ART2010 数据采集卡

WIN2000/XP 驱动程序使用说明书

北京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

提醒用户：

通常情况下，WINDOWS 系统在安装时自带的 DLL 库和驱动不全，所以您不管使用那种语言编程，请您最好先安装上 Visual C++6.0 版本的软件，方可使我们的驱动程序有更完备的运行环境。

有关设备驱动安装和产品二次发行请参考 ATR2010Inst.doc 文档。

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市阿尔泰科贸有限公司所有，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 PC104xxxx_ 则被省略。如 ATR2010_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、如何管理 PC104 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 CreateDevice 函数创建一个设备对象句柄 hDevice，有了这个句柄，您就拥有了对该设备的控制权。然后将此句柄作为参数传递给其他函数，如 InitDeviceProAD 可以使用 hDevice 句柄以初始化设备的 AD 部件并启动 AD 设备，ReadDeviceProAD 函数可以用 hDevice 句柄实现对 AD 数据的采样批量读取，SetDeviceDO 函数可用实现开关量的输出等。最后可以通过 ReleaseDevice 将 hDevice 释放掉。

第二节、如何批量取得 AD 数据

当您有了 hDevice 设备对象句柄后，便可用 InitDeviceProAD 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，然后这个函数启动 AD 设备。接着便可用 ReadDeviceProAD 反复读取 AD 数据以实现连续不间断采样当您需要关闭 AD 设备时，ReleaseDeviceAD 便可帮您实现（但设备对象 hDevice 依然存在）。（注：ReadDeviceProAD 虽然主要面对批量读取，高速连续采集而设计，但亦可用它以少量点如 32 个点读取 AD 数据，以满足慢速采集需要）。具体执行流程请看下面的图 2.1.1。

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。绿色虚线 InitDeviceProAD 与 ReleaseDeviceAD 成对称方式出现。

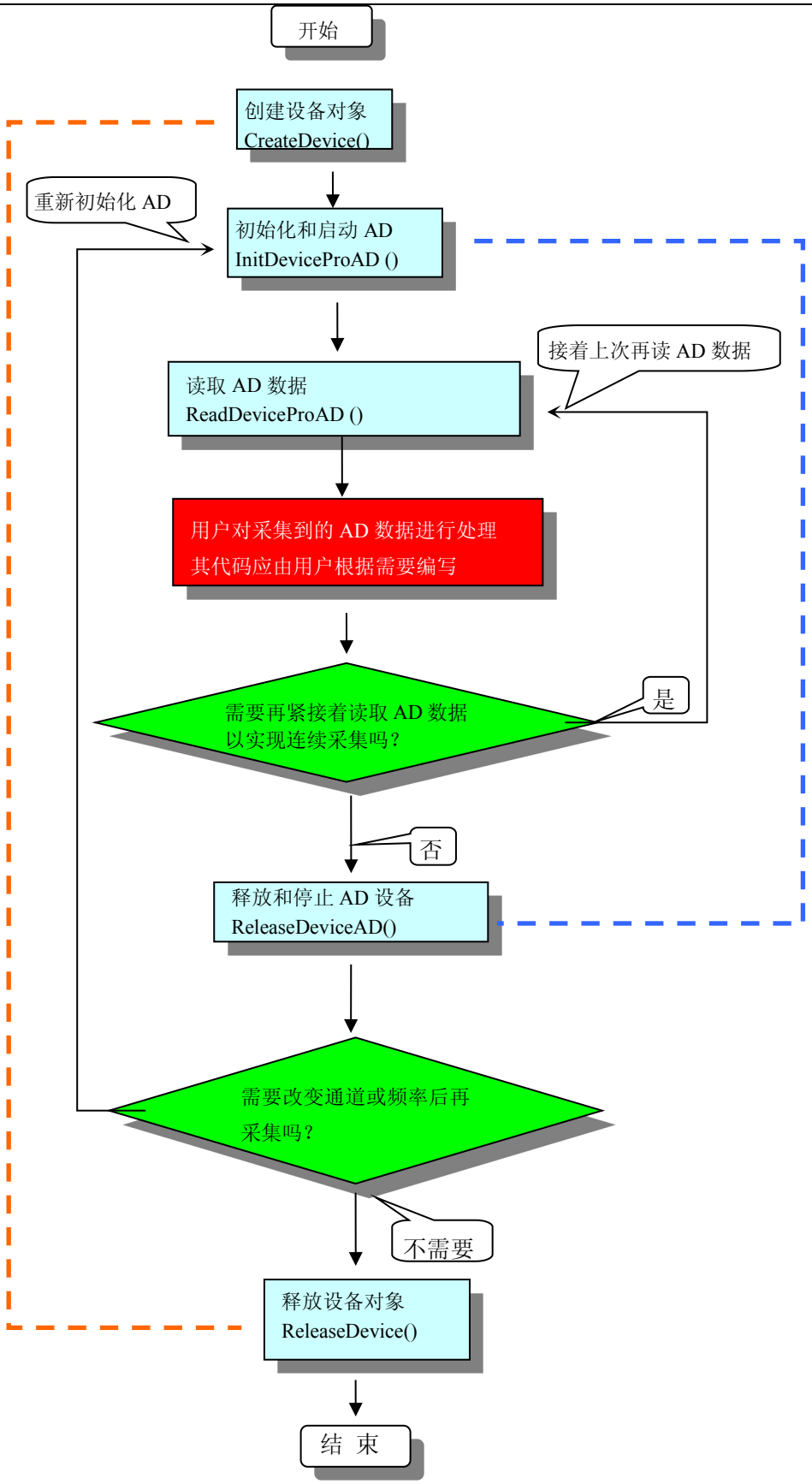


图 2.1.1 AD 采集实现过程

第三节、如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 SetDeviceDO 函数实现开关量的输出操作，其各路开关量的输

出状态由其 PParaDO 中的相应元素决定。由 GetDeviceDI 函数实现开关量的输入操作，其各路开关量的输入状态由其 PParaDI 中的相应元素决定。

第四节、哪些函数对您不是必须的

当公共函数如 CreateFileObject，WriteFile，ReadFile 等一般来说都是辅助性函数，除非您要使用存盘功能。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。

第三章　PC104 设备专用函数接口介绍

第一节、设备驱动接口函数列表（每个函数省略了前缀“ATR2010_”）

本章函数是设备使用 PC104 方式传输时所使用的。

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PC104 对象(用设备逻辑号)	
GetDeviceCurrentBaseAddr	取得当前设备基地址	
ReleaseDevice	关闭设备，且释放 PC104 总线设备对象	
② AD 采样操作函数		
InitDeviceProAD	初始化 PC104 设备 AD 部件，准备传数	
ReadDeviceProAD	连续批量读取 PC104 设备上的 AD 数据	
③ 辅助函数（硬件参数设置、保存、读取函数）		
LoadParaAD	从 Windows 系统中读取硬件参数	
SaveParaAD	往 Windows 系统保存硬件参数	
④计数器操作函数		
InitDevCounter	初始化各路计数器	
GetDeviceCNT	取得所有计数器的状态信息	
⑤ 开关量函数		
GetDeviceDI	开关输入函数	
SetDeviceDO	开关输出函数	

使用需知

Visual C++ & C++Builder：

首先将 ATR2010.h 和 ATR2010.lib 两个驱动库文件从相应的演示程序文件夹下复制到您的源程序文件夹中，然后在您的源程序头部添加如下语句，以便将驱动库函数接口的原型定义信息和驱动接口导入库(ATR2010.lib)加入到您的工程中。

#include “ATR2010.H”

在 VC 中，为了使用方便，避免重复定义和包含，您最好将以上语句放在 StdAfx.h 文件。一旦完成了以上工作，那么使用设备的驱动程序接口就跟使用 VC/C++Builder 自身的各种函数，其方法一样简单，毫无二别。

关于 ATR2010.h 和 ATR2010.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 ATR2010.Bas 驱动模块头文件从 VB 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 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 ATR2010.Bas 模块文件即可，一旦完成以上工作后，那么

使用设备的驱动程序接口就跟使用 VB 自身的各种函数，其方法一样简单，毫无二别。

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，我们不保证能完全顺利运行。

Delphi:

首先将 ATR2010.Pas 驱动模块头文件从 Delphi 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 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 ATR2010.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。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：“ATR2010”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ATR2010;  // 注意： 在此加入驱动程序接口单元 ATR2010
```

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 的驱动程序接口的详细说明请参考其演示源程序。

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++ & C++ Builder：

```
HANDLE CreateDevice(WORD BaseAddress)
```

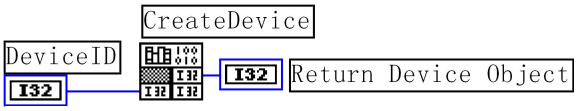
Visual Basic：

```
Declare Function CreateDevice Lib "ATR2010" (ByVal BaseAddress As Integer) As Long
```

Delphi：

```
Function CreateDevice(BaseAddress : Word):Integer;
                    StdCall; External 'ATR2010' Name 'CreateDevice';
```

LabView:



功能：该函数负责创建设备对象，并返回其设备对象句柄。

参数：

BaseAddress 设备基地址。板基地址可设置成 200H～3F0H 之间可被 16 整除的二进制码，板基地址默认为 300H，将占用基地址起的连续 32 个 I/O 地址。具体设置请参考硬件说明书。

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

◆ **取得当前设备基地址**

函数原型：

Visual C++ & C++Builder:

WORD GetDeviceCurrentBaseAddr (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCurrentBaseAddr Lib "ATR2010" (ByVal hDevice As Long) As Integer

Delphi:

Function GetDeviceCurrentBaseAddr (hDevice : Integer): Word;
StdCall; External 'ATR2010' Name ' GetDeviceCurrentBaseAddr ';

LabView:

请参考相关演示程序。

功能：取得当前设备基地址。

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

返回值：若成功，则返回实际设备台数，否则返回 0，用户可以用 GetLastError 捕获错误码。

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++ & C++Builder:

BOOL ReleaseDevice(HANDLE hDevice)

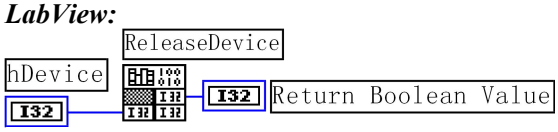
Visual Basic:

Declare Function ReleaseDevice Lib "ATR2010" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDevice(hDevice : Integer):Boolean;
StdCall; External 'ATR2010' 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，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 采样操作函数原型说明

◆ **初始化设备对象**

函数原型：

Visual C++ & C++Builder：

BOOL InitDeviceProAD (HANDLE hDevice,

PATR2010_PARA_AD pADPara)

Visual Basic:

```
Declare Function InitDeviceProAD Lib "ATR2010" (ByVal hDevice As Long, _
                                                 ByRef pADPara As ATR2010_PARA _AD) As Boolean
```

Delphi:

```
Function InitDeviceProAD ( hDevice : Integer;
                           pADPara : PATR2010_PARA_AD):Boolean;
                           StdCall; External 'ATR2010' Name ' InitDeviceProAD ';
```

LabView:

请参考相关演示程序。

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，如预置 AD 采集通道，采样频率等，然后启动 AD 设备开始 AD 采集，随后，用户便可以连续调用 ReadDeviceProAD 读取 PC104 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 PC104 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，如 AD 采样通道、采样频率等。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 GetLastError 捕获当前错误码，并加以分析。

相关函数： CreateDevice ReadDeviceProAD ReleaseDevice

♦ 批量读取 PC104 设备上的 AD 数据

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceProAD (HANDLE hDevice,
                      PLONG pADBuffer,
                      LONG nReadSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD Lib "ATR2010" (ByVal hDevice As Long, _
                                                ByRef pADBuffer As Long, _
                                                ByVal nReadSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD ( hDevice : Integer;
                           pADBuffer : Pointer;
                           nReadSizeBytes : LongInt) : Boolean;
                           StdCall; External 'ATR2010' Name ' ReadDeviceProAD ';
```

LabView:

请参考相关演示程序。

功能： 读取 PC104 设备 AD 部件上的批量数据。它不负责初始化 AD 部件，待读完整过指定长度的数据才返回。它必须在 InitDeviceProAD 之后，ReleaseDeviceAD 之前调用。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pADBuffer 指向接受数据的用户缓冲区。接受的是从设备上采集的 LSB 原码数据，关于如何将 LSB 原码数据转换成电压值，请参考《数据格式转换与排列规则》章节。

nReadSizeWords 相对于偏位点后读入的数据长度(字)。

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 InitDeviceProAD ReleaseDevice

❖ **以上函数调用一般顺序**

① CreateDevice
② InitDeviceProAD
③ ReadDeviceProAD
④ ReleaseDevice

用户可以反复执行第③步，以实现高速连续不间断数据采集。

第四节、AD 硬件参数系统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++ & C++Builder:

```
BOOL LoadParaAD(HANDLE hDevice,
                PATR2010_PARA_AD pADPara)
```

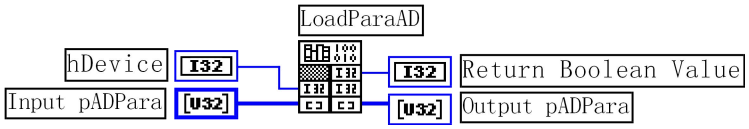
Visual Basic:

```
Declare Function LoadParaAD Lib "ATR2010" (ByVal hDevice As Long, _
                                           ByRef pADPara As ATR2010_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD( hDevice : Integer;
                     pADPara : PATR2010_PARA_AD):Boolean;
                     StdCall; External 'ATR2010' Name 'LoadParaAD';
```

LabView:



功能：负责从 Windows 系统中读取设备硬件参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pADPara 属于 PATR2010_PARA 的结构指针型，它负责返回 PC104 硬件参数值，关于结构指针类型 PATR2010_PARA 请参考相应 ATR2010.h 或该结构的帮助文档的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SaveParaAD ReleaseDevice

Visual C++ & C++Builder 举例：

```
        :
ATR2010_PARA_AD ADPara;
HANDLE   hDevice;
HDevice = CreateDevice(0) ;      // 管理秕一个 PC104 设备
if(!LoadParaAD(hDevice, &ADPara))
{
      AfxMessageBox("读入硬件参数失败，请确认您的驱动程序是否正确安装");
      Return;    // 若错误，则退出该过程
}
        :
```

Visual Basic 举例：

```
        :
```

```
Dim ADPara As ATR2010_PARA_AD
Dim hDevice As Long   :
hDevice = CreateDevice(0)    ‘ 管理第一个 PC104 设备
If Not LoadParaAD(hDevice, ADPara) Then
        MsgBox “读入硬件参数失败，请确认您的驱动程序是否正确安装”
        Exit Sub      ' 若错误，则退出该过程
End If
            :
```

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++ & C++Builder:

```
BOOL SaveParaAD(HANDLE hDevice,
                PATR2010_PARA_AD pADPara)
```

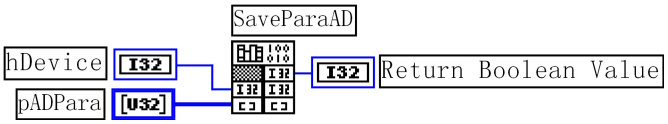
Visual Basic:

```
Declare Function SaveParaAD Lib "ATR2010" (ByVal hDevice As Long, _
                                           ByRef pADPara As ATR2010_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
                     pADPara : PATR2010_PARA_AD):Boolean;
                     StdCall; External 'ATR2010' Name 'SaveParaAD';
```

LabView:



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 AD 设备硬件参数，请参考《硬件参数结构》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　　LoadParaAD　　ReleaseDevice

第五节、计数器操作函数原型说明

♦ **初始化各路计数器**

函数原型：

Visual C++ & C++Builder:

```
BOOL InitDevCounter (HANDLE hDevice,
                     PART2010_PARA_COUNTER_CTRL pCtrlPara,
                     int InitCntrVal,
                     int nCntrChannel)
```

Visual Basic:

```
Declare Function InitDevCounter Lib "ATR2010" (ByVal hDevice As Long, _
                                               ByVal pCtrlPara As ART2010_PARA_COUNTER_CTRL,_
                                               ByVal InitCntrVal As Integer,_
                                               ByVal nCntrChannel As Integer) As Boolean
```

Delphi:

```
Function InitDevCounter (hDevice : Integer;
                         pCtrlPara : PART2010_PARA_COUNTER_CTRL;
```

```
                    InitCntrVal : Integer;
                    nCntrChannel : Integer) : Boolean;
                    StdCall; External 'ATR2010' Name ' InitDevCounter ';
```

LabVIEW:

请参考相关演示程序。

功能：初始化各路计数器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pCtrlPara 计数器控制字。

InitCntrVal 16 位计数初值。

nCntrChannel 计数器通道选择，取值为[0，2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceCNT ReleaseDevice

- **取得所有计数器的状态信息**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceCNT (HANDLE hDevice,
                   LONG *pCntrValue,
                   int nCntrChannel)
```

Visual Basic:

```
Declare Function GetDeviceCNT Lib "ATR2010" (ByVal hDevice As Long, _
                                             ByRef pCntrValue As Long, _
                                             ByVal nCntrChannel As Integer) As Boolean
```

Delphi:

```
Function GetDeviceCNT ( hDevice : Integer;
                        pCntrValue: Pointer;
                        nCntrChannel : Integer) : Boolean;
                        StdCall; External 'ATR2010' Name ' GetDeviceCNT ';
```

LabVIEW:

请参考相关演示程序。

功能：取得所有计数器的状态信息。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pCntrValue 所有计数器的状态信息。

nCntrChannel 计数器通道选择，取值为[0，2]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice InitDevCounter ReleaseDevice

第六节、DIO 数字开关量输入输出简易操作函数原型说明

- **十二路开关量输入**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceDI (HANDLE hDevice,
```

PART2010_PARA_DI PParaDI)

Visual Basic:

Declare Function GetDeviceDI Lib "ATR2010" (ByVal hDevice As Long, _
ByVal PParaDI As ART2010_PARA_DI) As Boolean

Delphi:

Function GetDeviceDI (hDevice : Integer;
PParaDI : PART2010_PARA_DI):Boolean;
StdCall; External 'ATR2010' Name ' GetDeviceDI ';

LabView：

功能：负责将 PC104 设备上的输入开关量状态读入内存。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

PParaDI 十二路开关量输入状态的参数结构，共有 12 个成员变量，分别对应于 DI0-DI11 路开关量输入状态位。如果 PParaDI 为“1”则使通道处于开状态，若为“0”则通道为关状态。其他同理。具体定义请参考《DI 数字开关量输入参数介绍》章节。

返回值：若成功，返回 TRUE，其 PParaDI 中的值有效；否则返回 FALSE，其 PParaDI 中的值无效。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

♦ **十二路开关量输出**

函数原型：

Visual C++ & C++Builder:

BOOL SetDeviceDO (HANDLE hDevice,
PART2010_PARA_DO PParaDO)

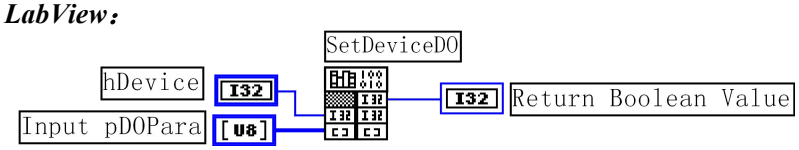
Visual Basic:

Declare Function SetDeviceDO Lib "ATR2010" (ByVal hDevice As Long, _
ByVal PParaDO As ART2010_PARA_DO) As Boolean

Delphi:

Function SetDeviceDO (hDevice : Integer;
PparaDO : PART2010_PARA_DO):Boolean;
StdCall; External 'ATR2010' Name ' SetDeviceDO ';

LabView：



功能：负责将 PC104 设备上的输出开关量置成相应的状态。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

PParaDO 十二路开关量输出状态的参数结构，共有 12 个成员变量，分别对应于 DO0-DO11 路开关量输出状态位。比如置 PParaDO 为“1”则使通道处于“开”状态，若为“0”则置通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO11 共 12 个成员变量赋初值，其值必须为“1”或“0”。具体定义请参考《DO 数字开关量输出参数介绍》。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

❖ **以上函数调用一般顺序**

① CreateDevice

② SetDeviceDO（或 GetDeviceDI，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 AD 采样可以同时进行，互不影响）。

第四章　硬件参数结构

第一节、AD 硬件参数结构（ATR2010_PARA_AD）

Visual C++ & C++Builder:

```
typedef struct _ATR2010_PARA_AD        // 板卡各参数值
{
    LONG FirstChannel;                 // 首通道,取值范围为[0, 31]
    LONG LastChannel;                  // 末通道,取值范围为[0, 31]
} ATR2010_PARA_AD, *PATR2010_PARA_AD;
```

Visual Basic：

```
Private Type ATR2010_PARA_AD
    FirstChannel As Long               ' 首通道,取值范围为[0, 31]
    LastChannel As Long                ' 末通道,取值范围为[0, 31]
End Type
```

Delphi:

```
  Type   // 定义结构体数据类型
    PATR2010_PARA_AD = ^ATR2010_PARA_AD;   // 指针类型结构
    ATR2010_PARA_AD = record           // 标记为记录型
        FirstChannel : LongInt;                // 首通道,取值范围为[0, 31]
        LastChannel : LongInt;                 // 末通道,取值范围为[0, 31]
    End;
```

LabView:

首先请您关注一下这个结构与前面 ISA 总线部分中的硬件参数结构 PARA 比较，该结构实在太简短了。其原因就是在于 PC104 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 PC104 设备的用户永远告别，一句话 PC104 设备简单得就象使用电源插头一样。

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硬件配置完全由 InitDeviceProAD 函数完成。

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

LastChannel 末通道值，取值范围应根据设备的总通道数设定，本设备的 AD 采样首通道取值范围为 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。

相关函数： InitDeviceProAD LoadParaAD SaveParaAD

第二节、计数器参数结构（ART2010_PARA_COUNTER_CTRL）

Visual C++ & C++Builder:

```
typedef struct _ART2010_PARA_COUNTER_CTRL
{
    BYTE OperateType;     // 操作类型
    BYTE CountMode;       // 计数方式
    BYTE BCD;             // 计数类型
} ART2010_PARA_COUNTER_CTRL, *PART2010_PARA_COUNTER_CTRL;
```

Visual Basic：

```
Type ART2010_PARA_COUNTER_CTRL
    OperateType As Byte
    CountMode As Byte
    BCD As Byte
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PPCI8602_ PARA_CNT = ^ PCI8602_ PARA_CNT;    // 指针类型结构
    PCI8602_ PARA_CNT = record        // 标记为记录型
    OperateType : Byte;
    CountMode : Byte;
    BCD : Byte;
End;
```

LabVIEW:

请参考相关演示程序。

OperateType 计数器操作类型选择，它的取值如下表：

常量名	常量值	功能定义
ART2010_OperateType_0	0x00	计数器锁存操作
ART2010_OperateType_1	0x01	只读/写低字节
ART2010_OperateType_2	0x02	只读/写高字节
ART2010_OperateType_3	0x03	先读/写低字节，后读/写高字节

CountMode 计数方式选择，它的取值如下表：

常量名	常量值	功能定义
ART2010_CountMode_0	0x00	计数方式 0，计数器结束中断方式
ART2010_CountMode_1	0x01	计数方式 1，可编程单次脉冲方式
ART2010_CountMode_2	0x02	计数方式 2，频率发生器方式
ART2010_CountMode_3	0x03	计数方式 3，方波频率发生器方式
ART2010_CountMode_4	0x04	计数方式 4，软件触发选通方式

ART2010_CountMode_5	0x05	计数方式 5，硬件触发选通方式

BCD 计数器计数类型选择，它的取值如下表：

常量名	常量值	功能定义
ART2010_BCD_0	0x00	计数类型 0，二进制计数
ART2010_BCD_1	0x01	计数类型 1，BCD 码计数

相关函数： CreateDevice GetDeviceCNT ReleaseDevice

第三节、计数器值结构（ART2010_PARA_COUNTER_VAL）

Visual C++ & C++Builder:

```
typedef struct _ART2010_PARA_COUNTER_ VAL
{
    LONG CounterValue0; //  计数器 0 的计数值
    LONG CounterValue1; //  计数器 1 的计数值
    LONG CounterValue2; //  计数器 2 的计数值
} ART2010_PARA_COUNTER_ VAL, *PART2010_PARA_COUNTER_ VAL;
```

Visual Basic：

```
Type ART2010_PARA_COUNTER_ VAL
    CounterValue0 As Long
    CounterValue1 As Long
    CounterValue2 As Long
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PPCI8602_ PARA_ VAL= ^ PCI8602_ PARA_ VAL;     // 指针类型结构
    PCI8602_ PARA_ VAL= record        // 标记为记录型
    CounterValue0 : LongInt;
    CounterValue1 : LongInt;
    CounterValue2 : LongInt;
End;
```

LabVIEW:

请参考相关演示程序。

CounterValue0 计数器 0 的计数值。
CounterValue1 计数器 1 的计数值。
CounterValue2 计数器 2 的计数值。

相关函数： CreateDevice GetDeviceCNT ReleaseDevice

第四节、数字量输入参数（ATR2010_PARA_DI）

Visual C++ & C++Builder:

```
typedef struct _ATR2010_PARA_DI        // 数字量输入参数
```

```
{
    BYTE DI0;          // 0 通道
    BYTE DI1;          // 1 通道
    BYTE DI2;          // 2 通道
    BYTE DI3;          // 3 通道
    BYTE DI4;          // 4 通道
    BYTE DI5;          // 5 通道
    BYTE DI6;          // 6 通道
    BYTE DI7;          // 7 通道
    BYTE DI8;          // 8 通道
    BYTE DI9;          // 9 通道
    BYTE DI10;         // 10 通道
    BYTE DI11;         // 11 通道
} ATR2010_PARA_DI,*PATR2010_PARA_DI;
```

Visual Basic：

```
Type ATR2010_PARA_DI
    DI0 As Byte   ' 0 通道
    DI1 As Byte   ' 1 通道
    DI2 As Byte   ' 2 通道
    DI3 As Byte   ' 3 通道
    DI4 As Byte   ' 4 通道
    DI5 As Byte   ' 5 通道
    DI6 As Byte   ' 6 通道
    DI7 As Byte   ' 7 通道
    DI8 As Byte   ' 8 通道
    DI9 As Byte   ' 9 通道
    DI10 As Byte   '10 通道
    DI11 As Byte   ' 11 通道
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
  PATR2010_PARA_DI = ^ATR2010_PARA_DI;   // 指针类型结构
  ATR2010_PARA_DI = record          // 标记为记录型
    DI0: Byte;     // 0 通道
    DI1: Byte;     // 1 通道
    DI2: Byte;     // 2 通道
    DI3: Byte;     // 3 通道
    DI4: Byte;     // 4 通道
    DI5: Byte;     // 5 通道
    DI6: Byte;     // 6 通道
    DI7: Byte;     // 7 通道
    DI8: Byte;     // 8 通道
    DI9: Byte;     // 9 通道
    DI10: Byte;     // 10 通道
```

```
    DI11: Byte;     // 11 通道
End;
```

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，只需要执行 GetDeviceDI 即可完成数字量输入操作。然后象 Visual Basic 中的属性操作那样，简单的进行属性成员分析即可确定各路状态。

关于 LabView 的参数，由于需要的是返回值，因此根据 LabView 的特点，应分配一个 16 字节的内存单元，每一个字节的内存单元对应相应位置上的开关量输入状态。要使用这些状态，则应在 GetDeviceDI 之后，将存放实际的当前开关量状态的内存单元用 Index Array 数组操作控件将其每一路开关量状态分离出来，即可确定每一路开关输入状态。详见开关量输入输出 LabView 演示部分。

其每一个成员变量对应于相应的 DI 通道，即 DI0～DI11 分别对应于 DI 通道 0～11。且这些成员变量只能是“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

第五节、数字量输出参数（ATR2010_PARA_DO）

Visual C++ & C++Builder:

```
typedef struct _ATR2010_PARA_DO         // 数字量输出参数
{
    BYTE DO0;          // 0 通道
    BYTE DO1;          // 1 通道
    BYTE DO2;          // 2 通道
    BYTE DO3;          // 3 通道
    BYTE DO4;          // 4 通道
    BYTE DO5;          // 5 通道
    BYTE DO6;          // 6 通道
    BYTE DO7;          // 7 通道
    BYTE DO8;          // 8 通道
    BYTE DO9;          // 9 通道
    BYTE DO10;          // 10 通道
    BYTE DO11;          // 11 通道
} ATR2010_PARA_DO,*PATR2010_PARA_DO;
```

Visual Basic：

```
Type ATR2010_PARA_DO
    DO0 As Byte   ' 0 通道
    DO1 As Byte   ' 1 通道
    DO2 As Byte   ' 2 通道
    DO3 As Byte   ' 3 通道
    DO4 As Byte   ' 4 通道
    DO5 As Byte   ' 5 通道
    DO6 As Byte   ' 6 通道
    DO7 As Byte   ' 7 通道
    DO8 As Byte   ' 8 通道
    DO9 As Byte   ' 9 通道
    DO10 As Byte   '10 通道
    DO11 As Byte   ' 11 通道
```

```
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PATR2010_PARA_DO = ^ATR2010_PARA_DO;   // 指针类型结构
    ATR2010_PARA_DO = record          // 标记为记录型
        DO0: Byte;     // 0 通道
        DO1: Byte;     // 1 通道
        DO2: Byte;     // 2 通道
        DO3: Byte;     // 3 通道
        DO4: Byte;     // 4 通道
        DO5: Byte;     // 5 通道
        DO6: Byte;     // 6 通道
        DO7: Byte;     // 7 通道
        DO8: Byte;     // 8 通道
        DO9: Byte;     // 9 通道
        DO10: Byte;     // 10 通道
        DO11: Byte;     // 11 通道
End;
```

LabView:

请参考相关演示程序。

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，只需要有简单的进行属性赋值，然后执行 SetDeviceDO 即可完成数字量输出。注意关于 LabView 的参数定义，他最主要表达了在 LabView 环境中怎样使用 SetDeviceDO 实现开关量输出操作的基本实现方法。在用户实际使用中，您可以将左边的常量图标换成开关控件图标等，以实现动态改变开关量输出状态。但需要注意的是开关控件图标（xxx Switch）输出的值是布尔变量，因此在开关控件图标与 PATR2010_PARA_DO 之间，应使用 Boolean To (0,1)逻辑转换控件，即先将布尔变量转换成 0 或 1 的整型值，再将这个整型值传递给 PATR2010_PARA_DO，详见开关量输入输出 LabView 演示部分。

其每一个成员变量对应于相应的 DO 通道，即 DO0～DO11 分别对应于 DO 通道 0～11。且这些成员变量只能被赋值为“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

相关函数： CreateDevice　　SetDeviceDO　　ReleaseDevice

第五章　数据格式转换与排列规则

第一节、AD 原始数据 LSB 转换成电压值 Volt 的换算方法

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 12 位的模板 CountLSB 为 4096。其他类型同理均按 2^n=LSB 总数（n 为 Bit 位数）换算即可。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000	Volt = (20000.00/4096) * (ADBuffer[0] &0x0FFF) – 10000.00	[-10000.00，　+ 9995.11]
±5000	Volt = (10000.00/4096) * (ADBuffer[0] &0x0FFF) – 5000.00	[-5000.00，　+ 4997.55]
0～10000	Volt = (10000.00/4096) * (ADBuffer[0] &0x0FFF)	[0.00，　+ 9997.55]

换算举例：（设量程为±10000mV，这里只转换第一个点）

Visual C++&C++Builder：

```
USHORT 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
float PerLsbVolt = 20000.00/4096; // 求出每个 LSB 原码单位电压值
Lsb = ADBuffer[0] &0x0FFF;
Volt = PerLsbVolt * Lsb – 10000.00;   // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

Visual Basic:

```
Dim Lsb As Long   ' 定义存放标准 LSB 原码的变量
Dim Volt As Single   ' 定义存放转换后的电压值的变量
Dim PerLsbVolt As Single
PerLsbVolt = 20000.00/4096   ' 求出每个 LSB 原码单位电压值
Lsb = ADBuffer(0) AND &H0FFF    ' 将其转换成无符号 13 位有效数据
Volt = PerLsbVolt * Lsb-10000.00 ' 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

Delphi:

```
Lsb : Word;   // 定义存放标准 LSB 原码的变量
Volt ： Single;   // 定义存放转换后的电压值的变量
PerLsbVolt : Single;
PerLsbVolt := 20000.00/4096;   // 求出每个 LSB 原码单位电压值
Lsb := ADBuffer[0] &$0FFF;
Volt ：= PerLsbVolt * Lsb-10000.00; // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

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 – CH1)

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，那么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（在 PC104 设备上同时也应 32 的整数倍），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 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	

第六章　上层用户函数接口应用实例

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

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

第一节、简易程序演示说明

一、怎样使用 ReadDeviceProAD 函数进行 AD 连续数据采集

其详细应用实例及工程级代码请参考 Visual C++简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 ATR2010.h 和 Sys.cpp)。

[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 采集演示源程序]

其简易程序默认存放路径为：系统盘\ART\ATR2010\SAMPLES\VC\SIMPLE\AD

二、怎样使用 SetDeviceDO 和 GetDeviceDI 函数进行开关量输入输出操作

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序，您先点击 Windows 系统的[开始]菜

单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 ATR2010.h 和 Sys.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨[简易代码演示]丨[开关量演示源程序]

其默认存放路径为：系统盘\ART\ATR2010\SAMPLES\VC\SIMPLE\DIO

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 ATR2010.h 和 DADoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨[高级代码演示]

其默认存放路径为：系统盘\ART\ATR2010\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　公共接口函数介绍

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“ATR2010_”）

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
ReleaseSystemEvent	释放系统内核事件对象	
DelayTimeUs	微秒级延时函数	
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
GetDevVersion	获取设备固件及程序版本	
③ 文件对象操作函数		
CreateFileObject	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	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	
ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备

第二节、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）

♦ **在 VB 环境中，创建子线程对象，以实现多线程操作**

Visual C++ & C++ Builder:

```
BOOL CreateVBThread(HANDLE *hThread,
                    LPTHREAD_START_ROUTINE StartThread)
```

Visual Basic：

```
Declare Function CreateVBThread Lib "ATR2010" (ByRef hThread As Long, _
                                               ByVal StartThread 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。

StartThread 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 Win32 API 函数 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 ATR2010.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          :
     :
```

♦ **在 VB 中，删除子线程对象**

Visual C++ & C++ Builder:

```
BOOL TerminateVBThread(HANDLE hThreadHandle)
```

Visual Basic:

```
Declare Function TerminateVBThread Lib "ATR2010" (ByVal hThreadHandle As Long) As Boolean
```

功能：在 VB 中删除由 CreateVBThread 创建的子线程对象。

参数：hThreadHandle 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
    :
```

◆ **创建内核系统事件**

Visual C++ & C++ Builder:

HANDLE CreateSystemEvent(void)

Visual Basic:

Declare Function CreateSystemEvent Lib " ATR2010 " ( ) As Long

Delphi:

Function CreateSystemEvent( ) : Integer;
StdCall; External 'ATR2010' 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)。

◆ **释放内核系统事件**

Visual C++ & C++ Builder:

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function ReleaseSystemEvent Lib " ATR2010 " (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
StdCall; External 'ATR2010' Name ' ReleaseSystemEvent ';

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。

◆ **高效高精度延时**

函数原型：

Visual C++ & C++ Builder:

```
BOOL DelayTimeUs (HANDLE hDevice,
                  LONG nTimeUs)
```

Visual Basic:

```
Declare Function DelayTimeUs Lib "ATR2010" (ByVal hDevice As Long, _
                                            ByVal nTimeUs As Long) As Boolean
```

Delphi:

```
Function DelayTimeUs (hDevice: Integer;
                      nTimeUs : LongInt) : Boolean;
                      StdCall; External 'ATR2010' Name ' DelayTimeUs ';
```

LabVIEW:

请参考相关演示程序。

功能：微秒级延时函数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 或 CreateDeviceEx 创建。

nTimeUs 时间常数。单位 1 微秒。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获错误码。

第三节、I0 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++ & C++ Builder:

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

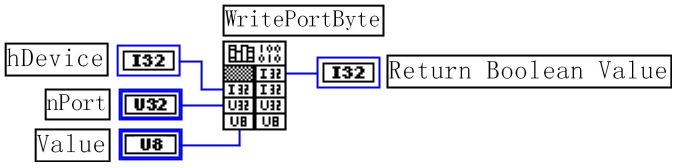
Visual Basic:

```
Declare Function WritePortByte Lib "ATR2010" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Byte) As Boolean
```

Delphi:

```
Function WritePortByte(hDevice : Integer;
                       nPort : LongWord;
                       Value : Byte) : Boolean;
                       StdCall; External 'ATR2010' 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。

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++ & C++ Builder:

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

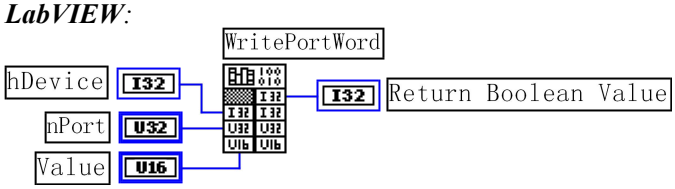
Visual Basic:

```
Declare Function WritePortWord Lib "ATR2010" (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 'ATR2010' 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 或 CreateDeviceEx 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 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++ & C++ Builder:

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

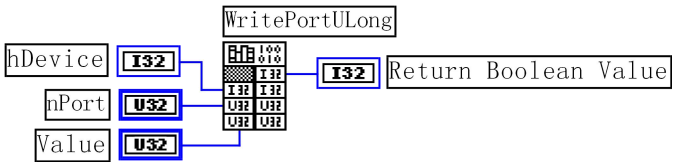
Visual Basic:

```
Declare Function WritePortULong Lib "ATR2010" (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 'ATR2010' 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。
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++ & C++ Builder:

```
BYTE ReadPortByte( HANDLE hDevice,
                   UINT nPort)
```

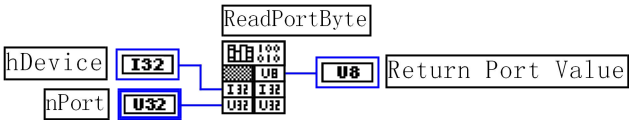
Visual Basic:

```
Declare Function ReadPortByte Lib "ATR2010" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

```
Function ReadPortByte(hDevice : Integer;
                      nPort : LongWord) : Byte;
                      StdCall; External 'ATR2010' 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。
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++ & C++ Builder:

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

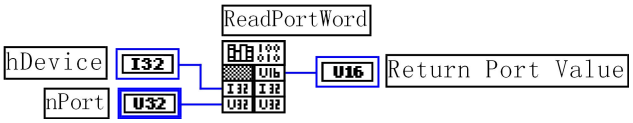
Visual Basic:

```
Declare Function ReadPortWord Lib "ATR2010" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

Delphi:

```
Function ReadPortWord(hDevice : Integer;
                      nPort : LongWord) : Word;
                      StdCall; External 'ATR2010' 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。
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++ & C++ Builder:

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

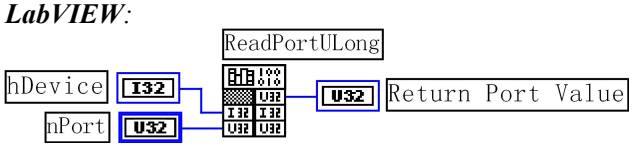
Visual Basic:

```
Declare Function ReadPortULong Lib "ATR2010" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                       nPort : LongWord) : LongWord;
                       StdCall; External 'ATR2010' 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 或 CreateDeviceEx 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、文件对象操作函数原型说明

◆ **创建文件对象**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function CreateFileObject Lib "ATR2010" (ByVal hDevice As Long, _
                                                 ByVal NewFileNameAs String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi:

```
Function CreateFileObject ( hDevice : Integer;
                            NewFileName: string;
                            Mode : Integer) : Integer;
                          Stdcall; external 'ATR2010' Name ' CreateFileObject ';
```

LabVIEW:

请参见相关演示程序。

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 或 CreateDeviceEx 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:\\ATR2010\\Data.Dat”，在 Basic 中，其语法格式如：“C:\ATR2010\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)：

常量名	常量值	功能定义
ATR2010_modeRead	0x0000	只读文件方式
ATR2010_modeWrite	0x0001	只写文件方式

ATR2010_modeReadWrite	0x0002	既读又写文件方式
ATR2010_modeCreate	0x1000	如果文件不存在可以创建该文件， 如果存在，则重建此文件，且清 0

返回值：若成功，则返回文件对象句柄。

相关函数： CreateDevice_ CreateFileObject WriteFile ReadFile ReleaseFile ReleaseDevice

◆ **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++ & C++ Builder:

```
BOOL WriteFile(HANDLE hFileObject,
               PVOID pDataBuffer,
               ULONG nWriteSizeBytes)
```

Visual Basic:

```
Declare Function WriteFile Lib "ATR2010" (ByVal hFileObject As Long,_
                                          ByRef pDataBuffer As Byte,_
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongWord) : Boolean;
                   Stdcall; external 'ATR2010' Name ' WriteFile ';
```

LabVIEW:

详见相关演示程序。

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 strFileName 指定。

参数：

hFileObject 设备对象句柄，它应由 CreateFileObject 创建。

pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。

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： CreateFileObject WriteFile ReadFile ReleaseFile

◆ **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++ & C++ Builder:

```
BOOL ReadFile( HANDLE hFileObject,
               PVOID pDataBuffer,
               ULONG OffsetBytes,
               ULONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "ATR2010" ( ByVal hFileObject As Long, _
```

```
                                        ByRef pDataBuffer As Integer, _
                                        ByVal OffsetBytes As Long, _
                                        ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile( hFileObject : Integer;
                   pDataBuffer : Pointer;
                   OffsetBytes : LongWord;
                   nReadSizeBytes : LongWord) : Boolean;
                   Stdcall; external 'ATR2010' Name ' ReadFile ';
```

LabVIEW:

详见相关演示程序。

功能：将磁盘数据从指定文件中读入用户内存空间中，其访问方式可由用户在创建文件对象时指定。

参数：

hFileObject 设备对象句柄，它应由 CreateFileObject 创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针，可以是用户分配的数组空间。

OffsetBytes 指定从文件开始端所偏移的读位置。

nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。

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： CreateFileObject WriteFile ReadFile ReleaseFile

- **设置文件偏移位置**

函数原型：

Visual C++ & C++ Builder:

```
BOOL SetFileOffset (HANDLE hFileObject,
                    ULONG nOffsetBytes)
```

Visual Basic:

```
Declare Function SetFileOffset Lib "ATR2010" (ByVal hFileObject As Long,
                                              ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset ( hFileObject : Integer;
                         nOffsetBytes : LongWord) : Boolean;
                         Stdcall; external 'ATR2010' Name ' SetFileOffset ';
```

LabVIEW:

详见相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由 CreateFileObject 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： CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：

Visual C++ & C++ Builder:

ULONG GetFileLength (HANDLE hFileObject)

Visual Basic:

Declare Function GetFileLength Lib "ATR2010" (ByVal hFileObject As Long) As Long

Delphi:

Function GetFileLength (hFileObject : Integer) : LongWord;
Stdcall; external 'ATR2010' Name ' GetFileLength ';

LabVIEW:

详见相关演示程序。

功能：取得文件长度。

参数：hFileObject 设备对象句柄，它应由 CreateFileObject 创建。

返回值：若成功，则返回>1，否则返回 0，用户可以用 GetLastError 捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

◆ **释放设备文件对象**

函数原型：

Visual C++ & C++ Builder:

BOOL ReleaseFile(HANDLE hFileObject)

Visual Basic:

Declare Function ReleaseFile Lib "ATR2010" (ByVal hFileObject As Long) As Boolean

Delphi:

Function ReleaseFile(hFileObject : Integer) : Boolean;
Stdcall; external 'ATR2010' Name ' ReleaseFile ';

LabVIEW:

详见相关演示程序。

功能：释放设备文件对象。

参数：hFileObject 设备对象句柄，它应由 CreateFileObject 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： CreateFileObject WriteFile ReadFile ReleaseFile

◆ **取得指定磁盘的可用空间**

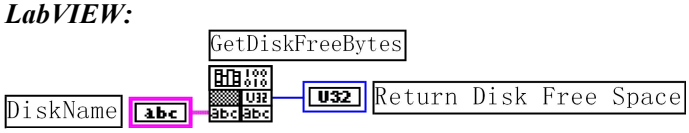
Visual C++ & C++ Builder:

ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName)

Visual Basic:

Declare Function GetDiskFreeBytes Lib "ATR2010" (ByVal strDiskName As String) As Currency

LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：strDiskName 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，否则返回零值，用户可用 GetLastError 捕获错误码。注

意使用 64 位整型变量。