CPCI8757A 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 CPCIxxxx_ 则被省略。如 CPCI8757A_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InitDeviceAD、InitDeviceIntAD、ReadDeviceAD_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 CPCI 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DMA方式初始化设备的AD部件，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、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

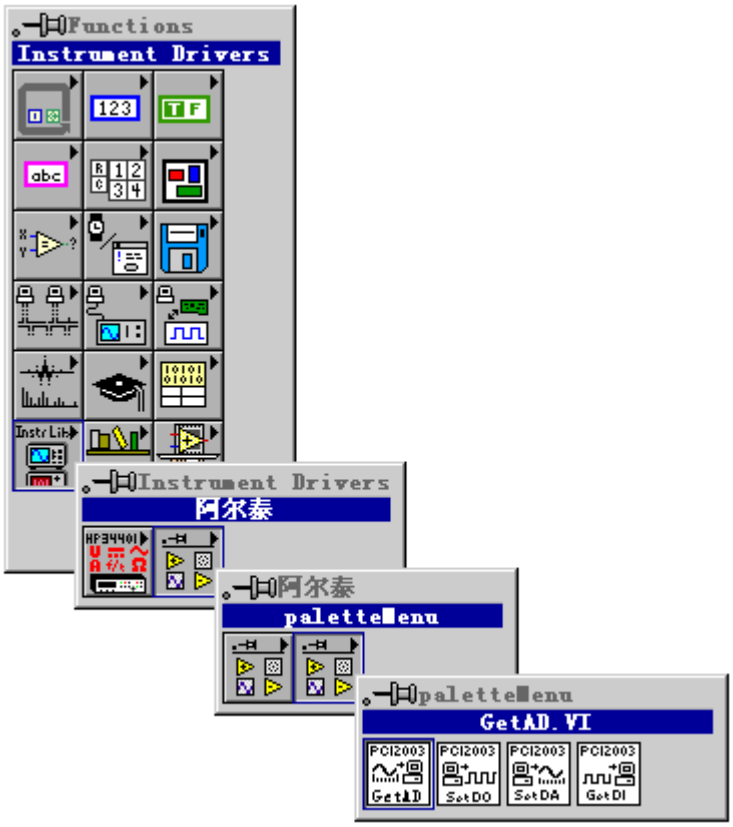
非您是作为底层用户管理设备。而WritetByte，WritetWord，WritetULong，ReadtByte， ReadtWord，ReadtULong则对CPCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 CPCI 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ReadDeviceAD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CPCI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CPCI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“CPCI8757A_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建设备对象	上层及底层用户
GetDeviceCount	取得设备总台数	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	列表系统当中的所有的该 CPCI 设备	上层及底层用户
ReleaseDevice	关闭设备,禁止传输,且释放资源	上层及底层用户
②AD 数据读取函数		
ADCalibration	设备校准	上层及底层用户
InitDeviceProAD	初始化设备	上层及底层用户
StartDeviceProAD	初始化 AD 部件准备传输	上层及底层用户
ReadDeviceProAD Npt	启动 AD 设备，开始转换	上层及底层用户
GetDevStatusProAD	连续读取当前 CPCI 设备上的 AD 数据	上层及底层用户
ReadDeviceDmaAD Half	取得当前 CPCI 设备 FIFO 半满状态	上层及底层用户
StopDeviceProAD	连续批量读取 CPCI 设备上的 AD 数据	上层及底层用户
ReleaseDeviceProAD	暂停 AD 设备	上层及底层用户
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

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\CPCI8757A\INCLUDE\CPCI8757A.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 CPCI8757A.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 CPCI8757A.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 CPCI8757A.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ReadDeviceAD接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

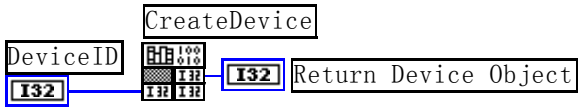
♦ 创建设备对象函数（逻辑号）

函数原型：

Visual C++：

HANDLE CreateDevice (int DeviceID = 0)

LabVIEW:



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

参数：

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 CPCI 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 CPCI8757A 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 CPCI8757A 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 CPCI 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 GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;    // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = CPCI8757A_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 = CPCI8757A_CreateDevice (DeviceLgcID)   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    MsgBox “创建设备对象失败”
    Exit Sub      ' 退出该过程
End If
    :
```

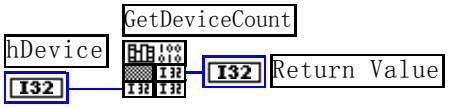
♦ 取得本计算机系统中 CPCI8757A 设备的总数量

函数原型：

Visual C++:

int GetDeviceCount (HANDLE hDevice)

LabVIEW:



功能：取得 CPCI8757A 设备的数量。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：返回系统中 CPCI8757A 的数量。

相关函数： CreateDevice 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)
```

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 GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **用对话框控件列表计算机系统中所有 CPCI8757A 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)

LabVIEW:

请参考相关演示程序。

功能：列表系统中 CPCI8757A 的硬件配置信息。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则弹出对话框控件列表所有 CPCI8757A 设备的配置情况。

相关函数： CreateDevice ReleaseDevice

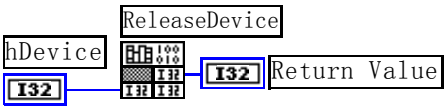
- **释放设备对象所占的系统资源及设备对象**

函数原型：

Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

LabVIEW:



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

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

相关函数： CreateDevice

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

第三节、AD 操作函数原型说明

- **设备校准**

函数原型：

Visual C++:

BOOL ADCalibration (HANDLE hDevice)

LabVIEW:

请参考相关演示程序。

功能：设备校准。

参数：

hDevice 设备校准句柄，它应由设备的CreateDevice创建。

返回值：设备校准成功，则返回 TRUE，且 AD 便被启动。否则返回 FALSE。

相关函数: CreateDevice InitDeviceProAD StartDeviceProAD ReadDeviceProAD Npt GetDevStatusProAD ReadDeviceProAD Half StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

- **初始化设备对象**

函数原型：

Visual C++:

BOOL InitDeviceProAD(HANDLE hDevice,

PCPCI8757A_PARA_AD pADPara)

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备操作就绪有关工作，如预置AD采集通道、采样频率等。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StartDeviceProAD。

参数：

hDevice 设备对象句柄，它应由设备的CreateDevice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。请参考《AD硬件参数介绍》。

返回值：如果初始化设备对象成功，则返回TRUE，且AD便被启动。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, InitDeviceProAD, StartDeviceProAD, ReadDeviceProAD Npt, GetDevStatusProAD, ReadDeviceProAD Half, StopDeviceProAD, ReleaseDeviceProAD, ReleaseDevice

♦ 启动 AD 设备(Start device AD for program mode)

函数原型：

Visual C++:

```
BOOL StartDeviceProAD ( HANDLE hDevice )
```

LabVIEW:

请参考相关演示程序。

功能：启动AD设备，它必须在调用InitDeviceProAD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, InitDeviceProAD, StartDeviceProAD, ReadDeviceProAD Npt, GetDevStatusProAD, ReadDeviceProAD Half, StopDeviceProAD, ReleaseDeviceProAD, ReleaseDevice

♦ 读取 CPCI 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++:

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

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 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。

nReadSizeWords 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。

当前错误码	功能定义
0xE1000000	其他不可预知的错误
0xE2000000	表示用户提前终止读操作

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　InitDeviceProAD　StartDeviceProAD
ReadDeviceProAD Npt　GetDevStatusProAD　ReadDeviceProAD Half
StopDeviceProAD　ReleaseDeviceProAD　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++:

```
BOOL GetDevStatusProAD (HANDLE hDevice,
                        PCPCI8757A_STATUS_AD pADStatus)
```

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 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（CPCI8757A_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 GetDevStatusProAD 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　InitDeviceProAD　StartDeviceProAD
ReadDeviceProAD Npt　GetDevStatusProAD　ReadDeviceProAD Half
StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

♦**当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型：

Visual C++:

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

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用 GetDevStatusProAD 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 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。

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第十章《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 InitDeviceProAD StartDeviceProAD
ReadDeviceProAD Npt GetDevStatusProAD ReadDeviceProAD Half
StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

◆ **暂停 AD 设备**

函数原型：

Visual C++:

BOOL StopDeviceProAD (HANDLE hDevice)

LabVIEW:

请参考相关演示程序。

功能：暂停AD设备。它必须在调用StartDevicePpoAD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StartDeviceProAD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 InitDeviceProAD StartDeviceProAD
ReadDeviceProAD Npt GetDevStatusProAD ReadDeviceProAD Half
StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

◆ **释放设备上的 AD 部件**

函数原型：

Visual C++:

BOOL ReleaseDeviceProAD (HANDLE hDevice)

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，InitDeviceProAD 必须和 ReleaseDeviceProAD 函数一一对应，即当您执行了一次InitDeviceProAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceProAD函数，以释放由InitDeviceProAD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InitDeviceProAD函数时，那些软硬件资源才可被再次使用。

相关函数: CreateDevice InitDeviceProAD ReleaseDeviceProAD
ReleaseDevice

第四节、AD 硬件参数保存与读取函数原型说明

◆ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++:

BOOL LoadParaAD(HANDLE hDevice,
PCPCI8757A_PARA_AD pADPara)

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara属于PCPCI8757A_PARA_AD的结构指针类型，它负责返回CPCI硬件参数值，关于结构指针类型PCPCI8757A_PARA_AD请参考CPCI8757A.h或CPCI8757A.Bas或CPCI8757A.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++:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PCPCI8757A_PARA_AD pADPara)
```

LabVIEW:

请参考相关演示程序。

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

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara设备硬件参数，关于CPCI8757A_PARA_AD的详细介绍请参考CPCI8757A.h或CPCI8757A.Bas或CPCI8757A.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++:

```
BOOL ResetParaAD (HANDLE hDevice,
                  PCPCI8757A_PARA_AD pADPara)
```

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CPCI8757A_PARA_AD的详细介绍请参考CPCI8757A.h或CPCI8757A.Bas或CPCI8757A.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

注意：在您编写工程应用软件时，若要更方便的保存和读取您特有的软件参数，请不防使用我们为您提供的辅助函数：SaveParaInt、 LoadParaInt、SaveParaString、LoadParaString，详细说明请参考共用函数介绍章节中的《各种参数保存和读取函数原型说明》。

第四章 硬件参数结构

第一节、AD 硬件参数结构（CPCI8757A_PARA_AD）

Visual C++:

```
typedef struct _CPCI8757A_PARA_AD
{
    LONG bChannelArray[4];    // 采样通道选择阵列，分别控制 4 个通道，=TRUE 表示该通道采样，否
```

则不采样

```
    LONG InputRange[4];        // 模拟量输入量程选择
    LONG Gains[4];             // 增益设置
    LONG Frequency;            // 采集频率，单位为 Hz, [1, 800000]
    LONG TriggerMode;          // 触发模式选择
    LONG TriggerSource;        // 触发源选择
    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 TrigLevelVolt;        // 触发电平
    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  bClockOutput;        // 允许时钟输出到 CLKOUT,=TRUE:允许时钟输出，=FALSE:禁止时钟输
出
} CPCI8757A_PARA_AD, *PCPCI8757A_PARA_AD
```

LabVIEW:

请参考相关演示程序。

该结构实在太简易了，其原因就是 CPCI 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 CPCI 设备的用户永远告别，一句话 CPCI 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
CPCI8757A_ADMODE_SEQUENCE	0x00	连续采集模式
CPCI8757A_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。

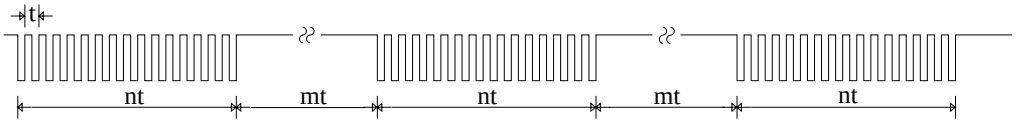


图 4.1

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

FirstChannel　AD采样首通道，其取值范围为[0, 15]，它应等于或小于LastChannel参数。
LastChannel　AD采样末通道，其取值范围为[0, 15]，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[31,250KHz]。

注意：

在内时钟(即ClockSource = CPCI8757A_CLOCKSRC_IN)方式下：

若连续采集(即ADMode = CPCI8757A_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= CPCI8757A_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 = CPCI8757A_CLOCKSRC_OUT)方式下：

若连续采集(即ADMode = CPCI8757A_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= CPCI8757A_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, 65535]。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。

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
CPCI8757A_INPUT_0_P10000mV	0x00	0～10000mV
CPCI8757A_INPUT_0_P5000mV	0x01	0～5000mV
CPCI8757A_INPUT_N10000_P10000mV	0x02	±10000mV
CPCI8757A_INPUT_N5000_P5000mV	0x03	±5000mV
CPCI8757A_INPUT_N2500_P2500mV	0x04	±2500mV

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
CPCI8757A_TRIGMODE_SOFT	0x00	软件触发(属于内触发)
CPCI8757A_TRIGMODE_POST	0x01	硬件后触发(属于外触发)

TriggerSource 触发源选择

常量名	常量值	功能定义
CPCI8757A_TRIGSRC_ATR	0x00	选择外部 ATR 作为触发源
CPCI8757A_TRIGSRC_DTR	0x01	选择外部 DTR 作为触发源

TrigLevelVolt 触发电平(0~10000mV)。

TriggerType　AD 触发类型。

常量名	常量值	功能定义
CPCI8757A_TRIGTYPE_EDGE	0x00	边沿触发
CPCI8757A_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
CPCI8757A_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
CPCI8757A_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
CPCI8757A_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：CPCI8757A_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。

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
CPCI8757A_CLOCKSRC_IN	0x00	内部时钟定时触发
CPCI8757A_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 = CPCI8757A_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 = CPCI8757A_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。

Gains[X] 硬件增益选项

常量名	常量值	功能定义
CPCI8757A_GAINS_1MULT	0x00	1 倍增益
CPCI8757A_GAINS_2MULT	0x01	2 倍增益
CPCI8757A_GAINS_4MULT	0x02	4 倍增益
CPCI8757A_GAINS_8MULT	0x03	8 倍增益

第二节、用于 AD 采样的实际硬件参数 CPCI8757A_STATUS_AD）

Visual C++：

```
typedef struct _CPCI8757A_STATUS_AD
{
    LONG bNotEmpty;          // 板载 FIFO 存储器的非空标志，=TRUE 非空， = FALSE 空
    LONG bHalf;              // 板载 FIFO 存储器的半满标志，=TRUE 半满以上， = FALSE 半满以下
    LONG bOverflow;          // 板载 FIFO 存储器的溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出
} CPCI8757A_STATUS_AD, *PCPCI8757A_STATUS_AD;
```

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，GetDevStatusDmaAD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。该ID号返回值的最大范围为 0 至 127，但其具体的返回值范围为InitDeviceDmaAD中的nSegmentCount参数决定，它的返回值为 0 至nSegmentCount-1。注意，每次调用InitDeviceDmaAD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InitDeviceDmaAD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InitDeviceDmaAD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 = (20000.00/8620) * (ADBuffer[0] &0x1FFF) – 10000.00	[-10000, +9997.55]
±5000mV	Volt = (10000.00/8620) * (ADBuffer[0] &0x1FFF) – 5000.00	[-5000, +4998.77]
±2500mV	Volt = (5000.00/8620) * (ADBuffer[0]&0x1FFF) –2500.00	[-2500, +2499.38]
0~10000mV	Volt = (10000.00/8620) * (ADBuffer[0] &0x1FFF)	[0，+9998.77]
0~5000mV	Volt = (1000.00/8620) * (ADBuffer[0] &0x1FFF)	[0，+4998.77]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++：

```
Lsb = (ADBuffer[0])&0x1FFF;
```

Volt = (20000.00/8620) * Lsb -10000.00;

Visual Basic：

Lsb = (ADBuffer [0]) And &H1FFF

Volt = (20000.00/8620) * 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(假如FirstChannel=0, LastChannel=1)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	…

四通道采集(假如FirstChannel=0, LastChannel=3)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	…

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceAD_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 HeadSizeBytes;          // 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)
    CPCI8757A_PARA_AD ADPara;    // 保存硬件参数
    LONG StaticOverFlow;         // 同批文件识别码
    LONG HeadEndFlag;            // 文件结束位
    CPCI8757A_STATUS_AD ADStatus;
} 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“CPCI8757A_”）

函数名	函数功能	备注
① CPCI 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、CPCI 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 pbPCIBar[6])
```

LabVIEW:

请参考相关演示程序。

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbPCIBar 返回 PCI BAR 所有地址,具体 PCI BAR 中有多少可用地址请看硬件说明书。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

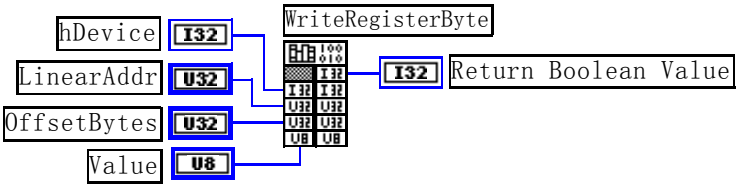
- **以单字节（即 8 位）方式写 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节（即 8 位）方式写 CPCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr CPCI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)
    :
```

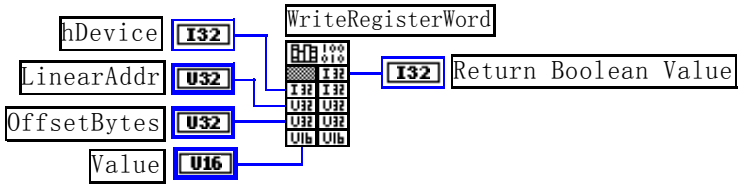
◆ **以双字节（即 16 位）方式写 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以双字节（即 16 位）方式写 CPCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr　CPCI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)
    :
```

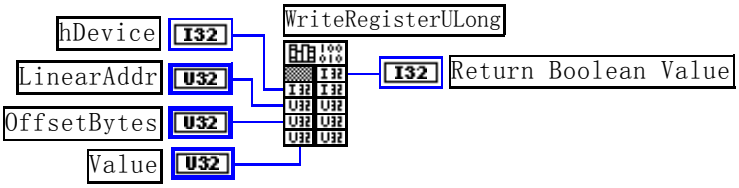
◆ **以四字节（即 32 位）方式写 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以四字节（即 32 位）方式写 CPCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr CPCI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)
    :
```

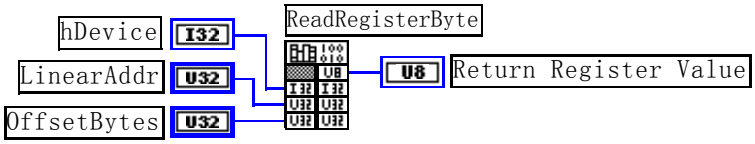
◆ **以单字节（即 8 位）方式读 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节（即 8 位）方式读 CPCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr　CPCI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 CPCI 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)
    :
```

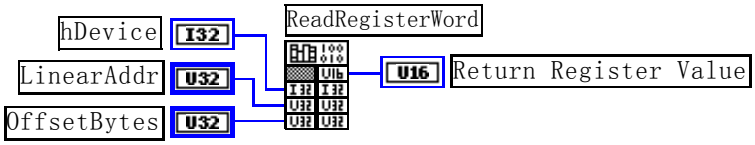
◆ **以双字节（即 16 位）方式读 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以双字节（即 16 位）方式读 CPCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr CPCI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 CPCI 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)
    :
```

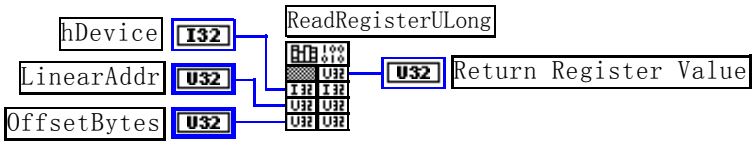
♦ **以四字节（即 32 位）方式读 CPCI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以四字节（即 32 位）方式读 CPCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr CPCI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 CPCI 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)
    :
```

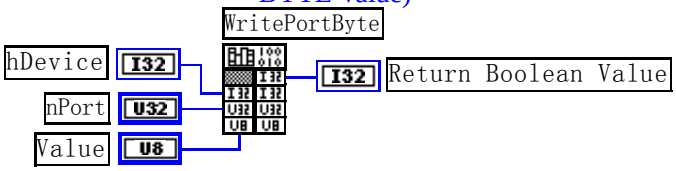
第三节、IO 端口读写函数原型说明

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**

Visual C++:

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



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pbPort 设备的 I/O 端口号。

Value 写入由 pbPort 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。

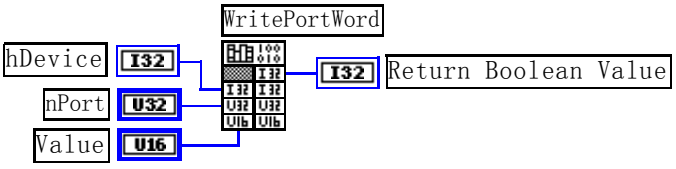
相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

Visual C++:

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

LabVIEW:



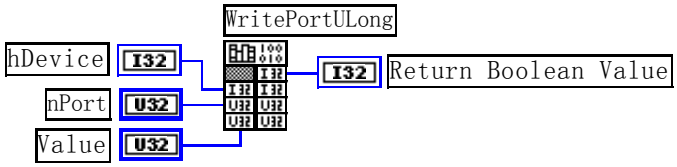
功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
Value　写入由 pbPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

Visual C++:

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

LabVIEW:



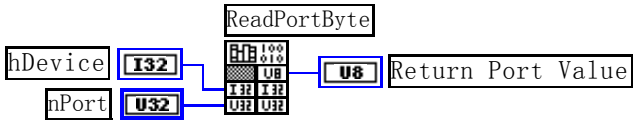
功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice　设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
Value　写入由 pbPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

Visual C++:

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

LabVIEW:



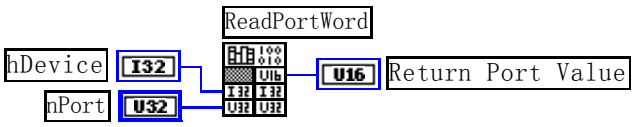
功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
返回值：返回由 pbPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

Visual C++:

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

LabVIEW:



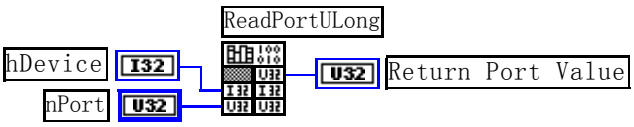
功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pbPort　设备的 I/O 端口号。
返回值：返回由 pbPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

◆ **以四字节(32Bit)方式读 I/O 端口**

Visual C++:

```
ULONG ReadPortULong(HANDLE hDevice,
                    __int64 pbPort,
                    ULONG offserBytes)
```

LabVIEW:



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

第四节、线程操作函数原型说明

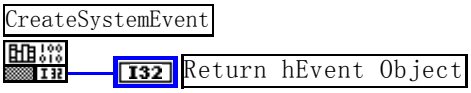
◆ **创建内核系统事件**

函数原型：

Visual C++:

```
HANDLE CreateSystemEvent(void)
```

LabVIEW:



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

◆ **释放内核系统事件**

函数原型：

Visual C++:

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

LabVIEW:

请参见相关演示程序。
功能：释放系统内核事件对象。
参数：hEvent　被释放的内核事件对象。它应由CreateSystemEvent成功创建的对象。
返回值：若成功，则返回 TRUE。

相关函数：CreateSystemEvent　　ReleaseSystemEvent