

# PCI8502 同步采集卡

WIN2000/XP/WIN7 驱动程序使用说明书



阿尔泰科技发展有限公司

产品研发部修订

请您务必阅读《[使用纲要](#)》，他会使您事半功倍！

## 目 录

|                                           |    |
|-------------------------------------------|----|
| 第一章 版权信息与命名约定.....                        | 2  |
| 第一节、版权信息.....                             | 2  |
| 第二节、命名约定.....                             | 2  |
| 第二章 使用纲要.....                             | 2  |
| 第一节、使用上层用户函数，高效、简单.....                   | 2  |
| 第二节、如何管理 PCI 设备.....                      | 2  |
| 第三节、怎么进行 AD 数采操作.....                     | 2  |
| 第四节、哪些函数对您不是必须的.....                      | 3  |
| 第三章 PCI 即插即用设备操作函数接口介绍.....               | 5  |
| 第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI8502_”）..... | 6  |
| 第二节、设备对象管理函数原型说明.....                     | 7  |
| 第三节、AD 数据读取函数原型说明.....                    | 9  |
| 第四节、AD 硬件参数保存与读取函数原型说明.....               | 12 |
| 第四章 硬件参数结构.....                           | 14 |
| 第一节、AD 硬件参数介绍（PCI8502_PARA_AD）.....       | 14 |
| 第二节、AD 状态参数结构（PCI8502_STATUS_AD）.....     | 16 |
| 第五章 数据格式转换与排列规则.....                      | 17 |
| 第一节、AD 原码 LSB 数据转换成电压值的换算方法.....          | 17 |
| 第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则.....    | 17 |
| 第三节、AD 测试应用程序创建并形成的数据文件格式.....            | 17 |
| 第六章 上层用户函数接口应用实例.....                     | 18 |
| 第一节、简易程序演示说明.....                         | 18 |
| 一、怎么进行 AD 数采操作.....                       | 18 |
| 第二节、高级程序演示说明.....                         | 18 |
| 第一节、公用接口函数总列表（每个函数省略了前缀“PCI8502_”）.....   | 18 |
| 第二节、PCI 内存映射寄存器操作函数原型说明.....              | 18 |
| 第三节、IO 端口读写函数原型说明.....                    | 24 |
| 第四节、线程操作函数原型说明.....                       | 27 |

## 第一章 版权信息与命名约定

### 第一节、版权信息

本软件产品及相关套件均属北京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当地代理商联系或直接与我们联系，我们将热情接待。

### 第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 `PCIxxxx_` 则被省略。如 `PCI8502_CreateDevice` 则写为 `CreateDevice`。

二、函数名及参数中各种关键字缩写规则

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

以上规则不局限于该产品。

## 第二章 使用纲要

### 第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如 Win32 API 的函数，具有相当的灵活性、可靠性和高效性。诸如 [InitDeviceAD](#)、[ReadDeviceAD](#) 等。而底层用户函数如 [WriteRegisterULong](#)、[ReadRegisterULong](#)、[WritePortByte](#)、[ReadtByte](#)……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插座等管脚分配情况。

### 第二节、如何管理 PCI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 [CreateDevice](#) 函数创建一个设备对象句柄 `hDevice`，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如 [InitDeviceAD](#) 可以使用 `hDevice` 句柄以程序查询方式初始化设备的 AD 部件，[ReadDeviceAD](#) 函数可以用 `hDevice` 句柄实现对 AD 数据的采样读取等。最后可以通过 [ReleaseDevice](#) 将 `hDevice` 释放掉。

### 第三节、怎么进行 AD 数采操作

当您有了 `hDevice` 设备对象句柄后，便可用 [InitDeviceAD](#) 函数初始化 AD 部件，关于采样通道、频率等的参数的设置是由这个函数的 `pADPara` 参数结构体决定的。您只需要对这个 `pADPara` 参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用 [StartDeviceAD](#) 即可启动 AD 部件，开始 AD 采样，然后

便可用 [ReadDeviceAD](#) 反复读取 AD 数据以实现连续不间断采样。当您需要暂停设备时，执行 [StopDeviceAD](#)，当您需要关闭 AD 设备时，[ReleaseDeviceAD](#) 便可帮您实现（但设备对象 hDevice 依然存在）。具体执行流程请看图 2.1.2。

注意：图中较粗的虚线表示对称关系。如红色虚线表示 [CreateDevice](#) 和 [ReleaseDevice](#) 两个函数的关系是：最初执行一次 [CreateDevice](#)，在结束是就须执行一次 [ReleaseDevice](#)。

#### 第四节、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，这些函数您可完全不必理会，除非您是作为底层用户管理设备。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。对 PCI 用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在 NT、Win2000 等操作系统中实现对您原有传统设备如 ISA 卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于 Windows NT 架构的操作系统中无法继续使用您原有的老设备。

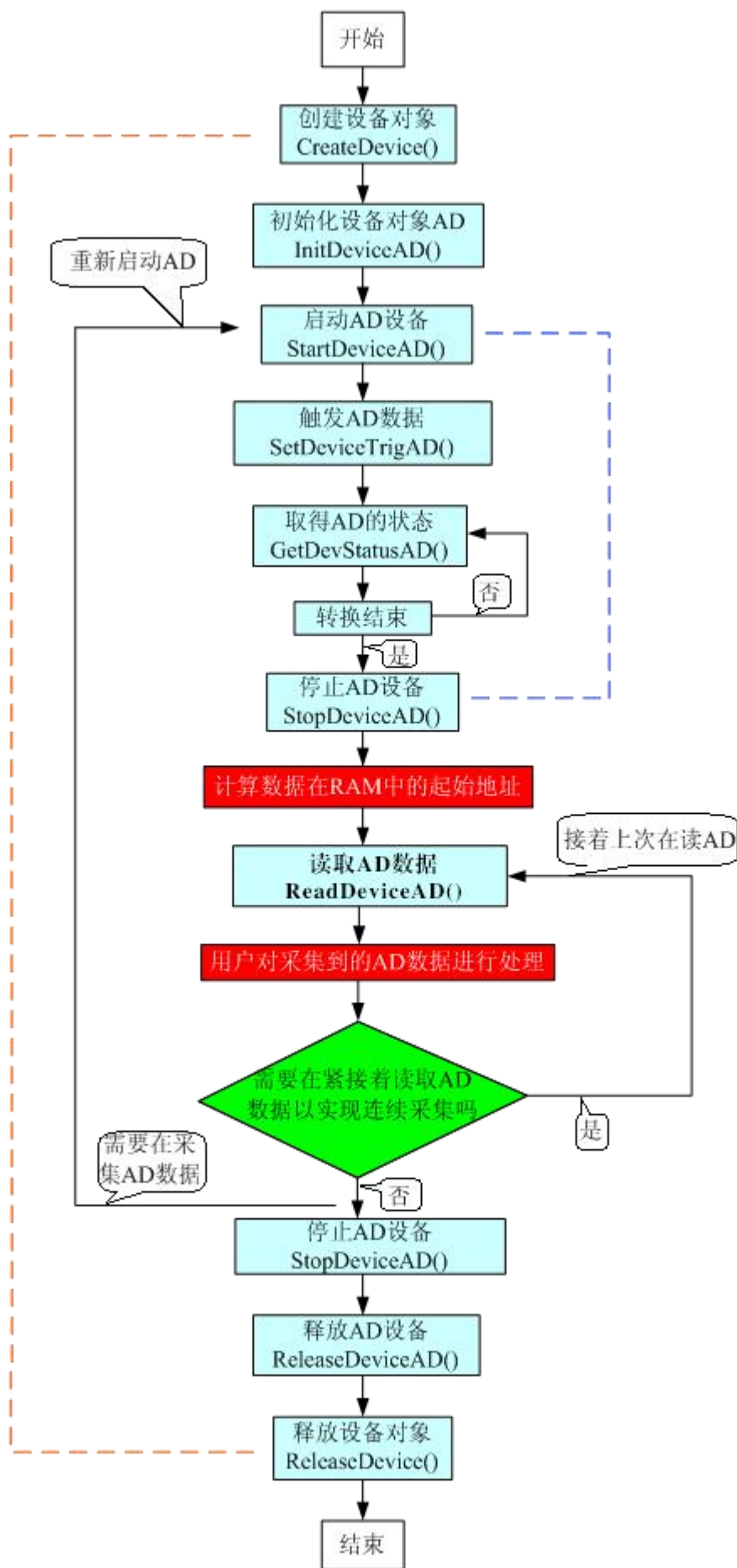


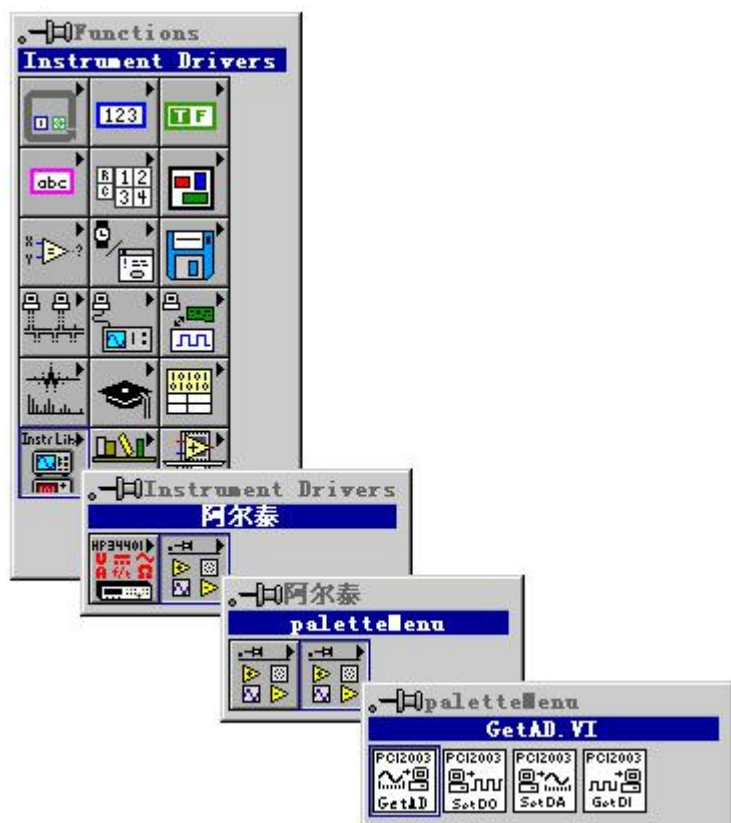
图 2.1.2 怎么进行 AD 数采操作

### 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、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的 AD 数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如 [InitDeviceAD](#)）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用 [ReadDeviceAD](#) 函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的 Bit 位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉 PCI 总线复杂的控制协议，同时还可以省掉您许多繁琐的工作。

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



LabView 内嵌式驱动接口的获取方法

## 第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI8502\_”）

| 函数名                                | 函数功能                             | 备注      |
|------------------------------------|----------------------------------|---------|
| <b>① 设备对象操作函数</b>                  |                                  |         |
| <a href="#">CreateDevice</a>       | 创建 PCI 设备对象(用设备逻辑号)              | 上层及底层用户 |
| <a href="#">GetDeviceCount</a>     | 取得同一种 PCI 设备的总台数                 | 上层及底层用户 |
| <a href="#">GetDeviceCurrentID</a> | 取得指定设备的逻辑 ID                     | 上层及底层用户 |
| <a href="#">ListDeviceDlg</a>      | 列表所有同一种 PCI 设备的各种配置              | 上层及底层用户 |
| <a href="#">ReleaseDevice</a>      | 关闭设备，且释放 PCI 总线设备对象              | 上层及底层用户 |
| <b>②AD 数据读取函数</b>                  |                                  |         |
| <a href="#">GetDDR2Length</a>      | 获取 DDR2 的存储长度                    | 上层用户    |
| <a href="#">ADCalibration</a>      | AD 校准                            | 上层用户    |
| <a href="#">InitDeviceAD</a>       | 初始化 AD 部件准备传输                    | 上层用户    |
| <a href="#">StartDeviceAD</a>      | 启动 AD 设备，开始转换                    | 上层用户    |
| <a href="#">SetDeviceTrigAD</a>    | 当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效） | 上层用户    |
| <a href="#">GetDevStatusAD</a>     | 取得当前设备状态                         | 上层用户    |
| <a href="#">ReadDeviceAD</a>       | 读取 AD 数据                         | 上层用户    |
| <a href="#">StopDeviceAD</a>       | 暂停 AD 设备                         | 上层用户    |
| <a href="#">ReleaseDeviceAD</a>    | 释放设备上的 AD 部件                     | 上层用户    |
| <b>③ AD 硬件参数系统保存、读取函数</b>          |                                  |         |
| <a href="#">LoadParaAD</a>         | 从 Windows 系统中读入硬件参数              | 上层用户    |
| <a href="#">SaveParaAD</a>         | 往 Windows 系统写入设备硬件参数             | 上层用户    |
| <a href="#">ResetParaAD</a>        | 将注册表中的 AD 参数恢复至出厂默认值             | 上层用户    |

### 使用需知：

#### Visual C++:

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

```
#include "C:\Art\PCI8502\INCLUDE\PCI8502.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 PCI8502.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

#### Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(\*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话框中选择 PCI8502.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 PCI8502.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标，然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。
- 二、根据 LabView 语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如 [ReadDeviceAD](#) 接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。
- 三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“[U16]”为无符号 16 位短整型数组或缓冲区或指针，“[U32]”与 “[U16]”同理，只是位数不一样。

## 第二节、设备对象管理函数原型说明

### ◆ 创建设备对象函数（逻辑号）

函数原型：

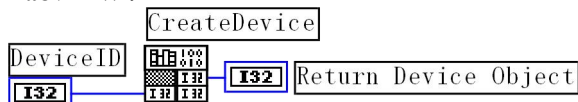
**Visual C++:**

[HANDLE CreateDevice \(int DeviceID = 0\)](#)

**Visual Basic:**

[Declare Function CreateDevice Lib "PCI8502\\_32" \(Optional ByVal DeviceID As Integer = 0\) 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：**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设备时，系统将以该设备的“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。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 INVALID\_HANDLE\_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

**相关函数：** [CreateDevice](#)      [GetDeviceCount](#)      [GetDeviceCurrentID](#)  
[ListDeviceDlg](#)      [ReleaseDevice](#)

**Visual C++ 程序举例：**

```
HANDLE hDevice; // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PCI8502_CreateDevice (DeviceLgcID); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALID_HANDLE_VALUE); // 判断设备对象句柄是否有效
{
    return; // 退出该函数
}
:
```

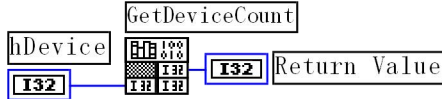
**Visual Basic 程序举例**

```
:
Dim hDevice As Long ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PCI8502_CreateDevice (DeviceLgcID) ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub ' 退出该过程
End If
```

:

## ◆ 取得本计算机系统中 PCI8502 设备的总数量

函数原型:

**Visual C++:**`int GetDeviceCount (HANDLE hDevice)`**Visual Basic:**`Declare Function GetDeviceCount Lib "PCI8502_32" (ByVal hDevice As Long) As Integer`**LabVIEW:**

功能: 取得 PCI8502 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 PCI8502 的数量。

相关函数: [CreateDevice](#)                      [GetDeviceCount](#)                      [GetDeviceCurrentID](#)  
[ListDeviceDlg](#)                      [ReleaseDevice](#)

## ◆ 取得该设备当前逻辑 ID

函数原型:

**Visual C++:**
`BOOL GetDeviceCurrentID (HANDLE hDevice,  
                              PLONG DeviceLgcID)`
**Visual Basic:**
`Declare Function GetDeviceCurrentID Lib "PCI8502_32" (ByVal hDevice As Long, _  
                                                          ByRef DeviceLgcID As Long) As Boolean`
**LabVIEW:**

请参考相关演示程序。

功能: 取得指定设备逻辑 ID 号。

参数:

hDevice 设备对象句柄, 它指向要取得逻辑号的设备, 它应由 [CreateDevice](#) 创建。

DeviceLgcID 返回设备的逻辑 ID, 它的取值范围为[0, 15]。

返回值: 如果初始化设备对象成功, 则返回 TRUE, 否则返回 FALSE, 用户可用 GetLastErrorEx 捕获当前错误码, 并加以分析。

相关函数: [CreateDevice](#)                      [GetDeviceCount](#)                      [GetDeviceCurrentID](#)  
[ListDeviceDlg](#)                      [ReleaseDevice](#)

## ◆ 用对话框控件列表计算机系统中所有 PCI8502 设备各种配置信息

函数原型:

**Visual C++:**`BOOL ListDeviceDlg (HANDLE hDevice)`**Visual Basic:**`Declare Function ListDeviceDlg Lib "PCI8502_32" (ByVal hDevice As Long) As Boolean`**LabVIEW:**

请参考相关演示程序。

功能: 列表系统中 PCI8502 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 PCI8502 设备的配置情况。

相关函数: [CreateDevice](#)                      [ReleaseDevice](#)

## ◆ 释放设备对象所占的系统资源及设备对象

函数原型:

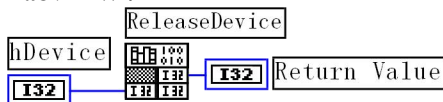
**Visual C++:**

BOOL ReleaseDevice(HANDLE hDevice)

**Visual Basic:**

Declare Function ReleaseDevice Lib "PCI8502\_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 GetLastErrorEx 捕获错误码。

**相关函数:** [CreateDevice](#)

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

### 第三节、AD 数据读取函数原型说明

#### ◆ 设备校准函数

函数原型:

**Visual C++:**

BOOL ADCalibration(HANDLE hDevice)

**Visual Basic:**

Declare Function ADCalibration Lib "PCI8502\_32" (ByVal hDevice As Long) As Boolean

**LabVIEW:**

请参考相关演示程序。

**功能:** 设备校准函数。

**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, 用户可以用 GetLastErrorEx 捕获错误码。

**相关函数:** [CreateDevice](#)

#### ◆ 初始化设备对象

函数原型:

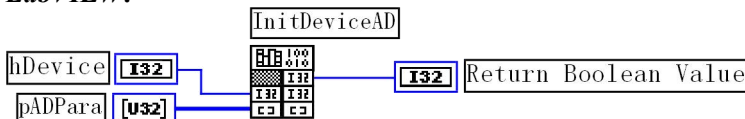
**Visual C++:**

BOOL InitDeviceAD(HANDLE hDevice,  
PCI8502\_PARA\_AD pADPara)

**Visual Basic:**

Declare Function InitDeviceAD Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pADPara As PCI8502\_PARA\_AD) As Boolean

**LabVIEW:**



**功能:** 它负责初始化设备对象中的 AD 部件, 为设备操作就绪有关工作, 然后启动 AD 设备开始 AD 采集, 随后, 用户便可以连续调用 [ReadDeviceAD](#) 读取设备上的 AD 数据以实现连续采集。

**参数:**

hDevice 设备对象句柄, 它应由设备的 [CreateDevice](#) 创建。

pADPara 设备对象参数结构, 它决定了设备对象的各种状态及工作方式。请参考《[AD 硬件参数介绍](#)》。

**返回值:** 如果初始化设备对象成功,则返回 TRUE,且 AD 便被启动。否则返回 FALSE,用户可用 GetLastErrorEx 捕获当前错误码,并加以分析。

**相关函数:**

|                                 |                                |                                 |
|---------------------------------|--------------------------------|---------------------------------|
| <a href="#">CreateDevice</a>    | <a href="#">InitDeviceAD</a>   | <a href="#">StartDeviceAD</a>   |
| <a href="#">SetDeviceTrigAD</a> | <a href="#">GetDevStatusAD</a> | <a href="#">ReadDeviceAD</a>    |
| <a href="#">ReleaseDevice</a>   | <a href="#">topDeviceAD</a>    | <a href="#">ReleaseDeviceAD</a> |

注意: 该函数将试图占用系统的某些资源,如系统内存区、DMA 资源等。所以当用户在反复进行数据采集之前,只须执行一次该函数即可,否则某些资源将会发生使用上的冲突,便会失败。除非用户执行了 [ReleaseDeviceAD](#) 函数后,再重新开始设备对象操作时,可以再执行该函数。所以该函数切忌不要单独放在循环语句中反复执行,除非和 [ReleaseDeviceAD](#) 配对。

#### ◆ 启动 AD 设备(Start device AD for program mode)

函数原型:

**Visual C++:**

BOOL StartDeviceAD (HANDLE hDevice)

**Visual Basic:**

Declare Function StartDeviceAD Lib "PCI8502\_32" (ByVal hDevice As Long) As Boolean

**LabVIEW:**

请参考相关演示程序。

**功能:** 启动 AD 设备,它必须在调用 [InitDeviceAD](#) 后才能调用此函数。该函数除了启动 AD 设备开始转换以外,不改变设备的其他任何状态。

**参数:** hDevice 设备对象句柄,它应由 [CreateDevice](#) 创建。

**返回值:** 如果调用成功,则返回 TRUE,且 AD 立刻开始转换,否则返回 FALSE,用户可用 GetLastErrorEx 捕获当前错误码,并加以分析。

**相关函数:**

|                                 |                                |                                 |
|---------------------------------|--------------------------------|---------------------------------|
| <a href="#">CreateDevice</a>    | <a href="#">InitDeviceAD</a>   | <a href="#">StartDeviceAD</a>   |
| <a href="#">SetDeviceTrigAD</a> | <a href="#">GetDevStatusAD</a> | <a href="#">ReadDeviceAD</a>    |
| <a href="#">ReleaseDevice</a>   | <a href="#">topDeviceAD</a>    | <a href="#">ReleaseDeviceAD</a> |

#### ◆ 产生软件触发事件

函数原型:

**Visual C++**

BOOL SetDeviceTrigAD (HANDLE hDevice)

**Visual Basic:**

Declare Function SetDeviceTrigAD Lib "PCI8502\_32" (ByVal hDevice As Long) As Boolean

**LabVIEW:**

请参考相关演示程序。

**功能:** 当设备使能允许后,产生软件触发事件(只有触发源为软件触发时有效)。

**参数:** hDevice 设备对象句柄,它应由 [CreateDevice](#) 创建。

**返回值:** 如果调用成功,则返回 TRUE,否则返回 FALSE,用户可用 GetLastErrorEx 捕获当前错误码,并加以分析。

**相关函数:**

|                                 |                                |                                 |
|---------------------------------|--------------------------------|---------------------------------|
| <a href="#">CreateDevice</a>    | <a href="#">InitDeviceAD</a>   | <a href="#">StartDeviceAD</a>   |
| <a href="#">SetDeviceTrigAD</a> | <a href="#">GetDevStatusAD</a> | <a href="#">ReadDeviceAD</a>    |
| <a href="#">ReleaseDevice</a>   | <a href="#">topDeviceAD</a>    | <a href="#">ReleaseDeviceAD</a> |

#### ◆ 取得 AD 状态标志

函数原型:

**Visual C++**

BOOL GetDevStatusAD (HANDLE hDevice,  
PCI8502\_STATUS\_AD pADStatus);

**Visual Basic:**

Declare Function GetDevStatusAD Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pADStatus As PCI8502\_STATUS\_AD) As Boolean

**LabVIEW:**

请参考相关演示程序。

**功能：**一旦用户使用 StartDeviceAD 后，应立即用此函数查询存储器的状态。

**参数：**

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pADStatus 获得 AD 的各种当前状态。它属于结构体，具体定义请参考《[AD 状态参数结构 \(PCI8502\\_STATUS\\_AD\)](#)》章节。

**返回值：**若调用成功则返回 TRUE，否则返回 FALSE，用户可以调用 GetLastErrorEx 函数取得当前错误码。

**相关函数：**

|                                 |                                |                                 |
|---------------------------------|--------------------------------|---------------------------------|
| <a href="#">CreateDevice</a>    | <a href="#">InitDeviceAD</a>   | <a href="#">StartDeviceAD</a>   |
| <a href="#">SetDeviceTrigAD</a> | <a href="#">GetDevStatusAD</a> | <a href="#">ReadDeviceAD</a>    |
| <a href="#">ReleaseDevice</a>   | <a href="#">topDeviceAD</a>    | <a href="#">ReleaseDeviceAD</a> |

#### ◆ 怎么进行 AD 数采操作

函数原型：

**Visual C++:**

```
BOOL ReadDeviceAD(HANDLE hDevice,  
                  PWORD pADBuffer,  
                  ULONG nReadSizeWords,  
                  ULONG ulStartAddr,  
                  PLONG nRetSizeWords);
```

**Visual Basic:**

```
Declare Function ReadDeviceAD Lib "PCI8502_32" ( _  
    ByVal hDevice As Long, _  
    ByRef pADBuffer As Integer, _  
    ByVal nReadSizeWords As Long, _  
    ByVal ulStartAddr As Long, _  
    ByRef nRetSizeWords As Long) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

**功能：**怎么进行 AD 数采操作。

**参数：**

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pADBuffer 将用于接受数据的用户缓冲区(该区必须开辟大于 M 加 N 个字的空间)。关于如何将这些 AD 数据转换成相应的电压值，请参考《[数据格式转换与排列规则](#)》。

nReadSizeWords 指定一次 ReadDeviceAD 操作应读取多少字数据到用户缓冲区。必须等于 M 加 N 的长度。

ulStartAddr 数据在 RAM 中的起始地址。

nRetSizeWords 返回实际读取的数据长度。

**返回值：**其返回值表示所成功读取的数据点数(字)，也表示当前读操作在 ADBuffer 缓冲区中的有效数据量。通常情况下其返回值应与 ReadSizeWords 参数指定量的数据长度(字)相等，除非用户在这个读操作以外的其他线程中执行了 ReleaseDeviceAD 函数中断了读操作，否则设备可能有问题。对于返回值不等于 nReadSizeWords 参数的，用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

注释：此函数也可用于单点读取和几个点的读取，只需要将 nReadSizeWords 设置成 1 或相应值即可。

**相关函数：**

|                                 |                                |                                 |
|---------------------------------|--------------------------------|---------------------------------|
| <a href="#">CreateDevice</a>    | <a href="#">InitDeviceAD</a>   | <a href="#">StartDeviceAD</a>   |
| <a href="#">SetDeviceTrigAD</a> | <a href="#">GetDevStatusAD</a> | <a href="#">ReadDeviceAD</a>    |
| <a href="#">ReleaseDevice</a>   | <a href="#">topDeviceAD</a>    | <a href="#">ReleaseDeviceAD</a> |

#### ◆ 暂停 AD 设备

函数原型：

**Visual C++:**

```
BOOL StopDeviceAD (HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function StopDeviceAD Lib "PCI8502_32" (ByVal hDevice As Long) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 暂停 AD 设备。它必须在调用 [StartDeviceAD](#) 后才能调用此函数。该函数除了停止 AD 设备不再转换以外, 不改变设备的其他任何状态。此后您可再调用 [StartDeviceAD](#) 函数重新启动 AD, 此时 AD 会按照暂停以前的状态 (如 FIFO 存储器位置、通道位置) 开始转换。

**参数:** [hDevice](#) 设备对象句柄, 它应由 [CreateDevice](#) 创建。

**返回值:** 如果调用成功, 则返回 TRUE, 且 AD 立刻停止转换, 否则返回 FALSE, 用户可用 [GetLastErrorEx](#) 捕获当前错误码, 并加以分析。

**相关函数:** [CreateDevice](#)      [InitDeviceAD](#)      [StartDeviceAD](#)  
[SetDeviceTrigAD](#)      [GetDevStatusAD](#)      [ReadDeviceAD](#)  
[ReleaseDevice](#)      [topDeviceAD](#)      [ReleaseDeviceAD](#)

#### ◆ 释放设备上的 AD 部件

函数原型:

**Visual C++:**

[BOOL ReleaseDeviceAD\(HANDLE hDevice\)](#)

**Visual Basic:**

[Declare Function ReleaseDeviceAD Lib "PCI8502\\_32" \(ByVal hDevice As Long\) As Boolean](#)

**LabVIEW:**

请参考相关演示程序。

**功能:** 释放设备上的 AD 部件。

**参数:** [hDevice](#) 设备对象句柄, 它应由 [CreateDevice](#) 创建。

**返回值:** 若成功, 则返回 TRUE, 否则返回 FALSE, 用户可以用 [GetLastErrorEx](#) 捕获错误码。

应注意的是, [InitDeviceAD](#) 必须和 [ReleaseDeviceAD](#) 函数一一对应, 即当您执行了一次 [InitDeviceAD](#) 后, 再一次执行这些函数前, 必须执行一次 [ReleaseDeviceAD](#) 函数, 以释放由 [InitDeviceAD](#) 占用的系统软硬件资源, 如映射寄存器地址、系统内存等。只有这样, 当您再次调用 [InitDeviceAD](#) 函数时, 那些软硬件资源才可被再次使用。

**相关函数:** [CreateDevice](#)      [InitDeviceAD](#)      [ReleaseDeviceAD](#)  
[ReleaseDevice](#)

#### ◆ AD 读取函数一般调用顺序

- ① [CreateDevice](#)
- ② [InitDeviceAD](#)
- ③ [StartDeviceAD](#)
- ④ [GetDevStatusAD](#)
- ⑤ [StopDeviceAD](#)
- ⑥ [Read Device AD](#)
- ⑦ [ReleaseDeviceAD](#)
- ⑧ [ReleaseDevice](#)

注明: 用户可以反复执行第⑥步, 以实现采集。

关于两个过程的图形说明请参考《[使用纲要](#)》。

### 第四节、AD 硬件参数保存与读取函数原型说明

#### ◆ 从 Windows 系统中读入硬件参数函数

函数原型:

**Visual C++:**

[BOOL LoadParaAD\(HANDLE hDevice,  
\[PCI8502\\\_PARA\\\_AD pADPara\\)\]\(#\)](#)

**Visual Basic:**

[Declare Function LoadParaAD Lib "PCI8502\\_32" \(ByVal hDevice As Long, \\_  
\[ByRef pADPara As PCI8502\\\_PARA\\\_AD\\) As Boolean\]\(#\)](#)

**LabVIEW:**

请参考相关演示程序。

参数:

**返回值：**若成功，返回 TRUE，否则返回 FALSE。

### ◆ 往 Windows 系统写入设备硬件参数函数

请参考相关演示程序。

参数:

**返回值：**若成功，返回 TRUE，否则返回 FALSE。

### ◆ AD 采样参数复位至出厂默认值函数

请参考相关演示程序。

参数:

**返回值：**若成功，返回 TRUE，否则返回 FALSE。

13

## 第四章 硬件参数结构

### 第一节、AD 硬件参数介绍 (PCI8502\_PARA\_AD)

**Visual C++:**

```
typedef struct _PCI8502_PARA_AD
{
    LONG Frequency;           // 采集频率,单位为 Hz, [10, 40000000]
    LONG InputRange[4];       // 模拟量输入量程选择
    LONG CouplingType;        // 耦合类型(直流耦合, 交流耦合)
    LONG M_Length;           // M 段长度(字)
    LONG N_Length;           // N 段长度(字)
    LONG TriggerMode;         // 触发模式选择
    LONG ATRTriggerChannel;   // ATR 通道选择
    LONG TriggerSource;       // 触发源选择
    LONG TriggerDir;          // 触发方向选择(正向/负向触发)
    LONG TrigLevelVolt;       // 触发电平(0--10000mV)
    LONG ClockSource;         // 时钟源选择(内/外时钟源)
    LONG OutClockSource;      // 时钟输入输出源
    LONG bClockSourceDir;     // 是否将时钟信号输出到 PXI 总线,=TRUE:允许输出,=FALSE:允许
    输入
} PCI8502_PARA_AD, *PPCI8502_PARA_AD;
```

**Visual Basic:**

```
Type PCI8502_PARA_AD
    Frequency As Long           ' 采集频率,单位为 Hz, [10, 40000000]
    InputRange(0 To 3) As Long  ' 模拟量输入量程选择
    CouplingType As Long        ' 耦合类型(直流耦合, 交流耦合)
    M_Length As Long           ' M 段长度(字),
    N_Length As Long           ' N 段长度(字),
    TriggerMode As Long         ' 触发模式选择
    ATRTriggerChannel As Long   ' ATR 通道选择
    TriggerSource As Long       ' 触发源选择
    TriggerDir As Long          ' 触发方向选择(正向/负向触发)
    TrigLevelVolt As Long       ' 触发电平(0--10000mV)
    ClockSource As Long         ' 时钟源选择(内/外时钟源)
    OutClockSource As Long      ' 时钟输入输出源
    bClockSourceDir As Long     ' 是否将时钟信号输出到 PXI 总线,=TRUE:允许输出,=FALSE:允
    许输入
End Type
```

**LabVIEW:**

请参考相关演示程序。

此结构主要用于设定设备 AD 硬件参数值, 用这个参数结构对设备进行硬件配置完全由 [InitDeviceAD](#) 函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

[InputRange](#) 模拟量输入量程选择。

| 常量名                         | 常量值  | 功能定义     |
|-----------------------------|------|----------|
| PCI8502_INPUT_N1000_P1000mV | 0x00 | ± 1000mV |
| PCI8502_INPUT_N5000_P5000mV | 0x01 | ± 5000mV |

[Frequency](#) 采集频率, 单位为 Hz, 取值范围为[10, 40000000]。

[CouplingType](#) 耦合类型(直流耦合, 交流耦合)选择。

| 常量名 | 常量值 | 功能定义 |
|-----|-----|------|
|-----|-----|------|

|                     |      |      |
|---------------------|------|------|
| PCI8502_COUPLING_AC | 0x00 | 交流耦合 |
| PCI8502_COUPLING_DC | 0x01 | 直流耦合 |

**M\_Length** M 段长度(字)，总的取值范围 1-32M，但是 M 加 N 长度不能大于 32M。

**N\_Length** N 段长度(字)，总的取值范围 1-32M，但是 M 加 N 长度不能大于 32M。

**TriggerMode** AD 触发模式。

| 常量名                    | 常量值  | 功能定义   |
|------------------------|------|--------|
| PCI8502_TRIGMODE_MIDL  | 0x00 | 中间触发   |
| PCI8502_TRIGMODE_POST  | 0x01 | 后触发    |
| PCI8502_TRIGMODE_PRE   | 0x02 | 预触发    |
| PCI8502_TRIGMODE_DELAY | 0x03 | 硬件延时触发 |

**ATRTriggerChannel** ATR 通道选择。

| 常量名                 | 常量值  | 功能定义             |
|---------------------|------|------------------|
| PCI8502_ATRTRIG_CH0 | 0x00 | 通道 0 信号作为 ATR 输入 |
| PCI8502_ATRTRIG_CH1 | 0x01 | 通道 1 信号作为 ATR 输入 |
| PCI8502_ATRTRIG_CH2 | 0x02 | 通道 2 信号作为 ATR 输入 |
| PCI8502_ATRTRIG_CH3 | 0x03 | 通道 3 信号作为 ATR 输入 |

**TriggerSource** AD 触发源。

| 常量名                     | 常量值  | 功能定义                 |
|-------------------------|------|----------------------|
| PCI8502_TRIGMODE_SOFT   | 0x00 | 软件触发                 |
| PCI8502_TRIGSRC_DTR     | 0x01 | 选择 DTR 作为触发源         |
| PCI8502_TRIGSRC_ATR     | 0x02 | 选择 ATR 作为触发源         |
| PCI8502_TRIGSRC_TRIGGER | 0x03 | Trigger 信号触发（用于多卡同步） |

**TriggerDir** AD 触发方向。它的选项值如下表：

| 常量名                         | 常量值  | 功能定义    |
|-----------------------------|------|---------|
| PCI8502_TRIGDIR_NEGATIVE    | 0x00 | 下降沿触发   |
| PCI8502_TRIGDIR_POSITIVE    | 0x01 | 上升沿触发   |
| PCI8502_TRIGDIR_POSIT_NEGAT | 0x02 | 上下边沿均触发 |

注明：PCI8502\_TRIGDIR\_POSIT\_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。

**TrigLevelVolt** 触发电平，范围为 0--10000mV。

**ClockSource** AD 时钟源选择。它的选项值如下表：

| 常量名                  | 常量值  | 功能定义   |
|----------------------|------|--------|
| PCI8502_CLOCKSRC_IN  | 0x00 | 使用内部时钟 |
| PCI8502_CLOCKSRC_OUT | 0x01 | 使用外部时钟 |

**OutClockSource** 时钟输入输出源。

| 常量名                          | 常量值  | 功能定义                      |
|------------------------------|------|---------------------------|
| PCI8502_OUTCLOCKSRC_TRIGGER0 | 0x00 | 选择 PCI 总线上的 TRIG0 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER1 | 0x01 | 选择 PCI 总线上的 TRIG1 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER2 | 0x02 | 选择 PCI 总线上的 TRIG2 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER3 | 0x03 | 选择 PCI 总线上的 TRIG3 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER4 | 0x04 | 选择 PCI 总线上的 TRIG4 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER5 | 0x05 | 选择 PCI 总线上的 TRIG5 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER6 | 0x06 | 选择 PCI 总线上的 TRIG6 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_TRIGGER7 | 0x07 | 选择 PCI 总线上的 TRIG7 输入/输出时钟 |
| PCI8502_OUTCLOCKSRC_PCISTAR  | 0x08 | PCI_START 信号触发            |

**bClockSourceDir** 是否将时钟信号输出到 PCI 总线，=TRUE：允许输出，=FALSE：允许输入

相关函数： [CreateDevice](#)      [LoadParaAD](#)      [SaveParaAD](#)  
[ReleaseDevice](#)

## 第二节、AD 状态参数结构（PCI8502\_STATUS\_AD）

### **Visual C++:**

```
typedef struct _PCI8502_STATUS_AD
{
    LONG bADEnable;
    LONG bTrigger;
    LONG bComplete;
    LONG bAheadTrig;
    LONG lEndAddr;
} PCI8502_STATUS_AD, *PCI8502_STATUS_AD;
```

### **Visual Basic:**

```
Private Type PCI8502_STATUS_AD
    bADEnable As Long
    bTrigger As Long
    bComplete As Long
    bAheadTrig As Long
    lEndAddr As Long
End Type
```

### **LabVIEW:**

请参考相关演示程序。

此结构体主要用于查询 AD 的各种状态，[GetDevStatusAD](#) 函数使用此结构体来实时取得 AD 状态，以便同步各种数据采集和处理过程。

**bADEnable** AD 是否已经使能，=TRUE：表示已使能，=FALSE：表示未使能。

**bTrigger** AD 是否被触发，=TRUE：表示已被触发，=FALSE：表示未被触发。

**bComplete** AD 是否整个转换过程是否结束，=TRUE：表示已结束，=FALSE：表示未结束。

**bAheadTrig** AD 触发点是否提前，=TRUE：表示触发点提前，=FALSE：表示触发点未提前。

**lEndAddr** 数据完成的结束地址。

相关函数： [CreateDevice](#)      [GetDevStatusAD](#)      [ReleaseDevice](#)

## 第五章 数据格式转换与排列规则

### 第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[] 中的第 1 个点 ADBuffer[0] 为例。

| 量程(mV)  | 计算机语言换算公式(ANSI C 语法)                                        | Volt 取值范围 (mV)    |
|---------|-------------------------------------------------------------|-------------------|
| ±5000mV | $Volt = (10000.00/4096) * (ADBuffer[0] \& 0xFFF) - 5000.00$ | [-5000, +4997.55] |
| ±1000mV | $Volt = (2000.00/4096) * (ADBuffer[0] \& 0xFFF) - 1000.00$  | [-1000, +999.51]  |

下面举例说明各种语言的换算过程（以±5000mV 量程为例）

**Visual C++:**

```
Lsb = ADBuffer[0] & 0xFFF;
Volt = (10000.00/4096) * Lsb - 5000.00;
```

**Visual Basic:**

```
Lsb = ADBuffer [0] And &HFFF
Volt = (10000.00/4096) * Lsb - 5000.00
```

**LabVIEW:**

请参考相关演示程序。

### 第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

|          |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |     |
|----------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|-----|
| 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  | ... |

### 第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++ 高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;           // 文件头信息长度
    LONG FileType;               // 该设备数据文件共有的成员
    LONG BusType;                // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;              // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;            // 头信息版本(D31-D16=Major,D15-D0=Minijor) = 1.0
    LONG VoltBottomRange;        // 量程下限(mV)
    LONG VoltTopRange;           // 量程上限(mV)
    LONG ChannelCount;           // 通道总数
    LONG DataWidth;              // 设备的精度(分辨率)
    LONG bXorHighBit;            // 是否高位求反(为 1 则求反)
    PCI8502_PARA_AD ADPara;      // 保存硬件参数
    PCI8502_STATUS_AD ADStatus;  // 保存硬件参数
    LONG CrystalFreq;            // 晶振频率
    LONG ChannelNum;             // 通道号
    LONG HeadEndFlag;            // 头信息结束位
} FILE_HEADER, *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

## 第六章 上层用户函数接口应用实例

### 第一节、简易程序演示说明

#### 一、怎么进行 AD 数采操作

**Visual C++:**

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

[程序] | [阿尔泰测控演示系统] | [PCI8502 同步采集 AD 卡] | [Microsoft Visual C++] | [简易代码演示]

### 第二节、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 PCI8502.h 和 ADDoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [PCI8502 同步采集 AD 卡] | [Microsoft Visual C++] | [高级代码演示]

其默认存放路径为: 系统盘\ART\PCI8502\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

## 第七章 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“PCI8502\_”)

| 函数名                                         | 函数功能                    | 备注        |
|---------------------------------------------|-------------------------|-----------|
| <b>① PCI 总线内存映射寄存器操作函数</b>                  |                         |           |
| <a href="#">GetDeviceBar</a>                | 取得指定的指定设备寄存器组 BAR 地址    | 底层用户      |
| <a href="#">GetDevVersion</a>               | 获取设备固件及程序版本             | 底层用户      |
| <a href="#">WriteRegisterByte</a>           | 以字节(8Bit)方式写寄存器端口       | 底层用户      |
| <a href="#">WriteRegisterWord</a>           | 以字(16Bit)方式写寄存器端口       | 底层用户      |
| <a href="#">WriteRegisterULong</a>          | 以双字(32Bit)方式写寄存器端口      | 底层用户      |
| <a href="#">ReadRegisterByte</a>            | 以字节(8Bit)方式读寄存器端口       | 底层用户      |
| <a href="#">ReadRegisterWord</a>            | 以字(16Bit)方式读寄存器端口       | 底层用户      |
| <a href="#">ReadRegisterULong</a>           | 以双字(32Bit)方式读寄存器端口      | 底层用户      |
| <b>② ISA 总线 I/O 端口操作函数</b>                  |                         |           |
| <a href="#">WritePortByte</a>               | 以字节(8Bit)方式写 I/O 端口     | 用户程序操作端口  |
| <a href="#">WritePortWord</a>               | 以字(16Bit)方式写 I/O 端口     | 用户程序操作端口  |
| <a href="#">WritePortULong</a>              | 以无符号双字(32Bit)方式写 I/O 端口 | 用户程序操作端口  |
| <a href="#">ReadPortByte</a>                | 以字节(8Bit)方式读 I/O 端口     | 用户程序操作端口  |
| <a href="#">ReadPortWord</a>                | 以字(16Bit)方式读 I/O 端口     | 用户程序操作端口  |
| <a href="#">ReadPortULong</a>               | 以无符号双字(32Bit)方式读 I/O 端口 | 用户程序操作端口  |
| <b>③ 创建 Visual Basic 子线程, 线程数量可达 32 个以上</b> |                         |           |
| <a href="#">CreateSystemEvent</a>           | 创建系统内核事件对象              | 用于线程同步或中断 |
| <a href="#">ReleaseSystemEvent</a>          | 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,  
PUCHAR pulPCIBar[6])

**Visual Basic:**

Declare Function GetDeviceBar Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByVal pulPCIBar (0 to 5) As Long) As Boolean

### 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。

pulPCIBar 返回 PCI BAR 所有地址。

**返回值:** 若成功，返回 TRUE，否则返回 FALSE。

**相关函数:** [CreateDevice](#) [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)
```

**Visual Basic:**

```
Declare Function GetDevVersion Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                ByRef pulFmwVersion As Long, _
                                                ByRef pulDriverVersion As Long) As Boolean
```

### LabVIEW:

请参见相关演示程序。

**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 指针参数，用于取得固件版本。

pulDriverVersion 指针参数，用于取得驱动版本。

**返回值:** 如果执行成功，则返回 TRUE，否则会返回 FALSE。

**相关函数:** [CreateDevice](#) [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:

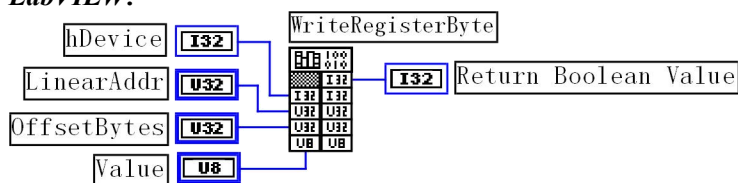
**Visual C++:**

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

**Visual Basic:**

```
Declare Function WriteRegisterByte Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                ByRef pbLinearAddr As String, _
                                                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](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定

[WriteRegisterByte](#) 函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

**返回值：**若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** [CreateDevice](#)      [WriteRegisterByte](#)      [WriteRegisterWord](#)  
[WriteRegisterULong](#)      [ReadRegisterByte](#)      [ReadRegisterWord](#)  
[ReadRegisterULong](#)      [ReleaseDevice](#)

#### Visual C++ 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20); // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice ); // 释放设备对象

```

#### Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
:

```

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

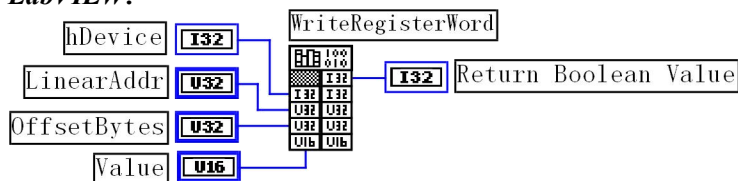
函数原型:

**Visual C++:**  
**BOOL** WriteRegisterWord(**HANDLE** hDevice,  
**PUCHAR** pbLinearAddr,  
**ULONG** OffsetBytes,  
**WORD** Value)

**Visual Basic:**

**Declare Function** WriteRegisterWord Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbLinearAddr As String, \_  
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](#) 创建。

pbLinearAddr PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定

[WriteRegisterWord](#) 函数所访问的映射寄存器的内存单元。

Value 输出 16 位整型值。

**返回值：**无。

**相关函数：** [CreateDevice](#)      [WriteRegisterByte](#)      [WriteRegisterWord](#)  
[WriteRegisterULong](#)      [ReadRegisterByte](#)      [ReadRegisterWord](#)

### Visual C++ 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000); // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice ); // 释放设备对象

```

### Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)

```

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

函数原型:

**Visual C++:**

```

BOOL WriteRegisterULong( HANDLE hDevice,
                        PCHAR pbLinearAddr,
                        ULONG OffsetBytes,
                        ULONG Value)

```

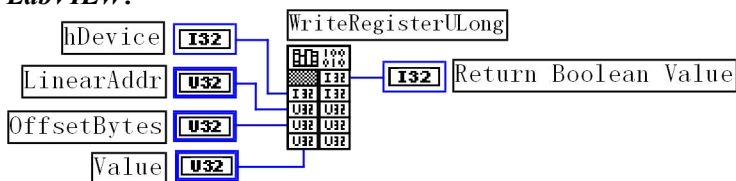
**Visual Basic:**

```

Declare Function WriteRegisterULong Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                    ByRef pbLinearAddr As String, _
                                                    ByVal OffsetBytes As Long, _
                                                    ByVal Value As Long) As Boolean

```

**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](#) 创建。

**pbLinearAddr** PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

**OffsetBytes** 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定

[WriteRegisterULong](#) 函数所访问的映射寄存器的内存单元。

**Value** 输出 32 位整型值。

**返回值:** 若成功，返回 TRUE，否则返回 FALSE。

**相关函数:** [CreateDevice](#)      [WriteRegisterByte](#)      [WriteRegisterWord](#)  
[WriteRegisterULong](#)      [ReadRegisterByte](#)      [ReadRegisterWord](#)  
[ReadRegisterULong](#)      [ReleaseDevice](#)

### Visual C++ 程序举例:

```

:
HANDLE hDevice;

```

```

ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败...”;
}
OffsetBytes=100;// 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000); // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice ); // 释放设备对象

```

### Visual Basic 程序举例:

```

:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
:

```

### ◆ 以单字节（即 8 位）方式读 PCI 内存映射寄存器的某个单元

函数原型:

**Visual C++:**

```

BYTE ReadRegisterByte( HANDLE hDevice,
                      PCHAR pbLinearAddr,
                      ULONG OffsetBytes)

```

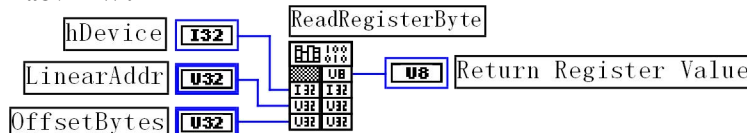
**Visual Basic:**

```

Declare Function ReadRegisterByte Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                ByRef pbLinearAddr As String, _
                                                ByVal OffsetBytes As Long) As Byte
                                                pbLinearAddr: Pointer;
                                                OffsetBytes : LongWord) : Byte;
StdCall; External 'PCI8502' Name ' ReadRegisterByte ';

```

**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](#) 创建。

**pbLinearAddr** PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

**OffsetBytes** 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定

[ReadRegisterByte](#) 函数所访问的映射寄存器的内存单元。

**返回值:** 返回从指定内存映射寄存器单元所读取的 8 位数据。

**相关函数:** [CreateDevice](#)      [WriteRegisterByte](#)      [WriteRegisterWord](#)  
[WriteRegisterULong](#)      [ReadRegisterByte](#)      [ReadRegisterWord](#)  
[ReadRegisterULong](#)      [ReleaseDevice](#)

### Visual C++ 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
BYTE Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 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)
GetDeviceAddr( hDevice, LinearAddr, 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,
                      PCHAR pbLinearAddr,
                      ULONG OffsetBytes)

```

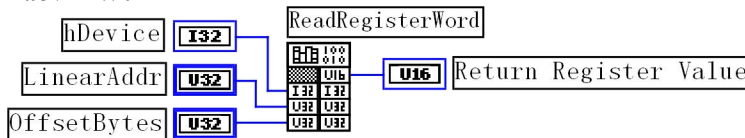
#### Visual Basic:

```

Declare Function ReadRegisterWord Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                ByRef pbLinearAddr As String, _
                                                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](#) 创建。

**pbLinearAddr** PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。

**OffsetBytes** 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定

[ReadRegisterWord](#) 函数所访问的映射寄存器的内存单元。

**返回值:** 返回从指定内存映射寄存器单元所读取的 16 位数据。

**相关函数:** [CreateDevice](#)      [WriteRegisterByte](#)      [WriteRegisterWord](#)  
[WriteRegisterULong](#)      [ReadRegisterByte](#)      [ReadRegisterWord](#)  
[ReadRegisterULong](#)      [ReleaseDevice](#)

### Visual C++ 程序举例:

```

:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 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)
GetDeviceAddr( hDevice, LinearAddr, 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

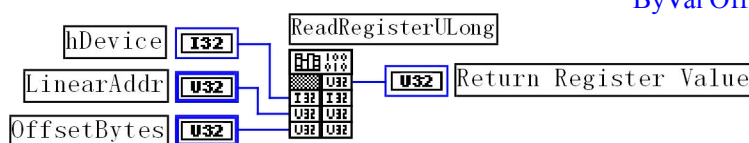
函数原型:

**Visual C++:**

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

**Visual Basic:**

```
Declare Function ReadRegisterULong Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                    ByRef pbLinearAddr As String, _
                                                    ByVal OffsetBytes As Long) As Long
```

**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](#) 创建。**pbLinearAddr** PCI 设备内存映射寄存器的线性基地址，它的值应由 [GetDeviceAddr](#) 确定。**OffsetBytes** 相对与 **LinearAddr** 线性基地址的偏移字节数，它与 **LinearAddr** 两个参数共同确定[WriteRegisterULong](#) 函数所访问的映射寄存器的内存单元。**返回值:** 返回从指定内存映射寄存器单元所读取的 32 位数据。**相关函数:**[CreateDevice](#)[WriteRegisterByte](#)[WriteRegisterWord](#)[WriteRegisterULong](#)[ReadRegisterByte](#)[ReadRegisterWord](#)[ReadRegisterULong](#)[ReleaseDevice](#)**Visual C++ 程序举例:**

:

```
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
ULONG Value;
hDevice = CreateDevice(0); // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0); // 取得 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)
GetDeviceAddr(hDevice, LinearAddr, 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 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 **WritePortByteEx** 或 **ReadPortByteEx** 等有“Ex”后缀的函数即可。

## ◆ 以单字节(8Bit)方式写 I/O 端口

**Visual C++:**

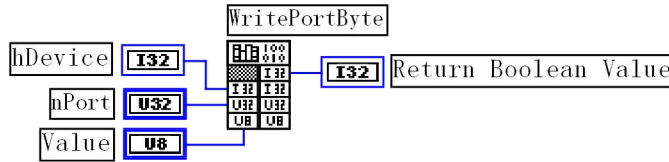
```
BOOL WritePortByte (HANDLE hDevice,
                    PCHAR pbPort,
```

ULONG OffsetBytes,  
BYTE Value)

#### Visual Basic:

Declare Function WritePortByte Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbPort As String, \_  
ByVal OffsetBytes As LongWord, \_  
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。

pbPort 指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。

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 GetLastErrorEx 捕获当前错误码。

**相关函数:** [CreateDevice](#)      [WritePortByte](#)      [WritePortWord](#)  
[WritePortULong](#)      [ReadPortByte](#)      [ReadPortWord](#)

### ◆ 以双字(16Bit)方式写 I/O 端口

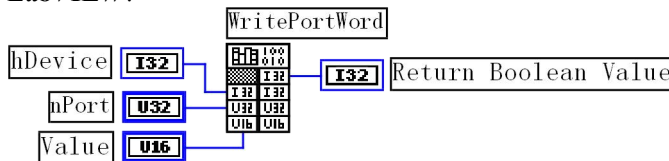
#### Visual C++:

BOOL WritePortWord (HANDLE hDevice,  
PUCHAR pbPort,  
ULONG OffsetBytes,  
WORD Value)

#### Visual Basic:

Declare Function WritePortWord Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbPort As String, \_  
ByVal OffsetBytes As LongWord, \_  
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。

pbPort 指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。

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 GetLastErrorEx 捕获当前错误码。

**相关函数:** [CreateDevice](#)      [WritePortByte](#)      [WritePortWord](#)  
[WritePortULong](#)      [ReadPortByte](#)      [ReadPortWord](#)

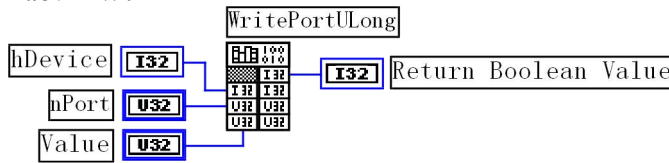
### ◆ 以四字节(32Bit)方式写 I/O 端口

#### Visual C++:

BOOL WritePortULong (HANDLE hDevice,  
PUCHAR pbPort,  
ULONG OffsetBytes,  
ULONG Value)

**Visual Basic:**

Declare Function WritePortULong Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbPort As String, \_  
ByVal OffsetBytes As LongWord, \_  
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。

**pbPort** 指定寄存器的物理基地址。

**OffsetBytes** 相对于物理基地址的偏移位置(字节)。

**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 GetLastErrorEx 捕获当前错误码。

**相关函数:** [CreateDevice](#)      [WritePortByte](#)      [WritePortWord](#)  
[WritePortULong](#)      [ReadPortByte](#)      [ReadPortWord](#)

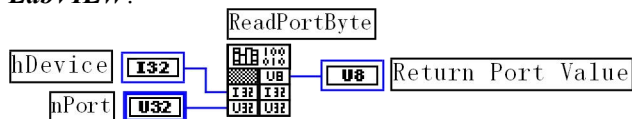
## ◆ 以单字节(8Bit)方式读 I/O 端口

**Visual C++:**

BYTE ReadPortByte( HANDLE hDevice,  
PUCHAR pbPort,  
ULONG OffsetBytes)

**Visual Basic:**

Declare Function ReadPortByte Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbPort As String, \_  
ByVal OffsetBytes As LongWord) 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。

**pbPort** 指定寄存器的物理基地址。

**OffsetBytes** 相对于物理基地址的偏移位置(字节)。

**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

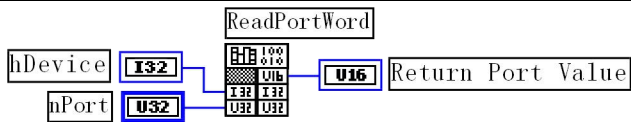
**Visual C++:**

WORD ReadPortWord(HANDLE hDevice,  
PUCHAR pbPort,  
ULONG OffsetBytes)

**Visual Basic:**

Declare Function ReadPortWord Lib "PCI8502\_32" (ByVal hDevice As Long, \_  
ByRef pbPort As String, \_  
ByVal OffsetBytes As LongWord) 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。

**pbPort** 指定寄存器的物理基地址。

**OffsetBytes** 相对于物理基地址的偏移位置(字节)。

**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

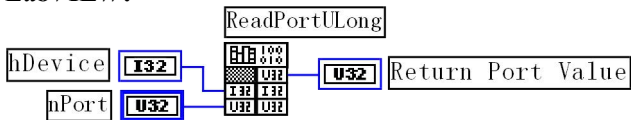
**Visual C++:**

```
ULONG ReadPortULong(HANDLE hDevice,
                    PCHAR pbPort,
                    ULONG OffsetBytes)
```

**Visual Basic:**

```
Declare Function ReadPortULong Lib "PCI8502_32" (ByVal hDevice As Long, _
                                                ByRef pbPort As String, _
                                                ByVal OffsetBytes As LongWord) 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。

**pbPort** 指定寄存器的物理基地址。

**OffsetBytes** 相对于物理基地址的偏移位置(字节)。

**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

函数原型:

**Visual C++:**

```
HANDLE CreateSystemEvent(void)
```

**Visual Basic:**

```
Declare Function CreateSystemEvent Lib "PCI8502" () 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

函数原型：

**Visual C++**

BOOL ReleaseSystemEvent(HANDLE hEvent)

**Visual Basic:**

Declare Function ReleaseSystemEvent Lib "PCI8502 " (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。