PCIE2331 64 路隔离通道数字量输入卡

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 PCIExxxx_ 则被省略。如 PCIE2331_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。而底层用户函数如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 PCIE 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SetDeviceDI函数可用实现开关量的输入等。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何实现开关量的简便操作

当您有了hDevice设备对象句柄后，便可用GetDeviceDI函数实现开关量的输入操作，其各路开关量的输入状态由其pbyDISts[64]中的相应元素决定。

第四节、哪些函数对您不是必须的

公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr等函数您可完全不必理会，除非您是作为底层用户管

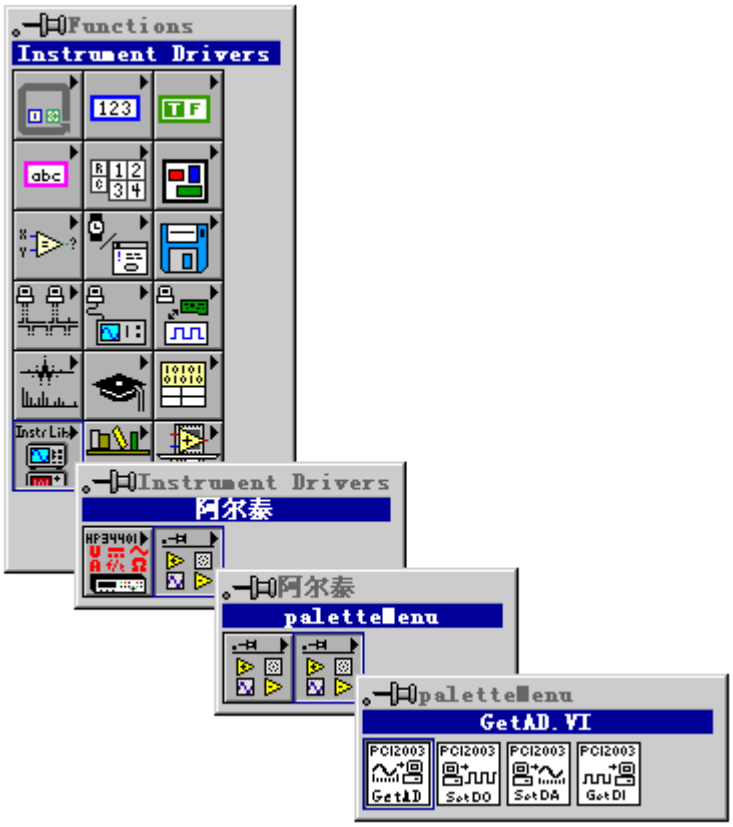
理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对PCIE用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PCIE 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用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PCIE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PCIE的资源配置空间、PNP即插即用管理，而只须用GetDeviceBar函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

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



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCIE2331_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PCIE 设备对象(用设备逻辑号)	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
GetDeviceCount	取得同一种 PCIE 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PCIE 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCIE 总线设备对象	上层及底层用户
② 中断操作函数		
InitDeviceInt	初始化中断	上层用户
GetDeviceIntCount	在中断初始化后，取得中断服务程序产生的次数	上层用户
GetDeviceIntSrc	在中断初始化后，用它取得中断源	上层用户
ReleaseDeviceInt	释放中断资源	上层用户
③ DIO 开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户

使用需知：

Visual C++：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\Art\PCIE2331\INCLUDE\PCIE2331.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 PCIE2331.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句：

#include “PCIE2331.H”

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 PCIE2331.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，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

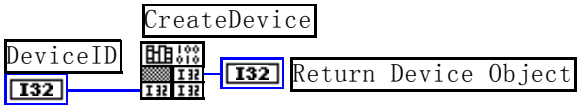
♦ 创建设备对象函数（逻辑号）

函数原型：

Visual C++：

HANDLE CreateDevice (int DeviceID = 0)

LabVIEW：



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

参数：DeviceID 设备 ID 标识号。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ReleaseDevice

Visual C++程序举例：

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

Visual Basic 程序举例：

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

：

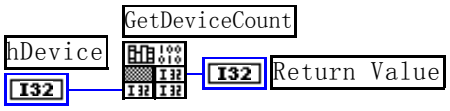
- **取得本计算机系统中 PCIE2331 设备的总数量**

  函数原型：

  Visual C++:

  int GetDeviceCount (HANDLE hDevice)

  LabVIEW：



  功能：取得 PCIE2331 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 PCIE2331 的数量。

  相关函数：CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

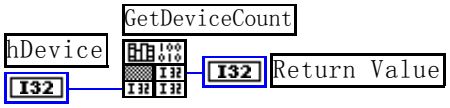
- **取得物理 ID**

  函数原型：

  Visual C++:

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

  LabVIEW：



  功能：取得 PCIE2331 设备的数量。

  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 PCIE2331 设备的配置情况。

  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 PCIE2331 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 PCIE2331 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 PCIE2331 设备的配置情况。

  相关函数：CreateDevice ReleaseDevice

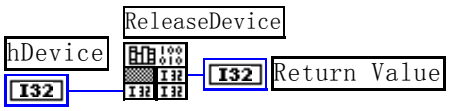
- **释放设备对象所占的系统资源及设备对象**

  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、中断操作函数原型说明

◆ **初始化中断**

函数原型：
Visual C++:

```
BOOL InitDeviceInt (HANDLE hDevice,
                    HANDLE hEventInt,
                    PPCIE2331_PARA_INT pIntPara,
                    PPCIE2331_INT_MODE pIntMode)
```

LabVIEW:
请参考相关演示程序。

功能：它负责初始化设备对象的硬件中断的方式工作。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
hEventInt中断事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEventInt事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。
PIntPara 中断源使能
PIntMode 各中断源触发中断模式设置
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。
相关函数：CreateDevice GetDeviceCount GetDeviceCurrentID
ListDeviceDlg ReleaseDevice

◆ **取得中断服务程序产生的次数**

函数原型：
Visual C++:

```
LONG GetDeviceIntCount (HANDLE hDevice)
```

LabVIEW:
请参考相关演示程序。

功能：在中断初始化后，用它取得中断服务程序产生的次数。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：若成功，返回取得中断服务程序产生的次数，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。
相关函数： CreateDevice InitDeviceInt GetDeviceIntCount

ReleaseDeviceInt GetDeviceIntSrc ReleaseDevice

◆ **在中断初始化后，用它取得中断源**

函数原型：

Visual C++:

BOOL GetDeviceIntSrc(HANDLE hDevice, PPCIE2331_INT_STATUS pSrcPara)

LabVIEW:

请参考相关演示程序。

功能：在中断初始化后，用它取得中断源。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice InitDeviceInt GetDeviceIntCount
ReleaseDeviceInt GetDeviceIntSrc ReleaseDevice

◆ **释放中断资源**

函数原型：

Visual C++:

BOOL ReleaseDeviceInt (HANDLE hDevice)

LabVIEW:

请参考相关演示程序。

功能：释放中断资源。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice InitDeviceInt GetDeviceIntCount
ReleaseDeviceInt GetDeviceIntSrc ReleaseDevice

第四节、DIO 数字量输入输出开关量操作函数原型说明

◆ **开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   BYTE pbyDISts[64])
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCIE 设备上的输入开关量状态读入内存。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbyDISts[64] 开关状态

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

◆ **以上函数调用一般顺序**

① CreateDevice
② GetDeviceDI
③ ReleaseDevice

第四章 硬件参数结构

第一节、中断信息控制参数结构（PCIE2331_PARA_INT）

Visual C++：

```
typedef struct _PCIE2331_PARA_INT    // 中断信息控制参数
{
    LONG bINTDI0;     // DI0 触发中断,=TRUE DI0 中断使能, = FALSE DI0 中断禁止
    LONG bINTDI16;    // DI8 触发中断,=TRUE DI16 中断使能, = FALSE DI16 中断禁止
    LONG bINTDI32;    // DI16 触发中断,=TRUE DI32 中断使能, = FALSE DI32 中断禁止
    LONG bINTDI48;    // DI24 触发中断,=TRUE DI48 中断使能, = FALSE DI48 中断禁止
} PCIE2331_PARA_INT, *PPCIE2331_PARA_INT;
```

第二节、触发中断模式设置参数结构（PCIE2331_INT_MODE）

Visual C++：

```
typedef struct _PCIE2331_INT_MODE
{
    LONG bINTModeDI0;        // DIO 中断模式,=TRUE DI0 上升沿触发, = FALSE DI0 下降沿触发
    LONG bINTModeDI16;       // DIO 中断模式,=TRUE DI16 上升沿触发, = FALSE DI16 下降沿触发
    LONG bINTModeDI32;       // DIO 中断模式,=TRUE DI32 上升沿触发, = FALSE DI32 下降沿触发
    LONG bINTModeDI48;       // DIO 中断模式,=TRUE DI48 上升沿触发, = FALSE DI48 下降沿触发
} PCIE2331_INT_MODE, *PPCIE2331_INT_MODE;
```

第三节、中断状态参数结构（PCIE2331_INT_STATUS）

Visual C++：

```
typedef struct _PCIE2331_INT_STATUS     // 中断状态
{
    LONG pbyDI0IntSts;        // DI0 触发中断,=TRUE DI0 中断, = FALSE DI0 未中断
    LONG pbyDI16IntSts;       // DI8 触发中断,=TRUE DI16 中断, = FALSE DI16 未中断
    LONG pbyDI32IntSts;       // DI16 触发中断,=TRUE DI32 中断, = FALSE DI32 未中断
    LONG pbyDI48IntSts;       // DI24 触发中断,=TRUE DI18 中断, = FALSE DI48 未中断
} PCIE2331_INT_STATUS, *PPCIE2331_INT_STATUS;
```

第五章 上层用户函数接口应用实例

第一节、怎样使用GetDeviceDI函数进行更便捷的数字开关量输入操作

Visual C++:

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

[程序]|[阿尔泰测控演示系统]|[PCIE2331 64 路隔离通道 DI 卡及中断卡]|[Microsoft Visual C++]|[简易代码演示]

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“PCIE2331_”）

函数名	函数功能	备注
① PCIE 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
GetDevVersion	获取设备固件及程序版本	
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 线程操作函数		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	用于线程同步

第二节、PCIE 内存映射寄存器操作函数原型说明

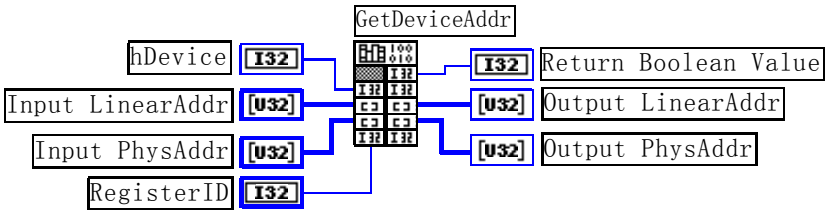
♦ 取得指定的指定设备寄存器组 BAR 地址

函数原型：

Visual C++:

BOOL GetDeviceBar (HANDLE hDevice, __int64 pbPCIBar[6])

LabVIEW:



功能：取得 PCIE 设备指定的内存映射寄存器的线性地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

返回值：如果执行成功，则返回 TRUE，否则返回 FALSE。

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

Visual C++程序举例:

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

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
```

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

♦ **获取设备固件及程序版本**

函数原型：

Visual C++:

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

LabVIEW:

请参考相关演示程序。

功能：获取设备固件及程序版本。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pulFmwVersion 固件版本。

pulDriverVersion 驱动版本。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

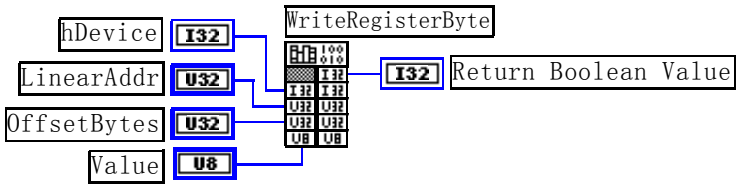
♦ **以单字节（即 8 位）方式写 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节（即 8 位）方式写 PCIE 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCIE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　GetDeviceBar　GetDevVersion　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　WriteRegisterByte　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)
    :
```

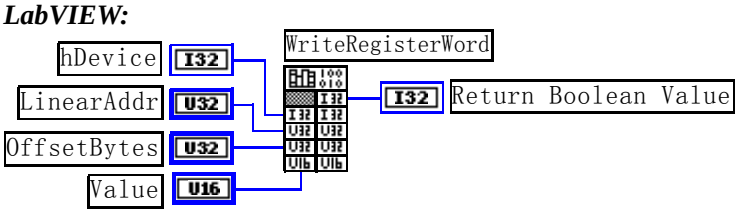
- **以双字节（即 16 位）方式写 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以双字节（即 16 位）方式写 PCIE 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr　PCIE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　GetDeviceBar　GetDevVersion　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　WriteRegisterByte　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)
    :
```

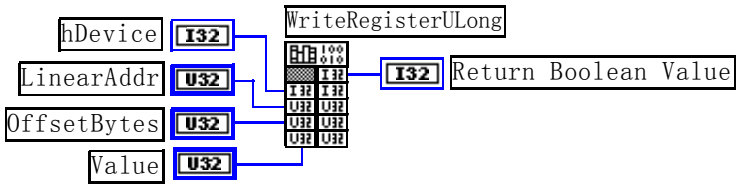
- **以四字节（即 32 位）方式写 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以四字节（即 32 位）方式写 PCIE 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCIE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　GetDeviceBar　GetDevVersion　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　WriteRegisterByte　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)
    :
```

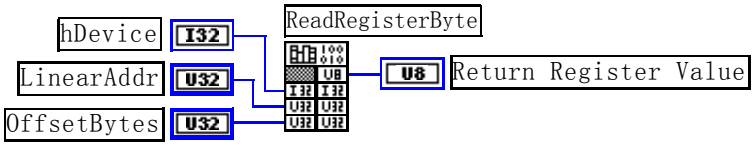
♦ **以单字节（即 8 位）方式读 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节（即 8 位）方式读 PCIE 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr　PCIE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　GetDeviceBar　GetDevVersion
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　WriteRegisterByte
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 PCIE 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)
    :
```

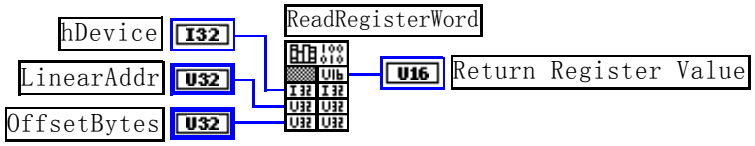
♦ **以双字节（即 16 位）方式读 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以双字节（即 16 位）方式读 PCIE 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCIE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　GetDeviceBar　GetDevVersion　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　WriteRegisterByte　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 PCIE 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)
    :
```

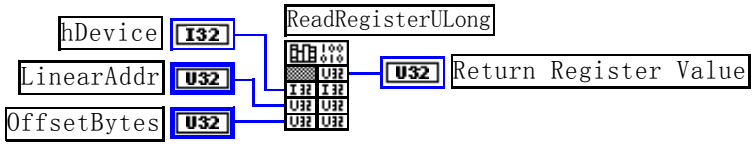
- **以四字节（即 32 位）方式读 PCIE 内存映射寄存器的某个单元**

函数原型：

Visual C++:

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

LabVIEW:



功能：以四字节（即 32 位）方式读 PCIE 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr　PCIE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 GetDeviceBar GetDevVersion WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong WriteRegisterByte 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 PCIE 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

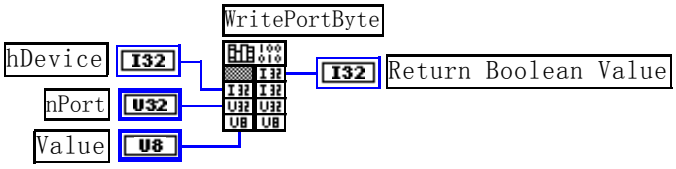
- **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++:

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

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pPort 设备的 I/O 端口号。

Value 写入由 pPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

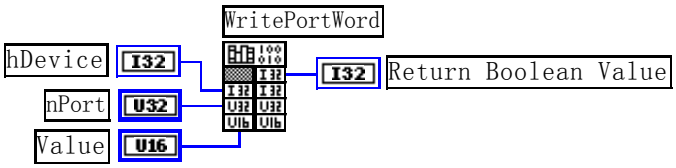
- **以双字节(16Bit)方式写 I/O 端口**

函数原型：

Visual C++:

BOOL WritePortWord (HANDLE hDevice,
__int64 pPort,
WORD Value)

LabVIEW:

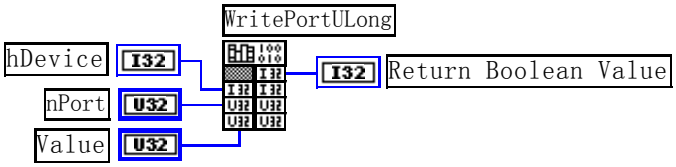


功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pPort　设备的 I/O 端口号。
Value　写入由 pPort 指定端口的值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

函数原型：
Visual C++:
BOOL WritePortULong(HANDLE hDevice,
__int64 pPort,
ULONG Value)

LabVIEW:

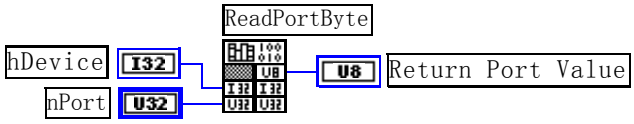


功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pPort　设备的 I/O 端口号。
Value　写入由 pPort 指定端口的值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

函数原型：
Visual C++:
BYTE ReadPortByte(HANDLE hDevice, __int64 pPort)

LabVIEW:



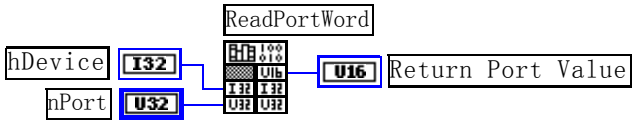
功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。

pPort 设备的 I/O 端口号。
返回值：返回由 pPort 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 pPort)

LabVIEW:

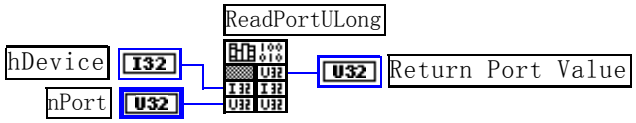


功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pPort 设备的 I/O 端口号。
返回值：返回由 pPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式读 I/O 端口**

函数原型：
Visual C++:
ULONG ReadPortULong(HANDLE hDevice, __int64 pPort)

LabVIEW:



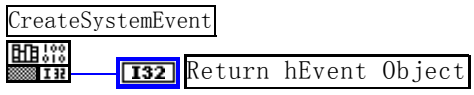
功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pPort 设备的 I/O 端口号。
返回值：返回由 pPort 指定端口的值。
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

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