DAM3000M 卡

WIN2000/NT/98/95 驱动程序使用说明书

阿尔泰科技发展有限公司
产品研发部修订

目 录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 DAMxxxx_ 则被省略。如 DAM3000M_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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

第二章 使用纲要

第一节、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，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 InitDeviceAD 、 ReadDeviceProAD Npt 、 SetDeviceDO 等。而底层用户函数如 WriteRegisterULong 、ReadRegisterULong、WritePortByte、ReadPortByte……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，您基本上不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。

第二节、如何管理 DAM-3000 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。在使用设备前，您还需要使用 InitDevice 初始化串口。然后便可以将此句柄作为参数传递给其他函数，如 GetDeviceInfo 可以使用 hDevice 句柄以获取设备设备信息，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。

第三节、如何取得 AD 数据

当您有了 hDevice 设备对象句柄后，便可用 SetSensorType 函数初始化 AD 部件，设置热传感器(热电阻、热电偶)的类型。您只需要对 ReadDeviceAD 中的 FirstChannel、LastChannel 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，停止读取 AD 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。

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。

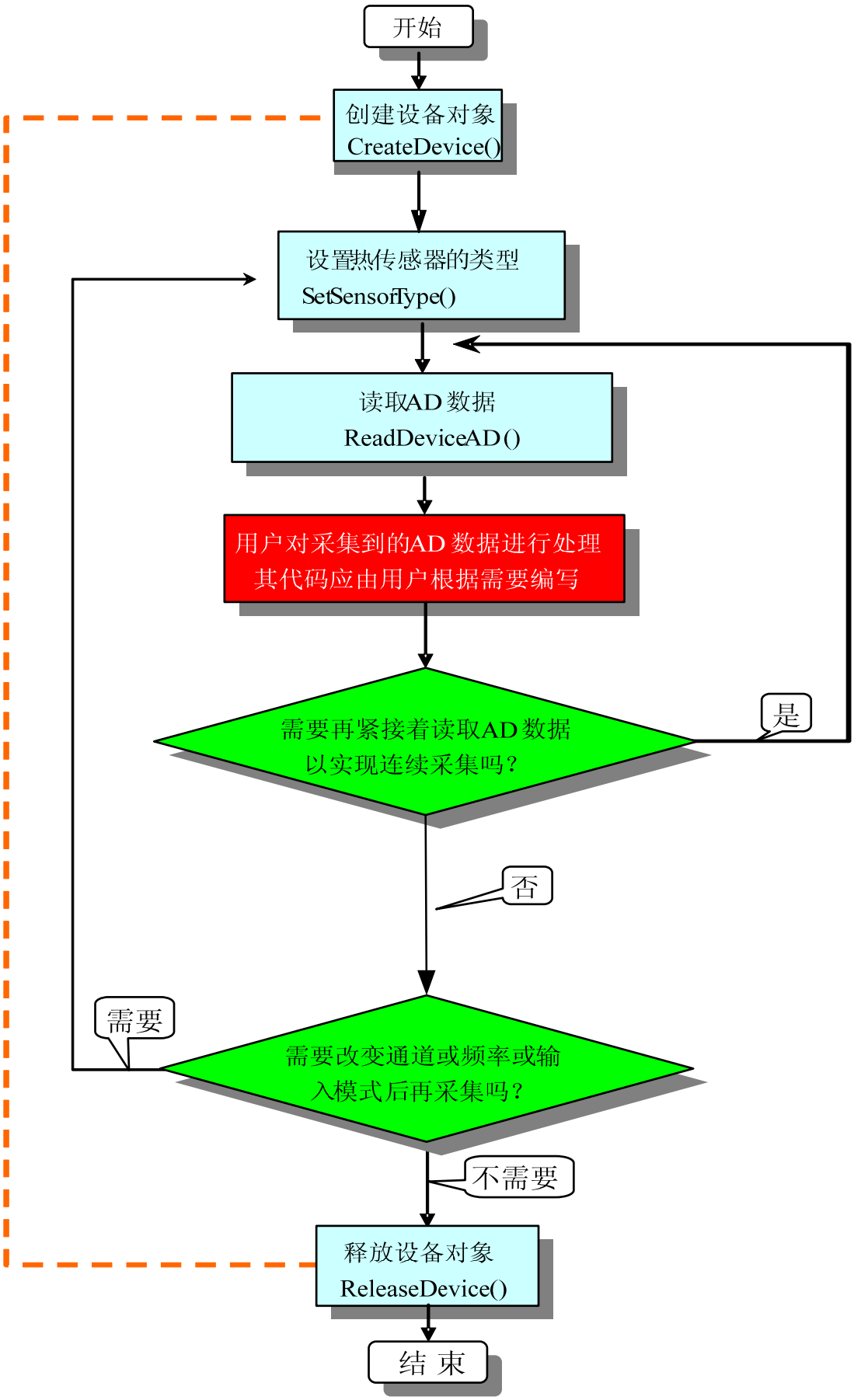

开始
创建设备对象
CreateDevice()
设置热传感器的类型
SetSensorType()
读取AD数据
ReadDeviceAD()
用户对采集到的AD数据进行处理
其代码应由用户根据需要编写
需要再紧接着读取AD数据
以实现连续采集吗？
是
否
需要
需要改变通道或频率或输
入模式后再采集吗？
不需要
释放设备对象
ReleaseDevice()
结 束

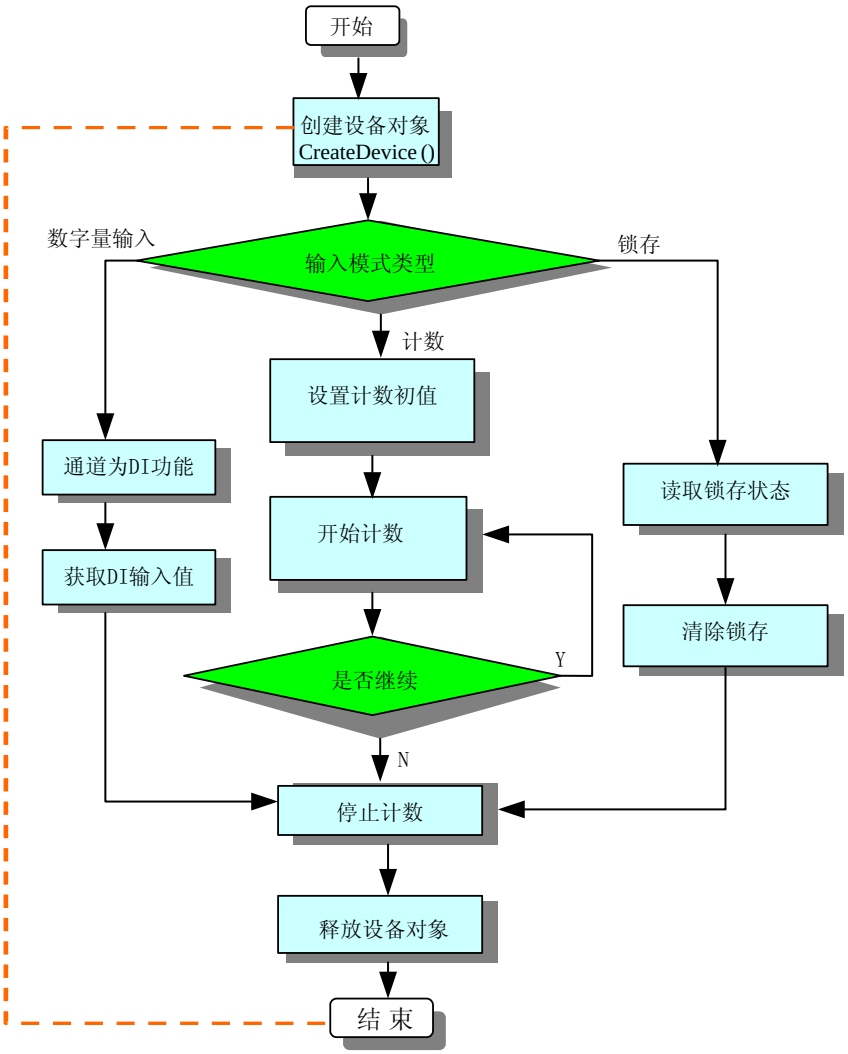
图 2.1.1 AD 采集过程

第四节、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，用 SetSlopeDA 函数设置 DA 的斜率，然后反复调用 WriteDeviceDA 函数输出每一个 DA 数据。

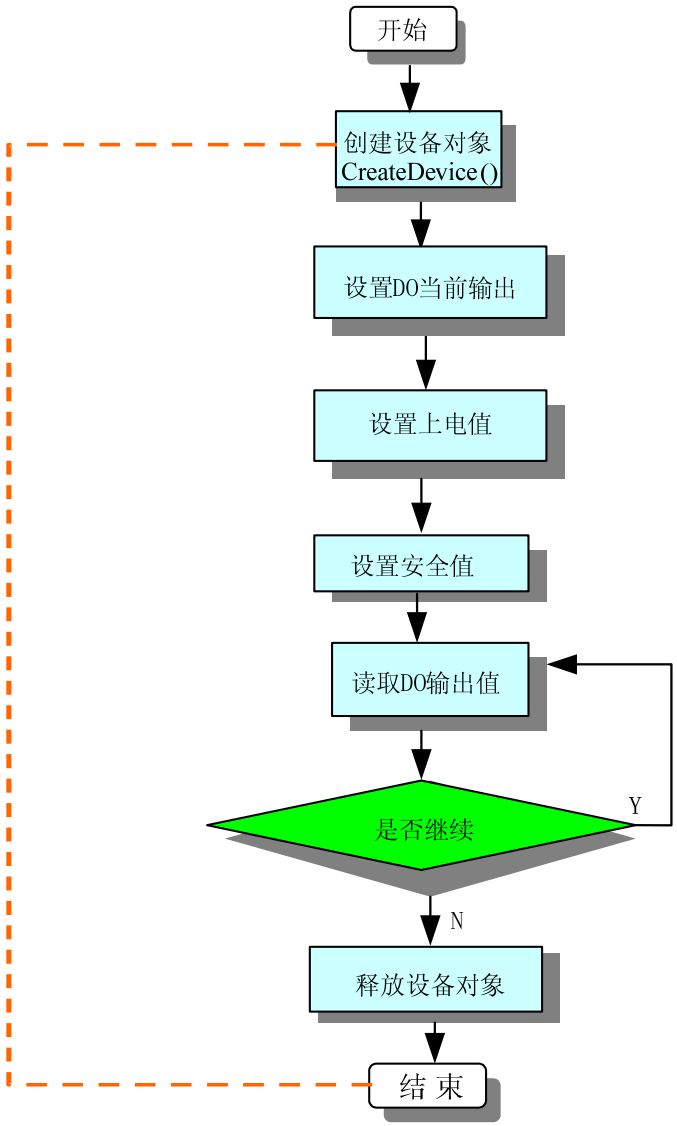
第五节、如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 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参数 pDIPara 中的成员变量 DI0-DI15 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 DAM_3000_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

第六节、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。

第七节、哪些函数对您不是必须的

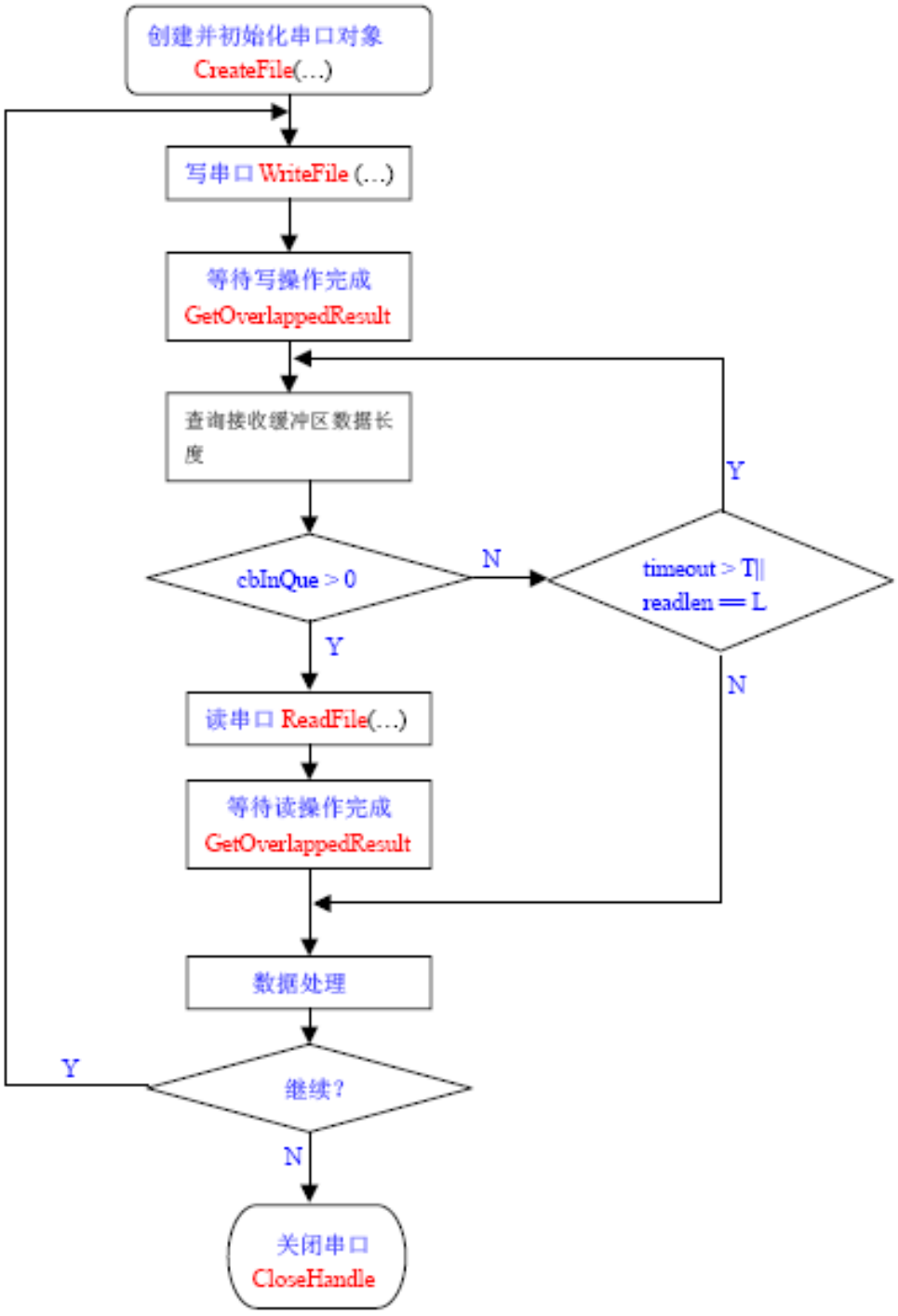
公共函数如 CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么 GetDeviceAddr，WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而 WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对DAM用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第八节、驱动程序功能概述

一、连接方式

DAM-3000 系列板卡提供了一种连接方式，即异步 I/O 模式的串口通信。

二、异步 I/O 模式的串口通信流程图



说明：上 面流程图中的决策框中的 cbInQue 是通过 ClearCommError 函数取得的接收缓冲区中的数据队列的长度。大于 0 表示 ，缓冲区中有数据，可以用 ReadFile 函数进行数据的读取；timeout > T || readlen == L 表示超时或者读取的数据长度已经 等于指定的长度，只要满足其中一个条 件，则完成一次指定数据长度和超时 范围的读取操作。

三、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。

四、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。

五、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。

六、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。

七、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。

八、多语言编程环境

本系统提供 Visual C++, 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。

九、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、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。

十一、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。

十二、**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。

十三、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）；

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；

第九节、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
DAM3000.VxD	动态虚拟设备驱动程序库	Window95/98	WinDir\System
DAM3000.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
DAM3000.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
DAM3000.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…
DAM3000.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
DAM3000.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
DAM3000.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
DAM3000.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

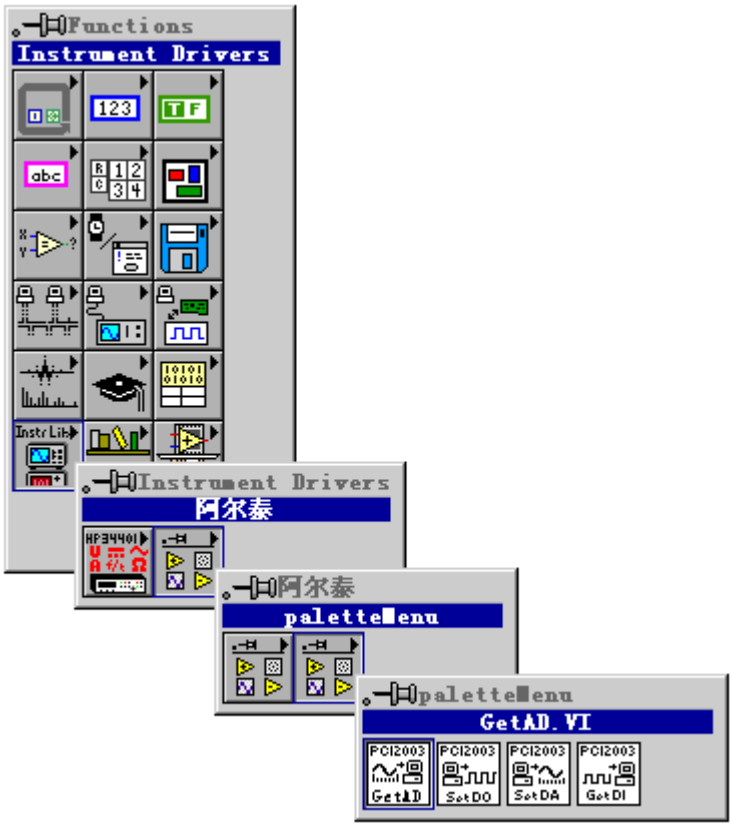
第三章 设备操作函数接口介绍

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 SetSensorType）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 Visual Basic 语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。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 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“DAM3000M_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建设备对象	上层及底层用户
InitDevice	初始与模块之间的通信参数	上层及底层用户
ReleaseDevice	释放设备对象	上层及底层用户
② 模块信息取得/修改函数		
GetDeviceInfo	读取模块信息(类型、地址、波特率、校验)	上层及底层用户
SetDeviceInfo	修改模块信息(地址、波特率、校验)	上层及底层用户
GetDevLastError	获得最后一个错误	上层及底层用户
③ AD 数据读取函数		
ReadDeviceAD	读取 AD 模拟量输入	上层及底层用户
GetModeAD	获得模拟量输入模式	上层及底层用户
SetModeAD	设置 AD 输入模式	上层及底层用户
GetGroundingAD	获得通道接地模式(只对可软件配置单/双端输入模块有效)	上层及底层用户
SetGroundingAD	设置通道接地模式(只对可软件配置单/双端输入模块有效)	上层及底层用户
GetLowLimitVal	获得模拟量输入报警下限值	上层及底层用户
GetHighLimitVal	获得模拟量输入报警上限值	上层及底层用户
SetLowLimitVal	设置下限报警值	上层及底层用户
SetHighLimitVal	设置上限报警值	上层及底层用户
GetAlarmPulse	获得报警的电平	上层及底层用户
SetAlarmPulse	设置模拟量输入报警电平	上层及底层用户
GetAlarmSts	获得报警状态	上层及底层用户
④ DA 数据读取函数		
GetDeviceDAVal	回读 DA 输出值	上层及底层用户
WriteDeviceDA	设定单通道 DA	上层及底层用户
GetOutPutRangeDA	读取模拟量输出量程	上层及底层用户
SetOutPutRangeDA	设置模拟量输出量程	上层及底层用户
GetPowerOnValueDA	获得 DA 上电值	上层及底层用户
SetPowerOnValueDA	设置 DA 上电值	上层及底层用户
GetSafeValueDA	获得 DA 安全值	上层及底层用户
SetSafeValueDA	设置 DA 安全值	上层及底层用户
GetSlopeDA	读模拟量输出斜率	上层及底层用户
SetSlopeDA	修改模拟量输出斜率	上层及底层用户
⑤ DI 输入输出操作函数		
GetModeDI	读取数字量输入的工作模式	上层及底层用户
SetModeDI	设置数字量输入的工作模式	上层及底层用户
GetDeviceDI	读取开关量输入	上层及底层用户
StartDeviceDI	启动 DI 计数	上层及底层用户
StopDeviceDI	停止 DI 计数	上层及底层用户
GetCNTDI	读取 DI 计数器值	上层及底层用户
SetCNTDI	设置 DI 计数器初始值	上层及底层用户
StartLatch	启动锁存	上层及底层用户
StopLatch	停止锁存	上层及底层用户
GetLatchStatus	读锁存状态	上层及底层用户
ClearCNTVal	清除计数值	上层及底层用户
ClearLatchStatus	清除锁存状态	上层及底层用户
⑥ DO 数字量输出函数		
GetDeviceDO	回读开关量输出	上层及底层用户

SetDeviceDO	设置 DO 开关量输出值	上层及底层用户
GetPowerOnValueDO	获取 DO 上电初始值	上层及底层用户
SetPowerOnValueDO	设置 DO 上电初始值	上层及底层用户
GetSafeValueDO	读 DO 安全值	上层及底层用户
SetSafeValueDO	设置安全值	上层及底层用户
⑦ 计数器		
SetCounterMode	对各个计数器进行参数设置	上层及底层用户
InitCounterAlarm	初始化报警的工作模式	上层及底层用户
SetCounterAlarmMode	设置计数器报警方式	上层及底层用户
GetCounterSts	获得计数器设备硬件参数状态	上层及底层用户
StartCounter	启动计数器计数	上层及底层用户
StopCounter	停止计数器计数	上层及底层用户
GetCounterCurVal	取得计数器当前值	上层及底层用户
GetFreqCurVal	取得频率器当前值	上层及底层用户
ResetCounter	计数器复位	上层及底层用户
InitCounterFilter	初始化滤波	上层及底层用户
EnableFilter	使能滤波状态	上层及底层用户
GetCounterAlarmSts	获得 DO 及报警状态	上层及底层用户
SetCounterDO	设置 DO	上层及底层用户
ClearAlarmSts	清报警方式 1 报警输出	上层及底层用户
GetLEDCounterCH	取得计数器 LED 显示通道	上层及底层用户
SetLEDCounterCH	设置计数器 LED 显示通道	上层及底层用户
⑧ 电量模块		
GetEnergyVal	获得电量值	上层及底层用户
ClrEnergyReg	清能量寄存器	上层及底层用户
GetEnergyPerLSB	获得能量单位	上层及底层用户
SetEnergyPerLSB	设置能量单位	上层及底层用户
GetInputRange	获得输入量程	上层及底层用户
SetInputRange	设置输入量程	上层及底层用户
GetEvrmTemp	获得环境温度	上层及底层用户
GetEvrmHum	获得环境湿度	上层及底层用户
⑨ 看门狗		
HostIsOK	下位机无返回信息	上层及底层用户
EnableWatchdog	打开主看门狗(先设置超时间隔，再使能看门狗)	上层及底层用户
CloseWatchdog	禁止看门狗工作	上层及底层用户
GetWatchdogStatus	读取主看门狗的状态	上层及底层用户
ResetWatchdogStatus	Func: 复位主看门狗的状态(S = 0)	上层及底层用户
GetWatchdogTimeoutVal	取得看门狗设置的时间间隔	上层及底层用户
SetWatchdogTimeoutVal	设置看门狗设置的时间间隔	上层及底层用户
⑩ DIGIT LED 设置函数		
GetDLedMode	获得显示模式请求	上层及底层用户
SetDLedMode	修改显示模式请求	上层及底层用户
GetDLedDispChannel	获得 LED 显示通道号	上层及底层用户
SetDLedDispChannel	设置 LED 显示通道号	上层及底层用户
SetDLedValueW	主机控制显示值	上层及底层用户
SetDLedValueA	主机控制显示值	上层及底层用户
⑪ 输入输出任意二进制字符		
WriteDeviceChar	直接写设备	上层及底层用户
ReadDeviceChar	直接读设备	上层及底层用户

⑫ 模块信息确认函数		
InitCheckInfo		上层及底层用户
ReadCheckInfo		上层及底层用户
⑬ 辅助函数		
AdjustSlopeVal	微调当前补偿斜率	上层及底层用户
StoreSlopeVal	设置当前值为输出补偿斜率	上层及底层用户
SetFaultSlopeVal	设定补偿斜率为默认值	上层及底层用户
SetZeroRepair	设置零点偏移补偿	上层及底层用户
SetDevTestMode	设置模块进入测试模式	上层及底层用户
ResetModule	模块软复位	上层及底层用户
GetEnvironmentTemp	取得环境温度(为取热电偶值作准备)	上层及底层用户
SetAdjustTemp	取得环境温度(为取热电偶值作准备)	上层及底层用户
⑭ 测温操作函数		
ReadMeasuringValue	读取测量值	上层及底层用户
TempResetModule	复位测温模块	上层及底层用户
SetSensorSerialNumber	修改传感器编号	上层及底层用户
GetSensorPara	读取传感器参数	上层及底层用户
⑮ Modus 基本功能操作函数		
ReadCoils	读继电器状态	上层及底层用户
ReadDiscretes	读开关量输入	上层及底层用户
ReadMultiRegs	读保持寄存器	上层及底层用户
ReadInputRegs	读输入寄存器	上层及底层用户
WriteCoil	设置单个继电器	上层及底层用户
WriteSingleReg	设置单个保持寄存器	上层及底层用户
ForceMultiCoils	设置多个继电器	上层及底层用户
WriteMultiRegs	设置多个保持寄存器	上层及底层用户
⑯ 数码管显示		
DisplayData	单显部分 显示数据	上层及底层用户
DisplaySpecialSymbols	单显部分 特殊符号	上层及底层用户
ReadData	单显部分 读数据	上层及底层用户
ReadSpecialSymbols	单显部分 读特殊符号	上层及底层用户
DisplayDataSymbols	混显部分 显示数据及特殊符号	上层及底层用户
ReadDataSymbols	混显部分 读数据及特殊符号	上层及底层用户
SetDecimalPoint	设置小数点位置	上层及底层用户
GetDecimalPoint	读取小数点位置	上层及底层用户

使用需知：

Visual C++：

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\DAM3000M\INCLUDE\DAM3000M.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 DAM3000M.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。然后加入如下语句：

#include “DAM3000M.H”

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 DAM3000M.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标

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

二、根据LabView语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如 ReadDeviceAD接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

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

第二节、设备对象管理函数原型说明

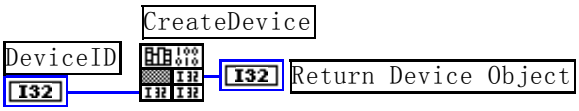
♦ 创建设备对象函数

函数原型：

Visual C++：

HANDLE CreateDevice (LONG lPortNum)

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：lPortNum 串口号，例如串口 1，2，3，4……

常量名	常量值	功能定义
DAM3000 COM1	0x01	串口 1
DAM3000 COM2	0x02	串口 2
DAM3000 COM3	0x03	串口 3

返回值：返回一个串口句柄，以后便可用此句柄对串口进行操作。

相关函数：CreateDevice InitDevice ReleaseDevice

Visual C++程序举例：

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = CreateDevice ( DeviceLgcID ); // 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
 }
    :
```

♦ 初始与模块之间的通信参数

函数原型：

Visual C++：

```
HANDLE InitDevice (HANDLE   hDevice,
                   LONG     lBaud,
                   LONG     lParity,
                   LONG     lTimeOut = DAM3000M_DEFAULT_TIMEOUT)
```

LabVIEW：

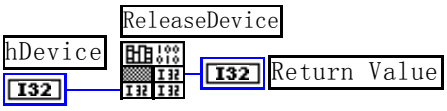
请参考相关演示程序。

功能：初始与模块之间的通信参数。
参数：hDevice 设备对象句柄
lBaud 波特率

常量名	常量值	功能定义
DAM3000_BAUD_1200	0x00	1200波特率
DAM3000 BAUD 2400	0x01	2400波特率
DAM3000 BAUD 4800	0x02	4800波特率
DAM3000 BAUD 9600	0x03	9600波特率
DAM3000 BAUD 19200	0x04	19200波特率
DAM3000 BAUD 38400	0x05	38400波特率
DAM3000 BAUD 57600	0x06	57600波特率
DAM3000 BAUD 115200	0x07	115200波特率

lParity 校验方式
lTimeOut = DAM3000M_DEFAULT_TIMEOUT 超时时间，主要用于接收数据，如果为-1 则使用默认超时时间。
返回值：若成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice InitDevice ReleaseDevice

♦ **释放设备对象所占的系统资源及设备对象**
函数原型：
Visual C++:
BOOL ReleaseDevice(HANDLE hDevice)
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。
相关函数： CreateDevice InitDevice ReleaseDevice

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

第三节、模块信息取得/修改函数

♦ **读取模块信息(类型、地址、波特率、校验)**
函数原型：
Visual C++:
BOOL GetDeviceInfo (HANDLE hDevice,
LONG lDeviceID,
PDAM3000M_DEVICE_INFO pInfo)
LabVIEW:
请参考相关演示程序。

功能：读取模块信息(类型、地址、波特率、校验)。
参数：
hDevice设备对象句柄，它应由 CreateDevice。
lDeviceID 设备地址
PDAM3000M_DEVICE_INFO pInfo 设备信息
返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceInfo GetDevLastError
SetDeviceInfo ReleaseDevice

♦ **修改模块信息(地址、波特率、校验)**

函数原型：

Visual C++:

```
BOOL SetDeviceInfo ( HANDLE hDevice,
                     LONGlDeviceID,
                     DAM3000M_DEVICE_INFO& Info）
```

LabVIEW

请参考相关演示程序。

功能：修改模块信息(地址、波特率、校验)。

参数：

hDevice 设备对象句柄，它应由 CreateDevice。

lDeviceID 设备地址。

DAM3000M_DEVICE_INFO& Info 设备信息

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDeviceInfo GetDevLastError
SetDeviceInfo ReleaseDevice

♦ **获得最后一个错误**

函数原型：

Visual C++:

```
BOOL GetDevLastError (HANDLE hDevice,
                      LONG lDeviceID)
```

LabVIEW:

请参考相关演示程序。

功能：获得最后一个错误

参数：

hDevice设备对象句柄，它应由 CreateDevice。

lDeviceID 模块地址。

返回值：如果初始化设备对象成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetDeviceInfo GetDevLastError
SetDeviceInfo ReleaseDevice

第四节、AD 数据读取函数原型说明

♦ **读取 AD 模拟量输入**

函数原型：

Visual C++:

```
BOOL ReadDeviceAD (HANDLE hDevice,
                   LONG    lDeviceID,
                   LONG    lpADBuffer[],
                   LONG    lBufferSize,
                   LONG    lFirstChannel = 0,
                   LONG    lLastChannel = 0)
```

请参考相关演示程序。

功能：读取 AD 模拟量输入。

参数：hDevice设备对象句柄，它应由 CreateDevice。

lDeviceID, 模块地址

lpADBuffer[] 接收 AD 数据的用户缓冲区 注意:lpADBuffer 最好大于等于 lLastChannel - lFirstChannel +1

lBufferSize 数组 lpADBuffer[]的大小
lFirstChannel = 0 首通道
lLastChannel = 0 末通道

返回值：如果调用成功，则返回 TRUE，且 AD 立刻开始转换。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

♦ **获得模拟量输入模式**

函数原型：

Visual C++:

```
BOOL GetModeAD (HANDLE hDevice,
                LONG lDeviceID,
                PLONG lpMode,
                LONG lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得模拟量输入模式。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpMode AD 模式，取值可有电压（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
DAM3000_VOLT_N15_P15	0x01	-15～+15mV
DAM3000_VOLT_N50_P50	0x02	-50～+50mV
DAM3000_VOLT_N100_P100	0x03	-100～+100mV
DAM3000_VOLT_N150_P150	0x04	-150～+150mV
DAM3000_VOLT_N500_P500	0x05	-500～+500mV
DAM3000_VOLT_N1_P1	0x06	-1～+1V
DAM3000_VOLT_N25_P25	0x07	-2.5～+2.5V
DAM3000_VOLT_N5_P5	0x08	-5～+5V
DAM3000_VOLT_N10_P10	0x09	-10～+10V
DAM3000_VOLT_N0_P5	0x0D	0～+5V
DAM3000_VOLT_N0_P10	0x0E	0～+10V
DAM3000_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
DAM3000_CUR_N0_P10	0x00	0～10mA
DAM3000_CUR_N20_P20	0x0A	-20～+20mA
DAM3000_CUR_N0_P20	0x0B	0～20mA
DAM3000_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
DAM3000_RTD_PT100_385_N200_P850	0x20	-200℃～850℃	Pt100(385)热电阻
DAM3000_RTD_PT100_385_N100_P100	0x21	-100℃～100℃	Pt100(385)热电阻
DAM3000_RTD_PT100_385_N0_P100	0x22	0℃～100℃	Pt100(385)热电阻
DAM3000_RTD_PT100_385_N0_P200	0x23	0℃～200℃	Pt100(385)热电阻
DAM3000_RTD_PT100_385_N0_P600	0x24	0℃～600℃	Pt100(385)热电阻
DAM3000_RTD_PT100_3916_N200_P850	0x25	-200℃～850℃	Pt100(3916)热电阻
DAM3000_RTD_PT100_3916_N100_P100	0x26	-100℃～100℃	Pt100(3916)热电阻
DAM3000_RTD_PT100_3916_N0_P100	0x27	0℃～100℃	Pt100(3916)热电阻
DAM3000_RTD_PT100_3916_N0_P200	0x28	0℃～200℃	Pt100(3916)热电阻
DAM3000_RTD_PT100_3916_N0_P600	0x29	0℃～600℃	Pt100(3916)热电阻
DAM3000_RTD_PT1000	0x30	-200℃～850℃	Pt1000 热电阻
DAM3000_RTD_CU50	0x40	-50℃～150℃	Cu50 热电阻
DAM3000_RTD_CU100	0x41	-50℃～150℃	Cu100 热电阻
DAM3000_RTD_BA1	0x42	-200℃～650℃	BA1 热电阻
DAM3000_RTD_BA2	0x43	-200℃～650℃	BA2 热电阻
DAM3000_RTD_G53	0x44	-50℃～150℃	G53 热电阻
DAM3000_RTD_Ni50	0x45	100℃	Ni50 热电阻
DAM3000_RTD_Ni508	0x46	0℃～100℃	Ni508 热电阻
DAM3000_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
DAM3000_THERMOCOUPLE_J	0x10	0～1200℃	J型热电偶
DAM3000_THERMOCOUPLE_K	0x11	0～1300℃	K型热电偶
DAM3000_THERMOCOUPLE_T	0x12	0～400℃	T型热电偶
DAM3000_THERMOCOUPLE_E	0x13	0～1000℃	E型热电偶
DAM3000_THERMOCOUPLE_R	0x14	0～1700℃	R型热电偶
DAM3000_THERMOCOUPLE_S	0x15	0～1768℃	S型热电偶
DAM3000_THERMOCOUPLE_B	0x16	0～1800℃	B型热电偶
DAM3000_THERMOCOUPLE_N	0x17	0～1300℃	N型热电偶
DAM3000_THERMOCOUPLE_C	0x18	0～2090℃	C型热电偶

与电压的换算关系如下：

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℃

lChannel = 0 通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse
ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit
GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD
GetGroundingAD ReleaseDevice

◆ **设置 AD 输入模式**

函数原型：

Visual C++:

```
BOOL SetModeAD (HANDLE hDevice,
                LONG lDeviceID,
                PLONG lMode,
                LONG lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：设置 AD 输入模式。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lMode AD 模式。

lChannel = 0 通道号

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **获得通道接地模式**

函数原型：

Visual C++:

```
BOOL GetGroundingAD ( HANDLE hDevice,
                      LONG lDeviceID,
                      PLONG lpGrounding,
                      LONG lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得通道接地模式(只对可软件配置单/双端输入模块有效)。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpGrounding AD 通道接地模式。

lChannel = 0 通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **设置通道接地模式**

函数原型：

Visual C++:

```
BOOL SetGroundingAD ( HANDLE hDevice,
                      LONG lDeviceID,
                      LONG lGrounding,
                      LONG lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：设置通道接地模式(只对可软件配置单/双端输入模块有效)。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lGrounding AD 通道接地模式。
lChannel = 0 通道号
返回值：若成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **获得模拟量输入报警下限值**

函数原型：
Visual C++:

```
BOOL GetLowLimitVal( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG lLowLimit[],
                     LONG lFirstChannel = 0,
                     LONG lLastChannel = 0)
```

LabVIEW:
请参考相关演示程序。

功能：获得模拟量输入报警下限值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lLowLimit[]下限报警值。
lFirstChannel = 0 首通道号。
lLastChannel = 0 末通道号。
返回值：若成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **获得模拟量输入报警上限值**

函数原型：
Visual C++:

```
BOOL GetLowLimitVal( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG lHighLimit[],
                     LONG lFirstChannel = 0,
                     LONG lLastChannel = 0)
```

LabVIEW:
请参考相关演示程序。

功能：获得模拟量输入报警上限值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lHighLimit[],上限报警值。
lFirstChannel = 0 首通道号。
lLastChannel = 0 末通道号。
返回值：若成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse

ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit
GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD
GetGroundingAD ReleaseDevice

♦ **设置下限报警值**

函数原型：

Visual C++:

```
BOOL SetLowLimitVal( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG lLowLimit[],
                     LONG lFirstChannel = 0,
                     LONG lLastChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：设置模拟量输入报警下限值。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lLowLimit[]下限报警值。

lFirstChannel = 0 首通道号。

lLastChannel = 0 末通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse
ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit
GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD
GetGroundingAD ReleaseDevice

♦ **设置模拟量输入报警上限值**

函数原型：

Visual C++:

```
BOOL SetLowLimitVal( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG lHighLimit[],
                     LONG lFirstChannel = 0,
                     LONG lLastChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：设置模拟量输入报警上限值。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lHighLimit[],上限报警值。

lFirstChannel = 0 首通道号。

lLastChannel = 0 末通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse
ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit
GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD
GetGroundingAD ReleaseDevice

♦ **获得报警的电平**

函数原型：

Visual C++:

```
BOOL GetAlarmPulse (HANDLE hDevice,
                    LONG lDeviceID,
                    PLONG  lpAlarmVal,
                    LONG   lChannel = 0)
```

LabVIEW：

请参考相关演示程序。

功能：获得报警的电平。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpAlarmVal 报警电平,0:低电平, 1:高电平

lChannel = 0 通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **设置模拟量输入报警电平**

函数原型：

Visual C++:

```
BOOL SetAlarmPulse (HANDLE hDevice,
                    LONG   lDeviceID,
                    LONG   lAlarmVal,
                    LONG   lChannel = 0)
```

LabVIEW：

请参考相关演示程序。

功能：设置模拟量输入报警电平。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lAlarmVal 报警电平,0:低电平, 1:高电平

lChannel = 0 通道号。

返回值：若成功，则返回 TRUE， 否则返回 FALSE。

相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD GetGroundingAD ReleaseDevice

◆ **获得报警状态**

函数原型：

Visual C++:

```
BOOL GetAlarmSts (HANDLE hDevice,
                  LONG   lDeviceID,
                  PLONG  lpAlarmSts,
                  LONG   lChannel = 0)
```

LabVIEW：

请参考相关演示程序。

功能：获得报警状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpAlarmSts 报警状态。
lChannel = 0 通道号。
返回值：若成功，则返回 TRUE， 否则返回 FALSE。
相关函数： CreateDevice GetAlarmSts SetHighLimitVal SetAlarmPulse
ValGetAlarmPulse SetModeAD GetLowLimitVal SetLowLimit
GetModeAD ReadDeviceAD GetHighLimitVal SetGroundingAD
GetGroundingAD ReleaseDevice

第五节、DA 数据采样操作函数原型说明

- **回读 DA 输出值**

函数原型：
Visual C++：

```
BOOL GetDeviceDAVal ( HANDLE hDevice,
                      LONG   lDeviceID,
                      PLONG lpDAValue,
                      LONG   lChannel = 0)
```

LabView：
请参考相关演示程序。

功能：回读 DA 输出值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lpDAValue DA 输出值

量程(伏)	计算机语言换算公式	Lsb 取值范围
±5000mV	Lsb = Volt / (10000 / 4096) + 2048	[0，4095]
0～10000mV	Lsb = Volt/ (10000 / 4096)	[0，4095]
±10000mV	Lsb = Volt/ (20000 / 4096) + 2048	[0，4095]

lChannel = 0 通道号。
返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA
GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA
SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

- **设定单通道 DA**

函数原型：
Visual C++：

```
BOOL WriteDeviceDA ( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG   lDAData,
                     LONG   lChannel = 0)
```

LabView：
请参考相关演示程序。

功能：设定单通道 DA。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lDAData DA 输出值
lChannel = 0 通道号。
返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA

GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA
SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

◆ **设定单通道 DA**

函数原型：

Visual C++：

```
BOOL WriteDeviceDA ( HANDLE hDevice,
                     LONG   lDeviceID,
                     LONG   lDAData,
                     LONG   lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：设定单通道 DA。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lDAData DA 输出值

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA
GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA
SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

◆ **读取模拟量输出量程**

函数原型：

Visual C++：

```
BOOL GetOutPutRangeDA ( HANDLE hDevice,
                        LONG lDeviceID,
                        PLONG lpRange,
                        LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：读取模拟量输出量程。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lpRange DA 输出量程。

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA
GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA
SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

◆ **设置模拟量输出量程**

函数原型：

Visual C++：

```
BOOL SetOutPutRangeDA ( HANDLE hDevice,
                        LONG lDeviceID,
                        PLONG lpRange,
                        LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：设置模拟量输出量程。

参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lRange DA 输出量程。
lChannel = 0 通道号。
返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

♦ **获得 DA 上电值**

函数原型：
Visual C++：

```
BOOL GetPowerOnValueDA (HANDLE hDevice,
                        LONG lDeviceID,
                        PLONG lpPowerOnVal,
                        LONG lChannel = 0)
```

LabView：
请参考相关演示程序。

功能：获得 DA 上电值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
1pPowerOnVal 上电值。
lChannel = 0 通道号。
返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

♦ **设置 DA 上电值**

函数原型：
Visual C++：

```
BOOL SetPowerOnValueDA ( HANDLE hDevice,
                         LONG lDeviceID,
                         LONG lPowerOnVal,
                         LONG lChannel = 0)
```

LabView：
请参考相关演示程序。

功能：设置 DA 上电值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
1PowerOnVal 上电值。
lChannel = 0 通道号。
返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

♦ **获得 DA 安全值**

函数原型：
Visual C++：

```
BOOL GetSafeValueDA ( HANDLE hDevice,
                      LONG lDeviceID,
                      PLONG lpSafeVal,
                      LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：获得 DA 安全值。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lpSafeVal 安全值。

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

- **设置 DA 安全值**

函数原型：

Visual C++：

```
BOOL SetSafeValueDA ( HANDLE hDevice,
                      LONG lDeviceID,
                      LONG lSafeVal,
                      LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：设置 DA 安全值。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lSafeVal 安全值。

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetPowerOnValueDA WriteDeviceDA SetOutPutRangeDA GetSlopeDA SetPowerOnValueDA SetSafeValueDA GetSafeValueDA SetSlopeDA GetOutPutRangeDA GetDeviceDAVal ReleaseDevice

- **读模拟量输出斜率**

函数原型：

Visual C++：

```
BOOL GetSlopeDA ( HANDLE hDevice,
                  LONG lDeviceID,
                  PLONG lpSlopeType,
                  LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：读模拟量输出斜率。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lpSlopeType 输出斜率类型。

表（3-7）模拟量输出斜率类型

常量名	常量值	功能定义
DAM3000_SLOPE_IMMEDIATE	0x00	Immediate
DAM3000_SLOPE_POINT125	0x01	0.125 mA/S
DAM3000_SLOPE_POINT25	0x02	0.25 mA/S
DAM3000_SLOPE_POINT5	0x03	0.5 mA/S
DAM3000_SLOPE_1	0x04	1.0 mA/S
DAM3000_SLOPE_2	0x05	2.0 mA/S
DAM3000_SLOPE_4	0x06	4.0 mA/S
DAM3000_SLOPE_8	0x07	8.0 mA/S
DAM3000_SLOPE_16	0x08	16.0 mA/S
DAM3000_SLOPE_32	0x09	32.0 mA/S
DAM3000_SLOPE_64	0x0A	64.0 mA/S
DAM3000_SLOPE_128	0x0B	128.0 mA/S
DAM3000_SLOPE_256	0x0C	256.0 mA/S
DAM3000_SLOPE_512	0x0D	512.0 mA/S
DAM3000_SLOPE_1024	0x0E	1024.0 mA/S
DAM3000_SLOPE_2048	0x0F	2048.0 mA/S

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetPowerOnValueDA　WriteDeviceDA　SetOutPutRangeDA
GetSlopeDA　SetPowerOnValueDA　SetSafeValueDA　GetSafeValueDA
SetSlopeDA　GetOutPutRangeDA　GetDeviceDAVal　ReleaseDevice

- **修改模拟量输出斜率**

函数原型：

Visual C++：

```
BOOL SetSlopeDA ( HANDLE hDevice,
                  LONG lDeviceID,
                  LONG lSlopeType,
                  LONG lChannel = 0)
```

LabView：

请参考相关演示程序。

功能：修改模拟量输出斜率。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lSlopeType　输出斜率类型。

lChannel = 0 通道号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetPowerOnValueDA　WriteDeviceDA　SetOutPutRangeDA
GetSlopeDA　SetPowerOnValueDA　SetSafeValueDA　GetSafeValueDA
SetSlopeDA　GetOutPutRangeDA　GetDeviceDAVal　ReleaseDevice

第六节、DI 输入操作函数原型说明

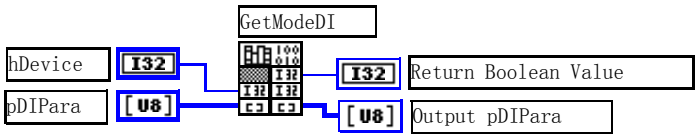
♦ 读取数字量输入的工作模式

函数原型：

Visual C++:

```
BOOL GetModeDI (HANDLE hDevice,
                LONG   lDeviceID,
                LONG   lMode[],
                LONG   lEdgeMode[],
                LONG   lFirstChannel = 0,
                LONG   lLastChannel = 0)
```

LabView：



功能：读取数字量输入的工作模式。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备对象句柄。

lMode[] 输入的工作模式,0:DI 模式,1:计数方式,2:锁存方式。

lEdgeMode[] 边沿方式,0:低电平,1:高电平。

lFirstChannel = 0 首通道。

lLastChannel = 0 末通道。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

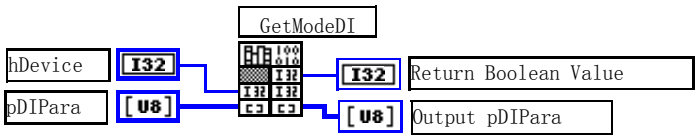
♦ 设置数字量输入的工作模式

函数原型：

Visual C++:

```
BOOL SetModeDI (HANDLE hDevice,
                LONG   lDeviceID,
                LONG   lMode，
                LONG   lEdgeMode
                LONG   lChannel = 0)
```

LabView：



功能：读取数字量输入的工作模式。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

lMode[] 输入的工作模式,0:DI 模式,1:计数方式,2:锁存方式。

lEdgeMode 边沿方式,0:低电平,1:高电平。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

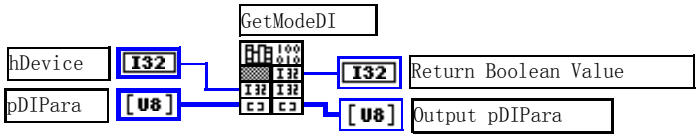
♦ **读取开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI (HANDLE hDevice,
                  LONG   lDeviceID,
                  PDAM3000M_PARA_DI pDIPara,
                  LONG   lBufferSize)
```

LabView：



功能：读取开关量输入。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

pDIPara DI 值。

lBufferSize 通道大小。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

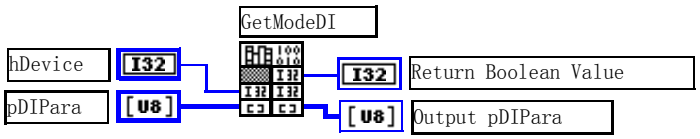
♦ **启动 DI 计数**

函数原型：

Visual C++:

```
BOOL StartDeviceDI (HANDLE hDevice,
                    LONG   lDeviceID,
                    LONG   lChannel = 0)
```

LabView：



功能：启动 DI 设计。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

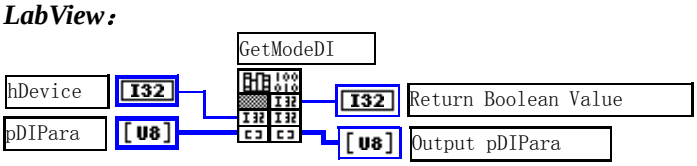
♦ **停止 DI 计数**

函数原型：

Visual C++:

```
BOOL StopDeviceDI (HANDLE hDevice,
                   LONGlDeviceID,
                   LONGlChannel = 0)
```

LabView：



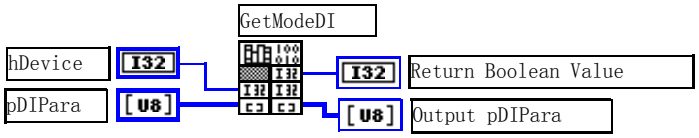
功能：停止 DI 设计。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

- **读取 DI 计数器值**

函数原型：
Visual C++:

```
BOOL GetCNTDI (HANDLE hDevice,
               LONG   lDeviceID,
               PLONG lpCounterValue,
               LONG   lFirstChannel = 0,
               LONG   lLastChannel = 0)
```

LabView:



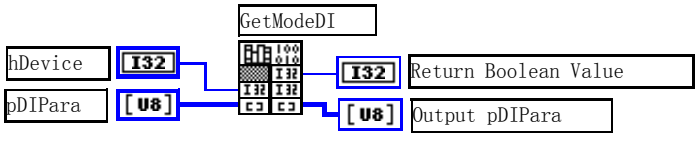
功能：读取 DI 计数器值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
lpCounterValue 范围(0~65535)
lFirstChannel = 0 首通道
lLastChannel = 0 末通道
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

- **设置 DI 计数器初始值**

函数原型：
Visual C++:

```
BOOL SetCNTDI (HANDLE hDevice,
               LONG   lDeviceID,
               LONG   lInitValue,
               LONG   lChannel = 0)
```

LabView:



功能：设置 DI 计数器初始值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lInitValue 计数初值。
lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

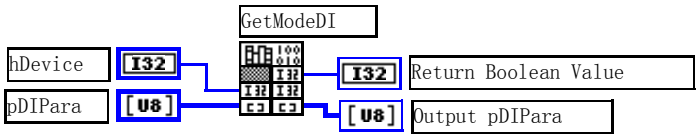
- **启动锁存**

函数原型

Visual C++:

```
BOOL StartLatch (HANDLE hDevice,
                 LONG   lDeviceID,
                 LONG   lChannel = 0)
```

LabView：



功能：启动锁存。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

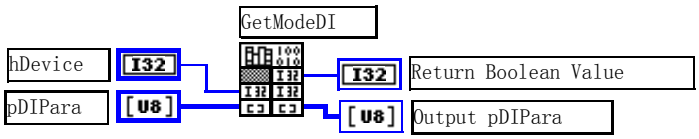
- **停止锁存**

函数原型：

Visual C++:

```
BOOL StopLatch(HANDLE hDevice,
               LONG   lDeviceID,
               LONG   lChannel = 0)
```

LabView：



功能：停止锁存。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

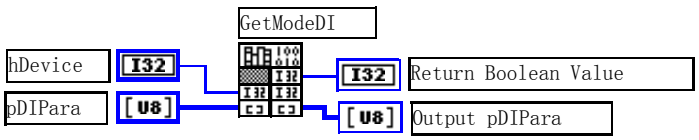
- **读锁存状态**

函数原型：

Visual C++:

```
BOOL GetLatchStatus (HANDLE hDevice,
                     LONG lDeviceID,
                     LONG lLatchType,
                     PDAM3000M_PARA_LATCH pLatchStatus,
                     LONG lBufferSize)
```

LabView：



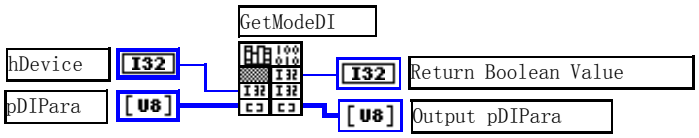
功能：读锁存状态。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
lLatchType 锁存类型，分为上升沿和下降沿锁存。
pLatchStatus 锁存状态。
lBufferSize 通道大小。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

♦ **清除计数值**

函数原型：
Visual C++:

```
BOOL GetLatchStatus (HANDLE hDevice,
                     LONG lDeviceID,
                     LONG lChannel = 0)
```

LabView：



功能：停止锁存。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetModeDI SetModeDI StartLatch StopDeviceDI GetDeviceDI StartDeviceDI GetCNTDI SetCNTDI GetLatchStatus ClearLatchStatus ClearCNTVal StopLatch ReleaseDevice

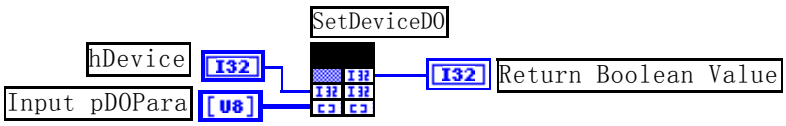
第七节、DO 数字量输出操作函数原型说明

♦ **回读开关量输出**

函数原型：
Visual C++:

```
BOOL GetDeviceDO (HANDLE hDevice,
                  LONG lDeviceID,
                  PDAM3000M_PARA_DO pDOPara)
```

LabView：



功能：回读开关量输出。
参数：

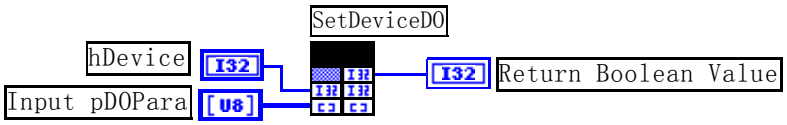
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
pDOPara 当前 DO 输出值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDeviceDI SetSafeValueDO GetSafeValueDO SetDeviceDO SetPowerOnValueDO GetPowerOnValueDO GetDeviceDO ReleaseDevice

- **设置 DO 开关量输出值**

函数原型：
Visual C++:

```
BOOL SetDeviceDO(HANDLE hDevice,
                 LONG lDeviceID,
                 BYTE byDOSts[],
                 LONG lFirstChannel,
                 LONG lLastChannel)
```

LabView：



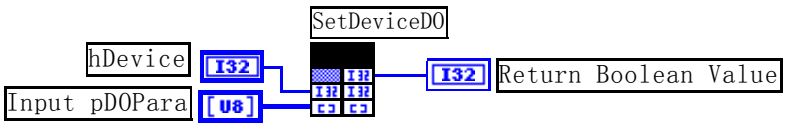
功能：设置 DO 开关量输出值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
byDOSts[] 设置 DO 输出值。
lFirstChannel 首通道号。
lLastChannel 末通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDeviceDI SetSafeValueDO GetSafeValueDO SetDeviceDO SetPowerOnValueDO GetPowerOnValueDO GetDeviceDO ReleaseDevice

- **获取 DO 上电初始值**

函数原型：
Visual C++:

```
BOOL GetPowerOnValueDO (HANDLE hDevice,
                        LONG   lDeviceID,
                        PDAM3000M_PARA_DO pPowerOnPara)
```

LabView：



功能：获取 DO 上电初始值。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
lDeviceID 设备地址。
pPowerOnPara 上电值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDeviceDI SetSafeValueDO GetSafeValueDO SetDeviceDO SetPowerOnValueDO GetPowerOnValueDO GetDeviceDO ReleaseDevice

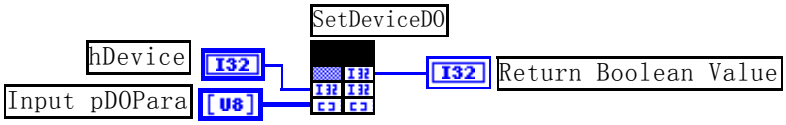
- **设置 DO 上电初始值**

函数原型：

Visual C++:

```
BOOL SetPowerOnValueDO (HANDLE hDevice,
                        LONG   lDeviceID,
                        DAM3000M_PARA_DO PowerOnPara)
```

LabView：



功能：设置 DO 上电初始值。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

PowerOnPara 上电值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceDI　SetSafeValueDO　GetSafeValueDO　SetDeviceDO　SetPowerOnValueDO　GetPowerOnValueDO　GetDeviceDO　ReleaseDevice

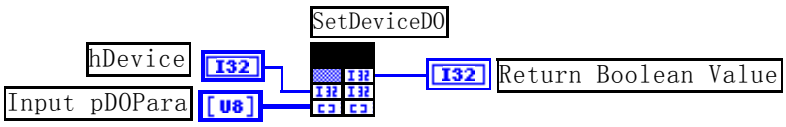
- **读 DO 安全值**

函数原型：

Visual C++:

```
BOOL GetSafeValueDO (HANDLE hDevice,
                     LONG   lDeviceID,
                     PDAM3000M_PARA_DO pDOSafePara)
```

LabView：



功能：读 DO 安全值。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

lDeviceID 设备地址。

pDOSafePara 安全值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceDI　SetSafeValueDO　GetSafeValueDO　SetDeviceDO　SetPowerOnValueDO　GetPowerOnValueDO　GetDeviceDO　ReleaseDevice

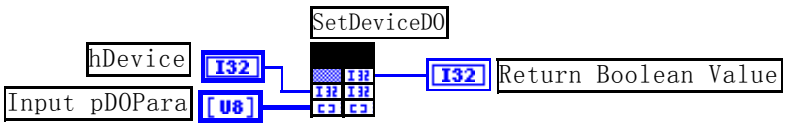
- **设置安全值**

函数原型：

Visual C++:

```
BOOL SetSafeValueDO (HANDLE hDevice,
                     LONG   lDeviceID,
                     DAM3000M_PARA_DO DOSafePara)
```

LabView：



功能：读 DO 安全值。

参数：

hDevice 设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。
DOSafePara 安全值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDeviceDI SetSafeValueDO GetSafeValueDO SetDeviceDO SetPowerOnValueDO GetPowerOnValueDO GetDeviceDO ReleaseDevice

第八节、计数器操作函数原型说明

- **对各个计数器进行参数设置**

函数原型：
Visual C++:

```
BOOL SetCounterMode (HANDLE hDevice,
                     LONG   lDeviceID,
                     PDAM3000M_PARA_CNT pCNTPara,
                     LONG    lChannel = 0)
```

LabVIEW:
请参考相关演示程序。

功能：对各个计数器进行参数设置。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
pCNTPara 基于各通道的计数器参数。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH GetCounterSts GetCounterCurVal StopCounter StartCounter GetFreqCurVal EnableFilter InitCounterFilter ResetCounter ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO SetCounterAlarmMode ReleaseDevice

- **初始化报警的工作模式**

函数原型：
Visual C++:

```
BOOL InitCounterAlarm (HANDLE hDevice,
                       LONG   lDeviceID,
                       PDAM3000M_CNT_ALARM pCNTAlarm)
```

LabVIEW:
请参考相关演示程序。

功能：初始化报警的工作模式。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
pCNTAlarm 报警参数设置。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH GetCounterSts GetCounterCurVal StopCounter StartCounter GetFreqCurVal EnableFilter InitCounterFilter ResetCounter ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO SetCounterAlarmMode ReleaseDevice

- **设置计数器报警方式**

函数原型：
Visual C++:

```
BOOL SetCounterAlarmMode (HANDLE hDevice,
                          LONG lDeviceID,
                          LONG lAlarmMode)
```

LabVIEW:

请参考相关演示程序。

功能：设置计数器报警方式。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lAlarmMode 报警方式。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetCounterMode, InitCounterAlarm, SetLEDCounterCH, GetCounterSts, GetCounterCurVal, StopCounter, StartCounter, GetFreqCurVal, EnableFilter, InitCounterFilter, ResetCounter, ClearAlarmSts, GetLEDCounterCH, GetCounterAlarmSts, SetCounterDO, SetCounterAlarmMode, ReleaseDevice

◆ **获得计数器设备硬件参数状态**

函数原型：

Visual C++:

```
BOOL GetCounterSts (HANDLE hDevice,
                    LONG   lDeviceID,
                    PDAM3000M_CNT_STATUS pStsCNT,
                    LONG    lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得计数器设备硬件参数状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

pStsCNT 返回值。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice, SetCounterMode, InitCounterAlarm, SetLEDCounterCH, GetCounterSts, GetCounterCurVal, StopCounter, StartCounter, GetFreqCurVal, EnableFilter, InitCounterFilter, ResetCounter, ClearAlarmSts, GetLEDCounterCH, GetCounterAlarmSts, SetCounterDO, SetCounterAlarmMode, ReleaseDevice

◆ **启动计数器计数**

函数原型：

Visual C++:

```
BOOL StartCounter (HANDLE hDevice,
                   LONG lDeviceID,
                   LONG   lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：启动计数器计数。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

◆ **停止计数器计数**

函数原型：

Visual C++:

```
BOOL StopCounter (HANDLE hDevice,
                  LONG   lDeviceID,
                  LONG   lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：停止计数器计数。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

◆ **取得计数器当前值**

函数原型：

Visual C++:

```
BOOL GetCounterCurVal (HANDLE hDevice,
                       LONG lDeviceID,
                       PULONG   pulCNTVal,
                       LONG lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：取得计数器当前值。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

pulCNTVal 计数值

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

◆ **取得频率器当前值**

函数原型：

Visual C++:

```
BOOL GetFreqCurVal (HANDLE hDevice,
                    LONG   lDeviceID,
                    PULONG pulFreqVal,
                    LONG   lChannel = 0)
```

LabVIEW:
请参考相关演示程序。

功能：取得频率器当前值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
pulFreqVal 频率值。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

◆ **计数器复位**
函数原型：
Visual C++:
```
BOOL ResetCounter (HANDLE hDevice,
                   LONG   lDeviceID,
                   LONG   lChannel = 0)
```
LabVIEW:
请参考相关演示程序。

功能：计数器复位。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

◆ **初始化滤波**
函数原型：
Visual C++:
```
BOOL InitCounterFilter (HANDLE hDevice,
                        LONG   lDeviceID,
                        PDAM3000M_PARA_FILTER pFilter,
                        LONG   lChannel = 0)
```
LabVIEW:
请参考相关演示程序。

功能：初始化滤波。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
pFilter 滤波参数。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter

GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

♦ **使能滤波状态**

函数原型：

Visual C++:

```
BOOL EnableFilter (HANDLE hDevice,
                   LONG   lDeviceID,
                   BOOL    bEnable,
                   LONG   lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：使能滤波状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

bEnable 使能滤波。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

♦ **获得 DO 及报警状态**

函数原型：

Visual C++:

```
BOOL GetCounterAlarmSts(HANDLE hDevice,
                        LONG   lDeviceID,
                        PLONG plEnableAlarm,
                        PLONG pbDO,
                        LONG   lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得 DO 及报警状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

plEnableAlarm 计数器报警状态。

pbDO DO

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

♦ **设置 DO**

函数原型：

Visual C++:

```
BOOL SetCounterDO(HANDLE hDevice,
                  LONG   lDeviceID,
```

BYTE byDOSts[],
LONG lFirstChannel,
LONG lLastChannel)

LabVIEW:
请参考相关演示程序。

功能：设置 DO。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
byDOSts[] DO。
LFirstChannel 首通道号。
LLastChannel 末通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, SetCounterMode, InitCounterAlarm, SetLEDCounterCH, GetCounterSts, GetCounterCurVal, StopCounter, StartCounter, GetFreqCurVal, EnableFilter, InitCounterFilter, ResetCounter, ClearAlarmSts, GetLEDCounterCH, GetCounterAlarmSts, SetCounterDO, SetCounterAlarmMode, ReleaseDevice

- **清报警方式 1 报警输出**

函数原型：
Visual C++:
BOOL ClearAlarmSts (HANDLE hDevice,
LONG lDeviceID)
LabVIEW:
请参考相关演示程序。

功能：清报警方式 1 报警输出。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, SetCounterMode, InitCounterAlarm, SetLEDCounterCH, GetCounterSts, GetCounterCurVal, StopCounter, StartCounter, GetFreqCurVal, EnableFilter, InitCounterFilter, ResetCounter, ClearAlarmSts, GetLEDCounterCH, GetCounterAlarmSts, SetCounterDO, SetCounterAlarmMode, ReleaseDevice

- **取得计数器 LED 显示通道**

函数原型：
Visual C++:
BOOLGetLEDCounterCH (HANDLE hDevice,
LONG lDeviceID,
PLONG plChannel)
LabVIEW:
请参考相关演示程序。

功能：取得计数器 LED 显示通道。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
PlChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice, SetCounterMode, InitCounterAlarm, SetLEDCounterCH, GetCounterSts, GetCounterCurVal, StopCounter, StartCounter

GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

♦ **设置计数器 LED 显示通道**

函数原型：

Visual C++:

```
BOOL SetLEDCounterCH (HANDLE hDevice,
                      LONG lDeviceID,
                      LONG lChannel)
```

LabVIEW:

请参考相关演示程序。

功能：设置计数器 LED 显示通道。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

PlChannel 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetCounterMode InitCounterAlarm SetLEDCounterCH
GetCounterSts GetCounterCurVal StopCounter StartCounter
GetFreqCurVal EnableFilter InitCounterFilter ResetCounter
ClearAlarmSts GetLEDCounterCH GetCounterAlarmSts SetCounterDO
SetCounterAlarmMode ReleaseDevice

第九节、电量模块函数原型说明

♦ **获得电量值**

函数原型：

Visual C++:

```
BOOL GetEnergyVal(HANDLE hDevice,
                  LONG lDeviceID,
                  LONG  lValue[],
                  LONG  lAanlogType,
                  LONG  lFirstChannel = 0,
                  LONG  lLastChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得电量值。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lValue[] 电量值。

lAanlogType 模拟量类型。

lFirstChannel = 0 首通道。

lLastChannel = 0 末通道。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB
SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB
GetEvrmHum ReleaseDevice

♦ **清能量寄存器**

函数原型：

Visual C++:

```
BOOL ClrEnergyReg (HANDLE hDevice,
                   LONG lDeviceID,
```

LONG lChannel = 0)

LabVIEW:

请参考相关演示程序。

功能：清能量寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lChannel = 0 通道。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB GetEvrmHum ReleaseDevice

- **获得能量单位**

函数原型：

Visual C++:

```
BOOL GetEnergyPerLSB (HANDLE hDevice,
                      LONG lDeviceID,
                      PLONG lpEnergyPerLSB)
```

LabVIEW:

请参考相关演示程序。

功能：获得能量单位。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lpEnergyPerLSB 能量单位。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB GetEvrmHum ReleaseDevice

- **设置能量单位**

函数原型：

Visual C++:

```
BOOL SetEnergyPerLSB (HANDLE hDevice,
                      LONG lDeviceID,
                      LONG lEnergyPerLSB)
```

LabVIEW:

请参考相关演示程序。

功能：设置能量单位。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lEnergyPerLSB 能量单位。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB GetEvrmHum ReleaseDevice

- **获得输入量程**

函数原型：

Visual C++:

```
BOOL GetInputRange (HANDLE hDevice,
```

LONG lDeviceID,
LONG lInputRangeV,
LONG lInputRangeI)

LabVIEW:
请参考相关演示程序。

功能：获得输入量程。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lInputRangeV 电压输入量程。
lInputRangeI 电流输入量程。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB GetEvrmHum ReleaseDevice

- **设置输入量程**

函数原型：
Visual C++:
BOOL SetInputRange (HANDLE hDevice,
LONG lDeviceID,
PLONG lpInputRangeV,
PLONG lpInputRangeI)

LabVIEW:
请参考相关演示程序。

功能：设置输入量程。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lpInputRangeV 电压输入量程。
lpInputRangeI 电流输入量程。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB GetEvrmHum ReleaseDevice

- **获得环境温度**

函数原型：
Visual C++:
BOOL GetEvrmTemp(HANDLE hDevice,
LONG lDeviceID,
PLONG lpEvrmTemp,
LONG lChannel = 0)

LabVIEW:
请参考相关演示程序。

功能：获得环境温度。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 设备地址。
lpEvrmTemp 温度。
lChannel = 0 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB

SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB
GetEvrmHum ReleaseDevice

◆ **获得环境湿度**

函数原型：

Visual C++:

```
BOOL GetEvrmTemp(HANDLE hDevice,
                 LONG   lDeviceID,
                 PLONG lpEvrmHum,
                 LONG   lChannel = 0)
```

LabVIEW:

请参考相关演示程序。

功能：获得环境湿度。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 设备地址。

lpEvrmHum 湿度。

lChannel = 0 通道号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ClrEnergyReg GetEnergyVal GetEnergyPerLSB
SetInputRange GetEvrmTemp GetInputRange SetEnergyPerLSB
GetEvrmHum ReleaseDevice

第十节、看门狗函数原型说明

◆ **下位机无返回信息**

函数原型：

Visual C++:

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

LabVIEW:

请参考相关演示程序。

功能：下位机无返回信息。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

◆ **打开主看门狗(先设置超时间隔，再使能看门狗)**

函数原型：

Visual C++:

```
BOOL EnableWatchdog (HANDLE hDevice,
                     LONG   lDeviceID)
```

LabVIEW:

请参考相关演示程序。

功能：打开主看门狗(先设置超时间隔，再使能看门狗)。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal

GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

- **禁止看门狗工作**

函数原型：

Visual C++:

```
BOOL EnableWatchdog (HANDLE hDevice,
                     LONG   lDeviceID)
```

LabVIEW:

请参考相关演示程序。

功能：禁止看门狗工作。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

- **读取主看门狗的状态(S = 1, Host is down; S = 0 OK)**

函数原型：

Visual C++:

```
BOOL GetWatchdogStatus (HANDLE hDevice,
                        LONG lDeviceID,
                        PLONG lpWatchdogStatus)
```

LabVIEW:

请参考相关演示程序。

功能：读取主看门狗的状态(S = 1, Host is down; S = 0 OK)。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpWatchdogStatus 看门狗状态。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

- **复位主看门狗的状态(S = 0)**

函数原型：

Visual C++:

```
BOOL ResetWatchdogStatus (HANDLE hDevice,
                          LONG lDeviceID)
```

LabVIEW:

请参考相关演示程序。

功能：复位主看门狗的状态(S = 0)。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

- **取得看门狗设置的时间间隔**

函数原型：
Visual C++:
BOOL GetWatchdogTimeoutVal (HANDLE hDevice,
LONG lDeviceID,
PLONG lpInterval)
LabVIEW:
请参考相关演示程序。

功能：取得看门狗设置的时间间隔。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lpInterval 时间间隔。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

- **设置看门狗设置的时间间隔**

函数原型：
Visual C++:
BOOL SetWatchdogTimeoutVal (HANDLE hDevice,
LONG lDeviceID,
LONG lInterval)
LabVIEW:
请参考相关演示程序。

功能：取得看门狗设置的时间间隔。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lpInterval 时间间隔。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice HostIsOK GetWatchdogStatus
EnableWatchdog CloseWatchdog GetWatchdogTimeoutVal
GetWatchdogStatus ResetWatchdogStatus ReleaseDevice

第十一节、DIGIT LED 设置函数原型说明

- **获得显示模式请求**

函数原型：
Visual C++:
BOOL GetDLedMode (HANDLE hDevice,
LONG lDeviceID,
PLONG lpDispMode)
LabVIEW:
请参考相关演示程序。

功能：获得显示模式请求。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lpDispMode 显示模式 0x00：温度格式,0x01:欧姆值。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode
SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

♦ **修改显示模式请求**

函数原型：

Visual C++:

```
BOOL SetDLedMode (HANDLE hDevice,
                  LONG   lDeviceID,
                  LONG   lDispMode)
```

LabVIEW:

请参考相关演示程序。

功能：修改显示模式请求。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lDispMode 显示模式 0x01：温度格式,0x02:欧姆值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

♦ **获得 LED 显示通道号**

函数原型：

Visual C++:

```
BOOLGetDLedDispChannel (HANDLE hDevice,
                        LONG lDeviceID,
                        PLONG lpChannel)
```

LabVIEW:

请参考相关演示程序。

功能：获得 LED 显示通道号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lpChannel 通道号,lpChannel = 0xff:主机控制显示。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

♦ **设置 LED 显示通道号**

函数原型：

Visual C++:

```
BOOL SetDLedDispChannel (HANDLE hDevice,
                         LONG lDeviceID,
                         LONG lChannel= 0)
```

LabVIEW:

请参考相关演示程序。

功能：设置 LED 显示通道号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

lDeviceID 模块地址。

lChannel= 0 通道号,lpChannel = 0xff:主机控制显示。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

♦ **主机控制显示值**

函数原型：
Visual C++:
```
BOOL SetDLedValueW (HANDLE hDevice,
                    LONG lDeviceID,
                    LPCWSTR strWriteBuf,
                    LONG    llength)
```
LabVIEW:
请参考相关演示程序。

功能：主机控制显示值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
strWriteBuf 显示的字符串
llength 数据长度。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

- **主机控制显示值**

函数原型：
Visual C++:
```
BOOL SetDLedValueA (HANDLE hDevice,
                    LONG lDeviceID,
                    LPCWSTR strWriteBuf,
                    LONG    llength)
```
LabVIEW:
请参考相关演示程序。

功能：主机控制显示值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
strWriteBuf 显示的字符串
llength 数据长度。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice GetDLedMode GetDLedDispChannel SetDLedMode SetDLedValueW SetDLedValueA SetDLedDispChannel ReleaseDevice

第十二节、输入输出任意二进制字符

- **直接写设备**

函数原型：
Visual C++:
```
BOOL WriteDeviceChar(HANDLE hDevice,
                     char* strWriteBuf,
                     long     llength,
                     long     timeout = 100)
```
LabVIEW:
请参考相关演示程序。

功能：直接写设备。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
strWriteBuf 写的数据。
Llength 数据长度。

timeout = 100 超时范围(mS)。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice　WriteDeviceChar　WriteDeviceChar　ReleaseDevice

- **直接读设备**

函数原型：
Visual C++:

```
BOOL WriteDeviceChar(HANDLE hDevice,
                     char*strReadBuf,
                     long    llength,
                     long    timeout = 100)
```

LabVIEW:
请参考相关演示程序。

功能：直接读设备。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
strReadBuf 读取的数据。
Llength 数据长度。
timeout = 100 超时范围(mS)。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice　WriteDeviceChar　WriteDeviceChar　ReleaseDevice

第十三节、辅助函数原型说明

- **微调当前补偿斜率**

函数原型：
Visual C++:

```
BOOL AdjustSlopeVal(HANDLE hDevice,
                    LONG   lDeviceID,
                    LONG lAdjustVal,
                    LONG lChannel)
```

LabVIEW:
请参考相关演示程序。

功能：微调当前补偿斜率。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lAdjustVal 微调值。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数：CreateDevice　AdjustSlopeVal　SetFaultSlopeVal　StoreSlopeVal
SetZeroRepair　SetDevTestMode　ResetModule　SetAdjustTemp
GetEnvironmentTemp　ReleaseDevice

- **设置当前值为输出补偿斜率**

函数原型：
Visual C++:

```
BOOL StoreSlopeVal(HANDLE hDevice,
                   LONG lDeviceID,
                   LONG lChannel)
```

LabVIEW:
请参考相关演示程序。

功能：设置当前值为输出补偿斜率。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

◆ **设定补偿斜率为默认值**

函数原型：
Visual C++:

```
BOOL SetFaultSlopeVal (HANDLE hDevice,
                       LONG lDeviceID,
                       LONG lChannel)
```

LabVIEW:
请参考相关演示程序。

功能：设定补偿斜率为默认值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

◆ **设置零点偏移补偿**

函数原型：
Visual C++:

```
BOOL SetZeroRepair (HANDLE hDevice,
                    LONG lDeviceID,
                    LONG lZeroRepair,
                    LONG  lChannel)
```

LabVIEW:
请参考相关演示程序。

功能：设置零点偏移补偿。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lDeviceID 模块地址。
lZeroRepair 零点值。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

◆ **设置模块进入测试模式**

函数原型：
Visual C++:

```
BOOL SetDevTestMode (HANDLE hDevice,
                     LONG lDeviceID)
```

LabVIEW:

请参考相关演示程序。

功能：设置模块进入测试模式。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

- **模块软复位**

函数原型：
Visual C++:
```
BOOL ResetModule(HANDLE hDevice,
                 LONG lDeviceID)
```
LabVIEW:
请参考相关演示程序。

功能：模块软复位。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

- **取得环境温度(为取热电偶值作准备)**

函数原型：
Visual C++:
```
BOOL GetEnvironmentTemp (HANDLE hDevice,
                         LONG lDeviceID,
                         PFLOAT lpETemprt)
```
LabVIEW:
请参考相关演示程序。

功能：取得环境温度(为取热电偶值作准备)。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

- **取得环境温度(为取热电偶值作准备)**

函数原型：
Visual C++:
```
BOOL SetAdjustTemp(HANDLE hDevice,
                   LONG lDeviceID,
                   BYTE   lETemprt)
```
LabVIEW:
请参考相关演示程序。

功能：取得环境温度(为取热电偶值作准备)。

参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
lChannel 通道号。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice AdjustSlopeVal SetFaultSlopeVal StoreSlopeVal SetZeroRepair SetDevTestMode ResetModule SetAdjustTemp GetEnvironmentTemp ReleaseDevice

第十四节、测温操作函数原型说明

◆ **读取测量值**

函数原型：
Visual C++:

```
BOOL ReadMeasuringValue(HANDLE hDevice,
                        LONG lDeviceID,
                        WORD  lpADBuffer[])
```

LabVIEW:
请参考相关演示程序。

功能：读取测量值。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
LDeviceID 模块地址。
lpADBuffer[] 接收数据的用户缓冲区 128。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice TempResetModule ReadMeasuringValue GetSensorPara SetSensorSerialNumber ReleaseDevice

◆ **复位测温模块**

函数原型：
Visual C++:

```
BOOL TempResetModule(HANDLE hDevice,
                     LONG lDeviceID)
```

LabVIEW:
请参考相关演示程序。

功能：复位测温模块。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
LDeviceID 模块地址。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice TempResetModule ReadMeasuringValue GetSensorPara SetSensorSerialNumber ReleaseDevice

◆ **修改传感器编号**

函数原型：
Visual C++:

```
BOOL SetSensorSerialNumber (HANDLE hDevice,
                            LONG lDeviceID,
                            BYTE   bChannel,
                            BYTE   bSequenceNumber,
                            BYTE   bNewNumber,
                            BOOL   bAuto)
```

LabVIEW:
请参考相关演示程序。

功能：修改传感器编号。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
LDeviceID 模块地址。
bChannel 通道号。
bSequenceNumber 通道内顺序号。
bNewNumber 新编号。
bAuto 是否自动编号 FALSE 手动编号 TURE 自动编号(通道号 通道内顺序号 新编号 参数不起作用)
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice TempResetModule ReadMeasuringValue GetSensorPara SetSensorSerialNumber ReleaseDevice

- **读取传感器参数**

函数原型：
Visual C++:

```
BOOL GetSensorPara (HANDLE hDevice,
                    LONG lDeviceID,
                    DAM3000M_SENSOR_PARA pInfo[128])
```

LabVIEW:
请参考相关演示程序。

功能：读取传感器参数。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
LDeviceID 模块地址。
pInfo[128] 传感器参数。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice TempResetModule ReadMeasuringValue GetSensorPara SetSensorSerialNumber ReleaseDevice

第十五节、Modus 基本功能操作函数原型说明

- **读继电器状态**

函数原型：
Visual C++:

```
BOOL ReadCoils (HANDLE hDevice,
                LONG lDeviceID,
                int  addr,
                int  len,
                BYTE  bCoilsFlag[])
```

LabVIEW:
请参考相关演示程序。
功能：读继电器状态。
参数：
hDevice设备对象句柄，它应由 CreateDevice创建。
LDeviceID 模块地址。
返回值：若成功，返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **读继电器状态**

函数原型：
Visual C++:

```
BOOL ReadCoils (HANDLE hDevice,
                LONG lDeviceID,
```

int addr,
int len,
BYTE bCoilsFlag[])

LabVIEW:

请参考相关演示程序。

功能：读继电器状态。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **读开关量输入**

函数原型：

Visual C++:

```
BOOL ReadDiscretes (HANDLE hDevice,
                    LONG lDeviceID,
                    int addr,
                    int len,
                    BYTE bDIState[])
```

LabVIEW:

请参考相关演示程序。

功能：读开关量输入。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **读保持寄存器**

函数原型：

Visual C++:

```
BOOL ReadMultiRegs (HANDLE hDevice,
                    LONG lDeviceID,
                    int addr,
                    int len,
                    BYTE buf[])
```

LabVIEW:

请参考相关演示程序。

功能：读保持寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **读输入寄存器**

函数原型：

Visual C++:

```
BOOL ReadInputRegs (HANDLE hDevice,
```

LONG lDeviceID,
int addr,
int len,
BYTE buf[])

LabVIEW:

请参考相关演示程序。

功能：读输入寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **设置单个继电器**

函数原型：

Visual C++:

BOOL ReadInputRegs (HANDLE hDevice,
LONG lDeviceID,
int addr,
int len,
BYTE status)

LabVIEW:

请参考相关演示程序。

功能：设置单个继电器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **设置单个保持寄存器**

函数原型：

Visual C++:

BOOL WriteSingleReg(HANDLE hDevice,
LONG lDeviceID,
int addr,
short val)

LabVIEW:

请参考相关演示程序。

功能：设置单个保持寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **设置多个继电器**

函数原型：

Visual C++:

BOOL ForceMultiCoils (HANDLE hDevice,

LONG lDeviceID,
int addr,
int len,
BYTE bDOState[])

LabVIEW:

请参考相关演示程序。

功能：设置多个继电器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **设置多个保持寄存器**

函数原型：

Visual C++:

```
BOOL WriteMultiRegs (HANDLE hDevice,
                     LONG lDeviceID,
                     int addr,
                     int len,
                     BYTE buf[])
```

LabVIEW:

请参考相关演示程序。

功能：设置多个保持寄存器。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

第十六节、数码管显示操作函数原型说明

- **单显部分 显示数据**

函数原型：

Visual C++:

```
BOOL DisplayData(HANDLE hDevice,
                 LONG lDeviceID,
                 WORD wData)
```

LabVIEW:

请参考相关演示程序。

功能：单显部分 显示数据。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

wData 数据。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **单显部分 特殊符号**

函数原型：

Visual C++:

```
BOOL DisplaySpecialSymbols (HANDLE hDevice,
                            LONG lDeviceID,
                            WORD wSymbols)
```

LabVIEW:

请参考相关演示程序。

功能：单显部分 特殊符号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

wSymbols 符号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **单显部分 读数据**

函数原型：

Visual C++:

```
BOOL ReadData (HANDLE hDevice,
               LONG lDeviceID,
               PWORD pwData)
```

LabVIEW:

请参考相关演示程序。

功能：单显部分 读数据。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

pwData 数据。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **单显部分 读特殊符号**

函数原型：

Visual C++:

```
BOOL ReadSpecialSymbols (HANDLE hDevice,
                         LONG lDeviceID,
                         PWORD pwSymbols)
```

LabVIEW:

请参考相关演示程序。

功能：单显部分 读特殊符号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

pwSymbols 符号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **混显部分 显示数据及特殊符号**

函数原型：

Visual C++:

```
BOOL DisplayDataSymbols (HANDLE hDevice,
                         LONG lDeviceID,
                         WORD  wData,
                         WORD  wSymbols)
```

LabVIEW:

请参考相关演示程序。

功能：混显部分 显示数据及特殊符号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

wData 数据。

wSymbols 符号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **混显部分 读数据及特殊符号**

函数原型：

Visual C++:

```
BOOL ReadDataSymbols (HANDLE hDevice,
                      LONG lDeviceID,
                      PWORD pwData,
                      PWORD pwSymbols)
```

LabVIEW:

请参考相关演示程序。

功能：混显部分 读数据及特殊符号。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

pwData 数据。

pwSymbols 符号。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **设置小数点位置**

函数原型：

Visual C++:

```
BOOL SetDecimalPoint (HANDLE hDevice,
                      LONG lDeviceID,
                      LONG  lDecimalPoint)
```

LabVIEW:

请参考相关演示程序。

功能：设置小数点位置。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

lDecimalPoint 小数点位置。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

- **读取小数点位置**

函数原型：

Visual C++:

```
BOOL GetDecimalPoint (HANDLE hDevice,
                      LONG lDeviceID,
                      PLONG  plDecimalPoint)
```

LabVIEW:

请参考相关演示程序。

功能：设置小数点位置。

参数：

hDevice设备对象句柄，它应由 CreateDevice创建。

LDeviceID 模块地址。

plDecimalPoint 小数点位置。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReadCoils ReadDiscretes ReadMultiRegs ReadInputRegs WriteCoil WriteSingleReg ForceMultiCoils WriteMultiRegs ReleaseDevice

第四章 硬件参数结构

第一节、开关量输出的参数结构

Visual C++：

```
typedef struct _DAM3000M_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
    BYTE DO8;           // 8 通道
    BYTE DO9;           // 9 通道
    BYTE DO10;          // 10 通道
    BYTE DO11;          // 11 通道
    BYTE DO12;          // 12 通道
    BYTE DO13;          // 13 通道
    BYTE DO14;          // 14 通道
    BYTE DO15;          // 15 通道
} DAM3000M_PARA_DO, *PDAM3000M_PARA_DO;
```

第二节、开关量输入的参数结构

Visual C++：

```
typedef struct    _DAM3000M_PARA_DI         // 数字量输入参数(1 为高电平)
{
    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
    BYTE DI8;           // 8 通道
    BYTE DI9;           // 9 通道
    BYTE DI10;          // 10 通道
    BYTE DI11;          // 11 通道
    BYTE DI12;          // 12 通道
    BYTE DI13;          // 13 通道
    BYTE DI14;          // 14 通道
    BYTE DI15;          // 15 通道
    BYTE DI16;          // 16 通道
    BYTE DI17;          // 17 通道
    BYTE DI18;          // 18 通道
    BYTE DI19;          // 19 通道
    BYTE DI20;          // 20 通道
    BYTE DI21;          // 21 通道
    BYTE DI22;          // 22 通道
    BYTE DI23;          // 23 通道
    BYTE DI24;          // 24 通道
    BYTE DI25;          // 25 通道
    BYTE DI26;          // 26 通道
    BYTE DI27;          // 27 通道
    BYTE DI28;          // 28 通道
    BYTE DI29;          // 29 通道
    BYTE DI30;          // 30 通道
    BYTE DI31;          // 31 通道
} DAM3000M_PARA_DI, *PDAM3000M_PARA_DI;
```

第三节、模拟量输入通道配置结构体

Visual C++：

```
typedef struct _DAM3000M_ADCHANNEL_ARRAY
{
    BYTE bChannel0;     // 1，有效；0，无效
    BYTE bChannel1;
    BYTE bChannel2;
    BYTE bChannel3;
    BYTE bChannel4;
    BYTE bChannel5;
    BYTE bChannel6;
    BYTE bChannel7;
}DAM3000M_ADCHANNEL_ARRAY, *PDAM3000M_ADCHANNEL_ARRAY;
```

第四节、计数器参数配置结构体

Visual C++：

```
typedef struct _DAM3000M_PARA_CNT   // 基于各通道的计数器参数结构体
{
    LONG WorkMode;          // 计数器/频率工作模式
    LONG FreqBuildTime;     // 测频器建立时间, 单位: s
    LONG InputMode;         // 计数器/频率输入方式    0: 非隔离    1: 隔离
    ULONG InitVal;          // 计数器初始值
    ULONG MaxVal;           // 计数器最大值
    LONG GateSts;           // 门槛值状态(计数模式)
} DAM3000M_PARA_CNT, *PDAM3000M_PARA_CNT;
typedef struct _DAM3000M_CNT_ALARM
```

```
{
    LONG AlarmMode;                 // 报警方式
    LONG EnableAlarm0;          // 0 通道报警使能
    LONG EnableAlarm1;          // 1 通道报警使能
    ULONG Alarm0Val;            // 0 通道报警值
    ULONG Alarm1Val;            // 1 通道报警值
    ULONG Alarm0HiHiVal;    // 0 通道上上限(Hi-Hi)报警值，报警方式 1 有效
} DAM3000M_CNT_ALARM, *PDAM3000M_CNT_ALARM;

typedef struct _DAM3000M_PARA_FILTER          // 用于计数器滤波的参数结构体
{
    LONG TrigLevelHigh;         // 触发高电平(非隔离输入)
    LONG TrigLevelLow;          // 触发低电平(非隔离输入)
    LONG MinWidthHigh;          // 高电平最小输入信号宽度
    LONG MinWidthLow;           // 低电平最小输入信号宽度
    LONG bEnableFilter;         // 使能滤波
} DAM3000M_PARA_FILTER, *PDAM3000M_PARA_FILTER;
//    LONG DisplayChannel; // 设置显示通道          0：0 通道计数/频率，1：1 通道计数/频率

typedef struct _DAM3000M_CNT_STATUS                 // 计数器硬件参数状态结构体
{
    LONG WorkMode;              // 计数器/频率工作模式*
    LONG FreqBuildTime;         // 测频器建立时间，单位: s*
    LONG InputMode;             // 计数器/频率输入方式     0: 非隔离      1: 隔离*
    LONG bCNTSts;               // 计数/频率器的状态(起停状态)*
    LONG FilterSts;             // 计数器的滤波状态*
    LONG MinWidthHigh;          // 高电平最小输入信号宽度*
    LONG MinWidthLow;           // 低电平最小输入信号宽度*
    LONG TrigLevelHigh;         // 触发高电平(非隔离输入)*
    LONG TrigLevelLow;          // 触发低电平(非隔离输入)*
    LONG GateSts;               // 门槛值设置状态(计数模式)*
    ULONG MaxVal;               // 计数器最大值*
    ULONG InitVal;              // 计数器初始值*
    LONG bOverflowSts;          // 计数器溢出状态*
    LONG AlarmMode;                 // 计数器报警方式*
    LONG EnableAlarm0;          // 计数器 0 报警使能状态*
    LONG EnableAlarm1;          // 计数器 1 报警使能状态*
    ULONG Alarm0Val;            // 0 通道报警值*
    ULONG Alarm1Val;            // 1 通道报警值*
    ULONG Alarm1HiHiVal;    // 报警方式 1 上上限(Hi-Hi)报警值*
    LONG bDO0;                  // DO0*
    LONG bDO1;                  // DO1*
} DAM3000M_CNT_STATUS, *PDAM3000M_CNT_STATUS;
```

第五节、设备基本信息的结构体

Visual C++：

```
typedef struct _DAM3000M_DEVICE_INFO
{
    LONG      DeviceType;       // 模块类型
    LONG      TypeSuffix;       // 类型后缀
    LONG    ModusType;          // M
    LONG    VesionID;           // 版本号(2 字节)
    LONG    DeviceID;           // 模块 ID 号(SetDeviceInfo 时，为设备的新 ID)
    LONG    BaudRate;           // 波特率
```

```
    LONG    bParity;        // 无校验 偶校验 奇校验(目前仅支持 3080D)
} DAM3000M_DEVICE_INFO, *PDAM3000M_DEVICE_INFO;
```

第六节、测温模块传感器参数的结构体

Visual C++：

```
typedef struct _DAM3000M_SENSOR_PARA
{
    BYTE    SerialNumber;// 编号
    BYTE    Channel;     // 所在通道号
    BYTE    SequenceNumber;  // 通道内顺序号
    WORD    Temperature;  // 温度
    BYTE    ID[8];              // ID 号
    BYTE    AlarmMark;       // 报警标志
} DAM3000M_SENSOR_PARA, *PDAM3000M_SENSOR_PARA;
```

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

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

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/4096) * (ADBuffer[0] &0x0FFF) – 10000.00	[-10000, +9995.11]
±5000mV	Volt = (10000.00/4096) * (ADBuffer[0] &0x0FFF) – 5000.00	[-5000, +4997.55]
0～10V	Volt = (10000.00/4096) * (ADBuffer[0] &0x0FFF)	[0, +9997.55]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++：

```
Lsb = ADBuffer[0] &0x0FFF;
Volt = (20000.00/4096) * Lsb -10000.00;
```

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，当通道总数首末通道相等时，假如此时首末通道=5，其排放规则如下：

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

两通道采集：

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：

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

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读

取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效呢？还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

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

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

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeaderSizeBytes;          // 文件头信息长度
    LONG FileType;
    // 该设备数据文件共有的成员
    LONG BusType;                  // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;                    // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;              // 头信息版本(D31-D16=Major   D15-D0=Minijor) = 1.0
    LONG VoltBottomRange;          // 量程下限(mV)
    LONG VoltTopRange;             // 量程上限(mV)
    DAM3000M_PARA_AD ADPara;       // 保存硬件参数

    LONGLONG nTriggerPos;          // 触发点位置
    LONG BatCode;                  // 同批文件识别码
    LONG HeadEndFlag;              // 文件结束位
} FILE_HEADER,   *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer[]缓冲区排放的规则一样，即每 16 位二

进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第四节、DA 的电压值如何转换成输出到 DA 转换器的 LSB 原码数据？

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

请注意这里求得的LSB数据就是用于 WriteDeviceBulkDA中的DABuffer[]参数的。

第五节、关于 DA 数据 DABuffer 缓冲区中的数据排放规则

由于各个通道的段信息与波形数据均共享一个板载物理 RAM，它们的排放顺序如图，系统默认值为两个通道均分整个 RAM 空间，即默认每通道 RAM 空间为 256K 点。从下图可以看出，各个通道所占 RAM 空间不一定相等，可大可小，只是两个通道的总空间不能大于板载物理 RAM 空间即可。

通道 0	通道 1
0 至(256K-1)	256 至(512K-1)

关于每个通道 RAM 空间的内部分配是这样的，其空间首部存放的是所有段的段信息数据，其后才是各个段的波形数据，再其后可能还有未用空间。假如有三个分段，如图：

段 0 信息	段 1 信息	段 2 信息	段 0 波形数据	段 1 波形数据	段 2 波形数据	未用空间

关于每个段的段信息包括的内容有：该段波形数据在 RAM 中的起始地址、终止地址、段循环次数，如下表，注意其段起始地址和终止地址是由 DAM3000M_PARA_DA 中的 SegmentInfo 决定的。

板载 RAM 内存单元(16Bit)	各单元定义	有效位
0	段 0 波形数据起始地址低 12 位	D11:D0
1	段 0 波形数据起始地址高 8 位	D7:D0
2	段 0 波形数据终止地址低 12 位	D11:D0
3	段 0 波形数据终止地址高 8 位	D7:D0
4	段 0 循环次数低 12 位	D11:D0
5	段 0 循环次数高 8 位	D7:D0
6	段 1 波形数据起始地址低 12 位	D11:D0
7	段 1 波形数据起始地址高 8 位	D7:D0
8	段 1 波形数据终止地址低 12 位	D11:D0
9	段 1 波形数据终止地址高 8 位	D7:D0
10	段 1 循环次数低 12 位	D11:D0
11	段 1 循环次数高 8 位	D7:D0
:	:	:
段信息结束后便是波形数据	段信息结束后便是波形数据	D11:D0

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。

Visual C++:

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]\|**[阿尔泰测控演示系统]**\|**[Microsoft Visual C++]**\|**[简易代码演示]**\|**[AD 演示]**

然后，您着重参考以下函数：

BOOL SetADMode(); // 设置模拟量输入模式

首先，使用 SetADMode 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 GetADMode(); // 获得模拟输入模式

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

Visual C++:

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 SeTypeDA(); // 设置模拟量输出模式

BOOL WriteDeviceDA(); // 写模拟量输出

第三节、怎样使用 GetDeviceDI函数进行更便捷的数字开关量输入操作

Visual C++:

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 工程。

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

BOOL SetDIMode（）； //设置 DI 输入模式

首先，使用 SetDIMode 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）
   SetCNTDI()
   StartDeciceDI()
   StopDeviceDI()
   GetCNTDI()
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 SetDeviceDO函数进行更便捷的数字开关量输出操作

Visual C++:

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]

SetDeviceDO(); // 设置 DO 当前输出值

GetDeviceDO(); // 读取 DO 输出值