USB2089　Windows2000/XP/2003 驱 动 程 序 使 用 说 明 书

请您务必阅读《使用纲要》，他会使您事半功倍!

目　　录

有关设备及驱动安装请参考 USB2089Inst.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。

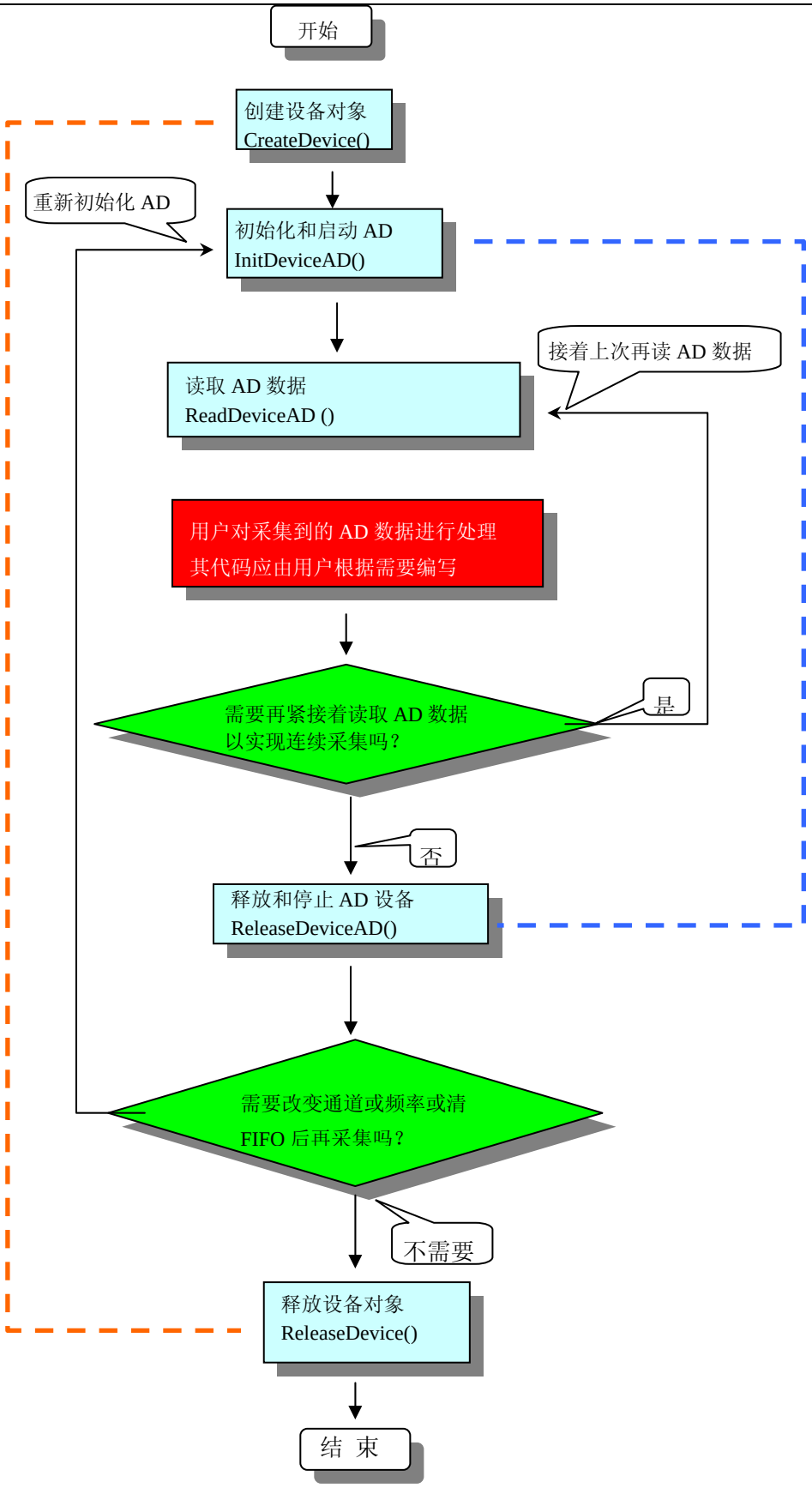


图 2.1.1 AD 采集实现过程

三、哪些函数对您不是必须的？

当公共函数如CreateFileObject, WriteFile,ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。它们只是对我公司驱动程序的一种功能补充，对用户额外提供的。

第三章　USB 设备优越性分析

第一节 USB 接口规范

现在电脑系统连接外围设备的接口并无统一的标准，如键盘用 PS/2 接口，连接打印机要用 25 针的并行接口，鼠标则要用串行或 PS/2 接口。USB 则将这些不同的接口统一起来，使用一个 4 针插头作为标准插头。通过这个标准插头，采用菊花链形式可以把所有的外设连接起来，并且不会损失带宽。

USB 规范中将 USB 分为五个部份：控制器、控制器驱动程序、 USB 芯片驱动程序、USB 设备以及针对不同 USB 设备的客户驱动程序。

根据设备对系统资源需求的不同，在 USB 规范中规定了四种不同的数据传输方式：

等时传输方式(Isochronous)、中断传输方式(Interrupt)、控制传输方式(Control)和批(Bulk)传输方式，这些传输方式各有特点，分别用于不同的场所。

USB 需要主机硬件、操作系统和外设三个方面的支持才能工作。目前主板一般都采用支持 USB 功能的控制芯片组，而且也安装了 USB 接口插座。Windows98 操作系统内置了对 USB 功能的支持（但 WindowsNT 尚不支持 USB)。目前已经有数字照相机、数字音箱、数字游戏杆、打印机、扫描仪、键盘、鼠标等很多 USB 外设问世。 **比如我公司研制生产的基于 USB 接口的各种数据采集卡及工业控制卡也早已投放市场，反映良好。现在在医疗、振动、监控、虚拟仪器、科研实验室、工业生产现场等领域已得到了较为广泛的运用。**

随着大量的支持 USB 的个人电脑的普及以及 Windows98 的广泛应用，USB 逐步成为 PC 机的一个标准接口已经是大势所趋。最新推出的 PC 机几乎 100％支持 USB，另一方面使用 USB 接口的设备也在以惊人的速度发展。

USB 是英文 Universal Serial Bus 的缩写，中文含义是“通用串行总线”。它不是一种新的总线标准，而是应用在 PC 领域的新型接口技术。早在 1995 年，就已经有 PC 带有 USB 接口了，但由于缺乏软件及硬件设备的支持，这些 PC 机的 USB 口都是闲置未用的。1997 年，微软在 WIN95OSR2（WIN97）中开始以外挂模块的形式提供对 USB 的支持， 1998 年后随着微软在 Windows98 中内置了对 USB 接口的支持模块，加上 USB 设备的日渐增多， USB 逐步走进了实用阶段。

第二节 USB 的历史及发展

在谈论 USB 技术之前，不妨让我们来看看外设接口技术的发展历程。多年来个人计算机的串口与并口的功能和结构并没有什么变化。串口的出现是在 1980 年前后，数据传输率是 115kbps～230kbps，串口一般用来连接鼠标和外置 Modem；并口的数据传输率比串口快 8 倍，标准并口的数据传输率为 1Mbps，一般用来连接打印机、扫描仪等。原则上每一个外设必须插在一个接口上，如果所有的接口均被用上了就只能通过添加插卡来追加接口了。串并口不仅速度有限，而且在使用上很不方便。

1994 年，Intel、Compaq、 Digital、IBM、Microsoft、NEC、Northern Telecom 等七家世界著名的计算机和通讯公司成立了 USB 论坛，花了近两年的时间形成了统一的意见，于 1995 年 11 月正式制定了 USB0．9 通用串行总线（Universal Serial Bus）规范，1997 年开始有真正符合 USB 技术标准的外设出现。USB1．1 是目前推出的在支持 USB 的计算机与外设上普遍采用的标准。1999 年初在 Intel 的开发者论坛大会上，与会者介绍了 USB2．0 规范，该规范的支持者除了原有的 Compaq、Intel、Microsoft 和 NEC 四个成员外，还有惠普、朗讯和飞利浦三个新成员。USB2．0 向下兼容 USB1．1，数据的传输率将达到 120Mbps～ 240Mbps，还支持宽带宽数字摄像设备及下一代扫描仪、打印机及存储设备。

目前普遍采用的 USB1．1 主要应用在中低速外部设备上，它提供的传输速度有低速 1．5Mbps 和全速 12Mbps 两种，低速的 USB 带宽（1．5Mbps）支持低速设备，例如显示器、调制解调器、键盘、鼠标、扫描仪、打印机、光驱、磁带机、软驱等。全速的 USB 带宽（12Mbps）将支持大范围的多媒体设备。

现在，支持 USB 的 PC 及外设越来越多，在软件上 USB 也已成为 Windows98 的一个关键部件，并很快在 WindowsCE 和 Windows2000 中得到支持。Apple 的操作平台早已提供对 USB 的支持，预计今后 Sun 和 Digital 的平台也将会提供对这一技术的支持。

第三节 USB 的特点及应用

USB 之所以能得到广泛支持和快速普及，是因为它具备下列的很多特点:

一、特点

1.使用方便

使用 USB 接口可以连接多个不同的设备，支持热插拔（即在主机带电情况下，可以动态的插入和拔出设备），在软件方面，为 USB 设计的驱动程序和应用软件可以自动启动，无需用户干预。 USB 设备也不涉及 IRQ、DMA、地址冲突等问题，它单独使用自己的保留中断，不会同其它设备争用 PC 机有限的资源，为用户省去了硬件配置的烦恼。USB 设备能真正做到“即插即用”。

2.速度加快

快速性能是 USB 技术的突出特点之一。USB 接口的最高传输率目前可达 12Mb / s，比串口快了整整 100 倍，比并口也快了十多倍。今后 USB 的速度还将会提高到 100Mb / s 以上。

3.连接灵活

USB 接口支持多个不同设备的串列连接，一个 USB 口理论上可以连接 127 个 USB 设备。连接的方式也十分灵活，既可以使用串行连接，也可以使用中枢转接头 (Hub)，把多个设备连接在一起，再同 PC 机的 USB 口相接。在 USB 方式下，所有的外设都在机箱外连接，不必打开机箱；允许外设热插拔，而不必关闭主机电源。USB 采用“级联”方式，即每个 USB 设备用一个 USB 插头连接到一个外设的 USB 插座上，而其本身又提供一个 USB 插座供下一个 USB 外设连接用。通过这种类似菊花链式的连接，一个 USB 控制器可以连接多达 127 个外设，而每个外设间距离(线缆长度)可达 5 米。USB 还能智能识别 USB 链上外围设备的接入或拆卸。

4.独立供电

普通使用串口、并口的设备都需要单独的供电系统，而 USB 设备则不需要，因为 USB 接口提供了内置电源。USB 电源能向低压设备提供 5V、500mA 的电源，因此新的设备就不需要专门的交流电源了,从而降低了这些设备的成本并提高了性价比。

5.支持多媒体

USB 提供了对电话的两路数据支持，USB 可支持异步以及等时数据传输,使电话可与 PC 集成,共享语音邮件及其它特性。USB 还具有高保真音频。由于 USB 音频信息生成于计算机外，因而减少了电子噪音干扰声音质量的机会,从而使音频系统具有更高的保真度。

二、USB 的应用

到目前为止，USB 已经在 PC 机的多种外设上得到应用，包括扫描仪、数码相机、数码摄像机、音频系统、显示器、输入设备等等。

扫描仪和数码相机、数码摄像机是从 USB 中最早获益的产品。传统的扫描仪，在执行扫描操作之前，用户必须先启动图像处理软件和扫描驱动软件，然后通过软件操作扫描仪。而 USB 扫描仪则不同，用户只需放好要扫描的图文，按一下扫描仪的按钮，屏幕上会自动弹出扫描仪驱动软件和图像处理软件，并实时监视扫描的过程。USB 数码相机、摄像机更得益于 USB 的高速数据传输能力，使大容量的图像文件传输在短时间内即可完成。

USB 在音频系统应用的代表产品是微软公司推出的 Microsoft DigitalSound System80(微软数字声音系统 80)。使用这个系统，可以把数字音频信号传送到音箱，不再需要声卡进行数模转换，音质也较以前有一定的提高。USB 技术在输入设备上的应用很成功，USB 键盘、鼠标器以及游戏杆都表现得极为稳定，很少出现问题。

早在 1997 年，市场上就已经出现了具备 USB 接口的显示器，为 PC 机提供附加的 USB 口。这主要是因为大多数的 PC 机外设都是桌面设备，同显示器连接要比同主机连接更方便、简单。目前市场上出现的 USB 设备还有 USB Modem、Iomega 的 USB ZIP 驱动器以及 eTek 的 USB PC 网卡等等。

对于笔记本电脑来说，使用 USB 接口的意义更加重大，通用的 USB 接口不仅使笔记本电脑对外的连接变得方便，更可以使笔记本电脑生产厂商不再需要为不同配件在主板上安置不同的接口，这使主板的线路、组件的数量以及复杂程度都有不同程度的削减，从而使系统运行中的散热问题得到了改善。也将促进更高主频的处理器可以迅速应用在移动计算机中，使笔记本电脑与桌面 PC 的差距进一步缩小。

USB 的应用会越来越广泛，一些业界人士甚至预测，未来的 PC 将是一个密封设备，所有外设都将通过 USB 或其他外部接口连接。

我公司也于 2000 年初在国内首家成功地推出了基于**医疗、振动、监控、虚拟仪器、科研实验室、工业生产现场领域的 USB 设备，及时的填补了国内基于 USB 接口数据采集领域的空白。特别是为便携式笔记本电脑和日益流行**

的掌上电脑数据采集提供了极为广阔的发展空间。

而USB数据采集器的软件操作显示更为容易，通常您只须调用我公司提供的驱动程序接口InitDeviceAD初始化设备，然后再用ReadDeviceAD反复读取AD数据即可。

第四章　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“USB2089_”）

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	
GetDevicestatusAD		
ReadDeviceAD	连续批量读取 USB 设备上的 AD 数据	
ReleaseDeviceAD	释放 USB 设备对象中的 AD 部件	
③ 辅助函数（硬件参数设置、保存、读取函数）		
LoadParaAD	从 Windows 系统中读取硬件参数	
SaveParaAD	往 Windows 系统保存硬件参数	
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	
④ 开关量函数		
GetDeviceDI	开关量输入函数	
SetDeviceDO	开关量输出函数	
⑤ DA 数据输出函数		
WritedeviceDA		

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\USB2089\INCLUDE\USB2089.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 USB2089.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。

其次，您还应该在 Visual C++编译环境软件包的 Project Setting 对话框的 Link 属性页中的 Object/Library Module 输入行中加入如下指令：

C:\Art\USB2089\USB2089.LIB

或者：单击 Visual C++编译环境软件包的 Project 菜单中的 Add To Project 的菜单项，在此项中再单击 Files…，在随后弹出的对话框中选择 USB2089.Lib，再单击“确定”，即可完成。

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 USB2089.LIB 的路径，当然也可以把此文件拷到您的源程序目录中。

另外，在 Visual C++演示工程的目录下，也有相应的 USB2089.h 和 USB2089.Lib 文件。

为了驱动程序和相关接口尽量精炼快速，所以没有加任何调试代码，因此用户在使用 VC 接口的时候应使

用发行版本进行源代码编译（Win32 Release），而不应该使用调试版本（Win32 Debug）。具体方法是在源代码编译前，执行 Build 总菜单中的 Set Active Configuration 子菜单命令，便可实现其发行版的设置，然后再编译，即可生成发行版的应用程序。

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(USB2089.H)包含进您的源程序头部。形如：

#include “\Art\USB2089\Include\USB2089.h”

然后再将 USB2089.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 USB2089.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 USB2089.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。

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 USB2089.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：“USB2089”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  USB2089;  // 注意： 在此加入驱动程序接口单元 USB2089
```

LabView / CVI:

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的驱动程序接口请见本文最后一部分关于 LabView 的专述。

请注意，因考虑 Visual C++和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所举的 Visual Basic 程序均是需要编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不保证能完全顺利运行。

第二节 设备对象管理函数原型说明

♦ 创建设备对象函数

函数原型：

Visual C++ & C++ Builder：

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

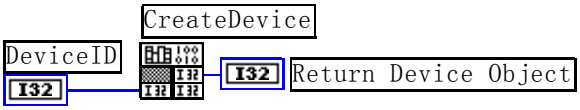
Visual Basic：

```
Declare Function CreateDevice Lib "USB2089" (Optional ByVal DeviceID As Long = 0) As Long
```

Delphi：

```
Function CreateDevice(DeviceID:Integer = 0):Integer; StdCall; External 'USB2089' Name 'CreateDevice';
```

LabView:



功能：该函数负责创建设备对象，并返回其设备对象句柄。

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 USB 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。比如若用户往 Windows 系统中加入第一个 USB2089 AD 模板时，系统则以“USB2089”作为基本名称，再以 DeviceID 的初值组合成该设备的标识符“USB2089-0”来确认和管理这第一个设备，若用户接着再添加第二个 USB2089 AD 模板时，则系统将以“USB2089-1”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，DeviceID 应置 0，第二应置 1，也以此类推。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

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
      :
```

♦ **取得在系统中的设备总台数**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "USB2089" (ByVal hDevice As Long) As Long

Delphi:

Function GetDeviceCount (hDevice : LongInt):LongInt; StdCall; External 'USB2089' Name ' GetDeviceCount ';

LabView:

请参考相应演示程序。

功能：取得在系统中物理设备的总台数。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回实际设备台数，否则返回 0，用户可以用 GetLastError 捕获错误码。

相关函数：CreateDevice　　ReleaseDevice

♦ **取得当前设备对象句柄指向的设备所在的设备 ID**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCurrentID (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCurrentID Lib "USB2089" (ByVal hDevice As Long) As Long

Delphi:

Function GetDeviceCurrentID (hDevice : Longint):LongInt;
StdCall; External 'USB2089' Name ' GetDeviceCurrentID ';

LabView:

请参考相应演示程序。

功能：取得指定设备对象所代表的设备在设备链中的当前设备 ID 号（即索引位置）。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回由hDevice参数代表的设备在设备链中的设备ID， 否则返回-1，用户可以用GetLastError捕获错误码。注意其返回的ID是一定与在CreateDevice函数中指定的DeviceID参数值相等。

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 "USB2089" (ByVal hDevice As Long) As Boolean

Delphi:

Function ResetDevice (hDevice : LongInt):Boolean; StdCall; External 'USB2089' Name 'ResetDevice';

LabView:

请参考相应演示程序。

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)

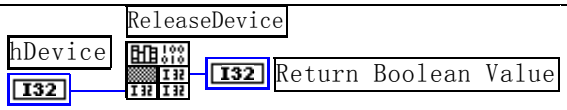
Visual Basic:

Declare Function ReleaseDevice Lib "USB2089" (ByVal hDevice As Long) As Boolean

Delphi:

Function ReleaseDevice(hDevice : Longint):Boolean; StdCall; External 'USB2089' Name 'ReleaseDevice';

LabView:



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice 设备对象句柄，它应由CreateDevice创建。

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

相关函数：CreateDevice

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

第三节 AD 采样操作函数原型说明

- **初始化设备对象**

函数原型:

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

```
BOOL InitDeviceAD( HANDLE hDevice,
                   PUSB2089_PARA_AD pADPara )
```

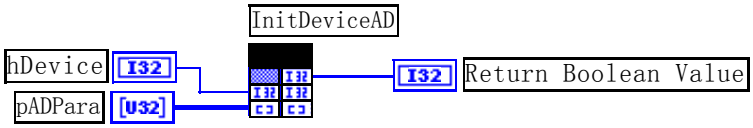
Visual Basic:

```
Declare Function InitDeviceAD Lib "USB2089" ( ByVal hDevice As Long, _
                                              ByRef pADPara As USB2089_PARA _AD ) As Boolean
```

Delphi:

```
Function InitDeviceAD( hDevice : Integer; pADPara:PUSB2089_PARA_AD):Boolean;
                       StdCall; External 'USB2089' 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）
```

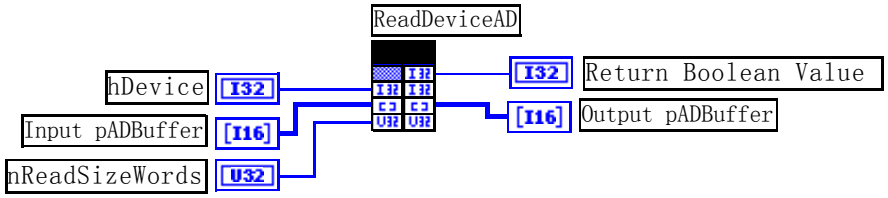
Visual Basic:

```
Declare Function ReadDeviceAD Lib "USB2089" (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 'USB2089' 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。

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，请参考第六章《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，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。注意此参数不能大于AD硬件参数中的ADPara. nReadSizeWords成员值。关于ADPara. nReadSizeWords的定义请参考《AD硬件参数介绍》章节。

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)
```

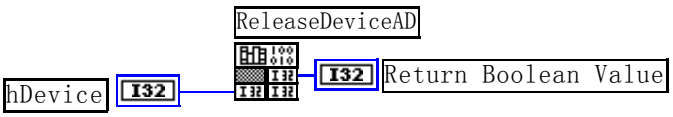
Visual Basic:

```
Declare Function ReleaseDeviceAD Lib "USB2089" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function ReleaseDeviceAD(hDevice : Longint):Boolean;
                         StdCall; External 'USB2089' 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, PUSB2089_PARA_AD pADPara)
```

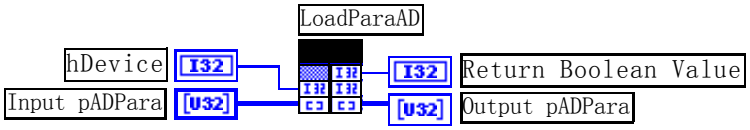
Visual Basic:

```
Declare Function LoadParaAD Lib "USB2089" ( ByVal hDevice As Long, _
                                            ByRef pADPara As USB2089_PARA_AD) As Boolean
```

Delphi:

```
Function LoadParaAD(hDevice : Integer; pADPara:PUSB2089_PARA_AD):Boolean;
                    StdCall; External 'USB2089' Name 'LoadParaAD';
```

LabView:



功能：负责从 Windows 系统中读取设备硬件参数。

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara 属于 PUSB2089_PARA 的结构指针型，它负责返回 USB 硬件参数值，关于结构指针类型 PUSB2089_PARA 请参考相应 USB2089.h 或该结构的帮助文档的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　　SaveParaAD　　ReleaseDevice

Visual C++ & C++Builder 举例：

```
        :
USB2089_PARA_AD ADPara;
```

```
HANDLE    hDevice;
HDevice = CreateDevice(0) ;      // 管理秕一个 USB 设备
if(!LoadParaAD(hDevice, &ADPara))
{
        AfxMessageBox("读入硬件参数失败，请确认您的驱动程序是否正确安装");
        Return;    // 若错误，则退出该过程
}
          :
```

Visual Basic 举例:

```
            :
Dim ADPara As USB2089_PARA_AD
Dim hDevice As Long   :
hDevice = CreateDevice(0)    ' 管理第一个 USB 设备
If Not LoadParaAD(hDevice, ADPara) Then
        MsgBox "读入硬件参数失败，请确认您的驱动程序是否正确安装"
        Exit Sub      ' 若错误，则退出该过程
End If
            :
```

♦ **往 Windows 系统写入设备硬件参数函数**

函数原型：

Viusal C++ & C++Builder:

BOOL SaveParaAD(HANDLE hDevice, PUSB2089_PARA_AD pADPara)

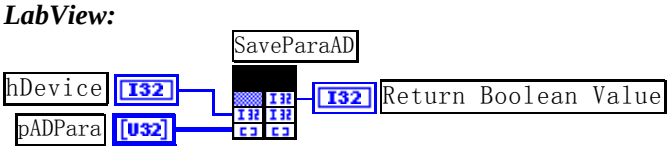
Visual Basic:

Declare Function SaveParaAD Lib "USB2089" (ByVal hDevice As Long, _
ByRef pADPara As USB2089_PARA_AD) As Boolean

Delphi:

Function SaveParaAD (hDevice : Integer; pADPara:PUSB2089_PARA_AD):Boolean;
StdCall; External 'USB2089' Name 'SaveParaAD';

LabView:



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

参数：

hDevice 设备对象句柄,它应由CreateDevice创建。

pADPara AD设备硬件参数，请参考《硬件参数结构》章节。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice LoadParaAD ReleaseDevice

Visual C++ & C++Builder 程序举例:

```
            :
USB2089_PARA_AD ADPara;     HANDLE hDevice ;
hDevice = CreateDevice(0);
// 可以手工设置硬件参数，然后保存
ADPara.FirstChannel=0;   // 首通道
ADPara.LastChannel=15;   // 末通道
        :   // 您应同时设置其他参数，否则未设置的参数将以 0 值保存
if(!SaveParaAD(hDevice, &ADPara))
{
        AfxMessageBox("保存参数有误");
}
          :
```

Visual Basic 程序举例:

```
            :
Dim ADPara As USB2089_PARA_AD
Dim hDevice As Long   :   hDeivce = CreateDevice()
ADPara.FirstChannel = 0     :  ADPara.LastChannel = 15
        :   ' 您应同时设置其他参数，否则未设置的参数将以 0 值保存
```

```
If Not SaveParaAD(hDevice, ADPara)   Then
        MsgBox “保存参数有误”
        Exit Sub      ' 若错误，则退出该过程
End IF
        :
```

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, PUSB2089_PARA_DO pDOPara)

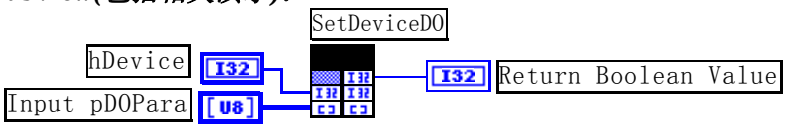
Visual Basic:

Declare Function SetDeviceDO Lib "USB2089" (ByVal hDevice As Long, _
ByRef pDOPara As USB2089_PARA_DO) As Boolean

Delphi:

Function SetDeviceDO (hDevice : Integer; pDOPara:PUSB2089_PARA_DO):Boolean;
StdCall; External 'USB2089' 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《数字开关量输出参数》。

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, PUSB2089_PARA_DI pDIPara)

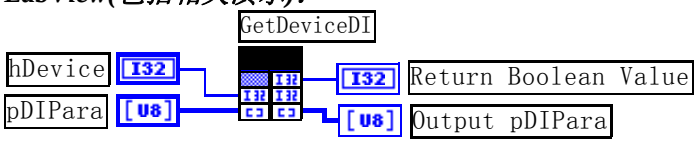
Visual Basic:

Declare Function GetDeviceDI Lib "USB2089" (ByVal hDevice As Long, _
ByRef pDIPara As USB2089_PARA_DI) As Boolean

Delphi:

Function GetDeviceDI (hDevice : Integer; pDIPara:PUSB2089_PARA_DI):Boolean;
StdCall; External 'USB2089' 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《数字开关量输入参数》。

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）。

第五章 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 _USB2089_PARA_AD        // 板卡各参数值
{
    LONG ADMode;              // AD 采集方式(分组或分频)
    LONG ReadSizeWords;       // 读取长度
    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 Gains;               // 采集增益
    LONG TriggerMode;         // 内外触发方式选择
    LONG TriggerType;         // 电平触发/边沿触发
    LONG TriggerDir;          // 正向/负向触发选择
    LONG ClockSource;         // 时钟源选择
    LONG bClockOutput;        // 允许时钟输出
} USB2089_PARA_AD, *PUSB2089_PARA_AD;
```

Visual Basic：

```
Private Type USB2089_PARA_AD
    ADMode As Long          ' AD 采集方式(分组或分频)
    ReadSizeWords As Long       ' 读取长度
    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)
    Gains As Long           ' 采集增益
    TriggerMode As Long         ' 内外触发方式选择
    TriggerType As Long         ' 电平触发/边沿触发
    TriggerDir As Long      ' 正向/负向触发选择
```

```
    ClockSource As Long          ' 时钟源选择
    bClockOutput As Long         ' 允许时钟输出
End Type
```

Delphi:

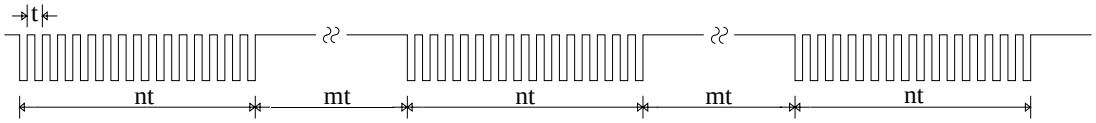
```
  Type   // 定义结构体数据类型
    PUSB2089_PARA_AD = ^USB2089_PARA_AD;   // 指针类型结构
    USB2089_PARA_AD = record     // 标记为记录型
    ADMode : LongInt;// AD 采集方式(分组或分频)
    ReadSizeWords : LongInt;        // 读取长度
    FirstChannel : LongInt;         // 首通道
    LastChannel : LongInt;          // 末通道
    Frequency : LongInt;            // AD 采集频率(Hz)
    GroupInterval : LongInt;// 分组采样时，相邻组的时间间隔(uS)
    Gains : LongInt;                // 采集增益
    TriggerMode : LongInt;          // 内外触发方式选择
    TriggerType : LongInt;          // 电平触发/边沿触发
    TriggerDir : LongInt;           // 正向/负向触发选择
    ClockSource : LongInt;          // 时钟源选择
    bClockOutput : LongInt;         // 允许时钟输出
    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USB2089_GROUP_MODE(或 0x00)，视为分组采集，若置为常量USB2089_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。

Gains AD 采样程控增益。

常量名	常量值	功能定义
USB2089_GAINS_1MULT	0x00	1 倍增益(使用 PGA202 或 PGA203 放大器)
USB2089_GAINS_10MULT	0x01	10 倍增益(使用 PGA202 放大器)
USB2089_GAINS_100MULT	0x02	100 倍增益(使用 PGA202 放大器)
USB2089_GAINS_1000MULT	0x03	1000 倍增益(使用 PGA202 放大器)
USB2089_GAINS_2MULT	0x01	2 倍增益(使用 PGA203 放大器)
USB2089_GAINS_4MULT	0x02	4 倍增益(使用 PGA203 放大器)
USB2089_GAINS_8MULT	0x03	8 倍增益(使用 PGA203 放大器)

TriggerMode AD触发模式，若等于常量USB2088_TRIGMODE_SOFT则为内部软件触发，若等于常量USB2088_TRIGMODE_POST则为外部硬件后触发。两种方式的主要区别是：外触发是当设备被InitDeviceAD函数初始化就绪后，并没有立即启动AD采集，仅当外接管脚ATR上有一个符合要求的信号时，AD转换器便被启动，且按用户预先设定的采样频率由板上的硬件定时器时定触发AD等间隔转换每一个AD数据，其触发条件由触发类型和触发方向及触发电平决定。

常量名	常量值	功能定义
USB2088_TRIGMODE_SOFT	0x0000	软件内触发方式
USB2088_TRIGMODE_POST	0x0001	硬件后触发方式

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

常量名	常量值	功能定义
USB2089_TRIGDIR_NEGATIVE	0x00	负向（即低电平）触发
USB2089_TRIGDIR_POSITIVE	0x01	正向（即高电平）触发
USB2089_TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低电平或上升/下降沿触发)

相关函数: InitDeviceAD LoadParaAD SaveParaAD

第二节 数字开关量输出参数（USB2089_PARA_DO）

Visual C++ & C++Builder:

```
typedef struct _USB2089_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
```

```
} USB2089_PARA_DO,*PUSB2089_PARA_DO;
```

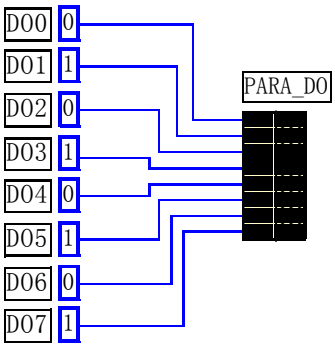
Visual Basic：

```
Type USB2089_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
  PUSB2089_PARA_DO = ^USB2089_PARA_DO;   // 指针类型结构
  USB2089_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PUSB2089_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PUSB2089_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（USB2089_PARA_DI）

Visual C++ & C++Builder:

```
typedef struct _USB2089_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
} USB2089_PARA_DI,*PUSB2089_PARA_DI;
```

Visual Basic：

```
Type USB2089_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
  PUSB2089_PARA_DI = ^USB2089_PARA_DI;   // 指针类型结构
  USB2089_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。

第六章　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 14 位，则为 16384。其他类型同理均按 2^n=LSB总数（n为Bit位数）换算即可。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000mV	Volt = (20000.00 / 16384) * ADBuffer[0] – 10000.00	[-10000，　+9998.77]
±5000mV	Volt = (10000.00 / 16384) * ADBuffer[0]– 5000.00	[-5000，　+4999.38]

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**（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 = (ADBuffer [0]^0x2000)&0x3FFF;　// 取得标准 LSB 原码

Volt = Lsb * (20000.00/16384) – 10000.00;　// 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 14 位无符号数，因此需要用 16383 与原值相与即刻变换为无效 14 位数：

```
Dim Lsb As Long
Lsb = ADBuffer (0) And 16383
Volt =    (20000.00/16383)* Lsb – 10000.00    ' 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 ： = ADBuffer [0] Xor $2000 And $3FFF;　　// 取得标准 LSB 原码

Volt ： = Lsb * (20000.00/16384) – 10000.00; // 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 ADBuffer 缓冲区中的数据排放规则

当首末通道相等时，即为单通道采集，假如FirstChannel=5, LastChannel=5,其排放规则如下：

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

两通道采集(CH0 – CH2)

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

四通道采集(CH0 - CH3)

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

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取AD数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐问题，尤其任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数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ReadDeviceAD函数读回，即便不考虑是否能一次读完的问题，但对用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

<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>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></td><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</td></tr>
</table>

第七章　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 USB2089.h 和 Sys.cpp)。

[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]

其简易程序默认存放路径为：系统盘\ART\USB2089\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 USB2089.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\USB2089\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 USB2089.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\USB2089\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

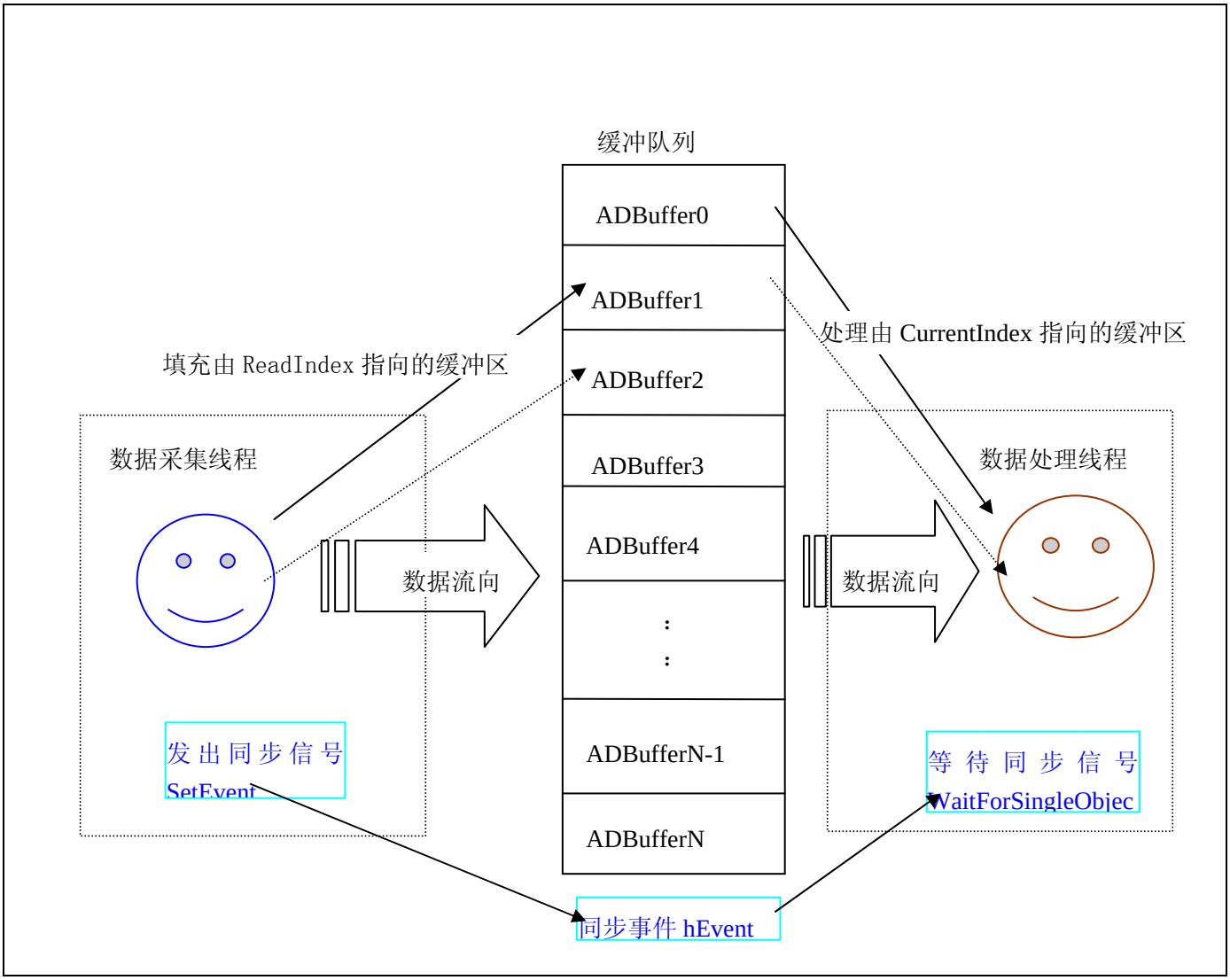
第八章 基于 USB 总线的大容量连续数据采集详述

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ADBuffer[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
```

第九章　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

一、 创建 VB 子线程

♦ **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 "USB2089" (ByVal 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 USB2089.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 "USB2089" (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);

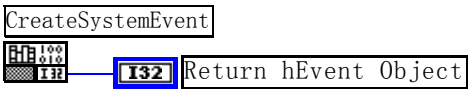
Visual Basic:

Declare Function CreateSystemEvent Lib " USB2089 " () As Long

Delphi:

Function CreateSystemEvent():Integer; StdCall; External 'USB2089' 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。

返回值：若成功，系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

Visual C++程序举例:

```
                :
HANDLE hEvent;
hEvent=CreateSystemEvent ()
if(hEvent==INVALID_HANDLE_VALUE)
      MessageBox("创建内核事件失败…");
      return;       // 退出该函数
}
            :
```

Visual Basic 程序举例:

```
        :
hEvent = CreateSystemEvent() ' 创建内核事件
If hEvent = INVALID_HANDLE_VALUE Then
      MsgBox "创建事件对象失败"
      Exit Sub
 End If
:
```

二、 文件对象操作函数

♦ **创建设备文件对象**

函数原型：

Visual C++:

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

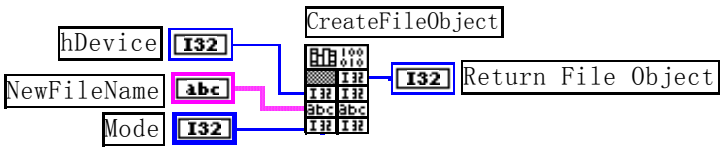
Visual Basic:

```
Declare Function CreateFileObject Lib "USB2089" ( ByVal hDevice As Long, _
                                                  ByVal NewFileName As String, _
                                                  ByVal Mode As Long) As Long
```

Delphi:

```
Function CreateFileObject (hDevice : Integer; NewFileName: string; Mode: Integer):LongInt;
        Stdcall; external 'USB2089' name ' CreateFileObject ';
```

LabView:



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:\\USB2089\\Data.Dat”， 在 Basic 中，其语法格式如：”C:\USB2089\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)

USB2089_modeRead 只读文件方式

USB2089_modeWrite 只写文件方式

USB2089_modeReadWrite 既读又写文件方式

USB2089_modeCreate 如果文件不存可以创建该文件，如果存在，则重建此文件，并清 0

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 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)
```

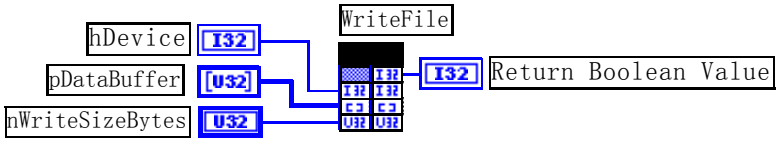
Visual Basic:

```
Declare Function WriteFile Lib "USB2089" (By REF hFileObjectAs Long, _
                                          ByVal pDataBuffer As Integer,_
                                          ByVal nWriteSizeBytes As Long ) As Boolean
```

Delphi:

```
Function WriteFile( hFileObject: Integer;
                    pDataBuffer:PWordArray;
                    nWriteSizeBytes: LongWord):Boolean;
                    Stdcall; external 'USB2089' name 'WriteFile';
```

LabView:



功能：通过向设备对象发送写盘消息，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：

hDevice 设备对象句柄,它应由CreateDevice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： CreateDevice CreateFileObject WriteFile
ReadFile ReleaseFile ReleaseDevice

◆ **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,)
```

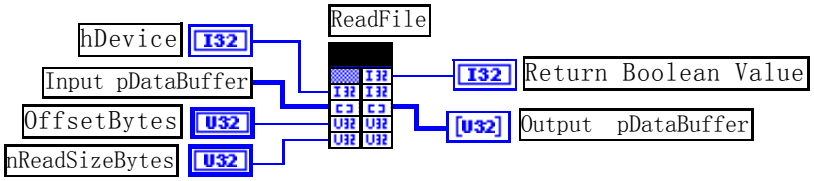
Visual Basic:

```
Declare Function ReadFile Lib "USB2089" ( ByVal hFileObjectAs Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal OffsetBytes As Long, _
                                          ByVal ReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile( hFileObject: Integer;
                   pDataBuffer:PWordArray;
                   OffsetBytes:LongWord;
                   nReadSizeBytes:LongWord):Boolean;
                   Stdcall; external 'USB2089' name 'ReadFile';
```

LabView:



功能：通过向设备对象发送写盘消息，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：

hDevice 设备对象句柄,它应由CreateDevice创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针。

OffsetBytes 指定从文件始端起所偏移的读位置。

ReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)，其取值范围为[1, 65535]。

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　CreateFileObject　WriteFile　ReadFile　ReleaseFile　ReleaseDevice

注意：　它读出磁盘的数据将放置于InitDeviceAD中的PADBuffer用户内存区前端。

♦ **释放设备文件对象**

函数原型：

Visual C++:

```
BOOL ReleaseFile (HANDLE hFileObject)
```

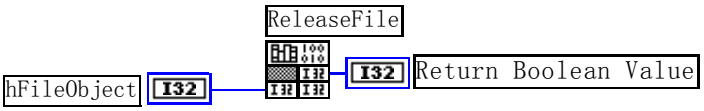
Visual Basic:

```
Declare Function ReleaseFile Lib "USB2089" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
function ReleaseFile ( hFileObject: Integer):Boolean;
                    Stdcall; external 'USB2089' name 'ReleaseFile';
```

LabView:



功能：释放设备文件对象。

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　CreateFileObject　WriteFile　ReadFile　ReleaseFile　ReleaseDevice

三、取得指定磁盘的可用空间

函数原型：

Visual C++:

```
LONGLONG GetDiskFreeBytes（LPCTSTR DiskName )
```

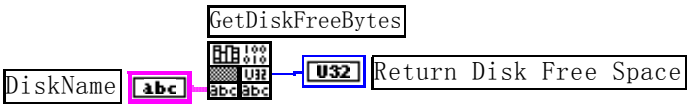
Visual Basic:

```
Declare Function GetDiskFreeBytes Lib "USB2089" (ByVal DiskName As String ) As Boolean
```

Delphi:

```
Function GetDiskFreeBytes ( DiskName: String):Boolean;
                        Stdcall; external 'USB2089' 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：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 0 的长整型值,否则返回负值，用户可用 GetLastError 捕获错误码，注意使用 64 位变量。

附录 A　LabView/CVI 图形语言专述

第一章　图形化编程语言 LabVIEW 环境及其开放性

图形化编程语言 LabVIEW 是著名的虚拟仪器开发平台。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。本文对 LabVIEW 开发环境及其开放性作一简述。

第一节 LabVIEW 概述

LabVIEW 使用了一种称为 G 的数据流编程模式，它有别于基于文本语言的线性结构。在 LabVIEW 中执行程序的顺序是由块之间的数据流决定的，而不是传统文本语言的按命令行次序连续执行的方式。

LabVIEW 程序称为虚拟仪表(Vitual Instrument)程序，简称 VI。VI 包括 3 个部分：前面板、框图程序和图标/连接口。前面板用于输入数值和观察输出量。

输入量被称为 Controls，输出量被称为 Indicators。用户可以使用许多图标，如旋钮、开关、文本框和刻度盘等来使前面板易看易懂。如图 1 所示，它是一个温度计程序(Thermometer VI)的前面板。

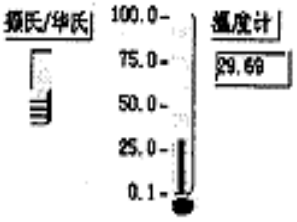


图 1　温度计的前面板

每一个前面板都伴有一个对应的框图(block diagram)程序。框图程序使用图形编程语言编写，可以把它理解成传统程序的源代码。框图中的程序可以看成程序节点，如循环控制、事件控制和算术功能等。这些部件用连线联接，以定义框图内的数据流动方向。上述温度计程序的框图程序如图 2 所示，框图程序的编写过程与人的思维过程非常接近。LabVIEW 提供的 3 类可移动的图形化工具模板用于创建和运行程序，它们是工具(Tools Palette)、控制(Controls Palette)和功能(Functions Palette)等。工具模板用于创建、修改和调试程序(如连线、着色等)；控制模板用来设计仪器的前面板(如增加输入控制量和输出指示量等)；功能模板用来创建相当于源代码的 LabVIEW 框图程序(如循环、数值运算、文件 I/O 等)。LabVIEW 平台的特点可归结为以下几个方面：

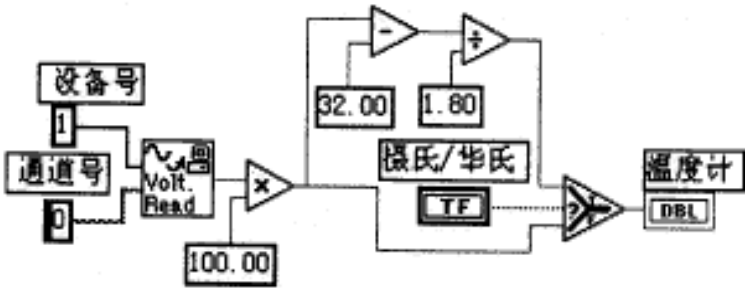


图 2　温度计的框图程序

(1)图形编程方式：使用直观形象的数据流程图式的语言书写程序源代码；

(2)提供程序调试功能，如设置断点或探针，单步执行，语法检查等；

(3)拥有数据采集、仪器控制、分析、网络、 ActiveX 等集成库；

(4)继承传统编程语言结构化和模块化的优点，这对于建立复杂应用和代码的可重用性来说是至关重要的；

(5)提供 DLL 库接口、CIN 节点以及大量的仪器驱动器、网络通信 VIs 与其它应用程序或外部设备进行连接；

(6)采用编译方式运行 32 位应用程序；

(7)支持多种系统平台，如 Macintosh、HP-UX、SUN SPARC 和 Windows 3.x/95/NT 等，LabVIEW 应用程序能在上述各平台之间跨平台进行移植；

(8)提供大量的函数库及附加工具。如数学函数、字串处理函数、数组运算函数、文件 I/O、高级数字信号处理函数、数据分析函数、仪器驱动和通信函数等。

第二节 程序设计结构

(1)层次化结构

LabVIEW 是模块化程序设计语言，用户可以把一个 VI 程序创建成自己的一个图标/接口(即 VI 子程序) ，然后被其它 VI 程序所调用。用这种方法可设计出一个有层次关系的 VIs 或子 VIs ，而且调用阶数是无限制的。

(2)并行工作

LabVIEW 是一个多任务的软件系统，当创建具有同步工作的程序块时，就可交互地运行并行 VIs 程序。

(3)常规语法结构：While Loops，For Loop，Case 结构，顺序结构等;

(4)基于文本的公式结(Formula Node)

公式结是一种用于书写数学公式的文本编辑框。

第三节 LabVIEW 的运算形式

(1)模块化图标运算

LabVIEW 中的图标/连接口表示一定的函数功能，将若干个图标/连接口组合起来就可进行有关运算，如算术、布尔逻辑、比较和数组运算、数值运算(三角函数、对数等)、字符串运算和文件 I/O 等；

(2)公式运算

使用公式结运行数学公式。公式结包含一个或多个公式表达式，各公式之间用分号"；"隔开。公式表达式使用了一种类似于大多数基于文本编程语言(如 BASIC 语言)的算术表达式的语法。如图 3 所示，输入变量为 m、b 和 x，经公式结运算后的输出变量为 y1 和 y2。公式结中使用的变量或公式的数量是无限制的。

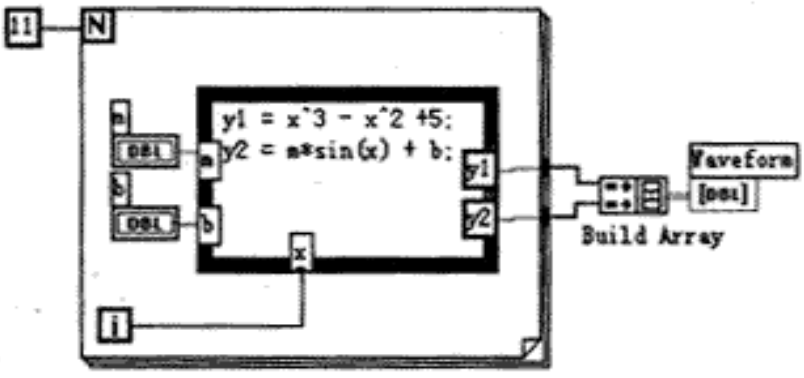


图 3　公式结运算例子

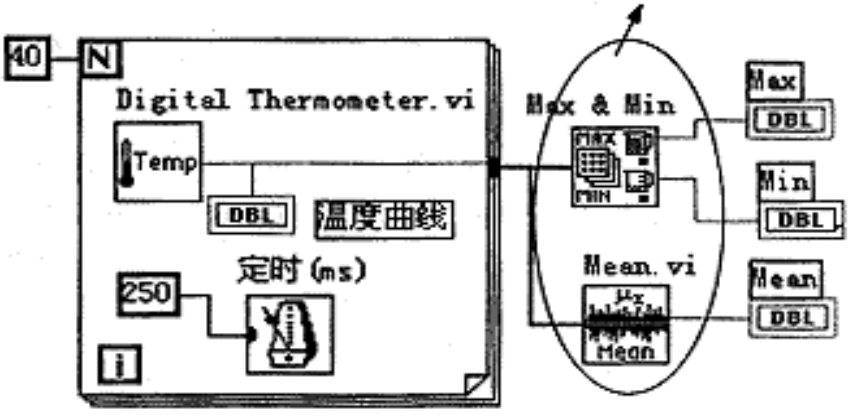


图 4 使用功能子模块进行温度曲线分析

(3) 使用集成库的功能子模板完成运算

LabVIEW 中集成了大量的生成图形界面的模板，丰富实用的数值分析、数字信号处理功能，以及多种硬件设备驱动器(包括 RS232、GPIB、VXI、DAQ 卡和网络等)。用户不需了解有关运算细节就能直接使用这些功能子模块，这对于编程工作来说，可节省了大量的时间开销。如图 4 所示，使用两个功能子模块进行温度曲线分析，以求出数组的最大值、最小值和平均值。

(4)通过链接 DLL 形式的代码进行运算

LabVIEW 提供 DLL 库接口和 CIN 节点来使用户有能力在该平台上使用其它软件开发平台生成的模块。即用户可通过其它开发平台(如 BC++)建立一个子例程，并生成动态链接库 DLL，然后与 LabVIEW 框图程序进行链接。LabVIEW 的这一开放性，为用户自行编写某些软件模块提供了方便。如用户可通过 C++/C 语言为某一新设备开发通信及驱动程序，或编写一控制算法软件，然后链入 LabVIEW 程序。

第四节 LabVIEW 的开放性

LabVIEW 是开放型的开发环境，它拥有大量的与其它应用程序进行通信的 VI 库。因此，LabVIEW 可从众多的外部设备获取或传送数据，这些设备包括 GPIB、VXI、PXI、串行设备、PLCs、和插件式 DAQ 板等；LabVIEW 甚至可以通过 Internet 取得外部数据源。

(1)DLLs

在 Windows 或其它平台下调用内部或外部的 DLL 形式的代码或分享其它平台(包括 Windows)中的库资源；使用 CodeLink，同样可自动分享在 LabWindows/CVI 中开发的 C 程序库；

(2)ActiveX，DDE,SQL

使用自动化 ActiveX、DDE 和 SQL，与其它 Windows 应用程序一起集成用户的应用程序；

(3)远程通信：Internet，TCP/IP

使用 TCP/IP 和 UDP 网络 VIs，与远程应用程序进行通信；在用户的应用程序中融入 e-mail、FTP 和浏览器等；通过远程自动控制 VIs，可远程操作其它机器上的分散 VIs 的执行。

第五节 调试工具

(1)语法检查：如果程序有错，则无需编译，工具栏的运行按钮就会出现一个折断的箭头。点击该箭头，就会给出错误列表信息。

(2)运行灯高亮：运行灯高亮用于在单步模式下跟踪框图程序中的数据流动。

(3)单步执行：按顺序一个节点一个节点地执行程序。

(4)探针：探针工具用来查看程序流经某一根连线上的数据。

(5)断点：设置断点可在程序的某一地方终止程序的执行，以观察调试部分的执行结果。

综上所述，列出 LabVIEW 的开发环境表，如表 1 所示。

第六节 工具软件包

NI 公司及其协作单位提供众多的软件工具箱和支持软件，用于扩展支持 LabVIEW。这些工具软件包有：

(1)常用工具箱

Application Builder:创建可单独运行的应用程序；

控件与指示器

按钮/开关 LED，滑块/数显，计量器/刻度盘/旋钮，水槽/温度表，曲线图/图表，表格/数组，密度图，菜单/列表/环，文本框；

仪器控制

GPIB，VXI，Serial，CAMAC，PLC 等 600 多种仪器驱动器；

文件 I/O 电子表格，二进制，ASCII 码，日志；

开放性联接

Internet，SQL，TCP/IP，Activex，DLLs，DDE 等；

数据采集

DAQ，单点输入/输出，波形采集/发生，图像采集，信号调理，触发/定时，TTL/CMOS 输入/输出，数字图案发生，数字握手，脉冲发生，事件计数，边界检测，周期和脉宽测量等；

程序设计结构

While Loops，For Loop，Case 结构，顺序结构，基于文本的公式结；

程序设计原则

算术运算，布尔逻辑，数组处理，串函数，时间/日期函数，多数据类型结构，用户子例程；

分析

信号发生，信号处理，图象处理，曲线拟合，窗体，过滤，线性，统计等；

优化与应用程序管理

用于存储管理和执行时间跟踪的 Profiler，在所有平台上的 TURE 编译性能，源代码控制，文档打印等；

调试

断点，探针，单步模式，执行高亮，帮助窗口，在线帮助 Test Suite:包括 600 多个仪器驱动程序软件包、连接到 30 多个本地或远程数据库的数据库连接工具、程序性能测试和分析软件等；

Test Executive:

多用途的附加软件包。使用该软件包，可以控制程序执行的次序，生成应用程序。按照自己的特定要求和标准来设置应用程序。在保持扩展升级兼容性的前提下，允许用户增强操作和人机接口；

SQL:用于与本地或远程数据库的直接访问；

SPC:过程控制中统计方法应用程序库；

Internet:把 VI 程序转换成可在 Internet 上执行的应用程序；

PID:给 LabVIEW 加入复杂的控制算法。该软件包带有许多误差反馈及外部复位的 PID 算法，同时含有超前-滞后补偿和设置点斜率生成等功能；

Picture Control: 一个多功能的图形软件包，用于生成前面板显示，如特殊的棒形图、饼形图和 Smith 图表等。

(2)分析工具箱

HIQ：一个交互式的工作环境，可以对数学、科学计算和工程问题的数据进行组织、可视化处理。HIQ 集成了数学运算用户接口控制、数值分析、矩阵运算及二维、三维和四维图形处理；

Signal Processing Suite: 提供数据处理功能和高级信号处理工具。如数字滤波器、1/3 倍频程分析和动态信号分析等；

G Math:算术运算、数据分析和数据可视化。如常微分方程、最优化、变换和过程控制模拟等；

Image Processing:提供图象处理功能和机器视觉功能等。

第七节 总结

LabVIEW 是开放型模块化程序设计语言，使用它可快速建立自己的仪器仪表系统，而又不用担心程序的质量和运行速度。LabVIEW 既适合编程经验丰富的用户使用，也适合编程经验不足的工程技术人员使用，所以被誉为工程师和科学家的语言。

第二章 图形化编程加标准 C 语言 LabWindowsCVI 环境及其开放性

LabWindows/CVI 很多功能与 LabView 差不多，比如提供了很多虚拟仪器控件、分析工具箱、各种数据处理算法等。但不同的是，LabWindows/CVI 是基于标准 C 语言提供的虚拟仪器开发平台。它不仅具有 LabView 在图形化、可视化方面的编程特点，它更具有 C 语言代码高效运行的特点。因此，对于更加高速运行，复杂控制的系统，选用 LabWindows/CVI 显得更得心应手。

关于我公司的 LabView/CVI 驱动请参考相应演示程序。