PXI2391 计数器卡

驱动程序使用手册

北京阿尔泰科技发展有限公司

产品研发部修订

2015

V6.00.00

阿尔泰科技

www.art-control.com

关于本手册

本手册为阿尔泰科技推出的 PXI2391 多功能计数器卡驱动程序使用手册，其中包括版权信息与命名约定、使用纲要、各功能操作流程介绍、PXI 设备操作函数接口介绍、上层用户函数接口应用实例、共用函数介绍、修改历史等。

文档版本：V6.00.00

目录

1 版权信息与命名约定

1.1 版权信息

1.2 命名约定

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 PXIxxxx_则被省略。如 PXI2391_CreateDevice 则写为 CreateDevice。

表 1-2-1：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			

以上规则不局限于该产品。

2 使用纲要

2.1 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 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。

2.2 如何管理 PXI 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控制权。然后将此句柄作为参数传递给其他函数，如 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。

2.3 如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 SetDeviceDO 函数实现开关量的输出操作，其各路开关量的输出状态由其 bDOISts[8]中的相应元素决定。

2.4 哪些函数对您不是必须的

公共函数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 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 PXI 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、Win7 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。

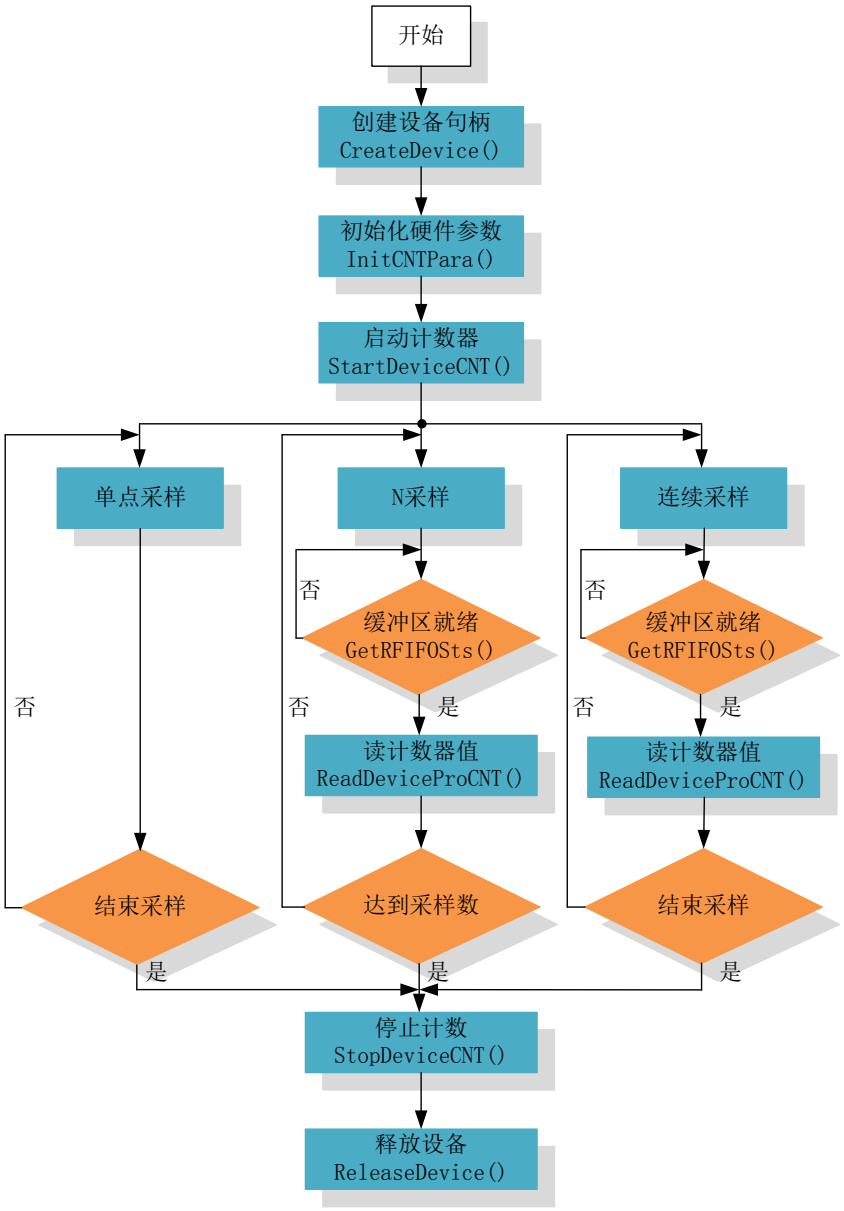
3 各功能操作流程介绍

本卡可进行边沿、编码器、半周期、脉宽测量、两边沿间隔、频率/周期、脉冲输出、开关量输入、开关量输出、双向 DIO 操作。

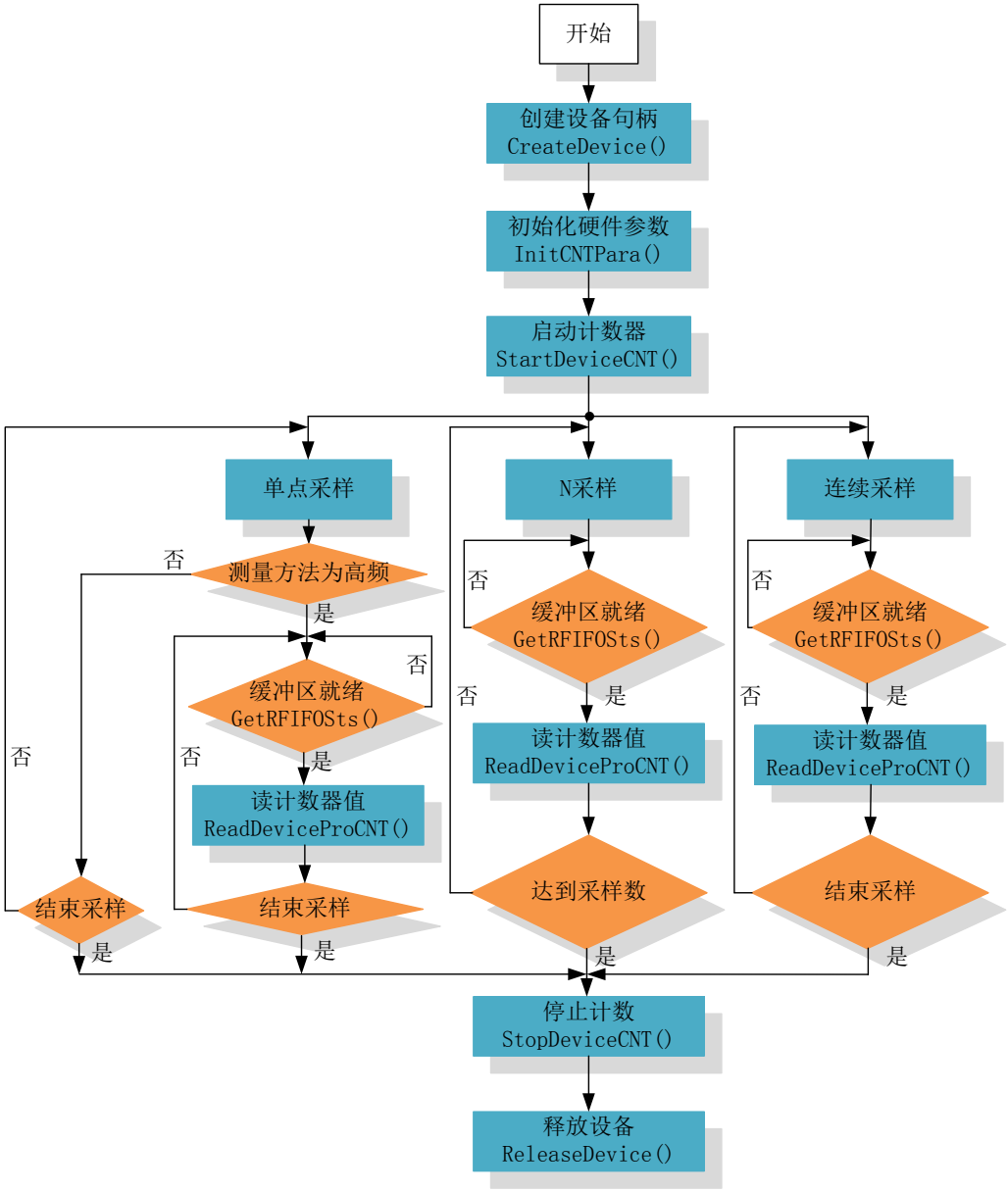
表 3：各功能对应的简易程序名

功能	简易程序名
边沿计数	CNT
半周期	HalfCycle
脉宽测量	PulseWidth
编码器	CodeCnt
脉冲输出	PulseOutput
两边沿间隔	TwoDdgeWidth
频率/周期	MeasureCycle
开关量输入	DI
开关量输出	DO
DIO	DIO

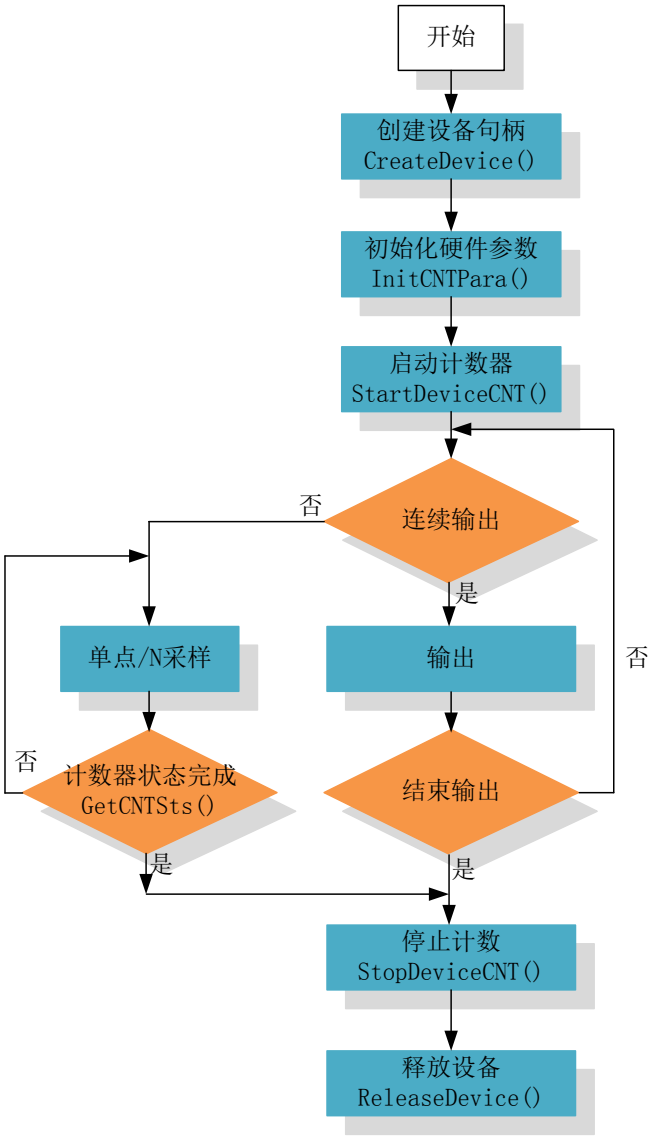
3.1 边沿、编码器、半周期、脉宽、两边沿间隔和开关量输入操作流程



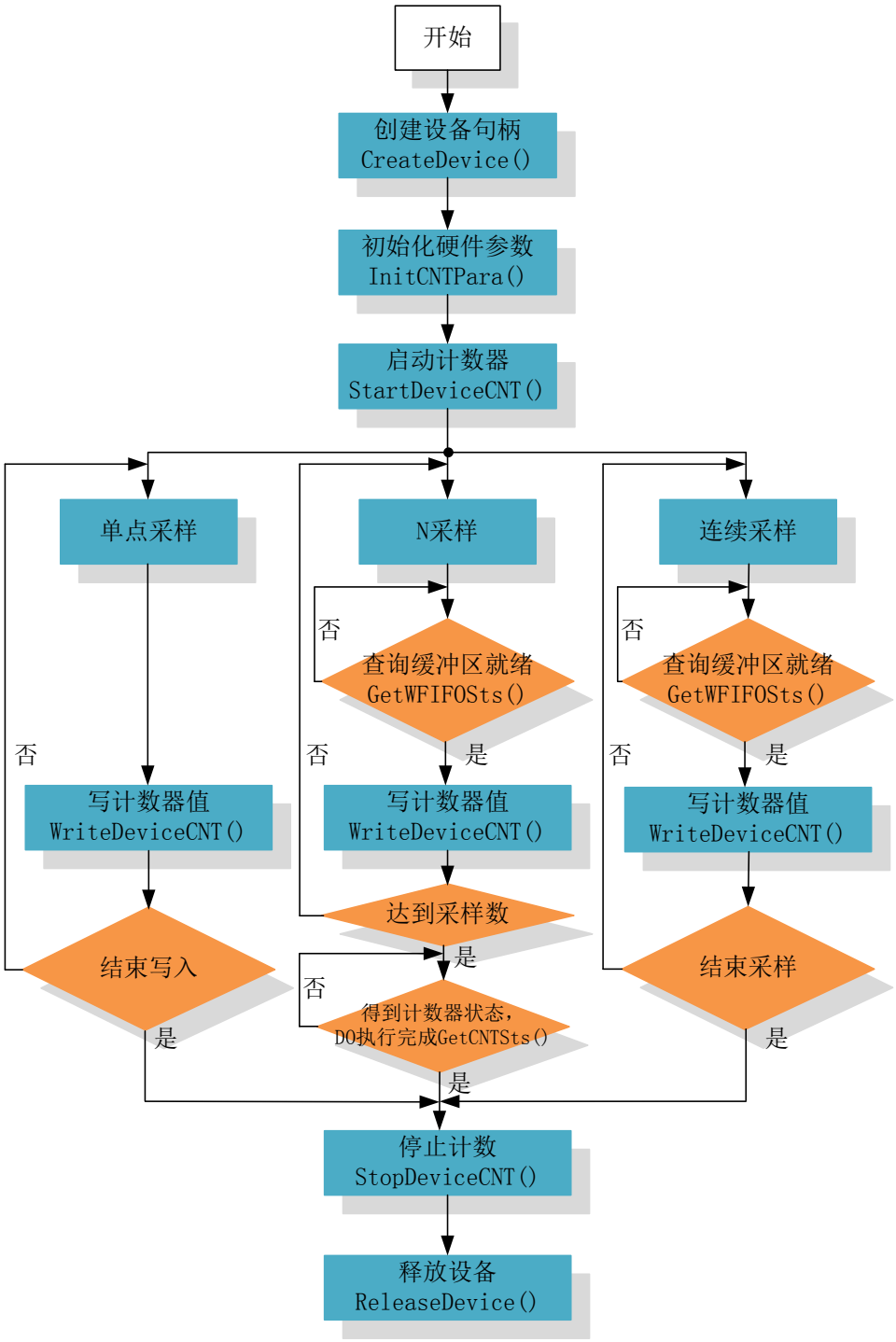
3.2 频率/周期操作流程


开始
创建设备句柄
CreateDevice()
初始化硬件参数
InitCNTPara()
启动计数器
StartDeviceCNT()
单点采样
测量方法为高频
否
是
缓冲区就绪
GetRFIFOSts()
否
是
读计数器值
ReadDeviceProCNT()
结束采样
否
是
否
N采样
缓冲区就绪
GetRFIFOSts()
否
是
读计数器值
ReadDeviceProCNT()
达到采样数
否
是
连续采样
缓冲区就绪
GetRFIFOSts()
否
是
读计数器值
ReadDeviceProCNT()
结束采样
否
是
停止计数
StopDeviceCNT()
释放设备
ReleaseDevice()

3.3 脉冲输出操作流程

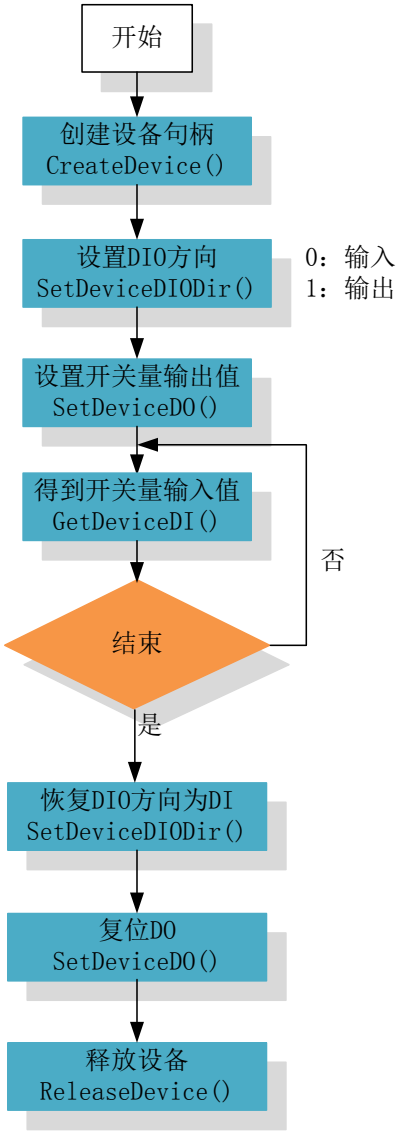

开始
创建设备句柄
CreateDevice()
初始化硬件参数
InitCNTPara()
启动计数器
StartDeviceCNT()
连续输出
否
是
单点/N采样
输出
否
否
计数器状态完成
GetCNTSts()
结束输出
是
是
停止计数
StopDeviceCNT()
释放设备
ReleaseDevice()

3.4 开关量输出操作流程



N 采样/连续采样需要从 GetWFIFOSts(HANDLE hDevice, PLONG lWFIFOSts, PLONG lWriteDataCount，ULONG ulChannel)参数 lWriteDataCount 中得到每次写入缓冲区的 DO 数据个数，即硬件缓冲区需要写入的数据个数，取得此个数后，即可进行 WriteDeviceProCNT 操作。

3.5 DIO 操作流程



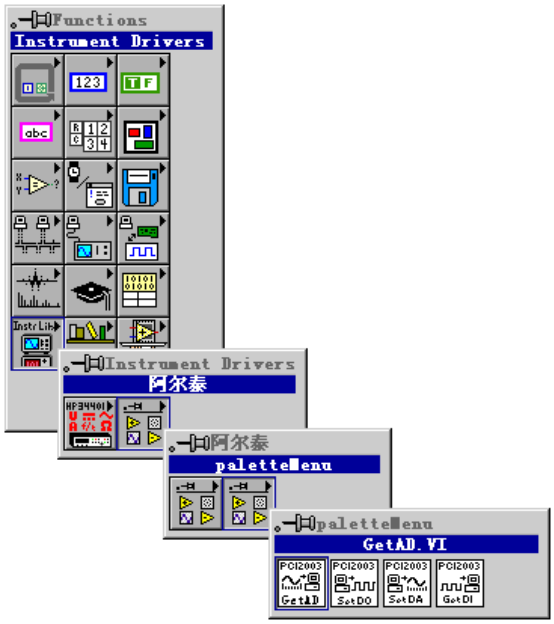
用户可根据需要循环进行开关量输出值的设置。

4 PXI 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，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 PXI 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 PXI 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++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

4.1 设备驱动接口函数总列表

表 4-1-1：驱动接口函数总列表（每个函数省略了前缀“PXI2391_”）

函数名	函数功能	备注
① 设备对象管理函数		
CreateDevice	用逻辑号创建设备对象	上层及底层用户
GetDeviceCount	取得设备总台数	上层及底层用户
GetDeviceCurrentID	取得当前设备相应的 ID 号	上层用户
ListDeviceDlg	以对话框窗体方式列表系统当中的所有的该 PCI 设备	上层及底层用户
ReleaseDevice	释放设备对象	上层用户
② 计数器控制函数		
InitCNTPara	初始化计数器参数	上层用户
StartDeviceCNT	计数器使能	上层用户
GetCNTSts	计数器工作状态	上层用户
GetRFIFOSts	获取读数据缓冲区的状态	上层用户
GetWFIFOSts	获取写数据缓冲区的状态	上层用户
ReadDeviceProCNT	读取数据	上层用户
WriteDeviceProCNT	写入数据	上层用户
StopDeviceCNT	计数器禁止	上层用户
③ 计数器硬件参数操作函数		
SaveParaCNT	将当前的 CNT 采样参数保存至系统中	上层用户
LoadParaCNT	将 CNT 采样参数从系统中读出	上层用户
ResetParaCNT	将 CNT 采样参数恢复至出厂默认值	上层用户
④ 数字 I/O 输入输出函数		
SetDeviceDIODir	设置开关量输入输出方向	上层用户
GetDeviceDI	取得数字量状态	上层用户
SetDeviceDO	输出数字量状态	上层用户

使用需知：

Visual C++：

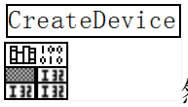
① 要使用如下函数关键的问题是必须在您的源程序中包含如下语句：#include “C:\Art\PXI2391\INCLUDE\PXI2391.H”（ 采用默认路径和默认板号），用户需根据自己的板号和安装情况确定 PXI2391.H 文件的正确路径。

② 用户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 “PXI2391.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：

（1）、在 LabView 中打开 PXI2391.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。

（2）、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。

（3）、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。

4.2 设备对象管理函数原型说明

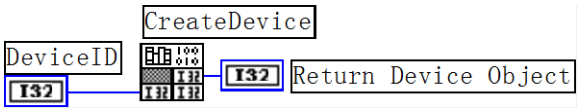
♦ **用逻辑号创建设备对象**

函数原型：

Visual C++：

```
HANDLE CreateDevice(int DeviceLgcID = 0);        // 用逻辑号创建设备对象
```

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 设备 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 DeviceLgcID 标识值为名称后缀的标识符来确认和管理该设备。默认值为 0。

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。

相关函数： CreateDevice　　ReleaseDevice

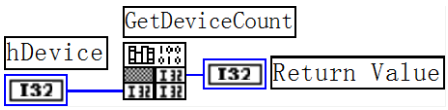
♦ **取得同一种 PXI 设备的总台数**

函数原型：

Visual C++:

```
Int GetDeviceCount(HANDLE hDevice);        // 取得设备总台数
```

LabVIEW：



功能：取得 PXI2391 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 PXI2391 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，     // 取得当前设备相应的 ID 号
                         PLONG DeviceLgcID，
                         PLONG DevicePhysID)
```

LabVIEW：

请参考相关演示程序。

功能：取得指定设备当前逻辑 ID 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]，具体值由卡上的拔码器 DID1 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。

相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **用对话框控件列表同一种 PXI 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)　　// 以对话框窗体方式列表系统当中的所有的该 PCI 设备

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PXI2391 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 PXI2391 设备的配置情况。

相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

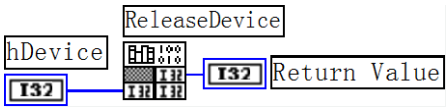
♦ **释放设备对象所占的系统资源及设备对象**

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。

相关函数： CreateDevice　ReleaseDevice

CreateDevice 函数必须和 ReleaseDevice 函数一一对应。即当您执行了一次 CreateDevice 后，再执行这些函数前，必须先执行一次 ReleaseDevice 函数，以释放由 CreateDevice 占用的系统软硬件资源，如系统内存等。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。

4.3 计数器控制函数原型说明

♦初始化计数器参数

函数原型：

Visual C++:

```
BOOL InitCNTPara(                          // 初始化计数器参数
                HANDLE hDevice,            // 设备对象句柄,它由 CreateDevice 函数创建
                PPXI2391_PARA_CNT pCNTPara,    // 计数器参数
                ULONG ulChannel);          // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：初始化计数器参数。

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。

相关函数：CreateDevice

♦计数器使能

函数原型：

Visual C++:

```
BOOL StartDeviceCNT(                       // 计数器使能
                HANDLE hDevice,            // 设备对象句柄,它由 CreateDevice 函数创建
                ULONG ulChannel);          // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：计数器使能。

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。

相关函数：CreateDevice

♦计数器工作状态

函数原型：

Visual C++:

```
BOOL GetCNTSts(                        //计数器工作状态
                HANDLE hDevice,            // 设备对象句柄,它由 CreateDevice 函数创建
                PLONG lCNTSts,             // 1:忙  0:空闲
                ULONG ulChannel);          // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：计数器工作状态。

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。

♦获取读数据缓冲区的状态

函数原型：

Visual C++:

```
BOOL GetRFIFOSts(                       // 获取读数据缓冲区的状态
                HANDLE hDevice,         // 设备对象句柄,它由 CreateDevice 函数创建
                PLONG lRFIFOSts,        // 1:溢出  0:未溢出
                PLONG lReadDataCount,   // 返回读数据缓冲区的数据个数
                ULONG ulChannel);       // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：获取读数据缓冲区的状态。

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。

相关函数：CreateDevice

♦获取写数据缓冲区的状态

函数原型：

Visual C++:

```
BOOL GetWFIFOSts(                       // 获取写数据缓冲区的状态
                HANDLE hDevice,         // 设备对象句柄,它由 CreateDevice 函数创建
                PLONG lWFIFOSts,        // 1:溢出  0:未溢出
                PLONG lWriteDataCount,  // 返回写数据缓冲区的数据个数
                ULONG ulChannel);       // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：获取写数据缓冲区的状态。

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。

相关函数：CreateDevice

♦读取数据

函数原型：

Visual C++:

```
BOOL ReadDeviceProCNT(                          // 读取数据
                      HANDLE hDevice,           // 设备句柄
                      ULONG CNTBuffer[],  // 将用于接受原始 CNT 数据的用户缓冲区
                      LONG nReadSizeWords,      // 读入的数据长度(字)
                      PLONG nRetSizeWords,          // 返回传输的实际长度(字)
                      ULONG ulChannel);         // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：读取数据。

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。

相关函数：CreateDevice

♦写入数据

函数原型：

Visual C++:

```
BOOL WriteDeviceProCNT(                         // 写入数据
                       HANDLE hDevice,          // 设备句柄
                       ULONG CNTBuffer[],  // 将用于接受原始 CNT 数据的用户缓冲区
                       LONG nWriteSizeWords,    // 写入的数据长度
                       PLONG nRetSizeWords,         // 返回传输的实际长度
                       ULONG ulChannel);        // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：写入数据。

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。

相关函数：CreateDevice

♦计数器禁止

函数原型：

Visual C++:

```
BOOL StopDeviceCNT(                    // 计数器禁止
                   HANDLE hDevice,     // 设备对象句柄,它由 CreateDevice 函数创建
                   ULONG ulChannel);   // 通道号[0~7]
```

LabVIEW：

请参考相关演示程序。

功能：计数器禁止。

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。

相关函数：CreateDevice

4.4 计数器硬件参数操作函数原型说明

♦将当前的计数器采样参数保存至系统中

函数原型：

Visual C++:

```
BOOL SaveParaCNT(                  // 将当前的 CNT 采样参数保存至系统中
                 HANDLE hDevice,
                 PPXI2391_PARA_CNT pCNTPara,
                 ULONG ulChannel);
```

LabVIEW：

请参考相关演示程序。

功能：将当前的计数器采样参数保存至系统中。

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。

相关函数：CreateDevice

♦将计数器采样参数从系统中读出

函数原型：

Visual C++:

```
BOOL LoadParaCNT(                  // 将 CNT 采样参数从系统中读出
                 HANDLE hDevice,
                 PPXI2391_PARA_CNT pCNTPara,
                 ULONG ulChannel);
```

LabVIEW：

请参考相关演示程序。

功能：将计数器采样参数从系统中读出。

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。

相关函数：CreateDevice

♦将计数器采样参数恢复至出厂默认值

Visual C++:

```
BOOL ResetParaCNT(                    // 将 CNT 采样参数恢复至出厂默认值
                  HANDLE hDevice,
                  PPXI2391_PARA_CNT pCNTPara,
                  ULONG ulChannel);
```

LabVIEW：

请参考相关演示程序。

功能：将计数器采样参数恢复至出厂默认值。

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。

相关函数：CreateDevice

4.5 数字 I/0 输入输出函数原型说明

♦ **设置开关量输入输出方向**

函数原型：

Visual C++:

```
BOOL SetDeviceDIODir(                 // 设置开关量输入输出方向
                  HANDLE hDevice,     // 设备对象句柄,它由 CreateDevice 函数创建
                  BYTE bDIODir [8]) ;  // 开关输入输出方向(0：输入；1：输出)
```

LabVIEW:

请参考相关演示程序。

功能：设置开关量输入输出方向。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

bDIODir 八路开关量输入输出状态选择的参数结构，共有 8 个成员变量，分别对应于 DIO0～DIO7 路开关量输入输出状态位。如果 bDIODir [0]为“0”则使 0 通道处于输入状态，若为“1”则 0 通道处于输出状态。其他同理。

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

相关函数： CreateDevice SetDeviceDIODir ReleaseDevice

♦ **取得数字量状态**

函数原型：

Visual C++:

```
BOOL GetDeviceDI(                    // 取得数字量状态
        HANDLE hDevice,     // 设备对象句柄,它由 CreateDevice 函数创建
        BYTE bDISts [8]) ; // 开关输入状态(注意: 必须定义为 8 个字节元素的数组)
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PXI 设备上的 DI0～DI7 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。

bDISts 八路开关量输入状态的参数结构，共有 8 个成员变量，分别对应于 DI0～DI7 路开关量输入状态位。如果 bDISts [0]为“1”则使 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

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

相关函数： CreateDevice GetDeviceDI ReleaseDevice

♦ **输出数字量状态**

函数原型：

Visual C++:

```
BOOL SetDeviceDO(                    // 输出数字量状态
        HANDLE hDevice,     // 设备对象句柄,它由 CreateDevice 函数创建
        BYTE bDOSts[8]) ; // 开关输出状态(注意: 必须定义为 8 个字节元素的数组)
```

LabView：

请参考相关演示程序。

功能：负责将 PXI 设备上的 DO0～DO7 输出开关量置成相应的状态。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

bDOSts 八路开关量输出状态的参数结构，共有 8 个成员变量，分别对应于 DO0～DO7 路开关量输出状态位。比如置 bDOSts[0]为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO7 共 8 个成员变量赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

以上函数调用一般顺序详见 DIO 操作流程

5 上层用户函数接口应用实例

如果您想快速的了解驱动程序的使用方法和调用流程，以最短的时间建立自己的应用程序，那么我们强烈建议您参考相应的简易程序。此种程序属于工程级代码，可以直接打开不用作任何配置和代码修改即可编译通过，运行编译链接后的可执行程序，即可看到预期效果。

如果您想了解硬件的整体性能、精度、采样连续性等指标以及波形显示、数据存盘与分析、历史数据回放等功能，那么请参考高级演示程序。特别是许多不愿意编写任何程序代码的用户，您可以使用高级程序进行采集、显示、存盘等功能来满足您的要求。甚至可以用我们提供的专用转换程序将高级程序采集的存盘文件转换成相应格式，即可在 Excel、MatLab 第三方软件中分析数据（此类用户请最好选用通过 Visual C++制作的高级演示系统）。

5.1 简易程序演示说明

5.1.1 怎样使用 GetDeviceDI 函数进行开关量输入操作

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，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(主要参考 PXI2391.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丿**[PXI2391 多功能计数器卡]**丨**[Microsoft Visual C++]**丿[简易代码演示]丿[DI 简易源程序]

其默认存放路径为：系统盘\ART\PXI2391\SAMPLES\VC\SIMPLE\DI

5.1.2 怎样使用 SetDeviceDO 函数进行开关量输出操作

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，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(主要参考 PXI2391.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丿**[PXI2391 多功能计数器卡]**丨**[Microsoft Visual C++]**丿[简易代码演示]丿[DO 简易源程序]

其默认存放路径为：系统盘\ART\PXI2391\SAMPLES\VC\SIMPLE\DO

5.2 高级程序演示说明

高级程序演示了本设备的所有功能，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(主要参考 PXI2391.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丿**[PXI2391 多功能计数器卡]**丨**[Microsoft Visual C++]**丿[高级代码演示]

其默认存放路径为：系统盘\ART\PXI2391\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

6 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。

6.1 公用接口函数总列表

表 6-1-1：公用接口函数总列表（每个函数省略了前缀“PXI2391_”）

函数名	函数功能	备注
① PXI 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	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 I/O 端口直接操作及读写函数		
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	

6.2 PXI 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 PXI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

♦ **获取设备固件及程序版本**

函数原型：

Visual C++:

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

LabVIEW:

请参考相关演示程序。

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, WriteRegisterByte, GetDeviceBar, GetDevVersion, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

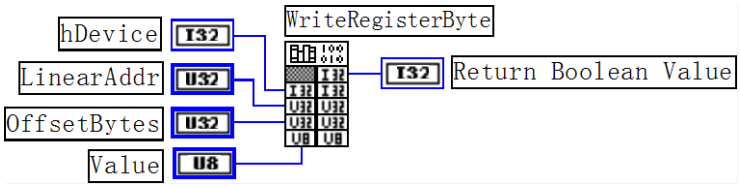
♦ **以单字节（即 8 位）方式写 PXI 内存映射寄存器的某个单元**

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 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

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, WriteRegisterByte, GetDeviceBar, GetDevVersion, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);  // 往指定映射寄存器单元写入 8 位的十六进制数据
20

ReleaseDevice( hDevice );  // 释放设备对象
    :
```

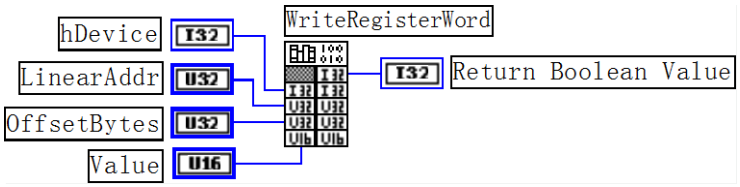
♦ **以双字节（即 16 位）方式写 PXI 内存映射寄存器的某个单元**

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 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

Value 输出 16 位整型值。

返回值：无。

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

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
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
    :
```

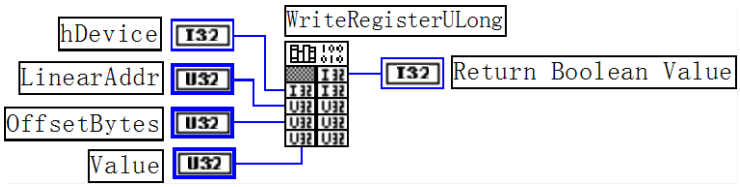
♦ **以四字节（即 32 位）方式写 PXI 内存映射寄存器的某个单元**

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 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

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	WriteRegisterByte	GetDeviceBar
GetDevVersion	WriteRegisterWord	WriteRegisterULong
ReadRegisterByte	ReadRegisterWord	ReadRegisterULong
ReleaseDevice		

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
OffsetBytes=100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);  // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

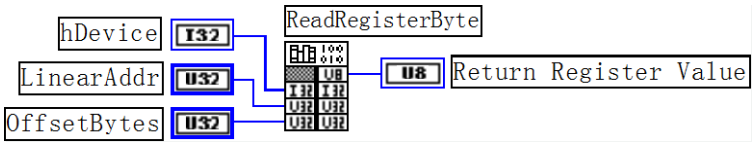
♦ **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

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 PXI 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 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

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	WriteRegisterByte	GetDeviceBar
GetDevVersion	WriteRegisterWord	WriteRegisterULong
ReadRegisterByte	ReadRegisterWord	ReadRegisterULong
ReleaseDevice		

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr,    PhysAddr, OffsetBytes;
BYTE    Value;
hDevice = CreateDevice(0);   // 创建设备对象
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
    :
```

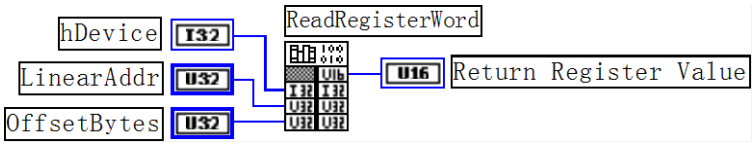
♦ **以双字节（即 16 位）方式读 PXI 内存映射寄存器的某个单元**

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 PXI 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 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

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, WriteRegisterByte, GetDeviceBar, GetDevVersion, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

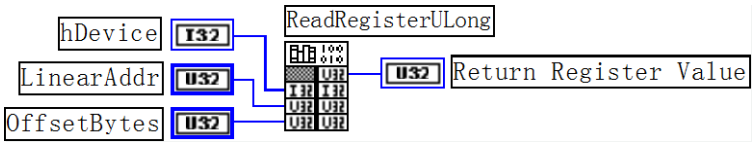
♦ **以四字节（即 32 位）方式读 PXI 内存映射寄存器的某个单元**

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 PXI 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 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数。

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, WriteRegisterByte, GetDeviceBar, GetDevVersion, WriteRegisterWord, WriteRegisterULong, ReadRegisterByte, ReadRegisterWord, ReadRegisterULong, ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
__int64 pbLinearAddr,   PhysAddr, OffsetBytes;
ULONG   Value;
hDevice = CreateDevice(0);   // 创建设备对象
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
    :
```

6.3 IO 端口读写函数原型说明

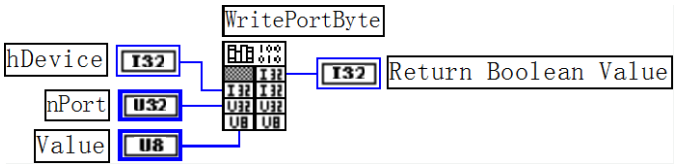
♦ **以单字节(8Bit)方式写 I/O 端口**

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　ReadPortULong　ReleaseDevice

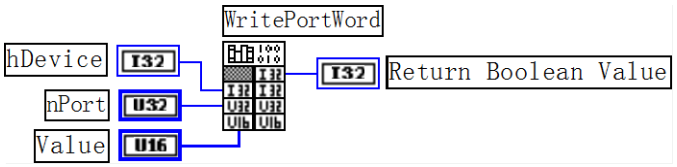
♦ **以双字(16Bit)方式写 I/O 端口**

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　ReadPortULong　ReleaseDevice

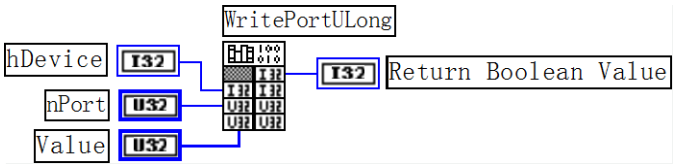
♦ **以四字节(32Bit)方式写 I/O 端口**

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，用户可用 GetLastErrorEx 捕获当前错误码。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord　ReadPortULong　ReleaseDevice

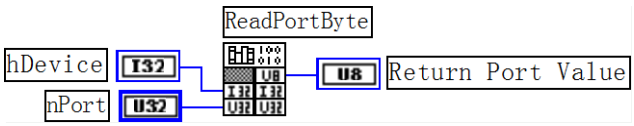
♦ **以单字节(8Bit)方式读 I/O 端口**

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　ReadPortULong　ReleaseDevice

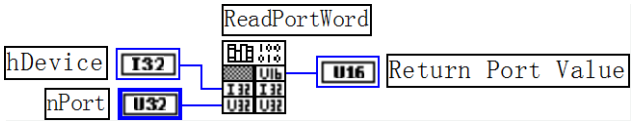
♦ **以双字节(16Bit)方式读 I/O 端口**

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　ReadPortULong　ReleaseDevice

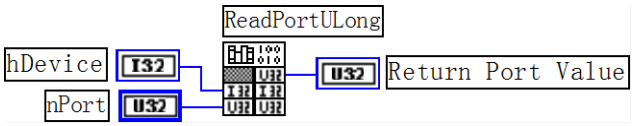
♦ **以四字节(32Bit)方式读 I/O 端口**

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　ReadPortULong　ReleaseDevice

6.4 线程操作函数原型说明

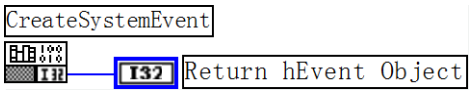
♦ **创建内核系统事件**

函数原型：

Visual C++:

HANDLE CreateSystemEvent(void)

LabVIEW:



功能：创建系统内核事件对象。

参数：无任何参数。

返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

♦ **释放内核系统事件**

函数原型：

Visual C++:

BOOL ReleaseSystemEvent(HANDLE hEvent)

LabVIEW:

请参见相关演示程序。

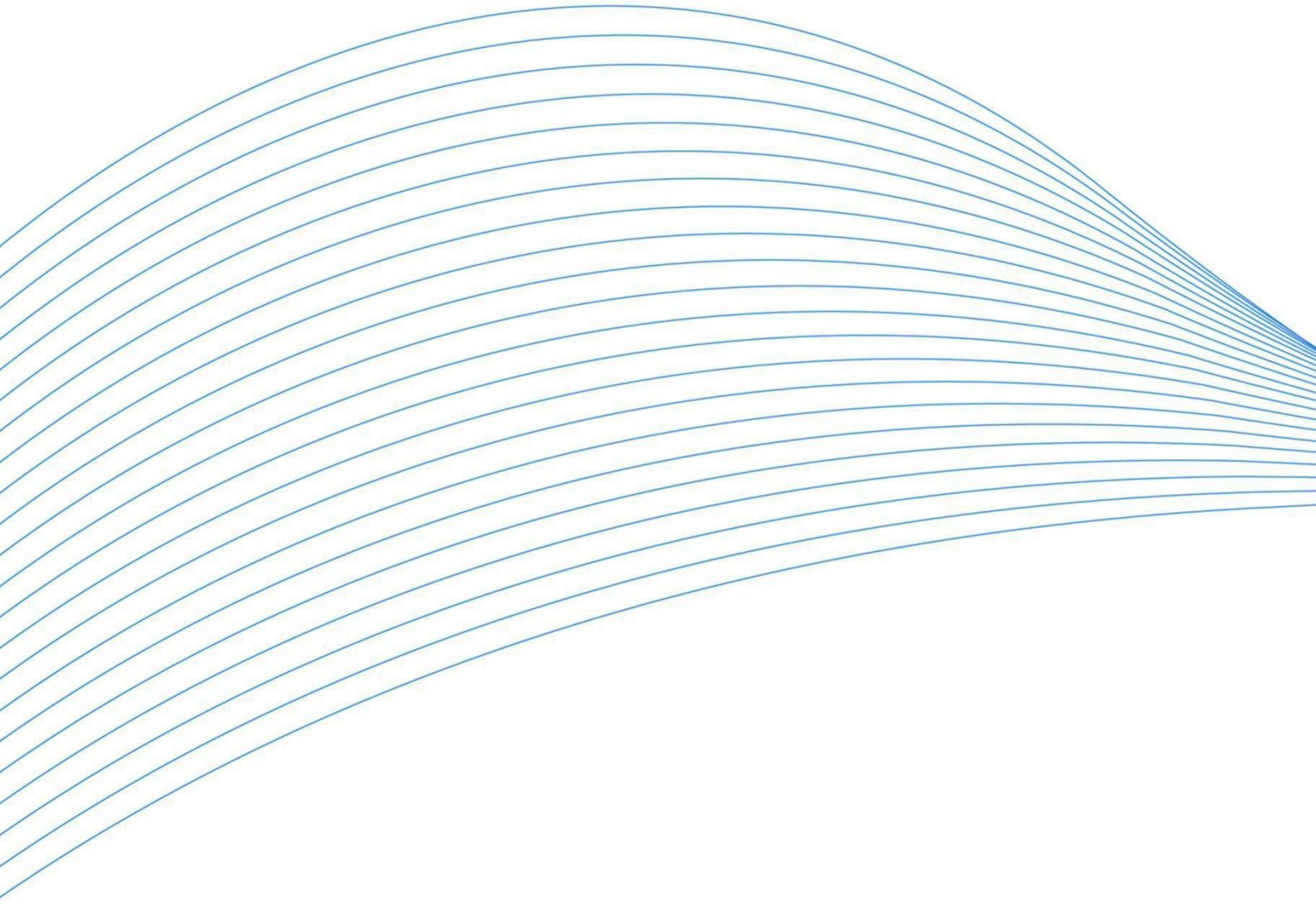
功能：释放系统内核事件对象。

参数：hEvent　被释放的内核事件对象。它应由 ReleaseSystemEvent 成功创建的对象。

返回值：若成功，则返回 TRUE。

7 修改历史

修改时间	版本号	修改内容
2015.8.21	V6.00.00	第一版



北京阿尔泰科技发展有限公司
服务热线：400-860-3335
邮编：100086
传真：010-62901157