

ART2932 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、如何用非空查询方式取得 AD 数据 | 3 |
| 第四节、如何用半满查询方式取得 AD 数据 | 3 |
| 第五节、如何实现开关量的简便操作 | 6 |
| 第六节、哪些函数对您不是必须的 | 6 |
| 第三章 PC104 设备专用函数接口介绍 | 6 |
| 第一节、设备驱动接口函数列表（每个函数省略了前缀“ART2932_”） | 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 | 9 |
| 第四节、AD 硬件参数系统保存与读取函数原型说明 | 14 |
| 第五节、DA 模拟量输出操作函数原型说明 | 17 |
| 第六节、CNT 计数与定时器操作函数原型说明 | 18 |
| 第七节、DIO 数字开关量输入输出简易操作函数原型说明 | 19 |
| 第四章 硬件参数结构 | 20 |
| 第一节、AD 硬件参数结构（ART2932_PARA_AD） | 20 |
| 第二节、AD 状态参数结构（ART2932_STATUS_AD） | 24 |
| 第五章 数据格式转换与排列规则 | 25 |
| 第一节、AD 原始数据 LSB 转换成电压值 Volt 的换算方法 | 25 |
| 第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则 | 26 |
| 第三节、DA 电压值转换成 LSB 原码数据的换算方法 | 27 |
| 第六章 上层用户函数接口应用实例 | 27 |
| 第一节、简易程序演示说明 | 27 |
| 第二节、高级程序演示说明 | 28 |
| 第七章 公共接口函数介绍 | 28 |
| 第一节、公用接口函数总列表（每个函数省略了前缀“ART2932_”） | 28 |
| 第二节、线程操作函数原型说明 | 29 |
| 第三节、IO 端口读写函数原型说明 | 31 |
| 第四节、文件对象操作函数原型说明 | 35 |

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 ART2932Inst.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。若您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 ART2932_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 [InitDeviceProAD](#)、[ReadDeviceProAD_Npt](#)、[SetDeviceDO](#) 等。而底层用户函数如 [WriteRegisterULong](#)、[ReadRegisterULong](#)、[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 [InitDeviceProAD](#) 可以使用 hDevice 句柄以程序查询方式初始化设备的 AD 部件，[ReadDeviceProAD_Npt](#)（或 [ReadDeviceProAD_Half](#)）函数可以用 hDevice 句柄实现对 AD 数据的采样读取，[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。

第三节、如何用非空查询方式取得 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 [ReadDeviceProAD_Npt](#) 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，[ReleaseDeviceProAD](#) 便可帮您实现（但设备对象 hDevice 依然存在）。（注：[ReadDeviceProAD_Npt](#) 虽然主要面对批量读取、高速连续采集而设计，但亦可用它以单点或几点的方式读取 AD 数据，以满足慢速、高实时性采集需要）。具体执行流程请看下面的图 2.1.1。

第四节、如何用半满查询方式取得 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，接着调用 [GetDevStatusAD](#) 函数以查询 AD 的存储器 FIFO 的半满状态，如果达到半满状态，即可用 [ReadDeviceProAD_Half](#) 函数读取一批半满长度（或半满以下）的 AD 数据，然后接着再查询 FIFO 的半满状态，若有效再读取，就这样反复查询状态反复读取 AD 数据即可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，[ReleaseDeviceProAD](#) 便可帮您实现（但设备对象 hDevice 依然存在）。（注：[ReadDeviceProAD_Half](#) 函数在半满状态有效时也可以单点或几点的方式读取 AD 数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）具体执行流程请看下面的图 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](#)。绿色虚线 [InitDeviceAD](#) 与 [ReleaseDeviceAD](#) 成对称方式出现。

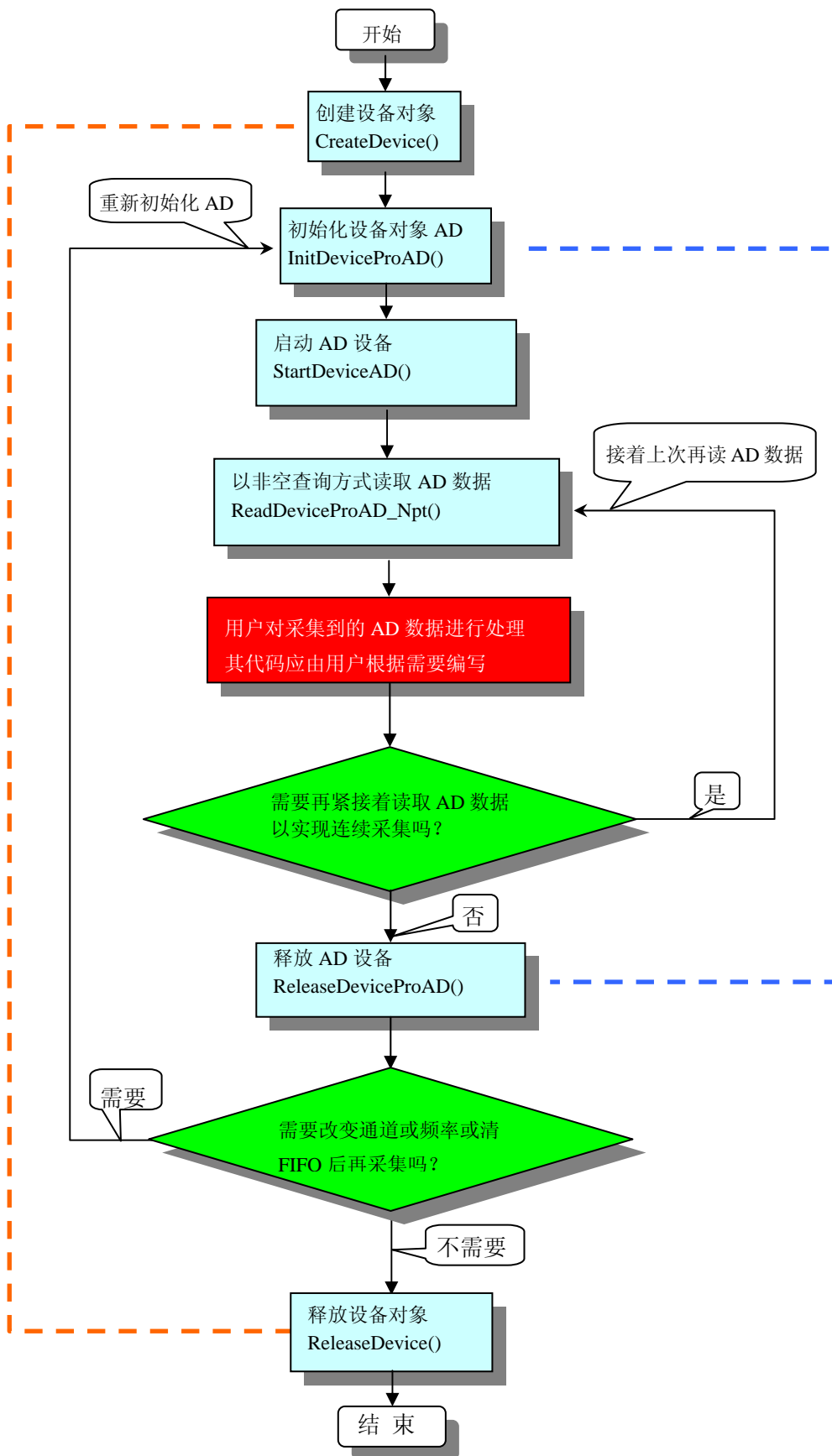


图 2.1.1 非空查询方式 AD 采集过程

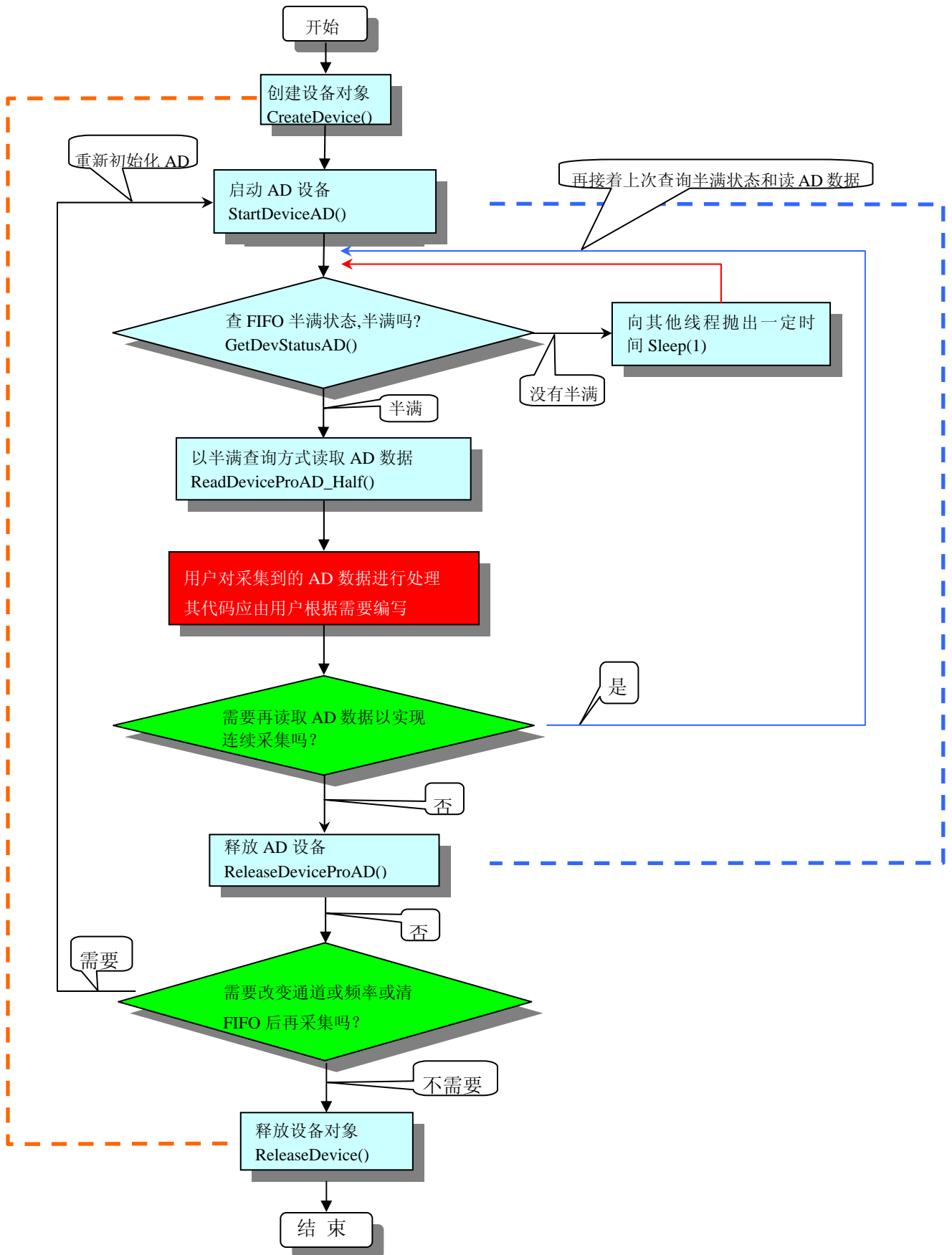


图 2.1.2 半满查询方式 AD 采集过程

第五节、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bDOSts[16]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bDISts[16]中的相应元素决定。

第六节、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“ART2932_”）

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 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 |
| ReadDeviceProAD_Npt | 连续读取当前 PC104 设备上的 AD 数据 | 上层用户 |
| GetDevStatusAD | 取得当前 PC104 设备 FIFO 半满状态 | 上层用户 |
| ReadDeviceProAD_Half | 连续批量读取 PC104 设备上的 AD 数据 | 上层用户 |
| StopDeviceAD | 暂停 AD 设备 | 上层用户 |
| ReleaseDeviceProAD | 释放设备上的 AD 部件 | 上层用户 |
| 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 | |
| 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 | |
| ④ DA 输出操作函数 | | |
| WriteDevProDA | 写 DA 数据 | 上层用户 |
| ⑤ 计数器操作函数 | | |
| SetDeviceCNT | 设置计数器的初值 | |
| GetDeviceCNT | 取得各路计数器的当前计数值 | |
| ⑥ 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 ART2932.h 和 ART2932.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 (ART2932.lib) 加入到您的工程中。

```
#include "ART2932.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 ART2932.h 和 ART2932.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 ART2932.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，在弹出的对话框中选择 ART2932.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 ART2932.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，在弹出的对话框中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 ART2932.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：“ART2932”。如：

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
ART2932; // 注意： 在此加入驱动程序接口单元 ART2932
```

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 wBaseAddress)
```

Visual Basic:

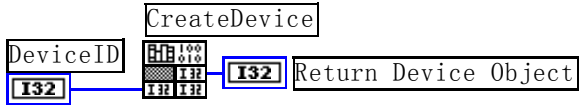
```
Declare Function CreateDevice Lib "ART2932" (ByVal wBaseAddress As Integer) As Long
```

Delphi:

```
Function CreateDevice(wBaseAddress : Word):Integer;
```

```
StdCall; External 'ART2932' 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:

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 "ART2932" (ByVal hDevice As Long) As Integer`

Delphi:

`Function GetDeviceCurrentBaseAddr (hDevice : Integer): Word;`

`StdCall; External 'ART2932' 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:

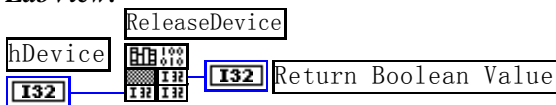
`Declare Function ReleaseDevice Lib "ART2932" (ByVal hDevice As Long) As Boolean`

Delphi:

`Function ReleaseDevice(hDevice : Integer):Boolean;`

`StdCall; External 'ART2932' 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 程序查询方式采样操作函数原型说明

◆ 初始化 AD 设备 (Initlize device AD for program mode)

函数原型:

Visual C++ & C++Builder:

```
BOOL InitDeviceProAD( HANDLE hDevice,  
PART2932_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceProAD Lib "ART2932" (ByVal hDevice As Long, _  
ByRef pADPara As ART2932_PARA_AD) As Boolean
```

Delphi:

```
Function InitDeviceProAD(hDevice : Integer;  
pADPara : PART2932_PARA_AD) : Boolean;  
StdCall; External 'ART2932' 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ART2932_PARA_AD具体定义请参考ART2932.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 GetLastError 捕获当前错误码, 并加以分析。

相关函数: [CreateDevice](#) [InitDeviceProAD](#) [StartDeviceAD](#)
[ReadDeviceProAD_Npt](#) [GetDevStatusAD](#) [ReadDeviceProAD_Half](#)
[StopDeviceAD](#) [ReleaseDeviceProAD](#) [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 "ART2932" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function StartDeviceAD (hDevice : Integer ): Boolean;  
StdCall; External 'ART2932' 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[InitDeviceProAD](#)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GetLastError捕获当前错误码，并加以分析。

相关函数：

| | | |
|-------------------------------------|------------------------------------|--------------------------------------|
| CreateDevice | InitDeviceProAD | StartDeviceAD |
| ReadDeviceProAD_Npt | GetDevStatusAD | ReadDeviceProAD_Half |
| StopDeviceAD | ReleaseDeviceProAD | ReleaseDevice |

◆ 取得 AD 的状态标志

函数原型：

Visual C++ & C++Builder:

```

BOOL GetDevStatusAD (HANDLE hDevice,
                    PART2153_STATUS_AD pADStatus)

```

Visual Basic:

```

Declare Function GetDevStatusAD Lib "ART2153" (ByVal hDevice As Long, _
                                             ByVal pADStatus As ART2153_STATUS_AD) As Boolean

```

Delphi:

```

Function GetDevStatusAD (hDevice : Integer;
                        pADStatus: PART2153_STATUS_AD):Boolean;
StdCall; External 'ART2153' Name 'GetDevStatusAD';

```

LabVIEW:

请参考相关演示程序。

功能： 一旦用户使用[StartDeviceAD](#)后，可以用此函数却查询AD状态，如是否被启动，是否被触发等信息。

参数：

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

pADStatus设备状态参数结构，它返回设备当前的各种状态，如板载RAM是否发生切换、重写、触发点是否产生等信息。关于具体操作请参考《[AD硬件参数结构](#)》。

返回值： 若 AD 成功取回标状态，则返回 TRUE，否则返回 FALSE。

相关函数：

| | | |
|-------------------------------------|------------------------------------|--------------------------------------|
| CreateDevice | InitDeviceProAD | StartDeviceAD |
| ReadDeviceProAD_Npt | GetDevStatusAD | ReadDeviceProAD_Half |
| StopDeviceAD | ReleaseDeviceProAD | ReleaseDevice |

◆ 读取 PC104 设备上的 AD 数据

① 使用 FIFO 的非空标志读取 AD 数据

函数原型：

Visual C++ & C++Builder:

```

BOOL ReadDeviceProAD_Npt( HANDLE hDevice,
                          WORD ADBuffer[],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords)

```

Visual Basic:

```

Declare Function ReadDeviceProAD_Npt Lib "ART2932" (ByVal hDevice As Long, _
                                                  ByVal ADBuffer[] As Integer, _
                                                  ByVal nReadSizeWords As Long, _
                                                  ByVal nRetSizeWords As Long) As Boolean

```

Delphi:

```
Function ReadDeviceProAD_Npt(hDevice : Integer;
                             ADBuffer[] : Pointer;
                             nReadSizeWords : LongInt;
                             nRetSizeWords : Pointer) : Boolean;
StdCall; External 'ART2932' Name 'ReadDeviceProAD_Npt';
```

LabVIEW:

请参考相关演示程序。

功能: 一旦用户使用[StartDeviceAD](#)后, 应立即用此函数读取设备上的AD数据。此函数使用FIFO的非空标志进行读取AD数据。

参数:

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

ADBuffer 接受AD数据的用户缓冲区, 它可以是一个用户定义的数组。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_Npt](#)操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间。此参数值只与ADBuffer[]指定的缓冲区大小有效, 而与FIFO存储器大小无效。

nRetSizeWords 返回实际读取的点数(或字数)。

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[ReleaseDeviceProAD](#)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GetLastError捕获当前错误码, 并加以分析。

| 当前错误码 | 功能定义 |
|------------|-------------|
| 0xE1000000 | 其他不可预知的错误 |
| 0xE2000000 | 表示用户提前终止读操作 |

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](#) [InitDeviceProAD](#) [StartDeviceAD](#)
 [ReadDeviceProAD_Npt](#) [GetDevStatusAD](#) [ReadDeviceProAD_Half](#)
 [StopDeviceAD](#) [ReleaseDeviceProAD](#) [ReleaseDevice](#)

② 使用 FIFO 的半满标志读取 AD 数据

◆ 当 FIFO 半满信号有效时, 批量读取 AD 数据

函数原型:

Visual C++ & C++Builder:

```
BOOL ReadDeviceProAD_Half( HANDLE hDevice,
                           WORD ADBuffer[],
                           LONG nReadSizeWords,
                           PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "ART2932" ( ByVal hDevice As Long, _
                                                    ByVal ADBuffer[] As Long, _
                                                    ByVal nReadSizeWords As Long, _
                                                    ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Half(hDevice : Integer;
                              ADBuffer[] : Pointer;
                              nReadSizeWords : LongInt;
                              nRetSizeWords : Pointer) : Boolean;
StdCall; External 'ART2932' Name 'ReadDeviceProAD_Half';
```

LabVIEW:

请参考相关演示程序。

功能: 一旦用户使用[GetDevStatusAD](#)后取得的FIFO状态**bHalf**等于TRUE(即半满状态有效)时, 应立即用此函数读取设备上FIFO中的半满AD数据。

参数:

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

ADBuffer 接受AD数据的用户缓冲区, 通常可以是一个用户定义的数组。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_Half](#)操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间, 而且应等于FIFO总容量的二分之一(如果用户有特殊需要可以小于FIFO的二分之一长)。比如设备上配置了 1K FIFO, 即 1024 字, 那么这个参数应指定为 512 或小于 512。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值: 如果成功的读取由nReadSizeWords参数指定量的AD数据到用户缓冲区, 则返回TRUE, 否则返回FALSE, 用户可用[GetLastError](#)捕获当前错误码, 并加以分析。

相关函数:

| | | |
|-------------------------------------|------------------------------------|--------------------------------------|
| CreateDevice | InitDeviceProAD | StartDeviceAD |
| ReadDeviceProAD_Npt | GetDevStatusAD | ReadDeviceProAD_Half |
| StopDeviceAD | ReleaseDeviceProAD | ReleaseDevice |

◆ **暂停 AD 设备**

函数原型:

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StopDeviceAD Lib "ART2932" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function StopDeviceAD (hDevice : Integer) : Boolean;
StdCall; External 'ART2932' 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[GetLastError](#)捕获当前错误码, 并加以分析。

相关函数:

| | | |
|-------------------------------------|------------------------------------|--------------------------------------|
| CreateDevice | InitDeviceProAD | StartDeviceAD |
| ReadDeviceProAD_Npt | GetDevStatusAD | ReadDeviceProAD_Half |
| StopDeviceAD | ReleaseDeviceProAD | ReleaseDevice |

◆ **释放设备上的 AD 部件**

函数原型:

Visual C++ & C++ Builder:

BOOL ReleaseDeviceProAD(HANDLE hDevice)

Visual Basic:

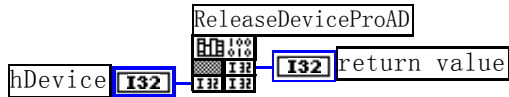
Declare Function ReleaseDeviceProAD Lib "ART2932" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDeviceProAD (hDevice : Integer) : Boolean;

StdCall; External 'ART2932' Name 'ReleaseDeviceProAD ';

LabVIEW:



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GetLastError捕获错误码。

应注意的是， [InitDeviceProAD](#) 必须和 [ReleaseDeviceProAD](#) 函数一一对应， 即当您执行了一次 [InitDeviceProAD](#)后，再一次执行这些函数前，必须执行一次[ReleaseDeviceProAD](#)函数，以释放由[InitDeviceProAD](#)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[InitDeviceProAD](#)函数时，那些软硬件资源才可被再次使用。

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

◆ **程序查询方式采样函数一般调用顺序**

非空查询方式:

- ① [CreateDevice](#)
- ② [InitDeviceProAD](#)
- ③ [StartDeviceAD](#)
- ④ [ReadDeviceProAD_Npt](#)
- ⑤ [StopDeviceAD](#)
- ⑥ [ReleaseDeviceProAD](#)
- ⑦ [ReleaseDevice](#)

注明: 用户可以反复执行第④步，以实现高速连续不间断大容量采集。

半满查询方式:

- ① [CreateDevice](#)
- ② [InitDeviceProAD](#)
- ③ [StartDeviceAD](#)
- ④ [GetDevStatusAD](#)
- ⑤ [ReadDeviceProAD_Half](#)
- ⑥ [StopDeviceAD](#)
- ⑦ [ReleaseDeviceProAD](#)
- ⑧ [ReleaseDevice](#)

注明: 用户可以反复执行第④、⑤步，以实现高速连续不间断大容量采集。

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,
                PART2932_PARA_AD pADPara)
```

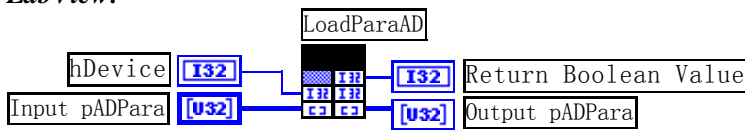
Visual Basic:

```
Declare Function LoadParaAD Lib "ART2932" (ByVal hDevice As Long, _
                                           ByRef pADPara As ART2932_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD( hDevice : Integer;
                    pADPara : PART2932_PARA_AD):Boolean;
StdCall; External 'ART2932' Name 'LoadParaAD';
```

Lab View:



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 PART2932_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 PART2932_PARA 请参考相应 ART2932.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:

```

:
ART2932_PARA_AD ADPara;
HANDLE hDevice;
HDevice = CreateDevice(0); // 管理一个 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 ART2932_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：

Visual C++ & C++Builder:

```
BOOL SaveParaAD(HANDLE hDevice,  
                PART2932_PARA_AD pADPara)
```

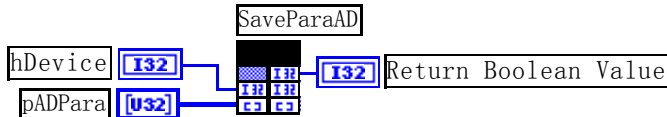
Visual Basic:

```
Declare Function SaveParaAD Lib "ART2932" (ByVal hDevice As Long, _  
                                           ByRef pADPara As ART2932_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;  
                    pADPara : PART2932_PARA_AD):Boolean;  
StdCall; External 'ART2932' 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](#)

◆ **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,  
                  PART2932_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "ART2932" (ByVal hDevice As Long, _  
                                             ByRef pADPara As ART2932_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD ( hDevice : Integer;  
                      pADPara : PART2932_PARA_AD) : Boolean;  
StdCall; External 'ART2932' 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从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ART2932_PARA_AD的详细介绍请参考ART2932.h或ART2932.Bas或ART2932.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 "ART2932" (ByVal hDevice As Long, _
                                             ByVal pBaseAddr As Integer) As Boolean
```

Delphi:

```
Function LoadBaseAddr(hDevice : Integer;
                    pBaseAddr : Pointer) : Boolean;
StdCall; External 'ART2932' 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 "ART2932" (ByVal hDevice As Long, _
                                             ByVal wBaseAddr As Integer) As Boolean
```

Delphi:

```
Function SaveBaseAddr ( hDevice : Integer;
                    wBaseAddr : Word) : Boolean;
StdCall; External 'ART2932' 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](#)

第五节、DA 模拟量输出操作函数原型说明

◆ 输出模拟信号到指定通道

函数原型:

Visual C++ & C++Builder:

```
BOOL WriteDeviceDA ( HANDLE hDevice,
                    LONG OutputRange,
                    SHORT nDAData,
                    int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceDA Lib "ART2932" (ByVal hDevice As Long, _
                                             ByVal OutputRange As Long, _
                                             ByVal nDAData As Integer, _
                                             ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceDA (hDevice : Integer;
                       OutputRange : Long;
                       nDAData : SmallInt;
                       nDAChannel : Integer) : Boolean;
  StdCall; External 'ART2932' Name 'WriteDeviceDA ';
```

LabVIEW:

请参考相关演示程序。

功能: 设置指定通道的模拟量输出量程范围。

参数:

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

OutputRange指定通道的输出量程范围，其取值如下表，关于各个量程电压值到原码值之间的换算请参考《[DA电压值转换成LSB原码数据的换算方法](#)》章节。

| 常量名 | 常量值 | 功能定义 |
|--------------------------------|--------|-----------|
| ART2932_OUTPUT_0_P5000mV | 0x0000 | 0~5000mV |
| ART2932_OUTPUT_0_P10000mV | 0x0001 | 0~10000mV |
| ART2932_OUTPUT_0_P10800mV | 0x0002 | 0~10800mV |
| ART2932_OUTPUT_N5000_P5000mV | 0x0003 | ±5000mV |
| ART2932_OUTPUT_N10000_P10000mV | 0x0004 | ±10000mV |
| ART2932_OUTPUT_N10800_P10800mV | 0x0005 | ±10800mV |

nDAData 指输出的DA原始码数据，它的取值范围为[0, 4095]，它与实际输出模拟量值的对应关系请参考《[DA电压值转换成LSB原码数据的换算方法](#)》章节。

nDAChannel 需要指定的模拟量通道号，其取值范围为[0, 3]。写入 4，代表四个通道都启动。

返回值: 若成功，返回TRUE，则由nDAChannel指定的通道被设置成由OutputRange指定的量程范围；否则返回FALSE，您可以调用GetLastError函数取得错误或错误字符信息。

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

◆ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [WriteDevProDA](#)
- ③ [ReleaseDevice](#)

用户可以反复执行第②步，以进行模拟量输出不断变化（可以和 AD 采样同时进行，互不影响）。

第六节、CNT 计数与定时器操作函数原型说明

◆ 设置计数器的初值

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceCNT(HANDLE hDevice,
                  WORD CNTVal,
                  WORD ControlMode,
                  WORD nChannel)
```

Visual Basic:

```
Declare Function SetDeviceCNT Lib "ART2932" (ByVal hDevice As Long, _
                                           ByVal CNTVal As Integer, _
                                           ByVal ControlMode As Integer, _
                                           ByVal nChannel As Integer) As Boolean
```

Delphi:

```
Function SetDeviceCNT (hDevice : Integer;
                      CNTVal : Word;
                      ControlMode : Word;
                      nChannel : Word) : Boolean;
StdCall; External 'ART2932' Name 'SetDeviceCNT';
```

LabVIEW:

请参考相关演示程序。

功能：设置计数器的初值。

参数：

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

CNTVal 计数初值。

ControlMode 方式控制字。其选项值如下表：

| 常量名 | 常量值 | 功能定义 |
|-----------------------------|--------|----------|
| ART2932_GATEMODE_POSITIVE_0 | 0x0000 | 计数结束产生中断 |
| ART2932_GATEMODE_RISING_1 | 0x0001 | 可编程单拍脉冲 |
| ART2932_GATEMODE_POSITIVE_2 | 0x0002 | 频率发生器 |
| ART2932_GATEMODE_POSITIVE_3 | 0x0003 | 方波发生器 |
| ART2932_GATEMODE_POSITIVE_4 | 0x0004 | 软件触发选通 |
| ART2932_GATEMODE_RISING_5 | 0x0005 | 硬件触发选通 |

nChannel 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](#)

◆ 取得各路计数器的当前计数值

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceCNT (HANDLE hDevice,
                  PWORD pCNTVal,
```

WORD nChannel)

Visual Basic:

Declare Function GetDeviceCNT Lib "ART2932" (ByVal hDevice As Long, _
ByRef pCNTVal As Integer, _
ByVal nChannel As Long) As Boolean

Delphi:

Function GetDeviceCNT (hDevice : Integer;
pCNTVal: Pointer;
nChannel: Word) : Boolean;
StdCall; External 'ART2932' Name 'GetDeviceCNT ';

LabVIEW:

请参考相关演示程序。

功能: 取得各路计数器的当前计数值。

参数:

hDevice设备对象句柄，它应由[CreateDevice](#)创建。

pCNTVal 取得计数初值。

nChannel 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](#) [SetDeviceCNT](#) [ReleaseDevice](#)

第七节、DIO 数字开关量输入输出简易操作函数原型说明

◆ 八路开关量输入

函数原型:

Visual C++ & C++Builder:

BOOL GetDeviceDI (HANDLE hDevice,
BYTE bDISts[8])

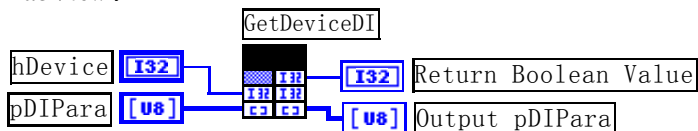
Visual Basic:

Declare Function GetDeviceDI Lib "ART2932" (ByVal hDevice As Long, _
ByVal bDISts(0 to 7) As Byte) As Boolean

Delphi:

Function GetDeviceDI (hDevice : Integer;
bDISts : Pointer):Boolean;
StdCall; External 'ART2932' 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。

bDISts 八路开关量输入状态的参数结构，共有 8 个成员变量，分别对应于 DI0-DI7 路开关量输入状态位。

如果 bDISts[0]为“1”则使 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

返回值: 若成功，返回 TRUE，其 bDISts[x]中的值有效；否则返回 FALSE，其 bDISts[x]中的值无效。

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

◆ 八路开关量输出

函数原型:

Visual C++ & C++Builder:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[8])
```

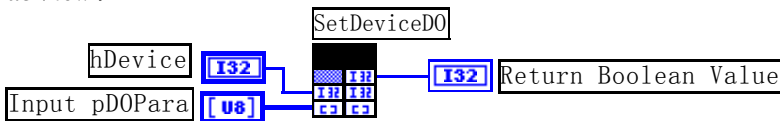
Visual Basic:

```
Declare Function SetDeviceDO Lib "ART2932" ( ByVal hDevice As Long, _
                                           ByVal bDOSts(0 to 7) As Byte) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer;
                    bDOSts : Pointer):Boolean;
StdCall; External 'ART2932' 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。

bDOSts 八路开关量输出状态的参数结构,共有 8 个成员变量,分别对应于 DO0-DO7 路开关量输出状态位。比如置 bDOSts[0]为“1”则使 0 通道处于“开”状态,若为“0”则置 0 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 DO7 共 8 个成员变量赋初值,其值必须为“1”或“0”。

返回值: 若成功,返回 TRUE,否则返回 FALSE。

相关函数: [CreateDevice](#) [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 (ART2932_PARA_AD)

Visual C++ & C++Builder:

```
typedef struct _ART2932_PARA_AD
{
    LONG ADMode;           // AD 模式选择(连续/分组方式)
    LONG FirstChannel;    // 首通道[0, 15]
    LONG LastChannel;     // 末通道[0, 15],要求末通道必须大于或等于首通道
    LONG Frequency;      // 采集频率,单位为 Hz, [1, 250000]
```

```

LONG GroupInterval; // 分组时的组间间隔(单位: 微秒)[1, 65536]
LONG LoopsOfGroup; // 组内循环次数[1, 255]
LONG Gains; // 增益设置
LONG InputRange; // 模拟量输入量程范围
LONG TriggerMode; // 触发模式选择
LONG TriggerSource; // 触发源选择
LONG TriggerType; // 触发类型选择(边沿触发/电平触发)
LONG TriggerDir; // 触发方向选择(正向/负向触发)
LONG TrigWindow; // 触发灵敏窗[1, 255]
LONG ClockSource; // 时钟源选择(内/外时钟源)
LONG bClockOutput; // 允许时钟输出到 CLKOUT,=TRUE:允许时钟输出, =FALSE:禁止时钟输

```

出

```

LONG GroundingMode; // 接地方式 (单端或双端选择)
}ART2932_PARA_AD, *PART2932_PARA_AD;

```

Visual Basic :

Private Type ART2932_PARA_AD

```

ADMode As Long ' AD 模式选择(连续/分组方式)
FirstChannel As Long ' 首通道
LastChannel As Long ' 末通道, 要求末通道必须大于或等于首通道
Frequency As Long ' 采集频率, 单位为 Hz
GroupInterval As Long ' 分组时的组间间隔(单位: 微秒)
LoopsOfGroup As Long ' 组内循环次数
Gains As Long ' 增益设置
InputRange As Long ' 模拟量输入量程范围
TriggerMode As Long ' 触发模式选择
TriggerSource As Long ' 触发源选择
TriggerType As Long ' 触发类型选择(边沿触发/脉冲触发)
TriggerDir As Long ' 触发方向选择(正向/负向触发)
TrigWindow As Long ' 触发灵敏窗[1, 255]
ClockSource As Long ' 时钟源选择(内/外时钟源)
bClockOutput As Long ' 允许时钟输出
GroundingMode As Long ' 接地方式 (单端或双端选择)

```

End Type

Delphi:

Type // 定义结构体数据类型

```

PART2932_PARA_AD = ^ ART2932_PARA_AD; // 指针类型结构
ART2932_PARA_AD = record // 标记为记录型
    ADMode : LontInt; // AD 模式选择(连续/分组方式)
    FirstChannel : LontInt; // 首通道
    LastChannel : LontInt; // 末通道, 要求末通道必须大于或等于首通道
    Frequency : LontInt; // 采集频率, 单位为 Hz
    GroupInterval : LontInt; // 分组时的组间间隔(单位: 微秒)
    LoopsOfGroup : LontInt; // 组内循环次数
    Gains : LontInt; // 增益设置

```

```

InputRange : LontInt;           // 模拟量输入量程范围
TriggerMode : LontInt;          // 触发模式选择
TriggerSource: LontInt;         // 触发源选择
TriggerType : LontInt;          // 触发类型选择(边沿触发/脉冲触发)
TriggerDir : LontInt;           // 触发方向选择(正向/负向触发)
TrigWindow: LontInt;           // 触发灵敏窗[1, 255]
ClockSource : LontInt;          // 时钟源选择(内/外时钟源)
bClockOutput : LontInt;         // 允许时钟输出
GroundingMode: LongInt;         // 接地方式(单端或双端选择)

End;

```

LabVIEW:
 请参考相关演示程序。

该结构实在太简易了，其原因就是 PC104 设备是系统全自动管理的设备，再加上驱动程序的合理设计与封装，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 设备是一种更易于管理和使用的设备。

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[InitDeviceProAD](#)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。

ADMode AD 采样模式。它的取值如下表：

| 常量名 | 常量值 | 功能定义 |
|-------------------------|------|--------|
| ART2932_ADMODE_SEQUENCE | 0x00 | 连续采集模式 |
| ART2932_ADMODE_GROUP | 0x01 | 分组采集模式 |

连续采集模式：表示所有通道在采样过程中均按相等时间间隔转换，即所有被采样的点在时间轴上其间隔完全相等。在图 4.1 中的过程，若没有mt的组间间隔时间[GroupInterval](#)，就属于连续采样的模式。

分组采集模式：表示所有采样的数据点均以指定的通道数分成若干组，组内各通道数据点按等间隔采样，其频率由[Frequency](#)参数决定，组与组之间则有相当的间隔时间，其间隔长度由参数[GroupInterval](#)决定，可以精确到微秒。如图 4.1 即是分组采样的整过情况。



图 4.1

说明： $t = 1/\text{Frequency}$
 $mt = \text{GroupInterval}$
 $n = \text{ChannelCount}$

FirstChannel AD采样首通道，其取值范围为[0, 15]，它应等于或小于[LastChannel](#)参数。

LastChannel AD采样末通道，其取值范围为[0, 15]，它应等于或大于[FirstChannel](#)参数。

Frequency AD 采样频率，本设备的频率取值范围为[1, 250KHz]。

注意：

在内时钟(即[ClockSource](#) = ART2932_CLOCKSRC_IN)方式下：

若连续采集(即[ADMode](#) = ART2932_ADMODE_SEQUENCE)时，此参数控制各处通道间的采样频率。若分组采集(即[ADMode](#)= ART2932_ADMODE_GROUP)时，则此参数控制各组组内的采样频率，而组间时间则由[GroupInterval](#)决定。

在外时钟(即ClockSource = ART2932_CLOCKSRC_OUT)方式下:

若连续采集(即ADMode = ART2932_ADMODE_SEQUENCE)时,此参数自动失效,因为外时钟的频率代替了此参数设定的频率。若为分组采集(即ADMode = ART2932_ADMODE_GROUP)时,则该参数控制各组组内的采样频率,而外时钟则是每组的触发频率。此时, GroupInterval参数无效。

GroupInterval 组间间隔,单位微秒 uS,其范围[1, 65536],通常由用户确定。但是一般情况下,此间隔时间应不小于组内相邻两通道的间隔。在内时钟连续采集模式和外时钟模式下,则参数无效。

LoopsOfGroup 在分组采集模式中,控制各组的循环次数。取值范围为[1, 255]。比如,1、2、3、4通道分组采样,当此参数为2时,则表示每1、2、3、4、1、2、3、4为一采样组,然后再延时GroupInterval指定的时间再接着采集1、2、3、4、1、2、3、4,依此类推。

Gains AD 采样程控增益。

| 常量名 | 常量值 | 功能定义 |
|---------------------|------|-------|
| ART2932_GAINS_1MULT | 0x00 | 1 倍增益 |
| ART2932_GAINS_2MULT | 0x01 | 2 倍增益 |
| ART2932_GAINS_4MULT | 0x02 | 4 倍增益 |
| ART2932_GAINS_8MULT | 0x03 | 8 倍增益 |

InputRange 被测模拟信号输入范围,取值如下表:

| 常量名 | 常量值 | 功能定义 |
|-------------------------------|------|-----------|
| ART2932_INPUT_N10000_P10000mV | 0x00 | ±10000mV |
| ART2932_INPUT_N5000_P5000mV | 0x01 | ±5000mV |
| ART2932_INPUT_N2500_P2500mV | 0x02 | ±2500mV |
| ART2932_INPUT_0_P10000mV | 0x03 | 0~10000mV |

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。

TriggerMode AD 触发模式。

| 常量名 | 常量值 | 功能定义 |
|-----------------------|------|--------------|
| ART2932_TRIGMODE_SOFT | 0x00 | 软件触发(属于内触发) |
| ART2932_TRIGMODE_POST | 0x01 | 硬件后触发(属于外触发) |

TriggerSource AD 触发源。

| 常量名 | 常量值 | 功能定义 |
|---------------------|------|----------------|
| ART2932_TRIGSRC_ATR | 0x00 | 选择外部 ATR 作为触发源 |
| ART2932_TRIGSRC_DTR | 0x01 | 选择外部 DTR 作为触发源 |

TriggerType AD 触发类型。

| 常量名 | 常量值 | 功能定义 |
|------------------------|------|----------|
| ART2932_TRIGTYPE_EDGE | 0x00 | 边沿触发 |
| ART2932_TRIGTYPE_PULSE | 0x01 | 脉冲触发(电平) |

TriggerDir AD 触发方向。它的选项值如下表:

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

| | | |
|-----------------------------|------|-----------------|
| ART2932_TRIGDIR_NEGATIVE | 0x00 | 负向触发(低脉冲/下降沿触发) |
| ART2932_TRIGDIR_POSITIVE | 0x01 | 正向触发(高脉冲/上升沿触发) |
| ART2932_TRIGDIR_POSIT_NEGAT | 0x02 | 正负方向均有效 |

注明: ART2932_TRIGDIR_POSIT_NEGAT 在边沿类型下, 则表示不管是上边沿还是下边沿均触发。而在电平类型下, 无论正电平还是负电平均触发。

TrigWindow 触发灵敏窗时间值, 取值范围为[1, 255]。

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 |
|----------------------|------|----------|
| ART2932_CLOCKSRC_IN | 0x00 | 内部时钟定时触发 |
| ART2932_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** = ART2932_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** = ART2932_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。

bClockOutput AD 内部时钟输出使能控制, =1 或 TRUE, 表示允许内部的 AD 工作时钟输出, 否则禁止。

GroundingMode AD 接地方式选择。它的选项值如下表:

| 常量名 | 常量值 | 功能定义 |
|--------------------|------|-----------------------|
| ART2932_GNDMODE_SE | 0x00 | 单端方式(SE:Single end) |
| ART2932_GNDMODE_DI | 0x01 | 双端方式(DI:Differential) |

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

第二节、AD 状态参数结构 (ART2932_STATUS_AD)

Visual C++ & C++Builder:

```
typedef struct _ART2932_STATUS_AD
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bOverFull;
} ART2932_STATUS_AD, *PART2932_STATUS_AD;
```

Visual Basic:

```
Private Type ART2932_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bOverFull As Long
End Type
```

Delphi:

```
Type // 定义结构体数据类型
PART2932_STATUS_AD = ^ ART2932_STATUS_AD; // 指针类型结构
ART2932_STATUS_AD = record // 标记为记录型
    bNotEmpty : LongInt;
    bHalf : LongInt;
    bOverFull : 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，[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。

bNotEmpty AD 板载存储器 FIFO 的非空标志，=TRUE 表示存储器处在非空状态，即有可读数据，否则表示空。

bHalf AD 板载存储器 FIFO 的半满标志，=TRUE 表示存储器处在半满状态，即有至少有半满以上数据可读，否则表示在半满以下，可能有小于半满的数据可读。

bOverFull AD 板载存储器 FIFO 的满标志， = TRUE 表示存储器处在满状态， = FALSE 存储器在满以下。

相关函数: [CreateDevice](#) [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 $2^n = \text{LSB 总数}$ （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 | $\text{Volt} = (20000.00/8192) * (\text{ADBuffer}[0] \&0x1FFF) - 10000.00$ | [-10000.00, + 9999.69] |
| ±5000 | $\text{Volt} = (10000.00/8192) * (\text{ADBuffer}[0] \&0x1FFF) - 5000.00$ | [-5000.00, + 4999.84] |
| ±2500 | $\text{Volt} = (5000.00/8192) * (\text{ADBuffer}[0] \&0x1FFF) - 2500.00$ | [-2500.00, + 2499.92] |
| 0~10000 | $\text{Volt} = (10000.00/8192) * (\text{ADBuffer}[0] \&0x1FFF)$ | [0.00, + 9999.84] |

换算举例：（设置量程为±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 = 20000.00/8192; // 求出每个 LSB 原码单位电压值
Lsb = ADBuffer[0] &0x1FFF;
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 = 20000.00/8192 ' 求出每个 LSB 原码单位电压值

Lsb = ADBuffer(0) AND 8191 ' 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 := 20000.00/8192; // 求出每个 LSB 原码单位电压值

Lsb := ADBuffer[0] &0x1FFF;

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

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，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，但对用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2=6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效

处理数据。

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

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

第三节、DA 电压值转换成 LSB 原码数据的换算方法

| 量程(伏) | 计算机语言换算公式(标准 C 语法) | Lsb 取值范围 |
|-----------|---|-----------|
| 0~5000mV | $Lsb = Volt / (5000.00 / 4096)$ | [0, 4095] |
| 0~10000mV | $Lsb = Volt / (10000.00 / 4096)$ | [0, 4095] |
| 0~10800mV | $Lsb = Volt / (10800.00 / 4096)$ | [0, 4095] |
| ±5000mV | $Lsb = Volt / (10000.00 / 4096) + 2048$ | [0, 4095] |
| ±10000mV | $Lsb = Volt / (20000.00 / 4096) + 2048$ | [0, 4095] |
| ±10800mV | $Lsb = Volt / (21600.00 / 4096) + 2048$ | [0, 4095] |

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

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_Npt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] | [ART2932 16 路 AD 4 路 DA 8 路 DIO 3 路计数器卡] | [Microsoft Visual C++] | [简易代码演示] | [AD 非空方式]

二、怎样使用ReadDeviceProAD_Half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] | [ART2932 16路AD 4路DA 8路DIO 3路计数器卡] | [Microsoft Visual C++] | [简易代码演示] | [AD 半满方式]

三、怎样使用WriteDevProDA函数取得DA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] | [ART2932 16路AD 4路DA 8路DIO 3路计数器卡] | [Microsoft Visual C++] | [简易代码演示] | [DA 方式]

四、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 ART2932.h 和 Sys.cpp)。

[程序] | [阿尔泰测控演示系统] | [ART2932 16路AD 4路DA 8路DIO 3路计数器卡] | [Microsoft Visual C++] | [简易代码演示] | [开关量演示源程序]

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 ART2932.h 和 DADoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [ART2932 16路AD 4路DA 8路DIO 3路计数器卡] | [Microsoft Visual C++] | [高级代码演示]

其默认存放路径为：系统盘\ART\ART2932\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“ART2932_”）

| 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 RoutineAddr)
```

Visual Basic :

```
Declare Function CreateVBThread Lib "ART2932" (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 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 ART2932.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 hThread)

Visual Basic:

Declare Function TerminateVBThread Lib "ART2932" (ByVal hThread As Long) As Boolean

功能: 在VB中删除由CreateVBThread创建的子线程对象。

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

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

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

Visual Basic 程序举例:

```

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

```

◆ 创建内核系统事件

Visual C++ & C++ Builder:

HANDLE CreateSystemEvent(void)

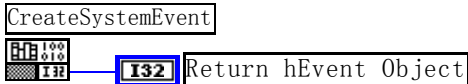
Visual Basic:

Declare Function CreateSystemEvent Lib " ART2932 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'ART2932' 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 "ART2932" (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
StdCall; External 'ART2932' 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 "ART2932" (ByVal hDevice As Long, _
ByVal nTimeUs As Long) As Boolean

Delphi:

Function DelayTimeUs (hDevice: Integer;
nTimeUs : LongInt) : Boolean;
StdCall; External 'ART2932' 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GetLastError捕获错误码。

第三节、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

Visual C++ & C++ Builder:

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

Visual Basic:

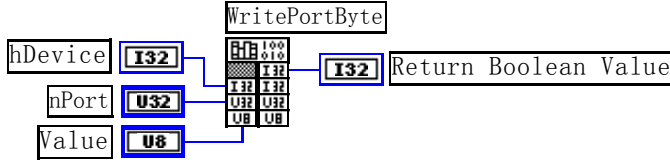
Declare Function WritePortByte Lib "ART2932" (ByVal hDevice As Long, _
ByVal nPort As Long, _
ByVal Value As Byte) As Boolean

Delphi:

Function WritePortByte(hDevice : Integer;
nPort : LongWord;
Value : Byte) : Boolean;

StdCall; External 'ART2932' 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GetLastError捕获当前错误码。

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

◆ 以双字(16Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

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

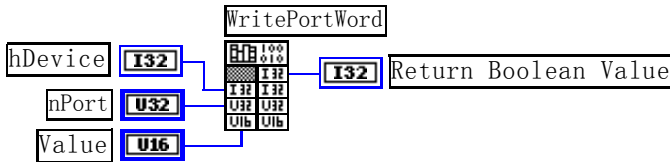
Visual Basic:

Declare Function WritePortWord Lib "ART2932" (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 'ART2932' 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GetLastError捕获当前错误码。

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

◆ 以四字节(32Bit)方式写 I/O 端口

Visual C++ & C++ Builder:

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

Visual Basic:

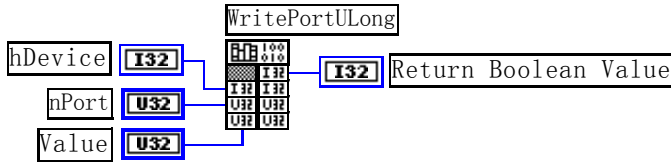
Declare Function WritePortULong Lib "ART2932" (ByVal hDevice As Long, _
 ByVal nPort As Long, _
 ByVal Value As Long) As Boolean

Delphi:

Function WritePortULong(hDevice : Integer;

nPort : LongWord;
 Value : LongWord) : Boolean;
 StdCall; External 'ART2932' 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GetLastError捕获当前错误码。

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

◆ 以单字节(8Bit)方式读 I/O 端口

Visual C++ & C++ Builder:

BYTE ReadPortByte(HANDLE hDevice,
 UINT nPort)

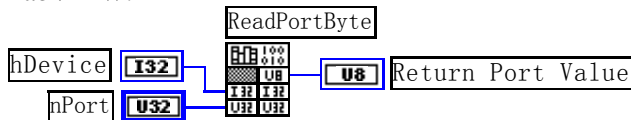
Visual Basic:

Declare Function ReadPortByte Lib "ART2932" (ByVal hDevice As Long, _
 ByVal nPort As Long) As Byte

Delphi:

Function ReadPortByte(hDevice : Integer;
 nPort : LongWord) : Byte;
 StdCall; External 'ART2932' 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

Visual C++ & C++ Builder:

WORD ReadPortWord(HANDLE hDevice,
 UINT nPort)

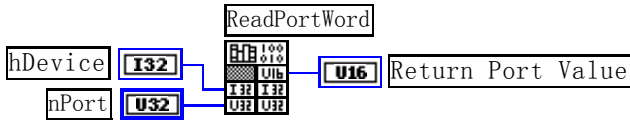
Visual Basic:

Declare Function ReadPortWord Lib "ART2932" (ByVal hDevice As Long, _
 ByVal nPort As Long) As Integer

Delphi:

Function ReadPortWord(hDevice : Integer;
 nPort : LongWord) : Word;
 StdCall; External 'ART2932' 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

Visual C++ & C++ Builder:

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

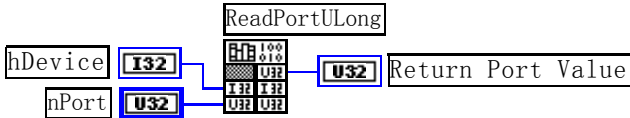
Visual Basic:

```
Declare Function ReadPortULong Lib "ART2932" ( ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                      nPort : LongWord) : LongWord;
StdCall; External 'ART2932' 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](#)

◆ 获取设备固件及程序版本

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function GetDevVersion Lib "ART2932" (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 'ART2932' Name 'GetDevVersion';
```

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 固件版本号。
pulDriverVersion 驱动版本号。
返回值: 返回由版本号。
相关函数: [CreateDevice](#) [ReleaseDevice](#)

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 "ART2932" (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 'ART2932' 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 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:\\\\ART2932\\\\Data.Dat”, 在 Basic 中, 其语法格式如: “C:\\ART2932\\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 |
|-----------------------|--------|------------------------------------|
| ART2932_modeRead | 0x0000 | 只读文件方式 |
| ART2932_modeWrite | 0x0001 | 只写文件方式 |
| ART2932_modeReadWrite | 0x0002 | 既读又写文件方式 |
| ART2932_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 |
| ART2932_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](#) [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 "ART2932" (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 'ART2932' 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 "ART2932" (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 'ART2932' 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 "ART2932" (ByVal hFileObject As Long,
 ByVal nOffsetBytes As Long) As Boolean

Delphi:

Function SetFileOffset (hFileObject : Integer;
 nOffsetBytes : LongWord) : Boolean;
Stdcall; external 'ART2932' 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 "ART2932" (ByVal hFileObject As Long) As Long

Delphi:

Function GetFileLength (hFileObject : Integer) : LongWord;
Stdcall; external 'ART2932' 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 "ART2932" (ByVal hFileObject As Long) As Boolean

Delphi:

Function ReleaseFile(hFileObject : Integer) : Boolean;

Stdcall; external 'ART2932' 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

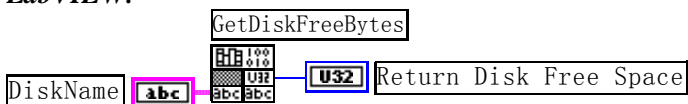
Visual C++ & C++ Builder:

ULONGLONG GetDiskFreeBytes(LPCTSTR DiskName)

Visual Basic:

Declare Function GetDiskFreeBytes Lib "ART2932" (ByVal DiskName 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: 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 GetLastError 捕获错误码。注意使用 64 位整型变量。