ART2153 Windows2000/XP 驱动程序使用说明书

请您务必阅读《使用纲要》，他会使您事半功倍!

目　　录

提醒用户：

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

有关设备驱动安装和产品二次发行请参考 ART2153Inst.doc 文档。

第一章 版权信息

第二章 使用纲要

一、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控制权。然后将此句柄作为参数传递给其他函数，如InitDeviceAD可以使用hDevice句柄以初始化设备的AD部件，StartDeviceAD启动AD设备，ReadDeviceProAD_Npt (或ReadDeviceProAD_Half)函数可以用hDevice句柄实现对AD数据的采样批量读取等。最后可以通过ReleaseDevice将hDevice释放掉。

二、如何批量取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceAD函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化，然后这个函数启动AD设备。接着便可用ReadDeviceProAD_Npt (或ReadDeviceProAD_Half)反复读取AD数据以实现连续不间断采样。当您需要关闭AD设备时，ReleaseDeviceAD便可帮您实现（但设备对象hDevice依然存在）。具体执行流程请看下面的图 2.1.1 和图 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。

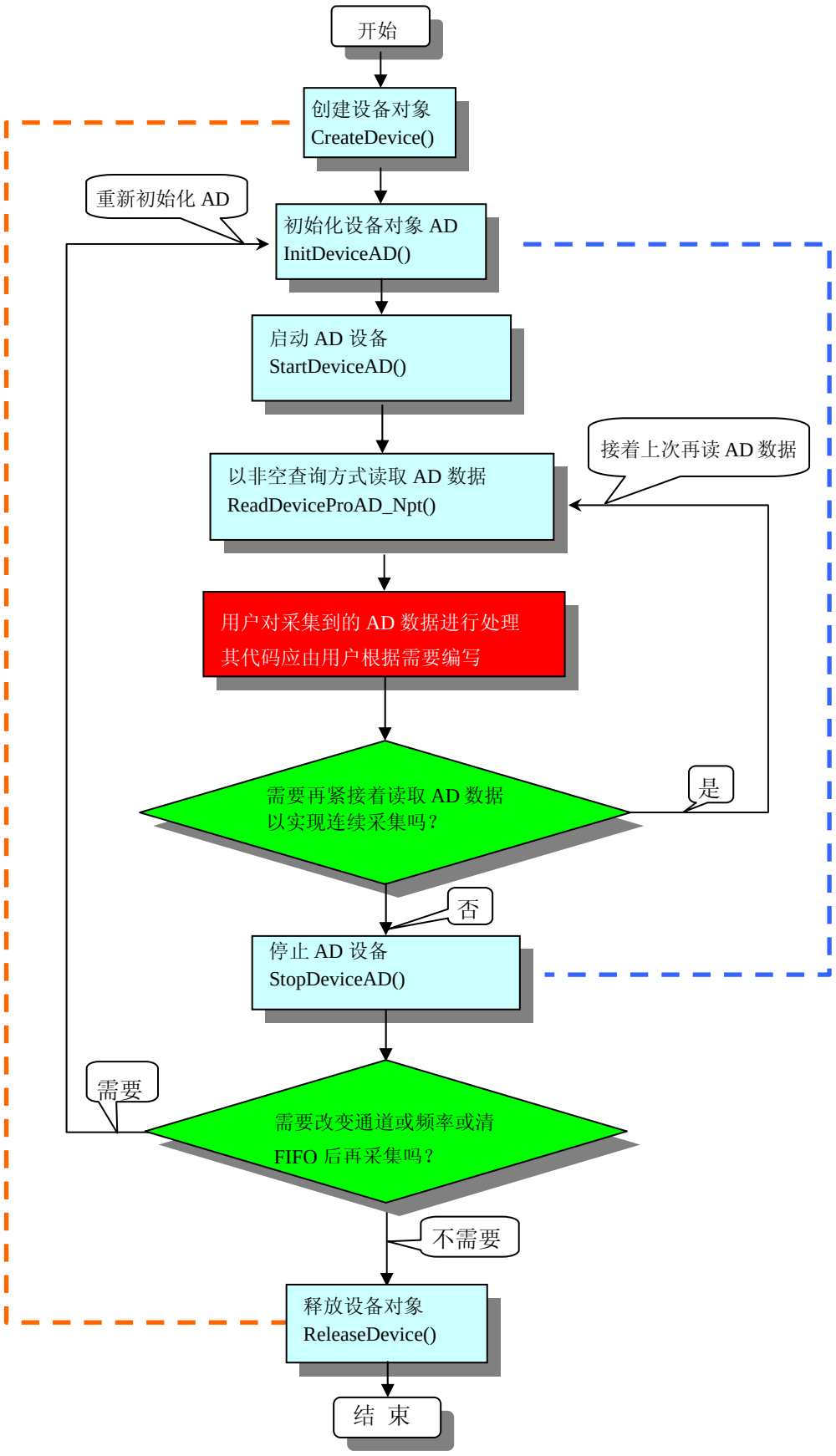


图 2.1.1 非空查询方式 AD 采集过程

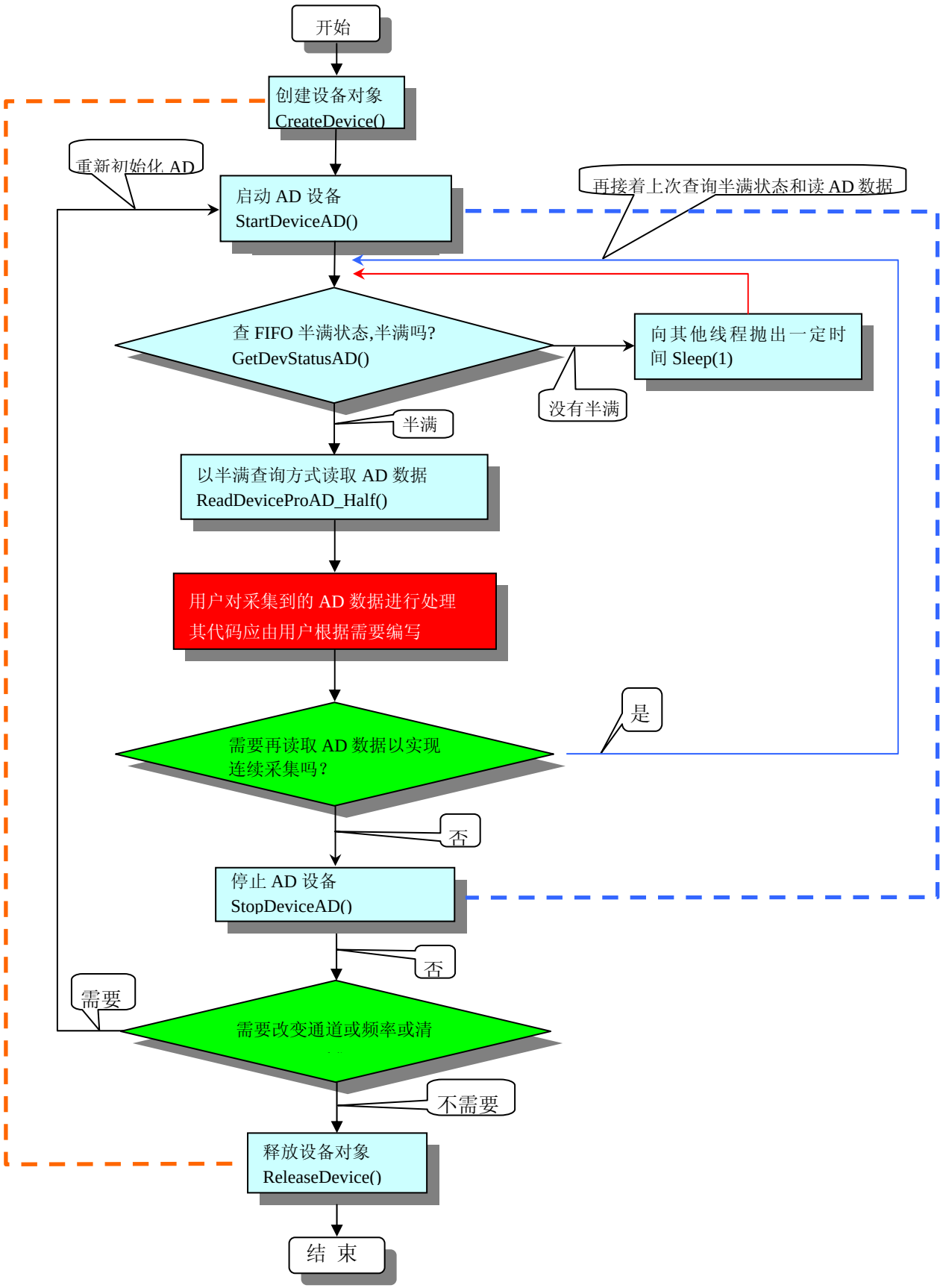


图 2.1.2 半满查询方式 AD 采集过程

三、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　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“ART2153_”）

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)	
ReleaseDevice	关闭设备，且释放总线设备对象	
② AD 采样操作函数		
InitDeviceAD	初始化设备 AD 部件，准备传数	
SetDevFrequencyAD	可动态改变 AD 采样频率	
StartDeviceAD	启动 AD 设备，开始转换	
StopDeviceAD	暂停 AD 设备	
GetDevStatusAD	取得各种状态	
ReadDeviceProAD_Npt	连续读取当前设备上的 AD 数据	
ReadDeviceProAD_Half	连续批量读取设备上的 AD 数据	
③ 辅助函数（硬件参数设置、保存、读取函数）		
LoadParaAD	从 Windows 系统中读取硬件参数	
SaveParaAD	往 Windows 系统保存硬件参数	
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
LoadBaseAddr	将基地址从系统中读出	
SaveBaseAddr	将基地址保存至系统中	

使用需知

Visual C++ & C++Builder：

首先将 ART2153.h 和 ART2153.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(ART2153.lib)加入到您的工程中。

#include “ART2153.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 ART2153.h 和 ART2153.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 ART2153.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 ART2153.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 ART2153.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 ART2153.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：“ART2153”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ART2153;  // 注意： 在此加入驱动程序接口单元 ART2153
```

LabView / CVI:

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的驱动程序接口的详细说明请参考其演示源程序。

第二节、设备对象管理函数原型说明

◆ 创建设备对象函数

函数原型：

Visual C++ & C++ Builder：

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

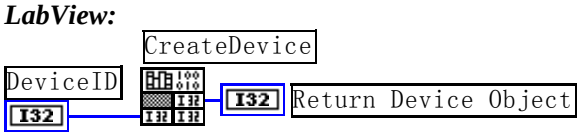
Visual Basic：

```
Declare Function CreateDevice Lib "ART2153" (ByVal BaseAddress As Integer) As Long
```

Delphi：

```
Function CreateDevice(BaseAddress :Word):Integer; StdCall; External 'ART2153' 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：BaseAddress 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
            :
```

◆ **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)

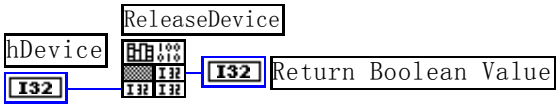
Visual Basic:

Declare Function ReleaseDevice Lib "ART2153" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDevice(hDevice : Longint):Boolean; StdCall; External 'ART2153' 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 InitDeviceAD( HANDLE hDevice,
                   PART2153_PARA_AD pADPara )
```

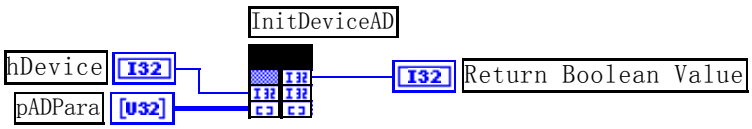
Visual Basic:

```
Declare Function InitDeviceAD Lib "ART2153" (ByVal hDevice As Long, _
                                             ByRef pADPara As ART2153_PARA _AD) As Boolean
```

Delphi:

```
Function InitDeviceAD( hDevice : Integer; pADPara:PART2153_PARA_AD):Boolean;
                       StdCall; External 'ART2153' Name 'InitDeviceAD';
```

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InitDeviceAD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 "ART2153" (ByVal hDevice as Long, _
                                                  ByVal Frequency As Long) As Boolean
```

Delphi:

```
Function SetDevFrequencyAD (hDevice : Integer;
                            Frequency: LongInt):Boolean;
                            StdCall; External 'ART2153' 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

♦ **启动 AD 设备**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StartDeviceADLib "ART2153" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function StartDeviceAD (hDevice : Integer): Boolean;
                        StdCall; External 'ART2153' Name ' StartDeviceAD';
```

LabVIEW：

请参考相关演示程序。

功能：启动AD设备，它必须在调用InitDeviceAD后才能调用此函数。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 "ART2153" (ByVal hDevice as Long) As Boolean
```

Delphi:

```
Function StopDeviceAD (hDevice : Integer) : Boolean;
                       StdCall; External 'ART2153' 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,
                     PART2153_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "ART2153" (ByVal hDevice As Long,_
                                               ByRef 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（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

♦当 FIFO 非空信号有效时，读取设备上的 AD 数据

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ReadDeviceProAD_Npt Lib "ART2153" (
                          ByVal hDevice As Long, _
                          ByRef ADBuffer As Integer,_
                          ByVal nReadSizeWords As Long,_
                          ByRef nRetSizeWords As Long) As Long
```

Delphi:

```
Function ReadDeviceProAD_Npt(hDevice : Integer;
                          ADBuffer : Pointer;
                          nReadSizeWords : LongInt;
                          nRetSizeWords : Pointer) : LongInt;
                          StdCall; External 'ART2153' 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 用户数据缓冲区地址。接受的是从设备上采集的LSB原码数据，关于如何将LSB原码数据转换成电压值，请参考《数据格式转换与排列规则》章节。

nReadSizeWords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：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 InitDeviceAD ReleaseDevice

♦ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型:

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "ART2153" (ByVal hDevice As Long, _
                                                     ByRef ADBuffer As Integer,_
                                                     ByVal nReadSizeWords As Long ,_
                                                     ByRef nRetSizeWords As Long) As Long
```

Delphi:

```
Function ReadDeviceProAD_Half(hDevice : Integer;
                              ADBuffer : Pointer;
                              nReadSizeWords : LongInt;
                              nRetSizeWords : Pointer) : LongInt;
                              StdCall; External 'ART2153' 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。

❖ **以上函数调用一般顺序**

非空查询方式:

① CreateDevice
② InitDeviceAD
③ StartDeviceAD
④ ReadDeviceProAD_Npt
⑤ StopDeviceAD
⑥ ReleaseDevice

用户可以反复执行第④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⑤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④步中，若其ReadDeviceProAD_Npt函数成功返回，且nRetSizeWords参数值等于 0，则需要重新执行第④步，直到不等于 0 为止。

半满查询方式:

① CreateDevice
② InitDeviceAD
③ StartDeviceAD
④ GetDevStatusAD
⑤ ReadDeviceProAD_Half
⑥ StopDeviceAD
⑦ 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, PART2153_PARA_AD pADPara)
```

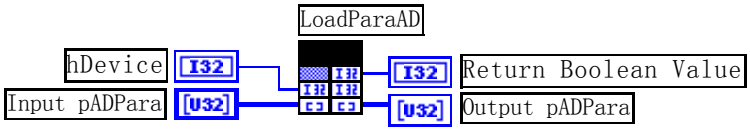
Visual Basic:

```
Declare Function LoadParaAD Lib "ART2153" (ByVal hDevice As Long, _
                                           ByRef pADPara As ART2153_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD(hDevice : Integer; pADPara:PART2153_PARA_AD):Boolean;
                    StdCall; External 'ART2153' 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 PART2153_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 PART2153_PARA 请参考相应 ART2153.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:

```
        :
ART2153_PARA_AD ADPara;
HANDLE   hDevice;
WORD BaseAddr;
hDevice = CreateDevice(BaseAddr) ;     // 管理秕一个设备
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 ART2153_PARA_AD
Dim hDevice As Long   :
 WORD BaseAddr;
hDevice = CreateDevice(BaseAddr) ;      // 管理秕一个设备
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：

Viusal C++ & C++Builder:

BOOL SaveParaAD(HANDLE hDevice, PART2153_PARA_AD pADPara)

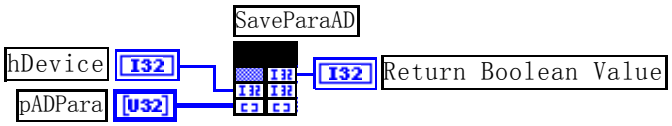
Visual Basic:

```
Declare Function SaveParaAD Lib "ART2153" (ByVal hDevice As Long, _
                                        ByRef pADPara As ART2153_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer; pADPara:PART2153_PARA_AD):Boolean;
                    StdCall; External 'ART2153' 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：

Viusal C++ & C++ Builder:

```
BOOL ResetParaAD (HANDLE hDevice,
                    PART2153_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "ART2153" (ByVal hDevice As Long, _
                                        ByRef pADPara As ART2153_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD ( hDevice : Integer;
                    pADPara : PART2153_PARA_AD) : Boolean;
                    StdCall; External 'ART2153' 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ART2153_PARA_AD的详细介绍请参考ART2153.h或ART2153.Bas或ART2153.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：

Viusal C++ & C++ Builder:

```
BOOL LoadBaseAddr(HANDLE hTDevice, PWORD pBaseAddr)
```

Visual Basic:

```
Declare Function LoadBaseAddr Lib "ART2153" (ByVal hDevice As Long, _
                                             ByRef pBaseAddr As Integer) As Boolean
```

Delphi:

```
Function LoadBaseAddr ( hDevice : Integer;
                        pBaseAddr: Pointer) : Boolean;
                        StdCall; External 'ART2153' 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 "ART2153" (ByVal hDevice As Long, _
                                             ByRef wBaseAddr As Integer) As Boolean
```

Delphi:

```
Function SaveBaseAddr ( hDevice : Integer;
                        wBaseAddr: Word) : Boolean;
                        StdCall; External 'ART2153' 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（ART2153_PARA_AD）

Visual C++ & C++Builder:

```
typedef struct _ART2153_PARA_AD        // 板卡各参数值
{
    LONG ADMode;              // AD 模式选择(同步/异步方式)
    LONG FirstChannel;        // 首通道,取值范围为[0, 31]
    LONG LastChannel;         // 末通道,取值范围为[0, 31]
    LONG Frequency;           // 采集频率,单位为 Hz
    LONG GroupInterval;       // 分组采样时的组间间隔(单位：微秒)
    LONG LoopsOfGroup;        // 组循环次数
    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(软件触发、后触发)
    LONG TriggerSource;       // 触发源选择
    LONG TriggerType;         // 触发类型选择(边沿触发/脉冲触发)
    LONG TriggerDir;          // 触发方向选择(正向/负向触发)
    LONG TrigWindow;          // 触发灵敏窗[1, 255]
    LONG TrigLevelVolt;       // 触发电平(0～10000mV)
    LONG ClockSource;         // 时钟源选择(内/外时钟源)
    LONG bClockOutput;        // 允许时钟输出
    LONG GroundingMode;       // 接地方式（单端或双端选择）
} ART2153_PARA_AD, *PART2153_PARA_AD;
```

Visual Basic：

```
Private Type ART2153_PARA_AD
    ADMode As Long             ' AD 模式选择(同步/异步方式)
    FirstChannel As Long       ' 首通道,取值范围为[0, 31]
    LastChannel As Long        ' 末通道,取值范围为[0, 31]
    Frequency As Long          ' 采样频率，单位 Hz
    GroupInterval As Long      ' 分组间隔，单位微秒
    LoopsOfGroup As LongInt    ' 组循环次数
    Gains As LongInt           ' 增益设置
    InputRange As LongInt      ' 模拟量输入量程范围
```

```
    TriggerMode As LongInt      ' 触发模式选择
    TriggerSource As Long       ' 触发源选择(内/外触发源)
    TriggerType As Long         ' 触发类型选择(边沿/电平触发类型)
    TriggerDir As Long          ' 触发方向选择(正向/负向触发方向)
    TrigWindow As Long          ' 触发灵敏窗[1, 255]
    TrigLevelVolt As Long       ' 触发电平(0～10000mV)
    ClockSource As Long         ' 时钟源选择(内时钟/外时钟源)
    bClockOutput As Long        ' 是否允许内时钟输出,要内部时钟输出则置 TRUE
    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
    PART2153_PARA_AD = ^ART2153_PARA_AD;   // 指针类型结构
    ART2153_PARA_AD = record          // 标记为记录型
        ADMode : LongInt;             // AD 模式选择(同步/异步方式)
        FirstChannel : LongInt;       // 首通道,取值范围为[0, 31]
        LastChannel : LongInt;        // 末通道,取值范围为[0, 31]
        Frequency : LongInt;          // 采集频率,单位为 Hz
        GroupInterval : LongInt;      // 分组采样时的组间间隔(单位：微秒)
        LoopsOfGroup : LongInt;       // 组循环次数
        Gains : LongInt;              // 增益设置
        InputRange : LongInt;         // 模拟量输入量程范围
        TriggerMode : LongInt;        // 触发模式选择(软件触发、后触发)
        TriggerSource : LongInt;      // 触发源选择
        TriggerType : LongInt;        // 触发类型选择(边沿触发/脉冲触发)
        TriggerDir : LongInt;         // 触发方向选择(正向/负向触发)
        TrigWindow: LongInt;          // 触发灵敏窗[1, 255]
        TrigLevelVolt: LongInt;       // 触发电平(0～10000mV)
        ClockSource : LongInt;        // 时钟源选择(内/外时钟源)
        bClockOutput : LongInt;       // 允许时钟输出
        GroundingMode : LongInt;      // 接地方式（单端或双端选择）
    End;
```

LabView:

首先请您关注一下这个结构与前面 ISA 总线部分中的硬件参数结构 PARA 比较，该结构实在太简短了。其原因就是在于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。

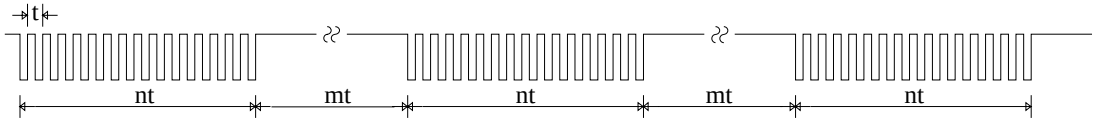
硬件参数说明：此结构主要用于设定设备硬件参数值，用这个参数结构对设备进行硬件配置完全由InitDeviceAD函数完成。

ADMode AD 采样模式。它的取值如下表：

常量名	常量值	功能定义
ART2153_ADMODE_SEQUENCE	0x0000	连续采集模式
ART2153_ADMODE_GROUP	0x0001	分组采集模式

连续采集方式的情况是：在由Frequency指定的采样频率下所采集的全部数据在时间轴上是等简隔的，比如

将Frequency指定为 100KHz, 即每隔 10 微秒采样一个点，总是这样重复下去。而分组采集方式的情况是：所有采集的数据点在时间轴上被划分成若干个等长的组，而组内通常有大于 2 个点的数据，组内各点频率由Frequency决定，组间间隔由GroupInterval决定。比如用户要求在对 0-15 通道共 16 个通道用 100KHz频率每采集一个轮回后，要求间隔 1 毫秒后，再对这 16 个通道采集下一个轮回，那么分组采集便是最佳方式，它可以将组间延时误差控制在 0.5 微秒以下。关于分组与连续采集更详细的说明请参考硬件说明书。如下图：



其中：t为所需触发A/D转换的周期Cycle，它由Frequency参数的倒数决定。Cycle = 1 / Frequency

n为每组的通道数ChannelCount决定，即ChannelCount = LastChannel-FirstChannel+1。

nt为每组操作所需时间即 Cycle * ChannelCount * LoopsofGroup (此处假定LoopsofGroup=1)

mt为每组操作之间所间隔的时间，它由GroupInterval参数决定，单位为1uS。

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

LastChannel 末通道值，取值范围应根据设备的总通道数设定，本设备的 AD 采样首通道取值范围为 0～31，要求末通道大于或等于首通道。

注：当首通道和末通道相等时，即为单通道采集。

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。

GroupInterval 分组间隔，指定两组间的时间间隔，单位微秒， 取值范围为[1, 65536]。

LoopsOfGroup 组循环次数(在分组采集时有效)，即指每一组采集从首通道依次采集到末通道后再回到首通道的次数，取值范围为[1, 65536]。比如=1 则表示从首通道采集到末通道后则自动进入分组间隔准备采集下一组。若=2，则表示从首通道采集到末通道后再回到首通道采集到末通道一遍后进入组间间隔，依此类推。

Gains 程控增益放大倍数，被采样的外界信号经通道开关选通后进入一个程控增益放大器，它可以将原始模拟信号放大指定倍数后再进入 AD 转换器被转换。

常量名	常量值	功能定义
ART2153_GAINS_1MULT	0x00	1 倍增益
ART2153_GAINS_2MULT	0x01	2 倍增益
ART2153_GAINS_4MULT	0x02	4 倍增益
ART2153_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
ART2153_INPUT_N10000_P10000mV	0x00	±10000mV
ART2153_INPUT_N5000_P5000mV	0x01	±5000mV
ART2153_INPUT_N2500_P2500mV	0x02	±2500mV
ART2153_INPUT_0_P10000mV	0x03	0～10000mV
ART2153_INPUT_0_P5000mV	0x04	0～5000mV
ART2153_INPUT_0_P2500mV	0x05	0～2500mV

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原始数据LSB转换成电压值Volt的换算方法》章节。

TriggerMode AD触发模式，若等于常量ART2153_TRIGMODE_SOFT则为内部软件触发，若等于常量ART2153_TRIGMODE_POST则为外部硬件后触发。两种方式的主要区别是：外触发是当设备被InitDeviceAD函数初始化就绪后，并没有立即启动AD采集，仅当外接管脚ATR上有一个符合要求的信号时，AD转换器便被启

动，且按用户预先设定的采样频率由板上的硬件定时器时定触发AD等间隔转换每一个AD数据，其触发条件由触发类型和触发方向及触发电平决定。

常量名	常量值	功能定义
ART2153_TRIGMODE_SOFT	0x0000	软件触发(属于内触发)
ART2153_TRIGMODE_POST	0x0001	硬件后触发(属于外触发)

TriggerSource AD 触发源选择。

常量名	常量值	功能定义
ART2153_TRIGSRC_ATR	0x0000	选择外部 ATR 作为触发源
ART2153_TRIGSRC_DTR	0x0001	选择外部 DTR 作为触发源

TriggerType AD 外触发方式使用信号类型。它的其选项值如下表：

常量名	常量值	功能定义
ART2153_TRIGTYPE_EDGE	0x0000	边沿触发
ART2153_TRIGTYPE_PULSE	0x0001	脉冲触发(电平方式),适用于采集馒头波信号

TriggerDir AD 外触发方式使用信号方向。它的其选项值如下表：

常量名	常量值	功能定义
ART2153_TRIGDIR_NEGATIVE	0x0000	负向触发(低脉冲/下降沿触发)
ART2153_TRIGDIR_POSITIVE	0x0001	正向触发(高脉冲/上升沿触发)
ART2153_TRIGDIR_POSIT_NEGAT	0x0002	正负向触发(高/低脉冲或上升/下降沿触发)

TrigWindow 触发灵敏窗时间值，取值范围为[1, 65535]，单位 25 纳秒。
TrigLevelVolt 触发电平(0～10000mV)。
ClockSource AD 外时钟选择：

常量名	常量值	功能定义
ART2153_CLOCKSRC_IN	0x0000	内部时钟
ART2153_CLOCKSRC_OUT	0x0001	外部时钟(CLKIN)

bClockOutput AD 允许时钟输出，为 TRUE 时允许输出到 CN1 上的 CLKOUT，为 FALSE 时禁止输出到 CN1 上的 CLKOUT。
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
ART2153_ GNDMODE_SE	0x00	单端方式(SE:Single end)
ART2153_ GNDMODE_DI	0x01	双端方式(DI:Differential)

相关函数: InitDeviceAD LoadParaAD SaveParaAD

第二节、AD 状态参数结构（ART2153_STATUS_AD）

Visual C++ & C++Builder：

```
typedef struct _ART2153_STATUS_AD
{
    LONG bNotEmpty;
```

```
    LONG bHalf;
    LONG bOverFull;
} ART2153_STATUS_AD, *PART2153_STATUS_AD;
```

Visual Basic：

```
Private Type    ART2153_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bOverFull As Long
End Type
```

Delphi:

```
Type    // 定义结构体数据类型
        PART2153_STATUS_AD = ^ ART2153_STATUS_AD;    // 指针类型结构
        ART2153_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]– 10000.00	[-10000.00，　+ 9999.69]
±5000	Volt =　(10000.00 / 65536) * ADBuffer[0]– 5000.00	[-5000.00，　+ 4998.84]
±2500	Volt =　(5000.00 / 65536) * ADBuffer[0]– 2500.00	[-2500.00，　+ 2499.92]
0～10000	Volt =　(10000.00 / 65536) * ADBuffer[0]	[0.00，　+ 9999.84]
0～5000	Volt =　(5000.00 / 65536) * ADBuffer[0]	[0.00，　+ 4999.92]

0～2500	Volt = (2500.00 / 65536) * ADBuffer[0]	[0.00， +2499.96]

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];
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) 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];
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“ART2153_”）

函数名	函数功能	备注
内存映射寄存器操作函数		
WriteRegisterWord	以字(16Bit)方式写寄存器端口	
ReadRegisterWord	以字(16Bit)方式读寄存器端口	

第二节、内存映射寄存器操作函数原型说明

◆ **以双字节（即 16 位）方式写内存映射寄存器的某个单元**

函数原型:

Visual C++ & C++ Builder:

```
BOOL WriteRegisterWord(HANDLE hDevice,
                       WORD Addr,
                       WORD Data)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "ART2153" (ByVal hDevice As Long, _
                                                  ByVal Addr As Integer, _
                                                  ByVal Data As Integer) As Boolean
```

Delphi:

```
Function WriteRegisterWord( hDevice : Integer;
                            Addr: Word;
                            Data: Word) : Boolean;
                            StdCall; External 'ART2153' Name ' WriteRegisterWord ';
```

LabVIEW:

请参见相关演示程序。

功能：以双字节（即 16 位）方式写内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

Addr　设备内存映射寄存器的线性基地址，它的值应由LoadBaseAddr 确定。

Data　相对于Addr线性基地址的偏移字节数，它与Addr两个参数共同确定WriteRegisterWord函数所访问的映射寄存器的内存单元。

返回值：无。

相关函数： CreateDevice　WriteRegisterWord　ReadRegisterWord　ReleaseDevice

◆ **以双字节（即 16 位）方式读内存映射寄存器的某个单元**

函数原型:

Visual C++ & C++ Builder:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       WORD Addr)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib "ART2153" (ByVal hDevice As Long, _
                                                 ByVal Addr As Integer) As Integer
```

Delphi:

```
Function ReadRegisteWord(hDevice : Integer;
                         Addr : Word) : Word;
                         StdCall; External 'ART2153' Name ' ReadRegisterWord ';
```

LabVIEW:

请参见相关演示程序。

功能：以双字节（即 16 位）方式读内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

Addr设备内存映射寄存器的线性基地址，它的值应由LoadBaseAddr 确定。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数： CreateDevice　WriteRegisterWord　ReadRegisterWord　ReleaseDevice