

APS_函数库_V2.0

创建日期: 8.20.2020

支持的产品:

DPAC-1000 DPAC-3000

PCI-8392(H)

PCI-8253/56

PCI-8144

PCI(e)-7856

MNET-4XMO MNET-4XMO-(C)

MNET-1XMO

HSL-4XMO HSL-DIO

PCI-8102/PCI-C154(+)

PCI-8154/8158 PCIe-8154/8158

EMX-100

PCI-8254/58 / AMP-204/8C

PCIe-833x

目录

APS_函数库_V2.0	1
目录	2
简介	6
01. 程序库	8
02. 所有函数列表	10
所有函数列表	10
DPAC-1000 函数列表	24
DPAC-3000 函数列表	26
PCI-8392(H) 函数列表.....	28
PCI-8253/56 函数列表.....	32
PCI-8144 函数列表.....	36
PCI(E)-7856 函数列表.....	38
MNET-4XMO 函数列表	40
MNET-4XMO-C 函数列表.....	43
MNET-1XMO 函数列表	47
HSL-4XMO 函数列表	50
HSL-DIO 函数列表.....	52
PCI-8102/PCI-C154(+)函数列表	52
PCI-8154/8158 函数列表	56
PCIe-8154/8158 函数列表	59
EMX-100 函数列表	63
PCI-8254/58 / AMP-204/8C 函数列表	65
PCIe-833x 函数列表.....	73
03. 系统和初始化	79
04. SSCNET 函数	114
05. 运动 IO 和运动状态	137
06. 单轴运动	165
07. 多轴运动触发和停止	195
08. 点动	201
09. 插补	209
10. 高级单步运动和插补	231
11. 中断	298
12. 采样	329
13. DIO & AIO	356
14. 点表运动	380
15. 高级点表	442
16. 现场总线函数	489
17. 齿轮/龙门函数	564

18. 比较触发器	578
19. 程序下载	619
20. 手动脉冲发生器输入	628
21. 螺距误插补偿功能	636
22. DPAC 系统函数	644
23. 非易失性存储器	649
24. 现场总线比较触发	654
25. 看门狗定时器	679
26. VAO/PWM 函数(激光函数)	687
27. 循环限制函数	715
28. 同动函数	718
29. 单锁存(SINGLE-LATCH)函数	724
30. 多锁存(MULTI-LATCH)函数	728
31. 环形计数器函数	742
32. 速度曲线计算	745
33. 背隙函数	750
34. 表定义	753
A. 板卡参数表	753
<i>DPAC-1000</i> 板卡参数表	753
<i>DPAC-3000</i> 板卡参数表	754
<i>PCI-8392(H)</i> 板卡参数表	756
<i>PCI-8253/56</i> 板卡参数表	757
<i>PCI(e)-7856</i> 板卡参数表	760
<i>EMX-100</i> 板卡参数表	760
<i>PCI-8254/58 / AMP-204/8C</i> 板卡参数表	760
<i>PCIe-833x</i> 板卡参数表	763
B. 轴参数表	766
<i>PCI-8392(H)</i> 轴参数表	766
<i>PCI-8253/56</i> 轴参数表	770
<i>PCI-8144</i> 轴参数表	775
<i>MNET-4XMO-(C)</i> 轴参数表	777
<i>MNET-1XMO</i> 轴参数表	783
<i>HSL-4XMO</i> 轴参数表	789
<i>PCI(e)-8154/8158, PCI-8102/PCI-C154(+)</i> 轴参数表	793
<i>EMX-100</i> 轴参数表	802
<i>PCI-8254/58 / AMP-204/8C</i> 轴参数表	807
<i>PCIe-833x</i> 轴参数表	815
C. 采样参数表	822
<i>PCI-8392(H)</i> 和 <i>PCI-8253/56</i> 和 <i>MNET-4XMO</i> 和 <i>PCI-8254/58 / AMP-204/8C</i> 和 <i>PCIe-833x</i> 采样参数表	822

D. 采样源表	823
PCI-8392(H)的采样源表	823
PCI-8253/56 采样源表	824
MNET-4XMO 采样源表	825
PCI-8254/58 / AMP-204/8C sampling source table	825
PCIe-833x 采样源表	827
E. 运动 IO 状态和运动状态定义	828
PCI-8392(H) 运动 IO 状态表。	828
PCI-8253/56 运动 IO 状态表。	828
MNET-4XMO-(C)/1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+)运动 IO 状态表。	829
PCI-8144 运动 IO 状态表。	829
运动 IO 状态描述表.....	829
EMX-100 运动 IO 状态描述表.....	830
EMX-100 运动 IO 状态描述表.....	830
PCI-8254/58 / AMP-204/8C 运动 IO 状态描述表.....	830
PCI-8254/58 / AMP-204/8C 运动 IO 状态描述表.....	830
PCIe-833x 运动 IO 状态描述表.....	831
PCIe-833x 运动 IO 状态描述表.....	831
F. 运动状态定义表	832
PCI-8392(H), 8253/56 运动状态定义表.....	832
MNET-4XMO-(C), HSL-4XMO, PCI(e)-8154/8158, PCI-8102	832
/PCI-C154(+)运动状态定义表.....	832
1XMO 运动状态定义表.....	833
PCI-8144 运动状态定义表.....	833
运动状态描述表.....	833
EMX-100 运动状态描述表.....	834
EMX-100 运动状态描述表.....	834
PCI-8254/58 / AMP-204/8C 运动状态描述表.....	835
PCI-8254/58 / AMP-204/8C 运动状态描述表.....	835
PCIe-833x 运动状态描述表.....	837
PCIe-833x 运动状态描述表.....	837
G. 中断信号表	838
PCI-8392(H) 中断项目定义表.....	838
PCI-8392(H) 轴中断信号定义 (项目 0~15)	838
PCI-8392(H) 轴中断信号描述表.....	838
PCI-8392(H) 系统中断信号的定义(项目 16)	839
PCI-8392(H) 系统中断信号的定义(项目 16)	839
PCI-8392(H) 系统中断信号描述表.....	839
PCI-8253/56 中断信号项目定义表.....	839

<i>PCI-8253/56 轴中断信号定义(项目 0~5)</i>	840
<i>PCI-8253/56 轴中断信号描述表</i>	840
<i>DPAC-1000 中断信号项目定义表</i>	840
<i>DPAC-3000 中断信号项目定义表</i>	841
<i>PCI(e)-7856 中断项目定义表</i>	842
<i>PCI-8144 中断项目定义表</i>	842
<i>MotionNet 中断项目定义表</i>	843
<i>PCI(e)-8154/8158, PCI-8102 中断项目定义表</i>	847
H. 现场总线参数表	858
I. 龙门参数表	860
J. 触发参数表	861
<i>PCI-8253/56 触发参数表</i>	861
<i>MNET-4XMO-C 触发参数表</i>	863
<i>HSL-4XMO 触发参数表</i>	867
<i>DB-8150 触发参数表</i>	869
<i>PCI - C154(+) 触发参数表</i>	873
<i>EMX-100 触发参数表</i>	878
<i>PCI-8254/58 / AMP-204/8C 触发参数表</i>	879
K. 锁存参数表	884
L. 设备信息表	885
M. 现场总线从站参数表	890
N. DPAC 显示索引表	892
O. DPAC 按钮状态表	893
P. SSCNET 伺服监控器源表	894
Q. VAO 参数表	895
35. APS 函数返回代码	897
A. APS 错误代码表	897
B. DSP 运动内核错误代码	898
C. ETHERCAT 主站错误代码	905

简介

APS 即 “自动化产品软件 (Automation Product Software) ” 。APS 库为用户提供了一个统一的接口，用以访问所有支持该函数库的凌华科技产品。它涵盖了大部分的自动化领域，尤其是机器自动化的应用。机器自动化最重要的组成部分是运动控制，而 APS 函数库最早就诞生于运动控制。运动控制系统中那些协同工作的组件，包含系统平台管理、现场总线通信功能、通用数字输入/输出、通用模拟输入/输出以及各种计数器/计时器支持，都是 APS 中内置的组件。APS 函数库是凌华科技产品在自动化领域中的一种一站式解决方案。

该函数库具有如下优势：

- A. 独立于硬件
- B. 独立于操作系统
- C. 编程风格一致

第一个优势是独立于硬件。在过去，每种产品都有其自己的软件函数集。每次用户为了同一个目的需要添加或者删除不同种类的产品时，都必须对其软件进行重新编程，以适应用户需求的改变。大多数的时候，用户必须重新研究新函数的用法，因此，在应用开发和维护方面不得不付出巨大的努力，而且未必能做到按时交付。现在，用户只要使用 APS 库并将其作为软件的中间层，就可以轻松地重新使用与 APS 函数库对接的软件组件，而无需关心那些不同种类的硬件产品。这就是独立于硬件的含义所在。

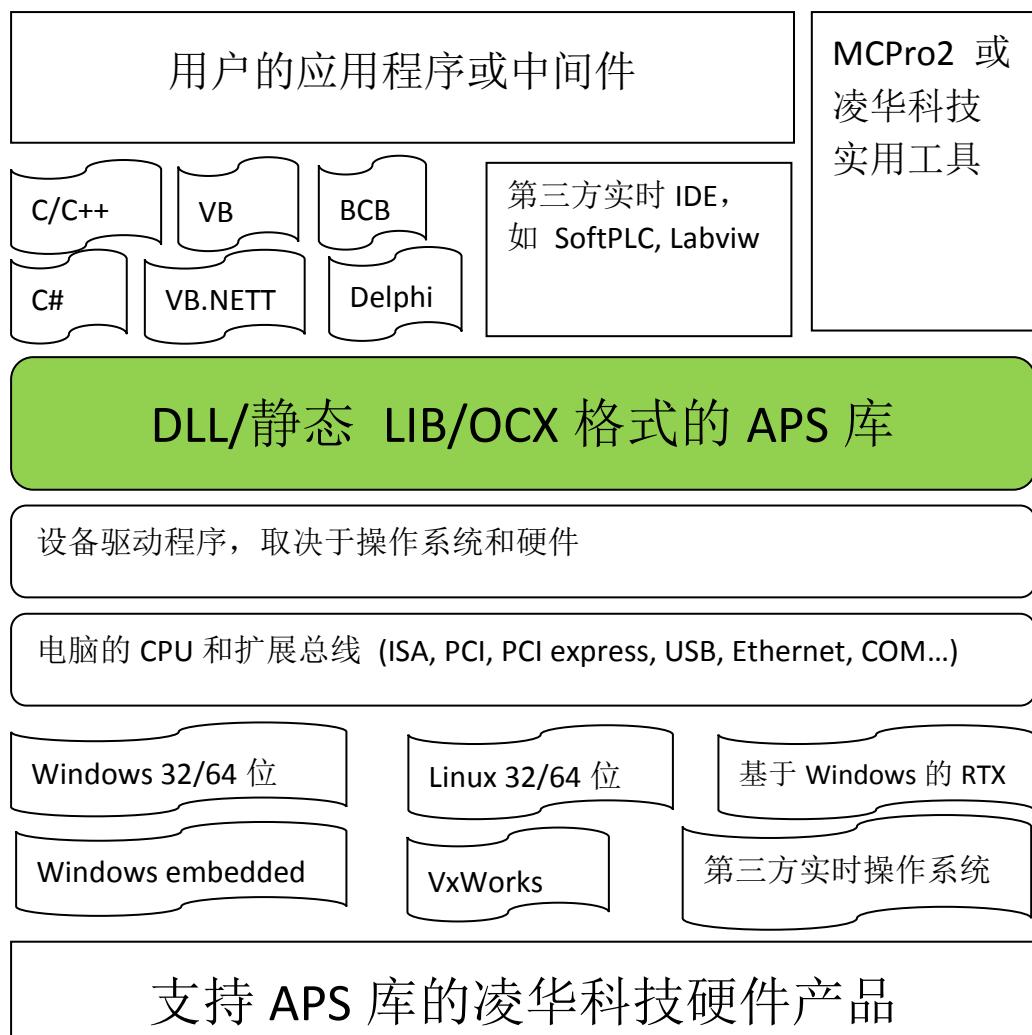
第二个优势是与操作系统无关。我们将持续进行研究和开发，以支持最新的操作系统。APS 标准软件包支持 Microsoft Windows 系列操作系统，例如

Windows XP/2000/Vista 以及即将推出的 Windows 操作系统。无论是 32 位还是 64 位，单核或者多核平台，APS 都可以确保在所有的操作系统中运行相同的函数，用户无需为此担心。APS 为用户节省了很多的时间，让他们可以更加专注于他们的机器设计。对于非 Windows 的操作系统，APS 也可以提供支持，不仅支持 Linux 和 DOS 等通用操作系统，还支持 RTX、VxWorks 等实时操作系统。这个好处是可以帮助用户进行产品定位——从低端机器到高端机器。

第三个优势是编程风格一致。APS 库让不同类型的应用（如运动控制、I/O 控制和通信）拥有相同的编程风格。无论是步进电机还是伺服电机，无论是分布式的还是集中式的拓扑结构，APS 库在编程和参数定义方面都具有相同的风格。APS 库还为 ANSI C/C++，Microsoft Visual C/C++，Visual Basic，C#，Visual Basic.NET 和 Borland Delphi，C/C++ builder 等用户提供了各种编程语言接口和示例，符合了不同用户基于不同目的进行的机器开发。APS 库还提供了基于 Windows 操作系统的可视化用户界面，用以测试产品的所有函数。换句话说，任何产品只要支持 APS 库，那么实用程序也能够支持这些产品。这个实用程序被称作“MotionCreatorPro2”或者更新的版本。对于软件程序员和系统设置人员来说，这是一大利好，因为用户在验证控制结果和硬件功能之前，不用编写任何代码。这是一个非常好的方法，可以从产品测试贯穿到系统开发和调试阶段。

APS 库不仅仅是一个库，它还是凌华科技提供的一个总的软件包，包含各种操作系统下的设备驱动程序、动态或者静态的链接库、各种编程语言接口、可视化实用程序、版本控制信息、丰富的文档、长期支持以及可一步安装的软件。APS 库支持大部分的凌华科技自动化产品，尤其是机器控制领域的相关产品。利用 APS 库，用户可以大幅减少开发时间，并且不必担心电脑 CPU 和操作系统的改变。

下图描述了 APS 库在软件层次架构中的位置。



01. 程序库

APS 支持多种编程语言。APS 库的头文件包含了函数声明、类型定义和常量可变定义。以下是 C/C++ 库的示例，其他请参考相应编程语言已安装的头文件。

函数原型和一些常见的数据类型在 **APS168.h** 中做了声明。为了实现兼容，我们建议在您的应用程序中使用这些数据类型。下表展示了数据类型的名称和数字范围。

类型 名称	C/C++ 数据类型	描述	范围
U8	unsigned char	8位 ASCII 字符	0 至 255
I16	Short	16位有符号整数	-32768 至 32767

U16	unsigned short	16位无符号整数	0 至 65535
I32	long	32位有符号长整数	-2147483648 至 2147483647
U32	unsigned long	32位无符号长整数	0 至 4294967295
F32	Float	32位单精度浮点	-3.402823E38 至 3.402823E38
F64	double	64位双精度浮点	-1.797683134862315E308 至 1.797683134862315E309
Boolean	Char	布尔逻辑值	TRUE, FALSE

APS 库的命名规则采用目的的全称。

在 ‘C’ 编程环境下:

APS_{purpose_name}.

例如 **APS_initial()**, **APS_get_position()**, **APS_relative_move()**

02. 所有函数列表

所有函数列表

部分	函数名称	描述
錯誤! 找不到 參照來 源。	系统和初始化	
	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_disable_device</u>	禁用卡
	<u>APS_set_board_param</u>	设置板卡的参数
	<u>APS_get_board_param</u>	获取板卡的参数
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_set_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_axis_param_f</u>	获取 64 位的轴参数
	<u>APS_get_system_timer</u>	获取系统计时器计数器
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_save_parameter_to_flash</u>	将系统和轴参数保存到闪存中
	<u>APS_load_parameter_from_flash</u>	从闪存中加载系统和轴参数
	<u>APS_load_parameter_from_default</u>	以默认值加载系统和轴参数。
	<u>APS_set_security_key</u>	设置安全密码
	<u>APS_check_security_key</u>	检查安全密码
	<u>APS_reset_security_key</u>	重置安全密码
	<u>APS_get_first_axisId</u>	获取卡的第一个轴 ID
	<u>APS_save_param_to_file</u>	将参数保存到文件中
	<u>APS_load_param_from_file</u>	从文件中加载参数
	<u>APS_register_emx</u>	在库中注册 EMX
	<u>APS_get_deviceIP</u>	获取设备 IP
	<u>APS_reset_emx_alarm</u>	重置设备的报警信号
	<u>APS_get_curr_sys_ctrl_mode</u>	获取当前系统的控制模式
錯誤! 找不到 參照來 源。	SSCNET 函数	
	<u>APS_start_sscnet</u>	开启 SSCNET 网络
	<u>APS_stop_sscnet</u>	停止 SSCNET 网络
	<u>APS_get_sscnet_servo_param</u>	读取当前伺服参数值
	<u>APS_set_sscnet_servo_param</u>	设置伺服参数
	<u>APS_get_sscnet_servo_alarm</u>	获取当前伺服的报警信息

	<u>APS_reset_sscnet_servo_alarm</u>	伺服报警复位
	<u>APS_save_sscnet_servo_param</u>	将伺服参数保存到闪存中
	<u>APS_get_sscnet_servo_abs_position</u>	从伺服驱动器获取绝对参考位置
	<u>APS_save_sscnet_servo_abs_position</u>	将绝对参考位置保存到闪存中
	<u>APS_load_sscnet_servo_abs_position</u>	从闪存中加载绝对参考位置
	<u>APS_get_sscnet_link_status</u>	获取 SSCNET 链接状态
	<u>APS_set_sscnet_servo_monitor_src</u>	设置伺服监控数据源
	<u>APS_get_sscnet_servo_monitor_src</u>	获取伺服监控数据源
	<u>APS_get_sscnet_servo_monitor_data</u>	获取伺服监控数据
运动 IO 和运动状态		
錯誤! 找不到 参照來 源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动 IO 状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_feedback_velocity</u>	获取反馈速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取 64 位的反馈位置
	<u>APS_set_position_f</u>	设置 64 位的反馈位置
	<u>APS_get_command_f</u>	获取 64 位的命令位置
	<u>APS_set_command_f</u>	设置 64 位的命令位置设置
	<u>APS_get_error_position_f</u>	获取 64 位的错误位置
	<u>APS_get_target_position_f</u>	获取 64 位的目标位置
	<u>APS_get_command_velocity_f</u>	获取 64 位的命令速度
	<u>APS_get_feedback_velocity_f</u>	获取 64 位的反馈速度
	<u>APS_get_mq_free_space</u>	获取运动队列的可用空间
	<u>APS_get_mq_usage</u>	获取运动队列的使用情况
	<u>APS_get_stop_code</u>	获取停止代码
	<u>APS_get_encoder</u>	获取原始的反馈计数器
	<u>APS_get_command_counter</u>	获取原始的命令计数器
	<u>APS_reset_command_counter</u>	重置原始的命令计数器
	<u>APS_get_actual_torque</u>	获取实际扭矩值
	<u>APS_get_axis_latch_data</u>	获取 ORG/EZ 锁存器数据
单轴运动		
錯誤! 找不到	<u>APS_relative_move</u>	开始一个相对距离的运动

參照來 源。	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_relative_move2</u>	根据速度曲线开始一个相对距离的运动
	<u>APS_absolute_move2</u>	根据速度曲线开始一个绝对位置的运动
	<u>APS_home_move2</u>	根据速度曲线开始返回原点
	<u>APS_speed_override</u>	快速改变速度
	<u>APS_relative_move_ovrd</u>	开始一个相对距离的运动或以新的距离和速度覆盖它
	<u>APS_absolute_move_ovrd</u>	开始一个绝对位置的运动或以新的位置和速度覆盖它
	<u>APS_home_escape</u>	离开原点开关
錯誤! 找不到 參照來 源。	多軸運動觸發和停止	
	<u>APS_move_trigger</u>	发送一个触发以同步所有等待的运动
	<u>APS_stop_move_multi</u>	停止多轴运动
	<u>APS_emg_stop_multi</u>	立即停止多轴运动
錯誤! 找不到 參照來 源。	點動	
	<u>APS_set_jog_param</u>	设置点动参数
	<u>APS_get_jog_param</u>	获取点动参数
	<u>APS_jog_mode_switch</u>	启用/禁用点动
	<u>APS_jog_start</u>	开始/停止点动
錯誤! 找不到 參照來 源。	插補	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
	<u>APS_absolute_arc_move_3pe</u>	利用通过和终点法开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move_3pe</u>	利用通过和终点法开始一个相对距离的圆弧插补
	<u>APS_absolute_helix_move</u>	开始一个绝对位置的螺旋插补
	<u>APS_relative_helix_move</u>	开始一个相对距离的螺旋插补
	<u>APS_absolute_helical_move</u>	开始一个绝对位置的螺旋插补
	<u>APS_relative_helical_move</u>	开始一个相对距离的螺旋插补
錯誤! 找不到 參照來 源。	高級單步運動和插補	
	<u>APS_ptp</u>	开始一个单步运动
	<u>APS_ptp_v</u>	根据运动曲线开始一个单步运动

源。	<u>APS_ptp_all</u>	根据所有曲线开始一个单步运动
	<u>APS_vel</u>	开始一个速度运动
	<u>APS_vel_all</u>	根据所有曲线开始一个速度运动
	<u>APS_line</u>	开始一个线性运动
	<u>APS_line_v</u>	根据运动曲线开始一个线性运动
	<u>APS_line_all</u>	根据所有曲线开始一个线性运动
	<u>APS_arc2_ca</u>	开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据运动曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca</u>	开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据速度曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce</u>	开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_v</u>	根据速度曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_all</u>	根据所有曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc3_ca</u>	开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_v</u>	根据速度曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_all</u>	根据所有曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ce</u>	开始一个终点位置类型的 3D 运动
	<u>APS_arc3_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 运动
	<u>APS_arc3_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 运动
	<u>APS_spiral_ca</u>	开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_v</u>	根据速度曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_all</u>	根据所有曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ce</u>	开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 螺旋运动

	<u>APS_spiral_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 螺旋运动
錯誤! 找不到 參照來 源。		中断
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_wait_error_int</u>	等待错误中断 (非屏蔽)
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄。 (Win32)
	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
	<u>APS_set_field_bus_int_factor_motion</u>	启用/禁用运动中断信号并为 MotionNet 系列获取中断句柄
	<u>APS_get_field_bus_int_factor_motion</u>	为 MotionNet 系列获取运动中断信号启用或禁用。
	<u>APS_set_field_bus_int_factor_error</u>	为 MotionNet 系列启用/禁用错误中断信号并获取中断句柄。
	<u>APS_get_field_bus_int_factor_error</u>	为 MotionNet 系列获取错误的中断信号状态。
	<u>APS_reset_field_bus_int_motion</u>	重置 MotionNet 系列轴的中断状态
	<u>APS_wait_field_bus_error_int_motion</u>	等待 MotionNet 系列的错误中断事件。
錯誤! 找不到 參照來 源。	<u>APS_set_field_bus_int_factor_di</u>	分配数字输入中断位并为 HSL 系列获得中断句柄
	<u>APS_get_field_bus_int_factor_di</u>	获取分配给 HSL 系列的数字输入中断位。
		采样
	<u>APS_set_sampling_param</u>	设置采样参数。
	<u>APS_get_sampling_param</u>	获取采样参数。
	<u>APS_wait_trigger_sampling</u>	等待采样数据。
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数。

		8 个通道
	<u>APS_get_sampling_param_ex</u>	按照结构获取采样参数。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_ex</u>	等待采样数据。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_async_ex</u>	等待异步采样数据。通过一个扩展至 8 个通道
	<u>APS_get_sampling_data_ex</u>	在自动采样模式下获取采样数据。通过一个扩展至 8 个通道
DIO & AIO		
13	<u>APS_set_field_bus_d_channel_output</u>	按通道设置现场总线数字输出
	<u>APS_get_field_bus_d_channel_output</u>	按通道获取现场总线数字输出
	<u>APS_get_field_bus_d_channel_input</u>	按通道获取现场总线数字输入
	<u>APS_set_field_bus_d_port_output</u>	按端口设置现场总线数字输出
	<u>APS_get_field_bus_d_port_input</u>	按端口设置现场总线数字输入
	<u>APS_get_field_bus_d_port_output</u>	按端口设置现场总线数字输出
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字量输出值
	<u>APS_read_d_input</u>	读取数字输入值
	<u>APS_write_d_channel_output</u>	设置每通道的数字输出值
	<u>APS_read_d_channel_output</u>	读取每通道的数字输出值
	<u>APS_read_d_channel_input</u>	读取每通道的数字输入值
	<u>APS_read_a_input_value</u>	回读模拟输入值 (伏特)
	<u>APS_read_a_input_data</u>	回读模拟输入原始数据
	<u>APS_write_a_output_value</u>	设置模拟输出值 (伏特)
	<u>APS_write_a_output_data</u>	将原始数据设置为模拟输出值
点表运动		
錯誤! 找不到 參照來 源。	<u>APS_set_point_table</u>	设置点表运动参数
	<u>APS_get_point_table</u>	获取点表运动参数
	<u>APS_point_table_move</u>	开始一个点表运动
	<u>APS_get_running_point_index</u>	当轴执行一个点表运动时，获取当前的点索引
	<u>APS_get_start_point_index</u>	当轴执行一个点表运动时，获取第一个点索引
	<u>APS_get_end_point_index</u>	当轴执行一个点表运动时，获取点索引的结尾。
	<u>APS_set_table_move_pause</u>	暂停点表运动
	<u>APS_set_table_move_ex_pause</u>	减速以停止运动并控制 I/O
	<u>APS_set_table_move_ex_rollback</u>	回滚到当前点索引的起始位置
	<u>APS_set_table_move_ex_resume</u>	重新开始点表运动并保持 I/O 状态

	<u>APS_set_table_move_repeat</u>	设置点表运动重复
	<u>APS_set_point_table_mode2</u>	设置点表模式
	<u>APS_set_point_table2</u>	设置点表
	<u>APS_point_table_continuous_move2</u>	开始一个点表的连续运动
	<u>APS_point_table_single_move2</u>	开始一个点表的单步运动
	<u>APS_get_running_point_index2</u>	当轴执行一个点运动时，获取当前点的运动索引
	<u>APS_point_table_status2</u>	获取点表状态
	<u>APS_set_point_table3</u>	设置点表
	<u>APS_point_table_move3</u>	开始一个点表单步运动
	<u>APS_set_point_table_param3</u>	设置速度参数
	<u>APS_set_feeder_group</u>	将轴设置给馈线(feeder)组
	<u>APS_get_feeder_group</u>	在一个馈线(feeder)组中返回配置
	<u>APS_free_feeder_group</u>	释放馈线(feeder)组及其资源
	<u>APS_reset_feeder_buffer</u>	重置馈线(feeder)的点缓冲区
	<u>APS_set_feeder_point_2D</u>	在馈线(feeder)缓冲区中添加一个点
	<u>APS_set_feeder_point_2D_ex</u>	在馈线(Feeder)的缓冲区中添加一个点
	<u>APS_start_feeder_move</u>	开始点表运动和馈点 (feed point)
	<u>APS_get_feeder_status</u>	获取馈线(Feeder)状态
	<u>APS_get_feeder_running_index</u>	获取正在运行的点。
	<u>APS_get_feeder_feed_index</u>	获取将哪个点设置到点表中。
	<u>APS_set_feeder_ex_pause</u>	运动已暂停 (已停止) 和馈线(feeder)已暂停
	<u>APS_set_feeder_ex_rollback</u>	返回到暂停索引的起始位置
	<u>APS_set_feeder_ex_resume</u>	恢复点表运动。
	<u>APS_set_point_table_ex</u>	设置点表运动参数的扩展选项
	<u>APS_get_point_table_ex</u>	获取点表运动参数的扩展选项
錯誤！找不到參照來源。	高级点表	
	<u>APS_pt_enable</u>	启用点表
	<u>APS_pt_disable</u>	禁用点表
	<u>APS_get_pt_info</u>	获取点表信息
	<u>APS_pt_set_vs</u>	将速度配置设置到点表中
	<u>APS_pt_get_vs</u>	获取点表中的速度配置
	<u>APS_pt_start</u>	将控制命令设置到点表中
	<u>APS_pt_stop</u>	停止点表
	<u>APS_get_pt_status</u>	获取点表的状态
	<u>APS_reset_pt_buffer</u>	重置点表缓冲区
	<u>APS_pt_roll_back</u>	回滚到上一点

	<u>APS_get_pt_error</u>	获取点表的错误代码
	<u>APS_pt_dwell</u>	将一个停顿运动推送到点表的点缓冲区。
	<u>APS_pt_line</u>	将一个线性运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ca</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ce</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ca</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ce</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ca</u>	将一个螺旋运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ce</u>	将一个螺旋运动推送到点表的点缓冲区。
	<u>APS_pt_ext_set_do_ch</u>	将 Do 扩展命令设置到命令缓冲区。当将一个运动推送到点表中时，命令缓冲区处于活动状态。
	<u>APS_pt_set_absolute</u>	将绝对曲线设置到曲线缓冲区
	<u>APS_pt_set_relative</u>	将相对曲线设置到曲线缓冲区
	<u>APS_pt_set_trans_buffered</u>	在曲线缓冲区中将转换设置为缓冲模式。
	<u>APS_pt_set_trans_inp</u>	在曲线缓冲区中将转换设置为就位模式。
	<u>APS_pt_set_trans_blend_dec</u>	在曲线缓冲区中将转换设置为带减速的混合模式。
	<u>APS_pt_set_trans_blend_dist</u>	在曲线缓冲区中将转换设置为带剩余距离的混合模式。
	<u>APS_pt_set_trans_blend_pcnt</u>	在曲线缓冲区中将转换设置为带剩余距离百分比的混合模式。
	<u>APS_pt_set_acc</u>	将加速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_dec</u>	将减速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_acc_dec</u>	将加速/减速曲线设置到曲线缓冲区中
	<u>APS_pt_set_s</u>	将 S-factor 曲线设置到曲线缓冲区中
	<u>APS_pt_set_vm</u>	将最大速度曲线设置到曲线缓冲区中
	<u>APS_pt_set_ve</u>	将终点速度曲线设置到曲线缓冲区中
錯誤! 找不到 參照來 源。	<u>現場總線函數</u>	
	<u>APS_set_field_bus_param</u>	設置現場總線相關參數
	<u>APS_get_field_bus_param</u>	獲取現場總線相關參數
	<u>APS_scan_field_bus</u>	掃描現場總線並生成 ENI 檔案
	<u>APS_start_field_bus</u>	啟動指定現場總線的網絡
	<u>APS_stop_field_bus</u>	停止指定現場總線的網絡
	<u>APS_field_bus_d_set_output</u>	設置現場總線數字輸出
	<u>APS_field_bus_d_get_output</u>	獲取現場總線數字輸出
	<u>APS_field_bus_d_get_input</u>	獲取現場總線數字輸入
	<u>APS_field_bus_d_set_output_ex</u>	為 64 位 DIO 操作設置現場總線數字輸出
	<u>APS_field_bus_d_get_output_ex</u>	為 64 位 DIO 操作獲取現場總線數字輸出

<u>APS_field_bus_d_get_input_ex</u>	为 64 位 DIO 操作获取现场总线数字输入
<u>APS_set_field_bus_slave_param</u>	设置现场总线从站模块的参数
<u>APS_get_field_bus_slave_param</u>	获取现场总线从站模块的参数
<u>APS_set_field_bus_a_output</u>	设置现场总线模拟输出
<u>APS_get_field_bus_a_output</u>	获取现场总线模拟输出
<u>APS_get_field_bus_a_input</u>	获取现场总线模拟输入
<u>APS_get_slave_connect_quality</u>	获取从站的连接质量
<u>APS_get_slave_online_status</u>	获取从站的在线状态
<u>APS_get_field_bus_master_status</u>	获取现场总线主站状态
<u>APS_get_field_bus_last_scan_info</u>	获取现场总线上的主站类型
<u>APS_get_field_bus_master_type</u>	获取现场总线上的从站类型
<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站名称
<u>APS_get_field_bus_slave_name</u>	获取从站模块的第一个轴
<u>APS_get_field_bus_slave_first_axisno</u>	获取指定现场总线上的设备信息
<u>APS_get_field_bus_device_info</u>	获取现场总线上的主站类型
<u>APS_get_field_bus_module_info</u>	获取从站信息
<u>APS_reset_field_bus_alarm</u>	重置从站的报警信号
<u>APS_get_field_bus_alarm</u>	获取从站的报警代码
<u>APS_get_field_bus_pdo</u>	从 PDO 内存中获取值
<u>APS_set_field_bus_pdo</u>	为 PDO 内存赋值
<u>APS_get_field_bus_pdo_offset</u>	获取 PDO 信息
<u>APS_get_field_bus_sdo</u>	从从站中获取 SDO 数据
<u>APS_set_field_bus_sdo</u>	为从站设置 SDO 数据
<u>APS_set_field_bus_od_data</u>	设置 EtherCAT OD 原始数据
<u>APS_get_field_bus_od_data</u>	获取 EtherCAT OD 原始数据
<u>APS_get_field_bus_od_module_info</u>	获取 EtherCAT 从站信息
<u>APS_get_field_bus_module_map</u>	在手动 ID 模式下获取映射的从站 ID
<u>APS_set_field_bus_module_map</u>	在手动 ID 模式下设置映射的从站 ID
<u>APS_get_field_bus_slave_state</u>	获取从站状态机器的状态
<u>APS_set_field_bus_slave_state</u>	设置从站状态机器的状态
<u>APS_get_field_bus_ESC_register</u>	获取 EtherCAT 从站控制器寄存器

	<u>APS_set_field_bus_ESC_register</u>	设置 EtherCAT 从站控制器寄存器
	<u>APS_get_system_loading</u>	获取系统循环加载
	<u>APS_get_field_bus_analysis_topology</u>	获取当前和过去的拓扑结构，然后进行分析
	<u>APS_get_field_bus_loss_package</u>	在接收总线方向上获取 EtherCAT 帧计数的丢失信息。
齿轮/龙门函数		
	<u>APS_set_gantry_param</u>	设置龙门函数相关参数
	<u>APS_get_gantry_param</u>	获取龙门函数相关参数
	<u>APS_set_gantry_axis</u>	在龙门组中设置两个轴
	<u>APS_get_gantry_axis</u>	获取龙门组中的轴
	<u>APS_get_gantry_error</u>	获取龙门轴偏差误差
	<u>APS_get_encoder</u>	获取编码器（用于龙门架归零补偿）
	<u>APS_get_latch_event</u>	按轴获取锁存事件（用于龙门架归零补偿）
	<u>APS_get_latch_counter</u>	按轴获取锁存计数器（用于龙门架归零补偿）
	<u>APS_start_gear</u>	启用/禁用一个指定的齿轮模式
	<u>APS_get_gear_status</u>	获取齿轮状态
	<u>APS_get_gantry_number</u>	获取主站对应的从站数量
	<u>APS_get_gantry_info</u>	获取从轴 ID 数组
	<u>APS_get_gantry_deviation</u>	获取主从站之间的位置偏差
比较触发		
	<u>APS_set_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_trigger_linear</u>	设置线性比较函数
	<u>APS_set_trigger_table</u>	设置表比较函数
	<u>APS_set_trigger_manual</u>	手动输出触发
	<u>APS_set_trigger_manual_s</u>	手动同步输出触发
	<u>APS_get_trigger_table_cmp</u>	获取当前的表比较值
	<u>APS_get_trigger_linear_cmp</u>	获取当前的线性比较值
	<u>APS_get_trigger_count</u>	获取触发计数。
	<u>APS_reset_trigger_count</u>	重置触发计数。
	<u>APS_enable_trigger_fifo_cmp</u>	启用触发 FIFO 比较器
	<u>APS_get_trigger_fifo_cmp</u>	获取触发 FIFO 比较器
	<u>APS_get_trigger_fifo_status</u>	获取触发 FIFO 状态
	<u>APS_set_trigger_fifo_data</u>	设置触发 FIFO 状态
	<u>APS_start_timer</u>	启动计时器

	<u>APS_get_timer_counter</u>	获取计时器计数值
	<u>APS_set_timer_counter</u>	设置计时器计数值
	<u>APS_start_trigger_timer</u>	启动计时器
	<u>APS_get_trigger_timer_counter</u>	获取计时器计数值
	<u>APS_set_multi_trigger_table</u>	设置表比较函数
	<u>APS_get_multi_trigger_table_cmp</u>	获取当前表比较值
	<u>APS_set_trigger_table_data</u>	设置表比较器数据 (快速比较表触发函数)
	<u>APS_get_trigger_table_status</u>	获取表比较器状态 (快速比较表触发函数)
	<u>APS_get_trigger_cmp_value</u>	获取表比较器值 (快速比较表触发函数)
	<u>APS_enable_trigger_table</u>	启用表比较器 (快速比较表触发函数)
	<u>APS_reset_trigger_table</u>	重置表比较器 (快速比较表触发函数)
錯誤! 找不到 參照來 源。	程序下载(*1)	
	<u>APS_load_vmc_program</u>	将 VMC 文件加载到任务存储器
	<u>APS_save_vmc_program</u>	从任务存储器中保存至 VMC 文件
	<u>APS_set_task_mode</u>	设置任务运行模式
	<u>APS_get_task_mode</u>	获取任务运行模式
	<u>APS_start_task</u>	启动任务控制命令
	<u>APS_get_task_info</u>	获取任务信息
	<u>APS_get_task_msg</u>	获取所有任务的信息
錯誤! 找不到 參照來 源。	手动脉冲发生器函数	
	<u>APS_manual_pulser_start</u>	启用/禁用手动脉冲发生器输入
	<u>APS_manual_pulser_velocity_move</u>	开始一个脉冲速度运动
	<u>APS_manual_pulser_relative_move</u>	开始一个脉冲相对距离运动
	<u>APS_manual_pulser_home_move</u>	开始一个脉冲归零运动
	<u>APS_get_pulser_counter</u>	获取脉冲输入计数器
	<u>APS_set_pulser_counter</u>	设置脉冲输入计数器
錯誤! 找 不 到 參 照來 源。	螺距误插补偿功能	
	<u>APS_set_pitch_table</u>	设置螺距误插补偿表的配置和数据
	<u>APS_get_pitch_table</u>	获取螺距误插补偿表的配置和数据
	<u>APS_start_pitch_comp</u>	开始螺距误插补偿
22	DPAC 系统函数	
	<u>APS_rescan_CF</u>	重新扫描 DPAC 从站 CF 插槽
	<u>APS_get_battery_status</u>	获取 DPAC SRAM 电池状态
	<u>APS_get_display_data</u>	获取 7 段显示数据
	<u>APS_set_display_data</u>	设置 7 段显示数据
錯誤!	非易失性 RAM 函数	

找不到 参照來 源。	<u>APS_set_nv_ram</u>	设置 RAM 数据
	<u>APS_get_nv_ram</u>	获取 RAM 数据
	<u>APS_clear_nv_ram</u>	清除 RAM 数据
錯誤! 找不到 參照來 源。	现场总线比较触发	
	<u>APS_set_field_bus_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_field_bus_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_field_bus_trigger_linear</u>	设置线性比较函数
	<u>APS_set_field_bus_trigger_table</u>	设置表比较函数
	<u>APS_set_field_bus_trigger_manual</u>	手动输出触发
	<u>APS_set_field_bus_trigger_manual_s</u>	手动同步输出触发
	<u>APS_get_field_bus_trigger_table_cmp</u>	获取当前表比较值
	<u>APS_get_field_bus_trigger_linear_cmp</u>	获取当前线性比较值
	<u>APS_get_field_bus_trigger_count</u>	获取触发计数。
	<u>APS_reset_field_bus_trigger_count</u>	重置触发计数。
	<u>APS_get_field_bus_linear_cmp_remain_count</u>	获取线性比较的剩余计数值
	<u>APS_get_field_bus_table_cmp_remain_count</u>	获取表比较的剩余计数值
	<u>APS_get_field_bus_encoder</u>	获取编码器计数值
	<u>APS_set_field_bus_encoder</u>	设置编码器计数值
錯誤! 找不到 參照來 源。	看门狗定时器	
	<u>APS_wdt_start</u>	开启/停至看门狗定时器
	<u>APS_wdt_get_timeout_period</u>	获取看门狗定时器的超时时间
	<u>APS_wdt_reset_counter</u>	重置看门狗计时器的计数器
	<u>APS_wdt_get_counter</u>	获取看门狗计时器的计数器
	<u>APS_wdt_set_action_event</u>	设置看门狗定时器的动作事件
26	VAO/PWM 函数(激光函数)	
	<u>APS_set_vao_param</u>	将参数设置到 VAO 参数表中
	<u>APS_get_vao_param</u>	获取 VAO 参数表中的参数
	<u>APS_set_vao_table</u>	设置 VAO 表
	<u>APS_switch_vao_table</u>	切换到指定的 VAO 表
	<u>APS_start_vao</u>	启用 VAO 输出通道
	<u>APS_get_vao_status</u>	获取 VAO 状态
	<u>APS_check_vao_param</u>	检查指定 VAO 表的参数设置
	<u>APS_set_vao_param_ex</u>	通过 VAO 结构设置表参数。
	<u>APS_get_vao_param_ex</u>	通过 VAO 结构获取表参数。
	<u>APS_set_pwm_on</u>	开始输出 PWM 信号
	<u>APS_set_pwm_width</u>	将脉冲宽度设置至一个 PWM 通道

	<u>APS_set_pwm_frequency</u>	将脉冲频率设置至一个 PWM 通道
	<u>APS_get_pwm_width</u>	从一个 PWM 通道获取脉冲宽度
	<u>APS_get_pwm_frequency</u>	从一个 PWM 通道获取脉冲频率
錯誤! 找不到 參照來 源。	环形限位函数	
	<u>APS_set_circular_limit</u>	设置环形限位配置
	<u>APS_get_circular_limit</u>	获取环形限位配置
錯誤! 找不到 參照來 源。	同动函数	
	<u>APS_set_relative_simultaneous_move</u>	设置一个相对的同动
	<u>APS_set_absolute_simultaneous_move</u>	设置一个绝对的同动
	<u>APS_start_simultaneous_move</u>	开始同动
	<u>APS_stop_simultaneous_move</u>	停止同动
錯誤! 找不到 參照來 源。	单锁存(Single-latch)函数	
	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据
錯誤! 找不到 參照來 源。	多锁存(Multi-latch)函数	
	<u>APS_set_ltc_counter</u>	设置锁存计数器
	<u>APS_get_ltc_counter</u>	获取轴的锁存数据
	<u>APS_set_ltc_fifo_param</u>	设置锁存参数
	<u>APS_get_ltc_fifo_param</u>	获取锁存参数
	<u>APS_manual_latch</u>	手动和同步锁存数据。
	<u>APS_enable_ltc_fifo</u>	启用/禁用锁存fifo
	<u>APS_reset_ltc_fifo</u>	重置锁存fifo
	<u>APS_get_ltc_fifo_data</u>	从fifo中获取一个锁存数据
	<u>APS_get_ltc_fifo_usage</u>	获取锁存fifo的使用情况
	<u>APS_get_ltc_fifo_free_space</u>	获取锁存fifo的可用空间
	<u>APS_get_ltc_fifo_status</u>	获取fifo状态
	<u>APS_get_ltc_fifo_point</u>	获取锁存点数组
錯誤! 找不到 參照來 源。	环形计数器函数	
	<u>APS_set_ring_counter</u>	启用环形计数器函数
	<u>APS_get_ring_counter</u>	获取环形计数器的极限值
錯誤! 找不到 參照來 源。	速度曲线计算	
	<u>APS_relative_move_profile</u>	获取相对速度曲线(PCI-C154)
	<u>APS_absolute_move_profile</u>	获取绝对速度曲线(PCI-C154)
	<u>APS_check_motion_profile_emx</u>	获取相对速度曲线(EMX-100)
錯誤! 找不到 參照來 源。	背隙函数	
	<u>APS_set_backlash_en</u>	启用/停止背隙
	<u>APS_get_backlash_en</u>	检查背隙是否启用/停止

<p style="text-align: center;">錯誤! 找不到 參照來 源。</p>	<u>表定义</u>
	<u>板卡参数表</u>
	<u>轴参数表</u>
	<u>采样参数表</u>
	<u>采样源表</u>
	<u>运动IO状态和运动状态定义</u>
	<u>运动状态定义表</u>
	<u>中断信号表</u>
	<u>现场总线参数表</u>
	<u>龙门参数表</u>
	<u>触发参数表</u>
	<u>锁存参数表</u>
	<u>设备信息表</u>
	<u>现场总线从站参数表</u>
	<u>DPAC显示索引表</u>
	<u>DPAC按钮状态表</u>
	<u>SSCNET伺服监控源表</u>
	<u>VAO参数表</u>
	<u>APS函数返回码</u>

DPAC-1000 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_card_name</u>	获取板卡的参数
	<u>APS_set_board_param</u>	设置板卡的参数
	<u>APS_get_board_param</u>	获取设备信息
	<u>APS_get_device_info</u>	获取板卡的名称
	<u>APS_load_param_from_file</u>	从文件中加载参数
中断		
錯 誤! 找不 到參 照來 源。	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄 (Win32)
DIO & AIO		
13	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
	<u>APS_read_d_input</u>	读取数字输入值
手动脉冲发生器函数		
錯 誤! 找不 到參 照來 源。	<u>APS_get_pulser_counter</u>	获取脉冲输入计数器
	<u>APS_set_pulser_counter</u>	设置脉冲输入计数器
DPAC 系统函数		
22	<u>APS_rescan_CF</u>	重新扫描DPAC从站CF插槽
	<u>APS_get_battery_status</u>	获取DPAC SRAM电池状态
	<u>APS_get_display_data</u>	获取7段显示数据
	<u>APS_set_display_data</u>	设置7段显示数据
	<u>APS_get_button_status</u>	获取按钮输入状态
非易失性 RAM 函数		
錯 誤!	<u>APS_set_nv_ram</u>	设置 RAM 数据

找不到參照來源。	<u>APS_get_nv_ram</u>	获取 RAM 数据
	<u>APS_clear_nv_ram</u>	清除 RAM 数据
錯 誤! 找不到參照來源。	<u>表定义</u>	
	板卡参数定义表	
	中断信号表	
	设备信息表	
	DPAC 显示索引表	
	DPAC 按钮状态表	
	APS 函数返回码	

DPAC-3000 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_set_board_param</u>	设置板卡的参数
	<u>APS_get_board_param</u>	获取板卡的参数
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_load_param_from_file</u>	从文件中加载参数
中断		
錯 誤! 找不 到參 照來 源。	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
DIO & AIO		
13	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
	<u>APS_read_d_input</u>	读取数字输入值
现场总线函数		
錯 誤! 找不 到參 照來 源。	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_set_field_bus_slave_param</u>	设置现场总线从站模块的参数
	<u>APS_get_field_bus_slave_param</u>	获取现场总线从站模块的参数
	<u>APS_set_field_bus_a_output</u>	设置现场总线模拟输出
 		
 		
 		

	<u>APS_get_field_bus_a_input</u>	获取现场总线模拟输入
	<u>APS_get_slave_connect_quality</u>	获取从站的连接质量
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
錯 誤! 找不 到參 照來 源。	手动脉冲发生器输入	
	<u>APS_get_pulser_counter</u>	获取脉冲输入计数器
	<u>APS_set_pulser_counter</u>	设置脉冲输入计数器
22	DPAC 系统函数	
	<u>APS_rescan_CF</u>	重新扫描 DPAC 从站 CF 插槽
	<u>APS_get_battery_status</u>	获取 DPAC SRAM 电池状态
	<u>APS_get_display_data</u>	获取 7 段显示数据
	<u>APS_set_display_data</u>	设置 7 段显示数据
	<u>APS_get_button_status</u>	获取按钮输入状态
錯 誤! 找不 到參 照來 源。	非易失性 RAM 功能	
	<u>APS_set_nv_ram</u>	设置 RAM 数据
	<u>APS_get_nv_ram</u>	获取 RAM 数据
	<u>APS_clear_nv_ram</u>	清除 RAM 数据
錯 誤! 找不 到參 照來 源。	表定义	
	板卡参数表	
	现场总线参数表	
	中断信号表	
	设备信息表	
	DPAC 显示索引表	
	DPAC 按钮状态表	
	APS 函数返回码	

PCI-8392(H) 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_board_param</u>	设置板卡的参数
	<u>APS_get_board_param</u>	获取板卡的参数
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_get_system_timer</u>	获取系统定时器计数器
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_save_parameter_to_flash</u>	将系统和轴参数保存到闪存中
	<u>APS_load_parameter_from_flash</u>	从闪存中加载系统和轴参数
	<u>APS_load_parameter_from_default</u>	以默认值加载系统和轴参数
	<u>APS_load_param_from_file</u>	从文件中加载参数
SSCNET 函数		
錯誤! 找不到參照來源。	<u>APS_start_sscnet</u>	开启 SSCNET 网络
	<u>APS_stop_sscnet</u>	停止 SSCNET 网络
	<u>APS_get_sscnet_servo_param</u>	读取当前伺服参数值
	<u>APS_set_sscnet_servo_param</u>	设置伺服参数
	<u>APS_get_sscnet_servo_alarm</u>	获取当前伺服的报警信息
	<u>APS_reset_sscnet_servo_alarm</u>	伺服报警复位
	<u>APS_save_sscnet_servo_param</u>	将伺服参数保存到闪存中
	<u>APS_get_sscnet_servo_abs_position</u>	从伺服驱动器获取绝对参考位置
	<u>APS_save_sscnet_servo_abs_position</u>	将绝对参考位置保存到闪存中
	<u>APS_load_sscnet_servo_abs_position</u>	从闪存中加载绝对参考位置
	<u>APS_get_sscnet_link_status</u>	获取 SSCNET 链接状态
	<u>APS_set_sscnet_servo_monitor_src</u>	设置伺服监控数据源
	<u>APS_get_sscnet_servo_monitor_src</u>	获取伺服监控数据源
	<u>APS_get_sscnet_servo_monitor_data</u>	获取伺服监控数据
运动 IO 和运动状态		
錯誤! 找不到參照來	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动 IO 状态
	<u>APS_set_servo_on</u>	设置伺服开/关

源。	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_feedback_velocity</u>	获取反馈速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
錯誤! 找不到參照來源。	单轴运动	
	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_relative_move2</u>	根据速度曲线开始一个相对距离的运动
	<u>APS_absolute_move2</u>	根据速度曲线开始一个绝对位置的运动
	<u>APS_home_move2</u>	根据速度曲线开始返回原点
錯誤! 找不到參照來源。	点动	
	<u>APS_set_jog_param</u>	设置点动参数
	<u>APS_get_jog_param</u>	获取点动参数
	<u>APS_jog_mode_switch</u>	启用/禁用点动
	<u>APS_jog_start</u>	开始/停止点动
錯誤! 找不到參照來源。	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
錯誤! 找不到參照來源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
錯誤!	采样	
	<u>APS_set_sampling_param</u>	设置采样参数

找不到參照來源。	<u>APS_get_sampling_param</u>	获取采样参数。
	<u>APS_wait_trigger_sampling</u>	等待采样数据。
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数。
	<u>APS_stop_wait_sampling</u>	强制停止等待采样
錯誤! 找不到參照來源。	点表运动	
	<u>APS_set_point_table</u>	设置点表运动参数
	<u>APS_get_point_table</u>	获取点表运动参数
	<u>APS_set_point_table_ex</u>	设置带扩展选项的点表运动参数
	<u>APS_get_point_table_ex</u>	获取带扩展选项的点表运动参数
	<u>APS_point_table_move</u>	开始一个点表运动
	<u>APS_get_running_point_index</u>	当轴执行一个点表运动时，获取当前的点索引
	<u>APS_get_start_point_index</u>	当轴执行一个点表运动时，获取第一个点索引
	<u>APS_get_end_point_index</u>	当轴执行一个点表运动时，获取点索引的结尾。
	<u>APS_set_table_move_pause</u>	暂停点表运动
錯誤! 找不到參照來源。	现场总线函数	
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_set_field_bus_slave_param</u>	设置现场总线从站模块的参数
	<u>APS_get_field_bus_slave_param</u>	获取现场总线从站模块的参数
	<u>APS_set_field_bus_a_output</u>	设置现场总线模拟输出
	<u>APS_get_field_bus_a_output</u>	获取现场总线模拟输出
	<u>APS_get_field_bus_a_input</u>	获取现场总线模拟输入
	<u>APS_get_slave_connect_quality</u>	获取从站的连接质量
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称

	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
錯 誤! 找不 到參 照來 源。	<u>齿轮/龙门函数</u>	
	<u>APS_set_gantry_param</u>	设置龙门函数相关参数
	<u>APS_get_gantry_param</u>	获取龙门函数相关参数
	<u>APS_set_gantry_axis</u>	在龙门组中设置两个轴
	<u>APS_get_gantry_axis</u>	获取龙门组中的轴
	<u>APS_get_gantry_error</u>	获取龙门轴偏差误差
	<u>表定义</u>	
	<u>板卡参数表</u>	
	<u>轴参数表</u>	
	<u>采样参数表</u>	
	<u>采样源表</u>	
	<u>运动 IO 状态和运动状态定义</u>	
	<u>运动状态定义表</u>	
	<u>中断信号表</u>	
	<u>现场总线参数表</u>	
	<u>龙门参数表</u>	
	<u>设备信息表</u>	
	<u>现场总线从站参数表</u>	
	<u>SSCNET 伺服监控源表</u>	
	<u>APS 函数返回码</u>	

PCI-8253/56 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	APS_initial	设备初始化
	APS_close	设备关闭
	APS_version	获取库的版本
	APS_device_driver_version	获取驱动程序的设备版本
	APS_get_axis_info	获取指定轴的信息
	APS_get_card_name	获取板卡的名称
	APS_set_board_param	获取板卡的参数
	APS_get_board_param	设置轴参数
	APS_set_axis_param	获取轴参数
	APS_get_axis_param	获取系统计时器计数器
	APS_get_system_timer	获取设备信息
	APS_get_device_info	将系统和轴参数保存到闪存中
	APS_save_parameter_to_flash	从闪存中加载系统和轴参数
	APS_load_parameter_from_flash	以默认值加载系统和轴参数
	APS_load_parameter_from_default	从文件中加载参数
	APS_load_param_from_file	获取板卡的参数
运动 IO 和运动状态		
錯誤! 找不到參照來源。	APS_motion_status	返回运动状态
	APS_motion_io_status	返回运动 IO 状态
	APS_set_servo_on	设置伺服开/关
	APS_get_position	获取反馈位置
	APS_set_position	设置反馈位置
	APS_get_command	获取命令位置
	APS_set_command	设置命令位置
	APS_get_command_velocity	获取命令速度
	APS_get_feedback_velocity	获取反馈速度
	APS_get_error_position	获取错误位置
	APS_get_target_position	获取目标位置
单轴运动		
錯誤! 找不到參照來源。	APS_relative_move	开始一个相对距离的运动
	APS_absolute_move	开始一个绝对位置的运动
	APS_velocity_move	开始一个速度运动
	APS_home_move	开始一个返回原点的运动
	APS_stop_move	停止运动

	<u>APS_emg_stop</u>	紧急停止
	<u>APS_relative_move2</u>	根据速度曲线开始一个相对距离的运动
	<u>APS_absolute_move2</u>	根据速度曲线开始一个绝对位置的运动
	<u>APS_home_move2</u>	根据速度曲线开始返回原点
錯誤! 找不到參照來源。		点动
<u>APS_set_jog_param</u>		设置点动参数
<u>APS_get_jog_param</u>		获取点动参数
<u>APS_jog_mode_switch</u>		启用/禁用点动
<u>APS_jog_start</u>		开始/停止点动
錯誤! 找不到參照來源。		插补
<u>APS_absolute_linear_move</u>		开始一个绝对位置的线性插补
<u>APS_relative_linear_move</u>		开始一个相对距离的线性插补
<u>APS_absolute_arc_move</u>		开始一个绝对位置的圆弧插补
<u>APS_relative_arc_move</u>		开始一个相对距离的圆弧插补
<u>APS_absolute_arc_move_3pe</u>		利用通过和终点法开始一个绝对位置的圆弧插补
<u>APS_relative_arc_move_3pe</u>		利用通过和终点法开始一个相对距离的圆弧插补
<u>APS_absolute_helix_move</u>		开始一个绝对位置的螺旋插补
<u>APS_relative_helix_move</u>		开始一个相对距离的螺旋插补
錯誤! 找不到參照來源。		中断
<u>APS_int_enable</u>		中断主开关
<u>APS_set_int_factor</u>		启用/禁用中断信号并获取中断句柄
<u>APS_get_int_factor</u>		获取中断信号启用或禁用
<u>APS_wait_single_int</u>		等待单个中断事件
<u>APS_wait_multiple_int</u>		等待多个中断事件
<u>APS_reset_int</u>		将中断事件重置为非信号状态。
<u>APS_set_int</u>		将中断事件设置为信号状态。
<u>APS_set_int_factorH</u>		启用/禁用中断信号并获取中断句柄(Win32)
錯誤! 找不到參照來源。		采样
<u>APS_set_sampling_param</u>		设置采样参数
<u>APS_get_sampling_param</u>		获取采样参数
<u>APS_wait_trigger_sampling</u>		等待采样数据
<u>APS_wait_trigger_sampling_async</u>		等待异步样本数据
<u>APS_get_sampling_count</u>		获取采样数据计数.
<u>APS_stop_wait_sampling</u>		强制停止等待采样
DIO & AIO		
13	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值

錯誤! 找不到參照來源。	<u>APS_read_d_input</u>	读取数字输入值
	<u>APS_read_a_input_value</u>	回读模拟输入值(伏特)
	<u>APS_read_a_input_data</u>	回读模拟输入原始数据
	<u>APS_write_a_output_value</u>	设置模拟输出值(伏特)
	<u>APS_write_a_output_data</u>	将原始数据设置为模拟输出值
	点表运动	
	<u>APS_set_point_table</u>	设置点表运动参数
	<u>APS_get_point_table</u>	获取点表运动参数
	<u>APS_point_table_move</u>	开始一个点表运动
	<u>APS_get_running_point_index</u>	当轴执行一个点表运动时，获取当前的点索引
錯誤! 找不到參照來源。	<u>APS_get_start_point_index</u>	当轴执行一个点表运动时，获取第一个点索引
	<u>APS_get_end_point_index</u>	当轴执行一个点表运动时，获取点索引的结尾。
	<u>APS_set_table_move_pause</u>	暂停点表运动
	<u>APS_set_table_move_ex_pause</u>	减速以停止运动并控制 I/O
	<u>APS_set_table_move_ex_rollback</u>	回滚到当前点索引的起始位置
	<u>APS_set_table_move_ex_resume</u>	重新开始点表运动并保持 I/O 状态
	<u>APS_set_table_move_repeat</u>	设置点表运动重复
	齿轮/龙门函数	
	<u>APS_set_gantry_param</u>	设置龙门函数相关参数
	<u>APS_get_gantry_param</u>	获取龙门函数相关参数
錯誤! 找不到參照來源。	<u>APS_set_gantry_axis</u>	在龙门组中设置两个轴
	<u>APS_get_gantry_axis</u>	获取龙门组中的轴
	<u>APS_get_gantry_error</u>	获取龙门轴偏差误差
	<u>APS_get_encoder</u>	获取编码器(用于龙门架归零补偿)
	<u>APS_get_latch_event</u>	获取轴的锁存事件(用于龙门架归零补偿)
	<u>APS_get_latch_counter</u>	获取轴的锁存计数(用于龙门架归零补偿)
	比较触发	
	<u>APS_set_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_trigger_linear</u>	设置线性比较函数
錯誤! 找不到參照來源。	<u>APS_set_trigger_table</u>	设置表比较函数
	<u>APS_set_trigger_manual</u>	手动输出触发
	<u>APS_set_trigger_manual_s</u>	手动同步输出触发
	<u>APS_get_trigger_table_cmp</u>	获取当前的表比较值
	<u>APS_get_trigger_linear_cmp</u>	获取当前的线性比较值
	<u>APS_get_trigger_count</u>	获取触发计数

	<u>APS_reset_trigger_count</u>	重置触发计数
錯 誤! 找不 到參 照來 源。	<u>手动脉冲发生器输入</u>	
	<u>APS_get_pulser_counter</u>	获取脉冲输入计数器
<u>VAO/PWM 函数(激光函数)</u>		
26	<u>APS_set_vao_param</u>	将参数设置到 VAO 参数表中
	<u>APS_get_vao_param</u>	获取 VAO 参数表中的参数
	<u>APS_set_vao_table</u>	设置 VAO 表
	<u>APS_switch_vao_table</u>	切换到指定的 VAO 表
	<u>APS_start_vao</u>	启用 VAO 输出通道
	<u>APS_get_vao_status</u>	获取 VAO 状态
	<u>APS_check_vao_param</u>	检查指定 VAO 表的参数设置
	<u>APS_set_vao_param_ex</u>	通过 VAO 结构设置表参数。
	<u>APS_get_vao_param_ex</u>	通过 VAO 结构获取表参数。
	<u>APS_set_pwm_on</u>	开始输出 PWM 信号
	<u>APS_set_pwm_width</u>	将脉冲宽度设置至一个 PWM 通道
	<u>APS_set_pwm_frequency</u>	将脉冲频率设置至一个 PWM 通道
	<u>APS_get_pwm_width</u>	从一个 PWM 通道获取脉冲宽度
	<u>APS_get_pwm_frequency</u>	从一个 PWM 通道获取脉冲频率
<u>表定义</u>		
錯 誤! 找不 到參 照來 源。	<u>板卡参数表</u>	
	<u>轴参数表</u>	
	<u>采样参数表</u>	
	<u>采样源表</u>	
	<u>运动IO状态和运动状态定义</u>	
	<u>运动状态定义表</u>	
	<u>中断信号表</u>	
	<u>龙门参数表</u>	
	<u>触发参数表</u>	
	<u>设备信息表</u>	
	<u>VAO 参数表</u>	
	<u>APS函数返回码</u>	

PCI-8144 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_set_security_key</u>	设置安全密码
	<u>APS_check_security_key</u>	修改安全密码
	<u>APS_reset_security_key</u>	重置安全密码
	<u>APS_load_param_from_file</u>	从文件中加载参数
运动 IO 和运动状态		
錯誤! 找不到參照來源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
单轴运动		
錯誤! 找不到參照來源。	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
中断		
錯誤! 找不到參照來源。	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄 (Win32)

DIO & AIO	
13	<u>APS_write_d_output</u> 设置数字输出值
	<u>APS_read_d_output</u> 读取数字输出值
	<u>APS_read_d_input</u> 读取数字输入值
錯 誤! 找不 到參 照來 源。	非易失性 RAM 功能
	<u>APS_set_nv_ram</u> 设置RAM数据
	<u>APS_get_nv_ram</u> 获取RAM数据
	<u>APS_clear_nv_ram</u> 清除RAM数据
錯 誤! 找不 到參 照來 源。	表定义
	<u>轴参数表</u>
	<u>运动IO状态和运动状态定义</u>
	<u>运动状态定义表</u>
	<u>中断信号表</u>
	<u>设备信息表</u>
	<u>APS函数返回码</u>

PCI(e)-7856 函数列表

章节	函数名称	描述
錯誤! 找不到參照來源。	系统和初始化	
	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	设置板卡的参数
	<u>APS_set_board_param</u>	获取板卡的参数
	<u>APS_get_board_param</u>	获取设备信息
	<u>APS_get_device_info</u>	将参数保存到文件中
錯誤! 找不到參照來源。	<u>APS_save_param_to_file</u>	从文件中加载参数
	<u>APS_load_param_from_file</u>	获取板卡的名称
	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
錯誤! 找不到參照來源。	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄 (Win32)
	现场总线函数	
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
錯誤! 找不到參照來源。	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_set_field_bus_slave_param</u>	设置现场总线从站模块的参数
	<u>APS_get_field_bus_slave_param</u>	获取现场总线从站模块的参数

	<u>APS_set_field_bus_a_output</u>	设置现场总线模拟输出
	<u>APS_get_field_bus_a_output</u>	获取现场总线模拟输出
	<u>APS_get_field_bus_a_input</u>	获取现场总线模拟输入
	<u>APS_get_slave_connect_quality</u>	获取从站的连接质量
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
	<u>APS_field_bus_d_set_output_ex</u>	为 64 位 DIO 操作设置现场总线数字输出
	<u>APS_field_bus_d_get_output_ex</u>	为 64 位 DIO 操作获取现场总线数字输出
	<u>APS_field_bus_d_get_input_ex</u>	为 64 位 DIO 操作获取现场总线数字输入
錯誤! 找不到參照來源。	非易失性 RAM 功能	
	<u>APS_set_nv_ram</u>	设置RAM数据
	<u>APS_get_nv_ram</u>	获取RAM数据
	<u>APS_clear_nv_ram</u>	清除RAM数据
錯誤! 找不到參照來源。	表定义	
	<u>板卡参数表</u>	
	<u>现场总线参数表</u>	
	<u>中断信号表</u>	
	<u>设备信息表</u>	
	<u>APS函数返回码</u>	

MNET-4XMO 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_save_param_to_file</u>	将参数保存到文件中
	<u>APS_load_param_from_file</u>	从文件中加载参数
运动 IO 和运动状态		
錯誤! 找不到參照來源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置
	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_command_velocity_f</u>	获取64位的命令速度
	<u>APS_get_error_position_f</u>	获取64位的错误位置
	<u>APS_get_target_position_f</u>	获取 64 位的目标位置
单轴运动		
錯誤! 找不到參照來源。	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_home_escape</u>	离开原点开关
插补		
錯誤! 找不到參照來源。	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补

源。	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
錯 誤! 找不 到參 照來 源。	<u>中断</u>	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_reset_field_bus_int_motion</u>	重置MotionNet系列的中断轴状态
	<u>APS_set_field_bus_int_factor_motion</u>	启用/禁用运动中断信号并为MotionNet系列获取中断句柄
	<u>APS_get_field_bus_int_factor_motion</u>	为MotionNet系列获取运动中断信号启用或禁用。
	<u>APS_set_field_bus_int_factor_error</u>	为MotionNet系列启用/禁用错误中断信号并获取中断句柄。
	<u>APS_get_field_bus_int_factor_error</u>	为MotionNet系列获取错误的中断信号状态。
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
	<u>APS_wait_field_bus_error_int_motion</u>	等待MotionNet系列的错误中断事件。
錯 誤! 找不 到參 照來 源。	<u>采样</u>	
	<u>APS_set_sampling_param</u>	设置采样参数
	<u>APS_get_sampling_param</u>	获取采样参数
	<u>APS_wait_trigger_sampling</u>	等待采样数据
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数.
	<u>APS_stop_wait_sampling</u>	强制停止等待采样
錯 誤! 找不 到參 照來 源。	<u>现场总线函数</u>	
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型

	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
同动函数		
錯誤! 找不到參照來源。	<u>APS_set_relative_simultaneous_move</u>	设置一个相对的同动
	<u>APS_set_absolute_simultaneous_move</u>	设置一个绝对的同动
	<u>APS_start_simultaneous_move</u>	开始一个同动
	<u>APS_stop_simultaneous_move</u>	停止同动
单锁存(Single-latch)函数		
錯誤! 找不到參照來源。	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据
表定义		
錯誤! 找不到參照來源。	轴参数表	
	运动IO状态和运动状态定义	
	运动状态定义表	
	中断信号表	
	现场总线参数表	
	设备信息表	
	APS函数返回码	

MNET-4XMO-C 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_save_param_to_file</u>	将参数保存到文件中
	<u>APS_load_param_from_file</u>	从文件中加载参数
运动 IO 和运动状态		
錯誤! 找不到參照來源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置
	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_command_velocity_f</u>	获取64位的命令速度
	<u>APS_get_error_position_f</u>	获取64位的错误位置
单轴运动		
錯誤! 找不到參照來源。	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_home_escape</u>	离开原点开关
插补		
錯誤! 找不到參照來源。	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补

錯 誤! 找不 到參 照來 源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_reset_field_bus_int_motion</u>	重置MotionNet系列的中断轴状态
	<u>APS_set_field_bus_int_factor_motion</u>	启用/禁用运动中断信号并为MotionNet系列获取中断句柄
	<u>APS_get_field_bus_int_factor_motion</u>	为MotionNet系列获取运动中断信号启用或禁用。
	<u>APS_set_field_bus_int_factor_error</u>	为MotionNet系列启用/禁用错误中断信号并获取中断句柄。
	<u>APS_get_field_bus_int_factor_error</u>	为MotionNet系列获取错误的中断信号状态。
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
錯 誤! 找不 到參 照來 源。	采样	
	<u>APS_set_sampling_param</u>	设置采样参数
	<u>APS_get_sampling_param</u>	获取采样参数
	<u>APS_wait_trigger_sampling</u>	等待采样数据
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数.
錯 誤! 找不 到參 照來 源。	点表运动	
	<u>APS_set_point_table_mode2</u>	设置点表模式
	<u>APS_set_point_table2</u>	设置点表
	<u>APS_point_table_continuous_move2</u>	开始一个点表的连续运动
	<u>APS_point_table_single_move2</u>	开始一个点表的单步运动
	<u>APS_get_running_point_index2</u>	当轴执行一个点表运动时，获取当前的点索引
錯 誤! 找不 到參 照來 源。	现场总线函数	
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络

	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
现场总线比较触发		
錯誤! 找不到參照來源。	<u>APS_set_field_bus_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_field_bus_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_field_bus_trigger_linear</u>	设置线性比较函数
	<u>APS_set_field_bus_trigger_table</u>	设置表比较函数
	<u>APS_set_field_bus_trigger_manual</u>	手动输出触发
	<u>APS_set_field_bus_trigger_manual_s</u>	手动同步输出触发
	<u>APS_get_field_bus_trigger_table_cmp</u>	获取当前的表比较值
	<u>APS_get_field_bus_trigger_linear_cmp</u>	获取当前的线性比较值
	<u>APS_get_field_bus_trigger_count</u>	获取触发计数
	<u>APS_reset_field_bus_trigger_count</u>	重置触发计数
	<u>APS_get_field_bus_linear_cmp_remain_count</u>	获取线性比较的剩余计数器
	<u>APS_get_field_bus_table_cmp_remain_count</u>	获取表比较的剩余计数器
	<u>APS_get_field_bus_encoder</u>	获取编码器计数器
	<u>APS_set_field_bus_encoder</u>	设置编码器计数器
同动函数		
錯誤! 找不到參照來源。	<u>APS_set_relative_simultaneous_move</u>	设置一个相对的同动
	<u>APS_set_absolute_simultaneous_move</u>	设置一个绝对的同动
	<u>APS_start_simultaneous_move</u>	开始同动
	<u>APS_stop_simultaneous_move</u>	停止同动
单锁存(Single-latch)函数		
錯誤! 找不到參照來源。	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据
錯	表定义	

誤!
找不
到參
照來
源。

轴参数表
运动IO状态和运动状态定义
运动状态定义表
中断信号表
现场总线参数表
触发参数表
设备信息表
APS函数返回码

MNET-1XMO 函数列表

章节	函数名称	描述
系统和初始化		
APS_get_axis_info		获取指定轴的信息
APS_set_axis_param		设置轴参数
APS_get_axis_param		获取轴参数
APS_save_param_to_file		将参数保存到文件中
APS_load_param_from_file		从文件中加载参数
运动 IO 和运动状态		
APS_motion_status		返回运动状态
APS_motion_io_status		返回运动IO状态
APS_set_servo_on		设置伺服开/关
APS_get_position		获取反馈位置
APS_set_position		设置反馈位置
APS_get_command		获取命令位置
APS_set_command		设置命令位置
APS_get_command_velocity		获取命令速度
APS_get_error_position		获取错误位置
APS_get_target_position		获取目标位置
APS_get_position_f		获取64位的反馈位置
APS_set_position_f		设置64位的反馈位置
APS_get_command_f		获取64位的命令位置
APS_set_command_f		设置64位的命令位置
APS_get_command_velocity_f		获取64位的命令速度
APS_get_error_position_f		获取64位的错误位置
APS_get_target_position_f		获取64位的目标位置
单轴运动		
APS_relative_move		开始一个相对距离的运动
APS_absolute_move		开始一个绝对位置的运动
APS_velocity_move		开始一个速度运动
APS_home_move		开始一个返回原点的运动
APS_stop_move		停止运动
APS_emg_stop		紧急停止
APS_speed_override		快速改变速度
APS_relative_move_ovrd		开始一个相对距离的运动或以新的距离和速度覆盖它
APS_absolute_move_ovrd		开始一个绝对位置的运动或以新的位置和速度覆盖它

	<u>APS_home_escape</u>	离开原点开关
錯誤! 找不到參照來源。		中断
	<u>APS_int_enable</u>	中断主开关
	<u>APS_reset_field_bus_int_motion</u>	重置MotionNet系列的中断轴状态
	<u>APS_set_field_bus_int_factor_motion</u>	启用/禁用运动中断信号并为MotionNet系列获取中断句柄
	<u>APS_get_field_bus_int_factor_motion</u>	为MotionNet系列获取运动中断信号启用或禁用。
	<u>APS_set_field_bus_int_factor_error</u>	为MotionNet系列启用/禁用错误中断信号并获取中断句柄。
	<u>APS_get_field_bus_int_factor_error</u>	为MotionNet系列获取错误的中断信号状态。
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
	<u>APS_wait_field_bus_error_int_motion</u>	等待MotionNet系列的错误中断事件。
錯誤! 找不到參照來源。		现场总线函数
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
錯誤! 找不到參照來源。		单锁存(Single-latch)函数
	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据
錯	表定义	

誤!
找不
到參
照來
源。

轴参数表

运动IO状态和运动状态定义

运动状态定义表

中断信号表

现场总线参数表

APS函数返回码

HSL-4XMO 函数列表

章节	函数名称	描述
錯 誤! 找不 到參 照來 源。	系统和初始化	
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_load_param_from_file</u>	从文件中加载参数
錯 誤! 找不 到參 照來 源。	运动 IO 和运动状态	
	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
錯 誤! 找不 到參 照來 源。	单轴运动	
	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
錯 誤! 找不 到參 照來 源。	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
錯 誤! 找不 到參 照來 源。	点表运动	
	<u>APS_set_point_table3</u>	设置点表
	<u>APS_point_table_move3</u>	开始一个点表的单步运动
錯 誤! 找不 到參 照來 源。	<u>APS_set_point_table_param3</u>	设置速度参数
	现场总线函数	
	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数

照來 源。	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
	<u>APS_get_field_bus_slave_first_axisno</u>	获取从站模块的第一个轴
	<u>APS_get_field_bus_device_info</u>	获取指定现场总线上的设备信息
现场总线比较触发		
錯 誤! 找不 到參 照來 源。	<u>APS_set_field_bus_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_field_bus_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_field_bus_trigger_linear</u>	设置线性比较函数
	<u>APS_set_field_bus_trigger_table</u>	设置表比较函数
	<u>APS_get_field_bus_trigger_table_cmp</u>	获取当前的表比较值
	<u>APS_get_field_bus_trigger_linear_cmp</u>	获取当前的线性比较值
錯 誤! 找不 到參 照來 源。	表定义	
	轴参数表	
	<u>运动IO状态和运动状态定义</u>	
	<u>运动状态定义表</u>	
	<u>现场总线参数表</u>	
	<u>触发参数表</u>	
	<u>设备信息表</u>	
<u>APS函数返回码</u>		

HSL-DIO 函数列表

章节	函数名称	描述
中断		
錯 誤! 找不 到參 照來 源。	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_field_bus_int_factor_di</u>	分配数字输入中断位并获得HSL系列的中断句柄
	<u>APS_get_field_bus_int_factor_di</u>	获取分配的数字输入中断位
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
现场总线函数		
錯 誤! 找不 到參 照來 源。	<u>APS_set_field_bus_param</u>	设置现场总线相关参数
	<u>APS_get_field_bus_param</u>	获取现场总线相关参数
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_field_bus_d_set_output</u>	设置现场总线数字输出
	<u>APS_field_bus_d_get_output</u>	获取现场总线数字输出
	<u>APS_field_bus_d_get_input</u>	获取现场总线数字输入
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息
	<u>APS_get_field_bus_master_type</u>	获取现场总线上的主站类型
	<u>APS_get_field_bus_slave_type</u>	获取现场总线上的从站类型
	<u>APS_get_field_bus_slave_name</u>	获取现场总线上的从站名称
表定义		
錯 誤! 找不 到參 照來 源。	<u>现场总线参数表</u>	
	<u>APS函数返回码</u>	

PCI-8102/PCI-C154(+)函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本

	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_load_param_from_file</u>	从文件中加载参数
錯誤! 找不到參照來源。	运动 IO 和运动状态	
	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置
	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_command_velocity_f</u>	获取64位的命令速度
	<u>APS_get_error_position_f</u>	获取64位的错误位置
	<u>APS_get_target_position_f</u>	获取64位的目标位置
錯誤! 找不到參照來源。	单轴运动	
	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_speed_override</u>	离开原点开关
	<u>APS_relative_move_ovrd</u>	快速改变速度 (仅PCI-C154+)
	<u>APS_absolute_move_ovrd</u>	开始一个相对距离的运动或以新的距离和速度覆盖它 (仅PCI-C154+)
錯	<u>APS_home_escape</u>	开始一个绝对位置的运动或以新的位置和速度覆盖它 (仅PCI-C154+)
	插补	

誤! 找不到參照來源。	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
錯 誤! 找不到參照來源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_wait_error_int</u>	等待错误中断(非屏蔽)
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄(Win32)
13	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。(Win32)
	DIO & AIO	
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
錯 誤! 找不到參照來源。	比较触发	
	<u>APS_set_trigger_param</u>	设置比较触发参数
	<u>APS_get_trigger_param</u>	获取比较触发参数
	<u>APS_reset_trigger_count</u>	重置触发计数
	<u>APS_get_trigger_count</u>	获取触发计数
	<u>APS_enable_trigger_fifo_cmp</u>	启用触发FIFO比较器
	<u>APS_get_trigger_fifo_cmp</u>	获取触发FIFO比较器
	<u>APS_get_trigger_fifo_status</u>	获取触发FIFO状态
	<u>APS_set_trigger_fifo_data</u>	设置触发FIFO数据
	<u>APS_set_trigger_linear</u>	设置触发线性比较
	<u>APS_get_trigger_linear_cmp</u>	获取触发线性比较
	<u>APS_set_trigger_manual</u>	手动设置触发
	<u>APS_set_trigger_manual_s</u>	手动同步输出触发
	<u>APS_start_timer</u>	启动计时器(模拟编码器)
	<u>APS_get_timer_counter</u>	获取计时器计数值(模拟编码器)
	<u>APS_set_timer_counter</u>	设置计时器计数值(模拟编码器)
	<u>APS_start_trigger_timer</u>	启动计时器(生成触发信号)
	<u>APS_get_trigger_timer_counter</u>	获取计时器计数值(生成触发信号)

錯 誤! 找不 到參 照來 源。	手动脉冲发生器输入	
	<u>APS_manual_pulser_start</u>	启用/禁用手动脉冲发生器输入
	<u>APS_manual_pulser_velocity_move</u>	开始一个脉冲速度运动
	<u>APS_manual_pulser_relative_move</u>	开始一个脉冲相对距离运动
錯 誤! 找不 到參 照來 源。	同动函数(仅 PCI-C154+)	
	<u>APS_set_relative_simultaneous_move</u>	设置一个相对的同动
	<u>APS_set_absolute_simultaneous_move</u>	设置一个绝对的同动
	<u>APS_start_simultaneous_move</u>	开始一个同动
錯 誤! 找不 到參 照來 源。	单锁存(Single-latch)函数	
	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据
	多锁存(Multi-latch)函数	
錯 誤! 找不 到參 照來 源。	<u>APS_set_ltc_counter</u>	设置锁存计数器
	<u>APS_get_ltc_counter</u>	获取轴的锁存数据
	<u>APS_set_ltc_fifo_param</u>	设置锁存参数
	<u>APS_get_ltc_fifo_param</u>	获取锁存参数
	<u>APS_manual_latch</u>	手动和同步锁存数据
	<u>APS_enable_ltc_fifo</u>	启用/禁用锁存fifo
	<u>APS_reset_ltc_fifo</u>	重置锁存fifo
	<u>APS_get_ltc_fifo_data</u>	从fifo中获取一个锁存数据
	<u>APS_get_ltc_fifo_usage</u>	获取锁存fifo的使用情况
	<u>APS_get_ltc_fifo_free_space</u>	获取锁存fifo的可用空间
錯 誤! 找不 到參 照來 源。	<u>APS_get_ltc_fifo_status</u>	获取fifo状态
	速度曲线计算	
	<u>APS_relative_move_profile</u>	获取相对速度曲线
	<u>APS_absolute_move_profile</u>	获取绝对速度曲线
錯 誤! 找不 到參 照來 源。	表定义	
	<u>轴参数表</u>	
	<u>运动IO状态和运动状态定义</u>	
	<u>运动状态定义表</u>	
	<u>中断信号表</u>	
	<u>设备信息表</u>	

	<u>APS函数返回码</u>
	<u>触发参数表</u>
	<u>轴参数表</u>

PCI-8154/8158 函数列表

章节	函数名称	描述
系统和初始化		
錯誤! 找不到參照來源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_set_security_key</u>	设置安全密码
	<u>APS_check_security_key</u>	修改安全密码
	<u>APS_reset_security_key</u>	重置安全密码
	<u>APS_load_param_from_file</u>	从文件中加载参数
运动 IO 和运动状态		
錯誤! 找不到參照來源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置
	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_command_velocity_f</u>	获取64位的命令速度
	<u>APS_get_error_position_f</u>	获取64位的错误位置
	<u>APS_get_target_position_f</u>	获取64位的目标位置
单轴运动		
錯誤! 找不到	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动

到參照來源。	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_speed_override</u>	快速改变速度
	<u>APS_relative_move_ovrd</u>	开始一个相对距离的运动或以新的距离和速度覆盖它
	<u>APS_absolute_move_ovrd</u>	开始一个绝对位置的运动或以新的位置和速度覆盖它
	<u>APS_home_escape</u>	离开原点开关
錯誤! 找不到參照來源。	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
	<u>APS_absolute_helical_move</u>	开始一个绝对位置的螺旋插补
錯誤! 找不到參照來源。	高级单步运动和插补	
	<u>APS_ptp_v</u>	根据运动曲线开始一个单步运动
	<u>APS_ptp_all</u>	根据所有曲线开始一个单步运动
	<u>APS_vel</u>	开始一个速度运动
	<u>APS_vel_all</u>	根据所有曲线开始一个速度运动
	<u>APS_line</u>	开始一个线性运动
	<u>APS_line_v</u>	根据运动曲线开始一个线性运动
	<u>APS_line_all</u>	根据所有曲线开始一个线性运动
	<u>APS_arc2_ca</u>	开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据运动曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个角度类型的 2D 运动
	<u>APS_spiral_ce</u>	开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 螺旋运动
錯誤! 找不到參照來源。	<u>APS_spiral_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 螺旋运动
	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件

	<u>APS_wait_error_int</u>	等待错误中断(非屏蔽)
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄 (Win32)
	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
13	DIO & AIO	
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
	<u>APS_read_d_input</u>	读取数字输入值
錯誤! 找不到參照來源。	比較觸發(僅 DB-8150)	
	<u>APS_set_trigger_param</u>	設置比較觸發參數
	<u>APS_reset_trigger_count</u>	重置觸發計數
	<u>APS_get_trigger_count</u>	獲取觸發計數
	<u>APS_enable_trigger_fifo_cmp</u>	啟用觸發FIFO比較器
	<u>APS_get_trigger_fifo_cmp</u>	獲取觸發FIFO比較器
	<u>APS_get_trigger_fifo_status</u>	獲取觸發FIFO狀態
	<u>APS_set_trigger_fifo_data</u>	設置觸發FIFO數據
	<u>APS_get_trigger_param</u>	獲取比較觸發參數
	<u>APS_set_trigger_linear</u>	設置觸發線性比較
	<u>APS_get_trigger_linear_cmp</u>	獲取觸發線性比較
	<u>APS_set_trigger_manual</u>	手動設置觸發
	<u>APS_start_timer</u>	啟動計時器(模擬編碼器)
	同動函數	
錯誤! 找不到參照來源。	<u>APS_set_relative_simultaneous_move</u>	設置一個相對的同動
	<u>APS_set_absolute_simultaneous_move</u>	設置一個絕對的同動
	<u>APS_start_simultaneous_move</u>	開始一個同動
	<u>APS_stop_simultaneous_move</u>	停止同動
錯誤! 找不到參照來源。	單鎖存(Single-latch)函數	
	<u>APS_manual_latch2</u>	手動鎖存
	<u>APS_get_latch_data2</u>	獲取軸的鎖存數據
錯誤! 找不到	環形計數器函數	
	<u>APS_set_ring_counter</u>	啟用環形計數器功能
	<u>APS_get_ring_counter</u>	獲取環形計數器的極限值

參 照 來 源。		
錯 誤! 找不 到參 照來 源。	<u>表定义</u>	
	<u>轴参数表</u>	
	<u>运动IO状态和运动状态定义</u>	
	<u>运动状态定义表</u>	
	<u>中断信号表</u>	
	<u>设备信息表</u>	
	<u>APS函数返回码</u>	
	<u>触发参数表</u>	

PCIe-8154/8158 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_set_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_device_info</u>	获取设备信息
	<u>APS_load_param_from_file</u>	从文件中加载参数
运动 IO 和运动状态		
錯 誤! 找不 到參 照來 源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置

	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置
	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_command_velocity_f</u>	获取64位的命令速度
	<u>APS_get_error_position_f</u>	获取64位的错误位置
	<u>APS_get_target_position_f</u>	获取64位的目标位置
錯 誤! 找 不 到參 照來 源。	单轴运动	
	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
	<u>APS_speed_override</u>	快速改变速度
	<u>APS_relative_move_ovrd</u>	开始一个相对距离的运动或以新的距离和速度覆盖它
	<u>APS_absolute_move_ovrd</u>	开始一个绝对位置的运动或以新的位置和速度覆盖它
錯 誤! 找 不 到參 照來 源。	<u>APS_home_escape</u>	离开原点开关
	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
	<u>APS_absolute_helical_move</u>	开始一个绝对位置的螺旋插补
錯 誤! 找 不 到參 照來 源。	<u>APS_relative_helical_move</u>	开始一个相对位置的螺旋插补
	高级单步运动和插补	
	<u>APS_ptp_v</u>	根据运动曲线开始一个单步运动
	<u>APS_ptp_all</u>	根据所有曲线开始一个单步运动
	<u>APS_vel</u>	开始一个速度运动
	<u>APS_vel_all</u>	根据所有曲线开始一个速度运动
	<u>APS_line</u>	开始一个线性运动
	<u>APS_line_v</u>	根据运动曲线开始一个线性运动
	<u>APS_line_all</u>	根据所有曲线开始一个线性运动
	<u>APS_arc2_ca</u>	开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据运动曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个角度类型的 2D 运动
	<u>APS_spiral_ce</u>	开始一个终点位置类型的 3D 螺旋运动

	<u>APS_spiral_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 螺旋运动
錯 誤! 找不 到參 照來 源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_wait_error_int</u>	等待错误中断 (非屏蔽)
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄 (Win32)
13	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
	DIO & AIO	
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
	<u>APS_read_d_input</u>	读取数字输入值
	<u>APS_write_d_channel_output</u>	按通道设置数字输出值
	<u>APS_read_d_channel_output</u>	按通道读取数字输出值
錯 誤! 找不 到參 照來 源。	手动脉冲发生器输入	
	<u>APS_manual_pulser_start</u>	启用/禁用手动脉冲发生器输入
	<u>APS_manual_pulser_velocity_move</u>	开始一个脉冲速度运动
	<u>APS_manual_pulser_relative_move</u>	开始一个脉冲相对距离运动
	<u>APS_manual_pulser_home_move</u>	开始一个脉冲归零运动
錯 誤! 找不 到參 照來 源。	同动函数	
	<u>APS_set_relative_simultaneous_move</u>	设置一个相对的同动
	<u>APS_set_absolute_simultaneous_move</u>	设置一个绝对的同动
	<u>APS_start_simultaneous_move</u>	开始一个同动
	<u>APS_stop_simultaneous_move</u>	停止同动
錯 誤! 找不 到參 照來 源。	单锁存(Single-latch)函数	
	<u>APS_manual_latch2</u>	手动锁存
	<u>APS_get_latch_data2</u>	获取轴的锁存数据

源。	
	<u>表定义</u>
	<u>轴参数表</u>
	<u>运动IO状态和运动状态定义</u>
	<u>运动状态定义表</u>
	<u>中断信号表</u>
	<u>设备信息表</u>
	<u>APS函数返回码</u>
	<u>触发参数表</u>

EMX-100 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_set_board_param</u>	设置板卡参数
	<u>APS_get_board_param</u>	获取板卡参数
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_get_device_info</u>	获取轴参数
	<u>APS_get_first_axisId</u>	获取卡的第一个轴 ID
	<u>APS_load_parameter_from_default</u>	按默认值加载系统和轴参数
	<u>APS_load_param_from_file</u>	从文件中加载参数
	<u>APS_register_emx</u>	在库中注册 EMX
	<u>APS_get_deviceIP</u>	获取设备 IP
	<u>APS_reset_emx_alarm</u>	重置设备的报警信号
运动 IO 和运动状态		
錯 誤! 找不 到參 照來 源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_feedback_velocity</u>	获取反馈速度
	单轴运动	
錯 誤! 找不 到參 照來 源。	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止

錯 誤! 找不 到參 照來 源。	点动	
	<u>APS_jog_start</u>	开启/停止点动
錯 誤! 找不 到參 照來 源。	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
13	DIO & AIO	
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字输出值
	<u>APS_read_d_input</u>	读取数字输入值
	<u>APS_write_d_channel_output</u>	按通道设置数字输出值
錯 誤! 找不 到參 照來 源。	比较触发	
	<u>APS_set_trigger_param</u>	设置比较触发参数
	<u>APS_get_trigger_param</u>	获取比较触发参数
	<u>APS_get_trigger_count</u>	获取触发计数
	<u>APS_reset_trigger_count</u>	重置触发计数
錯 誤! 找不 到參 照來 源。	速度曲线计算	
	<u>APS_check_motion_profile_emx</u>	获取相对速度曲线
錯 誤! 找不 到參 照來 源。	表定义	
	板卡参数表	
	轴参数表	
	运动IO状态和运动状态定义	
	运动状态定义表	
	触发参数表	
	设备信息表	
<u>APS函数返回码</u>		

PCI-8254/58 / AMP-204/8C 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_disable_device</u>	禁用卡
	<u>APS_set_board_param</u>	设置板卡参数
	<u>APS_get_board_param</u>	获取板卡参数
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_set_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_system_timer</u>	获取系统计时器计数器
	<u>APS_get_device_info</u>	获取轴参数
	<u>APS_get_first_axisId</u>	获取卡的第一个轴 ID
	<u>APS_save_parameter_to_flash</u>	将系统和轴参数保存到闪存中
	<u>APS_load_parameter_from_flash</u>	从闪存中加载系统和轴参数
	<u>APS_load_parameter_from_default</u>	以默认值加载系统和轴参数
	<u>APS_load_param_from_file</u>	从文件中加载参数
	<u>APS_get_curr_sys_ctrl_mode</u>	获取当前系统的控制模式
运动 IO 和运动状态		
錯 誤! 找不 到參 照來 源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动IO状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position</u>	获取反馈位置
	<u>APS_set_position</u>	设置反馈位置
	<u>APS_get_command</u>	获取命令位置
	<u>APS_set_command</u>	设置命令位置
	<u>APS_get_command_velocity</u>	获取命令速度
	<u>APS_get_feedback_velocity</u>	获取反馈速度
	<u>APS_get_error_position</u>	获取错误位置
	<u>APS_get_target_position</u>	获取目标位置
	<u>APS_get_position_f</u>	获取64位的反馈位置
	<u>APS_set_position_f</u>	设置64位的反馈位置

	<u>APS_get_command_f</u>	获取64位的命令位置
	<u>APS_set_command_f</u>	设置64位的命令位置
	<u>APS_get_error_position_f</u>	获取64位的命令速度
	<u>APS_get_target_position_f</u>	获取64位的错误位置
	<u>APS_get_command_velocity_f</u>	获取 64 位的命令速度
	<u>APS_get_feedback_velocity_f</u>	获取 64 位的反馈速度
	<u>APS_get_mq_free_space</u>	获取运动队列的可用空间
	<u>APS_get_mq_usage</u>	获取运动队列的使用情况
	<u>APS_get_stop_code</u>	获取停止代码
	<u>APS_get_encoder</u>	获取原始的反馈计数器
	<u>APS_get_command_counter</u>	获取原始的命令计数器
	<u>APS_get_axis_latch_data</u>	获取 ORG/EZ 锁存器数据
錯 誤! 找不 到參 照來 源。	单轴运动	
	<u>APS_relative_move</u>	开始一个相对距离的运动
	<u>APS_absolute_move</u>	开始一个绝对位置的运动
	<u>APS_velocity_move</u>	开始一个速度运动
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
錯 誤! 找不 到參 照來 源。	多轴运动触发和停止	
	<u>APS_move_trigger</u>	发送一个触发以同步所有等待的运动
	<u>APS_stop_move_multi</u>	停止多轴运动
	<u>APS_emg_stop_multi</u>	立即停止多轴运动
錯 誤! 找不 到參 照來 源。	点动	
	<u>APS_jog_start</u>	开始/停止点动
錯 誤! 找不 到參 照來 源。	插补	
	<u>APS_absolute_linear_move</u>	开始一个绝对位置的线性插补
	<u>APS_relative_linear_move</u>	开始一个相对距离的线性插补
	<u>APS_absolute_arc_move</u>	开始一个绝对位置的圆弧插补
	<u>APS_relative_arc_move</u>	开始一个相对距离的圆弧插补
錯 誤! 找不 到參 照來 源。	高级单步运动和插补	
	<u>APS_ptp</u>	开始一个单步运动
	<u>APS_ptp_v</u>	根据运动曲线开始一个单步运动
	<u>APS_ptp_all</u>	根据所有曲线开始一个单步运动
	<u>APS_vel</u>	开始一个速度运动

	<u>APS_vel_all</u>	根据所有曲线开始一个速度运动
	<u>APS_line</u>	开始一个线性运动
	<u>APS_line_v</u>	根据运动曲线开始一个线性运动
	<u>APS_line_all</u>	根据所有曲线开始一个线性运动
	<u>APS_arc2_ca</u>	开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据运动曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ce</u>	开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_v</u>	根据速度曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_all</u>	根据所有曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc3_ca</u>	开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_v</u>	根据速度曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_all</u>	根据所有曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ce</u>	开始一个终点位置类型的 3D 运动
	<u>APS_arc3_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 运动
	<u>APS_arc3_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 运动
	<u>APS_spiral_ca</u>	开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_v</u>	根据速度曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_all</u>	根据所有曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ce</u>	开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 螺旋运动
錯 誤! 找不 到參 照來 源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	等待错误中断 (非屏蔽)
	<u>APS_set_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int_factorH</u>	将中断事件设置为信号状态。

	<u>APS_int_no_to_handle</u>	启用/禁用中断信号并获取中断句柄 (Win32)
錯誤! 找不到參照來源。	<u>APS_set_sampling_param</u>	设置采样参数
	<u>APS_get_sampling_param</u>	获取采样参数
	<u>APS_wait_trigger_sampling</u>	等待采样数据
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数
	<u>APS_stop_wait_sampling</u>	强制停止等待采样
	<u>APS_auto_sampling</u>	开始/停止自动采样
	<u>APS_get_sampling_data</u>	通过 4 个通道以自动采样模式获取采样数据
	<u>APS_set_sampling_param_ex</u>	按照结构设置采样参数。通过一个扩展至 8 个通道
	<u>APS_get_sampling_param_ex</u>	按照结构获取采样参数。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_ex</u>	等待采样数据。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_async_ex</u>	等待异步采样数据。通过一个扩展至 8 个通道
	<u>APS_get_sampling_data_ex</u>	在自动采样模式下获取采样数据。通过一个扩展至 8 个通道
13	DIO & AIO	
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字量输出值
	<u>APS_read_d_input</u>	读取数字量输入值
	<u>APS_write_d_channel_output</u>	设置每通道的数字输出值
	<u>APS_read_d_channel_output</u>	读取每通道的数字输出值
	<u>APS_read_a_input_value</u> (*1)	回读模拟输入值 (伏特)
錯誤! 找不到參照來源。	<u>APS_write_a_output_value</u> (*1)	设置模拟输出值 (伏特)
	点表运动	
	<u>APS_set_feeder_group</u>	将轴设置给馈线(feeder)组
	<u>APS_get_feeder_group</u>	在一个馈线(feeder)组中返回配置
	<u>APS_free_feeder_group</u>	释放馈线(feeder)组及其资源
	<u>APS_reset_feeder_buffer</u>	重置馈线(feeder)的点缓冲区
	<u>APS_set_feeder_point_2D</u>	在馈线(Feeder)的缓冲区中添加一个点
	<u>APS_set_feeder_point_2D_ex</u>	在馈线(Feeder)的缓冲区中添加一个点
	<u>APS_start_feeder_move</u>	开始点表运动和馈点 (feed point)
	<u>APS_get_feeder_status</u>	获取馈线(Feeder)状态

	<u>APS_set_feeder_ex_pause</u>	运动已暂停 (已停止) 和馈线(feeder)已暂停
	<u>APS_set_feeder_ex_rollback</u>	返回到暂停索引的起始位置
	<u>APS_set_feeder_ex_resume</u>	恢复点表运动。
高级点表		
	<u>APS_pt_enable</u>	启用点表
	<u>APS_pt_disable</u>	禁用点表
	<u>APS_get_pt_info</u>	获取点表信息
	<u>APS_pt_set_vs</u>	将速度配置设置到点表中
	<u>APS_pt_get_vs</u>	获取点表中的速度配置
	<u>APS_pt_start</u>	将控制命令设置到点表中
	<u>APS_pt_stop</u>	停止点表
	<u>APS_get_pt_status</u>	获取点表的状态
	<u>APS_reset_pt_buffer</u>	重置点表缓冲区
	<u>APS_pt_roll_back</u>	回滚到上一点
	<u>APS_get_pt_error</u>	获取点表的错误代码
	<u>APS_pt_dwell</u>	将一个停顿运动推送到点表的点缓冲区。
	<u>APS_pt_line</u>	将一个线性运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ca</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ce</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ca</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ce</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ca</u>	将一个螺旋运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ce</u>	将一个螺旋运动推送到点表的点缓冲区。
	<u>APS_pt_ext_set_do_ch</u>	将 Do 扩展命令设置到命令缓冲区。当将一个运动推送到点表中时，命令缓冲区处于活动状态。
	<u>APS_pt_set_absolute</u>	将绝对曲线设置到曲线缓冲区
	<u>APS_pt_set_relative</u>	将相对曲线设置到曲线缓冲区
	<u>APS_pt_set_trans_buffered</u>	在曲线缓冲区中将转换设置为缓冲模式。
	<u>APS_pt_set_trans_inp</u>	在曲线缓冲区中将转换设置为就位模式。
	<u>APS_pt_set_trans_blend_dec</u>	在曲线缓冲区中将转换设置为带减速的混合模式。
	<u>APS_pt_set_trans_blend_dist</u>	在曲线缓冲区中将转换设置为带剩余距离的混合模式。

錯
誤!
找不
到參
照來
源。

	<u>APS_pt_set_trans_blend_pcnt</u>	在曲线缓冲区中将转换设置为带剩余距离百分比的混合模式。
	<u>APS_pt_set_acc</u>	将加速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_dec</u>	将减速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_acc_dec</u>	将加速/减速曲线设置到曲线缓冲区中
	<u>APS_pt_set_s</u>	将 S-factor 曲线设置到曲线缓冲区中
	<u>APS_pt_set_vm</u>	将最大速度曲线设置到曲线缓冲区中
	<u>APS_pt_set_ve</u>	将终点速度曲线设置到曲线缓冲区中
錯誤! 找不到參照來源。	齿轮/龙门函数	
	<u>APS_start_gear</u>	启用/禁用一个指定的齿轮模式
錯誤! 找不到參照來源。	比较触发	
	<u>APS_set_trigger_param</u>	设置比较触发相关参数
	<u>APS_get_trigger_param</u>	获取比较触发相关参数
	<u>APS_set_trigger_linear</u>	设置线性比较函数
	<u>APS_set_trigger_table</u>	设置表比较函数
	<u>APS_set_trigger_manual</u>	手动输出触发
	<u>APS_set_trigger_manual_s</u>	手动同步输出触发
	<u>APS_get_trigger_table_cmp</u>	获取当前的表比较值
	<u>APS_get_trigger_linear_cmp</u>	获取当前的线性比较值
	<u>APS_get_trigger_count</u>	获取触发计数
	<u>APS_reset_trigger_count</u>	重置触发计数
	<u>APS_get_timer_counter</u>	获取计时器计数值
	<u>APS_set_timer_counter</u>	设置计时器计数值
	<u>APS_set_multi_trigger_table</u>	设置表比较函数
	<u>APS_get_multi_trigger_table_cmp</u>	获取当前表比较值
	<u>APS_set_trigger_table_data</u>	设置表比较器数据 (快速比较表触发函数)
	<u>APS_get_trigger_table_status</u>	获取表比较器状态 (快速比较表触发函数)
	<u>APS_get_trigger_cmp_value</u>	获取表比较器值 (快速比较表触发函数)
	<u>APS_enable_trigger_table</u>	启用表比较器 (快速比较表触发函数)
	<u>APS_reset_trigger_table</u>	重置表比较器 (快速比较表触发函数)
錯誤! 找不到參照來源。	程序下载(*1)	
	<u>APS_load_vmc_program</u>	将 VMC 文件加载到任务存储器
	<u>APS_save_vmc_program</u>	从任务存储器中保存至 VMC 文件
	<u>APS_set_task_mode</u>	设置任务运行模式
	<u>APS_get_task_mode</u>	获取任务运行模式
	<u>APS_start_task</u>	启动任务控制命令

	<u>APS_get_task_info</u>	获取任务信息
	<u>APS_get_task_msg</u>	获取所有任务的信息
錯誤！找不到參照來源。	手动脉冲发生器输入	
	<u>APS_manual_pulser_start</u>	开始手动脉冲发生器操作
	<u>APS_manual_pulser_velocity_move</u>	在手动脉冲发生器操作中开始速度运动
錯誤！找不到參照來源。	螺距误插补偿功能	
	<u>APS_set_pitch_table</u>	设置螺距误插补偿表的配置和数据
	<u>APS_get_pitch_table</u>	获取螺距误插补偿表的配置和数据
	<u>APS_start_pitch_comp</u>	开始螺距误插补偿
錯誤！找不到參照來源。	看门狗定时器	
	<u>APS_wdt_start</u>	开启/停至看门狗定时器
	<u>APS_wdt_get_timeout_period</u>	获取看门狗定时器的超时时间
	<u>APS_wdt_reset_counter</u>	重置看门狗计时器的计数器
	<u>APS_wdt_get_counter</u>	获取看门狗计时器的计数器
	<u>APS_wdt_set_action_event</u>	设置看门狗定时器的动作事件
	<u>APS_wdt_get_action_event</u>	获取看门狗定时器的动作事件
26	VAO/PWM 函数(激光函数)	
	<u>APS_set_vao_param</u>	将参数设置到 VAO 参数表中
	<u>APS_get_vao_param</u>	获取 VAO 参数表中的参数
	<u>APS_set_vao_table</u>	设置 VAO 表
	<u>APS_switch_vao_table</u>	切换到指定的 VAO 表
	<u>APS_start_vao</u>	启用 VAO 输出通道
	<u>APS_get_vao_status</u>	获取 VAO 状态
	<u>APS_check_vao_param</u>	检查指定 VAO 表的参数设置
	<u>APS_set_vao_param_ex</u>	通过 VAO 结构设置表参数。
	<u>APS_get_vao_param_ex</u>	通过 VAO 结构获取表参数。
	<u>APS_set_pwm_on</u>	开始输出 PWM 信号
	<u>APS_set_pwm_width</u>	将脉冲宽度设置至一个 PWM 通道
	<u>APS_set_pwm_frequency</u>	将脉冲频率设置至一个 PWM 通道
	<u>APS_get_pwm_width</u>	从一个 PWM 通道获取脉冲宽度
	<u>APS_get_pwm_frequency</u>	从一个 PWM 通道获取脉冲频率
錯	循环限制函数	

誤! 找不到參照來源。	<u>APS_set_circular_limit</u>	设置环形限位配置
	<u>APS_get_circular_limit</u>	获取环形限位配置
錯誤! 找不到參照來源。	多锁存(Multi-latch)函数	
	<u>APS_enable_ltc_fifo</u>	启用位置锁存进程
	<u>APS_get_ltc_fifo_point</u>	获取锁存点数组
	<u>APS_set_ltc_fifo_param</u>	设置锁存参数
	<u>APS_get_ltc_fifo_param</u>	获取锁存参数
	<u>APS_reset_ltc_fifo</u>	重置锁存队列和 fifo
	<u>APS_get_ltc_fifo_usage</u>	获取锁存队列已用空间
	<u>APS_get_ltc_fifo_free_space</u>	获取锁存队列可用空间
	<u>APS_get_ltc_fifo_status</u>	获取锁存队列 fifo 状态
錯誤! 找不到參照來源。	背隙函数	
	<u>APS_set_backlash_en</u>	启用/停止背隙
	<u>APS_get_backlash_en</u>	检查背隙是否启用/停止
錯誤! 找不到參照來源。	表定义	
	板卡参数表	
	轴参数表	
	采样参数表	
	采样源表	
	运动IO状态和运动状态定义	
	运动状态定义表	
	中断信号表	
	触发参数表	
	<u>锁存参数表</u>	
	设备信息表	
	VAO参数表	
	APS函数返回码	

*1. AMP 系列不支持该特点。

PCIe-833x 函数列表

章节	函数名称	描述
系统和初始化		
錯 誤! 找不 到參 照來 源。	<u>APS_initial</u>	设备初始化
	<u>APS_close</u>	设备关闭
	<u>APS_version</u>	获取库的版本
	<u>APS_device_driver_version</u>	获取驱动程序的设备版本
	<u>APS_get_axis_info</u>	获取指定轴的信息
	<u>APS_get_card_name</u>	获取板卡的名称
	<u>APS_disable_device</u>	禁用卡
	<u>APS_set_board_param</u>	设置板卡参数
	<u>APS_get_board_param</u>	获取板卡参数
	<u>APS_set_axis_param</u>	设置轴参数
	<u>APS_get_axis_param</u>	获取轴参数
	<u>APS_set_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_axis_param_f</u>	设置 64 位的轴参数
	<u>APS_get_system_timer</u>	获取系统计时器计数器
	<u>APS_get_device_info</u>	获取轴参数
	<u>APS_save_parameter_to_flash</u>	将系统和轴参数保存到闪存中
	<u>APS_load_parameter_from_flash</u>	从闪存中加载系统和轴参数
	<u>APS_load_parameter_from_default</u>	按默认值加载系统和轴参数.
运动 IO 和运动状态		
錯 誤! 找不 到參 照來 源。	<u>APS_motion_status</u>	返回运动状态
	<u>APS_motion_io_status</u>	返回运动 IO 状态
	<u>APS_set_servo_on</u>	设置伺服开/关
	<u>APS_get_position_f</u>	获取 64 位的反馈位置
	<u>APS_set_position_f</u>	设置 64 位的反馈位置
	<u>APS_get_command_f</u>	获取 64 位的命令位置
	<u>APS_set_command_f</u>	设置 64 位的命令位置
	<u>APS_get_error_position_f</u>	获取 64 位的命令速度
	<u>APS_get_target_position_f</u>	获取 64 位的错误位置
	<u>APS_get_command_velocity_f</u>	获取 64 位的命令速度
	<u>APS_get_feedback_velocity_f</u>	获取 64 位的反馈速度
	<u>APS_get_mq_free_space</u>	获取运动队列的可用空间
	<u>APS_get_mq_usage</u>	获取运动队列的使用情况
	<u>APS_get_stop_code</u>	获取停止代码
	<u>APS_get_encoder</u>	获取原始的反馈计数器
	<u>APS_get_command_counter</u>	获取原始的命令计数器

	<u>APS_reset_command_counter</u>	重置原始的命令计数器
	<u>APS_get_actual_torque</u>	获取实际扭矩值
錯 誤! 找不 到參 照來 源。	单轴运动	
	<u>APS_home_move</u>	开始一个返回原点的运动
	<u>APS_stop_move</u>	停止运动
	<u>APS_emg_stop</u>	紧急停止
錯 誤! 找不 到參 照來 源。	多轴运动触发和停止	
	<u>APS_move_trigger</u>	发送一个触发以同步所有等待的运动
	<u>APS_stop_move_multi</u>	停止多轴运动
	<u>APS_emg_stop_multi</u>	立即停止多轴运动
錯 誤! 找不 到參 照來 源。	点动	
	<u>APS_jog_start</u>	开始/停止点动
	高级单步运动和插补	
	<u>APS_ptp</u>	开始一个单步运动
錯 誤! 找不 到參 照來 源。	<u>APS_ptp_v</u>	根据运动曲线开始一个单步运动
	<u>APS_ptp_all</u>	根据所有曲线开始一个单步运动
	<u>APS_vel</u>	开始一个速度运动
	<u>APS_vel_all</u>	根据所有曲线开始一个速度运动
	<u>APS_line</u>	开始一个线性运动
	<u>APS_line_v</u>	根据运动曲线开始一个线性运动
	<u>APS_line_all</u>	根据所有曲线开始一个线性运动
	<u>APS_arc2_ca</u>	开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_v</u>	根据运动曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ca_all</u>	根据所有曲线开始一个角度类型的 2D 运动
	<u>APS_arc2_ce</u>	开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_v</u>	根据速度曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc2_ce_all</u>	根据所有曲线开始一个终点位置类型的 2D 运动
	<u>APS_arc3_ca</u>	开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_v</u>	根据速度曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ca_all</u>	根据所有曲线开始一个角度类型的 3D 运动
	<u>APS_arc3_ce</u>	开始一个终点位置类型的 3D 运动
	<u>APS_arc3_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 运动

	<u>APS_arc3_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 运动
	<u>APS_spiral_ca</u>	开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_v</u>	根据速度曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ca_all</u>	根据所有曲线开始一个角度类型的 3D 螺旋运动
	<u>APS_spiral_ce</u>	开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_v</u>	根据速度曲线开始一个终点位置类型的 3D 螺旋运动
	<u>APS_spiral_ce_all</u>	根据所有曲线开始一个终点位置类型的 3D 螺旋运动
錯誤! 找不到參照來源。	中断	
	<u>APS_int_enable</u>	中断主开关
	<u>APS_set_int_factor</u>	启用/禁用中断信号并获取中断句柄
	<u>APS_get_int_factor</u>	获取中断信号启用或禁用
	<u>APS_wait_single_int</u>	等待单个中断事件
	<u>APS_wait_multiple_int</u>	等待多个中断事件
	<u>APS_reset_int</u>	将中断事件重置为非信号状态。
	<u>APS_set_int</u>	将中断事件设置为信号状态。
	<u>APS_set_int_factorH</u>	启用/禁用中断信号并获取中断句柄。 (Win32)
錯誤! 找不到參照來源。	<u>APS_int_no_to_handle</u>	将中断事件数字编号转换为中断句柄。 (Win32)
	采样	
	<u>APS_set_sampling_param</u>	设置采样参数。
	<u>APS_get_sampling_param</u>	获取采样参数。
	<u>APS_wait_trigger_sampling</u>	等待采样数据。
	<u>APS_wait_trigger_sampling_async</u>	等待异步样本数据
	<u>APS_get_sampling_count</u>	获取采样数据计数。
	<u>APS_stop_wait_sampling</u>	强制停止等待采样
	<u>APS_auto_sampling</u>	开始/停止自动采样
	<u>APS_get_sampling_data</u>	通过 4 个通道以自动采样模式获取采样数据
	<u>APS_set_sampling_param_ex</u>	按照结构设置采样参数。通过一个扩展至 8 个通道
	<u>APS_get_sampling_param_ex</u>	按照结构获取采样参数。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_ex</u>	等待采样数据。通过一个扩展至 8 个通道
	<u>APS_wait_trigger_sampling_async_ex</u>	等待异步采样数据。通过一个扩展至 8 个通道

		道
	<u>APS_get_sampling_data_ex</u>	在自动采样模式下获取采样数据。通过一个扩展至 8 个通道
DIO & AIO		
13	<u>APS_set_field_bus_d_channel_output</u>	按通道设置现场总线数字输出
	<u>APS_get_field_bus_d_channel_output</u>	按通道获取现场总线数字输出
	<u>APS_get_field_bus_d_channel_input</u>	按通道获取现场总线数字输入
	<u>APS_set_field_bus_d_port_output</u>	按端口设置现场总线数字输出
	<u>APS_get_field_bus_d_port_input</u>	按端口设置现场总线数字输入
	<u>APS_get_field_bus_d_port_output</u>	按端口设置现场总线数字输出
	<u>APS_write_d_output</u>	设置数字输出值
	<u>APS_read_d_output</u>	读取数字量输出值
	<u>APS_read_d_input</u>	读取数字输入值
	<u>APS_write_d_channel_output</u>	设置每通道的数字输出值
	<u>APS_read_d_channel_output</u>	读取每通道的数字输出值
	<u>APS_read_d_channel_input</u>	读取每通道的数字输入值
	高级点表	
錯 誤! 找不 到參 照來 源。	<u>APS_pt_enable</u>	启用点表
	<u>APS_pt_disable</u>	禁用点表
	<u>APS_get_pt_info</u>	获取点表信息
	<u>APS_pt_set_vs</u>	将速度配置设置到点表中
	<u>APS_pt_get_vs</u>	获取点表中的速度配置
	<u>APS_pt_start</u>	将控制命令设置到点表中
	<u>APS_pt_stop</u>	停止点表
	<u>APS_get_pt_status</u>	获取点表的状态
	<u>APS_reset_pt_buffer</u>	重置点表缓冲区
	<u>APS_pt_roll_back</u>	回滚到上一点
	<u>APS_pt_dwell</u>	将一个停顿运动推送到点表的点缓冲区。
	<u>APS_pt_line</u>	将一个线性运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ca</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc2_ce</u>	将一个 2D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ca</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_arc3_ce</u>	将一个 3D 圆弧运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ca</u>	将一个螺旋运动推送到点表的点缓冲区。
	<u>APS_pt_spiral_ce</u>	将一个螺旋运动推送到点表的点缓冲区。

	<u>APS_pt_ext_set_do_ch</u>	在 EPS-6000 从站的高级点位表中控制数字输出
	<u>APS_pt_set_absolute</u>	将绝对曲线设置到曲线缓冲区
	<u>APS_pt_set_relative</u>	将相对曲线设置到曲线缓冲区
	<u>APS_pt_set_trans_buffered</u>	在曲线缓冲区中将转换设置为缓冲模式。
	<u>APS_pt_set_trans_inp</u>	在曲线缓冲区中将转换设置为就位模式。
	<u>APS_pt_set_trans_blend_dec</u>	在曲线缓冲区中将转换设置为带减速的混合模式。
	<u>APS_pt_set_trans_blend_dist</u>	在曲线缓冲区中将转换设置为带剩余距离的混合模式。
	<u>APS_pt_set_trans_blend_pcnt</u>	在曲线缓冲区中将转换设置为带剩余距离百分比的混合模式。
	<u>APS_pt_set_acc</u>	将加速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_dec</u>	将减速曲线设置到曲线缓冲区中。
	<u>APS_pt_set_acc_dec</u>	将加速/减速曲线设置到曲线缓冲区中
	<u>APS_pt_set_s</u>	将 S-factor 曲线设置到曲线缓冲区中
	<u>APS_pt_set_vm</u>	将最大速度曲线设置到曲线缓冲区中
	<u>APS_pt_set_ve</u>	将终点速度曲线设置到曲线缓冲区中
錯 誤! 找 不 到參 照來 源。	现场总线函数	
	<u>APS_scan_field_bus</u>	扫描现场总线并生成 ENI 文件
	<u>APS_start_field_bus</u>	启动指定现场总线的网络
	<u>APS_stop_field_bus</u>	停止指定现场总线的网络
	<u>APS_set_field_bus_a_output</u>	设置现场总线数字输出
	<u>APS_get_field_bus_a_input</u>	获取现场总线数字输入
	<u>APS_get_field_bus_master_status</u>	获取现场总线主站状态
	<u>APS_get_slave_online_status</u>	获取从站的在线状态
	<u>APS_get_field_bus_last_scan_info</u>	系统扫描后，获取现场总线信息。
	<u>APS_get_field_bus_module_info</u>	获取从站信息
	<u>APS_reset_field_bus_alarm</u>	重置从站的报警信号
	<u>APS_get_field_bus_alarm</u>	获取从站的报警代码
	<u>APS_get_field_bus_pdo</u>	从 PDO 内存中获取值
	<u>APS_set_field_bus_pdo</u>	为 PDO 内存赋值
	<u>APS_get_field_bus_pdo_offset</u>	获取 PDO 信息
	<u>APS_get_field_bus_sdo</u>	从从站中获取 SDO 数据
	<u>APS_set_field_bus_sdo</u>	为从站设置 SDO 数据
	<u>APS_set_field_bus_od_data</u>	设置 EtherCAT OD 原始数据
	<u>APS_get_field_bus_od_data</u>	获取 EtherCAT OD 原始数据
	<u>APS_get_field_bus_od_module_info</u>	获取 EtherCAT 从站信息
	<u>APS_get_field_bus_module_map</u>	在手动 ID 模式下获取映射的从站 ID

	<u>APS_set_field_bus_module_map</u>	在手动 ID 模式下设置映射的从站 ID
	<u>APS_get_field_bus_slave_state</u>	获取从站状态机器的状态
	<u>APS_set_field_bus_slave_state</u>	设置从站状态机器的状态
	<u>APS_get_field_bus_ESC_register</u>	获取 EtherCAT 从站控制器寄存器
	<u>APS_set_field_bus_ESC_register</u>	设置 EtherCAT 从站控制器寄存器
	<u>APS_get_system_loading</u>	获取系统循环加载
	<u>APS_get_field_bus_analysis_topology</u>	获取当前和过去的拓扑结构，然后进行分析
	<u>APS_get_field_bus_loss_package</u>	在接收总线方向上获取 EtherCAT 帧计数的丢失信息。
錯 誤! 找 不 到參 照來 源。	齿轮/龙门函数	
	<u>APS_start_gear</u>	启用/禁用一个指定的齿轮模式
	<u>APS_get_gear_status</u>	获取齿轮状态
	<u>APS_get_gantry_number</u>	获取主站对应的从站数量
	<u>APS_get_gantry_info</u>	获取从轴 ID 数组
	<u>APS_get_gantry_deviation</u>	获取主从站之间的位置偏差
錯 誤! 找 不 到參 照來 源。	手动脉冲发生器输入	
	<u>APS_manual_pulser_start</u>	开始手动脉冲发生器操作
	<u>APS_manual_pulser_velocity_move</u>	在手动脉冲发生器操作中开始速度运动
	看门狗定时器	
	<u>APS_wdt_start</u>	开启/停至看门狗定时器
	<u>APS_wdt_get_timeout_period</u>	获取看门狗定时器的超时时间
錯 誤! 找 不 到參 照來 源。	<u>APS_wdt_reset_counter</u>	重置看门狗计时器的计数器
	<u>APS_wdt_get_counter</u>	获取看门狗计时器的计数器
	<u>APS_wdt_set_action_event</u>	设置看门狗定时器的动作事件
	<u>APS_wdt_get_action_event</u>	获取看门狗定时器的动作事件
	循环限制函数	
	<u>APS_set_circular_limit</u>	设置环形限位配置
錯 誤! 找 不 到參 照來 源。	<u>APS_get_circular_limit</u>	获取环形限位配置
	表定义	
	板卡参数表	
	轴参数表	
	采样参数表	
錯 誤! 找 不 到參 照來 源	采样源表	

照 來 源。	运动IO状态和运动状态定义
	运动状态定义表
	中断信号表
	设备信息表
	APS 函数返回码

03. 系统和初始化

APS_initial	设备初始化
-------------	-------

支持的产品:PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于初始化本地控制器中所有支持 APS 函数库的产品。它为每个板卡分配系统硬件资源，包括 I/O 地址，内存地址，IRQ 和 DMA（如果需要的话）。它检索每个由板载开关或操作系统分配的板卡 ID。板卡 ID 是板卡在系统中唯一的编号，任何其他 APS 函数可以通过这个 ID 号来访问相应的硬件。

如果用户选择板载开关（Mode = manual ID）的初始模式，并且系统中有些板卡不支持此功能，那么这些板卡的 ID 将在板载开关自动排列之后再进行排列。使用“manual-ID”时，无法将板卡 ID（DIP 开关，DIP Switch）设置为相同，否则函数将返回错误。

对于 EMX-100：

注意:

- (1) EMX 系列产品的电源打开或重新连接完成后，建议在 20 秒后执行 APS_initial()。
- (2) 在使用 APS_initial()之前，用户必须使用 APS_register_emx()在 APS 库中注册 EMX 系列产品。

句法:

C/C++:

```
I32 FNTYPE APS_initial(I32 *BoardID_InBits, I32 Mode);
```

Visual Basic:

```
APS_initial (BoardID_InBits As Long, ByVal Mode As Long) As Long
```

参数:

I32 * BoardID_InBits: Card ID information in bit format.

示例: 如果 BoardID_InBits 的值是 0x11，那就意味着您的系统中有两张板卡，并且板卡 ID 号是 0 和 4。

I32 Mode:

Bit 0	启用板载DIP开关 (SW1)来确定板卡 ID。 [0:系统分配, 1:DIP开关分配]
Bit 1	并行轴索引模式 0：自动模式（默认） 1：固定模式

Bit 2	串行轴索引模式 对于PCIe-833x: [备注 1] 0：自动模式（默认） 1：固定模式
Bit 4 Bit 5	加载系统和轴参数的方法选项。 对于 PCI-8253/6 和 PCI-8392(H) (00B) 0: 根据每个板卡参数中的引导模式设置进行加载。 (01B) 1: 为所有板卡加载默认值 (02B) 2: 为所有板卡从闪存中加载 对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x (00B) 0: 不执行任何操作，参数保留当前值。 (01B) 1: 加载默认值 (02B) 2: 从闪存中加载(*1)
Bit 6	选择系统模式的选项. (仅PCI(e)-7856) (0) – 轮询模式 (不支持运动中断) (1) – 中断模式 (支持运动中断)
Bit 9	用于选择运动状态MDN位行为的选项。 (仅PCI-8254/58/AMP-204/8C和PCIe-833x) 0 : 当CSTP或ASTP发生时 , MDN打开。 (默认) 1 : 当CSTP发生时 , MDN打开。
Bit 10	EtherCAT 从站ID编号设置。 (仅PCIe-833x) [备注1] 0 : 从站ID编号由系统自动分配。 (从零开始) 1 : 从站ID号编由用户设置手动分配。
其他	保留 (设置为0)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#)。

示例 :

```
I32 ret; // 返回值
I32 BoardID_InBits;
I32 Mode = 0; //由系统分配
对于 EMX-100 : ret = APS_register_emx( 1, 0); // 在 APS 库中注册 EMX 系列产品
ret = APS_initial( &BoardID_InBits, Mode);
```

```
...// 具体的工作
ret = APS_close(); //关闭系统中所有的板卡
```

还可以看看:

对于 EMX-100 : APS_register_emx()
APS_close();APS_get_axis_info()

仅适用于 PCIe-833X:

[备注 1]

手动设置从站 ID 编号的用法如下：

[第一步]

使用 MotionCreatorPro 2 (MCP2) 实用程序给每个在线的从站手动设置从站 ID 编号。之后，关闭 MCP2 并关闭所有在线的从站的电源。

[第二步]

打开所有在线的从站的电源。

根据[表 1]，用户可以通过 **APS_initial()** 中的参数 “**I32 Mode**” 选择从站 ID 和轴 ID 之间的关系映射。

[第三步]

执行 **APS_start_field_bus()** 开始通信。然后，从站 ID 和轴 ID 之间的关系映射将由[表 1]指定。现在，用户可以指定从站 ID 或轴 ID 来控制每个在线的从站。

[表 1]

	从站 ID	轴 ID
[选择 1] I32 Mode = 0x0 (bit 10 = 0, bit 2 = 0)	由系统分配 (从零开始) 示例： 从站 ID: 0,1,2,3...	由系统分配 (从零开始) 示例： 轴 ID: 0,1,2,3...
[选择 2] I32 Mode = 0x400 (bit 10 = 1, bit 2 = 0)	通过用户设置分配 (通过 MCP2 实用程序) 示例： 从站 ID: 100,200,300,400...	由 APS_start_field_bus() 中的参数 “I32 Starting_Axis_ID” 分配 示例： I32 Starting_Axis_ID = 1000; 轴 ID: 1000,1001,1002,1003...
[选择 3] I32 Mode = 0x404 (bit 10 = 1, bit 2 = 1)	通过用户设置分配 (通过 MCP2 实用程序) 示例： 从站 ID: 100,200,300,400...	请参阅从站 ID。 有关详细信息，请参见示例 [表 2]。

[表 2]

从站 ID (由用户设置分配)	轴数	轴 ID
100	1	100

200	2	200,201
300	3	300,301,302
400	1	400

APS_close	设备关闭
-----------	------

支持的产品:PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于关闭由 APS 库分配的所有资源。这些资源包括系统硬件资源，例如 I/O 地址，内存地址，IRQ 和 DMA。它还会删除由 APS 库分配的某些对象，句柄或内存。

句法:

C/C++:

```
I32 FNTYPE APS_close()
```

Visual Basic:

```
APS_close() As Long
```

参数:

无参数。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#)。

示例 :

```
I32 ret; // 返回值
I32 BoardID_InBits;
I32 Mode = 0; //由系统分配
对于 EMX-100 : ret = APS_register_emx( 1, 0); // 在 APS 库中注册 EMX 系列产品
ret = APS_initial( &BoardID_InBits, Mode);

...//具体的工作
ret = APS_close(); //关闭系统中所有的板卡
```

还可以看看:

[APS_initial\(\)](#)

APS_version	获取库的版本
支持的产品 : PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

该函数用于获取 APS 库 (DLL)的版本信息。

句法:

C/C++:

```
I32 FNTYPE APS_version();
```

Visual Basic:

```
APS_version() As Long
```

参数:

无参数

返回值:

返回库 (DLL)的版本。

示例 :

```
I32 version;  
version = APS_version();
```

还可以看看:

APS_device_driver_version	获取驱动程序的设备版本
支持的产品 : PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

该函数用于获取板卡设备的驱动程序版本信息。系统中同一类型板卡的版本信息相同。

句法:

C/C++

I32 FNTYPE APS_device_driver_version(I32 Board_ID)

Visual Basic:

APS_device_driver_version(ByVal Board_ID As Long) As Long

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

返回值:

正值: 设备驱动程序版本号。

负值: 错误代码: 请参考错误代码表。

示例 :

```
I32 version;
```

```
//获取板卡 0 的设备驱动程序版本
version = APS_device_driver_version( 0 );
```

还可以看看:

APS_get_axis_info	获取指定轴的信息
-------------------	----------

支持的产品: PCI-8253/56, PCI-8392 (H) , DPAC-3000 , PCI-8144, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于获取一个轴的信息。该信息包括附加板卡 ID , 串行端口 ID , 串行模块 ID 和模块类型。轴 ID 索引有两类 : 并行和串行。PCI-8392 SSCNET 3 是并行轴。MNET-4XMO- (C) 和 HSL-4XMO 是串行轴。

并行轴 ID 索引规则是根据板卡 ID 来确定的 , 计算公式如下 :

$$\text{轴 ID} = \text{板卡 ID} \times \text{一个板卡支持的最大轴数} + \text{轴数}$$

轴数这项参数是指板卡支持的轴数。一个板卡支持的最大轴数指的是 APS 库的内部系统变量。

如果 APS 系统运行在自动模式下 , 该值取决于板卡的类型。如果 APS 系统运行在固定模式下 , 该值默认为 32。如果系统中有些板卡没有轴 , 那么固定模式下检索时仍会使用计算公式。使用固定模式时 , 用户从系统中删除/添加板卡 , 可以无需重新安排轴索引。

例如, 一个用户有三片板卡: PCI-8392 和 PCI-8253 和 PCI-8258。

	PCI-8392 (ID=0), 8 轴	PCI-8253 (ID=1), 3 轴	PCI-8258 (ID=0), 8 轴
自动模式	轴 ID 范围 0~7	轴 ID 范围 8~10	轴 ID 范围 0~7
固定模式	轴 ID 范围 0~7	轴 ID 范围 32~34	轴 ID 范围 0~15

如果板卡 ID 不连续,

	PCI-8392 (ID=0), 8 轴	PCI-8253 (ID=2), 3 轴	PCI-8258 (ID=0), 8 轴
自动模式	轴 ID 范围 0~7	轴 ID 范围 8~10	轴 ID 范围 0~7
固定模式	轴 ID 范围 0~7	轴 ID 范围 64~66	轴 ID 范围 0~15

串行轴 ID 索引规则是根据模块 ID 来确定的 , 并首先会分配起始轴 ID。计算公式如下 :

$$\text{轴 ID} = \text{模块 ID} \times \text{一个模块支持的最大轴数} + \text{端口的起始轴 ID} + \text{轴数}$$

轴数这项参数是指模块支持的轴数。一个模块支持的最大轴数指的是 APS 库的内部系统变量。

如果 APS 系统运行在自动模式下 , 该值取决于模块的类型。如果 APS 系统运行在固定模式下 , 该值默认为 4。端口的起始轴 ID 是现场总线启动时用户分配的一个端口的起始轴 ID , 默认值为 0。在固定模式下 , 如果端口有一些没有轴的模块 , 那么检索时仍会使用计算公式。使用固定模式时 , 用户从系统中删除/添加模块 , 可以无需重新安排其他模块的轴索引。

例如, 一个用户的 PCI(e)-7856 上有两个 MNET 模块 , 且板卡 ID=0

	MNET-J3 (ID=0)	MNET-4XMO (ID=1), 4 轴
自动模式	轴 ID 范围仅为 0	轴 ID 范围 1~4
固定模式	轴 ID 范围 0~3	轴 ID 范围 4~7

如果模块 ID 不连续,

	MNET-J3 (ID=0)	MNET-4XMO (ID=2), 4 轴

自动模式	轴 ID 范围仅为 0	轴 ID 范围 1~4
固定模式	轴 ID 范围 0~3	轴 ID 范围 8~11

对于 EMX-100 和 PCIe-833x , 此函数用于获取指定轴的信息。该信息包括相应的板卡 ID , 板卡上的轴数 , 总线数和自动从站 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_axis_info( I32 Axis_ID, I32 *Board_ID, I32 *Axis_No, I32 *Port_ID,  
I32 *Module_ID );
```

Visual Basic:

```
APS_get_axis_info(ByVal Axis_ID As Long, Board_ID As Long, Axis_No As Long,  
Port_ID As Long, Module_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Board_ID: 返回板卡 ID 的轴 ID。范围是 0 到 31。

I32 *Axis_No: 板卡内的轴数。范围从 0 到此模块内的最大轴数。

I32 *Port_ID: 返回现场总线板卡的端口 ID , 范围是 0 到 15。

*Port_ID=-1 表示不存在串行端口。

对于 PCI(e)-7856, HSL 现场总线为端口 ID 0 , MNET 现场总线为端口 ID 1。
I32 *Module_ID: 返回端口的模块 ID。范围是 0~65535。*Module_ID=-1 表示不存在串行端口。

对于 HSL 类型的现场总线 , 模块数量范围是 1 到 63。注意 : 在 HSL 中, Module_ID 是模块占用的第一个 ID。

对于 MNET 现场总线 , 范围是 0~63。

对于 EMX-100 和 PCIe-833x :

I32 Axis_ID: 轴 ID。同时支持自动轴 ID 和手动轴 ID。

在手动轴 ID 中 , 输入范围是 1000 到 655357。

I32 *Board_ID: 返回相应的板卡 ID。

I32 *Axis_No: 返回板卡上相应的轴数。

I32 *Port_ID: 返回相应的总线数量。

I32 *Module_ID: 返回相应的自动从站 ID。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID = 0;
```

```
I32 Board_ID, Axis_No, Port_ID, Module_ID;  
  
//根据轴 ID , 获取相应的信息。  
APS_get_axis_info( Axis_ID, & Board_ID, &Axis_No, &Port_ID, &Module_ID );
```

还可以看看:

```
APS_start_field_bus();APS_initial()
```

APS_get_card_name	获取板卡的名称
支持的产品:PCI-8253/56, PCI-8392 (H) , DPAC-1000,DPAC-3000 , PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取板卡的名称。在执行APS_initial()之后，用户可以通过传递指定的板卡ID来获得每个板卡的名称。

句法:

C/C++:

```
I32 FNTYPE APS_get_card_name ( I32 Board_ID, I32 *CardName );
```

Visual Basic:

```
APS_get_card_name (ByVal Board_ID As Long, CardName As Long) As Long
```

参数:

I32 Board_ID: 板卡ID从 0 到 31。

I32 * CardName: 板卡 ID 所代表的板卡名称。

0: PCI_8392,	1: PCI_825x,	2: PCI_8154,	3: PCI_785X
4: PCI_8158,	5: PCI_7856,	6: ISA_DPAC1000,	7: ISA_DPAC3000
8: PCI_8144,	9: PCI_8258,	10: PCI_8102,	11: PCI_V8258
12: PCI_V8254,	13: PCI_8158A,	14: PCI_20408C,	15: PCI_8353
16: PCI_8392F,	17: PCI_C154,	18: PCI_C154_PLUS,	19: PCI_8353_RT
20: PCIe_8338,	21: PCIe_8154,	22: PCIE_8158,	23: ENET_EMX100,
24: PCIe_8334,	25:PCIe_8332,	26:PCIe-8331,	27: PCIE_7856

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 CardName;
//板号为0。获取板卡名称。
APS_get_card_name ( 0, & CardName);
```

还可以看看:

APS_disable_device	禁用指定的设备。用于在初始化期间忽略禁用设备。
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

此函数可以通过其名称禁用指定的板卡。用于在初始化期间忽略禁用设备。

句法:

C/C++:

```
I32 FNTYPE APS_disable_device( I32 DeviceName );
```

Visual Basic:

```
APS_disable_device( ByVal DeviceName As Long ) As Long
```

参数:

I32 DeviceName : 指定设备名称。

0: PCI_8392,	1: PCI_825x,	2: PCI_8154,	3: PCI_785X
4: PCI_8158,	5: PCI_7856,	6: ISA_DPAC1000,	7: ISA_DPAC3000
8: PCI_8144,	9: PCI_825458,	10: PCI_8102,	11: PCI_V8258
12: PCI_V8254,	13: PCI_8158A,	14: PCI_20408C,	15: PCI_8353
16: PCI_8392F,	17: PCI_C154,	18: PCI_C154_PLUS,	19: PCI_8353_RT
20: PCIe_8338,	21: PCIe_8154,	22: PCIE_8158,	23: ENET_EMX100,
24: PCIe_8334	25:PCIe_8332,	26:PCIE_8331,	27:PCIE_7856

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardID_InBits;
```

```
//禁用 PCI-8258
APS_disable_device( 9 );

//初始化所有卡，除了 PCI_8258。
APS_initial( &BoardID_InBits, 0 );
```

还可以看看:

APS_set_board_param	设置板卡参数
支持的产品:PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI(e)-7856, EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于设置与板卡相关的各种参数。

有关定义和详细描述，请参阅 [板卡参数表](#)。

句法:

C/C++

```
I32 FNTYPE APS_set_board_param( I32 Board_ID, I32 BOD_Param_No, I32
BOD_Param );
```

Visual Basic:

```
APS_set_board_param (ByVal Board_ID As Long, ByVal BOD_Param_No As Long, ByVal
BOD_Param As Long) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 BOD_Param_No: 板卡参数编号。请参考 [板卡参数表](#) 中的定义。

I32 BOD_Param: 板卡参数值。有关详细信息，请参考 [板卡参数表](#)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//板卡 ID 为 0 , 将 EMG 逻辑设置为 1。
APS_set_board_param( 0, 0x00, 1 );
```

还可以看看:

[APS_get_board_param\(\)](#)

APS_get_board_param	获取板卡参数
支持的产品:PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI(e)-7856, EMX-100 , PCIe-833x	

描述:

该函数用于获取与板卡相关的各种参数。有关定义和详细描述，请参阅板卡参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_board_param( I32 Board_ID, I32 BOD_Param_No, I32
*BOD_Param );
```

Visual Basic:

```
APS_get_board_param (ByVal Board_ID As Long, ByVal BOD_Param_No As Long,
BOD_Param As Long) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 BOD_Param_No: 板卡参数编号。请参考板卡参数表进行定义。

I32 *BOD_Param: 返回板卡参数值，请参考 [板卡参数表](#).

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 paramVal;
```

//板卡 ID 为 0，将 EMG 逻辑设置为 1。

```
APS_get_board_param( 0, 0x00, & paramVal );
```

还可以看看:

[APS_set_board_param\(\)](#)

APS_set_axis_param	设置轴参数
--------------------	-------

支持的产品:PCI-8253/56, PCI-8392 (H), PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于设置一个轴的各种参数。这些参数包括运行模式、加速速率、减速速率，加速度，运动 I/O 逻辑等。有关定义和详细描述，请参见轴参数表。

句法:

C/C++

```
I32 FNTYPE APS_set_axis_param( I32 Axis_ID, I32 AXS_Param_No, I32 AXS_Param );
```

Visual Basic:

```
APS_set_axis_param(ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, ByVal  
AXS_Param As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 AXS_Param_No: 轴参数号从 0 到 65535。每个参数是由 3-6 个字符组成的唯一符号定义的。请参考轴参数表。

I32 AXS_Param: 轴参数值。请参考轴参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

//轴 ID 为 0。将 EL 逻辑设置为 1。

```
APS_set_axis_param ( 0, 0x00, 1 );
```

还可以看看:

[APS_get_axis_param\(\)](#)

APS_get_axis_param	获取轴参数
支持的产品:PCI-8253/56, PCI-8392 (H), PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于设置一个轴的各种参数。这些参数包括运行模式、加速速率、减速速率、加速度、运动 I/O 逻辑等。有关定义和详细描述，请参见轴参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_axis_param( I32 Axis_ID, I32 AXS_Param_No, I32 *AXS_Param );
```

Visual Basic:

```
APS_get_axis_param (ByVal Axis_ID As Long, ByVal AXS_Param_No As Long,  
AXS_Param As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 AXS_Param_No: 轴参数号从 0 到 65535。每个参数是由 3-6 个字符组成的唯一符号定义的。请参考轴参数表。

I32 *AXS_Param: 轴参数值。请参考轴参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 paramVal;  
//轴 ID 为 0 , 获取 EL 逻辑值。  
APS_get_axis_param ( 0, 0x00, &paramVal );
```

还可以看看:

[APS_set_axis_param\(\)](#)

APS_set_axis_param_f	设置 64 位的轴参数
支持的产品 : PCIe-8154/8158, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于设置 64 位参数。这些 64 位的参数包括加速速率，减速速率，加加速度等。请参考轴参数表的定义和详细描述

句法:

C/C++

```
I32 FNTYPE APS_set_axis_param_f( I32 Axis_ID, I32 AXS_Param_No, F64 AXS_Param );
```

Visual Basic:

```
APS_set_axis_param_f(ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, ByVal  
AXS_Param As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 AXS_Param_No: 轴参数号从 0 到 65535。每个参数是由 3-6 个字符组成的唯一符号定义的。请参考轴参数表。

F64 AXS_Param: 轴参数值。(F64 type)请参考轴参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//轴 ID 为 0。将加速度设置为 100000.0
```

```
APS_set_axis_param_f ( 0, 0x13, 100000.0 );
```

还可以看看:

[APS_get_axis_param_f\(\)](#); [APS_set_axis_param\(\)](#); [APS_get_axis_param\(\)](#)

APS_get_axis_param_f	设置 64 位的轴参数
----------------------	-------------

支持的产品 : PCIe-8154/8158, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于通过浮点数获取 64 位轴参数。64 位的参数包括加速速率，减速速率，加速度等。请参考轴参数表的定义和详细描述。

句法:

C/C++:

```
I32 FNTYPE APS_get_axis_param_f( I32 Axis_ID, I32 AXS_Param_No, F64 *AXS_Param );
```

Visual Basic:

```
APS_get_axis_param_f(ByVal Axis_ID As Long, ByVal AXS_Param_No As Long,  
AXS_Param As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 AXS_Param_No: 轴参数号从 0 到 65535。每个参数是由 3-6 个字符组成的唯一符号定义的。请参考轴参数表。

F64 AXS_Param: 轴参数值。(F64 type)请参考轴参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
F64 paramVal;
```

```
//轴 ID 为 0。获取加速度。
```

```
APS_get_axis_param_f( 0, 0x13, &paramVal );
```

还可以看看:

[APS_set_axis_param_f\(\)](#); [APS_set_axis_param\(\)](#); [APS_get_axis_param\(\)](#)

APS_get_system_timer	获取系统统计时器计数器
支持的产品:PCI-8253/56, PCI-8392 (H), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取系统统计时器计数器。系统准备好后，该计数器将在每个循环周期时间内递增计数。用户可以使用此函数来检查系统是否处于受控状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_system_timer( I32 Board_ID, I32 *Timer );
```

Visual Basic:

```
APS_get_system_timer( ByVal Board_ID As Long, Timer As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 *Timer: 返回系统统计时器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 timer;
```

```
//获取板卡 ID 为 0 的系统定时器
```

```
APS_get_system_timer( 0, &timer );
```

还可以看看:

APS_get_device_info	获取设备信息
支持的产品:PCI-8253/56, PCI-8392 (H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取指定设备（板卡）的信息。这些信息包括驱动程序版本、固件版本、PCB 版本等。请参阅设备信息表。

句法:

C/C++

```
I32 FNTYPE APS_get_device_info( I32 Board_ID, I32 Info_No, I32 *Info );
```

Visual Basic:

```
APS_get_device_info( ByVal Board_ID As Long, ByVal Info_No As Long, Info As Long )
As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 Info_No: 参考设备信息表。

I32 *Info: 参考设备信息表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Info;
ret = APS_get_device_info( 0, 1, &Info );
if( ret != ERR_NoError )
{
    //显示设备信息。
}
```

还可以看看:

APS_get_first_axisId	获取指定板卡的第一个轴 ID
支持的产品:EMX-100, PCI-8254/58 / AMP-204/8C	

描述:

该函数用于获取指定板卡的第一个轴 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_first_axisId( I32 Board_ID, I32 *StartAxisID, I32 *TotalAxisNum );
```

Visual Basic:

```
APS_get_first_axisId (ByVal Board_ID As Long, StartAxisID As Long, TotalAxisNum As Long) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 * StartAxisID: 指定板卡 ID 的第一个轴 ID。

I32 * TotalAxisNum: 指定板卡 ID 的总轴数。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 StartAxisID;
```

```
I32 TotalAxisNum;
```

//板卡 ID 为 0。获取轴信息

```
APS_get_first_axisId ( 0, & StartAxisID, & TotalAxisNum );
```

还可以看看:

APS_save_parameter_to_flash	将系统和轴参数保存到闪存中
支持的产品:PCI-8253/56, PCI-8392 (H) , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于将系统和轴参数保存到闪存中。用户必须设置板卡参数，然后使用此函数将板卡的所有参数保存到闪存中。

句法:

C/C++:

```
I32 FNTYPE APS_save_parameter_to_flash( I32 Board_ID );
```

Visual Basic:

```
APS_save_parameter_to_flash( ByVal Board_ID As Long)As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
ret = APS_set_board_param ( 0 ,0x0,1); //设置 EMG LOGIC 为逆。
ret = APS_save_parameter_to_flash ( 0 );// 保存板卡和轴参数到闪存中
if( ret == ERR_NoError )
    // 成功保存参数。
```

还可以看看:

[APS_load_parameter_from_flash\(\)](#); [APS_load_parameter_from_default\(\)](#)

<code>APS_load_parameter_from_flash</code>	从闪存中加载系统和轴参数
支持的产品:PCI-8253/56, PCI-8392 (H) , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

从闪存中加载系统和轴参数。

句法:

C/C++:

```
I32 FNTYPE APS_load_parameter_from_flash( I32 Board_ID );
```

Visual Basic:

```
APS_load_parameter_from_flash(ByVal Board_ID As Long) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
ret = APS_load_parameter_from_flash ( 0 );
if( ret == ERR_NoError )
    //成功加载参数。
```

还可以看看:

`APS_save_parameter_to_flash(); APS_load_parameter_from_default()`

APS_load_parameter_from_default	以默认值加载系统和轴参数。
支持的产品:PCI-8253/56, PCI-8392(H), EMX-100 , PCIe-833x	

描述:

以默认值加载系统和轴参数。

句法:

C/C++:

```
I32 FNTYPE APS_load_parameter_from_default( I32 Board_ID );
```

Visual Basic:

```
APS_load_parameter_from_default( ByVal Board_ID As Long )As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
ret = APS_load_parameter_from_default( 0 );  
if( ret == ERR_NoError )  
    //成功加载参数。
```

还可以看看:

[APS_save_parameter_to_flash\(\)](#); [APS_load_parameter_from_flash\(\)](#)

<code>APS_set_security_key</code>	设置安全密码
支持的产品:PCI-8144, PCI-8154/58	

描述:

此函数用于将安全代码 (16 位) 设置到控制器上的 EEPROM 中。因此 , 关闭电源时永远不会清除安全代码。

请勿经常使用此功能。EEPROM 仅保证访问 1,000,000 次。

句法:

C/C++:

```
I32 FNTYPE APS_set_security_key( I32 Board_ID, I32 OldPassword, I32 NewPassword );
```

Visual Basic:

```
APS_set_security_key(ByVal Board_ID As Long, ByVal OldPassword As Long, ByVal  
NewPassword As Long )As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 OldPassword: 将当前 (旧) 密码存储在 EEPROM 中。 (16 位)

I32 NewPassword: 用新密码替换旧密码。 (16 位)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;  
I32 OldPassword = 0x1234;  
I32 NewPassword = 0x5678;
```

```
Ret = APS_set_security_key(0, OldPassword, NewPassword );  
// 检查返回值...
```

还可以看看:

[APS_check_security_key\(\)](#); [APS_reset_security_key\(\)](#)

APS_check_security_key	验证安全密码
支持的产品:PCI-8144, PCI-8154/58	

描述:

此函数用于验证用户通过 “APS_set_security_key()” 存储在 EEPROM 中的安全代码。

句法:

C/C++:

```
I32 FNTYPE APS_check_security_key( I32 Board_ID, I32 Password );
```

Visual Basic:

```
APS_check_security_key( ByVal Board_ID As Long, ByVal Password As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31.

I32 Password: 16 位密码。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;  
I32 OldPassword = 0x1234;  
I32 NewPassword = 0x5678;
```

```
Ret = APS_set_security_key(0, OldPassword, NewPassword );  
// 检查返回值...
```

```
Ret = APS_check_security_key(0, NewPassword );
```

```
If( Ret == ERR_NoError )
```

```
{
```

```
    // 密码检查通过。
```

```
}else
```

```
{
```

```
    // 密码检查失败。
```

```
}
```

还可以看看:

[APS_set_security_key\(\)](#); [APS_reset_security_key\(\)](#)

APS_reset_security_key	重置安全密码
支持的产品:PCI-8144, PCI-8154/58	

描述:

该函数用于将存储在 EEPROM 中的安全代码重置为默认值。默认的安全代码是 0x0000。

句法:

C/C++:

```
I32 FNTYPE APS_reset_security_key( I32 Board_ID );
```

Visual Basic:

```
APS_reset_security_key( ByVal Board_ID As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;  
Ret = APS_reset_security_key( 0 );  
If( Ret == ERR_NoError ) // Security key reset success.
```

还可以看看:

[APS_set_security_key\(\)](#); [APS_check_security_key\(\)](#)

APS_save_param_to_file	将参数保存到文件中
支持的产品:PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO	

描述:

此函数用于将轴参数和板卡参数保存到 XML 文件中。当用户指定一个现有的 XML 文件时，这些参数将覆盖指定的文件。当用户输入一个 NULL 文件时，系统会自动创建一个新的 XML 文件并将这些参数保存到其中。

对于现场总线运动系列，该现场总线上不同从站的所有轴参数均保存到 xml 文件中。如果通信质量不稳定，则返回 ERR_TimeOut.

注意: 使用此函数时，将调用另一个“ApsXmlParser.dll”动态 dll。该 dll 将在安装 SDK 后被安装到系统文档中。

注意: 如果用户输入一个 NULL 文件，对于 MotionNet 系列来说，所创建的 XML 文件的默认名称为“MotionNetParam.xml”。

句法:

C/C++:

```
I32 FNTYPE APS_save_param_to_file( I32 Board_ID, const char *pXMLFile );
```

Visual Basic:

```
APS_save_param_to_file( ByVal Board_ID As Long , pXMLFile As String ) As Long
```

参数:

const char *pXMLFile: 指定一个由 MCPro2.exe 创建的现有的 XML 文件。否则，输入一个空文件以自动创建一个新的 XML 文件。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;
I32 BoardID_InBits;
I32 Mode = 0; //由系统分配
I32 BoardID = 0;

APS_initial( &BoardID_InBits, Mode);
//输入一个现有的文件，然后将其覆盖。
Ret =
APS_save_param_to_file( BoardID , "C:\\WINDOWS\\system32\\ApsParameters.xml
" );
```

```
//否则，输入一个 NULL 文件创建一个新的 XML 文件。  
Ret = APS_save_param_to_file( BoardID, NULL );  
If( Ret != ERR_NoError )  
{ //错误 – 将参数保存到文件中。 }
```

还可以看看：

APS_get_axis_param(); APS_get_board_param()

APS_load_param_from_file	从文件中加载参数
支持的产品:All products.	

描述:

该函数用于加载在输入文件(XML 文件)中被重新编码过的所有参数。

您可以使用 Motion creator Pro2 实用工具来创建或者修改 XML 文件。

该函数使用以下函数来处理 XML 文件。

APS_set_axis_param()

APS_set_board_param()

APS_set_axis_param_f() (PCIe-8154/8158 and PCI-8254/58 / AMP-204/8C)

当它处理一个无法识别的参数或者错误参数时，加载过程将立即停止并且返回一个错误值。这样，在这之后的另外一个参数将不会设置到设备中。因此，在加载到系统之前，您必须有效地检查文件。

句法:

C/C++:

I32 FNTYPE APS_load_param_from_file(const char *pXMLFile);

Visual Basic:

APS_load_param_from_file(pXMLFile As String) As Long

参数:

const char *pXMLFile: 指定一个由 MCPro2.exe 创建的 XML 文件。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;
```

```
I32 BoardID_InBits;
```

```
I32 Mode = 0; //由系统分配。
```

```
APS_initial( &BoardID_InBits, Mode);
```

```
Ret =
```

```
APS_load_param_from_file( "C:\\\\WINDOWS\\\\system32\\\\ApsParameters.xml" );
```

```
If( Ret != ERR_NoError )
```

```
{ //从文件中加载参数发生错误.}
```

还可以看看:

APS_set_axis_param(); APS_set_board_param()

APS_register_emx	在库中注册 EMX
支持的产品 : EMX-100	

描述:

该函数用于用户在使用 APS_initial()开始初始化过程之前，在 APS 库中注册 EMX 系列产品。

句法:

C/C++:

```
I32 FNTYPE APS_register_emx(I32 emx_online, I32 option);
```

Visual Basic:

```
APS_register_emx (emx_online As Long, option As Long) As Long
```

参数:

I32 emx_online: 0: 不使用 EMX 产品; 1: 使用 EMX 产品

I32 option: 保留

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
ret = APS_register_emx( 1, 0); // 在 APS 库中注册 EMX 系列产品
ret = APS_initial( &BoardID_InBits, Mode);
...// 具体的工作
ret = APS_close(); //关闭系统中所有的板卡
```

还可以看看:

[APS_initial\(\)](#)

APS_get_deviceIP	获取设备 IP
支持的产品:EMX-100	

描述:

此函数用于获取设备的 IP。执行 APS_initial()之后，用户可以通过传递指定板卡的 ID 来获得每个板卡的 IP。

句法:

C/C++:

```
I32 FNTYPE APS_get_deviceIP(I32 Board_ID, char** ipAddress);
```

Visual Basic:

```
APS_get_deviceIP (ByVal Board_ID As Integer, ByRef options As String) As Integer
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

Char** ipAddress: 获取 EMX 的 IP。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;  
char** ip = (char**)malloc(sizeof(char*));  
ret = APS_get_deviceIP(Board_ID, ip);
```

还可以看看:

APS_reset_emc_alarm

重置设备的报警信号

支持的产品:EMX-100

描述:

当伺服驱动器发生告警，并且告警的严重性不是很关键时，您可以通过此函数重置告警信号。

句法:

C/C++:

```
I32 FNTYPE APS_reset_emx_alarm(I32 Axis_ID);
```

Visual Basic:

```
APS_reset_emx_alarm (ByVal Axis_ID As Integer) As Integer
```

参数:

I32 Axis_ID: 轴的编号。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Axis_ID = 0;  
ret = APS_reset_emc_alarm( Axis_ID);  
if( ret != ERR_NoError )  
{  
    printf( "Reset alarm successful.\n" );  
}
```

还可以看看:

APS_get_curr_sys_ctrl_mode	在 FPGA 中获取当前的系统控制模式
----------------------------	---------------------

支持的产品:, PCI-8254/58 / AMP-204/8C

描述:

此函数用于获取 FPGA 中的当前控制模式。有三种控制模式：一种是脉冲模式，一种是模拟模式，还有一种是步进模式。用户可以得到指定轴的控制方式。

句法:

C/C++:

```
I32 FNTYPE APS_get_curr_sys_ctrl_mode(I32 Axis_ID, I32 * Mode);
```

Visual Basic:

```
APS_get_curr_sys_ctrl_mode( ByVal Axis_ID As Long, Mode As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 * Mode: 控制模式:

0: 脉冲模式

1: 模拟模式

2: 步进模式

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Mode = 0;
```

```
//获取每个轴的控制模式。
```

```
APS_get_curr_sys_ctrl_mode( Axis_ID, &Mode );
```

还可以看看:

04. SSCNET 函数

APS_start_sscnet	开启 SSCNET 网络
支持的产品:PCI-8392(H)	

描述:

此函数用于启动 SSCNET 网络。一旦启动，SSCNET 将开始搜索连接到网络的伺服驱动器。它将返回轴连接状态，并以 32 位值内的位表示。该函数将一直保持，直到用户发布该函数时建立 SSCNET 通信为止。

在启动网络之前，应先设置一些 SSCNET 参数，例如 SSCNET 循环时间等。有关详细说明，请参考 SSCNET 参数表。

句法:

C/C++:

```
I32 FNTYPE APS_start_sscnet( I32 Board_ID, I32 *AxisFound_InBits );
```

Visual Basic:

```
APS_start_sscnet (ByVal Board_ID As Long, AxisFound_InBits As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 *AxisFound_InBits: 用 bit 来返回的已连接的轴。

例如, AxisFound_InBits = 0x111 表示轴开关索引 0, 4 和 8 连接在线。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"

I32 AxisFound_InBits;
I32 ret;

//在启动 sscnet 之前设置 SSCNET 的相关参数。

//开启 sscnet.
Ret = APS_start_sscnet( 0, &AxisFound_InBits );
if( ret == ERR_NoError )
{
    // 伺服控制...
```

```
}

// 停止 sscnet.
Ret = APS_stop_sscnet( 0 );
```

还可以看看:

APS_stop_sscnet();APS_set_board_param(); APS_get_board_param()

APS_stop_sscnet	停止 SSCNET 网络
支持的产品: PCI-8392(H)	

描述:

此函数用于停止 SSCNET 网络。一旦停止，SSCNET 将停止与伺服驱动器通信，此后所有伺服驱动器将自由运行。

句法:

C/C++:

```
I32 FNTYPE APS_stop_sscnet( I32 Board_ID );
```

Visual Basic:

```
APS_stop_sscnet (ByVal Board_ID As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"

I32 AxisFound_InBits;
I32 ret;

// 在启动 sscnet 之前设置 SSCNET 的相关参数。

// 开启 sscnet。
Ret = APS_start_sscnet( 0, &AxisFound_InBits );
if( ret == ERR_NoError )
{
    // 伺服控制...
}

// 停止 sscnet。
Ret = APS_stop_sscnet( 0 );
```

还可以看看:

[APS_start_sscnet\(\)](#)

APS_get_sscnet_servo_param	读取当前伺服参数值
支持的产品: PCI-8392(H)	

描述:

此函数用于从伺服驱动器获取伺服参数。用户可以一次读取两个伺服参数。也可使用 Para_No1 读取一个参数。如果用户设置 Para_No2 = 0 , Para_dat2 将被设置为 null。该函数仅在 SSCNET 网络启动后才有效。
切勿尝试更改制造商设置的参数。
伺服驱动器参数的定义 , 请参照 Mitsubishi J3B 使用手册。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_servo_param( I32 Axis_ID, I32 Para_No1, I32 *Para_Dat1,
I32 Para_No2, I32 *Para_Dat2 );
```

Visual Basic:

```
APS_get_sscnet_servo_param( ByVal Axis_ID As Long, ByVal Para_No1 As Long,
Para_Dat1 As Long, ByVal Para_No2 As Long, Para_Dat2 As Long) As Long
```

参数:

I32 Axis_ID: 轴ID从0到65535。

I32 Para_No1: 伺服参数编号。参数的含义 , 请参考伺服驱动器使用手册。

格式 : 0 x 0 N XX

N : PA : 0, PB : 1, PC : 2, PD : 3

XX: 参数编号。

例如. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 *Para_Dat1:

I32 Para_No2: 伺服参数编号。参数的含义 , 请参考伺服驱动器使用手册。

格式 : 0 x 0 N XX

N : PA : 0, PB : 1, PC : 2, PD : 3

XX: 参数编号。

例如. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 *Para_Dat2: I32 变量的指针。当Para_No2被设置为0时 , 可以将Para_Dat2设置为null (0)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
I32 AxisFound_InBits;
I32 ret;
I32 Para_Dat1, Para_Dat2;

// 在启动 sscnet 之前设置 SSCNET 的相关参数。

// 开启 sscnet。
Ret = APS_start_sscnet( 0, &AxisFound_InBits );
if( ret == ERR_NoError )
{
    //仅在建立网络时使用此函数。
    Ret = APS_get_sscnet_servo_param( 0, 0x0107, &Para_Dat1, 0x0108, &Para_Dat2 );
}
...
...
```

还可以看看:

APS_set_sscnet_servo_param()

APS_set_sscnet_servo_param	设置伺服参数
----------------------------	--------

支持的产品: PCI-8392(H)

描述:

此函数用于设置伺服驱动器。用户可以一次写入两个伺服参数。也可使用 Para_No1 写入一个参数。如果用户设置 Para_No2 = 0，则 Para_Dat2 不代表任何意义。

该函数仅在 SSCNET 网络启动后才有效。

网络启动后，不允许更改某些伺服参数。用户应重新启动网络以使其工作。

伺服驱动器参数的定义，请参照 Mitsubishi J3B 使用手册。

句法:

C/C++:

```
I32 FNTYPE APS_set_sscnet_servo_param( I32 Axis_ID, I32 Para_No1, I32 Para_Dat1, I32  
Para_No2, I32 Para_Dat2 );
```

Visual Basic:

```
APS_set_sscnet_servo_param(ByVal Axis_ID As Long, ByVal Para_No1 As Long, ByVal  
Para_Dat1 As Long, ByVal Para_No2 As Long, ByVal Para_Dat2 As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Para_No1: 伺服参数编号。参数的含义，请参考伺服驱动器使用手册。

格式 : 0 x 0 N XX

N : PA : 0, PB : 1, PC : 2, PD : 3

XX: 参数编号。

例如. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 Para_Dat1:

I32 Para_No2: 伺服参数编号。参数的含义，请参考伺服驱动器使用手册。

格式 : 0 x 0 N XX

N : PA : 0, PB : 1, PC : 2, PD : 3

XX: 参数编号。

例如. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 Para_Dat2: 伺服参数数据。当Para_No2被设置为0时，可以将Para_Dat2设置为null (0)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
I32 AxisFound_InBits;
I32 ret;

// 在启动 sscnet 之前设置 SSCNET 的相关参数。

// 开启 sscnet。
Ret = APS_start_sscnet( 0, &AxisFound_InBits );
if( ret == ERR_NoError )
{
    // 仅在建立网络时使用此函数。
    Ret = APS_set_sscnet_servo_param( 0, 0x0009, 13, 0, 0 );
    // 检查返回值查看函数成功返回...
}
```

还可以看看：

APS_get_sscnet_servo_param()

APS_get_sscnet_servo_alarm	获取当前伺服的报警信息
支持的产品: PCI-8392(H)	

描述:

该函数用于当发生伺服报警时获取报警编号。报警信息包括报警编号和报警的详细信息。详细说明请参考伺服驱动器使用手册。

当发生伺服报警时，用户应在复位报警前使用此函数，否则报警信息将被复位。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_servo_alarm( I32 Axis_ID, I32 *Alarm_No, I32
*Alarm_Detail );
```

Visual Basic:

```
APS_get_sscnet_servo_alarm(ByVal Axis_ID As Long, Alarm_No As Long, Alarm_Detail
As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Alarm_No: 报警编号。请参考伺服驱动器使用手册。

I32 *Alarm_Detail: 报警详情。请参考伺服驱动器使用手册。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Alarm_No;
```

```
I32 Alarm_Detail;
```

```
...//发生报警!
```

```
APS_get_sscnet_servo_alarm(Axis_ID, &Alarm_No, &Alarm_Detail ); //Get alarm
operation123i
```

```
...//消除告警原因
```

```
APS_reset_sscnet_servo_alarm(Axis_ID ); //复位伺服器报警
```

```
...
```

还可以看看:

APS_reset_sscnet_servo_alarm()

APS_reset_sscnet_servo_alarm	伺服报警复位
支持的产品: PCI-8392(H)	

描述:

当发生伺服报警时，伺服电机将停止运动。报警条件通过后，此函数可帮助清除报警并复位伺服器。

句法:

C/C++:

```
I32 FNTYPE APS_reset_sscnet_servo_alarm( I32 Axis_ID );
```

Visual Basic:

```
APS_reset_sscnet_servo_alarm(ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Alarm_No;
```

```
I32 Alarm_Detail;
```

```
...//发生报警！
```

```
APS_get_sscnet_servo_alarm(Axis_ID, &Alarm_No, &Alarm_Detail ); //Get alarm  
operation125i
```

```
...//消除告警原因
```

```
APS_reset_sscnet_servo_alarm(Axis_ID ); //复位伺服器报警...
```

还可以看看:

[APS_get_sscnet_servo_alarm\(\)](#)

APS_save_sscnet_servo_param	将伺服参数保存到闪存中
支持的产品: PCI-8392(H)	

描述:

此函数用于将伺服参数从 SDRAM 保存到控制器卡上的闪存中。

当系统（控制器）开机时，它将伺服参数从闪存或从默认表中复制到 SDRAM。在建立 SSCNET 网络后，伺服参数将传输到伺服驱动器中。用户可以从轴参数中选择另一种模式，即在建立网络后，伺服驱动器将保留其设置。如果 Axis 为空（不使用轴 ID），则该参数被保留为默认值。

当发布该函数时，所有轴（16 个轴）的伺服参数将立即保存。您不能单独保存每个伺服驱动器的参数。

句法:

C/C++:

```
I32 FNTYPE APS_save_sscnet_servo_param( I32 Board_ID );
```

Visual Basic:

```
APS_save_sscnet_servo_param(ByVal Board_ID as Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
// 配置伺服参数。
// APS_set_sscnet_servo_param ...
APS_save_sscnet_servo_param( Board_ID ); // 将伺服参数保存到闪存。
```

还可以看看:

[APS_set_sscnet_servo_param\(\)](#); [APS_get_sscnet_servo_param\(\)](#)

APS_get_sscnet_servo_abs_position

从伺服驱动器获取绝对参考位置

支持的产品: PCI-8392(H)

描述:

该函数用于从 SSCNET 伺服驱动器获取绝对位置。仅当 SSCNET 网络启动时才能使用此函数。通常，为了建立 ABS 位置系统，用户必须先执行归零操作，然后用户必须使用此函数才能从伺服驱动器中获得绝对位置。同时，控制器会将伺服驱动器的绝对位置复制到轴参数中。最后，用户可以使用 APS_save_sscnet_servo_abs_position() 将所有轴的 ABS 信息保存在闪存中，以备下次使用。

轴参数定义

PRA_SSC_SERVO_ABS_CYC_CNT

PRA_SSC_SERVO_ABS_RES_CNT

轴参数的详细信息请参考轴参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_servo_abs_position( I32 Axis_ID, I32 *Cyc_Cnt, I32  
*Res_Cnt );
```

Visual Basic:

```
APS_get_sscnet_servo_abs_position( ByVal Axis_ID As Long, Cyc_Cnt As Long, Res_Cnt  
As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Cyc_Cnt: 伺服驱动器的周期计数器

I32 *Res_Cnt: 伺服驱动器的精度计数器

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//1. 初始化板卡并启动 SSCNET 网络。
```

```
//2. 执行归零操作
```

```
Ret = APS_get_sscnet_servo_abs_position( Axis_ID, Cyc_Cnt, Res_Cnt );
```

```
// 记录 abs. 位置数据，下一次归零操作。
```

还可以看看:

[APS_save_sscnet_servo_abs_position\(\)](#); [APS_load_sscnet_servo_abs_position\(\)](#);

[APS_set_axis_param\(\)](#); [APS_get_axis_param\(\)](#)

APS_save_sscnet_servo_abs_position	将绝对参考位置保存到闪存中
支持的产品: PCI-8392(H)	

描述:

该函数用于将绝对参考位置保存到闪存中。通常，为了建立绝对位置系统，用户必须先执行回归零操作，然后使用“APS_get_sscnet_servo_abs_position”函数从驱动器中获得绝对位置。最后，用户必须调用此函数才能将轴的所有绝对位置保存到闪存中，以备下次使用。请注意，当用户使用此函数时，所有轴（16个轴）的伺服参数将立即保存。您不能单独保存每个伺服驱动器。

轴参数定义
PRA_SSC_SERVO_ABS_CYC_CNT
PRA_SSC_SERVO_ABS_RES_CNT

句法:

C/C++:

```
I32 FNTYPE APS_save_sscnet_servo_abs_position( I32 Board_ID );
```

Visual Basic:

```
APS_save_sscnet_servo_abs_position( ByVal Board_ID As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//1. 初始化板卡并启动 SSCNET 网络。
//2. 执行归零操作。
//3. 获取伺服驱动器的 abs 位置。
For(Axis_ID = 0; Axis_ID < 16; Axis_ID ++ )
{
    Ret = APS_get_sscnet_servo_abs_position( Axis_ID, Cyc_Cnt, Res_Cnt );
}
Ret = APS_save_sscnet_servo_abs_position( Board_ID ); //将所有的 abs.位置保存到闪存中...
```

还可以看看:

```
APS_get_sscnet_servo_abs_position();APS_load_sscnet_servo_abs_position();
APS_set_axis_param(); APS_get_axis_param()
```

APS_load_sscnet_servo_abs_position	从闪存中加载绝对参考位置
------------------------------------	--------------

支持的产品: PCI-8392(H)

描述:

该函数用于从闪存中加载伺服驱动器的绝对位置到轴参数中。如果用户从未保存过伺服驱动器的绝对位置，则调用该函数将返回错误。

用户可以通过指定的函数参数“Option”一次加载所有的ABS位置，以方便使用。请参阅参数说明。

通常，如果用户要使用ABS位置系统，必须在建立SSCNET网络之前，使用此函数将ABS信息从闪存加载到轴参数中。在建立SSCNET网络之前，还需要在轴参数中将ABS位置系统启用。

句法:

C/C++:

```
I32 FNTYPE APS_load_sscnet_servo_abs_position( I32 Axis_ID, I32 Option, I32 *Cyc_Cnt,  
I32 *Res_Cnt );
```

Visual Basic:

```
APS_load_sscnet_servo_abs_position( ByVal Axis_ID As Long, ByVal Option As Long,  
Cyc_Cnt As Long, Res_Cnt As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 加载选项。

0: 将一个轴的 ABS 位置加载到轴参数中。

1: 将所有轴的 ABS 位置加载到轴参数。

I32 *Cyc_Cnt: 从闪存中获取周期计数器。将此参数设置为 0 可忽略。

I32 *Res_Cnt: 从闪存中获取精度计数器。将此参数设置为 0 可忽略。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//1. 初始化板卡。  
//2. 从闪存中加载 abs. 位置。  
Ret = APS_load_sscnet_servo_abs_position(Axis_ID, 1, 0 ,0); //Option = 1 加载所有轴  
APS_set_axis_param( Axis_ID, PRA_SSC_SERVO_ABS_POS_OPT, 1 ); //启用 abs. 位置系  
统。 APS_start_sscnet( Board_ID, &AxisFound_InBits ); //启用 SSCNET 网络。  
// 通过绝对运动函数返回原点。
```

还可以看看:

```
APS_get_sscnet_servo_abs_position();APS_save_sscnet_servo_abs_position();
APS_set_axis_param();APS_get_axis_param()
```

APS_get_sscnet_link_status	获取 SSCNET 链接状态
支持的产品: PCI-8392(H)	

描述:

此函数用于获取 SSCNET 链接状态。您可以轻松地使用此函数来检查 SSCNET 连接是否已链接。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_link_status( I32 Board_ID, I32 *Link_Status );
```

Visual Basic:

```
APS_get_sscnet_link_status( ByVal Board_ID As Long, Link_Status As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID , 0 为基本参数

I32 *Link_Status: 链接状态

 Return 1 : SSCNET 已连接

 Return 0 : SSCNET 未连接

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
#include "ErrorCodeDef.h"
I32 link; // 获取 SSCNET 链接状态。
I32 err;
// 开启 sscnet。
// 检查 SSCNET 连接状态。
Do{
    err = APS_get_sscnet_link_status( 0, &link );
    if( link == 0 )
    {
        // 链接断开;
    }
}while( err == ERR_NoError )
```

还可以看看:

APS_set_sscnet_servo_monitor_src	设置伺服监控数据源
----------------------------------	-----------

支持的产品: PCI-8392(H)

描述:

此函数用于设置每个伺服监控通道的信号源。

在 SSCNETIII 控制器中，每个轴有 4 个通道，可用于监控 SSCNET 伺服驱动器的状态。您可以通过此函数更改监控源。监控源请参考 SSCNET 伺服监控源表。另外，您可以通过

"APS_get_sscnet_servo_monitor_data()" 获取监控数据。

当 SSCNET 已连接并通信时，该函数有效。

句法:

C/C++:

```
I32 FNTYPE APS_set_sscnet_servo_monitor_src( I32 Axis_ID, I32 Mon_No, I32  
Mon_Src );
```

Visual Basic:

```
APS_set_sscnet_servo_monitor_src( ByVal Axis_ID As Long, ByVal Mon_No As Long,  
ByVal Mon_Src As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Mon_No: 监控通道编号。0~3 指通道 0~通道 3。

I32 Mon_Src: 监控源编号。请参考 SSCNET 伺服监控源表。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
// 初始化 APS 库并先启动 SSCNET。
```

```
{
```

```
    I32 ret;
```

```
    I32 Axis_ID = 0;
```

```
    ret = APS_set_sscnet_servo_monitor_src( Axis_ID, 0, 1 ); //设置通道 0, 源 = 1.
```

```
    //检查返回值。
```

```
    Ret = APS_set_sscnet_servo_monitor_src( Axis_ID, 1, 2 ); //设置通道 1, 源 = 2.
```

```
    //检查返回值。
```

}

还可以看看：

APS_get_sscnet_servo_monitor_src(); APS_get_sscnet_servo_monitor_data()

支持的产品: PCI-8392(H)

APS_get_sscnet_servo_monitor_src	获取伺服监控数据源
----------------------------------	-----------

描述:

此函数用于获取每个伺服监控通道的信号源。

在 SSCNETIII 控制器中，每个轴有 4 个通道，可用于监控 SSCNET 伺服驱动器的状态。您可以通过此功能获取监控源。监控源请参考 [SSCNET 伺服监控源表](#)。

当 SSCNET 已连接并通信时，该函数有效。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_servo_monitor_src( I32 Axis_ID, I32 Mon_No, I32  
*Mon_Src );
```

Visual Basic:

```
APS_get_sscnet_servo_monitor_src( ByVal Axis_ID As Long, Mon_No As Long, ByVal  
Mon_Src As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Mon_No: 监控通道编号。0~3 指通道 0~通道 3。

I32 Mon_Src: 监控源编号。请参考 [SSCNET 伺服监控源表](#)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"  
#include "ErrorCodeDef.h"  
  
// 初始化 APS 库并先启动 SSCNET。  
{  
    I32 ret;  
    I32 Axis_ID = 0;  
    I32 Mon_Src;  
  
    ret = APS_get_sscnet_servo_monitor_src( Axis_ID, 0, &Mon_Src );  
    //检查返回值。  
    Ret = APS_get_sscnet_servo_monitor_src( Axis_ID, 1, &Mon_Src );  
    //检查返回值。  
}
```

还可以看看:

APS_set_sscnet_servo_monitor_src();APS_get_sscnet_servo_monitor_data()

APS_get_sscnet_servo_monitor_data	获取伺服监控数据
-----------------------------------	----------

支持的产品: PCI-8392(H)

描述:

此函数用于获取 sscnet 伺服监控数据。仅当 SSCNET 已连接时才能使用此函数。

在 SSCNETIII 控制器中，每个轴有 4 个通道，可用于监控 SSCNET 伺服驱动器的状态。您可以使用此函数一次获取所有（4 个通道）的监控数据。另外，您可以通过函数

“APS_set_sscnet_servo_monitor_src()” 来更改监控源。监控源请参阅 SSCNET 伺服监控源表。

句法:

C/C++:

```
I32 FNTYPE APS_get_sscnet_servo_monitor_data( I32 Axis_ID, I32 Arr_Size, I32
*Data_Arr );
```

Visual Basic:

```
APS_get_sscnet_servo_monitor_data( ByVal Axis_ID As Long, ByVal Arr_Size As Long,
Data_Arr As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Arr_Size: 指定数据数组的大小。最小值：1～最大值：4。

I32 *Data_Arr: 获取监控数据数组。数组大小依据 “Arr_Size” 而定。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//初始化 APS 库并先启动 SSCNET。
{
    I32 Axis_ID = 0; //轴 ID
    I32 Data_Arr[4]; //共 4 个通道
    I32 ret; //返回代码。

    // 获取 SSCNET 监控数据。
    Ret = APS_get_sscnet_servo_monitor_data(Axis_ID, 4, Data_Arr );
    if( ret == ERR_NoError )
    {   //显示 Data_Arr[];
```

```
 }  
 }
```

还可以看看:

APS_set_sscnet_servo_monitor_src();APS_get_sscnet_servo_monitor_src();

05. 运动 IO 和运动状态

APS_motion_status	返回运动状态
-------------------	--------

支持的产品:PCI-8253/56, PCI-8392(H), PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于获取一个轴的运动状态。状态包括运行、正常停止、由于某种原因异常停止、在等待其他轴、跟随状态、在某些模式下、在加速或减速等。状态可以超过两个，例如模式和运行。用户需要使用此函数检查轮询系统中是否完成了“发后即忘”功能。在事件驱动的系统中，用户也可以使用中断事件功能。

有关详细说明，请参阅运动状态表。

句法:

C/C++:

```
I32 FNTYPE APS_motion_status( I32 Axis_ID );
```

Visual Basic:

```
APS_motion_status (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID:轴 ID 从 0 到 65535。

返回值:

正值:

运动状态的值。数值含义请参考运动状态位编号的定义表。

负值:

错误代码：请参考错误代码表。

示例：

```
I32 MotionStatus;  
MotionStatus = APS_motion_status( Axis_ID ); //获取运动状态。  
...
```

还可以看看:

APS_motion_io_status();

APS_motion_io_status	返回运动 IO 状态
支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取一个轴的运动 I/O 信息，例如 ORG , PEL , MEL , SVON , INP 等。这些状态与外部开关或伺服驱动器有关联。

有关详细说明，请参阅运动 IO 状态表。

句法:

C/C++:

```
I32 FNTYPE APS_motion_io_status( I32 Axis_ID );
```

Visual Basic:

```
APS_motion_io_status (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

正值:

运动 IO 状态的值。数值含义请参考运动状态位编号的定义表。

负值:

错误代码: 请参考错误代码表。

示例 :

```
I32 MotionIO;
MotionIO = APS_motion_io_status(Axis_ID); //获取运动 IO 状态。
...

```

还可以看看:

APS_motion_status ();

APS_set_servo_on	设置伺服开/关
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于给指定轴的伺服驱动器命令，让其开始控制其伺服电机。然后运动函数就可以在该轴上应用。

句法:

C/C++:

```
I32 FNTYPE APS_set_servo_on( I32 Axis_ID, I32 Servo_on );
```

Visual Basic:

```
APS_set_servo_on (ByVal Axis_ID As Long, ByVal ServoOn As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Servo_on:

0: 伺服关, 1: 伺服开

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
...//初始化
APS_set_servo_on( Axis_ID, 1 ); // 设置伺服为开。
... //运动动作
APS_set_servo_on(Axis_ID, 0); //设置伺服为关
...//释放
```

还可以看看:

APS_get_position	获取反馈位置
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于获取一个轴的位置计数器。计数器以脉冲为单位。

句法:

C/C++:

```
I32 FNTYPE APS_get_position( I32 Axis_ID, I32 *Position );
```

Visual Basic:

```
APS_get_position (ByVal Axis_ID As Long, Position As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Position: 反馈位置，以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Position;
APS_get_position(Axis_ID, &Position ); //获取反馈位置
```

...

还可以看看:

[APS_get_command\(\)](#); [APS_set_position\(\)](#); [APS_set_command\(\)](#)

APS_set_position	设置反馈位置
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于设置一个轴的位置计数器。计数器以脉冲为单位。例如，它会在实例中分配了一个新位置，但是由于该函数的作用，电动机将不会运动。

句法:

C/C++:

```
I32 FNTYPE APS_set_position(I32 Axis_ID, I32 Position);
```

Visual Basic:

```
APS_set_position (ByVal Axis_ID As Long, ByVal Position As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Position: 设置反馈位置。以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

...

```
APS_set_position(Axis_ID, 0 ); // 设置反馈位置为零。
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_set_command\(\)](#)

APS_get_command	获取命令位置
支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于获取一个轴的命令计数器。计数器以脉冲为单位。

句法:

C/C++:

```
I32 FNTYPE APS_get_command( I32 Axis_ID, I32 *Command );
```

Visual Basic:

```
APS_get_command (ByVal Axis_ID As Long, Command As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Command: 命令位置。以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Command ;
APS_get_command(Axis_ID, &Command ); //获取命令位置
...//
```

还可以看看:

[APS_get_position\(\)](#); [APS_set_position\(\)](#); [APS_set_command\(\)](#)

APS_set_command	设置命令位置
支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于设置一个轴的命令计数器。计数器以脉冲为单位。它会在实例中分配一个新的命令计数器，但是由于该函数的作用，电动机将不会运动。

句法:

C/C++:

```
I32 FNTYPE APS_set_command(I32 Axis_ID, I32 Command);
```

Visual Basic:

```
APS_set_command (ByVal Axis_ID As Long, ByVal Command As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Command: 位置命令。以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
...//
```

```
APS_set_command(Axis_ID, 0); //设置命令位置为零。
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_set_position\(\)](#);

APS_get_command_velocity	获取命令速度
支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取命令速度。其最小值取决于系统的速度计算精度。

句法:

C/C++:

```
I32 FNTYPE APS_get_command_velocity(I32 Axis_ID, I32 *Velocity );
```

Visual Basic:

```
APS_get_command_velocity( ByVal Axis_ID As Long, Velocity As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Velocity: 返回命令速度。单位 : pps

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
I32 Velocity;
ret = APS_get_command_velocity ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //速度
}
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_get_feedback_velocity\(\)](#)

APS_get_feedback_velocity	获取反馈速度
支持的产品:PCI-8253/56, PCI-8392(H) , EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取反馈速度。其最小值取决于系统的速度计算精度。

句法:

C/C++:

```
I32 FNTYPE APS_get_feedback_velocity(I32 Axis_ID, I32 *Velocity);
```

Visual Basic:

```
APS_get_feedback_velocity( ByVal Axis_ID As Long, Velocity As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Velocity: 返回反馈速度。单位 : pps.

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
I32 Velocity;
ret = APS_get_feedback_velocity( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //速度
}
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_get_command_velocity \(\)](#);

APS_get_error_position	获取错误位置
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取错误位置值。该值的定义是命令位置减反馈位置。

句法:

C/C++:

```
I32 FNTYPE APS_get_error_position( I32 Axis_ID, I32 *Err_Pos );
```

Visual Basic:

```
APS_get_error_position( ByVal Axis_ID As Long, Err_Pos As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Err_Pos: 返回错误位置。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
I32 Err_Pos;
ret = APS_get_error_position(Axis_ID, &Err_Pos );
if( ret == ERR_NoError )
    //显示错误位置。
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_get_command_velocity\(\)](#); [APS_get_feedback_velocity\(\)](#)

APS_get_target_position	获取目标位置
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取目标位置记录。在线性定位模式下，该值为目标位置。在圆形定位模式下，该值与命令位置相同。在速度和点动模式下，该值与命令位置相同。

对于 EMX-100:

此函数用于获取点对点运动和线性运动的单轴目标位置。当使用点动，速度运动和归零运动时，目标位置不会更新。

句法:

C/C++:

```
I32 FNTYPE APS_get_target_position( I32 Axis_ID, I32 *Targ_Pos );
```

Visual Basic:

```
APS_get_target_position(ByVal Axis_ID As Long, Targ_Pos As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Targ_Pos: 返回目标位置。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
I32 Targ_Pos;
ret = APS_get_target_position(Axis_ID, &Targ_Pos );
if( ret == ERR_NoError )
    //显示目标位置。
```

还可以看看:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_get_command_velocity\(\)](#); [APS_get_feedback_velocity\(\)](#)

APS_get_position_f	获取 64 位的反馈位置
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于将一个轴 64 位的位置计数器。计数器以脉冲为单位。

句法:

C/C++:

```
I32 FNTYPE APS_get_position_f( I32 Axis_ID, F64 *Position );
```

Visual Basic:

```
APS_get_position_f(ByVal Axis_ID As Long, Position As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Position: 反馈位置，以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
F64 Position;
```

```
APS_get_position_f(Axis_ID, &Position ); //获取反馈位置。
```

...

还可以看看:

[APS_get_command_f\(\)](#); [APS_set_position_f\(\)](#); [APS_set_command_f\(\)](#)

<code>APS_set_position_f</code>	设置 64 位的反馈位置
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于设置一个轴的 64 位位置计数器。计数器以脉冲为单位。例如，它分配了一个新位置，但是由于该函数的作用，电动机将不会运动。

句法:

C/C++:

```
I32 FNTYPE APS_set_position_f(I32 Axis_ID, F64 Position);
```

Visual Basic:

```
APS_set_position_f(ByVal Axis_ID As Long, ByVal Position As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 Position: 设置 64 位反馈位置，以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

...

```
APS_set_position_f(Axis_ID, 0.0 ); // 设置反馈位置为零。
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_set_command_f\(\)](#)

<code>APS_get_command_f</code>	获取 64 位的命令位置
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取一个 64 位轴的命令计数器。计数器以脉冲为单位。

句法:

C/C++:

```
I32 FNTYPE APS_get_command_f( I32 Axis_ID, F64 *Command );
```

Visual Basic:

```
APS_get_command_f(ByVal Axis_ID As Long, Command As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Command: 64 位命令位置，以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
F64 Command ;
```

```
APS_get_command_f(Axis_ID, &Command ); //获取 64 位的命令位置。
```

```
...//
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_set_position_f\(\)](#); [APS_set_command_f\(\)](#)

APS_set_command_f	设置 64 位的命令位置
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于设置一个 64 位轴的命令计数器。计数器以脉冲为单位。它会在实例中分配一个新的命令计数器，但是由于该函数的作用，电动机将不会运动。

句法:

C/C++:

```
I32 FNTYPE APS_set_command_f(I32 Axis_ID, F64 Command);
```

Visual Basic:

```
APS_set_command_f(ByVal Axis_ID As Long, ByVal Command Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 Command: 64 位命令位置，以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
...//  
APS_set_command_f(Axis_ID, 0.0); //设置命令位置为 0.
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_set_position_f\(\)](#);

APS_get_target_position_f	获取 64 位的目标位置
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于获取 64 位目标位置记录。在线性定位模式下，该值为目标位置。在圆形定位模式下，该值与命令位置相同。在速度和点动模式下，该值与命令位置相同。

句法:

C/C++:

```
I32 FNTYPE APS_get_target_position_f( I32 Axis_ID, F64 *Targ_Pos );
```

Visual Basic:

```
APS_get_target_position_f(ByVal Axis_ID As Long, Targ_Pos As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Targ_Pos: 返回 64 位目标位置。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
F64 Targ_Pos;
ret = APS_get_target_position_f(Axis_ID, &Targ_Pos );
if( ret == ERR_NoError )
    //显示目标位置。
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_get_command_velocity_f\(\)](#); [APS_get_feedback_velocityf\(\)](#)

<code>APS_get_error_position_f</code>	获取 64 位的错误位置
支持的产品:MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取 64 位错误位置记录。此值定义为命令位置减去反馈位置。

句法:

C/C++:

```
I32 FNTYPE APS_get_error_position_f( I32 Axis_ID, F64 *Err_Pos );
```

Visual Basic:

```
APS_get_error_position_f(ByVal Axis_ID As Long, Err_Pos As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Err_Pos: 返回 64 位错误位置。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
F64 Err_Pos;
ret = APS_get_error_position_f(Axis_ID, &Err_Pos );
if( ret == ERR_NoError )
    //显示错误位置。
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_get_command_velocity_f\(\)](#); [APS_get_feedback_velocityf\(\)](#); [APS_get_target_position_f](#)

<code>APS_get_command_velocity_f</code>	获取 64 位的命令速度
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/58A, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取 64 位命令速度提高。最小值取决于系统的速度计算精度。

句法:

C/C++:

```
I32 FNTYPE APS_get_command_velocity_f(I32 Axis_ID, F64 *Velocity );
```

Visual Basic:

```
APS_get_command_velocity_f( ByVal Axis_ID As Long, Velocity As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Velocity: 返回 64 位命令速度。单位 : pps.

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
F64 Velocity;
ret = APS_get_command_velocity_f ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //速度。
}
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_get_feedback_velocityf\(\)](#)

<code>APS_get_feedback_velocity_f</code>	获取 64 位的反馈速度
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取 64 位反馈速度。最小值取决于系统的速度计算分辨率。

句法:

C/C++:

```
I32 FNTYPE APS_get_feedback_velocity_f(I32 Axis_ID, F64 *Velocity );
```

Visual Basic:

```
APS_get_feedback_velocity_f( ByVal Axis_ID As Long, Velocity As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

F64 *Velocity: 返回 64 位命令速度。单位 : pps。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
F64 Velocity;
ret = APS_get_feedback_velocity_f ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{}
```

还可以看看:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_get_command_velocityf\(\)](#)

APS_get_mq_free_space	获取运动队列的当前可用空间
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取运动队列的当前可用空间。每个轴都有自己的运动队列 (FIFO) , 用以缓冲运动命令。

句法:

C/C++:

```
I32 FNTYPE APS_get_mq_free_space( I32 Axis_ID, I32 *Sapce );
```

Visual Basic:

```
APS_get_mq_free_sapce (ByVal Axis_ID As Long, Sapce As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Sapce: 指定轴的运动队列的可用空间。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Space;
APS_get_mq_free_space(Axis_ID, &Space ); //Get free space of motion queue
...//
```

还可以看看:

[APS_get_mq_usage\(\)](#);

<code>APS_get_mq_usage</code>	获取运动队列的使用情况
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取运动队列的使用情况。

每个轴都有自己的运动队列 (FIFO) , 用以缓冲运动命令。

句法:

C/C++:

```
I32 FNTYPE APS_get_mq_usage( I32 Axis_ID, I32 *Usage );
```

Visual Basic:

```
APS_get_mq_usage(ByVal Axis_ID As Long, Usage As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Usage: 指定轴的运动队列使用情况

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Usage;
APS_get_mq_free_space(Axis_ID, &Usage ); //获取运动队列的使用情况
...//
```

还可以看看:

[APS_get_mq_free_space\(\)](#);

APS_get_stop_code	按轴获取停止代码
-------------------	----------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于获取停止代码。停止代码是运动停止时轴的停止原因。

可能的停止代码如下表所示。如果轴执行单轴的 PTP 运动并正常停止，则将获得属于 STOP_NORMAL 的停止代码。

符号	代码	描述
STOP_NORMAL	0	正常停止
STOP_EMG	1	当 EMG 打开时停止
STOP_ALM	2	当 ALM 打开时停止
STOP_SVNO	3	当伺服关闭时停止
STOP_PEL	4	由 PEL 信号接通停止
STOP_MEL	5	由 MEL 信号接通停止
STOP_SPEL	6	通过软限条件停止-正极限
STOP_SMEL	7	通过软限条件停止-负极限
STOP_USER_EMG	8	由用户紧急停止
STOP_USER	9	由用户停止
STOP_GAN_L1	10	当 E-Gear 龙门保护等级 1 条件满足时停止
STOP_GAN_L2	11	当 E-Gear 龙门保护等级 2 条件满足时停止
STOP_GEAR_SLAVE	12	由于齿轮从动轴而停止
STOP_ERROR_LEVEL	13	错误位置检查级别
STOP_DI	14	数字输入
STOP_ERROR_ECAT_HOME	15	EtherCAT 归零错误时停止

注意：如果运动是多轴运动（插补），则停止代码仅在参考轴上更新。其他轴保留先前的停止代码。

句法:

C/C++:

```
I32 FNTYPE APS_get_stop_code( I32 Axis_ID, I32 *Code );
```

Visual Basic:

```
APS_get_stop_code(ByVal Axis_ID As Long, Code As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 * Code: 停止代码（停止原因）。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 Code;  
I32 Axis_ID;
```

```
APS_get_stop_code(Axis_ID, &Code ); //获取停止代码  
...//
```

还可以看看:

APS_get_encoder	获取原始编码器计数器
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于获取一个轴的原始编码器计数器。 它是只读的，适用于调试或监视原始计数器。

句法:

C/C++:

```
I32 FNTYPE APS_get_encoder( I32 Axis_ID, I32 *Encoder );
```

Visual Basic:

```
APS_get_encoder(ByVal Axis_ID As Long, Encoder As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Encoder: 原始编码器计数器。单位脉冲。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Encoder;  
APS_get_encoder(Axis_ID, &Encoder ); //获取原始编码器计数器。  
...//
```

还可以看看:

APS_get_command_counter	获取原始的命令计数器
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于获取一个轴的原始命令计数器。它是只读的，适用于调试或监视原始计数器。

句法:

C/C++:

```
I32 FNTYPE APS_get_command_counter( I32 Axis_ID, I32 *Counter);
```

Visual Basic:

```
APS_get_command_counter(ByVal Axis_ID As Long, Counter As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Counter: 原始命令计数器。单位脉冲。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
I32 Counter;
```

```
APS_get_command_counter(Axis_ID, & Counter ); //获取原始的命令计数器  
...//
```

还可以看看:

APS_reset_command_counter	重置原始的命令计数器
---------------------------	------------

支持的产品:PCIe-833x

描述:

此函数用于重置一个轴的原始命令计数器（将原始编码器计数器复制到原始命令计数器）。

句法:

C/C++:

I32 APS_reset_command_counter(I32 Axis_ID);

Visual Basic:

APS_reset_command_counter(ByVal Axis_ID As Long) As Long

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

I32 Counter;

```
APS_reset_command_counter(Axis_ID, & Counter ); //重置原始的命令计数器  
...//
```

还可以看看:

APS_get_actual_torque	获取实际扭矩值
支持的产品:PCIe-833x	

描述:

此函数用于从设备获取实际扭矩值。

句法:

C/C++:

```
I32 FNTYPE APS_get_actual_torque( I32 Axis_ID, I32 *Torque )
```

Visual Basic:

```
APS_get_actual_torque(ByVal Axis_ID As Integer, ByRef Torque As Integer) As Integer
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Torque: 来自设备的实际扭矩值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_get_axis_latch_data	获取 ORG/EZ 锁存器数据
-------------------------	-----------------

描述:

用户可以使用此功能从属于轴的输入信号中获取锁存数据，例如 ORG 和 EZ。该函数获取锁存数据的行为将被清除，这意味着当用户调用此函数时，将检索锁存的数据，并且下一个锁存器的锁存计数器将清零。当用户知道有数据锁存时，用户可以调用此函数以从电机编码器中获取最新的锁存数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_axis_latch_data(I32 Axis_ID, I32 latch_channel, I32 *latch_data)
```

Visual Basic:

```
APS_get_axis_latch_data( ByVal Axis_ID As Long, ByVal latch_data As Long, latch_data  
As Long) As Long
```

参数:

I32 Axis_ID: 一个系统中从 0 到最大轴编号的轴 ID。

I32 latch_channel: 用于通道选择：0 为 ORG，1 为 EZ

I32 *latch_data: 一个数据指针，用来获取电机编码器锁存数据

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

06. 单轴运动

APS_relative_move	开始一个相对距离的运动
-------------------	-------------

支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C

描述:

该函数用于启动单轴相对运动。虽然在函数参数中设置了最大速度，但是，由于用户设置为达到最大速度，因此运动距离和加速率可能会不足。速度曲线的加减速率和曲线通过轴参数函数进行设置。

此函数为“发后即忘”的方式。这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查功能或中断事件等待函数来等待它完成。

对于 PCI-8253/56, PCI-8392(H), PCI-8254/58 / AMP-204/8C，用户可以在轴运动期间启动一个新的运动命令，包括停止命令，以覆盖前一个命令。轴将根据目标位置和新的速度等新设置，立即切换到新命令。

其他运动模式（如点动，归零，手动脉冲生成，轮廓运动）不能替代此命令。用户必须停止轴运动才能切换到上述那些模式。

对于 EMX-100，此函数用于使用相对位置的单轴点对点运动。用户可以给出两个速度曲线参数距离和最大速度，其他参数（如起始速度，加速度，减速度和 S-factor）可以通过轴参数表进行配置。实际的命令速度可能会由于行进距离小或加速率而无法达到最大速度。此函数使用“发后即忘”的模式，以避免在轴行进过程中阻塞用户的程序或过程。用户可以读取运动状态 MDN 以检查运动是否已完成（MDN = 1）（MDN = 0）。除停止命令外，用户无法在上一个动作完成之前启动的任何新的运动命令。

句法:

C/C++:

```
I32 FNTYPE APS_relative_move( I32 Axis_ID, I32 Distance, I32 Max_Speed );
```

Visual Basic:

```
APS_relative_move (ByVal Axis_ID As Long, ByVal Distance As Long, ByVal Max_Speed As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Distance: 相对距离，以脉冲为单位。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

对于 EMX-100: I32 Max_Speed: 此运动曲线的最大速度，范围为 1~8,000,000 (单位：脉冲/秒)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); //设置加速度  
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); //设置减速度  
//执行一个相对运动。  
APS_relative_move( Axis_ID, 10000, 10000 );
```

还可以看看:

```
APS_relative_move();APS_absolute_move();APS_velocity_move();APS_home_move();  
APS_stop_move(); APS_emg_stop()
```

APS_absolute_move	开始一个绝对位置的运动
支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C	

描述:

该函数用于启动一个单轴的绝对位置运动。虽然在函数参数中设置了最大速度，但是，由于用户设置为达到最大速度，因此运动距离和加速率可能会不足。速度曲线的加减速率和曲线通过轴参数函数进行设置。

此函数为“发后即忘”的方式。这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查功能或中断事件等待函数来等待它完成。

对于 PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C，用户可以在轴运动期间启动一个新的运动命令，包括停止命令，以覆盖前一个命令。轴将根据目标位置和新的速度等新设置，立即切换到新命令。其他运动模式（如点动，归零，手动脉冲生成，轮廓运动）不能替代此命令。用户必须停止轴运动才能切换到上述那些模式。

对于 EMX-100，此函数用于使用绝对位置的单轴点对点运动。用户可以给出两个速度曲线参数距离和最大速度，其他参数（如起始速度，加速度，减速度和 S-factor）可以通过轴参数表进行配置。实际的命令速度可能会由于行进距离小或加速率而无法达到最大速度。此函数使用“发后即忘”的模式，以避免在轴行进过程中阻塞用户的程序或过程。用户可以读取运动状态 MDN 以检查运动是否已完成（MDN = 1）（MDN = 0）。除停止命令外，用户无法在上一个动作完成之前启动的任何新的运动命令。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_move( I32 Axis_ID, I32 Position, I32 Max_Speed );
```

Visual Basic:

```
APS_absolute_move (ByVal Axis_ID As Long, ByVal Position As Long, ByVal Max_Speed  
As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Position: 绝对命令位置，以脉冲为单位。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

对于 EMX-100: I32 Max_Speed: 此运动曲线的最大速度，范围为 1~8,000,000 (单位：脉冲/秒)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); //设置加速度  
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); //设置减速度  
//执行一个绝对运动。  
APS_absolute_move( Axis_ID, 10000, 10000 );
```

还可以看看：

```
APS_relative_move();APS_absolute_move();APS_home_move();APS_stop_move();  
APS_emg_stop()
```

APS_velocity_move	开始一个速度运动
支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/ PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C	

描述:

该函数用于启动一个速度运动。用户发出停止运动的命令时，轴将停止。速度曲线的加减速率和曲线通过轴参数函数进行设置。

此函数为“发后即忘”的方式。这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查功能或中断事件等待函数来等待它完成。

对于 PCI-8253/56, PCI-8392(H), PCI-8254/58 / AMP-204/8C，用户可以在轴运动期间启动一个新的运动命令，包括停止命令，以覆盖前一个命令。轴将根据目标位置和新的速度等新设置，立即切换到新命令。其他运动模式（如点动，归零，手动脉冲生成，轮廓运动）不能替代此命令。用户必须停止轴运动才能切换到上述那些模式。

速度运动是一种定位控制。控制器将尝试使反馈位置赶上命令位置。这意味着，如果轴停止了，则由于控制器处于位置闭环模式，因此控制器将控制轴的位置以发出命令。

对于 EMX-100，该函数用于单轴速度移动。速度曲线参数的最大速度由用户指定，其他参数（如启动速度，加速度，减速度和 S-factor）可通过轴参数表进行配置。轴将首先达到最大速度并连续移动（MDN = 0），直到发出停止移动命令（MDN = 1）。

句法:

C/C++:

```
I32 FNTYPE APS_velocity_move( I32 Axis_ID, I32 Max_Speed );
```

Visual Basic:

```
APS_velocity_move (ByVal Axis_ID As Long, ByVal Max_Speed As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

对于 EMX-100: I32 Max_Speed: 此运动曲线的最大速度，范围为 1~8,000,000 (单位：脉冲/秒)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000); //设置加速度
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000); //设置减速度
APS_velocity_move(Axis_ID, Max_Speed); //开始速度运动
```

...

```
APS_stop_move(Axis_ID); //停止速度运动
```

还可以看看：

```
APS_relative_move(); APS_absolute_move(); APS_velocity_move(); APS_home_move();
APS_stop_move(); APS_emg_stop()
```

APS_home_move	开始一个归零运动
支持的产品:PCI-8253/56, PCI-8392(H), PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于启动轴的原点 (ORG 或 DOG) 位置。透过轴参数设置程序 , 可以选择几种模式。设置完成后 , 将根据原点的物理位置来更新轴的位置。
此功能为 “发后即忘” 的方式。这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查功能或中断事件等待功能来等待它完成。
用户无需编写归零序列即可完成返回原点的运动。所有序列都在没有 CPU 资源的情况下在板卡内进行控制。

注意:

1. 归零参数取决于产品的类型。请参考下面的 “轴参数表” 。
2. 有些产品没有 “Home ACC” , “Home VS” 和 “Home Curve” 参数;他们分别由 “PRA_ACC” , “PRA_VS” 和 “PRA_CURVE” 决定。请参考下面的 “轴参数表” 。

句法:

C/C++:

```
I32 FNTYPE APS_home_move( I32 Axis_ID );
```

Visual Basic:

```
APS_home_move (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI-8253/6

//设置归零参数

```
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 ); //设置归零模式
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); //设置归零方向
APS_set_axis_param( Axis_ID, PRA_HOME_CURVE, 0 ); //设置加速度曲线 ( T 曲线 )
APS_set_axis_param( Axis_ID, PRA_HOME_ACC, 1000000 ); //设置归零加速率
APS_set_axis_param( Axis_ID, PRA_HOME_VS, 0 ); //设置归零起始速度
APS_set_axis_param( Axis_ID, PRA_HOME_VM, 2000000 ); //设置归零最大速度。
APS_set_axis_param( Axis_ID, PRA_HOME_VO, 200000 ); //设置归零
```

```
APS_home_move(Axis_ID); //开始归零  
...//检查归零完成 ( 动作完成 )
```

示例 2:

以下示例适用于 MNET-4XMO, MNET-4XMO-C 和 PCI(e)-8154/8

//设置归零参数

```
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 ); //设置归零模式  
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); //设置归零方向  
APS_set_axis_param( Axis_ID, PRA_CURVE, 0 ); //设置加速度曲线 ( T 曲线 )  
APS_set_axis_param( Axis_ID, PRA_ACC, 1000000 ); //设置归零加速度  
APS_set_axis_param( Axis_ID, PRA_VS, 0 ); //设置归零起始速度. *1  
APS_set_axis_param( Axis_ID, PRA_HOME_VM, 2000000 ); //设置归零最大速度。  
APS_set_axis_param( Axis_ID, PRA_HOME_VO, 200000 ); //设置归零 FA 速度。 *1
```

```
APS_home_move(Axis_ID); //开始归零  
...//检查归零完成 ( 动作完成 )
```

示例 3:

以下示例适用于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x

//设置归零参数

```
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 ); //设置归零模式  
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); //设置归零方向  
APS_set_axis_param_f( Axis_ID, PRA_HOME_CURVE, 0.5 ); //设置 S-factor 为 0.5  
APS_set_axis_param_f( Axis_ID, PRA_HOME_ACC, 100000.0 ); //设置归零加速度  
APS_set_axis_param_f( Axis_ID, PRA_HOME_VM, 2000000.0 ); //设置归零最大速度。  
APS_set_axis_param_f( Axis_ID, PRA_HOME_VO, 200000.0 ); //设置归零离开原点的速度
```

```
APS_home_move(Axis_ID); //开始归零  
...//检查归零完成 ( 动作完成 )
```

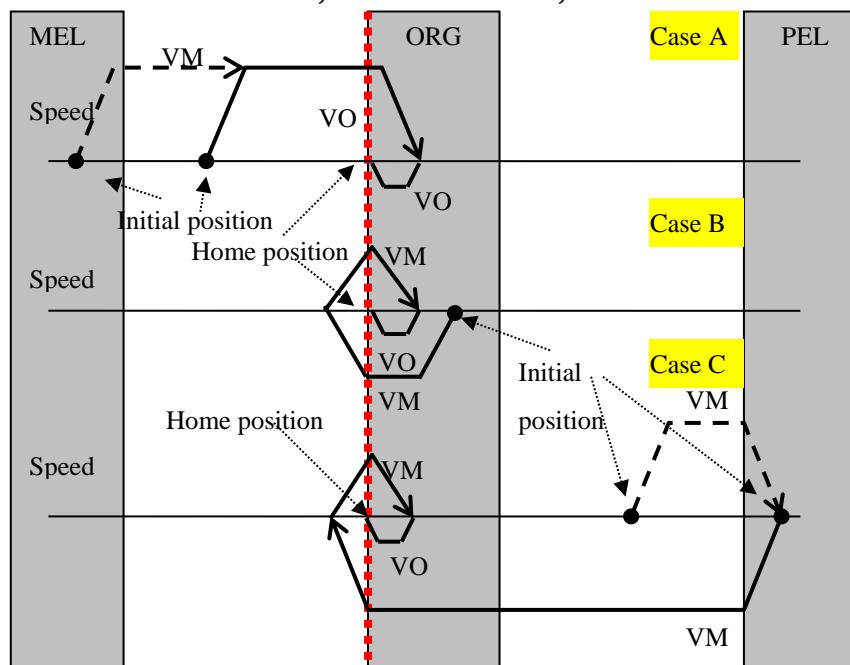
还可以看看:

APS_set_axis_param(); APS_get_axis_param(); APS_stop_move(); APS_emg_stop()

*1: 此值必须小于 PRA_HOME_VM。

归零模式方案 :

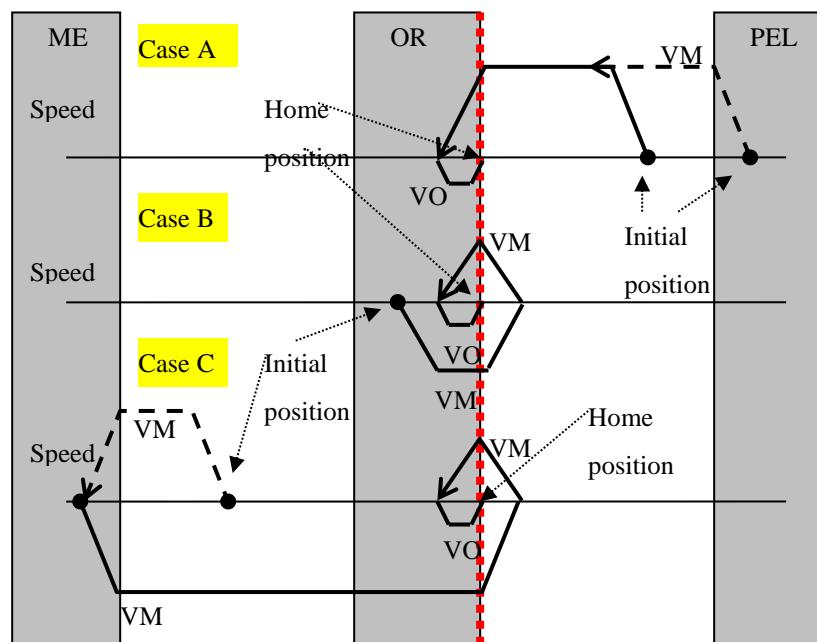
Home mode 0, Positive, EZA disable



VM : Maximum velocity VO : Homing velocity

图 2, 归零模式 0 (ORG), 正向, 禁用 EZA

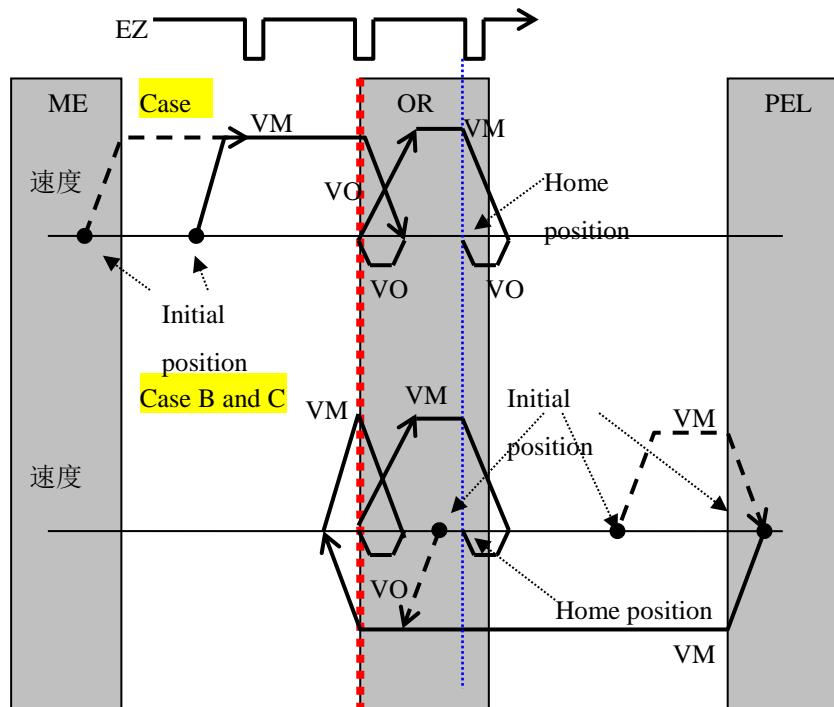
Home mode 0, Negative, EZA disable



VM : Maximum velocity VO : Homing velocity

图 3 归零模式 0 (ORG), 负向, EZA 禁用

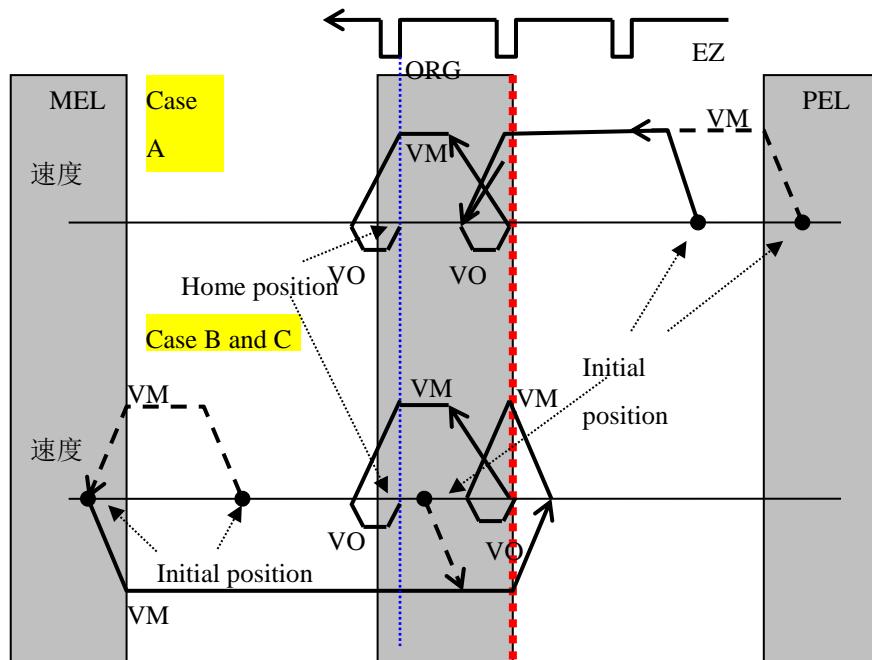
Home mode 0, Positive, EZA enable



VM : Maximum velocity VO : Homing velocity

图 4 归零模式 0 (ORG) , 正向 , 启用 EZA , 禁用 EZ_DIR

Home mode 0, Negative, EZA enable



VM : Maximum velocity VO : Homing velocity

图 5 归零模式 0 (ORG) , 负方向 , 启用 EZA , 禁用 EZ_DIR

Home mode 0, Positive, EZA enable, EZDIR = 1

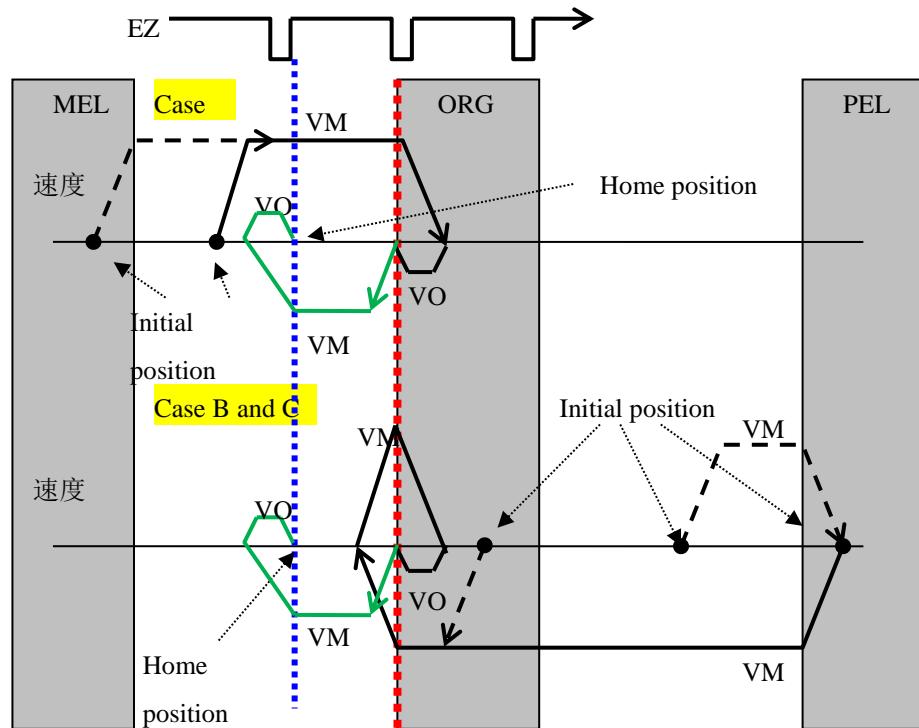
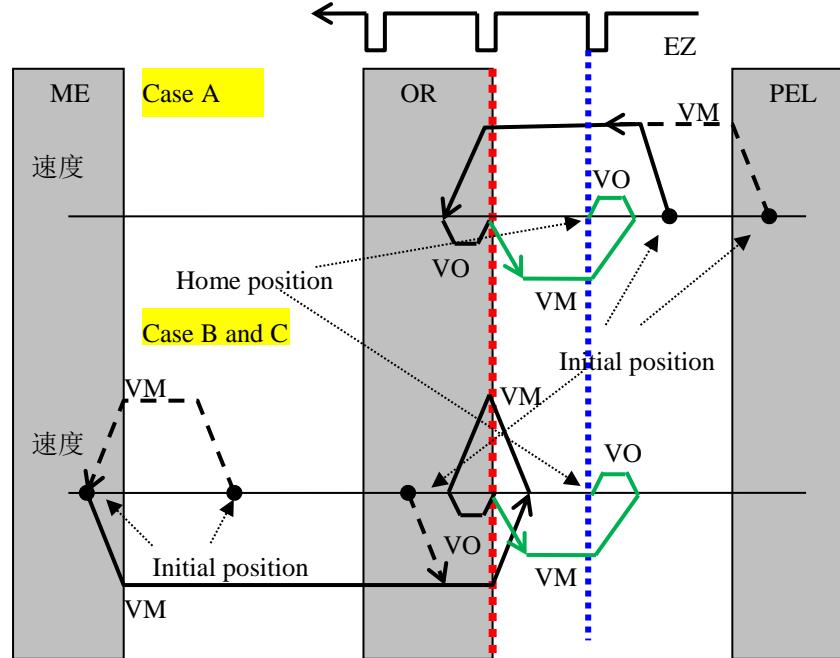


图 6 归零模式 0 (ORG) , 负方向 , 启用 EZA , 启用 EZ_DIR

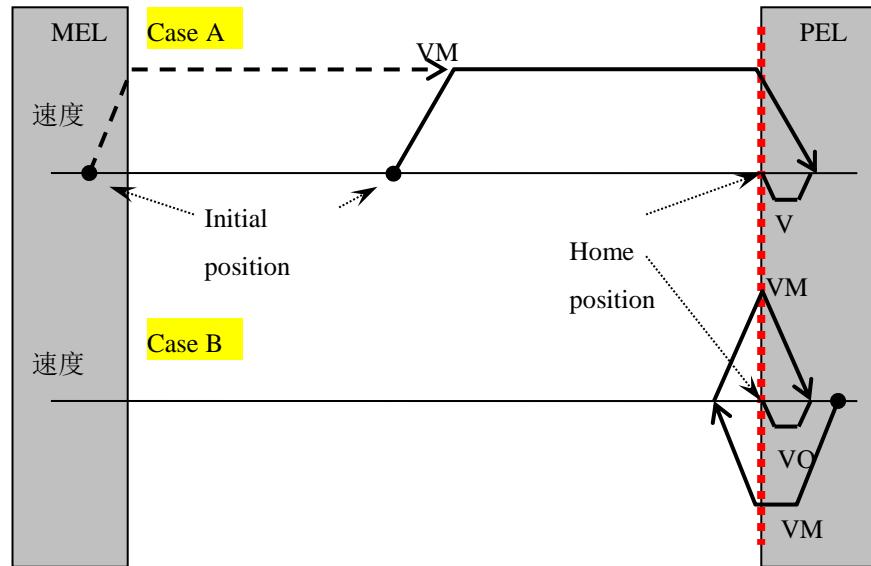
Home mode 0, Negative, EZA enable, EZDIR = 1



VM : Maximum velocity VO : Homing velocity

图 7 归零模式 0 (ORG) , 负方向 , 启用 EZA , 启用 EZ_DIR

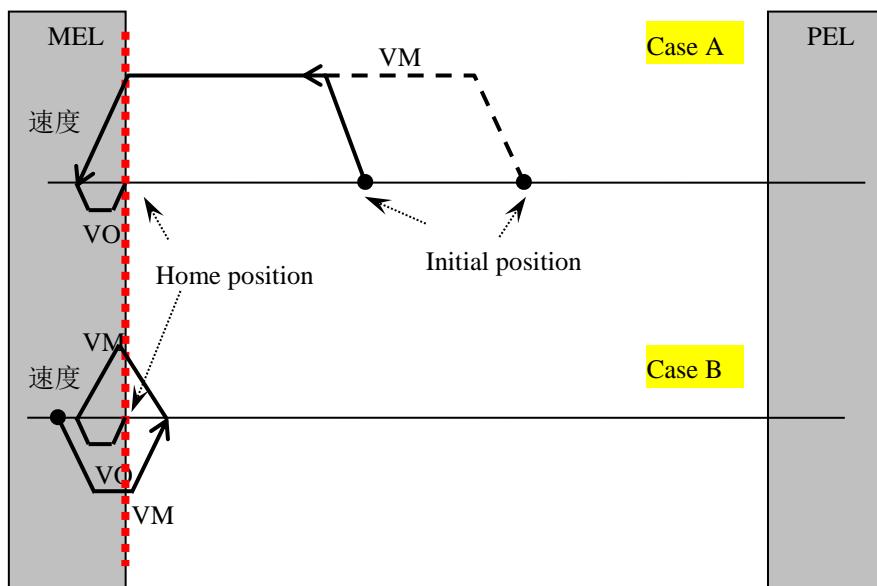
Home mode 1, Positive, EZA disable



VM : Maximum velocity VO : Homing velocity

图 8 归零模式 1 (EL) , 正方向 , EZA 禁用

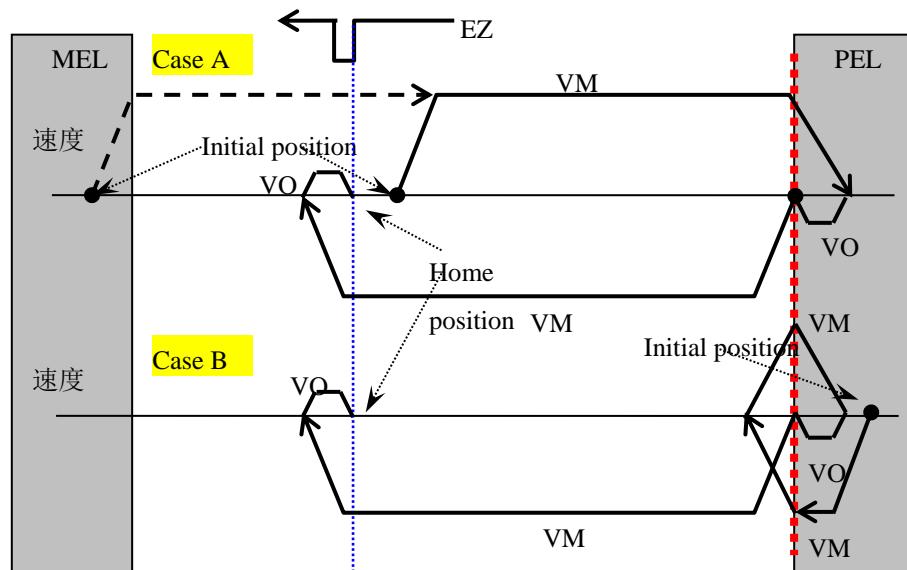
Home mode 1, Negative, EZA disable



VM : Maximum velocity VO : Homing velocity

图 9 归零模式 1 (EL) , 负方向 , EZA 禁用

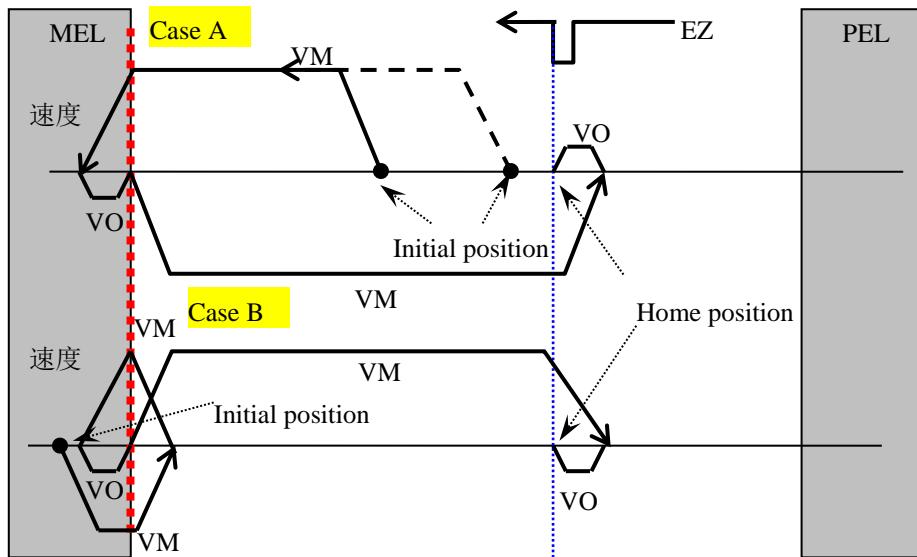
Home mode 1, Positive, EZA enable



VM : Maximum velocity VO : Homing velocity

图 10 归零模式 1 (EL) , 正方向 , 启用 EZA

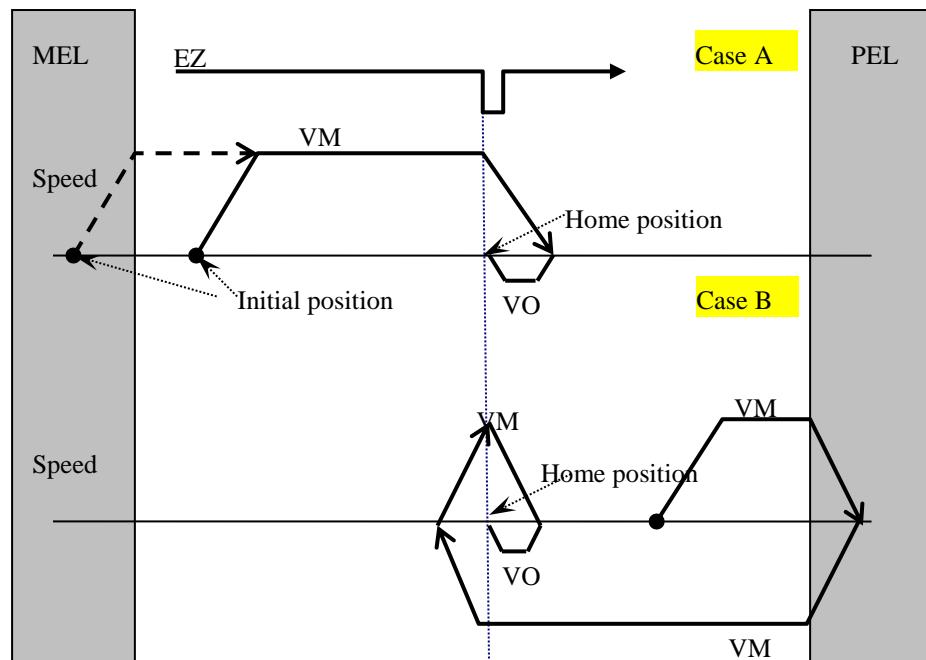
Home mode 1, Negative, EZA enable



VM : Maximum velocity VO : Homing velocity

图 11 归零模式 1 (EZ) , 负方向 , 启用 EZA

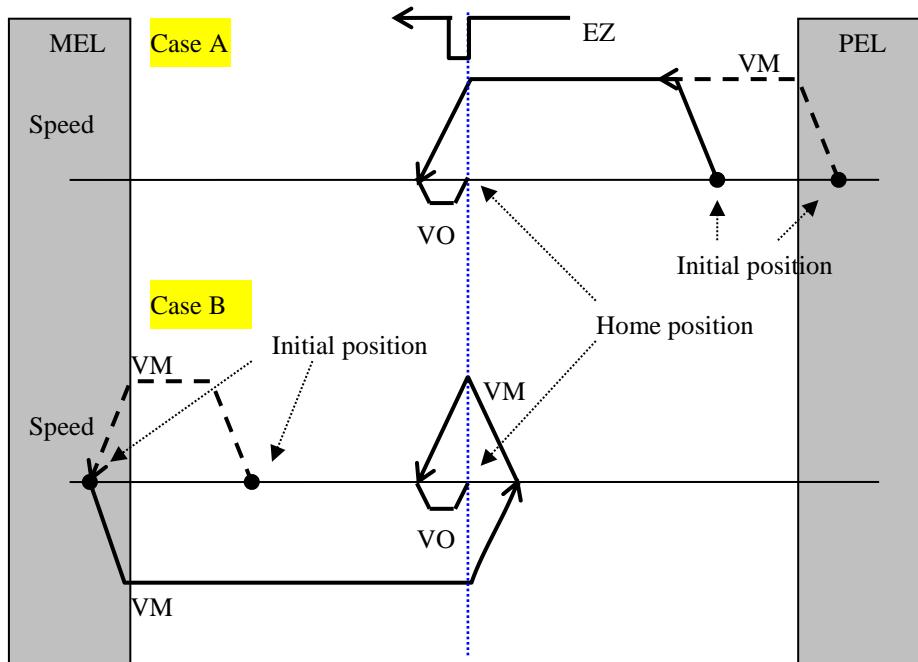
Home mode 2, Positive



VM : Maximum velocity VO : Homing velocity

图 12 归零模式 2 (EZ) , 位置方向

Home mode 2, Negative



VM : Maximum velocity VO : Homing velocity

图 13 归零模式 2 (EZ) , 负方向

Home mode 3, Positive

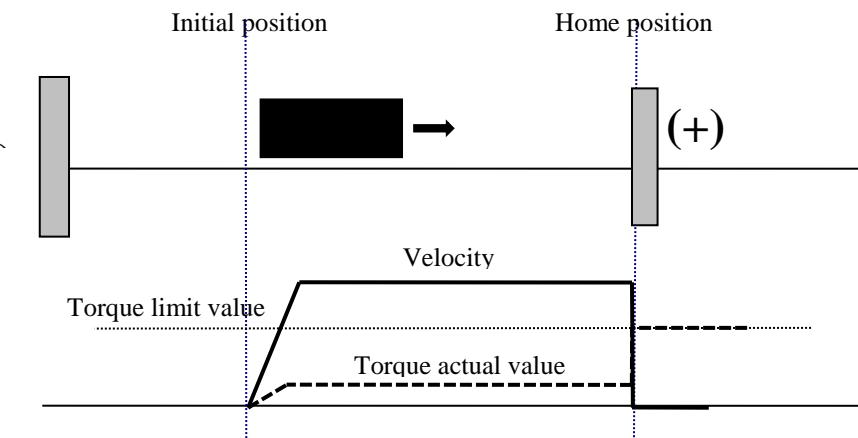


图 14 归零模式 3 (转矩) , 正方向

Home mode 3, Negative

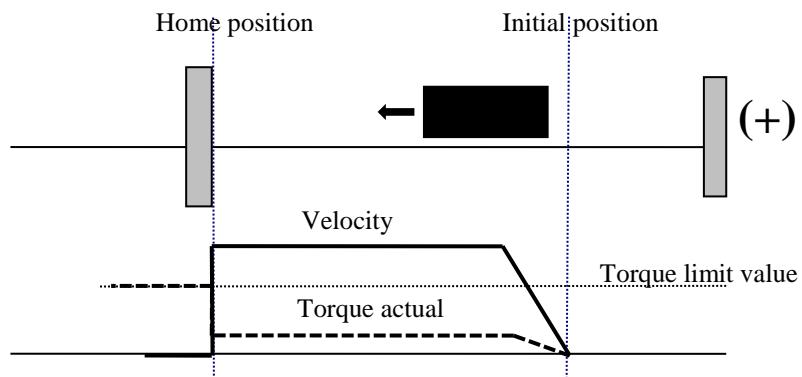
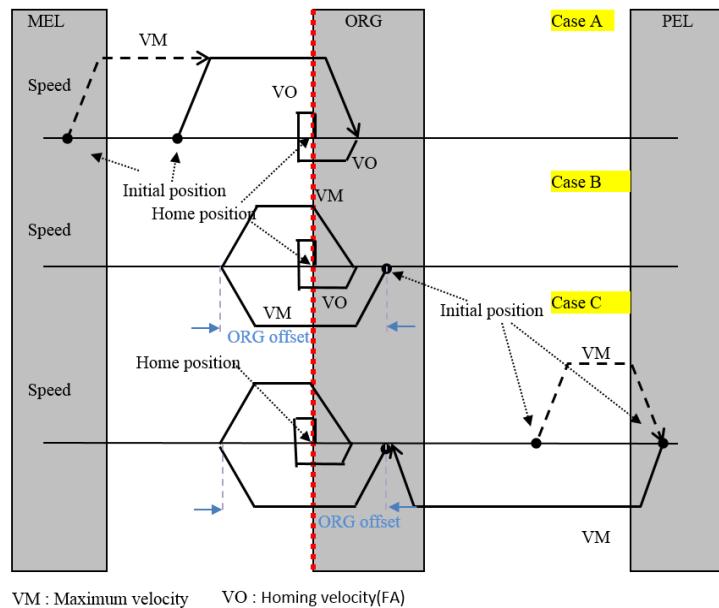


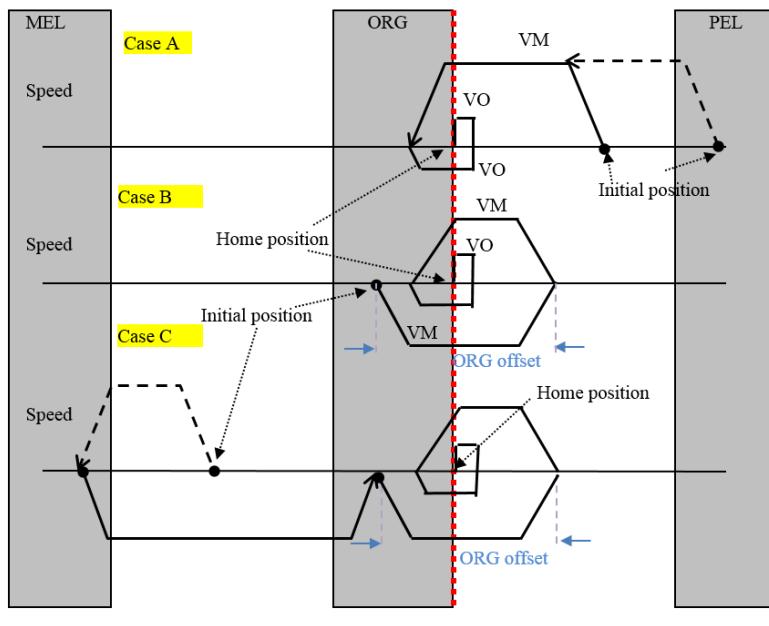
图 15 归零模式 3 (转矩), 负方向

**Home mode 4, Positive
ORG-> immediately stop**



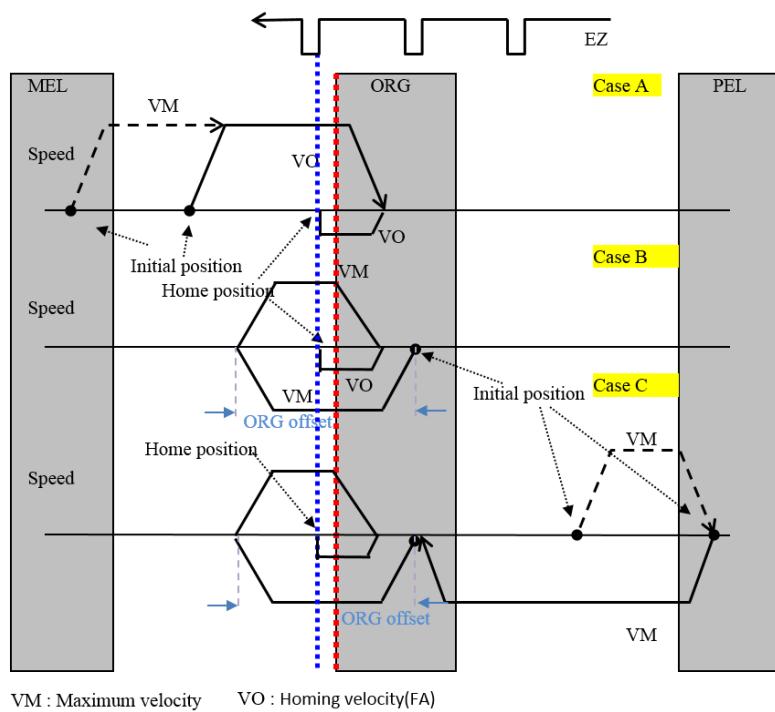
归零模式 4 (ORG , 立即停止) , 正方向

**Home mode 4, Negative
ORG-> immediately stop**



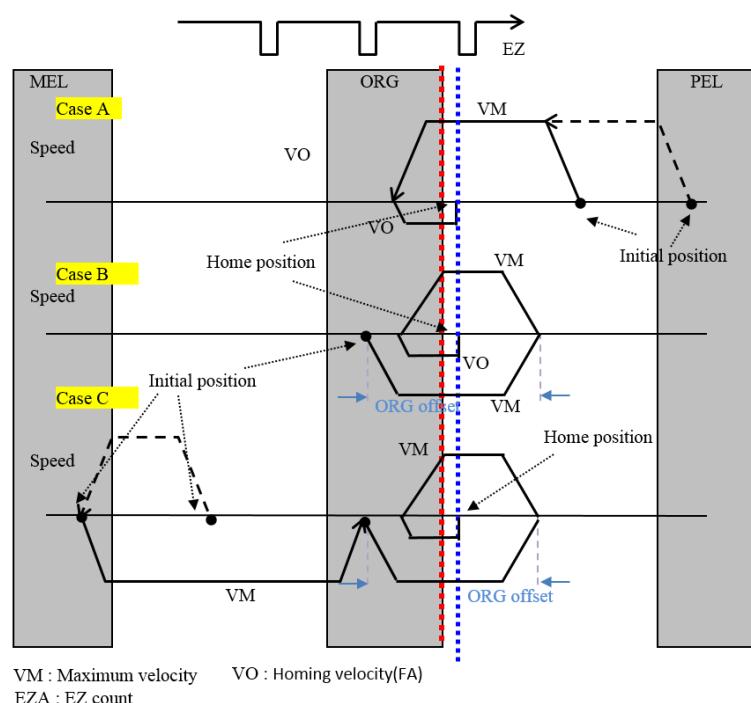
归零模式 4 (ORG , 立即停止) , 负方向

Home mode 5, Positive
ORG-> EZ -> immediately stop



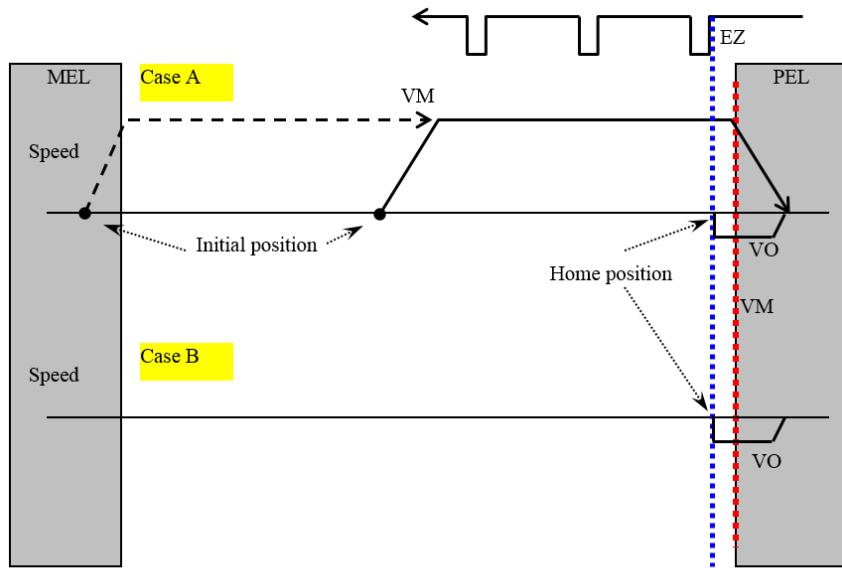
归零模式 5 (ORG+EZ , 立即停止) , 正方向

Home mode 5, Negative
ORG-> EZ -> immediately stop



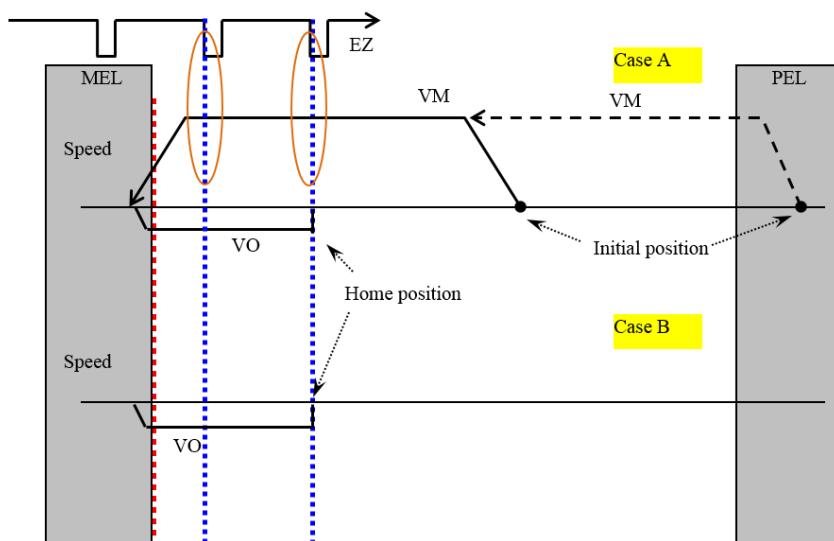
归零模式 5 (ORG+EZ , 立即停止) , 负方向

Home mode 6, Positive
EL-> Reverse -> EZ-> immediately stop



归零模式 6 (EL+EZ , 立即停止) , 正方向

Home mode 6, Negative
EL-> Reverse -> EZ-> immediately stop



归零模式 6 (EL+EZ , 立即停止) , 负方向

APS_stop_move	停止运动
---------------	------

支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于立即停止单轴或多轴运动。它可以停止单轴归零、定位和速度运动。当用户放置与插补运动有关的轴 ID 之一时，它也可以停止多轴插补运动。减速曲线与通常的减速设定不同，需通过轴参数函数进行设定。减速参数不同于正常运动的曲线，可以单独进行设置。
停止功能不能被其他功能取代。

句法:

C/C++:

```
I32 FNTYPE APS_stop_move(I32 Axis_ID);
```

Visual Basic:

```
APS_stop_move (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
// APS_absolute_move(Axis_ID, Position, Max_Speed );
// APS_home_move(Axis_ID ); //归零运动
...
APS_stop_move(Axis_ID); //停止运动
```

还可以看看:

[APS_emg_stop\(\)](#)

APS_emg_stop	紧急停止
--------------	------

支持的产品:PCI-8253/56, PCI-8392(H) , PCI-8144, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于立即停止单轴或多轴运动。它可以停止单轴归零、定位和速度运动。当用户放置与插补运动有关的轴 ID 之一时，它也可以停止多轴插补运动。由于停止功能会意外停止轴，因此如果设置了中断因素，它将产生异常的停止中断事件，而不是正常的停止事件。运动状态也将设置为异常停止状态。下一个运动命令将清除异常停止状态或事件。该函数没有减速曲线。

句法:

C/C++:

```
I32 FNTYPE APS_emg_stop(I32 Axis_ID);
```

Visual Basic:

```
APS_emg_stop (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
// APS_absolute_move(Axis_ID, Position, Max_Speed );
// APS_home_move(Axis_ID ); //归零运动
...
APS_emg_stop (Axis_ID); //紧急停止
```

还可以看看:

[APS_stop_move\(\)](#)

APS_relative_move2	根据速度曲线开始一个相对距离的运动
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于开始一个相对距离的运动。该函数与 “ APS_relative_move()” 函数的功能相似。这两个函数之间的区别在于，该函数在一个系统周期内是依据速度曲线进行发布。系统周期就是与主机 PC 控制器之间的握手时间。

句法:

C/C++:

```
I32 FNTYPE APS_relative_move2( I32 Axis_ID, I32 Distance, I32 Start_Speed, I32
Max_Speed, I32 End_Speed, I32 Acc_Rate, I32 Dec_Rate );
```

Visual Basic:

```
APS_relative_move2( ByVal Axis_ID As Long, ByVal Distance As Long, ByVal
Start_Speed As Long, ByVal Max_Speed As Long, ByVal End_Speed As Long, ByVal
Acc_Rate As Long, ByVal Dec_Rate As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Distance: 相对距离，以脉冲为单位。

I32 Start_Speed: 该运动曲线的开始速度，单位:脉冲/秒。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

I32 End_Speed: 该运动曲线的终止速度，单位:脉冲/秒。

I32 Acc_Rate: 加速率. Pulse/(sec²)

I32 Dec_Rate: 减速率. Pulse/(sec²)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_relative_move\(\)](#)

APS_absolute_move2	通过速度曲线开始绝对位置运动
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该用于开始绝对位置移动。 该函数与 “ APS_absolute_move() ” 函数相似。 这两个函数之间的区别在于，该函数通过速度曲线参数调用，并且此函数仅需一个系统周期即可传递参数。 系统周期意味着与主机 PC 控制器的握手时间。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_move2( I32 Axis_ID, I32 Position, I32 Start_Speed, I32
Max_Speed, I32 End_Speed, I32 Acc_Rate, I32 Dec_Rate );
```

Visual Basic:

```
APS_absolute_move2( ByVal Axis_ID As Long, ByVal Position As Long, ByVal
Start_Speed As Long, ByVal Max_Speed As Long, ByVal End_Speed As Long, ByVal
Acc_Rate As Long, I32 Dec_Rate As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Position: 绝对位置。 单位: 脉冲

I32 Start_Speed: 该运动曲线的开始速度，单位:脉冲/秒。

I32 Max_Speed: 此运动曲线的最大速度。 单位：脉冲/秒。

I32 End_Speed: 该运动曲线的终止速度，单位:脉冲/秒。

I32 Acc_Rate: 加速度。 脉冲/秒²

I32 Dec_Rate: 减速度。 脉冲/秒²

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_absolute_move\(\)](#)

APS_home_move2	根据速度曲线开始归零运动
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于开始一个归零运动操作。该函数与“APS_home_move()”函数的功能相似。这两个函数之间的区别在于，该函数通过速度曲线参数调用，并且该函数仅需一个系统周期即可传递参数。系统周期就是与主机PC控制器之间的握手时间。

句法:

C/C++:

```
I32 FNTYPE APS_home_move2( I32 Axis_ID, I32 Dir, I32 Acc, I32 Start_Speed, I32  
Max_Speed, I32 ORG_Speed );
```

Visual Basic:

```
APS_home_move2( ByVal Axis_ID As Long, ByVal Dir As Long, ByVal Acc As Long,  
ByVal Start_Speed As Long, ByVal Max_Speed As Long, ByVal ORG_Speed As Long) As  
Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Dir: 归零方向.

0: 正方向 (默认)

1: 反方向

I32 Acc: 归零加速/减速率。 单位：脉冲/秒²

I32 Start_Speed: 归零运动的开始速度。单位脉冲/秒

I32 Max_Speed: 归零运动的最大速度。单位：脉冲/秒.

I32 ORG_Speed: 归零运动离开原点的速度。 单位：脉冲/秒

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_home_move\(\)](#)

APS_speed_override	改变在线速度
--------------------	--------

支持的产品 : MNET-1XMO, MNET-4XMO, MNET-4XMO-C, PCI(e)-8154/58, PCI-C154(+)

描述 :

在轴运动期间 , 用户可以更改新的运动速度以覆盖先前的运动。轴将立即切换到新速度。

注意 : 如果原始距离不足以覆盖新速度 , 它将返回 ERR_DistantNotEnough.

注意 : 如果新速度与当前运动速度相同 , 它将返回 ERR_ParameterInvalid.

句法:

C/C++:

```
I32 FNTYPE APS_speed_override( I32 Axis_ID, I32 Max_Speed );
```

Visual Basic:

```
APS_speed_override (ByVal Axis_ID As Long, ByVal Max_Speed As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Max_Speed: 用最大的速度覆盖先前的运动。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Distance;
```

```
I32 Max_Speed;
```

```
I32 New_Speed;
```

```
I32 ret;
```

```
APS_relative_move(Axis_ID, Distance, Max_Speed ); //开始相对运动。
```

```
//速度覆盖。
```

```
ret = APS_speed_override(Axis_ID, New_Speed ); //更改为新速度。
```

```
...
```

还可以看看:

APS_relative_move_ovrd	开始一个相对距离运动，或以新的距离和速度覆盖它。
------------------------	--------------------------

支持的产品 : MNET-1XMO, MNET-4XMO, MNET-4XMO-C, PCI(e)-8154/58, PCI-C154(+)

描述 :

开始一个相对距离 :

该函数用于启动一个单轴相对运动。 尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。速度曲线的加减速率和曲线可通过轴参数函数进行设置。

在轴运动期间进行覆盖 :

在轴运动期间，用户可以启动一个新的运动命令以覆盖前一个命令。轴将根据新距离、新速度的新设置立即切换到新命令。 **请注意，如果新距离不足以覆盖新速度，它将返回ERR_DistantNotEnough。请注意，无论轴参数PRA_FEEDBACK_SRC的设置如何，覆盖时新的距离都会参考命令计数器。**

句法:

C/C++:

```
I32 FNTYPE APS_relative_move_ovrd ( I32 Axis_ID, I32 Distance, I32 Max_Speed );
```

Visual Basic:

```
APS_relative_move_ovrd (ByVal Axis_ID As Long, ByVal Distance As Long , ByVal  
Max_Speed As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Distance: 相对距离，以脉冲为单位。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Distance;
```

```
I32 Max_Speed;
```

```
I32 New_Distance:
```

```
I32 New_Speed;
```

```
I32 ret;
```

```
// 开始一段相对距离。
```

```
Ret = APS_relative_move_ovrd(Axis_ID, Distance, Max_Speed );  
// 当轴运动期间进行覆盖。  
ret = APS_relative_move_ovrd(Axis_ID, New_Distance , New_Speed );  
...
```

还可以看看：

APS_absolute_move_ovrd	开始一个绝对位置运动，或以新的距离和速度覆盖它。
------------------------	--------------------------

支持的产品 : MNET-1XMO, MNET-4XMO, MNET-4XMO-C, PCI(e)-8154/58, PCI-C154(+)

描述 :

开始一个绝对位置运动 :

该函数用于启动一个单轴的绝对位置运动。尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。速度曲线的加减速率和曲线可通过轴参数函数进行设置。

在轴运动期间进行覆盖 :

在轴运动期间，用户可以启动一个新的运动命令以覆盖前一个命令。轴将根据新距离、新速度的新设置立即切换到新命令。

请注意，如果新距离不足以覆盖新速度，它将返回ERR_DistantNotEnough。

请注意，无论轴参数PRA_FEEDBACK_SRC的设置如何，覆盖时新的距离都会参考命令计数器。

句法:

C/C++:

I32 FNTYPE APS_absolute_move_ovrd (I32 Axis_ID, I32 Position, I32 Max_Speed);

Visual Basic:

APS_absolute_move_ovrd (ByVal Axis_ID As Long, ByVal Position As Long , ByVal Max_Speed As Long) As Long

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Position: 绝对位置，以脉冲为单位。

I32 Max_Speed: 此运动曲线的最大速度。单位：脉冲/秒。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

I32 Position;

I32 Max_Speed;

I32 New_Position:

I32 New_Speed;

I32 ret;

```
// 开始一个绝对位置运动。  
Ret = APS_absolute_move_ovrd (Axis_ID, Position, Max_Speed );  
// 当轴运动期间进行覆盖。  
ret = APS_absolute_move_ovrd (Axis_ID, New_Position, New_Speed );  
...  
...
```

还可以看看:

APS_home_escape	离开原点开关
支持的产品:MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+)	

描述:

此函数用于离开原点 (ORG) 位置。

注意:

1. 归零参数取决于产品的类型。请参考下面的“轴参数表”。
2. 有些产品没有 “Home ACC” , “Home VS” 和 “Home Curve” 参数;他们分别由 “PRA_ACC” , “PRA_VS” 和 “PRA_CURVE” 决定。请参考下面的“轴参数表”。

句法:

C/C++:

```
I32 FNTYPE APS_home_escape( I32 Axis_ID );
```

Visual Basic:

```
APS_home_escape (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//设置归零 parameters
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); //设置归零方向
APS_set_axis_param( Axis_ID, PRA_HOME_CURVE, 0 ); //设置加速度曲线 ( T 曲线 )
APS_set_axis_param( Axis_ID, PRA_HOME_ACC, 10000 ); //设置归零加速度
APS_set_axis_param( Axis_ID, PRA_HOME_VS, 0 ); //设置归零起始速度
APS_set_axis_param( Axis_ID, PRA_HOME_VM, 10000 ); //设置归零最大速度。

APS_home_escape(Axis_ID ); //离开原点。
...//检查归零完成 ( 动作完成 )
```

还可以看看:

[APS_set_axis_param\(\)](#); [APS_get_axis_param\(\)](#); [APS_stop_move\(\)](#); [APS_emg_stop\(\)](#)

07. 多轴运动触发和停止

APS_move_trigger	发送一个触发以同步所有等待的运动
------------------	------------------

支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于发送一个触发以同步所有等待的运动。请参阅高级单步运动和插补章节的内容。用户可以通过调用高级运动功能，来设置选项参数的第 8 位，因此该运动将设置为等待状态。

句法:

C/C++

```
I32 FNTYPE APS_move_trigger( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_move_trigger(ByVal Dimension As Long, Axis_ID_Array As Long ) As Long
```

参数:

I32 Dimension: 同步轴的尺寸。

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 设置为 1。处于等待状态。
I32 opt = 0x0100; //绝对，等待触发，中止模式
ASYNCALL *wait = NULL;

//在等待状态下绝对运动到位置 10000
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// 发送一个触发以同步所有等待的运动
APS_move_trigger( 2, Axis_ID_Array );

...
// 停止同步运动
APS_stop_move_multi ( 2, Axis_ID_Array );
```

还可以看看:

[APS_stop_move_multi\(\)](#)

APS_stop_move_multi	停止多轴运动
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于同时停止多轴运动。一般来说，它是用来停止同步运动。通过调用 APS_set_axis_param_f() 来设置减速曲线（定义为 PRA_SD_DEC）。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_stop_move_multi ( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_stop_move_multi (ByVal Dimension As Long, Axis_ID_Array As Long) As Long
```

参数:

I32 Dimension: 停止轴的尺寸。

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 设置为 1。处于等待状态。
I32 opt = 0x0100; //绝对，等待触发，中止模式
ASYNCALL *wait = NULL;

//在等待状态下绝对运动到位置 10000
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// 发送一个触发以同步所有等待的运动
APS_move_trigger( 2, Axis_ID_Array );

...
// 停止同步运动
APS_stop_move_multi ( 2, Axis_ID_Array );
```

还可以看看:

[APS_move_trigger\(\)](#)

APS_emg_stop_multi	立即停止多轴运动
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于立即停止多轴运动。由于停止函数会意外让轴停止，因此如果设置了中断因素，它将产生异常的停止中断事件，而不是正常的停止事件。运动状态也将设置为异常停止状态。下一个运动命令将清除异常停止状态或事件。该函数没有减速曲线。

句法:

C/C++:

```
I32 FNTYPE APS_emg_stop_multi ( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_emg_stop_multi (ByVal Dimension As Long, Axis_ID_Array As Long) As Long
```

参数:

I32 Dimension: 停止轴的尺寸。

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 设置为 1。处于等待状态。
I32 opt = 0x0100; //绝对，等待触发，中止模式
ASYNDCALL *wait = NULL;

//在等待状态下绝对运动到位置 10000
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// 发送一个触发以同步所有等待的运动
APS_move_trigger( 2, Axis_ID_Array );

...
// 紧急停止同步运动
APS_emg_stop_multi ( 2, Axis_ID_Array );
```

还可以看看:

[APS_move_trigger\(\)](#)

08. 点动

APS_set_jog_param	设置点动参数
-------------------	--------

支持的产品:PCI-8253/56, PCI-8392(H)

描述:

此函数用于设置点动相关的参数。该参数在轴参数表中也可用。

句法:

C/C++:

```
I32 FNTYPE APS_set_jog_param( I32 Axis_ID, JOG_DATA *pStr_Jog, I32 Mask );
```

Visual Basic:

```
APS_set_jog_param( ByVal Axis_ID As Long, pStr_Jog As JOG_DATA, ByVal Mask As  
Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

JOG_DATA *pStr_Jog: 点动参数的结构。在 “type_def.h” 中定义
typedef struct

```
{  
    I16 i16_jogMode; // 点动模式。0:自由运行模式。1:步进模式。  
    I16 i16_dir;      // 点动方向。0: 正方向, 1:反方向  
    I16 i16_accType; // 加减速模式。0 : T 曲线 , 1 : S 曲线  
    I32 i32_acc;     // 加速度 (脉冲/秒 2 )  
    I32 i32_dec;     // 减速度 (脉冲/秒 2 )  
    I32 i32_maxSpeed; // 正值 , 最大速度。 (脉冲/秒 )  
    I32 i32_offset;   // 正值 , 步进偏移。用于步进点动模式。(脉冲)  
    I32 i32_delayTime; // 延迟时间 , 用于步进点动模式。 ( 范围 :0~65535 毫秒 , 按循环  
    时间对齐 )  
} JOG_DATA;
```

I32 Mask: 屏蔽参数设置。位格式 , 设置为 0 将被屏蔽。

屏蔽项目	屏蔽位编码
加速度 (i32_acc)	0
减速度(i32_dec)	1
最大速度 (i32_maxSpeed)	2
步进偏差 (i32_offset)	3
延迟时间 (i32_delayTime)	4
点动模式 (i16_jog mode)	5

点动方向 (i16_direction)	6
点动加/减速模型	7

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS168.h"
// 首先初始化板卡...

I32 ret;
JOG_DATA jog;

jog.i16_jogMode = 1; //Mask = 0x20
jog.i16_dir = 0; //Mask = 0x40

ret = APS_set_jog_param( Axis_ID, &jog, 0x20 | 0x40 );
if( ret != 0 ) //错误
```

还可以看看:

[APS_set_axis_param\(\)](#),[APS_get_axis_param\(\)](#),[APS_get_jog_param\(\)](#)

APS_get_jog_param	获取点动参数
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于获取点动相关的参数。

句法:

C/C++:

```
I32 FNTYPE APS_get_jog_param( I32 Axis_ID, JOG_DATA *pStr_Jog );
```

Visual Basic:

```
APS_get_jog_param( ByVal Axis_ID As Long, pStr_Jog As JOG_DATA) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

JOG_DATA *pStr_Jog: 点动参数的结构。在 “type_def.h” 中定义

```
typedef struct
```

```
{
```

```
    I16 i16_jogMode; // 点动模式。0:自由运行模式。1:步进模式。
```

```
    I16 i16_dir; // 点动方向。0: 正方向, 1:反方向
```

```
    I16 i16_accType; // 加减速模式。0 : T 曲线 , 1 : S 曲线
```

```
    I32 i32_acc; // 加速度 ( 脉冲/秒 2 )
```

```
    I32 i32_dec; // 减速度 (脉冲/秒 2 )
```

```
    I32 i32_maxSpeed; // 正值 , 最大速度。 (脉冲/秒 )
```

```
    I32 i32_offset; // 正值 , 步进偏移。用于步进点动模式。(脉冲)
```

```
    I32 i32_delayTime; // 延迟时间 , 用于步进点动模式。 ( 范围 :0~65535 毫秒 , 按循环  
时间对齐 )
```

```
} JOG_DATA;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
```

```
#include "APS168.h"
```

```
// 首先初始化板卡...
```

```
I32 ret;
```

```
JOG_DATA jog;
```

```
ret = APS_get_jog_param( Axis_ID, &jog );
```

```
if( ret != 0 ) //错误
```

还可以看看：

```
APS_set_axis_param();APS_get_axis_param();APS_set_jog_param()
```

APS_jog_mode_switch	启用/禁用点动
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于将指定轴切换为点动模式。当轴处于点动模式时，除停止命令外，它不能接受其他运动命令。

用户必须执行点动之前启动点动模式。

句法:

C/C++:

```
I32 FNTYPE APS_jog_mode_switch( I32 Axis_ID, I32 Turn_No );
```

Visual Basic:

```
APS_jog_mode_switch( ByVal Axis_ID As Long, ByVal Turn_No As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Turn_No: 0: 禁用点动模式, 1:启用电动模式。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
// 配置点动参数。
```

```
Ret = APS_jog_mode_switch(Axis_ID, 1); //打开点动模式。
// 执行点动 ... (APS_jog_start)
...
ret = APS_jog_mode_switch(Axis_ID, 0); //关闭点动模式。
// 执行其他运动命令。
```

还可以看看:

[APS_set_jog_param\(\)](#); [APS_get_jog_param\(\)](#); [APS_jog_start\(\)](#)

APS_jog_start	开始/停止点动
---------------	---------

支持的产品:PCI-8253/56, PCI-8392(H), EMX-100 , PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于开始/停止点动。在开始点动运动之前，必须使轴进入点动模式。

对于 EMX-100，此函数用于开始/停止点动运动。下面显示的轴参数用于配置点动运动的速度曲线，仅支持 s-factor= 0 (T 曲线)。

符号定义	参数编号	描述
PRA_JG_DIR	0x41	(I32) 点动方向 [0: 正方向, 1: 反方向]
PRA_JG_ACC	0x43	点动加速度
PRA_JG_VM	0x45	点动最大速度
PRA_JG_STOP	0x4C	点动停止模式

对于 PCI-8254/58/AMP-204/8C 和 PCIe-833x，下表显示的参数可用来配置点动参数：有关详细信息，请参见轴参数表。

符号定义	参数编号	描述
PRA_JG_MODE	0x40	(I32) 点动模式[0 : 连续模式， 1 : 步进模式]
PRA_JG_DIR	0x41	(I32)点动方向 [0: 正方向, 1: 反方向]
PRA_JG_SF	0x42	(F64) 点动 S factor [0 ~ 1]
PRA_JG_ACC	0x43	(F64) 点动加速度 [值 > 0]
PRA_JG_DEC	0x44	(F64) 点动减速度 [值 > 0]
PRA_JG_VM	0x45	(F64) 点动最大速度 [值 > 0]
PRA_JG_OFFSET	0x46	(F64) 点动偏移，步进模式 [值 > = 0]
PRA_JG_DELAY	0x47	(I32) 点动延迟，步进模式，微秒[0 ~ 10,000,000]
PRA_JG_MAP_DI_EN	0x48	(I32) 启用数字输入映射至点动命令信号
PRA_JG_P_JOG_DI	0x49	(I32) 正向点动和数字输入的映射配置。
PRA_JG_N_JOG_DI	0x4A	(I32) 负向点动和数字输入的映射配置。
PRA_JG_JOG_DI	0x4B	(I32) 点动和数字输入的映射配置。

句法:

C/C++:

```
I32 FNTYPE APS_jog_start( I32 Axis_ID, I32 STA_On );
```

Visual Basic:

```
APS_jog_start( ByVal Axis_ID As Long, ByVal STA_On As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。
I32 STA_On: 1:STA 信号打开, 0:STA 信号关闭

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI-8253/56, PCI-8392(H)
// 配置点动参数。

```
Ret = APS_jog_mode_switch(Axis_ID, 1 ); //打开点动模式。
```

```
// 执行点动 ... (APS_jog_start)
APS_jog_start( Axis_ID, 1 ); //STA 信号打开
...
APS_jog_start( Axis_ID, 0 ); //STA 信号关闭
```

```
ret = APS_jog_mode_switch(Axis_ID, 0 ); //关闭点动模式。
// 执行其他运动命令。
```

示例 2:

以下示例适用于 EMX-100

```
// 配置点动参数。
APS_set_axis_param( Axis_ID, PRA_JG_DIR, 1 ); //设置点动方向为负方向
APS_set_axis_param( Axis_ID, PRA_JG_ACC, 100000 ); //设置点动加速度
```

```
// 执行点动 ... (APS_jog_start)
APS_jog_start( Axis_ID, 1 ); //STA 信号打开
...
APS_jog_start( Axis_ID, 0 ); //STA 信号关闭
```

示例 3:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
// 配置点动参数。
APS_set_axis_param( Axis_ID, PRA_JG_MODE, 0 ); //设置为连续模式
APS_set_axis_param( Axis_ID, PRA_JG_DIR, 1 ); //设置点动方向为负方向
APS_set_axis_param_f( Axis_ID, PRA_JG_ACC, 100000.0 ); //设置点动加速度

// 执行点动 ... (APS_jog_start)
```

```
APS_jog_start( Axis_ID,1 ); //STA 信号打开
```

```
...
```

```
APS_jog_start(Axis_ID, 0); //STA 信号关闭
```

还可以看看:

```
APS_set_jog_param(); APS_get_jog_param(); APS_jog_mode_switch();
```

09. 插补

APS_absolute_linear_move	开始一个绝对位置的线性插补
--------------------------	---------------

支持的产品:PCI-8253/56,PCI-8392(H) , MNET-4XMO-(C),HSL-4XMO,PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C

描述:

该函数用于启动一个绝对线性插补定位的运动。尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。速度曲线的加减速率和曲线可通过轴参数函数进行设置。由于速度参数是矢量方向，因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。

在轴运行期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令。轴将根据目标位置、新速度等新设置立即切换到新的命令。

被覆盖的命令必须具有与前一个命令有相同的尺寸和轴 ID。这两个命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡。

对于 EMX-100，此函数用于使用绝对位置进行线性插补定位运动。它在同一时间同一设备上仅支持两个轴的运动。主轴是用户为插补指定的第一个轴 ID。用户可以给出两个速度曲线参数，即行进距离和最大速度，其他参数（如启动速度，加速率，减速率和 s-factor）可以通过主轴的参数表进行配置。实际的命令速度可能会由于行进距离小或加速率已给出而无法达到最大速度。

此函数使用“发后即忘”的模式，以避免在轴行进过程中阻碍用户的程序或进程。用户可以读取运动状态 MDN 以检查运动是否已完成（MDN = 1）（MDN = 0）。除停止命令外，用户无法在上一个动作完成之前启动任何新的运动命令。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_linear_move( I32 Dimension, I32 *Axis_ID_Array, I32  
*Position_Array, I32 Max_Linear_Speed );
```

Visual Basic:

```
APS_absolute_linear_move( ByVal Dimension As Long, Axis_ID_Array As Long,  
Position_Array As Long, ByVal Max_Linear_Speed As Long ) As Long
```

参数:

I32 Dimension: 插补轴的尺寸。（2~4 轴）

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

对于 EMX-100: I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。数组的第一个元素是主轴，并按升序排列。例如 Axis_ID_Array[2] = {1,3} 或 Axis_ID_Array[2] = {0,3} 或 Axis_ID_Array[2] = {0,1} 或 Axis_ID_Array[2] = {2,3}

I32 *Position_Array: 绝对位置数组。（单位：脉冲）

I32 Max_Linear_Speed: 最大线性插补速度。（单位：脉冲/秒）

对于 EMX-100: I32 Max_Linear_Speed: 最大线性插补速度；范围是 1~8,000,000（单位：脉冲/秒）

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
//...初始化板卡
I32 Dimension = 4;
I32 Master_Axis_ID = 1; //主轴
I32 Axis_ID_Array[4] = { 1, 2, 3, 4}; //轴 ID 1 是主轴.
I32 Position_Array [4] = {10000, 20000, 30000, 40000 };
I32 Max_Linear_Speed = 10000;
I32 Ret;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 0 ); //设置 T 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度
```

```
Ret = APS_absolute_linear_move ( Dimension, Axis_ID_Array, Position_Array,
Max_Linear_Speed );
```

...

还可以看看:

[APS_relative_linear_move\(\)](#)

APS_relative_linear_move	开始一个相对距离的线性插补
--------------------------	---------------

支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO-(C), HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C

描述:

该函数用于启动一个相对线性插补定位的运动。尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。速度曲线的加减速率和曲线可通过轴参数函数进行设置。由于速度参数是矢量方向，因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令。轴将根据目标位置、新速度等新设置立即切换到新的命令。

被覆盖的命令必须具有与前一个命令有相同的尺寸和轴 ID。这两个命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡。

对于 EMX-100，此函数用于使用绝对位置进行线性插补定位运动。它在同一时间同一设备上仅支持两个轴的运动。主轴是用户为插补指定的第一个轴 ID。用户可以给出两个速度曲线参数，即行进距离和最大速度，其他参数（如启动速度，加速率，减速率和 s-factor）可以通过主轴的参数表进行配置。实际的命令速度可能会由于行进距离小或加速率已给出而无法达到最大速度。

此函数使用“发后即忘”的模式，以避免在轴行进过程中阻碍用户的程序或进程。用户可以读取运动状态 MDN 以检查运动是否已完成 (MDN = 1) (MDN = 0)。除停止命令外，用户无法在上一个动作完成之前启动任何新的运动命令。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡。

句法:

C/C++:

```
I32 FNTYPE APS_relative_linear_move( I32 Dimension, I32 *Axis_ID_Array, I32
*Distance_Array, I32 Max_Linear_Speed );
```

Visual Basic:

```
APS_relative_linear_move( ByVal Dimension As Long, Axis_ID_Array As Long,
Distance_Array As Long, ByVal Max_Linear_Speed As Long) As Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (2~4 轴)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

对于 EMX-100: 轴 ID 数组范围为 0 到 65535。数组的第一个元素是主轴，并按升序排列。

例

如 Axis_ID_Array[2] = {1,3} 或 Axis_ID_Array[2] = {2,3} 或 Axis_ID_Array[2] = {0,1}

I32 *Distance_Array: 相对距离数组。（单位：脉冲）

I32 Max_Linear_Speed: 最大线性插补速度。（单位：脉冲/秒）

对于 EMX-100: I32 Max_Linear_Speed: 最大线性插补速度；范围是 1~8,000,000（单位：脉冲/秒）

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
//...初始化板卡
I32 Dimension = 4;
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; //轴 ID 0 是主轴。
I32 Distance_Array[4] = {10000, 20000, 30000, 40000 };
I32 Max_Linear_Speed = 10000;
I32 Ret;
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

Ret = APS_relative_linear_move( Dimension, Axis_ID_Array, Distance_Array,
Max_Linear_Speed );
...
```

还可以看看：

[APS_relative_linear_move\(\)](#); [APS_set_axis_param\(\)](#);

APS_absolute_arc_move	开始一个绝对位置的圆弧插补
-----------------------	---------------

支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C) , HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C

描述:

该函数用于启动一个绝对圆弧插补定位的运动。用户必须为圆弧插补指定一个中心位置和运动角度。速度曲线的加减速速度可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

符号定义	参数编号	描述
PRA_SF	0x20	运动 S-factor
PRA_ACC	0x21	加速度
PRA_DEC	0x22	减速度
PRA_VS	0x23	开始速度
PRA_VM	0x24	最大速度
PRA_VE	0x25	终止速度

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速度可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴 ID 2 和 3 正在执行圆弧插补，轴 ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。每个执行圆弧插补的轴的运动状态“圆弧插补信号 (CIP)”在命令开始时为打开，在命令结束时为关闭。如果圆弧插补正常停止，则正常停止信号 (NSTP) 将打开。相反，如果圆弧插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号 (ASTP) 将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

这个命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：Axis_ID_Array 中指定的 2 个轴必须属于同一张板卡。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_arc_move( I32 Dimension, I32 *Axis_ID_Array, I32  
*Center_Pos_Array, I32 Max_Arc_Speed, I32 Angle );
```

Visual Basic:

```
APS_absolute_arc_move( ByVal Dimension As Long, Axis_ID_Array As Long,  
Center_Pos_Array As Long, ByVal Max_Arc_Speed As Long, ByVal Angle As Long )As  
Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (最大尺寸参考产品规格)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 *Center_Pos_Array: 绝对圆心位置。 单位 : 脉冲。

I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度)。 单位 : 脉冲 / 秒

I32 Angle: 运动角度。值的范围 : -360~360 度。逆时针为正。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 2;  
I32 Axis_ID_Array[2] = { 2, 4 }; //轴 ID 2 是主轴  
I32 Master_Axis_ID = 2; //轴 ID 2 是主轴  
I32 Center_Pos_Array[2] = {100000, 0};  
I32 Max_Arc_Speed = 10000; // 脉冲 / 秒  
I32 Angle = -180; // 顺时针 180 度  
I32 Ret; //返回代码
```

//...

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线  
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度  
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度  
Ret = APS_absolute_arc_move( Dimension, Axis_ID_Array, Center_Pos_Array,  
Max_Arc_Speed, Angle ); //执行圆弧插补。
```

还可以看看:

[APS_relative_arc_move\(\)](#); [APS_set_axis_param\(\)](#); [APS_get_axis_param \(\)](#);
[APS_motion_status\(\)](#); [APS_stop_move\(\)](#); [APS_emg_stop\(\)](#)

APS_relative_arc_move	开始一个相对距离的圆弧插补
-----------------------	---------------

支持的产品:PCI-8253/56, PCI-8392(H) , MNET-4XMO-(C) , HSL-4XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C

描述:

该函数用于启动一个相对圆弧插补定位的运动。用户必须为圆弧插补指定一个中心位置相对当前命令的位置和运动角度。速度曲线的加减速速度和曲线可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

符号定义	参数编号	描述
PRA_SF	0x20	运动 S-factor
PRA_ACC	0x21	加速率
PRA_DEC	0x22	减速率
PRA_VS	0x23	开始速度
PRA_VM	0x24	最大速度
PRA_VE	0x25	终止速度

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴 ID 2 和 3 正在执行圆弧插补，轴 ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。每个执行圆弧插补的轴的运动状态“圆弧插补信号 (CIP)”在命令开始时为打开，在命令结束时为关闭。如果圆弧插补正常停止，则正常停止信号 (NSTP) 将打开。相反，如果圆弧插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号 (ASTP) 将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

该命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：Axis_ID_Array 中指定的 2 个轴必须属于同一张板卡。

句法:

C/C++:

```
I32 FNTYPE APS_relative_arc_move( I32 Dimension, I32 *Axis_ID_Array, I32  
*Center_Offset_Array, I32 Max_Arc_Speed, I32 Angle );
```

Visual Basic:

```
APS_relative_arc_move( ByVal Dimension As Long, Axis_ID_Array As Long,  
Center_Offset_Array As Long, ByVal Max_Arc_Speed As Long, ByVal Angle As Long ) As  
Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (最大尺寸参考产品规格)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 *Center_Offset_Array: 圆心位置与当前命令位置的相对距离。单位: 脉冲。

I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度)。 单位 : 脉冲/秒

I32 Angle: 运动角度。值的范围 : -360~360 度。逆时针为正。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 2;
```

```
I32 Axis_ID_Array[2] = { 1, 3}; //轴 ID 1 是主轴。
```

```
I32 Master_Axis_ID = 1; //轴 ID 1 是主轴。
```

```
I32 Center_Offset_Array [2] = {300000, 0};
```

```
I32 Max_Arc_Speed = 20000; // 脉冲/秒
```

```
I32 Angle = 90; // 逆时针 90 度。
```

```
I32 Ret; //返回代码
```

```
//...
```

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
```

```
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
```

```
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度
```

```
Ret = APS_relative_arc_move( Dimension, Axis_ID_Array, Center_Offset_Array,  
Max_Arc_Speed, Angle ); //执行圆弧插补。
```

还可以看看:

[APS_absolute_arc_move \(\)](#); [APS_set_axis_param \(\)](#); [APS_get_axis_param \(\)](#);

[APS_motion_status\(\)](#); [APS_stop_move\(\)](#); [APS_emg_stop\(\)](#)

APS_absolute_arc_move_3pe	利用通过和终点法开始一个绝对位置的圆弧插补
---------------------------	-----------------------

支持的产品:PCI-8253/56

描述:

该函数用于启动一个绝对圆弧插补定位的运动。用户必须为圆弧插补指定绝对通过位置和终点位置。速度曲线的加减速速度可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速度可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴 ID 2 和 3 正在执行圆弧插补，轴 ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。每个执行圆弧插补的轴的运动状态“圆弧插补信号 (CIP)”在命令开始时为打开，在命令结束时为关闭。如果圆弧插补正常停止，则正常停止信号 (NSTP) 将打开。相反，如果圆弧插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号 (ASTP) 将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

该命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：

1. 此模式支持 2D 和 3D 圆弧插补运动。
2. 在 Axis_ID_Array 中指定的 2 或 3 轴必须属于同一张板卡。
3. 通过和终点模式的圆弧插补不支持完整的圆弧。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_arc_move_3pe( I32 Dimension, I32 *Axis_ID_Array, I32  
*Pass_Pos_Array, I32 *End_Pos_Array, I32 Max_Arc_Speed );
```

Visual Basic:

```
APS_absolute_arc_move_3pe( ByVal Dimension As Long, Axis_ID_Array As Long,  
Pass_Pos_Array As Long, End_Pos_Array As Long, ByVal Max_Arc_Speed As Long )As  
Long
```

参数:

I32 Dimension: The dimension of interpolation axes. (The maximum dimension is support to 3D)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 *Pass_Pos_Array: 绝对通过位置。单位 : 脉冲。

I32 *End_Pos_Array: 绝对终点位置。单位 : 脉冲。

I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度) 。 单位 : 脉冲 / 秒

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 3;  
I32 Axis_ID_Array[3] = { 2, 3, 4 }; //轴 ID 2 是主轴  
I32 Master_Axis_ID = 2; //轴 ID 2 是主轴  
I32 Pass_Pos_Array[3] = {50000, 50000, 50000};  
I32 End_Pos_Array[3] = {100000, 100000, 0}  
I32 Max_Arc_Speed = 400000; // 脉冲 / 秒  
I32 Ret; //返回代码  
  
//...  
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线  
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 1000000 ); //设置加速度  
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 1000000 ); //设置减速度  
Ret = APS_absolute_arc_move_3pe( Dimension, Axis_ID_Array, Pass_Pos_Array,  
End_Pos_Array, Max_Arc_Speed ); //执行圆弧插补。
```

还可以看看:

```
APS_absolute_arc_move();APS_relative_arc_move();APS_relative_arc_move_3pe();  
APS_set_axis_param();APS_get_axis_param();APS_motion_status();APS_stop_move();  
APS_emg_stop()
```

APS_relative_arc_move_3pe	利用通过和终点模式开始一个相对距离的圆弧插补
支持的产品:PCI-8253/56	

描述:

该函数用于启动一个相对的圆弧插补定位的运动。用户必须为圆弧插补指定通过位置和终点位置相对的当前命令位置。速度曲线的加减速速度和曲线可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴 ID 2 和 3 正在执行圆弧插补，轴 ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。运动状态：“圆弧插补信号（CIP）”在命令开始时为打开，在命令结束时为关闭。如果圆弧插补正常停止，则正常停止信号（NSTP）将打开。相反，如果圆弧插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号（ASTP）将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

该命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：

1. 此模式支持 2D 和 3D 圆弧插补运动。
2. 在 Axis_ID_Array 中指定的 2 或 3 轴必须属于同一张板卡。
3. 通过和终点模式的圆弧插补不支持完整的圆弧。

句法:

C/C++:

```
I32 FNTYPE APS_relative_arc_move_3pe( I32 Dimension, I32 *Axis_ID_Array, I32
*Pass_PosOffset_Array, I32 *End_PosOffset_Array, I32 Max_Arc_Speed );
```

Visual Basic:

```
APS_relative_arc_move_3pe( ByVal Dimension As Long, Axis_ID_Array As Long,  
Pass_PosOffset_Array As Long, End_PosOffset_Array As Long, ByVal Max_Arc_Speed As  
Long ) As Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (最大尺寸支持 3D)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 *Pass_PosOffset_Array: 圆弧通过位置相对当前命令位置。单位：脉冲。

I32 *End_PosOffset_Array: 圆弧终点位置相对当前命令位置。单位：脉冲。

I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度)。 单位：脉冲/秒

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 3;
```

```
I32 Axis_ID_Array[3] = { 0, 1, 2}; //轴 ID 0 是主轴。
```

```
I32 Master_Axis_ID = 0; //轴 ID 0 是主轴。
```

```
I32 Pass_PosOffset_Array [3] = {50000, 50000, 50000};
```

```
I32 End_PosOffset_Array[3] = {50000, 50000, -50000};
```

```
I32 Max_Arc_Speed = 200000; // 脉冲/秒
```

```
I32 Ret; //返回代码
```

```
//...
```

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
```

```
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 1000000 ); //设置加速度
```

```
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 1000000 ); //设置减速度
```

```
Ret = APS_relative_arc_move_3pe( Dimension, Axis_ID_Array, Pass_PosOffset_Array,  
End_PosOffset_Array, Max_Arc_Speed ); //执行圆弧插补。
```

还可以看看:

[APS_relative_arc_move\(\)](#); [APS_absolute_arc_move\(\)](#); [APS_absolute_arc_move_3pe\(\)](#);

[APS_set_axis_param\(\)](#); [APS_get_axis_param\(\)](#); [APS_motion_status\(\)](#); [APS_stop_move\(\)](#); [APS_emg_stop\(\)](#)

APS_absolute_helix_move	开始一个绝对位置的螺旋插补
支持的产品:PCI-8253/56	

描述:

该函数用于开始一个绝对螺旋插补定位的运动。 用户必须指定绝对圆心位置（2D），螺距长度，总的螺钉高度和螺旋插补的运动方向。速度曲线的加减速速度可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴ID 2 和 3 在轴ID 4 中执行圆弧插补和同步线性运动，轴ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。如果螺旋插补正常停止，则正常停止信号（NSTP）将打开。相反，如果螺旋插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号（ASTP）将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

该命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意:

1. 螺旋插补仅支持 3D 坐标空间
2. Axis ID 数组中的最后一个轴号必须是线性轴。
3. 圆心位置仅支持 2D。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_helix_move( I32 Dimension, I32 *Axis_ID_Array, I32
*Center_Pos_Array, I32 Max_Arc_Speed, I32 Pitch, I32 TotalHeight, I32 CwOrCcw );
```

Visual Basic:

```
APS_absolute_helix_move( ByVal Dimension As Long, Axis_ID_Array As Long,
Center_Pos_Array As Long, ByVal Max_Arc_Speed As Long, ByVal Pitch As Long, ByVal
TotalHeight As Long, ByVal CwOrCcw As Long )As Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (仅支持 3D)
I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。
I32 *Center_Pos_Array: 绝对通过位置。 单位 : 脉冲。
I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度) 。 单位 : 脉冲/秒
I32 Pitch: 螺旋的螺距。 单位 : 脉冲
I32 TotalHeight: 螺旋的深度。 单位 : 脉冲
I32 CwOrCcw: 运动方向
 CwOrCcw = 0 ---> 顺时针
 CwOrCcw = 1----> 逆时针

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 3;
I32 Axis_ID_Array[3] = { 2, 3, 4 }; //轴 ID 2 是主轴
I32 Master_Axis_ID = 2; //轴 ID 2 是主轴
I32 Center_Pos_Array[2] = {50000, 0};
I32 Max_Arc_Speed = 400000; // 脉冲/秒
I32 Pitch = 2500;
I32 Total 高度= 5000;
I32 CwOrCcw = 1; // 逆时针
I32 Ret; //返回代码

//...
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 1000000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 1000000 ); //设置减速度
Ret = APS_absolute_helix_move ( Dimension, Axis_ID_Array, Center_Pos_Array,
Max_Arc_Speed , Pitch, TotalHeight, CwOrCcw ); //执行螺旋插补
```

还可以看看:

[APS_relative_helix_move\(\)](#); [APS_set_axis_param \(\)](#); [APS_get_axis_param \(\)](#);
[APS_motion_status\(\)](#); [APS_stop_move\(\)](#), [APS_emg_stop\(\)](#)

APS_relative_helix_move	开始一个相对距离的螺旋插补
支持的产品:PCI-8253/56	

描述:

该函数用于开始一个相对的螺旋插补定位的运动。 用户必须指定相对于当前命令位置的圆心位置（2D），螺距长度，总的螺钉高度和螺旋插补的运动方向。速度曲线的加减速速度和曲线可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。由于速度参数是矢量方向（与圆弧相切），因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴ID 2 和 3 在轴ID 4 中执行圆弧插补和同步线性运动，轴ID 2 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。如果螺旋插补正常停止，则正常停止信号（NSTP）将打开。相反，如果螺旋插补异常停止（例如 ALM，EMG，SEMG 等打开），则异常停止信号（ASTP）将打开。

在轴运动期间，用户可以启动一个新的运动命令，包括停止命令，以覆盖前一个命令（尺寸和 Axis_ID_Array 必须相同）。轴将根据目标中心位置、新的速度曲线等立即切换到新的命令。

该命令不能被其他运动模式（例如归零操作）覆盖。用户必须停止轴运动才能切换到上述那些模式。

注意：

1. 螺旋插补仅支持 3D 坐标空间
2. Axis ID 数组中的最后一个轴号必须是线性轴。
3. 圆心位置仅支持 2D。

句法:

C/C++:

```
I32 FNTYPE APS_relative_helix_move( I32 Dimension, I32 *Axis_ID_Array, I32
*Center_PosOffset_Array, I32 Max_Arc_Speed, I32 Pitch, I32 TotalHeight, I32
CwOrCcw );
```

Visual Basic:

```
APS_relative_helix_move( ByVal Dimension As Long, Axis_ID_Array As Long,  
Center_PosOffset_Array As Long, ByVal Max_Arc_Speed As Long, ByVal Pitch As Long,  
ByVal TotalHeight As Long, ByVal CwOrCcw As Long )As Long
```

参数:

I32 Dimension: 插补轴的尺寸。 (仅支持 3D)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 *Center_PosOffset_Array: 相对于当前命令位置的圆心位置。 单位：脉冲

I32 Max_Arc_Speed: 最大圆弧插补速度 (圆弧切线速度)。 单位：脉冲/秒

I32 Pitch: 螺旋的螺距。单位：脉冲

I32 TotalHeight: 螺旋的深度。 单位：脉冲

I32 CwOrCcw: 运动方向

CwOrCcw = 0 ---> 顺时针

CwOrCcw = 1----> 逆时针

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Dimension = 3;  
I32 Axis_ID_Array[3] = { 2, 3, 4 }; //轴 ID 2 是主轴  
I32 Master_Axis_ID = 2; //轴 ID 2 是主轴  
I32 Center_PosOffset_Array[2] = {50000, 0};  
I32 Max_Arc_Speed = 400000; // 脉冲/秒  
I32 Pitch = 2500;  
I32 Total 高度= 5000;  
I32 CwOrCcw = 1; // 逆时针  
I32 Ret; //返回代码  
  
//...  
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线  
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 1000000 ); //设置加速度  
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 1000000 ); //设置减速度  
Ret = APS_absolute_helix_move ( Dimension, Axis_ID_Array, Center_PosOffset_Array,  
Max_Arc_Speed , Pitch, TotalHeight, CwOrCcw ); //执行螺旋插补
```

还可以看看:

APS_absolute_helix_move();APS_set_axis_param();

APS_get_axis_param();APS_motion_status();APS_stop_move();APS_emg_stop()

APS_absolute_helical_move	开始一个绝对位置的螺旋插补
---------------------------	---------------

支持的产品:PCI(e)-8154/8158

描述:

该函数用于开始一个绝对的螺旋插补定位的运动。用户必须指定为螺旋插补指定绝对圆心位置(2D)，绝对终点位置，螺距和运动方向。速度曲线的加减速速度可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速率可能不足。由于速度参数是矢量方向(与圆弧相切)，因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴ID 0 和 1 在轴ID 2 中执行圆弧插补和同步线性运动，轴ID 0 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。如果螺旋插补正常停止，则正常停止信号(NSTP)将打开。相反，如果螺旋插补异常停止(例如ALM, EMG, SEMG等打开)，则异常停止信号(ASTP)将打开。

句法:

C/C++:

```
I16 APS_absolute_helical_move( I32 *Axis_ID_Array, I32 *Center_Pos_Array, I32  
*End_Pos_Array, I32 Pitch, I32 Dir, I32 Max_Speed );
```

Visual Basic:

```
APS_absolute_helical_move( Axis_ID_Array As Long, Center_Pos_Array As  
Long, End_Pos_Array As Long, ByVal Pitch As Long, ByVal Dir As Long, ByVal  
Max_Speed As Long )As Long
```

参数:

I32 *Axis_ID_Array: 轴ID数组。每个螺旋组需要4个轴，例如，PCIe-8154需要0~3轴，PCIe-8158需要0~3或4~7。

I32 *Center_Pos_Array: 绝对中心位置。单位：脉冲。

I32 *End_Pos_Array: 绝对终点位置。单位：脉冲。

I32 Max_Speed: 最大圆弧插补速度(圆弧切线速度)。单位：脉冲/秒

I32 Pitch: 螺距。单位：脉冲

I32 Dir: 运动方向

Dir = 0 → 顺时针

Dir = 1--→ 逆时针

示例：

```
I32 AxisArray[4] = {0, 1, 2, 3}; //设置轴 ID。 在此示例中，轴 0-1 进行圆弧插补，轴 2 沿垂直方向进行插补，轴 3 是用于内部计算的虚拟轴。  
I32 Center_Pos_Array[2] = {1000, 2000}; //设置运动的中心位置（单位：脉冲）  
I32 End_Pos_Array[2] = {2000, 4000}; // 设定运动的终点位置（单位：脉冲）  
I32 MaxVel = 5000; //设置最大速度，单位为脉冲/秒  
I32 Pitch = 500; //设置螺距长度  
I32 Dir = 0; //设定方向为顺时针  
for( int i = 0; i < total_axis; i ++)  
{  
    APS_set_axis_param(i, PRA_CURVE, 0 ); //设置 T 曲线 APS_set_axis_param(i, PRA_ACC,  
    50000 ); //设置加速度 APS_set_axis_param(i, PRA_DEC, 50000 ); //设置减速度  
    APS_set_axis_param(i, PRA_VS, 0 ); //设置开始速度  
}  
APS_absolute_helical_move (AxisArray, Center_Pos_Array, End_Pos_Array, Pitch, Dir,  
MaxVel);
```

还可以看看:

```
APS_relative_helical_move();APS_set_axis_param();  
APS_get_axis_param();APS_motion_status();APS_stop_move();APS_emg_stop()
```

APS_relative_helical_move	开始一个相对距离的螺旋插补
---------------------------	---------------

支持的产品:PCI(e)-8154/8158

描述:

该函数用于开始一个相对的螺旋插补定位的运动。用户必须为螺旋插补指定圆心偏移位置(2D)，圆弧末端偏移位置，螺距和运动方向。速度曲线的加减速速度可通过轴参数函数进行设置。在调用此函数之前，应先设置以下轴参数。

PRA_CURVE

PRA_ACC

PRA_DEC

PRA_VS

PRA_VE

参数的详细内容请参照轴参数表。

尽管在函数参数中设置了最大速度，但是由于用户设置为达到最大速度，因此运动距离和加速度可能不足。由于速度参数是矢量方向(与圆弧相切)，因此该函数将使用主轴的加减速时间常数进行计算。用户执行插补运动的最小轴编号就是主轴。例如，轴ID 0 和 1 在轴ID 2 中执行圆弧插补和同步线性运动，轴ID 0 是主轴。

此函数为“发后即忘”的方式，这就意味着在轴运动期间不会挂起用户的程序或过程。用户必须使用运动状态检查函数或中断事件等待功能来等待它完成。如果螺旋插补正常停止，则正常停止信号(NSTP)将打开。相反，如果螺旋插补异常停止(例如ALM, EMG, SEMG等打开)，则异常停止信号(ASTP)将打开。

句法:

C/C++:

```
I16 APS_relative_helical_move( I32 *Axis_ID_Array, I32 *Center_Offset_Array, I32  
*End_Offset_Array, I32 Pitch, I32 Dir, I32 Max_Speed );
```

Visual Basic:

```
APS_relative_helical_move( Axis_ID_Array As Long, Center_Offset_Array As  
Long, End_Offset_Array As Long, ByVal Pitch As Long, ByVal Dir As Long, ByVal  
Max_Speed As Long )As Long
```

参数:

I32 *Axis_ID_Array: 轴ID数组。每个螺旋组需要4个轴，例如，PCIe-8154需要0~3轴，PCIe-8158需要0~3或4~7。

I32 *Center_Offset_Array: 相对中心偏移位置。单位：脉冲。

I32 *End_Offset_Array: 相对终点偏移位置。单位：脉冲。

I32 Max_Speed: 最大圆弧插补速度(圆弧切线速度)。单位：脉冲/秒

I32 Pitch: 设置螺距长度

I32 Dir: 运动方向

Dir = 0 → 顺时针

Dir = 1 → 逆时针

示例：

```
I32 AxisArray[4] = {0, 1, 2, 3}; //设置轴 ID。 在此示例中，轴 0-1 进行圆弧插补，轴 2 沿垂直方向进行插补，轴 3 是用于内部计算的虚拟轴。  
I32 Center_Offset_Array[2] = {1000, 2000}; //设置中心位置的运动（单位：脉冲）  
I32 End_Offset_Array[2] = {2000, 4000}; // 设定运动的终点位置（单位：脉冲）  
I32 MaxVel = 5000; //设置最大速度，单位为脉冲/秒  
I32 Pitch = 500; //设置螺距长度  
I32 Dir = 0; //设定方向为顺时针( int i = 0; i < total_axis; i++)  
{  
    APS_set_axis_param(i, PRA_CURVE, 0); //设置 T 曲线  
    APS_set_axis_param(i, PRA_ACC, 50000); //设置加速度  
    APS_set_axis_param(i, PRA_DEC, 50000); //设置减速度  
    APS_set_axis_param(i, PRA_VS, 0); //设置开始速度  
}  
APS_relative_helical_move (AxisArray, Center_Offset_Array, End_Offset_Array, Pitch, Dir,  
MaxVel);
```

还可以看看：

```
APS_absolute_helical_move();APS_set_axis_param ();APS_get_axis_param ();  
APS_motion_status();APS_stop_move();APS_emg_stop()
```

10. 高级单步运动和插补

APS_ptp	开始一个单步运动
---------	----------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于执行一个单轴运动。没有任何后缀表示执行一个单步运动不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_ptp( I32 Axis_ID, I32 Option, F64 Position, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double, Wait As ASYNCALL) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI-8254/58 / AMP-204/8C:

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式						强制中止	等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9: 当发生不同方向或减速距离不足时，该位(bit)用于分配曲线中止行为。

0: 曲线将平滑改变。 (默认值)

1: 曲线将立即改变。

Bit 10 ~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)

0100b(5): 混合高的那个 (Blending high)

Bit 16~: 保留以备将来使用，设置为 0。

F64 Position: 值指定要运动的位置/距离。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCIe-833x:

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)

0100b(5): 混合高的那个 (Blending high)

Bit 16~: 保留以备将来使用，设置为 0。

F64 Position: 值指定要运动的位置/距离。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 opt = 0x1000; //绝对，不等待触发，缓冲模式
```

```
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//绝对运动到位置 10000
```

```
APS_ptp( Axis_ID, opt, 10000, wait );
```

还可以看看：

APS_ptp_v	根据运动曲线开始一个单步运动
支持的产品:PCI(e)-8154/58, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于执行一个单轴运动。后缀_v 表示执行单个运动仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_ptp_v( I32 Axis_ID, I32 Option, F64 Position, F64 Vm, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp_v(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double,  
ByVal Vm As Double, Wait As ASYNCALL) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI(e)-8154/58:

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8
缓冲模式							

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~11: 保留以备将来使用，设置为 0。

Bit12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)，会开始位置运动，或者在运动时以新的位置和速度覆盖旧的运动。

00001b(1): 已缓存 (Buffered) **注意：它被保留供将来使用。**

F64 Position: 值指定要运动的位置/距离。

F64 Vm: 值指定最大速度。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCI-8254/58 / AMP-204/8C:

7	6	5	4	3	2	1	Bit : 0
---	---	---	---	---	---	---	---------

							绝对 (0) /相对 (1)
15	14	13	12	11	10	9	8
缓冲模式						强制中止	等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用 , 设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9: 当发生不同方向或减速距离不足时 , 该位(bit)用于分配曲线中止行为。

0: 曲线将平滑改变。 (默认值)

1: 曲线将立即改变。

Bit 10~11: 保留以备将来使用 , 设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)

0100b(5): 混合高的那个 (Blending high)

Bit 16~: 保留以备将来使用 , 设置为 0。

F64 Position: 值指定要运动的位置/距离。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意 : 它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针 , 则调用将不等待 , 并且函数将立即返回。

对于 PCIe-833x :

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) /相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用 , 设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用 , 设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)
0100b(5): 混合高的那一个 (Blending high)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 Position: 值指定要运动的位置/距离。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针 , 则调用将不等待 , 并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 opt = 0x1000; //绝对 , 中止模式  
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//以 Vm ( 100000 ) 绝对运动到位置 ( 10000 )  
APS_ptp_v ( Axis_ID, opt, 10000, 100000, wait );
```

还可以看看:

APS_relative_move();APS_absolute_move();APS_relative_move_ovrd();APS_absolute_mo
ve_ovrd()

APS_ptp_all	根据所有运动曲线开始一个单步运动
支持的产品:PCI(e)-8154/58, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于执行一个单步移动。_all 后缀表示执行单步运动所需的所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 SFac。

句法:

C/C++:

```
I32 FNTYPE APS_ptp_all( I32 Axis_ID, I32 Option, F64 Position, F64 Vs, F64 Vm, F64 Ve,
F64 Acc, F64 Dec, F64 Sfac, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp_all(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double,
ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double,
ByVal Dec As Double, ByVal Sfac As Double, Wait As ASYNCALL) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI(e)-8154/58:

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8
缓冲模式							

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~11: 保留以备将来使用，设置为 0。

Bit12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)，会开始位置运动，或者在运动时以新的位置和速度覆盖旧的运动。

00001b(): 已缓冲 (Buffered)。 注意：它被保留供将来使用。

F64 Position: 值指定要运动的位置/距离。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 Sfac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 注意：它被保留供将来使用。 用 NULL 传递将定义一个等待的调用。如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCI-8254/58 / AMP-204/8C:

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式						强制中止	等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9: 当发生不同方向或减速距离不足时，该位(bit)用于分配曲线中止行为。

0: 曲线将平滑改变。 (默认值)

1: 曲线将立即改变。

Bit 10~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)

0100b(5): 混合高的那个 (Blending high)

Bit 16~: 保留以备将来使用，设置为 0。

F64 Position: 值指定要运动的位置/距离。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCIe-833x :

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)

15	14	13	12	11	10	9	8
缓冲模式						等待触发	

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用 , 设置为 0.

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用 , 设置为 0.

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting)

0001b(1): 已缓存 (Buffered)

0010b(2): 混合低的那个 (Blending low)

0011b(3): 混合前一个 (Blending previous)

0100b(4): 混合下一个 (Blending next)

0100b(5): 混合高的那个 (Blending high)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 Position: 值指定要运动的位置/距离。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 Sfac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 opt = 0x1000; //绝对 , 中止模式
```

```
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//使用 Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), Sfac(0.5) 绝对运动到位置(10000)
```

```
APS_ptp_all( Axis_ID, opt, 10000, 10, 100000, 20, 200000, 200000, 0.5, wait );
```

还可以看看:

[APS_relative_move\(\)](#); [APS_absolute_move\(\)](#); [APS_relative_move_ovrd\(\)](#); [APS_absolute_move_ovrd\(\)](#); [APS_ptp_v\(\)](#)

APS_vel	开始一个速度运动
---------	----------

支持的产品:PCI(e)-8154/58, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于执行 Vm 速度运动。没有任何后缀表示执行速度运动时不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_vel( I32 Axis_ID, I32 Option, F64 Vm, ASYNDCALL *Wait);
```

Visual Basic:

```
APS_vel(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Vm As Double, Wait As ASYNDCALL) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI(e)-8154/58:

7	6	5	4	3	2	1	0
							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8

Bit 0: 0: 正方向 1: 反方向

Bit 1~15: 保留以备将来使用，设置为 0。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。** 用 NULL 传递将定义一个等待的调用。如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCI-8254/58 / AMP-204/8C:

7	6	5	4	3	2	1	Bit : 0
							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8
						强制中止	等待触发

Bit 0: 0: 正方向 1: 反方向

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9: 当不同方向发生时，该位用于分配曲线中止行为。

0: 曲线将平滑改变。 (默认值)

1: 曲线将立即改变。

Bit 10~: 保留以备将来使用，设置为 0。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCIe-833x：

7	6	5	4	3	2	1	Bit : 0
							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8
							等待触发

Bit 0: 0: 正方向 1: 反方向

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~: 保留以备将来使用，设置为 0。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 opt = 0; // 正方向，不是等待触发
```

```
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//以 Vm ( 10000 ) 执行速度运动。
```

```
APS_vel( Axis_ID, opt, 10000, wait );
```

还可以看看：

[APS_velocity_move\(\)](#)

APS_vel_all	根据所有曲线开始一个速度运动
支持的产品:PCI(e)-8154/58, PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于执行速度移动。_all 后缀表示执行速度运动需要的所有运动曲线，包括位置，Vs，Vm，Acc 和 S-factor。

句法:

C/C++:

```
I32 FNTYPE APS_vel_all( I32 Axis_ID, I32 Option, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 Sfac, ASYNCALL *Wait);
```

Visual Basic:

```
APS_vel_all(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal Sfac As Double, Wait As ASYNCALL) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI(e)-8154/58:

7	6	5	4	3	2	1	0
							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8

Bit 0: 0: 正方向 1: 反方向

Bit 1~15: 保留以备将来使用，设置为 0。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 Sfac: 值指定 s-factor。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。**注意：它被保留供将来使用。** 用 NULL 传递将定义一个等待的调用。如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCI-8254/58 / AMP-204/8C:

7	6	5	4	3	2	1	Bit : 0
---	---	---	---	---	---	---	---------

							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8
						强制中止	等待触发

Bit 0: 0: 正方向 1: 反方向

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9: 当不同方向发生时，该位用于分配曲线中止行为。

0: 曲线将平滑改变。（默认值）

1: 曲线将立即改变。

Bit 10~: 保留以备将来使用，设置为 0。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

对于 PCIe-833x：

7	6	5	4	3	2	1	Bit : 0
							0: 正方向 / 1: 反方向
15	14	13	12	11	10	9	8
							等待触发

Bit 0: 0: 正方向 1: 反方向

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~: 保留以备将来使用，设置为 0。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 Sfac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 opt = 0; //正方向 , 不是等待触发  
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//以 Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), Sfac(0.5) 执行一个速度运动  
APS_vel_all( Axis_ID, opt, 10, 100000, 20, 200000, 200000, 0.5, wait );
```

还可以看看:

[APS_velocity_move\(\)](#); [APS_vel\(\)](#)

APS_line	开始一个线性插补
----------	----------

支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于执行线性插补。没有任何后缀表示执行线性插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_line( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64
*PositionArray, F64 *TransPara, ASYNCALL *Wait);
```

Visual Basic:

```
APS_line (ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long,
PositionArray As Double, TransPara As Double, Wait As ASYNCALL) As Long
```

参数:

I32 Dimension: 一个值指定轴的尺寸。范围是 2 到 4。

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

注意 : Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

对于 PCI(e)-8154/58:

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

注意 : 它被保留供将来使用。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意 : 它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式：

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止 (Aborting) – 强制中止

0010b(2): 正在中止 (Aborting) – 停止然后继续运行 (TransPara_0 作为减速度。 [dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0011b(3): 已缓冲 (Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的%值范围：0.0~1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。

注意：它被保留供将来使用。

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1：

以下示例适用于 PCI(e)-8154/58

```
I32 opt = 0; // 绝对
```

```
I32 Dimension = 4;
```

```
I32 Master_Axis_ID = 0;
```

```
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; // 轴 ID 0 是主轴。
```

```
I32 Distance_Array[4] = {10000, 20000, 30000, 40000};
```

```
I32 Max_Linear_Speed = 10000;
```

```
I32 Ret;
F64 TransPara = 0;
ASYNDCALL *wait = NULL;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

//执行一个线性运动
Ret = APS_line ( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, wait );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
I32 opt = 0x3000; //绝对 , 不等待触发 , 缓冲模式
```

```
F64 TransPara = 0; //不在缓冲模式下
```

```
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//执行一个线性运动
```

```
APS_line ( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, wait );
```

还可以看看:

APS_relative_linear_move();APS_absolute_linear_move()

APS_line_v	根据运动曲线开始一个线性运动
支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于执行线性插补。_v 后缀表示执行线性插补时仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_line_v( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64
*PositionArray, F64 *TransPara, F64 Vm, ASYNCALL *Wait);
```

Visual Basic:

```
APS_line (ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long,
PositionArray As Double, TransPara As Double, ByVal Vm As Double, Wait As
ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 Dimension: 一个值指定轴的尺寸。范围是 2 到 4。

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

注意：它被保留供将来使用。

F64 Vm: 值指定最大速度。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 Dimension: 一个值指定轴的尺寸。 范围是 2 到 6。

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式：

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速度。)

0001b(1): 正在中止 (Aborting) – 强制中止

0010b(2): Aborting – 停止然后继续运行 (TransPara_0 作为减速度。 [dec > 0]，如果 dec <= 0，则内核采用新的减速度。)

0011b(3): 已缓冲 (Buffered)

0100b(4): 混合 (Blending)- 减速事件

0101b(5): 混合 (Blending)- 剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合 (Blending)- 剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围： 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

```
以下示例适用于 PCI(e)-8154/58
I32 opt = 0; //绝对
I32 Dimension = 4;
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; //轴 ID 0 是主轴。
F64 Distance_Array[4] = {10000, 20000, 30000, 40000 };
F64 Max_Linear_Speed = 10000;
I32 Ret;
F64 TransPara = 0;
ASYNDCALL *wait = NULL;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

//执行一个线性运动
Ret = APS_line_v ( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara,
Max_Linear_Speed, wait );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
I32 opt = 0x3000; //绝对，不等待触发，缓冲模式
F64 TransPara = 0; //不在缓冲模式下
ASYNDCALL *wait = NULL; //一个等待的调用
```

```
//以 Vm(10000)执行一个线性运动
APS_line_v( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, 10000, wait );
```

还可以看看：

APS_relative_linear_move();APS_absolute_linear_move();APS_line()

APS_line_all	根据所有曲线开始一个线性运动
支持的产品:PCI(e)-8154/58, PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于执行线性插补。_all 后缀表示执行线插值时所需的所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 S-Factor。

句法:

C/C++:

```
I32 FNTYPE APS_line_all( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64
*PositionArray, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac,
ASYNCALL *Wait);
```

Visual Basic:

```
APS_line_all(ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long,
PositionArray As Double, TransPara As Double, ByVal Vs As Double, ByVal Vm As
Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac
As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 Dimension:一个值指定轴的尺寸。范围是 2 到 4。

I32 *Axis_ID_Array:指针指示轴数组的起始地址。 **注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。**

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

注意：它被保留供将来使用。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 Dimension: 一个值指定轴的尺寸。 范围是 2 到 6。

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式：

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速度。)

0001b(1): 正在中止 (Aborting) – 强制中止

0010b(2): Aborting – 停止然后继续运行 (TransPara_0 作为减速度。 [dec > 0]，

如果 dec <= 0，则内核采用新的减速度。)

0011b(3): 已缓冲 (Buffered)

0100b(4): 混合 (Blending)- 减速事件

0101b(5): 混合 (Blending)- 剩余距离 (TransPara_0 为残留距离 ≥ 0.0)

0110b(6): 混合 (Blending)- 剩余距离 (以行进距离的百分比表示) (TransPara_0 作
为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *PositionArray: 指针指示位置数组的起始地址。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI(e)-8154/58

```
I32 opt = 0; //绝对
I32 Dimension = 4;
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; //轴 ID 0 是主轴。
F64 Distance_Array[4] = {10000, 20000, 30000, 40000 };
I32 Ret;
F64 TransPara = 0;
ASYNCALL *wait = NULL;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

//执行一个线性运动
//以 Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5) 执行一个线性运动
APS_line_all( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, 10, 100000,
20, 200000, 200000, 0.5, wait );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
I32 opt = 0x3000; //绝对，不等待触发，缓冲模式
F64 TransPara = 0; //不在缓冲模式下
ASYNCALL *wait = NULL; //一个等待的调用
```

```
//以 Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5) 执行一个线性运动
APS_line_all( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, 10, 100000, 20,
200000, 200000, 0.5, wait );
```

还可以看看:

APS_relative_linear_move(); APS_absolute_linear_move();APS_line()

APS_arc2_ca	开始一个角度类型的 2D 运动
支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于执行角度类型为弧2_ca 的 2D 圆弧插补。它跟随中心位置和角度。当前位置和中心位置参数将决定圆弧的半径。没有任何后缀表示执行 2D 圆弧插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。 用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 Angle,
F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,
ByVal Angle As Double, TransPara As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

注意 : Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定了角度。单位是弧度。范围是-2 * PI ~ 2 * PI。正值是逆时针，负值是顺时针。

F64 *TransPara: 指针指示传输参数的起始地址。

注意 : 它被保留供将来使用。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意 : 它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
---	---	---	---	---	---	---	---------

							绝对(0)/相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式：

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。**注意：如果设置为模式 2，它将返回错误代码。**

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 >= 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的%值范围：0.0~1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 opt = 0; //绝对
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[2] = {0, 1}; //轴 ID 0 是主轴。
F64 Center_Pos_Array[2] = {100000, 0};
F64 Angle = -180 * (2PI / 360); // 顺时针 180 度
I32 Ret;
F64 TransPara = 0;
ASYNDCALL *wait = NULL;
```

APS_set_axis_param(Master_Axis_ID, PRA_CURVE, 1); //设置 S 曲线

```
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度  
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度  
  
//执行圆弧运动  
APS_arc2_ca (Axis_ID_Array, opt, Center_Pos_Array, Angle, &TransPara, wait );
```

还可以看看:

APS_relative_arc_move(); APS_absolute_arc_move()

APS_arc2_ca_v	根据运动曲线开始一个角度类型的 2D 运动
支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于执行角度类型为弧插值的 2D 弧插值。它跟随中心位置和角度。当前位置和中心位置参数将决定圆弧半径。后缀_v 表示执行 2D 圆弧插补仅需要一个运动曲线，即 Vm，其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
Angle, F64 *TransPara, F64 Vm, ASYNDCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,
ByVal Angle As Double, TransPara As Double, ByVal Vm As Double, Wait As
ASYNDCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定了角度。单位是弧度。范围是 -2 * PI ~ 2 * PI。正值是逆时针，负值是顺时针。

F64 *TransPara: 指针指示传输参数的起始地址。

注意：它被保留供将来使用。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用 , 设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用 , 设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0] , 如果 dec <= 0 , 则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意 : 如果设置为模式 2 , 它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示)

(TransPara_0 作为剩余距离 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用 , 设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 注意 : 它被保留供将来使用。

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针 , 则调用将不等待 , 并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 opt = 0; //绝对
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[2] = {0, 1}; //轴 ID 0 是主轴。
F64 Center_Pos_Array[2] = {100000, 0};
F64 Angle = -180 * (2*PI / 360); // 顺时针 180 度
F64 Speed = 10000.0;
I32 Ret;
F64 TransPara = 0;
```

```
ASYNDCALL *wait = NULL;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

//执行圆弧运动
APS_arc2_ca_v (Axis_ID_Array, opt, Center_Pos_Array, Angle, &TransPara, Speed ,wait );
```

还可以看看:

```
APS_relative_arc_move();APS_absolute_arc_move();APS_arc2_ca()
```

APS_arc2_ca_all	根据所有曲线开始一个角度类型的 2D 运动
支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C , PCIE-833x	

描述:

此函数用于执行角度类型为_arc2_ca 的 2D 圆弧插补。它跟随中心位置和角度。当前位置和中心位置参数将决定弧的半径。_all 后缀表示所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 SFac , 都是执行 2D 圆弧插补所必需的。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 Angle, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, ByVal Angle As Double, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

注意 : Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定了角度。单位是弧度。范围是-2 * PI ~ 2 * PI。正值是逆时针，负值是顺时针。

F64 *TransPara: 指针指示传输参数的起始地址。注意：它被保留供将来使用。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1: 相对运动 0: 绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式：

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度)。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。 **注意：如果设置为模式 2，它将返回错误代码。**

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 Angle: 值指定角度。 单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 opt = 0; //绝对
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[2] = {0, 1}; //轴 ID 0 是主轴。
F64 Center_Pos_Array[2] = {100000, 0};
F64 Angle = -180 * (2*PI / 360); // 顺时针 180 度
I32 Ret;
F64 TransPara = 0;
ASYNCALL *wait = NULL;

APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度

//以 Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5) 执行圆弧运动
APS_arc2_ca_all (Axis_ID_Array, opt, Center_Pos_Array, Angle, &TransPara, 10, 100000,
20, 200000,
200000, 0.5, wait );
```

还可以看看：

[APS_relative_arc_move\(\)](#); [APS_absolute_arc_move\(\)](#); [APS_arc2_ca\(\)](#)

APS_arc2_ce

开始一个终点位置类型的 2D 运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x**描述:**

此函数用于执行名为`_arc2_ce`的最终位置类型的 2D 圆弧插补。它跟随中心位置，终点位置和方向。没有任何后缀表示执行 2D 圆弧插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。 用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*EndArray, I16 Dir, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 FNTYPE APS_arc2_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray
As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double, Wait As
ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 （ 1 ）
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度= theta + Dir x 2PI , 其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看:

APS_arc2_ce_v

根据速度曲线开始一个终点位置类型的 2D 运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x**描述:**

此函数用于执行名为_arc2_ce 的终点位置类型的 2D 圆弧插补。它跟随中心位置，终点位置和方向。后缀_v 表示执行 2D 圆弧插补仅需要一个运动曲线（即 Vm）。其他运动曲线在轴参数中设置。 用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*EndArray, I16 Dir, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,
EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vm As Double,
Wait As ASYNCALL ) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度= theta + Dir x 2PI , 其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例：

还可以看看:

APS_arc2_ce_all	根据所有曲线开始一个终点位置类型的 2D 运动
-----------------	-------------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_arc2_ce 的终点位置类型的 2D 圆弧插补。它跟随中心位置，终点位置和方向。_all 后缀表示所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 SFac , 都是执行 2D 圆弧插补所必需的。

句法:

C/C++:

```
I32 FNTYPE APS_arc2_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*EndArray, I16 Dir, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac,  
ASYNDCALL *Wait );
```

Visual Basic:

```
APS_arc2_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vs As Double,  
ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double,  
ByVal SFac As Double, Wait As ASYNDCALL ) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) /相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0] , 如果 dec <= 0 , 则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2 , 它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离> = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0
作为剩余距离的%值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转 , dir=-1 则负向旋转。总旋转
角度 = theta + Dir x 2PI , 其中 theta 是两个向量的角度 : 中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针 , 则调用将不等待 , 并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_arc3_ca	开始一个角度类型的 3D 运动
-------------	-----------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_arc3_ca 的角度类型的 3D 圆弧插补。它跟随角度，中心位置和法向矢量。没有任何后缀表示执行 3D 圆弧插补时不需要任何运动轮廓。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 Angle, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
NormalArray As Double, ByVal Angle As Double, TransPara As Double, Wait As  
ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 * NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看：

APS_arc3_ca_v	根据速度曲线开始一个角度类型的 3D 运动
---------------	-----------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_arc3_ca 的角度类型的 3D 圆弧插补。它跟随角度，中心位置和法向矢量。_v 后缀表示执行 3D 圆弧插补仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。 用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 Angle, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
NormalArray As Double, ByVal Angle As Double, TransPara As Double, ByVal Vm As  
Double, Wait As ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 * NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看:

APS_arc3_ca_all	根据所有曲线开始一个角度类型的 3D 运动
-----------------	-----------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_arc3_ca 的角度类型的 3D 圆弧插补。它跟随角度，中心位置和法向矢量。_all 后缀表示执行 3D 弧插值所需的所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 S-Factor。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 Angle, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec,  
F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
NormalArray As Double, ByVal Angle As Double, TransPara As Double, ByVal Vs As  
Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As  
Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0
作为剩余距离的%值范围 : 0.0~1.0)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

还可以看看:

APS_arc3_ce

开始一个终点位置类型的 3D 运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x**描述:**

此函数用于执行名为_arc3_ce 的终点位置类型的 3D 圆弧插补。它跟随中心位置，终点位置和方向。没有任何后缀表示执行 3D 圆弧插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*EndArray, I16 Dir, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 FNTYPE APS_arc3_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray
As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double, Wait As
ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 （ 1 ）
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度= theta + Dir x 2PI , 其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNCALL *Wait: 指向 ASYNCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看:

APS_arc3_ce_v

根据速度曲线开始一个终点位置类型的 3D 运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x**描述:**

此函数用于执行名为_arc3_ce 的终端位置类型的 3D 圆弧插补。它跟随中心位置，终点位置和方向。_v 后缀表示执行 3D 圆弧插补时仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*EndArray, I16 Dir, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,
EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vm As Double,
Wait As ASYNCALL ) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度= theta + Dir x 2PI , 其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例：

还可以看看:

APS_arc3_ce_all	根据所有曲线开始一个终点位置类型的 3D 运动
-----------------	-------------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_arc3_ce 的终端位置类型的 3D 圆弧插补。它跟随中心位置，终点位置和方向。_all 后缀表示执行 3D 圆弧插补时需要的所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 S-Factor。

句法:

C/C++:

```
I32 FNTYPE APS_arc3_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*EndArray, I16 Dir, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac,  
ASYNDCALL *Wait );
```

Visual Basic:

```
APS_arc3_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vs As Double,  
ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double,  
ByVal SFac As Double, Wait As ASYNDCALL ) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) /相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0] , 如果 dec <= 0 , 则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2 , 它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离> = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0
作为剩余距离的%值范围 : 0.0~1.0)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转 , dir=-1 则负向旋转。总旋转
角度= theta + Dir x 2PI , 其中 theta 是两个向量的角度 : 中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针 , 则调用将不等待 , 并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_spiral_ca

开始一个角度类型的 3D 螺旋运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x**描述:**

此函数用于执行名为_spiral_ca 的角度类型的 3D 螺旋插补。它跟随角度，中心位置，法向矢量，DeltaH 和 FinalR。没有任何后缀表示执行 3D 螺旋插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*NormalArray, F64 Angle, F64 DeltaH, F64 FinalR, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,
NormalArray As Double, ByVal Angle As Double, ByVal DeltaH As Double, ByVal FinalR
As Double, TransPara As Double, Wait As ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 DeltaH: 值指定高度。

F64 FinalR: 值指定从终点位置到法向量的距离。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看：

APS_spiral_ca_v

根据速度曲线开始一个角度类型的 3D 螺旋运动

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_spiral_ca 的角度类型的 3D 螺旋插补。它跟随角度，中心位置，法向矢量，DeltaH 和 FinalR。_v 后缀表示执行 3D 螺旋插补时仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 Angle, F64 DeltaH, F64 FinalR, F64 *TransPara, F64 Vm, ASYNCALL  
*Wait );
```

Visual Basic:

```
APS_spiral_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
NormalArray As Double, ByVal Angle As Double, ByVal DeltaH As Double, ByVal FinalR  
As Double, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL) As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的 % 值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 DeltaH: 值指定高度。

F64 FinalR: 值指定从终点位置到法向量的距离。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看：

APS_spiral_ca_all	根据所有曲线开始一个角度类型的 3D 螺旋运动
-------------------	-------------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于执行名为_spiral_ca 的角度类型的 3D 螺旋插补。它跟随角度，中心位置，法向矢量，DeltaH 和 FinalR。_all 后缀表示执行 3D 螺旋插补时需要的所有运动曲线，包括 Vs，Vm，Ve，Acc，Dec 和 S-Factor。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 Angle, F64 DeltaH, F64 FinalR, F64 *TransPara, F64 Vs, F64 Vm, F64  
Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As  
Double, NormalArray As Double, ByVal Angle As Double, F64 DeltaH, F64 FinalR,  
TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double,  
ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL)  
As Long
```

参数:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。注意：如果设置为模式 2，它将返回错误代码。

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0
作为剩余距离的%值范围 : 0.0 ~ 1.0)

Bit 16~: 保留以备将来使用 , 设置为 0.

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 Angle: 值指定角度。单位是弧度。

F64 DeltaH: 值指定高度。

F64 FinalR: 值指定从终点位置到法向量的距离。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_spiral_ce	开始一个终点位置类型的 3D 螺旋运动
支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于执行名为_spiral_ce 的终点位置类型的 3D 螺旋插补。它跟随中心位置，法向矢量，终点位置和 Dir。没有任何后缀表示执行 3D 螺旋插补时不需要任何运动曲线。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

基于 PCI(e)-8154/58，该函数支持 1 个固定半径的圆弧插补和法向轴上的同步线性运动。旋转角度是两个向量的 theta：从中心到起点和从中心到终点。与支持软件运动(soft motion)的 PCIe-8338 和 PCI-8254/58 有所不同，圆形插补角度范围只有-360°至 360°。这意味着螺旋曲线只有 1 个螺距，就像螺旋曲线一样。

注意：由于 8154/58 的限制，第 4 轴/第 8 轴操作将是虚拟动作，不能用于任何其他用途。该轴需要设置为伺服关闭。如果没有，它将返回 ERR_InServoOnState (-48)

例如

PCI(e)-8154，选择{0,1,2}为 APS_spiral_ce_v()，第 3 轴操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

PCI(e)-8158，选择{4,5,6}为 APS_spiral_ce_v()，轴 7 操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*NormalArray, F64 *EndArray, I16 Dir, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 FNTYPE APS_spiral_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray
As Double, NormalArray As Double, EndArray As Double, ByVal Dir As Short, TransPara
As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 Dir > = 0 表示正向旋转（逆时针），Dir < = -1（顺时针）沿负方向旋转。

F64 *TransPara: 指针指示传输参数的起始地址。**注意：它被保留供将来使用。**

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。**注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度)。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。**注意：如果设置为模式 2，它将返回错误代码。**

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的%值范围：0.0~1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度 = theta + Dir x 2PI，其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。**注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

以下示例适用于 PCI(e)-8154/58

```
I32 opt = 1; //相对模式
I32 Axis_ID_Array[3] = {0, 1, 2};
F64 NormalArray[3] = {0, 1, 0}; //选择轴 1 为垂直轴
F64 Center_Pos_Array[2] = {10000, 0, 20000};
F64 End_Pos_array[2] = {20000, 20000, 40000};
// 高度= End_Pos_array[1] - Center_Pos_Array[1]
I16 Dir = 1; //旋转逆时针
F64 TransPara = 0;
ASYNDCALL *wait = NULL;
```

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度
```

```
APS_spiral_ce (
    Axis_ID_Array           // I32 *Axis_ID_Array
    , opt                  // I32 Option
    , Center_Pos_Array     // F64 * CenterArray
    , NormalArray          // F64 * NormalArray
    , End_Pos_array        // F64 * Enday
    , Dir                  // I16 Dir
    , &TransPara           // 预留
    , wait );              // 预留
```

描述:

初始化:

起点: 0, 0, 0

输入:

选项: 1(相对)

中心点 (相对于起点) : 10000, 0, 20000

法向量: 0, 1, 0 (至正 y 轴)

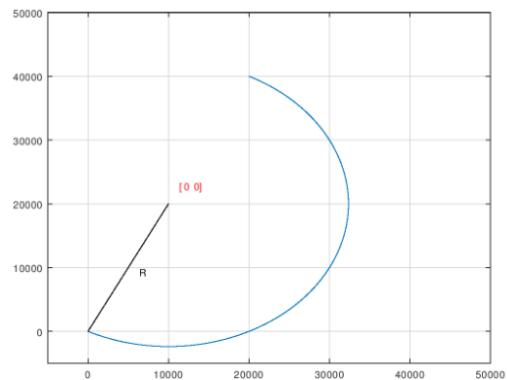
终点: 20000, 20000, 40000

输出:

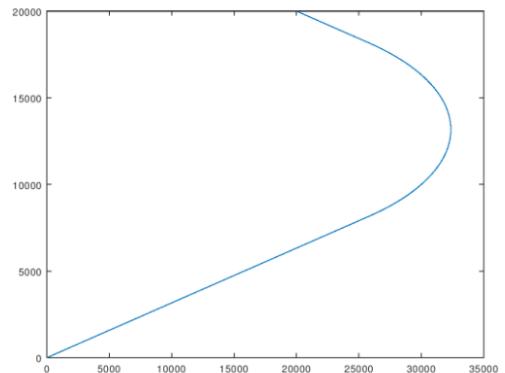
Theta: 180°, 逆时针

R: $\sqrt{10000 \cdot 10000 + 20000 \cdot 20000}$

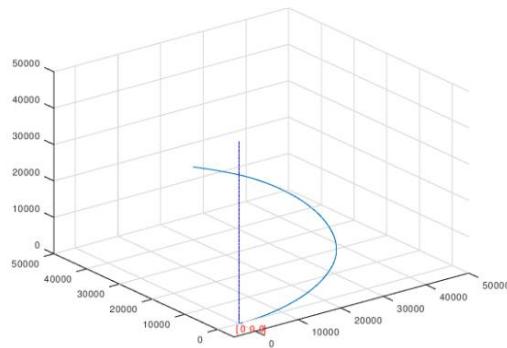
高度= 20000



XZ 平面



XY 平面



3D 视图

还可以看看:

APS_absolute_helical_move();APS_relative_helical_move()

APS_spiral_ce_v	开始一个终点位置类型的 3D 螺旋运动 with Vm profile
-----------------	-------------------------------------

支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于执行名为_spiral_ce 的终点位置类型的 3D 螺旋插补。它跟随中心位置，法向矢量，终点位置和 Dir。_v 后缀表示执行 3D 螺旋插补时仅需要一个运动曲线，即 Vm。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

基于 PCI(e)-8154/58，该函数支持 1 个固定半径的圆弧插补和法向轴上的同步线性运动。旋转角度是两个向量的 theta：从中心到起点和从中心到终点。与支持软件运动(soft motion)的 PCIe-8338 和 PCI-8254/58 有所不同，圆形插补角度范围只有-360°至 360°。这意味着螺旋曲线只有 1 个螺距，就像螺旋曲线一样。

注意：由于 8154/58 的限制，第 4 轴/第 8 轴操作将是虚拟动作，不能用于任何其他用途。该轴需要设置为伺服关闭。如果没有，它将返回 ERR_InServoOnState (-48)

例如

PCI(e)-8154，选择{0,1,2}为 APS_spiral_ce_v()，第 3 轴操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

PCI(e)-8158，选择{4,5,6}为 APS_spiral_ce_v()，轴 7 操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64  
*NormalArray, F64 *EndArray, I16 Dir, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double,  
NormalArray As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double,  
ByVal Vm As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)
15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 Dir > = 0 表示正向旋转（逆时针），Dir < = -1（顺时针）沿负方向旋转。

F64 *TransPara: 指针指示传输参数的起始地址。**注意：它被保留供将来使用。**

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。**注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C:

I32 *Axis_ID_Array: 指针指示轴数组的起始地址。

I32 Option: 一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度)。

[dec > 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。**注意：如果设置为模式 2，它将返回错误代码。**

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离 > = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的%值范围：0.0~1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度 = theta + Dir x 2PI，其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vm: 值指定最大速度。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 **注意：它被保留供将来使用。**

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

以下示例适用于 PCI(e)-8154/58

```
I32 opt = 1; //相对模式  
I32 Axis_ID_Array[3] = {0, 1, 2};  
F64 NormalArray[3] = {0, 1, 0}; //选择轴 1 为垂直轴  
F64 Center_Pos_Array[2] = {10000, 0, 20000};  
F64 End_Pos_array[2] = {20000, 20000, 40000};  
I16 Dir = 1; //旋转逆时针  
F64 TransPara = 0;  
F64 Vm = 10000;  
ASYNDCALL *wait = NULL;
```

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
```

```
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
```

```
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度
```

```
APS_spiral_ce_v ( // I32 *Axis_ID_Array  
    , opt // I32 Option  
    , Center_Pos_Array // F64 * CenterArray  
    , NormalArray // F64 * NormalArray  
    , End_Pos_array // F64 * Enday  
    , Dir // I16 Dir  
    , &TransPara // 预留  
    , Vm // Vm  
    , wait ); // 预留
```

还可以看看：

[APS_absolute_helical_move\(\)](#); [APS_relative_helical_move\(\)](#); [APS_spiral_ce\(\)](#)

APS_spiral_ce_all	开始一个终点位置类型的 3D 螺旋运动 with all profile
-------------------	--------------------------------------

支持的产品:PCI(e)-8154/58 , PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于执行名为_spiral_ce 的终点位置类型的 3D 螺旋插补。它跟随中心位置，法向矢量，终点位置和 Dir。_all 后缀表示执行 3D 螺旋插补时所需的所有运动曲线，包括 Vs , Vm , Ve , Acc , Dec 和 SFac。其他运动曲线在轴参数中设置。用户可以参考轴参数表中的详细信息。

基于 PCI(e)-8154/58，该函数支持 1 个固定半径的圆弧插补和法向轴上的同步线性运动。旋转角度是两个向量的 theta：从中心到起点和从中心到终点。与支持软件运动(soft motion)的 PCIe-8338 和 PCI-8254/58 有所不同，圆形插补角度范围只有-360°至 360°。这意味着螺旋曲线只有 1 个螺距，就像螺旋曲线一样。

注意：由于 8154/58 的限制，第 4 轴/第 8 轴操作将是虚拟动作，不能用于任何其他用途。该轴需要设置为伺服关闭。如果没有，它将返回 ERR_InServoOnState (-48)

例如

PCI(e)-8154 , 选择{0,1,2}为 APS_spiral_ce_v() , 第 3 轴操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

PCI(e)-8158 , 选择{4,5,6}为 APS_spiral_ce_v() , 轴 7 操作始终是伪动作，不能将其添加到* Axis_ID_Array 中。

句法:

C/C++:

```
I32 FNTYPE APS_spiral_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*NormalArray, F64 *EndArray, I16 Dir, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64
Acc,F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As
Double, NormalArray As Double, EndArray As Double, ByVal Dir As Short, TransPara As
Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As
Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

参数:

对于 PCI(e)-8154/58:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

注意：Axis_ID_Array 中指定的轴必须属于同一个板卡。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	0
							绝对(0) / 相对(1)

15	14	13	12	11	10	9	8

Bit 0: 1:相对运动 0:绝对运动

Bit 1~15: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray:指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。 如果 Dir> = 0 表示正向旋转（逆时针），Dir <= -1（顺时针）沿负方向旋转。

F64 *TransPara: 指针指示传输参数的起始地址。**注意：它被保留供将来使用。**

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。**注意：它被保留供将来使用。**

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

I32 *Axis_ID_Array:指针指示轴数组的起始地址。

I32 Option:一个位 (bit) 设置指定了选项，该选项可以启用指定的参数和函数。

7	6	5	4	3	2	1	Bit : 0
							绝对 (0) / 相对 (1)
15	14	13	12	11	10	9	8
缓冲模式							等待触发

Bit 0: 1:相对运动 0:绝对运动

Bit 1~7: 保留以备将来使用，设置为 0。

Bit 8: 设置为等待状态。 该轴直到被触发才运动。

Bit 9~11: 保留以备将来使用，设置为 0。

Bit 12~15: 缓冲模式:

0000b(0): 正在中止 (Aborting) – 停止并混合 (TransPara_0 作为减速速度。

[dec> 0]，如果 dec <= 0，则内核采用新的减速速度。)

0001b(1): 正在中止(Aborting) – 强制中止

0010b(2): 保留。**注意：如果设置为模式 2，它将返回错误代码。**

0011b(3): 已缓冲(Buffered)

0100b(4): 混合(Blending)-减速事件

0101b(5): 混合(Blending)-剩余距离 (TransPara_0 为残留距离> = 0.0)

0110b(6): 混合(Blending)-剩余距离 (以行进距离的百分比表示) (TransPara_0 作为剩余距离的%值范围 : 0.0~1.0)

Bit 16~: 保留以备将来使用，设置为 0。

F64 *CenterArray: 指针指示中心数组的起始地址。

F64 *NormalArray: 指针指示法向量数组的起始地址。

F64 *EndArray: 指针指示结束数组的起始地址。

I16 Dir: 值指定了旋转的方向。如果 dir 设置为 0 表示正向旋转，dir=-1 则负向旋转。总旋转角度= theta + Dir x 2PI，其中 theta 是两个向量的角度：中心到起点和中心到终点。

F64 *TransPara: 指针指示传输参数的起始地址。

F64 Vs: 值指定开始速度。

F64 Vm: 值指定最大速度。

F64 Ve: 值指定结束速度。

F64 Acc: 值指定加速度。

F64 Dec: 值指定减速度。

F64 SFac: 值指定 s-factor。

ASYNDCALL *Wait: 指向 ASYNDCALL 结构的指针。 注意：它被保留供将来使用。

用 NULL 传递将定义一个等待的调用。

如果它是一个有效的指针，则调用将不等待，并且函数将立即返回。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

以下示例适用于 PCI(e)-8154/58

```
I32 opt = 1; //相对模式
I32 Axis_ID_Array[3] = {0, 1, 2};
F64 NormalArray[3] = {0, 1, 0}; //选择轴 1 为垂直轴
F64 Center_Pos_Array[2] = {10000, 0, 20000};
F64 End_Pos_array[2] = {20000, 20000, 40000};
I16 Dir = 1; //旋转逆时针
F64 TransPara = 0;
ASYNDCALL *wait = NULL;
```

```
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1 ); //设置 S 曲线
APS_set_axis_param( Master_Axis_ID, PRA_ACC, 100000 ); //设置加速度
APS_set_axis_param( Master_Axis_ID, PRA_DEC, 100000 ); //设置减速度
```

```
APS_spiral_ce_all (
    Axis_ID_Array           // I32 *Axis_ID_Array
    , opt                  // I32 Option
    , Center_Pos_Array     // F64 * CenterArray
    , NormalArray          // F64 * NormalArray
```

```
, End_Pos_array          // F64 * Enday
, Dir                   // I16 Dir
, &TransPara            // 预留
, 0                     // Vs
, 10000                // Vm
, 0                     // Ve
, 11111                // 加速度
, 11111                // 减速度
, Sfac                 // 1, S-curve
, wait );              // 预留
```

还可以看看:

APS_absolute_helical_move();APS_relative_helical_move(); APS_spiral_ce()

11. 中断

APS_int_enable	中断主开关
----------------	-------

支持的产品:PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于开启/禁用一个板卡对主机的中断。这是板卡的硬件主开关。一旦禁用，即使启用了中断，主机也不会收到任何硬件中断。用户必须先启用此功能，然后才能使用任何与中断有关的函数，而当用户不再使用中断时，请禁用此功能。

句法:

C/C++:

```
I32 FNTYPE APS_int_enable( I32 Board_ID, I32 Enable );
```

Visual Basic:

```
APS_int_enable (ByVal Board_ID As Long, ByVal Enable As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Enable: 启用/禁用中断。

0: 禁用. 1: 启用

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Int_No; //中断编号  
I32 returnCode; //函数返回代码
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子  
APS_int_enable( Board_ID, 1 ); //启用中断主开关  
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断  
if( returnCode == ERR_NoError )  
{ //中断发生  
    APS_reset_int( Int_No );  
    ...//具体的工作  
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
```

```
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看：

```
APS_set_int_factor(); APS_get_int_factor();APS_wait_single_int();APS_wait_multiple_int();  
APS_reset_int(); APS_set_int();
```

APS_set_int_factor	启用/禁用中断并获取中断句柄
支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

该函数用于打开/关闭中断位。如果将其打开，该函数将为此位返回一个通知事件，并返回一个 I32 类型的事件号。用户可以通过将相应的事件编号分配给等待函数来等待该事件。事件编号在一个系统中是唯一的，但不是事件处理程序。这只是事件 APS 转换的虚拟编号。中断定义，请参考中断表。

句法:

C/C++:

```
I32 FNTYPE APS_set_int_factor( I32 Board_ID, I32 Item_No, I32 Factor_No, I32 Enable );
```

Visual Basic:

```
APS_set_int_factor (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No As Long, ByVal Enable As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Item_No: 中断表项目编号。请参考中断因子表。

I32 Factor_No: 项目的中断因子编号。请参考中断因子表。

I32 Enable: 启用中断。0 : 禁用 ; 1 : 启用

返回值:

When:

[Enable = 1] : 启用中断因子。

返回正值 : I32 中断事件编号。

返回负值 : I32 错误代码。请参考 [APS 函数返回代码](#).

[Enable = 0] : 禁用中断

返回 I32 错误代码 : 请参考 [APS 函数返回代码](#).

示例 1:

```
<设置 PCI-8392 或 PCI-8253/56 的第 2 轴为 NSTP 中断>
I32 Int_No; //中断编号
I32 returnCode; //函数返回代码

Int_No = APS_set_int_factor( Board_ID, Item_No=2, Factor_No=BIT12, 1 ); //启用中断因子
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
    ...//具体的工作
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

示例 2:

```
<设置 PCI-8254/58 的第 2 轴 IMDN 中断
I32 Int_No; //中断编号
I32 returnCode; //函数返回代码

Int_No = APS_set_int_factor( Board_ID, Item_No=2, Factor_No=BIT12, 1 ); //启用中断因子
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
    ...//具体的工作
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

APS_int_enable();APS_get_int_factor();APS_wait_single_int();APS_wait_multiple_int();
APS_reset_int(); APS_set_int()

<code>APS_get_int_factor</code>	获取中断因子的启用或禁用
支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述 :

此函数用于获取中断因子的设置。

句法:

C/C++:

```
I32 FNTYPE APS_get_int_factor( I32 Board_ID, I32 Item_No, I32 Factor_No, I32
*Enable );
```

Visual Basic:

```
APS_get_int_factor (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No
As Long, Enable As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 `APS_initial()` 来检索它。

I32 Item_No: 中断表项目编号。请参考中断因子表。

I32 Factor_No: 项目的中断因子编号。请参考中断因子表。

I32 *Enable: 返回启用或禁止。0: 禁止, 1:启用.

返回值:

I32 Error code: 请参考 `APS` 函数返回代码.

示例 :

```
I32 ReturnCode;
I32 Enable;
ReturnCode = APS_get_int_factor( Board_ID, Item_No, Factor_No, &Enable );
...

```

还可以看看:

`APS_int_enable();APS_set_int_factor();APS_wait_single_int();APS_wait_multiple_int();`
`APS_reset_int(); APS_set_int()`

APS_wait_single_int	等待单个中断事件
支持的产品:PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

当用户通过 “APS_set_int_factor” 为指定因素启用中断功能时，可以使用此函数等待指定的中断。 当运行此函数时，直到事件被触发或函数超时，该过程才会停止。发生以下情况之一时，此函数返回：

1. 指定的中断处于信号状态。
2. 超时间隔过去。

该函数检查指定中断的当前状态。如果状态为非信号状态，则调用线程进入等待状态。当等待 INT 状态变为信号或超时间隔过去时，该函数不占用处理器时间。

当发生中断并返回等待功能时，用户应使用 **APS_reset_int ()**自行重置中断。 如果用户不复位中断，等待功能将在下一次立即通过。

句法:

C/C++:

```
I32 FNTYPE APS_wait_single_int( I32 Int_No, I32 Time_Out );
```

Visual Basic:

```
APS_wait_single_int (ByVal Int_No As Long, ByVal Time_Out As Long) As Long
```

参数:

I32 Int_No: 中断事件编号。从 APS_set_int_factor() 函数中获取。

I32 Time_Out: 等待超时时间。单位为毫秒。如果将值设置为 -1，则函数的超时间隔将永远不会过去（无限）。如果 Time_Out 为零，则该函数测试中断的状态并立即返回。

返回值:

ERR_NoError(0): 该事件为等待成功。

I32 Error code: 请参考 APS 函数返回代码。

示例 :

```
I32 Int_No; //中断编号
I32 returnCode; //函数返回代码
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
```

```
    APS_reset_int( Int_No );
    ...//具体的工作
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

```
APS_int_enable();APS_set_int_factor();APS_get_int_factor();APS_wait_multiple_int();
APS_reset_int(); APS_set_int()
```

APS_wait_multiple_int	等待多个中断事件
支持的产品:PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C, PCIE-833x	

描述:

当用户通过 “APS_set_int_factor” 为指定因素启用中断功能时，可以使用此函数等待指定的中断。 当运行此函数时，直到事件被触发或函数超时，该过程才会停止。发生以下情况之一时，此函数返回：

1. 任何一个或所有中断都处于信号状态。
2. 超时间隔过去。

该函数检查指定中断的当前状态。如果状态为非信号状态，则调用线程进入等待状态。当等待 INT 状态变为信号或超时间隔过去时，该函数不占用处理器时间。

当发生中断并返回等待功能时，用户应使用 **APS_reset_int ()**自行重置中断。 如果用户不复位中断，等待功能将在下一次立即通过。

句法:

C/C++:

```
I32 FNTYPE APS_wait_multiple_int( I32 Int_Count, I32 *Int_No_Array, I32 Wait_All, I32 Time_Out );
```

Visual Basic:

```
APS_wait_multiple_int (ByVal Int_Count As Long, Int_No_Array As Long, ByVal Wait_All As Long, ByVal Time_Out As Long) As Long
```

参数:

I32 Int_Count: 指定的中断编号。最大编号为 64。

I32 *Int_No_Array: 中断事件编号数组。从 APS_set_int_factor() 函数获取。

I32 Wait_All: 等待选项。

 FALSE: (0) 当数组中任何一个事件的状态是信号时，函数返回此值。

 TRUE: (1) 当数组中所有事件的状态是信号时，函数返回此值。

I32 Time_Out: 等待超时时间。单位为毫秒。如果将值设置为 -1，则函数的超时间隔将永远不会过去（无限）。

返回值:

正值: (Int_Count - 1): 事件是等待成功。

 如果 Wait_All 为 FALSE(0)，则返回值表示所有指定对象的状态都是信号。

 如果 WaitAll 是 FALSE(0)，则返回值表示对象的数组索引符合等待。如果在调用期间有多个事件变为 185peratio，这就是 185peratio 对象的数组索引，而且是所有 185peratio 对象中最小的索引值。

负值: I32 error code: 请参考错误代码表。

示例 :

```
I32 Int_No[2]; //中断编号  
I32 returnCode; //函数返回代码
```

```
Int_No[0] = APS_set_int_factor( Board_ID, Item_No1, Factor_No1, 1 ); //启用中断因子  
Int_No[1] = APS_set_int_factor( Board_ID, Item_No2, Factor_No2, 1 ); //启用中断因子
```

```
APS_int_enable( Board_ID, 1 ); //启用中断主开关  
returnCode = APS_wait_multiple_int( 2, Int_No, 1, Time_Out ); //等待多个中断,(全部等待)  
if( returnCode >= ERR_NoError )  
{ //中断产生  
    APS_reset_int( Int_No[0] );  
    APS_reset_int( Int_No[1] );  
    ...//具体的工作  
}
```

```
APS_set_int_factor( Board_ID, Item_No1, Factor_No1, 0 ); //禁止中断因子  
APS_set_int_factor( Board_ID, Item_No2, Factor_No2, 0 ); //禁止中断因子  
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

```
APS_int_enable(); APS_set_int_factor();APS_get_int_factor();APS_wait_single_int();  
APS_reset_int(); APS_set_int()
```

APS_wait_error_int	等待错误中断 (非屏蔽)
支持的产品:PCI(e)-8154/8158, PCI-8102/PCI-C154(+)	

描述:

用户可以使用此函数来等待错误中断。运行此函数时，直到事件被触发或函数超时，该过程才会停止。发生以下情况之一时，此函数返回：

1. 任何一个或所有错误中断都处于信号状态。
2. 超时间隔过去。

此函数检查错误中断的当前状态。如果状态为非信号状态，则调用线程进入等待状态。在等待 INT 状态变成信号或超时间隔过去时，它不使用处理器时间。当发生错误中断时，等待函数返回。

句法:

```
I32 FNTYPE APS_wait_error_int( I32 Board_ID, I32 Item_No, I32 Time_Out );
APS_wait_single_int (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Time_Out
As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Item_No: 中断表项目编号。请参考中断因子表。

I32 Time_Out: 等待超时时间。单位为微秒。如果将值设置为 -1，则函数的超时间隔将永远不会过去（无限）。如果 *Time_Out* 为零，则该函数测试中断的状态并立即返回。

返回值:

当:

[Enable = 1] : 启用中断

 返回正值 : I32 错误中断事件编号或 Time_Out。

 返回负值 : I32 错误代码。请参考 APS 函数返回代码。

[Enable = 0] : 禁用中断

 I32 Error code: 请参考 APS 函数返回代码。

示例 :

```
I32 returnCode; //函数返回代码
```

```
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = APS_wait_error_int(Board_ID , Item_No, Time_Out ); //等待错误中断
if( returnCode >= 0 )
{
    //中断产生或 Time_Out
```

```
//具体的工作  
}  
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

```
APS_int_enable();APS_set_int_factor();APS_get_int_factor();APS_wait_single_int();  
APS_wait_multiple_int(); APS_reset_int(); APS_set_int()
```

APS_reset_int	将中断事件重置为非信号状态。
支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

此函数用于将信号事件重置为非信号状态。

句法:

C/C++:

```
I32 FNTYPE APS_reset_int( I32 Int_No );
```

Visual Basic:

```
APS_reset_int (ByVal Int_No As Long) As Long
```

参数:

I32 Int_No: 中断事件编号。从 APS_set_int_factor() 函数中获取。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 Int_No; //中断编号
I32 returnCode; //函数返回代码
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
    ...//具体的工作
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

APS_int_enable();APS_set_int_factor();APS_get_int_factor();APS_wait_single_int();
APS_wait_multiple_int(); APS_set_int()

APS_set_int	将中断事件设置为信号状态。
-------------	---------------

支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x

描述 :

该函数用于将指定事件中断信号化。 设置此函数后，等待函数将返回（通过）。

句法:

C/C++:

```
I32 FNTYPE APS_set_int( I32 Int_No );
```

Visual Basic:

```
APS_set_int (ByVal Int_No As Long) As Long
```

参数:

I32 Int_No: 中断事件编号。从 APS_set_int_factor() 函数中获取。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 Int_No; //中断编号
```

```
I32 returnCode; //函数返回代码
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子
```

```
APS_int_enable( Board_ID, 1 ); //启用中断主开关
```

```
APS_set_int(Int_No); //信号化中断事件
```

```
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待函数将立即通过。
```

```
if( returnCode == ERR_NoError )
```

```
{ //中断发生
```

```
    APS_reset_int( Int_No );
```

```
    ...//具体的工作
```

```
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
```

```
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

APS_int_enable();APS_set_int_factor();APS_get_int_factor();APS_wait_single_int();

APS_wait_multiple_int(); APS_reset_int()

APS_set_int_factorH	启用/禁用中断并获取中断句柄。 (Win32)
支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

该函数用于打开/关闭中断位。如果打开，该功能将为此位返回一个通知事件并返回一个 HANDLE 类型（在 windows.h 中定义）的事件句柄。用户可以直接使用 Win32 API 函数使用此句柄。事件编号在一个系统中是唯一的。

中断的定义，请参考中断表。

句法:

C/C++:

```
HANDLE APS_set_int_factorH( I32 Board_ID, I32 Item_No, I32 Factor_No, I32 Enable );
```

Visual Basic:

```
APS_set_int_factorH (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No As Long, ByVal Enable As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Item_No: 中断表项目编号。请参考中断因子表。

I32 Factor_No: 项目的中断因子编号。请参考中断因子表。

I32 Enable: 启用中断。0：禁用；1：启用

返回值:

When:

[Enable = 1]：启用中断因子。

如果函数成功，则返回 win32 事件句柄，否则返回 null (0)。

[Enable = 0]：禁用中断

返回 null(0)。

示例 :

```
#include <windows.h>
HANDLE hInt; //中断句柄
DWORD returnCode; //函数返回代码
```

```
hInt = APS_set_int_factorH( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子
APS_int_enable( Board_ID, 1 ); //启用中断主开关
returnCode = WaitForSingleObject( hInt, 1000 );
```

```
if( returnCode == WAIT_OBJECT_0 )
{ //中断发生
    ResetEvent (hInt ); //Win32 SDK 函数
    ...//具体的工作
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

APS_int_enable();APS_get_int_factor();APS_wait_single_int();APS_wait_multiple_int();
APS_reset_int(); APS_set_int()

APS_int_no_to_handle	将中断事件数字编号转换为中断句柄。 (Win32)
支持的产品 : PCI-8253/56, PCI-8392(H), DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述 :

此函数用于将中断编号转换为 HANDLE 类型 (在 windows.h 中定义) 的事件句柄。 用户可以通过 APS_set_factor() 获取 I32 类型的事件编号，然后将该编号转换为 HANDLE。

句法:

C/C++:

```
HANDLE APS_int_no_to_handle( I32 Int_No );
```

Visual Basic:

```
APS_int_no_to_handle( ByVal Int_No As Long ) As Long
```

参数:

I32 Int_No: 中断事件编号。从 APS_set_int_factor() 函数中获取。

返回值:

返回 win32 事件句柄。

示例 :

```
#include <windows.h>
HANDLE hInt; //中断句柄
I32 Int_No;
DWORD returnCode; //函数返回代码

Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //启用中断因子
hInt = APS_int_no_to_handle( Int_No ); //转换为一个句柄。
APS_int_enable( Board_ID, 1 ); //启用中断主开关

returnCode = WaitForSingleObject( hInt, 1000 );

if( returnCode == WAIT_OBJECT_0 )
{ //中断发生
    ResetEvent (hInt); //Win32 SDK function
    ...//具体的工作
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //禁止中断因子
```

```
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看：

```
APS_int_enable(); APS_set_int_factor(); APS_set_field_bus_int_factor_motion ()
```

APS_set_field_bus_int_factor_motion	在 PCI(e)-7856 上为 MotionNet 系列启用/禁用错误中断并获取中断句柄
-------------------------------------	---

支持的产品 : PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO

描述 :

此函数用于打开/关闭 MNET 产品上的中断位。如果将其打开，该函数将为此位返回一个通知事件，并返回一个 I32 类型的事件编号。用户可以通过将相应的事件编号分配给等待功能来等待该事件。事件编号在一个系统中是唯一的，但不是事件句柄。这只是事件 APS 转换的虚拟数量。

MotionNet 运动中断的定义，请参考中断表。

请注意，一定要通过调用 APS_initial() 将其设置为中断模式，将 bit 6 设置为 1。请注意，您应该在启动现场总线后调用此函数。确保通过调用 APS_start_field_bus() 构建了 MENT 现场总线的所有轴。然后，用户可以使用此函数将中断设置给指定轴。否则，将返回错误代码。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_int_factor_motion( I32 Axis_ID, I32 Factor_No, I32
Enable );
```

Visual Basic:

```
APS_set_field_bus_int_factor_motion ( ByVal Axis_ID As Long, ByVal Factor_No As
Long, ByVal Enable As Long) As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Factor_No: 轴的中断编号。请参考中断因子表。

I32 Enable: 启用中断。0 : 禁用 ; 1 : 启用

返回值:

When:

[Enable = 1] : 启用中断因子。

返回正值 : I32 中断事件编号。

返回负值 : I32 错误代码。请参考 APS 函数返回代码.

[Enable = 0] : 禁用中断

返回 I32 错误代码 : 请参考 APS 函数返回代码.

示例 :

<在 PCI(e)-7856 的 MotionNet 现场总线上将轴 1000 INSTP(BIT 0) 中断设置为开 >

I32 Axis_ID = 1000; //MNET 的轴

```
I32 returnCode; //函数返回代码

//启用中断因子
Int_No = APS_set_field_bus_int_factor_motion ( Board_ID, Axis_ID, Factor_No=0, 1 );
APS_int_enable( Board_ID, 1 ); //启用中断主开关
//重置轴的中断状态
APS_reset_field_bus_int_motion ( Axis_ID );
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
    ...//具体的工作
}

APS_set_field_bus_int_factor_motion ( Axis_ID, Factor_No, 0 ); //禁止中断因子
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

```
APS_int_enable();APS_get_field_bus_int_factor_motion();APS_wait_single_int();
APS_wait_multiple_int(); APS_reset_int(); APS_set_int()
```

APS_get_field_bus_int_factor_motion	在 PCI(e)-7856 上为 MotionNet 系列获取运动中断启用或禁用
-------------------------------------	--

支持的产品 : PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO

描述 :

此函数用于获取中断的设置。

请注意 , 您应该在启动现场总线后调用此函数。确保通过调用 APS_start_field_bus() 构建了端口的所有轴。然后 , 用户可以使用该函数从指定轴获取中断。否则 , 将返回错误代码。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_int_factor_motion( I32 Axis_ID, I32 Factor_No, I32
*Enable );
```

Visual Basic:

```
APS_get_field_bus_int_factor_motion ( ByVal Axis_ID As Long, ByVal Factor_No As
Long, Enable As Long) As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Factor_No: 轴的中断编号。请参考中断因子表。

I32 *Enable: 返回启用或禁止。0: 禁止, 1: 启用.

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 ReturnCode;
I32 Enable;
ReturnCode = APS_get_field_bus_int_factor_motion ( Axis_ID, Factor_No, &Enable );
...

```

还可以看看:

APS_int_enable();APS_set_field_bus_int_factor_motion();APS_wait_single_int();
 APS_wait_multiple_int(); APS_reset_int(); APS_set_int()

APS_set_field_bus_int_factor_error	在 PCI(e)-7856 上为 MotionNet 系列启用/禁用错误中断并获取中断句柄
------------------------------------	---

支持的产品 : PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO

描述 :

此函数用于打开/关闭 MNET 产品上的错误中断位。如果将其打开，该函数将为此位返回一个通知事件，并返回一个 I32 类型的事件编号。用户可以通过将相应的事件编号分配给等待功能来等待该事件。事件编号在一个系统中是唯一的，但不是事件句柄。这只是事件 APS 转换的虚拟数量。

MotionNet 错误中断的定义，请参考中断因素表。

请注意，所有默认错误中断均已打开。

请注意，一定要通过调用 APS_initial() 将其设置为中断模式，将 bit 6 设置为 1。请注意，您应该在启动现场总线后调用此函数。确保通过调用 APS_start_field_bus() 构建了 MENT 现场总线的所有轴。然后，用户可以使用此函数将中断设置给指定轴。否则，将返回错误代码。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_int_factor_error( I32 Axis_ID, I32 Factor_No, I32 Enable );
```

Visual Basic:

```
APS_set_field_bus_int_factor_error( ByVal Axis_ID As Long, ByVal Factor_No As Long,  
ByVal Enable As Long) As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Factor_No: 轴的中断编号。参考中断表

I32 Enable: 启用中断。0：禁用；1：启用

返回值:

当:

[Enable = 1]：启用错误中断

返回正值：I32 中断事件编号。

返回负值：I32 错误代码。请参考 APS 函数返回代码。

[Enable = 0]：禁用错误中断

返回 I32 错误代码：请参考 APS 函数返回代码。

示例 :

<在 PCI(e)-7856 的 MotionNet 现场总线上将轴 EPEL (BIT 5) 错误中断设置为开 >

I32 Axis_ID = 1000; //MNET 的轴。

I32 returnCode; //函数返回代码

```
//启用中断因子
Int_No = APS_set_field_bus_int_factor_error ( Board_ID, Axis_ID, Factor_No=5, 1 );
APS_int_enable( Board_ID, 1 ); //启用中断主开关
//重置轴的中断状态
APS_reset_field_bus_int_motion ( Axis_ID );
returnCode = APS_wait_single_int( Int_No, Time_Out ); //等待中断
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
    ...//具体的工作
}

APS_set_field_bus_int_factor_error ( Axis_ID, Factor_No, 0 ); //禁用错误中断
APS_int_enable( Board_ID, 0 ); //禁用中断主开关
```

还可以看看:

APS_int_enable();APS_set_field_bus_int_factor_motion();APS_get_field_bus_int_factor_motion(); APS_wait_single_int();APS_wait_multiple_int();APS_reset_int();APS_set_int();
APS_get_field_bus_int_factor_error(); APS_wait_field_bus_error_int_motion()

APS_get_field_bus_int_factor_error	在 PCI(e)-7856 上为 MotionNet 系列获取错误中断的状态
------------------------------------	--

支持的产品 : PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO

描述 :

此函数用于获取错误中断的设置。

请注意 , 所有默认错误中断均已打开。

请注意 , 您应该在启动现场总线后调用此函数。确保通过调用 APS_start_field_bus() 构建了 MENT 现场总线的所有轴。然后 , 用户可以使用此函数将中断设置给指定轴。否则 , 将返回错误代码。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_int_factor_error ( I32 Axis_ID, I32 Factor_No, I32
*Enable );
```

Visual Basic:

```
APS_get_field_bus_int_factor_error ( ByVal Axis_ID As Long, ByVal Factor_No As Long,
Enable As Long) As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Factor_No: 轴的中断编号。参考中断表

I32 *Enable: 返回启用或禁止。0: 禁止, 1: 启用

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 ReturnCode;
I32 Enable;
ReturnCode = APS_get_field_bus_int_factor_error ( Axis_ID, Factor_No, &Enable );
...

```

还可以看看:

APS_int_enable();APS_set_field_bus_int_factor_motion();APS_get_field_bus_int_factor_motion(); APS_wait_single_int();APS_wait_multiple_int();APS_reset_int();APS_set_int();
APS_get_field_bus_int_factor_error (); APS_wait_field_bus_error_int_motion()

APS_reset_field_bus_int_motion	在 PCI(e)-7856 上重置 MotionNet 系列轴的中断状态
--------------------------------	--------------------------------------

描述：

该函数用于复位轴的中断状态。

用户通过“APS_int_enable()”启用中断函数后，用户应使用此函数来复位保留在从站模块中的中断状态。从站模块中残留的中断状态将导致意外的过程，例如中断中断机制。用户启用了中断功能函数后，请务必复位那些轴的中断状态。

句法:

C/C++:

```
I32 FNTYPE APS_reset_field_bus_int_motion ( I32 Axis_ID );
```

Visual Basic:

```
APS_reset_field_bus_int_motion ( ByVal Axis_ID As Long ) As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Time_Out: 等待超时时间。单位为毫秒。如果将值设置为 -1，则函数的超时间隔将永远不会过去（无限）。

返回值:

正值: (Int_Count - 1):事件是等待成功。

返回值表示符合等待条件的错误事件的索引。如果在调用期间有多个事件变为 202peratio，这就是已转化为信号事件的数组索引，而且是所有信号化事件中最小的索引值。

负值: I32 error code: 请参考 APS 函数返回代码.

示例：

.. 设置轴中断

```
APS_int_enable( Board_ID, 1 ); //启用中断主开关  
//重置轴的中断状态  
APS_reset_field_bus_int_motion ( Axis_ID );
```

.. 等待事件

还可以看看:

APS_int_enable();APS_set_field_bus_int_factor_motion();APS_get_field_bus_int_factor_motion()

APS_wait_field_bus_error_int_motion	在 PCI(e)-7856 上为 MotionNet 系列等待错误中断事件。
-------------------------------------	--

支持的产品 : PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO

描述 :

此函数用于等待错误中断事件。

当用户通过 “ APS_int_enable()” 启用中断函数时，用户可以使用该函数来等待错误中断。运行此函数时，直到事件被触发或函数超时，该过程才会停止。发生以下情况之一时，此函数返回：

1. 任何一个或所有错误中断都处于信号状态。
2. 超时间隔过去。

此函数检查错误中断的当前状态。如果状态为非信号状态，则调用线程进入等待状态。在等待 INT 状态变成信号或超时间隔过去时，它不使用处理器时间。

如果触发了任何一个错误中断，则系统将自动重置该事件。

MotionNet 错误中断的定义，请参考中断表。

请注意，所有默认错误中断均已打开。

请注意，“APS_set_field_bus_int_factor_error()” 可能会关闭错误中断位。

句法:

C/C++:

```
I32 FNTYPE APS_wait_field_bus_error_int_motion( I32 Axis_ID, I32 Time_Out );
```

Visual Basic:

```
APS_wait_field_bus_error_int_motion ( ByVal Axis_ID As Long, ByVal Time_Out As Long )  
As Long
```

参数:

I32 Axis_ID: MNET 系统的专用轴。

I32 Time_Out: 等待超时时间。单位为毫秒。如果将值设置为 -1，则函数的超时间隔将永远不会过去（无限）。

返回值:

正值: 事件是等待成功。

返回值表示符合等待条件的错误事件的索引。如果在调用期间有多个事件变为 205peratio，这就是已转化为信号事件的数组索引，而且是所有信号化事件中最小的索引值。

负值: I32 Error code: 请参考 APS 函数返回代码。

示例 :

```
I32 ReturnCode;
```

```
I32 Time_Out = 1000;(意思是 1000 ms)
```

```
ReturnCode = APS_wait_field_bus_error_int_motion ( Axis_ID, Time_Out );
```

...

还可以看看：

APS_int_enable();APS_set_field_bus_int_factor_error();APS_get_field_bus_int_factor_error(); APS_reset_field_bus_int_motion()

APS_set_field_bus_int_factor_di	分配数字输入中断位并为 HSL 系列获得中断句柄
---------------------------------	--------------------------

支持的产品:PCI(e)-7856

描述:

该函数用于分配 HSL DI 中断位，并为 HSL DI 模块返回一个 I32 类型的事件编号。当分配的位的状态改变时（无论是 1 到 0，还是 0 到 1），您都可以通过事件编号等待中断事件。事件编号在一个系统中是唯一的，但不是事件句柄。这只是事件 APS 转换的虚拟数量。请注意，一个 DIO 模块只有一个事件编号。

句法:

C/C++:

```
APS_set_field_bus_int_factor_di ( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32  
bitsOfCheck );
```

Visual Basic:

```
APS_set_field_bus_int_factor_di ( ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal bitsOfCheck As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它 I32

BUS_No: 现场总线编号。值：0~1，在 PCI(e)-7856 中，该值必须为 0。

I32 MOD_No: HSL 从站模块占用的第一个 ID。不能为 0。

I32 bitsOfCheck: 此参数与位格式一起使用。该 206peratio 分配了可能导致从站模块发生二次中断的位。如果从站模块输入的位数超过 16 位，则高位字用于 bit16~31，低位字用于 bit0~15。

请注意，下一个 bitsOfCheck 会覆盖之前的 bitsOfCheck。

返回值:

返回正值：I32 中断事件编号。

返回负值：I32 错误代码。请参考 APS 函数返回代码。

Negative value: I32 Error code: 请参考 APS 函数返回代码。

示例：

在下面这个示例中，当 DI32 从站模块上的任何位更改时，DI 中断就会发生。该模块占用 ID 1。

```
I32 Module_No = 1;  
I32 BUS_No = 0;  
I32 IntNo; //中断编号  
I32 returnCode; //函数返回代码  
I32 bitsOfCheck = 0xffffffff;
```

```
//1. 启用中断
APS_int_enable( Board_ID, Enable );

//2. 中断设置
IntNo = APS_set_field_bus_int_factor_di ( Board_ID, BUS_No, MOD_No, bitsOfCheck );

//3. 等待中断
returnCode = APS_wait_single_int( IntNo, 10000 ); //等待 10 秒
If( ret == 0 ) //获取中断
{
    ....// 具体的工作
}
//清除中断
APS_reset_int( IntNo );
```

还可以看看:

```
APS_int_enable();APS_get_field_bus_int_factor_di();APS_wait_single_int();
APS_wait_multiple_int(); APS_reset_int(); APS_set_int()
```

APS_get_field_bus_int_factor_di	获取分配的数字输入中断位
---------------------------------	--------------

支持的产品:PCI(e)-7856

描述:

此函数用于获取分配的数字输入中断位。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_int_factor_di( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 *bitsOfCheck );
```

Visual Basic:

```
APS_get_field_bus_int_factor_di ( ByVal Board_ID As Long, ByVal BUS_No, ByVal  
MOD_No As Long, ByRef bitsOfCheck As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。 通过成功调用 APS_initial() 来检索它 I32

BUS_No: 现场总线编号。 值 : 0~1 , 在 PCI(e)-7856 中 , 该值必须为 0.

I32 MOD_No: HSL 从站模块占用的第一个 ID。 不能为 0.

I32 *bitsOfCheck: 返回数字输入中断位。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 ReturnCode;
```

```
I32 bitsOfCheck;
```

```
ReturnCode = APS_get_field_bus_int_factor_di ( Board_ID,  BUS_No, MOD_No,  
&bitsOfCheck);
```

```
...
```

还可以看看:

APS_int_enable();APS_set_field_bus_int_factor_di();APS_wait_single_int();

APS_wait_multiple_int(); APS_reset_int(); APS_set_int()

12. 采样

APS_set_sampling_param	设置采样参数
支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO , PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于设置采样参数，例如采样率，采样通道源等。有关其定义和详细说明，请参阅采样参数表。

对于 PCI-8253/56 和 PCI-8392(H)以及 PCI-8254/58 / AMP-204/8C 和 PCIe-833x，采样函数仅适用于内部装有 DSP 或 CPU 的板卡，以符合实时性的需求。采样函数确保每个采样点在恶劣的实时环境下也能都被记录。

在 MNET-4XMO 上，采样函数是基于系统计时器的。因此，系统状态将影响采样数据的准确性。根据我们的测试，设置的采样率越高，获得的精度越差。

句法:

C/C++:

```
I32 FNTYPE APS_set_sampling_param( I32 Board_ID, I32 Param_No, I32 Param_Dat );
```

Visual Basic:

```
APS_set_sampling_param( ByVal Board_ID As Long, ByVal ParaNum As Long, ByVal  
ParaDat As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Param_No: 指定的采样参数编号，定义请参考采样参数表。

I32 Param_Dat: 采样编号对应的参数值。请参阅采样表。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
//... 初始化板卡.
```

```
I32 Ret = APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2 ); //设置采样率
```

```
...
```

还可以看看:

APS_get_sampling_param();APS_wait_trigger_sampling()

APS_get_sampling_param	获取采样参数
------------------------	--------

支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO, PCI-8254/58 / AMP-204/8C, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于获取采样的参数，例如采样率，采样通道源等。有关其定义和详细说明，请参阅抽样参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_sampling_param( I32 Board_ID, I32 ParaNum, I32 *ParaDat );
```

Visual Basic:

```
APS_get_sampling_param( ByVal Board_ID As Long, ByVal ParaNum As Long, ParaDat  
As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 ParaNum: 采样参数编号，定义请参考采样参数表。

I32 *ParaDat: 返回采样参数值。请参阅采样参数表。

返回值:

I32 Error code: 请参考 APS 函数返回代码。

示例 :

I32 ParaDat:

```
Ret = APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, & ParaDat ); // 获取触发沿  
...
```

还可以看看:

APS_set_sampling_param();APS_wait_trigger_sampling()

APS_wait_trigger_sampling	等待采样数据
支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO, PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于从控制器重对数据进行采样。发布该函数后，程序将声明对信息进行采样并将数据放入内部缓冲区。在触发信号开启之前，程序从内部缓冲区到用户的数据缓冲区中提取了大小为预触发长度的大量数据，并连续采样数据，直到达到用户指定的长度为止。另一方面，如果达到超时时间并且触发信号未升起，则此函数将超时并返回错误消息。

使用 **APS_stop_wait_sampling** 强制停止等待采样。

警告:

APS_wait_trigger_sampling 和 **APS_wait_trigger_sampling_async** 和 **APS_auto_sampling** 函数不能同时使用。

句法:

C/C++:

```
I32 FNTYPE APS_wait_trigger_sampling( I32 Board_ID, I32 Length, I32 PreTrgLen, I32 TimeOutMs, STR_SAMP_DATA_4CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling(ByValBoard_ID As Long, ByValLength As Long, ByValPreTrgLen As Long, ByValTimeOutMs As Long, DataArr As STR_SAMP_DATA_4CH ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 **APS_initial()** 来检索它。

I32 Length: 采样数据数号。（数组大小）

I32 PreTrgLen: 预触发长度。

I32 TimeOutMs: 超时时间。单位为毫秒。

STR_SAMP_DATA_4CH *DataArr: 获取采样数据结构数组。数组大小必须大于参数“Length”的值。

返回值:

I32 Error code: 请参考 **APS** 函数返回代码。

示例 :

//... 初始化板卡.

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2 ); //设置采样率
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0 ); //设置触发沿 ( 上升沿 )
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL, 1); //设置触发电平(1)
```

```
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH, 0 ); //设置触发通道(channel 0)
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //设置
channel_0 的采样源
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //设置
channel_1 的采样源
I32 Length = 1024; //总的采样数据数组大小。
I32 PreTrgLen = 100; //预触发点编号 STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 秒超时
Ret =APS_wait_trigger_sampling( Board_ID, Length, PreTrgLen, TimeOutMs, DataArr );
If( Ret == ERR_NoError )
{ //采样成功
    // DataArr 准备使用。
}
```

还可以看看:

```
APS_set_sampling_param(); APS_get_sampling_param(); APS_stop_wait_sampling()
```

APS_wait_trigger_sampling_async	等待异步样本数据
---------------------------------	----------

支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于从控制器中采样数据。此函数将立即返回，并创建一个后台线程来采样数据。

使用 APS_get_sampling_count 函数获取要采样的数据计数。当采样计数达到数据长度 (length)时，表示采样完成。如果样本计数= -1，则表示等待失败。

使用 APS_stop_wait_sampling 函数强制停止异步等待采样。采样计数将变为-1。

警告:

APS_wait_trigger_sampling 和 APS_wait_trigger_sampling_async 和 APS_auto_sampling 函数不能同时使用。

句法:

C/C++:

```
I32 FNTYPE APS_wait_trigger_sampling_async( I32 Board_ID, I32 Length, I32 PreTrgLen,  
I32 TimeOutMs, STR_SAMP_DATA_4CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_async(ByVal Board_ID As Long, ByVal Length As Long,  
ByVal PreTrgLen As Long, ByVal TimeOutMs As Long, DataArr As  
STR_SAMP_DATA_4CH )As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它

I32 Length: 采样数据数号。(数组大小)

I32 PreTrgLen: 预触发长度。

I32 TimeOutMs: 超时时间。单位为毫秒。

STR_SAMP_DATA_4CH *DataArr: 获取采样数据结构数组。数组大小必须大于参数“Length”的值。

返回值:

I32 Error code: 请参考 APS 函数返回代码。

示例 :

```
//... 初始化板卡.
```

```
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0 ); //设置触发沿 ( 上升沿 )  
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL, 1 ); //设置触发电平(1)  
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH, 0); //设置触发通道(channel 0)
```

```

APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //设置
channel_0 的采样源
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //设置
channel_1 的采样源

//开始异步等待采样.
I32 Length = 1024; //总的采样数据数组大小。
I32 PreTrgLen = 100; //预触发点编号 STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 秒超时
I32 Ret;

Ret =APS_wait_trigger_sampling_async( Board_ID, Length, PreTrgLen, TimeOutMs,
DataArr );

if( Ret != ERR_NoError )
{
    //显示错误信息
}else
{
    while( count < Length )
    {
        APS_get_sampling_count( Board_ID, &count );
        If( count == -1 )
        {
            //采样失败,
            // 中断程序.:
        }

        If( ForceStop )
        {
            APS_stop_wait_sampling(Board_ID);
        }
    }

    If( count == Length )
    {
        //采样成功
        // DataArr 准备使用.
    }
}

```

还可以看看:

`APS_get_sampling_count(); APS_wait_trigger_sampling(); APS_stop_wait_sampling()`

<code>APS_get_sampling_count</code>	获取采样数据计数
-------------------------------------	----------

支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于获取异步等待采样数据计数。

第一步，使用 [APS_wait_trigger_sampling_async](#) 启动采样操作，您需要获取采样计数以检查该操作是成功完成还是失败。

第二步，使用 [APS_auto_sampling](#) 启动采样操作，用户可以获得采样计数。

句法:

C/C++:

```
I32 FNTYPE APS_get_sampling_count( I32 Board_ID, I32 *SampCnt );
```

Visual Basic:

```
APS_get_sampling_count(ByVal Board_ID As Long, SampCnt As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 [APS_initial\(\)](#) 来检索它。

I32 *SampCnt: 返回采样数据计数。如果返回-1，则表示采样失败。

返回值:

I32 Error code: 请参考 [APS](#) 函数返回代码.

示例 :

参考 [APS_wait_trigger_sampling_async](#) 示例.

还可以看看:

[APS_set_sampling_param\(\)](#); [APS_get_sampling_param\(\)](#); [APS_stop_wait_sampling\(\)](#);
[APS_wait_trigger_sampling\(\)](#); [APS_wait_trigger_sampling_async\(\)](#)

APS_stop_wait_sampling	强制停止等待采样
------------------------	----------

支持的产品:PCI-8253/56, PCI-8392(H), MNET-4XMO, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于强制停止 [APS_wait_trigger_sampling](#) 和 [APS_wait_trigger_sampling_async](#) 函数。

句法:

C/C++:

```
I32 FNTYPE APS_stop_wait_sampling( I32 Board_ID );
```

Visual Basic:

```
APS_stop_wait_sampling(ByVal Board_ID As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 [APS_initial\(\)](#) 来检索它。

返回值:

I32 Error code: 请参考 [APS](#) 函数返回代码.

示例 :

参考 [APS_wait_trigger_sampling_async](#) 示例

还可以看看:

[APS_wait_trigger_sampling\(\)](#); [APS_wait_trigger_sampling_async\(\)](#)

APS_auto_sampling	开启/停止自动采样
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于实现自动采样操作。它将创建一个后台线程来采样数据。

用户可以使用 [**APS_get_sampling_data**](#) / [**APS_get_sampling_data_ex**](#) 获取采样数据并监视内部缓冲区的状态。可以监视缓冲区的四个状态，包括“STOP”，“WORK”，“EMPTY”和“FULL”。使用 [**APS_get_sampling_count**](#) 函数获取所采样数据的计数。如果样本计数 = -1，则表示自动采样已停止。

启用后，一个采样数据线程开始运行，并且还将建立了一个内部缓冲区来存储采样数据。这个缓冲区的存储空间有限，最多可以包含 65535 个采样数据，用户需要使用

[**APS_get_sampling_data\(\)**](#) 连续获取采样数据。用户需要连续消除缓冲区的数据，这样缓冲区就可以重新使用以存储更多的采样数据。

一般来说，缓冲区始终处于工作(WORK)状态。这就意味着不可能丢失任何采样数据，并且需要保证用户端获取数据的轮询频率是合适的。

如果内部缓冲区中充满了从 DSP 端采样的数据，则可能会丢失采样数据。例如，在 FULL 状态下，可能会抛出来自 DSP 的顺序采样数据，直到消耗掉缓冲区部分数据为止，用户才可以重新获取缓冲区的数据。

另一方面，如果内部缓冲区处于 EMPTY 状态，则意味着从缓冲区获取数据比从 DSP 端采样要快。用户可以消除获取数据的频率，将轮询计时器更改为较慢的一个。这样系统将提升 CPU 性能去做其他的工作。

警告：这些是一组用于自动采样的 API 函数，包括 [**APS_auto_sampling\(\)**](#) 和 [**APS_get_sampling_data\(\)**](#) / [**APS_get_sampling_data_ex\(\)**](#)。不要与其他触发功能，例如 [**APS_wait_trigger_sampling\(\)**](#), [**APS_wait_trigger_sampling_async\(\)**](#) 和 [**APS_stop_wait_sampling\(\)**](#) 混合使用。

句法:

C/C++:

```
I32 FNTYPE APS_auto_sampling( I32 Board_ID, I32 StartStop );
```

Visual Basic:

```
APS_auto_sampling (ByVal Board_ID As Long, ByVal StartStop As Long )As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 [**APS_initial\(\)**](#) 来检索它。

I32 StartStop: 1：开始自动采样，0：停止自动采样。

返回值:

I32 Error code: 请参考 APS 函数返回代码。

示例：

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 1 ); //设置采样率
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //设置
channel_0 的采样源
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //设置
channel_1 的采样源

//开始自动采样
STR_SAMP_DATA_4CH DataArr[500]; //与长度相同
I32 ret;
I32 length = 500; //用户指定获取数据的长度
I32 retLength = 0; //返回数据的物理长度
I32 status; //监控缓冲区状态
I32 start = 1;

APS_auto_sampling( Board_ID, start ); //自动采样开始处理

timer( 10ms )
{
    if( start == 1 )
    {
        Length = 500; //用户指定获取数据的长度
        APS_get_sampling_data(Board_ID, &length, DataArr, & status ); //返回物理长度
        //监控缓冲区状态
        if(status == 1 ) //缓冲区处于 “WORK” 状态
        {
            //获取数据 – 返回的长度取决于缓冲区中的剩余数据
        }
        else if(status == 2 ) //缓冲区已满
        {
            //获取数据
            //采样数据可能会丢失
        }
        else if(status == 3) //缓冲区为空
        {
            //不获取数据 – 返回的长度为 0
        }
    }

    For( i=0; i<length; i++ )
```

```
{  
    //DataArr 准备使用。  
}  
}  
}  
  
APS_auto_sampling( Board_ID, 0 ); //停止自动采样
```

还可以看看:

APS_get_sampling_data(); APS_get_sampling_count()

APS_get_sampling_data	以自动采样模式获取采样数据
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于在开始自动采样后对数据进行采样。 它还用于监控采样状态。有关详细信息，请参考 [Aps_auto_sampling\(\)](#)。以下定义了四个状态：

状态	描述
STOP (0)	自动采样停止
WORK (1)	自动采样开始
FULL (2)	内部缓冲区已满。小心丢失采样数据。由于缓冲区中已充满数据，因此会自动抛出较新的采样数据，直到用户使用了缓冲区中的部分数据为止。
EMPTY (3)	内部缓冲区为空。返回的长度必须为零，并且不获取任何数据

警告：这些是一组用于自动采样的 API 函数，包括 [APS_auto_sampling\(\)](#) 和 [APS_get_sampling_data\(\)](#)。不要与其他触发功能，例如 [APS_wait_trigger_sampling\(\)](#), [APS_wait_trigger_sampling_async\(\)](#) 和 [APS_stop_wait_sampling\(\)](#) 混合使用。

句法:

C/C++:

```
I32 FNTYPE APS_get_sampling_data( I32 Board_ID, I32 *Length, STR_SAMP_DATA_4CH
*DataArr, I32 *Status );
```

Visual Basic:

```
APS_get_sampling_data(ByVal Board_ID As Long, Length As Long, DataArr As
STR_SAMP_DATA_4CH, Status As Long )As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 *Length: 双向。用户需要指定最大大小来获取数据，通常与 “DataArr” 的数组大小相同。返回的长度是取回采样数据的物理大小。

STR_SAMP_DATA_4CH *DataArr: 获取采样数据的结构数组。数组大小必须等于或大于参数 “ * Length”。

I32 *Status: 缓冲区状态。 0 : STOP 状态； 1 : WORK 状态； 2 : EMPTY 状态； 3 : FULL 状态。

返回值:

I32 Error code: 请参考 APS 函数返回代码。

示例：

//... 初始化板卡.

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 1 ); //设置采样率
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //设置
channel_0 的采样源
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //设置
channel_1 的采样源

//开始自动采样
STR_SAMP_DATA_4CH DataArr[500]; //与长度相同
I32 ret;
I32 length = 500; //用户指定获取数据的长度
I32 retLength = 0; //返回数据的物理长度
I32 status; //监控缓冲区状态
I32 start = 1;

APS_auto_sampling( Board_ID, start ); //自动采样开始处理
Timer( 10ms )
{
    If( start == 1 )
    {
        Length = 500; //用户指定获取数据的长度
        APS_get_sampling_data(Board_ID, &length, DataArr, & status ); //返回物理长度
        //监控缓冲区状态
        If(status == 1 ) //缓冲区处于 "WORK" 状态
        {
            //获取数据—返回的长度取决于缓冲区中的剩余数据
        }
        Else if(status == 2 ) //缓冲区已满
        {
            //获取数据
            //some sampling data may be lost
        }
        Else if(status == 3) //缓冲区为空
        {
            //不获取数据 – 返回的长度为 0
        }
    }

    For( i=0; i<length; i++ )
```

```
{  
    //DataArr 准备使用。  
}  
}  
}  
APS_auto_sampling( Board_ID, 0 ); //停止自动采样
```

还可以看看:

APS_auto_sampling(); APS_get_sampling_count()

APS_set_sampling_param_ex	设置采样参数。通过一个扩展至 8 个通道
---------------------------	----------------------

支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于立即设置采样参数，例如采样率，采样通道源等。8 个通道的相关参数由 SAMP_PARAM 构成。SAMP_PARAM 结构中有一些通用设置，包括采样率，采样边沿，采样级别和采样通道。每个通道还有其他设置，包括 SAMP_PARAM 结构中的采样源和轴。

采样功能仅适用于内部装有 DSP 电路板的情况。为了满足实时的需求。采样函数可以确保在硬件实时环境下记录每个采样点。

句法:

C/C++:

```
I32 FNTYPE APS_set_sampling_param_ex( I32 Board_ID, SAMP_PARAM *Param );
```

Visual Basic:

```
APS_set_sampling_param_ex (ByVal Board_ID As Long, ByRef Param As SAMP_PARAM)
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

SAMP_PARAM *Param: 设置采样参数的结构

typedef struct _SAMP_PARAM

{

 I32 rate; //采样率[1~65535 次循环]

 I32 edge; //触发边缘[0 : 上升沿； 1 : 下降沿]

 I32 level; //触发电平[-214743648 ~ 2147483647]

 I32 trigCh; //触发通道 [0 ~ 7]

 I32 sourceByCh[8][2]; //按通道对源进行采样，名为 sourceByCh [a] [b]，

 // a : 指定一个通道。总通道数为 8 个。

 // b : 0 : 采样源，请参见下表 1：采样轴

 //采样源：F64 数据占用两个通道，I32 数据占用一个通道。

}

SAMP_PARAM, *PSAMP_PARAM;

源	符号定义	数据类型	值范围
0x00	SAMP_SRC_COM_POS	I32	命令位置
0x01	SAMP_SRC_FBK_POS	I32	反馈位置
0x02	SAMP_SRC_CMD_VEL	I32	命令速度
0x03	SAMP_SRC_FBK_VEL	I32	反馈速度
0x04	SAMP_SRC_MIO	I32	运动 IO

0x05	SAMP_SRC_MSTS	I32	运动状态
0x06	SAMP_SRC_MSTS_ACC	I32	运动状态加速
0x07	SAMP_SRC_MSTS_MV	I32	最大速度下的运动状态
0x08	SAMP_SRC_MSTS_DEC	I32	减速下的运动状态
0x09	SAMP_SRC_MSTS_CSTP	I32	运动状态 CSTP
0x0A	SAMP_SRC_MSTS_MDN	I32	运动状态 MDN
0x0B	SAMP_SRC_MIO_INP	I32	运动状态 INP
0x0D	SAMP_SRC_MIO_ORG	I32	运动状态 OGR
0x20	SAMP_SRC_CONTROL_VOL	I32	控制命令电压
0x22	SAMP_SRC_ENCODER_RAW	I32	编码器原始数据
0x23	SAMP_SRC_ERR_POS	I32	错误位置
0x10	SAMP_SRC_COM_POS_F64	F64	命令位置
0x11	SAMP_SRC_FBK_POS_F64	F64	反馈位置
0x12	SAMP_SRC_CMD_VEL_F64	F64	命令速度
0x13	SAMP_SRC_FBK_VEL_F64	F64	反馈速度
0x14	SAMP_SRC_CONTROL_VOL_F64	F64	控制命令电压
0x15	SAMP_SRC_ERR_POS_F64	F64	错误位置

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
SAMP_PARAM Param;  
Param.rate = 1; // 采样率  
Param.edge = 0; //上升沿  
Param.level = 1000; //触发电平  
Param.trigCh = 0; //通道 0  
Param.sourceByCh[0][0] = 1; //将轴 1 设置为通道 0 的采样源  
Param.sourceByCh[0][1] = 0; //将命令位置 ( I32 ) 设置为通道 0 的采样源  
Param.sourceByCh[1][0] = 0; //将轴 0 设置为通道 1 的采样源  
Param.sourceByCh[1][1] = 1; //将反馈位置 ( I32 ) 设置为通道 1 的采样源  
//.....设置其他通道 , 包括通道 0 到通道 7  
  
I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //设置采样参数  
...
```

还可以看看:

APS_get_sampling_param_ex(); APS_wait_trigger_sampling_ex()

APS_get_sampling_param_ex	获取采样参数。通过一个扩展至 8 个通道
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于立即获取采样参数，例如采样率，采样通道源等。请参阅
APS_set_sampling_param_ex().

句法:

C/C++:

```
I32 FNTYPE APS_get_sampling_param_ex( I32 Board_ID, SAMP_PARAM *Param );
```

Visual Basic:

```
APS_get_sampling_param_ex (ByVal Board_ID As Long, ByRef Param As SAMP_PARAM)  
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

SAMP_PARAM *Param: 设置采样参数的结构

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
SAMP_PARAM Param;
```

```
Ret = APS_get_sampling_param_ex( Board_ID, &Param); //获取所有参数
```

```
...
```

还可以看看:

APS_set_sampling_param_ex();APS_wait_trigger_sampling_ex()

APS_wait_trigger_sampling_ex	等待采样数据。通过一个扩展至 8 个通道
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于从控制器采样数据。当该函数发布后，程序将声明对信息进行采样并将数据放入内部缓冲区。在触发信号接通之前，程序从内部缓冲区到用户的数据缓冲区中提取了大小为预触发长度的大量数据，并连续对数据进行采样，直到达到用户指定的长度。另一方面，如果达到超时时间且触发信号未升高，则此函数将超时并返回错误消息。

使用 **APS_stop_wait_sampling** 强制停止等待采样。

注意:

APS_wait_trigger_sampling_ex, APS_wait_trigger_sampling_async_ex and APS_auto_sampling_ex 函数不能同时使用。

句法:

C/C++:

```
I32 FNTYPE APS_wait_trigger_sampling_ex( I32 Board_ID, I32 Length, I32 PreTrgLen,
I32 TimeOutMs, STR_SAMP_DATA_8CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_ex(ByValBoard_ID As Long, ByVal Length As Long, ByVal
PreTrgLen As Long, ByVal TimeOutMs As Long, DataArr As STR_SAMP_DATA_8CH ) As
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 **APS_initial()** 来检索它。

I32 Length: 采样数据的数量。（数组大小）

I32 PreTrgLen: 预触发长度。

I32 TimeOutMs: 超时时间。单位为毫秒。

STR_SAMP_DATA_8CH *DataArr: 获取采样数据结构数组。数组大小必须大于参数“Length”。

返回值:

I32 Error code: 请参考 **APS** 函数返回代码。

示例 :

```
//... 初始化板卡.
```

```
SAMP_PARAM Param;
```

```
Param.rate = 1; // 采样率
```

```
Param.edge = 0; //上升沿
```

```

Param.level = 1000; //触发电平
Param.trigCh = 0; //通道 0
Param.sourceByCh[0][0] = 1; //将轴 1 设置为通道 0 的采样源
Param.sourceByCh[0][1] = 0; //将命令位置 ( I32 ) 设置为通道 0 的采样源
Param.sourceByCh[1][0] = 0; //将轴 0 设置为通道 1 的采样源
Param.sourceByCh[1][1] = 1; //将反馈位置 ( I32 ) 设置为通道 1 的采样源
//.....设置其他通道，包括通道 0 到通道 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //设置采样参数

I32 Length = 1024;//总的采样数据数组大小。
I32 PreTrgLen = 100; //预触发点数
STR_SAMP_DATA_8CH DataArr[1024];
I32 TimeOutMs = 10000; // 10 秒超时

Ret =APS_wait_trigger_sampling_ex( Board_ID, Length, PreTrgLen, TimeOutMs,
&DataArr );
If( Ret == ERR_NoError )
{
//采样成功
//DataArr 准备使用。
}

```

还可以看看:

APS_set_sampling_param_ex(); APS_get_sampling_param_ex();
APS_stop_wait_sampling_ex()

APS_wait_trigger_sampling_async_ex	等待异步采样数据。通过一个扩展至 8 个通道
X	

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于从控制器采样数据。此函数将立即返回。并创建一个后台线程来采样数据。

使用 **APS_get_sampling_count** 函数获取要采样的数据计数。当采样计数达到数据长度时，表示采样完成。如果样本计数= -1，则表示等待失败。

使用 **APS_stop_wait_sampling** 强制停止异步等待采样。采样数将变为-1。

警告：

APS_wait_trigger_sampling_ex, APS_wait_trigger_sampling_async_ex and APS_auto_sampling_ex 函数不能同时使用。

句法:

C/C++:

```
I32 FNTYPE APS_wait_trigger_sampling_async_ex( I32 Board_ID, I32 Length, I32
PreTrgLen, I32 TimeOutMs, STR_SAMP_DATA_8CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_async_ex(ByVal Board_ID As Long, ByVal Length As Long,
ByVal PreTrgLen As Long, ByVal TimeOutMs As Long, DataArr As
STR_SAMP_DATA_8CH )As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Length: 采样数据的数量。(数组大小)

I32 PreTrgLen: 预触发长度。

I32 TimeOutMs: 超时时间。单位为毫秒。

STR_SAMP_DATA_8CH *DataArr: 获取采样数据结构数组。数组大小必须大于参数“Length”。

返回值:

I32 Error code: 请参考 APS 函数返回代码。

示例 :

```
//... 初始化板卡.
```

```
SAMP_PARAM Param;
```

```
Param.rate = 1; // 采样率
```

```
Param.edge = 0; //上升沿
```

```

Param.level = 1000; //触发电平
Param.trigCh = 0; //通道 0
Param.sourceByCh[0][0] = 1; //将轴 1 设置为通道 0 的采样源
Param.sourceByCh[0][1] = 0; //将命令位置 ( I32 ) 设置为通道 0 的采样源
Param.sourceByCh[1][0] = 0; //将轴 0 设置为通道 1 的采样源
Param.sourceByCh[1][1] = 1; //将反馈位置 ( I32 ) 设置为通道 1 的采样源
//.....设置其他通道，包括通道 0 到通道 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //设置采样参数

//开始异步等待采样。
I32 Length = 1024;//总的采样数据数组大小。
I32 PreTrgLen = 100; //预触发点数
STR_SAMP_DATA_8CH DataArr[1024];
I32 TimeOutMs = 10000; // 10 秒超时
I32 Ret;

Ret =APS_wait_trigger_sampling_async_ex( Board_ID, Length, PreTrgLen, TimeOutMs,
DataArr );

if( Ret != ERR_NoError )
{
    //显示错误信息
}else
{
    while( count < Length )
    {
        APS_get_sampling_count( Board_ID, &count );
        If( count == -1 )
        {
            //采样失败 ,
            //中断程序。
        }

        If( ForceStop )
        {
            APS_stop_wait_sampling(Board_ID);
        }
    }

    If( count == Length )

```

```
{ //采样成功  
    //DataArr 准备使用。  
}  
}
```

还可以看看：

APS_get_sampling_count(); APS_wait_trigger_sampling_ex(); APS_stop_wait_sampling

APS_get_sampling_data_ex	在自动采样模式下获取采样数据。通过一个扩展至 8 个通道
--------------------------	------------------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于在开始自动采样后对数据进行采样。它还用于监控采样状态。 有关详细信息，请参考 [APS_auto_sampling\(\)](#)。 以下定义了四个状态：

状态	描述
STOP (0)	自动采样停止
WORK (1)	自动采样开始
EMPTY (2)	内部缓冲区已满。小心丢失采样数据。由于缓冲区中已充满数据，因此会自动抛出较新的采样数据，直到用户使用了缓冲区中的部分数据为止。
FULL (3)	内部缓冲区为空。返回的长度必须为零，并且不获取任何数据

警告：这些是一组用于自动采样的 API 函数，包括 [APS_auto_sampling\(\)](#) 和 [APS_get_sampling_data\(\)](#)。不要与其他触发功能，例如 [APS_wait_trigger_sampling\(\)](#), [APS_wait_trigger_sampling_async\(\)](#) 和 [APS_stop_wait_sampling\(\)](#) 混合使用。

句法:

C/C++:

```
I32 FNTYPE APS_get_sampling_data_ex( I32 Board_ID, I32 *Length,
STR_SAMP_DATA_8CH *DataArr, I32 *Status );
```

Visual Basic:

```
APS_get_sampling_data_ex(ByVal Board_ID As Long, Length As Long, DataArr As
STR_SAMP_DATA_8CH, Status As Long )As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 [APS_initial\(\)](#) 来检索它。

I32 *Length: 双向。用户需要指定最大大小来获取数据，通常与 “DataArr” 的数组大小相同。返回的长度是取回采样数据的物理大小。

STR_SAMP_DATA_8CH *DataArr: 获取采样数据的结构数组。数组大小必须等于或大于参数 “ * Length” 。

I32 *Status: 缓冲区状态。 0 : STOP 状态； 1 : WORK 状态； 2 : EMPTY 状态； 3 : FULL 状态。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```

//... 初始化板卡.
SAMP_PARAM Param;

Param.rate = 1; // 采样率
Param.edge = 0; //上升沿
Param.level = 1000; //触发电平
Param.trigCh = 0; //通道 0
Param.sourceByCh[0][0] = 1; //将轴 1 设置为通道 0 的采样源
Param.sourceByCh[0][1] = 0; //将命令位置 ( I32 ) 设置为通道 0 的采样源
Param.sourceByCh[1][0] = 0; //将轴 0 设置为通道 1 的采样源
Param.sourceByCh[1][1] = 1; //将反馈位置 ( I32 ) 设置为通道 1 的采样源
//.....设置其他通道，包括通道 0 到通道 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //设置采样参数

```

```

//开始自动采样
STR_SAMP_DATA_8CH DataArr[500]; //与长度相同
I32 ret;
I32 length = 500; //用户指定获取数据的长度
I32 retLength = 0; //返回数据的物理长度
I32 status; //监控缓冲区状态
I32 start = 1;

APS_auto_sampling( Board_ID, start ); //自动采样开始处理

Timer( 10ms )
{
    If( start == 1 )
    {
        Length = 500; //用户指定获取数据的长度
        APS_get_sampling_data_ex(Board_ID, &length, DataArr, & status ); //返回物理长度
                                            //监控缓冲区状态
        If(status == 1 ) //缓冲区处于 "WORK" 状态
        {
            //获取数据-返回的长度取决于缓冲区中的剩余数据
        }
        Else if(status == 2 ) //缓冲区已满
        {
            //获取数据
        }
    }
}

```

```
//some sampling data may be lost
}
Else if(status == 3) //缓冲区为空
{
    //不获取数据 - 返回的长度为 0
}

For( i=0; i<length; i++ )
{
    //DataArr 准备使用。
}
}

APS_auto_sampling( Board_ID, 0 ); //停止自动采样
```

还可以看看:

APS_auto_sampling(); APS_get_sampling_count()

13. DIO & AIO

APS_set_field_bus_d_channel_output	按通道设置现场总线数字输出
------------------------------------	---------------

支持的产品:PCIe-833x

描述:

该函数用于按通道设置现场总线数字输出.

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_d_channel_output( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 Ch_No, I32 DO_Value );
```

Visual Basic:

```
APS_set_field_bus_d_channel_output (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Ch_No As Long, ByVal DO_Value As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Ch_No: 数字输出通道的索引。 (从0开始)

I32 DO_Value: 数字输出的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_get_field_bus_d_channel_output	按通道获取现场总线数字输出
------------------------------------	---------------

支持的产品:PCIe-833x

描述:

该函数用于按通道获取现场总线数字输出.

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_d_channel_output( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 Ch_No, I32 *DO_Value );
```

Visual Basic:

```
APS_get_field_bus_d_channel_output(ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Ch_No As Long, ByRef DO_Value As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Ch_No: 数字输出通道的索引。 (从0开始)

I32 DO_Value: 返回数字输出的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_get_field_bus_d_channel_input

按通道获取现场总线数字输入

支持的产品:PCIe-833x

描述:

该函数用于按通道获取现场总线数字输入.

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_d_channel_input( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 Ch_No, I32 *DI_Value );
```

Visual Basic:

```
APS_get_field_bus_d_channel_input (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Ch_No As Long, ByRef DI_Value As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Ch_No: 数字输出通道的索引。 (从0开始)

I32 *DI_Value: 数字输入的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_set_field_bus_d_port_output	按端口设置现场总线数字输出
---------------------------------	---------------

支持的产品:PCIe-833x

描述:

该函数用于按端口设置现场总线数字输出

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_d_port_output( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Port_No, U32 DO_Value );
```

Visual Basic:

```
APS_set_field_bus_d_port_output (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Port_No As Long, ByVal DO_Value As UInteger) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Port_No: 数字量输出端口的索引。 (从0开始)

U32 DO_Value: 设置数字输出的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_get_field_bus_d_port_input

按端口设置现场总线数字输入

支持的产品:PCIe-833x

描述:

该函数用于按端口设置现场总线数字输出.

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_d_port_input( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Port_No, U32 *DI_Value );
```

Visual Basic:

```
APS_get_field_bus_d_port_input(ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Port_No As Long, ByRef DI_Value As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Port_No: 数字量输出端口的索引。 (从0开始)

U32 *DI_Value: 数字输入的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_get_field_bus_d_port_output

按端口设置现场总线数字输出

支持的产品:PCIe-833x

描述:

该函数用于按端口设置现场总线数字输出。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_d_port_output( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Port_No, U32 *DO_Value );
```

Visual Basic:

```
APS_get_field_bus_d_port_output (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Port_No As Long, ByRef DO_Value As UInteger) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 BUS_No: 现场总线索引 (仅支持索引0)

I32 MOD_No: 从站设备的索引。 (从0开始)

I32 Port_No: 数字量输出端口的索引。 (从0开始)

U32 *DO_Value: 返回数字输出的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_write_d_output	设置数字输出值
--------------------	---------

支持的产品:PCI-8253/56, DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于访问板载的通用数字输出。如果通道数超过 32 个，则用户必须分配一个组号才能访问更多 I/O。PCI-8256 有 8 个 (PCI-8253 有 4 个 , DPAC-1000, DPAC-3000 有 4 个, PCI(e)-8154 有 4 个, PCI(e)-8158 有 8 个, PCI-8102 有 2 个,PCI-C154(+) 有 4 个多功能 DO) 输出通道 , 用户可以将组号分配为常数 0。

PCI-8102 具有 16 个 (PCIe-8154/8158 , PCI-C154(+)具有 16 个通道扩展 DO) 输出通道 , 用户可以将组号分配为常数 1。.

PCI-8254/58 / AMP-204/8C 具有 24 个输出通道 , 用户可以将组号分配为常数 0。

数字输出通道按位(bit)定义如下 :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_write_d_output(I32 Board_ID, I32 DO_Group, I32 DO_Data);
```

Visual Basic:

```
APS_write_d_output (ByVal Board_ID As Long, ByVal DO_Group As Long, ByVal
DO_Data as Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group: 数字输出组号。 (仅支持组索引 0)

I32 DO_Data: 数字输出数据 (数据类型为位类型) 。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group: 数字输出组号。 (组索引 0~ 索引 1)

I32 DO_Data: 数字输出数据 (组 0 的数据长度为 8 位 , 组 1 的数据长度为 6 位)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 DO_Group = 0; // 如果数字输出通道小于 32
```

```
I32 DO_Data = 0x000F; // 分配位 0、1、2、3 输出。  
I32 returnCode; //函数返回代码  
  
returnCode = APS_write_d_output( Board_ID, DO_Group, DO_Data );  
if( returnCode != 0 )  
    return MessageBox( “设置数字输出函数失败” );
```

还可以看看:

APS_read_d_input()

APS_read_d_output	读取数字量输出值
-------------------	----------

支持的产品:PCI-8253/56, DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于获取板载的通用数字输出。如果通道数超过 32 个，那么用户必须分配一个组号才能访问更多的 I/O。PCI-8256 有 8 个(PCI-8253 有 4 个，DPAC-1000，DPAC-3000 有 4 个，PCI(e)-8154 有 4 个，PCI(e)-8158 有 8 个，PCI-8102 有 2 个，PCI-C154(+) 有 4 个多功能 DO) 输出通道，用户可以将组号分配为常数 0。

PCI-8102 具有 16 个 (PCIe-8154/8158，PCI-C154(+)具有 16 个通道扩展 DO) 输出通道，用户可以将组号分配为常数 1。

PCI-8254/58 / AMP-204/8C 具有 24 个输出通道，用户可以将组号分配为常数 0。

数字输出通道按位(bit)定义如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_read_d_output(I32 Board_ID, I32 DO_Group, I32 *DO_Data);
```

Visual Basic:

```
APS_read_d_output (ByVal Board_ID As Long, ByVal DO_Group As Long, DO_Data as Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DO_Group: 数字输出组号。(仅支持组索引 0)

I32 *DO_Data: 数字输出数据(数据类型为位类型)。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DO_Group: 数字输出组号。(组索引 0~索引 1)

I32 *DO_Data: 数字输出数据(组 0 的数据长度为 8 位, 组 1 的数据长度为 6 位)。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_write_d_output()

APS_read_d_input	读取数字输入值
------------------	---------

支持的产品:PCI-8253/56, DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-8154/8158, PCI-8102/PCI-C154(+), EMX-100 , PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于获取板载的通用数字输入。如果通道数超过 32 个，那么用户必须分配一个组号才能访问更多的 I/O。PCI-8256 有 8 个(PCI-8253 有 4 个，DPAC-1000，DPAC-3000 有 4 个，PCI(e)-8154 有 4 个，PCI(e)-8158 有 8 个，PCI-8102 有 2 个，PCI-C154(+) 有 4 个多功能 DO) 输入通道，用户可以将组号分配为常数 0。

PCI-8102 具有 16 个 (PCIe-8154/8158，PCI-C154(+)具有 16 个通道扩展 DO) 输入通道，用户可以将组号分配为常数 1。

PCI-8254/58 / AMP-204/8C 具有 24 个输出通道，用户可以将组号分配为常数 0。

数字输出通道按位(bit)定义如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_read_d_input(I32 Board_ID, I32 DI_Group, I32 *DI_Data);
```

Visual Basic:

```
APS_read_d_input (ByVal Board_ID As Long, ByVal DI_Group As Long, DI_Data as Long)
As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DI_Group: 数字输入组号。(仅支持组索引 0)

I32 *DI_Data: 返回的数字输入数据。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DI_Group: 数字输入组号。(组索引 0~索引 3)

I32 *DI_Data: 返回的数字输入数据。(每组数据长度为 8 位)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 DI_Group = 0;          //如果数字输入通道小于 32
I32 DI_Data = 0;           //数字输入数据
I32 returnCode;             //函数返回代码

returnCode = APS_read_d_input( Board_ID, DI_Group, &DI_Data );
if( returnCode != 0 )
    MessageBox( "获取数字输入函数失败" );
```

还可以看看:

APS_write_d_output()

APS_write_d_channel_output	设置每通道的数字输出值
----------------------------	-------------

支持的产品:PCIe-8154/8158, PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

此函数用于访问板载的通用数字输出。PCIe-8154/8158 , PCI-C154(+)具有 16 通道的扩展 DO 输出通道 , 用户可以将组号分配为常数 1。

PCI-8254/58 / AMP-204/8C 在组号 0 中具有 24 个输出通道。

组号为 0 的数字输出通道的定义 :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ch 0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_write_d_channel_output(I32 Board_ID, I32 DO_Group, I32 Ch_No, I32
DO_Data);
```

Visual Basic:

```
APS_write_d_channel_output (ByVal Board_ID As Long, ByVal DO_Group As Long,
ByVal Ch_No As Long, ByVal DO_Data as Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group: 数字输出组编号。

I32 Ch_No: 数字输出通道。

I32 DO_Data: 数字输出数据 (数据类型为位类型) 。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group: 数字输出组号。 (仅支持组索引 0)

I32 Ch_No: 数字输出通道 (范围为 0~13)

I32 DO_Data: 通道 (0~1) 的数字输出数据。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DO_Group: 数字输出组号。在 PCI-8254/8 上设置为 0。

I32 Ch_No: 数字输出通道 (0~23)

I32 DO_Data: 通道 (0~1) 的数字输出数据。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI(e)-8154/58

```
I32 DO_Group = 1;           // 如果数字输出通道小于 32
I32 DO_Data = 1;
I32 returnCode;             // 函数返回代码
I32 Ch_No = 0;              // 分配位 0 输出。
returnCode = APS_write_d_output( Board_ID, DO_Group, Ch_No, DO_Data );
if( returnCode != 0 )
    return MessageBox( "设置数字输出函数失败" );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 returnCode;             // 函数返回代码
// 在组号 0 中打开数字输出通道 3
returnCode = APS_write_d_channel_output( Board_ID, 0, 3, 1 );
if( returnCode != 0 )
    MessageBox( "设置数字通道输出函数失败" );
```

还可以看看:

[APS_read_d_channel_input\(\)](#)

APS_read_d_channel_output	读取每通道的数字输出值
---------------------------	-------------

支持的产品:PCIe-8154/8158, PCI-C154(+), EMX-100, PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于按通道访问板载通用数字输出。如果这些通道大于 32，则用户必须分配一个组号才能访问更多 I/O。

PCIe-8154/8158 , PCI-C154(+)具有 16 通道的扩展 DO 输出通道，用户可以将组号分配为常数 1。

PCI-8254/58 / AMP-204/8C 在组号 0 中具有 24 个输出通道。

组号为 0 的数字输出通道的定义：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ch 0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_read_d_channel_output(I32 Board_ID, I32 DO_Group, I32 Ch_No, I32
*DO_Data);
```

Visual Basic:

```
APS_read_d_channel_output (ByVal Board_ID As Long, ByVal DO_Group As Long, ByVal
Ch_No As Long, DO_Data As Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group : 数字输出组号。 (仅支持组索引 0)

I32 Ch_No : 数字输出通道。

I32 * DO_Data : 数字输出数据 (数据类型为位类型) 。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group : 数字输出组号。 (仅支持组索引 0)

I32 Ch_No : 数字输出通道 (范围为 0~13)

I32 * DO_Data : 数字输出数据 (0,1)

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 DO_Group : 数字输出组号。 在 PCI-8254/8 上设置为 0。

I32 Ch_No : 数字输出通道 (0~23)

I32 * DO_Data : 数字输出数据

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 DO_Data = 0;           // Do 数据
I32 returnCode;            //函数返回代码
//在组号 0 中获取数字输出通道 3 的状态
returnCode = APS_read_d_channel_output( Board_ID, 0, 3, &DO_Data );
if( returnCode != 0 )
    MessageBox( "获取数字通道输出函数失败" );
```

还可以看看:

[APS_write_d_channel_output\(\)](#)

APS_read_d_channel_input	读取每通道的数字输入值
--------------------------	-------------

支持的产品:PCIe-8154/8158, PCI-C154(+),EMX-100 , PCIe-833x

描述:

此函数用于按通道访问板载通用数字输入。PCIe-8154/8158 , PCI-C154 (+) 具有 16 个通道扩展 DI 输入通道 , 用户可以将组号分配为常数 1。

组号为 0 的数字输出通道的定义 :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ch 0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

句法:

C/C++:

```
I32 FNTYPE APS_read_d_channel_input(I32 Board_ID, I32 DI_Group, I32 Ch_No, I32
*DI_Data);
```

Visual Basic:

```
APS_read_d_channel_input (ByVal Board_ID As Long, ByVal DI_Group As Long, ByVal
Ch_No As Long, DI_Data as Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DI_Group: 数字输入组号。 (仅支持组索引 0)

I32 Ch_No: 数字输入通道。

I32 *DI_Data: 返回的数字输入数据。

对于 EMX-100:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 DI_Group : 数字输出组号。 (仅支持组索引 0)

I32 Ch_No : 数字输出通道 (范围为 0~31)

I32 * DI_Data : 数字输出数据 (0,1)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_write_d_channel_output\(\)](#)

APS_read_a_input_value	回读模拟输入值 (伏特)
------------------------	----------------

支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

模拟输入有两种函数。一种是转换后的数据。可以是电压或电流值。 另一个是原始数据。它与硬件设计的位精度有关。该函数用于获取一个轴的通用模拟输入值，模拟输入值单位为伏特。

根据硬件规格和设置，该转换是 APS 库中的一种转换。

注意：AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_read_a_input_value(I32 Board_ID, I32 Channel_No, F64  
*Convert_Data);
```

Visual Basic:

```
APS_read_a_input_value (ByVal Board_ID As Long, ByVal Channel_No As Long,  
Convert_Data as Double) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Channel_No: 通道编码，范围从 0 到 65535。

F64 *Convert_Data: 返回的是已转换的模拟数据。单位为伏特，范围为-10V 至 10V。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 Board_ID = 0;  
I32 Channel_No = 0;  
F64 Convert_Data = 0.0;  
I32 returnCode; //函数返回代码
```

```
returnCode = APS_read_a_input_value( Board_ID, Channel_No, & Convert_Data );  
if( returnCode != 0 )  
    MessageBox( "获取模拟输入函数失败" );
```

还可以看看:

[APS_read_a_input_data\(\)](#)

APS_read_a_input_data	回读模拟输入原始数据
支持的产品:PCI-8253/56	

描述:

模拟输入有两种函数。一种是转换后的数据。可以是电压或电流值。另一个是原始数据。它与硬件设计的位精度有关。该函数用于获取一个轴的通用模拟输入原始值。

句法:

C/C++:

```
I32 FNTYPE APS_read_a_input_data(I32 Board_ID, I32 Channel_No, I32 *Raw_Data);
```

Visual Basic:

```
APS_read_a_input_data (ByVal Board_ID As Long, ByVal Channel_No As Long,  
Raw_Data as Long) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Channel_No: 通道编码，范围从 0 到 65535。

I32 *Raw_Data: 返回的是模拟通道的原始数据。原始数据定义：

*Raw_Data = -32768 => 即 -10V

*Raw_Data = 0 => 即 0V

*Raw_Data = 32767 => 即 10V

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_read_a_input_value\(\)](#)

APS_write_a_output_value	设置模拟输出值 (伏特)
--------------------------	----------------

支持的产品:PCI-8253/56, PCI-8254/58 / AMP-204/8C

描述:

模拟输出有两种函数。一种是转换后的数据。可以是电压或电流值。 另一个是原始数据。它与硬件设计的位精度有关。

该函数用于访问板载的一个轴的通用模拟输出原始值，单位为伏特。在使用模拟输出函数之前，请确保轴的伺服开启信号相对于通道号已关闭。

注意：AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_write_a_output_value(I32 Board_ID, I32 Channel_No, F64  
Convert_Data);
```

Visual Basic:

```
APS_write_a_output_value (ByVal Board_ID As Long, ByVal Channel_No As Long, ByVal  
Convert_Data as Double) As Long;
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Channel_No: 通道编码，范围从 0 到 65535。

F64 Convert_Data: 已转换的、需要输出的模拟数据。单位为伏特，范围为 -10V 至 10V

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI-8253/56

```
I32 Channel_No = 1;      // 将通道 1 分配为输出通道  
F32 Convert_Data;  
I32 returnCode;          // 函数返回代码  
While( 1 )  
{  
    // From -10 ..... +10 step 0.1  
    Convert_Data = -10.0;  
    do  
    {  
        APS_write_a_output_value( Board_ID, Channel_No, Convert_Data );  
        Sleep(10);  
        Convert_Data += 0.1;  
    }  
}
```

```
    } while( Convert_Data < 10.0 )  
}
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C
I32 Channel_No = 1; //将通道 1 分配为输出通道
F32 Convert_Data = 5.2; //输出 5.2 伏
I32 returnCode; //函数返回代码

```
returnCode = APS_write_a_output_value( Board_ID, Channel_No, Convert_Data );  
if( returnCode != 0 )  
    MessageBox( "写入模拟输出函数失败" );
```

还可以看看:

APS_write_a_output_data()

APS_write_a_output_data	将原始数据设置为模拟输出值
支持的产品:PCI-8253/56	

描述:

模拟输出有两种函数。一种是转换后的数据。可以是电压或电流值。另一个是原始数据。它与硬件设计的位精度有关。该函数用于访问板载的一个轴的通用模拟输出原始值。在使用模拟输出函数之前，请确保轴的伺服开启信号相对于通道号已关闭。

句法:

C/C++:

```
I32 FNTYPE APS_write_a_output_data(I32 Board_ID, I32 Channel_No, I32 Raw_Data);
```

Visual Basic:

```
APS_write_a_output_data (ByVal Board_ID As Long, ByVal Channel_No As Long, ByVal  
Raw_Data as Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 Channel_No: 通道编码，范围从 0 到 65535。

I32 Raw_Data: 要输出的原始模拟数据。原始数据定义如下

Raw_Data = -32768 => 即 -10V

Raw_Data = 0 => 即 0V Raw_Data = 32767 => 即 10V

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Channel_No = 2;      // 将通道 1 分配为输出通道
I32 Raw_Data;
I32 returnCode;          // 函数返回代码

While( 1 )
{
    // 从 -10...+10, 每步 1 位 ( bit )
    Raw_Data = -32768;
    do
    {
        APS_write_a_output_Raw_Data( Board_ID, Channel_No, Raw_Data );
        Sleep(10);
        Raw_Data += 1;
    } while(Raw_Data < 0x7FFF)
```

}

还可以看看：

APS_write_a_output_value()

14. 点表运动

APS_set_point_table	设置点表运动参数
---------------------	----------

支持的产品:PCI-8253/56, PCI-8392(H)

描述:

此函数用于给指定轴设置一组点表参数。APS 中定义的点表不仅是一个点表，而且还是一个命令表。用户可以使用此点表连接到一个运动序列。该序列可用于不同的速度参数和曲线参数。可以为下一个运动为其分配结束操作。

一旦参考产品规格后，最高点可以下载到板载的内存中。通过设置重复运动，用户可以进行动态加载，而不受点的质量限制的影响。

句法:

C/C++:

```
I32 FNTYPE APS_set_point_table( I32 Axis_ID, I32 Index, POINT_DATA *Point );
```

Visual Basic:

```
APS_set_point_table( ByVal Axis_ID As Long, ByVal Index As Long, Point As  
POINT_DATA ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Index: 需要设置的指定点索引。范围

POINT_DATA *Point: 点表的结构参数。在 “ type_def.h” typedef 结构中
typedef struct

```
{  
    I32 i32_pos;      // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )  
    I16 i16_accType; // 加速模式 , 0 : T 曲线 , 1 : S 曲线  
    I16 i16_decType; // 减速模式 , 0 : T 曲线 , 1 : S 曲线  
    I32 i32_acc;      // 加速率 ( 脉冲 / 秒 2 )  
    I32 i32_dec;      // 减速率 ( 脉冲 / 秒 2 )  
    I32 i32_initSpeed; // 开始速度 ( 脉冲 / 秒 )  
    I32 i32_maxSpeed; // 最大速度 ( 脉冲 / 秒 )  
  
    I32 i32_endSpeed; // 终止速度 ( 脉冲 / 秒 )  
    I32 i32_angle;     // 圆弧运动角度 ( -360~360 度 )  
    U32 u32_dwell;     // 停留时间 ( 单位 : ms ) * 除以系统循环时间。  
    I32 i32_opt;       // 点动选项。 (*)  
}
```

(*)点动选项: *i32_opt*

7	6	5	4	3	2	1	Bit : 0
---	---	---	---	---	---	---	---------

-	-	最后一个点	完成条件	Table_Ctr I	线性/圆弧	-	绝对/相对
15	14	13	12	11	10	9	Bit : 8
Table_N o	Table_N o	Table_N o	Do_C h	Do_Ch	Do_C h	Do_OnOf f	Do_E n

Bit 0: 1:相对运动 0:绝对运动

Bit 2: 1 : 圆弧运动 , 0 : 线性运动

Bit 3: 1: 1: 启用 VAO 表切换控制 (启用时 , 设置表从 bit13 到 bit 15 有效) , 0 : 禁用

Bit 4: 1:INP ON (进入位置信号) , 0 : CSTP ON (命令停止信号)

Bit 5: 1: 最后的点索引。 0 : 不是最后的点索引。 (如果此位打开 , 则点表运动将在该点之后停止。)

Bit 8: 1 : 启用 DO , 0 : 禁用 DO

Bit 9: 1: 设置 DO 开 (设置为 1) , 0 : 设置 DO 关 (设置为 0)

Bit 10~12: 选择一个 Do 通道 (0~7)

Bit 13~15: 在 0 到 7 之间选择一个表编号。当启用 bit 3 时有效。在该点上运行点表时 , 它将自动切换到指定的 VAO 表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

I32 ret;

POINT_DATA Point;

Point.i32_pos = 10000; // (中心) 位置数据 (可以是相对值或绝对值) (脉冲)

Point.i16_accType = 1; // 加速模式 , 0 : T 曲线 , 1 : S 曲线

...

// 将点数据设置到存储卡中。

Ret = APS_set_point_table(Axis_ID, 0, &Point);

if(ret != ERR_NoError)

{ // 错误 (C)

}

还可以看看:

APS_get_point_table();APS_point_table_move();APS_get_next_point_index();

APS_get_start_point_index();APS_get_end_point_index()

APS_get_point_table	获取点表运动参数
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于获取指定轴的一组点表参数。

句法:

C/C++:

```
I32 FNTYPE APS_get_point_table( I32 Axis_ID, I32 Index, POINT_DATA *Point );
```

Visual Basic:

```
APS_get_point_table( ByVal Axis_ID As Long, ByVal Index As Long, Point As  
POINT_DATA ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Index: 需要设置的指定点索引。范围

POINT_DATA *Point: 点表的结构参数。在 “ type_def.h” typedef 结构中
typedef struct

```
{
    I32 i32_pos;      // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
    I16 i16_accType; // 加速模式 , 0 : T 曲线 , 1 : S 曲线
    I16 i16_decType; // 减速模式 , 0 : T 曲线 , 1 : S 曲线
    I32 i32_acc;     // 加速率 ( 脉冲 / 秒 2 )
    I32 i32_dec;     // 减速率 ( 脉冲 / 秒 2 )
    I32 i32_initSpeed; // 开始速度 ( 脉冲 / 秒 )
    I32 i32_maxSpeed; // 最大速度 ( 脉冲 / 秒 )
    I32 i32_endSpeed; // 终止速度 ( 脉冲 / 秒 )
    I32 i32_angle;   // 圆弧运动角度 ( -360~360 度 )
    U32 u32_dwell;   // 停留时间 ( 单位 : ms ) *除以系统循环时间。
    I32 i32_opt;     // 点动选项。 (*)
}
```

} POINT_DATA;

(*)点动选项: *i32_opt*

7	6	5	4	3	2	1	Bit : 0
-	-	最后 一个 点	完成条件	-	线性 / 圆弧	-	绝对 / 相对
15	14	13	12	11	10	9	Bit : 8
			Do_Ch	Do_Ch	Do_Ch	Do_OnOff	Do_En

Bit 0: 1: 相对运动 0: 绝对运动

Bit 2: 1 : 圆弧运动 , 0 : 线性运动
Bit 4: 1:INP ON (进入位置信号) , 0 : CSTP ON (命令停止信号)
Bit 5: 1: 最后的点索引。 0 : 不是最后的点索引。 (如果此位打开 , 则点表运动将在该点之后停止。)
Bit 8: 1 : 启用 DO , 0 : 禁用 DO
Bit 9: 1: 设置 DO 开 (设置为 1) , 0 : 设置 DO 关 (设置为 0)
Bit 10~12: Do 通道 (0~7)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"  
  
//... 初始化板卡.  
I32 ret ;  
POINT_DATA Point;  
ret =APS_get_point_table( Axis_ID, 0, &Point );  
if( ret != ERR_NoError )  
{  
    //错误。  
}
```

还可以看看:

[APS_set_point_table\(\)](#); [APS_point_table_move\(\)](#); [APS_get_next_point_index\(\)](#);
[APS_get_start_point_index\(\)](#); [APS_get_end_point_index\(\)](#)

APS_set_point_table_ex	设置点表运动参数的扩展选项
------------------------	---------------

支持的产品:PCI-8392(H)

描述:

此函数用于给指定轴设置一组带 extend 选项的点表运动参数。APS 中定义的点表不仅是一个点表，而且还是一个命令表。用户可以使用此点表连接到一个运动序列。该序列可用于不同的速度参数和曲线参数。可以为下一个运动为其分配结束操作。

一旦参考产品规格后，最高点可以下载到板载的内存中。通过设置重复运动，用户可以进行动态加载，而不受点的质量限制的影响。

如 APS_set_point_table 中所示，支持线性和圆弧运动。带有扩展选项的 APS_set_point_table_ex 额外支持螺旋运动。设置扩展选项可支持多维的运动，允许用户在一系列点动中更改运动尺寸。

句法:

C/C++:

I32 FNTYPE APS_set_point_table_ex(I32 Axis_ID, I32 Index, POINT_DATA_EX *Point);

Visual Basic:

APS_set_point_table_ex (ByVal Axis_ID As Integer, ByVal Index As Integer, ByRef Point As POINT_DATA_EX) As Integer

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Index: 要设置的指定点索引。

POINT_DATA_EX *Point: 点表的结构参数。在 “ type_def.h” typedef 结构中

```
typedef struct
{
    I32 i32_pos;      // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
    I16 i16_accType; // 加速模式 , 0 : T 曲线 , 1 : S 曲线
    I16 i16_decType; // 减速模式 , 0 : T 曲线 , 1 : S 曲线
    I32 i32_acc;     // 加速率 ( 脉冲 / 秒2 )
    I32 i32_dec;     // 减速率 ( 脉冲 / 秒2 )
    I32 i32_initSpeed; // 开始速度 ( 脉冲 / 秒 )
    I32 i32_maxSpeed; // 最大速度 ( 脉冲 / 秒 )
    I32 i32_endSpeed; // 终止速度 ( 脉冲 / 秒 )
    I32 i32_angle;   // 圆弧运动角度 ( -360~360 度 )
    U32 u32_dwell;   // 停留时间 ( 单位 : ms ) * 除以系统循环时间。
    I32 i32_opt;     // 点动选项。 ( * )
    I32 i32_pitch;   // 螺旋运动的俯仰角
    I32 i32_totalheight; // 总高度
    I16 i16_cw;      // 顺时针或逆时针
```

```
I16 i16_opt_ext;      // 选项扩展 ( ** )
} POINT_DATA_EX;
```

(*)点动选项: *i32_opt*

7	6	5	4	3	2	1	Bit : 0
-	-	最后一个点	完成条件	-	线性/圆弧	-	绝对/相对
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	-

Bit 0: 1:相对运动 0:绝对运动

Bit 2: 1 : 圆弧运动 , 0 : 线性运动

Bit 3: 1: 1: 启用 VAO 表切换控制 (启用时, 设置表从 bit13 到 bit 15 有效) , 0 : 禁用

Bit 4: 1:INP ON (进入位置信号) , 0 : CSTP ON (命令停止信号)

Bit 5: 1: 最后的点索引。 0 : 不是最后的点索引。 (如果此位打开, 则点表运动将在该点之后停止。)

Bit 8~15: 已预留。

(**) 点动选项: *i16_opt_ext*

7	6	5	4	3	2	1	Bit : 0
u	z	y	x	-	-	-	螺旋
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	-

Bit 0: 1 : 螺旋运动 , 0 : 线性或圆弧运动

如果 Bit 0 为 1 , 则运动类型为螺旋运动。

如果 Bit 0 为 0 , 则运动类型由 *i32_opt* 的 Bit 2 定义。

Bit 4: 1 : 第一轴运动 , 0 : 第一轴不运动

Bit 5: 1 : 第二轴运动 , 0 : 第二轴不运动

Bit 6: 1 : 第三轴运动 , 0 : 第三轴不运动

Bit 7: 1 : 第四轴运动 , 0 : 第四轴不运动

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

I32 ret;

```
POINT_DATA_EX Point;

Point.i32_pos = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i16_accType = 1; // 加速模式 , 0 : T 曲线 , 1 : S 曲线
...
// 将点数据设置到存储卡中。
Ret = APS_set_point_table_ex(Axis_ID, 0, &Point );
if( ret != ERR_NoError )
{ // 错误 (C)
}
```

还可以看看:

APS_set_point_table();APS_get_point_table();APS_get_point_table_ex();APS_point_table_move();APS_get_next_point_index();APS_get_start_point_index();APS_get_end_point_index()

APS_get_point_table_ex	获取点表运动参数的扩展选项
支持的产品:PCI-8392(H)	

描述:

此函数用于获取指定轴的一组带扩展选项的点表运动参数。

句法:

C/C++:

```
I32 FNTYPE APS_get_point_table_ex( I32 Axis_ID, I32 Index, POINT_DATA_EX *Point );
```

Visual Basic:

```
APS_get_point_table_ex (ByVal Axis_ID As Integer, ByVal Index As Integer, ByRef Point  
As POINT_DATA_EX) As Integer
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Index: 要设置的指定点索引。

POINT_DATA_EX *Point: 点表的结构参数。在 “ type_def.h” typedef 结构中
typedef struct

```
{
    I32 i32_pos;      // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
    I16 i16_accType; // 加速模式 , 0 : T 曲线 , 1 : S 曲线
    I16 i16_decType; // 减速模式 , 0 : T 曲线 , 1 : S 曲线
    I32 i32_acc;      // 加速度 ( 脉冲 / 秒 2 )
    I32 i32_dec;      // 减速度 ( 脉冲 / 秒 2 )
    I32 i32_initSpeed; // 开始速度 ( 脉冲 / 秒 )
    I32 i32_maxSpeed; // 最大速度 ( 脉冲 / 秒 )
    I32 i32_endSpeed; // 终止速度 ( 脉冲 / 秒 )
    I32 i32_angle;    // 圆弧运动角度 ( -360~360 度 )
    U32 u32_dwell;    // 停留时间 ( 单位 : ms ) * 除以系统循环时间。
    I32 i32_opt;      // 点动选项。 ( * )
    I32 i32_pitch;    // 螺旋运动的俯仰角
    I32 i32_totalheight; // 总高度
    I16 i16_cw;        // 顺时针或逆时针
    I16 i16_opt_ext;   // 选项扩展 ( ** )
}
```

POINT_DATA_EX;

(*)点动选项: *i32_opt*

7	6	5	4	3	2	1	Bit : 0
-	-	最后一个点	完成条件	-	线性/圆弧	-	绝对/相对
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	-

Bit 0: 1:相对运动 0:绝对运动

Bit 2: 1 : 圆弧运动 , 0 : 线性运动

Bit 3: 1: 1: 启用 VAO 表切换控制 (启用时, 设置表从 bit13 到 bit 15 有效) , 0 : 禁用

Bit 4: 1:INP ON (进入位置信号) , 0 : CSTP ON (命令停止信号)

Bit 5: 1: 最后的点索引。 0 : 不是最后的点索引。 (如果此位打开, 则点表运动将在该点之后停止。)

Bit 8~15: 已预留。

(**) 点动选项: *i16_opt_ext*

7	6	5	4	3	2	1	Bit : 0
u	z	Y	x	-	-	-	螺旋
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	-

Bit 0: 1 : 螺旋运动 , 0 : 线性或圆弧运动

如果 Bit 0 为 1 , 则运动类型为螺旋运动。

如果 Bit 0 为 0 , 则运动类型由 i32_opt 的 Bit 2 定义。

Bit 4: 1 : 第一轴运动 , 0 : 第一轴不运动

Bit 5: 1 : 第二轴运动 , 0 : 第二轴不运动

Bit 6: 1 : 第三轴运动 , 0 : 第三轴不运动

Bit 7: 1 : 第四轴运动 , 0 : 第四轴不运动

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

I32 ret;

POINT_DATA_EX Point;

```
ret =APS_get_point_table_ex( Axis_ID, 0, &Point );
if( ret != ERR_NoError )
{
    //错误。
}
```

还可以看看:

APS_set_point_table();APS_get_point_table();APS_set_point_table_ex();APS_point_table_move();APS_get_next_point_index();APS_get_start_point_index();APS_get_end_point_index()

APS_point_table_move	开始一个点表运动
----------------------	----------

支持的产品:PCI-8253/56, PCI-8392(H)

描述:

此函数用于开始一个点表运动。当点表运动开始时，系统将从“StartIndex”到“EndIndex”一个接一个地提取点参数。因此，用户必须先指定指向点表的点参数，然后再进行点表运动。

当轴在点表中运动时，用户无法执行其他运动，直到点表运动完成。

用户可以使用 stop_move, emg_stop 函数来强制停止点表运动。

相对运动状态说明		
PMV	点表运动状态	(控制轴) ON: 处于点表运动状态
PDW	点表停留状态	(控制轴) ON: 处于点表停留状态
PPS	点表暂停状态	(控制轴) ON: 处于点表暂停状态
SLV	从轴运动状态	(从轴) ON: 处于从轴运动状态

参考轴: 轴数组中的第一个轴。 用户可以指定它。

控制轴: 最小轴 ID 为控制轴。

从轴: 除控制轴外的其他轴。

示例：

Ex1.

I32 AxisArray[4] = {3, 1, 2, 4};

控制轴是 ID= 1.

参考轴是 ID = 3.

从轴是 ID = 2, 3, 4

Ex2.

I32 AxisArray[3] = { 1, 2, 4};

控制轴是 ID= 1.

参考轴是 ID = 1.

从轴是 ID = 2, 4

句法:

C/C++:

```
I32 FNTYPE APS_point_table_move( I32 Dimension, I32 *Axis_ID_Array, I32 StartIndex,  
I32 EndIndex );
```

Visual Basic:

```
APS_point_table_move( ByVal Dimension As Long, Axis_ID_Array As Long, ByVal  
StartIndex As Long, ByVal EndIndex As Long) As Long
```

参数:

I32 Dimension: 轴数组的尺寸。 (线性运动 :1~4) , (圆弧运动 :2)

I32 *Axis_ID_Array: 轴 ID 数组。

I32 StartIndex: 第一个运行点索引。

I32 EndIndex: 点索引的终点。

<Ex>

```
StartIndex = 3, EndIndex = 5.
```

运行顺序将是 3 -> 4 -> 5

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
I32 ret;  
POINT_DATA Point;  
I32 Axis_ID_Array;  
  
Point.i32_pos = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )  
Point.i16_accType = 1; // 加速模式 , 0 : T 曲线 , 1 : S 曲线  
...  
// 将点数据设置到存储卡中。  
Ret = APS_set_point_table(Axis_ID, 0, &Point );  
...  
if( ret != ERR_NoError )  
{ // 错误 (C)  
}
```

```
// 开始一个点表运动。  
Axis_ID_Array = Axis_ID;  
ret = APS_point_table_move( 1, &Axis_ID_Array, 0 , 3 );  
...  
...
```

还可以看看：

APS_set_point_table();APS_get_point_table();APS_point_table_move();APS_get_next_point_index();APS_get_start_point_index();APS_get_end_point_index()

APS_get_running_point_index	当轴执行一个点运动时，获取当前点运动索引
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于在轴执行点表运动时，获取运行的点索引。例如，如果系统正在运行索引 3，则此函数将返回索引= 3。

如果操作在最后一个点上运行，则此函数将返回“终点索引”。

注意：当系统处于开始状态时，默认值为-1。

句法:

C/C++:

```
I32 FNTYPE APS_get_running_point_index( I32 Axis_ID, I32 *Index );
```

Visual Basic:

```
APS_get_running_point_index( ByVal Axis_ID As Long, Index As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Index: 返回运行点索引。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

```
I32 Index;
I32 ret = APS_get_running_point_index ( Axis_ID, &Index );
If( ret != ERR_NoError )
{ //错误 (C)
}
```

还可以看看:

[APS_set_point_table\(\)](#); [APS_get_point_table\(\)](#); [APS_point_table_move\(\)](#); [APS_get_start_point_index\(\)](#); [APS_get_end_point_index\(\)](#)

APS_get_start_point_index	当轴执行一个点表运动时，获取第一个点索引
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于当轴执行一个点表运动时，获取第一个点索引

句法:

C/C++:

```
I32 FNTYPE APS_get_start_point_index( I32 Axis_ID, I32 *Index );
```

Visual Basic:

```
APS_get_start_point_index( ByVal Axis_ID As Long, Index As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Index: 返回第一个运行点索引。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

```
I32 Index;
I32 ret = APS_get_start_point_index ( Axis_ID, &Index );
If( ret != ERR_NoError )
{ //错误 (C)
}
```

还可以看看：

APS_set_point_table();APS_get_point_table();APS_point_table_move();APS_get_next_point_index();APS_get_end_point_index()

APS_get_end_point_index	当轴执行一个点表运动时，获取点索引的结尾。
-------------------------	-----------------------

支持的产品:PCI-8253/56, PCI-8392(H)

描述:

该函数用于当轴执行一个点表运动时，获取点索引的结尾。

句法:

C/C++:

I32 FNTYPE APS_get_end_point_index(I32 Axis_ID, I32 *Index);

Visual Basic:

APS_get_end_point_index(ByVal Axis_ID As Long, Index As Long) As Long

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Index: 返回运行点索引的结尾。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

```
I32 Index;
I32 ret = APS_get_end_point_index( Axis_ID, &Index );
If( ret != ERR_NoError )
{ //错误 (C)
}
```

还可以看看:

APS_set_point_table();APS_get_point_table();APS_point_table_move();APS_get_next_point_index();APS_get_start_point_index()

APS_set_table_move_pause	暂停点表运动
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于暂停点位表的运动。发出暂停命令时，它不会停在当前点，而是停在下一点索引的起始位置。

句法:

C/C++:

```
I32 FNTYPE APS_set_table_move_pause( I32 Axis_ID, I32 Pause_en );
```

Visual Basic:

```
APS_set_table_move_pause(ByVal Axis_ID As Long, ByVal Pause_en As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Pause_en:

1 : 暂停。 0 : 不暂停。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

I32 Index;

I32 ret = APS_set_table_move_pause (Axis_ID, 1); //暂停点表运动

If(ret != ERR_NoError)

{ //错误 (C)

}

还可以看看:

APS_set_table_move_ex_pause	减速以停止运动并控制 I/O。
支持的产品:PCI-8253/56	

描述:

该函数用于在运行点表时暂停运动。暂停命令发出后，它将减速以停止并控制 I/O。其他参数包括减速率和 I/O 设置，可以通过 APS_set_axis_para()进行配置。

APS_set_table_move_ex_pause() 和 APS_set_table_move_pause()的区别:

函数 描述	APS_set_table_move_ex_pause()	APS_set_table_move_pause()
运动状态	NSTP(CSTP , INP)	PPS
描述	减速以停止和控制 I/O	停在下一个点索引的起始位置。
回滚	APS_set_table_move_ex_rollback()	N/A
恢复	APS_set_table_move_ex_resume()	APS_set_table_move_pause()

在点表暂停或正常停止时可以控制 I/O，例如禁用激光。打开/关闭指定的 I/O 可通过 APS_set_axis_para()在轴参数表中进行配置。

轴参数表中的 I/O 设置:

编号	定义	描述	值	默认值
32h(50)	PRA_PT_STP_DO_EN	当点表停止/暂停时启用“执行”	0: 禁止 1: 启动	0
33h(51)	PRA_PT_STP_DO	当点表正常停止/暂停时设置“执行”的值	0: 设置为 0 1: 设置为 1	0

句法:

C/C++:

```
I32 FNTYPE APS_set_table_move_ex_pause( I32 Axis_ID );
```

Visual Basic:

```
APS_set_table_move_ex_pause(ByVal Axis_ID As Long, ByVal Pause_en As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
//... 初始化板卡.  
// 预配置参数  
// 点表停止/暂停时启用“执行”  
I32 ret = APS_set_axis_para( Axis_ID, 0x32, 1 );  
// 当点表正常停止/暂停时，将执行值设置为 1（例如打开激光）  
I32 ret = APS_set_axis_para( Axis_ID, 0x33, 1 );  
  
//...运动点表  
I32 ret = APS_set_table_move_ex_pause( Axis_ID );  
    //停止点表运动和控制 I/O。  
If( ret != ERR_NoError )  
{ //错误 (C)  
}
```

还可以看看：

```
APS_set_table_move_ex_rollback();APS_set_table_move_ex_resume();APS_set_axis_para()  
)
```

APS_set_table_move_ex_rollback	回滚到当前点索引的起始位置
--------------------------------	---------------

描述:

当点表暂停时，此功能用于执行回滚运动。此函数用于回滚到当前索引的起始位置。其他参数，包括起始速度、加速速率和减速速率，可以通过 APS_set_axis_para() 进行配置。
请注意，在调用 APS_set_table_move_ex_pause() 之后将使用此函数。否则，可能会运动到意外位置。

句法:

C/C++:

```
I32 FNTYPE APS_set_table_move_ex_rollback( I32 Axis_ID, I32 Max_Speed );
```

Visual Basic:

```
APS_set_table_move_ex_rollback( ByVal Axis_ID As Long, ByVal Max_Speed As Long )  
As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Max_Speed: 最大线性/圆形插补速度。单位：脉冲/秒。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//... 运动点表，并将其暂停

```
I32 ret = APS_set_table_move_ex_rollback ( Axis_ID, Max_Speed );  
        // 回滚运动  
If( ret != ERR_NoError )  
{ //错误 (C)  
}
```

还可以看看:

[APS_set_table_move_ex_pause\(\);APS_set_table_move_ex_resume\(\)](#)

APS_set_table_move_ex_resume	重新开始点表运动并保持 I/O 状态
------------------------------	--------------------

支持的产品:PCI-8253/56

描述:

当点表暂停时，此函数用于恢复从当前索引到结束索引的运动。发出“恢复”命令后，它将重新开始点表运动。通过暂停位置时，它将保持 I/O 状态。

请注意，将在调用 APS_set_table_move_ex_Rollback()之后使用此函数。否则，可能会运动到意外位置。

APS_set_table_move_ex_Resume() 和 APS_set_table_move_pause()的区别:

函数 描述	APS_set_table_move_ex_resume() ()	APS_set_table_move_pause() (when resuming move)
运动状态	PMV, SLV	PMV, SLV
描述	恢复从当前索引运动到结束索引。	恢复从下一个索引运动到结束索引。
暂停	APS_set_table_move_ex_pause()	APS_set_table_move_pause() (when pausing move)

句法:

C/C++:

```
I32 FNTYPE APS_set_table_move_ex_resume( I32 Axis_ID );
```

Visual Basic:

```
APS_set_table_move_ex_resume(ByVal Axis_ID As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorDef.h"  
  
//... 初始化板卡.  
//... 运动点表 ,  
//...暂停 , 然后回滚.
```

```
I32 ret = APS_set_table_move_ex_resume( Axis_ID );
    // 重新开始点表运动并保持 I/O 状态。
If( ret != ERR_NoError )
{ //错误 (C)
}
```

还可以看看:

APS_set_table_move_ex_pause();APS_set_table_move_ex_rollback()

APS_set_table_move_repeat	设置点表运动重复
---------------------------	----------

描述:

该函数用于设定点表的重复运动。当重复函数启用后，它将重复点动，直到禁用重复函数或发出停止功能为止。

句法:

C/C++:

```
I32 FNTYPE APS_set_table_move_repeat ( I32 Axis_ID, I32 Repeat_en );
```

Visual Basic:

```
APS_set_table_move_repeat (ByVal Axis_ID As Long, ByVal Repeat_en As Long ) As  
Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Repeat_en:

1 : 重复。 0 : 不重复。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

```
I32 Index;
```

```
I32 ret = APS_set_table_move_repeat ( Axis_ID, 1 ); // 重复点表运动
```

```
If( ret != ERR_NoError )
```

```
{ //错误 (C)
```

```
}
```

还可以看看:

APS_set_point_table_mode2	设置点表模式
---------------------------	--------

支持的产品: MNET-4XMO-C

描述:

该函数用于选择点表的模式。点表有两种模式：单（快速索引运动）模式和连续（路径运动）模式。相同的时间内在指定的从站模块上只能选择一种模式。在使用其他点表函数之前，用户应调用此函数来选择模式。

单步模式-快速索引运动（模式= 0）：

它提供了一种快速开始运动的方法。由于MNET使用通讯的方式来发送/接收命令和数据，因此访问时间取决于网络的速度和数据量的多少。它提供了一种快速让用户在SRAM上预设已知数据的方法。仅通过点索引命令可以节省大量的通信时间。

连续模式-路径运动（模式= 1）：

它不仅可以使路径轨迹在不受主机控制的情况下连续运行，而且还可以通过我们软件的自动计算来确保路径速度的连续性。

用户只需要提供最大速度和目标位置数据即可，而不必担心启动速度会影响内部命令速度的连续性。这就是所谓的自动速度配置文件的功能。

停顿运动是路径运动的一部分。停顿运动意味着轴仍会保持一定时间。

这些分段的点数据只有一个限制：每个段的距离必须足够长，以符合从当前速度加速或减速到目标最大速度所需的时间。否则它将返回ERR_DistantEnough。

句法:

C/C++:

```
I32 FNTYPE APS_set_point_table_mode2 ( I32 Axis_ID, I32 Mode );
```

Visual Basic:

```
APS_set_point_table_mode2 (ByVal Axis_ID As Long , ByVal Mode As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

对于 MENT-4XMO-C: 指定从站模块上的 Axis_ID。

I32 Mode: 指定的点表模式。（默认为 0）

0: 单步模式（快速索引运动）

1: 连续模式（路径运动）

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"
```

```
#include "APS168.h"
#include "ErrorCodeDef.h"

I32 ret;
I32 Axis_ID = 1000;

ret = APS_set_point_table_mode2 (Axis_ID, 1); //设置为连续模式
//... 设置其他点表参数
```

还可以看看:

APS_set_point_table2	设定点表 2 运动参数
----------------------	-------------

支持的产品: MNET-4XMO-C

描述:

此函数用于设置一组点表参数。APS 中定义的点表不仅是点表，而且是命令表。用户可以根据该点表执行。表内容可以用于不同的速度参数。

单步模式-快速索引运动（模式= 0）：

当点表在单步模式下运行时，最大点数为 1024。它支持轴 1 的绝对运动和相对运动。注意，它仅支持线性和圆弧多点插补运动的相对运动。它还支持停顿运动。在相同的尺寸和轴上，点 0 到点 N 并不是必需的。

连续模式-路径运动（模式= 1）：

当点表在连续模式下运行时，最大点数为 1,048,560。SRAM 缓冲区可以预设 2048 点，用于轴 1 的路径单步运动。如果用户需要插补，轴 2 1024 点或轴 3 682 点或轴 4 512 点都是可能的，包括圆周运动在内。圆周运动仅适用于轴 2 的设置。请注意，在相同的尺寸和轴上，点 0 至点 N 是必需的。

起动速度、加速度和减速度从开始设置就被固定下来，用于整个路径的运动。

注意：当点表在连续模式下运行时，每个点请确保依序从索引 0 设置到 N。

注意：将点被设置为索引 0 时，将导致点表重新初始化，

句法:

C/C++:

```
I32 FNTYPE APS_set_point_table2 ( I32 Dimension, I32 *Axis_ID_Array, I32 Index,  
POINT_DATA2 *Point );
```

Visual Basic:

```
APS_set_point_table2 (ByVal Dimension As Long , Axis_ID_Array As Long, ByVal Index  
As Long, Point As POINT_DATA2 ) As Long
```

参数:

I32 Dimension: 轴数组的尺寸（线性和停顿运动：1~4），（圆弧运动：2）

I32 *Axis_ID_Array: 指定从站模块上的轴 ID 数组

I32 Index: 要设置的指定点索引。

POINT_DATA2 *Point: 点表的结构参数。在“type_def.h” typedef 结构中

```
typedef struct
```

```
{
```

```
    I32 i32_pos[16];      // (中心) 位置数据 (可以是相对值或绝对值) (脉冲)
```

```
    I32 i32_initSpeed;    // 开始速度 (仅适用于单步模式) (脉冲/秒)
```

```
    I32 i32_maxSpeed;    // 最大速度 (脉冲/秒)
```

```
    I32 i32_angle;       // 圆弧运动角度 (-360~360 度)
```

```

    U32 u32_dwell;          //停留时间(单位：毫秒)
    I32 i32_opt;           //点动选项。(*)
} POINT_DATA2;

```

(*)点动选项: *i32_opt*

7	6	5	4	3	2	1	Bit : 0
-	-	最后一个点	完成条件	-	线性/圆弧	-	绝对/相对

Bit 0: 1:相对运动 0:绝对运动

Bit 2: 1 : 圆弧运动 , 0 : 线性运动

Bit 4: 1:INP ON (进入位置信号) , 0 : CSTP ON (命令停止信号)

Bit 5: 1: 最后的点索引。 0 : 不是最后的点索引。 (如果此位打开 , 则点表运动将在该点之后停止。) 仅适用于连续模式。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```

#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

```

```

I32 ret;
I32 Dimension = 2; // 轴 2 插补。
I32 Axis_ID_Array[2] = { 1000, 1001 };
POINT_DATA2 Point;

```

...预设的启动速度 , 加速和减速率

```

Point.i32_pos[0] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲 / 秒 )
Point.i32_opt = 0; // 绝对 , 线性 , CSTP 开启 , 而不是最后一个点索引

```

```

// 将点数据设置到板载 SRAM 上。
Ret = APS_set_point_table2 (Dimension, Axis_ID_Array, 0, &Point ); // 索引 0
//... 按顺序设置索引。
If( ret != ERR_NoError )

```

```
{ //错误 (C)  
}
```

还可以看看:

APS_point_table_continuous_move2	开始一个点表的连续运动
----------------------------------	-------------

支持的产品: MNET-4XMO-C

描述:

使用此函数之前，用户必须使用 APS_set_point_table_mode2() 将点表设置为连续模式。该函数用于启动点表的连续运动。当开始点表运动时，系统将从“0”到“最后的点”——读取点参数。因此，用户必须先指定指向点表的点参数，然后再执行点表运动。

用户可以使用 stop_move , emg_stop 函数来强制停止点表运动。

句法:

C/C++:

```
I32 FNTYPE APS_point_table_continuous_move2( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_point_table_continuous_move2( ByVal Dimension As Long, Axis_ID_Array As  
Long) As Long
```

参数:

I32 Dimension: 轴数组的尺寸（线性和停顿运动：1~4），（圆弧运动：2）

I32 *Axis_ID_Array: 指定从站模块上的轴 ID 数组

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
I32 ret;
```

```
I32 Dimension = 2; // 2 轴插补。
```

```
I32 Axis_ID_Array[2] = { 1000, 1001 };
```

```
I32 Index = 0;
```

```
POINT_DATA2 Point;
```

```
I32 PointTableStatus;
```

...预设的启动速度，加速和减速率

```
Point.i32_pos[0] = 10000; // (中心) 位置数据 (可以是相对值或绝对值) (脉冲)
```

```

Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲 / 秒 )
Point.i32_opt = 0; // 绝对 , 线性 , CSTP 开启 , 而不是最后一个点索引

// 将点数据设置到板载 SRAM 上。
Ret = APS_set_point_table2 ( Dimension, Axis_ID_Array, 0, &Point ); // 索引 0
Index++;
// ... 按顺序预设的点 ( 索引 )
If( ret != ERR_NoError )
{ // 错误 ( C )
}

ret = APS_set_point_table_mode2 ( Axis_ID, 1 ); // 设置为连续模式

// 开始点表的连续运动。
Ret = APS_point_table_continuous_move2 ( Dimension, Axis_ID_Array );
...
// 检查点表状态并按顺序重新加载点 ( 索引 )
ret = APS_point_table_status2( Axis_ID_Array[0], &PointTableStatus );
if( PointTableStatus == 1 ) // SRAM 未满
{
    // 加载点
    ret = APS_set_point_table2 ( Dimension, Axis_ID_Array, 0, &Point ); // 索引 0
    Index++;
}
else
{
    // 不能重新加载点
}

还可以看看:

```

APS_point_table_single_move2	开始一个点表的单步运动
------------------------------	-------------

支持的产品: MNET-4XMO-C

描述:

使用此函数之前，用户必须使用 APS_set_point_table_mode2() 将点表设置为单步模式。该函数用于启动一个点表的单步运动。当开始点表运动时，系统将根据指定的索引执行一次运动。因此，用户必须先指定指向点表的点参数，然后再执行点表运动。

用户可以使用 stop_move，emg_stop 函数来强制停止点表运动。

句法:

C/C++:

```
I32 FNTYPE APS_point_table_single_move2 ( I32 Axis_ID, I32 Index );
```

Visual Basic:

```
APS_point_table_single_move2 ( ByVal Axis_ID As Long, ByVal Index As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

对于 MENT-4XMO-C: 指定从站模块上的 Axis_ID。

I32 Index: 指定要运动的点索引。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
I32 ret;
```

```
I32 Dimension = 2; // 轴 2 插补。
```

```
I32 Axis_ID_Array[2] = { 1000, 1001 };
```

```
POINT_DATA2 Point;
```

...预设置加速度和减速度。

```
Point.i32_pos[0] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_initSpeed = 0; // 开始速度 ( 仅适用于单步模式 ) ( 脉冲 / 秒 )
```

```
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲 / 秒 )
```

```
Point.i32_opt = 1; // 相对，线性，CSTP 开启，不是最后一个点

index ret = APS_set_point_table_mode2 (Axis_ID, 0); //设置为单步模式

//将点数据设置到板载 SRAM 上。
Ret = APS_set_point_table2 (Dimension, Axis_ID_Array, 0, &Point ); //设置索引 0
if( ret != ERR_NoError )
{ //错误 (C)
}

// 开始一个点表单步运动。
Ret = APS_point_table_single_move2 (Axis_ID_Array[0], 0 ); //索引 0 运动
...
```

还可以看看:

APS_get_running_point_index2	当轴执行一个点运动时，获取当前点的运动索引
------------------------------	-----------------------

支持的产品: MNET-4XMO-C

描述:

执行点表运动时，该函数用于获取运行点的索引。例如，如果系统正在运行索引 3，则此函数将返回索引= 3。

如果操作在最后一个点上运行，则此函数将返回“最后一个点索引”。

句法:

C/C++:

```
I32 FNTYPE APS_get_running_point_index( I32 Axis_ID, I32 *Index );
```

Visual Basic:

```
APS_get_running_point_index( ByVal Axis_ID As Long, Index As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

对于 MENT-4XMO-C: 指定从站模块上的 Axis_ID。

I32 *Index: 返回运行点索引。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

//... 初始化板卡.
//...启动网络
I32 Index;
I32 ret = APS_get_running_point_index2 ( Axis_ID, &Index );
If( ret != ERR_NoError )
{ //错误 (C)
}

还可以看看:
```

APS_point_table_status2	当轴执行一个点表运动时，获取点表的状态
支持的产品:MNET-4XMO-C	

描述:

MNET-4XMO-C 提供了一个专用的板载 SRAM 来存储点数据，并使连续路径运动成为可能。执行点表连续运动时，此函数用于获取 SRAM 的状态。SRAM 未满时，用户可以重新加载表。

句法:

C/C++:

```
I32 FNTYPE APS_point_table_status2( I32 Axis_ID, I32 *Status );
```

Visual Basic:

```
APS_point_table_status2( ByVal Axis_ID As Long, Status As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

对于 MENT-4XMO-C: 指定从站模块上的 Axis_ID。

I32 *Status: 获取点表的 SRAM 状态。

0: SRAM 已满。

1: SRAM 未满。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

//... 初始化板卡.

//...启动网络

I32 Status;

```
I32 ret = APS_point_table_status2 ( Axis_ID, &Status );
```

```
If( ret != ERR_NoError )
```

```
{ //错误 (C)
```

```
}
```

还可以看看:

APS_set_point_table3	设置表 3 的运动参数
----------------------	-------------

描述:

此函数用于设置一组点表参数。APS 中定义的点表不仅是点表，而且是命令表。 用户可以根据该点表执行。表内容可以用于不同的速度参数。

点表总共可以存储 2000 个点（从 0 到 1999）。用户可以使用我们提供的结构变量 POINT_DATA3 来设置每个点的数据。POINT_DATA3 结构变量包括五个部分：位置、最大速度、终点位置、方向和命令函数。必须注意，在一个运动中，每个点上的轴数和轴数组中的轴数必须相等。运动类型由命令函数决定，并且必须符合用户设置的每个点的轴数。

从开始设置整个路径开始，起始速度、加速和减速率都是固定的。请通过 APS_set_point_table_param3 函数在运动路径之前预先设置这些值。

注意：下面列出了设置点表中的一些注意事项：

1. 起始速度必须小于最大速度。
2. 表中至少有两个点。
3. 当前一点是圆弧运动时，下一点的最大速度必须大于前一点。
4. 终点不能是圆弧运动。
5. 轴在轴数组中必须唯一的。
6. 轴数组中的轴必须在同一模块中。
7. 在一个运动中，每个点的轴数和轴数组中的轴数必须相等。

句法:

C/C++:

```
I32 FNTYPE APS_set_point_table3( I32 Dimension, I32 *Axis_ID_Array, I32 Index,  
POINT_DATA3 *Point );
```

Visual Basic:

```
APS_set_point_table3 (ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Index  
As Long, Point As POINT_DATA2 ) As Long
```

参数:

I32 Dimension: 轴数组的尺寸（线性运动：1~4），（圆弧运动：2）

I32 *Axis_ID_Array: 指定从站模块上的轴 ID 数组

I32 Index: 要设置的指定点索引。

POINT_DATA3 *Point: 点表的结构参数。在“type_def.h” typedef 结构中
typedef struct

```
{  
    I32 i32_pos[4];      // (中心) 位置数据 (可以是相对值或绝对值) (脉冲)
```

```

I32 i32_maxSpeed; //最大速度 ( 脉冲/秒 )
I32 i32_endPos[2] //圆弧运动
I32 i32_dir; //圆弧运动
I32 i32_opt; //点动选项。(*)
} POINT_DATA3;

```

(*)点动选项: *i32_opt*

值	运动类型	值	运动类型	值	运动类型
0	start_tr_move	7	start_sa_line2	14	start_sr_line3
1	start_ta_move-	8	start_tr_arc2	15	start_sa_line3
2	start_sr_move	9	start_ta_arc2	16	start_sa_line3
3	start_sa_move	10	start_sr_arc2	17	start_ta_line4
4	start_tr_line2	11	start_sa_arc2	18	start_sr_line4
5	start_ta_line2	12	start_tr_line3	19	start_sa_line4
6	start_sr_line2	13	start_ta_line3		

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```

#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

```

```

I32 ret;
I32 Dimension = 2; // 2 轴插补。
I32 Axis_ID_Array[2] = { 0, 1};
POINT_DATA3 Point;

```

...预设的启动速度 , 加速和减速率

```

Point.i32_pos[0] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲/秒 )
Point.i32_opt = 0; // 绝对 , 线性 , 不是最后一个点索引

```

还可以看看:

[APS_point_table_move3\(\)](#); [APS_set_point_table_param3\(\)](#)

APS_point_table_move3	开始一个点表运动
-----------------------	----------

描述:

此函数用于开始一个点表运动。当点表运动开始时，系统将从“StartIndex”到“EndIndex”一个接一个地提取点参数。因此，用户必须先指定指向点表的点参数，然后再进行点表运动。

当轴在点表中运动时，用户无法执行其他运动，直到点表运动完成。

用户可以使用 stop_move，emg_stop 函数来强制停止点表运动。

句法:

C/C++:

```
I32 FNTYPE APS_point_table_move3 (I32 Dimension, I32 *Axis_ID_Array, I32 StartIndex,  
I32 EndIndex)
```

Visual Basic:

```
APS_point_table_move3 ( ByVal Dimension As Long, Axis_ID_Array As Long, StartIndex  
As Long, EndIndex As Long) As Long
```

参数:

I32 Dimension: 轴数组的尺寸（线性&停顿运动：1~4），（圆弧运动：2）

I32 *Axis_ID_Array: 指定从站模块上的轴 ID 数组

注意:

1. 每个点的轴数和轴数组中的轴数必须相等。
2. 轴数组中的轴必须在同一模块中。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
I32 ret;  
I32 index;  
I32 Dimension = 2; // 2 轴插补。  
I32 Axis_ID_Array[2] = { 0, 1};  
I32 StartIndex = 0;
```

```
I32 EndIndex = 1;
```

```
POINT_DATA3 Point;
```

...预设的启动速度，加速和减速率。

```
Point.i32_pos[0] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲 / 秒 )
```

```
Point.i32_opt = 4; // start_tr_line2
```

```
Index = 0;
```

```
ret = APS_set_point_table3 (Dimension, Axis_ID_Array, index, &Point ); // 索引 0
```

```
Point.i32_pos[0] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_pos[1] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_maxSpeed = 10000; // 最大速度 ( 脉冲 / 秒 )
```

```
Point.i32_opt = 6; // start_sr_line2
```

```
Index = 1;
```

```
ret = APS_set_point_table3 (Dimension, Axis_ID_Array, index, &Point ); // 索引 1
```

```
ret = APS_point_table_move3( Dimension, Axis_ID_Array, 0, 1 )
```

还可以看看：

```
APS_set_point_table3(); APS_set_point_table_param3()
```

APS_set_point_table_param3	设置点表运动的速度参数
----------------------------	-------------

描述:

该函数用于设置点表运动的速度参数，包括起始速度，加速度，减速度，曲线加速度和曲线减速度。每个参数的编号与 APS_set_axis_param 使用的轴参数相同。用户可以参考轴参数表来设置速度参数。

句法:

C/C++:

```
I32 FNTYPE APS_set_point_table_param3 (I32 FirstAxiid, I32 ParaNum, I32 ParaDat );
```

Visual Basic:

```
APS_set_point_table_param3 ( ByVal FirstAxiid As Long, ParaNum As Long, ParaDat As Long ) As Long;
```

参数:

I32 FirstAxiid: 由 APS_set_point_table3 函数设置的轴数组中的第一个轴。

I32 ParaNum: 轴参数请参考轴表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "type_def.h"  
#include "APS_define.h"  
#include "APS168.h"  
#include "ErrorCodeDef.h"
```

```
I32 ret;  
I32 index;  
I32 Dimension = 2; // 2 轴插补。  
I32 Axis_ID_Array[2] = { 0, 1};  
I32 StartIndex = 0;  
I32 EndIndex = 1;  
POINT_DATA3 Point;
```

...预设的启动速度，加速和减速率

```
Point.i32_pos[0] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_pos[1] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
```

```
Point.i32_maxSpeed = 10000; //最大速度 ( 脉冲/秒 )
Point.i32_opt = 4; // start_tr_line2
Index = 0;
ret = APS_set_point_table3 (Dimension, Axis_ID_Array, index, &Point ); //索引 0

Point.i32_pos[0] = 20000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_pos[1] = 10000; // ( 中心 ) 位置数据 ( 可以是相对值或绝对值 ) ( 脉冲 )
Point.i32_maxSpeed = 10000; //最大速度 ( 脉冲/秒 )
Point.i32_opt = 6; // start_sr_line2
Index = 1;
ret = APS_set_point_table3 (Dimension, Axis_ID_Array, index, &Point ); //索引 1

ret = APS_set_point_table_param3 ( 0, PRA_ACC, 50000 ); //为点表运动设置加速度
ret = APS_set_point_table_param3 ( 0, PRA_DEC, 50000 ); //为点表运动设置减速度
ret = APS_set_point_table_param3 ( 0, PRA_VS, 100 ); //为点表运动设置起始速度
Ret = APS_set_point_table_param3 ( 0, PRA_SACC, 5000 ); //为点表运动设置曲线加速度
ret = APS_set_point_table_param3 ( 0, PRA_SDEC, 50000); //为点表运动设置曲线减速度

ret = APS_point_table_move3( Dimension, Axis_ID_Array, 0, 1 )
```

还可以看看:

APS_set_point_table3(); APS_point_table_move3()

<code>APS_set_feeder_group</code>	将轴设置给馈线(feeder)组
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于将轴设置给馈线组。在使用任何其他馈线函数之前，应将一些轴分配给馈线组。当不再使用馈线时，应通过 `APS_free_feeder_group()` 函数释放该组。

注意:

当前馈线仅支持二维轴 ID 组。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_set_feeder_group(ByVal GroupId As Long, ByVal Dimension As Long,  
Axis_ID_Array As Long) As Long
```

参数:

I32 GroupId: 组 ID。 取值范围 : 0~1。

I32 Dimension: 轴 ID 数组的长度。值范围 : 1~4

I32 *Axis_ID_Array: 轴 ID 数组，从 0 到 65535。数组大小必须与轴尺寸匹配。

Axis_ID_Array [0] 中的轴 ID 为控制轴，该轴必须是数组中的最小 ID 号。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"  
#include "ErrorCodeDef.h"  
  
I32 ret; // 返回代码  
I32 groupId = 0; // 馈线组 ID [0,1]  
I32 runIdx; // 哪个数据索引正在运行  
I32 fedIdx; // 馈线模块中加载了多少数据。  
I32 msts; // 运动状态  
I32 dim = 2; // 组尺寸  
I32 ax[2] = { 0, 1, }; // 轴 ID 数  
PNT_DATA_2D* pPn = NULL; // PNT_DATA_2D 的指针  
ret = APS_set_feeder_group( groupId, dim, ax );
```

```

if( ret != ERR_NoError ){ //异常处理 }

ret = APS_reset_feeder_buffer(groupId);

ret = APS_set_feeder_point_2D(groupId, pPnt, cnt, 1 );
if( ret != ERR_NoError ) { //异常处理 }

// 开始馈线和点表运动
ret = APS_start_feeder_move( groupId );
if( ret != ERR_NoError ) { //异常处理 }

// 检查点表是否结束运动程序
{
    ret = APS_get_feeder_running_index(groupId, &runIdx);
    if( ret != ERR_NoError ) 切断;
    ret = APS_get_feeder_feed_index(groupId, &fedIdx);
    if( ret != ERR_NoError ) 切断;
    msts = APS_motion_status( ax [0] );
    // 检查运动状态
}

}while( runIdx != ( fedIdx -1 ) );

```

```

ret = APS_free_feeder_group(groupId);
if( ret != ERR_NoError ) { //Exception handling }

```

还可以看看:

```

I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D ( I32 GroupId, POINT_DATA_2D* PtArray, I32
Size, I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );

```

APS_get_feeder_group	在一个馈线(feeder)组中返回配置
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取指定馈线的配置。该配置包括组尺寸和组中包含哪些轴 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_get_feeder_group (ByVal GroupId As Long, Dimension As Long, Axis_ID_Array As Long) As Long
```

参数:

I32 GroupId: 组 ID。 取值范围 : 0~1。

I32 *Dimension: 返回组的轴尺寸。 可能的返回值[0~4]。

I32 *Axis_ID_Array: 返回轴 ID , 从 0 到 65535。请给出一个恒定大小为 4 的数组。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 [APS_set_feeder_group\(\) 示例](#)

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_free_feeder_group	释放馈线(feeder)组及其资源
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于从馈线中释放轴及其资源。当您不再使用馈线时，必须使用此函数释放资源，否则它将保留资源，直到进程终止。

句法:

C/C++:

```
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
```

Visual Basic:

```
APS_free_feeder_group( ByVal GroupId As Long) As Long
```

参数:

I32 GroupId: 组 ID。取值范围：0~1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 `APS_set_feeder_group()` 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_reset_feeder_buffer	重置馈线(feeder)的点缓冲区
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于重置馈线的 2D 点表数据缓冲区。注意:

1. 当馈线将数据加载到控制器时，不能使用此函数重置馈线缓冲区。
2. 当发布 *APS_set_feeder_point_ [n] D()* 且 LastFlag 已被设置时，使用此函数重置缓冲区并清除 LastFlag。

句法:

C/C++:

```
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
```

Visual Basic:

```
APS_reset_feeder_buffer ( ByVal GroupId As Long ) As Long
```

参数:

I32 GroupId: 组 ID。 取值范围 : 0~1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 *APS_set_feeder_group()* 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_set_feeder_point_2D	在馈线(Feeder)的缓冲区中添加一个点
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于将 2D 轨迹数据设置到馈线的缓冲区中。当设置了最后一条轨迹数据时，必须设置参数 “LastFlag”。设置 “LastFlag” 后，可以执行函数 “*APS_start_feeder_move()*”。当设置 “LastFlag” 时，轨迹数据在调用 *APS_reset_feeder_buffer()*之前，不能将其设置为缓冲区。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_point_2D( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size,
I32 LastFlag );
```

Visual Basic:

```
APS_set_feeder_point_2D ( ByVal GroupId As Long, PtArray As PNT_DATA_2D, ByVal
Size As Long, ByVal LastFlag As Long ) As Long
```

参数:

I32 GroupId: 组 ID。取值范围 : 0~1。

PNT_DATA_2D* PtArray: 2D 轨迹信息数组。

I32 Size: PNT_DATA_2D 数组的大小。值必须大于 0。(Size > 0)

I32 LastFlag: 最后一个点数据标志。要注意馈线，点数组是馈线的最后一个。

0: 不是最后一个

1: 最后一个

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 *APS_set_feeder_group()* 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

<code>APS_set_feeder_point_2D_ex</code>	在馈线(Feeder)的缓冲区中添加一个点
支持的产品: PCI-8254/58 / AMP-204/8C	

描述:

此函数用于将二维轨迹数据设置到馈线的缓冲区中。当设置了最后一条轨迹数据时，必须设置参数“LastFlag”。设置“LastFlag”后，可以执行功能“*APS_start_feeder_move()*”。设置为“LastFlag”时，在调用*APS_reset_feeder_buffer()*之前，无法将轨迹数据设置到缓冲区中。

警告:

APS_set_feeder_point_2D_ex() 和 *APS_set_feeder_point_2D()* 函数不能同时使用。
F64 类型使用 *APS_set_feeder_point_2D_ex()*, I32 类型使用 *APS_set_feeder_point_2D()*。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_point_2D_ex( I32 GroupId, PNT_DATA_2D_F64 * PtArray,
I32 Size, I32 LastFlag );
```

Visual Basic:

```
APS_set_feeder_point_2D_ex( ByVal GroupId As Long, PtArray As PNT_DATA_2D_F64,
ByVal Size As Long, ByVal LastFlag As Long ) As Long
```

参数:

I32 GroupId: 组 ID。取值范围：0~1。

PNT_DATA_2D_F64* PtArray: 二维轨迹信息数组。

I32 Size: PNT_DATA_2D_F64 数组大小。值必须大于 0。（大小> 0）

I32 LastFlag: 最后一个点数据标志。要注意馈线，点数组是馈线的最后一个。

0: 不是最后一个

1: 最后一个

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 *APS_set_feeder_group()* 示例

还可以看看:

I32 FNTYPE APS_set_feeder_group(I32 GroupId, I32 Dimension, I32 *Axis_ID_Array);

I32 FNTYPE APS_get_feeder_group(I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array);

I32 FNTYPE APS_free_feeder_group(I32 GroupId);

I32 FNTYPE APS_reset_feeder_buffer(I32 GroupId);

```
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_start_feeder_move	开始点表运动和馈点 (feed point)
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

发布此函数时将执行以下项目 :

1. 将点加载到控制器中 (点表)
2. 开始点表运动.

如果未设置函数 APS_set_feeder_point_ [n] D() 的参数 “ LastFlag ” , 则该函数将失败。

句法:

C/C++:

```
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
```

Visual Basic:

```
APS_start_feeder_move ( ByVal GroupId As Long ) As Long
```

参数:

I32 GroupId: 组 ID。取值范围 : 0~1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 [APS_set_feeder_group\(\)](#) 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D ( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_get_feeder_status	获取馈线的状态
-----------------------	---------

支持的产品: PCI-8254/58 / AMP-204/8C

描述:

用户可以监视馈线的状态，包括馈线状态和馈线错误代码。馈线的三种状态如下：

0 : Feeder_Stop : 馈线停止。

1 : Feeder_Run : 馈线正在运行。

2 : Feeder_Pause : 馈线被 APS_set_feeder_ex_pause()暂停。

错误代码是 APS 函数返回代码.

句法:

C/C++:

```
I32 FNTYPE APS_get_feeder_status( I32 GroupId, I32 *State, I32 *ErrCode );
```

Visual Basic:

```
APS_get_feeder_status( ByVal GroupId As Long, State As Long, ErrCode As Long ) As  
Long
```

参数:

I32 *状态 : 馈线的状态

0 : Feeder_Stop : 馈线停止。

1 : Feeder_Run : 馈线正在运行。

2 : Feeder_Pause : 馈线暂停。

I32 * ErrCode : 馈线运行时的错误代码。请参阅 APS 函数返回代码。

返回值:

I32 Error code: 请参考 APS 函数返回代码.

示例 :

```
I32 state = 0;
```

```
I32 errorCode = 0;
```

```
I32 ret = 0;
```

```
//获取馈线(Feeder)状态
```

```
ret = APS_get_feeder_status( 0, &state, &errorCode );
```

还可以看看:

[APS_start_feeder_move\(\)](#)

<code>APS_get_feeder_running_index</code>	获取正在运行的点。
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于观察控制器当前正在处理哪个缓冲区的索引。缓冲区的索引是您要馈送到缓冲区的数组索引。

此函数与 `APS_get_running_point_index()` 类似，但不同之处在于索引的顺序。

`APS_get_running_point_index()` 返回点表索引，该点按点表本身在 287peration287 的内存中排序；`APS_get_feeder_running_index()` 返回缓冲区索引，该索引按主机 RAM 中的馈线缓冲区排序。

句法:

C/C++:

```
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
```

Visual Basic:

```
APS_get_feeder_running_index ( ByVal GroupId As Long, Index As Long ) As Long
```

参数:

I32 GroupId:组 ID。 取值范围：0~1。

I32 *Index: 返回正在运行的点。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 `APS_set_feeder_group()` 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D ( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_get_feeder_feed_index	获取将哪个点设置到点表中。
支持的产品: PCI-8253/56, PCI-8392(H) , PCI-8254/58 / AMP-204/8C	

描述:

此函数将返回馈线中最新的缓冲区索引已加载到控制器中。

句法:

C/C++:

```
I32 FNTYPE APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

Visual Basic:

```
APS_get_feeder_feed_index ( ByVal GroupId As Long, Index As Long ) As Long
```

参数:

I32 GroupId:组 ID。取值范围 : 0~1。

I32 *Index: 返回哪个缓冲区索引被加载到控制器中。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

参考 *APS_set_feeder_group()* 示例

还可以看看:

```
I32 FNTYPE APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 FNTYPE APS_free_feeder_group( I32 GroupId );
I32 FNTYPE APS_reset_feeder_buffer( I32 GroupId );
I32 FNTYPE APS_set_feeder_point_2D( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size,
I32 LastFlag );
I32 FNTYPE APS_start_feeder_move( I32 GroupId );
I32 FNTYPE APS_get_feeder_running_index( I32 GroupId, I32 *Index );
```

APS_set_feeder_ex_pause	运动已暂停 (已停止) 和馈线(feeder)已暂停
支持的产品: PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于在运行点表时暂停运动。发出暂停命令时，它将减速以停止并关闭 I/O，馈线也将同时暂停。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_ex_pause( I32 GroupId );
```

Visual Basic:

```
APS_set_feeder_ex_pause ( ByVal GroupId As Long ) As Long
```

参数:

I32 GroupId: 组 ID。取值范围：0~1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//当用户界面上按暂停按钮时。
I32 ret;
I32 groupId = 0;
ret = APS_set_feeder_ex_pause( groupId );
if( ret != ERR_NoError ) { //异常处理 }
//检查运动状态是否已停止。
...
...
```

还可以看看:

```
I32 FNTYPE APS_set_feeder_ex_pause( I32 GroupId );
I32 FNTYPE APS_set_feeder_ex_rollback( I32 GroupId, I32 Max_Speed );
I32 FNTYPE APS_set_feeder_ex_resume( I32 GroupId );
```

APS_set_feeder_ex_rollback	返回到暂停索引的起始位置
支持的产品: PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

该函数用于让轴组返回到由 APS_set_feeder_ex_pause()暂停的最后一个点位置。

该函数只能在 APS_set_feeder_ex_pause()之后调用。在其他情况下使用此函数时，行为未定义。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_ex_rollback( I32 GroupId, I32 Max_Speed );
```

Visual Basic:

```
APS_set_feeder_ex_rollback( ByVal GroupId As Long, ByVal Max_Speed As Long ) As Long
```

参数:

I32 GroupId:组 ID。 取值范围：0~1。

I32 Max_Speed: 最大线性插补速度。值> 0，单位：脉冲/秒。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//当在用户界面上按下“返回”按钮时。
I32 ret;
I32 groupId = 0;
I32 max_speed = 5000; // 脉冲/秒
ret = APS_set_feeder_ex_rollback( groupId, max_speed );
if( ret != ERR_NoError ) { //异常处理 }
// 检查运动状态是否完成。
...
...
```

还可以看看:

I32 FNTYPE APS_set_feeder_ex_pause(I32 GroupId);

I32 FNTYPE APS_set_feeder_ex_resume(I32 GroupId);

APS_set_feeder_ex_resume	恢复点表运动。
--------------------------	---------

支持的产品: PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于让已暂停的馈线运行索引恢复运动。通过暂停位置时，它将保持 I/O 状态。

该函数只能在 *APS_set_table_move_ex_Rollback()* 之后调用。在其他情况下使用此函数时，行为未定义。

句法:

C/C++:

```
I32 FNTYPE APS_set_feeder_ex_resume ( I32 GroupId );
```

Visual Basic:

```
APS_set_feeder_ex_resume ( ByVal GroupId As Long ) As Long
```

参数:

I32 GroupId: 组 ID。取值范围：0~1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//当在用户界面上按下“恢复”按钮时。
I32 ret;
I32 groupId = 0;
ret = APS_set_feeder_ex_resume ( groupId );
if( ret != ERR_NoError ) { //异常处理 }
// 检查运动状态是否已开始。
...
...
```

还可以看看:

I32 FNTYPE APS_set_feeder_ex_pause(I32 GroupId);

I32 FNTYPE APS_set_feeder_ex_rollback(I32 GroupId, I32 Max_Speed);

15. 高级点表

APS_pt_enable	启用点表
---------------	------

支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于启用点表。用户可以将板卡的相关轴设置到指定的点表中。在指定的板卡上，禁止将重复轴设置给点位表。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_enable( I32 Board_ID, I32 PtId, I32 Dimension, I32 *AxisArr );
```

Visual Basic:

```
APS_pt_enable (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Dimension As  
Long, ByVal AxisArr() As Long) As Long
```

参数:

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 Dimension:

I32 *AxisArr: 指定板卡的轴阵列为 0 到 N。

对于 PCI-8254, 它是 0 到 3.

对于 PCI-8258, 它是 0 到 7

对于 PCIe-833x :

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 Dimension: 轴的尺寸编号。

I32 *AxisArr: 指定板卡的轴数从 0 到 (N-1)。N 是不使用 EtherCAT 手动从站 ID 时，指定板卡拓扑中实际的总轴数。换句话说，使用 EtherCAT 手动从站 ID 时，轴数组将是“轴 ID”数组。例如：

APS_initial 中的模式	从站 ID 和轴编号	AxisArr 分配值
自动板卡 ID (bit 0 = 0) AxisNo 自动模式 (bit 1 = 0) 现场总线从轴无自动模式 (bit2 = 0) 现场总线从站无自动模式 (bit10 = 0) 模式= 0x0000	从站 0 - 轴 0 从站 1 - 轴 1	AxisArr[0] = 0; AxisArr[1] = 1;
手动板卡 ID(Bit 0 = 1). 板卡 ID = 15. AxisNo 固定模式(Bit 1 = 1) 现场总线从轴无自动模式(Bit 2 = 0) 现场总线从站无自动模式(Bit 10 = 0) 模式= 0x0003	从站 0 - 轴 960(15 * 64 + 0) 从站 1 - 轴 961(15 * 64 + 1)	AxisArr[0] = 0; AxisArr[1] = 1;
自动板卡 ID(Bit 0 = 0). AxisNo 自动模式(Bit 1 = 0) 现场总线从轴无自动模式(Bit 2 = 0) 现场总线从站无固定模式(Bit 10 = 1) 模式= 0x0400	从站 1000 - 轴 5555 从站 2000 - 轴 5556 *StartAxisID 设置为 “5555”	AxisArr[0] = 5555; AxisArr[1] = 5556;
自动板卡 ID(Bit 0 = 0). AxisNo 自动模式(Bit 1 = 0) 现场总线从轴无自动模式(Bit 2 = 1) 现场总线从站无固定模式(Bit 10 = 1) 模式 = 0x0404	从站 1000 - 轴 1000 从站 2000 - 轴 2000	AxisArr[0] = 1000; AxisArr[1] = 2000;
手动板卡 ID(Bit 0 = 1). 板卡 ID = 15. AxisNo 自动模式(Bit 1 = 0) 现场总线从轴无自动模式(Bit 2 = 1) 现场总线从站无固定模式(Bit 10 = 1) 模式= 0x0405	从站 1000 - 轴 1000 从站 2000 - 轴 2000	AxisArr[0] = 1000; AxisArr[1] = 2000;

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
I32 Dimension = 2; //二维尺寸
I32 AxisArr[2] = { 0, 1 }; //将轴 0 和轴 1 设置为点表 0
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
ret = APS_pt_enable(Board_ID , PtId, Dimension, & AxisArr );//启用点表 0
```

还可以看看：

APS_pt_disable	禁用点表
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于禁用点表。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_disable ( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_disable (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0

//禁用
ret = APS_pt_disable(Board_ID , PtId ); //禁用点表 0
```

还可以看看:

APS_get_pt_info	获取点表信息
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取点表信息。用户可以获得尺寸和轴数组的信息。如果禁用点表，则尺寸的返回信息定义为 0。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_get_pt_info( I32 Board_ID, I32 PtId, PPTINFO Info );
```

Visual Basic:

```
APS_get_pt_info (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Info As PTINFO)
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTINFO Info: 用于获取点表信息的结构指针

typedef struct

{

 I32 Dimension; //指定点表中的维数

 I32 AxisArr[6]; //点表的轴数组。最多 6 个轴，取决于维度。

} PTINFO, *PPTINFO;

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Board_ID = 0;
```

```
I32 PtId = 0; //点表 0
```

```
PTINFO Info;
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

```
ret = APS_get_pt_info(Board_ID , PtId, &Info ); //Get information of point table 0
```

还可以看看:

APS_pt_set_vs	将速度配置设置到点表中
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于设置点位表的启动速度 (Vs)。当点表移动时，Vs 仅应用于第一个点。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_vs( I32 Board_ID, I32 PtId, F64 Vs );
```

Visual Basic:

```
APS_pt_set_vs (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Vs As Double) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Vs: 点表的启动速度。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Board_ID = 0;
```

```
I32 PtId = 0; //点表 0
```

```
F64 Vs = 100.0;
```

```
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
```

```
....
```

```
//配置 Vs 到点表 0
```

```
ret = APS_pt_set_vs(Board_ID , PtId, Vs );
```

```
.....
```

还可以看看:

APS_pt_get_vs	获取点表中的速度配置
---------------	------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于启动点表的速度 (Vs)。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_get_vs( I32 Board_ID, I32 PtId, F64 *Vs );
```

Visual Basic:

```
APS_pt_get_vs (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Vs As Double) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 *Vs: 点表的启动速度。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Board_ID = 0;
```

```
I32 PtId = 0; //点表 0
```

```
F64 Vs;
```

```
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
```

```
....
```

```
//获取点表 0 的 vs
```

```
ret = APS_pt_get_vs(Board_ID , PtId, &Vs );
```

```
.....
```

还可以看看:

APS_pt_start	启动点表
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于启动点表。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_start( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_start (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

//配置 Vs 到点表 0, 将点推入点表。

//启动点表开始运动

```
ret = APS_pt_start(Board_ID , PtId );
```

还可以看看:

APS_pt_stop	停止点表
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于停止点表。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_stop( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_stop (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
```

//停止点表运动。

```
ret = APS_pt_stop(Board_ID , PtId );
```

还可以看看:

APS_get_pt_status	获取点表的状态
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于获取点表的状态。这些信息包括点表的状态，点缓冲区的状态，点缓冲区的使用和可用空间以及运行计数。 详细信息如下：

- 状态包括点表的开始和停止状态。
- 缓冲区状态包括点缓冲区的 FULL 或 EMPTY 状态。
- 使用空间是消耗缓冲区大小的计数器。
- 可用空间是剩余缓冲区大小的计数器。
- 运行计数表示启用点表后要执行的点数。

点表中总共有 50 个点缓冲区。用户可以将包括线性运动，圆弧运动或螺旋运动在内的指定运动推入缓冲区。然后，用户可以监视点表的状态以了解缓冲区状态和运行状态。如果缓冲区未满，则用户可以将更多的运动推入缓冲区。如果缓冲区已满，请调用 Sleep()一段时间以等待消耗。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_get_pt_status( I32 Board_ID, I32 PtbId, PPTSTS Status );
```

Visual Basic:

```
APS_get_pt_status (ByVal Board_ID As Long, ByVal PtbId As Long, ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtbId: 点表 ID 为 0 到 1。

PPTSTS Status: 点表的状态。

typedef struct

{

```
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作，0 : 停止]
                // b1 : 点缓冲区是否已满？ [1 : 完整，0 : 未完整]
                // b2 : 点缓冲区是否为空？ [1 : 空，0 : 非空]
                // b3 , b4 , b5 : 保留以备将来之用。
```

U16 PntBufFreeSpace; //点缓冲区的可用空间

U16 PntBufUsageSpace; //点缓冲区的使用空间

U32 RunningCnt; //启用点表后要执行多少点

```
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
....
```

还可以看看:

APS_reset_pt_buffer	重置点表的相关缓冲区
---------------------	------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于重置点表的某些缓冲区。点表中有三个缓冲区，包括运动缓冲区，命令缓冲区和配置文件缓冲区。

运动缓冲区最多可排队 50 个运动。

命令缓冲区可以控制运动。

配置文件缓冲区可以更改速度配置文件，包括 Acc , Dec , S-factor , Vm , Ve 等。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_reset_pt_buffer( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_reset_pt_buffer (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0
```

```
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
```

```
//重置点表缓冲区 0
```

```
ret = APS_reset_pt_buffer(Board_ID , PtId );
```

还可以看看:

APS_pt_roll_back	回滚到上一点
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于回滚到上一点。调用 APS_pt_stop()暂停点表后，用户可以将点表回滚到上一个点。然后，通过调用 APS_pt_start()重新启动点表以执行未完成的动作。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_roll_back( I32 Board_ID, I32 PtId, F64 Max_Speed );
```

Visual Basic:

```
APS_pt_roll_back (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Max_Speed As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Max_Speed: 最大浮点速度。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
F64 Max_Speed = 10000.0;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//将点插入点表。
//启动点表。然后，暂停点表。
//回滚到上一点
ret = APS_pt_roll_back( Board_ID, PtId, Max_Speed );
//然后，重新启动点表。
```

还可以看看:

APS_get_pt_error	获取点表的错误代码
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取点表的错误代码。如果点表操作有误，将记录错误代码。重新启用点表时，错误代码将被重置。错误代码指的是“ ErrorCodeDef.h”以获得物理含义。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_get_pt_error( I32 Board_ID, I32 PtId, I32 *ErrCode );
```

Visual Basic:

```
APS_pt_get_error (ByVal Board_ID As Long, ByVal PtId As Long, ByRef ErrCode As  
Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 *ErrCode: 运行点表的错误代码。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0  
I32 ErrCode = 0;  
  
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。  
//获取运行点表的错误代码  
ret = APS_get_pt_error(Board_ID , PtId, &ErrCode );  
.....
```

还可以看看:

APS_pt_dwell	将一个停顿运动推送到点表的点缓冲区。
--------------	--------------------

描述:

此函数用于将停顿运动推入点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_dwell( I32 Board_ID, I32 PtId, PPTDWL Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_dwell (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTDWL,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTDWL Prof: 暂停动作的曲线

```
typedef struct  
{  
    F64      DwTime; //设置停留时间，单位为毫秒。  
} PTDWL, *PPTDWL;
```

PPTSTS Status: 点表的状态。

```
typedef struct  
{  
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作，0 : 停止]  
                // b1 : 点缓冲区是否已满？ [1 : 完整，0 : 未完整]  
                // b2 : 点缓冲区是否为空？ [1 : 空，0 : 非空]  
                // b3 , b4 , b5 : 保留以备将来之用。  
                // b6~ : 始终为 0  
    U16 PntBufFreeSpace; //点缓冲区的可用空间  
    U16 PntBufUsageSpace; //点缓冲区的使用空间  
    U32 RunningCnt; //启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTDWL Prof;
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
if ( !( Status.BitSts & 0x02 ) )//点缓冲区未满
{
    //将运动推入点缓冲区
    Prof. DwTime = 100; //100ms
    ret = APS_pt_dwell( Board_ID, PtId, &Prof, &Status );
}
//启动点表运动
APS_pt_start( Board_ID, PtId, 0 );
```

还可以看看:

APS_pt_line	将一个线性运动推送到点表的点缓冲区。
-------------	--------------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

此函数用于将线性运动推入点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_line( I32 Board_ID, I32 PtId, PPTLINE Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_line (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTLINe,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTLINE Prof: 线性运动曲线

```
typedef struct  
{  
    I32      Dim; //维度  
    F64      Pos[6]; //线性运动的位置数组  
} PTLINe, *PPTLINE;
```

PPTSTS Status: 点表的状态。

```
typedef struct  
{  
    U16 BitSts; // b0 : PT 是否有效 ? [1 : 工作 , 0 : 停止]  
                // b1 : 点缓冲区是否已满 ? [1 : 完整 , 0 : 未完整]  
                // b2 : 点缓冲区是否为空 ? [1 : 空 , 0 : 非空]  
                // b3 , b4 , b5 : 保留以备将来之用。  
                // b6~ : 始终为 0  
    U16 PntBufFreeSpace; //点缓冲区的可用空间  
    U16 PntBufUsageSpace; //点缓冲区的使用空间  
    U32 RunningCnt; //启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTLINE Prof;
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
if ( !( Status.Bitsts & 0x02 ) )//点缓冲区未满
{
    //将运动推入点缓冲区
    Prof.Dim = 2;
    Prof.Pos[0] = 10000;
    Prof.Pos[1] = 10000;
    ret = APS_pt_line( Board_ID, PtId, &Prof, &Status );
}
//启动点表运动
APS_pt_start( Board_ID, PtId, 0 );
```

还可以看看:

APS_pt_arc2_ca	将一个 2D 圆弧运动推送到点表的点缓冲区。
----------------	------------------------

描述:

此函数用于将二维带角度的圆弧运动推入点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_arc2_ca( I32 Board_ID, I32 PtId, PPTA2CA Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_arc2_ca (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTA2CA,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTA2CA Prof: 圆弧运动曲线

```
typedef struct  
{  
    U8      index[2]; // [0~点表的尺寸]点表中的哪个轴索引  
    F64     Center[2]; //中心位置  
    F64     Angle; //角度，单位为弧度  
} PTA2CA, *PPTA2CA;
```

PPTSTS Status: 点表的状态。

```
typedef struct  
{  
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作, 0 : 停止]  
                // b1 : 点缓冲区是否已满？ [1 : 完整, 0 : 未完整]  
                // b2 : 点缓冲区是否为空？ [1 : 空, 0 : 非空]  
                // b3, b4, b5 : 保留以备将来之用。  
                // b6~ : 始终为 0  
    U16 PntBufFreeSpace; //点缓冲区的可用空间  
    U16 PntBufUsageSpace; //点缓冲区的使用空间  
    U32 RunningCnt; //启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTA2CA Prof;
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
if ( !( Status.BitSts & 0x02 ) )//点缓冲区未满
{
    //将运动推入点缓冲区
    Prof.Index[0] = 0; //选择尺寸 0
    Prof.Index[1] = 1; //选择尺寸 1
    Prof.Center[0] = 10000;
    Prof.Center[1] = 10000;
    Prof.Angle = 3.14159265;
    ret = APS_pt_arc2_ca( Board_ID, PtId, &Prof, &Status );
}
//启动点表运动
APS_pt_start( Board_ID, PtId );
```

还可以看看：

APS_pt_arc2_ce	将一个 2D 圆弧运动推送到点表的点缓冲区。
----------------	------------------------

描述:

此函数用于将终点位置的 2D 圆弧运动推入点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。 用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_arc2_ce( I32 Board_ID, I32 PtId, PPTA2CE Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_arc2_ce (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTA2CE,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTA2CE Prof: 圆弧运动曲线

```
typedef struct
{
    U8      索引[2]; // [0~点表的尺寸]点表中的哪个轴索引
    F64     Center[2]; //中心位置
    F64     End[2]; //终点位置
    I16     Dir; //值指定了旋转的方向，如果 Dir 设置为 0 表示正向旋转，则 dir = -1 沿负向旋转。
} PTA2CE, *PPTA2CE;
```

PPTSTS Status: 点表的状态。

```
typedef struct
{
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作, 0 : 停止]
                // b1 : 点缓冲区是否已满？ [1 : 完整, 0 : 未完整]
                // b2 : 点缓冲区是否为空？ [1 : 空, 0 : 非空]
                // b3, b4, b5 : 保留以备将来之用。
                // b6~ : 始终为 0
    U16 PntBufFreeSpace; //点缓冲区的可用空间
    U16 PntBufUsageSpace; //点缓冲区的使用空间
    U32 RunningCnt; //启用点表后要执行多少点
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTA2CE Prof;
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
if ( !( Status.Bitsts & 0x02 ) )//点缓冲区未满
{
    //将运动推入点缓冲区
    Prof.Index[0] = 0; //选择尺寸 0
    Prof.Index[1] = 1; //选择尺寸 1
    Prof.Center[0] = 10000;
    Prof.Center[1] = 10000;
    Prof.End[0] = 0;
    Prof.End[1] = 0;
    Prof.Dir = 0; //正方向
    ret = APS_pt_arc2_ce( Board_ID, PtId, &Prof, &Status );
}

//启动点表运动
APS_pt_start( Board_ID, PtId );
```

还可以看看:

APS_pt_arc3_ca	将一个 3D 圆弧运动推送到点表的点缓冲区。
----------------	------------------------

描述:

该函数将一个 3D 圆弧运动推送到点表的点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。 用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_arc3_ca( I32 Board_ID, I32 PtId, PPTA3CA Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_arc3_ca (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTA3CA,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTA2CA Prof: 带角度的 3d 圆弧运动曲线

```
typedef struct  
{  
    U8      Index[3]; // [0~点表尺寸] 点表中哪个轴索引  
    F64     Center[3]; // 中心位置  
    F64     Noraml[3]; // 法线向量  
    F64     Angle; // 角度，单位为弧度  
} PTA3CA, *PPTA3CA;
```

PPTSTS Status: 点表的状态。

```
typedef struct  
{  
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作, 0 : 停止]  
                // b1 : 点缓冲区是否已满？ [1 : 完整, 0 : 未完整]  
                // b2 : 点缓冲区是否为空？ [1 : 空, 0 : 非空]  
                // b3, b4, b5 : 保留以备将来之用。  
                // b6~ : 始终为 0  
    U16 PntBufFreeSpace; // 点缓冲区的可用空间  
    U16 PntBufUsageSpace; // 点缓冲区的使用空间  
    U32 RunningCnt; // 启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
PTA3CA Prof;
PTSTS Status;

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//获取点表的状态 0
ret = APS_get_pt_status(Board_ID , PtId, &Status );
if ( !( Status.Bitsts & 0x02 ) )//点缓冲区未满
{
    //将运动推入点缓冲区
    Prof.Index[0] = 0; //选取尺寸索引 0
    Prof.Index[1] = 1; //选取尺寸索引 1
    Prof.Index[2] = 2; //选取尺寸索引 2
    Prof.Center[0] = 10000;
    Prof.Center[1] = 10000;
    Prof.Center[2] = 10000;
    Prof.Normal[0] = 0;
    Prof.Normal[1] = 0;
    Prof.Normal[2] = 1;
    Prof.Angle = 3.14159265; //在弧度中
    ret = APS_pt_arc3_ca( Board_ID, PtId, &Prof, &Status );
}

//启动点表运动
APS_pt_start( Board_ID, PtId );
```

还可以看看:

APS_pt_arc3_ce	将一个 3D 圆弧运动推送到点表的点缓冲区。
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于将一个 3D 圆弧运动推送到点表的点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。 用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_arc3_ce( I32 Board_ID, I32 PtId, PPTA3CE Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_arc3_ce (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTA3CE,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTA3CE Prof: 带终点位置的 3d 圆弧运动曲线

```
typedef struct
{
    U8      Index[3]; // [0~点表的尺寸]点表中的哪个轴索引
    F64     Center[3]; //中心位置
    F64     End[3]; //终点位置
    I16     Dir; //值指定了旋转的方向，如果 Dir 设置为 0 表示正向旋转，则 dir = -1  
沿负向旋转。
} PTA3CE, *PPTA3CE;
```

PPTSTS Status: 点表的状态。

```
typedef struct
{
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作，0 : 停止]
                // b1 : 点缓冲区是否已满？ [1 : 完整，0 : 未完整]
                // b2 : 点缓冲区是否为空？ [1 : 空，0 : 非空]
                // b3, b4, b5 : 保留以备将来之用。
                // b6~ : 始终为 0
    U16 PntBufFreeSpace; //点缓冲区的可用空间
    U16 PntBufUsageSpace; //点缓冲区的使用空间
    U32 RunningCnt; //启用点表后要执行多少点
```

```
 } PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

Refer to APS_pt_arc3_ca().

还可以看看:

APS_pt_spiral_ca	将一个螺旋运动推送到点表的点缓冲区。
------------------	--------------------

描述:

该函数用于将一个螺旋运动推送到点表的点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。 用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_spiral_ca( I32 Board_ID, I32 PtId, PPTHCA Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_spiral_ca (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTHCA,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTHCA Prof: 带角度的螺旋运动曲线

```
typedef struct  
{  
    U8      Index[3]; // [0~点表的尺寸]点表中的哪个轴索引  
    F64     Center[3]; //中心位置  
    F64     Noraml[3]; //法线向量  
    F64     Angle; //角度，单位为弧度  
    F64     DeltaH; //螺旋运动的高度，用户单位，  
    F64     FinalR; //从终点位置到法线向量的距离，用户单位  
} PTHCA, *PPTHCA;
```

PPTSTS Status: 点表的状态。

```
typedef struct  
{  
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作，0 : 停止]  
                // b1 : 点缓冲区是否已满？ [1 : 完整，0 : 未完整]  
                // b2 : 点缓冲区是否为空？ [1 : 空，0 : 非空]  
                // b3 , b4 , b5 : 保留以备将来之用。  
                // b6~ : 始终为 0  
    U16 PntBufFreeSpace; //点缓冲区的可用空间  
    U16 PntBufUsageSpace; //点缓冲区的使用空间
```

```
    U32 RunningCnt; //启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

Refer to APS_pt_arc3_ca().

还可以看看:

APS_pt_spiral_ce	将一个螺旋运动推送到点表的点缓冲区。
------------------	--------------------

描述:

该函数用于将带有终点位置的螺旋运动推入点缓冲区。点表中最多有 50 个点缓冲区可用于预存储点。 用户可以监视点缓冲区的使用情况/可用空间以推入运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_spiral_ce( I32 Board_ID, I32 PtId, PPTHCE Prof, PPTSTS Status );
```

Visual Basic:

```
APS_pt_spiral_ce (ByVal Board_ID As Long, ByVal PtId As Long, ByRef Prof As PTHCE,  
ByRef Status As PTSTS) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

PPTHCE Prof: 带终点位置的螺旋运动曲线

```
typedef struct
{
    U8      Index[3]; // [0~点表的尺寸]点表中的哪个轴索引
    F64     Center[3]; //中心位置
    F64     Noraml[3]; //法线向量
    F64     End[3]; //终点位置
    I16     Dir; //值指定了旋转的方向，如果 Dir 设置为 0 表示正向旋转，则 dir = -1  
沿负向旋转。
} PTHCE, *PPTHCE;
```

PPTSTS Status: 点表的状态。

```
typedef struct
{
    U16 BitSts; // b0 : PT 是否有效？ [1 : 工作， 0 : 停止]
                // b1 : 点缓冲区是否已满？ [1 : 完整， 0 : 未完整]
                // b2 : 点缓冲区是否为空？ [1 : 空， 0 : 非空]
                // b3 , b4 , b5 : 保留以备将来之用。
                // b6~ : 始终为 0
    U16 PntBufFreeSpace; //点缓冲区的可用空间
    U16 PntBufUsageSpace; //点缓冲区的使用空间
```

```
    U32 RunningCnt; //启用点表后要执行多少点  
} PTSTS, *PPTSTS;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

Refer to APS_pt_arc3_ca().

还可以看看:

APS_pt_ext_set_do_ch	将 Do 扩展命令设置到命令缓冲区。当将一个运动推送到点表中时，命令缓冲区处于活动状态。/在 EPS-6000 从站的高级点位表内控制数字输出
----------------------	---

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

对于 PCI-8254/58 / AMP-204/8C，此函数用于将 Do 扩展命令设置到命令缓冲区。当将一个运动推送到点表中时，命令缓冲区处于活动状态。推入一个运动后，命令缓冲区将自动清除。用户最多可以将 7 条命令写入命令缓冲区。然后，将动作推入点位表会将这些命令一起带入点位表。现在可以支持 do 命令。其他命令保留以备将来使用。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

对于 PCIe-833x，此函数仅在拓扑中的数字输出模块内支持凌华科技 EPS-6000 从站，并且该从站应放置在拓扑中的第一位置。执行高级点位表函数时，此函数用于控制数字输出。当命令位置到达目标位置时，用户可以使用此 API 来控制数字输出。

句法:

C/C++:

```
I32 FNTYPE APS_pt_ext_set_do_ch( I32 Board_ID, I32 PtId, I32 Channel, I32 OnOff );
```

Visual Basic:

```
APS_pt_ext_set_do_ch (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Channel As Long, ByVal OnOff As Long) As Long
```

参数:

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 Channel: Do 通道

I32 OnOff: Do 开/关. 1: 开, 0: 关.

对于 PCIe-833x：

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 Channel: 数字量输出模块的通道号。

I32 OnOff: 0: 将数字输出值设置为 0。

1: 将数字输出值设置为 1。 .

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Board_ID = 0;
```

```
I32 PtId = 0; //点表 0
```

```
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
```

```
//将 do 扩展命令设置为命令缓冲区//将 do 通道 0 设置为打开
```

```
ret = APS_pt_ext_set_do_ch( Board_ID, PtId, 0, 1 );
```

```
//推入点缓冲区
```

还可以看看:

APS_pt_ext_set_table_no	将 VAO 表编号扩展命令设置为命令缓冲区。推入点表时，命令缓冲区处于活动状态。
-------------------------	--

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

该函数用于将 VAO 表编号扩展命令设置为命令缓冲区。推入点表时，命令缓冲区处于活动状态。推入运动后，命令缓冲区将自动清除。用户最多可以将 7 条命令写入命令缓冲区。然后，将运动推入点位表会将这些命令一起带入点位表。现在，支持表编号命令。其他命令保留以备将来使用。

注意：必须先调用 APS_pt_enable() 才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_ext_set_table_no( I32 Board_ID, I32 PtId, I32 CtrlNo, I32
TableNo );
```

Visual Basic:

```
APS_pt_ext_set_table_no (ByVal Board_ID As Long, ByVal PtId As Long, ByVal CtrlNo
As Long, ByVal TableNo As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

I32 CtrlNo: 控制编号为 0 到 1。

I32 TableNo: VAO 表编号是 -1 至 7。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

//将表编号扩展命令设置到命令缓冲区//将 VAO 表编号设置为 0，并将控制编号设置为 0

```
ret = APS_pt_ext_set_table_no( Board_ID, PtId, 0, 0 );
```

//推入点缓冲区

还可以看看:

APS_pt_set_absolute	将绝对曲线设置到曲线缓冲区
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于将绝对曲线设置到曲线缓冲区。推入点表时，轮廓缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_absolute ( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_set_absolute (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

//将绝对曲线设置到曲线缓冲区

```
ret = APS_pt_set_absolute ( Board_ID, PtId );
//推入点缓冲区
```

还可以看看:

APS_pt_set_relative	将相对曲线设置到曲线缓冲区
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

此函数用于将相对曲线设置到曲线缓冲区。将运动推入点表时，曲线缓冲区处于活动状态。 用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_relative ( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_set_relative (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//将相对曲线设置到曲线缓冲区
ret = APS_pt_set_relative ( Board_ID, PtId );
//推入点缓冲区
```

还可以看看:

APS_pt_set_trans_buffered	在曲线缓冲区中将转换设置为缓冲模式
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于在曲线缓冲区中将转换设置为缓冲模式。 将运动推入点表时，曲线缓冲区处于活动状态。 用户通常在启用点表之后一开始就开始一起设置曲线。 如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。 相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_buffered( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_set_trans_buffered (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。 通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//设置为缓冲模式
ret = APS_pt_set_trans_buffered( Board_ID, PtId );
//推入点缓冲区
```

还可以看看:

APS_pt_set_trans_inp	在曲线缓冲区中将转换设置为就位模式
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于在曲线缓冲区中将转换设置为就位模式。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_inp( I32 Board_ID, I32 PtId );
```

Visual Basic:

```
APS_pt_set_trans_inp (ByVal Board_ID As Long, ByVal PtId As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
// 设置为就位模式。
ret = APS_pt_set_trans_inp( Board_ID, PtId );
//推入点缓冲区
```

还可以看看:

APS_pt_set_trans_blend_dec	在曲线缓冲区中将转换设置为带减速的混合模式
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于在曲线缓冲区中将转换设置为带减速的混合模式。将运动推入点表时，曲线缓冲区处于活动状态。 用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_blend_dec( I32 Board_ID, I32 PtId, F64 Bp );
```

Visual Basic:

```
APS_pt_set_trans_blend_dec (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Bp  
As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Bp: 减速率. [Bp > 0, unit/s^2]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

// 设置为带减速的混合模式

```
ret = APS_pt_set_trans_blend_dec( Board_ID, PtId, 10000 );  
//推入点缓冲区
```

还可以看看:

APS_pt_set_trans_blend_dist	在曲线缓冲区中将转换设置为带剩余距离的混合模式
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于在曲线缓冲区中将转换设置为带剩余距离的混合模式。将运动推入点表时，曲线缓冲区处于活动状态。 用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_blend_dist( I32 Board_ID, I32 PtId, F64 Bp );
```

Visual Basic:

```
APS_pt_set_trans_blend_dist (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Bp  
As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Bp: 剩余距离。单位是用户单位，一般是脉冲。 [Bp > = 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0
```

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。

// 设置为带剩余距离的混合模式

```
ret = APS_pt_set_trans_blend_dist( Board_ID, PtId, 100 );  
//推入点缓冲区
```

还可以看看:

APS_pt_set_trans_blend_pcnt	在曲线缓冲区中将转换设置为带剩余距离百分比的混合模式
-----------------------------	----------------------------

支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x

描述:

该函数用于在曲线缓冲区中将转换设置为带剩余距离百分比的混合模式。将运动推入点表时，曲线缓冲区处于活动状态。 用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。 否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_blend_pcnt( I32 Board_ID, I32 PtId, F64 Bp );
```

Visual Basic:

```
APS_pt_set_trans_blend_pcnt (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Bp  
As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Bp: 剩余距离（以行进距离的百分比表示）。 单位是%。 [Bp : 0.0~1.0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0  
  
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。  
// 设置为带剩余距离百分比的混合模式。  
ret = APS_pt_set_trans_blend_pcnt( Board_ID, PtId, 0.05 );  
//推入点缓冲区
```

还可以看看:

APS_pt_set_acc	将加速曲线设置到曲线缓冲区中。
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将加速曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_acc( I32 Board_ID, I32 PtId, F64 Acc );
```

Visual Basic:

```
APS_pt_set_acc (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Acc As Double)  
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial()来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Acc: 加速率. [unit/s^2, > 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Board_ID = 0;
```

```
I32 PtId = 0; //点表 0
```

```
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
```

```
//设置加速度为 10000
```

```
ret = APS_pt_set_acc( Board_ID, PtId, 10000 );
```

```
//推入点缓冲区
```

还可以看看:

APS_pt_set_dec	将减速曲线设置到曲线缓冲区中。
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将减速曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_dec( I32 Board_ID, I32 PtId, F64 Dec );
```

Visual Basic:

```
APS_pt_set_dec (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Dec As Double)
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Dec: 减速率, [unit/s^2, > 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//设置减速度为 10000
ret = APS_pt_set_dec( Board_ID, PtId, 10000 );
//推入点缓冲区
```

还可以看看:

APS_pt_set_acc_dec	将加减速曲线设置到曲线缓冲区中
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将加减速曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_acc_dec( I32 Board_ID, I32 PtId, F64 AccDec );
```

Visual Basic:

```
APS_pt_set_acc_dec (ByVal Board_ID As Long, ByVal PtId As Long, ByVal AccDec As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 AccDec: 加/减速率. [unit/s^2, > 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//设置加速度/减速度为 10000
ret = APS_pt_set_acc_dec( Board_ID, PtId, 10000 );
//推入点缓冲区
```

还可以看看:

APS_pt_set_s	将 S-factor 曲线设置到曲线缓冲区中
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将 S-factor 曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_s( I32 Board_ID, I32 PtId, F64 Sf );
```

Visual Basic:

```
APS_pt_set_s (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Sf As Double) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Sf: s-factor [0 ~ 1]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0  
  
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。  
//设置 s-factor 为 0.5  
ret = APS_pt_set_s( Board_ID, PtId, 0.5 );  
//推入点缓冲区
```

还可以看看:

APS_pt_set_vm	将最大速度曲线设置到曲线缓冲区中
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将最大速度曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_vm( I32 Board_ID, I32 PtId, F64 Vm );
```

Visual Basic:

```
APS_pt_set_vm (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Vm As Double)
As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Vm : 最大速度 [Vm >= 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //点表 0

//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。
//设置 Vm 为 10000
ret = APS_pt_set_vm( Board_ID, PtId, 10000 );
//推入点缓冲区
```

还可以看看:

APS_pt_set_ve	将终点速度曲线设置到曲线缓冲区中
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于将终点速度曲线设置到曲线缓冲区中。将运动推入点表时，曲线缓冲区处于活动状态。用户通常在启用点表之后一开始一起设置曲线。如果用户想要更改某些曲线，则可以在将其运动到点缓冲区之前对其进行修改。相反，如果用户不想修改曲线，则相同的曲线将自动保留以用于后续运动。

注意：必须先调用 APS_pt_enable()才能启用点表。否则，它将是返回错误代码。

注意：请勿同时使用馈线函数调用 Pt 函数，因为它们是独占的。

句法:

C/C++:

```
I32 FNTYPE APS_pt_set_ve( I32 Board_ID, I32 PtId, F64 Ve );
```

Visual Basic:

```
APS_pt_set_ve (ByVal Board_ID As Long, ByVal PtId As Long, ByVal Ve As Double) As  
Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PtId: 点表 ID 为 0 到 1。

F64 Ve: 终止速度 [Ve >= 0]

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //点表 0  
  
//轴 0 和轴 1 启用点表 0 至 2d 的尺寸。  
//设置 Ve 为 100  
ret = APS_pt_set_ve( Board_ID, PtId, 100 );  
//推入点缓冲区
```

还可以看看:

16. 现场总线函数

APS_set_field_bus_param	设置现场总线相关参数
-------------------------	------------

支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO

描述:

此函数用于设置现场总线的系统参数。用户必须先使用此函数，然后才能开始现场总线通讯。

否则，默认情况下将启动现场总线。有关参数的详细信息，请参见现场总线参数表。

现场总线是在工业领域中使用的一种串行网络总线。最受欢迎的一种是 CAN 总线。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_param( I32 Board_ID, I32 BUS_No, I32 BUS_Param_No,  
I32 BUS_Param );
```

Visual Basic:

```
APS_set_field_bus_param( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal  
BUS_Param_No As Long, ByVal BUS_Param As Long)As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 BUS_Param_No: 现场总线参数编号，请参见表 491peration491

I32 BUS_Param: 现场总线参数数据。请参阅表定义。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_get_field_bus_param\(\)](#); [APS_start_field_bus\(\)](#)

APS_get_field_bus_param	获取现场总线相关参数
支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO	

描述:

此函数用于获取现场总线系统参数。请参考现场总线参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_param( I32 Board_ID, I32 BUS_No, I32 BUS_Param_No,
I32 *BUS_Param );
```

Visual Basic:

```
APS_get_field_bus_param( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
BUS_Param_No As Long, BUS_Param As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 BUS_Param_No: 现场总线参数编号 , 请参见表 492peration492

I32 *BUS_Param: Return 现场总线参数数据。请参阅表定义。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_field_bus_param\(\)](#)

APS_scan_field_bus	扫描现场总线并生成 ENI 文件
支持的产品:PCIe-833x	

描述:

该函数用于扫描现场总线并生成 ENI 文件。首次使用时，用户应在使用 APS_start_field_bus() 之前调用此函数。

句法:

C/C++:

```
I32 FNTYPE APS_scan_field_bus( I32 Board_ID, I32 BUS_No )
```

Visual Basic:

```
APS_scan_field_bus (ByVal Board_ID As Long, ByVal BUS_No As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。（端口号）值：仅支持数字0。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

案例 1 :

首次使用 PCIe-833x (不存在 ENI 文件)

```
APS_scan_field_bus( 0, 0 ); //扫描现场总线并首先生成 ENI 文件
```

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 2 :

ENI 文件确实存在，并且拓扑不会更改。

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 3 :

ENI 文件确实存在，并且拓扑确实发生了变化。

```
APS_scan_field_bus( 0, 0 ); //扫描现场总线并首先生成新的 ENI 文件
```

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

还可以看看：

```
APS_start_field_bus(); APS_stop_field_bus()
```

注意：在生成 ENI 之前，此函数将分别删除 EniBuilderForCpp.log 和

ADLINK_Config2.xml。如果 ENI 生成过程成功，则用户可以在 ENI 文件夹中找到

ADLINK_Config2.xml (ENI 文件) 以连接 EtherCAT 网络。否则，此函数将返回错误代码，

并且用户可以参考 EniBuilderForCpp.log 了解详细信息。

<code>APS_start_field_bus</code>	启动指定现场总线的网络
支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO , PCIe-833x	

描述:

该函数用于启动现场总线通讯。一旦启动，它将搜索连接到该端口的所有模块。由于端口上可能有运动从站，因此用户在使用此函数时应分配起始轴 ID。端口的所有轴将从起始轴 ID 开始轴 ID 排列。

即使在端口上只有 I/O 从站，也应在使用现场总线之前调用此函数。

请注意，由于从站会自动搜索，因此某些从站可能会由于通信质量的原因而丢失。用户必须检查所有能找到的从站，并且确认其类型正确，然后才能进行现场总线操作。

`APS_stop_field_bus()` 必须在总线操作结束时调用。

对于 PCIe-833x，此函数用于启动现场总线通信。一旦启动，它将搜索连接到该端口的所有模块。用户应在使用现场总线之前调用此函数。

必须在提交的总线操作结束时调用 `APS_stop_field_bus()`。

首次使用 PCIe-833x 时，用户应在使用 `APS_scan_field_bus()` 之前先调用 `APS_start_field_bus()`。

句法:

C/C++:

```
I32 FNTYPE APS_start_field_bus( I32 Board_ID, I32 BUS_No, I32 Starting_Axis_ID );
```

Visual Basic:

```
APS_start_field_bus( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal  
Starting_Axis_ID As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用`APS_initial()`来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

对于PCI(e)-7856，HSL现场总线为Bus_No 0，而MNET现场总线为Bus_No 1。

I32 Starting_Axis_ID: 该现场总线编号的起始轴 ID 号。

对于 PCIe-833x：

I32 Board_ID: 目标控制器的ID。通过成功调用`APS_initial()`来检索它。

I32 BUS_No: 现场总线编号。(端口号) 值：仅支持编号0。

I32 Starting_Axis_ID: 不用关心。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0; //总线编号。  
I32 startingAxisId = 1000; //现场总线的起始轴 ID。  
  
Ret = APS_start_field_bus( boardId, busNum, startingAxisId );  
// 现场总线操作...  
APS_stop_field_bus(boardId, busNum ); //停止现场总线。
```

示例 2:

案例 1 :

```
首次使用 PCIe-833x ( 不存在 ENI 文件 )  
APS_scan_field_bus( 0, 0 ); // 扫描现场总线并首先生成 ENI 文件  
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯  
...  
现场总线操作...  
做具体的任务...  
....  
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 2 :

```
ENI 文件确实存在 , 并且拓扑不会更改。  
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯  
...  
现场总线操作...  
做具体的任务...  
....  
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 3 :

```
ENI 文件确实存在 , 并且拓扑确实发生了变化。  
APS_scan_field_bus( 0, 0 ); //扫描现场总线并首先生成新的 ENI 文件  
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯  
...  
现场总线操作...  
做具体的任务...  
....  
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

还可以看看:

APS_stop_field_bus()

APS_stop_field_bus	停止指定现场总线的网络
--------------------	-------------

支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO , PCIe-833x

描述:

此函数用于停止现场总线通讯并释放其资源。

如果用户曾经使用 APS_start_field_bus() 来启动网络，则必须在过程结束时调用此函数。

句法:

C/C++:

```
I32 FNTYPE APS_stop_field_bus( I32 Board_ID, I32 BUS_No );
```

Visual Basic:

```
APS_stop_field_bus( ByVal Board_ID As Long, ByVal BUS_No As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

对于PCI(e)-7856 , HSL现场总线为Bus_No 0 , 而MNET现场总线为Bus_No 1。

对于PCIe-833x, I32 BUS_No: 现场总线编号。 (端口号) 值 : 仅支持编号0。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

```
I32 ret; //返回错误代码。
I32 boardId = 0;
I32 busNum = 0; //总线编号。
I32 startingAxisId = 1000; //现场总线的起始轴 ID。
```

```
Ret = APS_start_field_bus(boardId, busNum, startingAxisId );
```

```
// 现场总线操作...
```

```
APS_stop_field_bus(boardId, busNum ); //停止现场总线。
```

示例 2:

案例 1 :

首次使用 PCIe-833x (不存在 ENI 文件)

```
APS_scan_field_bus( 0, 0 ); // 扫描现场总线并首先生成 ENI 文件
```

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 2 :

ENI 文件确实存在 , 并且拓扑不会更改。

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

案例 3 :

ENI 文件确实存在 , 并且拓扑确实发生了变化。

```
APS_scan_field_bus( 0, 0 ); //扫描现场总线并首先生成新的 ENI 文件
```

```
APS_start_field_bus( 0, 0, 0 ); //开始现场总线通讯
```

...

现场总线操作...

做具体的任务...

....

```
APS_stop_field_bus( 0, 0 ); //停止现场总线通讯
```

还可以看看:

```
APS_start_field_bus()
```

APS_field_bus_d_set_output	设置现场总线数字输出
----------------------------	------------

支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO

描述:

此函数用于设置从站模块上的现场总线数字输出。一个模块 ID 的最大数据长度为 32 位。如果模块 ID 的通道数少于 32，则在输出时，较高的位必须保持为零。高位回读数据为零。

注意：对于 HSL_DI56DO32_FCN 模块，用户应调用 APS_field_bus_d_set_output_ex() 进行 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_set_output( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32 DO_Value );
```

Visual Basic:

```
APS_field_bus_d_set_output( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal MOD_No As Long, ByVal DO_Value As Long )As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场总线，模块编号的范围是 1 到 63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

I32 DO_Value: 数字输出值。以位格式。Bit 0 对应数字输出通道 0，其余部分依此类推。

对于 MNET-4XMO , DO 位的定义如下。默认值为 0xff。

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOIF4.Do2	IOIF3.Do2	IOIF2.Do2	IOIF1.Do2	IOIF4.Do1	IOIF3.Do1	IOIF2.Do1	IOIF1.Do1

对于 MNET-4XMO-C 和 HSL-4XMO , DO 位的定义如下。默认值为 0xf。

Bit3	Bit2	Bit1	Bit0
IOIF4.Do1	IOIF3.Do1	IOIF2.Do1	IOIF1.Do1

对于 MNET-1XMO , DO 位的定义如下。默认值为 0x0。

Bit3	Bit2	Bit1	Bit0
N/A	SZST 0(Low) 1(High)	STL 0(Low) 1(High)	AlmReset 0(Low) 1(High)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
I32 DO_Value = 0;  
  
//首先启动现场总线。  
// ret = APS_start_field_bus( boardId, busNum, startingAxisId );  
DO_Value = 0xF;  
ret = APS_field_bus_d_set_output(boardId, busNum,, moduleNum, DO_Value );
```

还可以看看:

[APS_field_bus_d_get_output\(\)](#)

APS_field_bus_d_get_output	获取现场总线数字输出
支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO	

描述:

该函数用于获取从站模块上的现场总线数字输出。某些模块 ID 无法被回读其输出信息。请检查每个模块的硬件规格。一个模块 ID 的最大数据长度为 32 位。如果模块 ID 的通道数少于 32，则在输出时，较高的位必须保持为零。高位回读的数据将为零。

注意：对于 HSL_DI56DO32_FCN 模块，用户应调用 APS_field_bus_d_get_output_ex() 进行 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_get_output( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
*DO_Value );
```

Visual Basic:

```
APS_field_bus_d_get_output( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, DO_Value As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场总线，模块编号的范围是 1 到 63。 注意：在 HSL 中， MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

I32 *DO_Value: 返回数字输出值。Bit 0 对应数字输出通道 0，其余部分依此类推。

对于 MNET-4XMO，DO 位的定义如下。默认值为 0xff。

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOIF4.Do2	IOIF3.Do2	IOIF2.Do2	IOIF1.Do2	IOIF4.Do1	IOIF3.Do1	IOIF2.Do1	IOIF1.Do1

对于 MNET-4XMO-C 和 HSL-4XMO，DO 位的定义如下。默认值为 0xf。

Bit3	Bit2	Bit1	Bit0
IOIF4.Do1	IOIF3.Do1	IOIF2.Do1	IOIF1.Do1

对于 MNET-1XMO，DO 位的定义如下。默认值为 0x0。

Bit3	Bit2	Bit1	Bit0
N/A	SZST 0(Low) 1(High)	STL 0(Low) 1(High)	AlmReset 0(Low) 1(High)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
I32 DO_Value = 0;  
  
//首先启动现场总线。  
// ret = APS_start_field_bus( boardId, busNum, startingAxisId );  
  
ret = APS_field_bus_d_get_output(boardId, busNum, moduleNum, &DO_Value );
```

还可以看看:

[APS_field_bus_d_set_output\(\)](#)

APS_field_bus_d_get_input	获取现场总线数字输入
---------------------------	------------

支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), HSL-4XMO, HSL-DIO

描述:

此函数用于从从站模块上的现场总线数字输入获取输入数据。一个模块 ID 的最大数据长度为 32 位。如果模块 ID 的通道数少于 32，则较高的位必须保持为零。

注意：对于 HSL_DI56DO32_FCN 模块，用户应调用 APS_field_bus_d_get_input_ex() 进行 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_get_input( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
*DI_Value );
```

Visual Basic:

```
APS_field_bus_d_get_input( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, DI_Value As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场总线，模块编号的范围是 1 到 63。 注意：在 HSL 中， MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

I32 *DI_Value: 返回数字输入值。

对于 MNET-4XMO , DI 位的定义如下。

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOIF4.Di2	IOIF3.Di2	IOIF2.Di2	IOIF1.Di2	IOIF4.Di1	IOIF3.Di1	IOIF2.Di1	IOIF1.Di1

对于 MNET-4XMO-C 和 4XMO , DI 位的定义如下。

Bit3	Bit2	Bit1	Bit0
IOIF4.Di1	IOIF3.Di1	IOIF2.Di1	IOIF1.Di1

对于 MNET-1XMO , DI 位的定义如下。

Bit3	Bit2	Bit1	Bit0
N/A	N/A	N/A	STLOV

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
I32 DI_Value = 0;  
  
//首先启动现场总线。  
//  ret = APS_start_field_bus( boardId, busNum, startingAxisId );  
ret = APS_field_bus_d_get_input( boardId, busNum, moduleNum, &DI_Value );
```

还可以看看：

APS_field_bus_d_set_output();APS_field_bus_d_get_output()

APS_field_bus_d_set_output_ex

为 64 位操作设置现场总线数字输出

支持的产品:PCI(e)-7856

描述:

该函数用于为 64 位 DIO 操作设置从站模块上的现场总线数字输出。一个模块 ID 的最大数据长度为 64 位。如果模块 ID 的通道数少于 64，则在输出时，较高的位必须保持为零。高位回读的数据将为零。

注意：仅在 HSL_DI56DO32_FCN 模块上可用于 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_set_output_ex( I32 Board_ID, I32 BUS_No, I32 MOD_No ,  
DO_DATA_EX DO_Value );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场总线，模块编号的范围是 1 到 63。 注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

DO_DATA_EX DO_Value: 数字输出值。Bit 0 对应数字输出通道 0，其余部分依此类推。 其结构的定义如下所示：

```
typedef struct
```

```
{  
    U32 Do_ValueL; //bit[0~31]  
    U32 Do_ValueH; //bit[32~63]  
} DO_DATA_EX, *PDO_DATA_EX;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
DO_DATA_EX DO_Value = {0, 0};
```

```
//首先启动现场总线。  
// ret = APS_start_field_bus( 507 boardId, busNum, startingAxisId );  
DO_Value. Do_ValueL = 0x0F; // 开启 bit 0 ~ 3  
DO_Value. Do_ValueH = 0x00;  
ret = APS_field_bus_d_set_output_ex(507boardId, busNum,, moduleNum, DO_Value );
```

还可以看看:

APS_field_bus_d_get_output_ex()

APS_field_bus_d_get_output_ex

为 64 位操作获取现场总线数字输出

支持的产品:PCI(e)-7856

描述:

该函数用于为 64 位 DIO 操作获取从站模块上的现场总线数字输出。一个模块 ID 的最大数据长度为 64 位。如果模块 ID 的通道数少于 64，则在输出时，较高的位必须保持为零。高位回读的数据将为零。

注意：仅在 HSL_DI56DO32_FCN 模块上可用于 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_get_output_ex( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
DO_DATA_EX *DO_Value );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场总线，模块编号的范围是 1 到 63。 注意：在 HSL 中，
MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

DO_DATA_EX *DO_Value: 返回数字输出值。Bit 0 对应数字输出通道 0，其余部分依此类推。其结构的定义如下所示：

```
typedef struct
```

```
{  
    U32 Do_ValueL; //bit[0~31]  
    U32 Do_ValueH; //bit[32~63]  
} DO_DATA_EX, *PDO_DATA_EX;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
DO_DATA_EX DO_Value = {0, 0};
```

```
//首先启动现场总线。  
// ret = APS_start_field_bus( 509 boardId, busNum, startingAxisId );  
  
ret = APS_field_bus_d_get_output_ex(509boardId, busNum, moduleNum,  
&DO_Value );
```

还可以看看:

APS_field_bus_d_set_output_ex()

APS_field_bus_d_get_input_ex	为 64 位 DIO 操作获取现场总线数字输入
------------------------------	-------------------------

支持的产品:PCI(e)-7856

描述:

该函数用于为 64 位 DIO 操作获取从站模块上的现场总线数字输入。一个模块 ID 的最大数据长度为 64 位。如果模块 ID 的通道数少于 64，则在输出时，较高的位必须保持为零。

注意：仅在 HSL_DI56DO32_FCN 模块上可用于 64 位 DIO 操作。

句法:

C/C++:

```
I32 FNTYPE APS_field_bus_d_get_input_ex( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
DI_DATA_EX *DI_Value );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于高速链接 (HSL) 型现场模块编号的范围是 1 到 63。 注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

对于 MNET 型现场总线，模块编号的范围是 0 到 63。

I32 *DI_Value: 返回数字输入值. Bit 0 对应数字输出通道 0，其余部分依此类推。其结构的定义如下所示：

```
typedef struct  
{  
    U32 Di_ValueL; //bit[0~31]  
    U32 Di_ValueH; //bit[32~63]  
} DI_DATA_EX, *PDI_DATA_EX;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 0;  
DI_DATA_EX DI_Value = { 0, 0 };  
  
//首先启动现场总线。
```

```
ret = APS_start_field_bus( boardId, busNum, startingAxisId );  
  
//获取 64 位 DI 数据。  
ret = APS_field_bus_d_get_input_ex(511boardId, busNum, moduleNum, &DI_Value );
```

还可以看看：

APS_field_bus_d_set_output_ex(); APS_field_bus_d_get_output_ex()

APS_set_field_bus_slave_param	设置现场总线从站模块的参数
-------------------------------	---------------

支持的产品:PCI-8392H, DPAC-3000, PCI(e)-7856

描述:

该函数用于设置现场总线从站的参数。

一些参数用于从站模块本身，而某些则用于从站通道。它取决于输入参数“ I32 Ch_no”。当 Ch_no 设置为-1 时，表示将参数设置为指定的模块（模块层参数）。否则，通过将通道编号赋值给 CH_no，这样就把参数设置给指定的通道。

现场总线从站参数的详细信息，请参见从站参数表。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_slave_param( I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 Ch_No, I32 ParaNum, I32 ParaDat );
```

Visual Basic:

```
APS_set_field_bus_slave_param( ByVal Board_ID As Long, ByVal BUS_No As Long,
ByVal MOD_No As Long, ByVal Ch_No As Long, ByVal ParaNum As Long, ByVal
ParaDat As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 Ch_No: 通道编号。如果将此参数设置为-1，则表示设置从站参数。

-1：将参数设置给指定的从站模块编号

0 ~：将参数设置给指定的通道编号（AIO 通道，DIO 通道等）

I32 ParaNum: 从站/通道参数编号。

请参考现场总线从站参数定义表。

I32 ParaDat: 从站/通道参数数据。.

请参考现场总线从站参数定义表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_get_field_bus_slave_param\(\)](#)

APS_get_field_bus_slave_param	获取现场总线从站模块的参数
-------------------------------	---------------

描述:

该函数用于获取现场总线从站的参数。

一些参数用于从站模块本身，而某些则用于从站通道。它取决于输入参数“ I32 Ch_no”。当 Ch_no 设置为-1 时，表示将参数设置为指定的模块。否则，通过将通道编号赋值给 CH_no，这样就把参数设置给指定的通道。

现场总线从站参数的详细信息，请参见从站参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_slave_param( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Ch_No, I32 ParaNum, I32 *ParaDat );
```

Visual Basic:

```
APS_get_field_bus_slave_param( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32 Ch_No,  
I32 ParaNum, I32 *ParaDat );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 Ch_No: 通道编号。如果将此参数设置为-1，则表示设置从站参数。

-1 : 将参数设置给指定的从站模块编号

0 ~ : 将参数设置给指定的通道编号 (AIO 通道，DIO 通道等)

I32 ParaNum: 从站/通道参数编号。

请参考现场总线从站参数定义表。

I32 *ParaDat: 返回从站/通道参数数据。

请参考现场总线从站参数定义表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_field_bus_slave_param\(\)](#)

APS_set_field_bus_a_output	设置现场总线模拟输出
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856, PCIe-833x	

描述:

该函数用于设置现场总线的模拟类型，以输出模拟量输出值。从数字值到浮点值的转换是根据硬件规范实现的，并内置在 APS 中。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_a_output( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
Ch_No, F64 AO_Value );
```

Visual Basic:

```
APS_set_field_bus_a_output( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByVal Ch_No As Long, ByVal AO_Value As Double ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 Ch_No: 通道编号。值范围 0~n (n=最大通道数-1)

F64 AO_Value: 模拟输出。值的单位取决于从站类型。 [V]表示电压/ [A]表示电流。

对于 PCIe-833x :

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。 (仅支持索引0)

I32 MOD_No: 从站设备索引 (从 0 开始)

I32 Ch_No: 通道编号 (从 0 开始)

F64 AO_Value: 模拟输出。值的单位取决于从站类型。 [V]表示电压/ [A]表示电流。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_get_field_bus_a_output\(\)](#); [APS_get_field_bus_a_input\(\)](#)

APS_get_field_bus_a_output	获取现场总线模拟输出
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856	

描述:

此函数用于获取模拟类型的现场总线从站的模拟输出。从数字值到浮点值的转换是根据硬件规范实现的，并内置在 APS 中。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_a_output( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
Ch_No, F64 *AO_Value );
```

Visual Basic:

```
APS_get_field_bus_a_output(ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByVal Ch_No As Long, AO_Value As Double ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 Ch_No: 通道编号。值范围 0~n (n=最大通道数-1)

F64 *AO_Value: 返回模拟输出。值的单位取决于从站类型。 [V]表示电压/ [A]表示电流。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_field_bus_a_output\(\);](#) [APS_get_field_bus_a_input\(\)](#)

<code>APS_get_field_bus_a_input</code>	获取现场总线模拟输入
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856 , PCIe-833x	

描述:

此函数用于获取模拟类型的现场总线从站的模拟输入。从数字值到浮点值的转换是根据硬件规范实现的，并内置在 APS 中。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_a_input( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
Ch_No, F64 *AI_Value );
```

Visual Basic:

```
APS_get_field_bus_a_input(ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByVal Ch_No As Long, AI_Value As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 Ch_No: 通道编号。值范围 0~n (n=最大通道数-1)

F64 *AI_Value: 返回模拟输入。值的单位取决于从站类型。 [V]表示电压/ [A]表示电流。

对于 PCIe-833x:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线号。 (仅支持索引0)

I32 MOD_No: 从站设备索引 (从 0 开始)

I32 Ch_No: 通道编号。

F64 *AI_Value: 返回模拟输入。值的单位取决于从站类型。 [V]表示电压/ [A]表示电流。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

`APS_set_field_bus_a_output(); APS_get_field_bus_a_output()`

<code>APS_get_slave_connect_quality</code>	获取从站的连接质量
支持的产品:PCI-8392(H), DPAC-3000, PCI(e)-7856	

描述:

此函数用于获取从站的连接质量。

开始扫描从站模块后，可以使用该函数检查通讯是否发生错误。结果仅显示执行时的状态，而不显示历史记录中的状态。用户可以通过 PRF_CHKERRCNT_LAYER 参数设置检查的程度。返回值的范围取决于模块占用的 id 数。

必须再次指出，函数此时仅显示连接的质量。

注意：此函数支持 HSL 总线。

注意：此函数不支持 MotionNet 总线。

句法:

C/C++:

```
I32 FNTYPE APS_get_slave_connect_quality( I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 *Sts_data );
```

Visual Basic:

```
APS_get_slave_connect_quality (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByRef Sts_data As Long);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1. This function only supports HSL bus now.

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID: 1~63。注意：在 HSL 中，MOD_No 是模块占用的第一个 ID。

I32 *Sts_data : 返回状态值。返回值以位的形式呈现。每一位分别声明了每个 id 的通信状态。 0 是正常的，1 是异常的。

例如：

HSL 模块可能占用一个以上的 ID。您可以通过返回值识别每个 ID 的状态。但是，如果返回值大于零，则表示模块中的通信不稳定。

0x00(0): 所有 ID 都是正常的。

0x01(1): 第一个 ID 是异常的。

0x05(5): 第一和第三个 ID 是异常的。

0x0f(15) : 所有 ID 都是异常的。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

HSL 总线示例:

```
//如果模块占用 4 个 ID。  
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 0;  
I32 moduleNum = 1;  
I32 Sts_data = 0;  
I32 bus_param = 5;  
I32 startingAxisId = 0;  
//首先启动现场总线。  
Ret = APS_start_field_bus(boardId, busNum, startingAxisId );  
ret = APS_set_field_bus_param (boardId, busNum, PRF_CHKERRCNT_LAYER,  
bus_param );  
ret = APS_get_slave_connect_quality(boardId, busNum, moduleNum, &Sts_data );  
//如果 Sts_data 为 5 , 则表示第一个和第三个 ID 是异常的。
```

还可以看看:

APS_get_slave_online_status()

<code>APS_get_slave_online_status</code>	获取从站的连接质量/获取从站的状态
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO , PCIe-833x	

描述:

此函数用于获取在线状态。

开始扫描从站模块后，此函数可用于检查从站模块是处于在线还是离线状态。

必须注意，函数此时仅显示通信状态。

注意：此函数支持 HSL 和 MotionNet 总线。

注意：对于 HSL 总线，返回值的范围取决于模块所占用的位数。

对于 PCIe-833x，此函数用于获取从站的状态。启动现场总线后应执行此函数。

句法:

C/C++:

```
I32 FNTYPE APS_get_slave_online_status ( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
* Live );
```

Visual Basic:

```
APS_get_slave_online_status (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByRef Live);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 从站模块编号。

对于HSL从站模块，取决于从站ID：1~63。

注意：在HSL中，Module_No是模块占用的第一个ID。

对于MNET从站模块，取决于从站ID：0~63

I32 * Live : 返回状态值。返回值以位的形式呈现。每一位分别声明了每个ID的状态。0代表离线，1代表在线。

HSL总线示例:

HSL模块可能占用一个以上的ID。您可以通过返回值识别每个ID的状态。

0x00(0): 所有ID都离线

0x01(1): 第一个ID在线

0x05(5): 第一和第三个ID在线

0x0f(15) : 所有ID都在线

Mnet总线示例:

用户此时可以通过调用此函数来识别特定从站的通讯错误。如果特定的从站在三个连续的通讯周期中发生了通讯错误，它将发出通讯错误。

0x00(0): 该 ID 离线。也就是说，此 ID 发出通信错误。

0x01(1): 此 ID 在线。也就是说，此 ID 的通信良好。

对于 PCIe-833x :

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。（端口号）仅支持编号0。

I32 MOD_No: 从站编号ID。

I32 *Live : 通过位(bit)的定义返回从站的状态。

以下为位(bit)的定义 :

Bit 0:

0: 从站不在 1: 从站在

Bit 1:

0: 不进行总线扫描 1: 总线扫描

Bit 2:

0: 不初始化 1: 初始话

Bit 3:

0: 不 PreOP 1: PreOP

Bit 4:

0: 不 SafeOP 1: SafeOP

Bit 5:

0: 不 OP 1: OP

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 1 :

以下示例适用于 HSL 总线:

//如果模块占用 4 个 ID。

I32 ret; //返回错误代码。

I32 boardId = 0;

I32 busNum = 0; //HSL 总线编号

I32 moduleNum = 1;

I32 on_line = 0;

I32 bus_param = 5;

I32 startingAxisId = 0;

//首先启动现场总线。

Ret = APS_start_field_bus(boardId, busNum, startingAxisId);

ret = APS_get_slave_online_status (boardId, busNum, moduleNum, & on_line);

//如果 on_line 为 5 , 则表示第一个和第三个 ID 在线。

示例 2:

以下示例适用于 MotionNet 总线:

```
I32 ret; //返回错误代码。  
I32 boardId = 0;  
I32 busNum = 1; // MotionNet 总线编号  
I32 moduleNo = 10;  
I32 on_line = 0;  
  
//启动现场总线.  
//此时检查特定模块的通讯是否有错误。  
Ret = APS_get_slave_online_status (boardId, busNum, moduleNo, & on_line );
```

示例 3:

以下示例适用于 PCIe-833x

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
I32 MOD_No = 0;  
I32 Live = 0;  
  
ret = APS_get_slave_online_status ( Board_ID, BUS_No, MOD_No, &Live );  
if( ret == ERR_NoError )  
{  
    if ( Live & 0x1 )  
        printf( "This slave is present.\n" );  
    else  
        printf( "This slave is absent.\n" );  
}
```

还可以看看:

APS_get_slave_connect_quality()

APS_get_field_bus_master_status	获取现场总线主站状态
---------------------------------	------------

支持的产品:PCIe-833x

描述:

为了获取现场总线主站状态，如在 EtherCAT 定义中的 INIT 状态，SAFEOP 状态和 OP 状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_master_status( I32 Board_ID, I32 BUS_No, U32 *Status )
```

Visual Basic:

```
APS_get_field_bus_master_status (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByRef Status As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)值：仅支持编号0。

U32 *Status: 现场总线主站的返回状态。

现场总线主站的状态如下：

EC_STATE_NOT_RDY	(0x0000)
EC_STATE_RDY	(0x0001)
EC_STATE_BUS_SCAN	(0x0002)
EC_STATE_INIT	(0x0003)
EC_STATE_PREOP	(0x0004)
EC_STATE_SAFEOP	(0x0005)
EC_STATE_OP	(0x0006)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

还可以看看:

APS_get_field_bus_last_scan_info

 系统扫描后，获取现场总线信息。. |

支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO , PCIe-833x

描述:

该函数用于在系统扫描后获取现场总线的信息。请参考现场总线信息表。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_last_scan_info ( I32 Board_ID, I32 BUS_No, I32 *  
Info_Array, I32 Array_Size, I32 *Info_Count );
```

Visual Basic:

```
APS_get_field_bus_last_scan_info (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByRef Info_Array As Long, ByVal Array_Size As Long, ByRef Info_Count As Long);
```

参数:

For MNET:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 * Info_Array: 返回扫描信息。参见现场总线信息表。

I32 Array_Size: 用户想要获取的数组大小。

I32 * Info_Count: 返回实际的大小。

对于 MNET 现场总线信息表

数组索引	返回扫描现场总线信息
0	扫描后的从站总数。
1	扫描后的轴总数。

对于 PCIe-833x :

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。 (端口号) 仅支持编号0。

I32 * Info_Array: 返回扫描信息。参见现场总线信息表。

I32 Array_Size: 用户想要获取的数组大小。

I32 * Info_Count: 返回实际的大小。

对于 PCIe-833x 现场总线信息表

数组索引	返回扫描现场总线信息
0	扫描后的从站总数。
1	不支持。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 MNET

```
I32 ret;  
I32 Info_Array[2];  
I32 Info_Count;  
ret = APS_get_field_bus_last_scan_info ( 0, 1, & Info_Array, 2, & Info_Count );  
if( ret != ERR_NoError )  
{  
    //获取现场总线信息  
}
```

示例 2:

以下示例适用于 PCIe-833x

```
I32 ret;  
I32 Info_Array[1];  
I32 Info_Count;  
I32 Slave_Count;  
ret = APS_get_field_bus_last_scan_info ( 0, 1, & Info_Array, 1, & Info_Count );  
if( ret == ERR_NoError )  
{  
    //获取现场总线中的从站数量  
    Slave_Count = Info_Array[0];  
}
```

还可以看看:

APS_get_field_bus_master_type	获取现场总线上的主站类型
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO	

描述:

该函数用于获取现场总线上主站的类型。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_master_type( I32 Board_ID, I32 BUS_No, I32
*BUS_Type );
```

Visual Basic:

```
APS_get_field_bus_master_type(ByVal Board_ID As Long, ByVal BUS_No As Long, ByRef
BUS_Type As Long);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 * BUS_Type: 返回。

0 : 保留

1 : HSL

2 : MNET

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 BUS_Type;
ret = APS_get_field_bus_master_type ( 0, 1, & BUS_Type );
if( ret != ERR_NoError )
{
    // 获取现场总线的主站类型
}
```

还可以看看:

<code>APS_get_field_bus_slave_type</code>	获取现场总线上从站的类型
支持的产品:PCI-8392(H) , DPAC-3000, PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO	

描述:

该函数用于获取现场总线上从站的类型。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_slave_type( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
*MOD_Type );
```

Visual Basic:

```
APS_get_field_bus_slave_type(ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long , ByRef MOD_Type As Long);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块 , 取决于从站 ID : 1~63。在 HSL 中 , Module_No 是模块占用的第一个 ID。

对于 MNET 从站模块 , 取决于从站 ID : 0~63

I32 * MOD_Type: 返回。

0 : 保留

1 : HSL

2 : MNET

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 MOD_Type;
ret = APS_get_field_bus_slave_type ( 0, 1, 10, & MOD_Type );
if( ret != ERR_NoError )
{
    // 获取现场总线上从站的类型
}
```

还可以看看:

APS_get_field_bus_slave_name	获取现场总线上的从站名称
支持的产品:PCI-8392(H) , DPAC-3000 , PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO, HSL-DIO	

描述:

该函数用于获取现场总线上的从站名称。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_slave_name( I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 *MOD_Name);
```

Visual Basic:

```
APS_get_field_bus_slave_name (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long , ByRef MOD_Type As Long);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID : 1~63。在 HSL 中，Module_No 是模块占用的第一个 ID。

对于 MNET 从站模块，取决于从站 ID : 0~63

I32 * MOD_Name: 返回模块名称.

0x000: 未知

0x100: HSL_DI32

0x101: HSL_DO32

0x102: HSL_DI16DO16

0x103: HSL_AO4

0x104: HSL_AI16AO2VV

0x105: HSL_AI16AO2_AV

0x106: HSL_DI16UL

0x107: HSL_DI16RO8

0x108: HSL 4XMO

0x109: HSL_DI16_UCT

0x10A: HSL_DO16_UCT

0x10B: HSL_DI8DO8

0x10C: HSL_DI56DO32_FCN

0x200: MNET_1XMO

0x201: MENT-4XMO

0x202: MENT-4XMO-C

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 MOD_Name;  
ret = APS_get_field_bus_slave_type ( 0, 1, 10, & MOD_Name );  
if( ret != ERR_NoError )  
{  
    // 获取现场总线上的从站名称  
}
```

还可以看看:

APS_get_field_bus_slave_first_axisno	获取从站模块的第一个轴
支持的产品:PCI-8392(H) , DPAC-3000 , PCI(e)-7856, MNET-4XMO-(C), MNET-1XMO, HSL-4XMO	

描述:

该函数用于获取从站模块的第一轴。开始扫描从站模块后，此函数可用于获取分配给从站模块的轴 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_slave_first_axisno ( I32 Board_ID, I32 BUS_No, I32
MOD_No, I32 *AxisNo, I32 *Totalaxes);
```

Visual Basic:

```
APS_get_field_bus_slave_first_axisno (ByVal Board_ID As Long, ByVal BUS_No As Long,
ByVal MOD_No As Long , ByRef AxisNo As Long, ByRef TotalAxes As Long);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID : 1~63。在 HSL 中，Module_No 是模块占用的第一个 ID。

对于 MNET 从站模块，取决于从站 ID : 0~63

I32 *AxisNo: 返回从站模块的第一轴。

I32 *TotalAxes: 返回该模块的总轴

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 AxisID;
I32 Totalaxes;
ret = APS_get_field_bus_slave_first_axisno ( 0, 1, 10, & AxisID,& Totalaxes );
if( ret != ERR_NoError )
{
    获取从模块的第一轴
}
```

还可以看看:

APS_get_field_bus_device_info	获取指定现场总线上的设备（从站）信息
支持的产品:PCI-8392(H) , DPAC-3000 , PCI(e)-7856, MNET-4XMO-(C), HSL-4XMO	

描述:

此函数用于获取指定的设备（从站）信息。该信息包括固件版本，PCB 版本等。请参阅设备信息表。

句法:

C/C++

```
I32 FNTYPE APS_get_field_bus_device_info( I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 Info_No, I32 *Info );
```

Visual Basic:

```
APS_get_field_bus_device_info ( ByVal Board_ID As Long, ByVal BUS_No As Long ,
ByVal MOD_No As Long , ByVal Info_No As Long, Info As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 从站模块编号。

对于 HSL 从站模块，取决于从站 ID : 1~63。在 HSL 中，Module_No 是模块占用的第一个 ID。

对于 MNET 从站模块，取决于从站 ID : 0~63 I32

Info_No: 参考设备信息表。

I32 *Info: 参考设备信息表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
I32 BUS_No = 1;
I32 MOD_No = 0;
I32 ret;
I32 Info;
ret = APS_get_field_bus_device_info (Board_ID, BUS_No, MOD_No , 0x20, &Info );
if( ret != ERR_NoError )
{
    //显示设备信息。
}
```

还可以看看：

APS_get_field_bus_module_info	获取从站信息
-------------------------------	--------

支持的产品:PCIe-833x

描述:

系统启动后，该函数用于获取从设站备的信息。您可以使用此函数获取诸如厂商 ID，产品代码，总的轴编号，IO 编号等信息，

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_module_info(I32 Board_ID, I32 BUS_No, I32 MOD_No,
PEC_MODULE_INFO Module_info );
```

Visual Basic:

```
APS_get_field_bus_module_info (ByVal Board_ID As Long, ByVal BUS_No As Long,
ByVal MOD_No As Long, ByRef Module_info As EC_MODULE_INFO) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。（端口号）仅支持编号0。

I32 MOD_No: 从站编号（从 0 开始）

PEC_MODULE_INFO Module_info : 从站信息的结构。

有关成员参数的详细信息，如下所示：

I32 VendorID: 从站的供应商ID编号。

I32 ProductCode: 从站的产品编号。

I32 RevisionNo: 从站的修订号。

I32 TotalAxisNum: 从站的总轴数。

I32 Axis_ID[64]: 自动从站ID模式的轴ID编号。

I32 Axis_ID_manual[64]: 手动从站ID模式的轴ID编号。

I32 All_ModuleType[32]: 子模块ID按顺序排列。

I32 DI_ModuleNum: 从站中数字输入模块的数量。

I32 DI_ModuleType[32]: 从站中数字输入模块的类型。

I32 DO_ModuleNum : 从站中数字输出模块的数量。

I32 DO_ModuleType [32] : 从站中数字输出模块的类型。

I32 AI_ModuleNum : 从站中模拟输入模块的数量。

I32 AI_ModuleType [32] : 从站中模拟输入模块的类型。

I32 AO_ModuleNum : 从站中模拟输出模块的数量。

I32 AO_ModuleType [32] : 从站中模拟输出模块的类型。

Char Name[128]: 已保留。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
EC_MODULE_INFO Module_info;
ret = APS_get_field_bus_module_info(Board_ID, BUS_No, MOD_No,&Module_info);
if( ret == ERR_NoError )
{
    printf( "Vendor ID is: 0x%x.\n" , Module_info.VendorID);
    printf( "Total axis number is: %d.\n" , Module_info.TotalAxisNum);
}
```

还可以看看：

APS_reset_field_bus_alarm	重置从站的报警信号.
---------------------------	------------

描述:

当伺服驱动器发生告警，并且告警的严重性不是很关键时，您可以通过此函数重置告警信号。

句法:

C/C++:

```
I32 FNTYPE APS_reset_field_bus_alarm( I32 Axis_ID );
```

Visual Basic:

```
APS_reset_field_bus_alarm (ByVal Axis_ID As Long) As Long
```

参数:

I32 Axis_ID: Number of axis.

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Axis_ID = 0;  
ret = APS_reset_field_bus_alarm( Axis_ID );  
if( ret == ERR_NoError )  
{  
    printf( "Reset alarm successful.\n" );  
}
```

还可以看看:

APS_get_field_bus_alarm	获取从站的报警代码
支持的产品:PCIe-833x	

描述:

当伺服驱动器发生告警时，您可以通过调用此函数来获取告警代码，以获取 OD 中的值（错误代码，0x603F）。告警代码的定义取决于伺服驱动器的每个供应商，您可以参考供应商的伺服驱动器手册。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_alarm( I32 Axis_ID, U32 *AlarmCode );
```

Visual Basic:

```
APS_get_field_bus_alarm (ByVal Axis_ID As Long, ByRef AlarmCode As UInteger) As Long
```

参数:

I32 Axis_ID: 轴的编号。

U32 *AlarmCode: 从站返回告警状态。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Axis_ID = 0;
U32 AlarmCode;
ret = APS_get_field_bus_alarm( Axis_ID, &AlarmCode );
if( ret == ERR_NoError )
{
    printf( "Display alarm code= %d\n" , AlarmCode);
}
```

还可以看看:

APS_get_field_bus_pdo	从 PDO 内存中获取值
-----------------------	--------------

支持的产品:PCIe-833x

描述:

这是最低级别的函数，您可以直接从 EtherCAT PDO 存储器获取值并与 EtherCAT 循环时间保持一致。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_pdo( I32 Board_ID, I32 BUS_No, U16 ByteOffset, U16  
Size, U32 *Value );
```

Visual Basic:

```
APS_get_field_bus_pdo (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal  
ByteOffset As Long, ByVal Size As Long, ByRef Value As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

U16 ByteOffset: 特定 PDO 数据的偏移地址，单位为字节。

U16 Size: PDO 数据的大小值，单位为字节。

U32 *Value: 返回 PDO 数据的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
U16 ByteOffset = 16;// PDO 的 OD 偏移为 16 个字节  
U16 Size = 4; //获取 4 个字节的数据  
U32 Value = 0;  
ret=APS_get_field_bus_pdo(Board_ID, BUS_No, ByteOffset, Size, &Value )  
if( ret == ERR_NoError )  
{  
    printf( "Display PDO value= %d\n" , Value);  
}
```

还可以看看:

APS_set_field_bus_pdo	为 PDO 内存赋值
-----------------------	------------

支持的产品:PCIe-833x

描述:

这是最低级别的函数，您可以将其值直接设置到 EtherCAT PDO 存储器中，并与 EtherCAT 循环时间保持一致。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_pdo( I32 Board_ID, I32 BUS_No, U16 ByteOffset, U16  
Size, U32 Value );
```

Visual Basic:

```
APS_set_field_bus_pdo(ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal  
ByteOffset As Long, ByVal Size As Long, ByVal Value As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

U16 ByteOffset: 特定 PDO 数据的偏移地址，单位为字节。

U16 Size: PDO 数据的大小值，单位为字节。

U32 *Value: 返回 PDO 数据的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
U16 ByteOffset = 16;// PDO 的 OD 偏移为 16 个字节  
U16 Size = 4; //获取 4 个字节的数据  
U32 Value = 65535;  
ret=APS_set_field_bus_pdo(Board_ID, BUS_No, ByteOffset, Size, Value )  
if( ret == ERR_NoError )  
{  
    printf( "Set data to PDO value successful\n" );  
}
```

还可以看看:

APS_get_field_bus_pdo_offset	获取 PDO 信息
------------------------------	-----------

支持的产品:PCIe-833x

描述:

这是最低级别的函数，您可以直接从所有 EtherCAT PDO 获取信息，例如数字，数据类型，大小，索引和名称。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_pdo_offset( I32 Board_ID, I32 BUS_No, I32 MOD_No,
PPDO_OFFSET *PPTx, U32 *NumOfTx, PPDO_OFFSET *PPRx, U32 *NumOfRx);
```

Visual Basic:

```
APS_get_field_bus_pdo_offset (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByRef PPTx As IntPtr, ByRef NumOfTx As UInteger, ByRef PPRx As
IntPtr, ByRef NumOfRx As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站编号(从0开始)

PPDO_OFFSET *PPTx: Tx PDO的返回信息

U32 *NumOfTx: 从站PDO Tx的编号

PPDO_OFFSET* PPRx: Rx PDO的返回信息

U32 *NumOfRx: 从站PDO Rx的编号

typedef struct

```
{
U16 数据类型;    : PDO 数据的类型。
U32 ByteSize;   : PDO 数据的大小，单位为字节。
U32 ByteOffset; : 特定 PDO 数据的偏移地址，单位为字节。
U32 索引;       : PDO 对象的索引
U8 NameArr [128];: PDO 对象的名称
} PDO_OFFSET, *PPDO_OFFSET;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
```

```

I32 BUS_No = 0;
I32 MOD_No= 0;
PPDO_OFFSET PPTx;
PPDO_OFFSET PPRx;
U32 Tx_cnt, Rx_cnt;
I32 i;

ret = APS_get_field_bus_pdo_offset(Board_ID, BUS_No, MOD_No, &PPTx, &Tx_cnt,
&PPRx, &Rx_cnt);

if(ret == ERR_NoError)
{
    // 从 PPDO_OFFSET 结构中加载数据
    for(i=0;i++;i<Tx_cnt)
    {
        printf("DataType : %d\n", (PPTx+i)-> DataType)
        printf("ByteSize : %d\n", (PPTx+i)-> ByteSize)
        printf("ByteOffset : %d\n", (PPTx+i)-> ByteOffset)
        printf("Name : %s\n", (PPTx+i)-> NameArr)
    }

    for(i=0;i++;i<Rx_cnt)
    {
        printf("DataType : %d\n", (PPRx+i)-> DataType)
        printf("ByteSize : %d\n", (PPRx+i)-> ByteSize)
        printf("ByteOffset : %d\n", (PPRx+i)-> ByteOffset)
        printf("Name : %s\n", (PPRx+i)-> NameArr)
    }
}

```

还可以看看:

APS_get_field_bus_sdo	从从站中获取 SDO 数据
-----------------------	---------------

描述:

使用此函数可通过 SDO 方法从特定从站获取 OD 数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_sdo( I32 Board_ID, I32 BUS_No, I32 MOD_No, U16  
ODIndex, U16 ODSubIndex, U8 *Data, U32 DataLen, U32 *OutDataLen, U32 Timeout,  
U32 Flags );
```

Visual Basic:

```
APS_get_field_bus_sdo (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal  
MOD_No As Long, ByVal ODIndex As UShort, ByVal ODSubIndex As UShort, ByRef Data  
As Byte, ByRef DataLen As UInteger, ByRef OutDataLen As UInteger, ByVal Timeout As  
UInteger, ByVal Flags As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站编号(从0开始)。

U16 ODIndex : 对象字典的索引。

U16 ODSubIndex : 对象字典的子索引。

U8 *Data : 返回特定OD的数据值。

U32 DataLen : 特定OD的数据长度,单位为字节。

U32 * OutDataLen : 返回特定OD的实际数据长度,单位为字节。

U32 Timeout: 从从设备获取数据的最大等待时间,单位为ms。

U32 Flags: 保留为0。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
I32 MOD_No = 0;  
U16 ODIndex = 0x60fd;  
U16 ODSubIndex = 0;  
U8 Data = 0;
```

```
U32 DataLen = 4;
U32 OutDataLen = 0;
U32 Timeout = 5000;
U32 Flags = 0;
ret= APS_get_field_bus_sdo( Board_ID,
                            BUS_No,
                            MOD_No,
                            ODIndex,
                            ODSubIndex,
                            &Data,
                            DataLen,
                            &OutDataLen,
                            Timeout,
                            Flags
                            );

if( ret == ERR_NoError )
{
    printf( "The OD data value =%d\n" ,Data);
}
```

还可以看看：

APS_set_field_bus_sdo	为从站设置 SDO 数据
-----------------------	--------------

支持的产品:PCIe-833x

描述:

使用此函数可以通过 SDO 方法将 OD 数据设置到特定的从站。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_sdo( I32 Board_ID, I32 BUS_No, I32 MOD_No, U16  
ODIndex, U16 ODSubIndex, U8 *Data, U32 DataLen, U32 Timeout, U32 Flags );
```

Visual Basic:

```
APS_set_field_bus_sdo (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal MOD_No  
As Long, ByVal ODIndex As UShort, ByVal ODSubIndex As UShort, ByRef Data As Byte,  
ByVal DataLen As UInteger, ByVal Timeout As UInteger, ByVal Flags As UInteger) As  
Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站编号(从0开始)。

U16 ODIndex: 对象字典的索引。

U16 ODSubIndex: 对象字典的子索引。

U8 *Data: 特定OD的数据值。

U32 DataLen: 特定OD的数据长度,单位为字节。

U32 Timeout: 从站设备获取数据的最大等待时间,单位为ms。

U32 Flags: 保留为0。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
I32 MOD_No = 0;  
U16 ODIndex = 0x60fe;  
U16 ODSubIndex = 1;  
U8 Data = 256;  
U32 DataLen = 4;  
U32 Timeout = 5000;
```

```
U32 Flags = 0;
ret= APS_set_field_bus_sdo( Board_ID,
                            BUS_No,
                            MOD_No,
                            ODIndex,
                            ODSubIndex,
                            &Data,
                            DataLen,
                            Timeout,
                            Flags
                            );

if( ret == ERR_NoError )
{
    printf( "Set OD data to slave successful.\n" );
}
```

还可以看看:

APS_set_field_bus_od_data	设置 EtherCAT OD 原始数据
支持的产品:PCIe-833x	

描述:

该函数用于通过操作特定的从站设备来设置 PDO 中的 EtherCAT OF 数据。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_od_data( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
SubMOD_No, I32 ODIndex, U32 RawData );
```

Visual Basic:

```
APS_set_field_bus_od_data (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByVal SubMOD_No As Long, ByVal ODIndex As Long, ByVal
RawData As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站的编号ID。

I32 SubMOD_No: 一个从站中的子模块。

I32 ODIndex: EtherCAT OF数据索引。

U32 RawData: EtherCAT OD 数据。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
I32 SubMOD_No = 0;
I32 ODIndex = 0;
U32 ODValue = 2048;
```

```
ret = APS_set_field_bus_od_data(Board_ID, BUS_No, MOD_No, SubMOD_No,ODIndex,
ODValue);
```

还可以看看:

APS_get_field_bus_od_data	获取 EtherCAT OD 原始数据
支持的产品:PCIe-833x	

描述:

该函数用于通过操作特定的从站设备来获取 PDO 中的 EtherCAT OD 数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_od_data( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
SubMOD_No, I32 ODIndex, U32 *RawData );
```

Visual Basic:

```
APS_get_field_bus_od_data (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByVal SubMOD_No As Long, ByVal ODIndex As Long, ByRef
RawData As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站的编号ID。

I32 SubMOD_No: 一个从站中的子模块。

I32 ODIndex: EtherCAT OF数据索引。

U32 *RawData: 返回 EtherCAT OD 数据。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
I32 SubMOD_No = 0;
I32 ODIndex = 0;
U32 RawData ;

ret = APS_get_field_bus_od_data( Board_ID, BUS_No, MOD_No, SubMOD_No,
ODIndex, &RawData );
if( ret == ERR_NoError )
{
```

```
    printf( "OD value is = %d\n" , RawData);  
}
```

还可以看看:

APS_get_field_bus_od_module_info	获取 EtherCAT 从站信息
----------------------------------	------------------

支持的产品:PCIe-833x

描述:

该函数用于获取 EtherCAT 从站信息，如厂商 ID，产品代码和模块 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_od_module_info( I32 Board_ID, I32 BUS_No, I32
MOD_No, PEC_Sub_MODULE_INFO Sub_Module_info );
```

Visual Basic:

```
APS_get_field_bus_od_module_info (ByVal Board_ID As Long, ByVal BUS_No As Long,
 ByVal MOD_No As Long, ByRef Sub_Module_info As EC_Sub_MODULE_INFO) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站的编号ID。

结构EC_Sub_MODULE_INFO的定义如下：

I32 VendorID: 从站的供应商ID编号

I32 ProductCode: 从站设备的产品代码号

I32 RevisionNo: 从站的版本号

I32 TotalSubModuleNum: 从站的最大子模块数

I32 SubModuleID[32]: 子模块的ID编号数组

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
EC_Sub_MODULE_INFO Sub_Module_info;
I32 i = 0;
ret = APS_get_field_bus_od_module_info( Board_ID, BUS_No, MOD_No,
&Sub_Module_info );
if( ret == ERR_NoError )
{
```

```
for ( i = 0 ; i < Sub_Module_info.TotalSubModuleNum ; i++)
{
    if ( Sub_Module_info.SubModuleID[i] != 0 )
        printf( "SubModuleID is = 0x%x\n" ,
Sub_Module_info.SubModuleID[i]);
}
}
```

还可以看看:

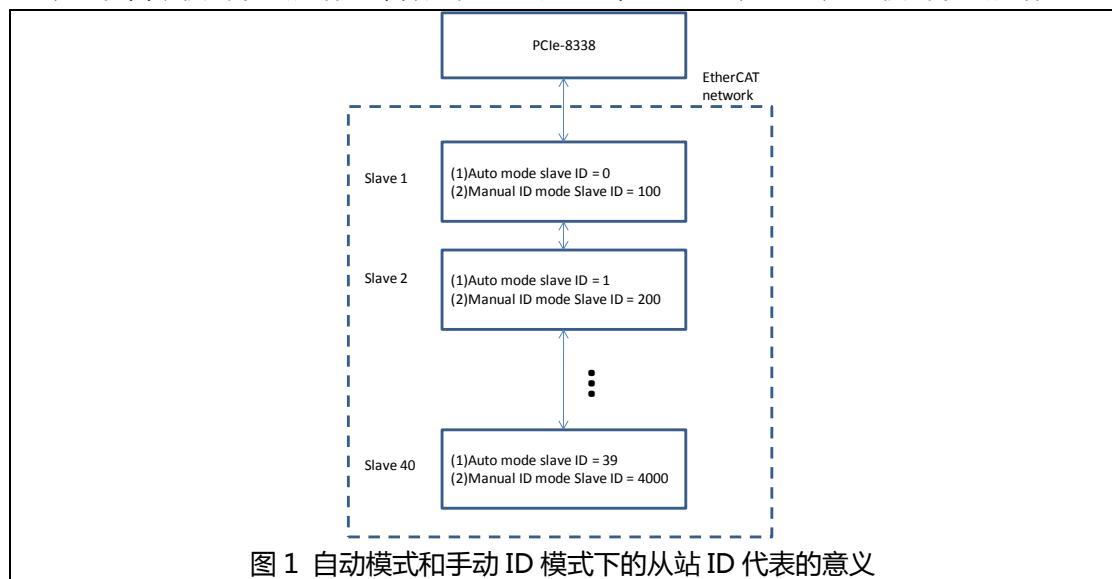
`APS_get_field_bus_module_map`

在手动 ID 模式下获取映射的从站 ID

支持的产品:PCIe-833x

描述:

使用手动 ID 模式时，此函数用于获取映射的从站设备 ID 总数。图 1 说明自动模式和手动 ID 模式下从站 ID 表示的示例。首先，用户可以使用 `APS_get_field_bus_last_scan_info()` 来获取 EtherCAT 网络中现在有多少个从站。这里假设使用了 40 个从站。其次，用户将通过此函数获得实际的映射的从站 ID 数组。在此数组中，数组索引表示自动模式下的从站 ID，而数组值表示手动模式下的从站 ID。例如，如果用户数组 MOD_No_Arr 通过此函数获得 MOD_No_Arr [0] = 100, MOD_No_Arr [1] = 200, ... 和 MOD_No_Arr [39] = 4000，则显示数组索引 0、1、... 39 表示在自动模式下的从站 ID，数组值 100、200、... 4000 表示手动 ID 模式下的从站 ID。



句法:

C/C++:

```
I32 APS_get_field_bus_module_map(I32 Board_ID, I32 BUS_No, U32 *MOD_No_Arr, U32 Size );
```

Visual Basic:

```
APS_get_field_bus_module_map(ByVal Board_ID As Long, ByVal BUS_No As Long, MOD_No_Arr As Int, ByVal Size As Int);
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用`APS_initial()`来检索它。

I32 BUS_No: 现场总线编号。（端口号）仅支持编号 0。

U32* MOD_No_Arr: 手动 ID 模式下映射的从站 ID 数组。

U32 Size: EtherCAT 网络中存在的从站的总数

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
U32 * MOD_No_Arr = NULL;
U32 Size = 0;
//如果从站总数为 5 , 则 Size = 5 ;
MOD_No_Arr = (U32 *) malloc( sizeof(U32) * Size );
ret = APS_get_field_bus_module_map ( Board_ID, BUS_No, MOD_No_Arr, Size );
```

还可以看看:

[APS_get_field_bus_last_scan_info\(\)](#)

APS_set_field_bus_module_map

在手动 ID 模式下设置映射的从站 ID

支持的产品:PCIe-833x

描述:

此函数用于在手动从站 ID 模式下获取映射的从站 ID。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_module_map ( I32 Board_ID, I32 BUS_No, U32*  
MOD_No_Arr, U32 Size);
```

Visual Basic:

```
APS_set_field_bus_module_map (ByVal Board_ID As Long, ByVal Bus_No As Long,  
ByVal MOD_No_Arr() As UInteger, ByVal Size As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

U32 *MOD_No_Arr: 手动ID模式下映射的从站ID数组。

U32 Size: 现场总线网络中存在的从站的总数

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
U32 * MOD_No_Arr = NULL;  
U32 Size = 0;  
//如果从站总数为 5 , 则 Size = 5 ;  
MOD_No_Arr = (U32 *) malloc( sizeof(U32) * Size );  
MOD_No_Arr [0] = 111; //拓扑手册 ID 的第一个从站为 111  
MOD_No_Arr [1] = 222; //拓扑手册 ID 的第二个从站是 222  
MOD_No_Arr [2] = 333; //拓扑手册 ID 的第三个从站为 333  
MOD_No_Arr [3] = 444; //拓扑手册 ID 的第四个从站为 444  
MOD_No_Arr [4] = 555; //拓扑手册 ID 的第五个从站是 555  
ret = APS_set_field_bus_module_map ( Board_ID, BUS_No, MOD_No_Arr, Size );
```

还可以看看:

APS_get_field_bus_slave_state	获取从站状态机器的状态
-------------------------------	-------------

支持的产品:PCIe-833x

描述:

该函数用于获取从站状态机器的状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_slave_state (I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 *State);
```

Visual Basic:

```
APS_get_field_bus_slave_state (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No As Long, ByRef State As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。(端口号)仅支持编号0。

I32 MOD_No: 从站编号ID。

I32 *State: 从站状态机器的状态。

值	状态定义
3	EC_STATE_INIT
4	EC_STATE_PREOP
5	EC_STATE_SAFEOP
6	EC_STATE_OP

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
I32 State = 0;

ret = APS_get_field_bus_slave_state (Board_ID,BUS_No,MOD_No, &State);
// 请参考状态表的APS_set_field_bus_slave_state。
```

还可以看看:

[APS_set_field_bus_slave_state\(\)](#)

APS_set_field_bus_slave_state	设置从站状态机器的状态
-------------------------------	-------------

支持的产品:PCIe-833x

描述:

该函数用于设置从站状态机器的状态.

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_slave_state( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32 State);
```

Visual Basic:

```
APS_set_field_bus_slave_state (ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal MOD_No As Long, ByVal State As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号。 (端口号) 仅支持编号0。

I32 MOD_No: 从站编号ID。

I32 State: 从站状态机器的状态。

值	状态定义
3	EC_STATE_INIT
4	EC_STATE_PREOP
5	EC_STATE_SAFEOP
6	EC_STATE_OP

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Board_ID = 0;
I32 BUS_No = 0;
I32 MOD_No = 0;
I32 State = 0 ;
State = 5 ; //安全操作模式
ret = APS_set_field_bus_slave_state ( Board_ID, BUS_No, MOD_No, State);
```

还可以看看:

APS_get_field_bus_ESC_register	获取 EtherCAT 从站控制器寄存器
--------------------------------	----------------------

描述:

此函数用于获取 EtherCAT 从控制器 (ESC) 寄存器。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_ESC_register( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 RegOffset, I32 DataSize, I32 *DataValue );
```

Visual Basic:

```
APS_get_field_bus_ESC_register (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal RegOffset As Long, ByVal DataSize As Long, ByRef  
DataValue As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 仅支持数字0的现场总线编号 (端口号) 。

I32 MOD_No: 从站编号ID。

I32 RegOffset: ESC寄存器的地址偏移量。

I32 DataSize: ESC寄存器的长度 , 单位为字节。 其范围应在1到8个字节之间。

I32 *DataValue: 获取ESC数据缓冲区。如果范围超过4个字节 , 则需要二维I32数组进行操作。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
I32 MOD_No = 0;  
I32 RegOffset = 0x920;  
I32 DataSize = 8;  
I32 GetDataValue[2];
```

```
ret = APS_get_field_bus_ESC_register (Board_ID, BUS_No, MOD_No, RegOffset,  
DataSize, &GetDataValue);
```

还可以看看:

`APS_set_field_bus_ESC_register()`

APS_set_field_bus_ESC_register	设置 EtherCAT 从站控制器寄存器
--------------------------------	----------------------

描述:

此函数用于设置 EtherCAT 从控制器 (ESC) 寄存器。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_ESC_Register( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 RegOffset, I32 DataSize, I32 *DataValue );
```

Visual Basic:

```
APS_set_field_bus_ESC_register (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal RegOffset As Long, ByVal DataSize As Long, ByRef  
DataValue As UInteger) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 仅支持数字0的现场总线编号 (端口号) 。

I32 MOD_No: 从站编号ID。

I32 RegOffset: ESC寄存器的地址偏移量。

I32 DataSize: ESC寄存器的长度 , 单位为字节。 范围应在1到8个字节之间。

I32 *DataValue: 获取ESC数据缓冲区。如果范围超过4个字节 , 则需要二维I32数组进行操作。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 Board_ID = 0;  
I32 BUS_No = 0;  
I32 MOD_No = 0;  
I32 RegOffset = 0x300;  
I32 DataSize = 1;  
I32 DataValue = 0;  
  
ret = APS_set_field_bus_ESC_register (Board_ID, BUS_No, MOD_No, RegOffset,  
DataSize, &DataValue);
```

还可以看看:

`APS_get_field_bus_ESC_register ()`

APS_get_system_loading	获取系统循环加载
支持的产品:PCIe-833x	

描述:

该函数用于获取系统循环加载.

句法:

C/C++:

```
I32 FNTYPE APS_get_system_loading(I32 Board_ID, F64* Loading1, F64* Loading2, F64*
Loading3, F64* Loading4);
```

Visual Basic:

```
APS_get_system_loading (ByVal Board_ID As Long, ByRef Loading1 As Double, ByRef
Loading2 As Double, ByRef Loading3 As Double, ByRef Loading4 As Double) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

F64* Loading1: 计算 PCIe-8334/8 运动回路的预定加载时间，单位为%。

F64* Loading2: 计算 PCIe-8334/8 EtherCAT 回路的预定加载时间，单位为%。

F64* Loading3: 保留。

F64* Loading4: 保留。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
F64 motion_loading;
F64 ECAT_loading;
F64 no_data1, no_data2;
```

```
ret = APS_get_system_loading (Board_ID, &motion_loading, &ECAT_loading,
&no_data1, &no_data2);
```

还可以看看:

APS_get_field_bus_analysis_topology	获取当前和过去的拓扑结构，然后进行分析。
-------------------------------------	----------------------

支持的产品:PCIe-833x

描述:

当 APS_start_filed_bus API 重新运行-4013 或-4043 错误时，此函数用于分析当前和过去的从站拓扑。分析条件包括供应商 ID，产品代码，版本号和带从站的子模块 ID。如果拓扑在扫描文件总线和启动现场总线进程之间来回混乱，则 API 将返回错误的从站编号以供用户参考。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_analysis_topology( I32 Board_ID, I32 BUS_No, I32  
*Error_Slave_No, PEC_MODULE_INFO Current_slave_info, I32  
*Current_slave_num, PEC_MODULE_INFO Past_slave_info, I32 * Past_slave_num);
```

Visual Basic:

```
APS_get_field_bus_analysis_topology (ByVal Board_ID As Long, ByVal Bus_No As Long,  
ByRef Error_Slave_No As Long, ByRef Current_slave_info As EC_Sub_MODULE_INFO,  
ByRef Current_slave_num As Long, ByRef Past_slave_info As EC_Sub_MODULE_INFO,  
ByRef Past_slave_num As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 仅支持数字 0 的现场总线编号 (端口号) 。

I32* Error_Slave_No: 自动从站 ID 模式下的错误从站编号。如果 Error_Slave_No = 0，则没有错误，过去的从站和当前的从站是相同的。如果返回 1，则当前从站 1 与过去的从站 1 不同。如果返回 5，则当前从站 5 与过去的从站 5 不同。

PEC_MODULE_INFO Current_slave_info :具有当前拓扑的从站信息的结构。

有关成员参数的详细信息，如下所示：

- I32 VendorID: 从站的供应商ID编号。
- I32 ProductCode: 从站的产品代码。
- I32 RevisionNo: 从站的修订号。
- I32 TotalAxisNum: 预留。
- I32 Axis_ID[64]: 预留。
- I32 Axis_ID_manual[64]: 预留。
- I32 All_ModuleType[32]: 子模块ID按顺序排列。
- I32 DI_ModuleNum: 预留。
- I32 DI_ModuleType[32]: 预留。
- I32 DO_ModuleNum: 预留。
- I32 DO_ModuleType[32]: 预留。
- I32 AI_ModuleNum: 预留。
- I32 AI_ModuleType[32]: 预留。

I32 AO_ModuleNum: 预留。
I32 AO_ModuleType[32]: 预留。.
Char Name[128]: 预留。

I32 *Current_slave_num : 当前拓扑的从站数量。
PEC_MODULE_INFO Past_slave_info : 具有过去拓扑的从站信息的结构。
I32 * Past_slave_num : 过去拓扑的从站数量。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardID = 0;  
I32 BusNo = 0 ;  
I32 Error_Slave_No = 0;  
I32 Current_slave_num =0 ;  
I32 Past_slave_num =0;  
EC_MODULE_INFO Current_slave_info[64] = {0};  
EC_MODULE_INFO Past_slave_info[64] = {0};  
  
ret = APS_get_field_bus_analysis_topology( BoardID,BusNo,  
&Error_Slave_No,Current_slave_info,&Current_slave_num,Past_slave_info,&Past_slave_  
num);
```

还可以看看:

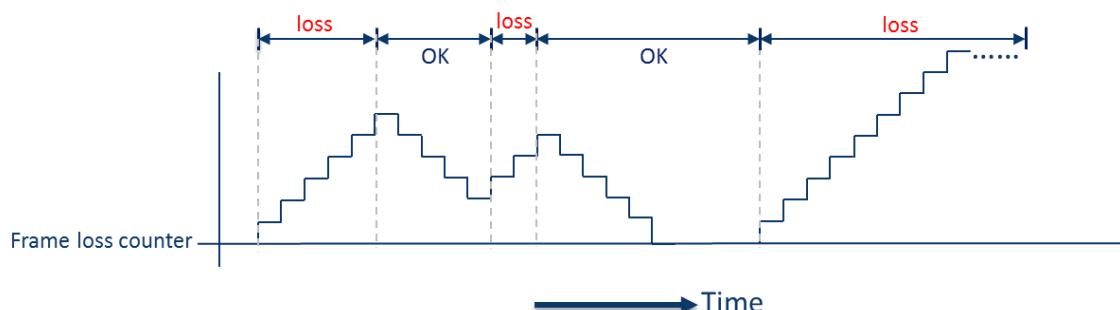
APS_get_field_bus_loss_package

在接收总线方向上获取 EtherCAT 帧计数的丢失

支持的产品:PCIe-833x

描述:

该函数用于在接收总线方向上获取 EtherCAT 帧计数的丢失信息。计数行为如下所示：



如果系统检测到帧丢失，则每个 EtherCAT 回路定时帧丢失计数将增加 1。当系统正常获取帧时，帧丢失计数将减少到零。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_loss_package (I32 Board_ID, I32 BUS_No,I32 *Loss_Count);
```

Visual Basic:

```
APS_get_field_bus_loss_package (ByVal Board_ID As Integer, ByVal BUS_No As Integer,  
ByRef Loss_Count As Integer) As Integer
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 仅支持数字0的现场总线编号（端口号）。

I32 *Loss_Count: 包丢失的计数值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 BoardID = 0;  
I32 BusNo = 0;  
I32 lossCount = 0;
```

```
ret = APS_get_field_bus_loss_package( Board_ID, BusNo, & lossCount );  
// 当丢失包数大于 10 时，停止 X , Y 和 Z 的运动。
```

```
if( lossCount >= 10 )  
{  
    ret = APS_emg_stop( X );  
    ret = APS_emg_stop( Y );  
    ret = APS_emg_stop( Z );  
}
```

还可以看看:

17. 齿轮/龙门函数

APS_set_gantry_param	Set gantry function related parameter / 启用/禁用一个指定的齿轮模式
----------------------	--

支持的产品:PCI-8253/56, PCI-8392(H)

描述:

此函数用于将参数设置给指定的龙门组。

参数编号和对应的参数数据，请参考龙门参数表。

句法:

C/C++:

```
I32 FNTYPE APS_set_gantry_param( I32 Board_ID, I32 GroupNum, I32 ParaNum, I32  
ParaDat );
```

Visual Basic:

```
APS_set_gantry_param( ByVal Board_ID As Long, ByVal GroupNum As Long, ByVal  
ParaNum As Long, I32 ParaDat As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 GroupNum: 指定一个龙门组编号。

I32 ParaNum: 参数编号。请参考龙门参数表。

I32 ParaDat: 参数数据。请参考龙门参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。
```

```
I32 boardId = 0;
```

还可以看看:

[APS_get_gantry_param\(\)](#); [APS_set_gantry_axis\(\)](#); [APS_get_gantry_axis\(\)](#)

APS_get_gantry_param	获取龙门函数相关参数
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于从指定的龙门组中获取参数。

参数编号和对应的参数数据，请参考龙门参数表。

句法:

C/C++:

```
I32 FNTYPE APS_get_gantry_param( I32 Board_ID, I32 GroupNum, I32 ParaNum, I32
*ParaDat );
```

Visual Basic:

```
APS_get_gantry_param( ByVal Board_ID As Long, ByVal GroupNum As Long, ByVal
ParaNum As Long, ParaDat As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 GroupNum: 指定一个龙门组编号。

I32 ParaNum: 指定一个参数编号。请参考龙门参数表。

I32 *ParaDat: 返回一个参数数据。请参考龙门参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。
```

```
I32 boardId = 0;
```

还可以看看:

[APS_set_gantry_param\(\)](#); [APS_set_gantry_axis\(\)](#); [APS_get_gantry_axis\(\)](#)

APS_set_gantry_axis	在龙门组中设置两个轴
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

此函数用于将两个轴指定为龙门组。启用该组的龙门模式后，这两个轴将具有龙门行为。启用龙门模式后，您将无法更改龙门轴的设置。

句法:

C/C++:

```
I32 FNTYPE APS_set_gantry_axis( I32 Board_ID, I32 GroupNum, I32 Master_Axis_ID, I32 Slave_Axis_ID );
```

Visual Basic:

```
APS_set_gantry_axis(ByValBoard_ID As Long, ByVal GroupNum As Long, ByVal Master_Axis_ID As Long, ByVal Slave_Axis_ID As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 GroupNum: 指定一个龙门组编号。最大组数号请参考规格。

I32 Master_Axis_ID: 指定一个轴 ID 为龙门的主轴。

I32 Slave_Axis_ID: 指定一个轴 ID 为龙门的从轴。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。
```

```
I32 boardId = 0;
```

```
I32 GroupNum = 0;
```

```
I32 Master_Axis_ID = 0, Slave_Axis_ID = 1;
```

//在设置龙门轴之前，必须禁用龙门模式。

```
Ret = APS_set_gantry_axis(Board_ID, GroupNum, Master_Axis_ID, Slave_Axis_ID );
```

```
if( ret != ERR_NoError )
```

//...检查错误代码。

```
Ret = APS_get_gantry_axis(Board_ID, GroupNum, &Master_Axis_ID, &Slave_Axis_ID );
```

```
if( ret != ERR_NoError )
```

//...检查错误代码。

还可以看看:

`APS_get_gantry_axis(); APS_set_gantry_param(); APS_get_gantry_param()`

APS_get_gantry_axis	获取龙门组中的轴
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于获取指定龙门组中的龙门主轴 ID 和从轴 ID。

句法:

C/C++:

```
I32 FNTYPE APS_get_gantry_axis( I32 Board_ID, I32 GroupNum, I32 *Master_Axis_ID,
I32 *Slave_Axis_ID );
```

Visual Basic:

```
APS_get_gantry_axis(ByVal Board_ID As Long, ByVal GroupNum As Long,
Master_Axis_ID As Long, Slave_Axis_ID As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 GroupNum: 指定一个龙门组编号。

I32 *Master_Axis_ID: 返回指定龙门组中的主轴 ID。

I32 *Slave_Axis_ID: 返回指定龙门组中的从轴 ID。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。
```

```
I32 boardId = 0;
```

```
I32 GroupNum = 0;
```

```
I32 Master_Axis_ID = 0, Slave_Axis_ID = 1;
```

//在设置龙门轴之前，必须禁用龙门模式。

```
Ret = APS_set_gantry_axis(Board_ID, GroupNum, Master_Axis_ID, Slave_Axis_ID );
```

```
if( ret != ERR_NoError )
```

//...检查错误代码。

```
Ret = APS_get_gantry_axis(Board_ID, GroupNum, &Master_Axis_ID, &Slave_Axis_ID );
```

```
if( ret != ERR_NoError )
```

//...检查错误代码。

还可以看看:

[APS_set_gantry_axis\(\)](#); [APS_set_gantry_param\(\)](#); [APS_get_gantry_param\(\)](#)

APS_get_gantry_error	获取龙门轴偏差误差
支持的产品:PCI-8253/56, PCI-8392(H)	

描述:

该函数用于获取龙门轴偏差误差。

偏差误差=主轴反馈位置 – 从轴反馈位置。

句法:

C/C++:

```
I32 FNTYPE APS_get_gantry_error( I32 Board_ID, I32 GroupNum, I32 *GentryError );
```

Visual Basic:

```
APS_get_gantry_error (ByVal Board_ID As Long, ByVal GroupNum As Long, GentryError  
As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 GroupNum: 指定一个龙门组编号。

I32 *GentryError: 返回龙门轴偏差误差。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret; //返回错误代码。
```

```
I32 boardId = 0;
```

```
I32 GroupNum = 0;
```

```
I32 GentryError;
```

```
ret = APS_get_gantry_error(boardId, GroupNum, &GentryError );
```

```
if( ret == ERR_NoError)
```

```
    // 显示龙门错误
```

还可以看看:

[APS_set_gantry_axis\(\)](#); [APS_set_gantry_param\(\)](#); [APS_get_gantry_param\(\)](#)

APS_get_encoder	获取编码器
支持的产品:PCI-8253/56	

描述:

此函数用于获取一个轴的编码器计数器。计数器以脉冲为单位。一般来说，它用于补偿龙门架归零。

句法:

C/C++:

```
I32 FNTYPE APS_get_encoder( I32 Axis_ID, I32 *Encoder );
```

Visual Basic:

```
APS_get_encoder(ByVal Axis_ID As Long, Encoder As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Encoder: 编码器计数器。以脉冲为单位。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Encoder;  
APS_get_encoder(Axis_ID, &Encoder ); //获取编码器计数器  
...//
```

还可以看看:

[APS_get_latch_event\(\)](#); [APS_get_latch_counter\(\)](#)

APS_get_latch_event	按轴获取锁存事件
支持的产品:PCI-8253/56	

描述:

此函数用于获取锁存事件。 有两个信号源，包括 Ez 和 Org 信号锁存器。 如果发生锁存，则事件打开。用户可以通过调用 APS_get_latch_counter()清除锁存事件。
一般来说，它用于补偿龙门架归零。

句法:

C/C++:

```
I32 FNTYPE APS_get_latch_event( I32 Axis_ID, I32 Src, I32 *Event );
```

Visual Basic:

```
APS_get_latch_event(ByVal Axis_ID As Long, ByVal Src As Long, Event As Long) As  
Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Src: 指定一个锁存源

0: Ez 锁存, 1: Org 锁存.

I32 *Event: 锁存事件

0 : 未发生任何锁存。 1 : A 锁存发生。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Event, latchCounter;
```

```
I32 SrcOrg = 1; //指定 Org.
```

```
APS_get_latch_event(Axis_ID, SrcOrg, &Event); //获取 ORG 锁存事件
```

```
If( Event == 1 ) // ORG 被锁存
```

```
{ //重置锁存事件和读取锁存计数器
```

```
    APS_get_latch_counter(Axis_ID, SrcOrg, &latchCounter);
```

```
}
```

还可以看看:

[APS_get_latch_counter\(\)](#); [APS_get_encoder\(\)](#)

APS_get_latch_counter	按轴获取锁存计数器
支持的产品:PCI-8253/56	

描述:

此函数用于获取锁存计数器。有两个信号源，包括 Ez 和 Org 信号锁存器。如果发生锁存，事件将打开并且编码器计数器将被锁存。用户可以通过调用此功能获取锁存计数器并重置（关闭）事件。

一般来说，它用于补偿龙门架归零。

句法:

C/C++:

```
I32 FNTYPE APS_get_latch_counter( I32 Axis_ID, I32 Src, I32 *Counter );
```

Visual Basic:

```
APS_get_latch_counter( ByVal Axis_ID As Long, ByVal Src As Long, Counter As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Src: 指定一个锁存源

0: Ez 锁存, 1: Org 锁存.

I32 *Counter: 锁存计数器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Event, latchCounter;
```

```
I32 SrcOrg = 1; //指定 Org
```

```
APS_get_latch_event(Axis_ID, SrcOrg, &Event); //获取 ORG 锁存事件
```

```
If( Event == 1 ) // ORG 被锁存
```

```
{
```

```
//重置锁存事件和读取锁存计数器
```

```
APS_get_latch_counter(Axis_ID, SrcOrg, &latchCounter);
```

```
}
```

还可以看看:

[APS_set_latch_event\(\)](#); [APS_get_encoder\(\)](#)

APS_start_gear	启用/禁用一个指定的齿轮模式
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

此函数用于启用指定的齿轮模式。两种齿轮模式，包括标准和龙门，可用于特定应用。

句法:

C/C++:

```
I32 FNTYPE APS_start_gear (I32 Axis_ID, I32 Mode);
```

Visual Basic:

```
APS_start_gear (ByVal Axis_ID As Long, ByVal Mode As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Mode: 齿轮模式。

0 : 禁用； 1 : 标准模式； 2 : 龙门模式。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
...//  
APS_start_gear(Axis_ID, 0); //禁用齿轮。  
...//  
APS_start_gear(Axis_ID, 1); //启用标准齿轮模式。
```

还可以看看:

[APS_get_gear_status\(\)](#)

APS_get_gear_status	获取齿轮状态
支持的产品:PCI-8254/58 / AMP-204/8C, PCIe-833x	

描述:

该函数用于获取齿轮应用的状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_gear_status( I32 Axis_ID, I32 *Status );
```

Visual Basic:

```
APS_get_gear_status(ByVal Axis_ID As Long, Status As Long) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Status: 齿轮状态。

0 : 处于禁用状态。

1 : 处于标准模式的启用状态。

2 : 处于龙门模式的启用状态。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Status;
```

```
APS_get_gear_status(Axis_ID, &Status ); //获取齿轮状态
```

```
...//
```

还可以看看:

[APS_start_gear\(\)](#)

APS_get_gantry_number	获取主站对应的从站数量
-----------------------	-------------

支持的产品:PCIe-833x

描述:

该函数用于在龙门模式下获取该主站对应的从站总数。用户需要使用轴参数 PRA_EGEAR_MASTER (0x65) 和 PRA_EGEAR_SOURCE (0x66) 和 APS_set_axis_param() 函数来指定主站和对应的从站，并使用 APS_start_gear() 启用龙门模式。然后，用户可以使用这些函数 APS_get_gantry_number() 来获取从站的总数，并可以使用 APS_get_gantry_info() 来获取从轴 ID 数组。

句法:

C/C++:

```
I32 APS_get_gantry_number(I32 MasterAxisID, I32 *SlaveAxisIDSize );
```

Visual Basic:

```
APS_get_gantry_number(ByVal MasterAxisID As Long, SlaveAxisIDSize As Long) As Long
```

参数:

I32 MasterAxisID: 主轴 ID ; 轴 ID 为 0 到 65535。

I32* SlaveAxisIDSize: 该主站对应的从站总数。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_get_gantry_info\(\)](#)

APS_get_gantry_info	获取从轴 ID 数组
支持的产品:PCIe-833x	

描述:

此函数用于在龙门模式下获取从轴 ID 数组。有关详细信息，请参考 APS_get_gantry_number() 的描述。

句法:

C/C++:

```
I32 APS_get_gantry_info (I32 MasterAxisID, I32 SlaveAxisIDSize, I32 *SlaveAxisIDArray );
```

Visual Basic:

```
APS_get_gantry_info (ByVal MasterAxisID As Long, ByVal SlaveAxisIDSize As Long,
SlaveAxisIDArray As Long ) As Long
```

参数:

I32 MasterAxisID: 主轴 ID ; 轴 ID 为 0 到 65535。

I32 SlaveAxisIDSize : 从站总数

I32 * SlaveAxisIDArray : 从轴 ID 数组

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_get_gantry_number\(\)](#)

APS_get_gantry_deviation	获取主从站之间的位置偏差
--------------------------	--------------

描述:

此函数用于获取主从站之间的位置偏差。此函数在 ASYNC 模式下实现。

句法:

C/C++:

```
I32 APS_get_gantry_deviation (I32 MasterAxisID, I32 SlaveAxisIDSize, I32
*SlaveAxisIDArray, F64 *DeviationArray );
```

Visual Basic:

```
APS_get_gantry_deviation (ByVal MasterAxisID As Long, ByVal SlaveAxisIDSize As Long,
SlaveAxisIDArray As Long, DeviationArray As Long ) As Long
```

参数:

I32 MasterAxisID: 主轴 ID ; 轴 ID 为 0 到 65535。

I32 SlaveAxisIDSize: 从站的总数量。

I32* SlaveAxisIDArray: 从轴 ID 数组

F64* DeviationArray: 主站和从站之间的位置偏差数组

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

18. 比较触发器

APS_set_trigger_param	设置比较触发器相关参数
-----------------------	-------------

支持的产品:PCI-8253/56, PCI-C154(+), PCI-8154/8158(DB-8150), EMX-100, PCI-8254/58 / AMP-204/8C

描述:

此函数用于设置与触发器相关的比较参数。触发器参数表中描述了触发器参数的所有定义。您还可以使用 “APS_get_trigger_param()” 函数获得参数设置。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_param( I32 Board_ID, I32 Param_No, I32 Param_Val );
```

Visual Basic:

```
APS_set_trigger_param(ByVal Board_ID As Long, ByVal Param_No As Long, ByVal  
Param_Val As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Param_No: 参数编号。参考触发器参数表

I32 Param_Val: 参数值。参考触发器参数表

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

参考 “APS_set_trigger_linear” , “APS_set_trigger_table” 示例

示例 2:

以下示例适用于 EMX-100

```
I32 BoardId = 0;
```

```
APS_set_trigger_param(BoardId, TGR0_CMP_ENC, 0); // 设置轴 0 以比较命令位置
```

示例 3:

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 BoardId = 0;
```

```
APS_set_trigger_param(BoardId, 0x0, 0); //设置线性比较源
```

还可以看看:

[APS_get_trigger_param\(\)](#)

APS_get_trigger_param	获取比较触发器相关参数
支持的产品:PCI-8253/56,PCI-C154(+), PCI-8154/8158(DB-8150), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取比较触发器相关的参数。触发器参数表中描述了触发器参数的所有定义。
您也可以使用“APS_set_trigger_param()”函数设置参数。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_param( I32 Board_ID, I32 Param_No, I32 *Param_Val );
```

Visual Basic:

```
APS_get_trigger_param(ByVal Board_ID As Long, ByVal Param_No As Long, Param_Val  
As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Param_No: 参数编号。参考触发器参数表。

I32 Param_Val: 返回参数值。参考触发器参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 EMX-100

```
I32 BoardId = 0;  
I32 Param_Val = 0;  
APS_get_trigger_param(BoardId, TGR0_CMP_ENC, &Param_Val );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 BoardId = 0;  
I32 Param_Val = 0;  
APS_get_trigger_param(BoardId, 0x0, &Param_Val ); //获取线性比较源。
```

还可以看看:

[APS_set_trigger_param\(\)](#)

APS_set_trigger_linear	设置线性比较函数
------------------------	----------

支持的产品:PCI-8253/56,PCI-C154(+), PCI-8154/8158(DB-8150) , PCI-8254/58 /

AMP-204/8C

描述:

该函数用于设置线性比较函数。

线性触发操作完成后，总的比较点将为：

总的比较点数=重复时间。 (以 StartPoint 作为第一个触发点)

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_linear( I32 Board_ID, I32 LCmpCh, I32 StartPoint, I32
RepeatTimes, I32 Interval );
```

Visual Basic:

```
APS_set_trigger_linear(ByVal Board_ID As Long, ByVal LCmpCh As Long, ByVal
StartPoint As Long, ByVal RepeatTimes As Long, ByVal Interval As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 LCmpCh: 线性比较设置通道。从零开始。

对于 PCI-8254/58 / AMP-204/8C , I32 LCmpCh : 线性比较集通道。从 0 开始。 范围是 0 到 3。

I32 StartPoint: 启动线性触发点。

I32 RepeatTimes: 触发重复次数。

I32 Interval: 触发间隔。

对于 PCI-8253/56 , 时间间隔：24 位无符号值。

对于 PCI-8254/58 / AMP-204/8C , I32 间隔：触发间隔。 (-16777215~16777215 , 单位是脉冲)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
APS_set_trigger_param(BoardId, 0x0, 0 ); //设置线性比较源
APS_set_trigger_param(BoardId, 0x10, 0 ); //将 LCMP0 设置为 TRG0 的源
APS_set_trigger_linear(BoardId, 0, 100, 49999, 10 ); //设置 LCMP0 线性比较算法。
// 起点= 100 , 重复时间= 49999 , 间隔= 10。
APS_set_trigger_param(BoardId, 0x04, 1 ); //Enable LCMP0
// 触发操作。
```

```
APS_set_trigger_param( 0, 0x04, 0 ); //禁用 LCMP0
```

还可以看看:

```
APS_set_trigger_table()
```

APS_set_trigger_table	设置表比较函数
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于配置指定的比较表。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_table( I32 Board_ID, I32 TCmpCh, I32 *DataArr, I32
ArraySize );
```

Visual Basic:

```
APS_set_trigger_table( ByVal Board_ID As Long, ByVal TCmpCh As Long, DataArr As
Long, ByVal ArraySize As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。

对于 PCI-8253/56, 有两个比较表

对于 PCI-8254/58 / AMP-204/8C, I32 TCmpCh: 指定比较表编号。从零开始，范围是 0 到 3。

I32 *DataArr: 比较数据数组。

I32 ArraySize : 比较数据数组的大小。请参考产品的规格。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#define POINTS 1000
I32 ret;
I32 data[POINTS];
I32 i;
for( i = 0; i < POINTS; i++ )
    data[i] = 10 + (C) * 10;
```

APS_set_trigger_param(BoardId, 0x2, 0); //将编码器计数器 0 设置为 TCMP0 的源。

APS_set_trigger_param(BoardId, 0x10, 4); //将 TCMP0 设置为 TRG0 的来源。

ret = APS_set_trigger_table(0, 0, data, POINTS);

APS_set_trigger_param(BoardId, 0x06, 1); //启用 TCMP0

// 触发操作...

//当完成触发操作时。

```
APS_set_trigger_param(BoardId, 0x06, 0 ); //启用 TCMP0
```

还可以看看：

```
APS_set_trigger_linear()
```

APS_set_trigger_manual	Manual output trigger
------------------------	-----------------------

支持的产品:PCI-8253/56,PCI-C154(+), PCI-8154/8158(DB-8150) , PCI-8254/58 /

AMP-204/8C

描述:

此函数用于在指定的触发输出通道上强制输出触发。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_manual( I32 Board_ID, I32 TrgCh );
```

Visual Basic:

```
APS_set_trigger_manual( ByVal Board_ID As Long, ByVal TrgCh As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TrgCh: 触发输出通道 (TRG) 编号。从零开始。

对于 PCI-8254/58 / AMP-204/8C, I32 TrgCh: 触发输出通道 (TRG) 编号。从零开始。范围是 0 到 3。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;  
I32 ret;  
ret = APS_set_trigger_manual( Board_ID, 1); //TRG1
```

还可以看看:

[APS_set_trigger_manual_s\(\)](#)

APS_set_trigger_manual_s	手动同步输出触发
支持的产品:PCI-8253/56/58A/PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

此函数用于强制输出触发脉冲。它旨在同步和手动输出一个或多个触发通道。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_manual_s( I32 Board_ID, I32 TrgChInBit );
```

Visual Basic:

```
APS_set_trigger_manual_s( ByValBoard_ID As Long, ByValTrgChInBit As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
ret = APS_set_trigger_manual_s( 0, 0xF ); //4 通道同时输出触发。
Ret = APS_set_trigger_manual_s( 0, 0x2 ); //TRG1 输出触发。
Ret = APS_set_trigger_manual_s( 0, 0x3 ); // TRG0 和 TRG1 同步输出触发。
//...
```

还可以看看:

[APS_set_trigger_manual\(\)](#)

APS_get_trigger_table_cmp	获取当前表比较值
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于在指定的表比较器中获取当前比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_table_cmp( I32 Board_ID, I32 TCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_trigger_table_cmp(ByVal Board_ID As Long, ByVal TCmpCh As Long, CmpVal  
As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定表比较器的通道编号。从零开始。

对于 PCI-8254/58 / AMP-204/8C, I32 TCmpCh: 指定表比较器的通道编号。从零开始。范围从 0 到 3。

I32 *CmpVal: 返回比较器中当前的比较值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 CmpVal;  
ret = APS_get_trigger_table_cmp ( 0, 0, &CmpVal );  
If( ret != ERR_NoError )  
{ // 错误 , 显示消息。  
}
```

还可以看看:

[APS_get_trigger_linear_cmp\(\)](#)

<code>APS_get_trigger_linear_cmp</code>	获取当前的线性比较值
---	------------

支持的产品:PCI-8253/56,PCI-C154(+), PCI-8154/8158(DB-8150) , PCI-8254/58 /

AMP-204/8C

描述:

此函数用于在指定的线性比较器中获取当前的比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_linear_cmp( I32 Board_ID, I32 LCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_trigger_linear_cmp(ByVal Board_ID As Long, ByVal LCmpCh As Long, CmpVal  
As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 LCmpCh: 指定线性比较器的通道编号。从零开始。

对于 PCI-8254/58 / AMP-204/8C, I32 LCmpCh: 指定线性比较器的通道编号。从零开始。范围从 0 到 3。

I32 *CmpVal: 返回比较器中当前的比较值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 CmpVal;  
ret = APS_get_trigger_linear_cmp( 0, 0, &CmpVal );  
If( ret != ERR_NoError )  
{ // 错误 , 显示消息。  
}
```

还可以看看:

[APS_get_trigger_table_cmp\(\)](#)

<code>APS_get_trigger_count</code>	获取触发计数.
支持的产品:PCI-8253/56, PCI-C154(+), PCI-8154/8158(DB-8150), EMX-100 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取触发的计数器值。该值表示最后一次计数器复位后的总触发脉冲。非常适合检查比较的次数。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_count( I32 Board_ID, I32 TrgCh, I32 *TrgCnt );
```

Visual Basic:

```
APS_get_trigger_count(ByVal Board_ID As Long, ByVal TrgCh As Long, TrgCnt As Long)
As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TrgCh: 指定的触发输出计数器通道编号。从零开始。

对于 PCI-8254/58 / AMP-204/8C, I32 TrgCh: 指定的触发输出计数器通道编号。从零开始。范围从 0 到 3。

对于 EMX-100: I32 TrgCh: 设备的指定触发输出计数器通道编号 (0 或 1)。

I32 *TrgCnt: 返回触发计数器值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Ret;
I32 TrgCnt;
Ret = APS_get_trigger_count( 0, 0, &TrgCnt );
If( ret != ERR_NoError )
{ // 错误，显示消息。
}
```

还可以看看:

[APS_reset_trigger_count\(\)](#)

APS_reset_trigger_count	重置触发计数.
-------------------------	---------

支持的产品:PCI-8253/56,PCI-C154(+), PCI-8154/8158(DB-8150), EMX-100 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于将触发计数器重置为零。

句法:

C/C++:

```
I32 FNTYPE APS_reset_trigger_count( I32 Board_ID, I32 TrgCh );
```

Visual Basic:

```
APS_reset_trigger_count( ByVal Board_ID As Long, ByVal TrgCh As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TrgCh: 触发计数器通道编号。从零开始。

对于 EMX-100: I32 TrgCh: 触发设备的计数器通道编号 (0 或 1)。

对于 PCI-8254/58 / AMP-204/8C, I32 TrgCh: 触发计数器通道编号。从零开始。范围从 0 到 3。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
ret = APS_reset_trigger_count( 0, 0 );
ret = APS_reset_trigger_count( 0, 1 );
ret = APS_reset_trigger_count( 0, 2 );
ret = APS_reset_trigger_count( 0, 3 );
```

...

还可以看看:

[APS_get_trigger_count\(\)](#)

APS_enable_trigger_fifo_cmp	启用触发 FIFO 比较器
支持的产品:PCI-C154(+), PCI-8154/8158(DB-8150) , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于启用/禁用 fifo 比较器。 当用户禁用 fifo 比较器时，fifo 数据将被重置。

句法:

C/C++:

```
I32 FNTYPE APS_enable_trigger_fifo_cmp( I32 Board_ID, I32 FCmpCh, I32 Enable );
```

Visual Basic:

```
APS_enable_trigger_fifo_cmp (ByVal Board_ID As Long, ByVal FCmpCh As Long, ByVal
Enable As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FCmpCh: 指定的通道编号。（在 DB-8150 中仅支持通道 0）

I32 Enable: 启用/禁用 fifo 比较器.

0: 禁用 fifo 比较器

1: 启用 fifo 比较器

注意：在开始 FIFO 比较之前，用户必须先启用 fifo 比较器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
I32 FCmpCh = 0;
I32 Enable = 1 // 启用 fifo 比较器
I32 ret = 0;
I32 DataArr[3]={1000,2000,3000};
I32 ArraySize=3;
I32 ShiftFlag = 1; //自动将一个数据切换至 FIFO 比较器
```

```
ret = APS_set_trigger_fifo_data(Board_ID, FCmpCh, DataArr, ArraySize, ShiftFlag );
ret = APS_enable_trigger_fifo_cmp(Board_ID, FCmpCh, Enable );
```

还可以看看:

<code>APS_get_trigger_fifo_cmp</code>	获取触发 FIFO 比较器
支持的产品:PCI-C154(+), PCI-8154/8158(DB-8150)	

描述:

此函数用于从 FIFO 比较器获取当前比较数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_fifo_cmp( I32 Board_ID, I32 FCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_trigger_fifo_cmp (ByVal Board_ID As Long, ByVal FCmpCh As Long, *CmpVal  
As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FCmpCh: 指定的通道编号。(在 DB-8150 中仅支持通道 0)

I32 *CmpVal: 比较器中当前的比较数据。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;  
I32 FCmpCh = 0;  
I32 CmpVal = 0  
I32 ret = 0;  
ret = APS_get_trigger_fifo_cmp(Board_ID, FCmpCh, &CmpVal);
```

还可以看看:

<code>APS_get_trigger_fifo_status</code>	获取触发 FIFO 状态
支持的产品:PCI-C154(+), PCI-8154/8158(DB-8150)	

描述:

获取触发 FIFO 状态

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_fifo_status( I32 Board_ID, I32 FCmpCh, I32 *Fifosts );
```

Visual Basic:

```
APS_get_trigger_fifo_status (ByVal Board_ID As Long, ByVal FCmpCh As Long, Fifosts  
As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FCmpCh: 指定的通道编号。(在 DB-8150 中仅支持通道 0)

I32 * Fifosts: fifo 数据的当前状态。

Bit0=0: 不是空的 , Bit0=1: 空的

Bit1=0: 不满 , Bit1=1; 满

Bit2=0: 等于或大于预设水平 ,

Bit2=1: 低于预设水平

其他位保留

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
I32 Board_ID = 0;  
I32 FCmpCh = 0;  
I32 Fifosts = 0  
I32 ret = 0;  
ret = APS_get_trigger_fifo_status(Board_ID, FCmpCh, & Fifosts);
```

还可以看看:

APS_set_trigger_fifo_data	设置触发 FIFO 状态
支持的产品:PCI-C154(+), PCI-8154/8158(DB-8150)	

描述:

此函数用于将比较数据数组设置为 FIFO。FIFO 的容量为 2097151。当 FIFO 的状态为满时，无法将数据设置为 FIFO。此函数不会检查 FIFO 状态。使用此函数时，还应通过“APS_enable_trigger_fifo_cmp”功能启用 fifo 比较器。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_fifo_data( I32 Board_ID, I32 FCmpCh, I32 *DataArr, I32
ArraySize, I32 ShiftFlag );
```

Visual Basic:

```
APS_set_trigger_fifo_data (ByVal Board_ID As Long, ByVal FCmpCh As Long, DataArr
As Long, ByVal ArraySize As Long, ByVal ShiftFlag As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FCmpCh: 指定的通道编号。（在 DB-8150 中仅支持通道 0）

I32 *DataArr : FIFO 数据数组的索引指针。

I32 ArraySize : FIFO 数据数组的大小。（1 – 1026）

I32 ShiftFlag : 自动将一个 FIFO 数据切换至比较器。

 0: 禁止将一个 FIFO 数据自动切换至比较器。

 1: 自动将一个 FIFO 数据切换至比较器

注意：在开始 FIFO 比较之前，用户必须先将一个 FIFO 数据自动切换至比较器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 Board_ID = 0;
I32 FCmpCh = 0;
I32 DataArr ={1000,2000,3000}
I32 ArraySize = 3;
I32 Enable = 1; // 开始 FIFO 比较
```

```
I32 ret = 0;  
I32 ShiftFlag = 1; // 自动将一个 FIFO 数据切换至比较器  
  
ret = APS_set_trigger_fifo_data(Board_ID, FCmpCh, DataArr, ArraySize, ShiftFlag );  
ret = APS_enable_trigger_fifo_cmp(Board_ID, FCmpCh, Enable );
```

注意：请先设置触发 FIFO 数据，然后启用 FIFO 比较器，比较器将正常触发中断。

还可以看看：

APS_start_timer	启动/停止计时器
支持的产品:PCI-C154(+), PCI-8154/8158(DB-8150)	

描述:

在 PCI-C154(+) 中 , 此函数用于启动/停止计时器 8。计时器 8 用于模拟编码器 , 该编码器用作比较器源。

句法:

C/C++:

```
32 FNTYPE APS_start_timer( I32 Board_ID, I32 TrgCh, I32 Start );
```

Visual Basic:

```
APS_start_timer (ByVal Board_ID As Long, ByVal TrgCh As Long, ByVal Start As Long )
As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TrgCh: 指定的通道编号。 (在 PCI-C154 (+) 中 : 仅支持 CH0)

I32 Start: 启动/停止计时器

0 : 停止计时器

1 : 启动计时器

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
I32 Board_ID = 0;
```

```
I32 TrgCh = 0;
```

```
I32 Start = 1 // 启动计时器
```

```
I32 ret = 0;
```

```
ret = APS_start_timer(Board_ID, TrgCh, Start );
```

还可以看看:

APS_get_timer_counter	获取计时器计数值
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取计时器计数值。

在 PCI-C154(+)中，此函数用于获取定时器 8 的计数值。计时器 8 用于模拟编码器，该编码器用作比较器源。

句法:

C/C++:

```
I32 FNTYPE APS_get_timer_counter (I32 Board_ID, I32 TmrCh, I32 *Cnt);
```

Visual Basic:

```
APS_get_timer_counter( ByVal Board_ID As Long, ByVal TmrCh As Long, Cnt As Long )As Long
```

参数:

对于 PCI-C154(+):

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TmrCh: 指定的通道编号。**(在 PCI-C154 (+) 中 : 仅支持 CH0)**

I32 *Cnt: 获取计时器计数值

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TmrCh: 指定的计时器通道编号。从零开始。

仅通道 0 可用。

I32 *TmrCnt: 返回计时器计数器值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI-C154(+)

```
I32 ret = 0;
I32 Board_ID = 0;
I32 TmrCh = 0;
I32 Cnt = 0;
ret = APS_get_timer_counter ( Board_ID, TmrCh, &Cnt );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 Ret;  
I32 TmrCnt;  
Ret = APS_get_timer_counter( 0, 0, &TmrCnt ); //从计时器通道 0 获取计数器  
If( ret != ERR_NoError )  
{ // 错误，显示消息。  
}
```

还可以看看：

[APS_set_timer_counter\(\)](#)

APS_set_timer_counter	设置计时器计数值
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

此函数用于设置计时器计数值。

在 PCI-C154(+)中，此函数用于设置计时器 8 的计数值。计时器 8 用于模拟编码器，该编码器用作比较器源。

句法:

C/C++:

```
I32 FNTYPE APS_set_timer_counter ( I32 Board_ID, I32 TmrCh, I32 Cnt );
```

Visual Basic:

```
APS_set_timer_counter( ByVal Board_ID As Long, ByVal TmrCh As Long, ByVal Cnt As Long )As Long
```

参数:

For PCI-C154(+):

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TmrCh: 指定的通道编号。(在 PCI-C154 (+) 中 : 仅支持 CH0)

I32 Cnt: 设置计时器计数值.

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TmrCh: 计时器计数器通道编号。从零开始。

PCI-8258 中只有一个通道可用。

I32 TmrCnt: 指定计时器计数器值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCI-C154(+)

```
I32 ret = 0;
I32 Board_ID = 0;
I32 TmrCh = 0;
I32 Cnt = 0;
ret = APS_set_timer_counter ( Board_ID, TmrCh, Cnt );
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C

```
I32 ret;  
  
//set timer counter channel 0 to 100  
ret = APS_set_timer_counter( 0, 0, 100 );  
  
...
```

还可以看看:

APS_get_timer_counter()

APS_start_trigger_timer	启动触发计时器
支持的产品:PCI-C154(+)	

描述:

此函数用于启动/停止定期生成触发信号的计时器。

句法:

C/C++:

```
I32 FNTYPE APS_start_trigger_timer ( I32 Board_ID, I32 TrgCh, I32 Start );
```

Visual Basic:

```
APS_start_trigger_timer ( ByVal Board_ID As Long, ByVal TrgCh As Long, ByVal Start As  
Long )As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TrgCh: 指定的通道编号。

在 **PCI-C154(+)中: 支持 CH0 ~ CH3**

I32 Start: Start=1; 启动计时器

Start=0; 停止计时器

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 Board_ID = 0;  
I32 TrgCh = 0;  
I32 Start = 1; // 启动计时器  
ret = APS_set_timer_counter ( Board_ID, TrgCh, Start);  
.....  
Start = 0;// 停止计时器  
ret = APS_set_timer_counter ( Board_ID, TrgCh, Start);
```

还可以看看:

[APS_get_trigger_timer_counter\(\)](#)

APS_get_trigger_timer_counter	获取计时器计数值
支持的产品:PCI-C154(+)	

描述:

此函数用于获取触发定时器计数值，该值会定期生成触发信号。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_timer_counter ( I32 Board_ID, I32 TmrCh, I32 *TmrCnt );
```

Visual Basic:

```
APS_get_trigger_timer_counter ( ByVal Board_ID As Long, ByVal TmrCh As Long,
TmrCnt As Long )As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TmrCh: 指定的通道编号。

在 PCI-C154(+)中: 支持 CH0 ~ CH3)

I32 *TmrCnt: 获取计时器计数值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret=0;
I32 Board_ID = 0;
I32 TmrCh = 0;
I32 TmrCnt=0;
ret = APS_get_trigger_timer_counter ( Board_ID, TmrCh, &TmrCnt );
```

还可以看看:

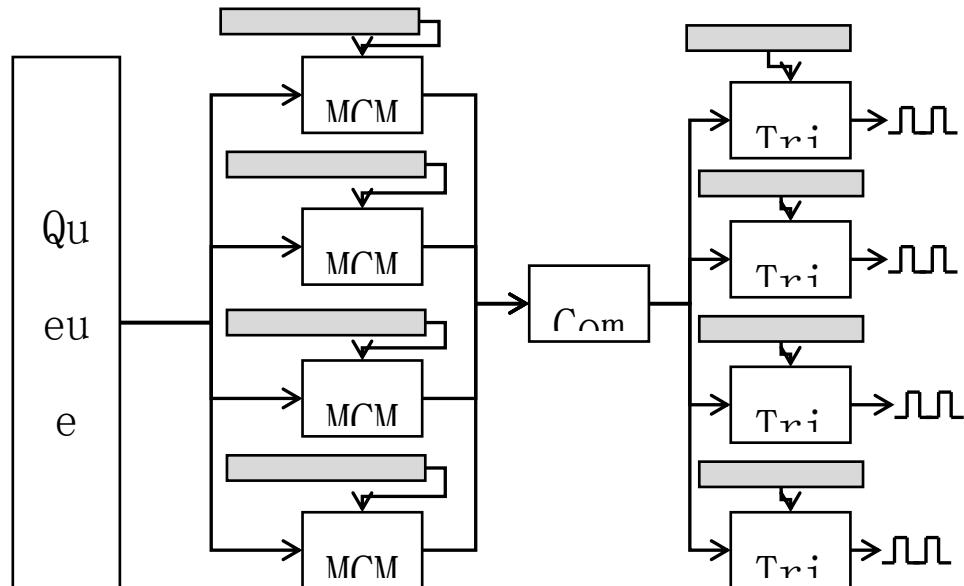
[APS_start_trigger_timer\(\)](#)

APS_set_multi_trigger_table	设置表比较
-----------------------------	-------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于将数据推入表中 (FIFO) 进行比较。有四个比较器专门设计用于多维比较应用。比较点最初被推入队列。用户可以使用触发参数从编码器0~7中选择比较器源。允许指定任意触发通道以生成PWM。比较该点后，指定的触发通道将产生一个PWM信号，其对应的计数器将同时加1。



比较器配置:

维度	配置
2	在触发参数TGR_MCMP0_SRC / TGR_MCMP1_SRC中选择源
3	在触发参数TGR_MCMP0_SRC / TGR_MCMP1_SRC / TGR_MCMP2_SRC中选择源
4	在触发参数TGR_MCMP0_SRC / TGR_MCMP1_SRC / TGR_MCMP2_SRC / TGR_MCMP3_SRC中选择源

触发输出配置

通道	配置
0	在触发参数TGR_TRG0_SRC中设置bit 6
1	在触发参数TGR_TRG1_SRC中设置bit 6
2	在触发参数TGR_TRG2_SRC中设置bit 6
3	在触发参数TGR_TRG3_SRC中设置bit 6

句法:

C/C++:

```
I32 FNTYPE APS_set_multi_trigger_table( I32 Board_ID, I32 Dimension, MCMP_POINT
*Point, I32 PointSize, I32 Window );
```

Visual Basic:

APS_set_multi_trigger_table(ByVal Board_ID As Long, ByVal Dimension As Long, ByVal DataArr() As MCMP_POINT, ByVal ArraySize As Long, ByVal Window As Long) As Long

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Dimention: 2~4 维

MCMP_POINT *Point: 用于比较器的点数组。有关详细信息，请参见下面的描述。

```
//多维比较器
typedef struct
{
    F64 axisX; // 多维比较器0的x轴数据
    F64 axisY; // 多维比较器1的y轴数据
    F64 axisZ; // 多维比较器2的z轴数据
    F64 axisU; // 多维比较器3的u轴数据
    U32 chInBit; // pwm输出通道，格式为Bit
}MCMP_POINT;
```

I32 PointSize: 点数组的大小。

I32 Window: 指定比较范围。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
void main()
{
    I32 ret = 0;
    U32 i = 0;
    I32 BoardID_InBits;
    I32 BoardID = 0;
    I32 Mode = 0; //按系统分配
    I32 msts; // 运动状态
    MCMP_POINT DataArr[10000];
    I32 data = 0;
    U32 totalPoint = 5000;
    U32 window = 10;
    U32 dimension = 2;
    I32 Axis_ID_Array[2] = {0, 1};
    I32 Distance_Array[2] = {1100, 2200 };
    I32 Max_Linear_Speed = 20000;
```

```

MCMP_POINT Point;
printf("\n");
// ****
// 初始化
// ****
ret = APS_initial( &BoardID_InBits, Mode);
if(ret)
{
    printf("APS initial fail\n");
    goto TEST_END;
}
printf("APS version = %d\n", (I32)APS_version() );
// ****
//设置触发参数
// ****
//设置比较器源：比较器0的编码为0，比较器1的编码器为1
ret = APS_set_trigger_param( BoardID, TGR_MCMP0_SRC, 0 );
if(ret)
{
    printf("APS_set_trigger_param1 fail\n");
    goto TEST_END;
}
ret = APS_set_trigger_param( BoardID, TGR_MCMP1_SRC, 1 );
if(ret)
{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

//设置PWM输出通道0
ret = APS_set_trigger_param( BoardID, TGR_TRG0_SRC, 0x40 );
if(ret)
{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

//设置PWM输出通道1
ret = APS_set_trigger_param( BoardID, TGR_TRG1_SRC, 0x40 );
if(ret)

```

```

{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

ret = APS_set_trigger_param( BoardID, TGR_TRG_EN, 0xF );
ret = APS_set_trigger_param( BoardID, TGR_TRG2_SRC, 0x40 );
ret = APS_set_trigger_param( BoardID, TGR_TRG3_SRC, 0x40 );

//启用所有触发输出通道
ret = APS_set_trigger_param( BoardID, TGR_TRG_EN, 0xF );

// *****
//重置并读取触发计数
// *****
//重置PWM通道0触发计数
ret = APS_reset_trigger_count( BoardID, 0 );
ret = APS_reset_trigger_count( BoardID, 1 );
ret = APS_reset_trigger_count( BoardID, 2 );
ret = APS_reset_trigger_count( BoardID, 3 );
if(ret)
{
    printf("APS_reset_trigger_count fail\n");
    goto TEST_END;
}

// *****
//开启伺服
// *****

//开启轴伺服
ret = APS_set_servo_on( 0, 1 );
if(ret)
{
    printf("Servo on fail\n");
    goto TEST_END;
}
ret = APS_set_servo_on( 1, 1 );
if(ret)
{

```

```

        printf("Servo on fail\n");
        goto TEST_END;
    }

//重置命令
ret = APS_set_command( 0, 0 );
if(ret)
{
    printf("APS_set_command fail\n");
    goto TEST_END;
}
ret = APS_set_command( 1, 0 );
if(ret)
{
    printf("APS_set_command fail\n");
    goto TEST_END;
}

// *****
//设定比较点
// *****

//准备比较点
for(i=0; i<totalPoint; i++)
{
    DataArr[i].axisX = i * 10 + 10;
    DataArr[i].axisY = i * 20 + 20;
    DataArr[i].axisZ = 0;
    DataArr[i].axisU = 0;
    DataArr[i].chInBit = 0xF;
}

//将比较点设置为队列
ret = APS_set_multi_trigger_table( BoardID, dimension, DataArr, totalPoint,
window );
if(ret)
{
    printf("APS_set_multi_trigger_table fail\n");
    goto TEST_END;
}

```

```

//检查比较器数据
ret = APS_get_multi_trigger_table_cmp( BoardID, dimension, &Point );
if(ret)
{
    printf("APS_get_trigger_table_cmp fail\n");
    goto TEST_END;
}
printf("Point in comparator: axisX = %f  axisY = %f\n", Point.axisX, Point.axisY );

// *****
//启动电机并读取状态
// *****

//开始插补
ret = APS_relative_linear_move( dimension, Axis_ID_Array, Distance_Array,
Max_Linear_Speed );
if(ret)
{
    printf("APS_relative_linear_move fail\n");
    goto TEST_END;
}

//检查CSTP
while(1)
{
    F64 data1,data2;
    I32 data3, data4, data6, data7;
    U32 data5 = 0;
    U32 data8;
    msts = APS_motion_status(0);

    //ret = APSI_8258_read_fpga( 0, 1, 0x37c, &data5 );
    APS_get_trigger_count( BoardID, 0, &data3 );
    APS_get_trigger_count( BoardID, 1, &data4 );
    APS_get_trigger_count( BoardID, 2, &data6 );
    APS_get_trigger_count( BoardID, 3, &data7 );
    APS_get_position_f( 0, &data1 );
    APS_get_position_f( 1, &data2 );
    printf("fbk0 = %f fbk1 = %f cnt0 = %d cnt1 = %d cnt2 = %d cnt3 = %d ch

```

```

= 0x%x\n", data1, data2, data3, data4,data6,data7, data5);
    if( msts & 0x1)
        break;
    Sleep(10);
}

// *****
//读取最终的PWM计数
// *****
//检查PWM通道0触发计数器
ret = APS_get_trigger_count( BoardID, 0, &data );
if(ret)
{
    printf("APS_get_trigger_count fail\n");
    goto TEST_END;
}
printf("Final pwm count 0 = %d\n", data );

//检查PWM通道1触发计数器
ret = APS_get_trigger_count( BoardID, 1, &data );
if(ret)
{
    printf("APS_get_trigger_count fail\n");
    goto TEST_END;
}
printf("Final pwm count 1 = %d\n", data );

//检查PWM通道2触发计数器
ret = APS_get_trigger_count( BoardID, 2, &data );
if(ret)
{
    printf("APS_get_trigger_count fail\n");
    goto TEST_END;
}
printf("Final pwm count 2 = %d\n", data );

//检查PWM通道3触发计数器
ret = APS_get_trigger_count( BoardID, 3, &data );
if(ret)
{

```

```
    printf("APS_get_trigger_count fail\n");
    goto TEST_END;
}
printf("Final pwm count 3 = %d\n", data );

TEST_END:

//轴伺服关闭
ret = APS_set_servo_on( 0, 0 );
ret = APS_set_servo_on( 1, 0 );
ret = APS_close();
system("PAUSE");
}
```

APS_get_multi_trigger_table_cmp	获取当前表比较值
---------------------------------	----------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于在指定的表比较器中获取当前比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_table_cmp( I32 Board_ID, I32 Dimension, MCMP_POINT  
*Point );
```

Visual Basic:

```
APS_get_trigger_table_cmp (ByVal Board_ID As Long, ByVal TCmpCh As Long, ByRef  
CmpVal As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Dimension: 2~4 维

MCMP_POINT *Point: 返回比较器中的当前比较值。有关详细信息，请参见
type_define.h。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

APS_set_trigger_table_data	设置表比较器数据 (快速比较表触发函数)
----------------------------	------------------------

描述:

该函数属于快速表比较触发函数，用于将比较数据设置为比较表。比较数据的大小受 FIFO 可用大小限制，其最大值为 40。

句法:

C/C++:

```
I32 FNTYPE APS_set_trigger_table_data( I32 Board_ID, I32 TCmpCh, I32 *DataArr, I32  
ArraySize );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。范围从 0 到 3。

I32 *DataArr: 比较数据数组。

I32 ArraySize : 比较数据数组的大小。范围是 1~40。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 FreeSize = 0;  
I32 FifoSts = 0  
I32 Data = 0;  
I32 DataArr[1000];  
U32 usedPoint=0;  
U32 residualPoint=0;  
U32 totalPoint = 500;  
  
//启用表比较触发器  
ret = APS_enable_trigger_table( 0, 0, 1 );  
  
//重置FIFO比较数据  
ret = APS_reset_trigger_table( 0, 0 );  
  
//生成比较数据  
for(i=0; i<totalPoint; i++)  
    DataArr[i] = i * 100 + 100;  
  
while(1)  
{
```

```

//获取剩余数据的大小
residualPoint = totalPoint - usedPoint;

//获取FIFO状态
ret = APS_get_trigger_table_status( 0, 0, &FreeSize, &FifoSts );

//获取当前的FIFO比较数据
APS_get_trigger_cmp_value( 0, 0, &Data );

// //将比较数据设置到FIFO
if(FreeSize >= 40)
{
    if(residualPoint >= 40)
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], 40 );
        if(ret ==0)
            usedPoint += 40;
    }
    else
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], residualPoint );
        if(ret ==0)
            usedPoint += residualPoint;
    }
}
else
{
    if(FreeSize >= residualPoint)
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], residualPoint );
        if(ret ==0)
            usedPoint += residualPoint;
    }
    else
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], FreeSize );
        if(ret ==0)
            usedPoint += FreeSize;
    }
}

//完成将比较数据设置到FIFO
if(usedPoint == totalPoint)
    break;

```

```
    Sleep(1);  
}
```

还可以看看：

APS_set_trigger_table_data();APS_get_trigger_table_status();APS_get_trigger_cmp_value(); APS_enable_trigger_table();APS_reset_trigger_table()

APS_get_trigger_table_status	获取表比较器状态 (快速比较表触发函数)
------------------------------	------------------------

描述:

该函数属于快速表比较触发函数，用于获取表比较器的FIFO状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_table_status( I32 Board_ID, I32 TCmpCh, I32 *FreeSpace,  
I32 *FifoSts );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。范围从 0 到 3。

I32 *FreeSpace: FIFO 的可用大小。总可用大小为 1254。

I32 *FifoSts: FIFO 状态 : 位 0 = 1 表示 FIFO 已满 , 位 1 = 1 表示 FIFO 空。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_set_trigger_table_data();APS_get_trigger_table_status();APS_get_trigger_cmp_value(); APS_enable_trigger_table();APS_reset_trigger_table()

APS_get_trigger_cmp_value	获取表比较器值 (快速比较表触发函数)
---------------------------	-----------------------

描述:

该函数属于快速表比较触发函数，用于获取表比较器的当前比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_trigger_cmp_value( I32 Board_ID, I32 TCmpCh, I32 *CmpVal );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。范围从 0 到 3。

I32 *CmpVal: 表比较器的当前比较值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_set_trigger_table_data();APS_get_trigger_table_status();APS_get_trigger_cmp_value(); APS_enable_trigger_table();APS_reset_trigger_table()

APS_enable_trigger_table	启用表比较器 (快速比较表触发函数)
--------------------------	----------------------

描述:

此函数属于快速表比较触发函数，用于启用表比较器。

句法:

C/C++:

```
I32 FNTYPE APS_enable_trigger_table( I32 Board_ID, I32 TCmpCh, I32 Enable );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。范围从 0 到 3。

I32 Enable: 设置 Enable = 1 以启动表比较器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_set_trigger_table_data();APS_get_trigger_table_status();APS_get_trigger_cmp_value(); APS_enable_trigger_table();APS_reset_trigger_table()

APS_reset_trigger_table	重置表比较器 (快速比较表触发函数)
-------------------------	----------------------

描述:

此函数属于快速表比较触发函数，用于重置表比较器的FIFO。

句法:

C/C++:

```
I32 FNTYPE APS_reset_trigger_table( I32 Board_ID, I32 TCmpCh );
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 TCmpCh: 指定比较表编号。从零开始。范围从 0 到 3。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

APS_set_trigger_table_data();APS_get_trigger_table_status();APS_get_trigger_cmp_value(); APS_enable_trigger_table();APS_reset_trigger_table()

19. 程序下载

APS_load_vmc_program	将 VMC 文件加载到任务存储器
----------------------	------------------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于将 VMC 文件获取到任务存储器。

注意 : AMP 系列不支持此函数.

句法:

C/C++:

```
I32 FNTYPE APS_load_vmc_program ( I32 Board_ID, I32 TaskNum, const char *pFile, I32 Password);
```

Visual Basic:

```
APS_load_vmc_program (ByVal Board_ID As Long, ByVal TaskNum As Long, pFile As String, ByVal Password As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TaskNum: 在 0 到 7 之间指定一个任务编号。

I32 *pFile: 指定由 MCPro2.exe 创建的 VMC 文件。

I32 Password: 输入指定的安全密码。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;
```

```
//将 “BubbleSort.txt” VMC 文件加载到指定任务。
```

```
ret = APS_load_vmc_program( boardId, taskNum, “BubbleSort.txt” , 0 );
```

还可以看看:

[APS_save_vmc_program\(\)](#)

APS_save_vmc_program

从任务存储器中保存至 VMC 文件

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于将任务程序从任务存储器保存到 VMC 文件。

注意 : AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_save_vmc_program( I32 Board_ID, I32 TaskNum, const char *pFile, I32 Password);
```

Visual Basic:

```
APS_save_vmc_program (ByVal Board_ID As Long, ByVal TaskNum As Long, pFile As String, ByVal Password As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TaskNum: 在 0 到 7 之间指定一个任务编号。

I32 *pFile: 指定一个 VMC 文件来保存。

I32 Password: 输入指定的安全密码。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;
```

//将任务程序保存到名为 “ BubbleSort.txt” 的 VMC 文件中。

```
ret = APS_save_vmc_program( boardId, taskNum, “BubbleSort.txt” , 0 );
```

还可以看看:

[APS_load_vmc_program\(\)](#)

APS_set_task_mode	设置任务运行模式
-------------------	----------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

该函数用于设置任务运行模式.

注意 : AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_set_task_mode( I32 Board_ID, I32 TaskNum, U8 Mode, U16 LastIP );
```

Visual Basic:

```
APS_set_task_mode (ByVal Board_ID As Long, ByVal TaskNum As Long, ByVal Mode As  
Byte, ByVal LastIP As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TaskNum: 在 0 到 7 之间指定一个任务编号。

U8 Mode: 设置两种运行模式。

0 : 普通模式。 1 : 重复模式。

U16 LastIP: 上一条命令偏移量。仅在重复模式下可用。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;  
U16 lastIP = 0;  
  
//将任务 0 设置为普通模式。正常模式下 IP 将被忽略。  
ret = APS_set_task_mode ( boardId, taskNum, 0, &lastIP );
```

还可以看看:

[APS_get_task_mode \(\)](#)

APS_get_task_mode	获取任务运行模式
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

该函数用于获取任务运行模式.

注意 : AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_get_task_mode( I32 Board_ID, I32 TaskNum, U8 *Mode, U16 *LastIP );
```

Visual Basic:

```
APS_get_task_mode (ByVal Board_ID As Long, ByVal TaskNum As Long, Mode As Byte,  
ByVal LastIP As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TaskNum: 在 0 到 7 之间指定一个任务编号。

U8 *Mode: 设置两种运行模式。

0 : 普通模式。 1 : 重复模式。

U16 *LastIP: 上一条命令偏移量。仅在重复模式下可用。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;  
U8 mode = 0;  
U16 lastIP = 0;  
  
//从任务 0 获取运行模式。正常模式下 IP 将被忽略。  
ret = APS_get_task_mode ( boardId, taskNum, &mode, &lastIP);
```

还可以看看:

[APS_set_task_mode \(\)](#)

APS_start_task	启动任务控制命令
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

该函数用于启动任务控制命令.

注意 : AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_start_task( I32 Board_ID, I32 TaskNum, I32 CtrlCmd );
```

Visual Basic:

```
APS_start_task (ByVal Board_ID As Long, ByVal TaskNum As Long, ByVal CtrlCmd As  
Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 CtrlCmd : 控制命令。

- 0 : TSK_RESET. 重置任务。 (不启动程序)
- 1 : TSK_RESTART. 重新启动任务。 (同时启动程序)
- 2 : TSK_STOP. 停止程序。
- 3 : TSK_RUN. 启动程序。
- 4 : TSK_STEP. 进行 (运行) 一个命令。 然后停下来。
- 5 : TSK_STEP_P. 运行 , 直到并行位 == 0

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;  
I32 CtrlCmd = 3;  
  
//运行任务 0 的程序  
ret = APS_start_task ( boardId, taskNum, CtrlCmd );
```

还可以看看:

[APS_get_task_info\(\)](#); [APS_get_task_msg\(\)](#)

APS_get_task_info	获取任务信息
-------------------	--------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

该函数用于获取任务信息。

注意 : AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_get_task_info( I32 Board_ID, I32 TaskNum, TSK_INFO *Info );
```

Visual Basic:

```
APS_get_task_info (ByVal Board_ID As Long, ByVal TaskNum As Long, pFile As String,  
ByRef Info As TSK_INFO) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TaskNum: 在 0 到 7 之间指定一个任务编号。

TSK_INFO *Info: 任务信息。

```
typedef struct _TSK_INFO
{
    U16 State;           //任务状态 : 0 : 停止 , 1 : 运行 , 2 : 步骤 , 3 : Step_p 4 : ??
    U16 RunTimeErr;     //状态处于 ERROR 时的运行错误代码。
    U16 IP;             //注册 IP
    U16 SP;             //注册 SP
    U16 BP;             //注册 BP
    U16 MsgQueueSts;   //消息队列状态 , 请参阅以下定义
} TSK_INFO, *PTSK_INFO;
```

U16 MsgQueueSts: (注意 : 所有任务仅共享一个消息队列。)

BitNum	状态描述
0	MPU_MSG_EMPTY: 消息队列为空
1	MPU_MSG_FULL: 队列已满
2	MPU_MSG_NOT_EMPTY
3~15	Reserved. 0

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
```

```
I32 boardId = 0;  
I32 taskNum = 0;  
TSK_INFO info;  
  
//获取任务 0 的信息  
ret = APS_get_task_info( boardId, taskNum, &info );
```

还可以看看:

APS_start_task(); APS_get_task_msg()

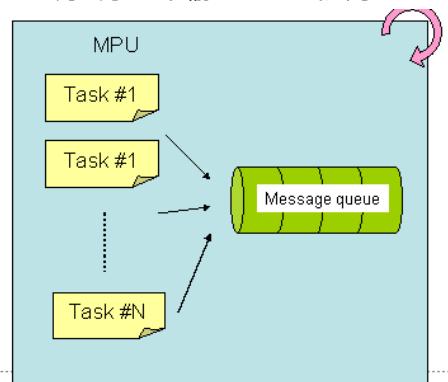
APS_get_task_msg

获取所有任务的信息

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于获取任务的信息。所有任务仅共享一个消息队列。这对于调试很有用。用户可以将一些调试字符串输出到消息队列。



注意：AMP 系列不支持此函数。

句法:

C/C++:

```
I32 FNTYPE APS_get_task_msg( I32 Board_ID, U16 *Queuests, U16 *ActualSize, U8  
*CharArr );
```

Visual Basic:

```
APS_get_task_msg (ByVal Board_ID As Long, ByRef Queuests As UShort, ByRef  
ActualSize As UShort, ByRef CharArr As Byte) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

U16 *Queuests: 消息队列状态。请参考下图。

U16 *ActualSize: 实际返回消息的大小。[0~128]

U8 *CharArr: 返回 char 消息。最大数组大小为 128 个字节。(U8 CharArr [n = 128])

取决于 ActualSize, 如果 n > = ActualSize, 数据毫无意义, 可以忽略。

Queuests 的定义:

BitNum	状态描述
0	MPU_MSG_EMPTY: 消息队列为空
1	MPU_MSG_FULL: 队列已满
2	MPU_MSG_NOT_EMPTY
3~15	Reserved. 0

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret = 0;  
I32 boardId = 0;  
U16 queueSts = 0;  
U16 actualSize = 0;  
U8 charArr[128];  
  
//获取所有任务的信息  
ret = APS_get_task_msg(boardId, &queueSts, actualSize, &charArr );;
```

还可以看看:

`APS_start_task(); APS_get_task_info()`

20. 手动脉冲发生器输入

APS_manual_pulser_start	开始手动脉冲发生器操作
-------------------------	-------------

支持的产品:PCIe-8154/58, PCI-C154(+), PCI-8254/58 / AMP-204/8C

描述:

对于 PCIe-8154/58, PCI-C154(+) , 此函数用于启用/禁用手动脉冲发生器输入。当禁用手动脉冲发生器时 , 从 PA/PB 引脚输入的脉冲信号将被忽略。

对于 PCI-C154+ , 默认设置为启用模式。

对于 PCI-8254/58 / AMP-204/8C , 此函数用于启动手动脉冲发生器操作。它支持一组 PA/PB 引脚 , 可以连接易于使用的手动脉冲发生器和解码器 , 以进行单轴位置控制。解码器允许来自 PA 和 PB 引脚的输入信号 , 包含正脉冲和负脉冲 (CW/CCW) , OUT/DIR 或 90 度相位差信号 (AB 相) 。用户应根据易于使用的手动脉冲发生器的规格 , 选择正确的输入信号类型。如有必要 , 还可以反转 PA 和 PB 信号或更改计数方向。这些设置可以通过轴参数进行配置。

发出 “启用” 命令后 , 手动脉冲发生器将继续等待接收新的输入信号 , 然后立即让电动机开始运动。运动状态的 Bit 29 可以指示启用或禁用手动脉冲发生器操作。在使用此函数之前 , 用户应使用 APS_manual_pulser_velocity_move()I 来指定哪个轴以及其最大速度。出于安全考虑 , 此脉冲发生器操作将在以下情况立即终止:

- 1、 “ disable” 命令由用户发出。
- 2、 运动 IO , 例如 PEL/MEL , ALM 和 EMG 被触发。

对于 PCIe-833x , 此函数用于启动手动脉冲发生器操作。它支持一组 PA/PB 引脚 , 可以连接易于使用的手动脉冲发生器和解码器 , 以进行单轴位置控制。解码器允许来自 PA 和 PB 引脚的输入信号 , 包含正脉冲和负脉冲 (CW/CCW) , OUT/DIR 或 90 度相位差信号 (AB 相) 。用户应根据易于使用的手动脉冲发生器的规格 , 选择正确的输入信号类型。如有必要 , 还可以反转 PA 和 PB 信号或更改计数方向。这些设置可以通过轴参数进行配置。

解码器允许来自 PA 和 PB 引脚的输入信号 , 包含正脉冲和负脉冲 (CW/CCW) , OUT/DIR 或 90 度相位差信号 (AB 相) 。用户应根据易于使用的手动脉冲发生器的规格 , 选择正确的输入信号类型。如有必要 , 还可以反转 PA 和 PB 信号或更改计数方向。这些设置可以通过轴参数进行配置。

- 1、 “ disable” 命令由用户发出。
- 2、 运动 IO , 例如 PEL/MEL , ALM 和 EMG 被触发。

句法:

C/C++:

```
I32 FNTYPE APS_manual_pulser_start ( I32 Board_ID, I32 Enable );
```

Visual Basic:

```
APS_manual_pulser_start ( ByVal Board_ID As Long, ByVal Enable As Long) As Long
```

参数:

For PCIe-8154/58, PCI-C154(+):

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Enable: 启用/禁用手动脉冲发生器输入。 0 : 禁用 , 1 : 启用。

对于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x:

I32 Board_ID: 目标控制器的 ID。 通过成功调用 APS_initial() 来检索它。

I32 Enable: 启用脉冲发生器操作。

1 : 启动手动脉冲发生器操作

0 : 禁用手动脉冲发生器操作

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCIe-8154/58, PCI-C154(+)

```
I32 ret;
```

```
ret = APS_manual_pulser_start(0, 1); //启用脉冲发生器输入。
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
ret = APS_manual_pulser_start( BoardID, 0 ); //禁用脉冲发生器进程
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_MODE, 2 );
```

```
//设置输入模式: 0: 1xAB; 2: 4xAB
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_LOGIC, 0 );
```

```
//设置逻辑: 0: InvPA = 0, InvPB = 0
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_DIR, 0 );
```

```
//设置方向: 0: InvPA = 0, InvPB = 0
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_RATIO_VALUE, 1 ); // 设定比例
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_ACC, 123456 ); // 设置加速度
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_JERK, 12345678 ); //设置 Jerk
```

```
ret = APS_manual_pulser_velocity_move( Axis, 12345 ); // 开始速度运动
```

```
ret = APS_manual_pulser_start( BoardID, 1 ); //启用脉冲发生器
```

还可以看看:

[APS_manual_pulser_velocity_move\(\)](#)

APS_manual_pulser_velocity_move	开始脉冲发生器速度运动/在手动脉冲发生器操作中开始速度运动
---------------------------------	-------------------------------

支持的产品:PCIe-8154/58, PCI-C154(+), PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

对于PCIe-8154/58 , PCI-C154 (+) , 此函数用于启动脉冲发生器的速度运动。当从脉冲发生器输入接收到一个具有默认值的脉冲时 , 轴将输出一个脉冲 , 用户可以通过轴参数164h和165h设置输出脉冲与输入脉冲的比率。用户发出停止移动命令时 , 轴可能会停止脉冲发生器函数。

用户可以为脉冲发生器函数指定一个有限的速度。例如 , 如果将 SpeedLimit 设置为 100 pps , 则即使输入脉冲发生器的信号速率大于 100 pps , 轴也会以最快的 100 pps 的速度运动。

当脉冲发生器运动函数生效时 , 会出现以下几种运动状态 :

- 1.当脉冲发生器函数生效后 , 运动状态的 bit 29 (PAPB) 将打开。表示等待来自 PA/PB 输入的信号。
- 2.当从 PA/PB 引脚输入连续信号时 , 运动状态的 bit 10 (VS) 将打开。表示根据用户指定速度的脉冲发生器输入作为轴的输出脉冲。

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x , 此函数将在手动脉冲发生器操作中开始速度运动。

句法:

对于 PCIe-8154/58, PCI-C154(+):

C/C++:

```
I32 FNTYPE APS_manual_pulser_velocity_move( I32 Axis_ID, F64 SpeedLimit );
```

Visual Basic:

```
APS_manual_pulser_velocity_move ( ByVal Axis_ID As Long, ByVal SpeedLimit As Double) As Long
```

对于 PCI-8254/58 / AMP-204/8C:

```
I32 FNTYPE APS_manual_pulser_velocity_move (I32 Axis_ID, F64 MaxVelocity)
```

```
APS_manual_pulser_velocity_move ( ByVal Axis_ID As Long, ByVal MaxVelocity As Double) As Long
```

参数:

对于 PCIe-8154/58, PCI-C154(+):

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 SpeedLimit: 此运动曲线的最大限制速度。单位 : 脉冲/秒

对于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x:

I32 Axis_ID: 轴编号为 0-7

F64 MaxVelocity: 手动脉冲发生器操作中的最大速度。该值应大于零。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 1:

以下示例适用于 PCIe-8154/58, PCI-C154(+)

I32 ret;

```
ret = APS_manual_pulser_start (0, 1 ); //启用脉冲输入。
```

```
ret =APS_manual_pulser_velocity_move (Axis_ID, 1000 ); // 开始脉冲发生器速度运动
```

示例 2:

以下示例适用于 PCI-8254/58 / AMP-204/8C 或 PCIe-833x

```
ret = APS_manual_pulser_start( BoardID, 0 ); // 禁用脉冲发生器进程
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_MODE, 2 );
```

```
// 设置输入模式 : 0: 1xAB; 2: 4xAB
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_LOGIC, 0 );
```

```
//设置逻辑: 0: InvPA = 0, InvPB = 0
```

```
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_DIR, 0 );
```

```
// 设置方向: 0: InvPA = 0, InvPB = 0
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_RATIO_VALUE, 1 ); // 设置比例
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_ACC, 123456 ); // 设置加速度
```

```
ret = APS_set_axis_param_f( Axis, PRA_PSR_JERK, 12345678 ); // 设置 jerk
```

```
ret = APS_manual_pulser_velocity_move( Axis, 12345 ); // 开始速度运动
```

```
ret = APS_manual_pulser_start( BoardID, 1 ); // 启用脉冲发生器
```

还可以看看:

[APS_manual_pulser_start\(\)](#)

APS_manual_pulser_relative_move	开始一个脉冲相对距离运动
---------------------------------	--------------

支持的产品:PCIe-8154/58, PCI-C154(+)

描述:

此函数用于开始一个脉冲发生器的相对运动。当从输入默认值的脉冲发生器接收到一个脉冲时，该轴将输出一个脉冲，用户可以通过轴参数164h和165h设置输出脉冲与输入脉冲之比。

当用户发出停止运动命令时，该轴可能会停止脉冲发生器函数。

用户可以为脉冲发生器函数指定一个有限的速度。例如，如果将 SpeedLimit 设置为 100 pps，那么即使输入脉冲发生器的信号速率超过 100 pps，轴也只能以 100 pps 的最快速度进行运动。

当脉冲发生器函数生效时，会出现以下几种运动状态：

- 1.当脉冲发生器函数生效后，运动状态的 bit 29 (PAPB) 将打开，即表示等待来自 PA/PB 输入的信号。
- 2.当从 PA/PB 引脚输入连续信号时，运动状态的 bit 10(VS)将打开，表示根据用户指定速度的脉冲发生器输入作为轴的输出脉冲。

句法:

C/C++:

```
I32 FNTYPE APS_manual_pulser_relative_move( I32 Axis_ID, F64 Distance, F64 SpeedLimit );
```

Visual Basic:

```
APS_manual_pulser_relative_move ( ByVal Axis_ID As Long, ByVal Distance As Double, ByVal SpeedLimit As Double) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Distance: 相对距离，以脉冲为单位。

I32 SpeedLimit: 此运动曲线的最大限制速度。单位：脉冲/秒。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
ret = APS_manual_pulser_start (0, 1); //启用脉冲发生器输入。  
ret =APS_manual_pulser_relative_move (Axis_ID, 100, 1000); //开始脉冲发生器的相对运动
```

APS_manual_pulser_home_move	开始一个脉冲归零运动
-----------------------------	------------

支持的产品:PCIe-8154/58, PCI-C154(+)

描述:

此函数用于开始一个脉冲发生器的归零运动。当从输入默认值的脉冲发生器接收到一个脉冲时，该轴将输出一个脉冲，用户可以通过轴参数 164h 和 165h 设置输出脉冲与输入脉冲之比。当用户发出停止运动命令时，该轴可能会停止脉冲发生器函数。

对于脉冲发生器的归零函数，用户可以参考“轴参数表”来设置指定的归零类型(166h)和归零极限速度(167h)。

用户可以为脉冲发生器函数指定一个有限的速度。例如，如果将 SpeedLimit 设置为 100 pps，那么即使输入脉冲发生器的信号速率超过 100 pps，轴也只能以 100 pps 的最快速度进行运动。

当脉冲发生器运用函数生效时，会出现以下几种运动状态：

- 1.当脉冲发生器函数生效后，运动状态的 bit 29 (PAPB) 将打开，即表示等待来自 PA/PB 输入的信号。
- 2.当从 PA/PB 引脚输入连续信号时，运动状态的 bit 10(VS) 将打开，表示根据用户指定速度的脉冲发生器输入作为轴的输出脉冲。

句法:

C/C++:

```
I32 FNTYPE APS_manual_pulser_home_move( I32 Axis_ID );
```

Visual Basic:

```
APS_manual_pulser_home_move ( ByVal Axis_ID As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
ret = APS_manual_pulser_start (0, 1); //启用脉冲发生器输入。
```

```
ret =APS_manual_pulser_home_move (Axis_ID); // 开始一个脉冲发生器的归零运动
```

APS_get_pulser_counter	获取脉冲发生器计数器
------------------------	------------

支持的产品:PCI-8253/56, DPAC-1000, DPAC-3000

描述:

此函数用于获取脉冲发生器的计数器值。Pulser 是手动脉冲发生器的缩写。它是用于手动生成工业计数器脉冲的设备。该设备有时称为“手轮”。

句法:

C/C++:

```
I32 FNTYPE APS_get_pulser_counter( I32 Board_ID, I32 *Counter );
```

Visual Basic:

```
APS_get_pulser_counter( ByVal Board_ID As Long, Counter As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 *Counter: 返回脉冲发生器计数器的值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Counter;
```

```
ret = APS_get_pulser_counter(0, &Counter );
if( ret == ERR_NoError )
    //显示计数器值。
```

APS_set_pulser_counter	设置 DPAC 脉冲输入计数器
------------------------	-----------------

支持的产品:DPAC-1000, DPAC-3000

描述:

对于 DPAC , 此函数用于设置输入脉冲计数器的编号。

句法:

C/C++:

I32 FNTYPE APS_set_pulser_counter (I32 Board_ID, I32 Counter);

Visual Basic:

APS_set_pulser_counter (ByVal Board_ID As Long, ByVal Counter As Long) As Long

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Counter: 输入脉冲计数器的编号。

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

```
I32 ret;  
I32 Counter;
```

```
Counter = 0; //设置输入脉冲计数器= 0  
ret = APS_set_pulser_counter (0, Counter );  
if( ret == ERR_NoError )  
    //显示计数器值.
```

还可以看看:

[APS_get_pls_ipcounter\(\)](#)

21. 螺距误差补偿功能

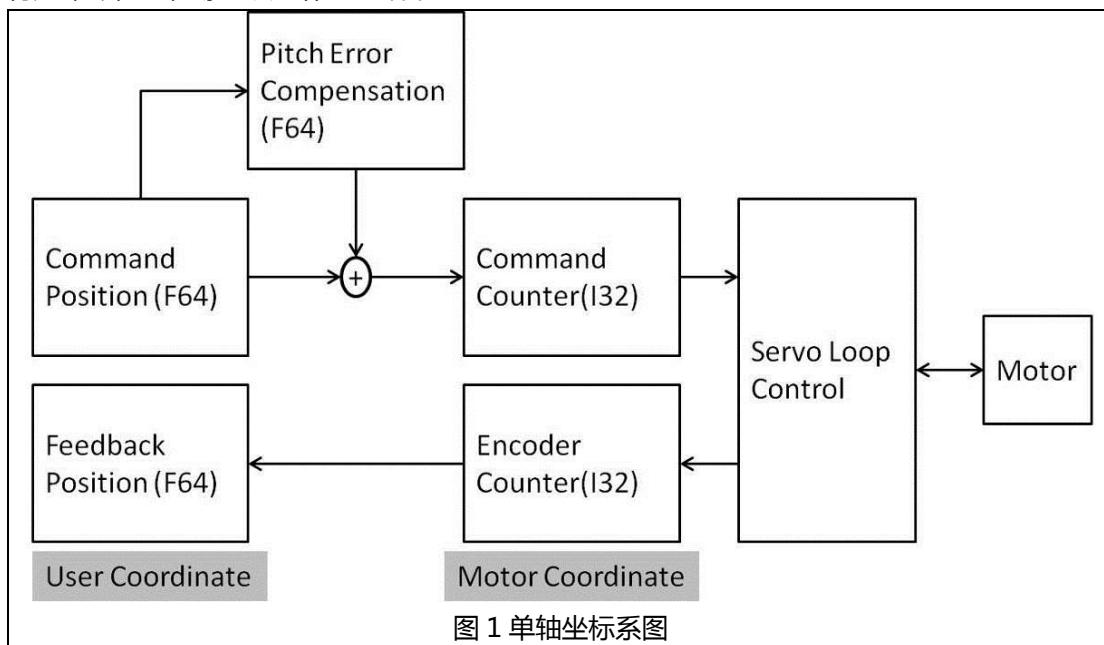
APS_set_pitch_table

设定螺距误差补偿表的数据。

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

图1介绍了PCI-8254/8中的两个坐标系：一个是用户坐标，另一个是电机坐标。在用户坐标中允许为命令位置和反馈位置指定任意值。通常，机器返回原点后，两个位置都将立即被设置为零。另一方面，由于它们用于伺服环路控制，因此禁止用户更改电动坐标中的命令计数器和编码器计数器。另外请注意，这两个坐标的数据类型分别是double和integer。图1中介绍的是螺距误差补偿的过程。它的输入是当前命令位置，从查询表获得的输出是误差补偿。命令计数器的实际值将是命令位置和螺距误差补偿的结果。



螺距误差补偿数据以每个轴指定的间隔作为每个补偿位置。补偿的原点是机器返回的原始位置。补偿数据为有符号值，并相对于原始位置进行设置（通常，原始位置的补偿数据为零）。为了进行螺距误差补偿，还必须设置最小位置、补偿位置间隔和总点数等配置。基于这些配置和补偿数据，可以成功建立螺距误差补偿表。用户的补偿方式有两种：常数型和线性型。完成以上所有设置后，用户可以通过APS函数启用螺距误差补偿。应该注意的是，如果机器行程在正方向或负方向上超出了指定范围，则螺距误差补偿将不会超出该范围，因此补偿值将为零。螺距误差补偿中使用的单位是脉冲（计数）。图2是螺距误差补偿表的示例。实线和虚线表示两种补偿类型。用户指定最小位置、间隔和总点数分别为0、100和5。因此最大位置将为500。每个补偿位置的补偿数据分别为0、1、2、-1和1。如果命令位置超出0到500的范围，则不进行补偿。机器返回原点（命令位置为零）后，误差补偿也为零。

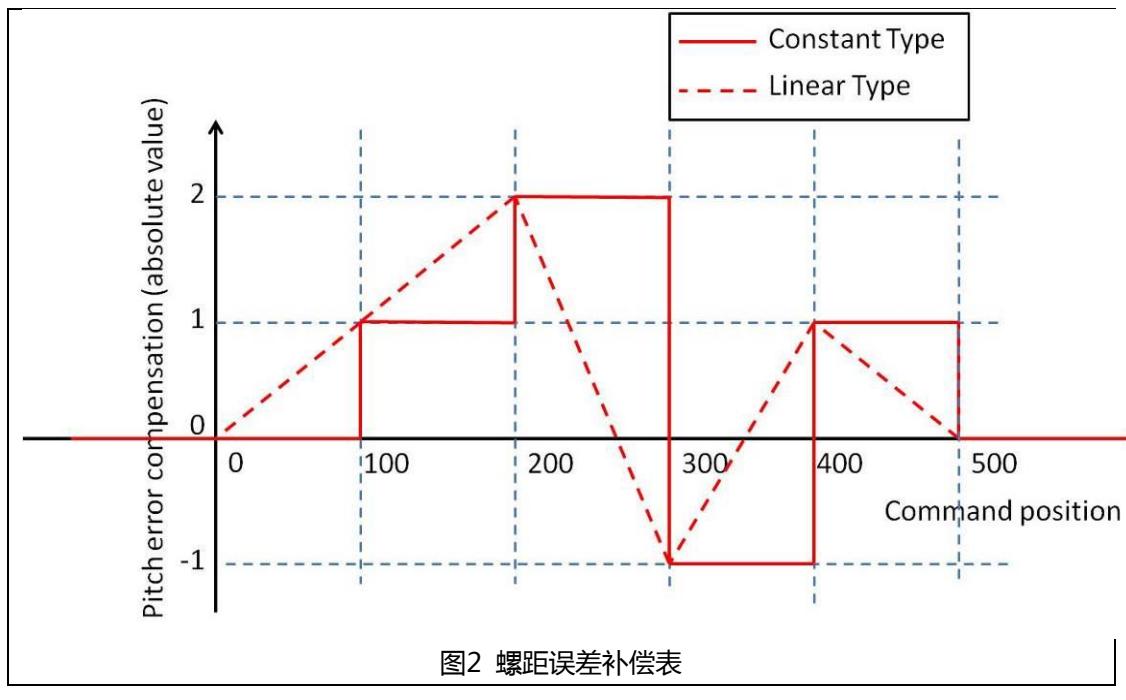


图2 螺距误差补偿表

表1显示了使用图2中的螺距误差补偿表的结果。例如，如果命令位置被设置为150.0，则对于常数补偿类型，命令计数器将为151；否则，计数器将变为152。相反，对于线性补偿类型，命令计数器将为152。这是由于截断错误引起的。

	Command position(F64)	0.0	50.0	100.0	150.0	200.0	250.0	300.0	350.0	400.0	450.0	500.0
Constant compensation	Error Compensation (F64)	0.0	0.0	1.0	1.0	2.0	2.0	-1.0	-1.0	1.0	1.0	0.0
	Command counter(I32)	0	50	101	151	202	252	299	349	401	451	500
Linear compensation	Error Compensation (F64)	0.0	0.5	1.0	1.5	2.0	0.5	-1.0	0.0	1.0	0.5	0.0
	Command counter (I32)	0	51	101	152	202	251	299	350	401	451	500

表1 使用不同补偿方法的结果

句法:

C/C++:

```
I32 FNTYPE APS_set_pitch_table( I32 Axis_ID, I32 Comp_Type, I32 Total_Points, I32
MinPosition, U32 Interval, I32 *Comp_Data);
```

Visual Basic:

```
APS_set_pitch_table (ByVal Axis_ID As Long, ByVal Comp_Type As Long, ByVal
Total_Points As Long, ByVal MinPosition As Long, ByVal Interval As UInteger, ByVal
Comp_Data() As UInteger) As Long
```

参数:

I32 Axis_ID: 0~7

I32 Comp_Type: 补偿类型； 0：常数补偿； 1：线性补偿

I32 Total_Points: 补偿数据总数；最大值为 500。

I32 MinPosition: 螺距误差补偿表中的最小位置

U32 Interval: 螺距误差补偿表中两个补偿点之间的间隔

I32 *Comp_Data: 螺距误差补偿表中的补偿数据

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
/* -----
程序名称：螺距误差补偿演示
作者:      Wei-li Chuang
日期:      2014/09/15
-----*/
void main()
{
    I32 ret; // 函数返回
    I32 BoardID_InBits; // 初始化
    I32 Axis_ID = 0; //轴 ID
    I32 MotionStatus; //运动状态 (以bit为单位 )
    I32 Comp_Data[5] = {0, 1, 2, -1, 1}; //螺距误差补偿数据
    I32 CommandPosition; // 命令位置
    I32 CommandCount; // 命令计数器
    I32 Comp_Type = 0; // 补偿类型
    I32 Total_Points = 5; // 总点数
    I32 MinPosition = 0; // 最小命令位置
    U32 Interval = 100; // 间隔
    I32 i;

    printf("/*Start pitch error compensation demo */ \n");

    // 初始化
    ret = APS_initial( &BoardID_InBits, 0 );
    if(ret)
    {
        printf("APS library initial fail! \n");
        goto END_PROGRAM;
    }

    //开启伺服
```

```

ret = APS_set_servo_on( Axis_ID, 1 );
if(ret)
{
    printf("Set servo on fail! \n");
    goto END_PROGRAM;
}

//获取当前命令位置和命令计数器
APS_get_command(Axis_ID, &CommandPosition );
APS_get_command_counter(Axis_ID, &CommandCount );
printf("Command position = %d Command count = %d \n",CommandPosition,
CommandCount );

//开始归零进程
printf("Return to home position... \n");
ret = APS_home_move( Axis_ID );
if(ret)
{
    printf("Start home fail! \n");
    goto END_PROGRAM;
}

//检查归零完成
do{
    MotionStatus = APS_motion_status( Axis_ID ); //获取运动状态
}while ( ( MotionStatus>>5 & 0x1 ) == 0 );

//在原点位置获取命令位置和命令计数器
APS_get_command(Axis_ID, &CommandPosition );
APS_get_command_counter(Axis_ID, &CommandCount );
printf("Command position = %d Command count = %d \n",CommandPosition,
CommandCount );

//设定螺距误差补偿表
ret = APS_set_pitch_table( Axis_ID, Comp_Type, Total_Points, MinPosition,
Interval, Comp_Data);
if(ret)
{
    printf("Set pitch error compensation data and configuration fail! \n");
    goto END_PROGRAM;
}

```

```

}

// 开始螺距误插补偿
ret = APS_start_pitch_comp( Axis_ID, 1 );
if(ret)
{
    printf("开始螺距误插补偿 fail! \n");
    goto END_PROGRAM;
}

//启动PTP以测试螺距误差补偿
for( i=0; i<Total_Points; i++ )
{
    ret = APS_absolute_move( Axis_ID, 100+i*100, 10000 );
    if(ret)
    {
        printf("Start PTP fail! \n");
        goto END_PROGRAM;
    }
}

//检查PTP是否完成
do{
    MotionStatus = APS_motion_status( Axis_ID ); //获取运动状态
}while ( ( MotionStatus>>5 & 0x1 ) == 0 );

//获取命令位置和命令计数器
APS_get_command(Axis_ID, &CommandPosition );
APS_get_command_counter(Axis_ID, &CommandCount );
printf("Command position = %d Command count = %d
\n",CommandPosition, CommandCount );
}

//关闭伺服
ret = APS_set_servo_on( Axis_ID, 0 );
if(ret)
{
    printf("Set servo off fail! \n");
    goto END_PROGRAM;
}

```

```
//停止螺距误差补偿
ret = APS_start_pitch_comp( Axis_ID, 0 );
if(ret)
{
    printf("Stop pitch error compensation fail! \n");
    goto END_PROGRAM;
}

END_PROGRAM:
printf("/* Stop pitch error compensation demo */ \n");
system("pause");
```

还可以看看：

APS_get_pitch_table();APS_start_pitch_comp()

APS_get_pitch_table	获取螺距误差补偿表的数据。
---------------------	---------------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

该函数用于获取螺距误差补偿表的配置和数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_pitch_table( I32 Axis_ID, I32 *Comp_Type, I32 *Total_Points, I32  
*MinPosition, U32 *Interval, I32 *Comp_Data);
```

Visual Basic:

```
APS_get_pitch_table (ByVal Axis_ID As Long, ByRef Comp_Type As Long, ByRef  
Total_Points As Long, ByRef MinPosition As Long, ByRef Interval As UInteger, ByRef  
Comp_Data As UInteger) As Long
```

参数:

I32 Comp_Type: 补偿类型 ; 0 : 常数补偿 ; 1 : 线性补偿

I32 *Total_Points: 补偿数据总数。

I32 MinPosition: 螺距误差补偿表中的最小位置

U32 *Interval: 螺距误差补偿表中两个补偿点之间的间隔

I32 *Comp_Data: 螺距误差补偿表中的补偿数据

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

请参阅 APS_set_pitch_table 中的示例程序

还可以看看:

APS_set_pitch_table();APS_start_pitch_comp()

APS_start_pitch_comp	开始螺距误插补偿
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

该函数用于开始螺距误插补偿表。

句法:

C/C++:

```
I32 FNTYPE APS_start_pitch_comp( I32 Axis_ID, I32 Enable );
```

Visual Basic:

```
APS_start_pitch_comp (ByVal Axis_ID As Long, ByVal Enable As Long) As Long
```

参数:

I32 Axis_ID: 0~7

I32 Enable: 0 : 禁用错误补偿 : 1 : 启用错误补偿

返回值:

I32 Error code: 请参考 [APS 函数返回代码.](#)

示例 :

请参阅 APS_set_pitch_table 中的示例程序

还可以看看:

APS_set_pitch_table();APS_get_pitch_table()

22. DPAC 系统函数

APS_rescan_CF	重新扫描 DPAC 从站 CF 插槽
---------------	--------------------

支持的产品:DPAC-1000, DPAC-3000

描述:

此函数用于重新扫描 DPAC 外部的 CF 插槽。在 Windows 中启动系统时，右下角有一个图标，用于管理 USB 闪存等可移动设备。如果用户从管理图标中移除了 DPAC 的外部 CF 卡（即 USB 设备），再插上一个 CF 卡，那么系统将无法对其进行重新扫描。 用户必须调用此函数才能激活重新扫描。

句法:

C/C++:

```
I32 FNTYPE APS_rescan_CF ( I32 Board_ID );
```

Visual Basic:

```
APS_rescan_CF ( ByVal Board_ID As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
ret = APS_rescan_CF ( 0 );  
if( ret != ERR_NoError )  
{  
    // 错误，显示消息。  
}
```

还可以看看:

APS_get_battery_status	获取 DPAC SRAM 电池状态
------------------------	-------------------

支持的产品:DPAC-1000, DPAC-3000

描述:

此函数用于获取 DPAC SRAM 的电池状态。DPAC 上有一个 SRAM，供用户快速存储。如果电池安装在 DPAC 上，那么 SRAM 可以是一个非易失性存储。 用户可以使用此函数来了解电池的状态。请注意，如果 DPAC 上未安装电池，则此函数将使电池返回高电平状态，但 SRAM 实际上不具有非易失性存储功能。请先检查电池是否存在。

句法:

C/C++:

```
I32 FNTYPE APS_get_battery_status( I32 Board_ID, I32 *Battery_status);
```

Visual Basic:

```
APS_get_battery_status( ByVal Board_ID As Long, Battery_status As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 *Battery_status: 1: 电量正常, 0: 电量低

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Battery_status;
ret = APS_get_battery_status ( 0, &Battery_status );
if( ret == ERR_NoError )
{
    //显示电池状态。
}
```

还可以看看:

APS_get_display_data	获取 7 段 LED 显示数据
----------------------	-----------------

支持的产品:DPAC-1000, DPAC-3000

描述:

此函数用于获取 7 段 LED 的数据。 DPAC LED 上有五位数字，每个数字可以显示一个字符。如果字符是数字，则它也可以显示一个字符和一个附加的点号。

句法:

C/C++:

```
I32 FNTYPE APS_get_display_data( I32 Board_ID, I32 displayDigit, I32 *displayIndex);
```

Visual Basic:

```
APS_get_display_data ( ByVal Board_ID As Long, ByVal displayDigit As Long,  
displayIndex As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 displayDigit: 7 段编号. (1~5)

I32 * displayIndex: 参考 DPAC 显示索引表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 displayNum;  
ret = APS_get_display_data( 0, 1, &displayNum );  
if( ret == ERR_NoError )  
{  
    //  displayNum 变量显示一个显示编号。  
}
```

还可以看看:

[APS_set_display_data\(\)](#);DPAC 显示索引表()

APS_set_display_data	设置 7 段 LED 显示数据
----------------------	-----------------

描述:

此函数用于设置 7 段 LED 的数据和显示。DPAC LED 上有五位数字，每个数字可以显示一个字符。如果字符是数字，则它也可以显示一个字符和一个附加的点号。

句法:

C/C++:

```
I32 FNTYPE APS_set_display_data( I32 Board_ID, I32 displayDigit, I32 displayIndex);
```

Visual Basic:

```
APS_set_display_data ( ByVal Board_ID As Long, ByVal displayDigit As Long, ByVal  
displayIndex As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

displayDigit: 7 段编号。(1~5)

I32 displayIndex: 参考显示索引表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 displayNum;
```

```
displayNum=0x01;  
ret = APS_set_display_data ( 0, 1, displayNum ); // 指示灯的第一位将显示 “ 1 ”  
if( ret != ERR_NoError )  
{  
    // 错误 , 显示消息。  
}
```

还可以看看:

[APS_get_display_data\(\)](#)

APS_get_button_status	获取按钮输入状态
-----------------------	----------

描述:

此函数用于获取 DPAC 的按钮状态。 DPAC 上有 4 个按钮，每个按钮都是单击类型。 , 这意味着当您松开按钮时，按钮将返回其原始位置。

句法:

C/C++:

```
I32 FNTYPE APS_get_button_status ( I32 Board_ID, I32 *buttonstatus);
```

Visual Basic:

```
APS_get_button_status ( ByVal Board_ID As Long, buttonstatus As Long ) As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 *buttonstatus: 参考按钮状态表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 buttonstatus;
ret = APS_get_button_status ( 0, &buttonstatus );
if( ret == ERR_NoError )
{
    //显示按钮状态。
}
Else
{
    "检查 B3 的开/关"
    1) 读取按钮状态
    2) 通过 “不” 按钮状态获取新的按钮状态
    3) 通过 “ Bit# = ( 4 - B# ) ” 将 B3 映射到 Bit# 。 我们得到 Bit1。
    4) 使用 Bit1 ( 0010b ) 来 “AND” 新按钮状态
    5) 如果结果为零，则表示未推动 B3。
    6) 如果结果不为零，则表示已推送 B3。
}
```

还可以看看:

DPAC push button status table()

23. 非易失性存储器

APS_set_nv_ram	设置 NVRAM 数据
----------------	-------------

支持的产品:DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856

描述:

此函数用于将值写入 NVRAM。 NVRAM 是一种非易失性存储器。即使系统电源关闭，它也可以永久地存储用户的数据。

PCI-8144 使用 EEPROM 作为 NVRAM。 它可以确保 1,000,000 次写入。

句法:

C/C++:

```
I32 FNTYPE APS_set_nv_ram( I32 Board_ID, I32 RamNo, I32 DataWidth, I32 Offset, I32 Data );
```

Visual Basic:

```
APS_set_nv_ram ( ByVal Board_ID As Long, ByVal RamNo As Long, ByVal DataWidth As Long, ByVal Offset As Long, ByVal Data As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 RamNo: RamNo=0(DPAC,PCI(e)-7856)

I32 DataWidth: 0: RW_WIDTH_8; 1: RW_WIDTH_16; 2: RW_WIDTH_32(PCI(e)-7856 Only)

I32 Offset: 从0x0000到0x75FF (DPAC)的偏移量。从0x0000到0x7FFF的偏移量 (PCI(e)-7856)

I32 Data: DataWidth: 0 数据从-128 到 127。 (DPAC, PCI(e)-7856)

 DataWidth: 1 数据从-32768 到 32767。 (DPAC, PCI(e)-7856)

 DataWidth: 2 数据从-2147483648 到 2147483647。 (仅 PCI(e)-7856)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I16 Data;
```

```
Data=0x5168;
```

```
ret = APS_set_nv_ram (0, 0, 1, 0x1000,Data );
```

```
//写入 RAM (offset =0x1000) value=0x5168. DataWidth: 1
```

```
if( ret != ERR_NoError )
```

```
{  
    // 错误，显示消息。  
}
```

还可以看看：

APS_get_nv_ram()

APS_get_nv_ram	获取 NVRAM 数据
支持的产品:DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856	

描述:

此函数用于从 NVRAM 中读取值。NVRAM 是一种非易失性存储器。它可以永久存储用户的数据。这就意味着即使系统电源关闭，数据仍会保存在内存中。下次系统恢复时，用户可以通过此函数获取数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_nv_ram( I32 Board_ID, I32 RamNo, I32 DataWidth, I32 Offset, I32
*Data );
```

Visual Basic:

```
APS_get_nv_ram ( ByVal Board_ID As Long, ByVal RamNo As Long, ByVal DataWidth
As Long, ByVal Offset As Long, Data As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 RamNo: RamNo=0(DPAC,PCI(e)-7856)

I32 DataWidth: 0: RW_WIDTH_8; 1: RW_WIDTH_16; 2: RW_WIDTH_32(PCI(e)-7856
Only)

I32 Offset: 从 0x0000 到 0x75FF(DPAC)的偏移量。从 0x0000 到 0x7FFF 的偏移量(PCI(e)-
7856)

I32 *Data: Datawidth: 0 数据从-128 到 127。 (DPAC,PCI(e)-7856)

 DataWidth: 1 数据从-32768 到 32767。 (DPAC,PCI(e)-7856)

 DataWidth: 2 数据从-2147483648 到 2147483647。 (仅 PCI(e)-7856)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
I32 Data;
```

```
ret = APS_get_nv_ram (0, 0, 1, 0x1000,&Data );
if( ret == ERR_NoError )
    //显示 RAM (offset =0x1000) DataWidth: 1 value.
```

还可以看看:

[APS_set_nv_ram\(\)](#)

APS_clear_nv_ram	清除 NVRAM 数据
------------------	-------------

支持的产品:DPAC-1000, DPAC-3000, PCI-8144, PCI(e)-7856

描述:

此函数用于清除 NVRAM 上的所有值。NVRAM 是一种非易失性存储器，这就意味着即使系统电源关闭，它也可以永久存储用户的数据。该函数生效时，将清除该存储器中存储的所有数据。

PCI-8144 使用 EEPROM 作为 NVRAM。 它可以确保 1,000,000 次写入。

句法:

C/C++:

```
I32 FNTYPE APS_clear_nv_ram( I32 Board_ID, I32 RamNo );
```

Visual Basic:

```
APS_clear_nv_ram ( ByVal Board_ID As Long, ByVal RamNo As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 RamNo: RamNo=0(DPAC,PCI(e)-7856)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
ret = APS_clear_nv_ram ( 0, 0 ); //清楚 RamNo=0 数据
if( ret != ERR_NoError )
{
    // 错误，显示消息。
}
```

还可以看看:

[APS_set_nv_ram\(\);APS_get_nv_ram\(\);APS_clear_nv_ram\(\)](#)

24. 现场总线比较触发

APS_set_field_bus_trigger_param	设置比较触发相关参数
---------------------------------	------------

支持的产品:MNET-4XMO-C, HSL-4XMO

描述:

此函数用于设置与触发器相关的比较参数。触发器参数表中描述了触发器参数的所有定义。

您也可以使用“APS_get_field_bus_trigger_param ()”函数进行参数设置。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_trigger_param( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Param_No, I32 Param_Val );
```

Visual Basic:

```
APS_set_field_bus_trigger_param (ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Param_No As Long, ByVal Param_Val As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于HSL现场总线，模块编号的范围是1到63。在HSL中，Module_No是模块占用的第一个ID。

对于MNET现场总线，模块编号的范围是0到63。

I32 Param_No: 参数编号。参考触发器参数表。

I32 Param_Val: 参数值。参考触发器参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

请参考“APS_set_field_bus_trigger_linear”，“APS_set_field_bus_trigger_table”示例

还可以看看:

[APS_get_field_bus_trigger_param\(\)](#)

APS_get_field_bus_trigger_param	获取比较触发相关参数
---------------------------------	------------

支持的产品:MNET-4XMO-C, HSL-4XMO

描述:

此函数用于获取与触发器相关的参数。触发器参数表中描述了触发器参数的所有定义。

您也可以使用“APS_set_field_bus_trigger_param()”函数进行参数设置。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_trigger_param( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 Param_No, I32 *Param_Val );
```

Visual Basic:

```
APS_get_field_bus_trigger_param(ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal Param_No As Long, Param_Val As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号

对于 HSL 现场总线 , 模块编号的范围是 1 到 63。在 HSL 中 , Module_No 是模块占用的第一个 ID。

对于 MNET 现场总线 , 模块编号的范围是 0 到 63。

I32 Param_No: 参数编号。参考触发器参数表

I32 Param_Val: Return 参数值。参考触发器参数表

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_field_bus_trigger_param\(\)](#)

APS_set_field_bus_trigger_linear	设置线性比较函数
----------------------------------	----------

支持的产品:MNET-4XMO-C, HSL-4XMO

描述:

该函数用于设置线性比较函数。

当线性触发操作完成时，总的比较点将为：

对于 MNET-4XMO-C，总的比较点数=重复时间。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_trigger_linear( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 LCmpCh, I32 StartPoint, I32 RepeatTimes, I32 Interval );
```

Visual Basic:

```
APS_set_field_bus_trigger_linear(ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No As Long, ByVal LCmpCh As Long, ByVal StartPoint As Long, ByVal  
RepeatTimes As Long, ByVal Interval As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值：0~1

I32 MOD_No: 模块编号

对于 HSL 现场总线，模块编号的范围是 1 到 63。在 HSL 中，Module_No 是模块占用的第一个 ID。

对于 MNET 现场总线，模块编号的范围是 0 到 63。

I32 LCmpCh: 线性比较设置通道。

对于 MNET-4XMO-C，LCmpCh 的范围是 0 到 4。（LCmpCh 0~3 用作常规比较器，LCmpCh 4 用作高速比较器。）

I32 StartPoint: 开始线性比较点

I32 RepeatTimes: 触发重复次数。

对于 MNET-4XMO-C，间隔：31 位无符号值。（值：1~0x7fffffff）

I32 Interval: 出发间隔。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
```

```
I32 Bus_No = 1;
```

```
I32 Mod_No = 0;
```

```
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x0, 1 ); //将 CMP0 设置为  
线性类型 APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x10, 1 ); //将  
CMP0 设置为 TRG0 的来源 APS_set_field_bus_trigger_linear(BoardId, Bus_No, Mod_No,  
0, 1000, 100000, 100 ); //设置 CMP0 线性比较算法。  
// 起点= 1000 , 重复次数= 100000 , 间隔= 100。  
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x04, 1 ); //启用 CMP0  
// 触发操作...  
// 当完成触发操作时。  
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x04, 0 ); //禁用 CMP0
```

还可以看看:

APS_set_field_bus_trigger_table()

APS_set_field_bus_trigger_table	设置表比较函数
---------------------------------	---------

支持的产品:MNET-4XMO-C, HSL-4XMO

描述:

此函数用于配置指定的比较表。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_trigger_table( I32 Board_ID, I32 BUS_No, I32 MOD_No,  
I32 TCmpCh, I32 *DataArr, I32 ArraySize );
```

Visual Basic:

```
APS_set_field_bus_trigger_table( ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No, ByVal TCmpCh As Long, DataArr As Long, ByVal ArraySize As Long) As  
Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号

对于 HSL 现场总线 , 模块编号的范围是 1 到 63。在 HSL 中 , Module_No 是模块占用的第一个 ID。

对于 MNET 现场总线 , 模块编号的范围是 0 到 63。

I32 TCmpCh:指定的比较表编号。

对于 MNET-4XMO-C , TCmpCh 的范围是 0 到 3。 (TCmpCh 0~3 用于通用比较器。)

I32 *DataArr: 比较数据数组。

I32 ArraySize: 比较数据数组的大小。

对于 MNET-4XMO-C , 每个通道的最大大小= 8192。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
#定义 POINTS 5000
```

```
I32 BoardId = 0;
```

```
I32 Bus_No = 1;
```

```
I32 Mod_No = 0;
```

```
I32 ret;
```

```
I32 data[POINTS];
```

```
I32 i;
```

```
for( i = 0; i < POINTS; i++ )
    data[i] = 10 + (C) * 10;

APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x0, 0 ); //将 CMP0 设置为
表类型
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x10, 1 ); //将 CMP0 设置
为 TRG0 的来源
ret = APS_set_field_bus_trigger_table(BoardId, Bus_No, Mod_No, 0, data, POINTS );
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x04, 1 ); //启用 CMP0
// 触发操作...
// 当完成触发操作时。
APS_set_field_bus_trigger_param(BoardId, Bus_No, Mod_No, 0x04, 0 ); //禁用 CMP0
```

还可以看看:

APS_set_field_bus_trigger_linear()

APS_set_field_bus_trigger_manual	手动输出触发
----------------------------------	--------

支持的产品:MNET-4XMO-C

描述:

此函数用于在指定的触发输出通道上强制输出触发。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_trigger_manual( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 TrgCh );
```

Visual Basic:

```
APS_set_field_bus_trigger_manual( ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No, ByVal TrgCh As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 TrgCh: 触发输出通道 (TRG) 编号。从零开始。

对于 MNET-4XMO-C, TrgCh 的范围是 0 到 3。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;  
I32 Bus_No = 1;  
I32 Mod_No = 0;  
I32 ret;  
ret = APS_set_field_bus_trigger_manual(BoardId, Bus_No, Mod_No, 0); //TRG0
```

还可以看看:

[APS_set_field_bus_trigger_manual_s\(\)](#)

APS_set_field_bus_trigger_manual_s	手动同步输出触发
------------------------------------	----------

支持的产品:MNET-4XMO-C

描述:

此函数用于强制输出触发。

通过此函数，所有输出通道都可以同步触发输出。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_trigger_manual_s( I32 Board_ID, I32 BUS_No, I32
MOD_No, I32 TrgChInBit );
```

Visual Basic:

```
APS_set_field_bus_trigger_manual_s( ByValBoard_ID As Long, ByVal BUS_No As Long,
 ByVal MOD_No, ByValTrgChInBit As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 TrgChInBit: 1: 输出触发，0 : 不输出触发

对于 MNET-4XMO-C : Bit0: TRG0, Bit1: TRG1, Bit2: TRG2, Bit3: TRG3

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
I32 Bus_No = 1;
I32 Mod_No = 0;
I32 ret;
ret = APS_set_field_bus_trigger_manual_s(BoardId, Bus_No, Mod_No, 0xF ); //4 通道同
步输出触发。
Ret = APS_set_field_bus_trigger_manual_s(BoardId, Bus_No, Mod_No, 0x2 ); //TRG1 输
出触发。 Ret = APS_set_field_bus_trigger_manual_s( 0, 0x3 ); // TRG0 和 TRG1 同步
输出触发。
//...
```

还可以看看:

[APS_set_field_bus_trigger_manual\(\)](#)

APS_get_field_bus_trigger_table_cmp	获取当前表比较值
支持的产品: MNET-4XMO-C, HSL-4XMO	

描述:

此函数用于从指定的表比较器中获取当前的比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_trigger_table_cmp( I32 Board_ID, I32 BUS_No, I32
MOD_No, I32 TCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_field_bus_trigger_table_cmp(ByVal Board_ID As Long, ByVal BUS_No As Long,
ByVal MOD_No, ByVal TCmpCh As Long, CmpVal As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于HSL现场总线，模块编号的范围是1到63。在HSL中，Module_No是模块占用的第一个ID。

对于MNET现场总线，模块编号的范围是0到63。

I32 TCmpCh: 指定表比较器的通道编号。从零开始。

对于 MNET-4XMO-C , TCmpCh 的范围是 0 到 3。 (TCmpCh 0~3 用于通用比较器。)

I32 *CmpVal: 返回比较器中当前的比较值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
I32 Bus_No = 1;
I32 Mod_No = 0;
I32 ret;
I32 CmpVal;
ret = APS_get_field_bus_trigger_table_cmp (BoardId, Bus_No, Mod_No, 0, &CmpVal );
If( ret != ERR_NoError )
{ // 错误，显示消息。
}
```

还可以看看:

APS_get_field_bus_trigger_linear_cmp()

APS_get_field_bus_trigger_linear_cmp	获取当前线性比较值
--------------------------------------	-----------

支持的产品:MNET-4XMO-C, HSL-4XMO

描述:

此函数用于从指定的线性比较器中获取当前的比较值。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_trigger_linear_cmp( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 LCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_field_bus_trigger_linear_cmp(ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No, ByVal LCmpCh As Long, CmpVal As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于HSL现场总线，模块编号的范围是1到63。在HSL中，Module_No是模块占用的第一个ID。

对于MNET现场总线，模块编号的范围是0到63。

I32 LCmpCh: 指定线性比较器的通道编号。从零开始。

对于 MNET-4XMO-C , TCmpCh 的范围是 0 到 4。 (LCmpCh 0~3 用于通用比较器。 LCmpCh 4 用于高速比较器。)

I32 *CmpVal: 返回比较器中当前的比较值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;  
I32 Bus_No = 1;  
I32 Mod_No = 0;  
I32 ret;  
I32 CmpVal;  
ret = APS_get_field_bus_trigger_linear_cmp(BoardId, Bus_No, Mod_No, 0, &CmpVal );  
If( ret != ERR_NoError )  
{ // 错误，显示消息。  
}
```

还可以看看:

APS_get_field_bus_trigger_table_cmp()

APS_get_field_bus_trigger_count	获取触发计数.
--	---------

支持的产品:MNET-4XMO-C

描述:

该函数用于获取触发计数。

您可以使用此函数检查输出多少个触发脉冲。

使用 **APS_reset_field_bus_trigger_count()** 将计数器重置为零。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_trigger_count( I32 Board_ID, I32 BUS_No, I32 MOD_No,
I32 TrgCh, I32 *TrgCnt );
```

Visual Basic:

```
APS_get_field_bus_trigger_count(ByVal Board_ID As Long, ByVal BUS_No As Long,
ByVal MOD_No, ByVal TrgCh As Long, TrgCnt As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 TrgCh: 指定的触发输出计数器通道编号。从零开始。

对于 MNET 现场总线，TrgCh 的范围是 0 到 3。

I32 *TrgCnt: 返回触发计数器值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
I32 Bus_No = 1;
I32 Mod_No = 0;
I32 Ret;
I32 TrgCnt;
Ret = APS_get_field_bus_trigger_count(BoardId, Bus_No, Mod_No, 0, &TrgCnt );
If( ret != ERR_NoError )
{ // 错误，显示消息。
}
```

还可以看看:

APS_reset_field_bus_trigger_count()

APS_reset_field_bus_trigger_count	重置触发计数.
-----------------------------------	---------

支持的产品:MNET-4XMO-C

描述:

此函数用于将触发计数器重置为零。

句法:

C/C++:

```
I32 FNTYPE APS_reset_field_bus_trigger_count( I32 Board_ID, I32 BUS_No, I32  
MOD_No, I32 TrgCh );
```

Visual Basic:

```
APS_reset_field_bus_trigger_count( ByVal Board_ID As Long, ByVal BUS_No As Long,  
ByVal MOD_No, ByVal TrgCh As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 TrgCh: 触发计数器通道编号。从零开始。

对于 MNET-4XMO-C, TrgCh 的范围是 0 到 3。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;  
I32 Bus_No = 1;  
I32 Mod_No = 0;  
I32 ret;  
ret = APS_reset_field_bus_trigger_count(BoardId, Bus_No, Mod_No, 0 );  
ret = APS_reset_field_bus_trigger_count(BoardId, Bus_No, Mod_No, 1 );  
ret = APS_reset_field_bus_trigger_count(BoardId, Bus_No, Mod_No, 2 );  
ret = APS_reset_field_bus_trigger_count(BoardId, Bus_No, Mod_No, 3 );  
...
```

还可以看看:

APS_get_field_bus_trigger_count()

APS_get_field_bus_linear_cmp_remain_count	获取线性比较的剩余计数值
---	--------------

支持的产品:MNET-4XMO-C

描述:

该函数用于获取线性比较的剩余计数值。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_linear_cmp_remain_count( I32 Board_ID, I32 BUS_No,  
I32 MOD_No, I32 LCmpCh, I32 *Cnt );
```

Visual Basic:

```
APS_get_field_bus_linear_cmp_remain_count ( ByVal Board_ID As Long, ByVal BUS_No  
As Long, ByVal MOD_No, ByVal LCmpCh As Long, Cnt As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 LCmpCh: 指定线性比较器的通道编号。从零开始。

对于 MNET-4XMO-C , TCmpCh 的范围是 0 到 4。 (LCmpCh 0~3 用于通用比较器。

LCmpCh 4 用于高速比较器。)

I32 *Cnt: 剩余计数器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;  
I32 Bus_No = 1;  
I32 Mod_No = 0;  
I32 ret;  
I32 Cnt;  
ret = APS_get_field_bus_linear_cmp_remain_count (BoardId, Bus_No, Mod_No, 0,  
&Cnt );  
If( ret != ERR_NoError )  
{ // 错误，显示消息。  
}
```

还可以看看:

APS_get_field_bus_table_cmp_remain_count()

APS_get_field_bus_table_cmp_remain_count	获取表比较的剩余计数值
--	-------------

支持的产品:MNET-4XMO-C

描述:

该函数用于获取表比较的剩余计数值。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_table_cmp_remain_count( I32 Board_ID, I32 BUS_No,  
I32 MOD_No, I32 TCmpCh, I32 *Cnt );
```

Visual Basic:

```
APS_get_field_bus_table_cmp_remain_count ( ByVal Board_ID As Long, ByVal BUS_No  
As Long, ByVal MOD_No, ByVal TCmpCh As Long, Cnt As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 TCmpCh: 指定表比较器的通道编号。从零开始。

对于 MNET-4XMO-C , TCmpCh 的范围是 0 到 4。 (LCmpCh 0~3 用于通用比较器。
LCmpCh 4 用于高速比较器。)

I32 *Cnt: 剩余计数器。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;  
I32 Bus_No = 1;  
I32 Mod_No = 0;  
I32 ret;  
I32 Cnt;  
ret = APS_get_field_bus_table _cmp_remain_count (BoardId, Bus_No, Mod_No, 0,  
&Cnt );  
If( ret != ERR_NoError )  
{ // 错误，显示消息。  
}
```

还可以看看:

APS_get_field_bus_linear_cmp_remain_count()

APS_get_field_bus_encoder	获取编码器计数值
支持的产品:MNET-4XMO-C	

描述:

此函数用于获取编码器计数值。

句法:

C/C++:

```
I32 FNTYPE APS_get_field_bus_encoder( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
EncCh, I32 *EncCnt );
```

Visual Basic:

```
APS_get_field_bus_encoder ( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No, ByVal EncCh As Long, EncCnt As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 EncCh: 指定编码器通道编号。从零开始。

对于 MNET-4XMO-C , EncCh 的范围是 0 到 4。 (LCmpCh 0~3 用于通用比较器。

LCmpCh 4 用于高速比较器。)

I32 * EncCnt: 编码器计数

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
I32 Bus_No = 1;
I32 Mod_No = 0;
I32 ret;
I32 EncCnt;
ret = APS_get_field_bus_encoder (BoardId, Bus_No, Mod_No, 0, & EncCnt);
If( ret != ERR_NoError )
{ // 错误 , 显示消息。
}
```

还可以看看:

[APS_set_field_bus_encoder\(\)](#)

APS_set_field_bus_encoder	设置编码器计数值
支持的产品:MNET-4XMO-C	

描述:

此函数用于设置编码器计数值。

句法:

C/C++:

```
I32 FNTYPE APS_set_field_bus_encoder( I32 Board_ID, I32 BUS_No, I32 MOD_No, I32
EncCh, I32 EncCnt );
```

Visual Basic:

```
APS_set_field_bus_encoder ( ByVal Board_ID As Long, ByVal BUS_No As Long, ByVal
MOD_No, ByVal EncCh As Long, ByVal EncCnt As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 BUS_No: 现场总线编号(端口号) 值 : 0~1

I32 MOD_No: 模块编号。

对于MNET现场总线，模块编号的范围是0到63。

I32 EncCh: 指定编码器通道编号。从零开始。

对于 MNET-4XMO-C , EncCh 的范围是 0 到 4。 (LCmpCh 0~3 用于通用比较器。
LCmpCh 4 用于高速比较器。)

I32 EncCnt: 编码器计数

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 BoardId = 0;
```

```
I32 Bus_No = 1;
```

```
I32 Mod_No = 0;
```

```
I32 ret;
```

```
ret = APS_set_field_bus_encoder (BoardId, Bus_No, Mod_No, 0, 0);
```

```
If( ret != ERR_NoError )
```

```
{ // 错误，显示消息。
```

```
}
```

还可以看看:

[APS_set_field_bus_encoder\(\)](#)

25. 看门狗定时器

APS_wdt_start	开启/停至看门狗定时器
---------------	-------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于开启/停至看门狗定时器.

句法:

C/C++:

```
I32 FNTYPE APS_wdt_start( I32 Board_ID, I32 TimerNo, I32 TimeOut );
```

Visual Basic:

```
APS_wdt_start (ByVal Board_ID As Long, ByVal TimerNo As Long, ByVal TimeOut As  
Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

I32 TimeOut:

设置为 0 禁用看门狗定时器。

将值设置为 N (1~100) 以启用看门狗定时器。

TimeOut = N * 100 ms

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
//启用看门狗定时器，超时时间为 2 秒。
```

```
ret = APS_wdt_start( boardId, 0, 20 );
```

还可以看看:

APS_wdt_get_timeout_period	获取看门狗定时器的超时时间
----------------------------	---------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于获取看门狗定时器的超时时间。如果超时时间为 0，则看门狗定时器被禁用。如果超时时间不为 0，则启用看门狗定时器。

句法:

C/C++:

```
I32 FNTYPE APS_wdt_get_timeout_period( I32 Board_ID, I32 TimerNo, I32 *TimeOut );
```

Visual Basic:

```
APS_wdt_get_timeout_period(ByVal Board_ID As Long, ByVal TimerNo As Long,  
TimeOut As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

I32 TimeOut:

0 表示看门狗定时器被禁用。

A value N(1 ~ 100) means that watch dog timer is enabled.

TimeOut = N * 100 ms

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;  
I32 boardId = 0;  
I32 timeOut = 0;
```

```
// 获取看门狗定时器的超时时间
```

```
ret = APS_wdt_get_timeout_period ( boardId, 0, &timeOut );
```

还可以看看:

APS_wdt_reset_counter	重置看门狗计时器的计数器
-----------------------	--------------

支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x

描述:

该函数用于重置看门狗计时器的计数器为零。计数器在每个 DSP 周期后加一。启用看门狗定时器后，用户可以定期重置看门狗定时器的计数器，以免触发动作事件。

句法:

C/C++:

```
I32 FNTYPE APS_wdt_reset_counter( I32 Board_ID, I32 TimerNo );
```

Visual Basic:

```
APS_wdt_reset_counter (ByVal Board_ID As Long, ByVal TimerNo As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
//重置看门狗计时器的计数器
```

```
ret = APS_wdt_reset_counter ( boardId, 0);
```

还可以看看:

APS_wdt_get_counter	获取看门狗计时器的计数器
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数获取看门狗计时器的计数器。如果启用，则计数器在每个 DSP 周期后加一。如果禁用，则计数器显示为零。

句法:

C/C++:

```
I32 FNTYPE APS_wdt_get_counter( I32 Board_ID, I32 TimerNo, I32 *Counter );
```

Visual Basic:

```
APS_wdt_get_counter (ByVal Board_ID As Long, ByVal TimerNo As Long, Counter As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

对于 PCI-8253/56 :

I32 Counter: 如果启用，则计数器在每个 DSP 周期后加一。如果禁用，则计数器显示为零。

对于 PCIe-833x :

I32 Counter: 如果启用，则计数器在每个系统周期时间后加一。如果禁用，则计数器显示为零。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
I32 boardId = 0;
I32 Counter = 0;
// 获取看门狗计时器的计数器
ret = APS_wdt_get_counter ( boardId, 0, &Counter );
```

还可以看看:

APS_wdt_set_action_event	设置看门狗定时器的动作事件
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于设置看门狗定时器的动作事件。如果超时，将触发动作事件。

句法:

C/C++:

```
I32 FNTYPE APS_wdt_set_action_event( I32 Board_ID, I32 TimerNo, I32 EventByBit );
```

Visual Basic:

```
APS_wdt_set_action_event (ByVal Board_ID As Long, ByVal TimerNo As Long, ByVal EventByBit As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

对于 PCI-8253/56：

I32 EventByBit: 设置事件

Bit0: 电机伺服关闭

Bit1: 数字输出关闭

Bit2: PWM 关闭

对于 PCIe-833x：

I32 EventByBit: 设置事件

Bit0: 所有轴均调用 EMG 停止功能

Bit1: 从站的所有数字输出均已关闭

Bit2: 从站的所有数字输出已打开

Bit3: 所有轴均调用伺服关闭功能

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
//设置动作事件。 如果超时，电机伺服将关闭。
```

```
ret = APS_wdt_set_action_event( boardId, 0, 1);
```

还可以看看:

APS_wdt_get_action_event	获取看门狗定时器的动作事件
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

该函数用于获取看门狗定时器的动作事件。

句法:

C/C++:

```
I32 FNTYPE APS_wdt_get_action_event( I32 Board_ID, I32 TimerNo, I32 *EventByBit );
```

Visual Basic:

```
APS_wdt_get_action_event (ByVal Board_ID As Long, ByVal TimerNo As Long,  
EventByBit As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 TimerNo: 指定计时器编号。

在 PCI-8254/58 或 PCIe-833x 中，计时器编号为 0。

对于 PCI-8253/56 :

I32 *EventByBit: 获取事件

Bit0: 电机伺服关闭

Bit1 : 数字输出关闭

Bit2 : PWM 关闭

对于 PCIe-833x :

I32 *EventByBit: 获取事件

Bit0 : 所有轴均调用 EMG 停止功能

Bit1 : 从站的所有数字输出均已关闭

Bit2 : 从站的所有数字输出已打开

Bit3 : 所有轴均调用伺服关闭功能

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
I32 EventByBit = 0;
```

```
// 获取看门狗定时器的动作事件
```

```
ret = APS_wdt_get_action_event( boardId, 0, &EventByBit );
```

还可以看看：

26. VAO/PWM 函数(激光函数)

APS_set_vao_param	将参数设置到 VAO 参数表中
支持的产品:PCI-8253/56, PCI-8254/58 / AMP-204/8C	

描述:

VAO 模块是一个激光控制应用。根据相应的线性速度提供模拟输出和 PWM 信号。

此函数用于设置 VAO 相关的参数。VAO 参数表中描述了 VAO 参数的所有定义。

您也可以使用 “APS_get_vao_param()” 函数获得 VAO 参数设置。

句法:

C/C++:

```
I32 FNTYPE APS_set_vao_param( I32 Board_ID, I32 Param_No, I32 Param_Val );
```

Visual Basic:

```
APS_set_vao_param (ByVal Board_ID As Long, ByVal Param_No As Long, ByVal  
Param_Val As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Param_No: 参数编号。请参考 VAO 参数表。

I32 Param_Val: 参数值。请参考 VAO 参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
//将电压模式的输出类型设置为 VAO 表 0。
```

```
ret = APS_set_vao_param(Board_ID, 0x00, 1);
```

还可以看看:

[APS_get_vao_param\(\)](#)

APS_get_vao_param	获取 VAO 表中的参数
支持的产品:PCI-8253/56, PCI-8254/58 / AMP-204/8C	

描述:

VAO 模块是一个激光控制应用。根据相应的线性速度提供模拟输出和 PWM 信号。

此函数用于设置 VAO 相关的参数。VAO 参数表中描述了 VAO 参数的所有定义。

您也可以使用 “APS_set_vao_param()” 函数获得 VAO 参数设置。

句法:

C/C++:

```
I32 FNTYPE APS_get_vao_param( I32 Board_ID, I32 Param_No, I32 *Param_Val );
```

Visual Basic:

```
APS_get_vao_param(ByVal Board_ID As Long, ByVal Param_No As Long, Param_Val As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Param_No: 参数编号。 请参考 VAO 参数表。

I32 Param_Val: 返回参数值。 请参考 VAO 参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Output_Type;
```

```
//获取 VAO 表 0 的输出类型。
```

```
ret = APS_set_vao_param(Board_ID, 0x00, &Output_Type );
```

还可以看看:

[APS_set_vao_param\(\)](#)

APS_set_vao_table	设置 VAO 表
支持的产品:PCI-8253/56, PCI-8254/58 / AMP-204/8C	

描述:

此函数用于设置一组 VAO 表。用户可以根据此表实现 VAO 应用。用户为激光应用配置相关的最小速度，速度间隔，总点数和映射输出值。因此，将构建“速度对能量”映射查找表。

请注意，在执行 APS_check_vao_param()时，将根据 VAO 输出类型检查映射输出值。如果映射输出值无效，则返回“ERR_ParameterInvalid”。

例如，如果将输出类型设置为电压模式，则映射输出电压不能大于 10000 mV。映射输出值的范围如下所述：

对于 PCI-8253/56:

输出类型(0~3)	输出范围
0: 电压	0 ~ 10000 mv 单位: 1 mv
1: PWM 模式	0 ~ 2000 (0.0% ~ 100%) 单位: 0.05%
2: 固定宽度的 PWM 频率模式	1 ~ 25M Hz 单位: 1 Hz
3: 具有固定占空比的 PWM 频率模式	1 ~ 25M Hz 单位: 1 Hz

对于 PCI-8254/58 / AMP-204/8C:

输出类型(0~3)	输出范围
0: 电压	0 ~ 10000 mv 单位: 1 mv
1: PWM 模式	0 ~ 2000 (0.0% ~ 100%) 单位: 0.05%
2: 固定宽度的 PWM 频率模式	3 ~ 50M Hz 单位: 1 Hz
3: 具有固定占空比的 PWM 频率模式	3 ~ 50M Hz 单位: 1 Hz

句法:

C/C++:

```
I32 FNTYPE APS_set_vao_table( I32 Board_ID, I32 Table_No, I32 MinVelocity, I32
VelInterval, I32 TotalPoints, I32 *MappingdataArray );
```

Visual Basic:

```
APS_set_vao_table ( ByVal Board_ID As Long , ByVal Table_No As Long, ByVal  
MinVelocity As Long, ByVal VelInterval As Long, ByVal TotalPoints As Long,  
MappingdataArray As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

Table_No: VAO 表编号。范围是 0~7。

I32 MinVelocity: 最小线性速度。

I32 VelInterval: 速度间隔。

I32 TotalPoints : 总点数。范围 1 ~ 32。

I32 *MappingdataArray: 输出数据组。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Minimum_Velocity;
```

```
I32 Velocity_Interval;
```

```
I32 TotalPoints = 32;
```

```
I32 OutputVoltageData[32];
```

```
//配置线性速度
```

```
//第 1 个速度 : 10000 , 第 2 个速度 : 20000 , ..... , 第 32 个速度 : 320000
```

```
Minimum_Velocity = 10000;
```

```
Velocity_Interval = 10000;
```

```
TotalPoints = 32;
```

```
//配置映射的输出电压
```

```
OutputVoltageData[0] = 500; // 第 1 个电压: 500 mv
```

```
OutputVoltageData[1] = 600; // 第 2 个电压: 600 mv
```

```
.....
```

```
OutputVoltageData[31] = 8600; // 第 32 个电压: 8600 mv
```

```
//设置 VAO 表 0 的映射表
```

```
Ret = APS_set_vao_table( Board_ID, 0, MinVelocity, VelInterval, TotalPoints,  
OutputVoltageData );
```

还可以看看:

APS_set_vao_param(); APS_get_vao_param(); APS_switch_vao_table(); APS_start_vao()

APS_set_vao_param_ex	通过 VAO 结构设置参数
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于通过 VAO 结构设置参数。 这是 APS_set_vao_param() 和 APS_set_vao_table() 的扩展。 通过调用 APS_set_vao_param_ex()，用户可以立即通过 VAO 结构设置所有参数。 通过调用 APS_set_vao_param()，用户可以一个一个地设置指定的参数。

此函数还可以用来设置映射表以替换 APS_set_vao_table()。 用户可以配置相关的最小速度、速度间隔、总点数以及用于激光应用的映射输出值。 然后，将构建“速度对能量”映射查找表。

请注意，APS_set_vao_param_ex() 可以取代 APS_set_vao_param() 和 APS_set_vao_table() 两个函数。可以选择其一。

句法:

C/C++:

```
I32 FNTYPE APS_set_vao_param_ex( I32 Board_ID, I32 Table_No, VAO_DATA*
VaoData );
```

Visual Basic:

```
APS_set_vao_param_ex (ByVal Board_ID As Integer, ByVal Table_No As Integer, ByRef
VaoData As VAO_DATA) As Integer
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Table_No: VAO 表编号。范围是 0~7。

VAO_DATA *VaoData: 用于设置所有参数的 VAO 结构。

```
Typedef struct _VAO_DATA
{
    //参数
    I32 outputType;    //输出类型, [0, 3]
    I32 inputType; //输入类型, [0, 1]
    I32 config; //根据输出类型进行 PWM 配置
    I32 inputSrc;      //轴输入源, [0, 0xf]

    //映射表
    I32 minVel; //最小线性速度, [ 正值 ]
    I32 VelInterval; //速度间隔, [正值]
    I32 totalPoints; //总点数, [1, 32]
```

```
I32 mappingDataArr[32]; //映射数据数组}
VAO_DATA, *PVAO_DATA;
```

对于 PCI-8253/56:

用于设置 VAO 参数的 VAO_DATA 结构定义

变量名	描述	值
outputType	表输出类型 (*1)	0 : 电压 1 : PWM 模式 2 : 固定宽度的 PWM 频率模式 3 : 固定占空比的 PWM 频率模式
inputType	表输入类型	0 : 反馈速度 1 : 命令速度
config	根据输出类型配置 PWM。	a. 模式 0 –不用关心 b. 模式 1 –设定固定频率 (1~25M Hz) c. 模式 2 –设置固定的脉冲宽度 (40~335544340 ns) d. 模式 3 –设置固定占空比： N * 0.05%。 (N : 1~2000)
inputSrc	指定 VAO 表的 axisID。 (多轴上的线速度) (*2)	Bit0: 轴 0 开 Bit1: 轴 1 开 Bit2: 轴 2 开 Bit3: 轴 3 开

(*1) : PCI-8253 不支持电压模式。

(*2): PCI-8253 支持 3 轴。 Bit 0 , Bit 1 和 Bit 2 可用。

用于设置 VAO 映射表的 VAO_DATA 结构定义

变量名	描述	值
minVel	最小线性速度	正值
velInterval	速度间隔	正值
totalPoints	总点数	1 ~ 32

mappingData Arr	映射数据数组	请参考下图
--------------------	--------	-------

VAO_DATA 结构的映射数据将根据 VAO 输出类型进行检查。如果映射数据无效，则返回“ERR_ParameterInvalid”。

例如，如果将输出类型设置为电压模式，则映射输出电压不能大于 10000 mV。映射输出值的范围如下所述：

输出类型(0~3)	输出范围
0: 电压	0 ~ 10000 mv 单位: 1 mv
1: PWM 模式	0 ~ 2000 (0.0% ~ 100%) 单位: 0.05%
2: 固定宽度的 PWM 频率模式	1 ~ 25M Hz 单位: 1 Hz
3: 具有固定占空比的 PWM 频率模式	1 ~ 25M Hz 单位: 1 Hz

对于 PCI-8254/58 / AMP-204/8C:

用于设置 VAO 参数的 VAO_DATA 结构定义

变量名	描述	值
outputType	表输出类型	0 : 电压 1 : PWM 模式 2 : 固定宽度的 PWM 频率模式 3 : 固定占空比的 PWM 频率模式
inputType	表输入类型	0 : 反馈速度 1 : 命令速度
config	根据输出类型配置 PWM。	e. 模式 0 –不用关心 f. 模式 1 –设定固定频率 (1~25M Hz) g. 模式 2 –设置固定的脉冲宽度 (40~335544340 ns) h. 模式 3 –设置固定占空比： N * 0.05%。 (N : 1~2000)

inputSrc	指定 VAO 表的 axisID。 (多轴上的线速度)	Bit0: 轴 0 开 Bit1: 轴 1 开 Bit2: 轴 2 开 Bit3: 轴 3 开
----------	-----------------------------	--

用于设置 VAO 映射表的 VAO_DATA 结构定义

变量名	描述	Value
minVel	最小线性速度	正值
velInterval	速度间隔	正值
totalPoints	总点数	1 ~ 32
mappingData Arr	映射数据数组	请参考下图

VAO_DATA 结构的映射数据将根据 VAO 输出类型进行检查。如果映射数据无效，则返回“ERR_ParameterInvalid”。

例如，如果将输出类型设置为电压模式，则映射输出电压不能大于 10000 mV。映射输出值的范围如下所述：

输出类型(0~3)	映射数据输出范围
0: 电压 (保留)	0 ~ 10000 mv 单位: 1 mv
1: PWM 模式	0 ~ 2000 (0.0% ~ 100%) 单位: 0.05%
2: 固定宽度的 PWM 频率模式	3 ~ 50M Hz 单位: 1 Hz
3: 具有固定占空比的 PWM 频率模式	3 ~ 50M Hz 单位: 1 Hz

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

I32 ret;

VAO_DATA VaoData;

```
VaoData.outputType = 1; // PWM 模式 VaoData.inputType = 0; //反馈速度
VaoData.Config = 1000; // 设置 PWM 模式的固定频率 1000hz
VaoData.inputSrc = 0x03; //轴 0 & 轴 1
```

```
VaoData. minVel = 1000; // 最小线性速度  
VaoData. velInterval = 100; //速度间隔  
VaoData. totalPoints = 2; //2 个点  
//PWM 模式的 10% ~ 15%  
VaoData. mappingDataArr[0] = 200;  
VaoData. mappingDataArr[1] = 300;  
  
//为表 0 设置参数。  
ret = APS_set_vao_param_ex(Board_ID, 0, &VaoData);
```

还可以看看：

```
APS_get_vao_param_ex(); APS_switch_vao_table(); APS_start_vao()
```

APS_get_vao_param_ex	通过 VAO 结构获取参数。
----------------------	----------------

支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于通过 VAO 结构获取参数。有关详细信息，参考 APS_set_vao_param_ex()。

句法:

C/C++:

```
I32 FNTYPE APS_get_vao_param_ex( I32 Board_ID, I32 Table_No, VAO_DATA*  
VaoData );
```

Visual Basic:

```
APS_get_vao_param_ex (ByVal Board_ID As Integer, ByVal Table_No As Integer, ByRef  
VaoData As VAO_DATA) As Integer
```

参数:

I32 Table_No: VAO 表编号。范围是 0~7。

VAO_DATA *VaoData: 用于设置所有参数的 VAO 结构。 有关更多详细信息，请参考
APS_set_vao_param_ex() for more details.

Typedef struct _VAO_DATA

```
{  
    //参数  
    I32 outputType;    //输出类型, [0, 3]  
    I32 inputType;    //输入类型, [0, 1]  
    I32 config;    //根据输出类型进行 PWM 配置  
    I32 inputSrc;    //轴输入源, [0, 0xf]  
  
    //映射表  
    I32 minVel;    //最小线性速度, [ 正值 ]  
    I32 VelInterval;    //速度间隔, [正值]  
    I32 totalPoints;    //总点数, [1, 32]  
    I32 *mappingDataArr;//映射数据数组}  
VAO_DATA, *PVAO_DATA;
```

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
VAO_DATA VaoData;
```

```
//获取 VAO 表 0 的 VAO 参数结构。  
ret = APS_get_vao_param_ex(Board_ID, 0, &VaoData );
```

还可以看看:

```
APS_set_vao_param_ex(); APS_start_vao()
```

APS_switch_vao_table

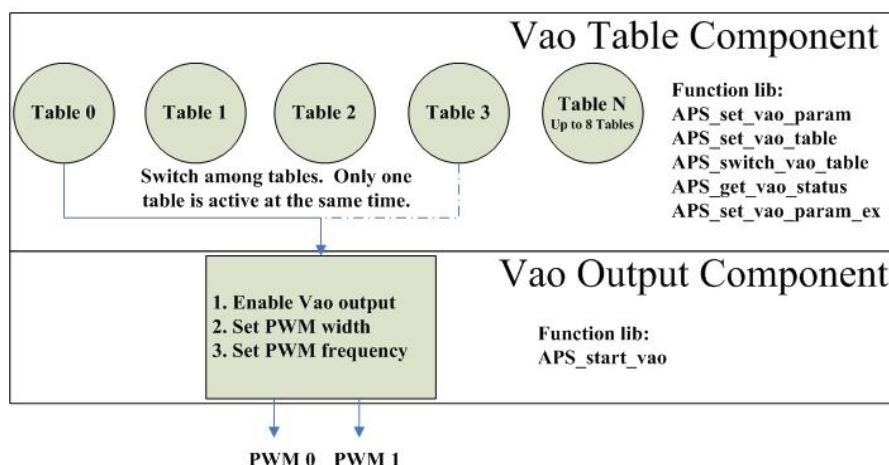
切换到指定的 VAO 表

支持的产品:PCI-8253/56

描述:

VAO 模块是一个激光控制应用。根据相应的线性速度提供模拟输出和 PWM 信号。

该函数用于切换到指定的 VAO 表，如下图，最多可以配置 8 个表。用户可以切换到其中的每个表。同一时间只有一个表处于活动状态。



请注意，如果在该点上正在运行点表，则将通过设置“opt”变量来自动切换到指定的表。

请参阅 **APS_set_point_table()**。另外，用户还可以通过 **APS_switch_vao_table()** 手动切换到指定的表。

句法:

C/C++:

```
I32 FNTYPE APS_switch_vao_table( I32 Board_ID, I32 Table_No );
```

Visual Basic:

```
APS_switch_vao_table(ByVal Board_ID As Long, ByVal Table_No As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Table_No: VAO 表编号.

0 ~ 7: 表编号.

-1: 禁用所有表.

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
ret = APS_switch_vao_table( Board_ID, 0 ); //切换至表 0。
```

还可以看看：

```
APS_set_vao_param(); APS_get_vao_param(); APS_set_vao_table (); APS_start_vao()
```

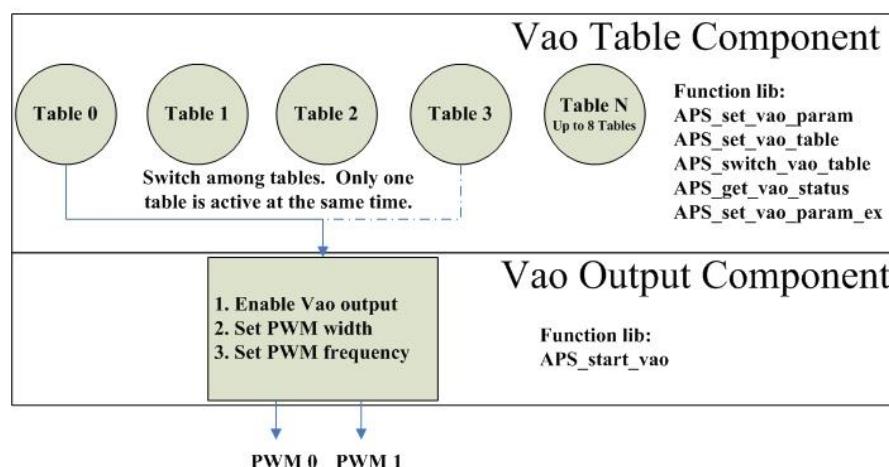
APS_start_vao	启用 VAO 输出通道
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

VAO 模块是一个激光控制应用。根据相应的线性速度提供模拟输出和 PWM 信号。

该函数用于启用 VAO 输出通道，如下图所示。当启用 VAO 输出时，模拟电压或 PWM 信号将根据相应的线性速度连续输出。

用户还可以使用 APS_start_vao()禁用 VAO 输出通道。



句法:

C/C++:

```
I32 FNTYPE APS_start_vao( I32 Board_ID, I32 Output_Ch, I32 Enable );
```

Visual Basic:

```
APS_start_vao (ByVal Board_ID As Long, ByVal Output_Ch As Long, ByVal Enable As Long) As Long
```

参数:

对于 PCI-8253/56:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Output_Ch: PWM 或模拟通道。范围是 0~1。

0: PWM 通道 0 或模拟输出 4

1: PWM 通道 1 或模拟输出 5

I32 Enable: 启用指定的通道以输出 PWM/电压。

0: 禁用. 1: 启用

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Output_Ch: PWM 或模拟通道。范围为 0 ~ 5.

0: PWM 通道 0

- 1: PWM 通道 1
- 2: PWM 通道 2 (仅 8258)
- 3: PWM 通道 3 (仅 8258)
- 4: 模拟输出 3 (脉冲模式)
- 5: 模拟输出 7 (脉冲模式)

I32 Enable: 启用指定的通道以输出 PWM/电压。

- 0: 禁用. 1: 启用

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
ret = APS_start_vao( Board_ID, 0, 1 ); // 启用 PWM 通道 0 进行输出
```

还可以看看:

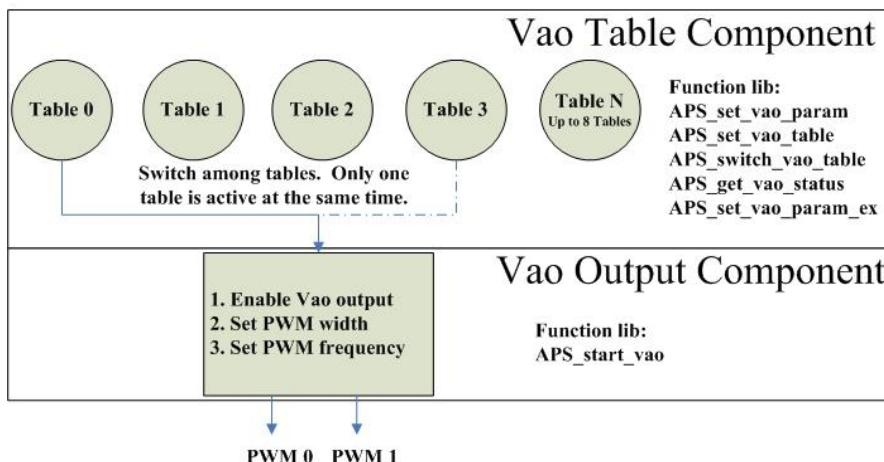
```
APS_set_vao_param();APS_get_vao_param(); APS_set_vao_table  
();APS_switch_vao_table()
```

APS_get_vao_status	获取 VAO 状态
--------------------	-----------

支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于获取 VAO 状态。用户可以监控哪个表处于活动状态，哪个 PWM 启用了，如下图所示。



句法:

C/C++:

```
I32 FNTYPE APS_get_vao_status( I32 Board_ID, I32 *Status );
```

Visual Basic:

```
APS_get_vao_status (ByVal Board_ID As Long, Status As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 *Status: 按位获取 VAO 状态。

Bit 0~7 : 表 0~7 有效。

Bit 8~15 : 保留

Bit 16 : 启用 PWM 0 或模拟 4。

Bit 17 : 启用 PWM 1 或模拟 5。

Bit 18~: 保留

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 status;
```

```
//获取 VAO 状态。
```

```
Ret = APS_get_vao_status(Board_ID, &status );
```

```
.....
```

还可以看看:

```
APS_start_vao( ); APS_switch_vao_table( ); APS_start_vao
```

APS_check_vao_param	检查指定 VAO 表的参数设置
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于检查指定 VAO 表的表参数。

句法:

C/C++:

```
I32 FNTYPE APS_check_vao_param( I32 Board_ID, I32 Table_No, I32 *Status );
```

Visual Basic:

```
APS_check_vao_param (ByVal Board_ID As Long, ByVal Table_No As Long, Status As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 Table_No: VAO表编号。范围是0~7。

I32 *Status: 参数的检查状态。请参考 VAO 参数表定义。

- 0: 没有任何参数错误
- 1: 表输入类型的参数超出范围。 (VAO_TABLE_INPUT_TYPE)
- 2: 表输出类型的参数超出范围。 (VAO_TABLE_OUTPUT_TYPE)
- 3: 表输入源的参数超出范围。 (VAO_TABLE_SRC)
- 4: 表 pwm 操作的参数超出范围。 (VAO_TABLE_PWM_CONFIG)
- 5: 映射表数据超出范围。 (参考 APS_set_vao_table())

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 Sts;
```

```
// 检查指定 VAO 表的参数设置
```

```
// VAO 表 0 的检查参数设置
```

```
ret = APS_check_vao_param (Board_ID, 0, & Sts );
```

```
.....
```

还可以看看:

[APS_set_vao_param\(\)](#); [APS_set_vao_table\(\)](#)

APS_set_pwm_on	开始输出 PWM 信号
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于输出 PWM 信号。用于激活激光、触发器等等。在主连接接口上有两个 PWM 通道 : TRG1 和 TRG2。

请注意，PWM 输出 (TRG) 由两个函数 API 使用，它们是 APS_set_pwm_on() 和 APS_start_vao()。请勿同时使用它们。确保仅启用其中之一，指定的 PWM 通道可以正常工作。

句法:

C/C++:

```
I32 FNTYPE APS_set_pwm_on( I32 Board_ID, I32 PWM_Ch, I32 PWM_On );
```

Visual Basic:

```
APS_set_pwm_on( ByVal Board_ID As Long , ByVal PWM_Ch As Long , ByVal PWM_On  
As Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从0到1。

I32 PWM_On: 0: PWM OFF, 1: PWM ON

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;  
I32 PWM_Ch = 0; //TRG 0 已使用  
I32 Width = 2000 ns; //脉冲宽度为 2 us.  
I32 Frequency = 10000 Hz;//脉冲频率为 10K Hz.
```

```
// 将脉冲宽度设置给 PWM 通道 0  
ret = APS_set_pwm_width( Board_ID, PWM_Ch, Width );  
// 将脉冲频率设置给 PWM 通道 0  
ret = APS_set_pwm_frequency( Board_ID, PWM_Ch, Frequency );  
// 输出 PWM 信号以激活激光  
ret = APS_set_pwm_on ( Board_ID, PWM_Ch, 1 );  
.....  
// 停止输出 PWM 信号
```

```
Ret = APS_set_pwm_on ( Board_ID, PWM_Ch, 0 );
```

还可以看看：

```
APS_set_pwm_width();APS_set_pwm_frequency();APS_get_pwm_width();
```

```
APS_get_pwm_frequency()
```

<code>APS_set_pwm_width</code>	将脉冲宽度设置给一个 PWM 通道
支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C	

描述:

此函数用于将脉冲宽度设置给专用的 PWM 通道。

对于 PCI-8253/56 :

请注意，脉冲宽度的范围是 40 到 335544340。单位是 ns。脉冲宽度的分辨率为 20 ns。

对于 PCI-8254/58 / AMP-204/8C:

请注意，脉冲宽度的范围是 20 到 335544300。单位是 ns。脉冲宽度的分辨率为 20 ns。

句法:

C/C++:

```
I32 FNTYPE APS_set_pwm_width( I32 Board_ID, I32 PWM_Ch, I32 Width );
```

Visual Basic:

```
I32 FNTYPE APS_set_pwm_width( ByVal Board_ID As Long , ByVal PWM_Ch As Long ,
ByVal Width As Long ) As Long
```

参数:

对于 PCI-8253/56 :

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围是 0 到 1。

I32 Width: 脉冲宽度。单位: ns。范围是 40 到 335544340。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 PWM_Ch: PWM输出通道 (TRG) 编号。从零开始。范围是0到1。

I32 Width: 脉冲宽度。单位: ns。范围是 20 到 335544300。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 PWM_Ch = 0; // TRG 0 已使用。
```

```
I32 Width = 2000 ns; //脉冲宽度是 2 us。
```

```
// 将脉冲宽度设置给 PWM 通道 0
```

```
ret = APS_set_pwm_width( Board_ID, PWM_Ch, Width );
```

还可以看看：

APS_set_pwm_on(); APS_set_pwm_frequency(); APS_get_pwm_width();
APS_get_pwm_frequency()

APS_set_pwm_frequency	将脉冲频率设置给一个 PWM 通道
支持的产品:PCI-8253/56, PCI-8254/58 / AMP-204/8C	

描述:

此函数用于将脉冲频率设置给专用的 PWM 通道。

对于 PCI-8253/56:

请注意，脉冲频率的范围是 1 到 25,000,000，单位是 Hz。实际输出频率和您设置的频率之间可能会有一点偏差。实际频率按以下公式计算：

$$\text{频率} = \frac{100,000,000}{2 \times N + 4}$$

N: 0 ~ 2147483647 (正的 32 位值)

例如，用户可以通过此函数为板卡设置频率=10005 Hz。在该函数的一侧，它从公式中获得 N = 4988 并将其发送到控制器，并且 PWM 的实际输出频率将为 10000 Hz (根据上述公式)。

对于 PCI-8254/58 / AMP-204/8C:

请注意，脉冲频率的范围是 3 到 50,000,000，单位是 Hz。实际输出频率和您设置的频率之间可能会有一点偏差。实际频率按以下公式计算：

$$\text{频率} = \frac{1,000,000,000}{20 \times N}$$

N: 0 ~ 16777215 (正的 32 位值)

例如，用户可以通过此函数为板卡设置频率=10005 Hz。在该函数的一侧，它从公式中获得 N = 5000 并将其发送到控制器，并且 PWM 的实际输出频率将为 10000 Hz (根据上述公式)。

句法:

C/C++:

```
I32 FNTYPE APS_set_pwm_frequency( I32 Board_ID, I32 PWM_Ch, I32 Frequency );
```

Visual Basic:

```
APS_set_pwm_frequency( ByVal Board_ID As Long , ByVal PWM_Ch As Long , ByVal
Frequency As Long ) As Long
```

参数:

对于 PCI-8253/56:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。

I32 Frequency: 脉冲频率。单位：Hz。范围从 1 到 25000000。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。
I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。
I32 Frequency: 脉冲频率。单位：Hz。范围从 3 到 50000000。

返回值：

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 ret;  
I32 PWM_Ch = 0; // TRG 0 已使用  
I32 Frequency = 10000 Hz; // 脉冲频率为 10k Hz.
```

```
// 将脉冲频率设置给 PWM 通道 0  
ret = APS_set_pwm_frequency( Board_ID, PWM_Ch, Frequency);
```

还可以看看：

```
APS_set_pwm_on(); APS_set_pwm_width(); APS_get_pwm_width();  
APS_get_pwm_frequency()
```

APS_get_pwm_width	从一个 PWM 通道获取脉冲宽度
-------------------	------------------

支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于从指定的 PWM 通道获取脉冲宽度。

句法:

C/C++:

I32 FNTYPE APS_get_pwm_width(I32 Board_ID, I32 PWM_Ch, I32 *Width);

Visual Basic:

I32 FNTYPE APS_get_pwm_width(ByVal Board_ID As Long , ByVal PWM_Ch As Long ,
Width As Long) As Long

参数:

对于 PCI-8253/56:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。

I32 Width: 脉冲宽度。单位 : ns。范围从 40 到 335544340。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。

I32 Width: 脉冲宽度。单位 : ns。范围从 20 到 335544300。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 PWM_Ch = 0; // TRG 0 已使用.
```

```
I32 Width;
```

```
// 从 PWM 通道 0 获取脉冲宽度
```

```
ret = APS_get_pwm_width( Board_ID, PWM_Ch, &Width );
```

还可以看看:

[APS_set_pwm_on\(\)](#); [APS_set_pwm_width\(\)](#); [APS_set_pwm_frequency\(\)](#);
[APS_get_pwm_frequency\(\)](#)

APS_get_pwm_frequency	从一个 PWM 通道获取脉冲频率
-----------------------	------------------

支持的产品:PCI-8253/56 , PCI-8254/58 / AMP-204/8C

描述:

此函数用于从指定的 PWM 通道获取脉冲频率。

句法:

C/C++:

I32 FNTYPE APS_get_pwm_frequency(I32 Board_ID, I32 PWM_Ch, I32 *Frequency);

Visual Basic:

APS_get_pwm_frequency(ByVal Board_ID As Long , ByVal PWM_Ch As Long ,
Frequency As Long) As Long

参数:

对于 PCI-8253/56:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。

I32 Frequency: 脉冲频率。单位 : Hz。范围从 1 到 25000000。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 PWM_Ch: PWM 输出通道 (TRG) 编号。从零开始。范围从 0 到 1。

I32 Frequency: 脉冲频率。单位 : Hz。范围从 3 到 50000000。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
I32 PWM_Ch = 0; // TRG 0 已占用.
```

```
I32 Frequency;
```

```
// 从 PWM 通道 0 获取脉冲频率。
```

```
ret = APS_get_pwm_frequency( Board_ID, PWM_Ch, &Frequency);
```

还可以看看:

[APS_set_pwm_on\(\)](#); [APS_set_pwm_frequency\(\)](#); [APS_set_pwm_width\(\)](#);

[APS_get_pwm_width\(\)](#)

27. 环形限位函数

APS_set_circular_limit	设置环形限位的配置
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

设置循环限制的配置。

句法:

C/C++:

```
I32 FNTYPE APS_set_circular_limit (I32 Axis_A, I32 Axis_B, F64 Center_A, F64 Center_B,  
F64 Radius, I32 Stop_Mode, I32 Enable);
```

Visual Basic:

```
APS_set_circular_limit (ByVal Axis_ID_A As Integer, ByVal Axis_ID_B As Integer, ByVal  
Center_A As Double, ByVal Center_B As Double, ByVal Radius As Double, ByVal  
Stop_mode As Integer, ByVal Enable As Integer) As Integer
```

参数:

I32 Axis_A: 轴 ID 0~7

I32 Axis_B: 轴 ID 0~7

F64 Center_A: Axis_A 的中心位置。

F64 Center_B: Axis_B 的中心位置。

F64 Radius: 圆弧极限边界到中心的距离。

I32 Stop_Mode: 触发圆弧极限时，只有 Axis_A 和 Axis_B 停止或所有轴停止。

I32 Enable: 0 : 禁用循环限制； 1 : 启用循环限制

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
ret = APS_initial( &BoardID, 0 );
```

```
// 开启伺服
```

```
ret = APS_set_servo_on( 0, 1 );
```

```
ret = APS_set_servo_on( 1, 1 );
```

```
//设定命令
```

```
ret = APS_set_command(0, 0);
```

```
ret = APS_set_command(1, 0);
```

```
// 启用循环限制
ret = APS_set_circular_limit( 0, 1, 0, 0, 100, 1, 1 );

// 设置中断
Int_No = APS_set_int_factor( 0, 0, 17, 1 ); //启用中断因子
APS_int_enable( 0, 1 ); //启用中断主开关
// 开始运动
ret = APS_relative_move( 0, 500, 1000 );

// 等待中断
returnCode = APS_wait_single_int( Int_No, -1 );
if( returnCode == ERR_NoError )
{ //中断发生
    APS_reset_int( Int_No );
}

// 禁用循环限制
ret = APS_set_circular_limit( 0, 1, 0, 0, 100, 1, 0 );
```

还可以看看:

APS_get_circular_limit()

<code>APS_get_circular_limit</code>	获取环形限位的配置
支持的产品:PCI-8254/58 / AMP-204/8C , PCIe-833x	

描述:

获取循环限制的配置。

句法:

C/C++:

```
I32 FNTYPE APS_get_circular_limit (I32 Axis_A, I32 Axis_B, F64 *Center_A, F64  
*Center_B, F64 *Radius, I32 *Stop_Mode, I32 *Enable);
```

Visual Basic:

```
APS_get_circular_limit (ByVal Axis_ID_A As Integer, ByVal Axis_ID_B As Integer, ByRef  
Center_A As Double, ByRef Center_B As Double, ByRef Radius As Double, ByRef  
Stop_mode As Integer, ByRef Enable As Integer) As Integer
```

参数:

I32 Axis_A: 轴 ID 0~7

I32 Axis_B: 轴 ID 0~7

F64 Center_A: Axis_A 的中心位置。

F64 Center_B: Axis_B 的中心位置。

F64 Radius: 圆弧极限边界到中心的距离。

I32 Stop_Mode: 触发圆弧极限时，只有 Axis_A 和 Axis_B 停止或所有轴停止。

I32 Enable: 0 : 禁用循环限制； 1 : 启用循环限制

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_circular_limit\(\)](#)

28. 同动函数

APS_set_absolute_simultaneous_move	设置一个绝对同动
------------------------------------	----------

支持的产品: MNET-4XMO-(C), PCIe-8154/8158, PCI-C154(+)

描述:

该函数用于设置一个绝对同动。用户可以设置指定的轴以实现同动。Distance_Array 和 Max_Speed_Array 的参数应用于指定的轴。

此后，用户可以调用“APS_start_simultaneous_move() / APS_stop_simultaneous_move()”函数，在同一时间，启动/停止与指定轴的同步操作。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡/模块。

句法:

C/C++:

```
I32 FNTYPE APS_set_absolute_simultaneous_move( I32 Dimension, I32 *Axis_ID_Array,  
I32 *Position_Array, I32 *Max_Speed_Array );
```

Visual Basic:

```
APS_set_absolute_simultaneous_move ( ByVal Dimension As Long, Axis_ID_Array As  
Long, Position_Array As Long, Max_Speed_Array As Long ) As Long
```

参数:

I32 Dimension: 同步轴的尺寸。(1~4 轴)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 Position_Array: 绝对位置数组。(单位：脉冲)

I32 Max_Speed_Array: 最大速度数组。(单位：脉冲/秒)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
//...初始化板卡  
I32 Dimension = 4;  
I32 Axis_ID_Array[4] = { 0, 1, 2, 3};  
I32 Position_Array = {10000, 10000, 10000, 10000};  
I32 Max_Speed_Array = {10000, 10000, 10000, 10000};  
I32 Ret;
```

```
// 设置绝对同动
```

```
Ret = APS_set_absolute_simultaneous_move ( Dimension, Axis_ID_Array,  
Position_Array, Max_Speed_Array );  
// 开始同动  
Ret = APS_start_simultaneous_move( Axis_ID_Array[0] );  
...  
// 停止同动  
Ret = APS_stop_simultaneous_move( Axis_ID_Array[0] );
```

还可以看看:

APS_set_relative_simultaneous_move();APS_start_simultaneous_move();
APS_stop_simultaneous_move()

APS_set_relative_simultaneous_move	设置一个绝对的同动
支持的产品: MNET-4XMO-(C), PCIe-8154/8158, PCI-C154(+)	

描述:

该函数用于设置一个相对同动。用户可以设置指定的轴以实现同动。Distance_Array 和 Max_Speed_Array 的参数应用于指定的轴。

此后，用户可以调用“APS_start_simultaneous_move() / APS_stop_simultaneous_move()”函数，在同一时间，启动/停止与指定轴的同步操作。

注意：Axis_ID_Array 中指定的轴必须属于同一板卡/模块。

句法:

C/C++:

```
I32 FNTYPE APS_set_relative_simultaneous_move( I32 Dimension, I32 *Axis_ID_Array,
I32 *Distance_Array, I32 *Max_Speed_Array );
```

Visual Basic:

```
APS_set_relative_simultaneous_move ( ByVal Dimension As Long, Axis_ID_Array As
Long, Distance_Array As Long, Max_Speed_Array As Long ) As Long
```

参数:

I32 Dimension: 同步轴的尺寸。 (1~4 axes)

I32 *Axis_ID_Array: 轴 ID 数组范围为 0 到 65535。

I32 Distance_Array: 相对距离数组。 (单位:脉冲)

I32 Max_Speed_Array: 最大速度数组。 (单位 : 脉冲/秒)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//...初始化板卡
I32 Dimension = 4;
I32 Axis_ID_Array[4] = { 0, 1, 2, 3};
I32 Distance_Array = {10000, 10000, 10000, 10000};
I32 Max_Speed_Array = {10000, 10000, 10000, 10000};
I32 Ret;
```

//设置相对同动

```
Ret = APS_set_relative_simultaneous_move ( Dimension, Axis_ID_Array, Distance_Array,
Max_Speed_Array );
// 开始同动
```

```
Ret = APS_start_simultaneous_move( Axis_ID_Array[0] );  
...  
// 停止同动  
Ret = APS_stop_simultaneous_move( Axis_ID_Array[0] );
```

还可以看看：

APS_set_absolute_simultaneous_move();APS_start_simultaneous_move();APS_stop_simultaneous_move()

APS_start_simultaneous_move	开始同动
-----------------------------	------

支持的产品: MNET-4XMO-(C), PCIe-8154/8158, PCI-C154(+)

描述:

该函数用于启动指定轴在相同的时间内进行同步操作。

句法:

C/C++

I32 FNTYPE APS_start_simultaneous_move (I32 Axis_ID);

Visual Basic:

APS_start_simultaneous_move (ByVal Axis_ID As Long) As Long

参数:

I32 Axis_ID: 指定同步轴的第一个轴。 轴 ID 为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//...初始化板卡
I32 Dimension = 4;
I32 Axis_ID_Array[4] = { 0, 1, 2, 3};
I32 Distance_Array = {10000, 10000, 10000, 10000};
I32 Max_Speed_Array = {10000, 10000, 10000, 10000};
I32 Ret;

// 设置相对同动
Ret = APS_set_relative_simultaneous_move ( Dimension, Axis_ID_Array, Distance_Array,
Max_Speed_Array );
// 开始同动
Ret = APS_start_simultaneous_move( Axis_ID_Array[0] );
...
// 停止同动
Ret = APS_stop_simultaneous_move( Axis_ID_Array[0] );
```

还可以看看:

APS_set_absolute_simultaneous_move();APS_set_relative_simultaneous_move();
APS_stop_simultaneous_move()

APS_stop_simultaneous_move	停止同步运动
----------------------------	--------

支持的产品: MNET-4XMO-(C), PCIe-8154/8158, PCI-C154(+)

描述:

该函数用于停止指定轴在相同的时间内进行同步操作。

句法:

C/C++

I32 FNTYPE APS_stop_simultaneous_move (I32 Axis_ID);

Visual Basic:

APS_stop_simultaneous_move (ByVal Axis_ID As Long) As Long

参数:

I32 Axis_ID: 指定同步轴的第一个轴。 轴 ID 为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
//...初始化板卡
I32 Dimension = 4;
I32 Axis_ID_Array[4] = { 0, 1, 2, 3};
I32 Distance_Array = {10000, 10000, 10000, 10000};
I32 Max_Speed_Array = {10000, 10000, 10000, 10000};
I32 Ret;

// 设置相对同动
Ret = APS_set_relative_simultaneous_move ( Dimension, Axis_ID_Array, Distance_Array,
Max_Speed_Array );
// 开始同动
Ret = APS_start_simultaneous_move( Axis_ID_Array[0] );
...
// 停止同动
Ret = APS_stop_simultaneous_move( Axis_ID_Array[0] );
```

还可以看看:

APS_set_absolute_simultaneous_move();APS_set_relative_simultaneous_move();
APS_start_simultaneous_move()

29. 单锁存(Single-latch)函数

APS_manual_latch2	对一个轴进行手动锁存
支持的产品: MNET-4XMO-(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+)	

描述:

该函数用于产生一个手动锁存信号。

句法:

C/C++:

```
I32 FNTYPE APS_manual_latch2( I32 Axis_ID );
```

Visual Basic:

APS_manual_latch2 (ByVal Axis_ID As Long) As Long

参数:

I32 Axis_ID: 轴 ID 数组范围为 0 到 65535。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 axisID = 0;
```

```
I32 ret = 0;
```

```
I32 LatchData = 0;
```

```
ret = APS_manual_latch2( axisID );
```

```
//锁存数据是命令计数器
```

```
ret = APS_get_latch_data2( axisID, 0, &LatchData );
```

还可以看看:

[APS_get_latch_data2\(\)](#)

APS_get_latch_data2	获取轴的锁存数据
支持的产品:MNET-4XMO(C), MNET-1XMO, PCI(e)-8154/8158, PCI-8102/PCI-C154(+)	

描述:

该函数用于获取锁存数据。锁存数据有两种获取方法。一种是来自物理锁存引脚的输入信号。

另一个是来自手动锁存的内部锁存信号。包括用户可以锁存在内的四种数据。他们是：

1. 命令计数器 (命令位置)
2. 反馈计数器 (反馈位置)
3. 错误计数器 (错误位置) /当前速度
4. 通用计数器

句法:

C/C++:

```
I32 FNTYPE APS_get_latch_data2( I32 Axis_ID, I32 LatchNum, I32 *LatchData );
```

Visual Basic:

```
APS_get_latch_data2 ( ByVal Axis_ID As Long ) As Long
```

参数:

I32 Axis_ID: 轴 ID 数组范围为 0 到 65535。

I32 LatchNum:

- 0: 指定计数器
- 1: 反馈计数器
- 2: 错误计数器/当前速度 (通过轴参数 22Dh PRA_LATCH_DATA_SPD)
- 3: 通用计数器

I32 *LatchData: 锁存数据

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 axisID = 0;
I32 ret = 0;
I32 LatchData = 0;
```

```
ret = APS_manual_latch2( axisID );
//锁存数据是命令计数器
ret = APS_get_latch_data2( axisID, 0, &LatchData );
```

还可以看看:

APS_manual_latch2()

30. 多锁存(Multi-latch)函数

APS_set_ltc_counter	设置编码器计数器值
支持的产品:PCI-C154(+)	

描述:

该函数用于设置编码器计数器值。

句法:

C/C++:

```
I32 FNTYPE APS_set_ltc_counter ( I32 Board_ID, I32 LtcCh, I32 CntValue );
```

Visual Basic:

```
APS_set_ltc_counter ( ByVal Board_ID As Long, ByVal LtcCh As Long, ByVal CntValue As  
Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 LtcCh: 指定的通道编号。

I32 CntValue: 编码器(计数器)值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
//将锁存计数器 0 设置为 100
```

```
ret = APS_set_ltc_counter ( Board_ID, 0, 100 );
```

还可以看看:

[APS_get_ltc_counter \(\)](#)

APS_get_ltc_counter	获取编码器计数器值
---------------------	-----------

支持的产品:PCI-C154(+)

描述:

该函数用于获取编码器计数器值。

句法:

C/C++:

```
I32 FNTYPE APS_get_latch_counter ( I32 Board_ID, I32 LtcCh, I32 *CntValue );
```

Visual Basic:

```
APS_get_latch_counter ( ByVal Board_ID As Long, ByVal LtcCh As Long, CntValue As  
Long ) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 LtcCh: 指定的通道编号。

I32 CntValue: 编码器(计数器)值。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
I32 CntValue= 0;
```

//从锁存计数器 0 获取计数器值。

```
ret = APS_get_ltc_counter ( Board_ID, 0, &CntValue );
```

还可以看看:

[APS_set_ltc_counter \(\)](#)

APS_set_ltc_fifo_param	设置锁存参数
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

该函数用于设置各种锁存器参数。请参阅锁存参数表以获取定义和详细描述。

句法:

C/C++

```
I32 FNTYPE APS_set_ltc_fifo_param( I32 Board_ID, I32 FLtcCh, I32 Param_No, I32
Param_Val );
```

Visual Basic:

```
APS_set_ltc_fifo_param (ByVal Board_ID As Long, ByVal FLtcCh As Long, ByVal
Param_No As Long, ByVal Param_Val As Long ) As Long
```

参数:

对于 PCI-C154(+):

I32 Board_ID: 板卡 ID 从 0 到 31。

FLtcCh: 指定的锁存通道。

I32 Param_No: 锁存参数编号。请参考锁存器参数表进行定义。

I32 Param_Val: 锁存参数值。有关详细信息，请参见锁存器参数表。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh: 锁存通道范围为 0~3

I32 Param_No: 锁存参数编号。请参考锁存器参数表进行定义。

I32 Param_Val: 锁存参数值。有关详细信息，请参见锁存器参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
//在通道 0 中将编码器输入模式 ( 0x00 ) 设置为 4xAB-Phase ( 4 )
ret = APS_set_ltc_fifo_param ( Board_ID, 0, 0, 4 );
```

还可以看看:

[APS_get_ltc_fifo_param \(\)](#)

APS_get_ltc_fifo_param	获取锁存参数
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

该函数用于获取各种锁存器参数。 请参阅锁存参数表以获取定义和详细描述。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_param( I32 Board_ID, I32 FLtcCh, I32 Param_No, I32
*Param_Val );
```

Visual Basic:

```
APS_get_ltc_fifo_param (ByVal Board_ID As Long, ByVal FLtcCh As Long, ByVal
Param_No As Long, Param_Val As Long) As Long
```

参数:

对于 PCI-C154(+):

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 FLtcCh: 指定的锁存通道。

I32 Param_No: 锁存参数编号。请参考锁存器参数表进行定义。

I32 *Param_Val: 锁存参数值。有关详细信息，请参见锁存器参数表。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh: 所存通道范围为0~3。

I32 Param_No: 锁存参数编号。请参考锁存器参数表进行定义。

I32 Param_Val: 锁存参数值。有关详细信息，请参见锁存器参数表。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
I32 Param_Val = 0;
```

```
//在通道 0 中获取编码器输入模式 ( 0x00 )
```

```
ret = APS_get_ltc_fifo_param ( Board_ID, 0, 0, &Param_Val );
```

还可以看看:

[APS_set_ltc_fifo_param \(\)](#)

APS_manual_latch	手动和同步锁存数据。
支持的产品:PCI-C154(+)	

描述:

此函数用于手动锁存数据。 它旨在同步锁存一个或多个通道。

句法:

C/C++:

```
I32 FNTYPE APS_manual_latch ( I32 Board_ID, I32 LtcChInBit );
```

Visual Basic:

```
APS_manual_latch ( ByVal Board_ID As Long, ByVal LtcChInBit As Long) As Long
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 LtcChInBit: 按位指定的锁存器通道。

Bit 0: 通道 0, Bit 1: 通道 1, ..., Bit 8: 通道 8

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
//同步锁存通道 0~3 的数据。
ret = APS_manual_latch( Board_ID, 0xf );
```

还可以看看:

APS_enable_ltc_fifo	启用/禁用锁存 fifo /启用位置锁存过程
---------------------	------------------------

支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C

描述:

对于 PCI-C154(+)，此函数用于启用/禁用锁存器 fifo。一旦禁用，锁存引脚将忽略任何锁存信号。用户必须在使用任何锁存相关功能之前启用此函数，并且在用户不再使用锁存时禁用此函数。锁存器 fifo 最多可以存储 258 个锁存器数据，用户可以监视 fifo 状态并获取锁存器数据。

对于PCI-8254/58 / AMP-204/8C，此函数用于启用位置锁存过程。图1显示了位置锁存模块的架构。它有四个位置锁存通道，每个通道都有专用的FIFO和队列。用户必须为位置锁存通道指定触发源和编码器编号。触发源可以是数字输入信号或PWM脉冲输出。此处还支持上升，下降或两个边沿触发模式。这些设置可以通过锁存参数表进行配置。

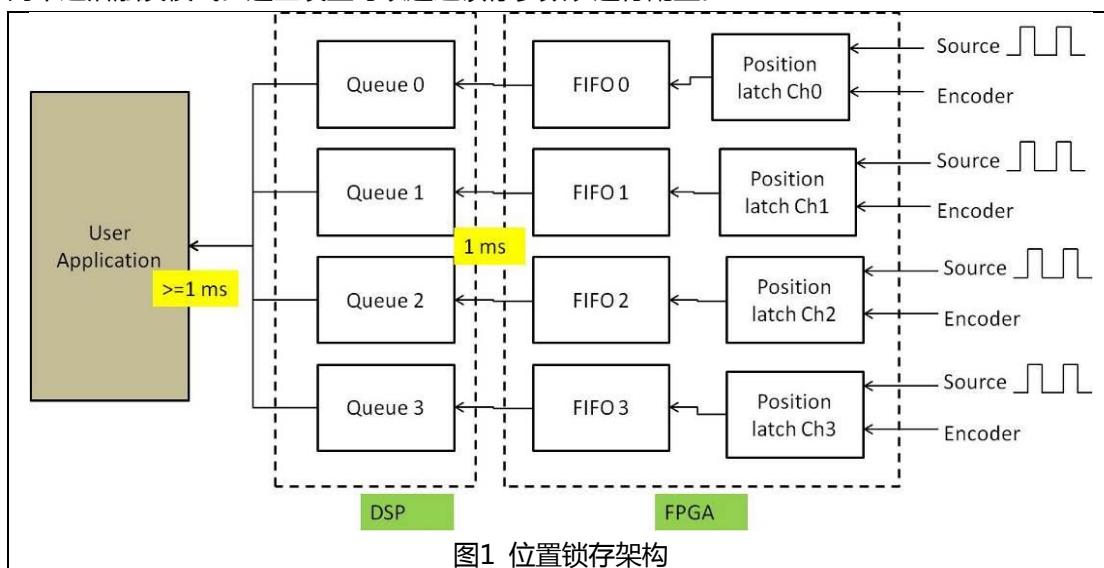


图1 位置锁存架构

句法:

C/C++:

```
I32 FNTYPE APS_enable_ltc_fifo( I32 Board_ID, I32 FLtcCh, I32 Enable );
```

Visual Basic:

```
APS_enable_ltc_fifo (ByVal Board_ID As Long, ByVal FLtcCh As Long, ByVal Enable As Long) As Long
```

参数:

For PCI-C154(+):

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 FLtcCh: 指定的锁存通道。

I32 Enable: 启用/禁用锁存

0: 禁用. 1: 启用

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 FLtcCh : 锁存通道范围为 0~3

I32Enable : 0 : 禁用 ; 1 : 启用

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

I32 ret;

//启用锁存 fifo 通道 0

ret = APS_enable_ltc_fifo (BoardID , 0 , 0) ; //禁用位置锁存

ret = APS_reset_ltc_fifo (BoardID , 0) ; //重置位置锁存队列

ret = APS_set_ltc_fifo_param (BoardID , 0 , LTC_IPT , 0xFFFF) ; //设置输入源

ret = APS_set_ltc_fifo_param (BoardID , 0 , LTC_ENC , 0) ; //设置编码器编号

ret = APS_set_ltc_fifo_param (BoardID , 0 , LTC_LOGIC , 0) ; //设置逻辑

ret = APS_enable_ltc_fifo (BoardID , 0 , 1) ; //开始位置锁存

还可以看看:

APS_set_ltc_fifo_param(); APS_get_ltc_fifo_param(); APS_reset_ltc_fifo()

APS_reset_ltc_fifo	重置锁存 fifo /重置锁存队列和 fifo
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

对于 PCI-C154(+)，此函数功能用于重置锁存器 fifo。锁存 fifo 将清除所有数据，并且 fifo 的状态为空。

对于 PCI-8254/58 / AMP-204/8C，在开始位置锁存之前，使用此函数可重置或清除 APS_enable_ltc_fifo 中引入的队列和 FIFO。注意，位置锁存也会被同时清除。

句法:

C/C++:

```
I32 FNTYPE APS_reset_ltc_fifo( I32 Board_ID, I32 FLtcCh );
```

Visual Basic:

```
APS_reset_ltc_fifo ( ByVal Board_ID As Long, ByVal FLtcCh As Long ) As Long
```

参数:

对于 PCI-C154(+)：

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 FLtcCh: 指定的锁存通道。

对于 PCI-8254/58 / AMP-204/8C：

I32 Board_ID: 目标控制器的 ID。通过成功调用 APS_initial() 来检索它。

I32 FLtcCh: 锁存通道范围为 0~3

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 ret;
```

```
//重置锁存 FIFO 通道 0
```

```
ret = APS_reset_ltc_fifo ( Board_ID, 0 );
```

还可以看看:

APS_get_ltc_fifo_data	从 fifo 中获取一个锁存数据
支持的产品:PCI-C154(+)	

描述:

此函数用于从 fifo 中获取一个锁存数据。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_data( I32 Board_ID, I32 FLtcCh, I32 Data );
```

Visual Basic:

```
APS_get_ltc_fifo_data (ByVal Board_ID As Long, ByVal FLtcCh As Long, Data As Long )
As Long
```

参数:

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 FLtcCh: 指定的锁存通道。

I32 Data: 获取储存在 fifo 中的锁存数据。.

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
I32 data= 0;
```

//从锁存 FIFO 通道 0 获得数据。

```
ret = APS_get_ltc_fifo_data ( Board_ID, 0, & data );
```

还可以看看:

<code>APS_get_ltc_fifo_usage</code>	获取锁存 fifo 的使用情况
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

对于 PCI-C154 (+) , 此函数用于获取锁存 fifo 的使用情况 , 即已经使用了多少个 fifo 空间。fifo 的使用范围是 0 到 258。

对于 PCI-8254/58 / AMP-204/8C , 此函数用于获取 APS_enable_ltc_fifo 中锁存队列的已用空间。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_usage( I32 Board_ID, I32 FLtcCh, I32 Usage );
```

Visual Basic:

```
APS_get_ltc_fifo_usage (ByVal Board_ID As Long, ByVal FLtcCh As Long, Usage As Long ) As Long
```

参数:

For PCI-C154(+):

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 FLtcCh: 指定的锁存通道。

I32 Usage: 获取锁存 fifo 的使用情况。

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh : 锁存通道范围为 0~3

I32 *Usage : 队列已使用的空间

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
I32 usage = 0;
```

```
//获取锁存 FIFO 通道 0 的使用情况
```

```
ret = APS_get_ltc_fifo_usage ( Board_ID, 0, &usage );
```

还可以看看:

<code>APS_get_ltc_fifo_free_space</code>	获取锁存 fifo 的可用空间
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

对于 PCI-C154(+) , t 该函数用于获取锁存器 fifo 的可用空间。可用空间范围是 0 到 258。可用空间是指用于在 fifo 中存储数据的剩余空间。

对于 PCI-8254/58 / AMP-204/8C , 此函数用于获取 APS_enable_ltc_fifo 中锁存队列的已用空间。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_free_space( I32 Board_ID, I32 FLtcCh, I32 FreeSpace );
```

Visual Basic:

```
APS_get_ltc_fifo_free_space (ByVal Board_ID As Long, ByVal FLtcCh As Long, FreeSpace As Long ) As Long
```

参数:

对于 PCI-C154(+) :

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 FLtcCh: 指定的锁存通道。

I32 FreeSpace: 获取锁存 fifo 的可用空间。

对于 PCI(e)-8154/58:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh : 锁存通道范围为 0~3

I32 * FreeSpace : 队列可用的空间

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;
```

```
I32 space = 0;
```

```
//获取锁存 FIFO 通道 0 的可用空间
```

```
ret = APS_get_ltc_fifo_free_space ( Board_ID, 0, &space );
```

还可以看看:

APS_get_ltc_fifo_status	获取 fifo 状态/获取锁存队列和 fifo 状态
支持的产品:PCI-C154(+), PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取锁存器 fifo 的状态。用户可以监控 fifo 的状态，包括空，满，电平和溢出状态。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_status( I32 Board_ID, I32 FLtcCh, I32 Status );
```

Visual Basic:

```
APS_get_ltc_fifo_status (ByVal Board_ID As Long, ByVal FLtcCh As Long, Status As Long ) As Long
```

参数:

对于 PCI-C154(+):

I32 Board_ID: 板卡 ID 从 0 到 31。

I32 FLtcCh : 指定的锁存通道。

I32 Status : 获取锁存器 fifo 的状态。

Bit0 → 0 : 不为空，1 : 为空

Bit1 → 0 : 未满，1 : 已满

Bit2 → 0 : 高电平下 1 : 高电平下

对于 PCI-8254/58 / AMP-204/8C:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh: 锁存通道范围为0~3

I32 * Status: 状态的bit定义如下：

Bit0 = 0 : FIFO不为空，1 : FIFO为空 (循环更新)

Bit1 = 0 : FIFO未满，1 : FIFO已满 (循环更新)

Bit2 = X

Bit3 = 0 : FIFO不溢出，1 : FIFO溢出 (通过复位队列和FIFO清除)

Bit4 = 0 : 队列不为空，1 : 队列为空 (循环更新)

Bit5 = 0 : 队列未满，1 : 队列已满 (循环更新)

Bit6 = 0 : 队列未溢出，1 : 队列溢出 (通过复位队列和FIFO清除)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Board_ID = 0;
```

```
I32 ret = 0;  
I32 status = 0;  
  
//获取锁存 FIFO 通道 0 的状态  
ret = APS_get_ltc_fifo_free_status ( Board_ID, 0, &status );
```

还可以看看：

APS_get_ltc_fifo_point	获取锁存点数组
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

此函数用于获取锁存点数组。每个锁存点将包括用户坐标中的位置和相应的触发源。最大锁存点数组大小为16。

句法:

C/C++:

```
I32 FNTYPE APS_get_ltc_fifo_point( I32 Board_ID, I32 FLtcCh, I32 *ArraySize,
LATCH_POINT *LatchPoint )
APS_get_ltc_fifo_point (ByVal Board_ID As Integer, ByVal FLtcCh As Integer, ByRef
ArraySize As Integer, ByRef LatchPoint As LATCH_POINT) As Integer
```

参数:

I32 Board_ID: 目标控制器的ID。通过成功调用APS_initial()来检索它。

I32 FLtcCh : 锁存通道范围为0~3

I32 * ArraySize : 锁存点数组的大小 ; ArraySize的最大值为16。

LATCH_POINT * LatchPoint : 锁存点数组。LATCH_POINT的定义如下所示

typedef struct

```
{
    F64 position ; //锁存位置
    I32 ltcSrcInBit; //锁存源 : 位0~7 : DI; bit 8~11 : 触发通道
} LATCH_POINT;
```

Members:

position

 锁存位置

ltcSrcInBit

 (1) bit 0~7: 数字输入信号

 (2) bit 8~11: 触发通道

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

[APS_set_ltc_fifo_param\(\)](#); [APS_get_ltc_fifo_param\(\)](#); [APS_enable_ltc_fifo\(\)](#)

31. 环形计数器函数

APS_set_ring_counter

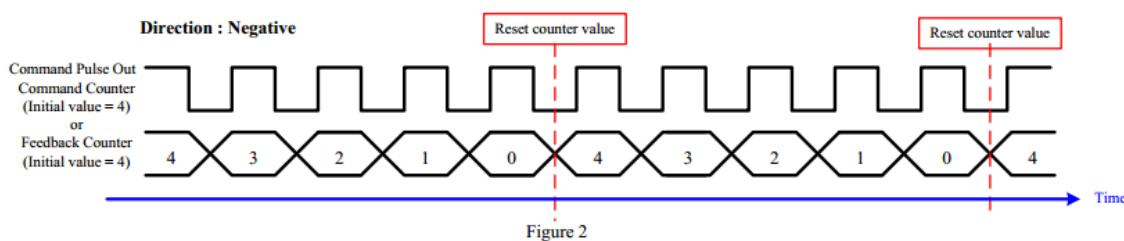
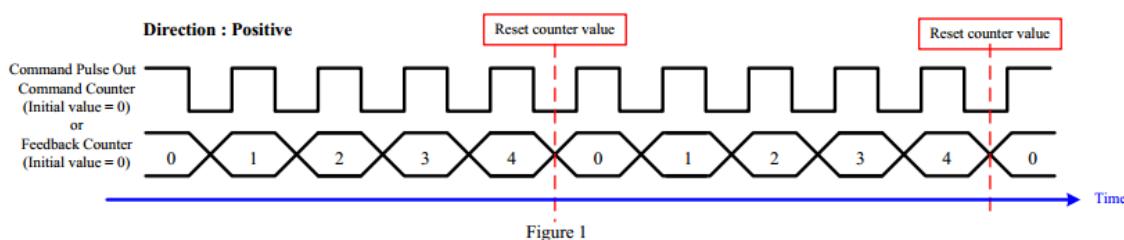
启用环形计数器函数

支持的产品:PCI-8154/58

描述:

此函数用于设置环形计数器限制值并启用环形计数器功能。当启用环形计数器功能时，命令和反馈计数器将作为环形计数器运行。

当环形计数器限制值设置为零时，环形计数器功能将被禁用。例如，在图 1 中，当环形计数器限制值 (I32 RingVal) 设置为 4 时，命令和反馈计数器将递增计数，直到计数器的值等于 4。然后命令和反馈计数器将被重置为零，并重复上述行为。相对地，在图 2 中，当环形计数器限制值 (I32 RingVal) 设置为 4 时，命令和反馈计数器将递减计数，直到计数器的值等于零，然后命令和反馈计数器将被重置为 4，并重复以上行为。



句法:

C/C++:

I32 FNTYPE APS_set_ring_counter(I32 AxisNo, I32 RingVal)

Visual Basic:

APS_set_ring_counter (ByVal AxisNo As Long, ByVal RingVal As Long) As Long

参数:

I32 AxisNo: 轴索引。

I32 RingVal: 环形计数器的极限值。 (0 < RingVal < 134217727)

如果 RingVal 等于零，则意味着禁用环形计数器函数。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例：

```
I32 AxisNo = 0; //设置轴 ID  
I32 RingVal = 1000; //设置环形计数器的极限值  
F64 Dist = 3000; //设置运动的相对距离 ( 单位 : 脉冲 )  
F64 MaxVel = 1000; //以脉冲 / 秒为单位设置最大速度  
APS_set_ring_counter(AxisNo, RingVal ); //启用环形计数器函数  
APS_relative_move(AxisNo, Dist, MaxVel); //开始相对运动  
.....  
APS_set_ring_counter(AxisNo, 0 ); //禁用环形计数器功能
```

还可以看看：

APS_get_ring_counter ()

APS_get_ring_counter	获取环形计数器的极限值
支持的产品:PCI-8154/58	

描述:

该函数用于获取环形计数器的极限值。

句法:

C/C++:

```
I32 FNTYPE APS_get_ring_counter( I32 AxisNo, I32 *RingVal )
```

Visual Basic:

```
APS_get_ring_counter (ByVal AxisNo As Long, RingVal As Long) As Long
```

参数:

I32 AxisNo: 轴索引。.

I32 *RingVal: 获取环形计数器的极限值。 (0 < RingVal < 134217727)

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 AxisNo = 0; //设置轴 ID
I32 RingVal = 1000; //设置环形计数器的极限值
F64 Dist = 3000; //设置运动的相对距离 ( 单位 : 脉冲 )
F64 StrVel = 0; //设置速度曲线的起始速度 , 以脉冲 / 秒为单位
F64 MaxVel = 1000; //以脉冲 / 秒为单位设置最大速度
APS_set_ring_counter(AxisNo, RingVal ); //启用环形计数器函数
APS_relative_move(AxisNo, Dist, MaxVel); //开始相对运动
```

```
APS_get_ring_counter(AxisNo, &RingVal ); //获取环形计数器的极限值
```

还可以看看:

[APS_set_ring_counter \(\)](#)

32. 速度曲线计算

APS_relative_move_profile	获取相对速度曲线
支持的产品:PCI-C154(+)	

描述:

此函数用于获取相对运动的速度曲线。通过此函数，用户可以获取运动前的实际速度曲线。因此，用户需要通过轴参数 PRA_CURVE(0x20)设置速度模式曲线，并通过 PRA_VS(0x23)设置启动速度，以用于计算配置文件。

句法:

C/C++:

```
I32 FNTYPE APS_relative_move_profile( I32 Axis_ID, I32 Distance, I32 Max_Speed, I32  
*StrVel, I32 *MaxVel, F64 *Tacc, F64 *Tdec, F64 *Tconst )
```

Visual Basic:

```
APS_relative_move_profile(ByVal Axis_ID As Long, ByVal Distance As Long, ByVal  
Max_Speed As Long, StrVel As Long, MaxVel As Long, Tacc As Double, Tdec As  
Double, Tconst As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Distance: 相对距离。单位：脉冲。

I32 Max_Speed: 此运动曲线的最大速度。 单位：脉冲/秒

I32 * StrVel: 起始速度。单位：脉冲/秒

I32 * MaxVel: 此运动曲线的最大速度。 单位：脉冲/秒

F64 * Tacc: 通过计算加速时间。单位：秒

F64 * Tdec: 通过计算减速时间。 单位：秒

F64 * Tconst: 恒速时间（最大速度）。单位：秒

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID = 0;  
I32 ret = 0;  
I32 Distance = 100000;  
I32 Max_Speed = 10000;  
I32 StrVel =0,MaxVel=0;  
F64 Tacc=0, Tdec=0, Tconst=0;  
ret = APS_set_axis_param( Axis_ID, PRA_VS, 1000 ); // 开始速度
```

```
ret = APS_set_axis_param( Axis_ID, PRA_CURVE, 0 ); // T 曲线  
Ret = APS_relative_move_profile( Axis_ID, Distance, Max_Speed, &StrVel, &MaxVel,  
&Tacc, &Tdec, &Tconst );
```

还可以看看:

APS_absolute_move_profile	获取绝对速度配置文件
支持的产品:PCI-C154(+)	

描述:

此函数用于获取绝对运动的速度曲线。通过此函数，用户可以获取运动前的实际速度曲线。因此，用户需要通过轴参数 PRA_CURVE(0x20)设置速度模式曲线，并通过 PRA_VS(0x23)设置启动速度，以用于计算配置文件。

句法:

C/C++:

```
I32 FNTYPE APS_absolute_move_profile( I32 Axis_ID, I32 Position, I32 Max_Speed, I32
*StrVel, I32 *MaxVel, F64 *Tacc, F64 *Tdec, F64 *Tconst )
```

Visual Basic:

```
APS_relative_move_profile(ByVal Axis_ID As Long, ByVal position As Long, ByVal
Max_Speed As Long, StrVel As Long, MaxVel As Long, Tacc As Double, Tdec As
Double, Tconst As Double ) As Long
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Position: 绝对命令位置。以脉冲为单位。

I32 Max_Speed: 此运动曲线的最大速度。 单位：脉冲/秒

I32 * StrVel: 起始速度。 单位：脉冲/秒

I32 * MaxVel: 此运动曲线的最大速度。 单位：脉冲/秒

F64 * Tacc: 通过计算加速时间。单位：秒

F64 * Tdec: 通过计算减速时间。 单位：秒

F64 * Tconst: 恒速时间（最大速度）。单位：秒

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID = 0;
I32 ret = 0;
I32 Position = 100000;
I32 Max_Speed = 10000;
I32 StrVel = 0,MaxVel=0;
F64 Tacc=0, Tdec=0, Tconst=0;
ret = APS_set_axis_param( Axis_ID, PRA_VS, 1000 ); // 开始速度
ret = APS_set_axis_param( Axis_ID, PRA_CURVE, 1 ); // S 曲线
```

```
Ret = APS_absolute_move_profile( Axis_ID, Position, Max_Speed, &StrVel, &MaxVel,  
&Tacc, &Tdec, &Tconst );
```

还可以看看:

APS_check_motion_profile_emx	获取相对速度曲线
------------------------------	----------

支持的产品:EMX-100

描述:

此函数用于获取相对运动速度曲线。通过此函数，用户可以获得实际运动前的速度曲线。如果计算出的最小距离小于用户期望，则控制器将自动计算新的减速，Vmax 参数，这些参数可以参考速度曲线标准。

句法:

C/C++:

```
I32 FNTYPE APS_check_motion_pfprofile_emx( I32 Axis_ID, Speed_profile *profile_input,  
Speed_profile *profile_output, I32 *MinDis);
```

Visual Basic:

```
APS_check_motion_pfprofile_emx(ByVal Axis_ID As Long, ByRef profile_input As  
Speed_profile, ByRef Param_Val As Speed_profile, ByRef MinDis As Long) As Long
```

参数:

I32 Axis_ID: 轴ID从0到65535。

Speed_profile profile_input : 输入速度曲线信息的结构。

Speed_profile profile_output : 输出速度曲线信息的结构。

I32 MinDis : 通过控制器计算用户配置输入速度曲线的最小距离。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

```
I32 Axis_ID = 0, MinDis = 0;  
Speed_profile Inputprofile;  
Speed_profile Outputprofile;  
Inputprofile.Acc = 1000000;  
Inputprofile.Dec = 1000000;  
Inputprofile.Vmax = 1000000;  
Inputprofile.VS = 1000;  
Inputprofile.s_factor = 10;  
ret = APS_check_motion_profile_emx(Axis_ID,&Inputprofile,&Outputprofile,&MinDis);
```

还可以看看:

33. 背隙函数

APS_set_backlash_en	启用/停止背隙
---------------------	---------

支持的产品:PCI-8254/58 / AMP-204/8C

描述:

此函数用于启用或禁用背隙函数。当轴移动的方向相反时，通过叠加将间隙补偿应用于该命令轴的移动。用户应在启用背隙之前配置轴参数PRA_BKL_DIST和PRA_BKL_CNSP。前者表示间隙补偿值，后者表示每个周期的间隙补偿增量值。如果已启用此函数，则不允许再次启用。如果用户将启用设置为2，则输出速度将受轴参数PRA_VM (0x12) 的限制。运动状态位30用于指示背隙补偿是否在运行中。

下表显示了两个背隙启用模式示例：背隙轴参数均为1000。方向表示运动状态DIR位，1表示正，0表示负。命令和位置分别表示用户坐标中的命令位置和反馈位置。编码器表示电机坐标的编码器值。假定在将“伺服”设置为“ON”后启用了背隙，然后逐步运行点对点功能，并且当“方向”反转时，会实现背隙补偿。

表1 将启用设置为1的示例：用户坐标位置未对齐

步骤	描述	方向	命令	位置	编码器	备注
1	初始条件	正向	0	0	0	伺服 ON
2	向前1000个脉冲	正向	1000	1000	1000	
3	向前1000个脉冲	正向	2000	2000	2000	
4	往后1000个脉冲	反向	1000	0	0	背隙补偿
5	往后1000个脉冲	反向	0	-1000	-1000	
6	往后1000个脉冲	反向	-1000	-2000	-2000	
7	向前1000个脉冲	正向	0	0	0	背隙补偿

表2 将“启用”设置为2的示例：用户坐标位置已对齐

步骤	描述	方向	命令	位置	编码器	备注
1	初始条件	正向	0	0	0	伺服 ON
2	向前1000个脉冲	正向	1000	1000	1000	
3	向前1000个脉冲	正向	2000	2000	2000	
4	往后1000个脉冲	反向	1000	1000	0	背隙补偿
5	往后1000个脉冲	反向	0	0	-1000	
6	往后1000个脉冲	反向	-1000	-1000	-2000	
7	向前1000个脉冲	正向	0	0	0	背隙补偿

句法:

C/C++:

I32 FNTYPE APS_set_backlash_en (I32 Axis_ID, I32 Enable);

Visual Basic:

APS_set_backlash_en (ByVal Board_ID As Integer, ByVal Enable As Integer) As Integer

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 Enable: 请参阅下面的说明

启用	描述
0	禁用背隙补偿
1	a. 启用背隙补偿 b. 用户坐标位置未对齐 c. 运动状态位30用于指示背隙补偿处于运行状态 (= 0) 或未运行 (= 1) 。
2	a. 启用背隙补偿 b. 用户坐标位置对齐 c. 输出速度受轴参数PRA_VM (0x12) 限制 d. 运动状态位30用于指示背隙补偿处于运行状态 (= 0) 或未运行 (= 1) 。

返回值:I32 Error code: 请参考 [APS 函数返回代码](#).**示例 :****还可以看看:**

APS_get_backlash_en	检查背隙补偿是否启用/停止
支持的产品:PCI-8254/58 / AMP-204/8C	

描述:

该函数用于检查背隙补偿是否启用/停止

句法:

C/C++:

```
I32 FNTYPE APS_get_backlash_en( I32 Axis_ID, I32 *Enable );
```

Visual Basic:

```
APS_get_backlash_en (ByVal Board_ID As Integer, ByRef Enable As Integer) As Integer
```

参数:

I32 Axis_ID: 轴 ID 从 0 到 65535。

I32 *Enable: 请参阅下面的说明

启用	描述
0	禁用背隙补偿
1	<ul style="list-style-type: none"> d. 启用背隙补偿 e. 用户坐标位置未对齐 f. 运动状态位30用于指示背隙补偿处于运行状态 (= 0) 或未运行 (= 1) 。
2	<ul style="list-style-type: none"> e. 启用背隙补偿 f. 用户坐标位置对齐 g. 输出速度受轴参数PRA_VM (0x12) 限制 h. 运动状态位30用于指示背隙补偿处于运行状态 (= 0) 或未运行 (= 1) 。

返回值:

I32 Error code: 请参考 [APS 函数返回代码](#).

示例 :

还可以看看:

34. 表定义

A. 板卡参数表

DPAC-1000 板卡参数表

DPAC-1000 板卡参数表				
编号	定义	描述	参数数据含义	默认值
10h	PRB_WDT0_VALUE	WDT 超时值。	0: 禁用 WDT N: 1~255 从 N 和倒数开始启动看门狗定时器。 递减计数周期为 WDT0_UNIT。计时器计数器达到零（超时）时， WDT_ACTION 会发生。	0
11h	PRB_WDT0_COUNTER	重新启动 WDT 计数或获取当前 WDT 计数器值	为 WDT0_VALUE 设置任何值，以重启 WDT 计数器 获取命令以获取当前的 WDT 计数器。	0
12h	PRB_WDT0_UNIT	WDT 计数器倒数周期单位	0: 保留 1: 秒 2: 分钟	0
13h	PRB_WDT0_ACTION	WDT 超时动作	0 : 系统重启	0
20h	PRB_TMR0_BASE	设置 TMR0 基本单元时钟	0~4095: TMR 值 计时器周期= (40 + (512 / 8.25) * TMR_value) us。 每次超时都会产生硬件中断。 要禁用计时器功能，必须禁用计时器中断。	0
21h	PRB_TMR0_VALUE	获取/设置timer0值	32位无符号值。 计时器中断发生时， 计数器每次加一	0
30h	PRB_SYS_TMP_MONITOR	获取系统温度监控数据	8位符号值。单位为摄氏度	0
31h	PRB_CPU_TMP_MONITOR	获取CPU温度监控数据	8位符号值。单位为摄氏度	0

32h	PRB_AUX_TMP_MONITOR	获取AUX温度监控数据	8位符号值。单位为摄氏度	0
40h	PRB_UART_MULTIPLIER	设置UART放大器	0 : x1模式。 如果波特率设置为115200 , 则实际波特率为115200。 1 : x8模式 如果波特率设置为 115200 , 则实际波特率为 $115200 * 8 = 921600$ 。	0
90h	PRB_PSR_MODE	设置手动脉冲发生器 (MPG) 输入模式	0: OUT/DIR 1: CW/CCW 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	4
91h	PRB_PSR_EA_LOGIC	设置EA信号逻辑	0: EA没有倒置 1: EA倒置	0
92h	PRB_PSR_EB_LOGIC	设置EB信号逻辑	0: EB没有倒置 1: EB倒置	0
10001h	PRB_DPAC _DISPLAY_MODE	DPAC 显示模式	0: 用户定义模式 1 : 演示模式	1
10002h	PRB_DPAC_DI_MODE	设置DI引脚模式	0: GPIO 模式 1: MPG输入模式	0

DPAC-3000 板卡参数表

DPAC-3000 板卡参数表				
编号	定义	描述	参数数据含义	默认值

10h	PRB_WDT0_VALUE	WDT超时值。	0: 禁用WDT N: 1~255 从N和倒数开始启动看门狗定时器。 递减计数周期为 WDT0_UNIT。计时器计数器 达到零（超时）时， WDT_ACTION 会发生。	0
11h	PRB_WDT0_COUNTER	重新启动WDT计数或获取当前WDT计数器值	为WDT0_VALUE设置任何值，以重启WDT计数器 获取命令以获取当前的WDT计数器。	0
12h	PRB_WDT0_UNIT	WDT计数器倒数周期单位	0: 保留 1: 秒 2: 分钟	0
13h	PRB_WDT0_ACTION	WDT超时动作	0: 系统重启	0
20h	PRB_TMR0_BASE	设置TMR0基本单元时钟	0~4095: TMR 值 计时器周期= (40 + (512 / 8.25) * TMR_value) us。 每次超时都会产生硬件中断。 要禁用计时器功能，必须禁用计时器中断	0
21h	PRB_TMR0_VALUE	获取/设置timer0值	32位无符号值。 计时器中断发生时，计数器每次加一	0
30h	PRB_SYS_TMP_MONITOR	获取系统温度监控数据	8位符号值。单位为摄氏度	0
31h	PRB_CPU_TMP_MONITOR	获取CPU温度监控数据	8位符号值。单位为摄氏度	0
32h	PRB_AUX_TMP_MONITOR	获取AUX温度监控数据	8位符号值。单位为摄氏度	0

40h	PRB_UART_MULTIPLIER	设置UART放大器	0 : x1模式。 如果波特率设置为115200 , 则实际波特率为115200。 1 : x8模式 如果波特率设置为115200 , 则实际波特率为115200 * 8 = 921600。	0
90h	PRB_PSR_MODE	设置手动脉冲发生器 (MPG) 输入模式	0: OUT/DIR 1: CW/CCW 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	4
91h	PRB_PSR_EA_LOGIC	设置EA信号逻辑	0: EA没有倒置 1: EA倒置	0
92h	PRB_PSR_EB_LOGIC	设置EB信号逻辑	0: EB没有倒置 1: EB倒置	0
10001h	PRB_DPAC_DISPLAY_MODE	DPAC 显示模式	0: 用户定义模式 1 : 演示模式	1
10002h	PRB_DPAC_DI_MODE	设置DI引脚模式	0: GPIO 模式 1: MPG输入模式	0

PCI-8392(H) 板卡参数表

PCI-8392(H) 板卡参数表				
编号	定义	描述	参数数据含义	默认值t
00h	PRB_EMG_LOGIC	EMG逻辑设置	0: 正常关闭 1: 正常打开	0

10h	PRB_WDT0_VALUE	WDT超时值。 (*1) 设置为0表示禁用看门狗。 设置一个正值以启用看门狗功能。 当启用看门狗定时器时，WDT计数器将在每个SSCNET周期递减计数。 一旦WDT计数器达到零（超时），SSCNET网络将被停止。	0: 禁用WDT N(1~2147483647) (31 位)	0
11h	PRB_WDT0_COUNTER	重新启动WDT计数或获取当前WDT计数器值. (*1)	为WDT0_VALUE设置任何值，以重启WDT计数器 获取命令以获取当前的WDT计数器。	0
10000h	PRB_SSC_CYCLE_TIME	SSCNET 3 通信周期时间设定	0: 0.888ms 1: 0.444ms 必须在开始SSCNET通信之前确定该值	0

(*1) 发布 “APS_save_parameter_to_flash” 时，此参数不会保存到非易失性存储器（闪存）中。

PCI-8253/56 板卡参数表

PCI-8253/56 板卡参数表				
编号	定义	描述	参数数据含义	默认值t
00h	PRB_EMG_LOGIC	EMG逻辑设置	0: 正常关闭 1: 正常打开	0

10h	PRB_WDTO_VALUE	WDT超时值。 (*1) 设置为0表示禁用看门狗。 设置一个正值表示启用看门狗功能。 当计时器计数器达到零(超时)时，伺服信号将被关闭。	0: 禁用WDT N: 1~2147483647 (31 位) 从N和倒数开始启动看门狗定时器。 递减计数周期是循环时间。	0
11h	PRB_WDTO_COUNTER	重新启动WDT计数或获取当前WDT计数器值.(*1)	为WDTO_VALUE设置任何值，以重启WDT计数器 获取命令以获取当前的WDT计数器。	0
80h	PRB_DENOMINATOR	分母	1~ 2147483647 浮点类型参数将被该值除以其实际值。	10,000
90h	PRB_PSR_MODE	设置手动脉冲发生器(MPG)输入模式	0: OUT/DIR 1: CW/CCW 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	4
100h	PRB_BOOT_SETTING	当DSP启动时，轴和系统参数的数据源。 通电或重置PCI总线时，DSP将重新启动。	0: 默认表 1: 快速闪存 (ROM)	0
110h	PRB_PWM0_MAP_DO	启用PWM0和Do之间的映射。	-1 : 禁用映射 正数 : 启用映射	-1
		指定一个Do通道来映射PWM0。在PWM0和Do之间选择其映射逻辑。	Bit0~7 : 指定一个Do通道。 Bit8 : 选择逻辑。 设置为1 : 打开Do映射启用PWM0。关闭Do映射禁用PWM0。设置为0 : 打开Do映射禁用PWM0。关闭Do映射启用PWM0。	

111h	PRB_PWM1_MAP_DO	启用PWM1和Do之间的映射。指定一个Do通道以映射PWM1。在PWM1和Do之间选择其映射逻辑。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个Do通道。 Bit8 : 选择逻辑。设置为1 : 打开Do映射启用PWM1。 关闭Do映射禁用PWM1。 设置为0 : 打开Do映射禁用PWM1。关闭Do映射启用PWM1。	-1
112h	PRB_PWM2_MAP_DO	启用PWM2和Do之间的映射。指定一个Do通道以映射PWM2。在PWM2和Do之间选择其映射逻辑。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个Do通道。 Bit8 : 选择逻辑。设置为1 : 打开Do映射启用PWM2。关闭Do映射禁用PWM2。设置为0 : 打开Do映射禁用PWM2。关闭Do映射启用PWM2。	-1
113h	PRB_PWM3_MAP_DO	启用PWM3和Do之间的映射。指定一个Do通道以映射PWM3。在PWM3和Do之间选择其映射逻辑。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个Do通道。 Bit8 : 选择逻辑。设置为1 : 打开Do映射启用PWM3。关闭Do映射禁用PWM3。设置为0 : 打开Do映射禁用PWM3。关闭Do映射启用PWM3。	-1

(*1) 发布 “APS_save_parameter_to_flash” 时，此参数不会保存到非易失性存储器（闪存）中。

PCI(e)-7856 板卡参数表

PCI(e)-7856 板卡参数表				
编号	定义	描述	参数数据含义	默认值
20h	PRB_TMR0_BASE	设置TMR基本单元时钟	0~127: TMR 值 计时器周期= ((TMR_value + 2) * 0.1) ms. 每次超时都会产生硬件中断。 要禁用计时器功能，必须禁用计时器中断	0

EMX-100 板卡参数表

EMX-100 板卡参数表				
编号	定义	描述	参数数据含义	默认值
51h(81)	PRB_DISCONNET_HANDLING	当网络断开时，设置伺服的处理方法	0: 停止 1: 伺服关闭	0

PCI-8254/58 / AMP-204/8C 板卡参数表

PCI-8254/58 / AMP-204/8C 板卡参数表				
编号	定义	描述	参数数据含义	默认值
00h	PRB_EMG_LOGIC	EMG 逻辑	0 : 不逆 1 : 逆	0
14h	PRB_DO_LOGIC	DO 逻辑	0 : 无反转； 1 : 反转	0
15h	PRB_DI_LOGIC	DI 逻辑	0 : 无反转； 1 : 反转	0
101h	PRS_EMG_MODE	EMG 条件模式	0 (EMO) : 直接关闭伺服 1 (EMS) : 紧急停机，不关闭伺服 2 : 忽略所有紧急处理，但仍将 FPGA EMG 状态更新为运动 IO 状态 3 : 忽略所有紧急情况处理，并停止将 FPGA EMG 状态更新为运动 IO 状态	0

110h	PRB_PWM0_MAP_DO	启用 PWM0 和 Do 之间的映射。指定一个 Do 通道来映射 PWM0。在 PWM0 和 Do 之间选择其映射逻辑。 注意：禁用此参数时，也可以禁用 VAO 表的 PWM 输出。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个 Do 通道。 Bit8 : 选择逻辑。 设置为 1 : 打开 Do 映射启用 PWM0。 关闭 Do 映射禁用 PWM0。 设置为 0 : 打开 Do 映射禁用 PWM0。关闭 Do 映射启用 PWM0。	-1
111h	PRB_PWM1_MAP_DO	启用 PWM1 和 Do 之间的映射。指定一个 Do 通道来映射 PWM1。在 PWM1 和 Do 之间选择其映射逻辑。 注意：禁用此参数时，也可以禁用 VAO 表的 PWM 输出。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个 Do 通道。 Bit8 : 选择逻辑。 设置为 1 : 打开 Do 映射启用 PWM1。 关闭 Do 映射禁用 PWM1。 设置为 0 : 打开 Do 映射禁用 PWM1。关闭 Do 映射启用 PWM1。	-1
112h	PRB_PWM2_MAP_DO	启用 PWM2 和 Do 之间的映射。指定一个 Do 通道来映射 PWM2。在 PWM2 和 Do 之间选择其映射逻辑。 注意：禁用此参数时，也可以禁用 VAO 表的 PWM 输出。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个 Do 通道。 Bit8 : 选择逻辑。 设置为 1 : 打开 Do 映射启用 PWM2。 关闭 Do 映射禁用 PWM2。 设置为 0 : 打开 Do 映射禁用 PWM2。关闭 Do 映射启用 PWM2。	-1
113h	PRB_PWM3_MAP_DO	启用 PWM3 和 Do 之间的映射。指定一个 Do 通道来映射 PWM3。在 PWM3 和 Do 之间选择其映射逻辑。 注意：禁用此参数时，也可以禁用 VAO 表的 PWM 输出。	-1 : 禁用映射 正数 : 启用映射 Bit0~7 : 指定一个 Do 通道。 Bit8 : 选择逻辑。 设置为 1 : 打开 Do 映射启用 PWM3。 关闭 Do 映射禁用 PWM3。 设置为 0 : 打开 Do 映射禁用 PWM3。关闭 Do 映射启用 PWM3。	-1

PCIe-833x 板卡参数表

PCIe-833x 板卡参数表				
编号	定义	描述	参数数据含义	默认值
00h	PRB_EMG_LOGIC	EMG 逻辑	0 : 不逆 1 : 逆	0
14h	PRB_DO_LOGIC	DO 逻辑	0 : 无反转； 1 : 反转	0
15h	PRB_DI_LOGIC	DI 逻辑	0 : 无反转； 1 : 反转	0
16h	PRB_IO_ACCESS_SEL	IO 访问选择	0 : 模式 0 1 : 保留 2 : 模式 2 (*备注 1)	0
17h	PRB_ECAT_SYNC_MODE	EtherCAT 同步模式选择	0 : 直流模式 (默认) 1 : 自由运行	0
18h	PRB_OP_RETRY_COUNT	EtherCAT 重试 OP 状态计数。		0
19h	PRB_ECAT_SERVO_ON_MODE	EtherCAT 伺服开启模式选择	0 : 标准模式 (默认) 1 : 快速模式，无检查状态字	0
1Ah	PRB_ECAT_SERVO_ON_NO_RESET_ALARM	EtherCAT 伺服器旁路复位报警	0 : 伺服开启时无旁路复位报警 (默认) 1 : 伺服开启时旁路复位报警	0
20h	PRB_ECAT_SYNC_OFFSET	DC 模式下的 EtherCAT 同步偏移百分比值。	单位 : 百分比 值的范围是 10 到 90。	66
101h	PRB_EMG_MODE	EMG 条件模式	0 (EMO) : 直接关闭伺服 1 (EMS) : 紧急停机，不关闭伺服	0
104h	PRB_ECAT_RESTORE_OUTPUT	保持 EtherCAT DIO/AIO 从站设备的状态。	0 : 不保留状态。 1 : 保持状态。 (默认) (*笔记 2)	1
0x105	PRB_DI_EMG_FILTER_ENABLE	板载 DI 和 EMG 信号滤波器的开关设置。	0 : 禁用 1 : 启用	1

0x10 6	PRB_DI_EMG_FILTER_RA NGE	板载 DI 和 EMG 信号滤波器的脉冲宽度设置。 如果输入信号的脉冲宽度小于该参数的设置值，则输入信号将被切断。	0: 5 uSec. 1: 10 uSec. 2: 20 uSec. 3: 40 uSec. 4: 80 uSec 5: 160 uSec	0
0x10 7	PRB_PULSER_FILTER_RA NGE	板载脉冲发生器信号滤波器的脉冲宽度设置。 如果输入信号的脉冲宽度小于该参数的设置值，则输入信号将被切断。（过滤器始终处于启用状态）	0: 5 uSec. 1: 10 uSec. 2: 20 uSec. 3: 40 uSec. 4: 80 uSec 5: 160 uSec	0
0x10 8	PRB_PULSER_FILTER_EN ABLE	脉冲发生器滤波器开关。	脉冲发生器滤波器开关。 0 : 禁用 1 : 启用	1
0x10 9	PRB_ECAT_AUTO_RECO VERY	EtherCAT 自动恢复功能	0 : 禁用 1 : 启用	1

备注 1:

如下所示的[表 1]用于在参数 **PRB_IO_ACCESS_SEL (0x16)** 的每个选择设置中标识 API 的相应行为和消耗时间。

[表 1]

	PRB_IO_ACCESS_SEL(0x16)= 0	PRB_IO_ACCESS_SEL(0x16)= 2
APS_set_field_bus_d_channel_output	模式: 同步	模式: 同步
APS_get_field_bus_d_channel_output	模式: 同步	模式 : 异步
APS_get_field_bus_d_channel_input	模式: 同步	模式 : 异步
	PRB_IO_ACCESS_SEL(0x16)= 0	PRB_IO_ACCESS_SEL(0x16)= 2
APS_set_field_bus_d_port_output	模式: 同步	模式: 同步
APS_get_field_bus_d_port_input	模式: 同步	模式 : 异步
APS_get_field_bus_d_port_output	模式: 同步	模式 : 异步
	PRB_IO_ACCESS_SEL(0x16)= 0	PRB_IO_ACCESS_SEL(0x16)= 2
APS_get_field_bus_a_input	模式: 同步	模式: 同步
APS_set_field_bus_a_output	模式: 同步	模式: 同步
APS_get_field_bus_a_output	模式: 同步	模式: 同步

备注 2 :

由于硬件限制，此参数 PRB_ECAT_RESTORE_OUTPUT 不支持 EU-6000 DO 模块。

B. 轴参数表

PCI-8392(H) 轴参数表

PCI-8392(H) 轴参数表				
编号	定义	描述	值	默认值
00h (0)	PRA_EL_LOGIC	PEL/MEL输入逻辑	0:不反向 1:反向	0
01h (1)	PRA_ORG_LOGIC	ORG输入逻辑	0:不反向 1:反向	0
02h (2)	PRA_EL_MODE	结束限制ON时的停止模式	0:减速停止 1:立即停止	0
03h (3)	PRA_MDN_COND1	运动完成条件 (运动统计信息NSTP位有效)	0 : 控制命令完成 (默认值) 1 : 使用 INP 完成命令 2 : 使用 ZSP 完成命令 3 : 使用 INP 和 ZSP 完成命令	0
04h (4)~ 06h(6)	Reserved	预留	预留	
07h(7)	PRA_STP_DEC	针对 APS_stop()的停止减速率 ;	单位: 脉冲/秒 ²	100,000, 000
08h(8)	PRA_SPEL_EN	设置编码器事件模式。	0: 禁用 1: 编码器事件 2: Soft-Limit	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。	0: 禁用 1: 编码器事件 2: Soft-Limit	0
0Ah(10)	PRA_EFB_POS0	SPEL / EFB 位置 0	单位: 脉冲(I32 值)	100,000
0Bh(11)	PRA_EFB_POS1	SMEL / EFB 位置 1	单位: 脉冲(I32 值)	-100,00 0
0Ch(12)	PRA_EFB_COND10	编码器比较条件。反馈位置>=或<= EFB pos0	0: 大于等于 (>=) 1: 小于 (<=)	0
0Dh(13)	PRA_EFB_COND11	编码器比较条件。反馈位置>=或<= EFB pos1	0: 大于等于 (>=) 1: 小于 (<=)	1
0Eh(14)	PRA_EFB_SRC0	编码器事件 pos0 比计数器源。	0: 反馈位置 1: 命令位置	0
0Fh(15)	PRA_EFB_SRC1	编码器事件 pos0 比计数器源。	0: 反馈位置 1: 命令位置	0

10h (16)	PRA_HOME_MODE	归零模式设定	0: 归零模式 1 0	
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0
12h (18)	PRA_HOME_CURVE	归零运动加速/减速速度模式	0: T-曲线 1: S-曲线	0
13h (19)	PRA_HOME_ACC	归零运动加/减速率	单位: 脉冲/秒 ²	22,520,00
14h (20)	PRA_HOME_VS	归零的开始速度	单位: 脉冲/秒	0
15h (21)	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	225,200
16h (22)	预留 (*1)			
17h (23)	预留 (*1)			
18h (24)	PRA_HOME_EZC	启用 EZ 信号校准	0: 禁用 1: 启用	0
19h (25)	PRA_HOME_VO	归零运动离开原点的速度	单位: 脉冲/秒	112,600
1Ah-1Fh	预留			
20h (32)	PRA_CURVE	加速/减速模式	0: T-曲线 1: S-曲线	0
21h (33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	10,000,00
22h (34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	10,000,00
23h (35)	PRA_VS	起始速度	单位: 脉冲/秒	0
24h (36)	预留			
25h (37)	PRA_VE	结束速度	单位: 脉冲/秒	0
26h~2Fh	预留			

30h	预留			
31h	预留			
32h~3Fh	预留			
40h (64)	PRA_JG_MODE	点动模式 0: 自由模式 1: 步进模式	0	
41h (65)	PRA_JG_DIR	点动方向 0: 正方向 1: 反方向	0	
42h (66)	PRA_JG_CURVE	点动速度模式 0: T-曲线 1:S-曲线	0	
43h (67)	PRA_JG_ACC	加速度 单位: 脉冲/秒 ²	10,000,000	
44h (68)	PRA_JG_DEC	减速度 单位: 脉冲/秒 ²	10,000,000	
45h (69)	PRA_JG_VM	最大速度 单位: 脉冲/秒	1,000,000	
46h (70)	PRA_JG_STEP	步进偏移 单位: 脉冲 (步进模式)	1,000	
47h (71)	PRA_JG_DELAY	延迟时间 单位: 毫秒 (周期时间校准) (步进模式)	500	
50h(80)	PRA_MDN_DELAY	运动完成延迟周期 (对运动统计信息NSTP位有效) 当符合运动完成条件时，运动状态 NSTP位将在指定的延迟周期后打开。	单位: 系统周期时间	0
51h(81)	PRA_SINP_WDW	软INP窗口设置 (对运动I/O状态INP位有效)。当位置进入软INP范围并且超过INP稳定周期时，运动I/O状态INP位将打开。 INP范围定义如下： INP范围= (目标+ window_setting) 到 (目标- window_setting)。 可以通过将PRA_MDM_CONDI参数设置为软INP命令来使用此功能。	单位：脉冲值=1~2147483647	200
52h(82)	PRA_SINP_STBL	软INP稳定周期 (对运动I/O状态INP位有效)。该值决定在定位成软INP范围连续性之后，打开INP位有多少个周期	单位: 系统周期时间 值 =1~2147483647	100

82h(130)	PRA_MAX_E_LIMIT	编码器最大数量。 2值=编码器数量限制 (*5)	单位：脉冲 0表示翻转模式，其他数字表示环形计数器值并启用环形计数器模式。	0
10000h	PRA_SSC_SERVO_PARA_M_SRC	启动SSCNET时选择伺服参数源	0：不更新 1：默认值 2：闪存	0
10001h	PRA_SSC_SERVO_ABS_P_O_S_OPT	启用绝对位置系统。	0：禁用。 1：启用绝对位置	0
10002h	PRA_SSC_SERVO_ABS_C_Y_CNT	伺服驱动器绝对周期计数器	0 ~ 65535 (16位)	0
10003h	PRA_SSC_SERVO_ABS_RES_CNT	伺服驱动器绝对分辨率计数器	0~262143 (18位)	0
10004h	PRA_SSC_TORQUE_LIM_IT_P	正转矩极限值 (0.1%) (*4)	0~32767	3,000
10005h	PRA_SSC_TORQUE_LIM_IT_N	负转矩极限值(0.1%) (*4)	0~32767	3,000
10006h	PRA_SSC_TORQUE_CTR_L	转矩控制启用 (*3)	0: 禁用, (电机最大转矩控制) 1: 启用, (转矩控制极限值)	0
10007h	PRA_SSC_RESOLUTION	电子齿轮系数 2值= 分辨率 (*5)	值 = 12~18 (分辨率 = 262144)	18
10008h	PRA_SSC_GMR	电子齿轮系数 分子 (*6)	值= 1~1000000	1
10009h	PRA_SSC_GDR	电子齿轮系数 分母(*6)	值e = 1~1000000	1

(*1): 不要设置任何参数数据。

(*2): 启动网络时重置为默认值。

(*3): 启动 SSCNET 网络时，某些 SSCNET 轴参数将恢复为默认值。

(*4) 0.1% 设定为 1000 意味着 100%

(*5): 重新启动 SSCNET 网络后，此参数有效。

(*6): PRA_SSC_RESOLUTION == 18 时以及重新启动 SSCNET 网络后，此参数有效。

$$\frac{1}{10} < \frac{PRA_SSC_GMR}{PRA_SSC_GDR} < 2000$$

PCI-8253/56 轴参数表.

PCI-8253/56 轴参数表.				
编号	定义	描述	值	默认值
00h	PRA_EL_LOGIC	PEL/MEL输入逻辑	0 : 不逆 1 : 逆	0
01h	PRA_ORG_LOGIC	ORG输入逻辑	0 : 不逆 1 : 逆	0
02h	PRA_EL_MODE	结束限制ON时的停止方式	0:减速停止 0:立即停止	1
03h	PRA_MDM_COND1	运动完成条件 (对运动统计信息NSTP位有效)	0: 控制命令完成 1 : 使用 INP 完成命令 2 : 使用 ZSP 完成命令 3 : 使用 INP 和 ZSP 完成命令 4 : 使用软 INP 完成命令	0
04h	PRA_ALM_LOGIC	设置ALM逻辑	0: 低电平有效 1: 高电平有效	0
05h	PRA_ZSP_LOGIC	设置ZSP逻辑	0: 低电平有效 1: 高电平有效	1
06h	PRA_EZ_LOGIC	设置EZ逻辑	0: 低电平有效 1: 高电平有效	0
07h	PRA_STP_DEC	APS_stop() 的停止减速率 ;	单位: 脉冲/秒 ²	100,000,00 0
08h	PRA_SPEL_EN	设置编码器事件模式。	0: 禁用 1: 编码器事件 2: Soft-Limit (SPEL)	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。	0: 禁用 1: 编码器事件 2: Soft-Limit(SMEL)	0
0Ah(10)	PRA_EFB_POS0	SPEL/ EFB 位置0	单位 : 脉冲。 (I32 值)	100,000
0Bh(11)	PRA_EFB_POS1	SMEL / EFB 位置 1	单位 : 脉冲。 (I32 值)	-100,000
0Ch(12)	PRA_EFB_COND10	编码器比较条件。 反馈位置 > = 或 < = EFB pos0	0: 大于等于 (>=) 1: 小于等于 (<=)	0
0Dh(13)	PRA_EFB_COND11	编码器比较条件。 反馈位置 > = 或 < = EFB pos1	0: 大于等于 (>=) 1: 小于等于 (<=)	1

0Eh(14)	PRA_EFB_SRC0	编码器事件 pos0 比计数器源。	0: 反馈位置 1: 命令位置	0
0Fh(15)	PRA_EFB_SRC1	编码器事件 pos0 比计数器源。	0: 反馈位置 1: 命令位置	0
10h (16)	PRA_HOME_MODE	归零模式设定	0: 归零模式 1 (ORG) 1: 归零模式 2 (EL) 2: 归零模式 3 (EZ)	0
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0
12h (18)	PRA_HOME_CURVE	归零运动加/减速度模式	0: T-曲线 1: S-曲线	0
13h (19)	PRA_HOME_ACC	归零运动 加/减速率	单位: 脉冲/秒 ²	22,520,000
14h (20)	PRA_HOME_VS	归零的开始速度	单位: 脉冲/秒	0
15h (21)	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	225,200
16h (22)	预留	(*1)		
17h (23)	预留	(*1)		
18h (24)	PRA_HOME_EZA	EZ 校准启用	0: 不启用 1: 启用	0
19h (25)	PRA_HOME_VO	归零运动离开原点的速度	单位: 脉冲/秒	112,600
1Ah-1Fh	预留	(*1)		
20h(32)	PRA_CURVE	加/减速度模式	0: T-曲线 1: S-曲线	0
21h(33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	10,000,000
22h(34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	10,000,000
23h(35)	PRA_VS	起始速度	单位: 脉冲/秒	0
24h(36)	预留	(*1)		
25h(37)	PRA_VE	结束速度	单位: 脉冲/秒	0
30h(48)	预留	(*1)		
31h(49)	预留	(*1)		
32h(50)	PRA_PT_STP_DO_EN	点表停止/暂停时启用 “执行”	0: 禁用 1: 启用	0
33h(51)	PRA_PT_STP_DO	点表停止时设定Do值	0: 设为0 1: 设为1	0

34h(52)	PRA_PWM_OFF	ASTP输入信号有效时，禁用指定的PWM输出。	0: 禁用. ASTP处于活动状态时，不执行任何操作。 1: ASTP有效时，PWM_CH0输出将被禁用。 2: ASTP有效时，PWM_CH1输出将被禁用。	0
35h(53)	PRA_DO_OFF	在ASTP输入信号有效时设置Do值。	0: 禁用. ASTP处于活动状态时，不执行任何操作。 Bit0~3: 选择DO通道 Bit8: 启用ASTP时设置Do输出值。 (*)	0
40h(64)	PRA_JG_MODE	点动模式	0: 自由模式 1: 步进模式	0
41h(65)	PRA_JG_DIR	点动方向	0: 正方向 1: 反方向	0
42h(66)	PRA_JG_CURVE	点动速度模式	0: T-曲线 1:S-曲线	0
43h(67)	PRA_JG_ACC	加速度	单位: 脉冲/秒 ²	10,000,00
44h(68)	PRA_JG_DEC	减速度	单位: 脉冲/秒 ²	10,000,00
45h(69)	PRA_JG_VM	最大速度	单位: 脉冲/秒	10,000
46h(70)	PRA_JG_STEP	步进偏移	单位: 脉冲 (步进模式)	1,000
47h(71)	PRA_JG_DELAY	延迟时间	单位: 毫秒 (周期时间校准) (用于步进模式)	800
50h(80)	PRA_MDN_DELAY	运动完成延迟周期 (对运动统计信息NSTP位有效)	单位: 系统周期时间	0
		当符合运动完成条件时，运动状态NSTP位将在指定的延迟周期后打开。		

51h(81)	PRA_SINP_WDW	软INP窗口设置（对运动I/O状态INP位有效）。当位置进入软INP范围并且超过INP稳定周期时，运动I/O状态INP位将打开。INP范围定义如下： INP范围=（目标+window_setting）到（目标>window_setting）。可以通过将PRA_MDM_COND1参数设置为软INP命令来使用此功能。	单位: 脉冲 值=1~2147483647	200
52h(82)	PRA_SINP_STBL	软INP稳定周期（对运动I/O状态INP位有效）。该值决定在定位成软INP范围连续性之后，打开INP位有多少个周期。	单位: 系统周期时间 值 =1~2147483647	100
80h(128)	PRA_PLS_IPT_MODE	脉冲输入模式	0: OUT/DIR 1: CW/CCW 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位(默认值)	4
81h(129)	预留	(*1)		
82h(130)	PRA_MAX_E_LIMIT	编码器最大数量	单位: 脉冲 0表示翻转模式， 其他数字表示环形计数器值并启用环形计数器模式	0
83h(131)	PRA_ENC_FILTER	编码器过滤器	0：禁用过滤器(默认值) 1：启用过滤器 (忽略小于80ns的信号)	0
84h(132)	PRA_EGEAR	电子齿轮系数=电机编码器分辨率 (112h)/值	值=1~电机编码器分辨率。	40,000
90h(144)	PRA_KP_GAIN	PID控制器Kp增益(*2,*3)	浮点数	500
91h(145)	PRA_KI_GAIN	PID控制器Ki增益(*2,*3)	浮点数	0
92h(146)	PRA_KD_GAIN	PID控制器Kd增益(*2,*3)	浮点数	0

93h(147)	PRA_KFF_GAIN	前馈Kff增益 (* 2 , * 3)	浮点数	0
94h(148)	PRA_KVGTY_GAIN	龙门Kvgty增益 (* 2 , * 3)	浮点数	0
95h(149)	PRA_KPGTY_GAIN	龙门Kpgty增益(*2, *3)	浮点数	0
96h(150)	PRA_IKP_GAIN	转矩模式下的PID控制器Kp增益(*2, *3)	浮点数	10
97h(151)	PRA_IKI_GAIN	转矩模式下的PID控制器Ki增益(*2, *3)	浮点数	0
98h(152)	PRA_IKD_GAIN	转矩模式下的PID控制器Kd增益(*2, *3)	浮点数	0
99h(153)	PRA_IKFF_GAIN	转矩模式下的前馈Kff增益(*2, *3)	浮点数	0
100h(256)	PRA_M_INTERFACE	运动接口	0: 模拟运动	0
110h(272)	PRA_M_VOL_RANGE	电机电压输入范围 (*3, *4)	输入值表示± (值) 伏	10
111h(273)	PRA_M_MAX_SPEED	电机最大转速(*3, *4)	单位 : RPS或毫米/秒	100 RPS
112h(274)	PRA_M_ENC_RES	电机编码器分辨率(*3, *4)	单位: 脉冲/转或脉冲/毫米	*40,000 脉冲/转
120h(288)	PRA_V_OFFSET	电压偏移(*2, *3)	单位 : 伏	0
121h(289)	PRA_DZ_LOW	盲区下侧 (*2, *3)	单位 : 伏	0
122h(290)	PRA_DZ_UP	盲区上侧(*2, *3)	单位 : 伏	0
123h(291)	PRA_SAT_LIMIT	电压饱和输出极限 (*2, *3)	单位 : 伏	100000
124h(292)	PRA_ERR_C_LEVEL	错误计数器检查级别	如果设为0 , 则表示不检查错误。 其他值表示错误检查然后停止	90000
125h(293)	PRA_V_INVERSE	输出逆电压	0: 非逆 1: 逆	0
126h(294)	PRA_DZ_VAL	分配盲区输出值(*2, *3)	单位 : 伏	0
127h(295)	PRA_IW_MAX	积分饱和上限值	单位: 脉冲 (值必须输入正值)	45000

128h(296)	PRA_IW_MIN	积分饱和下限值	单位: 脉冲 (值必须输入正值)	45000
129h(297)	PRA_BKL_DIST	使用此参数定义背隙长度。 如果设置为零，则背隙补偿功能将关闭。	单位: 脉冲	0
12Ah(298)	PRA_BKL_CNSP	此参数将定义背隙补偿的消耗值。因为背隙补偿器会在每个周期消耗脉冲，直到背隙距离用完为止。 并且用户在使用背隙功能之前，必须确保运动方向保持初始的状态（方向初始状态为负，因此用户正向运动会触发背隙补偿设备输出脉冲）。	单位: 脉冲	0
130h(304)	PRA_PSR_LINK	连接脉冲发生器	0: 禁用, 1: 启用	0
131h(305)	PRA_PSR_RATIO	脉冲比	值 = 1 ~ 2147483647	1
140h(320)	PRA_DA_TYPE	DAC 输出类型	0: 差分输出 1: 单路输出	0
141h(321)	PRA_CONTROL_MODE	闭环控制模式 (* 3)	0: 速度控制闭环 1: 转矩控制闭环	0

*1: 不要设置任何参数数据。

*2: 通过设置系统参数 80h 来更改单位，如果用户要在程序中更改单位，请记住在设置系统参数 80h 之后重新设置参数。

*3: 在使用模拟运动接口之前，请提供正确的值。

*4: 此参数用于计算速度单位与电压单位之间的比率

*5: 参数值详细说明：

7	6	5	4	3	2	1	Bit : 0
-	-	-	-	1~8 :DO_CH0~ DO_CH7			
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	ON/OFF

PCI-8144 轴参数表

PCI-8144 轴参数表.				
编号	定义	描述	值	默认值
00h	PRA_EL_LOGIC	限制输入逻辑	0: 正逻辑	1

			1: 负逻辑	
81h	PRA_PLS_OPT_MODE	脉冲输出方式 (模式) 选择。 (逻辑)	0 = CW/CCW 1 = CW/CCW (逻辑逆) 2 = OUT/DIR 3 = OUT/DIR (逻辑逆)	0
11h	PRA_HOME_DIR	归零方向	0: 正方 向 1: 反方 向	0
15h	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	1000
1Ah	PRA_ORG_STP	当ORG输入打开时，运动停止。	0: 禁用 1: 启用	1
20h	PRA_CURVE	加/减速度模式 更改此参数将影响运动参数， 包括PRA_ACC	0: T-曲线 1: S-曲线	0
21h	PRA_ACC	加速度 / 减速度。 如果ACC = 0，轴馈线为起始速度，如果ACC <0，轴馈线为最 大速度	单位: 脉冲/s^2	99903
22h	预留			
23h	PRA_VS	起始速度	单位: 脉冲/s	10
81h	PRA_PLS_OPT_MODE	脉冲输出方式 (模式) 选择。 (逻辑)	0 = CW/CCW 1 = CW/CCW (逻辑逆) 2 = OUT/DIR 3 = OUT/DIR (逻辑逆)	0
212h	PRA_SD_EN	当SD输入打开时，启用减速。	0: 禁用 1: 启用	0
240h	PRA_SPD_LIMIT	可能最大的轴操作速度。 更改此参数将影响其他运动 参数，包括PRA_ACC， PRA_VS	单位: 脉冲/秒	409550
10000h	PRA_CMD_CNT_EN	启用软指定计数器。	0: 禁用 1: 启用	0
10001h	PRA_MIO SEN	运动I/O : ORG , EL , STP输入灵敏度设置。	0: 高灵敏度 1: 低灵敏度	0

10002h	PRA_START_STA	通过外部输入引脚STA , 启动 (触发) 运动.	0: 禁用 1: 启用	0
10003h	PRA_SPEED_CHN	(仅设置) 设置更改速度命令。	1: 将速度更改 为初始速度 0: 将速度更改 为最大速度	0

MNET-4XMO-(C) 轴参数表.

MNET-4XMO-(C) 轴参数表.				
编号	定义	描述	值	默认值
00h (0)	PRA_EL_LOGIC	PEL/MEL输入逻辑	0 : 不逆 1 : 逆	0
01h (1)	PRA_ORG_LOGIC	ORG输入逻辑	0:低电平有效 1:高电平有效	0
02h (2)	PRA_EL_MODE	结束限制ON时的停止方式	0:减速停止 0:立即停止	0
03h (3)	PRA_MDN_CONDITI	运动完成条件 (对运动统计信息NSTP位有效)	0 : 控制命令完成 (默认值) 1 : 使用INP完成命令	0
04h (4)	PRA_ALM_LOGIC	设置ALM逻辑	0:低电平有效 1:高电平有效	0
05h(5)	预留			
06h(6)	PRA_EZ_LOGIC	设置EZ逻辑	0: 下降沿 1: 上升沿	0
07h(7)	预留			
08h	PRA_SPEL_EN	设置编码器事件模式。 (*3,)	0: 禁用 1: 预留 2: Soft-Limit (SPEL) 注意 : 模式1是已预留。如果设置，则返回错误。	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。 (*3,)	0: 禁用 1: 预留 2: Soft-Limit (SMEL) 注意 : 模式1是已	0

			预留。如果设置，则返回错误。	
0Ah(10)	PRA_EFB_P OS0	SPEL / EFB 位置0 (*3,)	单位：脉冲。(I32 值) (28位，带符号)	100,000
0Bh(11)	PRA_EFB_P OS1	SMEL / EFB 位置1 (*3,)	单位：脉冲。(I32 值) (28位，带符号)	-100,000
0Ch(12)	预留			
0Dh(13)	预留			
0Eh(14)	预留			
0Fh(15)	预留			
10h (16)	PRA_HOME _MODE	归零模式设定	原点搜索- 0 (第一 种模式) 至12 (第 13模式) 原点搜索- 20 (第 一种模式) 至32 (第 13种模式) 注意: 归零搜寻 (6 至8) 已预留。如果 设置，则返回错 误。	0
11h (17)	PRA_HOME _DIR	归零方向	0: 正方向 1: 反方向	0
12h (18)	预留			
13h (19)	预留			
14h (20)	预留			
15h (21)	PRA_HOME _VM	归零的最大速度	单位: 脉冲/秒	10000
16h (22)	预留			
17h (23)	预留			
18h (24)	PRA_HOME _EZA	指定EZ计数值	0000 (第1个计 数) 至1111 (第 16个计数)	0
19h (25)	PRA_HOME _VO	归零运动离开原点的速度 - 指 定FA速度	单位: 脉冲/秒	152
1Ah	PRA_HOME _OFFSET	归零运动离开原点的距离- 指 定ORG偏移	单位: 脉冲	100
1Bh-1Fh	预留			

20h (32)	PRA_CURVE	加/减速度模式(*4)	0: T-曲线 1: S-曲线	0
21h (33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	1000000
22h (34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	1000000
23h (35)	PRA_VS	起始速度	单位: 脉冲/秒	152
24h~26h	预留			
28h	PRA_ACC_SR	S曲线加速度比 (* 4)	单位 : 毫%。 值= 1 ~ 100,000	100,000
29h	PRA_DEC_SR	S曲线减速比 (* 4)	单位 : 毫%。 值= 1 ~ 100,000	100,000
2Ah~50h	预留			
53h(83)	PRA_SERVO_LOGIC	SERVO输出逻辑	0:低电平有效 1:高电平有效	0
80h(128)	PRA_PLS_IPT_MODE	脉冲输入模式	0: A/B X1 1: A/B X2 2: A/B X4 3: CW/CCW	0
81h(129)	PRA_PLS_OPT_MODE	脉冲输出模式	0: OUT/DIR (AL,H+) 1: OUT/DIR (AH,H+) 2: OUT/DIR (AL,L+) 3: OUT/DIR (AH,L+) 4: CW/CCW (AH) 5: CW/CCW (AL) 6: AB (领先) 7: AB (滞后)	0
84h(132)	PRA_EGEAR	电子齿轮系数 = 电机编码器分辨率 (112h) /值 (*1,)	值= 1 ~ 电机编码器分辨率.	40,000
112h(274)	PRA_M_ENC_RES	电机编码器分辨率 (*1,)	单位: 脉冲/转或脉冲/毫米	40,000
200h(512)	PRA_PLS_IPT_LOGIC	脉冲输入逻辑	0 : 不反向计数方向 1 : 反向计	0

			数方向	
201h(51 3)	PRA_FEEDB ACK_SRC	选择反馈源	0: 扩展编码器模式 扩展编码器计数器 和绝对模式参考编	0
			1 : 步进模式 扩展指定计数器 & 绝对模式参考指定 计数器。 2 : ACServo模式 扩展编码器计数器 和绝对模式参考指 定计数器。	
210h(52 8)	PRA_ALM_ MODE	ALM模式设定	0 : 立即停止 1 : 减速然后停止	0
211h(52 9)	PRA_INP_L OGIC	INP输入逻辑	0:低电平有效 1:高电平有效	0
212h(53 0)	PRA_SD_EN	启用 SD. (*2)	0: 禁用 1: 启用	0
213h(53 1)	PRA_SD_M ODE	SD模式设定	0 : 仅减速 1 : 减速并停止	0
214h(53 2)	PRA_SD_LO GIC	SD输入逻辑	0:低电平有效 1:高电平有效	0
215h(53 3)	PRA_SD_LA TCH	锁存SD输入	0 : 禁用锁存功 能 1 : 启用锁存功 能	0
216h(53 4)	PRA_ERC_M ODE	ERC模式设定	0: 禁用 1 : 当被EL、 ALM或EMG输 入停止时，输 出ERC 2 : 当完成归零 后，输出 ERC， 3 : 1和2	3
217h(53 5)	PRA_ERC_L OGIC	ERC输出逻辑	0:低电平有效 1:高电平有效	0
218h(53 6)	PRA_ERC_LE N	ERC设定的脉冲宽度	0: 12 us 1: 102 us	3

			2: 409 us 3: 1.6 ms 4: 13 ms 5: 52 ms 6: 104 ms 7: 电平输出	
219h(53 7)	预留			
21Ah(53 8)	预留			
21B(539)	PRA_PLS_IP T_FLT	启用EA/EB过滤器	0: 禁用 1: 启用	1
21C	预留			
21D	PRA_LTC_L OGIC	LTC输入逻辑	0: 下降沿 1: 上升沿	0
21E	PRA_IO_FIL TER	为PEL , MEL , SD , ORG , ALM , INP输入申请过滤器。 当过滤器启用时，短于4微妙的 信号脉冲将被忽略。	0 : 不申请过滤器 1 : 申请过滤器	1
21F~220	预留			
221	PRA_ COMPENSA TION _PULSE	背隙或滑差校正量	0 至 4095	0
222	PRA_ COMPENSA TION _MODE	背隙或滑差模式设定	0: 禁用 1: 背隙校正 2: 滑差校正	0
223	PRA_LTC_S RC	2: 滑差校正	0: LTC pin 1: ORG pin 2: GCMP ON	0
224	PRA_LTC_D EST	选择锁存目标	0: 指定计数器 1 : 位置计数器	0
225	PRA_LTC_D ATA	获取锁存数据 (仅读取)	脉冲 (28位 , 带符号)	0
226	PRA_GCMP _EN	通用比较器启用和设置方 法	0: 禁用 其他： 启用 1 : 数据= cmp计	0

			数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据 = cmp计数器 (下计数) 4: 数据> CMP计数器 5: 数据< CMP计数器	
227	PRA_GCMP_POS	通用比较器位置	脉冲 (28位, 带符号)	0
228	PRA_GCMP_SRC	选择通用比较器源	0: 指定计数器 1 : 位置计数器	0
229	PRA_GCMP_ACTION	符合GCMP时选择操作。	0 : 什么都不做 1 : 立即停止 2 : 减速停止	0
22A	PRA_GCMP_STS	检查GCMP是否符合 (仅读取)	0: 不符合 1: 符合	0
22B	PRA_VIBSU_P_RT	减振-逆转时间	单位: 1.6 us (16位, 不带符号)	0
22C	PRA_VIBSU_P_FT	减振-前进时间	单位: 1.6 us (16位, 不带符号)	0
22D	PRA_LATCH_DATA_SPD	为2号锁存器选择错误位置或当前速度的锁存器数据	0 : 锁存错误位置 1 : 锁存当前速度	0
22E~230	预留			
231	PRA_GPDI_SEL	选择GPIO输入- DI / LTC / SD. (*2)	0: DI 1: LTC 2: SD	0
232	预留			
233(563)	PRA_RDY_LOGIC	RDY输入逻辑	0: 高电平有效 1: 低电平有效	0
234h~23Fh	预留			
240h(576)	PRA_SPD_LIMIT	设定固定速度	单位: 脉冲/秒	9999847
241h(57)	PRA_MAX_	获取最大加速度/减速度, 该速	单位: 脉冲/秒2	57220458

7)	ACCDEC	度受固定速度限制。 (仅读取) (*5)		9
242h(57 8)	PRA_MIN_A CCDEC	获取最小加速度/减速度，该速 度受固定速度限制。 (仅读取) (*5)	单位: 脉冲/秒2	17462
260h(60 8)	PRA_SYNC_ STOP_MOD E	当停止同动时，设置停止模式	0 : 立即停止 1 : 减速停止	0

*1: 此参数用于计算运动比率。仅当 PRA_FEEDBACK_SRC 为 0 或 2 时有效。

*2: 当 PRA_GPDI_SEL 设置为 DI/LTC 时，PRA_SD_EN 自动设置为禁用。在 PRA_SD_EN 设置为启用之前，请确保 PRA_GPDI_SEL 设置为 SD 模式。

*3: 当选择正/负软件限位时，将指定计数器用作比较计数器。比较方法如下：

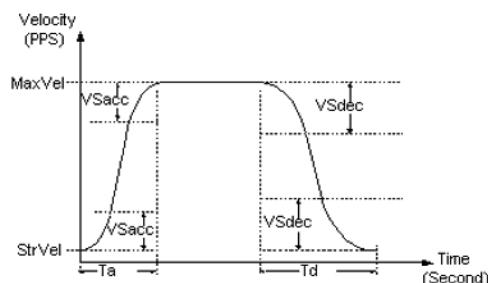
(EFB 位置 0 < 指定计数器) 为正软件限位(EFB 位置 1 > 指定计数器) 为负软件限位。

*4: 如果 PRA_ACC_SR 和 PRA_DEC_SR 设置为 100,000，则表示曲线轮廓为纯 S 曲线。如果 PRA_ACC_SR 或 PRA_DEC_SR 不等于 100,000，则表示曲线轮廓为具有线性范围的 S 曲线。

公式如下所示：

$$\text{PRA_ACC_SR} = \\ 2S_{vacc} / (\text{MaxV} - \text{StrV}) * 100,000 \text{ milli\%}$$

$$\text{PRA_DEC_SR} = \\ 2S_{vdec} / (\text{MaxV} - \text{StrV}) * 100,000 \text{ milli\%}$$



*5: 根据 (*4)，当曲线轮廓设置为具有线性范围的 S 曲线时，PRA_MAX_ACCDEC 和 PRA_MIN_ACCDEC 始终返回 0。PRA_MAX_ACCDEC 和 PRA_MIN_ACCDEC 仅在 T 曲线和纯 S 曲线模式下可用。

MNET-1XMO 轴参数表

MNET-1XMO 轴参数表.				
编号	定义	描述	值	默认值
00h (0)	预留			
01h (1)	PRA_ORG_LOGIC	ORG输入逻辑	0:低电平有效 1:高电平有效	0
02h (2)	PRA_EL_MODE	结束限制 ON时的停	0:减速停止 0:立即停止	0

		止方式		
03h (3)	PRA_MDN_CONDI	运动完成条件（对运动统计信息NSTP位有效）	0 : 控制命令完成 （默认值） 1 : 使用INP完成命令	0
04h (4)	PRA_ALM_LOGIC	设置ALM逻辑	0:低电平有效 1:高电平有效	1
05h(5)	预留			
06h(6)	PRA_EZ_LOGIC	设置EZ逻辑	0: 下降沿 1: 上升沿	0
07h(7)	预留			
08h	PRA_SPEL_EN	设置编码器事件模式。 (*2,)	0: 禁用 1: 预留 2: Soft-Limit (SPEL)	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。 (*2,)	0: 禁用 1: 预留 2: Soft-Limit (SMEL)	0
0Ah(10)	PRA_EFB_POS0	SPEL / EFB 位置0 (*2,)	单位：脉冲。 (I32 值) (28位 , 带符号)	100,000
0Bh(11)	PRA_EFB_POS1	SMEL / EFB 位置1 (*2,)	单位：脉冲。 (I32 值) (28位 , 带符号)	-100,000
0Ch(12)	预留			
0Dh(13)	预留			
0Eh(14)	预留			
0Fh(15)	预留			
10h (16)	PRA_HOME_MODE	归零模式设定	原点搜索- 0 (第一个模式) 至12 （ 13个模式 ） 归零运动- 20 （ 第一个模式 ） 至32 (第13个模式) 注意：原点搜索 (6 至8) 已预留。如果	0

			设置，则返回错误。	
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0
12h (18)	预留			
13h (19)	预留			
14h (20)	预留			
15h (21)	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	10000
16h (22)	预留			
17h (23)	预留			
18h (24)	PRA_HOME_EZA	指定EZ计数值	0000 (第1个计数) 至1111 (第16个计数)	0
19h (25)	PRA_HOME_VO	归零运动离开原点的速度 - 指定FA速度	单位: 脉冲/秒	0
1Ah	PRA_HOME_OFFSET	归零运动离开原点的距离-指定ORG偏移	单位: 脉冲	100
1Bh-1Fh	预留			
20h (32)	PRA_CURVE	加/减速度模式	0: T-曲线 1: S-曲线	0
21h (33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	1000000
22h (34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	1000000
23h (35)	PRA_VS	起始速度	单位: 脉冲/秒	66
24h ~ 27h	预留			
28h	PRA_ACC_SR	S曲线加速度比 (* 4)	单位 : 毫%。 值 = 1 ~ 100,000	100,000
29h	PRA_DEC_SR	S曲线减速比 (* 4)	单位 : 毫%。 值 = 1 ~ 100,000	100,000
2Ah~50h	预留			
53h(83)	PRA_SERVO_LOGIC	SERVO 输出逻辑	0:低电平有效 1:高电平有效	0
80h(128)	PRA_PLS_IPT_MODE	脉冲输入模式	0: A/B X1 1: A/B X2 2: A/B X4 3: CW/CCW	0
81h(129)	PRA_PLS_OPT_MODE	脉冲输出模式	0: OUT/DIR	0

			(AL,H+) 1: OUT/DIR (AH,H+) 2: OUT/DIR (AL,L+) 3: OUT/DIR (AH,L+) 4: CW/CCW (AH) 5: CW/CCW (AL) 6: AB (领先) 7: AB (滞后)	
84h(132)	PRA_EGEAR	电子齿轮系数 = 电机编码器分辨率 (112h) /值 (*1,)	值 = 1 ~ 电机编码器分辨率.	40,000
112h(274)	PRA_M_ENC_RES	电机编码器分辨率 (*1,)	单位: 脉冲/转或脉冲/毫米	40,000
200h(512)	PRA_PLS_IPT_LOGIC	脉冲输入逻辑	0 : 不反向EA/EB计数 1 : 反向EA/EB计数	0
201h(513)	PRA_FEEDBACK_SRC	选择反馈源	0: 扩展编码器模式 扩展编码器计数器和绝对模式参考编码器计数器。 1 : 步进模式 扩展指定计数器 & 绝对模式参考指定计数器。 2 : ACServo模式 扩展编码器计数器和绝对模式参考指定计数器。	0
210h(528)	PRA_ALM_MODE	ALM模式设定	0 : 立即停止 1 : 减速然后停止	0
211h(529)	PRA_INP_LOGIC	INP输入逻辑	0:低电平有效 1:高电平有效	0
212h(530)	PRA_SD_EN	启用SD	0: 禁用 1: 启用	0
213h(531)	PRA_SD_MODE	SD模式设定	0 : 仅减速	0

			1 : 减速并停止	
214h(532)	PRA_SD_LOGIC	SD输入逻辑	0:低电平有效 1:高电平有效	0
215h(533)	PRA_SD_LATCH	锁存SD输入	0 : 禁用锁存功能 1 : 启用锁存功能	0
216h(534)	PRA_ERC_MODE	ERC模式设定	0: 禁用 1 : 当被EL , ALM或EMG输入停止时 , 输出ERC 2 : 完成归零后 , 输出ERC 3 : 1和2	3
217h(535)	PRA_ERC_LOGIC	ERC输出逻辑	0:低电平有效 1:高电平有效	0
218h(536)	PRA_ERC_LEN	ERC设定的脉冲宽度	0: 12 us 1: 102 us 2: 409 us 3: 1.6 ms 4: 13 ms 5: 52 ms 6: 104 ms 7: 电平输出	3
219h(537)	预留			
21Ah(538)	预留			
21B(539)	PRA_PLS_IPT_FLT	启用EA/EB过滤器	0: 启用 1: 禁用	1
21C	预留			
21D	PRA_LTC_LOGIC	LTC输入逻辑	0: 下降沿 1: 上升沿	0
21E	PRA_IO_FILTER	为PEL , MEL , SD , ORG , ALM , INP输入申请过滤器。当过滤器启用时 , 短于4微妙的信号脉冲将被忽略。	0 : 不申请过滤器 1 : 申请过滤器	1
21F~220	预留			

221	PRA_COMPENSATION_PU LSE	背隙校正 量	0 至 4095	0
222	PRA_COMPENSATION _MODE	背隙模式设定	0: 禁用 1: 背隙校正	0
223	PRA_LTC_SRC	2: 滑差校正	0: LTC pin 1: ORG pin 2: GCMP ON	0
224	PRA_LTC_DEST	选择锁存目标	0: 指定计数器 1: 位置计数器	0
225	PRA_LTC_DATA	获取锁存数据 (仅读取)	脉冲 (28位 , 带符号)	0
226	PRA_GCMP_EN	通用比较器启 用和设置方法	0: 禁用 其他： 启用 1 : 数据= cmp计 数器 (与计数方向 无关) 2 : 数据= cmp 计数器 (上计 数) 3 : 数据 = cmp计 数器 (下计数) 4: 数据> CMP计 数器 5: 数据< CMP计 数器	0
227	PRA_GCMP_POS	一般比较器 位置	脉冲 (28位 , 带符 号)	0
228	PRA_GCMP_SRC	选择通用比较器源	0: 指定计数器 1: 位置计数器	0
229	PRA_GCMP_ACTION	符合GCMP时选择 操作。	0: 什么都不做 1: 立即停止 2: 减速停止	0
22A	PRA_GCMP_STS	检查GCMP是否符 合 (仅读取)	0: 不符合 1: 符合	0
22B	PRA_VIBSUP_RT	减振- 逆转 时间	单位: 1.6 us (16位 , 不带符号)	0
22C	PRA_VIBSUP_FT	减振- 前进 时间	单位: 1.6 us (16位 , 不带符号)	0

22D	PRA_LATCH_DATA_SPD	为2号锁存器选择错误位置或当前速度的锁存器数据	0 : 锁存错误位置 1 : 锁存当前速度	0
22F ~23F	预留			
240h(576)	PRA_SPD_LIMIT	设定固定速度	单位: 脉冲/秒	6666666
241h(577)	PRA_MAX_ACCDEC	获取最大加速度/减速度，该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	16666666 6
242h(578)	PRA_MIN_ACCDEC	获取最小加速度/减速度，该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	5086

*1: 此参数用于计算运动比率。仅当 PRA_FEEDBACK_SRC 为 0 或 2 时有效。

*2: 当选择正/负软件限位时，将指定计数器用作比较计数器。 比较方法如下：

(EFB 位置 0 < 指定计数器) 为正软件限位(EFB 位置 1 > 指定计数器) 为负软件限位。

HSL-4XMO 轴参数表.

HSL-4XMO 轴参数表.				
编号	定义	描述	值	默认值
00h (0)	PRA_EL_LOGIC	PEL/MEL输入逻辑	0 : 不逆 1 : 逆	0
01h (1)	PRA_ORG_LOGIC	ORG输入逻辑	0:低电平 有效1:高 电平有效	0
02h (2)	PRA_EL_MODE	结束限制ON时的停止 方式	0:减速停止 0:立即停止	0
03h (3)	预留			
04h (4)	PRA_ALM_LOGIC	设置ALM逻辑	0:低电平有效 1:高电平有效	0
05h(5)	预留			
06h(6)	PRA_EZ_LOGIC	设置EZ逻辑	0: 下降沿 1: 上升沿	0
07h(7)	预留			
08h	PRA_SPEL_EN	设置编码器事件模式。	0: 禁用 1: 预留 2: Soft-Limit (SPEL)	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。	0: 禁用 1: 预留 2: Soft-Limit (SMEL)	0
0Ah(10)	PRA_EFB_POS0	SPEL / EFB 位置0	单位：脉冲。(I32 值)	100,000

			范围: -10^8 ~ 10^8	
0Bh(11)	PRA_EFB_POS1	SMEL / EFB 位置1	单位 : 脉冲。 (I32 值) 范围: -10^8 ~ 10^8	-100,000
0Ch(12)	预留			
0Dh(13)	预留			
0Eh(14)	预留			
0Fh(15)	预留			
10h (16)	PRA_HOME_MODE	归零模式设定	原点搜索- 0 (第一个模式) 至 12 (13 个模式) 归零运动- 20 (第一个模式) 至 32 (第13个模式) 注意 : 原点搜索 (6至 8) 已预留。如果设置 , 则返回错误。	0
11h (17)	预留			
12h (18)	预留			
13h (19)	预留			
14h (20)	预留			
15h (21)	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	10000
16h (22)	预留			
17h (23)	预留			
18h (24)	PRA_HOME_EZA	指定EZ计数值	0000 (第1个计数) 至 1111 (第16个计数)	0
19h (25)	PRA_HOME_VO	归零运动离开原点的速度 - 指定FA速度	单位: 脉冲/秒	100
1Ah	PRA_HOME_OFFSET	归零运动离开原点的距离-指定ORG偏移	单位: 脉冲	100
1Bh-1Fh	预留			
20h (32)	PRA_CURVE	加/减速度模式	0: T-曲线 1: S-曲线	0
21h (33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	1000000
22h (34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	1000000
23h (35)	PRA_VS	起始速度	单位: 脉冲/秒	100
24h~50h	预留			
53h(83)	PRA_SERVO_LOGIC	SERVO 输出逻辑	0:低电平有效 1:高电平有效	0
80h(128)	PRA_PLS_IPT_MODE	脉冲输入模式	0: A/B X1 1: A/B X2	0

			2: A/B X4 3: CW/CCW	
81h(129)	PRA_PLS_OPT_MODE	脉冲输出模式	0: OUT/DIR (AL,H+) 1: OUT/DIR (AH,H+) 2: OUT/DIR (AL,L+) 3: OUT/DIR (AH,L+) 4: CW/CCW (AH) 5: CW/CCW (AL)	0
84h(132)	PRA_EGEAR	电子齿轮系数 = 电机编码器分辨率 (112h) /值 (*1,)	值= 1 ~ 电机编码器分辨率.	40,000
112h(274)	PRA_M_ENC_RES	电机编码器分辨率 (*1,)	单位: 脉冲/转或脉冲/毫米	40,000
200h(512)	PRA_PLS_IPT_LOGIC	脉冲输入逻辑	0 : 不反向EA/EB计数 1 : 反向EA/EB计数	0
201h(513)	PRA_FEEDBACK_SRC	选择反馈源	0 : 编码器计数器和绝对模式参考编码器计数器。 1: 指定计数器&绝对模式参考指定计数器。 2: 编码器计数器和绝对模式参考指定计数器。	0
210h(528)	PRA_ALM_MODE	ALM模式设定	0 : 立即停止 1 : 减速然后停止	0
211h(529)	PRA_INP_LOGIC	INP输入逻辑	0:低电平有效 1:高电平有效	0
212h(530)	PRA_SD_EN	启用 SD. (*2)	0: 禁用 1: 启用	0
213h(531)	PRA_SD_MODE	SD模式设定	0 : 仅减速 1 : 减速并停止	0
214h(532)	PRA_SD_LOGIC	SD输入逻辑	0:低电平有效 1:高电平有效	0
215h(533)	PRA_SD_LATCH	锁存SD输入	0 : 禁用锁存功能 1 : 启用锁存功能	0
216h(534)	PRA_ERC_MODE	ERC模式设定	0: 禁用 1 : 当被EL , ALM 或EMG输入停止时 , 输出ERC 2 : 完成归零后 , 输出ERC 3 : 1和2	3
217h(535)	PRA_ERC_LOGIC	ERC输出逻辑	0:低电平有效 1:高电平有效	0
218h(536)	PRA_ERC_LEN	ERC设定的脉冲宽度	0: 12 us 1: 102 us	3

			2: 409 us 3: 1.6 ms 4: 13 ms 5: 52 ms 6: 104 ms 7: 电平输出	
219h(537)	预留			
21Ah(538)	预留			
21Bh(539)	预留			
21Ch	预留			
21Dh	PRA_LTC_LOGIC	LTC输入逻辑	0: 下降沿 1: 上升沿	0
21Eh~220h	预留			
221h	PRA_COMPENSATION_脉冲	背隙或滑差校正量	0 至 4095	0
222h	PRA_COMPENSATION_MODE	背隙或滑差模式设定	0: 禁用 1: 背隙校正 2: 滑差校正	0
223h	PRA_LTC_SRC	2: 滑差校正	0: LTC pin 1: ORG pin 2: GCMP ON	0
224h	PRA_LTC_DEST	选择锁存目标	0: 指定计数器 1: 位置计数器	0
225h	PRA_LTC_DATA	获取锁存数据 (仅读取)	脉冲 (28位 , 带符号)	0
226h	PRA_GCMP_EN	通用比较器启用和设置方法	0: 禁用 其他 : 启用 1 : 数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据 = cmp计数器 (下计数) 4: 数据> CMP计数器 5: 数据< CMP计数器	0
227h	PRA_GCMP_POS	设置/获取通用比较器位置	脉冲 (28位 , 带符号)	0
228h	PRA_GCMP_SRC	选择通用比较器源	0: 指定计数器 1 : 位置计数器	0
229h	PRA_GCMP_ACTION	符合GCMP时选择操作。	0 : 什么都不做 1 : 立即停止 2 : 减速停止	0

22Ah	PRA_GCMP_STS	检查GCMP是否符合 (仅读取)	0: 不符合 1: 符合	0
22Bh	PRA_VIBSUP_RT	减振-逆转时间	单位: 1.6 us (16位, 不带符号)	0
22Ch	PRA_VIBSUP_FT	减振-前进时间	单位: 1.6 us (16位, 不带符号)	0
22Dh~22Fh	预留			
230h	预留			
231h	PRA_GPDI_SEL	选择GPIO输入- DI / LTC / SD. (*2)	0: LTC 1: SD	0
232h	预留			
233h(563)	PRA_RDY_LOGIC	RDY输入逻辑	0: 高电平有效 1: 低电平有效	0
234h~23Fh	预留			
240h(576)	PRA_SPD_LIMIT	设定固定速度	单位: 脉冲/秒	6553500
241h(577)	PRA_MAX_ACCDEC	获取最大加速度/减速度，该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	24576000 0
242h(578)	PRA_MIN_ACCDEC	获取最小加速度/减速度，该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	7500

*1: 此参数用于计算运动比率。仅当 PRA_FEEDBACK_SRC 为 0 时有效。

*2: 当 PRA_GPDI_SEL 设置为 DI/LTC 时, PRA_SD_EN 自动设置为禁用。在 PRA_SD_EN 设置为启用之前, 请确保 PRA_GPDI_SEL 设置为 SD 模式。

PCI(e)-8154/8158, PCI-8102/PCI-C154(+) 轴参数表

PCI(e)-8154/8158, PCI-8102/PCI-C154(+) 轴参数表				
编号	定义	描述	值	默认值
00h (0)	PRA_EL_LOGIC	PEL/MEL输入逻辑	0 : 不逆 1 : 逆	0
01h (1)	PRA_ORG_LOGIC	ORG输入逻辑	0:低电平有效 1:高电平有效	0
02h (2)	PRA_EL_MODE	当结束限制(包括软极限)为ON时的停止模式。 减速曲线根据PRA_DEC给出。	0:减速停止 0:立即停止	0
03h (3)	PRA_MDN_CONDI	运动完成条件 (对运动统计信息NSTP位有效)	0 : 控制命令完成(默认值) 1 : 使用INP完成命令	0

04h (4)	PRA_ALM_LOGIC	设置ALM逻辑	0:低电平有效 1:高电平有效	0
05h(5)	预留			
06h(6)	PRA_EZ_LOGIC	设置EZ逻辑	0: 下降沿 1: 上升沿	0
07h(7)	预留			
08h	PRA_SPEL_EN	设置编码器事件模式。 (*3,)	0: 禁用 1: 预留 2: Soft-Limit (SPEL) 注意：模式1是已预留。如果设置，则返回错误。	0
09h(9)	PRA_SMEL_EN	设置编码器事件模式。 (*3,)	0: 禁用 1: 预留 2: Soft-Limit (SMEL) 注意：模式1是已预留。如果设置，则返回错误。	0
0Ah(10)	PRA_EFB_POS0	SPEL /	单位：脉冲。(I32 值)	100,000
		EFB 位置0 (*3,)	(28位 , 带符号)	
0Bh(11)	PRA_EFB_POS1	SMEL / EFB 位置1 (*3,)	单位：脉冲。(I32 值) (28位 , 带符号)	- 100,000
0Ch(12)	预留			
0Dh(13)	预留			
0Eh(14)	预留			
0Fh(15)	预留			
10h (16)	PRA_HOME_MODE	归零模式设定	原点搜索- 0(第一个模式)至12(13个模式)归零运动- 20(第一个模式)至32(第13个模式) 注意：原点搜索(6至8)已预留。如果设置，则返回错误。	0
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0
12h (18)	预留			
13h (19)	预留			
14h (20)	预留			
15h (21)	PRA_HOME_VM	归零的最大速度	单位: 脉冲/秒	10000
16h (22)	预留			
17h (23)	预留			
18h (24)	PRA_HOME_EZA	指定EZ计数值	0000(第1个计数)至 1111 (第16个计数)	0

19h (25)	PRA_HOME_VO	归零运动离开原点的速度 – 指定FA速度	单位: 脉冲/秒	100
1Ah	PRA_HOME_OFFSET	归零运动离开原点的距离–指定ORG偏移	单位: 脉冲	100
1Bh~1Fh	预留			
20h (32)	PRA_CURVE	加/减速度模式	0: T-曲线 1: S-曲线	0
21h (33)	PRA_ACC	加速度	单位: 脉冲/秒 ²	100000 0
22h (34)	PRA_DEC	减速度	单位: 脉冲/秒 ²	100000 0
23h (35)	PRA_VS	起始速度	单位: 脉冲/秒	100
24h~27h	预留			
28h	预留			
29h	预留			
2Ah ~ 50h	预留			
53h(83)	PRA_SERVO_LOGIC	SERVO 输出逻辑	0:低电平有效 1:高电平有效	0
80h(128)	PRA_PLS_IPT_MODE	脉冲输入模式	0: A/B X1 1: A/B X2 2: A/B X4 3: CW/CCW	0
81h(129)	PRA_PLS_OPT_MODE	脉冲输出模式	0: OUT/DIR (AL,H+) 1:OUT/DIR (AH,H+) 2: OUT/DIR (AL,L+) 3: OUT/DIR (AH,L+) 4: CW/CCW (AH) 5: CW/CCW (AL) 6: AB (领先) 7: AB (滞后)	0
84h(132)	PRA_EGEAR	电子齿轮系数 = 电机编码器分辨率 (112h) /值 (*1,)	值= 1 ~ 电机编码器分辨率.	40,000
112h(274)	PRA_M_ENC_RES	电机编码器分辨率 (*1,)	单位: 脉冲/转或脉冲/毫米	40,000
160h(352)	PRA_PSR_IPT_MODE	通过PA和PB引脚设置手动脉冲输入模式	ipt_mode = 0 , 1X AB相位类型脉冲输入。 ipt_mode = 1 , 2X AB相位类型脉冲输入。 ipt_mode = 2 , 4X	0

			AB相位类型脉冲输入。 ipt_mode = 3 , CW/CCW类型脉冲输入。	
161h(353)	Reserved			
162h(354)	PRA_PSR_IPT_DIR	从脉冲方向相反的方向运动	Inverse =0, 不反转 Inverse =1, 反方向运动	0
163h(355)	PRA_PSR_RATIO_VALU E	预留		0
164h(356)	PRA_PSR_PDV	将实际输出脉冲率设置为手动脉冲率。脉冲输出率的公式为： (1) 当PDV = 1~2047时 , PMG = 0~31 输出脉冲计数=输入脉冲计数x (PMG+1) x PDV/2048 (2) 当PDV = 0时 , PMG = 0~31 输出脉冲计数=输入脉冲计数x (PMG+1)	PDV = 0~2047	0
165h(357)	PRA_PSR_PMG	参考PRA_PSR_PDV的描述	PMG = 0~31	0
166h(358)	PRA_PSR_HOME_TYPE	指定归零运动类型	HomeType = 0 , 命令原点。 (这意味着当指定计数器变为 “ 0 ” 时轴停止) HomeType = 1 , 反馈原点。 (这意味着当反馈计数器变为 “ 0 ” 时轴停止)	0
167h(359)	PRA_PSR_HOME_SPD	脉冲归零运动的最大速度。	单位: 脉冲/秒	1000
168h~169 h	预留			
200h(512)	PRA_PLS_IPT_LOGIC	脉冲输入逻辑	0 : 不反向计数 方向 1 : 反向计数方向	0
201h(513)	PRA_FEEDBACK_SRC	选择反馈源	0: 扩展编码器模式 扩展编码器计数器和绝对模式参考编码器计数器。	0

			1 : 步进模式 扩展指定计数器 & 绝对模式参考指定计数器。 2 : ACServo模式 扩展编码器计数器和绝对模式参考指定计数器。	
	预留			
210h(528)	PRA_ALM_MODE	ALM模式设定	0 : 立即停止 1 : 减速然后停止	0
211h(529)	PRA_INP_LOGIC	INP输入逻辑	0:低电平有效 1:高电平有效	0
212h(530)	PRA_SD_EN	启用 SD. (*2)	0: 禁用 1: 启用	0
213h(531)	PRA_SD_MODE	SD模式设定	0 : 仅减速 1 : 减速并停止	0
214h(532)	PRA_SD_LOGIC	SD输入逻辑	0:低电平有效 1:高电平有效	0
215h(533)	PRA_SD_LATCH	锁存SD输入	0 : 禁用锁存功能 1 : 启用锁存功能	0
216h(534)	PRA_ERC_MODE	ERC模式设定	0: 禁用 1 : 当被EL , ALM或EMG输入停止时 , 输出ERC 2 : 完成归零后 , 输出ERC 3 : 1和2	3
217h(535)	PRA_ERC_LOGIC	ERC输出逻辑	0:低电平有效 1:高电平有效	0
218h(536)	PRA_ERC_LEN	ERC设定的脉冲宽度	0: 12 us 1: 102 us 2: 409 us 3: 1.6 ms 4: 13 ms	3
			5: 52 ms 6: 104 ms 7: 电平输出	
219h(537)	PRA_RESET_COUNTER	完成归零运动后 , 将计数器的值重置为零。	按位设置： Bit0: 重置计数器1 (命令位置) 0: 禁用 1: 启用 Bit1: 重置计数器2 (机械位置)	15

			0: 禁用 1: 启用 Bit2: 重置计数器3 (偏转位置) 0: 禁用 1: 启用	
21Ah(538)	预留			
21Bh(539)	PRA_PLS_IPT_FLT	启用EA/EB过滤器. 当过滤器启用时, 脉冲输入少于3个CLK信号周期被忽略。	0: 禁用 1: 启用	1
21Ch(540)	预留			
21Dh(541)	PRA_LTC_LOGIC	LTC输入逻辑 用于单锁存函数使用	0: 下降沿 1: 上升沿	0
21Eh(542)	PRA_IO_FILTER	为PEL ,MEL ,SD ,ORG , ALM , INP输入申请过滤器。 (*6)	0 : 不申请过滤器 1 : 申请过滤器	1
21F~220	预留			
221h(545)	PRA_COMPENSATION _脉冲	背隙或滑差校正量	0 至 4095	0
222h(546)	PRA_COMPENSATION _MODE	背隙或滑差模式 设定	0: 禁用 1: 背隙校正 2: 滑差校正	0
223h(547)	PRA_LTC_SRC	2: 滑差校正 用于单锁存函数使用	0: LTC pin 1: ORG pin 2: GCMP ON	0
224h(548)	PRA_LTC_DEST	选择锁存目标	0: 指定计数器 1 : 位置计数器	0
225h(549)	PRA_LTC_DATA	获取锁存数据 (仅读取)	脉冲 (28位, 带符号)	0
226h(550)	PRA_GCMP_EN	通用比较器启用 和设置方法	0: 禁用 其他: 启用 1 : 数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp 计数器 (上计 数) 3 : 数据 = cmp计 数器 (下计数) 4: 数据> CMP计 数器 5: 数据< CMP计数器	0
227h(551)	PRA_GCMP_POS	通用比较器数据	脉冲	0

			(28位 , 带符号)	
228h(552)	PRA_GCMP_SRC	选择通用比较器源	0: 指定计数器 1 : 位置计数器	0
229h(553)	PRA_GCMP_ACTION	符合GCMP时选择操作。	0 : 什么都不做 1 : 立即停止 2 : 减速停止	0
22Ah(554)	PRA_GCMP_STS	检查GCMP是否符合 (仅读取)	0: 不符合 1: 符合	0
22Bh(555)	PRA_VIBSUP_RT	减振- 逆转时间	单位: 1.6 us (16位 , 不带符号)	0
22C(556)	PRA_VIBSUP_FT	减振- 前进时间	单位: 1.6 us (16位 , 不带符号)	0
22Dh(557)	PRA_LATCH_DATA_SP D	为 APS_get_latch_data 2选择错误位置或当前速度的锁存器数据, LatchNum = 2	0 : 锁存错误位置 1 : 锁存当前速度	0
22E~22F	预留			
230h(560) (8154/815 8 Only)	PRA_GPDO_SEL	选择DO/CMP输出 模式	0: DO 1: CMP	0
231(561) (8154/815 8/PCI-C15 4	PRA_GPD1_SEL	选择GPIO输入- DI / LTC / SD / PCS / CLR / EMG. (*2)	0: DI (低电平有效) 1: LTC 2: SD 3: PCS	0
(+) Only)			4: CLR 5: EMG	
231(561) (8102 Only)	PRA_GPD1_SEL	选择GPIO输入- CLR / LTC / SD / PCS.	0: CLR 1: LTC 2: SD 3: PCS	0
232h(562)	PRA_GPD1_LOGIC	选择GPIO输入逻辑	0: 低电平有效 1: 高电平有效	0
233(563)	PRA_RDY_LOGIC	RDY输入逻辑	0: 高电平有效 1: 低电平有效	0
234(564) (Only for PCI-C154(+))	PRA_DI_FILTER_WIDTH	DI 过滤器 宽度 (*4)	0 : 2660 nS 1 : 10340 nS 2 : 33380 nS 3 : 94820 nS 4 : 194660 nS 5 : 993380 nS	5
235(565) (Only for PCI-C154(PRA_DI_FILTER_EN	启用DI 过滤器	0: 禁用 1: 启用	1

+))				
236h~ 23Fh	Reserved			
240h(576)	PRA_SPD_LIMIT	设定固定速度	单位: 脉冲/秒	655350 0
241h(577)	PRA_MAX_ACCDEC	获取最大加速度/减速度 , 该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	245760 00 0
242h(578)	PRA_MIN_ACCDEC	获取最大加速度/减速度 , 该速度受固定速度限制。 (仅读取)	单位: 脉冲/秒 ²	7500
250h(592)	PRA_CONTI_MODE	连续运动模式 (*5)	0:禁用 1:启用	0
251h(593)	PRA_CONTI_BUFF	连续缓冲 (仅读取) (*5)	0:空 1: 满 (8102) 1~3: 缓 冲 (8154/58/PCI-C 154+)	0
270h(624)	PRA_TCMP_EN	触发比较器启用和设置方法	0: 禁用 其他 : 启用	0
			1 : 数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp 计数器 (上计 数) 3 : 数据 = cmp计 数器 (下计数) 4: 数据> CMP计 数器 5: 数据< CMP计数器	
271h(625)	PRA_TCMP_POS	触发比较器位置数据	脉冲 (28位 , 带符号)	0
272h(626)	PRA_TCMP_SRC	选择触发比较器源	0: 指定计数器 1 : 位置计数器	0
273h(627)	PRA_TCMP_STS	检查TCMP是否符合 (仅读取)	0: 不符合 1: 符合	0
274h(628)	PRA_TCMP_LOGIC	设置TCMP信号逻辑	0:负逻辑 1:正逻辑	1
275h(629)	PRA_TCMP_ACTION	当符合TCMP时 , 选 择操作。	0 : 什么都不做 1 : 立即停止 2 : 减速停止	0
280h(640)	PRA_ECMP_EN	错误比较器启用和设	0: 禁用 , 其他启用。	0

		置方法	1 : 比较器位置数据=错误计数器值 (与计数方向无关) , 4 : 比较器位置数据>错误计数器值 5 : 比较器位置数据<错误计数器值	
281h(641)	PRA_ECMP_POS	错误比较器绝对值位置数据.	脉冲 : 0 ~ 32767	0
283h(643)	PRA_ECMP_ACTION	当ECMP符合时 , 选择操作。	0 : 什么都不做 1 : 立即停止 2 : 减速停止	0
284h(644)	PRA_ECMP_STS	检查ECMP是否符合 (仅读取)	0: 不符合 1: 符合	0
			*如果符合 , 请通过轴参数 PRA_ERR_COUNTER重置错误计数器 , 然后将此符合状态自动更改为0。	
290h(656)	PRA_ERR_COUNTER	错误计数器值。	脉冲 : -32768 ~ 32767 (16位)	0
2A0h(672)	PRA_PCS_EN	启用 PCS	0: 禁用 1 : 从打开PCS输入信号开始 , 定位存储在PRMV中的脉冲数	0
2A1h(673)	PRA_PCS_LOGIC	PCS 逻辑	0:低电平有效 1:高电平有效	0

*1: 1: 此参数用于计算运动比率。仅当 PRA_FEEDBACK_SRC 为 0 或 2 时有效。

*2: 当 PRA_GPDI_SEL 设置为 DI/LTC 时 , PRA_SD_EN 自动设置为禁用。在 PRA_SD_EN 设置为启用之前 , 请确保 PRA_GPDI_SEL 设置为 SD 模式。

*3: 当选择正/负软件限位时 , 将指定计数器用作比较计数器。 比较方法如下 :

(EFB 位置 0 < 指定计数器) 为正软件限位(EFB 位置 1 > 指定计数器) 为负软件限位。

*4: 当我们将值设置为 0 时 , DI 过滤器的宽度为 2660 ns (纳秒) 。换句话说 , 截止频率为

$$\frac{1}{10^{-9} \times 2660 \times 2} = 187970 \text{ Hz}$$

*5:连续运动模式用法:

不允许在不同数量的轴之间连续运动。

I32 AxisNo = 0;//轴 ID

I32 Buffer = 0; //使用缓冲区编号

```

I32 ContinuousMoveCount = 0; //连续运动执行编号
ret = APS_set_axis_param(AxisNo, PRA_ACC, 100000); //设置加速度
ret = APS_set_axis_param(AxisNo, PRA_DEC, 100000); //设置减速度
ret = APS_set_axis_param(AxisNo, PRA_VS, 100000); //设置起始速度
ret = APS_set_axis_param(AxisNo, PRA_CONTI_MODE, 1); //启用连续运动
while(ISR==1)
{
    ret = APS_get_axis_param(AxisNo,PRA_CONTI_BUFF,& Buffer);// 获取使用缓冲
区编号
    if(Buffer<3)
    {
        ret = APS_relative_move(AxisNo,ContinuousMoveCount,100000);
        if (ContinuousMoveCount==1000){
            ISR=0;
            ContinuousMoveCount = 0;
            while(Buffer){
                ret = APS_get_axis_param(AxisNo,PRA_CONTI_BUFF,& Buffer);
            }
        }
        else{
            ContinuousMoveCount++;
        }
    }
}

```

注意：不允许在不同数量的轴之间连续运动。

*6: 当应用滤波器时，比以下情况短的信号脉冲将被忽略。

PCIe-8154/58:

- (1) SD= 1.667ms
- (2) EL=ORG= 1.79us
- (3) ALM=INP= 1.43us

PCI-C154+:

- (1) SD= 1.667ms
- (2) EL=ORG= 16us
- (3) ALM=INP= 1.43us

*7: 这些参数可以通过 F64 类型设置/获取参数。

EMX-100 轴参数表

EMX-100 轴参数表					
编号	定义	描述	值	默认值	类型

00h (0)	PRA_EL_LOGIC	PEL/MEL 输入逻辑	0 : 不反转 1 : 反转	0	I32
01h (1)	PRA_ORG_LOGIC	ORG 输入逻辑	0 : 不反转 1 : 反转	0	I32
02h (2)	PRA_EL_MODE	当 EL 打开时，停止模式。	0 : 减速停止 1 : 立即停止(*1)	0	I32
04h (4)	PRA_ALM_LOGIC	设置 ALM 逻辑	0 : 不反转 1 : 反转	0	I32
06h(6)	PRA_EZ_LOGIC	设置 EZ 逻辑	0 : 不反转 1 : 反转	0	I32
0Ah (10)	PRA_SPEL_POS	针对正端的软端限制	单位: 脉冲	100000	I32
0Bh (11)	PRA_SMEL_POS	针对负端的软端限制	单位: 脉冲	- 100000	I32
10h (16)	PRA_HOME_MODE	归零模式设定	0: 归零模式 0 1: 归零模式 1 2: 归零模式 2 16: 归零模式 16	0	I32
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0	I32
13h (19)	PRA_HOME_ACC	归零运动加速度/减速率 (*2)	范围 1 ~ 500,000,000 (单位: 脉冲/秒 ²)	10000	I32
14h (20)	PRA_HOME_VS	归零运动开始速度 (*2)	范围 1 ~ 4,000,000 (单位: 脉冲/秒)	1	I32
15h (21)	PRA_HOME_VM	归零运动最大速度 (*2)	范围 1 ~ 4,000,000 (单位: 脉冲/秒)	1000	I32
18h (24)	PRA_HOME_EZA	启用 EZ 对齐	0: 不启用 1: 启用	0	I32
19h (25)	PRA_HOME_VO	归零运动离开原点的速度 (*2)	范围 1 ~ 4,000,000 (单位: 脉冲/秒)	200	I32
1D (29)	PRA_HOME_EZ_DIR	归零运动 EZ 方向	0: 正方向 1: 反方向	0	I32
1Eh(30)	PRA_HOME_SEARCH_TARGET	归零运动搜索目标 *仅支持归零模式 1 和 2	0: ORG 1: EL	0	I32
20h (32)	PRA_SF	运动加减速速度模式(*2)	[0 ~ 10] 0: T 曲线	0	I32

			10: S 曲线		
21h (33)	PRA_ACC	加速率 (*2)	范围 1~ 500,000,000 (单位: 脉冲/秒 ²)	100000 00	I32
22h (34)	PRA_DEC	减速率 (*2)	范围 1~ 500,000,000 (单位: 脉冲/秒 ²)	100000 00	I32
23h (35)	PRA_VS	开始速度 (*2)	范围 1~ 4,000,000 (单位: 脉冲/秒)	1	I32
41h (65)	PRA_JG_DIR	电动方向	0: 正方向 1: 反方向	0	I32
43h (67)	PRA_JG_ACC	点动加速度 (*2)	范围 1~ 500,000,000 (单位: 脉冲/秒 2)	2000	I32
45h (69)	PRA_JG_VM	点动最大速度 (*2)	范围 1~ 4,000,000 (单位: 脉冲/秒)	1000	I32
4Ch(76)	PRA_JG_STOP	点动停止模式 (*2)	0 : 减速停止 1: 立即停止	1	I32
53h(83)	PRA_SERVO_LOGI C	SERVO 输出逻辑	0: 低电平有效 1: 高电平有效	0	I32
80h (128)	PRA_PLS_IPT_MO DE	脉冲输入模式(编码器类型)	0: AB 脉冲 & 1 边沿评估 1: AB 脉冲 & 2 边沿评估 2: AB 脉冲 & 4 边沿评估 3: 脉冲+ & 脉冲 - (CW & CCW)	0	I32
81h (129)	PRA_PLS_OPT_MO DE	脉冲输入模式	0: 脉冲+ & 脉冲 - (CW & CCW) 1: 脉冲&方向 2: AB 相位 & 4 边沿评估 3: AB 相位 & 2 边沿评估	0	I32
87h(135)	PRA_PLS_IPT_PIN_ DIR	设置编码器的 DIR	0: 不反转 1: 反转	0	I32

88h(136)	PRA_PLS_OPT_PIN _DIR	当 AB 相位和 CW/CCW 时设置 DIR 信号	0: 不反转 1: 反转	0	I32
B0h(176)	PRA_SOFT_EL_EN	软件 EL 启用	0: 不启用 1: 启用	0	I32
B1h(177)	PRA_SOFT_EL_SRC	选择比较编码器或命令位置的软件限制 (EL)	0: 命令位置 1: 编码器	0	I32
B2h(178)	PRA_PLS_IPT_NEG _DRIVE	通过反转编码器 EA 和 EB 信号设置反向驱动	0: 正向驱动 1: 反向驱动	0	I32
B3h(179)	PRA_PLS_OPT_NE G_DRIVE	通过反转脉冲信号设置反向驱动	0: 正向驱动 1: 反向驱动	0	I32
B4h(180)	PRA_PLS_OPT_DIR	设置 DIR 信号以确定 PRA_PLS_OPT_MODE 为脉冲和方向类型时的正向或反向运动	0 : 如果 DIR 低 , 则为正向运动 , 反之亦然 1 : 如果 DIR 高 , 则为正向运动 , 反之亦然	1	I32
B5h(181)	PRA_TRIG_VEL_PR EVENTION_EN	启用三角速度曲线保护	0: 不启用保护 1: 启用保护	0	I32
201h(51 3)	PRA_FEEDBACK_S RC	选择反馈源	0 : 命令位置 1 : 编码器	0	I32
211h(52 9)	PRA_INP_LOGIC	INP 输入逻辑	0 : 不反转 1 : 反转	0	I32
233h(56 3)	PRA_RDY_LOGIC	RDY 输入逻辑	0 : 不反转 1 : 反转	0	I32

(*1) 减速停止规则:

(1-1) S-factor =0, 减速以指定的减速度开始 , 当速度达到 Vmax 时 , 运动停止。

(1-2) S-factor ≠ 0, 有两种情况 , 具体取决于命令减速停止的时间 , 如下所示 :

(I). 在加速过程中 (红色部分 , 如下图所示) : 加速开始 (或继续减小) 到零 , 然后减速以指定的减速度和 S factor 开始 , 当速度达到 VS 时运动停止。 (请注意 , 为了使速度曲线保持平滑 , 速度不会立即降低。)

(II). 达到最大速度后 (③或蓝色部分 , 如图 1 所示) : 减速以指定的 DC 和 S factor 开始 (或继续) , 并且当速度达到 VS 时运动停止。

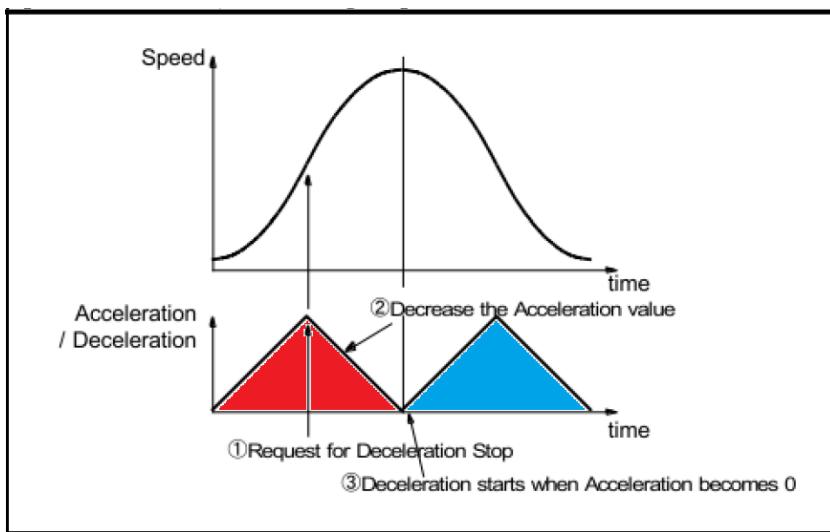


图 1

(*2) 速度曲线标准

S-factor	Dec	VS	DIS(脉冲)
$S = 0$	$Dec > Acc \cdot V_{max} / 8000000$	$VS \geq AC^{1/2}$	$DIS > 2 \cdot (DAC + DDE)$
$10 > S > 0$	用户配置的任何值	$VS \geq AC^{1/2}$	$DIS > (DAC + DDE)$
$S = 10$	用户配置的任何值	$VS \geq 0.1 \cdot AC \cdot (Dec - VS) (-\frac{1}{2})$	$DIS > (DAC + DDE)$

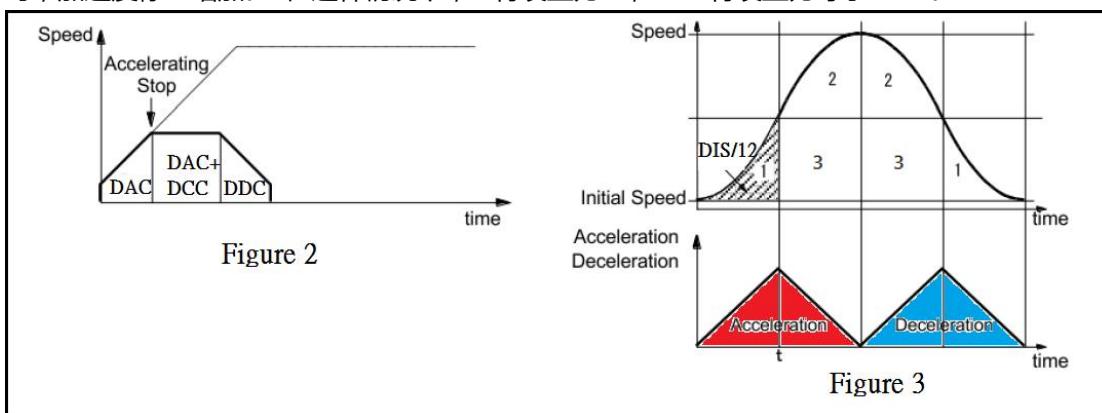
DIS : 运动距离

DAC : 从 VS 到 V_{max} 的加速距离 (图 3 的红色部分)

DDE : 从 V_{max} 到 VS 的减速距离 (图 3 的蓝色部分)

注意 : 当距离不能满足标准时 , 控制器将自动更早开始减速 , 并且速度不会达到 V_{max} 。以下两种情况 :

- (a) 如图 2 所示 , 当 $S = 0$ 时 , 加速提前停止 , 梯形形式使得 $DIS = 2 * (DAC + DDE)$ 。
- (b) 如图 3 所示 , 当 $S > 0$ 时 , 为了保持平滑的速度曲线 , 当输出脉冲数大于总脉冲数的 $1/12$ 时 , 加速度停止增加。在这种情况下 , S 将设置为 1 , DEC 将设置为等于 ACC。



PCI-8254/58 / AMP-204/8C 轴参数表

PCI-8254/58 / AMP-204/8C 轴参数表					
编号	定义	描述	值	默认值	类型
00h (0)	PRA_EL_LOGIC	PEL/MEL 输入逻辑	0:不反转 1:反转	0	I32
01h (1)	PRA_ORG_LOGIC	ORG 输入逻辑	0:不反转 1:反转	0	I32
02h (2)	PRA_EL_MODE	注意：根据 PRA_SD_DEC 给出减速曲线。	0 : 减速停止 1: 立即停止	0	I32
03h (3)	PRA_MDM_COND_I	运动完成的条件 (对运动统计信息 NSTP 位有效)	0: 命令完成 1: 使用 INP 完成命令	0	I32
04h (4)	PRA_ALM_LOGIC	设置 ALM 逻辑	0: 低电平有效 1: 高电平有效	0	I32
06h (6)	PRA_EZ_LOGIC	设置 EZ 逻辑	0: 低电平有效 1: 高电平有效	0	I32
07h (7)	PRA_SD_DEC	停止减速，包括 EL 停止，停止功能和 multi-stop()。	单位: 脉冲/秒 2	100000 000.0	F64
08h (8)	PRA_SPEL_EN	软 PEL 启用	0: 不启用 1: 预留 2: 软限制 (SPEL)	0	I32
09h (9)	PRA_SMEL_EN	软 MEL 启用	0: 不启用 1: 预留 2: 软限制 (SPEL)	0	I32
0Ah (10)	PRA_SPEL_POS	针对正端的软端限制 [F64]	单位: 脉冲	100000. 0	F64
0Bh (11)	PRA_SMEL_POS	针对负端的软端限制 [F64]	单位: 脉冲	- 100000. 0	F64
10h (16)	PRA_HOME_MODE	归零模式设定	0: 归零模式 1 (ORG) 1: 归零模式 2 (EL) 2: 归零模式 3 (EZ)	0	I32
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0	I32

12h (18)	PRA_HOME_CURVE	归零运动加速/减速速度模式	[0.0 ~ 1.0] 0: T 曲线 1: S 曲线	0.5	F64
13h (19)	PRA_HOME_ACC	归零运动加速度率/减速率	单位: 脉冲/秒 2	10000.0	F64
14h (20)	PRA_HOME_VS	归零起始速度	单位: 脉冲/秒	0.0	F64
15h (21)	PRA_HOME_VM	归零最大速度	单位: 脉冲/秒	1000.0	F64
16h (22)	Reserved				
17h (23)	PRA_HOME_SHIFT	来自 EZ , EL , ORG 信号的距离偏移。	单位: 脉冲	0.0	F64
18h (24)	PRA_HOME_EZA	启用 EZ 对齐	0: 不启用 1: 启用	0	I32
19h (25)	PRA_HOME_VO	归零运动离开原点的速度	单位: 脉冲/秒	0.0	F64
1A (26)	Reserved				
1B (27)	PRA_HOME_POS	归零后用户定义的位置。	单位: 脉冲	0.0	F64
20h (32)	PRA_SF	运动加速/减速速度模式	[0.0 ~ 1.0] 0: T 曲线 1: S 曲线	0.0	F64
21h (33)	PRA_ACC	加速度率	单位: 脉冲/秒 2	100000 00.0	F64
22h (34)	PRA_DEC	减速率	单位: 脉冲/秒 2	100000 00.0	F64
23h (35)	PRA_VS	开始速度	单位: 脉冲/秒	0.0	F64
24h (36)	PRA_VM	最大速度	单位: 脉冲/秒	1000.0	F64
25h (37)	PRA_VE	终止速度	单位: 脉冲/秒	0.0	F64
2Ah (42)	PRA_PRE_EVENT_DIST	事件发生前的距离	单位: 脉冲	0.0	F64
2Bh (43)	PRA_POST_EVENT_DIST	事件发生后的距离	单位: 脉冲	0.0	F64
32h(50)	PRA_PT_STP_DO_EN	点表停止/暂停时启用 “执行”	0: 不启用 1: 启用	0	I32
33h(51)	PRA_PT_STP_DO	设置点表停止时的 Do 值	0: 设置为 0 1: 设置为 1	0	I32
34h(52)	PRA_PWM_OFF	ASTP 输入信号有效时，禁用指定的 PWM 输出。	0 : 不启用。 ASTP 处于活动状态时不执行任何操作。	0	I32

			1 : 当 ASTP 有效时 , PWM_CH0 输出将被禁用。 2 : 当 ASTP 有效时 , PWM_CH1 输出将被禁用。		
35h(53)	PRA_DO_OFF	在 ASTP 输入信号有效时设置 Do 值。	0 : 不启用。 当 ASTP 处于活动 状态时不执行任何 操作。 Bit0~3 : 选择 DO 通道。 Bi8 : 设置 ASTP 激活时的 Do 输出 值。 (* 1)	0	I32
40h (64)	PRA_JG_MODE	点动模式	0 : 连续模式 1 : 步进模式	0	I32
41h (65)	PRA_JG_DIR	点动方向	0: 正方向 1: 反方向	0	I32
42h (66)	PRA_JG_SF	点动加速度/减速度模式	0 ~ 1	0.0	F64
43h (67)	PRA_JG_ACC	点动加速度	值 > 0	2000.0	F64
44h (68)	PRA_JG_DEC	点动减速度	值 > 0	2000.0	F64
45h (69)	PRA_JG_VM	点动最大速度	值 > 0	1000.0	F64
46h (70)	PRA_JG_OFFSET	点动偏移 , 步进模式	值 >= 0	1000.0	F64
47h (71)	PRA_JG_DELAY	点动延迟 , 步进模式 , 微秒	0 ~ 10,000,000 微秒	500000	I32
48h (72)	PRA_JG_MAP_DI_E N	(启用数字输入映射至点动命令 信号	1: 启用 0: 不启用 Bit 0: 对 PRA_JG_P_JOGL DI 有效 Bit 1: 对 PRA_JG_N_JOGL DI 有效 Bit 2: 对 PRA_JG_JOGL_DI 有效	0	I32
49h (73)	PRA_JG_P_JOGL_DI	(I32) 正向点动和数字输入的映 射配置。	DI 通道 0 ~ 23	0	I32

4Ah (74)	PRA_JG_N_JOG_DI	(I32) 负向点动和数字输入的映射配置。	DI 通道 0 ~ 23	1	I32
4Bh (75)	PRA_JG_JOG_DI	(I32) 点动和数字输入的映射配置。	DI 通道 0 ~ 23	2	I32
51h (81)	PRA_SINP_WDW	Soft-INP 窗口 , 单位 (脉冲计数)	0: 不启用 1~2147483647: 启用 INP 窗口	0	I32
52h (82)	PRA_SINP_STBT	Soft-INP 稳定时间 , 单位 (毫秒)	[0~10000] ms	0	I32
60h (96)	PRA_GEAR_MASTER	选择齿轮主站 : [0h~7h , 20h ~27h] 0x00~0x07 : 轴 0~7 命令位置偏差 0x20~0x27 : 轴 0~7 反馈位置偏差 注意 : 不允许在同一轴上设置主站和从站。 例如 , APS_set_axis_param (0 , 0x60 , 0); 1、主轴和从轴之间的关系不能闭环。例如 , APS_set_axis_param (0 , 0x60 , 1); APS_set_axis_param (1 , 0x60 , 0);	0x00~0x07: 轴 0 ~7 命令位置偏差 0x20~0x27: 轴 0~7 反馈位置偏差	0x00	I32
61h (97)	PRA_GEAR_ENGAGE_RATE	齿轮啮合率单位 = 1 /秒 , [> = 0.0] (*) 更改循环时间时 , 必须重置此参数。 (循环时间 = 1ms)		1000.0	F64
62h (98)	PRA_GEAR_RATIO	齿轮比	[-10000.0 ~ 10000.0]	1.0	F64
63h (99)	PRA_GANTRY_PROTECT_1	电子齿轮龙门模式保护等级 1 [> = 0.0] fbk_master - fbk_slave > = 值 , SD 停止运动	0.0: 不启用龙门错误检查。 > 0.0 : 启用龙门错误检查。	0.0	F64

64h (100)	PRA_GANTRY_PROTECT_2	电子齿轮龙门模式保护等级 2 [> = 0.0] fbk_master – fbk_slave > = 值，均伺服关闭	0.0 : 不启用机架错误检查。 > 0.0 : 启用龙门错误检查。	0.0	F64
65h(101)	PRA_EGEAR_MASTER	选择齿轮主轴	Axismid : 指定从 0 到 65535 的现有轴 ID 虚拟 axisid : 0x7fffffff	0x7fffffff f	I32
66h(102)	PRA_EGEAR_SOURCE	选择齿轮传动主信号源	0 : 命令位置偏差 1 : 反馈位置偏差	0	I32
80h (128)	PRA_PLS_IPT_MODE	脉冲输入方式	0 : DEC_OUT_DIR_MODE0 1 : DEC_CW_CCW_MODE0 2 : DEC_1XAB 3 : DEC_2XAB 4 : DEC_4XAB 5 : DEC_OUT_DIR_MODE1 6 : DEC_OUT_DIR_MODE2 7 : DEC_OUT_DIR_MODE3 8 : DEC_CW_CCW_MODE1	0	I32
81h (129)	PRA_PLS_OPT_MODE	脉冲输入模式	0x00: OUT/DIR 0x01: CW/CCW 0x02: 4xA/B 相位	0	I32
83h (131)	PRA_ENCODER_FILTER	编码器滤波器		0	I32
85h (133)	PRA_ENCODER_DIR	编码器方向		0	I32
86h (134)	PRA_POS_UNIT_ACTOR			1	F64
88h (136)	PRA_MOVE_RATIO	运动率		1.0	F64
90h (144)	PRA_KP_GAIN(*2)	PID 控制器 Kp 增益		0	I32

91h (145)	PRA_KI_GAIN(*2)	PID 控制器 Ki 增益		0	I32
92h (146)	PRA_KD_GAIN(*2)	PID 控制器 Kd 增益		0	I32
93h (147)	PRA_KVFF_GAIN(*2)	速度前馈增益		0	I32
9Ah (154)	PRA_KAFF_GAIN(*2)	加速度前馈增益		0	I32
9Bh(155)	PRA_KP_SHIFT(*2)	比例控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31~-31]	-5	I32
9Ch(156)	PRA_KI_SHIFT(*2)	整体控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31~-31]	-15	I32
9Dh(157)	PRA_KD_SHIFT(*2)	导数控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31 ~ -31]	0	I32
9Eh(158)	PRA_KVFF_SHIFT(*2)	速度前馈控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31 ~ -31]	0	I32
9Fh(159)	PRA_KAFF_SHIFT(*2)	加速前馈控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31 ~ -31]	0	I32
A0h(160)	PRA_PID_SHIFT(*2)	PID 控制结果偏移	> 0 : 左移 ; < 0 : 右移 ; = 0 : 无位移 ; [31 ~ -31]	-5	I32
120h (288)	PRA_SERVO_V_BIAS (*2)	伺服电压偏移		0	F64
123h (291)	PRA_SERVO_V_LIMIT (*2)	电压饱和输出极限		10.0	F64

124h (292)	PRA_ERR_POS_LE VEL	错误计数器检查级别		90000	F64
125h (293)	PRA_SERVO_V_IN VERSE(*2)	控制电压反转	1: 反转. 0: 不反转	0	
129h(29 7)	PRA_BKL_DIST	背隙补偿值		0.0	F64
12Ah(29 8)	PRA_BKL_CNSP	每个周期的背隙补偿增量值		0.0	F64
12Bh (299)	PRA_INTEGRAL_LI MIT (*2)	积分极限		214748 3647	I32
132h (306)	PRA_BIQUAD0_A1 (*2)	双二阶滤波器 0 系数 A1		0	F64
133h (307)	PRA_BIQUAD0_A2 (*2)	双二阶滤波器 0 系数 A2		0	F64
134h (308)	PRA_BIQUAD0_B0 (*2)	双二阶滤波器 0 系数 B0		1	F64
135h (309)	PRA_BIQUAD0_B1 (*2)	双二阶滤波器 0 系数 B1		0	F64
136h (310)	PRA_BIQUAD0_B2 (*2)	双二阶滤波器 0 系数 B2		0	F64
137h (311)	PRA_BIQUAD0_DI V(*2)	双二阶滤波器系数 DIVIDER		1	F64
138h (312)	PRA_BIQUAD1_A1 (*2)	双二阶滤波器 1 系数 A1		0	F64
139h (313)	PRA_BIQUAD1_A2 (*2)	双二阶滤波器 1 系数 A2		0	F64
13Ah (314)	PRA_BIQUAD1_B0 (*2)	双二阶滤波器 1 系数 B0		1	F64
13Bh (315)	PRA_BIQUAD1_B1 (*2)	双二阶滤波器 1 系数 B1		0	F64
13Ch (316)	PRA_BIQUAD1_B2 (*2)	双二阶滤波器 1 系数 B2		0	F64
13Dh (317)	PRA_BIQUAD1_DI V(*2)	双二阶滤波器 1 系数 DIVDER		1	F64
160h(35 2)	PRA_PSR_IPT_MO DE	脉冲输入模式	0: 1xAB; 1: 2xAB;	0	I32

			2: 4xAB 3: CW/CCW 4: OUT/DIR 模式;		
161h(35 3)	PRA_PSR_IPT_LOG IC	脉冲 EA 和 EB 逻辑	0: EAinv = 0, EBinv = 0 1: EAinv = 0, EBinv = 1 2: EAinv = 1, EBinv = 0 3: EAinv = 1, EBinv = 1	0	I32
162h(35 4)	PRA_PSR_IPT_DIR	脉冲输入方向	0: 不反转; 1: 反转	0	I32
163h(35 5)	PRA_PSR_RATIO_ VALUE	脉冲率	非零值	1	F64
168h(36 0)	PRA_PSR_ACC	脉冲加速	单位: 脉冲/秒 2	100000 0	F64
169h(36 1)	PRA_PSR_JERK	脉冲 jerk	单位: 脉冲/秒 3	100000 0000	F64
211h (529)	PRA_INP_LOGIC	设置 inp 逻辑	0: 低电平有效 1: 高电平有效	0	I32

*1: 参数值详细说明

7	6	5	4	3	2	1	Bit : 0
-	-	-	-	1~8 :DO_CH0~ DO_CH7			
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	ON/OFF

PCIe-833x 轴参数表

PCIe-833x 轴参数表					
编号	Define	描述	Value	Default	Type
02h (2)	PRA_EL_MODE	当 EL 打开时，停止模式。 注意：根据 PRA_SD_DEC 给出减速曲线。	0: 减速停止 1: 立即停止	0	I32
03h (3)	PRA_MDM_COND I	运动完成的条件 (对运动统计信息 NSTP 位有效)	0: 命令完成 1: 使用 INP 完成命令	0	I32
07h (7)	PRA_SD_DEC	停止减速，包括 EL 停止，停止功能和 multi-stop ()。	单位: 脉冲/秒 2 1000000 00.0	1000000 00.0	F64
08h (8)	PRA_SPEL_EN	软 PEL 启用	0: 不启用 1: 预留 2: 软限制 (SPEL)	0	I32
09h (9)	PRA_SMEL_EN	软 MEL 启用	0: 不启用 1: 预留 2: 软限制 (SPEL)	0	I32
0Ah (10)	PRA_SPEL_POS	针对正端的软端限制 [F64]	单位: 脉冲	100000.0	F64
0Bh (11)	PRA_SMEL_POS	针对负端的软端限制 [F64]	单位: 脉冲	- 100000.0	F64
10h (16)	PRA_HOME_MODE	归零模式设定 *如果用户使用 EtherCAT 归零模式，请确保配置其他四个参数 PRA_HOME_ACC , PRA_HOME_VM , PRA_HOME_VO , PRA_HOME_SHIFT。除上述四个参数外，其他归零参数与 EtherCAT 伺服归零运动无关。	0: 归零模式 0 (ORG) 1: 归零模式 1 (EL) 2: 归零模式 2 (EZ) 3: 归零模式 3 (Torque) 4: 归零模式 4 (ORG, 立即停止) 5: 归零模式 5 (ORG+EZ, 立即停止) 6: 归零模式 6 (EL+EZ, 立即停止)	0	I32

			----- ----- 这是从 872 到 1127 的 EtherCAT 伺服 归零模式 872 : 特定于供 应商 873 : 特定于供 应商 ~ 1000 : 不需要归 位操作 1001 : 伺服归零 模式 1 1002 : 伺服归零 模式 2 ~ 1035 : 伺服归零 模式 35 1036 : 保留 ~ 1127 : 保留 请参考* (1) 以 获得详细映射表 描述。		
11h (17)	PRA_HOME_DIR	归零方向	0: 正方向 1: 反方向	0	I32
12h (18)	PRA_HOME_CURVE	归零运动加速/减速速度模式	[0.0 ~ 1.0] 0: T curve 1: S curve	0.5	F64
13h (19)	PRA_HOME_ACC	归零运动加速率/减速率 注意：如果 IF 用户使用 EtherCAT 归零模式，则值类 型将更改为 U32。	单位: 脉冲/秒 2	10000.0	F64
14h (20)	PRA_HOME_VS	归零起始速度	单位: 脉冲/秒	0.0	F64

15h (21)	PRA_HOME_VM	归零最大速度 注意：如果 IF 用户使用 EtherCAT 归零模式，则该值将映射到 CiA402 归零速度 (0x6099) 子索引 1，并且数据类型将更改为 U32。	单位: 脉冲/秒	1000.0	F64
16h (22)	Reserved				
17h (23)	PRA_HOME_SHIFT	来自 EZ , EL , ORG 信号的距离偏移。注意：如果用户使用 EtherCAT 家庭模式，则此参数的用法与家庭模式 0 到 3 不同。 请参考* (2) 说明。此外，值类型将更改为 U32，请输入 U32 类型值。	单位: 脉冲	0.0	F64
18h (24)	PRA_HOME_EZA	在不同的归零模式下有不同的含义。 模式 0~3 : 启用 EZ 拆分 模式 4~6 : EZ 脉冲计数器	模式 0~3 0: 不启用 1: 启用 模式 4~6 : 计数器 0 ~ 15	0	I32
19h (25)	PRA_HOME_VO	归零运动离开原点的速度 注意：如果 IF 用户使用 EtherCAT Home 模式，则该值将映射到 CiA402 归零速度 (0x6099) 子索引 2，并且数据类型将更改为 U32。	单位: 脉冲/秒	200	F64
1Ah(26)	PRA_HOME_OFFSET	归零离开 ORG 的一次距离	单位: 脉冲	0	F64
1Bh(27)	PRA_HOME_POS	归零后用户定义的位置。	单位: 脉冲	0.0	F64
1Ch(28)	PRA_HOME_TORQUE	归零运动的转矩极限值设置。 关于此参数的定义和单位，请参考 EtherCAT 伺服驱动器用户手册中的 CiA 402 目标字典 (索引 : 0x6077) 。	范围: 0 ~ 32767	10	I32
1Dh (29)	PRA_HOME_EZ_DIRECTION	在归零模式 0 (ORG) 中选择 EZ 的搜索方向	0 : 原始 EZ 搜索方向 (默认) ; 1 : 与 EZ 搜索方向相反	0	I32
20h (32)	PRA_SF	移动加速/减速速度模式	[0.0 ~ 1.0] 0: T 曲线	0.0	F64

			1: S 曲线		
21h (33)	PRA_ACC	加速率	单位: 脉冲/秒 2 0.0	1000000 0.0	F64
22h (34)	PRA_DEC	减速率	单位: 脉冲/秒 2 0.0	1000000 0.0	F64
23h (35)	PRA_VS	开始速度	单位: 脉冲/秒 0.0	0.0	F64
24h (36)	PRA_VM	最大速度	单位: 脉冲/秒 1000.0	1000.0	F64
25h (37)	PRA_VE	终止速度	单位: 脉冲/秒 0.0	0.0	F64
2Ah (42)	PRA_PRE_EVENT_DIST	事件发生前的距离	单位: 脉冲 0.0	0.0	F64
2Bh (43)	PRA_POST_EVENT_DIST	事件发生后的距离	单位: 脉冲 0.0	0.0	F64
40h (64)	PRA_JG_MODE	点动模式	0 : 连续模式 1 : 步进模式	0	I32
41h (65)	PRA_JG_DIR	点动方向	0: 正方向 1: 反方向	0	I32
42h (66)	PRA_JG_SF	点动加速度 / deceleration speed pattern	0 ~ 1	0.0	F64
43h (67)	PRA_JG_ACC	点动加速度	值 > 0	2000.0	F64
44h (68)	PRA_JG_DEC	点动减速速度	值 > 0	2000.0	F64
45h (69)	PRA_JG_VM	点动最大速度	值 > 0	1000.0	F64
46h (70)	PRA_JG_OFFSET	点动偏移 , 步进模式	值 >= 0	1000.0	F64
47h (71)	PRA_JG_DELAY	点动加速度/减速速度模式	0 ~ 10,000,000 微秒	500000	I32
48h (72)	PRA_JG_MAP_DI_EN	(启用数字输入映射至点动命令信号	1: 启用 0: 不启用 Bit 0: 对 PRA_JG_P_JOGLDI 有效 Bit 1: 对 PRA_JG_N_JOGLDI 有效 Bit 2: 对 PRA_JG_JOGLDI 有效	0	I32
49h (73)	PRA_JG_P_JOGLDI	(I32) 正向点动和数字输入的映射配置。	DI 通道 0 ~ 3	0	I32
4Ah (74)	PRA_JG_N_JOGLDI	(I32) 负向点动和数字输入的映射配置。	DI 通道 0 ~ 3	1	I32

4Bh (75)	PRA_JG_JOG_DI	(I32) 点动和数字输入的映射配置。	DI 通道 0 ~ 3	2	I32
51h (81)	PRA_SINP_WDW	Soft-INP 窗口 , 单位 (脉冲计数)	0: 不启用 1~2147483647 : 启用 INP 窗口	0	I32
52h (82)	PRA_SINP_STBL	Soft-INP 稳定时间 , 单位 (毫秒)	[0~10000] ms	0	I32
61h (97)	PRA_GEAR_ENGA GE_RATE	齿轮啮合率单位 = 1 / 秒 , [> = 0.0] (*) 更改循环时间时 , 必须重置此参数。 (循环时间 = 1ms)		1000.0	F64
62h (98)	PRA_GEAR_RATIO	齿轮比	[-10000.0 ~ 10000.0]	1.0	F64
63h (99)	PRA_GANTRY_PR OTECT_1	电子齿轮龙门模式保护等级 1 [> = 0.0] fbk_master - fbk_slave > = 值 , SD 停止运动	0.0: 不启用龙门错误检查。 > 0.0 : 启用龙门错误检查。	0.0	F64
64h (100)	PRA_GANTRY_PR OTECT_2	电子齿轮龙门模式保护等级 2 [> = 0.0] fbk_master - fbk_slave > = 值 , 均伺服关闭	0.0 : 不启用机架错误检查。 > 0.0 : 启用龙门错误检查。	0.0	F64
65h(101)	PRA_EGEAR_MAS TER	选择齿轮主轴	Axismid : 指定从 0 到 65535 的现有轴 ID 虚拟 axisid : 0x7fffffff	0x7fffffff	I32
66h(102)	PRA_EGEAR_SOUC RCE	选择齿轮传动主信号源	0 : 命令位置偏差 1 : 反馈位置偏差	0	I32
86h (134)	PRA_POS_UNIT_F ACTOR	1) 用户期望命令乘以 PRA_POS_UNIT_FACTOR 等于实际命令脉冲输出。 2) 实际编码器脉冲除以 PRA_POS_UNIT_FACTOR 等于反馈脉冲。		1	F64
88h (136)	PRA_MOVE_RATI O	运动率		1.0	F64
124h (292)	PRA_ERR_C_LEVEL	错误计数器检查级别		0	F64
129h(297)	PRA_BKL_DIST	背隙补偿的总距离		0.0	F64

12Ah(298)	PRA_BKL_CNSP	该值将补偿每个循环的背隙，直到等于总距离		0.0	F64
144h(324)	PRA_CMD_PSF	命令重塑路径平滑因子	20~1000 : 截止频率 (rad / 秒) 0: 不启用	0	F64
160h(352)	PRA_PSR_IPT_MO DE	脉冲输入模式	0: 1xAB; 1: 2xAB; 2: 4xAB 3: CW/CCW	0	I32
161h(353)	PRA_PSR_IPT_LOG IC	脉冲 EA 和 EB 逻辑	0: EAinv = 0, EBinv = 0 1: EAinv = 0, EBinv = 1 2: EAinv = 1, EBinv = 0 3: EAinv = 1, EBinv = 1	0	I32
162h(354)	PRA_PSR_IPT_DIR	脉冲输入方向	0: 不反转; 1: 反转	0	I32
163h(355)	PRA_PSR_RATIO_ VALUE	脉冲率	非零值	1	F64
168h(360)	PRA_PSR_ACC	脉冲加速	单位: 脉冲/秒 2	1000000	F64
169h(361)	PRA_PSR_JERK	脉冲 jerk	单位: 脉冲/秒 3	1000000 000	F64

*(1) APS & CiA402 归零模式映射表：

DATA DESCRIPTION		CiA 402 Object 6098 h : Homing method
APS mode value	CiA402 Value	Description
872 ... 999	-128 .. -1	manufacturer specific
1000	0	No homing operation required
1001 .. 1035	1..35	Methods 1 to 35 (see the functional description)
1036 .. 1127	36 .. 127	reserved

凌华科技提供了使用 EtherCAT CiA402 标准 35 型伺服主站的用户界面，但可能会发生某些情况。

- CiA 402 定义了 35 种归零模式，但供应商的伺服驱动器仅支持少于 35 种类型。
这可能会使用户无法访问所有用户。
- CiA 402 定义的某些归零模式行为与供应商的行为不同。因为供应商可能未遵循所有 CiA402 归零行为。

换句话说，在使用 EtherCAT 伺服归零模式之前，用户应参考供应商的规格指南。

*(2) EtherCAT 伺服归零模式的归零偏移：

如果用户使用 EtherCAT 伺服归零模式，则参数 PRA_HOME_SHIFT (0x17) 将映射到 CiA402 对象 0x607C 归零偏移。因此，这不同于归零模式 0~3。此外，相同的归零偏移量 0x607C，这些供应商的定义也可能不同。以安川和松下为例，如下表 (2-1) 所示。将这两个伺服驱动器设置为 EtherCAT 归零模式 (0x6098) 1033 (CiA 归零模式 33 搜索 EZ) 和归零偏移量 (0x607C) 10000。归零启动之前，它们的实际位置 (0x6064) 在 3000 中。实际位置 (0x6064) 不同。YASKAWA 与 CiA402 定义相同，但 PANASONIC 拥有自己的定义，如下图 (2-2) 。

供应商 4	归零偏移	0x6064 (归零前)	0x6064 (归零完成)
YASKAWA	10000	3000	13000
PANASONIC	10000	3000	10000

(2-1)

• 原点位置检出后，此位置作为基准初始化下述的对象(预置)。

PANASONIC

6062h(Position demand value) = 6064h(Position actual value) = 607Ch(Home offset)
6063h(Position actual internal value) = 60FCh(Position demand internal value) = 0

(2-2)

C. 采样参数表

PCI-8392(H) 和 PCI-8253/56 和 MNET-4XMO 和 PCI-8254/58 / AMP-204/8C 和 PCIe-833x 采样参数表

采样参数表				
参数编号	定义	描述	参数数据含义	默认值
00h	SAMP_PA_RATE	采样率(周期), (取决于循环时间) 针对8392和8253/6	1~ 65535 (循环次数)	1
		采样率(ms), (取决于OS计时器) 针对MNET-4XMO	1~5	1
02h	SAMP_PA_EDGE	边缘触发	0 : 上升沿 , 1 : 下降沿	0
03h	SAMP_PA_LEVEL	触发电平	(I32) -2147483648至 2147483647	0
05h	SAMP_PA_TRIGCH	触发通道	0 ~ 3 (Ch0~Ch3)	0
10h	SAMP_PA_SRC_CH 0	通道0的采样源	请参阅采样源表。 (* 1)	0
11h	SAMP_PA_SRC_CH 1	通道1的采样源	请参阅采样源表。 (* 1)	0
12h	SAMP_PA_SRC_CH 2	通道2的采样源	请参阅采样源表。 (* 1)	0
13h	SAMP_PA_SRC_CH 3	通道3的采样源	请参阅采样源表。 (* 1)	0

*1. 此参数还必须包含轴 ID 的信息。 该参数需要四个字节的数据。 前两个字节是轴 ID 的信息，而后两个字节是采样源的类型。

例如：轴 ID = 150 (96h)，选择源为 SAMP_FBK_POS (01h)。 然后设置的参数值为 0x00960001。

D. 采样源表

PCI-8392(H)的采样源表

PCI-8392(H) 采样源表				
源	符号定义	描述	取值范围	备注
00h	SAMP_COM_POS	命令位置 (脉冲)	I32 值	
01h	SAMP_FBK_POS	反馈位置 (脉冲)	I32 值	
02h	SAMP_CMD_VEL	命令速度 (pps)	I32 值	
03h	SAMP_FBK_VEL	反馈速度 (pps)	I32 值	
04h	SAMP_MIO	运动IO状态 (与获取运动IO函数相同)	I32 值 (位格式)	
05h	SAMP_MSTS	动作状态 (与获取运动状态函数相同)	I32 值 (位格式)	
06h	SAMP_MSTS_ACC	加速时的运动状态 (命令速度)	0: 不加速 1: 加速	
07h	SAMP_MSTS_MV	最大速度下的运动状态 (指令速度)	0: 不是最大速度 1: 最大速度	
08h	SAMP_MSTS_DEC	减速时的动作状态 (命令速度)	0: 不减速 1: 减速	
09h	SAMP_MSTS_CSTP	运动状态命令停止 (CSTP)	0: CSTP状态开 1: CSTP状态关	
0Ah	SAMP_MSTS_NSTP	运动状态正常停止 (NSTP)	0: NSTP状态开 1: NSTP状态关	
0Bh	SAMP_MIO_INP	运动状态位置 (INP)	0: INP状态开 1: INP状态关	
0Ch	SAMP_MIO_ZERO	运动状态为零 (零)	0: ZERO状态开 1: ZERO状态关	
0Dh	SAMP_MIO_ORG	运动状态ORG状态	0: OGR状态开 1: OGR状态关	
10h	SAMP_SSC_MON_0	SSCNET 伺服监控器 0	I32 值	(*1)
11h	SAMP_SSC_MON_1	SSCNET 伺服监控器 1	I32 值	(*1)
12h	SAMP_SSC_MON_2	SSCNET 伺服监控器 2	I32 值	(*1)
13h	SAMP_SSC_MON_3	SSCNET 伺服监控器 3	I32 值	(*1)
20h	预留			
21h	SAMP_GTY_DEVIATION	主从编码器原始数据之间的龙门偏差	I32 值	
22h	预留			

23h	SAMP_ERROR_CO UNT ER	错误计数器数据	I32 值	
-----	-------------------------	---------	-------	--

(*1) 监视数据取决于监视数据源设置。请参考 SSCNET 伺服监控器源表。

PCI-8253/56 采样源表

PCI-8253/56 采样源表				
源	符号定义	描述	取值范围	备注
00h	SAMP_COM_POS	命令位置 (脉冲)	I32 值	
01h	SAMP_FBK_POS	反馈位置 (脉冲)	I32 值	
02h	SAMP_CMD_VEL	命令速度 (pps)	I32 值	
03h	SAMP_FBK_VEL	反馈速度 (pps)	I32 值	
04h	SAMP_MIO	运动IO状态 (与获取运动IO函数相同)	I32 值 (位格式)	
05h	SAMP_MSTS	动作状态 (与获取运动状态函数相同)	I32 值 (位格式)	
06h	SAMP_MSTS_ACC	加速时的运动状态 (命令速度)	0: 不加速 1: 加速	
07h	SAMP_MSTS_MV	最大速度下的运动状态 (命令速度)	0: 不是最大速度 1: 最大速度	
08h	SAMP_MSTS_DEC	减速时的动作状态 (命令速度)	0: 不减速 1: 减速	
09h	SAMP_MSTS_CSTP	运动状态命令停止 (CSTP)	0: CSTP状态开 1: CSTP状态关	
0Ah	SAMP_MSTS_NSTP	运动状态正常停止 (NSTP)	0: NSTP状态开 1: NSTP状态关	
0Bh	SAMP_MIO_INP	运动状态位置 (INP)	0: INP状态开 1: INP状态关	
0Ch	SAMP_MIO_ZERO	运动状态为零 (零)	0: ZERO状态开 1: ZERO状态关	
0Dh	SAMP_MIO_ORG	运动状态ORG状态	0: OGR状态开 1: OGR状态关	
20h	SAMP_CONTROL_VOL	控制电压	I32 值	
21h	SAMP_GTY_DEVIATION	主从编码器原始数据之间的龙门偏差	I32 值	
22h	SAMP_ENCODER_RAW	编码器原始数据	I32 值	

23h	SAMP_ERROR_COUN TER	错误计数器数据	I32 值	
-----	------------------------	---------	-------	--

MNET-4XMO 采样源表

MNET-4XMO 采样源表				
源	符号定义	描述	取值范围	备注
00h	SAMP_COM_POS	命令位置 (脉冲)	I32 值	
01h	SAMP_FBK_POS	反馈位置 (脉冲)	I32 值	
02h	SAMP_CMD_VEL	命令速度 (pps)	I32 值	

PCI-8254/58 / AMP-204/8C sampling source table

PCI-8254/58 / AMP-204/8C 采样源表					
源	符号定义	描述	取值范围	类型	参考 ID
0x00	SAMP_COM_POS	命令位置	I32 值	I32	轴 id
0x01	SAMP_FBK_POS	反馈位置	I32 值	I32	轴 id
0x02	SAMP_CMD_VEL	命令速度	I32 值	I32	轴 id
0x03	SAMP_FBK_VEL	反馈速度	I32 值	I32	轴 id
0x04	SAMP_MIO	运动 IO	I32 值(位格 式)	I32	轴 id
0x05	SAMP_MSTS	运动状态	I32 值(位格 式)	I32	轴 id
0x06	SAMP_MSTS_ACC	加速时的运动状态	0: 不加速 1: 加速	I32	轴 id
0x07	SAMP_MSTS_MV	最大速度下的运动 状态	0: 不是最大 速度 1: 最大速度	I32	轴 id
0x08	SAMP_MSTS_DEC	减速时的动作状态	0: 不减速 1: 减速	I32	轴 id
0x09	SAMP_MSTS_CSTP	运动状态 CSTP	0: CSTP 状态 开 1: CSTP 状态 关	I32	轴 id
0x0A	SAMP_MSTS_MDN	运动状态 MDN	0:NSTP 状态开	I32	轴 id

			1:NSTP 状态 关		
0x0B	SAMP_MIO_INP	运动状态 INP	0: INP 状态 开 1: INP 状态 关	I32	轴 id
0x0D	SAMP_MIO_ORG	运动状态 OGR	0: OGR 状态 开 1: OGR 状态 关	I32	轴 id
0x20	SAMP_CONTROL_VOL	控制命令电压	I32 值	I32	轴 id
0x21	SAMP_GTY_DEVIATION	龙门偏差	I32 值	I32	轴 id
0x22	SAMP_ENCODER_RAW	编码器原始数据	I32 值	I32	轴 id
0x23	SAMP_ERROR_POS	错误位置	I32 值	I32	轴 id
0x24	SAMP_PTBUFF_RUN_INDEX	点表运行索引	I32 值	I32	表 id 0~1
0x10	SAMP_COM_POS_F64	命令位置	F64 值	F64	轴 id
0x11	SAMP_FBK_POS_F64	反馈位置	F64 值	F64	轴 id
0x12	SAMP_CMD_VEL_F64	命令速度	F64 值	F64	轴 id
0x13	SAMP_FBK_VEL_F64	反馈速度	F64 值	F64	轴 id
0x14	SAMP_CONTROL_VOL_F64	控制命令电压	F64 值	F64	轴 id
0x15	SAMP_ERR_POS_F64	错误位置	F64 值	F64	轴 id
0x18	SAMP_PWM_FREQENCY_F64	PWM 频率(Hz)	F64 值	F64	PWM CH id
0x19	SAMP_PWM_DUTY_CYCLE_F64	PWM 占空比(%)	F64 值	F64	PWM CH id
0x1A	SAMP_PWM_WIDTH_F64	PWM 宽度 (ns)	F64 值	F64	PWM CH id
0x1B	SAMP_VAO_COMP_VEL_F64	激光功率控制的合成速度(pps)	F64 值	F64	VAO id

0x1C	SAMP_PTBUFF_CO MP_VEL_F64	点表合成速度	F64 值	F64	表 id 0~1
0x1D	SAMP_PTBUFF_CO MP_ACC_F64	点表的合成加速度	F64 值	F64	表 id 0~1

PCIe-833x 采样源表

PCIe-833x 采样源表					
源	符号定义	描述	取值范围	类型	参考 ID
0x00	SAMP_COM_POS	命令位置	I32 值	I32	轴 id
0x01	SAMP_FBK_POS	反馈位置	I32 值	I32	轴 id
0x02	SAMP_CMD_VEL	命令速度	I32 值	I32	轴 id
0x03	SAMP_FBK_VEL	反馈速度	I32 值	I32	轴 id
0x04	SAMP_MIO	运动 IO	I32 值(位格式)	I32	轴 id
0x05	SAMP_MSTS	运动状态	I32 值(位格式)	I32	轴 id
0x06	SAMP_MSTS_ACC	加速时的运动状态	0: 不加速 1: 加速	I32	轴 id
0x07	SAMP_MSTS_MV	最大速度下的运动状态	0: 不是最大速度 1: 最大速度	I32	轴 id
0x08	SAMP_MSTS_DEC	减速时的动作状态	0: 不减速 1: 减速	I32	轴 id
0x09	SAMP_MSTS_CSTP	运动状态 CSTP	0: CSTP 状态开 1: CSTP 状态关	I32	轴 id
0x0A	SAMP_MSTS_MDN	运动状态 MDN	0:NSTP 状态开 1:NSTP 状态关	I32	轴 id
0x0B	SAMP_MIO_INP	运动状态 INP	0: INP 状态开 1: INP 状态关	I32	轴 id
0x0D	SAMP_MIO_ORG	运动状态 OGR	0: OGR 状态开 1: OGR 状态关	I32	轴 id

0x21	SAMP_GTY_DEVIATION	龙门偏差	I32 值	I32	轴 id
0x22	SAMP_ENCODER_RAW	编码器原始数据	I32 值	I32	轴 id
0x23	SAMP_ERROR_POS	错误位置	I32 值	I32	轴 id
0x24	SAMP_PTBUFF_RUN_INDEX	点表运行索引	I32 值	I32	表 id 0~1
0x10	SAMP_COM_POS_F64	命令位置	F64 值	F64	轴 id
0x11	SAMP_FBK_POS_F64	反馈位置	F64 值	F64	轴 id
0x12	SAMP_CMD_VEL_F64	命令速度	F64 值	F64	轴 id
0x13	SAMP_FBK_VEL_F64	反馈速度	F64 值	F64	轴 id
0x15	SAMP_ERR_POS_F64	错误位置	F64 值	F64	轴 id
0x1C	SAMP_PTBUFF_COMP_VEL_F64	点表合成速度	F64 值	F64	表 id 0~1
0x1D	SAMP_PTBUFF_COMP_ACC_F64	点表的合成加速度	F64 值	F64	表 id 0~1

E. 运动 IO 状态和运动状态定义

PCI-8392(H) 运动 IO 状态表。

PCI-8392(H)运动IO状态表。								
BitNo	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
BitNo	15	14	13	12	11	10	9	8
		ABSL	TLC	SMEL	SPEL	ZSP	WARN	RDY

PCI-8253/56 运动 IO 状态表。

PCI-8253/56运动IO状态表。								
BitNo	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
BitNo	15	14	13	12	11	10	9	8

				SMEI	SPEL	ZSP	WARN	RDY
--	--	--	--	------	------	-----	------	-----

MNET-4XMO-(C)/1XMO, HSL-4XMO, PCI(e)-

8154/8158,PCI-8102/PCI-C154(+)运动 IO 状态表。

MNET-4XMO-(C)/1MXO, HSL-4XMO运动IO状态表。								
BitNo	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
BitNo	15	14	13	12	11	10	9	8
								RDY

PCI-8144 运动 IO 状态表。

PCI-8144运动IO状态表。								
BitNo	7	6	5	4	3	2	1	0
	--	--	--	EMG(STP)	ORG	MEL	PEL	--
BitNo	15	14	13	12	11	10	9	8
	ST A	--	--	--	--	--	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	MSD	PSD

运动 IO 状态描述表

运动IO状态描述表		
Bit	定义	描述
0	ALM	伺服报警
1	PEL	正限位
2	MEL	负限位
3	ORG	原始位置传感器 (原点传感器)
4	EMG	EMG 传感器
5	EZ	EZ 通过
6	INP	就位
7	SVON	伺服开启
8	RDY	准备就绪
9	WARN	告警
10	ZSP	零速，零速输出范围设置，请参考伺服驱动器手册。
11	SPEL	软件正限位
12	SMEI	软件负限位
13	TLC	转矩受转矩极限值限制。 (当转矩控制打开时)
14	ABSL	绝对位置丢失。
15		
16	PSD	正减速信号输入

17	MSD	负减速信号输入
----	-----	---------

EMX-100 运动 IO 状态描述表

EMX-100 运动IO状态描述表								
BitNo	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	--	--	RDY
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	--
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

EMX-100 运动 IO 状态描述表

运动 IO 状态描述表		
Bit	定义	描述
0	ALM	伺服报警
1	PEL	正限位
2	MEL	负限位
3	ORG	原始位置传感器 (原点传感器)
4	EMG	EMG 传感器
5	EZ	EZ 通过
6	INP	就位
7	SVON	伺服开启
8	RDY	准备就绪
25~31	Reserved	预留 , 通常为 0

PCI-8254/58 / AMP-204/8C 运动 IO 状态描述表

PCI-8254/58 运动IO状态描述表								
BitNo	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
BitNo	15	14	13	12	11	10	9	8
				SMEL	SPEL	SCL		

PCI-8254/58 / AMP-204/8C 运动 IO 状态描述表

运动 IO 状态描述表		
Bit	定义	描述
0	ALM	伺服报警

1	PEL	正限位
2	MEL	负限位
3	ORG	原始位置传感器 (原点传感器)
4	EMG	EMG 传感器
5	EZ	EZ 通过
6	INP	就位
7	SVON	伺服开启
8~9	Reserved	预留，通常为 0
10	SCL	软件循环限位
11	SPEL	软件正限位
12	SMEL	软件负限位
13~	Reserved	预留，通常为 0

PCIe-833x 运动 IO 状态描述表

PCIe-833x 运动IO状态描述表									
BitNo	7	6	5	4	3	2	1	0	
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM	
BitNo	15	14	13	12	11	10	9	8	
				SMEL	SPEL	SCL		RDY	
BitNo	23	22	21	20	19	18	17	16	
BitNo	31	30	29	28	27	26	25	24	
									OP

PCIe-833x 运动 IO 状态描述表

运动 IO 状态描述表		
Bit	定义	描述
0	ALM	伺服报警
1	PEL	正限位
2	MEL	负限位
3	ORG	原始位置传感器 (原点传感器)
4	EMG	EMG 传感器
5	EZ	EZ 通过
6	INP	就位
7	SVON	伺服开启
8	RDY	准备就绪
9	Reserved	预留，通常为 0
10	SCL	软件循环限位

11	SPEL	软件正限位
12	SMEL	软件负限位
13~23	Reserved	预留，通常为 0
24	OP	0：并非所有从站都处于 OP 模式。 1：所有从站都处于 OP 模式。
25~31	Reserved	预留，通常为 0

F. 运动状态定义表

PCI-8392(H), 8253/56 运动状态定义表

运动状态定义表								
BitNo	7	6	5	4	3	2	1	0
	SM	HMV	NSTP	DIR	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8
	JO	SLV	PPS	PDW	PMV	VS	CIP	LIP
BitNo	23	22	21	20	19	18	17	16
	EC	MELS	PELS	WANS	ALMS	EMGS	SVON	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	--	PAPB	GTM	GDCES	STPO	SMEL	SPELS

MNET-4XMO-(C), HSL-4XMO, PCI(e)-8154/8158, PCI-8102

/PCI-C154(+)运动状态定义表

运动状态定义表								
BitNo	7	6	5	4	3	2	1	0
	SMV	HMV	NSTP	--	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	VS	CIP	LIP

BitNo	23	22	21	20	19	18	17	16
	--	MELS	PELS	--	ALMS	EMGS	--	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	--	PAPB (仅 PCI-C154+)	--	--	--	SMELS	SPELS

1XMO 运动状态定义表

运动状态定义表								
BitNo	7	6	5	4	3	2	1	0
	SMV	HMV	NSTP	--	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	VS	--	--
BitNo	23	22	21	20	19	18	17	16
	--	MELS	PELS	--	ALMS	EMGS	--	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	SMELS	SPELS

PCI-8144 运动状态定义表

运动状态定义表								
BitNo	7	6	5	4	3	2	1	0
	--	HMV	--	DIR	DEC	ACC	--	CSTP
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	--	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	--
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

运动状态描述表

运动状态描述表		
Bit	定义	描述
0	CSTP	命令已停止
1	VM	处于最大速度
2	ACC:	加速
3	DEC:	减速
4	DIR:	运动方向。 1:正方向, 0:反方向
5	NSTP	正常停止(动作完成)
6	HM	正在归零

7	SMV	单轴运动(相对, 绝对, 速度运动)
8	LIP	线性插补
9	CIP	圆弧插补
10	VS	处于起始速度
11	PMV	点表运动
12	PDW	点表停顿运动
13	PPS	点表暂停状态
14	SLV	从轴运动
15	JOG	点动
16	ASTP	异常停止
17	SVONS	伺服停止
18	EMGS	EMG / SEMG停止
19	ALMS	报警停止
20	WANS	告警停止
21	PELS	PEL 停止
22	MELS	MEL停止
23	ECES	错误计数器检查级别达到并停止
24	SPELS	SPEL停止
25	SMELS	SMEL停止
26	STPOA	被其他轴停止
27	GDCES	龙门偏差误差水平达到并停止
28	GTM	龙门模式
29	PAPB	等待PA/PB输入
30	--	预留

EMX-100 运动状态描述表

运动状态描述表								
BitNo	7	6	5	4	3	2	1	0
	--	--	MDN	--	--	--	--	--
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	--	--	--
BitNo	23	22	21	20	19	18	17	16
	--	MELS	PELS	--	ALMS	EMGS	--	--
BitNo	31	30	29	28	27	26	25	24
	--	ORGS	HMES	EZS	--	--	SMELS	SPELS

EMX-100 运动状态描述表

运动状态描述表		
Bit	定义	描述

...	Reserved	预留 , 通常为 0
5	MDN	运动停止 ; 0 : 运动中 ; 1 : 运动完成 (可能是异常停止)
...	Reserved	预留 , 通常为 0
18	EMGS	EMG 停止
19	ALMS	报警停止
...	Reserved	预留 , 通常为 0
21	PELS	PEL 停止
22	MELS	MEL 停止
...	Reserved	预留 , 通常为 0
24	SPELS	SPEL 停止
25	SMELS	SMEL 停止
...	Reserved	预留 , 通常为 0
28	EZS	EZ 停止
29	HMES	归零错误停止。如果 PRA_HOME_EZA 在搜索 EZ 信号之前以 HOME_VO 速度启用 , 则已经触发 EZ 信号。
30	ORGS	ORG 停止

PCI-8254/58 / AMP-204/8C 运动状态描述表

运动状态描述表								
BitNo	7	6	5	4	3	2	1	0
	--	HMV	MDN	DIR	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8
	JOG	--	--	--	PTB	WAIT	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	POSTD	PRED	BLD	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	BACKLASH	--	GER	--	--	--	--

PCI-8254/58 / AMP-204/8C 运动状态描述表

运动状态描述表		
Bit	Define	描述
0	CSTP	命令停止 (但可能正在运行)
1	VM	处于最大速度
2	ACC	处于加速中
3	DEC	处于减速中
4	DIR	移动方向。 1 : 正方向 , 0 : 负方向
5	MDN	动作完成。 0 : 运动中 ; 1 : 运动完成 (可能是异常停止)
6	HMV	处于归零中

...	Reserved	预留 , 通常为 0
10	WAIT	轴处于等待状态。 (等待运动触发)
11	PTB	轴在点缓冲区中运动。 (此位打开时 , 将清除 MDN 和 ASTP)
...	Reserved	预留 , 通常为 0
15	JOG	处于点动中
16	ASTP	0 : 正常停止 ; 1 : 异常停止 ; 轴运动时 , 该位清零。
17	BLD	混合运动中的轴 (轴)
18	PRED	前距离事件。 1 : 事件到达。 当轴开始运动时 , 该事件将清除
19	POSTD	后距离事件。 1 : 事件到达。 当轴开始移动时 , 该事件将清除
...	Reserved	预留 , 通常为 0
28	GER	1 : 在齿轮传动中 (该轴为从轴 , 并遵循轴参数中指定的主动轴。)
29	Reserved	预留 , 通常为 0
30	BACKLASH	0 : 运行中 ; 1 : 空闲
31	Reserved	预留 , 通常为 0

PCIe-833x 运动状态描述表

运动状态描述表								
BitNo	7	6	5	4	3	2	1	0
	--	HMV	MDN	DIR	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8
	JOG	--	--	--	PTB	WAIT	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	POSTD	PRED	BLD	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	GRY	PSR	GER	--	--	--	--

PCIe-833x 运动状态描述表

运动状态描述表		
Bit	定义	描述
0	CSTP	命令停止 (但可能正在运行)
1	VM	处于最大速度
2	ACC	处于加速中
3	DEC	处于减速中
4	DIR	移动方向。 1：正方向，0：负方向
5	MDN	动作完成。 0：运动中； 1：运动完成（可能是异常停止）
6	HMV	处于归零中
...	Reserved	预留，通常为 0
10	WAIT	轴处于等待状态。（等待运动触发）
11	PTB	轴在点缓冲区中运动。（此位打开时，将清除 MDN 和 ASTP）
...	Reserved	预留，通常为 0
15	JOG	处于点动中
16	ASTP	0：正常停止； 1：异常停止；轴运动时，该位清零。
17	BLD	混合运动中的轴（轴）（仅用于插补运动）
18	PRED	前距离事件。1：事件到达。当轴开始运动时，该事件将清除
19	POSTD	后距离事件。1：事件到达。当轴开始移动时，该事件将清除
...	Reserved	预留，通常为 0
28	GER	1：在齿轮传动中（该轴为从轴，并遵循轴参数中指定的主动轴。）
29~	PSR	脉冲发生器功能状态。0：禁用，1：启用
30	GRY	1：启用龙门模式时，该轴为主轴，其运动状态位 30 (GRY) 将打开。 0：禁用龙门模式时，是否关闭该轴的运动状态位 30 (GRY) 取决于他的其他从站是否处于龙门模式。

注意

(1) : 如果用户使用 EtherCAT 原始模式 , 则 MEN , HMV 和 ASTO 的运动状态可用。

(2) : 如果用户使用 EtherCAT 原始模式并且过程发生错误 , 则 ASTP 位将打开。

G. 中断信号表

PCI-8392(H) 中断项目定义表

PCI-8392(H) 中断项目定义表	
项目	描述
0	轴0中断信号
1	轴1中断信号
...	...
15	轴15中断信号
16	系统中断信号

PCI-8392(H) 轴中断信号定义 (项目 0~15)

PCI-8392(H) 轴中断信号定义 (项目 0~15)								
BitNo	7	6	5	4	3	2	1	0
	IZERO	IWARN	IINP	IEZ	IORG	IMEL	IPEL	IALM
BitNo	15	14	13	12	11	10	9	8
	ISPEL	ITLC	IASTP	INSTP	IDEC	IACC	IVM	ICSTP
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	ISMEL
BitNo	31	30	29	28	27	26	25	24
	--	--	--	-	--	--	--	--

PCI-8392(H) 轴中断信号描述表

PCI-8392(H) 轴中断信号描述			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号开	
1	IPEL	正限位开关打开	
2	IMEL	负限位开关打开	
3	IORG	归零开关打开	
4	IEZ / IEZP	EZ通过信号打开	(1)
5	IINP	就位信号开	
6	IWARN	伺服警告开启	

7	IZSP	零速	
8	ICSTP	命令停止	(2)
9	IVM	处于最大速度	
10	IACC	加速中	
11	IDEC	减速中	
12	INSTP	正常停止(动作完成)	(2)
13	IASTP	异常停止	
14	ITLC	转矩限制控制已打开	
15	ISPEL	SPEL打开	
16	ISMEL	SMEL打开	
17~	预留		

(1), 在 SSCNET 系统中 , 当来自伺服驱动器的零位置信号 (EZ) 打开时 , 即使 EZ 关闭 , EZP 位也将打开。

(2), INSTP : 轴正常停止。 如果轴异常停止 (例如紧急停止和限位开关停止) , 则不会触发该中断信号。 包括归零运动在内的所有运动都可以通过该中断信号进行等待动作。

用户可以通过设置轴参数函数设置正常停止 (运动完成) 条件。

CSTP : 运动命令已停止 , 但轴可能仍在运动。

PCI-8392(H) 系统中断信号的定义(项目 16)

PCI-8392(H) 系统中断信号的定义(项目16)								
BitNo	7	6	5	4	3	2	1	0
								ILNK
BitNo	15	14	13	12	11	10	9	8
BitNo	23	22	21	20	19	18	17	16
BitNo	31	30	29	28	27	26	25	24

PCI-8392(H) 系统中断信号描述表

PCI-8392(H) 系统中断信号描述			
编号	定义	中断条件描述	备注
0	ILNK	当SSCNET链接状态为1-> 0时	

PCI-8253/56 中断信号项目定义表

PCI-8253/56 中断信号项目定义表	
项目	描述
0	轴0中断信号
1	轴1中断信号
...	...

5	轴5中断信号	
---	--------	--

PCI-8253/56 轴中断信号定义(项目 0~5)

PCI-8253/56轴中断信号定义(项目0~5)								
BitNo	7	6	5	4	3	2	1	0
	IZERO	IWARN	IINP	IEZ	IORG	IMEL	IPEL	IALM
BitNo	15	14	13	12	11	10	9	8
	ISPEL	ITLC	IASTP	INSTP	IDEC	IACC	IVM	ICSTP
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	ISMEL
BitNo	31	30	29	28	27	26	25	24
	--	--	--	-	--	--	--	--

PCI-8253/56 轴中断信号描述表

PCI-8253/56 轴中断信号描述表			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号开	
1	IPEL	正限位切换打开	
2	IMEL	负限位开关打开	
3	IORG	归零开关打开	
4	IEZ	EZ信号开	
5	IINP	就位信号开	
6	IWARN	伺服警告开启	
7	IZSP	零速	
8	ICSTP	命令停止	(1)
9	IVM	处于最大速度	
10	IACC	加速中	
11	IDEC	减速中	
12	INSTP	正常停止 (动作完成)	(1)
13	IASTP	异常停止	
14	ITLC	转矩限制控制打开	
15	ISPEL	SPEL打开	
16	ISMEL	SMEL打开	
17~	预留		

(1), INSTP : 轴正常停止。如果轴异常停止（例如紧急停止和限位开关停止），则不会触发该中断信号。包括归零运动在内的所有运动都可以通过该中断信号进行等待动作。

用户可以通过设置轴参数函数设置正常停止（运动完成）条件。

CSTP : 运动命令已停止，但轴可能仍在运动。

DPAC-1000 中断信号项目定义表

DPAC-1000 中断信号项目定义表

中断信号项目定义表								
项目	描述							
0	CPLD 中断							

DPAC-1000 CPLD 中断信号项目定义 (项目 0)

DPAC-1000 CPLD 中断信号项目定义 (项目 0)								
BitNo	7	6	5	4	3	2	1	0
	--	--	--	--	--	--	--	定时器
BitNo	15	14	13	12	11	10	9	8
	--	--	--	--	--	--	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	--
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

DPAC-3000 中断信号项目定义表

DPAC-3000 中断信号项目定义表

中断信号项目定义表								
项目	描述							
0	CPLD 中断							
1	HSL 中断							

DPAC-3000 CPLD 中断信号项目定义 (项目 0)

DPAC-3000 CPLD 中断信号项目定义 (项目 0)								
BitNo	7	6	5	4	3	2	1	0
	--	--	--	--	--	--	--	定时器
BitNo	15	14	13	12	11	10	9	8

	--	--	--	--	--	--	--	--	--
BitNo	23	22	21	20	19	18	17	16	
	--	--	--	--	--	--	--	--	
BitNo	31	30	29	28	27	26	25	24	
	--	--	--	--	--	--	--	--	

DPAC-3000 HSL 中断信号项目定义 (项目1)

DPAC-3000 HSL中断信号项目定义 (项目1)									
BitNo	7	6	5	4	3	2	1	0	
	--	--	--	--	--	--	--	--	DI
BitNo	15	14	13	12	11	10	9	8	
	--	--	--	--	--	--	--	--	
	23	22	21	20	19	18	17	16	
	--	--	--	--	--	--	--	--	---
	31	30	29	28	27	26	25	24	
	--	--	--	--	--	--	--	--	

PCI(e)-7856 中断项目定义表

PCI(e)-7856 中断信号项目定义表

中断信号项目定义表									
项目	描述								
0	CPLD / FPGA 中断								

※ PCI-7856 使用 CPLD 接口，PCIe-7856 使用 FPGA 接口。

PCI(e)-7856 CPLD / FPGA 中断信号项目定义 (项目0)

PCI(e)-7856 CPLD / FPGA 中断信号项目定义 (项目0)									
BitNo	7	6	5	4	3	2	1	0	
	--	--	--	--	--	--	--	--	定时器
BitNo	15	14	13	12	11	10	9	8	
	--	--	--	--	--	--	--	--	
BitNo	23	22	21	20	19	18	17	16	
	--	--	--	--	--	--	--	--	
BitNo	31	30	29	28	27	26	25	24	
	--	--	--	--	--	--	--	--	

PCI-8144 中断项目定义表

PCI-8144 中断信号项目定义表

中断信号项目定义表								
项目	描述							
0	Axis0运动中断							
1	Axis1运动中断							
2	Axis2运动中断							
3	Axis3运动中断							
4	数字输入中断(下降沿)							
5	数字输入中断(上升沿)							

PCI-8144 轴中断信号项目定义(项目0~5)

PCI-8144 轴中断信号项目定义(项目0~5)								
BitNo	7	6	5	4	3	2	1	0
	--	--	--	--	--	--	--	ICSTP

PCI-8144 轴中断信号描述表

PCI-8253/56 轴中断信号描述表								
编号	定义	中断条件描述						备注
0	ICSTP	运动指令输出停止中断C						

PCI-8144 数字中断信号项目定义(项目4)

PCI-8392(H) 系统中断信号的定义(项目16)								
BitNo	7	6	5	4	3	2	1	0
	DI7_F	DI6_F	DI5_F	DI4_F	DI3_F	DI2_F	DI1_F	DI0_F

PCI-8144 数字中断信号描述(项目4)

PCI-8392(H) 系统中断信号描述								
编号	定义	中断条件描述						备注
0	Din_F	数字量输入通道编号下降沿中断						

PCI-8144 数字中断信号项目定义(项目5)

PCI-8392(H) 系统中断信号的定义(项目16)								
BitNo	7	6	5	4	3	2	1	0
	DI7_R	DI6_R	DI5_R	DI4_r	DI3_R	DI2_R	DI1_R	DI0_R

PCI-8144 数字中断信号描述(项目5)

PCI-8392(H) 系统中断信号描述								
编号	定义	中断条件描述						备注
0	Din_R	数字量输入通道编号上升沿中断						

MotionNet 中断项目定义表

MotionNet 轴运动中断信号定义(4XMO(-C))

(MNET-4XMO / MNET-4XMO-C)

4XMO(C) 轴运动中断信号定义								
BitNo	7	6	5	4	3	2	1	0
	IDECE	IDECS	IACCE	IACCS	(*)	(*)	(*)	INSTP
BitNo	15	14	13	12	11	10	9	8
	IORG C	(*)	ICLRC	(*)	ICOMP4	(*)	ISMEL	ISPEL
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	(*)	(*)	(*)	ISD
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

*:已预留。

MotionNet 轴运动中断信号描述表

(MNET-4XMO/ MNET-4XMO-C)

4XMO(C) 轴运动中断信号描述表			
编号	定义	中断条件描述	备注
0	INSTP	正常停止	
1	预留	预留	
2	预留	预留	
3	预留	预留	
4	IACCS	加速开始	
5	IACCE	加速结束	
6	IDECS	减速开始	
7	IDECE	减速结束	
8	ISPEL	正软限位	
9	ISMEL	负软限位	
10	预留	预留	
11	ICOMP4	通用比较器打开	
12	预留	预留	
13	ICLRC	通过CLR输入重置计数器	
14	预留	预留	
15	IORG C	计数器被ORG输入锁存	
16	ISD	SD输入打开	
17	预留	预留	
18	预留	预留	
19	预留	预留	
20~	预留	预留 (始终设为0)	

MotionNet 轴运动中断信号定义(1XMO)

(MNET-1XMO)

1XMO 轴运动中断信号定义								
BitNo	7	6	5	4	3	2	1	0
	ICOMP	ISMEL	ISPEL	IDECE	IDECS	IACCE	IACCS	INSTP
BitNo	15	14	13	12	11	10	9	8
	--	--	--	(*)	ISD	IORG C	(*)	ICLRC
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	--
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

*:已预留。

MotionNet 轴运动中断信号描述表

(MNET-1XMO)

1XMO 轴运动中断信号描述表			
编号	定义	中断条件描述	备注
0	INSTP	正常停止	
1	IACCS	加速开始	
2	IACCE	加速结束	
3	IDECS	减速开始	
4	IDECE	减速结束	
5	ISPEL	正软限位	
6	ISMEL	负软限位	
7	ICOMP	通用比较器打开	
8	ICLRC	通过CLR输入重置计数器	
9	预留	预留	
10	IORG C	计数器被ORG输入锁存	
11	ISD	SD输入打开	
12	预留	预留	
13~	预留	预留 (始终设为0)	

MotionNet 轴错误中断信号定义(4XMO(-C))

(MNET-4XMO/ MNET-4XMO-C)

4XMO(C) 轴错误中断信号定义								
BitNo	7	6	5	4	3	2	1	0
	EALM	EMEL	EPEL	(*)	EGCM	(*)	ENSL	EPSL
BitNo	15	14	13	12	11	10	9	8
	EPCO	EPBO	ESIP	(*)	(*)	ESD	EEMG	(*)

BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	EPAB	EEAB
BitNo	31	30	29	28	27	26	25	24
	--	--	--	--	--	--	--	--

*:已预留。

MotionNet 轴错误中断信号表描述

(MNET-4XMO/ MNET-4XMO-C)

4XMO(C) 轴错误中断信号表描述			
编号	定义	中断条件描述	备注
0	EPSL	正软限位打开且轴已停止	
1	ENSL	负软限位打开且轴已停止	
2	预留		
3	EGCM	通用比较器打开且轴已停止	
4	预留		
5	EPEL	正限位打开且轴已停止	
6	EMEL	负限位打开且轴已停止	
7	EALM	ALM已发生且轴已停止	
8	预留		
9	EEMG	EMG打开且轴已停止	
10	ESD	SD输入打开并且轴减速停止	
11	预留		
12	预留		
13	ESIP	轴从其他轴的错误停止处停止	
14	EPBO	脉冲输入缓冲区溢出并停止	
15	EPCO	插补计数器溢出	
16	EEAB	编码器输入信号错误，但轴未停止	
17	EPAB	脉冲输入信号异常，轴未停止	
18~	预留	预留(始终设为0)	

MotionNet 轴错误中断信号定义(1XMO)

(MNET-1XMO)

1XMO 轴错误中断信号定义								
BitNo	7	6	5	4	3	2	1	0
	EEMG	(*)	EALM	EMEL	EPEL	EGCM	ENSL	EPSL
BitNo	15	14	13	12	11	10	9	8
	--	EPAB	EEAB	ESOR	(*)	ESTN	EPBO	ESD
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	--	--	--	--
BitNo	31	30	29	28	27	26	25	24

	--	--	--	--	--	--	--	--	--
--	----	----	----	----	----	----	----	----	----

*:已预留

MotionNet 轴错误中断信号描述表

(MNET-1XMO)

注意：所有默认值错误信号均已打开。

1XMO 轴中断信号描述表			
编号	定义	中断条件描述	备注
0	EPSL	正软限位打开且轴已停止	
1	ENSL	负软限位打开且轴已停止	
2	EGCM	通用比较器打开且轴已停止	
3	EPEL	正限位打开且轴已停止	
4	EMEL	负限位打开且轴已停止	
5	EALM	ALM已发生且轴已停止	
6	预留		
7	EEMG	EMG打开且轴已停止	
8	ESD	SD输入打开并且轴减速停止	
9	EPBO	脉冲输入缓冲区溢出并停止	
10	ESTN	因通讯错误而停止	
11	预留	预留（始终设为0）	
12	ESOR	无法执行位置覆盖	
13	EEAB	编码器输入信号错误，但轴未停止	
14	EPAB	脉冲输入信号异常，轴未停止	
15~	预留	预留（始终设为0）	

PCI(e)-8154/8158, PCI-8102 中断项目定义表

PCI(e)-8154 中断信号项目定义表

中断信号项目定义表		
项目	描述	
0	轴0错误中断	
1	轴0运动中断	
...		
6	轴3错误中断	
7	轴3运动中断	
.....		
9	DB-8150中断	

PCI(e)-8158 中断信号项目定义表

中断信号项目定义表		
项目	描述	
0	轴0错误中断	

1	轴0运动中断	
...		
14	轴7错误中断	
15	轴7运动中断	
....		
17	DB-8150中断	

PCI-8102 中断信号项目定义表

中断信号项目定义表								
项目	描述							
0	轴0错误中断							
1	轴0运动中断							
2	轴1错误中断							
3	轴1运动中断							
4	GPIO 中断信号							

DB-8150 中断信号定义项目

7	6	5	4	3	2	1	0
EZ1	EZ0	DI1	DI0	L1fin	L0fin	PWM1	PWM0
15	14	13	12	11	10	9	8
--	--	--	--	-	FIFO_full	FIFO_low	FIFO_empty
23	22	21	20	19	18	17	16
--	--	--	--	-	--	--	--
31	30	29	28	27	26	25	24
--	--	--	--	-	--	--	--

DB-8150 中断信号描述表

DB-8150中断信号描述表			
编号	定义	中断条件描述	备注
0	PWM0	PWM0触发事件	
1	PWM1	PWM1触发事件	
2	L0fin	LinearFunction0完成事件	
3	L1fin	LinearFunction1完成事件	
4	DI0	DI0边缘发生	
5	DI1	DI1边缘发生	
6	EZ0	EZ0边缘发生	
7	EZ1	EZ1边缘发生	
8	FIFO_empty	FIFO空事件	

9	FIFO_low	FIFO低事件	
10	FIFO_full	FIFO满事件	
11~31	预留	预留	

PCI(e)-8154/8158, PCI-8102 轴运动中断信号的项目定义

7	6	5	4	3	2	1	0
IDECE	IDECS	IACCE	IACCS	--	IWCOR2	INCBS	INSTP
15	14	13	12	11	10	9	8
IORG	--	ICLRC	ICOMP5	ICOMP4	ICOMP3	ISMEL	ISPEL
23	22	21	20	19	18	17	16
--	--	--	--	ICSTA	--	--	ISD
31	30	29	28	27	26	25	24
--	--	--	--	--	--	--	--

PCI(e)-8154/8158, PCI-8102 轴运动中断信号描述表

PCI(e)-8154/8158, PCI-8102 轴运动中断信号描述表			
编号	定义	中断条件描述	备注
0	INSTP	正常停止	
1	INCBS	缓冲区中的下一条命令启动	
2	IWCOR2	命令预寄存器2为空并且有新命令	
3	预留	预留	
4	IACCS	加速开始	
5	IACCE	加速结束	
6	IDECS	减速开始	
7	IDECE	减速结束	
8	ISPEL	正软限位	
9	ISMEL	负软限位	
10	ICOMP3	错误比较器或比较器3已打开	
11	ICOMP4	通用比较器打开	
12	ICOMP5	触发比较器或比较器5已打开	
13	ICLRC	通过CLR输入重置计数器	
14	预留	预留	
15	IORG	计数器被ORG输入锁存	
16	ISD	SD输入打开	
17	预留	预留	
18	预留	预留	
19	ICSTA	CSTA 输入或APS_start_simultaneous_move 打开	
20~	预留	预留(始终设为0)	

PCI(e)-8154/8158, PCI-8102 轴错误中断的项目定义：(返回代码)

错误中断源是不可屏蔽的，但如果不是超时，则可以从 APS_wait_error_int()的返回代码中获取错误的情况编号。

返回代码	中断条件说明	备注
0	正软限位已打开且轴已停止	
1	负软限位已打开且轴已停止	
2	错误比较器或比较器3打开且轴已停止	
3	通用比较器打开且轴已停止	
4	触发比较器或比较器5开启且轴已停止	
5	正限位已打开且轴已停止	
6	负限位已打开且轴已停止	
7	ALM已发生并且轴已停止	
8	CSTP开启或APS_stop_simultaneous_move开启且轴已停止	
9	CEMG已打开且轴已停止	
10	SD输入打开并且轴减速停止	
11	预留	
12	插补运算错误及停止	
13	轴从其他轴的错误停止处停止	
14	脉冲输入缓冲器溢出并停止	
15	插补计数器溢出	
16	编码器输入信号错误，但轴未停止	
18~	预留	

PCI-8102 GPIO 中断信号的项目定义

7	6	5	4	3	2	1	0
DI3 上升	DI2 上升	DI1 上升	DIO 上升	DI3 下降	DI2 下降	DI1 下降	DIO 下降
15	14	13	12	11	10	9	8
--	--	--	--	--	--	--	--
23	22	21	20	19	18	17	16
--	--	--	--	--	--	--	--
31	30	29	28	27	26	25	24
--	--	--	--	--	--	--	--

PCI-8102 GPIO 中断信号描述表

PCI-8102 GPIO中断信号描述表			
编号	定义	中断条件描述	备注
0	DIO 下降	DIO 下降沿	
1	DI1 下降	DI1 下降沿	
2	DI2 下降	DI2 下降沿	
3	DI3 下降	DI3 下降沿	
4	DIO 上升	DI0 上升沿	
5	DI1 上升	DI1 上升沿	
6	DI2 上升	DI2 上升沿	
7	DI3 上升	DI3 上升沿	
8~	预留	预留	

PCI-C154(+) 中断项目定义表

PCI-C154(+) 中断信号项目定义表

中断信号项目定义表		
项目	描述	
0	轴0错误中断	
1	轴0运动中断	
...		
6	轴3错误中断	
7	轴3运动中断	
8	锁存/比较通道0中断	
...		
11	锁存/比较通道3中断	

PCI-C154(+) 轴运动中断信号的项目定义

7	6	5	4	3	2	1	0
IDECE	IDEC5	IACCE	IACCS	--	IWCOR2	INCBS	INSTP
15	14	13	12	11	10	9	8
IORG5	--	ICLRC	ICOMP5	ICOMP4	ICOMP3	ISMEL	ISPEL
23	22	21	20	19	18	17	16
--	--	--	--	ICSTA	--	--	ISD
31	30	29	28	27	26	25	24
--	--	--	--	--	--	--	--

PCI-C154(+) 轴运动中断信号描述表

PCI-C154(+) 轴运动中断信号描述表			
编号	定义	中断条件描述	备注
0	INSTP	正常停止	

1	INCBS	缓冲区中的下一条命令启动	
2	IWCOR2	命令预寄存器2为空并且有新命令写入	
3	预留	预留	
4	IACCS	加速开始	
5	IACCE	加速结束	
6	IDECS	减速开始	
7	IDECE	减速结束	
8	ISPEL	正软限位	
9	ISMEL	负软限位	
10	ICOMP3	错误比较器打开	
11	ICOMP4	通用比较器打开	
12	ICOMP5	触发比较器打开	
13	ICLRC	通过CLR输入重置计数器	
14	预留	预留	
15	IORG C	计数器被ORG输入锁存	
16	ISD	SD输入打开	
17~18	预留	预留	
19	ICSTA	CSTA输入或 APS_start_simultaneous_move打开	
20~31	预留	预留 (始终设为0)	

PCI-C154(+) 轴错误中断的项目定义：(返回代码)

返回代码	中断条件描述	备注
0	正软限位已打开且轴已停止	
1	负软限位已打开且轴已停止	
2	错误比较器或打开且轴已停止	
3	通用比较器打开且轴已停止	
4	触发比较器开启且轴已停止	
5	正限位已打开且轴已停止	
6	负限位已打开且轴已停止	
7	ALM已发生并且轴已停止	
8	CSTP开启或APS_stop_simultaneous_move开启且轴已停止	
9	CEMG已打开且轴已停止	
10	SD输入打开并且轴减速停止	
11	预留	
12	插补运算错误及停止	

13	轴从其他轴的错误停止处停止					
14	脉冲输入缓冲器溢出并停止					
15	插补计数器溢出					
16	编码器输入信号错误，但轴未停止					
17	脉冲输入信号错误，但轴未停止					
18~	预留					

错误中断源是不可屏蔽的，但如果不是超时，则可以从 APS_wait_error_int()的返回代码中获取错误的情况编号。

PCI-C154(+) 锁存/比较中断信号的项目定义

7	6	5	4	3	2	1	0
CMPE	CMPF	PWMO	LINF	LTCFO	LTCFL	LTCFE	LTCFF
15	14	13	12	11	10	9	8
--	--	--	--	--	--	CMPEO	CMPL
23	22	21	20	19	18	17	16
--	--	--	--	--	--	--	--
31	30	29	28	27	26	25	24
--	--	--	--	--	--	--	--

PCI-C154(+) 锁存/比较中断信号描述表

PCI-C154+ 锁存/比较中断信号描述表			
编号	定义	中断条件描述	备注
0	LTCFF	锁存fifo处于满状态	
1	LTCFE	锁存fifo处于空状态	
2	LTCFL	锁存fifo处于水平状态以上（等于或大于水平）	
3	LTCFO	锁存FIFO处于溢出状态	
4	LINF	线性比较器完成	
5	PWMO	PWM信号重叠	
6	CMPF	比较器处于满状态	
7	CMPE	比较器fifo处于空状态	
8	CMPL	比较器fifo处于水平状态一下（等于或小于水平）	
9	CMPU	比较器fifo处于下溢状态	
10~31	预留	预留（始终设为0）	

PCI-8254/58 / AMP-204/8C 中断项目定义表

PCI-8254/58 / AMP-204/8C 中断信号项目定义表

中断项目定义表	
项目 编号	描述

0~7	轴中断 对于 PCI-8254 , 项目为 0 到 3。 (保留 4~7。) 对于 PCI-8254 , 项目为 0 到 7。	
8	系统中断	
9	DI -上升沿中断	
10	DI -下降沿中断	

PCI-8254/58 / AMP-204/8C 项目 0~7 的轴中断因素定义

PCI-8254/AMP-204C 轴中断因素的定义 , 项目 0~3

PCI-8258/AMP-208C 轴中断因素的定义 , 项目 0~7

轴中断因素描述表			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号开	
1	IPEL	正限位开关打开	
2	IMEL	负限位开关打开	
3	IORG	归零开关打开	
4	IEZ	EZ 通过信号打开	
5	IIINP	就位信号开	
6	IEMG	EMG 信号打开	
7	Reserved	保留 , 始终为 0	
8	ICSTP	命令停止	(*2)
9	IVM	处于最大速度	
10	IACC	加速中	
11	IDEC	减速中	
12	IMDN	运动完成	(*1)
13	IASTP	异常停止	
14	Reserved	保留 , 始终为 0	
15	ISPEL	正软限位中	
16	ISMEL	负软限位中	
17	ISCL	软闭环限位中	
18	IPTB1	P(V)T 缓冲器 1/4 空 (自由尺寸 > 250)	
19	IPTB2	P(V)T 缓冲器 1/2 空 (自由尺寸 > 250)	
20	IPTB3	P(V)T 缓冲区为空 (可用大小 = 1000)	
21~	Reserved	保留 , 始终为 0	

(*1) IMDN : 包括归零运动在内的所有运动 , 可以通过该中断因素进行等待动作。

用户可以通过设置轴参数函数设置正常停止 (运动完成) 条件。

(*2) ICSTP : 运动命令已停止 , 但轴可能仍在运动。

PCI-8254/58 / AMP-204/8C 系统中断因素的定义 (项目 8)

系统中断因素描述表			
编号	定义	中断条件描述	备注

0	IEMG	硬件紧急停止	
1	ILCF0	硬件线性比较器 0 完成事件	
2	ILCF1	硬件线性比较器 1 完成事件	
3	IFCF0	硬件 FIFO 比较器 0 完成事件	
4	IFCF1	硬件 FIFO 比较器 1 完成事件	
5	Reserved	保留	

PCI-8254/58 / AMP-204/8C DI 上升沿中断因素定义 (项目 9)

DI 中断因素描述表			
编号	定义	中断条件描述	备注
0 ~ 7	DI0~DI7	DI0 ~ 7 上升沿事件	
8 ~ 23	TTL0~TTL15	TTL0 ~ 15 上升沿事件	
24	Reserved	预留	

PCI-8254/58 / AMP-204/8C 下降沿中断因素定义 (项目 10)

DI 中断因素描述表			
编号	定义	中断条件描述	备注
0 ~ 7	DI0~DI7	DI0 ~ 7 下降沿事件	
8 ~ 23	TTL0~TTL15	TTL0 ~ 15 下降沿事件	
24	Reserved	预留	

PCIe-833x 中断项目定义表

PCIe-8332 中断项目定义表

中断项目定义表			
编号	描述		
0~15	轴中断 (16 轴)		
64	系统中断		
65	DI-上升沿中断		
66	DI-下降沿中断		

PCIe-8334 中断项目定义表

中断项目定义表			
编号	描述		
0~31	轴中断 (32 轴)		
64	系统中断		
65	DI-上升沿中断		
66	DI-下降沿中断		

PCIe-8338 中断项目定义表

中断项目定义表			
编号	描述		
0~63	轴中断 (64 轴)		

64	系统中断	
65	DI-上升沿中断	
66	DI-下降沿中断	

PCIe-8332 轴中断因素定义 (项目 0-15)

中断因素描述表			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号打开	
1	IPEL	正端限位开关打开	
2	IMEL	负端限位开关打开	
3	IORG	归零开关已打开	
4	Reserved	预留	
5	IIINP	就位	
6	IEMG	EMG 信号已打开	
7	Reserved	预留，总是为 0	
8	ICSTP	命令停止	(*2)
9	IVM	最大速度中	
10	IACC	加速中	
11	IDEC	减速中	
12	IMDN	运动完成	(*1)
13	IASTP	异常停止	
14	Reserved	保留，始终为 0	
15	ISPEL	正软限位中	
16	ISMEL	负软限位中	
17	ISCL	软闭环限位中	
18	IPTB1	P(V)T 缓冲器 1/4 空 (自由尺寸 > 250)	
19	IPTB2	P(V)T 缓冲器 1/2 空 (自由尺寸 > 250)	
20	IPTB3	P(V)T 缓冲区为空 (可用大小 = 1000)	
21~	Reserved	保留，始终为 0	

(*1) IMDN : 包括归零运动在内的所有运动，可以通过该中断因素进行等待动作。

用户可以通过设置轴参数函数设置正常停止（运动完成）条件。

(*2) ICSTP : 运动命令已停止，但轴可能仍在运动。

PCIe-8334 轴中断因素定义 (项目 0~31)

轴中断因素描述表			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号打开	
1	IPEL	正端限位开关打开	
2	IMEL	负端限位开关打开	
3	IORG	归零开关已打开	

4	Reserved	预留	
5	IINP	就位	
6	IEMG	EMG 信号已打开	
7	Reserved	预留，总是为 0	
8	ICSTP	命令停止	(*2)
9	IVM	最大速度中	
10	IACC	加速中	
11	IDEC	减速中	
12	IMDN	运动完成	(*1)
13	IASTP	异常停止	
14	Reserved	保留，始终为 0	
15	ISPEL	正软限位中	
16	ISMEL	负软限位中	
17	ISCL	软闭环限位中	
18	IPTB1	P(V)T 缓冲器 1/4 空 (自由尺寸> 250)	
19	IPTB2	P(V)T 缓冲器 1/2 空 (自由尺寸> 250)	
20	IPTB3	P(V)T 缓冲区为空 (可用大小= 1000)	
21~	Reserved	保留，始终为 0	

(*1) IMDN：包括归零运动在内的所有运动，可以通过该中断因素进行等待动作。

用户可以通过设置轴参数函数设置正常停止（运动完成）条件。

(*2) ICSTP：运动命令已停止，但轴可能仍在运动。

PCIe-8338 轴中断因素描述表 (项目 0~63)

PCIe-8338 轴中断因素描述表			
编号	定义	中断条件描述	备注
0	IALM	伺服报警信号打开	
1	IPEL	正端限位开关打开	
2	IMEL	负端限位开关打开	
3	IORG	归零开关已打开	
4	Reserved	预留	
5	IINP	就位	
6	IEMG	EMG 信号已打开	
7	Reserved	预留，总是为 0	
8	ICSTP	命令停止	(*2)
9	IVM	最大速度中	
10	IACC	加速中	
11	IDEC	减速中	

12	IMDN	运动完成	(*1)
13	IASTP	异常停止	
14	Reserved	保留, 始终为 0	
15	ISPEL	正软限位中	
16	ISMEL	负软限位中	
17	ISCL	软闭环限位中	
18	IPTB1	P(V)T 缓冲器 1/4 空 (自由尺寸> 250)	
19	IPTB2	P(V)T 缓冲器 1/2 空 (自由尺寸> 250)	
20	IPTB3	P(V)T 缓冲区为空 (可用大小= 1000)	
21~	Reserved	保留, 始终为 0	

(*1) IMDN : 包括归零运动在内的所有运动, 可以通过该中断因素进行等待动作。

用户可以通过设置轴参数函数设置正常停止(运动完成)条件。

(*2) ICSTP : 运动命令已停止, 但轴可能仍在运动。

PCIe-833x 系统中断因素的定义 (项目 64)

System 中断因素描述表			
编号	定义	中断条件描述	备注
0	IEMG	硬件紧急停止	

PCIe-833x DI 上升沿中断因素定义

DI 中断因素描述表			
编号	定义	中断条件描述	备注
0 ~ 3	DI0~DI3	DI0 ~ 3 上升沿事件	

PCIe-833x DI 下降沿中断因素定义 (项目 66)

DI 中断因素描述表			
编号	定义	中断条件描述	备注
0 ~ 3	DI0~DI3	DI0 ~ 3 下降沿事件	

H. 现场总线参数表

PCI-8392H HSL 参数表				
编号	定义	描述	值	默认值
00h	PRF_COMMUNICATION_TYPE	现场总线通讯类型	0: 半双 1: 全双工	1
01h	PRF_TRANSFER_RATE	网络传输速率。	1: 3 Mbps 2: 6 Mbps 3: 12 Mbps	2

02h	PRF_HUB_NUMBER	集线器(Hub)总数。	0~7	0
03h	PRF_INITIAL_TYPE	连接从站模块时，将数字输出复位为零。 1：取决于从站状态。	0：将数字输出复位为零。 1：取决于从站状态。	0
04h	PRF_CHKERRCNT_LAYER	设置检查错误计数的程度	1~7	7

PCI(e)-7856 MNET 参数表				
编号	定义	描述	值	默认值
00h	预留			
01h	PRF_TRANSFER_RATE	网络传输速率。	0: 2.5Mbps 1: 5 Mbps 2: 10 Mbps 3: 20 Mbps	3
02h~	预留			

PCI(e)-7856 HSL 参数表				
编号	定义	描述	值	默认值
00h	PRF_COMMUNICATION_TYPE	现场总线通讯类型	0: 半双 1: 全双工	1
01h	PRF_TRANSFER_RATE	网络传输速率。	1: 3 Mbps 2: 6 Mbps 3: 12 Mbps	2
02h	PRF_HUB_NUMBER	集线器(Hub)总数。	0~7	0
03h	PRF_INITIAL_TYPE	连接从站模块时，将数字输出复位为零。	0：将数字输出复位为零。 1：取决于从站状态。	0
04h	PRF_CHKERRCNT_LAYER	设置检查错误计数的程度	1~7	7

DPAC-3000 MNET 参数表

编号	定义	描述	值	默认值
00h	预留			
01h	PRF_TRANSFER_RATE	网络传输速率。	0: 2.5Mbps 1: 5 Mbps 2: 10 Mbps 3: 20 Mbps	3
02h~	预留			

DPAC-3000 HSL 参数表				
编号	定义	描述	值	默认值
00h	PRF_COMMUNICATION_TYPE	现场总线通讯类型	0 : 半双工 1 : 全双工	1
01h	PRF_TRANSFER_RATE	网络传输速率。	1: 3 Mbps 2: 6 Mbps 3: 12 Mbps	2
02h	PRF_HUB_NUMBER	集线器(Hub)总数。	0~7	0
03h	PRF_INITIAL_TYPE	连接从站模块时，将数字输出复位为零。	0 : 将数字输出复位为零。 1 : 取决于从站状态。	0
04h	PRF_CHKERRCNT_LAYER	设置检查错误计数的程度	1~7	7

I. 龙门参数表

PCI-8253/56 龙门参数表				
参数编号	定义	描述	参数数据含义	默认值
00h	GANTRY_模式	启用/禁用龙门关系。	0: 禁用 1: 启用	0
01h	GENTRY_DEVIATION	设置偏差保护。 如果偏差超过此设置，则轴将关闭伺服。	I32正值。	8,000
02h	GENTRY_DEVIATION_STP	设置偏差保护。 如果偏差超过该设	I32正值。	5,000

		置，则轴将停止。		
--	--	----------	--	--

PCI-8392(H) 龙门参数表				
参数编号	定义	描述	参数数据含义	默认值
00h	GANTRY_模式	启用/禁用龙门关系。	0: 禁用 1: 启用	0
01h	GENTRY_DEVIATION	设置偏差保护。 如果偏差超过此设置，则轴将关闭伺服。	I32正值。	8,000
02h	GENTRY_DEVIATION_STP	设置偏差保护。 如果偏差超过该设置，则轴将停止。	I32正值。	5,000

J. 触发参数表

PCI-8253/56 触发参数表

NO	定义	描述	值	默认值:
0x00	TG_LCMP0_SRC	线性比较0 (LCMP0) 源	0 ~ 5: 编码器计数器 0~5	0
0x01	TG_LCMP1_SRC	线性比较1 (LCMP1) 源	0 ~ 5: 编码器计数器 0~5	2
0x02	TG_TCMP0_SRC	表比较0 (TCMP0) 源	0 ~ 5: 编码器计数器 0~5	1
0x03	TG_TCMP1_SRC	表比较1 (TCMP1) 源	0 ~ 5: 编码器计数器 0~5	4
0x04	TG_LCMP0_EN	启用线性比较0 (LCMP0)	0: 禁用, 1:启用	0
0x05	TG_LCMP1_EN	启用线性比较1 (LCMP1)	0: 禁用, 1:启用	0
0x06	TG_TCMP0_EN	启用表比较0 (TCMP0)	0: 禁用, 1:启用	0
0x07	TG_TCMP1_EN	启用表比较1 (TCMP1)	0: 禁用, 1:启用	0
0x10	TG_TRG0_SRC	触发输出0 (TRG0) 源	0:无 1:LCMP0 (默认值) 2:LCMP1	1

			4:FCMP0 8:FCMP1 16: TMR	
0x11	TG_TRG1_SRC	触发输出1 (TRG1) 源	0:无 1:LCMP0 2:LCMP1 4:FCMP0 (默认值) 8:FCMP1 16: TMR	4
0x12	TG_TRG2_SRC	触发输出2 (TRG2) 源 (*1)	0:无 1:LCMP0 2:LCMP1 (默认值) 4:FCMP0	2
			8:FCMP1 16: TMR	
0x13	TG_TRG3_SRC	触发输出3 (TRG3) 源 (*1)	0:无 1:LCMP0 2:LCMP1 4:FCMP0 8:FCMP1 (默认值) 16: TMR	8
0x14	TG_TRG0_PWD	TRG0 脉冲宽度	脉冲宽度 = (N + 2) * 20 ns 24位值. 0 ~ 16777215	0 (40 ns)
0x15	TG_TRG1_PWD	TRG1 脉冲宽度	脉冲宽度 = (N + 2) * 20 ns 24位值. 0 ~ 16777215	0 (40 ns)
0x16	TG_TRG2_PWD	TRG2 脉冲宽度 (*1)	脉冲宽度 = (N + 2) * 20 ns 24位值. 0 ~ 16777215	0 (40 ns)
0x17	TG_TRG3_PWD	TRG3 脉冲宽度 (*1)	脉冲宽度 = (N + 2) * 20 ns 24位值. 0 ~ 16777215	0 (40 ns)
0x18	TG_TRG0_CFG	TRG 0 配置	Bit 0: 脉冲逻辑逆。 Bit 1: 脉冲 (0) / 切换 (切换(toggle)) (1) Bit 2~31: 预留 (设为0)	0
0x19	TG_TRG1_CFG	TRG 1 配置	Bit 0: 脉冲逻辑逆。 Bit 1: 脉冲 (0) / 切换 (toggle) (1)	0

			Bit 2~31: 预留 (设为0)	
0x1A	TG_TRG2_CFG	TRG 2 配置 (*1)	Bit 0: 脉冲逻辑逆。 Bit 1: 脉冲 (0) / 切换 (切换(toggle)) (1) Bit 2~31: 预留 (设为0)	0
0x1B	TG_TRG3_CFG	TRG 3 配置 (*1)	Bit 0: 脉冲逻辑逆。 Bit 1: 脉冲 (0) / 切换 (toggle) (1) Bit 2~31: 预留 (设为0)	0
0x20	TMR_ITV	计时器间隔	计时器间隔 = (N+2) * 20 ns 28位值. 0~268435455	(40 ns)
0x21	TMR_EN	启用计时器	0: 禁用, 1:启用	0

*1: 仅 PCI-8256.

MNET-4XMO-C 触发参数表

NO	定义	描述	值	默认值:
0x00	TG_CMP0_SRC	比较0源	0: 指定计数器 1 : 位置计数器	0
0x01	TG_CMP1_SRC	比较1源	0: 指定计数器 1 : 位置计数器	0
0x02	TG_CMP2_SRC	比较2源	0: 指定计数器 1 : 位置计数器	0
0x03	TG_CMP3_SRC	比较3源	0 : 命令计数器 1 : 位置计数器	0
0x04	TG_CMP0_EN	启动比较0	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	0
0x05	TG_CMP1_EN	启动比较1	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关)	0

			2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	
0x06	TG_CMP2_EN	启动比较2	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关)	0
			2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	
0x07	TG_CMP3_EN	启动比较3	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	0
0x08	TG_CMP0_TYPE	比较0类型	0: 表比较, 1: 线性比较	0
0x09	TG_CMP1_TYPE	比较1类型	0: 表比较, 1: 线性比较	0
0x0A	TG_CMP2_TYPE	比较2类型	0: 表比较, 1: 线性比较	0
0x0B	TG_CMP3_TYPE	比较3类型	0: 表比较, 1: 线性比较	0
0x0C	TG_CMPPH_EN	启用比较H	0: 禁用, 1:启用	0
0x0D	TG_CMPPH_DIR_EN	比较H方向启用	0: 禁用, 1:启用	0
0x0E	TG_CMPPH_DIR	比较H方向	0: 正方向, 1: 反方向.	0
0x10	TG_TRG0_SRC	触发输出0 (TRG0) 源	Bit 0: CMP 0 Bit 1: CMP 1 Bit 2: CMP 2 Bit 3: CMP	1

			3 Bit 4:CMP H 值: 0x00 ~ 0x1f	
0x11	TG_TRG1_SRC	触发输出1 (TRG1) 源	Bit 0:CMP 0 Bit 1:CMP 1 Bit 2:CMP 2 Bit 3:CMP 3	2
			Bit 4:CMP H 值: 0x00 ~ 0x1f	
0x12	TG_TRG2_SRC	触发输出2 (TRG2) 源	Bit 0:CMP 0 Bit 1:CMP 1 Bit 2:CMP 2 Bit 3:CMP 3 Bit 4:CMP H 值: 0x00 ~ 0x1f	4
0x13	TG_TRG3_SRC	触发输出3 (TRG3) 源	Bit 0:CMP 0 Bit 1:CMP 1 Bit 2:CMP 2 Bit 3:CMP 3 Bit 4:CMP H 值: 0x00 ~ 0x1f	8
0x14	TG_TRG0_PWD	TRG0 脉冲宽度	脉冲宽度 = (N+ 5) * 10 ns 值: 0x05 ~ 0x7fffffff 小于0x05的值视为0x05。	5 (100 ns)
0x15	TG_TRG1_PWD	TRG1 脉冲宽度	脉冲宽度 = (N+ 5) * 10 ns 值: 0x05 ~ 0x7fffffff 小于0x05的值视为0x05。	5 (100 ns)
0x16	TG_TRG2_PWD	TRG2 脉冲宽度	脉冲宽度 = (N+ 5) * 10 ns 值: 0x05 ~ 0x7fffffff 小于0x05的值视为0x05。	5 (100 ns)
0x17	TG_TRG3_PWD	TRG3 脉冲宽度	脉冲宽度 = (N+ 5) * 10 ns 值: 0x05 ~ 0x7fffffff 小于0x05的值视为	5 (100 ns)

			0x05。	
0x18	TG_TRG0_CFG	TRG 0 配置	Bit 0: 脉冲逻辑逆。 不逆(0) / 逆 (1) Bit 1~2: 脉冲 (0) / 切换(toggle) (1) / ByPass (2) / 禁用	0
			(3) Bit 3~31: 预留 (设为 0)	
0x19	TG_TRG1_CFG	TRG 1 配置	Bit 0: 脉冲逻辑逆。 不逆(0) / 逆 (1) Bit 1~2: 脉冲 (0) / 切换(toggle) (1) / ByPass (2) / 禁 用 (3) Bit 3~31: 预留 (设为 0)	0
0x1A	TG_TRG2_CFG	TRG 2 配置	Bit 0: 脉冲逻辑逆。 不逆(0) / 逆 (1) Bit 1~2: 脉冲 (0) / 切换(toggle) (1) / ByPass (2) / 禁 用 (3) Bit 3~31: 预留 (设为 0)	0
0x1B	TG_TRG3_CFG	TRG 3 配置	Bit 0: 脉冲逻辑逆。 不 逆(0) / 逆 (1) Bit 1~2: 脉冲 (0) / 切 换(toggle) (1) / ByPass (2) / 禁 用 (3) Bit 3~31: 预留 (设为 0)	0
0x20	TG_ENCH_CFG	Encoder H 配置	Bit 0: 启用过滤器. 1: 启用, 0: 禁用. Bit 1: 计数器方向逆。 0 : 不逆 , 1 : 逆. Bit 2~4: 解码器模式。 0x00: OUT/DIR, 0x01:	0

			CW/CCW, 0x02: 1XAB, 0x03: 2XAB, 0x04: 4XAB.	
--	--	--	--	--

HSL-4XMO 触发参数表

NO	定义	描述	值	默认值:
0x00	TG_CMP0_SRC	比较0源	0: 指定计数器 1 : 位置计数器	0
0x01	TG_CMP1_SRC	比较1源	0: 指定计数器 1 : 位置计数器	0
0x02	TG_CMP2_SRC	比较2源	0: 指定计数器 1 : 位置计数器	0
0x03	TG_CMP3_SRC	比较3源	0 : 命令计数器 1 : 位置计数器	0
0x04	TG_CMP0_EN	启动比较0	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	0
0x05	TG_CMP1_EN	启动比较1	0: 禁用 其他 : 启用. 1:数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	0
0x06	TG_CMP2_EN	启动比较2	0: 禁用 其他 : 启用.	0

			1:数据= cmp计数器 (与计数方向无关) 2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	
0x07	TG_CMP3_EN	启动比较3	0: 禁用 其他：启用. 1:数据= cmp计数器 (与计数方向无关)	0
			2 : 数据= cmp计数器 (上计数) 3 : 数据= cmp计数器 (下计数) 4 : 数据> cmp计数器 5 : 数据< cmp计数器	
0x08	保留			
0x09	保留			
0x0A	保留			
0x0B	保留			
0x0C	保留			
0x0D	保留			
0x0E	保留			
0x10	保留			
0x11	保留			
0x12	保留			
0x13	保留			
0x14	保留			
0x15	保留			
0x16	保留			
0x17	保留			
0x18	TG_TRG0_CFG	TRG 0 配置	不逆(0) / 逆 (1)	0
0x19	TG_TRG1_CFG	TRG 1 配置	不逆(0) / 逆 (1)	0
0x1A	TG_TRG2_CFG	TRG 2 配置	不逆(0) / 逆 (1)	0
0x1B	TG_TRG3_CFG	TRG 3 配置	不逆(0) / 逆 (1)	0
0x21	TG_CMP0_DIR	比较0方向	0: 正方向, 1: 反方向.	0

0x22	TG_CMP1_DIR	比较1方向	0: 正方向, 1: 反方向.	0
0x23	TG_CMP2_DIR	比较2方向	0: 正方向, 1: 反方向.	0
0x24	TG_CMP3_DIR	比较3方向	0: 正方向, 1: 反方向.	0

DB-8150 触发参数表

编号	定义	描述	值	默认值:
0x00	TG_PWM0_脉冲_WIDTH	设置PWM脉冲宽度(CH0)	1~65535 注意: 脉冲宽度(纳秒) = 参数 * 100 + 85	0x3E 7 (999) (100us ec)
0x01	TG_PWM1_脉冲_WIDTH	设置PWM脉冲宽度(CH1)	1~65535 注意: 脉冲宽度(纳秒) = 参数 * 100 + 85	0x3E 7 (999) (100us ec)
0x02	TG_PWM0_模式	选择脉冲输出或电平开关输出(CH0)	0: 脉冲输出 1: 电平开关输出 (切换(toggle) 输出)	0
0x03	TG_PWM1_模式	选择脉冲输出或电平开关输出(CH1)	0: 脉冲输出 1: 电平开关输出 (切换(toggle) 输出)	0
0x04	TG_计时器0_INTER VAL	设置计时器间隔 (CH0)	0~1073741823 注意: 计时器循环时间(纳秒) = (间隔 + 5) * 25	0 (125n s ec)
0x05	TG_计时器1_INTER VAL	设置计时器间隔 (CH1)	0~1073741823 注意: 计时器循环时间(纳秒) = (间隔 + 5) * 25	0 (125n s ec)
0x06	TG_ENC0_CNT_DIR	设置编码器计数方向(CH0)	0: 不逆 1: 逆	0
0x07	TG_ENC1_CNT_DIR	设置编码器计数方向(CH1)	0: 不逆 1: 逆	0
0x08	TG_IPT0_模式	设置脉冲输入模式(CH0)	0: OUT/DIR 1: CW/CCW 2: 1x AB相位	0

			3: 2x AB相位 4: 4x AB相位	
0x09	TG_IPT1_模式	设置脉冲输入模式 (CH1)	0: OUT/DIR 1: CW/CCW 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	0
0x0A	TG_EZ0_CLEAR_EN	启用EZ清除 (CH0)	0: 禁用 1: 启用	0
0x0B	TG_EZ1_CLEAR_EN	启用EZ清除 (CH1)	0: 禁用 1: 启用	0
0x0C	TG_EZ0_CLEAR_LOGIC	清除逻辑设置 (CH0)	0: 下降沿 1: 上升沿	0
0x0D	TG_EZ1_CLEAR_LOGIC	清除逻辑设置 (CH1)	0: 下降沿 1: 上升沿	0
0x0E	TG_CNT0_SOURCE	设置计数器的来源 (CH0)	0: 编码器0 (载板EA/B 0) 1: 编码器1 (载板EA/B 1) 2: 编码器2 (子板DEA/B 2) 3: 编码器3 (子板DEA/B 3) 4: 计时器0 5: 计时器1	0x2
0x0F	TG_CNT1_SOURCE	设置计数器的来源 (CH1)	0: 编码器0 (载板EA/B 0) 1: 编码器1 (载板EA/B 1) 2: 编码器2 (子板DEA/B 2) 3: 编码器3 (子板DEA/B 3) 4: 计时器0 5: 计时器1	0x3
0x10	TG_FTR0_EN	启用过滤器 (CH0)	0: 禁用 1: 启用	0
0x11	TG_FTR1_EN	启用过滤器 (CH1)	0: 禁用 1: 启用	0
0x12	TG_DI_LATCH0_EN	启用DI锁存 (CH0)	0: 禁用	0

			1: 启用 0: 禁用	
0x13	TG_DI_LATCH1_EN	启用DI锁存 (CH1)	0: 禁用 1: 启用	0
0x14	TG_DI_LATCH0_ED	设置DI锁存条件(CH0)	0: DI下降沿至锁存	0
	GE		1: DI上升沿至锁存	
0x15	TG_DI_LATCH1_ED GE	设置DI锁存条件(CH1)	0: DI下降沿至锁存 1: DI上升沿至锁存	0
0x16	TG_DI_LATCH0_VA LUE	获取DI锁存值 (CH0)		
0x17	TG_DI_LATCH1_VA LUE	获取DI锁存值 (CH1)		
0x18	TG_TRGOUT_MAP	设置触发输出映射	0~65535 (Bit16~Bit31 预留) *注意(1)	0x9
0x19	TG_TRGOUT_LOGI C	设置触发输出逻辑	0~255 (Bit8~Bit31 预留) *注意(2)	0
0x1A	TG_FIFO_LEVEL	设置/获取FIFO大小级别	0: level=0 (空) 1: level=1/4 2: level=1/2 (默认值) 3: level=3/4 注意:仅支持 CH0	0
0x1B	TG_PWM0_SOUR CE	设置PWM源 (CH0)	Bit 0: 计时器 0: 禁用 1: 启用 Bit 1: 线性比较 0: 禁用 1: 启用 Bit 2: FIFO 比较 0: 禁用 1: 启用 其他位预留 注意: FIFO比较器仅支 持CH0	0x4 (FIFO 比较 器)
0x1C	TG_PWM1_SOUR CE	设置PWM源 (CH1)	Bit 0: 计时器 0: 禁用 1: 启用 Bit 1: 线性比较 0: 禁用	0x4 (FIFO 比较 器)

			1: 启用	
			Bit 2: FIFO 比较 0: 禁用 1: 启用 其他位预留 注 意: FIFO比较器 仅支持CH0	

*注意(1)

Bit	7	6	5	4	3	2	1	0
函数	TRG3b	TRG3a	TRG2b	TRG2a	TRG1b	TRG1a	TRG0b	TRG0a
Bit	15	14	13	12	11	10	9	8
函数	TRG7b	TRG7a	TRG6b	TRG6a	TRG5b	TRG5a	TRG4b	TRG4a

DB-8150具有8个触发输出引脚和2个PWM通道。

通过此函数，触发输出引脚可以映射到2个PWM通道。

TRG0~TRG7符号代表触发输出引脚的pin0~pin7。

“ a” 符号代表PWM0。

“ b” 符号代表PWM1。

例如：

TRG0a = 1表示PWM0信号将由触发输出引脚0输出。

TRG0a = 0表示触发输出引脚0将不输出PWM0信号。

如果TRG0a和TRG0b同时设为1，则引脚0将通过使用或运算符的PWM0和PWM1输出信号。

*注意(2)

Bit	7	6	5	4	3	2	1	0
函数	TRGInv7	TRGInv6	TRGInv5	TRGInv4	TRGInv3	TRGInv2	TRGInv1	TRGInv0

该参数用于设置触发输出信号的逻辑。

例如：

TRGInv0 = 1表示触发输出信号将被pin0逆转。

TRGInv0 = 0表示触发输出信号不会被pin0逆转。

PCI - C154(+) 触发参数表

编号	定义	描述	值	默认值:
0x00	TIG_ENC_IPT_模式0	编码器脉冲输入模式 (CH0)	0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位0 3: 2x AB相位 4: 4x AB相位	4
0x01	TIG_ENC_IPT_模式1	编码器脉冲输入模式 (CH1)	0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位0 3: 2x AB相位 4: 4x AB相位	4
0x02	TIG_ENC_IPT_模式2	编码器脉冲输入模式 (CH2)	0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位0 3: 2x AB相位 4: 4x AB相位	4
0x03	TIG_ENC_IPT_模式3	编码器脉冲输入模式 (CH3)	0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位0 3: 2x AB相位 4: 4x AB相位	4
0x08	TIG_ENC_EA_INV0	逆转EA编码器信号 (CH0) 注意(3)	1: 逆 0: 不逆	0
0x09	TIG_ENC_EA_INV1	逆转EA编码器信号 (CH1) 注意(3)	1: 逆 0: 不逆	0
0x0A	TIG_ENC_EA_INV2	逆转EA编码器信号	1: 逆	0

		(CH2) 注意(3)	0: 不逆	
0x0B	TIG_ENC_EA_INV3	逆转EA编码器信号 (CH3) 注意(3)	1: 逆 0: 不逆	0
0x10	TIG_ENC_EB_INV0	逆转EB编码器信号 (CH0) 注意(3)	1: 逆 0: 不逆	0
0x11	TIG_ENC_EB_INV1	逆转EB编码器信号 (CH1) 注意(3)	1: 逆 0: 不逆	0
0x12	TIG_ENC_EB_INV2	逆转EB编码器信号 (CH2) 注意(3)	1: 逆 0: 不逆	0
0x13	TIG_ENC_EB_INV3	逆转EB编码器信号 (CH3) 注意(3)	1: 逆 0: 不逆	0
0x28	TIG_ENC_SIGNAL_FILITE R_EN0	CH0 编码器信 号低通滤波器 (截止频率 (3db) : 5MHz)	1: 启用 0: 禁用	1
0x29	TIG_ENC_SIGNAL_FILITE R_EN1	CH1 编码器信 号低通滤波器 (截止频率 (3db) : 5MHz)	1: 启用 0: 禁用	1
0x2A	TIG_ENC_SIGNAL_FILITE R_EN2	CH2 编码器信 号低通滤波器 (截止频率 (3db) : 5MHz)	1: 启用 0: 禁用	1
0x2B	TIG_ENC_SIGNAL_FILITE R_EN3	CH3 编码器信 号低通滤波器 (截止频率 (3db) : 5MHz)	1: 启用 0: 禁用	1
0x30	TIG_TIMER8_DIR	计时器8方向注意 (1)	0: 正数 1: 负数	0
0x31	TIG_TIMER8_ITV	计时器8间隔注意 (1)	计时器8间隔 (秒) = 值 * 30nS	0
0x32	TIG_CMP0_SRC	比较CH0源	0: 编码器计数器 0 1:计时器8计数器 注意(1)	0
0x33	TIG_CMP1_SRC	比较CH1源	0: 编码器计数器 1 1:计时器8计数器 注意(1)	0
0x34	TIG_CMP2_SRC	比较CH2源	0: 编码器计数器 2 1:计时器8计数器	0

			注意(1)	
0x35	TIG_CMP3_SRC	比较CH3源	0: 编码器计数器 3 1:计时器8计数器 注意(1)	0
0x42	TIG_TRGOUT0_MAP	触发输出引脚 (CH0) 映射	0: Output_Pin_0 1: Output_Pin_1 2: Output_Pin_2 3: Output_Pin_3	0
0x43	TIG_TRGOUT1_MAP	触发输出引脚 (CH1) 映射	0: Output_Pin_0 1: Output_Pin_1 2: Output_Pin_2 3: Output_Pin_3	0
0x44	TIG_TRGOUT2_MAP	触发输出引脚 (CH2) 映射	0: Output_Pin_0 1: Output_Pin_1 2: Output_Pin_2 3: Output_Pin_3	0
0x45	TIG_TRGOUT3_MAP	触发输出引脚 (CH3) 映射	0: Output_Pin_0 1: Output_Pin_1 2: Output_Pin_2 3: Output_Pin_3	0
0x4A	TIG_TRGOUT0_LOGIC	触发输出引脚 (CH0) 逻辑	0: 不逆 1: 逆	0
0x4B	TIG_TRGOUT1_LOGIC	触发输出引脚 (CH1) 逻辑	0: 不逆 1: 逆	0
0x4C	TIG_TRGOUT2_LOGIC	触发输出引脚 (CH2) 逻辑	0: 不逆 1: 逆	0
0x4D	TIG_TRGOUT3_LOGIC	触发输出引脚 (CH3) 逻辑	0: 不逆 1: 逆	0
0x52	TIG_PWM0_脉冲_WIDT H	PWM 脉冲宽度 (CH0)	PWM脉冲宽度(秒) = 值 * 480nS + 60nS 取值范围: 1~8191	2
0x53	TIG_PWM1_脉冲_WIDT H	PWM 脉冲宽度 (CH1)	PWM脉冲宽度(秒) = 值 * 480nS + 60nS 取值范围: 1~8191	2
0x54	TIG_PWM2_脉冲_WIDT H	PWM 脉冲宽度 (CH2)	PWM脉冲宽度(秒) = 值 * 480nS + 60nS 取值范围: 1~8191	2
0x55	TIG_PWM3_脉冲_WIDT H	PWM 脉冲宽度 (CH3)	PWM脉冲宽度(秒) = 值 * 480nS + 60nS	2

			取值范围: 1~8191	
0x5A	TIG_PWM0_模式	选择脉冲输出或电平开关输出(CH0)	0: 脉冲输出 1: 电平开关输出(切换(toggle) 输出)	0
0x5B	TIG_PWM1_模式	选择脉冲输出或电平开关输出(CH1)	0: 脉冲输出 1: 电平开关输出(切换(toggle) 输出)	0
0x5C	TIG_PWM2_模式	选择脉冲输出或电平开关输出(CH2)	0: 脉冲输出 1: 电平开关输出(切换(toggle) 输出)	0
0x5D	TIG_PWM3_模式	选择脉冲输出或电平开关输出(CH3)	0: 脉冲输出 1: 电平开关输出(切换(toggle) 输出)	0
0x62	TIG_计时器0_ITV	计时器0间隔 (仅写入) 注意(2)	计时器0间隔值(秒) = 值 * 30nS	0
0x63	TIG_计时器1_ITV	计时器1间隔 (仅写入) 注意(2)	计时器1间隔值(秒) = 值 * 30nS	0
0x64	TIG_TIMER2_ITV	计时器2间隔 (仅写入) 注意(2)	计时器2间隔值(秒) = 值 * 30nS	0
0x65	TIG_TIMER3_ITV	计时器3间隔 (仅写入) 注意(2)	计时器3间隔值(秒) = 值 * 30nS	0
0x6A	TIG_FIFO_LEVEL0	FIFO比较器 (CH0) 电平	FIFO比较器 (CH0) 电平值 (0~1023)	0
0x6B	TIG_FIFO_LEVEL1	FIFO比较器 (CH1) 电平	FIFO比较器 (CH1) 电平值 (0~1023)	0
0x6C	TIG_FIFO_LEVEL2	FIFO比较器 (CH2) 电平	FIFO比较器 (CH2) 电平值 (0~1023)	0
0x6D	TIG_FIFO_LEVEL3	FIFO比较器 (CH3) 电平	FIFO比较器 (CH3) 电平值 (0~1023)	0
0x72	TIG_OUTPUT_EN0	启用触发输出引脚 (CH0)	0: 禁用 1: 启用	1
0x73	TIG_OUTPUT_EN1	启用触发输出引脚 (CH1)	0: 禁用 1: 启用	1
0x74	TIG_OUTPUT_EN2	启用触发输出引脚 (CH2)	0: 禁用 1: 启用	1

0x75	TIG_OUTPUT_EN3	启用触发输出引脚 (CH3)	0: 禁用 1: 启用	1
------	----------------	-------------------	----------------	---

注意 (1) 计时器 8 用于模拟编码器。它用作 4 通道的比较器源。

注意 (2) 定时器 0~3 用于周期性地产生触发信号。

注意 (3) 当用户更改一次 EA/EB 逻辑时，编码器计数器将计数一次。

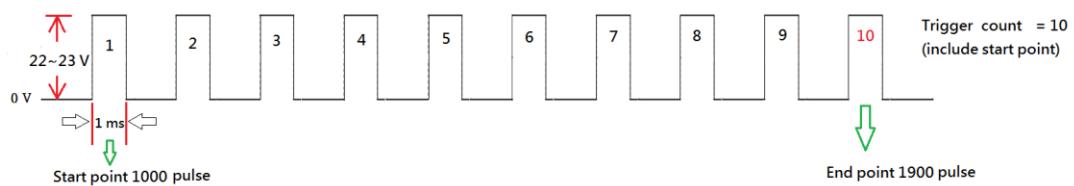
EMX-100 触发参数表

编号	定义	描述	值	默认值
0x00	TGR0_CMP_SRC	使用编码器或命令位置设置轴 0 比较源	0: 命令位置 1: 编码器	1
0x01	TGR1_CMP_SRC	使用编码器或命令位置设置轴 1 比较源	0: 命令位置 1: 编码器	1
0x02	TGR0_CMP_COND	设定轴 0 比较条件	0:大于&等于 (> =) 1:大于 (>) 2:等于 (=) 3:小于 (<)	2
0x03	TGR1_CMP_COND	设定轴 1 比较条件	0:大于&等于 (> =) 1:大于 (>) 2:等于 (=) 3:小于 (<)	2
0x04	TGR0_CMP_VALUE	设置轴 0 单个比较器值或通过比较触发模式的线性比较器起点	整数值	100
0x05	TGR1_CMP_VALUE	设置轴 1 单个比较器值或通过比较触发模式的线性比较器起点	整数值	100
0x06	TGR0_PULSE_WIDTH	设置轴 0 输出脉冲宽度	0 : 64 微秒 1 : 256 微秒， 2 : 1 微秒	0
0x07	TGR1_PULSE_WIDTH	设置轴 1 输出脉冲宽度	0 : 64 微秒 1 : 256 微秒， 2 : 1 微秒	0
0x08	TGR0_PULSE_LOGIC	设置轴 0 脉冲输出逻辑	0 : 低电平有效 1 : 高电平有效	0
0x09	TGR1_PULSE_LOGIC	设置轴 1 脉冲输出逻辑	0 : 低电平有效 1 : 高电平有效	0
0x0A	TGR0_CMP_EN	启用轴 0 比较触发功能； 当比较条件为 TRUE 时，通过 TRG1 引脚生成输出信号	0 : 禁用 1 : 启用	0

0x0B	TGR1_CMP_EN	启用轴 1 比较触发功能； 比较条件为 TRUE 时，通过 TRG2 引脚生成输出信号	0 : 禁用 1 : 启用	0
0x0C	TGR0_CMP_MODE	设置轴 0 比较触发模式	0 : 单比较 1 : 线性比较	0
0x0D	TGR1_CMP_MODE	设置轴 1 比较触发模式	0:单比较 1:线性比较	0
0x0E	TGR0_LCMP_INTER	设置轴 0 线性比较触发间隔 (脉冲)	整数值	0
0x0F	TGR1_LCMP_INTER	设置轴 1 线性比较触发间隔 (脉冲)	整数值	0
0x10	TGR0_LCMP_RETIME	设置轴 0 线性比较触发重复次数	整数值	0
0x11	TGR1_LCMP_RETIME	设置轴 1 线性比较触发重复次数	整数值	0

备注 1：这是一个简单的示例，如下所示。 实际触发计数包括起点。 触发次数=重复时间+ 1

Linear compare trigger ,for instance Source = 1 (command), Condition = 1 (large than)
Start point = 1000, Interval = 100 pulse, Active high,Pulse width = 1ms, Repeat time = 9



注 2：完成所有触发点后，用户必须重新配置并启用它才能起作用。

注 3：最大比较触发频率支持 500 Hz，如果用户配置最大速度为 4M pps，则间隔应大于 8000 (4000000/500) 脉冲

PCI-8254/58 / AMP-204/8C 触发参数表

编号	定义	描述	值	默认值
0x00	TGR_LCMP0_SRC	线性比较 0 (LCMP0) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9 : 禁用	9
0x01	TGR_LCMP1_SRC	线性比较 1 (LCMP0) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9 : 禁用	9
0x02	TGR_TCMP0_SRC	表比较 0 (TCMP0) 源	0~7 : 编码器计数器 0~7	9

			8 : 定时器 0 计数器 9 : 禁用	
0x0 3	TGR_TCMP1_SRC	表比较 1 (TCMP0) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9 : 禁用	9
0x0 4	TGR_TCMP0_DIR	表比较 0 (TCMP0) 方向	0 : 反方向 1 : 正方向 2 : 双向 (无方向)	0
0x0 5	TGR_TCMP1_DIR	表比较 1 (TCMP0) 方向	0 : 反方向 1 : 正方向 2 : 双向 (无方向)	0
0x0 6	TGR_TRG_EN	TRG 0~3 按位启用 注意 : 此参数也由板载参数 “ PWMx map DO” 和 “ VAO table” 函数控制。	Bit x: (0: 禁用, 1: 启用) Bit 0:TRG0 启用 Bit 1:TRG1 启用 Bit 2:TRG2 启用 Bit 3:TRG3 启用	0
0x1 0	TGR_TRG0_SRC	触发输出 0 (TRG0) 源 注意 : “或” 多源 , 然后输出到 TRG0)	Bit x:(1: 开, 0: 关) Bit 0:Manual0 Bit 1:预留 Bit 2:FCMP0 Bit 3:FCMP1 Bit 4:LCMP0 Bit 5:LCMP1 Bit 6:MCMP Bit 7:FCMP2 Bit 8:FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	0
0x1 1	TGR_TRG1_SRC	触发输出 1 (TRG1) 源 注意 : “或” 多源 , 然后输出到 TRG0)	Bit x:(1: 开, 0: 关) Bit 0:Manual0 Bit 1:预留 Bit 2:FCMP0 Bit 3:FCMP1 Bit 4:LCMP0 Bit 5:LCMP1 Bit 6:MCMP Bit 7:FCMP2 Bit 8:FCMP3	0

			Bit 9 LCMP2 Bit 10 LCMP3	
0x1 2	TGR_TRG2_SRC	触发输出 2 (TRG2) 源 注意： “或” 多源，然 后输出到 TRG0)	Bit x:(1: 开, 0: 关) Bit 0:Manual0 Bit 1:预留 Bit 2:FCMP0 Bit 3:FCMP1 Bit 4:LCMP0 Bit 5:LCMP1 Bit 6:MCMP Bit 7:FCMP2 Bit 8:FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	0
0x1 3	TGR_TRG3_SRC	触发输出 3 (TRG3) 源 注意： “或” 多源，然 后输出到 TRG0)	Bit x:(1: 开, 0: 关) Bit 0:Manual0 Bit 1:预留 Bit 2:FCMP0 Bit 3:FCMP1 Bit 4:LCMP0 Bit 5:LCMP1 Bit 6:MCMP Bit 7:FCMP2 Bit 8:FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	0
0x1 4	TGR_TRG0_PWD	TRG0 脉冲宽度	脉冲宽度= (N- 1)*20ns N = 2 ~ 0xffffffff	11
0x1 5	TGR_TRG1_PWD	TRG1 脉冲宽度	脉冲宽度= (N- 1)*20ns N = 2 ~ 0xffffffff	11
0x1 6	TGR_TRG2_PWD	TRG2 脉冲宽度	脉冲宽度= (N- 1)*20ns N = 2 ~ 0xffffffff	11
0x1 7	TGR_TRG3_PWD	TRG3 脉冲宽度	脉冲宽度= (N- 1)*20ns N = 2 ~ 0xffffffff	11
0x1 8	TGR_TRG0_LOGIC	TRG 0 逻辑	0: 不逆 1:逆	0

0x19	TGR_TRG1_LOGIC	TRG 1 逻辑	0: 不逆 1:逆	0
0xA	TGR_TRG2_LOGIC	TRG 2 逻辑	0: 不逆 1:逆	0
0xB	TGR_TRG3_LOGIC	TRG 3 逻辑	0: 不逆 1:逆	0
0xC	TGR_TRG0_TGL	TRG 0 toggle 模式	0 : 脉冲输出 1 : Toggle 输出	0
0xD	TGR_TRG1_TGL	TRG 1 toggle 模式	0 : 脉冲输出 1 : Toggle 输出	0
0xE	TGR_TRG2_TGL	TRG 2 toggle 模式	0 : 脉冲输出 1 : Toggle 输出	0
0xF	TGR_TRG3_TGL	TRG 3 toggle 模式	0 : 脉冲输出 1 : Toggle 输出	0
0x20	TIMR_ITV	计时器间隔	计时器间隔= N * 100 ns N = 1~ 53687091	1(100 ns)
0x21	TIMR_DIR	计时器方向	0 : 正方向数 1 : 反方向计数	0
0x22	TIMR_RING_EN	使定时器计数器成为环计数器	0: 禁用 (0x7fffffff+1→0x80000000) (0x80000001-1→0x80000000) 1: 启用 (0x7fffffff+1→0x00000000) (0x00000000-1→0x7fffffff)	0
0x23	TIMR_EN	启用定时器	0: 禁用, 1:启用	0
0x30	TGR_MCMP0_SRC	多轴比较器 0 (MCMP0) 源	0~7 : 编码器计数器 8 : 定时器 0 计数器 9: 禁用	9
0x31	TGR_MCMP1_SRC	多轴比较器 1 (MCMP1) 源	0~7 : 编码器计数器 8 : 定时器 0 计数器 9: 禁用	9

0x3 2	TGR_MCMP2_SRC	多轴比较器 2 (MCMP2) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9
0x3 3	TGR_MCMP3_SRC	多轴比较器 3 (MCMP3) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9
0x3 4	TGR_MCMP_MODE	决定多轴比较器何时可以触发 PWM 的模式	0 : 原始模式 : 当电机到达窗口定义的边界时 1 : 精确模式 : 当电动机处于窗口定义的边界内并且相对于比较点的位置偏差最短时	0
0x3 5	TGR_TRG0_TOGGLE_MODE	启用此模式可允许用户设置触发输出引脚的状态。 注意 : 此模式将影响其他 FPGA 模块 , 例如 “比较” 触发器。	0: 禁用 1: 启用	0
0x3 6	TGR_TRG1_TOGGLE_MODE	同上	0: 禁用 1: 启用	0
0x3 7	TGR_TRG2_TOGGLE_MODE	同上	0: 禁用 1: 启用	0
0x3 8	TGR_TRG3_TOGGLE_MODE	同上	0: 禁用 1: 启用	0
0x3 9	TGR_TRG0_TOGGLE_STATUS	写入和读取切换输出状态	0 : 输出低电平 1 : 输出高电平	0
0x3 A	TGR_TRG1_TOGGLE_STATUS	同上	0 : 输出低电平 1 : 输出高电平	0
0x3 B	TGR_TRG2_TOGGLE_STATUS	同上	0 : 输出低电平 1 : 输出高电平	0
0x3 C	TGR_TRG3_TOGGLE_STATUS	同上	0 : 输出低电平 1 : 输出高电平	0
0x4 0	TGR_TCMP2_SRC	表比较 2 (TCMP2) 源	0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9

0x41	TGR_TCMP3_SRC	表比较 3 (TCMP3) 源 0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9
0x42	TGR_TCMP2_DIR	表比较 2 (TCMP2) 方向 0 : 反方向 1 : 正方向 2 : 双向 (无方向)	0
0x43	TGR_TCMP3_DIR	表比较 3 (TCMP3) 方向 0 : 反方向 1 : 正方向 2 : 双向 (无方向)	0
0x44	TGR_LCMP2_SRC	线性比较 2 (LCMP2) 源 0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9
0x45	TGR_LCMP3_SRC	线性比较 3 (LCMP3) 源 0~7 : 编码器计数器 0~7 8 : 定时器 0 计数器 9: 禁用	9

K. 锁存参数表

PCI-C154(+) 锁存参数表				
编号	定义	描述	值	默认值:
0x00	LTC_ENC_IPT_模式	编码器脉冲输入模式 0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	0: OUT/DIR (EA = Out, EB = Dir) 1: CW/CCW (EA = CW, EB = CCW) 2: 1x AB相位 3: 2x AB相位 4: 4x AB相位	4
0x01	LTC_ENC_EA_INV	逆转EA编码器信号 1: 逆 0: 不逆	1: 逆 0: 不逆	0
0x02	LTC_ENC_EB_INV	逆转EB编码器信号 1: 逆 0: 不逆	1: 逆 0: 不逆	0
0x05	LTC_ENC_SIGNALFLT_EN	编码器信号低通滤波器 1: 启用	1: 启用	1

		(截止频率 (3db) : 10MHz)	0: 禁用	
0x06	LTC_FIFO_HIGH_LEVEL	锁存fifo高电平	0 ~ 255	0
0x07	LTC_SIGNAL_FLT_EN	锁存信号过滤器	1: 启用 0: 禁用	1
0x08	LTC_SIGNAL_TRIG_LOGIC	锁存信号触发逻辑	1: 上升有效 0: 下降有效	0

PCI-8254/58 / AMP-204/8C 锁存参数表				
编号	定义	描述	值	默认值:
0x10	LTC_IPT	锁存源	触发位置锁存的源： bit 0~7 : TTL0~TTL7 数字输入 信号; bit 8~11 : PWM 脉冲输出	0
0x11	LTC_ENC	锁存编码器	确定触发锁存源时哪个编码器被锁 存 ; 编码器编号范围 是 0~7	0
0x12	LTC_LOGIC	逻辑锁存	支持三种逻辑模式来触发锁存过程： 0 : 仅上升沿 1 : 只有下降沿 2 : 上升沿和下降沿	0

L. 设备信息表

PCI-8392 (H) 设备信息		
信息编号	信息含义	格式
0x00	预留	-
0x10	驱动版本	日期
0x20	CPLD版本	16 位
0x30	PCB版本	PCB
0x40	DSP版本	日期

PCI-8253/56 设备信息					
信息编号	信息	格式	信息编号	信息	
0x00	预留	--	0x01	预留	--
0x10	驱动版本	日期	0x11	预留	--
0x20	CPLD版本	16 位	0x21	FPGA版本	16 位
0x30	PCB版本 (载板)	PCB	0x31	PCB Ver.(DB)	PCB
0x40	DSP版本	日期	0x41	预留	--

PCI-8144 设备信息		
信息编号	信息含义	格式
0x00	预留	-
0x10	驱动版本	日期格式
0x20	CPLD版本	16 位

DPAC-1000设备信息		
信息编号	信息含义	格式
0x00	预留	-
0x10	驱动版本	日期
0x20	CPLD版本	16 位
0x30	PCB版本	PCB

DPAC-3000设备信息		
信息编号	信息含义	格式
0x00	预留	-
0x10	驱动版本	日期
0x20	CPLD版本	16 位
0x30	PCB版本	PCB

PCI(e)-7856设备信息		
信息编号	信息含义	格式
0x00	预留	-
0x10	驱动版本	日期格式
0x20	CPLD版本(PCI-7856) / FPGA版本(PCIe-7856)	CPLD : 16 位 / FPGA : 日期格式
0x30	PCB版本	PCB

MNET-4XMO设备信息					
信息编号	信息	格式	信息编	信息	
0x00	预留	--	0x01	预留	--
0x10	预留	--	0x11	预留	--
0x20	CPLD版本	16 位	0x21	预留	--
0x30	PCB版本 (按钮)	PCB	0x31	PCB Ver.(Top)	PCB

MNET-4XMO-C 设备信息					
信息编号	信息	格式	信息编	信息	

0x00	预留	--	0x01	预留	--
0x10	预留	--	0x11	预留	--
0x20	预留	--	0x21	FPGA版本	16 位
0x30	PCB版本 (按钮)	PCB	0x31	PCB Ver.(Top)	PCB

HSL-4XMO设备信息					
信息编号	信息	格式	信息编	信息	
0x00	预留	--	0x01	预留	--
0x10	预留	--	0x11	预留	--
0x20	CPLD版本	16 位	0x21	预留	--
0x30	预留	--	0x31	预留	--
0x40	DSP版本	日期 格式			

PCI(e)-8154/8158, PCI-8102 设备信息					
信息编号	信息	格式	信息编	信息	
0x00	预留	--	0x01	预留	--
0x10	驱动版本	日期	0x11	预留	--
0x20	CPLD 版本(载板)	16 位	0x21	FPGA/CPLD Ver.(DB)	16 位
0x30	PCB版本 (载板)	PCB	0x31	PCB Ver.(DB)	PCB

PCI-C154(+)设备信息					
信息编号	信息	格式	信息编	信息	
0x00	预留	--	0x01	预留	--
0x10	驱动版本	日期	0x11	预留	--
0x20	FPGA版本(载板)	16 位	0x21	预留	--
0x30	PCB版本 (载板)	PCB	0x31	预留	--

EMX-100 设备信息					
信息编号	信息	格式	信息编号	信息	格式
0x00	FMAC 软件版本	日期格式	0x01	FMAC 中间件 版本	日期格式
0x11	FMAC EXE 版本	日期格式	0x100	FMAC 库版本	日期格式

PCI-8254/58 / AMP-204/8C 设备信息					
信息编号	信息	格式	信息编号	信息	
0x00	预留	--	0x01	预留	--
0x10	驱动版本	日期	0x11	预留	--

0x20	预留	--	0x21	FPGA 版本	32 位
0x30	PCB 版本 (载板)	PCB	0x31	预留	--
0x40	DSP 版本	日期	0x41	预留	--

PCIe-833x 设备信息					
信息编号	信息	格式	信息编号	信息	
0x00	预留	--	0x01	预留	--
0x10	驱动版本	日期	0x11	预留	--
0x20	预留	--	0x21	FPGA 版本	32 位
0x30	产品和 PCB 版本(载板)	PRO,PCB	0x31	预留	--
0x40	Kernel 版本	日期	0x41	预留	--

格式描述:

- **日期格式:** 32位值

值= YYMMDD; Y:年, M:月, D:日

例如， 驱动版本 = 80212. 2008/2/12 发布

- **PCB 格式:** 4 位值。

00 00b = PCB A1 版

本00 00b



十进制	二进制	版本	十进制	二进制	版本
0	0000b	A1	8	1000b	C1
1	0001b	A2	9	1001b	C2
2	0010b	A3	10	1010b	C3
3	0011b	A4	11	1011b	C4
4	0100b	B1	12	1100b	D1
5	0101b	B2	13	1101b	D2
6	0110b	B3	14	1110b	D3
7	0111b	B4	15	1111b	D4

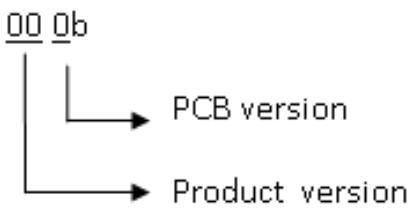
- **16 位格式:** 16位值 (1 ~ 255)

对于 PCIe-833x:

- **产品格式和PCB格式 (3位) :**

➢ PCB格式 : 位0值 , 0b : A1 , 1b : A2。

➢ 产品格式 : bit 1~2值 , 00b : PCIe-8338 , 01b : PCIe-8332 , 10b : PCIe-8334



十进制	二进制	版本
0	000b	PCIe-8338, A1
1	001b	PCIe-8338, A2
2	010b	PCIe-8332, A1
3	011b	PCIe-8332, A2
4	100b	PCIe-8334, A1
5	101b	PCIe-8334, A2
6		
7		

对于 PCI-8254/58 / AMP-204/8C 和 PCIe-833x:

- **32 位格式:: 32 位值**

M. 现场总线从站参数表

HSL-DI16-UL				
CH 编号	PA 编号	描述	值	默认值:
-1	0x0000	启用/禁用所有通道的拉伸(锁存)函数。 (仅设置)	0: 启用 1: 禁用	1
-1	0x0001	设置所有通道的拉伸(锁存)持续时间。(仅设置)	0 ~ 127 (ms) 0: 不拉伸.	0
0 ~ 15	0x0000	设置/获取每个通道的拉伸(锁存)函数。	0: 启用 1: 禁用	1
0 ~ 15	0x0001	设置/获取每个通道的拉伸(锁存)持续时间。	0 ~ 127 (ms) 0: 不拉伸.	0

HSL-AI16AO2				
CH 编号	PA 编号	描述	值	默认值:
-1	0x0000	设置/获取模拟输入范围。	0: +/- 10V 1: +/- 5V 2: +/- 2.5V	0

			3: +/- 1.25	
-1	0x0001	设置/获取上次扫描的模拟输入通道	0 ~ 15	15
-1	0x0002	启用/禁用模拟输入 (AD转换器) (仅设置)	0: 禁用 1: 启用	0

HSL-AO4				
CH 编号	PA 编号	描述	值	默认 值:
-1	0x0000	设置/获取保持模式。 (值格式 : 位格式) 保持启用表示通讯中断时将保留模拟输出。 设置为0 , 启用所有通道保持模式。 设置0xF , 禁用所有通道保持模式。	Bit ON:禁用 Bit OFF: 启用 Bit 0~3: Ch 0 ~ Ch3	0

N. DPAC 显示索引表

APS_get_display_data() 和 APS_set_display_data() 参考表。

对于字母类型，用户可以使用三个值之一来显示它。例如，对于字母“A”，用户可以设置 0x0A，0x41 或 0x61 来显示

7 段 LED 结果	*显示 索引 显示索 引	显示索引	显示索引
'0'	0x00	0X30(ASCII' 0')	
'1'	0x01	0X31(ASCII' 1')	
'2'	0x02	0X32(ASCII' 2')	
'3'	0x03	0X33(ASCII' 3')	
'4'	0x04	0X34(ASCII' 4')	
'5'	0x05	0X35(ASCII' 5')	
'6'	0x06	0X36(ASCII' 6')	
'7'	0x07	0X37(ASCII' 7')	
'8'	0x08	0X38(ASCII' 8')	
'9'	0x09	0X39(ASCII' 9')	
'A'	0x0A	0X41(ASCII' A)	0X61(ASCII' a')
'b'	0x0B	0X42(ASCII' B')	0X62(ASCII' b')
'C'	0x0C	0X43(ASCII' C')	0X63(ASCII' c')
'd'	0x0D	0X44(ASCII' D')	0X64(ASCII' d')
'E'	0x0E	0X45(ASCII' E')	0X65(ASCII' e')
'F'	0x0F	0X46(ASCII' F')	0X66(ASCII' f')
'G'	0x10	0X47(ASCII' G')	0X67(ASCII' g')
'H'	0x11	0X48(ASCII' H')	0X68(ASCII' h')
'i'	0x12	0X49(ASCII' I')	0X69(ASCII' i')
'j'	0x13	0X4A(ASCII' J')	0X6A(ASCII' j')
'K'	0x14	0X4B(ASCII' K')	0X6B(ASCII' k')
'L'	0x15	0X4C(ASCII' L')	0X6C(ASCII' l')
'M'	0x16	0X4D(ASCII' M')	0X6D(ASCII' m')
'n'	0x17	0X4E(ASCII' N')	0X6E(ASCII' n')
'o'	0x18	0X4F(ASCII' O')	0X6F(ASCII' o')
'p'	0x19	0X50(ASCII' P')	0X70(ASCII' p')
'q'	0x1A	0X51(ASCII' Q')	0X71(ASCII' q')
'r'	0x1B	0X52(ASCII' R')	0X72(ASCII' r')

'S'	0x1C	0X53(ASCII' S')	0X73(ASCII' s')
't'	0x1D	0X54(ASCII' T')	0X74(ASCII' t')
'U'	0x1E	0X55(ASCII' U')	0X75(ASCII' u')
'v'	0x1F	0X56(ASCII' V')	0X76(ASCII' v')
'W'	0x21	0X57(ASCII' W')	0X77(ASCII' w')
'X'	0x22	0X58(ASCII' X')	0X78(ASCII' x')
'Y'	0x23	0X59(ASCII' Y')	0X79(ASCII' y')
'Z'	0x24	0X5A(ASCII' Z')	0X7A(ASCII' z')
'0.'	0x25		
'1.'	0x26		
'2.'	0x27		
'3.'	0x28		
'4.'	0x29		
'5.'	0x2A		
'6.'	0x2B		
'7.'	0x2C		
'8.'	0X2D		
'9.'	0X2E		
' '	0X2F 0X20	0X20(ASCII' ')	

O. DPAC 按钮状态表

表中的“开”表示已按下。

示例步骤–检查 B3 的 ON / OFF

- 1) 读取按钮状态
- 2) 通过“不”按钮状态获取新的按钮状态
- 3) 通过“Bit # = (4-B #)”将B3映射到Bit #。 我们得到Bit1。
- 4) 使用Bit1 (0010b) 来“与”新按钮状态
- 5) 如果结果为零，则表示未按下B3。
- 6) 如果结果不为零，则表示已按下B3。

按钮状态	B1 (Bit3)	B2 (Bit2)	B3 (Bit1)	B4 (Bit0)
0x0F	OFF	OFF	OFF	OFF
0x0E	OFF	OFF	OFF	ON
0x0D	OFF	OFF	ON	OFF
0x0C	OFF	OFF	ON	ON
0x0B	OFF	ON	OFF	OFF

0x0A	OFF	ON	OFF	ON
0x09	OFF	ON	ON	OFF
0x08	OFF	ON	ON	ON
0x07	ON	OFF	OFF	OFF
0x06	ON	OFF	OFF	ON
0x05	ON	OFF	ON	OFF
0x04	ON	OFF	ON	ON
0x03	ON	ON	OFF	OFF
0x02	ON	ON	OFF	ON
0x01	ON	ON	ON	OFF
0x00	ON	ON	ON	ON

P. SSCNET 伺服监控器源表

监视源编号	内容	单位	备注 (字节)
0	位置反馈	脉冲	4
1	位置下降	脉冲	4
2	速度反馈	0.01 r/min	4
3	电流反馈 (转矩)	0.1%	2 字节
4	瞬时旋转一圈	脉冲	4 字节
5	旋转一圈即可获得原始位置	脉冲	4 字节
6	ZCT	脉冲	4 字节
7	位置编码器脉冲/转速计数器不固定。	rev	2 字节
8	原始位置编码器脉冲/转速计数器。	rev	2 字节
9	总线电压	V	2 字节
10	再生负载系数	%	2 字节
11	有效负荷率	%	2 字节
12	负载惯性矩与伺服电机惯性矩之比	倍数	2 字节
13	位置环增益	Rad/s	2 字节
14	警报/告警编号		
15	警报详细位		
16	参数编号		
17	警报状态 (AL10~AL1F)		
18	警报状态 (AL20~AL2F)		
19	警报状态 (AL30~AL3F)		
20	警报状态 (AL40~AL4F)		
21	警报状态 (AL50~AL5F)		
22	警报状态 (AL60~AL6F)		
23	警报状态 (AL70~AL7F)		
24	警报状态 (AL80~AL8F)		

25	警报状态 (AL90~AL9F)		
26	警报状态 (ALE0~ALEF)		

Q. VAO 参数表

PCI-8253/56 VAO 参数表				
编号	定义	描述	值	默认值:
0x00 + (2 * N)注意: N是表编 号, 范围 是 0 ~ 7. (*3)	VAO_TABLE_OU TPUT_TYPE	表输出类型 (*1)	0: 电压 1: PWM 模式 2: 固定宽度的PWM频率模式 3. 固定占空比的PWM频率模 式	1
0x01 + (2 * N)注意: N是表编 号, 范围 是 0 ~ 7. (*3)	VAO_TABLE_ INPUT _TYPE	表输入类型	0: 反馈速度 1: 命令速度	0
0x10 + N 注意: N是表编 号, 范围 是 0 ~ 7. (*3)	VAO_TABLE_ PWM_Config	根据输出类型 配置PWM。	a. 模式 0 – 不关心 b. 模式 1 -设定固定频率(1 ~ 25M Hz) c. 模式 2 -设定一个固定的 脉冲宽度(40 ~ 335544340 ns) d. 模式 3 -设置固定的占空 比 : N * 0.05 %. (N: 1 ~ 2000)	100
0x20 + N 注意:N是表编 号, 范围是 0 ~ 7. (*3)	VAO_TABLE_ SRC	指定VAO表的 轴ID。 (多轴 上的线速度) (*2)	Bit0: 轴0开 Bit1: 轴1开 Bit2: 轴2开 Bit3: 轴3开	0x01
0x30(*4)	预留	预留	预留	
0x40	VAO_DO_DELAY _TIME	在点表运行时 为Do输出指定 一个延迟时 间。	0: 无延迟时间. N: 延迟时间为N * DSP周 期。 对于PCI-8253 , DSP周 期为400 us。 对于PCI- 8256 , DSP周期为500 us。	0
0x50~	预留			

(* 1) : PCI-8253 不支持电压模式。

(* 2) : PCI-8253 支持 3 轴。 bit0 , bit1 和 bit2 可用。

(* 3) : Vao 支持 8 张表。每个表都有自己的参数设置。例如，用户可以使用 0x00 将表输出类型设置为 table0，使用 0x02 将输出类型设置为 table1。另一个示例，用户可以使用 0x20 为 table0 指定轴 ID，并使用 0x21 为 table1 指定轴 ID。

(* 4) : 因为支持多表设计，所以取消了用于设置输出通道的，名为 VAO_TABLE_TARGET (0x30) 的参数。借助新设计，用户可以通过 APS_start_vao() 设置输出通道。请参阅 APS_start_vao()。

PCI-8254/58 / AMP-204/8C VAO 参数表				
编号	定义	描述	值	默认值:
0x00 + (2 * N) 注意: N 是表编号， 范围是 0 ~ 7。 (*3)	VAO_TABLE_O UTPUT_TYPE	表输出类型(*1)	0: 电压 (预留) 1: PWM 模式 2: 固定宽度的 PWM 频率模式 3. 固定占空比的 PWM 频率模式	1
0x01 + N 注意: N 是表编号， 范围是 0 ~ 7。 (*1)	VAO_TABLE_INPUT_TYPE	表输入类型	0: 反馈速度 1: 命令速度	0
0x10 + N 注意: N 是表编号， 范围是 0 ~ 7。 (*1)	VAO_TABLE_PWM_Config	根据输出类型配置 PWM。	e. 模式 0 – 不关心 f. 模式 1 - 设定固定频率(1 ~ 25M Hz) g. 模式 2 - 设定一个固定的脉冲宽度(40 ~ 335544340 ns) h. 模式 3 – 设置固定的占空比 : N * 0.05 %. (N: 1 ~ 2000)	100
0x20 + N 注意: N 是表编号， 范围是 0 ~ 7。 (*1)	VAO_TABLE_SRC	指定 VAO 表的轴 ID。 (多轴上的线速度)	Bit0: 轴 0 开 Bit1: 轴 1 开 Bit2: 轴 2 开 Bit3: 轴 3 开	0x01
0x30	Reserved	预留	预留	

(* 1) : Vao 支持 8 张表。每个表都有自己的参数设置。例如，用户可以使用 0x00 将表输出类型设置为 table0，使用 0x02 将输出类型设置为 table1。另一个示例，用户可以使用 0x20 为 table0 指定轴 ID，并使用 0x21 为 table1 指定轴 ID。

35. APS 函数返回代码

下表提供了 APS 库中可能的返回值的列表。如果返回值为负值，则表示存在一些错误或警告。
我们提供了 C / C ++ 标准头文件 “ ErrorCodeDef.h ”，该文件定义了所有错误的返回值。

A. APS 错误代码表

代码	定义	错误描述和要检查的项目
0	ERR_NoError	成功，没有错误
-1	ERR_OsVersion	操作系统版本错误。 该函数不支持您使用的当前操作系统。
-2	ERR_OpenDriverFailed	打开驱动程序失败。 创建驱动程序接口失败。 检查设备驱动程序是否正确安装。 检查系统中是否正确安装了设备。
-3	ERR_InsufficientMemory	系统内存不足。 您的系统中没有足够的内存。
-4	ERR_DeviceNotInitial	设备或板卡未初始化。 检查卡号 设备已关闭 设备未初始化。
-5	ERR_NoDeviceFound	找不到装置 检查设备驱动程序是否正确安装。 检查系统中是否正确安装了设备。
-6	ERR_CardIdDuplicate	板卡号重复。 检查卡ID设置 (SW 跳转) 检查初始功能的参数是否正确。
-7	ERR_DeviceAlreadyIntialed	设备已经初始化。 1. 检查关闭板卡功能是否正常工作。
-8	ERR_InterruptNotEnable	中断事件未启用。 1. 启用硬件中断。 2. 检查中断因子设置是否正确。
-9	ERR_TimeOut	函数超时
-10	ERR_ParametersInvaild	参数的值不正确。 检查参数的设置范围。 将参数的设置值与用户手册进行比较。
-11	ERR_SetEEPROM	硬件内存写入错误。
-12	ERR_GetEEPROM	硬件内存读取错误。
-13	ERR_FunctionNotAvailable	该函数在当前阶段不可用。 设备不支持此函数。 系统处于错误状态。 1. 检查函数库。

		2.检查硬件连接（伺服驱动器连接） 3.重新初始化（重新引导）系统。
-14	ERR_FirmwareError	固件处理错误。 1.检查固件版本。
-15	ERR_CommandInProcess	上一条命令正在处理中。
-16	ERR_AxisIdDuplicate	轴ID重复。
-17	ERR_ModuleNotFound	找不到从站模块。
-18	ERR_InsufficientModuleNo	系统模块编号不足
-19	ERR_HandShakeFailed	与DSP握手不合时宜。
-20	ERR_FILE_Format	配置文件格式错误。（无法解析）
-21	ERR_ParametersReadOnly	函数参数为只读。
-22	ERR_DistantNotEnough	距离不足以完成运动
-1000	ERR_Win32Error	没有此类事件编号或WIN32_API错误， 请与凌华科技FAE工作人员联系。
-1001	ERR_NoENIFile	生成ADLINK_Config2.xml文件（ENI文件）失败。
-1002	ERR_TimeOut_SetVoltageEnable	在伺服开启过程中设置电压启用超时。
-1003	ERR_TimeOut_SetReadyToSwitch	设置就绪以便在伺服开启过程中切换超时。
-1004	ERR_TimeOut_SetShutdown	通过伺服开启过程设置关闭时间。
-1005	ERR_TimeOut_SetSwitchOn	设置伺服开启过程中的开启超时。
-1006	ERR_TimeOut_SetOperationEnable	通过伺服开启过程设置操作启用超时。
-1007	ERR_RegistryPath	系统注册表路径失败或没有注册表。
-1008	ERR_MasterNotOPState	主站未处于OP状态。
-1009	ERR_SlaveNotOPState	从站不处于OP状态。
-1010	ERR_SlaveTotalAxisNumber	扫描的EtherCAT从站轴数超过了PCIe-8334/8 可以支持的最大轴数。
-1011	ERR_MissESIFileOrMissENIPath	缺少ESI文件或ENI路径。
-1012	ERR_MissConfig_1_Xml	缺少Config_1 xml文件。
-1013	ERR_CopyConfig_1_Xml_fail	复制Config_1 xml文件失败。
-1014	ERR_MissConfig_2_Xml	缺少Config_2 xml文件。
-1015	ERR_CopyConfig_2_Xml_fail	复制Config_2 xml文件失败。

B. DSP 运动内核错误代码

代码	定义	错误描述和要检查的项目
-2001	MKERR_AXIS_INDEX	轴范围误差
-2002	MKERR_CHANNEL_INDEX	通道范围错误

- 2003	MKERR_PARA_UNDEFINE	参数编号未定义
- 2004	MKERR_PARA_FAULT	参数数据错误
- 2005	MKERR_STATE_UNAVAILABLE	此状态无法执行此操作。 (仅当轴处于空闲状态时才能设置命令位置)
- 2006	MKERR_CHECKSUM	校验和错误 (内部错误) , (校验码错误)
- 2007	MKERR_TIME_OUT	超时错误
- 2008	MKERR_MEM_TEST	内存测试错误
- 2009	MKERR_CTRL_CMD	未知命令或此状态不能接受此命令
- 2010	MKERR_AXES_DIMENSION	轴的维度无效。
- 2011	MKERR_MBUF_FULL	运动缓冲区已满
- 2012	MKERR_NO_AVAILABLE_SPG	轴处于混合状态时 , 此函数不能接受 (所有 spg 都在忙)
- 2013	MKERR_BLEND_PERCENT	转换参数 “percent” 超出范围。
- 2014	MKERR_TRANSITION_MODE	转换模式未定义或不支持。
- 2015	MKERR_COORD_TRANS_INDEX	转换矩阵列索引 > MAX_AXES (无效)
- 2016	MKERR_SINP_WIDTH	Soft-inp 宽度无效 (≥ 0)
- 2017	MKERR_SINP_STABLE_TIME	Soft-inp 稳定时间无效 (< 65535)
- 2018	MKERR_GANTRY_MASTER	龙门主轴无效
- 2019	MKERR_FCMP_SIZE	比较日期的大小超出范围。
- 2020	MKERR_VS_INVALID	开始速度无效 , ($vs \geq 0$)
- 2021	MKERR_VE_INVALID	终止速度无效 , ($ve \geq 0$)
-	MKERR_VM_INVALID	最大速度无效 , ($vm > 0$)

2022		
- 2023	MKERR_ACC_INVALID	加速无效。 (acc > 0)
- 2024	MKERR_DEC_INVALID	减速无效。 (dec > 0)
- 2025	MKERR_S_INVALID	S 无效。 (0 <= S <= 1)
- 2026	MKERR_SD_DEC_INVALID	SD Dec 无效。 (>0)
- 2027	MKERR_AXES_OVERLAPPING	轴数不能重叠。
- 2028	MKERR_BLEND_DISTANCE	转换参数 “ ResidueDistance” 不能 < 0.0。
- 2029	MKERR_ERROR_POS_LEVEL	错误位置检查级别必须> = 0.0
- 2030	MKERR_WAIT_MOVE	轴运动时不接受等待模式
- 2031	MKERR_AX_DISABLE	轴已禁用 (伺服关闭)
- 2032	MKERR_AX_ERROR	轴处于错误状态 (AX_ERR_STOPPING/AX_ERR_STOPPED) 。 您应该重置错误状态。
- 2033	MKERR_AX_MOVING	轴在运动中，命令无法接受 (无法覆盖) ，轴必须具有相同的尺寸，相同的轴
- 2034	MKERR_PRE_EVENT_DIST	事件发生前距离必须> = 0
- 2035	MKERR_POST_EVENT_DIST	事件发生后距离必须> = 0
- 2040	MKERR_ARC_PARA	该函数不能接受半圆弧或全圆弧参数
- 2041	MKERR_ARC_FINAL_R	finalR 不能为负值。
- 2042	MKERR_ARC_NORMAL	法线向量无效。 或弧参数无效。
- 2050	MKERR_GANTRY_DEV_PROTECT	龙门偏差保护值必须> = 0
- 2051	MKERR_INVAILD_IN_GANTRY	龙门模式下不允许使用此命令。
-	MKERR_INVAILD_GEAR_MASTER	未定义齿轮主站

2052		
- 2053	MKERR_ENGAGE_RATE	
- 2054	MKERR_GEAR_RATIO	
- 2055	MKERR_GEAR_ENABLE_MODE	
- 2056	MKERR_GEAR_LOOP	不能是齿轮环。
- 2057	MKERR_JOG_OFFSET	点动偏移量参数错误。 必须> 0
- 2062	MKERR_DI_GROUP	DI 组号错误。
- 2063	MKERR_DI_CH	DI 通道号错误。
- 2070	MKERR_FILTER_COEFFICIENT	滤波系数错误
- 2071	MKERR_FBK_VEL_COEFFICIENT	
- 2080	MKERR_PTBUFF_AXIS_NOT_IDLE	轴现在处于运动状态，无法开始点缓冲
- 2081	MKERR_PTBUFF_DIMENSION	无效的轴尺寸设置
- 2082	MKERR_PTBUFF_AXIS_IN_USE	轴编号已被另一个 PTB 使用
- 2083	MKERR_PTBUFF_NOT_ENABLE	PTBUFF 未启用
- 2084	MKERR_PTBUFF_FULL	PTBUFF 已满
- 2085	MKERR_PTBUFF_CURVE_TYPE	无效的运动曲线类型
- 2086	MKERR_PTBUFF_DIMENSION_MISS	马赫线运动尺寸和 PTBUFF 尺寸
- 2087	MKERR_PTBUFF_CURVE_DIMENSION	曲线类型及其尺寸不匹配
- 2088	MKERR_PTBUFF_ABNORMAL_STOP	轴异常停止，请检查轴停止代码。
- 2089	MKERR_PTBUFF_M_QUEUE_EXCEPTION	轴运动队列异常。

- 2090	MKERR_PTBUFF_AXIS_INDEX_INVALID	目标轴索引超过 PTBUFF 尺寸
- 2091	MKERR_PTBUFF_ID	无效的 PTBUFF ID。 检查输入参数。
- 2092	MKERR_PTBUFF_CTRL_COMMAND	无效的 PTBUF 控制命令。 检查输入参数。
- 2093	MKERR_PTBUFF_AXES_IN_MOTION	启动 PTBUFF 时，检测到的 PTBUFF 轴在运动。
- 2094	MKERR_PTBUFF_EXT_COMMANDS_NUM	PTBUFF 附加命令必须<= 7
- 2095	MKERR_PTBUFF_EXT_COMMAND_EMPTY	执行额外的命令，但命令队列为空。
- 2096	MKERR_PTBUFF_EXT_COMMAND_FULL	推送额外的命令，但命令队列已满。
- 2097	MKERR_PTBUFF_EXT_COMMNAD	无效的额外命令代码。
- 2098	MKERR_PTBUFF_NOT_STOPPED	点缓冲区运行时命令无效。
- 2099	MKERR_PTBUFF_DWELL_TIME	停留时间必须> = 0
- 2100	MKERR_HS_UNKNOWN_CMD	握手数据错误。命令未知。
- 2101	MKERR_HS_SIZE	握手数据错误。大小或尺寸无效。
- 2102	MKERR_HS_SUB_ERRORS	子函数有错误。请检查子返回码
- 2201	MKERR_DSP_SYSTEM_CONFIG	DSP 初始化错误。 (请致电凌华研发人员)
- 2208	MKERR_LOAD_CALIBRATION_DATA	负载校准数据失败。
- 2209	MKERR_DRAM_TEST_FAILED	DRAM 硬件测试失败。 (请致电凌华)
- 2210	MKERR_FPGA_VERSION	FPGA 版本已过时。 (请致电凌华)
- 2211	MKERR_PSC_TIME_OUT	
- 2212	MKERR_PLL_TIME_OUT	
-	MKERR_PSC_PARAM_ERR	

2213		
- 2300	MKERR_MOTION_LOOP_TIMING	运动循环的时间超出范围。
- 2401	MKERR_WRITE_ROM	
- 2402	MKERR_READ_ROM	
- 2403	MKERR_REF_5V	
- 2404	MKERR_SET_AI_OFFSET	
- 2405	MKERR_SET_AI_GAIN	
- 2406	MKERR_SET_AO_OFFSET	
- 2407	MKERR_SET_AO_GAIN	
- 2408	MKERR_NO_CALIB	EEPROM 中没有校准数据。 (并非 AO/AI 的所有增益/偏移都已校准)
- 2409	MKERR_DATA_SIZE	
- 2410	MKERR_BACKDOOR_PWD	后门密码错误。
- 2411	MKERR_ROM_PROG_SIZE	(闪存) 数据字节数过多或偏移量超出范围。
- 2500	MKERR_OVER_QUEUE_SIZE	队列大小小于输入数组
- 2600	MKERR_FRP_STATE	频率响应过程的状态无效
- 2601	MKERR_RCP_STATE	继电器控制过程的状态无效
- 2700	MKERR_TASK_NUM	任务编号错误
- 2701	MKERR_UNKNOWN_PROGRAM_REGISTER	未知程序寄存器。
- 2702	MKERR_CANNOT_SET_REG _WHEN_PG_NOT_STOP	程序正在运行，您不能发出此命令
- 2703	MKERR_OVER_STACK_SIZE	超出堆栈大小范围

- 2800	MKERR_ACCESS_UNDEFINE	访问类型未定义
- 2801	MKERR_ACCESS_DENIED	由于参数访问处于保存/加载过程中，因此被拒绝
- 2802	MKERR_ERASE_SECTION	由于超时，擦除部分失败
- 2803	MKERR_LAST_MARK_INVALID	加载数据错误；；闪存中的最后一个标记是错误的
- 2804	MKERR_CHK_PARITY	加载数据错误； 奇偶校验位错误
- 2805	MKERR_OVER_DATA_TYPE	加载数据错误； 超过最大数据类型定义
- 2806	MKERR_OVER_PA_TYPE	加载数据错误； 超过最大参数类型定义
- 2807	MKERR_OVER_MAX_BOARD	加载数据错误； 超过最大板参数定义
- 2808	MKERR_OVER_MAX_AXIS	加载数据错误； 超过最大轴参数定义
- 2900	MKERR_WDT1_STATE	看门狗定时器 1 已启用
- 2901	MKERR_WDT1_PERIOD	看门狗定时器 1 的最大复位周期超出范围

C. EtherCAT 主站错误代码

代码	定义	错误描述和要检查的项目
-4001	EC_INIT_MASTER_ERR	初始化的 EtherCAT 主站错误
-4011	EC_GET_SLV_NUM_ERR	获取总从站编号错误
-4012	EC_CONFIG_MASTER_ERR	配置 EtherCAT 主站错误
-4013	EC_BUSCONFIG_MISMATCH	拓扑信息与当前数据不匹配
-4014	EC_CONFIGDATA_READ_ERR	读取配置数据错误
-4015	EC_ENI_NO_SAFEOP_OP_SUPPORT	不支持 safeop 和 op 状态
-4021	EC_CONFIG_DC_ERR	配置直流参数错误
-4022	EC_DCM_MODE_NO_SUPPORT	不支持 dcm 模式
-4023	EC_CONFIG_DCM_FEATURE_DISABLED	dcm 功能已禁用
-4024	EC_CONFIG_DCM_ERR	配置 DCM 参数错误
-4031	EC_REG_CLIENT_ERR	注册客户端错误
-4041	EC_SET_INIT_STATE_ERR	将 EtherCAT 主站设置为初始状态错误
-4042	EC_SET_PREOP_STATE_ERR	将 EtherCAT 主站设置为操作前状态错误
-4043	EC_SET_SAFEOP_STATE_ERR	将 EtherCAT 主站设置为 safeop 状态错误
-4044	EC_SET_OP_STATE_ERR	将 EtherCAT 主站设置为操作状态错误
-4051	EC_DE_INIT_MASTER_ERR	初始化的 EtherCAT 主站错误
-4061	EC_ENI_FOPEN_ERR	无法打开 ENI 信息
-4062	EC_ENI_FREAD_ERR	无法读取 ENI 信息
-4063	EC_GEN_EBI_BUSSCAN_ERR	扫描 EtherCAT 总线错误
-4081	EC_WRONG_PORT_NO	输入错误的 EtherCAT 主站端口号
-4091	EC_GET_SLAVE_INFO_ERR	获取从站信息错误
-4101	EC_COE_SDO_UPLOAD_ERR	执行 sdo 上传输入数据错误
-4102	EC_COE_SDO_HOME_MODE_ERR	输入 ECAT 伺服归零模式错误或数据错误。
-4103	EC_COE_SDO_HOME_ACCDEC_ERR	输入 ECAT 伺服归零 ACC/DEC 错误或数据错误。
-4104	EC_COE_SDO_HOME_VM_SWITCH_ERR	输入 ECAT 伺服归零限位开关错误或数据错误。
-4105	EC_COE_SDO_HOME_VM_ZERO_ERR	输入 ECAT 伺服归零错误或错误数据。
-4106	EC_COE_SDO_HOME_OFFSET_ERR	输入 ECAT 伺服归零偏移错误或数据错误。
-4107	EC_CONTROL_WORD_HOME_ERR	输入 ECAT 伺服归零控制字错误或数据错误。
-4108	EC_COE_SDO_STOP_ERR	输入 ECAT 伺服归零停止错误或数据错误。
-4109	EC_CONTROL_WORD_STOP_ERR	输入 ECAT 伺服归零限位开关错误或数据错误。
-4110	EC_SET_OP_MODE_HOME_ERR	使用 ECAT 主进程设置 OP 模式错误。
-4201	EC_WRONG_SLAVE_NO	输入从站号超出限制

-4202	EC_WRONG_MODULE_NO	输入模块数量超出限制
-4203	EC_WRONG_AI_CHANNEL_NO	输入的 AI 通道数超出限制
-4204	EC_WRONG_AO_CHANNEL_NO	输入的 AO 通道数超出限制
-4205	EC_COE_SDO_DOWNLOAD_ERR	执行 sdo 下载输入数据错误
-4301	EC_COE_OD_INIT_ERR	初始化对象字典时创建内存错误
-4302	EC_COE_GET_OD_NUM_ERR	获取对象字典编号错误
-4303	EC_COE_GET_OD_NUM_LAST	输入对象字典号是最后一个
-4304	EC_COE_GET_OD_DESC_ERR	获取对象字典描述错误
-4305	EC_COE_GET_OD_DESC_ENTRY_ERR	获取对象字典描述输入错误
-4306	EC_COE_GET_OD_STATUS_PEND	获取对象字典待处理
-4503	EC_DUPLICATE_SLAVE_ID_ERR	从站 ID 号重复出现
-4504	EC_GET_SLAVE_REGISTER_ERR	获取从属寄存器错误
-4505	EC_SET_SLAVE_REGISTER_ERR	设置从寄存器错误