

USB2813 驱动程序使用说明书

For Windows2000/XP/2003

USB2813 驱动程序使用说明书	1
第一章 版权信息	2
第二章 绪 论	2
第三章 设备专用函数接口介绍	4
第一节 设备驱动接口函数列表（每个函数省略了前缀“USB2813_”）	4
第二节 设备对象管理函数原型说明	5
第三节 AD采样操作函数原型说明	8
第四节 AD硬件参数系统保存与读取函数原型说明	10
第五节 DA输出函数原型说明	12
第五节 数字开关量输入输出简易操作函数原型说明	12
第六节 CNT计数器操作函数原型说明	13
第四章 硬件参数结构	14
第一节 AD硬件参数介绍（USB2813_PARA_AD）	14
第二节 CNT计数器控制字（USB2813_PARA_CNT）	15
第一节 AD原始数据LSB转换成电压值Volt的换算方法	16
第二节 DA电压值转换成LSB原码数据的换算方法	17
第三节 AD采集函数的ADBuffer缓冲区中的数据排放规则	17
第六章 上层用户函数接口应用实例	18
第一节 简易程序演示说明	18
第二节 高级程序演示说明	19
第七章 基于USB总线的大容量连续数据采集详述	19

第一章 版权信息

本软件产品及相关套件均属北京市阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与我们联系，我们将热情接待。

第二章 绪 论

一、如何管理 USB 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部件并启动AD设备，[ReadDeviceAD](#)函数可以用hDevice句柄实现对AD数据的采样批量读取，[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。

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[ReadDeviceAD](#)反复读取AD数据以实现连续不间断采样。当您需关闭AD设备时，[ReleaseDeviceAD](#)便可帮您实现（但设备对象hDevice依然存在）。（注：[ReadDeviceAD](#)虽然主要面对批量读取，高速连续采集而设计，但亦可用它以少量点如 32 个点读取AD数据，以满足慢速采集需要）。具体执行流程请看下面的图 2.1.1。

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

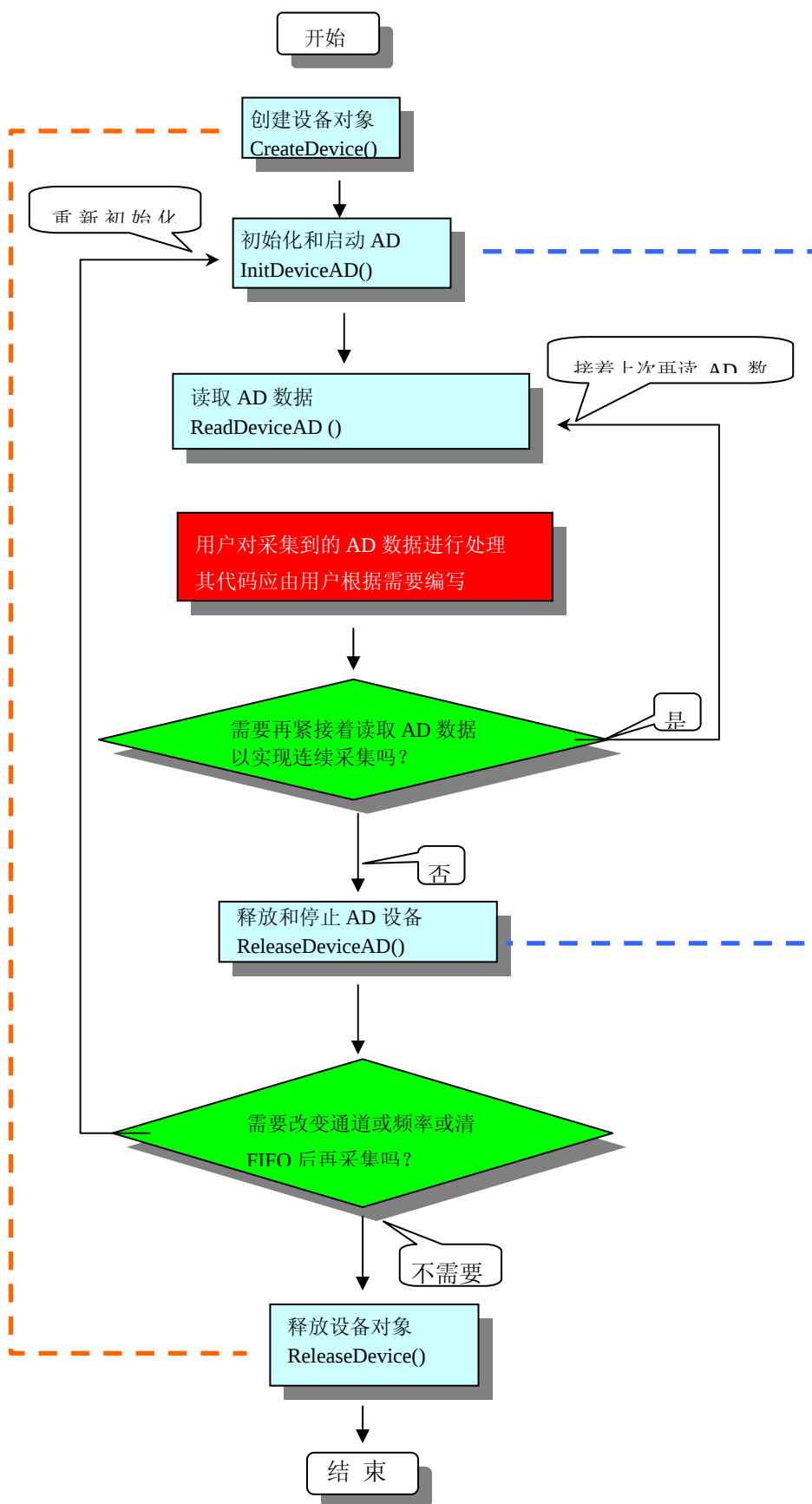


图 2.1.1 AD 采集实现过程

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“USB2813_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 USB 总线的设备对象	
GetDeviceCount	取得设备总数	
GetDeviceCurrentID	取得设备当前 ID 号	
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	
ResetDevice	复位 USB 设备	
ReleaseDevice	关闭设备，且释放 USB 总线设备对象	
② AD 采样操作函数		
InitDeviceAD	初始化 USB 设备 AD 部件，准备传数	
ReadDeviceAD	连续批量读取 USB 设备上的 AD 数据	
ReleaseDeviceAD	释放 USB 设备对象中的 AD 部件	
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 AD 采样参数恢复至出厂默认值	
④ 开关量函数		
GetDeviceDI	开关输入函数	
SetDeviceDO	开关输出函数	
⑤ DA 输出函数		
WriteDeviceDA	DA 输出函数	
⑥ 计数器操作函数		
InitDeviceCNT	设置计数器的初值	
GetDeviceCNT	取得各路计数器的当前计数值	

使用需知：

Visual C++:

要使用如下函数关键的问题是：

首先，将 USB2813.h 和 USB2813.lib 文件从 Visual C++ 的源程序目录下的任意一个子目录下复制到您的源程序目录下（若有 Advanced 高级源程序目录，则最好选择它），然后在您的源程序中包含如下语句（若想在整个工程的所有源代码文件中使用本驱动，请您最好在 StdAfx.h 全局头文件中包含如下语句）：

```
#include "USB2813.H"
```

那么对于导入库 USB2813.lib 文件您则可以不必再加入您的工程，因为 USB2813.h 头文件已帮助自动完成了。

Visual Basic:

要使用如下函数一个关键的问题是:

首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单,执行其中的"添加模块"(Add Module)命令,在弹出的对话框中选择 USB2813.Bas 模块文件,该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

请注意,因考虑 Visual C++和 Visual Basic 两种语言的兼容问题,在下列函数说明和示范程序中,所举的 Visual Basic 程序均是需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,我们不能保证完全顺利运行。

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。



- 一、在LabVIEW中打开USB2813.VI文件,用鼠标单击接口单元图标,比如[CreateDevice](#)图标然后按Ctrl+C或选择LabVIEW菜单Edit中的Copy命令,接着进入用户的应用程序LabVIEW中,按Ctrl+V或选择LabVIEW菜单Edit中的Paste命令,即可将接口单元加入到用户工程中,然后按以下函数原型说明或演示程序的说明连续该接口模块即可顺利使用。
- 二、根据LabVIEW语言本身的规定,接口单元图标以黑色的较粗的中竖线为中心,以左边的方格为数据输入端,右边的方格为数据的输出端,如[ReadDeviceAD](#)接口单元,左边为设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元,待单元接口被执行后,需要返回给用户的数据从接口单元右边的输出端输出,其他接口完全同理。
- 三、在单元接口图标中,凡标有“I32”为有符号长整型 32 位数据类型,“U16”为无符号短整型 16 位数据类型,“[U16]”为无符号 16 位短整型数组或缓冲区或指针,“[U32]”与“[U16]”同理,只是位数不一样。

第二节 设备对象管理函数原型说明**◆ 创建设备对象函数(逻辑号)**

函数原型:

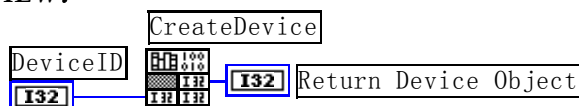
Visual C++:

HANDLE CreateDevice (int DeviceLgcID=0)

Visual Basic:

Declare Function CreateDevice Lib "USB2813_32" (Optional ByVal DeviceLgcID As Integer = 0) As Long

LabVIEW:



功能: 该函数使用逻辑号创建设备对象,并返回其设备对象句柄 hDevice。只有成功获取 hDevice,您才能实现对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PCI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USB2813 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USB2813 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USB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](#) [GetDeviceCount](#)
[GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

Visual C++ 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==INVALID_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 USB2813 设备的总数量

函数原型：

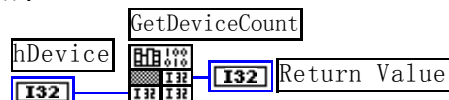
Visual C++:

[int GetDeviceCount \(HANDLE hDevice\)](#)

Visual Basic:

[Declare Function GetDeviceCount Lib "USB2813_32" \(ByVal hDevice As Long\) As Integer](#)

LabVIEW:



函数原型：

功能：取得 USB2813 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 USB2813 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++:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "USB2813_32" (ByVal hDevice As Long,_  
                                                    ByRef DeviceLgcID As Long,_  
                                                    ByRef DevicePhysID As Long) As Boolean
```

LabVIEW:

请参考演示源程序。

功能: 取得 USB2813 设备的数量。

参数:

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

DeviceLgcID 返回设备的逻辑 ID, 它的取值范围为[0, 7]。

DevicePhysID 返回设备的物理 ID, 它的取值范围为[0, 7], 它的具体值由卡上的拨码器 DID 决定。

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](#) [GetDeviceCount](#)
 [GetDeviceCurrentID](#) [ListDeviceDlg](#) [ReleaseDevice](#)

◆ 用对话框控件列表计算机系统中所有 USB2813 设备各种配置信息

函数原型:

Visual C++:

```
BOOL ListDeviceDlg (void)
```

Visual Basic:

```
Declare Function ListDeviceDlg Lib "USB2813_32" (void) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能: 列表系统中 USB2813 的硬件配置信息。

参数: 无。

返回值: 若成功, 则弹出对话框控件列表所有 USB2813 设备的配置情况。

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

◆ 复位整个 USB 设备

函数原型:

Visual C++:

```
BOOL ResetDevice (HANDLE hDevice)
```

Visual Basic:

```
Declare Function ResetDevice Lib "USB2813_32" (ByVal hDevice As Long) As Boolean
```

LabView:

请参考演示源程序。

功能：复位整个 USB 设备，相当于它与 PC 机端重新建立。其效果与重新插上 USB 设备等同。一般在出错情况下，想软复位来建决重连接问题，就可以调用该函数解决此问题。

参数：hDevice 设备对象句柄，它应由[CreateDevice](#)创建。由它指向要复位的设备。

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](#) [ReleaseDevice](#)

◆ 释放设备对象所占的系统资源及设备对象

函数原型：

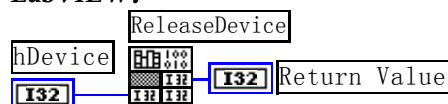
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "USB2813_32" (ByVal hDevice As Long) As Boolean

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](#)后，再一次执行这些函数前，必须执行一次[ReleaseDevice](#)函数，以释放由[CreateDevice](#)占用的系统软硬件资源，如DMA控制器，系统内存等。只有这样，当您再次调用[CreateDevice](#)函数时，那些软硬件资源才可被再次使用。

第三节 AD 采样操作函数原型说明

◆ 初始化设备对象

函数原型：

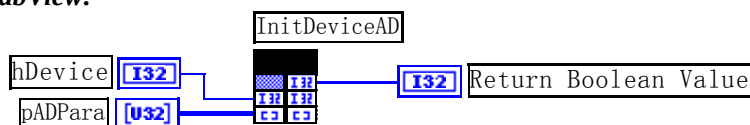
Visual C++:

BOOL InitDeviceAD(HANDLE hDevice,
PUSB2813_PARA_AD pADPara)

Visual Basic:

Declare Function InitDeviceAD Lib "USB2813_32" (ByVal hDevice As Long, _
ByRef pADPara As USB2813_PARA_AD) As Boolean

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USB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USB设备的[CreateDevice](#)创建。

pADPara 设备对象参数结构,其具体定义请参考《[AD硬件参数介绍](#)》章节,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。

返回值: 如果初始化设备对象成功,则返回 TRUE, 且 AD 便被启动。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。

◆ 批量读取 USB 设备上的 AD 数据

函数原型:

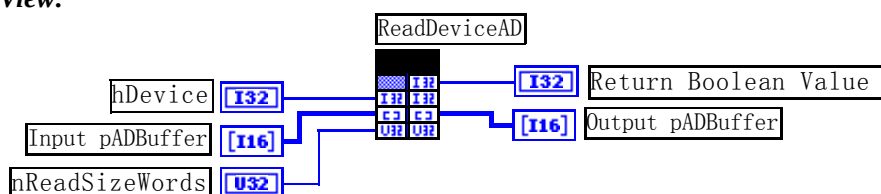
Visual C++:

```
BOOL ReadDeviceAD ( HANDLE hDevice,
                    USHORT pADBuffer,
                    LONG nReadSizeWords,
                    PLONG nRetSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceAD Lib "USB2813_32" (ByVal hDevice As Long, _
                                              ByRef ADBuffer As Integer, _
                                              ByVal nReadSizeWords As Long, _
                                              ByRef nRetSizeWords As Long) As Boolean
```

LabView:



功能: 读取USB设备AD部件上的批量数据。它不负责初始化AD部件,待读完整过指定长度的数据才返回。它必须在[InitDeviceAD](#)之后, [ReleaseDeviceAD](#)之前调用。

参数:

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

pADBuffer 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,请参考第五章《[数据格式转换与排列规则](#)》。

nReadSizeWords 读取数据的长度(以字为单位),为了提高读取速率,根据特定要求,其长度必须指定为 256 字的整数倍长,如 256、512、1024 …… 8192 等字长,同时,数据长度也要为采样通道数的整数倍,以便于通道数据对齐处理,所以 nReadSizeWords 为(256*(LastChannel - FirstChannel + 1))的整数倍。否则,USB 设备对象将失败该读操作。

nRetSizeWords 实际返回数据的长度(字)

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](#)

◆ 释放设备对象中的 AD 部件

函数原型:

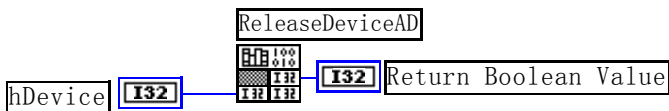
Visual C++:

`BOOL ReleaseDeviceAD(HANDLE hDevice)`

Visual Basic:

`Declare Function ReleaseDeviceAD Lib "USB2813_32" (ByVal hDevice As Long) As Boolean`

LabView:



功能: 释放设备对象中的 AD 部件所占用的系统资源。

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](#) [InitDeviceAD](#) [ReleaseDevice](#)

应注意的是, [InitDeviceAD](#) 必须和 [ReleaseDeviceAD](#) 函数一一对应, 即当您执行了一次 [InitDeviceAD](#), 再一次执行这些函数前, 必须执行一次 [ReleaseDeviceAD](#) 函数, 以释放由 [InitDeviceAD](#) 占用的系统软硬件资源, 如系统内存等。只有这样, 当您再次调用 [InitDeviceAD](#) 函数时, 那些软硬件资源才可被再次使用。这个对应关系对于非连续采样的场合特别适用。比如用户先采集一定长度的数据后, 然后对根据这些数据或其他条件, 需要改变采样通道或采样频率等配置时, 则可以先用 [ReleaseDeviceAD](#) 释放先已由 [InitDeviceAD](#) 占用的资源, 然后再用 [InitDeviceAD](#) 重新分配资源和初始化设备状态, 即可实现所提到的功能。

❖ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [InitDeviceAD](#)
- ③ [ReadDeviceAD](#)
- ④ [ReleaseDeviceAD](#)
- ⑤ [ReleaseDevice](#)

用户可以反复执行第③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④步后再回到第②步用新的状态信息重新初始设备。

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++:

`BOOL LoadParaAD(HANDLE hDevice, PUSB2813_PARA_AD pADPara)`

Visual Basic:

`Declare Function LoadParaAD Lib "USB2813_32" (ByVal hDevice As Long, _
ByRef pADPara As USB2813_PARA_AD) As Boolean`

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数:

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara属于PUSB2813_PARA_AD的结构指针类型，它负责返回PCI硬件参数值，关于结构指针类型PUSB2813_PARA_AD请参考USB2813.h或USB2813.Bas或USB2813.Pas函数原型定义文件，其具体定义请参考《[AD硬件参数介绍](#)》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

◆ 写设备硬件参数函数到 Windows 系统中

函数原型:

Visual C++:

BOOL SaveParaAD (HANDLE hDevice, PUSB2813_PARA_AD pADPara)

Visual Basic:

Declare Function SaveParaAD Lib "USB2813_32" (ByVal hDevice As Long, _
ByRef pADPara As USB2813_PARA_AD) As Boolean

LabVIEW:

请参考相关演示程序。

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

参数:

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara设备硬件参数，关于USB2813_PARA_AD的详细介绍请参考USB2813.h或USB2813.Bas或USB2813.Pas函数原型定义文件，其具体定义请参考《[AD硬件参数介绍](#)》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

◆ 将 AD 采样参数恢复至出厂默认值

函数原型:

Visual C++:

BOOL ResetParaAD(HANDLE hDevice, PUSB2813 PARA_AD pADPara)

Visual Basic:

Declare Function ResetParaAD Lib "USB2813_32" (ByVal hDevice As Long, _
ByRef pADPara As USB2813_PARA_AD) As Boolean

LabVIEW:

请参考相关演示程序。

功能：将 AD 采样参数恢复至出厂默认值。

参数:

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara设备硬件参数，关于USB2813_PARA_AD的详细介绍请参考USB2813.h或USB2813.Bas或USB2813.Pas函数原型定义文件，其具体定义请参考《[AD硬件参数介绍](#)》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

第五节 DA 输出函数原型说明

◆ DA 输出(DA output)

函数原型：

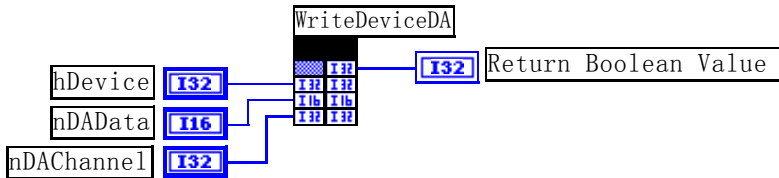
Visual C++:

```
BOOL WriteDeviceDA (HANDLE hDevice,
                    WORD nDALsb,
                    int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceDA Lib "USB2813_32" (ByVal hDevice As Long, _
ByVal nDALsb As Integer, ByVal nDAChannel As Integer) As Boolean
```

LabView:



功能：输出 DA 数据。

参数：

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

nDALsb 准备输出的 DA 数据 LSB 原码[0, 4095]。

nDAChannel DA 通道(0-3)。

返回值：若成功，返回 TRUE，否则返回 FALSE。

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

第五节 数字开关量输入输出简易操作函数原型说明

◆ 十六路开关量输出

函数原型：

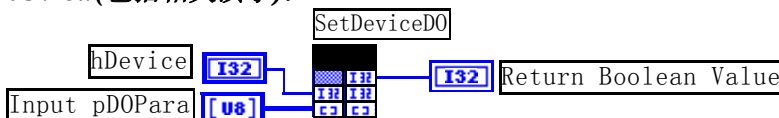
Visual C++:

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

Visual Basic:

```
Declare Function SetDeviceDO Lib "USB2813_32" (ByVal hDevice As Long, ByRef bDISTs As Byte) As Boolean
```

LabView(包括相关演示):



功能：负责将 USB 设备上的输出开关量置成相应的状态。

参数：

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

BYTE bDOSets[8] 开关输出状态(注意: 必须定义为 8 个字节元素的数组)。

返回值: 若成功, 返回 TRUE, 否则返回 FALSE。

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

◆ 十六路开关量输入

函数原型:

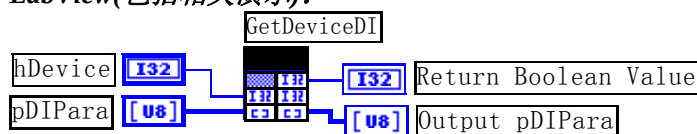
Visual C++:

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

Visual Basic:

Declare Function GetDeviceDI Lib "USB2813_32" (ByVal hDevice As Long, ByRef bDOSets As Byte) As Boolean

LabView(包括相关演示):



功能: 负责将 USB 设备上的输入开关量状态读入内存。

参数:

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

bDIs[8] 开关输入状态(注意: 必须定义为 8 个字节元素的数组)。

返回值: 若成功, 返回 TRUE, 其 pDIPara 中的值有效; 否则返回 FALSE, 其 pDIPara 中的值无效。

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

❖ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [SetDeviceDO](#) (或[GetDeviceDI](#), 当然这两个函数也可同时进行)
- ③ [ReleaseDevice](#)

用户可以反复执行第②步, 以进行数字 I/O 的输入输出 (数字 I/O 的输入输出及 AD 采样可以同时进行, 互不影响)。

第六节 CNT 计数器操作函数原型说明

◆ 初始化 8253 计数器(InitDeviceCNT)

函数原型:

Visual C++:

BOOL InitDeviceCNT (HANDLE hDevice,
PUSB2813_PARA_CNT pCNTPara,
int InitVal,
int nChannel)

Visual Basic:

Declare Function InitDeviceCNT Lib "USB2813_32" (ByVal hDevice As Long, _
ByRef pCNTPara As USB2813_PARA_CNT, _
ByVal InitVal As Integer, _
ByVal nChannel As Integer) As Boolean

LabView:

请参考相关演示程序。

功能: 负责初始化 8253 各通道的工作模式、计数方式等。

参数:

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

pCNTPara属于USB2813_PARA_CNT的结构指针型,它决定 8253 的计数方式、操作模型、计数类型,关于结构指针类型USB2813_PARA_CNT请参考USB2813.h或《[硬件参数结构](#)》章节。

InitVal 16 位计数初值。

nChannel 计数器通道号(0-2)。

返回值: 若成功,返回 TRUE, 否则返回 FALSE。

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

◆ 取得 8253 计数值(GetDeviceCNT)

函数原型:

Visual C++:

```
BOOL GetDeviceCNT ( HANDLE hDevice,
                    LONG CNTValue[3])
```

Visual Basic:

```
Declare Function GetDeviceCNT Lib "USB2813_32" (ByVal hDevice As Long,_
                                             ByRef CNTValue As Long) As Boolean
```

LabView:

请参考相关演示程序。

功能: 取得 8253 的当前计数值。

参数:

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

CNTVal[3] 属以数组形式获取 3 个计数器通道的当前计数值,低 16 位有效。

返回值: 若成功,返回 TRUE, 否则返回 FALSE。

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

第四章 硬件参数结构

第一节 AD 硬件参数介绍 (USB2813_PARA_AD)

Visual C++:

```
typedef struct _USB2813_PARA_AD      // 板卡各参数值
{
    LONG FirstChannel;      // 首通道,取值范围为[0, 15]
    LONG LastChannel;      // 末通道,取值范围为[0, 15]
    LONG InputRange;
```

```

    LONG GroundingMode;
} USB2813_PARA_AD, *PUSB2813_PARA_AD;

```

Visual Basic:

```

Private Type USB2813_PARA_AD
    FirstChannel As Long      ' 首通道,取值范围为[0, 15]
    LastChannel As Long      ' 末通道,取值范围为[0, 15]
    InputRange As Long
    GroundingMode As Long
End Type

```

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 USB 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 USB 设备的用户永远告别, 一句话 USB 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。

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 0~15, 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 0~15, 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。

InputRange 拟量输入范围所使用的选项

常量名	常量值	功能定义
USB2813_INPUT_N10000_P10000	0x00	±10000mV(默认配置)
USB2813_INPUT_N5000_P5000	0x01	±5000mV
USB2813_INPUT_N0_P10000	0x02	0~10000mV

GroundingMode 使用的模拟信号接地方式选项

常量名	常量值	功能定义
USB2813_GNDMODE_SE	0x0000	单端方式(SE:Single end)
USB2813_GNDMODE_DI	0x0001	双端方式(DI:Differential)

相关函数: [InitDeviceAD](#) [LoadParaAD](#) [SaveParaAD](#)

第二节 CNT 计数器控制字 (USB2813_PARA_CNT)

Visual C++:

```

typedef struct _USB2813_PARA_CNT      // 计数器控制字(CONTROL)
{
    BYTE OperateType;    // 操作类型
    BYTE CountMode;      // 计数方式
    BYTE CountType;      // 计数类型

```

```
} USB2813_PARA_CNT, *PUSB2813_PARA_CNT;
```

Visual Basic:

```
Type USB2813_PARA_CNT
```

```
    OperateType As Byte ' 操作类型
```

```
    CountMode As Byte ' 计数方式
```

```
    CountType As Byte ' 计数类型
```

```
End Type
```

LabView:

OperateType 计数器操作类型，它的取值范围应为如下常量之一。

常量名	常量值	功能定义
USB2813_OPT_TYPE_0	00H	计数器锁存操作
USB2813_OPT_TYPE_1	01H	只读/写低字节
USB2813_OPT_TYPE_2	02H	只读/写高字节
USB2813_OPT_TYPE_3	03H	先读/写低字节，后读/写高字节

CountMode 计数方式，它的取值范围应为如下常量之一。

常量名	常量值	功能定义
USB2813_CNT_MODE_0	00H	计数方式 0，计数器结束中断方式
USB2813_CNT_MODE_1	01H	计数方式 1，可编程单次脉冲方式
USB2813_CNT_MODE_2	02H	计数方式 2，频率发生器方式
USB2813_CNT_MODE_3	03H	计数方式 3，方波频率发生器方式
USB2813_CNT_MODE_4	04H	计数方式 4，软件触发选通方式
USB2813_CNT_MODE_5	05H	计数方式 5，硬件触发选通方式

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 16 位的模板CountLSB为 65536。其他类型同理均按 $2^n = \text{LSB总数}$ （n为Bit位数）换算即可。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000mV	$\text{Volt} = (20000.00 / 4096) * \text{ADBuffer}[0] - 10000.00$	[-10000, +9995.11]

换算举例：（设量程为±10000mV，这里只转换第一个点）

Visual C++:

```
SHORT Lsb; // 定义存放标准 LSB 原码的变量，一定要使用有符号 16 整型数
```

```
float Volt; // 定义存放转换后的电压值的变量
```

```
float PerLsbVolt = 20000.00/4096; // 求出每个 LSB 原码单位电压值
```

```
Lsb = ADBuffer[0]; // 一定要使用有符号 16 整型数
```

```
Volt = PerLsbVolt * Lsb - 10000.00; //用单位电压值与 LSB 原码相乘再减去偏移求得实际电压值
```

Visual Basic:

```
Dim Lsb As Integer ' 定义存放标准 LSB 原码的变量，一定要使用有符号 16 整型数
```

Dim Volt As Single ' 定义存放转换后的电压值的变量

Dim PerLsbVolt As Single

PerLsbVolt = 20000.00/4096 ' 求出每个 LSB 原码单位电压值

Lsb = ADBuffer(0) ' 一定要使用有符号 16 整型数

Volt = PerLsbVolt * Lsb - 10000.00 ' 用单位电压值与 LSB 原码数量相乘再减去偏移求得实际电压值

第二节 DA 电压值转换成 LSB 原码数据的换算方法

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

请注意这里求得的LSB数据就是用于[WriteDeviceDA](#)中的nDAData参数的。

第三节 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 - CH2)

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置为所选通道数量的整数倍长(在USB设备上同时也应 32 的整数倍), 这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个AD通道的数据进行连续循环采集, 则置每次读取长度为其 2 的整倍长 $2n$ (n 为每个通道的点数), 这里设为 2048。试想, 如此一来, 每次读取的 2048 个点中的第一个点始终对应于 1 通道数据, 第二个点始终对应于 2 通道, 第三个点再应于 1 通道, 第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据, 第 2048 个点对应 2 通道。这样一来, 每次读取的段长正好包含了从首通道到末通道的完整轮回, 如此一来, 用户只须按通道排列规则, 按正常的处理方法循环处理每一批数据。而对于其他情况也是如此, 比如 3 个通道采集, 则可以使用 $3n$ (n 为每个通道的点数)的长度采集。为了更加详细地说明问题, 请参考下表(演示的是采集 1、2、3 共三个通道的情况)。由于使用连续采样方式, 所以表中的数据序列一行的数字变化说明了数据采样的连续性, 即随着时间的延续, 数据的点数连续递增, 直至用户停止设备为止, 从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续, 其各通道数据在其整个数据链中的排放次序, 这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如[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 上层用户函数接口应用实例

如果您想快速的了解驱动程序的使用方法和调用流程，以最短的时间建立自己的应用程序，那么我们强烈建议您参考相应的简易程序。此种程序属于工程级代码，可以直接打开不用作任何配置和代码修改即可编译通过，运行编译链接后的可执行程序，即可看到预期效果。

如果您想了解硬件的整体性能、精度、采样连续性等指标以及波形显示、数据存盘与分析、历史数据回放等功能，那么请参考高级演示程序。特别是许多不愿意编写任何程序代码的用户，您可以使用高级程序进行采集、显示、存盘等功能来满足您的要求。甚至可以用我们提供的专用转换程序将高级程序采集的存盘文件转换成相应格式，即可在 Excel、MatLab 第三方软件中分析数据（此类用户请最好选用通过 Visual C++ 制作的高级演示系统）。

第一节 简易程序演示说明

一、怎样使用 [ReadDeviceAD](#) 函数进行 AD 连续数据采集

其详细应用实例及工程级代码请参考 Visual C++ 简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 USB2813.h 和 Sys.cpp)。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++] | [简易代码演示] | [AD 采集演示源程序]

其简易程序默认存放路径为：系统盘\ART\USB2813\SAMPLES\VC\SIMPLE\AD

二、怎样使用 [WriteDeviceDA](#) 函数进行 DA 输出操作

其详细应用实例及正确代码请参考 Visual C++ 简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 USB2813.h 和 Sys.cpp)。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++] | [简易代码演示] | [DA 输出演示源程序]

其默认存放路径为：系统盘\ART\USB2813\SAMPLES\VC\SIMPLE\DA

其他语言的演示可以用上面类似的方法找到。

三、怎样使用 [InitDeviceCNT](#) 和 [GetDeviceCNT](#) 函数进行计数器操作

其详细应用实例及正确代码请参考 Visual C++ 简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 USB2813.h 和 Sys.cpp)。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++] | [简易代码演示] | [CNT 计数器演示源程序]

其默认存放路径为：系统盘\ART\USB2813\SAMPLES\VC\SIMPLE\CNT

其他语言的演示可以用上面类似的方法找到。

四、怎样使用 [SetDeviceDO](#) 和 [GetDeviceDI](#) 函数进行开关量输入输出操作

其详细应用实例及正确代码请参考 Visual C++ 简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，

再按下列顺序点击, 即可打开基于 VC 的 Sys 工程(主要参考 USB2813.h 和 Sys.cpp)。

[\[程序\]](#) | [\[阿尔泰测控演示系统\]](#) | [\[Microsoft Visual C++\]](#) | [\[简易代码演示\]](#) | [\[开关量演示源程序\]](#)

其默认存放路径为: 系统盘\ART\USB2813\SAMPLES\VC\SIMPLE\DIO

第二节 高级程序演示说明

高级程序演示了本设备的所有功能, 您先点击 Windows 系统的[\[开始\]](#)菜单, 再按下列顺序点击, 即可打开基于 VC 的 Sys 工程(主要参考 USB2813.h 和 DADoc.cpp)。

[\[程序\]](#) | [\[阿尔泰测控演示系统\]](#) | [\[Microsoft Visual C++\]](#) | [\[高级代码演示\]](#)

其默认存放路径为: 系统盘\ART\USB2813\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

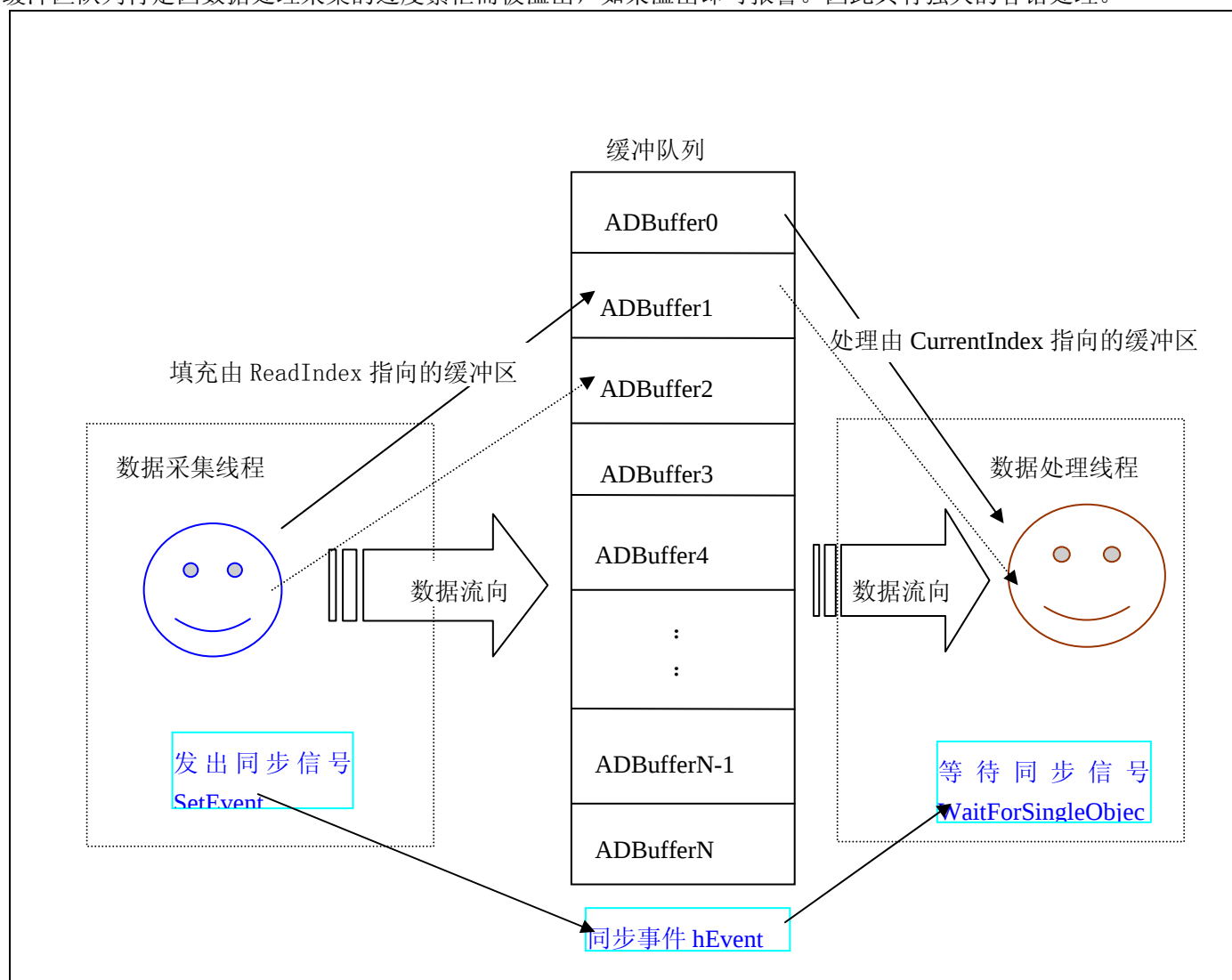
第七章 基于 USB 总线的大容量连续数据采集详述

与ISA、PCI设备同理, 使用子线程跟踪AD转换进度, 并进行数据采集是保持数据连续不间断的最佳方案。但是与ISA总线设备不同的是, USB设备在这里不使用动态指针去同步AD转换进度, 因为ISA设备环形内存池的动态指针操作是一种软件化的同步, 而USB设备不再有软件化的同步, 而完全由硬件和驱动程序自动完成。这样一来, 用户要用程序方式实现连续数据采集, 其软件实现就显得极为容易。每次用[ReadDeviceAD](#)函数读取AD数据时, 那么设备驱动程序会按照AD转换进度将AD数据一一放进用户数据缓冲区, 当完成该次所指定的点数时, 它便会返回, 当您再次用这个函数读取数据时, 它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次[ReadDeviceAD](#)之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中, 由于任务之间的调度切换非常平凡, 特别是当用户移动窗口、或弹出对话框等, 则会使当前线程猛地花掉大量的时间去处理这些图形操作, 因此如果处理不当, 则将无法实现高速连续不间断采集, 那么如何更好的克服这些问题呢? 用子线程则是必须的(在这里我们称之为数据采集线程), 但这还不够, 必须要求这个线程是绝对的工作者线程, 即这个线程在正常采集中不能有任何窗口等图形操作。只有这样, 当用户进行任何窗口操作时, 这个线程才不会被堵塞, 因此可以保证正常连续的数据采集。但是用户可能要问, 不能进行任何窗口操作, 那么我如何将采集的数据显示在屏幕上呢? 其实很简单, 再开辟一个子线程, 我们称之为数据处理线程, 也叫用户界面线程。最初, 数据处理线程不做任何工作, 而是在 Win32 API 函数 WaitForSingleObject 的作用下进入睡眠状态, 此时它不消耗 CPU 任何时间, 即可保证其他线程代码有充分的运行机会(这里当然主要指数据采集线程), 当数据采集线程取得指定长度的数据到用户空间时, 则再用 Win32 API 函数 SetEvent 将指定事件消息发送给数据处理线程, 则数据处理线程即刻恢复运行状态, 迅速对这批数据进行处理, 如计算、在窗口绘制波形、存盘等操作。

可能用户还要问, 既然数据处理线程是非工作者线程, 那么如果用户移动窗口等操作堵塞了该线程, 而数据采集线程则在不停地采集数据, 那数据处理线程难道不会因此而丢失数据采集线程发来的某一段数据吗? 如果不另加处理, 这个情况肯定有发生的可能。但是, 我们采用了一级缓冲队列和二级缓冲队列的设计方案, 足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K 数据, 那么我们就创建一个缓冲队列, 在用户程序中最简单的办法就是开辟一个二维数组如 PADBuffer[Count][DataLen], 我们将 DataLen 视为数据采集线程每次采集的数据长度, Count 则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32, 则这个缓冲队列实际上就是数组 ADBuffer[32][8192] 的形式。那么如何使用这个缓冲队列呢? 方法很简单, 它跟一个普通的缓冲区如一维数组差不多, 唯一不同是, 两个线程首先要通过改变 Count 字段的值, 即这个下标 Index 的值来填充和引用由 Index 下标指向某一段 DataLen 长度的数据缓冲区。需要注意的两个线程不共用一个 Index 下标变量。具体情况是当数据采集线程在 AD 部件被 [InitDeviceAD](#) 初始化之后, 首次采集数据时, 则将自己的 ReadIndex 下标置为 0, 即用第一个缓冲区采集 AD 数据。当采集完后, 则向数据处理线程发送消息, 且两个线程的公共变量 SegmentCounts 加 1, (注意 SegmentCounts 变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了, 但是却没被数据

处理线程处理掉的缓冲区数量。)然后再接着将ReadIndex偏移至 1,再用第二个缓冲区采集数据。再将SegmentCounts加 1,至到ReadIndex等于 15 为止,然后再回到 0 位置,重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数,然后逐一进行处理,最后再从SegmentCounts变量中减去在所接受到的当前事件下所处理的缓冲区个数。因此,即便应用程序突然很忙,使数据处理线程没有时间处理已到来的数据,但是由于缓冲区队列的缓冲作用,可以让数据采集线程先将数据连续缓存在这个区域中,由于这个缓冲区可以设计得比较大,因此可以缓冲很大的时间,这样即便是数据处理线程由于系统的偶而繁忙而被堵塞,也很难使数据丢失。而且通过这种方案,用户还可以在数据采集线程中对SegmentCounts加以判断,观察其值是否大小了 32,如果大于,则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出,如果溢出即可报警。因此具有强大的容错处理。



下面只是简要的策略说明,其详细应用实例请参考 Visual C++测试与演示系统,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++ Sample] |

下面只是基于 C 语言的简要的策略说明,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(ADDoc.h 和 ADDoc.cpp)。

[程序] | [阿尔泰测控演示系统] | [Microsoft Visual C++ Sample] |

然后,您着重参考 ADDoc.cpp 源文件中以下函数:

```
void CADDoc::StartDeviceAD()           // 启动线程函数
UINT ReadDataThread (PVOID hWnd)      // 读数据线程
UINT ProcessDataThread (PVOID hWnd)   // 绘制数据线程
```

```
void CADDoc::StopDeviceAD()           // 终止采集函数
```