ACST2100

驱动程序使用手册

V6.00.00

ART Technology

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)	Long	LongInt
U32	无符号 32 位整型数	0 to 4294967295	unsigned int(long)	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 1844674407370955161	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“ACTS2100_”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++	ACTS2100.h	ACTS2100.lib	C:\ART\ACTS2100\Include
Microsoft Visual Basic	ACTS2100.bas	无	C:\ART\ACTS2100\Include
Borland C++ Builder	ACTS2100.h	ACTS2100.lib	C:\ART\ACTS2100\Include
Borland Delphi	ACTS2100.pas	无	C:\ART\ACTS2100\Include
NI LabVIEW	ACTS2100.vi	无	C:\ART\ACTS2100\Include
NI LabWindows/CVI	ACTS2100.h	ACTS2100.lib	C:\ART\ACTS2100\Include

注：（1）、ACTS2100.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）、ACTS2100RSV.h 是产品的保留头文件，为了凸现 ACTS2100.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 ACTS2100_32.dll 从 Windows\System32 中复制到安装包中；

（2）、将 ACTS2100.inf、ACTS2100.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 当前设备支持哪些功能

调用 GetMainInfo 函数获取当前设备主要结构本(MAIN_INFO)；成员 nAIChannelCount 为 AI 通道数，其值表示支持 AI 的通道数；成员 nAOChannelCount 为 AO 通道数，其值表示支持 AO 的通道数；成员 nCTRChannelCount 为 CTR 通道数，其值表示支持 CTR 的通道数；nDIOPortCfg 为各 Port 是否支持，nDIOPortCfg [0]为 1 时表示支持端口 0；其它参数后序介绍。

PCIE5630 此卡 64 路 AI、 4 路 AO、2 路 CTR、 4 个 Port 均支持，前 1 个 Port 只能为开关量、后 3 个 Port 也可做 PFI 使用;

后序的工作流程中都省略了 GetMainInfo，当然如果您清楚当前设备的信息可以不调用此函数。

PFI 使用详情请参考 3.2 章，Port 及 PFI 使用介绍

2.5 AI 单点采样模式

单点采样，就是在一次开始后，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。具体流程如图 2-4-1 所示：

(1) DEV_Create() 创建设备句柄；
(2) AI_InitTask() 初始化 AI 采集任务, 参数 nSampleMode=0；
(3) AI_StartTask() 开始 AI 采集任务；
(4) AI_SendSoftTrig ()发送软件触发事件；
(5) AI_ReadAnalog() 或者 AI_ReadBinary() 读取 AI 各采样通道单点数据；
(6) AI_StopTask() 停止 AI 采集任务；
(7) AI_ReleaseTask() 释放 AI 采集任务；
(8) DEV_Release() 释放设备句柄。

如果每次采集参数相同的情况下，可以在循环 3、4、5、6 进行，即可实现单点循环采样；

如果每次采集参数有所不同，则可以在 2、3、4、5、6、7 步间重复进行。

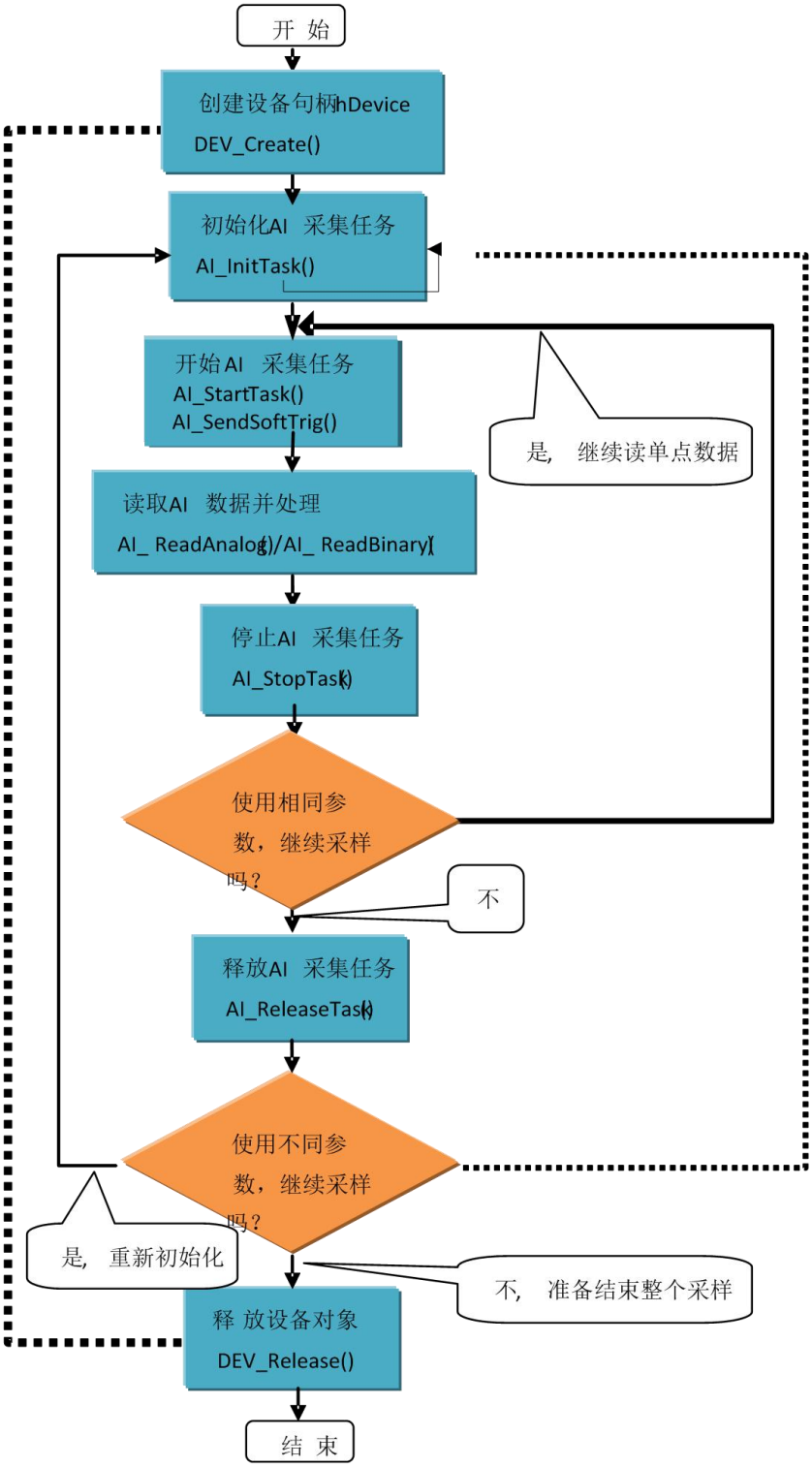


图2-4-1 AI 实时单点采样流程图例

2.6 AI 有限点采样模式

有限点采样模式，就是在一次开始后，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。无需等待采样完成 AI 启动就可以读数。具体流程如图 2-5-1 所示：

(1) DEV_Create()　创建设备句柄；

(2) AI_InitTask()　初始化 AI 采集任务, 注意参数 nSampleMode=2；

(3) AI_StartTask()　开始 AI 采集任务；

(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 采集任务；

(6) AI_ReleaseTask()　释放 AI 采集任务；

(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。

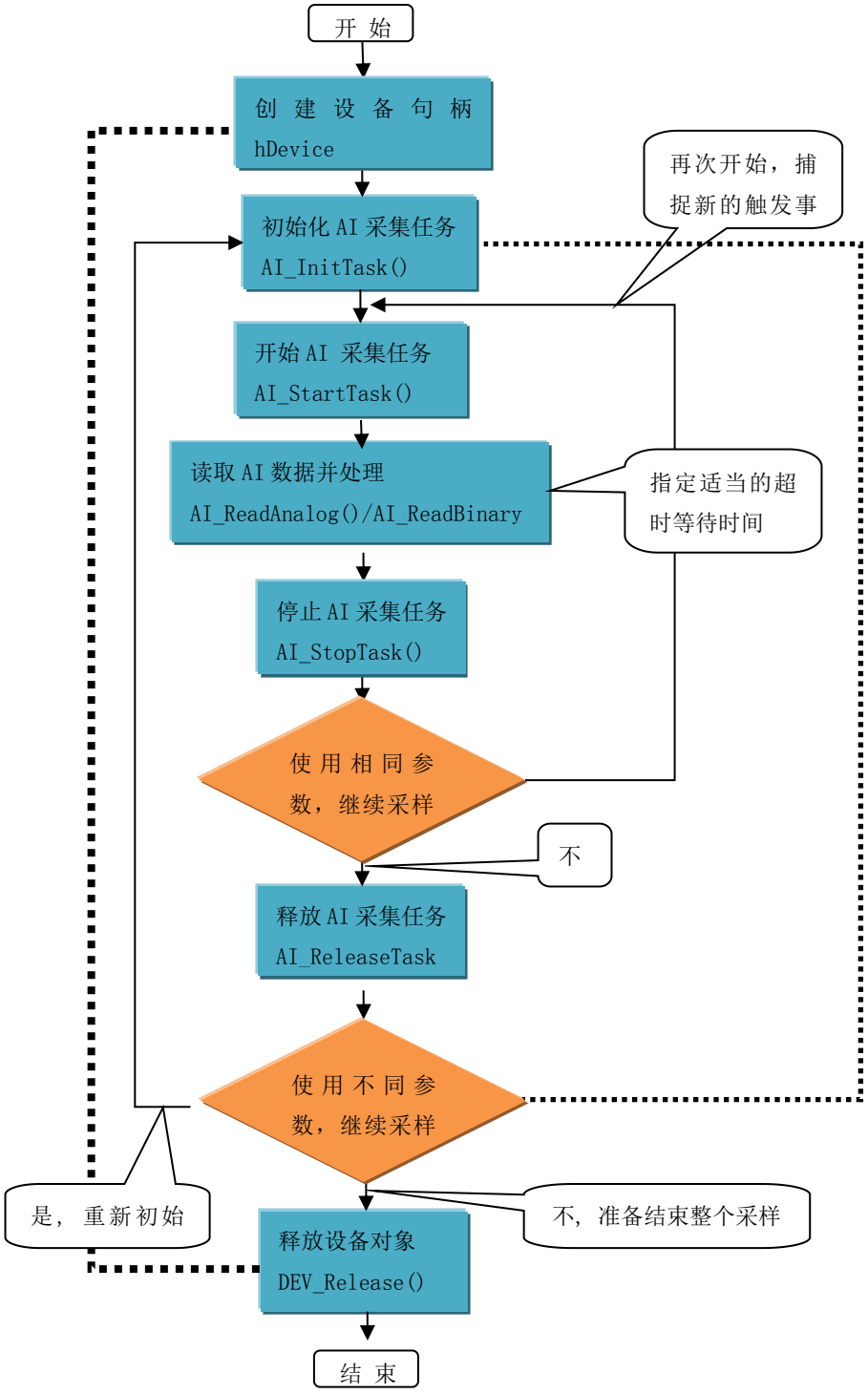


图 2-5-1　　AI 有限点采样流程图例

2.7 AI 连续采样模式

连续采样模式，就是在一次开始后，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。具体流程如图 2-6-1 所示：

（1）DEV_Create()　创建设备句柄；

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

（3）AI_StartTask()　开始 AI 采集任务；

（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 采集任务；

（6）AI_ReleaseTask() 释放 AI 采集任务；

（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。

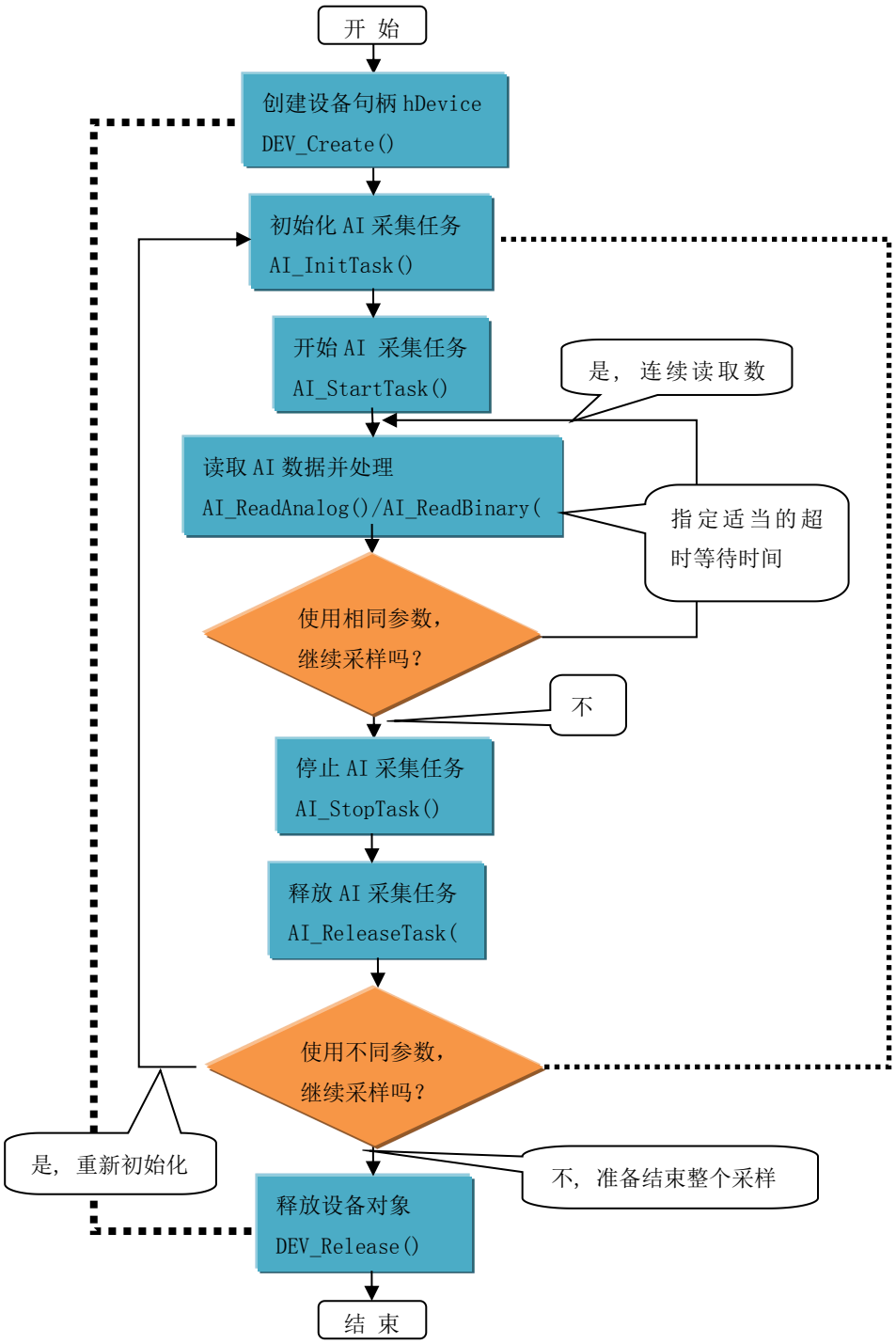


图 2-6-1　　AI 连续采样流程图例

上图虚线表示对称关系。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。AI_InitTask()和 AI_ReleaseTask()两个函数的对称关系是只有在 AI_ReleaseTask()之后才能再次 AI_InitTask()。

2.8 AO 单点采样模式

单点采样，就是在一次开始后，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 生成任务，参数 nSampleMode=0；

（3） AO_StartTask() 开始 AO 生成任务；

（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 生成任务；

（6） AO_ReleaseTask() 释放 AO 生成任务；

（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。

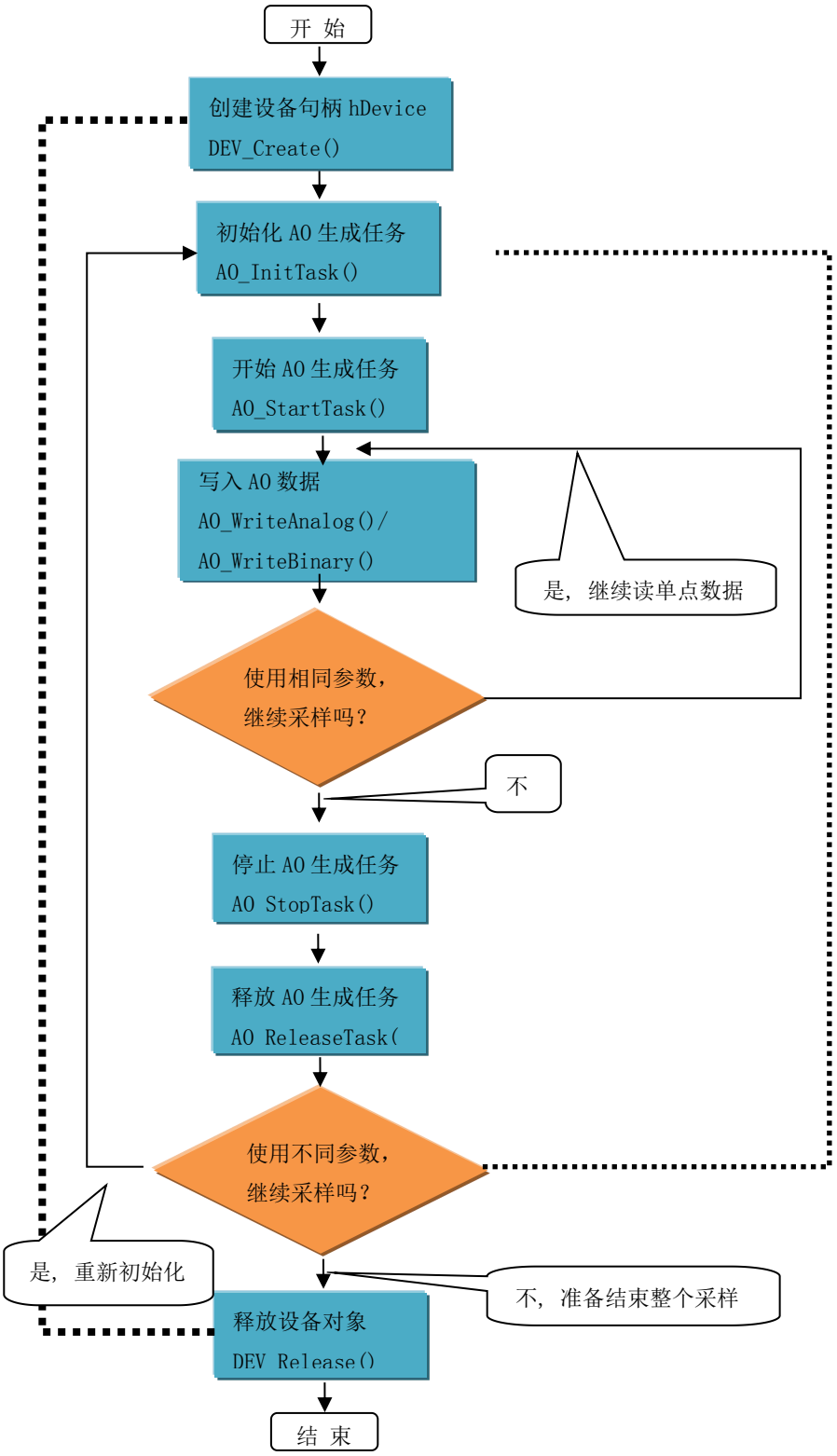


图 2-7-1　AO 实时单点采样流程图例

2.9　AO 有限点采样模式

有限点采样模式，就是在开始前，先写入指定点数的数据到任务中，开始后，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数据到任务中以待开始采样。

AO 有限点采样流程:

（1） DEV_Create() 创建设备句柄；

（2） AO_InitTask() 初始化 AO 生成任务，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() 开始 AO 生成任务；

（5） AO_GetStatus() 取得 AOStatus.bTaskDone 若等于 1 表示生成任务结束（或者调用 AO_WaitUntilTaskDone()）

（6） AO_StopTask() 停止 AO 生成任务；

（7） AO_ReleaseTask() 释放 AO 生成任务；

（8） 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、7 步间重复进行。请参考看下面流程图 2-8-1。

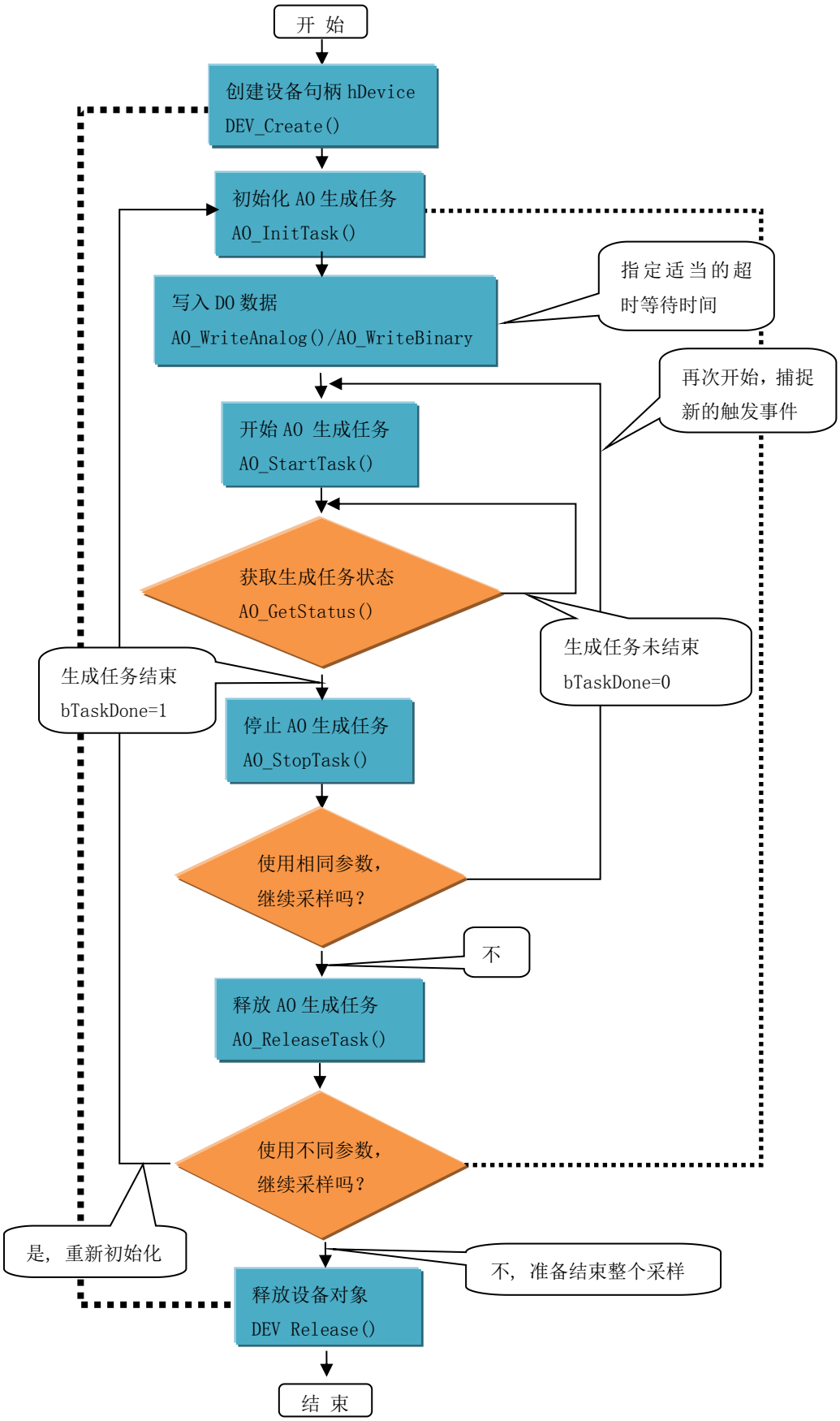


图 2-8-1 AO 有限点采样流程图例（AO_GetStatus()同步）

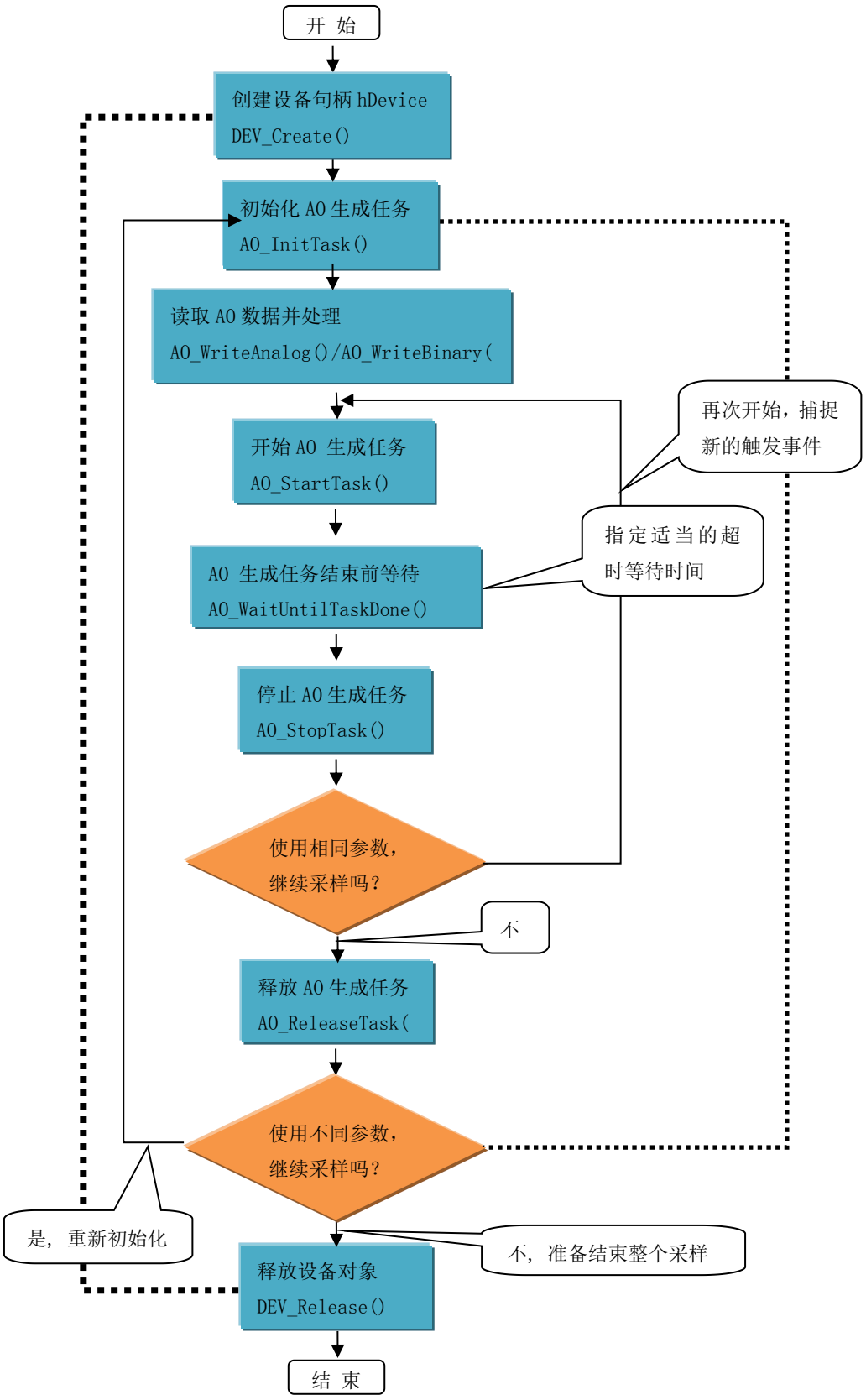


图 2-8-2　AO 有限点采样流程图例（AO_WaitUntilTaskDone ()同步）

2.10 AO 连续采样模式

连续采样模式，就是在一次开始后，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 生成任务，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() 开始 AO 生成任务；

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

(6) AO_StopTask() 停止 AO 生成任务；

(7) AO_ReleaseTask() 释放 AO 生成任务；

(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 步跟踪生成任务状态或处理相关事务，生成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。

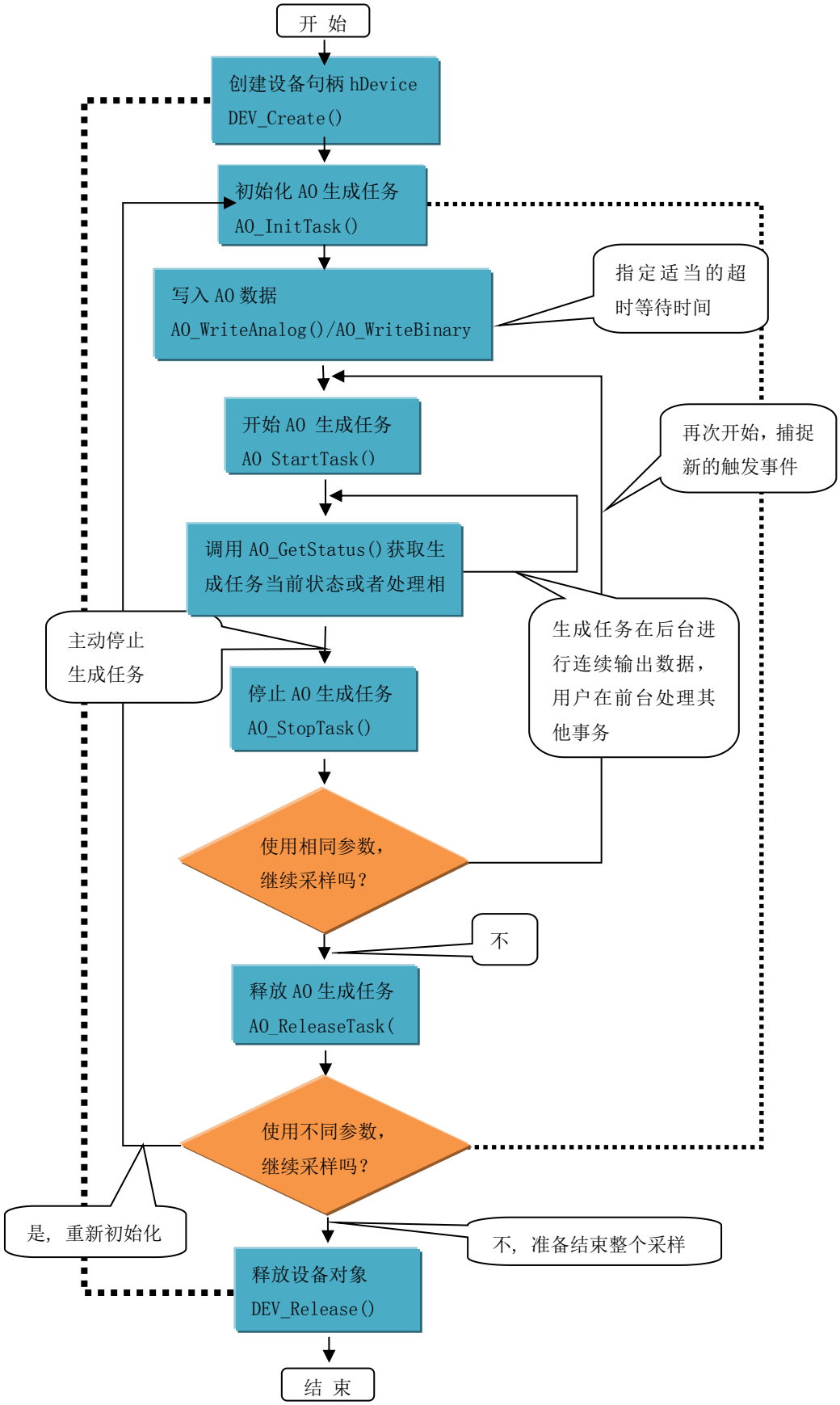


图 2-9-1　　AO 连续采样流程图例（重生成模式）

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 生成任务，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() 开始 AO 生成任务；

(5) AO_WriteAnalog()或者 AO_WriteBinary() 及时连续写入后续波形数据到生成任务中

(6) AO_StopTask() 停止 AO 生成任务；

(7) AO_ReleaseTask() 释放 AO 生成任务；

(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生成任务中（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。

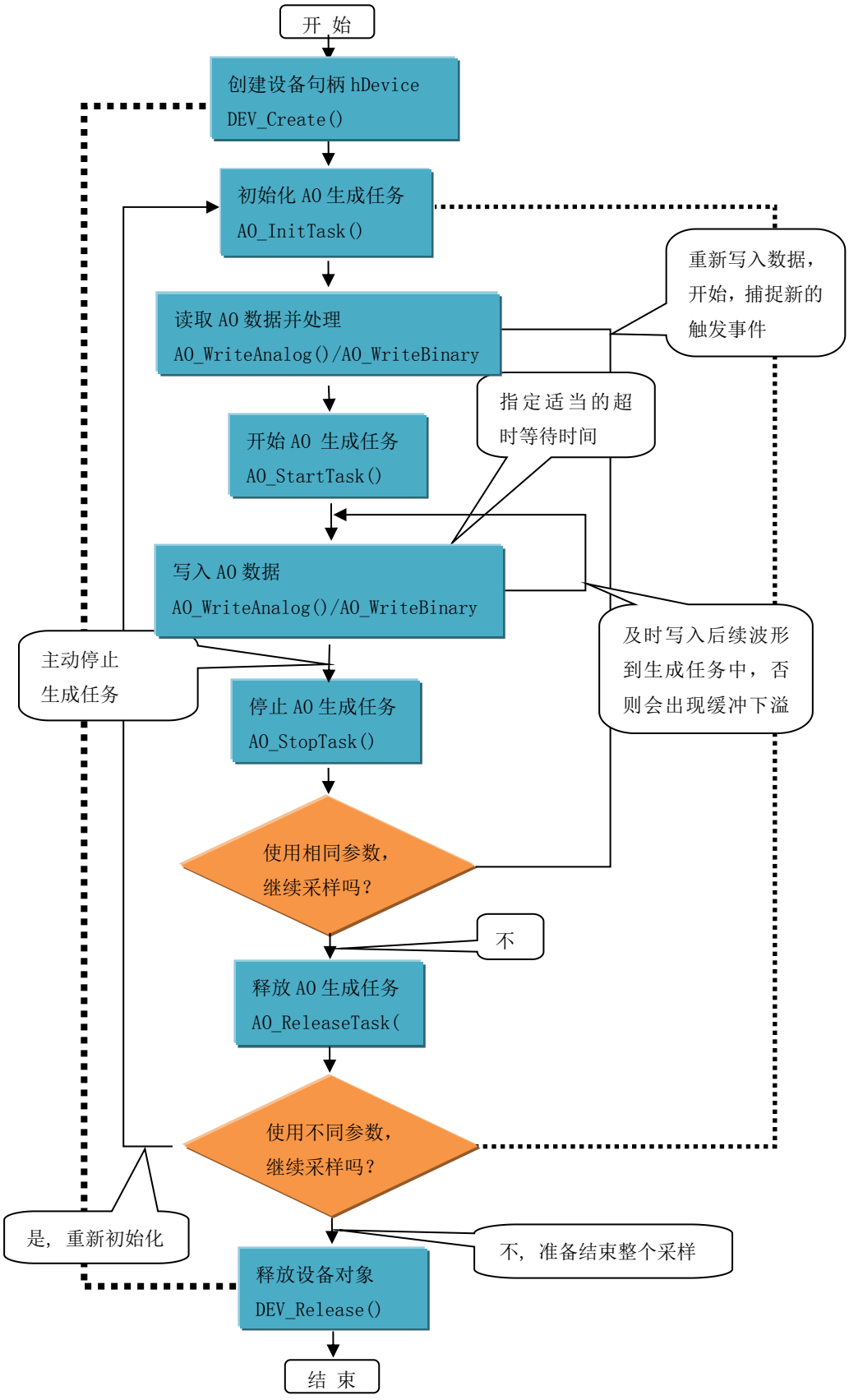


图 2-9-2　　AO 连续采样流程图例（非重生成模式）

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()）。

2.11 DIO 数字量读取操作

具体流程如下：

（1） DEV_Create() 创建设备句柄；

（2） Port_GetFuns () 获取各线功能；

（3） Port_Config () 配置 DIO 方向；

（4） DIO_ReadLines ()或者 DIO_ReadLine ()或者 DIO_ReadPort ()读取 DIO 数据；

（5） 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 DIO 实时读取 DI、DO 值；

2.12 DO 写数字量操作

具体步骤如下：

(1) DEV_Create() 创建设备句柄；

(2) Port_GetFuns () 获取各线功能；

(3) Port_Config () 配置 DIO 方向；

(4) DIO_WriteLines ()或者 DIO_WriteLine ()或者 DIO_WritePort ()写 DO 数据；

(5) 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 DO 循环输出；

2.13 CTR 介绍

表：CTR 下各简易程序对应功能

功能	简易程序名
边沿计数	CNT
半周期	HalfCycle
脉宽测量	PulseWidth
编码器	CodeCnt
脉冲输出	PulseOutput
两边沿间隔	TwoDdgeWidth
频率/周期	MeasureCycle

2.14 边沿、编码器、半周期、脉宽、两边沿间隔、频率/周期操作流程

具体步骤如下：

(1) DEV_Create() 创建设备句柄；

(2) CTR_InitTast() 初始化 CTR 采样任务；

(3) CTR_StartTask 启动 CTR 采样任务；

(4) CTR_ReadCounter 读取 CTR 数据；

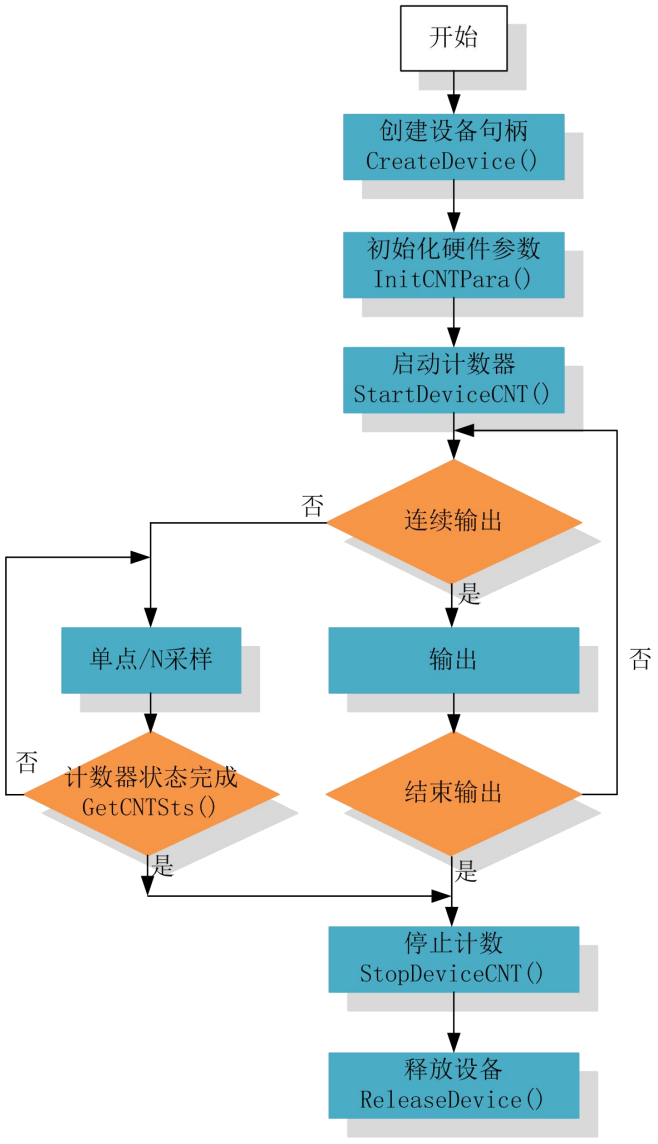
(5) CTR_StopTask 停止 CTR 采样任务；

(6) CTR_ReleaseTask 释放 CTR 采样任务；

(7) DEV_Release 释放 CTR 采样任务；

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 CTR 循环采样。

2.15 脉冲输出操作流程



2.16 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。

2.17 Port 及 PFI 使用介绍

此程序最多支持 4 个 Port，Port0 只能为开关量不可做为 PFI 使用。

Port1 8 线对应 PFI0~7

Port2 8 线对应 PFI8~15

Port3 8 线对应 PFI16~23(PXI、PXIE 板卡对应 PXI_Trig~PXI_Trig7)

线功能有两种，0：DIO， 1：PFI。可通过 Port_GetFuns 获取各线功能。线不提供功能选择函数，而是通过 AI、AO 中的一些参数设置自动设置功能。例：AI 的转换时钟选择 PFI0，自动将 Port1 的 0 线功能设置为 PFI 输入；多个参数可使用同一 PFI 输入，但输出参数独占 PFI，例 AI 的时钟输出。当 AI 释放时会将 AI 占用的 PFI 对应的线设置为 DI。

创建设备时，Port0 各线的方向和值不做修改，如果当前设备只打开一次(第一次创建设备或所有之前创建的设备已经释放)Port1~3 会回读端口下各线功能，如果是 PFI 会强制置为开关量输入(DI)，如果为 DIO，方向和值不做修改。

Port_Config 可配置功能为 DIO 的线，对已做为 PFI 使的线操作不起作用。

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++ / C++Builder / LabWindows/CVI:

HANDLE DEV_Create(LONG DeviceLgcID)

功能：使用逻辑 ID 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：

DeviceLgcID 入口参数，逻辑 ID 序号(DeviceLgcID)。逻辑序号的定义是：当向同一台计算机系统中加入若干张 ACTS2100 系列的卡时，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。

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)。

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

DEV_CreateEx ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
HANDLE    DEV_CreateEx (LONG DeviceID)
```

功能：使用物理 ID 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：

DeviceID 入口参数，设备物理 ID。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。当使用物理序号时，其取值范围为[0, 255]。

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)。

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

DEV_GetCount()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
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：如果有设备存在，则返回实际的设备数量，若该设备不存在，则返回 0 值，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。

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

DEV_GetCurrentIdx()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL DEV_GetCurrentIdx (HANDLE hDevice,
                        PLONG  pLgcIdx,
                        PLONG  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。

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

GetMainInfo

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL DEV (HANDLE hDevice，
PACTS2100_MAIN_INFO pMainInfo)

功能：获得板卡主要信息。

参数：

hDevice 入口参数，设备对象句柄，它应由 DEV_Create 创建。

pMainInfo 出口参数，设备信息结构体。

返回值：若成功，则弹出对话框控件列表设备的 BAR 情况。

DEV_SetReferenceClock

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL DEV_ListDlg (HANDLE hDevice, LONG nReferenceClock)

功能：设置板卡参考时钟(仅支持 PXIE 板卡)。

参数：

hDevice 入口参数，设备对象句柄，它应由 DEV_Create 创建。

nReferenceClock 入口参数，参考时钟,0:板载 10M 1:PXI_10M 2:PXIe_100M。

返回值：若成功,，则返回 TRUE， 否则返回 FALSE。

DEV_ListDlg

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL DEV_ListDlg (HANDLE hDevice)

功能：显示系统中设备数,并列表该设备 BAR 地址。

参数：

hDevice 入口参数，设备对象句柄，它应由 DEV_Create 创建。

返回值：若成功，则弹出对话框控件列表设备的 BAR 信息

DEV_Release()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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。

3.2 AI 模拟量输入函数原型说明

AI_InitTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AI_InitTask (HANDLE hDevice, PAI_PARAM pAIParam)

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。但它并不开始 AI 设备，若要开始 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《4 各种结构体描述》\《4.1 AI_PARAM（AI 工作参数结构）》。

返回值：如果初始化 AI 工作参数成功，则返回 TRUE， 否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AI_StartTask (HANDLE hDevice)

功能：开始 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 实际是否进入采集记录过程，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 立刻被开始， 否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AI_SendSoftTrig (HANDLE hDevice)

功能：发送软件触发事件(Send software trigger event for analog input)。软件触发时调用此函数无效,软件触发下 AI 启动后设备自动触发工作。硬件触发下设备进入等待触发状态，若用户需强制触发或需要随时手动给触发事件时，可调用此函数。

开始触发和暂停触发都选择无触发时即为软件触发。

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。

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++ / C++Builder / LabWindows/CVI:

BOOL AI_GetStatus (HANDLE hDevice, PAI_STATUS pStatus)

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。如果在开始采集任务后，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《4 各种结构体描述》\《4.2 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。

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++ / C++Builder / LabWindows/CVI:

BOOL AI_WaitUntilTaskDone (HANDLE hDevice, F64 fTimeout)

功能：在 AI 的采集任务结束前等待(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集任务结束再读数；也可以不调用此函数直接读（边采边读）。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 秒内采集任务结束，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（即总是等到采集任务结束才返回）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集任务是否结束，如果采集任务结束了，则返回 TRUE，否则返回 FALSE。

返回值：如果采集任务结束，则返回 TRUE，否则返回 FALSE。

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_ReadAnalog()

函数原型:

Visual C++ / C++Builder / LabWindows/CVI:

```
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。

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。

pSampsPerChanRead 出口参数，返回每通道实际读取的点数, 如果此值小于请求的点数小于读的点数即为超时。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，可相应日志(log)文件。

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++ / C++Builder / LabWindows/CVI:

```
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 AI 码值范围。各个采样通道的数据点是依次交替排列的。

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()

AI_OneReadBinary ()

函数原型:

Visual C++ / C++Builder / LabWindows/CVI:

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

功能：从采集任务中读取采样数据(原码数据序列)，此函数无需启动 触发 停止采样,只有单点方式下可执行此函数读数,使用流程请参考安装目录下的 OneSampleDemand_Ex 程序

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 AI 码值范围。各个采样通道的数据点是依次交替排列的。

nReadSampsPerChan 同 AI_ReadAnalog()。

pSampsPerChanRead 同 AI_ReadAnalog()。

pAvailSampsPerChan 同 AI_ReadAnalog()。

fTimeout 同 AI_ReadAnalog()

返回值：同 AI_ReadAnalog()

AI_StopTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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 采集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。

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++ / C++Builder / LabWindows/CVI:

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。

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_ScaleBinToVolt()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AI_ScaleBinToVolt(HANDLE hDevice,
PAI_VOLT_RANGE_INFO pRangeInfo,
PVOID pGainInfo,
F64 fVoltArray[],
U
16 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 入口参数，范围信息。它由 AI_GetVoltRangeInfo()函数获取。

pGainInfo 入口参数，增益信息。因本设备 AI 无增益功能，该参数无效，恒等于 NULL。

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 pRangeInfo 参数指定。

nBinArray 入口参数，用于传入待量化的原码数据，取值范围由 AI 精度决定。

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。

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

AI_ScaleVoltToBin()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AI_ScaleVoltToBin(
PAI_VOLT_RANGE_INFO pRangeInfo,
PVOID 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 入口参数，范围信息。它由 AI_GetVoltRangeInfo()函数获取。

pGainInfo 入口参数，增益信息。因本设备 AI 无增益功能，该参数无效，恒等于 NULL。

nBinArray 出口参数，用于传入待量化的原码数据，取值范围由 AI 精度决定。

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。

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

AI_GetVoltRangeInfo()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AI_GetVoltRangeInfo(HANDLE hDevice,
                         int nChannel,
                         int 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 入口参数，AI 通道号[0, nAIChannelCount-1]。

nSampleRange 入口参数，AI 采样范围，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《 4.4 AI_VOLT_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。

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

AI_GetRateInfo()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AI_GetRateInfo(HANDLE hDevice, AI_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《 4.5 AI_SAMP_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。

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

AI_VerifyParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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《4 各种结构体描述》\《4.1 AI_PARAM（AI 工作参数结构体）》。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 ACTS2100.log 记录不合法的参数名和原因，然后返回 FALSE。

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

AI_LoadParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AI_LoadParam (HANDLE hDevice,
PAI_PARAM pAIParam)

功能：从 ACTS2100.ini 参数配置文件中读取设备的硬件参数(Load parameter from ACTS2100.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 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 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。

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

AI_SaveParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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 ACTS2100.ini 参数配置文件中(Save parameter to ACTS2100.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 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 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。

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

AI_SaveParam()　　AI_ResetParam()

AI_ResetParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AI_ResetParam (HANDLE hDevice,
PAI_PARAM pAIParam)

功能：将 ACTS2100.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 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 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。

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

3.3 AO 模拟量输出函数原型说明

AO_InitTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AO_InitTask (HANDLE hDevice, PAO_PARAM pAOParam)

功能：初始化 AO 任务参数(Initialize task parameter for analog output)，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。但它并不开始 AO 设备，如果要开始 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《4 各种结构体描述》\《4.6 AO_PARAM（AO 工作参数结构体）》。

返回值：如果初始化 AO 工作参数成功，则返回 TRUE， 否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_StartTask (HANDLE hDevice)

功能：开始 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 立刻被开始， 否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_SendSoftTrig (HANDLE hDevice)

功能：产生强制触发事件(Send software trigger event for analog output)。在开始 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。

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++ / C++Builder / LabWindows/CVI:

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，表示至少可以往生成任务中写入 1 个点的数据。如果在开始生成任务后，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《4 各种结构体描述》\《4.6 AO_PARAM（AO 工作参数结构体）》。

返回值：如果成功获取 AO 状态，则返回 TRUE，否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_WaitUntilTaskDone (HANDLE hDevice, F64 fTimeout)

功能：在 AO 的生成任务结束前等待(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生成任务结束。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 秒内生成任务结束，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（即总是等到生成任务结束才返回）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生成任务是否结束，如果生成任务结束了，则返回 TRUE，否则返回 FALSE。

返回值：如果生成任务结束，则返回 TRUE，否则返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

```
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 AOStatus.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。

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++ / C++Builder / LabWindows/CVI:

```
LONG AO_WriteBinary(HANDLE hDevice,
                    I16 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生成任务中。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[-32768，32767]。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++ / C++Builder / LabWindows/CVI:

```
LONG AO_ReadbackAnalog(HANDLE hDevice,
                       F64 fAnlgArray[2]);
```

功能：回读 AO 的电压数据（Readback voltage data from the task）。

参数：

hDevice **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。

nAnlgArray 出口参数，用户缓冲区，用于接收回读的电压数据。nAnlgArray[0]为 AO0 的数据，nAnlgArray[1]为 AO1 的数据。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

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

AO_ReadbackBinary()

函数原型:

Visual C++ / C++Builder / LabWindows/CVI:

```
LONG AO_ReadbackBinary(HANDLE hDevice,
                       I16 nBinArray[2]);
```

功能：回读 AO 的电压数据（Readback binary data from the task）。

参数：

hDevice **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 出口参数，用户缓冲区，用于接收回读的二进制原码数据。nBinArray[0]为 AO0 的数据，nBinArray [1]为 AO1 的数据。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

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

DEV_Release()

AO_StopTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL AO_StopTask (HANDLE hDevice)

功能：停止 AO 生成任务(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 生成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。

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++ / C++Builder / LabWindows/CVI:

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。

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_ScaleBinToVolt()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AO_ScaleBinToVolt(
                PAO_VOLT_RANGE_INFO pRangeInfo
                F64 fVoltArray[],
                I16 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 入口参数，范围信息，它由 AO_GetVoltRangeInfo()函数获取。

fVoltArray 出口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由参数 pRangeInfo 指定。

nBinArray 入口参数，用于传入待量化的原码数据，取值范围[-32768, 32767]。

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。

相关函数： DEV_Create() AO_ScaleBinToVolt() AO_ScaleVoltToBin() AO() AO_GetVoltRangeInfo() AO_GetRateInfo() DEV_Release()

AO_ScaleVoltToBin()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AO_ScaleVoltToBin(
                ByRef pRangeInfo As AO_VOLT_RANGE_INFO, _
                I16 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 入口参数，范围信息，它由 AO_GetVoltRangeInfo()函数获取。

nBinArray 出口参数，用于传入待量化的原码数据，取值范围[-32768, 32767]。

fVoltArray 入口参数，用于返回量化后的电压数据，单位：伏（V），取值范围由参数 pRangeInfo 指定。

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。

相关函数： DEV_Create() AO_ScaleBinToVolt() AO_ScaleVoltToBin() AO() AO_GetVoltRangeInfo() AO_GetRateInfo() DEV_Release()

AO_GetVoltRangeInfo()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AO_GetVoltRangeInfo(HANDLE hDevice,
                int nChannel,
                int 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 入口参数，物理通道号，取值范围[0，3]。

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《4.10　AO_VOLT_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。

相关函数： DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO()　AO_GetVoltRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_GetRateInfo()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL AO_GetRateInfo(HANDLE hDevice,
                    AO_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《4.10　AO_SAMP_RATE_INFO (AO 采样速率信息结构体)》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

相关函数： DEV_Create()　AO_ScaleBinToVolt()　AO_ScaleVoltToBin()　AO()　AO_GetVoltRangeInfo()　AO_GetRateInfo()　DEV_Release()

AO_VerifyParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
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《4　各种结构体描述》\《4.6　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 ACTS2100.log 记录不合法的参数名和原因，然后返回 FALSE。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_LoadParam (HANDLE hDevice,
PAO_PARAM pAOParam);

功能：从 ACTS2100.ini 参数配置文件中读取设备的硬件参数(Load parameter from ACTS2100.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 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.6 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。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_SaveParam (HANDLE hDevice,
PAO_PARAM pAOParam)

功能：将配置好的 AO 工作参数保存在 ACTS2100.ini 参数配置文件中(Save parameter to ACTS2100.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 ACTS2100.h 等函数原型定义的头文件，也可参考本文《4.6 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。

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++ / C++Builder / LabWindows/CVI:

BOOL AO_ResetParam (HANDLE hDevice,
PAO_PARAM pAOParam)

功能：将 ACTS2100.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 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.6 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。

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

3.4 CTR 计数器函数原型说明

CTR_InitTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL CTR_InitTask (
                    HANDLE hDevice,
                    U32 nChannel,
                    PACTS2100_CTR_PARAM pCTRParam);
```

功能：初始 CTR 任务。

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，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

pCTRParam 入口参数，工作参数。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_StartTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL  CTR_StartTask (
                    HANDLE hDevice,
                    U32 nChannel);
```

功能：开珀 CTR 任务。

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，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_GetSts()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL  CTR_GetSts (
                    HANDLE hDevice,
                    PLONG lCNTSts,
```

U32 nChannel);

功能：计数器工作状态,判断脉冲输出是否完成。

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。

lCNTSts 出口参数，1:忙 0:空闲(0:输出完成)。

nChannel 入口参数，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_ReadCounter()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL  CTR_GetSts (
                    HANDLE hDevice,
                    U32 nChannel,
                    U32 nCountValArray[],
                    U32 nSizePoints,
                    U32* pSampsRead,
                    U32* pAvailSamps,
                    F64 fTimeout);
```

功能：从任务中读取数据。

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。

nChannel 入口参数，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

nCountValArray 出口参数，数据数组,用于返回计数数据。

nSizePoints 入口参数，请求读取的点数(单位:点)。

pSampsRead 出口参数，返回实际读取的点数(单位:点) 。

pAvailSamps 出口参数，任务中还存在的可读点数, =NULL,表示无须返回。

fTimeout 入口参数，超时时间,单位:秒(S)。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_StopTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL  CTR_GetSts (
                    HANDLE hDevice,
                    U32 nChannel);
```

功能：停止(或暂停)采集任务。

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，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_ReleaseTask()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL CTR_GetSts (
HANDLE hDevice,
PLONG lCNTSts,
U32 nChannel);

功能：释放采集任务。

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，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_LoadParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL CTR_LoadParam (HANDLE hDevice,
U32 nChannel,
PCTR_PARAM pCTRParam)

功能：从 ACTS2100.ini 参数配置文件中读取设备的硬件参数。

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。

nChannel 入口参数，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

pCTRParam 出口参数，属于 PCTR_PARAM 的结构指针类型，负责返回 CTR 工作参数值，关于结构指针类型 PCTR_PARAM 请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 CTR_PARAM（CTR 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_SaveParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL CTR_SaveParam (HANDLE hDevice,
U32 nChannel,
PCTR_PARAM pCTRParam)

功能：将用户配置好的 CTR 工作参数保存在 ACTS2100.ini 参数配置文件中(Save parameter to ACTS2100.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。

nChannel 入口参数，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

pCTRParam 入口参数，CTR 工作参数结构体指针，关于 CTR_PARAM 的详细介绍请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 CTR_PARAM（CTR 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

CTR_ResetParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL CTR_ResetParam (HANDLE hDevice,
U32 nChannel,
PCTR_PARAM pCTRParam)

功能：将 ACTS2100.ini 参数配置文件中原来的 CTR 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。

nChannel 入口参数，通道号(本程序最多支持 2 个 CTR 通道,取值范围[0,1])。

pCTRParam 出口参数，CTR 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 CTR_PARAM 的详细介绍请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 CTR_PARAM（CTR 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

3.5 IO 端口控制函数

Port_Config()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL Port_Config (
HANDLE hDevice,
U32 nPort,
PACTS2100_PORT_PARAM pPortParam);

功能：IO 端口配置函数,只有在线功能为 DIO 修改才会起作用。

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，nPort[0~3]。

pPortParam 入口参数，端口参数结构体。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

Port_GetConfig()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

BOOL Port_GetConfig (
HANDLE hDevice,
U32 nPort,
PACTS2100_PORT_PARAM pPortParam);

功能：获取 IO 端口配置。

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，nPort[0~3]。

pPortParam 出口参数，端口参数结构体。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

Port_GetFuns()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL Port_GetFuns (
                        HANDLE hDevice,
                        U32 bFuncArray[32]);
```

功能：获取各线功能。

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。

bFuncArray 出口参数，功能缓冲区, 0 表示为 DIO,1 表示为 PFI 或 PXI,Port0 对应 0~7,Port1 对应 8~15,如果 Port1 线功能为 PFI，则相应线做为 DIO 使用不起作用。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

PFI 使用详情请参考 2.23 章节。

PORT_LoadParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL PORT_LoadParam (HANDLE hDevice,
                        U32 nPort,
                        PPORT_PARAM pPORTParam)
```

功能：从 ACTS2100.ini 参数配置文件中读取设备的硬件参数。

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。

nPort 入口参数，nPort[0~3]。

pPORTParam 出口参数，属于 PPORT_PARAM 的结构指针类型，负责返回 PORT 工作参数值，关于结构指针类型 PPORT_PARAM 请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 PORT_PARAM（PORT 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

PORT_SaveParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL PORT_SaveParam (HANDLE hDevice,
                        U32 nPort,
                        PPORT_PARAM pPORTParam)
```

功能：将用户配置好的 PORT 工作参数保存在 ACTS2100.ini 参数配置文件中(Save parameter to ACTS2100.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。

nPort 入口参数，nPort[0~3]。

pPORTParam 入口参数，PORT 工作参数结构体指针，关于 PORT_PARAM 的详细介绍请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 PORT_PARAM（PORT 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

PORT_ResetParam()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL PORT_ResetParam (HANDLE hDevice,
                      U32 nPort,
                      PPORT_PARAM pPORTParam)
```

功能：将 ACTS2100.ini 参数配置文件中原来的 PORT 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。

nPort 入口参数，nPort[0~3]。

pPORTParam 出口参数，PORT 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 PORT_PARAM 的详细介绍请参考 ACTS2100.h 函数原型定义的头文件，也可参考本文《4.1 PORT_PARAM（PORT 工作参数结构体）》。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

3.6 DIO 数字量输入输出函数原型说明

DIO_ReadPort()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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

功能：读数字量端口值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

pPortData 入口参数，返回的端口数据，有效位 Bit[7:0]。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

DIO_WritePort ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL DIO_WritePort (HANDLE hDevice,
                    U32 nPort，
                    U32* pPortData,
                    U32 nLineEn);
```

功能：写数字量端口值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

nPortData 入口参数，返回的端口数据，有效位 Bit[7:0]。

nLineEn 入口参数，线输出使能位，有效位 Bit[7:0]。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

DIO_ReadLines ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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

功能：读数字量端口多线值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

bLineDataArray 出口参数，线数据缓冲区，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。

nLineCount 入口参数，所操作的线总数。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

DIO_WriteLines ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL DIO_ReadLines (HANDLE hDevice,
                    U32 nPort，
                    U32 bLineDataArray[],
                    U32 bLineEnArray [],
                    U32 nLineCount);
```

功能：写数字量端口多线值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

bLineDataArray 入口参数，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。

bLineEnArray 入口参数，线输出使能缓冲，和 **bLineDataArray 一一对应表示**。

nLineCount 入口参数，所操作的线总数。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

DIO_ReadLine ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

```
BOOL DIO_ReadLine (HANDLE hDevice,
```

U32 nPort，
U32 nLine,
U32* pLineData);

功能：读单线值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

nLine 入口参数，线号[0, 7]。

pLineData 出口参数，线数据，取值 0 或 1。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

DIO_WriteLine ()

函数原型：

Visual C++ / C++Builder / LabWindows/CVI:

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

功能：写单线值。

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。

nPort 入口参数，端口号，取值范围为[0,3]。

nLine 入口参数，线号[0, 7]。

bLineData 入口参数，线数据，取值 0 或 1。

返回值：如果成功，返回 TRUE，否则返回 FALSE。

3.7 内存映射寄存器操作函数原型说明

GetDeviceBar ()

函数原型：

Visual C++:

BOOL GetDeviceBar (HANDLE hDevice,
__int64 pbPCIBar[6])

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbPCIBar[6] 返回 PCI BAR 所有地址，具体 PCI BAR 中有多少可用地址请看硬件说明书。

返回值：若成功，返回 TRUE，否则返回 FALSE。

WriteRegisterByte ()

函数原型：

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        __int64 pbLinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

功能：以单字节（即 8 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterByte 函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

WriteRegisterWord ()

函数原型：

```
BOOL WriteRegisterWord (HANDLE hDevice,
                        __int64 pbLinearAddr,
                        ULONG OffsetBytes,
                        WORD Value)
```

功能：以双字节（即 16 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterWord 函数所访问的映射寄存器的内存单元。

Value 输出 16 位整型值。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE。

WriteRegisterULong ()

函数原型：

```
BOOL WriteRegisterULong (HANDLE hDevice,
                         __int64 pbLinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

功能：以四字节（即 32 位）方式写 PCIe 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确

定 WriteRegisterULong 函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

ReadRegisterByte ()

函数原型：

```
BYTE ReadRegisterByte (HANDLE hDevice,
                       __int64 pbLinearAddr,
                       ULONG OffsetBytes)
```

功能：以单字节（即 8 位）方式读 PCIe 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 ReadRegisterByte 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

ReadRegisterWord ()

函数原型：

```
WORD ReadRegisterWord (HANDLE hDevice,
                       __int64 pbLinearAddr,
                       ULONG OffsetBytes)
```

功能：以双字节（即 16 位）方式读 PCIe 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 ReadRegisterWord 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

ReadRegisterULong ()

函数原型：

```
ULONG ReadRegisterULong (HANDLE hDevice,
                         __int64 pbLinearAddr,
                         ULONG OffsetBytes)
```

功能：以四字节（即 32 位）方式读 PCIe 内存映射寄存器的指定单元。

参数：

hDevice 设备对象句柄，它应由 DEV_Create 创建。

pbLinearAddr PCIe 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对与 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterULong 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

WritePortByte ()

Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    __int64 pPort,
                    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 DEV_Create 创建。

pPort 指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。

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： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

WritePortWord ()

Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    __int64 pPort,
                    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 DEV_Create 创建。

pPort 指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。

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： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

WritePortULong ()

Visual C++:

```
BOOL WritePortULong (HANDLE hDevice,
                     __int64 pPort,
                     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 DEV_Create 创建。

pPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
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： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

ReadPortByte ()

Visual C++:

```
BYTE ReadPortByte( HANDLE hDevice,
                   __int64 pPort)
```

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 DEV_Create 创建。
pPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

ReadPortWord ()

Visual C++:

```
WORD ReadPortWord (HANDLE hDevice,
                   __int64 pPort)
```

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 DEV_Create 创建。
pPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

ReadPortULong ()

Visual C++:

```
ULONG ReadPortULong (HANDLE hDevice,
                     __int64 pPort)
```

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 DEV_Create 创建。
pPort 指定寄存器的物理基地址。
OffsetBytes 相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定端口的值。
相关函数： DEV_Create WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

CreateSystemEvent ()

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)。

ReleaseSystemEvent ()
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。

4 各种结构体描述

4.1 MAIN_INFO（主要信息结构体）

Visual C++ / C++Builder / LabWindows/CVI:
```
typedef struct _ACTS2100_MAIN_INFO
{
    U32 nDeviceType;            // 总线类型\设备类型 0x20125620
20115620...2012(PXIE)2011(PCIE)
    U32 nAIChannelCount;        // AI 通道数量

    U32 nAISampRangeCount;      // AI 采样范围挡位数量
    U32 nAISampRangeVal[9];     // AI 各采样范围对应档位值

    U32 nAISampSignalCount;     // AI 采样信号数量
    U32 nAISampSignalVal[17]; // AI 各采样信号对应值
```

```
    U32 nAIOConvertClockCount;      // AIO 转换时钟信号数量
    U32 nAIOConvertClockVal[33];// AIO 转换时钟信号对应值

    U32 nAIOClockOutputCount;// AIO 时钟输出输入数量
    U32 nAIOClockOutputVal[33];     // AIO 时钟输出输入对应值

    U32 nAIODTrigChannelCount;      // AIO DTR 触发输入数量
    U32 nAIODTrigChannelVal[33];    // AIO DTR 触发输入对应值

    U32 nAIOSyncTSOutCount;         // AIO 同步触发信号输出数量
    U32 nAIOSyncTSOutVal[33];       // AIO 同步触发信号输出对应值

    U32 nCTRClkSourceCount;         // CTR 触发源输入数量
    U32 nCTRClkSourceVal[33]; // CTR 触发源输入对应值

    U32 nAISampGainCount;           // AI 采样增益挡位数量
    U32 nAICouplingCount;       // AI 耦合挡位数量
    U32 nAIImpedanceCount;          // AI 阻抗的挡位数量
    U32 nAIDepthOfMemory;           // AI 板载存储器深度(点数)
    U32 nAICodeType;            // AI 码值类型(如=0 表示原码; =1 表示补码)
    U32 nAISampResolution;          // AI 采样分辨率(如=8 表示 8Bit; =12 表示 12Bit; =14 表示
14Bit; =16 表示 16Bit)
    U32 nAISampCodeCount;           // AI 采样编码数量(如 256, 4096, 16384, 65536)
    U32 nAISupportAPFI;             // AI 是否支持 APFI(ATR 触发下的一种方式)
    U32 nAITrigLvlResolution;   // AI 触发电平分辨率(如=8 表示 8Bit; =12 表示 12Bit; =16 表示
16Bit)
    U32 nAITrigLvlCodeCount;    // AI 触发电平编码数量(如 256, 4096)
    U32 nAIBaseRate;            // AI 基准频率 Hz
    U32 nAIMaxRate;                 // AI 最大频率 Hz

    U32 nReserved0;                 //  保留字段(暂未定义)
    U32 nReserved1;                 //  保留字段(暂未定义)
    U32 nReserved2;                 //  保留字段(暂未定义)
    U32 nReserved3;                 //  保留字段(暂未定义)

    U32 nAOChannelCount;        // AO 通道数量
    U32 nAOSampRangeCount;          // AO 采样范围挡位数量
    U32 nAOSampRangeVal[8];         // AO 各采样范围对应档位值
    U32 nAOSampGainCount;           // AO 增益挡位数量
    U32 nAOCouplingCount;           // AO 耦合挡位数量
    U32 nAOImpedanceCount;          // AO 阻抗的挡位数量
    U32 nAODepthOfMemory;           // AO 板载存储器深度(点数)
    U32 nAOSampResolution;          // AO 采样分辨率(如=8 表示 8Bit; =12 表示 12Bit; =14 表示
14Bit; =16 表示 16Bit)
```

```
    U32 nAOSampCodeCount;          // AO 采样编码数量(如 256, 4096, 16384, 65536)
    U32 nAOSupportAPFI;            // AO 是否支持 APFI(ATR 触发下的方式,AO 只支持 APFI
触发,如果 AO 不支持 APFI 即不支持 ATR 触发)
    U32 nAOTrigLvlResolution;  // AO 触发电平分辨率(如=8 表示 8Bit; =12 表示 12Bit; =16 表示
16Bit)
    U32 nAOTrigLvlCodeCount; // AO 触发电平编码数量(如 256, 4096)
    U32 nAOBaseRate;           // AO 基准频率 Hz
    U32 nAOGSingleMaxRate;         // AO 组内单通道最大频率 Hz(一般板卡如果有 4 通道,0 2
为一组,1 3 为一组)
    U32 nAOGMultiMaxRate;          // AO 组内多通道最大频率 Hz(一般板卡如果有 2 通道,0 1
为一组)
                                   // 5731,4 个通道为一组，单通道 1M,2 通道每通道 700K,3
通道每通道 500k,4 通道每通道 350K

    U32 nReserved4;                // 保留字段(暂未定义)
    U32 nReserved5;                // 保留字段(暂未定义)
    U32 nReserved6;                // 保留字段(暂未定义)
    U32 nReserved7;                // 保留字段(暂未定义)

    U32 nCTRChannelCount;          // CTR 通道数量
    U32 nDIOPortCfg[ACTS2100_DIO_MAX_PORTS];          // DIO Port 配置, 0:不支持 1:
支持
    U32 nCalibrationType;      // 校准类型(如=0 数字校准; =1 模拟校准)
    U32 nCalibrationResolution;  // 校准分辨率(如=8 表示 8Bit; =12 表示 12Bit; =16 表示 16Bit)

    U32 nReserved8;                // 保留字段(暂未定义)
    U32 nReserved9;                // 保留字段(暂未定义)
    U32 nReserved10;           // 保留字段(暂未定义)
    U32 nReserved11;           // 保留字段(已使用)
} ACTS2100_MAIN_INFO, *PACTS2100_MAIN_INFO;
```

nAISampRangeCount

AI 采样范围挡位数量。以 PCIE5630 卡为例，此值返的是 5，即此卡有 5 个档位，AI_PARAM 中的 nSampleRange 只能是 nAISampRangeVal 前 4 个元素中的一个，,而不是下表中的前 4 个。

nAISampRangeVal

AI 各采样范围对应档位值。

常量名	常量值	采样范围
AI_SAMPRANGE_N10_P10V	0	±10V
AI_SAMPRANGE_N5_P5V	1	±5V
AI_SAMPRANGE_N2D5_P2D5V	2	±2.5V
AI_SAMPRANGE_N2_P2V	3	±2V
AI_SAMPRANGE_N1_P1V	4	±1V

AI_SAMPRANGE_0_P10V	5	0~10V
AI_SAMPRANGE_0_P5V	6	0~5V

nAISampSignalCount

AI 采样信号数量。PCIE5630 此值返的是 8，即此卡有 8 个信号源，AI_PARAM 中的 nSampleSignal 只能是 nAISampSignalVal 前 8 个元素中的一个,而不是下表中的前 8 个。

nAISampSignalVal

AI 各采样信号对应值。

常量名	常量值	采样范围
AI_SAMPSIGNAL_AI	0	AI通道输入信号
AI_SAMPSIGNAL_0V	1	0V(AGND)
AI_SAMPSIGNAL_1V	2	1V(DC)
AI_SAMPSIGNAL_2D5V	3	2.5V(DC)
AI_SAMPSIGNAL_N2D5V	4	-2.5V(DC)
AI_SAMPSIGNAL_5V	5	5V(DC)
AI_SAMPSIGNAL_AO0	6	AO0(DC)
AI_SAMPSIGNAL_AO1	7	AO1(DC)
AI_SAMPSIGNAL_AO2	8	AO2(DC)
AI_SAMPSIGNAL_AO3	9	AO3(DC)

nAIOConvertClockCount

AI、AO 时钟源所数量。PCIE5630 此值返的是 25， AI_PARAM、AO_PARAM 中的 nConvertClock 只能是 nAIOConvertClockVal 前 25 个元素中的一个。

常量名	常量值	采样范围
AIO_CVCLK_LOCAL	0	本地时钟(通常为本地晶振时钟OSCCLK)，也叫内部时钟
AIO_CVCLK_PFI00	1	PFI00输入时钟
AIO_CVCLK_PFI01	2	PFI01输入时钟
AIO_CVCLK_PFI02	3	PFI02输入时钟
AIO_CVCLK_PFI03	4	PFI03输入时钟
AIO_CVCLK_PFI04	5	PFI04输入时钟
AIO_CVCLK_PFI05	6	PFI05输入时钟
AIO_CVCLK_PFI06	7	PFI06输入时钟
AIO_CVCLK_PFI07	8	PFI07输入时钟
AIO_CVCLK_PFI08	9	PFI08输入时钟

AIO_CVCLK_PFI09	10	PFI09输入时钟
AIO_CVCLK_PFI10	11	PFI10输入时钟
AIO_CVCLK_PFI11	12	PFI11输入时钟
AIO_CVCLK_PFI12	13	PFI12输入时钟
AIO_CVCLK_PFI13	14	PFI13输入时钟
AIO_CVCLK_PFI14	15	PFI14输入时钟
AIO_CVCLK_PFI15	16	PFI15输入时钟
AIO_CVCLK_PFI16	17	PFI16输入时钟(PXIE设备为PXI0)
AIO_CVCLK_PFI17	18	PFI17输入时钟(PXIE设备为PXI1)
AIO_CVCLK_PFI18	19	PFI18输入时钟(PXIE设备为PXI2)
AIO_CVCLK_PFI19	20	PFI19输入时钟(PXIE设备为PXI3)
AIO_CVCLK_PFI20	21	PFI20输入时钟(PXIE设备为PXI4)
AIO_CVCLK_PFI21	22	PFI21输入时钟(PXIE设备为PXI5)
AIO_CVCLK_PFI22	23	PFI22输入时钟(PXIE设备为PXI6)
AIO_CVCLK_PFI23	24	PFI23输入时钟(PXIE设备为PXI7)
AIO_CVCLK_PXISTAR	25	PXI_STAR输入时钟(仅PXIE设备支持)

nAIOClockOutputCount

AI、AO 时钟输出数量。

nAIOClockOutputVal

AI、AO 时钟输出输入对应值。AI_PARAM、AO_PARAM 中的 nClockOutput 使用。

常量名	常量值	采样范围
AIO_CLKOUT_DISABLE	0	禁止输出
AIO_CLKOUT_PFI00	1	PFI00输出时钟
AIO_CLKOUT_PFI01	2	PFI01输出时钟
AIO_CLKOUT_PFI02	3	PFI02输出时钟
AIO_CLKOUT_PFI03	4	PFI03输出时钟
AIO_CLKOUT_PFI04	5	PFI04输出时钟
AIO_CLKOUT_PFI05	6	PFI05输出时钟
AIO_CLKOUT_PFI06	7	PFI06输出时钟
AIO_CLKOUT_PFI07	8	PFI07输出时钟
AIO_CLKOUT_PFI08	9	PFI08输出时钟
AIO_CLKOUT_PFI09	10	PFI09输出时钟

AIO_CLKOUT_PFI10	11	PFI10输出时钟
AIO_CLKOUT_PFI11	12	PFI11输出时钟
AIO_CLKOUT_PFI12	13	PFI12输出时钟
AIO_CLKOUT_PFI13	14	PFI13输出时钟
AIO_CLKOUT_PFI14	15	PFI14输出时钟
AIO_CLKOUT_PFI15	16	PFI15输出时钟
AIO_CLKOUT_PFI16	17	PFI16输出时钟(PXIE设备为PXI0)
AIO_CLKOUT_PFI17	18	PFI17输出时钟(PXIE设备为PXI1)
AIO_CLKOUT_PFI18	19	PFI18输出时钟(PXIE设备为PXI2)
AIO_CLKOUT_PFI19	20	PFI19输出时钟(PXIE设备为PXI3)
AIO_CLKOUT_PFI20	21	PFI20输出时钟(PXIE设备为PXI4)
AIO_CLKOUT_PFI21	22	PFI21输出时钟(PXIE设备为PXI5)
AIO_CLKOUT_PFI22	23	PFI22输出时钟(PXIE设备为PXI6)
AIO_CLKOUT_PFI23	24	PFI23输出时钟(PXIE设备为PXI7)

nAIODTrigChannelVal

AIO DTR 触发输入对应值。AI_PARAM、AO_PARAM 中的 nDTrigChannel 使用。

常量名	常量值	采样范围
AIO_DTRIG_PFI00	0	PFI00触发
AIO_DTRIG_PFI01	1	PFI01触发
AIO_DTRIG_PFI02	2	PFI02触发
AIO_DTRIG_PFI03	3	PFI03触发
AIO_DTRIG_PFI04	4	PFI04触发
AIO_DTRIG_PFI05	5	PFI05触发
AIO_DTRIG_PFI06	6	PFI06触发
AIO_DTRIG_PFI07	7	PFI07触发
AIO_DTRIG_PFI08	8	PFI08触发
AIO_DTRIG_PFI09	9	PFI09触发
AIO_DTRIG_PFI10	10	PFI10触发
AIO_DTRIG_PFI11	11	PFI11触发
AIO_DTRIG_PFI12	12	PFI12触发
AIO_DTRIG_PFI13	13	PFI13触发
AIO_DTRIG_PFI14	14	PFI14触发
AIO_DTRIG_PFI15	15	PFI15触发
AIO_DTRIG_PFI16	16	PFI16触发(PXIE设备为PXI0)

AIO_DTRIG_PFI17	17	PFI17触发(PXIE设备为PXI1)
AIO_DTRIG_PFI18	18	PFI18触发(PXIE设备为PXI2)
AIO_DTRIG_PFI19	19	PFI19触发(PXIE设备为PXI3)
AIO_DTRIG_PFI20	20	PFI20触发(PXIE设备为PXI4)
AIO_DTRIG_PFI21	21	PFI21触发(PXIE设备为PXI5)
AIO_DTRIG_PFI22	22	PFI22触发(PXIE设备为PXI6)
AIO_DTRIG_PFI23	23	PFI23触发(PXIE设备为PXI7)
AIO_DTRIG_PXISTAR	24	PXI_STAR触发(仅PXIE设备支持)

nAIOSyncTSOutVal

AI、AO 同步触发输出输入对应值。AI_PARAM、AO_PARAM 中的 nSyncTSOut 使用。

常量名	常量值	采样范围
AIO_STSO_DISABLE	0	禁止同步触发信号
AIO_STSO_PFI00	1	PFI00输出同步触发信号
AIO_STSO_PFI01	2	PFI01输出同步触发信号
AIO_STSO_PFI02	3	PFI02输出同步触发信号
AIO_STSO_PFI03	4	PFI03输出同步触发信号
AIO_STSO_PFI04	5	PFI04输出同步触发信号
AIO_STSO_PFI05	6	PFI05输出同步触发信号
AIO_STSO_PFI06	7	PFI06输出同步触发信号
AIO_STSO_PFI07	8	PFI07输出同步触发信号
AIO_STSO_PFI08	9	PFI08输出同步触发信号
AIO_STSO_PFI09	10	PFI09输出同步触发信号
AIO_STSO_PFI10	11	PFI10输出同步触发信号
AIO_STSO_PFI11	12	PFI11输出同步触发信号
AIO_STSO_PFI12	13	PFI12输出同步触发信号
AIO_STSO_PFI13	14	PFI13输出同步触发信号
AIO_STSO_PFI14	15	PFI14输出同步触发信号
AIO_STSO_PFI15	16	PFI15输出同步触发信号
AIO_STSO_PFI16	17	PFI16输出同步触发信号(PXIE设备为PXI0)
AIO_STSO_PFI17	18	PFI17输出同步触发信号(PXIE设备为PXI1)
AIO_STSO_PFI18	19	PFI18输出同步触发信号(PXIE设备为PXI2)

AIO_STSO_PFI19	20	PFI19输出同步触发信号(PXIE设备为PXI3)
AIO_STSO_PFI20	21	PFI20输出同步触发信号(PXIE设备为PXI4)
AIO_STSO_PFI21	22	PFI21输出同步触发信号(PXIE设备为PXI5)
AIO_STSO_PFI22	23	PFI22输出同步触发信号(PXIE设备为PXI6)
AIO_STSO_PFI23	24	PFI23输出同步触发信号(PXIE设备为PXI7)

nCTRClkSourceVal

CTR 触发源输入对应值。CTR_PARAM 中的 lClkSource 使用。

常量名	常量值	采样范围
CTR_CLKSRC_NONE	0	无输入
CTR_CLKSRC_PFI00	1	PFI00输入时钟
CTR_CLKSRC_PFI01	2	PFI01输入时钟
CTR_CLKSRC_PFI02	3	PFI02输入时钟
CTR_CLKSRC_PFI03	4	PFI03输入时钟
CTR_CLKSRC_PFI04	5	PFI04输入时钟
CTR_CLKSRC_PFI05	6	PFI05输入时钟
CTR_CLKSRC_PFI06	7	PFI06输入时钟
CTR_CLKSRC_PFI07	8	PFI07输入时钟
CTR_CLKSRC_PFI08	9	PFI08输入时钟
CTR_CLKSRC_PFI09	10	PFI09输入时钟
CTR_CLKSRC_PFI10	11	PFI10输入时钟
CTR_CLKSRC_PFI11	12	PFI11输入时钟
CTR_CLKSRC_PFI12	13	PFI12输入时钟
CTR_CLKSRC_PFI13	14	PFI13输入时钟
CTR_CLKSRC_PFI14	15	PFI14输入时钟
CTR_CLKSRC_PFI15	16	PFI15输入时钟
CTR_CLKSRC_PFI16	17	PFI16输入(PXIE设备为PXI0)
CTR_CLKSRC_PFI17	18	PFI17输入(PXIE设备为PXI1)
CTR_CLKSRC_PFI18	19	PFI18输入(PXIE设备为PXI2)

CTR_CLKSRC_PFI19	20	PFI19输入(PXIE设备为PXI3)
CTR_CLKSRC_PFI20	21	PFI20输入(PXIE设备为PXI4)
CTR_CLKSRC_PFI21	22	PFI21输入(PXIE设备为PXI5)
CTR_CLKSRC_PFI22	23	PFI22输入(PXIE设备为PXI6)
CTR_CLKSRC_PFI23	24	PFI23输入(PXIE设备为PXI7)
CTR_CLKSRC_PXISTAR	25	PXI_STAR输入(仅PXIE设备支持)

nAOSampRangeVal

AO 采样范围对应值。AO_CH_PARAM 中的 nSampleRange 使用。

常量名	常量值	采样范围
AO_SAMPRANGE_N10_P10V	0	±10V
AO_SAMPRANGE_N5_P5V	1	±5V
AO_SAMPRANGE_N2D5_P2D5V	2	±2.5V
AO_SAMPRANGE_N2D5_P7D5V	3	-2.5~7.5V
AO_SAMPRANGE_0_P10V	4	0~10V
AO_SAMPRANGE_0_P5V	5	0~5V

4.2 AI_PARAM（AI 工作参数结构体）

4.1.1 AI_CH_PARAM(AI 通道参数结构体)

Visual C++ / C++Builder / LabWindows/CVI:

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

    U32 nReserved0;
    U32 nReserved1;
    U32 nReserved2;
}
```

nChannel

通道号,0~ nAIChannelCount-1。

nSampleGain

采样增益,具体放大倍数请参考下面常量定义。

常量名	常量值	功能定义
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)，请参考下面常量定义。

常量名	常量值	功能定义
AI_REFGND_RSE	0	接地参考单端(Referenced Single Endpoint)
AI_REFGND_NRSE	1	非参考单端(Non Referenced Single Endpoint)
AI_REFGND_DIFF	2	差分(Differential)

nReserved0-3

保留字段。

4.1.2 AI_PARAM (AI 工作参数结构体)

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _AI_CH_PARAM
{
    AI_CH_PARAM CHParam[AI_MAX_CHANNELS];
    U32 nSampChanCount;
     U32 nChanScanMode;
     U32 nGroupLoops;
    U32 nGroupInterval;
    U32 nSampleSignal;
    U32 nSampleRange;
    U32 nSampleMode;
    U32 nSampsPerChan;
    F64 fSampleRate;
    U32 nConvertClock;
    U32 nClockOutput;
    U32 nReserved0;
     U32 bDTriggerEn;
     U32 nDTriggerDir;
    U32 nDTriggerSens;
    U32 nDTrigChannel;
    U32 bATriggerEn;
    U32 nATriggerDir;
    F32 fTriggerLevel;
```

```
        U32 nATriggerSens;
        U32 nATrigSource;

        U32 nSyncTSOut;
        U32 nDelaySamps;
        U32 nReserved1;
        U32 nReserved2;
        U32 nReserved3;
        U32 nReserved4;
        U32 nReserved5;
    }
```

CHParam

通道参数配置结构体数组。

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]。

nChanScanMode

通道扫描模式,0=通道连续扫描, 1=通道分组扫描(此功能暂不支持)。

nGroupLoops

组内循环次数[1, 65535](此功能暂不支持)。

nGroupInterval

组与组之间的时间间隔，单位:微秒,取值范围[0, 107374182]，等于 0 时表示等间隔采样(不分组，此时 nGroupLoops 应等于 1)(此功能暂不支持)。

nSampleSignal

AI 采样信号(Sample Signal)，取值范围参考 MAIN_INFO。

nSampleRange

采样范围(Sample Range)档位选择，取值范围参考 MAIN_INFO。

nSampleMode

AI 采样模式(Sample Mode)，取值[0, 3]，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_SendSoftTrig 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。

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。在开始采集任务后，达到触发条件并采样完成指定点数的数据后，采集任务就会自动停止。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，在开始采集任务后，只要不软件停止采集任务，其采集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：该参数无意义；

有限点采样模式：该参数表示每通道采样点数。取值范围为[2，n]样点；

连续采样模式：该参数没有实际意义；

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，它与 nChannelCount 的乘积不能大于最大 AI 采样率。

nConvertClock

转换时钟信号选择，取值范围参考 MAIN_INFO。

nClockOutput

采样时钟输出选择，取值范围参考 MAIN_INFO。

bDTriggerEn

数字触发 DTR 允许(Digital Trigger Enable), =FALSE:表示禁止; =TRUE:表示允许。

nDTriggerDir

数字触发方向，取值[0, 3]，具体定义见下表：

常量名	常量值	功能定义	备注
AI_TRIGDIR_FALLING	0	下降沿/低电平	
AI_TRIGDIR_RISING	1	上升沿/高电平	
AI_TRIGDIR_CHANGE	2	变化(即上下边沿/高低电平均有效)	

nDTriggerSens

DTR 触发灵敏度(Trigger Sensitive for Digital and Analog trigger),单位:微秒(uS),取值范围[0, 1638]。

nDTrigChannel

DTR 触发通道选择，取值范围参考 MAIN_INFO。

bATriggerEn

模拟触发 ATR 允许(Analog Trigger Enable), =TRUE:表示允许, =FALSE:表示禁止。

nATriggerDir

模拟触发方向(Analog Trigger Direction)，取值[0, 3]，具体定义见下表：

常量名	常量值	功能定义	备注
AI_TRIGDIR_FALLING	0	下降沿/低电平	
AI_TRIGDIR_RISING	1	上升沿/高电平	

AI_TRIGDIR_CHANGE	2	变化(即上下边沿/高低电平均有效)	

fTriggerLevel

触发电平(Trigger Level), 单位 V; 选择 APFI 作为 ATR 触发源(触发范围为正负 10V), 选择 AI 采样首通道作为 ATR 触发源(触发范围为 AI 量程)。

nATriggerSens

ATR 触发灵敏度(Trigger Sensitive for Digital and Analog trigger),单位:微秒(uS),取值范围[0, 1638]。

nATrigSource

ATR 触发源选择，取值[0,1]，具体定义见下表：

常量名	常量值	功能定义	备注
AIO_ATRIG_APFI	0	选择APFI作为ATR触发源(触发范围为正负V)	
AI_ATRIG_FIRST	1	选择AI采样首通道作为ATR触发源(触发范围为AI量程)	

nSyncTSOut

同步触发信号输出通道选择，取值范围参考 MAIN_INFO。

nDelaySamps

触发延迟点数, 单位:采样点,取值范围 32 位有效[0, 4294967295]。

nReserved0-5

保留字段

4.3 AI_STATUS（AI 工作状态信息结构）

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _AI_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;
} 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 采集任务完成标志(Task Done)。=TRUE:表示采集任务已结束； =FALSE:表示采集任务正在进行中。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。在有限点采集任务中，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。在连续采集任务中，只有调用 AI_StopTask()函数手动停止采集任务，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

任务状态(Task State), 当等于 1 时表示正常, 其它值表示有异常情况。

nAvailSampsPerChan

有效点数，表示采集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

自开始采样后，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。该状态值在采集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，则意味着采集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

采集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，采集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

自开始采集任务后，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）。在开始采集任务后，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。如果用户重新开始采样，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）。在开始采样后，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。如果重新开始采样，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，用于检测初始化采集任务与释放采集任务是否前后匹配，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。用于检测开始采集任务与停止采集任务是否前后匹配，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，即每秒每通道传输了多少点的数据。比如设定的单个通道采样速率(fSampleRate)为 100000sps，则正常情况下，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)。

4.4 AI_VOLT_RANGE_INFO (AI 采样范围信息结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

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

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

    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
} AI_VOLT_RANGE_INFO, *P AI_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,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。

fNeadCode

电压换算所需码值，双极性为-2048(12 位精度),单级性为 0。

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	

nCodeCount

编码总数量，如比 12 位的 AI，其编码数量为 4096， 14 位的为 16384， 16 位的为 65536。

nMaxCode

采样二进制原码数据的最大值

nMinCode

采样二进制原码数据的最小值

nReserved1-4

保留字段。

4.5 AI_VOLT_GAIN_INFO (AI 增益信息结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _AI_VOLT_GAIN_INFO
{
    U32 nSampleGain;
    U32 nReserved0;
    F64 fAmpFactor;
    char strDesc[16];
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
}
```

nSampleGain

当前采样增益挡位号，[0~3]。

fAmpFactor

放大倍数。

strDesc

采样增益字符描述,如"10 倍", "100 倍"等。

4.6 AI_SAMP_RATE_INFO (AI 采样速率信息结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
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 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。

4.7 AO_PARAM（AO 工作参数结构体）

4.7.1 AO_CH_PARAM(AI 通道参数结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

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

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

bChannelEn

AO 物理通道使能(Channel Enable)，TRUE:使能，FALSE:禁止。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)，取值范围参考 MAIN_INFO。

nReserved0-3

保留字段

4.7.2 AO_PARAM(AI 工作参数结构体）

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _AO_CH_PARAM
{
    AO_CH_PARAM CHParam[4];
    U32 nSampleMode;
    U32 nSampsPerChan;
    F64 fSampleRate;
    U32 nConvertClock;
    U32 nClockOutput;
    U32 bRegenModeEn;
    U32 nReserved0;
    U32 bDTriggerEn;
    U32 nDTriggerDir;
    U32 nDTriggerSens;
    U32 nDTrigChannel;
    U32 bATriggerEn;
    U32 nATriggerDir;
    F32 fTriggerLevel;
    U32 nATriggerSens;
    U32 nATrigSource;

    U32 nSyncTSOut;
    U32 nDelaySamps;
    U32 nReserved1;
    U32 nReserved2;
    U32 nReserved3;
    U32 nReserved4;
    U32 nReserved5;
}
```

CHParam

通道参数配置结构体数组。

nSampleMode

AO 采样模式(Sample Mode)，取值[0, 3]，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	单点采样(硬件定时,Hardware Timed，本设备暂时不支持)	
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，但并不实际输出

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输出，且每个通道仅输出一个点的数据，并以最快的速度返回，此时该点数据立即输出到设备上。

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、等间隔的数据输出。在开始采集任务后，达到触发条件并输出完成指定点数的数据后，采集任务就会自动停止。

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、等间隔的数据输出，在开始采集任务后，只要不软件停止任务，其任务是永远不会终止的。

nSampsPerChan

AO 每通道待读取点数(Samples Per Channel)。

单点采样模式：该参数无意义；

有限点采样模式：该参数表示每通道采样点数。取值范围为[2，n]；

连续采样模式：该参数没有实际意义；

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的最高采样率决定。

nConvertClock

转换时钟信号选择，取值范围参考 MAIN_INFO。

nClockOutput

采样时钟输出选择，取值范围参考 MAIN_INFO。

bRegenModeEn

波形重生成模式允许(仅在连续采样模式中有效)。

bDTriggerEn

数字触发 DTR 允许(Digital Trigger Enable), =FALSE:表示禁止; =TRUE:表示允许。

nDTriggerDir

数字触发方向，取值[0, 3]，具体定义见下表：

常量名	常量值	功能定义	备注
AO_TRIGDIR_FALLING	0	下降沿/低电平	
AO_TRIGDIR_RISING	1	上升沿/高电平	
AO_TRIGDIR_CHANGE	2	变化(即上下边沿/高低电平均有效)	

nDTriggerSens

DTR 触发灵敏度(Trigger Sensitive for Digital and Analog trigger),单位:微秒(uS),取值范围[0, 1638]。

nDTrigChannel

DTR 触发通道选择，取值范围参考 MAIN_INFO。

bATriggerEn

模拟触发 ATR 允许(Analog Trigger Enable), =TRUE:表示允许, =FALSE:表示禁止。

nATriggerDir

模拟触发方向(Analog Trigger Direction), 取值[0, 3]，具体定义见下表：

常量名	常量值	功能定义	备注
AO_TRIGDIR_FALLING	0	下降沿/低电平	
AO_TRIGDIR_RISING	1	上升沿/高电平	
AO_TRIGDIR_CHANGE	2	变化(即上下边沿/高低电平均有效)	

fTriggerLevel

触发电平(Trigger Level), 单位 V; 选择 APFI 作为 ATR 触发源(触发范围为正负 10V)。

nATriggerSens

ATR 触发灵敏度(Trigger Sensitive for Digital and Analog trigger),单位:微秒(uS),取值范围[0, 1638]。

nATrigSource

ATR 触发源选择，取值[0,1]，具体定义见下表：

常量名	常量值	功能定义	备注
AIO_ATRIG_APFI	0	选择APFI作为ATR触发源(触发范围为正负V)	

nSyncTSOut

同步触发信号输出通道选择，取值范围参考 MAIN_INFO。

nDelaySamps

触发延迟点数, 单位:采样点,取值范围 32 位有效[0, 4294967295]。

nReserved0-5

保留字段

4.8 AO_STATUS（AO 工作状态信息结构）

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
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 生成任务完成标志(Task Done)。=TRUE:表示生成任务已结束, =FALSE:表示生成任务正在进行中。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。在有限点生成任务中，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。在连续生成任务中，只有调用 AO_StopTask()函数手动停止生成任务，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。

nTaskState

任务状态(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，表示生成任务缓冲中的可写入数据点数（Available Samples Per Channel）。在初始化生成任务后，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

自开始采样后，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。该状态值在生成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，则意味着生成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

生成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，生成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

自开始生成任务后，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）。在开始生成任务后，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。如果用户重新开始采样，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）。在开始生成任务后，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。如果重新开始采样，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，用于检测初始化生成任务与释放生成任务是否前后匹配，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。用于检测开始生成任务与停止生成任务是否前后匹配，如该计数值始终比 nStopTaskCnt 大 1，则表示每调用一次 AO_StartTask()后就会相应的调用 AO_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)。

4.9 AO_VOLT_RANGE_INFO (AO 采样范围信息结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
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

AO 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	

nCodeCount

编码总数量，如比 12 位的 AI，其编码数量为 4096， 14 位的为 16384， 16 位的为 65536。

nMaxCode

采样二进制原码数据的变化最大值

nMinCode

采样二进制原码数据的变化最值值

nReserved1-4

保留字段。

4.10 AO_SAMP_RATE_INFO (AO 采样速率信息结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _ AO_SAMP_RATE_INFO
{
    F64 fGSingleMaxRate;
    F64 fGMultiMaxRate;
    F64 fMinRate;
    F64 fTimerBase
    U32 nDivideMode;
    U32 nRateType;

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

fGSingleMaxRate

单通道最大采样速率(sps),多通道时各通道平分最大采样率。

fGMultiMaxRate

单通道最大采样速率(sps),多通道时各通道平分最大采样率(5731 除外)。

fMinRate

最小采样速率(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。

4.11 PORT_PARAM (IO 端口通道参数结构体)

结构体请参考 ACTS2100.h

Visual C++ / C++Builder / LabWindows/CVI:

```
typedef struct _AI_CH_PARAM
{
    U32 nDIODir[8];
    U32 nDIOVal[8];
}
```

nDIODir

DIO 方向,0:输入 1:输出。

nDIOVal

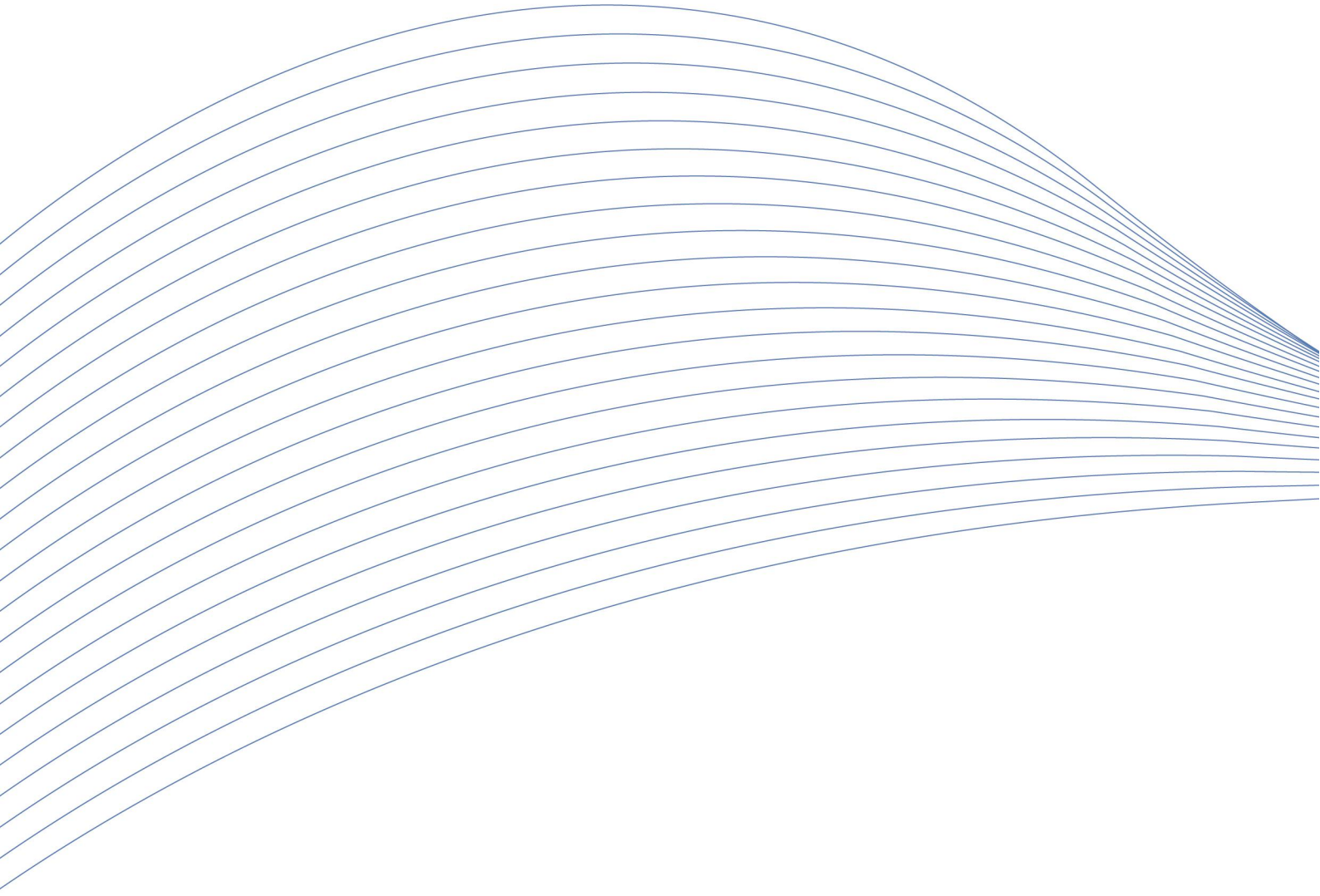
DIO 方向为输出时输出值,0:低高平 1:高电平。

4.12 CTR_PARAM (计数器参数结构体)

结构体请参考 ACTS2100.h，CTR 下各功能可使用参数请参考 CTR 下各功能简易程序。

5 修改历史

修改时间	版本号	修改内容
2020.07.30	V6.00.00	第一版



阿尔泰科技

服务热线：400-860-3335

网址：www.art-control.com