PCI2510 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_ 则被省略。如 PCI2510_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写规则

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

以上规则不局限于该产品。

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 InitDeviceDI 、 InitDeviceDmaDI 、 ReadDeviceProDI_Npt 等。而底层用户函数如 WriteRegisterULong 、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上插座等管脚分配情况。

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 InitDeviceDI可以使用hDevice句柄以程序查询方式初始化设备的DI部件，ReadDeviceProDI_Npt (或 ReadDeviceProDI_Half)函数可以用hDevice句柄实现对DI数据的采样读取等。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 DI 数据

当您有了hDevice设备对象句柄后，便可用 InitDeviceDI函数初始化DI部件。您只需要对这个pDIPara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 StartDeviceDI即可启动DI部件，

开始DI采样，然后便可用 ReadDeviceProDI_Npt反复读取DI数据以实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceDI，当您需要关闭DI设备时，ReleaseDevice便可帮您实现。（注：ReadDeviceProDI_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DI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 DI 数据

当您有了hDevice设备对象句柄后，便可用 InitDeviceDI函数初始化DI部件。您只需要对这个pDIPara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 StartDeviceDI即可启动DI部件，开始DI采样，接着调用 GetDevStatusDI函数以查询DI的存储器FIFO的半满状态，如果达到半满状态，即可用ReadDeviceProDI_Half函数读取一批半满长度（或半满以下）的DI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DI数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceDI，当您需要关闭DI设备时，ReleaseDevice便可帮您实现。（注：ReadDeviceProDI_Half函数在半满状态有效时也可以单点或几点的方式读取DI数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）具体执行流程请看下面的图 2.1.2。

第五节、如何用 Dma 直接内存方式取得 DI 数据

当您有了hDevice设备对象句柄后，便可用 InitDeviceDmaDI函数初始化DI部件，关于采样通道、频率等的参数的设置是由这个函数的pDIPara参数结构体决定的。您只需要对这个pDIPara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 InitDeviceDmaDI的相应参数，它将作为Dma事件的变量。然后用 StartDeviceDmaDI即可启动DI部件，开始DI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 GetDevStatusDmaDI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DI数据即可实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceDmaDI，当您需要关闭DI设备时，ReleaseDeviceDmaDI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。

第六节、如何用查询方式取得 DO 数据

当您有了hDevice设备对象句柄后，便可用 InitDeviceDO函数初始化DO部件。您只需要对这个pDOPara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 StartDeviceDO即可启动DO部件，开始DO采样，然后便可用 WriteDeviceProDO反复读取DO数据以实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceDO，当您需要关闭DO设备时，ReleaseDevice便可帮您实现。具体执行流程请看下面的图 2.1.4。

第七节、如何用 Dma 直接内存方式取得 DO 数据

当您有了hDevice设备对象句柄后，便可用 InitDeviceDmaDO函数初始化DO部件，关于采样通道、频率等的参数的设置是由这个函数的pDOPara参数结构体决定的。您只需要对这个pDOPara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 InitDeviceDmaDO的相应参数，它将作为Dma事件的变量。然后用 StartDeviceDmaDO即可启动DO部件，开始DO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 GetDevStatusDmaDO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DO数据即可实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceDmaDO，当您需要关闭DO设备时，ReleaseDeviceDmaDO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 2.1.5。

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。

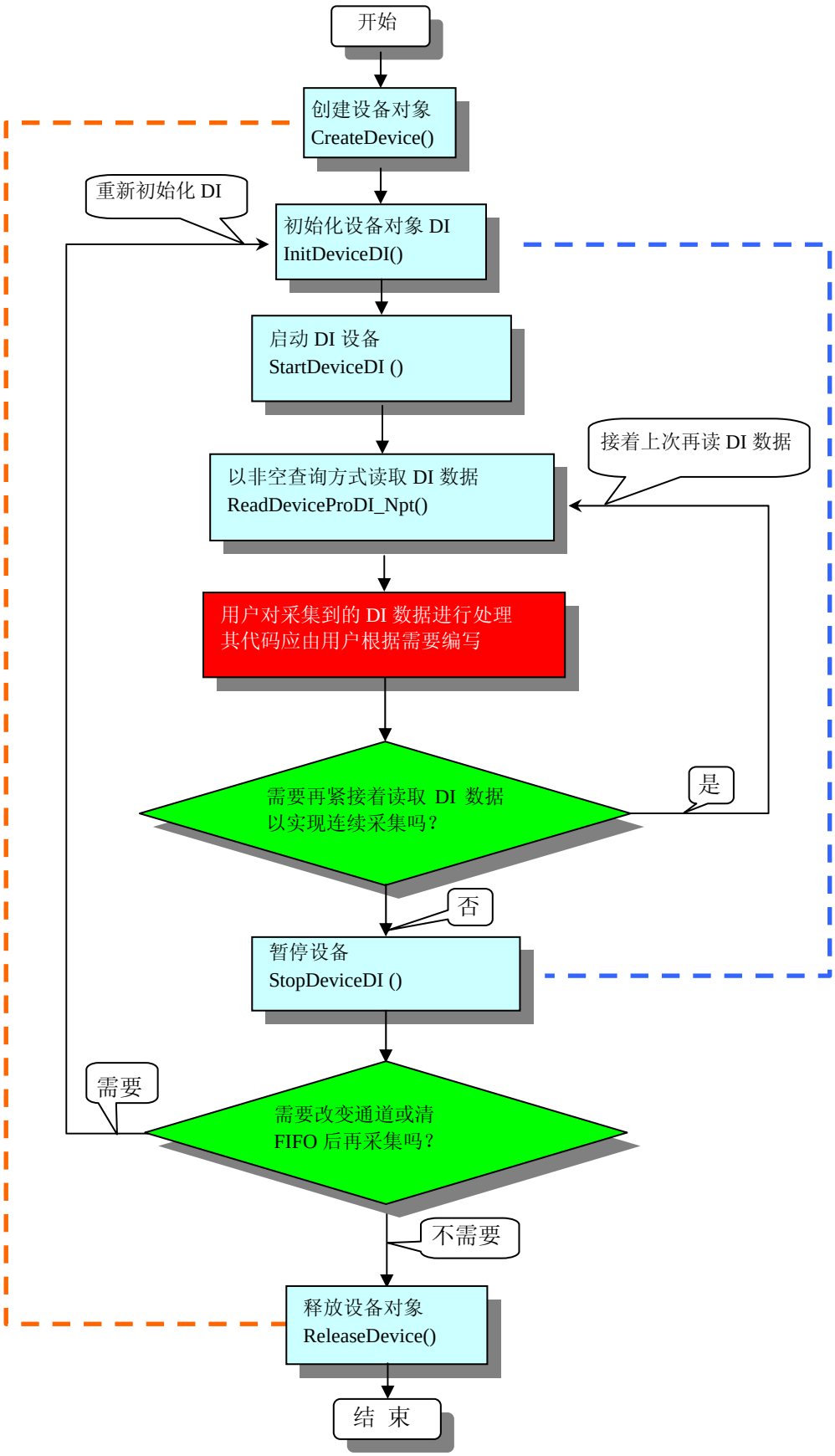


图 2.1.1 非空查询方式 DI 采集过程

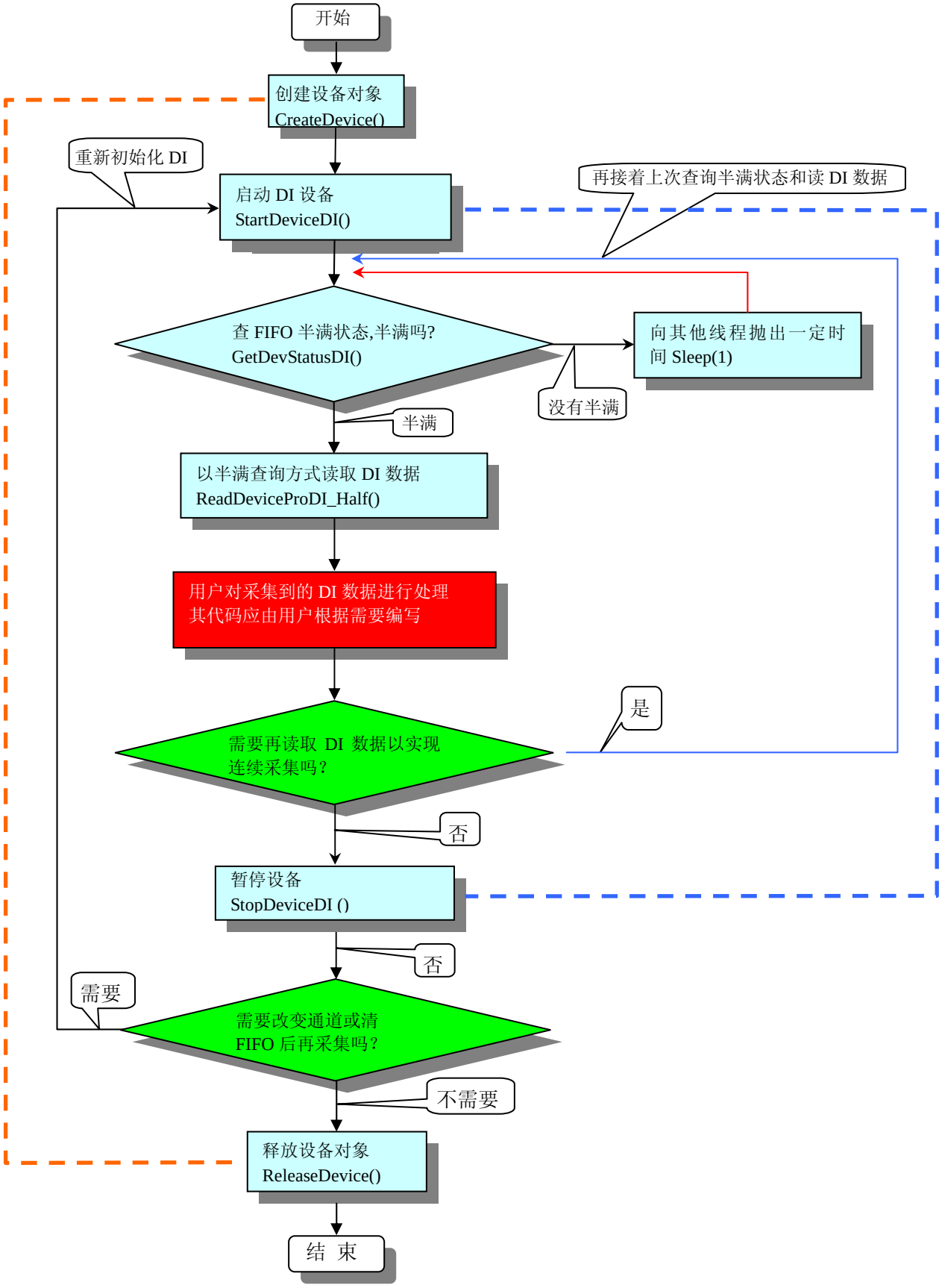


图 2.1.2 半满查询方式 DI 采集过程

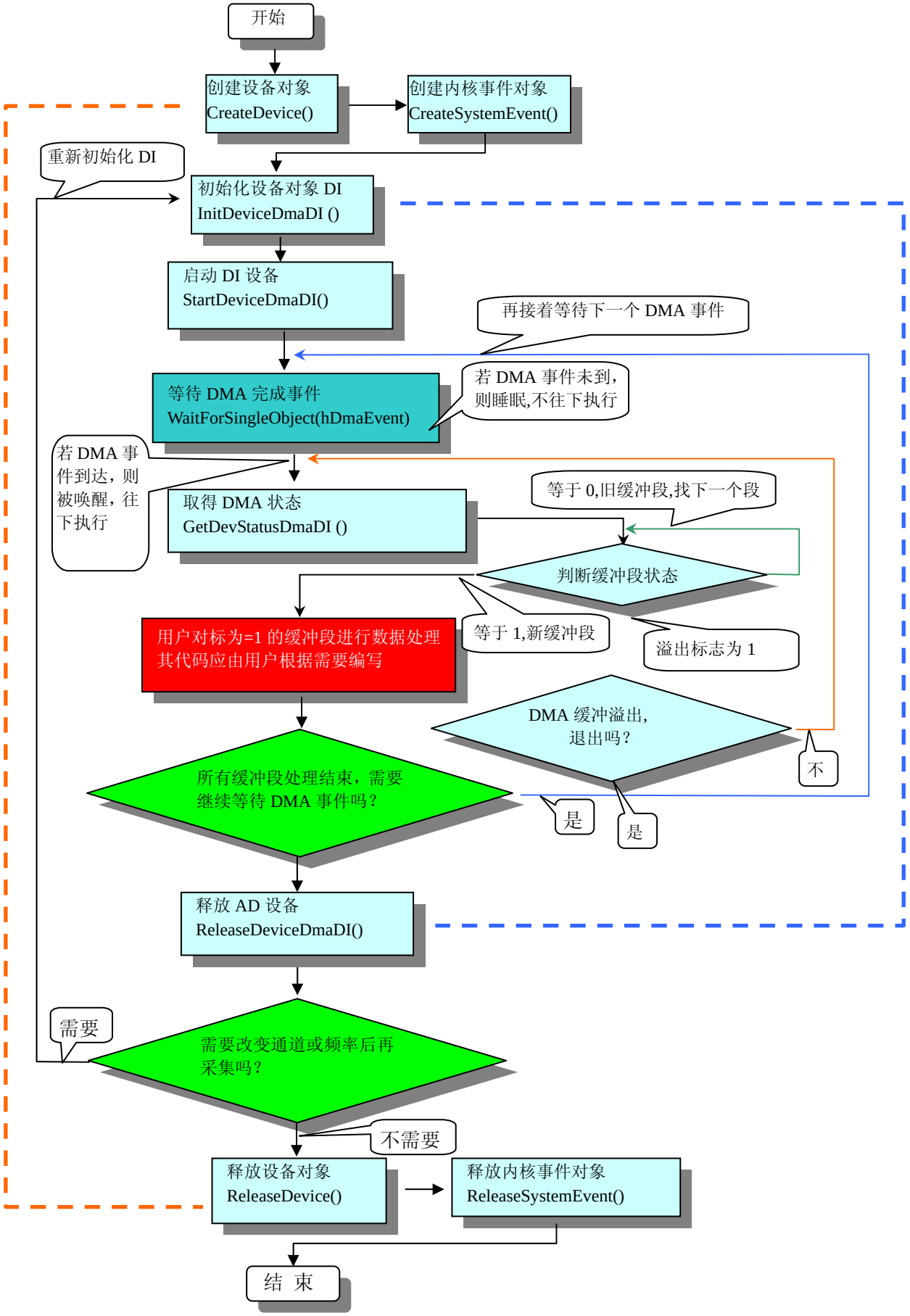


图 2.1.3 DMA 方式 DI 采集实现过程

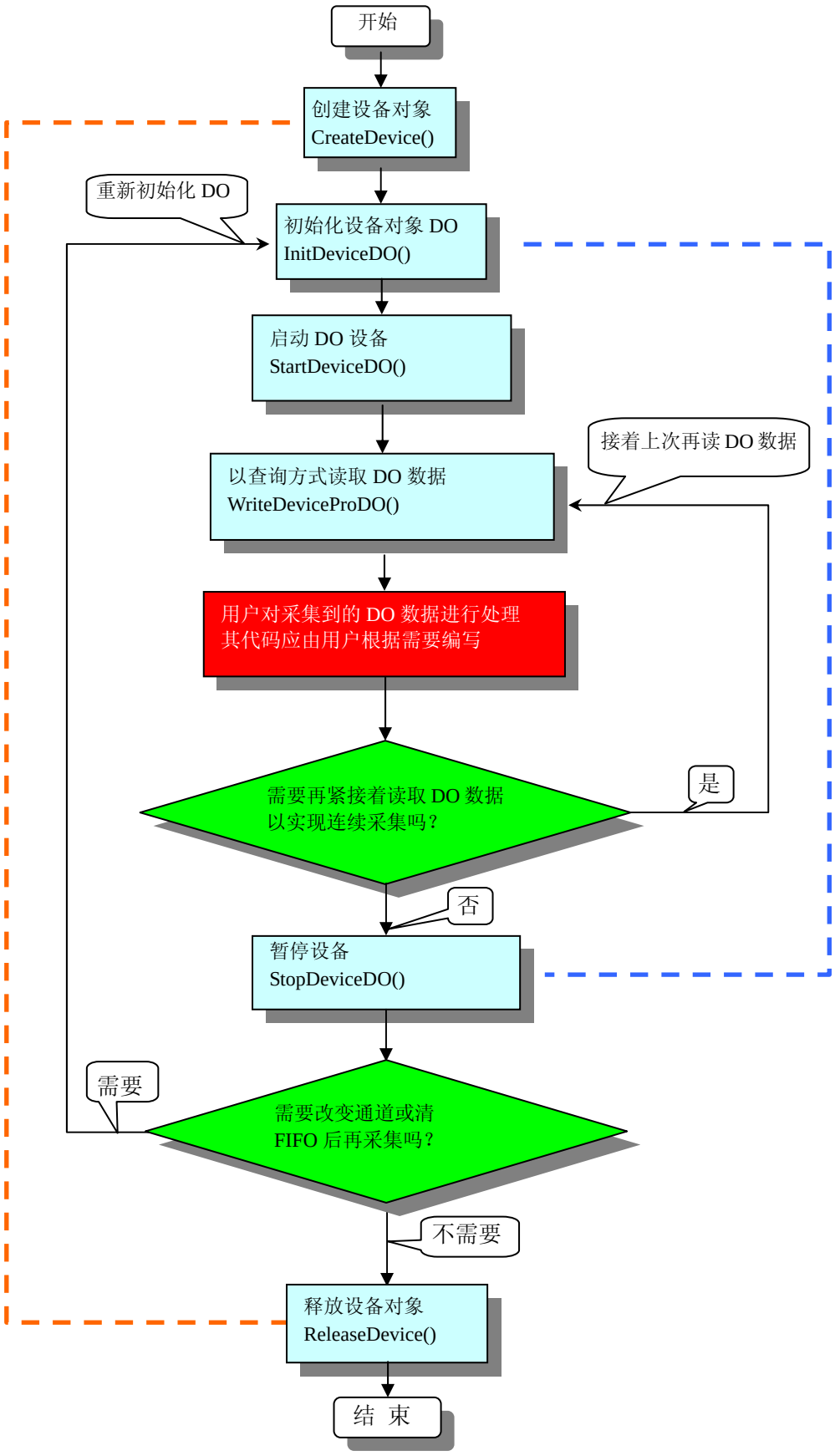


图 2.1.4 查询方式 DO 采集过程

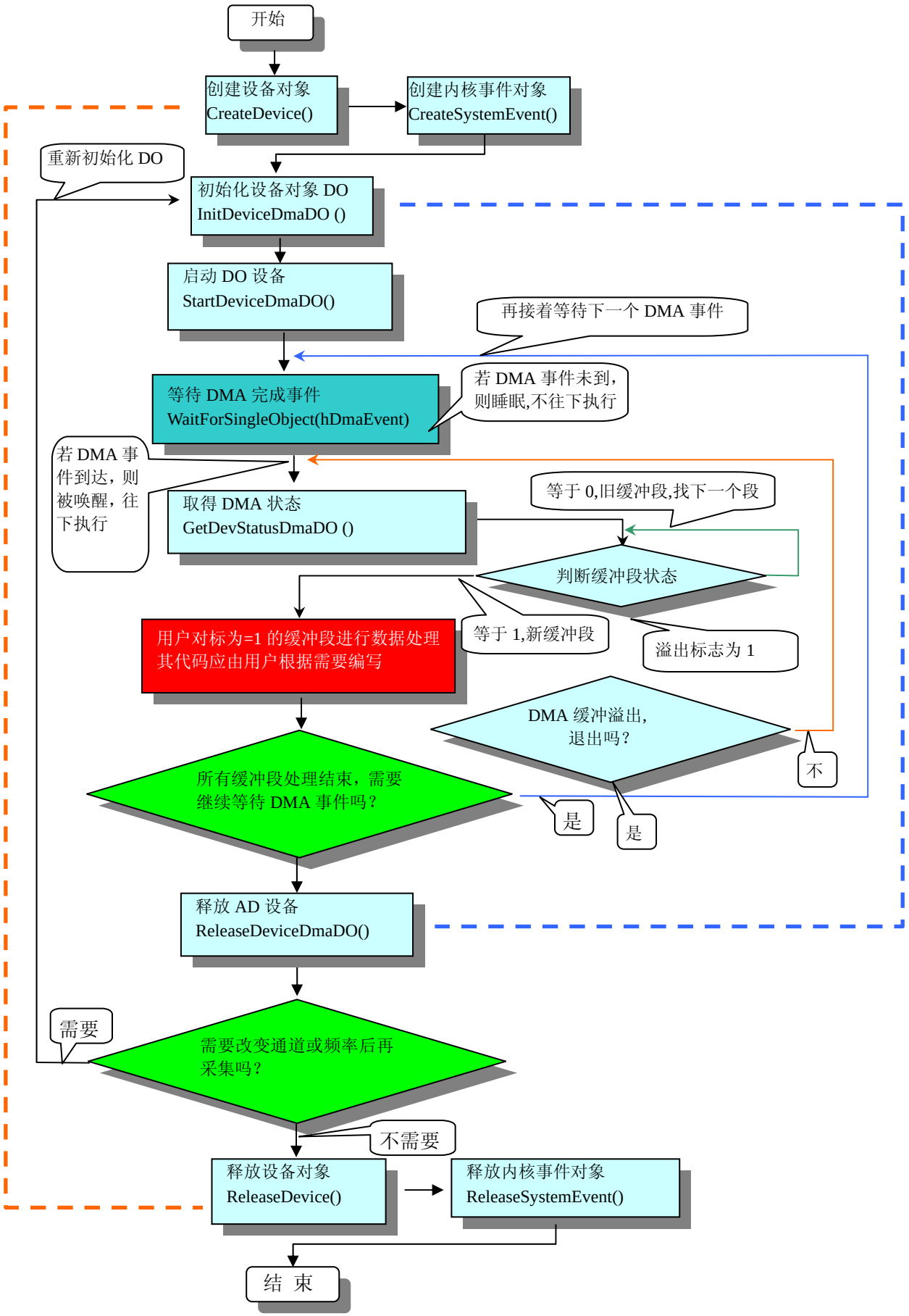


图 2.1.5 DMA 方式 DO 采集实现过程

第八节、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 设备操作函数接口介绍

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI2510_”）

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 DIO 组合控制函数		
SelectDIOChannel	选择 DIO 通道组合模式	上层用户
③ DI 控制函数		
ClrDIFIFO	清 DI 的 FIFO	上层用户
EnableDITerminator	是否使能 DI 终端连接	上层用户
InitDeviceDI	初始化 DI 设备	上层用户
StartDeviceDI	启动 DI 采样	上层用户
SetDICompareValue	设定 DI 比较值	上层用户
ReadDeviceProDI_Npt	非空方式读取 DI 数据	上层用户
GetDevStatusDI	取得设备的各种状态	上层用户
ReadDeviceProDI_Half	半满方式读取 DI 数据	上层用户
StopDeviceDI	停止 DI 采样	上层用户
④ DMA 方式 DI 读取函数（唯有此种方式效率最高）		
InitDeviceDmaDI	初始化 DI 部件	上层用户
StartDeviceDmaDI	启动 DI 采集	上层用户
GetDevStatusDmaDI	取得 DMA 的各种状态	上层用户
SetDevStatusDmaDI	清除 DMA 状态	上层用户
StopDeviceDmaDI	停止 DI 采集	上层用户
ReleaseDeviceDmaDI	释放设备上的 DI 部件	上层用户
⑤ DO 控制函数		
ClrDOFIFO	清 DO 的 FIFO	上层用户
EnableDOTerminator	是否使能 DO 终端连接	上层用户
InitDeviceDO	初始化 DO 设备	上层用户
StartDeviceDO	启动 DO 采样	上层用户
GetDevStatusDO	在 DO 输出过程中取得设备的各种状态,返回值表示函数是否成功	
WriteDeviceProDO	写入 DO 数据	上层用户
StopDeviceDO	停止 DO 采样	上层用户

⑥ DMA 方式 DO 读取函数（唯有此种方式效率最高）		
InitDeviceDmaDO	初始化 DO 部件	上层用户
StartDeviceDmaDO	启动 DO 采集	上层用户
GetDevStatusDmaDO	取得 DMA 的各种状态	上层用户
SetDevStatusDmaDO	清除 DMA 状态	上层用户
StopDeviceDmaDO	停止 DO 采集	上层用户
ReleaseDeviceDmaDO	释放设备上的 DO 部件	上层用户
⑦ 通用 DIO 控制函数		
GetCurrentDI	获取通用 DI	上层用户
SetCurrentDO	设置通用 DO	上层用户
⑧ 中断实现计数器控制函数		
InitDeviceInt	初始化中断	上层用户
ReleaseDeviceInt	释放中断资源	上层用户
ModifyDeviceInt	修改中断	上层用户
GetDeviceIntSrc	获得中断源	上层用户
GetDeviceIntCount	获得中断次数	上层用户
⑨ 计数器操作函数		
SetDeviceCNT	设置计数器的初值	上层用户
GetDeviceCNT	取得各路计数器的当前计数值	上层用户
⑩ 硬件参数系统保存、读取函数		
LoadParaDIMode	从 Windows 系统中读入 DI 硬件参数	上层用户
SaveParaDIMode	往 Windows 系统写入设备 DI 硬件参数	上层用户
ResetParaDIMode	将注册表中的 DI 参数恢复至出厂默认值	上层用户
LoadParaDOMode	从 Windows 系统中读入 DO 硬件参数	上层用户
SaveParaDOMode	往 Windows 系统写入设备 DO 硬件参数	上层用户
ResetParaDOMode	将注册表中的 DO 参数恢复至出厂默认值	上层用户

使用需知：

Visual C++：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\Art\PCI2510\INCLUDE\PCI2510.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 PCI2510.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

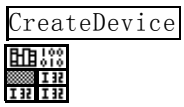
Visual Basic:

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 PCI2510.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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

LabVIEW/CVI :

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



一、在 LabView 中打开 PCI2510.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 ReadDeviceProDI_Npt接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

♦ 创建设备对象函数（逻辑号）

函数原型：

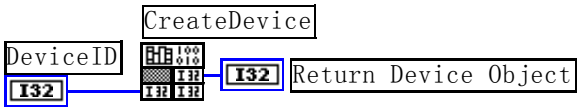
Visual C++：

HANDLE CreateDevice (int DeviceLgcID = 0)

Visual Basic：

Declare Function CreateDevice Lib "PCI2510_32" (ByVal DeviceID As Integer) As Long

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：

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PCI2510 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PCI2510 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++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI2510_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 = PCI2510_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++：

HANDLE CreateDeviceEx(int DevicePhysID = 0)

Visual Basic：

Declare Function CreateDeviceEx Lib"PCI2510_32" (ByVal DevicePhysID As Integer) As Long

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 PCI2510 设备的总数量**

函数原型：

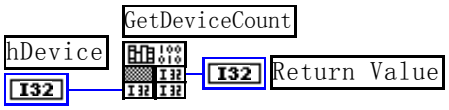
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "PCI2510_32" (ByVal hDevice As Long) As Integer

LabVIEW：



功能：取得 PCI2510 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 PCI2510 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++:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PCI2510_32" (_
                                    ByVal hDevice As Integer,_
                                    ByRef DeviceLgcID As Long,_
                                    ByRef DevicePhysID As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：取得指定设备逻辑和物理 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：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　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PCI2510 设备各种配置信息**

函数原型：

Visual C++:

```
BOOL ListDeviceDlg (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ListDeviceDlg Lib"PCI2510_32" (ByVal hDevice As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI2510 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 PCI2510 设备的配置情况。

相关函数： CreateDevice　ReleaseDevice

♦ **释放设备对象所占的系统资源及设备对象**

函数原型：

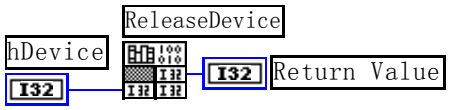
Visual C++:

```
BOOL ReleaseDevice (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDevice Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean
```

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、DIO 组合控制操作函数原型说明

♦ 选择 DIO 通道组合模式

函数原型：

Visual C++:

```
BOOL SelectDIOChannel (HANDLE hDevice,
                       LONG DIOChannelMode)
```

Visual Basic:

```
Declare Function SelectDIOChannel Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                    ByVal DIOChannelMode As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：选择 DIO 通道组合模式。

参数：

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

DIOChannelMode 模式选择，=0，32 路 DI 输入；=1，32 路 DO 输出；=2，16 路 DI、16 路 DO；=3，8 路 DI、8 路 DO。

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 ReleaseDevice

第四节、DI 操作函数原型说明

♦ 清 DI 的 FIFO

函数原型：

Visual C++:

```
BOOL ClrDIFIFO (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ClrDIFIFO Lib "PCI2510_32" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：清 DI 的 FIFO。

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

♦ 是否使能 DI 终端连接

函数原型：

Visual C++:

```
BOOL EnableDITerminator ( HANDLE hDevice,
                          BOOL bEnable)
```

Visual Basic:

```
Declare Function EnableDITerminator Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                      ByVal bEnable As Boolean) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：是否使能 DI 终端连接。

参数：

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

bEnable TRUE 表示使能，FALSE 表示禁止。

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 ReleaseDevice

♦ **初始化 DI 设备**

函数原型：

Visual C++:

```
BOOL InitDeviceDI(HANDLE hDevice,
                  PPCI2510_PARA_MODE_DIO pDIPara)
```

Visual Basic:

```
Declare Function InitDeviceDI Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByRef pDIPara As PCI2510_PARA_ MODE_DIO) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：它负责初始化设备，当返回 TRUE 后，设备即准备就绪。

参数：

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

pDIPara 硬件参数，它仅在此函数中决定硬件状态。

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 EnableDITerminator InitDeviceDI StartDeviceDI SetDICompareValue ReadDeviceProDI_Npt GetDevStatusDI ReadDeviceProDI_Half StopDeviceDI ReleaseDevice

♦ **启动 DI 采样**

函数原型：

Visual C++:

```
BOOL StartDeviceDI ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StartDeviceDI Lib "PCI2510_32" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：启动DI设备，它必须在调用 InitDeviceDI后才能调用此函数。该函数除了启动DI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DI立刻开始，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 EnableDITerminator InitDeviceDI StartDeviceDI SetDICompareValue ReadDeviceProDI_Npt GetDevStatusDI ReadDeviceProDI_Half StopDeviceDI ReleaseDevice

♦ **设定 DI 比较值**

函数原型：

Visual C++:

```
BOOL SetDICompareValue (HANDLE hDevice,
                        ULONG ulCmpValue)
```

Visual Basic:

```
Declare Function SetDICompareValue Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                     ByVal ulCmpValue As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：设定 DI 比较值。

参数：

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

ulCmpValue 比较值。

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, EnableDITerminator, InitDeviceDI, StartDeviceDI, SetDICompareValue, ReadDeviceProDI_Npt, GetDevStatusDI, ReadDeviceProDI_Half, StopDeviceDI, ReleaseDevice

◆ **读取 PCI 设备上的 DI 数据**

① 使用 FIFO 的非空标志读取 DI 数据

函数原型：

Visual C++:

```
BOOL ReadDeviceProDI_Npt (HANDLE hDevice,
                          ULONG DIBuffer[],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProDI_Npt Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                       ByRef DIBuffer As Long,_
                                                       ByVal nReadSizeWords As Long,_
                                                       1ByRef nRetSizeWords As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

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

参数：

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

DIBuffer 接受原始 DI 数据的用户缓冲区，它可以是一个用户定义的数组。

nReadSizeWords 指定一次 ReadDeviceProDI_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DIBuffer的最大空间。此参数值只与DIBuffer []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DIBuffer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 StopDeviceDI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。

相关函数: CreateDevice, EnableDITerminator, InitDeviceDI, StartDeviceDI, SetDICompareValue, ReadDeviceProDI_Npt, GetDevStatusDI, ReadDeviceProDI_Half, StopDeviceDI, ReleaseDevice

② 使用 FIFO 的半满标志读取 DI 数据

◆ **取得 FIFO 的状态标志**

函数原型：

Visual C++:

```
BOOL GetDevStatusDI (HANDLE hDevice,
                     PPCI2510_STATUS_DIO pDIStatus)
```

Visual Basic:

```
Declare Function GetDevStatusDI Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                  ByRef pDIStatus As PCI2510_STATUS_DIO)As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：在DI采样过程中取得设备的各种状态。一旦用户使用 StartDeviceDI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 ReadDeviceProDI_Half读取FIFO中的半满有效DI数据。

参数：

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

pDIStatusDI 的各种信息结构体。它属于结构体，具体定义请参考《DIO 状态参数结构（PCI2510_STATUS_DIO)》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数据，则当 GetDevStatusDI 函数取得的 bHalf 等于TRUE，应立即调用 ReadDeviceProDI_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。

相关函数: CreateDevice, EnableDITerminator, InitDeviceDI, StartDeviceDI, SetDICompareValue, ReadDeviceProDI_Npt, GetDevStatusDI, ReadDeviceProDI_Half, StopDeviceDI, ReleaseDevice

- **当 FIFO 半满信号有效时，批量读取 DI 数据**

函数原型：

Visual C++:

```
BOOL ReadDeviceProDI_Half(HANDLE hDevice,
                          ULONG DIBuffer [],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProDI_Half Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                        ByRef DIBuffer As Long,_
                                                        ByVal nReadSizeWords As Long ,_
                                                        ByRef nRetSizeWords As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用 GetDevStatusDI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中的半满数据。

参数：

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

DIBuffer 接受原始 DI 数据的用户缓冲区，它可以是一个用户定义的数组。

nReadSizeWords 指定一次 ReadDeviceProDI_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DIBuffer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DI数据到用户缓冲区，则返回TRUE，否则返回FALSE，用户可用 GetLastErrorEx捕获当前错误码，并加以分析。

相关函数: CreateDevice, EnableDITerminator, InitDeviceDI, StartDeviceDI, SetDICompareValue, ReadDeviceProDI_Npt, GetDevStatusDI, ReadDeviceProDI_Half, StopDeviceDI, ReleaseDevice

- **停止 DI 采样**

函数原型：

Visual C++:

```
BOOL StopDeviceDI (HANDLE hDevice)
```

Visual Basic:

Declare Function StopDeviceDI Lib "PCI2510_32" (ByVal hDevice As Long)As Boolean

LabVIEW：

请参考相关演示程序。

功能：停止DI采样。它必须在调用 StartDeviceDI后才能调用此函数。该函数除了停止DI设备以外，不改变设备的其他任何状态。此后您可再调用 StartDeviceDI函数重新启动DI。

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DI立刻停止，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, EnableDITerminator, InitDeviceDI, StartDeviceDI, SetDICompareValue, ReadDeviceProDI_Npt, GetDevStatusDI, ReadDeviceProDI_Half, StopDeviceDI, 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 InitDeviceDI
③ StartDeviceDI
④ SetDICompareValue
⑤ ReadDeviceProDI_Npt
⑥ StopDeviceDI
⑦ ReleaseDevice

注明：用户可以反复执行第⑤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 InitDeviceDI
③ StartDeviceDI
④ SetDICompareValue
⑤ GetDevStatusDI
⑥ ReadDeviceProDI_Half
⑦ StopDeviceDI
⑧ ReleaseDevice

注明：用户可以反复执行第⑤、⑥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》。

第五节、DI 直接内存存取 DMA 方式采样操作函数原型说明

（注：函数中的“Dma”字符是 Direct Memory Access 的缩写，标明以直接内存存取方式）

◆ **初始化设备**

函数原型：

Visual C++:

```
BOOL InitDeviceDmaDI(HANDLE hDevice,
                     HANDLE hDmaEvent,
                     ULONG DIBuffer[ ],
                     LONG nReadSizeWords,
                     LONG nSegmentCount,
                     LONG nSegmentSizeWords,
                     PPCI2510_PARA_MODE_DIO pDIPara )
```

Visual Basic:

```
Declare Function InitDeviceDmaDI Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                   ByVal hDmaEvent As Long, _
                                                   ByRef DIBuffer As Long,_
                                                   ByVal nReadSizeWords As Long,_
                                                   ByVal nSegmentCount As Long,_
                                                   ByVal nSegmentSizeWords As Long,_
                                                   ByRef pDIPara As PCI2510_PARA_MODE_DIO)As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的DI部件，为设备操作及DMA传输就绪有关工作。且让设备上的DI部件以硬件DMA的方式工作，但它并不启动DI，而是需要在此函数被成功调用之后，再调用 StartDeviceDmaDI函数即可启动DI。

参数：

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

hDmaEvent DMA 事件对象句柄，它应由 CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件每次DMA完一个指定段长(nSegmentSizeWords)的数据时这个内核系统事件被触发一次。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当该事件没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，因为它并不消耗CPU时间。当hDmaEvent事件被触发成发信号状态，那么WaitForSingleObject将复位该内核系统事件对象，使其处于不发信号状态，并立即唤醒所在线程，继而执行WaitForSingleObject其后的代码，比如移走DIBuffer中的数据、分析数据、显示数据等，待处理完数据后再循环调用WaitForSingleObject，让所在线程再次进入睡眠状态，重复以上过程。所以利用DMA方式采集数据，不仅等待DI转换指定数据不需要消耗CPU时间，同时将DI数据从卡上传输到计算机主存更是不需要花消CPU时间，其效率是最高的。

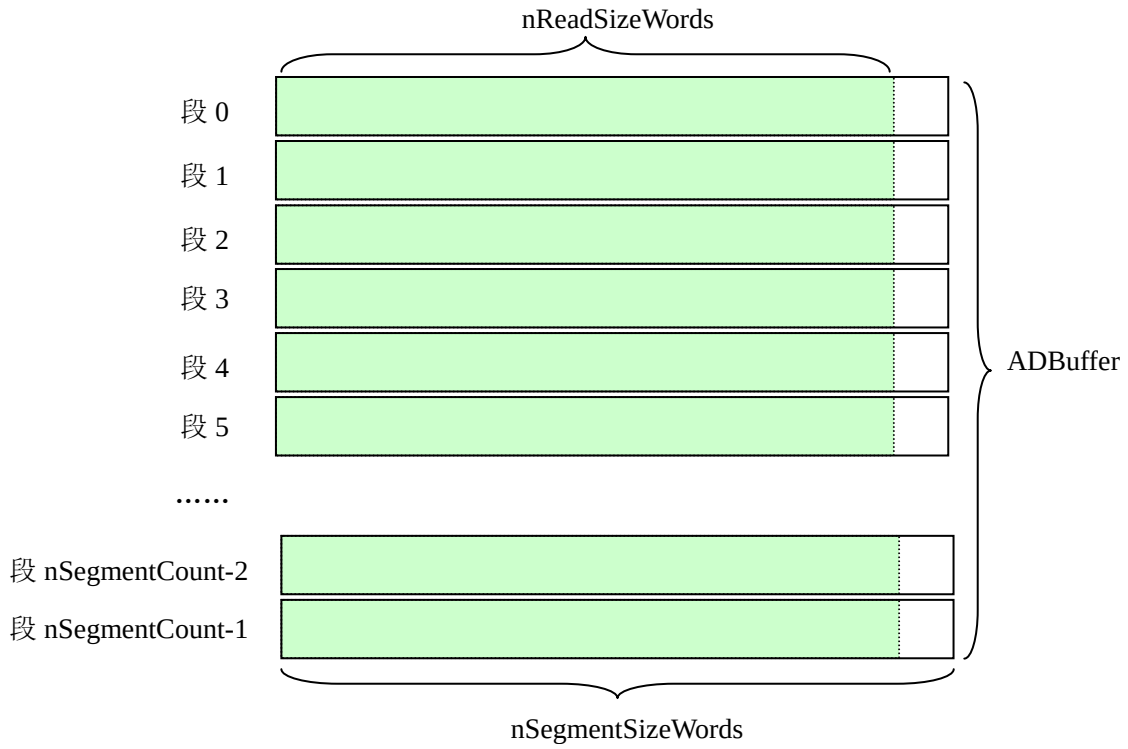
DIBuffer 接受 DI 数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。注意该缓冲区最好定义为两维缓冲或数组，以便 DMA 数据传输和缓冲区数据处理分时错开，以更好的达到 DI 转换、传输、处理等过程的并行工作。**注意：该缓冲区的生命周期必须跨越 DMA 的整个操作周期，建议最好将期置为全局缓冲区，即整个应用程序的生命周期内存在。否则，可能会造成严重的存储区访问违反**。

nReadSizeWords 在每个段缓冲中应 DMA 填充和用户读走的数据点数。它的取值范围不应小于 1，同时，不能大于段长 nSegmentSizeWords，其具体取值应根据采样通道数来确定其大小，通常应在段长范围内，取用为采样通道数整数倍长，同时又最接近段长的读取长度来设置本参数。也就是说每当用户接受到 hDmaEvent 事件后，对相应段缓冲区作数据处理时只能从该段缓冲首单元开始往后共处理 nReadSizeWords 个数据采样点。

nSegmentCount 缓冲区段数。其取值范围为[2-128]。为了提高整体效率和性能，将用户缓冲区人为的划分为若干段，让 DMA 分段传输整个数据序列，以便用户能够实时并发的处理。而每段的长度由 nSegmentSizeWords 参数决定。

nSegmentSizeWords 缓冲区各段的长度(字或点)。其取值范围应等于或小于板载 FIFO 的半满空间。而段数由 nSegmentCount 决定。

pDIPara 设备对象参数结构PCI2510_PARA_ MODE_DIO的指针，它仅在此函数中决定硬件状态。它的各成员值决定了设备上的DI对象的各种状态及工作方式。具体定义请参考PCI2510.h(.Bas或.Pas或.VI)驱动接口文件和本文档中的《硬件参数结构》章节。



DMA 缓冲区结构图

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。

备注：DMA是直接内存存取的意思，其英文定义为：Direct Memory Access。它的技术含义可以顾名思义，就是数据传输在设备和内存之间直接进行，无需要CPU的参与。该项技术的使用大大提高了数据实时采集和处理的效率。但是为了更好的配合这样好的机制，我们需要将用户缓冲区分段，比如分为 32 段，每段的长度等于FIFO半满长度 4096，因此可以定义一个两维数组。如：SHORT DIBuffer[32][4096]，即nSegmentCount=32，nSegmentSizeWords=4096，然后开始启动设备后，DIBuffer[0]首先被DMA占用，当传输完成后，hDmaEvent即被触发，用户即可处理DIBuffer[0]，而DMA接着占用DIBuffer[1]，当传输完成后，hDmaEvent即再次被触发，用户即可处理DIBuffer[1]，而DMA接着占用DIBuffer[2]，就这样依次类推。至到DIBuffer[31]被传输完后DMA再回到始端，占用DIBuffer[0]，就这样周而复始的进行下去。除了hDmaEvent事件对象可以通知用户何时处理数据外，其 GetDevStatusDmaDI 函数也可以实时返回DMA各种状态，如DMA正在占用的缓冲段ID（iCurSegmentID），整个缓冲链各个段的更新状态(bSegmentSts[])，整个缓冲链是否溢出(bBufferOverflow)等，跟踪这些信息，可以使数据转换、传输和处理之间有更大的时间弹性，高度保证数据的连续性。

切记：在 InitDeviceDmaDI 函数被调用之后若想再调用它改变硬件的某些参数，那么必须在 ReleaseDeviceDmaDI之后方可调用。即 InitDeviceDmaDI和 ReleaseDeviceDmaDI必须成对调用，且在应用程序被关闭前必须确保已调用 ReleaseDeviceDmaDI释放了各种DMA资源，否则可能会引起系统严重错误。

相关函数： CreateDevice　InitDeviceDmaDI　StartDeviceDmaDI
GetDevStatusDmaDI　SetDevStatusDmaDI　StopDeviceDmaDI
ReleaseDeviceDmaDI　ReleaseDevice

◆ **启动设备上的 DI 部件**

函数原型：

Visual C++:

BOOL StartDeviceDmaDI(HANDLE hDevice)

Visual Basic:

Declare Function StartDeviceDmaDI Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean

LabVIEW:

请参考相关演示程序。

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

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，意味着DI被启动，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。

相关函数： CreateDevice　InitDeviceDmaDI　StartDeviceDmaDI　GetDevStatusDmaDI　SetDevStatusDmaDI　StopDeviceDmaDI　ReleaseDeviceDmaDI　ReleaseDevice

◆ **取得 DMA 的状态标志**

Visual C++:

```
BOOL GetDevStatusDmaDI ( HANDLE hDevice,
                         PPCI2510_STATUS_DMA pDMAStatus )
```

Visual Basic:

```
Declare Function GetDevStatusDmaDI Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                     ByRef pDMAStatus As PCI2510_STATUS_DMA)As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用 StartDeviceDmaDI后，应立即用此函数查询DMA的状态（当前段缓冲ID、缓冲段新旧标志、DMA缓冲溢出标志）。我们通常用缓冲段新旧标志bSegmentSts[x]去同步缓冲区数据处理操作。当bSegmentSts[x]标志为 1 时表示其该段为新数据段，则可以处理x段数据，然后再执行 SetDevStatusDmaDI函数将x段新旧标志置为 0，表示已处理完，该段变为旧数据。

参数：

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

pDMAStatus它属于PCI2510_STATUS_DMA的结构体指针。该参数实时返回DMA的当前状态。关于PCI2510_STATUS_DMA具体定义请参考PCI2510.h(.Bas或.Pas或.VI)驱动接口文件以及本文档中的《DMA状态参数结构（PCI2510_STATUS_DMA）》。

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。

相关函数： CreateDevice　InitDeviceDmaDI　StartDeviceDmaDI　GetDevStatusDmaDI　SetDevStatusDmaDI　StopDeviceDmaDI　ReleaseDeviceDmaDI　ReleaseDevice

◆ **取得 DMA 的状态标志**

函数原型：

Visual C++:

```
BOOL SetDevStatusDmaDI ( HANDLE hDevice,
                         LONG iClrBufferID )
```

Visual Basic:

```
Declare Function SetDevStatusDmaDI Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                     ByVal iClrBufferID As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：当处理完 DMA 缓冲链中的某一段数据后，应该立即调用此函数将其缓冲段状态标志清除，使其复位至 0，表示该数据已被处理过，已变成了旧数据，以便在下一个 DMA 事件响应下，不会重复自理某一缓冲段的数据。同时也避免产生 DMA 缓冲区溢出的可能。

参数：

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

iClrBufferID 要被清除标志的缓冲段ID。当指定的缓冲段状态标志清除后，则从 GetDevStatusDmaDI函数返回的bSegmentSts[x]则会为 0。只有待到DMA事件下，其相应的缓冲段状态标志才会被置 1。

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。

相关函数： CreateDevice　InitDeviceDmaDI　StartDeviceDmaDI　GetDevStatusDmaDI　SetDevStatusDmaDI　StopDeviceDmaDI　ReleaseDeviceDmaDI　ReleaseDevice

◆ **暂停设备上的 DI 采样工作**

函数原型：
Visual C++:
BOOL StopDeviceDmaDI(HANDLE hDevice)
Visual Basic:
Declare Function StopDeviceDmaDI Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean
LabVIEW:
请参考相关演示程序。

功能：在 StartDeviceDmaDI被成功调用之后，用户可以在任何时候调用此函数停止DI采样(必须在ReleaseDeviceDmaDI之前被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceDmaDI，那么设备会接着停止前的状态(如通道位置)继续开始正常的DI数据转换。
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，意味着DI被停止，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice, InitDeviceDmaDI, StartDeviceDmaDI, GetDevStatusDmaDI, SetDevStatusDmaDI, StopDeviceDmaDI, ReleaseDeviceDmaDI, ReleaseDevice

◆ **释放设备上的 DI 部件**

函数原型：
Visual C++:
BOOL ReleaseDeviceDmaDI(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDeviceDmaDI Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean
LabVIEW:
请参考相关演示程序。

功能：释放设备上的DI部件，如果没有被 StopDeviceDmaDI函数停止，则此函数在释放DI部件之前先停止DI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。
相关函数： CreateDevice, InitDeviceDmaDI, StartDeviceDmaDI, GetDevStatusDmaDI, SetDevStatusDmaDI, StopDeviceDmaDI, ReleaseDeviceDmaDI, ReleaseDevice

应注意的是，InitDeviceDmaDI必须和 ReleaseDeviceDmaDI函数一一对应，即当您执行了一次InitDeviceDmaDI后，再一次执行这些函数前，必须执行一次 ReleaseDeviceDmaDI函数，以释放先前由InitDeviceDmaDI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InitDeviceDmaDI函数时，那些软硬件资源才可被再次使用。

◆ **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceDmaDI
④ StartDeviceDmaDI
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ GetDevStatusDmaDI
⑦ SetDevStatusDmaDI
⑧ StopDeviceDmaDI
⑨ ReleaseDeviceDmaDI
⑩ ReleaseSystemEvent（公共函数）
⑩ ReleaseDevice
注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。
关于这个过程的图形说明请参考《使用纲要》。
注意：若成功初始化 DMA 后，要退出整个应用程序，切记应先释放 DMA 才能退出。

第六节、DO 操作函数原型说明

- **清 DO 的 FIFO**

函数原型：

Visual C++:

```
BOOL ClrDOFIFO (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ClrDOFIFO Lib "PCI2510_32" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：清 DO 的 FIFO。

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

- **是否使能 DO 终端连接**

函数原型：

Visual C++:

```
BOOL EnableDOTerminator ( HANDLE hDevice,
                          BOOL bEnable)
```

Visual Basic:

```
Declare Function EnableDOTerminator Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                      ByVal bEnable As Boolean) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：是否使能 DO 终端连接。

参数：

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

bEnable TRUE 表示使能，FALSE 表示禁止。

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 ReleaseDevice

- **初始化 DO 设备**

函数原型：

Visual C++:

```
BOOL InitDeviceDO (HANDLE hDevice,
                   PPCI2510_PARA_MODE_DIO pDOPara)
```

Visual Basic:

```
Declare Function InitDeviceDO Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByRef pDOPara As PCI2510_PARA_ MODE_DIO) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：它负责初始化设备，当返回 TRUE 后，设备即准备就绪。

参数：

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

pDOPara 硬件参数，它仅在此函数中决定硬件状态。

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 EnableDOTerminator InitDeviceDO
StartDeviceDO GetDevStatusDO WriteDeviceProDO
StopDeviceDO ReleaseDevice

♦ **启动 DO 采样**

函数原型：

Visual C++:

```
BOOL StartDeviceDO ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StartDeviceDO Lib "PCI2510_32" (ByVal hDevice As Long ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：启动DO设备，它必须在调用 InitDeviceDO后才能调用此函数。该函数除了启动DO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DO立刻开始，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, EnableDOTerminator, InitDeviceDO, StartDeviceDO, GetDevStatusDO, WriteDeviceProDO, StopDeviceDO, ReleaseDevice

♦ **取得状态标志**

函数原型：

Visual C++:

```
BOOL GetDevStatusDO (HANDLE hDevice,
                     PPCI2510_STATUS_DIO pDOStatus)
```

Visual Basic:

```
Declare Function GetDevStatusDO Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                  ByRef pDOStatus As PCI2510_STATUS_DIO)As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：在 DO 采样过程中取得设备的各种状态。

参数：

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

pDOStatus DO 的各种信息结构体。它属于结构体，具体定义请参考《DIO 状态参数结构（PCI2510_STATUS_DIO）》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。

相关函数: CreateDevice, EnableDOTerminator, InitDeviceDO, StartDeviceDO, GetDevStatusDO, WriteDeviceProDO, StopDeviceDO, ReleaseDevice

♦ **写入 DO 数据**

函数原型：

Visual C++:

```
BOOL WriteDeviceProDO ( HANDLE hDevice,
                        ULONG DOBuffer [],
                        LONG nWriteSizeWords,
                        PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function WriteDeviceProDO Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                    ByRef DOBufferAs Long,_
                                                    ByVal nWriteSizeWords As Long,_
                                                    ByRef nRetSizeWords As Long) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：写入 DO 数据。

参数：

hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
DOBuffer 接受原始 DO 数据的用户缓冲区，它可以是一个用户定义的数组。
nWriteSizeWords 写入数据的长度。
nRetSizeWords 返回实际写入的点数(或字数)。
返回值：其返回值表示所成功写入的数据点数(字)，也表示当前写操作在DOBuffer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 StopDeviceDO函数中断了写操作，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写入和几个点的写入，只需要将 nReadSizeWords 设置成 1 或相应值即可。

相关函数: CreateDevice, EnableDOTerminator, InitDeviceDO, StartDeviceDO, GetDevStatusDO, WriteDeviceProDO, StopDeviceDO, ReleaseDevice

♦ **停止 DO 采样**

函数原型：

Visual C++:

```
BOOL StopDeviceDO (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceDO Lib "PCI2510_32" (ByVal hDevice As Long )As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：停止DO采样。它必须在调用 StartDeviceDO后才能调用此函数。该函数除了停止DO设备以外，不改变设备的其他任何状态。此后您可再调用 StartDeviceDO函数重新启动DO。
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DO立刻停止，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, EnableDOTerminator, InitDeviceDO, StartDeviceDO, GetDevStatusDO, WriteDeviceProDO, StopDeviceDO, ReleaseDevice

♦ **程序查询方式采样函数一般调用顺序**

① CreateDevice
② InitDeviceDO
③ StartDeviceDO
④ GetDevStatusDO
⑤ WriteDeviceProDO
⑥ StopDeviceDO
⑦ ReleaseDevice

注明：用户可以反复执行第⑤步，以实现高速连续不间断大容量采集。
关于这个过程的图形说明请参考《使用纲要》。

第七节、DO 直接内存存取 DMA 方式采样操作函数原型说明

（注：函数中的“Dma”字符是 Direct Memory Access 的缩写，标明以直接内存存取方式）

♦ **初始化设备**

函数原型：

Visual C++:

```
BOOL InitDeviceDmaDO(HANDLE hDevice,
                     HANDLE hDmaEvent,
                     ULONG DOBuffer[ ],
                     LONG nWriteSizeWords,
                     LONG nSegmentCount,
                     LONG nSegmentSizeWords,
                     PPCI2510_PARA_MODE_DIO pDOPara )
```

Visual Basic:

```
Declare Function InitDeviceDmaDO Lib "PCI2510_32" (
                                        ByVal hDevice As Long, _
                                        ByVal hDmaEvent As Long, _
                                        ByRef DOBuffer As Long,_
                                        ByVal nWriteSizeWordsAs Long,_
                                        ByVal nSegmentCount As Long,_
                                        ByVal nSegmentSizeWords As Long,_
                                        ByRef pDOPara As PCI2510_PARA_MODE_DIO)As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的DO部件，为设备操作及DMA传输就绪有关工作。且让设备上的DO部件以硬件DMA的方式工作，但它并不启动DO，而是需要在此函数被成功调用之后，再调用 StartDeviceDmaDO函数即可启动DO。

参数：

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

hDmaEvent DMA 事件对象句柄，它应由 CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件每次DMA完一个指定段长(nSegmentSizeWords)的数据时这个内核系统事件被触发一次。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当该事件没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，因为它并不消耗CPU时间。当hDmaEvent事件被触发成发信号状态，那么WaitForSingleObject将复位该内核系统事件对象，使其处于不发信号状态，并立即唤醒所在线程，继而执行WaitForSingleObject其后的代码，比如移走DOBuffer中的数据、分析数据、显示数据等，待处理完数据后再循环调用WaitForSingleObject，让所在线程再次进入睡眠状态，重复以上过程。所以利用DMA方式采集数据，不仅等待DO转换指定数据不需要消耗CPU时间，同时将DO数据从卡上传输到计算机主存更是不需要花消CPU时间，其效率是最高的。

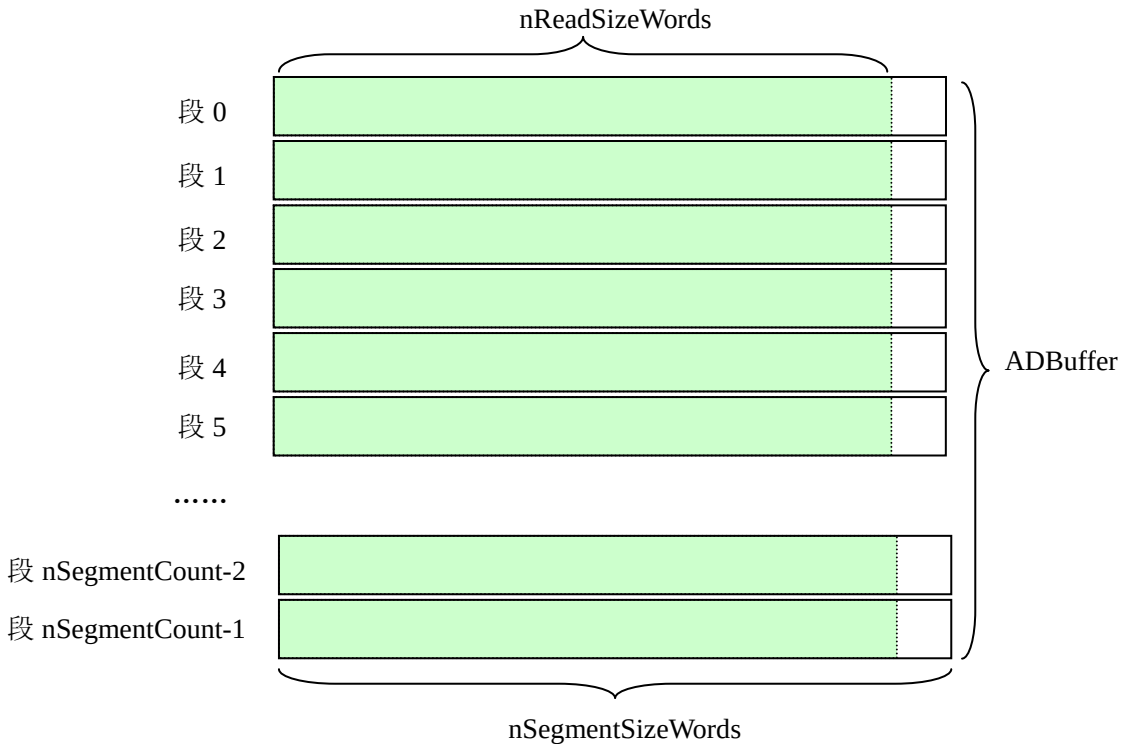
DOBuffer 接受 DO 数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。注意该缓冲区最好定义为两维缓冲或数组，以便 DMA 数据传输和缓冲区数据处理分时错开，以更好的达到 DO 转换、传输、处理等过程的并行工作。**注意：该缓冲区的生命周期必须跨越 DMA 的整个操作周期，建议最好将期置为全局缓冲区，即整个应用程序的生命周期内存在。否则，可能会造成严重的存储区访问违反**。

nWriteSizeWords 在每个段缓冲中应 DMA 填充和用户读走的数据点数。它的取值范围不应小于 1，同时，不能大于段长 nSegmentSizeWords，其具体取值应根据采样通道数来确定其大小，通常应在段长范围内，取用为采样通道数整数倍长，同时又最接近段长的读取长度来设置本参数。也就是说每当用户接受到 hDmaEvent 事件后，对相应段缓冲区作数据处理时只能从该段缓冲首单元开始往后共处理 nReadSizeWords 个数据采样点。

nSegmentCount 缓冲区段数。其取值范围为[2-128]。为了提高整体效率和性能，将用户缓冲区人为的划分为若干段，让 DMA 分段传输整个数据序列，以便用户能够实时并发的处理。而每段的长度由 nSegmentSizeWords 参数决定。

nSegmentSizeWords 缓冲区各段的长度(字或点)。其取值范围应等于或小于板载 FIFO 的半满空间。而段数由 nSegmentCount 决定。

pDOPara 设备对象参数结构PCI2510_PARA_ MODE_DIO的指针，它仅在此函数中决定硬件状态。它的各成员值决定了设备上的DO对象的各种状态及工作方式。具体定义请参考PCI2510.h(.Bas或.Pas或.VI)驱动接口文件和本文档中的《硬件参数结构》章节。



DMA 缓冲区结构图

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。

备注：DMA是直接内存存取的意思，其英文定义为：Direct Memory Access。它的技术含义可以顾名思义，就是数据传输在设备和内存之间直接进行，无需要CPU的参与。该项技术的使用大大提高了数据实时采集和处理的效率。但是为了更好的配合这样好的机制，我们需要将用户缓冲区分段，比如分为 32 段，每段的长度等于FIFO半满长度 4096，因此可以定义一个两维数组。如：SHORT DOBuffer[32][4096]，即nSegmentCount=32，nSegmentSizeWords=4096，然后开始启动设备后，DOBuffer[0]首先被DMA占用，当传输完成后，hDmaEvent即被触发，用户即可处理DOBuffer[0]，而DMA接着占用DOBuffer[1]，当传输完成后，hDmaEvent即再次被触发，用户即可处理DOBuffer[1]，而DMA接着占用DOBuffer[2]，就这样依次类推。至到DOBuffer[31]被传输完后DMA再回到始端，占用DOBuffer[0]，就这样周而复始的进行下去。除了hDmaEvent事件对象可以通知用户何时处理数据外，其 GetDevStatusDmaDO函数也可以实时返回DMA各种状态，如DMA正在占用的缓冲段ID（iCurSegmentID），整个缓冲链各个段的更新状态(bSegmentSts[])，整个缓冲链是否溢出(bBufferOverflow)等，跟踪这些信息，可以使数据转换、传输和处理之间有更大的时间弹性，高度保证数据的连续性。

切记：在 InitDeviceDmaDO 函数被调用之后若想再调用它改变硬件的某些参数，那么必须在 ReleaseDeviceDmaDO之后方可调用。即 InitDeviceDmaDO和 ReleaseDeviceDmaDO必须成对调用，且在应用程序被关闭前必须确保已调用 ReleaseDeviceDmaDO释放了各种DMA资源，否则可能会引起系统严重错误。

相关函数： CreateDevice　InitDeviceDmaDO　StartDeviceDmaDO
GetDevStatusDmaDO　SetDevStatusDmaDO　StopDeviceDmaDO
ReleaseDeviceDmaDO　ReleaseDevice

♦ **启动设备上的 DO 部件**

函数原型：

Visual C++:

BOOL StartDeviceDmaDO(HANDLE hDevice)

Visual Basic:

Declare Function StartDeviceDmaDO Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean

LabVIEW:

请参考相关演示程序。

功能：在 InitDeviceDmaDO被成功调用之后，调用此函数即可启动设备上的DO部件，让设备开始DO采样。
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，意味着DO被启动，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice InitDeviceDmaDO StartDeviceDmaDO GetDevStatusDmaDO SetDevStatusDmaDO StopDeviceDmaDO ReleaseDeviceDmaDO ReleaseDevice

♦ **取得 DMA 的状态标志**

Visual C++:

```
BOOL GetDevStatusDmaDO ( HANDLE hDevice,
                         PPCI2510_STATUS_DMA pDMAStatus )
```

Visual Basic:

```
Declare Function GetDevStatusDmaDO Lib "PCI2510_32" (ByVal hDevice As Long,_
                                    ByRef pDMAStatus As PCI2510_STATUS_DMA)As Boolean
```

LabVIEW:
请参考相关演示程序。

功能：一旦用户使用 StartDeviceDmaDO后，应立即用此函数查询DMA的状态（当前段缓冲ID、缓冲段新旧标志、DMA缓冲溢出标志）。我们通常用缓冲段新旧标志bSegmentSts[x]去同步缓冲区数据处理操作。当bSegmentSts[x]标志为 1 时表示其该段为新数据段，则可以处理x段数据，然后再执行 SetDevStatusDmaDO函数将x段新旧标志置为 0，表示已处理完，该段变为旧数据。
参数：
hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
pDMAStatus它属于PCI2510_STATUS_DMA的结构体指针。该参数实时返回DMA的当前状态。关于PCI2510_STATUS_DMA具体定义请参考PCI2510.h(.Bas或.Pas或.VI)驱动接口文件以及本文档中的《DMA状态参数结构（PCI2510_STATUS_DMA）》。
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。
相关函数： CreateDevice InitDeviceDmaDO StartDeviceDmaDO GetDevStatusDmaDO SetDevStatusDmaDO StopDeviceDmaDO ReleaseDeviceDmaDO ReleaseDevice

♦ **取得 DMA 的状态标志**

函数原型：

Visual C++:

```
BOOL SetDevStatusDmaDO ( HANDLE hDevice,
                         LONG iClrBufferID )
```

Visual Basic:

```
Declare Function SetDevStatusDmaDO Lib "PCI2510_32" (ByVal hDevice As Long,_
                                    ByVal iClrBufferID As Long) As Boolean
```

LabVIEW:
请参考相关演示程序。

功能：当处理完 DMA 缓冲链中的某一段数据后，应该立即调用此函数将其缓冲段状态标志清除，使其复位至 0，表示该数据已被处理过，已变成了旧数据，以便在下一个 DMA 事件响应下，不会重复自理某一缓冲段的数据。同时也避免产生 DMA 缓冲区溢出的可能。
参数：
hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
iClrBufferID 要被清除标志的缓冲段ID。当指定的缓冲段状态标志清除后，则从 GetDevStatusDmaDO函数返回的bSegmentSts[x]则会为 0。只有待到DMA事件下，其相应的缓冲段状态标志才会被置 1。
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。
相关函数： CreateDevice InitDeviceDmaDO StartDeviceDmaDO GetDevStatusDmaDO SetDevStatusDmaDO StopDeviceDmaDO ReleaseDeviceDmaDO ReleaseDevice

♦ **暂停设备上的 DO 采样工作**

函数原型：
Visual C++:
BOOL StopDeviceDmaDO(HANDLE hDevice)
Visual Basic:
Declare Function StopDeviceDmaDO Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean
LabVIEW:
请参考相关演示程序。

功能：在 StartDeviceDmaDO被成功调用之后，用户可以在任何时候调用此函数停止DO采样(必须在ReleaseDeviceDmaDO之前被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceDmaDO，那么设备会接着停止前的状态(如通道位置)继续开始正常的DO数据转换。
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，意味着DO被停止，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice, InitDeviceDmaDO, StartDeviceDmaDO, GetDevStatusDmaDO, SetDevStatusDmaDO, StopDeviceDmaDO, ReleaseDeviceDmaDO, ReleaseDevice

- **释放设备上的 DO 部件**

函数原型：
Visual C++:
BOOL ReleaseDeviceDmaDO(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDeviceDmaDO Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean
LabVIEW:
请参考相关演示程序。

功能：释放设备上的DO部件，如果没有被 StopDeviceDmaDO函数停止，则此函数在释放DO部件之前先停止DO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。
相关函数： CreateDevice, InitDeviceDmaDO, StartDeviceDmaDO, GetDevStatusDmaDO, SetDevStatusDmaDO, StopDeviceDmaDO, ReleaseDeviceDmaDO, ReleaseDevice

应注意的是，InitDeviceDmaDO必须和 ReleaseDeviceDmaDO函数一一对应，即当您执行了一次InitDeviceDmaDO后，再一次执行这些函数前，必须执行一次 ReleaseDeviceDmaDO函数，以释放先前由InitDeviceDmaDO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InitDeviceDmaDO函数时，那些软硬件资源才可被再次使用。

- **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceDmaDO
④ StartDeviceDmaDO
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ GetDevStatusDmaDO
⑦ SetDevStatusDmaDO
⑧ StopDeviceDmaDO
⑨ ReleaseDeviceDmaDO
⑩ ReleaseSystemEvent（公共函数）
⑪ ReleaseDevice

注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。
关于这个过程的图形说明请参考《使用纲要》。
注意：若成功初始化 DMA 后，要退出整个应用程序，切记应先释放 DMA 才能退出。

第八节、通用 DIO 控制函数原型说明

◆ **通用开关量输入**

函数原型：

Visual C++:

```
BOOL GetCurrentDI ( HANDLE hDevice,
                    BYTE byDISts [4])
```

Visual Basic:

```
Declare Function GetCurrentDI Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByVal byDISts (0 to 3) As Byte) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的通用输入开关量状态读入到 byDISts [x]数组参数中。

参数：

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

byDISts 四路通用开关量输入状态的参数结构，共有 4 个元素，分别对应于 DI0-DI3 路开关量输入状态位。如果 byDISts [0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

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

相关函数: CreateDevice SetCurrentDO ReleaseDevice

◆ **通用开关量输出**

函数原型：

Visual C++:

```
BOOL SetCurrentDO (HANDLE hDevice,
                   BYTE byDOSts [4])
```

Visual Basic:

```
Declare Function SetCurrentDO Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByVal byDOSts (0 to 3) As Byte) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的通用输出开关量置成由 byDOSts [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。

byDOSts 四路开关量输出状态的参数结构，共有 4 个元素，分别对应于 DO0-DO3 路开关量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice GetCurrentDI ReleaseDevice

◆ **以上函数调用一般顺序**

① CreateDevice

② SetCurrentDO (或 GetCurrentDI，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）。

第九节、中断实现计数器控制函数原型说明

◆ **初始化设备中断对象**

函数原型：

Visual C++:

```
BOOL InitDeviceInt (HANDLE hDevice,
                    HANDLE hEventInt,
```

PPCI2510_PARA_INT pIntPara)

Visual Basic:

Declare Function InitDeviceInt Lib "PCI2510_32" (ByVal hDevice As Long, _
ByVal hEventInt As Long, _
ByRef pIntPara As PCI2510_PARA_INT) As Boolean

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的部件，为设备操作就绪有关工作。且让设备上的部件以硬件中断的方式工作，其中断源信号由 FIFO 芯片半满管脚提供。

参数：

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

hEventInt中断事件对象句柄，它应由 CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEvent事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。

pIntPara设备对象中断配置参数结构指针，它的各成员值决定了设备上的对象的各种状态及工作方式。请参考《硬件参数结构》章节。

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

相关函数： CreateDevice ReleaseDeviceInt ReleaseDevice

- **释放中断资源**

函数原型：

Visual C++:

BOOL ReleaseDeviceInt (HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceInt Lib "PCI2510_32" (ByVal hDevice As Long) As Boolean

LabVIEW:

请参考相关演示程序。

功能：释放中断资源。

参数：hDevice设备对象句柄，它应由 CreateDevice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 InitDeviceInt ReleaseDevice

应注意的是，InitDeviceInt必须和 ReleaseDeviceInt函数一一对应，即当您执行了一次 InitDeviceInt后，再一次执行这些函数前，必须执行一次 ReleaseDeviceInt函数，以释放先前由 InitDeviceInt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 InitDeviceInt函数时，那些软硬件资源才可被再次使用。

- **修改中断**

Visual C++:

BOOL ModifyDeviceInt(HANDLE hDevice,
PPCI2510_PARA_INT pIntPara)

Visual Basic:

Declare Function ModifyDeviceInt Lib "PCI2510_32" (ByVal hDevice As Long,_
ByRef pIntPara As PCI2510_PARA_INT) As Boolean

LabVIEW:

请参考相关演示程序。

功能：修改中断，必修在初始化中断成功后才可以调用这个函数。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
pIntPara 中断配置参数。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice InitDeviceInt ReleaseDevice

- **获得中断源**

Visual C++:

```
BOOL GetDeviceIntSrc (HANDLE hDevice,
                      PPCI2510_PARA_INT pIntPara)
```

Visual Basic:

```
Declare Function GetDeviceIntSrc Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                   ByRef pIntPara As PCI2510_PARA_INT) As Boolean
```

LabVIEW:
请参考相关演示程序。

功能：获得中断源。
参数：
hDevice设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
pIntPara 中断配置参数。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice InitDeviceInt ReleaseDevice

- **得到中断次数**

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceIntCount Lib "PCI2510_32" (ByVal hDevice As Long,_
                                                     ByRef pIntSrc As PCI2510_PARA_INT) As Boolean
```

LabVIEW:
请参考相关演示程序。

功能：得到中断次数。
参数：
hDevice设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
pIntSrc 中断配置参数。
返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用 GetLastErrorEx捕获错误码。
相关函数： CreateDevice InitDeviceInt ReleaseDevice

- **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceInt
④ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑤ ReleaseDeviceInt
⑥ ReleaseSystemEvent（公共函数）
⑦ ReleaseDevice

第十节、CNT 计数与定时器操作函数原型说明

- **设置计数器的初值**

函数原型：

Visual C++:

```
BOOL SetDeviceCNT(HANDLE hDevice,
                  ULONG ContrlMode,
                  ULONG CNTVal,
```

ULONG ulChannel)

Visual Basic:

```
Declare Function SetDeviceCNT Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByVal ContrlMode As Long, _
                                                ByVal CNTVal As Long,_
                                                ByVal ulChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：设置计数器的初值。

参数：

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

ContrlMode 方式控制字。其选项值如下表：

常量名	常量值	功能定义
PCI2510_GATEMODE_POSITIVE_0	0x0000	计数结束产生中断
PCI2510_GATEMODE_RISING_1	0x0001	可编程单拍脉冲
PCI2510_GATEMODE_POSITIVE_2	0x0002	频率发生器
PCI2510_GATEMODE_POSITIVE_3	0x0003	方波发生器
PCI2510_GATEMODE_POSITIVE_4	0x0004	软件触发选通
PCI2510_GATEMODE_RISING_5	0x0005	硬件触发选通

CNTVal 计数初值(32 位)。

ulChannel 通道选择[0-2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceCNT ReleaseDevice

- **取得各路计数器的当前计数值**

函数原型：

Visual C++:

```
BOOL GetDeviceCNT (HANDLE hDevice,
                   PULONG pCNTVal,
                   ULONG  ulChannel)
```

Visual Basic:

```
Declare Function GetDeviceCNT Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByRef pCNTVal As Long,_
                                                ByVal ulChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：取得各路计数器的当前计数值。

参数：

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

pCNTVal 返回计数值(32 位)。

ulChannel 通道选择[0-2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetDeviceCNT ReleaseDevice

第十一节、硬件参数保存与读取函数原型说明

- **从 Windows 系统中读出 DI 模式参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaDIMode (HANDLE hDevice,
                     PPCI2510_PARA_MODE_DIO pDIModePara)
```

Visual Basic:

```
Declare Function LoadParaDIMode Lib "PCI2510_32" (ByVal hDevice As Long, _
```

ByRef pDIModePara As PCI2510_PARA_MODE_DIO) As Boolean

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取 DI 模式参数。

参数：

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

pDIModePara属于PPCI2510_PARA_MODE_DIO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PPCI2510_PARA_MODE_DIO请参考PCI2510.h或PCI2510.Bas或PCI2510.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 LoadParaDIMode SaveParaDIMode ResetParaDIMode ReleaseDevice

♦ 往 Windows 系统写入 DI 模式参数函数

函数原型：

Viusal C++:

```
BOOL SaveParaDIMode (HANDLE hDevice,
                     PPCI2510_PARA_MODE_DIO pDIModePara)
```

Visual Basic:

```
Declare Function SaveParaDIMode Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                  ByRef pDIModePara As PCI2510_PARA_MODE_DIO) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责把当前的 DI 模式参数保存在 Windows 系统中。

参数：

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

pDIModePara属于PPCI2510_PARA_MODE_DIO的结构指针类型，关于PPCI2510_PARA_MODE_DIO的详细介绍请参考PCI2510.h或PCI2510.Bas或PCI2510.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 LoadParaDIMode SaveParaDIMode ResetParaDIMode ReleaseDevice

♦ DI 模式参数复位至出厂默认值函数

函数原型：

Viusal C++:

```
BOOL ResetParaDIMode (HANDLE hDevice,
                      PPCI2510_PARA_MODE_DIO pDIModePara)
```

Visual Basic:

```
Declare Function ResetParaDIMode Lib "PCI2510_32" ( ByVal hDevice As Long, _
                                                    ByRef pDIModePara As PCI2510_PARA_MODE_DIO) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将系统中原来的 DI 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。

pDIModePara属于PPCI2510_PARA_MODE_DIO的结构指针类型，它负责在参数被复位后返回其复位后的值。关于PPCI2510_PARA_MODE_DIO的详细介绍请参考PCI2510.h或PCI2510.Bas或PCI2510.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 LoadParaDIMode SaveParaDIMode ResetParaDIMode ReleaseDevice

♦ **从 Windows 系统中读出 DO 模式参数函数**

函数原型：

Visual C++:

```
BOOL LoadParaDOMode(HANDLE hDevice,
                    PPCI2510_PARA_MODE_DIO pDOModePara)
```

Visual Basic:

```
Declare Function LoadParaDOMode Lib "PCI2510_32" (ByVal hDevice As Long, _
                    ByRef pDOModePara As PCI2510_PARA_MODE_DIO) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取 DO 模式参数。

参数：

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

pDOModePara属于PPCI2510_PARA_MODE_DIO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PPCI2510_PARA_MODE_DIO请参考PCI2510.h或PCI2510.Bas或PCI2510.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　LoadParaDOMode　SaveParaDOMode　ResetParaDOMode　ReleaseDevice

♦ **往 Windows 系统写入 DO 模式参数函数**

函数原型：

Viusal C++:

```
BOOL SaveParaDOMode (HANDLE hDevice,
                     PPCI2510_PARA_MODE_DIO pDOModePara)
```

Visual Basic:

```
Declare Function SaveParaDOMode Lib "PCI2510_32" (ByVal hDevice As Long, _
                    ByRef pDOModePara As PCI2510_PARA_MODE_DIO) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：负责把当前的 DO 模式参数保存在 Windows 系统中。

参数：

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

pDOModePara属于PPCI2510_PARA_MODE_DIO的结构指针类型，关于PPCI2510_PARA_MODE_DIO的详细介绍请参考PCI2510.h或PCI2510.Bas或PCI2510.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　LoadParaDOMode　SaveParaDOMode　ResetParaDOMode　ReleaseDevice

♦ **DO 模式参数复位至出厂默认值函数**

函数原型：

Viusal C++:

```
BOOL ResetParaDOMode (HANDLE hDevice,
                      PPCI2510_PARA_MODE_DIO pDOModePara)
```

Visual Basic:

```
Declare Function ResetParaDOMode Lib "PCI2510_32" ( ByVal hDevice As Long, _
                    ByRef pDOModePara As PCI2510_PARA_MODE_DIO) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：将系统中原来的 DO 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。

pDOModePara属于PPCI2510_PARA_MODE_DIO的结构指针类型，它负责在参数被复位后返回其复位后的值。关于PPCI2510_PARA_MODE_DIO的详细介绍请参考PCI2510.h或PCI2510.Bas或PCI2510.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 LoadParaDOMode SaveParaDOMode ResetParaDOMode ReleaseDevice

第四章 硬件参数结构

第一节、DIO 参数结构（PCI2510_PARA_MODE_DIO）

Visual C++：

```
typedef struct _PCI2510_PARA_MODE_DIO
{
    LONG InOutMode;          // 开关量输入输出模式
    LONG ClkSource;          // 采样时钟源选择
    LONG ClkFrequency;       // 内时钟频率
    LONG StartMode;          // DIO 普通模式启动方式
    LONG StopMode;           // DIO 普通模式停止方式
    LONG StartTrigMode;      // 普通模式时的启动触发方式
    LONG StopTrigMode;       // 普通模式时的停止触发方式
    LONG ReqTrigMode;        // REQ 模式的触发方式
    LONG AckTrigMode;        // ACK 模式的触发方式
    LONG ClkWorkMode;        // 采样时钟工作方式
    LONG HandShakeMode;      // 握手模式
} PCI2510_PARA_MODE_DIO, *PPCI2510_PARA_MODE_DIO;
```

Visual Basic：

```
Type PCI2510_PARA_ MODE_DIO
    InOutMode As Long           ' 开关量输入输出模式
    ClkSource As Long           ' 采样时钟源选择
    ClkFrequency As Long        ' 内时钟频率

    StartMode As Long           ' DIO 普通模式启动方式
    StopMode As Long            ' DIO 普通模式停止方式
    StartTrigMode As Long       ' 普通模式时的启动触发方式
    StopTrigMode As Long        ' 普通模式时的停止触发方式
    ReqTrigMode As Long         ' REQ 模式的触发方式
    AckTrigMode As Long         ' ACK 模式的触发方式
    ClkWorkMode As Long         ' 采样时钟工作方式
    HandShakeMode As Long       ' 握手模式
End Type
```

LabVIEW:

请参考相关演示程序。

此结构主要用于设定设备 DIO 模式参数值，用这个参数结构对设备进行硬件配置完全由 InitDeviceDI 或 InitDeviceDmaDI、InitDeviceDmaDO 或 InitDeviceDmaDO 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。

InOutMode 开关量输入输出模式。它的取值如下表：

常量名	常量值	功能定义
PCI2510_INPUTOROUTPUT_INVALIDATE	0x00	输入输出均无效
PCI2510_INPUTOROUTPUT_GENERAL	0x01	普通模式输入输出

PCI2510_INPUTOROUTPUT_HANDSHAKE	0x02	握手模式输入输出

ClkSource　采样时钟源选择。它的取值如下表：

常量名	常量值	功能定义
PCI2510_CLOCKSRC_INVALIDATE	0x00	时钟无效
PCI2510_CLOCKSRC_20M	0x01	20M 时钟
PCI2510_CLOCKSRC_15M	0x02	15M 时钟
PCI2510_CLOCKSRC_10M	0x03	10M 时钟
PCI2510_CLOCKSRC_IN	0x04	内时钟
PCI2510_CLOCKSRC_OUT	0x05	外部时钟

ClkFrequency　内时钟频率。

StartMode　DIO 普通模式启动方式。它的取值如下表：

常量名	常量值	功能定义
PCI2510_STRMODE_INVALIDATE	0x00	无效
PCI2510_STRMODE_SOFT	0x01	软件启动
PCI2510_STRMODE_DIOSTR	0x02	STR 触发启动
PCI2510_STRMODE_DI	0x03	DI 匹配启动，DO 没有此选项

StopMode DIO 普通模式停止方式。它的取值如下表：

常量名	常量值	功能定义
PCI2510_STPMODE_INVALIDATE	0x00	无效
PCI2510_STPMODE_SOFT	0x01	软件停止
PCI2510_STPMODE_DIOSTR	0x02	STR 触发停止
PCI2510_STPMODE_DI	0x03	DI 匹配停止，DO 没有此选项

StartTrigMode　普通模式时的启动触发方式。

常量名	常量值	功能定义
PCI2510_STRTRIGDIR_POSITIVE	0x00	上升沿触发
PCI2510_STRTRIGDIR_NEGATIVE	0x01	下降沿触发

StopTrigMode　普通模式时的停止触发方式。

常量名	常量值	功能定义
PCI2510_STPTRIGDIR_POSITIVE	0x00	上升沿触发
PCI2510_STPTRIGDIR_NEGATIVE	0x01	下降沿触发

ReqTrigMode REQ 模式的触发方式，取值如下表：

常量名	常量值	功能定义
PCI2510_REQTRIGDIR_POSITIVE	0x00	高电平有效
PCI2510_REQTRIGDIR_NEGATIVE	0x01	低电平有效

AckTrigMode　ACK 模式的触发方式。

常量名	常量值	功能定义
PCI2510_ACKTRIGDIR_POSITIVE	0x00	高电平有效
PCI2510_ACKTRIGDIR_NEGATIVE	0x01	低电平有效

ClkWorkMode　采样时钟工作方式。

常量名	常量值	功能定义
PCI2510_CLKWORDMODEDIR_POSITIVE	0x00	时钟上升沿触发采集
PCI2510_CLKWORDMODEDIR_NEGATIVE	0x01	时钟下降沿触发采集

HandShakeMode　握手模式。它的选项值如下表：

常量名	常量值	功能定义
PCI2510_HANDSHAKEMODE_BURST	0x00	突发握手
PCI2510_HANDSHAKEMODE_8255	0x01	8255 模型

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
PCI2510_modeRead	0x0000	只读文件方式
PCI2510_modeWrite	0x0001	只写文件方式
PCI2510_modeReadWrite	0x0002	既读又写文件方式
PCI2510_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
PCI2510_typeText	0x4000	以文本方式操作文件

第二节、DIO 状态参数结构（PCI2510_STATUS_DIO）

Visual C++：

```
typedef struct _PCI2510_STATUS_DIO
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bFull;
    LONG bflow;
} PCI2510_STATUS_DIO, *PPCI2510_STATUS_DIO;
```

Visual Basic：

```
Type   PCI2510_STATUS_DIO
    bNotEmpty As Long
    bHalf As Long
    bFull As Long
    bflow As Long
End Type
```

LabVIEW:

请参考相关演示程序。

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

bNotEmpty　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　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。

bFull　板载 FIFO 存储器的满标志，　=TRUE 表示存储器已满，=FALSE 表示存储器未满。

bflow　板载 FIFO 存储器的溢出标志，=TRUE 已发生溢出，=FALSE 未发生溢出。

第三节、DMA 状态参数结构（PCI2510_STATUS_DMA）

```
const int MAX_SEGMENT_COUNT = 128;
```

Visual C++：

```
typedef struct _PCI2510_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
} PCI2510_STATUS_DMA, *PPCI2510_STATUS_DMA;
```

Visual Basic：

```
Type    PCI2510_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
```

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，GetDevStatusDmaDI函数使用此结构体来实时取得DI的DMA状态，GetDevStatusDmaDO函数使用此结构体来实时取得DO的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。该ID号返回值的最大范围为 0 至 63，但其具体的返回值范围为 InitDeviceDmaDI中的nSegmentCount参数决定，它的返回值为 0 至nSegmentCount-1。注意，每次调用 InitDeviceDmaDI初始化设备后，其值自动被复位至 0。

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 InitDeviceDma 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 InitDeviceDma 初始化设备后，其值自动被复位至 0。

第四节、中断状态参数结构（PCI2510_PARA_INT）

Visual C++：

```
typedef struct _PCI2510_PARA_INT
{
    LONG DI0Int;            // DI0 输入中断使能(=TRUE 使能中断, =FALSE 禁止中断)
    LONG DI1Int;            // DI1 输入中断使能(=TRUE 使能中断, =FALSE 禁止中断)
    LONG DI2Int;            // DI2 输入中断使能(=TRUE 使能中断, =FALSE 禁止中断)
    LONG DI3Int;            // DI3 输入中断使能(=TRUE 使能中断, =FALSE 禁止中断)
    LONG DI0TrigInt;        // DI0 触发控制中断(=0 上升沿触发中断, =1 下降沿中断)
    LONG DI1TrigInt;        // DI1 触发控制中断(=0 上升沿触发中断, =1 下降沿中断)
    LONG DI2TrigInt;        // DI2 触发控制中断(=0 上升沿触发中断, =1 下降沿中断)
    LONG DI3TrigInt;        // DI3 触发控制中断(=0 上升沿触发中断, =1 下降沿中断)
    LONG DIChannelChInt; //  改变通道检测产生中断
    LONG DIStsChangeInt; // DI 状态改变检测中断
    LONG PMInt;             //  匹配检测中断
    LONG TimerInt;          // 8254 定时器 2 中断
    LONG DIOverInt;         // DI FIFO 上溢中断
    LONG DOUnderInt;        // DO FIFO 下溢中断
    LONG DIStpInt;          // DI_STP 中断
    LONG DOStpInt;          // DO_STP 中断
} PCI2510 _PCI2510_PARA_INT, *PPCI2510 _PCI2510_PARA_INT;
```

Visual Basic：

```
Type    PCI2510_PARA_INT
    DI0Int As Long
    DI1Int As Long
    DI2Int As Long
    DI3Int As Long
    DI0TrigInt As Long
```

```
    DI1TrigInt As Long
    DI2TrigInt As Long
    DI3TrigInt As Long
    DIChannelChInt As Long
    DIStsChangeInt As Long
    PMInt As Long
    TimerInt As Long
    DIOverInt As Long
    DOUnderInt As Long
    DIStpInt As Long
    DOStpInt As Long
End Type
```

第五章 上层用户函数接口应用实例

第一节、怎样使用 ReadDeviceProDI_Npt函数直接取得DI数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2510 高速数字量输入输出卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[非空方式]**

第二节、怎样使用 ReadDeviceProDI_Half函数直接取得DI数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2510 80MHz DIO 卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[半满方式]**

第三节、怎样使用 DMA 方式取得 DI 数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2510 80MHz DIO 卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DMA 方式]**

第四节、怎样使用 WriteDeviceProDO函数直接取得DO数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2510 80MHz DIO 卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[PRO]**

第五节、怎样使用 DMA 方式取得 DO 数据

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨**[阿尔泰测控演示系统]**丨**[PCI2510 80MHz DIO 卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DO DMA 方式]**

第六节、怎样使用函数进行更便捷的通用开关量输入输出操作

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨[阿尔泰测控演示系统]丨[PCI2510 80MHz DIO 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[CurrentDIO]

第七节、使用中断方式实现计数器控制功能

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨[阿尔泰测控演示系统]丨[PCI2510 80MHz DIO 卡]丨[Microsoft Visual C++]丨[简易演示程序]丨[中断方式]

第八节、怎样使用函数实现计数器控制功能

Visual C++:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]丨[阿尔泰测控演示系统]丨[PCI2510 80MHz DIO 卡]丨[Microsoft Visual C++]丨[简易演示程序]丨[CNT 方式]

第六章 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“PCI2510_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
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	

第二节、PCI 内存映射寄存器操作函数原型说明

- **取得指定的指定设备寄存器组 BAR 地址**

函数原型：

Visual C++:

```
BOOL GetDeviceBar ( HANDLE hDevice,
                    __int64 pulPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByVal pulPCIBar As Long) As Boolean
```

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：

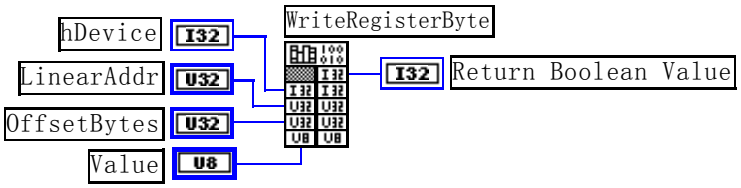
Visual C++:

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        __int64 LinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                     ByVal LinearAddr As Long, _
                                                     ByVal OffsetBytes As Long, _
                                                     ByVal Value As Byte ) As Boolean
```

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++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：

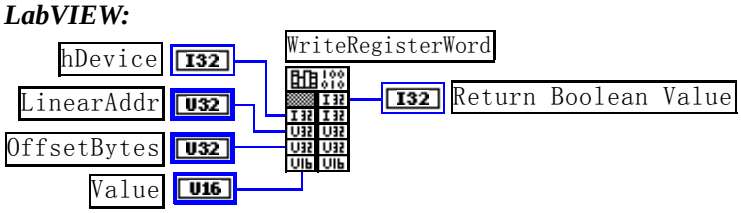
Visual C++:

```
BOOL WriteRegisterWord (HANDLE hDevice,
                        __int64 LinearAddr,
                        ULONG OffsetBytes,
                        WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                     ByVal LinearAddr As Long, _
                                                     ByVal OffsetBytes As Long, _
                                                     ByVal Value As Integer) As Boolean
```

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++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,    &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);   // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
```

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

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

函数原型：

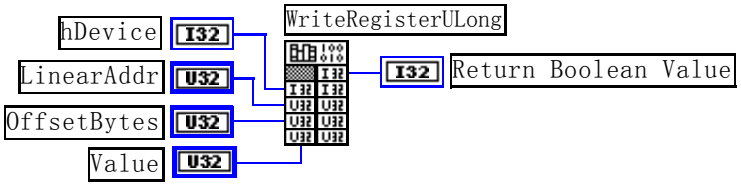
Visual C++:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         __int64 LinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                      ByVal LinearAddr As Long, _
                                                      ByVal OffsetBytes As Long, _
                                                      ByVal Value As Long) As Boolean
```

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++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：

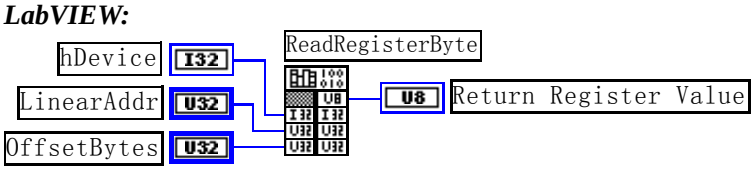
Visual C++:

```
BYTE ReadRegisterByte ( HANDLE hDevice,
                        __int64 LinearAddr,
                        ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                    ByVal LinearAddr As Long, _
                                                    ByVal OffsetBytes As Long) As Byte
```

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++程序举例:

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

Visual Basic 程序举例:

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

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

函数原型：

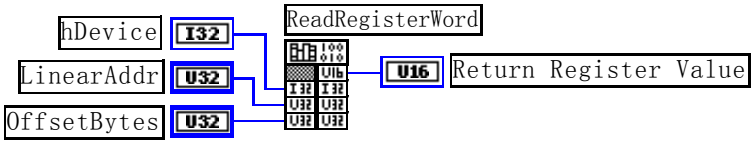
Visual C++:

```
WORD ReadRegisterWord ( HANDLE hDevice,
                        __int64 LinearAddr,
                        ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI2510_32" (    ByVal hDevice As Long, _
                                                        ByVal LinearAddr As Long, _
                                                        ByVal OffsetBytes As Long) As Integer
```

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++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：

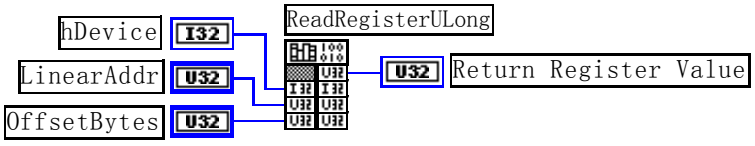
Visual C++:

```
ULONG ReadRegisterULong ( HANDLE hDevice,
                          __int64 LinearAddr,
                          ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                     ByVal LinearAddr As Long, _
                                                     ByVal OffsetBytes As Long) As Long
```

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++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**

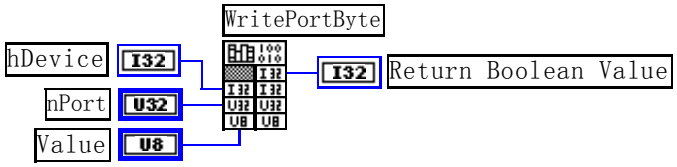
函数原型：
Visual C++:

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

Visual Basic:

```
Declare Function WritePortByte Lib "PCI2510_32" ( ByVal hDevice As Long, _
                                                  ByVal nPort As Long, _
                                                  ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice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：

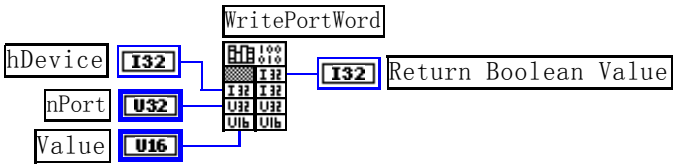
Visual C++:

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

Visual Basic:

```
Declare Function WritePortWord Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                 ByVal nPort As Long, _
                                                 ByVal Value As Integer) As Boolean
```

LabVIEW:



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

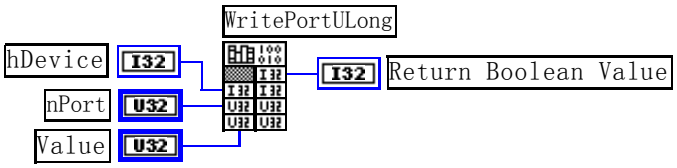
Visual C++:

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

Visual Basic:

```
Declare Function WritePortULong Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                  ByVal nPort As Long, _
                                                  ByVal Value As Long ) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice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：

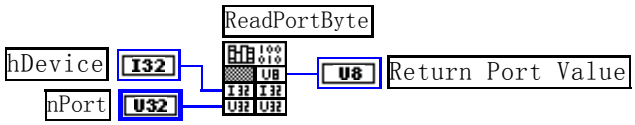
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortByte Lib "PCI2510_32" (ByVal hDevice As Long, _
                                                ByVal nPort As Long ) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice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：

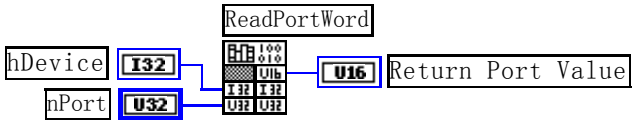
Visual C++:

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

Visual Basic:

```
Declare Function ReadPortWord Lib "PCI2510_32" ( ByVal hDevice As Long, _
                                                 ByVal nPort As Long ) As Integer
```

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice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：

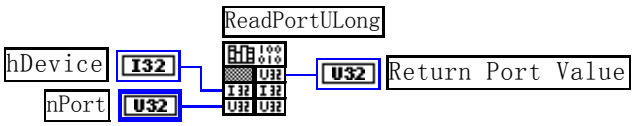
Visual C++:

```
ULONG ReadPortULong (HANDLE hDevice,
                     __int64 nPort)
```

Visual Basic:

```
Declare Function ReadPortULong Lib "PCI2510_32" ( ByVal hDevice As Long, _
                                                  ByVal nPort As Long ) As Long
```

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。

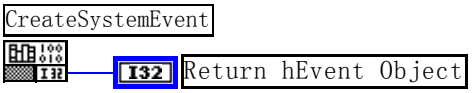
参数：
hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

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++:
HANDLE CreateSystemEvent (void)
Visual Basic:
Declare Function CreateSystemEvent Lib " PCI2510_32 " () As Long
LabVIEW:



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。
参数：无任何参数。
返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

♦ 释放内核系统事件

函数原型：
Visual C++:
BOOL ReleaseSystemEvent (HANDLE hEvent)
Visual Basic:
Declare Function ReleaseSystemEvent Lib " PCI2510 _32" (ByVal hEvent As Long) As Boolean
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。

返回值：若成功，则返回文件对象句柄。