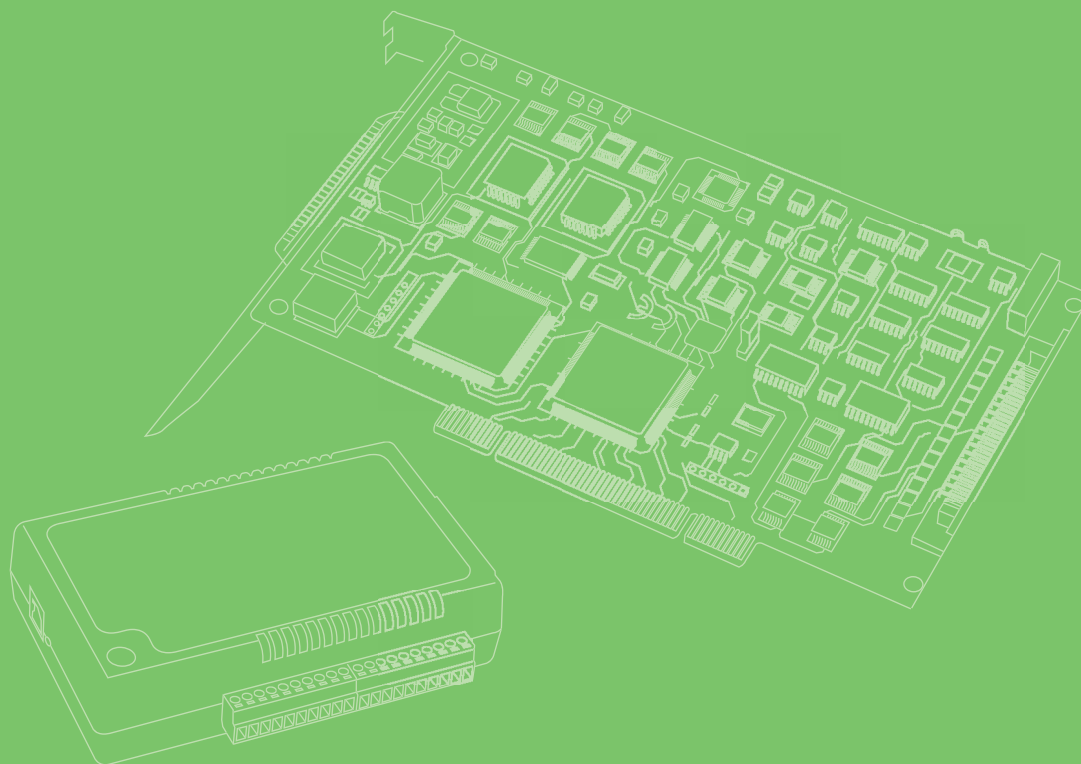


用户手册



Softmotion 软件手册

通用运动架构 (Common Motion)

ADVANTECH

Enabling an Intelligent Planet

版权声明

随附本产品发行的文件为研华公司 2015 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

PC-LabCard 是研华公司的商标。

IBM 和 PC 是 International Business Machines Corporation 的商标。

MS-DOS、Windows®, Microsoft® Visual C++ and Visual BASIC 为 Microsoft Corp. 的注册商标。

Intel® 和 Pentium® 为 Intel Corporation 的商标。

Delphi 和 C++Builder 为 Inprise Corporation 的商标。

所有其它产品名或商标均为各自所属方的财产。

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。

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。

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圆心联系。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

包装清单

安装系统之前，用户需确认包装中含有本设备以及下面所列各项，并确认设备完好。若有任何不符，请立即与经销商联系。

- 研华运动控制板卡
- 附带的 CD-ROM 光盘（包括 DLL 驱动）
- 快速入门手册

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。

目录

第 1 章 概述 1

1.1	本章简介.....	2
1.2	PCI-1285/1285E 运动控制卡	2
	表 1.1: PCI-1285/85E 系列运动控制卡功能表	2
1.3	PCI-1265 运动控制卡	3
	表 1.2: PCI-1265 运动控制卡功能表	3
1.4	PCI-1245/45E/45V 运动控制卡	5
	表 1.3: PCI-1245 系列板卡功能表	5
1.5	PCI-1245L 运动控制卡	6
	表 1.4: PCI-1245L 系列板卡功能表	6
1.6	MIC-3245/3285 运动控制卡	7

第 2 章 Common Motion Utility 9

2.1	Common Motion Utility 简介	10
2.2	Common Motion Utility 安装	10
2.3	Common Motion Utility 用户界面	11
	2.3.1 主界面.....	11
2.4	主界面 -- 菜单栏.....	11
	2.4.1 文件 (File) 菜单项.....	12
	2.4.2 语言 (Language) 菜单项.....	12
	2.4.3 视图 (View) 菜单项.....	12
	2.4.4 帮助 (Help) 菜单项.....	12
2.5	主界面 -- 工具栏.....	12
	2.5.1 安装 (Install) 工具项.....	12
	2.5.2 更新 (Refresh) 工具项.....	14
	2.5.3 保存 (Save) 工具项.....	14
	2.5.4 导入 (Load) 工具项.....	14
	2.5.5 下载 (Download) 工具项.....	14
	2.5.6 数据采集 (MotionDAQ) 工具栏.....	15
	2.5.7 隐藏树 (Hide Tree) 工具栏.....	18
2.6	Common Motion Utility 单轴运动控制	18
	图 2.1: 单轴运动控制界面.....	18
	2.6.1 轴操作 (Operate Axis)	18
	2.6.2 轴运动参数设置 (Motion Param Set)	19
	2.6.3 配置 (Configuration)	19
	2.6.4 运动测试.....	25
	2.6.5 运动位置 (Position) 检测.....	25
	2.6.6 轴的当前状态 (Current Axis Status)	25
	2.6.7 DI/O 状态	26
	2.6.8 Last Error Status.....	26
2.7	Common Motion Utility 多轴运动控制	27
	2.7.1 操作轴.....	27
	2.7.2 运动参数设置 (Motion Params Set)	27
	2.7.3 运动操作 (Motion Operation)	28
	2.7.4 运动曲线 (Path Plot)	31
	2.7.5 速度曲线 (Speed Chart)	32
	2.7.6 位置 (Position)	32
	2.7.7 组的状态 (Group State)	32
	2.7.8 最新错误状态 (Last Error)	33
2.8	Common Motion Utility 同步运动控制	33
	2.8.1 从轴运动操作 (Slave Axis Operation)	33
	2.8.2 主轴运动操作 (Master Axis Operation)	39
	2.8.3 最新错误状态 (Last Error Status)	40

2.9	数字量输入 (Digital Input)	40
2.9.1	数字量输入 (Digital Input).....	40
2.9.2	数字量输出 (Digital Output).....	41
2.10	Common Motion Utility GCode 操作	41
2.10.1	GCode 文本编辑器 (GCode File Editor)	41
2.10.2	GCode GCode 状态显示栏.....	44
2.10.3	GCode 命令窗口.....	45

第 3 章 关于研华运动控制卡接口函数库及其在编程工具中使用 47

3.1	本章简介	48
3.2	关于研华运动控制卡接口函数库	48
	图 3.1: 不同设备对接口函数的调用	48
3.2.1	运动控制卡接口函数库所需头文件	49
	图 3.2: 动态链接库所需头文件	49
3.2.2	接口函数库 --- 功能函数	49
3.2.3	接口函数库 --- 功能属性	50
	表 3.1: 属性的命名规则	50
	表 3.2: 设置 / 获取属性函数列表	51
	图 3.3: 属性数据类型与函数的关系	51
3.2.4	接口函数库 --- 设备类型	52
	表 3.3: 设备类型定义	52
3.2.5	接口函数库 --- 错误代码	52
3.2.6	接口函数库 --- 数据类型再定义	52
	表 3.4: 数据类型再定义	52
3.2.7	板卡的初始操作	53
	图 3.4: 板卡初始化流程图	54
3.3	运动控制卡接口函数库在编程工具中的使用	55
	图 3.5: 属性数据类型与函数的关系	55
3.3.1	接口函数库在 Visual C++ 6.0 中的使用	56
3.3.2	接口函数库在 Visual Basic 6.0 中的使用	56
3.3.3	接口函数库在 Visual Studio 2005 中的使用	57
3.3.4	接口函数库在 Borland C++ Builder 中的使用	58
3.4	关于在 Win7 下使用 Common Motion API 的注意事项	59
3.4.1	关于提升应用程序的权限	59
3.4.2	Win7 下运行范例.....	59
3.5	多块板卡编程说明	60
3.5.1	多块板卡的独立性	60
3.5.2	如何操作多快板卡	60

第 4 章 通用运动 API 架构 63

4.1	本章简介	64
4.2	通用运动架构简介	64
	图 4.1: 运动控制卡通用运动架构	64
4.3	物件定义	64
	表 4.1: 缩写及其含义	64
4.4	设备编号	65

第 5 章 属性配置文件说明 67

5.1	本章简介	68
5.2	系统配置基本概念	68
5.2.1	硬件资源配置	68
5.2.2	软件资源配置	71
5.2.3	资源整合	73

5.3	配置文件生成与下载.....	74
5.3.1	配置文件下载函数.....	74
5.3.2	配置文件重点说明.....	74
5.4	配置信息修改.....	75
5.5	控制器配置初始化状态.....	75
5.5.1	硬件资源配置初始化状态.....	75
5.5.2	软件资源配置初始化状态.....	77
5.6	伺服操作.....	78
5.7	脉冲输入输出方式.....	78
5.7.1	脉冲输入输出方式相关属性.....	78
5.8	浮點 PPU.....	78
5.8.1	浮点 PPU 相关属性.....	78
5.8.2	浮点 PPU 功能介绍.....	78
5.8.3	浮点 PPU 重点说明.....	79
5.8.4	例程.....	79
5.9	系统密码保护.....	80
5.9.1	密码保护函数.....	80
5.9.2	密码保护重点说明.....	80
5.9.3	例程.....	80

第 6 章 运动状态 81

6.1	本章简介.....	82
6.2	轴状态.....	82
6.2.1	轴状态相关函数.....	82
6.2.2	重点说明.....	82
	表 6.1: 轴当前状态定义.....	82
	表 6.2: 轴当前运动状态定义.....	83
6.2.3	例程.....	83
6.3	轴速度.....	84
6.3.1	轴速度相关函数与属性.....	84
6.3.2	轴速度重点说明.....	84
6.3.3	例程.....	85
6.4	编码器.....	85
6.4.1	编码器相关函数.....	85
6.4.2	编码器重点说明.....	85
6.4.3	例程.....	86
6.5	轴的运动 I/O 状态.....	88
6.5.1	轴的运动 I/O 状态函数.....	88
6.5.2	轴的运动 I/O 状态重点说明.....	88
6.5.3	例程.....	89
6.6	群组状态.....	89
6.6.1	群组状态相关函数.....	89
6.6.2	重点说明.....	89
	表 6.3: 群组当前状态定义.....	89
6.6.3	例程.....	90
6.7	群组速度.....	90
6.7.1	群组速度相关函数与属性.....	90
6.7.2	群组速度重点说明.....	91
6.7.3	例程.....	91

第 7 章 运动功能 93

7.1	本章简介.....	94
7.2	单轴运动模式.....	94
7.2.1	本节简介.....	94
7.2.2	点到点运动.....	94
	表 7.1: 点到点运动相关函数.....	94
	表 7.2: 点到点运动相关属性.....	95

	图 7.1: 点到点运动流程图	96
7.2.3	连续运动	97
	表 7.3: 连续运动相关函数	97
	表 7.4: 连续运动相关属性	98
	图 7.2: 连续运动流程图	98
7.2.4	变位运动	100
	表 7.5: 变位运动相关函数	100
	表 7.6: 变位运动相关属性	100
	图 7.3: 变位运动流程图	102
7.2.5	变速运动	103
	表 7.7: 变速运行相关函数	103
	表 7.8: 变速运行相关属性	104
	图 7.4: 变速运动流程图	105
7.2.6	同步起、同步停运动	106
	表 7.9: 同步起、同步停运动相关函数	106
	表 7.10: 同步起、同步停运动相关属性	107
	图 7.5: 同步起、同步停运动流程图	108
7.2.7	叠加运动	109
	表 7.11: 叠加运动相关函数	109
	表 7.12: 叠加运动相关属性	110
	图 7.6: 叠加运动	110
	图 7.7: 叠加运动流程图	111
7.2.8	JOG 运动、手轮运动	112
	表 7.13: JOG、手轮运动相关函数	112
	表 7.14: JOG、手轮运动相关属性	113
	图 7.8: JOG、手轮运动流程图	115
7.2.9	原点复归	117
	表 7.15: 原点复归相关 API	117
	表 7.16: 原点复归相关属性	117
	图 7.9: 原点复归流程图	126
7.2.10	轴的事件	128
	表 7.17: 轴的事件相关函数	128
	表 7.18: 轴的事件相关属性	129
	图 7.10: 轴的事件流程图	130
7.3	插补运动模式	132
7.3.1	本节简介	132
7.3.2	直线插补运动	132
	表 7.19: 直线插补运动相关函数	132
	表 7.20: 直线插补运动相关属性	133
	图 7.11: 直线插补运动流程图	134
7.3.3	圆弧插补	136
	表 7.21: 圆弧插补相关函数	136
	表 7.22: 圆弧插补运动相关函数	137
7.3.4	螺旋插补运动	143
	表 7.23: 螺旋插补运动相关函数	143
	表 7.24: 螺旋插补运动相关函数	144
	图 7.12: 螺旋插补运动流程图	146
7.3.5	群组的事件	148
	表 7.25: 轴的事件相关函数	148
	图 7.13: 群组事件流程图	150
7.4	跟随运动模式	152
7.4.1	本节简介	152
7.4.2	电子齿轮运动	152
	表 7.26: 电子齿轮运动相关函数	152
	表 7.27: 电子齿轮运动相关属性	153
	图 7.14: 电子齿轮流程图	154
7.4.3	电子凸轮运动	156
	表 7.28: 电子凸轮运动相关函数	156
	表 7.29: 电子凸轮运动相关属性	157
7.4.4	龙门运动	161

	表 7.30: 龙门运动相关函数.....	161
	表 7.31: 龙门运动相关属性.....	162
	图 7.15: 龙门运动流程图.....	163
7.4.5	切向跟随运动.....	165
	表 7.32: 切向跟随运动相关函数.....	165
	表 7.33: 切向跟随运动相关属性.....	165
	图 7.16: 切向跟随运动流程图.....	166
7.5	路径规划模式.....	168
7.5.1	本节简介.....	168
7.5.2	Path 运动相关函数与属性	168
	表 7.34: Path 运动相关函数	168
	表 7.35: Path 运动相关属性	168
7.5.3	Path 加载方式及运动模式	169
	表 7.36: Path 运动模式	169
	表 7.37: Path 运动相关指令	169
7.5.4	EndPath 的作用及影响	171
7.5.5	速度交接模式.....	172
	表 7.38: 速度交接模式关系表.....	172
7.5.6	Path DO 功能指令	175
	表 7.39:	176
7.5.7	Path 的执行	176
7.5.8	Path 的暂停和恢复	177
7.5.9	动态加载 Path	179
7.5.10	Path 运动流程图	181
	图 7.17: Path 运动流程图	181
7.5.11	Path 运动例程	181

第 8 章 DIO 与 AI 控制..... 185

8.1	本章简介.....	186
8.2	DI/O 与 AI 控制	186
	8.2.1 本节简介.....	186
	8.2.2 数字量输入输出.....	186
	8.2.3 访问模拟量输入.....	186
8.3	比较功能.....	188
	8.3.1 本节简介.....	188
	8.3.2 触发比较功能.....	188
8.4	锁存功能.....	189
	8.4.1 锁存单笔数据.....	189
	8.4.2 连续锁存功能.....	190
8.5	硬件安全.....	192
	8.5.1 本节简介.....	192
	8.5.2 限位.....	192
	8.5.3 报警.....	193
	8.5.4 停止.....	194

附录 A API 列表..... 195

A.1	设备特性属性.....	196
A.2	函数列表.....	199
	A.2.1 通用 API	199
	A.2.2 设备对象.....	200
A.3	DAQ.....	210
	A.3.1 数字量输入 / 输出	210
	A.3.2 模拟量输入 / 输出	212
A.4	轴.....	213
	A.4.1 系统.....	213
	A.4.2 运动 IO	214
	A.4.3 运动状态.....	215

	A. 4.4	速度运动	216
	A. 4.5	点到点运动	218
	A. 4.6	同步起停运动	220
	A. 4.7	回原点	221
	A. 4.8	位置 / 计数器控制	222
	A. 4.9	比较	223
	A. 4.10	锁存	224
	A. 4.11	Aux/Gen 输出	225
	A. 4.12	外部驱动	226
	A. 4.13	凸轮 / 齿轮	226
	A. 4.14	龙门 / 切线跟随	228
	A. 4.15	停止运动	229
A. 5		群组	230
	A. 5.1	系统	230
	A. 5.2	运动状态及速度	231
	A. 5.3	运动停止	232
	A. 5.4	插补运动	232
	A. 5.5	路径	240
	A. 5.6	暂停和恢复	244

附录 B 属性列表 245

B. 1	属性列表	246
B. 2	设备特性属性	250
B. 3	设备配置属性	253
B. 4	DAQ 特性属性	255
B. 5	DAQ 配置属性	256
B. 6	轴的特性属性	257
B. 6. 1	系统	257
B. 6. 2	速度模式	258
B. 6. 3	脉冲输入	259
B. 6. 4	脉冲输出	259
B. 6. 5	报警	260
B. 6. 6	到位	261
B. 6. 7	报警清除 (ERC)	261
B. 6. 8	减速 (SD)	261
B. 6. 9	硬件限位	262
B. 6. 10	软件限位	262
B. 6. 11	原点	263
B. 6. 12	背隙补偿	264
B. 6. 13	比较	264
B. 6. 14	锁存	264
B. 6. 15	Cam DO	265
B. 6. 16	外部驱动	265
B. 6. 17	外部驱动	266
B. 6. 18	Aux/Gen 输出	266
B. 6. 19	同步启停	267
B. 6. 20	触发停止	267
B. 7	轴的配置属性	268
B. 7. 1	系统	268
B. 7. 2	速度模式	269
B. 7. 3	脉冲输入	270
B. 7. 4	脉冲输出	270
B. 7. 5	报警	271
B. 7. 6	伺服到位	272
B. 7. 7	ERC	272
B. 7. 8	硬件限位	273
B. 7. 9	软件限位	274
B. 7. 10	原点	275

	B. 7. 11 背隙补偿.....	276
	B. 7. 12 比较.....	276
	B. 7. 13 锁存.....	278
	B. 7. 14 Aux、Gen 输出	279
	B. 7. 15 外部驱动.....	279
	B. 7. 16 凸轮区间 DO	281
	B. 7. 17 轴单圈脉冲数.....	282
	B. 7. 18 同步启用.....	282
	B. 7. 19 触发停止.....	283
B. 8	轴的参数属性.....	286
	B. 8. 1 速度参数.....	286
	B. 8. 2 原点.....	287
B. 9	群组的配置属性.....	289
	B. 9. 1 系统.....	289
	B. 9. 2 路径.....	289
B. 10	群组的参数属性.....	290
	B. 10. 1 速度模式.....	290
	B. 10. 2 系统.....	291
	B. 10. 3 路径.....	291

附录 C 错误代码 293

C. 1	错误代码.....	294
C. 2	常见错误.....	294
C. 3	DSP 常见错误	305
C. 4	不常见错误.....	318

第 1 章

概述

本章介绍 Common Motion 系列的基本信息、特殊特性以及详细规格。

1.1 本章简介

本章将列出所有研华 Common Motion 系列运动控制卡具有的功能，方便用户使用。

1.2 PCI-1285/1285E 运动控制卡

PCI-1285/1285E 系列运动控制卡是基于 DSP 的 SoftMotion PCI 总线的 8 轴运动控制器卡，其具有如下表所示功能。

表 1.1: PCI-1285/85E 系列运动控制卡功能表

项目	说明	PCI-1285	PCI-1285E
单轴运动	点到点运动	√	√
	连续运动	√	√
	变位运动	√	√
	变速运动	√	√
	背隙补偿	√	√
	T&S 形速度曲线	√	√
	运动中重设轴的加速度和减速度	√	√
	同步起、同步停运动	√	不支持
	叠加运动	√	不支持
	JOG 运动	√	√
	手轮运动	√	√
	原点复归 (16 种模式)	√	√
多轴运动 (群组运动)	群组个数	4 个群组	4 个群组
	直线插补	2~3 轴	2 轴
	直接线性插补	2~8 轴	2 轴
	2 轴圆弧插补	√	不支持
	3 轴圆弧插补	√	不支持
	螺旋插补	3~8 轴	不支持
	暂停和恢复执行功能	√	√
	运动中改变组的运行速度	√	不支持
跟随运动	电子齿轮	√	√
	电子凸轮	不支持	不支持
	龙门	√	不支持
	CAM DO	√	不支持
	切向跟随	√	不支持
路径运动	加载路径表功能	4 个表，每个表 7000 段 Path	4 个表，每个表 7000 段 Path
	暂停 / 恢复运动	√	√
	添加直线运动	√	√
	添加圆弧运动	√	不支持
	添加螺旋运动	√	不支持
	延迟功能	√	√
	添加 DO 开关运动	√	√
	开始 / 停止 / 重复	√	√
	路径速度交接功能	√	不支持
	路径速度前瞻功能	√	不支持

表 1.1: PCI-1285/85E 系列运动控制卡功能表

触发功能	轴的单一比较 (8 个通道)	√	不支持
	轴的表比较 (2 个通道)	√	不支持
	轴的线性比较 (表大小 :100K 个点)	√	不支持
位置锁存	锁存位置数据	√	不支持
中断	轴的停止运行功能	√	√
	轴的比较	√	不支持
	轴的误差反馈	√	√
	轴的锁存	√	不支持
	轴 VH 开始	√	√
	轴 VH 结束	√	√
	群组的停止运行功能	√	√
	群组 Motion Done 事件	√	√
	群组 VH 开始事件	√	√
	群组 VH 结束事件	√	√
Motion DAQ	运动数据采集	不支持	不支持

1.3 PCI-1265 运动控制卡

PCI-1265 运动控制卡是基于 DSP 的 SoftMotion PCI 总线的 6 轴控制器卡，专为各种电机自动化和其它机器自动化的广泛应用设计。其具有的功能如下表所示。

表 1.2: PCI-1265 运动控制卡功能表

项目	说明	PCI-1265
单轴运动	点到点运动	√
	连续运动	√
	变位运动	√
	变速运动	√
	背隙补偿	√
	T&S 形速度曲线	√
	运动中重设轴的加速度和减速度	√
	同步起、同步停运动	√
	叠加运动	√
	JOG 运动	√
	手轮运动	√
	原点复归 (16 种模式)	√
多轴运动 (群组运动)	群组个数	3 个群组
	直线插补	2~3 轴
	直接线性插补	2~6 轴
	2 轴圆弧插补	√
	3 轴圆弧插补	√
	螺旋插补	3~6 轴
	暂停和恢复执行功能	√
	运动中改变组的运行速度	√

表 1.2: PCI-1265 运动控制卡功能表

跟随运动	电子齿轮	√
	电子凸轮	√
	龙门	√
	CAM DO	√
	切向跟随	√
路径表运动	加载路径表功能	3 个表, 每个表 10000 个点
	暂停 / 恢复运动	√
	添加直线运动	√
	添加圆弧运动	√
	添加螺旋运动	√
	延迟功能	√
	添加 DO 开关功能	√
	开始 / 停止 / 重复	√
	路径速度交接功能	√
	路径速度前瞻功能	√
触发功能	轴的单一比较 (6 个通道)	√
	轴的表比较 (2 个通道)	√
	轴的线性比较 (表大小 :100K 个点)	√
位置锁存	锁存位置数据	√
中断	轴的停止运行功能	√
	轴的比较	√
	轴的错误反馈	√
	轴的锁存	√
	轴 VH 开始	√
	轴 VH 结束	√
	群组的停止运行功能	√
	群组 Motion Done 事件	√
	群组 VH 开始事件	√
	群组 VH 结束事件	√
Motion DAQ	运动数据采集	√

1.4 PCI-1245/45E/45V 运动控制卡

PCI-1245 系列运动控制卡是基于 DSP 的 SoftMotion PCI 总线的 4 轴控制器卡，专为各种电机自动化和其它机器自动化的广泛应用设计。其具有的功能如下表所示。

表 1.3: PCI-1245 系列板卡功能表

项目	说明	PCI-1245	PCI-1245E	PCI-1245V
单轴运动	点到点运动	√	√	√
	连续运动	√	√	√
	变位运动	√	√	√
	变速运动	√	√	√
	背隙补偿	√	√	√
	T&S 形速度曲线	√	√	√
	重设轴加速度和减速度	√	√	√
	同步起、同步停运动	√	不支持	不支持
	叠加运动	√	不支持	不支持
	JOG 运动	√	√	√
	手轮运动	√	√	√
	原点复归 (16 种模式)	√	√	√
多轴运动 (群组运动)	群组个数	2 个群组	2 个群组	2 个群组
	直线插补	2~3 轴	2 轴	2~3 轴
	直接线性插补	2~4 轴	2 轴	2~4 轴
	2 轴圆弧插补	√	不支持	√
	3 轴圆弧插补	√	不支持	不支持
	螺旋插补	3~4 轴	不支持	不支持
	暂停和恢复执行功能	√	√	√
	运动中改变组的运行速度	√	不支持	不支持
跟随运动	电子齿轮	√	√	√
	电子凸轮	√	不支持	不支持
	龙门	√	不支持	不支持
	CAM DO	√	不支持	不支持
	切向跟随	√	不支持	不支持
路径表运动	加载路径表功能	3 个表，每个表 10K 个点	3 个表，每个表 10K 个点	3 个表，每个表 10K 个点
	暂停 / 恢复运动列表	√	√	√
	添加直线运动	√	√	√
	添加圆弧运动	√	不支持	√(2 轴)
	添加螺旋运动	√	不支持	不支持
	延迟功能	√	√	√
	添加 DO 开关运动	√	√	√
	开始 / 停止 / 重复	√	√	√
	路径速度交接功能	√	不支持	不支持
	路径速度前瞻功能	√	不支持	不支持
	轴的单一比较 (8 个通道)	√	不支持	√
触发功能	轴的表比较 (2 个通道)	√	不支持	√
	轴的线性比较 (表大小 : 100K 个点)	√	不支持	√
位置锁存	锁存位置数据	√	不支持	√

表 1.3: PCI-1245 系列板卡功能表

中断	轴的停止运行功能	√	√	√
	轴的比较	√	不支持	√
	轴的错误反馈	√	√	√
	轴的锁存	√	不支持	√
	轴 VH 开始	√	√	√
	轴 VH 结束	√	√	√
	群组的停止运行功能	√	√	√
	群组 Motion Done 事件	√	√	√
	群组 VH 开始事件	√	√	√
	群组 VH 结束事件	√	√	√
Motion DAQ	运动数据采集	√	√	√

1.5 PCI-1245L 运动控制卡

PCI-1245L 是 4 轴的 SoftMotion PCI 总线控制器卡。其具有的功能如下表所示。

表 1.4: PCI-1245L 系列板卡功能表

项目	说明	PCI-1245L
单轴运动	点到点运动	√
	连续运动	√
	变位运动	√
	变速运动	√
	背隙补偿	√
	T&S 形速度曲线	√
	重设轴加速度和减速度	√
	同步起、同步停运动	√
	叠加运动	不支持
	JOG 运动	√
	手轮运动	√
	原点复归 (16 种模式)	√
多轴运动 (群组运动)	群组个数	1 个群组
	直线插补	2 轴插补
	直接线性插补	2 轴插补
	2 轴圆弧插补	不支持
	3 轴圆弧插补	不支持
	螺旋插补	不支持
	暂停和恢复执行功能	不支持
	运动中改变组的运行速度	不支持
跟随运动	电子齿轮	不支持
	电子凸轮	不支持
	龙门	不支持
	CAM DO	不支持
	切向跟随	不支持

表 1.4: PCI-1245L 系列板卡功能表

路径表运动	加载路径表功能	不支持
	暂停 / 恢复运动	不支持
	添加直线运动	不支持
	添加圆弧运动	不支持
	添加螺旋运动	不支持
	延迟功能	不支持
	添加 DO 开关运动	不支持
	开始 / 停止 / 重复	不支持
	路径速度交接功能	不支持
	路径速度前瞻功能	不支持
触发功能	轴的单一比较 (8 个通道)	不支持
	轴的表比较 (2 个通道)	不支持
	轴的线性比较 (表大小 :100K 个点)	不支持
中断	轴的停止运行功能	√
	轴的比较	不支持
	轴的错误反馈	√
	轴的锁存	不支持
	轴 VH 开始	√
	轴 VH 结束	√
	群组的停止运行功能	√
	群组 Motion Done 事件	√
	群组 VH 开始事件	√
	群组 VH 结束事件	√
Motion DAQ	运动数据采集	不支持

1.6 MIC-3245/3285 运动控制卡

MIC-3245/3285 系列运动控制卡是基于 DSP 的 SoftMotioncPCI 总线的 4/8 轴运动控制器卡，其具有如下表所示功能。

项目	说明	MIC-3245	MIC-3285
单轴运动	点到点运动	√	√
	连续运动	√	√
	变位运动	√	√
	变速运动	√	√
	背隙补偿	√	√
	T&S 形速度曲线	√	√
	运动中重设轴的加速度和减速度	√	√
	同步起、同步停运动	√	√
	叠加运动	√	√
	JOG 运动	√	√
	手轮运动	√	√
	原点复归 (16 种模式)	√	√

多轴运动 (群组运动)	群组个数	2 个群组	4 个群组
	直线插补	2~3 轴	2~3 轴
	直接线性插补	2~4 轴	2~8 轴
	2 轴圆弧插补	√	√
	3 轴圆弧插补	√	√
	螺旋插补	3~4 轴	3~8 轴插补
	暂停和恢复执行功能	√	√
	运动中改变组的运行速度	√	√
跟随运动	电子齿轮	√	√
	电子凸轮	√	不支持
	龙门	√	√
	CAM DO	√	√
	切向跟随	√	√
路径运动	加载路径表功能	3 个表，每个表 10K 个点	4 个表，每个表 7000 段 Path
	暂停 / 恢复运动	√	√
	添加直线运动	√	√
	添加圆弧运动	√	√
	添加螺旋运动	√	√
	延迟功能	√	√
	添加 DO 开关运动	√	√
	开始 / 停止 / 重复	√	√
	路径速度交接功能	√	√
	路径速度前瞻功能	√	√
触发功能	轴的单一比较 (8 个通道)	√	√
	轴的表比较 (2 个通道)	√	√
	轴的线性比较 (表大小 :100K 个点)	√	√
位置锁存	锁存位置数据	√	√
中断	轴的停止运行功能	√	√
	轴的比较	√	√
	轴的错误反馈	√	√
	轴的锁存	√	√
	轴 VH 开始	√	√
	轴 VH 结束	√	√
	群组的停止运行功能	√	√
	群组 Motion Done 事件	√	√
	群组 VH 开始事件	√	√
	群组 VH 结束事件	√	√
MotionDAQ	运动数据采集	√	不支持

第 2 章

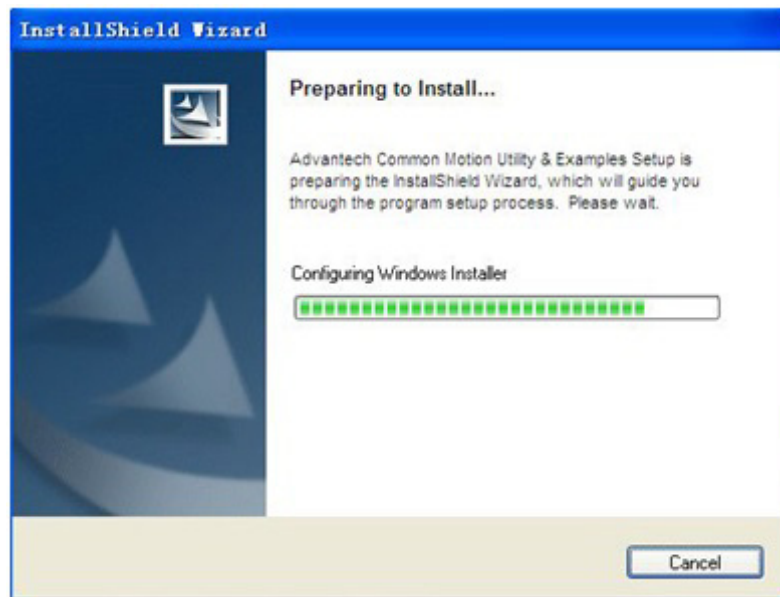
Common Motion Utility

2.1 Common Motion Utility 简介

Common Motion Utility 是研华为用户专门提供的测试用工具，该工具具有良好的测试界面，并可在该工具中进行参数配置，如设置速度参数，报警、硬件极限等信号的高低电平。该工具按照通用运动架构由 .Net 控件类库开发。该测试工具支持 PCI-1220U/PCI-1240U/PCI-1245/1245V/1245E/1265/PCI-1285 系列产品。

2.2 Common Motion Utility 安装

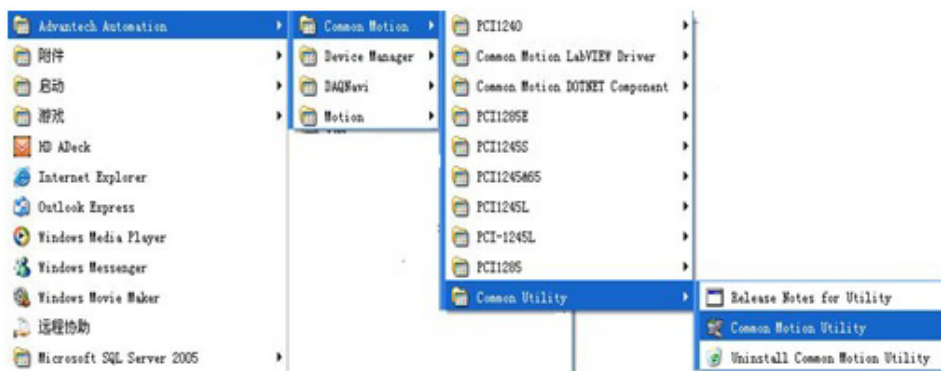
用户可在研华官网下载 Utility 安装包（或从光盘中获得），根据上位机平台类型选择安装 X86/X64 版本的 Utility，运行 exe 安装文件，如下所示。



第二步: 选择路径安装，默认路径为 “系统盘符 : \Program\Files\Advantech\Common Motion\”



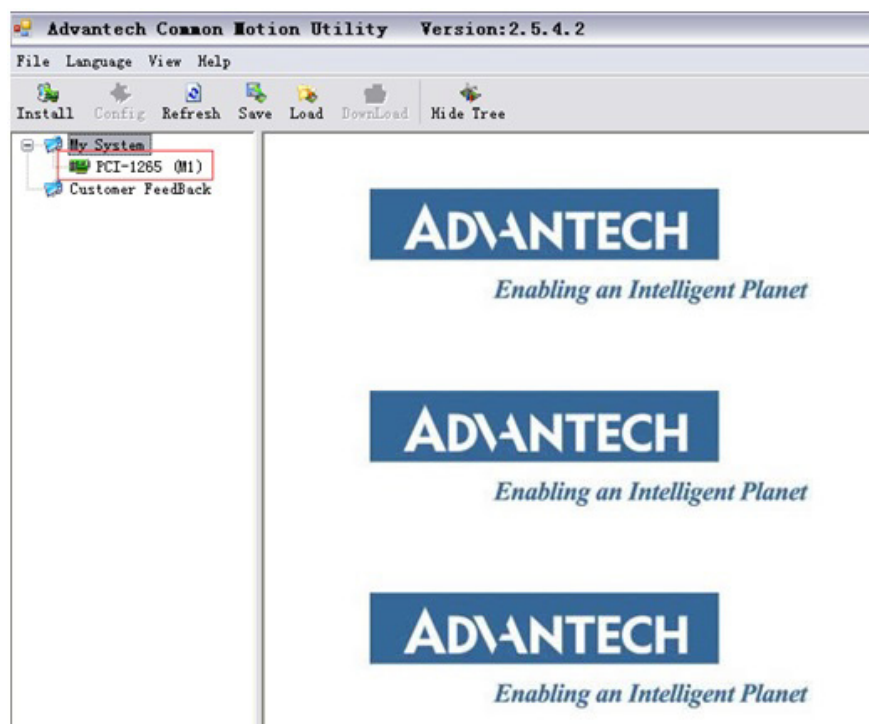
第三步：安装完成后，打开“开始 → 所有程序”，即可打开 Utility。



2.3 Common Motion Utility 用户界面

2.3.1 主界面

安装完成后，打开 Utility，将出现如下主界面，左边为设备列表，显示电脑上所有运动控制卡以及虚拟轴卡。



2.4 主界面 -- 菜单栏

主界面上菜单栏如下图所示：



2.4.1 文件 (File) 菜单项



单击 [Exit] 终止该进程。

2.4.2 语言 (Language) 菜单项



通过该菜单可切换测试工具的语言。测试工具支持三种语言：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，所选语言会将当前选择的 Utility 操作语言保存至注册表。

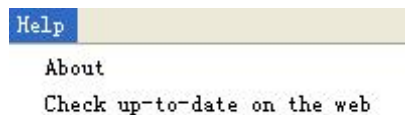
再次打开时，测试工具的语言将为上次使用的语言。

2.4.3 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。

2.4.4 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” 界面的版本信息检查固件、驱动和测试工具是否为最新版。

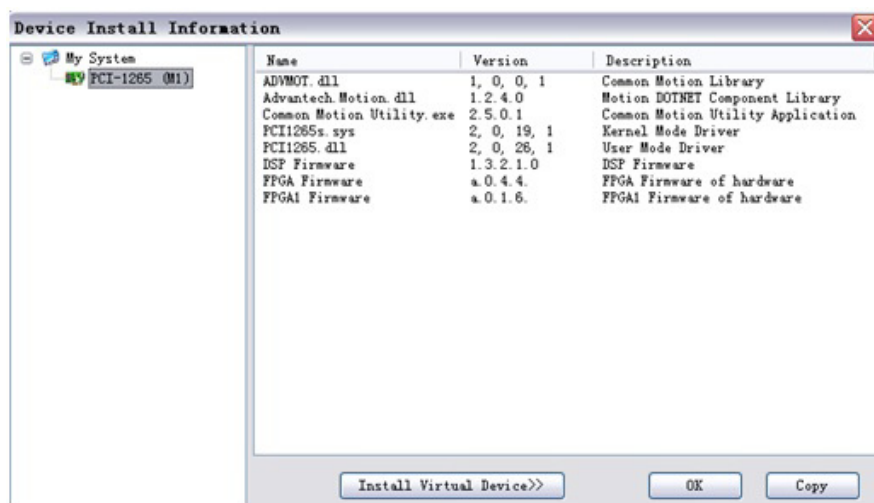
2.5 主界面 -- 工具栏

主界面上工具栏如图所示：



2.5.1 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。



点击 [Copy]，前面两列的信息将被保存到剪贴板中，方便客户将版本信息保存到 Word/ Txt 文件中。该窗口的第一栏为名称，第二栏为版本号，第三栏为说明信息。

ADVMOT.dll 是运动板卡的通用开发接口，Advantech.Motion.dll 为 .NET 运动控件程序集。

Common MotionUtility.exe 是正在运行的测试工具。

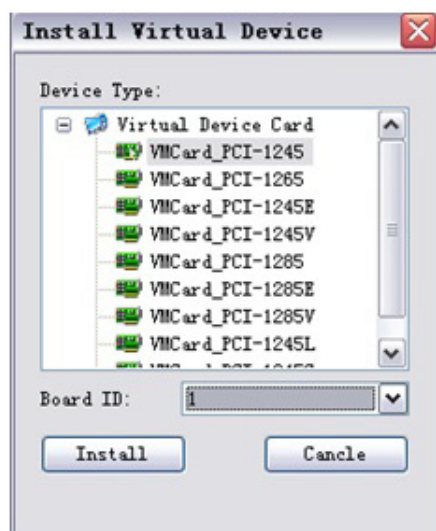
第四行和第五行是驱动文件（内核模式和用户模式），取决于设备类型。

第六行为 DSP 固件，第七行为硬件的 FPGA。

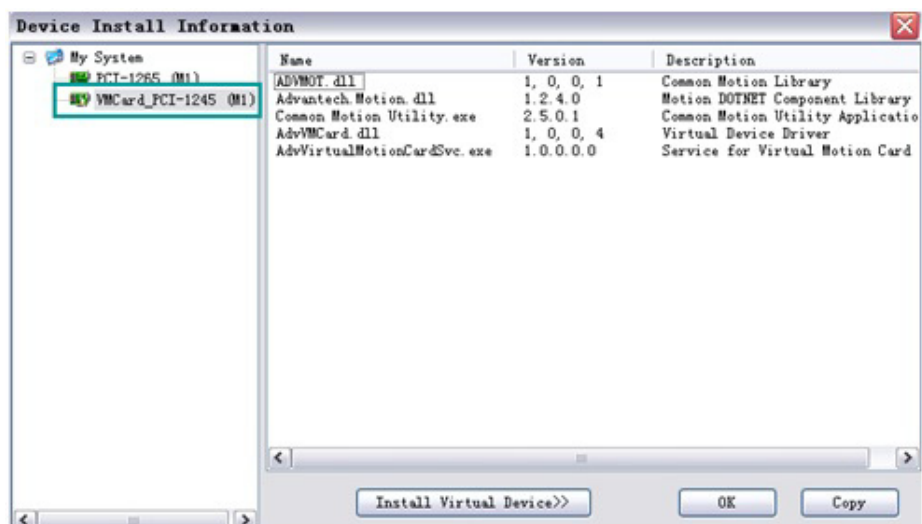
注！ 只有基于 DSP 的运动控制卡才会显示 DSP 和 FPGA 信息。



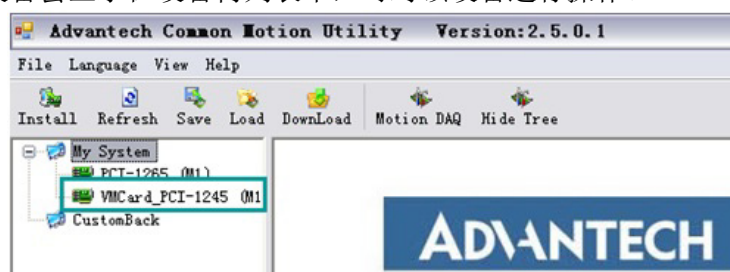
在该窗口中可安装虚拟板卡，如上图点击 ” Install Virtual Device” 按钮，出现如下对话框，选择设备类型和 BoardID，点击 ” Install”，即可安装该类型的虚拟设备。



安装完成后，该设备将出现在如下对话框中，选择该虚拟设备，将出现虚拟设备的安装信息。



同时该虚拟设备会显示在设备树列表中，可对该设备进行操作。



2.5.2 更新 (Refresh) 工具项

该按钮用于刷新功能。单击 [Refresh] 将重新加载设备树。操作后，默认没有选择任何设备。

2.5.3 保存 (Save) 工具项

该按钮用于保存所选设备的轴的全部属性。

2.5.4 导入 (Load) 工具项

该按钮用于导入所选设备的全部轴的配置信息。选择设备后，单击该按钮将出现“Open Dialog”对话框。选择之前导出的配置文件并单击 [OK]，用户即可将配置文件导入到设备硬件。


2.5.5 下载 (Download) 工具项

PCI-1245/1245V/1245E/1265 系列产品是基于 DSP 的运动控制器。单击设备后，用户将看到如下界面。




该工具按钮可实现 DSP Firmware 和 FPGA Firmware 的下载。FPGA Firmware 又分为 U2 和 U7 的下载。FPGA Firmware 的下载动作同 DSP Firmware。需要注意的是，FPGA Firmware 下载后，需要关机后再重启才真正更新。

对话框的顶部显示当前设备类型、设备名称及固件版本。单击 [Open File] 选择获取的最新固件文件。单击 [Start Download] 将激活硬件的下载过程，进度栏将显示任务进程。

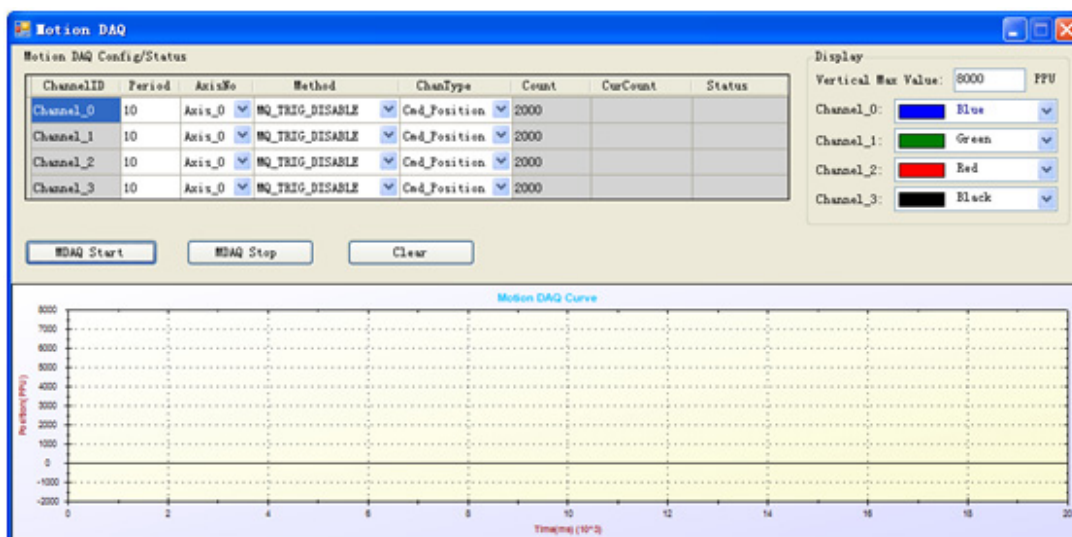
- 注!**  1. 单击 [Start Download] 后，下载固件至硬件过程中，对话框将不能关闭。
2. 下载过程中，如果由于断电或其它问题导致下载过程未能完成，硬件需要返回研华公司进行固件升级。

2.5.6 数据采集 (MotionDAQ) 工具栏

该工具按钮主要展示运动数据获取 (Motion Data Acquisition) 功能。在 PCI-1245/1245V/1245E/1265 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。

- 注!**  目前只有 PCI-1245, PCI-1245V, PCI-1245E 及 PCI-1265 支持此功能，能看到此工具按钮。

单击按钮后，用户将看到如下界面：



此界面由以下几部分组成：

- Motion DAQ 配置 / 状态 (Motion DAQ Config/Status)
- 功能操作
- 图形显示
- Display

关于以上各部分的功能和操作分别介绍如下：

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

Motion DAQ Config/Status							
ChannelID	Period	AxisNo	Method	ChanType	Count	CurCount	Status
Channel_0	10	Axis_0	MQ_TRIG_DISABLE	Cnd_Position	2000		
Channel_1	10	Axis_0	MQ_TRIG_DISABLE	Cnd_Position	2000		
Channel_2	10	Axis_0	MQ_TRIG_DISABLE	Cnd_Position	2000		
Channel_3	10	Axis_0	MQ_TRIG_DISABLE	Cnd_Position	2000		

可直接在 DataGridView 中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（背景色为灰色的）。

ChannelID: 表示为哪一个通道。共提供 4 个通道。

--Channel_0

--Channel_1

--Channel_2

--Channel_3

AxisNo: 可选择设备中的任一轴。

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 值将会随着变化。

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: 由软件触发（点击 MDAQ Start 触发）

--MQ_TRIG_DI: 由 DI 触发（保留）

--MQ_TRIG_AX0_START: 0 轴开始运动时触发

--MQ_TRIG_AX1_START: 1 轴开始运动时触发

--MQ_TRIG_AX2_START: 2 轴开始运动时触发

--MQ_TRIG_AX3_START: 3 轴开始运动时触发
 --MQ_TRIG_AX4_START: 4 轴开始运动时触发
 --MQ_TRIG_AX5_START: 5 轴开始运动时触发
 --MQ_TRIG_AX6_START: 6 轴开始运动时触发
 --MQ_TRIG_AX7_START: 7 轴开始运动时触发

PCI-1245、PCI-1245V, 支持 MQ_TRIG_DISABLE ~ MQ_TRIG_AX3_START 触发方式。
 PCI-1265 支持 MQ_TRIG_DISABLE ~ MQ_TRIG_AX5_START 种触发方式。此外, 由 DI 触发的方式暂保留。

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 (保留): 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;

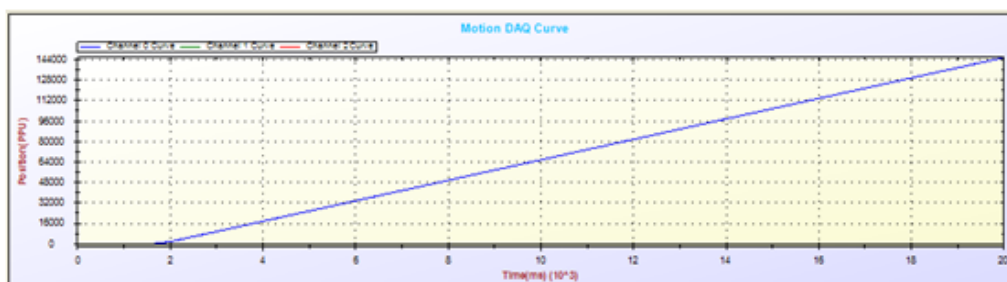
鼠标离开编辑框后, 设置值将生效。可在 DataGridView 查看目前各个通道的配置情况, 如下图:

2. 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 Channel 的曲线图

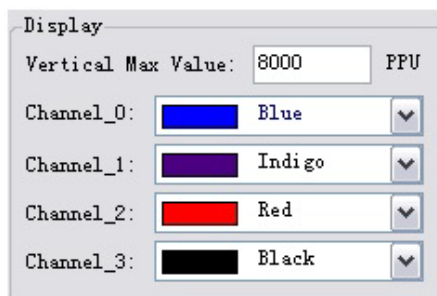
3. 图形显示

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. 显示 (Display)

右上角的 Display 区域用于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。



2.5.7 隐藏树 (Hide Tree) 工具栏

为方便用户隐藏 / 显示设备树，提供此工具按钮。

若 Device Tree 目前显示，则点击将隐藏 Device Tree，按钮上的文字变为 “Show Tree”；

若 Device Tree 已隐藏，则点击将显示 Device Tree，按钮上的文字变为 “Hide Tree”。

2.6 Common Motion Utility 单轴运动控制

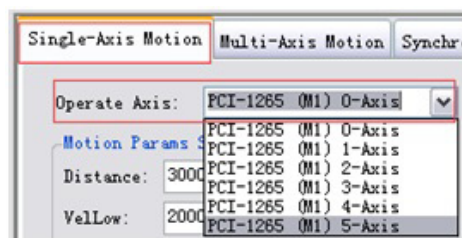
在设备数列表中选择操作的设备，出现单轴运动控制界面，如下图所示：



图 2.1：单轴运动控制界面

2.6.1 轴操作 (Operate Axis)

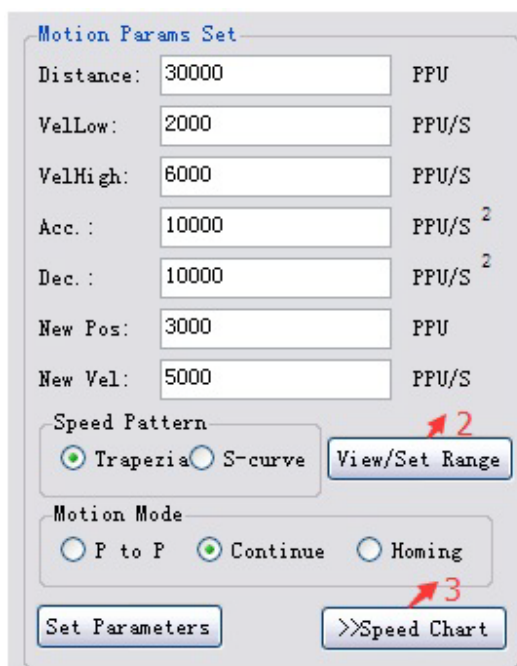
选择操作轴。单击复选框下拉列表图标，将显示所选设备的全部轴。并对选择的轴进行操作。



2.6.2 轴运动参数设置 (Motion Param Set)

完成操作参数设置后，单击 [Set Parameters] 将参数值设置到设备中。

完成如下图中距离、速度等操作参数设置后，单击 [Set Parameters]，将参数值设置到设备中。

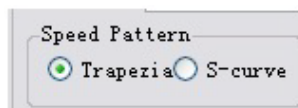


左图的 "2" 按钮为查看或设置最大速度、加速度、减速度按钮
左图的 "3" 按钮为查看速度曲线图按钮

2、3 按钮说明见 2.6.3、2.6.4 节

2.6.2.1 速度模式设置 (Speed Pattern)

设置运动的速度模式，可设置为梯形 (Trapezi) 或 S 形 (S-curve)。



2.6.2.2 运动模式设置 (Motion Mode)

选择运动模式。单轴运动有三种运动模式：P to P 点对点 (P To P)、连续运动 (Continue) 以及回原点运动 (Homing)。



2.6.3 配置 (Configuration)

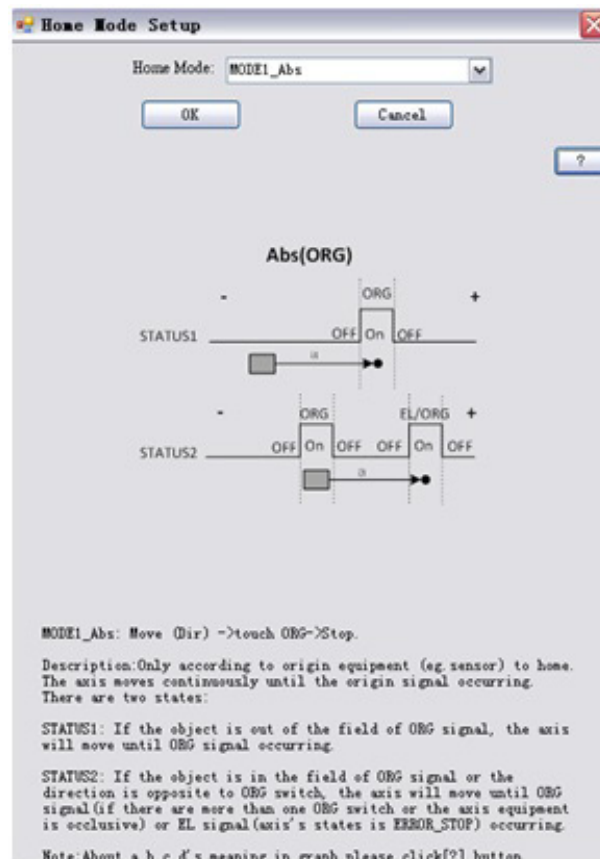
包括轴的回原点模式 (Home Mode)、外部驱动 (External Drive) 模式、轴的属性配置和轴的状态显示。



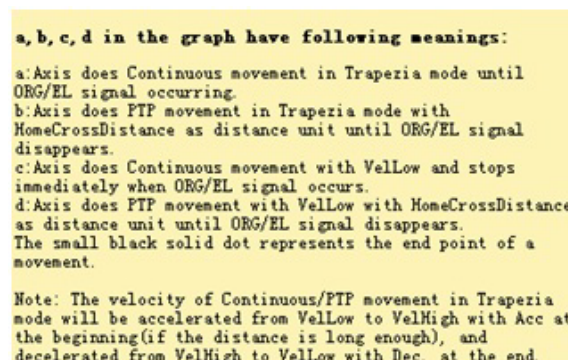
2.6.3.1 Home Mode

进行该轴的 Homing 操作前，需选择回 Home 模式。板卡提供了 16 种模式，是 ORG（返回原点）、Lmt（返回限位点）和 EZ（找到 Z 相位）的一种或任意组合。有关详细信息，请参考一章。单击 [Home Mode] 将出现如下对话框：

单击 [Home Mode] 将出现如下对话框：

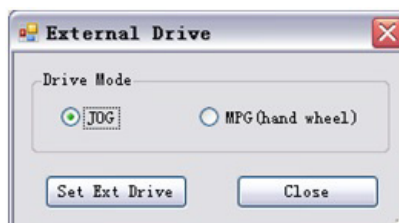


用户可从下拉列表框中选择任一模式，下面有相应的图解说明。用户可单击 [OK] 选择“Home Mode” 下拉列表框中的一种模式，或单击 [Cancel] 取消操作。默认设置为“Mode1_Abs”。点击 [?] 按钮将出现一个帮助窗体，说明图形中 a, b, c, d 和小黑实心圆点的含义，如下图：



2.6.3.2 外部驱动 (External Driver)

单击 [External Dirve] 将出现如下对话框，用户可选择一种外部驱动模式 (JOG/MPG) 来操作外部驱动。



选择 “JOG” 或 “MPG” 并单击 [Set Ext Drive]，外部驱动模式将设置成功，用户即可操作外部驱动。单击 [Close] 将关闭对话框，外部驱动设置为 “Disable”。

注！ 对于 PCI-1245/1245V/1245E/1265 series 系列产品，只有 0 轴可作为外部驱动的主轴。



2.6.3.3 轴配置 (Axis SetUp)

单击 Axis SetUp 按钮可检查 / 设置轴的属性。

轴配置界面由以下 4 部分组成：

- 左侧树状图显示板卡的属性分类列表

如果所选设备没有对应功能，则不在树状图中显示

- 中间表格显示分类中的属性和对应的属性值

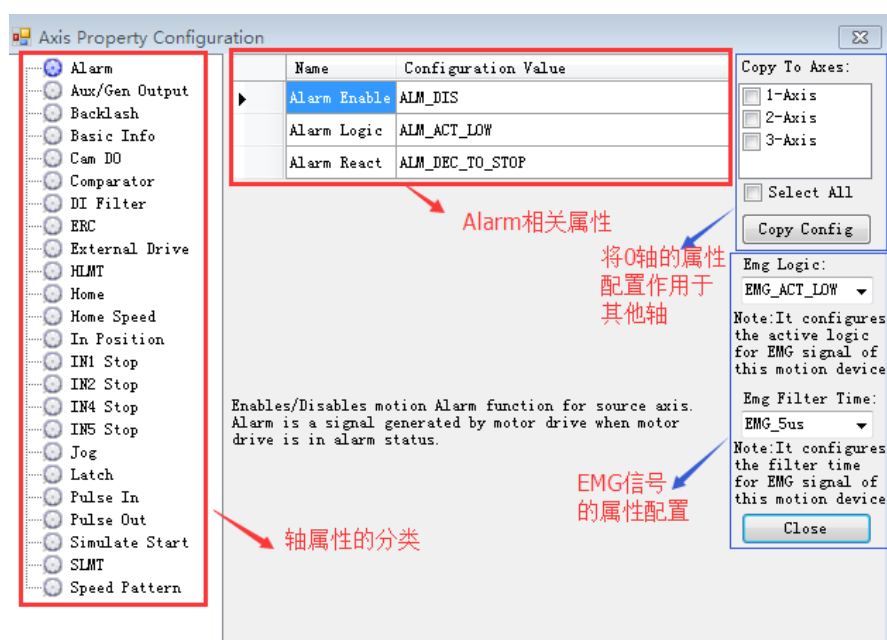
表格中的属性值是在选中设备后，从该设备的 .cfg 配置文件中读取（配置文件在 Utility 的安装目录下）。编辑完成后，属性值将在鼠标离开编辑框时生效（即完成配置）


关于配置文件详细介绍参加第五章节

- 右上方操作表示可将配置好的属性值作用于其他轴

只需勾选右侧复选框中相应的轴，然后单击 [Copy Config]

- 右下方可操作 EMG 信号的相关属性




注! 如用户希望将本次配置的信息保存, 可点击 *Utility* 测试工具中的  *Save* 按钮, 点击 *Load* 可导入保存的配置文件。


左侧的树状图显示轴属性的分类。当用户单击对应项目时, 右侧的数据显示将列出分类中的属性和对应属性值。属性分类如下:

分类	名称	简要介绍
Alarm	Alarm Enable	启用 / 禁用源轴的运动报警功能
	Alarm Logic	设置报警信号的有效逻辑电平
	Alarm React	设置报警信号的反应模式
Aux/Gen Output	AuxOut Enable	启用 / 禁用轴所在群组添加延时功能时辅助输出功能
	AuxOut Time	设置源轴在群组 AddPathDwell 功能中, 辅助输出的启动时间
	GenDo Enable	启用 / 禁用轴 DO 作为源轴的通用 DO 功能
Backlash	Backlash Enable	启用 / 禁用源轴的背隙补偿功能
	Backlash Pulses	设置源轴的补偿脉冲个数。方向发生变化时, 发送命令之前, 轴输出背隙校正脉冲
	Backlash Velocity	设置背隙补偿的速度
Basic Info	PhyID	源轴的物理 ID
	PPU	该轴的虚拟单位: PPU 个脉冲 / 单位。可以根据实际的马达来设置 PPU, 以消除不同马达精度不同的问题
	ModuleRange	轴单圈脉冲数范围
Cam DO	CamDO Enable	启用 / 禁用源轴的 CAM DO 功能
	CamDO Logic	设置 CAM DO 信号的有效逻辑电平
	CamDO Compare Source	设置 CAM DO 信号的比较源
	CamDO Mode	设置 CAM DO 信号的模式
	CamDO Direction	设置 CAM DO 的方向
	CamDO Low Limit	设置 CAM DO 信号的下边界位置
	CamDO High Limit	设置 CAM DO 信号的上边界位置
Comparator	Compare Enable	启用 / 禁用源轴的轴比较器
	Compare Source	设置比较器的源
	Compare Method	设置比较器的方法
	Compare Pulse Mode	设置比较器的脉冲模式
	Compare Pulse Logic	设置比较器脉冲的有效逻辑电平
	Compare Pulse Width	设置比较器的脉冲宽度
ERC	Erc Logic	设置 ERC 信号的有效逻辑电平
	Erc On Time	设置 ERC 的启动时间
	Erc Off Time	设置 ERC 的关闭时间
	Erc Enable Mode	启用 / 禁用源轴的 ERC 输出

External Drive	Ext Master Src	表示该轴被哪个轴的外来信号控制
	Ext Sel Enable	当外部驱动使能时，该属性表示通过数字输入通道使能轴驱动选择
	Ext Pulse Num	手轮模式时，输入脉冲边缘触发时输出的驱动脉冲
	Ext Preset Num	JOG 模式时，输入脉冲边缘触发时输出的驱动脉冲
	Ext Pulse In Mode	设置外部驱动的脉冲输入模式
HLMT	HLMT Enable	启用 / 禁用硬件限位信号
	HLMT Logic	设置硬件限位信号的有效逻辑电平
	HLMT React	设置硬件限位信号的反应模式
Home	Home Ex Mode	设置 HomeEx() 的停止模式
	Home Cross Distance	设置 Homing 运动中，Search 模式时的每一次 Search 的距离（详见 Home Mode 中的描述）
	Home Ex Switch Mode	设置 HomeEx() 的停止条件
	ORG Logic	设置 ORG 信号的有效逻辑电平
	EZ Logic	设置 EZ 信号的有效逻辑电平
	Home Reset Enable	源轴返回原点时，启用 / 禁用复位逻辑计数器
	ORG React	设置 ORG 信号的反应模式
In Position	Inp Enable	启用 / 禁用源轴的到位功能
	Inp Logic	设置到位信号的有效逻辑电平
Latch	Latch Enable	启用 / 禁用源轴的锁存功能
	Latch Logic	设置锁存信号的有效逻辑电平
Pulse In	Pulse In Mode	设置源轴的编码器反馈脉冲输入模式。
	Pulse In Logic	设置编码器反馈脉冲输入信号有效逻辑电平
	Pulse In Source	设置编码器反馈脉冲输入信号的源。
	Pulse In Max Frequency	设置编码器脉冲输入信号的最大频率。
Pulse Out	Pulse Out Mode	设置源轴的命令脉冲输出模式。
SD	SD Enable	启用 / 禁用源轴的 SD 信号。
	SD Logic	设置 SD 信号的有效逻辑电平。
	SD React	设置 SD 信号的反应模式。
	SD Latch	设置 SD 信号的 Latch 控制。
Simulate Start	Simulate Start Source	设置同步启停的触发源
SLMT	SLMT Mel Enable	启用 / 禁用源轴的负方向软件限位。
	SLMT Pel Enable	启用 / 禁用源轴的正方向软件限位
	SLMTN React	设置负方向软件限位的反应模式
	SLMTP React	设置正方向软件限位的反应模式
	SLMTN Value	设置负方向软件限位的值
	SLMTP Value	设置正方向软件限位的值

Speed Pattern	Max Velocity	配置源轴的最大速度
	Max Acc	配置源轴的最大加速度
	Max Dec	配置源轴的最大减速度
	Max Jerk	配置源轴的最大加加速度
	Vel Low	设置源轴的低速度（起始速度）（单位：PPU/S）
	Vel High	设置源轴的高速度（运行速度）（单位：PPU/S）
	Acc	设置源轴的加速度（单位：PPU/S ² ）
	Dec	设置源轴的减速度（单位：PPU/S ² ）
	Jerk	设置速度曲线类型：T 形曲线或 S 形曲线
Vibration	Vibration Enable	启用 / 禁用抑制源轴机械系统的振动
	Vibration Reverse Time	设置源轴的振动抑制时间。该功能主要是通过完成命令运动后马上先负向输出单个脉冲，然后正向输出单个脉冲来实现抑制机械系统的振动。
	Vibration Forward Time	设置源轴的振动抑制时间

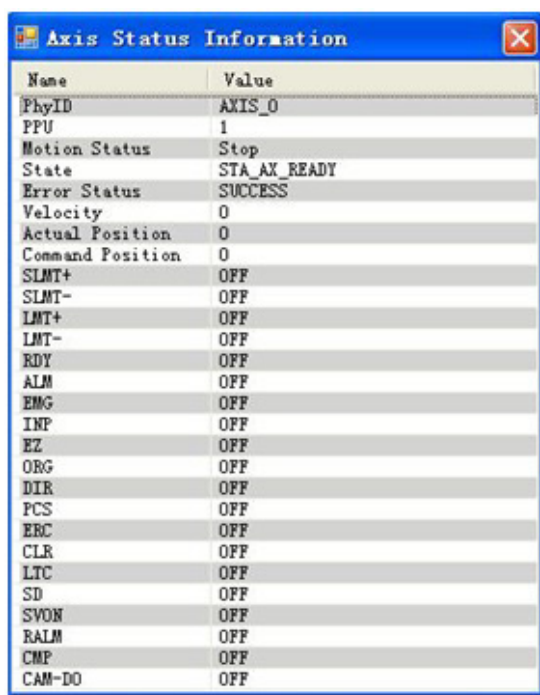
注！  在测试工具中，如果所选设备没有对应功能，则项目不会显示在树状图的左边。比如，所选设备为 PCI-1245/1265，该板卡不支持减速（SD）和振动抑制功能，因此用户在树状图左边无法看到该项。同时，由于单轴对话框支持速度参数设置，因此不会显示速度模式项。

注！  选择 “Pulse Out” 时，“Pulse Out Mode” 属性描述内容下面将出现对应模式的图解信息。

编辑完成后，属性值将在鼠标离开编辑框时生效（已经在设备中设置）。如果用户想要将属性设置复制给其它轴，只需勾选右侧复选框中相应的轴，然后单击 [Copy Config]。单击 [Close] 关闭窗口。

2.6.3.4 轴状态 (Axis Status)

单击 Axis Status 按钮查看指定的轴信息。比如，PhyID、PPU、基本状态 (Motion Status、State 和 Error Status 等) 以及 I/O 状态 (Alarm 和 SLMTP/N 等)。



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	OFF
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

2.6.4 运动测试

操作如下：



选择运动模式后，单击 [<--] 或 [-->]，轴将进行正向 / 反向的点对点 / 连续 / 返回原点运动。运动速度到达点对点运动的运行速度后，用户可单击 [Move Impose] 生成一个叠加运动。叠加运动的距离等于 New Pos 的值，叠加运动的速度等于 New Vel 的值。用户能够观察具体的运动 / 速度曲线。单击 [Stop] 停止运动。

2.6.5 运动位置 (Position) 检测



通过“Position”状态，用户能够在操作时观察理论位置和反馈位置。单击 [Reset] 可将值复位为“0”。

2.6.6 轴的当前状态 (Current Axis Status)



用户可查看当前状态和理论速度。

2.6.7 DI/O 状态

显示所选轴的4个DI端口和4个DO端口的当前状态。还可以将DO设置为“ON/OFF”。



2.6.7.1 DI

如上图所示，DI（3-0）状态从右到左依次为 DI0 到 DI3。其中，●表示 DI 有效（On）且值为 1，●表示 DI 无效（Off）且值为 0。

2.6.7.2 DO

如上图所示，DO（7-4）状态从右到左依次为 DO4 到 DO7。其中，●表示 DO 有效（On）且值为 1，●表示 DO 无效（Off）且值为 0。

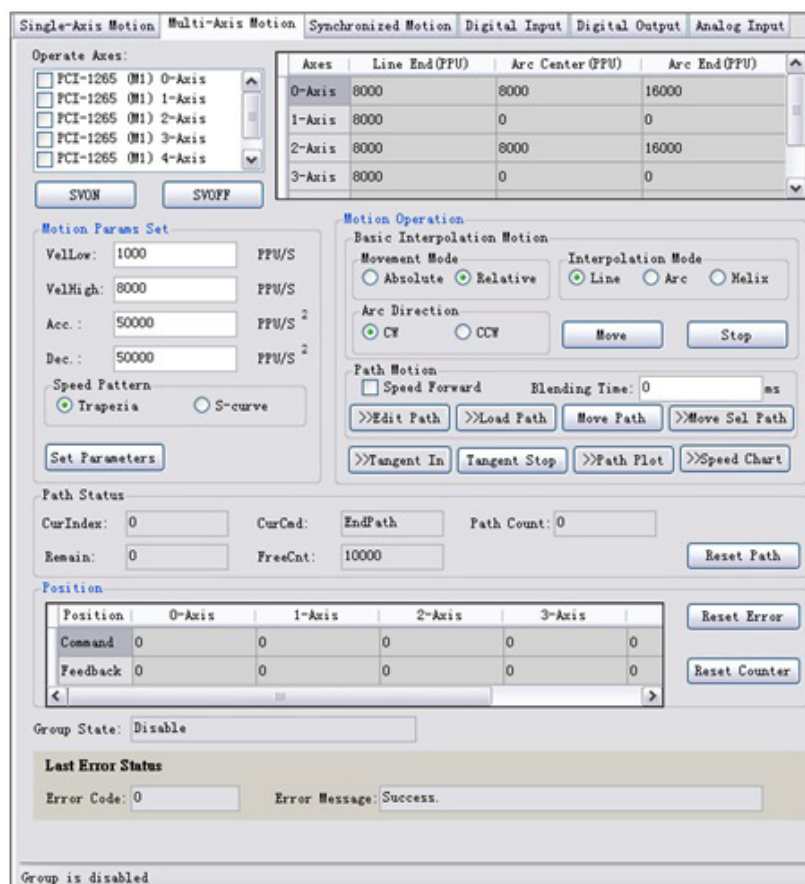
2.6.8 Last Error Status



用户可以查看最新错误代码和错误信息。如果没有错误，错误代码为“0”，错误信息为“SUCCESS”。

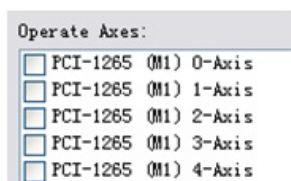
2.7 Common Motion Utility 多轴运动控制

多轴运动控制界面如下图所示。



2.7.1 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2，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”，如下图：



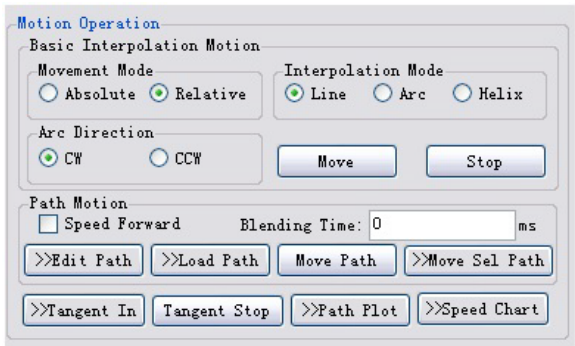
2.7.2 运动参数设置 (Motion Params Set)

运动参数设置。可对Group 进行初速度 (VelLow)、运行速度 (VelHigh)、加速度 (Acc)、减速度 (Dec) 及速度类型 (Speed Pattern) 的设置。



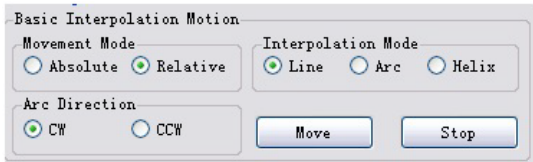
2.7.3 运动操作 (Motion Operation)

运动操作包括基本插补运动、Path 运动、切向跟随运动。



2.7.3.1 基本插补运动 (Basic Interpolation Motion)

基本插补操作包括：直线、圆弧、螺旋插补，如下图所示。在执行插补运动时，可选择运动模式：相对 (Relative)/ 绝对 (Absolute) 运动，执行圆弧插补时，可选择运行方向：顺时针 (CW)/ 逆时针 (CCW)。



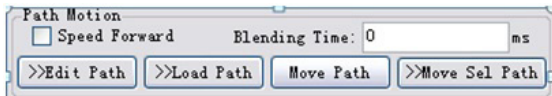
在执行插补运动时，可在表中配置圆心、端点等参数。

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
3-Axis	8000	0	0

如上图所示，1-axis 和 2-axis 添加到群组中且选择了线性插补模式，因此可写的编辑框为“Line End (PPU)”栏的“1-axis”和“2-axis”行，背景色为白色的区域。背景色为灰色的编辑框表示不可编辑。

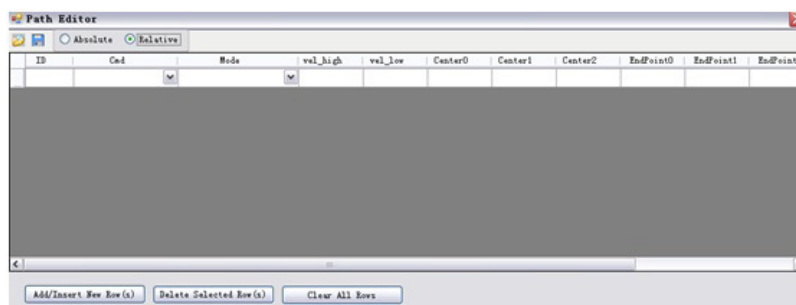
2.7.3.2 Path 运动 (Path Motion)

Path 运动相关操作如下图所示。可设置使用 / 禁用速度前瞻功能 (Speed Forward)，设置速度交接时间 (Blending Time)。用户可以通过点击“编辑 Path (Edit Path)”、“Load Path”、“运行 Path (Move Path)”、“运行选择的 Path (Move Sel Path)”按钮实现相应功能。在 Blending Time 中输入参数设置速度交接时间。



2.7.3.2.1 编辑 Path (Edit Path) 功能按钮

单击 [Edit Path]，将出现以下对话框：



其中，顶部的工具栏包括“Open File”、“Save File”和“Movement Mode”。

1. Open File ：打开相应路径下的 Path 文件，可以是一个二进制文件（.bin）或一个逗号分隔值文件（.CSV）。
2. Save File ：将编辑数据保存到路径文件，可以是一个二进制文件（.bin）或一个逗号分隔值文件（.CSV）。CSV 文件可以由 Excel 打开，方便事后查看修改。但是如果用户想要通过 [Load Path] 运行 Path，由于目前设备只支持通过 [Load Path] 导入 .bin 文件，因此必须将数据保存为 .bin 格式。
3. Movement mode: “Absolute”或“Relative”。如果用户选择“Absolute”，则 Cmd 栏中列出的命令将为绝对运动相关的命令；同样，如果用户选择“Relative”，则 Cmd 栏中列出的命令将为相对运动相关的命令。
4. Path 的编辑项目：包括命令 (Cmd)、运动模式 (Blending/No Blending)、运动速度 (vel_high)、初始速度 (vel_low)、圆心 (Center) 和终点 (EndPoint)。其中，默认的圆弧插补有三个轴 (Center0/Center1/Center2)，EndPoint 的数量由所选设备插补支持的轴的最大数量决定，如 PCI-1245，在直接插补运动中，EndPoint 的个数由 Device 的插补运动中最大支持的轴数决定，如 PCI-1245 最大支持 4 轴 Direct 运动，因此有 EndPoint0、EndPoint1、EndPoint2、EndPoint3。
5. Add/Insert New Row(s)：单击之后，会出现如下对话框，用户可编辑添加 / 插入的行的数量。



单击 [OK]，相应数量的行将会添加到所选行之后 / 插入到所选行中。

6. Delete Selected Row(s)：删除所选行。
7. Clear All Rows：清除所有行。

2.7.3.2.2 Load Path 功能按钮

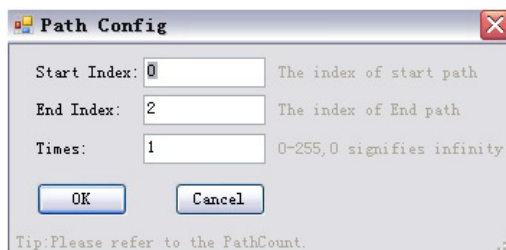
点击“Load Path”按钮，如果群组状态为“Ready”，用户可以从“Open File”对话框将所选 Path 文件（.bin 格式）导入到设备。

2.7.3.2.3 运行 Path(Move Path) 功能按钮

点击“Move Path”按钮，加载 Path 之后，如果编辑的 Path 正确，Path 将按照序列号一个接一个运行。

2.7.3.2.4 运行选定 Path(Move Sel Path) 功能按钮

用户可以从导入的 Path 中选择路径进行连续插补运动。加载 Path 之后，单击 [Move Sel Path] 弹出如下对话框：



1. Start Index: 选择 Path 的起始序列号。
2. End Index: 选择 Path 的终止序列号。
3. Times: 执行次数。值的范围为 0 到 255。如果设置为 0，表示这是一个无限循环，直到用户单击“Stop”终止循环。

2.7.3.2.5 观察 Path 状态 (Path Status)

运行 Path 过程中，用户可以通过 [Path Status] 观察运动状态，如下所示：



2.7.3.2.6 Path 曲线 (Path Plot)

点击 [Path Plot] 按钮，观察运动曲线。详见 2.7.4 节。

2.7.3.2.7 Path 状态 (Path Status)

点击 [Speed Chart] 按钮，观察速度曲线。

2.7.3.2.8 速度前瞻 (Speed Forward)

将使能 Group 的速度前瞻功能。有关详细信息，请参考编程指南属性列表中列出的群组中有关 CFG_GpSFEnable 的描述信息。

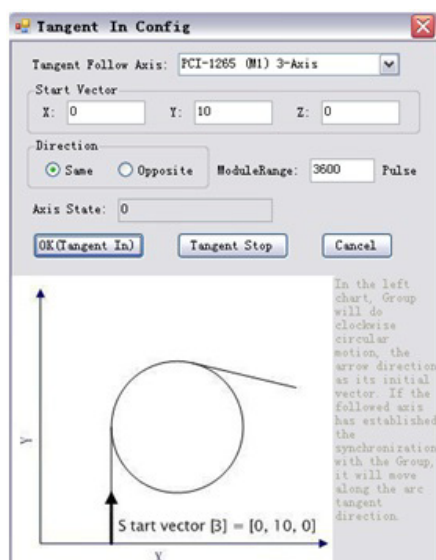
2.7.3.2.9 速度交接时间 (Blending Time)

有关详细信息，请参考编程指南属性列表中列出的群组中有关 CFG_GpBldTime 的描述信息。

2.7.3.3 切向跟随运动 (Tangent Motion)

2.7.3.3.1 切向跟随 (Tangent In) 功能按钮

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。

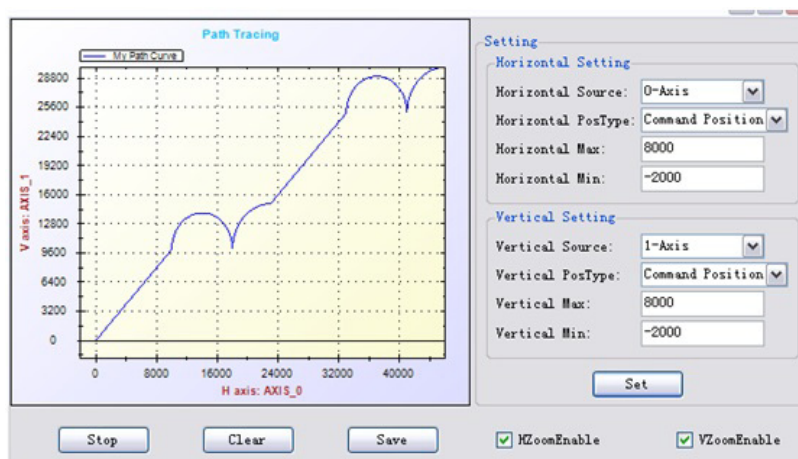
之后，如果群组进行插补运动，跟随轴将沿着插补运动的切向方向运动。如果切向跟随轴已经建立和群组的切向跟随同步运动，再次单击 [Tangent In]，对话框中的参数值将为配置值，用户可单击 [Tangent Stop] 解除同步关系。单击 [Cancel]，将不做任何修改同时关闭对话框。

2.7.3.3.2 停止切向跟随 (Tangent Stop) 功能按钮

单击 [Tangent Stop] 解除切向跟随轴和群组之间的同步关系。

2.7.4 运动曲线 (Path Plot)

显示群组的运动曲线。单击 [Path Plot]，将出现以下对话框：



2.7.4.1 设置 (Setting) 功能按钮

设置水平和垂直坐标。

1. Horizontal setting

- Horizontal Source: 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。
- Horizontal PosType: 水平位置类型，用户可选择命令或反馈位置。
- Horizontal Max: 水平最大坐标。
- Horizontal Min: 水平最小坐标。

2. Vertical setting: 与水平设置相同。

2.7.4.2 开始 (Start) 功能按钮

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”。

2.7.4.3 清除 (Clear) 功能按钮

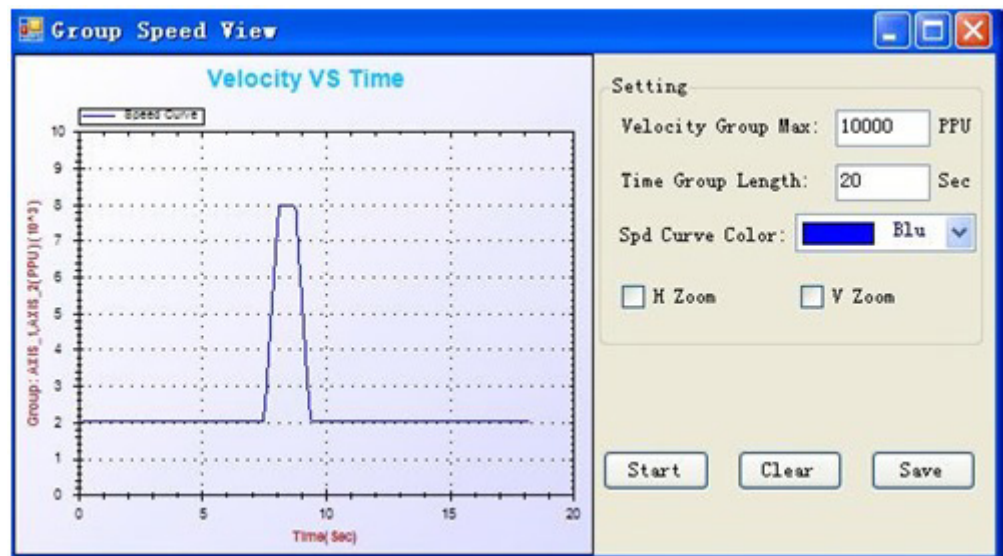
单击 [Clear] 将清除图形框中当前显示的曲线图。

2.7.4.4 保存 (Save) 功能按钮

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

2.7.5 速度曲线 (Speed Chart)

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



2.7.6 位置 (Position)

显示设备中所有轴的当前命令和反馈位置。单击 [Reset Counter] 将计数复位至 0。

Position	0-Axis	1-Axis	2-Axis	3-Axis	
Command	46000	30000	0	0	0
Feedback	0	0	0	0	0

Reset Error
Reset Counter

2.7.7 组的状态 (Group State)

Group State: 显示当前群组的状态。

Group State: Ready

2.7.8 最新错误状态 (Last Error)

Last Error Status: 显示上一个错误信息。

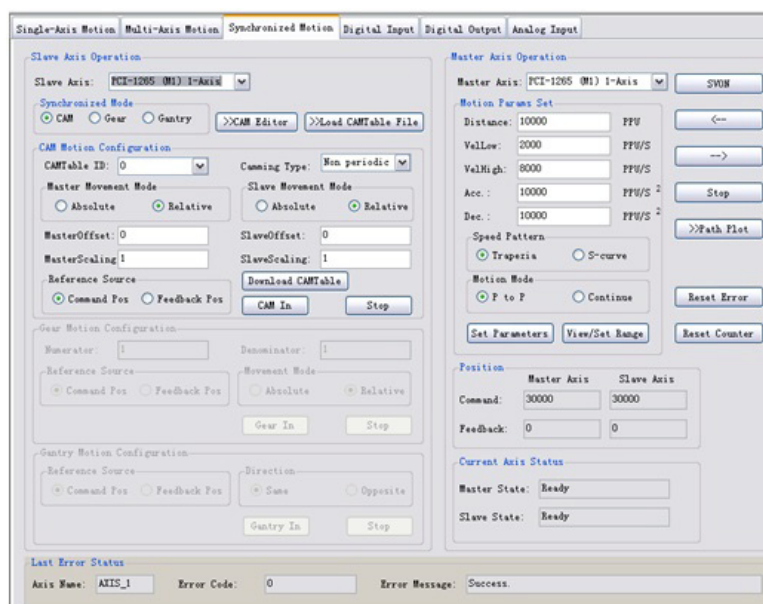
Last Error Status
Error Code: 0 Error Message: Success.

Error Code: 错误代码。

Error Message: 具体的错误信息。

2.8 Common Motion Utility 同步运动控制

同步运动操作主界面如下图所示：



2.8.1 从轴运动操作 (Slave Axis Operation)

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。

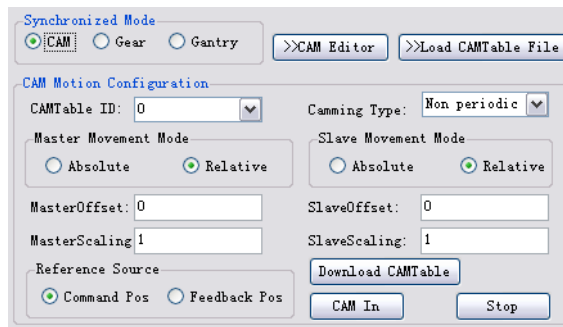
Slave Axis: PCI-1265 (M1) 2-Axis

同步模式 (Synchronized Mode) 包括龙门、电子齿轮、电子凸轮，如下图所示，可进行同步模式的选择。Utility 根据选择的同步模式界面做相应的变化，现分别介绍如下小节。

Synchronized Mode
☒ CAM ☐ Gear ☐ Gantry

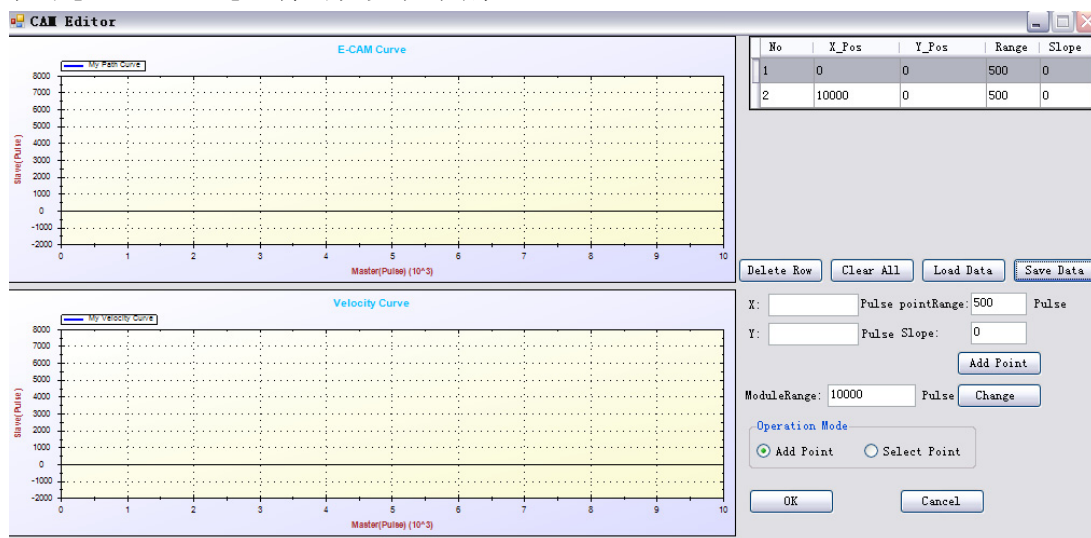
2.8.1.1 电子凸轮运动 (CAM)

当选择龙门运动时，将在如下界面中配置相关参数（默认为龙门操作模式）。



2.8.1.1.1 编辑电子凸轮 (CAM Editor) 功能按钮

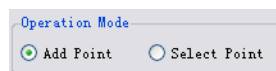
单击 [CAM Editor]，将出现以下对话框：



该编辑界面由以下几个部分组成：

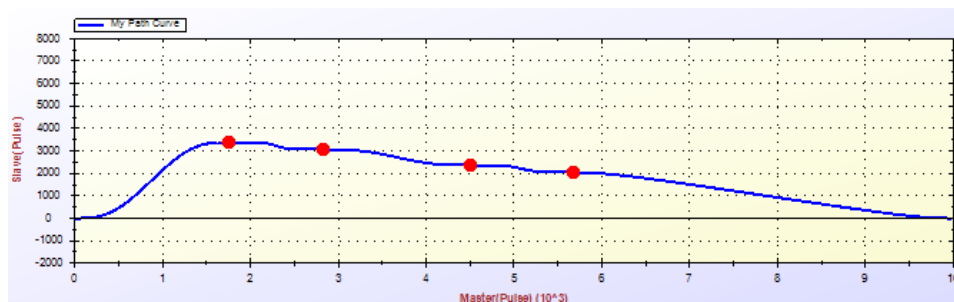
- 操作模式 (Operation Mode)
- CAM 表
- 速度曲线图形框
- CAM 曲线图形框

1. 操作模式 (Operation Mode)

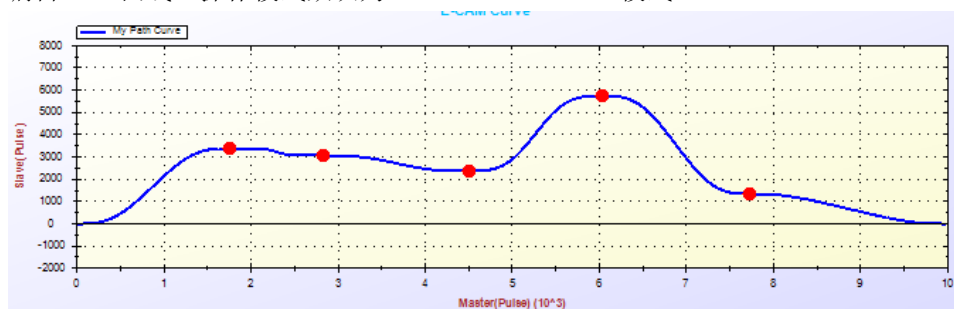


- a. **Add Point:** 用户可直接在 E-CAM 曲线上添加 CAM 点。如下图所示，将鼠标放在 E-CAM 曲线区域内，鼠标的形状变位十字形状。点击需要添加的 CAM 点，则该点变为红色实线圈，同时绘制凸轮曲线，如图中的蓝色曲线。无论何时用户添加了一个

点，CAM 曲线都将重新绘制。首次打开对话框或者还未编辑 CAM 曲线，操作模式默认为“Add Point”模式。



- b. **Select Point:** 用户可以选择相应的 CAM 点进行拖拉运动。同时，CAM 曲线也将随之改变。在这种操作模式中，鼠标的形状为箭头。当再次打开对话框或者已经编辑 CAM 曲线，操作模式默认为“Select Point”模式。



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，如下图所示。CAM 表中第一行的 X_Pos 和 Y_Pos，即首个 CAM 点，必须为 0，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。

No	X_Pos	Y_Pos	Range	Slope
1	0	0	500	0
3	1755.3	3387.511	500	0
4	3089.122	5048.888	500	0
5	4504.275	2365.125	500	0
6	5505.275	6582.467	500	0
7	7724.969	1342.739	500	0
2	10000	0	500	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 编程指南中的 Acm_DevDownloadCAMTable 函数的说明信息。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 编程指南中的 Acm_DevDownloadCAMTable 函数中对 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。如果用户想要改变编辑 CAM 点的 Slope，可直接在 CAM 表中进行修改。

注！



斜率值的范围为 -10 到 10。如果值小于 -10，则默认为 -10。如果值大于 10，则默认为 10。

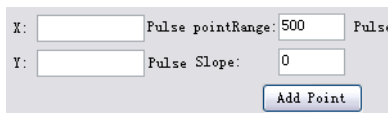
3. CAM 表操作



- 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。

4. Add Point 功能按钮

添加 CAM 点时，用户在如下图所示对应框中输入坐标点，然后单击 [Add Point] 即可。



5. 改变 Module Range

主轴的 Module Range 默认设置为 10000。如果用户想要编辑，可在 ModuleRange 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。

6. OK

单击 [OK] 保存 CAM 表。用户可通过 [Download CAMTable] 将 CAM 表将入硬件。

7. Cancel

单击 [Cancel] 放弃编辑。

2.8.1.1.2 导入 CAM 表文件 (Load CAMTable File) 功能按钮

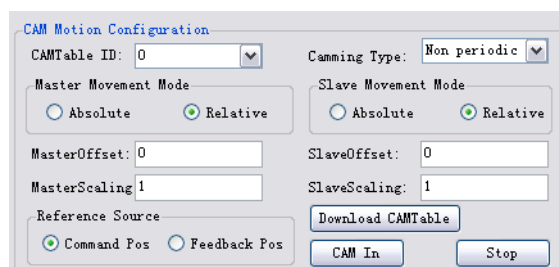
单击 [Load CAMTable File] 选择二进制文件 (.bin 文件，在 "CAM Editor" 中保存的文件)，CAM 表将导入硬件。保存之前，用户必须首先设置 "CAMTableID"。值为 0 或 1。完成该步后，在解除同步关系之前 CAMTableID 将不能修改。

2.8.1.1.3 下载 CAM 表 (Download CAMTable) 功能按钮

如果在 CAM Editor 中编辑过 CAM 表，用户可通过 [Download CAMTable] 将 CAM 表导入硬件。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。

2.8.1.1.4 CAM 运动参数配置 (CAM Motion Configuration)

配置 CAM 运动并建立 CAM 同步。



建立 CAM 同步之前，用户必须配置以下参数：

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:** 如果用户选择这种模式, 主轴的当前位置将被 CAM 曲线作为水平坐标的起始点。
- b. **Relative:** 如果用户选择这种模式, 主轴将以当前命令 / 实际位置作为起始点进行相对运动。

3. Slave Movement Mode

- a. **Absolute:** 如果用户选择这种模式, 从轴将从当前的命令 / 实际位置以设定的速度参数去追赶 CamTable 中对应的从轴的值。
- b. **Relative:** 如果用户选择这种模式, 从轴将以当前的命令 / 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。

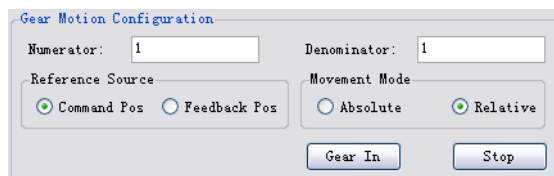
单击 [Stop] 解除同步关系。从轴的状态将恢复为 Ready。

2.8.1.1.5 CAM In

单击 [CAM In], 从轴将与主轴建立凸轮同步关系, 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 CAM 曲线和配置跟随主轴运动。

2.8.1.2 电子齿轮运动 (EGear)

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



2.8.1.2.1 电子齿轮运动参数配置 (EGear Motion Configuration)

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。
 - a. **Command Position:** 参考源为理论位置。
 - b. **Feedback Position:** 参考源为反馈位置。
4. **Movement Mode:**
 - a. **Absolute:** 若设置为该模式, 从轴将以设定的速度参数去追赶主轴的命令 / 实际位置, 直至两者一致。
 - b. **Relative:** 若设置为该模式, 从轴将与主轴保持初始时的速度差。

2.8.1.2.2 建立电子齿轮同步关系

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。

2.8.1.2.3 解除电子齿轮同步关系

单击 [Stop] 解除同步关系，从轴状态将为 “Ready”。

2.8.1.3 龙门运动 (Gantry)

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。



2.8.1.3.1 龙门运动参数配置 (EGear Motion Configuration)

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。
 - a. **Command Position:** 参考源为理论位置。
 - b. **Feedback Position:** 参考源为反馈位置。（轴卡不支持）
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。

2.8.1.3.2 建立龙门运动同步关系

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。

2.8.1.3.3 解除龙门运动同步关系

单击 [Stop] 解除同步关系，从轴状态将为 “Ready”。

2.8.2 主轴运动操作 (Master Axis Operation)

主轴运动操作如下图所示：

Master Axis Operation

Master Axis: PCI-1285 (M1) 1-Axis

SVON

Motion Params Set

Distance: 10000 PPU

VelLow: 2000 PPU/S

VelHigh: 8000 PPU/S

Acc.: 10000 PPU/S²

Dec.: 10000 PPU/S²

Speed Pattern

☒ Trapezia ☐ S-curve

Motion Mode

☒ P to P ☐ Continue

Set Parameters View/Set Range

Position

	Master Axis	Slave Axis
Command:	30000	45000
Feedback:	0	0

Current Axis Status

Master State: Ready

Slave State: Ready

2.8.2.1 主轴 (Master Axis)

从所选设备的所有轴中选择一个轴作为主轴。主轴和从轴不能为同一轴。默认的主轴为所选设备的 1 轴。

Master Axis: PCI-1285 (M1) 1-Axis

2.8.2.2 运动参数设置 (Motion Params Set)

设置步骤与 “Single-axis Motion”, “Motion Params Set” 相同。

Motion Params Set

Distance: 10000 PPU

VelLow: 2000 PPU/S

VelHigh: 8000 PPU/S

Acc.: 10000 PPU/S²

Dec.: 10000 PPU/S²

Speed Pattern

☒ Trapezia ☐ S-curve

Motion Mode

☒ P to P ☐ Continue

Set Parameters View/Set Range

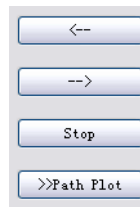
2.8.2.3 操作

2.8.2.3.1 SVON

单击 [SVON], 主轴和从轴的伺服将开启, 且按钮上的文字将变为 “SVOFF” ; 单击 [SVOFF], 轴的伺服将关闭, 且文字将变回 “SVON”。

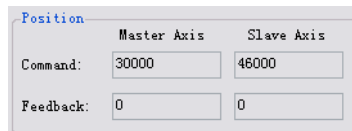
2.8.2.3.2 其它操作

其它操作, 请参考 “Single-axis Motion” -> “Move Test”。值得注意的是, 同步关系建立之后, 从轴将跟随主轴运动。用户可通过 [Path Plot] 查看运动曲线。



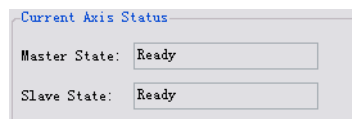
2.8.2.3.3 位置 (Position)

显示主轴和从轴的当前命令（理论）位置和反馈（实际）位置。单击 [Reset Counter] 将计数复位至 0。



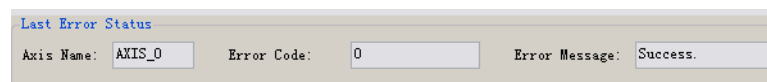
2.8.2.4 状态 (State)

用户可通过状态栏查看主轴和从轴的当前状态。有关详细信息，请参考通用 API 编程指南中的 Acm_AxGetState 函数中对 State 的说明信息。



2.8.3 最新错误状态 (Last Error Status)

Last Error Status: 显示上一个错误信息。



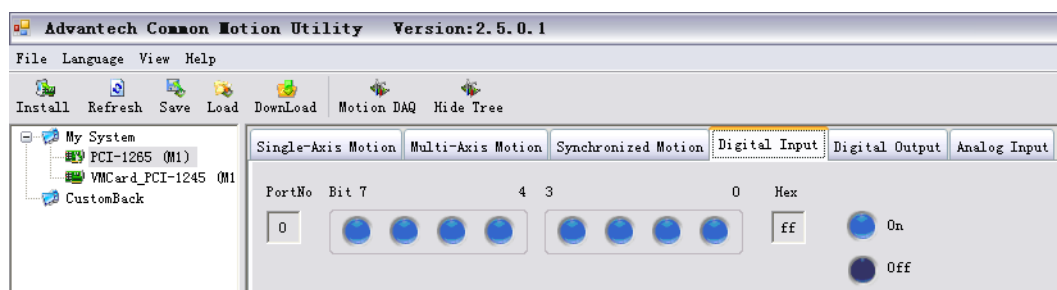
2.9 数字量输入 (Digital Input)

在 Utility 中，只有当板卡是 PCI-1265 时，会显示如下三个显示界面：数字量输入 (Digital Input)、数字量输出 Digital Output、模拟量输入 (Analog Input)。

2.9.1 数字量输入 (Digital Input)

主要显示设备的数字量输出端口的状态，以及 DO 上的相应 ON/OFF 操作。

PCI-1265 有 8 个 DO。

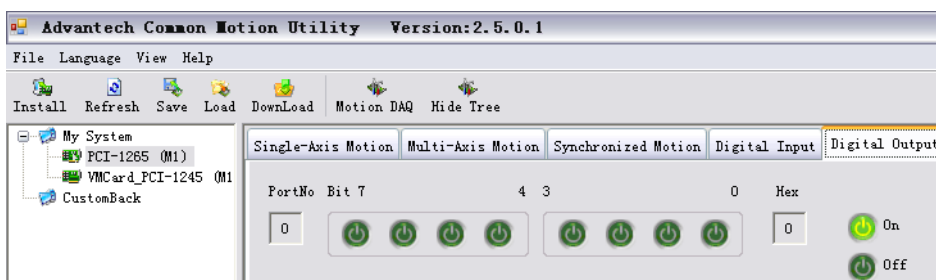




如上图所示，从右到左分别为数字量输入 DI0 ~ DI7。其中，●表示 DI 有效 (On)，bit 的值为 1；●表示 DI 无效 (Off)，bit 的值为 0。“Hex”表示 8 个 DI 组成的字节的十六进制值。

2.9.2 数字量输出 (Digital Output)

主要显示设备的数字量输出端口的状态，以及 DO 上的相应 ON/OFF 操作。

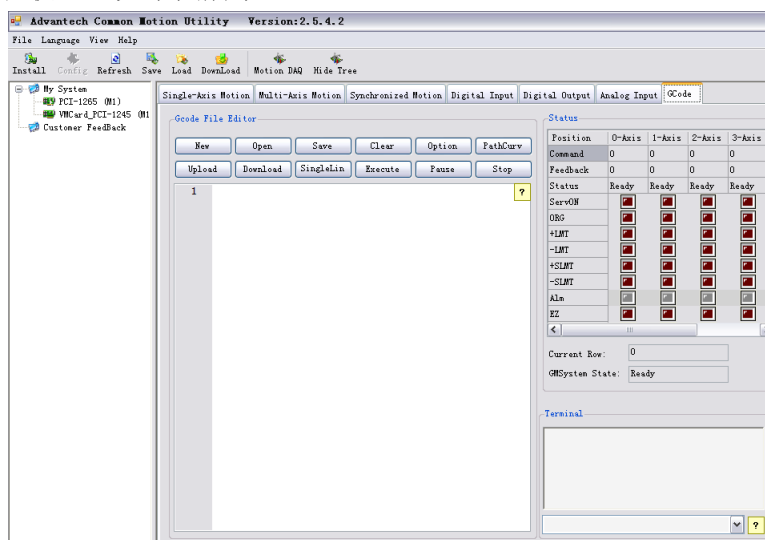
PCI-1265 有 8 个 DO。



如上图所示，从右到左分别为数字量输出 D00-D07。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。

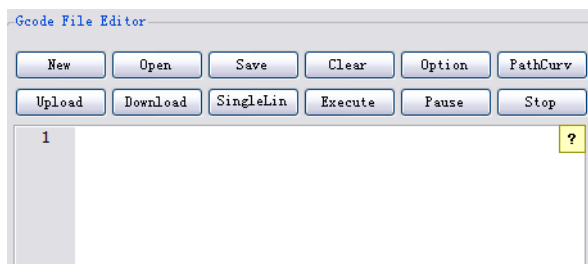
2.10 Common Motion Utility GCode 操作

Utility GCode 是利用 GCode Driver 建立的测试系统（关于 GCode 的详细说明，请参考 GCode 用户手册）。可以下载 GCode 文件，导入 DXF 文件，执行 GCode，并显示相关的运行轨迹和状态。如下图所示：

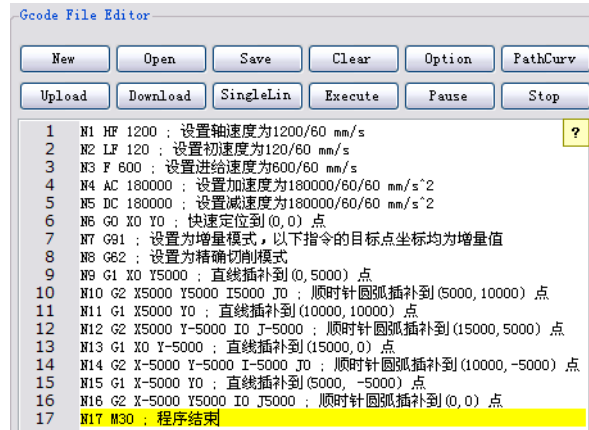


2.10.1 GCode 文本编辑器 (GCode File Editor)

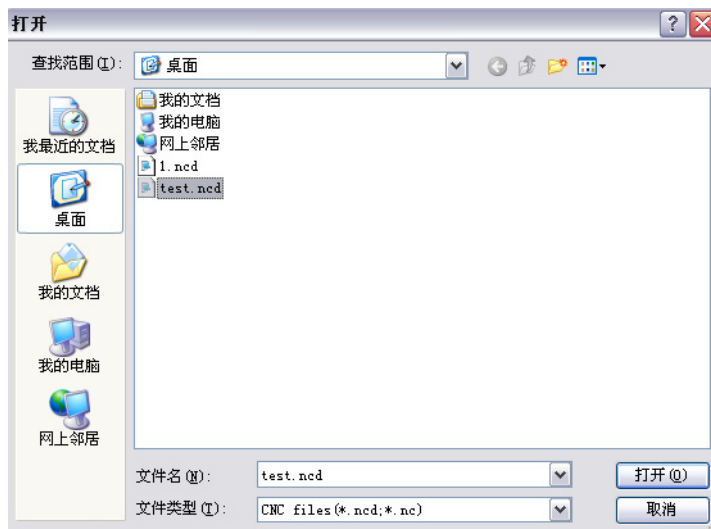
GCode 文本编辑器提供如下功能按钮：



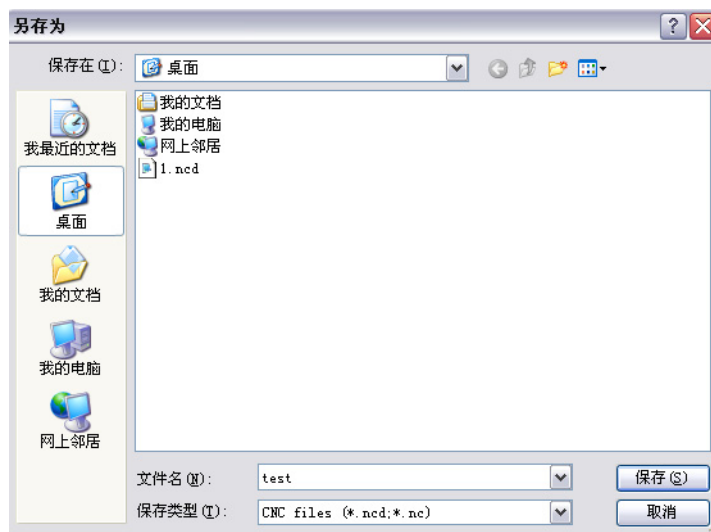
New 按钮: 该按钮表示新建一个 GCode 文档。点击该按钮后，即可在文本编译器中编写 G 指令。如在文本编辑器中输入以下内容



打开 (Open) 按钮: 点击该按钮后，将出现打开文件对话框，用户可选择文件，将文件中的代码夹杂到文本编辑器中。



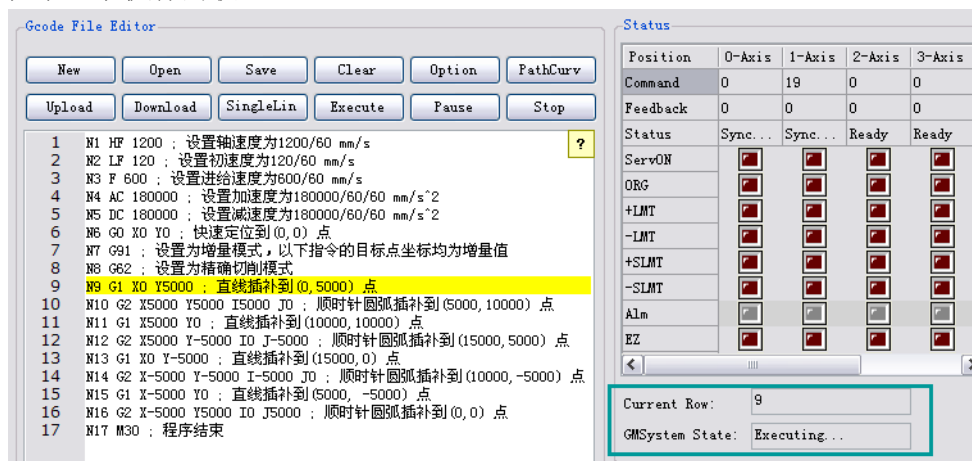
保存 (Save) 按钮: 点击该按钮，将文本编辑器的代码保存，如下文件名称为“test”，文件类型“.ncd”



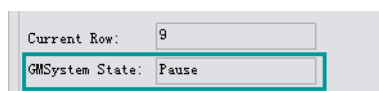
下载 (Download) 按钮: 该按钮将文本编辑器的代码下载到底层 GCode 解释器中。
 点击 Download, 当 Terminal 窗口中显示 “Download succeeded” 表示下载成功。



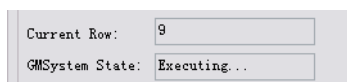
执行 (Execute) 按钮: 点击该按钮, 即可执行下载到解释器中的 G 指令。如下所示, 状态栏中显示执行的状态。



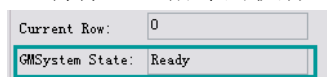
暂停 / 恢复 (Pause/Resum) 按钮: 该按钮初始状态为暂停 (Pause), 点击该按钮后, 即可暂停 G 指令的执行, 如下图 GMSysTem 的状态为 Pause



此时 Pause 按钮将变为 Resum, 点击 Resum, 将恢复 G 指令的执行

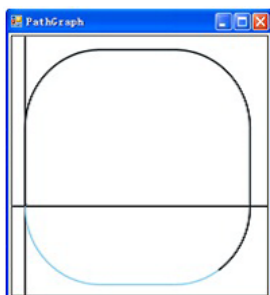


停止 (Stop) 按钮: 点击该按钮, 将停止 G 指令的执行

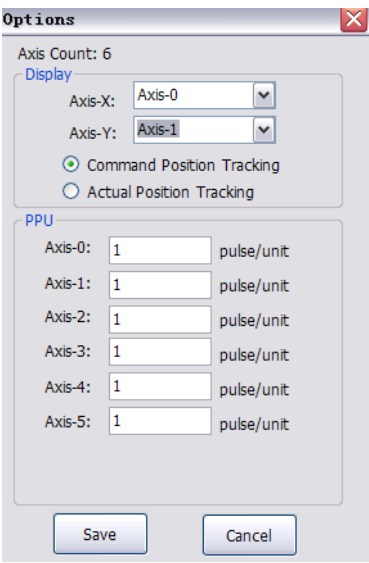


单步执行按钮 (SingleLin): 单步执行 G 指令

路径曲线 (PathCurv): 点击该按钮, 可查看 Path 运行曲线

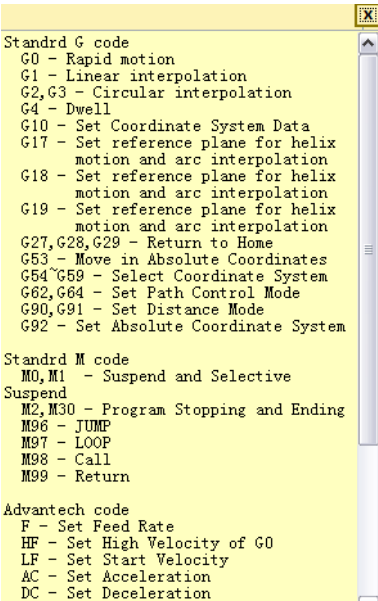


选择 (Option) 按钮： 点击该按钮将出现如下对话框，在该对话框中选择轴和位置选项，则在 Path 显示图形中显示选择轴的运行曲线。同时可以设置每个轴的 PPU，点击“Save”按钮，则设置生效。



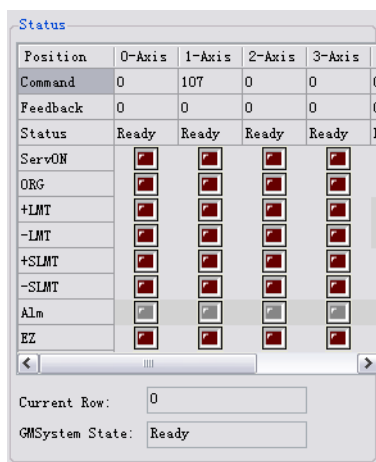
清除 (Clear) 按钮： 点击该按钮将清除 GCode 解释器中指令。

上传 (UpLoad) 按钮： 点击该按钮，将获取解释器中存储的 G 指令到文本编辑器中。文本编辑器中，点击黄色显示的 ”？” 按钮，可以查看所有 G 指令的默认值。



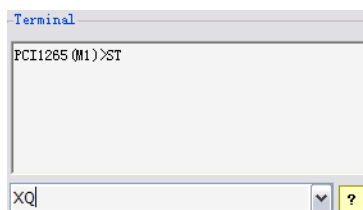
2. 10. 2GCode GCode 状态显示栏

在该状态栏中，可以查看轴的运行位置，包括命令位置和反馈位置，轴的状态，轴的 I/O 状态，包括 ServoOn、ORG、+LMT、-LMT、+SLMT、-SLMT、Alm、EZ 信号。在该状态栏中可查看正在执行的行和 GCode 解释器的状态。当 GCode 解释器的状态为 Ready 时，才能执行文本编辑器中的 G 指令。



2. 10. 3GCode 命令窗口

命令窗口如下图所示，例如用户发送 ” XQ” 指令，按回车键后，该指令将发送到 GCode 解释器中，向系统发送 ”XQ” 命令后，如果解释器的状态处于 Ready/Pause，解释器开始 / 继续执行 G 代码。



点击上图中的 ” ?” 按钮，将会显示自定义指令的默认值，如下：

```
GM System Command
XQ - Begin Program
ST - Stop Program
PS - Pause
RE - Resume
SL - Single Step
RS - Reset Error
CV - Change Velocity
GMCODE - Single GM Code
```

向系统发送 ”XQ” 命令后，如果解释器的状态处于 Ready/Pause，解释器开始 / 继续执行 G 代码。

向系统发送 ”ST” 命令后，解释器停止执行 G 代码，状态恢复为 Ready。

向系统发送 ”PS” 命令后，解释器暂停执行 G 代码，暂停状态可以通过 RE 指令恢复执行。

向系统发送 ”RE” 命令后，如果解释器处于暂停状态，将恢复执行 G 代码。

向系统发送 ”SL” 指令，每输入一个 ”SL”，程序往下执行一行。

当解释器的状态为 ”Error” 时，需发送 ”RS” 指令重置系统出现的错误，重置后解释器恢复为 Ready 状态后，才能执行其他操作。

向系统发送 ”CV”，动态改变运行速度。例如执行如下代码：

LF 18000

HF 48000

N1 G0 X100000 Y100000 Z100000

则执行 G0 时，轴的运行速度为 48000/60 mm/s，输入 CV 20000 后，轴的运行速度将变为 20000/60 mm/s。

向系统发送任意符合要求的 G CODE，系统会解释执行发送的指令。

第 3 章

关于研华运动控制卡接口函数库及其在编程工具中使用

3.1 本章简介

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

3.2 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。

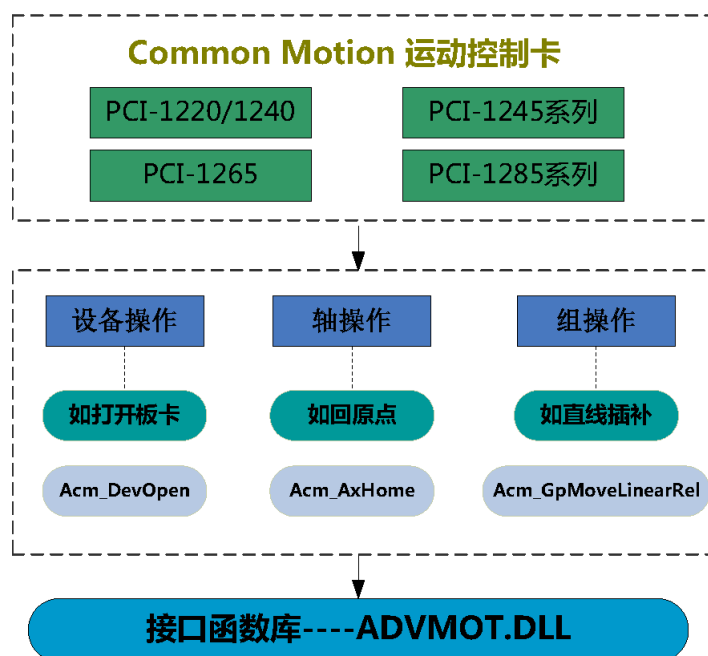


图 3.1： 不同设备对接口函数的调用

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。

3.2.1 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：

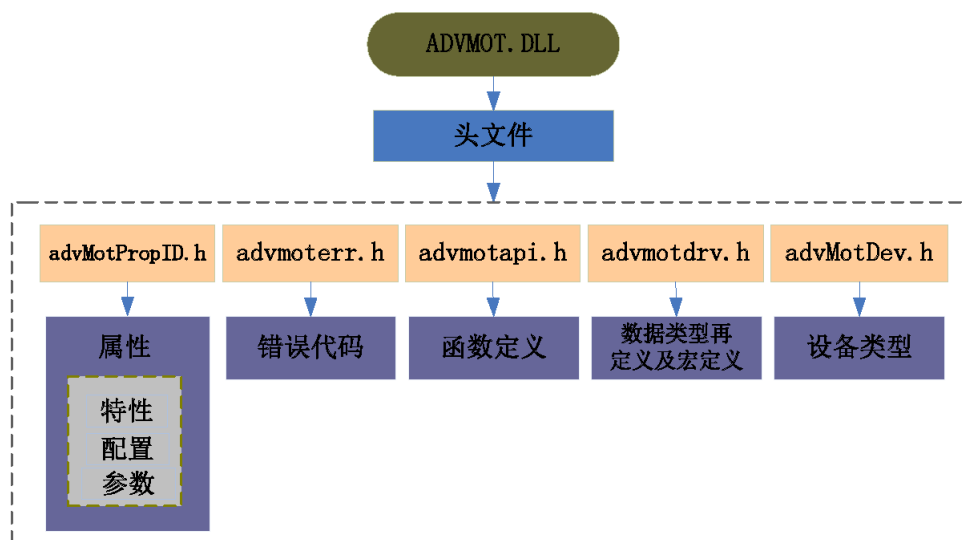


图 3.2：动态链接库所需头文件

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。

3.2.2 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。

3.2.2.1 功能函数命名规则

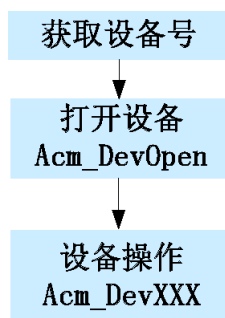
根据设备、轴、群组三种操作对象以及对 AIO、DIO 的操作，功能函数按如下规则命名：

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

3.2.2.2 功能函数简单使用说明

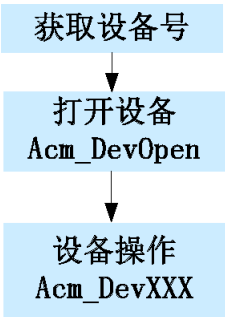
功能函数可在应用程序中直接调用。根据设备、轴、群组三种操作对象，简单说明对其操作流程。

对设备的基本操作如下：

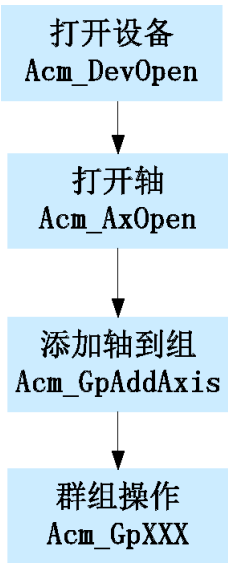


关于如何获取设备号，详见 4.4 章节。

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。

3.2.3 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 3.2.3.1 和 3.2.3.2 节。

3.2.3.1 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 3.1：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
配置	设备	CFG_DevXXX
	轴	CFG_AxXXXX
	群组	CFG_GpXXXX
	DIO/AIO	CFG_DaqXXX

表 3.1：属性的命名规则

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	

3.2.3.2 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 3.2：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 3.2.3 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：

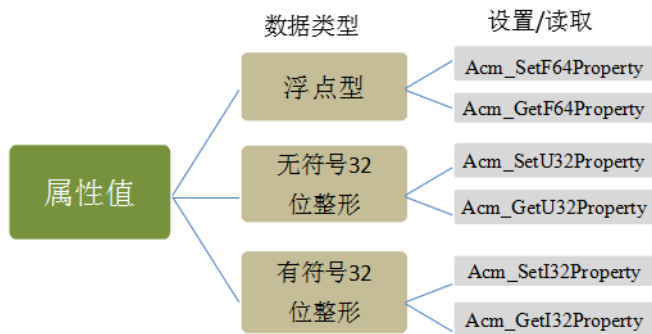


图 3.3：属性数据类型与函数的关系

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

3.2.4 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 2.4 章节。设备类型定义如下：

表 3.3：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-1265	Adv_PCI1265
PCI-1285	Adv_PCI1285
PCI-1285E	Adv_PCI1285E
PCI-1285V	Adv_PCI1285V
MIC-3245	Adv_MIC3245
MIC-3285	Adv_MIC3285

3.2.5 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。

3.2.6 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 3.4：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
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

表 3.4: 数据类型再定义

PF32	FLAOT *	指针指向 32 bit 浮点变量
PF64	DOUBLE *	指针指向 64 bit 浮点变量

3.2.7 板卡的初始操作

对于板卡的操作，如 3.2.2.2 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。

3.2.7.1 获取板卡设备号

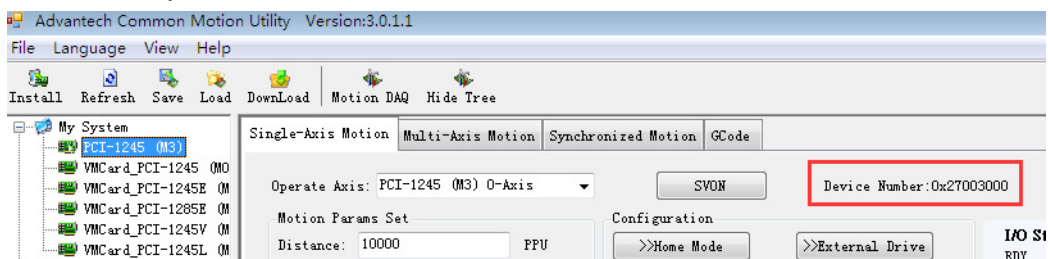
为了区分不同设备，将每张板卡进行编号，即设备号。每张板卡的设备号唯一，关于板卡编号的规则见 4.4 章节。

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 3.2.4 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_PCI1265, 1, &m_DevNum); // 获取设备号
Acm_DevOpen(m_DevNum, &m_Devhand); // 打开设备，获取设备 Handle
```

3.2.7.2 板卡的初始化流程图

板卡的初始化流程如下图所示：

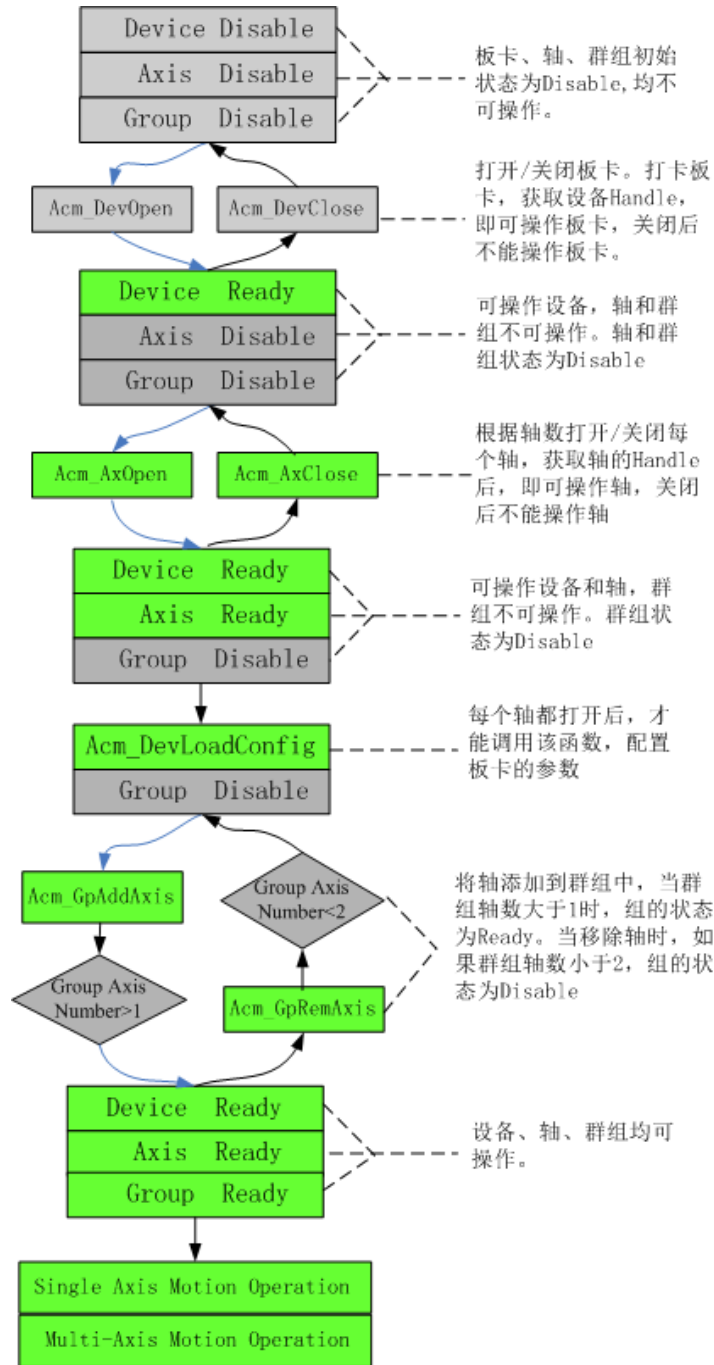


图 3.4：板卡初始化流程图

3.2.7.3 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3.2.8.1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[4]; // 轴的 handle
HAND    m_GpHand;     // 组的 handle
U32     m_DevNum;     // 设备号
U32     Ret;          // 函数返回值
Ret = Acm_GetDevNum(Adv_PCI1285, 15, &m_DevNum); // 获取板卡设备号
```

```
Ret = Acm_DevOpen(m_DevNum, &m_Devhand); // 打卡板卡, 获取设备 Handle
// 根据轴数, 将所有轴打卡, 对于 PCI1285 板卡, 轴数为 8
for(uint AxisNumber=0;AxisNumber<8;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 7.2.4 章节
// 执行多轴运动, 如直线运动, 请参考 7.2.4 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
```

3.3 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\Examples" 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:

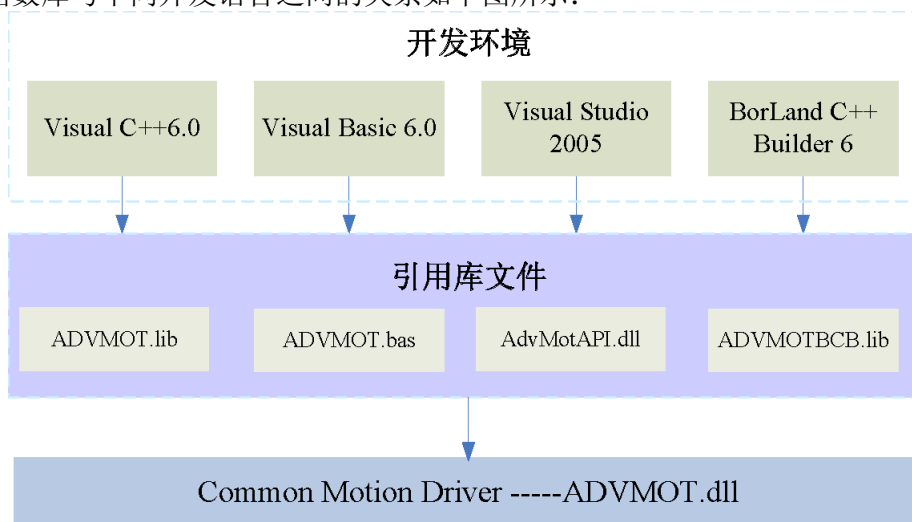


图 3.5: 属性数据类型与函数的关系

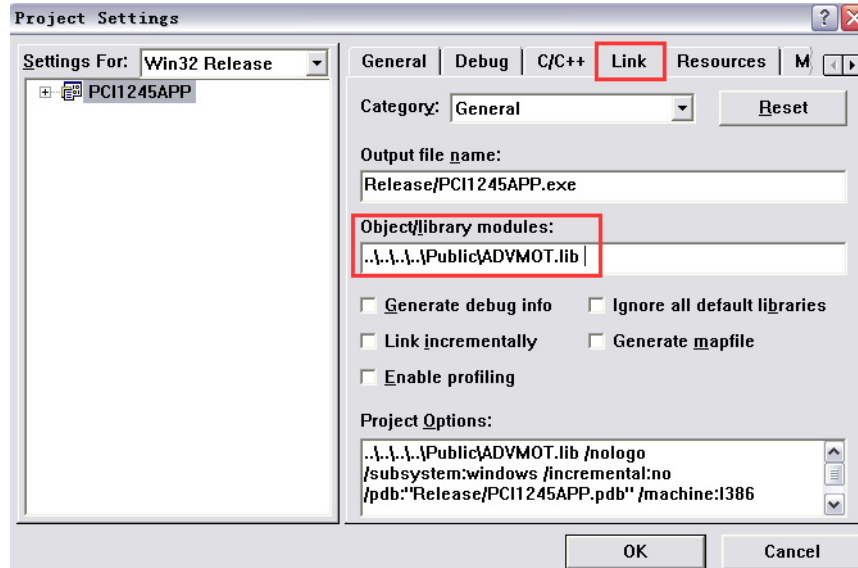
用于实现设备功能的所有 API 都可从 ADVMOT.DLL 获取。AdvMotAPI.dll、ADVMOT.bas、ADVMOT.lib、ADVMOTBCB.lib 都是基于 ADVMOT.dll 产生, 方便用户轻松开发应用程序。AdvMotAPI.dll 用于 C# 应用程序和 VB.net 应用程序。ADVMOT.bas 用于开发 VB 应用程序。ADVMOT.lib 用于开发 VC 应用程序。

ADVMOTBCB.lib 用于开发 BCB 应用程序。

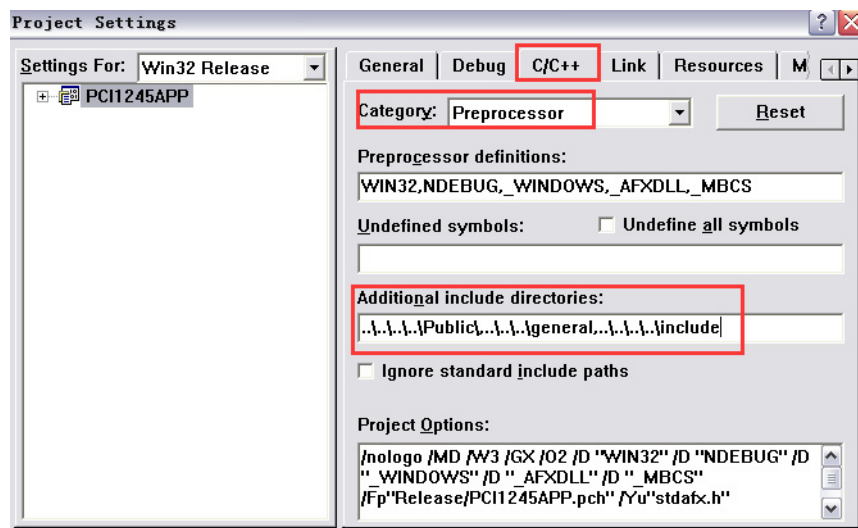
3.3.1 接口函数库在 Visual C++ 6.0 中的使用

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 ADVDMOT.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"
```

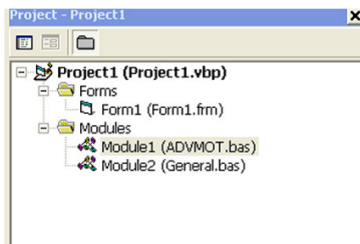
至此，用户就可以在 Visual C++ 中调用函数库中的任何函数，开始编写应用程序。

3.3.2 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。

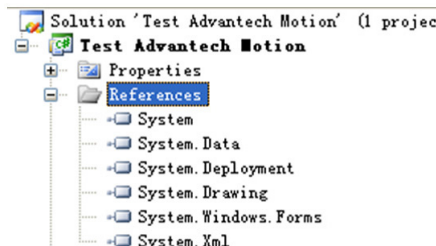


至此，即可调用接口函数库中的所有函数，可参考例程。

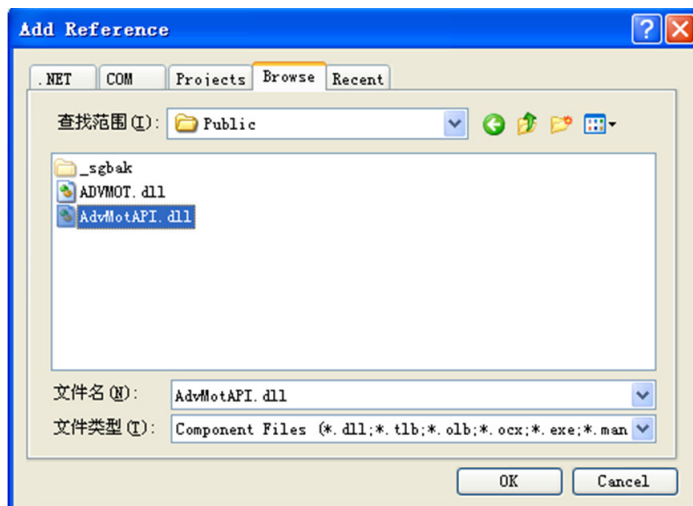
3.3.3 接口函数库在 Visual Studio 2005 中的使用

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

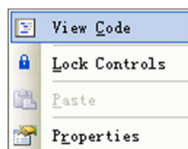
1. 启动 Visual Studio 2005, 按着“File”->“New”，选择建立 C# 工程
2. 添加相关 DLL 文件的引用
 - a. 环境右上角处的 [References]，如下图所示：



- 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：

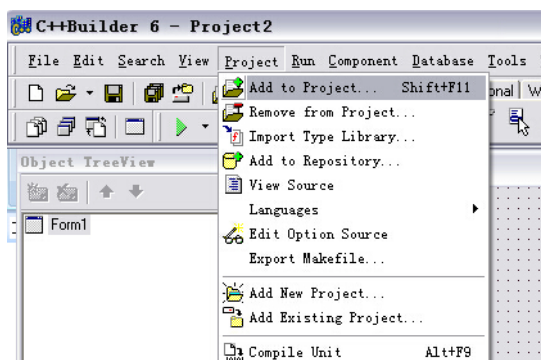
```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Text;
7 using System.Windows.Forms;
8 using Advantech.Motion;
```

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。

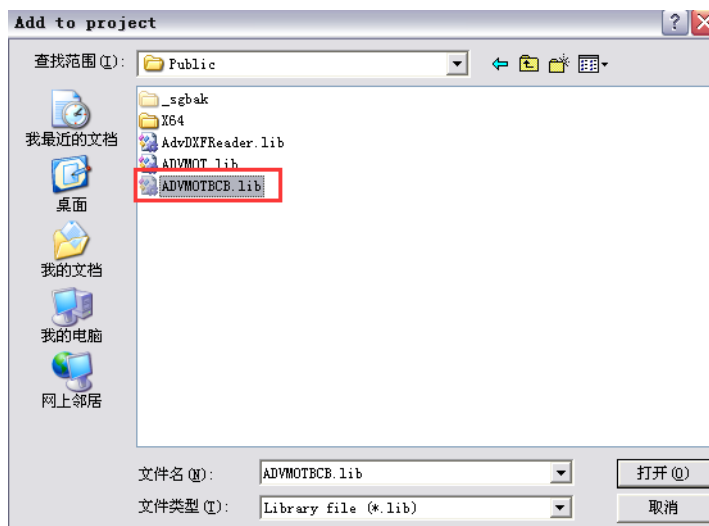
3.3.4 接口函数库在 Borland C++ Builder 中的使用

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。

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

3.4.1 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.

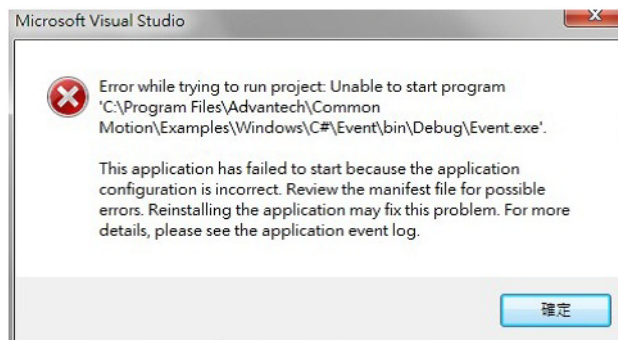
  <requestedExecutionLevel level="asInvoker" uiAccess="false" />
  <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
  <requestedExecutionLevel level="highestAvailable" uiAccess="false" />

  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>

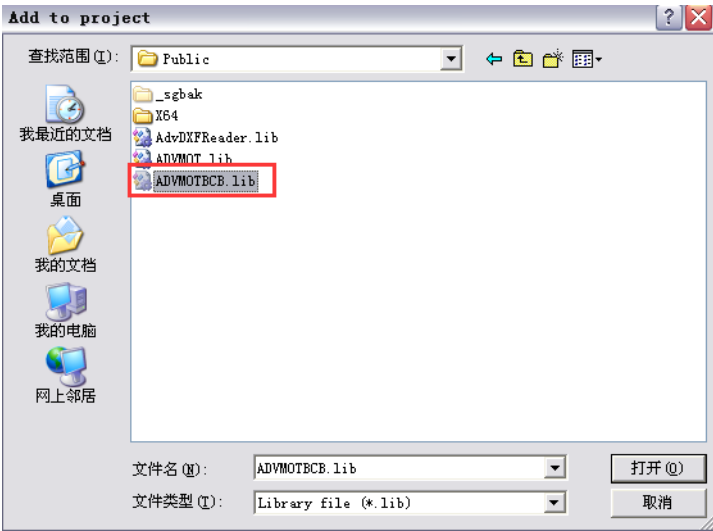
```

3.4.2 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。



3.5 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。

3.5.1 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-1245 板卡，其中 PCI-1245 (0) 只能控制该板卡的 4 个轴，PCI-1245 (1) 只能操作该板卡的 4 个轴，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。

卡片	PCI-1245 (BoardID = 0)				PCI-1245 (BoardID = 1)			
实体轴 ID	0	1	2	3	0	1	2	3

3.5.2 如何操作多快板卡

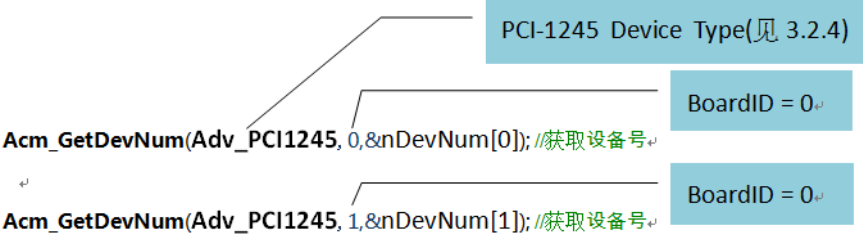
下面例子以一台电脑上插 2 块 PCI-1245 板卡，说明多块板卡操作过程：

板卡 1: PCI-1245 BoardID = 0

板卡 2: PCI-1245 BoardID = 1

1. 板卡的设备号 (Device Number):

DWORD nDevNum[2]; // 两块板卡的设备号



注！ 关于 Device Number 可参考手册 4.2 Device Number



2. 打开 2 块板卡和板卡上 4 轴（共 8 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-1245 板卡)
HAND    hAxis[8]; // 轴的 Handle(2 块 PCI-1245 板卡，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 4 章

通用运动 API 架构

4.1 本章简介

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。

4.2 通用运动架构简介

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。

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

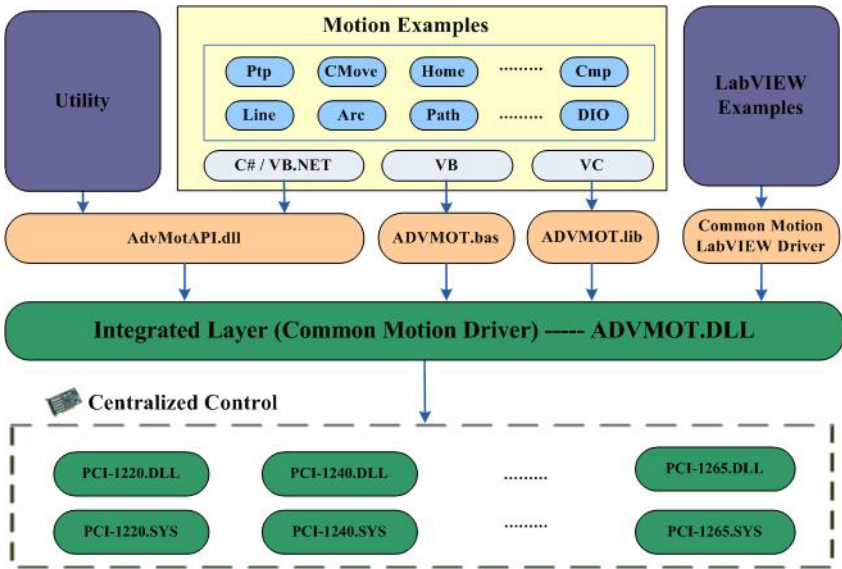


图 4.1: 运动控制卡通用运动架构

4.3 物件定义

通用运动架构中定义的缩写及含义如下表所示。

表 4.1: 缩写及其含义

缩写	全称	备注
PPU	每个单元脉冲	运动的一个虚拟单元 (Pulse per Unit)
Dev	设备	
Ax	轴	
Gp	群组	多个轴
Mas	主	基于通信机制的设备的主轴或主板卡
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	加速度	
Dec	减速度	
Emg	紧急	紧急停止
Sd	放慢	

表 4.1: 缩写及其含义

Info	信息	
Cmp	比较	
Inp	到位	
EZ	编码 Z	
EL	硬件限位	
Mel	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	

4.4 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的句柄, 该 API 需要传入参数的设备编号。设备编号由 32 位组成:

第四个字节	第三个字节	第二个字节高位	第二个低字节	第一个字节
主卡 / 设备类型 ID	主卡 / 设备板卡 ID		0	0

- 第四个字节
表示板卡类型 ID, 具体设备类型可参考 Public 文件夹下的头文件 AdvMotDev.h, 板卡对应的类型 ID 唯一。板卡类型定义见 3.3 节。
- 第三个字节和第二个字节的高位:
表示板卡的 ID, 该 Board ID 可通过硬件上的编码开关自由调整, 如果不进行板卡 boardID 的调整, 设备编号一般不会发生变化。
- 第二个低字节:
远程设备使用的主环路编号将 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

第四个字节	第三个字节	第二个字节高位	第二个低字节	第一个字节
板卡类型 ID	板卡 ID		0	0

比如, PCI-1245 的 BoardID 为 1, 设备编号 (十六进制) 则为:

27	001		0	0
----	-----	--	---	---

因此, 设备编号为 0x27001000。

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]; //两块板卡的设备号
```

```
nDevNum[0]= (Adv_PCI1245<<24) | (0 <<12);
```

```
nDevNum[1]= (Adv_PCI1245<<24) | (1 <<12);
```

PCI-1245 Device Type(见 3.3 节)

BoardID = 0

BoardID = 1

第 5 章

属性配置文件说明

5.1 本章简介

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。

5.2 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。

5.2.1 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)。

5.2.1.1 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
- 伺服报警清除信号 (ERC)
- 比较脉冲输出 (CMP)
- 凸轮输出 (CAMDO)
- 轴的通用 DO

命令脉冲输出

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	错误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	设置报警清除信号开启的时间长度
CFG_AxErcOffTime	ErcOffTime	设置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)

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

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	凸轮区间比较源
CFG_AxCamDOLogic	CamDoLogic	凸轮逻辑电平

轴的通用 DO 功能

属性	.CFG 文件缩写	说明
CFG_AxGenDoEnable	GenDoEnable	启用 / 禁用轴的通用 DO 功能

5.2.1.2 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
- 驱动报警数字量输入 (ALM)
- 伺服到位信号 (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

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_AxPulseInSource	PlsInSrc	反馈脉冲输入源（板卡暂不支持）
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	报警信号的有效逻辑电平
CFG_AxAlmReact	AlmReact	接收 ALARM 信号时的停止模式
CFG_AxALMFilterTime	ALMFilterTime	设定轴 ALM 信号滤波时间

到位信号输入 (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	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	硬件限位功能启用 / 禁用
CFG_AxEIReact	EIReact	EL 信号的反应模式
CFG_AxEILogic	EILogic	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	原点信号逻辑电平
CFG_AxEzLogic	OrgReact	回原点反应模式
CFG_AxORGFiterTime	ORGFiterTime	设定轴 ORG 信号滤波时间
PAR_AxHomeCrossDistance	HomeCrossDis	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	回原点停止模式
CFG_AxHomeOffsetDistance	HomeOffsetDistance	原点跨越距离
CFG_AxHomeOffsetVel	HomeOffsetVel	原点跨越速度

EZ 信号输入

属性	.CFG 文件缩写	说明
CFG_AxEzLogic	EzLogic	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	外部驱动源
CFG_AxExtSelEnable	ExtSelEnable	启用 / 禁用外部驱动
CFG_AxExtPulseNum	ExtPulseNum	驱动脉冲数
CFG_AxExtPulseInMode	ExtPulseInMode	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	锁存信号逻辑电平
CFG_AxLatchBufEnable	LatchBufEnable	启用 / 禁用连续锁存
CFG_AxLatchBufMinDistance	LatchBufMinDist	连续锁存最小距离
CFG_AxLatchBufEventNum	LatchBufEventNum	连续锁存事件数
CFG_AxLatchBufSource	LatchBufSource	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 触发时的停止模式
CFG_AxIN1StopReact	IN1StopReact	设定 IN1 触发停止功能的逻辑准位
CFG_AxIN2StopEnable	IN2StopEnable	启用 / 禁用 IN2 触发停止功能
CFG_AxIN2StopLogic	IN2StopLogic	设定 IN2 触发时的停止模式
CFG_AxIN2StopReact	IN2StopReact	设定 IN2 触发停止功能的逻辑准位
CFG_AxIN4StopEnable	IN4StopEnable	启用 / 禁用 IN4 触发停止功能
CFG_AxIN4StopLogic	IN4StopLogic	设定 IN4 触发时的停止模式
CFG_AxIN4StopReact	IN4StopReact	设定 IN4 触发停止功能的逻辑准位
CFG_AxIN5StopEnable	IN5StopEnable	启用 / 禁用 IN5 触发停止功能
CFG_AxIN5StopLogic	IN5StopLogic	设定 IN5 触发时的停止模式
CFG_AxIN5StopReact	IN5StopReact	设定 IN5 触发停止功能的逻辑准位

CFG_AxIN1FilterTime	IN1FilterTime	设定轴 IN1 信号滤波时间
CFG_AxIN2FilterTime	IN2FilterTime	设定轴 IN2 信号滤波时间
CFG_AxIN4FilterTime	IN4FilterTime	设定轴 IN4 信号滤波时间
CFG_AxIN5FilterTime	IN5FilterTime	设定轴 IN5 信号滤波时间

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
CFG_AxIN2FilterTime	IN2FilterTime	设定轴 IN2 信号滤波时间
CFG_AxIN4FilterTime	IN4FilterTime	设定轴 IN4 信号滤波时间
CFG_AxIN5FilterTime	IN5FilterTime	设定轴 IN5 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

5.2.2 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 Pos

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	PPUDenominator	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	设置负方向软件限位的反应模式
CFG_AxSwPelReact	SwPelReact	设置正方向软件限位的反应模式
CFG_AxSwMelValue	SwMelValue	设置负方向软件限位的值
CFG_AxSwPelValue	SwPelValue	设置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	设置补偿脉冲个数
CFG_AxBacklashVel	BacklashVel	设置背隙补偿的速度

轴的单圈脉冲个数 (ModuleRange)

属性	.CFG 文件缩写	说明
CFG_AxModuleRange	ModuleRange	设置当轴旋转 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	设置当前轴的同步起停模式

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	设置 Home 执行时的速度的曲线类型

JOG 速度参数

属性	.CFG 文件缩写	说明
CFG_AxJogVLTime	JogVLTime	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	设置 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 / 禁用软件正向极限误差功能
CFG_AxMelToleranceValue	MelToleranceValue	设置软件正向极限误差值
CFG_AxPelToleranceEnable	PelToleranceEnable	启用 / 禁用硬件正向极限误差功能
CFG_AxPelToleranceValue	PelToleranceValue	设置硬件正向极限误差值
CFG_AxSwMelToleranceEnable	SwMelToleranceEnable	启用 / 禁用软件负向极限误差功能
CFG_AxSwMelToleranceValue	SwMelToleranceValue	设置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	设置软件正向极限误差值

轴的减速

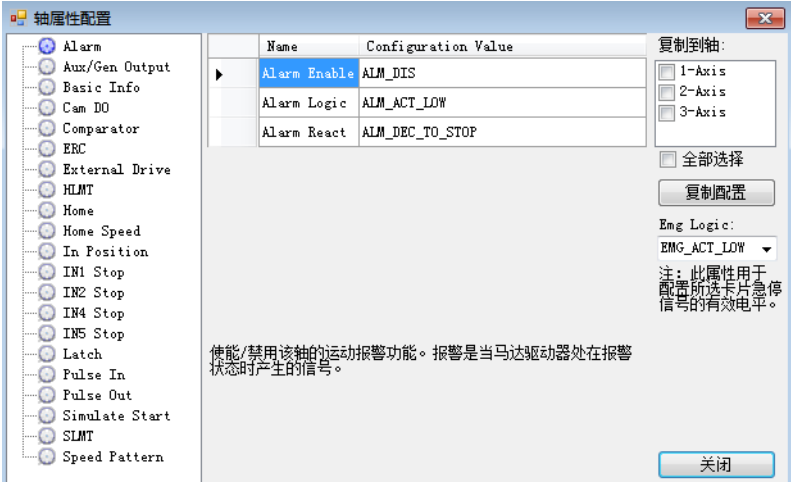
属性	.CFG 文件缩写	说明
CFG_AxKillDec	KillDec	DI Stop 的减速度

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

5.2.3 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：



5.3 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。

5.3.1 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

5.3.2 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。



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

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

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，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。5.3 节详细介绍了该配置文件的参数名称所对应的属性。

用户可直接修改该配置文件修改相应的属性值，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。

5.4 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 5.3 节中列出的属性修改相关的配置信息。

5.5 控制器配置初始化状态

5.5.1 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	凸轮区间下边界范围	10000	CFG_AxCamDOLoLimit
	凸轮区间上边界范围	10000	CFG_AxCamDOHiLimit
	凸轮区间比较源	理论位置	CFG_AxCamDOCmpSrc
	凸轮逻辑电平	高电平	CFG_AxCamDOLogic
轴报警清除信号输出	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_AxPulseInMode
	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	报警功能启用	禁用	CFG_AxAlmEnable
	报警信号电平	高电平	CFG_AxAlmLogic
	报警后停止模式	减速停止	CFG_AxAlmReact
	报警滤波时间	5us	CFG_AxALMFilterTime
到位信号输入	到位功能启用	禁用	CFG_AxInpEnable
	到位信号逻辑电平	高电平	CFG_AxInpLogic
正负限位数字量输入	硬件限位功能启用 / 禁用	启用	CFG_AxEIEnable
	EL 信号的响应模式	立即停止	CFG_AxEIReact
	硬件限位信号逻辑电平	低电平	CFG_AxEILogic
	正限位滤波时间	5us	CFG_AxLMTFilterTime
	负限位滤波时间	5us	CFG_AxLMTNFilterTime

原点信号数字量输入	原点信号逻辑电平	低电平	CFG_AxOrgLogic
	回原点反应模式	减速停止	CFG_AxEzLogic
	Home 运动中 search 的距离	10000	PAR_AxHomeCrossDistance
	启用 / 禁用复位逻辑计数器	启用	CFG_AxHomeResetEnable
	回原点停止模式	减速停止	PAR_AxHomeExSwitchMode
EZ 信号输入	EZ 信号的有效逻辑电平	低电平	CFG_AxEzLogic
外部驱动输入	外部驱动源	0 轴	CFG_AxExtMasterSrc
	启用 / 禁用外部驱动	禁用	CFG_AxExtSelEnable
	驱动脉冲数	1	CFG_AxExtPulseNum
	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
	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
	连续锁存最小距离	1000	CFG_AxLatchBufMinDistance
	连续锁存事件数	128	CFG_AxLatchBufEventNum
	连续锁存数据源	理论位置	CFG_AxLatchBufSource
	连续锁存轴	0 轴	CFG_AxLatchBufAxisID
	连续锁存触发沿		CFG_AxLatchBufEdge
减速信号输入	启用 / 禁用源 SD 信号	禁用	CFG_AxSdEnable
	设置 SD 信号逻辑电平	低电平	CFG_AxSdLogic
	设置 SD 信号的反应模式	仅减速	CFG_AxSdReact
	设置 SD 信号 Latch 控制	不锁存	CFG_AxSdLatch
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 触发停止功能的逻辑准位	高电平	CFG_AxIN1StopReact
	启用 / 禁用 IN2 触发停止功能	禁用	CFG_AxIN2StopEnable
	设定 IN2 触发时的停止模式	减速停止	CFG_AxIN2StopLogic
	设定 IN2 触发停止功能的逻辑准位	高电平	CFG_AxIN2StopReact
	启用 / 禁用 IN4 触发停止功能	禁用	CFG_AxIN4StopEnable
	设定 IN4 触发时的停止模式	减速停止	CFG_AxIN4StopLogic
	设定 IN4 触发停止功能的逻辑准位	高电平	CFG_AxIN4StopReact
	启用 / 禁用 IN5 触发停止功能	禁用	CFG_AxIN5StopEnable
	设定 IN5 触发时的停止模式	减速停止	CFG_AxIN5StopLogic
	设定 IN5 触发停止功能的逻辑准位	高电平	CFG_AxIN5StopReact
	设定轴 IN1 信号滤波时间	5us	CFG_AxIN1FilterTime
	设定轴 IN2 信号滤波时间	5us	CFG_AxIN2FilterTime
	设定轴 IN4 信号滤波时间	5us	CFG_AxIN4FilterTime
	设定轴 IN5 信号滤波时间	5us	CFG_AxIN5FilterTime

5.5.2 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	初速度	2000	PAR_AxVelLow
	运行速度	8000	PAR_AxVelHigh
	加速度	10000	PAR_AxAcc
	减速度	10000	PAR_AxDec
	最大运行速度	5000000	CFG_AxMaxVel
	最大减速度	5000000	CFG_AxMaxDec
	最大加速度	5000000	CFG_AxMaxAcc
	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	负方向软件限位功能	启用	CFG_AxSwMelEnable
	正方向软件限位功能	启用	CFG_AxSwPelEnable
	负方向软件限位反应模式	减速停止	CFG_AxSwMelReact
	正方向软件限位的反应模式	减速停止	CFG_AxSwPelReact
	负方向软件限位的值	-10000000	CFG_AxSwMelValue
	正方向软件限位的值	10000000	CFG_AxSwPelValue
轴的背隙补偿	启用 / 禁用背隙补偿	禁用	CFG_AxBacklashEnable
	设置补偿脉冲个数	10	CFG_AxBacklashPulses
	设置背隙补偿的速度	1000	CFG_AxBacklashVel
轴单圈脉冲数	单圈脉冲数范围	0	CFG_AxModuleRange
同步启停	启停模式	禁用	CFG_AxSimStartSource
回原点速度参数	回原点初速度	2000	PAR_AxHomeVelLow
	回原点运行速度	8000	PAR_AxHomeVelHigh
	回原点加速度	10000	PAR_AxHomeAcc
	回原点减速度	10000	PAR_AxHomeDec
	回原点曲线类型	T 型	PAR_AxHomeJerk
Jog 运动速度参数	Jog 运动低速运行时间	5000	CFG_AxJogVLTime
	Jog 运动初速度	2000	CFG_AxJogVelLow
	Jog 运动运行速度	8000	CFG_AxJogVelHigh
	Jog 运动加速度	10000	CFG_AxJogAcc
	Jog 运动减速度	10000	CFG_AxJogDec
	Jog 运动曲线类型	T 型	CFG_AxJogJerk
容差	启用负向容差	禁用	CFG_AxMelToleranceEnable
	负向容差值	5000	CFG_AxMelToleranceValue
	启用正向容差	禁用	CFG_AxPelToleranceEnable
	正向容差值	5000	CFG_AxPelToleranceValue
	启用软件负向容差功能	禁用	CFG_AxSwMelToleranceEnable
	软件负向容差值	5000	CFG_AxSwMelToleranceValue
	启用软件正向容差功能	禁用	CFG_AxSwPelToleranceEnable
	软件正向容差值	5000	CFG_AxSwPelToleranceValue
轴的减速度	DI Stop 的减速度	100000	CFG_AxKillDec

差值报警	设置差值数	0(不报警)	CFG_AxMaxErrorCnt
------	-------	--------	-------------------

5.6 伺服操作

打开 / 关闭电机函数如下表所示

函数	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。如下例程：

```
U32    Result;
HAND   m_Axishand[4]; // 轴的 handle
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
```

5.7 脉冲输入输出方式

5.7.1 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
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_AxPulseInMaxFreq	设置 / 获取频率中编码最大脉冲
CFG_AxPulseOutMode	设置 / 获取命令脉冲输出模式
CFG_AxPulseOutReverse	启用 / 禁用 Pulse Out 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。

5.8 浮點 PPU

5.8.1 浮点 PPU 相关属性

浮点 PPU 相关属性如下表所示：

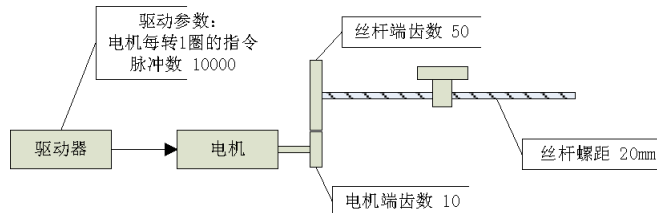
属性	说明
CFG_AxPPU	设定 PPU 的分子
CFG_AxPPUDenominator	设定 PPU 的分母

5.8.2 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

如下图所示机械结构：



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：

丝杆螺距为 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。

5.8.3 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。

5.8.4 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);
```


5.9 系统密码保护

5.9.1 密码保护函数

密码保护函数如下表所示：

函数	说明
Acm_DevReadEEPROM_Ex	读取私有数据
Acm_DevWriteEEPROM_Ex	写入私有数据及密码

5.9.2 密码保护重点说明

板卡的 EEPROM 编程存储器中可存储 8 组 (128bytes, 每组 16bytes) 密码保护资料, 每组由密码与资料组成: 密码 (8bytes) + 资料 (8bytes)。用户可通过密码保护功能保护产品版权。

每组资料的输入密码错误次数最多 3 次

1、超过 3 次无法读取, 重开机后错误次数归零

2、出厂预设密码为 0

当调用 Acm_DevWriteEEPROM_Ex 函数写入私有资料时, 同时需设定好密码, 板卡将不对已存在 EEPROM 上的密码, 直接更新。

当调用 Acm_DevReadEEPROM_Ex 函数读出私有资料时, 需输入设定的密码, 当密码输入正确后, 该函数将返回私有资料数据。

5.9.3 例程

当打开板卡后, 即可调用上述 API 实现数据保护功能

```
U32    Ret; // 函数返回值
U32    passwrд[2], wData[2];
U32    userwrд[2], rData[2];
--- 初始化过程见 3.2 节 ---
// 写入私有数据和密码
passwrд[0] = 123; // 密码数据
passwrд[1] = 123; // 密码数据
wData[0] = 24; // 资料数据
wData[1] = 30; // 资料数据
Ret = Acm_DevWriteEEPROM_Ex(m_Devhand, 0, passwrд, 2, wData, 1);
userwrд[0] = 123; // 密码数据
userwrд[1] = 123; // 密码数据
Ret = Acm_DevReadEEPROM_Ex(m_Devhand, 0, userwrд, 2, rData, 1);
具体使用步骤可参考 Data Protect 例程。
```


第 6 章

运动状态

6.1 本章简介

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。

6.2 轴状态

6.2.1 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

6.2.2 重点说明

6.2.2.1 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 6.1：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。

6.2.2.2 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 6.2：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

6.2.3 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 handle
U32     Ret;  // 函数返回值
U16     State;
CString strTemp;
--- 初始化过程见 3.2 节 ---
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;
    }
}

```

6.3 轴速度

6.3.1 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

6.3.2 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。

6.3.3 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：

```
HANDLE    m_Axishand; //0 轴的 handle
```

```
U32       Ret;
```

--- 初始化过程见 3.2 节 ---

// 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
```

6.4 编码器

6.4.1 编码器相关函数

用户可以从板卡的位置寄存器中读取编码器纪录当下轴的实际位置，相关函数如下表所示：

函数	说明	页码
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_AxGetState	获取轴的当前状态	

6.4.2 编码器重点说明

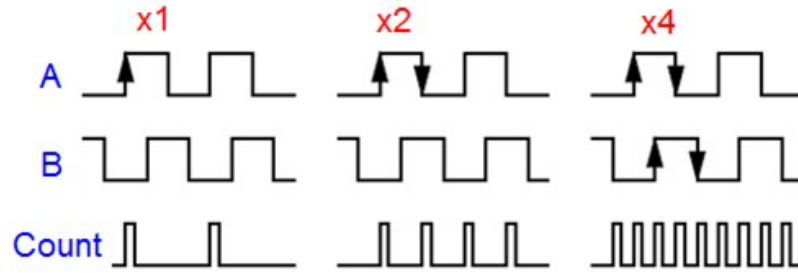
编码器是在玻璃、金属、塑料等材料上，由透光和不透光的区域相间组成，利用光学或磁性或是机械接点的方式感测位置，并将位置转换为电子讯号后输出，一般都应用于位置控制系统的检测组件。

依输出信号的类型，可简单区分为增量型编码器及绝对型编码器两种：

1. 绝对编码器是直接输出数字量的传感器，透过转盘上透光与不透光的区间编码，在转轴的任意位置可读出与位置相对应的数字码，没有累积误差且当系统断电后，仍可保留位置信息。
2. 增量式编码器是利用光电转换原理输出脉冲 A、B 和 Z 相；A、B 两组脉冲相位差 90°，由相位的领先及落后判断出旋转方向，而 Z 相为每旋转一周，会产生一个脉冲，用于定位。

优点是原理构造简单，缺点是无法输出马达转动时的绝对位置。增量型编码器的讯号是周期性的，讯号本身无法提供明确的位置信息，但若以某位置为标准，持续的相对讯号计数，可得到明确位置信息。这也是为何每次现场设备都必须回 Home 的原因。

下图是标准 A、B 相的 Encoder



由于 A、B 两相相差 90 度，可通过比较 A 相在前还是 B 相在前，以判别编码器的正转与反转，两组脉冲讯号可产生最高为 4 个正负缘计数（4 倍频）。通过 Z 相脉冲，可获得编码器的零相参考位。

6.4.3 例程

该实现打开板卡和轴后，获取轴的当前状态

```
HAND    ;//0 轴的 handle
U32     Ret; // 函数返回值
U16     State;
CString strTemp;
--- 初始化过程见 3.2 节 ---
Ret=Acm_AxGetState(m_Axishand, &State);
if (Ret ==SUCCESS)
{switch (State)
{
    case0:
        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;
case8:
                strTemp.Format("STA_AX_EXT_JOG ");
                break;
case9:
                strTemp.Format("STA_AX_EXT_MPG ");
                break;
            default:
                break;
        }
    }
    // 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
```

6.5 轴的运动 I/O 状态

6.5.1 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

6.5.2 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 DO

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。

6.5.3 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;
--- 初始化过程见 3.2 节 ---
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 }
}
```

6.6 群组状态

6.6.1 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

6.6.2 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 6.3：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 2 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。

6.6.3 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
--- 初始化过程见 3.2 节 ---
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;
    }
}
```

6.7 群组速度

6.7.1 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

6.7.2 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。

6.7.3 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
--- 初始化过程见 3.2 节 ---
// 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 7 章

运动功能

7.1 本章简介

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。

7.2 单轴运动模式

7.2.1 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
- 手轮 (MPG) 运动
- 原点复归

7.2.2 点到点运动

7.2.2.1 点到点运动相关函数与属性

点到点运动相关函数如下表 7.1 所示，该表中的函数可在应用程序中直接调用。

表 7.1：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.2 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 7.2: 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

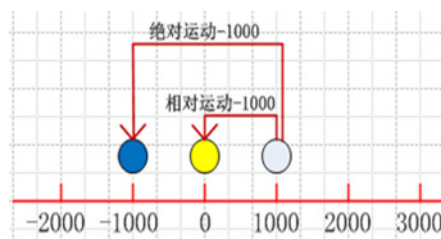
7.2.2.2 重点说明

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（7.2.4 和 7.2.5 章节）。

7.2.2.3 点到点运动流程图

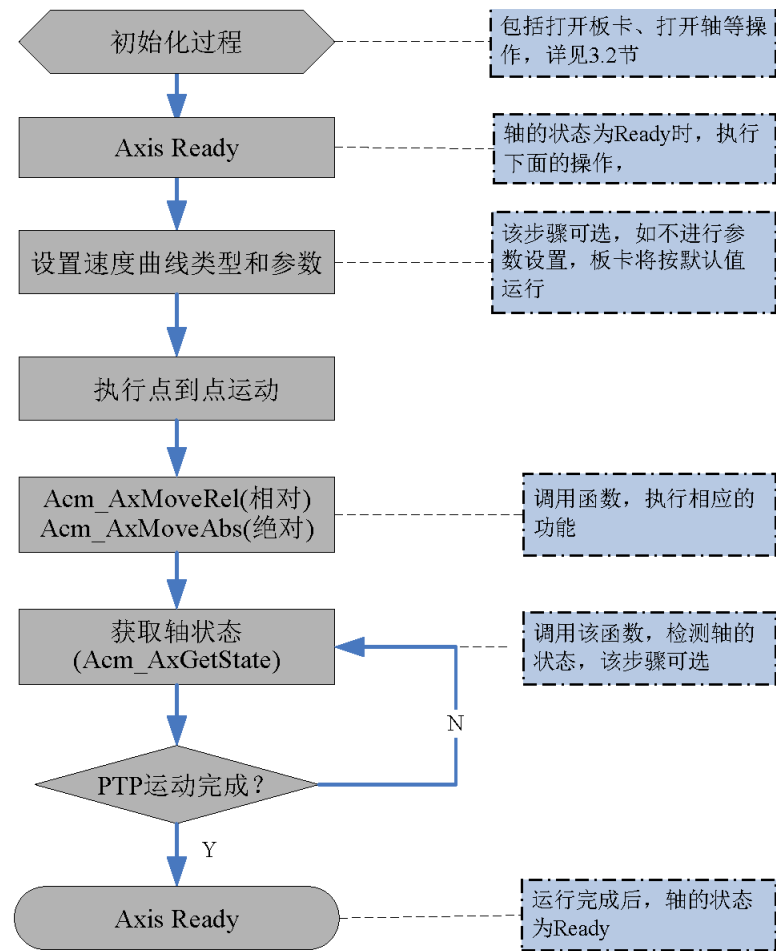


图 7.1： 点到点运动流程图

7.2.2.4 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-1245 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[4]; // 轴的 handle
U32     Ret; // 函数返回值
--- 初始化过程见 3.2.7 节 ---
// 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。

7.2.3 连续运动

7.2.3.1 连续运动相关函数与属性

连续运动相关函数如下表 7.3 所示。下表中的函数可在应用程序中直接调用。

表 7.3：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

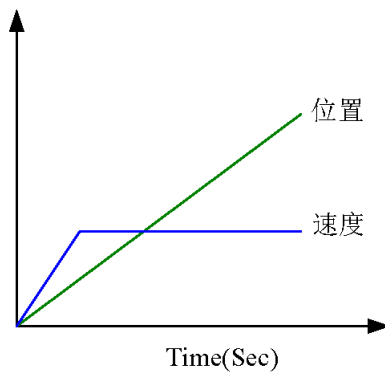
连续运动相关属性如下表 7.4 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 7.4：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

7.2.3.2 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。

连续运动过程中，可改变运行轴的运行速度（详见 7.2.5 章节）。

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第 6 章。

7.2.3.3 流程图

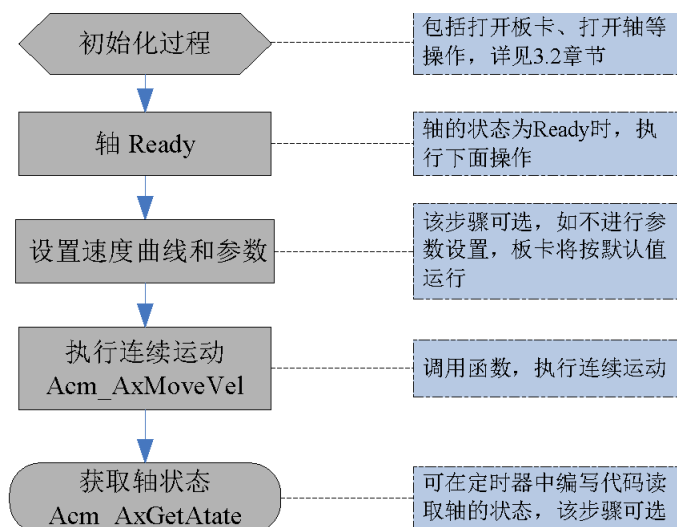


图 7.2：连续运动流程图

7.2.3.4 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-1245 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[4]; // 轴的 handle
U32     Ret; // 函数返回值
--- 初始化过程见 3.2.7 节 ---
// 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。
```

7.2.4 变位运动

7.2.4.1 变位运动相关函数与属性

变位运动相关函数如下表 7.5 所示，下表中的函数可直接调用。

表 7.5：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 位整形）
Acm_SetI32Property	设置属性值（属性值为 Int32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为 Int32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

变位运动相关属性如表 7.6 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。

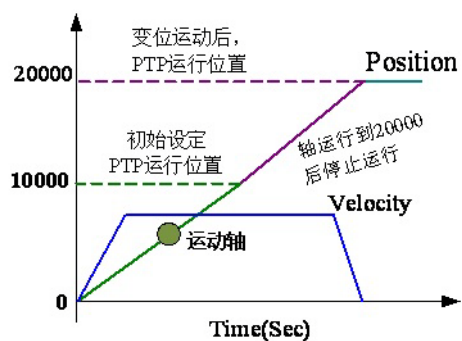
表 7.6：变位运动相关属性	
参数	说明
PAR_AxVelLow	设置 / 获取该轴的低速度（起始速度）
PAR_AxVelHigh	设置 / 获取该轴的高速度（驱动速度）
PAR_AxAcc	设置 / 获取该轴的加速度
PAR_AxDec	设置 / 获取该轴的减速度
PAR_AxJerk	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxMaxVel	配置运动轴的最大速度
CFG_AxMaxAcc	配置运动轴的最大加速度
CFG_AxMaxDec	配置运动轴的最大减速度

7.2.4.2 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。

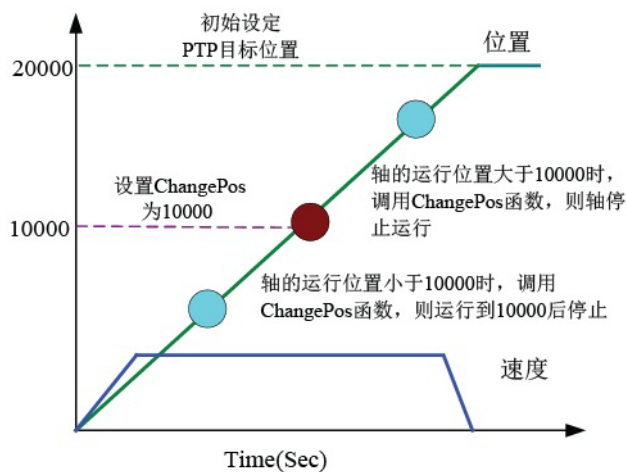
调用 ChangePos 函数改变运行轴的目标位置时，执行结果与轴的当前运动位置有关：

- 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：初始目标位置 20000，Change 的目标位置为 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



7.2.4.3 变位运动流程图

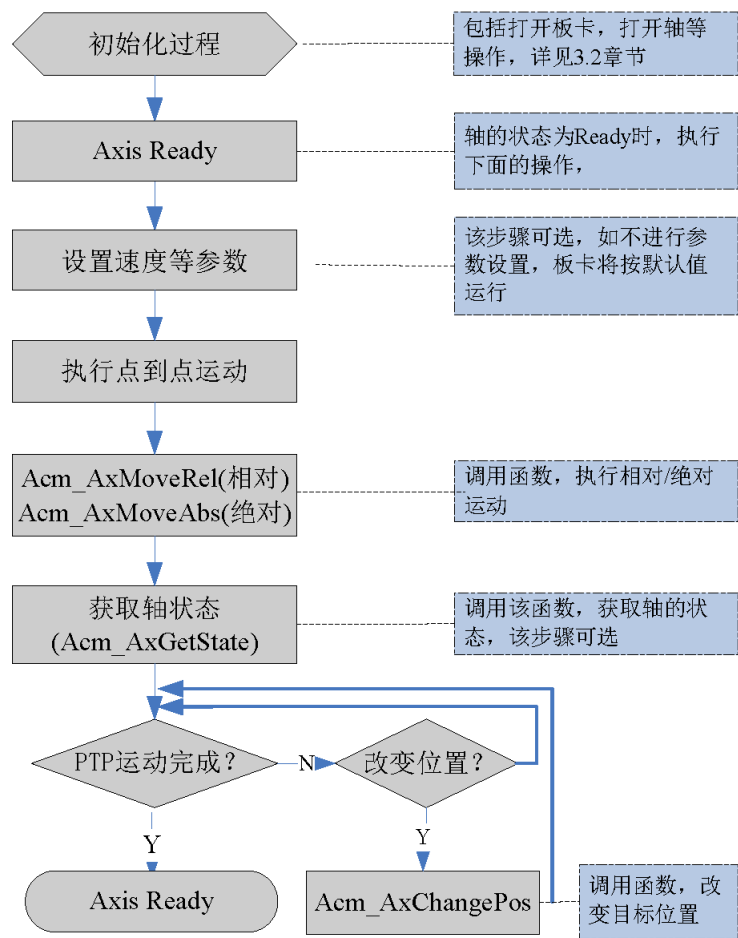


图 7.3: 变位运动流程图

7.2.4.4 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;
--- 初始化过程见 3.2.7 节 ---
// 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。

7.2.5 变速运动

7.2.5.1 变速运动相关函数与属性

变速运动相关函数如下表 7.7 所示，表中的函数可直接调用。

表 7.7：变速运行相关函数

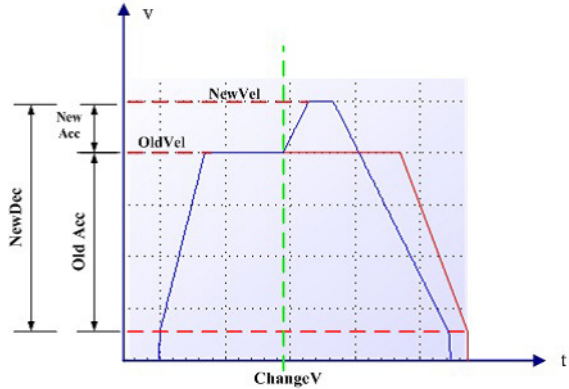
变速运动相关函数	说明
Acm_AxMoveVel	轴按照规定速度进行没有终点的运动
Acm_AxMoveAbs	开始单轴的绝对点到点运动
Acm_AxMoveRel	开始单轴的相对点到点运动
Acm_AxChangeVel	当轴在运动过程中，命令轴改变速度
Acm_AxChangeVelEx	改变运行轴的运行速度、加速度、减速度
Acm_AxChangeVelByRate	按照设定的比例改变当前轴的运行速度
Acm_AxChangeVelExByRate	按比率改变轴运行速度、加速度、减速度
Acm_AxGetCmdVelocity	获取指定轴的当前理论速度
Acm_AxSetCmdPosition	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 7.8 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 7.8：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

7.2.5.2 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。

7.2.5.3 变速运动流程图

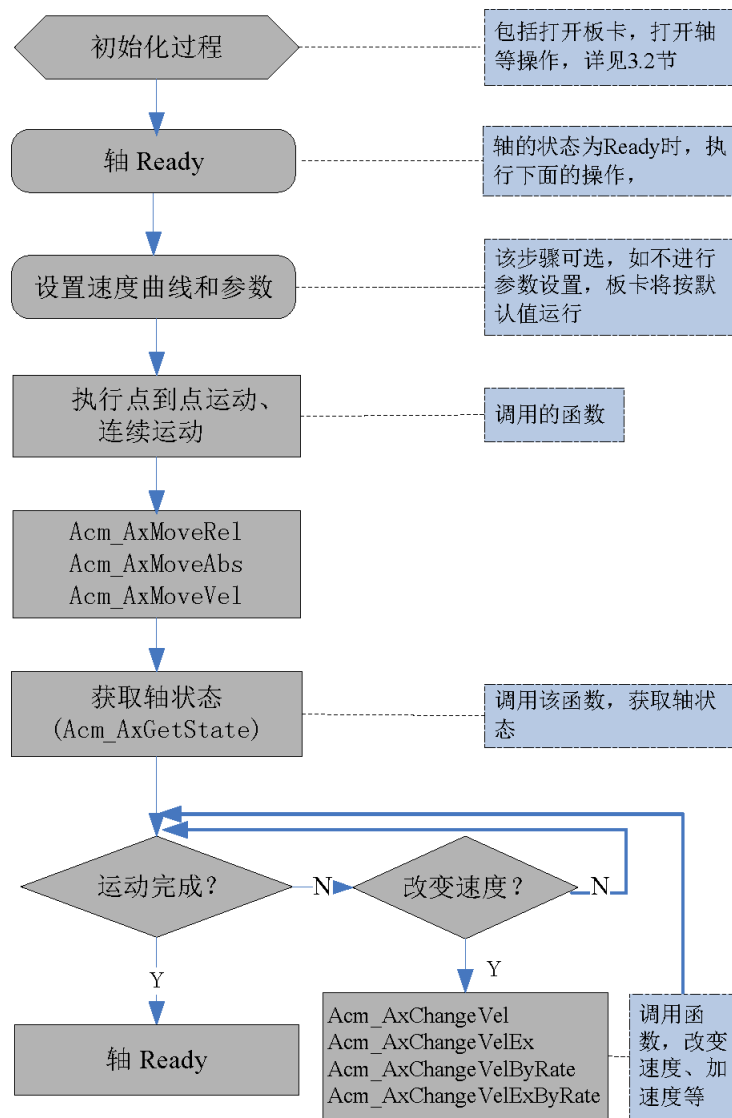


图 7.4: 变速运动流程图

7.2.5.4 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/S ²
减速度	10000PPU/S ²
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[4];
```

```
U32     Ret;
```

```
--- 初始化过程见 3.2.7 节 ---
```

```
// 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。

```

7.2.6 同步起、同步停运动

7.2.6.1 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 7.9 所示，下表中的函数可在应用程序中直接调用。

表 7.9：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

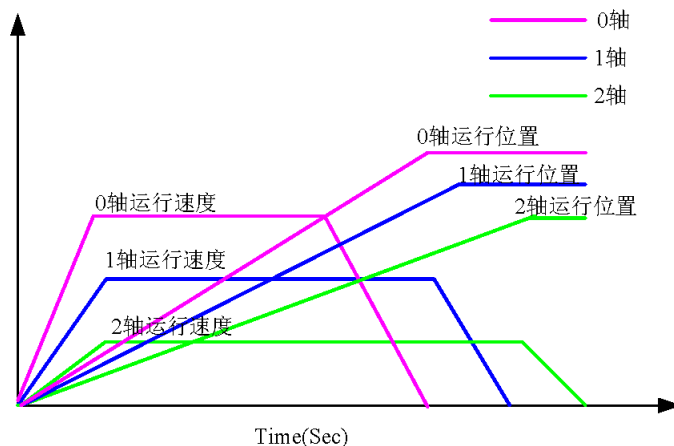
同步起、同步停运动相关属性如表 7.10 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 7.10：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。

7.2.6.2 重点说明

同步启、同步停运动指令等待轴（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。

7.2.6.3 同步起、同步停运动流程图

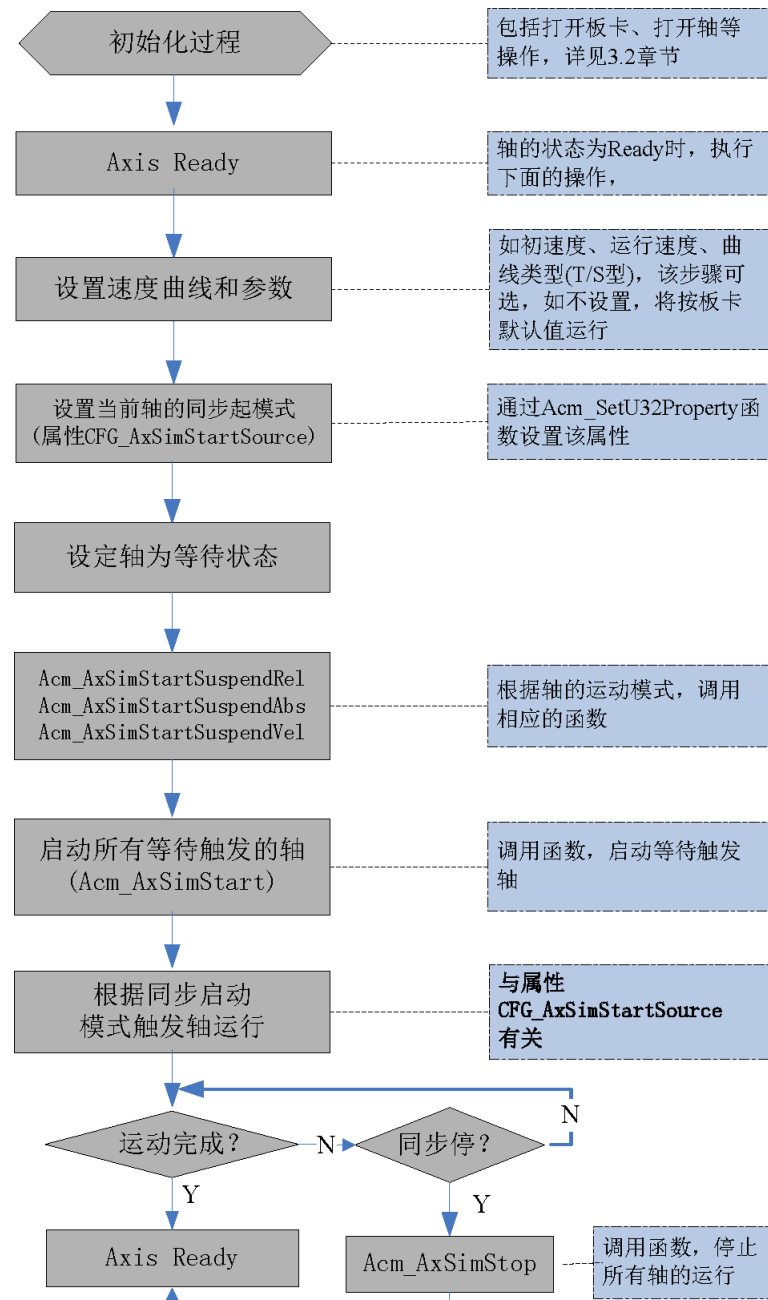


图 7.5：同步起、同步停运动流程图

7.2.6.4 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[4];
U32     Ret;
--- 初始化过程见 3.2 节 ---
// 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。

7.2.7 叠加运动

7.2.7.1 叠加运动相关函数

叠加运动相关函数如下表 7.11 所示，下表中的函数可在应用程序中直接调用。

表 7.11：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 7.11：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 7.12 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 7.12：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

7.2.7.2 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 7.6 所示。

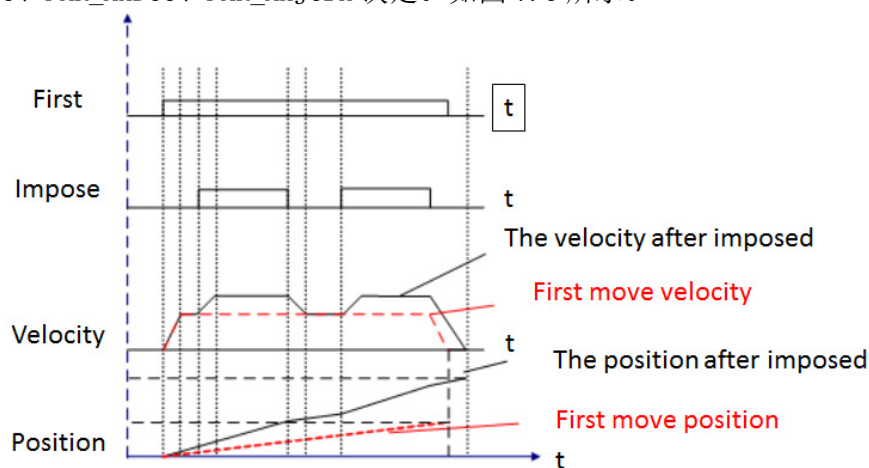


图 7.6：叠加运动

注！ 不能在叠加运动上叠加新的运动。



7.2.7.3 叠加运动流程图

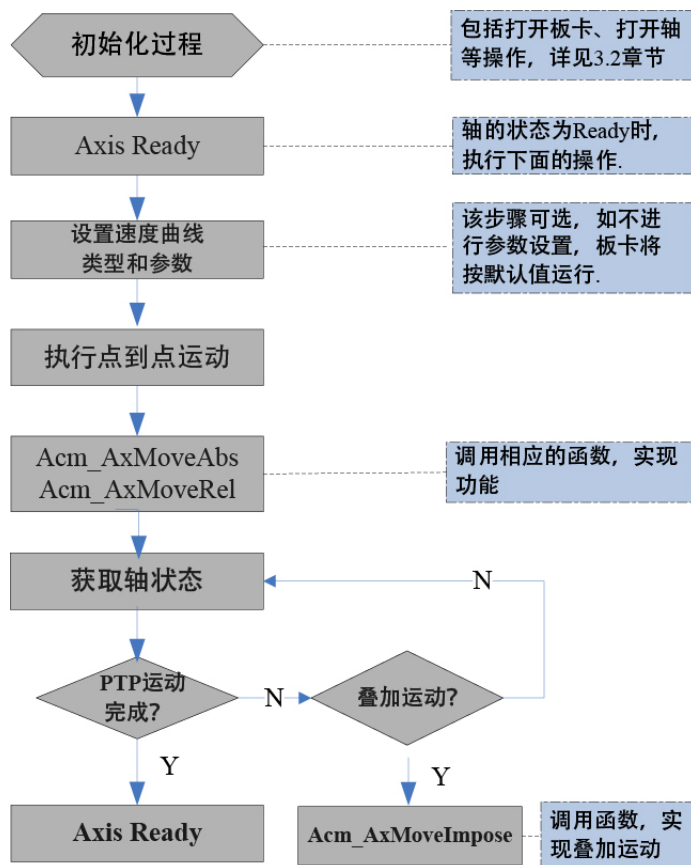


图 7.7: 叠加运动流程图

7.2.7.4 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[4];
U32     Ret;
--- 初始化过程见 3.2 节 ---
// 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], & 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
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。
```

7.2.8 JOG 运动、手轮运动

7.2.8.1 JOG、手轮运动相关函数与属性

JOG 运动、手轮运动相关函数如下表 7.13 所示。表中的函数可在应用程序中直接调用。

表 7.13：JOG、手轮运动相关函数	
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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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、手轮运动相关属性如表 7.14 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 7.14：JOG、手轮运动相关属性

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	设置速度曲线的类型：T/S 型曲线
配置	说明
CFG_AxExtSelEnable	启用 / 禁用外部驱动
CFG_AxExtPulseNum	理论脉冲个数
CFG_AxExtPulseInMode	设置 / 获取外部驱动脉冲输入模式
CFG_AxJogVelLow	设置 / 获取执行 Jog 运动时的初速度
CFG_AxJogVLTime	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
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

7.2.8.2 重点说明

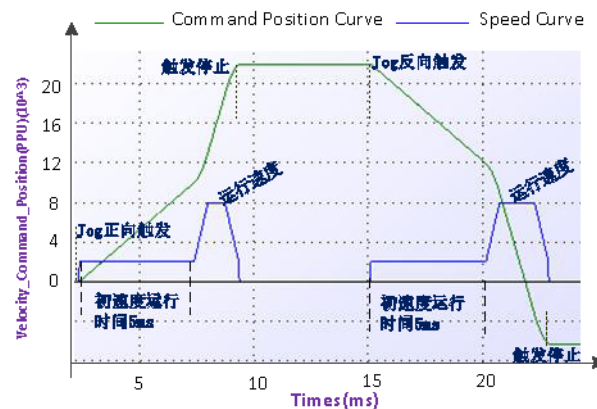
Jog 运动、手轮运动模式下，可单独设置各轴的初速度、运行速度、加速度等速度参数，支持同一时间有两个以上的轴在 JOG 状态。

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。

手轮运动模式下，通过设置参数 PAR_AxVelLow、PAR_AxVelHigh、PAR_AxVelAcc、PAR_AxVelDec 的值，设置手轮运动时的初速度、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。



根据板卡上针脚定义，X_IN4 及 X_IN5 可支持 X 轴的 JOG 和手轮模式。

X_IN4、有三种功能：

通用数字量输入

JOG+

MPG+

X_IN5 同样也有三种功能：

通用数字量输入

JOG-

MPG-

只有 X_IN4, X_IN5 可以接 JOG 信号，如果要设置其他轴执行 Jog 运动时，就输入该轴的 Handle。

7.2.8.3 JOG、手轮运动流程图

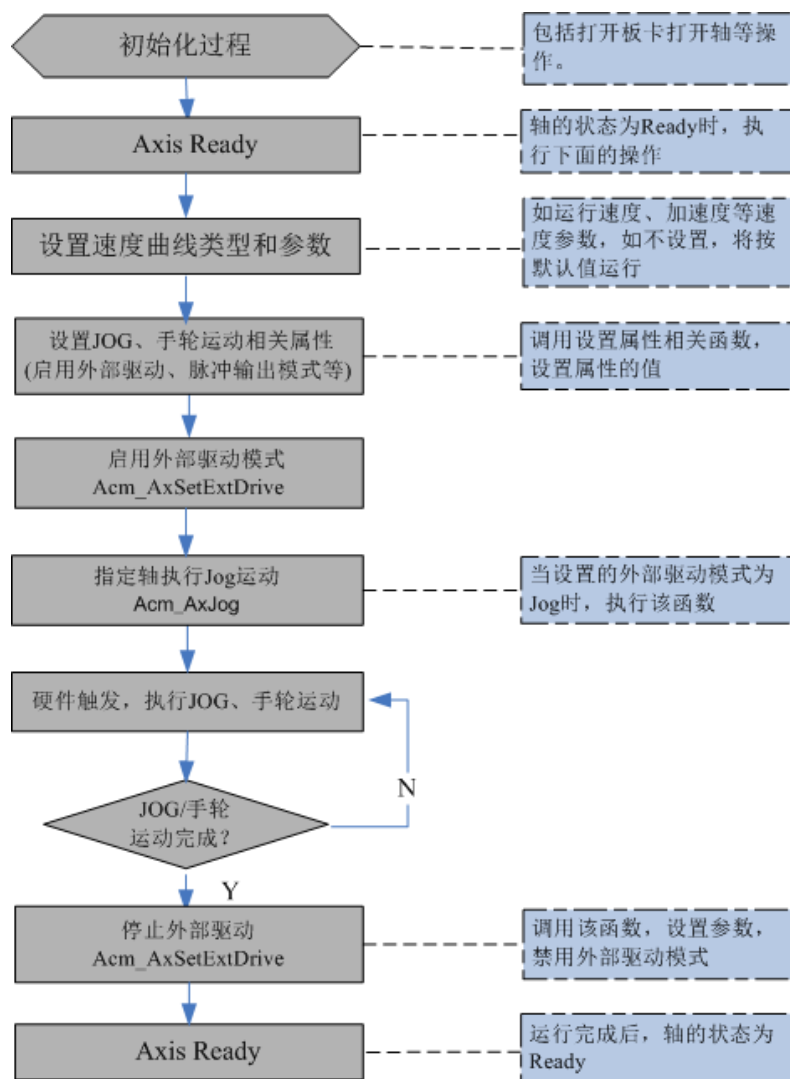


图 7.8: JOG、手轮运动流程图

7.2.8.4 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[4];
U32     Ret;
--- 初始化过程见 3.2 节 ---
// 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_AxExtSelEnable, 1); // 启用外部
驱动
Ret  = Acm_SetU32Property(m_Axishand[0], CFG_AxExtPulseNum, 1000); // 理论脉冲
数
Ret  = Acm_SetU32Property(m_Axishand[0], CFG_AxPulseOutMode, 1);
// 使能 jog 模式
Ret  = Acm_AxSetExtDrive(m_Axishand[0], 1); // Enable Jog Mode
// 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/MPG 例程
```

7.2.9 原点复归

7.2.9.1 原点复归相关函数与属性

原点复归相关函数如下表 7.15 所示，下表中的函数可直接调用。

表 7.15：原点复归相关 API

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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.16 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 7.16：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 7.16: 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

7.2.9.2 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。电机停止运行后，如果属性 CFG_AxHomeResetEnable 设置为“True”，则原点运动终止后，理论位置和实际位置将复位至零。

原点信号默认为上升沿触发。设置 CFG_AxOrgLogic、CFG_AxEILogic、CFG_AxEzLogic 属性可以修改 ORG、LMT+-、EZ 信号的逻辑准位。

研华运动控制卡共提供两个函数执行回 Home：

Acm_AxHome 和 Acm_AxMoveHome

调用 Acm_AxHome 执行回 Home 时，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 时，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。

研华运动控制卡共提供 16 种原点复归模式，在 Acm_AxHome 和 Acm_AxMoveHome 函数中设置。

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。同时通过 PAR_AxHomeCrossDistance 设置跨越距离。

16 种 Home 模式如下所示。

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

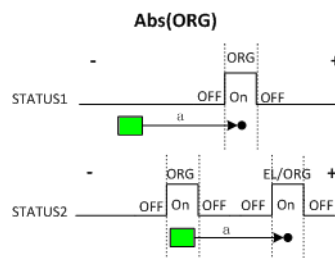
只依照原点设备（如传感器）返回原点。对象会一直运动，直至原点信号发生。

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。

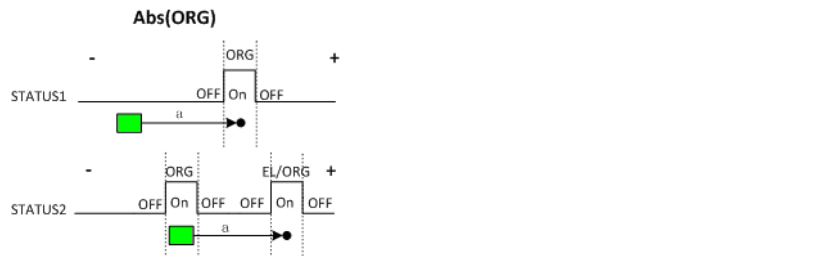
2. MODE2_Lmt: Move(Dir)->touch EL->Stop

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。

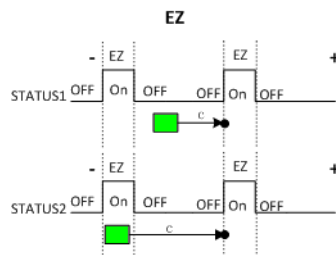
3. MODE3_Ref: Move (Dir) ->touch EZ->Stop

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

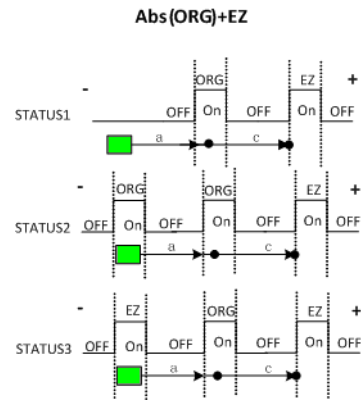
这是一种组合的原点模式。首先，对象将一直运动直到原始信号发生，然后将保持以 ORG 同一方向继续运动，直到 EZ 信号发生。

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。

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

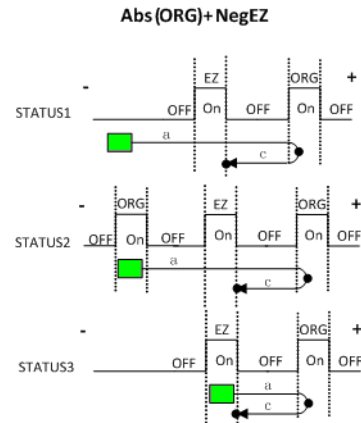
这是一种组合的原点模式。首先，对象将一直运动直到原始信号发生，然后将以与 ORG 相反方向继续运动，直到 EZ 信号发生。

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。

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

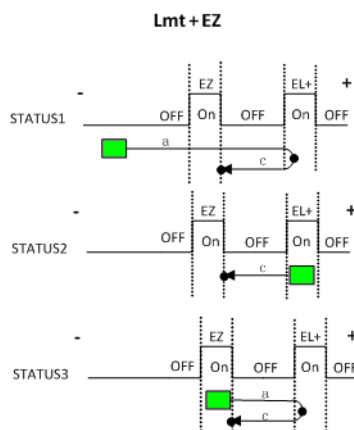
首先，对象将一直运动直到原始信号发生，然后将以与 ORG 相反方向继续运动，直到 EZ 信号发生。

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.

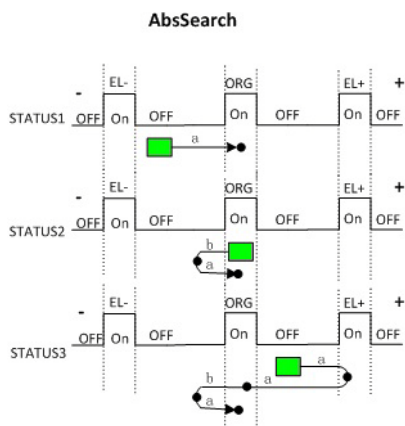
这是一种搜索 ORG 信号从无到有转换的模式。

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。

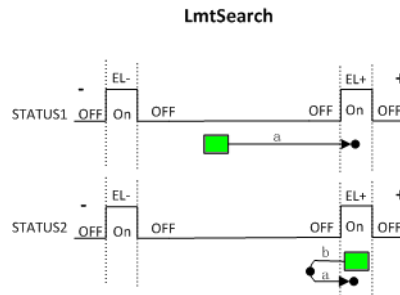
8. MODE8_LmtSearch: Move (Dir) ->Search EL ->Stop.

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.

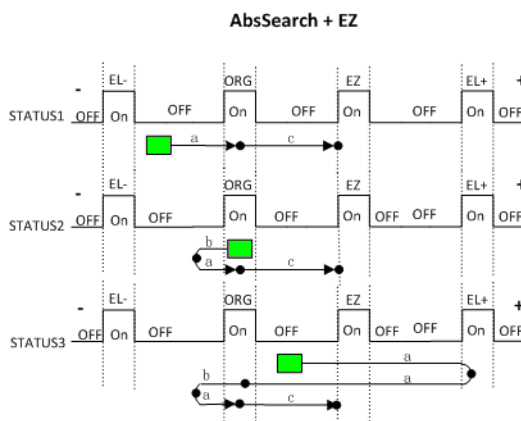
首先，对象以 MODE7_AbsSearch 方式运动，然后以相同方向运动直到 EZ 信号发生。

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.

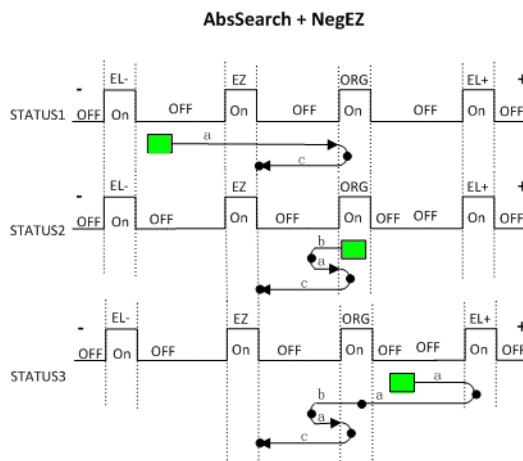
首先，对象以 MODE7_AbsSearch 方式运动，然后以相反方向运动直到 EZ 信号发生。

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。

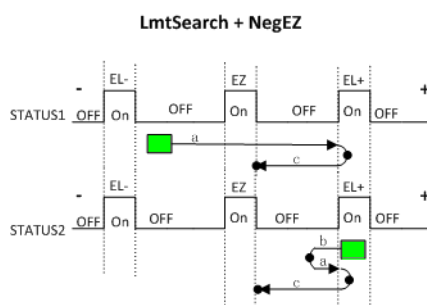
11. MODE11_LmtSearch_Ref: Search EL +NegEZ, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->touch EZ ->Stop.

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.

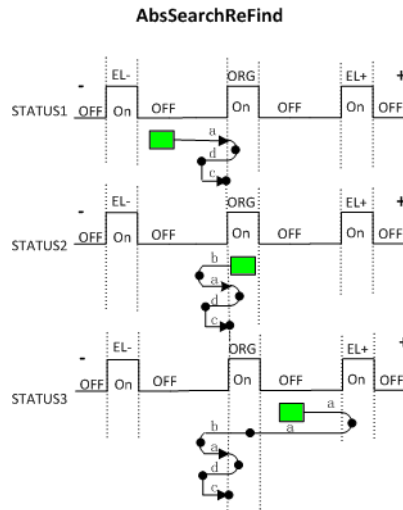
首先，轴以 MODE7_AbsSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 ORG 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 ORG 信号发生。

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。

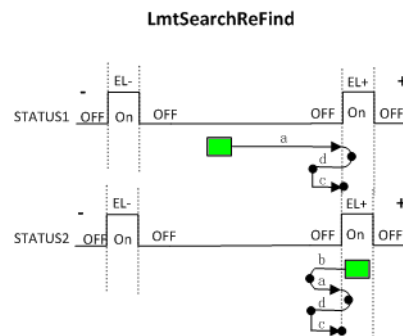
13. **MODE13_LmtSearchReFind**: Search EL +Refind EL, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->Leave EL(FL) ->Stop-> Move (-Dir)->Refind EL(FL)->Stop.

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.

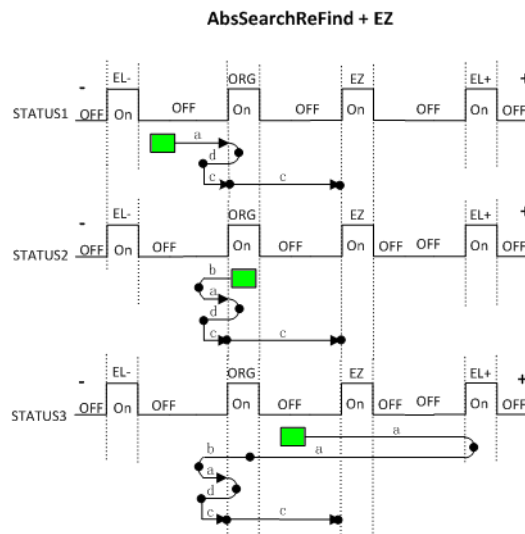
首先，轴以 MODE7_AbsSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 ORG 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 ORG 信号发生；最后轴沿相同方向运动至找到 Z 相。

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.

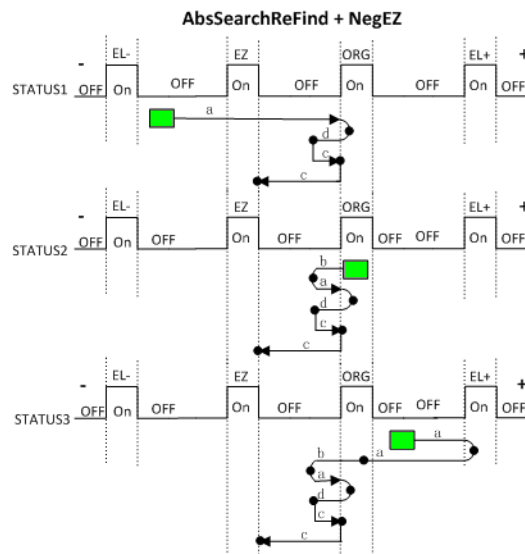
首先，轴以 MODE7_AbsSearch 模式运动；然后轴反向以低速（VelLow）等速运动直至 ORG 信号消失；然后轴再次反向以低速（VelLow）等速运动直至 ORG 信号发生；最后轴再次反向运动直至 EZ 信号发生。

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。

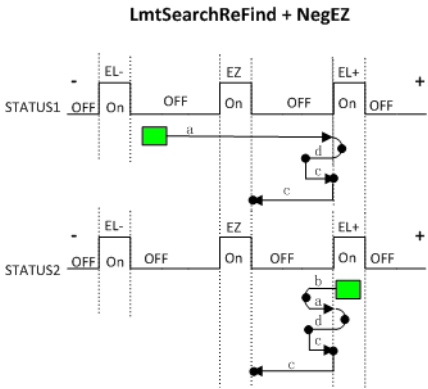
16. MODE16_LmtSearchRefind_Ref: Search EL +Refind EL+EZ, Move (Dir) ->Search EL ->Stop->Move (-Dir) ->Leave EL(FL) ->Stop-> Move (-Dir)->Refind EL(FL)->Stop->Move (-Dir) ->touch EZ ->Stop.

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。

7.2.9.3 原点复归流程图

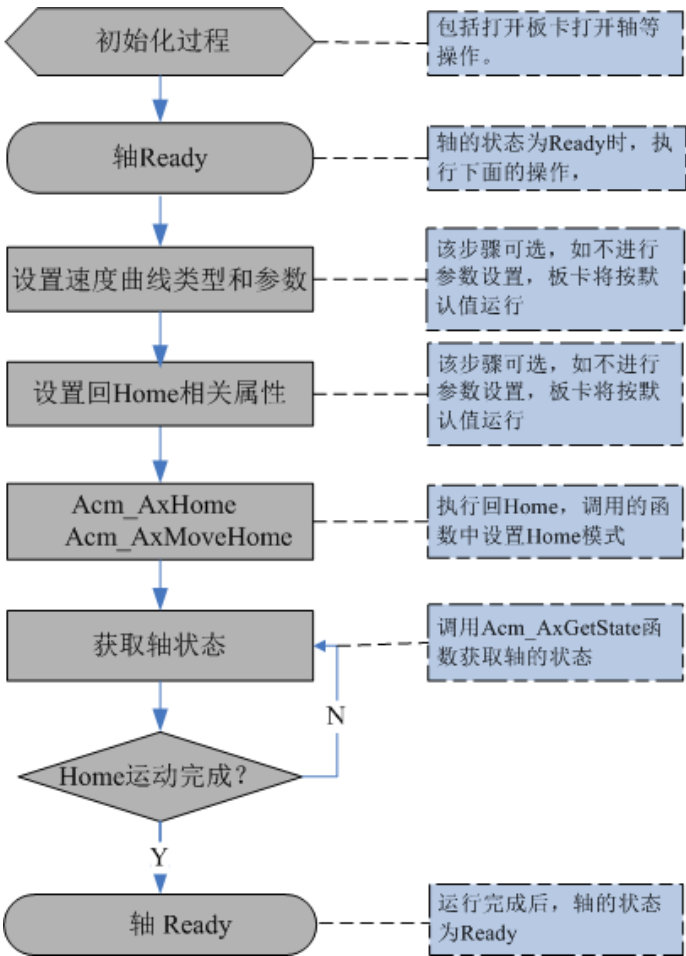


图 7.9： 原点复归流程图

7.2.9.4 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/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：

```

HAND    m_Axishand[4];
U32      Ret;
--- 初始化过程见 3.2 节 ---
// 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

```

7.2.10 轴的事件

7.2.10.1 轴的事件相关 API 与属性

轴的事件相关函数如下表 7.17 所示，下表中的函数可直接调用。

表 7.17：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

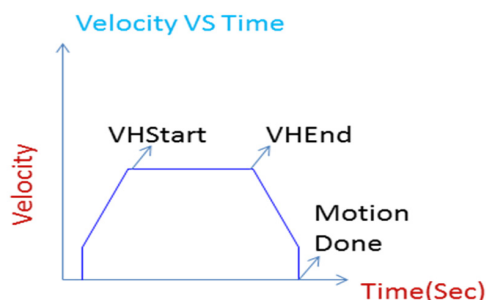
轴的事件相关属性如表 7.18 所示，属性不能直接调用，而是在设置和获取属性的函数中设置和获取属性的值。

表 7.18：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

7.2.10.2 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。

为了检测轴的事件，首先调用 Acm_EnableMotionEvent 函数，用来启用 / 禁用每个轴的事件状态。启用 / 禁用每个轴的事件，通过设置参数 AxEvtStatusArray[N] 实现。该参数为 U32 位整形数组阵列，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，如图所示。

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 位整形数组阵列，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，返回 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

7. 2. 10. 3轴的事件流程图

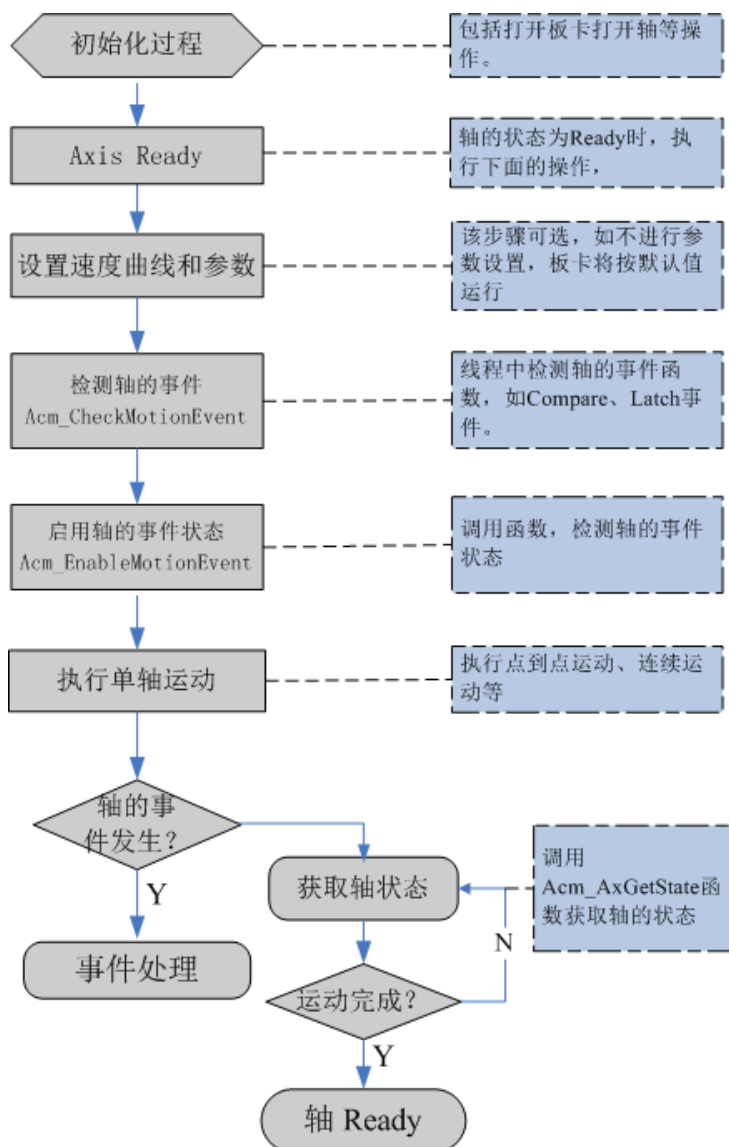


图 7. 10: 轴的事件流程图

7.2.10.4 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/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[4];
U32     Ret;
U32     AxEnableEvtArray[4];
U32     GpEnableEvt[3];
U32     m_ulAxisCount;
BOOL    m_bInit ;
--- 初始化过程见 3.2 节 ---
// 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; //Enable motion done
AxEnableEvtArray[0] |= EVT_AX_VH_START; //Enable VHStart
AxEnableEvtArray[0] |= EVT_AX_VH_END; //Enable VHEND
Ret = Acm_EnableMotionEvent(m_Devhand, AxEnableEvtArray, GpEnableEvt,
m_ulAxisCount ,3); //Enable event
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。

7.3 插补运动模式

7.3.1 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

7.3.2 直线插补运动

7.3.2.1 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 7.19 所示，该表中的函数可在应用程序中直接调用。

表 7.19：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.20 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。

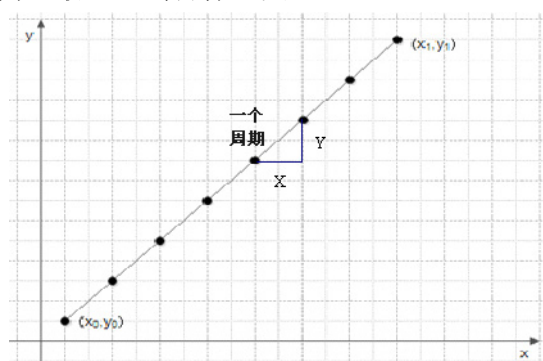
表 7.20：直线插补运动相关属性

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

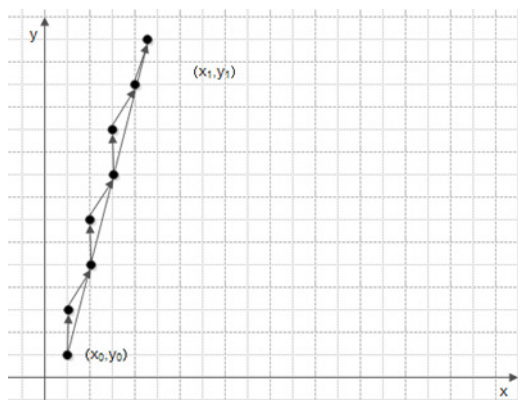
7.3.2.2 重点说明

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。

7.3.2.3 直线插补运动流程图

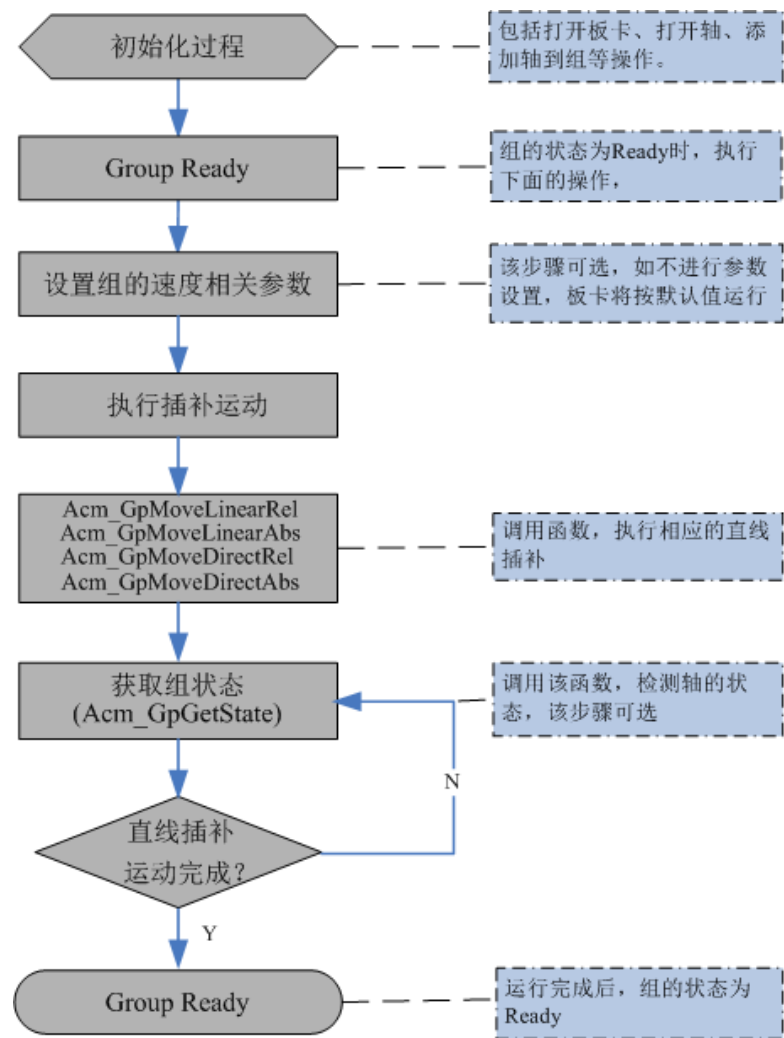


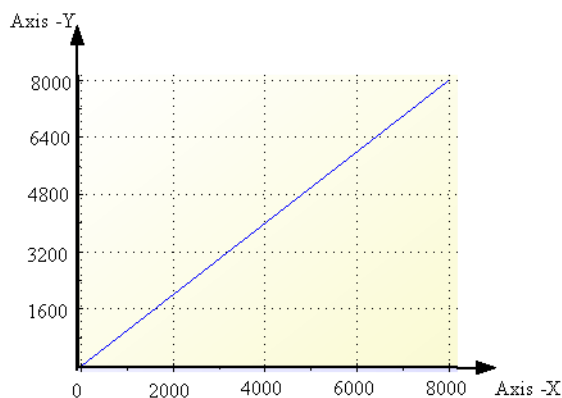
图 7.11： 直线插补运动流程图

7.3.2.4 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-1245 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};
--- 初始化过程见 3.2 节 ---
// 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。

7.3.3 圆弧插补

7.3.3.1 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 7.21 所示，该表中的函数可在应用程序中直接调用。

表 7.21：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 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_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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

圆弧插补相关属性如表 7.22 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 7.22：圆弧插补运动相关函数

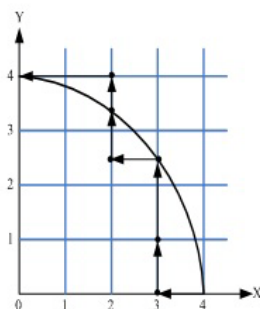
参数	说明
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID
配置	说明
CFG_GpAxesInGroup	获取哪个（哪些）轴在该群组中的信息

7.3.3.2 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。

7.3.3.2.1 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

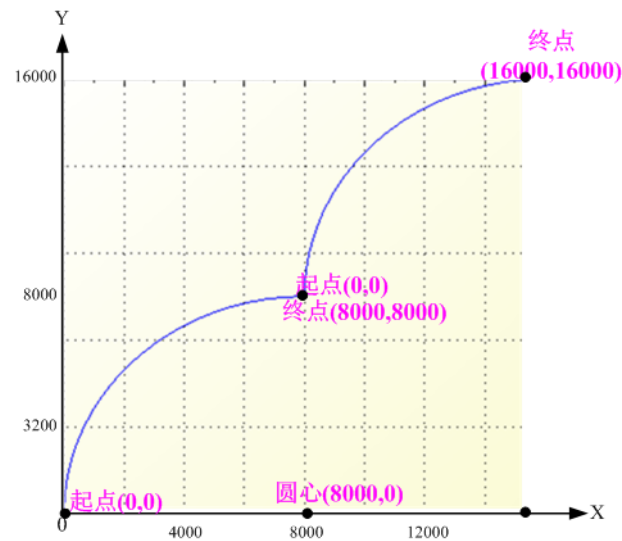
方法 3：根据圆心和角度确定圆弧，执行圆弧插补

现分别介绍如下：

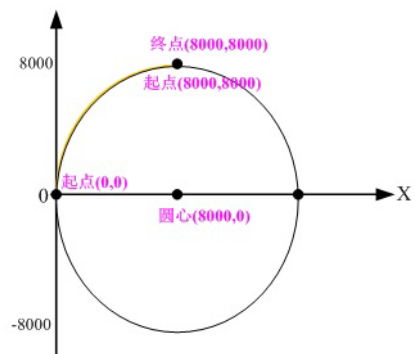
方法 1：根据起点、圆心、终点执行圆弧插补

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。



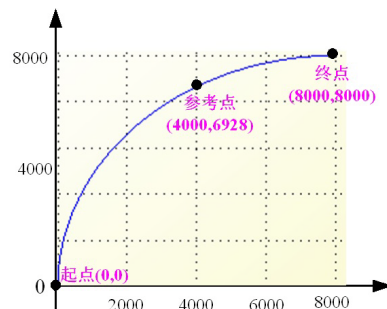
连续执行两次绝对圆弧插补，执行结果如下图所示。



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。

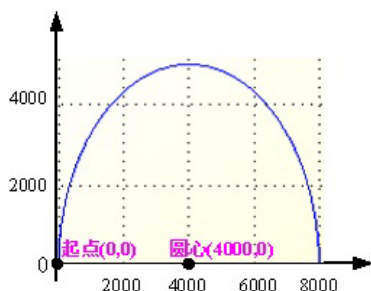
方法 2：根据三点坐标确定圆弧，执行圆弧插补

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。



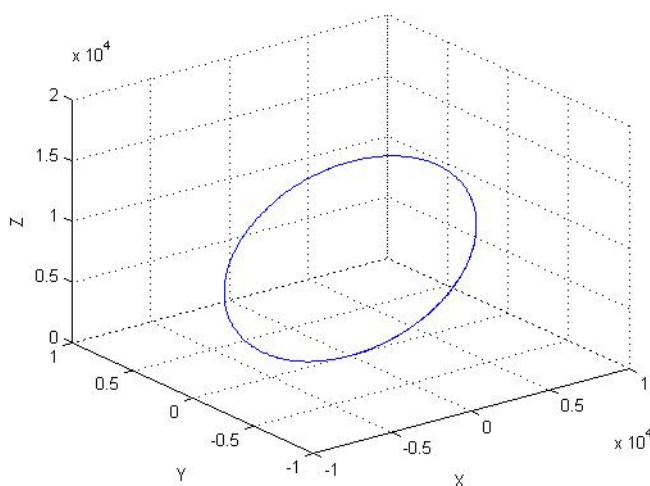
方法 3：根据圆心和角度确定圆弧，执行圆弧插补

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。



7.3.3.2.2 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

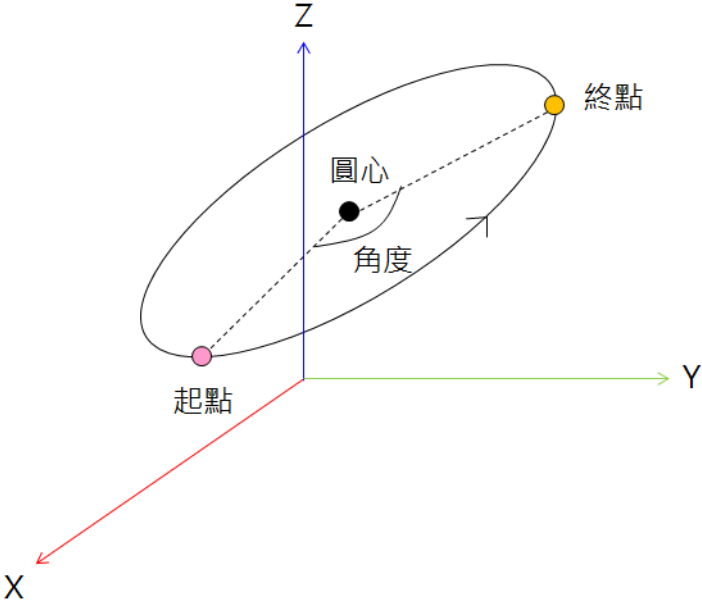
现分别介绍如下：

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

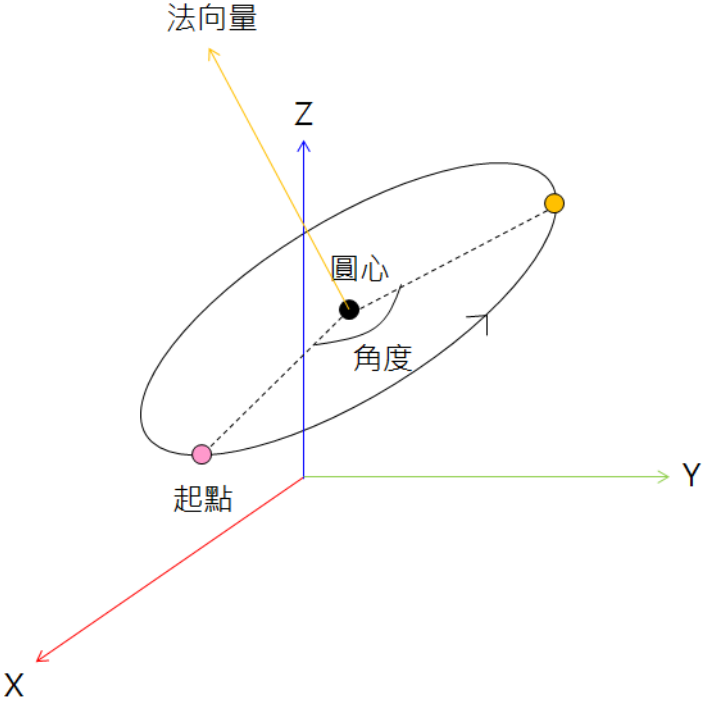
圆心、终点、180/360 度的圆在一条直线上，一条直线不能确定一个平面。遇到此种情况，可使用方法二，根据法向量确定圆弧所在平面，执行圆弧插补。三轴圆弧插补如下图所示。



三轴圆弧插补（方法一）

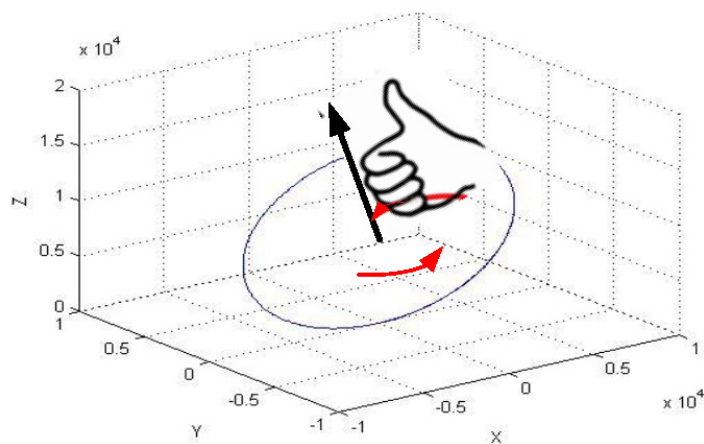
方法 2：根据圆心、法向量、角度和方向确定圆弧，执行圆弧插补。

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。

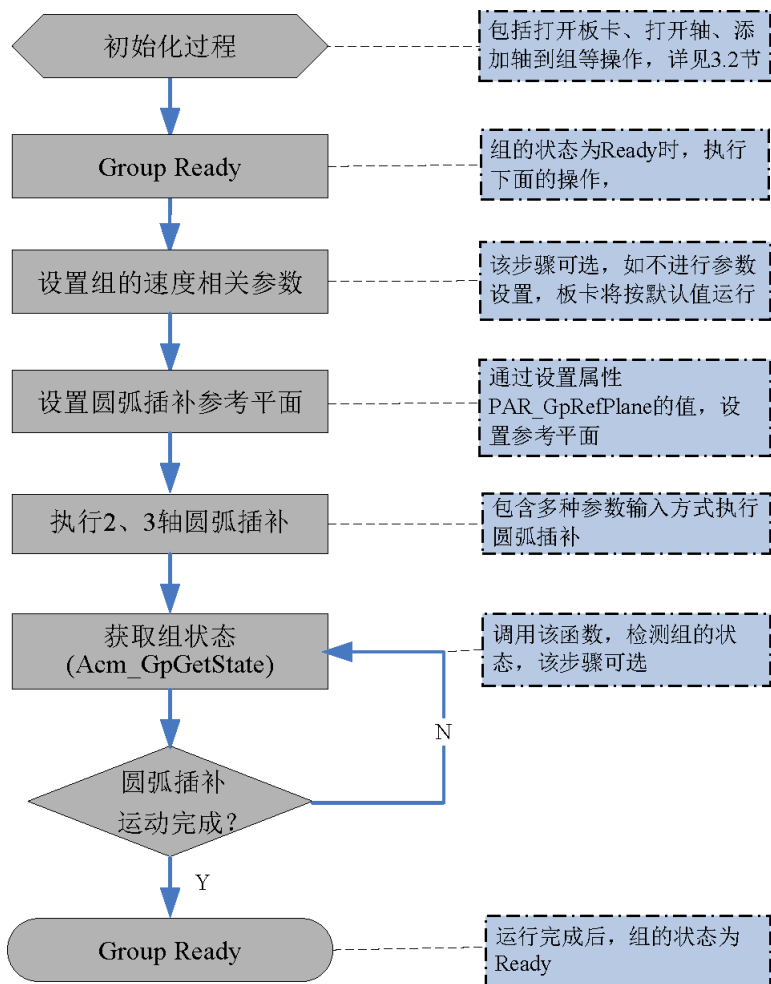


3 轴圆弧插补圆弧旋转方向按如下方式确定：

由右手定则来决定方向：拇指沿着法向量方向，其余四指的方向定义为正方向。



7.3.3.3 圆弧插补流程图

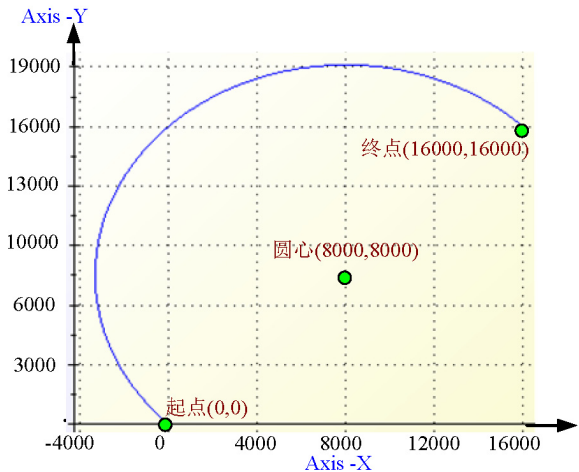


7.3.3.4 例程

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-1245 运动控制卡)
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
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
--- 初始化过程见 3.2 节 ---
// 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。

7.3.4 螺旋插补运动

7.3.4.1 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 7.23 所示，该表中的函数可在应用程序中直接调用。

表 7.23：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 7.24 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。

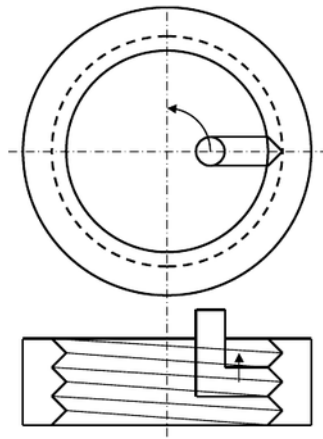
表 7.24：螺旋插补运动相关函数	
参数	说明
PAR_GpVelLow	设置 / 获取该轴的初速度（起始速度）
PAR_GpVelHigh	设置 / 获取该轴的运行速度
PAR_GpAcc	设置 / 获取该轴的加速度
PAR_GpDec	设置 / 获取该轴的减速度
PAR_GpJerk	设置 / 获取速度曲线的类型：T/S 型曲线
PAR_GpGroupID	获取组的 ID

7.3.4.2 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

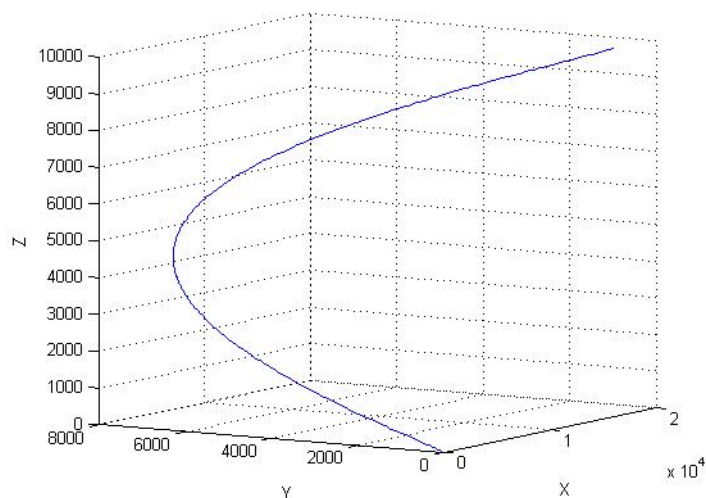
方法 2：根据圆心、角度、方向执行螺旋插补

方法 3：根据三点坐标执行螺旋插补

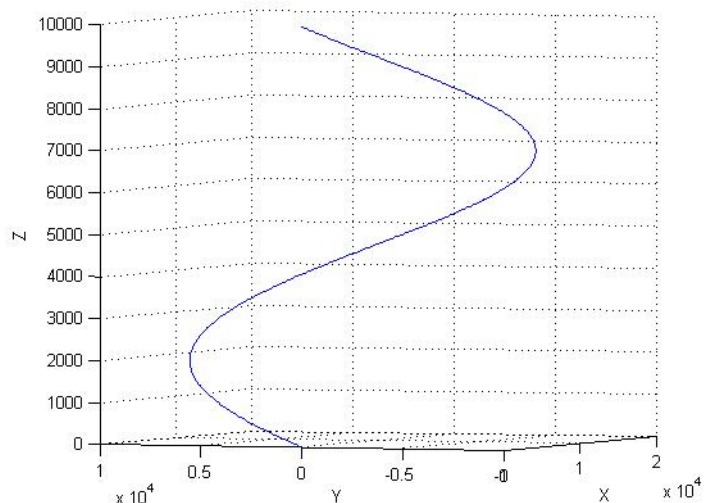
现分别介绍如下：

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。

7.3.4.3 螺旋插补运动流程图

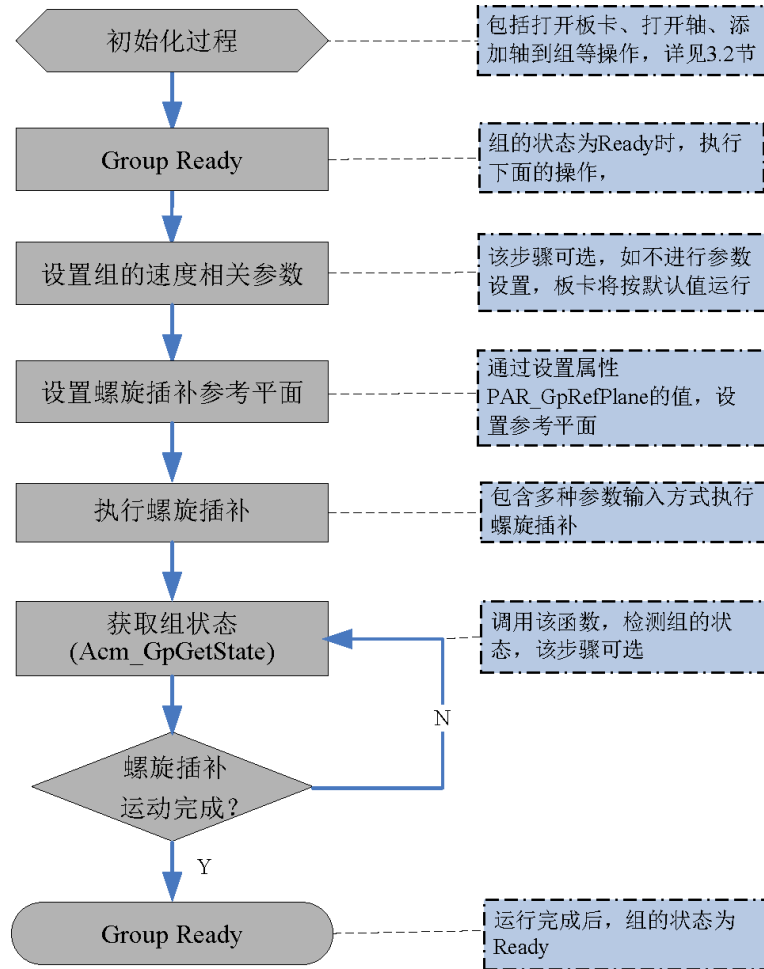


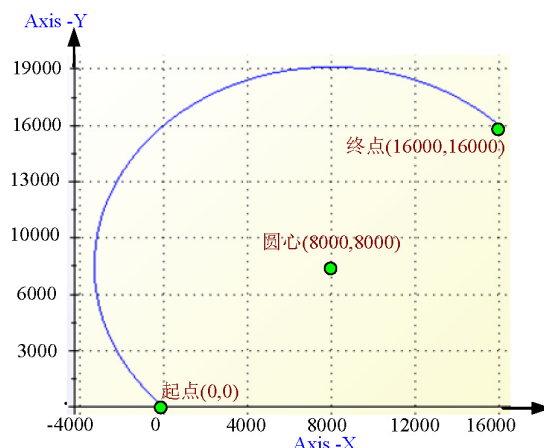
图 7.12: 螺旋插补运动流程图

7.3.4.4 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-1245 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
--- 初始化过程见 3.2 节 ---
// 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。

7.3.5 群组的事件

7.3.5.1 群组事件相关 API 与属性

群组事件相关函数如下表 7.25 所示，下表中的函数可直接调用。

表 7.25：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

7.3.5.2 重点说明

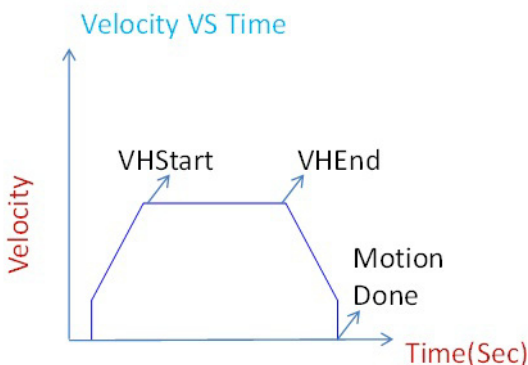
群组事件包括 Motion Done、VHStart、VHEnd。

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。

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-1245 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。

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

- 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-1245 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。

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

7.3.5.3 群组事件流程图

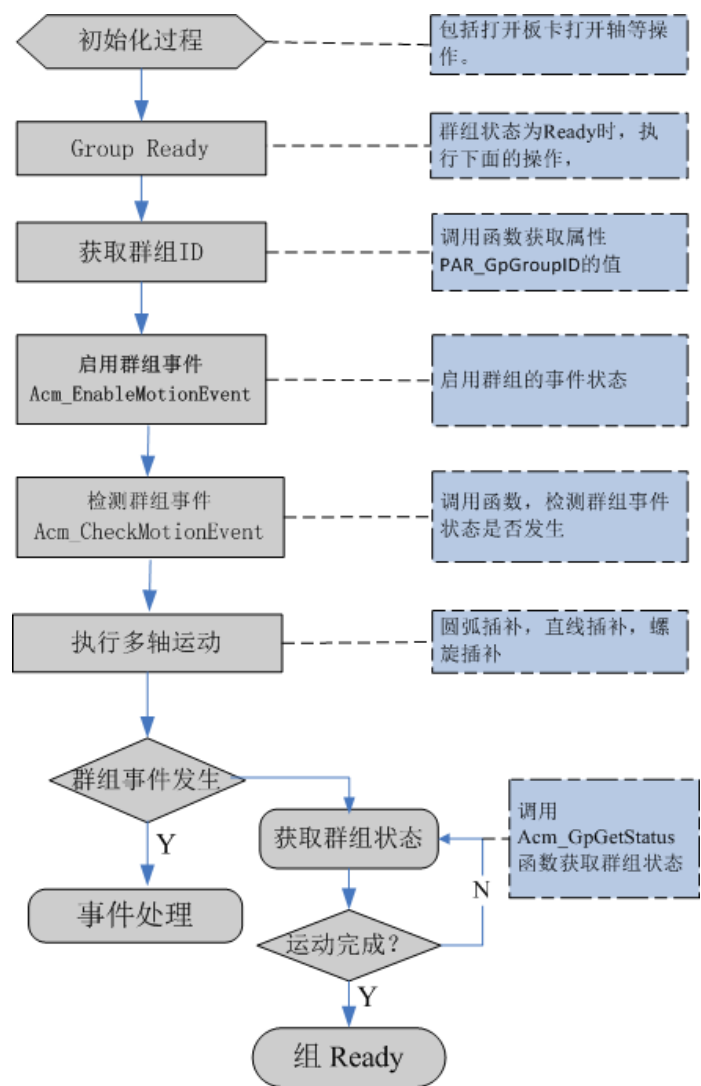


图 7.13: 群组事件流程图

7.3.5.4 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-1245
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
--- 初始化过程见 3.2 节 ---
```

```

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。

7.4 跟随运动模式

7.4.1 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

7.4.2 电子齿轮运动

7.4.2.1 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 7.26 所示，该表中的函数可在应用程序中直接调用。

表 7.26：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.27 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 7.27：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

7.4.2.2 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。

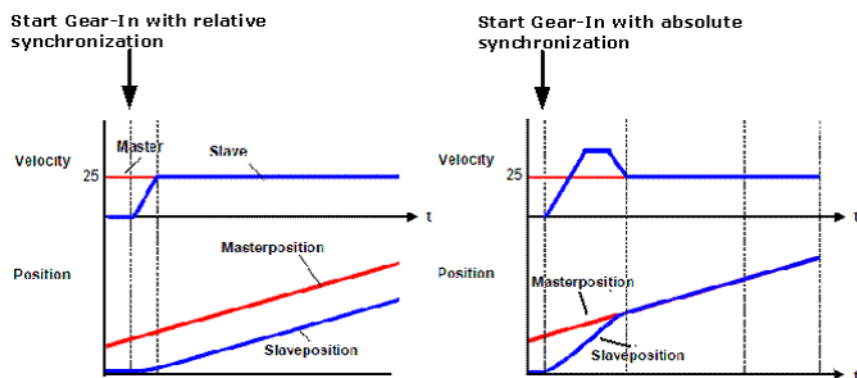
调用 Acm_AxGearInAx 函数，设定电子齿轮比，主轴和从轴之间将开始按设定的比例开始电子齿轮同步。可根据实际需求，多次调用该函数，以设定从轴和主轴的同步关系。

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。



7.4.2.3 电子齿轮运动流程图

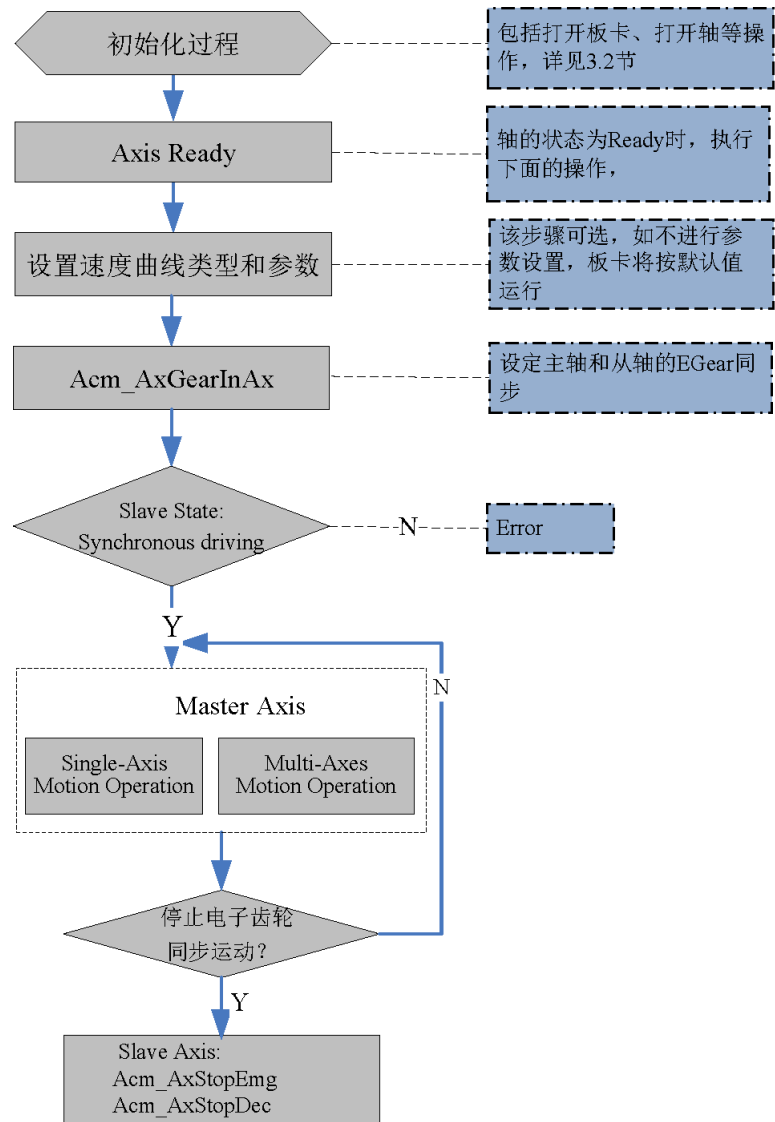


图 7.14: 电子齿轮流程图

7.4.2.4 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/S2
主轴减速度	10000PPU/S2
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[4]; // 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
--- 初始化过程见 3.2 节 ---
// 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。

7.4.3 电子凸轮运动

7.4.3.1 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 7.28 所示，该表中的函数可在应用程序中直接调用。

表 7.28：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.29 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 7.29：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_AxCamDOEnable	设置 / 获取 CamDO 启用 / 禁用
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_AxCamDOCmpSrc	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

7.4.3.2 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。

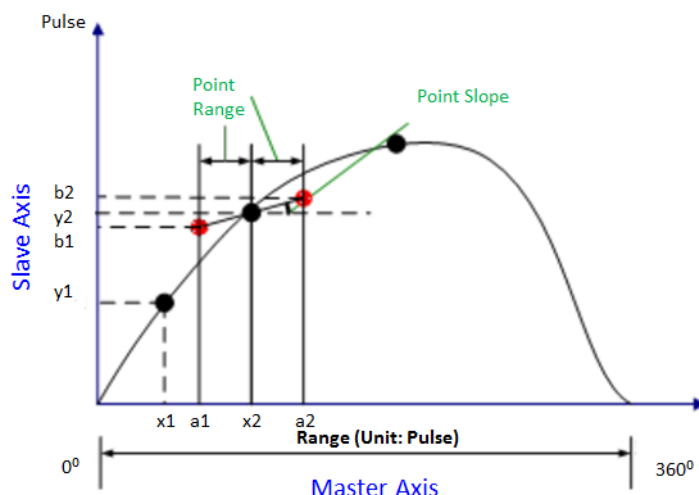
凸轮的主要特征是主轴和从轴之间的动态比例以及相位移动。主轴和从轴之间的关系通过 CAM 表描述，凸轮操作通过这张表执行操作。

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

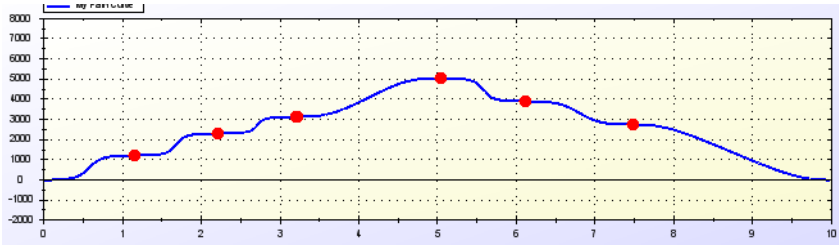
Acm_DevDownloadCAMTable 参数的含义如下：



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。

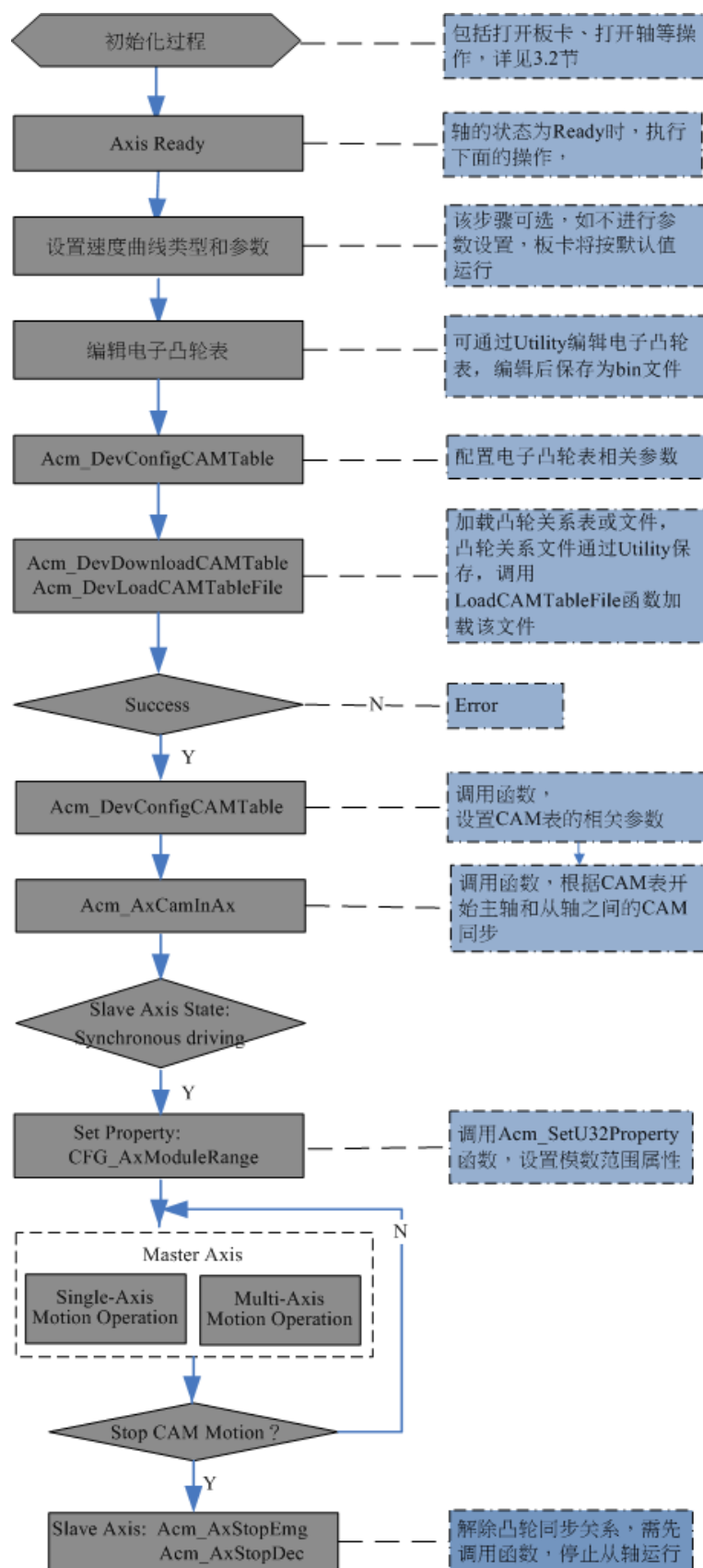
Range 必须通过属性 CFG_AxModuleRange 设置为主轴内。

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。

7.4.3.3 电子凸轮运动流程图



7.4.3.4 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[4]; // 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
--- 初始化过程见 3.2 节 ---
// 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。

```

7.4.4 龙门运动

7.4.4.1 龙门运动相关函数与属性

龙门运动相关函数如下表 7.30 所示，该表中的函数可在应用程序中直接调用。

表 7.30：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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

龙门运动相关属性如下表 7.31 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 7.31：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

7.4.4.2 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。

7.4.4.3 龙门运动流程图

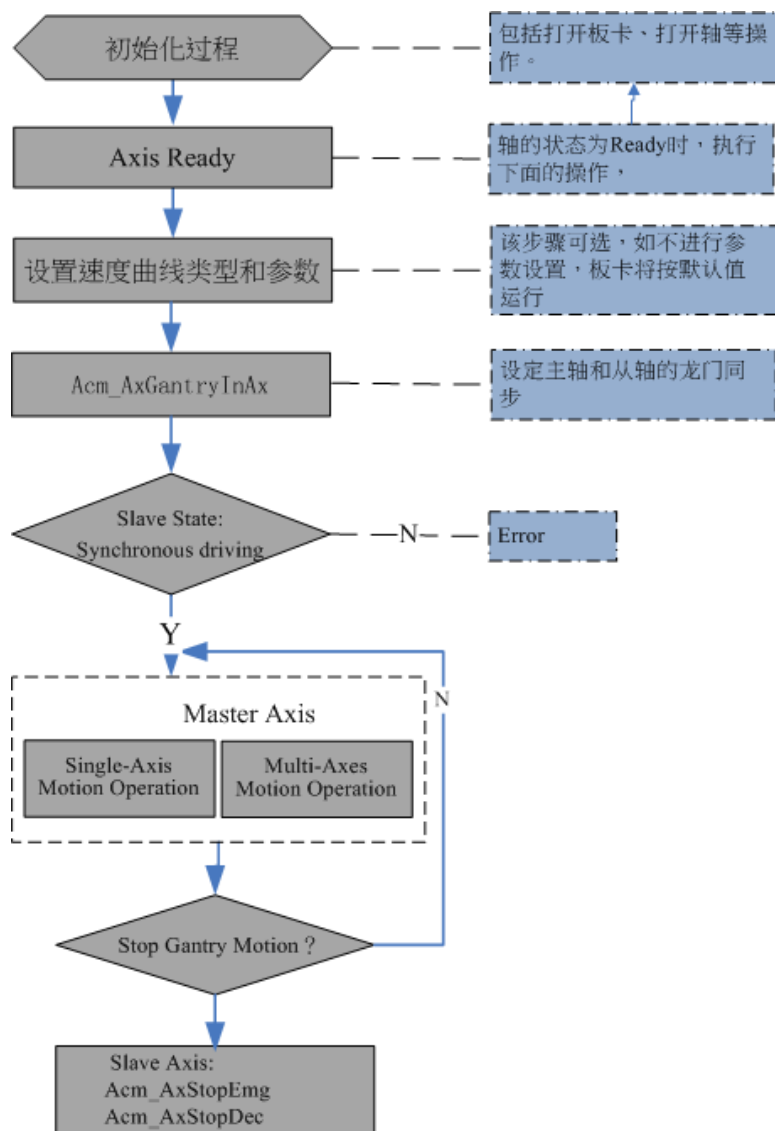


图 7.15: 龙门运动流程图

7.4.4.4 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[4]; // 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
--- 初始化过程见 3.2 节 ---
// 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。
```

7.4.5 切向跟随运动

7.4.5.1 切向跟随相关函数与属性

切向跟随相关函数如下表 7.32 所示，该表中的函数可在应用程序中直接调用。

表 7.32：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_GpMoveHelixAbs	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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
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 7.33 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。

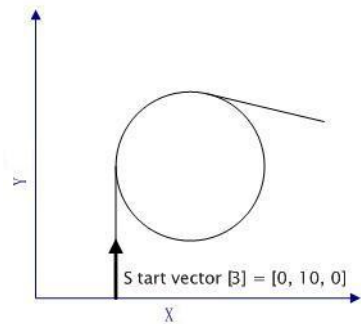
表 7.33：切向跟随运动相关属性

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

7.4.5.2 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。

7.4.5.3 切向跟随运动流程图

切向跟随运动流程图如下图所示：

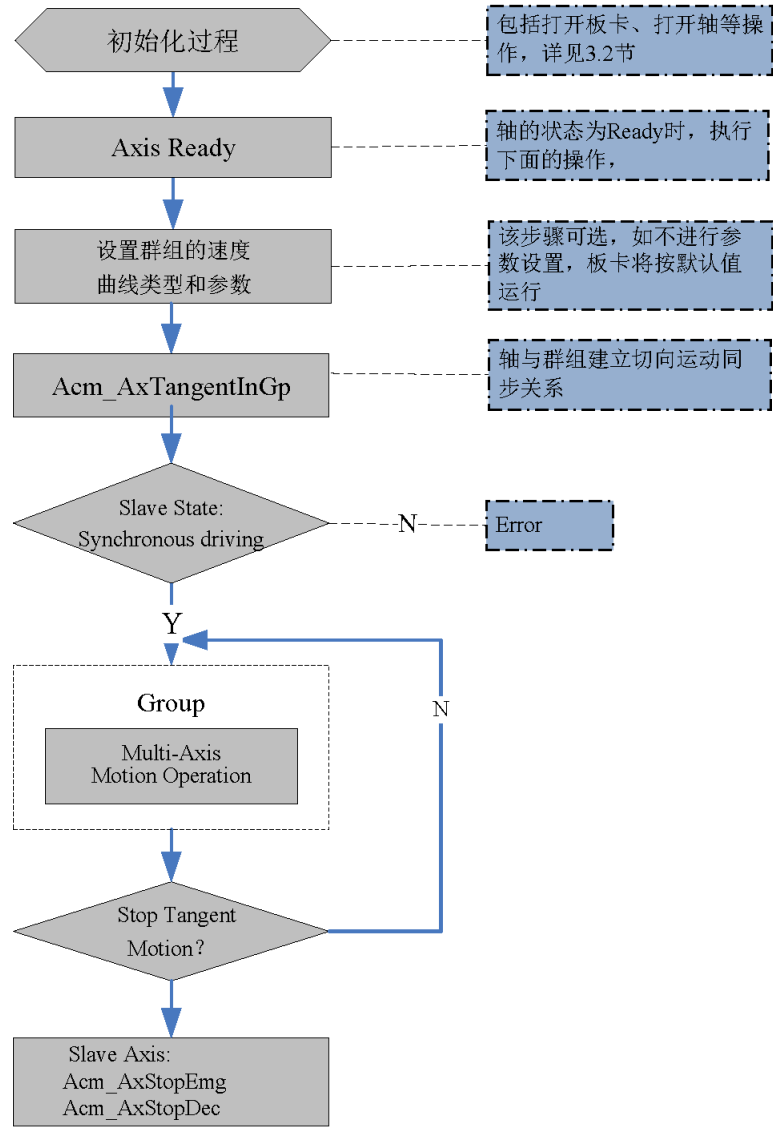


图 7.16：切向跟随运动流程图

7.4.5.4 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
--- 初始化过程见 3.2 节 ---
// 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。

```

7.5 路径规划模式

7.5.1 本节简介

路径规划模式支持用户预先给定最长达 10000 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。

7.5.2 Path 运动相关函数与属性

Path 运动相关函数如下表 7.34 所示，该表中的函数可在应用程序中直接调用。

表 7.34: 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 位整形）
Acm_SetI32Property	设置属性值（属性值为有符号 32 位整形）
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 位整形）
Acm_GetI32Property	获取属性值（属性值为有符号 32 位整形）
Acm_GetF64Property	获取属性值（属性值为 Double 型）

Path 运动相关属性如下表 7.35 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 7.35: 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
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

7.5.3 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 7.36 所示：

表 7.36: Path 运动模式		
运动模式	说明	板卡支持情况
直线插补	两轴直线插补（相对 / 绝对）	参见第一章
	三轴直线插补（相对 / 绝对）	参见第一章
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	参见第一章
圆弧插补	两轴圆弧插补（相对 / 绝对）	参见第一章
	三轴圆弧插补（相对 / 绝对）	参见第一章
螺旋插补	三轴及以上螺旋插补（相对 / 绝对），与板卡轴数有关	参见第一章
延迟命令	群组将延迟一段时间再执行下一路径	参见第一章
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	参见第一章

将上表中的运动模式分为如表 7.37 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 7.37: Path 运动相关指令			
运动模式	运动命令		说明
直线插补	<code>Abs2DLine</code>	<code>Rel2DLine</code>	2 轴绝对 / 相对直线插补
	<code>Abs3DLine</code>	<code>Rel3DLine</code>	3 轴绝对 / 相对直线插补
直接线性插补	<code>Abs2DDirect</code>	<code>Rel2DDirect</code>	2 轴绝对 / 相对直接线性插补
	<code>Abs3DDirect</code>	<code>Rel3DDirect</code>	3 轴绝对 / 相对直接线性插补
	<code>Abs4DDirect</code>	<code>Rel4DDirect</code>	4 轴绝对 / 相对直接线性插补
	<code>Abs5DDirect</code>	<code>Rel5DDirect</code>	5 轴绝对 / 相对直接线性插补
	<code>Abs6DDirect</code>	<code>Rel6DDirect</code>	6 轴绝对 / 相对直接线性插补
	<code>Abs7DDirect</code>	<code>Rel7DDirect</code>	7 轴绝对 / 相对直接线性插补
	<code>Abs8DDirect</code>	<code>Rel8DDirect</code>	8 轴绝对 / 相对直接线性插补
	<code>Abs2DArcCW</code>	<code>Rel2DArcCW</code>	2 轴绝对 / 相对顺时针圆弧插补
2 轴圆弧插补	<code>Abs2DArcCCW</code>	<code>Rel2DArcCCW</code>	2 轴绝对 / 相对逆时针圆弧插补
	<code>Abs2DArcCWAngle</code>	<code>Rel2DArcCWAngle</code>	2 轴绝对 / 相对顺时针角度圆弧插补
	<code>Abs2DArcCCWAngle</code>	<code>Rel2DArcCCWAngle</code>	2 轴绝对 / 相对逆时针角度圆弧插补
	<code>Abs2DArcCW_3P</code>	<code>Rel2DArcCW_3P</code>	2 轴绝对 / 相对顺时针 3 点圆弧插补
	<code>Abs2DArcCCW_3P</code>	<code>Abs2DArcCCW_3P</code>	2 轴绝对 / 相对逆时针 3 点圆弧插补

表 7.37: Path 运动相关指令

3 轴圆弧插补	Abs3DArcCW	Rel3DArcCW	3 轴绝对 / 相对顺时针圆弧插补
	Abs3DArcCCW	Rel3DArcCCW	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
螺旋插补	Abs3DHelixCW	Rel3DHelixCW	3 轴绝对 / 相对顺时针螺旋插补
	Abs3DHelixCCW	Rel3DHelixCCW	3 轴绝对 / 相对逆时针螺旋插补
	Abs4DHelixCW	Rel4DHelixCW	4 轴绝对 / 相对顺时针螺旋插补
	Abs4DHelixCCW	Rel4DHelixCCW	4 轴绝对 / 相对逆时针螺旋插补
	Abs5DHelixCW	Rel5DHelixCW	5 轴绝对 / 相对顺时针螺旋插补
	Abs5DHelixCCW	Rel5DHelixCCW	5 轴绝对 / 相对逆时针螺旋插补
	Abs6DHelixCW	Rel6DHelixCW	6 轴绝对 / 相对顺时针螺旋插补
	Abs6DHelixCCW	Rel6DHelixCCW	6 轴绝对 / 相对逆时针螺旋插补
	Abs7DHelixCW	Rel7DHelixCW	7 轴绝对 / 相对顺时针螺旋插补
	Abs7DHelixCCW	Rel7DHelixCCW	7 轴绝对 / 相对逆时针螺旋插补
	Abs8DHelixCW	Rel8DHelixCW	8 轴绝对 / 相对顺时针螺旋插补
	Abs8DHelixCCW	Rel8DHelixCCW	8 轴绝对 / 相对逆时针螺旋插补
	Abs3DHelixCWAngle	Rel3DHelixCWAngle	3 轴绝对 / 相对顺时针角度螺旋插补
	Abs3DHelixCCWAngle	Rel3DHelixCCWAngle	3 轴绝对 / 相对逆时针角度螺旋插补
	Abs4DHelixCWAngle	Rel4DHelixCWAngle	4 轴绝对 / 相对顺时针角度螺旋插补
	Abs4DHelixCCWAngle	Rel4DHelixCCWAngle	4 轴绝对 / 相对逆时针角度螺旋插补
	Abs5DHelixCWAngle	Rel5DHelixCWAngle	5 轴绝对 / 相对顺时针角度螺旋插补
	Abs5DHelixCCWAngle	Rel5DHelixCCWAngle	5 轴绝对 / 相对逆时针角度螺旋插补
	Abs6DHelixCWAngle	Rel6DHelixCWAngle	6 轴绝对 / 相对顺时针角度螺旋插补
	Abs6DHelixCCWAngle	Rel6DHelixCCWAngle	6 轴绝对 / 相对逆时针角度螺旋插补
	Abs7DHelixCWAngle	Rel7DHelixCWAngle	7 轴绝对 / 相对顺时针角度螺旋插补
	Abs7DHelixCCWAngle	Rel7DHelixCCWAngle	7 轴绝对 / 相对逆时针角度螺旋插补
	Abs8DHelixCWAngle	Rel8DHelixCWAngle	8 轴绝对 / 相对顺时针角度螺旋插补
	Abs8DHelixCCWAngle	Rel8DHelixCCWAngle	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 7.5.3.5 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。

7.5.3.1 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 7.37 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 7.5.6 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 7.5.5 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
CenPoint_DataArray	中心点坐标
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。

7.5.3.2 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。

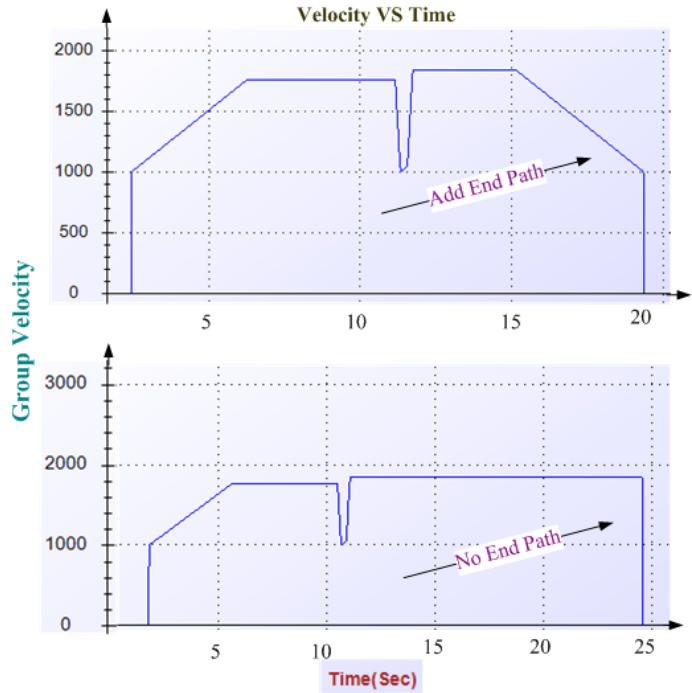
7.5.4 EndPath 的作用及影响

调用 7.5.3 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。

7.5.5 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 7.38：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等于 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。

7.5.5.1 BufferMode

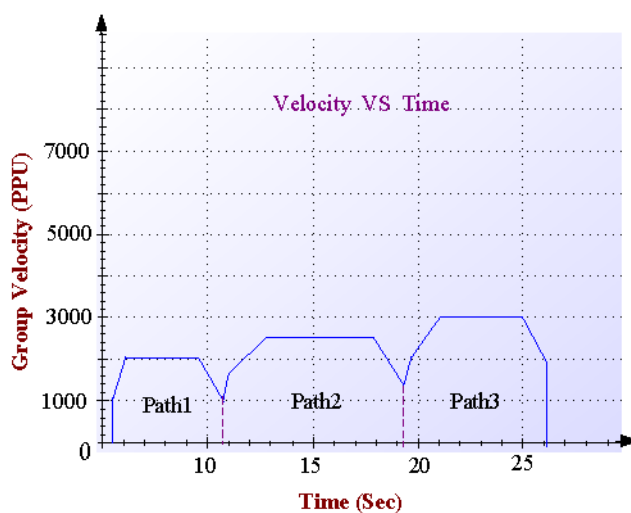
MoveMode	CFG_GpSFEnable	CFG_GpBldTime
1	无效	无效

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。

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：



7.5.5.2 Blending Mode

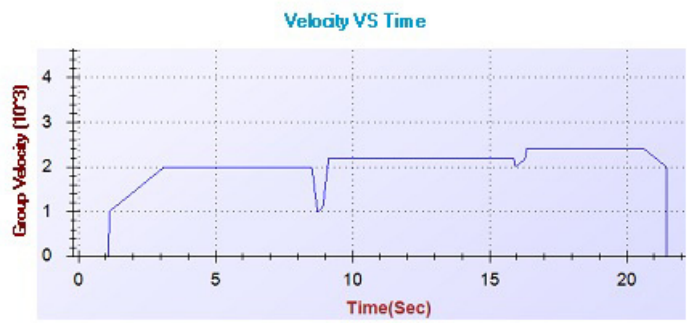
MoveMode	CFG_GpSFEnable	CFG_GpBldTime
0	Disable	大于 0

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 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：



7.5.5.3 Fly Mode

MoveMode	CFG_GpSFEnable	CFG_GpBldTime
0	Disable	等于 0

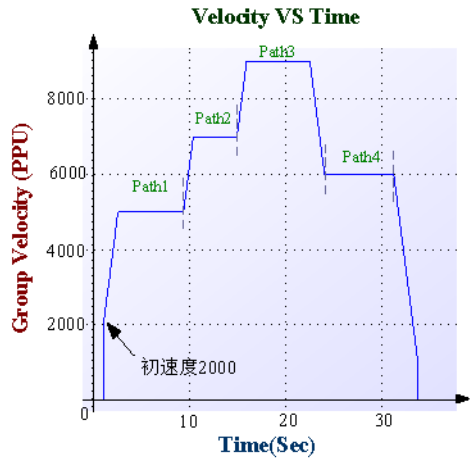
当启用速度交接模式，且设置速度交接时间等于零时，加载的 Path 将以 FlyMode 运行。该种模式下可开启速度前瞻功能。

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。

序号	命令	模式	运行速度	初速度	圆心0	圆心1	圆心2	终点0	终点1	终点2
0	Rel2DLine	0: Enable Blending	5000	2000	0	0	0	30000	25000	0
1	Rel2DLine	0: Enable Blending	7000	4000	0	0	0	25000	28000	0
2	Rel2DArcCW	0: Enable Blending	9000	3000	10000	10000	0	20000	0	0
3	Rel2DArcCWAngle	0: Enable Blending	6000	1000	20000	0	0	180	180	0
4	EndPath	0: Enable Blending	2000	500	0	0	0	0	0	0

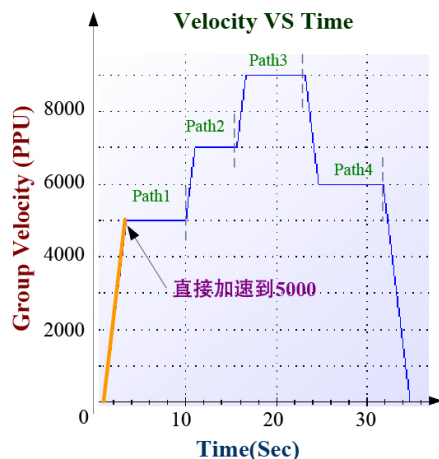
保存后调用 Acm_GpLoadPath 函数加载该 Path 文件，调用 Acm_GpMovePath 函数运行。如下图运行速度、初速度为表中设置的值



将表中的初速度修改如下：

序号	命令	模式	运行速度	初速度	圆心0	圆心1	圆心2	终点0	终点1	终点2	终点3
0	Rel2DLine	0: Enable Blending	5000	0	0	0	0	30000	25000	0	0
1	Rel2DLine	0: Enable Blending	7000	0	0	0	0	25000	28000	0	0
2	Rel2DArcCW	0: Enable Blending	9000	0	10000	10000	0	20000	0	0	0
3	Rel2DArcCWAngle	0: Enable Blending	6000	0	20000	0	0	180	180	0	0
4	EndPath	0: Enable Blending	2000	500	0	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

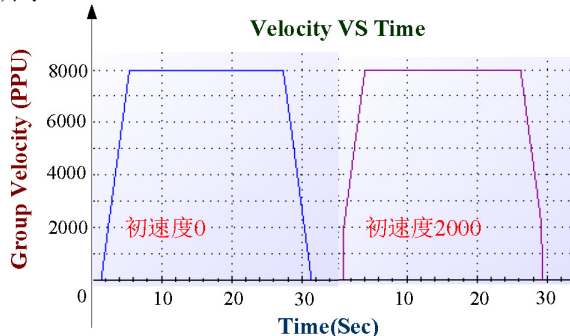
速度前瞻指根据移动距离，计算将要执行 Path 的运行速度。

当启用速度前瞻功能运行 Path 时，将不使用 Acm_GpAddPath 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。

序号	命令	模式	运行速度	初速度	圆心0	圆心1	圆心2	终点0	终点1	终点2
0	Bel2Line	1: Disable Blending	2000	1000	0	0	0	12000	12000	0
1	Bel30HelixCY	1: Disable Blending	2500	1500	10000	0	0	20000	0	10000
2	Bel20ArcCY	1: Disable Blending	3000	2000	10000	0	0	20000	0	0
3	EndPath	1: Disable Blending	1000	500	0	0	0	0	0	0

速度曲线如下图所示：



7.5.6 Path D0 功能指令

7.5.6.1 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。

PCI-1245、65 中各轴均有 OUT4, OUT5, OUT6, OUT7 四个 D0，并且 PCI-1265 中除了各轴上的 4 个 D0 外，还包含设备的 8 个 D0。用户可以将这些 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。

7.5.6.2 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	设定 PCI-1265 上的 D00~D07, PCI-1245 中请将此参数置 0。此值设定方式请见表 1 说明。
FL	请将此参数置 0。
EndPoint_DataArray	群组中各轴的 D0 设定, 此数组每个元素对应一个轴的 D0 设定, 方便运动过程中 D0 输出。例如 Group(X, Y, Z), 则 EndPoint_DataArray[0] 对应 X 轴的 D0 设定, EndPoint_DataArray[1] 对应 Y 轴的 D0 设定, EndPoint_DataArray[2] 对应 Z 轴的 D0 设定。数组中各元素 D0 设定方式请参考表 1 说明。
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:

表 7.39:				
Bit	24~31	16~23	8~15	0~7
Axis D0 设定, 通过 EndPoint_DataArray 参数传入	保留	设定 OUT0~OUT7 是否在 Path 中启用 D0 的开关控制功能。 0: 禁用 1: 启用 PCI-1245/65 上仅能设置 OUT4~OUT7	保留	打开 / 关闭 OUT0~OUT7 0: 关闭 1: 打开 PCI-1245/65 上仅能设置 OUT4~OUT7
Device 上的 D0 设定, 通过 FH 参数传入。	保留	设定 D00~D07 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 PCI-1265 上 D00~D07 打开 / 关闭 0: 关闭 1: 打开

7.5.6.3 Path D0 重点说明

由于 PCI-1245/65 上各轴的 OUT4~OUT7 是作为复用 D0, 其既可以作为普通 D0 使用, 也可以作为 CAMD0(OUT4)、CMP(OUT5)、SVON(OUT6)、ERC(OUT7) 的信号输出。所以, 如果需要在 Path 中使用这些 D0 的输出功能, 需要通过属性 CFG_AxGenDoEnable 将 OUT4~OUT7 设置为通用输出, 那么这些 D0 将不能再作为 CAMD0 等特殊输出信号使用。属性 CFG_GpBlendingTime 不可设置为非 0 值。

7.5.7 Path 的执行

通过上述两种方式加载 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。关于该函数的使用以及获取的状态参数详见附录。

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。关于该函数的使用详见附录。

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, 如果有群组正在执行路径, 则路径运动将停止。

7.5.8 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
Acm_GpPauseMotion	暂停群组运动命令
Acm_GpResumeMotion	恢复暂停的群组运动命令

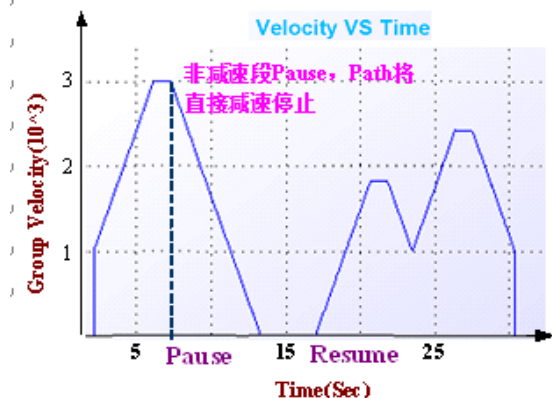
7.5.8.1 暂停 / 恢复在 BufferMode 中的使用

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。

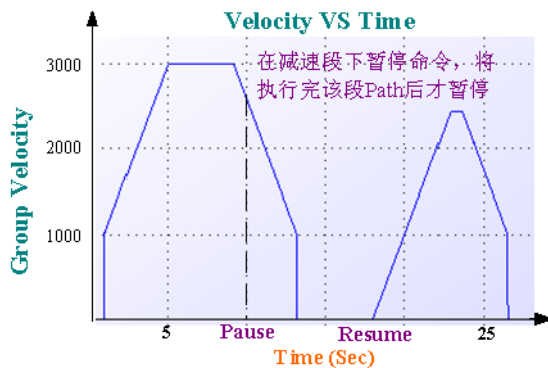
编辑 Path 如下，设置加速度、减速度均为 500。

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：

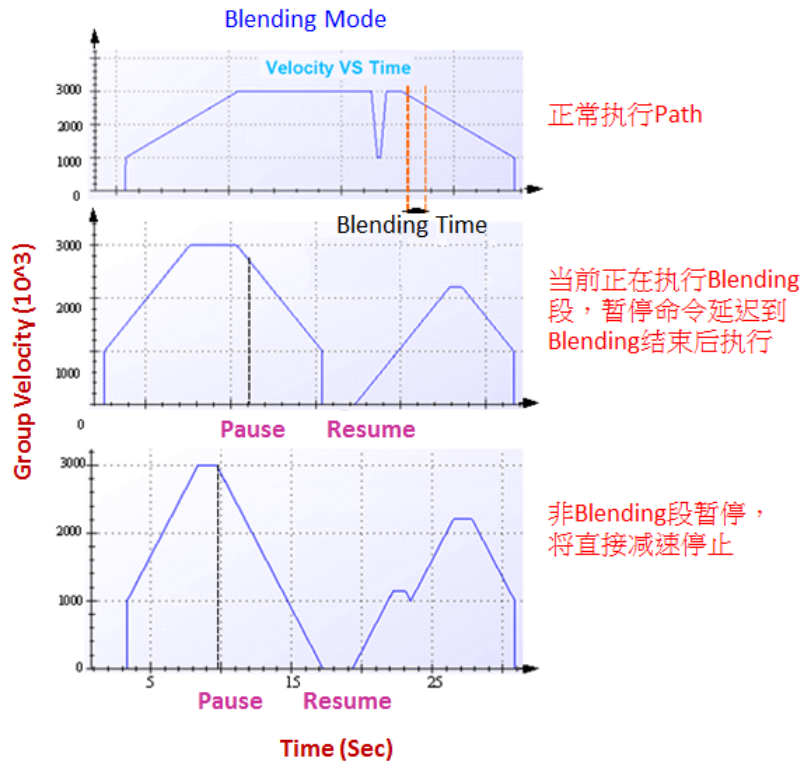


如果在减速段下暂停命令，则暂停命令在该段 Path 运行完成后才会起作用。如下运行上表中的 Path，速度曲线如下所示：



7.5.8.2 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。将 -- 表中 MoveMode 设为 Enable Blending，速度交接时间为 500ms，加速度、减速度均为 500，速度曲线如下图所示：



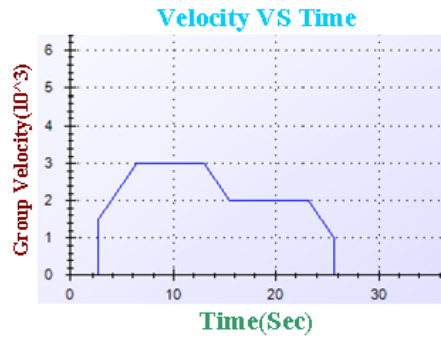
7.5.8.3 暂停 / 恢复在 FlyMode 中的使用

1. 禁用速度前瞻功能

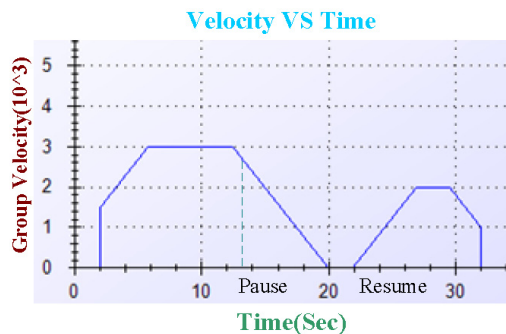
当禁用速度前瞻功能时，设置速度交接时间为 0，编辑 Path 如下表所示，加速度、减速度均为 500。

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：

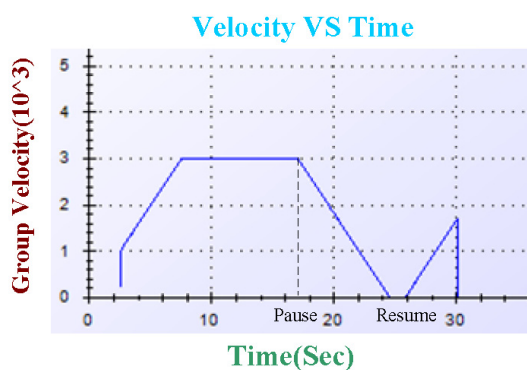


执行暂停和恢复功能的速度曲线如下图所示：



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：

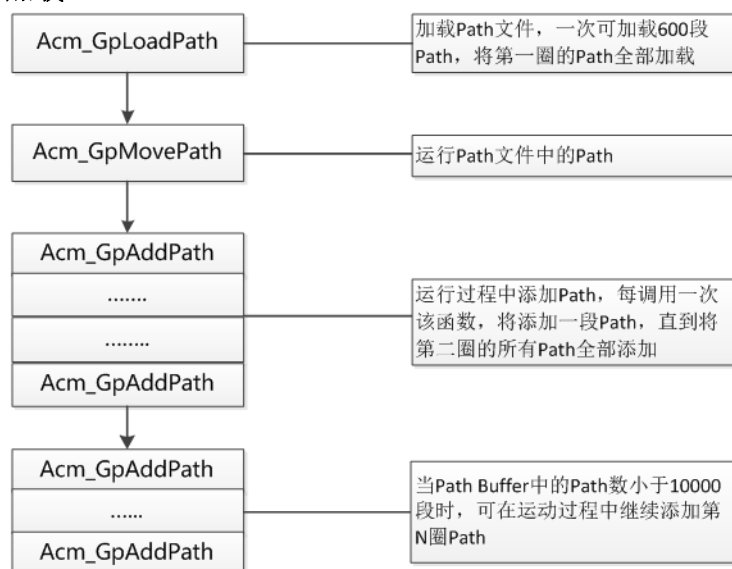


7.5.9 动态加载 Path

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 10000 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 10000 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 10000 段 Path，执行了 10 段 Path，将释放掉 10 段 Path 的缓冲。此时可继续 AddPath。

7.5.10 Path 运动流程图

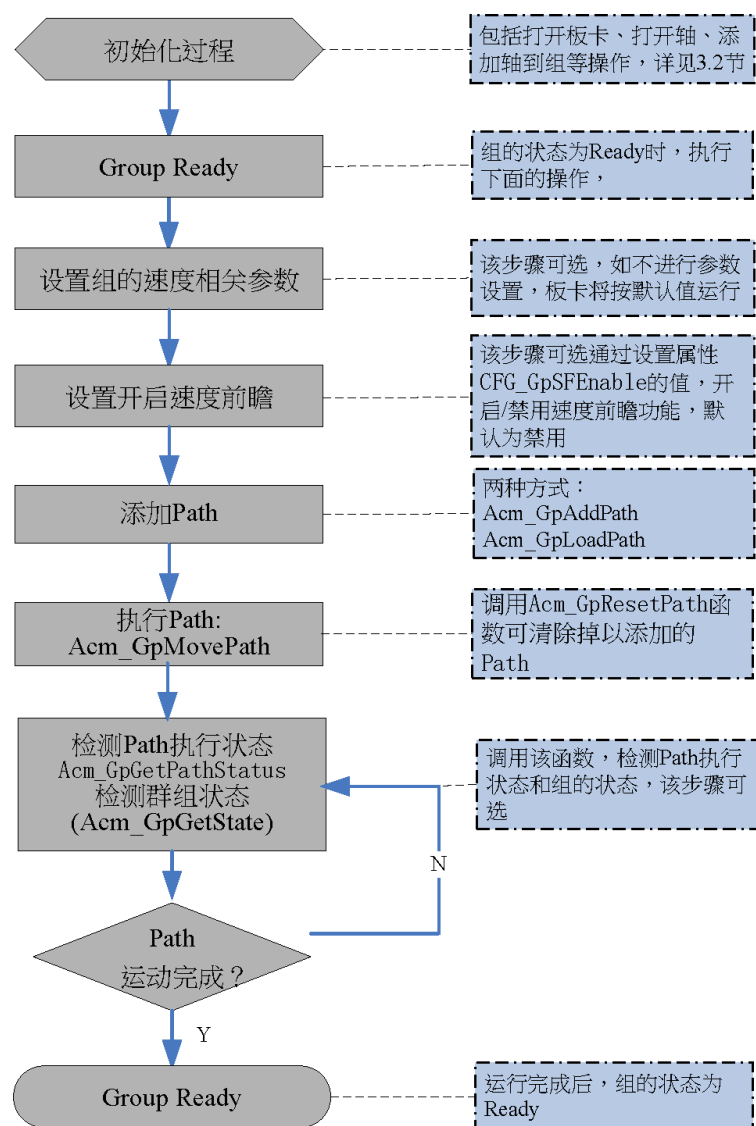


图 7.17: Path 运动流程图

7.5.11 Path 运动例程

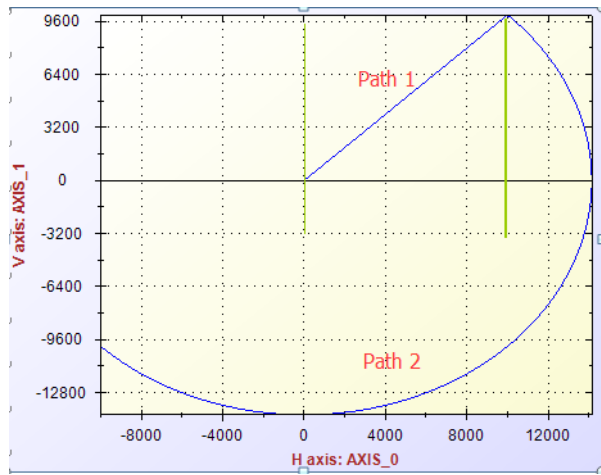
该例程实现 PCI-1245 执行 Path 运动，使用速度前瞻，添加的 Path 如下表所示：

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-1245 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
--- 初始化过程见 3.2 节 ---
// 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。

第 8 章

DIO 与 AI 控制

8.1 本章简介

本章主要介绍 DI/O 和 AI 的相关操作，以及板卡的其他重要功能。

8.2 DI/O 与 AI 控制

8.2.1 本节简介

本节介绍 DI/O 与 AI 的相关操作，其中只有 PCI-1265 板卡具有 AI 功能。

8.2.2 数字量输入输出

8.2.2.1 数字量输入输出相关函数与属性

访问数字量输入输出的相关函数与属性如下表所示：

数字量输入输出函数	说明
Acm_AxDoSetBit	设定指定通道的 D0 值
Acm_AxDoGetBit	获取指定通道的数字量输出 bit 值
Acm_AxDiGetBit	获取指定通道的 DI 值

访问数字量输入输出相关属性如下表所示：

数字量输入输出属性	说明
FT_DaqDiMaxChan	获取 DI 通道的最大个数
FT_DaqDoMaxChan	获取 D0 通道的最大个数
CFG_AxKillDec	DI Stop 的减速度

8.2.2.2 重点说明

运动控制卡的数字量输入输出以通道 (Channel) 数为单位，在设置 D0 及获取 DI 值时，需指定设置 / 获取哪一通道的值，不同板卡的通道数不同，可通过读取 FT_DaqDiMaxChan、FT_DaqDoMaxChan 属性值获取通道数。

8.2.2.3 例程

如下代码实现设置和获取 D0 的值，以及获取 DI 的值：

VC 部分代码：

```
HAND    m_Axishand; //0 轴的 handle
U32     Ret; // 函数返回值
U8      GetbitData;
U8      BitIn;
--- 初始化过程见 3.2 节 ---
for (int i=0;i<8;i++)
{Ret = Acm_AxDoGetBit(m_Axishand, i,&bitData);} // 获取 0 轴的 8 个 D0 值
    bitData[0] =1;
    Ret = Acm_AxDoSetBit(m_Axishand, 0, bitData[0]); // 设置 0 轴的通道 0 的 D0 值
    Ret = Acm_AxDiGetBit(m_Axishand, 0, &BitIn); // 获取 0 轴的通道 0 的 DI 值
    具体使用步骤请参考 DIO 例程。
```

8.2.3 访问模拟量输入

模拟量输入为 PCI-1265 板卡特有的功能。

8.2.3.1 模拟量输入相关函数与属性

模拟量输入相关函数如下表所示：

模拟量输入输出函数	说明
Acm_DaqAiGetRawData	获取模拟量输入值的二进制值
Acm_DaqAiGetVoltData	获取实际模拟量输入值

模拟量输入相关属性如下表所示：

模拟量输入相关函数	说明
FT_DaqAiRangeMap	获取支持的 AI 范围
FT_DaqAiMaxSingleChan	获取设备的最大单端 AI 通道个数
FT_DaqAiMaxDiffChan	获取设备的最大差分 AI 通道个数
FT_DaqAiResolution	获取设备的 AI 分辨率，单位为 bit
CFG_DaqAiChanType	配置 AI 通道类型
CFG_DaqAiRanges	设定通道的采值范围

8.2.3.2 重点说明

在 PCI-1265 中，总共有两个 AI 通道（通道 0 和 1），用户可设定 CFG_DaqAiChanType 属性设定通道类型为单端输入还是差分输入。PCI-1265 的最大差分 AI 通道个数为 1，最大单端 AI 通道个数为 2。因此，如果为差分输入，则 AI 值需从通道 1 中获取到。模拟量输入为设备上的输入，非轴上的模拟量输入，因此设置属性和调用 API 时，输入的 Handle 为设备的 Handle。

8.2.3.3 例程

如下代码实现设置和获取设备的 AI 相关属性值及 AI 的值

VC 部分代码如下：

```
HAND    m_Devhand; // 设备的 handle
U32     Ret; // 函数返回值
ULON    ulData;
U16     usAiData =0;
FLOAT   fVdata=0;
--- 初始化过程见 3.2 节 ---
Ret = Acm_GetU32Property(m_Devhand,CFG_DaqAiChanType,&ulData); // 获取 AI 通道类型
Ret = Acm_GetU32Property(m_Devhand,CFG_DaqAiRanges,&ulData); // 获取 AI 通道采值范围
ulData = 0; //AI 通道类型为单端输入
Ret = Acm_SetU32Property(m_Devhand,CFG_DaqAiChanType,ulData); // 设置 AI 通道类型
ulData = 0; // 设置采值范围为 +/- 10 V
Ret = Acm_SetU32Property(m_Devhand,CFG_DaqAiRanges,ulData); // 设置 AI 通道采值范围
Ret = Acm_DaqAiGetRawData(m_Devhand,0, &usAiData); // 获取 0 通道模拟量输入二进制值
Ret = Acm_DaqAiGetVoltData(m_Devhand,0, &fVdata); // 获取 0 通道实际模拟量输入值
具体使用步骤可参考 AIO 范例。
```

8.3 比较功能

8.3.1 本节简介

本节主要介绍触发比较功能相关应用。

8.3.2 触发比较功能

8.3.2.1 触发比较功能相关函数与属性

触发比较功能相关函数如下表所示：

比较功能相关函数	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

触发比较功能相关属性如下表所示：

比较功能相关属性	说明
FT_AxCompareMap	获取轴支持的比较特性
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	获取 / 设置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

8.3.2.2 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 100000 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。

8.3.2.3 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;
--- 初始化过程见 3.2 节 ---
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 1000
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
具体使用步骤可参考 Compare 例程
```

8.4 锁存功能

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。

8.4.1 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。

8.4.1.1 锁存功能相关函数与属性

锁存功能相关函数如下表所示：

锁存功能相关函数	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
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

锁存功能相关属性如下表所示：

锁存功能相关属性	说明
FT_AxLatchMap	获取轴支持的锁存特性
FT_DevLTCBufMaxCount	获取 Latch Buffer 中所支持的最大资料数量
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_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

8.4.1.2 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。

8.4.1.3 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
--- 初始化过程见 3.2 节 ---
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
具体使用步骤可参考 Latch 例程。
```

8.4.2 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。

8.4.2.1 连续锁存功能相关函数与属性

连续锁存功能相关函数如下表所示：

连续锁存功能相关函数	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

连续锁存功能相关属性如下表所示：

连续锁存功能相关属性	说明
FT_DevLTCBufMaxCount	获取 Latch Buffer 中所支持的最大资料数量
CFG_AxLatchLogic	设置 / 获取锁存信号的逻辑准位
CFG_AxLatchBufEdge	设置 / 获取 Latch Buffer 信号的触发沿
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_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_AxLatchBufAxisID	设定 / 获取被锁存数据的轴的 ID (仅 PCI-1285 支持该属性，通过该属性可设定一个轴锁存另一轴的数据)
CFG_AxLatchBufSource	设定 / 读取某轴的 Latch Buffer 中的资料类型

8.4.2.2 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。

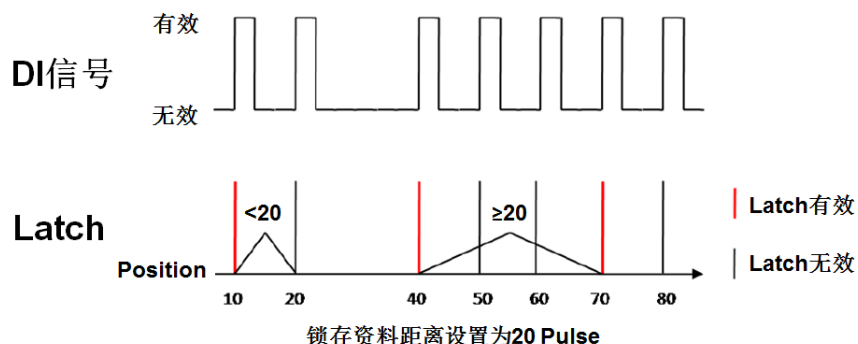
当连续锁存功能被禁用时，资料不会被记录到 Buffer 中，但不影响单笔资料的 Latch (前提是 Latch 功能被打开)。当此功能被打开时，每次进行 Latch 时，就会将资料记录到 Buffer 中。

■ 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。

8.4.2.3 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
--- 初始化过程见 3.2 节 ---
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_SetU32Property(m_Axishand, CFG_AxLatchBufEnable, 1);
// 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[m_CurAxis]); // 清除 Buffer 中锁存的数据
具体使用步骤可参考 LatchBuffer 例程。
```

8.5 硬件安全

8.5.1 本节简介

本节主要介绍硬件安全相关功能，包括限位、报警、停止功能。

8.5.2 限位

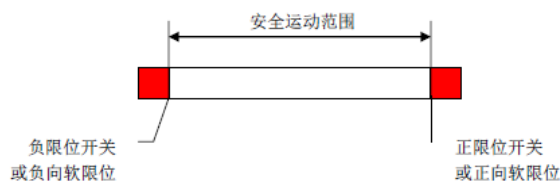
8.5.2.1 限位功能相关属性

限位功能相关属性如下表所示：

属性	说明
FT_AxSwMelMap	获取运动轴支持的软件负方向限位特性
FT_AxSwPelMap	获取运动轴支持的软件正方向限位特性
CFG_AxEIReact	设置 / 获取 EL 信号的反应模式
CFG_AxEILogic	设置 / 获取硬件限位信号的逻辑准位
CFG_AxEIEnable	设置 / 获取硬件限位功能启用 / 禁用
CFG_AxSwMelEnable	启用 / 禁用负方向软件限位功能
CFG_AxSwPelEnable	启用 / 禁用正方向软件限位功能
CFG_AxSwMelReact	设置 / 获取负方向软件限位的反应模式
CFG_AxSwPelReact	设置 / 获取正方向软件限位的反应模式
CFG_AxSwMelValue	设置 / 获取负方向软件限位的值
CFG_AxSwPelValue	设置 / 获取正方向软件限位的值

8.5.2.2 重点说明

运动控制器能通过硬件限位开关或者设置软件限位限制各轴的运动范围。
如下所示：



在使用轴的限位功能时，首先需要启用限位功能，如设置 `CFG_AxEIEnable` 属性的值启用硬件限位功能，设置 `CFG_AxSwMeIEnable`、`CFG_AxSwPeIEnable` 属性值启用软件限位功能。软限位和限位开关可以同时使用，当软限位触发时也会置起限位触发标志限位触发以后有两种停止模式：减速停止和立即停止，可通过设置属性的值 `CFG_AxEIReact` 设置硬极限停止模式，设置 `CFG_AxSwMeIValue`、`CFG_AxSwPeIValue` 属性的值设置软限位停止模式。

8.5.3 报警

8.5.3.1 报警功能相关属性

报警功能相关属性如下表所示：

属性	说明
<code>FT_AxAlmMap</code>	获取该运动轴支持的报警特性
<code>CFG_AxAlmLogic</code>	设置 / 获取报警信号的有效逻辑电平
<code>CFG_AxAlmEnable</code>	启用 / 禁用运动报警功能
<code>CFG_AxAlmReact</code>	设置 / 获取接收 ALARM 信号时的停止模式

8.5.3.2 重点说明

运动控制器提供专用的数字量驱动报警信号输入接口。当驱动报警信号输入后，运动控制器将会根据设置的报警信号停止模式停止运行，同时报警信号的逻辑电平发生变化。

轴的停止模式有立即停止和减速停止，可通过 `CFG_AxAlmReact` 属性设置。

报警信号的逻辑电平默认为低电平，当发生报警时，报警信号的逻辑电平将变成高电平。报警信号的逻辑电平通过属性 `CFG_AxAlmLogic` 设置。

当发生报警时，轴的状态将变为 `ErrorStop`，此时将不能执行任何运动操作，需要确定引起驱动器报警的原因，并加以改正，然后调用错误状态重置函数，重置轴的状态，才能继续执行其他操作。

8.5.4 停止

8.5.4.1 停止功能相关函数与属性

停止功能相关函数如下表所示：

函数	说明
Acm_AxStopDec	命令轴减速停止
Acm_AxStopEmg	命令轴立刻停止
Acm_AxStopDecEx	下达停止命令时可指定减速度
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

停止功能相关属性如下表所示：

属性	说明
FT_AxIN1Map	IN1 触发停止功能特性
FT_AxIN2Map	IN2 触发停止功能特性
FT_AxIN4Map	IN4 触发停止功能特性
FT_AxIN5Map	IN5 触发停止功能特性
CFG_AxIN1StopEnable	启用 / 禁用 IN1 触发停止功能
CFG_AxIN1StopReact	设定 / 获取 IN1 触发时的停止模式
CFG_AxIN1StopLogic	设定 / 获取 IN1 触发停止功能的逻辑准位
CFG_AxIN2StopEnable	启用 / 禁用 IN2 触发停止功能
CFG_AxIN2StopReact	设定 / 获取 IN2 触发时的停止模式
CFG_AxIN2StopLogic	设定 / 获取 IN2 触发停止功能的逻辑准位
CFG_AxIN4StopEnable	启用 / 禁用 IN4 触发停止功能
CFG_AxIN4StopReact	设定 / 获取 IN4 触发时的停止模式
CFG_AxIN4StopLogic	设定 / 获取 IN4 触发停止功能的逻辑准位
CFG_AxIN5StopEnable	启用 / 禁用 IN5 触发停止功能
CFG_AxIN5StopReact	设定 / 获取 IN5 触发时的停止模式
CFG_AxIN5StopLogic	设定 / 获取 IN5 触发停止功能的逻辑准位

8.5.4.2 重点说明

根据停止功能的函数和属性列表，轴的停止方式可通过调用函数停止和硬件触发停止两种。

当采用硬件触发停止时，可将操作轴与硬件停止输入引脚连接，设置停止模式及触发停止的逻辑准卫，即可实现该轴的停止操作。

附录 A

API 列表

A.1 设备特性属性

类型	方法 / 事件	PCI-1245 PCI-1265 PCI-1285	PCI-1245V PCI-1245E PCI-1285E	PCI-1245L
一般	Acm_GetAvailableDevs	✓	✓	✓
	Acm_GetErrorMessage	✓	✓	✓
方法	Acm_DevOpen	✓	✓	✓
	Acm_DevClose	✓	✓	✓
	Acm_DevLoadConfig	✓	✓	✓
	Acm_GetProperty	✓	✓	✓
	Acm_SetProperty	✓	✓	✓
	Acm_GetLastError	✓	✓	✓
	Acm_CheckMotionEvent	✓	✓	✓
	Acm_EnableMotionEvent	✓	✓	✓
	Acm_DevReadEEPROM_Ex	✓	✓	✓
	Acm_DevWriteEEPROM_Ex	✓	✓	✓
设备	Acm_DevDownloadCAMTable	PCI-1285 不支持	×	×
	Acm_DevLoadCAMTableFile	PCI-1285 不支持	×	×
	Acm_DevConfigCAMTable	PCI-1285 不支持	×	×
	Acm_DevMDaqConfig	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqGetConfig	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqStart	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqStop	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqReset	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqGetStatus	PCI-1285 不支持	PCI-1285E 不支持	×
	Acm_DevMDaqGetData	PCI-1285 不支持	PCI-1285E 不支持	×
事件	EVT_AX_MOTION_DONE	✓	✓	✓
	EVT_AX_COMPARED	✓	×	×
	EVT_AX_LATCH	✓	×	×
	EVT_AX_ERROR	✓	✓	✓
	EVT_AX_VH_START	✓	✓	✓
	EVT_AX_VH_END	✓	✓	✓
	EVT_GPn_MOTION_DONE	✓	✓	✓
	EVT_GPn_VH_START	✓	✓	✓
	EVT_GPn_VH_END	✓	✓	✓
DAQ	Acm_DaqDiGetByte	PCI-1245/85 不支持	×	×
	Acm_DaqDiGetBit	PCI-1245/85 不支持	×	×
	Acm_DaqDoSetByte	PCI-1245/85 不支持	×	×
	Acm_DaqDoSetBit	PCI-1245/85 不支持	×	×
	Acm_DaqDoGetByte	PCI-1245/85 不支持	×	×
	Acm_DaqDoGetBit	PCI-1245/85 不支持	×	×
	Acm_DaqDiGetBytes	×	×	×
	Acm_DaqDoSetBytes	×	×	×
	Acm_DaqDoGetBytes	×	×	×
	Acm_DaqAiGetRawData	PCI-1245/85 不支持	×	×
模拟量 输入 / 输出	Acm_DaqAiGetVoltData	PCI-1245/85 不支持	×	×
	Acm_DaqAiGetCurrData	×	×	×

轴	系统	Acm_AxOpen	✓	✓	✓
		Acm_AxClose	✓	✓	✓
		Acm_AxResetError	✓	✓	✓
	运动 I/O	Acm_AxSetSvOn	✓	✓	✓
		Acm_AxGetMotionIO	✓	✓	✓
	运动状态	Acm_AxGetMotionStatus	✓	✓	✓
		Acm_AxGetState	✓	✓	✓
		Acm_AxStopDec	✓	✓	✓
	停止	Acm_AxStopEmg	✓	✓	✓
		Acm_AxStopDecEx	✓	✓	✓
	速度运动	Acm_AxMoveVel	✓	✓	✓
		Acm_AxChangeVel	✓	✓	✓
		Acm_AxChangeVelByRate	✓	✓	✓
		Acm_AxChangeVelEx	✓	✓	✓
		Acm_AxChangeVelExByRate	✓	✓	✓
		Acm_AxGetCmdVelocity	✓	✓	✓
	点到点运动	Acm_AxMoveRel	✓	✓	✓
		Acm_AxMoveAbs	✓	✓	✓
		Acm_AxChangePos	✓	✓	✓
		Acm_AxMoveImpose	✓	×	✓
	同时运动	Acm_AxSimStartSuspendAbs	✓	×	✓
		Acm_AxSimStartSuspendRel	✓	×	✓
		Acm_AxSimStartSuspendVel	✓	×	✓
		Acm_AxSimStart	✓	×	✓
		Acm_AxSimStop	✓	×	✓
	返原点	Acm_AxHome	✓	✓	✓
	位置 / 计数器	Acm_AxSetCmdPosition	✓	✓	✓
		Acm_AxGetCmdPosition	✓	✓	✓
		Acm_AxSetActualPosition	✓	✓	✓
		Acm_AxGetActualPosition	✓	✓	✓
	比较	Acm_AxSetCmpData	✓	PCI-1245E/85E 不支持	×
		Acm_AxSetCmpTable	✓	PCI-1245E/85E 不支持	×
		Acm_AxSetCmpAuto	✓	PCI-1245E/85E 不支持	×
		Acm_AxGetCmpData	✓	PCI-1245E/85E 不支持	×
	锁存	Acm_AxGetLatchData	✓	PCI-1245E/85E 不支持	×
		Acm_AxTriggerLatch	✓	PCI-1245E/85E 不支持	×
		Acm_AxResetLatch	✓	PCI-1245E/85E 不支持	×
		Acm_AxGetLatchFlag	✓	PCI-1245E/85E 不支持	×
		Acm_AxReadLatchBuffer	✓	×	×
		Acm_AxResetLatchBuffer	✓	×	×
		Acm_AxGetLatchBufferStatus	✓	×	×
	Aux/ Gen 输出	Acm_AxDoSetBit	✓	✓	✓
		Acm_AxDoGetBit	✓	✓	✓
		Acm_AxDiGetBit	✓	✓	✓
	外部驱动	Acm_AxSetExtDrive	✓	✓	✓

轴	应用	Acm_AxCamInAx	PCI-1285 不支持	×	×
		Acm_AxGearInAx	√	√	×
		Acm_AxGantryInAx	√	×	×
		Acm_AxPhaseAx	√	√	×
		Acm_AxTangentInGp	√	×	×
系统		Acm_GpAddAxis	√	√	√
		Acm_GpRemAxis	√	√	√
		Acm_GpClose	√	√	√
		Acm_GpResetError	√	√	√
运动状态		Acm_GpGetState	√	√	√
速度		Acm_GpChangeVel	√	×	×
		Acm_GpChangeVelByRate	√	×	×
		Acm_GpGetCmdVel	√	√	√
运动停止		Acm_GpStopDec	√	√	√
		Acm_GpStopEmg	√	√	√
群组 插补运动		Acm_GpMoveLinearRel	√	√	√
		Acm_GpMoveLinearAbs	√	√	√
		Acm_GpMoveCircularRel	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveCircularAbs	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveCircularRel_3P	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveCircularAbs_3P	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveCircularRel_Angle	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveCircularAbs_Angle	√	PCI-1245E/85E 不支持	×
		Acm_GpMoveDirectAbs	√	√	√
		Acm_GpMoveDirectRel	√	√	√
		Acm_GpMoveHelixRel	√	×	×
		Acm_GpMoveHelixAbs	√	×	×
		Acm_GpMoveHelixRel_3P	√	×	×
		Acm_GpMoveHelixAbs_3P	√	×	×
		Acm_GpAddPath	√	√	×
		Acm_GpResetPath	√	√	×
		Acm_GpLoadPath	√	√	×
		Acm_GpUnloadPath	√	√	×
		Acm_GpMovePath	√	√	×
		Acm_GpGetPathStatus	√	√	×
		Acm_GpMoveSelPath	√	√	×
		Acm_GpGetPathIndexStatus	√	√	×
暂停 & 恢复		Acm_GpPauseMotion	√	√	×
		Acm_GpResumeMotion	√	√	×

A. 2 函数列表

A. 2.1 通用 API

Acm_GetAvailableDevs

函数原型	U32 Acm_GetAvailableDevs (DEVLIST *DeviceList, U32 MaxEntries, PU32OutEntries)		
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. 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	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	由 Acm_DevOpen 产生的设备句柄
ConfigPath	PI8	IN	指针指向保存配置文件路径的字符串

Acm_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 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
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 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
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 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
Value	PF64	OUT	属性的返回值

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 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	I32	IN	设置的属性值

Acm_SetF64Property

函数原型	U32 Acm_SeF64tProperty(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 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	需要设置的属性 ID
Value	F64	IN	设置的属性值

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_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	由 Acm_DevOpen 产生的设备句柄
AxEnableEvtArray	PU32	IN	Array[n]，返回每个轴的中断事件状态，n 表示运动设备的轴个数
GpEnableEvtArray	PU32	IN	Array[n]：返回每个群组的中断事件状态
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	由 Acm_DevOpen 产生的设备句柄
AxEnableEvtArray	PU32	IN	Array[n]，启用每个轴的中断事件，n 表示运动设备的轴个数。
GpEnableEvtArray	PU32	IN	Array[n]：启用每个群组的中断事件，n 为 GroupID。可从 PAR_GpGroupID 属性获取。
AxArrayElements	U32	IN	AxEvtStatusArray 元素个数
GpArrayElements	U32	IN	GpEvtStatusArray 元素个数
Millisecond	U32	IN	每次检查以毫秒为单位制定超时值

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 和 PCI-1265 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



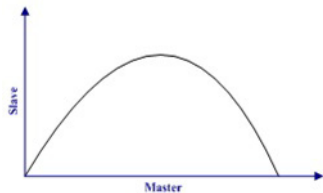
■ MasterAbsolute 和 SlaveAbsolute

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。

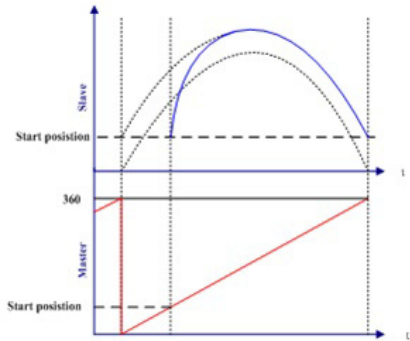
相对：当相对凸轮系统设置后，则表示系统不会补偿在同步过程中控制值和从值之间的任何偏移。比如，实用程序中剖面图如下所示：

初始 CAM 曲线如下图所示：

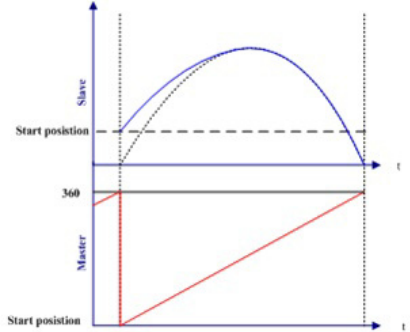
注解



- 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	由 Acm_DevOpen 产生的设备句柄

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	由 Acm_DevOpen 产生的设备句柄
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	由 Acm_DevOpen 产生的设备句柄
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)

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)
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: Ready 1: Waiting Trigger 2: Started

Acm_DevMDaqGetData

函数原型	U32 Acm_DevMDaqGetData(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

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	仅 PCI-1265 有 8 个 DI，DiPort 应为 0，其他板卡不支持设备 DI 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
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	仅 PCI-1265 有 8 个 DI, DiChannel 为 0 ~ 7, 其他板卡不支持设备 DI 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
DiChannel	U16	IN	DI 通道
BitData	PU8	OUT	返回的位数据, 值为 0 或 1

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	仅 PCI-1265 有 8 个 DO，DoPort 为 0，其他板卡不支持设备 DO 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
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	仅 PCI-1265 有 8 个 DO，DoChannel 为 0 ~ 7，其他板卡不支持设备 DO 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
DoChannel	U16	IN	DO 通道
BitData	U8	IN	需要设置的值，设置的值为 0 或 1

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	仅 PCI-1265 有 8 个 DO，DoPort 为 0，其他板卡不支持设备 DO 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
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 D0 通道的位值		
返回值	错误代码		
注 解	仅 PCI-1265 有 8 个 D0，DoChannel 为 0 ~ 7，其他板卡不支持设备 D0 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
DoChannel	U16	IN	D0 通道 0 ~ 7
BitData	PU8	Out	返回值 0 或 1

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	仅 PCI-1265 支持 AI 功能，共两个 AI 通道：0 和 1 模拟量输入有单端输入和差分输入两种，可通过 CFG_DaqAiChanType 属性设定 当设定为单端输入时，用户可通过 0 和 1 通道获取 AI 值 当设定为差分输入事，用户只能通过通道 1 获取 AI 值 详见 CFG_DaqAiChanType 属性说明		
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：0 或 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	仅 PCI-1265 支持 AI 功能，共两个 AI 通道：0 和 1 模拟量输入有单端输入和差分输入两种，可通过 CFG_DaqAiChanType 属性设定		
	当设定为单端输入时，用户可通过 0 和 1 通道获取 AI 值 当设定为差分输入事，用户只能通过通道 1 获取 AI 值 详见 CFG_DaqAiChanType 属性说明		
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：0 或 1
AiData	PF32	Out	指向返回的模拟量值的指针

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-1265 物理轴编号为：0、1、2、3、4、5 PCI-1285 系列物理轴编号为：0、1、2、3、4、5、6、7 PCI-1245 系列物理轴编号为：0、1、2、3		
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 IO

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

			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）
Status	PU32	OUT	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

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

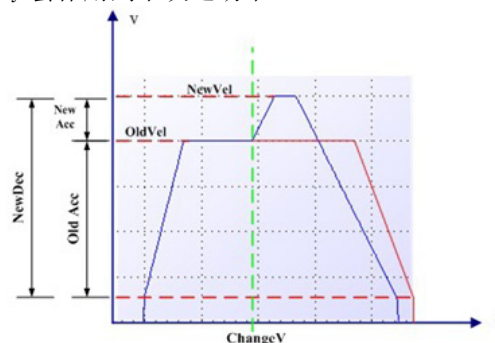
函数原型	U32 Acm_AxChangeVel (HAND AxisHandle, F64 NewVelocity)		
说明	当轴在运动过程中，命令轴改变速度		
返回值	错误代码		
注解	速度曲线由以下属性决定：PAR_AxVelLow、NewVelocity、PAR_AxAcc、PAR_AxDec 和 PAR_AxJerk。NewVelocity 的范围：0 ~ CFG_AxMaxVel。 若此命令成功下达，在下次运动之前若未重新设定速度，则 NewVelocity 会用到下次运动中		
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 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		
注解	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		



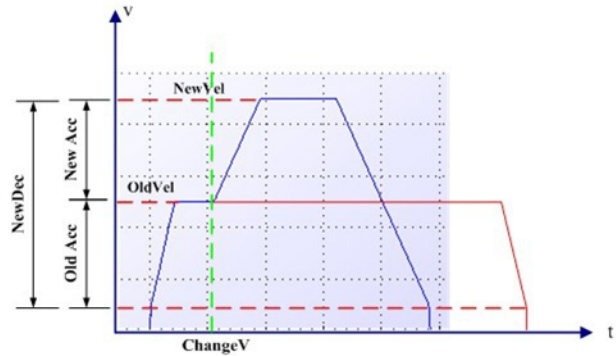
函数原型参数			
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
NewVelocity	F64	IN	新的速度值，单位：PPU/s
NewAcc	F64	IN	新的加速度值，单位：PPU/s ²
NewDec	F64	IN	新的减速度值，单位：PPU/s ²

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		

$\text{NewVel} = \text{OldVel} * \text{Rate} * 0.01$ 。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。 $\text{NewVel} = \text{OldVel} * \text{Rate} * 0.01$
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运动的运行速度
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

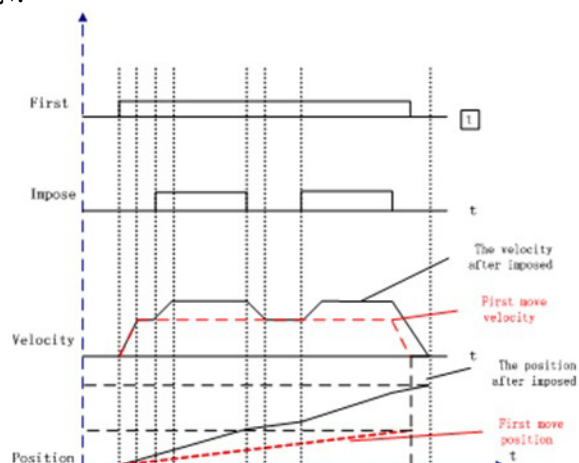
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
NewDistance	F64	IN	新的相对距离（单位 = PPU）

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）

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，典型原点运动的 16 种类型组成扩展原点		
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

Dir	U32	IN	0: 正方向 1: 负方向
-----	-----	----	------------------

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			
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		
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		
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			
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_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			

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）

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 Aux/Gen 输出

Acm_AxDoSetBit

函数原型	U32 Acm_AxDoSetBit (HAND AxisHandle, U16 DoChannel, U8 BitData)
说明	设定指定通道的 DO 值
返回值	错误代码
注解	如果用户想要使用该共用 DO 功能，必须首先将 CFG_AxGenDoEnable 设置为 GEN_DO_EN。启用 CFG_AxGenDoEnable 时，CamDo、Erc 和 Compare 功能将自动禁用。由于两种功能使用同一输出针脚（OUT4 ~ OUT7），因此 Acm_AxSetCmpData、Acm_AxSetCmpAuto 和 Acm_AxSetCmpTable 将不能生成触发脉冲。

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
DoChannel	U16	IN	数字量输出通道（4 ~ 7）
BitData	U8	IN	DO 值：0 或 1

Acm_AxDoGetBit

函数原型	U32 Acm_AxDoGetBit (HAND AxisHandle, U16 DoChannel, PU8 BitData)
说明	获取指定通道的数字量输出 bit 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
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄
DoChannel	U16	IN	数字量输出通道（4 ~ 7）
BitData	PU8	OUT	DO 值：0 或 1

Acm_AxDiGetBit

函数原型	U32 Acm_AxDiGetBit (HAND AxisHandle, U16 DiChannel, PU8 BitData)
说明	获取指定通道的 DI 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
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

DiChannel	U16	IN	数字量输入通道 (0 ~ 3)
BitData	PU8	OUT	DI 值: 0 或 1

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	U32Acm_AxJog (HANDAxisHandle,U16Direction)		
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-1245 和 PCI-1265 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

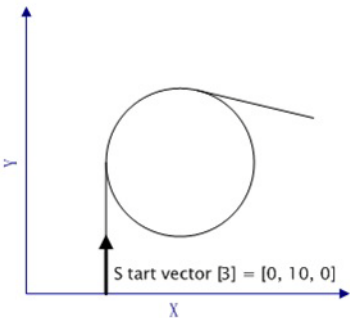
A. 4. 14龙门 / 切线跟随

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

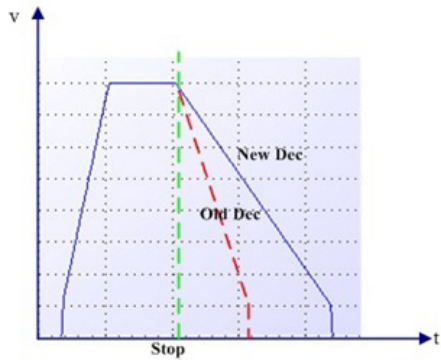
名称	类型	IN 或 OUT	说明
AxisHandle	HAND	IN	来自 Acm_AxOpen 的轴句柄

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. 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-1245 系列最多有 2 个群组，PCI-1265 最多有 3 个群组，PCI-1285 系列最多 4 个群组。 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	<p>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。</p> <p>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。</p> <p>PCI-1245/PCI-1265 最多只支持 3 个轴线性插补，PCI-1245V/PCI-1245E 最多只支持 2 个轴线性插补</p>		
----	--	--	--

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	<p>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。</p> <p>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。</p> <p>PCI-1245/1245V/1265 最多只支持 3 个轴线性插补，PCI-1245V/PCI-1245E 最多只支持 2 个轴线性插补</p>		
----	--	--	--

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_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。PCI-1245/1245V/1265 最多只支持 2 个轴 ARC 插补。如果群组中的轴个数为 3，那么用户可通过 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动，其它轴无响应。如果群组中的轴个数大于 3，将发生错误		
----	---	--	--

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。PCI-1245/1245V/1265 最多只支持 2 个轴 ARC 插补。如果群组中的轴个数为 3，那么用户可通过 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动，其它轴无响应。如果群组中的轴个数大于 3，将发生错误		
----	--	--	--

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。 PCI-1245/1245V/1265 最多只支持 2 个轴 ARC 插补。如果群组中的轴个数为 3，那么用户可通过 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动，其它轴无响应。如果群组中的轴个数大于 3，将发生错误。		
----	---	--	--

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。 PCI-1245/1245V/1265 最多只支持 2 个轴 ARC 插补。如果群组中的轴个数为 3，那么用户可通过 PAR_GpRefPlane 选择群组中的两个轴进行 ARC 插补运动，其它轴无响应。如果群组中的轴个数大于 3，将发生错误。		
----	---	--	--

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_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	<p>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。</p> <p>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。</p> <p>PCI-1245 最多只支持 4 个轴直接插补，PCI-1245V/PCI-1245E 最多只支持 2 个，PCI-1265 最多只支持 6 个。PCI-1285 最多支持 8 个，PCI-1285E 最多支持 2 个。</p>
----	---

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	<p>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。</p> <p>PCI-1245 最多只支持 4 个轴直接插补，PCI-1245V/PCI-1245E 最多只支持 2 个，PCI-1265 最多只支持 6 个。PCI-1285 最多支持 8 个，PCI-1285E 最多支持 2 个。</p>
----	--

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_GpChangeVel

函数原型	U32 Acm_GpChangeVel (HAND GroupHandle, F64 NewVelocity)
说明	命令群组在插补运动时改变速度
返回值	错误代码
注解	
函数原型参数	

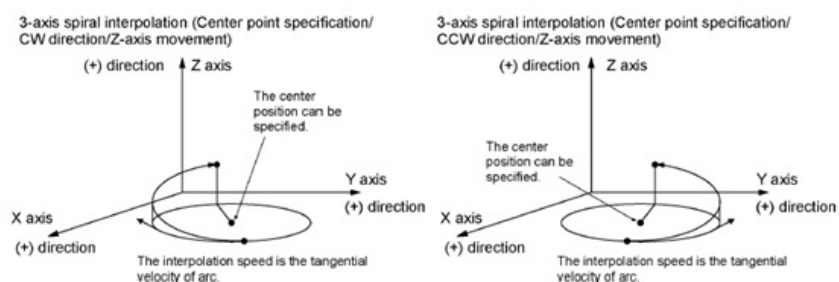
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
NewVelocity	F64	IN	新速度。(单位: PPU/s)

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_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			
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）

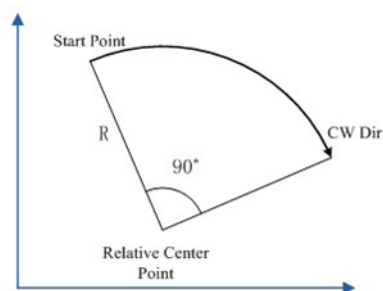
Acm_GpMoveHelixRel_3P

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			
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_GpMoveCircularRel_Angle

函数原型	U32 Acm_GpMoveCircularRel_Angle (HAND GroupHandle, PF64 CenterArray, U16 Degree, PU32 ArrayElements, I16 Direction)		
说明	通过相对圆心坐标，旋转角度以及方向进行圆弧插补		
返回值	错误代码		

注解

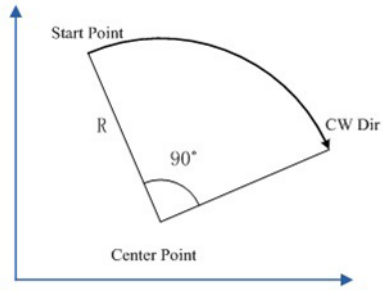


函数原型参数			
名称	类型	IN 或 OUT	说明
GroupHandle	HAND	IN	来自 Acm_GpAddaxis 的群组句柄
CenterArray	PF64	IN	圆心相对于起点的相对距离
Degree	U16	IN	旋转角度，范围：0~360
pArrayElements	PU32	IN	轴数
Direction	I16	IN	方向： 0: CW-Dir 1: CCW-Dir

Acm_GpMoveCircularAbs_Angle

函数原型	U32 Acm_GpMoveCircularAbs_Angle (HAND GroupHandle, PF64 CenterArray, U16 Degree, PU32 ArrayElements, I16 Direction)		
说明	通过相对圆心坐标，旋转角度以及方向进行圆弧插补		
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			
1: CCW_Dir			

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。新速度仅对此次运动有效。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

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	
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_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 PathHandle, 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	获取系统路径缓存中指定索引路径的状态
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：非混合； 1：混合。
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			
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 属性列表

类型	属性	PCI-1245 PCI-1265 PCI-1285	PCI-1245V PCI-1245E PCI-1285E	PCI-1245L
设备	特性	FT_DevIpoTypeMap	✓	✓
		FT_DevAxisCount	✓	✓
		FT_DevFunctionMap	✓	✓
		FT_DevOverflowCntr	✓	✓
		FT_DevMDAQTypeMap	✓	×
		FT_DevMDAQTrigMap	✓	×
		FT_DevMDAQMaxChan	✓	×
		FT_DevMDAQMaxBufCount	✓	×
	配置	CFG_DevBoardID	✓	✓
		CFG_DevBaseAddress	✓	✓
		CFG_DevInterrupt	✓	✓
		CFG_DevBusNumber	✓	✓
		CFG_DevSlotNumber	✓	✓
		CFG_DevDriverVersion	✓	✓
		CFG_DevDllVersion	✓	✓
		CFG_DevFwVersion	✓	✓
		CFG_DevCPLDVersion	✓	✓
		CFG_DevEmgLogic	✓	✓
DAQ	特性	FT_DaqDiMaxChan	仅 PCI-1265 支持	×
		FT_DaqDoMaxChan	仅 PCI-1265 支持	×
		FT_DaqAiRangeMap	仅 PCI-1265 支持	×
		FT_DaqAiMaxSingleChan	仅 PCI-1265 支持	×
		FT_DaqAiMaxDiffChan	仅 PCI-1265 支持	×
		FT_DaqAiResolution	仅 PCI-1265 支持	×
	配置	CFG_DaqAiChanType	仅 PCI-1265 支持	×
		CFG_DaqAiRanges	仅 PCI-1265 支持	×
轴	系统	FT_AxFunctionMap	✓	✓
		CFG_AxPPU	✓	✓
		CFG_AxPhyID	✓	✓
	速度模式	FT_AxMaxVel	✓	✓
		FT_AxMaxAcc	✓	✓
		FT_AxMaxDec	✓	✓
		FT_AxMaxJerk	✓	✓
		CFG_AxMaxVel	✓	✓
		CFG_AxMaxAcc	✓	✓
		CFG_AxMaxDec	✓	✓
		CFG_AxMaxJerk	✓	✓
		PAR_AxVelLow	✓	✓
		PAR_AxVelHigh	✓	✓
		PAR_AxAcc	✓	✓
		PAR_AxDec	✓	✓
		PAR_AxJerk	✓	✓

轴	脉冲输入	FT_AxPulseInMap	✓	✓	✓
		FT_AxPulseInModeMap	✓	✓	✓
		CFG_AxPulseInMode	✓	✓	✓
		CFG_AxPulseInLogic	✓	✓	✓
		CFG_AxPulseInMaxFreq	✓	✓	✓
	脉冲输出	FT_AxPulseOutMap	✓	✓	✓
		FT_AxPulseOutModeMap	✓	✓	✓
		CFG_AxPulseOutMode	✓	✓	✓
		CFG_AxPulseOutReverse	✓	✓	✓
	报警	FT_AxAlmMap	✓	✓	✓
		CFG_AxAlmEnable	✓	✓	✓
		CFG_AxAlmLogic	✓	✓	✓
		CFG_AxAlmReact	✓	✓	✓
	到位	FT_AxInpMap	✓	✓	✓
		CFG_AxInpEnable	✓	✓	✓
		CFG_AxInpLogic	✓	✓	✓
	ERC	FT_AxErcMap	✓	✓	✓
		FT_AxErcEnableModeMap	✓	✓	✓
		CFG_AxErcLogic	✓	✓	✓
		CFG_AxErcEnableMode	✓	✓	✓
	SD	FT_AxSdMap	×	×	×
	硬件限位	FT_AxEIMap	✓	✓	✓
		CFG_AxEIEnable	✓	✓	✓
		CFG_AxEILogic	✓	✓	✓
		CFG_AxEIReact	✓	✓	✓
	软件限位	FT_AxSwMeIMap	✓	✓	✓
		FT_AxSwPeIMap	✓	✓	✓
		CFG_AxSwMeIEnable	✓	✓	✓
		CFG_AxSwPeIEnable	✓	✓	✓
		CFG_AxSwMeIReact	✓	✓	✓
		CFG_AxSwPeIReact	✓	✓	✓
		CFG_AxSwMeIValue	✓	✓	✓
		CFG_AxSwPeIValue	✓	✓	✓
	返回原点	FT_AxHomeMap	✓	✓	✓
		FT_AxHomeModeMap	✓	✓	✓
		CFG_AxOrgLogic	✓	✓	✓
		CFG_AxOrgReact	✓	✓	✓
		CFG_AxEzLogic	✓	✓	✓
		CFG_AxHomeResetEnable	✓	✓	✓
		PAR_AxHomeVelLow	✓	✓	✓
		PAR_AxHomeVelHigh	✓	✓	✓
		PAR_AxHomeAcc	✓	✓	✓
		PAR_AxHomeDec	✓	✓	✓
		PAR_AxHomeJerk	✓	✓	✓
		PAR_AxHomeCrossDistance	✓	✓	✓
		PAR_AxHomeExSwitchMode	✓	✓	✓

轴	背隙	FT_AxBacklashMap	✓	✓	✓
		CFG_AxBacklashEnable	✓	✓	✓
		CFG_AxBacklashPulses	✓	✓	✓
		CFG_AxBacklashVel	✓	✓	✓
	比较	FT_AxCompareMap	✓	✓	×
		CFG_AxCmpSrc	✓	×	×
		CFG_AxCmpMethod	✓	×	×
		CFG_AxCmpPulseMode	✓	×	×
		CFG_AxCmpPulseLogic	✓	×	×
		CFG_AxCmpPulseWidth	✓	×	×
		CFG_AxCmpEnable	✓	×	×
		CFG_AxCmpPulseMode	✓	✓	✓
	锁存	FT_AxLatchMap	✓	✓	×
		FT_DevLTCBufMaxCount	✓	✓	×
		CFG_AxLatchLogic	✓	×	×
		CFG_AxLatchEnable	✓	×	×
		CFG_AxLatchBufEnable	✓	✓	×
		CFG_AxLatchBufMinDist	✓	✓	×
		CFG_AxLatchBufEventNum	✓	✓	×
		CFG_AxLatchBufSource	✓	✓	×
	CAM DO	FT_AxCamDOMap	✓	✓	×
		CFG_AxCamDOEnable	✓	×	×
		CFG_AxCamDOLOLimit	✓	×	×
		CFG_AxCamDOHiLimit	✓	×	×
		CFG_AxCamDOCmpSrc	✓	×	×
		CFG_AxCamDOLogic	✓	×	×
	外部驱动	FT_AxExtDriveMap	✓	✓	✓
		FT_AxExtMasterSrcMap	✓	✓	✓
		FT_AxJogMap	✓	✓	✓
		CFG_AxJogVLTime	✓	✓	✓
		CFG_AxJogVelLow	✓	✓	✓
		CFG_AxJogVelHigh	✓	✓	✓
		CFG_AxJogAcc	✓	✓	✓
		CFG_AxJogDec	✓	✓	✓
		CFG_AxJogJerk	✓	✓	✓
		CFG_AxExtMasterSrc	✓	✓	✓
		CFG_AxExtSelEnable	×	×	×
		CFG_AxExtPulseNum	✓	✓	✓
		CFG_AxExtPulseInMode	✓	✓	✓
		CFG_AxExtPresetNum	✓	✓	✓
	Aux/Gen输出	FT_AxGenDOMap	✓	✓	✓
		FT_AxGenDIMap	✓	✓	✓
		CFG_AxGenDoEnable	✓	✓	✓
	轴单圈脉冲数	CFG_AxModuleRange	✓	×	×
	同步启停	FT_AxSimStartSourceMap	✓	✓	✓
		CFG_AxSimStartSource	✓	✓	✓

轴	DI 停止	FT_AxIN1Map	√	√	√
		FT_AxIN2Map	√	√	√
		FT_AxIN4Map	√	√	√
		FT_AxIN5Map	√	√	√
		CFG_AxIN1StopEnable	√	√	√
		CFG_AxIN1StopReact	√	√	√
		CFG_AxIN1StopLogic	√	√	√
		CFG_AxIN2StopEnable	√	√	√
		CFG_AxIN2StopReact	√	√	√
		CFG_AxIN2StopLogic	√	√	√
		CFG_AxIN4StopEnable	√	√	√
		CFG_AxIN4StopReact	√	√	√
		CFG_AxIN4StopLogic	√	√	√
		CFG_AxIN5StopEnable	√	√	√
		CFG_AxIN5StopReact	√	√	√
		CFG_AxIN5StopLogic	√	√	√
群组	系统	PAR_GpGroupID	√	√	√
		CFG_GpAxesInGroup	√	√	√
	应用	CFG_GpSFEnable	√	×	×
		CFG_GpBldTime	√	×	×
		PAR_GpRefPlane	√	×	×
	速度 模式	PAR_GpVelLow	√	√	√
		PAR_GpVelHigh	√	√	√
		PAR_GpAcc	√	√	√
		PAR_GpDec	√	√	√
		PAR_GpJerk	√	√	√

B.2 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：

设备的插补类型如下表所示，获取的属性值为 32 位整型字，数据的每一位代表一种插补类型，当该位为 1，表示板卡支持该插补功能，为 0 则表示不支持。

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;  // 设备的 Handle
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_DevAxisCount

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：

设备的特性，只读。

设备支持的功能如下表所示，获取的值为 32 位整型，每一位代表一个功能。当该位的值为 1 时，表示板卡支持该功能，当值为 0 时，表示不支持。

位	说明	宏定义（定义在头文件 AdvMotDrv.h）
0	运动	DEV_FUNC_MOT
1	DI（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DI
2	DO（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DO
3	AI（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_AI
4	AO（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_AO
5	定时器	DEV_FUNC_TMR
6	计数器	DEV_FUNC_CNT
7	DAQ DI（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DAQDI
8	DAQ DO（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DAQDO
9	DAQ AI（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DAQAI
10	DAQ AO（PCI-1245/1245V/1245E 不支持）	DEV_FUNC_DAQAO
11	Emg	DEV_FUNC_EMG
12	DEV_FUNC_MDAQ	DEV_FUNC_MDAQ

例程：如下获取设备支持的功能

```
HAND      m_Devhand;  // 设备的 Handle
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	2147483647
含义	位置计数器的最大值		

注解：

设备特性，值为 2147483647。

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	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;  // 设备的 Handle
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)
{ // 支持理论位置与实际位置的偏差 }
```

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;  // 设备的 Handle
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。PCI-1245/1245V/1245E/1265 中所支持的最大通道数为 4。

FT_DevMDAQMaxBufCount

数据类型	R/W	属性 ID	默认值
U32	R	9	–
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。在 PCI-1245/1245V/1245E/1265 中每个 MotionDAQ 通道最多记录 2000 笔资料。

B.3 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	板卡配置属性，只读
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_DevFwVersion

数据类型	R/W	属性 ID	默认值
char*	R	214	–
含义	获取固件版本		

注解：

格式为 1.0.0.1。

CFG_DevCPLDVersion

数据类型	R/W	属性 ID	默认值
char*	R	218	–
含义	获取设备的 CPLD 版本		

注解：

格式为 1.0.0.1。

CFG_DevEmgLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	220	低电平
含义	设定紧急停止信号的逻辑电平		

注解：

当设置的属性值为 1 时，表示紧急停止信号为高电平有效，否则低电平有效。

当设置的属性值为 1 时，表示紧急停止信号为高电平有效，否则低电平有效

值	说明
0	低电平（默认值）
1	高电平

B. 4 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。

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。

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：

板卡支持的 AI 范围如下表所示，获取的属性值为 32 位整型，每一位表示的 AI 范围如下，当该位值为 1 时表示支持该 AI 范围，值为 0 表示不支持。

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
16	0 ~ 20 mA	DAQ_AI_0MA_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; // 设备的 Handle
```

```
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_DaqAiMaxSingleChan

数据类型	R/W	属性 ID	默认值
U32	R	54	PCI1265 值为 2
含义	获取设备的最大单端 AI 通道个数		

注解：
只读。

FT_DaqAiMaxDiffChan

数据类型	R/W	属性 ID	默认值
U32	R	55	PCI-1265 值为 1
含义	获取设备的最大差分 AI 通道个数		

注解：
只读。

FT_DaqAiResolution

数据类型	R/W	属性 ID	默认值
U32	R	56	–
含义	获取设备的 AI 分辨率，单位为 bit		

注解：
只读。

B. 5 DAQ 配置属性

CFG_DaqAiChanType

数据类型	R/W	属性 ID	默认值
U32	R/W	251	–
含义	AI 通道类型 0: 单端 1: 差分		

注解：
在 PCI1265 中，总共有两个 AI 通道，用户可设定通道类型为单端输入还是差分输入，如果为差分输入，则 AI 值需从通道 1 中获取到。

CFG_DaqAiRanges

数据类型	R/W	属性 ID	默认值
U32	R/W	252	–
含义	设定通道采值范围		

注解：

在 PCI1265 中，仅支持 $\pm 10\text{V}$ 的采值范围。

设置、获取的属性值代表的范围如下：

值	描述	宏定义（头文件 AdvMotDrv.h 中定义）
0x0	$\pm 10\text{ V}$	CFG_DAQ_AI_NEG_10V_TO_10V
0x1	$\pm 5\text{ V}$	CFG_DAQ_AI_NEG_5V_TO_5V
0x2	$\pm 2.5\text{ V}$	CFG_DAQ_AI_NEG_2500MV_TO_2500MV
0x3	$\pm 1.25\text{ V}$	CFG_DAQ_AI_NEG_1250MV_TO_1250MV
0x4	$\pm 0.625\text{V}$	CFG_DAQ_AI_NEG_625MV_TO_625MV
0x5	$\pm 1\text{V}$	CFG_DAQ_AI_NEG_1V_TO_1V
0x6	$\pm 0.5\text{V}$	CFG_DAQ_AI_NEG_500MV_TO_500MV
0x7	$\pm 0.15\text{V}$	CFG_DAQ_AI_NEG_150MV_TO_150MV
0x8	$0 \sim 10\text{V}$	CFG_DAQ_AI_NEG_0_TO_10V
0x9	$0 \sim 500\text{ mA}$	CFG_DAQ_AI_NEG_0_TO_500mV
0x10	$0 \sim 20\text{mA}$	CFG_DAQ_AI_0MA_TO_20MA
0x11	$4 \sim 20\text{mA}$	CFG_DAQ_AI_4MA_TO_20MA
0x12	$\pm 20\text{mA}$	CFG_DAQ_AI_NEG_20MA_TO_20MA

例程：如下设置、获取板卡 AI 范围

```
HAND      m_Devhand;  // 设备的 Handle
```

```
U32       AIRangeMap; // 获取的属性值
```

```
AIRangeMap = CFG_DAQ_AI_NEG_10V_TO_10V; //AI 范围为  $\pm 10\text{V}$ 
```

```
Acm_SetU32Property(m_Devhand, CFG_DaqAiRanges, AIRangeMap); // 设置 AI 范围
```

```
Acm_GetU32Property(m_Devhand, CFG_DaqAiRanges, &AIRangeMap); // 获取 AI 范围
```

```
If (AIRangeMap & CFG_DAQ_AI_NEG_10V_TO_10V)
```

```
{ //AI 的范围为  $\pm 10\text{V}$ }
```

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 1：支持 0：不支持		

注解：

获取的属性值为 32 位整型字，每位代表的功能如下表所示，当该位值为 1 时，表示轴支持该功能：

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
18	IN1 停止	AX_FUNC_IN1_STOP
19	IN2 停止	AX_FUNC_IN2_STOP
20	IN3 停止	AX_FUNC_IN3_STOP
21	IN4 停止	AX_FUNC_IN4_STOP
22	IN5 停止	AX_FUNC_IN5_STOP
23~31	未定义	

例程：如下获取轴支持的功能

```
HAND      m_Axhand;  // 轴的 Handle
```

```
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	5, 000, 000
含义	获取轴支持的最大速度（单位 PPU/S）		

FT_AxMaxAcc

数据类型	R/W	属性 ID	默认值
F64	R	303	500, 000, 000
含义	获取轴支持的最大加速度（单位 PPU/S ² ）		

FT_AxMaxDec

数据类型	R/W	属性 ID	默认值
F64	R	304	500, 000, 000
含义	获取轴支持的最大减速度（单位 PPU/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 PPU/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	—
含义	获取该运动设备支持的脉冲输入特性		

注解：

运动设备支持的脉冲输入特性如下表所示。获取的属性值为 32 位整型字，每位代表是否支持该种输入特性，当该位为 1 时，表示支持，0 表示不支持。

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	只读
含义	获取轴支持的脉冲输入模式		

注解：

轴的脉冲输入模式如下表所示。获取的属性值为 32 位整型字，每位代表是否支持该种输入模式，当该位为 1 时，表示支持，为 0 表示不支持。

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 ~ 31	未定义	

例程：获取轴脉冲输入模式

```
HAND      m_Axhand;  // 轴的 Handle
```

```
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	—
含义	获取该运动设备支持的脉冲输出特性		

注解：

运动设备脉冲输出特性如下表所示：

位	说明
0	模式

1~31	未定义
------	-----

FT_AxPulseOutModeMap

数据类型	R/W	属性 ID	默认值
U32	R	309	—
含义	获取该运动设备支持的脉冲输出模式		

注解：

运动设备支持的脉冲输出模式如下表所示，获取的属性值为 32 位整型字，每位代表是否支持脉冲输出模式，该位为 1 表示支持，为 0 表示不支持。

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	

例程：获取轴支持的脉冲输入模式

```
HAND      m_Axhand;  // 轴的 Handle
```

```
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：

轴的报警特性如下表所示：

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：

轴的到位特性如小表所示：

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	—
含义	获取该运动轴支持的 ERC 特性		

注解：

轴的报警清除信号如下表所示：

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	—
含义	获取该运动轴支持的 ERC 模式		

注解：

轴的 ERC 模式如下表所示：

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 减速（SD）

FT_AxSdMap

数据类型	R/W	属性 ID	默认值
U32	R	316	—
含义	获取该运动轴支持的减速特性		

注解:

轴的减速特性如下表所示，所有板卡均不支持减速特性。

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

B. 6. 9 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特性		

注解:

轴的限位特性如下表所示:

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

B. 6. 10 软件限位

FT_AxSwEIMap

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:

轴的软件负方向限位特性如下表所示:

位	说明
0	启用
1	逻辑
2	反应
3~31	未定义

FT_AxSwPEIMap

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:

轴的软件正方向限位特性如下表所示:

位	说明
0	启用
1	逻辑
2	反应

3~31	未定义
------	-----

B. 6. 11 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：

轴的原点相关特性如下表所示：

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：

轴的回原点方式如下表所示：

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; // 轴的 Handle
```

```
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. 12 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：

轴的背隙补偿特性如下表所示：

位	说明
0	启用
1	值
2~31	未定义

B. 6. 13 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：

轴的比较特性如下表所示：

位	说明
0	启用
1	逻辑
2	CmpSrc
3	CmpMethod
4	CmpPulseMode
5	CmpPulseWidth
6~31	未定义

B. 6. 14 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：

轴的锁存特性如下表所示：

位	说明
0	启用
1	逻辑
2~31	未定义

B. 6. 15 Cam DO

FT_AxCamDMap

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:

轴 CAM DO 特性如下表所示:

位	说明
0	启用
1	CamDOLogic
2	CamDOCmpSrc
3	CamDOLoLimit
4	CamDOHiLimit
5	CamDOMode
6	CamDODir
7~31	未定义

B. 6. 16 外部驱动

FT_AxExtDriveMap

数据类型	R/W	属性 ID	默认值
U32	R	327	—
含义	获取轴支持的外部驱动特性		

注解:

轴的外部驱动特性如下表所示:

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	—
含义	获取轴支持的外部源		

注解:

轴的外部驱动源如下表所示:

位	说明
0	轴 0
1	轴 1
2	轴 2
3	轴 3
4~31	未定义

B. 6. 17 外部驱动

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:

轴的 Jog 特性如下表所示:

位	说明
0	Jog Speed 0: 不支持, 1: 支持
1	JogVltime 0: 不支持, 1: 支持
2	Soft jog 0: 不支持, 1: 支持

B. 6. 18 Aux/Gen 输出

FT_AxGenDMap

数据类型	R/W	属性 ID	默认值
U32	R	329	-
含义	获取轴支持的通用输出 OUT4~OUT7		

注解:

轴的通用输出如下表所示:

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	-
含义	获取轴支持的通用输入 IN1, IN2, IN4, IN5		

注解:

轴的通用输入如下表所示:

位	说明
0	IN1/LTC
1	IN2/RDY
2	IN4/JOG+
3	IN5/JOG-
4~31	未定义

B. 6. 19 同步启停

FT_AxSimStartSourceMap

数据类型	R/W	属性 ID	默认值
U32	R	331	—
含义	轴支持的同步启停模式		

注解：

轴的同步启停模式如下表所示：

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. 20 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：

IN1 触发停止功能特性如下表所示：

值	说明
0	启用
1	逻辑
2	反应

FT_AxIN2Map

数据类型	R/W	属性 ID	默认值
U32	R	334	—
含义	IN2 触发停止功能特性		

注解：

IN2 触发停止功能特性如下表所示：

值	说明
0	启用
1	逻辑
2	反应

FT_AxIN4Map

数据类型	R/W	属性 ID	默认值
U32	R	336	—
含义	IN4 触发停止功能特性		

注解：

IN4 触发停止功能特性如下表所示：

值	说明
0	启用
1	逻辑
2	反应

FT_AxIN5Map

数据类型	R/W	属性 ID	默认值
U32	R	337	只读
含义	IN5 触发停止功能特性		

注解：

IN5 触发停止功能特性如下表所示：

值	说明
0	启用
1	逻辑
2	反应

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：

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:

获取值的含义如下表所示:

值	说明
0	0- 轴
1	1- 轴
2	2- 轴
3	3- 轴
4	4- 轴
5	5- 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	5000000
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	50000000
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	50000000
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
0	1XAB
1	2XAB
2	4XAB(默认值)
3	CCW/CW

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

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	500 KHz
1	1 MHz(默认值)
2	2 MHz
3	4 MHz

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
1	OUT/DIR
2	OUT/DIR, OUT 负逻辑
4	OUT/DIR, DIR 负逻辑
8	OUT/DIR, OUT&DIR 负逻辑
16	CW/CCW(默认值)
32	CW/CCW, CW&CCW 负逻辑
256	CW/CCW, OUT 负逻辑
512	CW/CCW, DIR 负逻辑

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	Disable(默认值)
1	Enable

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:

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_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:

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）

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:

设置、获取的值说明如下表所示：

值	说明
0	电机立刻停止
1	电机减速，然后停止（默认值）

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:

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_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:

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）

B. 7. 7 ERC

CFG_AxErcLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	566	1
含义	设置 / 获取 ERC 信号的有效逻辑电平		

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）

CFG_AxErcEnableMode

数据类型	R/W	属性 ID	默认值
U32	R/W	569	默认值为禁用
含义	设置 / 获取 ERC 输出模式或禁用 ERC 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	返回原点后 ERC 输出
2	EMG/ALM/EL 激活时 ERC 输出（不支持）
3	返回原点后或 EMG/ALM/EL 激活时 ERC 输出（不支持）

B. 7.8 硬件限位

CFG_AxEIEnable

数据类型	R/W	属性 ID	含义	默认值
U32	R/W	574	启用 / 获取硬件限位功能	1

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：

设置、获取的值说明如下表所示：

值	说明
0	低准位（默认值）
1	高准位

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：

设置、获取的值说明如下表所示：

值	说明
0	电机立即停止（默认值）
1	电机减速，然后停止

B. 7. 9 软件限位

CFG_AxSwMelEnable

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:

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_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:

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_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:

设置、获取的值说明如下表所示:

值	说明
0	电机立即停止
1	电机减速, 然后停止 (默认值)

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:

设置、获取的值说明如下表所示:

值	说明
0	电机立即停止
1	电机减速, 然后停止 (默认值)

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. 10 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:

设置、获取的值说明如下表所示:

值	说明
0	低准位 (默认值)
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:

设置、获取的值说明如下表所示:

值	说明
0	低准位 (默认值)
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:

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_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：

设置、获取的值说明如下表所示：

值	说明
0	立即停止
1	减速停止（默认值）

B. 7. 11 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：

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_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	1000
含义	启用 / 禁用背隙补偿速度该速度是在原运行速度上进行叠加		

B. 7. 12 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

CFG_AxCmpMethod

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）（不支持）

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）

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)

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

CFG_AxCmpPulseMode

数据类型	R/W	属性 ID	默认值
U32	R/W	605	0
含义	设置 / 获取轴比较陌生		

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

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。

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)

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:

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

B. 7. 14 Aux、Gen 输出

CFG_AxGenDoEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	610	0
含义	启用 / 禁用轴的通用 DO 功能		

注解：

启用 CFG_AxGenDoEnable 后，属性 CFG_AxCmpEnable、CFG_AxCamDoEnable 和 CFG_AxErcEnableMode 会自动禁用。函数 Acm_AxSetCmpData、Acm_AxSetCmpTable 和 Acm_AxSetCmpAuto 将不能输出信号。

B. 7. 15 外部驱动

CFG_AxJogVlTime

数据类型	R/W	属性 ID	默认值
U32	R/W	692	5000
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, $2^{32}-1$]

CFG_AxJogVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	695	2000
含义	Jog 执行时的低速度。单位：PPU/S		

注解：

范围：[0, MaxVel]

CFG_AxJogVelHigh

数据类型	R/W	属性 ID	默认值
F64	R/W	696	8000
含义	Jog 执行时的运行速度。单位：PPU/S		

注解：

范围：[0, MaxVel]

CFG_AxJogAcc

数据类型	R/W	属性 ID	默认值
F64	R/W	697	10000
含义	Jog 执行时的加速度。单位：PPU/S ²		

注解：

范围：[0, MaxAcc]

CFG_AxJogDec

数据类型	R/W	属性 ID	默认值
F64	R/W	698	10000
含义	Jog 执行时的减速度。单位：PPU/S ²		

注解：

范围：[0, MaxDec]

CFG_AxJogJerk

数据类型	R/W	属性 ID	默认值
F64	R/W	699	0
含义	Jog 执行时的速度曲线类型		

注解：

只支持 T 型曲线。

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_AxExtMasterSrc

数据类型	R/W	属性 ID	默认值
U32	R/W	611	0
含义	设置 / 获取外部驱动的输入引脚		

注解：

仅支持 0。

设置、获取的值说明如下表所示：

值	说明
0	轴 0（默认值）
1	轴 1（不支持）
2	轴 2（不支持）
3	轴 3（不支持）

CFG_AxExtSelEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	612	0
含义	当采用外部驱动时，通过数字输入通道提供驱动轴选项		

注解：

仅支持 0。

值	说明
0	禁用（默认值）
1	启用（不支持）

CFG_AxExtPulseNum

数据类型	R/W	属性 ID	默认值
U32	R/W	613	1
含义	当轴的外部驱动模式为 MPG 且 A/B 或 B/A 相位信号触发时，设置理论脉冲个数		

注解：

默认值为 1，改值需大于 0。

CFG_AxExtPulseInMode

数据类型	R/W	属性 ID	默认值
U32	R/W	617	1
含义	设置 / 获取外部驱动脉冲输入模式		

注解：

设置、获取的值说明如下表所示：

值	说明
0	1XAB
1	2XAB
2	4XAB
3	CCW/CW

CFG_AxExtPresetNum

数据类型	R/W	属性 ID	默认值
U32	R/W	618	1
含义	当“JOG”模式接收到输入脉冲的一个有源沿时，设置 / 获取外部驱动个数		

B. 7. 16 凸轮区间 D0

CFG_AxCamDOEnable

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：

CamD0 和 OUT4 使用同一针脚，当启用 CFG_AxGenDoEnable 时，OUT4 不能够输出 CamD0 信号。

CFG_AxCamDOLoLimit

数据类型	R/W	属性 ID	默认值
U32	R/W	623	10000
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：-2147483647 ~ 2147483647。

CFG_AxCamDOHiLimit

数据类型	R/W	属性 ID	默认值
U32	R/W	624	10000
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：-2147483647 ~ 2147483647。

CFG_AxCamD0CmpSrc

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:

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 (不支持)

CFG_AxCamD0Logic

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:

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)

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。
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：

设置、获取的值说明如下：

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
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

B. 7. 19 触发停止

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：

设置、获取的值说明如下：

值	说明
0	禁用（默认值）
1	启用

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：

设置、获取的值说明如下：

值	说明
0	立即停止
1	减速停止（默认值）

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：

设置、获取的值说明如下：

值	说明
0	低准位（默认值）
1	高准位

CFG_AxIN2StopEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	638	0
含义	启用 / 禁用 IN2 触发停止功能		

注解：

设置、获取的值说明如下：

值	说明
0	禁用（默认值）
1	启用

CFG_AxIN2StopReact

数据类型	R/W	属性 ID	默认值
U32	R/W	639	1
含义	设定 / 获取 IN2 触发停止时的停止模式		

注解：

设置、获取的值说明如下：

值	说明
0	立即停止
1	减速停止（默认值）

CFG_AxIN2StopLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	640	0
含义	设定 / 获取 IN2 触发停止功能的逻辑准位		

注解：

设置、获取的值说明如下：

值	说明
0	低准位（默认值）
1	高准位

CFG_AxIN4StopEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	641	0
含义	启用 / 禁用 IN4 触发停止功能		

注解：

设置、获取的值说明如下：

值	说明
---	----

0	禁用（默认值）
1	启用

CFG_AxIN4StopReact

数据类型	R/W	属性 ID	默认值
U32	R/W	642	1
含义	设定 / 获取 IN4 触发时的停止模式		

注解：

设置、获取的值说明如下：

值	说明
0	立即停止
1	减速停止（默认值）

CFG_AxIN4StopLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	643	1
含义	设定 / 获取 IN4 触发停止功能的逻辑准位		

注解：

设置、获取的值说明如下：

值	说明
0	低准位（默认值）
1	高准位

CFG_AxIN5StopEnable

数据类型	R/W	属性 ID	默认值
U32	R/W	644	0
含义	启用 / 禁用 IN5 触发停止功能		

注解：

设置、获取的值说明如下：

值	说明
0	禁用（默认值）
1	启用

CFG_AxIN5StopReact

数据类型	R/W	属性 ID	默认值
U32	R/W	645	1
含义	设定 / 获取 IN5 触发时的停止模式		

注解：

设置、获取的值说明如下：

值	说明
0	立即停止
1	减速停止（默认值）

CFG_AxIN5StopLogic

数据类型	R/W	属性 ID	默认值
U32	R/W	646	1
含义	设定 / 获取 IN5 触发停止功能的逻辑准位		

注解:

设置、获取的值说明如下:

值	说明
0	低准位 (默认值)
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	2000
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	8000
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	10000
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	8000
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：

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

值	说明
0	T 型曲线（默认值）
1	S 型曲线

B. 8. 2 原点

PAR_AxHomeVelLow

数据类型	R/W	属性 ID	默认值
F64	R/W	415	2000
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	8000
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	10000
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	10000
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:

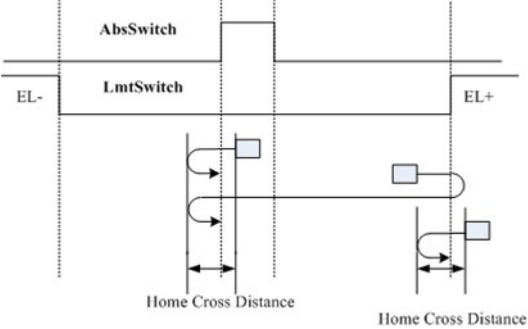
值	说明
0	T 型曲线（默认值）
1	S 型曲线

PAR_AxHomeCrossDistance

数据类型	R/W	属性 ID	默认值
F64	R/W	408	10000
含义	设置远点跨越距离。单位：PPU		

注解:

该属性值必须大于 0。跨越距离如下图所示:



PAR_AxHomeExSwitchMode

数据类型	R/W	属性 ID	默认值
F64	R/W	407	默认值为 0
含义	设置 AcM_AxHomeEx 的停止条件		

注解:

设置、获取的值说明如下:

值	定义	说明
0	Level On（默认值）	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：

该属性为按位操作，每位代表的值说明如下：

位	说明
0	0 轴
1	1 轴
2	2 轴
3	3 轴
4	4 轴
5	5 轴

B. 9. 2 路径

CFG_GpBldTime

数据类型	R/W	属性 ID	默认值
U32	R/W	808	10000
含义	设置 / 获取添加的路径与前一个路径的交接时间		

注解：

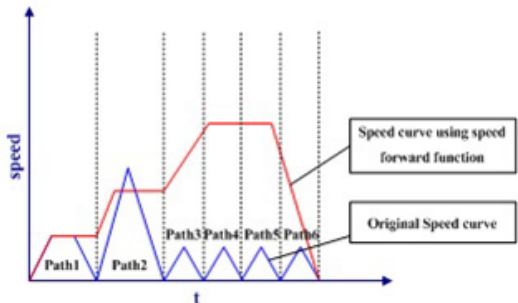
该值范围为 0 ~ 65535，且必须为 2 的倍数。单位：ms

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，与前一个路径的交接时间		

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	群组添加的第一个轴的初始速度
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	群组添加的第一个轴的 的执行速度
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 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。

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：

PCI1265 有 3 个 GroupID，分别为 0、1、2。用户不能同时处理多于三个群组，如果已经创建 3 个群组，用户需关闭一个才能创建新的群组。

PCI1245 系列板卡有 2 个 GroupID，分别为 0、1。如果已经创建 2 个群组，那么用户必须关闭一个才能创建新的群组。

B. 10.3 路径

PAR_GpRefPlane

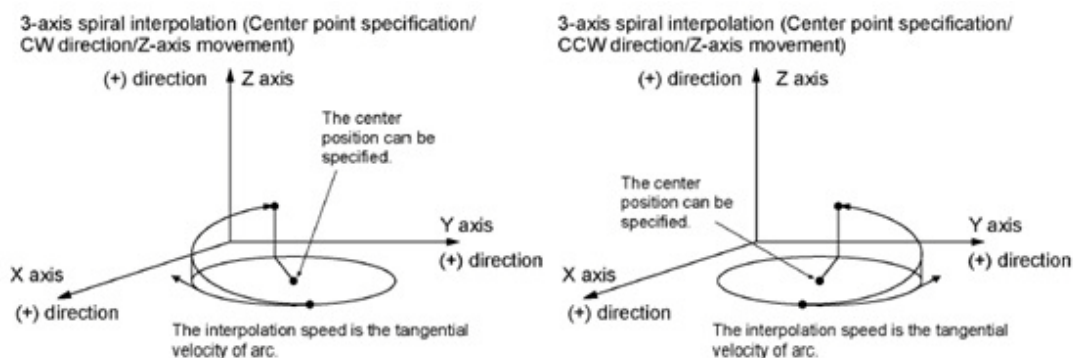
数据类型	R/W	属性 ID	默认值
U32	R/W	709	0
含义	设置 / 获取螺旋运动和圆弧插补的参考平面		

注解：

值	说明
0	XY 平面
1	YZ 平面
2	XZ 平面

注解：

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。



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, 表示正确调用了该函数

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 PCI-1265 板卡（其他板卡不支持设备 D0），调用 Acm_DaqDoSetByte 函数设置 D0 端口的值，PCI-1265 有 8 个 D0，因此当输入的参数 D0Port 大于 0 时，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 / 获取设备、轴、群组属性的函数
实例	调用函数设置 CFG_DaqAiRanges 属性的值，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	轴的最大运行速度为 5000000，当调用 Acm_SetF64Property 函数配置其值为 5000001 时，将会提示该错误，配置的属性为 CFG_AxMaxVel

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	轴的最大减速度为 50000000，当调用 Acm_SetF64Property 函数配置其值为 50000001 时，将会提示该错误，配置的属性为 CFG_AxMaxAcc

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	轴的最大减速度为 50000000，当调用 Acm_SetF64Property 函数配置其值为 50000001 时，将会提示该错误，配置的属性为 CFG_AxMaxDec

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	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	调用上述函数设置 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	调用上述函数设置 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	0x8000002e
错误信息	InvalidAxPosition
说明	无效的轴的位置
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：-2147483647/PPU ~ 2147483647/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	设置属性 PAR_GpVelLow 的值小于 0 或大于 CFG_AxMaxVel 的值时，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 的值大于 CFG_AxMaxAcc 或小于 0 时，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 的值大于 CFG_AxMaxDec 或小于 0 时，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 100000 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	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	例如调用函数 Acm_AxMoveImpose 执行叠加运动，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。例如 PCI-1245 共 4 轴，当设定的轴号为 5 时，报错

错误代码	0x80000069
错误信息	InvalidPath
说明	<p>对于 PCI-1240 板卡，提示该错误有两种情况</p> <p>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</p> <p>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</p> <p>对于 PCI-1245 系列 /65/85 系列板卡，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</p>
相关函数	Acm_GpLoadPath
实例	PCI-1245 板卡，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，设置的比较方法大于 3 或等于 2 时，提示该错误
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	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，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，必须板卡的所有轴都要打开，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 Acm_DevLoadConfig 函数下载配置文件失败时，提示该错误

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	0x800000d0
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 8000000 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	
相关函数	
实例	

错误代码	0x800000e6
错误信息	InvalidFileLength
说明	
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 100000 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。调用 Acm_CheckMotionEvent 函数检测事件时，在规定时间内没有检测到任何事件，提示该错误
相关函数	Acm_CheckMotionEvent
实 例	

错误代码	0x80004002
错误信息	DevNoEvt
说明	事件未发生。调用 Acm_CheckMotionEvent 函数检测事件时，没有检测到事件时提示该错误。没有检测到事件的原因是没有调用函数 Acm_CheckMotionEvent 将事件 Enable
相关函数	Acm_CheckMotionEvent
实例	

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	
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	调用函数设置 CFG_DaqAiRanges 属性的值，当板卡不支持该属性时，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 D0 字节数，当输入的参数 D0Port 大于 0 时，提示该错误

错误代码	0x80005005
错误信息	Dsp_DataOutBufExceeded
说明	
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	PCI-1245L 不支持圆弧插补，当调用圆弧插补函数时，将提示该错误

错误代码	0x80005007
错误信息	Dsp_InvalidConfigFile
说明	
相关函数	
实例	

错误代码	0x80005008
错误信息	Dsp_InvalidIntervalData
说明	
相关函数	
实例	

错误代码	0x80005009
错误信息	Dsp_InvalidTableSize
说明	
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 10000 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	当调用获取属性函数只写属性时，提示错误

错误代码	0x8000500f
错误信息	Dsp_NcError
说明	
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	
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	轴的最大运行速度为 5000000，当调用 Acm_SetF64Property 函数配置其值为 5000001 时，将会提示该错误，配置的属性为 CFG_AxMaxVel

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	轴的最大减速度为 50000000，当调用 Acm_SetF64Property 函数配置其值为 50000001 时，将会提示该错误，配置的属性为 CFG_AxMaxAcc

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	轴的最大减速度为 50000000，当调用 Acm_SetF64Property 函数配置其值为 50000001 时，将会提示该错误，配置的属性为 CFG_AxMaxAcc

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	
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，执行连续运动时，将提示该错误
错误代码	0x8000510c
错误信息	Dsp_InvalidAxSvOnOff
说明	
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	
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。例如 PCI-1245 共 4 轴，当设定的轴号为 5 时，报错

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	建立跟随关系时，主从轴必须为不同的轴
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，即当前轴状态下不能切换为外部驱动方式
相关函数	Acm_AxSetExtDrive
实例	
错误代码	0x8000511e
错误信息	InvalidGpRefPlane
说明	无效的参考平面
相关函数	Acm_AxGearInAx, Acm_AxCamInAx
实 例	

错误代码	0x8000511f
错误信息	Dsp_AxVelExceededMaxVel
说明	轴运行速度超过最大速度
相关函数	Acm_AxChangeVel
实例	调用上述函数，该表轴的运行速度，当新速度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	需要从群组中移除的轴未找到
相关函数	Acm_GpRemAxis
实例	调用上述函数移除轴 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	调用上述函数添加轴到群组中，当该轴在其他群组中时，提示该错误

错误代码	0x80005204
错误信息	Dsp_AxCannotIntoGp
说明	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	调用上述函数添加轴到群组中，当超过群组的最大轴数时，提示错误

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	
相关函数	
实例	

错误代码	0x80005207
错误信息	Dsp_InvalidGpCfgAcc
说明	
相关函数	
实例	

错误代码	0x80005208
错误信息	Dsp_InvalidGpCfgDec
说明	
相关函数	
实例	

错误代码	0x80005209
错误信息	Dsp_InvalidGpCfgJerk
说明	
相关函数	
实例	

错误代码	0x8000520a
错误信息	Dsp_InvalidGpParVelLow
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_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	
相关函数	
实例	

错误代码	0x80005210
错误信息	Dsp_ThreeAxNotSupport
说明	
相关函数	
实例	

错误代码	0x80005211
错误信息	Dsp_DevIpoNotFinished
说明	
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	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	
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	当 Path 缓存中的 Path 数小于 2 时，提示该错误

错误代码	0x80005215
错误信息	Dsp_InvalidPathHandle
说明	
相关函数	
实例	

错误代码	0x80005216
错误信息	Dsp_InvalidPath
说明	
相关函数	
实例	

错误代码	0x80005217
错误信息	Dsp_GpSlavePositionOverMaster
说明	
相关函数	
实例	

错误代码	0x80005219
错误信息	Dsp_GpPathBufferOverflow
说明	Path 缓存区已满（10000）
相关函数	Acm_GpAddPath
实例	调用 Acm_GpAddPath 函数添加的 Path 数超过 10000 段时，提示该错误

错误代码	0x8000521a
错误信息	Dsp_InvalidPathFunctionID
说明	
相关函数	
实例	

错误代码	0x8000521b
错误信息	Dsp_SysBufAllocateFailed
说明	
相关函数	
实例	

错误代码	0x8000521c
错误信息	Dsp_InvalidGpCenterPosition
说明	无效的 3D 圆弧中心位置
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 3D 圆弧终点位置
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 path 范围，即 path 起始索引号大于 path 的终止索引号或者 path 的起始索引号大于剩余 path 数
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), StartIndex 和 EndIdnex 则可取范围为 0~9

C.4 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	
0x80000013	ConnectWinIrqFailed	
0x80000024	InvalidAxAlarmEn	
0x80000025	InvalidAxAlarmLogic	
0x80000026	InvalidAxInPEn	
0x80000027	InvalidAxInPLogic	
0x80000028	InvalidAxHLmtEn	
0x80000029	InvalidAxHLmtLogic	
0x80000030	InvalidAxHLmtReact	
0x80000031	InvalidAxSLmtPEn	
0x80000032	InvalidAxSLmtPReact	
0x80000033	InvalidAxSLmtPValue	
0x80000034	InvalidAxSLmtMEn	
0x80000035	InvalidAxSLmtMReact	
0x80000036	InvalidAxSLmtMValue	
0x80000037	InvalidAxOrgLogic	
0x80000038	InvalidAxOrgEnable	
0x80000039	InvalidAxEzLogic	
0x80000040	InvalidAxEzEnable	
0x80000041	InvalidAxEzCount	
0x80000043	InvalidAxInEnable	
0x80000044	InvalidAxSvOnOff	
0x80000046	InvalidAxPosition	
0x80000047	InvalidAxHomeModeKw	
0x80000052	GpInDevNotFound	
0x80000053	InvalidGpCfgVel	
0x80000054	InvalidGpCfgAcc	
0x80000055	InvalidGpCfgDec	
0x80000056	InvalidGpCfgJerk	
0x80000064	DevIpoNotFinished	
0x80000070	IoctlError	
0x80000071	AmnetRingUsed	
0x80000072	DeviceNotOpened	
0x80000073	InvalidRing	
0x80000074	InvalidSlaveIP	
0x80000075	InvalidParameter	
0x80000076	InvalidGpCenterPosition	
0x80000077	InvalidGpEndPosition	
0x80000078	InvalidAddress	
0x80000079	DeviceDisconnect	

0x80000080	DataOutBufExceeded
0x80000081	SlaveDeviceNotMatch
0x80000082	SlaveDeviceError
0x80000083	SlaveDeviceUnknow
0x80000086	InvalidVelocity
0x80000087	InvalidAxPulseInLogic
0x80000088	InvalidAxPulseInSource
0x80000089	InvalidAxErcLogic
0x80000090	InvalidAxErcOnTime
0x80000091	InvalidAxErcOffTime
0x80000092	InvalidAxErcEnableMode
0x80000093	InvalidAxSdEnable
0x80000094	InvalidAxSdLogic
0x80000095	InvalidAxSdReact
0x80000096	InvalidAxSdLatch
0x80000097	InvalidAxHomeResetEnable
0x80000098	InvalidAxBacklashEnable
0x80000099	InvalidAxBacklashPulses
0x80000100	InvalidAxVibrationEnable
0x80000101	InvalidAxVibrationRevTime
0x80000102	InvalidAxVibrationFwdTime
0x80000103	InvalidAxAlarmReact
0x80000104	InvalidAxLatchLogic
0x80000105	InvalidFwMemoryMode
0x80000106	InvalidConfigFile
0x80000107	InvalidAxEnEvtArraySize
0x80000108	InvalidAxEnEvtArray
0x80000109	InvalidGpEnEvtArraySize
0x80000110	InvalidGpEnEvtArray
0x80000111	InvalidIntervalData
0x80000112	InvalidEndPosition
0x80000113	InvalidAxisSelect
0x80000114	InvalidTableSize
0x80000116	InvalidCmpSource
0x80000118	InvalidCmpPulseMode
0x80000119	InvalidCmpPulseLogic
0x80000120	InvalidCmpPulseWidth
0x80000121	InvalidPathFunctionID
0x80000122	SysBufAllocateFailed
0x80000150	SlaveIOUpdateError
0x80000151	NoSlaveDevFound
0x80000152	MasterDevNotOpen
0x80000153	MasterRingNotOpen
0x80000200	InvalidDIPort
0x80000201	InvalidDOPort
0x80000202	InvalidDOValue
0x80000204	CreateThreadFailed
0x80000205	InvalidHomeModeEx

0x80000206	InvalidDirMode
0x80000207	AxHomeMotionFailed
0x80000209	PathBufIsFull
0x80000210	PathBufIsEmpty
0x80000211	GetAuthorityFailed
0x80000212	GpIDAllocatedFailed
0x80000213	FirmWareDown
0x80000215	InvalidAxCmd
0x80000216	InvalidaxExtDrv
0x80000218	SpeedCurveNotSupported
0x80000219	InvalidCounterNo
0x80000221	PathSelStartCantRunInSpeedForewareMode
0x80002000	HLmtPExceeded
0x80002001	HLmtNExceeded
0x80002002	SLmtPExceeded
0x80002003	SLmtNExceeded
0x80002004	AlarmHappened
0x80002005	EmgHappened
0x80002006	TimeLmtExceeded
0x80002007	DistLmtExceeded
0x80002008	InvalidPositionOverride
0x80002009	OperationErrorHappened
0x80002010	SimultaneousStopHappened
0x80002011	OverflowInPAPB
0x80002012	OverflowInIPO
0x80002013	STPHappened
0x80002014	SDHappened
0x80002015	AxsiNoCmpDataLeft

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 2015