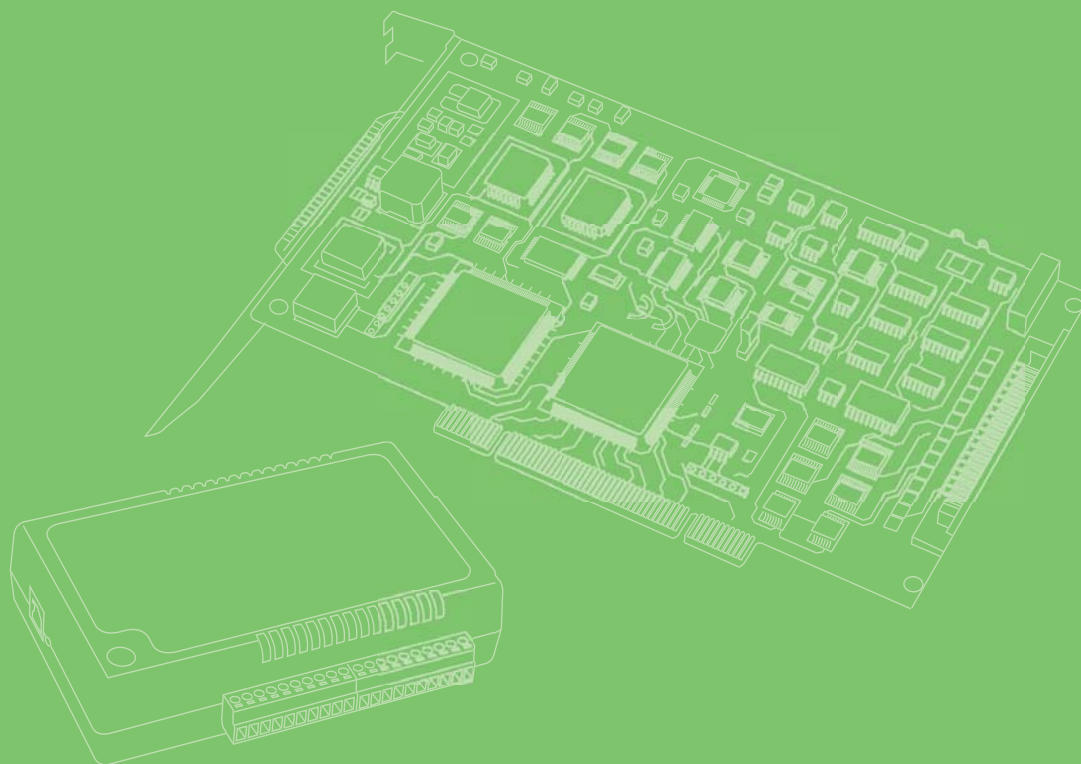


用户手册



PCI-1203

EtherCAT Master PCI 卡

ADVANTECH

Enabling an Intelligent Planet

版权声明

随附本产品发行的文件为研华公司 2017 年版权所有，并保留相关权利。针对本手册中相关产品的说明，研华公司保留随时变更的权利，恕不另行通知。未经研华公司书面许可，本手册所有内容不得通过任何途径以任何形式复制、翻印、翻译或者传输。本手册以提供正确、可靠的信息为出发点。但是研华公司对于本手册的使用结果，或者因使用本手册而导致其它第三方的权益受损，概不负责。

认可声明

IBM 和 PC 为 International Business Machines Corporation 的商标。
所有其它产品名或商标均为各自所属方的财产。

产品质量保证（两年）

从购买之日起，研华为原购买商提供两年的产品质量保证。但对那些未经授权的维修人员维修过的产品不予提供质量保证。研华对于不正确的使用、灾难、错误安装产生的问题有免责权利。

如果研华产品出现故障，在质保期内我们提供免费维修或更换服务。对于出保产品，我们将会酌情收取材料费、人工服务费用。请联系相关销售人员了解详细情况。

如果您认为您购买的产品出现了故障，请遵循以下步骤：

1. 收集您所遇到的问题信息（例如，CPU 主频、使用的研华产品及其它软件、硬件等）。请注意屏幕上出现的任何不正常信息显示。
2. 打电话给您的供货商，描述故障问题。请借助手册、产品和任何有帮助的信息。
3. 如果您的产品被诊断发生故障，请从您的供货商那里获得 RMA（Return Material Authorization）序列号。这可以让我们尽快地进行故障产品的回收。
4. 请仔细地包装故障产品，并在包装中附上完整的售后服务卡片和购买日期证明（如销售发票）。我们对无法提供购买日期证明的产品不提供质量保证服务。
5. 把相关的 RMA 序列号写在外包装上，并将其运送给销售人员。

符合性声明

CE

本设备已通过 CE 测试，符合以屏蔽电缆进行外部接线的环境规格标准。建议用户使用屏蔽电缆，此种电缆可从研华公司购买。如需订购，请与当地分销商联系。

FCC A 级

注意：根据 FCC 规则第 15 款，本设备已经过检测并被判定符合 A 级数字设备标准。这些限制旨在为商业环境下的系统操作提供合理保护，使其免受有害干扰。本设备会产生、耗费和发射无线电频率能量，如果没有按照手册说明正确安装和使用，可能会对无线电通讯造成有害干扰。此时，用户需自行解决干扰问题。

FM

本设备已经通过 FM 认证。根据危险环境的不同，美国消防协会将危险区域分为不同的级别、类和组。本设备适用于第 1 级、第 2 类、第 A、B、C、D 组的室内危险环境中。

技术支持与服务

1. 有关该产品的最新信息，请访问研华公司的网站：
<http://support.advantech.com.cn>
2. 用户若需技术支持，请与当地分销商、销售代表或研华客服中心联系。进行技术咨询前，用户须将下面各项产品信息收集完整：
 - 产品名称及序列号
 - 外围附加设备的描述
 - 用户软件的描述（操作系统、版本、应用软件等）
 - 产品所出现问题的完整描述
 - 每条错误信息的完整内容

安全措施 – 静电防护

为了保护您和您的设备免受伤害或损坏，请遵照以下安全措施：

- 操作设备之前，请务必断开机箱电源，以防触电。不可在电源接通时接触 CPU 卡或其它卡上的任何元件。
- 在更改任何配置之前请断开电源，以免在您连接跳线或安装卡时，瞬间电涌损坏敏感电子元件。

安全指示

1. 请仔细阅读此安全操作说明。
2. 请妥善保存此用户手册供日后参考。
3. 用湿抹布清洗设备前，请从插座拔下电源线。请不要使用液体或去污喷雾剂清洗设备。
4. 对于使用电源线的设备，设备周围必须有容易接触到的电源插座。
5. 请不要在潮湿环境中使用设备。
6. 请在安装前确保设备放置在可靠的平面上，意外跌落可能会导致设备损坏。
7. 设备外壳的开口是用于空气对流，从而防止设备过热。**请不要覆盖这些开口。**
8. 当您连接设备到电源插座上前，请确认电源插座的电压是否符合要求。
9. 请将电源线布置在人们不易绊到的位置，并不要在电源线上覆盖任何杂物。
10. 请注意设备上的所有警告标识。
11. 如果长时间不使用设备，请将其同电源插座断开，避免设备被超标的电压波动损坏。
12. 请不要让任何液体流入通风口，以免引起火灾或者短路。
13. 请不要自行打开设备。为了确保您的安全，请由经过认证的工程师来打开设备。
14. 如遇下列情况，请由专业人员来维修：
 - 电源线或者插头损坏；
 - 设备内部有液体流入；
 - 设备曾暴露在过于潮湿的环境中使用；
 - 设备无法正常工作，或您无法通过用户手册来使其正常工作；
 - 设备跌落或者损坏；
 - 设备有明显的外观破损。
15. 请不要把设备放置在超出我们建议的温度范围的环境，即不要低于 -20°C (-4°F) 或高于 60°C (140°F)，否则可能会损坏设备。
16. **注意：**计算机配置了由电池供电的实时时钟电路，如果电池放置不正确，将有爆炸的危险。因此，只可以使用制造商推荐的同一种或者同等型号的电池进行替换。请按照制造商的指示处理旧电池。

根据 IEC 704-1: 1982 的规定，操作员所在位置的声压级不可高于 70dB(A)。

免责声明：该安全指示符合 IEC 704-1 的要求。研华公司对其内容的准确性不承担任何法律责任。

目录

第 1 章	概述	1
1.1	EtherCAT 简介	2
1.1.1	EtherCAT	2
	图 1.1: EtherCAT Function Principle	2
	图 1.2: EtherCAT 协议	3
	图 1.3: EtherCAT 拓扑结构	3
	图 1.4: EtherCAT 分布式时钟	4
	图 1.5: EtherCAT 分布式时钟抖动	4
1.2	PCI-1203 特性	5
1.2.1	多主站模式	5
1.2.2	多轴插补	5
1.2.3	强大的 I/O 数据处理功能	5
1.2.4	板载实时 OS 支持	5
1.2.5	分布式时钟 (DC)	5
1.2.6	支持 Common Motion SDK	5
1.2.7	错误检测与诊断	6
1.3	产品规格	6
1.3.1	EtherCAT	6
1.3.2	隔离数字量输入	6
1.3.3	隔离数字量输出	6
1.3.4	General	7
第 2 章	安装	9
2.1	打开包装	10
2.2	安装驱动	10
2.3	安装硬件	10
第 3 章	硬件服务	13
3.1	外围	14
	表 3.1: 接口及开关	14
3.2	板卡 ID 开关 (SW1)	14
	表 3.2: 板卡 ID 设置	14
3.3	LAN 端口 (CN20, CN21)	15
3.4	D-sub 接口 (CN26)	15
	表 3.3: D-Sub 针脚定义	15
3.5	信号连接	15
3.5.1	数字量输入布线参考	15
3.5.2	数字量输出布线参考	16
第 4 章	EtherCAT 实用程序	17
4.1	简介	18
4.2	主界面	18
4.2.1	主菜单	18
4.2.2	工具栏	19
4.3	主站页面	22
4.3.1	单轴运动控制 (Single-Axis Motion)	23
4.3.2	多轴运动 (Multi-Axis Motion)	29
4.3.3	同步运动 (Synchronized Motion)	36
4.3.4	DO / DI / AO / AI	42

4.3.5	I/O 映射表.....	44
4.3.6	信息	48
4.4	从站页面	49
4.4.1	单轴运动控制 (Single-Axis Motion).....	49
4.4.2	数字量输入 (Digital Input) 与数字量输出 (Digital Output)	49
4.4.3	运动 I/O (Motion I/O).....	50
4.4.4	I/O 列表.....	51
4.4.5	从站信息	52
4.4.6	ADAM-E5000 模块页面.....	53

第 5 章 编程指南 63

5.1	简介	64
5.2	关于研华运动控制卡接口函数库	64
	图 5.1: 不同设备对接口函数的调用	64
5.3	运动控制卡接口函数库所需头文件	65
	图 5.2: 动态链接库所需头文件	65
5.3.1	接口函数库 --- 功能函数	65
5.3.2	接口函数库 --- 功能属性	66
	表 5.1: 属性的命名规则	66
	表 5.2: 设置 / 获取属性函数列表	67
	图 5.3: 属性数据类型与函数的关系	67
5.3.3	接口函数库 --- 设备类型	68
	表 5.3: 设备类型定义	68
5.3.4	接口函数库 --- 错误代码	68
5.3.5	接口函数库 --- 数据类型再定义	68
	表 5.4: 数据类型再定义	68
5.3.6	板卡的初始操作	69
	图 5.4: 板卡初始化流程图	70
5.4	运动控制卡接口函数库在编程工具中的使用	71
	图 5.5: 属性数据类型与函数的关系	72
5.4.1	接口函数库在 Visual C++ 6.0 中的使用	72
5.4.2	接口函数库在 Visual Basic 6.0 中的使用	73
5.4.3	接口函数库在 Visual Studio 2005 中的使用	73
5.4.4	接口函数库在 Borland C++ Builder 中的使用	74
5.5	关于在 Win7 下使用 Common Motion API 的注意事项	75
5.5.1	关于提升应用程序的权限	75
5.5.2	Win7 下运行范例	76
5.6	多块板卡编程说明	77
5.6.1	多块板卡的独立性	77
5.6.2	如何操作多快板卡	77

第 6 章 通用运动 API 架构 79

6.1	简介	80
6.2	通用运动架构	80
	图 6.1: 运动控制卡通用运动架构	80
6.3	物件定义	81
	表 6.1: 缩写及其含义	81
6.4	设备编号	81

第 7 章 属性配置文件说明 83

7.1	简介	84
7.2	系统配置基本概念	84
7.2.1	硬件资源配置	84
7.2.2	软件资源配置	88

7.2.3	资源整合.....	90
7.3	配置文件生成与下载.....	90
7.3.1	配置文件下载函数.....	90
7.3.2	配置文件重点说明.....	90
7.4	配置信息修改.....	91
7.5	控制器配置初始化状态.....	91
7.5.1	硬件资源配置初始化状态、.....	91
7.5.2	软件资源配置初始化状态.....	93
7.6	伺服操作.....	94
7.7	脉冲输入输出方式.....	94
7.7.1	脉冲输入输出方式相关属性.....	94
7.8	浮點 PPU.....	95
7.8.1	浮点 PPU 相关属性.....	95
7.8.2	浮点 PPU 功能介绍.....	95
7.8.3	浮点 PPU 重点说明.....	95
7.8.4	例程.....	96

第 8 章 运动状态 97

8.1	简介.....	98
8.2	轴状态.....	98
8.2.1	轴状态相关函数.....	98
8.2.2	重点说明.....	98
	表 8.1: 轴当前状态定义.....	98
	表 8.2: 轴当前运动状态定义.....	99
8.2.3	例程.....	99
8.3	轴速度.....	100
8.3.1	轴速度相关函数与属性.....	100
8.3.2	轴速度重点说明.....	100
8.3.3	例程.....	101
8.4	轴的运动 I/O 状态.....	102
8.4.1	轴的运动 I/O 状态函数.....	102
8.4.2	轴的运动 I/O 状态重点说明.....	102
8.4.3	例程.....	103
8.5	群组状态.....	103
8.5.1	群组状态相关函数.....	103
8.5.2	重点说明.....	103
	表 8.3: 群组当前状态定义.....	103
8.5.3	例程.....	104
8.6	群组速度.....	104
8.6.1	群组速度相关函数与属性.....	104
8.6.2	群组速度重点说明.....	105
8.6.3	例程.....	105

第 9 章 运动功能 107

9.1	简介.....	108
9.2	单轴运动模式.....	108
9.2.1	本节简介.....	108
9.2.2	点到点运动.....	108
	表 9.1: 点到点运动相关函数.....	108
	表 9.2: 点到点运动相关属性.....	109
	图 9.1: 点到点运动流程图.....	110
9.2.3	连续运动.....	111
	表 9.3: 连续运动相关函数.....	111
	表 9.4: 连续运动相关属性.....	112
	图 9.2: 连续运动流程图.....	112
9.2.4	变位运动.....	114
	表 9.5: 变位运动相关函数.....	114

	表 9.6:	变位运动相关属性	114
	图 9.3:	变位运动流程图	116
9.2.5	变速运动		117
	表 9.7:	变速运行相关函数	117
	表 9.8:	变速运行相关属性	118
	图 9.4:	变速运动流程图	119
9.2.6	同步起、同步停运动		120
	表 9.9:	同步起、同步停运动相关函数	120
	表 9.10:	同步起、同步停运动相关属性	121
	图 9.5:	同步起、同步停运动流程图	122
9.2.7	叠加运动		123
	表 9.11:	叠加运动相关函数	123
	表 9.12:	叠加运动相关属性	124
	图 9.6:	叠加运动	124
	图 9.7:	叠加运动流程图	125
9.2.8	JOG 运动		126
	表 9.13:	JOG 运动相关函数	126
	表 9.14:	JOG 运动相关属性	127
	图 9.8:	JOG 运动流程图	129
9.2.9	原点复归		131
	表 9.15:	原点复归相关函数	131
	表 9.16:	原点复归相关属性	131
	图 9.9:	原点复归流程图	145
9.2.10	轴的事件		147
	表 9.17:	轴的事件相关函数	147
	表 9.18:	轴的事件相关属性	148
	图 9.10:	轴的事件流程图	150
9.2.11	PT 运动		152
	表 9.19:	PT 运动函数	152
	图 9.11:	PT 运动流程图	153
9.2.12	PVT 运动		155
	表 9.20:	PVT 运动函数	155
	图 9.12:	PVT 运动流程图	156
9.3	插补运动模式		157
9.3.1	本节简介		157
9.3.2	直线插补运动		157
	表 9.21:	直线插补运动相关函数	157
	表 9.22:	直线插补运动相关属性	158
	图 9.13:	直线插补运动流程图	160
9.3.3	圆弧插补		162
	表 9.23:	圆弧插补相关函数	162
	表 9.24:	圆弧插补运动相关函数	163
	图 9.14:	圆弧插补运动流程图	167
9.3.4	螺旋插补运动		169
	表 9.25:	螺旋插补运动相关函数	169
	表 9.26:	螺旋插补运动相关函数	170
	图 9.15:	螺旋插补运动流程图	172
9.3.5	群组的事件		174
	表 9.27:	轴的事件相关函数	174
	图 9.16:	群组事件流程图	176
9.4	跟随运动模式		178
9.4.1	本节简介		178
9.4.2	电子齿轮运动		178
	表 9.28:	电子齿轮运动相关函数	178
	表 9.29:	电子齿轮运动相关属性	179
	图 9.17:	电子齿轮流程图	180
9.4.3	电子凸轮运动		182
	表 9.30:	电子凸轮运动相关函数	182
	表 9.31:	电子凸轮运动相关属性	183
9.4.4	龙门运动		187

	表 9.32: 龙门运动相关函数.....	187
	表 9.33: 龙门运动相关属性.....	188
	图 9.18: 龙门运动流程图.....	189
9.4.5	切向跟随运动.....	191
	表 9.34: 切向跟随运动相关函数.....	191
	表 9.35: 切向跟随运动相关属性.....	191
	图 9.19: 切向跟随运动流程图.....	192
9.5	路径规划模式.....	194
9.5.1	本节简介.....	194
9.5.2	路径规划.....	194
	表 9.36: Path 运动相关函数.....	194
	表 9.37: Path 运动相关属性.....	194
	表 9.38: Path 运动模式.....	195
	表 9.39: Path 运动相关指令.....	195
9.5.3	EndPath 的作用及影响.....	199
9.5.4	速度交接模式.....	199
	表 9.40: 速度交接模式关系表.....	199
9.5.5	Path DO 功能指令.....	203
9.5.6	Path 的执行.....	203
9.5.7	Path 的暂停和恢复.....	204
9.5.8	动态加载 Path.....	207
9.5.9	Path 运动流程图.....	208
	图 9.20: Path 运动流程图.....	208
9.5.10	Path 运动例程.....	208

第 10 章 DI/O 与 AI/O 控制..... 211

10.1	简介.....	212
10.2	DI/O 与 AI/O 控制.....	212
	10.2.1 本节简介.....	212
	10.2.2 IO 控制相关函数与属性.....	212
	表 10.1: IO 控制函数.....	212
	表 10.2: IO 控制属性.....	213
	10.2.3 I/O 映射.....	214
	10.2.4 流程图.....	214
	10.2.5 事件.....	215
	10.2.6 例程.....	215
10.3	比较功能.....	218
	10.3.1 本节简介.....	218
	10.3.2 比较功能相关函数与属性.....	218
	表 10.3: 比较功能函数.....	218
	表 10.4: 比较功能属性.....	218
	10.3.3 重点说明.....	218
	10.3.4 例程.....	218
10.4	多轴比较功能.....	219
	10.4.1 本节简介.....	219
	10.4.2 多轴比较功能相关函数与属性.....	219
	表 10.5: 多轴比较功能函数.....	219
	表 10.6: 多轴比较功能属性.....	220
	10.4.3 重点说明.....	220
	10.4.4 多轴比较流程图.....	223
	图 10.1: 多轴比较流程图.....	223
10.5	锁存功能.....	223
	10.5.1 本节简介.....	223
	10.5.2 锁存单笔数据.....	223
	表 10.7: 锁存功能函数.....	224
	表 10.8: 锁存功能属性.....	224
	10.5.3 连续锁存功能.....	224
	表 10.9: 连续锁存函数.....	225

表 10.10: 连续锁存属性	225
10.5.4 例程	226

Appendix A API 列表 229

A.1	通用	230
A.2	设备对象	231
A.3	DAQ	248
A.3.1	数字量输入 / 输出	248
A.3.2	模拟量输入 / 输出	253
A.3.3	计数器	257
A.4	轴	259
A.4.1	系统	259
A.4.2	运动 IO	260
A.4.3	运动状态	261
A.4.4	速度运动	262
A.4.5	点到点运动	265
A.4.6	同步起停运动	266
A.4.7	回原点	268
A.4.8	位置 / 计数器控制	269
A.4.9	比较	270
A.4.10	锁存	272
A.4.11	连续锁存	273
A.4.12	外部驱动	274
A.4.13	凸轮 / 齿轮	275
A.4.14	龙门 / 切线跟随	276
A.4.15	停止运动	278
A.4.16	PT/PVT 运动	279
A.5	群组	281
A.5.1	系统	281
A.5.2	运动状态及速度	282
A.5.3	运动停止	283
A.5.4	插补运动	283
A.5.5	路径	295
A.5.6	暂停和恢复	298

Appendix B 属性列表 299

B.1	设备特性属性	300
B.2	设备配置属性	304
B.3	DAQ 特性属性	308
B.4	DAQ 配置属性	310
B.5	DAQ 通道配置属性	311
B.6	轴的特性属性	317
B.6.1	系统	317
B.6.2	速度模式	318
B.6.3	脉冲输入	318
B.6.4	脉冲输出	319
B.6.5	报警	320
B.6.6	到位	321
B.6.7	报警清除 (ERC)	321
B.6.8	硬件限位	321
B.6.9	软件限位	322
B.6.10	原点	322
B.6.11	背隙补偿	323
B.6.12	比较	324
B.6.13	锁存	324
B.6.14	Cam DO	324
B.6.15	外部驱动	325

	B. 6. 16 Jog.....	325
	B. 6. 17 Aux/Gen 输出.....	325
	B. 6. 18 同步启停.....	326
	B. 6. 19 触发停止.....	327
B. 7	轴的配置属性.....	328
	B. 7. 1 系统.....	328
	B. 7. 2 速度模式.....	328
	B. 7. 3 脉冲输入.....	329
	B. 7. 4 脉冲输出.....	330
	B. 7. 5 报警.....	330
	B. 7. 6 伺服到位.....	331
	B. 7. 7 硬件限位.....	332
	B. 7. 8 软件限位.....	332
	B. 7. 9 原点.....	334
	B. 7. 10 背隙补偿.....	334
	B. 7. 11 比较.....	335
	B. 7. 12 多轴比较.....	337
	B. 7. 13 锁存.....	337
	B. 7. 14 连续锁存.....	337
	B. 7. 15 外部驱动.....	339
	B. 7. 16 凸轮区间 DO	340
	B. 7. 17 轴单圈脉冲数.....	341
	B. 7. 18 同步启用.....	342
	B. 7. 19 龙门.....	342
	B. 7. 20 触发停止.....	343
B. 8	轴的参数属性.....	344
	B. 8. 1 速度参数.....	344
	B. 8. 2 原点.....	345
B. 9	群组的配置属性.....	347
	B. 9. 1 系统.....	347
	B. 9. 2 路径.....	347
B. 10	群组的参数属性.....	348
	B. 10. 1 速度模式.....	348
	B. 10. 2 系统.....	349
	B. 10. 3 路径.....	349

Appendix C 错误代码 351

C. 1	错误代码.....	352
C. 2	常见错误.....	352
C. 3	DSP 常见错误.....	363
C. 4	EtherCAT 常见错误.....	376
C. 5	不常见错误.....	379

第 1 章

概述

1.1 EtherCAT 简介

EtherCAT (Ethernet Control Automation Technology) 是一款基于以太网的高性能工业网络现场总线系统。该协议是 IEC 61158 标准的组成部分，专用于自动化应用以实现更快、更高效的通信。EtherCAT 支持超短数据更新时间和超精准数据同步性能，能够满足自动化技术对于实时特性的需求。

1.1.1 EtherCAT

1.1.1.1 Functional Principle

EtherCAT 是一种实时、高速、灵活的以太网协议。在 EtherCAT 网络中，主站发出的以太网帧会经过所有的从节点。每个节点不再需要接收标准以太网包或帧，并将其解释和复制为过程数据。取而代之的是，当报文通过一个从设备时，该设备会识别并读取与自身相关的数据；与此同时，输入数据会被插入至报文中。因此，数据在传输过程中就已经得到相应处理。最特别的是，整个网络只需一个以太网帧便可完成通讯。与其他基于以太网的通信解决方案相比，EtherCAT 充分利用了可用的全双工带宽。

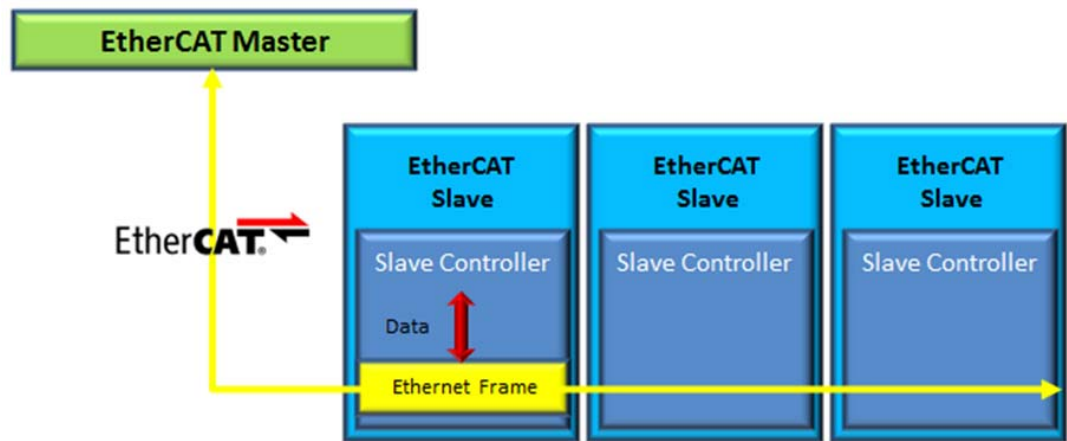


图 1.1: EtherCAT Function Principle

1.1.1.2 Protocol

EtherCAT 主站和从站之间执行周期性数据交换。EtherCAT 帧中的数据通过 Ethertype 协议 0x88a4 直接传送至标准 IEEE 802.3 以太网帧，并将在传输过程中由 EtherCAT 从站控制器进行处理。每个 EtherCAT 报文都是一条命令，包含文头、数据和工作计数器。数据报文文头包含主设备的访问类型信息：

- 读，写，读 - 写
- 通过直接寻址访问指定从设备
- 通过逻辑寻址访问多个从设备

逻辑寻址主要用于过程数据的周期性交换。文头和数据用于指定从站需要执行的操作；工作计数器由从站进行更新，之后主站会收到从站已完成命令的通知。

每个 EtherCAT 报文都以 16 位工作计数器（WKC）结尾。工作计数器对已通过 EtherCAT 报文成功寻址的设备进行计数。

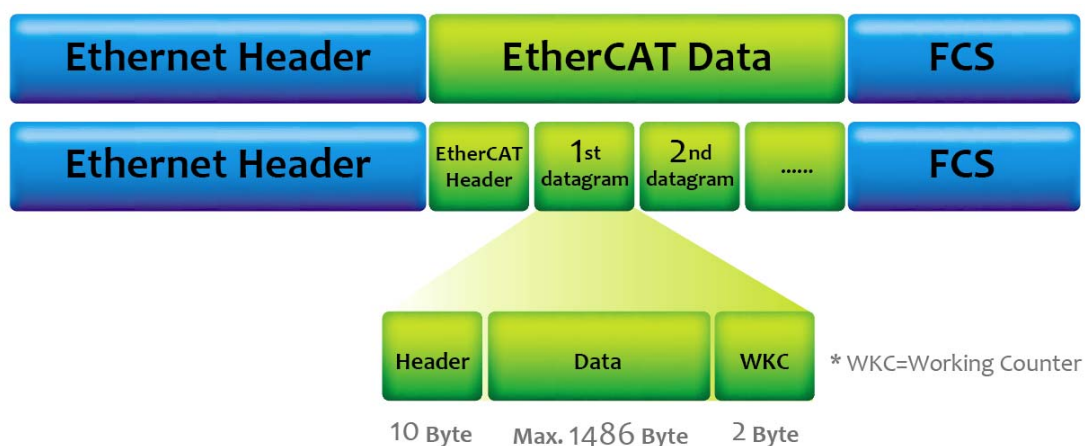


图 1.2: EtherCAT 协议

对 EtherCAT 报文的处理在整个帧被接收完成之前业已开始。

若数据无效，则帧检查概要（FCS）为无效，且从站不会将相应数据设置为对本地应用有效的数据。

1.1.1.3 拓扑结构

EtherCAT 支持各种网络拓扑结构，包括总线形、树形、环形或星形。其中，所需连接更少的总线形和树形结构更加适合现场总线应用。这两种结构采用更加简单灵活的布线模式，无需交换机和集线器。

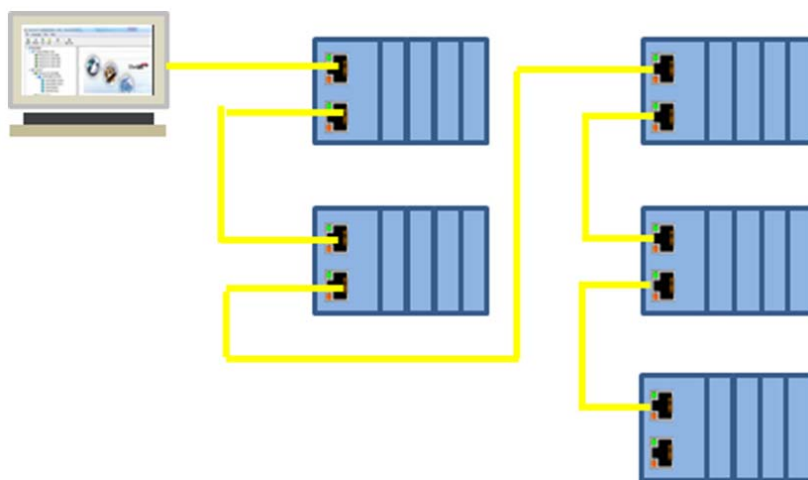


图 1.3: EtherCAT 拓扑结构

在 100BASE-TX 模式中，通过价格低廉的工业以太网电缆相连的两个节点之间的距离可达 100 米。EtherCAT 可通过几百个可用节点构建一个完整总线或一个总线型拓扑结构，并且不受任何限制。此外，EtherCAT 可连接多达 65,535 台设备，因此其网络扩展能力接近于无限。

在操作过程中，EtherCAT 协议支持连接或断开单个节点。即使网络中的某个从站被移除，剩余的网络组成部分仍然能正常运行。另外，EtherCAT 还支持其他通信特性，例如电缆冗余、乃至支持热备份的主站冗余。

1.1.1.4 同步

分布式时钟（DC）机制支持 EtherCAT 网络中两个从站之间的高精度时间同步，完全符合 IEEE 1588 精确时间协议标准。EtherCAT 可通过分布式时钟实现所有本地总线设备之间冗余极小的时间同步。所有的 EtherCAT 从站都内置有一个名为系统时间（ $t_{Local\ Time}$ ）的时钟。通常情况下，第一个 EtherCAT 从站将作为参考时钟进行周期性的时钟分布。

有时，参考时钟与其他从站时钟之间可能发生失调，其原因在于：当一个从站启动时，保持当前时间的内部独立寄存器被重置为零。不幸的是，该操作不会在所有不同从站中同时发生，因此会在时钟之间造成初始偏移量（ t_{offset} ）并需对其进行补偿。

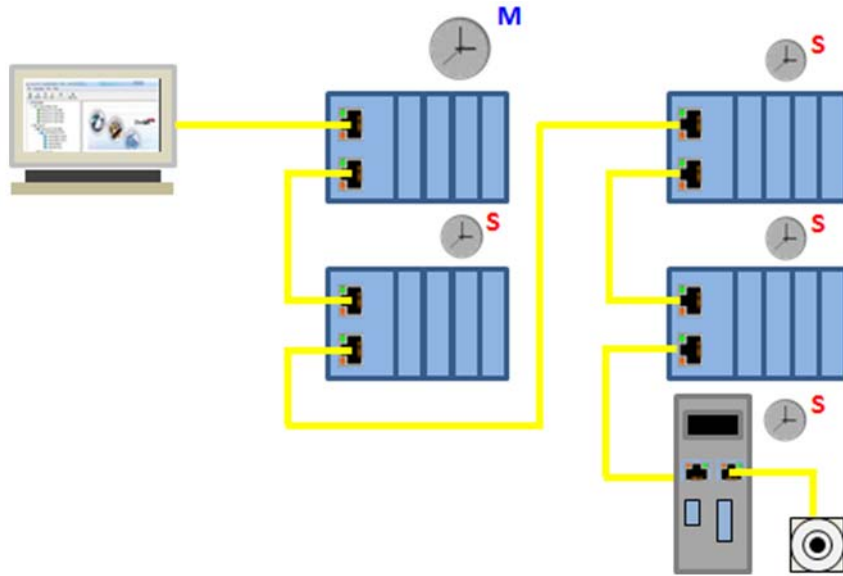


图 1.4：EtherCAT 分布式时钟

主站将向系统中的所有从站发送广播。一旦接收到信息，从站将锁存器内部时钟的值。锁存值有两种：接收时的值和返回时的值。之后，主站将读取所有锁存值，并计算每个从站的延迟（ $t_{Propagation\ Delay}$ ）。该延迟将被存储在偏移量寄存器中。接下来，主站会定期向 EtherCAT 网络中的所有从站发送信息，并将第一个从站作为参考时钟，促使所有从站根据计算出的偏移量来设置各自的内部时钟。

$$\Delta t = (t_{Local\ Time} + t_{Offset} - t_{Propagation\ Delay}) - t_{Received\ System\ Time}$$

在 DC 模式中，从站之间的同步由内部时钟通过硬件执行，因此 EtherCAT 能够保证时间抖动少于 1 微秒。

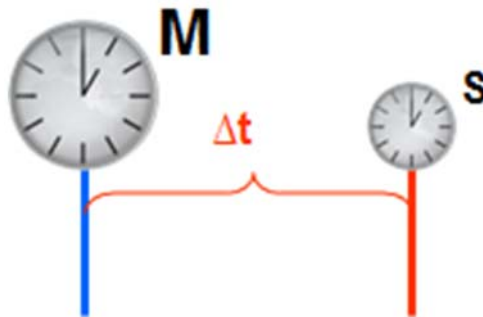


图 1.5：EtherCAT 分布式时钟抖动

1.1.1.5 Diagnosis with exact localization

EtherCAT 是一个超高速 I/O 系统。为了实现最出色的高速通信，就要求具有超高通信精确度。EtherCAT 具备系统固有的广泛诊断特性，可精确检测和定位系统故障。除了线缆破损检测和定位之外，EtherCAT 系统的协议、物理层和拓扑结构使每个单独传输段具备独立的质量监控能力。

如上所述，每个 EtherCAT 报文都以一个 16 位工作计数器（WKC）结尾，从而对该 EtherCAT 报文成功寻址的设备进行计数。主站可以通过 WKC 检测相应周期的数据交换情况，并通过分析节点错误计数器发现错误帧。只有在接收到正确帧时，从站应用才开始运行。相应错误计数器的自动评估功能将精确定位关键网络段。

通过分析循环冗余校验码（CRC）的校验和能够可靠检测传输中的位错误。CRC 作为错误检测码，通常被数字网络和存储设备用于检测原始数据的偶然更改。除了错误检测和定位协议，EtherCAT 系统的传输物理层和拓扑结构支持每个传输路径的独立质量监控。EtherCAT 支持一种非常高效的监控机制。

1.2 PCI-1203 特性

研华 EtherCAT 硬件主站解决方案包含 PCI-1203 以及两个分别作为运动控制器和 I/O 总控制器的 EtherCAT MAC。PCI-1203 能够为 EtherCAT 通信自动执行数据交换与内存寻址进程。EtherCAT 的高性能通信特性使用户无需应对复杂的 EtherCAT 通信协议。

1.2.1 多主站模式

- 支持多个单独的主站同时使用 PCI-1203 上的不同 EtherCAT 端口。
- 运动主控制器的循环时间小于 500 微秒
- I/O 主控制器的循环时间小于 200 微秒

1.2.2 多轴插补

- 支持 Panasonic A5B EtherCAT 伺服电机 32 轴插补
- 支持 6 个群组，每个群组支持 8 轴线性插补

1.2.3 强大的 I/O 数据处理功能

- 支持研华 ADAM-5000/ECAT EtherCAT I/O 从站
- 额外支持板载 8 路隔离 DI 和 4 路隔离 DO

1.2.4 板载实时 OS 支持

- 通过板载 650 MHz 双核 ARM 处理器处理数据，无需占用 PC 中的 CPU 资源
- 支持高准确度轨迹规划和快速响应时间
- 提升周期性数据交换速度

1.2.5 分布式时钟（DC）

- 通过一个从站参考时钟同步所有从站时钟
- 延迟补偿
- 运行时监控和偏差修正

1.2.6 支持 Common Motion SDK

- 通过“Common Motion API”整合所有研华 SoftMotion 控制器

- EtherCAT 从站完全支持通过 API 配置，无需 ENI 数据
- 图形实用程序界面易于使用并可配置 EtherCAT 系统

1.2.7 错误检测与诊断

- 链接中断监控
- 超时及 EtherCAT 命令失败时检测并重试
- 物理、设备及主站层的综合性诊断数据

1.3 产品规格

1.3.1 EtherCAT

环数	2
浪涌保护	10kV
通信时间	Motion: 500us I/O: 200us
通信运动从轴	32 Servo Drive Max. (eq. Panasonic A5B)
通信 I/O 从轴	128 port DI (128 byte)
	128 port DO (128 byte)
	128 channel AI (256 byte)
	128 channel AO (256 byte)
	*(参考 ADAM-5000/ECAT)

1.3.2 隔离数字量输入

通道	8
输入电压	Logic 0: 5 V max.
	Logic 1: 6 V min. (24V max.)
	需要 24VDC 外部电源
输入电阻	8.4 k Ω
输入延迟时间	150us
隔离保护	1,000 VDC

1.3.3 隔离数字量输出

通道	4
输出类型	Sink
输出电压	12 ~ 24 VDC
汇点电流	300mA/ 每通道 (23° C)
	200mA/ 每通道 (60° C)
切换频率	30KHz @ 24VDC 外部电源
隔离保护	1,000 VDC

注!

1. 外部电源范围: +12VDC to +24VDC
2. DO 切换频率根据 4.7K ohm 的电阻负载估算。
 - 30KHz @ 24VDC 外部电源
 - 50KHz @ 22VDC 外部电源
 - 100KHz @ 18VDC 外部电源
 - 200KHz @ 12VDC 外部电源

1.3.4 General

总线类型	通用 PCI V2.2
电源	+5VDC (PCI-Bus)
认证	CE、FCC A 级
接口	RJ45 x 2, DB15 x 1
尺寸 (L x H)	175 x 100 mm (6.9" x 3.9")
功耗	5 VDC @ 0.5 A typical
湿度	5 ~ 95% RH, 非凝结 (IEC 60068-2-3)
工作温度	0 ~ 60° C (32 ~ 140° F)
储存温度	-20 ~ 85° C (-4 ~ 185° F)

第 2 章

安装

2.1 打开包装

收到 PCI-1203 包装后，请首先检查里面的物品。包装内应包括以下各项：

- PCI-1203 卡
- CD-ROM（包含 DLL 驱动程序和用户手册）

PCI-1203 卡的一些电子元件极易受到静电放电（ESD）的损害。如果保护措施不当，则集成电路和某些元件极易被 ESD 损害。

将卡从静电屏蔽袋中取出之前，用户应按照以下步骤的指导来防止可能的 ESD 损害：

- 用手触摸机箱的金属部分来释放身体所附的静电，或者也可以使用接地母线。
- 打开静电屏蔽袋之前，使其接触机箱的金属部分。
- 取卡时，只能握住卡的金属托架。

将卡取出后，请首先：

- 检查卡上是否有明显的外部损伤（元件松动或损坏等）。如果有明显损坏，请立即联系研华服务部门或者当地销售代表。切勿将损坏的卡安装至系统。

另外，安装时也请注意以下事项：

- 用户还应避免接触带有静电的材料，如塑料、乙烯基和泡沫聚苯乙烯。
- 拿卡时请只握住卡的边缘。不要碰触接口或电子元件露在外部的金属针脚。

2.2 安装驱动

建议用户先安装驱动，再安装 PCI-1203 板卡。

板卡的 DLL 驱动安装程序位于板卡包装所附的光盘中。按照以下步骤安装驱动程序：

1. 将产品所附光盘插入光驱。
2. 如果用户的系统开启了自动播放功能，安装程序将自动运行。
3. 根据您的操作系统选择适合的 Windows 操作系统选项。然后请按照指导逐步完成 DLL 驱动的安装。
4. 选择自动安装 PCI-1203 Motion Utility。

如需驱动相关的更多信息，请参考设备驱动手册的在线版本：

`\Program Files\Advantech\Common Motion\Manual`

示例源代码可在相应安装文件夹下找到，如默认安装路径为：

`\Program Files\Advantech\Common Motion\Example_1203`

2.3 安装硬件

注！ 安装卡之前，请确认已安装了驱动。（请参考 2.2 节“安装驱动”）



DLL 驱动安装完成后，用户即可将 PCI-1203 卡插入计算机的任一 PCI 插槽。若有任何疑问，请参考计算机的用户手册或其它相关文档。请按照以下步骤安装卡。

1. 关闭计算机并移除与之相连接的附件。

警告！ 无论何时安装或移除板卡、连接或断开电缆时，都请关闭计算机电源。



2. 断开连接到计算机后部的电源线和其它电缆。
3. 移除计算机盖。
4. 选择一个空闲的 +3.3/5 V PCI 插槽。移除用于将扩展插槽盖固定在系统上的螺丝。螺丝将用于固定接口卡上的固定支架。
5. 小心地握住 PCI-1203 的上缘。将固定支架上的孔对齐扩展插槽上的孔，然后将金手指接口对准扩展插槽。轻轻地将卡向下稳按入插槽。请确认卡已固定在插槽上。请避免用力过大，否则也许会损坏卡。
6. 用螺丝将 PCI 卡托架固定在计算机后面板导轨上。
7. 将相关附件（10 电缆、接线端子等，若有必要）连接至 PCI 卡。
8. 将计算机盖安装回原位，并连接第 2 步中断开的所有电缆。
9. 启动计算机。

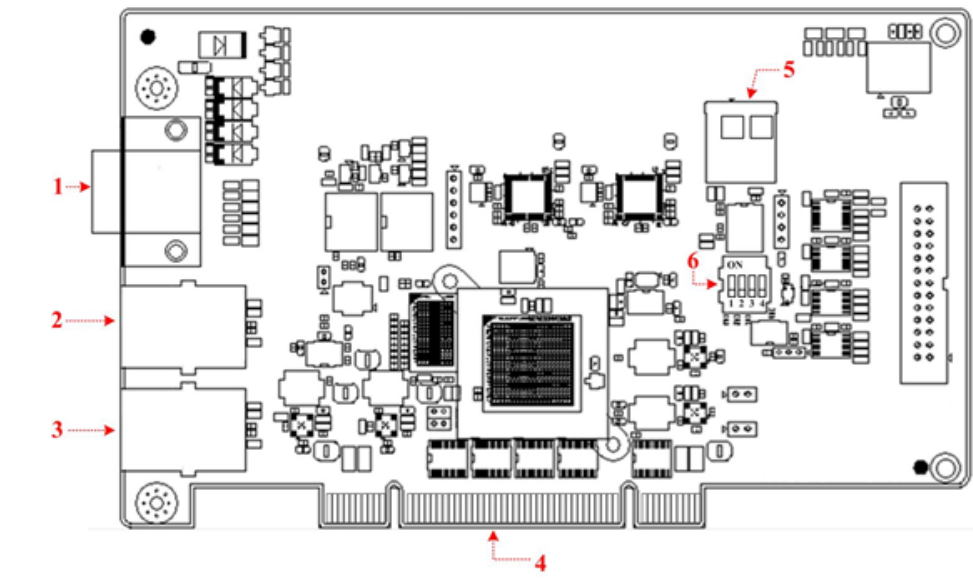
第 3 章

硬件服务

3.1 外围

PCI-1203 的接口及开关位置见表 3.1。

表 3.1：接口及开关		
编号	接口	功能
1	CN26	隔离数字量输入 / 输出接口
2	CN21	RJ45，带变压器（LAN_1）
3	CN20	RJ45，带变压器（LAN_0）
4	金手指	PCI 总线
5	CN25	Micro-SD 卡槽（不支持）
6	SW1	板卡 ID 开关
7	CN30	从闪存启动主板（Pin1 与 Pin2 短接）



注！ 使用 PCI-1203 之前，请确认 Pin1 与 Pin2 已通过跳线短接。



3.2 板卡 ID 开关（SW1）

表 3.2：板卡 ID 设置				
	针脚	标签	Switch ON	Switch OFF
	1	ID3	0 (ID3 Value =0)	1 (ID3 Value =1)
	2	ID2	0 (ID2 Value =0)	1 (ID2 Value =1)
	3	ID1	0 (ID1 Value =0)	1 (ID1 Value =1)
	4	ID0	0 (ID0 Value =0)	1 (ID0 Value =1)

板卡 ID = (8×ID3 值) + (4×ID2 值) + (2×ID1 值) + (ID0 值)
默认设置为 ON，例如：默认值为 0。

3.3 LAN 端口 (CN20, CN21)

两个 RJ45 10/100/1000 LAN 端口通过 Cat. 5e LAN 线缆连接 EtherCAT 从站。CN20 专为 EtherCAT 电机驱动器从站而设计（如：Panasonic A5B）。CN21 专用于 EtherCAT IO 从站（如：ADAM-5000/ECAT）。

3.4 D-sub 接口 (CN26)

PCI-1203 上的 D-Sub 15 针接口支持 8 路汇点式数字量输入 (DI) 和 4 路汇点式数字量输出 (DO)，并可通过屏蔽电缆连接布线板。CN26 的针脚定义如表 3.3 所示。屏蔽电缆设计中的线对，如 (pin1, pin6)、(pin2, pin 7)、(pin3, pin8)、(pin4, pin9)、(pin5, pin14)、(pin11, pin12) 和 (pin14, pin15)，都需要是双绞线。

表 3.3: D-Sub 针脚定义



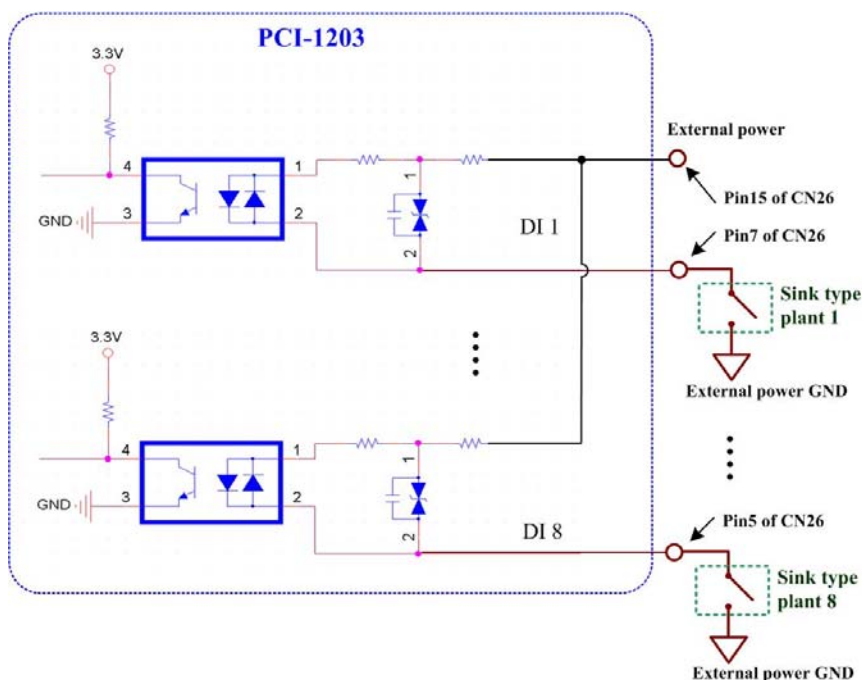
CN26

针脚	说明	针脚	说明
1	DO 3	9	DI 5
2	DI 2	10	DI 7
3	DI 4	11	DO 2
4	DI 6	12	DO 4
5	DI 8	13	外部电源接地
6	DO 1	14	外部电源接地
7	DI 1	15	外部电源 (+24V)
8	DI 3		

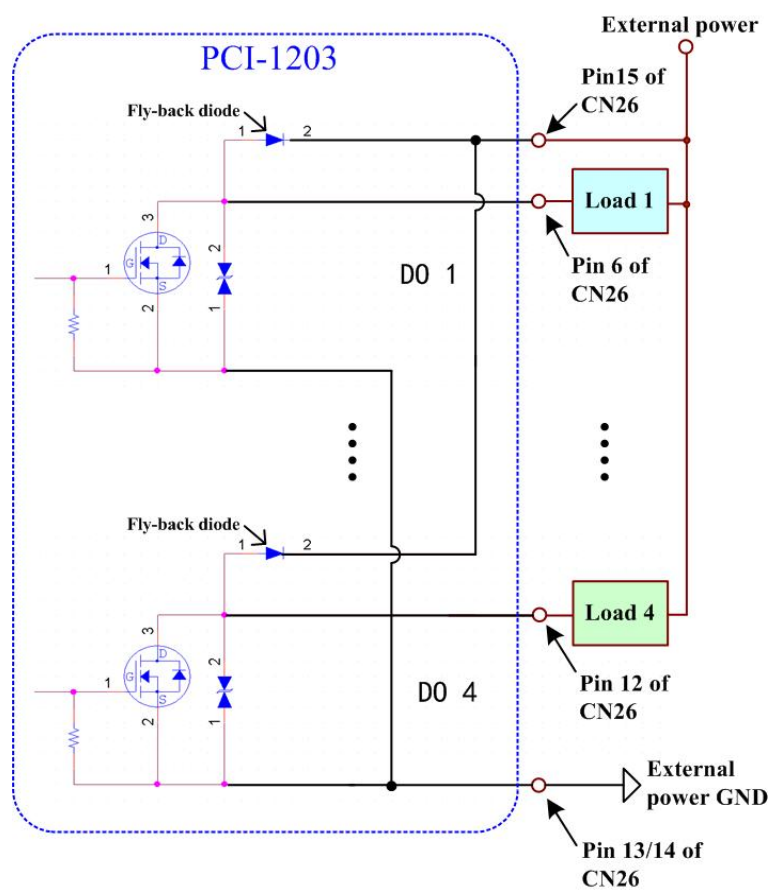
3.5 信号连接

本节内容介绍如何连接 PCI-1203 的输入和输出信号。

3.5.1 数字量输入布线参考



3.5.2 数字量输出布线参考



第 4 章

EtherCAT 实用程序

4.1 简介

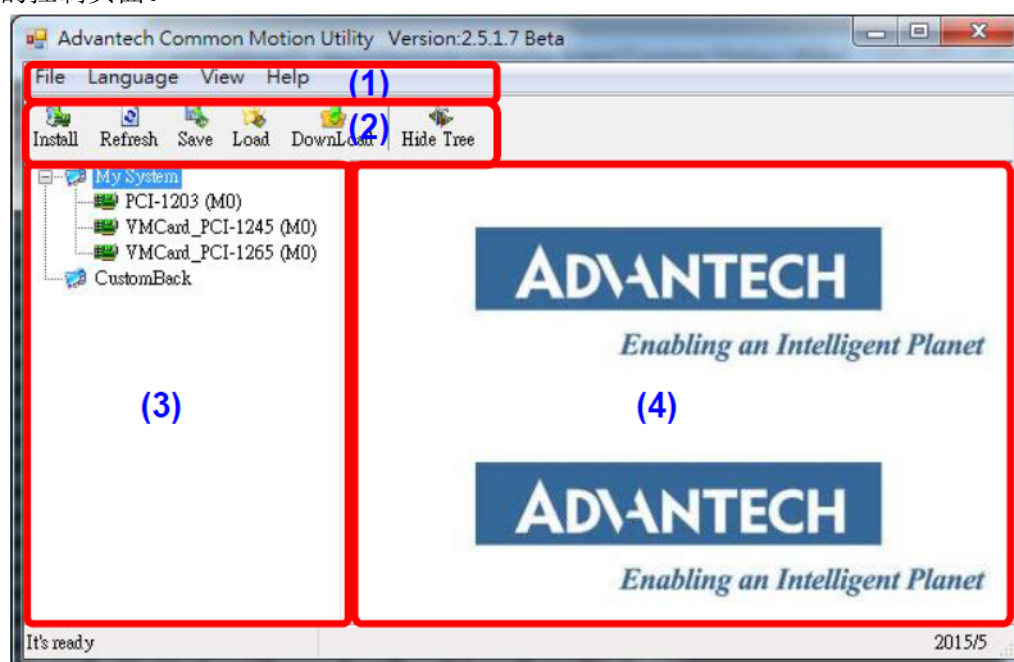
PCI-1203 使用 Common Motion 实用程序进行功能测试。该实用程序的开发基于 Common Motion API 和 Common Motion Component，因此支持所有基于 Common Motion 的产品，例如 PCI-1240U、PCI-1245 和 PCI-1285 系列等运动控制卡。也就是说，用户可以使用同一实用程序来测试 PC 上所有基于 Common Motion 的板卡。

本章将陆续介绍以下接口：

1. 主界面：实用程序主界面。包含主菜单、工具栏设备树和控制面板。
2. 设备页面：包含 PCI-1203 的大部分操作选项。包含 Single Axis Motion、Multi-Axis Motion、Synchronized Motion、IO Operations 和 Mapping Table。
3. 从站页面：包含从站的操作选项。包含 Single Axis Motion、GPIO Operations 和 Motion IO Settings。
4. ADAM-E5000 模块页面：包含 ADAM-E5000 模块的操作和配置选项。

4.2 主界面

主界面的组成部分包括：(1) 主菜单，(2) 工具栏，(3) 设备树，(4) 控制面板。若安装成功，设备将显示在设备树中。用户在设备树中选定设备后，控制面板中将显示相应的控制页面。

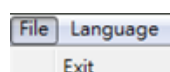


4.2.1 主菜单

主菜单选项如下图所示：

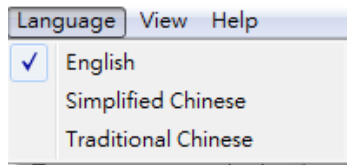


4.2.1.1 文件 (File) 菜单项



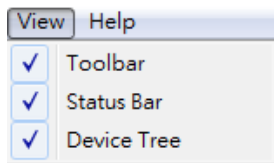
单击 [Exit] 终止该进程。

4. 2. 1. 2 语言 (Language) 菜单项



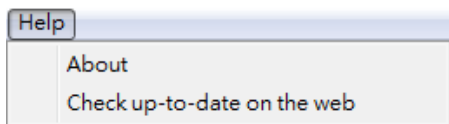
通过该菜单可切换测试工具的语言。该实用程序支持三种语言：英文、简体中文和繁体中文。选择一种语言后，相应的菜单项为勾选状态。关闭测试工具时，所选语言会保存至注册表。再次打开时，测试工具的语言将为上次使用的语言。

4. 2. 1. 3 视图 (View) 菜单项



该菜单允许用户显示 / 隐藏工具栏、状态栏和设备树。如果可以看到工具栏 / 状态栏 / 设备树，则对应菜单项为勾选状态。

4. 2. 1. 4 帮助 (Help) 菜单项



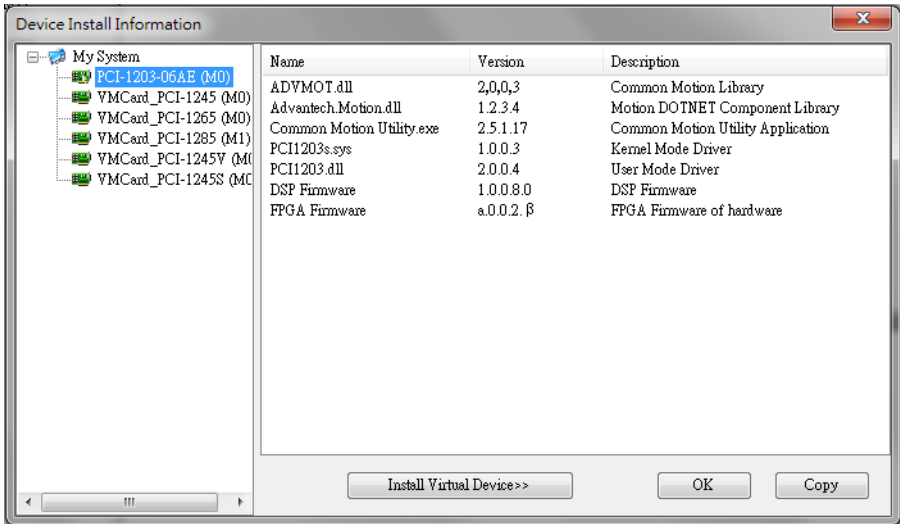
[About] 菜单项提供设备驱动和测试工具的版权声明信息。单击 [Check up-to-date on the web] 将链接到公司的网站，通过比较 “Install” 界面的的版本信息检查固件、驱动和测试工具是否为最新版。

4. 2. 2 工具栏



4. 2. 2. 1 安装 (Install) 工具项

单击 [Install] 将弹出一个新窗口，显示驱动、硬件、固件和测试工具的版本信息。



ADVMOT.dll: 用于开发的通用运动 API。
Advantech.Motion.dll: NET 运动控制数据库。
Common Motion Utility.exe: 正在运行的测试工具。
PCI1203s.sys: 内核模式驱动程序。
PCI1203.dll: 用户模式驱动程序。

4.2.2.2 更新 (Refresh) 工具项

重新搜索计算机中可用的通用运动卡，重新加载设备树。

4.2.2.3 保存 (Save) 工具项

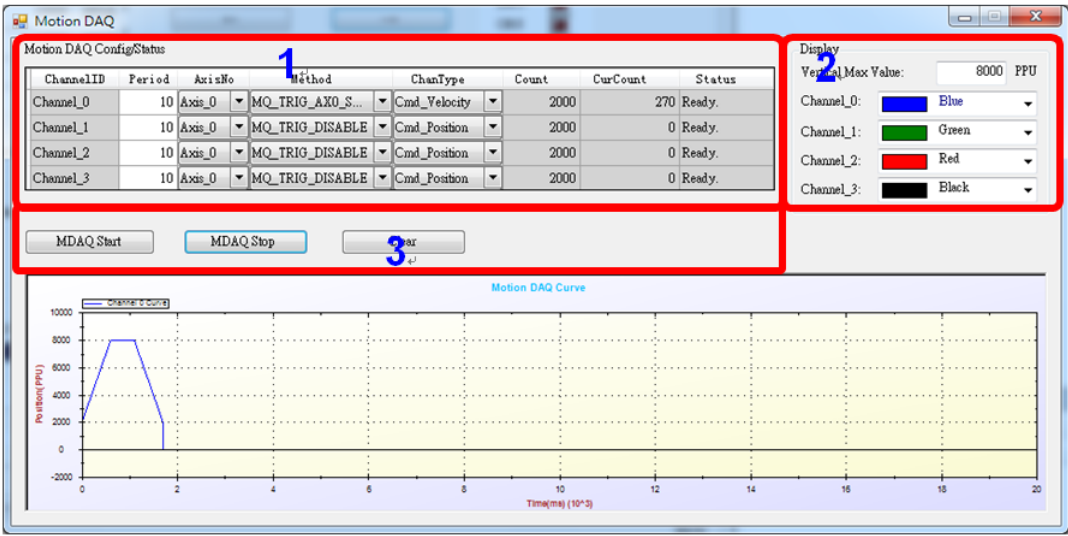
导出所选设备的轴的全部属性。

4.2.2.4 导入 (Load) 工具项

导入所选设备的全部轴的配置信息。选择之前导出的配置文件，然后可将配置文件导入到设备硬件。

4.2.2.5 数据采集 (MotionDAQ) 工具栏

单击按钮后，用户将看到如下界面：



该工具按钮主要展示运动数据获取（Motion Data Acquisition）功能。
在 PCI-1203 中，提供了 4 个通道进行运动数据的采集，任一通道都可采集任一轴的理论（Command）/ 实际（Actual）/ 偏差位置（Lag，理论与实际之差）的运动数据，采集数据的最大数为 2000。

该界面由以下部分组成：

1. Motion DAQ 配置 / 状态 (Motion DAQ Config/Status)

进行各个通道采集数据的配置。其中，ChannelID、Count、CurCount 和 Status 列是只读的（背景色为灰色）。

ChannelID	Period	AxisNo	Method	ChanType	Count	CurCount	Status
Channel_0	10	Axis_0	MQ_TRIG_DISABLE	Cmd_Position	2000		
Channel_1	10	Axis_0	MQ_TRIG_DISABLE	Cmd_Position	2000		
Channel_2	10	Axis_0	MQ_TRIG_DISABLE	Cmd_Position	2000		
Channel_3	10	Axis_0	MQ_TRIG_DISABLE	Cmd_Position	2000		

ChannelID: 共提供 4 个通道。

AxisNo: 选择要测量的轴。

Period: 采样周期，即每隔多长时间（ms）采集一个数据。范围为 1-255ms。为了统一图形框的横坐标最大值，各通道的Period将采用同一个值。因此，若一个通道的Period值改了，则所有通道的 Period 值将会相应改变。

Method: 触发方式。数据采集的触发方式有如下几种：

MQ_TRIG_DISABLE: 禁用数据采集功能。

MQ_TRIG_SW: 点击 [MDAQ Start] 触发采集。

MQ_TRIG_AXN_START: N 轴开始运动时触发。

ChanType: 数据采集的来源。可选值如下：

--Cmd_Position: 理论位置 (Command Position)。

--Act_Position: 实际位置 (Actual Position)。

--Lag_Position: 偏差位置 (Lag Position)，即理论位置与实际位置的差值。

--Cmd_Velocity: 理论速度 (Command Velocity)。

Count: 采集数据的个数，范围为 0-2000，Utility 中默认为最大值 2000。

CurCount: 开始采集运动数据后，会返回当前已采集的数据个数。

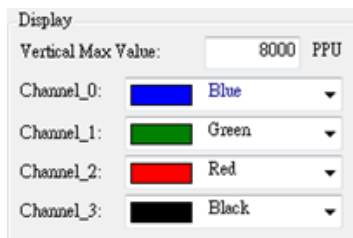
Status: 显示当前采集的状态：

--Ready: 尚未启动运动数据采集功能；

--Wait Trigger: 已启动运动数据采集功能，但条件尚未触发；

--Started: 正在进行运动数据采集。

2. 显示 (Display)



配置各个通道曲线的颜色和图形框纵坐标最大值。从下列框中选中相应的颜色，切换后，对应通道曲线的颜色将会随着变化。

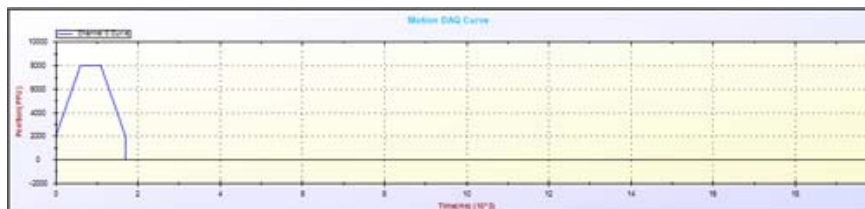
3. 功能操作

--MDAQ Start: 启动运动数据采集功能。当满足触发条件后，开始采集运动数据。

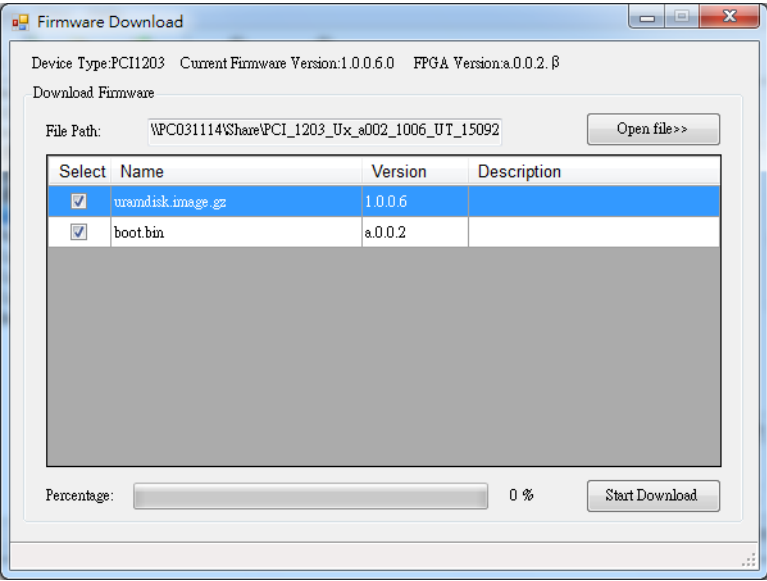
--MDAQ Stop: 停止运动数据采集。

--Clear: 清除各个通道的曲线图。

当 MDAQ Start 开始后，若满足触发条件，则开始采集数据，可查看当前各个通道采样点数和目前的状态，当数据采集完后，将在下面的图形框中画出各个通道的相应采集数据的曲线图，如下图所示：



4.2.2.6 下载 (Download) 工具栏



该工具用于升级 PCI-1203 固件。该对话框的顶部工具栏包括：当前设备类型、名称和固件版本。点击 [Open File] 选择所获取的最新固件文件。该固件包中的 FPGA 和固件版本将显示在数据列表中。点击 [Start Download] 激活下载至硬件的进程，进度条将显示任务进度。升级完成后，请关机并重启计算机，以确保 PCI-1203 已载入最新固件版本。

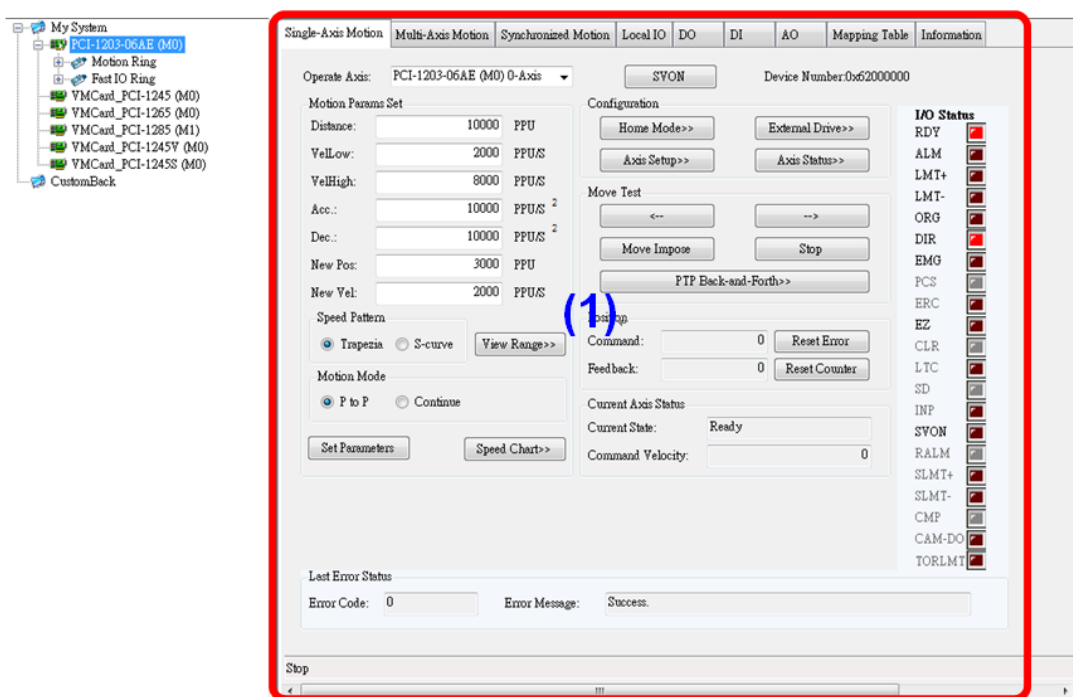
注！ 在下载固件至硬件的过程中，切勿关闭升级对话框和测试工具，
 当固件升级正在进行时切勿关闭电源，以防造成损坏。若下载进程因断电或其他问题而中断，用户需将硬件返回研华进行固件更新。

4.2.2.7 隐藏树 (Hide Tree) 工具栏

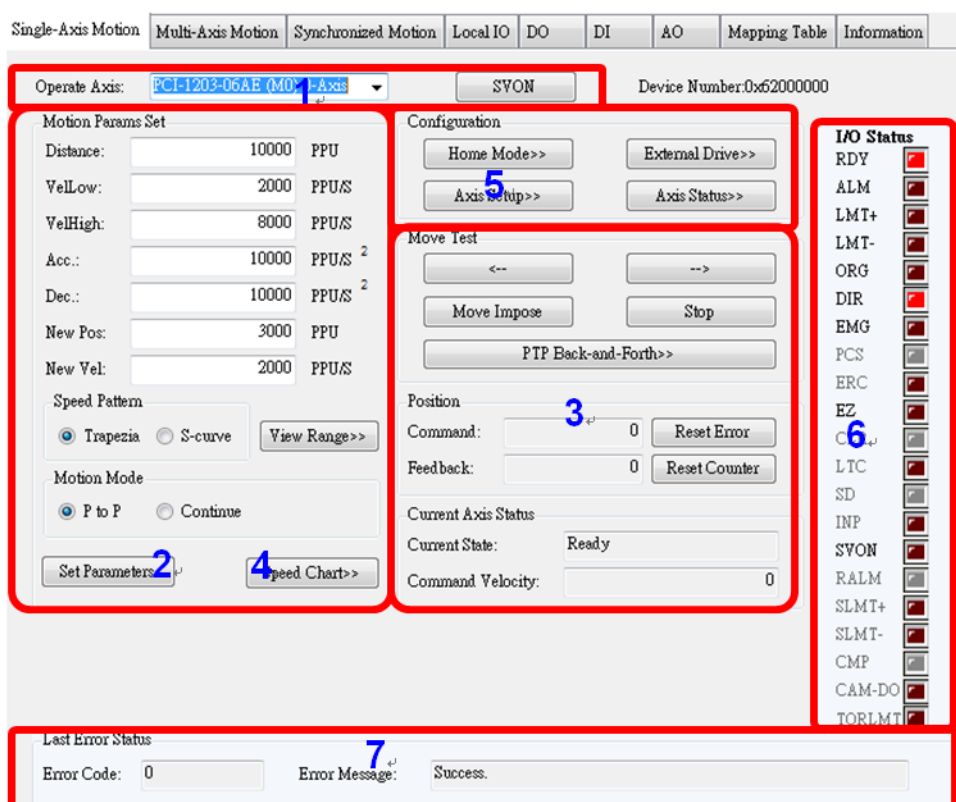
此按钮用于隐藏 / 显示设备树。
若 Device Tree 目前为显示状态，则点击此按钮将隐藏 Device Tree，且按钮上的文字变为 “Show Tree” ；
若 Device Tree 已隐藏，则点击此按钮将显示 Device Tree，且按钮上的文字变为 “HideTree” 。

4.3 主站页面

在 Device Tree 中选择 PCI-1203 卡后，操作页面将显示在 (1) 控制面板中；同时，所有连接至 PCI-1203 的从设备将显示为 PCI-1203 节点的子节点。
PCI-1203 节点的控制页面包含：单轴运动、多轴运动、同步运动、DAQ（包括 DI，DO，AI，AO）、IO 映射表和设备信息页面。有关从设备的更多操作选项，用户可点击 Device Tree 中的从节点进行查看。相应的控制页面将显示在控制面板中。



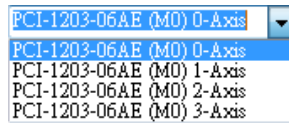
4.3.1 单轴运动控制 (Single-Axis Motion)



4.3.1.1 轴操作和 Servo ON

点击下拉组合框，选择需要操作的轴。轴索引由网络连接位置定义。连接至 PCI-1203 的第一个轴将对应索引 0。第二个轴将对应索引 1，以此类推。组合框中的条目数取决于 PCI-1203 所连接的轴数。

例如，下图中的组合框有 4 个条目，表示 PCI-1203 连接了 4 个轴。

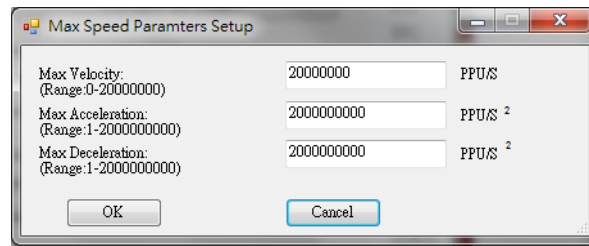


点击 [SVON] 按钮可以开启 / 关闭所选轴的电机。

4.3.1.2 设置运动参数

设置距离 (Distance)、初速度 (VelLow)、运行速度 (VelHigh)、加速度 (Acc.)、减速度 (Dec.) 以及速度类型 (Trapezia or S-Curve) 以支持点对点 (P To P) 和连续运动 (Continuous)。参数 New Pos. 和 New Vel. 用于叠加运动。

点击 [View Range] 查看或设置速度最大值、加速度和减速度。对话框如下图所示：



注！ 单轴运动 (Single-axis Motion) 中 VelHigh 的值不能大于速度最大值；Acc. 的值不能大于加速度最大值；Dec. 的值不能大于减速度最大值。



完成参数设置之后，单击 [设置参数]，将参数值设置到设备中。

4.3.1.3 运动测试



完成参数设置之后，点击 [←] 或 [→]，指定轴将执行正向或反向点对点 / 连续运动。点击 [Stop] 按钮停止轴运动。命令 (Command) 位置和反馈 (Feedback) 位置将显示在位置 (Position) 区块上。当前轴状态 (Current Axis Status) 区块将显示当前状态 (Current State) 和命令速度 (Command Velocity)。

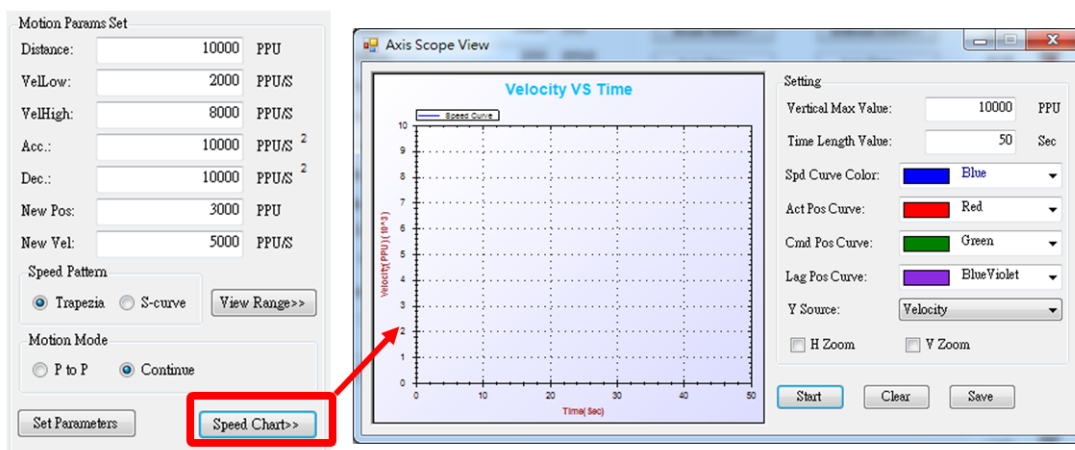
关于轴状态的更多信息，请参考通用 API 编程指南中的 Acm_AxGetState 函数。

当运动速度达到 P2P 运动中的 VelHigh 值后，用户可以点击 [Move Impose] 生成叠加运动，该运动的距离为 New Pos 值，速度为 New Vel 值。

点击 [PTP Back-and-Forth] 之后，将显示进阶 P2P 运动窗口。该功能支持用户设置多轴同时进行 P2P 运动。

4.3.1.4 速度曲线 (Speed Chart)

点击该按钮查看速度曲线（如下图所示）。



参数说明如下：

1. Vertical Max Value: 最大垂直坐标。
2. Time Length Value: 最大水平坐标。
3. Spd Curve Color: 速度曲线的颜色。
4. Act Pos Curve: 实际位置曲线的颜色。
5. Cmd Pos Curve: 理论位置曲线的颜色。
6. Y Source: 垂直坐标的数据来源。可能是速度、实际位置、理论位置，或三者皆是。
7. H Zoom: 若选中该项，表示支持水平缩放，即用户可用鼠标选中并放大相应区域。
8. V Zoom: 若选中该项，表示支持垂直缩放，即用户可用鼠标选中并放大相应区域。

点击 **Start** 描绘曲线。

点击 **Clear** 清空绘画区域。

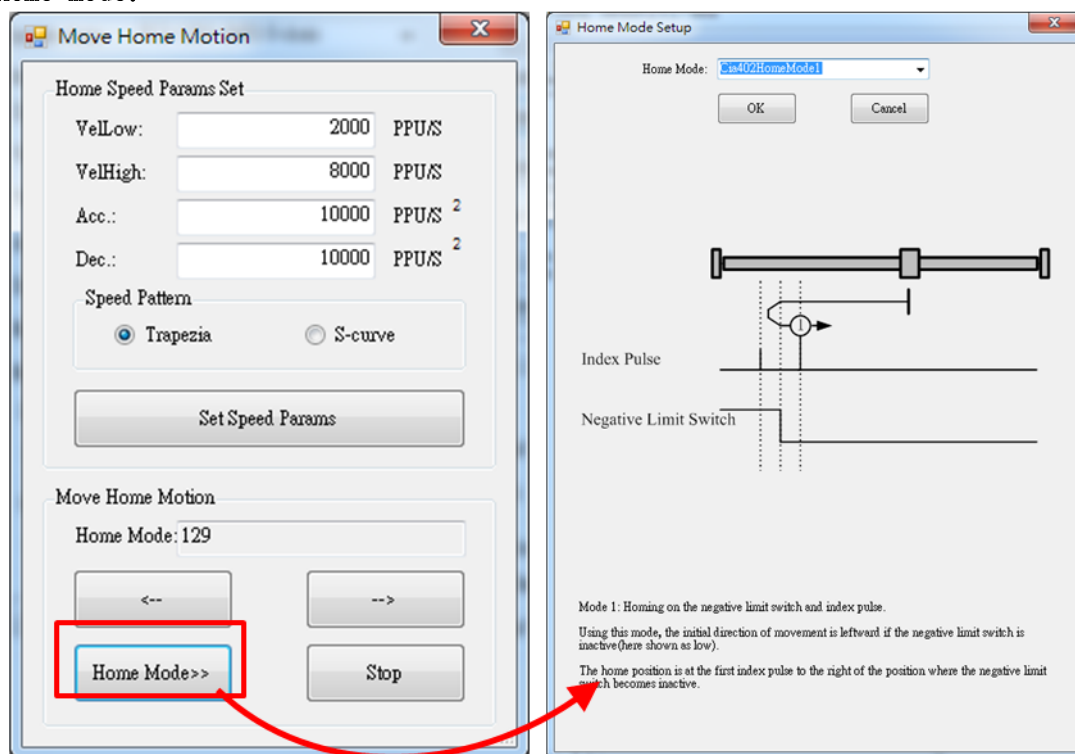
点击 **Save** 将曲线保存为 .png、.gif、.jpg、.tif 或 .bmp 格式的文件。

4.3.1.5 配置 (Configuration)



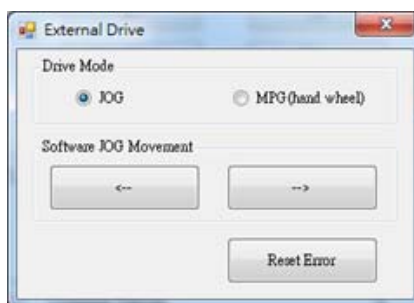
该项包括轴的回原点模式 (Home Mode)、外部驱动 (External Drive) 模式、轴的属性配置 (Axis Setup) 和轴的状态显示 (Axis Status)。

Home mode:



在执行回零运动之前，用户需要先选择回零模式。PCI-1203 支持 CiA402 中定义的所有回原点模式。Homing 操作的简明步骤见回零模式设置（Home Mode Setup）窗口。详情请参考通用 API 编程指南中对于 Home Mode 的描述。在“Home Speed Params Set”区块设置回原点速度参数，点击 [Set Speed Params] 按钮应用设置。完成所有设置后，点击 [←] 或 [→] 开始 Homing 操作。

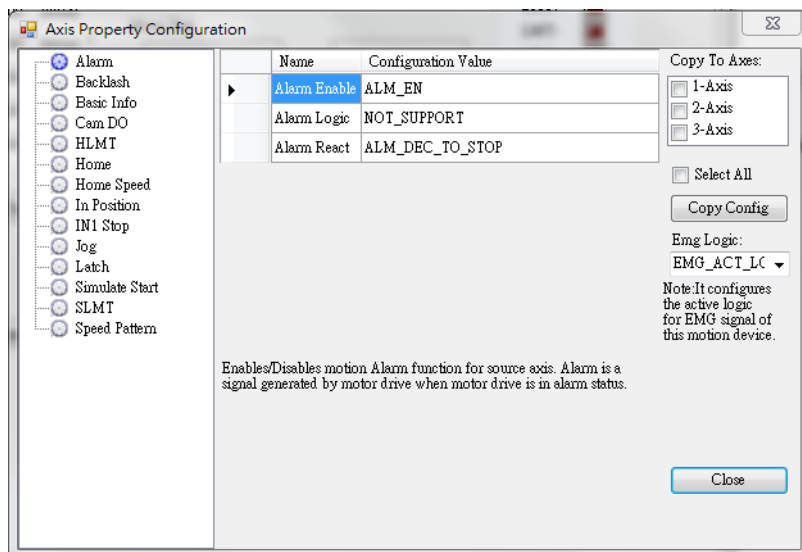
外部驱动 (External Driver)



点击此按钮启动外部驱动功能。PCI-1203 目前只支持 JOG 驱动（不支持 MPG）。设置外部 DI 信号，触发 JOG 功能，请参考轴页面中的运动 IO 环节。

轴配置 (Axis Setup)

点击此按钮可检查 / 设置轴的属性，如下图所示：



左侧树状图显示州的属性分类列表，当用户点击相应选项，右侧数据视图 (Data View) 将显示属性和相应属性值。详情请参考轴的 Feature、Configuration 和 Parameter 描述，见编程指南中的属性列表。

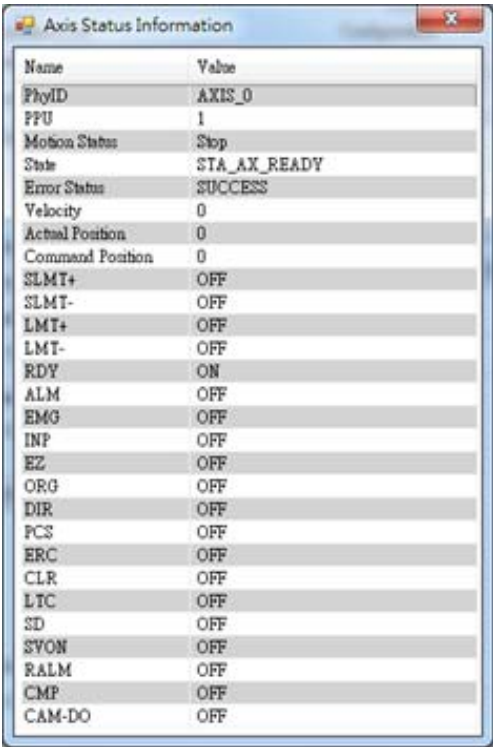
注!



- 在该实用程序中，如果选中的设备不支持相应功能，则该设备将不会显示在左侧树状图中。例如，若选中设备为 PCI-1245/1265 且该设备不支持减速 (SD) 和振动抑制功能，则用户将不会在树状图中看到该设备。同时，由于单轴对话框带有速度参数设置，因此速度类型选项将不作显示。
- 当 “Pulse Out” 分类被选中，“Pulse Out Mode” 属性描述下将显示相应模式的说明。

编辑完成后，当鼠标离开编辑框时，属性值即生效（已在设备中设置）。若要将属性值复制至其他轴，请查看轴列表并点击 [Copy Config]，从而将设置应用至选中轴。

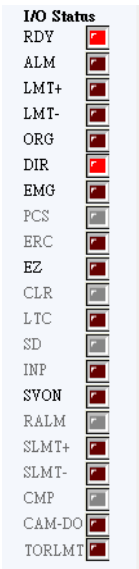
轴状态 (Axis Status)

A screenshot of a software window titled "Axis Status Information". It contains a table with two columns: "Name" and "Value". The table lists various axis parameters and their current values.

Name	Value
PhyID	AXIS_0
PPU	1
Motion Status	Stop
State	STA_AX_READY
Error Status	SUCCESS
Velocity	0
Actual Position	0
Command Position	0
SLMT+	OFF
SLMT-	OFF
LMT+	OFF
LMT-	OFF
RDY	ON
ALM	OFF
EMG	OFF
INP	OFF
EZ	OFF
ORG	OFF
DIR	OFF
PCS	OFF
ERC	OFF
CLR	OFF
LTC	OFF
SD	OFF
SVON	OFF
RALM	OFF
CMP	OFF
CAM-DO	OFF

点击此按钮可查看指定轴信息。例如，PhyID、PPU、基本状态（Motion Status、State、Error Status 等）和 I/O 状态（Alarm、SLMTP/N 等）。

4.3.1.6 I/O 状态 (I/O Status)

A vertical list of I/O status indicators. Each item consists of a label and a small square icon. The icons are either red with a white symbol (indicating ON) or grey (indicating OFF or not supported).

I/O Status	
RDY	
ALM	
LMT+	
LMT-	
ORG	
DIR	
EMG	
PCS	
ERC	
EZ	
CLR	
LTC	
SD	
INP	
SVON	
RALM	
SLMT+	
SLMT-	
CMP	
CAM-DO	
TORLMT	

用户可通过 LED 栏查看 I/O 状态。其中， 表示设备不支持该功能或没有相应 I/O； 表示设备支持该功能，但 I/O 未触发 (OFF)； 表示相应 I/O 已触发 (ON)。

详情请参考通用 API 编程指南中，Acm_AxGetMotionIO 函数对 Status 的描述。

不支持的功能或不存在的项，相应的文本将显示为灰色。对于板卡支持但未被激活的功能，相应的文本也将显示为灰色。对于板卡支持且已被激活的功能，相应的文本将正常显示。

4.3.1.7 错误状态 (Error Status)

Last Error Status	
Error Code: 0	Error Message: Success.

用户可以查看最新的错误代码和错误信息。若没有错误，则错误代码为“0”且错误信息为“SUCCESS”。

4.3.2 多轴运动 (Multi-Axis Motion)

The screenshot shows the Multi-Axis Motion configuration window. It is divided into several sections:

- Operate Axes:** A list of axes (0-Axis to 3-Axis) with checkboxes. A red box highlights this section, with a blue '1' pointing to the list.
- Motion Params Set:** Fields for VelLow, VelHigh, Acc., and Dec., along with Speed Pattern (Trapezia or S-curve). A red box highlights this section, with a blue '2' pointing to the Speed Pattern field.
- Motion Operation:** Basic Interpolation Motion settings, including Movement Mode (Absolute or Relative) and Interpolation Mode (Line, Arc, or Helix). A red box highlights this section, with a blue '4' pointing to the Interpolation Mode field.
- Path Motion:** Settings for Speed Forward, Blending Time, and buttons for Edit Path, Load Path, Move Path, and Move Sel Path. A red box highlights this section, with a blue '5' pointing to the Blending Time field.
- Path Status:** Fields for CurIndex, CurCmd, Path Count, Remain, and FreeCnt, along with a Reset Path button. A red box highlights this section, with a blue '6' pointing to the CurCmd field.
- Position:** A table showing Position (Command and Feedback) for 0-Axis, 1-Axis, 2-Axis, and 3-Axis. A red box highlights this section, with a blue '7' pointing to the 2-Axis Command field.
- Group State:** A field showing the current state (Ready). A red box highlights this section, with a blue '8' pointing to the Group State field.

4.3.2.1 操作轴

窗口中的列表框列出所选设备的全部轴，勾选对应轴，将其添加到群组中。如果添加到群组的轴的数量小于1，则群组状态将为“Disable”。如果添加到群组的轴的数量大于或等于2，群组状态将为“Ready”。进行参数设置后，用户可进行相应插补操作。点击 [SVON]/[SVOFF] 开启和关闭群组中的伺服电机。

4.3.2.2 设置运动参数

可设置的参数包括：群组初速度 (Group VelLow)、群组运行速度 (Group VelHigh)、群组加速度 (Group Acc)、群组减速度 (Group Dec) 及速度类型 (Speed Pattern)。点击 [Set Parameters] 应用设置。

4.3.2.3 设置目标位置

在下图所示界面配置运动中心 / 终点。该对话框将根据群组轴和插补模式来自动启动可写编辑框。

Axes	Line End (PPU)	Arc Center (PPU)	Arc End (PPU)
0-Axis	8000	8000	16000
1-Axis	8000	0	0
2-Axis	8000	8000	16000

如图所示，轴 1 和轴 2 已添加至群组中。若选择 Line 插补模式，可写编辑框为 “1-axis” 和 “2-axis” 的 “Line End (PPU)” 栏（底色为白色）。底色为灰色的编辑框不可编辑。

4.3.2.4 运动操作 (Motion Operation)

基本插补运动 (Basic Interpolation Motion)

基本插补运动包含直线插补 (Line)、圆弧插补 (Arc) 和螺旋插补 (Helix)。

Basic Interpolation Motion

Movement Mode: ☐ Absolute ☒ Relative

Interpolation Mode: ☒ Line ☐ Arc ☐ Helix

Arc Direction: ☒ CW ☐ CCW

Buttons: Move, Stop

■ 运动模式

插补运动将直接应用设置后的位置参数。

插补运动将在位置参数中添加初始偏移量，并应用于运动中。

■ 圆弧方向

CW: 顺时针。

CCW: 逆时针。

■ 插补模式

Line: 直线插补

Arc: 圆弧插补

Helix: 螺旋插补

完成相应配置后，点击 [Move] 按钮，群组将执行指定插补操作。当群组执行插补运动时，点击 [Stop] 按钮停止运动。

路径运动

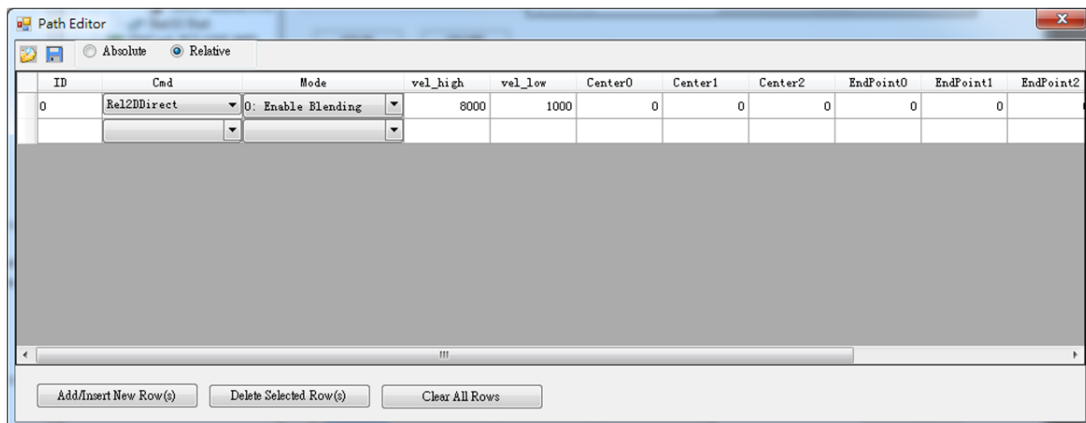
Path Motion

☐ Speed Forward Blending Time: 0 ms

Buttons: Edit Path>>, Load Path>>, Move Path, Move Sel Path>>, Tangent In>>, Tangent Stop, Path Plot>>, Speed Chart>>

■ 编辑路径

点击 [Edit Path]，将显示如下对话框：



– 打开文件

在相应文件路径打开选定的 Path 文件，该文件可以是二进制文件 (.bin) 或逗号分隔值文件 (.CSV)。

– 保存文件

将编辑后的数据存入 Path 文件，该文件可以是二进制文件 (.bin) 或逗号分隔值文件 (.CSV)。CSV 文件可用 Excel 打开，便于后续查看或更改。但是，若用户要通过 [Load Path] 运行 Path 文件，则需将数据保存为 .bin 格式，因为当前设备通过 [Load Path] 导入文件时仅支持 .bin 格式。

– 运动模式 ☐ Absolute ☒ Relative

绝对 (Absolute) 或 (Relative)。若用户选择 “Absolute”，则 Cmd 栏将显示与绝对运动有关的命令；同样，若用户选择 “Relative”，则 Cmd 栏将显示与相对运动有关的命令。

– 路径编辑

ID	Cmd	Mode	vel_high	vel_low	Center0	Center1	Center2	EndPoint0	EndPoint1
0	Rel2DDirect	0: Enable Blending	8000	1000	0	0	0	0	0
1	Rel2DLine	0: Enable Blending	8000	1000	0	0	0	0	0
2	EndPath	0: Enable Blending	8000	1000	0	0	0	0	0

可设置的参数包括：命令 (Cmd)、模式 (Blending/No Blending)、运动速度 (vel_high)、初速度 (vel_low)、中点 (Center) 和端点 (EndPoint)。其中，圆弧插补默认有三个轴 (Center0/Center1/Center2)，EndPoint 的值取决于选定设备插补所支持的最大轴数。以 PCI-1245 为例，Direct 插补运动所支持的最大轴数为 4，因此端点为 EndPoint0、EndPoint1、EndPoint2 和 EndPoint3。

– 添加 / 插入新行

点击此按钮将显示如图对话框：用户可以编辑需要添加 / 插入的行数。点击 [OK]，所选之前 / 之后将添加 / 插入相应数量的行。

– 删除所选行

删除选定行。

– 清楚所有行

清楚所有行的内容。

■ 下载路径

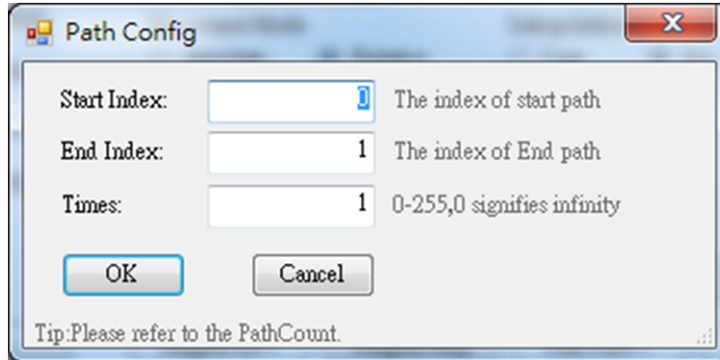
点击 [Load Path]，若群组状态为 “Ready”，则用户可将选中的 Path 文件 (.bin 格式) 从 “Open File” 对话框导入设备。

■ 运行路径

若 Path 文件在载入后未出错，则 Path 将根据序号逐个运行。用户可通过 [Path Status]、[Path Plot] 和 [Speed Chart] 观察运动曲线和相应状态。

■ 运行路径选择

根据在载入 Path 文件中所选的路径执行连续插补运动。载入 Path 后，点击 [Move Sel Path] 将显示如下对话框：



1. Start Index: 选择 Path 的开始序号
2. End Index: 选择 Path 的结束序号
3. Times: 执行时间值为 0-255 如果用户将该值设置为 1，则意味着将进行无限循环，除非用户点击 “Stop” 终止循环。

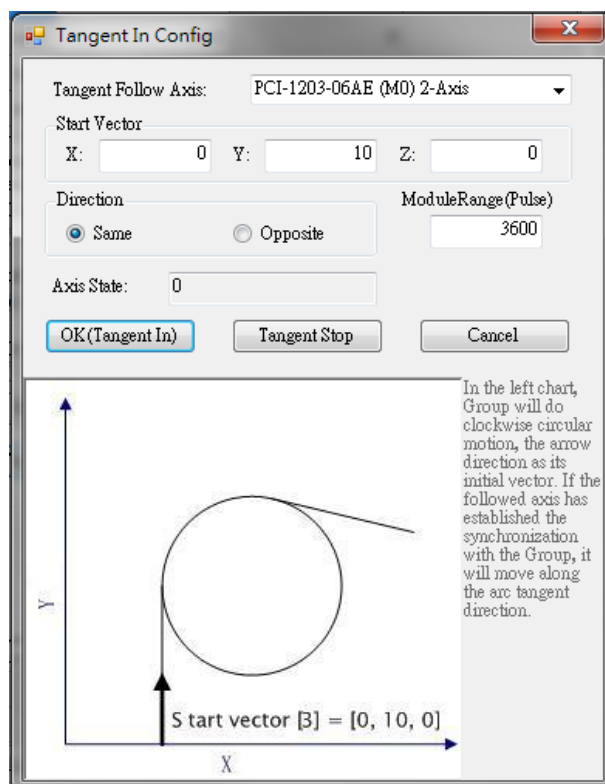
■ 速度前瞻 (Speed Forward) 和交接时间 (Blending Time)

若选中该值，则群组的速度前瞻功能开启。有关详细信息，请参考编程指南属性列表列出的群组中有关 CFG_GpSFEnable 的描述信息。

关于交接时间的详细信息，请参考编程指南属性列表列出的群组中有关 CFG_GpBldTime 的描述信息。

■ 建立和停止切向跟随

单击 [Tangent In] 将出现如下对话框。



用户需配置以下参数：

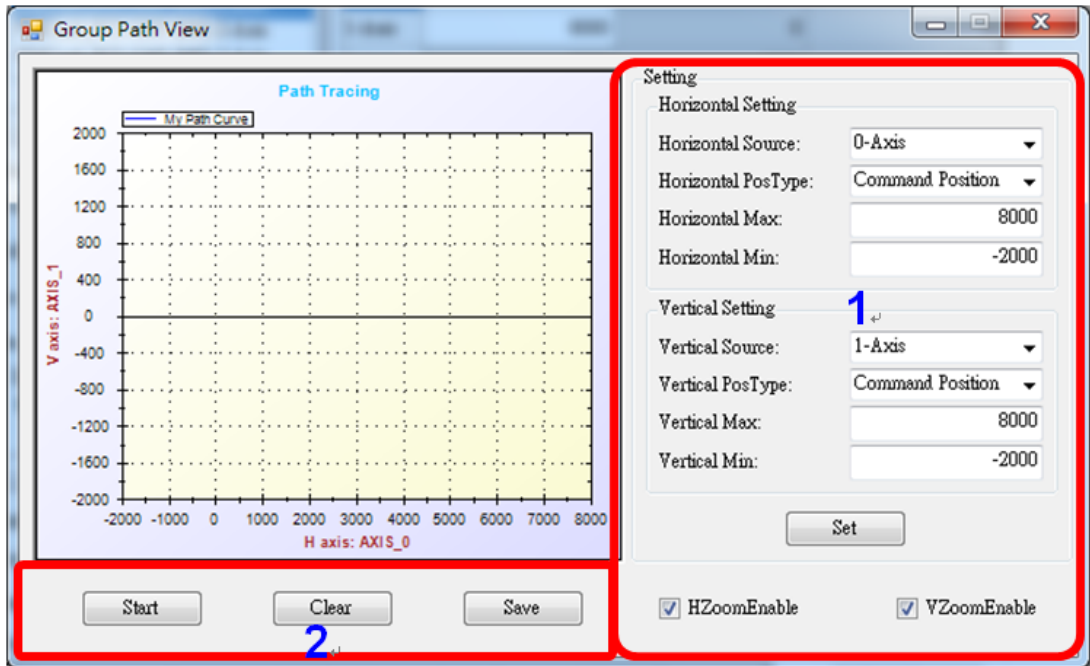
1. **Tangent Follow Axis:** 选择切向跟随轴。由于该轴不能为群组中添加的轴，因此下拉列表框中不包括群组中添加的轴。
2. **Start Vector:** 切向跟随运动的起始向量。在该测试工具中，默认参考面为 X-Y，用户只需要设置 X 向量和 Y 向量，无需编辑 Z 轴。
3. **Direction:** 运动中切向跟随轴的方向，可以和群组运动方向相同或相反。
4. **ModuleRange:** 切向跟随轴的单圈脉冲数范围（即跟随轴旋转一周（360 度）的脉冲数）。

配置下面有相关示意图和描述信息。单击 [OK (Tangent In)]，切向跟随轴将与群组建立切向跟随同步关系。之后，如果群组进行插补运动，跟随轴将沿着插补运动的切向方向运动。如果切向跟随轴已经建立和群组的切向跟随同步运动，再次单击 [Tangent In]，对话框中的参数值将为配置值，用户可单击 [Tangent Stop] 解除同步关系。单击 [Cancel]，将不做任何修改，同时关闭对话框。

单击 [Tangent Stop] 解除切向跟随轴和群组之间的同步关系。

■ 运动曲线 (Path Plot)

显示群组的运动曲线。单击 [Path Plot]，将出现如下对话框：



1. 设置

设置水平和垂直坐标。

- 数据源：水平数据源，默认为 Group 的第一轴（按照添加顺序排序），可以选择 Group 中的任一轴。
- 位置类型：水平位置类型，用户可选择理论或反馈位置。
- 最大值：最大水平坐标。
- 最小值：最小水平坐标。

单击 [Set] 激活生效。

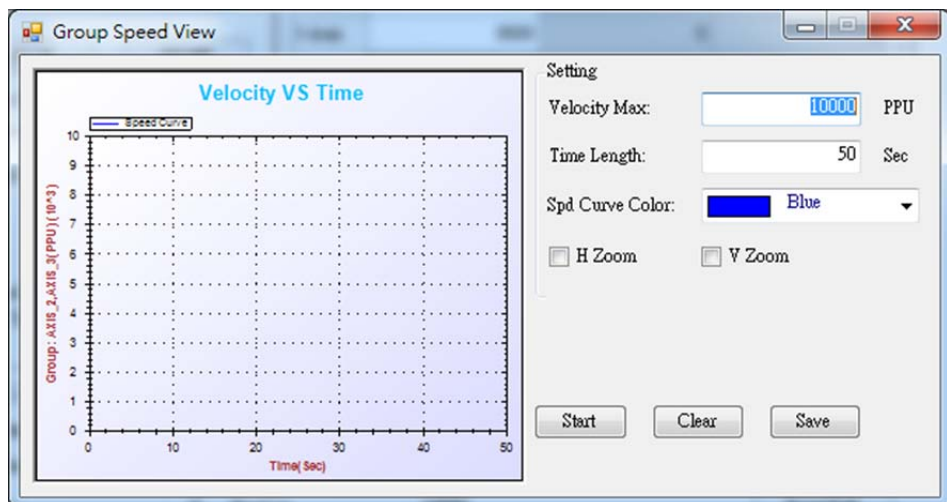
2. 操作

单击 [Start]，图形框即开始绘制曲线。如果群组在运动中，那么用户可以看到运动轨迹。单击之后，[Start] 按钮上的文字将变成“Stop”；单击 [Stop]，曲线图绘制将停止，且文字将变回“Start”。

单击 [Clear] 将清除图形框中当前显示的曲线图。

单击 [Save]，路径的曲线图将存为 .png、.gif、.jpg、.tif 或 .bmp 格式。

■ 速度曲线 (Speed Chart)



设置和操作与“Single Axis Motion”中的 [Speed Chart] 类似。

4.3.2.5 路径状态 (Path Status)

Path Status			
CurIndex:	0	CurCmd:	EndPath
Path Count:	0		
Remain:	0	FreeCnt:	7000
			Reset Path

当前序号：正在执行的路径对应的当前索引。

当前命令：正在执行的路径的理论代码。

未执行数：未执行的路径个数。

空余数：Path Buffer 中的可用空间。

路径数：载入的 Path 文件中的路径总个数。

4.3.2.6 轴位置 (Axis Position)

Position			
Position	0-Axis	1-Axis	2-Axis
Command	0	0	0
Feedback	0	0	0
Reset Error			
Reset Counter			

显示设备所有轴的当前理论和反馈位置。单击 [Reset Counter] 重置为 0。

4.3.2.7 群组状态 (Group State) 和错误状态 (Error Status)

Group State:	Ready	
Last Error Status		
Error Code:	0	Error Message: Success.

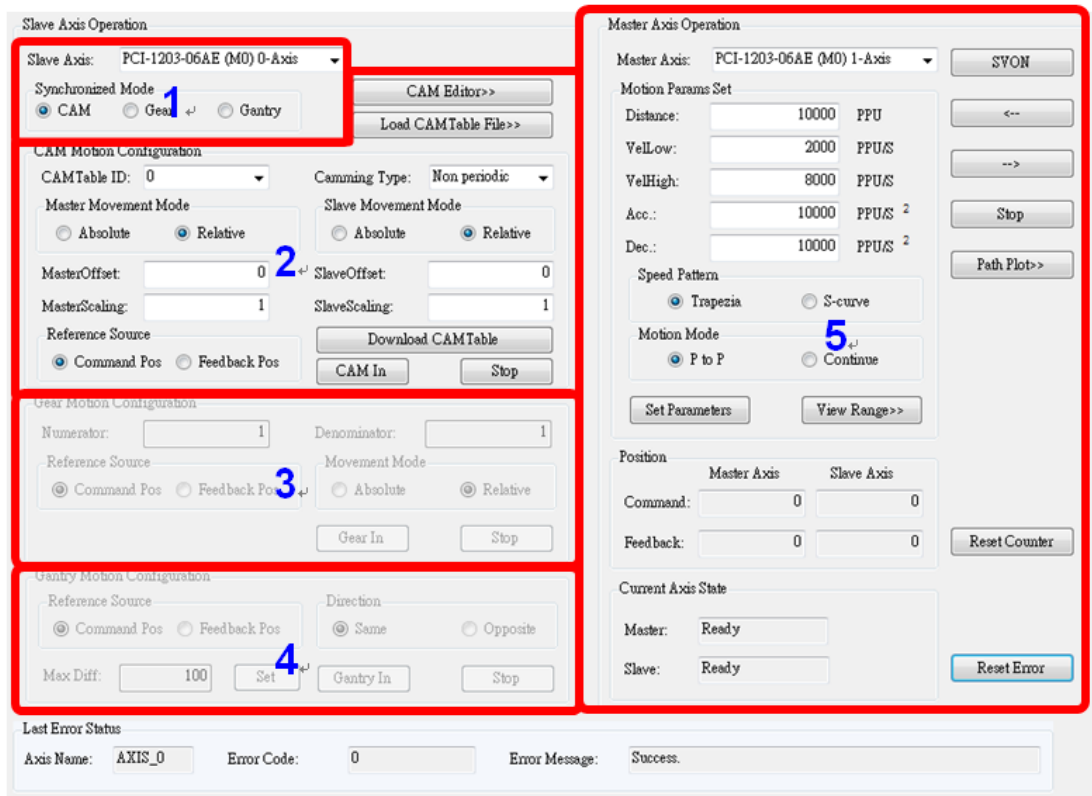
Group 状态：显示当前群组的状态。更多详细信息，请参考通用 API 编程指南中对于 Acm_GpGetState 函数的 State 的描述。

最新错误状态：显示最新错误信息。

错误代码：错误代码。

错误信息：指定错误信息。

4.3.3 同步运动 (Synchronized Motion)



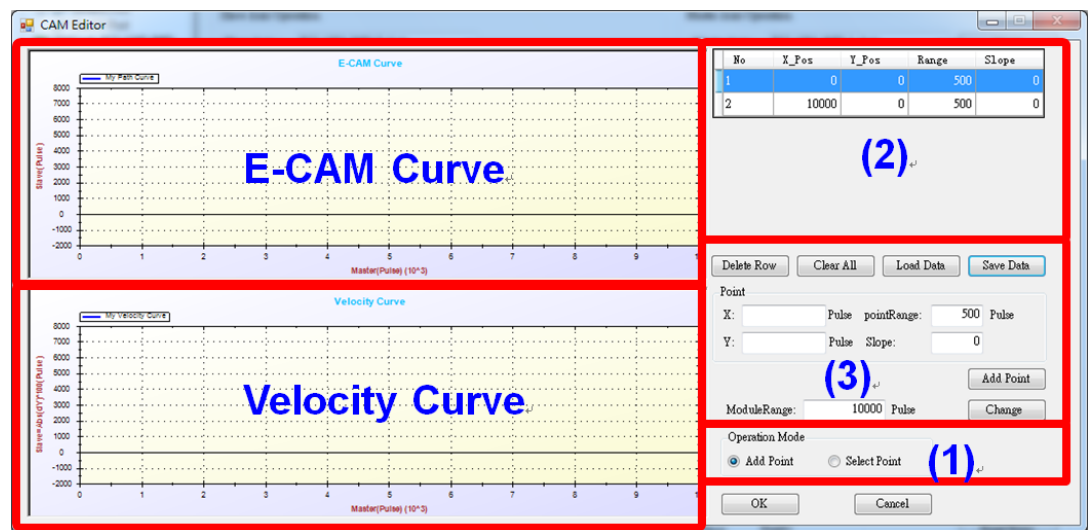
4.3.3.1 选择轴 (Axis) 和同步模式 (Synchronized Mode)

选择设备的其中一个轴作为从轴。主轴和从轴不能为同一轴。默认的从轴为所选设备的 0 轴。

选择同步模式：CAM（电子凸轮）、Gear（电子齿轮）或 Gantry（龙门）。在配置和建立同步运动之前，用户必须先选择同步模式。

4.3.3.2 电子凸轮运动 (CAM)

单击 [CAM Editor] 将出现如下对话框：



左上角为 E-CAM 曲线图框；左下角为 Velocity 曲线图框；右上角为 CAM 表；右下角为操作面板 (Operation Panel)。

1. 操作面板

- a. Add Point: 用户可以在 E-CAM Curve 上直接添加 CAM 点。每当用户添加一个点, CAM 曲线都会重绘。其中, CAM Point 以一个红色实心小圆点表示, CAM Curve 以蓝色曲线表示。在此操作模式中, 鼠标光标为一个十字。初次打开该界面或尚未编辑 CAM 曲线时, 默认操作模式为 “Add Point”。
- b. Select Point: 用户可选择相应的 CAM 点进行拖放。同时, CAM 曲线将发生相应改变。在此操作模式中, 鼠标光标为箭头。当再次打开该界面或已经编辑 CAM 曲线, 默认操作模式为 “Select Point”。

2. CAM 表

当在 CAM 曲线上添加 CAM 点时, 对应点的坐标将显示在 CAM 表中。CAM 表中第一行的 X_Pos 和 Y_Pos, 即首个 CAM 点, 必须为 0, 这表示主轴的起始位置为 0, 从轴的起始位置为 0。CAM 表中最后一行的 X_Pos 和 Y_Pos, 即最后一个 CAM 点, 必须为 (ModuleRange, 0), 这表示主轴旋转了一圈, 从轴返回至起始位置 0。

- a. No: CAM 点的序号。
- b. X_Pos: 水平位置坐标 (主轴位置)。
- c. Y_Pos: 垂直位置坐标 (从轴位置)。
- d. Range: 参考点和 CAM 点之间的距离。有关详细信息, 请参考通用 API 编程指南对 Acm_DevDownloadCAMTable 函数中的 PointRange 的说明信息。默认值为 pointRange 的编辑值。用户可通过编辑改变该值。当用户添加多个 CAM 点时, pointRange 也将随之改变。如果用户想要改变已编辑 CAM 点的 pointRange, 可直接在 CAM 表中进行修改。
- e. Slope: CAM 点的两个参考点之间的斜率。有关详细信息, 请参考通用 API 编程指南对 Acm_DevDownloadCAMTable 函数中的 PointSlope 的说明信息。默认值为 “Slope” 编辑框中的值, 用户可通过编辑修改该值。后续添加的 CAM 点的斜率将为编辑框中修改后的值。如果用户想要改变已编辑 CAM 点的 Slope, 可直接在 CAM 表中进行修改。

注! Slope 范围为 $-10 \sim 10$ 。如果值小于 -10 , 则默认为 -10 。如果值大于 10 , 则默认为 10 。



3. CAM 表操作

- a. Delete Row: 删除所选行。
- b. Clear All: 清除所有 CAM 点 (除起始点和终结点)。
- c. Load Data: 插入所选 CAM 表文件。文件格式可以为二进制文件 (.bin) 或 Excel 可读的 .csv 文件。
- d. Save Data: 保存 CAM 表。文件格式可以为二进制文件 (.bin) 或 Excel 可读的 .csv 文件。但是如果用户想要通过 [Load CAMTable File] 操作导入 CAM 表, 则需要将 CAM 表保存为 .bin 格式, 因为目前设备仅支持通过 [Load CAMTable File] 导入 bin 文件。
- e. Add Point 功能按钮: 添加 CAM 点时, 用户也可以在右下角编辑 X_Pos、Y_Pos、pointRange 和 Slope, 然后单击 [Add Point] 即可。
- f. 改变 Module Range: 框中进行编辑, 然后单击 [Change] 完成。修改之后, E-CAM 曲线和速度曲线的水平最大坐标将为修改值。如果修改值之前编辑过 CAM 点, 则 CAM 点的 X_Pos 和 pointRange 将变为 ModuleRange (修改后) / Pre_ModuleRange (修改前) 倍。

单击 [OK] 保存 CAM 表。用户可在同步运动 (Synchronized Motion) 页面点击 [Download CAMTable] 将 CAM 表将入硬件。单击 [Cancel] 放弃编辑。

导入 CAM 表文件 (Load CAMTable File) 功能按钮

单击 [Load CAMTable File] 选择二进制文件 .bin 文件，CAM 表将导入硬盘。

注！ 保存之前，用户必须首先设置 “CAMTableID”。值为 0 或 1。完成该步后，在解除同步关系之前 CAMTableID 将不能修改。



CAM 表配置

配置 CAM 运动并建立 CAM 同步。

The screenshot shows the 'CAM Motion Configuration' dialog box. It has two columns of settings. The left column includes 'CAMTable ID' (set to 0), 'Master Movement Mode' (with 'Relative' selected), 'MasterOffset' (0), 'MasterScaling' (1), and 'Reference Source' (with 'Command Pos' selected). The right column includes 'Camming Type' (set to 'Non periodic'), 'Slave Movement Mode' (with 'Relative' selected), 'SlaveOffset' (0), 'SlaveScaling' (1), and a 'Download CAMTable' button. At the bottom right are 'CAM In' and 'Stop' buttons.

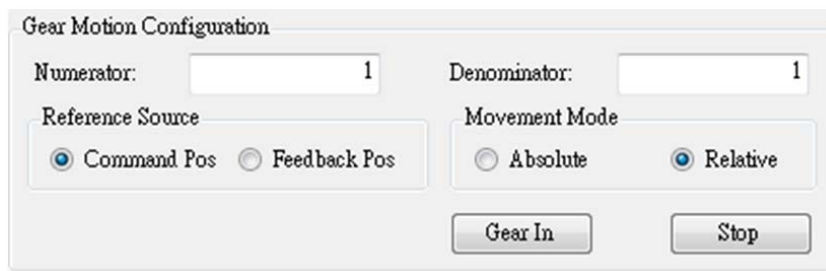
在建立 CAM 同步之前，用户需配置下列参数：

1. 凸轮运动类型 (Camming Type):
 - a. Non periodic: 非周期模式。如果用户选择这种模式，当主轴运行完一个完整周期之后，从轴将不会按照 CAM 曲线跟随主轴运动。
 - b. Periodic: 周期模式。如果用户选择这种模式，从轴将始终按照 CAM 运动中 CAM 曲线跟随主轴运动。
2. 主轴运动模式 (Master Movement Mode)
 - a. Absolute: 若用户选择该模式，主轴的当前位置将被 CAM 曲线作为水平坐标的起始点。
 - b. Relative: 若用户选择该模式，主轴将以当前命令 / 实际位置作为起始点进行相对运动。
3. 从轴运动模式 (Slave Movement Mode)
 - a. Absolute: 若用户选择该模式，从轴将从当前的理论 / 实际位置以设定的速度参数去追赶 CamTable 中对应的 Y_Pos 值。
 - b. Relative: 若用户选择该模式，从轴将以当前的理论 / 实际位置按照 CAM 曲线跟随主轴做相对运动。
4. Master Offset: 相对于主轴的偏移值。
5. Slave Offset: 相对于从轴的偏移值。
6. Master Scaling: 主轴比率因数。CAM 曲线在水平方向的缩放。
7. Slave Scaling: 从轴比率因数。CAM 曲线在垂直方向的缩放。
8. Reference Source: 主轴的位置参考源。
 - a. Command Position: 参考源为理论位置。
 - b. Feedback Position: 参考源为反馈（实际）位置。

配置完如上参数后，点击 ”CAM IN” 功能按钮从轴将与主轴建立凸轮同步关系，从轴的状态将变为 “SynchronousDriving”。此后，如果主轴为 “P to P” 或 “Continue” 运动，从轴将按照 CAM 曲线和配置跟随主轴运动。

单击 [Stop] 解除同步关系。从轴的状态将恢复为 Ready。

4.3.3.3 齿轮运动 (Gear Motion)



The image shows a 'Gear Motion Configuration' dialog box. It has two input fields at the top: 'Numerator:' with a value of '1' and 'Denominator:' with a value of '1'. Below these are two sections: 'Reference Source' with radio buttons for 'Command Pos' (selected) and 'Feedback Pos'; and 'Movement Mode' with radio buttons for 'Absolute' and 'Relative' (selected). At the bottom right are two buttons: 'Gear In' and 'Stop'.

建立齿轮同步之前，用户必须配置以下参数：

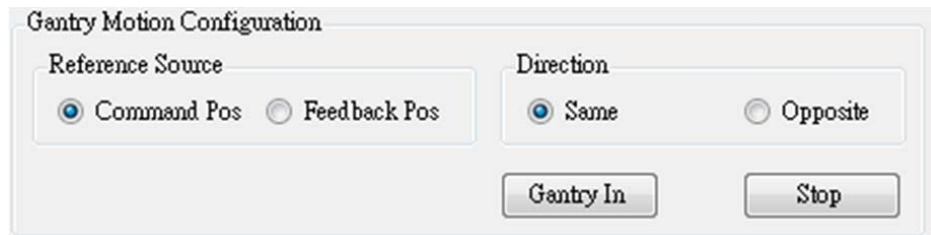
1. Numerator: 齿轮比的分子。
2. Denominator: 齿轮比的分母。
3. Reference Source: 主轴的位置参考源：理论位置或反馈位置。
4. Movement Mode:
 - Absolute: 若用户选择该模式，从轴将以设定的速度参数追赶主轴的理论 / 实际位置，直至两者一致。
 - Relative: 若用户选择该模式，从轴将与主轴保持初始时的速度差。

单击 [Gear In]，从轴将与主轴建立齿轮同步关系，从轴的状态将变为 “Synchronous Driving”。此后，如果主轴为 “P to P” 或 “Continue” 运动，从轴将按照配置跟随主轴运动。

单击 [Stop] 解除同步关系，从轴状态将为 “Ready”。

4.3.3.4 龙门运动 (Gantry Motion)

在同步模式 (Synchronized Mode) 中选择龙门 (Gantry)，用户即可配置和操作齿轮运动。



建立龙门同步之前，用户必须配置以下参数：

1. Reference Source: 主轴的位置参考源：理论位置或反馈位置。
2. Direction: 从轴相对于主轴的运动方向。
 - a. Same: 与主轴相同。
 - b. Opposite: 与主轴相反。

单击 [Gantry In]，从轴将与主轴建立龙门同步关系，从轴的状态将变为“Synchronous Driving”。此后，如果主轴为“P to P”或“Continue”运动，从轴将按照配置跟随主轴运动。

单击 [Stop] 解除同步关系，从轴状态将为“Ready”。

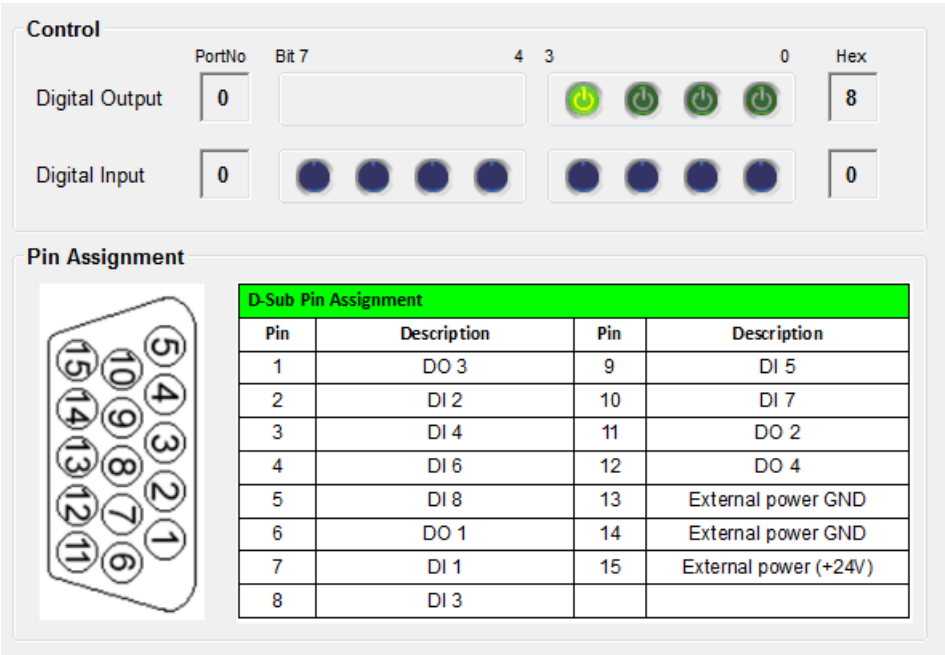
4.3.3.5 主轴操作及同步运动测试

完成从轴设置之后，用户可以通过主轴操作面板来测试同步运动。该操作类似于单轴运动。选择主轴并在“Motion Params Set”面板上设置运动参数。单击[Set Command]按钮应用设置。单击[SVON]按钮开启伺服；点击[<-]或[->]按钮启动运动。当主轴开始运动时，从轴也将根据同步模式的设置开始自动运动。

用户可以在位置面板查看理论 / 反馈位置，还可通过状态栏查看株洲和从轴的当前状态。更多详细信息，请参考通用 API 编程指南中的 Acm_AxGetState 函数中对 State 的说明信息。

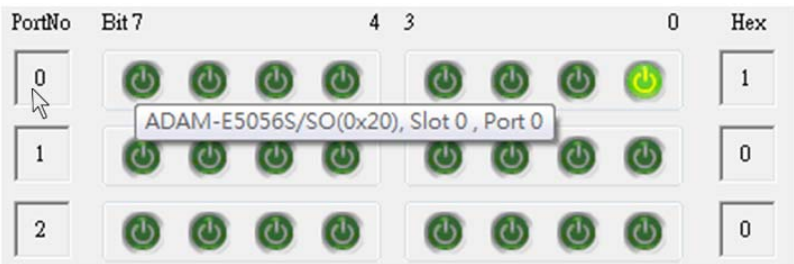
4. 3. 4 DO / DI / AO / AI

4. 3. 4. 1 本地 I/O



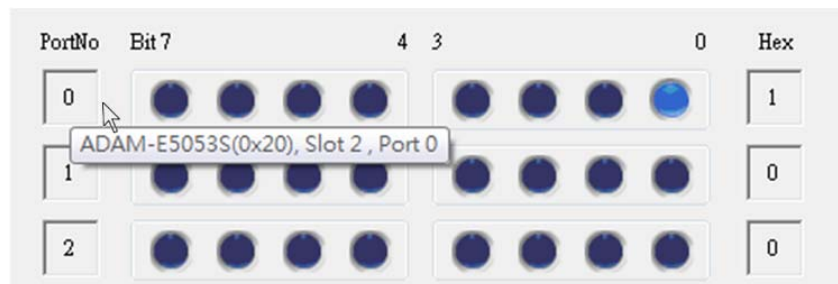
该页以 D-Sub 15 针接口显示本地 DI/O 状态。● 或 ● 表示 DI 状态为活跃 (ON) 且该位值为 1；● 或 ● 表示 DI 状态为禁用 (OFF) 且该位值为 0。Hex 表示由 8 个 IO 组成的字节的十六进制值。用户点击 ● 或 ● 按钮可以开启 / 关闭数字量输出通道。该页还显示了 D-Sub 15 针接口的针脚分配。请注意，针脚定义从 1 开始，位定义从 0 开始。例如：DO 的 Bit 0 是针脚分配中的 DO 1，而 DO 的 Bit 1 是针脚分配中的 DO 2，以此类推。PCI-1203 共有 4 路数字量输出 (DO 1 - DO 4)，因此，DO 的最高位编号为 Bit 3。

4. 3. 4. 2 数字量输出



显示连接至 PCI-1203 的数字量输出端口的状态。如上图所示，从右到左分别为数字量输出 Bit0 至 Bit7。其中，● 表示 DO 有效 (On)，bit 的值为 1；● 表示 DO 无效 (Off)，bit 的值为 0。“Hex”表示 8 个 DO 组成的字节的十六进制值。PortNo 为端口编号。若要获取实际 DO 位置，只需将鼠标光标移到 PortNo 区域。此时轻点鼠标即可显示相应端口的实际位置。本例中，Port 0 位于 ADAM-E5056S/SO 模块，从轴 ID 为 0x20，插槽编号为 0。

4.3.4.3 数字量输入



显示连接至 PCI-1203 的数字量输入端口的状态。如上图所示，从右到左分别为数字量输入 Bit0 至 Bit7。其中，● 表示 DI 有效 (On)，bit 的值为 1；● 表示 DI 无效 (Off)，bit 的值为 0。“Hex” 表示 8 个 DI 组成的字节的十六进制值。

PortNo 为端口编号。若要获取实际 DI 位置，只需将鼠标光标移到 PortNo 区域。此时轻点鼠标即可显示相应端口的实际位置。本例中，Port 0 位于 ADAM-E5053S 模块，从轴 ID 为 0x20，插槽编号为 0。

4.3.4.4 模拟量输出

Channel	Value	Range	Raw
CH-0	0.000	-10 ~ 10V	0x0
CH-1	0.000	-10 ~ 10V	0x0
CH-2	0.000	-10 ~ 10V	0x0
CH-3	0.000	-10 ~ 10V	0x0

显示连接至 PCI-1203 的模拟量输出通道的状态。首次打开该页面时，扫描每个从轴的 AO 设置会花费一定时间。该值为该通道的模拟量值，其单位由 AO 范围决定，可能为 mA、mV 或 V。Raw 为该通道的 16 位原始数据。本例中，Channel 0 位于 ADAM-E5024H 模块，从轴 ID 为 0x20，插槽编号为 3，值为 0V。

4.3.4.5 模拟量输入

Channel	Value	Range	Raw
CH-0	0.007	-10 ~ 10V	0x801
CH-1	0.007	-10 ~ 10V	0x801
CH-2	*****		*****
CH-3	*****		*****

显示连接至 PCI-1203 的模拟量输入通道的状态。首次打开该页面时，扫描每个从轴的 AI 设置会花费一定时间。该值为该通道的模拟量值，其单位由 AI 范围决定，可能为 mA、mV 或 V。Raw 为该通道的 16 位原始数据。本例中，Channel 0 位于 ADAM-E5017UH 模块，从轴 ID 为 0x20，插槽编号为 1，值为 0.007V。

4.3.5 IO 映射表

该页显示的是物理和逻辑索引映射表。

DI

DO

AI

AO

	Port	Ring	ID	Slot	Description	Edit
	0	0	0x20	2	Port 0 [ADAM-E5053S]	Select
	1	0	0x20	2	Port 1 [ADAM-E5053S]	Select
	2	0	0x20	2	Port 2 [ADAM-E5053S]	Select
	3	0	0x20	2	Port 3 [ADAM-E5053S]	Select
	4	0	0x200	1	Port 0 [ADAM-E5051S]	Select
	5	0	0x200	1	Port 1 [ADAM-E5051S]	Select
	6	0	0x0		Port 0 [MADHT1505BA1]	Select
	7	0	0x0		Port 1 [MADHT1505BA1]	Select
	8	0	0x0		Port 2 [MADHT1505BA1]	Select
	9	0	0x0		Port 3 [MADHT1505BA1]	Select
	10	0	0x102		Port 0 [MADHT1505BA1]	Select
	11	0	0x102		Port 1 [MADHT1505BA1]	Select
	12	0	0x102		Port 2 [MADHT1505BA1]	Select
	13	0	0x102		Port 3 [MADHT1505BA1]	Select
	14	0	0x103		Port 0 [MADHT1505BA1]	Select
	15	0	0x103		Port 1 [MADHT1505BA1]	Select
	16	0	0x103		Port 2 [MADHT1505BA1]	Select
	17	0	0x103		Port 3 [MADHT1505BA1]	Select

Find

Port

Slave ID

Find Next

Start Offset : 0 Set

Edit Port (0 ~ 19) : 0 Select

Undo all changesExport CSVLoadSave AsSave

Filename: Default Mapping Settings

对于通用运动 API 而言，用户需要通过一个指定数字从 / 向从轴读 / 写数值。我们将这个指定的数字称为逻辑索引或逻辑端口 / 通道。

在该页中，用户需要指定逻辑索引，并将指定的映射设置导出至文件。将匹配文件导入通用运动 API 之后，用户可以通过逻辑索引操作这些端口和通道。

4.3.5.1 如何读取映射表

DI

DO

AI

AO

	Port	Ring	ID	Slot	Description	Edit
	0	0	0x20	2	Port 1 [ADAM-E5053S]	Select
	1	0	0x20	2	Port 0 [ADAM-E5053S]	Select
	2	0	0x20	2	Port 2 [ADAM-E5053S]	Select
	3	0	0x20	2	Port 3 [ADAM-E5053S]	Select
	4	0	0x200	1	Port 0 [ADAM-E5051S]	Select

上图为DI映射表示例。第一栏“Port”为逻辑端口编号。“Ring”、“ID”、“Slot”和“Description”栏显示端口的实际位置。如该表的第一行所示，逻辑端口 0 的实际位置信息包括：Ring0 的 Port1，从轴 ID 为 0x20，插槽编号为 2，位于 ADAM-E5053S 模块。

4.3.5.2 编辑映射表

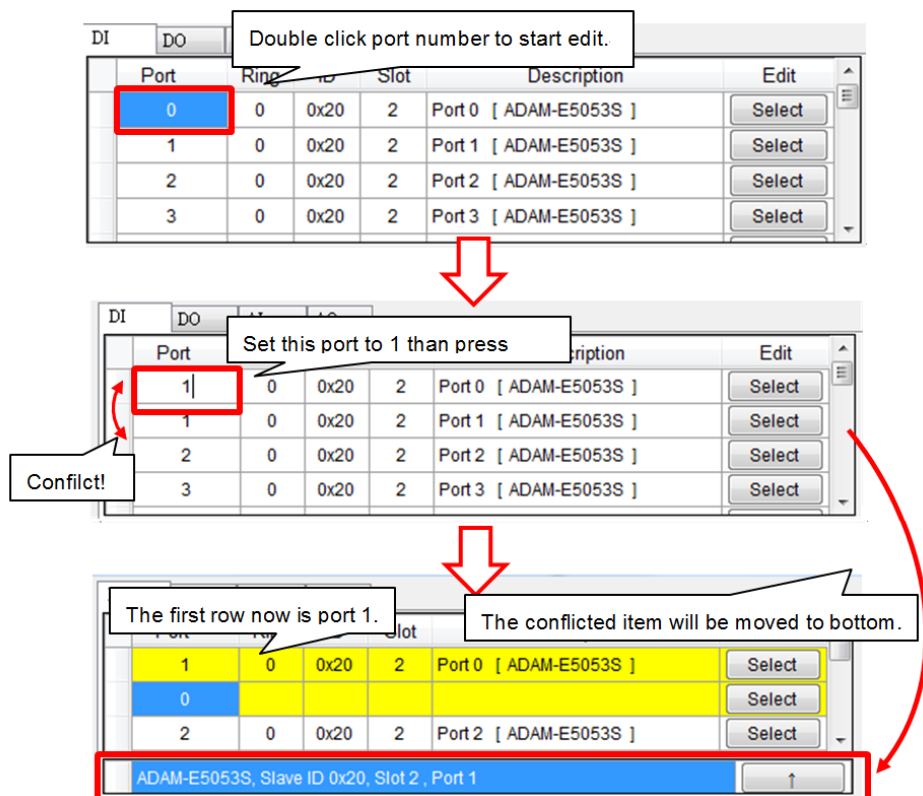
用户可以通过编辑映射表来指定逻辑端口。编辑映射表时需要遵守以下两条规则：第一，需要为所有实际端口 / 通道赋予一个专有逻辑索引。不允许存在未经映射的端口 / 通道。不允许两个实际端口共享同一个逻辑索引。请注意，映射表的 DI/DO/AI/AO 均单独列出，因此 DI、DO、AI 和 AO 都可以拥有自己的逻辑端口 0，而不会发生冲突。第二，逻辑索引需要是连续的。

在下节内容中，我们将描述如何设置映射表。

编辑逻辑索引

用户双击逻辑索引即可进行编辑。若该索引已存在，那么冲突项将被高亮显示，并被移至表格底部。

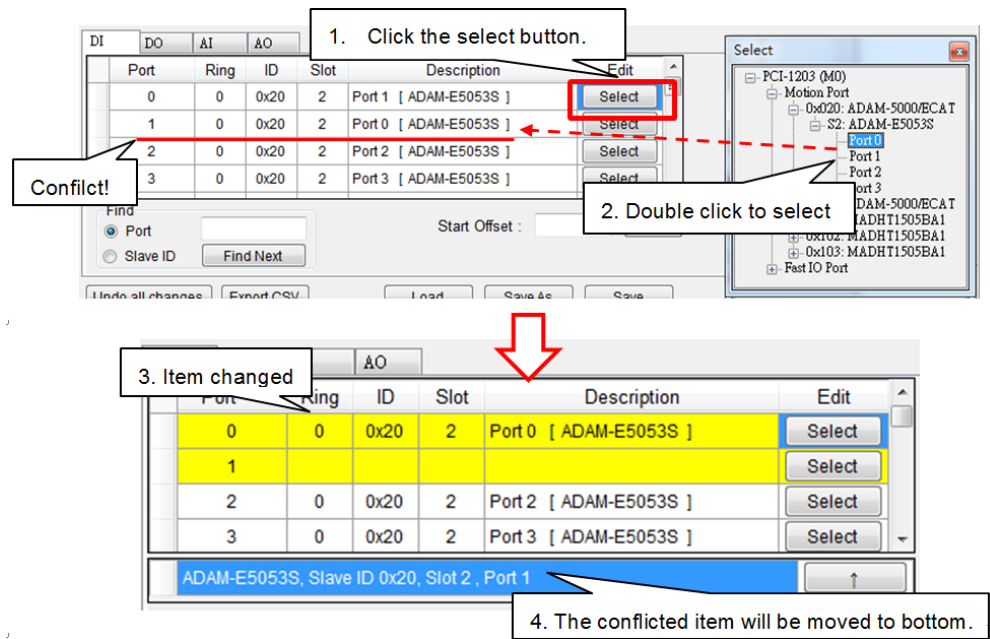
请参考下图中示例：



编辑实际位置

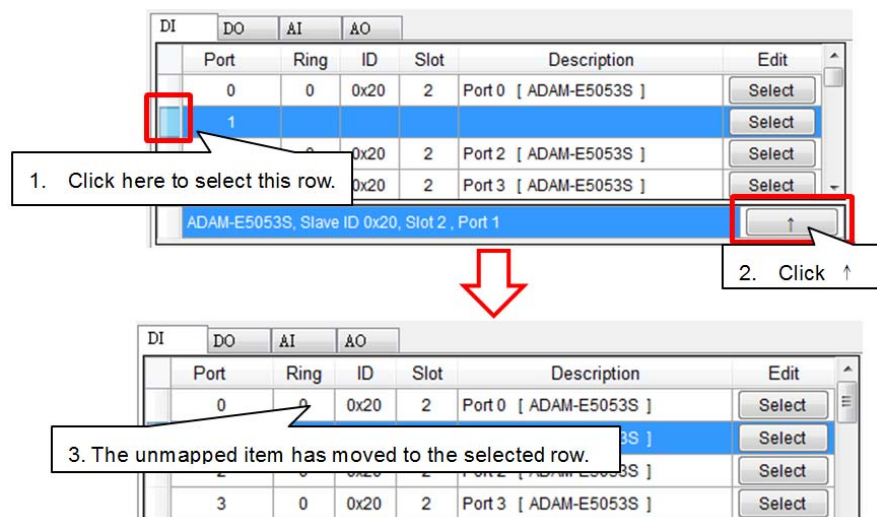
用户可以双击 [Select] 按钮改变逻辑端口的实际位置。单击 [Select] 按钮会跳出一个选择工具。若选取的 port 已经映射至一个逻辑索引，冲突项将被高亮显示，并移至表格底部。

请参考如下示例：

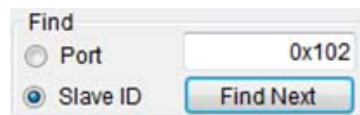


映射尚未映射项

请参考如下示例：



4.3.5.3 查找映射表资料



用户可以使用 [Find] 面板查找目标项。在文本框中键入从轴 ID（包括“0x”）或逻辑索引，点击 [Find Next] 按钮进行查找。

4.3.5.4 初始偏移量

DI	DO	AI	AO	
Channel	Ring	ID	Slot	Description
11	0	0x20	3	Channel 0 [ADAM-E5024H]
12	0	0x20	3	Channel 1 [ADAM-E5024H]
13	0	0x20	3	Channel 2 [ADAM-E5024H]
14	0	0x20	3	Channel 3 [ADAM-E5024H]
15	0	0x200	2	Channel 0 [ADAM-E5024]
16	0	0x200	2	Channel 1 [ADAM-E5024]
17	0	0x200	2	Channel 2 [ADAM-E5024]
18	0	0x200	2	Channel 3 [ADAM-E5024]

Find
☒ Channel

映射的索引需要是持续的，但索引范围可以定制。例如，EtherCAT 网络中总共有 8 个 AO 通道，初始偏移量为 11。有效逻辑通道范围为 11~18。

键入初始偏移量，单击 [Set] 按钮应用设置。

4.3.5.5 保存与加载

Filename: Default Mapping Settings

单击 [Save] 或 [Save As] 按钮保存映射表。单击 [Load] 按钮将映射表载入 Utility。若该映射表有效，将自动应用在 Utility 中。

[Filename] 将显示当前应用于 Utility 中的映射表。若用户打开 Utility 后未加载或编辑映射表，则“Filename”将为“Default Mapping Settings”，即系统默认映射设置。

单击 [Undo all changes] 撤销所有更改并重新加载映射表。

单击 [Export CSV] 将映射表输出至 .CSV 文件。用户可以使用 Excel 或任意支持 .CSV 格式的工具来查看或打印映射表。

4. 3. 6 信息

Device Information

Device Name

PCI-1203-06AE (M0)

Description

Advantech EtherCAT Master Card

Firmware Version

1.0.0.9.0

FPGA Version

a.0.0.2.β

Kernel Driver Version

1.0.0.4

User Driver Version

2.0.0.5

System Log

☐ Enable

EtherCAT

Master (R0)

Topology

Master Name

Motion Ring

Cycle Time

500μs

Connected Slave Count

5

Description

Support EtherCAT servo motor and ADAM-5000/ECAT.

Master (R1)

Topology

Master Name

Fast IO Ring

Cycle Time

200μs

Connected Slave Count

0

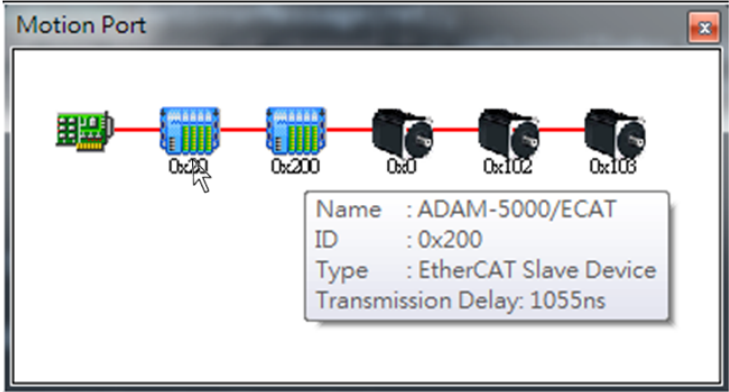
Description

Support ADAM-5000/ECAT.
(Servo motor not support)

此页显示卡的信息和主轴的信息。若 PCI-1203 未连接任何从轴，则此页将为用户能够操作的唯一页面。此页可供用户查看固件和驱动版本。

请注意，PCI-1203 支持 2 个 EtherCAT 环网：Motion Ring 和 Fast IO Ring。Fast IO Ring 不支持伺服电机。当连接至 PCI-1203 时，伺服电机可能被忽视。

单击 [Topology] 按钮，将显示如下窗口。用户可在此页查看网络连接。若 EtherCAT 连接有误，例如，若用户将输入电缆连接至输出端口，则从轴将出现 ⚠ 图标来提示问题。



单击从轴图标将显示相应操作页。

4.4 从站页面

单击设备树中的从站节点。相应的操作页将显示在操作面板中。

Single-Axis Motion | Motion IO | Information

Operate Axis: PCI-1203-06AE (M0) 0-Axis | SVON | Device Number: 0x62000000

Motion Params Set

Distance: 10000 PPU
 VelLow: 2000 PPU/S
 VelHigh: 8000 PPU/S
 Acc.: 10000 PPU/S²
 Dec.: 10000 PPU/S²
 New Pos: 3000 PPU
 New Vel: 2000 PPU/S

Speed Pattern

☒ Trapezia ☐ S-curve | View Range>>

Motion Mode

☒ P to P ☐ Continue

Set Parameters | Speed Chart>>

Configuration

Home Mode>> | External Drive>>
 Axis Setup>> | Axis Status>>

Move Test

<-- | -->
 Move Impose | Stop
 PTP Back-and-Forth>>

Position

Command: 0 | Reset Error
 Feedback: 0 | Reset Counter

Current Axis Status

Current State: Ready
 Command Velocity: 0

I/O Status

RDY ☒
 ALM ☒
 LMT+ ☒
 LMT- ☒
 ORG ☒
 DIR ☒
 EMG ☒
 PCS ☒
 ERC ☒
 EZ ☒
 CLR ☒
 LTC ☒
 SD ☒
 INP ☒
 SVON ☒
 RALM ☒
 SLMT+ ☒
 SLMT- ☒
 CMP ☒
 CAM-DO ☒
 TORLMT ☒

Last Error Status

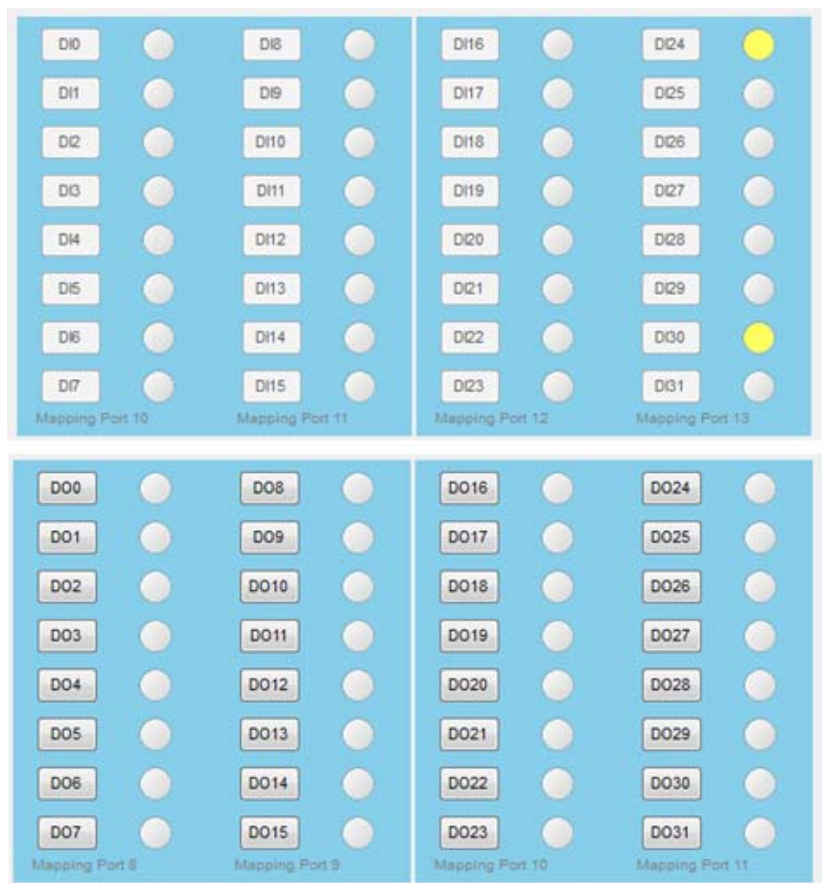
Error Code: 0 | Error Message: Success.

4.4.1 单轴运动控制 (Single-Axis Motion)

若所选从站为伺服电机，则单轴运动页将显示并可供选择。此页中的操作与章节 4.3.1 中描述的页面操作相同。

4.4.2 数字量输入 (Digital Input) 与数字量输出 (Digital Output)

若所选从站支持数字量输入或数字量输出，则如下页面将显示并可供选择。



● 表示信号有效 (ON) 且值为 1。● 表示 DO 信号无效 (OFF) 且值为 0。 Mapping Port 表示映射端口号。用户可以单击 DO 按钮以启动 / 关闭输出信号。

4. 4. 3 运动 IO (Motion IO)

Apply

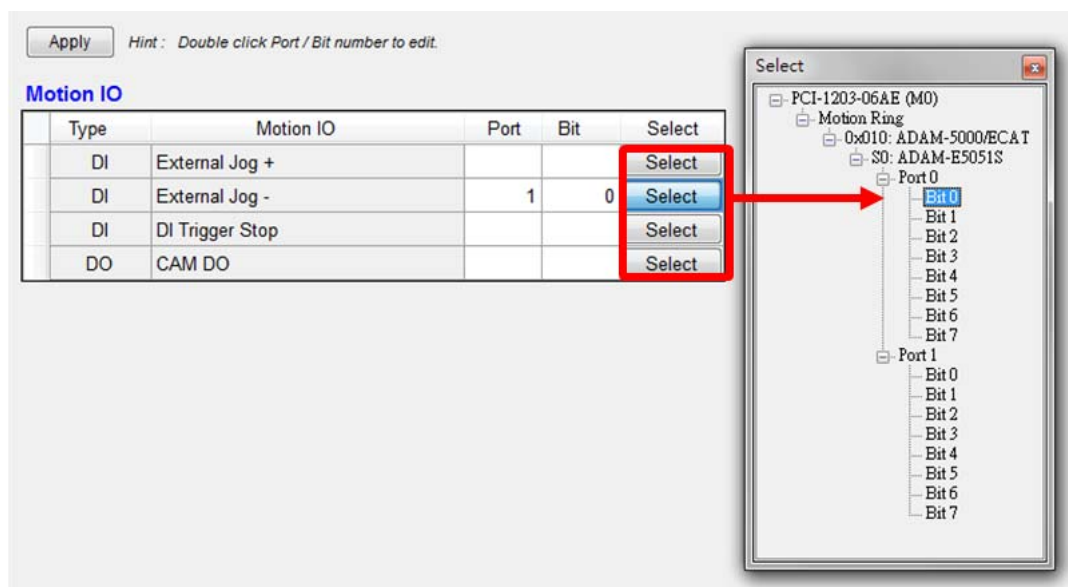
Hint : Double click Port / Bit number to edit.

Click this button to apply the settings.

Motion IO

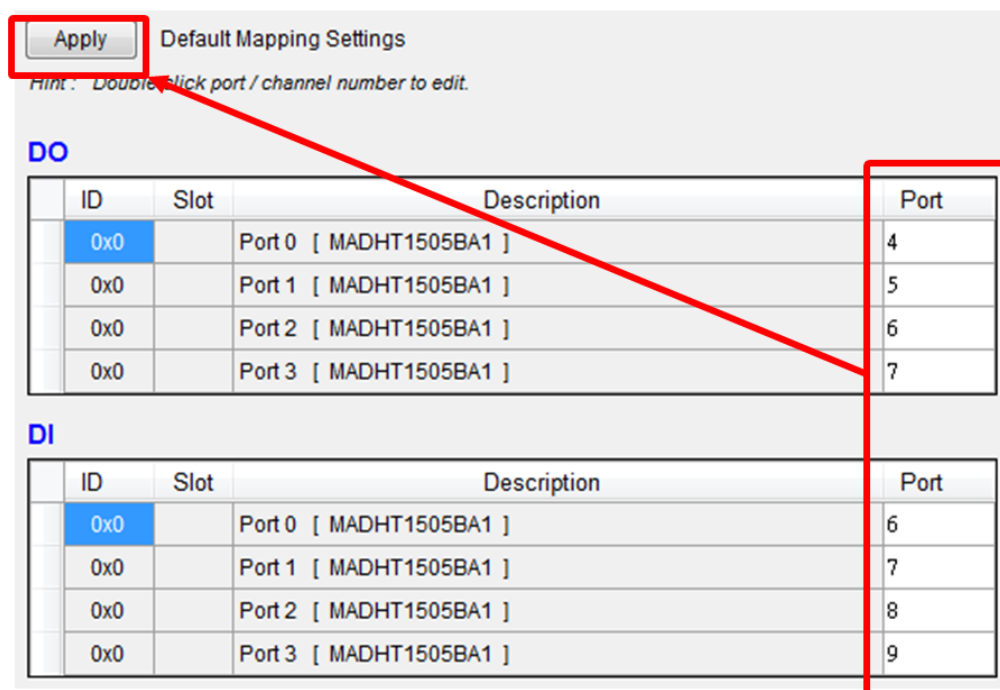
Type	Motion IO	Port	Bit	Select
DI	External Jog +			Select
DI	External Jog -	0	0	Select
DI	DI Trigger Stop	Double click to edit.		Select
DO	CAM DO			Select

此页用于将 IO 触发运动功能映射至外部 IO 端口。PCI-1203 支持外部 DI-jog，外部 DI-stop 和凸轮区间 DO (CAM DO)。编辑映射信息的方式有两种：双击文本框编辑映射位置，或单击 [Select] 按钮显示设备树窗口进行选择。



如上图所示，轴的外部 DI-jog-trigger stop 功能映射至 DI port 1 的 bit 0。该 DI 的实际位置由映射表定义。若映射信息更改，实际触发通道也可能改变。强烈建议用户在使用该功能之前查看映射表，从而避免冲突。完成所有设置之后，单击 [Apply] 将设置下载至运动卡。

4.4.4 IO 列表



此页显示从站的所有 IO。端口栏为相应的逻辑端口。双击端口文本框即可编辑映射端口。完成所有设置之后，单击 [Apply] 保存映射表，将设置应用至运动卡。

注！



我们建议客户使用 PCI-1203 节点中的映射表页面来编辑 IO 映射并进行最终检查，此页中的映射信息更为详尽。

4. 4. 5 从站信息

Slave Information

Vendor

Advantech (0x13FE)

Device Name

ADAM-5000/ECAT

Device Type

EtherCAT Slave Device

ProductCode

0x00015530

Revision No

0x00000000

Serial No

0x00000000

ID No

0x 200

Change ID

Link

Port A:

Link up

Port B:

Link up

Firmware

Current Firmware:

V0352_000

Firmware

Open

Password

0x00000000

Upgrade

此页显示经销商名称、经销商 ID、设备名称、设备类型、产品代码、版本号、序列号和从轴 ID 等从轴信息，这些信息都在 EEPROM 和相应 ESI 文件中定义。若 ESI 文件中对经销商商标进行了详细描述，则该商标可能会显示在经销商名称之前。用户还可检查从轴的固件版本以及升级固件。

ID No

Alias ID 由 EEPROM 定义，并可手动修改。对于 Common Motion API 而言，Alias ID 的最低值为 0x0001，最高值为 0xFFFF。若从轴为 ADAM-5000/ECAT，Alias ID 的后 3 位由从轴右手边的 3 个定位开关进行分配。当 ADAM-5000/ECAT 通电后，EEPROM 中 Alias 的后 3 位值将被定位开关的设置覆盖，但千位上的值将保留，将直至用户手动改变 ID。选择为从轴分配的 ID No，然后单击 [Change ID] 按钮应用设置。

端口状态

该项表示 EtherCAT 从轴的独立端口的 DLL 状态（数据链路层状态）。DLL 状态有两种：Link up: 端口已建立链接并发送 EtherCAT 数据报。
Loop Closed: 端口闭合。

4. 4. 5. 1 固件升级

Firmware

Current Firmware:

V0352_000

Firmware

Open

Password

0x00000000

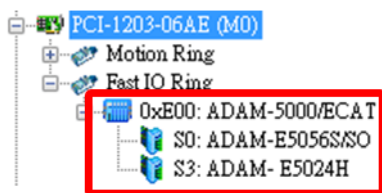
Upgrade

单击 [Open] 选择已获取的固件文件。单击 [Upgrade] 将激活向硬件下载文件的进程，进度栏将显示任务进度。固件升级可能花费若干分钟。进行固件升级的过程中不可关闭电源，否则将造成损坏。

升级完成后，请在操作前再次点击并打开设备树中的 PCI-1203 节点。

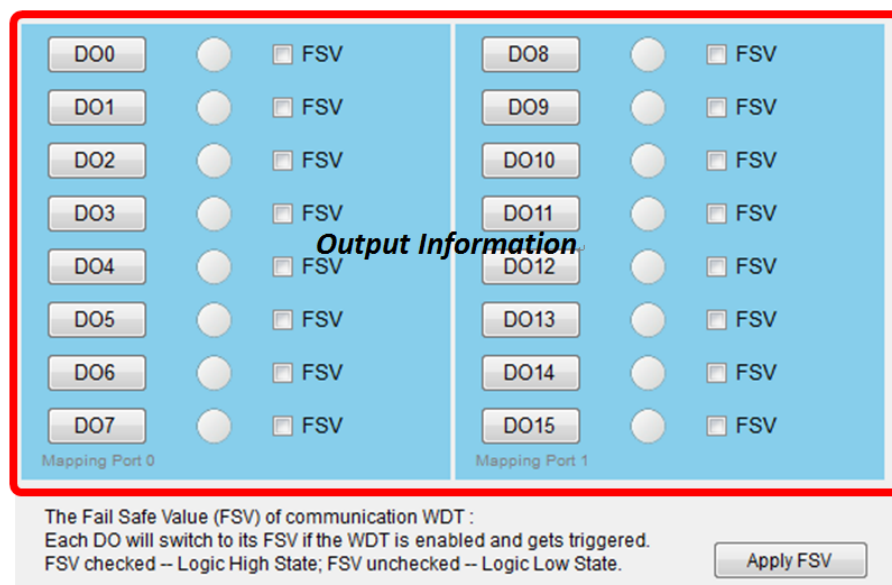
4.4.6 ADAM-E5000 模块页面

若 EtherCAT 从站为 ADAM-5000/ECAT，则 ADAM-5000/ECAT 的模块插头将显示在从站设备树中。单击模块节点，操作面板将显示相应页面。



4.4.6.1 数字量输出

该页支持带 LED 的 16 路隔离数字量输出模块 ADAM-E5056S，32 路隔离数字量输出模块 ADAM-E5057S 以及继电器输出模块 ADAM-E5069。



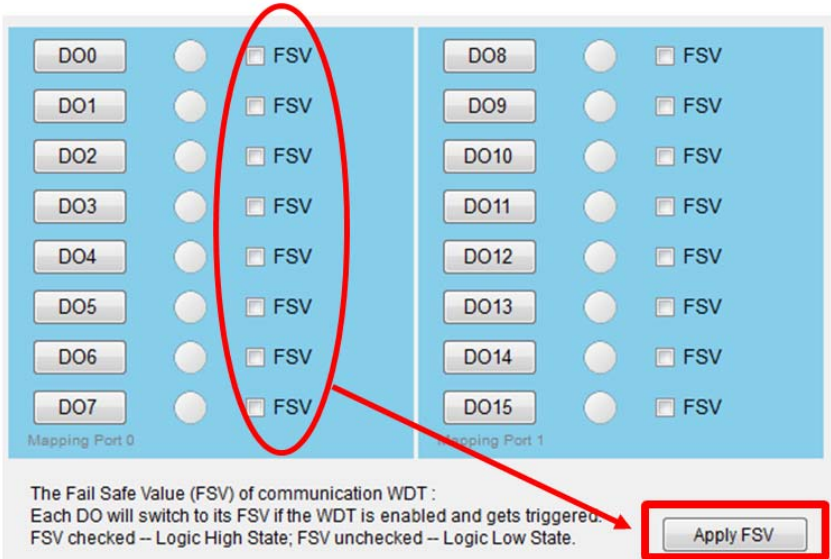
输出信息

根据通道数的不同，每个模块的信息框中的内容也有细微差别。

- 表示信号生效 (ON)，相应位的值为 1。
- 表示 DO 失效 (OFF)，相应位的值为 0。

Mapping Port 表示映射端口号。

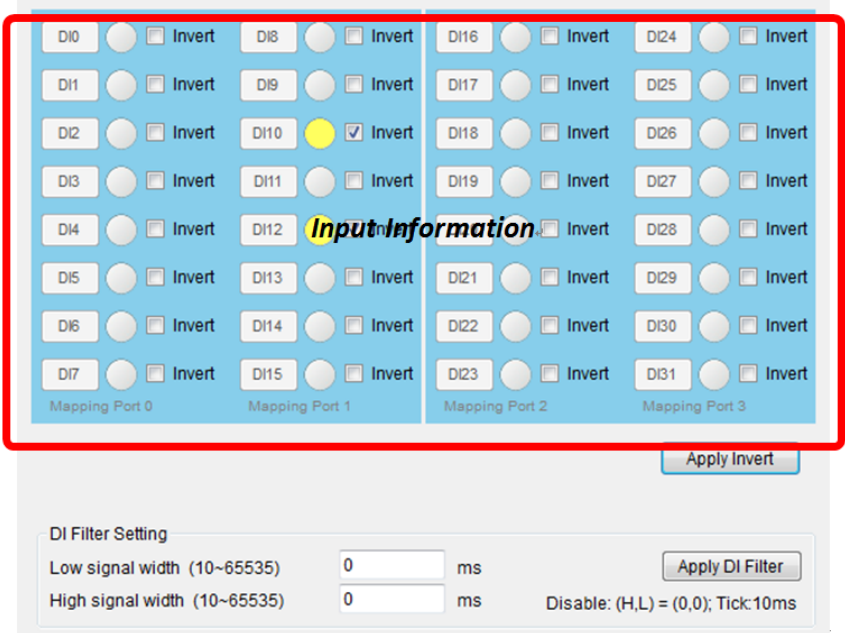
FSV（通信 WDT 的故障安全值）



当模块的 WDT 被启用和触发后，每个 DO 都将切换至自己的 FSV。若用户需要启用 FSV，选中每个通道的 FSV 勾选框，并单击 [ApplyFSV] 应用设置。

4. 4. 6. 2 数字量输入

该页支持带 LED 的 16 路隔离数字量输入模块 ADAM-E5051S 和 32 路隔离数字量输入模块 ADAM-E5053S。



输入信息

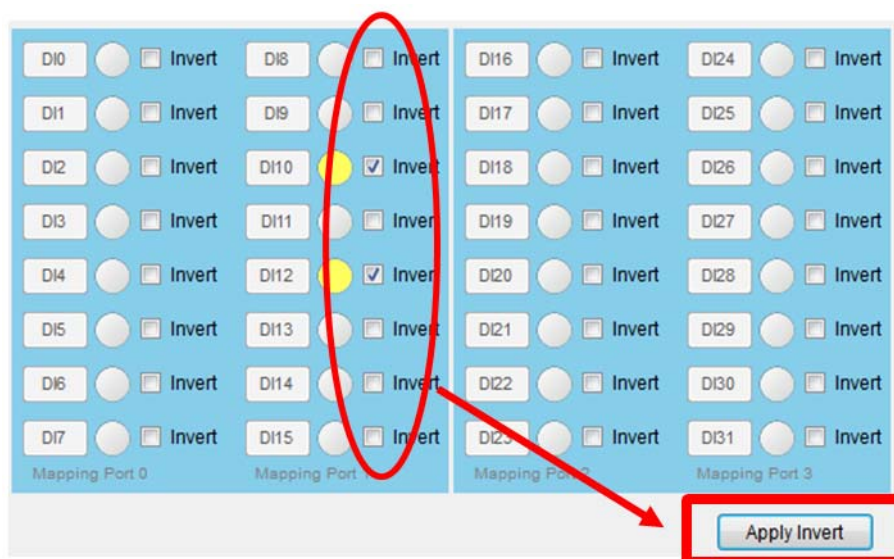
根据通道数的不同，ADAM-E5051S 和 ADAM-E5053S 的信息框中的内容也有细微差别。ADAM-E5053S 的页面显示 32 个通道，ADAM-E5051S 的页面显示 16 个通道。

- 表示信号生效 (ON)，相应位的值为 1。
- 表示 DI 失效 (OFF)，相应位的值为 0。

Mapping Port 表示映射端口号。

反向

若用户需要反转通道信号，可选中每个通道的 Invert 复选框，并单击 [Apply Invert] 应用设置。



数字量过滤器 (Low Signal Width)

数字量输入过滤器可消除输入信号中的噪音。一旦设置完成，数字量输入过滤器将在每次采样的同时储存数据，并将输入端状态的数据进行对比。若所有输入信号在过滤时间内达到低电平，则状态数据可作为输入端的值。反之，将使用上一个值。

数字量过滤器 (High Signal Width)

数字量输入过滤器可消除输入信号中的噪音。一旦设置完成，数字量输入过滤器将在每次采样的同时储存数据，并将输入端状态的数据进行对比。若所有输入信号在过滤时间内达到高电平，则状态数据可作为输入端的值。反之，将使用上一个值。



数字量过滤器的时间范围为 10 ~ 65535 毫秒。若要禁用过滤器，该索引的值必须设置为 0。完成所有设置之后，单击 [Apply DI Filter] 将设置应用至模块。

4.4.6.3 模拟量输出

该页支持 4 路模拟量输出模块 ADAM-E5024 和 4 路模拟量输出模块 ADAM-E5024H。

Channel	Value	Output Range	Raw
CH-0	0.000	-10 ~ 10V	0x0
CH-1	0.000	-10 ~ 10V	0x0
CH-2	0.000	-10 ~ 10V	0x0
CH-3	0.000	-10 ~ 10V	0x0

Output Information

Select Channel : 0

Value: -10

Current Value: 0.000

Apply output

Set current as startup

Set current as FSV

Configuration

Output range: -10 ~ 10V

☐ Apply to all

Apply

Calibration

Trim for -10V

Trim for 10V

Trim for 0V

Reset as default

输出信息

信息框将以工程单位和原始数据两种格式显示当前模拟量输出值。同时，还将显示其他设置，如：每个通道的输出范围。

获取工程单位数据时，转换公式和支持列表如下所示：

ADAM-E5024

输出范围	原始数据单位 (R) / 工程数值 (E)	工程数值 (E) / 原始数据单位 (R)
0~20 mA	$E = R * 20 / 4095 \text{ (mA)}$	$R = E * 4095 / 20$
4~20 mA	$E = R * 16 / 4095 + 4 \text{ (mA)}$	$R = (E - 4) * 4095 / 16$
0~10 V	$E = R * 10 / 4095 \text{ (V)}$	$R = E * 4095 / 10$

ADAM-E5024H

输出范围	原始数据单位 (R) / 工程数值 (E)	工程数值 (E) / 原始数据单位 (R)
0~20 mA	$E = R * 20 / 65535 \text{ (mA)}$	$R = E * 65535 / 20$
4~20 mA	$E = R * 16 / 65535 + 4 \text{ (mA)}$	$R = (E - 4) * 65535 / 16$
0~10 V	$E = R * 10 / 65535 \text{ (V)}$	$R = E * 65535 / 10$
-5~5 V	$E = R * 10 / 65535 \text{ (V)}$	$R = E * 65535 / 10$
-10~10 V	$E = R * 20 / 65535 \text{ (V)}$	$R = E * 65535 / 20$

设置输出值

Value: -10

Current Value: 0.000

Apply output

Set current as startup

Set current as FSV

ADAM-E5024 和 ADAM-E5024H 共有 4 个通道。用户可以通过滚动条设置输出值。在应用当前输出值之前，请确认输出范围类型是否正确。

上电输出值 (Startup)

操作 ADAM-E5024H 时，用户可以将当前模拟量输出值设置为启动输出值。在 ADAM-5000/ECAT 上电后，将通过相应通道输出电压 / 电流。对于启动输出值的设置将显示在输出信息区块。

注！ ADAM-E5024 不支持启动值。



故障安全输出值 (FSV)

操作 ADAM-E5024H 时，用户可以将当前模拟量输出值设置为 FSV（故障安全值）输出值。若操作 ADAM-5000/ECAT 时出现某些问题，例如网络断开，则每个 AO 通道将输出其故障安全值。对于 FSV 输出值的设置将显示在输出信息区块。

注！ ADAM-E5024 不支持 FSV。



输出范围



针对不同通道，输出范围可以设置为不同类型。选择需要更改的通道或单击 [Apply to all] 将设置应用于所有通道。选择适合的输出范围，然后单击 [Apply] 应用设置。

校准



若需校准最大负值、最大正值或进行零值校准，可单击 Calibration 区块中的按钮。用户必须精准仪器来校准模块，否则将导致通道数值不精确。操作 ADAM-E5024H 时，若用户要载入默认校准设置，可单击 [Default Setting] 按钮以恢复出厂设置。

注！ ADAM-E5024 不支持恢复出厂设置。



4.4.6.4 模拟量输入

该页支持8路模拟量输入模块ADAM-E5017和8路超高速模拟量输入模块ADAM-E5017UH。

Enable	Channel	Value	Input Range	Raw	Integration Time
☒	CH-0	0.002	-10 ~ 10V	0x800	60Hz
☒	CH-1	0.002	-10 ~ 10V	0x800	60Hz
☒	CH-2	0.007	-10 ~ 10V	0x801	60Hz
☒	CH-3	0.007	-10 ~ 10V	0x801	60Hz
☐	CH-4	*****		*****	
☐	CH-5	*****		*****	
☐	CH-6	*****		*****	
☐	CH-7	*****		*****	

Select Channel : 0

Configuration

Input Range: -10 ~ 10V

Integration Time: 60HZ

☐ Apply to all

Apply

Calibration

Zero Calibration

Span Calibration

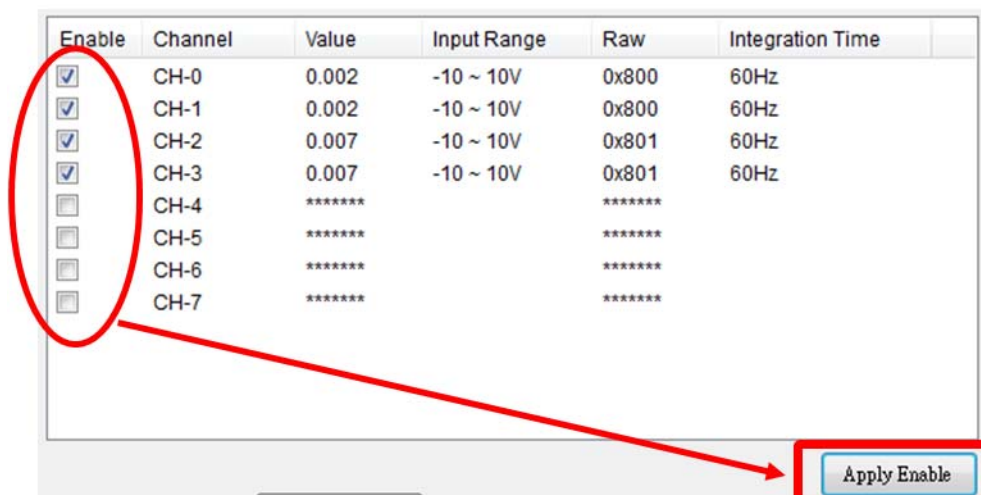
输入信息

该信息框将以工程单位（Value 栏）和原始数据（Raw 栏）两种格式显示当前模拟量输入值。同时，还将显示其他设置，如：每个通道的输入范围和积分时间。若通道被禁用，则值为原始数据。

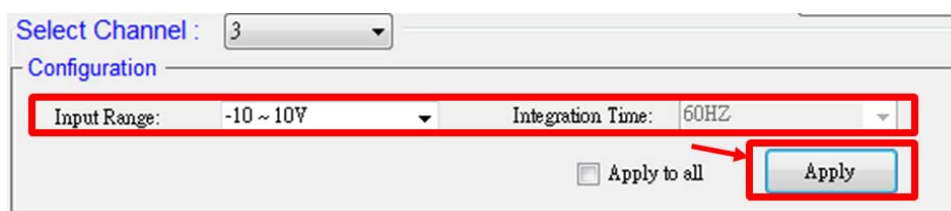
获取工程单位数据时，转换公式和支持列表如下所示：

输入范围	原始数据 (R) / 工程数值 (E)	工程数值 (E) / 原始数据单位 (R)	5017	5017 UH
+/- 10 V	$E = R * 20 / 65535 - 10 \text{ (V)}$	$R = (E + 10) * 65535 / 20$	√	√
+/- 5 V	$E = R * 10 / 65535 - 5 \text{ (V)}$	$R = (E + 5) * 65535 / 10$		√
+/- 1 V	$E = R * 2 / 65535 - 1 \text{ (V)}$	$R = (E + 1) * 65535 / 2$		√
+/- 500 mV	$E = R * 1000 / 65535 - 500 \text{ (mV)}$	$R = (E + 500) * 65535 / 1000$		√
+/- 150 mV	$E = R * 300 / 65535 - 150 \text{ (mV)}$	$R = (E + 150) * 65535 / 300$		√
+/- 20 mA	$E = R * 40 / 65535 - 20 \text{ (mA)}$	$R = (E + 20) * 65535 / 40$		√
0~10 V	$E = R * 10 / 65535 \text{ (V)}$	$R = E * 65535 / 10$	√	
0~500 mV	$E = R * 500 / 65535 \text{ (mV)}$	$R = E * 65535 / 500$	√	
0~20 mA	$E = R * 20 / 65535 \text{ (mA)}$	$R = E * 65535 / 20$	√	
4~20 mA	$E = R * 16 / 65535 + 4 \text{ (mA)}$	$R = (E - 4) * 65535 / 16$	√	√

在信息区块中，用户可以通过取消勾选每个通道之前的复选框来禁用模拟量输入通道，然后单击 [Apply Enable] 应用设置。



配置



用户可以在此区域设置输入范围和积分时间。选择输入范围和积分时间，然后单击 [Apply] 按钮应用设置。

注！ ADAM-E5017 与 ADAM-E5017UH 有一些差异。



对于 ADAM-E5017 而言，模块的所有通道都使用相同的输入范围设置。用户单击 [Apply] 按钮之后，可将该设置应用至模块的所有通道。

对于 ADAM-E5017UH 而言，积分时间功能不可用，而每个通道都支持单独设置输入范围。在通道选项组合框中选择需要设置的通道，单击 [Apply to all] 将设置应用至所有通道。

4.4.6.5 计数器

该页支持 ADAM-E5082 计数器模块。用户可在通道 0 和通道 1 的标签页之间进行切换。

Channel 0 Channel 1 IO List

Channel

Counting: ☒
Current Mode: CW/CCW
Current Value: 0 Reset

DIO Status

DO: 0 ☐ 1 ☐

DI: 0 ☐ Invert A ☐ Invert

B ☐ Invert Z ☐ Invert

Apply Invert

Startup Value
0 Set

Overflow

Mode: ☒ Normal ☐ Ring Counter

Max. Value: 22222101 Set

Latch

Enable: ☐ Data: 0 Get Flag: ☐ Type: ☒ Rising ☐ Falling ☐ Both

Compare

Enable: ☐ Value/Mode

Data: 0 Set Flag: ☐ Type: ☒ Greater ☐ Smaller ☐ Both

DO: ☐ Trigger

Output

Level: ☒ Normal ☐ Reverse

Mode: Toggle Pulse Width: 1 Set

计数器通道值和计数模式

Channel

Counting: ☒
Current Mode: CW/CCW
Current Value: 0 Reset

信息区块显示了所选通道的当前计数器值和计数模式。点击计数按钮 ☒ 可启用 / 禁用该通道。ADAM-E5082 模块支持 5 种计数模式：A/B 1X、A/B 2X、A/B 4X、CW/CCW 和 PULSE/DIR。单击模块复合框可选择计数模式。若用户需要重置计数值，可点击重置按钮将计数值重置为起始值。

起始值和溢出值

Startup Value
0 Set

Overflow

Mode: ☒ Normal ☐ Ring Counter

Max. Value: 22222101 Set

用户可以配置计数器的起始值。在文本框中输入设置的值，然后单击设置按钮完成配置。

ADAM-E5082 支持两种溢出模式。在环式计数模式中，当计数值超出最大值时，计数值将重新从起始值开始，计数方式如同一个环。在普通模式中，当计数值超出最大值时，计数值将溢出，并以 -（最大值 + 1）继续。用户可以为不同应用选择不同模式。

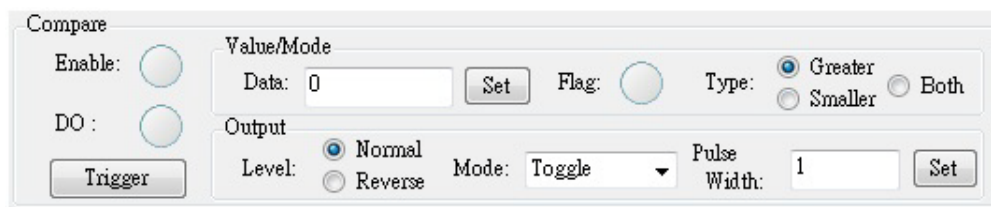
锁存

Latch

Enable: ☐ Data: 0 Get Flag: ☐ Type: ☒ Rising ☐ Falling ☐ Both

单击启用按钮即可启用锁存功能。用户可以选择上升、下降、或上升与下降出发沿。当数字量输入被触发时，模块将锁存当前计数值，且 Flag 按钮  将变亮。单击获取按钮  可读取锁存值并重置锁存功能，以便支持该模块的下一轮锁存动作。

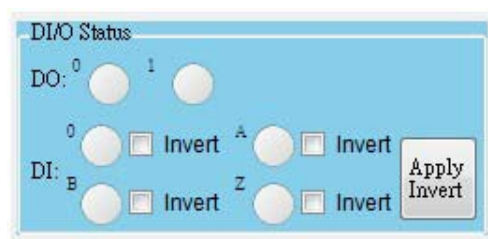
比较



在数据文本框中输入数值，然后单击设置按钮即可完成比较值设置。单击启用按钮即可启用该功能。完成数值比较之后，Flag 按钮  将变亮。单击 Flag 按钮可为下一个比较值重新执行比较操作。用户可选择更大、更小、或更大与更小比较模式。

单击 DO 按钮  可启用比较触发数字量输出功能。在脉冲模式中，用户可设置输出电平、输出模式和触发宽度。

DI / DO 状态



该区块显示模块的所有 4 个 DO 和 2 个 DI 的状态。DO 0 和 DO 1 位于通道 0 的页面，DO 2 和 DO 3 位于通道 1 的页面。单击 DO 按钮可开启 / 关闭输出。该区块不仅显示通用 DI 状态，还显示计数器的 A / B / Z 输入状态。用户若要反转输入电平，可以选中反转复选框，然后单击“Apply”按钮以应用反转设置。

第 5 章

编程指南

5.1 简介

本章主要介绍如下两部分内容：

- 关于研华运动控制卡接口函数库
介绍接口函数库相关内容
- 研华运动控制卡接口函数库在编程工具中的使用，如下：
在 Visual C++ 6.0 中的使用
在 Visual Basic 6.0 中的使用
在 Visual Studio 2005 中的使用
在 Borland C++ Builder 中的使用

5.2 关于研华运动控制卡接口函数库

为了统一研华所有运动设备的用户接口（即不同的运动设备采用统一的用户接口），设备驱动采用了新的软件架构，名为“通用运动架构（Common Motion Architecture 关于该架构的说明，见 4.1 章节）”。在该架构下，用户能够调用相同的接口函数操作不同的设备，如下图所示。

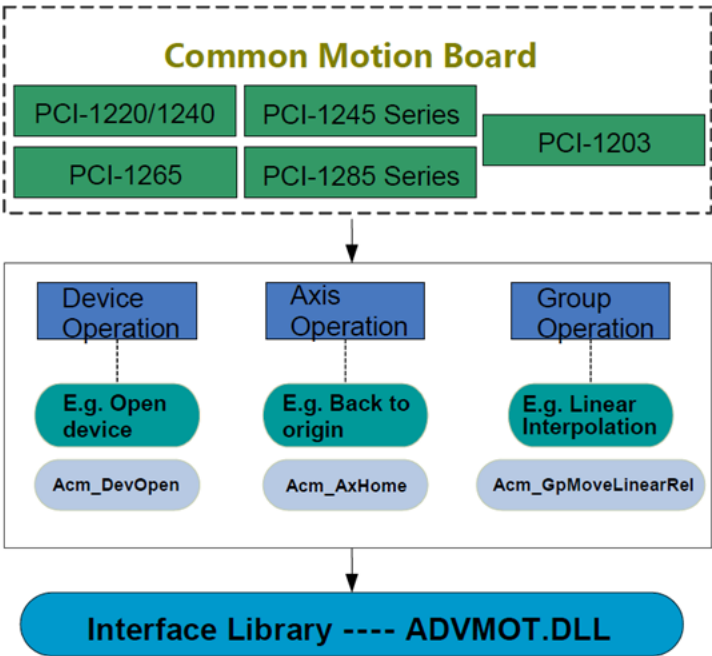


图 5.1：不同设备对接口函数的调用

如上图所示，实现设备功能的所有 API 都可从接口函数库 ADVMOT.DLL 中获取（在不同的编程环境中，获取方式不同，参见 3.3 节）。为了调用该接口函数库，需要通过安装包安装驱动，该接口函数库将被安装到 Windows 系统文件夹下。

5.3 运动控制卡接口函数库所需头文件

为了实现板卡功能，运动控制卡接口函数库需调用以下头文件。如图所示：

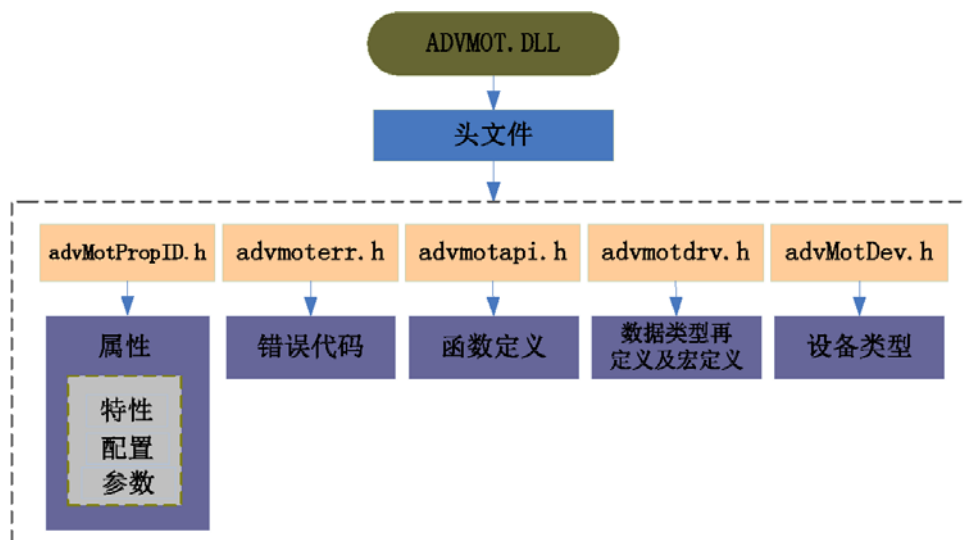


图 5.2：动态链接库所需头文件

接口函数库所需头文件，将放在安装目录的“Public”文件夹中。根据头文件的功能，以下章节将分别对其介绍。

5.3.1 接口函数库 --- 功能函数

研华通用运动架构定义了三种类型的操作对象：设备、轴和群组。每个类型都有自己的功能函数，功能函数原型在 AdvmotApi.h 中定义。

5.3.1.1 功能函数命名规则

根据设备、轴、群组三种操作对象以及对 AIO、DIO 的操作，功能函数按如下规则命名：

操作对象	命名规则	说明	举例
设备	Acm_DevXXX	执行设备功能	Acm_DevOpen
AIO/DIO 操作	Acm_DaqXXX	执行 DI/DO、AI /AO 功能	Acm_DaqDoGetBit
轴操作	Acm_AxXXXX	执行轴功能	Acm_AxOpen
群组操作	Acm_GpXXXX	执行群组功能	Acm_GpAddAxis

5.3.1.2 功能函数简单使用说明

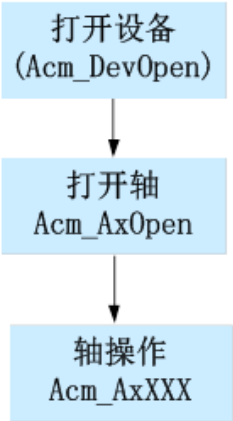
功能函数可在应用程序中直接调用。根据设备、轴、群组三种操作对象，简单说明对其操作流程。

对设备的基本操作如下：

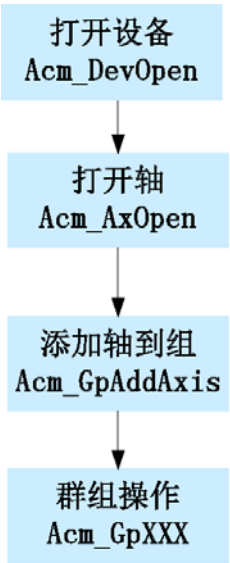


关于如何获取设备号，详见 5.3.6.1 章节。

对轴的基本操作如下：



对群组的基本操作如下：



注！



打开设备和轴后，可调用 *Acm_DevLoadConfig* 函数加载配置文件，设置设备的所有配置，如轴的初速度、加速度、报警信号的逻辑电平等。关于加载配置文件的说明详见第五章节。

5.3.2 接口函数库 —— 功能属性

对应设备、轴、群组，DIO/AIO，每个操作对象都有各自的属性，属性的名称及属性值的定义请参考头文件 *AdvMotPropID.h*。关于属性的命名规则及属性的设置 / 获取参见 5.3.2.1 和 5.3.2.2 节。

5.3.2.1 属性的命名规则

属性根据特点分为特性、配置、参数属性，属性的命名按如下规则：

表 5.1：属性的命名规则		
属性分类	命名规则	分类规则
特性	设备	FT_DevXXX
	轴	FT_AxXXX
	DIO/AIO	FT_DaqXXX
		该属性与硬件特性有关，只能获取设备的相关属性，不能设置

表 5.1: 属性的命名规则

配置	设备	CFG_DevXXX	配置属性值部分可重新设置，其他不可设置，主要设置属性可变但不会经常变化
	轴	CFG_AxXXXX	
	群组	CFG_GpXXXX	
	DIO/AIO	CFG_DaqXXX	
参数	设备	PAR_DevXXX	参数属性值会经常变化
	轴	PAR_AxXXXX	
	群组	PAR_GpXXXX	
	DIO/AIO	PAR_DaqXXX	

5.3.2.2 属性的设置和获取

属性不能直接调用，而是通过设置和获取属性的 API 实现，设置 / 获取属性的 API 如下表所示：

表 5.2: 设置 / 获取属性函数列表

设置 / 获取属性函数	说明
Acm_SetU32Property	设置数据类型为无符号 32 位整形的属性值
Acm_SetI32Property	设置数据类型为有符号 32 位整形的属性值
Acm_SetF64Property	设置数据类型为 Double 型的属性值
Acm_GetU32Property	获取数据类型为无符号 32 位整形的属性值
Acm_GetI32Property	获取数据类型为有符号 32 位整形的属性值
Acm_GetF64Property	获取数据类型为 Double 型的属性值

如表 5.2 函数说明，设置 / 获取属性调用的 API 与该属性的数据类型有关，如下图：

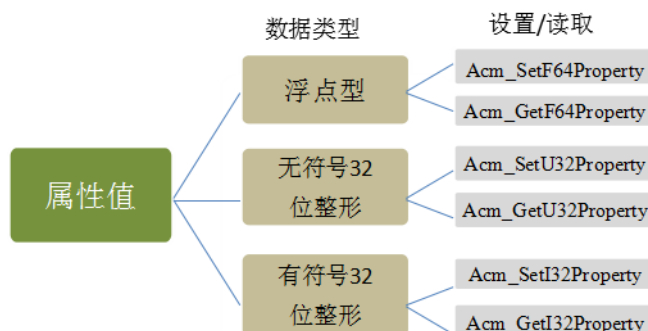


图 5.3: 属性数据类型与函数的关系

如下表为轴的运行速度属性定义：

属性名	数据类型	读 / 写	属性 ID	含义
PAR_AxVelHigh	F64	RW	401	设置 / 获取该轴的运行速度

该属性的数据类型为 F64（即 Double 型），因此需调用 Acm_SetF64Property / Acm_GetF64Property 函数设置 / 获取该属性值，获取属性 API 说明如下：

函数原型：Acm_SetF64Property (HAND Handle, U32 PropertyID, F64 Value)

参数一: Handle(操作对象的 Handle)	参数二: PropertyID	参数三: Value
设备属性 设备 Handle，通过函数 Acm_DevOpen 获取		
轴属性 轴 Handle，通过函数 Acm_AxOpen 获取	可直接输入属性名 如: PAR_AxVelHigh	设置的属性值，数据类型为 F64
组属性 组 Handle，通过函数 Acm_GpAddAxis 获取		

如设置轴的运行速度为 8000，则设置如下：

Acm_SetF64Property(Axishand, PAR_AxVelHigh, 8000); // 运行速度 8000

5.3.3 接口函数库 --- 设备类型

设备类型的定义在头文件 AdvMotDev.h 中，方便用户编写程序。例如获取板卡的设备号，关于获取板卡的设备号见 6.4 章节。设备类型定义如下：

表 5.3：设备类型定义	
设备类型	类型定义
PCI-1220	Adv_PCI1220
PCI-1240	Adv_PCI1240
PCI-1245	Adv_PCI1245
PCI-1245L	Adv_PCI1245L
PCI-1245E	Adv_PCI1245E
PCI-1245V	Adv_PCI1245V
PCI-1245S	Adv_PCI1245S
PCI-1245HP	Adv_PCI1245HP
PCI-1265	Adv_PCI1265
PCI-1285	Adv_PCI1285
PCI-1285E	Adv_PCI1285E
PCI-1285V	Adv_PCI1285V
MIC-3245	Adv_MIC3245
MIC-3285	Adv_MIC3285
ADAM-5000ECAT (Lan Port)	Adv_ADAM5000
PCI-1203	Adv_PCI1203

5.3.4 接口函数库 --- 错误代码

调用接口函数库中的 API 时，每个 API 都将获得一个返回代码。返回代码表示调用结果。用户可根据 Acm_GetErrorMessage 查看返回的错误代码对应的错误信息。根据错误信息，用户能够进行适当修改。关于错误代码的详细信息，请参考附录。

5.3.5 接口函数库 --- 数据类型再定义

为了简化代码，对数据类型进行重新定义，数据类型再定义和 Windows 数据类型对照表如下所示：

表 5.4：数据类型再定义		
新类型	Windows 数据类型	备注
U8	UCHAR	8 bit 无符号整数
U16	USHORT	16 bit 无符号整数
U32	ULONG	32 bit 无符号整数
U64	ULONGLONG	64 bit 无符号整数
I8	CHAR	8 bit 带符号整数
I16	SHORT	16 bit 带符号整数
I32	INT	32 bit 带符号整数
I64	LONGLONG	64 bit 带符号整数
F32	FLAOT	32 bit 浮点变量
F64	DOUBLE	64 bit 浮点变量
PU8	UCHAR *	指针指向 8 bit 无符号整数
PU16	USHORT *	指针指向 16 bit 无符号整数
PU32	ULONG *	指针指向 32 bit 无符号整数

表 5.4：数据类型再定义

PU64	ULONGLONG *	指针指向 64 bit 无符号整数
PI8	CHAR *	指针指向 8 bit 带符号整数
PI16	SHORT *	指针指向 16 bit 带符号整数
PI32	INT*	指针指向 32 bit 带符号整数
PI64	LONGLONG *	指针指向 64 bit 带符号整数
PF32	FLAOT *	指针指向 32 bit 浮点变量
PF64	DOUBLE *	指针指向 64 bit 浮点变量

5.3.6 板卡的初始操作

对于板卡的操作，如 5.3.1.2 节所介绍，首先需要根据设备号打开板卡，获取设备 Handle，然后打开轴，即可执行单轴运动操作；如需执行群组操作，打开轴后，将轴添加到群组中，获取群组的 Handle，即可执行群组操作。

根据这一流程，本节重点介绍板卡的初始操作。

5.3.6.1 获取板卡设备号

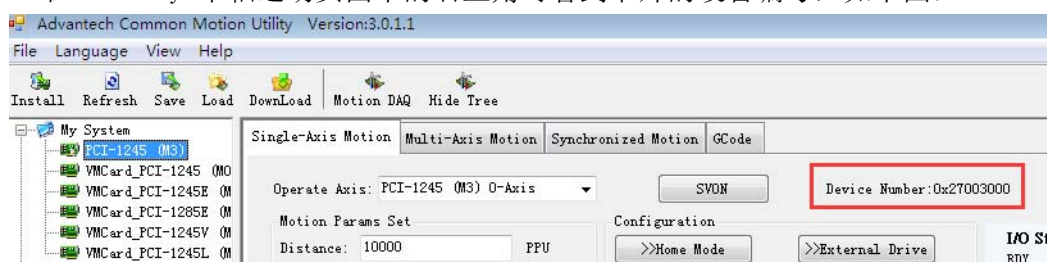
为了区分不同设备，将每张板卡进行编号，即设备号。每张板卡的设备号唯一，关于板卡编号的规则见 6.4 章节。

操作板卡需要调用 Acm_DevOpen 函数打开设备以获取设备的句柄，该函数需要传入设备号，获取板卡设备号有以下两种方式：

- 方式一：通过 Utility 获取
- 方式二：通过函数获取

1. 方式一：通过 Utility 获取

在 Utility 单轴运动页面中的右上角可看到卡片的设备编号，如下图：



可在打开设备函数中直接输入该 Device Number 获取设备句柄。

2. 方式二：通过函数获取

接口函数库中提供两个函数获取设备号：

Acm_GetAvailableDevs (DEVLIST *DeviceList, U32 MaxEntries, PU32 OutEntries)

Acm_GetDevNum(U32 DevType, U32 BoardID, PU32 DeviceNumber)

关于函数 Acm_GetAvailableDevs 的说明见附录 A，其使用可参考范例。

关于函数 Acm_GetDevNum 的使用说明如下：

参数：

名称	类型	IN 或 OUT	说明
DevType	U32	IN	头文件定义的设备类型，见 5.3.3 节
BoardID	U32	IN	板卡 ID
DeviceNumber	PU32	OUT	获取的设备号

Board ID 可从硬件设备上直接获取，且可自由调整，如不调整，BoardID 一般不会发生变化。BoardID 也可从 Utility 上获取。

如下获取设备号打开设备：

```
HAND    m_Devhand;
```

```

U32      m_DevNum;
Acm_GetDevNum(Adv_PCI1203, 1, &m_DevNum); // 获取设备号
Acm_DevOpen(m_DevNum, &m_Devhand); // 打开设备，获取设备 Handle

```

5.3.6.2 板卡的初始化流程图

板卡的初始化流程如下图所示：

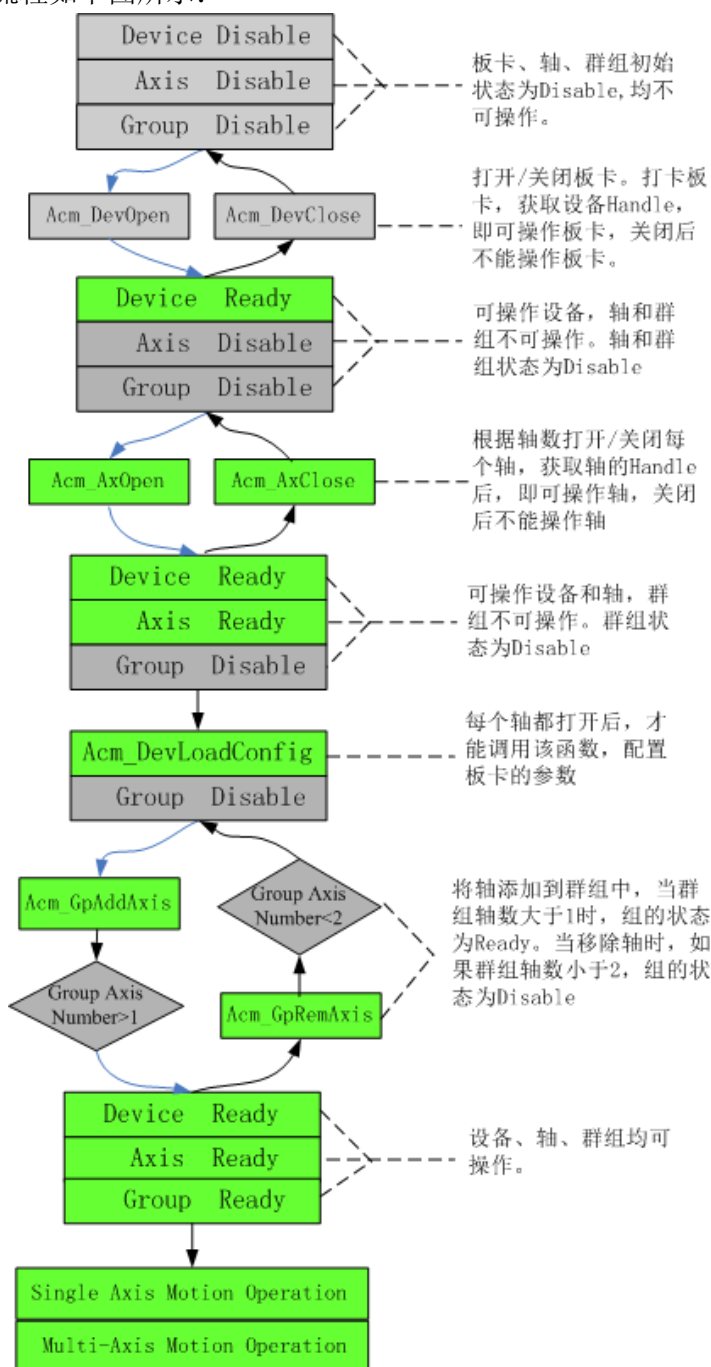


图 5.4：板卡初始化流程图

5.3.6.3 板卡的初始化例程

板卡的初始化例程如下，在打开设备之前，首先需要获取设备号（获取方式参见5.3.6.1节），本例程通过函数获取。

VC代码如下：

```

HAND    m_Devhand;           // 设备的 Handle
HAND    m_Axishand[32];      // 轴的 handle
HAND    m_GpHand;            // 组的 handle

```



```

U32    m_DevNum;           // 设备号
U32    m_ulAxisCount;      // 轴数量
U32    Ret;                // 函数返回值
Ret = Acm_GetDevNum(Adv_PCI1203, 15, &m_DevNum); // 获取板卡设备号
Ret = Acm_DevOpen(m_DevNum, &m_Devhand); // 打开板卡，获取设备 Handle
Ret = Acm_GetU32Property(m_Devhand, FT_DevAxesCount, &m_ulAxisCount); // 获取
轴数量
// 根据轴数，将所有轴打卡
for(uint AxisNumber=0;AxisNumber<m_ulAxisCount;AxisNumber++)
{
Ret = Acm_AxOpen(m_Devhand, (USHORT)AxisNumber, &m_Axishand[AxisNumber]); //
打开轴，获取每个轴的 Handle
Acm_AxSetCmdPosition(m_Axishand[AxisNumber], 0); // 设置轴的命令位置为 0
Acm_AxSetActualPosition(m_Axishand[AxisNumber], 0); // 设置轴的实际位置为 0
}
// 将 0 轴和 1 轴添加到群组中
Ret = Acm_GpAddAxis(&m_GpHand, m_Axishand[0]);
Ret = Acm_GpAddAxis(&m_GpHand, m_Axishand[1]);
// 执行单轴运动，如点到点运动，请参考 9.2 章节
// 执行多轴运动，如直线运动，请参考 9.3 章节
// 执行完操作后，需关闭轴，关闭群组，关闭设备
Acm_GpClose(&m_GpHand); // 关闭群组
for(uint AxisNum=0;AxisNum<m_ulAxisCount;AxisNum++)
    { Acm_AxClose(&m_Axishand[AxisNum]); // 关闭每个被打开的轴 }
Acm_DevClose(&m_Devhand); // 关闭设备

```

5.4 运动控制卡接口函数库在编程工具中的使用

研华运动控制卡提供了 VC、VB、BCB、C#、VB.Net 语言的范例，用户安装范例安装包后，在 “Advantech\Motion Common\Example_1203” 文件夹下有不同语言编写的示例，用户可参考开发自己的应用程序。

安装成功后，在 “\Advantech\Motion Common” 目录下有两个文件夹：Include 和 Public。“Public” 文件夹中的文件可供用户在不同的编程工具中调用。

接口函数库与不同开发语言之间的关系如下图所示：

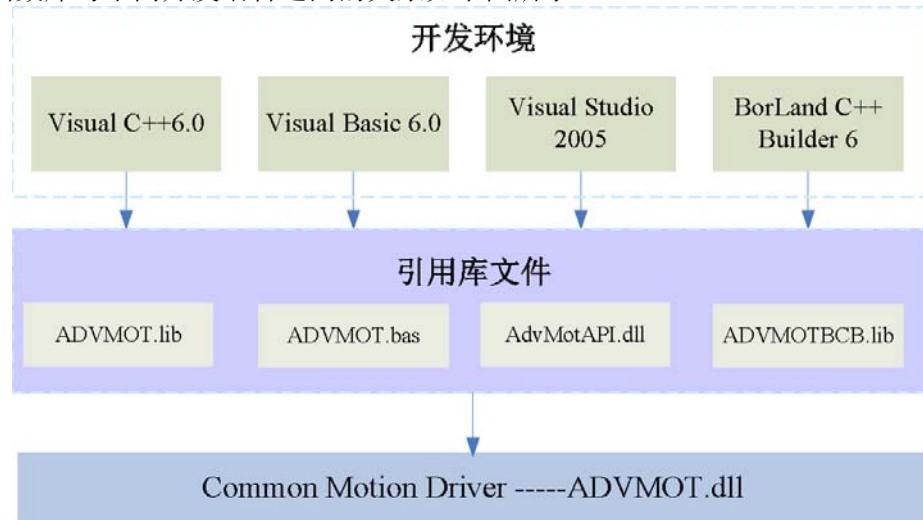


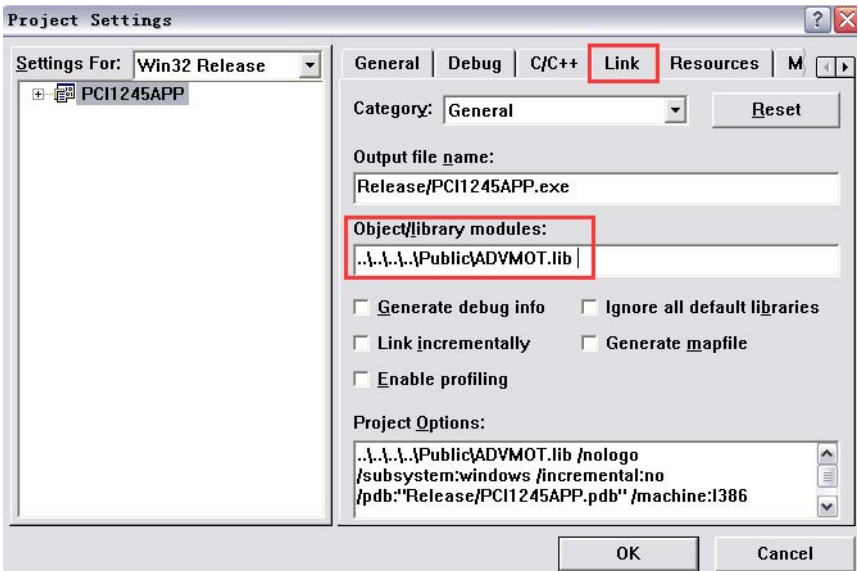
图 5.5： 属性数据类型与函数的关系

用于实现设备功能的所有 API 都可从 ADVMOT.DLL 获取。AdvMotAPI.dll、ADVMOT.bas、ADVMOT.lib、ADVMOTBCB.lib 都是基于 ADVMOT.dll 产生，方便用户轻松开发应用程序。AdvMotAPI.dll 用于 C# 应用程序和 VB.net 应用程序。ADVMOT.bas 用于开发 VB 应用程序。ADVMOT.lib 用于开发 VC 应用程序。ADVMOTBCB.lib 用于开发 BCB 应用程序。

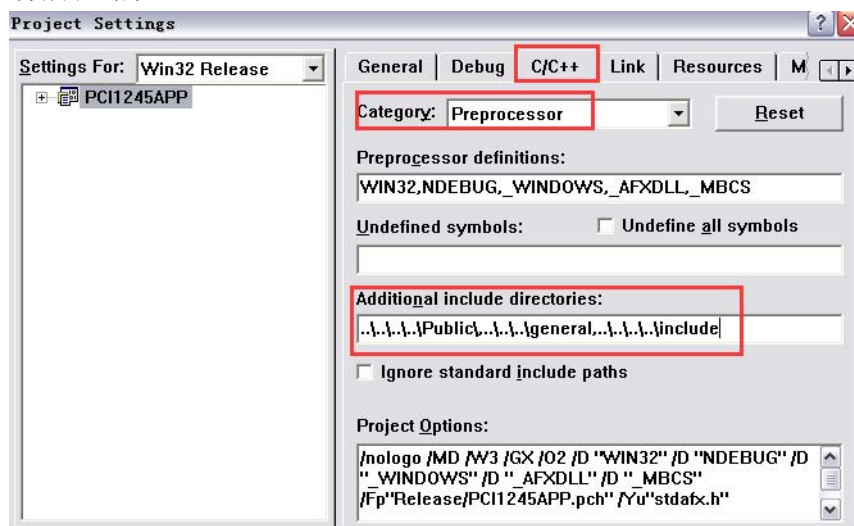
5.4.1 接口函数库在 Visual C++ 6.0 中的使用

在 Visual C++ 6.0 中设置所需头文件和库文件过程如下：

1. 启动 Visual C++ 6.0，新建一个工程
2. 选择“Project”菜单下的“Settings...”菜单项
3. 切换到“Link”标签页，在“Object\library modules”栏中输入 ADVMOT.lib 文件所在路径



4. 切换到“C/C++”标签页，在“Additional include directories”栏中输入头文件所在路径：



5. 在应用程序文件中加入函数库头文件的声明，如下所示：

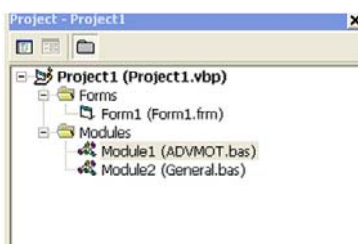
```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "advmotdrv.h"
#include "General.h"
#include "AdvMotApi.h"
```

至此，用户就可以在 Visual C++ 中调用函数库中的任何函数，开始编写应用程序。

5.4.2 接口函数库在 Visual Basic 6.0 中的使用

在 Visual Basic 6.0 中设置所需库文件过程如下：

1. 启动 Visual Basic。选择“Standard EXE”图标，然后按“Open”按钮。这样，就创建了一个新工程。
2. 将模块添加到工程中。从“View”菜单中单击“Project Explorer”。通过单击“Project”窗口菜单中的“Add Module”，添加 ADVMOT.bas（安装示例包后，位于“Advantech\Motion Common\Public”路径下）模块和 general.bas（安装示例包后，位于“\Advantech\Motion Common\Examples”路径下）模块。



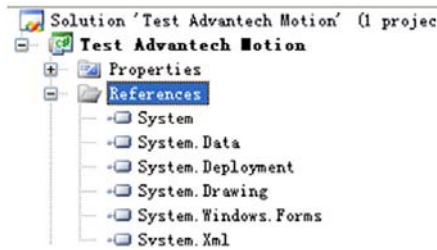
至此，即可调用接口函数库中的所有函数，可参考例程。

5.4.3 接口函数库在 Visual Studio 2005 中的使用

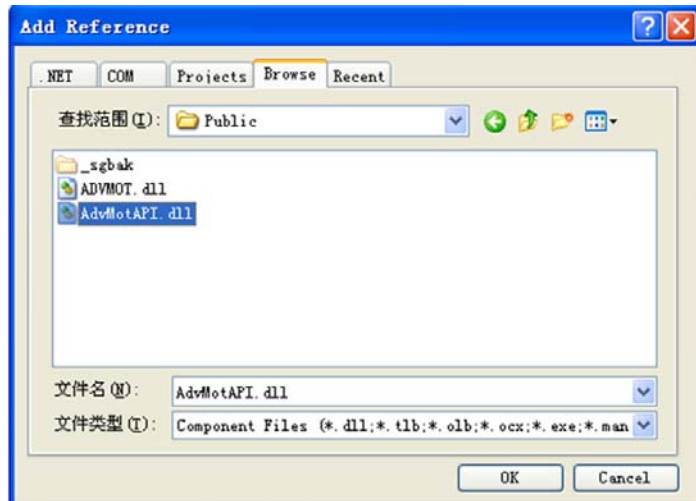
使用 Visual Studio 2005 开发 c# 项目，添加库文件过程如下：

1. 启动 Visual Studio 2005，按着“File”->“New”，选择建立 C# 工程
2. 添加相关 DLL 文件的引用

- a. 环境右上角处的 [References]，如下图所示：



- b. 单击 [Add Reference] 对话框的 [Browse]，从搜索路径选择 “Public” 文件夹中的 “AdvMotAPI.dll”，然后单击 [OK]，如下图所示：



- c. 在编辑界面上右击，选择 [View Code] 进入程序源代码编辑界面，如下图所示：



- d. 在原始参考命名空间下添加 “using Advantech.Motion”，如下图所示：

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Text;
7 using System.Windows.Forms;
8 using Advantech.Motion;
```

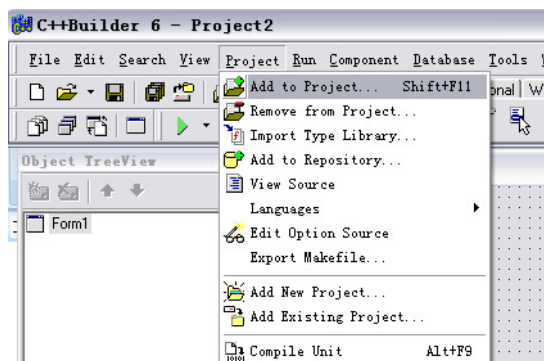
至此，即可调用接口函数库中的所有函数，可参考例程，关于用 Visual Studio 2005 开发 VB.NET 项目添加的库文件同 c# 相同。

5.4.4 接口函数库在 Borland C++ Builder 中的使用

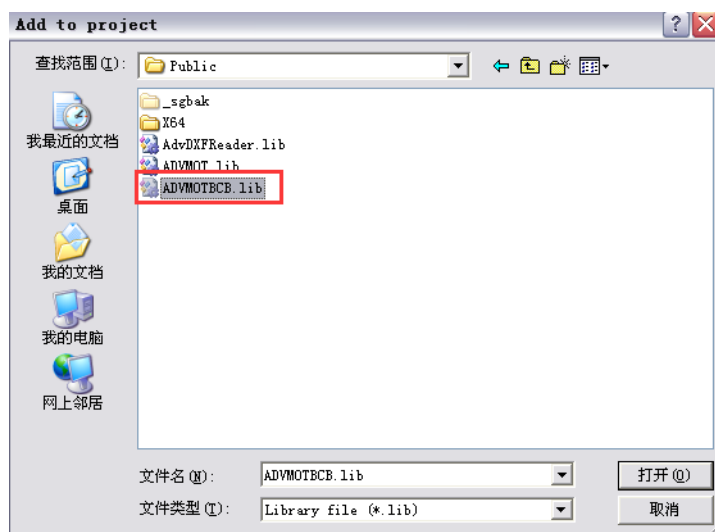
使用 Borland C++ Builder 6.0 开发 BCB 项目，添加库文件过程如下：

1. 启动 Borland C++ Builder 6.0，新建应用 (New-Application)

2. 添加库文件到项目中，如下，Project→Add to Project



3. 在从搜索路径选择 “Public” 文件夹中的 “ADVMOTBCB.lib”，然后单击 [OK]，如下图所示：



添加完成后，即可调用 Common Motion API 开发项目。

5.5 关于在 Win7 下使用 Common Motion API 的注意事项

5.5.1 关于提升应用程序的权限

在调用 Acm_GetAvailableDevs 函数获取板卡信息时，需要读取注册表。在 Win7 系统下，此操作需要有管理员权限才能完成，否则将不能获取到板卡。因此如果应用程序要调用此函数，应将应用程序权限提升为管理员权限，提升方法如下：

1. 使用 Microsoft Visual Studio 2005 开发项目

从 C#/VB.net 范例中 Properties 文件夹下拷贝 Manifest 文件 “app.manifest” 到本工程的 Properties 文件夹下，通过 Project→Add Existing Item 选项添加到工程中即可。

2. 使用 Microsoft Visual C++ 6.0 开发项目

从 VC 范例中拷贝 Manifest 文件：App.manifest 到本工程路径下，在资源中导入这个文件，资源类型 24，资源 ID 为 1 即可。

3. 使用 Microsoft Visual Studio 2008/2010 开发项目

方法一：可以同上面 VS2005 一样，从范例中拷贝 app.manifest 文件加载到所开发的工程中

方法二：直接更改工程权限管理的设置即可：在工程属性 --> Configuration Properties --> Linker --> Manifest File --> UAC Execution Level --> require Administrator。

方法三：勾选 Project properties 对话框中的 Security 栏中的 “Enable ClickOnce Security Setting” 选项，在 Properties 下将自动生成 Manifest 文件，打开 Manifest 文件，将如下截图中的红框行改为

“<requestedExecutionLevel level=“requireAdministrator”

uiAccess=“false” />”；然后再将 Project properties 对话框中的 Security 栏中的 “Enable ClickOnce Security Setting” 选项的勾去掉即可。

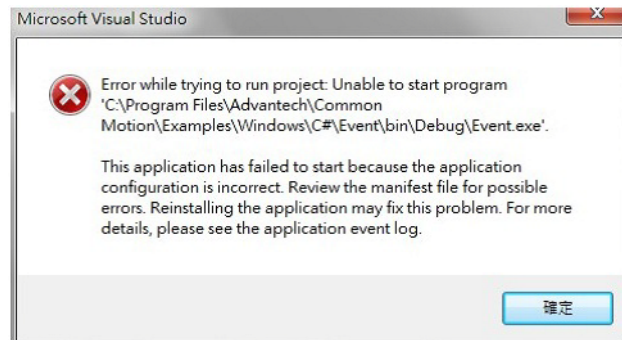
```
<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
  <!-- UAC Manifest Options
  If you want to change the Windows User Account Control level replace the
  requestedExecutionLevel node with one of the following.

  <requestedExecutionLevel level="asInvoker" uiAccess="false" />
  <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
  <requestedExecutionLevel level="highestAvailable" uiAccess="false" />

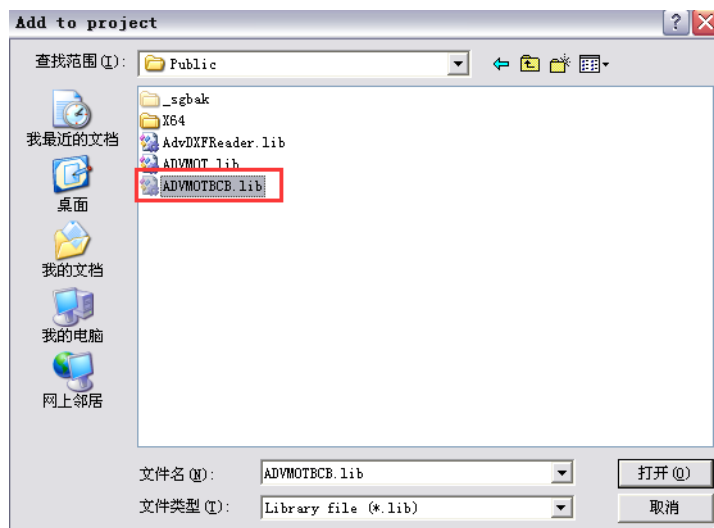
  Specifying requestedExecutionLevel node will disable file and registry virtualization.
  If you want to utilize File and Registry Virtualization for backward
  compatibility then delete the requestedExecutionLevel node.
-->
  <requestedExecutionLevel level="asInvoker" uiAccess="false" />
</requestedPrivileges>
```

5.5.2 Win7 下运行范例

在使用 VS2008 或者 VS2010 打开 C#/VB.net 范例，若执行的时候提示如下错误：



请将 Project properties 对话框中的 Security 栏中的 “Enable ClickOnce Security Setting” 的打钩去掉，重新编译即可正常运行。



5.6 多块板卡编程说明

当用户电脑上有多块板卡（两块及以上）时，使用方法如下介绍。

5.6.1 多块板卡的独立性

当电脑上有多块板卡时，每块板卡是独立的，群组 (Group) 差补功能 (Line/Arc) 所使用的轴必须在同一块板卡，不能跨板卡。例如，两块 PCI-1203 板卡，其中 PCI-1203(0) 与 PCI-1203(1) 只能操作连接到该板卡运动环上的轴，不能选择该板卡 (0) 的轴和另一板卡 (1) 的任一轴做插补运动。

5.6.2 如何操作多快板卡

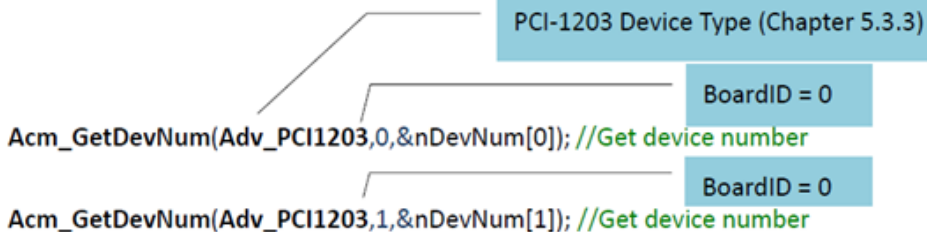
下面例子以一台电脑上插 2 块 PCI-1203 板卡，说明多块板卡操作过程：

板卡 1: PCI-1203 BoardID = 0

板卡 2: PCI-1203 BoardID = 1

1. 板卡的设备号 (Device Number):

```
DWORD nDevNum[2]; // Device number of two PCI Card
```



注！ 关于 Device Number 可参考手册 5.3.6.1



2. 打开 2 块板卡和板卡上的轴

获取到板卡的设备号后，调用 Acm_DevOpen 打开板卡，获取设备 Handle，同时调用 Acm_AxOpen 打开设备上的轴，获取轴的 Handle。

```
HAND hDevice[2]; // 设备 Handle (2 块 PCI-1203 板卡)
HAND hAxis[8]; // 轴的 Handle。例如每块 PCI-1203 板卡各们连接 4 轴（共 8 轴）
int nAxis=0;
int nDev=0;
for( nDev=0; nDev<2; nDev++)
{ Acm_DevOpen(nDevNum[nDev], &hDevice[nDev]); // 打开板卡
for ( U16 PhyAxisID=0; PhyAxisID <4; PhyAxisID ++ )
{ Acm_AxOpen( hDevice[nDev], PhyAxisID, &hAxis[nAxis] ); // 打开板卡的所有轴
nAxis ++; }}
```

3. 板卡的运动操作

■ 板卡的单轴运动

获取到轴的 Handle 后，就可以操作轴，例如执行 8 个轴 PTP 运动。

```
for(int AxisNum =0; AxisNum<8; AxisNum++)
Acm_AxMoveRel( hAxis[AxisNum], 10000); // 对 2 块板卡的 8 个轴做 PTP 运动
```

■ 板卡的 2 轴插补运动

当执行插补运动时，首先需要将轴添加到组中，获取组的 Handle，然后去执行操作，如下所示：

```
HAND    hGp[2];                                // 2 个群组 (Group) 的 Handle 值
double  m_End[2] = {10000, 10000};
// 轴组 0: 添加 2 个轴 (hAxis [0], hAxis [1] )
Acm_GpAddAxis(&hGp[0], hAxis [0]);
Acm_GpAddAxis(&hGp[0], hAxis [1]);
// 轴组 1: 添加 2 个轴 (hAxis [4], hAxis [5] )
Acm_GpAddAxis(&hGp[1], hAxis [4]);
Acm_GpAddAxis(&hGp[1], hAxis [5]);
Acm_GpMoveLinearRel(hGp[0], m_End, 2); //轴组0: 直线插补{10000, 10000}
Acm_GpMoveLinearRel(hGp[1], m_End, 2); //轴组1: 直线插补{10000, 10000}
```

第 6 章

通用运动 API 架构

6.1 简介

本章主要介绍通用运动架构的概念及实现、研华运动控制卡的物件定义、设备编号。

6.2 通用运动架构

为了统一所有研华运动设备的用户接口，研华运动设备采用了新的软件架构，名为“通用运动架构”。该架构定义了所有用户接口和具有的所有运动功能，包括单个轴和多轴。这种统一的编程平台使用户能够以相同的方式操作设备。

该架构包括三层：设备驱动层、整合层和应用层。用户无需了解如何操作特定设备的特定驱动，只需了解通用运动驱动即可。即使支持该架构的设备发生变化，应用也无需修改。

研华通用运动（ACM）架构定义了三种操作对象：设备、轴和群组。每种对象都拥有自己的方法、属性和状态。

开始单轴运动之前，用户需要遵循以下步骤：

打开设备 -> 打开轴 -> 设置轴参数 -> 开始运动。

所有操作都可通过调用相应 ACM API 来完成。设备、轴和群组的常用调用流程由通用运动架构进行定义。详细信息请参考“调用流程”。

(API: Application Programming Interface 应用程序编程接口)

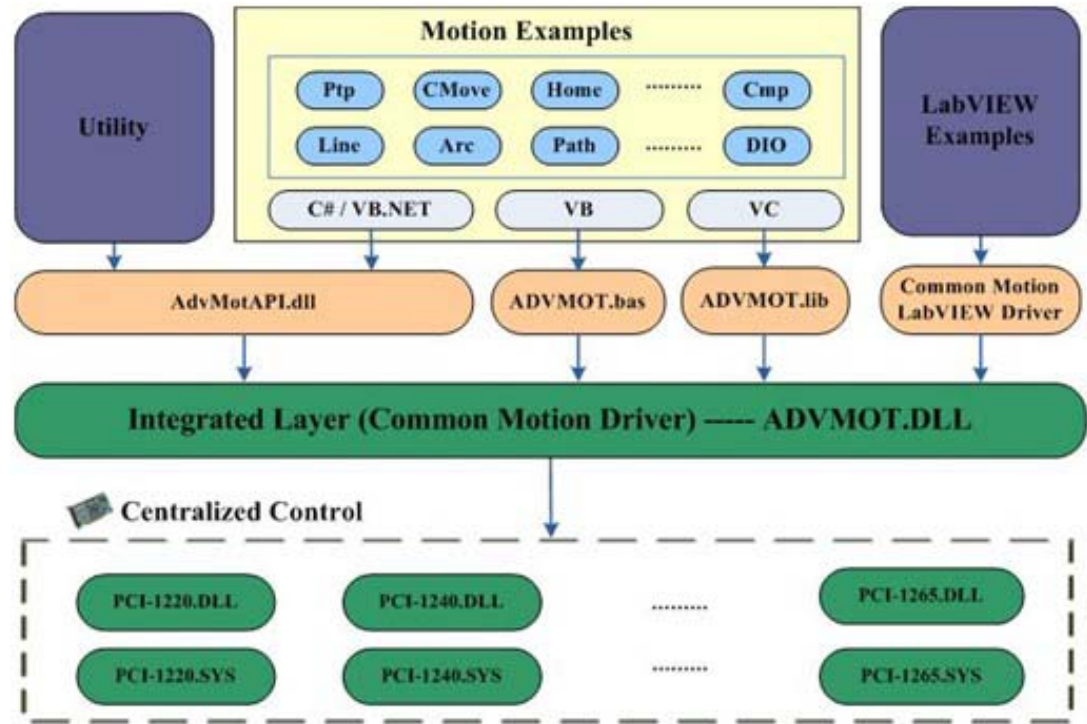


图 6.1: 运动控制卡通用运动架构

6.3 物件定义

命名规则建立在以下三个对象之上：设备对象、轴对象和群组对象。用户将在 API 中遇到许多缩略用法。及说明见下表：

表 6.1：缩写及其含义

缩略语	全称	备注
PPU	每个单元脉冲	运动的一个虚拟单元
Dev	设备	
Ax	轴	
Gp	群组	多轴
Mas	主设备	通信机制中设备的主轴或主卡
Daq	数据采集	AI/AO/DI/DO 的共用名称
Rel	相对	
Abs	绝对	
Cmd	命令	
Vel	速度	
Acc	加速	
Dec	减速	
Emg	紧急	紧急停止
Sd	降低速度	
Info	信息	
Cmp	比较	
Inp	到位	
EZ	编码 Z	
EI	硬件限位	
MeI	负向限位	
PeI	正向限位	
Org	原点	
Ext	外部	
FT	特性	特性属性
CFG	配置	配置属性
PAR	参数	参数属性
Ipo	插补	
Chan	通道	

6.4 设备编号

在 Common Motion 架构中，操作运动控制卡首先需要调用 Acm_DevOpen 打开设备以获取设备句柄，且该 API 需要将设备编号作为输入参数。设备编号由 32 位组成：

第四个字节	第三个字节和第二个高位字节	第二个低位字节	第一个字节
主卡 / 设备类型 ID	主卡 / 设备板卡 ID (或 BaseAddr)	环	从卡 ID

第四个字节：主卡 / 设备类型 ID（参考主卡设备类型 ID 表）

第三个字节和第二个高位字节：主卡 / 设备类型 ID（或基地址），板卡 ID 可通过切换 PCI 卡进行更改

第二个低位字节：远程设备使用的主环编号；将 0 作为本地设备的默认值。

第一个字节：远程设备使用的从卡 ID；将 0 作为本地设备的默认值。

设备编号

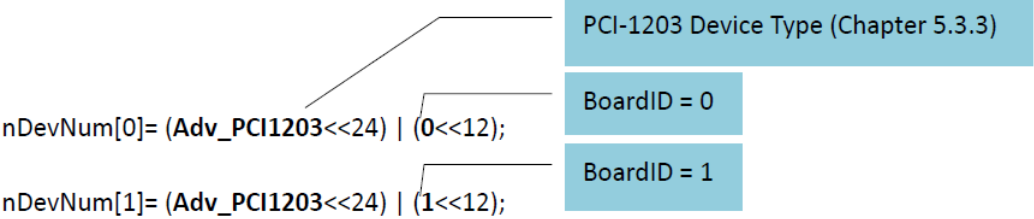
第四字节	第三个字节和第二个高位字节	第二个低位字节	第一个字节
主卡 / 设备类型 ID	设备板卡 ID	0	0

例如，PCI-1203 的 BoardID 为 1，则设备编号（十六进制）则为：

62	001	0	0
----	-----	---	---

因此，设备编号为 0x62001000。
如上所述，在获取板卡设备号时，也可通过如下方式获取。

```
DWORD nDevNum[2]; //Device number of two PCI card
```



第 7 章

属性配置文件说明

7.1 简介

在使用运动控制卡进行各种操作之前，需要对速度、ALM 信号、左极限、右极限等属性进行配置，以满足用户的需求。配置板卡属性的过程称为系统配置过程。

在测试工具 Utility 中，提供“保存 (Save) 和导入 (Load)”配置文件 (*.cfg) 工具项，用于保存和导入所选设备的全部轴的配置信息。用户在编写程序时，通过调用 API 将配置文件的配置信息传递给运动控制器，即可完成运动控制器的配置工作。

7.2 系统配置基本概念

研华运动控制器包含了硬件和软件两部分资源，软硬件资源之间的相互配合，即可实现运动控制器的各种应用。

7.2.1 硬件资源配置

硬件资源配置包括数字量输入 (DI) / 输出 (DO)，模拟量输入 (AI) / 输出 (AO)。

7.2.1.1 数字量输出 (DO)

数字量输出包括如下信号输出：

- 命令脉冲输出（仅支持特定 EtherCAT 从站）
- 伺服报警清除信号 (ERC)
- 比较脉冲输出 (CMP)（仅支持特定 EtherCAT 从站）
- 多轴比较脉冲输出（仅支持特定 EtherCAT 从站）
- 凸轮输出 (CAMDO)

命令脉冲输出（仅支持特定 EtherCAT 从站）

属性	.CFG 文件缩写	说明
CFG_AxPulseOutMode	PlsOutMde	命令脉冲输出模式
CFG_AxPulseOutReverse	PulseOutReverse	开启 / 禁用脉冲输出引脚对调功能

伺服报警清除信号 (ERC)

属性	.CFG 文件缩写	说明
CFG_AxErcLogic	ErcLogic	清除信号的有效逻辑电平
CFG_AxErcEnableMode	ErcEnMde	启用 \ 禁用源轴的报警清除功能
CFG_AxErcOnTime	ErcOnTime	设定报警清除信号开启的时间长度
CFG_AxErcOffTime	ErcOffTime	设定报警清除信号关闭的时间长度

* 运动控制卡暂不支持 CFG_AxErcOnTime、CFG_AxErcOffTime 属性功能。

比较脉冲输出 (CMP)（仅支持特定 EtherCAT 从站）

属性	.CFG 文件缩写	说明
CFG_AxCmpEnable	CmpEnable	启用 / 禁用比较脉冲输出功能
CFG_AxCmpSrc	CmpSrc	比较源
CFG_AxCmpPulseMode	CmpPulseMode	比较模式
CFG_AxCmpMethod	CmpMethod	比较方法
CFG_AxCmpPulseLogic	CmpPulseLogic	比较逻辑电平
CFG_AxCmpPulseWidth	CmpPulseWidth	比较脉冲宽度
CFG_AxCmpPulseWidthEx	CmpPulseWidthEx	比较脉冲宽度扩展

多轴比较脉冲输出（仅支持特定 EtherCAT 从站）

属性	.CFG 文件缩写	说明
CFG_DevMultiCmpAxisEnable	--	启用 / 禁用多轴比较脉冲输出功能
CFG_DevMultiCmpOutChannel	--	启用 / 禁用 OUT5 脉冲输出
CFG_DevMultiCmpPulseLogic	--	多轴比较逻辑电平
CFG_DevMultiCmpPulseWidth	--	多轴比较脉冲宽度
CFG_DevMultiTrigPulseMode	--	多轴比较模式
CFG_DevMultiTrigPWMFreq	--	多轴比较 PWM 输出频率
CFG_DevMultiTrigPWMDuty	--	多轴比较 PWM 输出占空比
CFG_DevMultiTrigAutoEmptyEn	--	启用 / 禁用多轴比较时自动清空比较资料
CFG_AxisMultiCmpDeviation	--	各轴进行比较时的误差范围

凸轮输出 (CAMDO)

属性	.CFG 文件缩写	说明
CFG_AxCamDOEnable	CamDoEnable	启用 / 禁用 CAMDO
CFG_AxCamDOLoLimit	CamDOLoLimit	凸轮区间下边界位置
CFG_AxCamDOHiLimit	CamDOHiLimit	凸轮区间上边界位置
CFG_AxCamDOCmpSrc	CamDoCmpSrc	设定凸轮区间比较源
CFG_AxCamDOLogic	CamDoLogic	凸轮逻辑电平

7.2.1.2 数字量输入 (DI)

数字量输入包括如下信号输入：

- 编码器反馈脉冲输入（仅支持特定 EtherCAT 从站）
- 驱动报警数字量输入 (ALM)
- 到位信号 (IN Position)
- 正负限位数字量输入
- 原点信号数字量输入
- EZ 信号输入
- 外部驱动输入
- 锁存输入信号
- 停止信号输入

编码器反馈脉冲输入（仅支持特定 EtherCAT 从站）

属性	.CFG 文件缩写	说明
CFG_AxPulseInMode	PlsInMde	反馈脉冲输入模式
CFG_AxPulseInLogic	PlsInLogic	反馈脉冲输入逻辑电平
CFG_AxPulseInMaxFreq	PlsInMaxFreq	频率中编码最大脉冲

驱动报警数字量输入 (ALM)

属性	.CFG 文件缩写	说明
CFG_AxAlmEnable	AlmEnable	启用 / 禁用报警功能
CFG_AxAlmLogic	AlmLogic	设定报警信号的有效逻辑电平
CFG_AxAlmReact	AlmReact	设定接收 ALARM 信号时的停止模式
CFG_AxALMFilterTime	ALMFilterTime	设定轴 ALM 信号滤波时间

到位信号 (IN Position)

属性	.CFG 文件缩写	说明
CFG_AxInpEnable	InpEnable	启用 / 禁用到位功能
CFG_AxInpLogic	InpLogic	设定到位信号逻辑电平

正负限位数字量输入

属性	.CFG 文件缩写	说明
CFG_AxEIEnable	EIEnable	启用 / 禁用硬件限位功能
CFG_AxEIReact	EIReact	设定 EL 信号的反应模式
CFG_AxEILogic	EILogic	设定硬件限位信号的逻辑准位
CFG_AxLMTFilterTime	LMTFilterTime	设定轴 LMT+ 信号滤波时间
CFG_AxLMTNFilterTime	LMTNFilterTime	设定轴 LMT- 信号滤波时间

原点信号数字量输入

属性	.CFG 文件缩写	说明
CFG_AxOrgLogic	OrgLogic	设定原点信号逻辑电平
CFG_AxEzLogic	OrgReact	设定回原点反应模式
CFG_AxORGFilterTime	ORGFilterTime	设定轴 ORG 信号滤波时间
PAR_AxHomeCrossDistance	HomeCrossDis	设定 Home 运动中 search 的距离
CFG_AxHomeResetEnable	HomeResetEnable	启用 / 禁用复位逻辑计数器
PAR_AxHomeExSwitchMode	HomeExSwitchMode	设定回原点停止模式
CFG_AxHomeOffsetDistance	HomeOffsetDistance	设定原点跨越距离
CFG_AxHomeOffsetVel	HomeOffsetVel	设定原点跨越速度

EZ 信号输入

属性	.CFG 文件缩写	说明
CFG_AxEzLogic	EzLogic	设定 EZ 信号的有效逻辑电平

外部驱动输入

属性	.CFG 文件缩写	说明
CFG_AxExtMasterSrc	ExtMasterSrc	设定外部驱动源
CFG_AxExtSelEnable	ExtSelEnable	启用 / 禁用外部驱动
CFG_AxExtPulseNum	ExtPulseNum	设定外部驱动脉冲数
CFG_AxExtPulseInMode	ExtPulseInMode	设定外部驱动脉冲输入模式
CFG_AxExtPresetNum	ExtPresetNum	外部驱动个数

锁存输入信号

属性	.CFG 文件缩写	说明
CFG_AxLatchEnable	LatchEn	启用 / 禁用锁存功能
CFG_AxLatchLogic	LatchLogic	设定锁存信号逻辑电平
CFG_AxLatchBufEnable	LatchBufEnable	启用 / 禁用连续锁存
CFG_AxLatchBufMinDist	LatchBufMinDist	设定连续锁存最小距离
CFG_AxLatchBufEventNum	LatchBufEventNum	设定连续锁存事件数
CFG_AxLatchBufSource	LatchBufSource	设定连续锁存源
CFG_AxLatchBufAxisID	LatchBufAxisID	连续锁存轴的 ID
CFG_AxLatchBufEdge	LatchBufEdge	连续锁存触发源

停止信号输入

属性	.CFG 文件缩写	说明
CFG_AxIN1StopEnable	IN1StopEnable	启用 / 禁用 IN1 触发停止功能
CFG_AxIN1StopLogic	IN1StopLogic	设定 IN1 触发停止逻辑电平
CFG_AxIN1StopReact	IN1StopReact	设定 IN1 触发停止模式

滤波信号输入

属性	.CFG 文件缩写	说明
CFG_AxIN1FilterTime	IN1FilterTime	设定轴 IN1 信号滤波时间

紧急停止输入

属性	.CFG 文件缩写	说明
CFG_DevEmgLogic	EmgLogic	设定紧急停止信号的逻辑准位
CFG_DevEmgFilterTime	EmgFilterTime	设定设备 EMG 信号滤波时间

7.2.2 软件资源配置

轴的软件资源包括如下配置：

- 轴的速度参数配置
- 轴的 PPU
- 轴的软极限
- 轴的背隙补偿
- 轴的单圈脉冲数 (ModuleRange)
- 轴的同步启停
- 轴的回原点速度参数
- JOG 速度参数
- 容差
- 轴的减速
- 错误计算

轴的速度参数配置

属性	.CFG 文件缩写	说明
PAR_AxVelLow	VelLow	轴的初速度
PAR_AxVelHigh	VelHigh	轴的运行速度
PAR_AxAcc	Acc	轴的加速度
PAR_AxDec	Dec	轴的减速度
CFG_AxMaxVel	MaxVel	配置轴的最大运行速度
CFG_AxMaxDec	MaxDec	配置轴的最大减速度
CFG_AxMaxAcc	MaxAcc	配置轴的最大加速度
CFG_AxMaxJerk	MaxJerk	配置轴的最大加加速度

轴的 PPU

属性	.CFG 文件缩写	说明
CFG_AxPPU	PlsPerUnit	PPU 分子
CFG_AxPPUDenominator	PPU Denominator	PPU 分母

轴的软极限

属性	.CFG 文件缩写	说明
CFG_AxSwMelEnable	SwMelEnable	启用 / 禁用负方向软件限位功能
CFG_AxSwPelEnable	SwPelEnable	启用 / 禁用正方向软件限位功能
CFG_AxSwMelReact	SwMelReact	设定负方向软件限位的反应模式
CFG_AxSwPelReact	SwPelReact	设定正方向软件限位的反应模式
CFG_AxSwMelValue	SwMelValue	设定负方向软件限位的值
CFG_AxSwPelValue	SwPelValue	设定正方向软件限位的值

轴的背隙补偿

属性	.CFG 文件缩写	说明
CFG_AxBacklashEnable	BacklashEnable	启用 / 禁用背隙补偿
CFG_AxBacklashPulses	BacklashPulses	设定补偿脉冲个数
CFG_AxBacklashVel	BacklashVel	设定背隙补偿的速度

轴的单圈脉冲个数 (ModuleRange)

属性	. CFG 文件缩写	说明
CFG_AxModuleRange	ModuleRange	设定当轴旋转 360° 时的脉冲个数

轴的同步启停

属性	. CFG 文件缩写	说明
CFG_AxSimStartSource	SimStartSource	设定当前轴的同步启停模式

轴的回原点速度参数

属性	. CFG 文件缩写	说明
PAR_AxHomeVelLow	HomeVelLow	Home 执行时的初速度
PAR_AxHomeVelHigh	HomeVelHigh	Home 执行时的运行速度
PAR_AxHomeAcc	HomeAcc	Home 执行时的加速度
PAR_AxHomeDec	HomeDec	Home 执行时的减速度
PAR_AxHomeJerk	HomeJerk	设定 Home 执行时的速度的曲线类型

JOG 速度参数配置

属性	. CFG 文件缩写	说明
CFG_AxJogVTime	JogVTime	执行 JOG 时低速运转保持的时间
CFG_AxJogVelLow	JogVelLow	Jog 执行时的初速度
CFG_AxJogVelHigh	JogVelHigh	Jog 执行时的运行速度
CFG_AxJogAcc	JogAcc	Jog 执行时的加速度
CFG_AxJogDec	JogDec	Jog 执行时的减速度
CFG_AxJogJerk	JogJerk	设定 Jog 执行时的速度的曲线类型

容差

属性	. CFG 文件缩写	说明
CFG_AxMelToleranceEnable	MelToleranceEnable	启用 / 禁用负向极限误差功能
CFG_AxMelToleranceValue	MelToleranceValue	设定负向极限误差值
CFG_AxPelToleranceEnable	PelToleranceEnable	启用 / 禁用正向极限误差功能
CFG_AxPelToleranceValue	PelToleranceValue	设定正向极限误差值
CFG_AxSwMelToleranceEnable	SwMelToleranceEnable	启用 / 禁用软件负向极限误差功能
CFG_AxSwMelToleranceValue	SwMelToleranceValue	设定软件负向极限误差值
CFG_AxSwPelToleranceEnable	SwPelToleranceEnable	启用 / 禁用软件正向极限误差功能
CFG_AxSwPelToleranceValue	SwPelToleranceValue	设定软件正向极限误差值

轴的减速

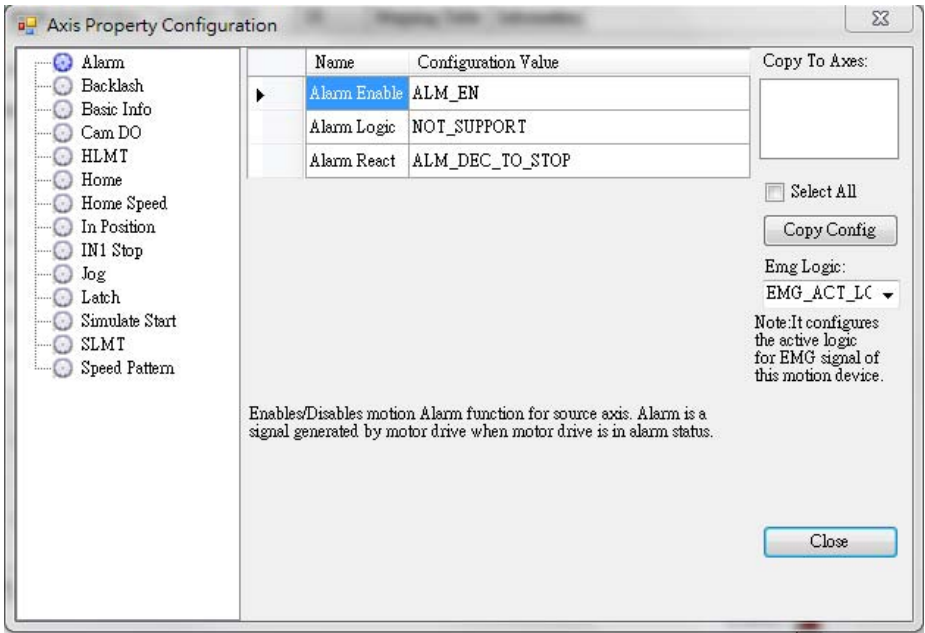
属性	. CFG 文件缩写	说明
CFG_AxKillDec	KillDec	设定 DI Stop 的减速度

错误计算

属性	. CFG 文件缩写	说明
CFG_AxMaxErrorCnt	MaxErrorCnt	Feedback 值和 Command 值差值报警数

7.2.3 资源整合

将上述的硬件资源和软件资源相互组合，并对各个资源的基本属性进行配置的过程称为资源整合，以实现相应的功能需求。可通过测试工具 Utility 配置上述属性，如下图所示：有关属性配置的详细信息，请参考附录。



```
62007000 - Notepad
File Edit Format View Help
[[Device Config]
EmgLogic=0
EmgFilterTime=0
[Group Config]
GpvelLow=1000
GpvelHigh=8000
GpAcc=50000
GpDec=50000
GpJerk=0
SFEnable=0
BldTime=0
[AXIS 0 Config]
PlsPerUnit=1
Maxvel=20000000
MaxAcc=2000000000
MaxDec=2000000000
```

7.3 配置文件生成与下载

对于硬件和软件资源的配置，可通过 Utility 实现。配置完成后，点击 Utility 界面上的保存 (save) 按钮，即可将配置文件保存。

同时，点击 Load 按钮，也可将配置文件导入到 Utility 中，则配置文件中的参数设置将生效。

7.3.1 配置文件下载函数

用户可通过调用如下 API 将配置文件导入到自己的工程项目中。

属性	说明
Acm_DevLoadConfig	根据加载的配置文件，设置设备的所有配置。

7.3.2 配置文件重点说明

如上所述，用户通过 Utility 配置属性文件，调用函数将保存的配置文件导入到工程项目中。配置文件可以为二进制或文本文件。如果文件扩展名为 .bin，则驱动会以二进制格式读取文件。否则，驱动将以 .INI (文本格式) 读取文件。

如下为通过 Utility 生成的配置文件，其中 AXIS 0 Config 表示 0 轴的属性配置。例如，PCI-1245 共 4 个轴，则配置文件中将有 AXIS 0 Config ~AXIS 3 Config 的属性配置信息。章节 7.2 详细介绍了该配置文件的参数名称所对应的属性。



```

2a00f000.cfg - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

[Device Config]
EmgLogic=0
EmgFilterTime=0
[AXIS 0 Config]
PisPerUnit=1
MaxAcc=50000000
MaxVel=5000000
MaxDec=50000000
MaxJerk=1
VelHigh=8000
VelLow=2000
Acc=10000
Dec=10000
  
```

用户可直接修改该配置文件中相应的属性值，如 MaxAcc=400000，调用 Acm_DevLoadConfig 函数导入修改后的配置文件，该属性值将生效，板卡的最大加速度将设置为 400000。

7.4 配置信息修改

用户除了用上述的配置文件修改配置信息外，也可用章节 7.2 中列出的属性修改相关的配置信息。

7.5 控制器配置初始化状态

7.5.1 硬件资源配置初始化状态、

控制器硬件资源配置初始状态如下表所示：

资源	配置选项	默认状态	相关指令
板卡	紧急停止输入电平	高电平	CFG_DevEmgLogic
	EMG 信号滤波时间	5us	CFG_DevEmgFilterTime
轴脉冲输出	脉冲输出模式	CW/CCW	CFG_AxPulseOutMode
	脉冲输出翻转	禁用	CFG_AxPulseOutReverse
轴比较输出	启用 / 禁用比较功能	禁用	CFG_AxCmpEnable
	比较源	理论位置	CFG_AxCmpSrc
	比较脉冲模式	脉冲模式	CFG_AxCmpPulseMode
	比较方法	大于等于位置计数器	CFG_AxCmpMethod
	比较逻辑电平	高电平	CFG_AxCmpPulseLogic
	比较脉冲宽度	5 us	CFG_AxCmpPulseWidth
	比较脉冲宽度扩展	0	CFG_AxCmpPulseWidthEx
	启用 / 禁用 DO 输出	禁用	CFG_AxCamDOEnable
凸轮输出	设定凸轮区间下边界	10,000	CFG_AxCamDOLoLimit
	设定凸轮区间上边界	10,000	CFG_AxCamDOHiLimit
	设定凸轮区间比较源	理论位置	CFG_AxCamDOCmpSrc
	凸轮逻辑电平	高电平	CFG_AxCamDOLogic
轴报警信号输出	错误清除信号电平	高电平	CFG_AxErcLogic
	启用 / 禁用清除功能	禁用	CFG_AxErcEnableMode
	错误清除开启时间	12us	CFG_AxErcOnTime
	错误清除关闭时间	0us	CFG_AxErcOffTime
轴的通用 DO	启用 / 禁用通用 DO	禁用	CFG_AxGenDOEnable

编码器反馈 脉冲输入	脉冲输入模式	4XAB	CFG_AxPulseOutMode
	脉冲输入电平	不倒转方向	CFG_AxPulseInLogic
	脉冲输入源	不支持	CFG_AxPulseInSource
	脉冲输入最大频率	1 MHz	CFG_AxPulseInMaxFreq
驱动报警 数字量输入	启用 / 禁用报警功能	禁用	CFG_AxAlmEnable
	设定报警信号的有效逻辑电平	高电平	CFG_AxAlmLogic
	设定报警后停止模式	减速停止	CFG_AxAlmReact
	设定轴报警信号滤波时间	5us	CFG_AxALMFilterTime
到位信号输入	启用 / 禁用到位功能	禁用	CFG_AxInpEnable
	设定到位信号逻辑电平	高电平	CFG_AxInpLogic
正负限位数 字量输入	启用 / 禁用硬件限位功能	启用	CFG_AxEIEnable
	设定 EL 信号的响应模式	立即停止	CFG_AxEIReact
	设定硬件限位信号的逻辑准位	低电平	CFG_AxEILogic
	设定轴正限位信号滤波时间	5us	CFG_AxLMTFilterTime
	设定轴负限位信号滤波时间	5us	CFG_AxLMTNFilterTime
ORG 信号 数字量输入	设定原点信号逻辑电平	低电平	CFG_AxOrgLogic
	设定回原点反应模式	减速停止	CFG_AxEzLogic
	设定 Home 运动中 search 的距离	10,000	PAR_AxHomeCrossDistance
	启用 / 禁用复位逻辑计数器	启用	CFG_AxHomeResetEnable
	设定回原点停止模式	减速停止	PAR_AxHomeExSwitchMode
EZ 信号输入	设定 EZ 信号的有效逻辑电平	低电平	CFG_AxEzLogic
外部驱动输入	设定外部驱动源	Axis 0	CFG_AxExtMasterSrc
	启用 / 禁用外部驱动	禁用	CFG_AxExtSelEnable
	设定外部驱动脉冲数	1	CFG_AxExtPulseNum
	设定外部驱动脉冲输入模式	4XAB	CFG_AxExtPulseInMode
	外部驱动个数	1	CFG_AxExtPresetNum
锁存输入信号	启用 / 禁用锁存功能	禁用	CFG_AxLatchEnable
	设定锁存信号逻辑电平	高电平	CFG_AxLatchLogic
	启用 / 禁用连续锁存	禁用	CFG_AxLatchBufEnable
	设定连续锁存最小距离	1,000	CFG_AxLatchBufMinDist
	设定连续锁存事件数	128	CFG_AxLatchBufEventNum
	设定连续锁存源	理论位置	CFG_AxLatchBufSource
	连续锁存轴的 ID	Axis 0	CFG_AxLatchBufAxisID
	连续锁存触发源		CFG_AxLatchBufEdge
紧急停止信号	启用 / 禁用 IN1 触发停止功能	禁用	CFG_AxIN1StopEnable
	设定 IN1 触发时的停止模式	减速停止	CFG_AxIN1StopLogic
	设定 IN1 触发停止逻辑电平	高电平	CFG_AxIN1StopReact
	设定轴 IN1 信号滤波时间	5us	CFG_AxIN1FilterTime

7.5.2 软件资源配置初始化状态

控制器软件资源配置初始化状态如下表所示：

资源	配置选项	默认状态	相关指令
轴的 速度参数	轴的初速度	2,000	PAR_AxVelLow
	轴的运行速度	8,000	PAR_AxVelHigh
	轴的加速度	10,000	PAR_AxAcc
	轴的减速度	10,000	PAR_AxDec
	设定轴的最大运行速度	20,000,000	CFG_AxMaxVel
	设定轴的最大减速度	2,000,000,000	CFG_AxMaxDec
	设定轴的最大加速度	2,000,000,000	CFG_AxMaxAcc
	设定轴的最大加加速度	1	CFG_AxMaxJerk
轴的 PPU	PPU 分子	1	CFG_AxPPU
	PPU 分母	1	CFG_AxPPUDenominator
轴的 软极限	启用 / 禁用负方向软件限位功能	启用	CFG_AxSwMelEnable
	启用 / 禁用正方向软件限位功能	启用	CFG_AxSwPelEnable
	设定负方向软件限位的反应模式	减速停止	CFG_AxSwMelReact
	设定正方向软件限位的反应模式	减速停止	CFG_AxSwPelReact
	设定负方向软件限位的值	-10,000,000	CFG_AxSwMelValue
	设定正方向软件限位的值	10,000,000	CFG_AxSwPelValue
轴的 背隙补偿	启用 / 禁用背隙补偿	禁用	CFG_AxBacklashEnable
	设定补偿脉冲个数	10	CFG_AxBacklashPulses
	设定背隙补偿的速度	1,000	CFG_AxBacklashVel
轴的单圈脉冲数	设定单圈脉冲数范围	0	CFG_AxModuleRange
同步启停	设定当前轴的同步启停模式	禁用	CFG_AxSimStartSource
回原点速度 参数	Home 执行时的初速度	2,000	PAR_AxHomeVelLow
	Home 执行时的运行速度	8,000	PAR_AxHomeVelHigh
	Home 执行时的加速度	10,000	PAR_AxHomeAcc
	Home 执行时的减速度	10,000	PAR_AxHomeDec
	设定 Home 执行时的速度的曲线类型	T 型	PAR_AxHomeJerk
Jog 运动速度 参数	执行 JOG 时低速运转保持的时间	5,000	CFG_AxJogVLTime
	Jog 执行时的初速度	2,000	CFG_AxJogVelLow
	Jog 执行时的运行速度	8,000	CFG_AxJogVelHigh
	Jog 执行时的加速度	10,000	CFG_AxJogAcc
	Jog 执行时的减速度	10,000	CFG_AxJogDec
	设定 Jog 执行时的速度的曲线类型	T 型	CFG_AxJogJerk

容差	启用 / 禁用负向极限误差功能	禁用	CFG_AxMeIToleranceEnable
	设定负向极限误差值	5, 000	CFG_AxMeIToleranceValue
	启用 / 禁用正向极限误差功能	禁用	CFG_AxPeIToleranceEnable
	设定正向极限误差值	5, 000	CFG_AxPeIToleranceValue
	启用 / 禁用软件负向极限误差功能	禁用	CFG_AxSwMeIToleranceEnable
	设定软件负向极限误差值	5, 000	CFG_AxSwMeIToleranceValue
	启用 / 禁用软件正向极限误差功能	禁用	CFG_AxSwPeIToleranceEnable
	设定软件正向极限误差值	5, 000	CFG_AxSwPeIToleranceValue
轴的减速	设定 DI Stop 的减速度	100, 000	CFG_AxKillDec
差值报警	设置 Feedback 值和 Command 值差值报警数	0 (不报警)	CFG_AxMaxErrorCnt

7.6 伺服操作

打开 / 关闭电机函数如下表所示：

功能	说明
Acm_AxSetSvOn	打开 / 关闭伺服驱动器

调用 Acm_AxSetSvOn 函数，将打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。

请参考如下例程：

```

U32      Result;
HAND     m_Axishand[32];           // 轴句柄
Result =Acm_AxSetSvOn(m_Axishand[0], 0); // 打开 0 轴的伺服
Result =Acm_AxSetSvOn(m_Axishand[0], 1); // 关闭 0 轴的伺服

```

7.7 脉冲输入输出方式

7.7.1 脉冲输入输出方式相关属性

脉冲输入输出方式相关属性如下表所示：

属性	说明
FT_AxPulseInMap	获取该运动设备支持的脉冲输入特性
FT_AxPulseInModeMap	获取轴支持的脉冲输入模式
CFG_AxPulseInMode	设置 / 获取编码器反馈脉冲输入模式
CFG_AxPulseInLogic	设置 / 获取编码器返回脉冲的逻辑
CFG_AxPulseInMaxFreq	设置 / 获取频率中编码最大脉冲
FT_AxPulseOutMap	获取该运动设备支持的脉冲输出特性
FT_AxPulseOutModeMap	获取该运动设备支持的脉冲输出模式
CFG_AxPulseOutMode	设置 / 获取命令脉冲输出模式
CFG_AxPulseOutReverse	启用 / 禁用脉冲输出引脚对调功能

用户可通过设置 / 获取以上相关属性，实现脉冲输入输出方式。

7.8 浮点 PPU

7.8.1 浮点 PPU 相关属性

浮点 PPU 相关属性如下表所示：

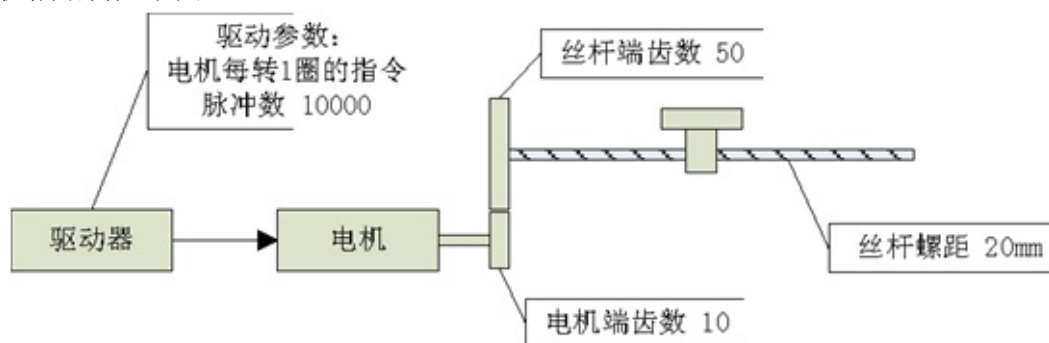
属性	说明
CFG_AxPPU	设定 PPU 的分子
CFG_AxPPUDenominator	设定 PPU 的分母

7.8.2 浮点 PPU 功能介绍

PPU：板卡的虚拟单位，方便不同机构的脉冲输出。

PPU 的值 = CFG_AxPPU 属性值 / CFG_AxPPUDenominator 属性值

机械结构如下图：



对于上述机械机构和驱动参数，PPU 的设定推导如下：

机构条件：

丝杆螺距：20mm

丝杆端齿数为 50，电机端齿数为 10，即丝杆转 1 圈，电机需要转 50/10=5 圈。

驱动器参数为电机每转 1 圈的指令脉冲数为 10000。

若用户预设单位为 1 毫米 (mm) 时，则依照上列机构条件，可以得到以下计算公式：

$$PPU = (10000 * 50) / (10 * 20)$$

化简后可得到：

$$PPU = 500000 / 200 = 2500$$

即实际机构移动每毫米 (mm) 轴卡需输出的脉冲数为 2500。

因此可设置：

CFG_AxPPU 属性值为 2500

CFG_AxPPUDenominator 属性值为 1

浮点 PPU 是当用户的距离单位对应的 PPU 不能为整数时，可通过设置浮点 PPU 实现。

例如客户移动 10mm，需要 1005 个脉冲，所以每移动 1mm，所需要的脉冲数为 100.5 个 pulse。为了解决距离误差，可通过设定 PPU 的分母，得到 double 类型的实际 PPU 参与距离与速度等的计算中。

7.8.3 浮点 PPU 重点说明

PPU 分子和 PPU 分母属性需首先设定，否则会影响其他 PPU 相关属性如 HomeCrossDistance。通过 PPU 分子和 PPU 分母可实现实际 PPU 为浮点值的问题，但是因为内部机制的限制，尚存在一些限制。例如，PPU 分子 / PPU 分母得到实际 PPU 为 3.33333333...，dis = 100.0PPU，当内部将 dis 转成实际 pulse 数时取整为：100.0 * 3.33333333... 为 333，最终得到的位置转成实际 PPU 后为 333 / 3.33333333... 约为 99.9。得到的值与设定的值略有差别。

该属性值的变化将影响 CFG_AxMaxVel、CFG_AxMaxAcc、CFG_AxMaxDec、PAR_AxVelHigh、PAR_AxVelLow、PAR_AxAcc、PAR_AxDec、PAR_GpVelHigh、PAR_GpVelLow、PAR_GpAcc、PAR_GpDec 和 PAR_HomeCrossDistance, 以及所有的移动距离（以上这些都是使用实际 PPU 来计算）。

默认情况下, PPU 分子与分母为 1。

7.8.4 例程

如需要设定 X 轴实际 PPU 为 100.5 个脉冲。

```
U32 Result;
```

```
Result =Acm_SetU32Property(m_Axishand[0], CFG_AxPPU, 201);
```

```
Result =Acm_SetU32Property(m_Axishand[0], CFG_AxPPUDenominator, 2);
```

第 8 章

运动状态

8.1 简介

当用户连接好运动控制器、驱动器、电机后，可通过调用函数获取该运动系统的运动状态及速度参数，如当前的运动位置、运动速度、加减速速度等。本章主要介绍如果获取板卡的运动信息。

8.2 轴状态

8.2.1 轴状态相关函数

用户可以从板卡的状态寄存器中读取轴的状态，获取轴状态的相关函数。
如下表所示：

函数	说明
Acm_AxGetState	获取轴的当前状态
Acm_AxGetMotionStatus	获取轴的当前运动状态

8.2.2 重点说明

8.2.2.1 轴当前状态重点说明

当调用 Acm_AxGetState 函数获取轴的当前状态后，将返回 16 位的轴状态字轴的状态定义如下表所示：

表 8.1：轴当前状态定义	
位	定义
0	STA_AxDisable，轴被禁用，用户可打开并激活
1	STA_AxReady，轴已准备就绪，等待新的命令
2	STA_Stopping，轴停止
3	STA_AxErrorStop，出现错误，轴停止
4	STA_AxHoming，轴正在执行返回原点运动
5	STA_AxPtpMotion，轴正在执行 PTP 运动
6	STA_AxContiMotion，轴正在执行连续运动
7	STA_AxSyncMotion，轴在一个群组中，群组正在执行插补运动；或轴是一个从轴，正在执行 E-cam/E-gear/Gantry 运动。
8	STA_AX_EXT_JOG，轴由外部信号控制。当外部信号激活时，轴将执行 JOG 模式运动
9	STA_AX_EXT_MPG，轴由外部信号控制。当外部信号激活时，轴将执行 MPG 模式运动

当打开板卡，打开轴后，正常情况下，轴的状态将为 Ready。只有当轴的状态为 Ready 时，才能执行新的运动操作，如连续运动。

当打开板卡未打开轴时，轴的状态为禁用 (STA_AxDisable)，此时执行运动操作将会报错。

8.2.2.2 轴当前运动状态重点说明

当调用 Acm_AxGetMotionStatus 函数，获取轴的当前运动状态时，将返回 32 位的轴当前运动状态，轴的当前运动状态如下定义。

表 8.2：轴当前运动状态定义

位	定义
0	Stop, 停止
1	Res1, 保留
2	WaitERC, 等待 ERC 完成
3	Res2, 保留
4	CorrectBksh, 背隙补偿
5	Res3, 保留
6	InFA, 处于特定速度中 = FA
7	InFL, 处于低速中 = FL
8	InACC, 加速中
9	WaitINP, 到位等待

8.2.3 例程

该实现打开板卡和轴后，获取轴的当前状态

```
HAND    m_Axishand; //0 轴的句柄
U32     Ret;         // 函数返回值
U16     State;
CString strTemp;
--- 初始化过程 ---
Ret = Acm_AxGetState(m_Axishand, &State);
if (Ret == SUCCESS)
{switch (State)
{
    case 0:
        strTemp.Format("STA_AX_DISABLE");
        break;
    case 1:
        strTemp.Format("STA_AX_READY");
        break;
    case 2:
        strTemp.Format("STA_AX_STOPPING");
        break;
    case 3:
        strTemp.Format("STA_AX_ERROR_STOP");
        break;
    case 4:
        strTemp.Format("STA_AX_HOMING");
        break;
    case 5:
        strTemp.Format("STA_AX_PTP_MOT");
        break;
    case 6:
```

```

        strTemp.Format("STA_AX_CONTI_MOT");
        break;
    case 7:
        strTemp.Format("STA_AX_SYNC_MOT");
        break;
case 8:
        strTemp.Format("STA_AX_EXT_JOG ");
        break;
case 9:
        strTemp.Format("STA_AX_EXT_MPG ");
        break;
    default:
        break;
    }
}

```

8.3 轴速度

8.3.1 轴速度相关函数与属性

轴速度相关函数与属性如下表所示：

函数	说明
Acm_AxChangeVel	当轴在运动过程中，命令轴改变运行速度
Acm_AxGetCmdVelocity	获取指定轴的当前运行速度
Acm_AxChangeVelEx	改变运行轴的运行速度、加速度、减速度
Acm_AxChangeVelExByRate	按比率改变运行轴的运行速度、加速度、减速度
Acm_AxChangeVelByRate	按比例改变运行轴的运行速度
FT_AxMaxVel	获取轴支持的最大速度
FT_AxMaxAcc	获取轴支持的最大加速度
FT_AxMaxDec	获取轴支持的最大减速度
FT_AxMaxJerk	获取轴支持的最大减速度
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度
CFG_AxMaxJerk	获取运动轴的最大加加速度配置
PAR_AxVelLow	设置 / 获取该轴的起始速度
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T 形曲线或 S 形曲线

8.3.2 轴速度重点说明

打开板卡和轴后，可设置 / 获取轴的当前速度值，如初速度、加速度、运行速度。关于属性的设置、获取可参考附录 B。

设置的速度参数不能大于运动轴的最大速度参数，否则会报错，例如配置轴的最大速度 (CFG_AxMaxVel) 为 10000，设置轴的运行速度 (PAR_AxVelHigh) 为 11000，运行时将会提示报错。

在轴的运行过程中，可调用函数改变运行轴的运行速度、加速度、减速度，详见。

8.3.3 例程

如下例程实现设置 0 轴的速度参数及运动过程中改变轴的运行速度。

VC 代码如下：

```
HAND    m_Axishand;                                //0 轴的 handle
U32      Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
Ret  = Acm_SetF64Property(m_Axishand, PAR_AxVelLow, 2000); // 起始速度 2000
Ret  = Acm_SetF64Property(m_Axishand, PAR_AxVelHigh, 8000); // 运行速度 8000
Ret  = Acm_SetF64Property(m_Axishand, PAR_AxAcc, 10000);    // 加速度 10000
Ret  = Acm_SetF64Property(m_Axishand, PAR_AxDec, 10000);    // 减速度 10000
Ret  = Acm_SetF64Property(m_Axishand, PAR_AxJerk, 0);        //T 型曲线
// 执行相对点到点运动，运动过程中改变运行速度
Ret  = Acm_AxMoveRel(m_Axishand[0], 10000); // 相对点到点运动，目标位置 10000
Ret  = Acm_AxChangeVel(m_Axishand[0], 6000); // 改变运行速度为 6000PPU/S
```

8.4 轴的运动 I/O 状态

8.4.1 轴的运动 I/O 状态函数

获取轴的运动 I/O 状态函数如下表所示：

函数	说明
Acm_AxGetMotionIO	获取轴的运动 I/O 状态

8.4.2 轴的运动 I/O 状态重点说明

用户可通过调用 Acm_AxGetMotionIO 函数获取轴的运动 I/O 状态，调用该函数后，将返回 32 位的轴状态字，该轴的状态字定义如下表所示：

位	定义	说明
0	RDY	RDY 针脚输入
1	ALM	报警信号输入
2	LMT+	限位开关 +
3	LMT-	限位开关 -
4	ORG	原始开关
5	DIR	DIR 输出
6	EMG	紧急信号输入
7	PCS	PCS 信号输入
8	ERC	输出偏转计数器清除信号至伺服电机驱动
9	EZ	编码器 Z 信号
10	CLR	外部输入至清除位置计数器
11	LTC	锁存信号输入
12	SD	减速信号输入
13	INP	到位信号输入
14	SVON	伺服开启（OUT6）
15	ALRM	报警复位输出状态
16	SLMT+	软件限位 +
17	SLMT-	软件限位 -
18	CMP	比较信号
19	CAMDO	凸轮区间 D0

驱动器报警标志、限位触发标志触发以后，不会自动清 0。只有当产生异常的原因消除后，调用 Acm_AxResetError 函数，轴的状态变为 Ready。

8.4.3 例程

实现打开板卡和轴后，获取 0 轴的运动 I/O 状态，判断是否发生报警、回原点、左极限、右极限信号。VC 代码如下：

```
HAND    m_Axishand;    //0 轴的 handle
U32     Ret;           // 函数返回值
U16     Status;
CString strTemp;
--- 初始化过程 ---
Ret = Acm_AxGetMotionIO (m_Axishand,& Status);
if (Ret ==SUCCESS)
{
    if (Status & AX_MOTION_IO_ALM)    //ALM
    { // 报警信号输出 }
    if (Status & AX_MOTION_IO_ORG)    //ORG
    { // 回原点信号输出 }
    if (Status & AX_MOTION_IO_LMTP) //+EL
    { // 右极限信号输出 }
    if (Status & AX_MOTION_IO_LMTN) //-EL
    { // 左极限信号输出 }
}
```

8.5 群组状态

8.5.1 群组状态相关函数

用户可以从板卡的状态寄存器中读取群组的状态，获取群组状态的相关函数如下表所示：

函数	说明
Acm_GpGetState	获取群组的当前状态

8.5.2 重点说明

当调用 Acm_GpGetState 函数获取群组的当前状态后，将返回 16 位的群组状态字，群组状态定义如下表所示：

表 8.3：群组当前状态定义	
位	定义
0	STA_GP_DISABLE，群组状态为 Disable，不能执行群组运动
1	STA_GP_READY，群组已准备好，等待新的命令
2	STA_GP_STOPPING，群组停止
3	STA_GP_ERROR_STOP，出现错误，群组停止
4	STA_GP_MOTION，群组正在运行
5	STA_GP_AX_MOTION（不支持）
6	STA_GP_MOTION_PATH，群组正在执行 Path 运动

当打开板卡，打开轴后，需要调用函数添加轴到群组中，当群组中轴的个数大于等于 1 时，正常情况下，群组的状态将为 Ready，此时可执行群组操作。当群组的状态为 Disable 时，此时执行群组运动操作将会报错。

8.5.3 例程

实现打开板卡和轴后，添加轴到群组中，获取群组的当前状态。

```
HAND    m_GpHand; // 组的 handl
--- 初始化过程 ---
U32     Ret;       // 函数返回值
U16     GpState;
CString strTemp;
Ret =  Acm_GpGetState(m_GpHand, &GpState);
if (Ret ==SUCCESS)
{
    switch (GpState)
    {
    case 0:
        strTemp.Format("STA_GP_DISABLE ");
        break;
    case 1:
        strTemp.Format("STA_GP_READY ");
        break;
    case 2:
        strTemp.Format("STA_GP_STOPPING ");
        break;
    case 3:
        strTemp.Format("STA_GP_ERROR_STOP ");
        break;
    case 4:
        strTemp.Format("STA_GP_AX_MOTION ");
        break;
    case 5:
        strTemp.Format("STA_GP_MOTION_PATH ");
        break;
    }
}
```

8.6 群组速度

8.6.1 群组速度相关函数与属性

群组速度相关函数与属性如下表所示：

函数	说明
Acm_GpGetCmdVel	获取群组的当前的速度值
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线

8.6.2 群组速度重点说明

板卡初始化后，可设置 / 获取群组的当前速度值，如初速度，加速度，运行速度。运行过程中，可调用 Acm_GpGetCmdVel 函数获取群组的当前速度值。

8.6.3 例程

如下例程实现设置群组的速度参数及运动过程中获取群组的运行速度。

VC 代码如下：

```
HAND    m_GpHand;
U32     Ret;
U32     m_CW=0           // 顺时针执行圆弧运动
double  CenterArray[2]={8000,0};
Double  EndArray[2]={16000,0};
U32     AxisNum =2;      // 运行轴数
double  CmdVel;          // 群组运行速度
--- 初始化过程 ---
// 设置速度参数如下代码
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 起始速度 2000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000);   // 加速度 10000
Ret  = Acm_SetF64Property(m_GpHand, PAR_AxDec, 10000);   // 减速度 10000
// 执行圆弧运动，
Ret  = Acm_GpMoveCircularRel(m_GpHand, CenterArray, EndArray, &AxisNum, m_CW);
// 运动过程中获取群组速度
Ret  = Acm_GpGetCmdVel(m_GpHand, &CmdVel);
```


第 9 章

运动功能

9.1 简介

本章介绍单轴运动模式、插补运动模式、跟随运动模式、路径规划模式。

9.2 单轴运动模式

9.2.1 本节简介

单轴运动模式指规划单轴运动的方式，包括如下运动模式：

- 点到点运动
- 连续运动
- 变位运动
- 变速运动
- 同步起、同步停运动
- 叠加运动
- JOG 运动
- 原点复归

9.2.2 点到点运动

9.2.2.1 点到点运动相关函数与属性

点到点运动相关函数如下表 9.1 所示，该表中的函数可在应用程序中直接调用。

表 9.1：点到点运动相关函数	
点到点运动相关函数	说明
Acm_AxMoveAbs	单轴的绝对点到点运动
Acm_AxMoveRel	单轴的相对点到点运动
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_AxStopDec	命令轴按设定的减速度停止运行
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

点到点运动相关属性如下表 9.2 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.2：点到点运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置 / 获取速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

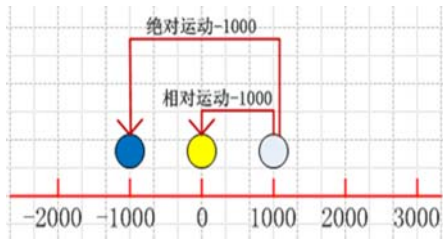
9.2.2.2 重点说明

当轴的状态为 Ready 时，将能执行点到点运动。关于轴的状态详见 6.2 节。

点到点运动分为两种：相对运动和绝对运动。

相对运动：以当前的理论指令位置为参考位置作位置的移动。

绝对运动：以绝对零点指令位置为参考位置作位置的移动。



如上图所示，当前轴的控制部件停在 1000 pulse 的位置，如此时下相对运动 -1000 pulse 的函数指令，则控制部件将停在 0 pulse 的位置；如此时下绝对运动 -1000 pulse 的函数指令，则控制部件将停在 -1000 pulse 的位置，相当于控制部件向负方向移动了 2000 个 pulse 的距离。

点到点运动模式下，可单独设置各轴的目标位置、初速度、加速度等运动参数，各轴独立运行或停止。在运动过程中，可改变目标位置和运行速度（9.2.4 和 9.2.5 章节）。

9.2.2.3 点到点运动流程图

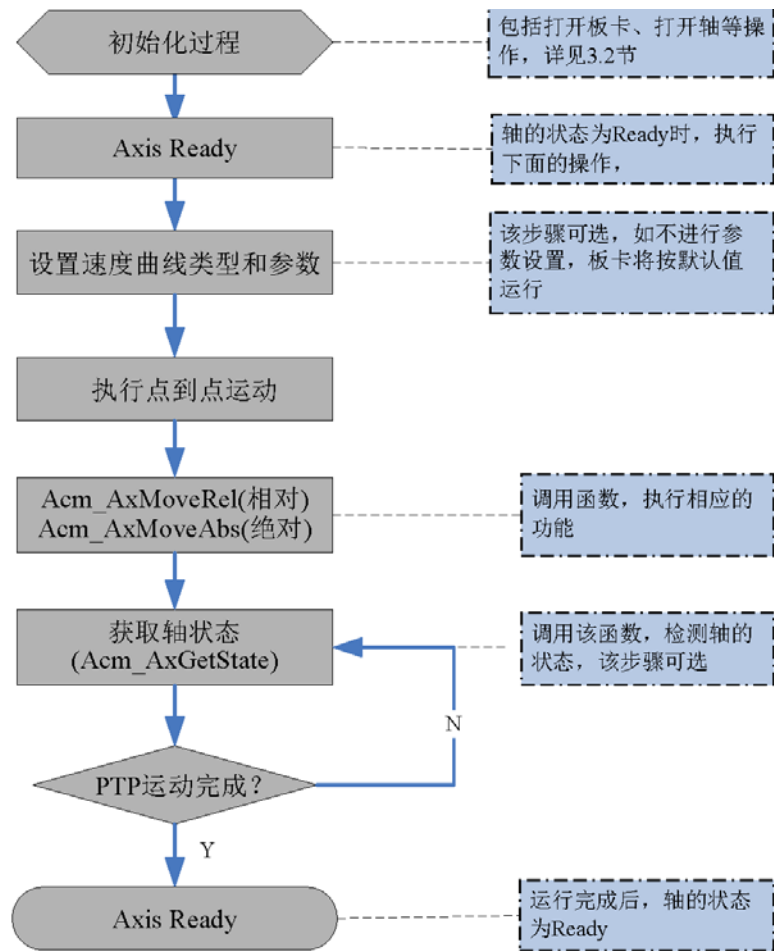


图 9.1： 点到点运动流程图

9.2.2.4 例程

该例程实现 0 轴执行相对点到点运动，设置的速度等参数如下表所示：

功能	0 轴执行相对点到点运动（PCI-1203 运动控制卡）
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S ²
减速度	10000PPU/S ²
速度曲线形式	T 型速度曲线
目标位置	10000PPU

VC 代码如下：

```
HAND    m_Axishand[32];                //0 轴的 handle
U32     Ret;                            // 函数返回值
--- 初始化过程 ---
// 设置参数
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度8000
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000);   // 加速度 10000
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000);   // 减速度 10000
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0);      // T 型曲线
// 执行点到点运动
```

```
Ret = Acm_AxMoveRel(m_Axishand[0], 10000);    // 相对点位运动
// 读取轴的状态和位置
F64    CmdPos;
F64    ActPos;
U16    State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);    // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos); // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);           // 获取 0 轴的状态
```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 PTP 例程。

9.2.3 连续运动

9.2.3.1 连续运动相关函数与属性

连续运动相关函数如下表 9.3 所示。下表中的函数可在应用程序中直接调用。

表 9.3：连续运动相关函数

连续运动相关函数	说明
Acm_AxMoveVel	轴按照规定速度进行没有终点的运动
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_AxStopDec	命令轴按设定的减速度停止运行
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	按指定减速度停止轴的运行
Acm_AxGetState	获取轴的当前状态。
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

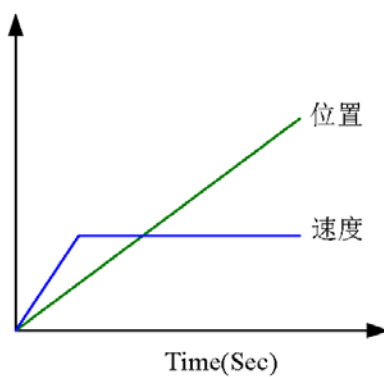
连续运动相关属性如下表 9.4 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.4：连续运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.2.3.2 重点说明

连续运动指令命令轴按规定速度和方向执行没有终点的运动。如图所示。



执行连续运动过程中，可单独设置各轴的速度参数。如初速度、加速度等。各轴将独立运行或停止（详见 9.2.5 章节）。

连续运动过程中，可改变运行轴的运行速度。执行连续运动时，轴的状态必须为 Ready。轴的状态可通过 Acm_AxGetState 函数获取。关于轴的运动状态见章节 8.2.2.1。

9.2.3.3 流程图

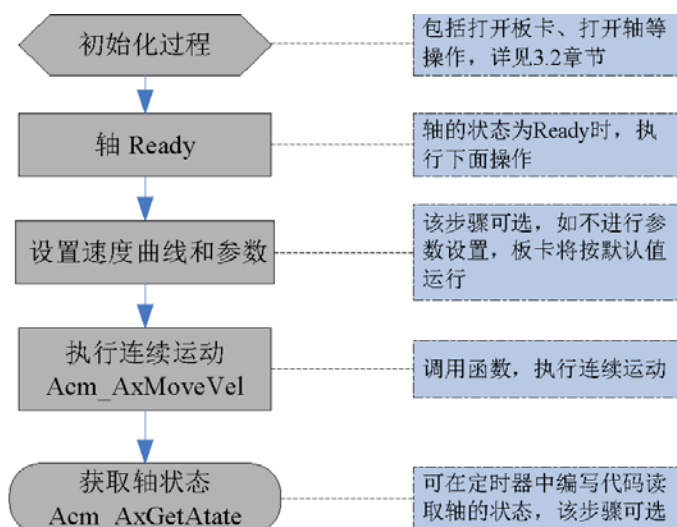


图 9.2：连续运动流程图

9.2.3.4 例程

该例程实现 0 轴执行正方向的连续运动，设定的速度等参数如下表所示。

功能	0 轴执行连续运动 (PCI-1203 运动控制卡)
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S ²
减速度	10000PPU/S ²
速度曲线形式	T 型速度曲线
运行方向	正方向

VC 代码如下：

```
HAND    m_Axishand[32];           // 轴的 handle
U32      Ret;                      // 函数返回值
--- 初始化过程 ---
// 设置参数
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度8000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000);   // 加速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000);   // 减速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0);      //T 型曲线
// 执行连续运动
Ret = Acm_AxMoveVel(m_Axishand[0], 0);                      // 正方向连续运动
// 读取轴的状态和位置
F64      CmdPos;
F64      ActPos;
U16      State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);               // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos);             // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);                       // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。
具体使用步骤可参考 CMove 例程。
```

9.2.4 变位运动

9.2.4.1 变位运动相关函数与属性

变位运动相关函数如下表 9.5 所示，下表中的函数可直接调用。

表 9.5：变位运动相关函数

变位运动函数	说明
Acm_AxChangePos	改变执行点到点运动轴的终点位置
Acm_AxMoveAbs	单轴的绝对点到点运动
Acm_AxMoveRel	单轴的相对点到点运动
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxStopDec	命令轴按设定减速度减速停止
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态。
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为 UInt32 位整数）
Acm_SetI32Property	设置属性值（属性值为 Int32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为 UInt32 位整数）
Acm_GetI32Property	获取属性值（属性值为 Int32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

变位运动相关属性如表 9.6 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。详细信息请参考附录。

表 9.6：变位运动相关属性

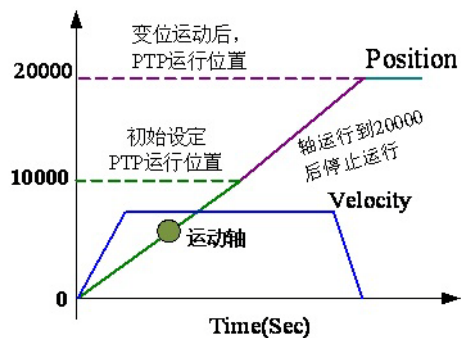
参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.2.4.2 重点说明

变位运动指轴在执行点到点运动过程中，可改变点到点运动的目标位置。

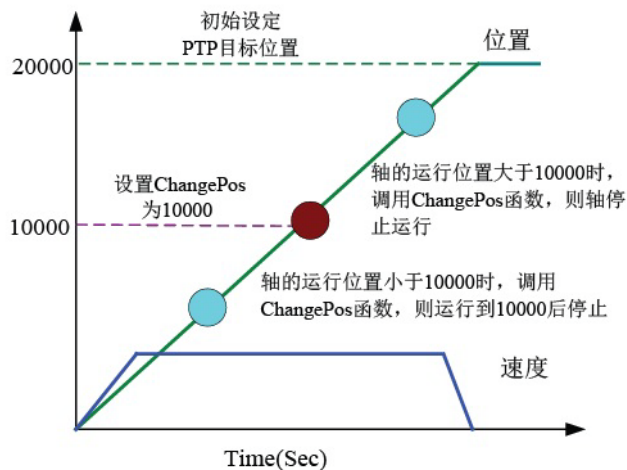
调用 ChangePos 函数改变运行轴的目标位置时，执行结果与轴的当前运动位置有关：

- 当 Change 的位置大于当前运动位置时，将运行到 Change 的位置



- 当 Change 的位置小于初始设定的目标位置，运行结果与轴的当前位置有关。如图所示：0 轴执行相对点到点运动；初始目标位置 20000，在运行过程通过调用 Acm_AxChangePos 将目标位置更改为 10000：

1. 当轴的运行位置小于 10000 时，改变目标位置，轴将运行到 10000 后停止运行。
2. 当轴的运行位置大于 10000 时，改变目标位置为 10000，轴将停止运行。



9.2.4.3 变位运动流程图

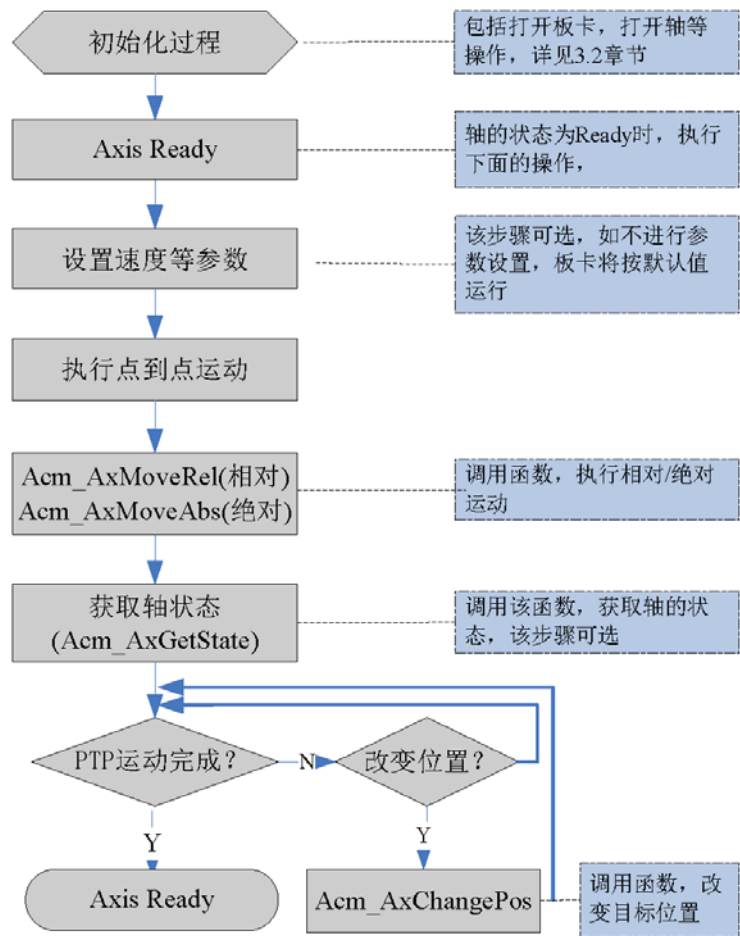


图 9.3：变位运动流程图

9.2.4.4 例程

例程实现 0 轴执行点到点运动过程中，改变轴的初始设定的目标位置，设置速度等参数如下表所示。

功能	0 轴执行点到点运动过程中，改变轴的目标位置
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S ²
减速度	10000PPU/S ²
速度曲线类型	T 型速度曲线
目标位置	10000PPU
改变目标位置	20000PPU

VC 代码如下：

```
U32    Ret;
DWORD  m_dwDevNum;
--- 初始化过程 ---
// 设置参数如下代码
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 初速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000); // 加速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000); // 减速度
```

```

Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0); //T 型曲线
// 执行点到点运动，运动过程中改变终点位置
Ret = Acm_AxMoveRel(m_Axishand[0], 10000);           // 相对点到点运动，目
                                                    标位置 10000
Ret = Acm_AxChangePos(m_Axishand[0], 20000);         // 改变终点位置为 20000
// 读取轴的状态和位置
F64    CmdPos;
F64    ActPos;
U16    State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);        // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos);      // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);               // 获取 0 轴的状态

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Change_P 例程。

9.2.5 变速运动

9.2.5.1 变速运动相关函数与属性

变速运动相关函数如下表 9.7 所示，表中的函数可直接调用。

表 9.7：变速运行相关函数

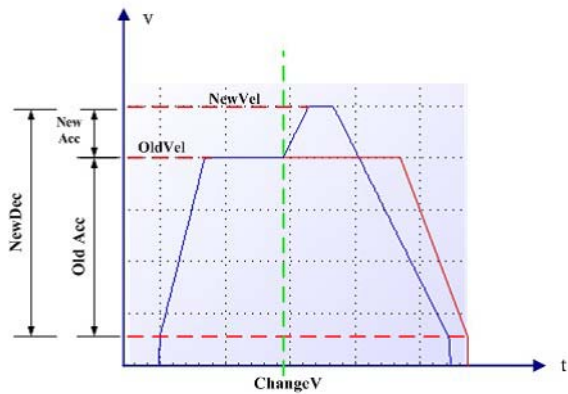
变速运动相关函数	说明
Acm_AxMoveVel	轴按照规定速度进行没有终点的运动
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxChangeVel	当轴在运动过程中，命令轴改变速度
Acm_AxChangeVelEx	改变运行轴的运行速度、加速度、减速度
Acm_AxChangeVelByRate	按照设定的比例改变当前轴的运行速度
Acm_AxChangeVelExByRate	按比率改变轴运行速度、加速度、减速度
Acm_AxGetCmdVelocity	获取指定轴的当前理论速度
Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置
Acm_AxStopDec	命令轴减速停止
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态。
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

变速运动相关属性如表 9.8 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.8：变速运行相关属性	
参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.2.5.2 重点说明

变速运动指当轴执行点到点运动或者连续运动时，改变当前轴的运行速度、加速度、减速度。改变的速度需大于初速度。变速运动如下图所示。



调用 Acm_AxChangeVel 函数改变指定轴的运行速度。若命令成功下达，在下次运动之前没有重新设定运行速度，则新的运行速度会作用到下次运动中。

调用 Acm_AxChangeVelEx 函数改变运行速度、加速度、减速度。若命令成功下达，在下次运动之前没有重新设定运动速度，则新的运行速度会作用到下次运动中。若新的加速度、减速度设置为 0，则使用上一次设定的加速度或减速度值。

调用 Acm_AxChangeVelExByRate 函数根据比率改变运行速度，同时改变加速度和减速度。新的速度，加速度和减速度仅对当前运动有效。若新的加速度减速度为 0，则使用上一次设定的加速度或减速度值。

调用 Acm_AxChangeVelByRate 函数根据比率改变运行速度。新的速度仅对当前运动有效。

9.2.5.3 变速运动流程图

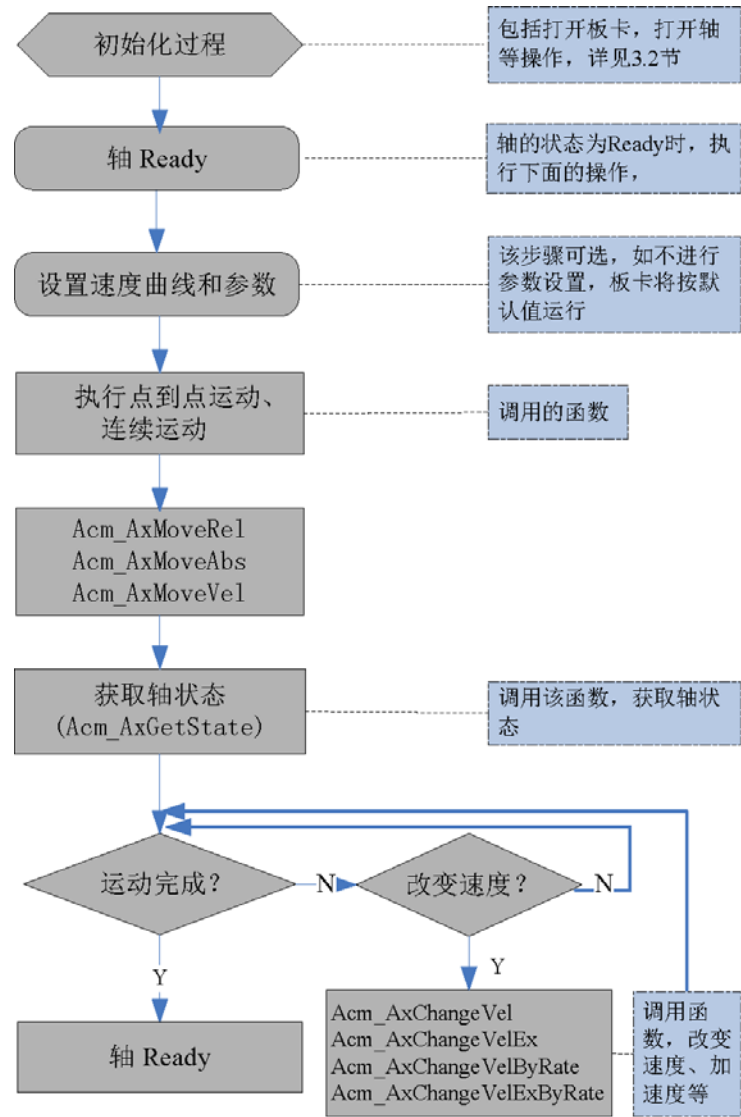


图 9.4：变速运动流程图

9.2.5.4 例程

例程实现 0 轴执行点到点运动过程中，改变运行轴的运行速度，设定的相关参数如下表所示。

功能	0 轴执行点到点运动过程中，改变运行速度
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S2
减速度	10000PPU/S2
速度曲线形式	T 型速度曲线
改变运行速度	6000 PPU/S
目标位置	10000PPU

VC 代码如下：

```
HAND    m_Axishand[32];
U32     Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
```



```
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 起始速度 2000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度8000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000); // 加速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000); // 减速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0); //T 型曲线
// 执行相对点到点运动，运动过程中改变运行速度
Ret = Acm_AxMoveRel(m_Axishand[0], 10000); // 相对点到点运动, 目标位置10000
Ret = Acm_AxChangeVel(m_Axishand[0], 6000); // 改变运行速度为 6000PPU/S
// 获取轴的位置和状态
F64 CmdPos;
F64 ActPos;
U16 State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos); // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos); // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State); // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。
具体参考 Change_V 例程。
```

9.2.6 同步起、同步停运动

9.2.6.1 同步起、同步停运动相关函数及属性

同步起、同步停运动相关函数如表 9.9 所示，下表中的函数可在应用程序中直接调用。

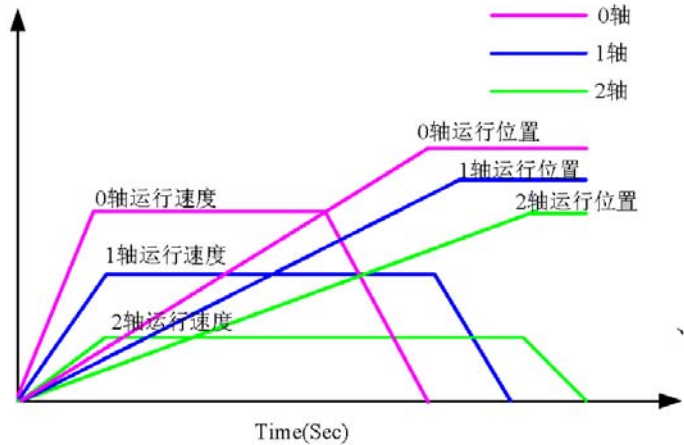
表 9.9：同步起、同步停运动相关函数	
同步起、同步停运动 相关函数	说明
Acm_AxSimStartSuspendAbs	设定轴为等待做相对点到点运动状态
Acm_AxSimStartSuspendRel	设定轴为等待做绝对点到点运动状态
Acm_AxSimStartSuspendVel	设定轴为等待做连续运动状态。
Acm_AxSimStart	启动所有等待启动触发的轴。
Acm_AxSimStop	停止所有触发的轴
Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_AxGetState	获取轴的运动状态
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

同步起、同步停运动相关属性如表 9.10 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.10：同步起、同步停运动相关属性	
参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度
CFG_AxSimStartSource	设置 / 获取当前轴的同步起停模式
特性	说明
FT_AxSimStartSourceMap	轴支持的同步起停模式。

9.2.6.2 重点说明

同步启、同步停运动指命令等待轴（等待同时开始运行的多个轴（一个及以上）），根据同步启动模式，同时做点到点运动或连续运动，并可同时停止所有轴的运行。同步启运动的所有轴，同时启动后将以各自设定的速度运行到目标位置。



运行过程中执行 Acm_AxSimStop 命令，将同时给所有轴停止的讯号，各轴会依据各自的设置的减速度停止运行。

同步启、同步停模式通过属性 CFG_AxSimStartSource 设置，不同的属性值，代表不同的同步启动模式。具体参考 CFG_AxSimStartSource 属性说明章节。

设置同步启动模式后，根据运动模式设置轴为等待状态：

1. 当运动模式为绝对点到点运动时，调用 Acm_AxSimStartSuspendAb 函数将轴设为等待状态。
2. 当运动模式为相对点位运动时，调用 Acm_AxSimStartSuspendRel 函数将轴设为等待状态。
3. 当运动模式为连续运动时，调用 Acm_AxSimStartSuspendRel 函数将轴设为等待状态。

最后调用 Acm_AxSimStart 函数发出同步启停信号，启动所有等待启动触发的轴，轴会根据同步启停模式启动触发。

9.2.6.3 同步起、同步停运动流程图

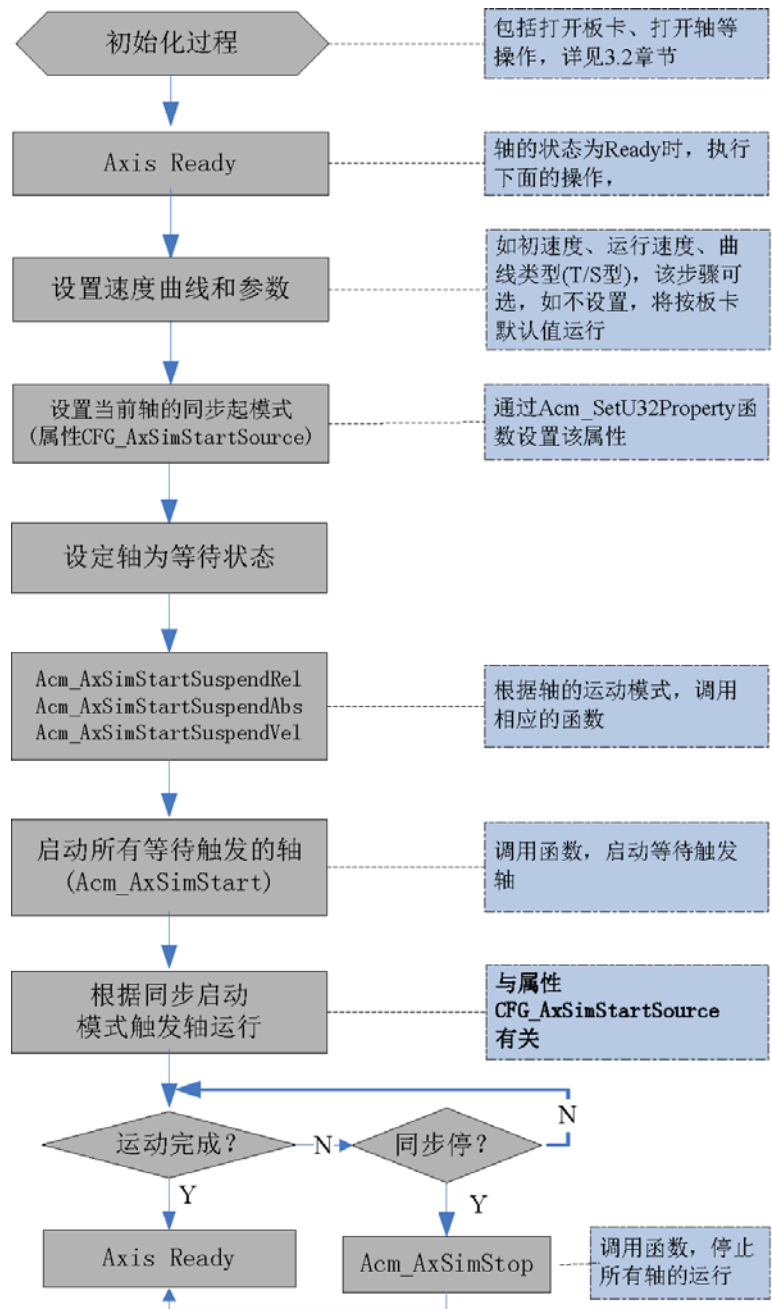


图 9.5：同步起、同步停运动流程图

9.2.6.4 例程

例程实现 1、2、3 轴同时起动做相对点到点运动，设定的速度等参数如下表所示。

功能	指定 1、2、3 轴为等待轴，同时做相对点到点运动
同步启动模式	SIM_STOP_AX0(表示 0 轴停止运行后，等待轴同时运行)
信号轴	0 轴
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S2
减速度	10000PPU/S2
速度曲线形式	T 型速度曲线

首先设置同步起动模式，即设置属性 CFG_AxSimStartSource 的值，每个等待轴都需单独设置同步起动模式。

VC 主要代码如下：

```
HAND    m_Axishand[32];
U32     Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
for (int i = 0; i < 4; i++)
{Ret = Acm_SetF64Property(m_Axishand[i], PAR_AxVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_Axishand[i], PAR_AxVelHigh, 8000); // 运行速度
                        8000
Ret = Acm_SetF64Property(m_Axishand[i], PAR_AxAcc, 10000); // 加速度 10000
Ret = Acm_SetF64Property(m_Axishand[i], PAR_AxDec, 10000); // 减速度 10000
Ret = Acm_SetF64Property(m_Axishand[i], PAR_AxJerk, 0); //T 型曲线 }
for (int i = 1; i < 4; i++)
{Ret=Acm_SetU32Property(m_Axishand[i], CFG_AxSimStartSource, SIM_STOP_AX0);/
/ 每个轴单独设置同步启停模式 }
for (int i = 1; i < 4; i++)
{Ret =Acm_AxSimStartSuspendRel(m_Axishand[i], 50000); // 设置轴为等待状态 }
Ret = Acm_AxSimStart(m_Axishand[0]); // 开始同步启停运动
```

完成上述设定，当 0 轴运动停止，1、2、3 轴将同时做点到点运动，

调用 Acm_AxSimStop 函数可停止轴的运行。如果设置同步启停模式为比较信号触发，则需要设置比较事件相关的属性和 API。

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤请参考 SimulateOpe 例程。

9.2.7 叠加运动

9.2.7.1 叠加运动相关函数

叠加运动相关函数如下表 9.11 所示，下表中的函数可在应用程序中直接调用。

表 9.11：叠加运动相关函数	
叠加运动相关函数	说明
Acm_AxMoveImpose	在当前运动上叠加新的运动。
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxStopDec	命令轴减速停止
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态。
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

表 9.11：叠加运动相关函数

Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置

叠加运动相关属性如表 9.12 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.12：叠加运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.2.7.2 重点说明

叠加运动指在当前运动上叠加新的运动，运动停止的终止位置将为原始位置加、减新设定的位置。

例如轴执行点到点运动，目标位置为 10000，调用叠加运动函数 Acm_AxMoveImpose 设定叠加相对位置为 3000，则轴运行到 13000 后停止运行；当设定的目标位置为 -3000 时，则轴运行到 7000 后停止运行。支持新叠加的运动的曲线为 T 型曲线。

叠加运动功能函数 Acm_AxMoveImpose 可设置新叠加段的位置和速度。

整个速度曲线由该运动的 NewVel、原始运动的 PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec、PAR_AxJerk 决定。如图 9.6 所示。

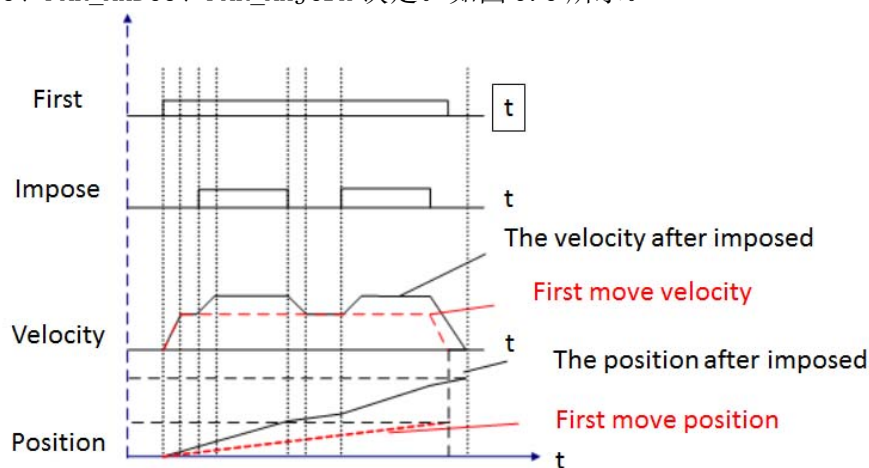


图 9.6：叠加运动

注！ 不能在叠加运动上叠加新的运动。



9.2.7.3 叠加运动流程图

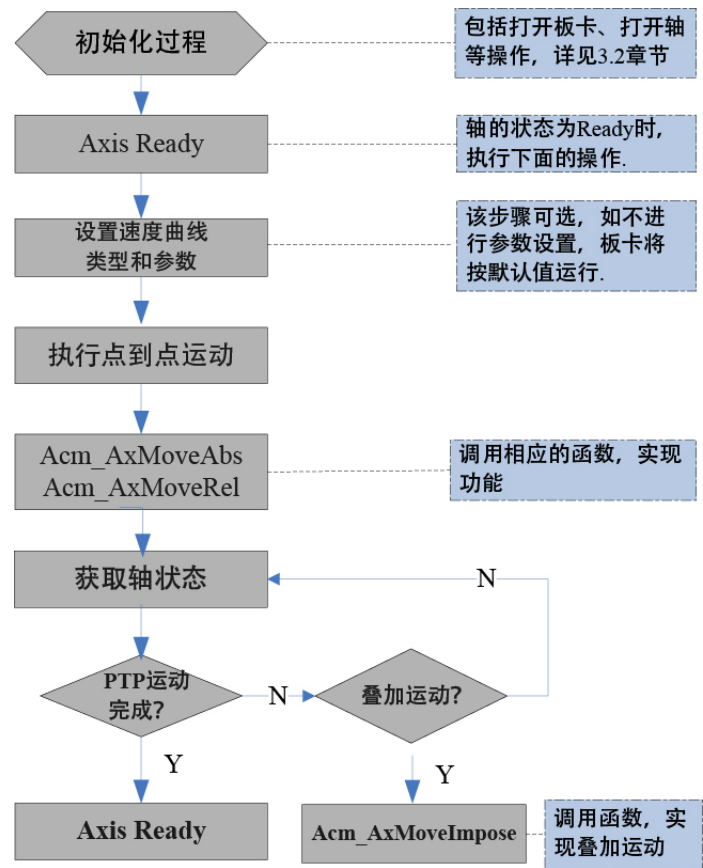


图 9.7： 叠加运动流程图

9.2.7.4 例程

例程实现 0 轴执行点到点运动过程中，叠加新的运动，设置的速度等参数值如下表所示。

功能	0 轴执行点到点运动过程中，叠加新的运动
目标位置	30000
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S ²
减速度	10000PPU/S ²
速度曲线形式	T 型速度曲线
叠加段运行速度	3000 PPU/S
叠加段距离	5000PPU

VC 代码如下：

```

HAND    m_Axishand[32];
U32     Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 起始速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000);    // 加速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000);   // 减速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0);      // T 型曲线
  
```



```
// 执行点位运动，在点位运动上叠加新的运动
Ret  = Acm_AxMoveRel(m_Axishand[0], 30000);           // 点到点运动，目标位置
                                                    100000

Ret  = Acm_AxMoveImpose(m_Axishand[0], 5000, 3000);    // 叠加新的运动
// 获取轴的位置和状态
F64   CmdPos;
F64   ActPos;
F64   NewVel;
U16   State;
Acm_AxGetCmdVelocity(m_Axishand[0], & NewVel)          // 获取 0 轴运动速度
Acm_AxGetCmdPosition(m_Axishand[0], &CmPos);           // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos);        // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);                  // 获取 0 轴的状态
运行后叠加段的速度为 11000，运行的距离为 35000
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。
详细使用步骤参考 MoveImpose 例程。
```

9.2.8 JOG 运动

9.2.8.1 JOG 运动相关函数与属性

JOG 运动运动相关函数如下表 9.13 所示。表中的函数可在应用程序中直接调用。

表 9.13：JOG 运动相关函数	
JOG、手轮运动相关函数	说明
Acm_AxSetExtDrive	启用或禁用外部驱动模式
Acm_AxJog	指定轴执行 Jog 运动
Acm_AxGetState	获取轴的当前状态。
Acm_AxResetError	复位轴的状态。
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）
Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置

JOG 运动相关属性如表 9.14 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性值。

表 9.14：JOG 运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxJogVelLow	设置 / 获取执行 Jog 运动时的初速度
CFG_AxJogVTime	设置 / 获取执行 Jog 运动时低速运转保持的时间
CFG_AxJogVelHigh	设置 / 获取执行 Jog 运动的运行速度
CFG_AxJogAcc	设置 / 获取执行 Jog 运动时的加速度
CFG_AxJogDec	设置 / 获取执行 Jog 运动时的减速度
CFG_AxJogJerk	设置 / 获取执行 Jog 运动时的速度曲线
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度
CFG_AxJogPAssign	设置 / 获取 DI 通道为 JOG+ 信号源
CFG_AxJogNAssign	设置 / 获取 DI 通道为 JOG- 信号源
特性	说明
FT_AxExtDriveMap	获取轴支持的外部驱动特性
FT_AxJogMap	获取轴支持的 Jog 特性
FT_AxExtMasterSrcMap	获取轴支持的外部驱动源

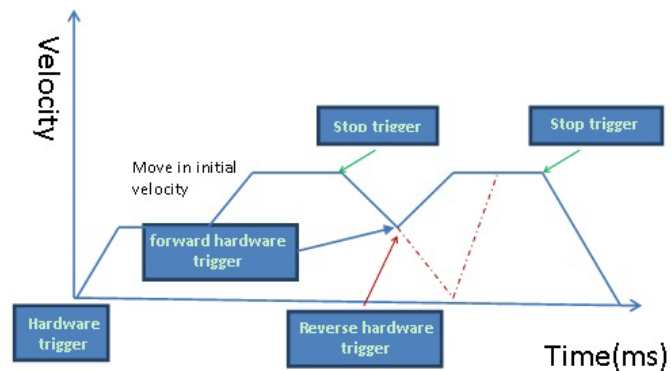
9.2.8.2 重点说明

Jog 运动、手轮运动模式下，可单独设置各轴的初速度、运行速度、加速度等速度参数，支持同一时间有两个以上的轴在 JOG 状态。

Jog 运动模式下，通过设置属性 CFG_AxJogVelLow、CFG_AxJogVelHigh、CFG_AxJogAcc、CFG_AxJogDec 的值，设置 Jog 运行时的初速度、运行速度、加速度、减速度。

执行 Jog 运动时，可通过设置属性 CFG_AxJogVTime 的值，设置低速运转保持的时间。即在该段时间内，轴将以初速度运行。

Jog 运动如下图所示。在停止 Jog 运行时，可继续触发 Jog 运动，当同向硬件触发时，轴将直接加速到运行速度，当反向硬件触发时，轴将减速到 0 后，再加速到运行速度。



通过设置 CFG_AxJogPAssign 或 CFG_AxJogNAssign，用户可以指定 ADAM-5000/ECAT 的 DI 通道为 JOG+ 和 JOG- 信号源。

9.2.8.3 JOG 运动流程图

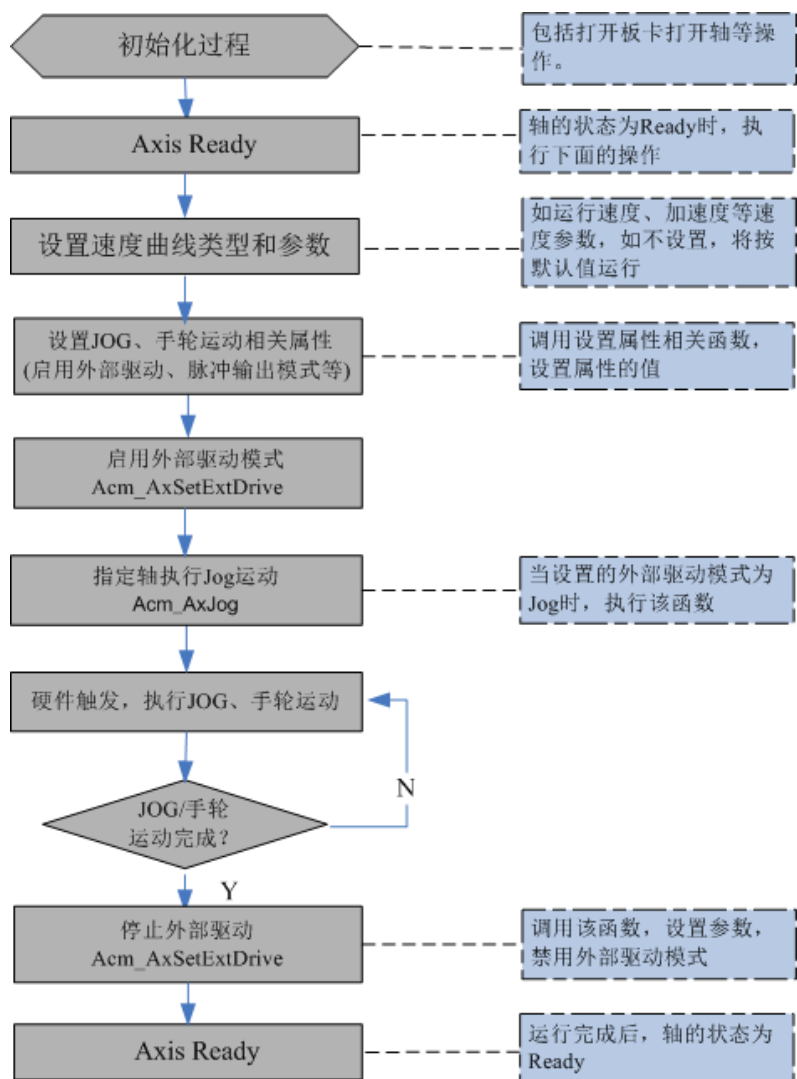


图 9.8: JOG 运动流程图

9.2.8.4 例程

例程实现 0 轴执行 JOG 运动，设置的速度等参数值如下表所示。

功能	0 轴执行 Jog 运动
理论脉冲数	1000
脉冲输出模式	2000PPU/S
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S ²
减速度	10000PPU/S ²
速度曲线形式	T 型速度曲线

VC 代码如下所示：

```
HAND    m_Axishand[32];
U32     Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 起始速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000);    // 加速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000);    // 减速度
Ret  = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0);       // T 型曲线
Ret  = Acm_AxSetExtDrive(m_Axishand[0], 1);                    // 启用 Jog 模式
// 获取轴的位置和状态
F64     CmdPos;
F64     ActPos;
U16     State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);                // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos);              // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);                        // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。
详细使用步骤请参考 JOG 例程
```

9.2.9 原点复归

9.2.9.1 原点复归相关函数与属性

原点复归相关函数如下表 9.15 所示，下表中的函数可直接调用。

表 9.15：原点复归相关函数

原点复归运动函数	说明
Acm_AxHome	命令轴开始回原点运动
Acm_AxMoveHome	使指定单轴执行回原点运动
Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置
Acm_AxStopDec	命令轴减速停止
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	按指定减速度停止轴的运行
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

原点复归运动相关属性如下表 9.16 所示，下表中属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.16：原点复归相关属性

参数	说明
PAR_AxHomeCrossDistance	设置原点跨越距离（单位：PPU）
PAR_AxHomeExSwitchMode	设置 Acm_AxHomeEx 的停止条件
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
PAR_AxHomeVelLow	设置 / 获取回 Home 时的初速度
PAR_AxHomeVelHigh	设置 / 获取回 Home 时的运行速度
PAR_AxHomeAcc	设置 / 获取回 Home 时的加速度
PAR_AxHomeDec	设置 / 获取回 Home 时的减速度
PAR_AxHomeJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxOrgLogic	设置 / 获取 ORG 信号的逻辑准位
CFG_AxEILogic	设置 / 获取硬件限位信号的逻辑准位
CFG_AxEzLogic	设置 / 获取 EZ 信号的有效逻辑电平
CFG_AxHomeResetEnable	启用 / 禁用逻辑计数器复位功能
CFG_AxOrgReact	设定回原点结束时的行为模式
CFG_AxMaxVel	配置运动轴的最大速度

表 9.16: 原点复归相关属性

CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度
特性	说明
FT_AxHomeMap	获取轴所支持的原点相关特性
FT_AxHomeModeMap	所支援的回原点方式

9.2.9.2 重点说明

原点复归指轴回 ORG、LMT+、LMT-、EZ 的行为模式。

原点复归模式下，ORG、LMT+、EZ 信号也称为原点信号 (Home 信号)，即电机回原点时控制它停止的信号。电机停止模式分为减速停止和立即停止，通过属性 CFG_AxOrgReact 设置。原点复归模式由 CiA402 定义，每种模式的运行方式将在后文中得以说明。

研华通用运动支持以两种函数来执行原点复归运动：Acm_AxHome 和 Acm_AxMoveHome。

调用 Acm_AxHome 执行回 Home 之前，通过 PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec、PAR_AxJerk 设置初速度，运行速度、加速度、减速度、速度曲线类型。

调用 Acm_AxMoveHome 执行回 Home 之前，通过 PAR_AxHomeVelLow、PAR_AxHomeVelHigh、PAR_AxHomeAcc、PAR_AxHomeDec、PAR_AxHomeJerk 设置初速度，运行速度、加速度、减速度、速度曲线类型。

原点复归模式分为两种类型，一种由研华定义，另一种是 CiA402 定义的典型原点复归运动。

第一类：由研华定义

要求轴开始典型原点复归运动。研华运动控制卡共提供 16 种原点复归模式。

注释：

16 种 Home 模式中的 MODE3_Ref~MODE16_LmtSearchReFind_Ref，某些阶段会使用初速度，所以通过 PAR_AxVelLow 设定的初速度须大于 0。

若 CFG_AxHomeResetEnable 被设置为 “True”，则命令位置和实际位置将在原点复归运动后被设置为 0。

说明：

下图中的 a、b、c、d 的含义如下：

- a: 当梯形 PTP 运动遇到 ORG/EL 信号时，速度将下降。
- b: 梯形 PTP 运动在结束前，由 HomeCrossDistance 决定运行距离。ORG/EL 信号有效。
- c: 梯形 PTP 以 VelLow 执行匀速运动，并将在遇到 ORG/EL 信号时迅速停止。
- d: 梯形 PTP 运动以 VelLow 运行，在运动结束前，由 HomeCrossDistance 决定运行距离。ORG/EL 信号有效。
- : 此实心原点表示运动终点。

注！



梯形 PTP 运动的特性：运动开始时，速度将以 Acc 从 VelLow 加速至 VelHigh（若距离足够长）；运动结束时，速度将以 Dec 从 VelHigh 减速至 VelLow。

16 种 Home 模式如下所示。

1. MODE1_Abs: Move (Dir) ->touch ORG->Stop.

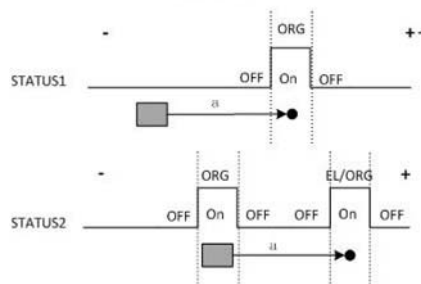
只依照原点设备（如传感器）返回原点。对象会一直运动，直至原点信号发生。

比如：

Dir: 正。

Org Logic (CFG_AxOrgLogic): 高准位。

EL (硬限位开关) 逻辑 (CFG_AxEILogic): 高准位。



- STATUS1: 如果对象超出 ORG 信号区域，当写入原点命令时，对象将一直运动直到 ORG 信号发生。
- STATUS2: 如果对象在 ORG 信号区域内或和 ORG 开关的方向相反，对象将一直运动直到 ORG 信号（如果有多于一个 ORG 开关或轴设备关闭）或 EL 信号（轴处于发生错误停止状态）发生。

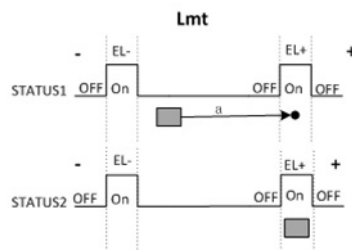
2. MODE2_Lmt: Move(Dir)->touch EL->Stop

只依照限位设备（如传感器）回原点。对象会一直运动，直至限位信号发生。

比如：

Dir: 正。

限位逻辑 (CFG_AxEILogic): 高准位。



- STATUS1: 如果对象超出 EL 信号区域，当写入原点命令时，对象将一直运动直到 EL 信号发生。
- STATUS2: 如果对象在 EL 信号区域内，将不作出响应。

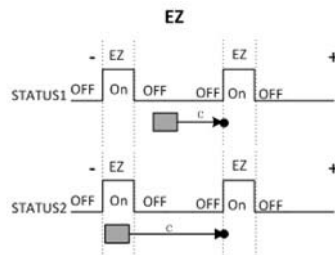
3. MODE3_Ref: Move (Dir) ->touch EZ->Stop

只按照 EZ 返回原点。对象会一直运动，直至 EZ 信号发生。

比如：

Dir: 正。

EZ 逻辑 (CFG_AxEzLogic): 高准位。



- STATUS1: 如果对象超出 EZ 信号区域，当写入原点命令时，对象将一直运动直到 EZ 信号发生。
- STATUS2: 如果对象在 EZ 信号区域内，对象将一直运动直到 EZ 信号发生。

4. **MODE4_Abs_Ref: ORG+EZ, Move(Dir) ->touch ORG ->Stop ->Move(Dir)->touch EZ ->Stop**

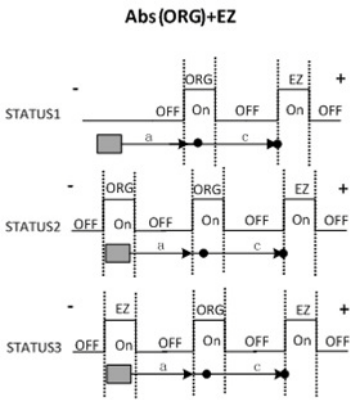
这是一种组合的原点模式。首先，对象将一直运动直到原始信号发生，然后将保持以 ORG 同一方向继续运动，直到 EZ 信号发生。

比如：

Dir: 正。

ORG 逻辑: 高准位。

EZ 逻辑: 高准位。



- STATUS1: 如果对象超出 EZ 信号和 ORG 信号区域，当写入原点命令时：首先，对象会一直运动直到 ORG 信号发生，然后继续运动，直到 EZ 信号发生。
- STATUS2: 如果对象在 ORG 信号区域内，写入原点命令时，对象开始运动。首先，ORG 信号消失，然后下一个 ORG 信号发生。最后，当 EZ 信号发生时运动停止。
- STATUS3: 如果对象在 EZ 信号区域内，写入原点命令时，对象开始运动。首先，EZ 信号消失，然后 ORG 信号发生。最后，当 EZ 信号发生时运动停止。

注！ 若遇到 EL 信号，则原点复归运动停止。



5. **MODE5_Abs_NegRef: ORG+EZ, Move (Dir) ->touch ORG ->Stop ->Move (-Dir) ->touch EZ ->Stop.**

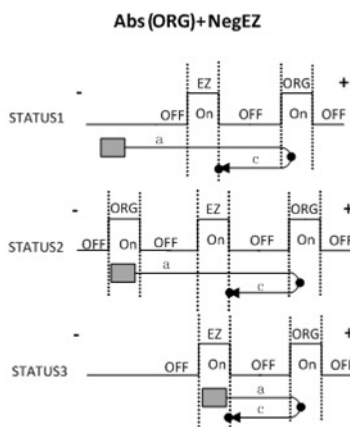
这是一种组合的原点模式。首先，对象将一直运动直到原始信号发生，然后将以与 ORG 相反方向继续运动，直到 EZ 信号发生。

比如：

Dir: 正。

ORG 逻辑: 高准位。

EZ 逻辑: 高准位。



- STATUS1: 如果对象超出 EZ 信号和 ORG 信号区域，当写入原点命令时：首先，对象会一直运动直到 ORG 信号发生，然后继续以相反方向运动，直到 EZ 信号发生。
- STATUS2: 如果对象在 ORG 信号区域内，写入原点命令时，对象开始运动。首先，ORG 信号消失，然后下一个 ORG 信号发生，同时倒转运动方向。最后，当 EZ 信号发生时运动停止。
- STATUS3: 如果对象在 EZ 信号区域内，写入原点命令时，对象开始运动。首先，EZ 信号消失，然后 ORG 信号发生，同时倒转运动方向。最后，当 EZ 信号发生时运动停止。

注！ 若遇到 EL 信号，则原点复归运动停止。



6. MODE6_Lmt_Ref: EL + NegEZ, Move (Dir) -> touch EL -> Stop -> Move (-Dir) -> touch EZ -> Stop.

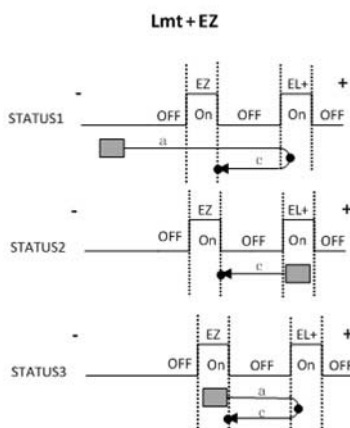
首先，对象将一直运动直到原始信号发生，然后将以与 ORG 相反方向继续运动，直到 EZ 信号发生。

比如：

Dir: 正。

EZ 逻辑：高准位。

限位逻辑：高准位。



- STATUS1: 如果对象超出 EZ 信号和 EL 信号区域，当写入原点命令时：首先，对象会一直运动直到 EL 信号发生，然后继续以相反方向运动，直到 EZ 信号发生。
- STATUS2: 如果对象在 EL 信号区域内，对象将一直以相反方向运动，直到 EZ 信号发生。

- STATUS3: 如果对象在 EZ 信号区域内, 写入原点命令时, 对象开始运动。首先, EZ 信号消失, 然后 EL 信号发生, 同时倒转运动方向。最后, 当 EZ 信号发生时运动停止。

7. MODE7_AbsSearch: Move (Dir) ->Search ORG ->Stop.

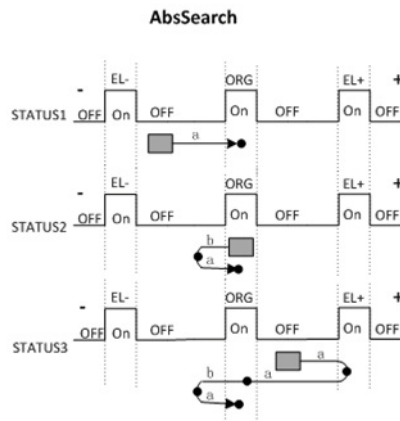
这是一种搜索 ORG 信号从无到有转换的模式。

比如:

Dir: 正。

ORG 逻辑: 高准位。

限位逻辑: 高准位。



- STATUS1: 如果没有 ORG 信号发生, 则 ORG 信号发生时对象将停止运动。
- STATUS2: 如果对象在 ORG 信号区域内, 对象以相反方向运动直到信号消失, 然后转换方向继续运动, 直到 ORG 信号发生。
- STATUS3: 如果没有 ORG 信号发生, 在运动时 EL 信号首先发生, 对象倒转方向并继续运动, 然后 ORG 信号将从有到无。然后, 再次倒转方向并运动, 直到 ORG 信号发生, 运动停止。

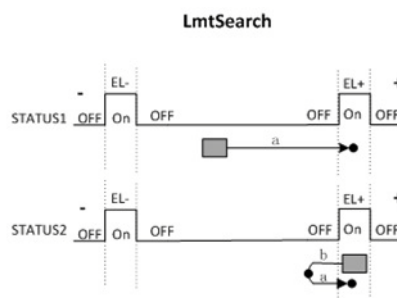
8. MODE8_LmtSearch: Move (Dir) ->Search EL ->Stop.

这是一种搜索限位信号从无到有转换的模式。

比如:

Dir: 正。

限位逻辑: 高准位。



- STATUS1: 如果限位信号在对象运动过程中首先发生, 则原点过程终止。
- STATUS2: 如果对象在限位信号区域内, 对象以相反方向运动直到信号消失, 然后转换方向继续运动, 直到限位信号发生。

9. MODE9_AbsSearch_Ref: Search ORG + EZ, Move (Dir) ->Search ORG ->Stop ->Move (Dir) ->touch EZ ->Stop.

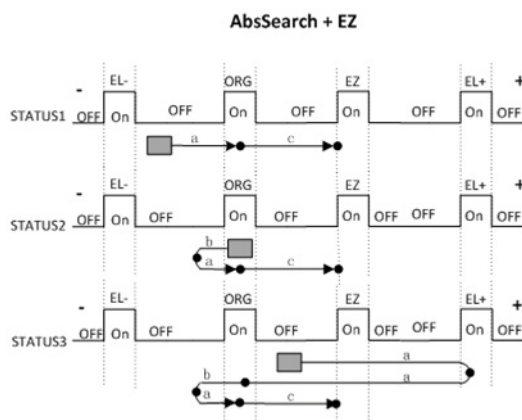
首先, 对象以 MODE7_AbsSearch 方式运动, 然后以相同方向运动直到 EZ 信号发生。

比如：

Dir：正。

限位逻辑：高准位。

ORG 逻辑：高准位。



- STATUS1: 如果对象超出 EZ 信号和 ORG 信号区域，写入原点命令时：首先，对象将一直运动，直到 ORG 信号发生，然后继续运动，直到 EZ 信号发生。
- STATUS2: 如果对象在 ORG 信号区域内，写入原点命令时：首先，对象倒转方向并运动，ORG 信号消失，然后再次倒转方向并继续运动，ORG 信号再次发生。最后，当 EZ 信号发生时运动停止。
- STATUS3: 如果没有 ORG 信号发生，EL 信号在 ORG 信号之前发生，当 EL 信号发生时对象倒转方向并继续运动，然后 ORG 信号从有到无。然后再次倒转方向并继续运动，ORG 信号将再次发生并消失。最后，当 EZ 信号发生时运动停止。

10. MODE10_AbsSearch_NegRef: Search ORG + NegEZ, Move (Dir) ->Search ORG ->Stop ->Move (-Dir) ->touch EZ ->Stop.

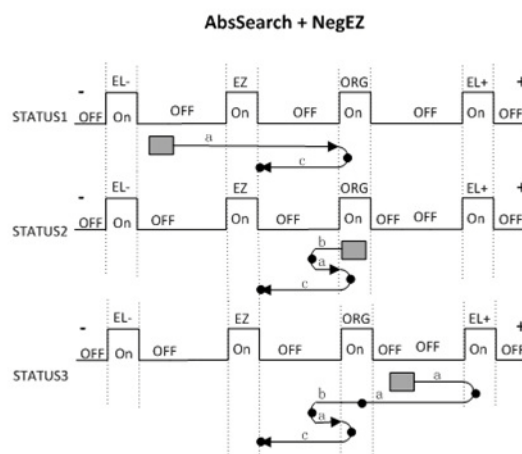
首先，对象以 MODE7_AbsSearch 方式运动，然后以相反方向运动直到 EZ 信号发生。

比如：

Dir：正。

限位逻辑：高准位。

ORG 逻辑：高准位。



- STATUS1: 如果对象超出 EZ 信号和 ORG 信号区域，当写入原点命令时：首先，对象会一直运动直到 ORG 信号发生，然后倒转方向并继续运动，直到 EZ 信号发生。
- STATUS2: 如果对象在 ORG 信号区域内，写入原点命令时：首先，对象倒转方向并运动，ORG 信号消失，然后再次倒转方向并继续运动，ORG 信号再次发生，继续倒转方向并运动。最后，当 EZ 信号发生时运动停止。

- STATUS3 如果没有 ORG 信号发生, EL 信号在 ORG 信号之前发生, 当 EL 信号发生时对象倒转方向并继续运动, 然后 ORG 信号从有到无。然后再次倒转方向并继续运动, ORG 信号将再次发生, 再次倒转方向。最后, 当 EZ 信号发生时运动停止。

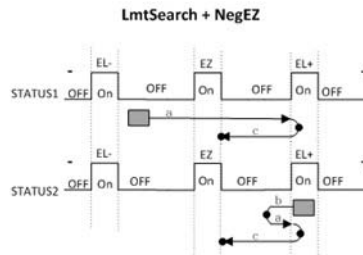
11. **MODE11_LmtSearch_Ref: Search EL +NegEZ, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->touch EZ ->Stop.**

首先, 对象以 MODE8_LmtSearch 方式运动, 然后以相反方向运动直到 EZ 信号发生。

比如:

Dir: 正。

限位逻辑: 高准位。



- STATUS1: 当对象不在限位信号区域内, 首先, 对象会一直运动直到 EL 信号发生, 然后倒转方向并继续运动, 直到 EZ 信号发生。
- STATUS2: 当对象不在限位信号区域内, 首先, 对象倒转方向并运动, EL 信号消失, 然后再次倒转方向并继续运动, EL 信号再次发生, 再次倒转方向并运动。最后, 当 EZ 信号发生时运动停止。

12. **MODE12_AbsSearchRefind: Search ORG +Refind ORG, Move (Dir) ->Search ORG ->Stop->Move (-Dir) ->Leave ORG(FL) ->Stop-> Move (-Dir)->Refind ORG(FL)->Stop.**

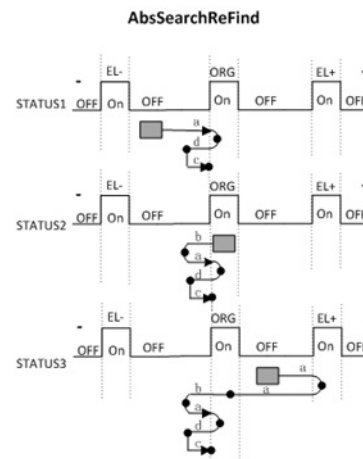
首先, 轴以 MODE7_AbsSearch 模式运动; 然后轴反向以低速 (VelLow) 等速运动直至 ORG 信号消失; 然后轴再次反向以低速 (VelLow) 等速运动直至 ORG 信号发生。

比如:

Dir: 正。

ORG 逻辑: 高准位。

限位逻辑: 高准位。



AbsSearch 过程有三种状况, 具体请参考 MODE7_AbsSearch 中的描述。

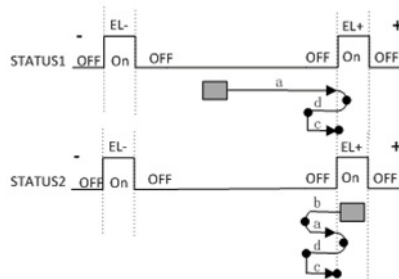
13. **MODE13_LmtSearchRefind: Search EL +Refind EL, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->Leave EL(FL) ->Stop-> Move (-Dir)->Refind EL(FL)->Stop.**

首先，轴以 MODE8_LmtSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 EL 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 EL 信号发生。

比如：

Dir：正。

限位逻辑：高准位。



14. **MODE14_AbsSearchRefind_Ref: Search ORG +Refind ORG+EZ, Move (Dir) ->Search ORG ->Stop->Move (-Dir) ->Leave ORG (FL) ->Stop-> Move (-Dir)->Refind ORG (FL)->Stop->Move (Dir) ->touch EZ ->Stop.**

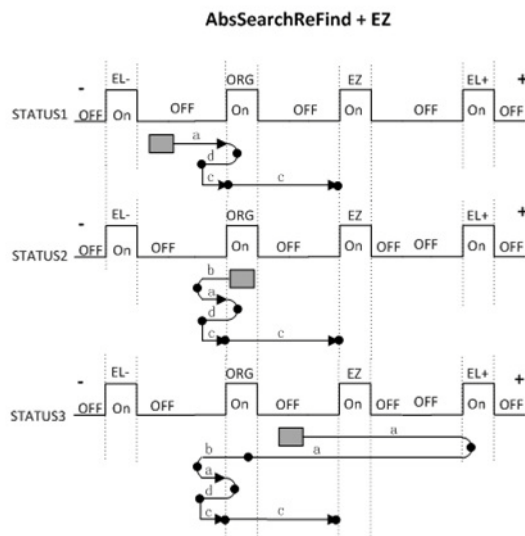
首先，轴以 MODE7_AbsSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 ORG 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 ORG 信号发生；最后轴沿相同方向运动至找到 Z 相。

比如：

Dir：正。

限位逻辑：高准位。

ORG 逻辑：高准位。



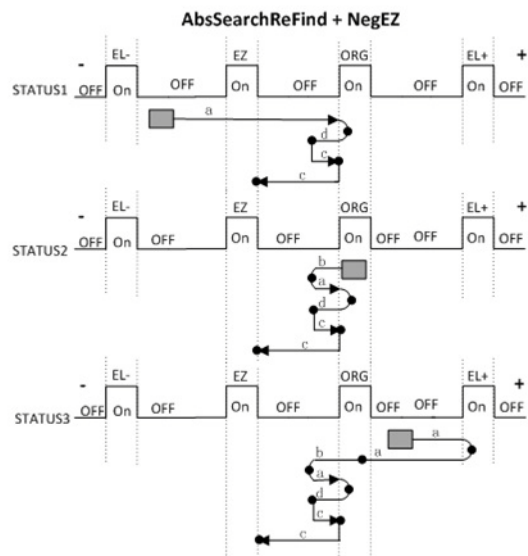
AbsSearch 过程有三种状况，具体请参考 MODE7_AbsSearch 中的描述。

15. **MODE15_AbsSearchRefind_NegRef: Search ORG +Refind ORG+NegEZ, Move (Dir) ->Search ORG ->Stop->Move (-Dir) ->Leave ORG (FL)->Stop-> Move (-Dir)->Refind ORG (FL)-> Stop-> Move (-Dir) ->touch EZ ->Stop.**

首先，轴以 MODE7_AbsSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 ORG 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 ORG 信号发生；最后轴再次反向运动直至 EZ 信号发生。

比如：

Dir: 正。
限位逻辑: 高准位。
ORG 逻辑: 高准位。



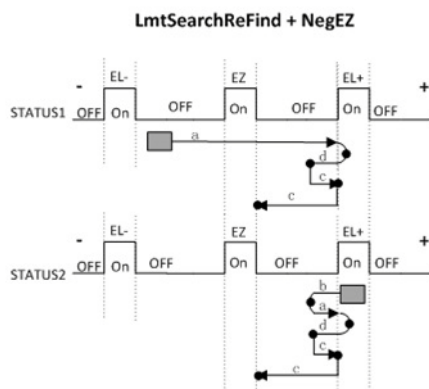
AbsSearch 过程有三种状况，具体请参考 MODE7_AbsSearch 中的描述。

16. **MODE16_LmtSearchReFind_Ref: Search EL +Refind EL+EZ, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->Leave EL(FL) ->Stop-> Move (-Dir)->Refind EL(FL)->Stop->Move (-Dir) ->touch EZ ->Stop.**

首先，轴以 MODE8_LmtSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 EL 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 EL 信号发生，最后轴再次反向运动直至 EZ 信号发生。

比如：

Dir: 正。
限位逻辑: 高准位。



LmtSearch 过程有三种状况，具体请参考 MODE8_LmtSearch 中的描述。

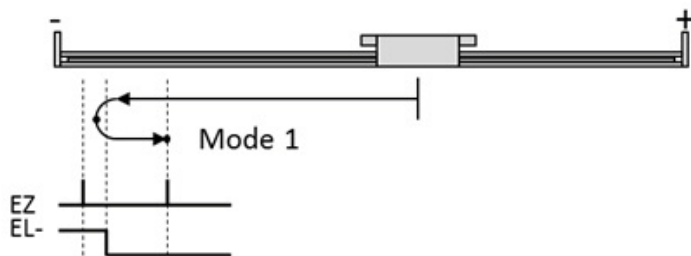
第二类：由 CiA402 定义

CiA402 定义的 35 种典型的原点复归运动可通过 Acm_AxHome 和 Acm_AxMoveHome 函数进行设置。

以下原点复归模式的说明和图片均来自 CiA402，且针对不同马达驱动会有细微区别。关于马达驱动原点复归模式的正确含义，请参考马达驱动厂商提供的使用手册。

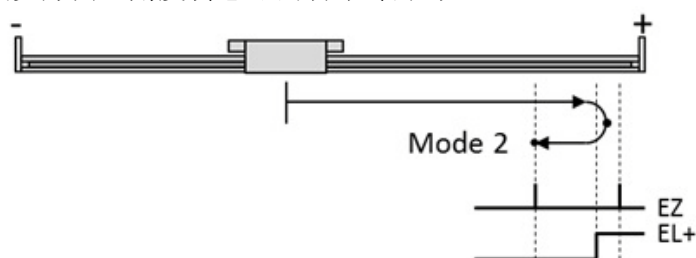
方法 1：采用零脉冲评估的负限位开关

在这个方法里驱动器首先在负方向上较快运动，直至它到达负限位开关位置。这在示意图中通过上升沿来表示。之后驱动器慢慢返回寻找限位开关的精确位置。零点位置指的是限位开关正方向上角度传感器的首个零脉冲。



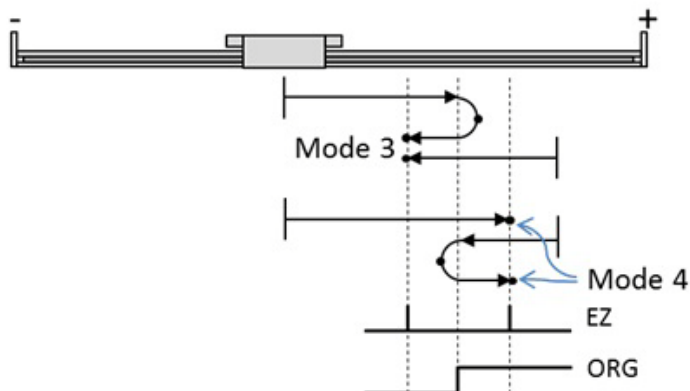
方法 2：采用零脉冲评估的正限位开关

在这个方法里驱动器首先在正方向上较快运动，直至它到达正限位开关位置。这在示意图中通过上升沿来表示。之后驱动器慢慢返回寻找限位开关的精确位置。零点位置指的是限位开关负方向上角度传感器的首个零脉冲。



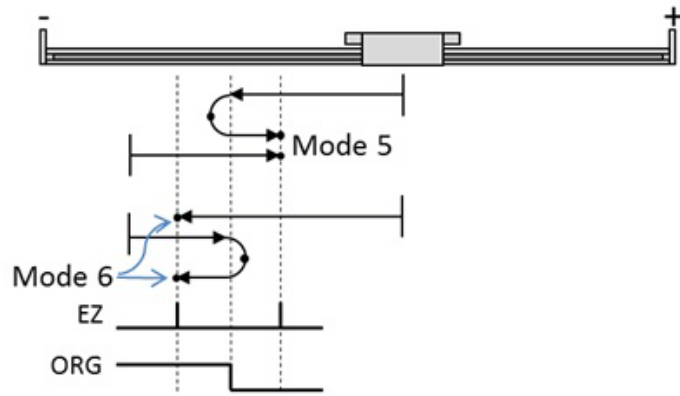
方法 3 和 4：采用零脉冲评估的正向参考开关

采用方法 3 和 4 时，运动的初始方向决定于参考开关的状态。参考位置指的是参考开关改变状态的点向左或向右的零脉冲。若初始位置已设定，在执行原点回归运动时，运动方向必须改变，而发生方向改变的点可以是参考开关改变状态之后的任何位置。



方法 5 和 6：采用零脉冲评估的负向参考开关

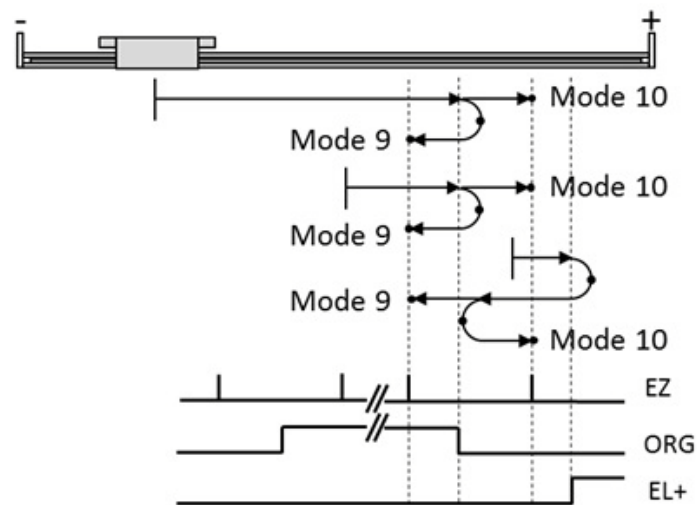
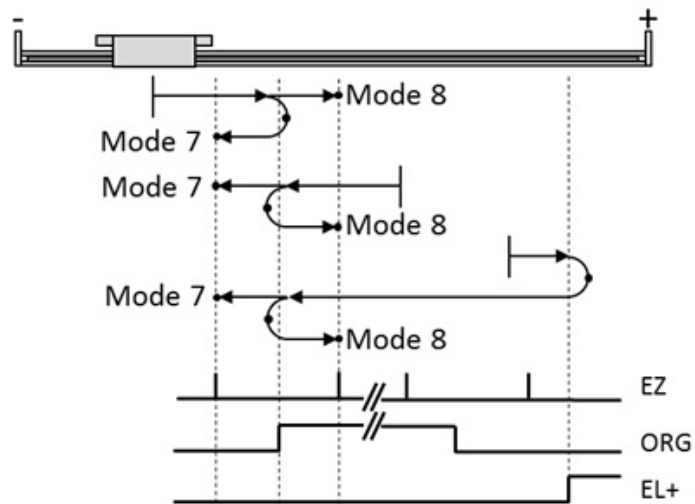
采用方法 5 和 6 时，运动的初始方向决定于参考开关的状态。参考位置指的是参考开关改变状态的点向左或向右的零脉冲。若初始位置已设定，在执行原点回归运动时，运动方向必须改变，而发生方向改变的点可以是参考开关改变状态之后的任何位置。

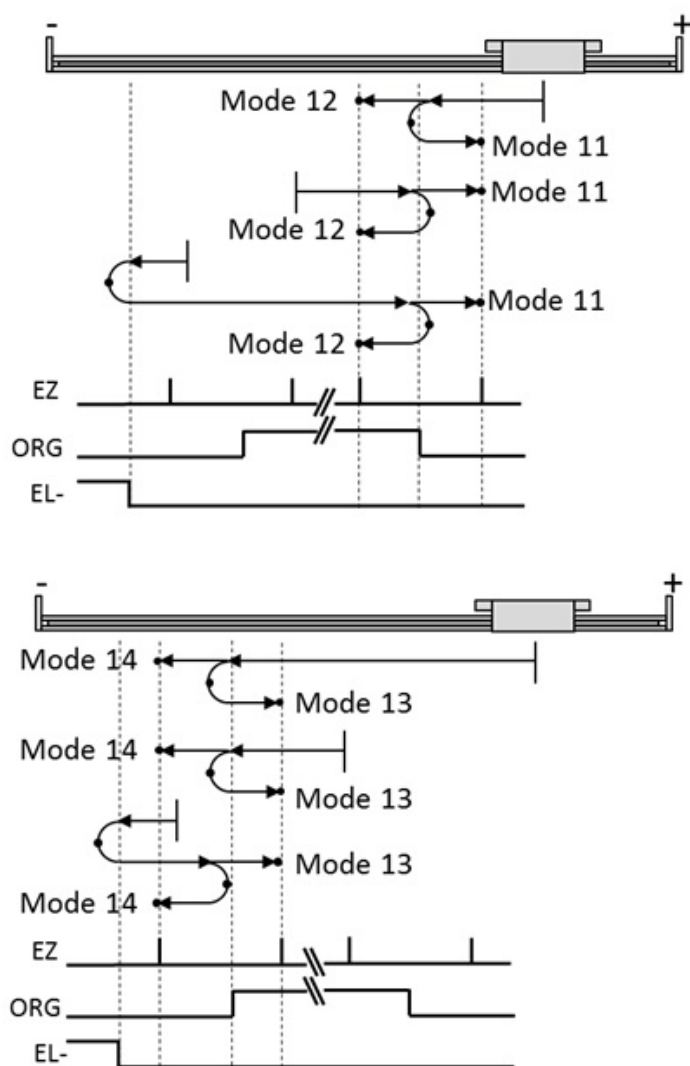


方法 7 至 14: 参考开关和零脉冲评估

这些方法使用仅在部分路段上激活的参考开关，当轴经过开关时，开关将采取 “临时” 响应。

使用方法 7 至 10 时，首先在正向运动，使用方法 11 至 14 时，首先在负向运动，除非参考开关在运动启动时就已激活。这时，运动的初始方向取决于寻找中的沿。如下图所示，参考位置是参考开关的上升沿或下降沿的零脉冲。若运动初始方向远离参考开关，则驱动必须在到达相应限位开关时改变方向。



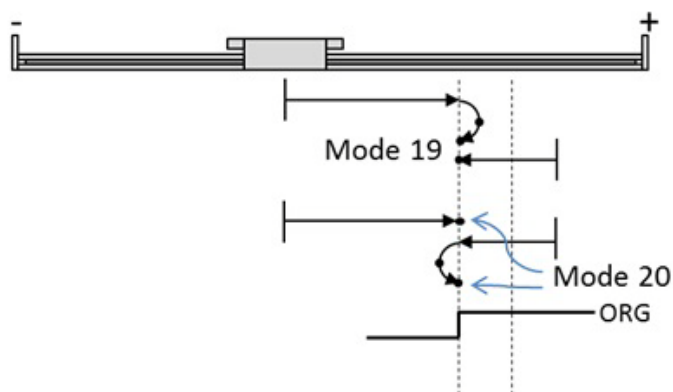


方法 15 和 16：预留

这两种方法为将来可能的扩展功能预留。

方法 17 至 30：与零脉冲无关的原点复归

这些方法与方法 1 至 14 相似，不同的是，原点复归与零脉冲无关，与相对零点位置或限位开关转换有关。例如，方法 19 和 20 与方法 3 和 4 相似（如下图所示）。

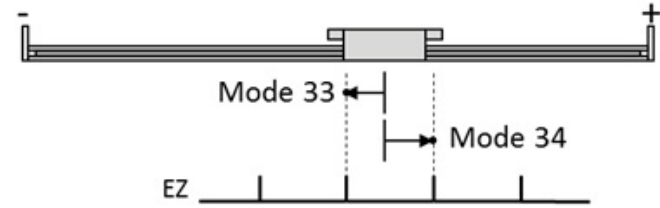


方法 31 和 32：预留

这两种方法为将来可能的扩展功能预留。

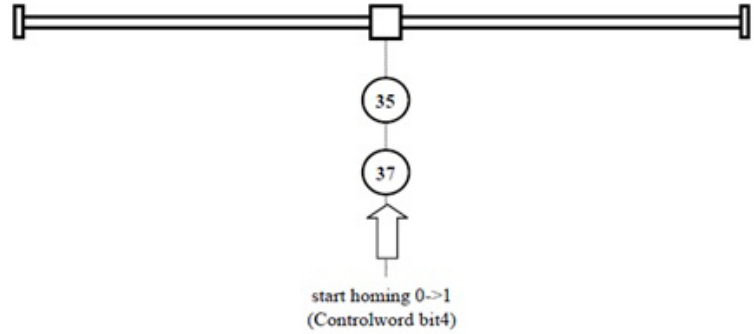
方法 33 至 34：直至零脉冲产生的原点复归

采用方法 33 或 34 时，原点复归的方向分别是负向或正向。零点位置指的是寻找方向上角度传感器的首个零脉冲。



方法 35 和 37：至当前位置的原点复归

方法35由CiA402定义，方法37 由Panasonic EtherCAT Motor 定义且只能在Panasonic EtherCAT Motor 中使用。在方法 35 中，零点位置指的是当前位置。尽管方法 35 和 37 相同，根据 ETG 标准在新设计中使用方法 37。



方法 37 由 Panasonic EtherCAT Motor 定义

9.2.9.3 原点复归流程图

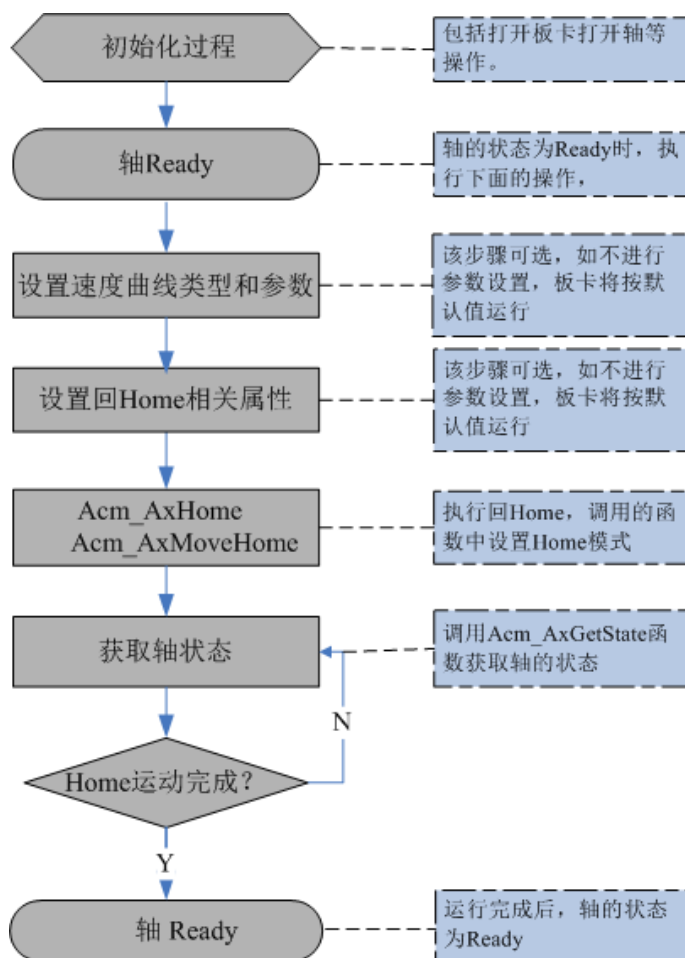


图 9.9： 原点复归流程图

9.2.9.4 例程

该例程实现 0 轴执行原点回归运动，设置的速度等参数值如下表所示。

功能	0 轴执行原点回归运动
回 Home 模式	MODE8_LmtSearch(搜索限位信号从无到有转换的模式)
ORG 信号电平	低电平有效
LMT 信号电平	低电平有效
EZ 信号电平	低电平有效
跨越距离	100
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S2
减速度	10000PPU/S2
速度曲线形式	T 型速度曲线

VC 代码如下：

```
HAND    m_Axishand[32];
U32     Ret;
--- 初始化过程 ---
// 设置速度参数如下代码
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 起始速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000);    // 加速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000);    // 减速度
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0);       //T 型曲线
Ret = Acm_SetU32Property(m_Axishand[0], CFG_AxEILogic, 0);    // LMT 低电平有效

Ret = Acm_SetU32Property(m_Axishand[0], CFG_AxOrgLogic, 0); //ORG 低电平有效
Ret = Acm_SetU32Property(m_Axishand[0], CFG_AxEzLogic, 0);  //EZ 低电平有效
Ret = Acm_SetU32Property(m_Axishand[0], PAR_AxHomeExSwitchMode, 0);
// 传感器开启
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxHomeCrossDistance, 100);
// 跨越距离
Ret = Acm_AxHome(m_Axishand[0], 7, 0); //HomeMode 为 MODE8_LmtSearch 的正向回
Home
// 获取轴的位置和状态
F64     CmdPos;
F64     ActPos;
U16     State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);    // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axishand[0], &ActPos); // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);           // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。
详细使用步骤请参考 Home 例程
```

9.2.10 轴的事件

9.2.10.1 轴的事件相关函数与属性

轴的事件相关函数如下表 9.17 所示，下表中的函数可直接调用。

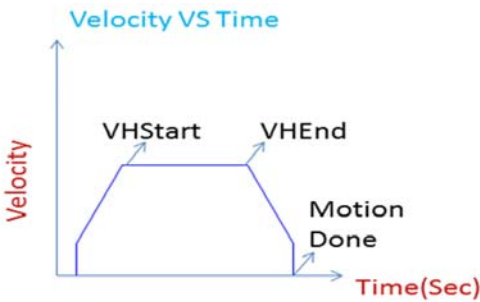
表 9.17：轴的事件相关函数	
轴的事件函数	说明
Acm_EnableMotionEvent	启用轴和群组的事件状态。
Acm_CheckMotionEvent	检测轴和群组的事件状态
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxMoveVel	轴按照规定速度进行没有终点的运动
Acm_AxSimStartSuspendAbs	设定轴为等待做相对点到点运动状态
Acm_AxSimStartSuspendRel	设定轴为等待做绝对点到点运动状态
Acm_AxSimStartSuspendVel	设定轴为等待做连续运动状态。
Acm_AxSimStart	启动所有等待启动触发的轴。
Acm_AxSimStop	停止所有触发的轴
Acm_AxSetExtDrive	启用或禁用外部驱动模式
Acm_AxJog	指定轴执行 Jog 运动
Acm_AxHome	命令轴开始回原点运动
Acm_AxSetCmdPosition	设置指定轴的理论位置
Acm_AxSetActualPosition	设置指定轴的实际位置
Acm_AxGetCmdPosition	获取指定轴的当前理论位置
Acm_AxGetActualPosition	获取指定轴的当前实际位置
Acm_AxStopDec	命令轴减速停止
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	按指定减速度停止轴的运行
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

轴的事件相关属性如表 9.18 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.18：轴的事件相关属性	
参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度
特性	说明
FT_AxMaxVel	获取轴支持的最大速度
FT_AxMaxAcc	获取轴支持的最大加速度
FT_AxMaxDec	获取轴支持的最大减速度
FT_AxMaxJerk	获取轴支持的最大加加速度

9.2.10.2重点说明

轴的事件包括 Motion Done、VHStart、VHEnd、Compare、Latch、LatchBuffer。本节主要介绍 Motion Done、VHStart、VHEnd，其他事件请参考相关章节。Motion Done、VHStart、VHEnd 如图所示。



Motion Done 事件指单轴运动结束时，硬件触发产生的中断事件。
VHStart 事件指轴加速到运行速度时，硬件触发产生的中断事件
VHEnd 事件指轴运行速度运行结束时，硬件触发产生的中断事件。
VHStart、VHEnd 分别指高速度运行开始、高速度运行结束，高速度指运行速度。

注！ *VHStart* 和 *VHEnd* 分别表示运行速度开始和运行速度结束。



若要获取轴或群组的事件状态，用户需调用 Acm_EnableMotionEvent 来启用这些事件。启用 / 禁用每个轴的事件，可通过设置参数 AxEvtStatusArray[N] 实现。该参数为 U32 位整数数组阵列，N 代表设备的轴数。例如板卡轴数为 4，则 AxEvtStatusArray[0] 代表 0 轴的所有事件状态，每一位分别表示不同的事件。

Bit n = 1 启用事件

Bit n = 0 禁用事件

如下所示。

AxEvtStatus[0] \ Axis	Axis0	Axis1	Axis2	Axis3
Bit0	Evt_Ax_Motion_Done	...	...	...
Bit1	Evt_Ax_Compared	...	...	...
Bit2	Evt_Ax_Latched	...	...	...
Bit3	Evt_Ax_Error	...	...	...
Bit4	Evt_AxVHSTART	...	...	...
Bit5	Evt_AxVHEnd	...	...	...
Bit6	Evt_Ax_LatchBuffer	...	...	...
Bit7-31	Reserved	...	...	...

启用轴的事件后，创建一个新的线程，调用 Acm_CheckMotionEvent 函数，该函数的参数 AxEvtStatusArray[N] 将返回所有轴的中断事件状态。该参数为 U32 位整数数组阵列，N 代表设备的轴数。例如板卡轴数为 4，则 N=4，AxEvtStatusArray[0] 代表 0 轴的所有事件状态，每一位分别表示不同的事件。

Bit n = 1 事件发生

Bit n =0 事件未发生

如下表所示。

AxEvtStatus[0] \ Axis	Axis0	Axis1	Axis2	Axis3
Bit0	Evt_Ax_Motion_Done	...	...	...
Bit1	Evt_Ax_Compared	...	...	...
Bit2	Evt_Ax_Latched	...	...	...
Bit3	Evt_Ax_Error	...	...	...
Bit4	Evt_AxVHSTART	...	...	...
Bit5	Evt_AxVHEnd	...	...	...
Bit6	Evt_Ax_LatchBuffer	...	...	...
Bit7-31	Reserved	...	...	...

9. 2. 10. 3轴的事件流程图

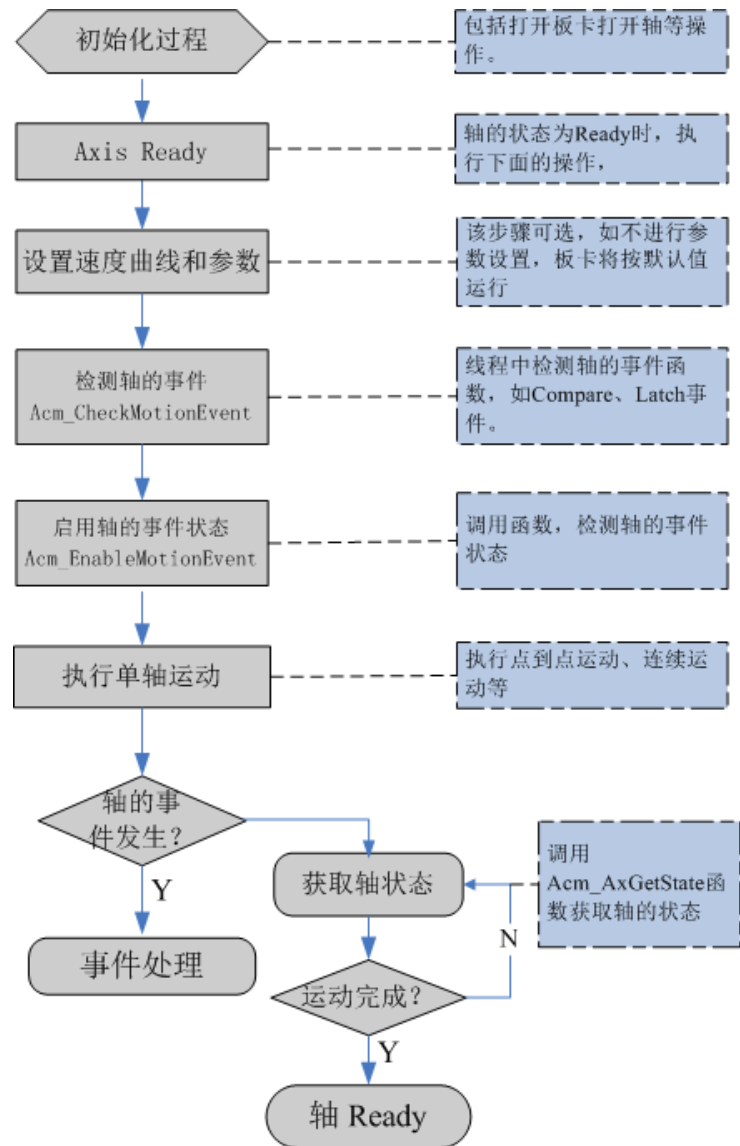


图 9. 10: 轴的事件流程图

9. 2. 10. 4例程

例程实现检测 0 轴的事件，设置的速度等参数值如下表所示。

功能	检测 0 轴的 Motion Done、VHStart、VHEnd 事件
初速度	2000PPU/S
运行速度	8000PPU/S
加速度	10000PPU/S2
减速度	10000PPU/S2
速度曲线形式	T 型速度曲线
目标位置	10000PPU

VC 代码如下：

```
HAND    m_Axishand[32];
U32     Ret;
U32     AxEnableEvtArray[4];
U32     GpEnableEvt[3];
U32     m_ulAxisCount;
```

```

BOOL    m_bInit ;
--- 初始化过程 ---
// 设置速度参数如下代码
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxAcc, 10000); // 加速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxDec, 10000); // 减速度 10000
Ret = Acm_SetF64Property(m_Axishand[0], PAR_AxJerk, 0); // T 型曲线
pThreadObject = AfxBeginThread( (AFX_THREADPROC)CDlg::CheckEvtThread,
this, THREAD_PRIORITY_TIME_CRITICAL, 0, 0, NULL); // 创建线程对象
m_ulAxisCount =4; // 板卡轴数
AxEnableEvtArray[0] |= EVT_AX_MOTION_DONE; // 启用 Motion Done
AxEnableEvtArray[0] |= EVT_AX_VH_START; // 启用 VHStart
AxEnableEvtArray[0] |= EVT_AX_VH_END; // 启用 VHEND
Ret = Acm_EnableMotionEvent(m_Devhand, AxEnableEvtArray, GpEnableEvt,
m_ulAxisCount ,3); // 启用事件
m_bInit = true;
UINT CDlg::CheckEvtThread(LPVOID ThreadArg) // 检测事件的线程函数
{
    U32 Ret;
    U32 AxEvtStatusArray[4];
    U32 GpEvtStatusArray[3];
    CDlg *pThreadInfo;
    pThreadInfo = (CDlg*)ThreadArg;
    while (pThreadInfo->m_bInit)
    {
        Ret = Acm_CheckMotionEvent(pThreadInfo->m_Devhand, AxEvtStatusArray,
        GpEvtStatusArray, pThreadInfo->m_ulAxisCount, 3, 10000); // 检测事件状态 }}
    // 获取轴的位置和状态
    F64 CmdPos;
    F64 ActPos;
    U16 State;
    Acm_AxGetCmdPosition(m_Axishand[0], &CmPos); // 获取 0 轴的理论位置
    Acm_AxGetActualPosition(m_Axishand[0], &ActPos); // 获取 0 轴的实际位置
    Acm_AxGetState(m_Axishand[0], &State); // 获取 0 轴的状态
}

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。具体使用步骤可参考 EVENT 例程。

9.2.11PT 运动

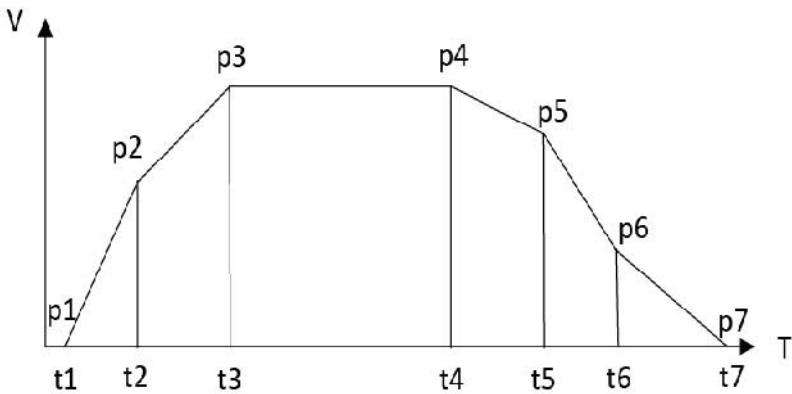
9.2.11.1PT 运动相关函数与属性

PT 运动相关函数如下表 9.19 所示，下表中的函数可直接调用。

表 9.19：PT 运动函数	
PT 运动函数	说明
Acm_AxAddPTData	将一笔 PT 资料添加至 PT 缓存
Acm_AxStartPT	开始 PT 运动
Acm_AxStartAllPT	指定多轴开始 PT 运动
Acm_AxCheckPTBuffer	获取 PT 缓存中剩余空间的个数
Acm_AxResetPTData	清空 PT 缓存
Acm_AxGetState	获取轴的当前状态

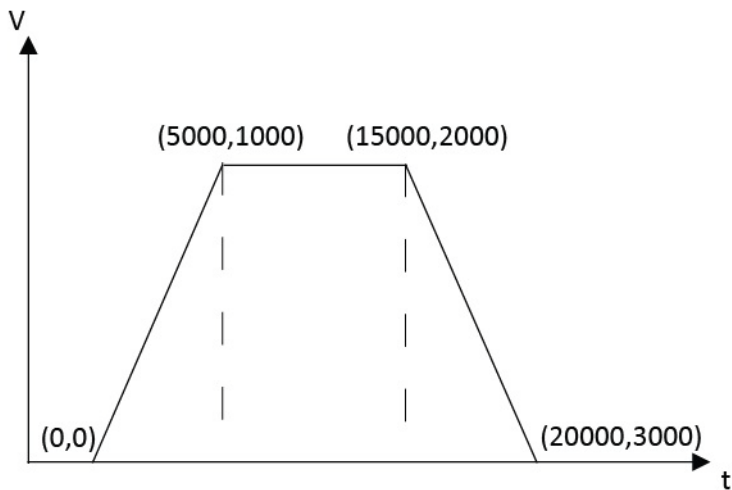
9.2.11.2重点说明

当轴的状态为 Ready 时，将能够执行 PT 移动。关于轴状态请详见 6.2 节。
PT 运动使用位置和时间数据来形成速度曲线。因此，可以根据输入的 PT 数据点来操纵加速和减速运动。



如上图所示，完整的速度曲线可以被划分为几个运动段，并且每个段基于第一数据点 (p1, t1) 来计算其位移。每个数据点是相对于起点的绝对值。数据点示例如下所示：

	P(position, PPU)	T(time, ms)
数据 1	0	0
数据 2	5000	1000
数据 3	15000	2000
数据 4	20000	3000



用户可以在 PT 运动操作时更新数据。如果用户需要在 PT 运动开始后将新数据更新到 PT 存，则需要重置 PT 缓存以清除先前的信息。

9.2.11.3 PT 运动流程图

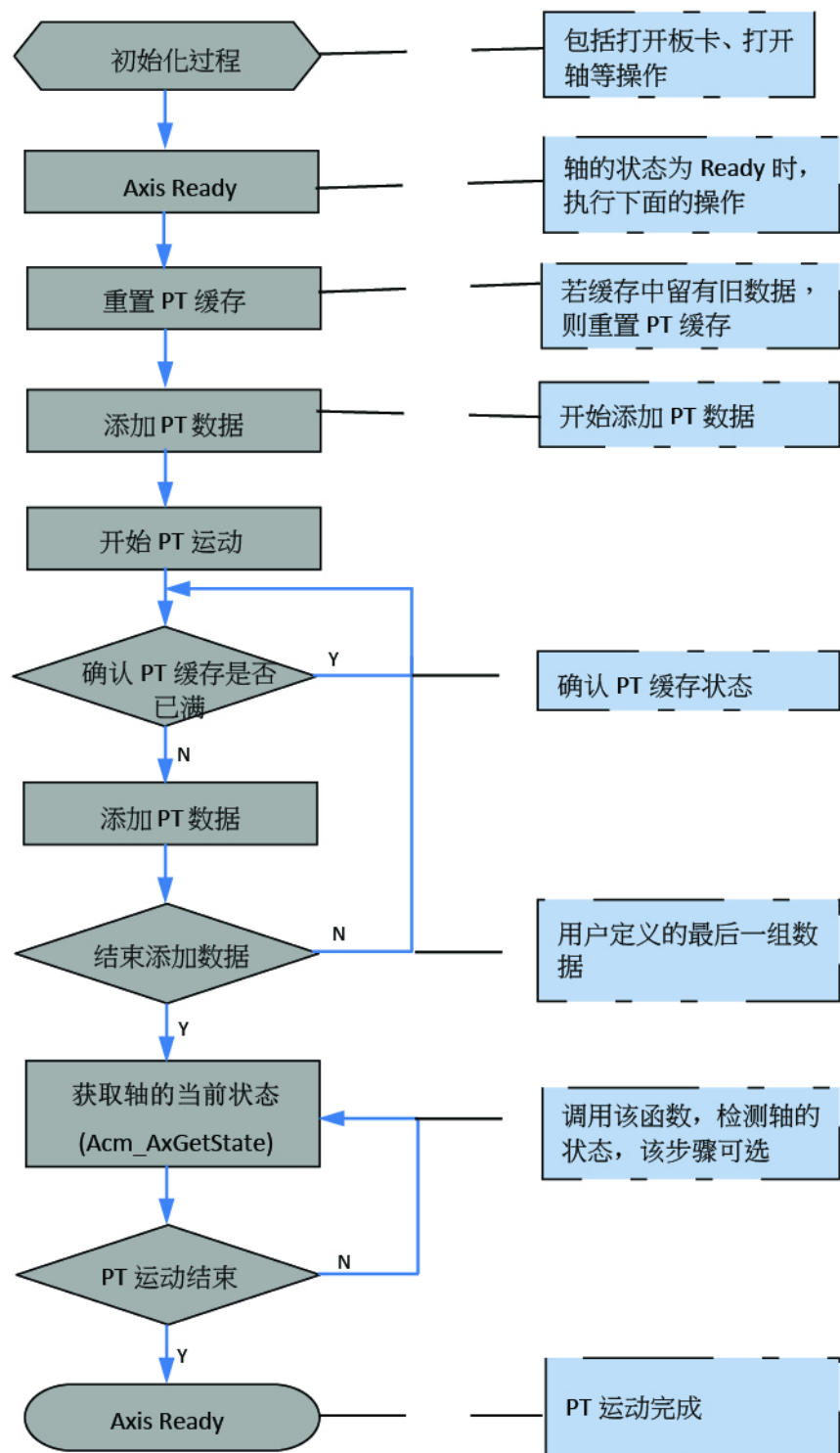


图 9.11: PT 运动流程图

9.2.11.4 例程

该例程实现 0 轴执行 PT 运动

VC 代码如下:

```
HANDm_Axishand[32]; //0 轴的 handle  
U32Ret; // 函数返回值
```

— 初始化过程 —

// 设置参数

```
Ret =Acm_AxResetPTData(m_Axishand[0]); // 重置 PT 缓存  
Ret =Acm_AxAddPTData(m_Axishand[0], 0, 0); // 添加第一笔 PT 数据  
Ret = Acm_AxAddPTData (m_Axishand[0], 5000, 1000); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 15000, 2000); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 20000, 3000); // 添加 PT 数据
```

// 执行 PT 运动且重复两次

```
Ret =Acm_AxStartPT(m_Axishand[0], 2);
```

// 持续添加 PT 数据

```
Ret =Acm_AxAddPTData (m_Axishand[0], 21024, 4024); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 23072, 5048); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 24096, 6072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 29096, 7072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 39096, 8072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 44096, 9072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 49096, 10072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 59096, 11072); // 添加 PT 数据  
Ret =Acm_AxAddPTData (m_Axishand[0], 64096, 61272); // 添加 PT 数据
```

// 读取轴的状态和位置

```
F64CmdPos;
```

```
F64 ActPos;
```

```
U16State;
```

```
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos); // 获取 0 轴的理论位  
Acm_AxGetActualPosition(m_Axishand[0], &ActPos); // 获取 0 轴的实际位置  
Acm_AxGetState(m_Axishand[0], &State); // 获取 0 轴的状态
```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

9. 2. 12PVT 运动

9. 2. 12. 1PVT 运动相关函数与属性

PVT 运动相关函数如下表 9. 20 所示，下表中的函数可直接调用。

表 9. 20：PVT 运动函数	
PT 运动函数	说明
Acm_AxResetPVTable	重置 PVT 缓存
Acm_AxLoadPVTable	加载 PVT 表到 PVT 缓存
Acm_AxStartPVT	开始 PVT 运动
Acm_AxGetState	获取轴的当前状态

9. 2. 12. 2重点说明

PVT 运动根据位置、速度和时间信息配置速度以满足下面所示的三阶方程，从该方程计算的适当参数可以获得更平滑的速度运动。、

$$p = at^3 + bt^2 + ct + d$$

$$\frac{dp}{dt} = v = 3at^2 + 2bt + c$$

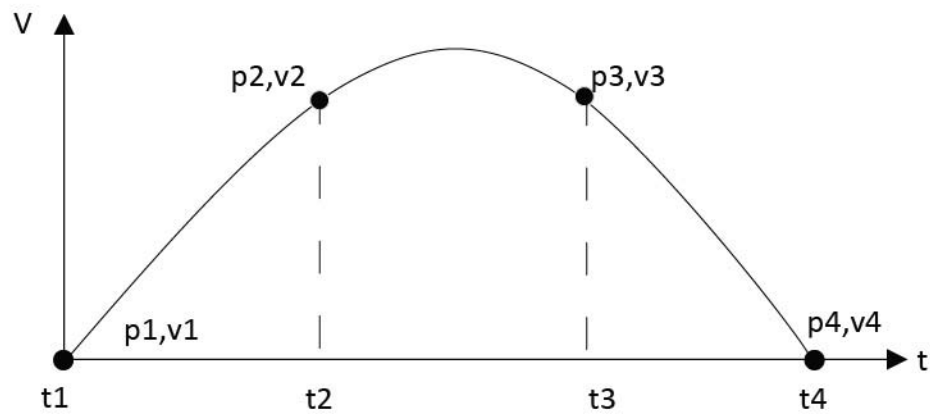
代入两个相邻点 和 到方程中，可以确定方程中参数 。

$$p_1 = at_1^3 + bt_1^2 + ct_1 + d$$

$$v_1 = 3at_1^2 + 2bt_1 + c$$

$$p_2 = at_2^3 + bt_2^2 + ct_2 + d$$

$$v_2 = 3at_2^2 + 2bt_2 + c$$



9. 2. 12. 3PVT 运动流程图

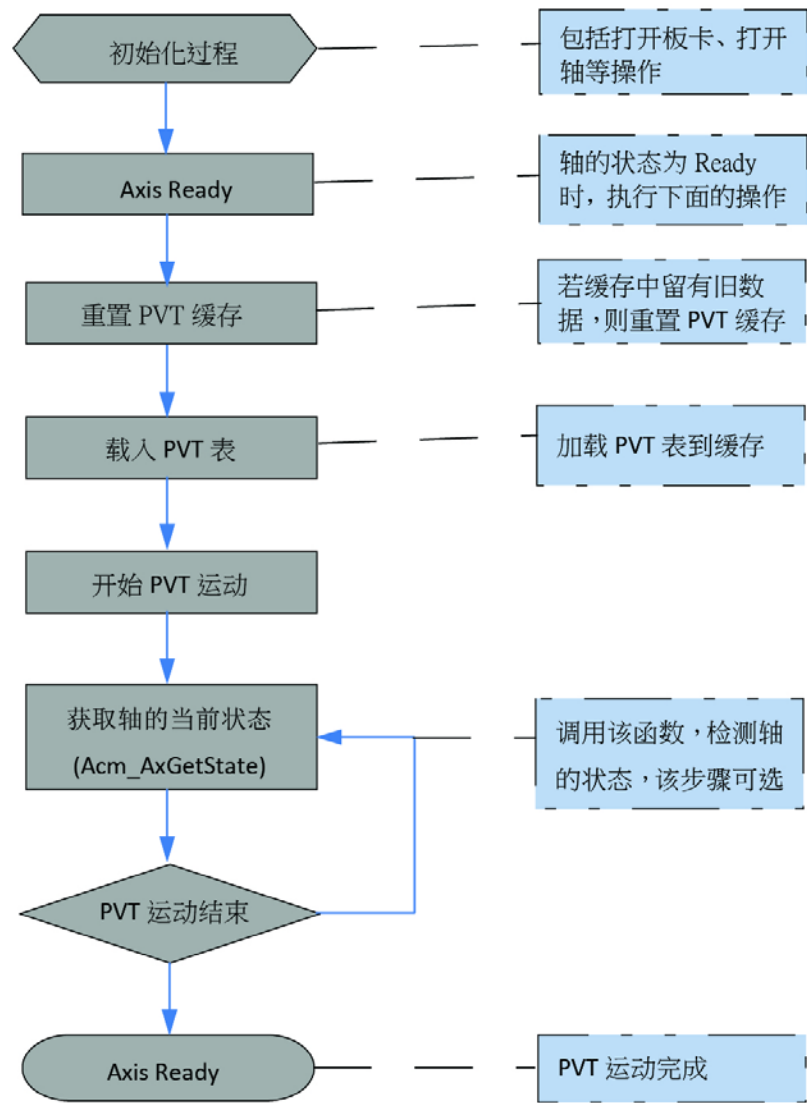


图 9. 12: PVT 运动流程图

9. 2. 12. 4例程

该例程实现 0 轴执行 PVT 运动

VC 代码如下:

```
HANDm_Axishand[32]; //0 轴的 handle
U32Ret; // 函数返回值
F64pos[4]={0, 5000, 15000, 20000}; // 位置表
F64vel[4]={0, 7500, 7500, 0}; // 速度表
F64t[4]={0, 1000, 2333, 3333}; // 时间表
```

— 初始化过程 —

```
// 设置参数
Ret =Acm_AxResetPVTable(m_Axishand[0]); // 重置 PVT 缓存
Ret =Acm_AxLoadPVTable(m_Axishand[0],pos,vel,t, 4); // 加载 PVT 表到缓存
```

// 执行 PVT 运动且重复两次

```
Ret =Acm_AxStartPVT (m_Axishand[0],2);
```

```

// 读取轴的状态和位置
F64 CmdPos;
F64 ActPos;
U16 State;
Acm_AxGetCmdPosition(m_Axishand[0], &CmdPos);    // 获取 0 轴的理论位
Acm_AxGetActualPosition(m_Axishand[0], &ActPos);  // 获取 0 轴的实际位置
Acm_AxGetState(m_Axishand[0], &State);           // 获取 0 轴的状态

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

9.3 插补运动模式

9.3.1 本节简介

插补运动在数控机床，切削加工工艺等数控装置中应用广泛，它可以实现多轴的协调运动。

研华运动控制卡提供如下插补运动模式：

- 直线插补
- 圆弧插补
- 螺旋插补

9.3.2 直线插补运动

9.3.2.1 直线插补运动相关函数与属性

直线插补运动相关函数如下表 9.21 所示，该表中的函数可在应用程序中直接调用。

表 9.21：直线插补运动相关函数

直线插补运动相关函数	说明
Acm_GpMoveLinearRel	命令群组执行相对线性插补
Acm_GpMoveLinearAbs	命令群组执行绝对线性插补
Acm_GpMoveDirectRel	命令群组执行相对直接线性插补
Acm_GpMoveDirectAbs	命令群组执行绝对直接线性插补
Acm_GpAddAxis	添加一个轴到指定群组
Acm_GpRemAxis	从指定群组中移除一个轴
Acm_GpClose	移除群组中的所有轴并关闭群组句柄
Acm_GpResetError	复位群组状态
Acm_GpChangeVel	命令群组在插补运动时改变速度
Acm_GpChangeVelByRate	按设定比例改变运行群组的运行速度
Acm_GpGetCmdVel	获取群组的当前的速度值
Acm_GpStopDec	命令该群组中的轴减速停止
Acm_GpGetState	获取群组的当前状态
Acm_GpStopEmg	命令该群组中的轴立刻停止（无减速）
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）

表 9.21：直线插补运动相关函数

Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

直线插补运动相关属性如下表 9.22 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

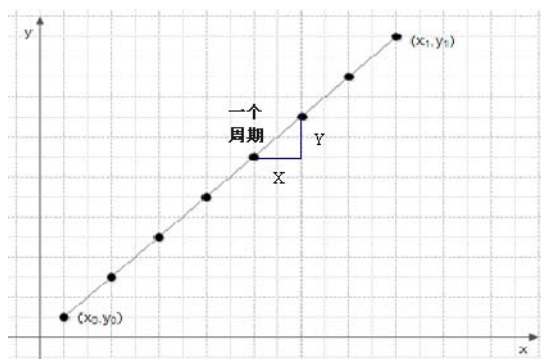
表 9.22：直线插补运动相关属性

参数	说明
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID
配置	说明
CFG_GpAxesInGroup	获取哪个（哪些）轴在该群组中的信息

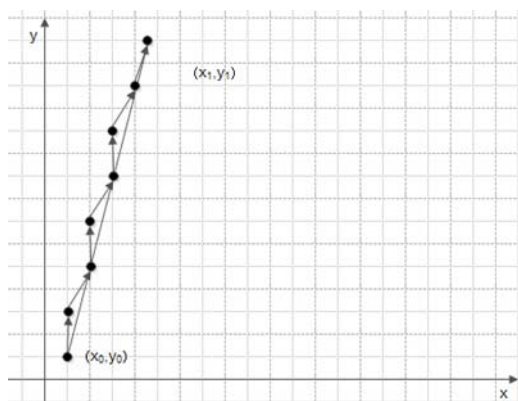
9.3.2.2 重点说明

直线插补运动首先将数据段所描述的曲线的起点、终点之间的空间进行数据密化，然后根据密化后的数据向各个坐标发出进给脉冲，对应每个脉冲，机床在相应的坐标方向上移动一个脉冲当量的距离，从而将工件加工出所需的轮廓形状。

如下图所示，假设某数控机床刀具在 xy 平面上，起点坐标 (X_0, Y_0) ，终点坐标 (X_1, Y_1) 。首先将两点间的空间进行数据密化，然后两点间的插补沿着直线上的点群来逼近。在一个运动周期内，X 轴、Y 轴根据 X、Y 坐标上的斜率发送脉冲。当 X、Y 坐标的斜率为 1:1 时，X 轴、Y 轴也将以 1:1 的比率发送脉冲。对应每个脉冲，机床将在相应的坐标方向上移动一定的距离，最终达到目标终点。



当 X、Y 坐标的斜率不为 1:1 时。如下所示，Y 轴斜率大于 X 轴，Y 轴将会移动一定距离并发送脉冲，然后 X 轴、Y 轴同时运行到一定位置后发送脉冲，最终到达轮廓终点位置。



研华运动控制卡提供两种直线插补模式：

线性插补：速度被分解为其中各个轴的向量速度，轴将以该速度运动。多数情况中，线性插补应用于以直角组合的轴。

直接线性插补：直接线性插补的运行速度设置为主轴（以距离最长的轴为主轴）的速度。主轴的速度即群组速度，其他轴会与主轴同时启动同时停止。例如设置群组的速度为 8000PPU/s，主轴将以 8000PPU/s 的速度运行。根据距离和速度，可以计算出主轴运行所用时间，从轴根据主轴运行时间和从轴的距离，计算出运行速度。多数情况中，直接插补应用于以斜角组合的轴。

9.3.2.3 直线插补运动流程图

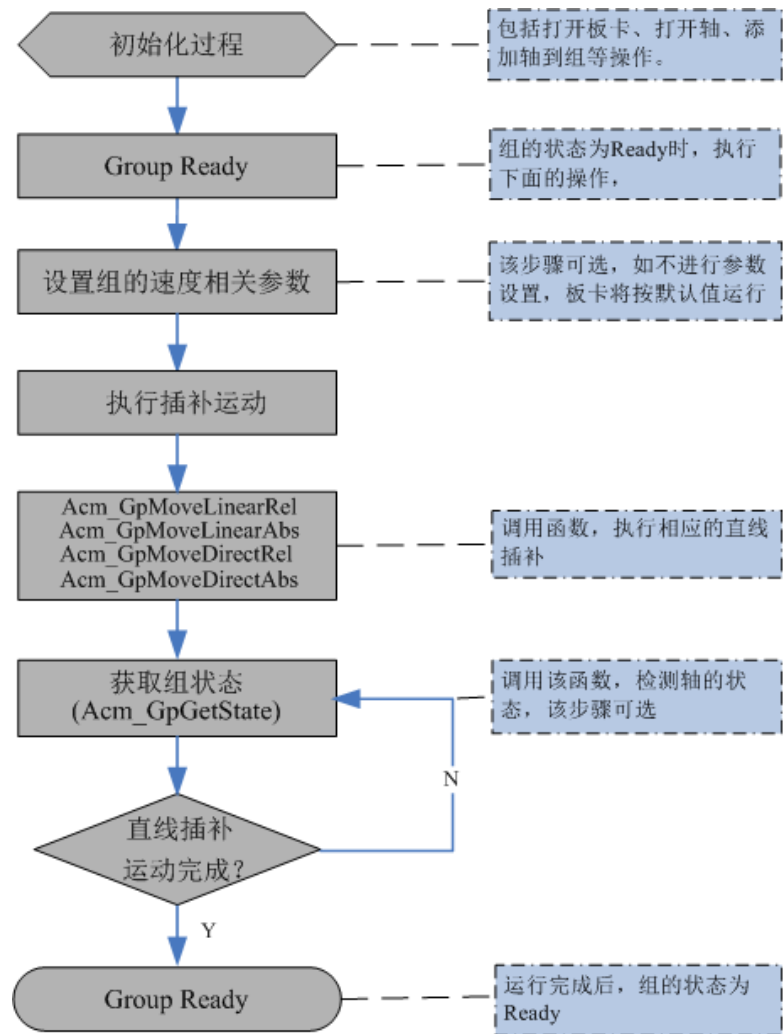


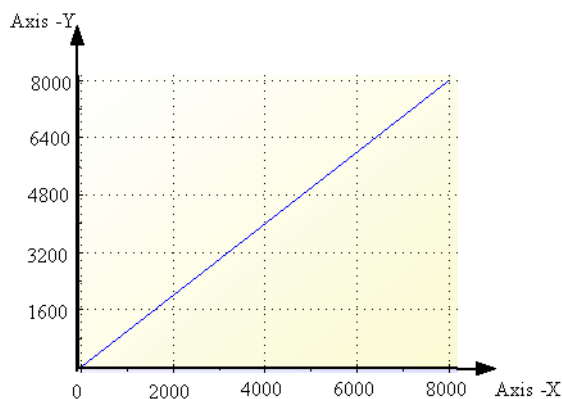
图 9.13: 直线插补运动流程图

9.3.2.4 例程

假设某数控机床刀具从原点出发，执行目标位置为（8000,8000）的直线插补。实现功能和相关的参数如下表所示。

功能	XY 轴完成相对直线插补（PCI-1203 运动控制卡）
初速度	2000PPU/s
运行速度	8000PPU/s
加速度	10000PPU/s ²
减速度	10000PPU/s ²
速度曲线形式	T 型速度曲线
起始位置	XY (0, 0)
目标位置	XY (8000, 8000)

实现的直线插补运动轨迹如下图所示：



VC 代码如下：

```

HAND    m_GpHand;           // 组的 handle
U32      Ret;                // 函数返回值
Double  m_End[2] = {8000, 8000};
--- 初始化过程 ---
// 设置参数
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 初速度 2000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000);   // 加速度 10000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpDec, 10000);   // 减速度 10000
Ret  = Acm_SetF64Property(m_GpHand, PAR_GpJerk, 0);      // T 型曲线
// 执行直线插补
Ret  = Acm_GpMoveRel(m_GpHand, m_End);                   // 相对直线插补
// 读取组的状态
U16   GpState;
Acm_GpGetState(m_GpHand, &GpState);                     // 获取组的状态

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Line 和 Direct 例程。

9.3.3 圆弧插补

9.3.3.1 圆弧插补运动相关函数与属性

圆弧插补运动相关函数如下表 9.23 所示，该表中的函数可在应用程序中直接调用。

表 9.23：圆弧插补相关函数

圆弧插补运动相关函数	说明
Acm_GpMoveCircularRel	命令群组执行相对 ARC 插补
Acm_GpMoveCircularAbs	命令群组执行绝对 ARC 插补
Acm_GpMoveCircularRel_3P	根据三个指定点执行相对 ARC 插补
Acm_GpMoveCircularAbs_3P	根据三个指定点执行绝对 ARC 插补
Acm_GpMove3DArcRel	根据终点等参数执行相对 3D ARC 插补
Acm_GpMove3DArcAbs	根据终点等参数执行绝对 3D ARC 插补
Acm_GpMove3DArcAbs_V	根据法向量等参数执行绝对 3D ARC 插补
Acm_GpMove3DArcRel_V	根据法向量等参数执行相对 3D ARC 插补
Acm_GpAddAxis	添加一个轴到指定群组
Acm_GpRemAxis	从指定群组中移除一个轴
Acm_GpClose	移除群组中的所有轴并关闭群组句柄
Acm_GpResetError	复位群组状态
Acm_GpChangeVel	命令群组在插补运动时改变速度
Acm_GpChangeVelByRate	按设定比例改变当前群组的运行速度
Acm_GpGetCmdVel	获取群组的当前的速度值
Acm_GpStopDec	命令该群组中的轴减速停止
Acm_GpStopEmg	命令该群组中的轴立刻停止（无减速）
Acm_GpGetState	获取群组的当前状态
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

圆弧插补相关属性如表 9.24 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.24：圆弧插补运动相关函数

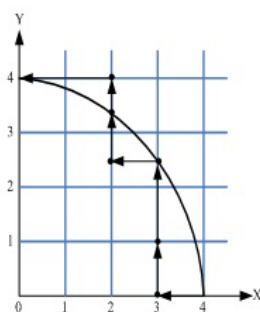
参数	说明
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID
配置	说明
CFG_GpAxesInGroup	获取哪个（哪些）轴在该群组中的信息

9.3.3.2 重点说明

圆弧插补是给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制刀具沿这些点运动，加工出圆弧曲线。研华运动控制卡支持 2 轴圆弧插补和 3 轴圆弧插补，并提供多样化的输入方式，以满足各种应用。下面分别介绍 2 轴 3 轴圆弧运动及各种命令建构方式。

1. 2 轴圆弧插补

2 轴圆弧插补所运行的参考平面是 XY、YZ、XZ 中的任一平面。假设某数控机床刀具在 xy 平面第一象限走一段反向圆弧，圆心为原点，半径为 4，起点 A(4, 0)，终点 B(0, 4)，其圆弧插补的加工过程如下图所示。



2 轴圆弧插补的描述方式有如下三种：

方法 1：根据起点、圆心、终点执行圆弧插补

方法 2：根据三点坐标确定圆弧，执行圆弧插补

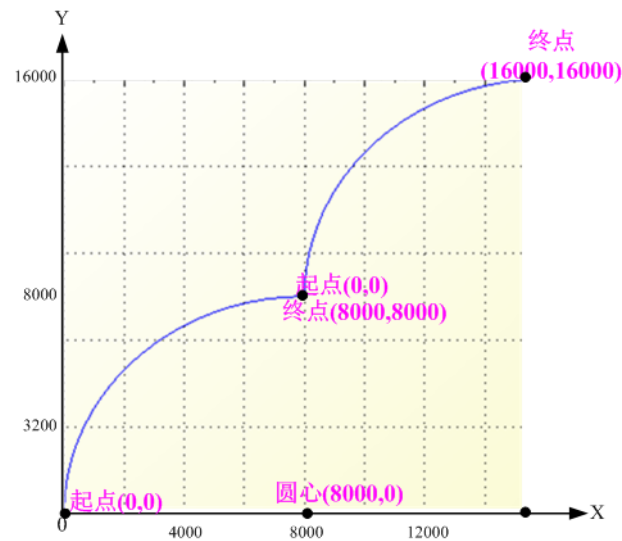
方法 3：根据圆心和角度确定圆弧，执行圆弧插补

现分别介绍如下：

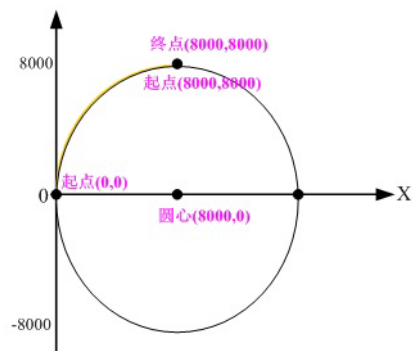
方法 1：根据起点、圆心、终点执行圆弧插补

调用 Acm_GpMoveCircularRel/Acm_GpMoveCircularAbs 函数，输入圆心和终点坐标，执行相对 / 绝对圆弧插补。

例如设定圆心坐标 (8000, 0)，终点坐标 (8000, 8000)，执行相对圆弧插补。则连续执行两次的结果如下图所示。即相对圆弧插补以当前理论位置坐标为起点。



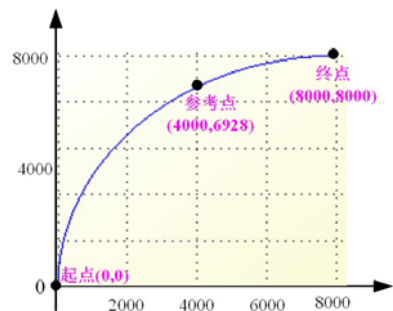
连续执行两次绝对圆弧插补，执行结果如下图所示。



第一次执行结果为图中的黄色部分，第二次执行以第一次执行的终点为起点，执行结果为完成圆弧。绝对圆弧插补以绝对零点指令为参考位置。

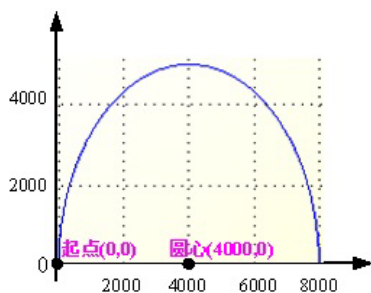
方法 2：根据三点坐标确定圆弧，执行圆弧插补

调用 Acm_GpMoveCircularRel_3P/Acm_GpMoveCircularAbs_P 函数，输入参考点和终点坐标，执行相对 / 绝对圆弧插补。如下图设置参考点坐标 (4000, 6928)，终点坐标 (8000, 8000)，执行相对圆弧插补，默认起点为 (0, 0)，结果如图所示。



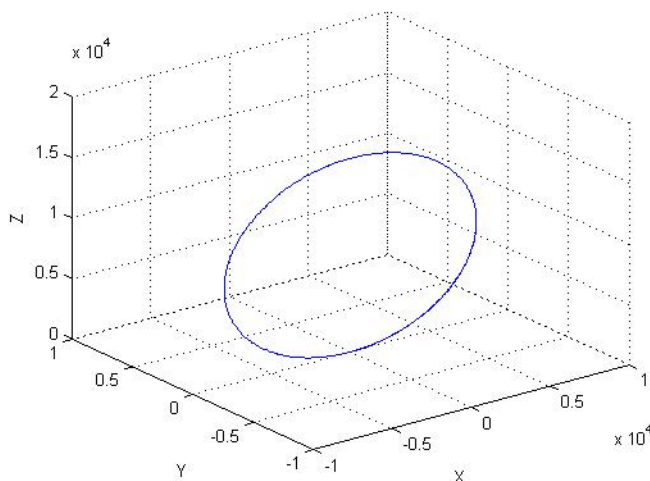
方法 3：根据圆心和角度确定圆弧，执行圆弧插补

调用 `Acm_GpMoveCircularRel_Angle/Acm_GpMoveCircularAbs_Angle` 函数，输入圆心坐标和角度，执行相对 / 绝对圆弧插补。如下图所示，设置圆心坐标为 (4000, 0)，角度为 180，则执行结果如下图所示。



2. 3 轴圆弧插补

3 轴圆弧插补所运行的空间是 3 维空间。如下图所示，在 XYZ 平面内，X、Y、Z 轴执行圆弧插补。



3 轴圆弧插补的描述方式分为：

方法 1：根据圆心、终点和方向确定圆弧，执行 3 轴圆弧插补

方法 2：根据圆心、法向量、角度和方向确定圆弧，执行 3 轴圆弧插补

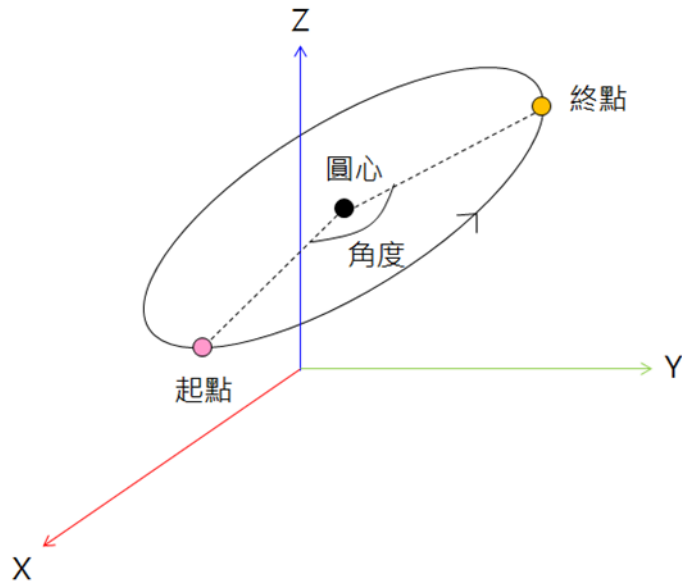
现分别介绍如下：

方法 1：根据圆心、终点和方向确定圆弧，执行 3 轴圆弧插补

给定圆心、终点坐标及运行方向，调用 `Acm_GpMove3DArcAbs`、`Acm_GpMove3DArcRel` 函数，即可执行 3 轴圆弧插补，该方法有如下限制：

1. 无法执行角度为 180 度的半圆
2. 无法执行角度为 360 度的全圆

圆心、终点、180/360 度的圆在一条直线上，一条直线不能确定一个平面。遇到此种情况，可使用方法二，根据法向量确定圆弧所在平面，执行圆弧插补。三轴圆弧插补如下图所示。



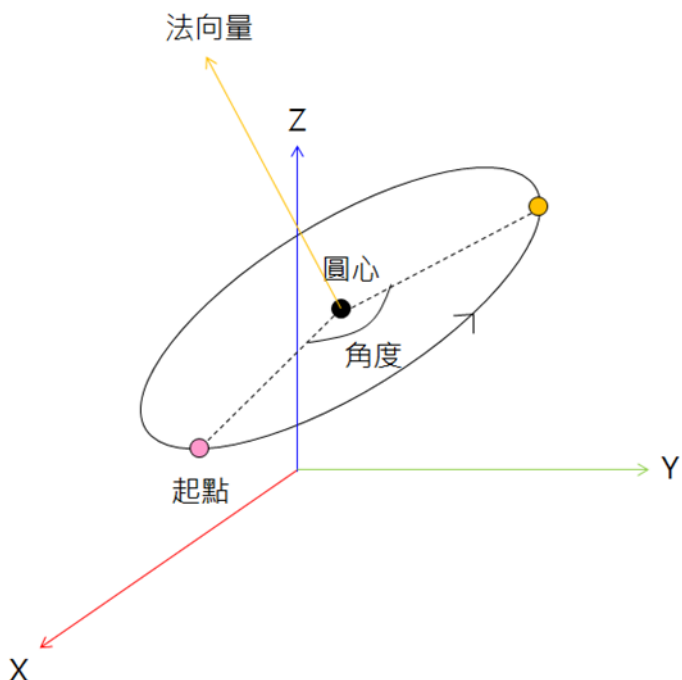
三轴圆弧插补（方法一）

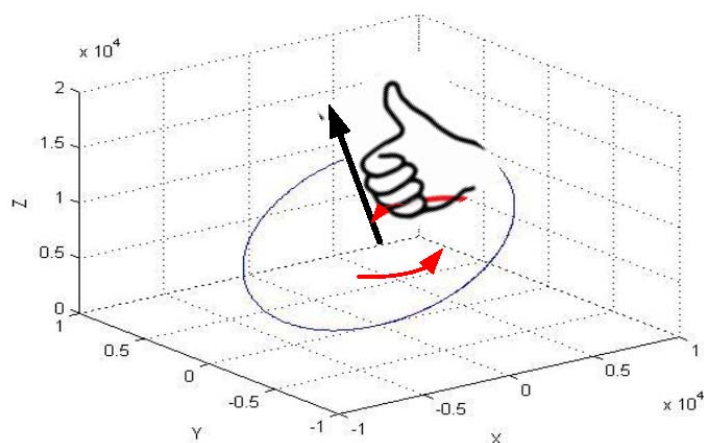
方法 2：根据圆心、法向量、角度和方向确定圆弧，执行圆弧插补。

给定圆心、法向量、角度和方向，调用 `Acm_GpMove3DArcAbs_V`、`Acm_GpMove3DArcAbs_V` 函数即可执行 3 轴圆弧插补。

3 轴圆弧插补圆弧旋转方向（CW 或 CCW）按如下方式确定：

- 若法向量与 Z 轴之间的角度为锐角（ $<90^\circ$ ），则法向量的方向为正向，反之则为反向。
- 若法向量与 Z 轴之间的角度为直角，且法向量与 Y 轴之间的角度为锐角（ $<90^\circ$ ），则方向量的方向为正向。
- 若法向量与 Z 轴和 Y 轴之间的角度为直角，且法向量与 X 轴之间的角度为锐角（ $<90^\circ$ ），则方向量的方向为正向。





9.3.3.3 圆弧插补流程图

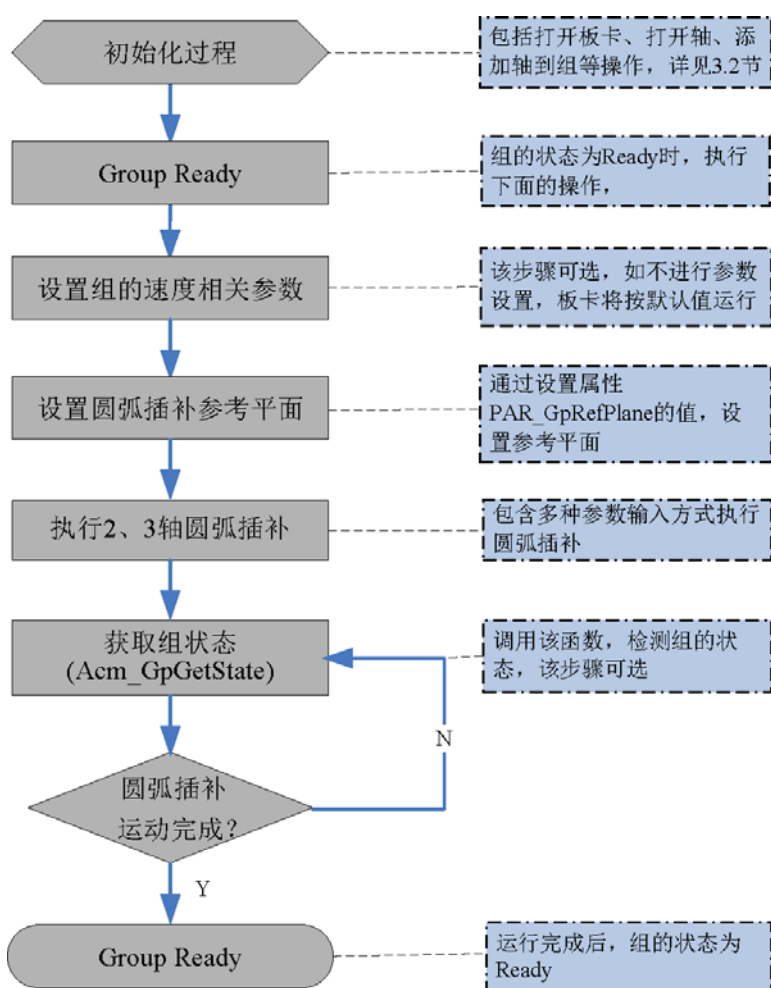


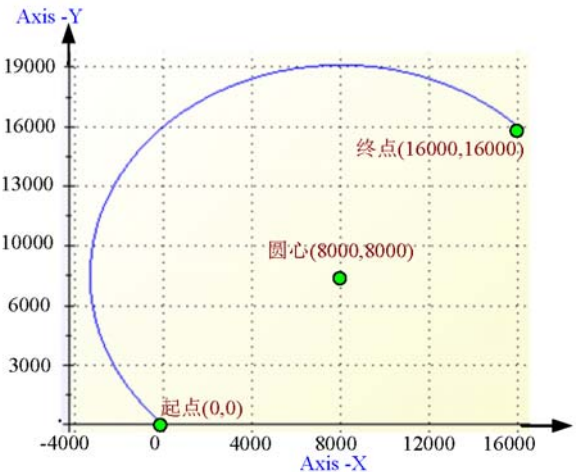
图 9.14: 圆弧插补运动流程图

9.3.3.4 例程

例程实现某数控机床刀具在 XY 平面内执行相对圆弧插补。圆心 (4000, 0)，终点 (8000, 8000)，运行方向正向，默认起点为 (0, 0)。实现功能和相关的参数如下表所示。

功能	XY 轴执行相对圆弧插补 (PCI-1203 运动控制卡)
初速度	2000PPU/s
运行速度	8000PPU/s
加速度	10000PPU/s ²
减速度	10000PPU/s ²
速度曲线形式	T 型速度曲线
起点	(0, 0)
圆心	(4000, 0)
终点	(8000, 8000)
方向	正向 (CW)
参考平面	XY 平面

实现的圆弧插补运动轨迹如下图所示。



VC 代码如下：

```
HAND    m_GpHand;           // 组的 handle
U32     Ret;                 // 函数返回值
U32     m_GpReferencePlane; // 参考平面
Double  CenterArray[3] = {8000, 8000};
Double  EndArray[3] = {16000, 16000};
U32     AxisNum;             // 添加到群组的轴数
I16     Dir;                 // 圆弧插补执行反向
--- 初始化过程 ---
// 设置参数
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000); // 加速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpDec, 10000); // 减速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpJerk, 0); // T 型曲线
// 设置参考平面为 xy 平面
m_GpReferencePlane = 0;
Ret = Acm_SetU32Property(m_GpHand, PAR_GpRefPlane, m_GpReferencePlane);
// 执行圆弧插补运动
```

```
AxisNum =2;
Ret=Acm_GpMoveCircularRel (m_GpHand, CenterArray, EndArray, &AxisNum, 0);
// 读取组的状态
U16    GpState;
Acm_GpGetState(m_GpHand, &GpState);          // 获取组的状态
```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 ARC 例程。

9.3.4 螺旋插补运动

9.3.4.1 螺旋插补运动相关函数与属性

螺旋插补运动相关函数如下表 9.25 所示，该表中的函数可在应用程序中直接调用。

表 9.25：螺旋插补运动相关函数

螺旋插补运动相关函数	说明
Acm_GpMoveHelixAbs	通过圆心等参数执行绝对螺旋插补
Acm_GpMoveHelixRel	通过圆心等参数执行相对螺旋插补
Acm_GpMoveHelixAbs_3P	通过三个指定点执行绝对螺旋插补
Acm_GpMoveHelixRel_3P	通过三个指定点执行相对螺旋插补
Acm_GpMoveHelixAbs_Angle	通过角度等参数执行绝对螺旋插补
Acm_GpMoveHelixRel_Angle	通过角度等参数执行相对螺旋插补
Acm_GpAddAxis	添加一个轴到指定群组
Acm_GpRemAxis	从指定群组中移除一个轴
Acm_GpClose	移除群组中的所有轴并关闭群组句柄
Acm_GpResetError	复位群组状态
Acm_GpChangeVel	命令群组在插补运动时改变速度
Acm_GpChangeVelByRate	按比例改变运动群组的运行速度
Acm_GpGetCmdVel	获取群组的当前的速度值
Acm_GpStopDec	命令该群组中的轴减速停止
Acm_GpStopEmg	命令该群组中的轴立刻停止（无减速）
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_GpGetState	获取群组的当前状态
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）

属性如下表 9.26 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

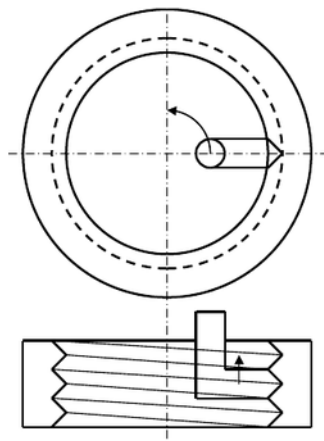
表 9.26：螺旋插补运动相关函数	
参数	说明
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID

9.3.4.2 重点说明

螺旋插补是指定两轴在 XY、YZ、XZ 平面内做圆弧插补，其他轴跟随做直线插补的合成运动。

执行圆弧插补的两轴与添加到组的轴的 ID 有关，例如轴添加到组的顺序为 3、4、2、1 轴，根据轴的 ID(0 轴的 ID=0...)，执行圆弧插补的轴为 1、2 轴，3、4 轴做跟随运动。在螺旋插补中，一般使用 X、Y 作为圆弧插补，Z 轴为直线垂直于 XY 平面，如果添加一个与圆弧平面垂直的旋转轴 C，使刀具与运动路径成一个固定的夹角，可保证刀具始终垂直于切割面。

如下图所示为螺旋插补的正视图侧视图。



当任一轴发生 Error 时，所有轴停止运行。群组的速度通过群组的速度属性进行设置。螺旋插补适用于 3 轴或 3 轴以上联动的系统中。

研华运动控制卡提供多种螺旋插补参数输入方式。

方法 1：根据圆心、终点、方向执行螺旋插补

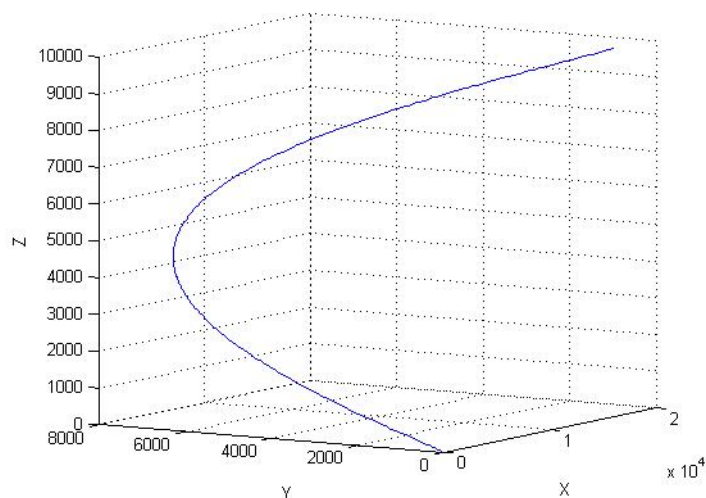
方法 2：根据圆心、角度、方向执行螺旋插补

方法 3：根据三点坐标执行螺旋插补

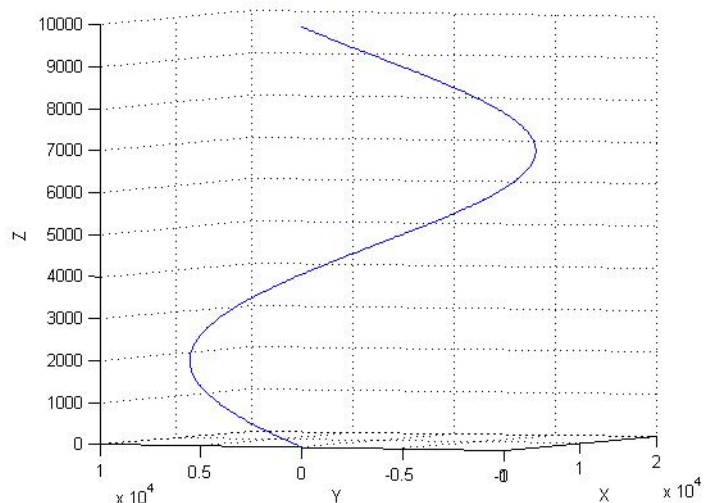
现分别介绍如下：

方法 1：根据圆心、终点、方向执行螺旋插补

调用 `Acm_GpMoveHelixRel/ Acm_GpMoveHelixAbs` 函数，输入圆心、终点坐标、轴数及方向执行螺旋插补。如下图所示圆心坐标 (8000, 0, 0)，终点坐标 (16000, 0, 10000)，圆弧插补轴为 X、Y 轴，跟随轴 Z 轴。

**方法 2：根据圆心、角度、终点、方向执行螺旋插补**

调用 `Acm_GpMoveHelixRel_Angle/ Acm_GpMoveHelixAbs_Angle` 函数，输入圆心、角度、终点、轴数及方向执行螺旋插补。如下图所示圆心坐标 (8000, 0, 0)，终点坐标 (1080, 0, 10000)，其中 1080 为旋转角度，方向正向，圆弧插补轴为 X、Y 轴，跟随轴 Z 轴。

**方法 3：根据三点坐标执行螺旋插补**

调用 `Acm_GpMoveHelixRel/ Acm_GpMoveHelixAbs` 函数，输入螺旋上的三点坐标，执行螺旋插补。

9.3.4.3 螺旋插补运动流程图

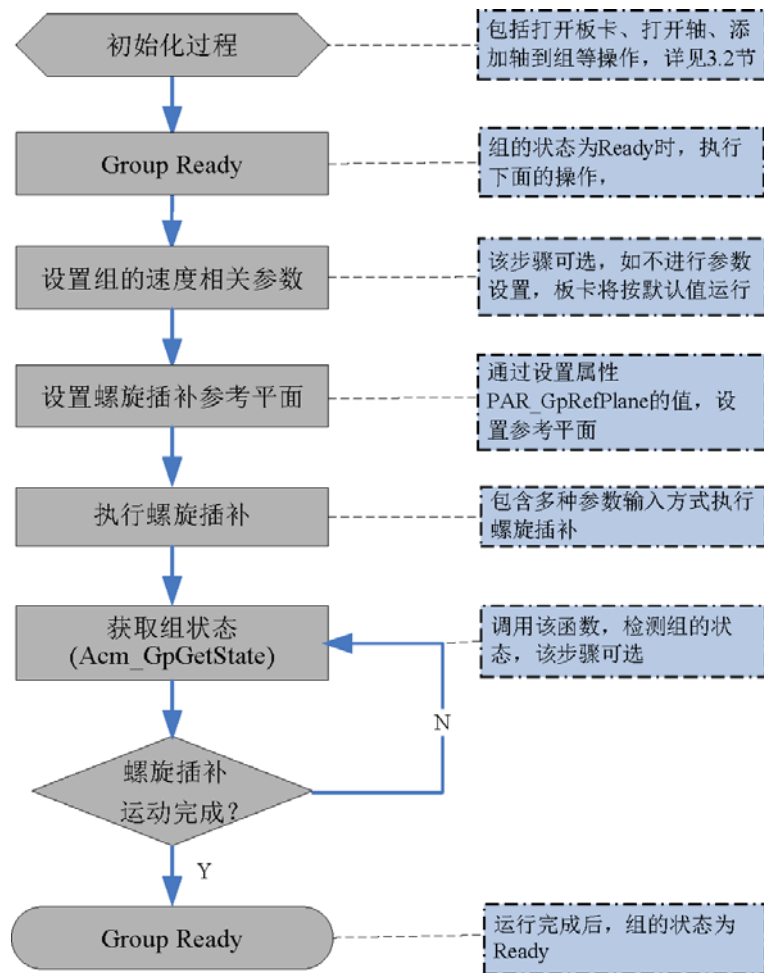


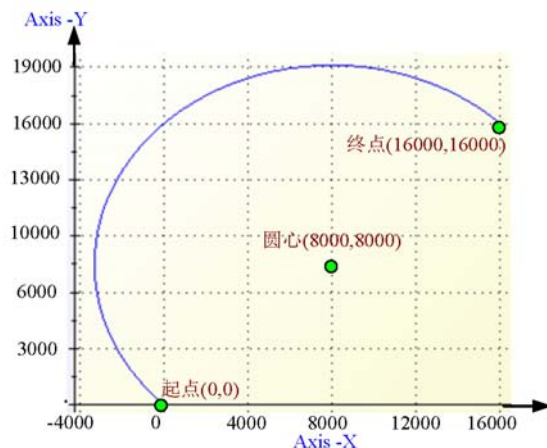
图 9.15: 螺旋插补运动流程图

9.3.4.4 例程

例程实现某数控机床刀具在 XY 平面内执行相对螺旋插补，指定 0 轴 1 轴执行圆弧插补，2、3 轴跟随 0、1 轴执行点到点运动。圆心 (4000, 0)，终点 (8000, 8000)，运行方向正向，默认起点为 (0, 0)，点到点运动距离为 10000。实现功能和相关的参数如下表所示。

功能	XY 平面内执行相对螺旋插补 (PCI-1203 运动控制卡)
初速度	2000PPU/s
运行速度	8000PPU/s
加速度	10000PPU/s ²
减速度	10000PPU/s ²
速度曲线形式	T 型速度曲线
起点	(0, 0)
圆心	(4000, 0)
终点	(8000, 8000)
方向	正向 (CW)
参考平面	XY 平面

实现的螺旋插补运动轨迹如下图所示：



VC 代码如下：

```

HAND    m_GpHand;           // 组的 handle
U32      Ret;                // 函数返回值
U32      m_GpReferencePlane; // 参考平面
Double   m_CenterArray[3] = {8000, 0, 0};
Double   m_EndArray[4] = {16000, 16000, 10000, 10000};
U32      AxisNum;            // 添加到群组的轴数
I16      Dir;                // 螺旋插补执行反向
--- 初始化过程 ---
// 设置参数
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000);    // 加速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpDec, 10000);    // 减速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpJerk, 0);       // T 型曲线
// 设置参考平面为 xy 平面
m_GpReferencePlane = 0;
Dir = 0;
Ret = Acm_SetU32Property(m_GpHand, PAR_GpRefPlane, m_GpReferencePlane);
// 执行螺旋插补运动
Ret = Acm_GpMoveHelixRel(m_GpHand, m_CenterArray, m_EndArray, &AxisNum, Dir);
// 读取组的状态
U16    GpState;
Acm_GpGetState(m_GpHand, &GpState); // 获取组的状态

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Helix 例程。

9.3.5 群组的事件

9.3.5.1 群组事件相关 API 与属性

群组事件相关函数如下表 9.27 所示，下表中的函数可直接调用。

表 9.27：轴的事件相关函数	
群组事件函数	说明
Acm_EnableMotionEvent	启用轴和群组的事件状态。
Acm_CheckMotionEvent	检测轴和群组的事件状态
Acm_GpMoveLinearRel	命令群组执行相对线性插补
Acm_GpMoveLinearAbs	命令群组执行绝对线性插补
Acm_GpMoveDirectRel	命令群组执行相对直接线性插补
Acm_GpMoveDirectAbs	命令群组执行绝对直接线性插补
Acm_GpMoveCircularRel	命令群组执行相对 ARC 插补
Acm_GpMoveCircularAbs	命令群组执行绝对 ARC 插补
Acm_GpMoveCircularRel_3P	根据三个指定点执行相对 ARC 插补
Acm_GpMoveCircularAbs_3P	根据三个指定点执行绝对 ARC 插补
Acm_GpMove3DArcRel	根据终点等参数执行相对 3D ARC 插补
Acm_GpMove3DArcAbs	根据终点等参数执行绝对 3DARC 插补
Acm_GpMove3DArcAbs_V	根据法向量等参数执行绝对 3D ARC 插补
Acm_GpMove3DArcRel_V	根据法向量等参数执行相对 3D ARC 插补
Acm_GpMoveHelixAbs	通过圆心等参数执行绝对螺旋插补
Acm_GpMoveHelixRel	通过圆心等参数执行相对螺旋插补
Acm_GpMoveHelixAbs_3P	通过三个指定点执行绝对螺旋插补
Acm_GpMoveHelixRel_3P	通过三个指定点执行相对螺旋插补

9.3.5.2 重点说明

群组事件包括 Motion Done、VHStart、VHEnd。

Motion Done 事件：群组运动结束时，硬件触发产生的中断事件。

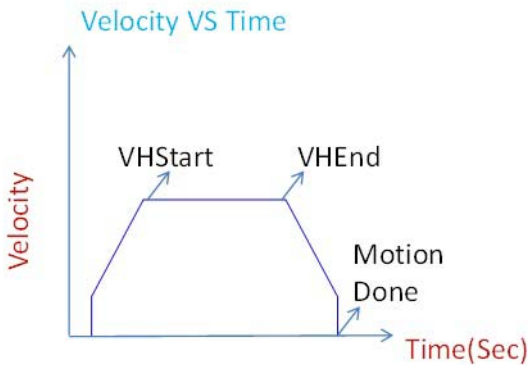
VHStart 事件：群组高速度运行开始时，硬件触发产生的中断事件。

VHEnd 事件：群组高速度运行结束时，硬件触发产生的中断事件。

注！ *VHStart* 和 *VHEnd* 分别表示运行速度开始和运行速度结束。



高速度指运行速度，如下图所示：



检测群组事件过程如下：

- 首先调用 `Acm_EnableMotionEvent` 函数，设置参数 `GpEnableEvtArray` 的值，启用 / 禁用（触发 / 不触发中断事件）群组的中断事件。参数说明如下：
`GpEnableEvtArray[N]`：每个元素表示群组的事件。N 的取值为 0~2，分别代表 `Gp_Motion_Done`、`Gp_VH_START`、`Gp_VH_END` 事件。
`Bitn`：数组元素每一位，设置元素每位的值，以启用 / 禁用群组事件。
`Bit n = 1`：启用事件
`Bit n = 0`：禁用事件
n 为群组 ID。例如 PCI-1203 板卡共有三个群组，则 n 的取值为 0~2，分别表示群组 0、1、2 的中断事件。群组 ID 通过 `PAR_GpGroupID` 属性获取。

<code>GpEnableEvtArray[N]</code>	<code>Bitn</code> (n 为群组 ID，与板块有关)
<code>GpEnableEvtArray[0]</code>	<code>Evt_GpnMotion_DONE (n=0~31)</code>
<code>GpEnableEvtArray[1]</code>	<code>Evt_GpnVH_START (n=0~31)</code>
<code>GpEnableEvtArray[2]</code>	<code>Evt_GpnVH_END (n=0~31)</code>

- 创建一个新的线程，调用 `Acm_CheckMotionEvent` 函数，检测事件。
该函数的参数 `GpEvtStatusArray` 将返回每个群组的中断事件状态。
`GpEvtStatusArray [N]`：每个元素表示群组的事件。N 的取值为 0~2，分别代表 `Gp_Motion_Done`、`Gp_VH_START`、`Gp_VH_END` 事件。
`Bitn`：数组元素的每一位，读取每位的值，判断是否发生中断事件。
`Bit n = 1`：事件发生
`Bit n = 0`：事件未事件
n 为群组 ID。如 PCI-1203 板卡共有三个群组，则 n 的取值为 0~2，分别表示群组 0、1、2 的事件。群组 ID 通过 `PAR_GpGroupID` 属性获取。

<code>GpEnableEvtArray[N]</code>	<code>Bitn</code> (n 为群组 ID，与板块有关)
<code>GpEnableEvtArray[0]</code>	<code>Evt_GpnMotion_DONE (n=0~31)</code>
<code>GpEnableEvtArray[1]</code>	<code>Evt_GpnVH_START (n=0~31)</code>
<code>GpEnableEvtArray[2]</code>	<code>Evt_GpnVH_END (n=0~31)</code>

9.3.5.3 群组事件流程图

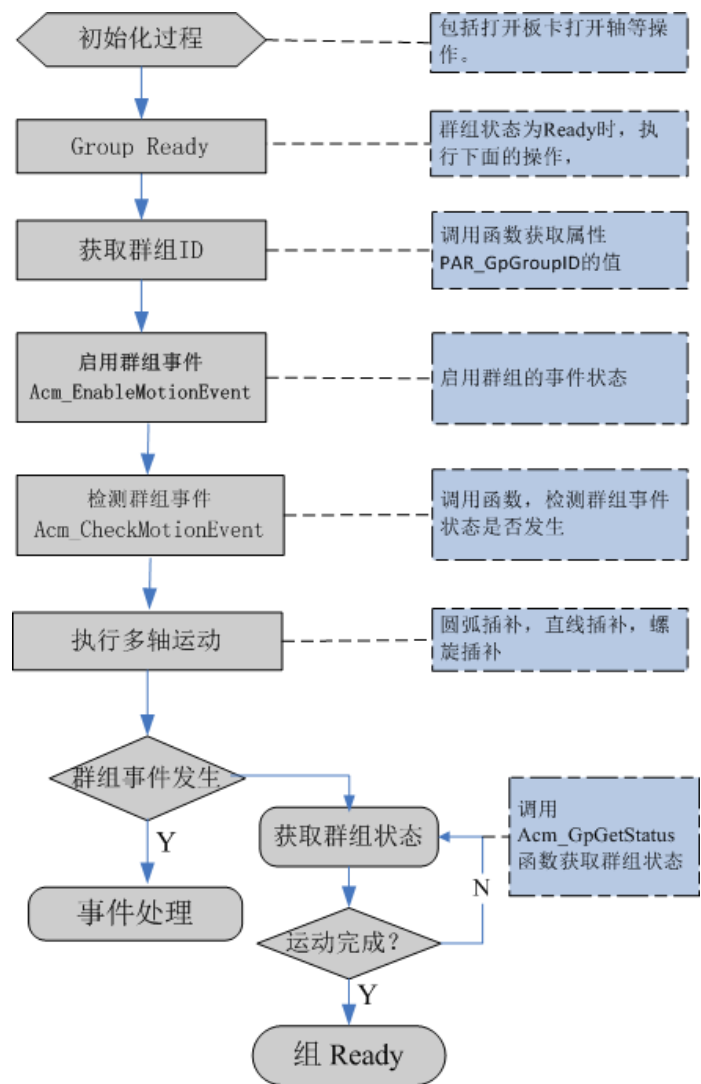


图 9.16: 群组事件流程图

9.3.5.4 例程

例程实现检测群组事件，设置的速度等参数值如下表所示。

功能	检测群组的 Motion Done、VHStart、VHEnd 事件
板卡	PCI-1203
运动类型	两轴直线插补
目标位置	{10000, 10000}

VC 代码如下：

```
HAND    m_GpHand;
U32     PropertyVal =0;          // 群组 ID
U32     Ret;
U32     AxEnableEvtArray[4];
U32     GpEnableEvt[3];
U32     m_ulAxisCount=4;        // 板卡轴数
BOOL    m_bInit ;
CWinThread* pThreadObject;      // 线程指针
--- 初始化过程 ---
```

```

Acm_GetU32Property(m_GpHand, PAR_GpGroupID, &PropertyVal); // 获取群组 ID
GpEnableEvt[0] |= EVT_GP1_MOTION_DONE << PropertyVal; // Enable motion done
GpEnableEvt[1] |= EVT_GP1_VH_START << PropertyVal;      // Enable VHStart
GpEnableEvt[2] |= EVT_GP1_VH_END << PropertyVal;        // Enable VHEnd
Ret = Acm_EnableMotionEvent(m_Devhand, AxEnableEvtArray, GpEnableEvt,
m_ulAxisCount, 3);                                     // Enable Event
// 开辟线程，检测群组事件是否发生
pThreadObject = AfxBeginThread( (AFX_THREADPROC) CDlg::CheckEvtThread,
this, THREAD_PRIORITY_TIME_CRITICAL, 0, 0, NULL); // 创建线程对象
UINT CDlg::CheckEvtThread(LPVOID ThreadArg)           // 检测事件的线程函数
{ U32 Ret;
U32 AxEvtStatusArray[4];
U32 GpEvtStatusArray[3];
CDlg*pThreadInfo;
pThreadInfo = (CDlg*)ThreadArg;
while (pThreadInfo->m_bInit)
{Ret = Acm_CheckMotionEvent(pThreadInfo->m_Devhand, AxEvtStatusArray,
GpEvtStatusArray, pThreadInfo->m_ulAxisCount, 3, 10000); // 检测事件状态
If (Ret == Success) {... 判断事件是否发生及事件处理 }}
}

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 EVENT 例程。

9.4 跟随运动模式

9.4.1 本节简介

跟随运动模式能够将多轴连接起来，实现精确的同步运动。将被跟随的轴叫主轴，跟随轴叫从轴，从轴可以跟随主轴的规划位置运行。

研华运动控制卡提供如下跟随运动模式：

- 电子齿轮
- 电子凸轮
- 龙门控制
- 切向跟随

9.4.2 电子齿轮运动

9.4.2.1 电子齿轮运动相关函数与属性

电子齿轮运动相关函数如下表 9.28 所示，该表中的函数可在应用程序中直接调用。

表 9.28：电子齿轮运动相关函数	
电子齿轮运动相关函数	说明
Acm_AxGearInAx	按照比例开始从轴和主轴之间的齿轮同步
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveVel	命令轴按照规定速度进行没有终点的运动
Acm_AxStopDec	命令轴按设定的减速度停止运行
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

电子齿轮运动相关属性如下表 9.29 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.29：电子齿轮运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置 / 获取速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.4.2.2 重点说明

电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置、编码器位置运行。

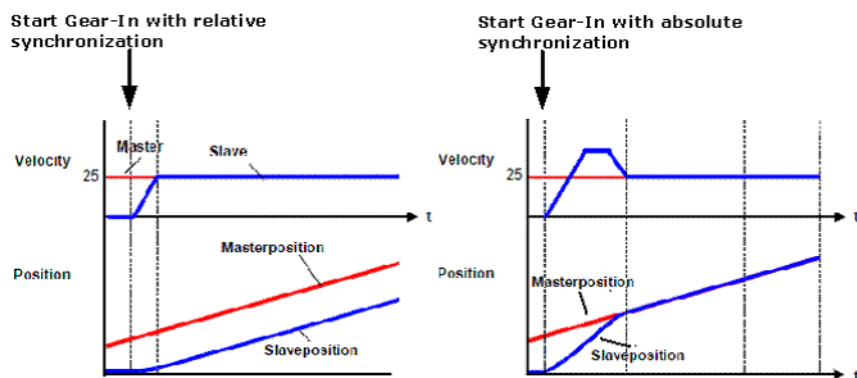
调用 Acm_AxGearInAx 函数，设定电子齿轮比，主轴和从轴之间将开始按设定的比例开始电子齿轮同步。可根据实际需求，多次调用该函数，以设定从轴和主轴的同步关系。

电子齿轮比：分子 / 分母。如果值为正数，那么从轴将以与主轴相同的方向运动；否则，从轴将以与主轴相反的方向运动。当主轴速度变化时，从轴会根据设定的电子齿轮比自动改变速度。

在设定电子齿轮同步关系时，可设定同步关系是相对的还是绝对的。

相对同步关系：从轴将不会补偿主轴的位置偏移。

绝对同步关系：从轴将补偿主轴的偏移。在这种模式下，如果起始位置不为 0，则从轴的加速度和减速需大于 1000PPS。如下图所示，左边的图展示主轴与从轴为相对同步关系，主轴与从轴最终会保持相对位置，但具有相同的速度；右边的图展示主轴与从轴为绝对关系，主轴与从轴最终会具有相同的位置及速度。



9.4.2.3 电子齿轮运动流程图

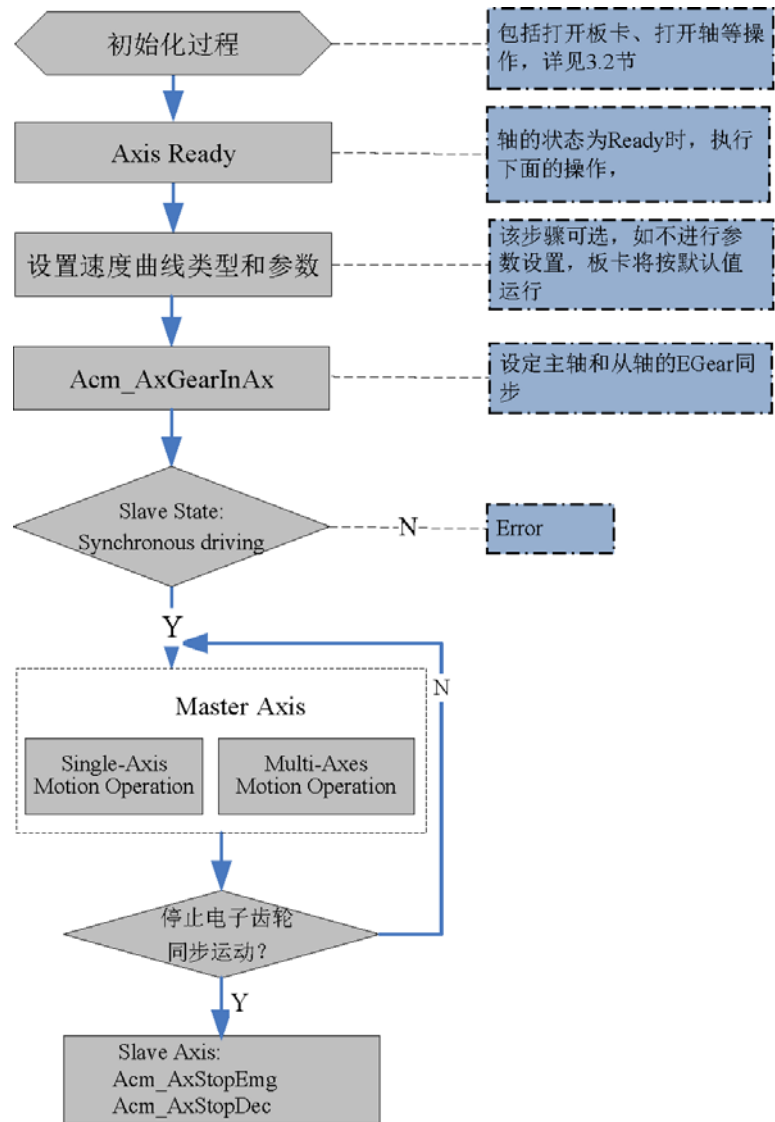


图 9.17: 电子齿轮流程图

9.4.2.4 例程

例程实现 0 轴为主轴, 1 轴为从轴, 从轴跟随主轴执行电子齿轮运动, 电子齿轮比为 10:1。实现功能和相关参数如下表所示。

功能	1 轴跟随 0 轴执行电子齿轮运动
主轴初速度	2000PPU/s
主轴运行速度	8000PPU/s
主轴加速度	10000PPU/s ²
主轴减速度	10000PPU/s ²
主轴速度曲线形式	T 型速度曲线
主轴起始位置	0
主轴目标位置	10000
电子齿轮比分子	10
电子齿轮比分母	1
从轴参考主轴位置	理论位置
主轴与从轴同步关系	相对同步关系

VC 代码如下:

```

HAND    m_Axhand[32];          // 轴的 handle
U32      Ret;                  // 函数返回值
int      m_Num =10;            // 电子齿轮比分子
int      m_Den=1;              // 电子齿轮比分母
U32      RefSrc=0;              // 从轴参考主轴理论位置
U32      Mode=0;                // 相对同步关系
--- 初始化过程 ---
// 设置主轴参数
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxVelHigh, 8000); // 运行速度 8000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxAcc, 10000);    // 加速度 10000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxDec, 10000);    // 减速度 10000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxJerk, 0);        // T 型曲线
// 设定从轴和主轴之间比率关系及相关参数
Ret  = Acm_AxGearInAx(m_Axhand[1], m_Axhand[0], m_Num, m_Den, RefSrc, Mode);
// 主轴执行点到点运动
Ret  = Acm_AxMoveRel(m_Axhand[0], 10000);                    // 相对点位运动
// 读取轴的状态和位置
F64    CmdPos;
F64    ActPos;
U16     State;
Acm_AxGetCmdPosition(m_Axhand[0], &CmdPos);                  // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axhand[0], &ActPos);                // 获取 0 轴的实际位置
Acm_AxGetState(m_Axhand[0], &State);                          // 获取 0 轴的状态
建议用户编写程序时, 检测函数返回值, 以判断函数的执行状态。并建立错误处理机制, 保证程序安全可靠运行。
具体使用步骤可参考 EGear 例程。

```

9.4.3 电子凸轮运动

9.4.3.1 电子凸轮运动相关函数与属性

电子凸轮运动相关函数如下表 9.30 所示，该表中的函数可在应用程序中直接调用。

表 9.30：电子凸轮运动相关函数

电子凸轮运动相关函数	说明
Acm_DevDownloadCAMTable	加载描述主轴和从轴关系的凸轮关系表
Acm_DevConfigCAMTable	数设置 CAM 表的相关参数
Acm_DevLoadCAMTableFile	加载编辑过的 Cam Table 文件
Acm_AxCamInAx	开始从轴和主轴之间的 CAM 同步
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveVel	命令轴按规定速度进行没有终点的运动
Acm_AxStopDec	命令轴按设定的减速度停止运行
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

电子凸轮运动相关属性如下表 9.31 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.31：电子凸轮运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置 / 获取速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxCamDOAssign	设置 / 获取 CamDO 输出的 DO 通道
CFG_AxCamDOEnable	启用 / 禁用 CamDO
CFG_AxCamDOLoLimit	设置 / 获取凸轮区间的低限位
CFG_AxCamDOHiLimit	设置 / 获取凸轮区间的高限位
CFG_AxCamDOPmpSrc	设置 / 获取凸轮区间的比较源
CFG_AxCamDOLogic	设置 / 获取 CamDO 的逻辑准位
CFG_AxModuleRange	设置当轴旋转 360° 时的脉冲个数
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.4.3.2 重点说明

凸轮机构主要作用是使从动杆按照工作要求完成各种复杂的运动，包括直线运动、摆动、等速运动和不等速运动。

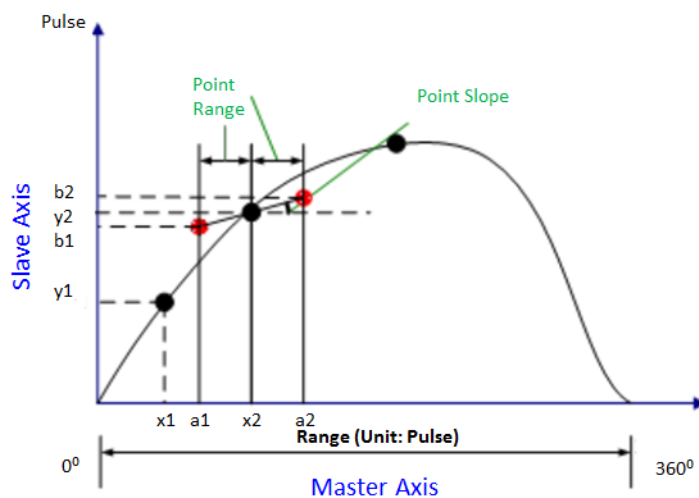
凸轮的主要特征是主轴和从轴之间的动态比例以及相位移动。主轴和从轴之间的关系通过 CAM 表描述，凸轮操作通过这张表执行操作。

研华运动控制卡提供如下两个函数用来加载凸轮关系表：

Acm_DevDownloadCAMTable: 该函数用于加载凸轮关系表，描述主轴和从轴之间的关系

Acm_DevLoadCAMTableFile: 加载编辑过的 Cam Table 文件，并通过 Utility 将其保存至设备

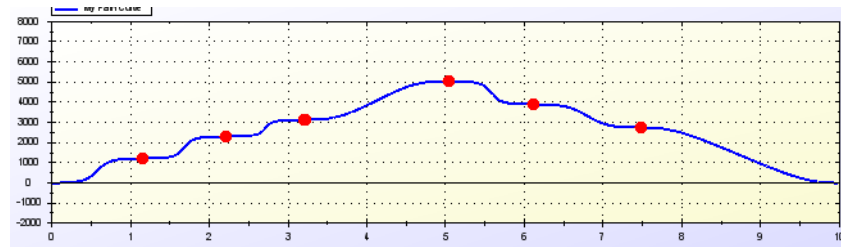
Acm_DevDownloadCAMTable 参数的含义如下：



如上图所示，当主轴旋转 360 度时，Range 为所需的脉冲个数。图上的黑色点组成 MasterArray 的主轴位置（如 X1 和 X2）和 SlaveArray 中的相应从轴位置。由黑色点以及指定的 PointRange 和 PointSlope 产生的红色点称为辅助参考点。CAM 曲线是通过由 MasterArray 和 SlaveArray 组成的 CAM 表拟合出来的。水平轴为当主轴旋转到某个角度时对应的脉冲数。垂直轴为当主轴旋转到某个角度时从轴对应的位置。

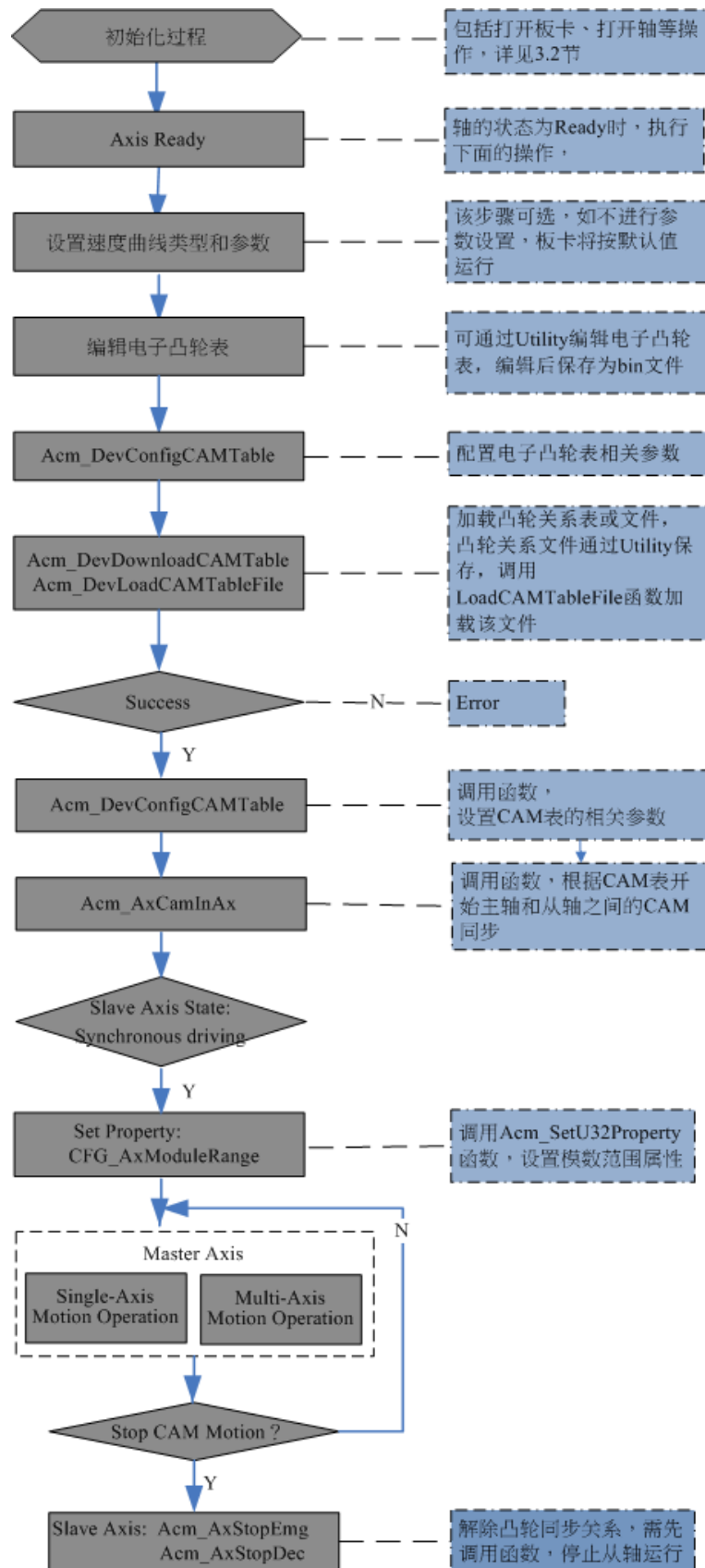
Range 必须通过属性 CFG_AxModuleRange 设置在主轴内。

电子凸轮表也可通过 Utility 编辑。如图所示为在 Utility 中通过增加 CAM 点，描绘出的电子凸轮曲线，将数据导出为 bin 文件，调用 Acm_DevLoadCAMTableFile 函数将该文件加载到项目中即可。



在实现电子凸轮过程中，通过 Acm_DevConfigCAMTable 函数设置 CAM 表的相关参数。

9.4.3.3 电子凸轮运动流程图



9.4.3.4 例程

例程实现 1 轴跟随 0 轴执行电子凸轮运动，通过 Utility 编辑电子凸轮表，设定的参数如下表所示。

功能	1 轴跟随 0 轴执行电子凸轮运动
主轴初速度	2000PPU/s
主轴运行速度	8000PPU/s
主轴加速度	10000PPU/s ²
主轴减速度	10000PPU/s ²
主轴速度曲线形式	T 型速度曲线
主轴起始位置	0
主轴目标位置	10000
执行 CAM 曲线方式	定期执行
CAM 曲线与主轴	CAM 曲线绝对对于主轴
CAM 曲线与从轴	CAM 曲线相对于从轴
主轴方向坐标偏移值	0
从轴方向坐标偏移值	0
主轴坐标 CAM 比例因子	1
从轴坐标 CAM 比例因子	1
CAM 表参考位置	理论位置

VC 代码如下：

```
HAND    m_Axhand[32];                // 轴的 handle
U32      Ret;                          // 函数返回值
char *   pstrString = new char[100]; // 保存 CamTable 文件路径的字符串
int      m_CurCamID;                  // 电子凸轮表 ID
U32      m_PointsCount = 0;           // 主轴在一个周期内需要的脉冲个数
U32      m_ModuleRange = 0;           // CamTable 中的点数量
Int      m_CurCamType = 1;            // 定期执行 CAM 曲线
U32      m_MasterAbsOrRel = 1;        // CAM 曲线绝对对于主轴
U32      m_SlaveAbsOrRel = 0;         // CAM 曲线相对于从轴
double   m_MasterOffset=0;            // 主轴方向的坐标偏移值
double   m_SlaveOffset=0;             // 从轴方向的坐标偏移值
double   m_MasterScale = 1.0;         // 主轴坐标中 CAM 的比例因子
double   m_SlaveScale = 1.0;          // 从轴坐标中 CAM 的比例因子
int      m_CmdOrFdb = 0;              // CAM 表的参考位置为理论位置
--- 初始化过程 ---
// 设置主轴参数
Ret = Acm_SetF64Property(m_Axhand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_Axhand[0], PAR_AxVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_Axhand[0], PAR_AxAcc, 10000);    // 加速度 10000
Ret = Acm_SetF64Property(m_Axhand[0], PAR_AxDec, 10000);    // 减速度 10000
Ret = Acm_SetF64Property(m_Axhand[0], PAR_AxJerk, 0);       // T 型曲线
// 加载通过 Utility 编辑过的 Cam Table 文件
Ret = Acm_DevLoadCAMTableFile(m_Devhand, pstrString, m_CurCamID,
&m_ModuleRange, &m_PointsCount);
// 设置 CAM 表相关参数
Ret = Acm_DevConfigCAMTable(m_Devhand, m_CurCamID, m_CurCamType,
```

```
m_MasterAbsOrRel, m_SlaveAbsOrRel);
//CAM 同步关系
Ret = Acm_AxCamInAx(m_Axhand[1],m_Axhand[0],m_MasterOffset,m_SlaveOffset,
m_MasterScale, m_SlaveScale, m_CurCamID, m_CmdOrFdb);
// 主轴执行点到点运动
Ret = Acm_AxMoveRel(m_Axhand[0],10000);          // 相对点位运动
// 读取轴的状态和位置
F64    CmdPos;
F64    ActPos;
U16    State;
Acm_AxGetCmdPosition(m_Axhand[0],&CmdPos);    // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axhand[0],&ActPos); // 获取 0 轴的实际位置
Acm_AxGetState(m_Axhand[0],&State);          // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建
立错误处理机制，保证程序安全可靠运行。
具体使用步骤可参考 ECAM 例程。
```

9.4.4 龙门运动

9.4.4.1 龙门运动相关函数与属性

龙门运动相关函数如下表 9.32 所示，该表中的函数可在应用程序中直接调用。

表 9.32：龙门运动相关函数	
龙门运动相关函数	说明
Acm_AxGantryInAx	命令两个轴进行龙门运动
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveVel	命令轴按规定速度进行没有终点的运动
Acm_AxStopDec	命令轴按设定的减速度停止运行
Acm_AxStopEmg	命令轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

龙门运动相关属性如下表 9.33 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.33：龙门运动相关属性	
参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置 / 获取速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxGantryMaxDiffValue	设置 / 获取龙门差值保护的值得
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.4.4.2 重点说明

龙门运动模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置运行。调用 Acm_AxGantryInAx 函数，设置参考源和从轴与主轴的方向，即可命令两个轴进行龙门运动。

龙门运动有如下几种限制：

- 不能给从轴设置任何运行命令
- 从轴不能添加到任何群组中
- 如果轴已经为群组中的一个轴，那么该轴不能作为 Gantry 的从轴。

如果复位主轴的理论 / 实际位置，则从轴的命令 / 实际位置也应复位至相同值。

龙门运动中提供龙门差值保护功能，当主从轴的反馈位置之间的最大差值大于设定值时，主从轴将停止脉冲输出。该功能通过设置 CFG_AxGantryMaxDiffValue 属性的值实现。

9. 4. 4. 3 龙门运动流程图

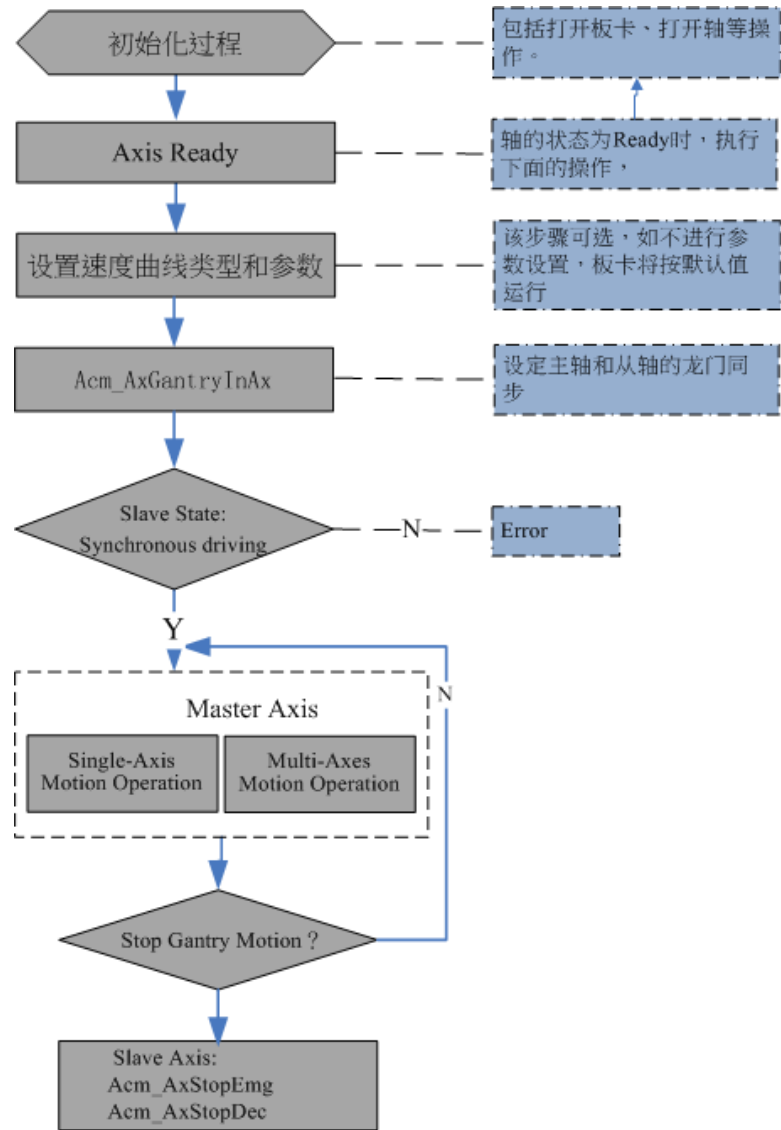


图 9.18： 龙门运动流程图

9. 4. 4. 4 例程

例程实现 0 轴为主轴，1 轴为从轴，从轴跟随主轴执行龙门运动。实现功能和相关参数如下表所示。

功能	1 轴跟随 0 轴执行龙门运动
主轴初速度	2000PPU/s
主轴运行速度	8000PPU/s
主轴加速度	10000PPU/s ²
主轴减速度	10000PPU/s ²
主轴速度曲线形式	T 型速度曲线
主轴起始位置	0
主轴目标位置	10000
从轴参考主轴位置	理论位置
从轴和主轴的方向	同向

VC 代码如下：

```

HAND    m_Axhand[32]; // 轴的 handle
U32     Ret;           // 函数返回值

```

```

I16    RefSrc=0; // 参考源为理论位置
I16    Dir=0;    // 和主轴方向相同
--- 初始化过程 ---
// 设置主轴参数
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxVelLow, 2000); // 初速度 2000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxVelHigh, 8000); // 运行速度8000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxAcc, 10000);   // 加速度 10000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxDec, 10000);   // 减速度 10000
Ret  = Acm_SetF64Property(m_Axhand[0], PAR_AxJerk, 0);       // T 型曲线
// 设定从轴和主轴的龙门同步关系
Ret  = Acm_AxGantryInAx(m_Axhand[1], m_Axhand[0], RefSrc, Dir);
// 主轴执行点到点运动
Ret  = Acm_AxMoveRel(m_Axhand[0], 10000);                   // 相对点位运动
// 读取轴的状态和位置
F64    CmdPos;
F64    ActPos;
U16    State;
Acm_AxGetCmdPosition(m_Axhand[0], &CmdPos);                 // 获取 0 轴的理论位置
Acm_AxGetActualPosition(m_Axhand[0], &ActPos);               // 获取 0 轴的实际位置
Acm_AxGetState(m_Axhand[0], &State);                         // 获取 0 轴的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立
错误处理机制，保证程序安全可靠运行。
具体使用步骤可参考 Gantry 例程。

```

9.4.5 切向跟随运动

9.4.5.1 切向跟随相关函数与属性

切向跟随相关函数如下表 9.34 所示，该表中的函数可在应用程序中直接调用。

表 9.34：切向跟随运动相关函数

电子齿轮运动相关函数	说明
Acm_AxTangentInGp	轴按照与群组路径切线相同的方向运动
Acm_GpMoveLinearRel	群组执行相对线性插补
Acm_GpMoveLinearAbs	群组执行绝对线性插补
Acm_GpMoveCircularRel	群组执行相对 ARC 插补
Acm_GpMoveCircularAbs	群组执行绝对 ARC 插补
Acm_GpMoveHelixRel	群组进行相对螺旋运动
Acm_GpMoveHelixRel	群组进行绝对螺旋运动
Acm_AxStopDec	轴按设定的减速度停止运行
Acm_AxStopEmg	轴立刻停止（无减速）
Acm_AxStopDecEx	下达停止命令时可指定减速度
Acm_AxGetState	获取轴的当前状态
Acm_AxResetError	复位轴的状态
Acm_AxSetCmdPosition	设置指定轴的理论（指令）位置
Acm_AxSetActualPosition	设置指定轴的实际（反馈）位置
Acm_AxGetCmdPosition	获取指定轴的当前理论（指令）位置
Acm_AxGetActualPosition	获取指定轴的当前实际（反馈）位置
Acm_AxGetCmdVelocity	获取当前轴的理论（指令）速度
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

切向跟随运动相关属性如下表 9.35 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

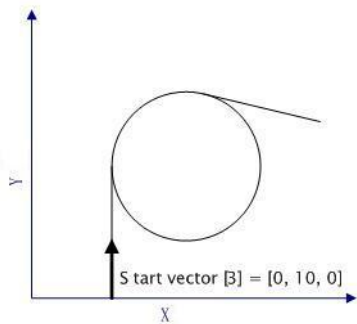
表 9.35：切向跟随运动相关属性

参数	说明
PAR_AxVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的运行速度
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置 / 获取速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

9.4.5.2 重点说明

在裁床类控制系统中，裁刀具有一定宽度，加工过程中要保证刃口和刀具裁割的运动方向保持一致，也就是曲线轮廓的切向方向，这种运动形式称为刀具切向跟随运动。切向跟随运动由轮廓轨迹运动与刀片的旋转运动相互配合构成。在运动控制过程中，实现裁剪头在沿 X、Y 平面曲线轮廓进行插补的同时，转角电机需控制裁剪刀的角度，

使刀片与裁片外轮廓曲线始终保持相切，需要控制 X、Y、Z 三个坐标轴联动，实现这类运动的三轴联动插补算法称为切向跟随运动控制算法。研华运动控制器可实现直线、圆弧、螺旋的切向跟随插补。



如图所示，群组将以顺时针方向做圆弧运动，箭头方向为其初始向量。
如果建立了同步，跟随轴将沿着圆弧的切向方向运动。

9.4.5.3 切向跟随运动流程图

切向跟随运动流程图如下图所示：

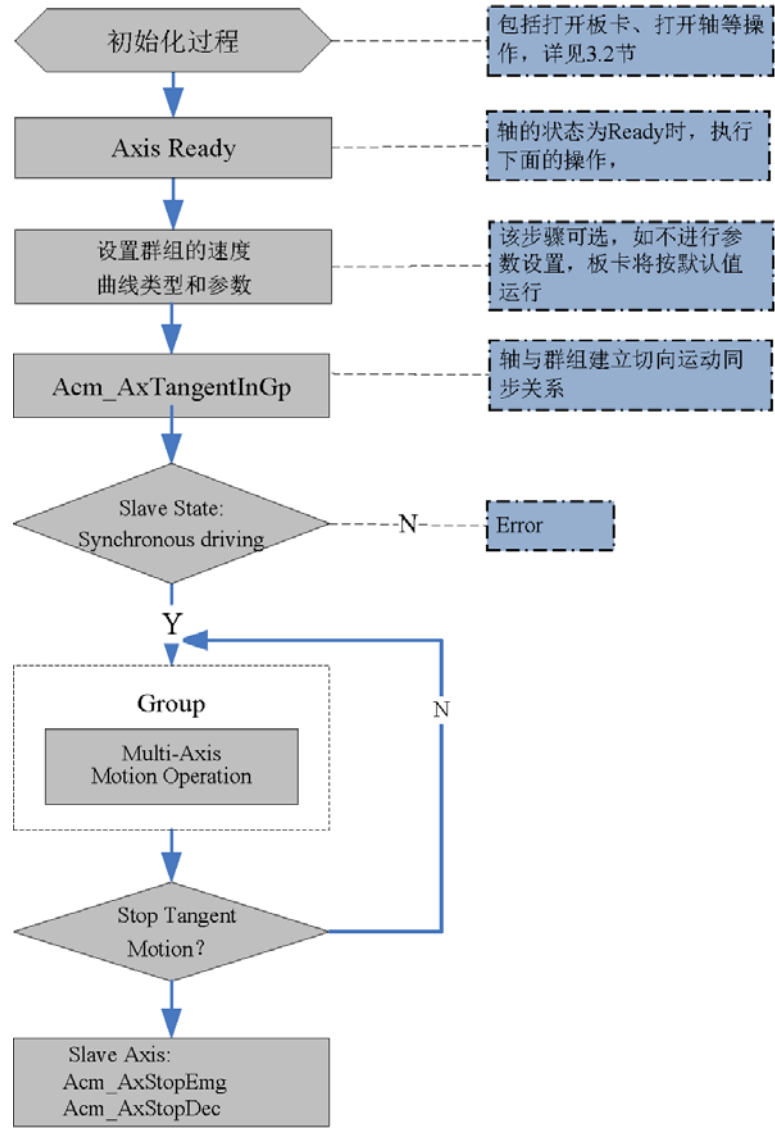


图 9.19：切向跟随运动流程图

9.4.5.4 例程

例程实现 0 轴和 1 轴执行圆弧插补，2 轴跟随 0、1 轴执行切向跟随。实现功能和相关参数如下表所示。如下表所示：

功能	1 轴跟随 0 轴执行圆弧插补
主轴初速度	2000PPU/s
主轴运行速度	8000PPU/s
主轴加速度	10000PPU/s ²
主轴减速度	10000PPU/s ²
主轴速度曲线形式	T 型速度曲线
主轴起始位置	0
主轴目标位置	10000
从轴参考主轴位置	理论位置
从轴和主轴的方向	同向

VC 代码如下：

```

HAND    m_GpHand;           // 组的 handle
U32     Ret;                 // 函数返回值
U32     m_GpReferencePlane;  // 参考平面
Double  CenterArray[3] = {8000, 8000};
Double  EndArray[3] = {16000, 16000};
U32     AxisNum;             // 添加到群组的轴数
short   startVector[3] = {0, 10, 0}; // 切向方向初始向量
byte    WorkingPlane=0;      // 参考平面为 XY 平面
int     m_Dir =0;            // 运行方向为同向
--- 初始化过程 ---
// 设置参数
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000); // 加速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpDec, 10000); // 减速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpJerk, 0); // T 型曲线
// 设置轴旋转 360 度时输出脉冲个数
Ret = Acm_SetU32Property(m_Axishand[2], CFG_AxModuleRange, 3600);
// 设置轴与群组的切向跟随关系
Ret = Acm_AxTangentInGp(m_Axishand[2], m_GpHand, startVector, WorkingPlane,
m_Dir);
// 执行圆弧插补运动
AxisNum =2;
Ret = Acm_GpMoveCircularRel(m_GpHand, CenterArray, EndArray, &AxisNum, 0);
// 读取组的状态
U16    GpState;
Acm_GpGetState(m_GpHand, &GpState); // 获取组的状态
建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建
立错误处理机制，保证程序安全可靠运行。
具体使用步骤可参考 Tangent 例程。

```

9.5 路径规划模式

9.5.1 本节简介

路径规划模式支持用户预先给定最多达 7000 段路径的路径表。支持用户为每段路径设置不同运动命令及速度。支持添加圆弧插补、直线插补、螺旋插补、延时命令和 D0 控制命令。支持速度前瞻功能，支持运动过程中暂停和恢复 Path 的执行功能。

9.5.2 路径规划

9.5.2.1 Path 运动相关函数与属性

Path 运动相关函数如下表 9.36 所示，该表中的函数可在应用程序中直接调用。

表 9.36: Path 运动相关函数	
Path 运动相关函数	说明
Acm_GpAddPath	将一个插补路径添加至系统路径缓存
Acm_GpResetPath	清空系统路径缓存
Acm_GpLoadPath	加载来自路径文件的路径数据
Acm_GpUnloadPath	卸载路径数据
Acm_GpMovePath	开始连续插补运动
Acm_GpGetPathStatus	获取路径缓存的当前状态
Acm_GpMoveSelPath	运行缓存中起始索引到终止索引范围内 Path
Acm_GpGetPathIndexStatus	获取系统路径缓存中指定索引路径的状态
Acm_GpPauseMotion	暂停群组运动命令
Acm_GpResumeMotion	恢复暂停的群组运动命令
Acm_GpGetState	获取群组的当前状态
Acm_GpGetCmdVel	获取群组的当前的速度值。
Acm_GpStopDec	命令该群组中的轴减速停止。
Acm_GpStopEmg	命令该群组中的轴立刻停止
Acm_SetU32Property	设置属性值（属性值为无符号 32 位整数）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整数）
Acm_SetF64Property	设置属性值（属性值为 Double 型）
Acm_GetU32Property	获取属性值（属性值为无符号 32 位整数）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整数）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

Path 运动相关属性如下表 9.37 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 9.37: Path 运动相关属性	
参数	说明
PAR_GpRefPlane	设置 / 获取螺旋运动和圆弧插补的参考平面
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID
配置	说明
CFG_GpAxesInGroup	获取哪个（哪些）轴在该群组中的信息

表 9.37: Path 运动相关属性

CFG_GpBldTime	设置 / 获取添加的路径与前一个路径的交接时间
CFG_GpSFEnable	启用 / 禁用速度前瞻功能

9.5.2.2 Path 加载方式及运动模式

研华运动控制卡提供如下两种加载 Path 的方式，如下说明：

方式一：调用 Acm_GpAddPath 函数将一个插补路径添加至系统路径缓存，可多次调用该函数，从而加载多段插补路径。

方式二：调用 Acm_GpLoadPath 函数加载来自路径文件的路径数据。一次可加载最多 600 个路径数据。

运动模式分类如表 9.38 所示：

表 9.38: Path 运动模式

运动模式	说明
直线插补	两轴直线插补（相对 / 绝对）
	三轴直线插补（相对 / 绝对）
直接线性插补	可选择板卡的部分轴 / 所有轴执行直接线性插补，与板卡轴数有关
圆弧插补	两轴圆弧插补（相对 / 绝对）
	三轴圆弧插补（相对 / 绝对）
螺旋插补	三轴及以上螺旋插补（相对 / 绝对），与板卡轴数有关
延迟命令	群组将延迟一段时间再执行下一路径
D0 开关功能	连续轨迹执行中控制 D0 开关而不影响整个连续轨迹的速度

将上表中的运动模式分为如表 9.39 所示运动命令，可在加载 Path 的两种方式中添加表中的运动命令。

表 9.39: Path 运动相关指令

运动模式	运动命令		说明
直线插补	Abs2DLine	Rel2DLine	2 轴绝对 / 相对直线插补
	Abs3DLine	Rel3DLine	3 轴绝对 / 相对直线插补
	Abs1DDirect	Rel1DDirect	绝对 / 相对点对点运动
直接线性插补	Abs2DDirect	Rel2DDirect	2 轴绝对 / 相对直接线性插补
	Abs3DDirect	Rel3DDirect	3 轴绝对 / 相对直接线性插补
	Abs4DDirect	Rel4DDirect	4 轴绝对 / 相对直接线性插补
	Abs5DDirect	Rel5DDirect	5 轴绝对 / 相对直接线性插补
	Abs6DDirect	Rel6DDirect	6 轴绝对 / 相对直接线性插补
	Abs7DDirect	Rel7DDirect	7 轴绝对 / 相对直接线性插补
	Abs8DDirect	Rel8DDirect	8 轴绝对 / 相对直接线性插补

表 9.39: Path 运动相关指令

2 轴圆弧插补	Abs2DArcCW	Rel2DArcCW	2 轴绝对 / 相对顺时针圆弧插补
	Abs2DArcCCW	Rel2DArcCCW	2 轴绝对 / 相对逆时针圆弧插补
	Abs2DArcCWAngle	Rel2DArcCWAngle	2 轴绝对 / 相对顺时针角度圆弧插补
	Abs2DArcCCWAngle	Rel2DArcCCWAngle	2 轴绝对 / 相对逆时针角度圆弧插补
	Abs2DArcCW_3P	Rel2DArcCW_3P	2 轴绝对 / 相对顺时针 3 点圆弧插补
	Abs2DArcCCW_3P	Abs2DArcCCW_3P	2 轴绝对 / 相对逆时针 3 点圆弧插补
3 轴圆弧插补	Abs3DArcCW	Rel3DArcCW	3 轴绝对 / 相对顺时针圆弧插补
	Abs3DArcCW	Rel3DArcCW	3 轴绝对 / 相对逆时针圆弧插补
	Abs3DArcCWAngle	Rel3DArcCWAngle	3 轴绝对 / 相对顺时针角度圆弧插补
	Abs3DArcCCWAngle	Rel3DArcCCWAngle	3 轴绝对 / 相对逆时针角度圆弧插补

表 9.39: Path 运动相关指令

螺旋插补	Abs3DSpiralCW	Rel3DSpiralCW	3 轴绝对 / 相对顺时针螺旋插补
	Abs3DSpiralCCW	Rel3DSpiralCCW	3 轴绝对 / 相对逆时针螺旋插补
	Abs4DSpiralCW	Rel4DSpiralCW	4 轴绝对 / 相对顺时针螺旋插补
	Abs4DSpiralCCW	Rel4DSpiralCCW	4 轴绝对 / 相对逆时针螺旋插补
	Abs5DSpiralCW	Rel5DSpiralCW	5 轴绝对 / 相对顺时针螺旋插补
	Abs5DSpiralCCW	Rel5DSpiralCCW	5 轴绝对 / 相对逆时针螺旋插补
	Abs6DSpiralCW	Rel6DSpiralCW	6 轴绝对 / 相对顺时针螺旋插补
	Abs6DSpiralCCW	Rel6DSpiralCCW	6 轴绝对 / 相对逆时针螺旋插补
	Abs7DSpiralCW	Rel7DSpiralCW	7 轴绝对 / 相对顺时针螺旋插补
	Abs7DSpiralCCW	Rel7DSpiralCCW	7 轴绝对 / 相对逆时针螺旋插补
	Abs8DSpiralCW	Rel8DSpiralCW	8 轴绝对 / 相对顺时针螺旋插补
	Abs8DSpiralCCW	Rel8DSpiralCCW	8 轴绝对 / 相对逆时针螺旋插补
	Abs3DSpiralCWAngle	Rel3DSpiralCWAngle	3 轴绝对 / 相对顺时针角度螺旋插补
	Abs3DSpiralCCWAngle	Rel3DSpiralCCWAngle	3 轴绝对 / 相对逆时针角度螺旋插补
	Abs4DSpiralCWAngle	Rel4DSpiralCWAngle	4 轴绝对 / 相对顺时针角度螺旋插补
	Abs4DSpiralCCWAngle	Rel4DSpiralCCWAngle	4 轴绝对 / 相对逆时针角度螺旋插补
	Abs5DSpiralCWAngle	Rel5DSpiralCWAngle	5 轴绝对 / 相对顺时针角度螺旋插补
	Abs5DSpiralCCWAngle	Rel5DSpiralCCWAngle	5 轴绝对 / 相对逆时针角度螺旋插补
	Abs6DSpiralCWAngle	Rel6DSpiralCWAngle	6 轴绝对 / 相对顺时针角度螺旋插补
	Abs6DSpiralCCWAngle	Rel6DSpiralCCWAngle	6 轴绝对 / 相对逆时针角度螺旋插补
	Abs7DSpiralCWAngle	Rel7DSpiralCWAngle	7 轴绝对 / 相对顺时针角度螺旋插补
	Abs7DSpiralCCWAngle	Rel7DSpiralCCWAngle	7 轴绝对 / 相对逆时针角度螺旋插补
	Abs8DSpiralCWAngle	Rel8DSpiralCWAngle	8 轴绝对 / 相对顺时针角度螺旋插补
	Abs8DSpiralCCWAngle	Rel8DSpiralCCWAngle	8 轴绝对 / 相对逆时针角度螺旋插补
延迟命令	GPDELAY		延时一段时间 Path 的执行 (单位 ms)
D0 开关	DOControl		连续轨迹执行中控制 D0 开关, 关于其说明见 9.5.5 节
End Path	EndPath		所有 Path 添加完成后, 添加该运动命令

注!

如上表所示运动命令可分为绝对和相对命令, 绝对和相对命令不能再系统路径缓存中混合添加, 除 EndPath 和 GPDELAY 外, 否则将返回错误代码。

9.5.2.3 以 Add Path 方式加载路径

路径加载方式一，通过调用 Acm_GpAddPath 函数将一个插补路径添加至系统路径缓存。调用该函数之前，用户可通过 CFG_GpSFEnable 启用 / 禁用速度前瞻功能，通过 CFG_GpBldTime 设置速度交接时间。该函数的原型如下所示：

U32 Acm_GpAddPath (HAND GroupHandle, U16 MoveCmd, U16 MoveMode, F64 FH, F64 FL, PF64 EndPoint_DataArray, PF64 CenPoint_DataArray, PU32 ArrayElements)

关于该函数参数的使用说明如下表：

参数	说明
GroupHandle	代表组的句柄。系统缓存中每个路径的群组句柄必须相同。如果系统缓存中存在一些未执行的路径，且用户想要通过调用 Acm_GpAddPath 添加新的路径，参数 GroupHandle 必须和之前未执行的路径群组句柄相同。
MoveCmd	代表运动命令，即表 9.39 中所列出的运动命令。当添加的运动命令为延迟命令时，用户可通过 FH 设置延时时间，并且当启用速度前瞻功能时，将不能添加延时命令，延时效应的单位为 ms。关于 D0 开关运动命令的说明详见 9.5.5 节。
MoveMode	速度交接模式和非交接模式。关于速度交接模式和非交接模式详见 9.5.4 章节。
FH, FL	添加的插补 Path 的运行速度和起始速度。支持用户为每段路径设置不同运行速度和起始速度。当添加的运动命令为延迟命令时，FH 表示延时时间。
EndPoint_DataArray	终点坐标。（单位：各轴 PPU） 1. 若运动命令为绝对 / 相对 2 轴顺 / 逆时针圆弧插补 "(Abs/Rel)2DArc (CW/CCW)"，则参数为旋转角度。（单位：度） 2. 若运动命令为绝对 / 相对 3 轴顺 / 逆时针角度圆弧插补 "(Abs/Rel)3DArc (CW/CCW)Angle"，则参数 EndPoint_DataArray[0~2] 为法向量，EndPoint_DataArray[3] 为旋转角度。（单位：度） 3. 若运动命令为绝对 / 相对 3~8 轴顺 / 逆时针角度螺旋插补 "(Abs/Rel) (3~8)DSpiral (CW/CCW)Angle"，旋转角度必须输入基准面定义的 EndPoint_DataArray[0~2] 中，而基准面可通过 Par_GpRefPlane 设置，其他参数为终点。请参考章节 9.3.4.2 中的方法 2，其中，基准面为 X-Y 平面；EndPoint_DataArray[0] 和 EndPoint_DataArray[1] 为旋转角度，值为 1080；EndPoint_DataArray[2] 值为 10000，是 Z 轴终点。
CenPoint_DataArray	中心点坐标（单位：各轴 PPU）
ArrayElements	该参数为 EndPoint_DataArray、CenPoint_DataArray 的元素个数，且不能小于群组中的轴数，即元素个数不能小于群组的轴数。根据运动命令，所需的轴个数分别说明如下： 1) 当运动命令所需轴个数小于等于群组的轴个数时，所有路径可加载至相同的系统缓存，如群组轴个数为 4，则 Rel2DLine、Rel3DLine 和 4DDirect 等路径可加载至设备 2) 当群组中的轴个数大于路径中所需轴个数时，将选择群组前面的轴实现运动。当选择的运动命令为 2 轴圆弧插补时，可通过 Par_GpRefPlane 选择群组中前面三个轴中的两个轴实现圆弧运动。

9.5.2.4 以 Load Path 方式加载路径

调用 Acm_GpLoadPath 函数加载来自路径文件的路径数据。一次可加载最多 600 个路径数据。该函数的函数原型如下所示：

U32 Acm_GpLoadPath (Hand GroupHandle, PI8 FilePath, PHAND PathHandle, PU32 pTotalCount)

该函数的第二个参数 FilePath 指向需要加载的运动路径文件。路径数据文件（二进制）可通过 Utility 进行编写，编写并保存后调用上述 API 下载到应用程序中，即可执行编写好的 Path。当不再使用 PathHandle 或应用关闭时，必须通过 Acm_GpUnloadPath 卸载 PathHandle，同时 PathHandle 中包含的路径将从驱动中删除。

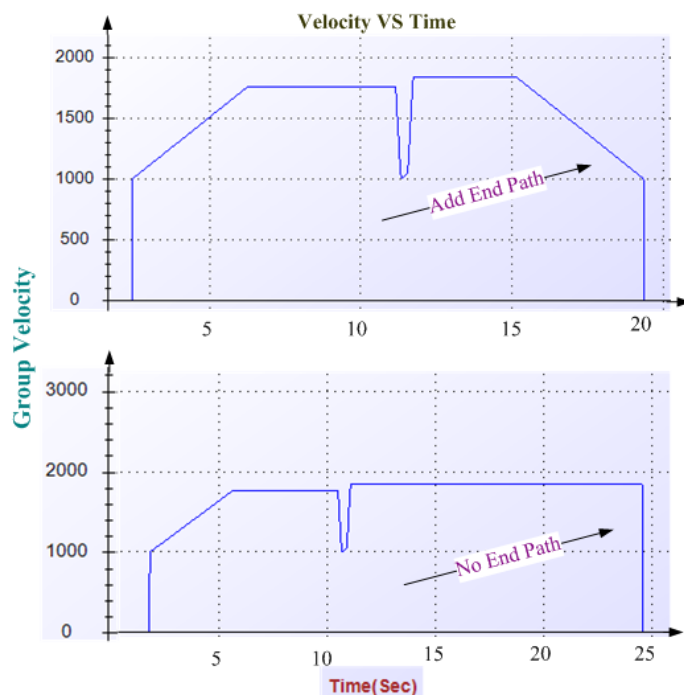
9.5.3 EndPath 的作用及影响

调用 9.5.2.2 节介绍的两种 Path 加载方式时，最后一段 Path 请加入以 EndPath 为命令的 Path 至路径缓存中，否则会影响最后段 Path 的减速度。

调用 API 运行如下表编辑的 Path。加速度、减速度均为 500，启用速度交接模式，设置速度交接时间为 200ms：

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	EndPoint0	EndPoint1
0	Rel2DLine	Enable Blending	2000	1000	0	0	10000	10000
1	Rel2DArcCW	Enable Blending	2000	1000	4000	0	8000	0
2	EndPath	Enable Blending	1000	400	0	0	0	0

运行添加 EndPath 和没有添加 EndPath 的速度曲线如下图所示：



添加 EndPath 将能够重算最后一段运动 Path 的运动速度，当不添加 EndPath 时，将不重现最后段 Path 的速度，可能会出现断尾现象。

9.5.4 速度交接模式

速度交接模式指运行 Path 时，各段 Path 之间速度的交接方式。在加载 Path 时，可通过速度交接模式 (MoveMode)、速度前瞻 (CFG_GpSFEnable) 及速度交接时间 (CFG_GpBldTime) 之间的组合来达到各线段间不同的速度交接方式，其关系如下表所示：

表 9.40：速度交接模式关系表

	Buffer mode	Blending mode	Fly mode
速度前瞻 (CFG_GpSFEnable)	禁用速度前瞻，否则提示错误，不能运行 Path	禁用速度前瞻，否则提示错误，不能运行 Path	启用速度前瞻，将起作用（仅限 T 型速度曲线）
速度交接时间 (CFG_GpBldTime)	不能设定速度交接时间，否则提示错误，不能运行 Path	设定速度交接时间需大于 0	设定速度交接时间需大于 0
速度交接 (MoveMode)	禁用速度交接 (MoveMode =1)	启用速度交接 (MoveMode =0)	启用速度交接 (MoveMode =0)

根据上表，在启用速度交接功能时，Path 有两种速度交接方式：Blending Mode 和 FlyMode，当设置的速度交接时间大于零时，以 Blending Mode 运行；当设置的速度交接时间等于零时，以 FlyMode 运行，且当速度曲线类型为 T 型时，可启用速度交接功能。

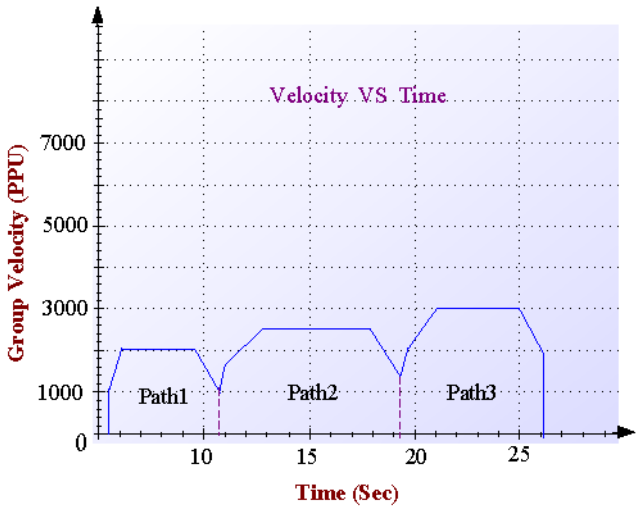
9.5.4.1 BufferMode

MoveMode	CFG_GpSFEnable	CFG_GpBldTime
1	无效	无效

当禁用速度交接模式时，添加的 Path 将以 BufferMode 运行，该种运行模式下，每段路径都包括加速和减速的过程；速度前瞻功能不支持，故需设置禁用速度前瞻。
编辑如下 Path(可通过 Utility 编辑)，设置加速度、减速度均为 2000。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	Center2	EndPoint0	EndPoint1	EndPoint2
0	Rel2DLine	Disable Blending	2000	1000	0	0	0	12000	12000	0
1	Rel3DHelix CW	Disable Blending	2500	1500	10000	0	0	20000	0	10000
2	Rel2DArcCW	Disable Blending	3000	2000	10000	0	0	20000	0	0
3	EndPath	Disable Blending	1000	500	0	0	0	0	0	0

调用 API 运行后速度曲线如下图所示：



9.5.4.2 Blending Mode

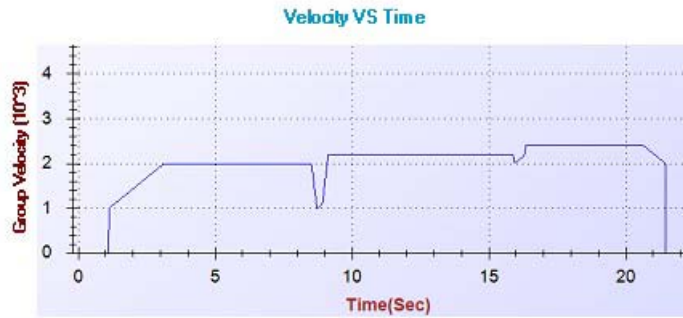
MoveMode	CFG_GpSFEnable	CFG_GpBldTime
0	Disable	大于 0

当启用速度交接模式，且设置速度交接时间大于零时，加载的 Path 将以 Blending Mode 运行。该种运行方式下，每段路径都包括加速和减速的过程，每段 Path 之间将根据速度交接时间重新计算起始速度和减速度。
编辑如下 Path (可通过 Utility 编辑)，设置速度交接时间 400ms，加速度、减速度分别为 500、500。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	Center2	EndPoint0	EndPoint1	EndPoint2
0	Rel2DLine	Enable Blending	2000	1000	0	0	0	10000	10000	0
1	Rel3DHelixCW	Enable Blending	2500	1500	5000	0	0	10000	0	10000

2	Rel2DArcCW	Enable Blending	3000	2000	4000	0	0	8000	0	0
3	EndPath	Enable Blending	1000	500	0	0	0	0	0	0

运行后的速度曲线如下图所示：



9.5.4.3 Fly Mode

MoveMode	CFG_GpSFEnable	CFG_GpBldTime
0	Disable	等于 0

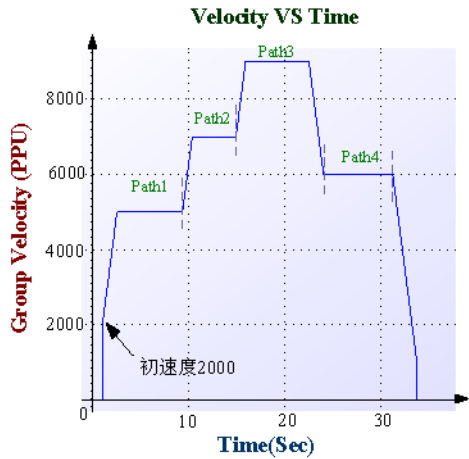
当启用速度交接模式，且设置速度交接时间等于零时，加载的 Path 将以 FlyMode 运行。该种模式下可开启速度前瞻功能。

1. 禁用速度前瞻功能

编辑如下Path（可通过Utility编辑），加速度和减速度均为2000，速度交接时间为零。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center 0	Center 1	Center 2	EndPoint 0	EndPoint 1	EndPoint 2
0	Rel2DLine	Enable Blending	5000	2000	0	0	0	30000	25000	0
1	Rel2DLine	Enable Blending	7000	4000	0	0	0	25000	28000	0
2	Rel2DArcCW	Enable Blending	9000	3000	10000	10000	0	20000	0	0
3	Rel2DArcCWAngel	Enable Blending	6000	1000	20000	0	0	180	180	0
4	EndPath	Enable Blending	2000	500	0	0	0	0	0	0

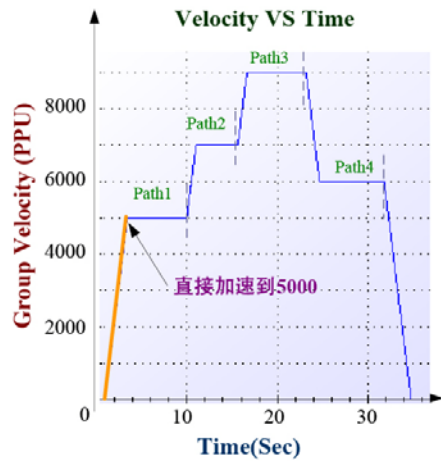
保存后调用 Acm_GpLoadPath 函数加载该 Path 文件，调用 Acm_GpMovePath 函数运行。如下图运行速度、初速度为表中设置的值



将表中的初速度修改如下：

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center 0	Center 1	Center 2	EndPoint 0	EndPoint 1	EndPoint 2
0	Rel2DLine	Enable Blending	5000	0	0	0	0	30000	25000	0
1	Rel2DLine	Enable Blending	7000	0	0	0	0	25000	28000	0
2	Rel2DArcCW	Enable Blending	9000	0	10000	10000	0	20000	0	0
3	Rel2DArcCWAngel	Enable Blending	6000	0	20000	0	0	180	180	0
4	EndPath	Enable Blending	2000	500	0	0	0	0	0	0

速度曲线如下图所示，



由图可知，除第一段 Path 不同外，其他段速度曲线相同。

2. 启用速度前瞻功能

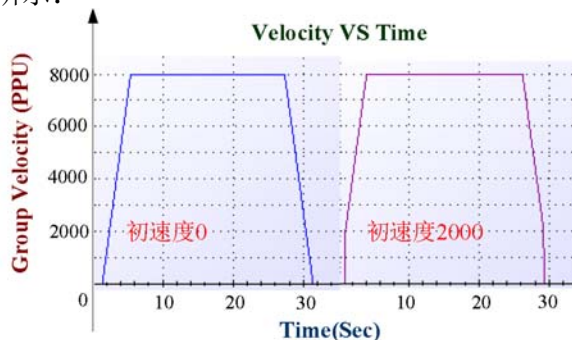
速度前瞻指根据移动距离，计算将要执行 Path 的运行速度。

当启用速度前瞻功能运行 Path 时，将不使用 AcmeGpAddPath 中设置的速度参数，仅使用群组的速度参数值（通过属性设置）运行 Path。

编辑 Path 如下（可通过 Utility 编辑），启用速度前瞻功能，设置 PAR_GpVelHigh（群组运行速度）、PAR_GpAcc（群组加速度）、PAR_GpDec（群组减速度）属性值分别为 8000、2000、2000，PAR_GpVelLow（群组初速度）分别为 0 和 2000。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center 0	Center 1	Center 2	EndPoint 0	EndPoint 1	EndPoint 2
0	Rel2DLine	Disable Blending	2000	1000	0	0	0	12000	12000	0
1	Rel3DHelixCW	Disable Blending	2500	1500	10000	0	0	20000	0	10000
2	Rel2DArcCW	Disable Blending	3000	2000	10000	0	0	20000	0	0
3	EndPath	Disable Blending	1000	500	0	0	0	0	0	0

速度曲线如下图所示：



9.5.5 Path D0 功能指令

9.5.5.1 Path D0 功能说明

在实际案例中，需要在 Path 中使用 D0 功能。以点胶机为例，当在特定路径上需要点胶时，用户不需要检测是否到达需要点胶的路径，只需在需要点胶的路径上设置 D0 输出即可完成点胶过程。

在 PCI-1203 上，用户可以指定网络上的 D0 通道将这些 D0 的开关控制添加到 Path Buffer 中，以便在连续轨迹的执行中控制某些 D0 的开关而不会影响到整个连续轨迹的速度。

9.5.5.2 Path D0 实现方式

其实现方法同样是通过 API：Acm_GpAddPath 进行设定。重要参数使用方式如下：

名称	描述
GroupHandle	群组句柄。
MoveCmd	运动命令：DoControl
MoveMode	运动模式
FH	网络上的 D0 端口 ID，端口 ID 可以从 utility 上得知。
FL	请将此参数置 0。
EndPoint_DataArray	群组中各轴的 D0 设定，仅 EndPoint_DataArray[0] 有效。
CenPoint_DataArray	请将此参数置为 Null。
ArrayElements	群组中的轴数。

其中 FH 以及 EndPoint_DataArray 中各元素 D0 设定方式如下表：

Bit	24~31	16~23	8~15	0~7
网络上的 D0 ID 设定，通过 FH 参数传入。	保留	设定端口中的 OUT0~OUT7 是否在 Path 中启用 D0 的开关控制功能。 0: 禁用 1: 启用	保留	打开 / 关闭指定端口的 OUT0~OUT7 0: 关闭 1: 打开

9.5.6 Path 的执行

通过 9.5.2.2 节描述的两种方式加载 Path 成功后，即可调用 Acm_GpMovePath 函数开始连续插补运动。调用 Acm_GpGetPathStatus 函数获取路径缓存的当前状态。

加载 Path 成功后，也可调用 Acm_GpMoveSelPath 函数执行路径缓存中指定的起始索引到终止索引范围内的路径。

调用 Acm_GpGetPathIndexStatus 函数获取系统路径缓存中指定索引路径的状态。调用 Acm_GpResetPath 函数可清空系统路径缓存，若此时有群组正在执行路径，则路径运动将停止。关于每种函数的使用详见附件。

操作 PCI-1203 时，用户可通过 `Acm_GpMoveAllPath` 同时启动多群组路径运动。载入每个群组的路径之后，调用该函数将同时启动指定群组的路径。

注！ 由于路径已经通过 `Acm_GpLoadPath` 载入系统，作为 `Acm_GpMovePath` API 的第二参数，`PathHandle` 可能会被设置为 `NULL`，否则路径将被载入两次。



9.5.7 Path 的暂停和恢复

当 Group 在执行 Path 运动过程中，可下达暂停命令，板卡收到命令后，会减速停止下来。当对其再做恢复命令后，则继续执行暂停之前的尚未完成的部分。

暂停和恢复调用的 API 如下：

功能函数	说明
<code>Acm_GpPauseMotion</code>	暂停群组运动命令
<code>Acm_GpResumeMotion</code>	恢复暂停的群组运动命令

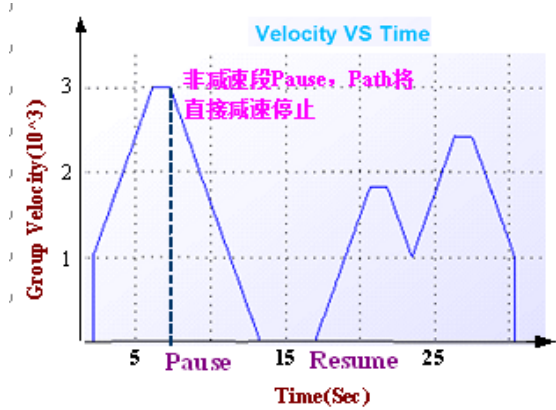
9.5.7.1 暂停 / 恢复在 BufferMode 中的使用

在 BufferMode 中，每段都有自己的加减速过程，当收到暂停命令后，当前正在执行的某段会减速停止，当恢复时，板卡则会计算停止的那段剩余的 Pulse 所能够支持的速度，然后继续执行剩余的路段。

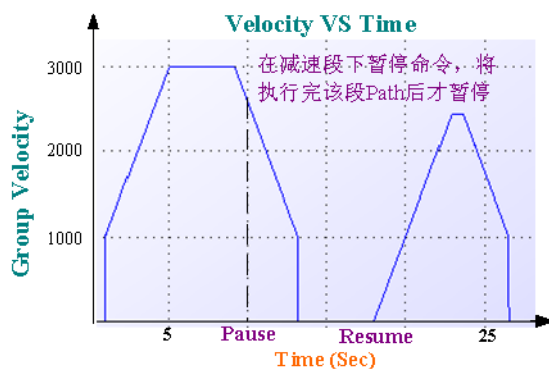
编辑 Path 如下，设置加速度、减速度均为 500。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	EndPoint0	EndPoint1
0	Rel2DLine	Disable Blending	3000	1000	0	0	20000	20000
1	Rel2DArcCW	Disable Blending	3000	1000	4000	0	8000	0
2	EndPath	Disable Blending	1000	400	0	0	0	0

速度曲线如下图所示：



如果在减速段下暂停命令，则暂停命令在该段 Path 运行完成后才会起作用。如下运行上表中的 Path，速度曲线如下所示：



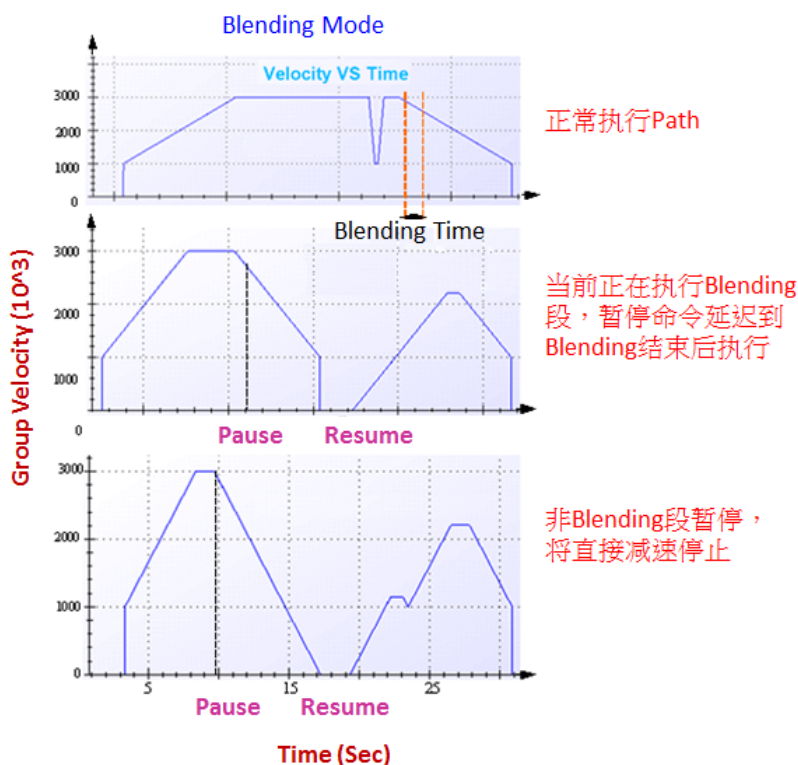
9.5.7.2 暂停 / 恢复在 BlendingMode 中的使用

在 Blending 模式下，如果暂停命令下达时，当前正在执行 Blending 段，则暂停命令延迟到 Blending 结束后执行。

编辑 Path 如下，设置速度交接时间为 500ms，加速度、减速度均为 500。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	EndPoint0	EndPoint1
0	Rel2DLine	Enable Blending	3000	1000	0	0	20000	20000
1	Rel2DArcCW	Enable Blending	3000	1000	4000	0	8000	0
2	EndPath	Enable Blending	1000	400	0	0	0	0

速度曲线如下图所示：



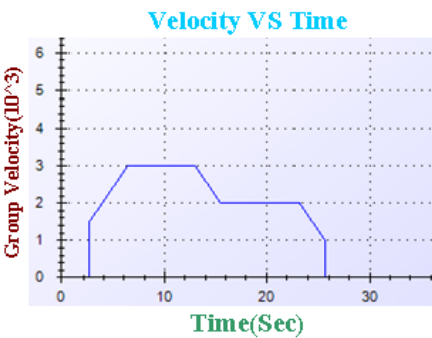
9.5.7.3 暂停 / 恢复在 FlyMode 中的使用

1. 禁用速度前瞻功能

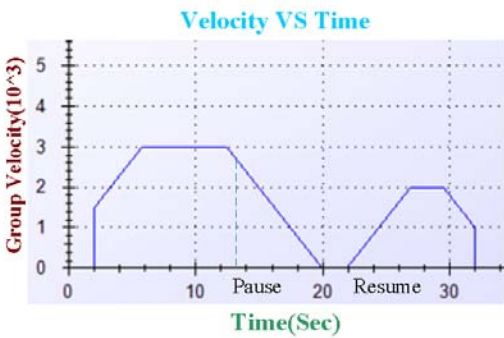
当禁用速度前瞻功能时，设置速度交接时间为 0，编辑 Path 如下表所示，加速度、减速度均为 500。

ID	MoveCmd	MoveMode	VelHigh	VelLow	Center0	Center1	EndPoint0	EndPoint1
0	Rel2DLine	Enable Blending	3000	1500	0	0	20000	20000
1	Rel2DArcCW	Enable Blending	2000	1000	8000	0	16000	0
2	EndPath	Enable Blending	1000	400	0	0	0	0

正常运行速度曲线如下图所示：

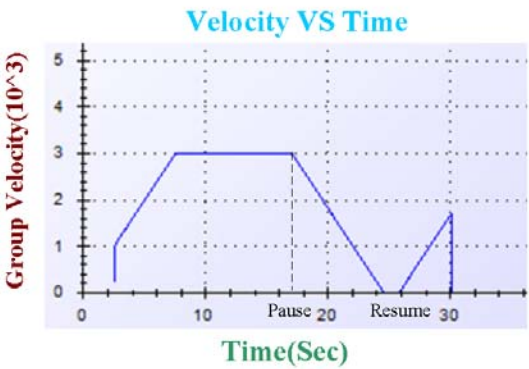


执行暂停和恢复功能的速度曲线如下图所示：



2. 启用速度前瞻功能

当启用速度前瞻功能时，Path 的速度为群组速度，设置加速度、减速度均为 500，运行速度 3000，初速度 1000，速度曲线如下图所示：

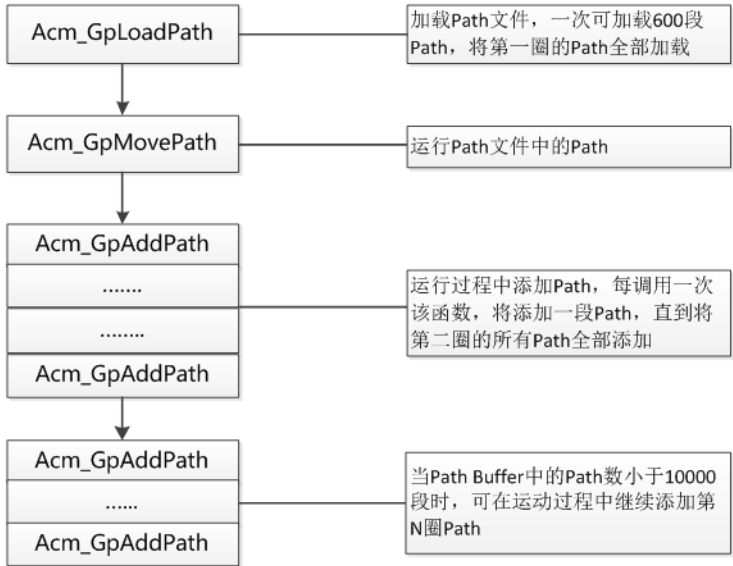


9.5.8 动态加载 Path

在实际应用中，需要在板卡运动过程中，动态添加多段 Path。例如玻璃磨边过程中，共有多圈轨迹：第一圈、第二圈 …，每圈由多段 Path 组成。

当添加完第一圈多段 Path 并执行过程中，将第二圈的多段 Path 加载到 Buffer 中。为了实现上述过程，可通过两种方法实现：

方法一：动态加载 Path



如上图所示，动态加载 Path 时，首先可通过 LoadPath 函数加载 Path 文件，Path 文件中的 Path 数不超过 600 段，运动 Path 过程中，当路径缓存中 Path 数小于 7000 段时，继续添加 Path，路径缓存中 Path 状态可调用函数 **Acm_GpGetPathStatus** 获取。

方法二：动态加载 Path



如上图所示，动态加载 Path 时，首先可通过 AddPath 函数加载多段 Path。运动 Path 过程中，当路径缓存中 Path 数小于 7000 段时，可继续添加 Path，路径缓存中 Path 状态可调用函数 **Acm_GpGetPathStatus** 获取。

以上两种加载 Path 的方法中，动态加载 Path 只能通过 **Acm_GpAddPath** 函数实现。Path 执行过程中，将不断释放执行过的 Path 缓存，例如加载了 7000 段 Path，执行了 10 段 Path，将释放掉 10 段 Path 的缓冲。此时可继续 AddPath。

9.5.9 Path 运动流程图

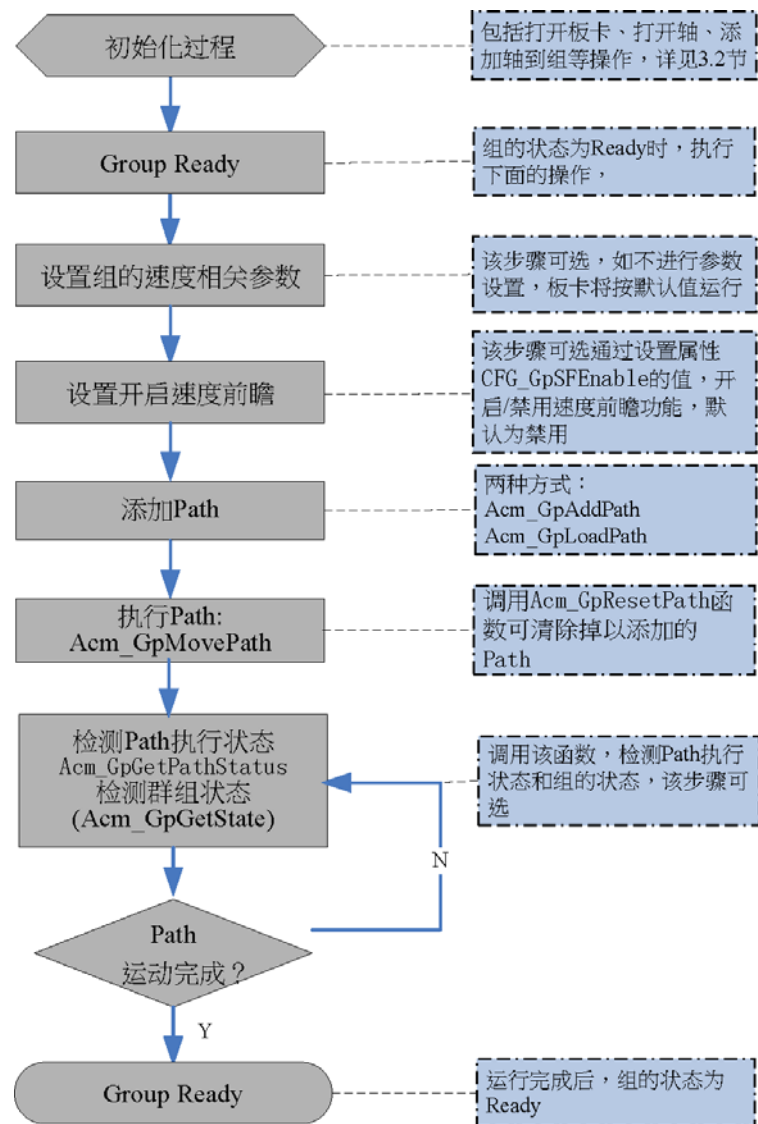


图 9.20: Path 运动流程图

9.5.10 Path 运动例程

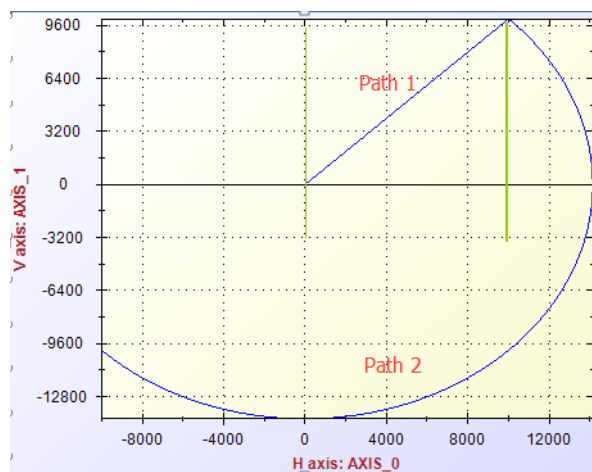
该例程实现 PCI-1203 执行 Path 运动，使用速度前瞻，添加的 Path 如下表所示：

参数	Path1	Path2	Path3
MoveCmd	Rel2DLine	Rel2DArcCW	EndPath
MoveMode	0	0	0
FH	8000	8000	8000
FL	2000	2000	2000
EndPoint_DataArray	(-10000, -20000, 0, 0)	(-20000, -20000, 0, 0)	NULL
CenPoint_DataArray	NULL	(-10000, -10000, 0, 0)	NULL
ArrayElements	4	4	4

实现功能及速度参数如下表所示：

功能	执行连续插补运动 (PCI-1203 运动控制卡)
初速度	2000PPU/s
运行速度	8000PPU/s
加速度	10000PPU/s ²
减速度	10000PPU/s ²
速度曲线形式	T 型速度曲线
速度前瞻	启用

上述功能执行后的 Path 曲线如下图所示：



VC 代码如下：

```

HAND    m_GpHand;          // 组的 handle
U32     Ret;                // 函数返回值
U16     MoveMode = 0;       //Eable blending 支持速度前瞻模式
U16     MoveCmd;
F64     FH = 80000;
F64     FL = 2000;
U32     ArrayAxCnt= 4;      // 元素个数
F64     CenPosArray[4];     // 圆心
F64     EndPosArray[4];     // 终点
--- 初始化过程 ---
// 设置参数
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelLow, 2000); // 初速度 2000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpVelHigh, 8000); // 运行速度 8000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpAcc, 10000);   // 加速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpDec, 10000);   // 减速度 10000
Ret = Acm_SetF64Property(m_GpHand, PAR_GpJerk, 0);      //T 型曲线
// 添加相对直线运动
EndPosArray[0] = -10000;
EndPosArray[1] = -20000;
MoveCmd =RelMoveLine;
Ret = Acm_GpAddPath(m_GpHand, MoveCmd, MoveMode, FH, FL, EndPosArray, NULL,
&ArrayAxCnt);
// 添加相对圆弧运动

```

```
CenPosArray[0] = -10000;
CenPosArray[1] = -10000;
EndPosArray[0] = -20000;
EndPosArray[1] = -20000;
MoveCmd = RelMoveArcCW;
// 添加相对圆弧运动
Ret = Acm_GpAddPath(m_GpHand, MoveCmd, MoveMode, FH, FL, EndPosArray,
CenPosArray, &ArrayElements);
MoveMode=0;
// 添加 EndPath
Ret = Acm_GpAddPath(m_GpHand, MoveCmd, MoveMode, FH, FL, NULL, NULL,
&ArrayElement);
// 执行 Path
Ret = Acm_GpMovePath(m_GpHand, NULL);
// 获取 Path 执行状态
U32Index, FreeCnt, Remain, CurCmd;
Ret = Acm_GpGetPathStatus(m_GpHand, &Index, &CurCmd, , &Remain, &FreeCnt);
// 读取组的状态
U16GpState;
Acm_GpGetState(m_GpHand, &GpState); // 获取组的状态
```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Path 例程。

第 10 章

DI/O 与 AI/O 控制

10.1 简介

本章主要介绍 I/O 的相关操作，以及板卡的其他重要功能。

10.2 DI/O 与 AI/O 控制

10.2.1 本节简介

PCI-1203 支持 ADAM-5000/ECAT EtherCAT 从站。

本章内容介绍编程工具通用运动 API 的定义，以及用户如何使用这些 API 来操作 ADAM-5000/ECAT 高速系统。

10.2.2 IO 控制相关函数与属性

IO 控制函数如表 10.1 所示，其中的函数可在程序中直接调用。

表 10.1: IO 控制函数

函数	说明
Acm_GetAvailableDevs	获取可用设备数和设备名列表
Acm_DevOpen	打开指定设备以获取设备句柄
Acm_DevClose	关闭设备
Acm_LoadMapFile	在 EtherCAT 网络中载入 I/O 映射文件
Acm_DevReOpen	重新打开设备
Acm_DevEnableEvent	启用事件检测功能
Acm_DevCheckEvent	检测事件
Acm_DevGetSlaveInfo	根据设备上的旋转开关获取从轴信息
Acm_DaqDiGetBit	获取指定 DI 通道的位数据
Acm_DaqDiGetByte	获取指定 DI 端口的字节数据
Acm_DaqDiGetBytes	获取连续 DI 端口的字节数据
Acm_DaqDoSetBit	设定指定 DO 通道的位数据
Acm_DaqDoSetByte	设定指定 DO 端口的字节数据
Acm_DaqDoSetBytes	设定连续 DO 端口的字节数据
Acm_DaqDoGetBit	获取指定 DO 通道的位数据
Acm_DaqDoGetByte	获取指定 DO 端口的字节数据
Acm_DaqDoGetBytes	获取连续 DO 端口的字节数据
Acm_DaqAiGetRawData	获取模拟量输入通道的二进制值
Acm_DaqAiGetVoltData	进行电压输入时，获取模拟量输入通道的电压值
Acm_DaqAiGetCurrData	进行电流输入时，获取模拟量输入通道的电流值
Acm_DaqAoSetRawData	设定模拟量输出通道的二进制值
Acm_DaqAoSetVoltData	将指定模拟量输出通道的电压输出值设定在模拟量电压输出范围之内
Acm_DaqAoSetCurrData	将指定模拟量输出通道的电流输出值设定在模拟量电流输出范围之内
Acm_DaqAoGetRawData	获取指定模拟量输出通道的二进制输出值
Acm_DaqAoGetVoltData	获取指定模拟量输出通道的电压输出值，该值在模拟量电压输出范围之内
Acm_DaqAoGetCurrData	获取指定模拟量输出通道的电流输出值，该值在模拟量电流输出范围之内
Acm_DaqCntGetCounterData	获取指定 Counter 通道的计数器数据
Acm_DaqCntResetCnt	重置指定 Counter 通道的计数器数据
Acm_DaqCntGetLatchData	获取指定 Counter 通道的锁存数据

表 10.1: IO 控制函数

Acm_DaqCntResetLatch	重置指定 Counter 通道的锁存数据
Acm_DaqCntGetCmpData	获取指定 Counter 通道的比较数据
Acm_DaqCntResetCmp	重置指定 Counter 通道的比较数据
Acm_DaqCntTriggerCmp	触发指定 Counter 通道的命令以锁存位置数据
Acm_GetProperty	通过分配的 PropertyID 获取属性值
Acm_GetU32Property	获取无符号 32 位整型属性值
Acm_GetI32Property	获取带符号 32 位整型属性值
Acm_GetF64Property	获取 Double 型属性值
Acm_GetChannelProperty	获取 DI/DO/AI/AO 通道属性值
Acm_SetChannelProperty	设定 DI/DO/AI/AO 通道属性值
Acm_GetMultiChannelProperty	获取由起始通道 ID 和通道数分配的数值连续通道
Acm_SetMultiChannelProperty	设定由起始通道 ID 和通道数分配的数值连续通道

与 IO 控制相关的属性见表 10.2, 这些属性无法被直接调用, 只能通过调用相应属性函数来设定或获取属性值。详细信息请参考附录。

表 10.2: IO 控制属性

配置	说明
CFG_MasCycleTime	获取数据通信的运动环循环时间
CFG_IoCycleTime	获取数据通信的 IO 环循环时间
CFG_CH_DaqDiInvertEnable	设定 / 获取 DI Invert
CFG_CH_DaqDiLowFilter	设定 / 获取 DI 过滤器的下限值
CFG_CH_DaqDiHighFilter	设定 / 获取 DI 过滤器的上限值
CFG_CH_DaqDoFsvEnable	设定 / 获取 DO 故障安全值
CFG_CH_DaqAoRange	设定 / 获取 AO 范围
CFG_CH_DaqAiRange	设定 / 获取 AI 通道输入范围
CFG_CH_DaqAiEnable	启用 / 禁用该 AI 通道
CFG_CH_DaqAiIntegrationTime	设定 / 获取 AI 设定时间
CFG_CH_DaqAoFsv	获取 AO 故障安全值 (Failure Safe Value), 或将 AO 故障安全值设定为当前 AO 值
CFG_CH_DaqCntEnable	启用 / 禁用该 Counter 通道
CFG_CH_DaqCntPulseInMode	设置 / 获取 Counter 反馈脉冲输入模式
CFG_CH_DaqCntInitValue	设置 / 获取 Counter 初始值
CFG_CH_DaqCntMaxValue	设置 / 获取 Counter 最大值
CFG_CH_DaqCntOverflowMode	设置 / 获取 Counter 计数模式
CFG_CH_DaqCntLatchEnable	启用 / 禁用该 Counter 通道锁存功能
CFG_CH_DaqCntLatchEdge	设置 / 获取该 Counter 通道 Latch 信号的触发沿
CFG_CH_DaqCntCmpEnable	启用 / 禁用比较功能
CFG_CH_DaqCntCmpMethod	设置 / 获取比较方法
CFG_CH_DaqCntCmpPulseEnable	启用 / 禁用轴比较输出
CFG_CH_DaqCntCmpPulseMode	设置 / 获取轴比较模式
CFG_CH_DaqCntCmpPulseLogic	设置 / 获取比较信号的逻辑准位

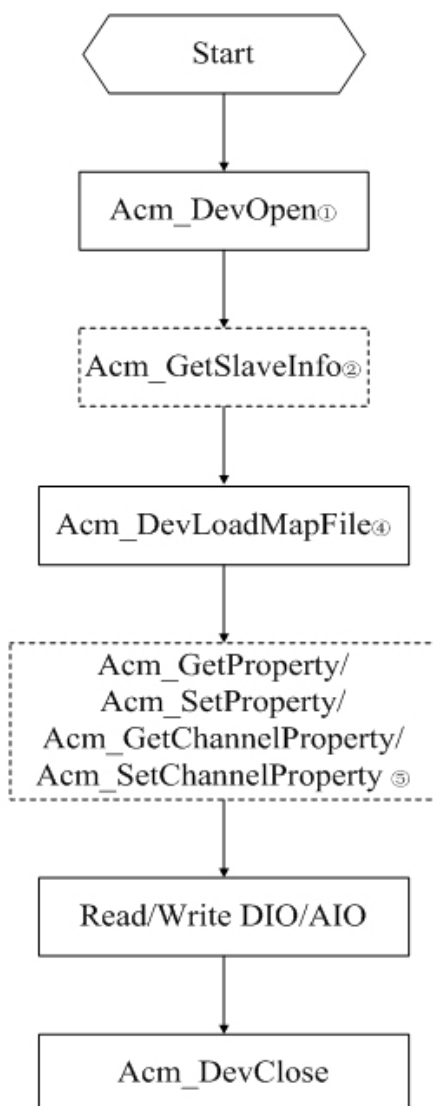
表 10.2: IO 控制属性

CFG_CH_DaqCntCmpPulseWidth	获取 / 设置比较信号延迟的宽度
CFG_CH_DaqAoStartup	获取 A0 启动值 (Startup Value)，或将 A0 启动值设定为当前 A0 值
特性	说明
FT_DaqDiMaxChan	获取网络中所有 DI 通道的数目
FT_DaqDoMaxChan	获取网络中所有 DO 通道的数目
FT_DaqAiMaxDiffChan	获取网络中所有差分 AI 通道的数目
FT_DaqAoMaxChan	获取网络中所有 AO 通道的数目
FT_DaqCntMaxChan	获取网络中所有 CNT 通道的数目

10.2.3 I/O 映射

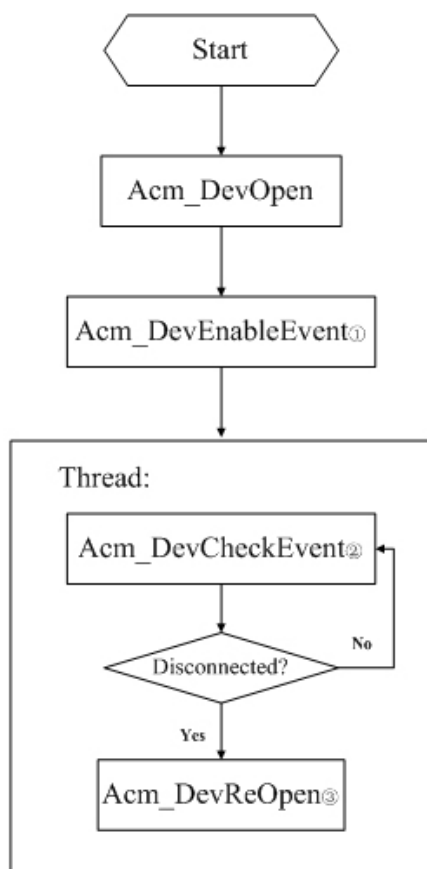
EtherCAT 网络中可能有多个 IO 从站，且每个从站中有多种 DI/DO/AI/AO 模块。映射信息可用于配置和操作这些 IO 模块。使用通用运动 API 时，用户以指定数字需从 / 向从站读 / 写数值。这里，指定的数字被称作逻辑索引或逻辑端口 / 通道。用户可在 Common Motion Utility 中定义逻辑端口 / 通道。关于如何读取和定义映射表的详细信息，请参考章节 4.3.5。

10.2.4 流程图



- (1). DeviceNumber 需要被传递给 Acm_DevOpen。DeviceNumber 可以通过 Acm_GetAvailableDevs 获取。
- (2). 用户可以通过 Acm_GetSlaveInfo 查看从站信息。(非必须)
- (3). 用户可以导入 Utility 生成的映射文件，若用户未更改 DI/O 和 AI/O 的默认映射索引，则无需导入，但用户需要通过 Utility 了解 DI/O 和 AI/O 的逻辑映射索引。
- (4). DI/DO/AI/AO 的属性可通过 Acm_GetChannelProperty/Acm_SetChannelProperty 设置。

10.2.5 事件



- (1). 在检测事件之前，用户需要先启用 Acm_DevEnableEvent。
- (2). 启用事件线程，并在线程中循环调用 Acm_DevCheckEvent。
- (3). 当事件发生时，用户需要调用 Acm_DevReOpen 以再次打开设备。但是，在调用该 API 之前，用户必须确认网络已再次连接。

10.2.6 例程

VC 代码:

```

HAND  m_DevHand; // 设备句柄
U32    Ret; // 函数返回值
U32    DoCount, DiCount, AoCount, AiCount; // DI/O/AIO 数
U8     BitData, ByteData;

---Initialization---
Ret = Acm_GetU32Property(m_DevHand, FT_DaqDoMaxChan, &DoCount);
Ret = Acm_GetU32Property(m_DevHand, FT_DaqDiMaxChan, &DiCount);
  
```



```

Ret = Acm_GetU32Property(m_DevHand, FT_DaqAoMaxChan, &AoCount);
Ret = Acm_GetU32Property(m_DevHand, FT_DaqAiMaxDiffChan, &AiCount);
U8      *DIDataArray =new U8[DiCount/8], *DODataArray =new U8[DoCount/8];
// Set Do and Get Di
for (U32 i = 0; i <DoCount/8; i++)
{
    Ret = Acm_DaqDoSetByte(m_DevHand, i, 0xFF);
    Sleep(100);
    Ret = Acm_DaqDoGetByte(m_DevHand, i, &ByteData);
}
DODataArray[0] = 0x11;
DODataArray[1] = 0xAA;
Ret = Acm_DaqDoSetBytes(m_DevHand, 0, 2, DODataArray);
Ret = Acm_DaqDoGetBytes(m_DevHand, 0, 2, DODataArray);

for (U32 i = 0; i <DiCount; i++)
{
    Ret = Acm_DaqDiGetBit(m_DevHand, i, &BitData);
    if (DiCount%8 == 7)
        Ret = Acm_DaqDiGetByte(m_DevHand, i/8, &ByteData);
}
Ret = Acm_DaqDiGetBytes(m_DevHand, 0, DiCount/8, DIDataArray);

// SetAo and Get Aii
for(U8i = 0; i<AoCount; ++i)
{
    double    aorange = 0.;
    float     AoData = 0;
    U16       AoRawData = 0;
    Ret = Acm_GetChannelProperty(m_DevHand, i, CFG_CH_DaqAoRange,
&aorange);
    Ret = Acm_DaqAoGetRawData(m_DevHand, i, &AoRawData);
    switch((U32)aorange)
    {
        case CFG_DAQ_AO_NEG_10V_TO_10V:
        case CFG_DAQ_AO_NEG_5V_TO_5V:
        case CFG_DAQ_AO_NEG_2500MV_TO_2500MV:
        case CFG_DAQ_AO_NEG_1250MV_TO_1250MV:
        case CFG_DAQ_AO_NEG_625MV_TO_625MV:
        case CFG_DAQ_AO_NEG_0V_TO_10V:
        case CFG_DAQ_AO_NEG_0V_TO_5V:
            Ret = Acm_DaqAoSetVoltData(m_DevHand, i, 4.78);
            Ret = Acm_DaqAoGetVoltData(m_DevHand, i, &AoData);
            printf("Ch %d:Raw:%4X Eng:%fV\n",i, AoRawData, AoData);
            break;
        case CFG_DAQ_AO_0MA_TO_20MA:

```

```

        case CFG_DAQ_AO_4MA_TO_20MA:
            Ret = Acm_DaqAoSetRawData(m_DevHand, i, 0xFFFF);
            Ret = Acm_DaqAoGetCurrData(m_DevHand, i, &AoData);
            printf("Ch %d:Raw:%4X  Eng:%f mA\n", i, AoRawData, AoData);
            break;
    }
}

for(USHORT i = 0; i < AiCount; ++i)
{
    double    airange = 0.;
    float      AiData = 0;
    U16        AiRawData = 0;
    errcde = Acm_GetChannelProperty(devHandle, i, CFG_CH_DaqAiRange,
    &airange);
    errcde = Acm_DaqAiGetRawData(devHandle, i, &AiRawData);
    switch((U32)airange)
    {
        case CFG_DAQ_AI_NEG_10V_TO_10V:
        case CFG_DAQ_AI_NEG_5V_TO_5V:
        case CFG_DAQ_AI_NEG_2500MV_TO_2500MV:
        case CFG_DAQ_AI_NEG_1250MV_TO_1250MV:
        case CFG_DAQ_AI_NEG_625MV_TO_625MV:
        case CFG_DAQ_AI_NEG_1V_TO_1V:
        case CFG_DAQ_AI_NEG_500MV_TO_500MV:
        case CFG_DAQ_AI_NEG_150MV_TO_150MV:
        case CFG_DAQ_AI_NEG_0_TO_10V:
        case CFG_DAQ_AI_NEG_0_TO_500MV:
            Ret = Acm_DaqAiGetVltData(devHandle, i, &AiData);
            printf("Ch %d:Raw:%4X  Eng:%f V\n", i, AiRawData, AiData);
            break;
        case CFG_DAQ_AI_0MA_TO_20MA:
        case CFG_DAQ_AI_4MA_TO_20MA:
        case CFG_DAQ_AI_NEG_20MA_TO_20MA:
            errcde = Acm_DaqAiGetCurrData(devHandle, i, &AiData);
            printf("Ch %d:Raw:%4X  Eng:%f mA\n", i, AiRawData, AiData);
            break;
    }
}
}

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。详细信息请参考 EthcatDI、EthcatDO 和 EthcatAI、EthcatAO 例程。

10.3 比较功能

10.3.1 本节简介

PCI-1203 支持 AMAX-3285 脉冲输出型 EtherCAT 从站。
本节主要介绍触发比较功能相关应用。

10.3.2 比较功能相关函数与属性

比较功能函数如表 10.3 所示，其中的函数可在程序中直接调用。

表 10.3：比较功能函数

函数	说明
Acm_AxSetCmpData	设置指定轴的比较数据
Acm_AxSetCmpTable	设置指定轴的比较数据列表
Acm_AxSetCmpAuto	设置指定轴的线性比较数据
Acm_AxSetCmpDO	设置指定轴的比较 DO 输出
Acm_AxGetCmpData	获取比较器中的当前比较数据

与比较功能相关的属性见表 10.4，这些属性无法被直接调用，只能通过调用相应属性函数来设定或获取属性值。详细信息请参考附录。

表 10.4：比较功能属性

特性	说明
FT_AxCompareMap	获取轴支持的比较特性
配置	说明
CFG_AxCmpSrc	设置 / 获取比较源
CFG_AxCmpMethod	设置 / 获取比较方法
CFG_AxCmpPulseLogic	设置 / 获取比较信号的逻辑准位
CFG_AxCmpPulseWidth	设置 / 获取比较信号延迟的宽度
CFG_AxCmpEnable	启用 / 禁用轴比较功能
CFG_AxCmpPulseMode	设置 / 获取轴比较模式

10.3.3 重点说明

设置轴的比较数据有三种方式，如上函数列表，Acm_AxSetCmpData、Acm_AxSetCmpTable、Acm_AxSetCmpAuto。三者的区别如下表所示：

函数	说明
Acm_AxSetCmpData	每次设定一个比较数据
Acm_AxSetCmpTable	一次可设定多个比较数据
Acm_AxSetCmpAuto	指定比较开始、最后比较的数据，通过比较间隔从开始的数据开始比较，直至最后的比较数据

设置比较数据前，用户需要首先将 CFG_AxCmpEnable 设置为 CMP_Enable。如果用户想要关闭比较功能，只需将属性 CFG_AxCmpEnable 设置 CMP_Disable，而无需清除比较数据。

调用 Acm_AxSetCmpData、Acm_AxSetCmpAuto 和 Acm_AxSetCmpTable 其中的任一函数将清除之前比较数据。比较数据最多可设置 100,000 个。

如果启用通用 DO 输出属性 CFG_AxGenDoEnable，比较功能将禁用，因此将不会输出比较信号。

10.3.4 例程

如下代码实现设置 0 轴的比较事件相关属性及比较值，获取比较数据。

VC 部分代码如下：

```

HAND    m_Axishand;           //0 轴的 handle
U32      Ret;                 // 函数返回值
double   m_dwInterval;
double   m_dwStart;
double   m_dwEnd;
— 初始化过程 —
Ret = Acm_SetU32Property(m_Axishand, CFG_AxCmpSrc, 0); //设置比较源为理论位置
// 设置比较方法为大于等于位置计数器
Ret = Acm_SetU32Property(m_Axishand, CFG_AxCmpMethod, 0);
Ret = Acm_SetU32Property(m_Axishand, CFG_AxCmpEnable, 1); // 启用轴比较功能
m_dwInterval = 1000;           // 比较间隔 1,000
m_dwStart = 1000;             // 首个比较数据 1000
m_dwEnd = 10000;             // 最后比较数据 10000
Ret = Acm_AxSetCmpAuto(m_Axishand, m_dwStart, m_dwEnd, m_dwInterval);
Ret = Acm_AxGetCmpData(m_Axishand, &CurCmpData); // 获取比较数据

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Compare 例程。

10.4 多轴比较功能

10.4.1 本节简介

PCI-1203 支持 AMAX-3285 脉冲输出型 EtherCAT 从站。

本节主要介绍多轴触发比较功能相关应用。

10.4.2 多轴比较功能相关函数与属性

多轴比较功能函数如表 10.5 所示，其中的函数可在程序中直接调用。

表 10.5：多轴比较功能函数	
函数	说明
Acm_AxSetCmpData	设置指定轴的比较数据
Acm_AxSetCmpTable	设置指定轴的比较数据列表
Acm_AxSetCmpAuto	设置指定轴的线性比较数据
Acm_AxGetCmpData	获取比较器中的当前比较数据
Acm_AxSetCmpDO	设置指定轴的比较 DO 输出
Acm_DevMultiTrigSetCmpDO	设置多轴比较相关的 DO 输出
Acm_DevMultiTrigForceCmpOut	手动触发多轴比较
Acm_DevMultiTrigSetPWMTTableOnTime	设置输出方式为 PWM 时，Pb 启动后持续的时间列表

与多轴比较功能相关的属性见表 10.6，这些属性无法被直接调用，只能通过调用相应属性函数来设定或获取属性值。详细信息请参考附录。

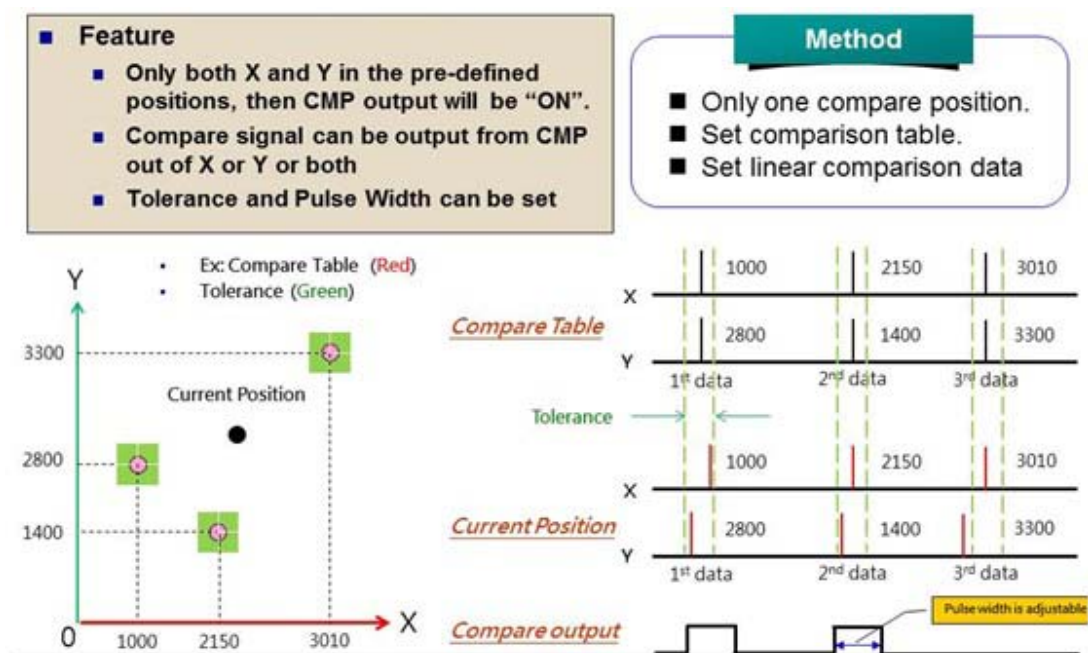
表 10.6：多轴比较功能属性

特性	说明
FT_AxCompareMap	获取轴支持的比较特性
配置	说明
CFG_DevMultiCmpAxisEnable	启用 / 禁用多轴比较功能
CFG_DevMultiCmpOutChannel	启用 / 禁用多轴比较脉冲输出
CFG_DevMultiCmpPulseLogic	设置 / 获取多轴比较信号的逻辑准位
CFG_DevMultiCmpPulseWidth	设置 / 获取多轴比较信号延迟的宽度
CFG_DevMultiTrigPulseMode	设置 / 获取多轴比较模式
CFG_DevMultiTrigPWMEq	设置 / 获取输出方式为 PWM 时，Pb 的输出频率
CFG_DevMultiTrigPWMDuty	设置 / 获取输出方式为 PWM 时，Pb 的占空比
CFG_DevMultiTrigAutoEmptyEn	启用 / 禁用多轴比较结束后，是否自动清空比较数据
CFG_AxisMultiCmpDeviation	设置 / 获取多轴比较时各轴进行比较的误差范围
CFG_AxCmpSrc	设置 / 获取比较源
CFG_AxCmpMethod	设置 / 获取比较方法
CFG_AxCmpPulseLogic	设置 / 获取比较信号的逻辑准位
CFG_AxCmpPulseWidth	设置 / 获取比较信号延迟的宽度
CFG_AxCmpEnable	启用 / 禁用轴比较功能
CFG_AxCmpPulseMode	设置 / 获取轴比较模式

10.4.3 重点说明

10.4.3.1 脉冲输出

用户可以任意指定两轴以上设置比较资料，当进行多轴插补运动时，运动控制卡即能针对位置进行比较并保留数据。只有这些轴同时在各当前 Compare 数据的有效范围内，才算多轴 Compare Match，此时通过指定轴的 OUT5 输出一个宽度可指定的脉冲，并自动加载下一组比较数据。



各轴的 Compare 数据可参照单轴的 Compare 功能，利用 Acm_AxSetCmpData 一次添加一个比较数据，或者利用 Acm_AxSetCmpTable/Acm_AxSetCmpAuto 一次添加多个比较数据。

当各轴运动无法同时进入比较的允许误差范围，则无法触发 D0 输出，并各轴都加载下一组比较数据（如果为 Acm_AxSetCmpTable 或 Acm_AxSetCmpAuto）

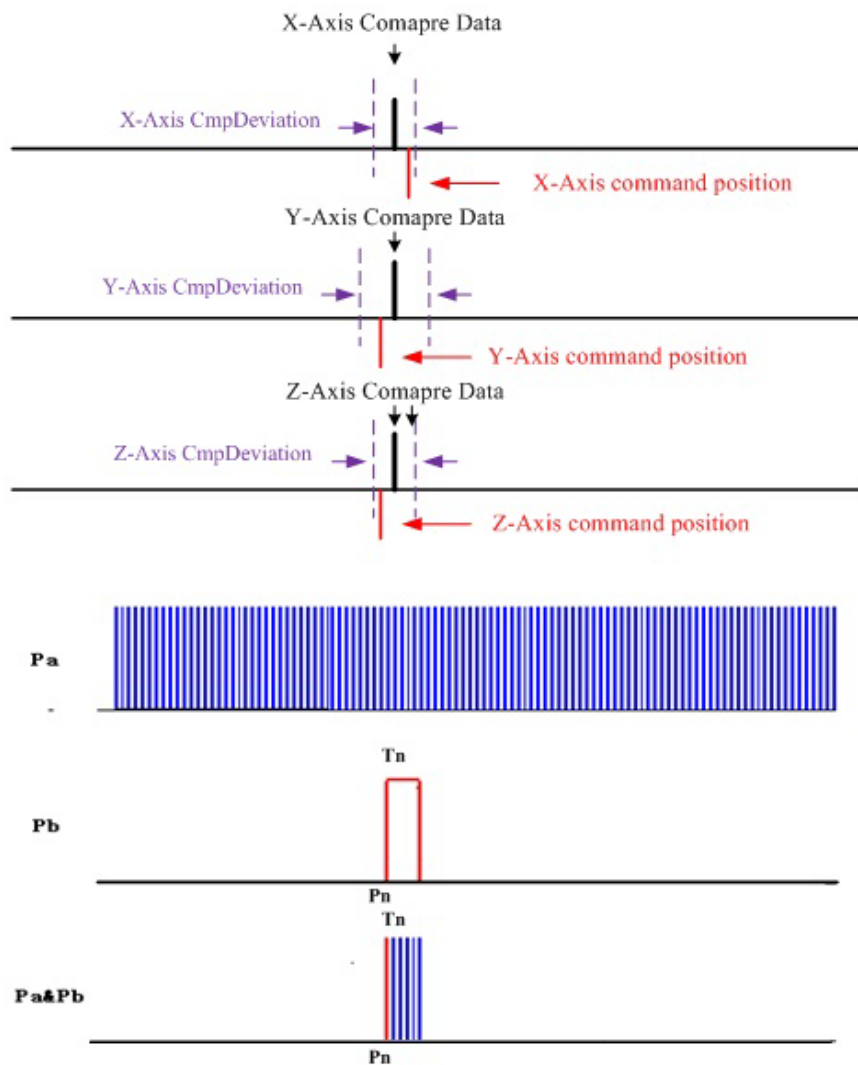
如果用户设定比较结束后自动清空比较寄存器的功能，使用 Acm_AxSetCmpData 设定的比较资料比较成功后，此数据会自动被清除，如果用户想再次比较，则需要再次调用 Acm_AxSetCmpData/ Acm_AxSetCmpTable/ Acm_AxSetCmpAuto，若 Acm_AxSetCmpTable/ Acm_AxSetCmpAuto 添加多笔比较数据，则当所有的数据比较完后，数据会自动被清空，如果下次再需要比较，则需要再次调用 Acm_AxSetCmpData/ Acm_AxSetCmpTable/ Acm_AxSetCmpAuto。

注！ 当 CFG_DevMultiCmpAxisEnable 被启用时，单轴的 CFG_AxCmpEnable 则会被 disable，即两者不可同时使用。



10.4.3.2 PWM 输出

如下图所示，PWM 功能是 Pa 和 Pb 两路脉冲的与输出。



Pa(蓝色脉冲)为持续输出，其频率和占空比为固定值，可以设定。

Pb(红色脉冲)的输出位置和输出的持续时间可以设定。

P1(T1), P2(T2), P3(T3)⋯Pn(Tn) 为启动输出的位置及启动后持续的时间。

Pn: 比较的位置。 Tn: 启动后持续的时间。

也就是说每一个比较点(位置)，将伴随一个时间(PWM持续的时间)。一次可下载30,000点的数据(位置+时间)到卡片上。数据下载到卡片后，当群组中的各轴处在各自当前的Compare数据的有效范围内，才算多轴Compare Match(Pn位置)，此时通过指定轴的OUT5输出PWM并维持“指定的事件(Tn)”。

10.4.4 多轴比较流程图

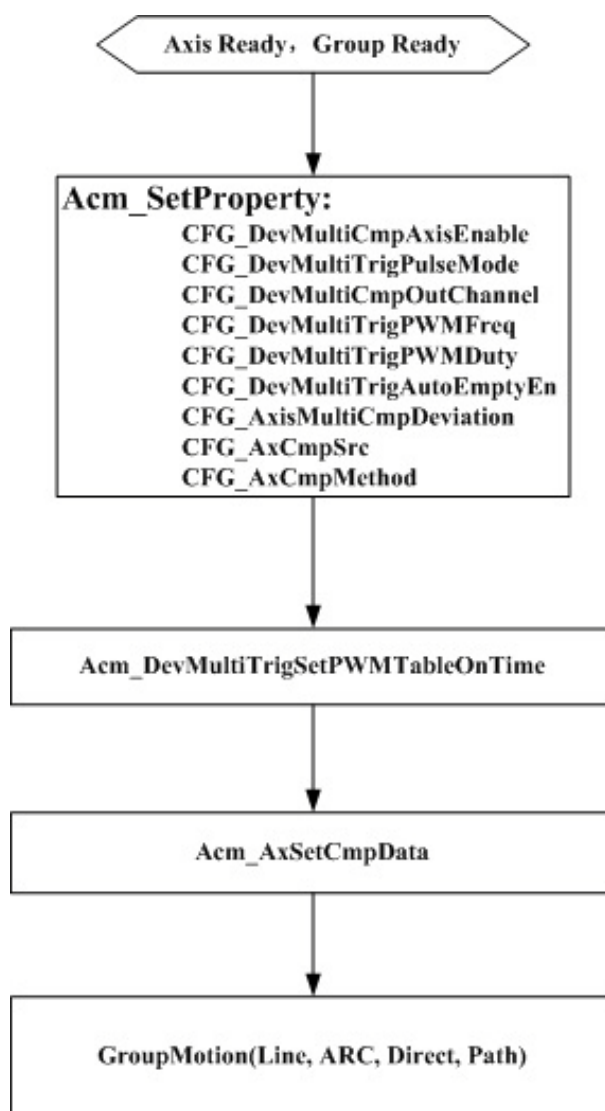


图 10.1：多轴比较流程图

10.5 锁存功能

10.5.1 本节简介

锁存功能一般应用于测量行业。运动控制卡支持锁存单笔及多笔数据，本节将分别介绍该功能。

10.5.2 锁存单笔数据

在轴的运动过程中，可触发锁存信号，锁存当前轴所在的理论位置和实际位置。同时调用 API 可获取锁存的值。

10.5.2.1 锁存功能相关函数与属性

锁存功能函数如表 10.7 所示，其中的函数可在程序中直接调用。

表 10.7：锁存功能函数

函数	说明
Acm_AxGetLatchData	触发锁存后获取设备中的锁存数据
Acm_AxTriggerLatch	触发命令以锁存位置数据
Acm_AxResetLatch	清除设备中的锁存数据和锁存标记
Acm_AxGetLatchFlag	获取设备中的锁存标记

与多轴比较功能相关的属性见表 10.8，这些属性无法被直接调用，只能通过调用相应属性函数来设定或获取属性值。详细信息请参考附录。

表 10.8：锁存功能属性

特性	说明
FT_AxLatchMap	获取轴支持的锁存特性
配置	说明
CFG_AxLatchLogic	设置 / 获取锁存信号的逻辑准位
CFG_AxLatchEnable	启用 / 禁用锁存功能

10.5.2.2 重点说明

当用户只需要锁存单笔数据时，设置 Latch 相关属性及 API 即可实现。在锁存数据之前，需要设置 CFG_AxLatchEnable 属性开启锁存功能，同时设置 CFG_AxLatchLogic 属性设置锁存信号的有效电平。

10.5.2.3 例程

如下代码实现 0 轴执行 PTP 运动中锁存其实际位置。

VC 部分代码如下所示：

```

HAND    m_Axishand;                //0 轴的 handle
U32      Ret;                      // 函数返回值
— 初始化过程 —
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchEnable,1); // 启用 Latch 功能
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchLogic,1); //Latch信号有效电平
Ret =Acm_AxMoveRel(m_Axishand,10000); // 0 轴执行 PTP 运动
Ret =Acm_AxTriggerLatch(m_Axishand); // 触发命令以锁存数据
Ret =Acm_AxGetLatchData(m_Axishand, 1, &LatchPos); // 获取锁存轴的实际位置
Ret =Acm_AxGetLatchFlag(m_Axishand, &LatchFlag); // 获取锁存标记
Ret =Acm_AxResetLatch(m_Axishand); // 清除锁存数据

```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 Latch 例程。

10.5.3 连续锁存功能

在轴的运动过程中，可连续锁存轴的位置信息。

10.5.3.1 连续锁存功能相关函数与属性

连续锁存功能函数如表 10.9 所示，其中的函数可在程序中直接调用。

表 10.9: 连续锁存函数

函数	说明
Acm_AxReadLatchBuffer	从 Latch Buffer 中读取指定数量的资料
Acm_AxGetLatchBufferStatus	获取 Latch Buffer 的状态
Acm_AxResetLatchBuffer	清空 Latch Buffer

与连续锁存功能相关的属性见表 10.10, 这些属性无法被直接调用, 只能通过调用相应属性函数来设定或获取属性值。详细信息请参考附录。

表 10.10: 连续锁存属性

特性	说明
FT_DevLTCBufMaxCount	获取 Latch Buffer 中所支持的最大资料数量
配置	说明
CFG_AxLatchLogic	设置 / 获取锁存信号的逻辑准位
CFG_AxLatchEnable	启用 / 禁用锁存功能
CFG_AxLatchBufEnable	设定 / 读取 Latch Buffer 功能是否开启 / 关闭
CFG_AxLatchBufEdge	设置 / 获取 Latch Buffer 信号的触发沿
CFG_AxLatchBufMinDist	设定 / 读取相邻两笔锁存资料的间隔距离
CFG_AxLatchBufEventNum	设定 / 读取产生事件的数量
CFG_AxLatchBufSource	设定 / 读取某轴的 Latch Buffer 中的资料类型
CFG_AxLatchBufAxisID	设定 / 获取被锁存数据的轴的 ID, 通过该属性可设定一个轴锁存另一轴的数据

10.5.3.2 重点说明

关于使用连续锁存功能的重点说明如下:

■ Latch Buffer 功能的开启与禁用

在使用连续锁存功能时, 首先需要设置属性 CFG_AxLatchEnable 开启 Latch 功能。然后设置属性 CFG_AxLatchBufEnable 开启 Latch Buffer 功能。

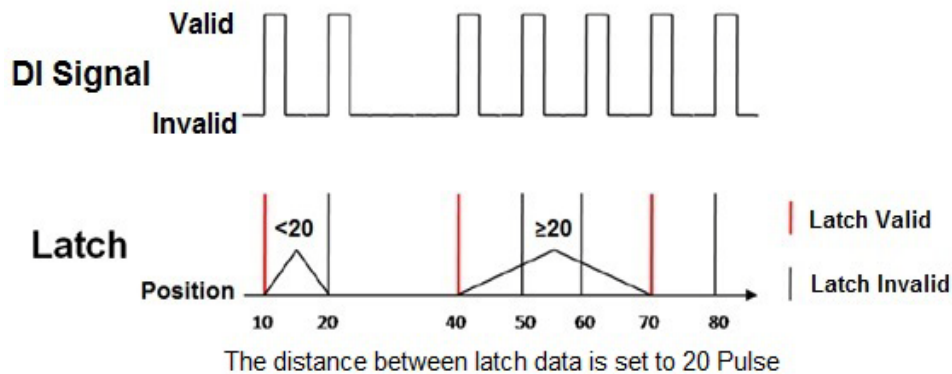
当连续锁存功能被禁用时, 资料不会被记录到 Buffer 中, 但不影响单笔资料的 Latch (前提是 Latch 功能被打开)。当此功能被打开时, 每次进行 Latch 时, 就会将资料记录到 Buffer 中。

■ 锁存最大资料数量

Latch Buffer 中所支持的最大资料数为 128 笔, 当 Latch Buffer 中保存的资料大于 128 笔时, 再次触发 Latch, 资料将不被记录到 Buffer 中。

■ 锁存资料距离

当 Latch 信号过于密集时, 可能会产生一些无效的资料被锁存下来。在这种情况下, 用户可指定相邻两笔资料至少满足相隔多少 pulse 才会被认为是有效的资料, 凡是相较上一次的有效锁存资料的距离大于等于设定值的资料才会被锁存在 Buffer 中。如下图所示:



■ 发生中断

当 Latch Buffer 中的资料大于等于设定的数量后，硬体就会发出中断。用户可以检测到事件 (EVT_AX_LATCHBUF_DONE) 发生，从而做进一步的操作。如果用户需要检测此事件，首先需要通过 Acm_EnableMotionEvent 将 EVT_AX_LATCHBUF_DONE 所在 Bit 写 1 以启用此事件，然后通过 Acm_CheckMotionEvent 进行检测是否有 EVT_AX_LATCHBUF_DONE 发生。

设置属性 CFG_AxLatchBufEventNum 的值，可实现产生中断的 Latch 资料数量。

■ 读取 Latch Buffer 中的资料以及状态

用户可通过 Acm_AxReadLatchBuffer 函数获取轴 Buffer 中的资料，并指定获取多少笔资料，例如当前 Latch Buffer 中有 64 笔资料，用户指定读取 32 笔资料，则读取时将顺序读取前 32 笔资料。

在没有新的 Latch 发生的情况下，每读取一笔资料，Latch Buffer 中的资料数就减一。当指定读取的资料数量大于等于 Buffer 中总的数量时，将读取 Latch Buffer 中的所有资料。

Buffer 中记录的笔数和所剩空间数可通过 Acm_AxGetLatchBufferStatus 函数获取。

10.5.4 例程

如下代码实现 0 轴执行 PTP 运动中连续锁存其实际位置。

VC 部分代码如下：

```

HAND    m_Axishand;           //0 轴的 handle
U32      Ret;                 // 函数返回值
U32      RemainCnt=0;
U32      SpaceCnt=0;
double   m_DataBuffer[128];   // 锁存的数据
— 初始化过程 —
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchEnable, 1); // 启用 latch 功能
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchLogic, 1); //latch信号有效电平
// 启用 latchBuffer 功能
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchBufEdge, 0); // 设置触发沿
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchBufMinDist, 1000); // 锁存距离 1000
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchBufEventNum, 4); // 锁存事件数
Ret =Acm_SetU32Property(m_Axishand, CFG_AxLatchBufSource, 1); // 锁存实际位置
Ret =Acm_AxMoveRel(m_Axishand, 10000); // 0 轴执行 PTP 运动
// 获取 Buffer 中记录的资料数和剩余空间数
Ret =Acm_AxGetLatchBufferStatus(m_Axishand, &RemainCnt, &SpaceCnt);

```

```
DataCnt = 20; // 读取锁存数据的数量
Ret =Acm_AxReadLatchBuffer(m_Axishand,m_DataBuffer,&DataCnt); // 读取锁存的数据
Ret =Acm_AxResetLatchBuffer(m_Axishand); // 清除 Buffer 中锁存的数据
```

建议用户编写程序时，检测函数返回值，以判断函数的执行状态。并建立错误处理机制，保证程序安全可靠运行。

具体使用步骤可参考 LatchBuffer 例程。

附录 A

API 列表

A.1 通用

Acm_GetAvailableDevs

函数原型	U32 Acm_GetAvailableDevs (DEVLIST *DeviceList, U32 MaxEntries, PU32 OutEntries)		
说明	获取已成功加载驱动设备的设备编号和名称列表		
返回值	错误代码		
注解	用安装包安装驱动后，调用该函数，将会检测到所有板卡的编号和名称信息		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceList	DEVLIST*	OUT	指针指向返回可用设备信息列表
MaxEntries	U32	IN	需要获取的最大设备计数
OutEntries	PU32	OUT	指向实际返回的设备数的指针

DEVLIST 为自定义结构体，定义如下：

```
Typedef struct tagPT_DEVLIST
{
    DWORD    DeviceNum;          // Acm_DevOpen 所需要的设备号
    CHAR     DeviceName[50];    // 设备名。例如，PCI-1265/PCI-1245
    SHORT    NumOfSubDevices;    //用于AMONET板卡, 对于Common Motion板卡该值为0
}
```

Acm_GetErrorMessage

函数原型	BOOL Acm_GetErrorMessage (U32 ErrorCode, LPTSTR lpszError, U32 nMaxError)		
说明	根据 API 返回的错误代码，获取错误信息		
返回值	如果成功，返回非零；如果没有错误信息文本内容，返回 0		
注解	Acm_GetErrorMessage 将复制 nMaxError -1 个字符到缓存，并在字符串末尾添加 \0。如果缓存过小，错误信息可能被截断		
函数原型参数			
名称	类型	IN 或 OUT	说明
ErrorCode	U32	IN	API 返回的错误代码
lpszError	LPTSTR	OUT	指针指向错误信息串
nMaxError	U32	IN	接收错误信息的最大串长度

A. 2 设备对象

Acm_DevOpen

函数原型	U32 Acm_DevOpen (U32 DeviceNumber, PHAND DeviceHandle)		
说明	打开一个指定设备以获取设备句柄		
返回值	错误代码		
注解	对设备执行任何操作之前，请先调用该函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceNumber	U32	IN	设备编号
DeviceHandle	PHAND	OUT	返回一个指针，指向设备句柄

Acm_DevClose

函数原型	U32 Acm_DevClose (PHAND DeviceHandle)		
说明	关闭设备		
返回值	错误代码		
注解	通过该函数关闭设备		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄

Acm_DevReOpen

函数原型	U32 Acm_DevReOpen (HAND DeviceHandle)		
说明	重新打开设备		
返回值	错误代码		
注解	在以下情况发生时，用户需调用该 API 重新打开设备：		
	1. 从设备连接断开后，再次建立连接		
	2. 通信不处于 OP 模式		
	3. 有任一从设备不处于 OP 模式		
	4. 获取的从设备状态与实际连接不匹配		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄

Acm_DevLoadConfig

函数原型	U32 Acm_DevLoadConfig (HAND DeviceHandle, PI8 ConfigPath)		
说明	根据加载的配置文件，设置设备的所有配置		
返回值	错误代码		
注解	该函数的具体使用参考第五章		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
ConfigPath	PI8	IN	指针指向保存配置文件路径的字符串

Acm_DevLoadMapFile

函数原型	U32 Acm_DevLoadMapFile (HAND DeviceHandle, PI8 FilePath)		
说明	向 EtherCAT 网络中载入 I/O 映射文件		
返回值	错误代码		
注解	在控制 DI/O 和 AI/O 之前，用户需要将 I/O 映射文件载入自有程序。该文件由 Utility 定义，记录了整个 EtherCAT 网络的所有物理和逻辑映射关系。若用户直接使用默认 I/O 映射关系而不作任何修改，则无需调用该 API 载入映射文件，但用户需要通过 Utility 了解 I/O 映射关系。所有的 DI/O 都由 “Port”（1 port=8 bits）按顺序安排。用户可以重新安排端口号以映射至 ADAM-5000/ECAT 的真实 DI/O 或驱动程序。同样，所有的 AI/O 都由 “Channel” 按顺序安排。用户可以重新安排通道 ID 以映射至 ADAM-5000/ECAT 的真实 AI/O。 请注意，用户需要明了 PortNumber 或 ChannelID 与物理 DI/DO/AI/AO 之间的映射关系。		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
FilePath	PI8	IN	指向保存 I/O 映射文件路径的字符串的指针。

Acm_GetProperty

函数原型	U32 Acm_GetProperty(HAND Handle, U32 PropertyID, PVOID Buffer, PU32 BufferLength)		
说明	通过分配的 PropertyID 获取属性 （特性属性、配置属性或参数属性）。		
返回值	错误代码		
注解	用户需要特别注意数据类型和 BufferLength，以便根据 PropertyID 获取属性值。		
	若 BufferLength 的大小有误，返回值将为错误代码“DataSizeNotCorrect”。此时，Buffer 将返回 BufferLength 属性值。		
	关于 PropertyID 的详细信息，请参考属性表中的 Acm_GetU32Property、Acm_GetI32Property 和 Acm_GetF64Property。		
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acm_DevOpen 的设备句柄，或来自 Acm_AxOpen 的轴句柄，或来自 Acm_GpAddAxis 的群组句柄
PropertyID	U32	IN	要获取的属性 ID
Buffer	PVOID	OUT	返回属性值
BufferLength	PU32	IN/OUT	属性的 Buffer 字节大小。该值必须与相关属性的长度一致，否则将发生错误，且将返回缓存长度属性的真实数值。

Acm_GetU32Property

函数原型	U32 Acm_GetU32Property(HAND Handle, U32 PropertyID, PU32 Value)		
说明	获取所需数据类型为无符号 32 位整型的属性值		
返回值	错误代码		

注解	HAND: 当获取设备的属性时, 输入设备 Handle, 获取轴的属性, 输入轴的 Handle, 获取组的属性, 输入群组的 Handle PropertyID: 每个属性都有唯一属性 ID 与其对应, 所需读取的属性值的数据类型必须为无符号 32bit 整型 Value: 返回的属性值
----	--

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acme_DevOpen 的设备句柄, 或来自 Acme_AxOpen 的轴句柄, 或来自 Acme_GpAddAxis 的群组句柄
PropertyID	U32	IN	要获取的属性 ID
Value	PU32	OUT	属性的返回值

Acm_GetI32Property

函数原型	U32 Acm_GetI32Property(HAND Handle, U32 PropertyID, PI32 Value)
------	---

说明	获取所需数据类型为有符号 32 位整型的属性值
----	-------------------------

返回值	错误代码
-----	------

注解	HAND: 当获取设备的属性时, 输入设备 Handle, 获取轴的属性, 输入轴的 Handle, 获取组的属性, 输入群组的 Handle PropertyID: 每个属性都有唯一属性 ID 与其对应, 所需读取的属性值的数据类型必须为有符号 32bit 整型 Value: 返回的属性值
----	--

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acme_DevOpen 的设备句柄, 或来自 Acme_AxOpen 的轴句柄, 或来自 Acme_GpAddAxis 的群组句柄
PropertyID	U32	IN	要获取的属性 ID
Value	PI32	OUT	属性的返回值

Acm_GetF64Property

函数原型	U32 Acm_GetF64Property(HAND Handle, U32 PropertyID, PF64 Value)
------	---

说明	获取所需数据类型为浮点型的属性值
----	------------------

返回值	错误代码
-----	------

注解	HAND: 当获取设备的属性时, 输入设备 Handle, 获取轴的属性, 输入轴的 Handle, 获取组的属性, 输入群组的 Handle PropertyID: 每个属性都有唯一属性 ID 与其对应, 所需读取的属性值的数据类型必须为浮点型 Value: 返回的属性值
----	---

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acme_DevOpen 的设备句柄, 或来自 Acme_AxOpen 的轴句柄, 或来自 Acme_GpAddAxis 的群组句柄
PropertyID	U32	IN	要获取的属性 ID
Value	PF64	OUT	属性的返回值

Acm_SetProperty

函数原型	U32 Acm_SetProperty(HAND Handle, U32 PropertyID, PVOID Buffer, U32 BufferLength)		
说明	通过分配的 PropertyID 设置属性（特性属性、配置属性或参数属性）。		
返回值	错误代码		
注解	用户需要特别注意，并不是所有属性列表中的属性都能被设置新数值；只有可写属性才能被赋予新数值。		
	若 BufferLength 的大小有误，返回值将为错误代码“DataSizeNotCorrect”。		
	关于 PropertyID 的详细信息，请参考属性表中的 Acm_GetU32Property、Acm_GetI32Property 和 Acm_GetF64Property。		
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acm_DevOpen 的设备句柄，或来自 Acm_AxOpen 的轴句柄，或来自 Acm_GpAddAxis 的群组句柄
PropertyID	U32	IN	要获取的属性 ID
Buffer	PVOID	OUT	返回属性值
BufferLength	PU32	IN/OUT	属性的 Buffer 字节大小。该值必须与相关属性的长度一致，否则将发生错误，且将返回缓存长度属性的真实数值。

Acm_SetU32Property

函数原型	U32 Acm_SeU32tProperty(HAND Handle, U32 PropertyID, U32 Value)		
说明	设定属性值为无符号 32 位整型的属性		
返回值	错误代码		
注解	HAND: 当获取设备的属性时，输入设备 Handle，获取轴的属性，输入轴的 Handle，获取组的属性，输入群组的 Handle PropertyID: 每个属性都有唯一属性 ID 与其对应，所需设置的属性值的数据类型必须为无符号 32 位整型 Value: 需设定的属性值		
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acm_DevOpen 的设备句柄，或来自 Acm_AxOpen 的轴句柄，或来自 Acm_GpAddAxis 的群组句柄
PropertyID	U32	IN	要设置的属性 ID
Value	U32	IN	设置的属性值

Acm_SetI32Property

函数原型	U32 Acm_SeI32tProperty(HAND Handle, U32 PropertyID, I32 Value)
说明	设定属性值为有符号 32 位整型的属性
返回值	错误代码
注解	HAND: 当获取设备的属性时，输入设备 Handle，获取轴的属性，输入轴的 Handle，获取组的属性，输入群组的 Handle
	PropertyID: 每个属性都有唯一属性 ID 与其对应，所需设置的属性值的数据类型必须为无符号 32 位整型
	Value: 需设定的属性值
函数原型参数	

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acme_DevOpen 的设备句柄，或来自 Acme_AxOpen 的轴句柄，或来自 Acme_GpAddAxis 的群组句柄
PropertyID	U32	IN	要设置的属性 ID
Value	I32	IN	设置的属性值

Acme_SetF64Property

函数原型	U32 Acme_SetF64Property (HAND Handle, U32 PropertyID, F64 Value)		
说明	设定属性值为浮点型的属性		
返回值	错误代码		
注解	HAND: 当获取设备的属性时，输入设备 Handle，获取轴的属性，输入轴的 Handle，获取组的属性，输入群组的 Handle PropertyID: 每个属性都有唯一属性 ID 与其对应，所需设置的属性值的数据类型必须为浮点型 Value: 需设定的属性值		

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	对象句柄。该句柄可以是来自 Acme_DevOpen 的设备句柄，或来自 Acme_AxOpen 的轴句柄，或来自 Acme_GpAddAxis 的群组句柄
PropertyID	U32	IN	需要设置的属性 ID
Value	F64	IN	设置的属性值

Acme_GetChannelProperty

函数原型	U32 Acme_GetChannelProperty (HAND Handle, U32 ChannelID, U32 PropertyID, PF64 Value)		
说明	获取 DI/DO/AI/AO 通道属性值		
返回值	错误代码		
注解	若用户需要通过通道获取 DI/DO 属性值。该 channelID 可通过计算获取： $\text{ChannelID} = \text{Port number} * 8 + \text{Port Index}$ 以 Port 3 的第二个 DO 为例， $\text{ChannelID} = 3 * 8 + 2 = 26$ 在使用该 API 获取属性之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 用户可以通过属性列表获取属性值类型的详细信息。 注： 使用该 API 设置属性值之前，需要先调用 Acme_DevLoadMapFile（若用户使用默认的映射关系，则无需载入该 API）		

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acme_DevOpen 的设备句柄
ChannelID	U32	IN	DI/DO/AI/AO 通道 ID
PropertyID	U32	IN	要获取的属性 ID
Value	PF64	OUT	获取属性值

Acm_SetChannelProperty

函数原型	U32 Acm_SetChannelProperty (HAND Handle, U32 ChannelID, U32 PropertyID, F64 Value)		
说明	设置 DI/DO/AI/AO 通道属性值		
返回值	错误代码		
	若用户需要通过通道设置 DI/DO 属性值。该 channelID 可通过计算获取： ChannelID= Port number * 8 + Port Index 以 Port 3 的第二个 DO 为例， ChannelID= 3*8+2=26		
注解	在使用该 API 获取属性之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 用户可以通过属性列表获取属性值类型的详细信息。		
	注： 使用该 API 设置属性值之前，需要先调用 Acm_DevLoadMapFile（若用户使用默认的映射关系，则无需载入该 API）		
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
ChannelID	U32	IN	DI/DO/AI/AO 通道 ID
PropertyID	U32	IN	需要设置的属性 ID
Value	F64	IN	设置属性值

Acm_GetMultiChannelProperty

函数原型	U32 Acm_GetMultiChannelProperty (HAND Handle, U32 PropertyID, U32 StartChID, U32 ChCount, PF64 ValueArray)		
说明	获取由起始通道 ID 和通道数分配的数值连续通道		
返回值	错误代码		
注解	若用户需要通过通道获取 DI/DO 属性值。该 channelID 可通过计算获取： ChannelID= Port number * 8 + Port Index 以 Port 3 的第二个 DO 为例， ChannelID= 3*8+2=26		
	在使用该 API 获取属性之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 用户可以通过属性列表获取属性值类型的详细信息。		
注：			
(1). 使用该 API 设置属性值之前，需要先调用 Acm_DevLoadMapFile（若用户使用默认的映射关系，则无需载入该 API）			
(2). StartChID + ChCount 的和不能大于通道总数			
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
PropertyID	U32	IN	要获取的属性 ID
StartChID	U32	IN	起始通道 ID
ChCount	U32	IN	通道数
ValueArray	PF64	OUT	属性值数列

Acm_SetMultiChannelProperty

函数原型	U32 Acm_SetMultiChannelProperty (HAND Handle, U32 PropertyID, U32 StartChID, U32 ChCount, PF64 ValueArray)
说明	设置由起始通道 ID 和通道数分配的数值连续通道
返回值	错误代码
注解	若用户需要通过通道设置 DI/D0 属性值。该 channelID 可通过计算获取： ChannelID= Port number * 8 + Port Index 以 Port 3 的第二个 D0 为例， ChannelID= 3*8+2=26 在使用该 API 获取属性之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 用户可以通过属性列表获取属性值类型的详细信息。 注： (1). 使用该 API 设置属性值之前，需要先调用 Acm_DevLoadMapFile（若用户使用默认的映射关系，则无需载入该 API） (2). StartChID + ChCount 的和不能大于通道总数

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
PropertyID	U32	IN	需要设置的属性 ID
StartChID	U32	IN	起始通道 ID
ChCount	U32	IN	通道数
ValueArray	PF64	IN	属性值数列

Acm_GetLastError

函数原型	U32 Acm_GetLastError (HAND ObjectHandle)
说明	获取设备、轴或群组最后一个的错误代码
返回值	错误代码
注解	该函数获取的错误代码，可调用 Acm_GetErrorMessage 函数获取错误代码对应的错误信息

函数原型参数

名称	类型	IN 或 OUT	说明
ObjectHandle	HAND	IN	对象句柄。该句柄可以是来自 Acm_DevOpen 的设备句柄，或来自 Acm_AxOpen 的轴句柄，或来自 Acm_GpAddAxis 的群组句柄

Acm_DevEnableEvent

函数原型	U32 Acm_DevEnableEvent (HAND DeviceHandle, U32 MasEnableEvt)
说明	启用事件检测函数
返回值	错误代码
注解	该 API 用于启用事件检测函数。打开连接断开检测函数时，调用 Acm_DevCheckEvent 将发出连接断开事件。 注： 检测事件时，需打开线程以调用 Acm_DevCheckEvent。

函数原型参数

名称	类型	IN 或 OUT	说明
----	----	----------	----

DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄								
启用事件类型:											
MasEnableEvt	U32	IN	<table><tr><th>Bit</th><th>Status</th></tr><tr><td>0</td><td>Motion ring disconnection</td></tr><tr><td>1</td><td>IO ring disconnection</td></tr><tr><td>2~31</td><td>Reserved</td></tr></table>	Bit	Status	0	Motion ring disconnection	1	IO ring disconnection	2~31	Reserved
			Bit	Status							
			0	Motion ring disconnection							
			1	IO ring disconnection							
2~31	Reserved										

Acm_DevCheckEvent

函数原型	U32 Acm_DevCheckEvent (HAND DeviceHandle, PU32 MasCheckEvt, U32 Millisecond)		
说明	启用事件检测函数		
返回值	错误代码		
注解	该 API 仅用于检测由调用 Acm_DevEnableEvent 而产生的事件的状态。打开连接断开检测函数时，调用 Acm_DevCheckEvent 将发出连接断开事件。用户还可以通过调用 Acm_DevGetErrorTable 获取连接断开的轴的信息。 注： 检测事件时，需打开线程以调用 Acm_DevCheckEvent。		

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
MasCheckEvt	PU32	OUT	检测启用事件的状态
Millisecond	U32	IN	为每次检测 （通常为 INFINITE）制定超时时间值（单位：毫秒）

Acm_CheckMotionEvent

函数原型	U32 Acm_CheckMotionEvent (HAND DeviceHandle, PU32 AxEvtStatusArray, PU32 GpEvtStatusArray, U32 AxArrayElements, U32 GpArrayElements, U32 Millisecond)		
说明	检测轴和群组的事件状态		
返回值	错误代码		
注解	当用户获取轴和群组的事件状态时，需要调用 Acm_EnableMotionEvent 函数启用事件。同时，用户创建一个新的线程，用于检查事件状态		

函数原型参数

名称	类型	IN 或 OUT	说明																
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄																
			Array[n]：每个轴的中断事件状态，n 表示运动设备的轴个数。 数列为 32 位数据类型，每一位都代表不同事件类型：																
AxEnableEvtArray	PU32	IN	<table><tr><th>Bit</th><th>31...6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr><tr><td>Description</td><td>Reserved</td><td>EVT_A X_VH_ END</td><td>EVT_A X_VH_ START</td><td>EVT_A X_ERR OR</td><td>EVT_A X_LAT CHED</td><td>EVT_A X_CO MPAR ED</td><td>EVT_A X_MO TION_ DONE</td></tr></table> Bit n = 1: 启用 Axis Motion Done 事件； Bit n = 0: 事件未发生或被禁用。	Bit	31...6	5	4	3	2	1	0	Description	Reserved	EVT_A X_VH_ END	EVT_A X_VH_ START	EVT_A X_ERR OR	EVT_A X_LAT CHED	EVT_A X_CO MPAR ED	EVT_A X_MO TION_ DONE
Bit	31...6	5	4	3	2	1	0												
Description	Reserved	EVT_A X_VH_ END	EVT_A X_VH_ START	EVT_A X_ERR OR	EVT_A X_LAT CHED	EVT_A X_CO MPAR ED	EVT_A X_MO TION_ DONE												

Array[n]: 每个群组的中断事件状态, n 值为 1。可从 PAR_GpGroupID 属性获取。
GpEvtStatus 为 32 为数据类型数列, n 的当前值只能为 1。

GpEnableEvtArray PU32 IN

Bit	Data	31...n	1	0
Description	GpEnable	EVT_GPn_MO	EVT_GP1_MO	EVT_GP0_MO
	EvtArray[0]	TION_DONE	TION_DONE	TION_DONE
	GpEnable	EVT_GPn_VH	EVT_GP1_VH	EVT_GP0_VH
	EvtArray[1]	_START	_START	START
	GpEnable	EVT_GPn_VH	EVT_GP1_VH	EVT_GP0_VH
	EvtArray[2]	_END	_END	END

Bit n = 1: 启用 Axis Motion Done 事件;
Bit n = 0: 事件未发生或被禁用。

注:

EVT_GPn_MOTION_DONE/ EVT_GPn_VH_START/
EVT_GPn_VH_END, n 为 GroupID, 可通过
PAR_GpGroupID 属性获取。

AxArrayElements	U32	IN	AxEvtStatusArray 中元素个数
GpArrayElements	U32	IN	GpEvtStatusArray 中元素个数
Millisecond	U32	IN	设定每次 Check 事件时的等待时间

Acm_EnableMotionEvent

函数原型	U32 Acm_EnableMotionEvent (HAND DeviceHandle, U32 GpArrayElements)
说明	启用轴和群组的事件状态
返回值	错误代码
注解	启用轴或群组的一些事件之后, 可从 Acm_CheckMotionEvent 获取事件状态

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄

Array[n], 启用每个轴的中断事件, n 表示运动设备的轴个数。

数列为 32 位数据类型, 每一位都代表不同事件类型:

AxEnableEvtArray PU32 IN

Bit	31...6	5	4	3	2	1	0
Description	Reserved	EVT_A	EVT_A	EVT_A	EVT_A	EVT_A	EVT_AX_
		X_VH_	X_VH_	X_ERR	X_LAT	X_CO	MOTION
		END	START	OR	CHED	MPAR	_DONE
						ED	

Bit n = 1: 启用事件;
Bit n = 0: 禁用事件。

GpEnableEvtArray PU32 IN

Bit	Data	31....n	1	0
Description	GpEnable	EVT_GPn_MO	EVT_GP1_MO	EVT_GP0_MO
	EvtArray[0]	TION_DONE	TION_DONE	TION_DONE
	GpEnable	EVT_GPn_VH	EVT_GP1_VH	EVT_GP0_VH_
	EvtArray[1]	_START	_START	START
	GpEnable	EVT_GPn_VH	EVT_GP1_VH	EVT_GP0_VH_
	EvtArray[2]	_END	_END	END

Bit n = 0: 禁用事件。

对于 EVT_GPn_MOTION_DONE, , n 为 GroupID, 它可通过 PAR GpGroupID 属性获取。

GpArrayElements	U32	IN	GpEvtStatusArray 元素个数
-----------------	-----	----	-----------------------

函数原型	U32 Acm_DevGetMasInfo(HAND DeviceHandle, PVOID pMasInfo, PU16 SlaveIPArray, PU32 SlvCnt)
------	--

返回值	错误代码
0	成功
1	找不到文件
2	文件已存在
3	文件不存在
4	文件是目录
5	文件不是目录
6	文件是符号链接
7	文件不是符号链接
8	文件是硬链接
9	文件不是硬链接
10	文件是软链接
11	文件不是软链接
12	文件是块设备
13	文件不是块设备
14	文件是字符设备
15	文件不是字符设备
16	文件是 FIFO
17	文件不是 FIFO
18	文件是 socket
19	文件不是 socket
20	文件是普通文件
21	文件不是普通文件
22	文件是目录
23	文件不是目录
24	文件是符号链接
25	文件不是符号链接
26	文件是硬链接
27	文件不是硬链接
28	文件是软链接
29	文件不是软链接
30	文件是块设备
31	文件不是块设备
32	文件是字符设备
33	文件不是字符设备
34	文件是 FIFO
35	文件不是 FIFO
36	文件是 socket
37	文件不是 socket
38	文件是普通文件
39	文件不是普通文件
40	文件是目录
41	文件不是目录
42	文件是符号链接
43	文件不是符号链接
44	文件是硬链接
45	文件不是硬链接
46	文件是软链接
47	文件不是软链接
48	文件是块设备
49	文件不是块设备
50	文件是字符设备
51	文件不是字符设备
52	文件是 FIFO
53	文件不是 FIFO
54	文件是 socket
55	文件不是 socket
56	文件是普通文件
57	文件不是普通文件
58	文件是目录
59	文件不是目录
60	文件是符号链接
61	文件不是符号链接
62	文件是硬链接
63	文件不是硬链接
64	文件是软链接
65	文件不是软链接
66	文件是块设备
67	文件不是块设备
68	文件是字符设备
69	文件不是字符设备
70	文件是 FIFO
71	文件不是 FIFO
72	文件是 socket
73	文件不是 socket
74	文件是普通文件
75	文件不是普通文件
76	文件是目录
77	文件不是目录
78	文件是符号链接
79	文件不是符号链接
80	文件是硬链接
81	文件不是硬链接
82	文件是软链接
83	文件不是软链接
84	文件是块设备
85	文件不是块设备
86	文件是字符设备
87	文件不是字符设备
88	文件是 FIFO
89	文件不是 FIFO
90	文件是 socket
91	文件不是 socket
92	文件是普通文件
93	文件不是普通文件
94	文件是目录
95	文件不是目录
96	文件是符号链接
97	文件不是符号链接
98	文件是硬链接
99	文件不是硬链接
100	文件是软链接
101	文件不是软链接
102	文件是块设备
103	文件不是块设备
104	文件是字符设备
105	文件不是字符设备
106	文件是 FIFO
107	文件不是 FIFO
108	文件是 socket
109	文件不是 socket
110	文件是普通文件
111	文件不是普通文件
112	文件是目录
113	文件不是目录
114	文件是符号链接
115	文件不是符号链接
116	文件是硬链接
117	文件不是硬链接
118	文件是软链接
119	文件不是软链接
120	文件是块设备
121	文件不是块设备
122	文件是字符设备
123	文件不是字符设备
124	文件是 FIFO
125	文件不是 FIFO
126	文件是 socket
127	文件不是 socket
128	文件是普通文件
129	文件不是普通文件
130	文件是目录
131	文件不是目录
132	文件是符号链接
133	文件不是符号链接
134	文件是硬链接
135	文件不是硬链接
136	文件是软链接
137	文件不是软链接
138	文件是块设备
139	文件不是块设备
140	文件是字符设备
141	文件不是字符设备
142	文件是 FIFO
143	文件不是 FIFO
144	文件是 socket
145	文件不是 socket
146	文件是普通文件
147	文件不是普通文件
148	文件是目录
149	文件不是目录
150	文件是符号链接
151	文件不是符号链接
152	文件是硬链接
153	文件不是硬链接
154	文件是软链接
155	文件不是软链接
156	文件是块设备
157	文件不是块设备
158	文件是字符设备
159	文件不是字符设备
160	文件是 FIFO
161	文件不是 FIFO
162	文件是

注解

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
pMasInfo	PVOID	IN	0: Motion ring 1: IO ring
SlaveIPArray	PU16	OUT	从指定环返回的从站 ID 列表
SlvCnt	PU32	OUT	从站数

函数原型	U32 Acm_DevGetSlaveInfo(HAND DeviceHandle, U16 RingNo, U16 SlaveIP, PVOID pInfo)
------	--

返回值	错误代码
-----	------

注解

函数原型参数

名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
RingNo	U16	IN	0: Motion ring 1: IO ring
SlaveIP	U16	IN	从站 ID 范围：0x001~0xffff

			由从站返回的信息。 该值为指向下列 Struct 类型的指针： typedef struct _ADVAPI_SLAVE_INFO_ { ULONG SlaveID; ULONG Position; ULONG VendorID; ULONG ProductID; ULONG RevisionNo; ULONG SerialNo; ULONG driverCnts; port_info ports[4]; ULONG transmission_delay; char DeviceName[64]; }ADVAPI_SLAVE_INFO, *PADVAPI_SLAVE_INFO;
pInfo	PVOID	OUT	

Acm_DevGetModuleInfo

函数原型	U32 Acm_DevGetModuleInfo(HAND DeviceHandle, U16 RingNo, U16 SlaveIP, PU32 ModIDArray, PU32 ModCnt)		
说明	根据设备上的旋转开关获取模块信息		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
RingNo	U16	IN	0: Motion ring 1: IO ring
SlaveIP	U16	IN	从站 ID 范围：0x001~0xffff
ModIDArray	PU32	OUT	从指定的从站返回的模块 ID 列表
ModCnt	PU32	OUT	模块数

Acm_DevSetSlaveID

函数原型	U32 Acm_DevSetSlaveID(HAND DeviceHandle, U16 RingNo, U16 SlaveIP, U16 SlaveNewIP)		
说明	为从站设置新 ID		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
Handle	HAND	IN	来自 Acm_DevOpen 的设备句柄
RingNo	U16	IN	0: Motion ring 1: IO ring
SlaveIP	U16	IN	从站 ID 范围：0x001~0xffff
SlaveNewIP	U16	IN	新的从站 ID，无法与环中其他从站 ID 进行复制。 范围：0x001~0xffff

Acm_DevDownloadCAMTable

函数原型	U32 Acm_DevDownloadCAMTable (HAND DeviceHandle, U32 CamTableID, PF64 pMasterArray, PF64 pSlaveArray, PF64 pPointRangeArray, PF64 pPointSlopeArray, U32 ArrayElements)		
说明	该函数用于加载凸轮关系表，该表描述主轴和从轴之间的关系		
返回值	错误代码		
注解	关于该函数的详细信息，参见电子凸轮章节		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
CamTableID	U32	IN	CAM 表的标识符。PCI-1245、PCI-1265 和 PCI-1203 保留了 2 组 CAM 表。因此 ID 为 0 和 1
pMasterArray	PF64	IN	CAM 表中主轴的位置列表
pSlaveArray	PF64	IN	CAM 表中从轴的位置列表
pPointRangeArray	PF64	IN	CAM 表中各点与辅助点之间的距离
pPointSlopeArray	PF64	IN	CAM 表中各点与其辅助点之间连线的斜率
ArrayElements	U32	IN	pMasterArray/pSlaveArray/pPointRangeArray/pPointSlopeArray 阵列中的元素个数。最大元素个数为 128

Acm_DevConfigCAMTable

函数原型	U32 Acm_DevConfigCAMTable (HAND DeviceHandle, U32 CamTableID, U32 Periodic, U32 MasterAbsolute, U32 SlaveAbsolute)		
说明	该函数设置 CAM 表的相关参数		
返回值	错误代码		

凸轮操作是靠这张表执行的（二维 - 描述主和从位置）
 凸轮系统有两种类型

■ Periodic

定期：一旦开始执行，凸轮系统将立刻从曲线的起始处重新开始



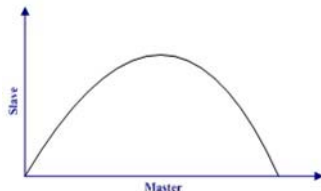
非定期：执行曲线后，凸轮执行完一个周期后，结束凸轮关系的执行



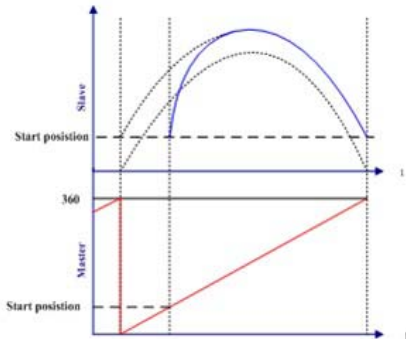
■ MasterAbsolute 和 SlaveAbsolute

绝对：当绝对凸轮系统设置后，基于 CamTable 的控制值或从值将被认为是绝对的。系统会补偿同步过程中主从轴之间的任何偏差。当达到同步时，控制值和从值之间将建立一种确定的相位关系。
 相对：当相对凸轮系统设置后，则表示系统不会补偿在同步过程中控制值和从值之间的任何偏移。比如，实用程序中剖面图如下所示：
 初始 CAM 曲线如下图所示：

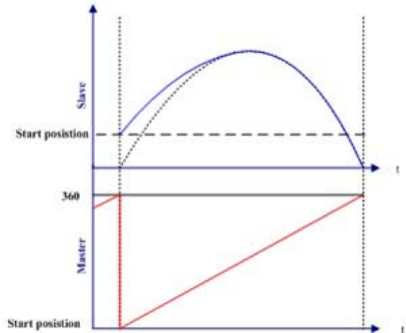
注解



- 主轴：绝对；从轴：相对



- 主轴：相对；从轴：绝对



函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄

CamTableID	U32	IN	CAM 表的标识符，Acm_DevDownloadCAMTable 分配设备内保留了 2 个 CAM 表，ID 为 0 和 1
Periodic	U32	IN	定期或非定期执行 CAM 曲线。 0: 非定期 1: 定期
MasterAbsolute	U32	IN	指定 CAM 曲线相对（0）或绝对（1）于主轴。 0: 相对 1: 绝对
SlaveAbsolute	U32	IN	指定 CAM 曲线相对（0）或绝对（1）于从轴。 0: 相对 1: 绝对

Acm_DevLoadCAMTableFile

函数原型	U32 Acm_DevLoadCAMTableFile (HAND DeviceHandle, PI8 FilePath, U32 CamTableID, PU32 Range, PU32 PointsCount)
说明	加载编辑过的 Cam Table 文件，并通过 Utility 将其保存至设备
返回值	错误代码
注解	CamTable 文件在 Utility 中保存为 .bin 格式。当该 API 成功加载文件时，则不需要调用 Acm_DevDownLoadCACMTable

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
FilePath	PI8	IN	指针指向保存 CamTable 文件路径的字符串
CamTableID	U32	IN	CamTableID: 0 或 1
Range	PU32	OUT	主轴在一个周期内需要的脉冲个数
PointsCount	PU32	OUT	CamTable 中的点数量

Acm_DevMDaqConfig

函数原型	U32 Acm_DevMDaqConfig (HAND DeviceHandle, U16 ChannelID, U32 Period, U32 AxisNo, U32 Method, U32 ChanType, U32 Count)
说明	设定 MotionDAQ 相关配置
返回值	错误代码
注解	

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
ChannelID	U16	IN	指定 Channel ID, 范围: 0 ~ 3
Period	U32	IN	设置每记录一笔资料的间隔时间 范围: 0 ~ 255, 单位: ms
AxisNo	U32	IN	指定记录哪一个轴的信息。范围: 0 ~ 3 (PCI-1245 series), 0 ~ 5 (PCI-1265), 0 ~ 31 (PCI-1203)

Method	U32	IN	触发 MDaq 的方式 0: 禁用 1: 软体触发方式 2: DI 触发 3: 轴 0 启动触发 4: 轴 1 启动触发 5: 轴 2 启动触发 6: 轴 3 启动触发 7: 轴 4 启动触发 8: 轴 5 启动触发 9: 轴 6 启动触发 10: 轴 7 启动触发 11: 轴 8 启动触发 12: 轴 9 启动触发 13: 轴 10 启动触发 14: 轴 11 启动触发
ChanType	U32	IN	获取 Channel Type: 0: 理论位置; 1: 实际位置; 2: 理论位置与实际位置的偏差值; 3: 理论速度值 (暂不支持)
Count	U32	IN	指定要记录的资料数量, 范围: 0 ~ 2000

Acm_DevMDaqGetConfig

函数原型	U32 Acm_DevMDaqGetConfig (HAND DeviceHandle, U16 ChannelID, PU32 Period, PU32 AxisNo, PU32 Method, PU32 ChanType, PU32 Count)		
说明	获取指定 ChannelID 的 MotionDAQ 相关设定值		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
ChannelID	U16	IN	获取指定通道的 MDaq 配置信息，范围：0 ~ 3
Period	PU32	OUT	获取每记录一笔资料的间隔时间 范围：0 ~ 255，单位：ms
AxisNo	PU32	OUT	获取记录哪一个轴的相关资料 范围：0 ~ 3(PCI-1245 series), 0 ~ 5(PCI-1265), 0 ~ 31(PCI-1203)

Method	PU32	OUT	触发 MDAQ 的方式： 0: 禁用 1: 软体触发方式 2: DI 触发 3: 轴 0 启动触发 4: 轴 1 启动触发 5: 轴 2 启动触发 6: 轴 3 启动触发 7: 轴 4 启动触发 8: 轴 5 启动触发 9: 轴 6 启动触发 10: 轴 7 启动触发 11: 轴 8 启动触发 12: 轴 9 启动触发 13: 轴 10 启动触发 14: 轴 11 启动触发
ChanType	PU32	OUT	获取 Channel Type: 0: 理论位置 1: 实际位置 2: 理论位置与实际位置的偏差值 3: 理论速度值（暂不支持）
Count	PU32	OUT	获取设定的最大记录笔数，范围：0 ~ 2000

Acm_DevMDaqStart

函数原型	Acm_DevMDaqStart(HAND DeviceHandle)		
说明	启动 MotionDAQ 功能，以记录相关资料		
返回值	错误代码		
注解	当 Acm_DevMDaqConfig 中 Method 设定为 MQ_TRIG_SW 时，通过此 API 下达命令以触发 MotionDAQ 功能		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄

Acm_DevMDaqStop

函数原型	U32 Acm_DevMDaqStop (HAND DeviceHandle)		
说明	停止记录 MotionDAQ 相关资料		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄

Acm_DevMDaqReset

函数原型	U32 Acm_DevMDaqReset (HAND DeviceHandle, U16 ChannelID)
说明	清除相应 ChannelID 的 MotionDAQ 资料
返回值	错误代码
注解	
函数原型参数	

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
ChannelID	U16	IN	Channel ID, 范围: 0 ~ 3

Acm_DevMDaqGetStatus

函数原型	U32 Acm_DevMDaqGetStatus(HAND DeviceHandle, U16 ChannelID, PU16 CurrentCnt, PU8 Status)		
说 明	获取相应 ChannelID 的 MotionDAQ 状态		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
ChannelID	U16	IN	Channel ID。范围: 0 ~ 3
CurrentCnt	PU16	OUT	返回当前已记录的笔数
Status	PU8	OUT	返回当前状态: 0: 待命 1: 等待触发 2: 启动

Acm_DevMDaqGetData

函数原型	U32 (HAND DeviceHandle, U16 ChannelID, U16 StartIndex, U16 MaxCount, PI32 DataBuffer)		
说明	获取相应 Channel 已经记录的 MotionDAQ 的指定范围内的所有资料		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
ChannelID	U16	IN	Channel ID。范围: 0 ~ 3。
StartIndex	U16	IN	设置获取资料的起始位置
MaxCount	U16	IN	设置获取资料的数量
DataBuffer	PI32	OUT	返回指定范围内的资料

Acm_DevMultiTrigSetCmpD0

函数原型	U32 Acm_DevMultiTrigSetCmpD0(HAND DeviceHandle, U32 OFForON)		
说明	设定多轴比较相关的 D0 输出		
返回值	错误代码		
注解	参考 ACM_AxSetCmpD0, 不同的是这个功能是对多轴比较的各个 OUT 同时作用。 详见 10.4 节。 仅特定 EtherCAT 从站支持此函数。		
函数原型参数			
名称	类型	IN 或 OUT	说明

DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
OFForON	U32	IN	0: OFF, 1:ON

Acm_DevMultiTrigForceCmpOut

函数原型	U32 Acm_DevMutiTrigForceCmpOut (HAND DeviceHandle, U32 OnOff)		
说明	手动触发多轴比较输出		
返回值	错误代码		
注解	手动触发 PWM 的输出时间是通过 Acm_DevMultiTrigSetPWMTTableOnTime 设定的，每次在手动触发之前先需要设定比较数据（Acm_SetCmpData），设定输出时间（Acm_DevMultiTrigSetPWMTTableOnTime）。详见 10.4 节。 仅特定 EtherCAT 从站支持此函数。		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
OnOff	U32	IN	输出 / 不输出（保留参数）

Acm_DevMultiTrigSetPWMTTableOnTime

函数原型	U32 Acm_DevMultiTrigSetPWMTTableOnTime (HAND DeviceHandle, PU32 TimeTableArray, U32 ArrayCount)		
说明	设定 PWM 功能中 Pb 启动后持续的时间列表		
返回值	错误代码		
注解	每次运行结束，重启 PWM 功能前，需要通过该 API 重新下载时间数组，并重新 SetCmpData。 详见 10.4 节。 仅特定 EtherCAT 从站支持此函数。		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
TimeTableArray	PU32	IN	每个比较位置启动后的持续时间。（单位：us）一次最多设定 30000 笔。
ArrayCount	U32	IN	表中的时间数据个数。该值大于实际数组个数时，将数组后续补 0 下载到板卡中；该值若小于实际数组个数，将只截取 ArrayCount 个时间值写入板卡。

A. 3 DAQ

A. 3. 1 数字量输入 / 输出

Acm_DaqDiGetByte

函数原型	U32 Acm_DaqDiGetByte (HAND DeviceHandle, U16 DiPort, PU8 ByteData)		
说明	以字节形式获取指定 DI 端口的数据		
返回值	错误代码		

注解	通过 Port Number 获取 DI 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DI 值会受 CFG_CH_DaqDiInvertEnable、CFG_CH_DaqDiLowFilter、CFG_CH_DaqDiHighFilter 影响。请先确认属性值是否正确。（请参考属性章节） 注： (1) DI 端口号必须与 Utility 中的映射信息一致。 (2) 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	---

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
DiPort	U16	IN	数值应为 0
ByteData	PU8	OUT	用于获取数值的指针

Acm_DaqDiGetBit

函数原型	U32 Acm_DaqDiGetBit (HAND DeviceHandle, U16 DiChannel, PU8 BitData)
说明	获取指定 DI 通道的位数据
返回值	错误代码

注解	通过 Port Number 与 ChannelID 获取 DI 值。 该 channelID 可通过计算获取： ChannelID= Port number * 8 + Port Index 以 Port 3 的第二个 DI 为例， ChannelID= 3*8+2=26 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DI 值会受 CFG_CH_DaqDiInvertEnable、CFG_CH_DaqDiLowFilter、CFG_CH_DaqDiHighFilter 影响。请先确认属性值是否正确。（请参考属性章节） 注： 用户在使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	---

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
DiChannel	U16	IN	DI 通道 ID。范围：0~最大 DI 数-1
BitData	PU8	OUT	返回的位数据，值为 0 或 1

Acm_DaqDiGetBytes

函数原型	U32 Acm_DaqDiGetBytes (HAND DeviceHandle, U16 StartPort, U16 NumPort, PU8 ByteDataArray)
说明	以字节形式获取从 StartPort 到 (StartPort + NumPort - 1) 的 DI 端口数据
返回值	错误代码

注 解	通过 Port Number 获取连续 DI 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DI 值会受 CFG_CH_DaqDiInvertEnable、CFG_CH_DaqDiLowFilter、CFG_CH_DaqDiHighFilter 影响。请先确认属性值是否正确。（请参考属性章节） 注： (1) (StartPort + NumPort) 必须小于端口总数。 (2) 使用该 API 之前，需要先调用 Acm_DevLoadMapFile 。（若用户使用默认的映射关系，则无需载入该 API）
-----	---

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
StartPort	U16	IN	起始 DI 通道
NumPort	U16	IN	要获取的 DI 通道数
ByteDataArray	PU8	OUT	返回信息个数

Acm_DaqDoSetByte

函数原型	U32 Acm_DaqDoSetByte (HAND DeviceHandle, U16 DoPort, U8 ByteData)
说 明	设置指定 DO 端口的值
返回值	错误代码
注 解	通过 Port Number 设置 DO 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile 。（若用户使用默认的映射关系，则无需载入该 API）

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
DoPort	U16	IN	DO 端口
ByteData	U8	IN	需要设置的值

Acm_DaqDoSetBit

函数原型	U32 Acm_DaqDoSetBit (HAND DeviceHandle, U16 DoChannel, U8 BitData)
说 明	设置指定 DO 通道的值
返回值	错误代码

注解	通过 Port Number 与 ChannelID 设置 DO 值。 该 channelID 可通过计算获取： $\text{ChannelID} = \text{Port number} * 8 + \text{Port Index}$ 以 Port 3 的第二个 DO 为例， $\text{ChannelID} = 3 * 8 + 2 = 26$ 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： 用户在使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
DoChannel	U16	IN	DO 通道 ID。范围：0~最大 DO 数-1
BitData	U8	IN	需要设置的值，设置的值为 0 或 1

Acm_DaqDoSetBytes

函数原型	U32 Acm_DaqDoSetBytes (HAND DeviceHandle, U16 StartPort, U16 NumPort, PU8 ByteDataArray)
说明	以字节形式设置从 StartPort 到 (StartPort + NumPort - 1) 的 DO 端口数据
返回值	错误代码

注解	通过 Port Number 设置连续 DO 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： (1) (StartPort + NumPort) 必须小于端口总数。 (2) 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	---

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
StartPort	U16	IN	起始 DO 通道
NumPort	U16	IN	要获取的 DO 通道数
ByteDataArray	PU8	OUT	返回信息个数

Acm_DaqDoGetByte

函数原型	U32 Acm_DaqDoGetByte (HAND DeviceHandle, U16 DoPort, PU8 ByteData)
说明	获取指定 DO 端口的字节数据
返回值	错误代码

注解	通过 Port Number 获取 DO 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile 。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
DoPort	U16	IN	DO 端口
ByteData	PU8	OUT	返回值

Acm_DaqDoGetBit

函数原型	U32 Acm_DaqDoGetBit (HAND DeviceHandle, U16 DoChannel, PU8 BitData)
说 明	获取指定 DO 通道的位值
返回值	错误代码

注 解	通过 Port Number 与 ChannelID 获取 DO 值。 该 channelID 可通过计算获取： ChannelID= Port number * 8 + Port Index 以 Port 3 的第二个 DO 为例， ChannelID= 3*8+2=26 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： 用户在使用该 API 之前，需要先调用 Acm_DevLoadMapFile 。（若用户使用默认的映射关系，则无需载入该 API）
-----	---

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	来自 Acm_DevOpen 的设备句柄
DoChannel	U16	IN	DO 通道 ID。范围：0~ 最大 DO 数 -1
BitData	PU8	OUT	返回值 0 或 1

Acm_DaqDoGetBytes

函数原型	U32 Acm_DaqDoGetBytes (HAND DeviceHandle, U16 StartPort, U16 NumPort, PU8 ByteDataArray)
说 明	以字节形式获取从 StartPort 到 (StartPort + NumPort - 1) 的 DO 端口数据
返回值	错误代码

注 解	通过 Port Number 获取连续 DO 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 DO 值会受 CFG_CH_DaqDoFsvEnable 影响。请先确认属性值是否正确。（请参考属性章节） 注： (1) (StartPort + NumPort) 必须小于端口总数。 (2) 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）		
	函数原型参数		
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
StartPort	U16	IN	起始 DO 通道
NumPort	U16	IN	要获取的 DO 通道数
ByteDataArray	PU8	OUT	返回信息个数

A. 3. 2 模拟量输入 / 输出

Acm_DaqAiGetRawData

函数原型	U32 Acm_DaqAiGetRawData(HAND DeviceHandle, U16 AiChannel, PU16 AiData)		
说 明	获取模拟量输入值的二进制值		
返回值	错误代码		
注 解	通过 ChannelID 获取二进制 AI 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 AI 值会受 CFG_CH_DaqAiRange、CFG_CH_DaqAiEnable、CFG_CH_DaqAiIntegrationTime 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）		
	函数原型参数		
名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
AiChannel	U16	IN	AI 通道 ID。范围：0~最大 AI 数-1
AiData	PU16	OUT	指向返回的二进制 AI 值的指针

Acm_DaqAiGetVoltData

函数原型	U32 Acm_DaqAiGetVoltData(HAND DeviceHandle, U16 AiChannel, PF32 AiData)		
说明	获取实际模拟量输入值		
返回值	错误代码		

注解	通过 ChannelID 获取 AI 电压值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 AI 通道通过 CFG_CH_DaqAiRange 设置为电压电平，用户可以获得输入电压值；当 AI 通道设置为电流电平时调用此 API 时将发生错误。 AI 值会受 CFG_CH_DaqAiRange、CFG_CH_DaqAiEnable、CFG_CH_DaqAiIntegrationTime 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
AiChannel	U16	IN	AI 通道 ID。范围：0~ 最大 AI 数 -1
AiData	PF32	OUT	指向返回的模拟量值的指针

Acm_DaqAiGetCurrData

函数原型	U32 Acm_DaqAiGetCurrData (HAND DeviceHandle, U16 AiChannel, PF32AiData)
说明	获取模拟量输入通道电流值
返回值	错误代码

注解	通过 ChannelID 获取 AI 电流值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 AI 通道通过 CFG_CH_DaqAiRange 设置为电流档位，用户可以获得输入电流值；当 AI 通道设置为电压电平时调用此 API 时将发生错误。 AI 值会受 CFG_CH_DaqAiRange、CFG_CH_DaqAiEnable、CFG_CH_DaqAiIntegrationTime 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AiChannel	U16	IN	AI 通道 ID。范围：0~ 最大 AI 数 -1
AiData	PF32	OUT	指向返回的模拟量值的指针

Acm_DaqAoSetRawData

函数原型	U32 Acm_DaqAoSetRawData (HAND DeviceHandle, U16 AoChannel, U16AoData)
说明	设置模拟量输出值的二进制值
返回值	错误代码

注解	通过 ChannelID 设置二进制 AO 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 AO 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	---

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	AO 通道 ID。范围：0~最大 AO 数-1
AoData	PF32	OUT	二进制 AO 值

Acm_DaqAoSetVoltData

函数原型	U32 Acm_DaqAoSetVoltData (HAND DeviceHandle, U16 AoChannel, F32AoData)
说明	设置模拟量输出电压值
返回值	错误代码

注解	通过 ChannelID 设置 AO 电压值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 AO 通道通过 CFG_CH_DaqAoRange 设置为电压电平，用户可以设置输出电压值；当 AO 通道设置为电流电平时调用此 API 时将发生错误。 AO 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	AO 通道 ID。范围：0~最大 AO 数-1
AoData	PF32	IN	AO 电压值

Acm_DaqAoSetCurrData

函数原型	U32 Acm_DaqAoSetCurrData (HAND DeviceHandle, U16 AoChannel, F32AoData)
说明	设置模拟量输出电流值
返回值	错误代码

注解	通过 ChannelID 设置 AO 电流值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 AO 通道通过 CFG_CH_DaqAoRange 设置为电流电平，用户可以设置输出电流值；当 AO 通道设置为电压电平时调用此 API 时将发生错误。 AO 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	A0 通道 ID。范围：0~ 最大 A0 数 -1
AoData	PF32	IN	A0 电压值

Acm_DaqAoGetRawData

函数原型	U32 Acm_DaqAoGetRawData (HAND DeviceHandle, U16 AoChannel, PU16 AoData)
说明	获取模拟量输出值的二进制值
返回值	错误代码

注解	通过 ChannelID 获取二进制 A0 值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 A0 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	--

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	A0 通道 ID。范围：0~ 最大 A0 数 -1
AoData	PU16	OUT	指向返回的二进制 A0 值的指针

Acm_DaqAoGetVoltData

函数原型	U32 Acm_DaqAoGetVoltData (HAND DeviceHandle, U16 AoChannel, PF32AoData)
说明	获取模拟量输出电压值
返回值	错误代码

注解	通过 ChannelID 获取 A0 电压值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 A0 通道通过 CFG_CH_DaqAoRange 设置为电压电平，用户可以获取输出电压值；当 A0 通道设置为电流电平时调用此 API 时将发生错误。 A0 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile。（若用户使用默认的映射关系，则无需载入该 API）
----	---

函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	A0 通道 ID。范围：0~ 最大 A0 数 -1
AoData	F32	OUT	指向返回的模拟量电压值的指针

Acm_DaqAoGetCurrData

函数原型	U32 Acm_DaqAoGetCurrData (HAND DeviceHandle, U16 AoChannel, PF32 AoData)		
说明	获取模拟量输出电流值		
返回值	错误代码		
注解	通过 ChannelID 获取 AO 电流值。 在使用该 API 获取值之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 当 AO 通道通过 CFG_CH_DaqAoRange 设置为电流电平，用户可以获取输出电流值；当 AO 通道设置为电压电平时调用此 API 时将发生错误。 AO 值会受 CFG_CH_DaqAoRange 影响。请先确认属性值是否正确。（请参考属性章节） 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile 。（若用户使用默认的映射关系，则无需载入该 API） 函数原型参数		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	由 Acm_DevOpen 产生的设备句柄
AoChannel	U16	IN	AO 通道 ID。范围：0~最大 AO 数 -1
AoData	PF32	OUT	指向返回的模拟量电流值的指针

A. 3. 3 计数器

Acm_DaqCntGetCounterData

函数原型	U32 Acm_DaqCntGetCounterData (HAND DeviceHandle, U16 CntChannel, PI32 CounterData)		
说明	获取指定 Counter 通道的计数器数据		
返回值	错误代码		
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）		
函数原型参数			
名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~最大 CNT 数 -1
CounterData	PI32	OUT	返回计数器当前数据

Acm_DaqCntResetCnt

函数原型	U32 Acm_DaqCntResetCnt (HAND DeviceHandle, U16 CntChannel)
说明	重置指定 Counter 通道的计数器数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）
函数原型参数	

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~ 最大 CNT 数 -1

Acm_DaqCntGetLatchData

函数原型	U32 Acm_DaqCntGetLatchData (HAND DeviceHandle, U16 CntChannel, PI32 LatchData)
说明	获取指定 Counter 通道的锁存数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~ 最大 CNT 数 -1
LatchData	PI32	OUT	返回计数器当前锁存数据

Acm_DaqCntResetLatch

函数原型	U32 Acm_DaqCntResetLatch (HAND DeviceHandle, U16 CntChannel)
说明	重置指定 Counter 通道的锁存数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~ 最大 CNT 数 -1

Acm_DaqCntGetCmpData

函数原型	U32 Acm_DaqCntGetCmpData (HAND DeviceHandle, U16 CntChannel, PI32 CmpData)
说明	获取指定 Counter 通道的比较数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~最大 CNT 数-1
CmpData	PI32	OUT	返回计数器当前比较数据

Acm_DaqCntResetCmp

函数原型	U32 Acm_DaqCntResetCmp (HAND DeviceHandle, U16 CntChannel)
说明	重置指定 Counter 通道的比较数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API） 函数原型参数

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID

Acm_DaqCntTriggerCmp

函数原型	U32 Acm_DaqCntTriggerCmp (HAND DeviceHandle, U16 CntChannel)
说明	触发指定 Counter 通道的命令以锁存位置数据
返回值	错误代码
注解	在使用该 API 之前，用户需要通过 Utility 中的映射信息来确认或设置整个网络中的 Port number 和 ChannelID。 注： 使用该 API 之前，需要先调用 Acm_DevLoadMapFile （若用户使用默认的映射关系，则无需载入该 API）

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	PHAND	IN	来自 Acm_DevOpen 的设备句柄
CntChannel	U16	IN	计数器通道 ID。范围：0~最大 CNT 数-1

A. 4 轴

A. 4. 1 系统

Acm_AxOpen

函数原型	U32 Acm_AxOpen (HAND DeviceHandle, U16 PhyAxis, PHAND AxisHandle)
说明	打开指定轴，获取该轴的对象句柄
返回值	错误代码
注解	进行任何轴操作之前，应首先调用该 API PCI-1203 物理轴编号为：0、1、2...31

函数原型参数

名称	类型	IN 或 OUT	说明
DeviceHandle	HAND	IN	由 Acm_DevOpen 产生的设备句柄
PhyAxis	U16	IN	物理轴编号
AxisHandle	PHAND	OUT	返回一个指针，指向轴句柄

Acm_AxClose

函数原型	U32 Acm_AxClose (PHAND AxisHandle)
说明	关闭已经打开的轴
返回值	错误代码
注解	调用该 API 之后，将不能再使用轴句柄

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	指针指向轴句柄

Acm_AxResetError

函数原型	U32 Acm_AxResetError (HAND AxisHandle)
说明	复位轴的状态。如果轴处于 “ErrorStop” 状态，则调用该函数后状态将变为 “Ready”
返回值	错误代码
注解	

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

A. 4. 2 运动 I/O

Acm_AxSetSvOn

函数原型	U32 Acm_AxSetSvOn (HAND AxisHandle, U32 OnOff)
说明	打开 / 关闭伺服驱动器
返回值	错误代码
注解	

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
OnOff	U32	IN	设置 SVON 信号的动作 0: 关闭 1: 打开

Acm_AxGetMotionIO

函数原型	U32 Acm_AxGetMotionIO (HAND AxisHandle, PU32 Status)
说明	获取轴的运动 I/O 状态
返回值	错误代码
注解	

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Status	PU32	OUT	位 说明 0: RDY----- RDY 针脚输入 1: ALM ----- 报警信号输入 2: LMT+ ----- 限位开关 + 3: LMT- ----- 限位开关 - 4: ORG----- 原始开关 5: DIR ----- DIR 输出 6: EMG ----- 紧急信号输入 7: PCS ----- PCS 信号输入（不支持） 8: ERC ---- 输出偏转计数器清除信号至伺服电机驱动（OUT7） 9: EZ ----- 编码器 Z 信号 10: CLR ----- 外部输入至清除位置计数器（PCI-1245/1245V/1245E/1265 不支持） 11: LTC ----- 锁存信号输入 12: SD ----- 减速信号输入（不支持） 13: INP ----- 到位信号输入 14: SVON ----- 伺服开启（OUT6） 15: ALRM ----- 报警复位输出状态 16: SLMT+ ----- 软件限位 + 17: SLMT- ----- 软件限位 - 18: CMP----- 比较信号（OUT5） 19: CAMDO ----- 凸轮区间 DO（OUT4）

A. 4. 3 运动状态

Acm_AxGetMotionStatus

函数原型	U32 Acm_AxGetMotionStatus (HAND AxisHandle, PU32 Status)
说明	获取轴的当前运动状态
返回值	错误代码

注解

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Status	PU32	OUT	位 说明 0: Stop ----- 停止 1: Res1 ----- 保留 2: WaitERC----- 等待 ERC 完成 3: Res2 ----- 保留 4: CorrectBksh ----- 背隙补偿 5: Res3 ----- 保留 6: InFA ----- 处于特定速度中 = FA 7: InFL ----- 处于低速中 = FL 8: InACC ----- 加速中 9: InFH ----- 处于最大速度中 = FH 10: InDEC ----- 减速中 11: WaitINP----- 到位等待

Acm_AxGetState

函数原型	U32 Acm_AxGetState (HAND AxisHandle, PU16 State)
说明	获取轴的当前状态

返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Status	PU16	IN	值 说明 0: STA_AxDisable -- 轴被禁用，用户可打开并激活 1: STA_AxReady ---- 轴已准备就绪，等待新的命令 2: STA_Stopping ---- 轴停止 3: STA_AxErrorStop --- 出现错误，轴停止 4: STA_AxHoming ---- 轴正在执行返回原点运动 5: STA_AxPtpMotion ---- 轴正在执行 PTP 运动 6: STA_AxContiMotion ---- 轴正在执行连续运动 7: STA_AxSyncMotion --- 轴在一个群组中，群组正在执行插补运动；或轴是一个从轴，正在执行 E-cam/E-gear/Gantry 运动。 8: STA_AX_EXT_JOG -- 轴由外部信号控制。当外部信号激活时，轴将执行 JOG 模式运动。 9: STA_AX_EXT_MPG --- 轴由外部信号控制。当外部信号激活时，轴将执行 MPG 模式运动

Acm_AxGetMachPosition

函数原型	U32 Acm_AxGetMachPosition(HAND AxisHandle, PF64 Position)		
说明	获取指定轴的当前机台位置		
返回值	错误代码		
注解	当举例：由于背隙机台原因，用户要启用背隙补偿，设补偿值为 10pulse。开始时机台坐标、电机坐标起始位置为 0，用户下 ABS(1000)，则 CMD POSITION=1000，机台也到达 1000 这个位置；再反向运动，如用户下 ABS(0)，因为反向需要补偿，所以电机会送出 1010 个 PULSE，此时电机位置为 -10，CMD POSITION=-10，机台位置为 0（虽然送出了 1010 个 PULSE，但因为有背隙机台只走了 1000）。		
	目前用户想知道机台位置 Mach Position = 0。但用 Acm_AxGetCmdPosition 读到数据为 CMD POSITION=-10 不能满足用户需求，故增加此 API。 不支持背隙补偿或未启用背隙补偿情况下，此 API 与 Acm_AxGetCmdPosition API 获取位置数据一致。		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	PF64	IN	返回理论机台位置

A. 4. 4 速度运动

Acm_AxMoveVel

函数原型	U32 Acm_AxMoveVel (HAND AxisHandle, U16 Direction)
说明	命令轴按照规定速度进行没有终点的运动
返回值	错误代码
注解	速度曲线由以下属性决定：PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk
函数原型参数	

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Direction	U16	IN	方向： 0：正方向 1：负方向

Acm_AxChangeVel

函数原型	U32 Acm_AxChangeVel (HAND AxisHandle, F64 NewVelocity)
说明	当轴在运动过程中，命令轴改变速度
返回值	错误代码
注解	速度曲线由以下属性决定：PAR_AxVelLow、NewVelocity、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk。NewVelocity 的范围：0 ~ CFG_AxMaxVel。若此命令成功下达，在下次运动之前若未重新设定速度，则 NewVelocity 会用到下次运动中

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
NewVelocity	F64	IN	新速度（单位 = PPU/s）

Acm_AxGetCmdVelocity

函数原型	U32 Acm_AxGetCmdVelocity (HAND AxisHandle, PF64 Velocity)
说明	获取指定轴的当前理论速度
返回值	错误代码
注解	

函数原型参数

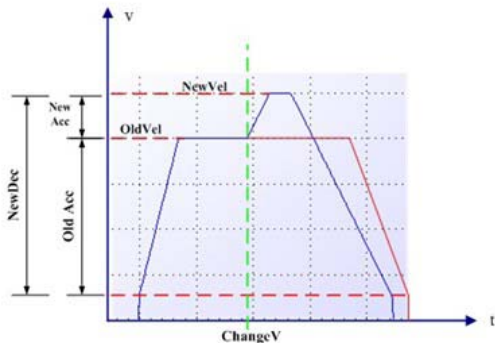
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Velocity	PF64	OUT	返回理论速度（单位 = PPU/s）

Acm_AxChangeVelEx

函数原型	U32 Acm_AxChangeVelEx (HAND AxisHandle, F64 NewVelocity, F64 NewAcc, F64 NewDec)
说明	在运动的过程中可同时改变速度，加速度和减速度
返回值	错误代码

NewVelocity 不能超过通过 CFG_AxMaxVel 设定的最大值，NewAcc 不能超过 CFG_AxMaxAcc 设定的最大加速度，NewDec 不能超过 CFG_AxMaxDec 设定的最大减速度。若 NewAcc 或 NewDec 为 0，则使用上一次设定的加速度或减速度值。若此命令成功下达，在下次运动之前若未重新设定速度，NewVelocity 会作用到下次运动中

注解



函数原型参数

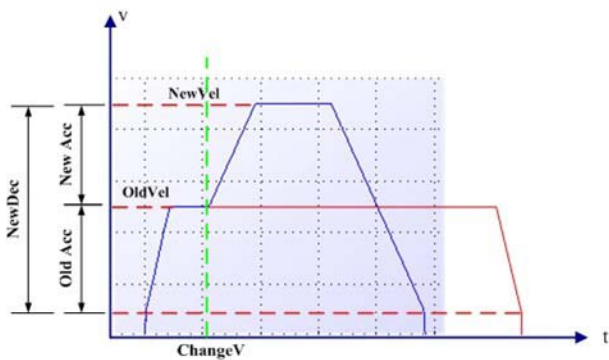
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
NewVelocity	F64	IN	新的速度值，单位：PPU/s
NewAcc	F64	IN	新的加速度值，单位：PPU/s ²
NewDec	F64	IN	新的减速度值，单位：PPU/s ²

Acm_AxChangeVelExByRate

函数原型	U32 Acm_AxChangeVelExByRate (HAND AxisHandle, U32 Rate, F64 NewAcc, F64 NewDec)
说明	在运动的过程中可根据比率改变运行速度，同时可改变加速度和减速度
返回值	错误代码

NewVel = OldVel*Rate*0.01。根据 Rate 计算出来的 NewVel。不能超过通过 CFG_AxMaxVel 设定的最大值，NewAcc 不能超过 CFG_AxMaxAcc 设定的最大加速度，NewDec 不能超过 CFG_AxMaxDec 设定的最大减速度。若 NewAcc 或 NewDec 为 0，则使用上一次设定的加速度或减速度值。新的速度，NewAcc 和 NewDec 仅对当前运动有效

注解



函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Velocity	U32	IN	速度改变百分值。NewVel = OldVel * Rate * 0.01
NewAcc	F64	IN	新的加速度值，单位：PPU/s ²
NewDec	F64	IN	新的减速度值，单位：PPU/s ²

Acm_AxChangeVelByRate

函数原型	U32 Acm_AxChangeVelByRate (HAND AxisHandle, U32 Rate)		
说明	按照设定的比例改变当前正在执行的动作的运行速度		
返回值	错误代码		
注解	新速度 = 之前的运行速度 * Rate * 0.01。Rate 须大于 0 且小于 CFG_AxMaxVel 与之前的速度的比值。新速度仅对当前运动有效		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Rate	U32	IN	速度改变百分比值

A. 4. 5 点到点运动

Acm_AxMoveRel

函数原型	U32 Acm_AxMoveRel (HAND AxisHandle, F64 Distance)		
说明	开始单轴的相对点到点运动		
返回值	错误代码		
注解	速度曲线由以下属性决定：PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk。 Distance 的范围：-2147483647/PPU ~ 2147483647/PPU		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Distance	F64	IN	相对距离 （单位 = PPU）

Acm_AxMoveAbs

函数原型	U32 Acm_AxMoveAbs (HAND AxisHandle, F64 Position)		
说明	开始单轴的绝对点到点运动		
返回值	错误代码		
注解	速度曲线由以下属性决定：PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk Distance 的范围：-2147483647/PPU ~ 2147483647/PPU		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	绝对位置 （单位 = PPU）

Acm_AxChangePos

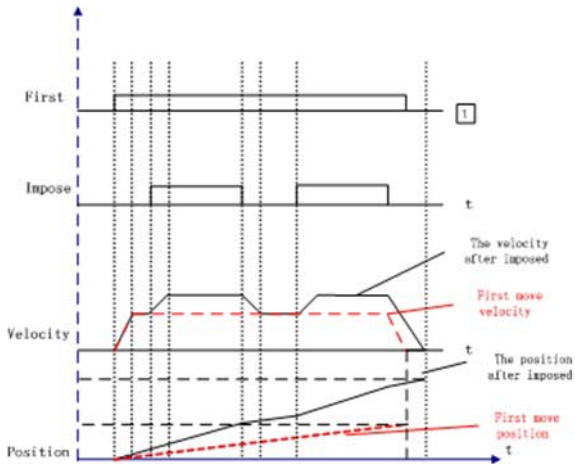
函数原型	U32 Acm_AxChangePos (HAND AxisHandle, F64 NewDistance)		
说明	当轴处于点到点运动时，命令轴改变终点位置		
返回值	错误代码		
注解	Distance 的范围： -2147483647/PPU ~ 2147483647/PPU		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
NewDistance	F64	IN	新的相对距离 （单位 = PPU）

Acm_AxMoveImpose

函数原型	U32 Acm_AxMoveImpose (HAND AxisHandle, F64 Position, F64 NewVel)
说明	在当前运动上叠加新的运动
返回值	错误代码

该函数仅支持该新叠加的运动的曲线为T型曲线。
运动停止的终止位置将为原始位置加 / 减 Position，且驱动速度也将改变整个速度曲线由该运动的 NewVel 决定，由原始运动的 PAR_AxVelLow、PAR_AxVelHigh、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk 决定。
Position + 原始位置的范围：-2147483647/ PPU ~ 2147483647/ PPU，且 NewVel + 原始 FH 的范围：0 ~ CFG_AxMaxVel。
如下图所示：

注解



注：不能在叠加运动上叠加新的运动！

函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	新运动的相对位置（单位 = PPU）
NewVel	F64	IN	该叠加运动的速度（单位 = PPU/s）

A. 4. 6 同步起停运动

Acm_AxSimStartSuspendAbs

函数原型	U32 Acm_AxSimStartSuspendAbs (HAND AxisHandle, F64 EndPoint)
说明	设定轴为等待状态。当轴收到启动信号开始运动时，轴将开始点到点绝对运动，直至到达指定终止位置
返回值	错误代码

注解
如果不止一个轴想要进行同步操作，那么每个轴都需要调用该函数。
EndPoint 的范围：-2147483647/ PPU ~ 2147483647/ PPU

函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
EndPoint	F64	IN	运动到的绝对位置（单位 = PPU）

Acm_AxSimStartSuspendRel

函数原型	U32 Acm_AxSimStartSuspendRel (HAND AxisHandle, F64 Distance)		
说明	设定轴为等待状态。当轴收到启动信号开始运动时，轴将开始点到点相对运动，直至到达指定终止位置		
返回值	错误代码		
注解	如果不止一个轴想要进行同步操作，那么每个轴都需要调用该函数。 EndPoint 的范围：-2147483647/ PPU ~ 2147483647/ PPU		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
EndPoint	F64	IN	运动到的相对位置（单位 = PPU）

Acm_AxSimStartSuspendVel

函数原型	U32 Acm_AxSimStartSuspendVel (HAND AxisHandle, U16 DriDir)		
说明	设定轴为等待状态。当轴收到启动信号开始运动时，轴将开始连续运动		
返回值	错误代码		
注解	如果不止一个轴想要进行同步操作，那么每个轴都需要调用该函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
DriDir	U16	IN	方向： 0: 正方向 1: 负方向

Acm_AxSimStart

函数原型	U32 Acm_AxSimStart (HAND AxisHandle)		
说明	发出同步启动信号，以启动所有等待启动触发的轴		
返回值	错误代码		
注解	如果不止一个轴等待启动触发，那么在调用该 API 之前，用户应通过 CFG_AxSimStartSource 设置同时开始 / 停止的模式		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm AxOpen 的轴句柄

Acm_AxSimStop

函数原型	U32 Acm_AxSimStop (HAND AxisHandle)		
说明	发出同步停止信号，以停止所有等待停止触发的轴		
返回值	错误代码		
注解	进行同步操作时，如果使用 STA 信号启动模式，则用户能够在任一轴上进行该操作以停止所有轴。否则，每个同时轴都需要调用该 API 才能停止同时运动。有关同步启停模式的详细信息，请参考 CFG_AxSimStartSource		
函数原型参数			
名称	类型	IN 或 OUT	说明

AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
------------	------	----	--------------------

A. 4. 7 回原点

Acm_AxHome

函数原型	U32 Acm_AxHome (HAND AxisHandle, U32 HomeMode, U32 Dir)
说明	命令轴开始回原点运动
返回值	错误代码
注解	详见回原点章节
函数原型参数	

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
HomeMode	U32	IN	HomeMode: 0: MODE1_Abs 1: MODE2_Lmt 2: MODE3_Ref 3: MODE4_Abs_Ref 4: MODE5_Abs_NegRef 5: MODE6_Lmt_Ref 6: MODE7_AbsSearch 7: MODE8_LmtSearch 8: MODE9_AbsSearch_Ref 9: MODE10_AbsSearch_NegRef 10: MODE11_LmtSearch_Ref 11: MODE12_AbsSearchReFind 12: MODE13_LmtSearchReFind 13: MODE14_AbsSearchReFind_Ref 14: MODE15_AbsSearchReFind_NegRef 15: MODE16_LmtSearchReFind_Ref 101~137: CiA402_MODE1 ~ CiA402_MODE37
Dir	U32	IN	0: 正方向 1: 负方向

Acm_AxMoveHome

函数原型	U32 Acm_AxMoveHome (HAND AxisHandle, U32 HomeMode, U32 DirMode)
说明	回原点运动
返回值	错误代码
注解	U32 Result; Result = Acm_AxMoveHome (m_Axishand[0], 1, 0);
函数原型参数	

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	轴句柄
HomeMode	U32	IN	回原点模式
DirMode	U32	IN	回原点方向

注! 速度设定使用 *PAR_AxHomeVelLow*, *PAR_AxHomeVelHigh*, *PAR_AxHomeAcc*, *PAR_AxHomeDec*, *PAR_AxHomeJerK*.



A. 4.8 位置 / 计数器控制

Acm_AxSetCmdPosition

函数原型	U32 Acm_AxSetCmdPosition (HAND AxisHandle, F64 Position)		
说明	设置指定轴的理论位置		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	新的理论位置 （单位：PPU）

Acm_AxGetCmdPosition

函数原型	U32 Acm_AxGetCmdPosition (HAND AxisHandle, PF64 Position)		
说明	获取指定轴的当前理论位置		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	返回理论位置 （单位：PPU）

Acm_AxSetActualPosition

函数原型	U32 Acm_AxSetActualPosition (HAND AxisHandle, F64 Position)		
说明	设置指定轴的实际位置		
返回值	错误代码		
注解	仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	新的实际位置 （单位：PPU）

Acm_AxGetActualPosition

函数原型	U32 Acm_AxGetActualPosition (HAND AxisHandle, PF64 Position)		
说明	获取指定轴的当前实际位置		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Position	PF64	IN	返回实际位置 （单位：PPU）

A. 4.9 比较

Acm_AxSetCmpData

函数原型	U32 Acm_AxSetCmpData (HAND AxisHandle, F64 CmpPosition)		
说明	设置指定轴的比较数据		
返回值	错误代码		
注解	详见 Compare 章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
CmpPosition	F64	IN	需要比较的数据 （单位：PPU）

Acm_AxSetCmpTable

函数原型	U32 Acm_AxSetCmpTable (HAND AxisHandle, PF64 TableArray, I32 ArrayCount)		
说明	设置指定轴的比较数据列表		
返回值	错误代码		
注解	详见 Compare 章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
TableArray	PF64	IN	比较数据表（单位：PPU）
ArrayCount	I32	IN	表中的比较数据个数

Acm_AxSetCmpAuto

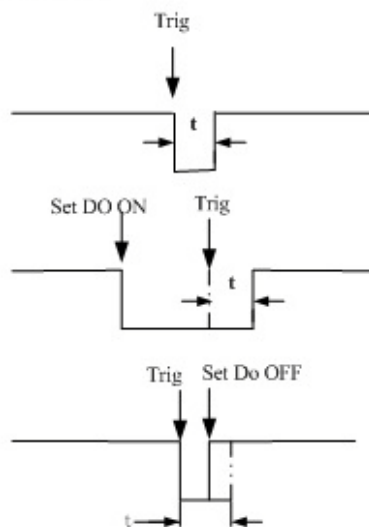
函数原型	U32 Acm_AxSetCmpAuto (HAND AxisHandle, F64 Start, F64 End, F64 Interval)		
说明	设置指定轴的线性比较数据		
返回值	错误代码		
注解	详见 Compare 章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Start	F64	IN	首个比较数据 （单位：PPU）
End	F64	IN	最后比较的数据 （单位：PPU）
Interval	F64	IN	比较间隔 （单位：PPU）

Acm_AxSetCmpDO

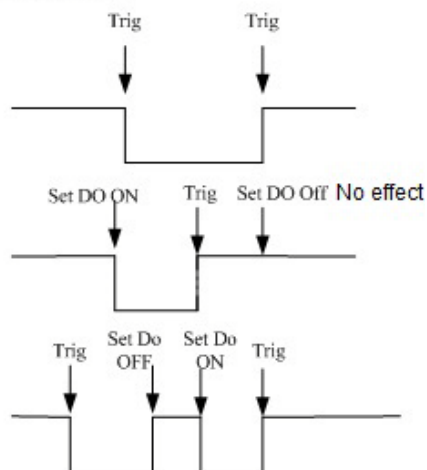
函数原型	U32 Acm_AxSetCmpDO (HAND AxisHandle, U32OFForON)		
说明	设置指定轴的比较 DO 输出		
返回值	错误代码		

当 Compare Pulse Mode 为 Pulse 时，调用该 API，ON 会触发 OUT5 输出一个脉冲；当 Compare Pulse Mode 为 Toggle 时，调用该 API，ON 会触发 OUT5 输出 active Logic, OFF 会触发 OUT5 输出 no active Logic。

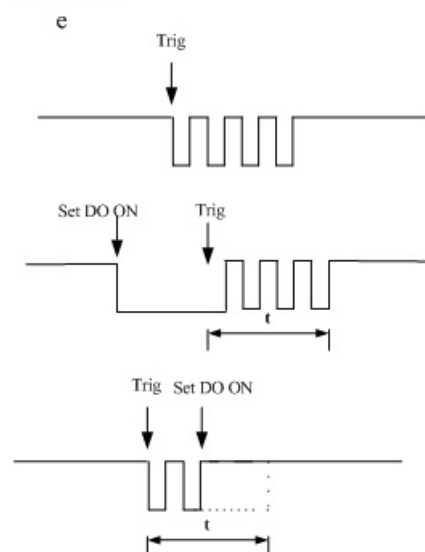
PulseMode



ToggleMode



PWMMod



注解

仅特定 EtherCAT 从站支持此函数。

函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
OFFForON	U32	IN	触发 Compare 的 D0 输出。 0: OFF ; 1: ON。

Acm_AxGetCmpData

函数原型	U32 Acm_AxGetCmpData (HAND AxisHandle, PF64 CmpPosition)		
说明	获取比较器中的当前比较数据		
返回值	错误代码		
注解	详见 Compare 章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
CmpPosition	PF64	OUT	比较数据 （单位：PPU）

Acm_AxChangeCmpIndex

函数原型	U32 Acm_AxChangeCmpIndex (HAND AxisHandle, U32 CmpIndex)		
说明	改变 Compare 位置比较索引		
返回值	错误代码		
注解	当需要进行大笔数据比较时，先调用 Acm_AxSetCmpTable 设置比较数据，以后每次比较只需改变 CmpIndex 数值即可从开始位置重新比较，而无需每次调用 Acm_AxSetCmpTable 花费时间写入比较数据。 1245/65 分位 1319 版本、1285 分位 1320 版本只支持 CmpIndex 设置为 0；1245/65 分位 1320 版本、1285 分位 1321 版本支持 CmpIndex 设置为 0~(Cmp 索引最大值-1)		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
CmpIndex	U32	OUT	Compare 位置比较索引 范围：0~(Cmp 索引最大值-1)

A. 4. 10 锁存

Acm_AxGetLatchData

函数原型	U32 Acm_AxGetLatchData (HAND AxisHandle, U32 PositionNo, F64 Position)		
说明	触发锁存后获取设备中的锁存数据		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
PositionNo	U32	IN	0: 理论位置 1: 实际位置

Position	PF64	OUT	锁存数据（单位 = PPU）
----------	------	-----	----------------

Acm_AxTriggerLatch

函数原型	U32 Acm_AxTriggerLatch (HAND AxisHandle)		
说明	触发命令以锁存位置数据		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxResetLatch

函数原型	U32 Acm_AxResetLatch (HAND AxisHandle)		
说明	清除设备中的锁存数据和锁存标记		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxGetLatchFlag

函数原型	U32 Acm_AxGetLatchFlag (HAND AxisHandle, PU8 LatchFlag)		
说明	获取设备中的锁存标记		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
LatchFlag	PU8	OUT	标记数据 1: 数据锁存 0: 数据未锁存

A. 4. 11 连续锁存

Acm_AxReadLatchBuffer

函数原型	U32 Acm_AxReadLatchBuffer (HAND AxisHandle, PF64 LatchDataArray, PU32 DataCnt)		
说明	从 Latch Buffer 中读取指定数量的资料		
返回值	错误代码		
注解	详见连续锁存章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄

LatchdataArray	PF64	OUT	连续锁存数据阵列
DataCnt	PU32	IN	阵列中的元素个数

Acm_AxGetLatchBufferStatus

函数原型	U32 Acm_AxGetLatchBufferStatus (HAND AxisHandle, PU32 RemainCnt, PU32 SpaceCnt)		
说明	获取 Latch Buffer 的状态		
返回值	错误代码		
注解	详见连续锁存章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
RemainCnt	PU32	OUT	Latch Buffer 中记录的笔数
SpaceCnt	PU32	OUT	Latch Buffer 中所剩空间数

Acm_AxResetLatchBuffer

函数原型	U32 Acm_AxResetLatchBuffer (HAND AxisHandle)		
说明	清空 Latch Buffer		
返回值	错误代码		
注解	详见连续锁存章节 仅特定 EtherCAT 从站支持此函数		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄

A. 4. 12 外部驱动

Acm_AxSetExtDrive

函数原型	U32 Acm_AxSetExtDrive (HAND AxisHandle, U16 ExtDrvMode)		
说明	启用或禁用外部驱动模式		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
ExtDrvMode	U16	IN	0: 禁用 （停止命令） 2: MPG 模式 3: JOG 步进模式

Acm_AxJog

函数原型	U32 Acm_AxJog (HAND AxisHandle, U16 Direction)		
说明	指定轴执行 Jog 运动		
返回值	错误代码		

注解	调用该 API 前，必须先调用 Acm_AxSetExtDrive 设置外部驱动为 Jog 模式		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
Direction	U16	IN	Jog 运动的方向，0：正向，1：负向

A. 4. 13 凸轮 / 齿轮

Acm_AxCamInAx

函数原型	U32 Acm_AxCamInAx (HAND AxisHandle, HAND MasAxisHandle, F64 MasterOffset, F64 SlaveOffset, F64 MasterScaling, F64 SlaveScaling, U32 CamTableID, U32 RefSrc)		
说明	该函数使用 CAM 表开始从 （其它）轴和主 （首个）轴之间的 CAM 同步。Camming 是一个表内完成 （二维 - 描述主和从位置）。表应严格为单调上升或下降，随着主轴同时向相反和相同方向发展		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	由 Acm_AxOpen 生成的轴句柄，应为从轴句柄
MasAxisHandle	HAND	IN	由 Acm_AxOpen 生成的轴句柄，应为主轴句柄
MasterOffset	F64	IN	主轴方向的坐标偏移值
SlaveOffset	F64	IN	从轴方向的坐标偏移值
MasterScaling	F64	IN	主轴坐标中 CAM 的比例因子。整个主轴坐标乘以该比例因子，结果应大于 0
SlaveScaling	F64	IN	从轴坐标中 CAM 的比例因子。整个从轴坐标乘以该比例因子，结果应大于 0
CamTableID	U32	IN	CAM 表的标识符。由 Acm_DevDownloadCAMTable 分配。PCI-1203 保留了 2 个 CAM 表。因此 ID 为 0 和 1。
RefSrc	U32	IN	CAM 表参考位置 0：理论位置 1：实际位置

Acm_AxGearInAx

函数原型	U32 Acm_AxGearInAx (HAND AxisHandle, HAND MasAxisHandle, I32 Numerator, I32 Denominator, U32 RefSrc, U32 Absolute)		
说明	该函数按照比例开始从（其它）轴和主（首个）轴之间的 Gear 同步		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	由 Acm_AxOpen 生成的轴句柄，应为从轴句柄
MasAxisHandle	HAND	IN	由 Acm_AxOpen 生成的轴句柄，应为主轴句柄
Numerator	I32	IN	齿轮比的分子
Denominator	I32	IN	齿轮比分母
RefSrc	U32	IN	从轴参考主轴的理论位置（0）或实际位置（1）

Absolute	U32	IN	同步关系是相对的还是绝对的 0: 相对 1: 绝对
----------	-----	----	---------------------------------

Acm_AxPhaseAx

函数原型	U32 Acm_AxPhaseAx (HAND AxisHandle, F64 Acc, F64 Dec, F64 PhaseSpeed, F64 PhaseDist)		
说明	在电子凸轮或电子齿轮过程中使从轴进行相位超前或落后运动		
返回值	错误代码		
注解	在电子凸轮或电子齿轮过程中，通过此 API 可使从轴超前 / 落后原先的运动，当 PhaseDist>0，则进行相位超前运动，当 PhaseDist<0，则进行相位落后运动。若剩余 Pulse 不足，则从轴无法达到指定的相位。且当相位超前 / 落后运动未结束之前，再次下达此命令，则返回错误。因浮点数计算误差，从轴通过属性 CFG_AxMaxAcc/CFG_AxMaxDec 设定的最大加速度 / 最大减速度的值须比 Acc/Dec 至少大 100,000。		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	由 Acm_AxOpen 生成的轴句柄，应为从轴句柄
Acc	HAND	IN	相位超前 / 落后运动的加速度，单位：PPU/s ²
Dec	F64	IN	相位超前 / 落后运动的减速度，单位：PPU/s ²
PhaseSpeed	F64	IN	相位超前 / 落后运动的速度，单位：PPU/s
PhaseDist	F64	IN	相位超前 / 落后运动的距离，单位：PPU

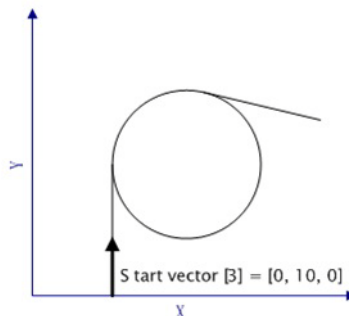
A. 4. 14龙门 / 切线跟随

Acm_AxTangentInGp

函数原型	U32 Acm_AxTangentInGp (HAND AxisHandle, HAND MasGroupHandle, PI16 StartVectorArray, U8 Working_plane, I16 Direction)		
说明	命令轴按照与群组路径切线相同的方向运动		
返回值	错误代码		

如果该函数调用成功，轴将为同步状态，且按照与群组路径切线相同的方向跟随群组运动。轴和群组不能为复位理论 / 实际位置。
通过 Acm_AxStopDec/Acm_AxStopEmg 可终止轴的同步状态，且将变为 SteadBy 状态。跟随轴不能为群组中的一个轴。
StartVectorArray 为初始向量，也是轴的起始方向。比如，
StartVectorArray[3] = {0, 10, 0}, Working plane = 0: XY 平面

注解



起始向量必须尽可能与群组运动的起始方向接近，否则可能出现运动加速大于设备最大加速的错误。如果两个连接路径之间的角度过大，也会发生错误，因此用户须注意路径之间的角度。计算最大角度偏差的公式如下：

比如设置单圈脉冲数范围（CFG_AxModuleRange）：3600 个脉冲。

最大加速（CFG_AxMaxAcc）： 10^7 。

斜率转换的最大角度：

$$10^7 \times 10^{-6} \times 360^\circ / 3600 = 1$$

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
MasGroupHandle	HAND	IN	来自 Acm_GpAddAxis 的群组句柄
StartVectorArray	PI16	IN	必须为三维
Working_plane	U8	IN	0: XY; 1: YZ; 2: XZ
Direction	I16	IN	相同: 0; 相反: 1

Acm_AxGantryInAx

函数原型	U32 Acm_AxGantryInAx (HAND AxisHandle, HAND MasAxisHandle, I16 RefMasterSrc, I16 Direction)
说明	命令两个轴进行 E-Gantry 运动
返回值	错误代码

注解

从轴将于主轴同步运动。通过调用 Acm_AxStopDec 或 Acm_AxStopEmg 可终止主轴和从轴的关系，且轴将变为 SteadBy 状态。

Gantry 有如下几种限制：

1. 不能给从轴设置任何运行命令；
 2. 从轴不能添加到任何群组中；
 3. 如果轴已经为群组中的一个轴，那么该轴不能作为 Gantry 的从轴。
- 如果复位主轴的理论 / 实际位置，则从轴的命令 / 实际位置也应复位至相同值

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
MasGroupHandle	HAND	IN	来自 Acm_GpAddAxis 的群组句柄

RefMasterSrc	I16	IN	参考源： 0：理论位置； 1：实际位置（预留）
Direction	I16	IN	和主轴的方向 0：相同 1：相反

A. 4. 15停止运动

Acm_AxStopDec

函数原型	U32 Acm_AxStopDec (HAND AxisHandle)		
说明	命令轴减速停止		
返回值	错误代码		
注解	如果轴处于同步驱动模式，如 E-cam/E-gear/Tangent 运动中的从轴，该 API 能够用于停止同步关系		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxStopEmg

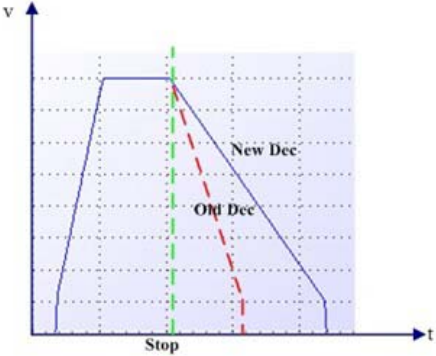
函数原型	U32 Acm_AxStopEmg (HAND AxisHandle)		
说明	命令轴立刻停止（无减速）		
返回值	错误代码		
注解	如果轴处于同步驱动模式，如 E-cam/E-gear/Tangent 运动中的从轴，该 API 能够用于停止同步关系		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxStopDecEx

函数原型	U32 Acm_AxStopDecEx (HAND AxisHandle, F64 NewDec)		
说明	下达停止命令时可指定减速度		
返回值	错误代码		

如果减速停止命令下达之后，剩余的脉冲量不足以支持指定的 NewDec，则会发生断尾现象

注解



函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

NewDec	F64	IN	减速停止时的减速度，单位：PPU/s ²
--------	-----	----	---------------------------------

A. 4. 16PT/PVT 运动

Acm_AxResetPTData

函数原型	U32 Acm_AxResetPTData(HAND AxisHandle)		
说明	清空 PT 缓存		
返回值	错误代码		
注解	此函数需要在写入新数据之前执行		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxAddPTData

函数原型	U32 Acm_AxAddPTData(HAND AxisHandle, F64 Position, F64 Time)		
说明	将一笔 PT 资料添加至 PT 缓存		
返回值	错误代码		
注解	位置和时间值必须从 0 开始。最少需要 2 笔资料（位置和时间）来形成梯形速度曲线。		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Position	F64	IN	位置。单位：PPU
Time	F64	IN	时间。单位：ms

Acm_AxStartPT

函数原型	U32 Acm_AxStartPT (HAND AxisHandle, U8 Repeat)		
说明	开始 PT 运动		
返回值	错误代码		
注解	在执行 Acm_AxStartPT 之前最少需要写入 2 笔以上的数据到 PT 缓存。		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Repeat	U8	IN	重复次数。(0 ~ 255)

Acm_AxStartAllPT

函数原型	U32 Acm_AxStartAllPT (PHAND AxisHandle, U8 Repeat, U32 ArrayElements)
说明	指定多轴开始 PT 运动
返回值	错误代码
注解	同时启动多个轴进行 PT 运动。将 PT 数据加载到每个轴后，用户可以调用此功能，指定轴中将同时开始运行。在执行 Acm_AxStartAllPT 之前最少需要写入 2 笔以上的数据到 PT 缓存。
函数原型参数	

名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Repeat	U8	IN	重复次数。(0 ~ 255)
ArrayElements	U32	IN	需要同时开始移动的轴的个数

Acm_AxCheckPTBuffer

函数原型	U32 Acm_AxCheckPTBuffer(HAND AxisHandle, PU16Freespace)
说明	获取 PT 缓存中剩余空间的个数
返回值	错误代码
注解	PT 缓存提供的最大数量为 512。

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Freespace	PU16	OUT	剩余空间的个数

Acm_AxResetPVTTable

函数原型	U32 Acm_AxResetPVTTable(HAND AxisHandle)
说明	重置 PVT 缓存
返回值	错误代码
注解	PVT 表需要在新表加载到缓存之前重置

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_AxLoadPVTTable

函数原型	U32 Acm_AxLoadPVTTable(HAND AxisHandle, PF64 Position, PF64 Velocity, PF64 Time, U32 ArrayElements)
说明	加载 PVT 表到 PVT 缓存。PVT 缓存提供的最大数量为 42
返回值	错误代码
注解	位置和时间值必须从 0 开始。最少需要 2 笔资料来形成梯形速度曲线。

函数原型参数

名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Position	PF64	IN	位置信息。单位：PPU
Velocity	PF64	IN	速度信息。单位：PPU/ms
Time	PF64	IN	时间信息。单位：ms
ArrayElements	U32	IN	信息个数

Acm_AxStartPVT

函数原型	U32 Acm_AxStartPVT(HAND AxisHandle, U8Repeat)
说明	开始 PVT 运动
返回值	错误代码

注解	在执行 PVT 运动之前最少需要写入 2 笔以上的数据。		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	PHAND	IN	来自 Acm_AxOpen 的轴句柄
Repeat	U8	IN	重复次数。（0 ~ 255）

A. 5 群组

A. 5.1 系统

Acm_GpAddAxis

函数原型	U32 Acm_GpAddAxis (PHAND GpHandle, HAND AxHandle)		
说明	添加一个轴到指定群组		
返回值	错误代码		
注解	如果 GpHandle 指向 NULL，驱动会创建一个新的群组并将轴添加至该群组。		
	如果 GpHandle 指向一个有效群组句柄，驱动将只把群组添加到群组中。PCI-1203 最多有 6 个群组，一个群组最多不超过 8 个轴。		
	相同的轴不能添加到不同的群组中。		
	群组中的主轴为行走距离最长的轴。		
	添加第一个轴时，群组的参数会初始化， 如 CFG_GpPPU、PAR_GpVelLow、PAR_GpVelHigh、PAR_GpAcc、PAR_GPDec 和 PAR_GpJerk		
函数原型参数			
名称	类型	IN 或 OUT	说明
GpHandle	PHAND	IN/OUT	指针指向群组句柄（Null 或无）
AxHandle	HAND	IN	来自 Acm AxOpen 的轴句柄

Acm_GpRemAxis

函数原型	U32 Acm_GpRemAxis (HAND GpHandle, HAND AxHandle)		
说明	从指定群组中移除一个轴		
返回值	错误代码		
注解	调用 Acm_GpRemAxis 之后，群组中没有轴，GpHandle 仍可使用。用户可使用该群组句柄添加其它轴。但是，如果用户调用 Acm_GpClose 关闭该群组句柄，群组句柄不能再次使用		
函数原型参数			
名称	类型	IN 或 OUT	说明
GpHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
AxHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

Acm_GpClose

函数原型	U32 Acm_GpClose (PHAND pGroupHandle)
说明	移除群组中的所有轴并关闭群组句柄
返回值	错误代码
注解	如果群组数量大于设备的最大群组数，则不能创建新的群组。这时，如果用户想要创建新的群组，则必须关闭一个现有群组
函数原型参数	

名称	类型	IN 或 OUT	说明
pGroupHandle	PHAND	IN	来自 Acm_GpAddaxis 的群组句柄

Acm_GpResetError

函数原型	U32 Acm_GpResetError (HAND GroupHandle)		
说明	复位群组状态		
返回值	错误代码		
注解	如果群组处于 STA_GP_ERROR_STOP 状态，那么调用该函数后，状态将变为 STA_GP_READY		

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

A. 5. 2 运动状态及速度

Acm_GpGetState

函数原型	U32 Acm_GpGetState (HAND GroupHandle, PU16 pState)		
说明	获取群组的当前状态		
返回值	错误代码		
注解	如果群组的一个轴正在执行单轴运动命令，群组状态将不会改变		

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
pState	PU16	OUT	群组状态： 0: STA_GP_DISABLE 1: STA_GP_READY 2: STA_GP_STOPPING 3: STA_GP_ERROR_STOP 4: STA_GP_MOTION 5: STA_GP_AX_MOTION （不支持） 6: STA_GP_MOTION_PATH

Acm_GpGetCmdVel

函数原型	U32 Acm_GpGetCmdVel (HAND GroupHandle, PF64 CmdVel)		
说明	获取群组的当前的速度值		
返回值	错误代码		
注解	通过 API，可以获得群组在执行插补或连续插补动作时当前的速度值		

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CmdVel	PF64	OUT	返回群组的当前速度值，单位：PPU/s。（PPU 为轴 ID 最小的轴的 PPU）

A. 5.3 运动停止

Acm_GpStopDec

函数原型	U32 Acm_GpStopDec (HAND GroupHandle)		
说明	命令该群组中的轴减速停止		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

Acm_GpStopEmg

函数原型	U32 Acm_GpStopEmg(HAND GroupHandle)		
说明	命令该群组中的轴立刻停止（无减速）		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

A. 5.4 插补运动

Acm_GpMoveLinearRel

函数原型	U32 Acm_GpMoveLinearRel(HAND GroupHandle, PF64 DistanceArray, PU32 pArrayElements)		
说明	命令群组执行相对线性插补		
返回值	错误代码		
注解	DistanceArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴的顺序。比如，如果一个群组有两个轴：Y 轴和 U 轴。DistanceArray 中的第一个数据表示 Y 轴的相对距离，第二个数据表示 U 轴的相对距离。DistanceArray 中距离的单位为群组中每个轴的 PPU。		
	线性插补和直接插补的不同之处在于：线性插补的速度被分解为其中各个轴的向量速度，轴将以该速度运动。多数情况中，线性插补应用于以直角组合的轴。而直接插补的线性速度设置为主轴（行走距离最长的轴）的速度，其他轴会与主轴同时启动同时停止。多数情况中，直接插补应用于以斜角组合的轴。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
DistanceArray	PF64	IN	群组中轴的距离阵列，阵列元素的每个值都表示轴的相对位置
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）

Acm_GpMoveLinearAbs

函数原型	U32 Acm_GpMoveLinearAbs (HAND GroupHandle, PF64 PositionArray, PU32 pArrayElements)		
说明	命令群组执行绝对线性插补		

返回值	错误代码		
注解	PositionArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴的顺序。比如，如果一个群组有两个轴：Y 轴和 U 轴。PositionArray 中的第一个数据表示 Y 轴的绝对距离，第二个数据表示 U 轴的绝对距离。		
	PositionArray 中距离的单位为群组中每个轴的 PPU。		
	线性插补和直接插补的不同之处在于：线性插补的速度被分解为其中各个轴的向量速度，轴将以该速度运动。多数情况中，线性插补应用于以直角组合的轴。而直接插补的线性速度设置为主轴（行走距离最长的轴）的速度，其他轴会与主轴同时启动同时停止。多数情况中，直接插补应用于以斜角组合的轴。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
PositionArray	PF64	IN	群组中轴的位置阵列，阵列元素的每个值都表示轴的绝对位置
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）

Acm_GpMoveDirectRel

函数原型	U32 Acm_GpMoveDirectRel (HAND GroupHandle, PF64 DistanceArray, PU32 ArrayElements)		
说明	命令群组执行相对直接线性插补		
返回值	错误代码		
注解	DistanceArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴等的顺序。比如，如果一个群组有两个轴：Y 轴和 U 轴。DistanceArray 中的第一个数据表示 Y 轴的相对距离，第二个数据表示 U 轴的相对距离。DistanceArray 中距离的单位为群组中每个轴的 PPU。 线性插补和直接插补的不同之处在于：线性插补的速度被分解为其中各个轴的向量速度，轴将以该速度运动。多数情况中，线性插补应用于以直角组合的轴。而直接插补的线性速度设置为主轴（距离最长轴）的速度，其他轴与主轴同时启动，同时停止。多数情况中，直接插补应用于以斜角组合的轴。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
DistanceArray	PF64	IN	群组中轴的距离阵列，阵列元素的每个值都表示轴的相对位置
ArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）

Acm_GpMoveDirectAbs

函数原型	U32 Acm_GpMoveDirectAbs (HAND GroupHandle, PF64 PositionArray, PU32 ArrayElements)		
说明	命令群组执行绝对直接线性插补		
返回值	错误代码		

注解	PositionArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴等的顺序。比如，如果一个群组有两个轴：Y 轴和 U 轴。PositionArray 中的第一个数据表示 Y 轴的绝对距离，第二个数据表示 U 轴的绝对距离。PositionArray 中距离的单位为群组中每个轴的 PPU。线性插补和直接插补的不同之处在于：线性插补的速度被分解为其中各个轴的向量速度，轴将以该速度运动。多数情况中，线性插补应用于以直角组合的轴。而直接插补的线性速度设置为主轴（行走距离最长的轴）的速度，其他轴与主轴同时启动，同时停止。多数情况中，直接插补应用于以斜角组合的轴。
----	---

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
PositionArray	PF64	IN	群组中轴的距离阵列，阵列元素的每个值都表示轴的绝对位置
ArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）

Acm_GpMoveCircularRel

函数原型	U32 Acm_GpMoveCircularRel (HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组执行相对 ARC 插补
返回值	错误代码
注解	CenterArray 和 EndArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴等的顺序。比如，如果一个群组有 Y 轴和 U 轴，那么 CenterArray 中的第一个数据表示 Y 轴的圆心距离，第二个数据表示 U 轴的圆心距离。CenterArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心点的相对距离
EndArray	PF64	IN	轴相对于终止点的相对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0: DIR_CW （顺时针） 1: DIR_CCW （逆时针）

Acm_GpMoveCircularAbs

函数原型	U32 Acm_GpMoveCircularAbs (HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组执行绝对 ARC 插补
返回值	错误代码
注解	CenterArray 和 EndArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴的顺序。比如，如果一个群组有 Y 轴和 U 轴，那么 CenterArray 中的第一个数据表示 Y 轴的圆心位置，第二个数据表示 U 轴的圆心位置。CenterArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

CenterArray	PF64	IN	轴相对于圆心点的绝对距离
EndArray	PF64	IN	轴相对于终止点的绝对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveCircularRel_3P

函数原型	U32 Acm_GpMoveCircularRel_3P (HAND GroupHandle, PF64 RefArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组通过三个指定点执行相对 ARC 插补
返回值	错误代码
注解	RefArray 和 EndArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴等的顺序。比如，如果一个群组有 Y 轴和 U 轴，那么 RefArray 中的第一个数据表示 Y 轴的参考距离，第二个数据表示 U 轴的参考距离。 RefArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
RefArray	PF64	IN	参考点的相对距离
EndArray	PF64	IN	终止点的相对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0: DIR_CW（顺时针）； 1: DIR_CCW（逆时针）

Acm_GpMoveCircularAbs_3P

函数原型	U32 Acm_GpMoveCircularAbs_3P (HAND GroupHandle, PF64 RefArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组通过三个指定点执行绝对 ARC 插补
返回值	错误代码
注解	RefArray 和 EndArray 中的数据顺序必须遵循 X 轴、Y 轴、Z 轴和 U 轴等的顺序。比如，如果一个群组有 Y 轴和 U 轴，那么 RefArray 中的第一个数据表示 Y 轴的参考位置，第二个数据表示 U 轴的参考位置。 RefArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。

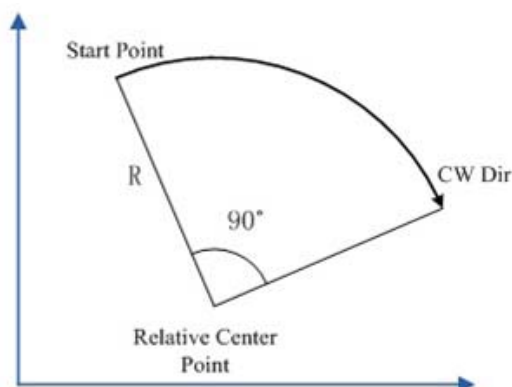
函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
RefArray	PF64	IN	参考点的绝对距离
EndArray	PF64	IN	终点的绝对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveCircularRel_Angle

函数原型	U32 Acm_GpMoveCircularRel_Angle (HAND GroupHandle, PF64 CenterArray, U16 Degree, PU32 ArrayElements, I16 Direction)
说明	按照相对中心坐标、角度和旋转方向进行圆弧插补。
返回值	错误代码

注解



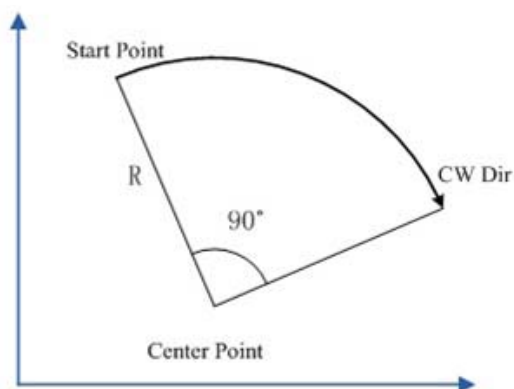
函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的绝对距离
Degree	U16	IN	旋转角度（范围：0~360）
pArrayElements	PU32	IN	轴数
Direction	I16	IN	方向： 0: CW_Dir（顺时针） 1: CCW_Dir（逆时针）

Acm_GpMoveCircularAbs_Angle

函数原型	U32 Acm_GpMoveCircularAbs_Angle (HAND GroupHandle, PF64 CenterArray, U16 Degree, PU32 ArrayElements, I16 Direction)
说明	按照绝对中心坐标、角度和旋转方向进行圆弧插补。
返回值	错误代码

注解



函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	圆心坐标
Degree	U16	IN	旋转角度（范围：0~360）
ArrayElements	PU32	IN	轴数

Direction	I16	IN	方向： 0: CW_Dir（顺时针） 1: CCW_Dir（逆时针）
-----------	-----	----	--

Acm_GpMove3DArcRel

函数原型	U32 Acm_GpMove3DArcRel(HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)		
说明	按照指定的点执行群组相对 3D 圆弧插补。		
返回值	错误代码		
注解	用户可以输入圆心、终点和方向，然后执行相对 3D 圆弧插补。请参考章节 9.3.3.2：3 轴圆弧插补。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的绝对距离
EndArray	PF64	IN	轴相对于终点的绝对距离
pArrayElements	PU32	IN/OUT	数列中的轴数（该值必须与群组中的轴数相同，否则将返回群组中的轴数。）
Direction	I16	IN	方向： 0：DIR_CW（顺时针） 1：DIR_CCW（逆时针）

Acm_GpMove3DArcAbs

函数原型	U32 Acm_GpMove3DArcAbs(HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)		
说明	执行群组绝对圆弧插补。		
返回值	错误代码		
注解	用户可以输入圆心、终点和方向，然后执行绝对 3D 圆弧插补。请参考章节 9.3.3.2：3 轴圆弧插补。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的绝对距离
EndArray	PF64	IN	轴相对于终点的绝对距离
pArrayElements	PU32	IN/OUT	数列中的轴数 （该值必须与群组中的轴数相同，否则将返回群组中的轴数。）
Direction	I16	IN	方向： 0：DIR_CW （顺时针） 1：DIR_CCW （逆时针）

Acm_GpMove3DArcRel_V

函数原型	U32 Acm_GpMove3DArcRel_V(HAND GroupHandle, PF64 CenterArray, PF64 NVectorArray, F64 Degree, PU32 pArrayElements, I16 Direction)		
说明	按照法向量执行群组相对 3D 圆弧插补。		
返回值	错误代码		
注解	用户可以输入圆心、法向量、角度和方向，然后执行相对 3D 圆弧插补。请参考章节 9.3.3.2：3 轴圆弧插补。		
函数原型参数			
名称	类型	IN 或 OUT	说明

GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的相对距离
NVectorArray	PF64	IN	基准面法向量
Degree	F64	IN	旋转角度（范围：0~360）
pArrayElements	PU32	IN/OUT	数列中的轴数（该值必须与群组中的轴数相同，否则将返回群组中的轴数。）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMove3DArcAbs_V

函数原型	U32 Acm_GpMove3DArcAbs_V(HAND GroupHandle, PF64 CenterArray, PF64 NVectorArray, F64 Degree, PU32 pArrayElements, I16 Direction)		
说明	按照法向量执行群组绝对 3D 圆弧插补。		
返回值	错误代码		
注解	用户可以输入圆心、法向量、角度和方向，然后执行绝对 3D 圆弧插补。请参考章节 9.3.3.2：3 轴圆弧插补。		

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的绝对距离
NVectorArray	PF64	IN	基准面法向量
Degree	F64	IN	旋转角度（范围：0~360）
pArrayElements	PU32	IN/OUT	数列中的轴数（该值必须与群组中的轴数相同，否则将返回群组中的轴数。）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveHelixRel

函数原型	U32 Acm_GpMoveHelixRel (HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)		
说明	命令群组进行相对螺旋运动。		
返回值	错误代码		
注解	了解 Acm_GpMoveHelixAbs。		

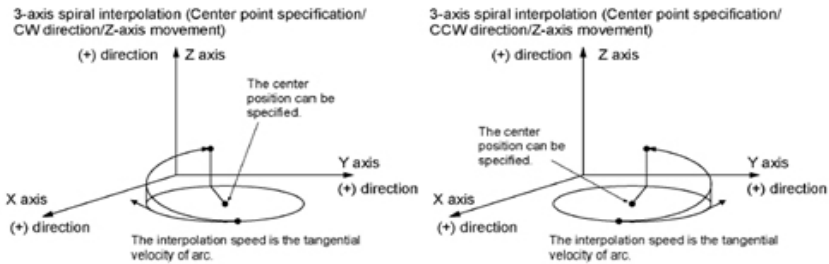
函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	轴相对于圆心的相对距离
EndArray	PF64	IN	轴相对于终点的绝对距离
pArrayElements	PU32	IN/OUT	数列中的轴数（该值必须与群组中的轴数相同，否则将返回群组中的轴数。）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveHelixAbs

函数原型	U32 Acm_GpMoveHelixAbs (HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组进行绝对螺旋运动。
返回值	错误代码
	该命令只支持 3 个轴。CenterArray 和 EndArray 中的元素必须按照轴的 PhysicalID 的顺序。用户可通过属性 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动，其它轴决定螺旋线的高度。CenterArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。 比如：Group (Y、Z、U)，CenterArray (Y、Z、U)，EndArray (Y、Z、U)。如果 PAR_GpRefPlane =1(YZ_Plane)，Z 轴和 U 轴将进行 ARC 插补运动，EndArray 中的 Y 值为螺旋线的高度 螺旋线如下图所示：

注解



函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	圆心点的绝对距离
EndArray	PF64	IN	终点的绝对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向：0：DIR_CW（顺时针）1：DIR_CCW（逆时针）

Acm_GpMoveHelixRel_3P

函数原型	U32 Acm_GpMoveHelixRel_3P (HAND GroupHandle, PF64 RefArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)
说明	命令群组通过三个指定点执行相对螺旋插补
返回值	错误代码
注解	了解 Acm_GpMoveHelixAbs_3P。

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
RefArray	PF64	IN	参考点的相对距离
EndArray	PF64	IN	终止点的相对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0：DIR_CW（顺时针） 1：DIR_CCW（逆时针）

Acm_GpMoveHelixAbs_3P

函数原型	U32 Acm_GpMoveHelixAbs_3P (HAND GroupHandle, PF64 RefArray, PF64 EndArray, PU32 pArrayElements, I16 Direction)		
说明	命令群组通过三个指定点执行绝对螺旋插补		
返回值	错误代码		
注解	必须为群组中的 3 个轴。通过指定三个点命令螺旋运动。参数 RefArray、CenterArray 和 EndArray 值的顺序必须按照轴的 PhysicalID 的顺序。RefArray 和 EndArray 中距离的单位为群组中每个轴的 PPU。 用户可通过 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动。比如：Group (Y、Z、U)，RefArray (Y、Z、U)，CenterArray (Y、Z、U)，EndArray (Y、Z、U)，PAR_GpRefPlane =1(YZ_Plane)。Z 轴和 U 轴将进行 ARC 插补运动，EndArray 中的 Y 值为螺旋线的高度。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
RefArray	PF64	IN	参考点的绝对距离
EndArray	PF64	IN	终点的绝对距离
pArrayElements	PU32	IN/OUT	阵列中的元素个数（该个数必须等于该群组中的轴个数，否则将返回群组中的轴个数）
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveHelixRel_Angle

函数原型	U32 Acm_GpMoveHelixRel_Angle(HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 ArrayElements, I16 Direction)		
说明	按照相对圆心坐标、角度、终点和旋转方向执行圆弧插补。		
返回值	错误代码		
注解	旋转角度必须输入基准面定义的 EndArray[0~2] 中，而基准面可通过 Par_GpRefPlane 设置，其他参数为终点。请参考章节 9.3.4.2 中的方法 2，其中，基准面为 X-Y 平面； EndArray[0] 和 EndArray[1] 为旋转角度，值为 1,080； EndArray[2] 值为 10,000，是 Z 轴终点。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	圆心与起点的相对距离
EndArray	PF64	IN	圆心与终点的相对距离
pArrayElements	PU32	IN	轴数
Direction	I16	IN	方向： 0: DIR_CW（顺时针） 1: DIR_CCW（逆时针）

Acm_GpMoveHelixAbs_Angle

函数原型	U32 Acm_GpMoveHelixAbs_Angle(HAND GroupHandle, PF64 CenterArray, PF64 EndArray, PU32 ArrayElements, I16 Direction)		
说明	按照绝对圆心坐标、角度、终点和旋转方向执行圆弧插补。		
返回值	错误代码		

注解

旋转角度必须输入基准面定义的 EndArray[0~2] 中，而基准面可通过 Par_GpRefPlane 设置，其他参数为终点。请参考章节 9.3.4.2 中的方法 2，其中，基准面为 X-Y 平面； EndArray[0] 和 EndArray[1] 为旋转角度，值为 1,080； EndArray[2] 值为 10,000，是 Z 轴终点。

函数原型参数

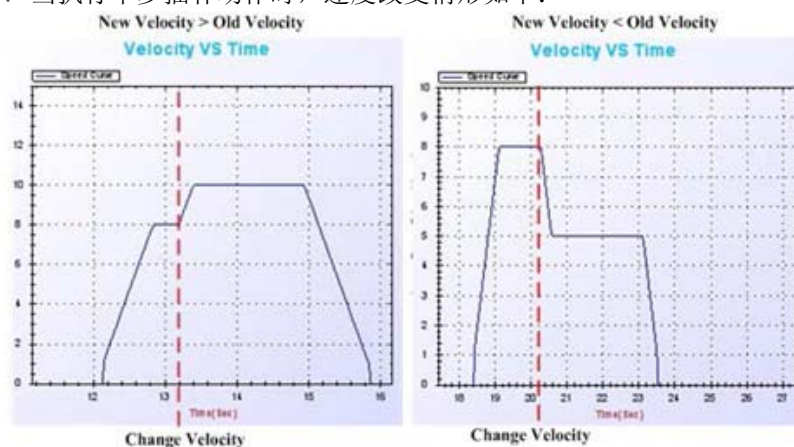
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	圆心坐标
EndArray	PF64	IN	终点坐标
ArrayElements	PU32	IN	轴数
Direction	I16	IN	方向： 0: DIR_CW （顺时针） 1: DIR_CCW （逆时针）

A.5.4.1 Acm_GpChangeVel

函数原型	U32 Acm_GpChangeVelByRate(HAND GroupHandle, U32 Rate)
说明	当群组执行直线插补时，改变群组运行速度。
返回值	错误代码

当群组在运动的过程中，可以对其下达改变速度命令，运动卡在收到此命令后会根据情况作出改变速度的动作。当群组在执行线性插补、圆弧插补、螺旋插补以及连续插补等动作时，可以通过此命令改变运动的速度。若此命令成功下达，如果在下次运动之前未进行重新设定速度，NewVelocity 会作用到下次的运动中。

(1). 当执行单步插补动作时，速度改变情形如下：



如果剩余的 Pulse 数不足以支持达到设定的新速度，则板卡会自行计算能达到的速度值。

(2). 当执行连续插补时

■ BufferMode: Blending Disabled

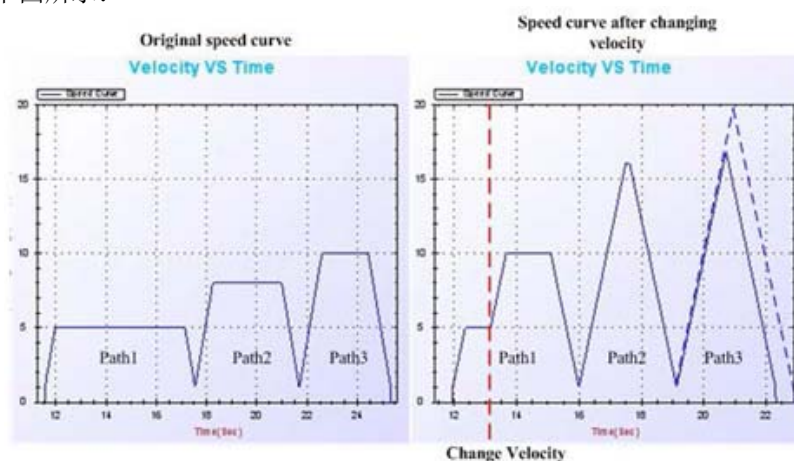
在此模式下，Path Buffer 中的各段 Path，都有自己完整的加减速过程，在运动的过程中，如果执行 ChangeV 操作，则当前执行段的 Path 的速度会加速 / 减速到新速度，则以后各段的速度会成比例增加 / 减少。各段在改变速度时，如果段剩余的 Pulse 不够，则自动计算能达到的新速度。

例如：path1: VL = 1000, VH = 5000

path 2: VL = 1000, VH = 8000

path 3: VL = 1000, VH = 10000

当执行 Path1 的过程中，下达 ChangeV, New Velocity = 10000，则第二段 path 的速度应该为 16000，第三段的速度应该为 20000。但是实际执行情况如下图所示：



注解

■ BlendingMode: Blending Enable, BlendingTime >0

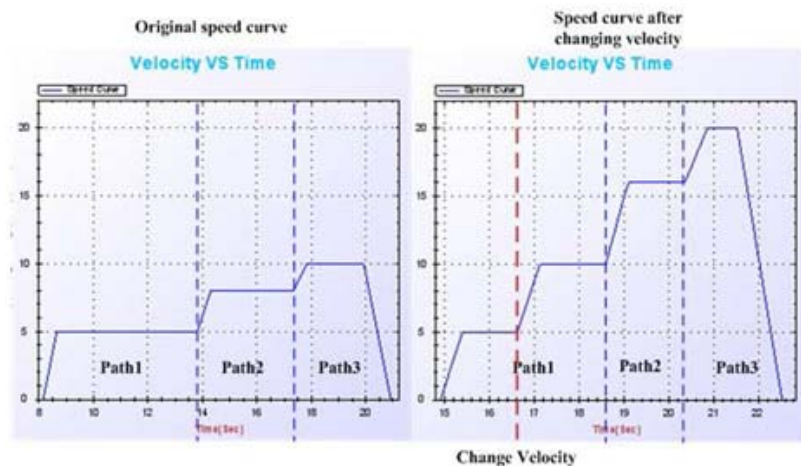
在此模式下, Path Buffer 中的各段 Path, 没有完整的加减速过程, 其速度轨迹由 BlendingTime 以及各段的 FL 决定, 详细请参阅 Acm_GpAddPath。在运动的过程中, 如果执行 ChangeV 操作, 则当前执行段的 Path 的速度会加速 / 减速到新速度, 则以后各段的速度会成比例增加 / 减少。各段在改变速度时, 如果段剩余的 Pulse 不够, 则自动计算能达到的新速度。

例如: path1: VL = 0, VH = 5000

path 2: VL = 0, VH = 8000

path 3: VL = 0, VH = 10000

当执行 Path1 的过程中, 下达 ChangeV, New Velocity = 10000, 则第二段 path 的速度应该为 16000, 第三段的速度应该为 20000。如下图所示:



函数原型参数

如果 Change velocity 命令下达在 Blending 阶段, 则 ChangeV 推迟到下一段 Path 执行。

■ FlyMode: Blending Enable, BlendingTime=0

此模式下的速度轨迹与 Blending 模式下类似, 只是其速度轨迹中两段 Path 之间不会减速到 FL。详细请参阅 Acm_GpAddPath。在运动的过程中, 如果执行 ChangeV 操作, 则当前执行段的 Path 的速度会加速 / 减速到新速度, 则以后各段的速度会成比例增加 / 减少。各段在改变速度时, 如果段剩余的 Pulse 不够, 则自动计算能达到的新速度。

Note:

减速段不支持 ChangeV 操作。

函数原型

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
NewVelocity	F64	IN	新速度 (单位: PPU/s)

Acm_GpChangeVelByRate

函数原型	U32 Acm_GpChangeVelByRate (HAND GroupHandle, U32 Rate)
说明	按照设定的比例改变当前正在执行的群组动作的运行速度
返回值	错误代码
注解	新速度 = 群组之前的运行速度 * Rate * 0.01。Rate 须大于 0 且小于群组中 ID 最小的轴的最大速度与当前 Group 的速度的比值。新速度仅对此运动有效。关于在插补或连续插补过程中改变速度的情形, 详细可参考 Acm_GpChangeVel。

函数原型参数

名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

A. 5.5 路径

Acm_GpAddPath

函数原型	U32 Acm_GpAddPath (HAND GroupHandle, U16 MoveCmd, U16 MoveMode, F64 FH, F64 FL, PF64 EndPoint_DataArray, PF64 CenPoint_DataArray, PU32 ArrayElements)		
说明	将一个插补路径添加至系统路径缓存		
返回值	错误代码		
注解	详细信息请参考章节 9.5。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
MoveCmd	U16	IN	运动指令。请参考章节 9.5。
MoveMode	U16	IN	运动模式： 0：使能速度交接模式 1：禁用速度交接模式
FH	F64	IN	运行速度 （单位：群组的 PPU/s）/GPDELAY 命令中的延迟时间 （单位：ms）。
FL	F64	IN	低速度 （起始速度）（单位：群组的 PPU/s）
EndPoint_DataArray	PF64	IN	终点坐标 （单位：每个轴的 PPU）
CenPoint_DataArray	PF64	IN	中心点坐标 （单位：每个轴的 PPU）
ArrayElements	PU32	IN/OUT	阵列元素的个数必须小于群组中轴的个数，否则将返回群组中的轴个数。

Acm_GpResetPath

函数原型	U32 Acm_GpResetPath (PHAND GroupHandle)		
说明	清空系统路径缓存。如果有群组正在执行路径，则路径运动将停止		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	PHAND	IN	来自 Acm_GpAddaxis 的群组句柄

Acm_GpLoadPath

函数原型	U32 Acm_GpLoadPath(HAND GroupHandle, PI8 FilePath, PHAND PathHandle, PU32 pTotalCount)		
说明	加载来自路径文件的路径数据。一次可加载最多 600 个路径数据		
返回值	错误代码		
注解	路径数据文件（二进制）通常由运动实用程序的 [Path Editor] 生成。如果用户熟悉研华运动产品，那么可以自行创建文件。当不再使用 PathHandle 或应用关闭时，必须通过 Acm_GpUnloadPath 卸载 PathHandle，同时 PathHandle 中包含的路径将从驱动中删除。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

FilePath	PI8	IN	指向需要加载的运动路径文件
PathHandle	PHAND	OUT	返回指针指向路径句柄
pTotalCount	PU32	OUT	返回路径文件中路径数据的实际总个数

Acm_GpUnloadPath

函数原型	U32 Acm_GpUnloadPath (HAND GroupHandle, PHAND PathHandle)		
说明	卸载路径数据		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
PathHandle	PI8	IN	指向需要加载的运动路径文件

Acm_GpMovePath

函数原型	U32 Acm_GpMovePath (HAND GroupHandle, HAND PathHandle)		
说明	开始连续插补运动（路径）		
返回值	错误代码		
注解	如果 Acm_GpLoadPath 返回 PathHandle，那么路径数据将首先加载到系统路径缓存，然后执行路径缓存中的路径。如果 PathHandle 为 NULL，将直接执行系统路径缓存中的路径数据		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
PathHandle	PHAND	IN	指针指向来自 Acm_GpLoadPath 的路径句柄

Acm_GpMoveAllPath

函数原型	U32 Acm_GpMoveAllPath(PHAND GroupHandle, U32 ArrayElements)		
说明	开始选中群组的连续插补运动（路径）		
返回值	错误代码		
注解	开始同步执行多群组路径表运动。将路径载入每个群组之后，用户即可调用该函数，且指定群组中的所有路径将同步运行。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
ArrayElements	U32	IN	需要开始同步运动的群组数

Acm_GpGetPathStatus

函数原型	U32 Acm_GpGetPathStatus (HAND GroupHandle, PU32 pCurIndex, PU32 pCurCmdFunc, PU32 pRemainCount, PU32 pFreeSpaceCount)		
说明	获取路径缓存的当前状态		
返回值	错误代码		
注解	用户必须输入 GroupHandle 并获取该群组的路径状态		

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
pCurIndex	PU32	OUT	返回路径缓存中当前正在执行的路径对应的的当前索引
pCurCmdFunc	PU32	OUT	
pRemainCount	PU32	OUT	返回路径中未执行的路径数据的个数
pFreeSpaceCount	PU32	OUT	返回路径缓存中剩余空间的个数

Acm_GpMoveSelPath

函数原型	U32 Acm_GpMoveSelPath (HANDGroupHandle, HAND PathHanle, U32 StartIndex, U32EndIndex, U8Repeat)
说明	执行路径缓存中指定的起始索引到终止索引范围内的路径
返回值	错误代码
注解	命令运动路径索引在 StartIndex 和 EndIndex 之间的路径。 如果 PathHandle 是 Null, 将运动系统路径缓存中的指定路径; 如果 PathHandle 不是 Null, PathHandle 中的路径将首先加载至系统路径缓存, 然后运动到指定路径。不支持绝对模式。 如果 Repeat 的值为 0, 那么将连续且重复执行指定路径, 直到停止群组运动。如果 EndIndex 的值大于系统路径缓冲中的路径个数, 将把 StartIndex 路径和最后一个索引路径之间的路径运动至系统路径缓冲。StartIndex 和 EndIndex 是针对当前 path buffer 中所保留的 path。例如当前 path buffer 中剩余有 10 段 Path (可通过调用 Acm_GpGetPathStatus 查看剩余的 path 数量), 则 StartIndex 和 EndIndex 则可取范围为 0~9

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
PathHandle	HAND	IN	指针指向来自 Acm_GploadPath 的路径句柄
StartIndex	U32	IN	起始索引。(0 ~ 9999)
EndIndex	U32	IN	结尾索引。(0 ~ 9999)
Repeat	U8	IN	重复个数。(0 ~ 255)

Acm_GpGetPathIndexStatus

函数原型	U32 Acm_GpGetPathIndexStatus (HAND GroupHandle, U32 Index, PU16 CmdFunc, PU16 MoveMode, PF64 FH, PF64 FL, F64 EndPoint_DataArray, PF64 CenPoint_DataArray, PU32 ArrayElements)
说明	读取连续补间 (Path, 路径) 的执行笔数的 API。
返回值	错误代码
注解	如果用户想要了解路径设置, 可调用该 API。StartIndex 和 EndIndex 是针对当前 path buffer 中所保留的 path。例如当前 path buffer 中剩余有 10 段 Path (可通过调用 Acm_GpGetPathStatus 查看剩余的 path 数量), 则 StartIndex 和 EndIndex 则可取范围为 0~9

函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
Index	U32	IN	路径的索引

CmdFunc	PU16	OUT	返回正在执行的当前命令函数
MoveMode	PU16	OUT	运动模式： 0：使能速度交接模式 1：禁用速度交接模式
FH	PF64	OUT	单位：主轴的 PPU（行走距离最长的轴）
FL	PF64	OUT	单位：主轴的 PPU（行走距离最长的轴）
EndPoint_DataArray	PF64	OUT	单位：各轴的 PPU（行走距离最长的轴）
CenPoint_DataArray	PF64	OUT	单位：各轴的 PPU（行走距离最长的轴）
ArrayElements	PU32	IN/OUT	返回轴个数

A. 5. 6 暂停和恢复

Acm_GpPauseMotion

函数原型	U32 Acm_GpPauseMotion(HAND GroupHandle)		
说明	暂停群组运动命令		
返回值	错误代码		
注解	请参考章节 9.5.7。		
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

Acm_GpResumeMotion

函数原型	U32 Acm_GpResumeMotion (HAND GroupHandle)		
说明	恢复暂停的群组运动命令		
返回值	错误代码		
注解			
函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄

附录 B

属性列表

B.1 设备特性属性

FT_DevIpoTypeMap

数据类型	R/W	属性 ID	默认值
U32	R	0	–
含义	获取设备支持的插补类型		

注解：

1: 支持； 0: 不支持。

位	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	线性插补，2 轴	DEV_IPO_LINE_2AX
1	线性插补，3 轴	DEV_IPO_LINE_3AX
2	线性插补，4 轴	DEV_IPO_LINE_4AX
3	线性插补，5 轴	DEV_IPO_LINE_5AX
4	线性插补，6 轴	DEV_IPO_LINE_6AX
5	线性插补，7 轴	DEV_IPO_LINE_7AX
6	线性插补，8 轴	DEV_IPO_LINE_8AX
7	未定义	
8	2 轴圆弧插补	DEV_IPO_ARC_2AX
9	3 轴圆弧插补	DEV_IPO_ARC_3AX
10	螺旋插补	DEV_IPO_SPIRAL
11	速度交接	DEV_IPO_BLENDING
12	速度前瞻	DEV_IPO_SPEED_FORWARD
13~15	未定义	
16	同步电子齿轮	DEV_IPO_GEAR
17	同步电子凸轮	DEV_IPO_CAM
18	龙门控制	DEV_IPO_GANTRY
19	切线跟随	DEV_IPO_TANGENT
20~23	未定义	
24	选择需要执行的路径	DEV_IPO_SELPTH
25~31	未定义	

例程：如下获取设备插补类型

```
HAND      m_Devhand;  // 设备句柄
U32       DevIpoType; // 获取的属性值
Acm_GetU32Property(m_Devhand, FT_DevIpoTypeMap, &DevIpoType);
If ( DevIpoType & DEV_IPO_LINE_2AX)
{ // 支持两轴直线插补 }
If (DevIpoType & DEV_IPO_LINE_3AX)
{ // 支持三轴直线插补 }
If () .....
```

FT_DevAxesCount

数据类型	R/W	属性 ID	默认值
U32	R	1	–
含义	获取设备的轴数		

注解：

设备的特性，只读。

FT_DevFunctionMap

数据类型	R/W	属性 ID	默认值
U32	R	2	–
含义	获取设备支持的功能		

注解:

位	说明	宏定义 (定义在头文件 AdvMotDrv.h)
0	运动	DEV_FUNC_MOT
1	DI	DEV_FUNC_DI
2	DO	DEV_FUNC_DO
3	AI	DEV_FUNC_AI
4	AO	DEV_FUNC_AO
5	定时器	DEV_FUNC_TMR
6	计数器	DEV_FUNC_CNT
7	DAQ DI	DEV_FUNC_DAQDI
8	DAQ DO	DEV_FUNC_DAQDO
9	DAQ AI	DEV_FUNC_DAQAI
10	DAQ AO	DEV_FUNC_DAQAO
11	Emg	DEV_FUNC_EMG
12	DEV_FUNC_MDAQ	DEV_FUNC_MDAQ
13~31	未定义	

例程: 如下获取设备支持的功能

```

HAND      m_Devhand;  // 设备句柄
U32       DevFunMap;  // 获取的属性值
Acm_GetU32Property(m_Devhand, FT_DevFunctionMap, & DevFunMap);
If (DevFunMap & DEV_FUNC_MOT)
{ // 支持运动功能 }
If (DevFunMap & DEV_FUNC_DI)
{ // 支持 DI 功能 }
If () .....

```

FT_DevOverflowCntr

数据类型	R/W	属性 ID	默认值
U32	R	3	2, 147, 483, 647
含义	位置计数器的最大值		

注解:

最大值为 2, 147, 483, 647。

FT_MasCyclicCnt_R0

数据类型	R/W	属性 ID	默认值
U32	R	4	–
含义	获取连接至 Motion 环的从站数。		

FT_MasCyclicCnt_R1

数据类型	R/W	属性 ID	默认值
U32	R	5	–
含义	获取连接至 IO 环的从站数。		

FT_DevMDAQTypeMap

数据类型	R/W	属性 ID	默认值
U32	R	6	–
含义	MotionDAQ 支持的资料类型		

注解：

设备特性，只读。MotionDAQ 支持的资料类型如下表所示：

位	说明	宏定义（定义在头文件 AdvMotDrv.h）
0	理论位置	MQ_TYPE_CMDPOSI
1	实际位置	MQ_TYPE_ACTPOSI
2	延迟位置（理论位置与实际位置的偏差）	MQ_TYPE_LAGPOSI
3	理论速度值	MQ_TYPE_CMDVEL

例程：如下获取 MotionDAQ 支持的资料类型

```

HAND      m_Devhand;    // 设备句柄
U32      MDAQTypeMap; // 获取的属性值
Acm_GetU32Property(m_Devhand, FT_DevMDAQTypeMap, & MDAQTypeMap);
If(MDAQTypeMap & MQ_TYPE_CMDPOSI)
{ // 支持理论位置 }
If(MDAQTypeMap & MQ_TYPE_ACTPOSI)
{ // 支持实际位置 }
If(MDAQTypeMap & MQ_TYPE_LAGPOSI)
{ // 支持延迟位置 }

```

FT_DevMDAQTrigMap

数据类型	R/W	属性 ID	默认值
U32	R	7	–
含义	触发 MotionDAQ 功能的几种方式		

注解：

设备特性，只读。触发 MotionDAQ 功能的几种方式：

位	说明	宏定义（定义在头文件 AdvMotDrv.h）
0	禁用 MotionDAQ 功能	MP_MQ_TRIG_DISABLE
1	软体命令触发方式（下达 Start 命令触发）	MP_MQ_TRIG_SW
2	DI 触发	MP_MQ_TRIG_DI
3	指定轴运动开始即触发 MotionDAQ 功能	MP_MQ_TRIG_AX_START

例程：如下获取触发 MotionDAQ 的方式

```

HAND      m_Devhand;    // 设备句柄
U32      MDAQTrigMap; // 获取的属性值
Acm_GetU32Property(m_Devhand, FT_DevMDAQTrigMap, & MDAQTrigMap);
If(MDAQTrigMap & MP_MQ_TRIG_DISABLE)
{ // 支持禁用 MotionDAQ 功能 }

```

```
If (MDAQTrigMap & MP_MQ_TRIG_SW)
{// 支持软体触发 MotionDAQ 功能 }
If (MDAQTrigMap & MP_MQ_TRIG_DI)
{// 支持 DI 触发 MotionDAQ 功能 }
.....
```

FT_DevMDAQMaxChan

数据类型	R/W	属性 ID	默认值
U32	R	8	–
含义	记录 MotionDAQ 资料的最大通道数		

注解:

设备特性，只读。

FT_DevMDAQMaxBufCount

数据类型	R/W	属性 ID	默认值
U32	R	9	2000
含义	每个 MotionDAQ 通道最多可记录的资料笔数		

注解:

设备特性，只读。

FT_DevLTCBufMaxCount

数据类型	R/W	属性 ID	默认值
U32	R	10	128
含义	Latch Buffer 最多可记录的资料笔数		

注解:

设备特性，只读。

B.2 设备配置属性

CFG_DevBoardID

数据类型	R/W	属性 ID	默认值
U32	R	201	—
含义	获取设备 ID		

注解：

运动控制卡的属性值为 0~15。

CFG_DevBaseAddress

数据类型	R/W	属性 ID	默认值
U32	R	203	—
含义	返回 IO 基地址		

注解：

板卡配置属性，只读。

CFG_DevInterrupt

数据类型	R/W	属性 ID	默认值
U32	R	204	—
含义	获取设备中断编号		

注解：

板卡配置属性，只读。

CFG_DevBusNumber

数据类型	R/W	属性 ID	默认值
U32	R	205	—
含义	获取设备总线编号		

注解：

板卡配置属性，只读。

CFG_DevSlotNumber

数据类型	R/W	属性 ID	默认值
U32	R	206	—
含义	获取设备插槽编号		

注解：

板卡配置属性，只读。

CFG_DevDriverVersion

数据类型	R/W	属性 ID	默认值
char*	R	207	—
含义	获取 SYS 驱动版本		

注解：

格式为 1.0.0.1。

CFG_DevDllVersion

数据类型	R/W	属性 ID	默认值
char*	R	208	-
含义	获取 dll 驱动版本		

注解:

格式为 1.0.0.1。

CFG_DevLogMsg

数据类型	R/W	属性 ID	默认值
U32	R/W	229	0
含义	选择是否将错误记录在文档中		

注解:

值	说明
0	不记录
1	记录在 .txt 文档中

CFG_DevErrorReact

数据类型	R/W	属性 ID	默认值
U32	R/W	230	0
含义	设置 / 获取当任意轴发生错误时的设备反应		

注解:

设置、获取的值说明如下表所示:

值	说明	宏定义 (头文件 AdvMotDrv.h 中定义)
0	其他轴不反应	DEV_ERR_ACT_DISABLE
1	全部轴立刻停止 (无减速)	DEV_ERR_IMMED_STOP
2	全部轴减速停止	DEV_ERR_DEC_TO_STOP

CFG_DevMultiCmpAxisEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	223	0
含义	启用 / 禁用多轴比较功能		

注解:

0: 禁用 (默认值), 1: 启用

位	说明
0~31	Axis 0 ~ Axis 31

如果某个 (些) 轴被用作多轴比较, 则轴上原有的单轴比较功能 (CFG_AxCmpEnable) 会自动被禁用。

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiCmpOutChannel

数据类型	R/W	属性 ID	默认值
U32	R/W	224	0
含义	指定当多轴比较比对到时，哪些轴的 OUT5 输出脉冲		

注解：

0：禁用（默认值），1：启用

位	说明
0~31	Axis 0 ~ Axis 31

可以多个轴的 OUT5 同时输出。

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiCmpPulseLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	225	1
含义	设置 / 获取多轴比较信号的逻辑电平		

注解：

设置、获取的值说明如下表所示：

值	说明
0	低准位
1	高准位（默认值）

仅用于 CmpPulseOutMode 为脉冲模式和开关模式时，PWM 模式下不起作用。

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiCmpPulseWidth

数据类型	R/W	属性 ID	默认值
U32	R/W	226	1,000
含义	设置 / 获取多轴比较触发时输出的脉冲宽度。单位：us		

注解：

范围：5~85,899,000

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiTrigPulseMode

数据类型	R/W	属性 ID	默认值
U32	R/W	266	0
含义	设置 / 获取多轴比较模式		

注解：

设置、获取的值说明如下表所示：

值	说明
0	脉冲模式（默认值）
1	开关模式

2	PWM 模式
---	--------

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiTrigPWMFreq

数据类型	R/W	属性 ID	默认值
U32	R/W	267	100,000
含义	设定多轴比较输出方式为 PWM 时，Pa 的输出频率。单位：Hz		

注解：

范围：1~250,000

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiTrigPWMDuty

数据类型	R/W	属性 ID	默认值
U32	R/W	268	50.0
含义	设定多轴比较输出方式为 PWM 时，Pa 的占空比，精度 0.5%。		

注解：

范围：0.0~100.0

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

CFG_DevMultiTrigAutoEmptyEn

数据类型	R/W	属性 ID	默认值
U32	R/W	269	0
含义	启用 / 禁用多轴比较结束后，是否自动清空比较数据		

注解：

设置、获取的值说明如下表所示：

值	说明
0	不清空（默认值）
1	清空

详见多轴比较章节。

仅特定 EtherCAT 从站支持此函数。

B.3 DAQ 特性属性

FT_DaqDiMaxChan

数据类型	R/W	属性 ID	默认值
U32	R	50	–
含义	获取 DI 通道的最大个数		

注解：

只读。

若用户使用默认映射关系，则无需下载 Acm_DevLoadMapFile；否则，用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

FT_DaqDoMaxChan

数据类型	R/W	属性 ID	默认值
U32	R	51	–
含义	获取 DO 通道的最大个数		

注解：

只读。

若用户使用默认映射关系，则无需下载 Acm_DevLoadMapFile；否则，用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

FT_DaqAiRangeMap

数据类型	R/W	属性 ID	默认值
U32	R	52	–
含义	获取板卡支持的 AI 范围		

注解：

1：支持； 0：不支持。

位	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	+/- 10V	DAQ_AI_NEG_10V_TO_10V
1	+/- 5V	DAQ_AI_NEG_5V_TO_5V
2	+/- 2.5V	DAQ_AI_NEG_2500MV_TO_2500MV
3	+/- 1.25V	DAQ_AI_NEG_1250MV_TO_1250MV
4	+/- 0.625V	DAQ_AI_NEG_625MV_TO_625MV
5	+/-1V	DAQ_AI_NEG_1V_TO_1V
6	+/-0.5V	DAQ_AI_NEG_500MV_TO_500MV
7	+/-0.15V	DAQ_AI_NEG_150MV_TO_150MV
8	0~10V	DAQ_AI_NEG_0_TO_10V
9	0~500MV	DAQ_AI_NEG_0_TO_500MV
10~15	未定义	
16	0 ~ 20 mA	DAQ_AI_OMA_TO_20MA
17	4 ~ 20 mA	DAQ_AI_4MA_TO_20MA
18	+/-20mA	DAQ_AI_NEG_20MA_TO_20MA
19~31	未定义	

例程：如下获取板卡支持的 AI 范围

```
HAND    m_Devhand;  // 设备句柄
U32     AIRangeMap; // 获取的属性值
```

```
Acm_GetU32Property(m_Devhand, FT_DaqAiRangeMap, & AIRangeMap);  
If(AIRangeMap & CFG_DAQ_AI_NEG_10V_TO_10V)  
{// 支持 +/- 10V }  
If (AIRangeMap & CFG_DAQ_AI_NEG_5V_TO_5V)  
{// 支持 +/-5V}  
If (AIRangeMap & CFG_DAQ_AI_NEG_2500MV_TO_2500MV)  
{// 支持 +/- 2.5V }  
.....
```

FT_DaqAoRangeMap

数据类型	R/W	属性 ID	默认值
U32	R	53	-
含义	获取板卡支持的 AO 范围		

注解:

1: 支持; 0: 不支持。

位	说明	宏定义 (头文件 AdvMotDrv.h 中定义)
0	+/- 10V	DAQ_AO_NEG_10V_TO_10V
1	+/- 5V	DAQ_AO_NEG_5V_TO_5V
2	+/- 2.5V	DAQ_AO_NEG_2500MV_TO_2500MV
3	+/- 1.25V	DAQ_AO_NEG_1250MV_TO_1250MV
4	+/- 0.625V	DAQ_AO_NEG_625MV_TO_625MV
5	0 ~ 10V	DAQ_AO_NEG_0V_TO_10V
6	0 ~ 5V	DAQ_AO_NEG_0V_TO_5V
7~15	预留	
16	0 ~ 20 mA	DAQ_AO_0MA_TO_20MA
17	4 ~ 20 mA	DAQ_AO_4MA_TO_20MA
18~31	预留	

FT_DaqAiMaxSingleChan

数据类型	R/W	属性 ID	默认值
U32	R	54	-
含义	获取设备的最大单端 AI 通道个数		

注解:

只读。

若用户使用默认映射关系, 则无需下载 Acm_DevLoadMapFile; 否则, 用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

FT_DaqAiMaxDiffChan

数据类型	R/W	属性 ID	默认值
U32	R	55	-
含义	获取设备的最大差分 AI 通道个数		

注解:

只读。

若用户使用默认映射关系, 则无需下载 Acm_DevLoadMapFile; 否则, 用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

FT_DaqAoMaxChan

数据类型	R/W	属性 ID	默认值
U32	R	57	–
含义	获取网络中的 A0 通道总数		

注解：

只读。

若用户使用默认映射关系，则无需下载 Acm_DevLoadMapFile；否则，用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

FT_DaqCntMaxChan

数据类型	R/W	属性 ID	默认值
U32	R	59	–
含义	获取网络中的 CNT 通道总数		

注解：

只读。

若用户使用默认映射关系，则无需下载 Acm_DevLoadMapFile；否则，用户需要在获取该属性之前通过 Acm_DevLoadMapFile 下载映射文件。

B. 4 DAQ 配置属性

CFG_MasCycleTime

数据类型	R/W	属性 ID	默认值
U32	R	261	500
含义	为数据通信获取运动环的循环时间（单位：微秒）		

CFG_IoCycleTime

数据类型	R/W	属性 ID	默认值
U32	R	262	200
含义	为数据通信获取 IO 环的循环时间（单位：微秒）		

注解：

只读。

B.5 DAQ 通道配置属性

属性值可以通过下列函数获得：

`Acm_GetChannelProperty`

`Acm_GetMultiChannelProperty`

并通过下列函数设置：

`Acm_SetChannelProperty`

`Acm_SetMultiChannelProperty`

DI/DO/AI/AO 的通道 ID 需要被传入属性。

对于该章节中的所有属性，用户需要通过 Utility 了解 DI/DO 端口和 AI/AO/CNT 通道的映射信息。若用户使用默认映射关系，则无需下载 `Acm_DevLoadMapFile`；否则，用户需要在获取该属性之前通过 `Acm_DevLoadMapFile` 下载映射文件。

CFG_CH_DaqDiInvertEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	1500	0
含义	设置 / 获取 DI Invert		

注解：

位	说明
0	不逆变
1	逆变

CFG_CH_DaqDiLowFilter

数据类型	R/W	属性 ID	默认值
F64	R/W	1501	0
含义	设置 / 获取 DI 过滤器的下限值。范围：0~65, 535		

注解：

一旦某 DI 被成功设置并用于 ADAM-E5017，DI 过滤器下限值将被复制至所有 DI。

CFG_CH_DaqDiHighFilter

数据类型	R/W	属性 ID	默认值
F64	R/W	1502	0
含义	设置 / 获取 DI 过滤器的上限值。范围：0~65, 535		

注解：

一旦某 DI 被成功设置并用于 ADAM-E5017，DI 过滤器上限值将被复制至所有 DI。

CFG_CH_DaqDoFsvEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1503	0
含义	设置 / 获取 DO 故障安全值		

注解：

位	说明
0	出现通信错误时，DO 将处于低电平状态

1	出现通信错误时，D0 将处于高电平状态
---	---------------------

CFG_CH_DaqAoRanges

数据类型	R/W	属性 ID	默认值
F64	R/W	1504	–
含义	设置 / 获取 AO 范围		
	值	说明	
	0	+/-10V	
	1	+/-5V	
	2	+/-2.5V（不支持）	
	3	+/-1.25V（不支持）	
	4	+/-0.625V（不支持）	
	5	0~10V	
	6	0~5V	
	7~15	预留	
	16	0~20mA	
	17	4~20mA	

注解：

值	描述	宏定义（头文件 AdvMotDrv.h 中定义）
0x0	+/- 10 V	CFG_DAQ_AO_NEG_10V_TO_10V
0x1	+/- 5 V	CFG_DAQ_AO_NEG_5V_TO_5V
0x2	+/- 2.5 V	CFG_DAQ_AO_NEG_2500MV_TO_2500MV
0x3	+/- 1.25 V	CFG_DAQ_AO_NEG_1250MV_TO_1250MV
0x4	+/- 0.625 V	CFG_DAQ_AO_NEG_625MV_TO_625MV
0x5	0~10 V	CFG_DAQ_AO_NEG_0V_TO_10V
0x6	0~5 V	CFG_DAQ_AO_NEG_0V_TO_5V
0x10	0~20mA	CFG_DAQ_AO_0MA_TO_20MA
0x11	4~20mA	CFG_DAQ_AO_4MA_TO_20MA

CFG_CH_DaqAiRanges

数据类型	R/W	属性 ID	默认值
F64	R/W	1505	–
含义	设置 / 获取 AI 范围		
	值	说明	
	0	+/-10V	
	1	+/-5V（仅适用于 ADAM-E5017）	
	2	+/-2.5V（不支持）	
	3	+/-1.25V（不支持）	
	4	+/-0.625V（不支持）	
	5	+/-1V（仅适用于 ADAM-E5017）	
	6	+/-0.5V（仅适用于 ADAM-E5017）	
	7	+/-0.15V（仅适用于 ADAM-E5017）	
	8	0~10V（仅适用于 ADAM-E5017UH）	
	9	0~500mV（仅适用于 ADAM-E5017UH）	
	10~15	预留	

16	0~20mA（仅适用于 ADAM-E5017UH）
17	4~20mA
18	+/-20mA（仅适用于 ADAM-E5017）

注解：

AI 范围设置将被复制至 ADAM-E5017 的所有通道；与 ADAM-E5017 不同，ADAM-E5017UH 支持单独设置每个通道的输入范围。

值	描述	宏定义（头文件 AdvMotDrv.h 中定义）
0x0	+/- 10 V	CFG_DAQ_AI_NEG_10V_TO_10V
0x1	+/- 5 V	CFG_DAQ_AI_NEG_5V_TO_5V
0x2	+/- 2.5 V	CFG_DAQ_AI_NEG_2500MV_TO_2500MV
0x3	+/- 1.25 V	CFG_DAQ_AI_NEG_1250MV_TO_1250MV
0x4	+/- 0.625 V	CFG_DAQ_AI_NEG_625MV_TO_625MV
0x5	+/- 1 V	CFG_DAQ_AI_NEG_1V_TO_1V
0x6	+/- 0.5 V	CFG_DAQ_AI_NEG_500MV_TO_500MV
0x7	+/- 0.15 V	CFG_DAQ_AI_NEG_1500MV_TO_1500MV
0x8	0~10 V	CFG_DAQ_AI_NEG_0_TO_10V
0x9	0~500 mA	CFG_DAQ_AI_NEG_0_TO_500MV
0x10	0~20 mA	CFG_DAQ_AI_0MA_TO_20MA
0x11	4~20 mA	CFG_DAQ_AI_4MA_TO_20MA
0x12	+/- 20 mA	CFG_DAQ_AI_NEG_20MA_TO_20MA

CFG_CH_DaqAiEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1506	1
含义	启用 / 禁用 AI		

注解：

值	说明
0	禁用 AI 通道
1	启用 AI 通道

CFG_CH_DaqAiIntegrationTime

数据类型	R/W	属性 ID	默认值
F64	R/W	1507	0
含义	设置 / 获取 AI 设置时间		

注解：

值	说明
0	60Hz
1	50Hz

CFG_CH_DaqAoFsv

数据类型	R/W	属性 ID	默认值
F64	R/W	1508	-
含义	获取 AO 故障安全值，或将 AO 故障安全值设置为 AO 值		

注解：
该属性仅适用于 ADAM-E5024H。

	值	说明
获取	--	出现通信错误时的 A0 输出值
设置	1	将 CFG_CH_DaqAoFsv 的值设置为 1 之后，A0 故障安全值将成为指定通道的当前 A0 值。出现通信错误时，A0 将输出故障安全值。

CFG_CH_DaqAoStartup

数据类型	R/W	属性 ID	默认值
F64	R/W	1509	-
含义	获取 A0 故障安全值，或将 A0 故障安全值设置为 A0 值		

注解：
该属性仅适用于 ADAM-E5024H。

	值	说明
获取	--	ADAM-5000/ECAT 上电后的 A0 输出值
设置	1	将 CFG_CH_DaqAoStartup 的值设置为 1 之后，A0 启动值将成为指定通道的当前 A0 值。ADAM-5000/ECAT 上电后，A0 将输出启动值。

CFG_CH_DaqCntEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1510	--
含义	启用 / 禁用该 Counter 通道		

注解：
设置、获取的值说明如下表所示：

值	说明
0	禁用（默认值）
1	启用

CFG_CH_DaqCntPulseInMode

数据类型	R/W	属性 ID	默认值
F64	R/W	1511	--
含义	设置 / 获取 Counter 反馈脉冲输入模式		

注解：
设置、获取的值说明如下表所示：

值	说明
0	AB1X
1	AB2X
2	AB4X
3	CW/CCW
4	Pulse/DIR

CFG_CH_DaqCntInitValue

数据类型	R/W	属性 ID	默认值
F64	R/W	1512	0
含义	设置 / 获取 Counter 初始值。范围：-2 ³¹ ~(2 ³¹ -1)		

注解：

CFG_CH_DaqCntMaxValue

数据类型	R/W	属性 ID	默认值
F64	R/W	1513	2, 147, 483, 647
含义	设置 / 获取 Counter 最大值。范围：0~(2 ³¹ -1)		

注解：

CFG_CH_DaqCntOverflowMode

数据类型	R/W	属性 ID	默认值
F64	R/W	1514	0
含义	设置 / 获取 Counter 计数模式		

注解：

设置、获取的值说明如下表所示：

值	说明
0	正常计数模式 范围：[-(CounterMax+1) ~ CounterMax]
1	溢出模式 范围：[0 ~ CounterMax]

CFG_CH_DaqCntLatchEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1515	0
含义	启用 / 禁用该 Counter 通道锁存功能		

注解：

设置、获取的值说明如下表所示：

值	说明
0	禁用（默认值）
1	启用

CFG_CH_DaqCntLatchEdge

数据类型	R/W	属性 ID	默认值
F64	R/W	1516	0
含义	设置 / 获取该 Counter 通道 Latch 信号的触发沿		

注解：

值	说明
0	上升沿
1	下降沿
2	上升与下降沿

CFG_CH_DaqCntCmpEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1517	0
含义	启用 / 禁用比较功能		

注解:

设置、获取的值说明如下表所示:

值	说明
0	禁用 (默认值)
1	启用

CFG_CH_DaqCntCmpMethod

数据类型	R/W	属性 ID	默认值
F64	R/W	1518	0
含义	设置 / 获取比较方法		

注解:

设置、获取的值说明如下表所示:

值	说明
0	>= 位置计数器 (默认值)
1	<= 位置计数器
2	= 计数器 (无方向)

CFG_CH_DaqCntCmpPulseEnable

数据类型	R/W	属性 ID	默认值
F64	R/W	1519	0
含义	启用 / 禁用轴比较输出		

注解:

设置、获取的值说明如下表所示:

值	说明
0	禁用 (默认值)
1	启用

CFG_CH_DaqCntCmpPulseMode

数据类型	R/W	属性 ID	默认值
F64	R/W	1520	0
含义	设置 / 获取轴比较模式		

注解:

设置、获取的值说明如下表所示:

值	说明
0	脉冲模式 (默认值)
1	开关模式
2	开启模式

CFG_CH_DaqCntCmpPulseLogic

数据类型	R/W	属性 ID	默认值
F64	R/W	1521	0
含义	设置 / 获取比较信号的逻辑准位		

注解:

设置、获取的值说明如下表所示:

值	说明
0	正常输出
1	反向输出

CFG_CH_DaqCntCmpPulseWidth

数据类型	R/W	属性 ID	默认值
F64	R/W	1522	0
含义	设置 / 获取多轴比较触发时输出的脉冲宽度。单位: nsec		

注解:

实际脉冲宽度: 值 x 20

B. 6 轴的特性属性

B. 6. 1 系统

FT_AxFunctionMap

数据类型	R/W	属性 ID	默认值
U32	R	301	—
含义	获取轴支持的功能		

注解:

1: 支持; 0: 不支持

位	说明	宏定义 (头文件 AdvMotDrv.h 中定义)
0	到位	AX_FUNC_INP
1	报警	AX_FUNC_ALM
2	清除伺服驱动中的偏转计数器	AX_FUNC_ERC
3	减速	AX_FUNC_SD
4	硬件限位开关	AX_FUNC_EL
5	软件限位开关	AX_FUNC_SW_EL
6	原点传感器	AX_FUNC_ORG
7	编码 Z 相位传感器	AX_FUNC_EZ
8	背隙校正	AX_FUNC_BACKLASH_CORRECT
9	抑制振动	AX_FUNC_SUPPRESS_VIBRATION
10	返回原点	AX_FUNC_HOME
11	叠加	Ax_FUNC_IMPOSE
12	比较	Ax_FUNC_CMP
13	锁存	Ax_FUNC_LATCH
14	CAMDO	Ax_FUNC_CAMDO
15	外部驱动	Ax_FUNC_EXTDRV
16	同步启停	Ax_FUNC_SIMSTART

17	IN1 停止	AX_FUNC_IN1_STOP
18	IN2 停止	AX_FUNC_IN2_STOP
19	IN3 停止	AX_FUNC_IN3_STOP
20	IN4 停止	AX_FUNC_IN4_STOP
21	IN5 停止	AX_FUNC_IN5_STOP
22~31	未定义	

例程：如下获取轴支持的功能

```
HAND      m_Axhand;      // 轴句柄
U32       AxFunctionMap; // 获取的属性值
Acm_GetU32Property(m_Devhand, FT_AxFunctionMap, AxFunctionMap); // 设置AI范围
If(AxFunctionMap & AX_FUNC_INP)
{ // 支持轴到位功能 }
```

B. 6. 2 速度模式

FT_AxMaxVel

数据类型	R/W	属性 ID	默认值
F64	R	302	20,000,000
含义	获取轴支持的最大速度（单位 Pulse/s）		

FT_AxMaxAcc

数据类型	R/W	属性 ID	默认值
F64	R	303	2,000,000,000
含义	获取轴支持的最大加速度（单位 Pulse/s ² ）		

FT_AxMaxDec

数据类型	R/W	属性 ID	默认值
F64	R	304	2,000,000,000
含义	获取轴支持的最大减速度（单位 Pulse/s ² ）		

FT_AxMaxJerk

数据类型	R/W	属性 ID	默认值
F64	R	305	1
含义	获取轴支持的最大加加速度（单位 Pulse/s ³ ）		

B. 6. 3 脉冲输入

FT_AxPulseInMap

数据类型	R/W	属性 ID	默认值
U32	R	306	0
含义	获取该运动设备支持的脉冲输入特性		

注解：

1：支持； 0：不支持。

位	说明
0	模式
1	逻辑

2	源
3~31	未定义

FT_AxPulseInModeMap

数据类型	R/W	属性 ID	默认值
U32	R	307	0
含义	获取轴支持的脉冲输入模式		

注解:

1: 支持; 0: 不支持。

位	说明	宏定义 (头文件 AdvMotDrv.h 中定义)
0	1X A/B	AB_1X
1	2X A/B	AB_2X
2	4X A/B	AB_4X
3	CW/CCW	I_CW_CCW
4	PULSE/DIR	I_PULSE_DIR
5 ~ 31	未定义	

例程: 获取轴脉冲输入模式

```

HAND      m_Axhand;  // 轴句柄
U32       AxPulseInMode; // 获取的属性值
Acm_GetU32Property(m_Axhand, FT_AxPulseInModeMap, AxPulseInMode);
// 获取脉冲输入模式
If (AxPulseInMode & AB_1X)
{ // 支持 1X A/B 脉冲输入模式 }

```

B. 6. 4 脉冲输出

FT_AxPulseOutMap

数据类型	R/W	属性 ID	默认值
U32	R	308	0
含义	获取该运动设备支持的脉冲输出特性		

注解:

位	说明
0	模式
1	引脚对调
2~31	未定义

FT_AxPulseOutModeMap

数据类型	R/W	属性 ID	默认值
U32	R	309	0
含义	获取该运动设备支持的脉冲输出模式		

注解:

1: 支持; 0: 不支持。

位	说明	宏定义 (头文件 AdvMotDrv.h 中定义)
0	OUT/DIR	OUT_DIR

1	OUT/DIR, OUT 负逻辑	OUT_DIR_OUT_NEG
2	OUT/DIR, DIR 负逻辑	OUT_DIR_DIR_NEG
3	OUT/DIR, OUT&DIR 负逻辑	OUT_DIR_ALL_NEG
4	CW/CCW	O_CW_CCW
5	CW/CCW, CW&CCW 负逻辑	CW_CCW_ALL_NEG
6	A/B 相位	AB_PHASE
7	B/A 相位	BA_PHASE
8	CW/CCW, OUT 负逻辑（不支持）	CW_CCW_OUT_NEG
9	CW/CCW, DIR 负逻辑（不支持）	CW_CCW_DIR_NEG
10~31	未定义	

位	说明	正向		负向	
		OUT 输出	DIR 输出	OUT 输出	DIR 输出
0	OUT/DIR		High		Low
1	OUT/DIR, OUT 负逻辑		High		Low
2	OUT/DIR, DIR 负逻辑		Low		High
3	OUT/DIR, OUT&DIR 负逻辑		Low		High
4	CW/CCW		High	High	
5	CW/CCW, CW&CCW 负逻辑		Low	Low	
6	A/B 相位				
7	B/A 相位				

例程：获取轴支持的脉冲输入模式

```

HAND      m_Axhand;          // 轴句柄
U32      AxPulseOutMode; // 获取的属性值
Acm_GetU32Property(m_Axhand, FT_AxPulseOutModeMap, AxPulseOutMode); // 获取输出模式
If(AxPulseOutMode & OUT_DIR)
{ // 支持 OUT/DIR 脉冲输出模式 }

```

B. 6. 5 报警

FT_AxAImMap

数据类型	R/W	属性 ID	默认值
U32	R	310	—
含义	获取该运动轴支持的报警特性		

注解：

位	说明
0	启用
1	逻辑

2	反应
3~31	未定义

B. 6. 6 到位

FT_AxInpMap

数据类型	R/W	属性 ID	默认值
U32	R	311	—
含义	获取该运动轴支持的到位特性		

注解:

位	说明
0	模式
1	逻辑
2~31	未定义

B. 6. 7 报警清除 (ERC)

FT_AxErcMap

数据类型	R/W	属性 ID	默认值
U32	R	312	0
含义	获取该运动轴支持的 ERC 特性		

注解:

位	说明
0	启用模式
1	逻辑
2	启用时间 (不支持)
3	关闭时间 (不支持)
4~31	未定义

FT_AxErcEnableModeMap

数据类型	R/W	属性 ID	默认值
U32	R	313	0
含义	获取该运动轴支持的 ERC 模式		

注解:

位	说明
0	返回原点后 ERC 输出
1	EMG/ALM/EL 激活时 ERC 输出
2	返回原点后或 EMG/ALM/EL 激活时 ERC 输出
3~31	未定义

B. 6. 8 硬件限位

FT_AxEIMap

数据类型	R/W	属性 ID	默认值
U32	R	317	—
含义	获取该运动轴支持的硬件终端限位 (EL) 特性		

注解:

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

B. 6. 9 软件限位

FT_AxSwMe1Map

数据类型	R/W	属性 ID	默认值
U32	R	318	-
含义	获取运动轴支持的软件负方向限位特性		

注解:

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

FT_AxSwPe1Map

数据类型	R/W	属性 ID	默认值
U32	R	319	-
含义	获取运动轴支持的软件正方向限位特性		

注解:

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

B. 6. 10 原点

FT_AxHomeMap

数据类型	R/W	属性 ID	默认值
U32	R	320	127
含义	获取轴支持的原点相关特性		

注解:

位	说明
0	原点模式
1	ORG 逻辑
2	EZ 逻辑
3	启用复位
4	ORG 高低电平有效
5	HomeOffsetDistance
6	HomeSpeed

FT_AxHomeModeMap

数据类型	R/W	属性 ID	默认值
U32	R	332	—
含义	支持的回原点方式		

注解：

各模式的详细信息请参考章节 9.2.9。

位	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	MP_MODE1_Abs	MODE1_Abs
1	MP_MODE2_Lmt	MODE2_Lmt
2	MP_MODE3_Ref	MODE3_Ref
3	MP_MODE4_Abs_Ref	MODE4_Abs_Ref
4	MP_MODE5_Abs_NegRef	MODE5_Abs_NegRef
5	MP_MODE6_Lmt_Ref	MODE6_Lmt_Ref
6	MP_MODE7_AbsSearch	MODE7_AbsSearch
7	MP_MODE8_LmtSearch	MODE8_LmtSearch
8	MP_MODE9_AbsSearch_Ref	MODE9_AbsSearch_Ref
9	MP_MODE10_AbsSearch_NegRef	MODE10_AbsSearch_NegRef
10	MP_MODE11_LmtSearch_Ref	MODE11_LmtSearch_Ref
11	MP_MODE12_AbsSearchReFind	MODE12_AbsSearchReFind
12	MP_MODE13_LmtSearchReFind	MODE13_LmtSearchReFind
13	MP_MODE14_AbsSearchReFind_Ref	MODE14_AbsSearchReFind_Ref
14	MP_MODE15_AbsSearchReFind_NegRef	MODE15_AbsSearchReFind_NegRef
15	MP_MODE16_LmtSearchReFind_Ref	MODE16_LmtSearchReFind_Ref

例程：获取轴支持的回原点模式

```
HAND    m_Axhand;    // 轴句柄
U32     AxHomeMode;  // 获取的属性值
Acm_GetU32Property(m_Axhand, FT_AxHomeModeMap, AxHomeMode); // 获取 Home 模式
If(AxHomeMode & MODE1_Abs)
{ // 支持 MP_MODE1_Abs 回原点模式 }
If(AxHomeMode & MODE2_Lmt)
{ // 支持 MP_MODE2_Lmt 回原点模式 }
```

B.6.11 背隙补偿

FT_AxBackLashMap

数据类型	R/W	属性 ID	默认值
U32	R	321	—
含义	获取该运动轴支持的背隙补偿特性		

注解：

位	说明
0	启用
1	值
2~31	未定义

B. 6. 12 比较

FT_AxCompareMap

数据类型	R/W	属性 ID	默认值
U32	R	324	—
含义	获取轴支持的比较特性		
注解:			
位	说明		
0	启用		
1	逻辑		
2	CmpSrc		
3	CmpMethod		
4	CmpPulseMode		
5	CmpPulseWidth		
6~31	未定义		

B. 6. 13 锁存

FT_AxLatchMap

数据类型	R/W	属性 ID	默认值
U32	R	325	—
含义	获取轴支持的锁存特性		
注解:			
位	说明		
0	启用		
1	逻辑		
2~31	未定义		

B. 6. 14 Cam DO

FT_AxCamDOMap

数据类型	R/W	属性 ID	默认值
U32	R	326	—
含义	获取轴支持的 Cam DO 特性		
注解:			
位	说明		
0	启用		
1	CamDOLogic		
2	CamDOCmpSrc		
3	CamDOLoLimit		
4	CamDOHiLimit		
5	CamDOMode		
6	CamDODir		
7	CamDOAssign		
8~31	未定义		

B. 6. 15 外部驱动

FT_AxExtDriveMap

数据类型	R/W	属性 ID	默认值
U32	R	327	0
含义	获取轴支持的外部驱动特性		

注解:

位	说明
0	ExtMasterSrc
1	ExtSelEnable
2	ExtPulseNum
3	ExtPulseMode
4	ExtPresetNum
5~31	未定义

FT_AxExtMasterSrcMap

数据类型	R/W	属性 ID	默认值
U32	R	328	0
含义	获取轴支持的外部源		

注解:

位	说明
0	轴 0
1	轴 1
2	轴 2
3	轴 3
4~31	未定义

B. 6. 16 Jog

FT_AxJogMap

数据类型	R/W	属性 ID	默认值
U32	R	339	7
含义	获取轴支持的 Jog 特性		

注解:

1: 支持; 0: 不支持。

位	说明
0	Jog Speed
1	JogVltime
2	Soft jog
3	Jog pin assign

B. 6. 17 Aux/Gen 输出

FT_AxGenDMap

数据类型	R/W	属性 ID	默认值
U32	R	329	0
含义	获取轴支持的通用输出 OUT4~OUT7		

注解:

位	说明
0	OUT4/CAM_DO
1	OUT5/TRIG_Position
2	OUT6/SVON
3	OUT7/ERC
4~31	未定义

FT_AxGenDMap

数据类型	R/W	属性 ID	默认值
U32	R	330	0
含义	获取轴支持的通用输入 IN1, IN2, IN4, IN5		

注解:

位	说明
0	IN1/LTC
1	IN2/RDY
2	IN4/JOG+
3	IN5/JOG-
4~31	未定义

B. 6. 18同步启停

FT_AxSimStartSourceMap

数据类型	R/W	属性 ID	默认值
U32	R	331	-
含义	轴支持的同步启动模式。请参考 CFG_AxSimStartSource		

注解:

位	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	从设备 STA 针脚上启动同步运动模式	SimSrc_STA
1~7	未定义	
8	从轴 _0 的比较信号启动同步运动	SimSrc_TRIGP_AX0
9	从轴 _1 的比较信号启动同步运动	SimSrc_TRIGP_AX1
10	从轴 _2 的比较信号启动同步运动	SimSrc_TRIGP_AX2
11	从轴 _3 的比较信号启动同步运动	SimSrc_TRIGP_AX3
12	从轴 _4 的比较信号启动同步运动	SimSrc_TRIGP_AX4
13	从轴 _5 的比较信号启动同步运动	SimSrc_TRIGP_AX5
14	从轴 _6 的比较信号启动同步运动	SimSrc_TRIGP_AX6
15	从轴 _7 的比较信号启动同步运动	SimSrc_TRIGP_AX7
16	当轴 _0 停止时启动同步运动	SimSrc_STOP_AX0
17	当轴 _1 停止时启动同步运动	SimSrc_STOP_AX1
18	当轴 _2 停止时启动同步运动	SimSrc_STOP_AX2
19	当轴 _3 停止时启动同步运动	SimSrc_STOP_AX3
20	当轴 _4 停止时启动同步运动	SimSrc_STOP_AX4
21	当轴 _5 停止时启动同步运动	SimSrc_STOP_AX5
22	当轴 _6 停止时启动同步运动	SimSrc_STOP_AX6
23	当轴 _7 停止时启动同步运动	SimSrc_STOP_AX7

24~31 未定义

B. 6. 19 触发停止

FT_AxIN1Map

数据类型	R/W	属性 ID	默认值
U32	R	333	—
含义	IN1 触发停止功能特性		
注解：			
值	说明		
0	启用 / 禁用停止功能		
1	停止电平：高电平或低电平		
2	停止模式：减速 / 突然停止		

FT_AxIN2Map

数据类型	R/W	属性 ID	默认值
U32	R	334	0
含义	IN2 触发停止功能特性		
注解:			
值	说明		
0	启用 / 禁用停止功能		
1	停止电平: 高电平或低电平		
2	停止模式: 减速 / 突然停止		

FT_AxIN4Map

数据类型	R/W	属性 ID	默认值
U32	R	336	0
含义	IN4 触发停止功能特性		
注解:			
值	说明		
0	启用 / 禁用停止功能		
1	停止电平: 高电平或低电平		
2	停止模式: 减速 / 立刻停止		

FT_AxIN5Map

数据类型	R/W	属性 ID	默认值
U32	R	337	0
含义	IN5 触发停止功能特性		
注解:			
值	说明		
0	启用 / 禁用停止功能		
1	停止电平: 高电平或低电平		
2	停止模式: 减速 / 突然停止		

B. 7 轴的配置属性

B. 7. 1 系统

CFG_AxPPU

数据类型	R/W	属性 ID	默认值
U32	R/W	551	1
含义	获取轴的物理 ID		

注解:

单位脉冲 (PPU) 是一个虚拟单位。该属性值必须大于 0。

该属性值的变化将影响 CFG_AxMaxVel、CFG_AxMaxAcc、CFG_AxMaxDec、PAR_AxVelHigh、PAR_AxVelLow、PAR_AxAcc、PAR_AxDec、PAR_GpVelHigh、PAR_GpVelLow、PAR_GpAcc、PAR_GpDec 和 PAR_HomeCrossDistance。

CFG_AxPhyID

数据类型	R/W	属性 ID	默认值
U32	R	552	只读
含义	获取轴的物理 ID		

注解:

值	说明
0~31	0 轴 ~31 轴

B. 7. 2 速度模式

CFG_AxMaxVel

数据类型	R/W	属性 ID	默认值
F64	R/W	553	20, 000, 000
含义	配置运动轴的最大速度。单位 PPU/s		

注解:

最大值 = $FT_AxMaxVel / CFG_AxPPU$

最小值 = 1 / CFG_AxPPU

CFG_AxMaxAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	554	2, 000, 000, 000
含义	配置运动轴的最大加速度。单位 PPU/s^2		

注解:

最大值 = $FT_AxMaxAcc / CFG_AxPPU$

最小值 = 1 / CFG_AxPPU

CFG_AxMaxDec

数据类型	R/W	属性 ID	默认值
F64	R/W	555	2, 000, 000, 000
含义	配置运动轴的最大减速度。单位 PPU/s^2		

注解:

最大值 = $FT_AxMaxDec / CFG_AxPPU$

最小值 = $1 / CFG_AxPPU$

CFG_AxMaxJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	556	1
含义	获取运动轴的最大加加速度，单位 PPU/s^3		

B. 7. 3 脉冲输入

CFG_AxPulseInMode

数据类型	R/W	属性 ID	默认值
U32	R/W	557	2
含义	设置 / 获取编码器反馈脉冲输入模式		

注解：

设置、获取的属性值代表的脉冲输入模式如下表所示：

值	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	1XAB	AB_1X
1	2XAB	AB_2X
2	4XAB（默认值）	AB_4X
3	CCW/CW	I_CW_CCW
4	PULSE/DIR（不支持）	I_PULSE_DIR

仅特定 EtherCAT 从站支持此属性。

CFG_AxPulseInLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	558	0
含义	设置 / 获取编码器反馈的逻辑		

注解：

设置、获取的属性值代表的编码器反馈逻辑如下表所示：

值	说明
0	不倒转方向（默认值）
1	倒转方向

仅特定 EtherCAT 从站支持此属性。

CFG_AxPulseInMaxFreq

数据类型	R/W	属性 ID	默认值
U32	R/W	632	1
含义	设置 / 获取频率中编码最大脉冲		

注解：

设置、获取的属性值代表的编码最大脉冲频率如下表所示：

值	说明
0	500KHz
1	1MHz（默认值）

2	2MHz
3	4MHz

仅特定 EtherCAT 从站支持此属性。

B. 7. 4 脉冲输出

CFG_AxPulseOutMode

数据类型	R/W	属性 ID	默认值
U32	R/W	560	16
含义	设置 / 获取命令脉冲输出模式		

注解：

脉冲输出模式如下表所示：

值	说明	宏定义（头文件 AdvMotDrv.h 中定义）
1	OUT/DIR	OUT_DIR
2	OUT/DIR, OUT 负逻辑	OUT_DIR_OUT_NEG
4	OUT/DIR, DIR 负逻辑	OUT_DIR_DIR_NEG
8	OUT/DIR, OUT&DIR 负逻辑	OUT_DIR_ALL_NEG
16	CW/CCW（默认值）	0_CW_CCW
32	CW/CCW, CW&CCW 负逻辑	CW_CCW_ALL_NEG
256	CW/CCW, OUT 负逻辑	CW_CCW_OUT_NEG
512	CW/CCW, DIR 负逻辑	CW_CCW_DIR_NEG

仅特定 EtherCAT 从站支持此属性。

CFG_AxPulseOutReverse

数据类型	R/W	属性 ID	默认值
U32	R/W	693	0
含义	启用 / 禁用 Pulse Out 引脚对调功能		

注解：

设置、获取的值如下表所示：

值	说明
0	禁用（默认值）
1	启用

仅特定 EtherCAT 从站支持此属性。

B. 7. 5 报警

CFG_AxAlmEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	561	0
含义	启用 / 禁用运动报警功能，报警是当电机驱动处于报警状态时，电机驱动生成的信号		

注解：

值	说明
0	禁用
1	启用

CFG_AxAlmLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	562	1
含义	设置 / 获取报警信号的有效逻辑电平		

注解:

值	说明
0	低准位
1	高准位

CFG_AxAlmReact

数据类型	R/W	属性 ID	默认值
U32	R/W	563	1
含义	设置 / 获取接收 ALARM 信号时的停止模式		

注解:

值	说明
0	电机立刻停止
1	电机减速, 然后停止

CFG_AxCmdErrorProtection

数据类型	R/W	属性 ID	默认值
U32	R/W	708	1,000
含义	设置 / 获取指令错误保护值		

注解:

当切换操作模式时（例如从原点复归模式切换至 P2P 模式），若理论为止与实际位置的差值超过指令错误保护值，则会提示错误。

B. 7.6 伺服到位

CFG_AxInpEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	564	0
含义	启用、禁用到位功能		

注解:

值	说明
0	禁用
1	启用

CFG_AxInpLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	565	1
含义	设置 / 获取到位信号的有效逻辑电平		

注解:

值	说明
---	----

0	低准位
1	高准位（默认值）

B. 7. 7 硬件限位

CFG_AxEIEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	574	1
含义	启用 / 获取硬件限位功能		

注解：

设置、获取的值说明如下表所示：

值	说明
0	禁用
1	启用（默认值）

CFG_AxEILogic

数据类型	R/W	属性 ID	默认值
U32	R/W	575	0
含义	设置 / 获取硬件限位的逻辑准位		

注解：

值	说明
0	禁用
1	启用

CFG_AxEIReact

数据类型	R/W	属性 ID	默认值
U32	R/W	576	0
含义	设置 / 获取 EL 信号的反应模式		

注解：

值	说明
0	电机立即停止
1	电机减速，然后停止

B. 7. 8 软件限位

CFG_AxSwMeIEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	577	0
含义	启用 / 禁用负方向软件限位功能		

注解：

值	说明
0	禁用（默认值）
1	启用

CFG_AxSwPelEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	578	0
含义	启用 / 禁用正方向软件限位功能		

注解:

值	说明
0	禁用
1	启用

CFG_AxSwMelReact

数据类型	R/W	属性 ID	默认值
U32	R/W	579	1
含义	设置 / 获取负方向软件限位的反应模式		

注解:

值	说明
0	电机立即停止
1	电机减速，然后停止

CFG_AxSwPelReact

数据类型	R/W	属性 ID	默认值
U32	R/W	580	1
含义	设置 / 获取正方向软件限位的反应模式		

注解:

值	说明
0	电机立即停止
1	电机减速，然后停止

CFG_AxSwMelValue

数据类型	R/W	属性 ID	默认值
I32	R/W	581	—
含义	设置 / 获取负方向软件限位的值		

注解:

范围: -2, 147, 483, 647 ~ +2, 147, 483, 647。

CFG_AxSwPelValue

数据类型	R/W	属性 ID	默认值
I32	R/W	582	—
含义	设置 / 获取正方向软件限位的值		

注解:

范围: -2, 147, 483, 647 ~ +2, 147, 483, 647。

B. 7. 9 原点

CFG_AxOrgLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	589	0
含义	设置 / 获取 ORG 信号的有效逻辑电平		

注解:

值	说明
0	低准位
1	高准位

CFG_AxEzLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	591	0
含义	设置 / 获取 EZ 信号的有效逻辑电平		

注解:

值	说明
0	低准位
1	高准位

CFG_AxHomeResetEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	602	1
含义	回原点后，启用 / 禁用逻辑计数器的复位功能		

注解:

值	说明
0	禁用
1	启用

CFG_AxOrgReact

数据类型	R/W	属性 ID	默认值
U32	R/W	634	1
含义	设定回原点结束时的行为模式		

注解:

值	说明
0	立即停止
1	减速停止

B. 7. 10 背隙补偿

CFG_AxBacklashEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	593	0
含义	启用 / 禁用背隙补偿		

注解：

值	说明
0	禁用（默认值）
1	启用

CFG_AxBacklashPulses

数据类型	R/W	属性 ID	默认值
U32	R/W	594	10
含义	设置 / 获取补偿脉冲个数（单位：脉冲）		

注解：

范围为 0 ~ 4095。当方向发生变化时，轴在发送命令前会先输出背隙补偿脉冲。

CFG_AxBacklashVel

数据类型	R/W	属性 ID	默认值
U32	R/W	630	1,000
含义	启用 / 禁用正确背隙补偿的速度（速度：Pulse/s）		

B. 7. 11 比较

CFG_AxCmpSrc

数据类型	R/W	属性 ID	默认值
U32	R/W	603	0
含义	设置 / 获取比较源		

注解：

设置、获取的值说明如下表所示：

值	说明
0	理论位置（默认值）
1	实际位置

仅特定 EtherCAT 从站支持此属性。

CFG_AxCmpMethod

数据类型	R/W	属性 ID	默认值
U32	R/W	604	0
含义	设置 / 获取比较方法		

注解：

值	说明
0	>= 位置计数器（默认值）
1	<= 位置计数器
2	= 计数器（无方向）（不支持）

仅特定 EtherCAT 从站支持此属性。

CFG_AxCmpPulseLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	606	1
含义	设置 / 获取比较信号的逻辑电平		

注解:

设置、获取的值说明如下表所示:

值	说明
0	低准位
1	高准位 (默认值)

仅特定 EtherCAT 从站支持此属性。

CFG_AxCmpPulseWidth

数据类型	R/W	属性 ID	默认值
U32	R/W	607	0
含义	设置 / 获取比较信号延迟的宽度		

注解:

设置、获取的值说明如下表所示:

值	说明
0	5 Microsecond(us)
1	10 Microsecond(us)
2	20 Microsecond(us)
3	50 Microsecond(us)
4	100 Microsecond(us)
5	200 Microsecond(us)
6	500 Microsecond(us)
7	1000 Microsecond(us)

仅特定 EtherCAT 从站支持此属性。

CFG_AxCmpEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	608	0
含义	启用 / 禁用轴比较功能		

注解:

设置、获取的值说明如下表所示:

值	说明
0	禁用 (默认值)
1	启用

仅特定 EtherCAT 从站支持此属性。

CFG_AxCmpPulseMode

数据类型	R/W	属性 ID	默认值
U32	R/W	605	0
含义	设置 / 获取轴比较模式		

注解：

设置、获取的值说明如下表所示：

值	说明
0	脉冲模式（默认值）
1	开关模式
2	PWM 模式

仅特定 EtherCAT 从站支持此属性。

B. 7. 12 多轴比较

CFG_AxisMultiCmpDeviation

数据类型	R/W	属性 ID	默认值
U32	R/W	687	0
含义	设置 / 获取多轴比较时各轴进行比较的误差范围。范围：0 ~ 31		

注解：

仅特定 EtherCAT 从站支持此属性。

B. 7. 13 锁存

CFG_AxLatchLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	601	1
含义	设置 / 获取锁存信号的逻辑电平		

注解：

当锁存触发时，将锁存理论位置、实际位置和滞后距离。

值	说明
0	低准位
1	高准位

CFG_AxLatchEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	631	0
含义	启用 / 禁用锁存功能		

注解：

值	说明
0	禁用（默认值）
1	启用

B. 7. 14 连续锁存

CFG_AxLatchBufEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	647	0
含义	设定 / 读取 Latch Buffer 功能是否开启 / 关闭		

注解：

设置、获取的值说明如下表所示：

值	说明
0	禁用（默认值）
1	启用

仅特定 EtherCAT 从站支持此属性。

CFG_AxLatchBufMinDist

数据类型	R/W	属性 ID	默认值
U32	R/W	648	1,000
含义	设定 / 读取相邻两笔锁存资料的间隔距离		

注解：

仅特定 EtherCAT 从站支持此属性。

CFG_AxLatchBufEventNum

数据类型	R/W	属性 ID	默认值
U32	R/W	649	128
含义	设定 / 读取产生事件的数量		

注解：

仅特定 EtherCAT 从站支持此属性。

CFG_AxLatchBufSource

数据类型	R/W	属性 ID	默认值
U32	R/W	650	0
含义	设定 / 读取某轴的 Latch Buffer 中的资料类型		

注解：

设置、获取的值说明如下表所示：

值	说明	宏定义（头文件 AdvMotDrv.h 中定义）
0	理论位置（默认值）	SRC_COMMAND_POSITION
1	实际位置	SRC_ACTUAL_POSITION

仅特定 EtherCAT 从站支持此属性。

CFG_AxLatchBufAxisID

数据类型	R/W	属性 ID	默认值
U32	R/W	651	0
含义	设定 / 获取被锁存数据的轴的 ID		

注解：

通过该属性可设定一个轴锁存另一轴的数据。

仅特定 EtherCAT 从站支持此属性。

CFG_AxLatchBufEdge

数据类型	R/W	属性 ID	默认值
U32	R/W	652	0
含义	设置 / 获取 Latch Buffer 信号的触发沿		

注解：

设置、获取的值说明如下表所示：

值	说明
0	上升沿
1	下降沿
2	上升与下降沿

仅特定 EtherCAT 从站支持此属性。

B. 7. 15 外部驱动

CFG_AxJogPAssign

数据类型	R/W	属性 ID	默认值
U32	R/W	721	–
含义	设置 / 获取作为外部 Jog+ 的 DI 通道的数据。对于 PIC-1203 而言，用户可以将连接至运动环的 DI 通道作为 JOG+ 针脚，并如同其他研华运动控制卡一样去进行操作。		

注解：

该属性只能连接至运动环的 DI 通道。

CFG_AxJogNAssign

数据类型	R/W	属性 ID	默认值
U32	R/W	722	–
含义	设置 / 获取作为外部 Jog- 的 DI 通道的数据。对于 PIC-1203 而言，用户可以将连接至运动环的 DI 通道作为 JOG- 针脚，操作方式与其他研华运动控制卡相同。		

注解：

只支持连接至运动环的 DI 通道。

CFG_AxJogVTime

数据类型	R/W	属性 ID	默认值
U32	R/W	692	5,000
含义	Jog 执行时，低速运转保持的时间，时间结束后才开始加速。单位：ms		

注解：

范围：0~+4, 294, 967, 295

CFG_AxJogVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	695	2,000
含义	设置 / 获取 Jog 执行时轴的低速度（起始速度）。单位：PPU/s		

注解：

范围：[1, MaxVel]

CFG_AxJogVelHigh

数据类型	R/W	属性 ID	默认值
------	-----	-------	-----

F64	R/W	696	8,000
含义	设置 / 获取 Jog 执行时轴的高速度（运行速度）。单位：PPU/s		

注解：

范围：[1, MaxVel]

CFG_AxJogAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	697	10,000
含义	设置 / 获取 Jog 执行时轴的加速度。单位：PPU/s ²		

注解：

范围：[1, MaxAcc]

CFG_AxJogDec

数据类型	R/W	属性 ID	默认值
F64	R/W	698	10,000
含义	设置 / 获取 Jog 执行时轴的减速度。单位：PPU/s ²		

注解：

范围：[1, MaxDec]

CFG_AxJogJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	699	0
含义	设置 / 获取 Jog 执行时的速度曲线类型：T 型或 S 型		

注解：

只支持 T 型曲线。

值	说明
0	T 型
1	S 型（不支持）

B. 7. 16 凸轮区间 D0

CFG_AxCamD0Assign

数据类型	R/W	属性 ID	默认值
U32	R/W	723	—
含义	设置 / 获取 D0 通道以输出 CamD0 信号。对于 PCI-1203 而言，用户可以将连接至运动环的 D0 通道作为 CAM_D0 针脚，操作方式与其他研华运动控制卡相同。		

注解：

只支持连接至运动环的 D0 通道。

CFG_AxCamD0Enable

数据类型	R/W	属性 ID	默认值
U32	R/W	622	—
含义	启用 / 禁用 Cam D0		

注解：

值	说明
0	禁用
1	启用

CFG_AxCamDOLoLimit

数据类型	R/W	属性 ID	默认值
U32	R/W	623	10,000
含义	设置 / 获取凸轮区间的低限位		

注解：

启用 CamD0，当理论 / 实际位置处于低限位值和高限位值之间时，将触发 CamD0 信号。

范围：-2,147,483,647 ~ +2,147,483,647。

CFG_AxCamDOHiLimit

数据类型	R/W	属性 ID	默认值
U32	R/W	624	10,000
含义	设置 / 获取凸轮区间的高限位		

注解：

启用 CamD0，当理论 / 实际位置处于低限位值和高限位值之间时，将触发 CamD0 信号。

范围：-2,147,483,647 ~ +2,147,483,647。

CFG_AxCamDOCmpSrc

数据类型	R/W	属性 ID	默认值
U32	R/W	627	0
含义	设置 / 获取凸轮区间的比较源		

注解：

值	说明
0	理论位置
1	实际位置（不支持）

CFG_AxCamDOLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	628	1
含义	设置 / 获取 CamD0 的逻辑准位		

注解：

值	说明
0	低准位
1	高准位

B. 7. 17 轴单圈脉冲数

CFG_AxModuleRange

数据类型	R/W	属性 ID	默认值
------	-----	-------	-----

U32	R/W	629	0
含义	设置当轴旋转 360 度时的脉冲个数		

注解：

该值必须为 4 的倍数。

该值用于切向运动和 E-Cam 运动。请参考 Acm_AxTangentInGp、Acm_DevDownloadCAMTable 和 Acm_AxCamInAx。

范围：0 ~ 8,000,000。

B. 7. 18 同步启用

CFG_AxSimStartSource

数据类型	R/W	属性 ID	默认值
U32	R/W	633	1
含义	设置 / 获取当前轴的同步启停模式		

注解：

值	说明
0	禁用
1	从设备 STA 针脚的信号上启动同步运动模式（默认）
256	从轴 _0 的比较信号启动同步运动
512	从轴 _1 的比较信号启动同步运动
1024	从轴 _2 的比较信号启动同步运动
2048	从轴 _3 的比较信号启动同步运动
4096	从轴 _4 的比较信号启动同步运动
8192	从轴 _5 的比较信号启动同步运动
16384	从轴 _6 的比较信号启动同步运动
32768	从轴 _7 的比较信号启动同步运动
65536	当轴 _0 停止时启动同步运动
131072	当轴 _1 停止时启动同步运动
262144	当轴 _2 停止时启动同步运动
524288	当轴 _3 停止时启动同步运动
1048576	当轴 _4 停止时启动同步运动
2097152	当轴 _5 停止时启动同步运动
4194304	当轴 _6 停止时启动同步运动
8388608	当轴 _7 停止时启动同步运动

B. 7. 19 龙门

CFG_AxGantryMaxDiffValue

数据类型	R/W	属性 ID	默认值
F64	R/W	658	—
含义	启用 / 禁用错误护功能的值		

注解：

从轴将于主轴同步运动。为从轴设置错误保护值，若主轴与从轴之间的距离差值大于该值，则同步运动将停止。

B. 7. 20 触发停止

CFG_AxIN1StopAssign

数据类型	R/W	属性 ID	默认值
U32	R/W	724	0
含义	设置 / 获取作为 IN1 的 DI 通道。对于 PCI-1203 而言，用户可以将连接至运动环的 DI 通道作为通用输入 IN1，操作方式与其他研华运动控制卡相同。		

注解：

仅支持连接至运动环的 DI 通道。

CFG_AxIN1StopEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	635	0
含义	启用 / 禁用 IN1 触发停止功能		

注解：

值	说明
0	启用
1	禁用

CFG_AxIN1StopReact

数据类型	R/W	属性 ID	默认值
U32	R/W	636	1
含义	设定 / 获取 IN1 触发时的停止模式		

注解：

值	说明
0	立即停止
1	减速停止

CFG_AxIN1StopLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	637	0
含义	设定 / 获取 IN1 触发停止功能的逻辑准位		

注解：

值	说明
0	低准位
1	高准位

B. 8 轴的参数属性

B. 8. 1 速度参数

PAR_AxVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	401	2, 000
含义	设置 / 获取轴的起始速度。单位：PPU/s		

注解：

该属性值必须小于或等于 PAR_AxVelHigh。

PAR_AxVelHigh

数据类型	R/W	属性 ID	默认值
F64	R/W	402	8, 000
含义	设置 / 获取轴的运行速度。单位：PPU/s		

注解：

该属性值必须小于 CFG_AxMaxVel 且大于 PAR_AxVelLow。

PAR_AxAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	403	10, 000
含义	设置 / 获取轴的加速度。单位：PPU/s ²		

注解：

该属性值必须小于或等于 CFG_AxMaxAcc。

PAR_AxDec

数据类型	R/W	属性 ID	默认值
F64	R/W	404	10, 000
含义	设置 / 获取轴的减速度。单位：PPU/s ²		

注解：

该属性值必须小于或等于 CFG_AxMaxDec。

PAR_AxJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	405	0
含义	设置速度曲线类型： T 型曲线 S 型曲线		

注解：

实际加速度由驱动程序计算得出。

当 PAR_AxJerk 设置为 1 时，

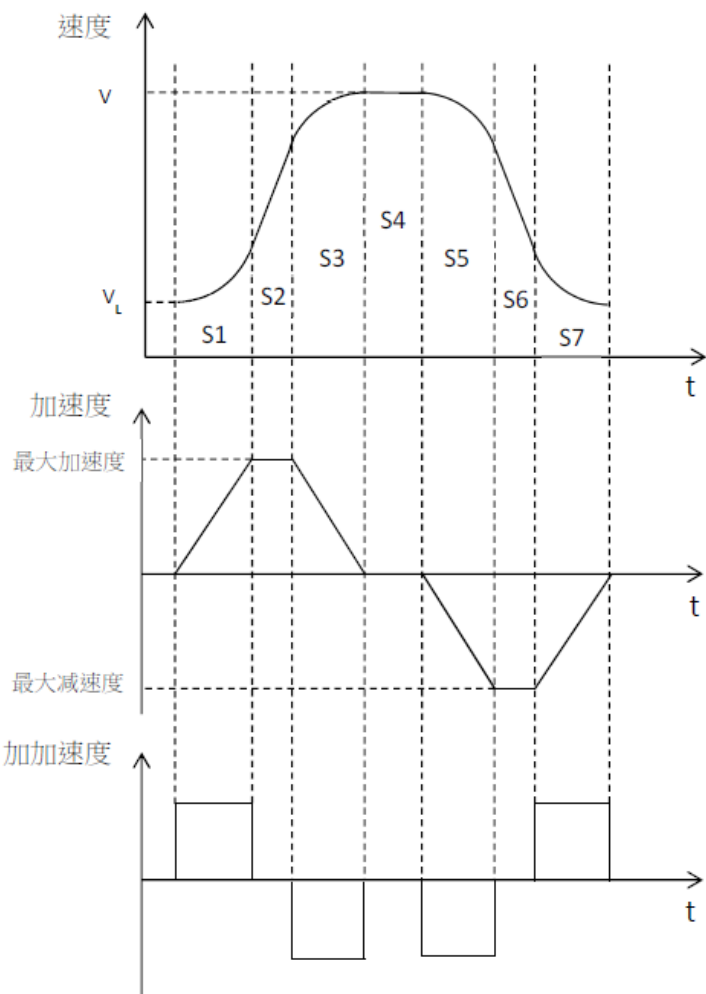
PAR_AxAcc 表示最大加速度而非加速度

PAR_AxDec 表示最大减速度而非减速度

PAR_AxJerkFactor

数据类型	R/W	属性 ID	默认值
F64	R/W	443	50.0
含义	设置 / 获取 S 型曲线第一段的时间比例		

注解：
速度 / 加速度 / 加加速度曲线如下图所示。速度分布可以分为 7 个部分，并且该属性用以指定第一部分时间（S1 或 S5）与完整加速部分时间（S1 + S2 + S3）或完整减速部分时间（S5 + S6 + S7）的比率。S1 和 S3 的时间（S5 和 S7）相同，比率值在 1 到 50 之间。如果 PAR_AxJerkFactor 设置为 50，则速度曲线将是纯 S 型曲线，其加速度曲线呈三角形。



B. 8. 2 原点

PAR_AxHomeVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	415	2,000
含义	Home 执行时的初速度。单位：PPU/s		

注解：
该属性值必须小于或等于 CFG_AxMaxVel。

PAR_AxHomeVelHigh

数据类型	R/W	属性 ID	默认值
------	-----	-------	-----

F64	R/W	416	8,000
含义	Home 执行时的运行速度。单位：PPU/s		

注解：

该属性值必须小于或等于 CFG_AxMaxVel。

PAR_AxHomeAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	417	10,000
含义	Home 执行时的加速度。单位：PPU/s ²		

注解：

该属性值必须小于或等于 CFG_AxMaxAcc。

PAR_AxHomeDec

数据类型	R/W	属性 ID	默认值
F64	R/W	418	10,000
含义	Home 执行时的减速度。单位：PPU/s ²		

注解：

该属性值必须小于或等于 CFG_AxMaxDec。

PAR_AxHomeJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	419	0
含义	设置回原点时速度曲线类型		

注解：

值	说明
0	T 型曲线
1	S 型曲线

PAR_AxHomeJerkFactor

数据类型	R/W	属性 ID	默认值
F64	R/W	444	50.0
含义	设置 / 获取回原点时 S 型曲线第一段的时间比例		

注解：

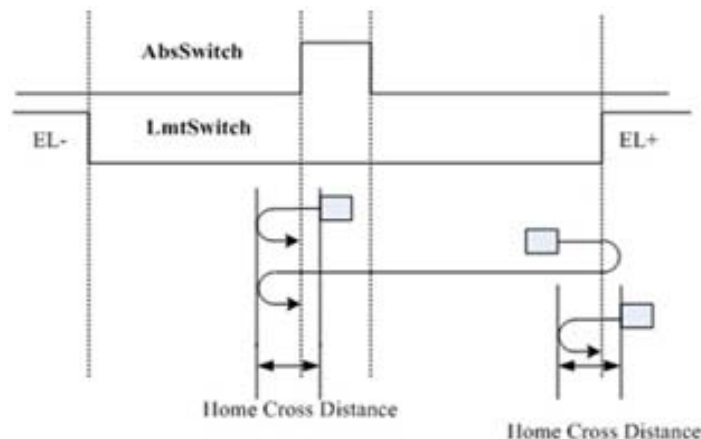
此属性仅在调用 Acm_AxMoveHome 时使用

PAR_AxHomeCrossDistance

数据类型	R/W	属性 ID	默认值
F64	R/W	408	10,000
含义	设置远点跨越距离。单位：PPU		

注解：

该属性值必须大于 0。



PAR_AxHomeExSwitchMode

数据类型	R/W	属性 ID	默认值
F64	R/W	407	0
含义	设置 Acm_AxHomeEx 的停止条件		

注解：

值	定义	说明
0	Level On	传感器开启（激活）
1	Level Off	传感器关闭（未激活）
2	Rising Edge	传感器从关闭到开启的转换
3	Falling Edge	传感器从开启到关闭的转换

B. 9 群组的配置属性

B. 9. 1 系统

CFG_GpAxesInGroup

数据类型	R/W	属性 ID	默认值
U32	R	806	–
含义	获取关于哪些轴在该群组中的信息		

注解：

位	说明
0~31 ‘	0 轴 ~31 轴

B. 9. 2 路径

CFG_GpBldTime

数据类型	R/W	属性 ID	默认值
U32	R/W	808	10, 000
含义	设置 / 获取添加的路径与前一个路径的交接时间		

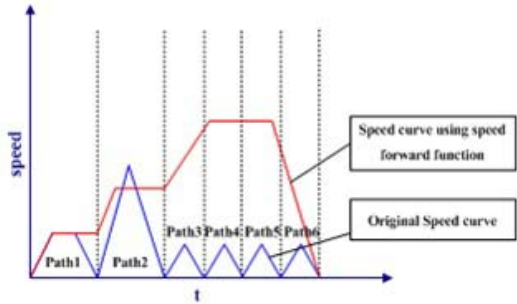
注解：

该值范围为 0 ~ 65, 535，且必须为 2 的倍数。单位：ms

CFG_GpSFEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	809	–
含义	启用 / 禁用速度前瞻功能		

注解：
不支持 S 形速度曲线。该模式下，Acm_GpAddPath 函数中设置的速度参数无效，仅使用群组的速度设置。



B. 10 群组的参数属性

B. 10. 1 速度模式

PAR_GpVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	701	添加的第一个轴的初始速度 (PAR_AxVelLow)
含义	设置该群组的起始速度。单位 :PPU/s		

注解：
该属性值必须小于或等于 PAR_GpVelHigh。

PAR_GpVelHigh

数据类型	R/W	属性 ID	默认值
F64	R/W	702	添加的第一个轴的执行速度 (PAR_AxVelHigh)
含义	设置该群组的运行速度。单位 :PPU/s		

注解：
该属性值必须小于添加第一个轴的 CFG_AxMaxVel 且大于 PAR_AxVelHigh。

PAR_GpAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	703	添加的第一个轴的加速度 (PAR_AxAcc)
含义	设置该群组的加速度。单位 :PPU/s ²		

注解：
该属性值必须小于或等于添加的第一个轴的 CFG_AxMaxAcc。

PAR_GpDec

数据类型	R/W	属性 ID	默认值
F64	R/W	704	添加的第一个轴的减速度 (PAR_AxDec)
含义	设置该群组的减速度。单位 :PPU/S ²		

注解:

该属性值必须小于或等于添加的第一个轴的 CFG_AxMaxDec。

PAR_GpJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	705	添加的第一个轴的加速度
含义	设置速度曲线类型: S 型曲线 T 型曲线		

注解:

如果 PAR_GpJerk 设置为 1, PAR_GpAcc 表示最大加速度, 而非加速度; PAR_GpDec 表示最大减速度, 而非减速度。

PAR_GpJerkFactor

数据类型	R/W	属性 ID	默认值
F64	R/W	710	50.0
含义	设置 / 获取 S 型曲线第一段的时间比例		

注解:

B. 10.2 系统

PAR_GpGroupID

数据类型	R/W	属性 ID	默认值
U32	R	706	—
含义	通过 GroupHandle 获取 GroupID		

注解:

PCI-1203 只有 6 个可用的 GroupID, 分别为 0、1、……4 和 5。用户无法同时操作 6 个以上群组, 因此若已有 6 个群组, 则必须关闭一个才能创建新的群组。

B. 10.3 路径

PAR_GpRefPlane

数据类型	R/W	属性 ID	默认值
U32	R/W	709	0
含义	设置 / 获取螺旋运动和圆弧插补的参考平面		

注解:

值	说明
0	XY 平面
1	YZ 平面
2	XZ 平面

附录 C

错误代码

C.1 错误代码

调用函数列表中的函数时，每个函数都将获得一个返回代码，表示调用结果。用户可根据 Acm_GetErrorMessage 函数查看返回的错误代码对应的错误信息，以便进行修改。

返回值	0x00000000
信息	SUCCESS

下面列出调用函数时常见的错误信息，并说明产生该错误的可能原因

C.2 常见错误

错误代码	0x80000000
错误信息	InvalidDevNumber
说明	无效的设备号
相关函数	Acm_AxCamInAx、Acm_AxTangentInGp、Acm_AxGantryInAx、AxLoadConfig
实例	调用 Acm_AxCamInAx 函数建立主从轴关系，当设置的从轴不属于该设备时，提示错误

错误代码	0x80000004
错误信息	MemAllocateFailed
说明	Windows 全局堆内存分配失败
相关函数	Acm_DevOpen、Acm_GpLoadPath、Acm_DevLoadConfig、 Acm_DevLoadCAMTableFile、Acm_GpAddAxis、Acm_AxOpen
实例	

错误代码	0x80000005
错误信息	InvalidHandle
说明	无效的 Handle
相关函数	所有需要输入设备的 Handle、轴 Handle、组 Handle 的函数
实例	调用 Acm_AxOpen 函数打开设备，当输入的设备 Handle 不存在或不正确时，提示错误

错误代码	0x80000006
错误信息	CreateFileFailed
说明	创建文件失败
相关函数	Acm_DevOpen
实例	

错误代码	0x80000009
错误信息	InvalidInputParam
说明	无效的输入参数
相关函数	Acm_EnableMotionEvent、Acm_DevDownloadCAMTable、Acm_DaqDiGetByte 等
实例	使用 Acm_DaqDoSetByte API 设置 DO 值时，当输入参数 DOPort 大于连接至设备的最大 DO Port (FT_DaqDoMaxChan/8)，将提示该错误。

错误代码	0x8000000a
错误信息	PropertyIDNotSupport
说明	属性 ID 不支持
相关函数	设置 / 获取属性函数
实例	当板卡不支持该属性时，提示该错误

错误代码	0x8000000b
错误信息	PropertyIDReadOnly
说明	属性 ID 只读
相关函数	设置属性函数
实例	调用设置属性函数设置 FT_DevEmgMap 属性，但该属性只读，因此将报错

错误代码	0x8000000d
错误信息	InvalidAxCfgVel
说明	配置轴的最大运行速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property API 将轴的最大运行速度 (CFG_AxMaxVel) 设置为 20,000,001 时，由于该值上限为 20,000,000，因此将报错

错误代码	0x8000000e
错误信息	InvalidAxCfgAcc
说明	配置轴的最大加速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property API 将轴的最大加速度 (CFG_AxMaxAcc) 设置为 2,000,000,001 时，由于该值上限为 2,000,000,000，因此将报错

错误代码	0x8000000f
错误信息	InvalidAxCfgDec
说明	配置轴的最大减速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property API 将轴的最大减速度 (CFG_AxMaxDec) 设置为 2,000,000,001 时，由于该值上限为 2,000,000,000，因此将报错

错误代码	0x80000011
错误信息	InvalidAxParVelLow
说明	无效的轴的初速度，
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的初速度 (PAR_AxVelLow) 小于零或大于轴的最大运行速度 (CFG_AxMaxVel) 时，提示该错误

错误代码	0x80000012
错误信息	InvalidAxParVelHigh
说明	无效的轴的运行速度

相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的运行速度 (PAR_AxVelHigh) 小于等于零或大于轴的最大运行速度 (CFG_AxMaxVel) 时，提示该错误
错误代码	0x80000013
错误信息	InvalidAxParAcc
说明	无效的轴的加速度。
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的加速度小于等于零或大于轴的最大加速度 (CFG_AxMaxAcc) 时，提示该错误
错误代码	0x80000014
错误信息	InvalidAxParDec
说明	无效的轴的减速度。
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的减速度 PAR_AxDec 小于等于零或大于轴的最大减速度 (CFG_AxMaxDec) 时，提示该错误
错误代码	0x80000015
错误信息	InvalidAxParJerk
说明	无效的轴的加加速度
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 设置加加速度 PAR_AxJerk，该值为 0 或 1，当设置为其他值时，将提示错误
错误代码	0x80000016
错误信息	InvalidAxPulseInMode
说明	无效的脉冲输入模式
相关函数	Acm_SetU32Property
实例	调用 Acm_SetF64Property 设置 CFG_AxPulseInMode 属性值，当设置的值不在输入范围内时，将提示错误
错误代码	0x80000017
错误信息	InvalidAxPulseOutMode
说明	无效的脉冲输出模式
相关函数	Acm_SetU32Property
实例	调用 Acm_SetF64Property 设置 CFG_AxPulseOutMode 属性值，当设置的值不在输入范围内时，将提示错误
错误代码	0x8000002a
错误信息	InvalidAxState
说明	无效的轴状态

相关函数	Acm_AxGantryInAx、Acm_AxTangentInGp、Acm_AxGearInAx、Acm_AxCamInAx、Acm_AxMoveHome、Acm_AxHomeEx、Acm_AxSimStart、Acm_AxSimStartSuspendVel、Acm_AxSimStartSuspendRel、Acm_AxSimStartSuspendAbs、Acm_AxChangeVelExByRate、Acm_AxChangeVelEx、Acm_AxChangeVel、Acm_AxChangeVelByRate
实例	例如调用 Acm_AxHome 函数执行回原点，当轴的状态不为 Ready 时，将提示该错误
错误代码	0x8000002d
错误信息	InvalidAxDistance
说明	无效的轴距离
相关函数	Acm_AxMoveRel、Acm_AxMoveAbs 等
实例	调用 Acm_AxMoveRel 函数执行相对点位运动，Distance 的范围为：-2,147,483,647/PPU ~ 2,147,483,647/PPU。当设定的 Distance 不在该范围内时，将报错
错误代码	0x80000030
错误信息	InvalidAxCntInGp
说明	群组内的轴数不正确。
相关函数	Acm_GpMoveArcAbs_Angle、Acm_GpMoveArcRel_Angle 等
实例	例如执行三轴圆弧插补，当群组的轴数为 2 时，提示该错误
错误代码	0x80000031
错误信息	AxInGpNotFound
说明	该轴没有在群组内。
相关函数	Acm_GpRemAxis
实例	该错误发生在调用 Acm_GpRemAxis 函数移除群组中的轴时，当输入的轴号不在群组内时发生
错误代码	0x80000032
错误信息	AxisInOtherGp
说明	该轴在其他的群组内。
相关函数	Acm_GpAddAxis
实例	当调用 Acm_GpAddAxis 函数添加轴到群组，输入的轴号已经添加到其他群组时提示该错误。每个轴只能存在一个群组内
错误代码	0x80000033
错误信息	AxCannotIntoGp
说明	不能添加轴到群组内。
相关函数	Acm_GpAddAxis
实例	例如 PCI-1245V 支持群组轴数最多为 2 个，当调用 Acm_GpAddAxis 函数添加第三个轴到群组时，提示该错误
错误代码	0x80000039
错误信息	InvalidGpParVellow
说明	群组初速度无效

相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 设置群组初始速度 (PAR_GpVelLow) 时，若该值小于 0 或大于最大运行速度 (CFG_AxMaxVel)，则提示该错误
错误代码	0x8000003a
错误信息	InvalidGpParVelHigh
说明	群组运行速度无效
相关函数	Acm_GpMoveCircularRel、Acm_GpMoveCircularAbs、Acm_GpChangeVelByRate 等
实例	调用 Acm_GpChangeVel 函数改变群组运行速度，当设置的新速度小于 0 或大于 CFG_AxMaxVel 的值时，提示该错误
错误代码	0x8000003b
错误信息	InvalidGpParAcc
说明	群组加速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_GpAcc 的值，若该值大于 CFG_AxMaxAcc 或小于 0 时，则提示该错误
错误代码	0x8000003c
错误信息	InvalidGpParDec
说明	群组减速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_GpDec 的值，若该值大于 CFG_AxMaxDec 或小于 0 时，则提示该错误
错误代码	0x8000003d
错误信息	InvalidGpParJerk
说明	无效的群组曲线类型。
相关函数	Acm_GpMoveCircularRel_Angle、Acm_GpMoveCircularAbs_Angle、 Acm_GpMoveArcRel_Angle、Acm_GpMoveArcAbs_Angle、 Acm_GpMoveHelixAbs_Angle、Acm_GpMoveHelixRel_Angle、 Acm_GpMove3DArcAbs_V、Acm_GpMove3DArcRel_V
实例	调用函数 Acm_GpMoveCircularRel_Angle 执行角度圆弧插补，当设置的曲线类型 PAR_GpJerk 为 S 时，提示错误
错误代码	0x8000003e
错误信息	JerkNotSupport
说明	不支持 S 型曲线
相关函数	Acm_GpAddPath
实例	调用 Acm_GpAddPath 函数，设置 MoveMode 为 Blending Mode 或 FlyMode，这两种模式不支持 S 型曲线类型
错误代码	0x80000041
错误信息	InvalidGpState
说明	无效的群组状态

相关函数	Acm_GpClose、Acm_GpChangeVelByRate、Acm_GpChangeVel
实例	调用 Acm_GpClose 函数关闭群组，当群组状态 usState!=STA_GP_DISABLE&&usState!= STA_GP_READY&&usState!= STA_GP_ERROR_STOP 时，提示该错误
错误代码	0x80000042
错误信息	OpenFileFailed
说明	打开文件失败。
相关函数	Acm_DevLoadConfig、Acm_GpLoadPath、Acm_DevLoadCAMTableFile
实例	
错误代码	0x80000043
错误信息	InvalidPathCnt
说明	无效的 Path 数
相关函数	Acm_GpLoadPath、Acm_DevLoadCAMTableFile
实例	调用 Acm_GpLoadPath 函数下载 Path 文件，当 Path 数小于 2 或大于 600 时，提示错误，调用 Acm_DevLoadCAMTableFile 函数加载 CAM 表文件，当表内点数小于 2 大于 100,000 时，提示错误
错误代码	0x80000044
错误信息	InvalidPathHandle
说明	无效的 Path Handle。
相关函数	Acm_GpLoadPath、Acm_GpUnloadPath
实例	调用 Acm_GpLoadPath 和 Acm_GpUnloadPath 函数加载和卸载函数时，输入的 Path Handle 不存在或不正确时，提示该错误
错误代码	0x8000004A
错误信息	InvalidSlaveIP
说明	无效的从轴 IP。
相关函数	支持 SlaveIP 输入的所有函数
实例	调用 Acm_DevGetSlaveInfo 获取从轴信息时，若设备中不存在 SlaveIP，则提示该错误
错误代码	0x80000054
错误信息	FunctionNotSupport
说明	功能函数不支持。
相关函数	所有函数
实例	当板卡不支持该功能时，提示该错误
错误代码	0x80000055
错误信息	InvalidPhysicalAxis
说明	无效的物理轴。
相关函数	Acm_DevMDaqConfig

实例	调用 Acm_DevMDaqConfig 函数设定 MDAQ 相关配置，当指定的轴号大于板卡的轴数时，提示该错误。
----	---

错误代码	0x80000069
错误信息	InvalidPath
说明	对于 PCI-1240 板卡，提示该错误有两种情况 1、加载的 Path 为两轴直线、圆弧插补，但群组轴数大于 2 时，提示该错误 2、当加载的 Path 为多轴（大于等于 3 轴）直线插补，但群组的轴数小于 3 时，提示该错误 对于其他 PCI 运动控制卡，当用户添加到群组的轴数小于执行 Path 所需的轴数时，提示该错误
相关函数	Acm_GpLoadPath
实例	调用 Acm_GpLoadPath 函数加载 Path 文件，添加到群组中的轴数为 2，但 Path 文件中有 3 轴执行插补的 PATH，该情况下，将提示该错误

错误代码	0x80000073
错误信息	InvalidGpHandle
说明	群组 Handle 无效
相关函数	调用的函数参数有群组的 Handle
实例	调用 Acm_GpGetPathStatus 函数获取 Path 状态，当输入的群组 Handle 不正确时，提示该错误

错误代码	0x80000075
错误信息	InvalidCmpMethod
说明	无效的比较方法
相关函数	Acm_SetU32Property
实例	调用上述函数设置属性 CFG_AxCmpMethod，该属性可设置为 0（大于等于位置计数器）或 1（小于等于位置计数器），当设置为其他值时，提示该错误

错误代码	0x8000007b
错误信息	SpeedFordFunNotSpported
说明	当前模式不支持速度前瞻功能
相关函数	Acm_GpAddPath
实例	当调用函数 Acm_GpAddPath 添加 Path，且 MoveMode 为 BufferMode 或 BlendingMode 时，开启速度前瞻 CFG_GpSFEnable 功能将提示该错误

错误代码	0x80000081
错误信息	InvalidAxParHomeVellow
说明	无效的 Home 初速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Home 执行时的初速度（Par_AxHomeVellow），当设置的值小于等于 0 或大于 CFG_AxMaxVel 的值时，提示该错误

错误代码	0x80000082
错误信息	InvalidAxParHomeVelHigh
说明	无效的 Home 运行速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64Property 函数设置 Home 执行时的运行速度 (Par_AxHomeVelHigh)，当设置的值小于等于 0 或大于 CFG_AxMaxVel 的值时，提示该错误

错误代码	0x80000083
错误信息	InvalidAxParHomeAcc
说明	无效的 Home 加速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64Property 函数设置 Home 执行时的加速度 (Par_AxHomeAcc)，当设置的值小于等于 0 或大于 CFG_AxMaxAcc 的值时，提示该错误

错误代码	0x80000084
错误信息	InvalidAxParHomeDec
说明	无效的 Home 减速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64Property 函数设置 Home 执行时的减速度 (Par_AxHomeDec)，当设置的值小于等于 0 或大于 CFG_AxMaxDec 的值时，提示该错误

错误代码	0x80000085
错误信息	InvalidAxParHomeJerk
说明	无效的 Home 加速曲线类型
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64Property 函数设置 Home 执行时的速度曲线类型 (Par_AxHomeJerk)，速度类型为 0 或 1，当设置为其他值时，提示该错误

错误代码	0x80000086
错误信息	InvalidAxCfgJogVelLow
说明	无效的 Jog 初速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Jog 的初速度 (CFG_AxJogVelLow)，当设置的值小于等于 0 或大于 CFG_AxMaxVel 的值时，提示该错误

错误代码	0x80000087
错误信息	InvalidAxCfgJogVelHigh
说明	无效的 Jog 运行速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Jog 的运行速度 (CFG_AxJogVelHigh)，当设置的值小于等于 0 或大于 CFG_AxMaxVel 的值时，提示该错误

错误代码	0x80000088
错误信息	InvalidAxCfgJogAcc
说明	无效的 Jog 加速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Jog 的加速度 (CFG_AxJogVelAcc)，当设置的值小于等于 0 或大于 CFG_AxMaxAcc 的值时，提示该错误

错误代码	0x80000089
错误信息	InvalidAxCfgJogDec
说明	无效的 Jog 减速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Jog 的加速度 (CFG_AxJogVelDec)，当设置的值小于等于 0 或大于 CFG_AxMaxDec 的值时，提示该错误

错误代码	0x8000008A
错误信息	InvalidAxCfgJogJerk
说明	无效的 Jog 加速曲线类型
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 Jog 的速度曲线类型 (CFG_AxJogJerk)，该属性只能设置为 0，当设置为其他值时，提示错误

错误代码	0x8000008B
错误信息	InvalidAxCfgKillDec
说明	无效的 DI 减速度
相关函数	Acm_SetF64 Property
实例	调用 Acm_SetF64 Property 函数设置 DI Stop 的减速度，当设置的值小于等于零大于 FT_AxMaxDec 属性值时，提示该错误

错误代码	0x8000008C
错误信息	NotOpenAllAxes
说明	未打开所有轴，LoadCfg 时有轴未 Open
相关函数	Acm_DevLoadConfig
实例	调用 Acm_DevLoadConfig 函数下载配置文件时，板卡的所有轴都必须打开，否则提示该错误

错误代码	0x800000d0
错误信息	ReadFileFailed
说明	读取文件失败。
相关函数	Acm_DevLoadConfig
实例	调用函数载入配置文件或镜像表格文件时，若因操作系统错误导致文件错误或打开文件失败，则提示该错误

错误代码	0x800000d3
错误信息	GetAuthorityFailed
说明	PCI-1203 在开卡时扫描 EtherCAT 网络及改变状态机时 SDO 超时
相关函数	Acm_DevOpen, Acm_DevReOpen
实例	

错误代码	0x800000d6
错误信息	InvalidGpRadius
说明	设置的群组半径不正确
相关函数	圆弧插补
实例	当执行圆弧插补时，设置的半径不正确

错误代码	0x800000d9
错误信息	InvalidGpMovCmd
说明	无效的 Path 运行命令
相关函数	Acm_GpLoadPath、Acm_GpAddPath
实例	调用 Acm_GpLoadPath、Acm_GpAddPath 函数。设置的运行命令不在支持的范围内时，提示该错误。或者添加的运行命令板卡不支持

错误代码	0x800000dc
错误信息	InvalidPathMoveMode
说明	无效的 Path 运动模式。
相关函数	Acm_GpAddPath
实例	函数 Acm_GpAddPath 设置参数 MoveMode，该值为 0 或 1，当设置为其他值时，提示错误

错误代码	0x800000de
错误信息	InvalidCamTableID
说明	无效的凸轮表 ID
相关函数	Acm_DevDownloadCAMTable
实例	函数 Acm_DevDownloadCAMTable 的参数 CAM 表的 ID 要求为 0 或 1。当设置为其他值时，提示该错误

错误代码	0x800000df
错误信息	InvalidCamPointRange
说明	无效的凸轮执行点的范围。
相关函数	Acm_DevDownloadCAMTable
实例	调用 Acm_DevDownloadCAMTable 函数。当参数 pPointRangeArray[i] 大于 (pMasterArray[i] - pMasterArray[i-1])*0.5) 时提示该错误

错误代码	0x800000e0
错误信息	CamTableIsEmpty
说明	凸轮表为空。

相关函数	Acm_DevConfigCAMTable
实例	调用 Acm_DevConfigCAMTable 函数配置凸轮表。当凸轮表的没有值时，提示该错误
错误代码	0x800000e1
错误信息	InvalidPlaneVector
说明	设置的平面向量值无效
相关函数	Acm_AxTangentInGp
实例	调用 Acm_AxTangentInGp 函数建立轴和群组切线跟随关系，设置向量 StartVectorArray 的值，该向量必须为 3 维。当设置的 3 维向量都是 0 时，提示该错误
错误代码	0x800000e2
错误信息	MasAxIDSameSlvAxID
说明	主轴 ID 和从轴 ID 相同
相关函数	Acm_AxCamInAx、Acm_AxGantryInAx、Acm_AxGearInAx
实例	调用 Acm_AxCamInAx 函数建立主从轴关系，当主轴和从轴设置为同一轴时，提示该错误
错误代码	0x800000e3
错误信息	InvalidGpRefPlane
说明	无效的组的参考平面
相关函数	圆弧插补、螺旋插补、添加 Path
实 例	调用 Acm_GpMoveCircularRel_3P 函数执行 3 点圆弧插补，设置参考平面为 YZ 平面，当群组中的轴个数小于 3 时，提示该错误 当群组中的轴数小于 3 时，提示错误
错误代码	0x800000e4
错误信息	InvalidAxModuleRange
说明	设置的轴的单圈脉冲数无效
相关函数	Acm_SetU32Property
实例	调用 Acm_SetU32Property 函数，设置属性 CFG_AxModuleRange，当设置的值大于 8,000,000 或者该值不能被 4 整除时，提示错误
错误代码	0x800000e5
错误信息	DownloadFileFailed
说明	下载文件失败
相关函数	
实例	
错误代码	0x800000e6
错误信息	InvalidFileLength
说 明	无效的文件长度
相关函数	
实 例	

错误代码	0x800000e7
错误信息	InvalidCmpCnt
说 明	无效的比较个数
相关函数	Acm_AxSetCmpTable、Acm_AxSetCmpAuto
实 例	调用 Acm_AxSetCmpTable 设置比较表，函数中的参数 ArrayCount 设置为大于 100,000 或小于 0 时，提示该错误

错误代码	0x80004001
错误信息	DevEvtTimeOut
说 明	检测事件时间超时。
相关函数	Acm_CheckMotionEvent
实 例	调用 Acm_CheckMotionEvent 检测运动事件时，若规定时间内没有检测到任何事件，则提示该错误

错误代码	0x80004002
错误信息	DevNoEvt
说明	事件未发生。
相关函数	Acm_CheckMotionEvent
实例	调用 Acm_CheckMotionEvent 函数检测事件时，没有检测到事件时将提示该错误。

C.3 DSP 常见错误

错误代码	0x10000001
错误信息	Warning_AxWasInGp
说明	该轴已经添加到群组里
相关函数	Acm_GpAddAxis
实例	调用上述函数添加轴到群组中时，当该轴以在群组中时，提示该错误

错误代码	0x80005001
错误信息	ERR_SYS_TIME_OUT
说明	系统超时
相关函数	
实例	

错误代码	0x80005002
错误信息	Dsp_PropertyIDNotSupport
说明	PropertyID 不支持
相关函数	获取 / 设置属性的函数
实例	板卡不支持该属性 ID 时，提示该错误

错误代码	0x80005003
错误信息	Dsp_PropertyIDReadOnly
说明	PropertyID 只读
相关函数	设置属性的函数
实例	属性 FT_AxFunctionMap 只读，当调用设置属性的函数设置该属性时，提示错误

错误代码	0x80005004
错误信息	Dsp_InvalidParameter
说明	无效的输入参数
相关函数	Acm_DaqDoSetByte 等
实例	调用 Acm_DaqDoSetByte 函数设置 DO 字节数，当输入的参数 DOPort 大于最大 DOPort (FT_DaqDoMaxChan/8) 时，提示该错误

错误代码	0x80005005
错误信息	Dsp_DataOutBufExceeded
说明	数据缓存不足
相关函数	
实例	

错误代码	0x80005006
错误信息	Dsp_FunctionNotSupport
说明	功能函数不支持
相关函数	Acm_GpMove3DArcAbs、Acm_GpMove3DArcRel 等
实例	运动控制卡不支持该函数时，将提示该错误

错误代码	0x80005007
错误信息	Dsp_InvalidConfigFile
说明	无效的配置文件
相关函数	
实例	

错误代码	0x80005008
错误信息	Dsp_InvalidIntervalData
说明	无效的间隔数据
相关函数	
实例	

错误代码	0x80005009
错误信息	Dsp_InvalidTableSize
说明	无效的表格大小
相关函数	
实例	

错误代码	0x8000500a
错误信息	Dsp_InvalidTableID
说明	无效的凸轮表 ID 或者比较列表 ID
相关函数	Acm_DevDownloadCAMTable
实例	函数 Acm_DevDownloadCAMTable 的参数 CAM 表的 ID 要求为 0 或 1. 当设置为其他值时，提示该错误

错误代码	0x8000500b
错误信息	Dsp_DataIndexExceedBufSize
说明	超过比较列表的最大点数（100000）或单次读取点数超过最大点数（32）
相关函数	Acm_AxSetCmpTable、Acm_AxSetCmpAuto、Acm_AxReadLatchBuffer
实例	调用 Acm_AxSetCmpTable 函数设置比较表，该表支持的最多比较数据为 100,000 个，当超出时，提示该错误

错误代码	0x8000500c
错误信息	Dsp_InvalidCompareInterval
说明	无效的比较间隔
相关函数	Acm_AxSetCmpAuto
实例	

错误代码	0x8000500d
错误信息	Dsp_InvalidCompareRange
说明	比较列表的起点数据和终点数据的距离小于比较间隔
相关函数	Acm_AxSetCmpAuto
实例	当调用上述函数设置线性比较数据，当起点数据和终点数据的距离小于比较间隔时，提示该错误

错误代码	0x8000500e
错误信息	Dsp_PropertyIDWriteOnly
说明	该属性只读
相关函数	获取属性函数
实例	当调用函数获取的属性信息为只写属性时，提示错误

错误代码	0x8000500f
错误信息	Dsp_NcError
说明	NC 错误
相关函数	
实例	

错误代码	0x80005010
错误信息	Dsp_CamTableIsInUse
说明	当前 ID 的凸轮表正在被使用
相关函数	Acm_AxCamInAx
实例	

错误代码	0x80005011
错误信息	Dsp_EraseBlockFailed
说明	擦除 block 失败
相关函数	
实例	

错误代码	0x80005012
错误信息	Dsp_ProgramFlashFailed
说明	存储 (Flash) 程序失败
相关函数	
实例	

错误代码	0x80005101
错误信息	Dsp_InvalidAxCfgVel
说明	配置轴的最大运行速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 将最大运行速度 (CFG_AxMaxVel) 设置为 20,000,001 时，由于该值上限为 20,000,000，因此将报错

错误代码	0x80005102
错误信息	Dsp_InvalidAxCfgAcc
说明	配置轴的最大加速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 将最大加速度 (CFG_AxMaxAcc) 设置为 2,000,000,001 时，由于该值上限为 2,000,000,000，因此将报错

错误代码	0x80005103
错误信息	Dsp_InvalidAxCfgDec
说明	配置轴的最大减速度失败
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 将最大减速度 (CFG_AxMaxDec) 设置为 2,000,000,001 时，由于该值上限为 2,000,000,000，因此将报错

错误代码	0x80005104
错误信息	Dsp_InvalidAxCfgJerk
说明	配置轴的加加速度失败
相关函数	Acm_SetF64Property
实例	

错误代码	0x80005105
错误信息	Dsp_InvalidAxParVelLow
说明	无效的轴的初速度，
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的初速度 (PAR_AxVelLow) 小于零或大于轴的最大运行速度 (CFG_AxMaxVel) 时，提示该错误

错误代码	0x80005106
错误信息	Dsp_InvalidAxParVelHigh
说明	无效的轴的运行速度
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的运行速度 (PAR_AxVelHigh) 小于等于零或大于轴的最大运行速度 (CFG_AxMaxVel) 时，提示该错误

错误代码	0x80005107
错误信息	Dsp_InvalidAxParAcc
说明	无效的轴的加速度
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的加速度 PAR_AxAcc 小于等于零或大于轴的最大加速度 (CFG_AxMaxAcc) 时，提示该错误

错误代码	0x80005108
错误信息	Dsp_InvalidAxParDec
说明	无效的轴的减速度
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置轴的加速度 PAR_AxAcc 小于等于零或大于轴的最大减速度 (CFG_AxMaxAcc) 时，提示该错误

错误代码	0x80005109
错误信息	Dsp_InvalidAxParJerk
说明	无效的轴的加加速度
相关函数	Acm_SetF64Property
实例	调用上述函数设置加加速度 PAR_AxJerk，该值为 0 或 1，当设置为其他值时，将提示错误

错误代码	0x8000510a
错误信息	Dsp_InvalidAxPptValue
说明	无效的属性值
相关函数	
实例	

错误代码	0x8000510b
错误信息	Dsp_InvalidAxState
说明	无效的轴状态
相关函数	Acm_AxMoveVel, Acm_AxStopDec 等
实例	当轴的状态不是 Ready 时，若执行连续运动，将提示该错误

错误代码	0x8000510c
错误信息	Dsp_InvalidAxSvOnOff
说明	无效的伺服开 / 关操作
相关函数	
实例	

错误代码	0x8000510d
错误信息	Dsp_InvalidAxDistance
说明	无效的轴距离
相关函数	Acm_AxMoveImpose
实例	

错误代码	0x8000510e
错误信息	Dsp_InvalidAxPosition
说明	无效位置
相关函数	
实例	

错误代码	0x8000510f
错误信息	Dsp_InvalidAxHomeMode
说明	无效的 home 模式
相关函数	Acm_AxHome
实例	轴的回 Home 共 16 种模式，当输入的值不在该范围内时，提示错误

错误代码	0x80005110
错误信息	Dsp_InvalidPhysicalAxis
说明	无效的轴 ID
相关函数	Acm_DevMDaqConfig 等
实例	调用 Acm_DevMDaqConfig 函数设定 MDaq 相关配置，当指定的轴号大于板卡的轴数时，提示该错误。

错误代码	0x80005111
错误信息	Dsp_HLmtPExceeded
说明	超出正向硬极限
相关函数	
实例	

错误代码	0x80005112
错误信息	Dsp_HLmtNExceeded
说明	超出负向硬极限
相关函数	
实例	

错误代码	0x80005113
错误信息	Dsp_SLmtPExceeded
说明	超出正向软极限
相关函数	
实例	

错误代码	0x80005114
错误信息	Dsp_SLmtNExceeded
说明	超出负向软极限
相关函数	
实例	

错误代码	0x80005115
错误信息	Dsp_AlarmHappened
说明	Alarm 信号发生
相关函数	
实例	

错误代码	0x80005116
错误信息	Dsp_EmgHappened
说明	Emg 信号发生
相关函数	
实例	

错误代码	0x80005117
错误信息	Dsp_CmdValidOnlyInConstSec
说明	叠加运动只能在匀速阶段执行
相关函数	Acm_AxMoveImpose
实例	当在非匀速段添加叠加运动时，提示该错误

错误代码	0x80005118
错误信息	Dsp_InvalidAxCmd
说明	无效的轴命令
相关函数	
实例	

错误代码	0x80005119
错误信息	Dsp_InvalidAxHomeDirMode
说明	无效的回 home 方向
相关函数	Acm_AxHome
实例	

错误代码	0x8000511a
错误信息	Dsp_AxisMustBeModuloAxis
说明	切向跟随轴模式不能为 0
相关函数	Acm_AxTangentInGp
实例	

错误代码	0x8000511b
错误信息	Dsp_AxIdCantSameAsMasId
说明	从轴 ID 不能与主轴相同
相关函数	Acm_AxCamInAx, Acm_AxGearInAx、Acm_AxGantryInAx
实例	调用 Acm_AxCamInAx 建立主轴和从轴之间的关系时，主轴与从轴必须为不同的轴，否则将报错

错误代码	0x8000511c
错误信息	Dsp_CantResetPosiOfMasAxis
说明	同步状态下不能设置理论位置或实际位置
相关函数	Acm_AxSetCmdPosition, Acm_AxSetActualPosition
实例	在同步状态下，不能调用上述函数重置位置

错误代码	0x8000511d
错误信息	Dsp_InvalidAxExtDrvOperation
说明	无效的轴外部驱动操作
相关函数	Acm_AxSetExtDrive
实例	

错误代码	0x8000511e
错误信息	Dsp_AxAccExceededMaxAcc
说明	轴加速度超出最大加速度
相关函数	
实 例	

错误代码	0x8000511f
错误信息	Dsp_AxVelExceededMaxVel
说明	轴运行速度超出最大速度
相关函数	Acm_AxChangeVel
实例	调用上述函数设置轴的新速度，当该值超过最大运行速度时，提示错误

错误代码	0x80005201
错误信息	Dsp_InvalidAxCntInGp
说明	执行运动的轴数量与群组中的轴数量不一致
相关函数	Acm_GpMoveLinearRel, Acm_GpMoveCircularRel 等
实例	例如执行三轴执行插补，群组中的轴数为 2，则提示错误

错误代码	0x80005202
错误信息	Dsp_AxInGpNotFound
说明	未在群组中找到相应轴
相关函数	Acm_GpRemAxis
实例	调用 Acm_GpRemAxis 移除轴 1，但是轴 1 未在该群组内，将报错

错误代码	0x80005203
错误信息	Dsp_AxisInOtherGp
说明	需要添加的轴已经在其他群组中
相关函数	Acm_GpAddAxis
实例	调用 Acm_GpAddAxis 添加轴到群组中，当该轴已在其他群组中时，提示该错误。每个轴仅能存在于一个群组中。

错误代码	0x80005204
错误信息	Dsp_AxCannotIntoGp
说明	无法将轴加入群组。添加的轴无效或者群组中的轴数量已经达到上限
相关函数	Acm_GpAddAxis
实例	调用 Acm_GpAddAxis 添加第 9 轴到群组中时，由于超过一个群组支持的最大轴数（8 轴），因此将报错

错误代码	0x80005205
错误信息	Dsp_GpInDevNotFound
说明	群组未找到
相关函数	
实例	

错误代码	0x80005206
错误信息	Dsp_InvalidGpCfgVel
说明	无效的群组速度
相关函数	
实例	

错误代码	0x80005207
错误信息	Dsp_InvalidGpCfgAcc
说明	无效的群组加速度
相关函数	
实例	

错误代码	0x80005208
错误信息	Dsp_InvalidGpCfgDec
说明	无效的群组减速度
相关函数	
实例	

错误代码	0x80005209
错误信息	Dsp_InvalidGpCfgJerk
说明	无效的群组加加速度
相关函数	
实例	

错误代码	0x8000520a
错误信息	Dsp_InvalidGpParVelLow
说明	群组起始速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_GpVelLow 的值大于 CFG_AxMaxVel 或小于 0 时，提示该错误

错误代码	0x8000520b
错误信息	Dsp_InvalidGpParVelHigh
说明	群组运行速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_GpVelHigh 的值大于 CFG_AxMaxVel 或小于 0 时，提示该错误

错误代码	0x8000520c
错误信息	Dsp_InvalidGpParAcc
说明	群组加速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_Gp Acc 的值大于 CFG_AxMaxAcc 或小于 0 时，提示该错误

错误代码	0x8000520d
错误信息	Dsp_InvalidGpParDec
说明	群组减速度无效
相关函数	Acm_SetF64Property
实例	调用 Acm_SetF64Property 函数设置 PAR_Gp_Dec 的值大于 CFG_AxMaxDec 或小于 0 时，提示该错误

错误代码	0x8000520e
错误信息	Dsp_InvalidGpParJerk
说明	群组速度曲线类型无效
相关函数	Acm_GpMoveCircularRel_Angle、Acm_GpMoveCircularAbs_Angle、 Acm_GpMoveArcRel_Angle、Acm_GpMoveArcAbs_Angle、 Acm_GpMoveHelixAbs_Angle、Acm_GpMoveHelixRel_Angle、 Acm_GpMove3DArcAbs_V、Acm_GpMove3DArcRel_V
实例	调用函数 Acm_GpMoveCircularRel_Angle 执行角度圆弧插补，当设置的曲线类型 PAR_GpJerk 为 S 时，提示错误

错误代码	0x8000520f
错误信息	Dsp_JerkNotSupport
说明	不支持加加速度
相关函数	
实例	

错误代码	0x80005210
错误信息	Dsp_ThreeAxNotSupport
说明	不支持三轴
相关函数	
实例	

错误代码	0x80005211
错误信息	Dsp_DevIpoNotFinished
说明	未完成设备插补
相关函数	
实例	

错误代码	0x80005212
错误信息	Dsp_InvalidGpState
说明	无效的群组状态
相关函数	Acm_GpMoveLinearRel, Acm_GpMoveCircularRel 等
实例	调用上面函数执行直线插补运动，当群组的状态不是 Ready 时，提示该错误

错误代码	0x80005213
错误信息	Dsp_OpenFileFailed
说明	打开文件失败
相关函数	
实例	

错误代码	0x80005214
错误信息	Dsp_InvalidPathCnt
说明	无效的 path 数量
相关函数	Acm_GpMovePath
实例	调用 Acm_GpLoadPath 载入路径文件时，若路径数小于 2，则提示该错误

错误代码	0x80005215
错误信息	Dsp_InvalidPathHandle
说明	无效的路径句柄
相关函数	
实例	

错误代码	0x80005216
错误信息	Dsp_InvalidPath
说明	无效路径
相关函数	
实例	

错误代码	0x80005217
错误信息	Dsp_GpSlavePositionOverMaster
说明	从轴位置超过主轴位置
相关函数	
实例	

错误代码	0x80005219
错误信息	Dsp_GpPathBufferOverflow
说明	群组中的 Path 缓存区已满（7,000）
相关函数	Acm_GpAddPath
实例	调用 Acm_GpAddPath 函数添加的 Path 数超过 7,000 段时，提示该错误

错误代码	0x8000521a
错误信息	Dsp_InvalidPathFunctionID
说明	无效的路径函数 ID
相关函数	
实例	

错误代码	0x8000521b
错误信息	Dsp_SysBufAllocateFailed
说明	分配系统缓存失败
相关函数	
实例	

错误代码	0x8000521c
错误信息	Dsp_InvalidGpCenterPosition
说明	无效的群组圆心位置
相关函数	Acm_GpMove3DArcRel、Acm_GpMove3DArcAbs、Acm_GpMove3DArcAbs_V、 Acm_GpMove3DArcAbs_V
实例	调用上述函数执行 3D 圆弧插补，输入的中心点不正确时，提示该错误

错误代码	0x8000521d
错误信息	Dsp_InvalidGpEndPosition
说明	无效的群组终点位置
相关函数	Acm_GpMove3DArcRel、Acm_GpMove3DArcAbs
实例	调用上述函数执行 3D 圆弧插补，输入的终点不正确时，提示该错误

错误代码	0x8000521e
错误信息	Dsp_InvalidGpCmd
说明	无效的群组命令
相关函数	
实例	

错误代码	0x8000521f
错误信息	Dsp_AxHasBeenInInGp
说明	轴已经在群组中
相关函数	Acm_GpAddAxis、Acm_AxGearInAx 等
实例	调用函数 Acm_AxGearInAx 建立齿轮关系，当该轴已经和其他轴建立齿轮关系时，提示该错误

错误代码	0x80005220
错误信息	Dsp_InvalidPathRange
说明	无效的路径范围
相关函数	Acm_GpMoveSelPath
实例	当前 path buffer 中剩余有 10 段 Path（调用 Acm_GpGetPathStatus 函数查看剩余的 path 数量）。若 StartIndex 或 EndIndex 的设置值大于 10，或 StartIndex 的值大于 EndIndex，则提示错误

C.4 EtherCAT 常见错误

错误代码	0x8300002B
错误信息	EC_SlaveIDConflicted
说明	从轴 ID 冲突
相关函数	Acm_DevOpen, Acm_DevReOpen
实例	两个从轴的 ID 相同并连接至同一环中，因此将报错

错误代码	0x8300003D
错误信息	EC_AxCntExceeded
说明	所连接的轴的总数超出轴数上限
相关函数	Acm_DevOpen, Acm_DevReOpen
实例	连接至 PCI-1203 的电机驱动数超出上限，因此将报错

错误代码	0x8300003E
错误信息	EC_FWUpgraded
说明	更新固件后，请关闭电脑并再次开启电脑以完成固件更新。
相关函数	Acm_DevOpen, Acm_DevReOpen
实例	

错误代码	0x83010001
错误信息	ECAT_MasterEcatError
说明	EtherCAT 主站报警
相关函数	
实例	

错误代码	0x83010002
错误信息	ECAT_SlaveDisconnect_R0
说明	从站断开与 Ring 0 的连接
相关函数	
实例	

错误代码	0x83010003
错误信息	ECAT_SlaveDisconnect_R1
说明	从站断开与 Ring 1 的连接
相关函数	
实例	

错误代码	0x83030004
错误信息	ECAT_MOTION_SLAVE_NOT_SUPPORT
说明	Ring 0 中连接不支持的 EtherCAT 从站
相关函数	Acm_DevOpen, Acm_DevReOpen

实例	
错误代码	0x83040004
错误信息	ECAT_IO_SLAVE_NOT_SUPPORT
说明	Ring 1 中连接不支持的 EtherCAT 从站。IO 环不支持连接电机驱动器。
相关函数	Acm_DevOpen, Acm_DevReOpen
实例	
错误代码	0x83050001
错误信息	ECAT_RetryError
说明	重置错误或重置位置计数器时，从站未响应（超时）
相关函数	Acm_AxResetError, Acm_AxSetCmdPosition
实例	
错误代码	0x83050002
错误信息	ECAT_ResetCounterError
说明	重置位置计数器时，从站发生错误
相关函数	Acm_AxSetCmdPosition
实例	
错误代码	0x83050003
错误信息	ECAT_CmdErrorProtection
说明	指令错误保护
相关函数	
实例	切换操作模式时（例如从原点模式切换至 P2P 模式），若理论位置与实际位置之间的距离差值大于指令错误保护值，则提示该错误
错误代码	0x83050004
错误信息	ECAT_SlaveEcatALM
说明	EtherCAT 从站报警
相关函数	
实例	
错误代码	0x83050005
错误信息	ECAT_AxFollowError
说明	跟随错误保护
相关函数	
实例	若电机的理论位置与实际位置之间的距离差值大于驱动的设置值，则提示该错误
错误代码	0x83050006
错误信息	ECAT_AxHomeError
说明	执行原点运动时，从站发生错误

相关函数	Acm_AxMoveHome, Acm_AxHomeEx
实例	
错误代码	0x8310xxxx
错误信息	ECAT_DriveError
说明	驱动器错误 (0x8310xxxx)。请参考驱动器手册或驱动前置 LED 面板，了解驱动器定义的错误码 xxxx。
相关函数	
实例	

C.5 不常见错误

错误代码	错误信息	说明
0x80000001	DevRegDataLost	
0x80000002	LoadDllFailed	
0x80000003	GetProcAddressFailed	
0x80000007	OpenEventFailed	
0x80000008	EventTimeOut	
0x8000000C	ConnectWinIrqFailed	
0x80000018	InvalidAxAlarmEn	
0x80000019	InvalidAxAlarmLogic	
0x8000001A	InvalidAxInPEn	
0x8000001B	InvalidAxInPLogic	
0x8000001C	InvalidAxHLmtEn	
0x8000001D	InvalidAxHLmtLogic	
0x8000001E	InvalidAxHLmtReact	
0x8000001F	InvalidAxSLmtPEn	
0x80000020	InvalidAxSLmtPReact	
0x80000021	InvalidAxSLmtPValue	
0x80000022	InvalidAxSLmtMEn	
0x80000023	InvalidAxSLmtMReact	
0x80000024	InvalidAxSLmtMValue	
0x80000025	InvalidAxOrgLogic	
0x80000026	InvalidAxOrgEnable	
0x80000027	InvalidAxEzLogic	
0x80000028	InvalidAxEzEnable	
0x80000029	InvalidAxEzCount	
0x8000002B	InvalidAxInEnable	
0x8000002C	InvalidAxSvOnOff	
0x8000002E	InvalidAxPosition	
0x8000002F	InvalidAxHomeModeKw	
0x80000034	GpInDevNotFound	
0x80000035	InvalidGpCfgVel	
0x80000036	InvalidGpCfgAcc	
0x80000037	InvalidGpCfgDec	
0x80000038	InvalidGpCfgJerk	
0x80000040	DevIpoNotFinished	
0x80000046	IoctlError	
0x80000047	AmnetRingWhile usingd	
0x80000048	DeviceNotOpened	
0x80000049	InvalidRing	
0x8000004A	InvalidSlaveIP	
0x8000004B	InvalidParameter	
0x8000004C	InvalidGpCenterPosition	
0x8000004D	InvalidGpEndPosition	
0x8000004E	InvalidAddress	
0x8000004F	DeviceDisconnect	

0x80000050	DataOutBufExceeded
0x80000051	SlaveDeviceNotMatch
0x80000052	SlaveDeviceError
0x80000053	SlaveDeviceUnknow
0x80000056	InvalidVelocity
0x80000057	InvalidAxPulseInLogic
0x80000058	InvalidAxPulseInSource
0x80000059	InvalidAxErcLogic
0x8000005A	InvalidAxErcOnTime
0x8000005B	InvalidAxErcOffTime
0x8000005C	InvalidAxErcEnableMode
0x8000005D	InvalidAxSdEnable
0x8000005E	InvalidAxSdLogic
0x8000005F	InvalidAxSdReact
0x80000060	InvalidAxSdLatch
0x80000061	InvalidAxHomeResetEnable
0x80000062	InvalidAxBacklashEnable
0x80000063	InvalidAxBacklashPulses
0x80000064	InvalidAxVibrationEnable
0x80000065	InvalidAxVibrationRevTime
0x80000066	InvalidAxVibrationFwdTime
0x80000067	InvalidAxAlarmReact
0x80000068	InvalidAxLatchLogic
0x80000069	InvalidFwMemoryMode
0x8000006A	InvalidConfigFile
0x8000006B	InvalidAxEnEvtArraySize
0x8000006C	InvalidAxEnEvtArray
0x8000006D	InvalidGpEnEvtArraySize
0x8000006E	InvalidGpEnEvtArray
0x8000006F	InvalidIntervalData
0x80000070	InvalidEndPosition
0x80000071	InvalidAxisSelect
0x80000072	InvalidTableSize
0x80000074	InvalidCmpSource
0x80000076	InvalidCmpPulseMode
0x80000077	InvalidCmpPulseLogic
0x80000078	InvalidCmpPulseWidth
0x80000079	InvalidPathFunctionID
0x8000007A	SysBufAllocateFailed
0x80000096	SlaveIOUpdateError
0x80000097	NoSlaveDevFound
0x80000098	MasterDevNotOpen
0x80000099	MasterRingNotOpen
0x800000C8	InvalidDIPort
0x800000C9	InvalidDOPort
0x800000CA	InvalidDOValue
0x800000CC	CreateThreadFailed
0x800000CD	InvalidHomeModeEx

0x800000CE	InvalidDirMode
0x800000CF	AxHomeMotionFailed
0x800000D1	PathBufIsFull
0x800000D2	PathBufIsEmpty
0x800000D3	GetAuthorityFailed
0x800000D4	GpIDAllocatedFailed
0x800000D5	FirmWareDown
0x800000D7	InvalidAxCmd
0x800000D8	InvalidaxExtDrv
0x800000DA	SpeedCurveNotSupported
0x800000DB	InvalidCounterNo
0x800000DD	PathSelStartCantRunInSpeed- ForewareMode
0x80002000	HLmtPExceeded
0x80002001	HLmtNExceeded
0x80002002	SLmtPExceeded
0x80002003	SLmtNExceeded
0x80002004	AlarmHappened
0x80002005	EmgHappened
0x80002006	TimeLmtExceeded
0x80002007	DistLmtExceeded
0x80002008	InvalidPositionOverride
0x80002009	OperationErrorHappened
0x8000200A	SimultaneousStopHappened
0x8000200B	OverflowInPAPB
0x8000200C	OverflowInIPO
0x8000200D	STPHappened
0x8000200E	SDHappened
0x8000200F	AxsiNoCmpDataLeft

错误代码	错误信息	说明
0x83000000	EC_GetNICNumberFailed	
0x83000001	EC_GetNICInfoFailed	
0x83000002	EC_OpenMasterDevFailed	
0x83000003	EC_GetSlaveFailed	
0x83000004	EC_StartOpModeFailed	
0x83000005	EC_CloseDeviceFailed	
0x83000006	EC_MemAllocateFailed	
0x83000007	EC_InvalidNicIndex	
0x83000008	EC_OpenDevFailed	
0x83000009	EC_ReadFileFailed	
0x8300000A	EC_GetNicInfoFailed	
0x8300000B	EC_GetSD0Failed	
0x8300000C	EC_InvalidParameter	
0x8300000D	EC_GetPD00ffsetFailed	
0x8300000E	EC_InitialMappingInfoFalied	
0x8300000F	EC_InitResourceFailed	
0x83000010	EC_SetSD0Failed	

0x83000011	EC_InvalidPortType
0x83000012	EC_SetCycleTimeFailed
0x83000013	EC_InvalidAoRange
0x83000014	EC_InvalidAiRange
0x83000015	EC_GetSlaveInfoFailed
0x83000016	EC_GetNetWorkStateFailed
0x83000017	EC_RegisterEventFailed
0x83000018	EC_InvalidIntegrationTime
0x83000019	EC_InvalidAIEnable
0x8300001A	EC_InvalidDIFilter
0x8300001B	EC_SetSlaveStateFailed
0x8300001C	EC_ZeroCalibrationFailed
0x8300001D	EC_InvalidMasterHandle
0x8300001E	EC_InvalidENIFile
0x8300001F	EC_InvalidCaliType
0x83000020	EC_SetCaliValueFailed
0x83000021	EC_AOCalibrationFailed
0x83000022	EC_InvalidIOMapping
0x83000023	EC_PortIndexGreaterThanPortNum
0x83000024	EC_ChannelIDGreaterThanChannelNum
0x83000025	EC_InputIndexGreaterThanInputNum
0x83000026	EC_OutputIndexGreaterThanOutputNum
0x83000027	EC_SetEnableFailed
0x83000028	EC_SetAIRangeFailed
0x83000029	EC_SetIntegrationTimeFailed
0x8300002A	EC_PropertyNotSupported
0x8300002B	EC_SlaveIDConflicted
0x8300002C	EC_SpanCalibrationFailed
0x8300002D	EC_InvalidAiValue
0x8300002E	EC_InvalidAoValue
0x8300002F	EC_GetModuleFailed
0x83000030	EC_InvalidCntEnable
0x83000031	EC_InvalidCntPulseInMode
0x83000032	EC_InvalidCntInitValue
0x83000033	EC_InvalidCntMaxValue
0x83000034	EC_InvalidCntOverflowMode
0x83000035	EC_InvalidCntLatchEnable
0x83000036	EC_InvalidCntLatchEdge
0x83000037	EC_InvalidCntCmpEnable
0x83000038	EC_InvalidCntCmpMethod
0x83000039	EC_InvalidCntCmpPulseEnable
0x8300003A	EC_InvalidCntCmpPulseMode
0x8300003B	EC_InvalidCntCmpPulseLogic
0x8300003C	EC_InvalidCntCmpPulseWidth

www.advantech.com.cn

使用前请检查核实产品的规格。本手册仅作为参考。

产品规格如有变更，恕不另行通知。

未经研华公司书面许可，本手册中的所有内容不得通过任何途径以任何形式复制、翻印、翻译或者传输。

所有的产品品牌或产品型号均为公司之注册商标。

© 研华公司 2017