long, _
                                           ByVal bDOSts(0 to 7) As Byte) As Boolean
```

Delphi:

```
Function RetDeviceDO (hDevice : Integer;
                      bDOSts : Pointer) : Boolean;
                      StdCall; External 'PCI8603' Name ' RetDeviceDO';
```

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由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

bDOSts 获得开关输出状态(注意：必须定义为 8 个字节元素的数组)。

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（PCI8603_PARA_AD）

Visual C++ & C++Builder：

```
typedef struct _PCI8603_PARA_AD
{
	LONG ADMode;			// AD 模式选择(连续/分组方式)
	LONG FirstChannel;		// 首通道[0,31]
	LONG LastChannel;		// 末通道[0,31]，要求末通道必须大于或等于首通道
	LONG Frequency;			// 采集频率，单位为 Hz，[1, 500000]
	LONG GroupInterval;		// 分组时的组间间隔(单位：微秒) [1, 419430]
	LONG LoopsOfGroup;		// 组内循环次数[1, 255]
	LONG Gains;			// 增益设置
	LONG InputRange;		// 模拟量输入量程范围
	LONG TriggerMode;		// 触发模式选择
	LONG TriggerType;		// 触发类型选择(边沿触发/脉冲触发)
	LONG TriggerDir;		// 触发方向选择(正向/负向触发)
	LONG TriggerSource;		// 触发源选择
	LONG TrigLevelVolt;		// 触发电平(±10000mV)
	LONG TrigWindow;		// 触发灵敏窗[1, 65535], 单位 25 纳秒
	LONG ClockSource;		// 时钟源选择(内/外时钟源)
	LONG GroundingMode;		// 接地方式（单端或双端选择）
}PCI8603_PARA_AD, *PPCI8603_PARA_AD;
```

Visual Basic：

```
Private Type PCI8603_PARA_AD
	ADMode As Long			' AD 模式选择(连续/分组方式)
	FirstChannel As Long		' 首通道[0,31]
	LastChannel As Long		' 末通道[0,31]，要求末通道必须大于或等于首通道
	Frequency As Long		' 采集频率，单位为 Hz，[1, 500000]
	GroupInterval As Long		' 分组时的组间间隔(单位：微秒) [1, 419430]
	LoopsOfGroup As Long		' 组内循环次数[1, 255]
	Gains As Long			' 增益设置
	InputRange As Long		' 模拟量输入量程范围
	TriggerMode As Long		' 触发模式选择
	TriggerType As Long		' 触发类型选择(边沿触发/脉冲触发)
	TriggerDir As Long		' 触发方向选择(正向/负向触发)
	TriggerSource As Long		' 触发源选择
	TrigLevelVolt As Long		' 触发电平(±10000mV)
	TrigWindow As Long		' 触发灵敏窗[1, 65535], 单位 25 纳秒
	ClockSource As Long		' 时钟源选择(内/外时钟源)
	GroundingMode As Long		' 接地方式（单端或双端选择）
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
```

```
    PPCI8603_PARA_AD = ^ PCI8603_PARA_AD;    // 指针类型结构
    PCI8603_PARA_AD = record                 // 标记为记录型
      ADMode : LontInt;                      // AD 模式选择(连续/分组方式)
      FirstChannel : LontInt;                // 首通道[0,31]
      LastChannel : LontInt;                 // 末通道[0,31]，要求末通道必须大于或等于首通道
      Frequency : LontInt;                   // 采集频率，单位为 Hz，[1, 500000]
      GroupInterval : LontInt;               // 分组时的组间间隔(单位：微秒) [1, 419430]
      LoopsOfGroup : LontInt;                // 组内循环次数[1, 255]
      Gains : LontInt;                       // 增益设置
      InputRange : LontInt;                  // 模拟量输入量程范围
      TriggerMode : LontInt;                 // 触发模式选择
      TriggerType : LontInt;                 // 触发类型选择(边沿触发/脉冲触发)
      TriggerDir : LontInt;                  // 触发方向选择(正向/负向触发)
      TriggerSource: LontInt;                // 触发源选择
      TrigLevelVolt: LontInt;                // 触发电平(±10000mV)
      TrigWindow : LontInt;                  // 触发灵敏窗[1, 65535]，单位 25 纳秒
      ClockSource : LontInt;                 // 时钟源选择(内/外时钟源)
      GroundingMode: LongInt;                // 接地方式（单端或双端选择）
  End;
```

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、DMA 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或InitDeviceIntAD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。

ADMode　AD 采样模式。它的取值如下表：

常量名	常量值	功能定义
PCI8603_ADMODE_SEQUENCE	0x00	连续采集模式
PCI8603_ADMODE_GROUP	0x01	分组采集模式

连续采集模式：表示所有通道在采样过程中均按相等时间间隔转换，即所有被采样的点在时间轴上其间隔完全相等。在图 4.1 中的过程，若没有mt的组间间隔时间GroupInterval，就属于连续采样的模式。

分组采集模式：表示所有采样的数据点均以指定的通道数分成若干组，组内各通道数据点按等间隔采样，其频率由Frequency参数决定，组与组之间则有相当的间隔时间，其间隔长度由参数GroupInterval决定，可以精确到微秒。如图 4.1 即是分组采样的整过情况。

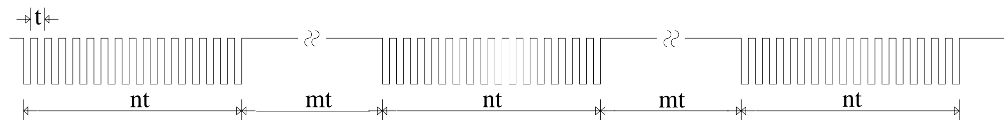


图 4.1

说明：t = 1/Frequency
　　　mt = GroupInterval
　　　n = ChannelCount

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。

Frequency　AD 采样频率，本设备的频率取值范围为[1, 500000] Hz。
注意：
在内时钟(即ClockSource = PCI8603_CLOCKSRC_IN)方式下：
若连续采集(即ADMode = PCI8603_ADMODE_SEQUENCE)时，此参数控制各处通道间的采样频率。若分

组采集(即ADMode= PCI8603_ADMODE_GROUP)时，则此参数控制各组组内的采样频率，而组间时间则由GroupInterval决定。

在外时钟(即ClockSource = PCI8603_CLOCKSRC_OUT)方式下：

若连续采集(即ADMode = PCI8603_ADMODE_SEQUENCE)时，此参数自动失效，因为外时钟的频率代替了此参数设定的频率。若为分组采集(即ADMode= PCI8603_ADMODE_GROUP)时，则该参数控制各组组内的采样频率，而外时钟则是每组的触发频率。此时，GroupInterval参数无效。

GroupInterval 组间间隔，单位微秒 uS，其范围[1, 419430]，通常由用户确定。但是一般情况下，此间隔时间应不小于组内相邻两通道的间隔。在内时钟连续采集模式和外时钟模式下，则参数无效。

LoopsOfGroup 在分组采集模式中，控制各组的循环次数。取值范围为[1, 255]。比如，1、2、3、4 通道分组采样，当此参数为 2 时，则表示每 1、2、3、4、1、2、3、4 为一采样组，然后再延时GroupInerval指定的时间再接着采集 1、2、3、4、1、2、3、4，依此类推。

Gains AD 采样程控增益。

常量名	常量值	功能定义
PCI8603_GAINS_1MULT	0x00	1 倍增益
PCI8603_GAINS_2MULT	0x01	2 倍增益
PCI8603_GAINS_4MULT	0x02	4 倍增益
PCI8603_GAINS_8MULT	0x03	8 倍增益

InputRange AD 模拟信号输入范围，取值如下表：

常量名	常量值	功能定义
PCI8603_INPUT_N10000_P10000mV	0x00	±10000mV
PCI8603_INPUT_N5000_P5000mV	0x01	±5000mV
PCI8603_INPUT_N2500_P2500mV	0x02	±2500mV
PCI8603_INPUT_0_P10000mV	0x03	0～10000mV

关于各个量程下采集的数据ADBuffer[]如何换算成相应的电压值，请参考《AD原码LSB数据转换成电压值的换算方法》章节。

TriggerMode AD 触发模式。

常量名	常量值	功能定义
PCI8603_TRIGMODE_SOFT	0x00	软件触发(属于内触发)
PCI8603_TRIGMODE_POST	0x01	硬件后触发(属于外触发)

TriggerType AD 触发类型。

常量名	常量值	功能定义
PCI8603_TRIGTYPE_EDGE	0x00	边沿触发
PCI8603_TRIGTYPE_PULSE	0x01	脉冲触发(电平)

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
PCI8603_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
PCI8603_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
PCI8603_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：PCI8603_TRIGDIR_POSIT_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。而在电平类型下，无论正电平还是负电平均触发。

TriggerSource 触发源选择。

常量名	常量值	功能定义
PCI8603_TRIGSRC_ATR_AD	0x00	选择外部 ATR 作为触发源
PCI8603_TRIGSRC_DTR_AD	0x01	选择外部 DTR 作为触发源

TrigLevelVolt 触发电平(±10000mV)。
TrigWindow 触发灵敏窗[1, 65535]，单位 25 纳秒。
ClockSource AD 触发时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PCI8603_CLOCKSRC_IN	0x00	内部时钟定时触发
PCI8603_CLOCKSRC_OUT	0x01	外部时钟定时触发

当选择内时钟时，其AD定时触发时钟为板上时钟振荡器经分频得到。它的大小由Frequency参数决定。
当选择外时钟时：
当选择连续采集时(即ADMode = PCI8603_ADMODE_SEQUENCE)，其AD定时触发时钟为外界时钟输入CLKIN得到，而Frequency参数则自动失效。
但是当选择分组采集时(即ADMode = PCI8603_ADMODE_GROUP)，外时钟则是每一组的触发时钟信号，而组内的触发频率则由Frequency参数决定，由此可见，此时外时钟触发周期必须大于每组总周期，否则紧跟其后的某一外时钟可能会被失效。

GroundingMode AD 接地方式选择。它的选项值如下表：

常量名	常量值	功能定义
PCI8603_ GNDMODE_SE	0x00	单端方式(SE:Single end)
PCI8603_ GNDMODE_DI	0x01	双端方式(DI:Differential)

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第二节、AD 状态参数结构（PCI8603_STATUS_AD）

Visual C++ & C++Builder：

```
typedef struct _PCI8603_STATUS_AD
{
    LONG bNotEmpty; LONG bHalf;
    LONG bDynamic_Overflow;
    LONG bStatic_Overflow;
    LONG bConverting;
    LONG bTriggerFlag;
} PCI8603_STATUS_AD, *PPCI8603_STATUS_AD;
```

Visual Basic:

```
Private Type    PCI8603_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bDynamic_Overflow As Long
    bStatic_Overflow As Long
    bConverting As Long
    bTriggerFlag As Long
End Type
```

Delphi:

```
Type    // 定义结构体数据类型
    PPCI8603_STATUS_AD = ^ PCI8603_STATUS_AD;   // 指针类型结构
    PCI8603_STATUS_AD = record          // 标记为记录型
        bNotEmpty : LongInt;
        bHalf : LongInt;
        bDynamic_Overflow : LongInt;
        bStatic_Overflow : LongInt;
        bConverting : LongInt;
        bTriggerFlag : LongInt;
 End;
```

LabVIEW:

请参考相关演示程序。

此结构体主要用于查询AD的各种状态，GetDevStatusDA函数使用此结构体来实时取得AD状态，以便同步各种数据采集和处理过程。

bNotEmpty AD 板载存储器 FIFO 的非空标志，=TRUE 表示存储器处在非空状态，即有可读数据，否则表示空。

bHalf AD 板载存储器 FIFO 的半满标志，=TRUE 表示存储器处在半满状态，即有至少有半满以上数据可读，否则表示在半满以下，可能有小于半满的数据可读。

bDynamic_Overflow AD 板载存储器 FIFO 的溢出标志，=TRUE 表示存储器处在全满或溢出状态，即全满的数据可读数据，但此时的数据很有可能已有丢点现象。否则表示满以下状态。该状态处于动态溢出状态，即 FIFO 随时溢出，它随时=TRUE，而随时不溢出，则随时=FALSE。

bStatic_Overflow AD 板载存储器 FIFO 的溢出标志，=TRUE 表示存储器至少有过一次出现全满或溢出状态，然后永远为 TRUE，除非用户重新开始采集数据则会自动变为 FALSE。在启动采集过程中，只有一次全满或溢出状态都未发生过，则此标志恒等于 FALSE。所以用此标志可以确定在整过采集中是否有过溢出丢点现象。当然要避免丢点现象的发生，您需要考虑应用软件设计的合理性、效率性等各方因素，我们提供的高级演示程序（尤其是 VC）便很好的展示了此类思想。

bConverting AD 是否正在转换，=TRUE:表示正在转换，=FALS 表示转换完成。

bTriggerFlag AD 触发标志，=TRUE 表示已被触发(即产生触发事件)，=FALSE 表示未产生触发事件。

相关函数： CreateDevice GetDevStatusDA ReleaseDevice

第三节、DMA 状态参数结构（PCI8603_STATUS_DMA）

const int PCI8603_MAX_SEGMENT_COUNT = 64;

Visual C++ & C++Builder：

```
typedef struct _PCI8603_STATUS_DMA
{
    LONG iCurSegmentID; //  当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    LONG bSegmentSts[MAX_SEGMENT_COUNT];
    LONG bBufferOverflow;     // 返回溢出状态
} PCI8603_STATUS_DMA, *PPCI8603_STATUS_DMA;
```

Visual Basic：

```
Private Type    PCI8603_STATUS_DMA
    iCurSegmentID As Long                    ' 当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    bSegmentSts(MAX_SEGMENT_COUNT) As Long
    bBufferOverflow As Long               ' 返回溢出状态
End Type
```

Delphi:

```
Type   //  定义结构体数据类型
    P PCI8603_STATUS_DMA = ^ PCI8603_STATUS_DMA;   //  指针类型结构
    PCI8603_STATUS_DMA = record          //  标记为记录型
    iCurSegmentID : LongInt;    //  当前段缓冲 ID，表示 DMA 正在传输的缓冲区段
    bSegmentSts [MAX_SEGMENT_COUNT] : Array [0…63] of LongInt;
    bBufferOverflow : LongInt;   //  返回溢出状态
End;
```

LabVIEW:

请参考相关演示程序。

此结构体主要用于DMA传输时的状态监控，ReadDeviceDmaAD函数使用此结构体来实时取得DMA状态，以便同步各种数据处理过程。

iCurSegmentID DMA 正在传输的当前缓冲段 ID 号。

bSegmentSts[ ] DMA缓冲区各段的状态。 如bSegmentSts[0]=0，表示缓冲区段 0 此时为旧数据段，若=1

则段 0 为新数据段，可以对其进行数据处理。同理，bSegmentSts[1]=0，表示缓冲区段 1 此时为旧数据段，若=1 则段 1 为新数据段，可以对其进行数据处理。注意，每次调用InitDeviceAD初始化设备后，其值自动被复位至 0。

bBufferOverflow 组缓冲区溢出标志。若等于 0，则表示整个DMA缓冲链未发生溢出，若等于 1，则表示整个DMA缓冲链已发生溢出。注意，每次调用InitDeviceAD初始化设备后，其值自动被复位至 0。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

第四节、DA 各段信息参数结构（PCI8603_SEGMENT_INFO）

Visual C++ & C++Builder：

```
typedef struct _PCI8603_SEGMENT_INFO
{
      LONG SegLoopCount;         // 每个段在大循环中的小循环次数，取值为[1, 16777215]
      LONG SegmentSize;          // 每个段在 RAM 中的长度(单位：字/点)
} PCI8603_SEGMENT_INFO, *PPCI8603_SEGMENT_INFO;
```

Visual Basic：

```
Type PCI8603_SEGMENT_INFO
      SegLoopCount As Long       ' 每个段在大循环中的小循环次数，取值为[1, 16777215]
      SegmentSize As Long        ' 每个段在 RAM 中的长度(单位：字/点)
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
PPCI8603_SEGMENT_INFO = ^ PCI8603_SEGMENT_INFO;   // 指针类型结构
PCI8603_SEGMENT_INFO = record    // 标记为记录型
      SegLoopCount : LongInt;         // 每个段在大循环中的小循环次数，取值为[1, 16777215]
      SegmentSize : LongInt;       // 每个段在 RAM 中的长度(单位：字/点)
End;
```

LabVIEW:

请参考相关演示程序。

此参数结构主要用于建立段信息，它包括段长、段循环等，它主要用于函数InitDeviceDA的SegmentInfo[]参数。

SegLoopCount 段循环次数，如果是多段输出，则首先循环输出段 0 的波形数据，直到段 0 循环到该参数指定的次数再跳转到段 1 输出，同样循环段 1 直至循环结束再输出下一段，当然实际情况要看用户选择的触发模式。其取值范围为 20 位，即[1, 16777215]。

SegmentSize 段长，指的是该段的波形数据长度，单位点。其取值范围受该通道分配的板载 RAM 空间大小、总有效输出段数及各段长度决定。

第五节、DA 硬件参数结构（PCI8603_PARA_DA）

Visual C++ & C++Builder：

```
typedef struct _PCI8603_PARA_DA
{
      LONG OutputRange;          // 输出量程
      LONG Frequency;            // 采集频率[0.010Hz, 1MHz]，为正数时单位 Hz，为负数时单位为 0.001Hz
      LONG LoopCount;            // 整过 RAM 的大循环次数，=0:无限循环， =n:表示 n 次循环(1<n<32767)
      LONG TriggerMode;          // 触发模式选择
      LONG TriggerSource;        // 触发源选择
      LONG TriggerDir;           // 触发方向选择
      LONG bSingleOut;           // 是否单点输出
      LONG ClockSource;          // 时钟源选择
```

```
} PCI8603_PARA_DA, *PPCI8603_PARA_DA;
```

Visual Basic：

```
Type PCI8603_PARA_DA
    OutputRange As Long        ' 输出量程
    Frequency As Long          ' 采集频率[0.010Hz, 1MHz]， 为正数时单位 Hz，为负数时单位为 0.001Hz
    LoopCount As Long          ' 整过 RAM 的大循环次数，=0:无限循环， =n:表示 n 次循环(1<n<32767)
    TriggerMode As Long        ' 触发模式选择
    TriggerSource As Long      ' 触发源选择
    TriggerDir As Long         ' 触发源选择
    bSingleOut As Long         ' 是否单点输出
    ClockSource As Long        ' 时钟源选择
End Type
```

Delphi:

```
Type  // 定义结构体数据类型
PPCI8603_PARA_DA = ^ PCI8603_PARA_DA;  // 指针类型结构
PCI8603_PARA_DA = record        // 标记为记录型
    OutputRange : LongInt;      // 输出量程
    Frequency : LongInt;        // 采集频率[0.010Hz, 1MHz]， 为正数时单位 Hz，为负数时单位为 0.001Hz
    LoopCount: LongInt;         // 整过 RAM 的大循环次数，=0:无限循环， =n:表示 n 次循环(1<n<32767)
    TriggerMode: LongInt;       // 触发模式选择
    TriggerSource : LongInt;    // 触发源选择
    TriggerDir : LongInt;       // 触发方向选择
    bSingleOut : LongInt;       // 是否单点输出
    ClockSource : LongInt;      // 时钟源选择
End;
```

LabVIEW:

请参考相关演示程序。

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

OutputRange 输出信号的量程范围，取值如下表：

常量名	常量值	功能定义
PCI8603_OUTPUT_0_P5000mV	0x00	0～5000mV
PCI8603_OUTPUT_0_P10000mV	0x01	0～10000mV
PCI8603_OUTPUT_N10000_P10000mV	0x02	±10000mV
PCI8603_OUTPUT_N5000_P5000mV	0x03	±5000mV

关于电压值到设备原始码之间的换算公式请参考《DA的电压值如何转换成输出到DA转换器的LSB原码数据》。

Frequency DA 输出时的点频率，即刷新频率，单位视该参数取值正负而定：若为正数，则单位为 Hz，如为 120 时表示其点频为 120Hz，若为负数，则单位为 0.001Hz，如为 35 时表示其点频为 0.035Hz。注意实际输出的信号频率是由该点频率与每周期 DA 数据点数共同作用的结果。如点频率是 100KHz，每周期的正弦波数据点数是 256 个点，则输出正弦波信号的频率为 0.390625KHz(即 100 / 256 所得)。

LoopCount 总循环次数，=0：无限循环，=n：表示 n 次循环(1<n<32768)。该参数只有触发模式为连续触发时有效。它表示指定通道指定有效段输出的次数。比如有效段数 SegmentCount 为 3 时，该参数为 4 时，则实际输出的段序列为：

循环第 1 次			循环第 2 次			循环第 3 次			循环第 4 次		
段 0	段 1	段 2	段 0	段 1	段 2	段 0	段 1	段 2	段 0	段 1	段 2

TriggerMode DA 触发模式选择。选项值如下表：

常量名	常量值	功能定义

PCI8603_TRIGMODE_SINGLE	0x0000	单次触发
PCI8603_TRIGMODE_CONTINUOUS	0x0001	连续触发
PCI8603_TRIGMODE_STEPED	0x0002	单步触发
PCI8603_TRIGMODE_BURST	0x0003	紧急触发

TriggerSource DA 转换触发源。选项值如下表：

常量名	常量值	功能定义
PCI8603_TRIGSRC_SOFT_DA	0x0000	软件触发
PCI8603_TRIGSRC_ATR_DA	0x0001	ATR 硬件模拟触发
PCI8603_TRIGSRC_DTR_DA	0x0002	DTR 硬件数字触发

TriggerDir DA 外触发方式使用信号方向，只对硬件模拟 ATR 触发源和硬件 DTR 数字触发源有效。选项值如下表：

常量名	常量值	功能定义
PCI8603_TRIGDIR_NEGATIVE	0x0000	负向触发(低脉冲/下降沿触发)
PCI8603_TRIGDIR_POSITIVE	0x0001	正向触发(高脉冲/上升沿触发)
PCI8603_TRIGDIR_POSIT_NEGAT	0x0002	正负向触发(高/低脉冲或上升/下降沿触发)

当 TriggerDir = PCI8603_TRIGDIR_NEGATIVE 时，表示外部触发信号须由大于 TrigLevelVolt 变成小于 TrigLevelVolt 时产生触发事件（即下降沿触发）。

当 TriggerDir = PCI8603_TRIGDIR_POSITIVE 时，表示外部触发信号由小于 TrigLevelVolt 变成大于 TrigLevelVolt 时产生触发事件 (即上升沿触发)。

当 TriggerDir = PCI8603_TRIGDIR_POSIT_NEGAT 时，凡外部触发信号发生以上两种情况中的任意一种则产生触发事件。

bSingleOut 是否单点输出。

ClockSource DA 外时钟选择，选项值如下表：

常量名	常量值	功能定义
PCI8603_CLOCKSRC_IN	0x0000	内部时钟
PCI8603_CLOCKSRC_OUT	0x0001	外部时钟

第六节、DA 状态参数结构（PCI8603_STATUS_DA）

Visual C++ & C++Builder：

```
typedef struct _PCI8603_STATUS_DA
{
    LONG bEnable;          // DA 使能启动标志，＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止
    LONG bTrigFlag;        // 触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    LONG bConverting;      // DA 是否正在转换，  =TRUE:表示正在转换，  = FALS 表示转换完成
    LONG nCurSegNum;       // 可读取的 RAM 段号，取值为[0, SegmentCount-1], (注 SegmentCount 为 InitDeviceDA 函数的参数)
    LONG nCurSegAddr;      // 可读取的 RAM 段地址
    LONG nCurLoopCount; // 当前总循环次数
    LONG nCurSegLoopCount; // 当前段循环次数
} PCI8603_STATUS_DA, *PPCI8603_STATUS_DA;
```

Visual Basic：

```
Type PCI8603_ STATUS _DA
    bEnable As Long          ' DA 使能启动标志，＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止
    bTrigFlag As Long          '触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    bConverting As Long        ' DA 是否正在转换，  =TRUE:表示正在转换，  = FALS 表示转换完成
    nCurSegNum As Long         ' 可读取的 RAM 段号
    nCurSegAddr As Long        ' 可读取的 RAM 段地址
    nCurLoopCount As Long             ' 当前总循环次数
```

```
        nCurSegLoopCount As Long              ' 当前段循环次数
    End Type
```

Delphi:

```
Type   // 定义结构体数据类型
PPCI8603_ STATUS _DA = ^ PCI8603_ STATUS _DA;  // 指针类型结构
PCI8603_ STATUS _DA = record          // 标记为记录型
    bEnable : LongInt;        // DA 使能启动标志, ＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止
    bTrigFlag : LongInt;          //触发标志是否有效，=TRUE 表示触点标有效， = FALSE 表示无效（即触发点未到）
    bConverting : LongInt;          // DA 是否正在转换， =TRUE:表示正在转换， = FALS 表示转换完成
    nCurSegNum : LongInt;       //可读取的 RAM 段号
    nCurSegAddr : LongInt;       //可读取的 RAM 段地址
    nCurLoopCount: LongInt;     // 当前总循环次数
    nCurSegLoopCount: LongInt;      // 当前段循环次数
End;
```

LabVIEW:

请参考相关演示程序。

bEnable DA 使能启动标志, ＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止。

bTrigFlag 触发标志是否有效，=TRUE 表示触点标有效，= FALSE 表示无效（即触发点未到）。

bConverting DA 是否正在转换，=TRUE 表示正在转换，= FALS 表示转换完成。

nCurSegNum 可读取的 RAM 段号，取值为[0, SegmentCount-1], (注 SegmentCount 为 InitDeviceDA 函数的参数)。

nCurSegAddr 可读取的 RAM 段地址。

nCurLoopCount 当前总循环次数。

nCurSegLoopCount 当前段循环次数。

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]
±2500mV	Volt = (5000.00/4096) * (ADBuffer[0] &0x0FFF) – 2500.00	[-2500, +2498.77]
0～10V	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++&C++Builder：

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

Visual Basic：

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

Delphi:

```
Lsb:   = ADBuffer[0] And $0FFF;
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，当通道总数首末通道相等时，假如此时首末通道=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：

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：

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;
    // 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;                    // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;              // 头信息版本(D31-D16=Major   D15-D0=Minijor) = 1.0
    LONG VoltBottomRange;          // 量程下限(mV)
    LONG VoltTopRange;             // 量程上限(mV)
    PCI8603_PARA_AD ADPara;        // 保存硬件参数

    LONGLONG nTriggerPos;          // 触发点位置
    LONG BatCode;                  // 同批文件识别码
    LONG HeadEndFlag;              // 文件结束位
} 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 的电压值如何转换成输出到 DA 转换器的 LSB 原码数据？

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

请注意这里求得的LSB数据就是用于WriteDeviceBulkDA中的DABuffer[]参数的。

第五节、关于 DA 数据 DABuffer 缓冲区中的数据排放规则

由于各个通道的段信息与波形数据均共享一个板载物理 RAM，它们的排放顺序如图，系统默认值为两个通道均分整个 RAM 空间，即默认每通道 RAM 空间为 256K 点。从下图可以看出，各个通道所占 RAM 空间不一定相等，可大可小，只是两个通道的总空间不能大于板载物理 RAM 空间即可。

通道 0	通道 1
0 至(256K-1)	256 至(512K-1)

关于每个通道 RAM 空间的内部分配是这样的，其空间首部存放的是所有段的段信息数据，其后才是各个段的波形数据，再其后可能还有未用空间。假如有三个分段，如图：

段 0 信息	段 1 信息	段 2 信息	段 0 波形数据	段 1 波形数据	段 2 波形数据	未用空间

关于每个段的段信息包括的内容有：该段波形数据在 RAM 中的起始地址、终止地址、段循环次数，如下表，注意其段起始地址和终止地址是由 PCI8603_PARA_DA 中的 SegmentInfo 决定的。

板载 RAM 内存单元(16Bit)	各单元定义	有效位
0	段 0 波形数据起始地址低 12 位	D11:D0
1	段 0 波形数据起始地址高 8 位	D7:D0
2	段 0 波形数据终止地址低 12 位	D11:D0
3	段 0 波形数据终止地址高 8 位	D7:D0
4	段 0 循环次数低 12 位	D11:D0

5	段 0 循环次数高 8 位	D7:D0
6	段 1 波形数据起始地址低 12 位	D11:D0
7	段 1 波形数据起始地址高 8 位	D7:D0
8	段 1 波形数据终止地址低 12 位	D11:D0
9	段 1 波形数据终止地址高 8 位	D7:D0
10	段 1 循环次数低 12 位	D11:D0
11	段 1 循环次数高 8 位	D7:D0
:	:	:
段信息结束后便是波形数据	段信息结束后便是波形数据	D11:D0

第六章 上层用户函数接口应用实例

第一节、怎样使用ReadDeviceProAD_Npt函数直接取得AD数据

Visual C++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**╯**[Microsoft Visual C++]**╯[简易代码演示]╯**[AD 非空方式]**

第二节、怎样使用ReadDeviceProAD_Half函数直接取得AD数据

Visual C++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**╯**[Microsoft Visual C++]**╯[简易代码演示]╯**[AD 半满方式]**

第三节、怎样使用ReadDeviceDmaAD函数直接取得AD数据

Visual C++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**╯**[Microsoft Visual C++]**╯[简易代码演示]╯**[AD DMA 直接内存方式]**

第四节、怎样使用WriteDeviceBulkDA函数取得DA数据

Visual C++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**╯**[Microsoft Visual C++]**╯[简易代码演示]╯**[DA 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++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 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++ & C++Builder:

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]╯**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[DIO…]**

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ReadDeviceProAD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_Npt 、ReadDeviceProAD_Half (或者ReadDeviceDmaAD)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InitDeviceAD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已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。

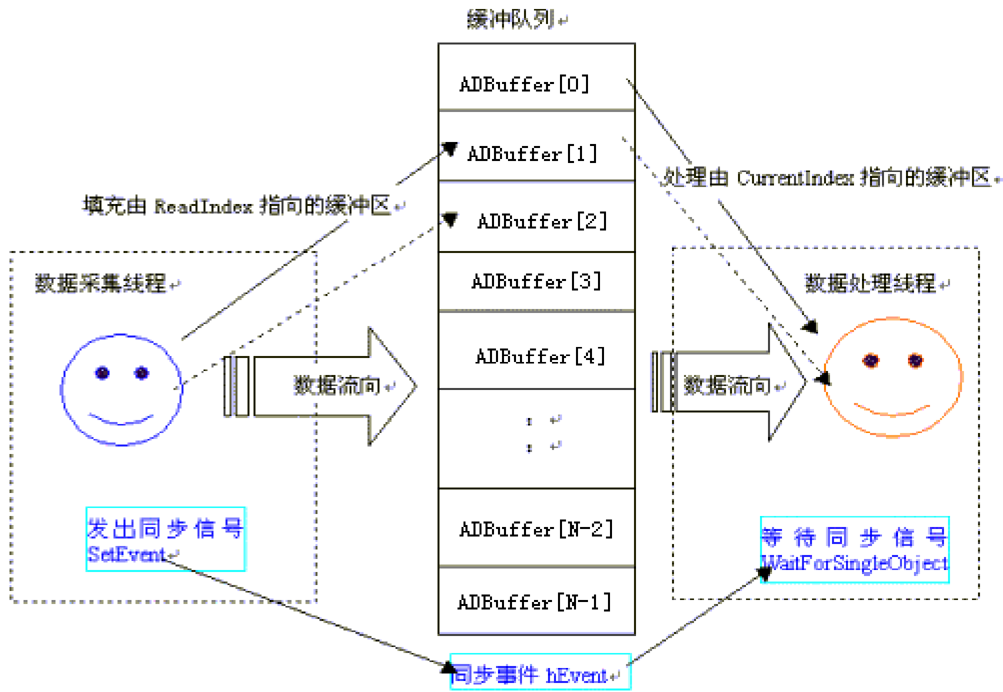


图 7.1

第一节、使用程序查询方式实现该功能

一、使用ReadDeviceProAD_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]丨[阿尔泰测控演示系统]丨**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**丨**[Microsoft Visual C++]**丨[高级演示程序]

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ReadDeviceProAD_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]丨[阿尔泰测控演示系统]丨**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 卡]**丨**[Microsoft Visual C++]**丨[高级演示程序]

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 DMA 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]」**[PCI8603 16 路 AD 2 路 DA 和 8 路 DIO 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 () // 启动采集线程函数
UINT ReadDataThread () // 采集线程函数
BOOL StopDeviceAD () // 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“PCI8603_”）

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
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	
④ **文件对象操作函数**		
CreateFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	
ReadFile	请求文件对象读数据到用户空间	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	

ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备
⑤ 各种参数保存和读取函数		
SaveParaInt	保存整型参数到注册表	
LoadParaInt	从注册表中读取整型参数值	
SaveParaString	保存字符参数到注册表	
LoadParaString	从注册表中读取字符参数值	
⑥ 其他函数		
kbhit	探测用户是否有击键动作	
getch	等待并获取用户击键值	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间
GetLastErrorEx	取得驱动函数错误信息	
RemoveLastErrorEx	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++ & C++ Builder:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PULONG MemCPLDBase,
                    PULONG MemADBuffer,
                    PULONG MemDA0Buffer,
                    PULONG MemDA1Buffer,
                    int RegisterID = 0)
```

Visual Basic:.

```
Declare Function GetDeviceAddr Lib "PCI8603" (ByVal hDevice as Long, _
                                    ByRef MemCPLDBase As Long, _
                                    ByRef MemADBuffer As Long, _
                                    ByRef MemDA0Buffer As Long, _
                                    ByRef MemDA1Buffer As Long, _
                                    Optional ByVal RegisterID As Integer = 0) As Boolean
```

Delphi:

```
Function GetDeviceAddr(hDevice : Integer;
                    MemCPLDBase : Pointer;
                    MemADBuffer : Pointer;
                    MemDA0Buffer: Pointer;
                    MemDA1Buffer: Pointer;
                    RegisterID:Integer = 0):Boolean;
                    StdCall; External ' PCI8603 ' Name ' GetDeviceAddr ';
```

LabVIEW:

请参考相关演示程序。

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或CreateDeviceEx创建。

MemCPLDBase 返回指定映射寄存器的线性地址。

MemADBuffer 返回指定映射寄存器的线性地址。

MemDA0Buffer 返回指定映射寄存器的线性地址。

MemDA1Buffer 返回指定映射寄存器的线性地址。

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。

常量名	常量值	功能定义
PCI8603_REG_RAM_CPLD	0x0000	1 号寄存器对应配置所有相关寄存器使用(使用 LinearAddr)

PCI8603_REG_RAM_ADBUFFER	0x0001	2 号寄存器对应板上控制单元所使用的内存模式基地址(使用 LinearAddr)
PCI8603_REG_RAM_DA0BUFFER	0x0002	3 号寄存器对应板上 DA 缓冲区所使用的内存模式基地址(使用 LinearAddr)
PCI8603_REG_RAM_DA1BUFFER	0x0003	3 号寄存器对应板上 DA 缓冲区所使用的内存模式基地址(使用 LinearAddr)

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

◆ 取得指定的指定设备寄存器组 **BAR** 地址

函数原型：

Visual C++ & C++ Builder:

```
BOOL GetDeviceBar ( HANDLE hDevice,
                    ULONG   pulPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib "PCI8603" (ByVal hDevice As Long, _
                                   ByVal pulPCIBar (0 to 5) As Long) As Boolean
```

Delphi:

```
Function GetDeviceBar (hDevice : Integer;
                       pulPCIBar : Pointer) : Boolean;
                       StdCall; External 'PCI8603' Name ' GetDeviceBar';
```

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或CreateDeviceEx创建。

pulPCIBar 返回 PCI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　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++ & C++ Builder:

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

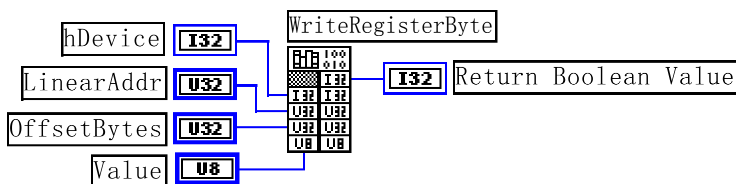
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI8603" (ByVal hDevice As Long, _
                                        ByVal LinearAddr As Long, _
                                        ByVal OffsetBytes As Long, _
                                        ByVal Value As Byte ) As Boolean
```

Delphi:

```
Function WriteRegisterByte(  hDevice : Integer;
                             LinearAddr : LongWord;
                             OffsetBytes : LongWord;
                             Value : Byte) : Boolean;
                             StdCall; External 'PCI8603' Name ' WriteRegisterByte ';
```

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或CreateDeviceEx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++ & C++ Builder 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
    :
```

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
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

♦ **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++ & C++ Builder:

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

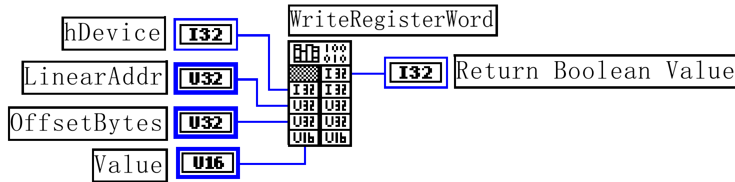
Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI8603" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WriteRegisterWord( hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : Word) : Boolean;
                            StdCall; External 'PCI8603' Name ' WriteRegisterWord ';
```

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或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:

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

Visual Basic:

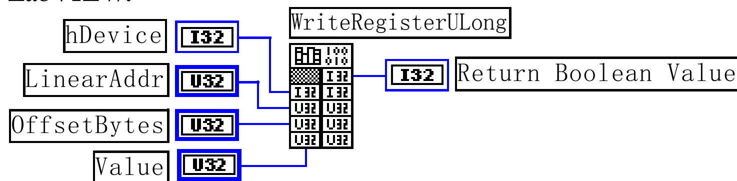
```
Declare Function WriteRegisterULong Lib "PCI8603" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long) As Boolean
```

Delphi:

```
Function WriteRegisterULong(hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : LongWord) : Boolean;
```

StdCall; External 'PCI8603' Name ' WriteRegisterULong ';

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或CreateDeviceEx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++ & C++ Builder 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
        :
```

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
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
        :
```

◆ **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++ & C++ Builder:

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

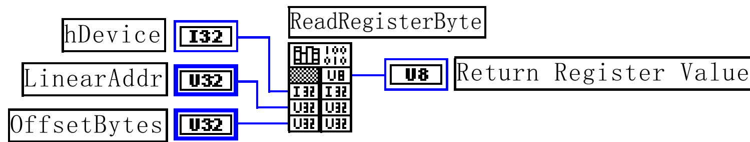
Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI8603" (ByVal hDevice As Long, _
                                                 ByVal LinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

Delphi:

```
Function ReadRegisterByte(hDevice : Integer;
                          LinearAddr : LongWord;
                          OffsetBytes : LongWord) : Byte;
                          StdCall; External 'PCI8603' Name ' ReadRegisterByte ';
```

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或CreateDeviceEx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++ & C++ Builder 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++ & C++ Builder:

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

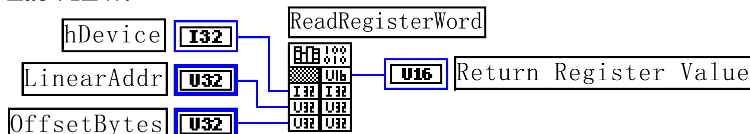
Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI8603" (  ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long) As Integer
```

Delphi:

```
Function ReadRegisteWord(hDevice : Integer;
                        LinearAddr : LongWord;
                        OffsetBytes : LongWord) : Word;
                        StdCall; External 'PCI8603' Name ' ReadRegisterWord ';
```

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或CreateDeviceEx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++ & C++ Builder 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
:
```

Visual Basic 程序举例:

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

◆ **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++ & C++ Builder:

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

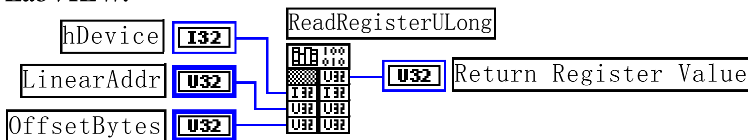
Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI8603" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```

Delphi:

```
Function ReadRegisterULong(hDevice : Integer;
                           LinearAddr : LongWord;
                           OffsetBytes : LongWord) : LongWord;
                           StdCall; External 'PCI8603' Name ' ReadRegisterULong ';
```

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或CreateDeviceEx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++ & C++ Builder 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、IO 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++ & C++ Builder:

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

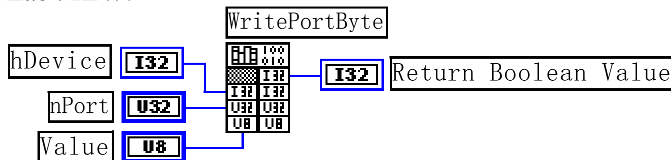
Visual Basic:

```
Declare Function WritePortByte Lib "PCI8603" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Byte) As Boolean
```

Delphi:

```
Function WritePortByte(hDevice : Integer;
                       nPort : LongWord;
                       Value : Byte) : Boolean;
                       StdCall; External 'PCI8603' Name ' WritePortByte ';
```

LabVIEW:



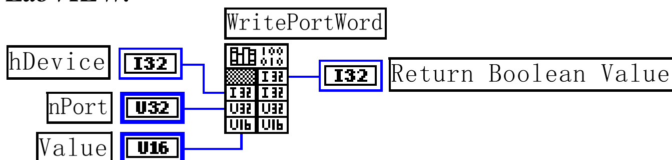
功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

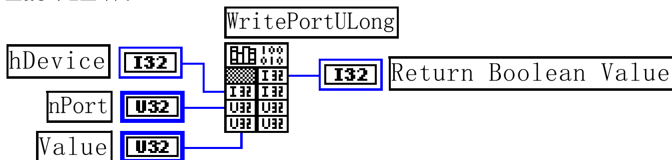
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

◆ 以双字(16Bit)方式写 I/O 端口
函数原型：
Visual C++ & C++ Builder:
```
BOOL WritePortWord (HANDLE hDevice,
                    UINT nPort,
                    WORD Value)
```
Visual Basic:
```
Declare Function WritePortWord Lib "PCI8603" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```
Delphi:
```
Function WritePortWord(hDevice : Integer;
                    nPort : LongWord;
                    Value : Word) : Boolean;
                    StdCall; External 'PCI8603' Name ' WritePortWord ';
```
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或CreateDeviceEx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

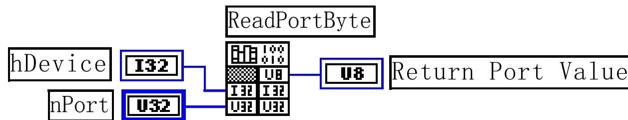
◆ 以四字节(32Bit)方式写 I/O 端口
函数原型：
Visual C++ & C++ Builder:
```
BOOL WritePortULong(HANDLE hDevice,
                    UINT nPort,
                    ULONG Value)
```
Visual Basic:
```
Declare Function WritePortULong Lib "PCI8603" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```
Delphi:
```
Function WritePortULong(hDevice : Integer;
                    nPort : LongWord;
                    Value : LongWord) : Boolean;
                    StdCall; External 'PCI8603' Name ' WritePortULong ';
```
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或CreateDeviceEx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**

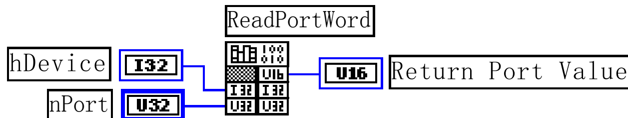
函数原型：
Visual C++ & C++ Builder:
```
BYTE ReadPortByte( HANDLE hDevice,
                   UINT nPort)
```
Visual Basic:
```
Declare Function ReadPortByte Lib "PCI8603" (ByVal hDevice As Long, _
                                   ByVal nPort As Long ) As Byte
```
Delphi:
```
Function ReadPortByte(hDevice : Integer;
                   nPort : LongWord) : Byte;
                   StdCall; External 'PCI8603' Name ' ReadPortByte ';
```
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或CreateDeviceEx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++ & C++ Builder:
```
WORD ReadPortWord(HANDLE hDevice,
                   UINT nPort)
```
Visual Basic:
```
Declare Function ReadPortWord Lib "PCI8603" ( ByVal hDevice As Long, _
                                   ByVal nPort As Long ) As Integer
```
Delphi:
```
Function ReadPortWord(hDevice : Integer;
                   nPort : LongWord) : Word;
                   StdCall; External 'PCI8603' Name ' ReadPortWord ';
```
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或CreateDeviceEx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++ & C++ Builder:

```
ULONG ReadPortULong(HANDLE hDevice,
                    UINT nPort)
```

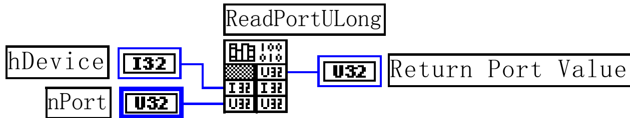
Visual Basic:

```
Declare Function ReadPortULong Lib "PCI8603" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                       nPort : LongWord) : LongWord;
                       StdCall; External 'PCI8603' Name ' ReadPortULong ';
```

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或CreateDeviceEx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

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

◆ 创建内核系统事件

函数原型：

Visual C++ & C++ Builder:

```
HANDLE CreateSystemEvent(void)
```

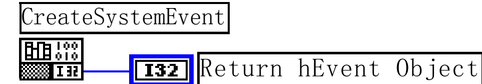
Visual Basic:

```
Declare Function CreateSystemEvent Lib " PCI8603 " () As Long
```

Delphi:

```
Function CreateSystemEvent() : Integer;
                       StdCall; External 'PCI8603' Name ' CreateSystemEvent ';
```

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++ & C++ Builder:

```
BOOL ReleaseSystemEvent(HANDLE hEvent)
```

Visual Basic:

```
Declare Function ReleaseSystemEvent Lib " PCI8603 " (ByVal hEvent As Long) As Boolean
```

Delphi:

```
Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
                    StdCall; External 'PCI8603' Name ' ReleaseSystemEvent ';
```

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++ & C++ Builder:

```
HANDLE CreateFileObject ( HANDLE hDevice,
                          LPCTSTR strFileName,
                          int Mode)
```

Visual Basic:

```
Declare Function CreateFileObject Lib "PCI8603" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer;
                          strFileName : String;
                          Mode : Integer) : Integer;
                          Stdcall; external 'PCI8603' Name ' CreateFileObject ';
```

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或CreateDeviceEx创建。
strFileName 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
PCI8603_modeRead	0x0000	只读文件方式
PCI8603_modeWrite	0x0001	只写文件方式
PCI8603_modeReadWrite	0x0002	既读又写文件方式
PCI8603_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
PCI8603_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++ & C++ Builder:

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

Visual Basic:

```
Declare Function WriteFile Lib "PCI8603" ( ByVal hFileObject As Long,_
                                          ByRef pDataBuffer As Integer,_
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongInt) : Boolean;
                   Stdcall; external 'PCI8603' Name ' WriteFile ';
```

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++ & C++ Builder:

```
BOOL ReadFile( HANDLE hFileObject,
               PVOID pDataBuffer,
               LONG nOffsetBytes,
               LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PCI8603" ( ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal nOffsetBytes As Long, _
                                          ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile(hFileObject : Integer;
                  pDataBuffer : Pointer;
                  nOffsetBytes : LongInt;
                  nReadSizeBytes : LongInt) : Boolean;
                  Stdcall; external 'PCI8603' Name ' ReadFile ';
```

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。

nOffsetBytes 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++ & C++ Builder:

```
BOOL SetFileOffset (HANDLE hFileObject,
                    LONG nOffsetBytes)
```

Visual Basic:

```
Declare Function SetFileOffset Lib "PCI8603" ( ByVal hFileObject As Long,_
                                               ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset ( hFileObject : Integer;
                                   nOffsetBytes : LongInt) : Boolean;
                                   Stdcall; external 'PCI8603' Name ' SetFileOffset ';
```

LabVIEW:

详见相关演示程序。

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++ & C++ Builder:

```
ULONG GetFileLength (HANDLE hFileObject)
```

Visual Basic:

```
Declare Function GetFileLength Lib "PCI8603" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject : Integer) : LongWord;
                                   Stdcall; external 'PCI8603' Name ' GetFileLength ';
```

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++ & C++ Builder:

```
BOOL ReleaseFile(HANDLE hFileObject)
```

Visual Basic:

```
Declare Function ReleaseFile Lib "PCI8603" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
Function ReleaseFile(hFileObject : Integer) : Boolean;
                                   Stdcall; external 'PCI8603' Name ' ReleaseFile ';
```

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++ & C++ Builder:

```
ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName)
```

Visual Basic:

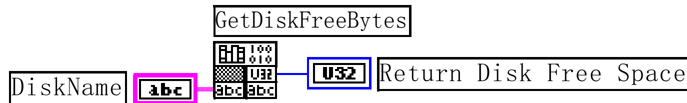
```
Declare Function GetDiskFreeBytes Lib "PCI8603" (ByVal strDiskName As String ) As Currency
```

Delphi:

```
Function GetDiskFreeBytes (strDiskName : String) : Currency;
```

Stdcall; external 'PCI8603' Name ' GetDiskFreeBytes ';

LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：szDiskName 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、各种参数保存和读取函数原型说明

◆ 将整型变量的参数值保存在系统注册表中

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaInt( HANDLE hDevice,
                  LPCTSTR strParaName,
                  int nValue)
```

Visual Basic:

```
Declare Function SaveParaInt Lib "PCI8603" (ByVal hDevice As Long,_
                                   ByVal strParaName As String,_
                                   ByVal nValue As Integer) As Boolean
```

Delphi:

```
Function SaveParaInt( hDevice : Integer;
                      strParaName : String;
                      nValue : Integer) : Boolean;
                      Stdcall; external 'PCI8603' Name ' SaveParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8603\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nValue 整型参数值。它保存在由 strParaName 命名的键项里。

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： SaveParaInt LoadParaInt SaveParaString LoadParaString

◆ 将整型变量的参数值从系统注册表中读出

函数原型：

Visual C++ & C++ Builder:

```
UINT LoadParaInt(HANDLE hDevice,
                 LPCTSTR strParaName,
                 int nDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaInt Lib "PCI8603" (ByVal hDevice As Long,_
                                   ByVal strParaName As String,_
                                   ByVal nDefaultVal As Integer) As Long
```

Delphi:

```
Function LoadParaInt ( hDevice : Integer;
                       strParaName : String;
                       nDefaultVal: Integer) : Longword;
                       Stdcall; external 'PCI8603' Name ' LoadParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8603\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

返回值：若指定的整型参数项存在，则返回其整型值。否则返回由 nDefaultVal 指定的默认值。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

♦将字符变量的参数值保存在系统注册表中

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaString ( HANDLE hDevice,
                      LPCTSTR strParaName,
                      LPCTSTR strParaVal)
```

Visual Basic:

```
Declare Function SaveParaString Lib "PCI8603" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String) As Boolean
```

Delphi:

```
Function SaveParaString (hDevice : Integer;
                         strParaName : String;
                         strParaVal: String) : Boolean;
                         Stdcall; external 'PCI8603' Name ' SaveParaString';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8603\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 字符参数值。它保存在由 strParaName 命名的键项里。

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： SaveParaInt LoadParaInt SaveParaString LoadParaString

♦将字符变量的参数值从系统注册表中读出

函数原型：

Visual C++ & C++ Builder:

```
BOOL LoadParaString ( HANDLE hDevice,
                      LPCTSTR strParaName,
                      LPCTSTR strParaVal,
                      LPCTSTR strDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaString Lib "PCI8603" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String,_
                                               ByVal strDefaultVal As String) As Boolean
```

Delphi:

```
Function LoadParaString ( hDevice : Integer;
                          strParaName : String;
                          strParaVal : String;
                          strDefaultVal : String) : Boolean;
                          Stdcall; external 'PCI8603' Name ' LoadParaString ';
```

LabVIEW:
详见相关演示程序。

功能：将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\PCI8603\Device-0\Others。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。
strParaVal 取得 strParaName 指定的键项的字符值。
strDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。
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： SaveParaInt LoadParaInt SaveParaString LoadParaString

第六节、其他函数原型说明

◆ 探测用户是否有按键动作

函数原型：
Visual C++ & C++ Builder:
```
BOOL kbhit (void)
```
Visual Basic:
```
Declare Function kbhit Lib "PCI8603" () As Boolean
```
Delphi:
```
Function kbhit () : Boolean;
            Stdcall; external 'PCI8603' Name ' kbhit ';
```
LabVIEW:
详见相关演示程序。

功能：探测用户是否用键盘按键动作，主要应在基于 VB、DELPHI 等控制台应用程序中。
参数：无。
返回值：若自上次探测过后，若用户有键盘按键动作，则返回 TRUE，否则返回 FALSE。
相关函数： getch kbhit

◆ 等待按键动作并返回按键值

函数原型：
Visual C++ & C++ Builder:
```
char getch (void)
```
Visual Basic:
```
Declare Function getch Lib "PCI8603" () As String
```
Delphi:
```
Function getch () : char;
            Stdcall; external 'PCI8603' Name 'getch';
```
LabVIEW:
详见相关演示程序。

功能：探等待用户键盘按键并以字符方式返回按键值，主要应在基于 VB、DELPHI 等控制台应用程序中。
参数：无。
返回值：若用户没有按键动作，此函数一直不返回，一旦用户有按键动作，便立即返回，且返回其当前按键值(ACII 码)。
相关函数： getch kbhit

◆ 高效高精度延时

函数原型：
Visual C++ & C++ Builder:
```
BOOL DelayTimeUs (HANDLE hDevice,
                  LONG nTimeUs)
```
Visual Basic:

Declare Function DelayTimeUs Lib "PCI8603" (ByVal hDevice As Long, _
ByVal nTimeUs As Long) As Boolean

Delphi:

Function DelayTimeUs (hDevice : Integer;
nTimeUs : LongInt) : Boolean;
StdCall; External 'PCI8603' Name ' DelayTimeUs ';

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或CreateDeviceEx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。

♦怎样获取驱动函数错误信息

函数原型：

Visual C++ & C++ Builder:

DWORD GetLastErrorEx (LPCTSTR strFuncName,
LPTSTR strErrorMsg)

Visual Basic:

Declare Function GetLastErrorEx Lib "PCI8603" (ByVal strFuncName As String,_
ByVal strErrorMsg As String) As Long

Delphi:

Function GetLastErrorEx (strFuncName: String;
strErrorMsg: String) : LongWord;
Stdcall; external 'PCI8603' Name ' GetLastErrorEx ';

LabVIEW:

详见相关演示程序。

功能：将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字串。

参数：

strFuncName 出错函数的名称。注意此函数必须是完整名称，如 AD 初始化函数 PCI8603_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PCI8603_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串。

返回值：返回错误码。

相关函数： 无。

Visual C++ & C++Builder 程序举例

```
    :
char strErrorMsg[256];   // 用于返回错误信息字串，要求其空间足够大
DWORD dwErrorCode;
int DeviceLgcID = 0;
hDevice = PCI8603_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
 {
    dwErrorCode = PCI8603_GetLastErrorEx("PCI8603_CreateDevice", strErrorMsg);
    AfxMessageBox(strErrorMsg); // 以对话框方式显示错误信息
    return;   // 退出该函数
 }
   :
```

Visual Basic 程序举例

```
   :
Dim strErrormsg As String  ' 用于返回错误信息字串，要求其空间足够大
Dim dwErrorCode As Long
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI8603_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
```

```
        dwErrorCode = PCI8603_GetLastErrorEx("PCI8603_CreateDevice", strErrorMsg)
        MsgBox strErrorMsg   ' 以对话框方式显示错误信息
        Exit Sub      ' 退出该过程
    End If
        :
```

♦ 移除驱动函数错误信息

函数原型：

Visual C++ & C++ Builder:

BOOL RemoveLastErrorEx (LPCTSTR strFuncName)

Visual Basic:

Declare Function RemoveLastErrorEx Lib "PCI8603" (ByVal strFuncName As String) As Boolean

Delphi:

Function RemoveLastErrorEx (strFuncName: String) : Boolean;
Stdcall; external 'PCI8603' Name ' RemoveLastErrorEx';

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 出错函数的名称。注意此函数必须是完整名称，如 AD 初始化函数 PCI8603_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为"PCI8603_InitDeviceAD"，否则得不到相应信息。

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：无。