PCI2318

WIN2000/XP/WIN7 驱动程序使用说明书

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

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

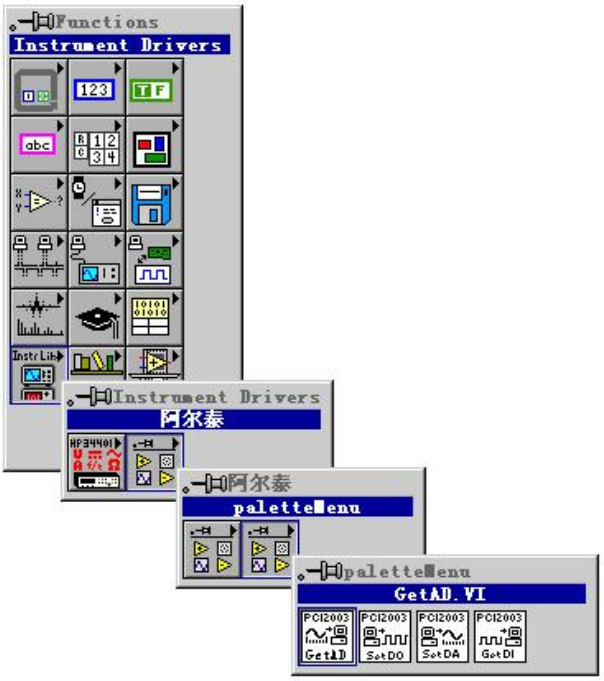
第二章 PCI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节、只关心 AD 的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的 AD 数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群需要直接跟设备端口打交道的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，他们最希望他们在软件操作上所要面对的全是他们最关心的问题，比如在采集时，通过 PARA_AD 初始化 AD 设备，告诉设备我要使用多少个通道，接地方式是什么，然后便可以用 ReadDevBulkAD 函数指定每次采集的点数，即可实现连续不间断批量读取 AD 数据(也可单点)。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的 Bit 位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉 PCI 总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解 PCI 的资源配置空间、PNP 即插即用管理，而只须用 GetDeviceBar 函数便可以取得指定设备寄存器组 BAR 地址。这个时候您便可以用这个寄存器组 BAR 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，先得明白自己是上层用户还是底层用户，因为在《接口函数列表》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出的关于 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 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。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。**此功能由于 LabView 自身版本兼容的问题，我们不便提供内嵌式驱动，如果用户确有此要求，请与我们的代理商或公司总部联系，但我们不保证完全免费。**

关于 LabView 的外挂式驱动和内嵌式驱动更详细的描述，请参考相关演示程序。



LabView 内嵌式驱动接口的获取方法

第一节、接口函数列表

（每个函数省略了前缀“PCI2318_”）

函数名	函数功能	备注
PCI 通用函数		
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
ListDeviceDlg	列表系统当中的所有的该 PCI 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
AD 操作函数		
ReadDevOneAD	单点采集 AD 数据	上层用户
ReadDevBulkAD	批量采集 AD 数据	上层用户
DA 操作函数		
WriteDeviceProDA	单点输出 DA 数据	上层用户
AD 硬件参数系统保存、读取函数		
SaveParaAD	将当前的 AD 采样参数保存至系统中	上层用户
LoadParaAD	将 AD 采样参数从系统中读出	上层用户
ResetParaAD	将 AD 采样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\PCI2318\INCLUDE\PCI2318.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 PCI2318.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。

其次，您还应该在 Visual C++编译环境软件包的 Project Setting 对话框的 Link 属性页中的 Object/Library Module 输入行中加入指令 C:\Art\PCI2318\PCI2318.LIB

或者：单击 Visual C++编译环境软件包的 Project 菜单中的 Add To Project 的菜单项，在此项中再单击 Files…，在随后弹出的对话框中选择 PCI2318.Lib，再单击“确定”，即可完成。

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 PCI2318.LIB 的路径，当然也可以把此文件拷到您的源程序目录中。

另外，在 Visual C++演示工程的目录下，也有相应的 PCI2318.h 和 PCI2318.Lib 文件。

为了驱动程序和相关接口尽量精炼快速，所以没有加任何调试代码，因此用户在使用 VC 接口的时候应使用发行版本进行源代码编译（Win32 Release），而不应该使用调试版本（Win32 Debug）。具体方法是在源代码编译前，执行 Build 总菜单中的 Set Active Configuration 子菜单命令，便可实现其发行版的设置，然后再编译，即可生成发行版的应用程序。

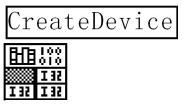
Visual Basic:

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 PCI2318.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 PCI2318.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续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坚线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端。

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

函数原型：

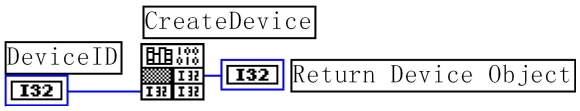
Visual C++ ：

HANDLE CreateDevice (int DeviceLgcID=0)

Visual Basic：

Declare Function CreateDevice Lib “PCI2318_32”(Optional ByVal DeviceLgcID As Integer = 0) As long

LabVIEW：



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

参数：

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PCI 设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PCI2318 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 PCI2318 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 PCI 设备时，DeviceLgcID 应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由 BIOS 和操作系统加载设备时，依据主板总线编号等信息进行这个设备 ID 号分配，说得简单点，就是加载设备的顺序编号，编号的递增顺序为 0、1、2、3……。

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： GetDeviceCount ListDeviceDlg ReleaseDevice

♦ 取得本计算机系统中 PCI2318 设备的总数量

函数原型：

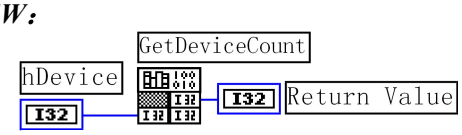
Visual C++ :

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib “PCI2318_32” (ByVal hDevice As Long) As Integer

LabVIEW：



功能：取得 PCI2318 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 PCI2318 的数量。

相关函数： CreateDevice ListDeviceDlg ReleaseDevice

♦ 用对话框控件列表计算机系统中所有 PCI2318 设备各种配置信息

函数原型：

Visual C++ :

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib “PCI2318_32” (ByVal hDevice As Long) As Boolean

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCI2318 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 PCI2318 设备的配置情况。

相关函数： CreateDevice GetDeviceCount ReleaseDevice

◆ **释放设备对象所占的系统资源及设备对象**

函数原型：

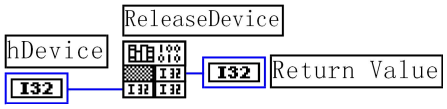
Visual C++ :

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib “PCI2318_32” (ByVal hDevice As Long) As Boolean

LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。

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

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

相关函数： CreateDevice GetDeviceCount ListDeviceDlg

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、程序方式 AD 读取函数

1、单点读取 PCI 设备上的 AD 数据

函数原型:

Visual C++ :

```
WORD ReadDevOneAD ( HANDLE hDevice,
                    int nADChannel
                    LONG GroundingMode)
```

Visual Basic:

```
Declare Function ReadDevOneAD Lib "PCI2318_32" (ByVal hDevice As Long,
                                                ByVal nADChannel As Integer,
                                                ByVal GroundingMode As Long) As Integer
```

LabView：

请参考相关演示程序。

功能：用户每调用一次该函数，即可从 PCI 设备上取得一个点的 AD 原始数据。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

nADChannel 接受 AD 数据的用户缓冲区。关于如何将这些 AD 数据转换成相应的电压值，请参考请第四章《数据转换》。

GroundingMode 通道接地方式。

返回值：返回由 ADChannel 参数指定通道上的一个 AD 数据。关于如何将这些 AD 数据转换成相应的电压值，请参考请第四章《数据转换》。

相关函数： CreateDevice ReadDevBulkAD ReleaseDevice

2、批量读取 PCI 设备上的 AD 数据

函数原型:

Visual C++ & :

```
BOOL ReadDevBulkAD ( HANDLE hDevice,
                     WORD pADBuffer[],
                     ULONG nReadSizeWords,
                     PPCI2318_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ReadDevBulkAD Lib "PCI2318_32" (ByVal hDevice As Long,
                                                 ByRef pADBuffer As Integer,
                                                 ByVal nReadSizeWords As Long,
                                                 ByRef pADPara As PCI2318_PARA_AD) As Boolean
```

LabView：

请参考相关演示程序。

功能：用户每调用一次该函数，即可从 PCI 设备上取得一批量的 AD 原始数据。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pADBuffer 接受批量 AD 数据的缓冲区。关于如何将这些 AD 数据转换成相应的电压值，请参考请第四章《数据转换》。

nReadSizeWords 每一次批量 AD 数据的长度（点数）。

pADPara AD 硬件参数，它主要指定采样的始末通道号。

返回值：如果成功，即通过 pADBuffer 返回指定长度的 AD 数据。

相关函数： CreateDevice ReadDevOneAD ReleaseDevice

以上函数调用一般顺序

① CreateDevice

② ReadDevOneAD(或 ReadDevBulkAD)

③ ReleaseDevice

用户可以反复执行第②步，以不断读取新的 AD 转换结果。

第四节、DA 操作函数原型说明

◆ **输出 DA 数据**

函数原型:

Visual C++ :

```
BOOL WriteDeviceProDA ( HANDLE hDevice,
                        SHORT nDAData,
                        int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceProDA Lib  “PCI2318_32”  ( ByVal hDevice As Long, _
                                                       ByVal nDAData As Long, _
                                                       ByVal nDAChannel As Long) As Boolean
```

LabView：

请参考相关演示程序。

功能：向指定通道上输出一个点的 DA 数据。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

nDAData 准备输出的 12 位 DA 原始数据，注意它的换算关系请参考第四章《数据转换》。

nDAChannel DA 通道，取值范围为 0-3。

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

相关函数： CreateDevice ReleaseDevice

以上函数调用一般顺序

① CreateDevice
② WriteDeviceProDA
③ ReleaseDevice

用户可以反复执行第②步，以不断读取新的计数结果。

第五节、AD 硬件参数保存与读取函数原型说明

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++ :

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPCI2318_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI2318_32" (ByVal hDevice As Long, _
                                              ByRef pADPara As PCI2318_PARA_AD) As Boolean
```

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 PCI2318_PARA_AD 的详细介绍请参考 PCI2318.h 或 PCI2318.Bas 或 PCI2318.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

♦ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++ :

```
BOOL LoadParaAD(HANDLE hDevice,
                PPCI2318_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PCI2318_32" (ByVal hDevice As Long, _
                                              ByRef pADPara As PCI2318_PARA_AD) As Boolean
```

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 PPCI2318_PARA_AD 的结构指针类型，它负责返回 PCI 硬件参数值，关于结构指针类型 PPCI2318_PARA_AD 请参考 PCI2318.h 或 PCI2318.Bas 或 PCI2318.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：

Viusal C++ :

```
BOOL ResetParaAD (HANDLE hDevice,
                  PPCI2318_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "PCI2318_32" ( ByVal hDevice As Long, _
                                                ByRef pADPara As PCI2318_PARA_AD) As Boolean
```

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 PCI2318_PARA_AD 的详细介绍请参考 PCI2318.h 或 PCI2318.Bas 或 PCI2318.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《其他函数原型说明》。

第三章 硬件参数结构

第一节、AD 硬件参数结构（PCI2318_PARA_AD）

Visual C++ ：

```
Typedef struct _PCI2318_PARA_AD        // 板卡各参数值
{
    LONG FirstChannel;          // 首通道
    LONG LastChannel;           // 末通道
    LONG GroundingMode[32]; // 接地方式
} PCI2318_PARA_AD,* PPCI2318_PARA_AD;
```

Visual Basic：

```
Private Type PCI2318_PARA_AD
    FirstChannel As Long        ' 首通道号
    LastChannel As Long
    GroundingMode(32) As Long   ' 接地方式
End Type
```

LabView:

首先请您关注一下这个结构与前面 ISA 总线部分中的两个结构 PARA、PARAEX 比较，该结构实在太简短了。其原因就是 PCI 设备是系统全自动管理的设备，什么端口地址，中断号，DMA 等将与 PCI 设备的用户永远告别，一句话 PCI 设备是一种更易于管理和使用的设备。

硬件参数说明: 此结构主要用于设定设备硬件参数值，用这个参数结构对设备进行批量读数。

FirstChannel:AD 采样首通道值，取值范围应根据设备的总通道数设定，本设备的通道取值为：0～31，要求首通小于或等于末通道。

LastChannel : AD 采样末通道值，取值范围应根据设备的总通道数设定，本设备的通道取值为：0～31，要求末通道大于或等于首通道。

注：当首通道和末通道相等时，即为单通道采集。否则，为多通道采集。如果为多通道采集，即采集通道总数由 LastChannel 减去 FirstChannel 再加 1，其结果便是通道数。为多通道采集时，采集的通道顺序为：FirstChannel, FirstChannel+1, FirstChannel+2……LastChannel,然后重复下去。比如设首通道为 0，末通道为 3，则采样顺序时：0、1、2、3、0、1、2、3、0、1、2、3……。

GroundingMode : 接地方式，其选项为:

常量名	常量值	功能定义
PCI2318_SE_MODE	0x0000	单端方式
PCI2318_DI_MODE	0x0001	双端方式

相关函数： ReadDevBulkAD

第四章 数据转换

第一节、如何将 AD 原始数据 LSB 转换电压值 Volt

在换算过程中弄清模板精度（即 Bit 位数）是很关键的，因为它决定了 LSB 数码的总宽度 CountLSB。比如 8 位的模板 CountLSB 为 256。而本设备的 AD 为 16 位，则为 65536。其他类型同理均按 2^n=LSB 总数（n 为 Bit 位数）换算即可。

设从设备上读入的某个 AD 原码数据为变量 Lsb，其对应的电压为变量 Volt（单位 mV）：

量程(毫伏)	计算机语言换算公式	Volt 取值范围 mV
±10000mV	Volt = Lsb * (20000 / 65536) – 10000	[-10000, +10000]
±5000mV	Volt = Lsb * (10000 / 65536) – 5000	[-5000，+5000]
±2500mV	Volt = Lsb * (5000 / 65536) – 2500	[-2500，+2500]
0～10000mV	Volt = Lsb * (10000 / 65536)	[0, +10000]
0～5000mV	Volt = Lsb * (5000 / 65536)	[0, +5000]

注意：以上所列 Lsb 必须是将从设备上读入的 AD 数(设为 ADBuffer[0])后得到的。

换算举例：（设量程为±10000mV，这里只转换第一个点）

Visual C++：

```
WORD Lsb;// 定义存放标准 LSB 原码的变量
float Volt;// 定义存放转换后的电压值的变量
Lsb=ADBuffer［0］; // 取得标准 LSB 原码
Volt=Lsb*（20000.0/65536）-10000.0;// 用 LSB 原码与单位电压值相乘求得实际电压值
```

Visual Basic:

```
Dim Lsb As Integer ’ 定义存放标准 LSB 原码的变量
Dim Volt As Simple ’ 定义存放转换后的电压值的变量
Lsb=ADBuffer（0）’取得标准 LSB 原码
Volt=Lsb*（20000.0/65536）-10000.0 ’ 用 LSB 原码与单位电压值相乘求得实际电压值
```

第二节、关于采集函数的 ADBuffer 缓冲区中的数据排放规则(多通道批量采集时)

当首末通道相等时，即为单通道采集，假如 FirstChannel=5, LastChannel=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(CH0 – CH2)

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(CH0 - CH3)

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，那以需要用户特别注意的是应处理好各通道数据排列和对齐问题，尤其任意通道数采集时。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。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民，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的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，但对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。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	…
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	…
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	…
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、如何将电压值转换成 DA 原码 LSB

如何在电压值与 DA 输出原始码值之间进行转换，请参考下表：

量程(伏)	计算机语言换算公式	Lsb 取值范围
0～5000mV	Lsb = Volt/ (5000 / 4096)	[0，4095]
0～10000mV	Lsb = Volt/ (10000 / 4096)	[0，4095]
±5000mV	Lsb = Volt / (10000 / 4096) + 2048	[0，4095]
±10000mV	Lsb = Volt/ (20000 / 4096) + 2048	[0，4095]

请注意这里求得的 LSB 数据就是用于 WriteDeviceProDA 中的 nDAData 参数的。

第五章　上层用户函数接口应用实例

第一节、如何实现 AD 采集

关于 AD 数据采集可以通过两种方式实现，一是单点 ReadDevOneAD，而是批量 ReadDevBulkAD。

详细演示请参考工程。

[程序]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[简易代码演示]丨[AD 单点采集演示源程序]

第二节、如何实现 DA 输出

[程序]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[简易代码演示]丨[DA 单点输出演示源程序]

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“PCI2318_”）

函数名	函数功能	备注
① PCI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口

WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 内核事件操作函数		
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	

第二节、PCI 内存映射寄存器操作函数原型说明

♦ **取得指定的指定设备寄存器组 BAR 地址**

函数原型：

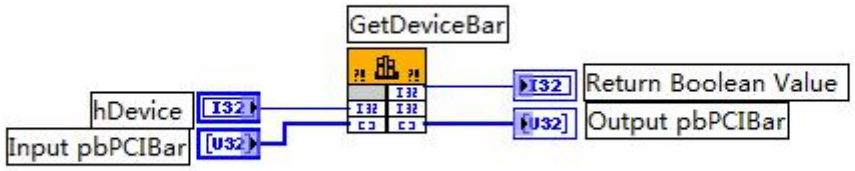
Visual C++：

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

Visual Basic：

```
Declare Function GetDeviceBar Lib "PCI2318_32" (ByVal hDevice As Long, _
                                                ByRef pulPCIBar As Long) As Boolean
```

LabVIEW:



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,它由 CreateDevice 函数创建。

pbPCIBar 指针参数，用于返回 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：如果执行成功，则返回 TRUE，否则会返回 FALSE。

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

♦ **以单字节（即 8 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型：

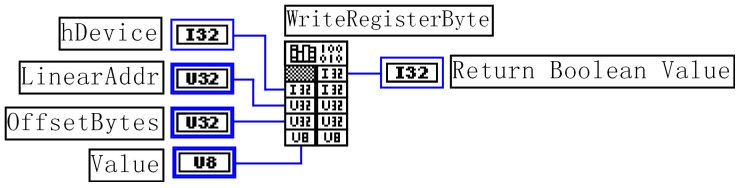
Visual C++ :

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

Visual Basic:

```
Declare Function WriteRegisterByte Lib   “PCI2318_32”  (ByVal hDevice As Long, _
                                                        ByVal LinearAddr As Long, _
                                                        ByVal OffsetBytes As Long, _
                                                        ByVal Value As Byte ) As Boolean
```

LabVIEW:



功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceBar(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)
GetDeviceBar( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

♦以双字节（即 16 位）方式写 PCI 内存映射寄存器的某个单元

函数原型:

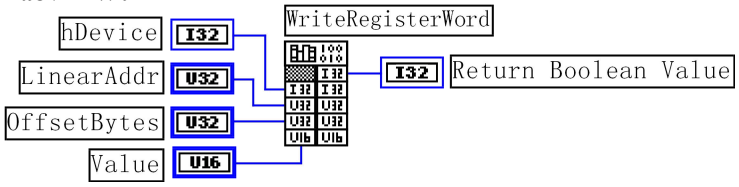
Visual C++ :

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

Visual Basic:

```
Declare Function WriteRegisterWord Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                       ByVal LinearAddr As Long, _
                                                       ByVal OffsetBytes As Long, _
                                                       ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 确定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceBar(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 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)
GetDeviceBar( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

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

函数原型:

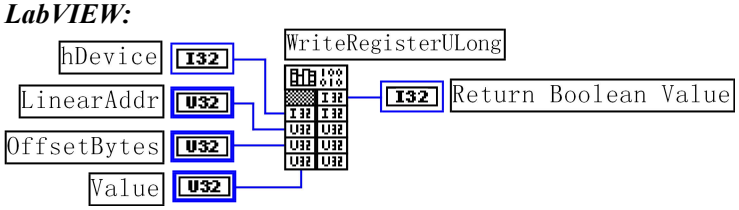
Visual C++ :

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

Visual Basic:

```
Declare Function WriteRegisterULong Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                        ByVal LinearAddr As Long, _
                                                        ByVal OffsetBytes As Long, _
                                                        ByVal Value As Long)
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte

ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceBar(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 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)
GetDeviceBar( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
    :
```

◆ **以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型:

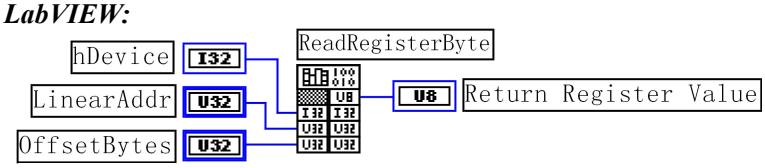
Visual C++ & :

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

Visual Basic:

```
Declare Function ReadRegisterByte Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                      ByVal LinearAddr As Long, _
                                                      ByVal OffsetBytes As Long) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 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
GetDeviceBar(hDevice, &LinearAddr, &PhysAddr, 0);  // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
```

```
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

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

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

函数原型:

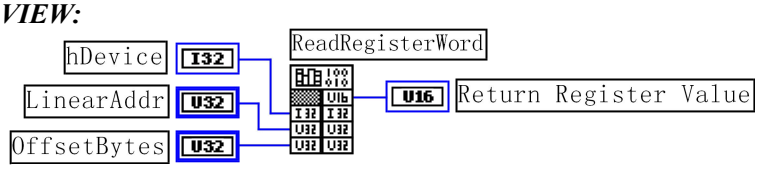
Visual C++ :

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

Visual Basic:

```
Declare Function ReadRegisterWord Lib  "PCI2318_32"  (  ByVal hDevice As Long, _
                                                        ByVal LinearAddr As Long, _
                                                        ByVal OffsetBytes As Long) As Integer
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 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
GetDeviceBar(hDevice, &LinearAddr, &PhysAddr, 0);  // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

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

```
ReleaseDevice(hDevice)
    :
```

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

函数原型:

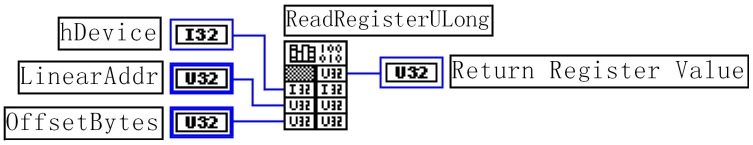
Visual C++ :

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

Visual Basic:

```
Declare Function ReadRegisterULong Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                       ByVal LinearAddr As Long, _
                                                       ByVal OffsetBytes As Long) As Long
```

LabVIEW:



功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

LinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceBar 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 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
GetDeviceBar(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceBar( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

第三节、I/O 端口读写函数原型说明

注意：若您想在 WIN2000/XP/WIN7 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**

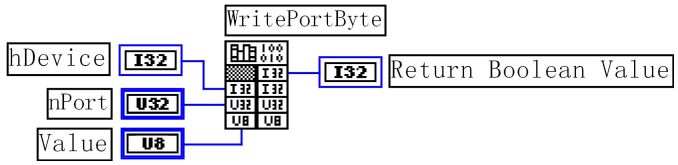
Visual C++ :

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

Visual Basic:

```
Declare Function WritePortByte Lib  "PCI2318_32"  ( ByVal hDevice As Long, _
                                                    ByVal nPort As Long, _
                                                    ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastError 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**

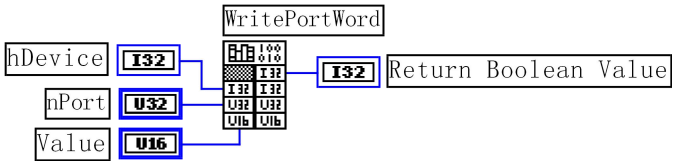
Visual C++ :

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

Visual Basic:

```
Declare Function WritePortWord Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                   ByVal nPort As Long, _
                                                   ByVal Value As Integer) As Boolean
```

LabVIEW：



功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastError 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**

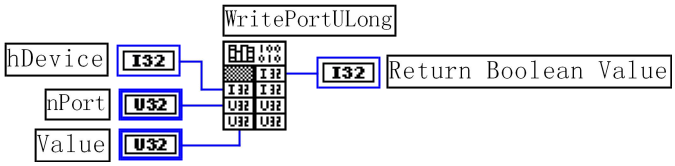
Visual C++ :

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

Visual Basic:

```
Declare Function WritePortULong Lib  "PCI2318_32"  (ByVal hDevice As Long, _
                                                    ByVal nPort As Long, _
                                                    ByVal Value As Long ) As Boolean
```

LabVIEW：

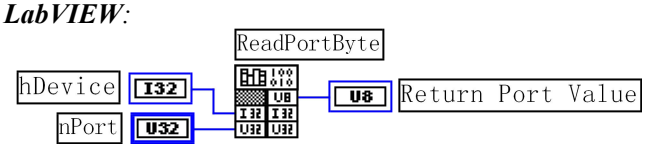


功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

Visual C++ :
BYTE ReadPortByte(HANDLE hDevice, UINT nPort)
Visual Basic:
Declare Function ReadPortByte Lib “PCI2318_32” (ByVal hDevice As Long, _
ByVal nPort As Long) As Byte

LabVIEW:

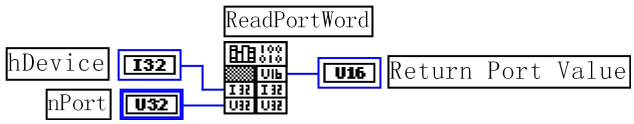


功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

Visual C++ :
WORD ReadPortWord(HANDLE hDevice, UINT nPort)
Visual Basic:
Declare Function ReadPortWord Lib “PCI2318_32” (ByVal hDevice As Long, _
ByVal nPort As Long) As Integer

LabVIEW:



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

◆ **以四字节(32Bit)方式读 I/O 端口**

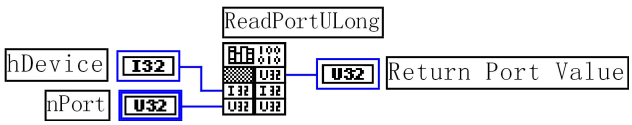
Visual C++ :

ULONG ReadPortULong(HANDLE hDevice, UINT nPort)

Visual Basic:

Declare Function ReadPortULong Lib “PCI2318_32” (ByVal hDevice As Long, _
ByVal nPort As Long) As Long

LabVIEW：



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

第四节、线程操作函数原型说明

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

◆ **创建内核系统事件**

Visual C++:

HANDLE CreateSystemEvent(void);

Visual Basic:

Declare Function CreateSystemEvent Lib “PCI2318_32” () As Long

LabVIEW：

功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

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

◆ **释放内核系统事件**

Visual C++ :

BOOL ReleaseSystemEvent(HANDLE hEvent);

Visual Basic:

Declare Function ReleaseSystemEvent Lib “PCI2318_32” (ByVal hEvent As Long) As Boolean

LabVIEW：

请参见相关演示程序。

功能：释放系统内核事件对象。

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

返回值：若成功，则返回 TRUE。