ART2753 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

第一章 版权信息与命名约定

第一节、版权信息

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 ARTxxxx_ 则被省略。如 ART2753_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InitDeviceProAD、ReadDeviceProAD_Half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_Half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StartDeviceProAD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 2.1.1。

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。

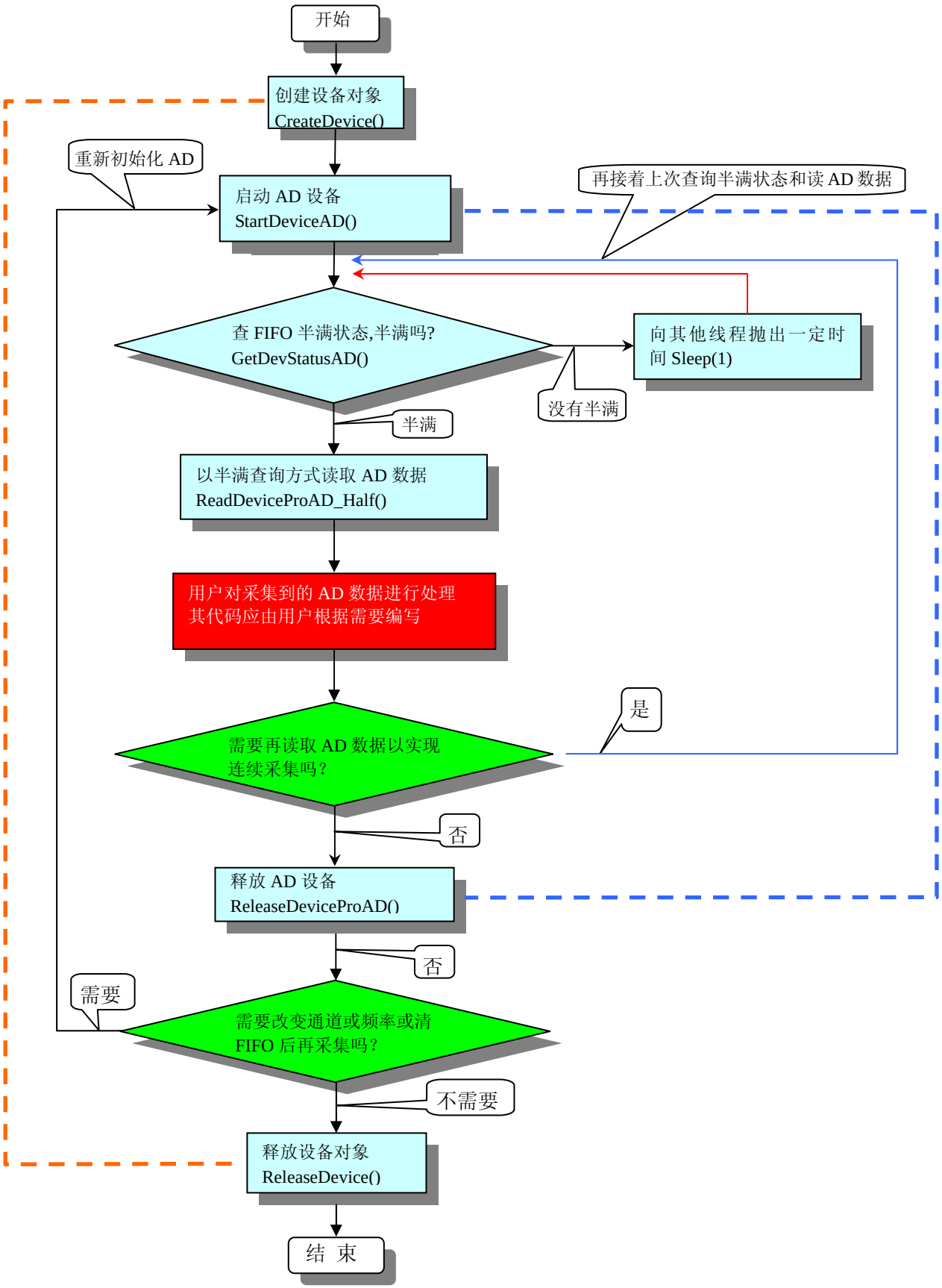


图 2.1.1 半满查询方式 AD 采集过程

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。WritePortByte，WritePortWord，WritePortULong，ReadPortByte， ReadPortWord，ReadPortULong函数对用户来讲，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 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“ART2753_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建设备对象(用设备基地址)	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，且释放总线设备对象	上层及底层用户
② AD 采样操作函数		
InitDeviceProAD	初始化设备 AD 部件，准备传数	上层用户
SetDevFrequencyAD	可动态改变 AD 采样频率	上层用户
SetDevPostTrigAD	设置触发模式	上层用户
StartDeviceAD	启动 AD 设备，开始转换	上层用户
StopDeviceAD	暂停 AD 设备	上层用户
GetDevStatusAD	取得各种状态	上层用户
GetSecondPlanetNum	获得可见和跟踪卫星数	上层用户
ReadDeviceProAD_Half	连续批量读取设备上的 AD 数据	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	上层用户
SaveParaAD	往 Windows 系统保存硬件参数	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	上层用户
SaveBaseAddr	将基地址保存至系统中	上层用户

使用需知

Visual C++ & C++Builder：

首先将 ART2753.h 和 ART2753.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(ART2753.lib)加入到您的工程中。

#include “ART2753.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 ART2753.h 和 ART2753.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 ART2753.Bas 驱动模块头文件从 VB 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令,在弹出的对话中选择 ART2753.Bas 模块文件即可，一旦完成以上工作后，那么使用设备的驱动程序接口就跟使用 VB 自身的各种函数，其方法一样简单，毫无二别。

请注意，因考虑 Visual C++和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所

举的 Visual Basic 程序均是需要编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不保证能完全顺利运行。

Delphi:

首先将 ART2753.Pas 驱动模块头文件从 Delphi 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单,执行其中的"Project Manager"命令,在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 ART2753.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：“ART2753”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ART2753;  // 注意： 在此加入驱动程序接口单元 ART2753
```

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)
```

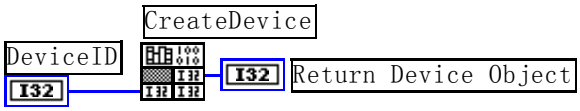
Visual Basic：

```
Declare Function CreateDevice Lib "ART2753" (ByVal wBaseAddress As Integer) As Long
```

Delphi：

```
Function CreateDevice(wBaseAddress :Word):Integer;
                    StdCall; External 'ART2753' Name 'CreateDevice';
```

LabView:



功能：该函数使用基地址创建设备对象，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 基地址。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice ReleaseDevice

Visual C++ & C++Builder 程序举例:

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

Visual Basic 程序举例:

```
    :
```

```
Dim hDevice As Long   ' 定义设备对象句柄
hDevice = CreateDevice (BaseAddress )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效

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

◆ **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 "ART2753" (ByVal hDevice As Long) As Integer

Delphi:

Function GetDeviceCurrentBaseAddr (hDevice : Integer): Word;
StdCall; External 'ART2753' 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)

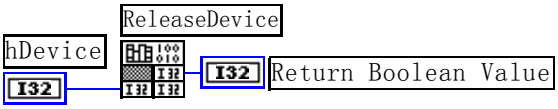
Visual Basic:

Declare Function ReleaseDevice Lib "ART2753" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDevice(hDevice : Longint):Boolean;
StdCall; External 'ART2753' Name 'ReleaseDevice';

LabView:



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回 TRUE，否则返回 FALSE，用户可以用 GetLastError 捕获错误码。

相关函数：CreateDevice

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

第三节、AD 采样操作函数原型说明

◆ **初始化设备对象**

函数原型：

Visual C++ & C++Builder：

BOOL InitDeviceProAD(HANDLE hDevice,
PART2753_PARA_AD pADPara)

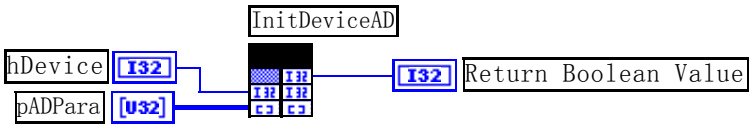
Visual Basic:

Declare Function InitDeviceProAD Lib "ART2753" (ByVal hDevice As Long, _
ByRef pADPara As ART2753_PARA _AD) As Boolean

Delphi:

```
Function InitDeviceProAD (hDevice : Integer;
                          pADPara:PART2753_PARA_AD):Boolean;
                          StdCall; External 'ART2753' Name ' InitDeviceProAD ';
```

LabView:



功能：它负责初始化设备对象中的AD部件,为设备操作就绪有关工作,如预置AD采集通道,采样频率等,然后启动AD设备开始AD采集，随后，用户便可以连续调用ReadDeviceAD读取设备上的AD数据以实现连续采集。注意：每次在InitDeviceProAD之后所采集的所有数据，其第一个点是无效的，必须丢掉，有效数据从第二个点开始。

参数：

hDevice 设备对象句柄,它应由设备的CreateDevice创建。

pADPara 设备对象参数结构，它决定了设备对象的各种状态及工作方式,如AD采样通道、采样频率等。请参考《AD硬件参数介绍》。

返回值：如果初始化设备对象成功,则返回 TRUE。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 ReadDeviceAD ReleaseDevice

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

- **动态改变采样频率**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevFrequencyAD (HANDLE hDevice,
                        LONG Frequency)
```

Visual Basic:

```
Declare Function SetDevFrequencyAD Lib "ART2753" (ByVal hDevice as Long, _
                                                  ByVal Frequency As Long) As Boolean
```

Delphi:

```
Function SetDevFrequencyAD (hDevice : Integer;
                            Frequency: LongInt):Boolean;
                            StdCall; External 'ART2753' Name ' SetDevFrequencyAD ';
```

LabVIEW：

请参考演示源程序。

功能：在 AD 采样过程中，可动态改变采样频率。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

Frequency 采样频率，范围为 1Hz—250kHz。

返回值：如果调用成功，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

- **设置触发模式**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevPostTrigAD (HANDLE hDevice,
                       LONG lTriMode)
```

Visual Basic:

```
Declare Function SetDevTrigModeAD Lib "ART2753" (ByVal hDevice as Long, _
                                                 ByVal lTriMode As Long) As Boolean
```

Delphi:

```
Function SetDevPostTrigAD (hDevice : Integer;
                           lTriMode: LongInt):Boolean;
                           StdCall; External 'ART2753' Name ' SetDevPostTrigAD ';
```

LabVIEW：

请参考演示源程序。

功能：在 AD 采样过程中，设置触发模式。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

lTriMode 触发模式。=00：内触发，=01：外触发，=02：GPS。

返回值：如果调用成功，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

♦ **启动 AD 设备**

函数原型：

Visual C++ & C++Builder:

```
BOOL StartDeviceAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceADLib "ART2753" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function StartDeviceAD (hDevice : Integer): Boolean;
                        StdCall; External 'ART2753' 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。调用该函数后它可能立即启动，这就要取决您选择的触发方式或触发源。

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 准备就绪，等待触发事件的到来就开始实际的 AD 输入，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

♦ **暂停 AD 设备**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StopDeviceAD Lib "ART2753" (ByVal hDevice as Long) As Boolean
```

Delphi:

```
Function StopDeviceAD (hDevice : Integer) : Boolean;
                       StdCall; External 'ART2753' Name ' StopDeviceAD';
```

LabVIEW：

请参考相关演示程序。

功能：暂停 AD 设备。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工作参数。

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 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

♦ **取得 AD 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDevStatusAD (HANDLE hDevice,
                     PART2753_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "ART2753" (ByVal hDevice As Long,_
                                               ByRef pADStatus As ART2753_STATUS_ AD) As Boolean
```

Delphi:

```
Function GetDevStatusAD (hDevice : Integer;
                         pADStatus: PART2753_STATUS_ AD):Boolean;
                         StdCall; External 'ART2753' 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（bConverting），是否被触发(bTrigger)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 StartDeviceAD

♦ **获得可见和跟踪卫星数**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetSecondPlanetNum(HANDLE hDevice,
                        PWORD pVisibiNum,
                        PWORD pScoutNum)
```

Visual Basic:

```
Declare Function GetSecondPlanetNum Lib "ART2753" (ByVal hDevice As Long,_
                                                   ByRef pVisibiNum As Integer,_
                                                   ByRef pScoutNum As Integer) As Boolean
```

Delphi:

```
Function GetSecondPlanetNum(hDevice : Integer;
                            pVisibiNum : Pointer;
                            pScoutNum : Pointer):Boolean;
                            StdCall; External 'ART2753' Name ' GetSecondPlanetNum ';
```

LabVIEW:

请参考相关演示程序。

功能：获得可见和跟踪卫星数。

hDevice 设备对象句柄，它应由CreateDevice创建。

pVisibiNum 获得可见卫星数。

pScoutNum 获得跟踪卫星数。

返回值：如果调用成功，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

♦ **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,
                          SHORT ADBuffer[],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "ART2753" (ByVal hDevice As Long, _
```

```
                                        ByRef ADBuffer As Integer,_
                                        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 'ART2753' 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。

♦ 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 "ART2753" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDeviceProAD(hDevice : Integer) : Boolean;
                            StdCall; External 'ART2753' Name ' ReleaseDeviceProAD ';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的 AD 部件。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

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

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

相关函数: CreateDevice　InitDeviceAD　ReleaseDeviceAD　ReleaseDevice

❖ 以上函数调用一般顺序

半满查询方式：

① CreateDevice
② InitDeviceProAD
③ StartDeviceAD
④ GetDevStatusAD
⑤ ReadDeviceProAD_Half

⑥ StopDeviceAD
⑦ ReleaseDeviceProAD
⑧ ReleaseDevice

用户可以反复执行第⑤步，以实现高速连续不间断数据采集。如果在采集过程中要改变设备状态信息，如采样通道等，则执行到第⑥步后再回到第②步用新的状态信息重新初始设备。

注意在第⑤步中，若其ReadDeviceProAD_Half函数成功返回，且nRetSizeWords参数值等于 0，则需要重新执行第⑤步，直到不等于 0 为止。

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,
                PART2753_PARA_AD pADPara)
```

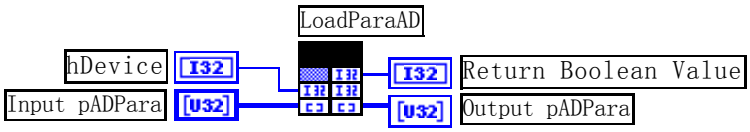
Visual Basic:

```
Declare Function LoadParaAD Lib "ART2753" (ByVal hDevice As Long, _
                                           ByRef pADPara As ART2753_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD(hDevice : Integer;
                    pADPara:PART2753_PARA_AD):Boolean;
                    StdCall; External 'ART2753' Name 'LoadParaAD';
```

LabView:



功能：负责从 Windows 系统中读取设备硬件参数。

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara 属于 PART2753_PARA 的结构指针型，它负责返回硬件参数值，关于结构指针类型 PART2753_PARA 请参考相应 ART2753.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:

```
        :
ART2753_PARA_AD ADPara;
HANDLE   hDevice;
WORD BaseAddr;
hDevice = CreateDevice(BaseAddr) ;      // 管理秘一个设备
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 ART2753_PARA_AD
Dim hDevice As Long   :
 WORD BaseAddr;
hDevice = CreateDevice(BaseAddr) ;      // 管理秘一个设备
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,
                PART2753_PARA_AD pADPara)
```

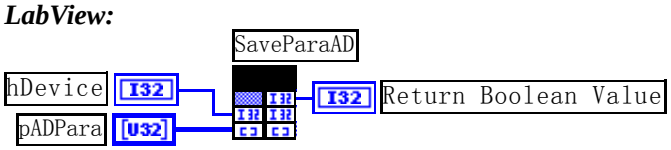
Visual Basic:

```
Declare Function SaveParaAD Lib "ART2753" (ByVal hDevice As Long, _
                                           ByRef pADPara As ART2753_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
                     pADPara:PART2753_PARA_AD):Boolean;
                     StdCall; External 'ART2753' 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,
                  PART2753_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "ART2753" (ByVal hDevice As Long, _
                                            ByRef pADPara As ART2753_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD ( hDevice : Integer;
                       pADPara : PART2753_PARA_AD) : Boolean;
                       StdCall; External 'ART2753' 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ART2753_PARA_AD的详细介绍请参考ART2753.h或ART2753.Bas或ART2753.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 "ART2753" (ByVal hDevice As Long, _
```

ByRef pBaseAddr As Integer) As Boolean

Delphi:

```
Function LoadBaseAddr ( hDevice : Integer;
                        pBaseAddr: Pointer) : Boolean;
                        StdCall; External 'ART2753' 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：

Viusal C++ & C++ Builder:

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

Visual Basic:

```
Declare Function SaveBaseAddr Lib "ART2753" (ByVal hDevice As Long, _
                                             ByRef wBaseAddr As Integer) As Boolean
```

Delphi:

```
Function SaveBaseAddr ( hDevice : Integer;
                        wBaseAddr: Word) : Boolean;
                        StdCall; External 'ART2753' 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（ART2753_PARA_AD）

Visual C++ & C++Builder:

```
typedef struct _ART2753_PARA_AD        // 板卡各参数值
{
    LONG FirstChannel;          // 首通道,取值范围为[0, 11]
    LONG LastChannel;           // 末通道,取值范围为[0, 11]
    LONG Frequency;             // 采集频率,单位为 Hz，[1, 250000]
    LONG TriggerMode;           // 触发模式选择(软件触发、后触发)
} ART2753_PARA_AD, *PART2753_PARA_AD;
```

Visual Basic：

```
Private Type ART2753_PARA_AD
    FirstChannel As Long        ' 首通道,取值范围为[0, 11]
    LastChannel As Long         ' 末通道,取值范围为[0, 11]
    Frequency As Long           ' 采样频率，单位 Hz
```

```
    TriggerMode As LongInt      ' 触发模式选择
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PART2753_PARA_AD = ^ART2753_PARA_AD;   // 指针类型结构
    ART2753_PARA_AD = record          // 标记为记录型
        FirstChannel : LongInt;       // 首通道,取值范围为[0, 11]
        LastChannel : LongInt;        // 末通道,取值范围为[0, 11]
        Frequency : LongInt;          // 采集频率,单位为 Hz
        TriggerMode : LongInt;        // 触发模式选择(软件触发、后触发)
End;
```

LabView:

请参考相关演示程序。

首先请您关注一下这个结构与前面 ISA 总线部分中的硬件参数结构 PARA 比较，该结构实在太简短了。其原因就是在于设备是系统全自动管理的设备，什么端口地址，中断号，DMA 等将与设备的用户永远告别，一句话设备简单得就象使用电源插头一样。

硬件参数说明：此结构主要用于设定设备硬件参数值，用这个参数结构对设备进行硬件配置完全由InitDeviceProAD函数完成。

FirstChannel 首通道值，取值范围应根据设备的总通道数设定，本设备的 AD 采样首通道取值范围为 0～11，要求首通道等于或小于末通道。

LastChannel 末通道值，取值范围应根据设备的总通道数设定，本设备的 AD 采样首通道取值范围为 0～11，要求末通道大于或等于首通道。

注：当首通道和末通道相等时，即为单通道采集。

Frequency 它指各个通道的采样频率，单位 Hz，最大频率可为 250KHz，但其最小值不能小于 1Hz。

TriggerMode AD 触发模式。

常量名	常量值	功能定义
ART2753_TRIGMODE_SOFT	0x0000	内触发
ART2753_TRIGMODE_POST	0x0001	外触发
ART2753_TRIGMODE_GPS	0x0002	GPS 同步触发

相关函数: InitDeviceProAD LoadParaAD SaveParaAD

第二节、AD 状态参数结构（ART2753_STATUS_AD）

Visual C++ & C++Builder：

```
typedef struct _ART2753_STATUS_AD
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bOverFull;
} ART2753_STATUS_AD, *PART2753_STATUS_AD;
```

Visual Basic：

```
Private Type   ART2753_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bOverFull As Long
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
    PART2753_STATUS_AD = ^ ART2753_STATUS_AD;   // 指针类型结构
    ART2753_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。

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，即有满数据可读，否则表示在满以下，可能有小于满的数据可读。

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。

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 原始数据 LSB 转换成电压值 Volt 的换算方法

在换算过程中弄清模板精度（即 Bit 位数）是很关键的，因为它决定了 LSB 数码的总宽度 CountLSB。比如 12 位的模板 CountLSB 为 4096。其他类型同理均按 2^n=LSB 总数（n 为 Bit 位数）换算即可。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000	Volt = (20000.00/65536) * ((ADBuffer[0] ^0x8000)&0xFFFF) – 10000.00	[-10000.00，+ 9999.69]
±5000	Volt = (10000.00/65536) * ((ADBuffer[0] ^0x8000)&0xFFFF) – 5000.00	[-5000.00，+ 4998.84]

换算举例：（设量程为±10000mV, 这里只转换第一个点）

Visual C++&C++Builder：

```
USHORT Lsb; // 定义存放标准 LSB 原码的变量
float Volt;    // 定义存放转换后的电压值的变量
float PerLsbVolt = 20000.00/65536; // 求出每个 LSB 原码单位电压值
Lsb = (ADBuffer[0] ^0x8000)&0xFFFF;
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 = 20000.00/65536   ' 求出每个 LSB 原码单位电压值
Lsb = (ADBuffer(0) XOR 32768) AND 65535    ' 将其转换成无符号 16 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 := 20000.00/65536;   // 求出每个 LSB 原码单位电压值
Lsb := (ADBuffer[0] XOR $8000) AND $FFFF;
Volt ： = PerLsbVolt * Lsb-10000.00; // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

当首末通道相等时，即为单通道采集，假如FirstChannel=5，LastChannel=5,其排放规则如下：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	…

两通道采集(CH0 – CH1)

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	…

四通道采集(CH0 - CH3)

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	…

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取AD数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐问题，尤其任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长（在设备上同时也应 32 的整数倍），这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个AD通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n为每个通道的点数)的长度采集。为了更加详细地说明问题民，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如ReadDeviceAD函数读回，即便不考虑是否能一次读完的问题，但对用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

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

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

第六章 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“ART2753_”）

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	
DelayTimeUs	微秒级延时函数	
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、I0 端口读写函数原型说明

注意：若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

◆ **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

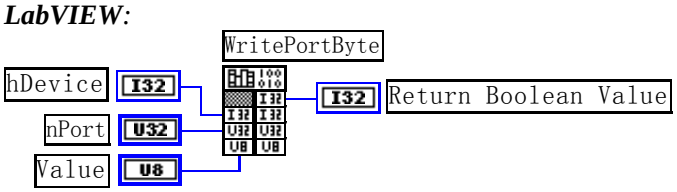
Visual Basic:

```
Declare Function WritePortByte Lib "ART2753" ( 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 'ART2753' 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)
```

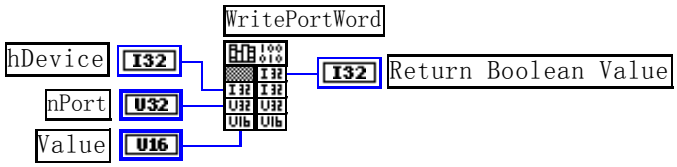
Visual Basic:

```
Declare Function WritePortWord Lib "ART2753" (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 'ART2753' 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)
```

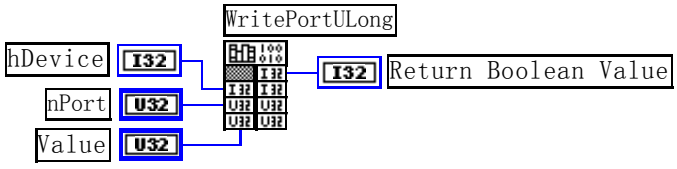
Visual Basic:

```
Declare Function WritePortULong Lib "ART2753" (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 'ART2753' 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)
```

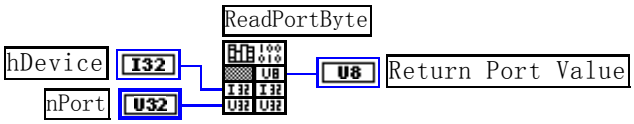
Visual Basic:

```
Declare Function ReadPortByte Lib "ART2753" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

```
Function ReadPortByte(hDevice : Integer;
                      nPort : LongWord) : Byte;
                      StdCall; External 'ART2753' 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)
```

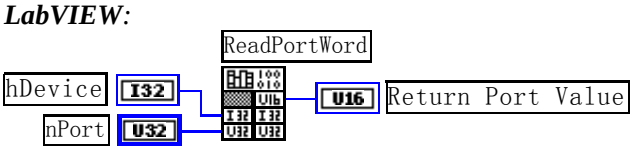
Visual Basic:

```
Declare Function ReadPortWord Lib "ART2753" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

Delphi:

```
Function ReadPortWord(hDevice : Integer;
                      nPort : LongWord) : Word;
                      StdCall; External 'ART2753' 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)
```

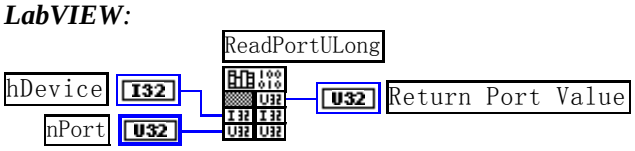
Visual Basic:

```
Declare Function ReadPortULong Lib "ART2753" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                       nPort : LongWord) : LongWord;
                       StdCall; External 'ART2753' 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 "ART2753" (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 ART2753.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 "ART2753" (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)

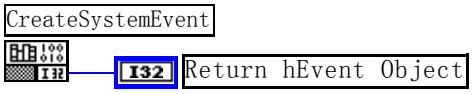
Visual Basic:

Declare Function CreateSystemEvent Lib " ART2753 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'ART2753' 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 " ART2753 " (ByVal hEvent As Long) As Boolean
```

Delphi:

```
Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
                                    StdCall; External 'ART2753' 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 "ART2753" (ByVal hDevice As Long, _
                                            ByVal nTimeUs As Long) As Boolean
```

Delphi:

```
Function DelayTimeUs (hDevice: Integer;
                      nTimeUs : LongInt) : Boolean;
                      StdCall; External 'ART2753' 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。

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 "ART2753" (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 'ART2753' 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
ART2753_modeRead	0x0000	只读文件方式
ART2753_modeWrite	0x0001	只写文件方式
ART2753_modeReadWrite	0x0002	既读又写文件方式
ART2753_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
ART2753_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 "ART2753" ( ByVal hFileObject As Long,_
                                           ByRef pDataBuffer As Integer,_
                                           ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongWord) : Boolean;
                   Stdcall; external 'ART2753' 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 hFileObject,
               PVOID pDataBuffer,
               ULONG OffsetBytes,
               ULONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "ART2753" (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 'ART2753' 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 "ART2753" (ByVal hFileObject As Long,_
ByVal nOffsetBytes As Long) As Boolean
Delphi:
Function SetFileOffset (hFileObject : Integer;
nOffsetBytes : LongWord) : Boolean;
Stdcall; external 'ART2753' 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 "ART2753" (ByVal hFileObject As Long) As Long
Delphi:
Function GetFileLength (hFileObject : Integer) : LongWord;
Stdcall; external 'ART2753' 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

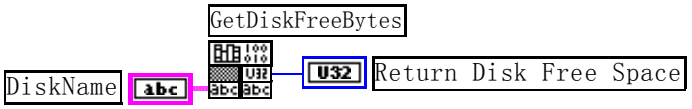
- **释放设备文件对象**

函数原型：
Visual C++ & C++ Builder:

BOOL ReleaseFile(HANDLE hFileObject)
Visual Basic:
Declare Function ReleaseFile Lib "ART2753" (ByVal hFileObject As Long) As Boolean
Delphi:
Function ReleaseFile(hFileObject : Integer) : Boolean;
Stdcall; external 'ART2753' 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 "ART2753" (ByVal DiskName As String) As Currency
Delphi:
Function GetDiskFreeBytes (DiskName: String) : Currency;
Stdcall; external 'ART2753' 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。