DAM-E3000

Win95/98/NT/2000 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

第一章 版权信息

第二章 绪 论

第一节 使用纲要

一、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，便可如能所需，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如 Win32 API 的函数，具有相当的灵活性、可靠性和高效性。诸如 GetDeviceVersion、WriteDeviceDA、ReadDeviceAD 等。而底层用户函数如 WriteDeviceChar、ReadDeviceChar、……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上可以必参考硬件说明书。

二、如何管理 DAM-E3000 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 CreateDevice 函数创建一个设备对象句柄 hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给其他函数，如 GetDeviceVersion 可以使用 hDevice 句柄以获取设备设备版本信息，ReadDeviceAD 函数可以用 hDevice 句柄实现对 AD 数据的采样读取，SetDeviceDO 函数可用实现开关量的输出等。最后可以通过 ReleaseDevice 将 hDevice 释放掉。

三、如何取得 AD 数据

当您有了 hDevice 设备对象句柄后，便可用 ReadDeviceAD 函数初始化 AD 部件，设置读取 AD 数据的通道。您只需要对这个 nChannel 参数进行设置，便可以读取相应通道的数据。然后便可用 ReadDeviceAD 反复读取 AD 数据以实现连续不间断采样。当您需要停止采集时，只需要退出采集循环就可了。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）。具体执行流程请看下面的图 2.1.1。

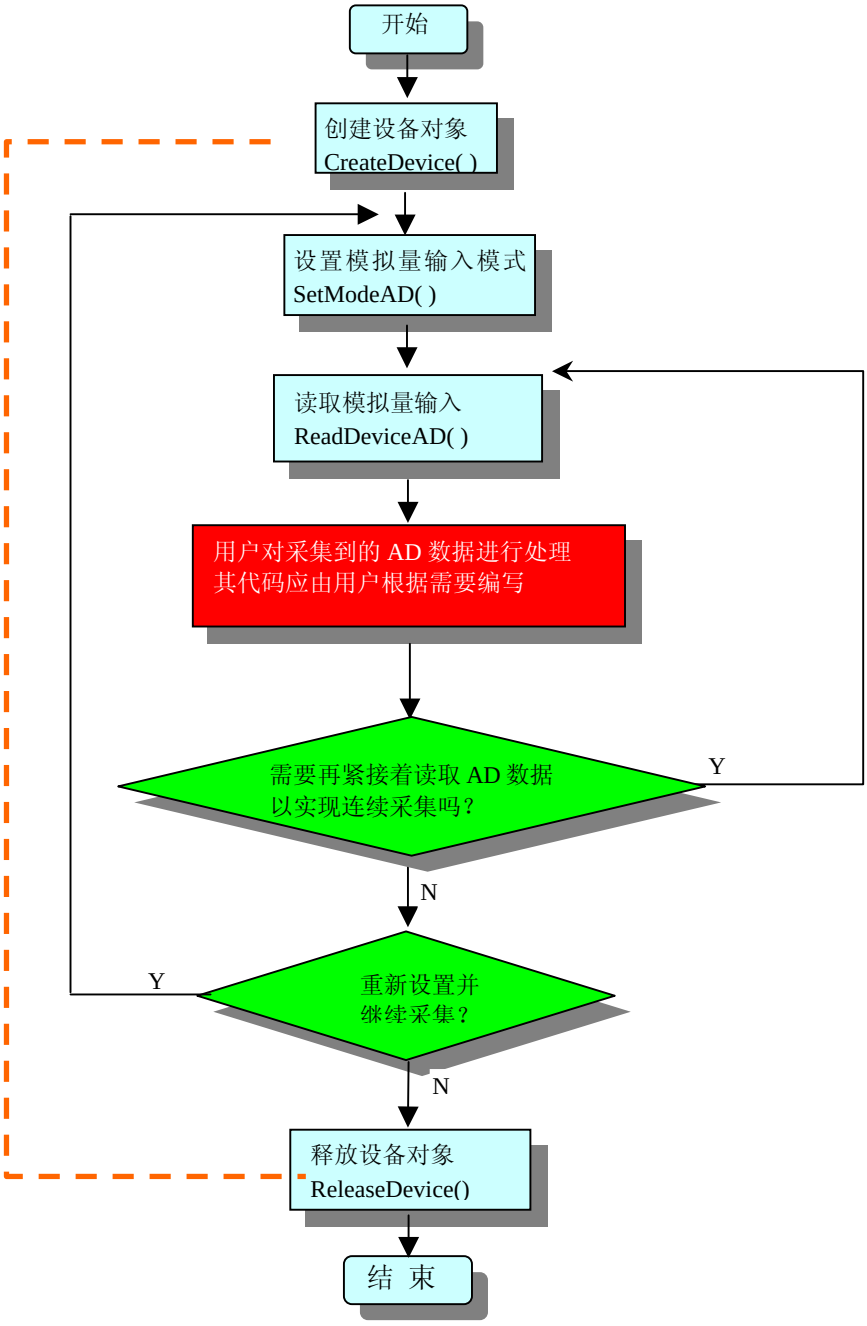


图 2.1.1 AD 采集实现过程

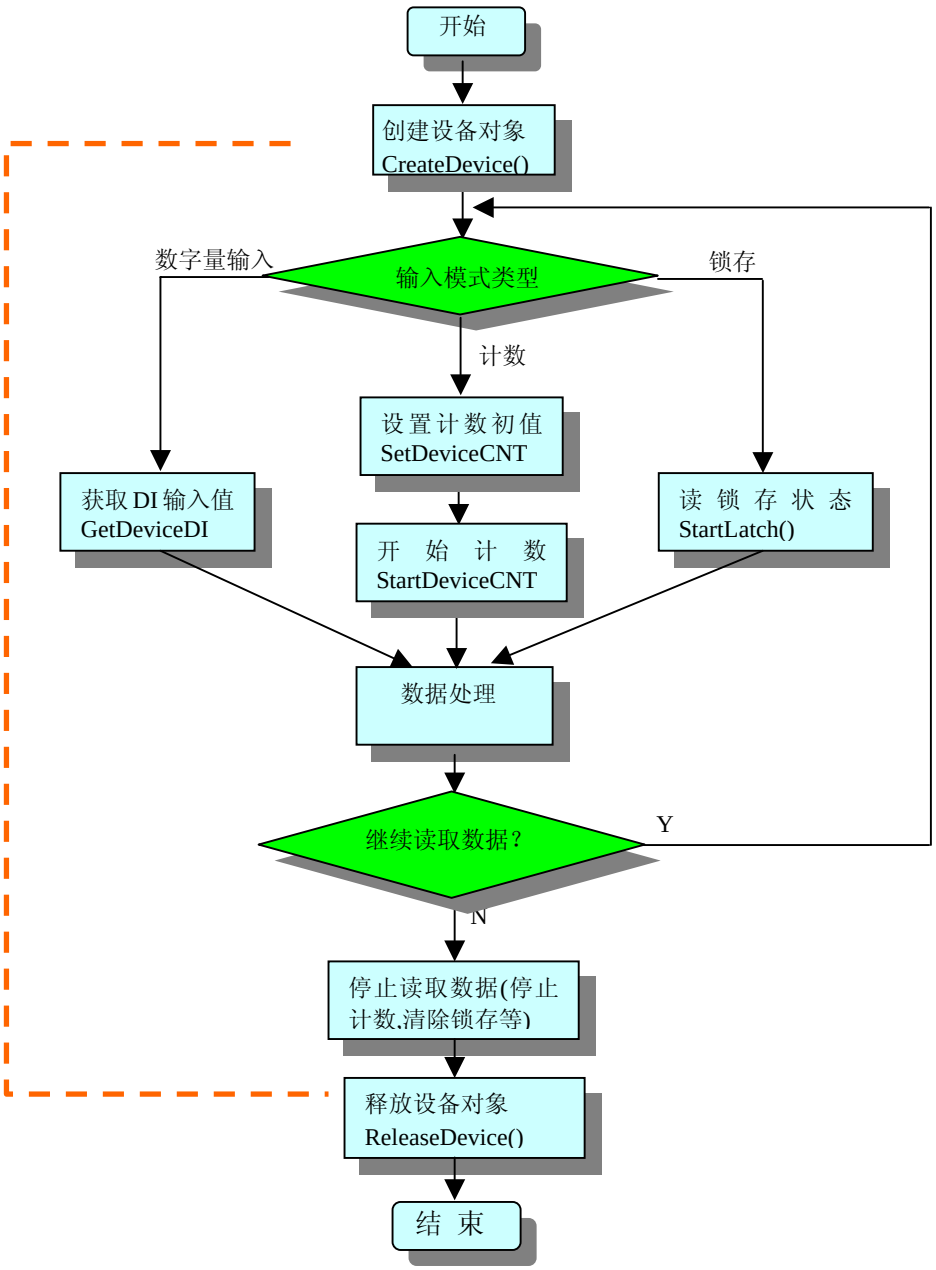
注意：图中较粗的虚线表示对称关系。如红色虚线表示 CreateDevice 和 ReleaseDevice 两个函数的关系是：最初执行一次 CreateDevice，在结束是就须执行一次 ReleaseDevice。

四、如何实现 DA 的简便输出

当您有了 hDevice 设备对象句柄后，首先用 SetModeDA 函数设置 DA 的输出模式，然后反复调用 WriteDeviceDA 函数输出每一个 DA 数据。

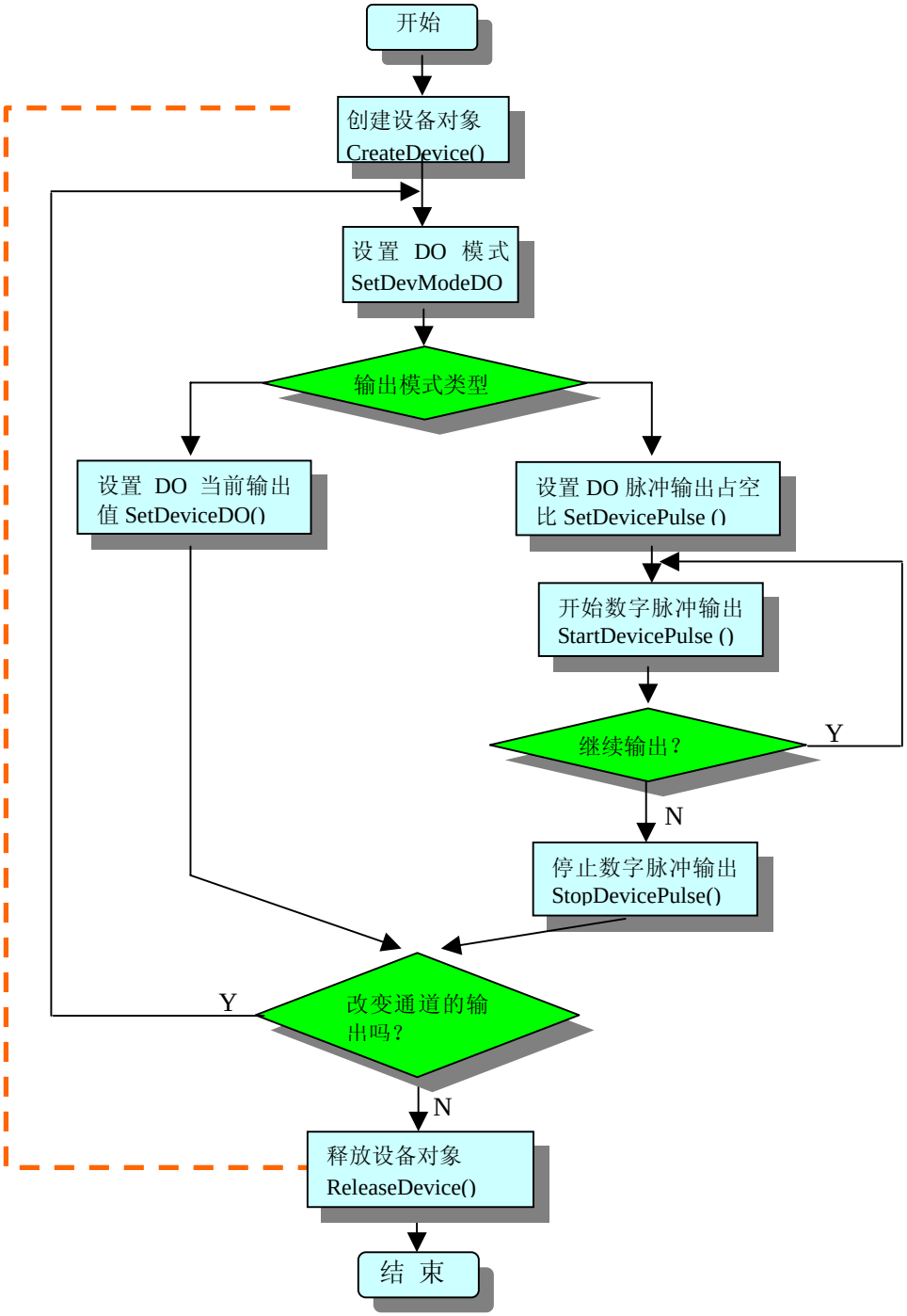
五、如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 GetDeviceDI 函数实现开关量的输入操作，其各路开关量的输入状态由其数组参数 DIStatus 决定。具体执行流程请看下面的图 2.1.2。



2.1.2 DI 数字量输入

同样，您也可用 SetDeviceDO 函数实现开关量的输出操作，其各路开关量的输出状态由其结构体 DAM3000E_PARA_DO 中的成员变量 DO0-DO15 决定。具体执行流程请看下面的图 2.1.3。



2.1.3 DO 数字量输出

六、如何使用软件看门狗

当您有了 hDevice 设备对象句柄后，首先用 SetWatchdogTimeoutVal 来设置看门狗的溢出值，然后用函数 EnableWatchdog 使看门狗有效。在使用看门狗时，可以用 ResetWatchdogStatus 对看门狗进行复位。

同时，您必须用函数 HostIsOK 函数按一定的时间间隔（小于看门狗溢出时间）发送主机正常命令，否则，看门狗将溢出，。在看门狗溢出时，将输出您预先设置的安全值。

七、哪些函数对您不是必须的

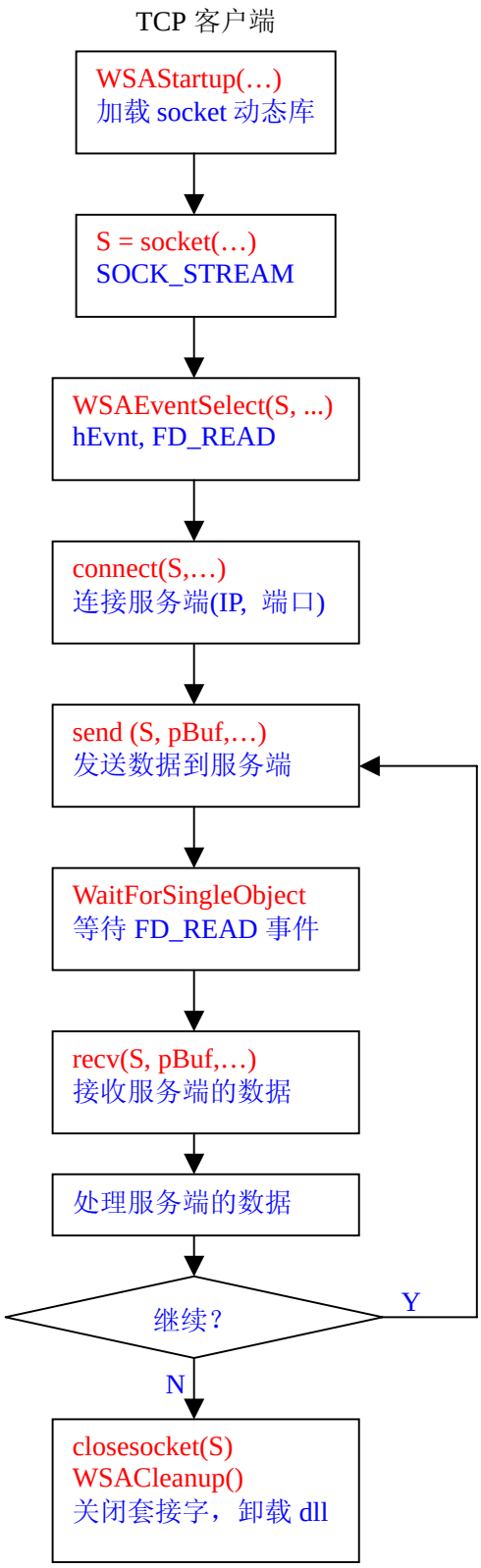
当公共函数如 WriteDeviceChar, ReadDeviceChar 等一般来说都是辅助性函数，除非您要直接使用命令字对设备进行操作。WriteDeviceChar, ReadDeviceChar 等函数，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您检测设备是否运行正常，并从中获取必要的信息。

第二节 驱动程序功能概述

一、 连接方式

DAM-E3000 系列板卡提供了两种连接方式，分别是面向连接的 MODBUSTCP 方式。

二、 有连接的异步模式（TCP）



三、套接字 SOCKET

Socket 实际上是一个通信端口，一个 Socket 是通讯的一端。网络通信将通过各自己的套接字相联系。在实际应用当中，就像使用文件句柄一样。应用程序向操作系统申请，然后由操作系统来分配本地唯一的端口号(1024～5000)，然后可以对 Socket 句柄进行读写操作。

四、异步模式（非阻塞模式）

本驱动程序采用异步模式，避免阻塞。使用函数 WSAEventSelect()设置一个或多个网络事件，当设置的网络事件发生时，能过异步选择函数 WSAEventSelect()绑定的事件自动变成有信号。通过枚举网络事件，如果是 FD_READ(已经收到数据)，则可以通过读函数 recv()进行读取。

五、后台工作方式

我们的驱动程序为用户提供了后台工作方式进行数据传输，这样可以保证您的前台应用程序能实时高效的进行数据处理。后台方式的特点是在进行数据采集和传输过程中不占用客户程序的任何时间，当采集的数据长度达到客户指定的值时便触发客户事件，客户程序接受该事件便开始进行数据处理。在数据处理的同时，驱动程序依然在进行下一批数据的传输，即实现了并行操作，极大的提高了数据的吞吐量和计算机系统的整体处理能力。

六、与设备无关性

通过总结各数据采集卡的的共同特点，设计了完全一致的接口方式，可以让您的应用程序不仅能适应您所购买的我公司第一种产品，同时也能不经修改地适应我公司的其他同类产品。所以可以保证您的应用程序在我们的硬件产品基础上极为容易地进行功能和应用扩展，节省您的大部分软件投资，极大的缩短工程开发周期。

七、驱动程序的坚固性

我们的驱动程序都是经过严密彻底的测试和验证，并经部分用户试用之后，确认没有任何问题后才予以正式发行的，所以当您使用起来应该有十足的安全感。

八、函数接口数量

我们提供的驱动程序用户接口不象有些公司提供的多达上百个函数，使您眼花缭乱、不知所从。我们所提供的关键函数实际上只有不到 5 个，其它的都是一些辅助性的函数，用户可用可不用。其原因是我们把所有复杂的大量的工作为您一一解决，尽可能地把复杂的问题封装在驱动程序内部，但同时也不缺乏灵活性，故而使您编程容易、使用方便。通常情况下，您稍稍熟悉一下我们的设备驱动程序说明书，您花上一刻钟时间便可以用我们的驱动程序接口编写出对设备访问的基本代码。

九、安装程序特点

关于驱动程序的安装方式我们采用大多数 Windows 应用程序所使用的标准模式，因而简捷、方便、直观。您只需执行安装盘上的 Setup.exe 启动文件即可进行驱动程序的安装工作。在安装过程中您设置好安装目标路径以及文件夹名称等信息后，安装程序便自动而又快捷地为您安装好驱动程序，随后您便可以用驱动程序接口编写应用程序或用我们提供的简易测试程序测试设备了。

十、多语言编程环境

本系统提供 Visual C++，C++ Builder, Visual Basic，Delphi, LabView/CVI 的函数接口，使您完全可以根据自己的需要和喜爱选择合适的编程语言。请记住，您得使用 32 位编程模式。另外，局于编幅所限。

十一、我公司动态库与其他公司动态库的比较

值得注意的是，我们的 DLL 库不同于其它许多公司所编写的那样，只是对动态库的简单直接地调用,其硬件控制、数据传输代码都放在 DLL 中，那么其代码的优先执行级别跟一般的用户程序是一样的，它总要定期地、不断地被系统级任务调度器调度，所以当这些代码在负责传输数据时往往被瞬时中断，有时这个时间还很长，故此，极有可能造成丢点的严重现象。为了解决这些问题，在 **Win95、Win98** 环境下，我们没有把硬件控制、数据传输代码简单地放在 DLL 中，而是通过动态虚拟技术以 **VxD** 的形式放在了 Windows 系统空间中，以 CPU 的 0 级环级别同系统代码协同工作，也就是说它可以获得与任务调用器一样的级别，且不受任务调度器的调度管理。在 **NT** 环境下，我们通过**微内核技术**把硬件控制、数据传输代码以微内核代码（简称**微代码**）形式放在 NT 的内核模式中，成为 NT 操作系统的一部分,并可根据代码的重要程度进一步迅速临时提升 CPU 的 IRQL 级别,使这些代码以高优先级,高速度工作，极大的提高了数据采集和传输的质量。而我们的 DLL 的主要任务不是采集数据，而是对驱动程序的全面封装，对用户负责简化所有复杂的繁琐的操作细节，特别是 Windows 底层管理，提供简洁一致的函数接口供用户使用。它具体表现在从用户空间到系统空间（Windows95,98）、从用户模式到内核模式(Windows NT)、从 CPU 的 3 级环到 0 级环(Windows)等相互间的转换以及设备 I/O 请求的来回传递。所以，我们的驱动程序不是 DLL，而是形如***.VxD**（Win95）或***.SYS**(NT)的代码文件。通过这样的技术便能实现设备所有功能，极大范围地满足用户需要。

十二、跨平台设计

至今，Windows98 与 Windows 2000 是两大主流操作系统，它们各有其优点，但随着计算机的进一步网络化以及追求高可靠性和高稳定性，Windows2000 将成为用户更好的操作系统。所以我们尽力做到了跨平台设计，使您的用户程序基本不作修改，就象 Microsoft Word 软件一样，便可运行在其他平台上。

十三、自动卸载功能

在您已安装了本软件系统后，如果不再准备使用本系统，您可以通过我们为您提供的组件 unInstallShield 从 Windows 系统中自动卸载本软件系统。

十四、LabView/CVI 支持

LabView/CVI 是美国国家仪器公司(National Instrument)的虚拟仪器开发平台，特别是基于图形化编程的 LabView 语言，在测量、工控、虚拟仪器方面受到广大工程师和用户的青睐。其全球销售量仅次于 C++语言。我们自主开发的硬件（PCI、USB、ISA 总线系列）产品提供了基于 LabView 的驱动软件接口模块，与 LabView 软件平台完全兼容，让您轻松实现图形化编程。

十五、所提供的组件

如果您采用 Typical 安装选项，那么您一般可以得到我们为您提供的如下组件：

Hardware Help 硬件使用说明 Word 帮助文档；

ReadmeFile 安装目录等信息简介；

Software Help 软件使用说明 Word 帮助文档；

Test Application 基于 Microsoft Visual C++代码的硬件测试应用程序；

Visual C++ Sample Microsoft VC++演示程序（这个程序对驱动程序演示说明最全面）；

Visual Basic Microsoft VB 演示及接口程序文件(USB2000A.Bas)

C++ Builder Borland C++ Builder 演示程序；

Delphi Borland Delphi 演示及接口程序文件（USB2000A.Pas）；

LabView 美国国家仪器公司(National Instrument)的虚拟仪器开发平台的演示程序及接口模块程序

UnInstallShield 本软件卸载应用程序；

第三节 本驱动程序软件的关键文件

（WinDir 指 Windows 的系统根目录, UserDir 为本驱动软件的用户安装根目录）

文件名	文件类型及功能	适用的操作系统	文件位置
DAME3000.VxD	动态虚拟设备驱动程序库	Window95/98	WinDir\System
DAME3000.Sys	Win32 标准设备驱动 WDM 模式的设备驱动程序库	Windows NT/2000	WinDir\System32\Drivers
DAME3000.Dll	底层驱动程序库的用户级函数接口封装所用的动态库。	所有操作系统	WinDir\System
DAME3000.Lib	基于 Microsoft Visual C++工程开发环境的驱动程序函数接口输入库。	所有操作系统	UserDir\Include 或 UserDir\Samples\VC…
DAME3000.Lib	基于 Borland C++ Builder 工程开发环境的驱动程序函数接口输入库。	所有操作系统	UserDir\Samples\C_Builder
DAME3000.Bas	基于 Microsoft Visual Basic 工程开发环境的驱动程序函数接口输入模块文件	所有操作系统	UserDir\Samples\VB
DAME3000.Pas	基于 Borland Delphi 工程开发环境的驱动程序函数接口输入单元文件。	所有操作系统	UserDir\Samples\Delphi
DAME3000.VI	基于 National Instrument LabView 工程开发环境的驱动程序函数接口输入部件文件。(**只是外挂驱动接口**)	所有操作系统	UserDir\Samples\LabView

第三章 设备操作函数接口介绍

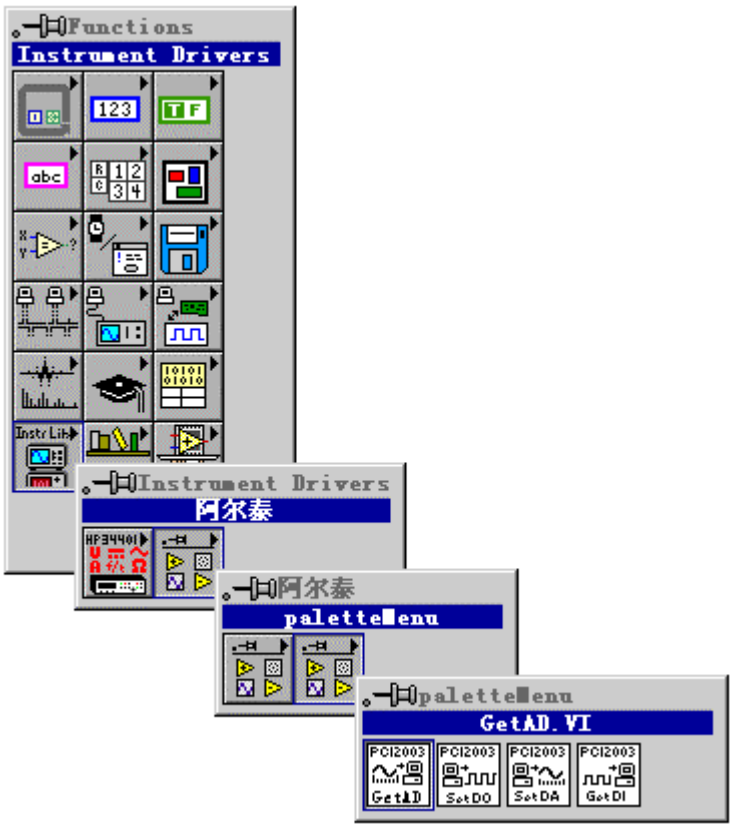
由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节、只关心 AD、DA、DI、DO 等，然后就能通过一两个简易的采集函数便能轻松得到所需要的数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群需要直接跟设备端口打交道的用户称之为底层用户。因此总的看来，上层用户要求简单，快捷，他们最希望他们在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如 SetInputModeAD）告诉设备需要输入的模式是什么等，然后便可以用 ReadDeviceAD 函数即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的 Bit 位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是

这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉 TCP/IP、UDP 复杂的控制协议，同时还可以省掉您许多繁琐的工作。这个时候您便可以用 CreateDevice 创建的句柄，再根据硬件使用说明书中各种命令字的功能说明，然后使用 WriteDeviceChar 和 ReadDeviceChar 对这些模块进行读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先得明白自己是上层用户还是底层用户，因为在《**第一节 接口函数列表**》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。**此功能由于 LabView 自身版本兼容的问题，我们不便提供内嵌式驱动，如果用户确有此要求，请与我们的代理商或公司总部联系，但我们不保证完全免费。**

关于LabView的外挂式驱动和内嵌式驱动更详细的叙述，请参考附录A的《LabView驱动程序接口》章节。



LabView 内嵌式驱动接口的获取方法

第一节 设备驱动接口函数列表（每个函数省略了前缀“DAM3000E_”）

表 3-1 设备驱动接口函数列表

函数名	函数功能	备注
设备对象操作函数		
CreateDevice	创建以太网设备对象	上层及底层用户
ReleaseDevice	关闭设备，且释放以太网设备对象	上层及底层用户
GetDeviceVersion	取得设备版本信息	上层用户
GetNetworkConfig	获得设备的网络配置信息	上层用户
SetNetworkConfig	修改设备网络配置信息	上层用户
AD 读取函数		
ReadDeviceAD	读取模拟量输入	上层用户

SetModeAD	设置模拟量输入模式	上层用户
GetModeAD	获得模拟量输入模式	上层用户
SetLowLimitVal	设置下限报警值	上层用户
GetLowLimitVal	获得报警下限值	上层用户
SetHighLimitVal	设置上限报警值	上层用户
GetHighLimitVal	获得报警上限值	上层用户
GetAlarmPulse	获得模拟量输入报警电平	上层用户
SetAlarmPulse	设置模拟量输入报警电平	上层用户
GetAlarmSts	获得报警状态	上层用户
DA 输出函数		
DI 输入函数		
GetDeviceDI	获得 DI 值	上层用户
GetDeviceCNT	获得计数值	上层用户
SetDeviceCNT	设置计数初值	上层用户
GetModeCNT	获得计数方式	上层用户
SetModeCNT	设置计数方式	上层用户
StartDeviceCNT	开始计数	上层用户
StopDeviceCNT	停止计数	上层用户
GetEnableStsCNT	获得计数使能状态	上层用户
GetLatchEnableSts	获得锁存使能状态	上层用户
StartLatch	启动锁存	上层用户
StopLatch	停止锁存	上层用户
GetLowLatchSts	获得下降沿锁存状态	上层用户
GetHiLatchSts	获得上升沿锁存状态	上层用户
DO 输出函数		
SetDeviceDO	设置 DO 当前输出值	上层用户
GetDeviceDO	读取 DO 输出值	上层用户
StartDevicePulse	开始数字脉冲输出	上层用户
StopDevicePulse	停止数字脉冲输出	上层用户
GetDevicePulse	取得 DO 脉冲占空比	上层用户
SetDevicePulse	设置 DO 脉冲输出占空比	上层用户
GetPulseEnableSts	获得脉冲使能状态	上层用户
GetSafeValueDO	读 DO 安全值	上层用户
SetSafeValueDO	修改设备中的 DO 安全值	上层用户
GetPowerOnValueDO	获取 DO 上电初始值	上层用户
SetPowerOnValueDO	修改 DO 上电初始值	上层用户
软件看门狗函数		
HostIsOK	发 Host is OK 命令给下位机	上层用户
EnableWatchdog	使能看门狗	上层用户
CloseWatchdog	关闭看门狗	上层用户
GetWatchdogSts	读取看门狗状态	上层用户
ResetWatchdog	复位看门狗	上层用户
SetWatchdogTimeoutVal	设置看门狗溢出值	上层用户
辅助函数		
SaveIPAddress	保存 IP 到注册表	上层用户
LoadIPAddress	载入 IP 到应用程序	上层用户
直接命令函数		
ReadDeviceChar	读取文件信息	底层用户
WriteDeviceChar	写入文件信息	底层用户

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\DAM3000E\INCLUDE\ DAM3000E.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 DAM3000E.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 输入行中加入指令 C:\Art\ DAM3000E \ DAM3000E.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 DAM3000E.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 DAM3000E.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 DAM3000E.h 和 DAM3000E.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。

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

C++ Builder:

要使用如下函数一个关键的问题是首先必须将我们提供的头文件(DAM3000E.H)写进您的源程序头部。如：#include “\Art\ DAM3000E \Include\ DAM3000E.h”

然后再将 DAM3000E.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 DAM3000E.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 DAM3000E.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 DAM3000E.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：“DAM3000E”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DAM3000E;  // 注意： 在此加入驱动程序接口单元 DAM3000E
```

LabView/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针,到其丰富的函数功能、数值分析、信号处理和设备驱动等功能,都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：

一、在 LabView 中打开 DAM3000E.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标

然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连续该接口模块即可顺利使用。

二、根据 LabView 语言本身的规定，接口单元图标以黑色的较粗的中竖线为中心，以左边的方格为数据输

入端，右边的方格为数据的输出端，如 ReadDeviceAD 接口单元，左边为设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“[U16]”为无符号 16 位短整型数组或缓冲区或指针，“[U32]”与“[U16]”同理，只是位数不一样。

第二节 设备对象管理函数原型说明

◆ 创建设备对象函数

Visual C++ & C++Builder：

```
HANDLE CreateDevice(
                        char        szIP[],
                        LONG        uPort,
                        LONG        iSendTimeout,
                        LONG        iRcvTimeout);
```

功能：该函数负责创建以太网设备对象，并返回其设备对象句柄。

参数：

szIP 为设备 IP 地址。当向同一个 Windows 系统中加入若干相同类型的以太网设备时，将以该设备的 IP 地址来确认和管理该设备。

Uport 为设备端口号。TCP/UDP 端口号(默认情况：TCP 端口为 5001，UDP 为 4000)

表 3-2 默认端口(TCP/UDP)

常量名	常量值	功能定义
DAM3000E_DEFAULT_TCP_PORT	5001	TCP 方式下的端口地址
DAM3000E_DEFAULT_UDP_PORT	4000	UDP 方式的端口地址

iSendTimeOut 为发送数据的超时范围。

ircvTimeOut 为接收数据的超时范围。

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 ( “192.168.2.80”, 502, 100, 100);   // 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
}
```

◆ 释放设备对象函数

Visual C++ & C++Builder：

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

功能： 释放设备对象。

参数： hDevice 设备对象句柄。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

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。

相关函数：CreateDevice

◆ 取得设备版本信息

Visual C++ & C++Builder：

```
BOOL GetDeviceVersion(
                        HANDLE      hDevice,
                        char        szVersion[]);
```

功能： 取得设备版本信息，形如“DAM-E3000 V6.10 2005.06.01”。

参数： hDevice 设备对象句柄。
szVersion 设备版本信息。
返回值： 若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, ReleaseDevice

◆ **获得设备的网络配置信息**

Visual C++ & C++Builder：

```
BOOL GetNetworkConfig(
                    HANDLE      hDevice,
                    PDEVICE_NET_INFO pNetInfo);
```

功能： 获得设备的网络配置信息，形如“192.168.2.80 255.255.255.0 192.168.2.1 00-0A-EB-57-5F-96”。
参数： hDevice 设备对象句柄。
pNetInfo DEVICE_NET_INFO 型结构体，保存了设备的网络配置信息。DEVICE_NET_INFO 结构体如下：

```
typedef struct _DEVICE_NET_INFO
{
    char    szIP[16];       // IP 地址，数组长度不小于 16
    char    SubnetMask[16]; // 子网掩码，数组长度不小于 16
    char    Gateway[16];    // 网关，数组长度不小于 16
    char    MAC[16];        // 网卡物理地址，数组长度不小于 16
    PLONG   pTCPPort;       // TCP 端口
    PLONG   pUDPPort;       // UDP 端口
}DEVICE_NET_INFO, *PDEVICE_NET_INFO;
```

返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, SetNetworkConfig, ReleaseDevice

◆ **设置设备的网络配置信息**

Visual C++ & C++Builder：

```
BOOL SetNetworkConfig(
                    HANDLE      hDevice,
                    PDEVICE_NET_INFO NetInfo);
```

功能： 修改设备的网络配置信息。
参数： hDevice 设备对象句柄。
NetInfo DEVICE_NET_INFO 型结构体变量，保存了网络配置信息。
DEVICE_NET_INFO 结构体见函数 GetNetworkConfig；
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, GetNetWorkConfig, ReleaseDevice
注： 在对网络参数进行设置后，必须重新上电后修改的值才能生效。

第三节 AD 模拟量输入函数原型说明

◆ **读取模拟量输入值**

Visual C++ & C++Builder：

```
BOOL ReadDeviceAD(
                    HANDLE      hDevice,
                    LONG        ADValue[ ],
                    LONG        lFirstChannel,
                    LONG        lLastChannel);
```

功能： 读取模拟量的当前输入值。
参数： hDevice 设备对象句柄。
ADValue 获得的输入值(原码)。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, GetModeAD, SetModeAD , ReleaseDevice

◆ **读取模拟量输入模式**

Visual C++ & C++Builder：

```
BOOL GetModeAD(
                    HANDLE      hDevice,
                    LONG        ADMode[ ],
                    LONG        lFirstChannel,
                    LONG        lLastChannel);
```

功能： 读取模拟量某通道的输入模式。

参数： hDevice 设备对象句柄。

ADMode 获得的输入模式。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, ReadDeviceAD, SetModeAD, ReleaseDevice

◆ **设置模拟量输入模式**

Visual C++ & C++Builder：

```
BOOL SetModeAD(
                    HANDLE      hDevice,
                    LONG        ADMode[ ],
                    LONG        lFirstChannel,
                    LONG        lLastChannel);
```

功能： 设置模拟量某个通道的输入模式。

参数： hDevice 设备对象句柄。

ADMode 要设置的输入模式。有电压（表 3-4），电流（表 3-5），热电阻（表 3-6）、热电偶（表 3-7）四大类型

表 3-4（电压类型）供 nADMode 参数使用

常量名	常量值	功能定义
DAM3000E_VOLT_N15_P15	0x01	-15～+15mV
DAM3000E_VOLT_N50_P50	0x02	-50～+50mV
DAM3000E_VOLT_N100_P100	0x03	-100～+100mV
DAM3000E_VOLT_N150_P150	0x04	-150～+150mV
DAM3000E_VOLT_N500_P500	0x05	-500～+500mV
DAM3000E_VOLT_N1_P1	0x06	-1～+1V
DAM3000E_VOLT_N25_P25	0x07	-2.5～+2.5V
DAM3000E_VOLT_N5_P5	0x08	-5～+5V
DAM3000E_VOLT_N10_P10	0x09	-10～+10V
DAM3000E_VOLT_N0_P5	0x0D	0～+5V
DAM3000E_VOLT_N0_P10	0x0E	0～+10V
DAM3000E_VOLT_N0_P25	0x0F	0～+2.5V

表 3-5（电流类型）供 nADMode 参数使用

常量名	常量值	功能定义
DAM3000E_CUR_N0_P10	0x00	0～10mA
DAM3000E_CUR_N20_P20	0x0A	-20～+20mA
DAM3000E_CUR_N0_P20	0x0B	0～20mA
DAM3000E_CUR_N4_P20	0x0C	4～20mA

表 3-6（热电阻类型）供 nADMode 参数使用

常量名	常量值	功能定义	备注
DAM3000E_RTD_PT100_385_N200_P850	0x20	-200℃～850℃	Pt100(385)热电阻
DAM3000E_RTD_PT100_385_N100_P100	0x21	-100℃～100℃	Pt100(385)热电阻
DAM3000E_RTD_PT100_385_N0_P100	0x22	0℃～100℃	Pt100(385)热电阻
DAM3000E_RTD_PT100_385_N0_P200	0x23	0℃～200℃	Pt100(385)热电阻
DAM3000E_RTD_PT100_385_N0_P600	0x24	0℃～600℃	Pt100(385)热电阻

DAM3000E_RTD_PT100_3916_N200_P850	0x25	-200℃～850℃	Pt100(3916)热电阻
DAM3000E_RTD_PT100_3916_N100_P100	0x26	-100℃～100℃	Pt100(3916)热电阻
DAM3000E_RTD_PT100_3916_N0_P100	0x27	0℃～100℃	Pt100(3916)热电阻
DAM3000E_RTD_PT100_3916_N0_P200	0x28	0℃～200℃	Pt100(3916)热电阻
DAM3000E_RTD_PT100_3916_N0_P600	0x29	0℃～600℃	Pt100(3916)热电阻
DAM3000E_RTD_PT1000	0x30	-200℃～850℃	Pt1000 热电阻
DAM3000E_RTD_CU50	0x40	-50℃～150℃	Cu50 热电阻
DAM3000E_RTD_CU100	0x41	-50℃～150℃	Cu100 热电阻
DAM3000E_RTD_BA1	0x42	-200℃～650℃	BA1 热电阻
DAM3000E_RTD_BA2	0x43	-200℃～650℃	BA2 热电阻
DAM3000E_RTD_G53	0x44	-50℃～150℃	G53 热电阻
DAM3000E_RTD_Ni50	0x45	100℃	Ni50 热电阻
DAM3000E_RTD_Ni508	0x46	0℃～100℃	Ni508 热电阻
DAM3000E_RTD_Ni1000	0x47	-60℃～160℃	Ni1000 热电阻

表 3-7（热电偶类型）供 nADMode 参数使用

常量名	常量值	功能定义	备注
DAM3000E_THERMOCOUPLE_J	0x10	0～1200℃	J 型热电偶
DAM3000E_THERMOCOUPLE_K	0x11	0～1300℃	K 型热电偶
DAM3000E_THERMOCOUPLE_T	0x12	0～400℃	T 型热电偶
DAM3000E_THERMOCOUPLE_E	0x13	0～1000℃	E 型热电偶
DAM3000E_THERMOCOUPLE_R	0x14	500～1700℃	R 型热电偶
DAM3000E_THERMOCOUPLE_S	0x15	500～1768℃	S 型热电偶
DAM3000E_THERMOCOUPLE_B	0x16	500～1800℃	B 型热电偶
DAM3000E_THERMOCOUPLE_N	0x17	0～1300℃	N 型热电偶
DAM3000E_THERMOCOUPLE_C	0x18	0～2090℃	C 型热电偶

与电压和温度的换算关系如下：

Value = (Lsb /0xFFFF) * (量程上限 – 量程下限) +量程下限

(注:Lsb 为采集的原码值， Value 为转换所得的电压值或温度值.)

以-10～+10V 为例：

Lsb = 0x8000

Value = (Lsb / 0xFFFF) × (10 – (-10)) + (-10) = (0x8000 /0xFFFF) * 20 - 10 = 0.0001V

以 0～1200℃为例：

Lsb = 0x8000

Value = (Lsb / 0xFFFF) × (1200 – 0)) + 0 = (0x8000 / 0xFFFF) *1200 + 0 = 600.0℃

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, ReadDeviceAD, GetModeAD, ReleaseDevice

◆ 设置下限报警值

Visual C++ & C++Builder：

```
BOOL SetLowLimitVal(
                    HANDLE    hDevice,
                    LONG      LowVal[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 设置下限报警值。

参数： hDevice 设备对象句柄。

LowVal 存放下限值的数组。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, GetLowLimitVal， ReleaseDevice

◆ **获得报警下限值**

Visual C++ & C++Builder

```
BOOL GetLowLimitVal(
                    HANDLE    hDevice,
                    LONG      LowLimit[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 获得报警下限值。

参数： hDevice 设备对象句柄。

LowVal 存放下限值的数组。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, SetLowLimitVal, ReleaseDevice

◆ **设置上限报警值**

Visual C++ & C++Builder

```
BOOL SetHighLimitVal(
                    HANDLE    hDevice,
                    LONG      HighVal[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 设置上限报警值。

参数： hDevice 设备对象句柄。

LowVal 存放上限值的数组。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, GetHighLimitVal, ReleaseDevice

◆ **获得报警上限值**

Visual C++ & C++Builder

```
BOOL GetHighLimitVal(
                    HANDLE    hDevice,
                    LONG      HighLimit[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 获得报警上限值。

参数： hDevice 设备对象句柄。

LowVal 存放上限值的数组。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, SetHighLimitVal, ReleaseDevice

◆ **获得报警输出值**

Visual C++ & C++Builder：

```
BOOL GetAlarmPulse(
                    HANDLE    hDevice,
                    PLONG     AlarmValue);
```

功能： 获得报警输出值。

参数： hDevice 设备对象句柄。

AlarmValue 指向获得的报警输出值。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, SetAlarmPulse, ReleaseDevice

◆ **设置报警输出值**

Visual C++ & C++Builder：

```
BOOL SetAlarmPulse(
                        HANDLE      hDevice,
                        LONG        AlarmValue);
```

功能： 设置报警输出值。

参数： hDevice 设备对象句柄。

AlarmValue 报警输出值。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, GetAlarmPulse, ReleaseDevice

◆ **获得报警状态**

Visual C++ & C++Builder：

```
BOOL GetAlarmSts(
                        HANDLE      hDevice,
                        LONG        Status[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 获得报警状态。

参数： hDevice 设备对象句柄。

Status 报警状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, ReleaseDevice

第四节 DA 模拟量输出函数原型说明

注:待完善

第五节 DI 数字量输入函数原型说明

◆ **获得数字量输入值**

Visual C++ & C++Builder：

```
BOOL GetDeviceDI(
                        HANDLE      hDevice,
                        BYTE        DIStatus[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 获得数字量输入值。

参数： hDevice 设备对象句柄。

DIStatus 数字量输入状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, ReleaseDevice

◆ **获得计数值**

Visual C++ & C++Builder：

```
BOOL GetDeviceCNT(
                        HANDLE      hDevice,
                        LONG        CurrentVal[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 获得数字量输入值。

参数： hDevice 设备对象句柄。

CurrentVal 当前计数值。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, SetDeviceCNT, ReleaseDevice

◆ **设置计数初值**

Visual C++ & C++Builder：

```
BOOL SetDeviceCNT(
                        HANDLE      hDevice,
                        LONG        InitVal[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 设置计数初值。
参数： hDevice 设备对象句柄。
InitVal 设置的初始值。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, GetDeviceCNT, ReleaseDevice

◆ **获得计数使能状态**

Visual C++ & C++Builder：

```
BOOL GetEnableStsCNT (
                        HANDLE      hDevice,
                        BYTE        CNTStatus[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 获得计数使能状态。
参数： hDevice 设备对象句柄。
CNTStatus 指向计数状态的指针。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, ReleaseDevice

◆ **获得计数方式**

Visual C++ & C++Builder：

```
BOOL GetModeCNT (
                        HANDLE      hDevice,
                        LONG        CNTMode
                        LONG        nChannel)
```

功能： 获得计数方式。
参数： hDevice 设备对象句柄。
CNTMode 通道的计数方式。
nChannel 通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, SetModeCNT, ReleaseDevice

◆ **设置计数方式**

Visual C++ & C++Builder：

```
BOOL SetModeCNT(
                        HANDLE      hDevice,
                        LONG        CNTMode,
                        LONG        nChannel)
```

功能： 设置计数方式。
参数： hDevice 设备对象句柄。
CNTMode 通道的计数方式。
nChannel 通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, GetModeCNT, ReleaseDevice

◆ **开始计数**

Visual C++ & C++Builder：

```
BOOL StartDeviceCNT(
                        HANDLE      hDevice,
                        LONG        nChannel);
```

功能： 开始计数。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, StopDeviceCNT, ReleaseDevice

◆ **停止计数**

Visual C++ & C++Builder：

```
BOOL StopDeviceCNT(
                        HANDLE      hDevice,
                        LONG        nChannel );
```

功能： 停止计数。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, StartDeviceCNT, ReleaseDevice

◆ **启动锁存**

Visual C++ & C++Builder：

```
BOOL StartLatch(
                        HANDLE      hDevice,
                        LONG        nChannel);
```

功能： 启动锁存。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, StopLatch, ReleaseDevice

◆ **停止锁存**

Visual C++ & C++Builder：

```
BOOL StopLatch(
                        HANDLE      hDevice,
                        LONG        nChannel);
```

功能： 停止锁存。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, StartLatch, ReleaseDevice

◆ **获得锁存使能状态**

Visual C++ & C++Builder：

```
BOOL GetLatchEnableSts(
                        HANDLE      hDevice,
                        BYTE        LatchStatus[ ],
                        LONG        lFirstChannel,
                        LONG        lLastChannel);
```

功能： 获得锁存使能状态。

参数： hDevice 设备对象句柄。

LatchStatus 锁存使能状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, ReleaseDevice

◆ **获得下降沿锁存状态**

Visual C++ & C++Builder：

```
BOOL  GetLowLatchSts(
                        HANDLE    hDevice,
                        BYTE      LowLatchSts[],
                        LONG      lFirstChannel,
                        LONG      lLastChannel);
```

功能： 获得下降沿锁存状态。

参数： hDevice 设备对象句柄。

LowLatchSts 锁存状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, GetHiLatchSts, ReleaseDevice

◆ **获得上升沿锁存状态**

Visual C++ & C++Builder

```
BOOL  GetHiLatchSts(
                        HANDLE    hDevice,
                        BYTE      HiLatchSts[],
                        LONG      lFirstChannel,
                        LONG      lLastChannel);
```

功能： 获得上升沿锁存状态。

参数： hDevice 设备对象句柄。

HiLatchSts 锁存状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, GetLowLatchSts, ReleaseDevice

第六节 DO 数字量输出函数原型说明

◆ **获得当前数字量输出**

Visual C++ & C++Builder：

```
BOOL GetDeviceDO(
                        HANDLE    hDevice,
                        BYTE      DOSts[ ],
                        LONG      lFirstChannel,
                        LONG      lLastChannel);
```

功能： 获得当前数字量输出。

参数： hDevice 设备对象句柄。

DOSts 数字量输出状态。

lFirstChannel 首通道号。

lLastChannel 末通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetDeviceDO, ReleaseDevice

◆ **设置 DO 输出值**

Visual C++ & C++Builder：

```
BOOL SetDeviceDO(
                        HANDLE    hDevice,
                        BYTE      DOSts[ ],
                        LONG      lFirstChannel,
                        LONG      lLastChannel);
```

功能： 设置 DO 输出值。
参数： hDevice 设备对象句柄。
DOSts 设置数字量输出的状态。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, GetDeviceDO, ReleaseDevice

◆ **获得脉冲输出宽度**

Visual C++ & C++Builder：

```
BOOL GetDevicePulse(
                    HANDLE    hDevice,
                    LONG      LowWidth[ ],
                    LONG      HighWidth[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 获得脉冲输出宽度。
参数： hDevice 设备对象句柄。
LowWidth 低电平宽度。
HighWidth, 高电平宽度。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, SetDevicePulse , ReleaseDevice

◆ **设置脉冲输出宽度**

Visual C++ & C++Builder：

```
BOOL SetDevicePulse(
                    HANDLE    hDevice,
                    LONG      LowWidth[ ],
                    LONG      HighWidth[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 设置脉冲输出宽度。
参数： hDevice 设备对象句柄。
LowWidth 低电平宽度。
HighWidth, 高电平宽度。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, GetDevicePulse, ReleaseDevice

◆ **获得脉冲使能状态**

Visual C++ & C++Builder：

```
BOOL GetPulseEnableSts(
                    HANDLE    hDevice,
                    BYTE      PulseStatus[ ],
                    LONG      lFirstChannel,
                    LONG      lLastChannel);
```

功能： 获得脉冲使能状态。
参数： hDevice 设备对象句柄。
PulseStatus 各个通道脉冲使能的状态。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, SetDevicePulse, ReleaseDevice

◆ **开始数字脉冲输出**

Visual C++ & C++Builder：

```
BOOL StartDevicePulse(
                        HANDLE      hDevice,
                        LONG        nChannel);
```

功能： 开始数字脉冲输出。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, StopDevicePulse, ReleaseDevice

◆ **停止数字脉冲输出**

Visual C++ & C++Builder：

```
BOOL StopDevicePulse(
                        HANDLE      hDevice,
                        LONG        nChannel);
```

功能： 停止数字脉冲输出。

参数： hDevice 设备对象句柄。

nChannel 通道号。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, StartDevicePulse, ReleaseDevice

◆ **设置 DO 安全值**

Visual C++ & C++Builder：

```
BOOL SetSafeValueDO(
                        HANDLE      hDevice,
                        WORD        SafeValue);
```

功能： 设置 DO 安全值。

参数： hDevice 设备对象句柄。

SafeValue 安全值。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, GetSafeValueDO, ReleaseDevice

◆ **获得 DO 安全值**

Visual C++ & C++Builder：

```
BOOL GetSafeValueDO(
                        HANDLE      hDevice,
                        WORD        SafeValue);
```

功能： 获得 DO 安全值。

参数： hDevice 设备对象句柄。

SafeValue 安全值。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetSafeValueDO, ReleaseDevice

◆ **设置 DO 上电值**

Visual C++ & C++Builder：

```
BOOL SetPowerOnValueDO(
                        HANDLE      hDevice,
                        WORD        PowerOnValue);
```

功能： 设置 DO 上电值。

参数： hDevice 设备对象句柄。

PowerOnValue 上电值。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, GetPowerOnValueDO, ReleaseDevice

◆ **获取 DO 上电初始值**

Visual C++ & C++Builder

```
BOOL GetPowerOnValueDO(
                        HANDLE      hDevice,
                        WORD        PowerOnValue);
```

功能： 获取 DO 上电初始值。
参数： hDevice 设备对象句柄。
PowerOnValue 上电值。。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, SetPowerOnValueDO, ReleaseDevice

第七节 看门狗函数原型说明

◆ **Host Is OK**

Visual C++ & C++Builder:

```
BOOL HostIsOK(
                        HANDLE      hDevice);
```

功能： Host Is OK，主机正常命令。
参数： hDevice 设备对象句柄。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, ReleaseDevice

◆ **使能看门狗**

Visual C++ & C++Builder:

```
BOOL EnableWatchdog(
                        HANDLE      hDevice);
```

功能： 使能看门狗。
参数： hDevice 设备对象句柄。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, GetWatchdogSts, CloseWatchdog, ReleaseDevice

◆ **关闭看门狗**

Visual C++ & C++Builder:

```
BOOL CloseWatchdog(
                        HANDLE      hDevice);
```

功能： 关闭看门狗。
参数： hDevice 设备对象句柄。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, GetWatchdogSts, ReleaseDevice

◆ **获得看门狗状态**

Visual C++ & C++Builder:

```
BOOL GetWatchdogSts(
                        HANDLE      hDevice,
                        PLONG       DogStatus);
```

功能： 获得看门狗状态。
参数： hDevice 设备对象句柄。
DogStatus 看门狗状态。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, EnableWatchdog, CloseWatchdog, ReleaseDevice

◆ **复位看门狗**

Visual C++ & C++Builder:

```
BOOL ResetWatchdog(
                        HANDLE      hDevice);
```

功能： 复位看门狗。
参数： hDevice 设备对象句柄。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice, ReleaseDevice

◆ **设置看门狗溢出值**

Visual C++ & C++Builder:

```
BOOL SetWatchdogTimeoutVal(
                        HANDLE      hDevice,
                        LONG        TimeOutVal);
```

功能： 设置看门狗溢出值。

参数： hDevice 设备对象句柄。

TimeOutVal 看门狗溢出时间。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice, ReleaseDevice

第八节 辅助函数原型说明

◆ **保存 IP 到注册表**

Visual C++ & C++Builder:

```
BOOL   SaveIPAddress(char szIP[]);   // 保存 IP 到注册表
```

功能： 保存 IP 到注册表。

参数： szIP 保存的 IP 地址。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： LoadIPAddress

◆ **载入 IP 到应用程序**

Visual C++ & C++Builder:

```
BOOL   LoadIPAddress(char szIP[]); // 载入 IP 到应用程序
```

功能： 保存 IP 到注册表。

参数： szIP 读取的 IP 地址。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： SaveIPAddress

第四章 共用函数介绍

第一节 公用接口函数列表

表 4-1 公用接口函数列表

函数名	函数功能	功能定义
WriteDeviceChar	直接写设备	底层用户
ReadDeviceChar	直接读设备	底层用户

第二节 公用接口函数原型说明

◆ **直接写设备**

Visual C++ & C++Builder：

```
int    WriteDeviceChar(
                    HANDLE hDevice,
                    char* writeBuffer,
                    LONG length);
```

功能： 直接写设备。

参数： hDevice 设备对象句柄。

writeBuffer 写入设备的命令，形如测试连接状态“A5 03 0A”。

length 写入命令长度。

返回值：若成功，则返回 0，否则返回-1。

相关函数：ReadDeviceChar

◆ **直接读设备**

Visual C++ & C++Builder：

```
int   ReadDeviceChar(
                    HANDLE hDevice,
                    char* readBuffer,
                    LONG length);
```

功能： 直接读设备。

参数： hDevice 设备对象句柄。

readBuffer 保存返回数据，形如测试连接状态“5A03 EA”。

length 返回数据的长度。

返回值：若成功，则返回 0，否则返回-1。

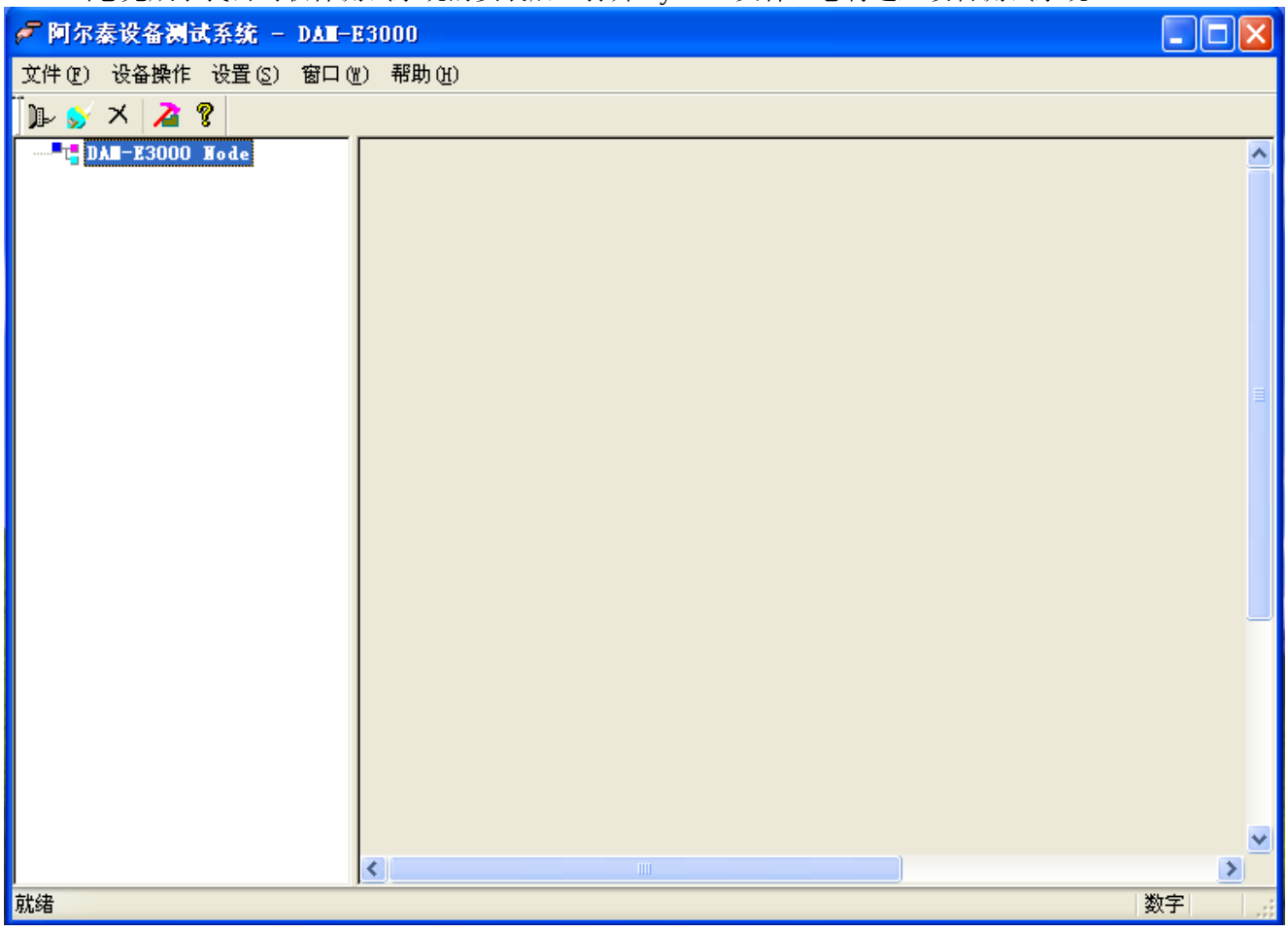
相关函数：WriteDeviceChar

Visual C++ 程序举例:

```
HANDLE hDevice;
Char readBuf[100];
hDevice   = CreateDevice(“192.168.2.80”, 5001, 100, 100);
WriteDeviceChar(hDevice, “A5 03 0A”, 3);
ReadDeviceChar(hDevice, readBuf, 3);
```

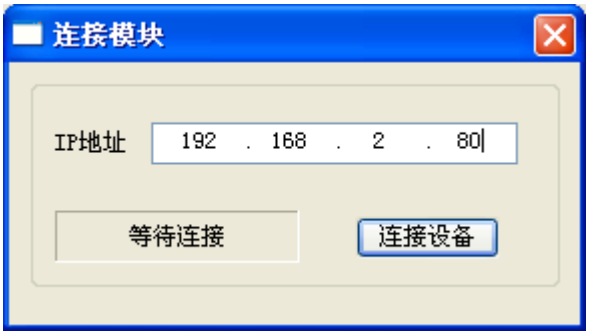
第五章 DAM-E3000 设备软件测试系统的介绍

当您完成了我公司软件测试系统的安装后，打开 Sys.exe 文件，您将进入设备测试系统。



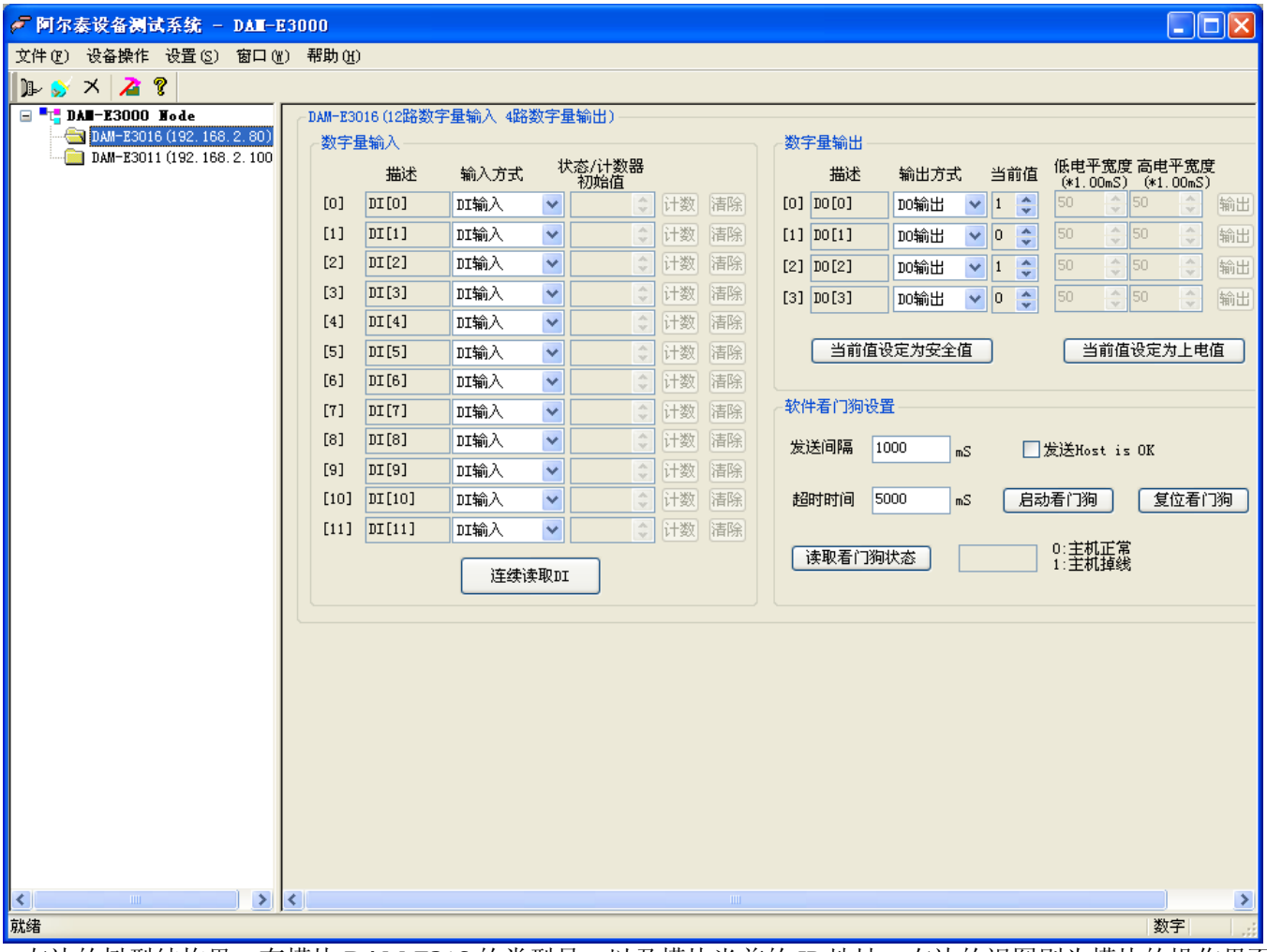
第一节 搜索模块

在您对模块进行操作前，您必须先进行搜索，并获得模块的句柄，这样才能对模块进行操作。在搜索时，模块 IP 为“192.168.2.80”，ModBus TCP/IP 方式下默认端口号为“502”。



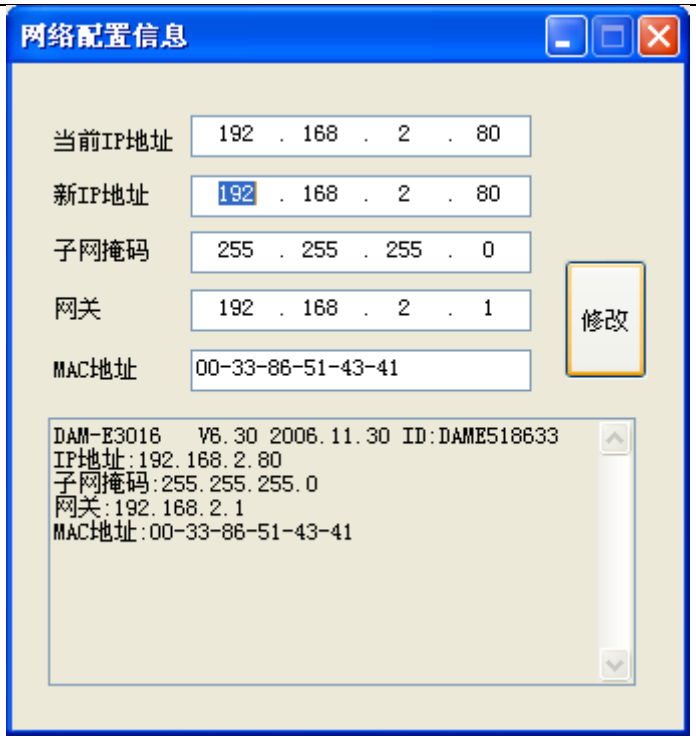
第二节 修改模块信息

当搜索成功后，将弹出相应的模块界面及信息。下面以 DAM-E316 为例：



左边的树型结构里，有模块 DAM-E316 的类型号，以及模块当前的 IP 地址。右边的视图则为模块的操作界面。

如果你想改变模块的 IP 地址或其它信息时，您只需要双击树型结构中相应的模块，便会弹出下图中的对话框：



在修改了信息后，需要重新为模块上电后，新 IP 地址才能生效。

注：各个功能模块的详细操作说明，请见简易说明书。

第六章 上层用户函数接口应用实例

第一节 怎样使用 ReadDeviceAD 函数直接取得 AD 数据

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 AD 简易程序演示

[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 演示]**

然后，您着重参考以下函数:

BOOL SetInputModeAD(); // 设置模拟量输入模式

首先，使用 SetInputModeAD 设置 DA 模式（电压类型，电流类型，热电阻类型，热电偶类型）,其中每个类型又分各个型号，详细请参阅附录 B；然后根据选择的模式进行操作。

BOOL ReadDeviceAD(); // 读取模拟量输入

BOOL GetInputModeAD(); // 获得模拟量输入模式

AD 采样的要求是:首先设置模拟量输入模式，然后便可读取所需的数据。

第二节 怎样使用 WriteDeviceDA 函数实现 DA 的波形输出

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 DA 简易程序演示

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DA 演示]**

BOOL SetOutputModeDA(); // 设置模拟量输出模式

BOOL SetDataTypeDA(); // 设置模拟量输出数据格式

BOOL WriteDeviceDA(); // 写模拟量输出

第三节 怎样使用 SetDeviceDO 函数进行更便捷的数字开关量输出操作

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 DO 简易程序演示

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DIO 演示]**

BOOL SetDevModeDI(); // 设置 DI 输入模式

首先，使用 SetDevModeDI 设置 DI 输入模式（常规数字量输入：0x00，计数器模式：0x01，锁存模式：0x02）；然后根据选择的模式进行操作。

1. 常规数字量输入（0x00）

GetDeviceDI() // 获取 DI 输入值

2. 计数器模式 (0x01)
   SetDeviceCNT()
   StartDeviceCNT()
   StopDeviceCNT()
   GetDeviceCNT()
3. 锁存模式 (0x02)
   GetLatchStatus()
   ClearLatchStatus()

当输入模式为计数器模式或锁存模式时，又分为上升沿和下降沿两种类型。

第四节 怎样使用 GetDeviceDI 函数进行更便捷的数字开关量输入操作

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 DI 简易程序演示

[程序]丨**[阿尔泰测控演示系统]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[DIO 演示]**

BOOL SetDevModeDO ();　　//设置 DO 输出模式

首先，使用 SetDevModeDO 设置 DO 输出模式（常规数字量输出：0x00，脉冲输出方式：0x01）；然后根据选择的模式输出。

1. 常规数字量输出（0x00）
   SetDeviceDO(); // 设置 DO 当前输出值
   GetDeviceDO(); // 读取 DO 输出值
2. 脉冲输出方式(0x01)
   GetDeviceDO(); // 读取 DO 输出值
   StartDevicePulse(); // 开始数字脉冲输出
   StopDevicePulse(); // 停止数字脉冲输出

第七章 底层用户函数接口应用实例

对函数 WriteDeviceChar 和 ReadDeviceChar 进行操作，要求用户对设备的协议及各种命令字都非常熟悉。对硬件参数也需要的了解。

第一节 怎样使用 WriteDeviceChar 函数实现直接写设备

WriteDeviceChar(HANDLE hDevice, char* writeBuffer, LONG length);　// 直接写设备

函数 WriteDeviceChar 是实现对设备的直接写操作。它与函数 ReadDeviceChar 是成对出现的函数。一般 ReadDeviceChar 紧跟在 WriteDeviceChar 之后。

第二节 怎样使用 ReadDeviceChar 函数实现直接读设备

ReadDeviceChar(HANDLE hDevice, char* readBuffer, LONG length);　// 直接读设备

函数 ReadDeviceChar 是实现对设备的直接读操作。它与函数 WriteDeviceChar 是成对出现的函数。一般 ReadDeviceChar 紧跟在 WriteDeviceChar 之后。

Visual C++ 程序举例:

```
BYTE szSend[50] = {0xa5, 0x04, 0x81}; // command line
szSend[3] = (BYTE)0xff; // DA Channel Number
BYTE szRecv[50];
DAM3000E_WriteDeviceChar(m_pModObj->m_hDevice, (char*)szSend, 0x04); // Send request
DAM3000E_ReadDeviceChar(m_pModObj->m_hDevice, (char*)szRecv, 12);
if( (szRecv[0] != 0x5A) || (szRecv[2]  != 0xEA) )
    return FALSE;
    m_nADMode = szRecv[4];
```

附录 A LabView/CVI 图形语言专述

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。

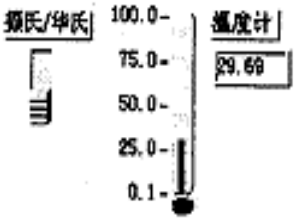


图 1 温度计的前面板

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：

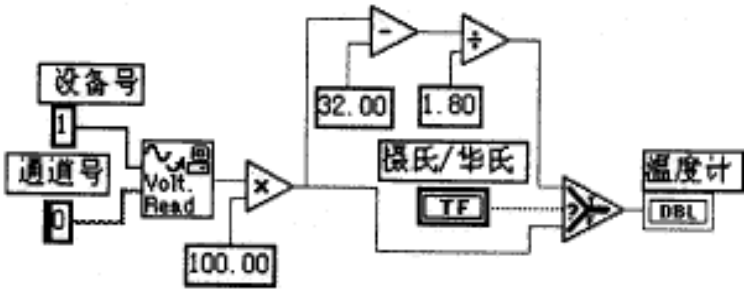


图 2 温度计的框图程序

(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。

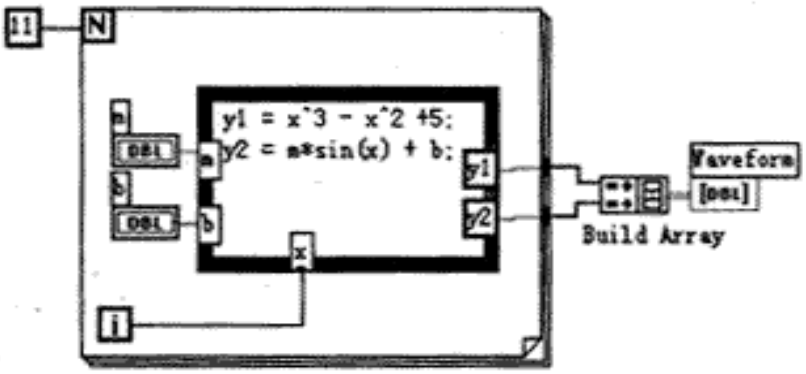


图 3 公式结运算例子

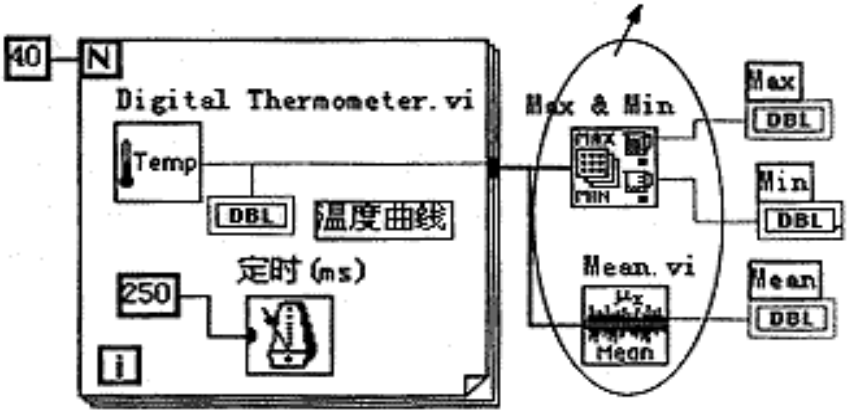


图 4 使用功能子模块进行温度曲线分析

(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。

7 (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。