PCI2362
DOS/Win95/98/NT/2000 驱 动 程 序 使 用 说 明 书

目　　录

第一章　版权信息

第二章 本驱动程序软件的关键文件

（WinDir 指 Windows 的系统根目录, UserDir 为本驱动软件的用户安装根目录）

文件名	文件类型及功能	适用的操作系统	文件位置
PCI2362.VxD	动态虚拟设备驱动程序库	Window95/98	WinDir\System
PCI2362.Sys	Win32 标准设备驱动 WDM 模式的设备驱动程序库	Windows NT/2000	WinDir\System32\Drivers
PCI2362.Dll	底层驱动程序库的用户级函数接口封装所用的动态库。	所有操作系统	WinDir\System
PCI2362.Lib	基于 Microsoft Visual C++工程开发环境的驱动程序函数接口输入库。	所有操作系统	UserDir\Include 或 UserDir\Samples\VC…
PCI2362.Lib	基于 Borland C++ Builder 工程开发环境的驱动程序函数接口输入库。	所有操作系统	UserDir\Samples\C_Builder
PCI2362.Bas	基于 Microsoft Visual Basic 工程开发环境的驱动程序函数接口输入模块文件	所有操作系统	UserDir\Samples\VB
PCI2362.Pas	基于 Borland Delphi 工程开发环境的驱动程序函数接口输入单元文件。	所有操作系统	UserDir\Samples\Delphi
PCI2362.VI	基于 National Instrument LabView 工程开发环境的驱动程序函数接口输入部件文件。**(只是外挂驱动接口)**	所有操作系统	UserDir\Samples\LabView

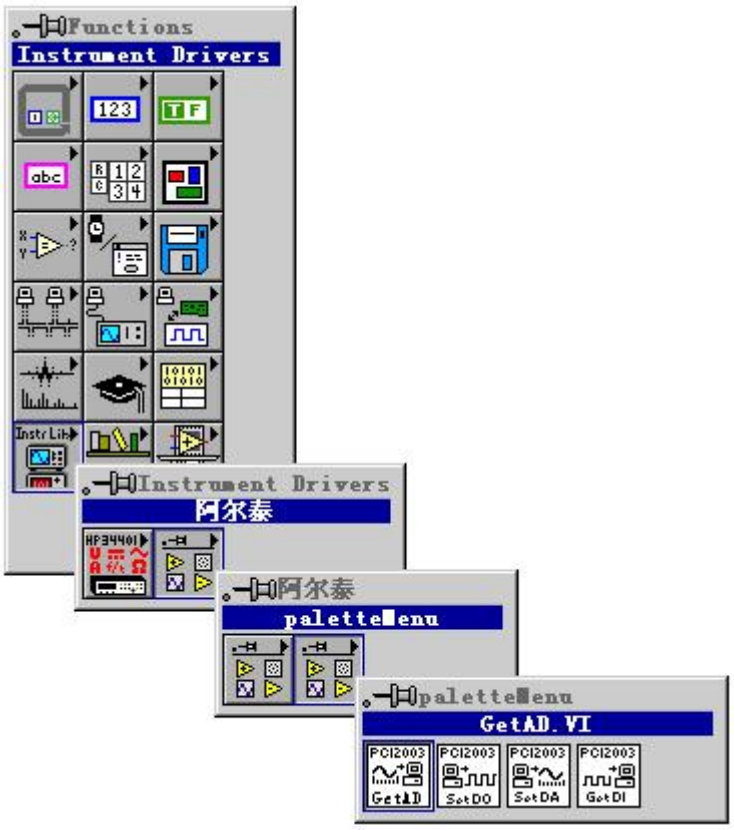
第三章 PCI 即插即用设备操作函数接口介绍

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，比如在正式采集数据之前，只须用户调用一个简易的 ReadDevBulkAD 函数，告诉设备我要使用多少个通道等信息后便采集到指定的点数。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 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 ReadPortULong 和 WritePortULong 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《**第一节 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、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。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叙述，请参考附录 A 的《LabView 驱动程序接口》章节。



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“PCI2362_”）

函数名	函数功能	备注
PCI 通用函数		
CreateDevice	创建 PCI 设备对象	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
ListDevice	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
计数器操作函数		
InitDevCounter	初始化计数器	上层用户
GetDevCounter	取得计数器值	上层用户
开关量简易操作函数		
#SetDevDIODir	设定 0-47 路开关量方向	上层用户
#ProcessDevDIO	操作 0-47 路开关量状态	上层用户
SetDeviceDO	48-71 路开关量输出	上层用户
GetDeviceDI	72-95 路开关量输入	上层用户
PCI 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定 PCI 设备寄存器操作基地址	底层用户
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

使用需知：

Visual C++ & C++Builder：

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\PCI2362\INCLUDE\PCI2362.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 PCI2362.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\PCI2362\PCI2362.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 PCI2362.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 PCI2362.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 PCI2362.h 和 PCI2362.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。

C++ Builder:

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头文件(PCI2362.H)写进您的源程序头部。如：#include “\Art\PCI2362\Include\PCI2362.h”

然后再将 PCI2362.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”（Add to Project）命令，在弹出的对话框中分别选择文件类型：Library file (*.lib)，即可选择 PCI2362.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下

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 PCI2362.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。

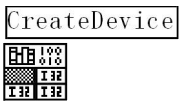
Delphi:

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单元模块文件（*.Pas）加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单,执行其中的"Project Manager"命令,在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PCI2362.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCI2362”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PCI2362;  // 注意：  在此加入驱动程序接口单元 PCI2362
```

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 PCI2362.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竖线为中心，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

1、创建设备对象函数

Visual C++ & C++Builder：

```
HANDLE CreateDevice (int DeviceID)
```

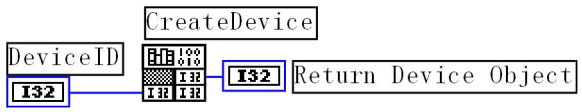
Visual Basic：

```
Declare Function CreateDevice Lib  “PCI2362”(Byval DeviceID as long)as long
```

Delphi：

```
Function CreateDevice(DeviceID:Integer):Integer;
    StdCall; External 'PCI2362' Name ' CreateDevice';
```

LabView：



功能：该函数负责创建 PCI 设备对象，并返回其设备对象句柄。

参数：

DeviceID 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 PCI 设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceID 标识值为名称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 PCI2362 AD 模板时，驱动程序则以“PCI2362”作为基本名称，再以 DeviceID 的初值组合成该设备的标识符“PCI2362-0”来确认和管理这第一个设备，若用户接着再添加第二个 PCI2362 AD 模板时，则系统将以“PCI2362-1”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，DeviceID 应置 0，第二个应置 1，也以此类推。

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： ReleaseDevice

Visual C++ & C++Builder 程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
hDevice=CreateDevice ( 0 ); // 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
hDevice = CreateDevice ( 0 )   ' 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

Else
  Exit Sub       ' 退出该过程
End If
      :
```

2、取得本计算机系统中 PCI2362 设备的总数量

函数原型：

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

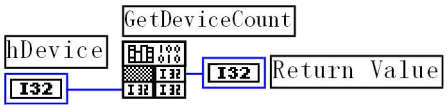
Visual Basic:

Declare Function GetDeviceCount Lib "PCI2362" (ByVal hDevice As Long) As Long

Delphi：

Function PCI2362_GetDeviceCount (hDevice : Integer):Integer;
StdCall; External 'PCI2362' Name 'GetDeviceCount';

LabView：



函数原型：

功能：取得 PCI2362 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 PCI2362 的数量。

相关函数： CreateDevice
ReleaseDevice

3、用对话框控件列表计算机系统中所有 PCI2362 设备各种配置信息

函数原型：

Visual C++ & C++Builder:

BOOL ListDevice (HANDLE hDevice)

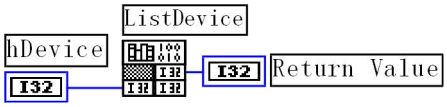
Visual Basic:

Declare Function ListDevice Lib "PCI2362" (ByVal hDevice As Long) As Boolean

Delphi：

Function ListDevice (hDevice : Integer):Boolean;
StdCall; External 'PCI2362' Name ' ListDevice ';

LabView：



功能：列表系统中 PCI2362 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，则返回列表设备。

相关函数： CreateDevice
ReleaseDevice

4、释放设备对象所占的系统资源及设备对象

函数原型：

Visual C++ & C++Builder:

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

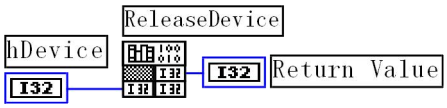
Visual Basic:

```
Declare Function ReleaseDevice Lib "PCI2362" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDevice(hDevice : Integer):Boolean;
StdCall; External 'PCI2362' Name ' ReleaseDevice';
```

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

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、计数器操作函数

◆计数器初始化

函数原型:

Visual C++ & C++Builder:

```
BOOL InitDevCounter ( HANDLE hDevice,
                      PPCI2362_PARA_COUNTER_CTRL pCtrlPara，
                      int InitCntrVal，
                      int nCntrChannel)
```

Visual Basic:

```
 Declare Function InitDevCounter Lib "PCI2362" (
                                  ByVal hDevice As Long, _
                                  ByRef pCtrlPara As PCI2362_PARA_COUNTER_CTRL,_
                                  ByVal InitCntrVal As Long,_
                                  ByVal nCntrChannel As Long ) As Integer
```

Delphi:

```
Function InitDevCounter (hDevice : Integer;
                         pCtrlPara : PPCI2362_PARA_COUNTER_CTRL;
                         InitCntrVal: LongInt;
                         nCntrChannel: LongInt):Boolean;
        StdCall; External 'PCI2362' Name ' InitDevCounter ';
```

LabView：

功能：初始化指定通道，包括计数方式、工作模式、计数初值等。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pCtrlPara 为 8253（或 8254）计数器控制字。它决定计数方式、工作模式等。

nCntrChannel 为 8253 计数器通道号，取值范围为 0-2

返回值：如果初始化计数器成功，则返回 FALSE，否则返回 TRUE。

相关函数： CreateDevice GetDevCounter ReleaseDevice

◆读取指定计数器的计数值

函数原型:

Visual C++ & C++Builder:

```
BOOL GetDevCounter ( HANDLE hDevice,
                     PLONG   pCntrValue,
                     int nCntrChannel)
```

Visual Basic:

```
Declare Function GetDevCounter Lib "PCI2362" (
                                ByVal hDevice As Long, _
                                ByRef pCntrValue As Long,_
                                nCntrChannel As Long) As Boolean
```

Delphi:

```
Function GetDevCounter (hDevice : Integer;
                        pCntrValue: PDwordArray;
                        nCntrChannel: LongInt ):Boolean;
        StdCall; External 'PCI2362' Name ' GetDevCounter ';
```

LabView：

功能：取得当前计数器的计数值

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pCntrValue 返回计数器值（16 位）

nCntrChannel 计数器通道，取值范围为 0-2

返回值：如果成功，返回 TRUE，且 pCntrValue 的值为有效，否则返回 FALSE，且 pCntrValue 值为无效。

相关函数： CreateDevice InitDevCounter ReleaseDevice

以上函数调用一般顺序

① CreateDevice
② InitDevCounter
③ GetDevCounter
④ ReleaseDevice

用户可以反复执行第③步，以不断读取新的计数结果。

第六节、简易的数字 IO 输入输出开关量操作函数

◆设置 0-47 路开关量的输入输出方向

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevDIODir (HANDLE hDevice, PPCI2362_PARA_DIR pDIODirPara)
```

Visual Basic:

```
Declare Function SetDevDIODir Lib "PCI2362" (ByVal hDevice As Long, _
                                             ByRef pDIODirPara As PCI2362_PARA_DIR) As Boolean
```

Delphi:

```
Function SetDevDIODir (hDevice : Integer; pDIODirPara:PPCI2362_PARA_DIR):Boolean;
        StdCall; External 'PCI2362.dll';
```

LabView(包括相关演示)：

功能：此函数负责设置 DIO0-DIO47 路开关的输入输出方向。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pDIODirPara 开关量输输入出方向的参数结构，共有 6 个成员变量，分别对应于 6 组（每组为 8 路开关量）的输入输出方向。比如置为“0”则使该组为输入向，若为“1”则使该组为输出向。请注意，在实际执行这个函数之前，必须对这个参数结构的每个成员变量赋初值，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 **#SetDevDIODir**

#ProcessDevDIO **#SetDeviceDO**
GetDeviceDI ReleaseDevice

◆操作 0-47 路开关量

函数原型：

Visual C++ & C++Builder:

```
BOOL ProcessDevDIO (HANDLE hDevice,    BYTE bDIOSts[48])
```

Visual Basic:

```
Declare Function ProcessDevDIO Lib "PCI2362" (ByVal hDevice As Long, _
                                              ByRef bDIOSts(48) As Byte) As Boolean
```

Delphi:

```
Function ProcessDevDIO (hDevice : Integer; bDIOSts[48]: Byte):Boolean;
        StdCall; External 'PCI2362.dll';
```

LabView(包括相关演示)：

功能：此函数根据 SetDeviceDIODir 设定的方向输出或读入 0-47 路开关量状态。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

bDIOSts[48] 开关量输入输出的状态值，共有 48 个元素，分别对应于 0-47 路开关量的输出或读入状态。若相应通道被 SetDeviceDIODir 置成输出状态，则对应数组元素应先赋成相应状态（0 或 1），然后再执行本函数。若相应通道被 SetDeviceDIODir 置成输入状态，则在执行本函数后，其相应数组元素将返回其相应通道的状态。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice **#SetDevDIODir**
#ProcessDevDIO **#SetDeviceDO**
GetDeviceDI ReleaseDevice

◆开关量 48-71 路输出

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceDO (HANDLE hDevice, BYTE bDOSts[24])
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI2362" (ByVal hDevice As Long, _
                                            ByRef bDOSts(24) As Byte) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer; bDOSts[24]:Byte):Boolean;
        StdCall; External 'PCI2362' Name ' SetDeviceDO';
```

LabView(包括相关演示)：

功能：此函数负责将 DO48-DO71 路开关量输出置成相应的状态

参数：

hDevice 设备对象句柄,它应由 CreateDevice 决定。

bDOSts[24] DO48-DO71 路开关量输出状态的数组，共有 24 个成员变量，分别对应于 DO48-DO71 路开关量输出状态位。比如置 bDOSts[0]为“1”则使 48 通道处于“开”状态，若为“0”则置 48 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个数组的共 24 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 **#SetDevDIODir**
#ProcessDevDIO **#SetDeviceDO**
GetDeviceDI ReleaseDevice

◆开关量 48-71 路输出

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceDI (HANDLE hDevice, BYTE bDISts[24])
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI2362" (ByVal hDevice As Long, _
                                            ByRef bDISts(24) As Byte) As Boolean
```

Delphi:

```
Function GetDeviceDI (hDevice : Integer; bDISts[24]:Byte):Boolean;
        StdCall; External 'PCI2362.dll';
```

LabView(包括相关演示)：

功能：此函数负责将取得 DO72-DO95 路开关量输入状态

参数：

hDevice 设备对象句柄,它应由 CreateDevice 决定。

bDISts[24] 返回 I72-DO95 路开关量输入状态的数组，共有 24 个成员变量，分别对应于 DI72-DO95 路开关量输入状态位。比如 bDISts[0]为“1”则使 72 通道处于“开”状态，若为“0”则置 72 通道为“关”状态。其他同理。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice **#SetDevDIODir** **#ProcessDevDIO** **#SetDeviceDO** **GetDeviceDI** ReleaseDevice

对前 48 路开关量的函数调用一般顺序

① CreateDevice
② SetDevDIODir
③ ProcessDevDIO
④ ReleaseDevice

用户可以反复执行第③步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及计数器同时进行，互不影响）。

对后 48 路开关量的函数调用一般顺序

① CreateDevice
② SetDeviceDO、SetDeviceDO(当然这两个函数也可同时进行)
③ ReleaseDevice

用户可以反复执行第②步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及计数器可以同时进行，互不影响）。

第四节、PCI 寄存器操作函数

υ取得指定寄存器的线性地址和物理地址

函数原型:

Visual C++ & C++ Builder:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PULONG LinearAddr,
                    PULONG nPort,
                    int RegisterID)
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCI2306" (ByVal hDevice as Long, _
                                              ByRef LinearAddr As Long, _
                                              ByRef nPort As Long, _
                                              ByVal RegisterID As Long, _
                                                   ) As Boolean
```

Delphi:

```
Function GetDeviceAddr(hDevice : Integer;
                       LinearAddr:Pointer;
                       nPort:Pointer;
                       RegisterID:Integer):Boolean;
  StdCall; External 'PCI2306' Name 'GetDeviceAddr';
```

Labview:

功能：取得 PCI 设备的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。

LinearAddr 指针参数，用于返回所取得的映射寄存器指向的线性地址，它可用于 WriteRegisterX 或 ReadRegisterX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。

nPort 所取得的映射寄存器指向的物理地址，它指明该设备位于系统空间的物理位置,它主要供 WritePortX 和 ReadPortX 使用。

RegisterID 指定映射寄存器的 ID 号，其取值范围为[0，5]，通常情况下，用户应使用 0 号映射寄存器，特殊情况下，我们为用户加以申明。(但需要用户特别注意的是：本设备使用了三个地址映射寄存器 0、1、3，而 0、1 号寄存器是仅供 PCI 设备本身所专用的，用户只能使用 3 号地址寄存器的物理地址 PhysAddr 来操作 AD 和开关量)。

返回值：如果执行成功，则返回 TRUE， 它表明由 RegisterID 指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回，否则会返回 FALSE，同时还要检查其 LinearAddr 和 PhysAddr 是否为 0，若为 0 则依然视为失败。用户可用 GetLastError 捕获当前错误码,并加以分析。

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

Visual C++ & C++ Builder 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
    :
```

1 以单字节(8Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

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

Visual Basic:

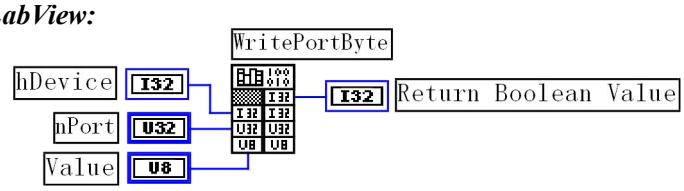
```
Declare Function WritePortByte Lib "PCI2306" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Byte) As Boolean
```

(在 Windows NT/2000 用户模式程序中直接访问 I/O 端口)

Delphi:

```
Function WritePortByte(hDevice : Integer; nPort:LongWord; Value:Byte):Boolean;
        StdCall; External 'PCI2306' Name 'WritePortByte';
```

LabView:



功能：以单字节(8Bit)方式写 I/O 端口

参数：

hDevice 设备对象句柄,它应由 CreateDevice 或 CreateDeviceEx 创建。
nPort 设备的 I/O 端口号。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
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
ReadPortULong ReleaseDevice

2 以双字(16Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

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

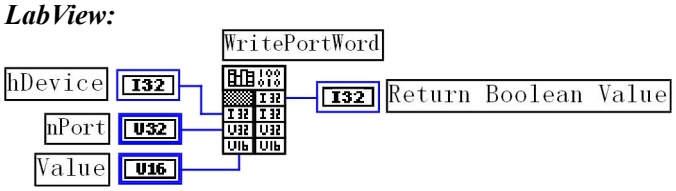
Visual Basic:

```
Declare Function WritePortWord Lib "PCI2306" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WritePortWord(hDevice : Integer; nPort:LongWord; Value:Word):Boolean;
        StdCall; External 'PCI2306' Name 'WritePortWord';
```

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。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
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
ReadPortULong ReleaseDevice

3 以四字节(32Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

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

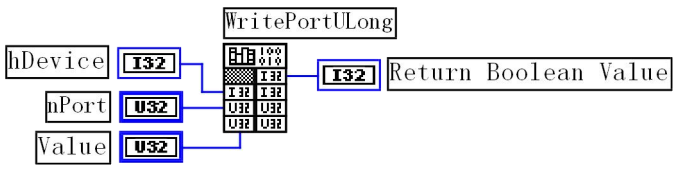
Visual Basic:

```
Declare Function WritePortULong Lib "PCI2306" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

Delphi:

```
Function WritePortULong(hDevice : Integer; nPort:LongWord; Value:LongWord):Boolean;
        StdCall; External 'PCI2306' Name 'WritePortULong';
```

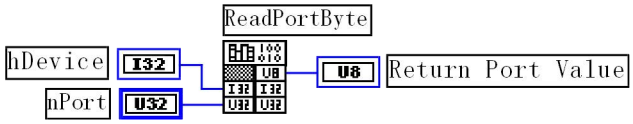
LabView:



功能：以四字节(32Bit)方式写 I/O 端口
参数：
hDevice 设备对象句柄,它应由 CreateDevice 或 CreateDeviceEx 创建。
nPort 设备的 I/O 端口号。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
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 ReadPortULong ReleaseDevice

4 以单字节(8Bit)方式读 I/O 端口
Visual C++ & C++ Builder:
```
BYTE ReadPortByte(HANDLE hDevice, UINT nPort)
```
Visual Basic:
```
Declare Function ReadPortByte Lib "PCI2306" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```
Delphi:
```
Function ReadPortByte(hDevice : Integer; nPort:LongWord):Byte;
        StdCall; External 'PCI2306' Name 'ReadPortByte';
```

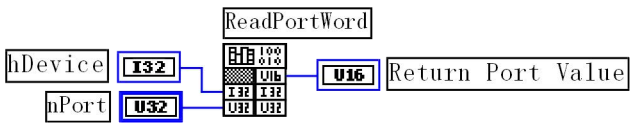
LabView:



功能：以单字节(8Bit)方式读 I/O 端口
参数：
hDevice 设备对象句柄,它应由 CreateDevice 或 CreateDeviceEx 创建。
nPort 设备的 I/O 端口号。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
返回值：返回由 nPort 指定的端口的值
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord ReadPortULong ReleaseDevice

5 以双字节(16Bit)方式读 I/O 端口
Visual C++ & C++ Builder:
```
WORD ReadPortWord(HANDLE hDevice, UINT nPort)
```
Visual Basic:
```
Declare Function ReadPortWord Lib "PCI2306" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Integer
```
Delphi:
```
Function ReadPortWord(hDevice : Integer; nPort:LongWord):Word;
        StdCall; External 'PCI2306' Name 'ReadPortWord';
```

LabView:



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

nPort 设备的 I/O 端口号。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
返回值：返回由 nPort 指定的端口的值
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord ReadPortULong ReleaseDevice

6 以四字节(32Bit)方式读 I/O 端口

Visual C++ & C++ Builder:

```
WORD ReadPortULong(HANDLE hDevice, UINT nPort)
```

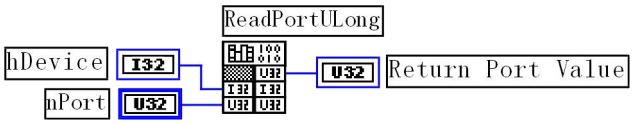
Visual Basic:

```
Declare Function ReadPortULong Lib "PCI2306" (ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer; nPort:LongWord):LongWord;
        StdCall; External 'PCI2306' Name 'ReadPortULong';
```

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。（本设备应使用 GetDeviceAddr 函数返回的 PhysAddr 地址作为基地址）
返回值：返回由 nPort 指定端口的值
相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord ReadPortULong ReleaseDevice

注意：若用户想在用户模式下直接访问硬件设备以提高访问速度，那么请您使用安装光盘上的共用驱动中的带 Ex 后缀的端口访问函数。如 WritePortByteEx、ReadPortByteEx 等。

该公用驱动位于光盘上 ISA\CommUser\App 目录下，安装即可。其驱动文件如下：
CommUser.h
CommUser.Lib
CommUser.Dll
CommUser.VxD（Win9x）
CommUser.Sys (WinNT/2000/XP)

第四章 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 公用接口函数列表

函数名	函数功能	备注
① 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateVBThread	在 VB 环境中建立子线程对象	在 VB 中可实现多线程
TerminateVBThread	终止 VB 的子线程	
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
② 文件对象操作函数		
CreataFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	

ReadFile	请求文件对象读数据到用户空间	
ReleaseFile	释放已有的文件对象	
④ 其他函数		
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备

第二节 公用接口函数原型说明

一、创建 VB 子线程

1、在 VB 环境中,创建子线程对象,以实现多线程操作

Visual Basic

```
Declare Function CreateVBThread Lib "PCI2362" (hThread As Long, _
                                               ByVal RoutineAddr As Long _
                                              ) As Boolean
```

功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题.通过该函数用户可以很轻松地实现多线程操作.

参数：

hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程,暂停线程,以及删除线程等。

RoutineAddr 作为子线程运行的函数的地址，在实际使用时,请用 AddressOf 关键字取得该子线程函数的地址,再传递给 CreateVBThread 函数。

返回值：当成功创建子线程时,返回 TRUE，且所创建的子线程为挂起状态,用户需要用 ResumeThread 函数启动它.若失败,则返回 FALSE，用户可用 GetLastError 捕获当前错误码。

相关函数： CreateVBThread
TerminateVBThread

注意: RoutineAddr 指向的函数或过程必须放在 VB 的模块文件中,如 PCI2362.Bas 文件中.

Visual Basic 程序举例:

```
' 在模块文件中定义子线程函数(注意参考实例)
Function NewRoutine() As Long        ' 定义子线程函数
        :                            ' 线程运行代码
       NewRoutine = 1   ' 返回成功码
End Function
'
' 在窗体文件中创建子线程
        :
  Dim hNewThread As Long
If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
ResumeThread (hNewThread)   '启动新线程              :
        :
```

2 在 VB 中,删除子线程对象

Visual Basic:

```
Declare Function TerminateVBThread Lib "PCI2362" (ByVal hThread As Long) As Boolean
```

功能：在 VB 中删除由 CreateVBThread 创建的子线程对象.

参数：hThread 指向需要删除的子线程对象的句柄，它应由 CreateVBThread 创建。

返回值：当成功删除子线程对象时,返回 TRUE.,否则返回 FALSE,用户可用 GetLastError 捕获当前错误码。

相关函数： CreateVBThread
TerminateVBThread

Visual Basic 程序举例:

```
        :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
        :
```

二、创建内核系统事件

Visual C++:

```
HANDLE CreateSystemEvent(void);
```

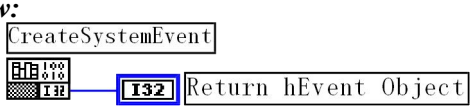
Visual Basic:

```
Declare Function CreateSystemEvent Lib " PCI2362 " () As Long
```

Delphi:

Function CreateSystemEvent():Integer; StdCall; External 'PCI2362' Name 'CreateSystemEvent';

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)。

Visual C++程序举例：

```
        :
HANDLE hEvent;
hEvent=CreateSystemEvent ()
if(hEvent==INVALID_HANDLE_VALUE)
    MessageBox("创建内核事件失败…");
    return;      // 退出该函数
}
        :
```

Visual Basic 程序举例:

```
      :
hEvent = CreateSystemEvent() ' 创建内核事件
If hEvent = INVALID_HANDLE_VALUE Then
    MsgBox "创建事件对象失败"
    Exit Sub
 End If
:
```

三、文件对象操作函数

1 初始化设备文件对象

函数原型：

Visual C++:

```
Handle CreateFileObject (
                        HANDLE hDevice,
                        LPCTSTR NewFileName,
                        int Mode)
```

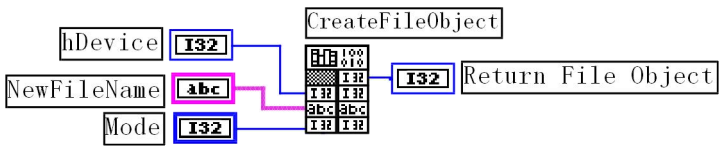
Visual Basic:

```
Declare Function CreateFileObjectLib "PCI2362" (ByVal hDevice As Long, _
                                                ByVal NewFileName As String, _
                                                ByVal Mode As Long
                                               ) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer; NewFileName: string; Mode: Integer):LongInt;
        stdcall; external 'PCI2362' name CreateFileObject;
```

LabView:



功能：初始化设备文件对象，以期待 WriteFile 请求准备文件对象进行文件操作。

参数：

hDevice： 设备对象句柄,它应由 CreateDevice 创建。

NewFileName： 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：”C:\\PCI2362\\Data.Dat”， 在 Basic 中，其语法格式如：”C:\PCI2362\Data.Dat”

Mode 文件操作方式，所用的文件操作方式控制字定义如下(可通过或指令实现多种方式并操作)

PCI2362_modeRead 只读文件方式

PCI2362_modeWrite 只写文件方式

PCI2362_modeReadWrite 既读又写文件方式

PCI2362_modeCreate 如果文件不存在，可以创建该文件，如果存在，则重建此文件，并清 0

返回值：若成功，则返回设备对象句柄.

相关函数： CreateDevice CreateFileObject
WriteFile ReadFile
ReleaseFile
ReleaseDevice

2 通过设备对象,往指定磁盘上写入用户空间的采样数据.

函数原型：

Visual C++:

```
BooL WriteFile( HANDLE hDevice,
                PWORD pDataBuffer,
                LONG nWriteSizeBytes)
```

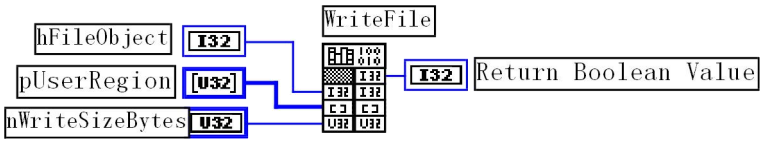
Visual Basic:

```
Declare Function WriteFile Lib "PCI2362" (By REF hDevice As Long,
                                          ByVal pDataBuffer As Integer,_
                                           ByVal nWriteSizeBytes As Long, _
                                          ) As Boolean
```

Delphi:

```
function WriteFile(hDevice : Integer;
                   pDataBuffer:PWordArray;
                   nWriteSizeBytes: LongWord):Boolean;
                   stdcall; external 'PCI2362' name 'WriteFile';
```

LabView:



功能：通过向设备对象发送“写磁盘消息”，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由 CreateFileObject 函数中的 NewFileName 指定。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pDataBuffer 用户数据空间地址。

nWriteSizeBytes 告诉设备对象往磁盘上一次写入数据的长度(以字节为单位)

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 CreateFileObject
WriteFile ReadFile
ReleaseFile
ReleaseDevice

3 通过设备对象,从指定磁盘文件中读采样数据.

函数原型：

Visual C++:

```
BOOL ReadFile( HANDLE hDevice,
               PVOID pDataBuffer,
               LONG OffsetBytes,
               LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PCI2362" (ByVal hDevice As Long, _
                                         ByRef pDataBuffer As Integer, _
                                         ByVal OffsetBytes As Long, _
                                          ByVal ReadSizeBytes As Long, _
                                         ) As Boolean
```

Delphi:

```
Function ReadFile(hDevice : Integer;
                  pDataBuffer:PWordArray;
```

```
                    OffsetBytes:LongWord;
                    nReadSizeBytes:LongWord;):Boolean;
                    stdcall; external 'PCI2362' name 'ReadFile';
```

LabView:

功能：通过向设备对象发送写磁盘消息，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由 CreateFileObject 函数中的 NewFileName 指定。

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针。

OffsetBytes 指定从文件始端起所偏移的读位置。

ReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)，其取值范围为[1, 65535]。

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 CreateFileObject
WriteFile ReadFile
ReleaseFile
ReleaseDevice

4 释放设备文件对象

函数原型：

Visual C++:

```
BOOL ReleaseFile(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseFile Lib "PCI2362" (ByVal hDevice As Long) as Boolean
```

Delphi:

```
Function ReleaseFile(hDevice : Integer):Boolean;
        stdcall; external 'PCI2362' name 'ReleaseFile';
```

LabView:

功能：释放设备文件对象。

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 CreateFileObject
WriteFile ReadFile
ReleaseFile ReleaseDevice

第三节 其他函数

一、取得指定磁盘的可用空间

Visual C++:

```
LONG GetDiskFreeBytes（LPCTSTR DiskName )
```

Visual Basic:

```
Declare Function GetDiskFreeBytes Lib "PCI2362" (ByVal DiskName As String ) As Boolean
```

LabView:

功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：需要访问的盘符,若为 C 盘为"C:\\", D 盘为"D:\\"，以此类推。

返回值：若成功，返回大于或等于 0 的长整型值,否则返回负值，用户可用 GetLastError 捕获错误码

第五章　硬件参数结构

第一节、　计数器参数结构（PCI2362_PARA_COUNTER_CTRL）

Visual C++ & C++Builder:

```
typedef struct _PCI2362_PARA_COUNTER_CTRL          // 计数器控制字(CONTROL)
{
    BYTE OperateType;   // 操作类型
    BYTE CountMode;     // 计数方式
    BYTE BCD;           // 计数类型
    // 信号源控制
    BYTE ClockSource;   // 时钟基准源选择
    BYTE GateSource;    //  GATE 门信号源选择
} PCI2362_PARA_COUNTER_CTRL,* PPCI2362_PARA_COUNTER_CTRL;
```

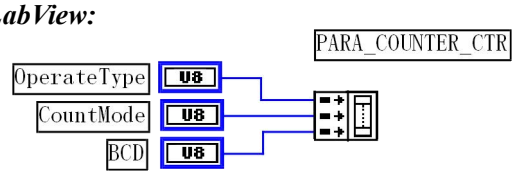
Visual Basic：

```
Type PCI2362_PARA_COUNTER_CTRL
    OperateType As Byte   ‘ 操作类型
    CountMode As Byte     ‘ 计数方式
    BCD As Byte           ‘ 计数类型
    ‘ 信号源控制
    ClockSource As Byte   ‘ 时钟基准源选择
    GateSource As Byte    ‘  GATE 门信号源选择
End Type
```

Delphi:

```
  Type   // 定义结构体数据类型
  pPCI2362_PARA_COUNTER_CTRL = ^ PCI2362_PARA_COUNTER_CTRL;   // 指针类型结构
  PCI2362_PARA_COUNTER_CTRL = record          // 标记为记录型
    OperateType : Byte;   // 操作类型
    CountMode : Byte;     // 计数方式
    BCD : Byte;           // 计数类型
    // 信号源控制
    ClockSource : Byte;   // 时钟基准源选择
    GateSource : Byte     //  GATE 门信号源选择
  End;
```

LabView:



OperateType 计数器操作类型，它的取范围应为如下常量之一。

常量名	常量值	功能定义
PCI2362_OperateType_0	00H	计数器锁存操作
PCI2362_OperateType_1	01H	只读/写低字节
PCI2362_OperateType_2	02H	只读/写高字节
PCI2362_OperateType_3	03H	先读/写低字节，后读/写高字节

CountMode 计数方式，它的取值范围应为如下常量之一。

常量名	常量值	功能定义
PCI2362_CountMode_0	00H	计数方式 0，计数器结束中断方式
PCI2362_CountMode_1	01H	计数方式 1，可编程单次脉冲方式
PCI2362_CountMode_2	02H	计数方式 2，频率发生器方式
PCI2362_CountMode_3	03H	计数方式 3，方波频率发生器方式
PCI2362_CountMode_4	04H	计数方式 4，软件触发选通方式
PCI2362_CountMode_5	05H	计数方式 5，硬件触发选通方式

BCD 计数类型，它的取值范围应为如下常量之一。

常量名	常量值	功能定义

PCI2362_BCD_0	00H	计数类型 0，二进制计数
PCI2362_BCD_1	01H	计数类型 1，BCD 码计数

ClockSource 时钟源选择，它的取值范围应为如下常量之一。

常量名	常量值	功能定义
PCI2362_IN_CLOCK	00H	内部时钟定时触发(板上 10MHz)
PCI2362_OUT_CLOCK	01H	外部时钟定时触发
PCI2362_OTHER_CLOCK	02H	其他计数器的 OUT 输出提供的 Clock(CLK2 接 OUT1,CLK1 接 OUT0, CLK0 接外部 CLK)

GateSource GATE 门信号源选择，它的取值范围应为如下常量之一。

常量名	常量值	功能定义
PCI2362_LOW_VOLT_GATE	00H	低电平(内部软件自动实现)
PCI2362_HIGH_VOLT_GATE	01H	高电平(内部软件自动实现)
PCI2362_OUTSIDE_GATE	02H	外部 GATE 信号输入(D 型头上)
PCI2362_NOUTSIDE_GATE	03H	外部 GATE 信号反向输入(D 型头上)

相关函数： InitDevCounter

第二节、 用于数字量输入输出方向参数（PCI2362_PARA_DIR）

Visual C++ & C++Builder:

```
typedef struct _PCI2362_PARA_DIR          // 数字量输入输出参数
{
    BYTE bDIO00_07Dir;   // 0 - 7 通道, =0:输入，=1:输出
    BYTE bDIO08_15Dir;   // 8 - 15 通道, =0:输入，=1:输出
    BYTE bDIO16_23Dir;   // 16 - 23 通道, =0:输入，=1:输出
    BYTE bDIO24_31Dir;   // 24 - 31 通道, =0:输入，=1:输出
    BYTE bDIO32_40Dir;   // 32 - 40 通道, =0:输入，=1:输出
    BYTE bDIO41_47Dir;   // 41 - 47 通道, =0:输入，=1:输出
} PCI2362_PARA_DIR,*PPCI2362_PARA_DIR;
```

Visual Basic：

```
Type PCI2362_PARA_DIR
    bDIO00_07Dir As Byte  ‘ 0 - 7 通道, =0:输入，=1:输出
    bDIO08_15Dir As Byte  ‘8 - 15 通道, =0:输入，=1:输出
    bDIO16_23Dir As Byte  ‘16 - 23 通道, =0:输入，=1:输出
    bDIO24_31Dir As Byte  ‘24 - 31 通道, =0:输入，=1:输出
    bDIO32_40Dir As Byte  ‘32 - 40 通道, =0:输入，=1:输出
    bDIO41_47Dir As Byte  ‘41 - 47 通道, =0:输入，=1:输出
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
  P PCI2362_PARA_DIR = ^ PCI2362_PARA_DIR;   // 指针类型结构
  PCI2362_PARA_DIR = record          // 标记为记录型
    bDIO00_07Dir As Byte;   // 0 - 7 通道, =0:输入，=1:输出
    bDIO08_15Dir As Byte;   // 8 - 15 通道, =0:输入，=1:输出
    bDIO16_23Dir As Byte;   // 16 - 23 通道, =0:输入，=1:输出
    bDIO24_31Dir As Byte;   // 24 - 31 通道, =0:输入，=1:输出
    bDIO32_40Dir As Byte;   // 32 - 40 通道, =0:输入，=1:输出
    bDIO41_47Dir As Byte;   // 41 - 47 通道, =0:输入，=1:输出 End;
```

LabView:

该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要象 Visual Basic 中的属性操作那样，简单的进行属性赋值，然后执行 SetDevDIODir 即可完成数字量输入输出方向的设定。

其每一个成员变量对应于一组开关量的状态，且这些成员变量只能被赋值为“0”或“1”数值。“0”代表输入状态，“1”代表输出状态。

相关函数: **#SetDevDIODir** **#ProcessDevDIO** **#SetDeviceDO** **GetDeviceDI** ReleaseDevice

第六章 上层用户函数接口应用实例

第一节、怎样实现计数器 Counter 的操作

Visual C++ & C++Builder:

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

[程序]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[简易代码演示]丨[Counter 的演示]

第二节、怎样实现开关量 DIO 的操作

Visual C++ & C++Builder:

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

[程序]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[简易代码演示]丨[DIO 的演示]

其他语言类同

第七章 PCI2362 的 DOS 驱动程序说明

本处只例举 TurboC 和 Borland C 语言的使用方法。首先将 PCI2362.H 头文件包含进您的源程序。

第一节、如何取得设备地址

使用 GetDeviceAddr 函数，它返回本设备的物理地址 PhysAddr。

第二节、如何访问用户寄存器

将 GetDeviceAddr 函数返回的 PhysAddr 地址再加上相应寄存器的偏移地址然后传递给 outpw 或 inpw 函数。即可实现对用户寄存器的访问。

第三节、程序演示

请见提供的源程序。

第八章 LabView 及 LabWindows 驱动程序说明

请参考其源程序