# USB2080 驱动程序使用说明书

**For Win98/Me/2000/XP**



有关设备及驱动安装请参考 USB2080Inst.doc 文档。

# 第一章 版权信息



# 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。**

三、哪些函数对您不是必须的？

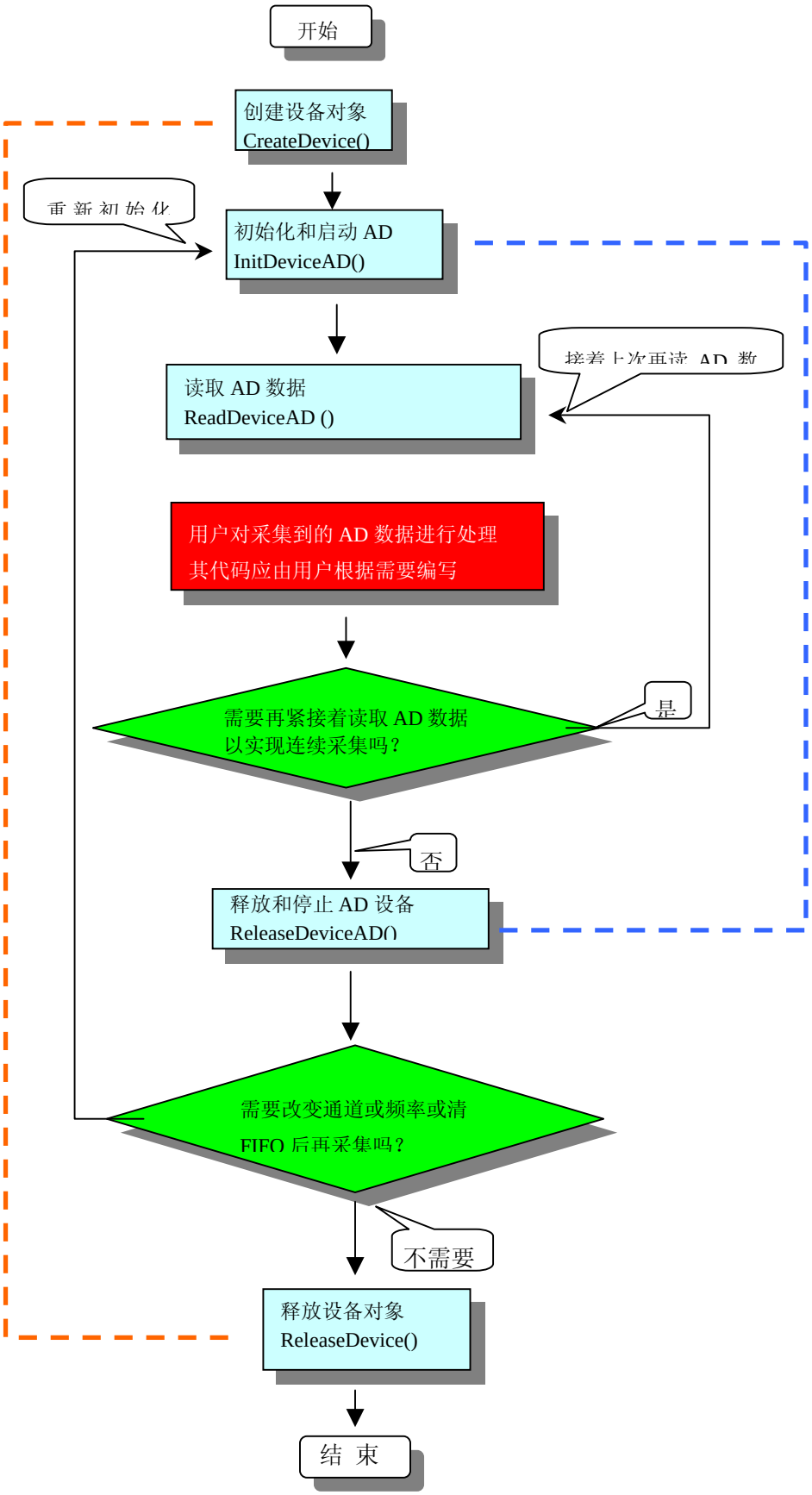



图 2.1.1 AD 采集实现过程

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“USB2080_”）

| 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 | |
| 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 | |
| | | |
| | | |

**使用需知：**

**Visual C++ & C++Builder**：

要使用如下函数关键的问题是：

首先，将 USB2080.h 和 USB2080.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 “USB2080.H”

那么对于导入库 USB2080.lib 文件您则可以不必再加入您的工程，因为 USB2080.h 头文件已帮助自动完成了。

***C++ Builder:***

首先，将 USB2080.h 和 USB2080.lib 文件从 C++Builder 的源程序目录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：

#include “USB2080.H”

那么对于导入库 USB2080.lib 文件您则可以不必再加入您的工程，因为 USB2080.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,在弹出的对话中选择 USB2080.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 USB2080.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：“USB2080”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  USB2080;  // 注意： 在此加入驱动程序接口单元 USB2080
```

***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 USB2080.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 ReadDeviceProAD_NotEmpty 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）**

***Visual C++ & C++Builder：***

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

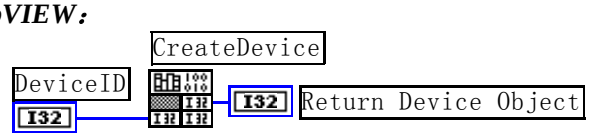
***Visual Basic：***

```
Declare Function CreateDevice Lib  “USB2080”(Optional ByVal DeviceLgcID As long = 0) As long
```

***Delphi：***

```
Function CreateDevice(DeviceLgcID:Integer = 0):Integer;
                    StdCall; External 'USB2080' 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：

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USB2080 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USB2080 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　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

***Visual C++ & C++Builder 程序举例***

```
    :
 HANDLE hDevice;     // 定义设备对象句柄
  hDevice=CreateDevice ( 0 ); // 创建设备对象,并取得设备对象句柄
  if(hDevice==INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
{                return;     // 退出该函数

  )
    :
```

***Visual Basic 程序举例***

```
     :

Dim hDevice As Long    ' 定义设备对象句柄
hDevice = CreateDevice ( 0 )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then    ' 判断设备对象句柄是否有效

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

♦ **创建设备对象函数（物理号）**

***Visual C++ & C++Builder：***

HANDLE CreateDeviceEx(int DevicePhysID=0)

***Visual Basic：***

Declare Function CreateDeviceEx Lib “USB2080”(Optional ByVal DevicePhysID As long = 0) As long

***Delphi：***

```
Function CreateDeviceEx(DevicePhysID:Integer = 0):Integer;
                    StdCall; External 'USB2080' Name ' CreateDevice';
```

***LabVIEW：***

**请参考演示源程序**

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：

DevicePhysID 物理设备 ID( Physic Device Identifier )标识号。由 CreateDevice 函数的 DevieLgcID 参数说明中可以看出，逻辑 ID 号是系统动态自动分配的，即某个已定功能的卡可能在设备链中的位置是不确定的，而在很多场合这可能带来诸多麻烦，比如咱们使用多个卡，如 A、B、C、D 四个卡，构成 256 个通道（64*4），其通道序列为 0-255，每个通道接入不同物理意义的模拟信号，我们要求 A 卡位于 0-63 通道上，B 卡位于 63-127 通道上，C 卡位于 128-191 通道上，而 D 卡则位于 192-255 通道上，而其逻辑设备 ID 号在同一台计计算机上按不同顺序插入会发生变化，即便在不同计算机上按相同顺序插入也可能会因主板制造商的不同定义而发生变化，所以您可能由此无法确定 0-255 的通道分别接入了什么信号。那么如何将各个设备在设备链中的物理位置固定下来呢？那么物理设备 ID 的使用帮您解决了这个问题。它是在卡上提供了一个拔码器 DID，可以由用户为各个设备手动设置不同的物理 ID 号，当调用 CreateDeviceEx 函数时，只需要指定该参数的值与您在拔码器上设定的值一样即可，驱动程序会自动跟踪拔码器值与此相等的设备。

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 CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- **取得本计算机系统中 USB2080 设备的总数量**

函数原型：

***Visual C++ & C++Builder:***

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

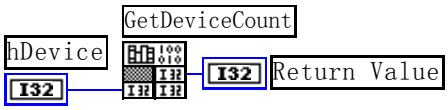
***Visual Basic:***

```
Declare Function GetDeviceCount Lib "USB2080" (ByVal hDevice As Long ) As Long
```

***Delphi：***

```
Function USB2080_GetDeviceCount (hDevice : Integer):Integer;
StdCall; External 'USB2080' Name 'GetDeviceCount';
```

***LabVIEW：***




函数原型：

功能：取得 USB2080 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 USB2080 的数量。

**相关函数：** CreateDevice CreateDeviceEx GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

- 取得该设备当前逻辑 ID 和物理 ID

函数原型：

***Visual C++ & C++Builder:***

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

***Visual Basic:***

```
Declare Function GetDeviceCurrentID Lib "USB2080" (ByVal hDevice As Long,_
                                                   ByRef DeviceLgcID As Long,_
                                                   ByRef DevicePhysIDAs Long ) As Long
```

***Delphi：***

```
Function GetDeviceCurrentID (hDevice : Integer;
                             DeviceLgcID : PLongArray;
                             DevicePhysID : PLongArray):Integer;
StdCall; External 'USB2080' Name 'GetDeviceCurrentID';
```

***LabVIEW：***

请参考演示源程序

函数原型：

功能：取得 USB2080 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 GetLastError 捕获当前错误码,并加以分析。

**相关函数：** CreateDevice　CreateDeviceEx　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　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 "USB2080" (ByVal hDevice As Long ) as Boolean
```

***Delphi:***

```
Function ResetDevice (hDevice : LongInt):Boolean; StdCall; External 'USB2080' Name 'ResetDevice';
```

***LabView:***

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++ & C++Builder:***

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

***Visual Basic:***

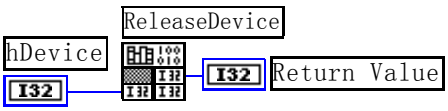
```
Declare Function ReleaseDevice Lib "USB2080" (ByVal hDevice As Long ) As Boolean
```

***Delphi：***

```
Function ReleaseDevice(hDevice : Integer):Boolean;
```

StdCall; External 'USB2080' Name ' ReleaseDevice';

***LabVIEW：***




功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice 设备对象句柄，它应由 CreateDevice 创建。

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

**相关函数：**CreateDevice

应注意的是，CreateDevice 必须和 ReleaseDevice 函数一一对应，即当您执行了一次 CreateDevice 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,
                   PUSB2080_PARA_AD pADPara )
```

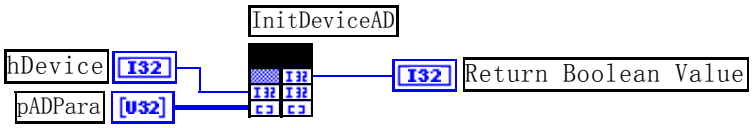
***Visual Basic:***

```
Declare Function InitDeviceAD Lib "USB2080" (ByVal hDevice as Long, _
                  ByRef pADPara as USB2080_PARA _AD
                  ) As Boolean
```

***Delphi:***

```
Function InitDeviceAD(hDevice : Integer; pADPara:PUSB2080_PARA_AD):Boolean;
    StdCall; External 'USB2080' Name 'InitDeviceAD';
```

***LabView:***




功能：它负责初始化设备对象中的 AD 部件,为设备操作就绪有关工作,如预置 AD 采集通道,采样频率等,然后启动 AD 设备开始 AD 采集，随后，用户便可以连续调用 ReadDeviceAD 读取 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，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++ & C++Builder:***

```
BOOL ReadDeviceAD (HANDLE hDevice,
                   PWORD pADBuffer,
                   ULONG nReadSizeWords）
```

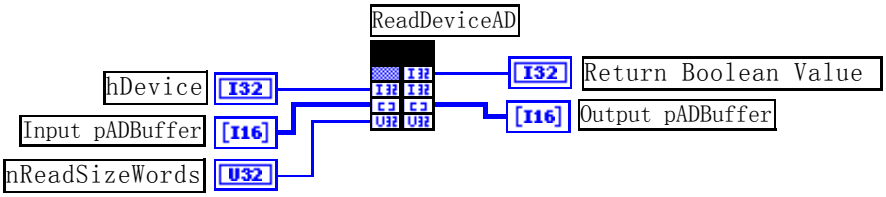
***Visual Basic:***

```
Declare Function ReadDeviceAD Lib "USB2080" (ByVal hDevice As Long, _
                                             ByRef pADBuffer As Integer, _
                                             ByVal nReadSizeWords As Long, _
                                             ) As Boolean
```

***Delphi:***

```
 Function ReadDeviceAD(hDevice : Integer;
                       pADBuffer : Word;
                       nReadSizeBytes:LongWord) : Boolean;
     StdCall; External 'USB2080' Name 'ReadDeviceAD';
```

***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。注意在每次 InitDeviceAD 之后，用 ReadDeviceAD 函数读取的所有数据，其第一个点无效，必须丢掉，从第二个点开始全部有效。

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，请参考第八章《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 32 字的整数倍长，如 32、64、128 …… 8192 等字长，否则，USB 设备对象将失败该读操作。

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++ & C++Builder:***

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

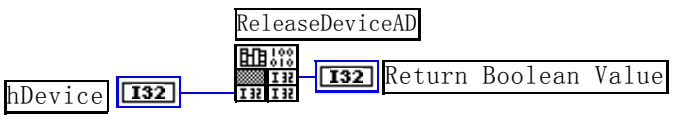
***Visual Basic:***

```
Declare Function ReleaseDeviceAD Lib "USB2080" (ByVal hDevice As Long ) as Boolean
```

***Delphi:***

```
Function ReleaseDeviceAD(hDevice : Longint):Boolean;
        StdCall; External 'USB2080' Name 'ReleaseDeviceAD';
```

***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++ & C++ Builder:***

```
BOOL LoadParaAD(HANDLE hDevice, PUSB2080_PARA_AD pADPara)
```

***Visual Basic:***

```
Declare Function LoadParaAD Lib "USB2080" (ByVal hDevice As Long, _
                                           pADPara As USB2080_PARA_AD) As Boolean
```

***Delphi:***

```
Function LoadParaAD (hDevice : Integer; pADPara:PUSB2080_PARA_AD):Boolean;
                     StdCall; External 'USB2080' 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 PUSB2080_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 PUSB2080_PARA_AD 请参考 USB2080.h 或 USB2080.Bas 或 USB2080.Pas 函数原型定义文件，也可参考本文第四章《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

♦写设备硬件参数函数到 Windows 系统中

函数原型：

***Viusal C++ & C++ Builder:***

BOOL SaveParaAD (HANDLE hDevice, PUSB2080_PARA_AD pADPara)

***Visual Basic:***

```
Declare Function SaveParaAD Lib "USB2080" (ByVal hDevice As Long, _
                                           pADPara As USB2080_PARA_AD) As Boolean
```

***Delphi:***

```
Function SaveParaAD (hDevice : Integer; pADPara:PUSB2080_PARA_AD):Boolean;
                    StdCall; External 'USB2080' 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USB2080_PARA_AD的详细介绍请参考USB2080.h或USB2080.Bas或USB2080.Pas函数原型定义文件，也可参考本文第四章《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

## 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++ & C++Builder:***

BOOL SetDeviceDO (HANDLE hDevice, PUSB2080_PARA_DO pDOPara)

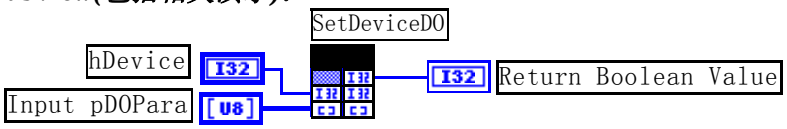
***Visual Basic:***

```
Declare Function SetDeviceDO Lib "USB2080" (ByVal hDevice As Long, _
                                            ByVal pDOPara As USB2080_PARA_DO) As Boolean
```

***Delphi:***

```
Function SetDeviceDO (hDevice : Integer; pDOPara:PUSB2080_PARA_DO):Boolean;
    StdCall; External 'USB2080' Name ' SetDeviceDO ';
```

***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。

pDOPara八路开关量输出状态的参数结构，共有八个成员变量，分别对应于DO0-DO7 路开关量输出状态位。比如置pPara->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”。具体定义请参考第六章，第一节 用于数字输出参数

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++ & C++Builder:***

BOOL GetDeviceDI (HANDLE hDevice, PUSB2080_PARA_DI pDIPara)

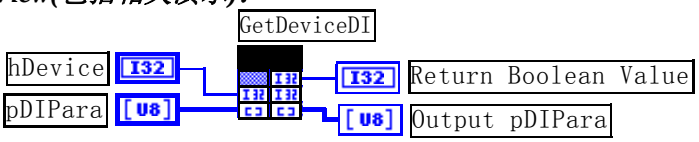
***Visual Basic:***

```
Declare Function GetDeviceDI Lib "USB2080" (ByVal hDevice As Long, _
                                            ByVal pDIPara As USB2080_PARA_DI) As Boolean
```

***Delphi:***

```
Function GetDeviceDI (hDevice : Integer; pDIPara:PUSB2080_PARA_DI):Boolean;
        StdCall; External 'USB2080' Name ' GetDeviceDI ';
```

***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。

pDIPara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 pDIPara->DI0 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。具体定义请参考第六章，第二节 用于数字输入参数。

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）。

# 第四章 硬件参数结构

## 第一节 AD 硬件参数介绍（主要用于 AD 数据采集部分）

***Visual C++ & C++Builder:***

```
typedef struct _USB2080_PARA_AD        // 板卡各参数值
{
    LONG ADMode;              // AD 采集方式(分组或连续)
    LONG FirstChannel;         // 首通道
    LONG LastChannel;         // 末通道
    LONG Frequency;           // AD 采集频率(Hz)
    LONG GroupInterval;       // 分组采样时，相邻组的时间间隔(uS)
    LONG TriggerSource;       // 内/外触发方式选择
    LONG TriSignalDir;        // 正向/负向触发选择(电平)
}USB2080_PARA_AD,*PUSB2080_PARA_AD;
```

***Visual Basic：***

```
Private Type USB2080_PARA_AD
```

```
    ADMode As Long          ' AD 采集方式(分组或连续)
    FirstChannel As Long    ' 首通道
    LastChannel As Long     ' 末通道
    Frequency As Long       ' AD 采集频率(Hz)
    GroupInterval As Long   ' 分组采样时，相邻组的时间间隔(uS)
    TriggerSource As Long   ' 内/外触发方式选择
    TriSignalDir As Long    ' 正向/负向触发选择(电平)
End Type
```

***Delphi:***

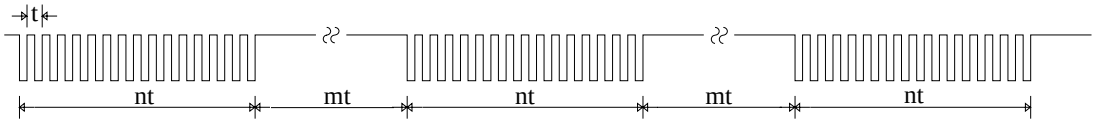
```
    Type   // 定义结构体数据类型
    PUSB2080_PARA_AD = ^USB2080_PARA_AD;   // 指针类型结构
    USB2080_PARA_AD = record     // 标记为记录型
        ADMode : LongInt         // AD 采集方式(分组或连续)
        FirstChannel : LongInt   // 通道
        LastChannel : LongInt    // 末通道
        Frequency : LongInt      // AD 采集频率(Hz)
        GroupInterval : LongInt  // 分组采样时，相邻组的时间间隔(uS)
        TriggerSource : LongInt  // 内/外触发方式选择
        TriSignalDir : LongInt   // 正向/负向触发选择(电平)
    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。

ADMode 采集方式选择，若置为常量 USB2080_GROUP_MODE(或 0x00)，视为分组采集，若置为常量 USB2080_SEQUENCE_MODE(或 0x01)视为连续采集。连续采集方式的情况是：在由 Frequency 指定的采样频率下所采集的全部数据在时间轴上是等简隔的，比如将 Frequency 指定为 100KHz, 即每隔 10 微秒采样一个点，总是这样重复下去。而分组采集方式的情况是：所有采集的数据点在时间轴上被划分成若干个等长的组，而组内通常有大于 2 个点的数据，组内各点频率由 Frequency 决定，组间间隔由 GroupInterval 决定。比如用户要求在对 0-15 通道共 16 个通道用 100KHz 频率每采集一个轮回后，要求间隔 1 毫秒后，再对这 16 个通道采集下一个轮回，那么分组采集便是最佳方式，它可以将组间延时误差控制在 0.5 微秒以下。关于分组与连续采集更详细的说明请参考硬件说明书。如下图：




其中：t为所需触发A/D转换的周期，它由Frequency参数的倒数决定。

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

nt为每组操作所需时间。

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

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

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

注：当首通道和末通道相等时，即为单通道采集。

Frequency 在连续采集方式中，它属于 AD 等间隔采样频率，在分组采集方式中，它成为组内采样频率。单位 Hz，其范围应根据具体的设备而定，本设备的最大频率可为 400KHz，但其最小值不能小于 1Hz。

GroupInterval 在分组采集方式中，它成为组间间隔的时间常数。单位微秒，其范围应根据具体的设备而定，本设备的最大间隔时间为 2048 微秒(板上时钟分辨率 1 微秒 乘以十一位计数器值 2048)，但其最小间隔不能小于 1 微秒。而在连续采集方式中，此参数无效，可以赋任意值。

TriggerSource AD 转换触发源，若等于常量 USB2080_IN_TRIGGER 则为内部定时触发，若等于常量 USB2080_OUT_TRIGGER 则为外触发。两种方式的主要区别是：外触发是当设备被 InitDeviceAD 函数初始化就绪后，并没有立即启动 AD 采集，仅当外接管脚 DTR(在 37 芯 D 型头上)上有一个符合要求的 TTL 电平时，AD 转换器便被启动，且按用户预先设定的采样频率由板上的硬件定时器时定触发 AD 等间隔转换每一个 AD 数据，此后，除非用户重新初始化设备，否则，DTR 管脚上新的电平信号变化并不影响 AD 转换进程。因此如果用户不断的使下一个触发信号有效，那么您必须在每一个外触发信号到来之前重新初始化设备。而对于内触发方式则与 DTR 管脚上的信号无任何关系，它是在用户调用 InitDeviceAD 函数初始化设备时，由这个函数中的最后一条软件指令立即启动 AD 转换器，AD 转换器便以指定的频率由板上定时器等间隔定时触发 AD 转换。指两种触发方式的应用场合：对于瞬间变化(持续时间短、变化频率较高)、或随机性较强的信号的测量和采样。或是需要精确定位所要采集的一批 AD 数据中的第一个点的时间轴，那么您需要使用外触发方式。而对于持续变化时间较长，不需要精确定位信号起点的信号，则一般使用内触发方式。

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| USB2080_IN_TRIGGER | 0x0000 | 内触发方式 |
| USB2080_OUT_TRIGGER | 0x0001 | 外触发方式 |

TriSignalDir 外触发方式电平使用的方向。只有 TriggerSource 设置为外触发方式时，TriSignalDir 才有用。它的选项值如下表：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| USB2080_NEGATIVE_DIR | 0x0000 | 负向（即低电平）触发 |
| USB2080_POSITIVE_DIR | 0x0001 | 正向（即高电平）触发 |

**相关函数:** InitDeviceAD LoadParaAD SaveParaAD

## 第二节 数字开关量输出参数（USB2080_PARA_DO）

***Visual C++ & C++Builder:***

```
typedef struct _USB2080_PARA_DO        // 数字量输出参数
{
    BYTE DO0;          // 0 通道
    BYTE DO1;          // 1 通道
    BYTE DO2;          // 2 通道
    BYTE DO3;          // 3 通道
    BYTE DO4;          // 4 通道
    BYTE DO5;          // 5 通道
    BYTE DO6;          // 6 通道
    BYTE DO7;          // 7 通道
```

} USB2080_PARA_DO,*PUSB2080_PARA_DO;

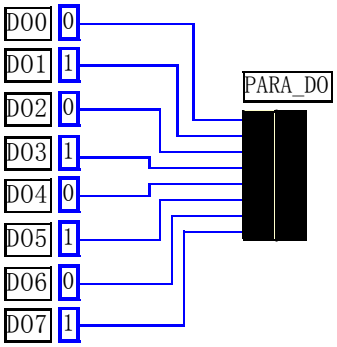
***Visual Basic：***

```
Type USB2080_PARA_DO
    DO0 As Byte   ‘ 0 通道
    DO1 As Byte   ‘ 1 通道
    DO2 As Byte   ‘ 2 通道
    DO3 As Byte   ‘ 3 通道
    DO4 As Byte   ‘ 4 通道
    DO5 As Byte   ‘ 5 通道
    DO6 As Byte   ‘ 6 通道
    DO7 As Byte   ‘ 7 通道
End Type
```

***Delphi:***

```
  Type   // 定义结构体数据类型
  PUSB2080_PARA_DO = ^USB2080_PARA_DO;   // 指针类型结构
  USB2080_PARA_DO = record          // 标记为记录型
    DO0: Byte;     // 0 通道
    DO1: Byte;     // 1 通道
    DO2: Byte;     // 2 通道
    DO3: Byte;     // 3 通道
    DO4: Byte;     // 4 通道
    DO5: Byte;     // 5 通道
    DO6: Byte;     // 6 通道
    DO7: Byte;     // 7 通道
    End;
```

***LabView:***




该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要象 Visual Basic 中的属性操作那样，只需要有简单的进行属性赋值，然后执行 SetDeviceDO 即可完成数字量输出。注意关于 LabView 的参数定义，他最主要表达了在 LabView 环境中怎样使用 SetDeviceDO 实现开关量输出操作的基本实现方法。在用户实际使用中，您可以将左边的常量图标换成开关控件图标等，以实现动态改变开关量输出状态。但需要注意的是开关控件图标（xxx Switch）输出的值是布尔变量，因此在开关控件图标与 PUSB2080_PARA_DO 之间，应使用 Boolean To (0,1)逻辑转换控件，即先将布尔变量转换成 0 或 1 的整型值，再将这个整型值传递给 PUSB2080_PARA_DO，详见开关量输入输出 LabView 演示部分。

其每一个成员变量对应于相应的 DO 通道，即 DO0-DO7 分别对应于 DO 通道 0-7。且这些成员变量只能被赋值为“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

## 第三节 数字开关量输入参数（USB2080_PARA_DI）

***Visual C++ & C++Builder:***

```
typedef struct _USB2080_PARA_DI          // 数字量输入参数
{
    BYTE DI0;          // 0 通道
    BYTE DI1;          // 1 通道
    BYTE DI2;          // 2 通道
    BYTE DI3;          // 3 通道
    BYTE DI4;          // 4 通道
    BYTE DI5;          // 5 通道
    BYTE DI6;          // 6 通道
    BYTE DI7;          // 7 通道
} USB2080_PARA_DI,*PUSB2080_PARA_DI;
```

***Visual Basic：***

```
Type USB2080_PARA_DI
    DI0 As Byte   ' 0 通道
    DI1 As Byte   ' 1 通道
    DI2 As Byte   ' 2 通道
    DI3 As Byte   ' 3 通道
    DI4 As Byte   ' 4 通道
    DI5 As Byte   ' 5 通道
    DI6 As Byte   ' 6 通道
    DI7 As Byte   ' 7 通道
End Type
```

***Delphi:***

```
Type   // 定义结构体数据类型
  PUSB2080_PARA_DI = ^USB2080_PARA_DI;   // 指针类型结构
  USB2080_PARA_DI = record          // 标记为记录型
    DI0: Byte;     // 0 通道
    DI1: Byte;     // 1 通道
    DI2: Byte;     // 2 通道
    DI3: Byte;     // 3 通道
    DI4: Byte;     // 4 通道
    DI5: Byte;     // 5 通道
    DI6: Byte;     // 6 通道
    DI7: Byte;     // 7 通道
End;
```

该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要执行 GetDeviceDI 即可完成数字量输入操作。然后象 Visual Basic 中的属性操作那样，简单的进行属性成员分析即可确定各路状态。

关于 LabView 的参数，由于需要的是返回值，因此根据 LabView 的特点，应分配一个 8 字节的内存单元，每一

个字节的内存单元对应相应位置上的开关量输入状态。要使用这些状态，则应在 GetDeviceDI 之后，将存放实际的当前开关量状态的内存单元用 Index Array 数组操作控件将其每一路开关量状态分离出来，即可确定每一路开关输入状态。详见开关量输入输出 LabView 演示部分。

其每一个成员变量对应于相应的 DI 通道，即 DI0-DI7 分别对应于 DI 通道 0-7。且这些成员变量只能是“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

# 第五章 数据格式转换与排列规则

## 第一节 如何将 AD 原始数据 LSB 转换电压值 Volt

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 8 位的模板CountLSB为 256。而本设备的AD为 16 位，则为 65536。其他类型同理均按 $2^n$=LSB总数（n为Bit位数）换算即可。

| 量程(毫伏) | 计算机语言换算公式 | Volt 取值范围 mV |
|---|---|---|
| ±10000mV | Volt = Lsb * (20000 / 65536 ) – 10000.0 | [-10000， +10000] |
| 0-10000mV | Volt = Lsb * (10000 / 65536 ) | [0， +10000] |
| ±5000mV | Volt = Lsb * (10000 / 65536 ) – 5000.0 | [-5000， +5000] |

换算举例：（设量程为±5000mV，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（必须为 16 位无符号数）
float Volt;    // 定义存放转换后的电压值的变量
Lsb = pADBuffer [0];   // 取得标准 LSB 原码
Volt = Lsb * (10000.0/65536) – 5000.0;   // 用 LSB 原码与单位电压值相乘求得实际电压值
```

***Visual Basic:***

由于 Visula Basic 中不具备 16 位无符号数，因此需要构建自定义 16 位无符号数据的等效结构：

```
Type UInteger
    Byte DataL
    Byte DataH
End
```

然后再将 ReadDeviceAD 的缓冲区类型重新定义为 UInteger 类型。待数据返回后再转换

```
Dim Lsb As Long   ' 定义存放标准 LSB 原码的变量（用有符号 32 位存储无符号 16 位数据）
Dim Volt As Single   ' 定义存放转换后的电压值的变量
' 将自定义结构体数据转换成 32 位数据（高 16 位无效）
Lsb = pADBuffer (0).DataH * 256 + pADBuffer (0).DataL
Volt = Lsb * (10000.0/65536) – 5000.0   ' 用 LSB 原码与单位电压值相乘求得实际电压值
```

***Delphi:***

```
Lsb : Word;   // 定义存放标准 LSB 原码的变量（必须为 16 位无符号数）
Volt ： Single;   // 定义存放转换后的电压值的变量
Lsb ： = pADBuffer [0];   // 取得标准 LSB 原码
Volt ： = Lsb * (10000.0/65536) – 5000.0; // 用 LSB 原码与单位电压值相乘求得实际电压值
```

## 第二节 关于采集函数的 pADBuffer 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 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

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应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民，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 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 USB2080.h 和 Sys.cpp)。

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

**其简易程序默认存放路径为：系统盘\ART\USB2080\SAMPLES\VC\SIMPLE\AD**

### 二、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 USB2080.h 和 Sys.cpp)。

**[程序]**丨**[阿尔泰测控演示系统]**丿**[Microsoft Visual C++]**丿**[简易代码演示]**丿**[开关量演示源程序]**

**其默认存放路径为：系统盘\ART\USB2080\SAMPLES\VC\SIMPLE\DIO**

**其他语言的演示可以用上面类似的方法找到。**

## 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 USB2080.h 和 DADoc.cpp)。

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

**其默认存放路径为：系统盘\ART\USB2080\SAMPLES\VC\ADVANCED**

**其他语言的演示可以用上面类似的方法找到。**

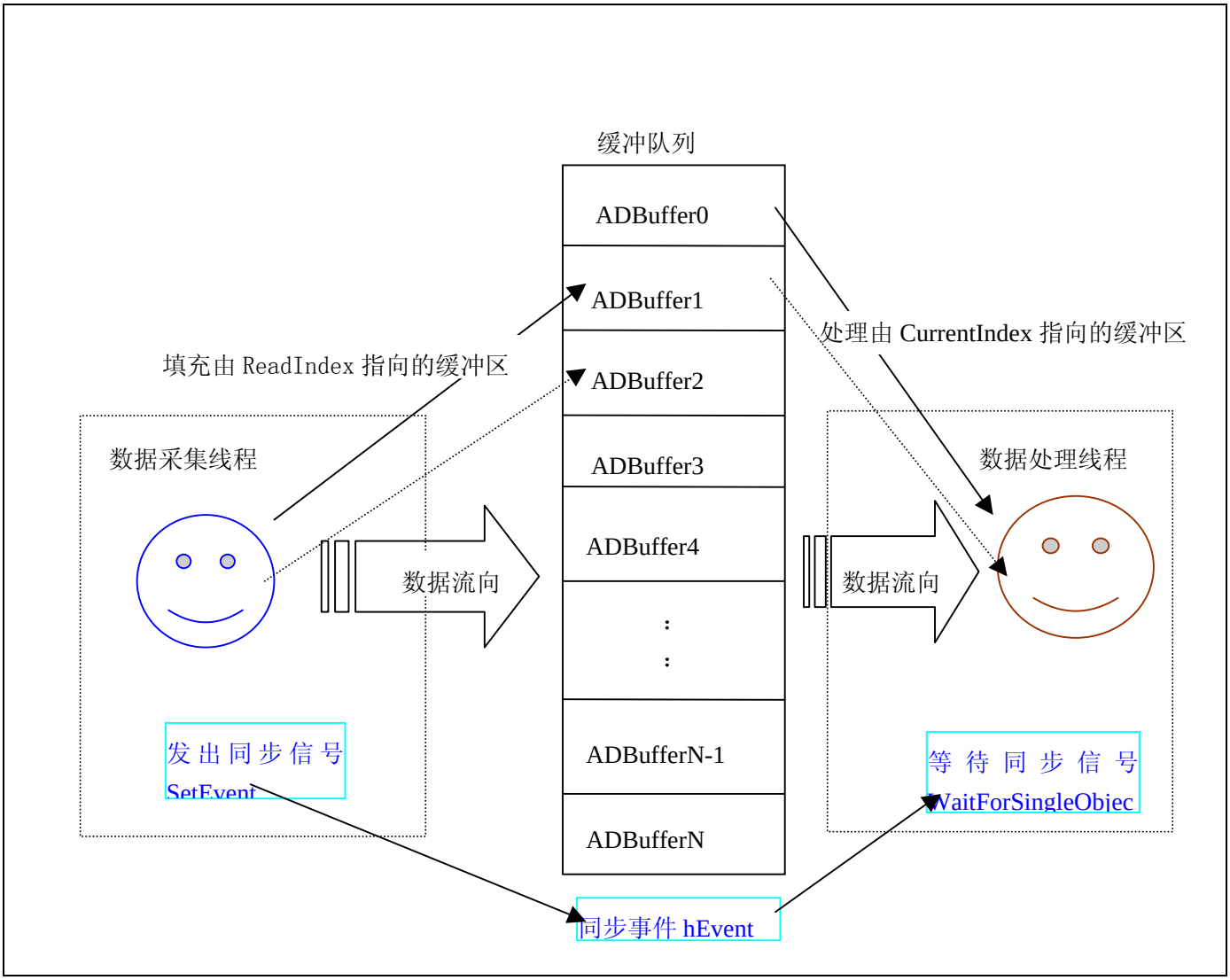
# 第七章 基于 USB 总线的大容量连续数据采集详述

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两维数组如 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面被堵塞，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]**丨**[阿尔泰测控演示系统]**丨**[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]**丨**[阿尔泰测控演示系统]**丨**[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
```

# 第八章 公共接口函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序有力的辅助手段，使您编写应用程序更容易，使您的应用程序更高效。

## 第一节 公用接口函数列表

| 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 中的子线程 | |
| CreateSystemEvent | 创建系统内核事件对象 | 用于线程同步或中断 |
| ***② 文件对象操作函数*** | | |
| CreateFileObject | 初始设备文件对象 | |
| WriteFile | 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 | |
| ReleaseFile | 释放已有的文件对象 | |
| **③ 其他函数** | | |
| GetDiskFreeBytes | 取得指定磁盘的可用空间(字节) | 适用于所有设备 |

## 第二节 公用接口函数原型说明

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

### ♦ 在 VB 环境中,创建子线程对象,以实现多线程操作

***Visual Basic***

```
Declare Function CreateVBThread Lib "USB2080" (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 ResumeThread 函数启动它。若失败,则返回 FALSE，用户可用 GetLastError 捕获当前错误码。

**相关函数：** CreateVBThread TerminateVBThread

**注意: RoutineAddr 指向的函数或过程必须放在 VB 的模块文件中,如 USB2080.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 Basic:***

Declare Function TerminateVBThread Lib "USB2080" (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**

***Visual C++:***

HANDLE CreateSystemEvent(void);

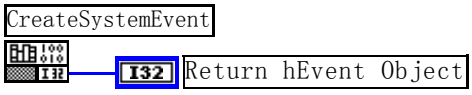
***Visual Basic:***

Declare Function CreateSystemEvent Lib " USB2080 " () As Long

***Delphi:***

Function CreateSystemEvent():Integer; StdCall; External 'USB2080' 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**

***Visual C++:***

BOOL ReleaseSystemEvent(HANDLE hEvent);

***Visual Basic:***

Declare Function ReleaseSystemEvent Lib " USB2080 " (ByVal hEvent As Long) As Boolean

***Delphi:***

Function ReleaseSystemEvent(hEvent : LongInt):Integer; StdCall; External 'USB2080' Name ' ReleaseSystemEvent ';

***LabVIEW:*** （请参见演示程序）

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

### ♦ 初始化设备文件对象

函数原型：

***Visual C++:***

```
Handle CreateFileObject (
                    HANDLE hDevice,
                    LPCTSTR strFileName,
                    int Mode)
```

***Visual Basic:***

```
Declare Function CreateFileObjectLib "USB2080" (ByVal hDevice As Long, _
                                                ByVal strFileName As String, _
                                                ByVal Mode As Long
                                                ) As Long
```

***Delphi:***

```
Function CreateFileObject (hDevice : Integer; strFileName: string; Mode: Integer):LongInt;
                    stdcall; external 'USB2080' name 'CreateFileObject';
```

***LabVIEW:*** *（请参见相关演示程序）*

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

参数：

hDevice 设备对象句柄,它应由 CreateDevice 创建。

strFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：”C:\\USB2080\\Data.Dat”， 在 Basic 中，其语法格式如：”C:\USB2080\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 |
|---|---|---|
| USB2080_modeRead | 0x0000 | 只读文件方式 |
| USB2080_modeWrite | 0x0001 | 只写文件方式 |
| USB2080_modeReadWrite | 0x0002 | 既读又写文件方式 |
| USB2080_modeCreate | 0x1000 | 如果文件不存可以创建该文件，否则重建此文件，并清 0 |
| USB2080_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++:***

```
BooL WriteFile( HANDLE hFileObject,
                PWORD pDataBuffer,
                LONG nWriteSizeBytes)
```

***Visual Basic:***

```
Declare Function WriteFile Lib "USB2080" (ByRef hFileObject As Long,
                                          ByVal pDataBuffer As Integer,_
                                          ByVal nWriteSizeBytes As Long, _
                                          ) As Boolean
```

***Delphi:***

```
function WriteFile(hFileObject: Integer;
                   pDataBuffer:PWordArray;
                   nWriteSizeBytes: LongWord):Boolean;
                   stdcall; external 'USB2080' 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 NewFileName 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。

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 GetLastError 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++:***

```
BOOL ReadFile( HANDLE hFileObject,
               PVOID pDataBuffer,
               LONG OffsetBytes,
               LONG nReadSizeBytes)
```

***Visual Basic:***

```
Declare Function ReadFile Lib "USB2080" (ByVal hFileObject As Long, _
                                         ByRef pDataBuffer As Integer, _
                                         ByVal nOffsetBytes As Long, _
                                          ByVal nReadSizeBytes As Long, _
                                         ) As Boolean
```

***Delphi:***

```
Function ReadFile(hFileObject: Integer;
```

pDataBuffer:PWordArray;
nOffsetBytes:LongWord;
nReadSizeBytes:LongWord;):Boolean;
stdcall; external 'USB2080' name 'ReadFile';

***LabVIEW:（详见相关演示程序）***

功能：将磁盘数据从指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。
nOffsetBytes 指定从文件开始端所偏移的读位置。
nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。
返回值：若成功，则返回 TRUE，否则返回 FALSE，用户可以用 GetLastError 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++:***

BOOL SetFileOffset (HANDLE hFileObject, int nOffsetBytes)

***Visual Basic:***

Declare Function SetFileOffset Lib "USB2080" (ByVal hFileObject As Long,
ByVal nOffsetBytes As Long) As Boolean

***Delphi:***

Function SetFileOffset (hFileObject: Integer):Boolean;
stdcall; external 'USB2080' 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 GetLastError 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++:***

BOOL GetFileLength (HANDLE hFileObject)

***Visual Basic:***

Declare Function GetFileLength Lib "USB2080" (ByVal hFileObject As Long) As Boolean

***Delphi:***

Function GetFileLength (hFileObject: Integer):Boolean;
stdcall; external 'USB2080' Name ' GetFileLength ';

***LabVIEW:***

功能：取得文件长度。

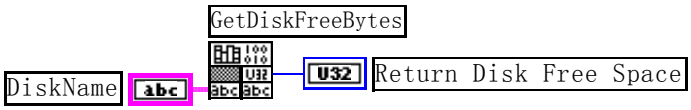
参数：hFileObject 设备对象句柄，它应由 CreateFileObject 创建。
返回值：若成功，则返回 TRUE，否则返回 FALSE，用户可以用 GetLastError 捕获错误码。
**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

♦ **释放设备文件对象**
函数原型：
***Visual C++:***
BOOL ReleaseFile(HANDLE hFileObject)
***Visual Basic:***
Declare Function ReleaseFile Lib "USB2080" (ByVal hDevice As Long) As Boolean
***Delphi:***
Function ReleaseFile(hDevice : Integer):Boolean;
stdcall; external 'USB2080' 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 GetLastError 捕获错误码。
**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

## 第四节 其他函数

♦ **取得指定磁盘的可用空间**
***Visual C++:***
ULONGLONG GetDiskFreeBytes（LPCTSTR DiskName )
***Visual Basic:***
Declare Function GetDiskFreeBytes Lib "USB2080" (ByVal DiskName As String ) As Currency
***LabVIEW:***




功能：取得指定磁盘的可用剩余空间(以字为单位)。
参数：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 GetLastError 捕获错误码。注意使用 64 位整型变量。

♦ **高效高精度延时**
***Visual C++:***
BOOL DelayTimeUs (HANDLE hDevice, LONG nTimeUs)
***Visual Basic:***
Declare Function DelayTimeUs Lib "USB2080" (ByVal hDevice As Long, ByVal nTimeUs As Long) As Boolean
***LabVIEW:***

功能：微秒级延时函数

参数：

hDevice 设备对象句柄，它应由 CreateDevice 决定。

nTimeUs 时间常数。单位 1 微秒

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