PXI2512 光电隔离DI/DO板

驱动程序使用手册

北京阿尔泰科技发展有限公司

V6.00.00

ART Technology

目　录

1 版权信息与命名约定

1.1 版权信息与命名约定

本软件产品及相关套件均属北京阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位、我公司授权的代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，我们将热情接待。

1.2 版权信息与命名约定

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 PXIxxxx_则被省略。如 PXI2512_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			

2 使用纲要

2.1 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 SetDeviceDO 等。而底层用户函数如 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。

2.2 如何管理 PXI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 CreateDevice 函数创建一个设备对象句柄 hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如 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。

2.3 如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 SetDeviceDO 函数实现开关量的输出操作，其各路开关量的输出状态由其 bDOSts[32]中的相应元素决定。由 GetDeviceDI 函数实现开关量的输入操作，其各路开关量的输入状态由其 bDISts[32]中的相应元素决定。

2.4 DIO 数字量的输入输出

当用户调用 DEV_Create()函数创建了 hDevice 设备对象句柄后，可调用 DIO_ReadPort()函数实现数字量的端口输入操作，调用 DIO_ReadLines()或 DIO_ReadLine()实现数字量的线输入操作，可调用 DIO_WritePort()函数实现数字量的端口输出操作，调用 DIO_WriteLines()或 DIO_WriteLine()实现数字量的线输出操作。

2.5 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 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 PXI 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。

3 PXI 即插即用设备操作函数接口介绍

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

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

3.1 设备驱动接口函数总列表（每个函数省略了前缀“PXI2512_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PXI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCurrentID	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
GetDeviceCount	取得同一种 PXI 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PXI 总线设备对象	上层及底层用户
② DIO 开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	上层用户

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\PXI2512\INCLUDE\PXI2512.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 PXI2512.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 “PXI2512.H”

3.2 设备对象管理函数原型说明

♦ **创建设备对象函数（逻辑号）**

函数原型：

Visual C++:

HANDLE CreateDevice (int DeviceID)

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

参数：DeviceID 设备 ID 标识号。

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

相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PXI2512_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
 }
    :
```

♦ **取得本计算机系统中 PXI2512 设备的总数量**

函数原型：

Visual C++:

int GetDeviceCount (HANDLE hDevice)

功能：取得 PXI2512 设备的数量。

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

返回值：返回系统中 PXI2512 的数量。

相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

♦**取得该设备当前相应的 ID 号**

函数原型：

Visual C++:

BOOL GetDeviceCurrentID (HANDLE hDevice,
PLONG DeviceLgcID,
PLONG DevicePhysID）

功能：取得指定设备相应 ID 号。

参数：

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

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

DevicePhysID　返回设备的物理 ID 号。

返回值：如果初始化设备对象成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

相关函数： GetDeviceCount　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

◆ **用对话框控件列表计算机系统中所有 PXI2512 设备各种配置信息**

函数原型：

Visual C++:

BOOL ListDeviceDlg (HANDLE hDevice)

功能：列表系统中 PXI2512 的硬件配置信息。

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

返回值：若成功，则弹出对话框控件列表所有 PXI2512 设备的配置情况。

相关函数： CreateDevice　ReleaseDevice

◆ **释放设备对象所占的系统资源及设备对象**

函数原型：

Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

功能：释放设备对象所占用的系统资源及设备对象自身。

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

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

相关函数： CreateDevice

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

3.3　DIO 数字量输入输出开关量操作函数原型说明

◆ **开关量输入**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   BYTE bDISts[32])
```

功能：负责将 PXI 设备上的输入开关量状态读入内存。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

bDISts 三十二路开关量输入状态的参数结构，共有 32 个元素，分别对应于 DI0~DI31 路开关量

输入状态位。如果 bDISts[0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

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

相关函数： CreateDevice GetDeviceDI ReleaseDevice

♦ **开关量输出**

函数原型：

Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[32])
```

功能：负责将 PXI 设备上的输出开关量置成相应的状态。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

bDOSts 三十二路开关量输出状态的参数结构，共有 32 个成员变量，分别对应于 DO0~DO31 路开关量输出状态位。比如置 bDOSts[0]为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO31 共 32 个成员变量赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice SetDeviceDO ReleaseDevice

♦ **以上函数调用一般顺序**

① CreateDevice

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

③ ReleaseDevice

用户可以反复执行第②步，以进行数字 I/O 的输入输出。

4 上层用户函数接口应用实例

4.1 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 Sys 工程。

[程序]｜[阿尔泰测控演示系统]｜[PXI2512 继电器、开关量输入及中断卡]｜[Microsoft Visual C++]｜[简易代码演示]

4.2 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 Sys 工程。

[程序]｜[阿尔泰测控演示系统]｜[PXI2512 32-channel DIO 卡]｜[Microsoft Visual C++]｜[简易代码演示]

5 共用函数介绍

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

5.1 公用接口函数总列表（每个函数省略了前缀“PXI2512_”）

函数名	函数功能	备注
① ISA 总线 I/O 端口操作函数		
GetDeviceAddr	取得指定 PXI 设备寄存器操作基地址	底层用户
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
②线程操作函数		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

5.2 IO 端口读写函数原型说明

注意：若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

◆ **取得指定内存映射寄存器的线性地址和物理地址**

函数原型：

Visual C++:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PUCHAR*LinearAddr,
                    PUCHAR*PhysAddr,
                    int RegisterID = 0)
```

功能：取得 PXI 设备指定的内存映射寄存器的线性地址。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

LinearAddr 指针参数，用于取得的映射寄存器指向的线性地址，RegisterID 指定的寄存器组属于 MEM 模式时该值不应为零，也就是说它可用于 WriteRegisterX 或 ReadRegisterX（X 代表 Byte、

ULong、Word）等函数，以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。但如果 RegisterID 指定的寄存器组属于 I/O 模式时该值通常为零，您不能通过以上函数访问设备。

PhysAddr 指针参数，用于取得的映射寄存器指向的物理地址，它指明该设备位于系统空间的物理位置。如果由 RegisterID 指定的寄存器组属于 I/O 模式，则可用于 WritePortX 或 ReadPortX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。

RegisterID 指定映射寄存器的 ID 号，其取值范围为[0，5]，通常情况下，用户应使用 0 号映射寄存器，特殊情况下，我们为用户加以申明。

返回值：如果执行成功，则返回 TRUE，它表明由 RegisterID 指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回，否则会返回 FALSE，同时还要检查其 LinearAddr 和 PhysAddr 是否为 0，若为 0 则依然视为失败。用户可用 GetLastErrorEx 捕获当前错误码，并加以分析。

相关函数： CreateDevice　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr;
hDevice = CreateDevice(0);
if(!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox("取得设备地址失败…");
}
    :
```

- **以单字节(8Bit)方式写 I/O 端口**

函数原型：

Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    UINT nPort,
                    BYTE Value)
```

功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码。

相关函数： CreateDevice　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

- **以双字(16Bit)方式写 I/O 端口**

函数原型：

Visual C++:

BOOL WritePortWord (HANDLE hDevice,
UINT nPort,
WORD Value)

功能：以双字(16Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码。

相关函数： CreateDevice GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以四字节(32Bit)方式写 I/O 端口**

函数原型：

Visual C++:

BOOL WritePortULong(HANDLE hDevice,
UINT nPort,
ULONG Value)

功能：以四字节(32Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回 TRUE，否则返回 FALSE，用户可用 GetLastErrorEx 捕获当前错误码。

相关函数： CreateDevice GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以单字节(8Bit)方式读 I/O 端口**

函数原型：

Visual C++:

BYTE ReadPortByte(HANDLE hDevice,
UINT nPort)

功能：以单字节(8Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数： CreateDevice GetDeviceAddr WritePortByte
WritePortWord WritePortULong ReadPortByte
ReadPortWord ReadPortULong ReleaseDevice

- **以双字节(16Bit)方式读 I/O 端口**

函数原型：

Visual C++:

WORD ReadPortWord(HANDLE hDevice,
UINT nPort)

功能：以双字节(16Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数： CreateDevice　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

- **以四字节(32Bit)方式读 I/O 端口**

函数原型：

Visual C++:

ULONG ReadPortULong(HANDLE hDevice,
UINT nPort)

功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

nPort　设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

相关函数： CreateDevice　GetDeviceAddr　WritePortByte
WritePortWord　WritePortULong　ReadPortByte
ReadPortWord　ReadPortULong　ReleaseDevice

5.3　线程操作函数原型说明

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

- **创建内核系统事件**

函数原型：

Visual C++:

HANDLE CreateSystemEvent(void)

功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

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

- **释放内核系统事件**

函数原型：

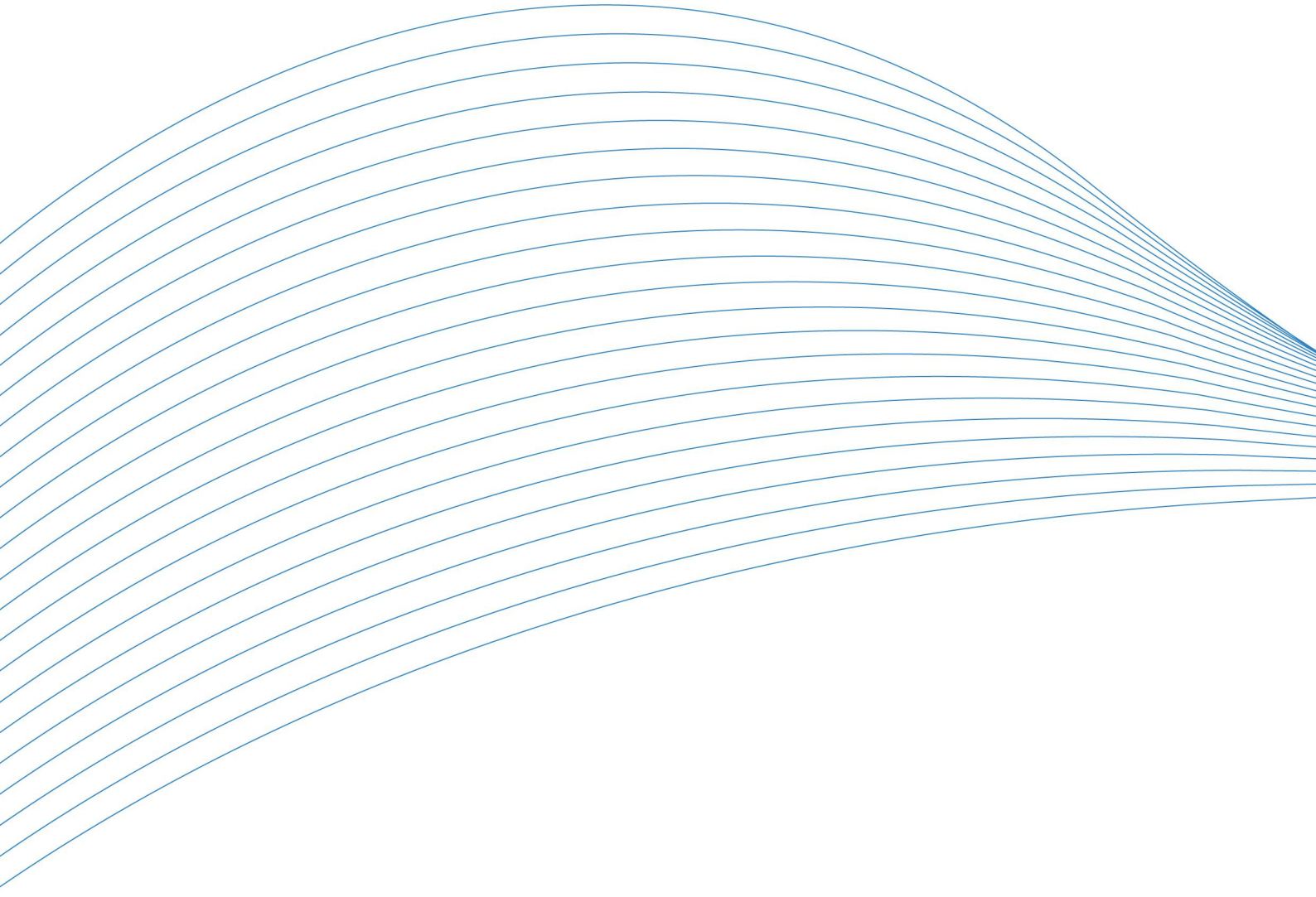
Visual C++:

BOOL ReleaseSystemEvent(HANDLE hEvent)

功能：释放系统内核事件对象。

参数：hEvent　被释放的内核事件对象。它应由 CreateSystemEvent 成功创建的对象。

返回值：若成功，则返回 TRUE。



北京阿尔泰科技发展有限公司
服务热线：400-860-3335
邮编：100086
传真：010-62901157