

ART8011 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

目 录	1
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 PC104 设备	2
第三节、如何批量取得 AD 数据	2
第四节、哪些函数对您不是必须的	4
第三章 PC104 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“ART8011_”）	6
第二节、设备对象管理函数原型说明	7
第三节、AD 程序查询方式采样操作函数原型说明	8
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 硬件参数结构（ART8011_PARA_AD）	14
第二节、AD 状态参数结构（ART8011_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	18
第三节、AD 测试应用程序创建并形成的数据文件格式	19
第六章 上层用户函数接口应用实例	19
第一节、简易程序演示说明	19
第二节、高级程序演示说明	19
第七章 共用函数介绍	19
第一节、公用接口函数总列表（每个函数省略了前缀“ART8011_”）	20
第二节、IO 端口读写函数原型说明	20
第三节、线程操作函数原型说明	23
第四节、文件对象操作函数原型说明	25
第五节、其他函数原型说明	28

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。若您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 PC104xxxx_ 则被省略。如 ART8011_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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

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](#)、[ReadDeviceProAD](#) 等。而底层用户函数如 [WritePortByte](#)、[ReadPortByte](#)……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上 D 型插座等管脚分配情况。

第二节、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，[ReadDeviceProAD](#) 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、如何批量取得 AD 数据

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](#)。

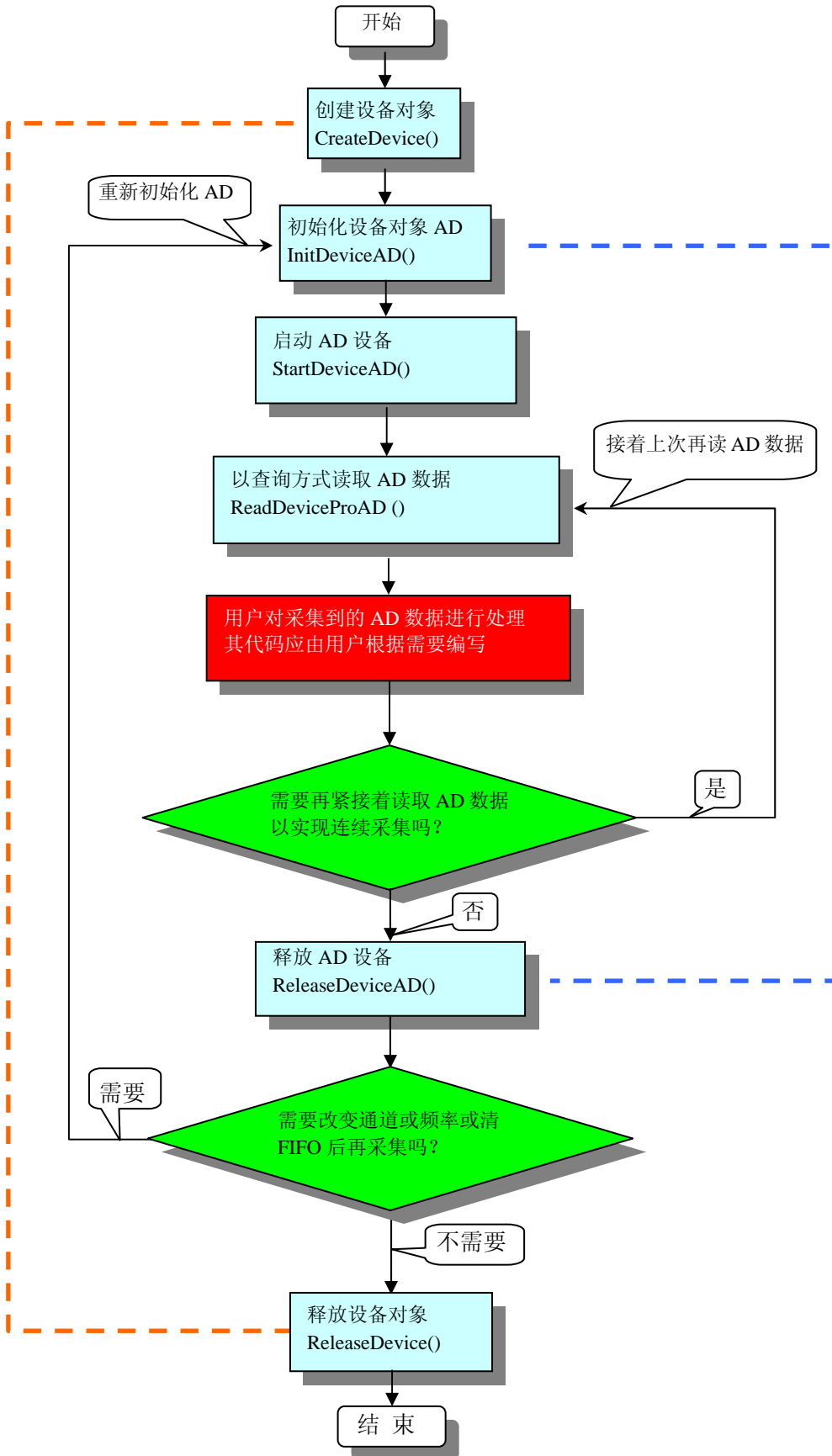


图 2.1.1 查询方式 AD 采集过程

第四节、哪些函数对您不是必须的

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。而

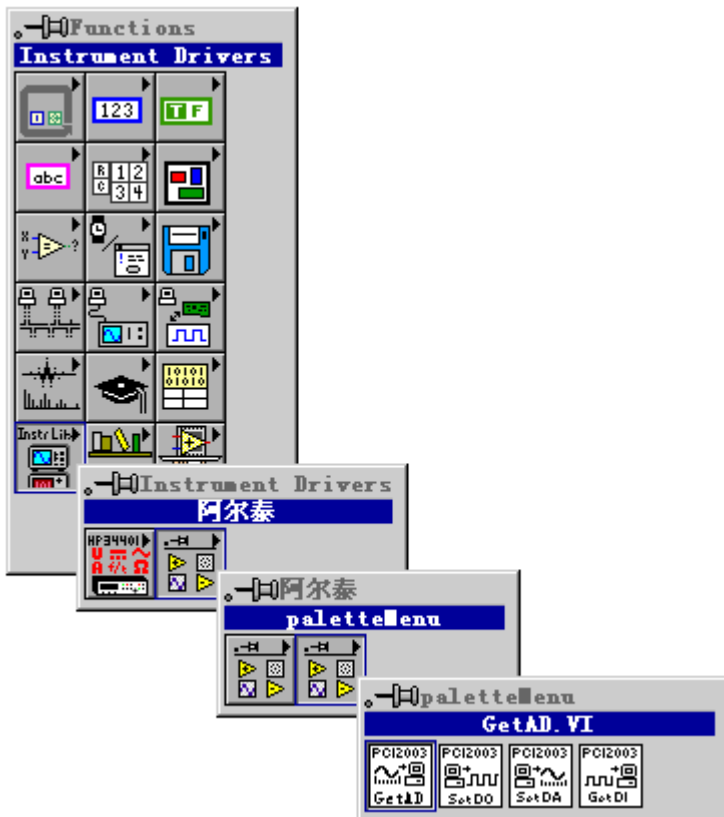
[WritePortByte](#), [WritePortWord](#), [WritePortULong](#), [ReadPortByte](#), [ReadPortWord](#), [ReadPortULong](#)则对PC104 用户来讲,可以说完全是辅助性,它们只是对我公司驱动程序的一种功能补充,对用户额外提供的,它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问,而没有这些函数,您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PC104 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[ReadDeviceProAD](#)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PC104 总线复杂的控制协议,同是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“ART8011_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PC104 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCurrentBaseAddr	取得当前设备基地址	上层及底层用户
ReleaseDevice	关闭设备, 且释放 PC104 总线设备对象	上层及底层用户
② 程序方式 AD 读取函数		
ADCalibration	设备校准函数	上层用户
InitDeviceAD	初始化 AD 部件准备传输	上层用户
StartDeviceAD	启动 AD 设备, 开始转换	上层用户
SetDeviceTrigAD	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
GetDevStatusAD	取得当前设备状态	上层用户
ReadDeviceProAD	连续批量读取设备上的 AD 数据	上层用户
StopDeviceAD	暂停 AD 设备	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
③ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
LoadBaseAddr	将基地址从系统中读出	
SaveBaseAddr	将基地址保存至系统中	

使用需知:**Visual C++ & C++Builder:**

要使用如下函数关键的问题是:

首先, 必须在您的源程序中包含如下语句:

```
#include "C:\Art\ART8011\INCLUDE\ART8011.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 ART8011.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。然后加入如下语句:

```
#include "ART8011.H"
```

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。

C++ Builder:

要使用如下函数一个关键的问题是首先必须将我们提供的头文件(ART8011.H)写进您的源程序头部。如:

```
#include "\Art\ART8011\Include\ART8011.h", 然后再将 ART8011.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”(Add to Project)命令, 在弹出的对话框中分别选择文件类型: Library file (*.lib), 即可选择 ART8011.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。
```

Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单, 执行其中的“添加模块”(Add Module)命令, 在弹出的对话框中选择 ART8011.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 程序均是需编译后在独立环境中运行。所以用户若在解释环境中运行这些代码, 我们不能保证完全顺利运行。

Delphi:

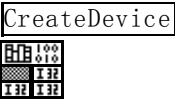
要使用如下函数一个关键的问题是首先必须将我们提供的单元模块文件 (*.Pas)加入到您的 Delphi 工程

中。其方法是选择 Delphi 编程环境中的 View 菜单，执行其中的"Project Manager"命令，在弹出的对话框中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 ART8011.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“ART8011”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ART8011; // 注意： 在此加入驱动程序接口单元 ART8011
```

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 ART8011.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ReadDeviceProAD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++ & C++Builder :

```
HANDLE CreateDevice (WORD wBaseAddress)
```

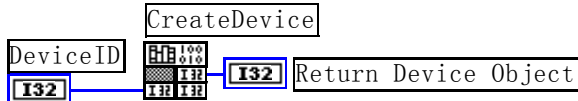
Visual Basic :

```
Declare Function CreateDevice Lib "ART8011" (ByVal wBaseAddress As Integer) As Long
```

Delphi:

```
Function CreateDevice(wBaseAddress : Word) : Integer;
  StdCall; External 'ART8011' Name 'CreateDevice';
```

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：

wBaseAddress 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 "ART8011" (ByVal hDevice As Long) As Integer

Delphi:

Function GetDeviceCurrentBaseAddr (hDevice : Integer): Word;
StdCall; External 'ART8011' 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GetLastErrorEx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)

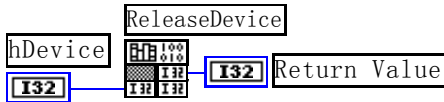
Visual Basic:

Declare Function ReleaseDevice Lib "ART8011" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDevice(hDevice : Integer) : Boolean;
StdCall; External 'ART8011' 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 程序查询方式采样操作函数原型说明

◆ AD 校准

函数原型:

Visual C++ & C++Builder:

BOOL ADCalibration(HANDLE hDevice)

Visual Basic:

Declare Function ADCalibration Lib "ART8011" (ByVal hDevice As Long) As Boolean

Delphi:

Function ADCalibration(hDevice : Integer) : Boolean;
StdCall; External 'ART8011' Name 'ADCalibration';

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++ & C++Builder:

BOOL InitDeviceAD(HANDLE hDevice,
PART8011_PARA_AD pADPara)

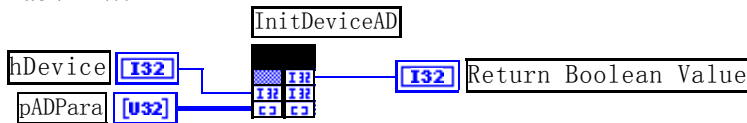
Visual Basic:

Declare Function InitDeviceAD Lib "ART8011" (ByVal hDevice As Long, _
ByRef pADPara As ART8011_PARA_AD) As Boolean

Delphi:

Function InitDeviceAD(hDevice : Integer;
pADPara :PART8011_PARA_AD):Boolean;
StdCall; External 'ART8011' Name ' InitDeviceAD ';

LabVIEW:



功能: 它负责初始化设备对象中的AD部件, 为设备的操作就绪做有关准备工作, 如预置AD采集通道、采样频率等。但它并不启动AD设备, 若要启动AD设备, 须在调用此函数之后再调用[StartDeviceAD](#)。

参数:

hDevice 设备对象句柄, 它应由[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。关于ART8011_PARA_AD具体定义请参考ART8011.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《[AD硬件参数结构](#)》章节。

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](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

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

函数原型:

Visual C++ & C++Builder:

BOOL StartDeviceAD (HANDLE hDevice)

Visual Basic:

Declare Function StartDeviceAD Lib "ART8011" (ByVal hDevice As Long) As Boolean

Delphi:

Function StartDeviceAD (hDevice : Integer): Boolean;
StdCall; External 'ART8011' Name ' StartDeviceAD ';

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: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
[SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
[StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 产生软件触发事件

函数原型:

Visual C++ & C++Builder:

BOOL SetDeviceTrigAD (HANDLE hDevice)

Visual Basic:

Declare Function SetDeviceTrigAD Lib "ART8011" (ByVal hDevice As Long) As Boolean

Delphi:

Function SetDeviceTrigAD (hDevice : Integer) : Boolean;
StdCall; External 'ART8011' Name ' SetDeviceTrigAD ';

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: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
 [SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
 [StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 取得 AD 状态标志

函数原型:

Visual C++ & C++Builder:

BOOL GetDevStatusAD (HANDLE hDevice,
PART8011_STATUS_AD pADStatus);

Visual Basic:

Declare Function GetDevStatusAD Lib "ART8011" (ByVal hDevice As Long,_
ByRef pADStatus As ART8011_STATUS_AD) As Boolean

Delphi:

Function GetDevStatusAD (hDevice : Integer;
pADStatus : PART8011_STATUS_AD) : Boolean;
StdCall; External 'ART8011' Name ' GetDevStatusAD ';

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 \(ART8011 STATUS_AD\)](#)》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: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
 [SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
 [StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ 程序方式读取设备上的 AD 数据

函数原型:

Visual C++ & C++Builder:

BOOL ReadDeviceProAD (HANDLE hDevice,
 PWORD pADBuffer,
 ULONG nReadSizeWords,
 PLONG nRetSizeWords)

Visual Basic:

Declare Function ReadDeviceProAD Lib "ART8011" (ByVal hDevice As Long,_
ByRef pADBuffer As Integer,_
ByVal nReadSizeWords As Long,_
ByRef nRetSizeWords As Long) As Boolean

Delphi:

Function ReadDeviceProAD (hDevice : Integer;
pADBuffer : Pointer;

```
nReadSizeWords : LongInt;  
nRetSizeWords : Pointer) : Boolean;  
StdCall; External 'ART8011' Name ' ReadDeviceProAD ';
```

LabVIEW:

请参考相关演示程序。

功能: 程序方式读 AD 数据。

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 ReadDeviceProAD 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。

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参数值的, 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: [CreateDevice](#) [InitDeviceAD](#) [StartDeviceAD](#)
 [SetDeviceTrigAD](#) [GetDevStatusAD](#) [ReadDeviceProAD](#)
 [StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ **暂停 AD 设备**

函数原型:

Visual C++ & C++Builder:

BOOL StopDeviceAD (HANDLE hDevice)

Visual Basic:

Declare Function StopDeviceAD Lib "ART8011" (ByVal hDevice As Long)As Boolean

Delphi:

Function StopDeviceAD (hDevice : Integer) : Boolean;
 StdCall; External 'ART8011' Name ' StopDeviceAD ';

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](#) [ReadDeviceProAD](#)
 [StopDeviceAD](#) [ReleaseDeviceAD](#) [ReleaseDevice](#)

◆ **释放设备上的 AD 部件**

函数原型:

Visual C++ & C++ Builder:

BOOL ReleaseDeviceAD(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceAD Lib "ART8011" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDeviceAD (hDevice : Integer) : Boolean;
 StdCall; External 'ART8011' Name ' ReleaseDeviceAD ';

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。

需要注意的是, [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](#)

◆ 程序查询方式采样函数一般调用顺序

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

注明: 用户可以反复执行第⑤步, 以实现采集。

关于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++ & C++ Builder:

```
BOOL LoadParaAD(HANDLE hDevice,
                PART8011_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "ART8011" (ByVal hDevice As Long, _
                                           ByRef pADPara As ART8011_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD (hDevice : Integer;
                    pADPara : PART8011_PARA_AD) : Boolean;
  StdCall; External 'ART8011' Name 'LoadParaAD';
```

LabVIEW:

请参考相关演示程序。

功能: 负责从 Windows 系统中读取设备的硬件参数。

参数:

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

pADPara属于PART8011_PARA_AD的结构指针类型, 它负责返回PC104 硬件参数值, 关于结构指针类型PART8011_PARA_AD请参考ART8011.h或ART8011.Bas或ART8011.Pas函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

◆ 往 Windows 系统写入设备硬件参数函数

函数原型:

Visual C++ & C++ Builder:

BOOL SaveParaAD (HANDLE hDevice,
PART8011_PARA_AD pADPara)

Visual Basic:

Declare Function SaveParaAD Lib "ART8011" (ByVal hDevice As Long, _
ByRef pADPara As ART8011_PARA_AD) As Boolean

Delphi:

Function SaveParaAD (hDevice : Integer;
pADPara : PART8011_PARA_AD) : Boolean;
StdCall; External 'ART8011' Name ' SaveParaAD ';

LabVIEW:

请参考相关演示程序。

功能: 负责把用户设置的硬件参数保存在 Windows 系统中, 以供下次使用。

参数:

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

pADPara 设备硬件参数, 关于 ART8011_PARA_AD 的详细介绍请参考 ART8011.h 或 ART8011.Bas 或 ART8011.Pas 函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

◆ **AD 采样参数复位至出厂默认值函数**

函数原型:

Visual C++ & C++ Builder:

BOOL ResetParaAD (HANDLE hDevice,
PART8011_PARA_AD pADPara)

Visual Basic:

Declare Function ResetParaAD Lib "ART8011" (ByVal hDevice As Long, _
ByRef pADPara As ART8011_PARA_AD) As Boolean

Delphi:

Function ResetParaAD (hDevice : Integer;
pADPara : PART8011_PARA_AD) : Boolean;
StdCall; External 'ART8011' Name ' ResetParaAD ';

LabVIEW:

请参考相关演示程序。

功能: 将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无法确定错误原因的后果。

参数:

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

pADPara 设备硬件参数, 它负责在参数被复位后返回其复位后的值。关于 ART8011_PARA_AD 的详细介绍请参考 ART8011.h 或 ART8011.Bas 或 ART8011.Pas 函数原型定义文件, 也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

◆ **读基地址函数**

函数原型:

Visual C++ & C++ Builder:

BOOL LoadBaseAddr (HANDLE hTDevice,
PWORD pBaseAddr)

Visual Basic:

Declare Function LoadBaseAddr Lib "ART8011" (ByVal hDevice As Long, _
ByRef pBaseAddr As Integer) As Boolean

Delphi:

Function LoadBaseAddr (hDevice : Integer;

pBaseAddr: Pointer) : Boolean;
StdCall; External 'ART8011' Name 'LoadBaseAddr';

LabVIEW:

请参考相关演示程序。

功能: 将基地址从系统中读出。

参数:

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

pBaseAddr 返回基地址的值。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

◆ **保存基地址函数**

函数原型:

Visual C++ & C++ Builder:

```
BOOL SaveBaseAddr(HANDLE hTDevice,
                  WORD wBaseAddr)
```

Visual Basic:

```
Declare Function SaveBaseAddr Lib "ART8011" (ByVal hDevice As Long, _
                                             ByVal wBaseAddr As Integer) As Boolean
```

Delphi:

```
Function SaveBaseAddr (hDevice : Integer;
                       wBaseAddr: Word) : Boolean;
StdCall; External 'ART8011' Name 'SaveBaseAddr';
```

LabVIEW:

请参考相关演示程序。

功能: 将基地址保存至系统中。

参数:

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

wBaseAddr 基地址。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

第四章 硬件参数结构

第一节、AD 硬件参数结构 (ART8011_PARA_AD)

Visual C++ & C++ Builder:

```
typedef struct _ART8011_PARA_AD
{
```

LONG bChannelArray[2]; // 采样通道选择阵列, 分别控制 2 个通道, =TRUE 表示该通道采样, 否则不采样

LONG Frequency; // 采集频率,单位为 Hz

LONG InputRange[2]; // 模拟量输入量程范围

LONG CouplingType; // 耦合类型(直流耦合, 交流耦合)

LONG M_Length; // M 段长度(字), 总的取值范围 1-1M, 但是 M 加 N 长度不能大于 1M

LONG N_Length; // N 段长度(字), 总的取值范围 1-1M, 但是 M 加 N 长度不能大于 1M

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 bAvailableTrig; // 当触发事件提前时, 是否有效? (=TRUE:有效, =FALSE:忽略)

LONG BandwidthLimit; // 带宽限制使能(=TRUE:带宽限制 10M, =FALSE:全带宽)

```

LONG TrigLevelVolt;      // 触发电平，量程与 AD 量程一致
LONG ClockSource;       // 时钟源选择(内/外时钟源)
} ART8011_PARA_AD, *PART8011_PARA_AD;

```

Visual Basic :

```

Private Type ART8011_PARA_AD
    bChannelArray(0 to 1) As Long
    Frequency As Long
    InputRange(0 to 1) As Long
    CouplingType As Long
    M_Length As Long
    N_Length As Long
    TriggerMode As Long
    ATRTriggerChannel As Long
    TriggerSource As Long
    TriggerDir As Long
    bAvailableTrig As Long
    BandwidthLimit As Long
    TrigLevelVolt As Long
    ClockSource As Long
End Type

```

Delphi:

```

type // 定义结构体数据类型
PART8011_PARA_AD = ^ ART8011_PARA_AD; // 指针类型结构
ART8011_PARA_AD = record // 标记为记录型
    bChannelArray: Array[0...1] of LongInt;
    Frequency : LongInt;
    InputRange : Array[0...1] of LongInt;
    CouplingType: LongInt;
    M_Length : LongInt;
    N_Length : LongInt;
    TriggerMode : LongInt;
    ATRTriggerChannel : LongInt;
    TriggerSource : LongInt;
    TriggerDir : LongInt;
    bAvailableTrig : LongInt;
    BandwidthLimit : LongInt;
    TrigLevelVolt : LongInt;
    ClockSource : LongInt;
End;

```

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。

硬件参数说明:

bChannelArray 采样通道选择阵列，分别控制 2 个通道，=TRUE 表示该通道采样，否则不采样。

Frequency AD 采样频率，单位 Hz，其范围应根据具体的设备而定，但其最小值为 10Hz。切忌不能等于 0，本设备的 AD 采样频率取值最大为 100M。

InputRange 模拟量输入量程范围选择。

常量名	常量值	功能定义
ART8011_INPUT_N1000_P1000mV	0x00	±1000mV
ART8011_INPUT_N5000_P5000mV	0x01	±5000mV

关于各个量程下采集的数据ADBuffer[]如何换算成相应的电压值，请参考《[AD原码LSB数据转换成电压值的](#)

[换算方法](#)》章节。

CouplingType 耦合类型选择(直流耦合, 交流耦合)。

常量名	常量值	功能定义
ART8011_COUPLING_AC	0x00	交流耦合
ART8011_COUPLING_DC	0x01	直流耦合

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

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

TriggerMode AD 触发模式选择。

常量名	常量值	功能定义
ART8011_TRIGMODE_MIDL	0x00	中间触发
ART8011_TRIGMODE_POST	0x01	后触发
ART8011_TRIGMODE_PRE	0x02	预触发
ART8011_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
ART8011_ATRTRIG_CH0	0x00	通道 0 信号作为 ATR 输入
ART8011_ATRTRIG_CH1	0x01	通道 1 信号作为 ATR 输入

TriggerSource 触发源选择。

常量名	常量值	功能定义
ART8011_TRIGMODE_SOFT	0x00	软件触发
ART8011_TRIGSRC_DTR	0x01	选择 DTR 作为触发源
ART8011_TRIGSRC_ATR	0x02	选择 ATR 作为触发源

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
ART8011_TRIGDIR_NEGATIVE	0x00	下降沿触发
ART8011_TRIGDIR_POSITIVE	0x01	上升沿触发
ART8011_TRIGDIR_POSIT_NEGAT	0x02	上下边沿均触发

bAvailableTrig 当触发事件提前时, 是否有效? =TRUE: 有效, =FALSE: 忽略。

BandwidthLimit 带宽限制使能。=TRUE: 带宽限制 10M, =FALSE: 全带宽。

TrigLevelVolt 触发电平, 量程与 AD 量程一致。

ClockSource AD 触发时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
ART8011_CLOCKSRC_IN	0x00	内部时钟定时触发
ART8011_CLOCKSRC_OUT	0x01	外部时钟定时触发

当选择内时钟时, 其AD定时触发时钟为板上时钟振荡器经分频得到。它的大小由**Frequency**参数决定。

当选择外时钟时:

当选择连续采集时(即**ADMode** = ART8011_ADMODE_SEQUENCE), 其AD定时触发时钟为外界时钟输入 CLKIN得到, 而**Frequency**参数则自动失效。

但是当选择分组采集时(即**ADMode** = ART8011_ADMODE_GROUP), 外时钟则是每一组的触发时钟信号, 而组内的触发频率则由**Frequency**参数决定, 由此可见, 此时外时钟触发周期必须大于每组总周期, 否则紧跟其后的某一外时钟可能会被失效。

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

第二节、AD 状态参数结构 (ART8011_STATUS_AD)

Visual C++ & C++Builder :

```
typedef struct _ART8011_STATUS_AD
{
    LONG bADEanble;
    LONG bTrigger;
    LONG bComplete;
    LONG bAheadTrig;
} ART8011_STATUS_AD, *PART8011_STATUS_AD;
```

Visual Basic :

```
Private Type ART8011_STATUS_AD
    bADEanble As Long
    bTrigger As Long
    bComplete As Long
    bAheadTrig As Long
End Type
```

Delphi:

```
Type // 定义结构体数据类型
PART8011_STATUS_AD = ^ART8011_STATUS_AD; // 指针类型结构
ART8011_STATUS_AD = record // 标记为记录型
    bADEanble : LongInt;
    bTrigger : LongInt;
    bComplete : LongInt;
    bAheadTrig : LongInt;
End;
```

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，[GetDevStatusProAD](#)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。

bADEanble 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。

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	$Volt = (10000.00 / 4096) * (ADBuffer[0] \& 0xFFFF) - 5000.00$	[-5000, +4997.55]
±1000mV	$Volt = (2000.00 / 4096) * (ADBuffer[0] \& 0xFFFF) - 1000.00$	[-1000, +999.51]

下面举例说明各种语言的换算过程（以±1000mV 量程为例）

Visual C++&C++Builder :

```
Lsb = (ADBuffer[0])&0x0FFFF;
Volt = (2000.00/4096) * Lsb - 1000.00;
```

Visual Basic :

```
Lsb = (ADBuffer [0]) And &H0FFFF
Volt = (2000.00/4096) * Lsb - 1000.00
```

Delphi:

Lsb: = (ADBuffer[0]) And \$0FFFF;

Volt: = (20000.0/4096) * Lsb - 1000.00;

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

单通道采集, 当通道总数首末通道相等时, 假如此时首末通道=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(假如FirstChannel=0, LastChannel=1):

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(假如FirstChannel=0, LastChannel=3):

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的问题, 尤其是在任意通道数采集时。否则, 用户无法将规则排在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢? 我们建议的方法是, 每次从设备上读取的点数应置为所选通道数量的整数倍长, 这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集, 则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数), 这里设为 2048。试想, 如此一来, 每次读取的 2048 个点中的第一个点始终对应于 1 通道数据, 第二个点始终对应于 2 通道, 第三个点再应于 1 通道, 第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据, 第 2048 个点对应 2 通道。这样一来, 每次读取的段长正好包含了从首通道到末通道的完整轮回, 如此一来, 用户只须按通道排列规则, 按正常的处理方法循环处理每一批数据。而对于其他情况也是如此, 比如 3 个通道采集, 则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题, 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断的多通道数据链。而通道序列一行则说明了随着连续采样的延续, 其各通道数据在其整个数据链中的排放次序, 这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回, 即便不考虑是否能一次读完的问题, 仅对于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, 而且还高效呢? 还是正如前面所说, 采用通道数的整数倍长读取每一段数据。如表中列举的方法 1 (为了说明问题, 我们每读取一段数据只读取 2n 即 3*2=6 个数据)。从方法 1 不难看出, 每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长, 则出现问题, 从表中可以看出, 第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道, 而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据, 而第三段缓冲区中的数据则对应于第 3 通道……, 这显然不利于循环有效处理数据。

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

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

第三节、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)
    ART8011_PARA_AD ADPara; // 保存硬件参数
    LONG StaticOverFlow;  // 同批文件识别码
    LONG HeadEndFlag;     // 文件结束位
    ART8011_STATUS_AD ADStatus;
} 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

怎样使用[ReadDeviceProAD](#)函数直接取得AD数据

Visual C++ & C++Builder:

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

[程序] | [阿尔泰测控演示系统] | [ART8011 100M 2 通道高速 AD 采集卡] | [Microsoft Visual C++] | [简易代码演示] | [AD 方式]

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 ART8011.h 和 ADDoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [ART8011 100M 2 通道高速 AD 采集卡] | [Microsoft Visual C++] | [高级代码演示]

其默认存放路径为：系统盘\ART\ART8011\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“ART8011_”）

函数名	函数功能	备注
① 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
② 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	
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	
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
④ 其他函数		
GetDevVersion	获取设备固件及程序版本	
DelayTimeUs	微秒级延时函数	

第二节、IO 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

函数原型：

Visual C++ & C++ Builder:

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

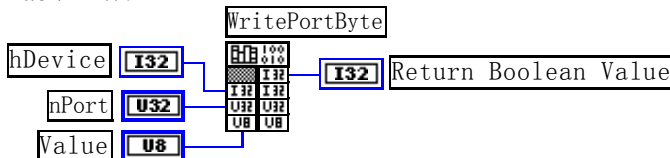
Visual Basic:

Declare Function WritePortByte Lib "ART8011" (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 'ART8011' 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](#)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

函数原型：

Visual C++ & C++ Builder:

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

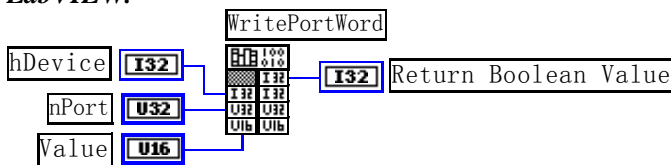
Visual Basic:

Declare Function WritePortWord Lib "ART8011" (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 'ART8011' Name ' WritePortWord ';

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

nPort 设备的 I/O 端口号。

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

函数原型：

Visual C++ & C++ Builder:

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

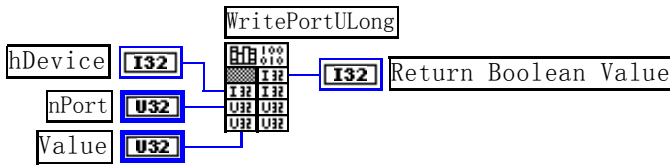
Visual Basic:

Declare Function WritePortULong Lib "ART8011" (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 'ART8011' 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](#) 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

函数原型:

Visual C++ & C++ Builder:

BYTE ReadPortByte(HANDLE hDevice,
 UINT nPort)

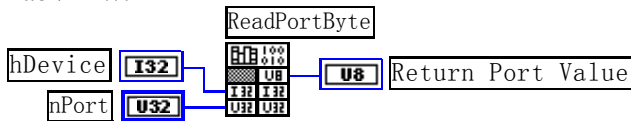
Visual Basic:

Declare Function ReadPortByte Lib "ART8011" (ByVal hDevice As Long, _
 ByVal nPort As Long) As Byte

Delphi:

Function ReadPortByte(hDevice : Integer;
 nPort : LongWord) : Byte;
 StdCall; External 'ART8011' 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](#) 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

函数原型:

Visual C++ & C++ Builder:

WORD ReadPortWord(HANDLE hDevice,
 UINT nPort)

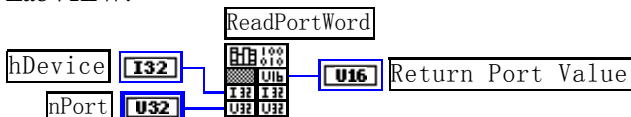
Visual Basic:

Declare Function ReadPortWord Lib "ART8011" (ByVal hDevice As Long, _
 ByVal nPort As Long) As Integer

Delphi:

Function ReadPortWord(hDevice : Integer;
 nPort : LongWord) : Word;
 StdCall; External 'ART8011' 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](#)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

函数原型：

Visual C++ & C++ Builder:

ULONG ReadPortULong(HANDLE hDevice,
 UINT nPort)

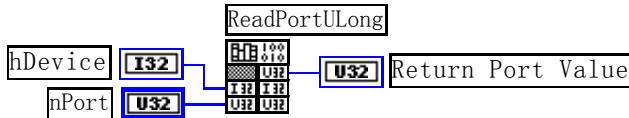
Visual Basic:

Declare Function ReadPortULong Lib "ART8011" (ByVal hDevice As Long, _
 ByVal nPort As Long) As Long

Delphi:

Function ReadPortULong(hDevice : Integer;
 nPort : LongWord) : LongWord;
 StdCall; External 'ART8011' Name ' ReadPortULong ';

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

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

第三节、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

函数原型：

Visual C++ & C++ Builder:

BOOL CreateVBThread(HANDLE *hThread,
 LPTHREAD_START_ROUTINE RoutineAddr)

Visual Basic :

Declare Function CreateVBThread Lib "ART8011" (ByRef hThread As Long, _
 ByVal RoutineAddr As Long) As Boolean

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

参数：

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

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

返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： [CreateVBThread](#) [TerminateVBThread](#)

注意: StartThread 指向的函数或过程必须放在 VB 的模块文件中, 如 ART8011.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

函数原型:

Visual C++ & C++ Builder:

[BOOL TerminateVBThread\(HANDLE hThread\)](#)

Visual Basic:

[Declare Function TerminateVBThread Lib "ART8011" \(ByVal hThread As Long\) As Boolean](#)

功能: 在VB中删除由[CreateVBThread](#)创建的子线程对象。

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

返回值: 当成功删除子线程对象时, 返回TRUE, 否则返回FALSE, 用户可用GetLastErrorEx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

函数原型:

Visual C++ & C++ Builder:

[HANDLE CreateSystemEvent\(void\)](#)

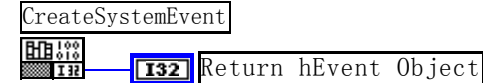
Visual Basic:

[Declare Function CreateSystemEvent Lib "ART8011" \(\) As Long](#)

Delphi:

[Function CreateSystemEvent\(\) : Integer; StdCall; External 'ART8011' 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

函数原型:

Visual C++ & C++ Builder:

BOOL ReleaseSystemEvent(HANDLE hEvent);

Visual Basic:

Declare Function ReleaseSystemEvent Lib " ART8011 " (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;

StdCall; External 'ART8011' 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。

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 "ART8011" (ByVal hDevice As Long, _
ByVal NewFileName As String, _
ByVal Mode As Integer) As Long

Delphi:

Function CreateFileObject (hDevice : Integer;

NewFileName : String;

Mode : Integer) : Integer;

Stdcall; external 'ART8011' 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](#)创建。

NewFileName 新文件名。

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
ART8011_modeRead	0x0000	只读文件方式
ART8011_modeWrite	0x0001	只写文件方式
ART8011_modeReadWrite	0x0002	既读又写文件方式
ART8011_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
ART8011_typeText	0x4000	以文本方式操作文件

返回值: 若成功, 则返回文件对象句柄。

相关函数: [CreateDevice](#)

[ReadFile](#)

[CreateFileObject](#)

[ReleaseFile](#)

[WriteFile](#)

[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 "ART8011" (ByVal hObject As Long, _
ByRef pDataBuffer As Integer, _
ByVal nWriteSizeBytes As Long) As Boolean

Delphi:

Function WriteFile(hObject: Integer;
pDataBuffer : Pointer;
nWriteSizeBytes : LongWord) : Boolean;
Stdcall; external 'ART8011' 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GetLastErrorEx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 hObject,
PVOID pDataBuffer,
ULONG nOffsetBytes,
ULONG nReadSizeBytes)

Visual Basic:

Declare Function ReadFile Lib "ART8011" (ByVal hObject As Long, _
ByRef pDataBuffer As Integer, _
ByVal nOffsetBytes As Long, _
ByVal nReadSizeBytes As Long) As Boolean

Delphi:

Function ReadFile(hObject : Integer;
pDataBuffer : Pointer;
nOffsetBytes : LongWord;
nReadSizeBytes : LongWord) : Boolean;
Stdcall; external 'ART8011' 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。

nOffsetBytes 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GetLastErrorEx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 "ART8011" (ByVal hFileObject As Long,_
ByVal nOffsetBytes As Long) As Boolean

Delphi:

Function SetFileOffset (hFileObject : Integer;
nOffsetBytes : LongWord) : Boolean;
Stdcall; external 'ART8011' 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GetLastErrorEx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 "ART8011" (ByVal hFileObject As Long) As Long

Delphi:

Function GetFileLength (hFileObject : Integer) : LongWord;
Stdcall; external 'ART8011' 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GetLastErrorEx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 "ART8011" (ByVal hFileObject As Long) As Boolean

Delphi:

Function ReleaseFile(hFileObject : Integer) : Boolean;
Stdcall; external 'ART8011' 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GetLastErrorEx捕获错误码。

相关函数: [CreateFileObject](#) [WriteFile](#) [ReadFile](#)
[ReleaseFile](#)

◆ 取得指定磁盘的可用空间

函数原型:

Visual C++ & C++ Builder:

ULONGLONG GetDiskFreeBytes(LPCTSTR DiskName)

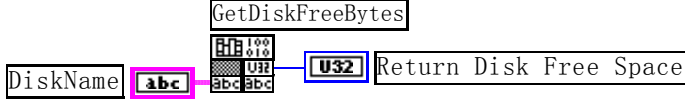
Visual Basic:

Declare Function GetDiskFreeBytes Lib "ART8011" (ByVal DiskName As String) As Currency

Delphi:

Function GetDiskFreeBytes (DiskName: String) : Currency;
Stdcall; external 'ART8011' Name ' GetDiskFreeBytes ';

LabVIEW:



功能: 取得指定磁盘的可用剩余空间(以字为单位)。

参数: DiskName 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GetLastErrorEx捕获错误码。注意使用 64 位整型变量。

第五节、其他函数原型说明

◆ 获取设备固件及程序版本

函数原型:

Visual C++ & C++ Builder:

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

Visual Basic:

Declare Function GetDevVersion Lib "ART8011" (ByVal hDevice As Long, _
ByRef pulFmwVersion As Long,_
ByRef pulDriverVersion As Long) As Boolean

Delphi:

Function GetDevVersion (hDevice : Integer;
pulFmwVersion : Pointer;
pulDriverVersion : Pointer) : Boolean;
StdCall; External 'ART8011' Name ' GetDevVersion ';

LabVIEW:

请参考相关演示程序。

功能: 获取设备固件及程序版本。

参数:

hDevice设备对象句柄, 它应由CreateDevice创建。

pulFmwVersion 固件版本。

pulDriverVersion 驱动版本。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

◆ 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 "ART8011" (ByVal hDevice As Long, _
ByVal nTimeUs As Long) As Boolean

Delphi:

Function DelayTimeUs (hDevice: Integer;
nTimeUs : LongInt) : Boolean;
StdCall; External 'ART8011' 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](#)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。