USB5630 数据采集卡

驱动程序使用手册

北京阿尔泰科技发展有限公司

V6.00.00

阿尔泰科技
www.art-control.com

关于本手册

本手册为阿尔泰科技推出的 USB5630 数据采集卡驱动程序使用手册，其中包括版权信息与命名约定、使用纲要、各功能操作流程介绍、设备操作函数接口介绍、硬件参数结构、数据格式转换与排列规则、上层用户函数接口应用实例、共用函数介绍、修改历史等。

文档版本：V6.00.00

目　录

1 规范与约定

1.1 关键字缩写命名约定

缩写	全称	汉语意思	缩写	全称	汉语意思
DEV/Dev	Device	设备	DIR/Dir	Direction	方向
AI	Analog Input	模拟量输入	CPLG	Coupling	耦合
AO	Analog Output	模拟量输出	ATR	Analog Trigger	模拟量触发
DI	Digital Input	数字量单向输入	DTR	Digital Trigger	数字量触发
DO	Digital Output	数字量单向输出	Cur	Current	当前的
DIO	Digital Input/Output	数字量双向输入输出	ID	Identifier	标识
CTR	Counter	计数器或定时器	Idx	Index	索引
PARAM/Param	Parameter	参数	DI	Differential	差分(接地方式)
TRIG/Trig	Trigger	触发	SE	Single end	单端(接地方式)
CLK	Clock	时钟	REG	Register	寄存器
GND	Ground	地	Sens	Sensitivity	灵敏度
AGND	Analog Ground	模拟地	Pt	Point	点
DGND	Digital Ground	数字地	Pts	Points	点数
Lgc	Logical	逻辑的	Chan/CH	Channel	通道号
Phys	Physical	物理的	AUX	Auxiliary	辅助
Pio	Program I/O	软件 IO 传输模式	Buf	Buffer	缓冲
Int	Interrupt	中断传输模式	En	Enable	允许或使能
Dma	Direct Memory Access	直接内存存取(传输方式)	SRC/Src	Source	源
SAMP/Samp	Sample	采样			

1.2 数据类型

1.2.1 基本数据类型

类型名称	类型描述	数据范围	各编程语言支持类型		
			C/C++/CVI/ C_Builder	Visual Basic	Pascal(Delphi)
I8	有符号 8 位整型数	-128 to 127	char	无此数据类型用 Byte 代替	ShortInt
U8	无符号 8 位整型数	0 to 255	unsigned char	Byte	Byte
I16	有符号 16 位整型数	-32768 to +32767	short	Integer	SamllInt
U16	无符号 16 位整型数	0 to 65535	unsigned short	无此数据类型用 Integer 代替	Word
I32	有符号 32 位整型数	-2147483648 to 2147483647	int	Long	LongInt
U32	无符号 32 位整型数	0 to 4294967295	unsigned int	无此数据类型用 Long 代替	LongWord/ Cardinal
I64	有符号 64 位整型数	-9223372036854775808 to 9223372036854775807	__int64		Int64
U64	无符号 64 位整型数	0 to 18446744073709551615	unsigned __int64		无此数据类型用 Int64 代替
F32	32 位单精度浮点数	-3.402823E38 to 3.402823E38	float	Single	Single
F64	64 位双精度浮点数	-1.797683134862315E308 to 1.797683134862315E309	double	Double	Double
F64L	64 位多精度浮点数	1.189731495357231765E+4932 to 3.362103143112093506 3E-4932	long double		Extnded

1.2.2 Visual C++扩展数据类型

Visual C++基本数据类型	Visual C++扩展数据类型	Visual C++扩展指针类型
char	CHAR	PCHAR
unsigned char	UCHAR/BYTE	PUCHAR/PBYTE
short	SHORT	PSHORT
unsigned short	WORD/USHORT	PUSHORT/PWORD
int	long/LONG/ INT/BOOL	PLONG/PINT/PBOOL
unsigned long	ULONG	PULONG
float	FLOAT	PFLOAT
double	无	无

1.2.3 布尔变量数据类型

编程语言类型	布尔变量命名	字节数
Visual C++	bool	1
	BOOL	4
Visual Basic	Boolean	2(-1=真；0=假)
C++Builder	BOOL	4
Delphi	Boolean, ByteBool	1
	WordBool	2
	BOOL, LongBool	4

1.3 特别约定

为简化文字，同时又为了体现阿尔泰公司所注重的标准化、重用化、人性化、可扩展化，尽可能的延伸后续设计，保护用户的前期投资，便将文档中的产品标识前缀名“USB5630_”省略掉，只保留其产品统一的功能群组名和动作名称，如“DEV_Create”、“AI_InitTask”等关键部分，尽可能让用户看到的就是最关心的功能部分，且只要功能一样，那么其命名形式、参数形式、参数取值也尽可能一样。但在实际的头文件和代码中此前缀是不能省略掉的。

凡是以“b”为前缀的变量或参数，均表示布尔型（bool），其取值总是为 TRUE 或 FALSE(即 1 或 0)；

凡是以“n”为前缀的变量或参数，均表示整型(integer)，包括 8 位、16 位、32 位、64 位有符号数和无符号数。（由于整型变量最普通，因此如果没有标注前缀的变量或参数，通常可以理解为整型）；

凡是以“f”为前缀的变量或参数，均表示浮点型(float/double)；

凡是变量或参数中带有“Buffer”或“Buf”等字样的，均表示为缓冲或数组或指针(指针必须指向有一定长度的连续内存空间)。

2 使用提要

2.1 驱动函数的导入方法

为了用户在演示工程中或开发工程中更明确的看出驱动头文件的信息，所有语言的驱动头文件都是以产品名称为基本名，以各种语言的相关特征为扩展名。如下表：

语言	函数接口头文件	函数接口导入库	默认所在安装位置
Microsoft Visual C++	USB5630.h	USB5630.lib	C:\ART\USB5630\Include
Microsoft Visual Basic	USB5630.bas	无	C:\ART\USB5630\Include
Borland C++ Builder	USB5630.h	USB5630.lib	C:\ART\USB5630\Include
Borland Delphi	USB5630.pas	无	C:\ART\USB5630\Include
NI LabVIEW	USB5630.vi	无	C:\ART\USB5630\Include
Microsoft Visual C++	USB5630.h	USB5630.lib	C:\ART\USB5630\Include

注：（1）、USB5630.h 是产品的最基础的头文件，强烈建议用户关注和使用该头文件中的函数接口以实现 AI、CTR、DIO 等功能。

（2）、USB5630RSV.h 是产品的保留头文件，为了凸现 USB5630.h 中关键函数的基础功能和保证用户在使用主要函数接口的简单易用性，阿尔泰不对保留头文件（RSV）中的函数作专门的文字型说明和售后的技术支持。

2.2 产品二次发布

如果用户使用阿尔泰公司的某款产品已做好了应用系统的开发，准备向市场发布，那么用户需要做的部分工作有：

（1）、将 USB5630.dll 从 Windows\System32 中复制到安装包中；

（2）、将 USB5630.inf、USB5630.sys 从安装光盘相应产品文件夹下的 Driver 中复制到安装盘中。

2.3 管理设备

阿尔泰的设备驱动程序采用的是面向对象编程技术，通过调用 DEV_Create()函数可以创建无限多个设备对象的实例，并返回与设备实例关联的对象句柄 hDevice。有了这个句柄，用户就拥有了对该设备开放功能的所有控制权。如 AI_InitTask()使用 hDevice 句柄初始化 AI 工作参数，DIO_ReadPort()函数可用实现数字量的端口数据的读取等。最后通过 DEV_Release()函数将 hDevice 释放掉。

2.4 AI 单点采样模式

单点采样，就是在一次启动后，用户每次发出读命令 AI_ReadAnalog()或 AI_ReadBinary()时 AI 以设备最快速度获取各采样通道单个点的数据。具体步骤如下：

(1) DEV_Create() 创建设备句柄；

(2) AI_InitTask() 初始化 AI 采样任务，参数 nSampleMode=0；

(3) AI_StartTask() 启动 AI 采样任务；

(4) AI_ReadAnalog() 或者 AI_ReadBinary() 读取 AI 各采样通道单点数据；

(5) AI_StopTask() 停止 AI 采样任务；

(6) AI_ReleaseTask() 释放 AI 采样任务；

(7) DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在第 4 步处循环进行，即可实现单点实时循环采样；

如果每次采集参数有所不同，则可以在 2、3、4、5、6 步间重复进行。

具体执行流程请看下面的图 2-4-1。

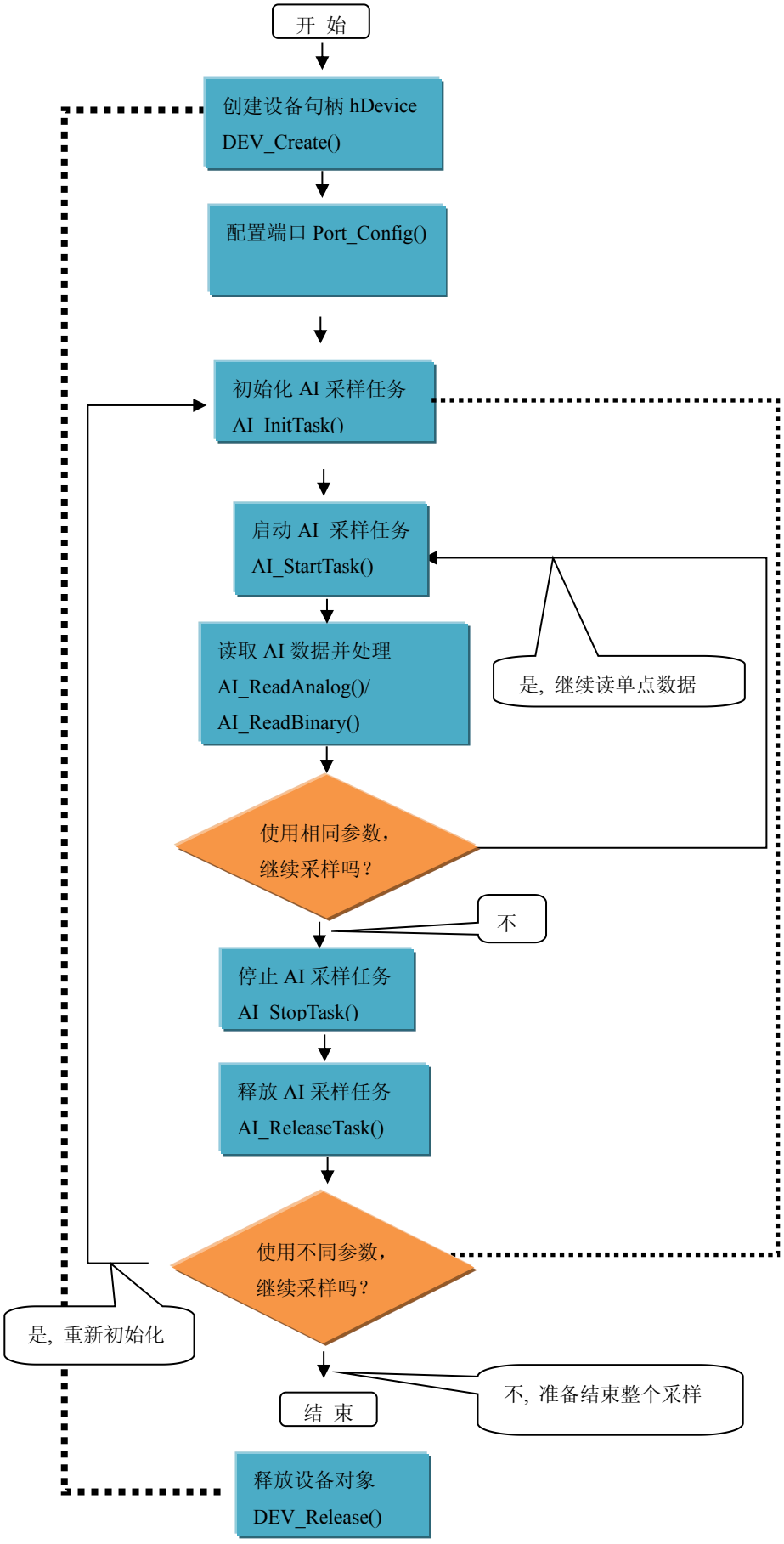


图 2-4-1　AI 实时单点采样流程

2.5 AI 有限点采样模式

有限点采样模式，就是在一次启动后，AI 按照设定的采样速率，触发条件进行指定时间的或指定点数的连续采样，达到指定点数后设备会自动停止。且在采样结束后才可以读取到完整的数据片断。

AI 有限点采样流程：

(1) DEV_Create()　创建设备句柄；

(2) AI_InitTask()　初始化 AI 采样任务, 注意参数 nSampleMode=3；

(3) AI_StartTask()　启动 AI 采样任务；

(4) AI_ReadAnalog() 或者 AI_ReadBinary() 读取 AI 数据（注意指定适当的超时等待时间）；

(5) AI_StopTask()　停止 AI 采样任务；

(6) AI_ReleaseTask()　释放 AI 采样任务；

(7) DEV_Release() 释放设备句柄。

如果在采集参数相同的情况下，多次捕捉触发事件采样多个片断的数据，可以在 3、4、5 步间循环进行，即可实现高速多次跟踪多个触发事件；

如果每次采集参数有所不同，则可以在 2、3、4、5、6 步间重复进行。请参考看下面流程图 2-5-1。

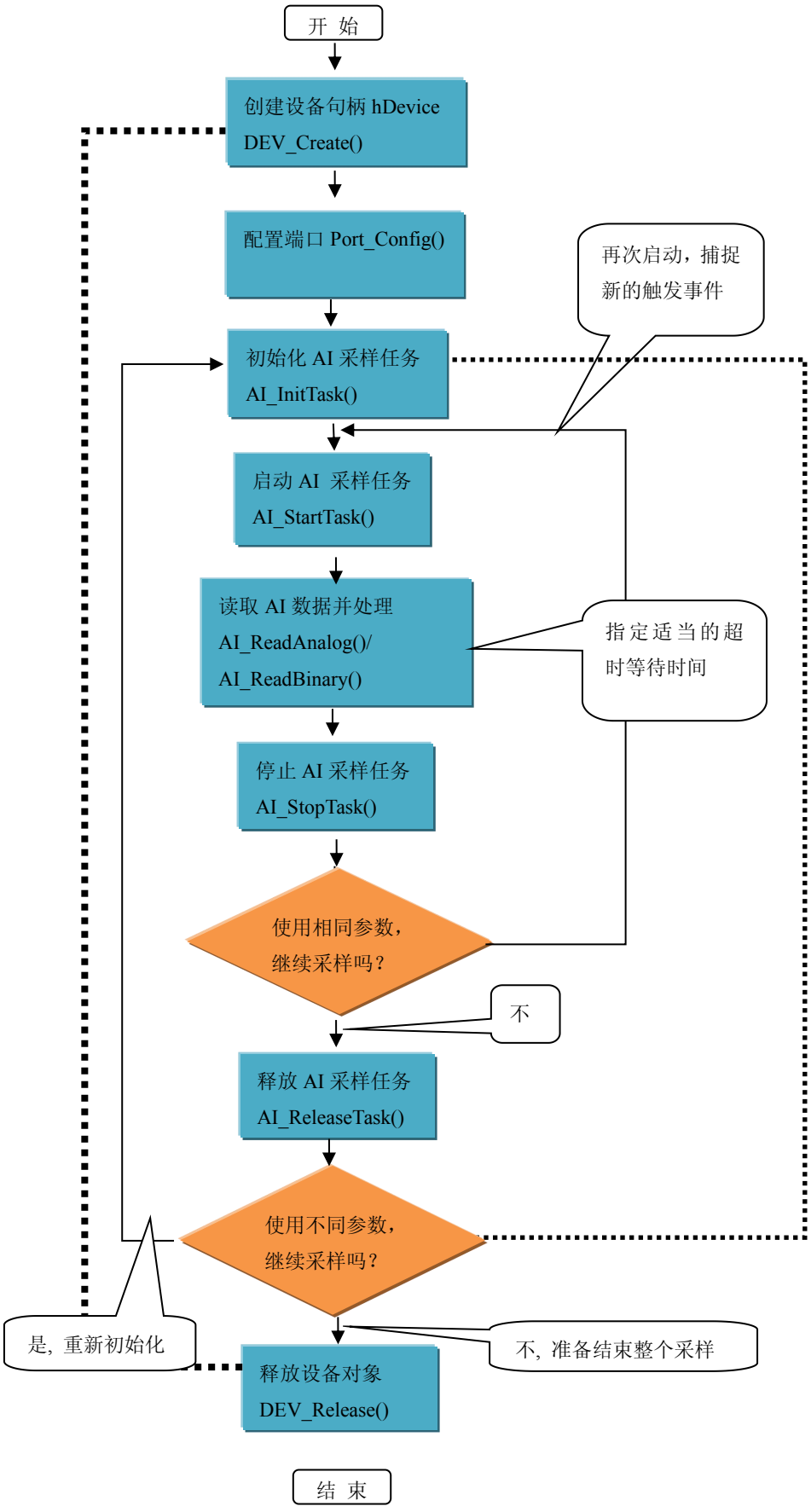


图 2-5-1　AI 有限点采样流程图例

2.6 AI 连续采样模式

连续采样模式，就是在一次启动后，AI 按照设定的采样速率，触发条件进行长时间的，无限点的连续不间断采样，设备永远不会自动停止（除非用户手动强制停止）。且在采样过程中可以实时读取采样的连续数据。

AI 连续采样流程：

（1）DEV_Create()　创建设备句柄；

（2）AI_InitTask()　初始化 AI 采样任务，注意参数 nSampleMode=3；

（3）AI_StartTask()　启动 AI 采样任务；

（4）AI_ReadAnalog() 或者 AI_ReadBinary() 读取 AI 数据（注意指定适当的超时等待时间）；

（5）AI_StopTask()　停止 AI 采样任务；

（6）AI_ReleaseTask() 释放 AI 采样任务；

（7）DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在 4 步间循环进行，即可实现高速连续不间断采样；

如果是在参数不变的情况下需要捕获新的触发时，可以在 3、4、5 之间循环进行；

如果每次采集参数有所不同，则可以在 2、3、4、5、6 步间重复进行。请参考看下面流程图 2-6-1。

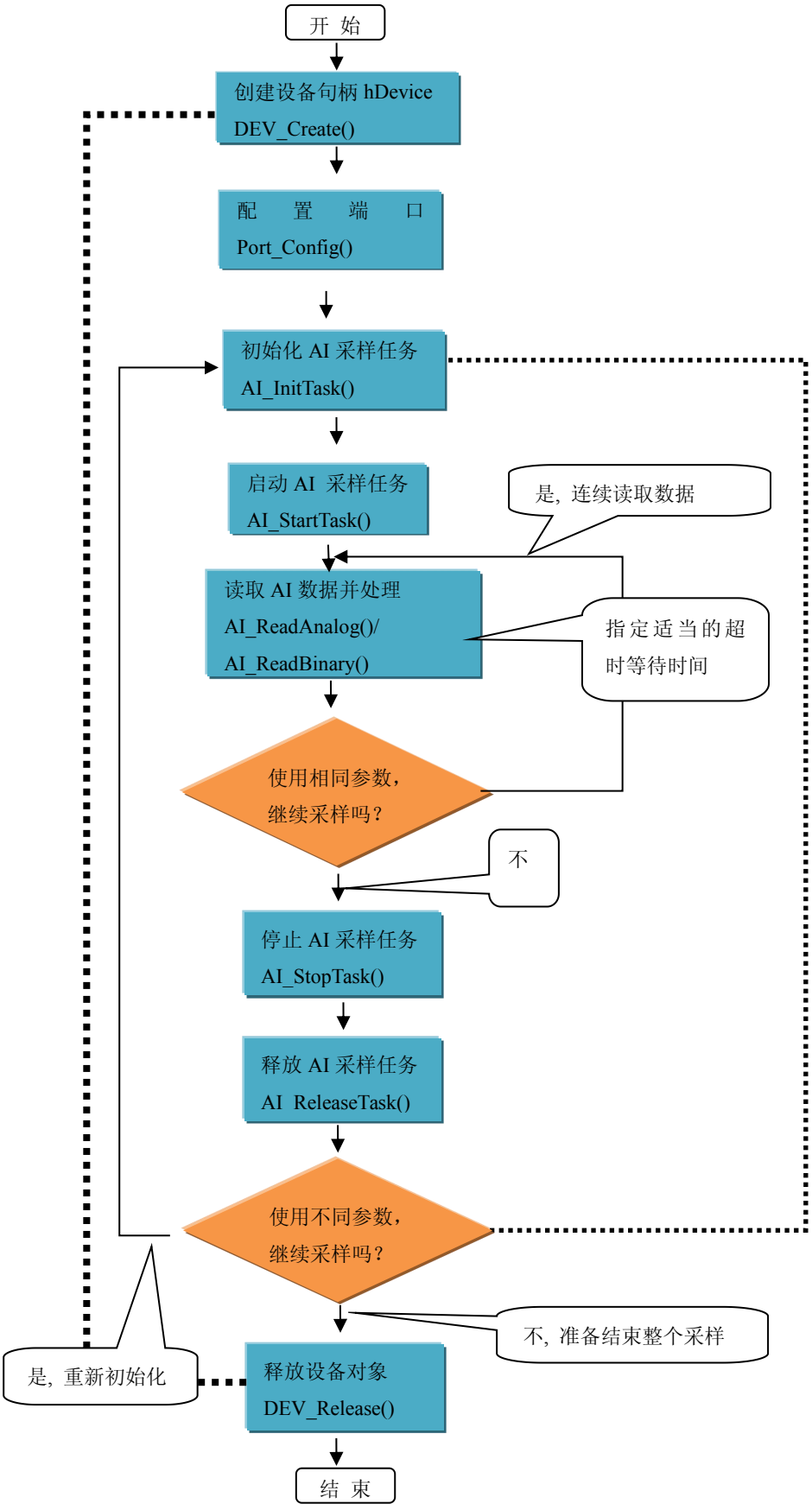


图 2-6-1　AI 连续采样流程图例

上面图 2-4-1、图 2-5-1、图 2-6-1 中虚线表示对称关系。较粗的虚线表示 DEV_Create()和 DEV_Release()两个函数的对称关系是：最初执行一次 DEV_Create()，在结束时就须执行一次 DEV_Release() (但并不是说只有 DEV_Release()后才能再次 DEV_Create()，因为阿尔泰的驱动程序是可以重入的)。而较细的虚线则表示 AI_InitTask()和 AI_ReleaseTask()两个函数的对称关系（即只有在 AI_ReleaseTask()之后才能再次 AI_InitTask()。

2.7 AO 单点采样模式

单点采样，就是在一次启动后，用户每次发出写命令 AO_WriteAnalog()或 AO_WriteBinary()时 AO 以设备最快速度写入各采样通道单个点的数据。具体步骤如下：

（1） DEV_Create() 创建设备句柄；

（2） AO_InitTask() 初始化 AO 采样任务，参数 nSampleMode=0；

（3） AO_StartTask() 启动 AO 采样任务；

（4） AO_WriteAnalog()或者 AO_WriteBinary() 写入 AO 各采样通道单点数据；

（5） AO_StopTask() 停止 AO 采样任务；

（6） AO_ReleaseTask() 释放 AO 采样任务；

（7） DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在第 4 步处循环进行，即可实现单点实时循环采样；

如果每次采集参数有所不同，则可以在 2、3、4、5、6 步间重复进行。

具体执行流程请看下面的图 2-7-1。

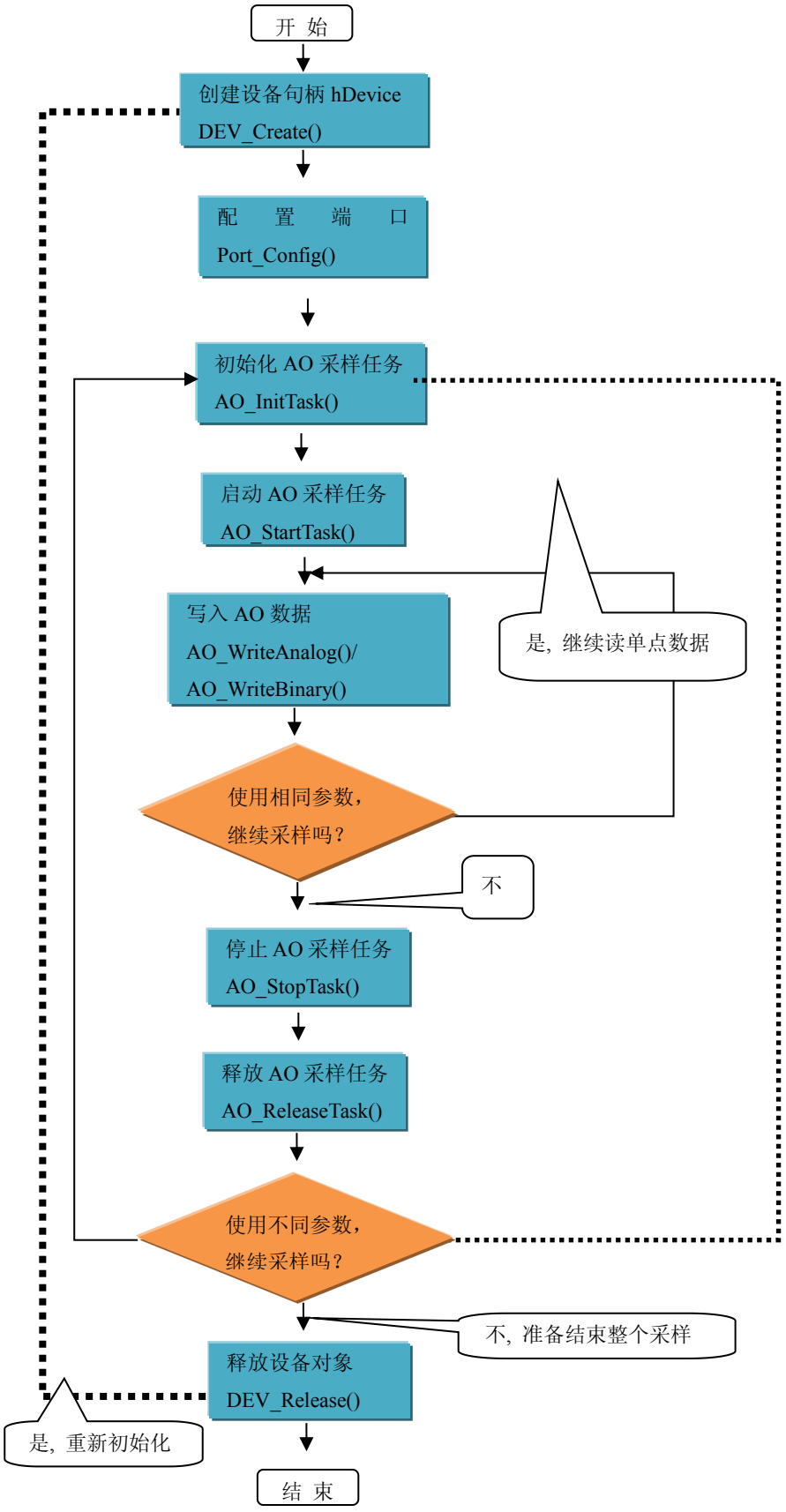


图 2-7-1　AO 实时单点采样流程

2.8 AO 有限点采样模式

有限点采样模式，就是在启动前，先写入指定点数的数据到任务中，启动后，AO 按照设定的采样速率，触发条件进行指定时间的或指定点数的连续采样，达到指定点数后设备会自动停止。且在采样结束后才可以写入下一批待数据到任务中以待启动采样。

AO 有限点采样流程：

（1） DEV_Create() 创建设备句柄；

（2） AO_InitTask() 初始化 AO 采样任务，参数 nSampleMode=2；

（3） AO_WriteAnalog()或者 AO_WriteBinary() 写入 AO 各采样通道波形数据；

（4） AO_StartTask() 启动 AO 采样任务；

（5） AO_GetStatus() 取得 AOStatus.bTaskDone 若等于 1 表示采样任务结束（或者调用 AO_WaitUntilTaskDone()）

（6） AO_StopTask() 停止 AO 采样任务；

（6） AO_ReleaseTask() 释放 AO 采样任务；

（7） DEV_Release() 释放设备句柄。

如果在采集参数相同的情况下，多次捕捉触发事件输出多个片断的波形数据，可以在 3、4、5、6 步间循环进行，即可实现高速多次跟踪多个触发事件；

如果每次采集参数有所不同，则可以在 2、3、4、5、6 步间重复进行。请参考看下面流程图 2-8-1。

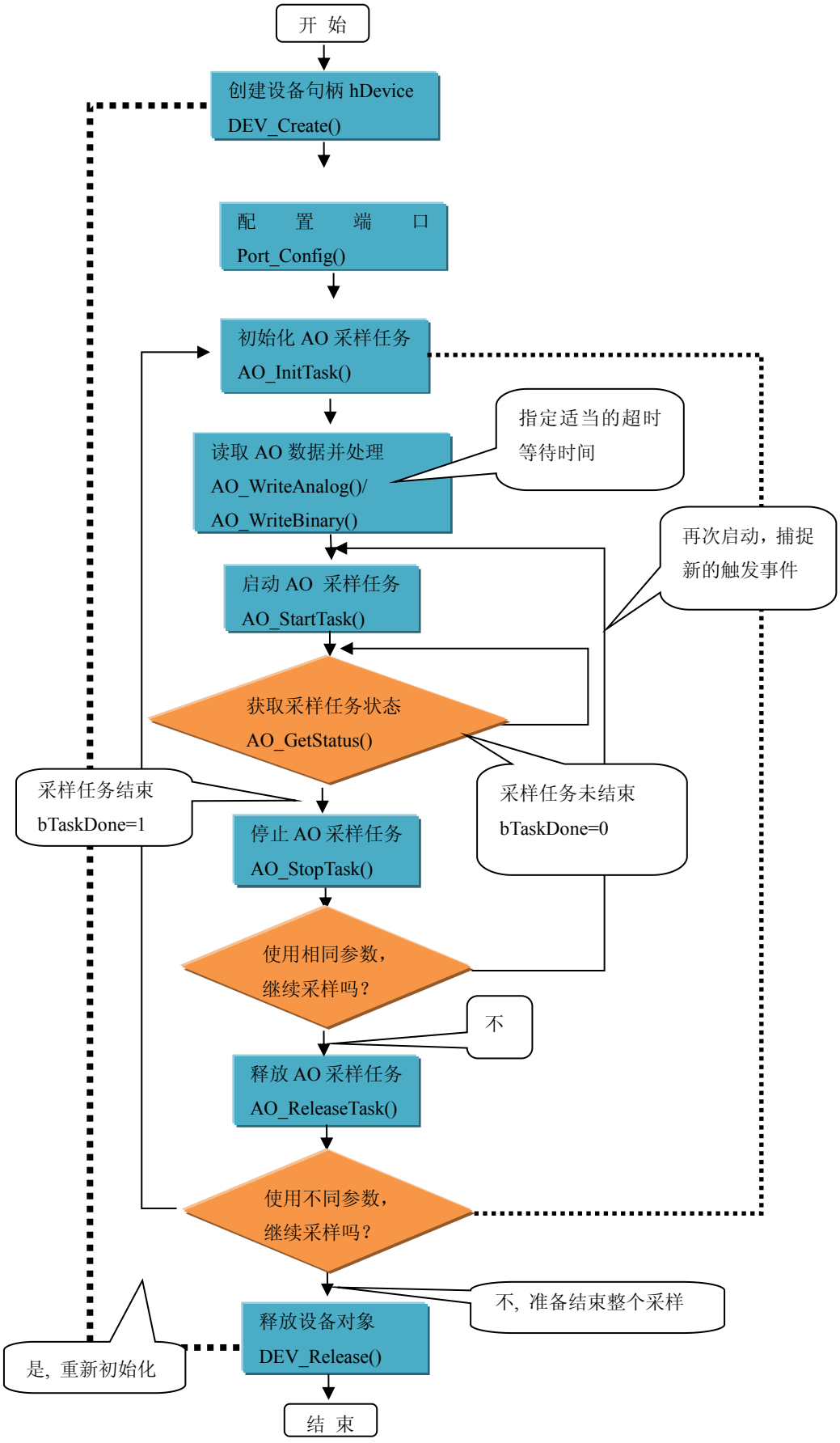


图 2-8-1 AO 有限点采样流程图例（AO_GetStatus()同步）

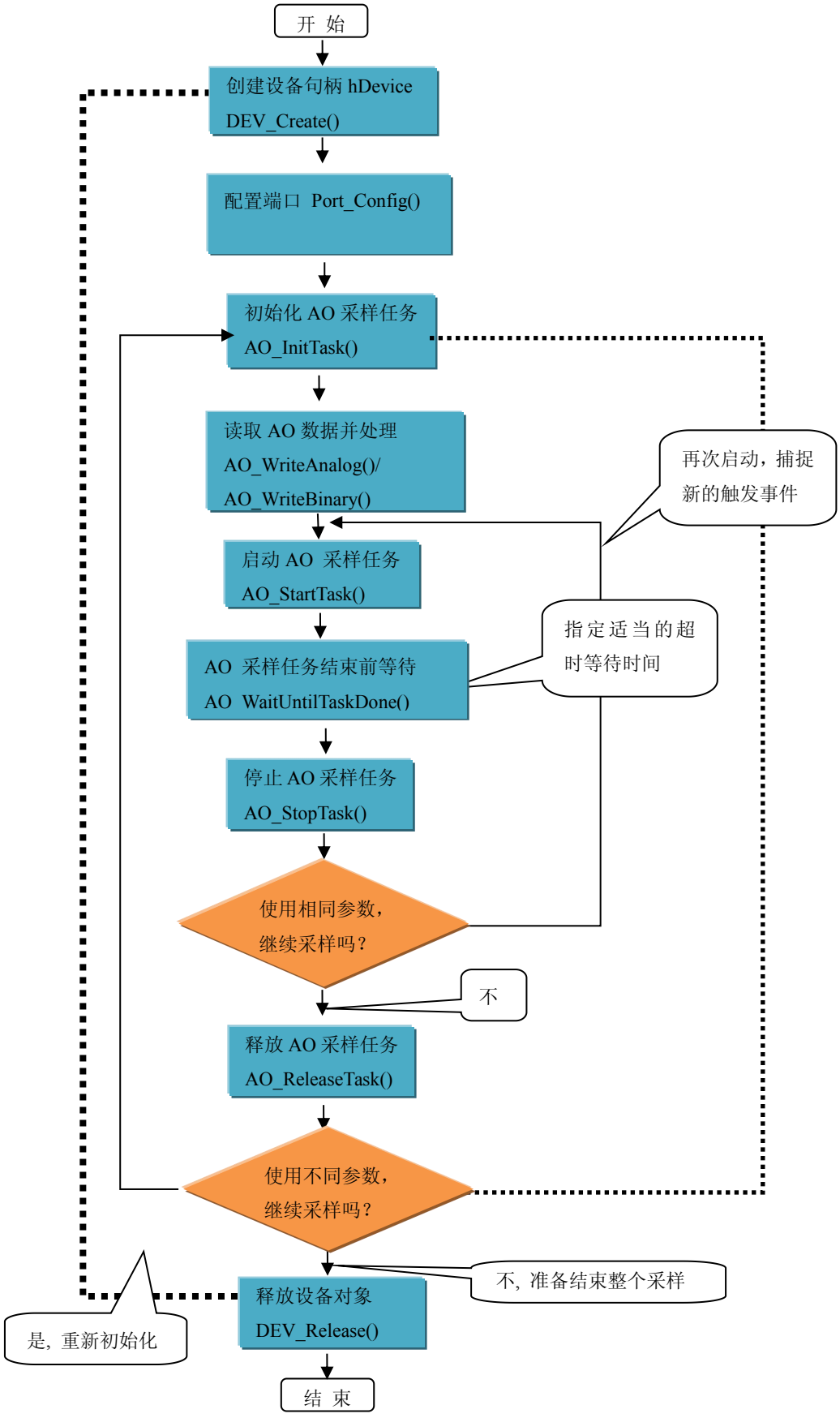


图 2-8-2　AO 有限点采样流程图例（AO_WaitUntilTaskDone ()同步）

2.9 AO 连续采样模式

连续采样模式，就是在一次启动后，AO 按照设定的采样速率，触发条件进行长时间的，无限点的连续不间断采样，设备永远不会自动停止（除非用户手动强制停止）。且在采样过程中可以实时改变输出波形数据。

AO 连续采样流程(重生成模式):

(1) DEV_Create() 创建设备句柄；

(2) AO_InitTask() 初始化 AO 采样任务，参数 nSampleMode=3；bRegenModeEn=TRUE；

(3) AO_WriteAnalog()或者 AO_WriteBinary() 写入 AO 各采样通道波形数据；

(4) AO_StartTask() 启动 AO 采样任务；

(5) AO_GetStatus() 取得 AO 采样任务的各种状态或者处理其他事务（采样任务后台输出连续数据）

(6) AO_StopTask() 停止 AO 采样任务；

(7) AO_ReleaseTask() 释放 AO 采样任务；

(8) DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在 5 步跟踪采样任务状态或处理相关事务，采样任务在后台连续不间断输出数据；

如果是在参数不变的情况下需要捕获新的触发时，可以在 4、5、6 之间循环进行；

如果每次采集参数有所不同，则可以在 2、3、4、5、6、7 步间重复进行。请参考看下面流程图 2-9-1。

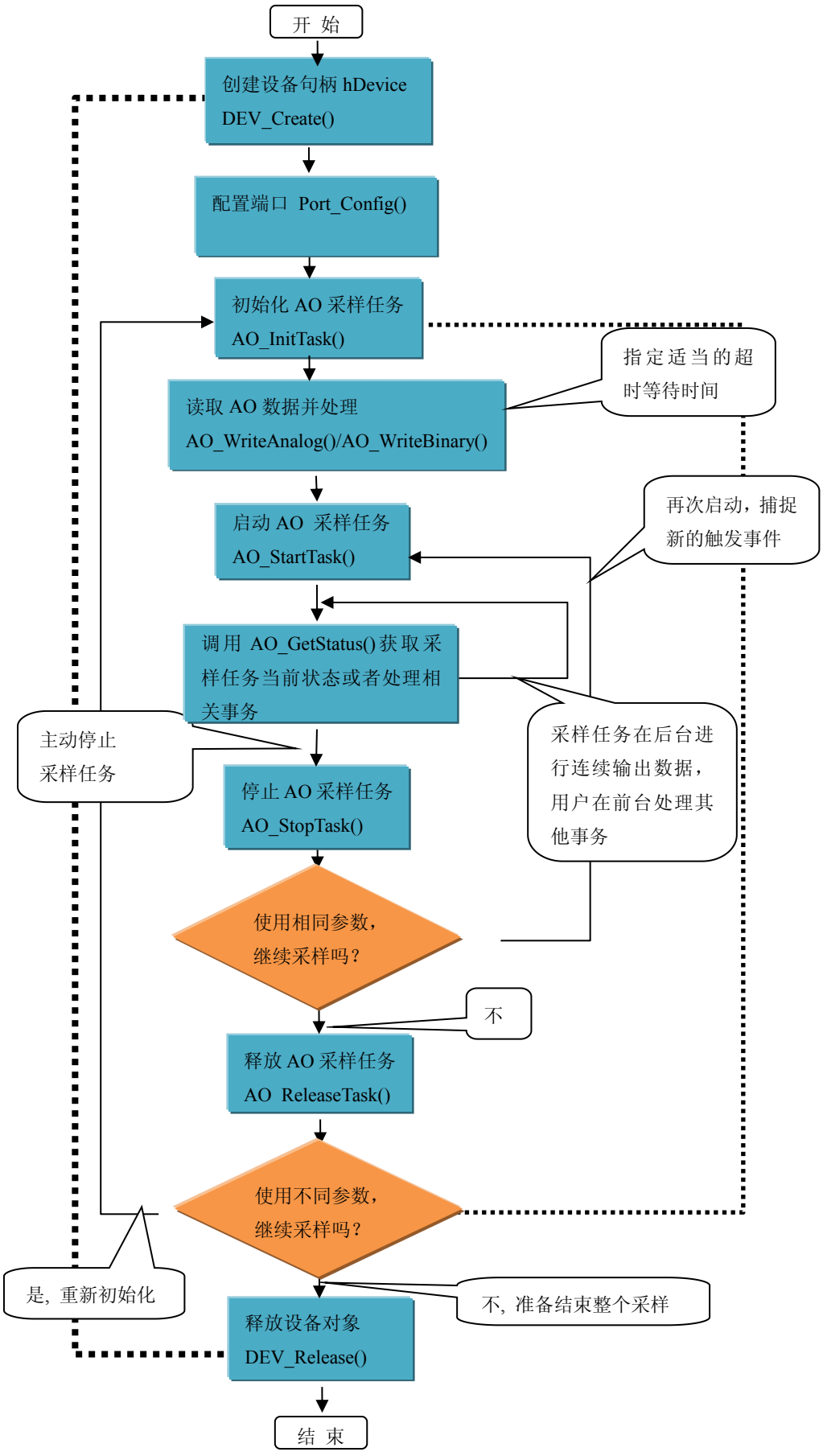


图 2-9-1 AO 连续采样流程图例（重生成模式）

AO 连续采样流程(非重生成模式):

(1) DEV_Create() 创建设备句柄;

(2) AO_InitTask() 初始化 AO 采样任务, 参数 nSampleMode=3; bRegenModeEn=FALSE;

(3) AO_WriteAnalog()或者 AO_WriteBinary() 写入 AO 各采样通道波形数据;

(4) AO_StartTask() 启动 AO 采样任务;

(5) AO_WriteAnalog()或者 AO_WriteBinary() 及时连续写入后续波形数据到采样任务中

(6) AO_StopTask() 停止 AO 采样任务;

(7) AO_ReleaseTask() 释放 AO 采样任务;

(8) DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在 5 步及时连续写入后续波形数据到采样任务中（注意要及时迅速，否则会出现缓冲下溢的风险，从而造成断波现象）;

如果是在参数不变的情况下需要捕获新的触发时，可以在 3、4、5、6 之间循环进行;

如果每次采集参数有所不同，则可以在 2、3、4、5、6、7 步间重复进行。请参考看下面流程图 2-9-2。

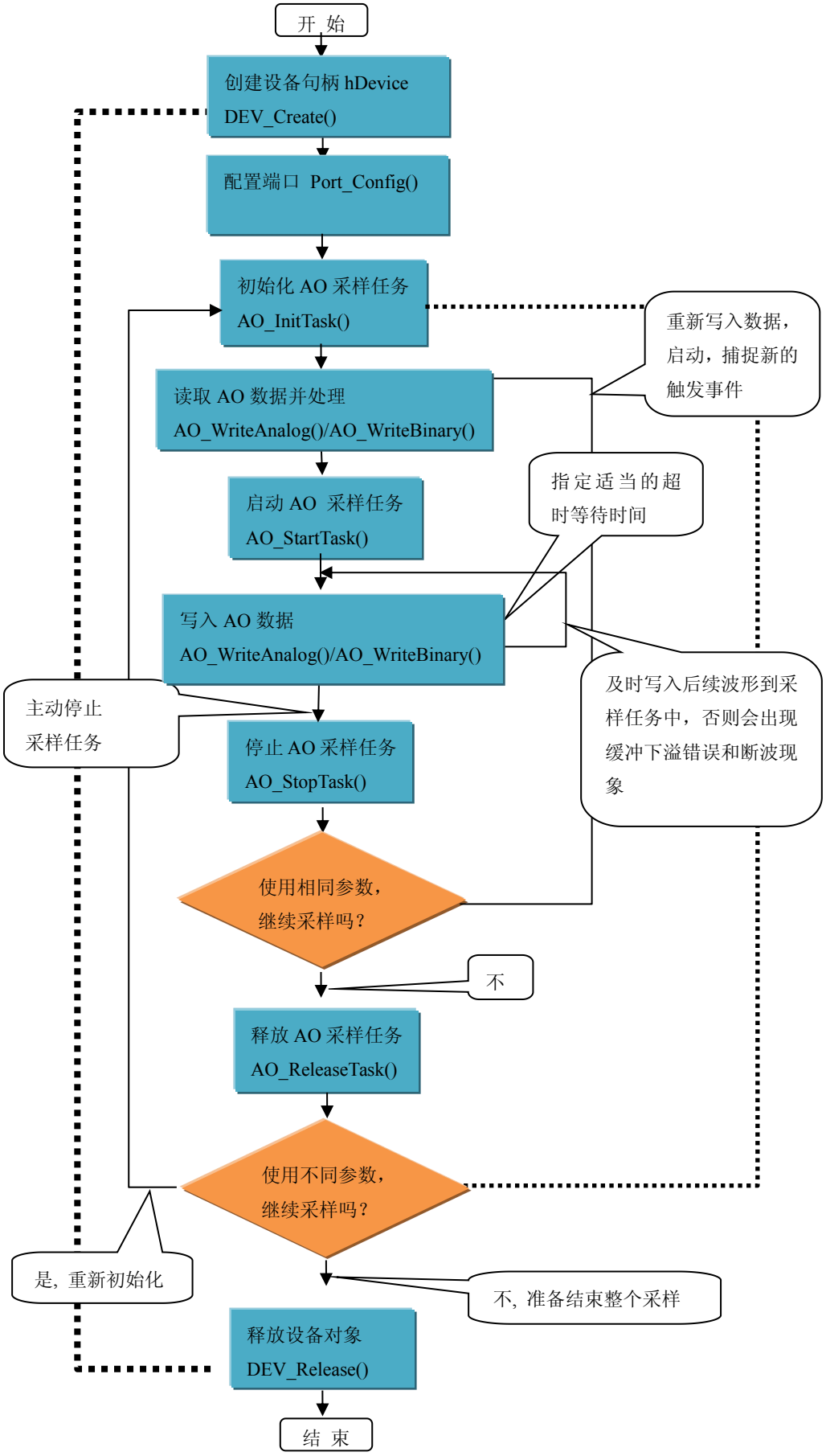


图 2-9-2　　AO 连续采样流程图例（非重生成模式）

上面图 2-7-1、图 2-8-1、图 2-9-1 中虚线表示对称关系。较粗的虚线表示 DEV_Create()和 DEV_Release()两个函数的对称关系是：最初执行一次 DEV_Create()，在结束时就须执行一次 DEV_Release() (但并不是说只有 DEV_Release()后才能再次 DEV_Create()，因为阿尔泰的驱动程序是可以重入的)。而较细的虚线则表示 AO_InitTask()和 AO_ReleaseTask()两个函数的对称关系（即只有在 AO_ReleaseTask()之后才能再次 AO_InitTask()）。

2.10 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 DO_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。

2.11 用户开发所必须的函数

首先，不管用户购买的是什么产品，其设备对象管理函数(以“DEV”为关键字段)对用户都是必须的，特别是 DEV_Create()、DEV_Release()两个函数则是必不可少的。因为对于所有的设备访问都要有这两个函数的帮助。

3 主要功能组函数介绍

3.1 DEV 设备对象管理函数原型说明

DEV_Create()

函数原型：

Visual C++ :

```
HANDLE DEV_Create(U32 nDeviceIdx,
                  BOOL bUsePhysIdx);
```

功能：创建设备对象(Create device object)，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，用户才能顺利调用其它相关的接口函数以实现对设备的控制。

参数：

nDeviceIdx 入口参数，设备序号(Device Index)。设备序号有两种：逻辑序号(Logical Index)和物理序号(Physical Index)。逻辑序号的定义是：当向同一台计算机系统中加入若干张相同类型的 USB 卡时，驱动程序自动以逻辑号来管理每张卡。比如某台计算机系统中插入该卡共四张，则根据操作系统的加载顺序依次分配的逻辑号为 0、1、2、3。因为每个设备的逻辑号是不能事先由用户硬性决定的，而是由操作系统加载设备时的顺序决定的。而设备物理号则由可以用户事先对硬件进行配置决定的号，这个号是固定的，跟插入 USB 的顺序没有关系。究竟使用哪一种设备号，由参数 bUsePhysIdx 决定。当使用物理序号时，其取值范围为[0, 255]。

bUserPhysIdx 入口参数，是否使用物理序号，是否使用物理序号, =TRUE:使用物理序号, =FALSE:使用逻辑序号 。

返回值：如果执行成功，则返回设备对象句柄；如果执行失败，则返回错误码 INVALID_HANDLE_VALUE(或-1), 可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() DEV_GetCount() DEV_GetCurrentIdx() DEV_Release() DEV_GetSpeed()

DEV_GetCount()

函数原型：

Visual C++ :

```
int DEV_GetCount(void);
```

功能：取得该设备在系统中的总数量(Get device count)。

参数：**无**

返回值：成功返回值>0, 失败返回 0(可调用 GetLastError()分析错误原因)。此则有两种可能的情况存在：其一，设备根本不存在，致使驱动程序无法安装加载。其二、设备存在，但其驱动程序未正确安装。对于具体原因，可以在操作系统的“设备管理器”中查看是否有该设备的信息项。在返回 0 时，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() DEV_GetCount() DEV_GetCurrentIdx() DEV_Release() DEV_GetSpeed()

DEV_GetCurrentIdx()

函数原型：

Visual C++ :

BOOL DEV_GetCurrentIdx (HANDLE hDevice，
U32* pLgcIdx,
U32* pPhysIdx);

功能：取得指定设备物理号和逻辑号(Get logical and physical index of the device)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pLgcIdx 出口参数，取得设备的逻辑索引号(Logical Index)，取值范围为[0, 255]。如果=NULL 则表示忽略此参数。

pPhysIdx 出口参数，取得设备的物理索引号(Physical Index)，取值范围为[0, 255]，具体值由 hDevice 指定的硬件决定。如果=NULL 则表示忽略此参数。

返回值：如果成功，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError() 捕获错误码以确定具体原因。

相关函数： DEV_Create() DEV_GetCount() DEV_GetCurrentIdx() DEV_Release() DEV_GetSpeed()

DEV_GetSpeed()

函数原型：

Visual C++ :

BOOL DEV_GetCurrentIdx (HANDLE hDevice，
U32* pSpeed);

功能：读取设备连接的 USB 端口速度。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pSpeed USB 接口速度，=1:USB1.0, =2:USB2.0, =3:USB3.0

返回值：如果成功，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError() 捕获错误码以确定具体原因。

相关函数： DEV_Create() DEV_GetCount() DEV_GetCurrentIdx() DEV_Release()

DEV_Release()

函数原型：

Visual C++ :

BOOL DEV_Release(HANDLE hDevice)

功能：释放设备对象（Release device object）， 包括释放所占用的系统资源。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果成功，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError() 捕获错误码以确定具体原因。

相关函数： DEV_Release()

3.2 I/O 端口控制函数原型说明

Port_Config()

函数原型：

Visual C++:

BOOL Port_Config(HANDLE hDevice, PPORT_PARAM pPortParam);

功能：IO 端口配置函数，调用此函数时各功能(AI AO CTR DIO)要处于停止工作状态。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pPortParam 入口参数，端口配置结构体指针，决定了各端口功能、方向、值。

返回值：如果调用成功，则返回 TRUE， 否则返回 FALSE。

相关函数： DEV_Create()

Port_LoadParam()

函数原型：

Visual C++:

BOOL Port_LoadParam(HANDLE hDevice, PPORT_PARAM pPortParam);

功能：从 USB5630.ini 中加载 Port 参数

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pPortParam 出口参数，属于 PORT_PARAM 的结构指针类型，负责返回端口参数值

返回值：如果调用成功，则返回 TRUE， 否则返回 FALSE。

相关函数： DEV_Create()

Port_SaveParam()

函数原型：

Visual C++:

BOOL Port_SaveParam(HANDLE hDevice, PPORT_PARAM pPortParam);

功能：保存 Port 参数至 USB5630.ini

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pPortParam 入口参数，属于 PORT_PARAM 的结构指针类型，保存端口参数值。

返回值：如果调用成功，则返回 TRUE， 否则返回 FALSE。

相关函数： DEV_Create()

Port_ResetParam()

函数原型：

Visual C++:

BOOL Port_ResetParam(HANDLE hDevice, PPORT_PARAM pPortParam);

功能：将当前 Port 参数复位至出厂值

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pPortParam 出口参数，Port 参数结构指针，负责在参数被复位后返回其复位后的参数值

返回值：如果调用成功，则返回 TRUE， 否则返回 FALSE。

相关函数： DEV_Create()

3.3 AI 模拟量输入函数原型说明

AI_InitTask()

函数原型：

Visual C++:

BOOL AI_InitTask (HANDLE hDevice, PAI_PARAM pAIParam, HANDLE* pSampEvent)

功能：初始化 AI 任务参数(Initialize task parameter for analog input)，为设备操作就绪有关状态，如预置 AI 采样率、各通道模拟量采样范围等。但它并不启动 AI 设备，若要启动 AI 设备，须在成功调用此函数之后再调用 AI_StartTask()函数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIParam 入口参数，AI 工作参数结构体指针，决定了 AI 工作时的各种状态及参数，如采样率等。关于其具体定义及说明请参考《第四章 各种结构体描述》\《第一节、AI_PARAM（AI 工作参数结构）》。

pSampEvent 出口参数，事件句柄，该事件属性为自动发信号，初始状态为不发信号，它由任务自动创建。如果该参数=NULL，则视为用户不需要驱动程序触发任何事件。

该事件句柄的作用是：当任务中每通道有 AIParam.nSampsPerChan 个采样点的数据时则会触发此事件。用户可调用 WIN32 API 函数 WaitForSingleObject()来跟踪该事件。当事件未发生时，调用 WaitForSingleObject()函数的采集线程会自动阻塞(等待)状态(此时调用线程不会消耗 CPU 时间)，当事件发生时，则意味着任务中至少有 nSampsPerChan 个数据可读，则采集线程立即进入运行状态，并迅速调用 AI_ReadAnalog()或 AI_ReadBinary()循环读取任务中的数据，直至 nAvailSampsPerChan 小于 nReadSampsPerChan。

如果简单循环调用 AI_GetStatus()查询 AI 的各种状态以同步数据读操作，则会在等待中消耗许多 CPU 时间，而影响系统的整体性能。如果采用事件同步相结合的办法，则会节省大量的 CPU 时间。因为在调用 WaitForSingleObject()时，若事件未被触发，则当前线程会进入阻塞(等待)状态，而不消耗 CPU 时间，而事件一旦触发，则当前线程立即进入运行状态。总之它可以在一定程度上提升系统的整体性能，让应用程序有更多的 CPU 时间去处理采样数据或其他任务。当然最简单的办法则是使用 AI_ReadAnalog()或 AI_ReadBinary()函数的超时机制，即利用 fTimeout 参数的合理设置来同步读入操作，而无须关注和跟踪 pSampEvent 事件。

返回值：如果初始化 AI 工作参数成功，则返回 TRUE， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

AI_StartTask()

函数原型：

Visual C++ :

BOOL AI_StartTask (HANDLE hDevice)

功能：启动 AI 采集(Start task for analog input)。必须在成功调用 AI_InitTask()函数后才能调用此函数，调用该函数后 AI 立即准备就绪，但 AI 实际是否进入采样记录过程，须依赖于触发事件的产生。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，AI 立刻被启动， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

AI_SendSoftTrig()

函数原型：

Visual C++ :

BOOL AI_SendSoftTrig (HANDLE hDevice)

功能：发送软件触发事件(Send software trigger event)。设备进入等待触发状态，若用户需软件触发或需要随时手动给触发事件时，可调用此函数。该方式也叫软件触发或手动触发或内触发。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，即 AI 立刻被触发采样一次， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

AI_GetStatus()

函数原型：

Visual C++ :

BOOL AI_GetStatus (HANDLE hDevice, PAI_STATUS pAIStatus)

功能：取得 AI 的各种状态(Get status for analog input)。一旦调用 AI_StartTask()函数后，应立即调用此函数查询 AI 状态去同步采样数据的读操作。nAvailSampsPerChan>0 时，表示采集任务中至少有 1 个数据段可供读取，用户应立即调用 AI_ReadBinary()或 AI_ReadAnalog()函数循环读取若干可读段数据，直到 nAvailSampsPerChan>nReadSampsPerChan 时可调用读数函数。如果在启动采样任务后，设备迟迟不能被触发采样，可以根据需要调用 AI_SendSoftTrig()函数以强制触发设备采样，便可以很快得到有效的可读采样数据。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIStatus 出口参数，设备状态参数结构体，它返回设备当前的各种状态，如是否完成采样、是否已被触发等信息。关于具体状态信息请参考《第四章 各种结构体描述》\《第二节、AI_STATUS（AI 状态信息结构）》。

返回值：如果成功获取 AI 状态，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

AI_WaitUntilTaskDone()

函数原型：

Visual C++ :

```
BOOL AI_WaitUntilTaskDone (HANDLE hDevice, F64 fTimeout)
```

功能：在 AI 的采样任务结束前等待(Wait until task done for analog input)。一旦调用 AI_StartTask()函数后，可以调用此函数等待采样任务结束。它通常用在有限点采样模式中。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

fTimeout 入口参数，超时时间，单位：秒（S）。指定该次等待所用时间，比如设定为 10.0，即 10 秒钟的时间，如果在 10 秒内采样任务结束，则函数立即返回 TRUE，否则 10 秒钟后函数返回值 FALSE，如果采样速率极慢或触发事件长时间都不能达到的情况下，建议该超时时间应足够长；如果想禁止超时返回（即总是等到采样任务结束才返回）则赋值小于 0 即可，如-1.0；如果 fTimeout=0.0，则意味着该函数只是简单查询采样任务是否结束，如果采样任务结束了，则返回 TRUE，否则返回 FALSE。

返回值：如果采样任务结束，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

AI_ReadAnalog()

函数原型:

Visual C++ :

```
LONG AI_ReadAnalog(HANDLE hDevice,
                   F64 fAnlgArray[],
                   U32 nReadSampsPerChan,
                   U32* pSampsPerChanRead,
                   U32* pAvailSampsPerChan,
                   F64 fTimeout);
```

功能：读取模拟量采样数据(主要是电压数据)（Read analog data from the task）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

fAnlgArray 出口参数，用户缓冲区，用于接收所有采样通道模拟量数据，值区间由相应采样通道的选用采样范围决定，单位: 伏(V)，数据类型为双精度浮点。各个采样通道的数据点是依次交替排列的。它被开辟的点空间不能小于 nReadSampsPerChan*AIParam.nSampChanCount。如果选择的不是 1 倍增益挡位，调用 AI_GetGainInfo() 获得相应增益挡位下的放大倍数 fAmpFactor，然后将该函数读取到的电压值再除以 fAmpFactor 后才是实际信号的测量结果。

nReadSampsPerChan 入口参数，每通道请求读入的数据点数。

在有限点和连续采样模式中，它指定该次从设备的当前可读数据位置读取的数据点数（单位：点）。注意此参数的值如大于当前的可读数点 nAvailSampsPerChan 则会继续等待直到至少有 nReadSampsPerChan 个点可读后读函数才会返回。等待期间，如果所等时间超过 fTimeout 指定时间也会返回，并置超时错误码。

在连续采样过程中，如果要保持连续不丢点，此参数应尽可能接近于甚至等于当前的可读点数(nAvailSampsPerChan),但不能大于 AIParam.nSampsPerChan。当然此参数值也不能大于 fAnlgArray 的缓冲区长度，所以为避免出错，所开辟的缓冲区不能小于 nReadSampsPerChan*AIParam.nSampChanCount。

pSampsPerChanRead 出口参数，返回每通道实际读取的点数。在单点采样模式中，如果读取成功，返回的每通道已读取点数总是为 1。注意每次都要检查 pSampsPerChanRead 的返回值，如果返回值等于 0，则要慎重处理。

pAvailSampsPerChan 出口参数，返回该次读操作完成时的每通道还可读而未读的数据点数。它跟 AI_GetStatus()函数取得的状态信息 AIStatus.nAvailSampsPerChan 是同一个状态信息。返回可读点数的意义在于通过数据读操作直接提供给用户，避免再次调用 AI_GetStatus()函数，即在读数据函数返回时判断可读点数，若大于 nReadSampsPerChan，则可紧接着再次调用读数据函数，直到 pAvailSampsPerChan 返回值小于 nReadSampsPerChan。在单点采样模式中,它的返回值总是为 0。

fTimeout 入口参数，超时时间，单位：秒（S）。指定等待写入额定点数的时间。比如设定为 10.0，如果在 10 秒内读取的点数达到 nReadSampsPerChan 时立即返回 TRUE，否则 10 秒钟后函数返回 FALSE，并通过 pAvailSampsPerChan 告之实际读入的点数，如果采样速率极慢或触发事件长时间都不能达到的情况下，建议该超时时间应足够长；如果想禁止超时返回（即等待请求数据点数完全从任务中读到才返回）则置为负数，如-1.0 即可。如果 fTimeout=0.0，则意味着该函数仅简单判断能否立即读取到请求的点数，如果不能，则不等待，立即返回 FALSE，否则返回 TRUE。

返回值： 如果函数调用成功则返回 TRUE，否则返回 FALSE，可以调用 Win32 API 函数 GetLastError()以取得进一步的错误码信息 nErrorCode，其定义如下表。

常量名	常量值	功能定义
ERROR_NO_AVAILABLE_SAMPS	0xE0000000+1	无有效点数据
ERROR_SAMPLE_TASK_FAIL	0xE0000000+2	采样任务失败
ERROR_TIMEOUT	1460L	超时错误(见微软定义 WinError.h)

相关函数： DEV_Create() AI_InitTask() AI_StartTask() AI_SendSoftTrig() AI_GetStatus() AI_WaitUntilTaskDone() AI_ReadAnalog() AI_ReadBinary() AI_StopTask() AI_ReleaseTask() DEV_Release()

AI_ReadBinary()

函数原型:

Visual C++ :

```
LONG AI_ReadBinary(HANDLE hDevice,
                   U16 nBinArray[],
                   U32 nReadSampsPerChan,
                   U32* pSampsPerChanRead,
                   U32* pAvailSampsPerChan,
                   F64 fTimeout);
```

功能：读取二进制原码采样数据（Read binary data from the task）。该函数不对采样结果进行物理量的换算，它是设备采样后的直接结果。二进制原码具有数据紧凑、数据量小、传输快、处理快、存盘也快的特点

参数：

hDevice 同 AI_ReadAnalog()。

nBinArray 出口参数，用户缓冲区，用于接收所有采样通道二进制原码数据，值区间[0，65535]。各个采样通道的数据点是依次交替排列的。它点空间不能小于nReadSampsPerChan*AIParam.nSampChanCount。如果选择的不是 1 倍增益挡位，那么还得调用 AI_GetGainInfo() 获得相应增益挡位下的放大倍数 fAmpFactor，然后将该函数读取到的原码数据再除以 fAmpFactor 后才是实际信号的测量结果。

nReadSampsPerChan 同 AI_ReadAnalog()。

pSampsPerChanRead 同 AI_ReadAnalog()。

pAvailSampsPerChan 同 AI_ReadAnalog()。

fTimeout 同 AI_ReadAnalog()

返回值：同 AI_ReadAnalog()

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

关于实时连续采样调用流程的图例请参考《第二章 使用提要》中《第四节、实现 AI 实时连续采样 》相关部分。

该函数读取的是原码数据，如果换算成电压值后，还得考虑所选择的增益挡位是多少？如果选择的不是 1 倍增益挡位，那么还得调用 AI_GetGainInfo() 获得相应增益挡位下的放大倍数 fAmpFactor，然后将之前换算后的电压值再除以 fAmpFactor 后才是实际信号的测量结果。

AI_StopTask()

函数原型：

Visual C++:

BOOL AI_StopTask (HANDLE hDevice)

功能：停止 AI 采样(Stop task for analog input)。必须在成功调用 AI_StartTask()函数后才能调用此函数。该函数除了停止 AI 采样外不改变设备的其他状态。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，且 AI 立刻停止转换， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AI_InitTask()	AI_StartTask()
AI_SendSoftTrig()	AI_GetStatus()	AI_WaitUntilTaskDone()
AI_ReadAnalog()	AI_ReadBinary()	AI_StopTask()
AI_ReleaseTask()	DEV_Release()	

AI_ReleaseTask()

函数原型：

Visual C++:

BOOL AI_ReleaseTask (HANDLE hDevice)

功能：释放 AI(Release task for analog input)。必须在重新调用 AI_InitTask() 函数之前被调用一次，即该函数必须和 AI_InitTask() 成对出现。注意此函数在内部首先执行 AI_StopTask()函数停止 AI 采集后，才释放被占用的 AI 资源。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AI_InitTask()　AI_StartTask()　AI_SendSoftTrig()　AI_GetStatus()　AI_WaitUntilTaskDone()　AI_ReadAnalog()　AI_ReadBinary()　AI_StopTask()　AI_ReleaseTask()　DEV_Release()

AI_GetMainInfo()

函数原型：

Visual C++ :

```
BOOL AI_GetMainInfo (HANDLE hDevice,
                     PAI_MAIN_INFO pMainInfo)
```

功能：取得 AI 功能的主要信息，如通道数、分辨率等（Get main information for analog input）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pMainInfo 出口参数，AI 主要信息结构指针，负责返回 AI 的主要描述信息。关于 AI_MAIN_INFO 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头文件，也可参考本文《各种结构体描述》关于该结构的有关说明。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AI_ScaleBinToVolt()　AI_ScaleVoltToBin()　AI_GetMainInfo()　AI_GetVoltRangeInfo()　AI_GetVoltGainInfo()　AI_GetRateInfo()　DEV_Release()

AI_GetVoltRangeInfo()

函数原型：

Visual C++:

```
BOOL AI_GetRangeInfo(HANDLE hDevice,
                     U32 nChannel,
                     U32   nSampleRange,
                     PAI_VOLT_RANGE_INFO pRangeInfo)
```

功能：取得 AI 指定通道、指定采样范围档的上下限电压、幅度等信息（Get range information for analog input）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

nSampleRange 入口参数，AI 采样范围，取值范围为[0,3]。同 AIParam.nSampleRange。

pRangeInfo 出口参数，AI 采样范围信息，负责返回 AI 的采样范围大值、最小值、幅度、编码宽度等。详情请参考《 第四节、AI_SAMP_RANGE_INFO (AI 采样范围信息结构体) 》

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AI_ScaleBinToVolt() AI_ScaleVoltToBin() AI_GetMainInfo() AI_GetVoltRangeInfo() AI_GetVoltGainInfo() AI_GetRateInfo() DEV_Release()

AI_GetVoltGainInfo()

函数原型：

Visual C++ :

```
BOOL AI_GetGainInfo(HANDLE hDevice,
                    U32 nChannel,
                    U32 nSampleGain,
                    PAI_VOLT_GAIN_INFO pGainInfo)
```

功能:取得 AI 指定通道、指定采样增益的放大倍数等信息（Get gain information for analog input）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 入口参数，AI 通道号。

nSampleGain 入口参数，AI 采样增益挡位索引号，同 AIParam.CHParam[n].nSampleGain。

pGainInfo 出口参数，AI 采样增益信息，负责返回 AI 的采样增益的放大倍数等信息。详情请参考《 第五节、AI_SAMP_GAIN_INFO (AI 采样增益信息结构体) 》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AI_ScaleBinToVolt() AI_ScaleVoltToBin() AI_GetMainInfo() AI_GetVoltRangeInfo() AI_GetVoltGainInfo() AI_GetRateInfo() DEV_Release()

AI_GetRateInfo()

函数原型：

Visual C++ :

```
BOOL AI_GetRateInfo(HANDLE hDevice, PAI_SAMP_RATE_INFO pRateInfo);
```

功能：获得采样速率信息（Get rate information for analog input）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pRateInfo 出口参数，AI 采样速率信息，负责返回 AI 的最大采样率，最小采样率等。详情请参考《 第五节、AI_RATE_INFO (AI 采样率信息结构体) 》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AI_ScaleBinToVolt() AI_ScaleVoltToBin() AI_GetMainInfo() AI_GetVoltRangeInfo() AI_GetVoltGainInfo() AI_GetRateInfo() DEV_Release()

AI_ScaleBinToVolt()

函数原型：

Visual C++ :

```
BOOL AI_ScaleBinToVolt(PAI_VOLT_RANGE_INFO pRangeInfo,
                       PAI_VOLT_GAIN_INFO pGainInfo
                       F64 fVoltArray[],
                       U16 nBinArray[],
                       U32 nScaleSamps,
                       U32* pSampsScaled);
```

功能: 将二进制原码数据转换为电压数据(Scale binary data to voltage data)。与 AI_ScaleVoltToBin()函数的功能相反。

参数：

pRangeInfo 当前转换数据需要的采样范围信息

pGainInfo 当前转换数据需要的采样增益信息(若于=NULL，表示不使用增益)

fVoltArray 用于返回量化后的电压数据，单位：伏（V），取值范围由 nSampleRange 参数选择而定。

nBinArray 用于传入待量化的原码数据，取值范围[0, 65535]。

nScaleSamps 请求量化的数据点数。

pSampsScaled 出口参数，返回实际量化后数据点数。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AI_ScaleBinToVolt() AI_ScaleVoltToBin() AI_GetMainInfo() AI_GetVoltRangeInfo() AI_GetVoltGainInfo() AI_GetRateInfo() DEV_Release()

AI_ScaleVoltToBin()

函数原型：

Visual C++ :

```
BOOL AI_ScaleVoltToBin(PAI_VOLT_RANGE_INFO pRangeInfo,
                       PAI_VOLT_GAIN_INFO pGainInfo,
                       U16 nBinArray[],
                       F64 fVoltArray[],
                       U32 nScaleSamps,
                       U32* pSampsScaled);
```

功能：将电压数据转换为原码数据(Scale voltage data to binary data)。与 AI_ScaleBinToVolt()函数的功能相反。

参数：

pRangeInfo 当前转换数据需要的采样范围信息

pGainInfo 当前转换数据需要的采样增益信息(若于=NULL，表示不使用增益)

nBinArray 出口参数，用于传入待量化的原码数据，取值范围[0, 65535]。

fVoltArray 入口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由 nSampleRange 参数选择而定。

nScaleSamps 入口参数，请求量化的数据点数。

pSampsScaled 出口参数，返回实际量化后数据点数。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AI_ScaleBinToVolt() AI_ScaleVoltToBin() AI_GetMainInfo() AI_GetVoltRangeInfo() AI_GetVoltGainInfo() AI_GetRateInfo() DEV_Release()

AI_VerifyParam()

函数原型：

Visual C++ :

```
BOOL AI_VerifyParam (HANDLE hDevice, PAI_PARAM pAIParam)
```

功能：校验 AI 工作参数(Verify task parameter for analog input)，这是一个辅助功能的函数，在初始化 AI 前，先校验 AI 参数的合法性。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIParam 入口和出口参数，AI 工作参数结构体指针，关于其具体定义及说明请参考《第四章 各种结构体描述》\《第一节、AI_PARAM（AI 工作参数结构）》。函数调用时，它传入用户要校验的各项参数，函数返回时，它将经过调整为合法的参数值返回给用户，以便后续初始化 AI 使用。

返回值：如果所有的参数均合法，则返回 TRUE； 如果有一个参数不合法，立即被调整为合法取值，并向日志文件 USB5630.log 记录不合法的参数名和原因，然后返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： AI_InitTask() AI_VerifyParam() AI_LoadParam() AI_SaveParam() AI_ResetParam()

AI_LoadParam()

函数原型：

Visual C++:

```
BOOL AI_LoadParam (HANDLE hDevice,
                   PAI_PARAM pAIParam)
```

功能：从 USB5630.ini 参数配置文件中加载硬件参数(Load parameter from USB5630.ini for analog input)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIParam 出口参数，属于 PAI_PARAM 的结构指针类型，负责返回 AI 工作参数值，关于结构指针类型 PAI_PARAM 请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头文件，也可参考本文《第一节、AI_PARAM（AI 工作参数结构体）》

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()

捕获错误码以确定具体原因。

相关函数： AI_InitTask() AI_VerifyParam() AI_LoadParam() AI_SaveParam() AI_ResetParam()

AI_SaveParam()

函数原型：

Visual C++ :

```
BOOL AI_SaveParam (HANDLE hDevice,
                   PAI_PARAM pAIParam)
```

功能：将用户配置好的 AI 工作参数保存在 USB5630.ini 参数配置文件中(Save parameter to USB5630.ini for analog input)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIParam 入口参数，AI 工作参数结构体指针，关于 AI_PARAM 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 等函数原型定义的头文件，也可参考本文《第一节、AI_PARAM（AI 工作参数结构体）》

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： AI_InitTask() AI_VerifyParam() AI_LoadParam() AI_SaveParam() AI_ResetParam()

AI_ResetParam()

函数原型：

Visual C++ :

```
BOOL AI_ResetParam (HANDLE hDevice,
                    PAI_PARAM pAIParam)
```

功能:将 USB5630.ini 参数配置文件中原来的 AI 参数值复位至出厂时的默认值(Reset parameter to default value for analog input)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAIParam 出口参数，AI 工作参数结构指针，负责在参数被复位后返回其复位后的参数值。关于 AI_PARAM 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头文件，也可参考本文《第一节、AI_PARAM（AI 工作参数结构体）》

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： AI_InitTask() AI_VerifyParam() AI_LoadParam() AI_SaveParam() AI_ResetParam()

3.4 AO 模拟量输出函数原型说明

AO_InitTask()

函数原型：

Visual C++ :

BOOL AO_InitTask (HANDLE hDevice, PAO_PARAM pAOParam, HANDLE* pSampEvent)

功能：初始化 AO 采集任务(Initialize task)，为设备操作就绪有关状态，如预置 AO 采样率、各通道模拟量采样范围等。但它并不启动 AO 设备，如果要启动 AO 设备，须在成功调用此函数之后再调用 AO_StartTask()函数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOParam 入口参数，AO 工作参数结构体指针，决定了 AO 工作时的各种状态及参数，如采样率等。关于其具体定义及说明请参考《第四章 各种结构体描述》\《第七节、AO_PARAM（AO 工作参数结构体）》。

pSampEvent 出口参数，事件句柄，该事件属性为自动发信号，初始状态为不发信号，它由任务自动创建。如果该参数=NULL，则视为用户不需要驱动程序触发任何事件。该事件句柄的作用是：当任务中每完成 AOParam.nSampsPerChan 个点数的数据输出时则会触发此事件。用户可调用 WIN32 API 函数 WaitForSingleObject()来跟踪该事件。当事件未发生时，调用 WaitForSingleObject()函数的采集线程会自动阻塞(等待)状态(此时调用线程不会消耗 CPU 时间)，当事件发生时，则意味着设备中至少有 nSampsPerChan 个点的数据可写，则采集线程立即进入运行状态，并迅速调用 AO_WriteAnalog()或 AO_WriteBinary()循环往任务中写入数据，直至 nAvailSampsPerChan 小于 nWriteSampsPerChan,。如果简单循环调用 AO_GetStatus()查询 AO 的各种状态以同步数据写操作，则会在等待中消耗许多 CPU 时间，而影响系统的整体性能。如果采用事件同步相结合的办法，则会节省大量的 CPU 时间。因为在调用 WaitForSingleObject()时，如果事件未被触发，则当前线程会进入阻塞(等待)状态，而不消耗 CPU 时间，而事件一旦触发，则当前线程立即进入运行状态。总之它可以在一定程度上提升系统的整体性能，让应用程序有更多的 CPU 时间去处理采样数据或其他任务。当然最简单的办法则是使用 AO_WriteAnalog()或 AO_WriteBinary()函数的超时机制，即利用 fTimeout 参数的合理设置来同步写入操作，而无须关注和跟踪 pSampEvent 事件。

返回值：如果初始化 AO 工作参数成功，则返回 TRUE， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_StartTask()

函数原型：

Visual C++ :

BOOL AO_StartTask (HANDLE hDevice)

功能：启动 AO 采集(Start task for analog output)，必须在成功调用 AO_InitTask()函数后才能调用此函数，调用该函数后 AO 立即准备就绪，但 AO 实际是否进入采样记录状态，须依赖于触发事件的产生。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，AO 立刻被启动， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AO_InitTask() AO_StartTask() AO_GetStatus() AO_GetStatus() AO_WaitUntilTaskDone() AO_WriteAnalog() AO_WriteBinary() AO_StopTask() AO_ReleaseTask() DEV_Release()

AO_SendSoftTrig()

函数原型：

Visual C++:

BOOL AO_SendSoftTrig (HANDLE hDevice)

功能：发送软件触发事件(Send Software Trigger),软件触发也叫强制触发。在启动 AO 采集后，来自外部的触发事件可能一直无法产生，用户也可能不知道外部发生了什么，但想马上让设备进入采集状态以便看到具体的信号情况，那么可以调用此函数以软件方式强制设备产生一个触发事件使硬件被正常触发采样一次。该方式也叫软件触发或手动触发或内触发。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，即 AO 立刻被触发采样一次， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AO_InitTask() AO_StartTask() AO_GetStatus() AO_GetStatus() AO_WaitUntilTaskDone() AO_WriteAnalog() AO_WriteBinary() AO_StopTask() AO_ReleaseTask() DEV_Release()

AO_GetStatus()

函数原型：

Visual C++ :

BOOL AO_GetStatus (HANDLE hDevice, PAO_STATUS pAOStatus)

功能：取得 AO 的各种状态(Get status for analog output)。一旦调用 AO_StartTask()函数后，可以调用此函数查询 AO 状态去同步采样数据的写操作。nAvailSampsPerChan>1 时，表示至少可以往采样任务中写入 1 个点的数据。如果在启动采样任务后，设备迟迟不能被触发采样，可以根据需要调用 AO_SendSoftTrig()函数以强制触发设备采样，很快得到有效的可写入采样数据点数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOStatus 出口参数，设备状态参数结构体，它返回设备当前的各种状态，如是否完成采样、是否已被触发等信息。关于具体状态信息请参考《第四章 各种结构体描述》\《第八节、AO_STATUS（AI 工作状态信息结构）》。

返回值：如果成功获取 AO 状态，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() AO_InitTask() AO_StartTask() AO_GetStatus() AO_GetStatus() AO_WaitUntilTaskDone() AO_WriteAnalog() AO_WriteBinary() AO_StopTask() AO_ReleaseTask() DEV_Release()

AO_WaitUntilTaskDone()

函数原型：

Visual C++:

BOOL AO_WaitUntilTaskDone (HANDLE hDevice, F64 fTimeout)

功能：在 AO 的采样任务结束前等待(Wait until task done for analog output)。一旦调用 AO_StartTask()函数后，可以调用此函数等待采样任务结束。它通常用在有限点采样模式中。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

fTimeout 入口参数，超时时间，单位：秒（S）。指定该次等待所用时间，比如设定为 10.0，即 10 秒钟的时间，如果在 10 秒内采样任务结束，则函数立即返回 TRUE，否则 10 秒钟后函数返回值 FALSE，如果采样速率极慢或触发事件长时间都不能达到的情况下，建议该超时时间应足够长；如果想禁止超时返回（即总是等到采样任务结束才返回）则赋值小于 0 即可，如-1.0；如果 fTimeout=0.0，则意味着该函数只是简单查询采样任务是否结束，如果采样任务结束了，则返回 TRUE，否则返回 FALSE。

返回值：如果采样任务结束，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_WriteAnalog()

函数原型:

Visual C++ :

```
LONG AO_WriteAnalog(HANDLE hDevice,
                    F64 fAnlgArray[],
                    U32 nWriteSampsPerChan,
                    U32* pSampsPerChanWritten,
                    U32* pAvailSampsPerChan,
                    F64 fTimeout);
```

功能：写入模拟量采样数据(主要是电压数据) （Write analog data to the task）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

fAnlgArray 入口参数，用户缓冲区，用于接收模拟量数据，值区间由相应采样通道的选用采样范围决定，单位: 伏(V)，获得的电压值数据为双精度浮点型。各个采样通道的数据点是依次交替排列的。它被开辟的点空间不能小于 nWriteSampsPerChan*nSampChanCount(实际采样通道数)。

nWriteSampsPerChan 入口参数，每通道请求写入的数据点数。

在有限点和连续采样模式中，它指定该次从设备的当前可写数据位置写入的数据点数（单位：点）。注意此参数的值如大于当前的可读数点 nAvailSampsPerChan 则会继续等待直到至少有 nWriteSampsPerChan 个点写入后写函数才会返回。等待期间，如果所等时间超过 fTimeout 指定时间也会返回，并置超时错误码。

在连续采样过程中，如果置波形非重生成模式，如果要保持连续不丢点，此参数应尽可能接近于甚至等于当前的可读点数(nAvailSampsPerChan),但不能大于 AOParam.nSampsPerChan。当然此参

数值也不能大于 fAnlgArray 的缓冲区长度，所以为避免出错，所开辟的缓冲区不能小于 nWriteSampsPerChan*nSampChanCount(实际采样通道数)。

pSampsPerChanWritten 出口参数，返回每通道实际写入的点数。在单点采样模式中，如果写入成功，返回的每通道已写入点数总是为 1。注意每次都要检查 pSampsPerChanRead 的返回值，如果返回值等于 0，则要慎重处理。

pAvailSampsPerChan 出口参数，返回该次写操作完成时的每通道还可写而未写的数据点数。它跟 AO_GetStatus()函数取得的状态信息 AIStatus.nAvailSampsPerChan 是同一个状态信息。返回可读点数的意义在于通过数据写操作直接提供给用户，避免再次调用 AO_GetStatus()函数，即简化了实现方法，又提高了效率，即在读数据函数返回时判断可写点数，如果大于 nWriteSampsPerChan，则可紧接着再次调用写数据函数，直到 pAvailSampsPerChan 返回值小于 nWriteSampsPerChan。在单点采样模式中,它的返回值总是为 0。

fTimeout 入口参数，超时时间，单位：秒（S）。指定等待写入额定点数的时间。比如设定为 10.0，如果在 10 秒内写入点数达到 nWriteSampsPerChan 时立即返回 TRUE，否则 10 秒钟后函数返回值 FALSE，并通过 pAvailSampsPerChan 告之实际写入的点数，如果采样速率极慢或触发事件长时间都不能达到的情况下，建议该超时时间应足够长；如果想禁止超时返回（即等待请求数据点数完全写入任务中才返回）则置为负数，如-1.0 即可。如果 fTimeout=0.0，则意味着该函数仅简单判断能否立即写入请求的点数，如果不能，则不等待，立即返回 FALSE，否则返回 TRUE。

返回值： 如果函数调用成功则返回 TRUE，否则返回 FALSE，可以调用 Win32 API 函数 GetLastError()以取得进一步的错误码信息 nErrorCode，其定义如下表。

常量名	常量值	功能定义
ERROR_NO_AVAILABLE_SAMPS	0xE0000000+1	无有效点数据
ERROR_SAMPLE_TASK_FAIL	0xE0000000+2	采样任务失败
ERROR_TIMEOUT	1460L	超时错误(见微软定义 WinError.h)

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_WriteBinary()

函数原型:

Visual C++ :

```
LONG AO_WriteBinary(HANDLE hDevice,
                    U16 nBinArray[],
                    U32 nWriteSampsPerChan,
                    U32* pSampsPerChanWritten,
                    U32* pAvailSampsPerChan,
                    F64 fTimeout);
```

功能：写入二进制原码采样数据（Write binary data to the task）。它是将用户二进制原码数据写入采样任务中。二进制原码具有数据紧凑、数据量小、传输快、处理快、存盘也快的特点。

参数：

hDevice 同 AO_WriteAnalog()。

nBinArray 入口参数，用户缓冲区，用于接收二进制原码数据，值区间[0，65535]。各个采样

通道的数据点是依次交替排列的。它被开辟的点空间不能小于nWriteSampsPerChan*nSampChanCount(实际采样通道数)。

nWriteSampsPerChan 同 AO_WriteAnalog()。

pSampsPerChanWritten 同 AO_WriteAnalog()。

pAvailSampsPerChan 同 AO_WriteAnalog()。

fTimeout 同 AO_WriteAnalog()。

返回值：同 AO_WriteAnalog()。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_ReadbackAnalog()

函数原型:

Visual C++ :

```
BOOL AO_ReadbackAnalog(HANDLE hDevice,
                       F64 fAnlgArray[4]);
```

功能:回读所有 AO 通道的当前生成的模拟量数据(电压数据序列)(Read back analog data from the task)。

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

fAnlgArray fAnlgArray[0]=AO0 电压数据,fAnlgArray[1]=AO1 电压数据,fAnlgArray[2]=AO2 电压数据, ,fAnlgArray[3]=AO3 电压数据, 取值区间由相应通道的采样范围决定。

返回值：同 AO_WriteAnalog()。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_ReadbackBinary()

函数原型:

Visual C++ :

```
BOOL AO_ReadbackBinary(HANDLE hDevice,
                       U16 nBinArray[4]);
```

功能:回读所有 AO 通道的当前生成的模拟量数据(电压数据序列)(Read back analog data from the task)。

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nBinArray nBinArray[0]=AO0 原码数据,nBinArray[1]=AO1 原码数据,nBinArray[2]=AO2 原码数据,,nBinArray[3]=AO3 原码数据, 取值区间[0, 4095]

返回值：同 AO_WriteAnalog()。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_StopTask()

函数原型：

Visual C++:

BOOL AO_StopTask (HANDLE hDevice)

功能：停止 AO 采样(Stop task for analog output)。必须在成功调用 AO_StartTask()函数后才能调用此函数。该函数除了停止 AO 采样外不改变设备的其他状态。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，且 AO 立刻停止转换， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_ReleaseTask()

函数原型：

Visual C++ :

BOOL AO_ReleaseTask(HANDLE hDevice)

功能：释放 AO(Release AO Task)。必须在重新调用 AO_InitTask() 函数之前被调用一次，即该函数必须和 AO_InitTask() 成对出现。注意此函数在内部首先执行 AO_StopTask()函数停止 AO 采集后，才释放被占用的 AO 资源。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：

DEV_Create()	AO_InitTask()	AO_StartTask()
AO_GetStatus()	AO_GetStatus()	AO_WaitUntilTaskDone()
AO_WriteAnalog()	AO_WriteBinary()	AO_StopTask()
AO_ReleaseTask()	DEV_Release()	

AO_GetMainInfo()

函数原型：

Visual C++ :

BOOL AO_GetMainInfo (HANDLE hDevice,
PAO_MAIN_INFO pMainInfo)

功能：取得 AO 功能的主要信息，如通道数、分辨率等（Get main information for analog output）。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pMainInfo 出口参数，AO 主要信息结构指针，负责返回 AO 的主要描述信息。关于 AO_MAIN_INFO 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头

文件，也可参考本文《第九节、AO_MAIN_INFO (AO 主要信息结构体)》关于该结构的有关说明。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO_GetMainInfo()　AO_GetVOLTRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_GetVoltRangeInfo()

函数原型：

Visual C++:

```
BOOL AO_GetVoltRangeInfo(HANDLE hDevice,
                         U32 nChannel,
                         U32 nSampleRange,
                         PAO_VOLT_RANGE_INFO pRangeInfo)
```

功能：取得 AO 指定通道、指定采样范围档的上下限电压、幅度等信息（Get range information for analog output）。

参数：

hDevice　入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel　通道号(本设备有 2 个通道,取值范围[0,1])。

nSampleRange　入口参数，AO 采样范围，同 AOParam.nSampleRange。

pRangeInfo　出口参数，AO 采样范围信息，负责返回 AO 的采样范围大值、最小值、幅度、编码宽度等。请参考《第十节、AO_SAMP_RANGE_INFO (AO 采样范围信息结构体)》

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO_GetMainInfo()　AO_GetVOLTRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_GetRateInfo()

函数原型：

Visual C++ :

```
BOOL AO_GetRateInfo(HANDLE hDevice, PAO_SAMP_RATE_INFO pRateInfo);
```

功能：获得采样速率信息（Get rate information for analog output）。

参数：

hDevice　入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pRateInfo　出口参数，AO 采样速率信息，负责返回 AO 的最大采样率，最小采样率等。详情请参考《第五节、AO_RATE_INFO (AO 采样率信息结构体)》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO_GetMainInfo()　AO_GetVOLTRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_ScaleBinToVolt()

函数原型：

Visual C++:

```
BOOL AO_ScaleBinToVolt(PAO_VOLT_RANGE_INFO pRangeInfo,
                       F64 fVoltArray[],
                       U16 nBinArray[],
                       U32 nScaleSamps,
                       U32* pSampsScaled);
```

功能：将二进制原码数据转换为电压数据(Scale binary data to voltage data)。与AO_ScaleVoltToBin()函数的功能相反。

参数：

pRangeInfo 当前转换数据需要的采样范围信息

fVoltArray 出口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由 nSampleRange 参数选择而定。

nBinArray 入口参数，用于传入待量化的原码数据，取值范围[0, 4095]。

nScaleSamps 入口参数，请求量化的数据点数。

pSampsScaled 出口参数，返回实际量化后数据点数。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError() 捕获错误码以确定具体原因。

相关函数： DEV_Create() AO_ScaleBinToVolt() AO_ScaleVoltToBin() AO_GetMainInfo() AO_GetVOLTRangeInfo() AO_GetRateInfo() DEV_Release()

AO_ScaleVoltToBin()

函数原型：

Visual C++ :

```
BOOL AO_ScaleVoltToBin(PAO_VOLT_RANGE_INFO pRangeInfo,
                       U16 nBinArray[],
                       F64 fVoltArray[],
                       U32 nScaleSamps,
                       U32* pSampsScaled);
```

功能：将电压数据转换为原码数据(Scale voltage data to binary data)。与 AO_ScaleBinToVolt()函数的功能相反。

参数：

pRangeInfo 当前转换数据需要的采样范围信息

fVoltArray 出口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由 nSampleRange 参数选择而定。

nBinArray 出口参数，用于传入待量化的原码数据，取值范围[0, 4095]。

fVoltArray 入口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由 nSampleRange 参数选择而定。

nScaleSamps 入口参数，请求量化的数据点数。

pSampsScaled 出口参数，返回实际量化后数据点数。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO_GetMainInfo()　AO_GetVOLTRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_VerifyParam()

函数原型：

Visual C++ :

```
BOOL AO_VerifyParam (HANDLE hDevice, PAO_PARAM pAOParam)
```

功能：校验 AO 工作参数(Verify task parameter for analog output)，这是一个辅助功能的函数，在初始化 AO 前，事先校验 AO 参数的合法性。

参数：

hDevice　入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOParam　入口和出口参数，AO 工作参数结构体指针，关于其具体定义及说明请参考《第四章 各种结构体描述》\《第七节、AO_PARAM（AO 工作参数结构体）》。函数调用时，它传入要校验的各项参数，函数返回时，它将经过调整为合法的参数值返回给调用者，以便后续初始化 AO 使用。

返回值：如果所有的参数均合法，则返回 TRUE；如果有一个参数不合法，立即被调整为合法取值，并向日志文件 USB5630.log 记录不合法的参数名和原因，然后返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_InitTask()　AO_VerifyParam()　AO_LoadParam()　AO_SaveParam()　AO_ResetParam()　DEV_Release()

AO_LoadParam()

函数原型：

Visual C++ :

```
BOOL AO_LoadParam (HANDLE hDevice,
                   PAO_PARAM pAOParam)
```

功能：从 USB5630.ini 参数配置文件中读取设备的硬件参数(Load parameter from USB5630.ini for analog output)。

参数：

hDevice　入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOParam　出口参数，属于 PAO_PARAM 的结构指针类型，负责返回 AO 工作参数值，关于结构指针类型 PAO_PARAM 请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头文件，也可参考本文《第七节、AO_PARAM（AO 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create()　AO_InitTask()　AO_VerifyParam()　AO_LoadParam()　AO_SaveParam()　AO_ResetParam()　DEV_Release()

AO_SaveParam()

函数原型：

Visual C++:

BOOL AO_SaveParam (HANDLE hDevice,
PAO_PARAM pAOParam)

功能：将配置好的 AO 工作参数保存在 USB5630.ini 参数配置文件中(Save parameter to USB5630.ini for analog output)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOParam 入口参数，AO 工作参数结构体指针，关于 AO_PARAM 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 等函数原型定义的头文件，也可参考本文《第七节、AO_PARAM（AO 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() AO_InitTask() AO_VerifyParam() AO_LoadParam() AO_SaveParam() AO_ResetParam() DEV_Release()

AO_ResetParam()

函数原型：

Visual C++ :

BOOL AO_ResetParam (HANDLE hDevice,
PAO_PARAM pAOParam)

功能：将 USB5630.ini 参数配置文件中原来的 AO 参数值复位至出厂时的默认值(Reset parameter to default value for analog output)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

pAOParam 出口参数，AO 工作参数结构指针，负责在参数被复位后返回其复位后的参数值。关于 AO_PARAM 的详细介绍请参考 USB5630.h 或 USB5630.bas 或 USB5630.pas 函数原型定义的头文件，也可参考本文《第七节、AO_PARAM（AO 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() AO_InitTask() AO_VerifyParam() AO_LoadParam() AO_SaveParam() AO_ResetParam() DEV_Release()

3.5 CTR 计数器函数原型说明

CTR_InitTask()

函数原型：

Visual C++:

BOOL CTR_InitTask(HANDLE hDevice, U32 nChannel, PCTR_PARAM pCTRParam);

功能：初始化指定计数器通道的工作参数 (Initialize task parameter for Counter)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

pCTRParam 入口参数，计数器工作参数结构体，详情参考《第十二节、CTR_PARAM (CTR 计数器工作参数结构体)》。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() CTR_InitTask() CTR_StartTask() CTR_ReadCounter() CTR_StopTask() CTR_ReleaseTask() DEV_Release()

CTR_StartTask()

函数原型：

Visual C++ :

```
BOOL AO_StartTask (HANDLE hDevice, U32 nChannel)
```

功能：启动 CTR 计数(Start task for counter)，必须在成功调用 CTR_InitTask()函数后才能调用此函数，调用该函数后 CTR 立即启动。如果在调用 CTR_StopTask()之后调用此函数，则意味着让 CTR 接着之前的计数值继续计数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

返回值：如果调用成功，则返回 TRUE，CTR 立刻被启动， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() CTR_InitTask() CTR_StartTask() CTR_ReadCounter() CTR_StopTask() CTR_ReleaseTask() DEV_Release()

CTR_ReadCounter()

函数原型：

Visual C++ :

```
BOOL CTR_ReadCounter(HANDLE hDevice, U32 nChannel, U32* pCountVal);
```

功能：读取指定通道的计数器值 (Read the count value for counter)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

pCountVal 出口参数，计数器当前计数值。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数：DEV_Create() CTR_InitTask() CTR_StartTask() CTR_ReadCounter() CTR_StopTask() CTR_ReleaseTask() DEV_Release()

CTR_StopTask()

函数原型：

Visual C++ :

BOOL CTR_StopTask(HANDLE hDevice, U32 nChannel)

功能：启动 CTR 计数(Stop task for counter)，必须在成功调用 CTR_StartTask()函数后才能调用此函数，调用该函数后 CTR 立即停止计数。之后还可以再调用 CTR_StartTask()继续计数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

返回值：如果调用成功，则返回 TRUE，CTR 立刻被停止， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() CTR_InitTask() CTR_StartTask() CTR_ReadCounter() CTR_StopTask() CTR_ReleaseTask() DEV_Release()

CTR_ReleaseTask()

函数原型：

Visual C++ :

BOOL CTR_ReleaseTask (HANDLE hDevice, U32 nChannel)

功能：释放 CTR (Relase task for Counter)，必须在成功调用 CTR_InitTask()函数后才能调用此函数。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nChannel 通道号(本设备有 2 个通道,取值范围[0,1])。

返回值：如果调用成功，则返回 TRUE，CTR 被成功释放， 否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() CTR_InitTask() CTR_StartTask() CTR_ReadCounter() CTR_StopTask() CTR_ReleaseTask() DEV_Release()

CTR 控制流程

(1) DEV_Create() 创建设备句柄；

(2) CTR_InitTask() 初始化 CTR 任务参数；

(3) CTR_StartTask() 启动 CTR 计数任务(或重启)；

(4) CTR_ReadCounter() 实时读取 CTR 当前计数值；

(5) CTR_StopTask() 停止 CTR 计数任务（或暂停）；

(6) CTR_ReleaseTask() 释放 CTR 任务；

(7) DEV_Release() 释放设备句柄。

可以反复执行第(4)步，以实时读取计数器值。

3.6 DIO 数字量输入输出函数原型说明

DIO_ReadPort()

函数原型：

Visual C++ :

BOOL DIO_ReadPort (HANDLE hDevice, U32 nPort, U32* pPortData);

功能：读取 DI 的端口数据 (Read port data for digital input)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 端口号, 取值范围为[0,2]。

pPortData 返回的端口数据, 有效位 Bit[7:0], 读 Port0 时为 DI 输入，读 Port1 时为 DO 回读。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO_WritePort()

函数原型：

Visual C++:

BOOL DIO_WritePort (HANDLE hDevice, U32 nPort, U32 nPortData);

功能：写入 DO 端口数据 (Write port data for digital output)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 端口号, 取值范围为[0,2]。

nPortData 端口数据, 有效位 Bit[7:0]。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO_ReadLines()

函数原型：

Visual C++ :

BOOL DIO_ReadLines (HANDLE hDevice, U32 nPort, U32 bLineDataArray[8]);

功能：读数字量端口多线值(Read Mult Lines Data from the DI Port)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 入口参数，端口号，本设备仅支持一个端口，故恒等于 0。

bLineDataArray 线数据缓冲区, 同时返回端口中各线的状态值 bLineDataArray[n]=0:表示关

(或低)状态, =1 表示开(或高)状态。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO_WriteLines()

函数原型：

Visual C++ :

BOOL DIO_WriteLines(HANDLE hDevice, U32 nPort, U32 bLineDataArray[8]);

功能：写入 DO 的多线数据 (Write lines data for digital output)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 端口号, 取值范围为[0,2]。

bLineDataArray 线数据缓冲区, 端口中各线的状态值 bLineDataArray[n]=0:表示关(或低)状态, =1 表示开(或高)状态。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO_ReadLine()

函数原型：

Visual C++:

BOOL DIO_ReadLine (HANDLE hDevice, U32 nPort, U32 nLine, U32* pLineData);

功能：读取 DI 的单线数据 (Read line data for digital input)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 端口号, 取值范围为[0,2]。

nLine 入口参数，指定端口中的线号。本设备有一个端口共 8 条线，故取值范围为[0, 7]，分别代表 Line0(DI0)、Line1(DI1)… Line7(DI7)。

pLineData 出口参数，从指定端口中指定线号上读入的线数据，线数据只有两种取值：0（低）或 1（高）。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO_WriteLine()

函数原型：

Visual C++ :

BOOL DIO_WriteLine(HANDLE hDevice, U32 nPort, U32 nLine, U32 bLineData);

功能：写入 DO 的单线数据 (Write line data for digital output)。

参数：

hDevice 入口参数，设备对象句柄，由 DEV_Create()函数创建，该句柄指向要访问的设备。

nPort 端口号, 取值范围为[0,2]。

nLine 入口参数，指定端口中的线号。本设备有一个端口共 8 条线，故取值范围为[0, 7]，分别代表 Line0(DO0)、Line1(DO1)…Line7(DO7)。

bLineData 出入口参数，向指定端口中指定线号上写入的线数据，线数据只有两种取值：0（低）或 1（高）。

返回值：如果成功，返回 TRUE，否则返回 FALSE，可立即调用 WIN32 API 函数 GetLastError()捕获错误码以确定具体原因。

相关函数： DEV_Create() DIO_ReadPort() DIO_WritePort() DIO_WriteLines() DIO_ReadLine() DIO_WriteLine() DIO_ReadLines() DEV_Release()

DIO 控制流程

(1) DEV_Create() 创建设备句柄

(2) DIO_ReadPort()或 DIO_WritePort()、DIO_WriteLines()、DIO_ReadLine() DO_WriteLine()、DIO_ReadLine()实时读写 DI、DO 端口或线数据

(3) DEV_Release() 释放设备句柄

可以反复执行第(2)步，以进行数字量输入输出

4 各种结构体描述

4.1 AI_PARAM（AI 工作参数结构体）

Visual C++ :

```
typedef struct _AI_CH_PARAM
{
    U32 nChannel;
    U32 nSampleGain;
    U32 nRefGround;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
}
AI_CH_PARAM, *PAI_CH_PARAM;

typedef struct _AI_PARAM
{
    U32 nSampChanCount;
    USB5630_AI_CH_PARAM CHParam[64];
    U32 nChanScanMode;
    U32 nGroupLoops;
    U32 nGroupInterval;
    U32 nSampleSignal;
    U32 nSampleRange;

    U32 nSampleMode;
    U32 nSampsPerChan;
    F64 fSampleRate;
    U32 nClockSource;
    U32 nClockOutput;
    U32 nReserved0;
    U32 nReserved1;

    U32 bDTriggerEn;
    U32 nDTriggerDir;
    U32 nDTrigChannel;
    U32 bATriggerEn
    U32 nATriggerDir;
    F32 fTriggerLevel;
    U32 nTriggerSens;
    I32 nDelaySamps;
    U32 nReserved2;
```

```
    U32 nReserved3;
    U32 nReserved4;
    U32 nReserved5;
    U32 nReserved6;
    U32 nReserved7;
} AI_PARAM, *PAI_PARAM;
```

4.1.1 AI_CH_PARAM(AI 通道参数结构体)

nChannel

AI 物理通道号，取值范围[0, 7]，即 AI0－AI7。

nSampleGain

采样增益选择(Sample Gain)，取值范围[0, 3]，具体定义参考下表

常量名	常量值	功能定义	备注
AI_SAMPGAIN_1MULT	0	1 倍增益	
AI_SAMPGAIN_2MULT	1	2 倍增益	
AI_SAMPGAIN_4MULT	2	4 倍增益	
AI_SAMPGAIN_8MULT	3	8 倍增益	

nRefGround

参考地(Referenced Ground)，取值范围[0, 2]，具体定义参考下表

常量名	常量值	功能定义	备注
AI_REFGND_RSE	0	单端输入，有参考地(Referenced Single Endpoint)	
AI_REFGND_NRSE	1	单端输入，无参考地(Non Referenced Single Endpoint)	
AI_REFGND_DI	2	差分输入(Differential)，也叫双端输入	

当参考地选择为单端时，则采样信号通过 AI0-AI15 分 16 路输入；当参考地为差分输入时，采样信号通过[AI0+, AI8-]；[AI1+、AI9-]； [AI2+、AI10-]； [AI3+、AI11-]…共 8 路输入。

nReserved0-2

保留字段。

4.1.2 AI_PARAM(AI 工作参数结构体）

nSampChanCount

采样通道数量(Sample Channel Count)，进入采样过程的通道个数，取值范围[1，64]。即决定了 CHParam[]通道组阵列中有效单元的个数。若 nSampChanCount=1，则表示仅 CHParam[0]单元决定的物理通道号有效；若 nSampChanCount=2，则表示仅 CHParam[0]、CHParam[1]两个单元决定的物理通道号有效；若 nSampChanCount=3，则表示仅 CHParam[0]、CHParam[1]、、CHParam[2]三个单元决定的物理通道号有效，依次类推。

CHParam[64]

通道参数配置(此参数在单点采样有效，未此注明的参数则在单点采样模式中无效)。具体定义请

参考《一、AI_CH_PARAM(AI 通道参数结构体)》。

nChanScanMode

通道扫描模式(Channel Scan Mode)，分为通道连续扫描模式和通道分组扫描模式。取值和定义如下表

常量名	常量值	功能定义	备注
AI_CHAN_SCANMODE_CONTINUOUS	0	通道连续扫描	默认值
AI_CHAN_SCANMODE_GROUP	1	通道分组扫描	

nGroupLoops

通道组循环次数(Group Loops)，取值范围为[1, 65535]。它仅在通道扫描模式为分组扫描时有效。决定着在分组采样过程中，每个通道组（由 nSampChanCount 和 CHParam[]共同决定）在每个通道组扫描时钟下循环采样的次数（即每个通道的点数）。

nGroupInterval

每两个通道组间的时间间隔(Interval Between Channel Group)，单位：微秒(uS)。取值范围为[0, 107374182]。它仅在通道扫描模式为分组扫描时有效。

在分组采样中，组内或组循环内的通道采样间隔是相等的，且时间由 fSampleRate*nSampChanCount 的乘积求倒数得到(单位：秒)。而这一组（包括组循环）与下一组（包括组循环）之间的时间间隔则由 nGroupInterval 决定；在等间隔采样中，所有采样数据的时间间隔均由 fSampleRate*nSampChanCount 的乘积求倒数得到的时间决定(单位：秒)。

nSampleSignal

AI 采样信号(Sample Signal)，取值范围为[0, 7]，如下表：

常量名	常量值	功能定义	备注
AI_SAMPSIGNAL_AI	0	AI 通道输入信号	默认值
AI_SAMPSIGNAL_0V	1	0V(AGND)	
AI_SAMPSIGNAL_2D5V	2	2.5V(DC)	
AI_SAMPSIGNAL_5V	3	5V(DC)	
AI_SAMPSIGNAL_AO0	4	AO0(DC)	
AI_SAMPSIGNAL_AO1	5	AO1(DC)	
AI_SAMPSIGNAL_AO2	6	AO2(DC)	
AI_SAMPSIGNAL_AO3	7	AO3(DC)	

nSampleRange

AI 采样范围(Sample Range)，取值范围为[0, 4]，本设备 AI 采样分辨率为 16Bit，因此各采样范围和编码宽度如下表：

常量名	常量值	采样范围	实际采样范围(V)	编码宽度 CodeWidth(V)
AI_SAMPRANGE_N10_P10V	0	±10V	[-10.0000, +9.99969]	0.0003052
AI_SAMPRANGE_N5_P5V	1	±5V	[-5.0000, +4.999847]	0.0001526
AI_SAMPRANGE_N2D5_P2D5V	2	±2.5V	[-2.5000, +2.499924]	0.0000763
AI_SAMPRANGE_0_P10V	3	0-10V	[0.00000, +9.999847]	0.0001526

AI_SAMPRANGE_0_P5V	4	0-5V	[0.00000, +4.999237]	0.0000763

nSampleMode

AI 采样模式(Sample Mode)，具体定义见下表：

常量名	常量值	功能定义	备注
AI_SAMPMODE_ONE_DEMAND	0	软件按需单点采样	
AI_SAMPMODE_FINITE	2	有限点采样	
AI_SAMPMODE_CONTINUOUS	3	连续采样	

软件按需单点采样模式：就是调用 AI_StartTask()后，AI 任务只是就绪，但并不实际采样数据，而要等到每次软件调用 AI_ReadAnalog()或 AI_ReadBinary()函数时任务才开始实际采样，且每个通道仅采样一个点的数据，并以最快的速度返回。这种采样模式主要针对简单采样或采样实时性要求较高，数据量很少，且时间不确定的应用中，比如应用在对时间边界没有严格要求的 PID，PLC 等快速伺服闭环控制系统。不支持触发和外时钟。

有限点采样模式：按照设定好的采样速率和触发条件，触发模式等参数进行定长时间的、限制点数的、连续的、等间隔的波形数据采集。在启动采样任务后，达到触发条件并采样完成指定点数的数据后，采样任务就会自动停止。这个功能主要是应用在频繁捕捉触发事件、有点数限制和时间长度限制、尽可能还原外界信号的场合。

连续采样模式：按照设定好的采样速率和触发条件，触发模式等参数进行长时间的、不限点数的、连续的、等间隔的波形数据采集，在启动采样任务后，只要不软件停止采样任务，其采样任务是永远不会终止的。这个功能主要是应用在不限制点数和时间长度的，且不丢点的，尽可能还原外界信号的场合。

nSampsPerChan

AI 每通道待读取点数(Samples Per Channel)。

单点采样模式下，该参数无意义；

有限点采样模式下，该参数表示每通道采样点数。取值范围为[2，1024*1024*16]样点，最大取值还要受制于系统可用内存，采样通道数等；

连续采样模式下，决定着触发采样事件 hSampEvent 时的点数条件。比如指定该参数值为 1024 点，则每采样到不小于 1024 点时就会触发采样事件 hSampEvent，它也决定着每次调用 AI_ReadAnalog()或 AI_ReadBinary()时能最快返回的点数边界（请注意：是不小于 nSampsPerChan，意思是不可能正好等于 nSampsPerChan 时就能得到事件通知，而是通常在超过 nSampsPerChan 时才能得到事件通知）。即该参数的取值大小决定着每两次读到数据的时间间隔，也就是实时响应度。点数越小，实时响应就越高，反之越低。但不能为了一味的追求实时响应度就将该参数的值设得很小，这个还要看采样速率的高低，如果采样速率很高，而 nSampsPerChan 的值很小，则可能造成任务负担过重而发生缓冲区溢出，以致造成丢点现象的发生。建议实时响应度不低于 20 个毫秒是比较合适的。比如每通道采样速率为 100Ksps，即 10 微秒一个点，则 nSampsPerChan 不小于 2000 个点是比较合适的。该参数的取值范围为[2, 1024*1024]，具体还要受制于系统可用内存和采样通道数。

fSampleRate

AI 采样速率(Sample Rate)，单位：每秒样点 sps(sample per second)，它指每个采样通道的采样速率，决定了单个采样通道每秒钟采样的点数。而每个采样点的周期则由 fSampleRate 求倒数取得，单位：秒(S)。它的最小取值等于 1sps，而最大取值则由设备的总采样率(250000sps)、采样通道数量(nSampChanCount)决定，比如采样通道数量为 3 个通道，则该参数最大取值为 250000sps/3 =

83333sps，比如采样通道数量为 4 个通道，则该参数最大取值为 500000sps/4 = 62500sps；比如采样通道数量为 8 个通道，则该参数最大取值为 500000sps/8 = 31250sps。

nClockSource

时钟源(Clock Source)。本设备的 AI 采样时钟支持本地时钟源(LOCAL)和外部时钟源(EXCLK)。本地时钟源即指板上晶振时钟源(OSCCLK)。外部时钟源则来自于由连接器 J1 上的 CLKIN 输入，该参数的取值范围为[0，16]，具体定义如下表：

nClockSource 选项(常量名)	常量值	功能定义	备注
AIO_CLKSRC_LOCAL	0	本地时钟，也叫内部时钟(通常为本地晶振时钟 OSCCLK)	默认值
AIO_CLKSRC_PFI00	1	PFI00 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI01	2	PFI01 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI02	3	PFI02 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI03	4	PFI03 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI04	5	PFI04 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI05	6	PFI05 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI06	7	PFI06 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI07	8	PFI07 输入时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI08	9	PFI08 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI09	10	PFI09 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI10	11	PFI10 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI11	12	PFI11 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI12	13	PFI12 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI13	14	PFI13 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI14	15	PFI14 输入时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI15	16	PFI15 输入时钟(时钟输入输出、DIO、DTR 触发复用)	

nClockOutput

本地采样时钟是否输出(Clock Output Enable)。如果置为 TRUE：表示允许由 fSampleRate 决定的采样时钟会直接输出到 J1 的 CLKOUT， 如果置为 FALSE：禁止输出。

nClockSource 选项(常量名)	常量值	功能定义	备注
AIO_CLKOUT_DISABLE	0	禁止采样时钟输出	默认值
AIO_CLKOUT_PFI00	1	PFI00 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI01	2	PFI01 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI02	3	PFI02 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI03	4	PFI03 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI04	5	PFI04 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI05	6	PFI05 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI06	7	PFI06 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI07	8	PFI07 输出时钟(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AIO_CLKSRC_PFI08	9	PFI08 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI09	10	PFI09 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI10	11	PFI10 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI11	12	PFI11 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI12	13	PFI12 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI13	14	PFI13 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI14	15	PFI14 输出时钟(时钟输入输出、DIO、DTR 触发复用)	
AIO_CLKSRC_PFI15	16	PFI15 输出时钟(时钟输入输出、DIO、DTR 触发复用)	

下面所讲述的是开始触发的参数。关于有限点采样模式中的参考触发本设备不提供，如果用户需要参考触发请关注其他有关产品。

nDTrigChannel DTR

数字量触发 DTR 允许(Digital Trigger Enable)。如果等于 TRUE，表示外部数字量触发输入信号允许进入 AI 的触发系统，如果等于 FALSE，表示不允许(即禁用)。触发信号由连接器 J1 上的 DTR 输入。它的取值如下表：

常量名	常量值	功能定义	备注
AI_DTRIG_PFI00	0	PFI00 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	默认值
AI_DTRIG_PFI01	1	PFI01 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI02	2	PFI02 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI03	3	PFI03 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI04	4	PFI04 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI05	5	PFI05 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI06	6	PFI06 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI07	7	PFI07 触发(时钟输入输出、DIO、DTR 触发、CNT 复用)	
AI_DTRIG_PFI08	8	PFI08 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI09	9	PFI09 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI10	10	PFI10 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI11	11	PFI11 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI12	12	PFI12 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI13	13	PFI13 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI14	14	PFI14 触发(时钟输入输出、DIO、DTR 触发复用)	
AI_DTRIG_PFI15	15	PFI15 触发(时钟输入输出、DIO、DTR 触发复用)	

nDTriggerDir

数字量触发极性(Digital Trigger Direction)。它的取值如下表：

常量名	常量值	功能定义	备注
AI_TRIGDIR_FALLING	0	下降沿触发	默认值
AI_TRIGDIR_RISING	1	上升沿触发	

AI_TRIGDIR_CHANGE	2	变化触发(上或下边沿触发)	

bATriggerEn

模拟量触发 ATR 允许(Analog Trigger Enable)。如果等于 TRUE，表示模拟通道触发输入信号允许进入 AI 的触发系统，如果等于 FALSE，表示不允许(即禁用)。

nATriggerDir

模拟量触发极性(Analog Trigger Direction)。它的取值如下表：

常量名	常量值	功能定义	备注
AI_TRIGDIR_FALLING	0	下降沿触发	默认值
AI_TRIGDIR_RISING	1	上升沿触发	
AI_TRIGDIR_CHANGE	2	变化触发(上或下边沿触发)	

fTriggerLevel

AI 采样通道的触发电平(Trigger Level)，单位：伏(V)，分辨率 12Bit。取值范围要受到 nSampleRange 参数所选择的模拟量输入范围档的限定，如下表所示：

采样范围挡位号	常量值	采样范围	fTriggerLevel 的范围(V)	编码宽度(V)
AI_SAMPRANGE_N10_P10V	0	±10V	[-10.0000, +9.9951]	0.0048828
AI_SAMPRANGE_N5_P5V	1	±5V	[-5.0000, +4.9975]	0.0024414
AI_SAMPRANGE_N2D5_P2D5V	2	±2.5V	[-2.5000, +2.4987]	0.0012207
AI_SAMPRANGE_0_P10V	3	0-10V	[0.00000, +9.9975]	0.0024414
AI_SAMPRANGE_0_P5V	4	0-5V	[0.00000, +4.99877]	0.0012207

nTriggerSens

AI 触发灵敏度（Trigger Sense)，单位：微秒(uS)。取值范围为[0, 1638]，它的实际功能是为避免触发信号中较高频的噪声分量也进入触发系统而设定的一种时间门限，凡是触发信号跳变后稳定保持时间不小于该门限时间则进入触发系统，否则不进入而被屏蔽掉。此参数仅对数字量和模拟量触发均有效。跳变保持时间：指的是触发信号由一个状态跳变至另一个状态时所能保持的时间。举例说明，一个数字量触发信号原来是低电平，然后跳变至高电平，保持 10 微秒后又回到低电平，即跳变保持时间指的就是高电平在这个瞬间保持的时间 10 微秒。这个时间越短越有可能是高频噪声，因此需要这个参数加以控制或过滤，以避免噪声信号产生误触发。

nDelaySamps

触发延迟点数（Delay Samples）。取值范围 32 位有效[0, 4294967295]。其值等于 0 时为后触发（Post Trigger)；其值大于 0 时为正延迟触发(Delay Trigger)。就是在开始触发产生时，再延迟若干个采样点后再记录数据，其延迟的点数就是由 nDelaySamps 指定，而单个点的时间周期由 nSampleRate 指定的每通道采样速率决定的。比如 nDelaySamps=100，nSampleRate=1000Hz（即每采样点周期为 1 毫秒)，则意味着在启动采样任务后，当发生开始触发时再等待 100 毫秒（1 毫秒*100）才实际进入数据采样与记录阶段。在有些应用中，用户关注的焦点信号是在触发事件之后的一段时间才发生，那么这个功能就是满足此种应用的。

nReserved2-7

保留字段(未作定义)，可强制赋 0。

相关函数：　DEV_Create()　AI_InitTask()　AI_LoadParam()
AI_SaveParam()　AI_ResetParam　DEV_Release()

4.2　AI_STATUS（AI 工作状态信息结构）

Visual C++ :

```
typedef struct _AI_STATUS
{
    U32 bTaskDone;
    U32 bTriggered;

    U32 nTaskState;
    U32 nAvailSampsPerChan;
    U32 nMaxAvailSampsPerChan;
    U32 nBufSampsPerChan;
    I64 nSampsPerChanAcquired;

    U32 nHardOverflowCnt;
    U32 nSoftOverflowCnt;
    U32 nInitTaskCnt;
    U32 nReleaseTaskCnt;
    U32 nStartTaskCnt;
    U32 nStopTaskCnt;
    U32 nTransRate;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AI_STATUS, *PAI_STATUS;
```

此结构体主要用于查询 AI 的工作状态信息，AI_GetStatus()函数使用此结构体来实时取得 AI 的工作状态信息，以便同步数据采样和处理过程。

bTaskDone

AI 采样任务完成标志(Task Done)。=TRUE:表示采样任务已结束； =FALSE:表示采样任务正在进行中。在设备上电之初或执行 AI_StopTask()函数后其值为 TRUE，当执行 AI_StartTask()函数后为 FALSE。在有限点采样任务中，如果达到触发条件和采样点数后，任务会自动停止，此标志会被自动置成 TRUE。在连续采样任务中，只有调用 AI_StopTask()函数手动停止采样任务，此标志才会被置成 TRUE。

bTriggered

AI 触发标志。=TRUE:表示已被触发, =FALSE:表示未被触发或等待触发。在设备上电之初或当执行 AI_StartTask()后其值为 FALSE。在被正常触发后自动变为 TRUE。执行 AI_StopTask()后其值不变。

nTaskState

采样任务状态(Sample Task State), 当=1 时表示正常, 其它值表示有异常情况。

nAvailSampsPerChan

有效点数，表示采样任务缓冲中的有效数据点数（Available Samples Per Channel）。如果它小于 nReadSampsPerChan 的时候调用 AI_ReadAnalog() 或 AI_ReadBinary()时，则读数函数会自动进入超时等待睡眠状态，直至可读点数达到指定读取点数 nReadSampsPerChan 才会返回，如果等待时间超过 fTimeout 的值，也会返回 FALSE，并置超时错误状态。如果它大于 nReadSampsPerChan 的时候调用 AI_ReadAnalog() 或 AI_ReadBinary()时，则读数函数会迅速返回指定点数的数据并返回 TRUE。在连续采样模式中，如果 nAvailSampsPerChan 的值大于或等于 nBufSampsPerChan，则意味着采样缓冲区已经发生溢出，其溢出次数可以通过 nHardOverflowCnt 和 nSoftOverflowCnt 观察到。这个状态值的监测主要针对于连续采样模式和有限点采样模式，在单点采样模式下，它总是为 0。

nMaxAvailSampsPerChan

自启动采样后，曾经出现过的最多的有效点数（Available Samples Per Channel）。比如在某一时刻 nAvailSampsPerChan=200，则该状态值就会等于 200，只要过后 nAvailSampsPerChan 永远小于 200，则该状态值就永远等于 200，除非 nAvailSampsPerChan 后来又超过 200，比如有个一次 350，则该状态值就保持在 350，依此类推。该状态值的作用是为了验证程序的整体效率而提供的。该状态值在采样任务长期运行过程中越小越好，则表示应用程序的读取效率和处理效率都很高，设备采样溢出丢点的可能性几乎为 0。如果此值比较贴近于 nBufSampsPerChan（即每通道缓冲区点数），那么溢出的可能性就比较大了。如果超越了 nBufSampsPerChan，则意味着采样任务已经发生过溢出了，其溢出次数则可以通过 nHardOverflowCnt 和 nSoftOverflowCnt 观察到。这个状态值的监测主要针对于连续采样模式，对于有限点和单点采样无监测意义。

nBufSampsPerChan

采样任务支持的每通道缓冲区点数（Samples per channel in task buffer）。表示在任务缓冲中，每通道最多可容量的数据点数。在有限点采样模式下，其每通道缓冲区点数直接由 AI 参数 AIParam.nSampsPerChan 决定。在连续采样模式下，采样任务会根据采样参数中的 nSampsPerChan，nSampChanCount 以及 nSampleRate 来决定使用缓冲区的大小，并由 nBufSampsPerChan 状态值得到其大小。单点采样模式，该状态值始终为 0

nSampsPerChanAcquired

自启动采样任务后，每通道已经采样过的点数（Samples Acquired Per Channel）。注意此状态值是 64Bit 的。

nHardOverflowCnt

硬件溢出计数（Hardware Overflow Count）。在启动采样任务后，绝大多数情况下，设备中的硬件缓存是不会溢出。但如果因为某种原因，如计算机系统极度繁忙或应用程序处理不当，效率不高，则有可能会引起硬件缓存溢出，则该计数器就会自动加 1，之后又快速地读走了缓存中的采样数据，

那么缓冲区又为不溢出状态，如果之后又溢出了，则该计数器又会自动加 1。如果用户重新启动采样，则该计数器自动清零。因此，该计数信息是作为分析采集应用软件设计是否高效，计算机系统是否高效提供了有利的参考信息。对于软件按需单点采样无任何意义。

nSoftOverflowCnt

软件溢出计数（Software Overflow Count）。在启动采样后，绝大多数情况下，设备中的软件缓存是不会溢出。但如果因为某种原因，如计算机系统极度繁忙或应用程序处理不当，效率不高，则有可能会引起软件缓存溢出，则该计数器就会自动加 1，之后又快速地读走了缓存中的采样数据，那么缓冲区又为不溢出状态，如果之后又溢出了，则该计数器又会自动加 1。如果重新启动采样，则该计数器自动清零。因此，该计数器值是作为分析应用软件设计是否高效，计算机系统是否高效提供了有利的参考信息。这个计数信息主要服务于连续采样模式。对于有限点采样和单点采样没有任何意义。

nInitTaskCnt

调用 AI_InitTask()的次数，用于检测初始化采样任务与释放采样任务是否前后匹配，如该计数值始终比 nReleaseTaskCnt 大 1，则表示每调用一次 AI_InitTask()后就会相应的调用 AI_ReleastTask()一次。

nReleaseTaskCnt

调用 AI_ReleaseTask()的次数。原理同上。

nStartTaskCnt

调用 AI_StartTask()的次数。用于检测启动采样任务与停止采样任务是否前后匹配，如该计数值始终比 nStopTaskCnt 大 1，则表示每调用一次 AI_StartTask()后就会相应的调用 AI_StopTask()一次。

nStopTaskCnt

调用 AI_StopTask()的次数。原理同上。

nTransRate

设备传输速率（Transfer Rate），单位：点/秒(P/S)。它反应了在 AI 采样过程中，实时传输 AI 采样数据的平均秒速度，即每秒传输了多少点的数据（它是指所有采样通道的数据传输速率）。比如设定的单个通道采样速率(fSampleRate)为 125000sps，采样通道数(nSampChanCount)为 4，则总采样速率为 500000sps（125000*4），那么正常情况下，该状态值应等于 500000 左右。因此该状态值也是为判断系统性能提供的有利参考信息。

nReserved0-4

保留字段(未作定义)。

相关函数： AI_GetStatus()

4.3 AI_MAIN_INFO (AI 主要信息结构体)

Visual C++ :

typedef struct _ AI_MAIN_INFO

```
{
    U32 nChannelCount;
    U32 nSampRangeCount;
    U32 nSampleGainCount;
    U32 nCouplingCount;
    U32 nImpedanceCount;
    U32 nDepthOfMemory;
    U32 nSampResolution;
    U32 nSampCodeCount;
    U32 nTrigLvlResolution;
    U32 nTrigLvlCodeCount;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
} AI_MAIN_INFO, *PAI_MAIN_INFO;
```

nChannelCount

物理通道数量(Channel Count)。

nSampRangeCount

采样范围挡位数量(Sample Range Count)。

nSampleGainCount

采样增益挡位数量(Sample Gain Count)。

nCouplingCount

耦合方式挡位数量(Coupling Count)。

nImpedanceCount

阻抗挡位数量(Impedance Count)。

nDepthOfMemory

各板载通道内存容量深度(Memory Depth)，单位：点数。

nSampResolution

采样分辨率(Sample Resolution)(如=8 表示 8Bit; =12 表示 12Bit; =14 表示 14Bit; =16 表示 16Bit)

nSampCodeCount

采样编码数量(Sample Code Count)(如 256, 4096, 16384, 65536)

nTrigLvlResolution

触发电平(Trigger Level Resolution)分辨率(如=8 表示 8Bit; =12 表示 12Bit; =16 表示 16Bit)

nTrigLvlCodeCount

触发电平编码数量(Trigger Level Code Count) (如 256, 4096)

nReserved0-3

保留字段(未作定义)

相关函数： AI_GetMainInfo()

4.4 AI_VOLT_RANGE_INFO (AI 电压范围信息结构体)

Visual C++ :

```
typedef struct _ AI_VOLT_RANGE_INFO
{
    U32 nSampleRange;
    U32 nReserved0;
    F64 fMaxVolt;
    F64 fMinVolt;
    F64 fAmplitude;
    F64 fHalfOfAmp;
    F64 fCodeWidth;
    F64 fOffsetVolt;
    F64 fOffsetCode;
    char strDesc[16];

    U32 nPolarity;
    U32 nCodeCount;
    I32 nMaxCode;
    I32 nMinCode;

    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AI_SAMP_RANGE_INFO, *P AI_SAMP_RANGE_INFO;
```

nSampleRange

当前采样范围索引号（Sample Range Index）

nReserved0

保留字段

fMaxVolt

采样范围的上限电压值(Max Voltage)，单位：伏(V)。

fMinVolt

采样范围的下限电压值(Min Voltage)，单位：伏(V)。

fAmplitude

采样范围的幅度值，单位：伏(V)。它也可以由 fMaxVolt – fMinVolt 得到。

fHalfOfAmp

幅度的二分之一(Half Of Amplitude)，单位：伏(V)。它也可以由 fAmplitude/2 得到。

fCodeWidth

编码宽度(Code Width)。如果幅度值为 20V，且分辨率为 12Bit(即总的 LSB 个数为 4096)，那么 fCodeWidth 应为 20/4096，即约等于 0.00488 伏。

fOffsetVolt

偏移电压,单位:伏(V),一般用于零偏校准(本设备无效)。

fOffsetCode

偏移码值,一般用于零偏校准,它代表的电压值等价于 fOffsetVolt(本设备无效)。

strDesc[16]

关于采样范围的字符描述信息(Description String)，如"±10V", "0-10V"等。

nPolarity

AI 采样范围的极性。

nPolarity 选项(常量名)	**常量值**	**功能定义**	**备注**
AI_POLAR__BIPOLAR	0	双极性，即指正负电压均可输入	默认值
AI_POLAR__UNIPOLAR	1	单极性，即指只能正电压可输入	

nPolarity

原码数量

nMaxCode

原码最大值

nMinCode

原码最小值

nReserved1-3

保留字段。

相关函数： AI_GetRangeInfo()

4.5 AI_SAMP_GAIN_INFO (AI 采样增益信息结构体)

Visual C++:

```
typedef struct _AI_VOLT_GAIN_INFO
{
    U32 nSampleGain;
    U32 nReserved0;
    F64 fAmpFactor;
    char strDesc[16];

    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AI_VOLT_GAIN_INFO, *PAI_VOLT_GAIN_INFO;
```

nSampleGain

当前采样增益索引号(Sample Gain Index) 。

nReserved0

保留字段。

fAmpFactor

采样增益放大倍数(Amplitude Factor)。

strDesc[16]

关于采样增益的字符描述信息(Description String)，,如"10 倍", "100 倍"等。

nReserved0-4

保留字段。

相关函数： AI_GetRangeInfo()

4.6 AI_SAMP_RATE_INFO (AI 采样速率信息结构体)

Visual C++ :

```
typedef struct _ AI_SAMP_RATE_INFO
{
    F64 fMaxRate;
    F64 fMinRate;
    F64 fTimerBase
    U32 nDivideMode;
    U32 nRateType;
```

```
    U32 nReserved0;
    U32 nReserved1;
} AI_SAMP_RATE_INFO, *PAI_SAMP_RATE_INFO;
```

fMaxRate

AI 单通道最大采样率(Max Rate)，单位：点/秒(sps)。多通道时各通道平分最大采样率

fMinRate

AI 单通道最小采样率(Min Rate)，单位：点/秒(sps)。多通道时各通道平分最小采样率

fTimerBase

时钟基准(Timer Base)，即板上使用的晶振大小，单位：赫兹(Hz)。

nDivideMode

分频模式(Divide Mode)，0=整数分频(INTDIV), 1=DDS 分频(DDSDIV)。

nRateType

速率类型,指 fMaxRate 和 fMinRate 的类型, =0:表示为所有采样通道的总速率, =1:表示为每个采样通道的速率。

nReserved0-1

保留字段。

相关函数： AI_GetRateInfo()

4.7 AO_PARAM（AO 工作参数结构体）

Visual C++ :

```
typedef struct _AO_CH_PARAM
{
    U32 bChannelEn;
    U32 nSampleRange;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
}
AO_CH_PARAM, *PAO_CH_PARAM;

typedef struct _AO_PARAM
{
    USB5630_AO_CH_PARAM CHParam[4];
```

```
    U32 nSampleMode;
    U32 nSampsPerChan;
    F64 fSampleRate;
    U32 nClockSource;
    U32 bClockOutput;
    U32 bRegenModeEn
    U32 nReserved0;

    U32 bDTriggerEn;
    U32 nDTriggerDir;
    U32 nDTrigChannel;
    U32 bATriggerEn;
    U32 nATriggerDir;
    F32 fTriggerLevel;
    U32 nTriggerSens;
    I32 nDelaySamps;
    U32 nReserved1;

    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
    U32 nReserved5;
} AO_PARAM, *PAO_PARAM;
```

4.7.1 AO_CH_PARAM(AI 通道参数结构体)

bChannelEn

AO 物理通道使能(Channel Enable)，取值范围[0, 1]，即 AO0-AO1，只有使能的通道才能输出波形。如 bChannelEn[0]=TRUE, bChannelEn[1]=FALSE 时，则表示 AO0 允许输出，AO1 禁止输出。

nSampleRange

AO 采样范围(Sample Range)，取值范围为[0, 5]，本设备 AO 采样分辨率为 12Bit，因此各采样范围和编码宽度如下表：

常量名	常量值	采样范围
AO_SAMPRANGE_0_P5V	0	0-5V
AO_SAMPRANGE_0_P10V	1	0-10V
AO_SAMPRANGE_0_N5_P5V	2	±5V
AO_SAMPRANGE_N10_P10V	3	±10V
AO_SAMPRANGE_N2D5_P2D5V	4	±2.5V
AO_SAMPRANGE_N2D5_P7D5V	5	-2.5~7.5V

nReserved0-3

保留字段

4.7.2 AO_PARAM(AI 工作参数结构体）

CHParam[4]

通道组(Channel Parameter)，共 4 个单元，分别控制 4 个采样通道的选择。每个单元决定要采集的物理通道号，采样范围等工作参数。

nSampleMode

AI 采样模式(Sample Mode)，具体定义见下表：

常量名	常量值	功能定义	备注
AO_SAMPMODE_ONE_DEMAND	0	软件按需单点采样	
AO_SAMPMODE_ONE_HWTIMED	1	硬件定时单点采样	本设备不支持
AO_SAMPMODE_FINITE	2	有限点采样	
AO_SAMPMODE_CONTINUOUS	3	连续采样	

软件按需单点采样模式：就是调用 AO_StartTask()后，AO 任务只是就绪，但并不实际采样数据，而要等到每次软件调用 AO_WriteAnalog()或 AO_WriteBinary()函数时任务才开始实际采样，且每个通道仅采样一个点的数据，并以最快的速度返回，此时该点数据立即输出到设备上。这种采样模式主要针对简单采样或采样实时性要求较高，数据量很少，且时间不确定的应用中，比如应用在对时间边界没有严格要求的 PID，PLC 等快速伺服闭环控制系统。不支持触发和外时钟。

硬件定时单点采样模式：在调用 AO_StartTask()后，AI 采样任务就会按照 AOParam.fSampleRate 设定的速率定时地，不断的为每个通道采样单点数据，每次调用 AO_WriteAnalog()或 AO_WriteBinary()时也为每个通道仅写入一个点的数据到设备缓冲区中，且以最快的速度返回，此时该数据并没有实际输出，而是要等到下一个采样时钟边沿上才将该点数据实际输出到设备上。这种采样模式主要针对简单采样或采样实时性要求较高，数据量很少，且时间有严格要求的确定的应用中，比如应用在对时间边界有严格要求的 PID，PLC 等快速伺服闭环控制系统。不支持触发和外时钟(但本设备不支持)。

有限点采样模式：按照设定好的采样速率和触发条件，触发模式等参数进行定长时间的、限制点数的、连续的、等间隔的波形数据采集。在启动采样任务后，达到触发条件并采样完成指定点数的数据后，采样任务就会自动停止。这个功能主要是应用在频繁捕捉触发事件、有点数限制和时间长度限制、尽可能还原外界信号的场合。

连续采样模式：按照设定好的采样速率和触发条件，触发模式等参数进行长时间的、不限点数的、连续的、等间隔的波形数据采集，在启动采样任务后，只要不软件停止采样任务，其采样任务是永远不会终止的。这个功能主要是应用在不限制点数和时间长度的，且不丢点的，尽可能还原外界信号的场合。

nSampsPerChan

AO 每通道待读取点数(Samples Per Channel)。

单点采样模式下，该参数无意义；

有限点采样模式下，该参数表示每通道采样点数。取值范围为[2，1024*1024*16]样点，最大取值还要受制于系统可用内存，采样通道数等；

连续采样模式下，决定着触发采样事件 hSampEvent 时的点数条件。比如指定该参数值为 1024 点，则每采样到不小于 1024 点时就会触发采样事件 hSampEvent，它也决定着每次调用 AO_WriteAnalog()或 AO_WriteBinary()时能最快返回的点数边界（请注意：是不小于 nSampsPerChan，意思是不可能正好等于 nSampsPerChan 时就能得到事件通知，而是通常在超过 nSampsPerChan 时才能得到事件通知）。即该参数的取值大小决定着每两次读到数据的时间间隔，也就是实时响应度。点数越小，实时响应就越高，反之越低。但不能为了一味的追求实时响应度就将该参数的值设得很小，

这个还要看采样速率的高低，如果采样速率很高，而 nSampsPerChan 的值很小，则可能造成任务负担过重而发生缓冲区溢出，以致造成丢点现象的发生。建议实时响应度不低于 20 个毫秒是比较合适的。比如每通道采样速率为 100Ksps，即 10 微秒一个点，则 nSampsPerChan 不小于 2000 个点是比较合适的。该参数的取值范围为[2, 1024*1024]，具体还要受制于系统可用内存和采样通道数。

fSampleRate

AO 采样速率(Sample Rate)，单位：每秒样点 sps(sample per second)，它指每个采样通道的采样速率，决定了单个采样通道每秒钟采样的点数。而每个采样点的周期则由 fSampleRate 求倒数取得，单位：秒(S)。它的最小取值等于 1sps，而最大取值则由设备最高采样率(500000sps)决定。。

nClockSource

时钟源(Clock Source)。本设备的 AO 采样时钟支持本地时钟源(LOCAL)和外部时钟源(CLKIN)。本地时钟源即指板上晶振时钟源(OSCCLK)。外部时钟源则来自于由连接器 J1 上的 CLKIN 复用输入，因此当选择为外时钟后初始化 AO_InitTask() 时会将 DIO2 的方向强制置为输入。该参数具体定义同 AI 中的 nClockSource。

nClockOutput

本地采样时钟是否输出（Clock Output Enable）。=TRUE：表示允许由 fSampleRate 决定的采样时钟会直接输出到 J1 的 CLKOUT， =FALSE：禁止输出。

bRegenModeEn

波形重生成模式允许(在连续采样模式中有效), =TRUE:只是对开始前写入任务中的波形点数据进行循环重复生成，=FALSE:禁止重生成模式，在开始后，还要不断的往任务中写入新的波形数据。

bDTriggerEn

数字量触发 DTR 允许(Digital Trigger Enable)。如果等于 TRUE，表示外部数字量触发输入信号允许进入 AO 的触发系统，如果等于 FALSE，表示不允许(即禁用)。触发信号由连接器 CN2 上的 DTR 输入。

nDTriggerDir

数字量触发极性(Digital Trigger Direction)。它的取值如下表：

常量名	常量值	功能定义	备注
AO_TRIGDIR_FALLING	0	下降沿触发	默认值
AO_TRIGDIR_RISING	1	上升沿触发	
AO_TRIGDIR_CHANGE	2	变化触发(上或下边沿触发)	

bATriggerEn

模拟量触发 ATR 允许(Analog Trigger Enable)。如果等于 TRUE，表示模拟通道触发输入信号 ATR 允许进入 AO 的触发系统，如果等于 FALSE，表示不允许(即禁用)。

nATriggerDir

模拟量触发极性(Analog Trigger Direction)。它的取值如下表：

常量名	常量值	功能定义	备注

AO_TRIGDIR_FALLING	0	下降沿触发	默认值
AO_TRIGDIR_RISING	1	上升沿触发	
AO_TRIGDIR_CHANGE	2	变化触发(上或下边沿触发)	

fTriggerLevel

AO 采样通道的触发电平(Trigger Level)，单位：伏(V)，分辨率 12Bit。取值范围要受到 nSampleRange 参数所选择的模拟量输入范围档的限定，如下表所示：

常量名	常量值	采样范围
AO_SAMPRANGE_0_P5V	0	0-5V
AO_SAMPRANGE_0_P10V	1	0-10V
AO_SAMPRANGE_0_N5_P5V	2	±5V
AO_SAMPRANGE_N10_P10V	3	±10V
AO_SAMPRANGE_N2D5_P2D5V	4	±2.5V
AO_SAMPRANGE_N2D5_P7D5V	5	-2.5~7.5V

这与 AI 的触发电平 fTriggerLevel 共用一个触发源，因此 AO 这边改变触发电平，那么 AI 那边也会同时改成相同的触发电平。

nTriggerSens

AO 触发灵敏度（Trigger Sense)，单位：微秒(uS)。取值范围为[0, 1638]，它的实际功能是为避免触发信号中较高频的噪声分量也进入触发系统而设定的一种时间门限，凡是触发信号跳变后稳定保持时间不小于该门限时间，则进入触发系统，否则不进入而被屏蔽掉。此参数仅对数字量和模拟量触发均有效。跳变保持时间：指的是触发信号由一个状态跳变至另一个状态时所能保持的时间。举例说明，一个数字量触发信号原来是低电平，然后跳变至高电平，保持 10 微秒后又回到低电平，即跳变保持时间指的就是高电平在这个瞬间保持的时间 10 微秒。这个时间越短越有可能是高频噪声，因此需要这个参数加以控制或过滤，以避免噪声信号产生误触发。

nDelaySamps

触发延迟点数（Delay Samples）。取值范围 32 位有效[0, 4294967295]。其值等于 0 时为后触发（Post Trigger)；其值大于 0 时为正延迟触发(Delay Trigger)。就是在开始触发产生时，再延迟若干个采样点后再记录数据，其延迟的点数就是由 nDelaySamps 指定，而单个点的时间周期由 nSampleRate 指定的每通道采样速率决定的。比如 nDelaySamps=100，nSampleRate=1000Hz（即每采样点周期为 1 毫秒)，则意味着在启动采样任务后，当发生开始触发时再等待 100 毫秒（1 毫秒*100）才实际进入数据采样与记录阶段。在有些应用中，用户关注的焦点信号是在触发事件之后的一段时间才发生，那么这个功能就是满足此种应用的。

nReserved1-5

保留字段(未作定义)，可强制赋 0

相关函数： DEV_Create() AO_InitTask() AO_LoadParam() AO_SaveParam() AO_ResetParam DEV_Release()

4.8 AO_STATUS（AO 工作状态信息结构）

Visual C++ :

```
typedef struct _AO_STATUS
{
    U32 bTaskDone;
    U32 bTriggered;

    U32 nTaskState;
    U32 nAvailSampsPerChan;
    U32 nMaxAvailSampsPerChan;
    U32 nBufSampsPerChan;
    U64 nSampsPerChanAcquired;

    U32 nHardOverflowCnt;
    U32 nSoftOverflowCnt;
    U32 nInitTaskCnt;
    U32 nReleaseTaskCnt;
    U32 nStartTaskCnt;
    U32 nStopTaskCnt;
    U32 nTransRate;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AO_STATUS, *PAO_STATUS;
```

此结构体主要用于查询 AO 的工作状态信息，AO_GetStatus()函数使用此结构体来实时取得 AO 的工作状态信息，以便同步数据采样和处理过程。

bTaskDone

AO 采样任务完成标志(Task Done)。=TRUE:表示采样任务已结束, =FALSE:表示采样任务正在进行中。在设备上电之初或执行 AO_StopTask()函数后其值为 TRUE，当执行 AO_StartTask()函数后为 FALSE。在有限点采样任务中，如果达到触发条件和采样点数后，任务会自动停止，此标志会被自动置成 TRUE。在连续采样任务中，只有调用 AO_StopTask()函数手动停止采样任务，此标志才会被置成 TRUE。

bTriggered

AO 触发标志。=TRUE:表示已被触发, =FALSE:表示未被触发或等待触发。在设备上电之初或当执行 AO_StartTask()后其值为 FALSE。在被正常触发后自动变为 TRUE。执行 AO_StopTask()后其值不变。

nSampTaskState

采样任务状态(Sample Task State), 若等于 1 表示正常, 其它值表示有异常情况。

nAvailSampsPerChan

每通道有效点数，表示采样任务缓冲中的可写入数据点数（Available Samples Per Channel）。在初始化采样任务后，nAvailSampsPerChan 会被复位至 nBufSampsPerChan 的值。每写入一个采样数据则 nAvailSampsPerChan 会自动减 1，直至 0 值。每输出一个数据到设备时该状态值会自动加 1，直至 nBufSampsPerChan。在写入数据前应判断此值，如果它小于或等于参数 nWriteSampsPerChan 的时就调用 AO_WriteAnalog()或 AO_WriteBinary()时，则写数据函数会自动进入超时等待睡眠状态，直至可写点数达到指定写入点数 nWriteSampsPerChan 才会返回，如果等待时间超过 fTimeout 的值，也会返回 FALSE，并置超时错误状态。如果 nAvailSampsPerChan 大于 nWriteSampsPerChan 的时候调用 AO_WriteAnalog()或 AO_WriteBinary()时，则写数据函数会迅速写入指定点数的数据并返回 TRUE。在连续采样和非重生成模式中，如果 nAvailSampsPerChan 的值等于 0，表示写入的数据点数刚好填满任务缓冲区，不能再写入数据，否则存在着缓冲区溢出的风险（即造成断波）。如果 nAvailSampsPerChan 的值大于或等于 nBufSampsPerChan，表示任务缓冲区已经被腾空，存在着数据下溢的风险（即造成断波），解决的办法是应迅速写入后续数据，保证缓冲区不被任务腾空。其下溢次数可以通过 nHardOverflowCnt 和 nSoftOverflowCnt 观察到。这个状态值的监测主要针对于连续采样模式和有限点采样模式，在单点采样模式下，它总是为 0。

nMaxAvailSampsPerChan

自启动采样后，曾经出现过的最大的可有效点数（Available Samples Per Channel）。比如在某一时刻 nAvailSampsPerChan=200，则该 nMaxAvailSampsPerChan 就会等于 200，只要过后 nAvailSampsPerChan 永远小于 200，则该状态值就永远等于 200，除非 nAvailSampsPerChan 后来又超过 200，比如有个一次 350，则该状态值就保持在 350，依此类推。它的值越小反映了任务缓冲区越接近满状态（因为没有更多的空闲位置可写入新数据），此时越不易发生输出缓冲下溢而断波的可能。反之越大，则反映了任务缓冲区越接近于空状态（因为需要更多的空闲位置需要及时写入新数据），此时越容易发生输出缓冲下溢而断波的可能。因此该状态值的作用是为了验证程序的整体效率而提供的。该状态值在采样任务长期运行过程中越小越好，则表示应用程序的写入效率和处理效率都很高，设备采样下溢丢点的可能性几乎为 0。如果此状态值等于或大于了 nBufSampsPerChan，则意味着采样任务已经发生过下溢了，其溢出次数则可以通过 nHardUnderflowCnt 和 nSoftUnderflowCnt 观察到。这个状态值的监测主要针对于连续非重生成模式，对于有限点和单点采样无监测意义。

nBufSampsPerChan

采样任务支持的每通道缓冲区点数（Samples per channel in task buffer）。表示在任务缓冲中，每通道最多可容量的数据点数。在有限点采样模式下，其每通道缓冲区点数直接由 AO 参数 AOParam.nSampsPerChan 决定。在连续采样模式下，采样任务会根据采样参数中的 nSampsPerChan，nSampChanCount 以及 nSampleRate 来决定使用缓冲区的大小，并由 nBufSampsPerChan 状态值得到其大小。对于单点采样模式，该状态值始终为 0。

nSampsPerChanAcquired

自启动采样任务后，每通道已经采样过的点数（Samples Acquired Per Channel）。注意此状态值是 64Bit 的。

nHardUnderflowCnt

硬件下溢计数（Hardware Underflow Count）。在启动采样任务后，绝大多数情况下，设备中的硬件缓存是不会溢出。但如果因为某种原因，如计算机系统极度繁忙或应用程序处理不当，效率不高，任务没有及时往设备中写入数据则有可能会引起硬件缓存溢出，则该计数器就会自动加 1，之后用户又快速地读走了缓存中的采样数据，那么缓冲区又为不溢出状态，如果之后又溢出了，则该计数器又会自动加 1。如果用户重新启动采样，则该计数器自动清零。因此，该计数信息是作为分析采集应用软件设计是否高效，计算机系统是否高效提供了有利的参考信息。对于软件按需单点采样无任何意义。

nSoftUnderflowCnt

软件下溢计数（Software Underflow Count）。在启动采样任务后，绝大多数情况下，设备中的软件缓存是不会下溢。但如果因为某种原因，如计算机系统极度繁忙或应用程序处理不当，效率不高，没有及时往任务缓冲区中写入新的数据则有可能会引起软件缓存下溢，则该计数器就会自动加 1，之后又快速地读走了缓存中的采样数据，那么缓冲区又为不溢出状态，如果之后又溢出了，则该计数器又会自动加 1。如果重新启动采样，则该计数器自动清零。因此，该计数器值是作为分析应用软件设计是否高效，计算机系统是否高效提供了有利的参考信息。这个计数信息主要服务于连续采样和非重生成模式。对于有限点采样和单点采样没有任何意义。

nInitTaskCnt

调用 AO_InitTask()的次数，用于检测初始化采样任务与释放采样任务是否前后匹配，如该计数值始终比 nReleaseTaskCnt 大 1，则表示每调用一次 AO_InitTask()后就会相应的调用 AO_ReleastTask()一次。

nReleaseTaskCnt

调用 AO_ReleaseTask()的次数。原理同上。

nStartTaskCnt

调用 AO_StartTask()的次数。用于检测启动采样任务与停止采样任务是否前后匹配，如该计数值始终比 nStopTaskCnt 大 1，则表示每调用一次 AI_StartTask()后就会相应的调用 AI_StopTask()一次。

nStopTaskCnt

调用 AO_StopTask()的次数。原理同上。

nTransRate

设备传输速率（Transfer Rate），单位：点/秒(P/S)。它反应了在 AO 采样过程中，实时传输 AO 采样数据的平均秒速度，即每秒传输了多少点的数据（它是指所有采样通道的数据传输速率）。比如设定的单个通道采样速率(fSampleRate)为 100000sps，采样通道数(nSampChanCount)为 2，则总采样速率为 200000sps（100000*2），那么正常情况下，该状态值应等于 200000 左右。因此该状态值也是为判断系统性能提供的有利参考信息。

nReserved0-4

保留字段(未作定义)。

相关函数：AO_GetStatus()

4.9 AO_MAIN_INFO (AO 主要信息结构体)

Visual C++ :

```
typedef struct _ AO_MAIN_INFO
{
    U32 nChannelCount;
    U32 nSampRangeCount;
    U32 nSampleGainCount;
    U32 nCouplingCount;
    U32 nImpedanceCount;
    U32 nDepthOfMemory;
    U32 nSampResolution;
    U32 nSampCodeCount;
    U32 nTrigLvlResolution;
    U32 nTrigLvlCodeCount;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
} AO_MAIN_INFO, *PAO_MAIN_INFO;
```

nChannelCount

物理通道数量(Channel Count)。

nSampRangeCount

采样范围挡位数量(Sample Range Count)。

nSampleGainCount

采样增益挡位数量(Sample Gain Count)。

nCouplingCount

耦合方式挡位数量(Coupling Count)。

nImpedanceCount

阻抗挡位数量(Impedance Count)。

nDepthOfMemory

各板载通道内存容量深度(Memory Depth)，单位：点数。

nSampResolution

采样分辨率(Sample Resolution)(如=8 表示 8Bit; =12 表示 12Bit; =14 表示 14Bit; =16 表示 16Bit)。

nSampCodeCount

采样编码数量(Sample Code Count)(如 256, 4096, 16384, 65536)。

nTrigLvlResolution

触发电平(Trigger Level Resolution)分辨率(如=8 表示 8Bit; =12 表示 12Bit; =16 表示 16Bit)

nTrigLvlCodeCount

触发电平编码数量(Trigger Level Code Count) (如 256, 4096)。

nReserved0-3

保留字段(未作定义)。

相关函数： AO_GetMainInfo()

4.10 AO_VOLT_RANGE_INFO (AO 采样范围信息结构体)

Visual C++ :

```
typedef struct _ AO_VOLT_RANGE_INFO
{
    U32 nSampleRange;
    U32 nReserved0;
    F64 fMaxVolt;
    F64 fMinVolt;
    F64 fAmplitude;
    F64 fHalfOfAmp;
    F64 fCodeWidth;
    F64 fOffsetVolt;
    F64 fOffsetCode;
    char strDesc[16];

    U32 nPolarity;
    U32 nCodeCount;
    I32 nMaxCode;
    I32 nMinCode;

    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AO_VOLT_RANGE_INFO, *P AO_VOLT_RANGE_INFO;
```

nSampleRange

当前采样范围索引号（Sample Range Index）。

nReserved0

保留字段

fMaxVolt

采样范围的上限电压值(Max Voltage)，单位：伏(V)。

fMinVolt

采样范围的下限电压值(Min Voltage)，单位：伏(V)。

fAmplitude

采样范围的幅度值，单位：伏(V)。它也可以由 fMaxVolt – fMinVolt 得到。

fHalfOfAmp

幅度的二分之一(Half Of Amplitude)，单位：伏(V)。它也可以由 fAmplitude/2 得到。

fCodeWidth

编码宽度(Code Width)。如果幅度值为 20V，且分辨率为 12Bit(即总的 LSB 个数为 4096)，那么 fCodeWidth 应为 20/4096，即约等于 0.00488 伏。

fOffsetVolt

偏移电压(Offset Volt),单位:伏(V),一般用于零偏校准(暂时未用)。

fOffsetCode

偏移码值,一般用于零偏校准,它代表的电压值等价于 fOffsetVolt(本设备无效)。

strDesc[16]

关于采样范围的字符描述信息(Description String)，如"±10V", "0-10V"等。

nPolarity

AI 采样范围的极性。

nPolarity 选项(常量名)	常量值	功能定义	备注
AO_POLAR__BIPOLAR	0	双极性，即指正负电压均可输入	默认值
AO_POLAR__UNIPOLAR	1	单极性，即指只能正电压可输入	

nReserved1-4

保留字段

相关函数： AO_GetRangeInfo()

4.11 AO_SAMP_RATE_INFO (AO 采样速率信息结构体)

Visual C++ :

```
typedef struct _ AO_SAMP_RATE_INFO
{
```

```
    F64 fMaxRate;
    F64 fMinRate;
    F64 fTimerBase
    U32 nDivideMode;
    U32 nRateType;

    U32 nReserved0;
    U32 nReserved1;
} AO_SAMP_RATE_INFO, *PAO_SAMP_RATE_INFO;
```

fMaxRate

AI 最大采样率(Max Rate)，单位：点/秒(sps)。

fMinRate

AI 最小采样率(Min Rate)，单位：点/秒(sps)。

fTimerBase

时钟基准(Timer Base)，即板上使用的晶振大小，单位：赫兹(Hz)。

nDivideMode

分频模式(Divide Mode)，0=整数分频(INTDIV), 1=DDS 分频(DDSDIV)。

nRateType

速率类型，指 fMaxRate 和 fMinRate 的类型, =0:表示为所有采样通道的总速率, =1:表示为每个采样通道的速率。

nReserved0-1

保留字段。

相关函数： AO_GetRateInfo()

4.12 CTR_PARAM (CTR 计数器工作参数结构体)

Visual C++:

```
typedef struct _CTR_PARAM
{
    U32 nGateMode;
    U32 nInitValue;

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
```

} CTR_PARAM, *PCTR_PARAM;

nGateMode

门控方式(Gate Mode)，其取值范围[0，5]，具体定义见下表：

nPolarity 选项(常量名)	常量值	功能定义	备注
CTR_GATEMODE_POSITIVE_0	0	计数结束产生中断: GATE 高电平时计数，低电平时停止计数，计数时重新写入新的初值，按新值减法计数时，计数到 0 时 OUT 为 1；	默认值
CTR_GATEMODE_RISING_1	1	可编程单拍脉冲：GATE 上边沿触发计数，计数中出现 GATE 上升沿重新装入初值计数, 当写入初值时 OUT 为 1，当开始计数时 OUT 为 0, 减法计数时，计数到 0 时 OUT 为 1	
CTR_GATEMODE_POSITIVE_2	2	频率发生器：GATE 高电平时计数，低电平时停止计数，如果计数中改变初值，下次有效,计数期间 OUT 为 1, 减法计数时，计数到 0 后输出一个周期的 0，并重新装入计数值计数；	
CTR_GATEMODE_POSITIVE_3	3	方波发生器：GATE 高电平时计数，低电平时停止计数，如果计数中改变初值，下次有效, 计数期间 OUT 为 1, 减法计数时，计数到 0 后输出一个周期的 0，并重新装入计数值计数	
CTR_GATEMODE_POSITIVE_4	4	软件触发选通：GATE 高电平时计数，低电平时停止计数，如果计数中改变初值，本次有效,写入初值 OUT 为 1, 减法计数时，计数到 0 后输出一个周期的低电平信号；	
CTR_GATEMODE_RISING_5	5	硬件触发选通：GATE 上边沿触发计数，计数中出现 GATE 上升沿重新装入初值计数, 写入初值 OUT 为 1, 减法计数时，计数到 0 后输出一个周期的低电平信号；	

nInitValue

计数器初值（Initialize Value），32Bit 有效。

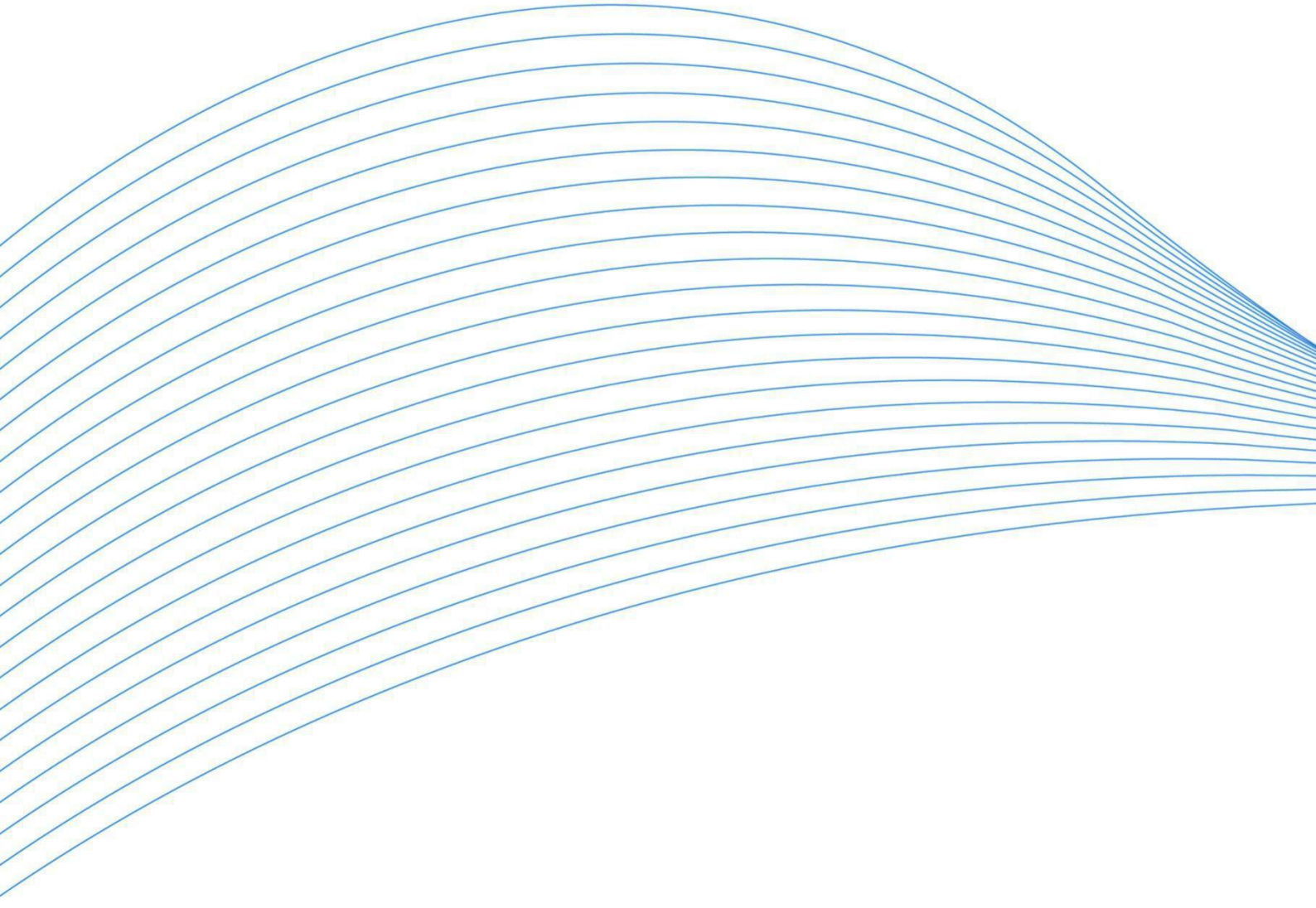
nReserved0-3

保留字段。

相关函数： CTR_InitTask() CTR_StartTask() CTR_ReadCounter()
CTR_StopTask() CTR_ReleaseTask()

5 修改历史

修改时间	版本号	修改内容
2017.6.15	V6.00.00	第一版



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