PCH2153 数据采集卡

WIN2000/XP 驱动程序使用说明书

北京阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

提醒用户：

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

有关设备驱动安装和产品二次发行请参考 PCH2153Inst.doc 文档。

第一章 版权信息

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、InitDeviceDmaAD、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StartDeviceProAD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StopDeviceProAD，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StartDeviceProAD即可启动AD部件，开始AD采样，接着调用GetDevStatusProAD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StopDeviceProAD，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。

第五节、如何用中断方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceIntAD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pPara参数结构体决定的。您只需要对这个pPara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。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hEvent赋给InitDeviceIntAD的相应参数，它将作为接受AD半满中断事件的变量。然后用StartDeviceIntAD即可启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hEvent中断事件的发生，在中断未到时，自动使所在线程进入睡眠状态（不消耗CPU时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用ReadDeviceIntAD函数一批半满长度（或半满以下）的AD数据，然后再接着再等待FIFO的半满中断事件，若有效再读取，就这样反复读取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StopDeviceIntAD，当您需要关闭AD设备时，ReleaseDeviceIntAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceIntAD函数在半满中断事件发生时可以单点或几点的方式读取AD数据，只是到下一次半满中断事件到来时的时间间隔会变得非常短，而不再是半满间隔，但它不同于半满查询方式读取，由于半满中断属于硬件中断，其优先级别高于所有软件，所以您单点或几点读取AD数据时，千万不能让中断间隔太短，否则，有可能使您的整个系统被半满中断事件吞没，就象死机一样，不能动弹。　切忌、切忌！）具体执行流程请看图 2.1.3。

注意：图中较粗的虚线表示对称关系。如红色虚线表示CreateDevice和ReleaseDevice两个函数的关系是：最初执行一次CreateDevice，在结束是就须执行一次ReleaseDevice。绿色虚线InitDeviceProAD与ReleaseDeviceProAD成对称方式出现。

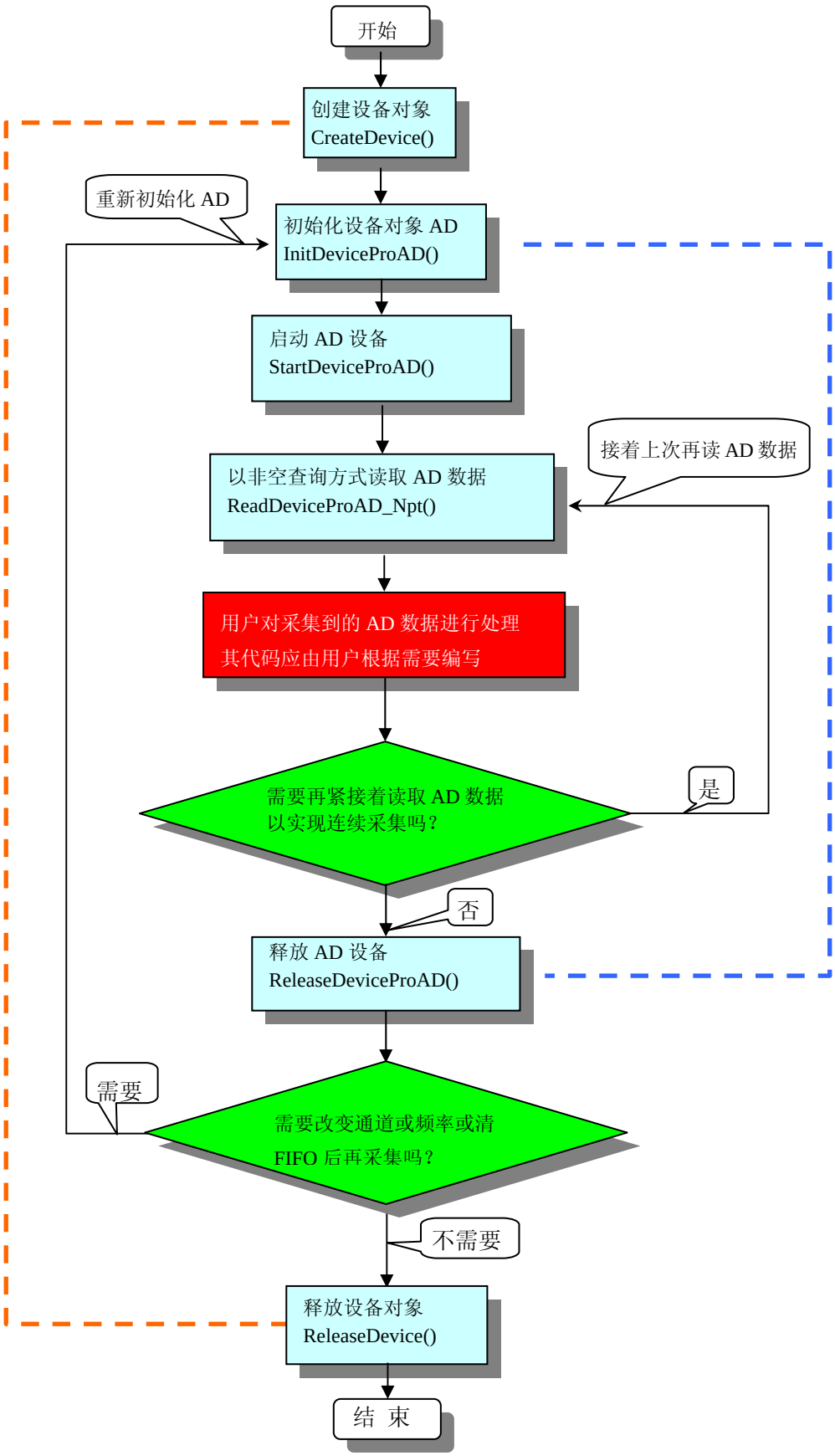


图 2.1.1 非空查询方式 AD 采集过程

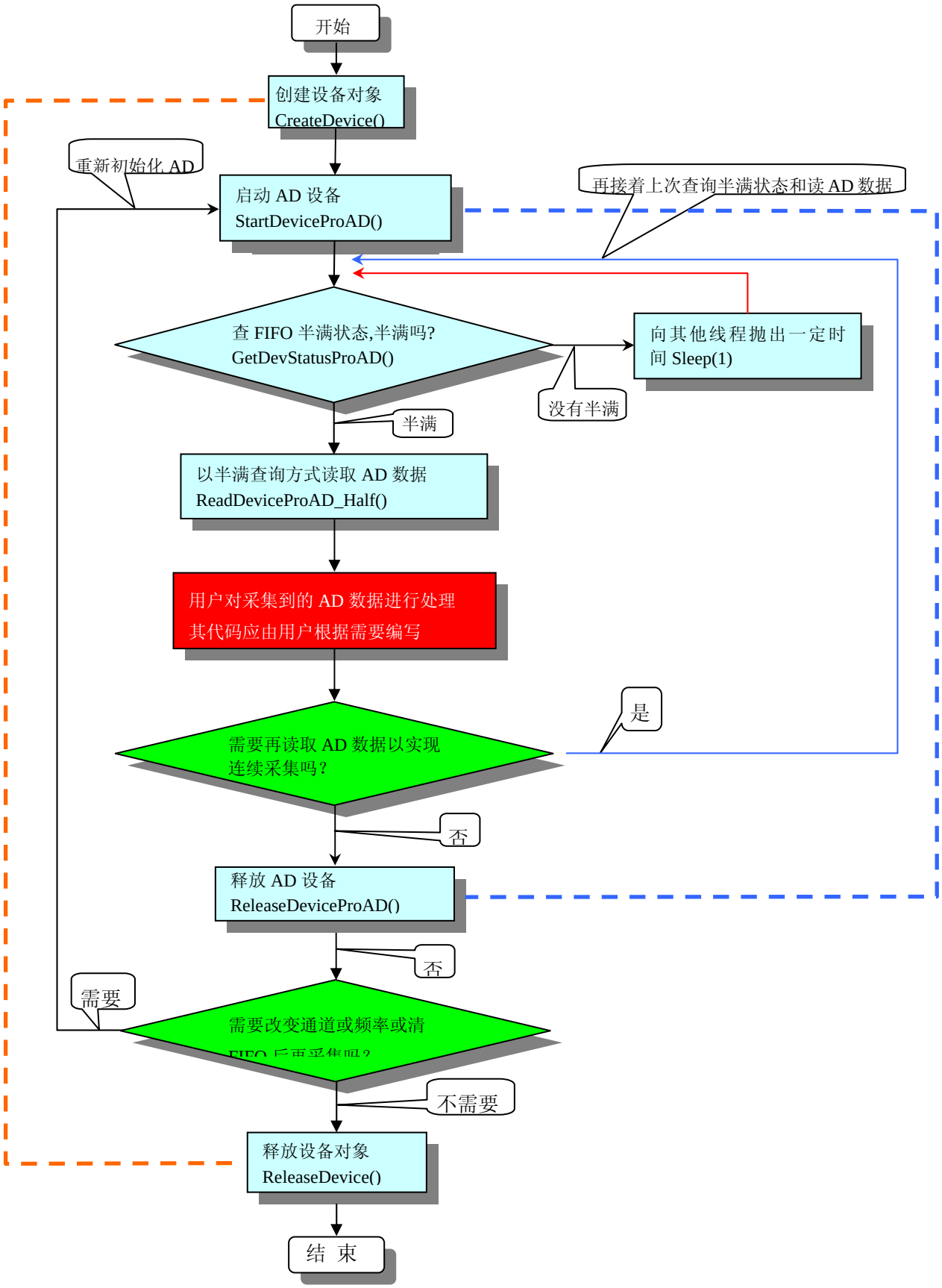


图 2.1.2 半满查询方式 AD 采集过程

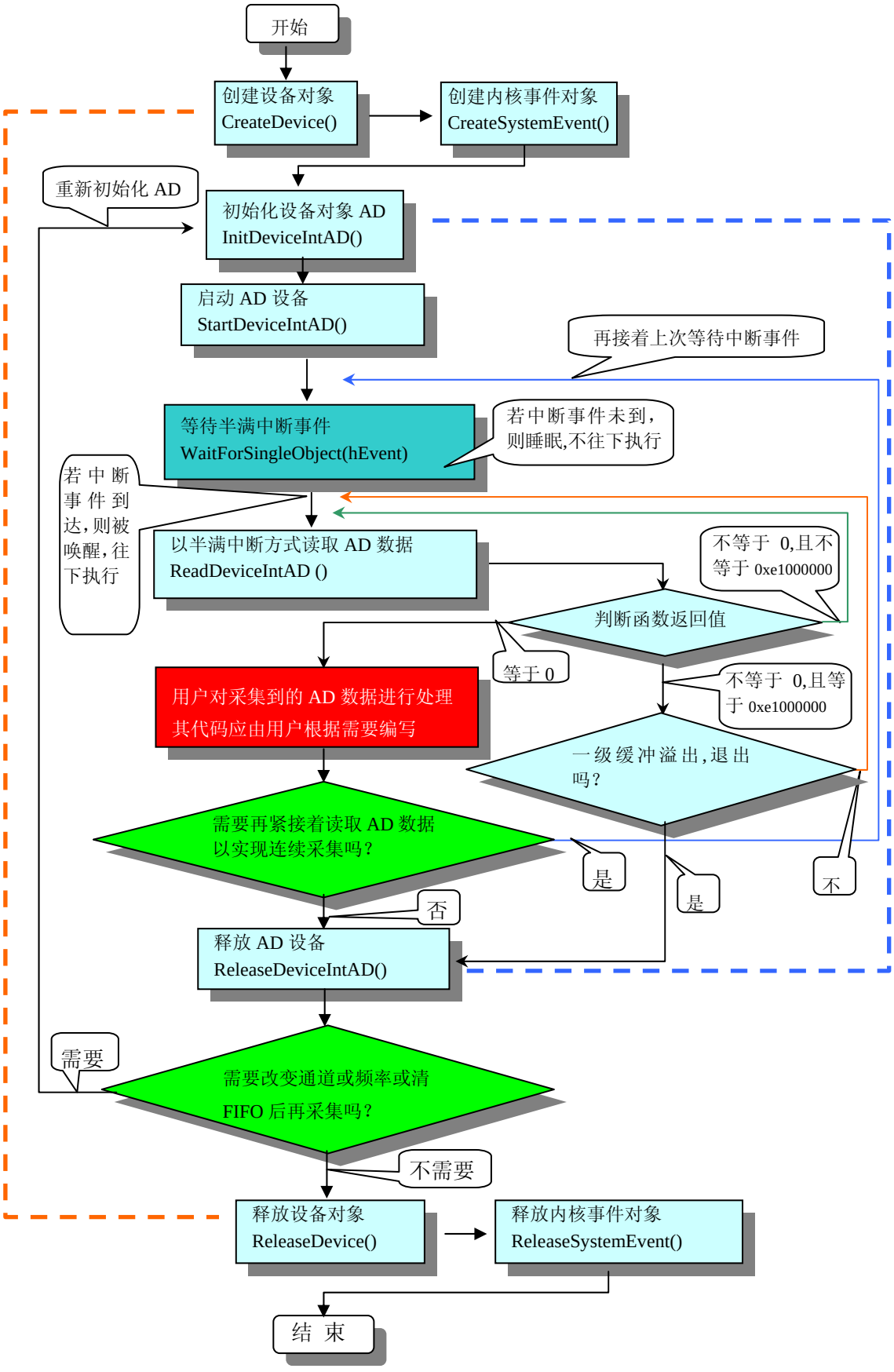


图 2.1.3 中断方式 AD 采集实现过程

第六节、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。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对PXI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“PCH2153_”）

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)	
GetDeviceCount	取得同一种设备的总台数	上层及底层用户
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
ListDeviceDlg	列表所有同一种设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放总线设备对象	
② AD 采样操作函数		
InitDeviceProAD	初始化设备 AD 部件，准备传数	
SetDevFrequencyAD	可动态改变 AD 采样频率	
StartDeviceProAD	启动 AD 设备，开始转换	
StopDeviceProAD	暂停 AD 设备	
GetDevStatusProAD	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 中断方式 AD 读取函数（唯有此种方式采用强制二级队列缓冲和动态链表技术）		
InitDeviceIntAD	初始化设备 AD 部件，如通道等	上层用户
StartDeviceIntAD	启动 AD 采集	上层用户
ReadDeviceIntAD	连续批量读取设备上的 AD 数据	上层用户
StopDeviceIntAD	停止 AD 采集	上层用户
ReleaseDeviceIntAD	释放设备上的 AD 部件	上层用户
④ AD 硬件参数系统保存、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	

使用需知

Visual C++ & C++Builder：

首先将 PCH2153.h 和 PCH2153.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(PCH2153.lib)加入到您的工程中。

#include “PCH2153.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 PCH2153.h 和 PCH2153.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 PCH2153.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 PCH2153.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 PCH2153.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 PCH2153.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：“PCH2153”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PCH2153;  // 注意： 在此加入驱动程序接口单元 PCH2153
```

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 (int DeviceLgcID = 0)
```

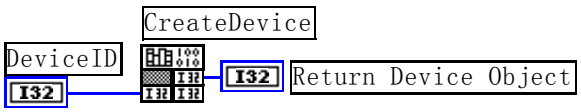
Visual Basic：

```
Declare Function CreateDevice Lib "PCH2153" (Optional ByVal DeviceLgcID As Integer = 0) As Long
```

Delphi：

```
Function CreateDevice(DeviceLgcID:Integer = 0):Integer;
                    StdCall; External 'PCH2153' 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：DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PCH2153 模板时，驱动程序逻辑号为“0”来确认和管理第一个设备，若用户接着再添加第二个 PCH2153 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个设备时，DeviceLgcID 应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由 BIOS 和操作系统加载设备时，依据主板总线编号等信息进行这个设备 ID 号分配，说得简单点，就是加载设备的顺序编号，编号的递增顺序为 0、1、2、3……。所以用户无法直接固定某一个设备的在设备列表中的物理位置，若想固定，则必须须使用物理 ID 号，调用 CreateDeviceEx 函数实现。

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 (0 ); // 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 (0 )   ' 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
          :
```

◆ **取得本计算机系统中 PCH2153 设备的总数量**

函数原型：

Visual C++ & C++Builder:

```
int GetDeviceCount (HANDLE hDevice)
```

Visual Basic:

```
Declare Function GetDeviceCount Lib "PCH2153" (ByVal hDevice As Long ) As Integer
```

Delphi：

```
Function PCH2153_GetDeviceCount (hDevice : Integer):Integer;
                                    StdCall; External 'PCH2153' Name 'GetDeviceCount';
```

LabVIEW：

请参考相关演示程序。

功能：取得 PCH2153 设备的数量。
参数：hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：返回系统中 PCH2153 的数量。
相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦ **取得该设备当前逻辑 ID 和物理 ID**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceCurrentID (HANDLE hDevice,
                         PLONG DevicePhysID,
                         PLONG DeviceLgcID)
```

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PCH2153" (ByVal hDevice As Long, _
                                                   ByRef DevicePhysID As Long, _
                                                   ByRef DeviceLgcID As Long ) As Boolean
```

Delphi:

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DevicePhysID: Pointer;
                             DeviceLgcID: Pointer;): Boolean;
                             StdCall; External 'PCH2153' Name ' GetDeviceCurrentID ';
```

LabVIEW：

请参考相关演示程序。

功能：取得 PCH2153 设备的数量。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
DevicePhysID 返回设备的物理 ID，它的取值范围为[0, 15]，它的具体值由卡上的拔码器 DID1 决定。
DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。
返回值：如果初始化设备对象成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。
相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PCH2153 设备各种配置信息**

函数原型：

Visual C++ & C++Builder:

```
BOOL ListDeviceDlg (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ListDeviceDlg Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ListDeviceDlg (hDevice : Integer):Boolean;
                        StdCall; External 'PCH2153' Name ' ListDeviceDlg ';
```

LabVIEW：

请参考相关演示程序。

功能：列表系统中 PCH2153 的硬件配置信息。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则弹出对话框控件列表所有 PCH2153 设备的配置情况。

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)
```

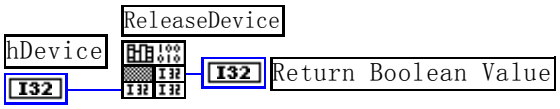
Visual Basic:

```
Declare Function ReleaseDevice Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function ReleaseDevice(hDevice : Longint):Boolean;
                    StdCall; External 'PCH2153' 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,
                      PPCH2153_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceProAD Lib "PCH2153" (ByVal hDevice As Long, _
                                                ByRef pADPara As PCH2153_PARA_AD) As Boolean
```

Delphi:

```
Function InitDeviceProAD(hDevice : Integer;
                         pADPara : PPCH2153_PARA_AD) : Boolean;
                         StdCall; External 'PCH2153' 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StartDeviceProAD。

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PCH2153_PARA_AD具体定义请参考PCH2153.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《AD硬件参数结构》章节。

返回值：如果初始化设备对象成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice SetDevFrequencyAD InitDeviceProAD StartDeviceProAD ReadDeviceProAD_Npt GetDevStatusProAD ReadDeviceProAD_Half StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

◆ **动态改变采样频率(Set device AD frequency)**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function SetDevFrequencyAD Lib "PCH2153" (ByVal hDevice as Long, _
                                                  ByVal nFrequency As Long) As Boolean
```

Delphi:

```
Function SetDevFrequencyAD (hDevice : Integer;
                            nFrequency: LongInt):Boolean;
                            StdCall; External 'PCH2153' 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(在分组采样中只能改变组内频率 Frequency)。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

nFrequency 指定 AD 的当前采样频率。本设备的频率取值范围为[1Hz, 250KHz]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

相关函数： CreateDevice SetDevFrequencyAD InitDeviceProAD StartDeviceProAD ReadDeviceProAD_Npt GetDevStatusProAD ReadDeviceProAD_Half StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

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

函数原型：

Visual C++ & C++Builder:

```
BOOL StartDeviceProAD ( HANDLE hDevice )
```

Visual Basic:

```
Declare Function StartDeviceProAD Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function StartDeviceProAD (hDevice : Integer ): Boolean;
                           StdCall; External 'PCH2153' Name ' StartDeviceProAD ';
```

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GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

♦ **读取设备上的 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 "PCH2153" (
                          ByVal hDevice As Long, _
                          ByRef ADBuffer As Integer,_
                          ByVal nReadSizeWords As Long,_
                          ByRef nRetSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Npt(hDevice : Integer;
                          ADBuffer : Pointer;
                          nReadSizeWords : LongInt
                          nRetSizeWords : Pointer) : Boolean;
                          StdCall; External 'PCH2153' Name ' ReadDeviceProAD_Npt ';
```

LabVIEW：

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD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GetLastErrorEx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　SetDevFrequencyAD　InitDeviceProAD
StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD
ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD
ReleaseDevice

② 使用 FIFO 的半满标志读取 AD 数据

♦ **取得 FIFO 的状态标志**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDevStatusProAD ( HANDLE hDevice,
                         PPCH2153_STATUS_AD pADStatus)
```

Visual Basic:

```
Declare Function GetDevStatusProAD Lib "PCH2153" (ByVal hDevice As Long,_
                                                  ByRef pADStatus As PCH2153_STATUS_AD)As Boolean
```

Delphi:

```
Function GetDevStatusProAD (hDevice : Integer;
                            PADStatus : PPCH2153_STATUS_AD) : Boolean;
                            StdCall; External 'PCH2153' Name ' GetDevStatusProAD ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数查询FIFO存储器的状态（半满标志、非空标志、溢出标志）。我们通常用半满标志去同步半满读操作。当半满标志有效时，再紧接着用ReadDeviceProAD_Half读取FIFO中的半满有效AD数据。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pADStatus 获得AD的各种当前状态。它属于结构体，具体定义请参考《AD状态参数结构（PCH2153_STATUS_AD）》章节。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。若用户选择半满查询方式读取AD数据，则当GetDevStatusProAD函数取得的bHalf等于TRUE，应立即调用ReadDeviceProAD_Half读取FIFO中的半满数据。否则用户应继续循环轮询FIFO半满状态，直到有效为止。注意在循环轮询期间，可以用Sleep函数抛出一定时间给其他应用程序(包括本应用程序的主程序和其他子线程)，以提高系统的整体数据处理效率。

相关函数： CreateDevice　SetDevFrequencyAD　InitDeviceProAD
StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD
ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD
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:

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

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "PCH2153" (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 'PCH2153' Name ' ReadDeviceProAD_Half ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用GetDevStatusProAD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。

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GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevFrequencyAD	InitDeviceProAD
StartDeviceProAD	ReadDeviceProAD_Npt	GetDevStatusProAD
ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
ReleaseDevice		

♦ 暂停 AD 设备

函数原型：

Visual C++ & C++Builder:

```
BOOL StopDeviceProAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceProAD Lib "PCH2153" (ByVal hDevice As Long )As Boolean
```

Delphi:

```
Function StopDeviceProAD (hDevice : Integer ) : Boolean;
                          StdCall; External 'PCH2153' Name ' StopDeviceProAD ';
```

LabVIEW

请参考相关演示程序。

功能：暂停AD设备。它必须在调用StartDeviceProAD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StartDeviceProAD函数重新启动AD，此时AD会按照暂停以前的状态（如FIFO存储器位置、通道位置）开始转换。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：如果调用成功，则返回TRUE，且AD立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice　SetDevFrequencyAD　InitDeviceProAD　StartDeviceProAD　ReadDeviceProAD_Npt　GetDevStatusProAD　ReadDeviceProAD_Half　StopDeviceProAD　ReleaseDeviceProAD　ReleaseDevice

♦ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseDeviceProAD(HANDLE hDevice)
```

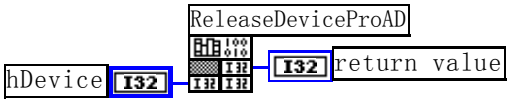
Visual Basic:

```
Declare Function ReleaseDeviceProAD Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDeviceProAD (hDevice : Integer) : Boolean;
                                StdCall; External 'PCH2153' 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GetLastErrorEx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 StartDeviceProAD
④ ReadDeviceProAD_Npt
⑤ StopDeviceProAD
⑥ ReleaseDeviceProAD
⑦ 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 InitDeviceProAD
③ StartDeviceProAD
④ GetDevStatusProAD
⑤ ReadDeviceProAD_Half
⑥ StopDeviceProAD
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。

关于两个过程的图形说明请参考《使用纲要》。

第四节、AD 中断方式采样操作函数原型说明

♦ 初始化设备上的 AD 对象

函数原型：

Visual C++ & C++ Builder:

```
BOOL InitDeviceIntAD(HANDLE hDevice,
                     HANDLE hEvent,
                     ULONG nFifoHalfLength,
                     PPCH2153_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceIntAD Lib "PCH2153" (ByVal hDevice As Long, _
                                                ByVal hEvent As Long, _
                                                ByVal nFifoHalfLength As Long,_
                                                ByRef pADPara As PCH2153_PARA_AD ) As Boolean
```

Delphi:

```
Function InitDeviceIntAD (hDevice : Integer;
                          hEvent : Integer;
                          nFifoHalfLength : Longword;
                          pADPara : PPCH2153_PARA_AD) : Boolean;
                          StdCall; External 'PCH2153' Name ' InitDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备操作就绪有关工作，如预置AD采集通道，采样频率等。且让设备上的AD部件以硬件中断的方式工作，其中断源信号由FIFO芯片半满管脚提供。但它并不启动AD采样，那么需要在此函数被成功调用之后，再调用StartDeviceIntAD函数即可启动AD采样。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

hEvent 中断事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的

内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEvent事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。

nFifoHalfLength 告诉设备对象，FIFO存储器半满长度大小。该参数很关键，因为不仅决定了设备对象每次产生半满中断时应读入AD数据的点数，同时，它也决定了一级缓冲队列中每个元素对应的缓冲区大小。比如，nFifoHalfLength等于 2048，则设备对象在系统空间中建立具有 64 个元素，且每个元素对应于 2048 个字长且物理连续的一级缓冲队列。但是该参数可以根据用户特殊需要，将其置成小于FIFO存储器实际的半满长度的值。比如用户要求在频率一定的情况下，提高FIFO半满中断事件的频率等，那么可以将此参数置成小于FIFO半满长度的值，但是绝不能大小半满长度。在工作期间，此队列的维护和管理完全由设备对象管理，与用户无关，用户只需要用ReadDeviceIntAD函数简单地读取AD数据，并注意检查其返回值即可。

pADPara设备对象参数结构指针，它的各成员值决定了设备上的AD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《硬件参数结构》章节。

返回值：如果初始化设备对象成功，则返回TRUE， 否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦ 启动设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL StartDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceIntAD Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StartDeviceIntAD (hDevice : Integer) : Boolean;
                    StdCall; External 'PCH2153' Name ' StartDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：在InitDeviceIntAD被成功调用之后，调用此函数即可启动设备上的AD部件，让设备开始AD采样。

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，意味着AD被启动，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦读取设备上的 AD 数据

Visual C++ & C++Builder:

```
DWORD ReadDeviceIntAD (HANDLE hDevice,
                    PWORD pADBuffer,
                    LONG nReadSizeWords,
```

```
                                PLONG nRetSizeWords )
```

Visual Basic:

```
Declare Function ReadDeviceIntAD Lib "PCH2153" (ByVal hDevice As Long,_
                                                ByRef pADBuffer As Integer,_
                                                ByVal nReadSizeWords As Long,_
                                                ByRef nRetSizeWords As Long) As Long
```

Delphi:

```
Function ReadDeviceIntAD (hDevice : Integer;
                          pADBuffer : Pointer;
                          nReadSizeWords : LongInt;
                          nRetSizeWords : Pointer) : Longword;
                          StdCall; External 'PCH2153' Name ' ReadDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用StartDeviceIntAD后，应立即用WaitForSingleObject等待中断事件hIntEvent的发生，如果FIFO还没有达到半满状态，即中断事件还发生，则数据采集线程WaitForSingleObject的作用下自动进入睡眠状态(此状态下，数据采集线程或代码不消耗CPU时间)。 当中断事件发生时线程被突发唤醒，即在WaitForSingleObject后的代码将被立即得到执行，因此为了提高数据吞吐率，在WaitForSingleObject之后，应紧接着用ReadDeviceIntAD函数读取FIFO半满数据。注意看演示程序如何处理这个问题。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADBuffer接受AD数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。关于如何将缓冲区中的这些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ReadDeviceIntAD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pADBuffer的最大空间长度，且由于是半满读取数据，所以这个参数必须等于板上FIFO存储器总容量的二分之一，比如FIFO为 1K长度（即 1024 点），则此参数应为 512，若为 4K（即 4096 点）长度，则此参数应为 2048，其他情况以此类推。当然特殊情况下，比如用户不要求数据连续或不担心丢点问题，则可以将此参数设得比FIFO存储器的半满长度小。需要用户特别注意的是此参数必须与InitDeviceIntAD函数中的nFifoHalfLength参数相等，才能实现连续数据采集。如果大于nFifoHalfLength，此会造成缓冲访问异常，严重时可能会使整过Windows系统崩溃，如果小于nFifoHalfLength，则会丢失n个点的数据在一级缓冲内(n为nFifoHalfLength 减去 nReadSizeWords的差值)。

nRetSizeWords 返回实际读取的点数(或字数)。

返回值：如果失败，即一级缓冲队列溢出则返回 0xe1000000 码，如果成功，则返回一级缓冲队列中的未被ReadDeviceIntAD读空的缓冲队列元素数量。一个元素对应于一个由InitDeviceIntAD函数的nFifoHalfLength参数指定大小的系统物理缓冲区。

注释：由于设备对象在系统空间中维护两个当前指针，且这两个指针最初都指向缓冲队列中的第一个元素。为了便于说明，我们将这两个指针分别命名为：用户指针和系统指针。当每执行此函数一次，设备对象将用户指针指向的缓冲区中的数据映射到用户空间pADBuffer中，且将用户指针下移一个元素位置。而系统指针则不随用户的操作而改变。它是设备对象强制自动维护的指针。它的改变速度只与AD数据转换有关。可见，不管用户有没有读走当前指针指向的一级缓冲区中的数据，或者整个Windows系统有多忙，但这个系统指针每到一个半满状态时，它总会自动下移一个元素位置。设备对象根据某些状态信息和利用巧妙算法，统计出已经采集了但用户迟迟没有读走的缓冲区数量，这个数量便是ReadDeviceIntAD返回的正确值，且判断数据是否重叠或溢出，这个状态便是ReadDeviceIntAD返回的 0xe1000000 码。如果用户的处理速度与得到设备对象的传输速度一样，那么ReadDeviceIntAD的返回值应等于 0，如果某一次在WaitForSingleObject之后执行ReadDeviceIntAD所

返回的值不为 0，且不为 0xe1000000，假如是 5，则视为一级缓冲区中的数据已有 5 个元素指向的数据是已采集的新数据，那么用户应接着用循环语句连读 5 次数据，直到ReadDeviceIntAD返回 0 为止。其他情况以此类推。此种方案的使用，让用户即便是在高速采集数据时，也能在很大程度上象往常一样随意进行窗口菜单等突发操作。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦ 暂停设备上的 AD 采样工作

函数原型：

Visual C++ & C++ Builder:

```
BOOL StopDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StopDeviceIntAD Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function StopDeviceIntAD (hDevice : Integer) : Boolean;
                    StdCall; External 'PCH2153' Name ' StopDeviceIntAD ';
```

LabVIEW:

请参考相关演示程序。

功能：在StartDeviceIntAD被成功调用之后，用户可以在任何时候调用此函数停止AD采样(必须在ReleaseDeviceIntAD之间被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceIntAD，那么设备会接着停止前的状态(如通道位置)继续开始正常的AD数据转换。

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，意味着AD被停止，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

♦ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDeviceIntAD Lib "PCH2153" (ByVal hDevice As Long ) As Boolean
```

Delphi：

```
Function ReleaseDeviceIntAD (hDevice : Integer) : Boolean;
                    StdCall; External 'PCH2153' Name ' ReleaseDeviceIntAD ';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的AD部件，如果AD没有被StopDeviceIntAD函数停止，则此函数在释放AD部件之前先停止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。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD
ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD
ReleaseDevice

应注意的是，InitDeviceIntAD必须和ReleaseDeviceIntAD函数一一对应，即当您执行了一次InitDeviceIntAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceIntAD函数，以释放先前由InitDeviceIntAD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InitDeviceIntAD函数时，那些软硬件资源才可被再次使用。

♦ **函数一般调用顺序**

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceIntAD
④ StartDeviceIntAD
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ ReadDeviceIntAD
⑦ StopDeviceIntAD
⑧ ReleaseDeviceIntAD
⑨ ReleaseSystemEvent（公共函数）
⑩ ReleaseDevice

注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。
关于这个过程的图形说明请参考《使用纲要》。

第五节、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,
                PPCH2153_PARA_AD pADPara)
```

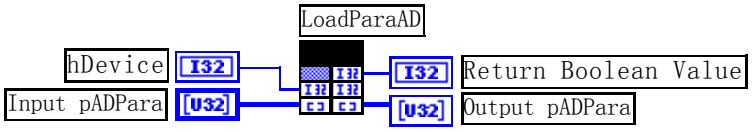
Visual Basic:

```
Declare Function LoadParaAD Lib "PCH2153" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCH2153_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD(hDevice : Integer;
                    pADPara:PPCH2153_PARA_AD):Boolean;
                    StdCall; External 'PCH2153' 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 PPCH2153_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 PPCH2153_PARA 请参考相应 PCH2153.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:

```
        :
PCH2153_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 PCH2153_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,
                PPCH2153_PARA_AD pADPara)
```

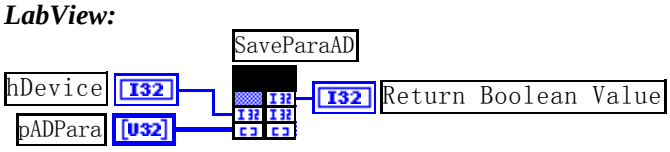
Visual Basic:

```
Declare Function SaveParaAD Lib "PCH2153" (ByVal hDevice As Long, _
                                           ByRef pADPara As PCH2153_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD ( hDevice : Integer;
                      pADPara:PPCH2153_PARA_AD):Boolean;
                      StdCall; External 'PCH2153' Name 'SaveParaAD';
```

LabView:



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

参数:

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara AD设备硬件参数，请参考《硬件参数结构》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice LoadParaAD ReleaseDevice

第四章　硬件参数结构

第一节、AD 硬件参数结构（PCH2153_PARA_AD）

Visual C++ & C++Builder:

```
typedef struct _PCH2153_PARA_AD        // 板卡各参数值
{
    LONG ADMode;              // AD 模式选择(连续/分组方式)
    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 InputRange;          // 模拟量输入量程范围
    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 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 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）
} PCH2153_PARA_AD, *PPCH2153_PARA_AD;
```

Visual Basic：

```
Private Type PCH2153_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
    InputRange As LongInt      ' 模拟量输入量程范围
    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
    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)
    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
    PPCH2153_PARA_AD = ^PCH2153_PARA_AD;   // 指针类型结构
```

```
PCH2153_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
    InputRange : LongInt;         // 模拟量输入量程范围
    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
    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)
    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:

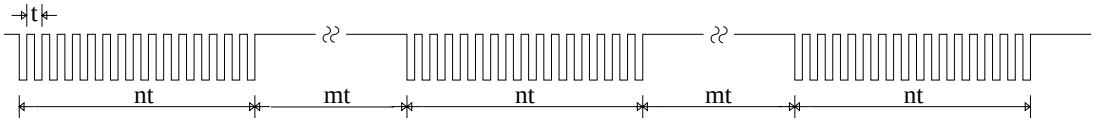
首先请您关注一下这个结构与前面 ISA 总线部分中的硬件参数结构 PARA 比较，该结构实在太简短了。其原因就是在于设备是系统全自动管理的设备，什么端口地址，中断号，DMA 等将与设备的用户永远告别，一句话设备简单得就象使用电源插头一样。

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InitDeviceProAD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
PCH2153_ADMODE_SEQUENCE	0x0000	连续采集模式
PCH2153_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。

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
PCH2153_INPUT_N10000_P10000mV	0x00	±10000mV
PCH2153_INPUT_N5000_P5000mV	0x01	±5000mV
PCH2153_INPUT_N2500_P2500mV	0x02	±2500mV
PCH2153_INPUT_0_P10000mV	0x03	0～10000mV
PCH2153_INPUT_0_P5000mV	0x04	0～5000mV

关于各个量程下采集的数据ADBuffer[]如何换算成相应的电压值，请参考《AD原始数据LSB转换成电压值Volt的换算方法》章节。

GroupInterval 分组间隔，指定两组间的时间间隔，单位微秒。

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, 65535]。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
PCH2153_GAINS_1MULT	0x00	1 倍增益
PCH2153_GAINS_2MULT	0x01	2 倍增益
PCH2153_GAINS_4MULT	0x02	4 倍增益
PCH2153_GAINS_8MULT	0x03	8 倍增益

TriggerMode AD触发模式，若等于常量PCH2153_TRIGMODE_SOFT则为内部软件触发，若等于常量PCH2153_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InitDeviceProAD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
PCH2153_TRIGMODE_SOFT	0x0000	软件触发(属于内触发)
PCH2153_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
PCH2153_TRIGSRC_ATR	0x0000	选择外部 ATR 作为触发源
PCH2153_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
PCH2153_TRIGTYPE_EDGE	0x0000	边沿触发
PCH2153_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
PCH2153_TRIGDIR_NEGATIVE	0x0000	负向触发(低脉冲/下降沿触发)
PCH2153_TRIGDIR_POSITIVE	0x0001	正向触发(高脉冲/上升沿触发)
PCH2153_TRIGDIR_POSIT_NEGAT	0x0002	正负向触发(高/低脉冲或上升/下降沿触发)

TrigLevelVolt 触发电平(0～10000mV)。

ClockSource AD 外时钟选择：

常量名	常量值	功能定义
PCH2153_CLOCKSRC_IN	0x0000	内部时钟
PCH2153_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。

常量名	常量值	功能定义
PCH2153_CLOCKOUT_DISABLE	0x0000	禁止本卡上的自带时钟向外输出
PCH2153_CLOCKOUT_ENABLE	0x0001	允许本卡上的自带时钟向外输出

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
PCH2153_ GNDMODE_SE	0x00	单端方式(SE:Single end)
PCH2153_ 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（PCH2153_STATUS_AD）

Visual C++ & C++Builder：

```
typedef struct _PCH2153_STATUS_AD
{
    LONG bNotEmpty;
    LONG bHalf;
    LONG bOverFull;
} PCH2153_STATUS_AD, *PPCH2153_STATUS_AD;
```

Visual Basic：

```
Private Type    PCH2153_STATUS_AD
    bNotEmpty As Long
    bHalf As Long
    bOverFull As Long
End Type
```

Delphi:

```
Type    // 定义结构体数据类型
        PPCH2153_STATUS_AD = ^ PCH2153_STATUS_AD;    // 指针类型结构
```

```
        PCH2153_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，GetDevStatusProAD函数使用此结构体来实时取得AD状态，以便同步各种数据采集和处理过程。

相关函数： CreateDevice　GetDevStatusProAD　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]&0xFFFF)– 10000.00	[-10000.00， + 9999.69]
±5000	Volt = (10000.00 / 65536) * (ADBuffer[0]&0xFFFF)– 5000.00	[-5000.00， + 4998.84]
±2500	Volt = (5000.00 / 65536) * (ADBuffer[0]&0xFFFF)– 2500.00	[-2500.00， + 2499.92]
0～10000	Volt = (10000.00 / 65536) * (ADBuffer[0]&0xFFFF)	[0.00， + 9999.84]
0～5000	Volt = (5000.00 / 65536) * (ADBuffer[0]&0xFFFF)	[0.00， + 4999.92]

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]&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) 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
```

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] AND 65535;
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)

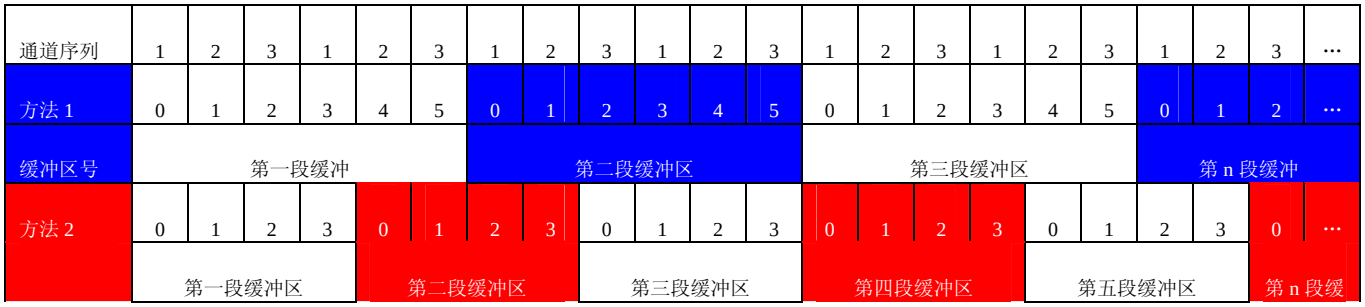
数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	…

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	…



第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;          // 文件头信息长度
    LONG FileType;
    // 该设备数据文件共有的成员
    LONG BusType;                // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;              // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;            // 头信息版本(D31-D16=Major   D15-D0=Minijor) = 1.0
    LONG VoltBottomRange;        // 量程下限(mV)
    LONG VoltTopRange;           // 量程上限(mV)
    PCH2153_PARA_AD ADPara;      // 保存硬件参数
    LONG StaticOverFlow;         // 同批文件识别码
    LONG HeadEndFlag;            // 文件结束位
    PCH2153_STATUS_AD ADStatus;
} FILE_HEADER,   *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第六章 上层用户函数接口应用实例

第一节、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]丨**[阿尔泰测控演示系统]**丨**[PCH2153 32 路 AD 卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[非空方式]**

第二节、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]丨[阿尔泰测控演示系统]丨[PCH2153 32 路 AD 卡]丨[Microsoft Visual C++]丨[简易代码演示]丨[半满方式]

第三节、怎样使用中断方式取得 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]丨[阿尔泰测控演示系统]丨[PCH2153 32 路 AD 卡]丨[Microsoft Visual C++]丨[简易演示程序]丨[中断方式]

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“PCH2153_”）

函数名	函数功能	备注
① 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定设备寄存器操作基地址	底层用户
GetDevVersion	获取设备固件及程序版本	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
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
③ 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	
④ 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、内存映射寄存器操作函数原型说明

◆ **取得指定内存映射寄存器的线性地址和物理地址**

函数原型：

Visual C++ & C++ Builder:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PULONG LinearAddr,
                    PULONG PhysAddr,
                    int RegisterID = 0)
```

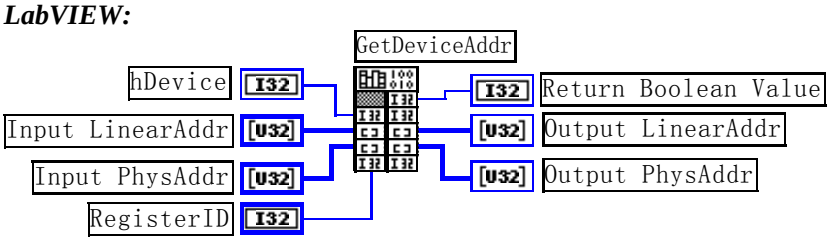
Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCH2153" (ByVal hDevice As Long, _
                                              ByRef LinearAddr As Long, _
                                              ByRef PhysAddr As Long, _
                                              Optional ByVal RegisterID As Integer = 0) As Boolean
```

Delphi:

```
Function GetDeviceAddr(hDevice : Integer;
                    LinearAddr : Pointer;
                    PhysAddr : Pointer;
                    RegisterID : Integer = 0) : Boolean;
                    StdCall; External 'PCH2153' Name ' GetDeviceAddr ';
```

LabVIEW:



功能：取得设备指定的内存映射寄存器的线性地址。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

LinearAddr 指针参数，用于取得的映射寄存器指向的线性地址，RegisterID 指定的寄存器组属于 MEM 模式时该值不应为零，也就是说它可用于 WriteRegisterX 或 ReadRegisterX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。但如果 RegisterID 指定的寄存器组属于 I/O 模式时该值通常为零，您不能通过以上函数访问设备。

PhysAddr 指针参数，用于取得的映射寄存器指向的物理地址，它指明该设备位于系统空间的物理位置。如果由 RegisterID 指定的寄存器组属于 I/O 模式，则可用于 WritePortX 或 ReadPortX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。

RegisterID 指定映射寄存器的 ID 号，其取值范围为[0，5]，通常情况下，用户应使用 0 号映射寄存器，特殊情况下，我们为用户加以申明。本设备的寄存器组 ID 定义如下：

常量名	常量值	功能定义
PCH2153_REG_MEM_PLXCHIP	0x0000	0 号寄存器对应 PLX 芯片所使用的内存模式基地址(使用 LinearAddr)
PCH2153_REG_IO_PLXCHIP	0x0001	1 号寄存器对应 PLX 芯片所使用的 IO 模式基地址(使用 PhysAddr)
PCH2153_REG_IO_CPLD	0x0002	2 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr)
PCH2153_REG_IO_ADFIFO	0x0003	3 号寄存器对应板上 AD FIFO 缓冲区所使用的 IO 模式基地址(使用 PhysAddr)

返回值：如果执行成功，则返回TRUE，它表明由RegisterID指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回，否则会返回FALSE，同时还要检查其LinearAddr和PhysAddr是否为 0，若为 0 则依然视为失败。用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr;
hDevice = CreateDevice(0);
if(!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox("取得设备地址失败…");
}
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr    As Long
hDevice = CreateDevice(0)
if Not GetDeviceAddr(hDevice, LinearAddr,    PhysAddr, 0) then
    MsgBox "取得设备地址失败…"
End If
    :
```

◆ **获取设备固件及程序版本**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function GetDevVersion Lib "PCH2153" (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 'PCH2153' Name ' GetDeviceBar';
```

LabVIEW:

请参考相关演示程序。

功能：获取设备固件及程序版本。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pulFmwVersion 固件版本。

pulDriverVersion 驱动版本。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

- **以单字节（即 8 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 WriteRegisterByte( HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

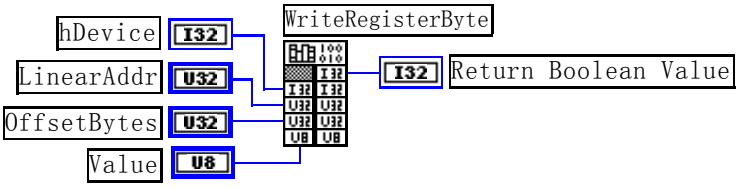
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCH2153" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte ) As Boolean
```

Delphi:

```
Function WriteRegisterByte(  hDevice : Integer;
                             LinearAddr : LongWord;
                             OffsetBytes : LongWord;
                             Value : Byte) : Boolean;
                             StdCall; External 'PCH2153' Name ' WriteRegisterByte ';
```

LabVIEW:



功能：以单字节（即 8 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。

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

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

Value 输出8位整数。

返回值：若成功，返回TRUE，否则返回FALSE。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);  // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
:
```

Visual Basic 程序举例:

```
:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
:
```

♦ **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,
                       ULONG LinearAddr,
                       ULONG OffsetBytes,
                       WORD Value)
```

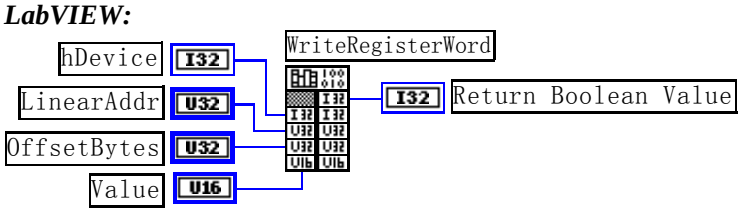
Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCH2153" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WriteRegisterWord( hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : Word) : Boolean;
                            StdCall; External 'PCH2153' Name ' WriteRegisterWord ';
```

LabVIEW:



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。

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

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 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。

Value 输出 16 位整型值。

返回值：无。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);  // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

♦ **以四字节（即 32 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 WriteRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

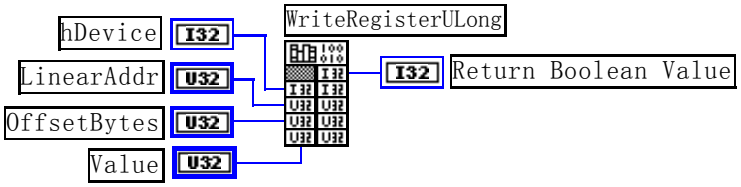
Visual Basic:

```
Declare Function WriteRegisterULong Lib "PCH2153" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long) As Boolean
```

Delphi:

```
Function WriteRegisterULong(hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : LongWord) : Boolean;
                            StdCall; External 'PCH2153' Name ' WriteRegisterULong ';
```

LabVIEW:



功能：以四字节（即 32 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。

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

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

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,   &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
```

```
OffsetBytes=100;        // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);   // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
    :
```

◆ **以单字节（即 8 位）方式读内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

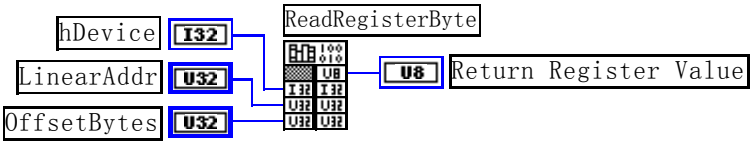
Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCH2153" (ByVal hDevice As Long, _
                                                 ByVal LinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

Delphi:

```
Function ReadRegisterByte(hDevice : Integer;
                          LinearAddr : LongWord;
                          OffsetBytes : LongWord) : Byte;
                          StdCall; External 'PCH2153' Name ' ReadRegisterByte ';
```

LabVIEW:



功能：以单字节（即 8 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。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,    PhysAddr, OffsetBytes;
BYTE    Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);     // 取得设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

◆ **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,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

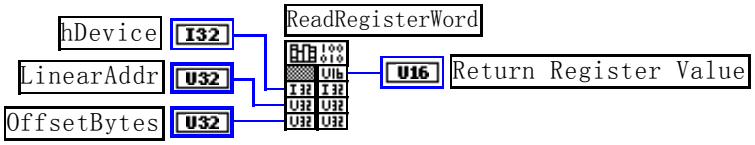
Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCH2153" ( ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Integer
```

Delphi:

```
Function ReadRegisteWord(hDevice : Integer;
                         LinearAddr : LongWord;
                         OffsetBytes : LongWord) : Word;
                         StdCall; External 'PCH2153' Name ' ReadRegisterWord ';
```

LabVIEW:



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。

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

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

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 GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );   // 释放设备对象
:
```

Visual Basic 程序举例:

```
:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:
```

♦ **以四字节（即32位）方式读内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes)
```

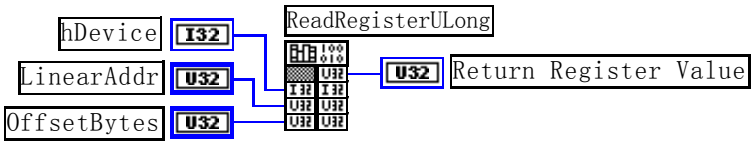
Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCH2153" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```

Delphi:

```
Function ReadRegisterULong(hDevice : Integer;
                           LinearAddr : LongWord;
                           OffsetBytes : LongWord) : LongWord;
                           StdCall; External 'PCH2153' Name ' ReadRegisterULong ';
```

LabVIEW:



功能：以四字节（即 32 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。

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

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

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr,  PhysAddr, OffsetBytes;
ULONG  Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);  // 取得设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );  // 释放设备对象
:
```

Visual Basic 程序举例:

```
:
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
:
```

第三节、I0 端口读写函数原型说明

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

◆ **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

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

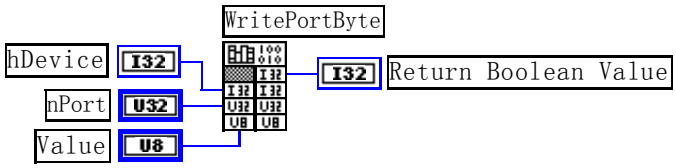
Visual Basic:

```
Declare Function WritePortByte Lib "PCH2153" ( 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 'PCH2153' 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)
```

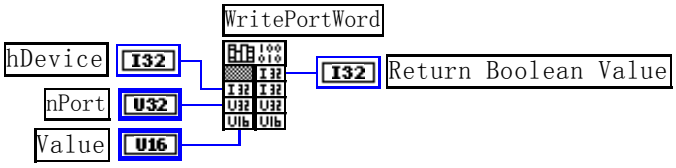
Visual Basic:

```
Declare Function WritePortWord Lib "PCH2153" (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 'PCH2153' 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)
```

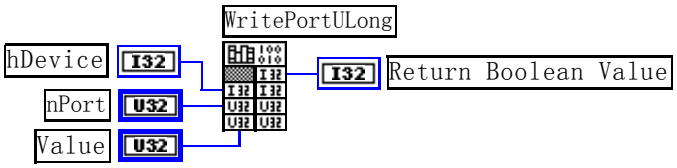
Visual Basic:

```
Declare Function WritePortULong Lib "PCH2153" (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 'PCH2153' 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)
```

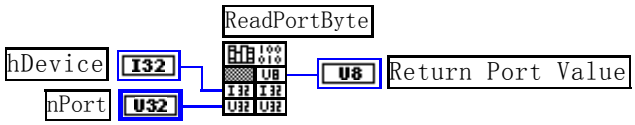
Visual Basic:

```
Declare Function ReadPortByte Lib "PCH2153" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

```
Function ReadPortByte(hDevice : Integer;
                      nPort : LongWord) : Byte;
                      StdCall; External 'PCH2153' 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)
```

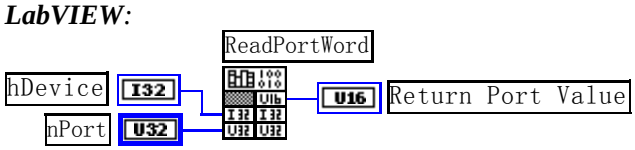
Visual Basic:

```
Declare Function ReadPortWord Lib "PCH2153" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer
```

Delphi:

```
Function ReadPortWord(hDevice : Integer;
                      nPort : LongWord) : Word;
                      StdCall; External 'PCH2153' 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)
```

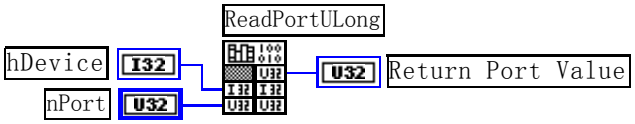
Visual Basic:

```
Declare Function ReadPortULong Lib "PCH2153" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long ) As Long
```

Delphi:

```
Function ReadPortULong(hDevice : Integer;
                       nPort : LongWord) : LongWord;
                       StdCall; External 'PCH2153' 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

第四节、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 "PCH2153" (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 PCH2153.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 "PCH2153" (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**

函数原型：

Visual C++ & C++ Builder:

```
HANDLE CreateSystemEvent(void)
```

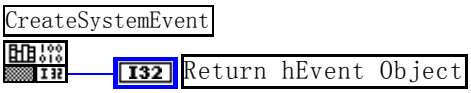
Visual Basic:

```
Declare Function CreateSystemEvent Lib " PCH2153 " () As Long
```

Delphi:

```
Function CreateSystemEvent() : Integer;
                    StdCall; External 'PCH2153' 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 " PCH2153 " (ByVal hEvent As Long) As Boolean
```

Delphi:

```
Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
                              StdCall; External 'PCH2153' Name ' ReleaseSystemEvent ';
```

LabVIEW:

请参见相关演示程序。

功能：释放系统内核事件对象。
参数：hEvent 被释放的内核事件对象。它应由CreateSystemEvent成功创建的对象。
返回值：若成功，则返回 TRUE。

第五节、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 strFileName,
                         int Mode)
```

Visual Basic:

```
Declare Function CreateFileObject Lib "PCH2153" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer;
                           strFileName : String;
                           Mode : Integer) : Integer;
                           Stdcall; external 'PCH2153' 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。

strFileName 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:\\PCH2153\\Data.Dat”，在 Basic 中，其语法格式如：“C:\PCH2153\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
PCH2153_modeRead	0x0000	只读文件方式
PCH2153_modeWrite	0x0001	只写文件方式
PCH2153_modeReadWrite	0x0002	既读又写文件方式
PCH2153_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
PCH2153_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,
               LONG nWriteSizeBytes)
```

Visual Basic:

```
Declare Function WriteFile Lib "PCH2153" ( ByVal hFileObject As Long,_
                                           ByRef pDataBuffer As Byte,_
                                           ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongInt) : Boolean;
                   Stdcall; external 'PCH2153' 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,
               LONG nOffsetBytes,
               LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PCH2153" ( ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal nOffsetBytes As Long, _
                                          ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile(hFileObject : Integer;
                  pDataBuffer : Pointer;
                  nOffsetBytes : LongInt;
                  nReadSizeBytes : LongInt) : Boolean;
                  Stdcall; external 'PCH2153' Name ' ReadFile ';
```

LabVIEW:

详见相关演示程序。

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

参数：

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

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

nOffsetBytes 指定从文件开始端所偏移的读位置。

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

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

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- **设置文件偏移位置**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function SetFileOffset Lib "PCH2153" ( ByVal hFileObject As Long,
                                               ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset ( hFileObject : Integer;
                         nOffsetBytes : LongInt) : Boolean;
                         Stdcall; external 'PCH2153' 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 "PCH2153" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject : Integer) : LongWord;
                        Stdcall; external 'PCH2153' 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 "PCH2153" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
Function ReleaseFile(hFileObject : Integer) : Boolean;
                     Stdcall; external 'PCH2153' 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：

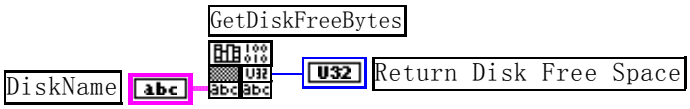
Visual C++ & C++ Builder:

```
ULONGLONG GetDiskFreeBytes(LPCTSTR strDiskName )
```

Visual Basic:

```
Declare Function GetDiskFreeBytes Lib "PCH2153" (ByVal strDiskName As String ) As Currency
```

LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：strDiskName 需要访问的盘符，若为 C 盘为"C:\\"， D 盘为"D:\\"，以此类推。

返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用GetLastErrorEx捕获错误码。注意使用 64 位整型变量。