USB2851 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司

产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 USBxxxx_则被省略。如 USB2851_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写规则

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

以上规则不局限于该产品。

第二章 使用纲要

第一节、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InitDeviceAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD函数可以用hDevice句柄实现对AD数据的采样读取等。最后可以通过ReleaseDevice将hDevice释放掉。

第二节、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。当您需要关闭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。

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，在结束时就须执行一次ReleaseDevice。

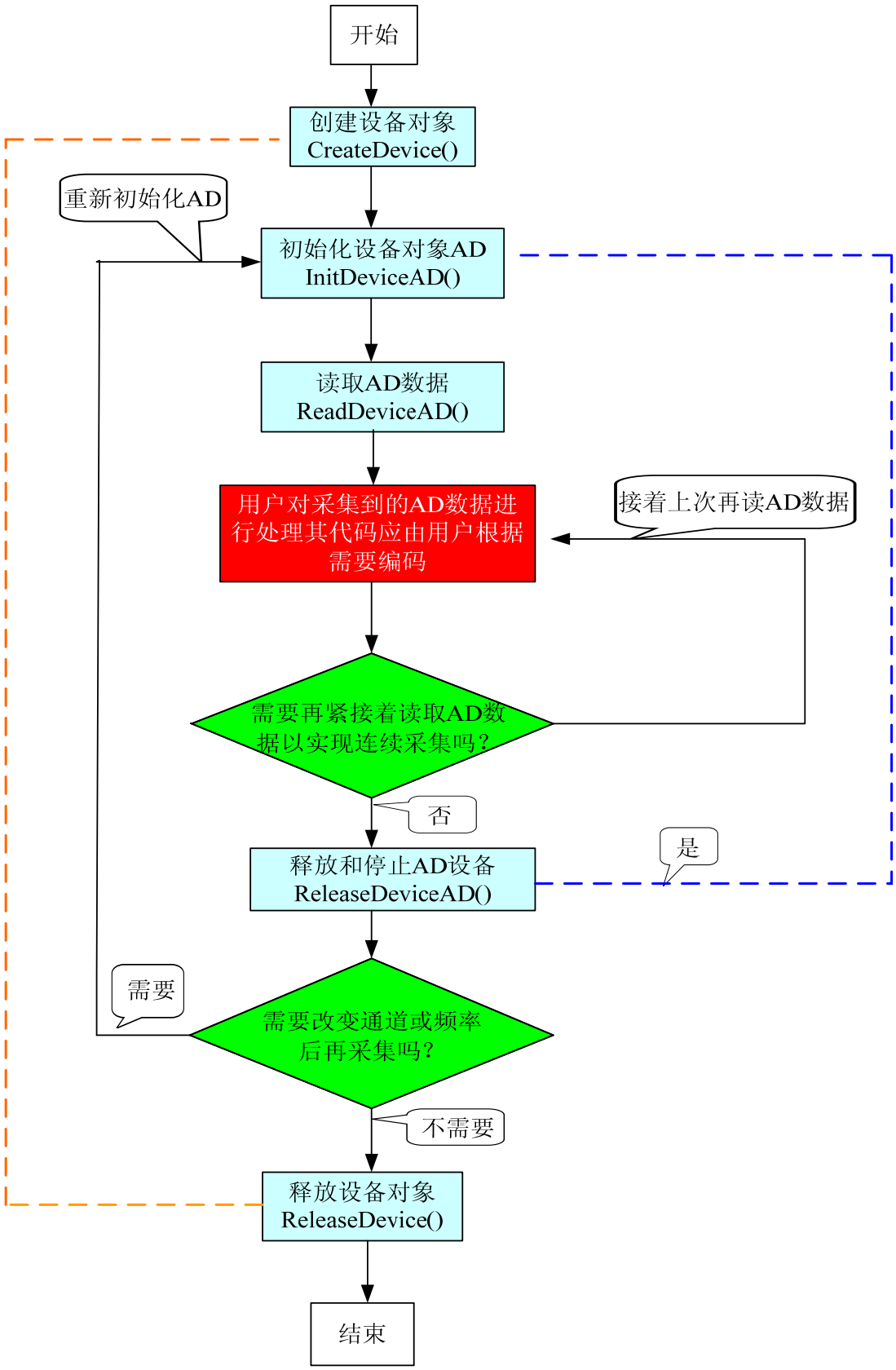


图 2.1.1　AD 采样实现过程

第三节、如何实现开关量的简便操作

当您有了hDevice设备对象句柄后，便可用SetDeviceDO函数实现开关量的输出操作，其各路开关量的输出状态由其bDOSts[8]中的相应元素决定。由GetDeviceDI函数实现开关量的输入操作，其各路开关量的输入状态由其bDOSts[8]中的相应元素决定。

第四节、哪些函数对您不是必须的

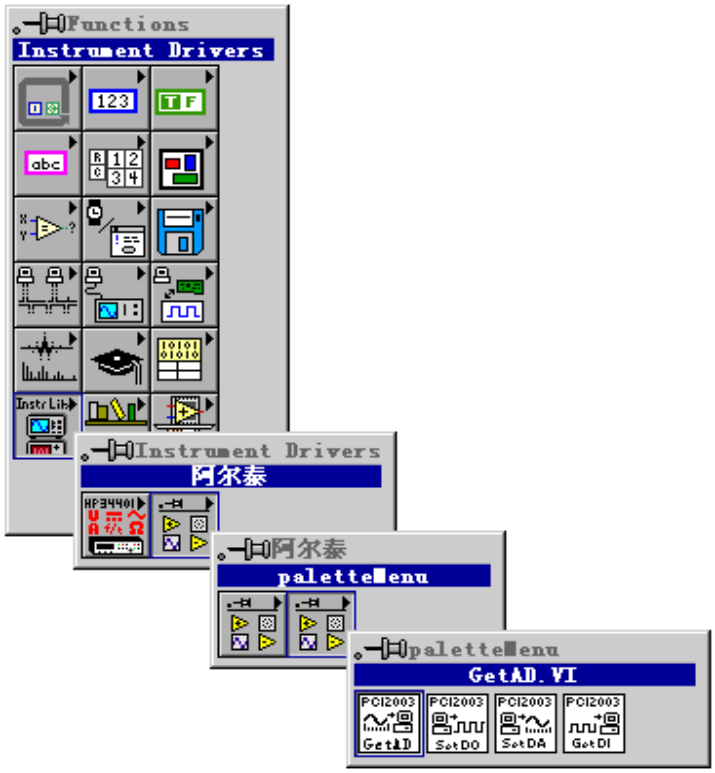
公共函数如CreateFileObject，WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 USB 设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如InitDeviceAD）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用ReadDeviceAD）函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉USB总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解USB的资源配置空间、PNP即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用ReadRegisterULong和WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“USB2851_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 USB 设备对象(用设备逻辑号)	上层及底层用户
CreatDeviceEx	创建 USB 对象（用设备物理号）	
GetDeviceCount	取得同一种 USB 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 USB 设备的各种配置	上层及底层用户
ResetDevice	复位 USB 设备	
ReleaseDevice	关闭设备，且释放 USB 总线设备对象	上层及底层用户
② AD 采样操作函数		
InitDeviceAD	初始化 AD 部件准备传输	上层用户
ReadDeviceAD	连续读取当前 USB 设备上的 AD 数据	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
③ AD 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
④ DA 数据输出操作函数		
WriteDeviceDA	写 DA 数据	上层用户
⑤ 开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	上层用户
RetDeviceDO	回读数字量输出状态	上层用户
⑥ 以太网操作函数		
SetNetCfg	设置以太网参数	
GetNetCfg	获得设备的以太网设置参数	

使用需知：

Visual C++ & C++Builder：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\Art\USB2851\INCLUDE\USB2851.H”

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 USB2851.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

C++ Builder:

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头文件(USB2851.H)写进您的源程序头部。如：#include “\Art\USB2851\Include\USB2851.h”，然后再将 USB2851.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”（Add to Project）命令，在弹出的对话框中分别选择文件类型：Library file (*.lib) ，即可选择 USB2851.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。

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，在弹出的对话中选择 USB2851.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。

Delphi:

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单元模块文件（*.Pas）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 USB2851.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。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。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：“USB2851”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  USB2851;  // 注意： 在此加入驱动程序接口单元 USB2851
```

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 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：

一、在 LabView 中打开 USB2851.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接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间线为中心，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，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++ & C++Builder：

```
HANDLE CreateDevice (int DeviceID = 0)
```

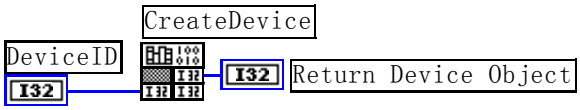
Visual Basic：

```
Declare Function CreateDevice Lib "USB2851" (Optional ByVal DeviceID As Integer = 0) As Long
```

Delphi：

```
Function CreateDevice(DeviceID : Integer = 0) : Integer;
                    StdCall; External 'USB2851' Name ' CreateDevice ';
```

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，您才能实现对该设备所有功能的访问。

参数：

DeviceID 设备 ID(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，系统将以该设备的“基本名称”与 DeviceID 标识值为名称后缀的标识符来确认和管理该设备。默认值为 0。

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

相关函数： ReleaseDevice

◆ **创建设备对象函数（扩展函数）**

函数原型：

Visual C++ & C++ Builder:

```
HANDLE CreateDeviceEx(int DevicePhysID = 0)
```

Visual Basic:

```
Declare Function CreateDeviceEx Lib "USB2851" (ByVal DevicePhysID As Long = 0) As Long
```

Delphi:

```
Function CreateDeviceEx(DevicePhysID:Integer = 0):Integer;
                    StdCall; External 'USB2851' Name 'CreateDeviceEx';
```

LabView:

请参考相关演示程序。

功能：该函数负责创建设备对象，并返回其设备对象句柄。设备对象句柄是访问某一台设备的唯一依据。不同 DevicePhysID 创建的设备对象句柄用于访问不同的设备。只有成功创建 DevicePhysID 指定设备的句柄后，设备对用户来讲才是可用的。因为每一个访问设备的驱动函数接口都需要 hDevice 这个设备句柄参数。

参数：

DevicePhysID 设备物理 ID(Identifier)标识号。如果您使用单个 USB2851 设备，该参数等于 0 即可。

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

相关函数： ReleaseDevice

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，管理第一个 USB 设备
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设备的总数量**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function GetDeviceCount Lib "USB2851" (ByVal hDevice As Long) As Integer
```

Delphi：

```
Function GetDeviceCount (hDevice : Integer) : Integer;
                          StdCall; External 'USB2851' Name ' GetDeviceCount ';
```

LabVIEW：

请参考相关演示程序。

功能：取得在系统中物理设备的数量。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

返回值：若成功返回实际设备的数量。

相关函数： CreateDevice 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 DeviceLgcID，
                         PLONG DevicePhysID)
```

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "USB2851" (ByVal hDevice As Long, _
                                                   ByRef DeviceLgcID As Long, _
                                                   ByRef DevicePhysIDAs Long) As Boolean
```

Delphi：

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DeviceLgcID : Pointer;
                             DevicePhysID : Pointer) : Boolean;
                             StdCall; External 'USB2851' Name ' GetDeviceCurrentID ';
```

LabVIEW：

请参考相关演示程序。

功能：取得指定设备逻辑和物理 ID 号。

参数：

hDevice 设备对象句柄，它指向要取得逻辑和物理号的设备，它应由CreateDevice创建。

DeviceLgcID 返回设备的逻辑 ID，它的取值范围为[0, 15]。

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 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GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice ReleaseDevice

- **用对话框控件列表计算机系统中所有 USB2851 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 "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi：

```
Function ListDeviceDlg (hDevice: Integer) : Boolean;
                        StdCall; External 'USB2851' Name ' ListDeviceDlg ';
```

LabVIEW：

请参考相关演示程序。

功能：列表系统中 USB2851 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 USB2851 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++ & C++Builder:

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

Visual Basic:

```
Declare Function ResetDevice Lib "USB2852" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function ResetDevice (hDevice: Integer) :Boolean;
                      StdCall; External 'USB2851' Name 'ResetDevice';
```

LabView:

请参考相关演示程序。

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，想用软复位来解决重连接问题，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++ & C++Builder:

```
BOOL ReleaseDevice(HANDLE hDevice)
```

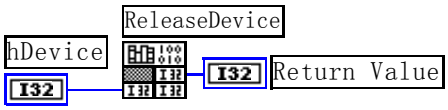
Visual Basic:

```
Declare Function ReleaseDevice Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi：

```
Function ReleaseDevice(hDevice : Integer) : Boolean;
                        StdCall; External 'USB2851' 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GetLastErrorEx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++ & C++Builder：

```
BOOL InitDeviceAD(HANDLE hDevice,
                  USB2851_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceAD Lib "USB2851" (ByVal hDevice As Long, _
                                             ByRef pADPara As USB2851_PARA _AD) As Boolean
```

Delphi:

```
Function InitDeviceAD( hDevice : Integer;
                       pADPara:USB2851_PARA_AD):Boolean;
                       StdCall; External 'USB2851' Name ' InitDeviceAD ';
```

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。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。

参数：

hDevice 设备对象句柄，它应由设备的CreateDevice创建。

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

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

♦ 批量读取 USB 设备上的 AD 数据

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceAD (HANDLE hDevice,
                   USHORT ADBuffer[],
                   LONG nReadSizeWords,
```

PLONG nRetSizeWords = NULL)

Visual Basic:

```
Declare Function ReadDeviceAD Lib "USB2851" (ByVal hDevice As Long, _
                                             ByRef ADBuffer As Integer, _
                                             ByVal nReadSizeWords As Long, _
                                             Optional ByRef nRetSizeWords As Long = 0) As Boolean
```

Delphi:

```
Function ReadDeviceAD( hDevice : Integer;
                       ADBuffer : SmallInt;
                       nReadSizeBytes:LongWord;
                       nRetSizeWords : Pointer = 0) : Boolean;
                       StdCall; External 'USB2851' Name 'ReadDeviceAD';
```

LabVIEW:

请参考相关演示程序。

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。

hDevice设备对象句柄，它应由 CreateDevice创建。

ADBuffer 用户数据缓冲区地址。接受的是从设备上采集的 LSB 原码数据，关于如何将 LSB 原码数据转换成电压值，请参考《数据格式转换与排列规则》章节。

nReadSizeWords 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 在当前操作中该函数实际读取的点数。只有当函数成功返回时该参数值才有意义，而当函数返回失败时，则该参数的值与调用此函数前的值相等，不会因为函数被调用而改变，因此最好在读取 AD 数据前，将此参数值赋初值 0。需要注意的是在函数成功返回后，若此参数值等于 0，则需要重新调用此函数读取 AD 数据，直到此参数的值不等于 0 为止。

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 ReadDeviceAD ReleaseDevice

◆ **释放设备上的 AD 部件**

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseDeviceAD(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDeviceAD Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi：

```
Function ReleaseDeviceAD (hDevice : Integer) : Boolean;
                          StdCall; External 'USB2851' Name ' ReleaseDeviceAD ';
```

LabVIEW:

请参考相关演示程序。

功能：释放设备上的 AD 部件。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

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

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

♦ **以上函数一般调用顺序**

① CreateDevice
② InitDeviceAD
③ ReadDeviceAD
④ ReleaseDeviceAD
⑤ ReleaseDevice

注明：用户可以反复执行第③步，以实现高速连续不间断大容量采集。

第四节、AD 硬件参数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,
                USB2851_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "USB2851" (ByVal hDevice As Long, _
                                           ByRef pADPara As USB2851_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD (hDevice : Integer;
                     pADPara : USB2851_PARA_AD) : Boolean;
                     StdCall; External 'USB2851' Name ' LoadParaAD ';
```

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USB2851_PARA_AD 的结构指针类型，它负责返回USB硬件参数值，关于结构指针类型USB2851_PARA_AD请参考USB2851.h或USB2851.Bas或USB2851.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **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,
                 USB2851_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "USB2851" (ByVal hDevice As Long, _
                                           ByRef pADPara As USB2851_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
                     pADPara : USB2851_PARA_AD) : Boolean;
                     StdCall; External 'USB2851' Name ' SaveParaAD ';
```

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USB2851_PARA_AD的详细介绍请参考USB2851.h或USB2851.Bas或USB2851.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

Viusal C++ & C++ Builder:

```
BOOL ResetParaAD (HANDLE hDevice,
                  USB2851_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "USB2851" (ByVal hDevice As Long, _
                                            ByRef pADPara As USB2851_PARA_AD) As Boolean
```

Delphi:

```
Function ResetParaAD (hDevice : Integer;
                      pADPara : USB2851_PARA_AD) : Boolean;
                      StdCall; External 'USB2851' Name ' ResetParaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无法确定错误原因的后果。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara设备硬件参数，它负责在参数被复位后返回其复位后的值。关于USB2851_PARA_AD的详细介绍请参考USB2851.h或USB2851.Bas或USB2851.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ResetParaAD ReleaseDevice

注意：在您编写工程应用软件时，若要更方便的保存和读取您特有的软件参数，请不防使用我们为您提供的辅助函数：SaveParaInt、 LoadParaInt、SaveParaString、LoadParaString，详细说明请参考共用函数介绍章节中的《各种参数保存和读取函数原型说明》。

第五节、DA 模拟量输出操作函数原型说明

♦ **输出模拟信号到指定通道**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function WriteDeviceDA Lib "USB2851" (ByVal hDevice As Long,
                                              ByVal nDAData As Integer, _
                                              ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceDA(hDevice:Integer;
                       nDAData : SmallInt;
```

nDAChannel : Integer) : Boolean;
StdCall; External 'USB2851' Name ' WriteDeviceDA ';

LabVIEW:

请参考相关演示程序。

功能：设置指定通道的模拟量输出量程范围。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

nDAChannel 需要指定的模拟量通道号，其取值范围为[0, 3]。

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

相关函数： CreateDevice ReleaseDevice

♦ **以上函数调用一般顺序**

① CreateDevice
② WriteDeviceDA
③ ReleaseDevice

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

第六节、DIO 数字开关量输入输出操作函数原型说明

♦ **开关量输入**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

Declare Function GetDeviceDI Lib "USB2851" (ByVal hDevice As Long, _
ByVal bDISts (0 to 7)As Byte）As Boolean

Delphi:

Function GetDeviceDI (hDevice : Integer;
bDISts: Pointer):Boolean;
StdCall; External 'USB2851' Name ' GetDeviceDI ';

LabVIEW:

请参考相关演示程序。

功能：负责将 USB 设备上的输入开关量状态读入到 bDISts[x]数组参数中。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

bDISts 八路开关量输入状态的参数结构，共有 8 个元素，分别对应于 DI0～DI7 路开关量输入状态位。如果bDISts[0]等于“1”则表示 0 通道处于开状态。若为“0”则 0 通道为关状态。其他同理。

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

相关函数： CreateDevice SetDeviceDO ReleaseDevice

♦ **开关量输出**

函数原型：

Visual C++ & C++Builder:

BOOL SetDeviceDO (HANDLE hDevice,
BYTE bDOSts[8)

Visual Basic:

Declare Function SetDeviceDO Lib "USB2851" (ByVal hDevice As Long, _

ByVal bDOSts(0 to 7) As Byte) As Boolean

Delphi:

Function SetDeviceDO (hDevice : Integer;
bDOSts : Pointer) : Boolean;
StdCall; External 'USB2851' Name ' SetDeviceDO ';

LabVIEW:

请参考相关演示程序。

功能：负责将 USB 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

bDISts 八路开关量输出状态的参数结构，共有 8 个元素，分别对应于 DO0～DO7 路数字量输出状态位。比如置 DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数数组中的每个元素赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

- **回读数字量输出状态**

函数原型：

Visual C++ & C++Builder:

BOOL RetDeviceDO (HANDLE hDevice,
BYTE bDOSts[8])

Visual Basic:

Declare Function RetDeviceDO Lib "USB2851" (ByVal hDevice As Long, _
ByVal bDOSts(0 to 7) As Byte) As Boolean

Delphi:

Function RetDeviceDO (hDevice : Integer;
bDOSts : Pointer) : Boolean;
StdCall; External 'USB2851' Name ' RetDeviceDO ';

LabVIEW:

请参考相关演示程序。

功能：负责将 USB 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由CreateDevic或CreateDeviceEx创建。

bDISts 获得开关量输出状态。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

- **以上函数调用一般顺序**

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

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

第七节、以太网控制函数原型说明

- **设置以太网参数**

函数原型：

Visual C++ & C++Builder:

BOOL SetNetCfg(HANDLE hDevice,

PDEVICE_NET_INFO pNetInfo)

Visual Basic:

Declare Function SetNetCfg Lib "USB2851" (ByVal hDevice As Long, _
ByRef pNetInfo As PDEVICE_NET_INFO) As Boolean

Delphi:

Function SetNetCfg (hDevice : Integer;
pNetInfo: Pointer) : Boolean;
StdCall; External 'USB2851' Name ' SetNetCfg ';

LabVIEW：

请参考相关演示程序。

功能：设置以太网参数。

参数：

hDevice设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。

pNetInfo 网络配置参数结构体。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetNetCfg ReleaseDevice

◆ **获得设备的以太网设置参数**

函数原型：

Visual C++ & C++Builder:

BOOL GetNetCfg (HANDLE hDevice,
PDEVICE_NET_INFO pNetInfo)

Visual Basic:

Declare Function GetNetCfg Lib "USB2851" (ByVal hDevice As Long, _
ByRef pNetInfo As PDEVICE_NET_INFO) As Boolean

Delphi:

Function GetNetCfg (hDevice : Integer;
pNetInfo: Pointer) : Boolean;
StdCall; External 'USB2851' Name ' GetNetCfg ';

LabVIEW：

请参考相关演示程序。

功能：获得设备的以太网设置参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice或 CreateDeviceEx创建。

pNetInfo 网络配置参数结构体。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetNetCfg ReleaseDevice

第四章 以太网设备操作函数接口介绍

第一节、设备驱动接口函数总列表（每个函数省略了前缀“USB2851E_”）

函数名	函数功能	备注
① 以太网操作函数		
CreateDevice	创建对象	
ReleaseDevice	释放设备对象	
GetDeviceVer	获得版本	
SetNetCfg	设置以太网参数	
GetNetCfg	获得设备的以太网设置参数	

②AD 采样操作函数		
InitDeviceAD	初始化 USB 设备 AD 部件，准备传数	
StartDeviceAD	启动 AD 设备	
ReadDeviceAD	连续批量读取 USB 设备上的 AD 数据	
StopDeviceAD	停止 AD 采集，释放 AD 对象所占资源	
ReleaseDeviceAD	释放 USB 设备对象中的 AD 部件	
③ DA 输出操作函数		
WriteDeviceDA	输出模拟信号到指定通道	
④ 开关量函数		
GetDeviceDI	开关输入函数	
SetDeviceDO	开关输出函数	
RetDeviceDO	回读数字量输出状态	

第二节、以太网控制函数原型说明

- **创建设备**

函数原型：

Visual C++ & C++Builder:

```
HANDLE CreateDevice (LPCSTR strIPAddr,
                     LONG lPort,
                     LONG lSendTimeout,
                     LONG lRecvTimeout)
```

Visual Basic:

```
Declare Function CreateDevice Lib "USB2851" (ByVal strIPAddr As String, _
                                             ByVal lPort As Long, _
                                             ByVal lSendTimeout As Long, _
                                             ByVal lRecvTimeout As Long) As Long
```

Delphi:

```
Function CreateDevice (strIPAddr: String;
                       lPort: LongInt;
                       lSendTimeout : LongInt;
                       lRecvTimeout : LongInt) : Integer;
                       StdCall; External 'USB2851' Name ' CreateDevice ';
```

LabVIEW：

请参考相关演示程序。

功能：创建设备。

参数：

strIPAddr　IP 地址。

1Port　端口号。

1SendTimeout 发送超时时间。

1RecvTimeout 接收超时时间。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数：　ReleaseDevice

- **释放设备对象**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReleaseDevice(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDevice Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function ReleaseDevice (hDevice : Integer):Boolean;
                        StdCall; External 'USB2851' Name ' ReleaseDevice ';
```

LabView:

请参考相关演示程序。

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。

相关函数：CreateDevice

◆ **获得版本号**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceVer (HANDLE hDevice，
                   PLONG plVerNum)
```

Visual Basic:

```
Declare Function GetDeviceVer Lib "USB2851" (ByVal hDevice As Long, _
                                             ByRef plVerNum As Long) As Boolean
```

Delphi:

```
Function GetDeviceVer (hDevice : Integer;
                       plVerNum : Pointer):Boolean;
                       StdCall; External 'USB2851' Name ' GetDeviceVer ';
```

LabView:

请参考相关演示程序。

功能：获得版本号。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

PlVerNum 版本号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice ReleaseDevice

◆ **设置以太网参数**

函数原型：

Visual C++ & C++Builder:

```
BOOL SetNetCfg(HANDLE hDevice,
               PDEVICE_NET_INFO pNetInfo)
```

Visual Basic:

```
Declare Function SetNetCfg Lib "USB2851" (ByVal hDevice As Long, _
                                          ByRef pNetInfo As PDEVICE_NET_INFO) As Boolean
```

Delphi:

```
Function SetNetCfg (hDevice : Integer;
                    pNetInfo: Pointer) : Boolean;
                    StdCall; External 'USB2851' Name ' SetNetCfg ';
```

LabVIEW:

请参考相关演示程序。

功能：设置以太网参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pNetInfo 网络配置参数结构体。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetNetCfg

说明：由简易程序 NetworkConfig 可以获得、设置以太网参数（IP 地址、子网掩码、网关、MAC 地址、TCP 端口号），请谨慎操作，如果因设置错误导致不能够采用以太网连接设备，请复位硬件设备。默认 IP 地址是 192.168.2.80，端口号是 502。

- **获得设备的以太网设置参数**

函数原型：

Visual C++ & C++Builder:

```
BOOL GetNetCfg (HANDLE hDevice,
                PDEVICE_NET_INFO pNetInfo)
```

Visual Basic:

```
Declare Function GetNetCfg Lib "USB2851" (ByVal hDevice As Long, _
                                          ByRef pNetInfo As PDEVICE_NET_INFO) As Boolean
```

Delphi:

```
Function GetNetCfg (hDevice : Integer;
                    pNetInfo: Pointer) : Boolean;
                    StdCall; External 'USB2851' Name ' GetNetCfg ';
```

LabVIEW：

请参考相关演示程序。

功能：获得设备的以太网设置参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

pNetInfo 网络配置参数结构体。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetNetCfg

第三节、AD 采样操作函数原型说明

- **初始化设备对象**

函数原型:

Visual C++ & C++Builder：

```
BOOL InitDeviceAD(HANDLE hDevice,
                  USB2851_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceAD Lib "USB2851" (ByVal hDevice As Long, _
                                             ByRef pADPara As USB2851_PARA _AD) As Boolean
```

Delphi:

```
Function InitDeviceAD (hDevice : Integer; pADPara:USB2851_PARA_AD):Boolean;
                       StdCall; External 'USB2851' Name ' InitDeviceAD ';
```

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，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。

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

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

相关函数： CreateDevice ReadDeviceAD ReleaseDevice

◆ **启动设备**

函数原型:

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StartDeviceAD Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function StartDeviceAD (hDevice : Integer):Boolean;
                        StdCall; External 'USB2851' Name ' StartDeviceAD ';
```

LabView:

请参考相关演示程序。

功能：启动设备。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReleaseDevice

◆ **批量读取设备上的 AD 数据**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceAD (HANDLE hDevice,
                   USHORT ADBuffer[],
                   LONG nReadSizeWords,
                   PLONG nRetSizeWords = NULL)
```

Visual Basic:

```
Declare Function ReadDeviceAD Lib "USB2851" (ByVal hDevice As Long, _
                                             ByRef ADBuffer As Integer, _
                                             ByVal nReadSizeWords As Long, _
                                             Optional ByRef nRetSizeWords As Long = 0) As Boolean
```

Delphi:

```
Function ReadDeviceAD ( hDevice : Integer;
                        ADBuffer : SmallInt;
                        nReadSizeBytes:LongWord;
                        nRetSizeWords : Pointer = 0) : Boolean;
                        StdCall; External 'USB2851' Name ' ReadDeviceAD ';
```

LabView:

请参考相关演示程序。

功能：读取设备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。

ADBuffer 用户数据缓冲区地址。接受的是从设备上采集的LSB原码数据，关于如何将LSB原码数据转换成电压值，请参考《数据格式转换与排列规则》章节。

nReadSizeWords 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 在当前操作中该函数实际读取的点数。只有当函数成功返回时该参数值才有意义，而当函数返回

失败时，则该参数的值与调用此函数前的值相等，不会因为函数被调用而改变，因此最好在读取 AD 数据前，将此参数值赋初值 0。需要注意的是在函数成功返回后，若此参数值等于 0，则需要重新调用此函数读取 AD 数据，直到此参数的值不等于 0 为止。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice InitDeviceAD ReleaseDevice

- **暂停 AD 设备**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function StopDeviceAD Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function StopDeviceAD (hDevice : Integer):Boolean;
                        StdCall; External 'USB2851' Name ' StopDeviceAD ';
```

LabView:

请参考相关演示程序。

功能：暂停 AD 设备。

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。

相关函数： CreateDevice InitDeviceAD ReleaseDevice

- **释放设备对象中的 AD 部件**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReleaseDeviceAD(HANDLE hDevice)
```

Visual Basic:

```
Declare Function ReleaseDeviceAD Lib "USB2851" (ByVal hDevice As Long) As Boolean
```

Delphi:

```
Function ReleaseDeviceAD (hDevice : Integer):Boolean;
                          StdCall; External 'USB2851' Name ' ReleaseDeviceAD ';
```

LabView:

请参考相关演示程序。

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。

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。

第四节、DA 模拟量输出操作函数原型说明

♦ **输出模拟信号到指定通道**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function WriteDeviceDA Lib "USB2851" (ByVal hDevice As Long, _
                                              ByVal nDAData As Integer, _
                                              ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceDA (hDevice : Integer;
                        nDAData : SmallInt;
                        nDAChannel : Integer) : Boolean;
                        StdCall; External 'USB2851' Name ' WriteDeviceDA ';
```

LabVIEW：

请参考相关演示程序。

功能：设置指定通道的模拟量输出量程范围。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

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

nDAChannel 需要指定的模拟量通道号，其取值范围为[0, 3]。

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

相关函数: CreateDevice ReleaseDevice

第五节、DIO 数字开关量输入输出简易操作函数原型说明

♦ **八路开关量输入**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function GetDeviceDI Lib "USB2851" (ByVal hDevice As Long, _
                                            ByVal bDISts(0 to 7 ) As Long) As Boolean
```

Delphi:

```
Function GetDeviceDI ( hDevice : Integer; bDISts:Pointer):Boolean;
                       StdCall; External 'USB2851' Name ' GetDeviceDI ';
```

LabVIEW：

请参考相关演示程序。

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。

bDISts八路开关量输入状态的参数结构，共有 8 个成员变量，分别对应于DI0~DI7 路开关量输入状态位。如果bDISts[0]为“1”则使 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。具体定义请参考《DI数字开关量

输入参数介绍》章节。

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

相关函数： CreateDevice SetDeviceDO ReleaseDevice

♦ **八路开关量输出**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function SetDeviceDO Lib "USB2851" (ByVal hDevice As Long, _
                                            ByVal bDOSts(0 to 7 )As Long) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer; bDOSts:Pointer):Boolean;
                      StdCall; External 'USB2851' Name ' SetDeviceDO ';
```

LabVIEW：

请参考相关演示程序。

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。

bDOSts 八路开关量输出状态的参数结构，共有 8 个成员变量，分别对应于DO0~DO7 路开关量输出状态位。比如置bDOSts[0]为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的DO0 至DO7 共 8 个成员变量赋初值，其值必须为“1”或“0”。具体定义请参考《DO数字开关量输出参数介绍》。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

♦ **回读数字量输出状态**

函数原型：

Visual C++ & C++Builder:

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

Visual Basic:

```
Declare Function RetDeviceDO Lib "USB2851" (ByVal hDevice As Long, _
                                            ByVal DOSts(0 to 7) As Byte) As Boolean
```

Delphi:

```
Function RetDeviceDO (hDevice : Integer;
                      DOSts : Pointer) : Boolean;
                      StdCall; External 'USB2851' Name ' RetDeviceDO ';
```

LabVIEW：

请参考相关演示程序。

功能：负责将 USB 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

bDOSts[x] 获得开关输出状态。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI ReleaseDevice

♦ **以上函数调用一般顺序**

① CreateDevice

② SetDeviceDO（或 GetDeviceDI，当然这两个函数也可同时进行）

③ ReleaseDevice

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

第五章 硬件参数结构

第一节、AD 硬件参数介绍（USB2851_PARA_AD）

Visual C++ & C++Builder:

```
typedef struct _USB2851_PARA_AD
{
    LONG CheckStsMode;        // 检查状态模式
    LONG ADMode;              // AD 模式选择（连续/分组方式）
    LONG FristChannel;        // 首通道[0, 15]
    LONG LastChannel;         // 末通道[0, 15]，要求末通道必须大于或等于首通道
    LONG Frequency;           // 采集频率,单位为 Hz, [31, 250000]
    LONG GroupInterval;       // 分组时的组间间隔（单位：微秒） [1, 32767]
    LONG LoopsOfGroup;        // 组内循环次数[1, 255]
    LONG Gains;               // 增益设置
    LONG TriggerMode;         // 触发模式选择（软件触发/后触发）
    LONG TriggerType ;        // 触发类型选择(边沿触发/脉冲触发)
    LONG TriggerSource;       // 触发源选择（ATR/DTR）
    LONG TriggerDir;          // 触发方向选择(正向/负向触发)
    LONG TrigLevelVolt;       // 触发电平(0-10000mV)
    LONG TrigWindow;          // 触发灵敏度（1-255），单位：0.5 微秒
    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）
} USB2851_PARA_AD, *USB2851_PARA_AD;
```

Visual Basic：

```
Private Type USB2851_PARA_AD
    CheckStsMode As Long      ' 检查状态模式
    ADMode As Long            'AD 模式选择（连续/分组方式）
    FristChannel As Long      ' 首通道[0, 15]
    LastChanne As Long        ' 末通道[0, 15]，要求末通道必须大于或等于首通道
    Frequency As Long         ' 采集频率,单位为 Hz, [31, 250000]
    GroupInterval As Long     ' 分组时的组间间隔（单位：微秒） [1, 32767]
    LoopsOfGroup As Long      ' 组内循环次数[1, 255]
    Gains As Long             ' 增益设置
    TriggerMode As Long       ' 触发模式选择（软件触发/后触发）
    TriggerType As Long       ' 触发类型选择(边沿触发/脉冲触发)
    TriggerSource As Long     ' 触发源选择（ATR/DTR）
    TriggerDir As Long        ' 触发方向选择(正向/负向触发)
    TrigLevelVolt As Long     ' 触发电平(0-10000mV)
    TrigWindow As Long        ' 触发灵敏度（1-255），单位：0.5 微秒
    ClockSource As Long       ' 时钟源选择(内/外时钟源)
    bClockOutput As Long      ' 允许时钟输出
    GroundingMode As Long     ' 接地方式（单端或双端选择）
End Type
```

Delphi:

```
Type   // 定义结构体数据类型
USB2851_PARA_AD = ^USB2851_PARA_AD; // 指针类型结构
USB2851_PARA_AD = record                    // 标记为记录型
    CheckStsMode  : LontInt;          // 检查状态模式
    ADMode : LontInt;                 // AD 模式选择(连续/分组方式)
    FirstChannel : LontInt;           // 首通道[0, 15]
    LastChannel : LontInt;            // 末通道[0, 15]，要求末通道必须大于或等于首通道
    Frequency : LontInt;              // 采集频率，单位为 Hz，[31, 250000]
    GroupInterval : LontInt;          // 分组时的组间间隔 (单位：微秒)  [1, 32767]
    LoopsOfGroup : LontInt;           // 组内循环次数[1, 255]
    Gains : LontInt;                  // 增益设置
    TriggerMode : LontInt;            // 触发模式选择（软件触发/后触发）
    TriggerType : LontInt;            // 触发类型选择(边沿触发/脉冲触发)
    TriggerSource: LontInt;           // 触发源选择（ATR/DTR）
    TriggerDir : LontInt;             // 触发方向选择(正向/负向触发)
    TrigLevelVolt: LontInt;           // 触发电平(0~10000mV)
    TrigWindow : LongInt;;            // 触发灵敏度（1-255），单位：0.5 微秒
    ClockSource : LongInt;            // 时钟源选择(内/外时钟源)
    bClockSourceDir: LongInt;         // 允许时钟输出
    GroundingMode : LongInt;          // 接地方式（单端或双端选择）
End;
```

LabVIEW:

请参考相关演示程序。

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nitDeviceAD函数完成。

CheckStsMode 查询 FIFO 状态的模式，它用于同步 AD 的采样时序。由于有不同的采样频率，因此写入 FIFO 数据的速度也不一样，所以需要依据 FIFO 的半满或者是非空标志同步对 FIFO 数据的读取。如果选择半满查询模式，则在启动 AD 采样后不能马上读取 FIFO 中的数据，而是要等到 FIFO 的半满状态建立时才能读取数据，此种模式用于大批量高速采样情况下(实时性要求不大)。如果选择非空查询模式，则在启动 AD 采样后，只要 FIFO 中不为空，那怕只有一个数据时也能读取，此种模式用于较低速，但快速取得小批量数据的情况下(实时性要求较高)。

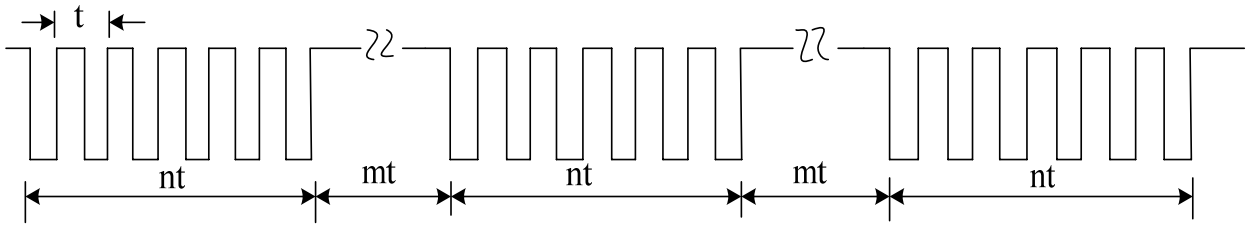
常量名	常量值	功能定义
USB2851_CHKSTSMODE_HALF	0x0000	查询 FIFO 半满标志（建议高频率采集时使用）
USB2851_CHKSTSMODE_NPT	0x0001	查询 FIFO 非空标志（建议高实时采集时使用）

ADMode AD 采样模式。它的取值如下表：

常量名	常量值	功能定义
USB2851_ADMODE_SEQUENCE	0x00	连续采集模式
USB2851_ADMODE_GROU[P	0x01	分组采集模式

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

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



说明： t=1/Frequency
mt = GroupInterval
n = ChannelCount

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

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

Frequency AD 采样频率，本设备的频率取值范围为[31,250KHz]。

注意：

在内时钟(即ClockSource = USB2851_CLOCKSRC_IN)方式下：

若连续采集(即ADMode = USB2851_ADMODE_SEQUENCE)时，此参数控制各处通道间的采样频率。若分组采集(即ADMode= USB2851_ADMODE_GROUP)时，则此参数控制各组组内的采样频率，而组间时间则由GroupInterval决定。

在外时钟(即 ClockSource = USB2851_CLOCKSRC_OUT)方式下：

若连续采集(即ADMode = USB2851_ADMODE_SEQUENCE)时，此参数自动失效，因为外时钟的频率代替了此参数设定的频率。若为分组采集(即ADMode= USB2851_ADMODE_GROUP)时，则该参数控制各组组内的采样频率，而外时钟则是每组的触发频率。此时，GroupInterval参数无效。

GroupInterval 组间间隔，单位微秒 uS，其范围[1, 32767]，通常由用户确定。但是一般情况下，此间隔时间应不小于组内相邻两通道的间隔。在内时钟连续采集模式和外时钟模式下，则参数无效。

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

Gain AD 采样程控增益。

常量名	常量值	功能定义
USB2851_GAINS_1MULT	0x00	1 倍增益
USB2851_GAINS_2MULT	0x01	2 倍增益
USB2851_GAINS_4MULT	0x02	4 倍增益
USB2851_GAINS_8MULT	0x03	8 倍增益

TriggerMode AD 触发模式。

常量名	常量值	功能定义
USB2851_TRIGMODE_SOFT	0x00	软件触发（属于内触发）
USB2851_TRIGMODE_POST	0x01	硬件后触发（属于外触发）

TriggerSource 触发源选择。

常量名	常量值	功能定义
USB2851_TRIGSRC_ATR	0x00	选择 ATR 作为触发源
USB2851_TRIGSRC_DTR	0x01	选择 DTR 作为触发源

TrigLevelVolt 触发电平（0～10000mV）。

TrigWindow 触发灵敏窗时间值，取值范围为(1-255)，单位为 0.5 微秒。

TriggerType AD 触发类型。

常量名	常量值	功能定义
USB2851_TRIGTYPE_EDGE	0x00	边沿触发
USB2851_TRIGTYPE_PULSE	0x01	脉冲触发（电平）

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
USB2851_TRIGDIR_NEGATIVE	0x00	负向触发（低脉冲/下降沿触发）
USB2851_TRIGDIR_POSITIVE	0x01	正向触发（高脉冲/上升沿触发）
USB2851_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：USB2851_TRIGDIR_POSIT_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。而在电平类型下，无论正电平还是负电平均触发。

ClockSource　AD 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
USB2851_CLOCKSRC_IN	0x00	内部时钟定时触发
USB2851_CLOCKSRC_OUT	0x01	外部时钟定时触发

GroundingMode　AD 接地方式的选择。

常量名	常量值	功能定义
USB2851_GNDMODE_SE	0x00	单端方式
USB2851_GNDMODE_DI	0x01	双端方式

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ReleaseDevice

第二节、设备网络配置结构参数介绍（DEVICE_NET_INFO）

Visual C++ & C++Builder:

```
typedef struct _DEVICE_NET_INFO
{
    char strIP[16];                 // IP 地址, "192.168.2.70"
    char strSubnetMask[16];         //  子网掩码, "255.255.255.255"
    char strGateway[16];            //  网关, "192.168.2.1"
    char strMAC[20];                //  网卡物理地址,    "00-01-02-03-04-05",用户一般不可修改
    WORD  wTCPPort;                 // TCP  端口
}DEVICE_NET_INFO, *DEVICE_NET_INFO;
```

Visual Basic：

```
Private Type DEVICE_NET_INFO
    strip(0 to 15) As Byte                  ' IP 地址, "192.168.2.70"
    strSubnetMask(0 to 15) As Byte          '  子网掩码, "255.255.255.255"
    strGateway(0 to 15) As Byte             '  网关, "192.168.2.1"
    strMAC(0 to 19) As Byte                 '  网卡物理地址,    "00-01-02-03-04-05",用户一般不可修改
    wTCPPort As Integer                     ' TCP  端口
End Type
```

Delphi:

```
Type   //  定义结构体数据类型
   DEVICE_NET_INFO = ^ DEVICE_NET_INFO;    //  指针类型结构
   DEVICE_NET_INFO = record                //  标记为记录型
       strIP: Array[0..15] of char;        // IP 地址, "192.168.2.70"
       strSubnetMask: Array[0.. 15] of char;   //  子网掩码, "255.255.255.255"
       strGateway: Array[0.. 15] of char;      //  网关, "192.168.2.1"
       strMAC: Array[0..19] of char;           //  网卡物理地址,    "00-01-02-03-04-05",用户一般不可修改
       wTCPPort: Word;                     // TCP  端口
```

第六章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/65536) * (ADBuffer[0]&0xFFF) – 10000.00	[-10000, +9997.55]
±5000mV	Volt = (10000.00/65536) * (ADBuffer[0]&0xFFF) – 5000.00	[-5000, +4998.77]
±2500mV	Volt = (5000.00/65536) * (ADBuffer[0]&0xFFF) – 2500.00	[-2500, +2499.38]
0～10000mV	Volt = (10000.00/65536) * (ADBuffer[0]&0xFFF)	[0, +9998.77]
0～5000mV	Volt = (10000.00/65536) * (ADBuffer[0]&0xFFF)	[0, +4997.55]

换算举例：（设量程为±10000mV, 这里只转换第一个点）

Visual C++&C++Builder：

```
USHORT Lsb;                          // 定义存放标准 LSB 原码的变量
float Volt;                          // 定义存放转换后的电压值的变量
float PerLsbVolt = 20000.00/65536; // 求出每个 LSB 原码单位电压值
Lsb = ADBuffer[0];
Volt = PerLsbVolt *  Lsb – 10000.00;    // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

Visual Basic:

```
Dim Lsb As Long                    ' 定义存放标准 LSB 原码的变量
Dim Volt As Single                 ' 定义存放转换后的电压值的变量
Dim PerLsbVolt As Single
PerLsbVolt = 20000.00/65536        ' 求出每个 LSB 原码单位电压值
Lsb = ADBuffer(0) AND 65535        ' 将其转换成无符号 16 位有效数据
Volt = PerLsbVolt * Lsb-10000.00   ' 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

Delphi:

```
Lsb : Word;                        // 定义存放标准 LSB 原码的变量
Volt  ：Single;                    // 定义存放转换后的电压值的变量
PerLsbVolt : Single;
PerLsbVolt := 20000.00/65536;    // 求出每个 LSB 原码单位电压值
Lsb := ADBuffer[0];
 Volt  ：= PerLsbVolt * Lsb-10000.00; // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，当通道总数首末通道相等时，假如此时首末通道=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（假如 FirstChannel=0，LastChannel=1）：

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（假如 FirstChannel=0，LastChannel=1）：

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的问题，尤其是在任意通道数采集时。否则，用户无将

规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？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，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断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于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，而且还高效呢？ 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。

<table>
<tr><td>数据序列</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>...</td></tr>
<tr><td>通道序列</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>1</td><td>2</td><td>3</td><td>...</td></tr>
<tr><td>方法 1</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>0</td><td>1</td><td>2</td><td></td></tr>
<tr><td>缓冲区号</td><td colspan="6">第一段缓冲</td><td colspan="6">第二段缓冲区</td><td colspan="6">第三段缓冲区</td><td colspan="6">第 n 段缓冲</td></tr>
<tr><td rowspan="2">方法 2</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td></td></tr>
<tr><td colspan="4">第一段缓冲区</td><td colspan="4">第二段缓冲区</td><td colspan="4">第三段缓冲区</td><td colspan="4">第四段缓冲区</td><td colspan="4">第五段缓冲区</td><td colspan="2">第 n 段缓冲</td></tr>
</table>

第三节、DA 电压值转换成 LSB 原码数据的换算方法

量程（伏）	计算机语言换算公式	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]

第七章 上层用户函数接口应用实例

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

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**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[AD 采集演示源程序]**

二、怎样使用SetDeviceDO和GetDeviceDI 函数进行开关量输入输出操作

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**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[开关量演示源程序]**

其默认存放路径为：系统盘\ART\USB2851\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 USB2851.h 和 ADDoc.cpp)。

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨**[高级代码演示]**

其默认存放路径为：系统盘\ART\USB2851\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第八章 基于 USB 总线的大容量连续数据采集详解

与ISA、USB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，因此可以保证其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，此时它基本不消耗 CPU 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两维数组如ADBuffer [SegmentCount][SegmentSize]，我们将SegmentSize视为数据采集线程每次采集的数据长度，SegmentCount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SegmentCount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SegmentSize长度的数据缓冲区。需要注意的是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SegmentCount加 1，（注意SegmentCount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SegmentCount加 1，直到ReadIndex等于 31 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SegmentCount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，具体处理哪个缓冲区由CurrentIndex指向。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SegmentCount加以判断，观察其值是否大于了 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。

图 7.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面

填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程填充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

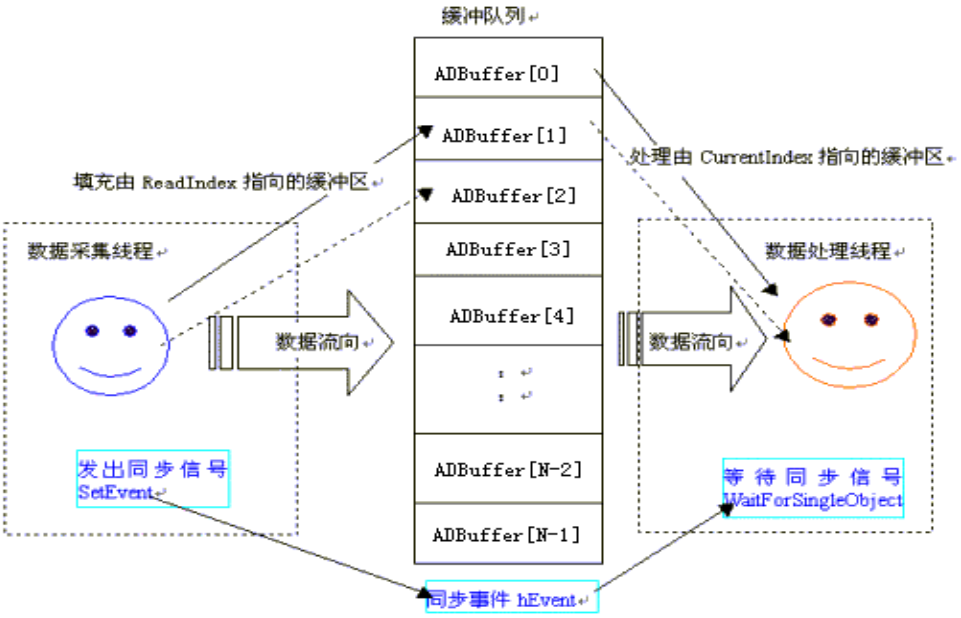


图 8.1

下面用 Visual C++程序举例说明：

使用ReadDeviceAD函数读取设备上的AD数据

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, ADThread.h 和 ADThread.cpp)。

[程序]→**[阿尔泰测控演示系统]**→**[Microsoft Visual C++]**→**[高级演示程序]**

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
BOOL MyStartDeviceAD(HANDLE hDevice);           // 位于 ADThread.cpp
UINT ReadDataThread_Npt (PVOID pThreadPara)     // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara)       // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice);            // 位于 ADThread.cpp
void CADDoc::StopDeviceAD()                     // 终止采集函数
```

当然用 FIFO 非空标志读取 AD 数据，能获得接近 FIFO 总容量的栈深度，这样用户在两批数据之间，便有更多的时间来处理某些数据。而用半满标志，则最多只能达到 FIFO 总容量的二分之一的栈深度，那么用户在两批数据之间处理数据的时间会相对短些，但是半满读取时，查询 AD 转换标志的时间则最少。当然究意那种方案最好，还得看用户的实际需要。

第九章 共用函数介绍

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“USB2851_”）

函数名	函数功能	备注
① 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateVBThread	在 VB 环境中建立子线程对象	在 VB 中可实现多线程
TerminateVBThread	终止 VB 的子线程	
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	
② 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
③ 各种参数保存和读取函数		
SaveParaInt	保存整型参数到注册表	
LoadParaInt	从注册表中读取整型参数值	
SaveParaString	保存字符参数到注册表	
LoadParaString	从注册表中读取字符参数值	
④ 其他函数		
kbhit	探测用户是否有击键动作	
getch	等待并获取用户击键值	
GetLastErrorEx	取得驱动函数错误信息	

第二节、线程操作函数原型说明

◆ **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 "USB2851" (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 Resume Thread 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 **USB2851.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 "USB2851" (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)

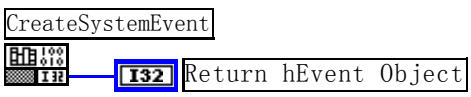
Visual Basic:

Declare Function CreateSystemEvent Lib " USB2851 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'USB2851' 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 " USB2851 " (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;

```
StdCall; External 'USB2851' 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。

第三节、文件对象操作函数原型说明

♦ 创建文件对象

函数原型：

Visual C++ & C++ Builder:

```
HANDLE CreateFileObject (HANDLE hDevice,
                         LPCTSTR NewFileName,
                         int Mode)
```

Visual Basic:

```
Declare Function CreateFileObject Lib "USB2851" (ByVal hDevice As Long, _
                                                 ByVal NewFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer;
                           NewFileName : string;
                           Mode : Integer) : Integer;
                           Stdcall; external 'USB2851' Name ' CreateFileObject ';
```

LabVIEW:

请参见相关演示程序。

功能：初始化设备文件对象，以期待 WriteFile 请求准备文件对象进行文件操作。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

NewFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：“C:\\USB2851\\Data.Dat”，在 Basic 中，其语法格式如：“C:\USB2851\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
USB2851_modeRead	0x0000	只读文件方式
USB2851_modeWrite	0x0001	只写文件方式
USB2851_modeReadWrite	0x0002	既读又写文件方式
USB2851_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
USB2851_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 "USB2851" (ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Byte, _
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                   pDataBuffer : Pointer;
                   nWriteSizeBytes : LongWord) : Boolean;
                   Stdcall; external 'USB2851' 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 OffsetBytes,
              LONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "USB2851" ( ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal OffsetBytes As Long, _
                                          ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile(hFileObject : Integer;
                  pDataBuffer : Pointer;
                  OffsetBytes : LongWord;
                  nReadSizeBytes : LongWord) : Boolean;
                  Stdcall; external 'USB2851' Name ' ReadFile ';
```

LabVIEW:

详见相关演示程序。

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

参数：

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

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

OffsetBytes 指定从文件开始端所偏移的读位置。

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

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

相关函数： CreateFileObject WriteFile ReadFile

ReleaseFile

♦ **设置文件偏移位置**

函数原型：

Visual C++ & C++ Builder:

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

Visual Basic:

```
Declare Function SetFileOffset Lib "USB2851" (ByVal hFileObject As Long,
                                              ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset (hFileObject : Integer;
                        nOffsetBytes : LongWord) : Boolean;
                        Stdcall; external 'USB2851' 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 "USB2851" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject : Integer) : LongWord;
                        Stdcall; external 'USB2851' 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 "USB2851" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
Function ReleaseFile(hFileObject : Integer) : Boolean;
                     Stdcall; external 'USB2851' 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**

Visual C++ & C++ Builder:

```
ULONGLONG GetDiskFreeBytes(LPCTSTR DiskName)
```

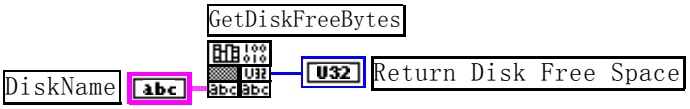
Visual Basic:

```
Declare Function GetDiskFreeBytes Lib "USB2851" (ByVal DiskName As String) As Currency
```

Delphi:

```
Function GetDiskFreeBytes (DiskName : String) : Currency;
                                Stdcall; external 'USB2851' Name ' GetDiskFreeBytes ';
```

LabVIEW:



功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：

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

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

第四节、各种参数保存和读取函数原型说明

- **将整型变量的参数值保存在系统注册表中**

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaInt(HANDLE hDevice,
                 LPCTSTR strParaName,
                 int nValue)
```

Visual Basic:

```
Declare Function SaveParaInt Lib "USB2851" (ByVal hDevice As Long, _
                                            ByVal strParaName As String, _
                                            ByVal nValue As Integer) As Boolean
```

Delphi:

```
Function SaveParaInt( hDevice : Integer;
                      strParaName : String;
                      nValue : Integer) : Boolean;
                      Stdcall; external 'USB2851' Name ' SaveParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB2851\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nValue 整型参数值。它保存在由 strParaName 命名的键项里。

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： SaveParaInt LoadParaInt SaveParaString LoadParaString

♦ **将整型变量的参数值从系统注册表中读出**

函数原型：

Visual C++ & C++ Builder:

```
UINT LoadParaInt(HANDLE hDevice,
                 LPCTSTR strParaName,
                 int nDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaInt Lib "USB2851" (ByVal hDevice As Long, _
                                            ByVal strParaName As String, _
                                            ByVal nDefaultVal As Integer) As Long
```

Delphi:

```
Function LoadParaInt ( hDevice : Integer;
                       strParaName : String;
                       nDefaultVal: Integer) : LongWord;
                       Stdcall; external 'USB2851' Name ' LoadParaInt ';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB2851\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

返回值：若指定的整型参数项存在，则返回其整型值。否则返回由 nDefaultVal 指定的默认值。

相关函数： SaveParaInt LoadParaInt SaveParaString LoadParaString

♦ **将字符变量的参数值保存在系统注册表中**

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaString (HANDLE hDevice,
                     LPCTSTR strParaName,
                     LPCTSTR strParaVal)
```

Visual Basic:

```
Declare Function SaveParaString Lib "USB2851" (ByVal hDevice As Long, _
                                               ByVal strParaName As String, _
                                               ByVal strParaVal As String) As Boolean
```

Delphi:

```
Function SaveParaString (hDevice : Integer;
                         strParaName : String;
                         strParaVal: String) : Boolean;
                         Stdcall; external 'USB2851' Name ' SaveParaString';
```

LabVIEW:

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB2851\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 字符参数值。它保存在由 strParaName 命名的键项里。

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： SaveParaInt LoadParaInt SaveParaString LoadParaString

- **将字符变量的参数值从系统注册表中读出**

函数原型：

Visual C++ & C++ Builder:

```
BOOL LoadParaString (HANDLE hDevice,
                     LPCTSTR strParaName,
                     LPCTSTR strParaVal,
                     LPCTSTR strDefaultVal)
```

Visual Basic:

```
Declare Function LoadParaString Lib "USB2851" (ByVal hDevice As Long, _
                                               ByVal strParaName As String, _
                                               ByVal strParaVal As String, _
                                               ByVal strDefaultVal As String) As Boolean
```

Delphi:

```
Function LoadParaString (hDevice : Integer;
                         strParaName : String;
                         strParaVal : String;
                         strDefaultVal : String) : Boolean;
                         Stdcall; external 'USB2851' Name ' LoadParaString ';
```

LabVIEW:

详见相关演示程序。

功能：将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB2851\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

strDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

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： SaveParaInt LoadParaInt SaveParaString LoadParaString

第五节、其他函数原型说明

- **探测用户是否有按键动作**

函数原型：

Visual C++ & C++ Builder:

```
BOOL kbhit (void)
```

Visual Basic:

```
Declare Function kbhit Lib "USB2851" () As Boolean
```

Delphi:

```
Function kbhit () : Boolean;
                Stdcall; external 'USB2851' Name ' kbhit ';
```

LabVIEW:

详见相关演示程序。

功能：探测用户是否用键盘按键动作，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若自上次探测过后，若用户有键盘按键动作，则返回 TRUE，否则返回 FALSE。

相关函数： getch kbhit

♦ **等待按键动作并返回按键值**

函数原型：

Visual C++ & C++ Builder:

```
char getch (void)
```

Visual Basic:

```
Declare Function getch Lib "USB2851" () As String
```

Delphi:

```
Function getch () : char;
                Stdcall; external 'USB2851' Name 'getch';
```

LabVIEW:

详见相关演示程序。

功能：探等待用户键盘按键并以字符方式返回按键值，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若用户没有按键动作，此函数一直不返回，一旦用户有按键动作，便立即返回，且返回其当前按键值(ACII 码)。

相关函数： getch kbhit

♦ **怎样获取驱动函数错误信息**

函数原型：

Visual C++ & C++ Builder:

```
DWORD GetLastErrorEx( LPCTSTR strFuncName,
                      LPTSTR strErrorMsg)
```

Visual Basic:

```
Declare Function GetLastErrorEx Lib "USB2851" (ByVal strFuncName As String, _
                                               ByVal strErrorMsg As String） As Long
```

Delphi:

```
Function GetLastErrorEx (strFuncName : String;
                         strErrorMsg : String): LongWord;
                         Stdcall; external 'USB2851' Name ' GetLastErrorEx ';
```

LabVIEW:

详见相关演示程序。

功能：将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字串。

参数：

strFuncName 出错函数的名称。注意此函数必须是完整名称。如 AD 初始化函数 USB2851_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“USB2851_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串，该串为字符数组，其分配空间最好不要小于 256 字节。

返回值：返回错误码。

相关函数：无。