



深圳市雷赛控制技术有限公司

SHENZHEN LEADSHINE CONTROL TECHNOLOGY CO.,LTD

EMC 系列总线型独立式控制器 用户使用手册

2024.1

©Copyright 2024Leadshine Control Technology Co., Ltd.
All Rights Reserved.

版 权 说 明

本手册版权归深圳市雷赛控制技术有限公司所有，未经本公司书面许可，任何人不得翻印、翻译和抄袭本手册中的任何内容。

本手册中的信息资料仅供参考。由于改进设计和功能等原因，雷赛控制技术保留对本资料的最终解释权，内容如有更改，恕不另行通知。



调试机器要注意安全！用户必须在机器中设计有效的安全保护装置，在软件中加入出错处理程序。否则所造成的损失，雷赛控制技术没有义务或责任负责。

目录

第 1 章	产品概述	4
1.1	雷赛控制 EMC 系列总线控制器的特点	4
1.2	EMC 系列总线控制器主要技术指标	5
1.3	EMC 系列总线控制器的典型应用	6
1.4	订货信息	7
1.5	产品图片	9
第 2 章	雷赛控制 EMC 系列总线控制器功能介绍	10
2.1	通讯功能	10
2.2	安全设置	12
2.3	参数设置	13
2.4	运动控制功能	13
2.5	通用 IO 控制	21
2.6	EMC 控制器组网	21
第 3 章	硬件接口电路	22
3.1	接口布局	22
3.2	接口及引脚定义	23
3.3	通用 I/O 接口电路	25
第 4 章	控制器 Motion 的使用方法	28
4.1	控制器 Motion 的使用步骤	28
4.2	Motion 软件的功能与使用方法	30
第 5 章	应用软件开发方法	46
5.1	基于 WINDOWS 平台的应用软件结构	46
5.2	采用 VB 6.0 开发应用软件的方法	47
5.3	采用 VC 6.0 开发应用软件的方法	49
5.4	采用 C# 开发应用软件的方法	52
第 6 章	雷赛控制 EMC 系列总线控制器基本功能实现方法	56
6.1	控制器初始化	56
6.2	脉冲当量的设置	56
6.3	停止命令的设置	57
6.4	总线复位及电机使能	58
6.5	回原点运动的实现	59

6.6	点位运动的实现	60
6.7	连续运动的实现	63
6.8	插补运动的实现	65
6.9	连续插补运动的实现	86
6.10	PWM 输出功能的实现	99
6.11	异常减速停止时间设置功能的实现	108
6.12	检测轴到位状态功能的实现	109
6.13	通用 I/O 控制的实现	110
6.14	位置比较功能的实现	112
6.15	位置锁存功能的实现	115
6.16	螺距补偿功能的实现	117
6.17	龙门功能的实现	119
6.18	PVT 运动功能的实现	120
6.19	看门狗功能的实现	130
6.20	软着陆功能的实现	131
6.21	软启动功能的实现	133
6.22	圆形区域限位功能的实现	134
6.23	椭圆插补及切向跟随的实现	135
第 7 章	总线操作函数说明	137
7.1	总线配置函数	137
7.2	总线通讯函数	138
7.3	总线 IO 及轴控制函数	144
7.4	回零运动函数	153
第 8 章	基本功能函数说明	162
8.1	控制器设置	162
8.2	脉冲当量与限位设置	165
8.3	单轴运动速度曲线设置	167
8.4	单轴运动	169
8.5	插补速度设置	176
8.6	插补运动	177
8.7	连续插补运动	182
8.8	连续插补缓冲区检测	188
8.9	连续插补小线段前瞻功能	189
8.10	连续插补 IO 控制	190

8.11	圆弧限速	193
8.12	连续插补位置跟随	194
8.13	PWM 功能	194
8.14	连续插补 DA 跟随	199
8.15	位置计数器控制	200
8.16	运动状态检测及控制	200
8.17	输入输出 IO	207
8.18	轴 IO 映射	211
8.19	编码器函数	213
8.20	低速位置比较	214
8.21	软件位置锁存	217
8.22	螺距补偿	219
8.23	龙门功能	220
8.24	减速停止时间设置	222
8.25	检测轴到位状态	223
8.26	反向间隙设置	225
8.27	PVT 运动	226
8.28	看门狗功能	228
8.29	密码管理	229
8.30	打印输出函数	229
附 录		231
附录 1	常见电路接线方法	231
附录 2	常见错误码说明	234
附录 3	常用 PDO 对象表	258
附录 4	运动控制函数索引	262
附录 5	常见问题解决方法	268

第 1 章 产品概述

1.1 雷赛控制 EMC 系列总线控制器的特点

EMC-E3000/5000 系列运动控制器是深圳市雷赛控制技术有限公司开发的具有自主知识产权的新型高性能 EtherCAT 总线点位/轨迹型运动控制器。该控制器在通用的 EtherCAT 总线功能基础上,还拥有强大的运动功能。通讯接口方面,使用以太网和上位机进行通讯,通信速率可达到 100Mbps;同时新增背板总线,可直接在控制器背板上扩展 IO、模拟量等模块。其主要特点如下:

- 1) 接线方便,通过网线连接EtherCAT总线伺服/步进驱动器、EtherCAT总线IO模块、EtherCAT总线模拟量模块、EtherCAT总线定位模块等。
- 2) 接口丰富,支持254个控制器组网控制,拥有RS232/485等主流通讯接口,本地扩展支持多达32个模块,同时自带本体12入12出高速IO。
- 3) 总线负载能力强,最快可达500us,轴扩展方便,最多可扩展64轴。
- 4) 开放易用,提供易用的上位机调试软件,支持Windows/Linux/MacOS等操作系统。
- 5) 算法优异,高性能的点位运动算法,支持T形、S形、S Plus等多种速度规划。
- 6) 支持位置控制、速度控制,支持运动中变速、变位功能。
- 7) 支持4个插补坐标系,每个坐标系支持2~16轴直线插补、两轴圆弧插补、空间圆弧插补、两轴及三轴螺旋线插补、两轴矩形插补等运动。4个坐标系的速度可独立设置,可同时进行4组插补运动。当轴数大于3时,支持前三轴做螺旋插补或圆弧插补的同时,后续轴跟随前三轴做线性运动。
- 8) 性能卓越,拥有优秀的前瞻轨迹运动。EMC-E5000系列支持连续插补运动,连续插补缓冲区可装5000条指令。插补运动中支持直线插补、两轴圆弧插补、两轴及三轴螺旋线插补、空间圆弧插补、两轴矩形插补等算法;支持小线段前瞻、连续插补暂停、延时、IO控制等功能。
- 9) EMC-E5000支持圆弧插补限速,减小设备冲击,保证加工精度,提升圆弧加工品质。
- 10) 支持反向间隙补偿功能,可降低机械传动反向间隙的影响。
- 11) 支持PWM输出功能。
- 12) 支持螺距补偿功能、龙门运动功能。
- 13) 支持位置锁存、一维、二维低速位置比较。
- 14) 设置精良小巧,外形美观,安装尺寸较友商同类产品小50%,安装方式兼容标准DIN导轨安装。

 **注意:** EMC-E5000连接64轴,开启4个插补系满负载运行时,总线通信周期必须设置大于等于2ms。EMC-E3000系列不支持连续插补前瞻轨迹功能。

1.2 EMC 系列总线控制器主要技术指标

表 1.1 EMC-E3000/5000 系列主要技术指标

技术指标 \ 控制器类型	EMC-E3000系列	EMC-E5000系列
可扩展电机轴数	64	64
总线通讯速率	最大100Mbps	最大100Mbps
支持的总线周期	500us、 1 ms 、 2 ms	500us、 1 ms 、 2 ms
支持的插补坐标系个数	4	4
通用数字输入口数量	16（可扩展）	16（可扩展）
通用数字输出口数量	16（可扩展）	16（可扩展）
通用数字输入口	光电隔离，RC滤波	光电隔离，RC滤波
通用数字输入口导通电流	≥4.2 mA（15V） 典型值6.9mA （24V）	≥4.2 mA（15V） 典型值6.9mA （24V）
通用数字输入口最高响应频率	4 kHz	4 kHz
通用数字输出口	光电隔离，集电极开路	光电隔离，集电极开路
通用数字输出口最大电流	500 mA（5~24Vdc，吸入）	500 mA（5~24Vdc，吸入）
高速输入口数量	12	12
高速输出口数量	12	12
工作温度	0~50℃	0~50℃
贮存温度	-20~80℃	-20~80℃
湿度	5~85%，非结露	5~85%，非结露
外部电源（输入）	24VDC±5%，1A	24VDC±5%，1A
运动控制函数库	支持VC、VB6.0、C#、VB.NET、 LabVIEW、Delphi等多种语言	支持VC、VB6.0、C#、VB.NET、 LabVIEW、Delphi等多种语言
调试软件	Motion调试软件	Motion调试软件

表 1.2 EMC-E3000/5000 系列主要功能

功能 \ 控制器类型	EMC-E3000系列	EMC-E5000系列
点位运动	√	√
在线变速/在线变位	√	√
回零运动	√	√
PVT运动	√	√
软件锁存	√	√
螺距补偿	√	√
反向间隙	√	√
轴IO映射	√	√
直线插补	√	√
圆弧插补	√	√
椭圆插补	×	√
连续插补	×	√
连续插补前瞻	×	√
连续插补圆弧限速	×	√
连续插补缓存IO控制	×	√
连续插补PWM跟随	×	√
连续插补DA跟随	×	√
连续插补位置跟随	×	√
连续插补切向跟随	×	√
CSP（周期位置控制模式）	√	√
CST（周期转矩控制模式）	√	√

1.3 EMC 系列总线控制器的典型应用

EMC-E3000/5000系列总线运动控制器可应用于各行各业自动化设备中。主要面向3C、电子、点胶、激光等行业客户群体。

主要设备有：

- 1) 电子产品加工、装配设备，如：丝印机、贴片机、PCB钻孔机等
- 2) 绕线机，半导体自动化设备、汽车电子自动化设备等。
- 3) 机器视觉及自动检测设备，如：影像测量仪、电路板自动检测设备等

雷赛控制 EMC 系列总线控制器典型系统架构如下：

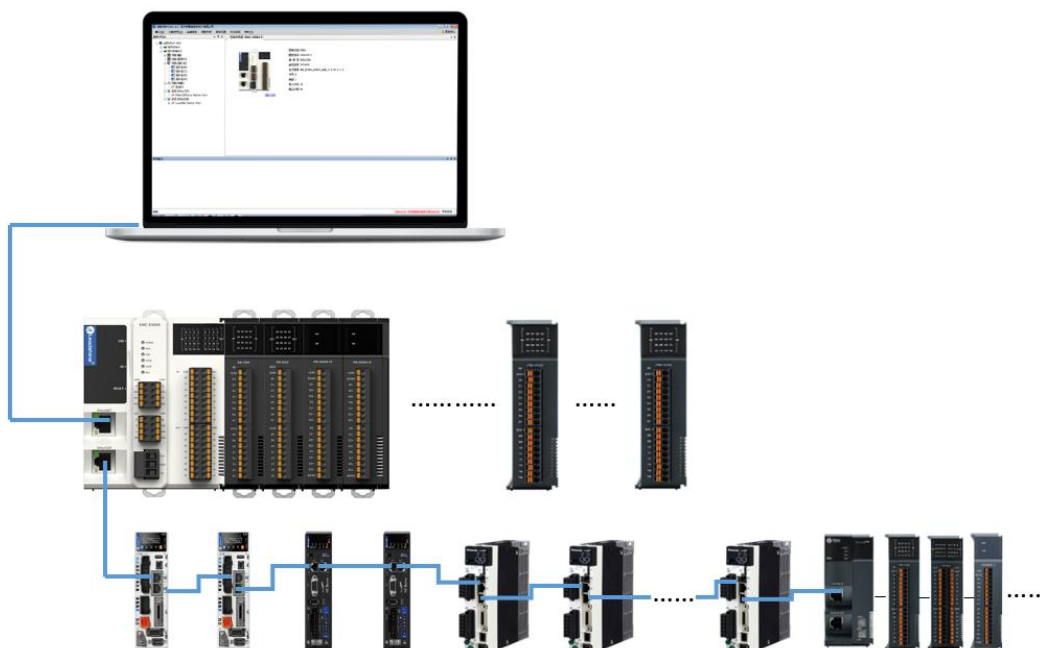


图1.1 EMC3000/5000系列运动控制器典型系统架构

1.4 订货信息

表 1.3 EMC-E3000/E5000 控制器订货信息

产品系列	控制器型号	备注
EMC-E5000系列高端轨迹型	EMC-E5064-A64	最大配置128节点，其中A64表示A64轴控制器，A32代表32轴控制，以此类推；
	EMC-E5064-A32	
	EMC-E5064-A24	
	EMC-E5064-A16	
	EMC-E5064-A12	
	EMC-E5064-A08	
EMC-E3000系列通用点位型	EMC-E3064-A64	
	EMC-E3064-A32	
	EMC-E3064-A24	
	EMC-E3064-A16	
	EMC-E3064-A12	
	EMC-E3064-A08	

表 1.4 EMC-E3000/E5000 控制器背板总线配件订货信息

类型	型号	点数	接线端子
IO模块	PM-1600	16入	弹压式
	PM-3200	32入	弹压式
	PM-3200-1	32入	MIL接头
	PM-3200-2	32入	富士通接头
	PM-0016-N	16出	弹压式
	PM-0016-P	16出	弹压式
	PM-0016-R	16出	弹压式
	PM-0032-N	32出	弹压式
	PM-0032-N-1	32出	MIL接头
	PM-0032-N-2	32出	富士通接头
	PM-1616-N	16入16出	弹压式
模拟量模块	PM-A0400-IV	4AD	弹压式
	PM-A0004-IV	4DA	弹压式

1.5 产品图片

EMC-E3000/5000系列运动控制器外观示意图如图1.2所示。



图1.2 EMC-E3000/5000系列总线运动控制器外观示意图

EMC-E3000/5000 系列运动控制器尺寸示意图

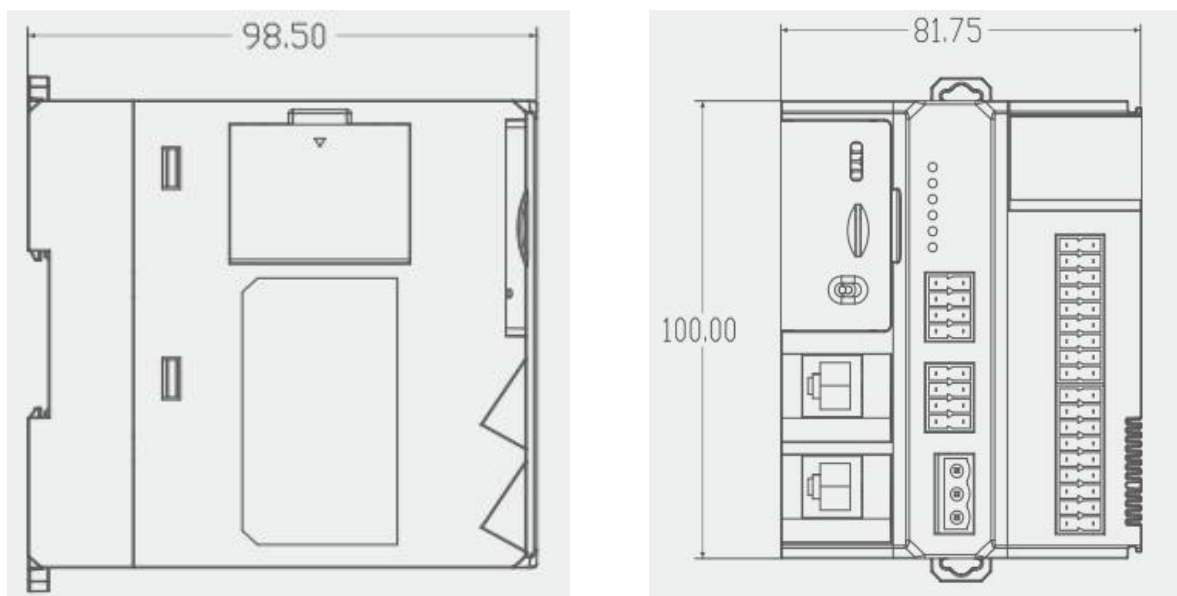


图 1.3 EMC-E3000/5000 系列总线运动控制器结构尺寸图

第 2 章 雷赛控制 EMC 系列总线控制器功能介绍

雷赛控制 EMC-E3000/5000 系列总线运动控制器是基于 EtherCAT 总线设计，控制相关操作全部通过 EtherCAT 总线来控制，可以通过总线扩展 EtherCAT 总线伺服驱动器、EtherCAT 总线步进驱动器、EtherCAT 总线 IO 模块、EtherCAT 总线模拟量模块等其他符合 ETG 标准的从站。除了本地 IO 以及通信功能之外，本章节的其他的功能说明都是通过 EtherCAT 总线来实现。

EMC 系列运动控制器覆盖多轴点位运动、插补运动、轨迹规划、编码器位置检测、IO 控制、位置比较、位置锁存等多种功能。不仅运动控制函数库功能丰富、易学易用，而且用户开发应用软件十分方便。

2.1 通讯功能

2.1.1 RS232 通信（保留）

EMC 系列运动控制器提供一路 RS232 串口，通讯参数可以根据实际情况配置，支持 MODBUS RTU 协议以及自定义协议。

2.1.1.1 参数配置

EMC 系列运动控制器的 RS232 通信参数如表 2.1 所示。

表 2.1 RS232 通信参数

参数名称	出厂设置	数值范围
数据位	8	7, 8
停止位	1	1, 2
校验	2	0 无校验, 1 奇校验, 2 偶校验

注意：使用串口通信时请注意通讯速率和线长相匹配，通讯速率和通讯线长成反比，波特率设置为 115200 时，线长应不超过 3 米，宜采用完整带屏蔽的通信线，不建议使用转接线。

2.1.1.2 通讯协议

EMC 系列运动控制器的 RS232 串口支持 MODBUS RTU 协议，可以和任何支持标准 MODBUS RTU 协议的外设进行数据交互。

EMC 系列运动控制器的 RS232 串口还支持自由协议，用户可以自定义协议规则，使用对

应的读写指令进行操作。

2.1.2 RS485 通讯（保留）

EMC 系列运动控制器提供一路 RS485 串口，通讯参数可以根据实际情况配置，支持 MODBUS RTU 协议以及自定义协议。

2.1.2.1 参数配置

EMC 系列运动控制器的 RS485 通信参数如表 3.2 所示。

表 2.2 RS485 通信参数

参数名称	出厂设置	数值范围
数据位	8	7, 8
停止位	1	1, 2
校验	2	0 无校验, 1 奇校验, 2 偶校验

2.1.2.2 通信协议

EMC 系列运动控制器的 RS485 串口支持 MODBUS RTU 协议，可以和任何支持标准 MODBUS RTU 协议的外设进行数据交互。

EMC 系列运动控制器的 RS485 串口还支持自由协议，用户可以自定义协议规则，使用对应的读写指令进行操作。

2.1.3 以太网通讯

2.1.3.1 参数配置

EMC 系列运动控制器出厂默认 IP 地址为：Ethernet: 192.168.5.11。

用于以太网连接 PC 和控制器，控制器的 IP 地址第 4 网段可以任意设置。设置方式可在 Motion 中操作。用户也可根据自己的需要调用相关函数，为控制器设置不同的 IP 地址。在 PC 机与控制器连接之前，需要保证 PC 机的 IP 地址与控制器 IP 地址前 3 个处于同一网段，如 EMC 默认 IP 地址为 192.168.5.11，PC 机设置的 IP 与 EMC 的 IP 前 3 个字段要相同，第 4 个字段要不同。如图 2.1 所示，PC 机设置的 IP 设为 192.168.5.6 即可。

注意：网口通讯只用于控制器与上位机通讯。

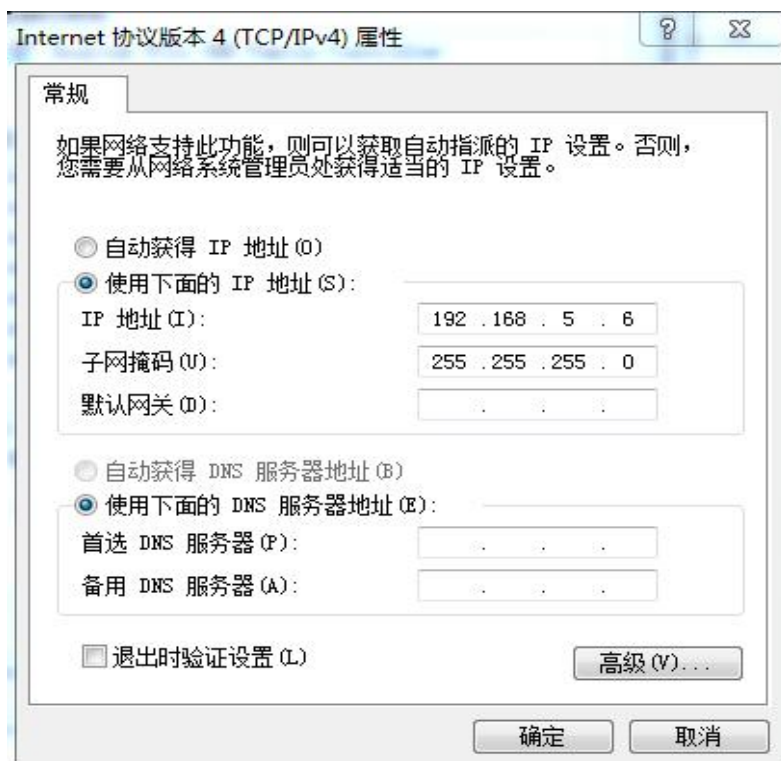


图 2.1 PC 网段设置

2.1.3.2 通讯协议

EMC 系列运动控制器 Ethernet 网口支持标准 TCP/IP 协议，可实现多台控制器组网。最大支持 254 台控制器进行组网。使用方便简洁，可极大的为用户节省工控机的成本。用户只需将每个控制器的地址 IP 设置不相同，链接号进行区分。即可精准的操作每个控制器。

2.2 安全设置

2.2.1 急停映射

EMC 系列运动控制器上没有提供专用的 EMG 信号接口，为了保证设备在调试或者运行时候的安全，我们建议在控制器的本地通用输入端口选择一个输入口作为急停信号（EMG）使用。使用 IO 映射功能将急停信号映射到通用输入口，当此通用输入口检测到信号输入时，会触发急停操作。

2.2.2 看门狗功能

EMC 系列运动控制器支持看门狗功能，能够实现控制器的安全保护功能，当控制器开启

看门狗功能，在设定的时间内未得到响应即触发看门狗事件。看门狗触发事件有停止更新指令位置， 停止轴使能等。具体实现方法可参考 6.21 章节。

2.3 参数设置

EMC 系列运动控制器支持将轴或者 IO 等相关参数，包括脉冲当量、运动参数、回零参数、IO 配置以及 IO 映射等参数下载到控制器本地。控制器通过文件的形式存储。同时控制器支持将参数文件导出以及将文件下载到控制器本地。

2.4 运动控制功能

2.4.1 点位运动

点位运动是指：控制器控制运动平台从当前位置开始以设定的速度运动到指定位置后准确地停止。

点位运动只关注终点坐标，对运动轨迹的精度没有要求。通过调用点位运动指令函数，使控制器按设定的速度发出指令，运动的距离由点位运动指令决定。位移与时间的关系如图 2.2 所示。

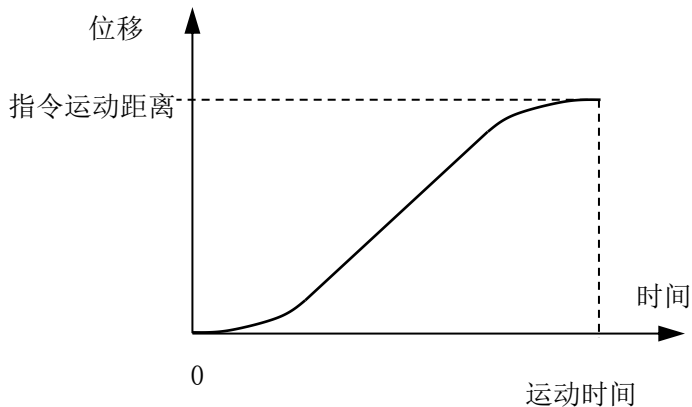


图2.2 定长运动位移曲线

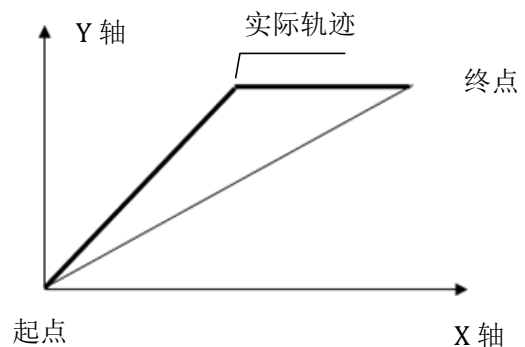


图2.3 两轴复合运动的轨迹

多轴同时做点位运动，称之为多轴联动。由于软件处理速度远比机械系统响应速度快，虽然多条运动指令连续发出，需几个微秒，可对机械系统而已，可认为是同时启动。如果每个轴的运动速度相同，则多轴运动的轨迹就可能是折线，如图 2.3 所示。

如果从起点到终点都需要按照规定的路径运动，就必须采用直线插补或圆弧插补功能。

2.4.1.1 梯形速度控制

为了让平台在运动过程中能平稳加速、准确停止，一般采用梯形速度曲线控制运动过程，如图2.4所示。即：电机以起始速度开始运动，加速至最大速度后保持速度不变，结束前减速至停止速度，并停止。运动的距离由点位运动指令决定。

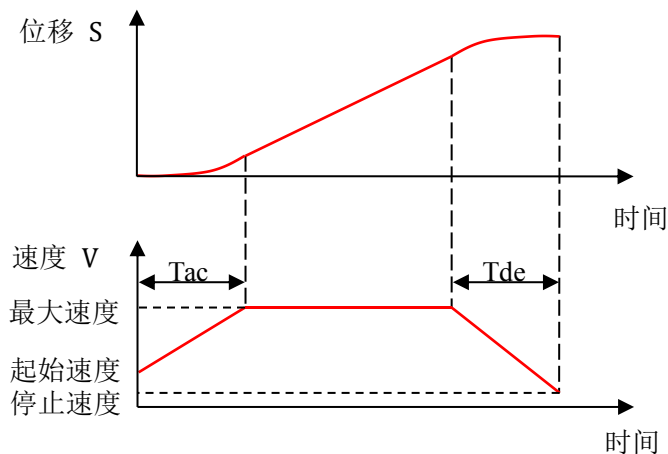


图2.4 梯形速度曲线及对应的位移曲线

Tacc: 总加速时间, Tdec: 总减速时间

2.4.1.2 S 形速度控制

为改善平台运动的平稳性，控制器还提供了S形速度控制曲线。

在S形速度控制过程中，指令速度从一个内部设定的速度快速加速到起始速度，然后作S形加速运动；运动结束前，指令速度作S形减速运动到停止速度，然后再快速减速到一个内部设定的速度，这时运动停止，如图2.5所示。

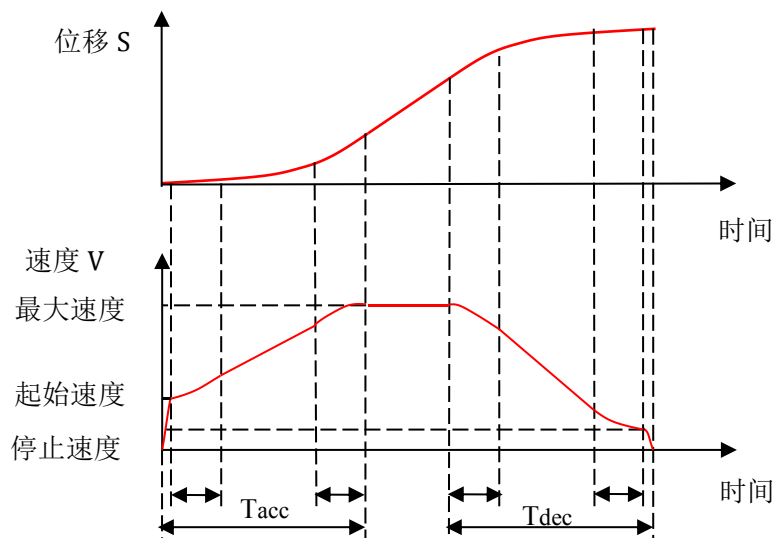


图2.5 S形速度曲线及对应的位移曲线

2.4.1.3 S Plus 形速度控制

S Plus速度曲线规划同S 形曲线规划类似，但S plus可自主设置加加速度(Ajerk)，减减速度(Djerk)。可以更加灵活的探索平台能承受的极限，使设备能够实现最大化效率。

在S Plus速度控制过程中，运动到最大加速度的时间是加加速度与加速度的一个比值。在到达最大加速度的这个阶段可以称为平滑阶段。从最大速度减小到0的过程和加速过程一致

因此在用户了解设备所能承受的最大加速度值后，便可以灵活调节Ajerk与Djerk值，使设备达到最大效率。如图2.6所示。

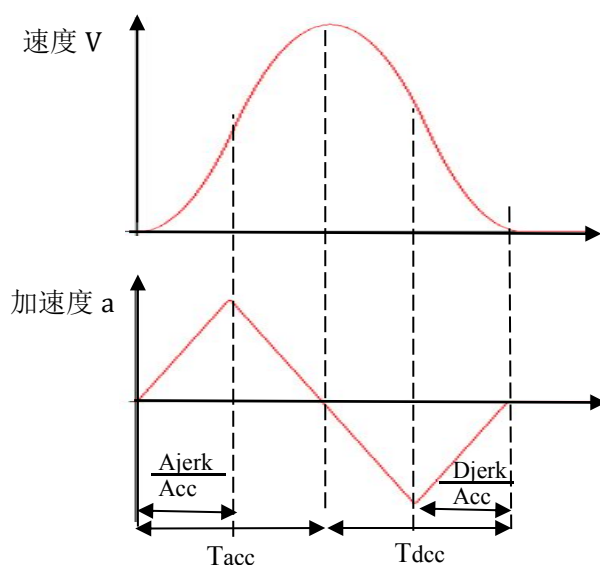


图2.6 S Puls形速度曲线及对应的位移曲线

2.4.1.4 运动中改变终点位置

在进行点位运动的过程中，控制器可以改变运动的终点位置，且位置可增可减。如图 2.7 所示，原本点位运动的距离是 50000；但当运动了 550ms 时，将终点改为 100000，电机从减速变为加速，继续向前运动，然后减速停在新的终点位置。

该功能在固晶机上使用十分方便。首先，取料臂开始向一个理想位置运动；摄像头检测晶片的实际位置；然后给出终点的修整位置；取料臂向新的位置运动。这样可以大大提高设备的生产效率。

spara: S 段时间, Tacc: 总加速时间, Tdec: 总减速时间

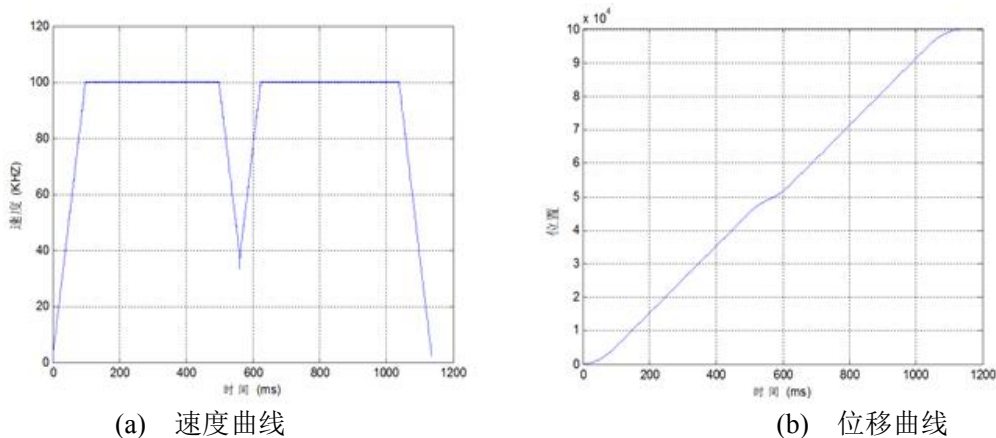


图2.7 改变终点位置的过程

2.4.1.5 运动中改变当前速度

在点位运动（或连续运动）过程中，EMC-E3000/5000 系列总线运动控制器可以改变运动的速度，且速度变化过程和预设的梯形速度曲线或 S 形速度曲线相同，如图 2.8 所示。

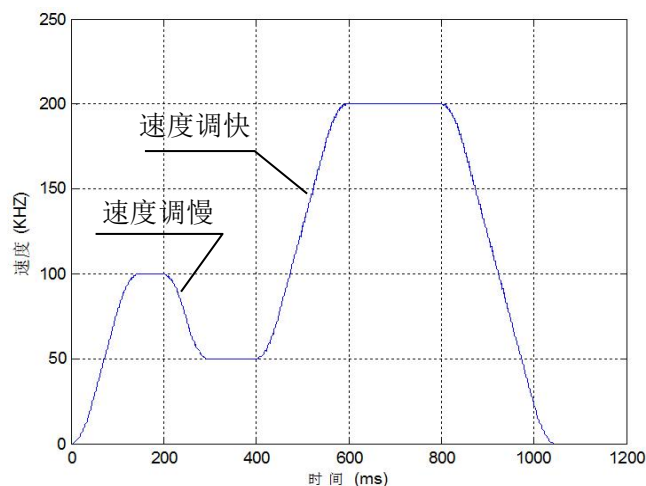


图2.8 调速过程

2.4.2 连续运动

连续运动是指：电机从起始速度开始运行，加速至最大速度后连续运动；只有当接收到停止指令或外部停止信号后，才减速停止。

连续运动指令其实就是速度控制指令，国外控制器将此指令称为 JOG 指令。

EMC-E3000/5000 系列总线运动控制器可以控制电机以梯形或 S 形速度曲线在指定的加速时间内从起始速度加速至最大速度，然后以该速度连续运行，直至调用停止指令或者该轴遇到限位信号、急停信号才会按设定的减速方式减速停止。连续运动指令的速度与时间曲线如图

2.9 所示。

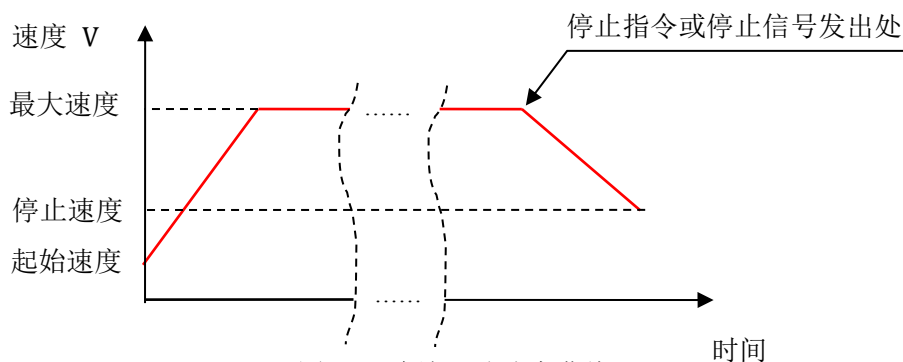


图2.9 连续运动速度曲线

该功能的主要用途是：速度控制，如：传送带的速度、包装机连续送料速度等。

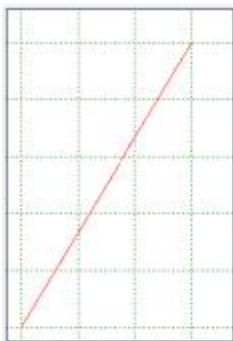
2.4.3 插补运动

为了实现轨迹控制，控制器按照一定的控制策略控制多轴联动，使运动平台用微小直线段精确地逼近轨迹的理论曲线，保证运动平台从起点到终点上的所有轨迹点都控制在允许误差范围内。这种控制策略称为插补算法，因此轨迹运动通常称为插补运动。插补运动有许多种类，如：直线插补、圆弧插补、螺旋线插补等。如图 2.10 所示为各种曲线轨迹的示意图。

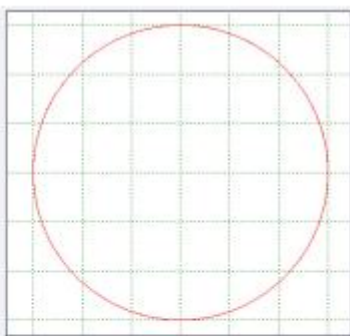
插补运动模式可以分为单段插补与多段连续插补两种。

单段插补可实现单段直线插补、单段圆弧插补、单段螺旋插补等运动。

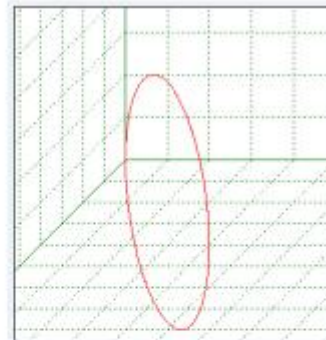
多段连续插补可分为非前瞻和前瞻。可实现各种插补曲线连续运行，可实现速度的平滑过渡，减少机器振动，提高加工效率与加工精度。



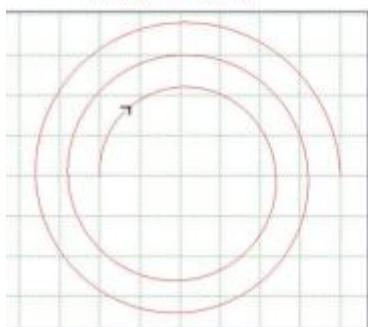
两轴直线插补



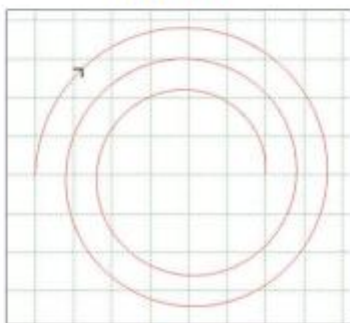
两轴圆弧



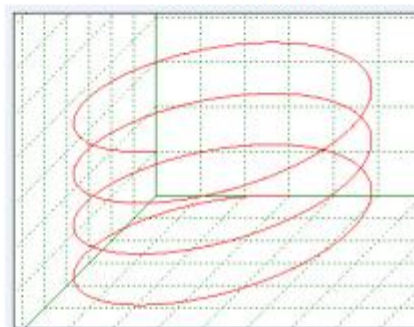
空间圆弧



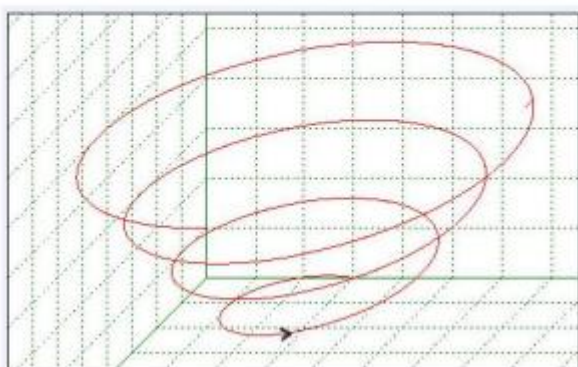
两轴螺旋线（绽放型）



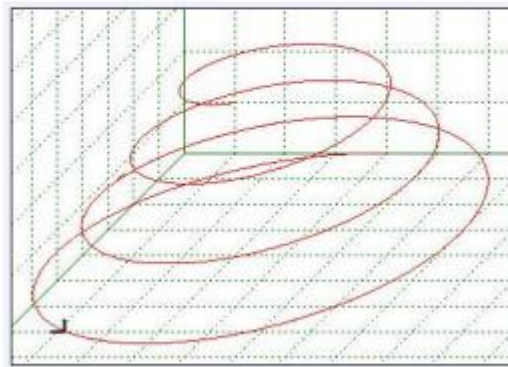
两轴螺旋线（收敛型）



圆柱螺旋线



空间螺旋线（绽放型）



空间螺旋线（收敛型）

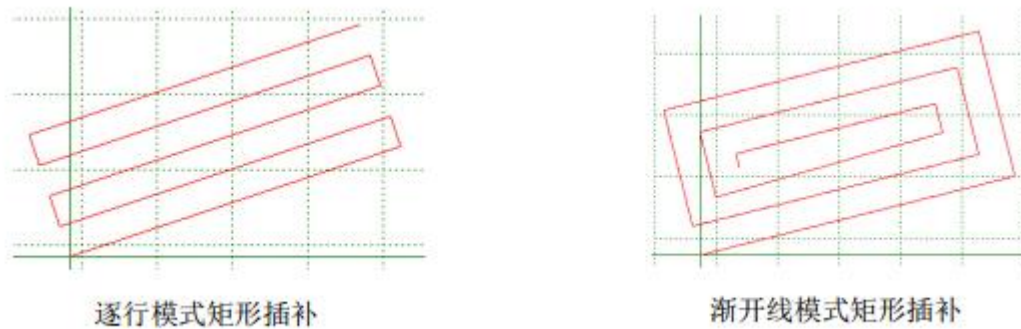


图 2.10 各种曲线轨迹示意图

2.4.3.1 单段插补运动

单段插补各段插补独立不连续，插补指令逐条边解释边执行，一条指令在执行的过程中不允许插入下一条指令。

EMC-E3000/5000 系列运动控制器可以进行任意 2~16 轴直线插补，同时 EMC-E3000/5000 系列运动控制器不仅可以进行两轴圆弧插补，并且可以进行两轴及三轴螺旋线插补、空间圆弧插补等。此外，当插补轴数大于 3 时，EMC-E3000/5000 系列运动控制器还支持前三轴做螺旋插补或圆弧插补的同时，剩余轴数可根据插补运动做同时启停的点位运动。

2.4.3.2 多段连续插补运动

EMC-E5000 系列控制器提供了多段连续插补运动功能，采用多段连续插补可实现速度的平滑过渡，减小机器的振动，生产出高质量的工件，同时能提高机器的加工速度。连续插补指令支持直线插补、圆弧插补、螺旋线插补、IO 控制等。各种曲线轨迹示意图见图 2.10。

多段连续插补运动功能，包含前瞻和非前瞻运动。连续插补指令支持直线插补、圆弧插补、螺旋线插补、IO 控制等，前瞻和非前瞻主要区别在于前瞻可很好应用于小线段轨迹，其轨迹连接处更平滑。

连续插补对于圆弧提供了圆弧限速功能，主要为限制运行速度，使加速度值不超出设定范围。具体功能见下表：

连续插补功能	非前瞻	前瞻
插补系	4 个	4 个
长线段轨迹	支持	支持
小线段轨迹	不支持	支持
圆弧限速	支持	支持
T\S 形曲线	支持 T、S 形	支持 T、S 形

加、减速时间	可自由设定	对称曲线
起始、停止速度	可自由设定	不支持
速度倍率	支持(下一轨迹生效)	支持(立即生效)
插补延时	支持	支持
IO 等待输入	支持	支持
IO 立即输出	支持	支持
IO 超前、滞后输出	支持	支持

此外,EMC-E5000 系列控制器支持 4 个坐标系,每个坐标系的连续缓冲区最多可缓存 5000 条指令。4 个坐标系的速度可独立设置,执行连续插补时 4 个坐标系可独立进行连续插补运动,即可同时进行 4 组连续插补运动。

2.4.4 回原点运动

在进行精确的运动控制之前,需要设定运动坐标系的原点。运动控制系统通常会执行寻找该原点的动作将系统坐标位置归零,即回原点运动。如图 2.11 所示,在运动平台上,每个轴都有一个位置传感器用于设置一个位置参考点,即原点位置,以便进行位置控制。在正常运动之前,都需要用回零指令控制平台向原点方向运动,当回零完成后,平台自动停止,此时系统可将该位置作为一个机械标准位置作为系统参考。EMC-E3000/5000 系列回原点参数可由控制器设置,由驱动器完成,具体回零方法请参考驱动器手册。

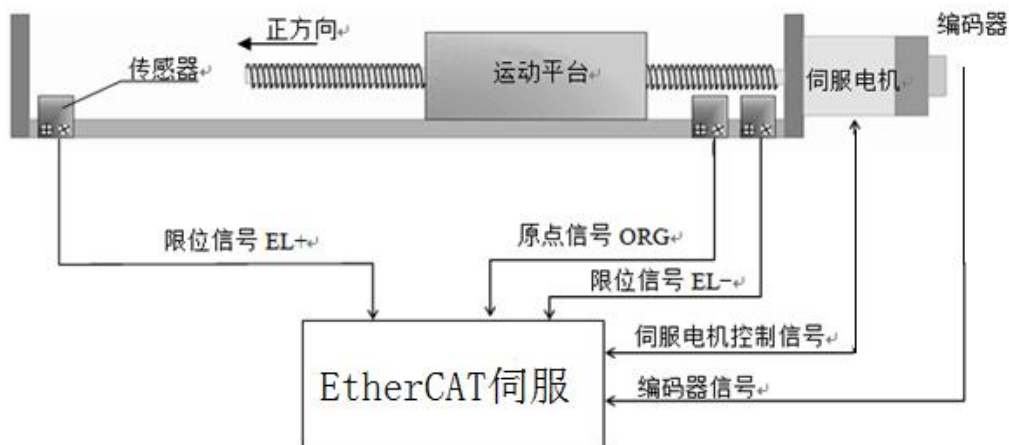


图 2.11 运动平台传感器信号及电机控制信号与伺服驱动器的关系

2.4.5 异常减速停止时间设置功能

EMC-E3000/5000 系列运动控制器支持异常减速停止时间设置功能,异常减速停止包括命令减速停止、硬限位减速停止、软限位减速停止、IO 触发减速停止等,用户根据现场实际需求情况设定减速停止时间可达到理想的减速效果。

2.5 通用 IO 控制

EMC-E3000/5000 系列运动控制器除了运动控制功能外，还提供了数字式输入信号（Input）和输出信号（Output），即 IO 信号的控制功能。

2.5.1 通用 IO 信号

在电机运动的同时，EMC-E3000/5000 系列运动控制器本地 IO 可接受按键、数字式传感器等输入信号，输出控制指示灯、继电器、电磁阀等器件。

2.5.2 EtherCAT IO 扩展功能

当设备需求的 IO 端口数量较多时，EMC-E3000/5000 系列运动控制器可配套 EtherCAT 总线 IO 模块，对通用输入输出端口进行扩展。

2.5.3 背板 IO 扩展功能

EMC-E3000/5000 系列运动控制器除拥有 EtherCAT IO 扩展功能外，还支持背板总线扩展，可以支持扩展 IO，最大扩展模块数多达 32 台。具体操作可参考 4.2.1.2 背板总线配置章节。

2.6 EMC 控制器组网

EMC-E3000/5000 系列运动控制器通过 Ethernet 网口和 PC 进行通讯，可同时支持 254 控制器进行组网。使用方便简洁，可极大的为用户节省工控机的成本。用户只需将每个控制器的地址 IP 设置不相同，链接号进行区分，即可精准的操作每个控制器。

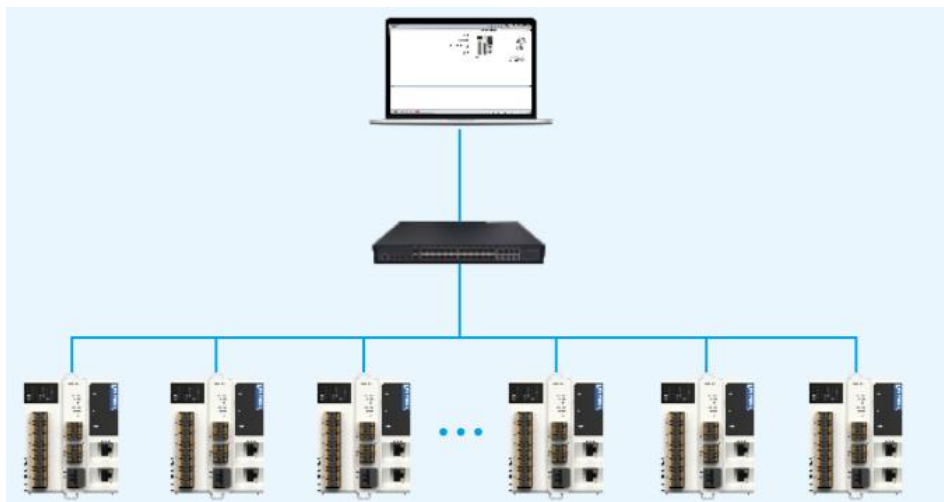


图 2.12 EMC 控制器组网示意图

第3章 硬件接口电路

3.1 接口布局

本章节主要介绍 EMC 系列运动控制器的外围接口以及相应的连接方式。用户可参照本章节内容，完成控制器与外部执行单元（电机驱动器、开关器件等）的电气连接部分工作

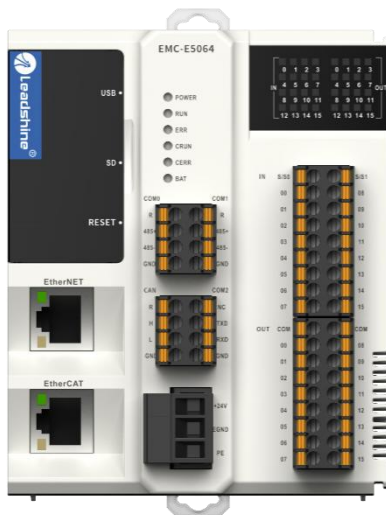


图3.1 EMC-E3000/5000系列运动控制器实物图

EMC-E3000/5000 系列运动控制器硬件布置及尺寸如图 3.2 所示。

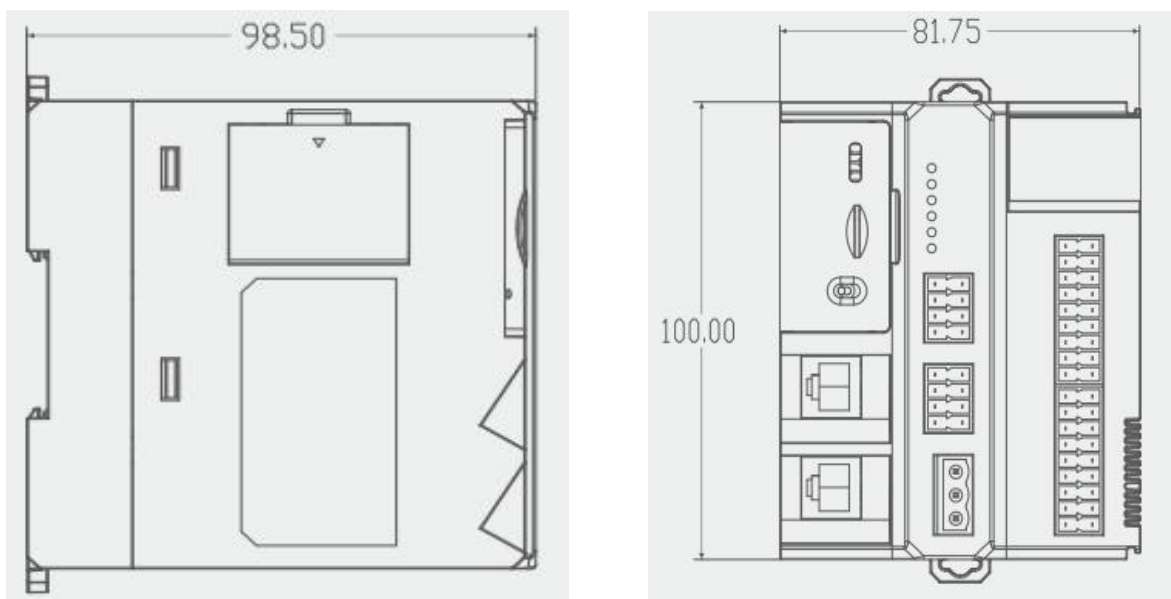


图 3.2 EMC-E3000/5000 系列运动控制器结构尺寸图

3.2 接口及引脚定义

EMC-E3000/5000 系列总线运动控制器的接口及脚位如表 3.1 所示。

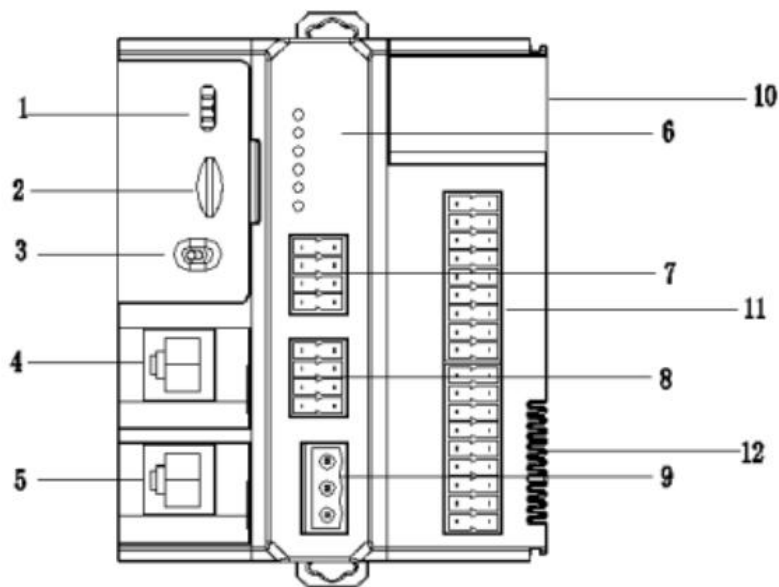


图 3.3 EMC-E3000/5000 系列运动控制器引脚分布图

表 3.1 脚位定义

名称	功能介绍
1	Type C 接口
2	SD 卡接口（保留）
3	RESET 拨码
4	Ethernet 网口
5	Ethercat 网口
6	状态指示灯
7	RS485 接口
8	CAN 接口（保留）/RS232 接口
9	电源接口
10	IO 指示灯
11	通用输入输出
12	背板总线扩展接口

引脚定义描述：

1) Type C 接口：Type C 接口用于 EMC-E3000/5000 系列总线运动控制器和上位机连接，主要用于查看控制器 IP 等相关状态的查询显示，不建议用于正常运行调试时的通讯。

2) SD 卡接口：该功能暂时保留。

3) RESET 拨码：用于实现恢复出厂设置；恢复出厂设置功能可通过多次切换 RESET 拨码状态，实现 5 秒内拨动次数超过 5 次则认为触发恢复出厂设置，触发后，需将控制器断电后即能恢复到出厂设置；恢复出厂设置效果：恢复出厂设置参数和状态，配置文件删除，配成出厂默认配置；

4) Ethernet 网口：用于和 PC 的通讯；

5) Ethercat 扩展口：EMC-E3000/5000 系列总线运动控制器支持 EtherCAT 扩展从站，最大轴数支持 64 轴，最大节点数支持 128；

6) 状态指示灯：

POWER：用于表示电源状态，电源正常：亮，异常：灭；

RUN：用于系统是否正常运行指示；系统正常运行：RUN 灯闪烁；系统异常：RUN 灯常亮或者常灭；

ERR：表示当前报警状态；正常时：常灭；报警状态下：常亮；

CRUN：CAN 总线运行状态（保留）；

CERR：CAN 总线报错状态（保留）；

BAT：表示电池供电错误状态；供电不足，需要替换：常亮；电池供电正常：常灭

7) RS485 接口：EMC-E3000/5000 系列总线运动控制器支持两路 RS485 接口，支持 Modbus RTU 及自由通讯协议；

8) CAN 接口（保留）/RS232 接口；

9) 电源接口：EMC-E3000/5000 系列总线运动控制器采用 24V 直流电源进行供电，接口采用 3 PIN 插拔式端子；

10) IO 指示灯：输入/输出有效：指示灯亮；输入/输出无效：指示灯灭；

11) 通用输入输出端子；

12) 模块扩展接口：背板总线扩展接口，可扩展我司 R2 系列插片式模块，最大扩展模块数量为 32 个；

3.3 通用 I/O 接口电路

3.3.1 通用数字输入信号接口

EMC-E3000/5000 系列运动控制器本体有 16 路通用数字输入信号（其中 INT0~INT11 为高速输入）。所有输入接口均加有光电隔离元件，可以有效隔离外部电路的干扰，以提高系统的可靠性。通用数字输入信号接口原理图如图 3.4 所示。

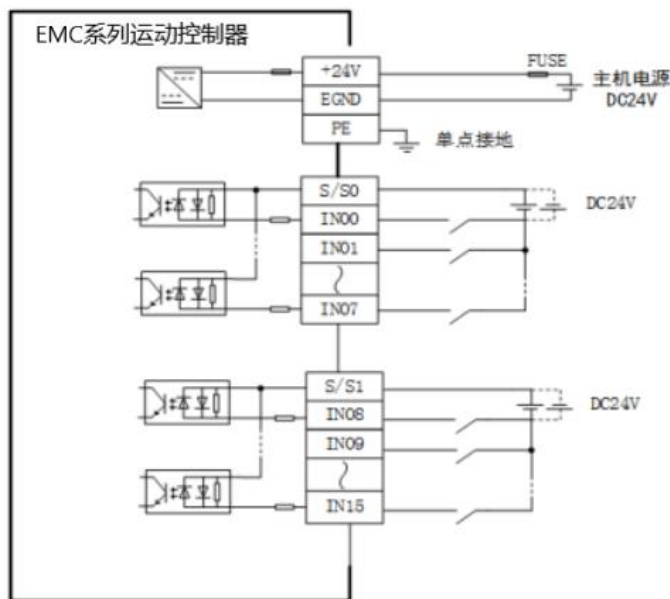


图 3.4 通用输入信号接口原理图

3.3.2 通用数字输出信号接口

EMC-E3000/5000 系列运动控制器有 16 路通用数字输出信号（其中 OUT0~OUT11 为高速输出），由 MOS 管驱动，其最大工作电流为 500 mA（5~24Vdc，吸入），可用于控制继电器、电磁阀、信号灯或其它设备。

下面给出了通用数字输出信号接口控制几种常用元器件的接线图。

1、发光二极管通用数字输出端口控制发光二极管时，需要接一限流电阻 R ，限制电流在 10mA 左右，电阻需根据使用的电源来选择，电压越高，使用的电阻值越大。接线图如图 3.5 所示。

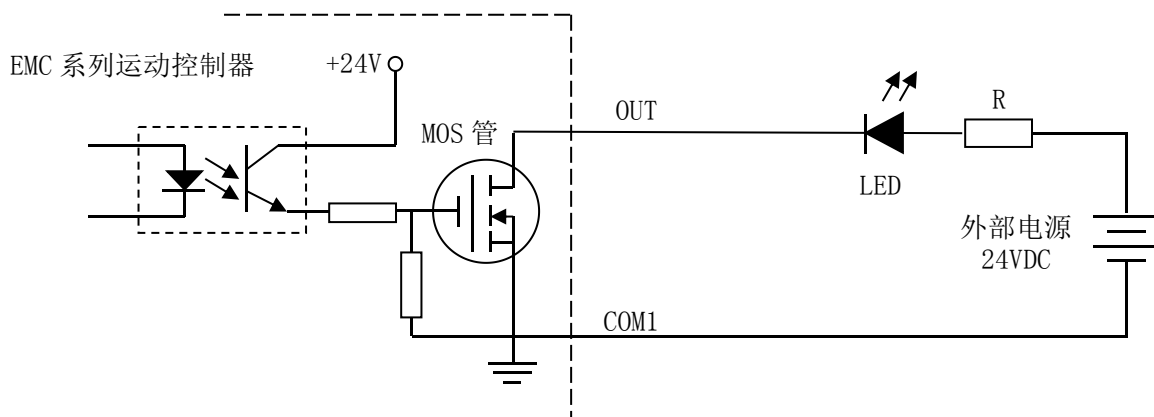


图 3.5 EMC-E3000/5000 系列运动控制器输出接口接发光二极管接线图

2、灯丝型指示灯

通用数字输出端口控制灯丝型指示灯时，为提高指示灯的寿命，需要接预热电阻 R，电阻值的大小，以电阻接上后，输出为 1 时，灯不亮为原则。接线图如图 3.6 所示。

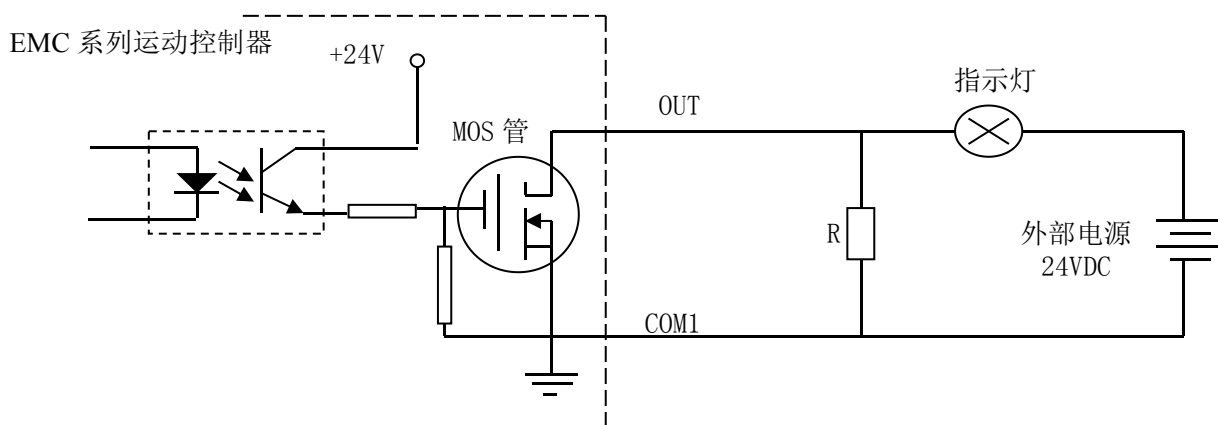


图 3.6 EMC-E3000/5000 系列运动控制器灯丝型指示灯接线图

3、小型继电器

继电器为感性负载，必须并联一个续流二极管。当继电器突然关断时，继电器中的电感线圈产生的感应电动势可由续流二极管消耗，以免 ULN2803 或 MOS 管被感应电动势击穿。其接线图如图 3.7 所示。

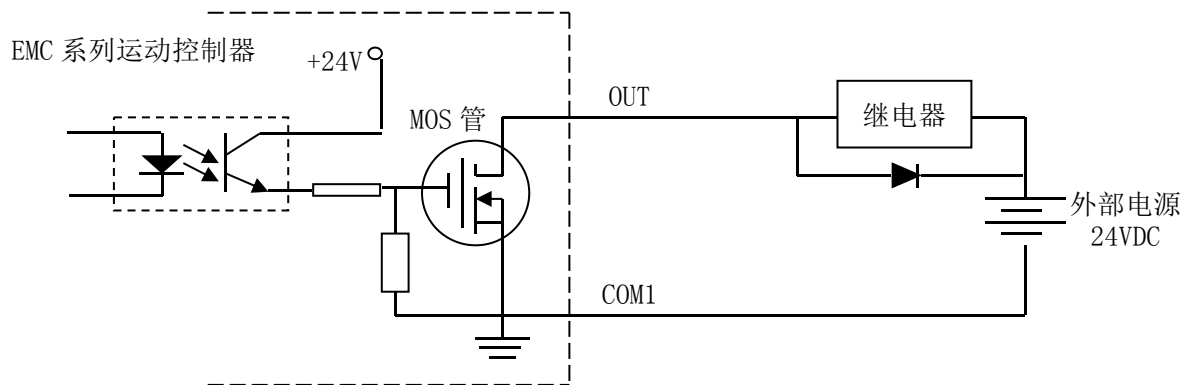


图 3.7 EMC-E3000/5000 系列运动控制器连接小型继电器接线图

注意：在使用通用数字输出端口时，切勿把外部电源直接连接至通用数字输出端口上，否则会损坏输出口。

第 4 章 控制器 Motion 的使用方法

用户在使用 VB、VC 或其它高级语言编写应用程序之前，可利用控制器 Motion 软件快速熟悉 EMC-E3000/5000 系列运动控制器的硬件、软件功能，还可以方便快捷地测试电机、传感器、开关元件、平台等在执行各种动作时的性能特点。

此外，在编程控制之前，需用 Motion 扫描从站并进行总线配置。

4.1 控制器 Motion 的使用步骤

安装控制器 Motion 测试软件的步骤如下：

1) 使用控制器 Motion 前，请确保已经通读本手册。

2) 启动 PC 机，进入 Windows 操作系统。

3) 将 EMC-E3000/5000 系列运动控制器资料包中的“\Motion\控制器 Motion”，全部复制到硬盘中。

4) 在确保控制器 IP 地址与 PC 机 IP 地址处于同一网段的条件下，复制到硬盘中的控制器 Motion 目录中找到 Leadshine.EMC.IDE.exe，双击运行。在 Motion 首次解压时，打开 Motion 后需点击功能选项，勾选连接设置输入控制器地址进行连接。之后打开 Motion 都会自动与控制器通讯上。如图 4.1 与 4.2 所以示。



图 4.1 演示软件主界面



图 4.2 演示软件 IP 地址连接

通讯成功后便如图 4.3 所示的演示软件主界面。在此界面上可以通过点击相应按钮，进入各功能界面。



图 4.3 演示软件主界面

- 5) 选择所需的控制器，然后进行相应的操作。
- 6) 至此，即可使用控制器 Motion 软件进行测试。

注意：本软件是基于 .NET 4.0 开发，请在安装软件前确保操作系统内已经安装了 .NET 4.0，若未安装请先安装。

4.2 Motion 软件的功能与使用方法

当使用 EMC-E3000/5000 系列运动控制器时，控制器 Motion 提供了参数设置、IO 检测、运动测试、帮助这四个主要的操作界面。设置好界面上的参数后，就可以进行一些基本的控制操作，如：点位运动、直线插补、圆弧插补、I/O 信号检测等。

控制器 Motion 软件主要用于设备扫描配置以及功能测试使用，界面简单，操作方便。

软件主界面主要分为 5 个部分：

(1) 菜单栏，用于 Motion 软件的设置、刷新列表和紧急停止。

(2) 控制器列表窗口，控制器列表区的控制器列表是在软件启动时先扫描 PC 系统上已经插入的控制器并显示在列表上。轴列表会显示控制器上已经连接的轴，插补列表和功能列表根据控制器的不同而有所不同，设备列表会显示已经加入 EtherCAT 网络的设备。

(3) 功能页面窗口。

(4) 信息输出窗口，支持所有函数调用过程的信息输出。

(5) 状态栏，显示总线的状态，会根据控制器列表区选择的控制器类型不同而有所不同。

主界面如下图：



图 4.4 主界面

各功能详细使用方法见 Motion 使用手册。

4.2.1 EtherCAT 总线配置

4.2.1.1 设备总线配置

EMC 系列总线控制器 EMC-E3000/5000 系列在使用前必须配置总线参数,下载总线轴 PDO 配置和轴 IO 映射。不同的 PDO 对象组合,可以实现不同的功能, PDO 常用对象可参见附录 3, EMC 系列总线控制器 CSP 模式最基本的 PDO 对象如下表:

主索引 (HEX)	子索引 (HEX)	对象名称	数据类型	PDO 映射
603F	00	错误码(Error Code)	UINT16	T_PDO
6040	00	控制字(Controlword)	UINT16	R_PDO
6041	00	状态字(Statuword)	UINT16	T_PDO
6060	00	操作模式(Modes of operatIOn)	INT8	T_PDO
6061	00	操作模式显示(modes of operatIOn display)	INT8	R_PDO
6064	00	实际位置(positIOn actual value)	INT32	T_PDO
607A	00	目标位置(Target positIOn)	INT32	R_PDO
60FD	00	数字输入(Digital inputs)	UINT32	T_PDO

从站 PDO 对象的删减可以通过 Motion 操作,但需根据从站实际需要的 PDO 对象合理的增减,下面介绍 EMC 系列总线控制器连接从站的操作方法。

在控制器列表区,点击“设备【EtherCat】”按钮进入总线配置界面。总线参数可以手动配置也可以自动配置,一般自动配置总线即可。扫描完成后可以手动修改 PDO 对象。



图 4.5 总线配置界面

1) 自动配置总线

总线配置可以通过扫描来自动匹配完成总线结构配置，在扫描前需要把总线上的从站设备都接好并上电，然后右键点击主站“EtherCATSuite Master Unit”，选择“扫描设备”进行扫描。

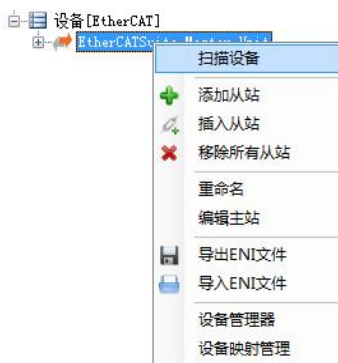


图 4.6 扫描设备菜单

当扫描完成后，主站节点下会显示总线上所有的从站。

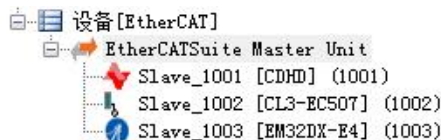


图 4.7 从站列表

2) 手动配置总线

在主站右键菜单选择“添加从站”或“插入从站”菜单项可以添加或插入从站到该主站下。

在从站右键菜单选择“插入从站”“移除从站”可以在当前位置插入从站设备或移除当前选择的从站设备。

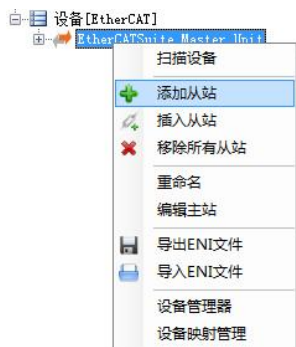


图 4.8 从站操作菜单

3) 配置从站 PDO 参数

设备厂商提供的设备描述文件中已经有默认的 PDO 配置，但实际应用过程如果需要用到额外的参数，就需要对 PDO 配置进行修改和配置。若用户需要修改从站 PDO 参数，可以双击列表中的从站名称，即可显示从站相关配置项，修改从站过程数据的 PDO 对象。在配置完相关参数后，需将配置文件下载到控制器中。

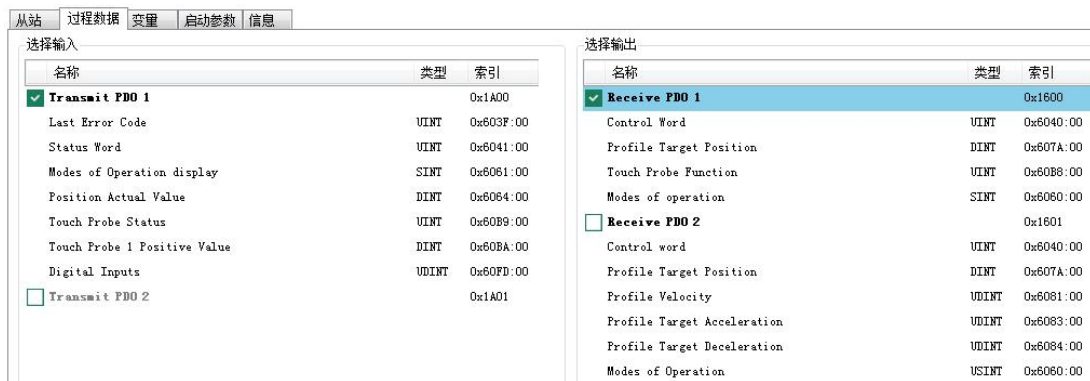


图 5.9 从站参数

4) 配置主站参数

双击选择主站名称，即可显示主站参数项。



图 4.10 主站编辑

5) 配置映射

轴映射指的是建立 EtherCAT 驱动器 (EtherCAT 地址)和 EMC 系列总线控制器中的轴号的对应关系。映射控制器的轴号到从站的轴和控制器的 IO 索引到从站的 IO 索引，映射完成后就可以像操作本地轴和本地 IO 一样操作从站的功能。

默认扫描完成或添加从站时映射关系软件已经自动配置完成，注意：控制器的本地轴和本地 IO 默认被映射到最前面。

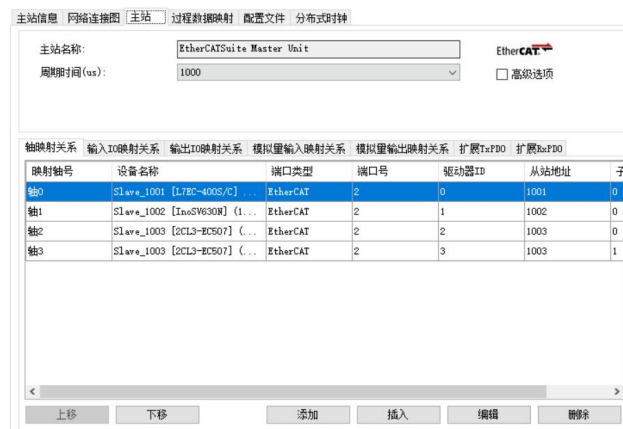


图 4.11 映射配置

⚠注意：对于 Motion 没有自动映射轴号的驱动器设备(设备描述文件没有集成到 EMC 系列总线控制器 Motion 软件中)，用户需要手动添加该驱动器和轴号的映射关系。

6) 配置下载

在配置完成从站和主站以及映射关系后，需要把配置文件下载到控制器。

点击“下载文件”完成配置文件下载，若软件未自动复位控制器，需要手动点击“复位系统”来复位重启控制器。至此，EtherCAT 总线配置完成，可以像操作脉冲控制器一样操作了。



图 4.12 下载配置文件

7) 总线状态

在主界面右下角状态栏显示总线状态以及本地总线状态，若出现错误将显示总线错误码，点击该错误码，会弹出错误的具体意义对话框。

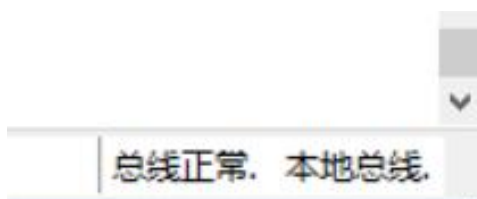


图 4.13 总线状态

注意：总线错误码说明见附录 2。

4.2.1.2 背板总线配置

背板总线在打开 Mtion 或者刷新列表时会自动扫描背板上所连接的扩展模块，用户无需手动进行扫描，且背板总线自动扫描到的配置文件会随 EtherCAT 文件一起进行下载，所以在使用上，用户仍只需要按照 4.2.1.1 的配置步骤进行操作即可，背板总线使用上未新增任何操作步骤。而且 EtherCAT 总线使用同一个配置文件，故只需要在设备总线配置好后，导出配置文件，用户软件下载配置文件即可。具体 EMC 总线配置步骤参照下图：

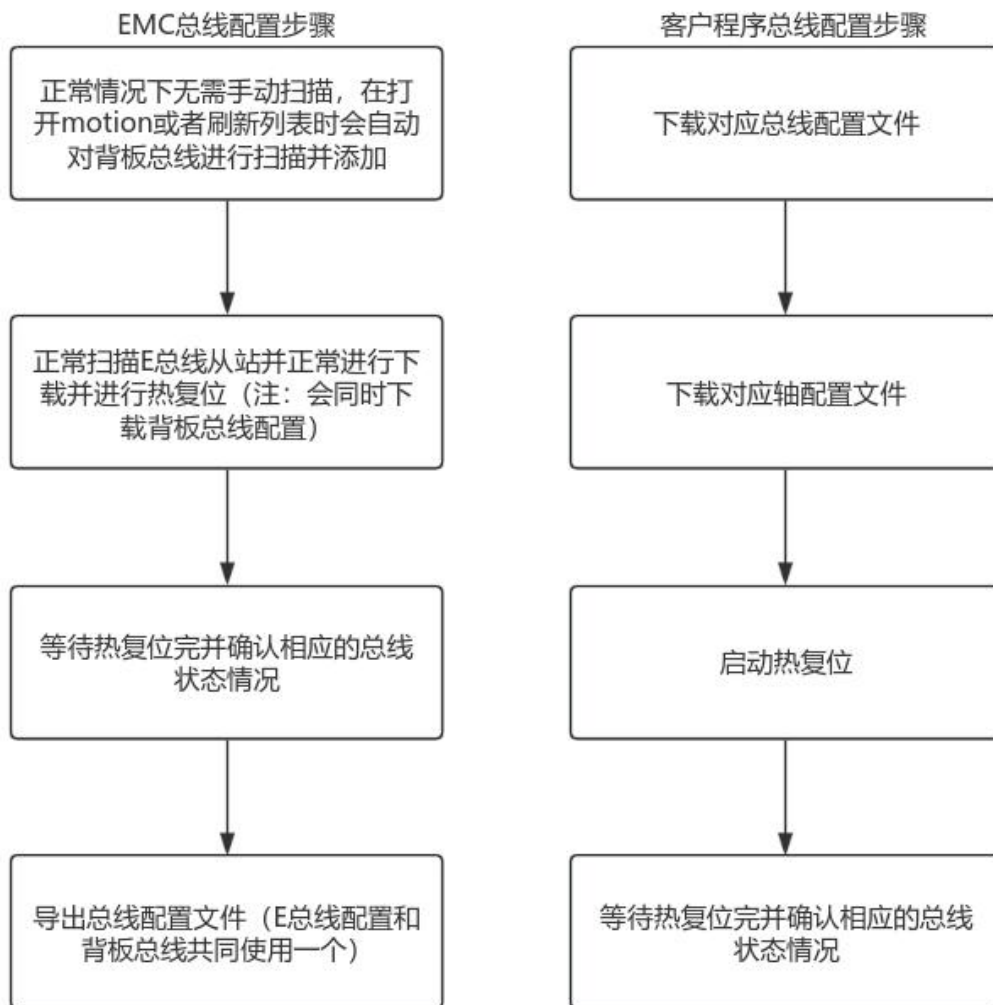


图 4.14 EMC 总线配置流程图

4.2.2 辅助功能

4.2.2.1 重启

当使用控制器出现异常情况时，可以通过复位来实现控制器重启。右键选择对应的控制器，即可出现复位控制器的菜单项。

复位分为两种：冷复位和热复位。

冷复位：重启控制器操作系统，控制器内部所有程序重启，复位速度较慢；

热复位：重启控制器应用程序，重新连接从站，复位速度较快。

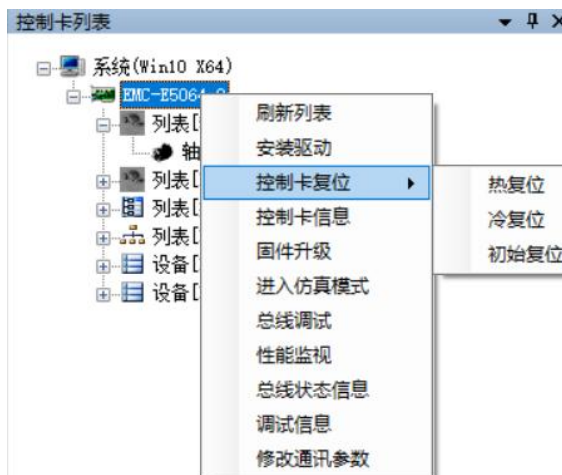


图 4.15 复位控制器菜单

注意：系统中增加或减少从站后都要执行重新扫描配置主从站并下载。

4.2.2.2 控制器信息

当需要了解控制器相关信息时，可右键点击对应的控制器，选择“控制器信息”菜单项。



图 4.16 控制器信息菜单

4.2.2.3 信息输出

控制器 Motion 软件支持所有函数调用过程的信息输出，在菜单栏“窗口”选择“函数输出窗口”即可显示输出窗口。

函数输出窗口中输出了函数的调用名称、参数以及返回值，当返回值为错误码时会以红色标记。

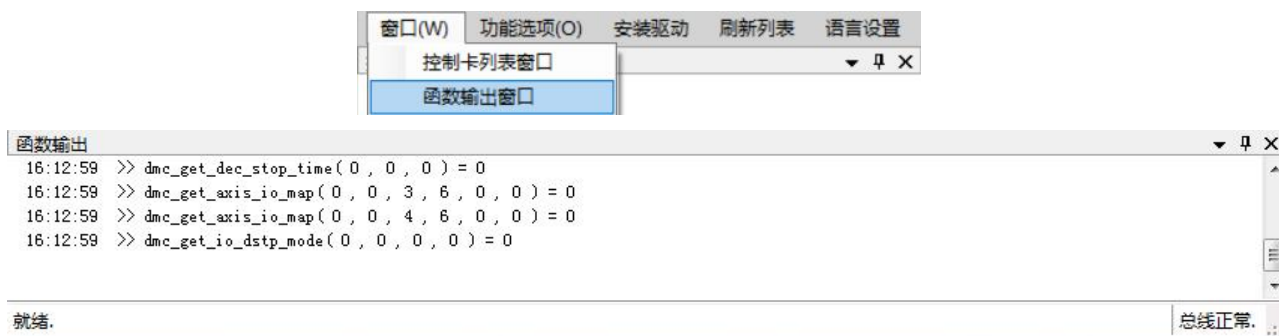


图 4.17 函数输出窗口

4.2.2.4 错误码查询

在函数输出窗口右击，选择“错误码查询”菜单项，即可弹出错误码查询窗口，输入错误码，点击“查询”即可显示错误码对应的信息。

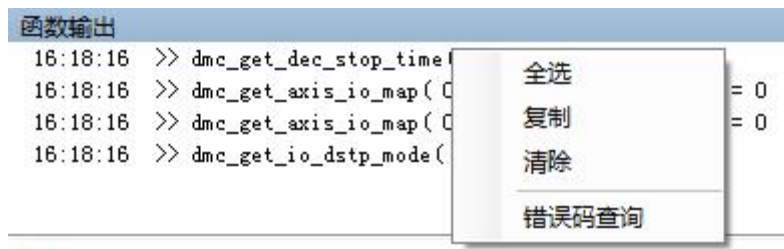


图 4.18 错误码查询菜单



图 4.19 错误码查询窗口

4.2.3 参数配置

当使用控制器前，需要对控制器的轴运动、IO 信号以及伺服信号等相关参数进行合理配置。注意：不同型号控制器，请根据现场情况以及产品使用手册进行合理配置。

选择“控制器列表区”的控制器->展开“列表[轴]”列表->选择要操作的轴->双击“轴 0”按钮->单击“单轴参数”选项页，进入轴参数配置窗口。

在进入参数配置窗口后，可以通过点击“上传”按钮上传控制器内的参数到当前窗口显示，点击“下载”按钮把当前界面的参数下载到控制器；当所有轴参数设置相同时，点击“应用到所有轴”按钮可以把当前轴设置的参数应用到所有轴。窗口打开时默认会上传控制内最新的参数。

注意：不同类型控制器参数项可能有所不同



图 4.20 参数配置窗口

4.2.3.1 轴参数

1) 基本设置

设置各个轴的基本运动参数，包括脉冲当量、初始速度、定位速度、终止速度、加减速时间、S 段时间等参数。

参数名	参数值	单位	说明
基本设置			
脉冲当量	1	pulse/unit	每个单位距离对应的脉冲数量
初始速度	0	unit/s	最大值根据驱动器决定
定位速度	5000	unit/s	最大值根据驱动器决定
终止速度	0	unit/s	最大值根据驱动器决定
加速时间	0.1	s	最小值为 0.001s
减速时间	0.1	s	最小值为 0.001s
S段时间	0	s	范围：0~0.5 s
减速停止时间	0	s	异常减速停止时间 范围：0~∞

图4.21 基本参数设置

脉冲当量：根据控制器所接电机的轴的参数设置对应的当量参数，当量为一个运动单位所需要的运动数量，单位为 pulse/unit。

初始速度：设置单轴运动时，轴运动开始的起步速度，单位为 unit/s。

定位速度：设置单轴运动时，轴运动的最大速度，单位为 unit/s。

终止速度：设置单轴运动时，轴运动结束时的停止速度，单位为 unit/s。

加速时间：设置单轴运动时，轴运动由初始速度加速到最大速度时所需要的时间，单位为 s。

减速时间：设置单轴运动时，轴运动快结束时由最大速度降速到终止速度时所需要的时间，单位为 s。

S 段时间：设置单轴运动时，轴 S 形曲线速度的时间，单位为 s。

减速停止时间：设置单轴运动时，调用减速停止指令或遇到异常信号减速后，减速停止的时间，单位 s。

2) 回零设置

设置各个轴的回零参数，回零参数包括回零低速、回零高速、加减速时间、回零模式。

回零设置			
回零低速 (unit/s)	10		
回零高速 (unit/s)	100		
回零偏移 (unit)	0		
加速时间 (s)	0.1		
减速时间 (s)	0.1		
回零模式	0		

图4.22 回零参数设置

回零低速：回零时低速运动时的速度。

回零高速：回零时高速运动时的速度。

加速时间：回零时由低速到高速的加速段时间。

减速时间：回零时由高速到低速的减速段时间。

回零模式：设置回零的回零方式。

3) 软限位设置

设置各个轴的软限位参数，参数包括：软限位启用、软限位正、软限位负、停止模式、编码器启用等。

软限位设置			
限位启用	是		
软限位正	1000		
软限位负	0		
停止模式	减速停止		
编码器启用	否		

图4.23 软限位参数设置

限位启用：设置软限位是否启用。

软限位正：设置软限位正方向位置，轴运动的最大范围。

软限位负：设置软限位负方向位置，轴运动的最小范围。

停止模式：设置轴触发软限位后，停止的模式。

编码器启用：位置是否是检测编码器反馈位置。

4) 急停设置

设置急停参数设置，参数包括：急停启用、有效电平、急停 IO 映射等。

急停设置			
急停启用	否		
有效电平	低		
急停IO映射	通用输入:0,滤波:0s		

图4.24 急停设置

急停启用：设置急停是否启用。

有效电平：设置急停 IO 信号触发的有效电平。

急停 IO 映射：设置急停 IO 信号映射到其他 IO 上。

5) 减速停止设置

设置减速停止参数，参数包括：信号启用、有效电平、减速停止 IO 映射等。

减速停止设置			
信号启用	否		
有效电平	低		
减速停止IO映射	通用输入:0,滤波:0s		

图4.25 减速停止设置

信号启用：设置是否启用减速停止信号。

有效电平：设置减速停止信号触发电平。

减速停止 IO 映射：设置减速停止 IO 信号映射到其他 IO 信号上。

4.2.3.2 插补参数

选择“控制器列表”区的控制器->展开“列表[插补系]”列表->选择要操作的插补坐标系->双击“插补系0”按钮->单击“插补系参数”选项页，进入插补系参数配置窗口。

插补运动 插补系参数			
应用到所有插补系 上传 下载			
参数名	参数值	单位	说明
插补设置			
起始速度 (unit/s)	0	unit/s	最大值根据驱动器决定
插补速度	10000	unit/s	最大值根据驱动器决定
终止速度 (unit/s)	0	unit/s	最大值根据驱动器决定
加速时间	0.5	s	最小值为 0.001s
减速时间	0.5	s	最小值为 0.001s
S段时间	0	s	范围: 0~0.5 s
减速停止时间(s)	0.01	s	异常减速停止时间 范围: 0~∞

图4.26 插补参数

注意：插补参数中所有的速度都是坐标系内的各个轴的合成速度。

起始速度：设置插补运动的起始速度。

插补速度：设置插补运动的最大速度。

终止速度：设置插补运动的停止速度。

加速时间：设置插补运动由起始速度到最大速度的加速时间。

减速时间：设置插补运动由最大速度到停止速度的减速时间。

S 段时间：设置插补运动 S 形速度曲线的时间。

4.2.4 功能测试

测试控制器各项功能，包括单轴测试、多轴测试、PVT 测试、单轴比较、二维比较、原点锁存、连续插补、PWM 测试、AD 测试、DA 测试等。注意：不同类型的控制器可能测试项不同。

4.2.4.1 单轴测试

测试各个轴的定长、定速以及回零运动，支持在线变速和在线变位运动，监视各个轴的运动状态以及轴状态信息。

单轴运动
单轴参数
总线轴IO

运动轴： 轴0

I/O状态

ORG ALM ELP ELN SLP SLN SEVON

轴状态

指令位置： 0
反馈位置： 0
当前速度： 0
运动状态： 停止
运动模式： 空闲
轴类型： EtherCAT
轴状态机： 初始化完成状态
轴错误码： 16# 0

参数设置

运动距离： 1000 unit
起始速度： 0 unit/s
运动速度： 5000 unit/s
停止速度： 0 unit/s
加速时间： 0.1 S
减速时间： 0.1 S
S段时间： 0 S
运动方向： 正方向

模式设置

运动模式： 定长运动
停止模式： 减速停止
坐标模式： 相对坐标

☐ 位置-时间曲线
☐ 速度-时间曲线

运动测试

启动 停止 紧急停止 在线变位 在线变速 清除错误 所有轴使能开
计数清零 <-- --> SEVON 所有轴使能关

图4.27 单轴测试界面

一般操作步骤:

- 1、设置模式，设置好运动模式及其他参数，注意：不同运动模式参数项不一样。
- 2、设置参数。
- 3、设置曲线显示，勾选需要采集的曲线进行曲线显示。
- 4、点击“启动”按钮，即开始运动。

1) 模式设置

运动模式：选择测试运动的模式，包括定长运动、连续运动以及回零运动。

停止模式：设置点击“停止”按钮时，停止是减速停止还是立即停止。

位置类型：设置参数项中的目标位置是绝对定位还是相对定位。

2) 参数项

运动轴选择：勾选需要参数运动的轴。

起始速度：定位运动起始的速度。

最大速度：定位运动达到的最大运动速度。

终止速度：定位运动结束运动时的速度。

加速时间：定位运动从起始速度到最大速度的加速段时间。

减速时间：定位运动从最大速度到停止速度的减速段时间。

S 段时间：S 曲线速度段的时间。

目标位置：设置定位运动的运动到的目标位置，若位置坐标类型设置为“相对坐标”则该值为相对于当前位置的相对距离。

运动方向：设置“相对坐标”模式下，相对定位方向。

回零低速：回零运动低速时的速度。

回零高速：回零运动高速时的速度。

回零方向：回零运动的方向，正方向/负方向。

回零模式：设置回零的模式，具体模式请参照对应型号的控制手册。

回零速度模式：设置回零时低速回零还是高速回零。

3) 图形显示

位置-时间曲线：采集显示位置与时间关系曲线。

速度-时间曲线：采集显示速度与时间关系曲线。

4) 操作

启动：启动选择的轴运动。

单轴停止：停止选择的轴。

紧急停止：紧急停止所有的轴运动。

计数器清零：设置所有轴指令位置和反馈位置为 0。

在线变速：在轴定位运动时改变选择的轴的运动速度。

在线变位：在轴定位运动时改变选择轴的定位目标位置。

SEVON：使能/失能轴。

4.2.4.2 多轴测试

测试多轴插补运动，包括直线插补、各类型圆弧/螺旋线插补等。

选择“控制器列表”区的控制器->展开“列表[插补系]”列表->选择要操作的插补坐标系->双击“插补系 0”按钮->单击“插补运动”选项页，进入插补运动操作窗口。

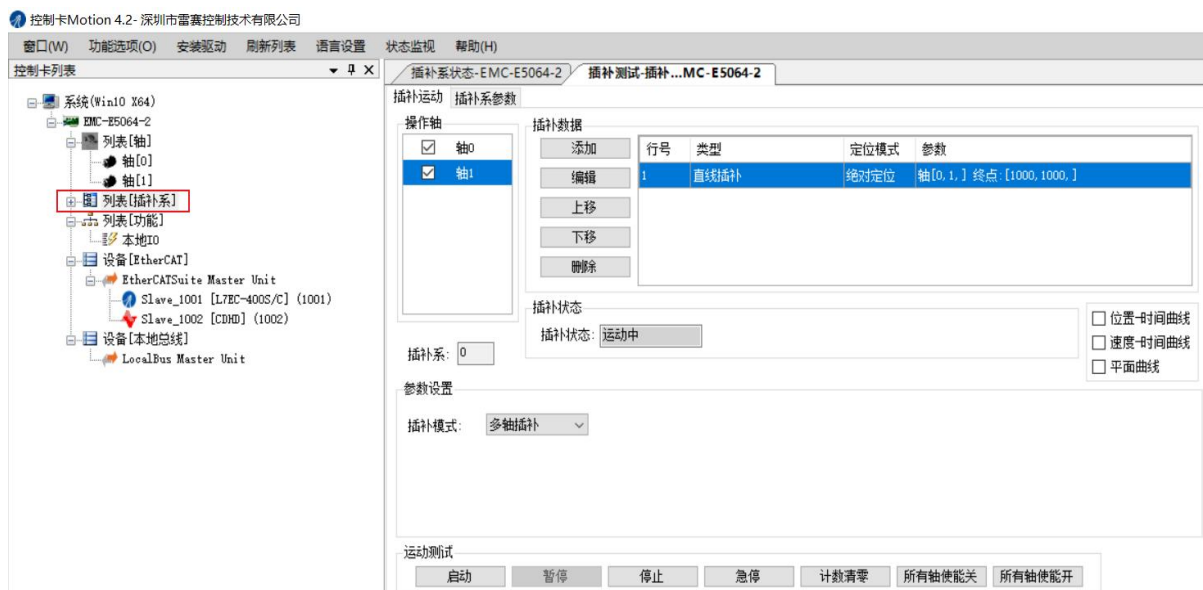


图 4.28 多轴测试界面

一般操作步骤

- 1、使能参与运动的轴，点击“SEVON”使能操作的轴。
- 2、选择操作轴，选择需要参与插补运动的轴号。
- 3、设置插补模式。
- 4、添加插补数据
- 5、轴参数设置，设置多轴插补速度参数或者连续插补速度参数。
- 6、图形显示，勾选需要显示的曲线。
- 7、点击“启动”按钮，启动运动。

1) 插补模式

设置插补运动是连续插补或者多段插补。选择连续插补后，下方会出现连续插补设置项
在线变速（%）：设置在线变速的百分比

圆弧限速：设置是否启用圆弧限速功能

前瞻模式：设置是否启用前瞻模式

前瞻段数：设置前瞻模式的段数

允许误差：设置开启前瞻模式时的允许误差范围

前瞻加速度：设置前瞻模式的前瞻加速度

PWM 启用：设置是否启用连续插补运动中的 PWM 跟随功能。点击设置，可以进入 PWM 跟随功能参数设置窗口。

2) 添加插补数据操作过程

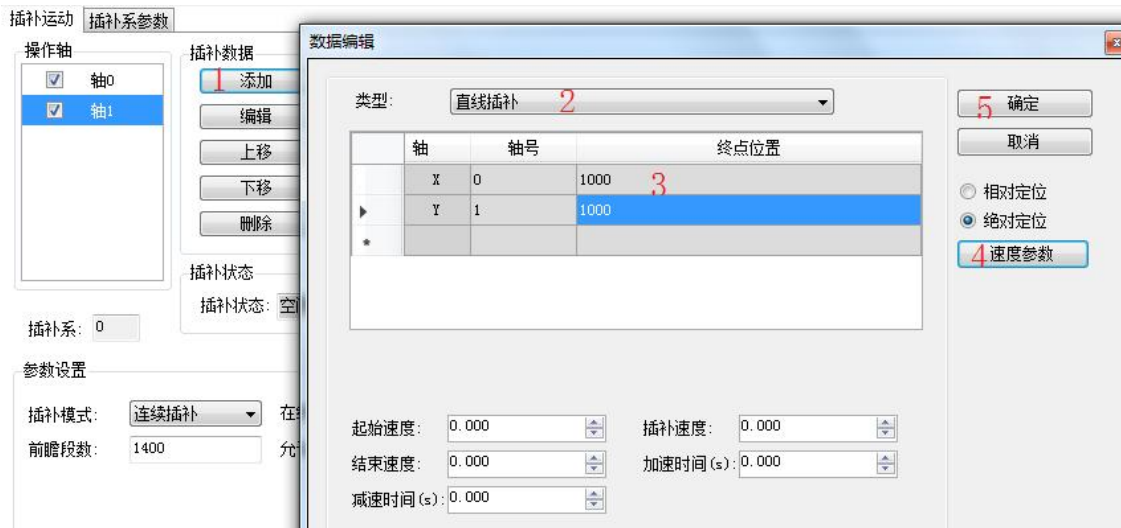


图 4.29 添加插补数据界面

4.2.4.3 单轴比较

测试单轴比较。在控制器列表去选择控制器->展开“列表[功能]”列表->双击“一维比较”按钮，进入一维比较测试窗口。



图 4.30 单轴比较测试界面

一般操作步骤

- 1、添加数据表。
- 2、选择测试轴，设置轴速度相关参数。
- 3、启动运动。

比较数据编辑

- 1) 添加数据，右键菜单选择“添加比较点”菜单项，在编辑窗口中添加数据，确定即可。

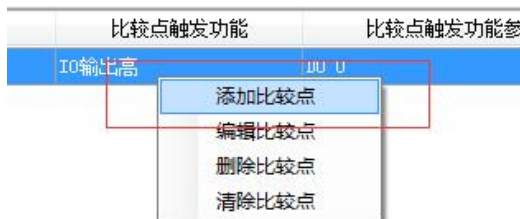


图 4.31 添加比较点菜单



图 4.32 比较点编辑窗口

- 2) 编辑比较点，右键菜单选择“编辑比较点”菜单项或双击需要编辑的行，再在编辑窗口编辑数据确定即可。

- 3) 删除比较点，右键菜单选择“删除比较点”菜单项，即可删除选择的比较点数据。

- 4) 清除比较点，右键菜单选择“清除比较点”菜单项，即可清除所有的比较点数据。

比较状态监视

实时显示比较器的当前比较位置、已比较点数和可加比较点数，同时实时显示比较数据图。



图 4.33 比较状态

第 5 章 应用软件开发方法

EMC-E3000/5000 系列总线运动控制器的应用软件可以在 Visual Basic 、 Visual C++、 C# 等高级语言环境下开发。应用软件开发之前，需保证 EMC-E3000/5000 系列连接好从站，通过控制器 Motion 的 EtherCAT 总线配置界面扫描从站、设置总线通信周期，并下载总线配置文件，操作过程详见 4.2.1 节。

如果您对 VB、VC、C#语言都不熟悉，建议您花两天时间阅读一本 VB 语言的培训教材，并且通过练习掌握该语言的基本技巧，如：编写简单的程序、创建窗体和调用函数。

如果您曾用 VB 或 VC 等程序语言开发过运动控制软件，并具有丰富的经验，则可直接阅读第 7 章“总线操作函数说明”及第 8 章“基本功能函数说明”。

5.1 基于 WINDOWS 平台的应用软件结构

使用雷赛控制器的自动化设备运动控制系统构架如图 5.1 所示：

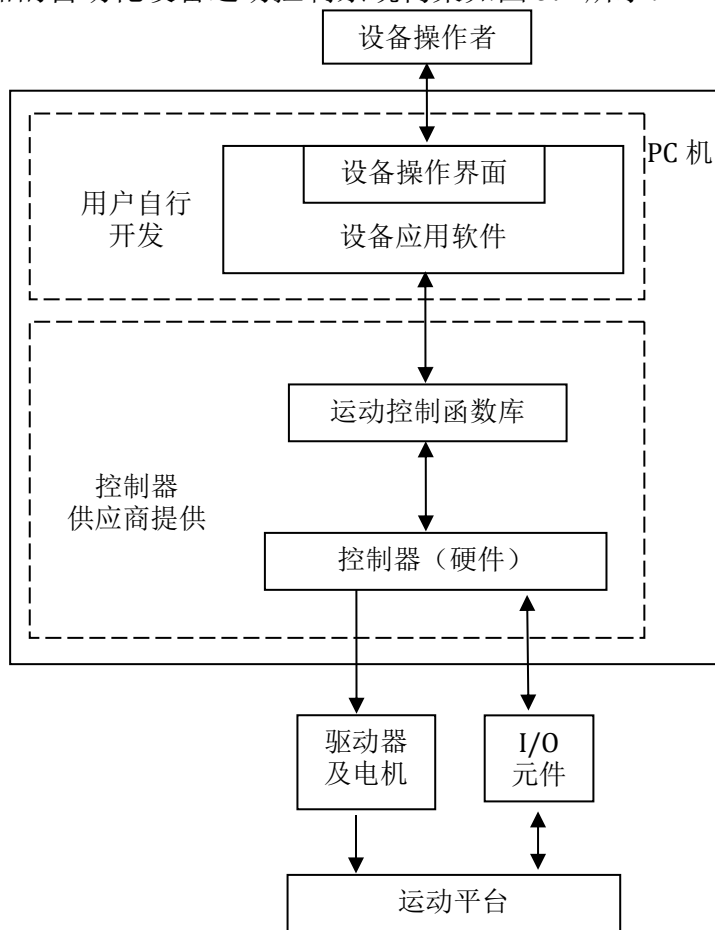


图 5.1 基于雷赛控制器的自动化设备运动控制系统构架

从图 5.1 中可看出，运动控制系统的工作原理可以简单描述为：

- 1) 操作员通过操作界面（包括显示屏和键盘）将指令信息传递给设备应用软件；
- 2) 设备应用软件将操作者的信息以及应用软件中已有的运动流程、运动轨迹等数据转化为运动参数，并根据这些参数调用 DLL 库中运动函数；
- 3) 运动函数通过工控机向控制器发出控制指令；
- 4) 控制器根据控制指令发出相应的指令脉冲给驱动器及电机、读写通用输入输出口、读取编码器数据。

用户根据设备的工艺流程、运动轨迹和友好的人机界面等要求开发设备应用软件。雷赛公司已提供支持 EMC-E3000/5000 系列总线运动控制器的 DLL 运动函数库，包括控制器初始化、单轴及多轴控制、数字量输入/输出控制等多种函数。这些函数可以方便地完成与运动控制相关的功能，用户不需要更多了解硬件电路的细节以及运动控制和插补算法的细节，就能使用 VB、VC 等程序语言开发出自己的运动控制系统应用软件。

用户编写的设备应用软件的典型流程如图 5.2 所示。

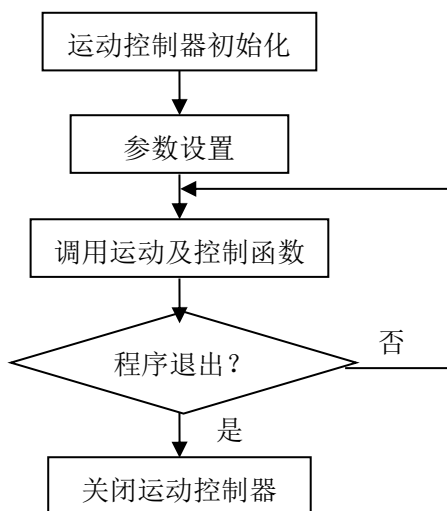


图 5.2 设备应用软件的典型流程

5.2 采用 VB 6.0 开发应用软件的方法

下面以 Visual Basic6.0 环境下编写一个点位运动的应用软件为例，讲解用 VB 开发应用软件的一般方法。

- 1) Motion 软件中，扫描驱动器，并将轴使能。在磁盘上新建一个目录，如 E:\test1
- 2) 打开 Visual Basic 6.0，新建一个“标注 EXE”工程，在对话框上添加按钮“启动”和“停止”，并将其名称分别修改为“CB_Start”和“CB_Stop”，如图 5.3 所示。

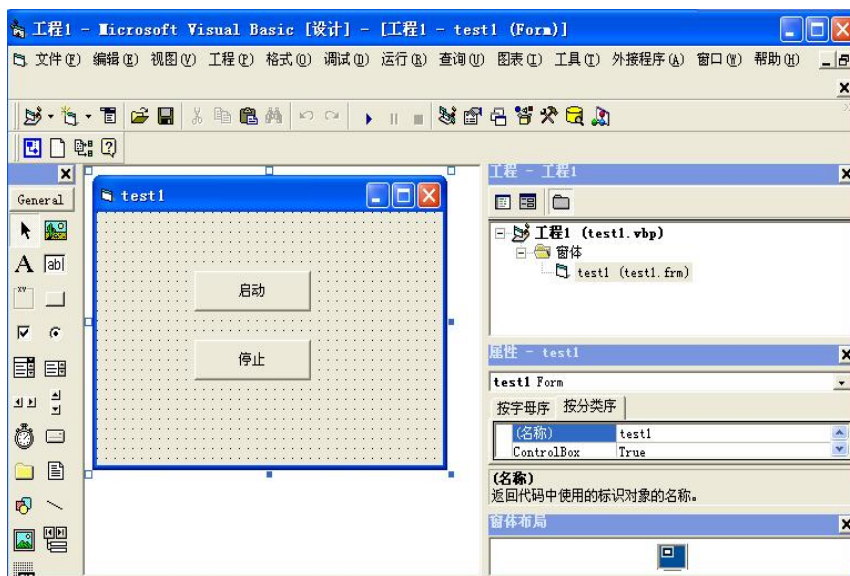


图 5.3 修改对话框（VB）

3) 工程保存在 E:\test1 目录下。

4) 在资料光盘相应目录下找到 LTDMC.bas、LTDMC.dll 和 PVT.dll 文件，拷贝到 test1 目录下。

5) 菜单中选择“工程”->“添加模块”->“现存”，找到 test1 目录下的 LTDMC.bas 文件，添加到工程中，如图 5.4 所示。

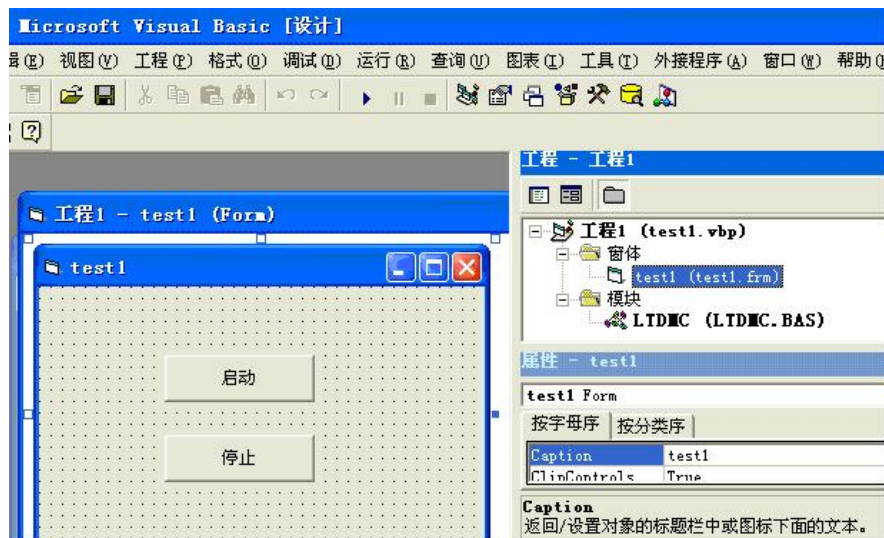


图 5.4 添加头文件

6) 双击窗口控件，在 Form_Load 事件中添加代码 dmc_board_init_eth。选择 UnLoad 事件，在 Form_Unload 事件中添加代码 dmc_board_close 双击“启动”按钮，在 CB_Start_Click 事件中添加代码如下：

```
dmc_set_profile_unit 0,0,500,5000, 0.01,0.01,500
dmc_pmove_unit 0,0,200000,0
```

双击“停止”按钮，在 CB_Stop_Click()事件中添加代码如下：

```
dmc_stop 0,0,0
```

7) 程序编写完成。运行程序，显示界面如图 5.5 所示。按下“启动”按钮，第 0 轴就会输出长度为 200000 的脉冲；运动中可以按下“停止”按钮，便会减速停止脉冲输出。



图 5.5 程序运行界面（VB）

5.3 采用 VC 6.0 开发应用软件的方法

下面以 Visual C++ 6.0 环境下编写一个点位运动的应用软件为例，讲解用 VC 开发应用软件的一般方法。

- 1) 打开 Visual C++ 6.0。
- 2) 新建一个工程。
- 3) 选择 MFC APPWizard(exe)。
- 4) 选择工程保存路径，如：E:\。
- 5) 输入工程名，如：test1。如图 5.6。

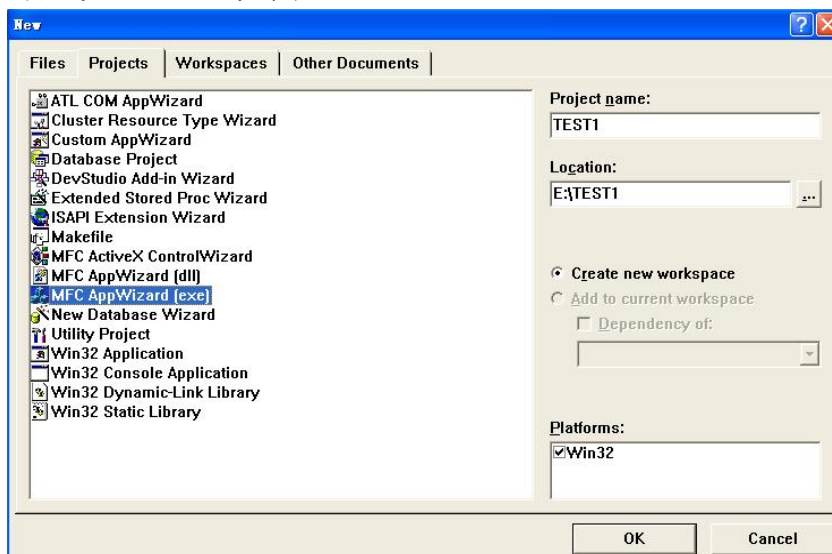


图 5.6 创建新工程

6) 在应用程序类型中选择“基于对话框”，按“完成”键，建立工程。

7) 给对话框进行简单的修改，增加按钮“启动”、“停止”和“使能”；并分别命名为“IDC_BUTTON_Start”、“IDC_BUTTON_Stop”和“IDC_BUTTON_enable”，如图 5.7 所示。

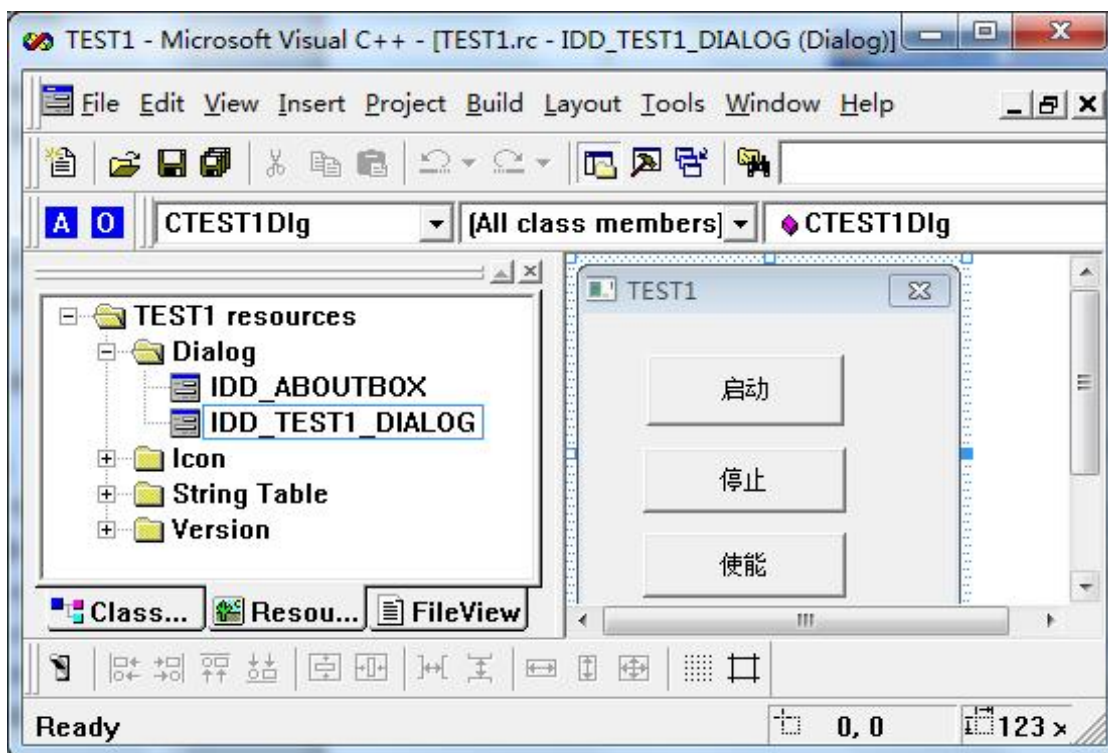


图 5.7 修改对话框

8) 在相应的目录下找到 LTDMC.h、LTDMC.lib、LTDMC.dll，拷贝到 E:\tes1 目录。

9) 在菜单中选择“工程”->“添加工程”->“文件”，选中 LTDMC.lib 文件加入到工程中。

10) 打开 test1.cpp 文件，在程序开始部分添加相应语句：#include “LTDMC.h”，如图 5.8 所示。

11) 在 CTest1Dlg::OnInitDialog()函数中添加代码：dmc_board_init_eth(); 如图 5.9。

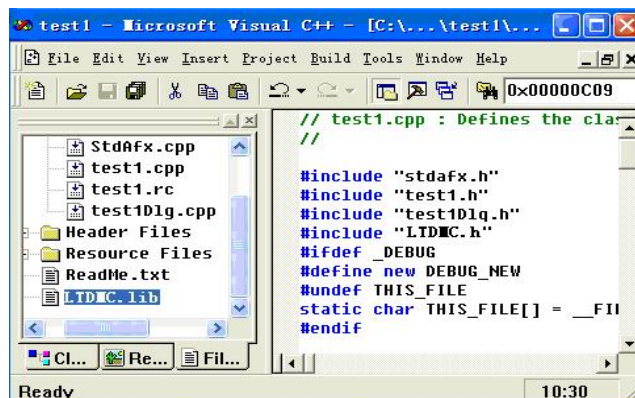


图 5.8 程序增加头文件

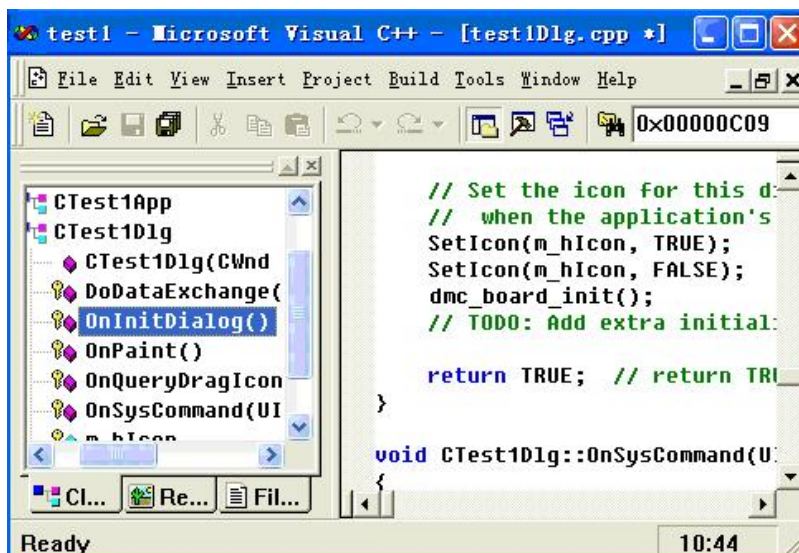


图 5.9 程序增加初始化函数

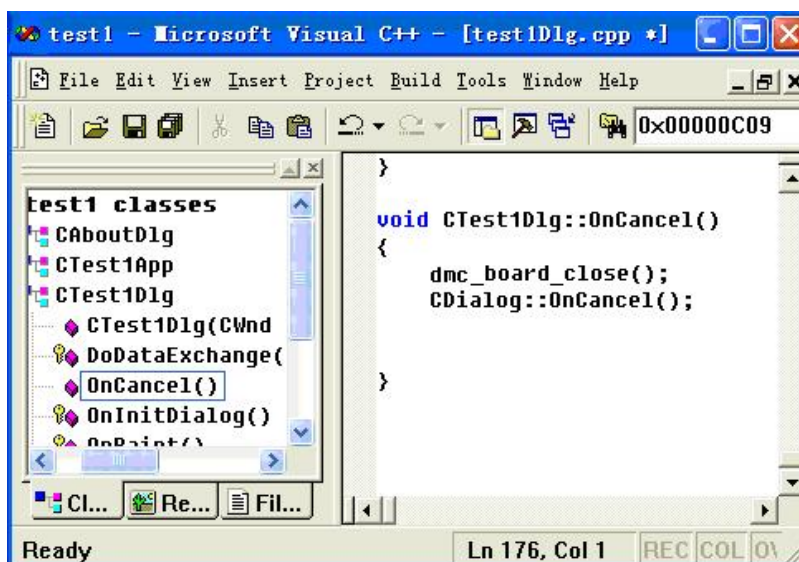


图 5.10 程序增加 OnCancel 函数

12) 如图 5.10 所示, 在 Ctest1Dlg 中添加一个成员函数 OnCancel, 在 OnCancel 函数中添加代码如下:

```
dmc_board_close();
CDialog::OnCancel();
```

13) 双击“启动”按钮在按钮点击事件中输入代码如下:

```
dmc_set_profile_unit(0,0,500,5000, 0.01,0.01,500);
dmc_pmove_unit(0,0,200000,0);
```

双击“停止”按钮在按钮点击事件中输入代码:

```
dmc_stop(0,0,0);
```

双击“使能”按钮在按钮点击事件中输入代码:

```
nmc_set_axis_enable(0,0);
```

14) 编译程序后，运行程序，显示图 5.11 所示的界面。按下“使能”，再按下“启动”按钮，第 0 轴就会输出长度为 200000 的脉冲；运动中可以按下“停止”按钮便会减速停止脉冲输出。



图 5.11 程序运行界面

5.4 采用 C# 开发应用软件的方法

下面以 C# 环境下编写一个点位运动的应用软件为例，讲解用 C# 开发应用软件的一般方法。

1) 在磁盘上新建一个目录，如 E:\Test

2) 打开 C#，新建一个“Windows 窗体应用程序”，并将名称修改为“test1”，如图 5.12 所示。在对话框上添加按钮“启动”和“停止”，并将其名称分别修改为“Start”和“Stop”，如图 5.13 所示。

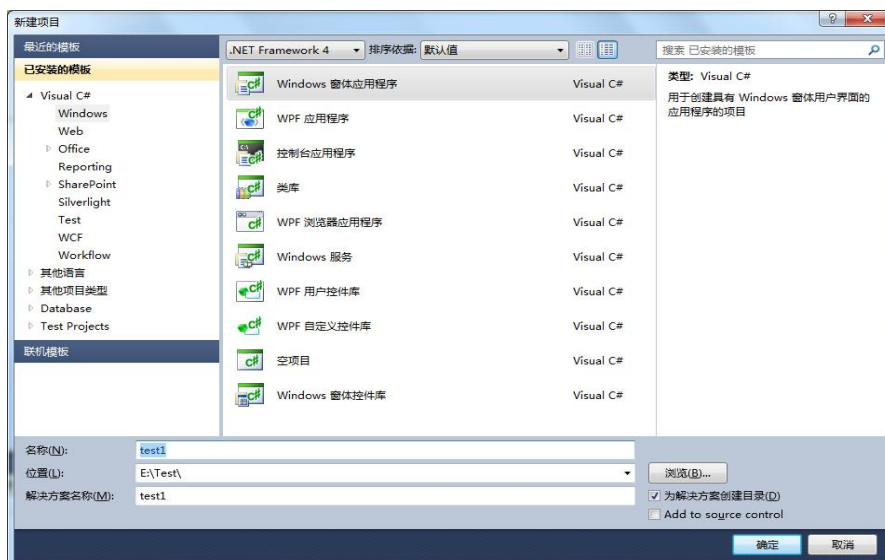


图 5.12 新建 Windows 窗体应用程序

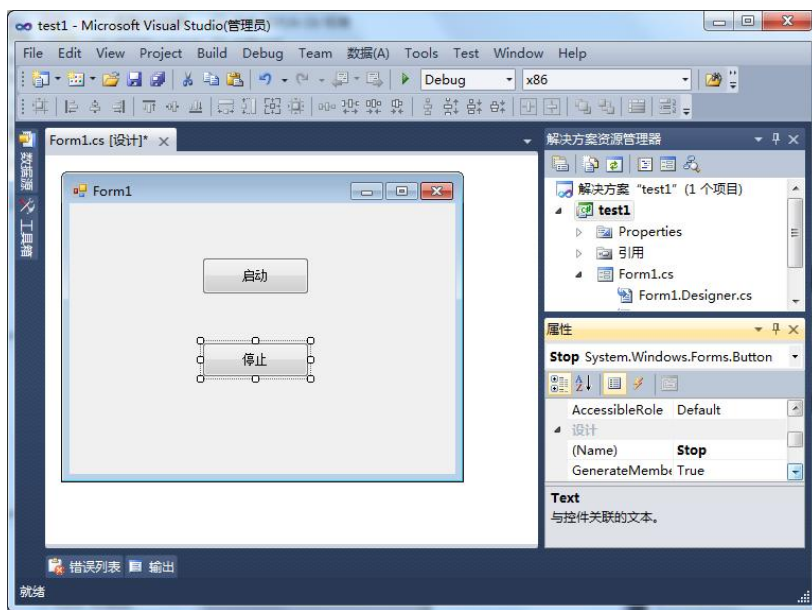


图 5.13 修改对话框

- 3) 点击“全部保存”，将应用程序保存在 E:\Test 目录下。
- 4) 在资料光盘相应目录下找到 LTDMC.dll 和 LTDMC.cs 文件，将 LTDMC.dll 文件拷贝至 E:\Test\test1\test1\bin\debug 目录下，LTDMC.cs 文件拷贝至 E:\Test\test1\test1 目录下。
- 5) 菜单中选择“项目”->“添加现有项”，找到 test1 目录下的 LTDMC.cs 文件，添加到应用程序中，如图 5.14 所示。

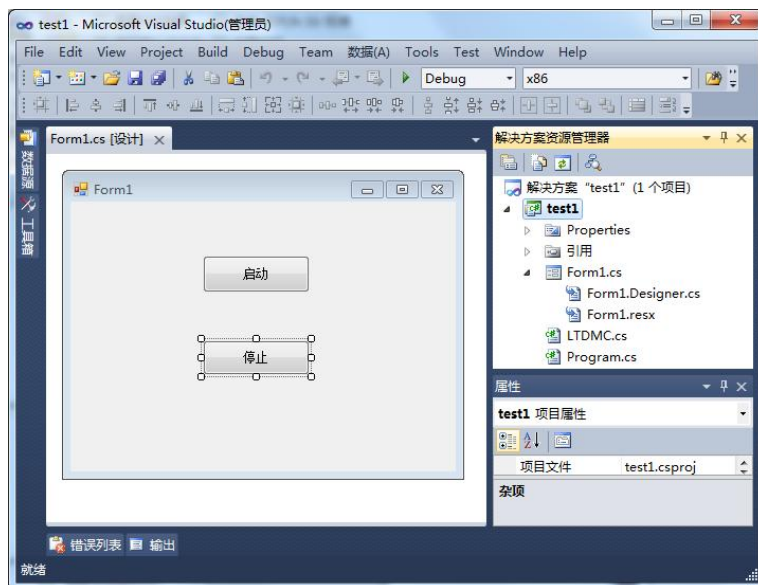


图 5.14 添加头文件

- 6) 在代码文件开头处添加控制器的命名空间：using csLTMC; 如图 5.15 所示。

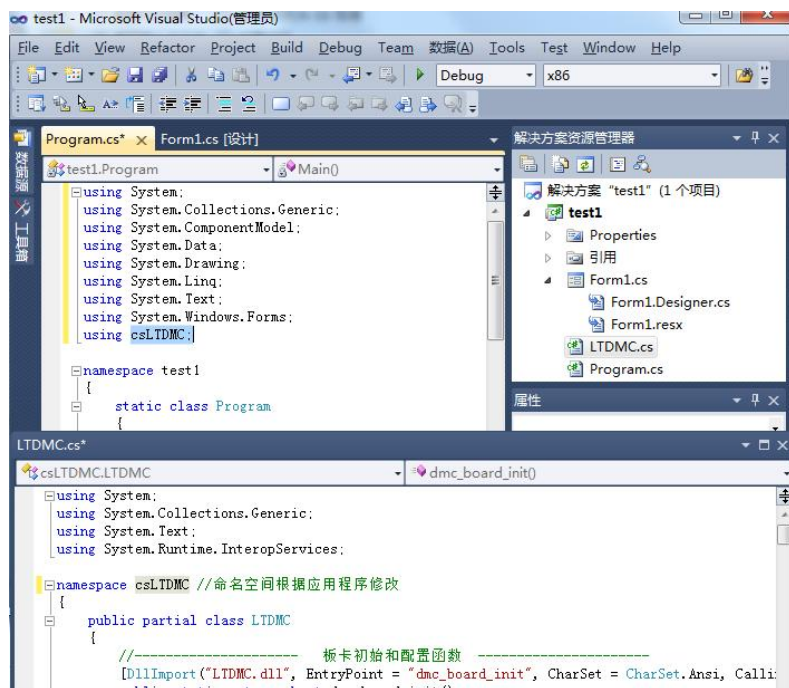


图 5.15 添加控制器的命名空间

7) 设定控制器链接号为 3, 如图 5.16 所示, 双击窗口控件, 在 Form1_Load 事件中添加代码 LTDMC.dmc_board_init_eth(); 双击属性窗口中 FormClosed, 在 Form1_FormClosed 事件中添加代码 LTDMC.dmc_board_close(); 双击“启动”按钮, 在 Start_Click 事件中添加代码如下:

```
LTDMC.nmc_set_axis_enable(3, 0);
```

```
LTDMC.dmc_set_profile_unit(3, 0, 500, 5000, 0.01, 0.01, 500);
```

```
LTDMC.dmc_pmove_unit(3, 0, 200000, 0);
```

双击“停止”按钮, 在 Stop_Click 事件中添加代码如下:

```
LTDMC.dmc_stop (3,0,0);
```

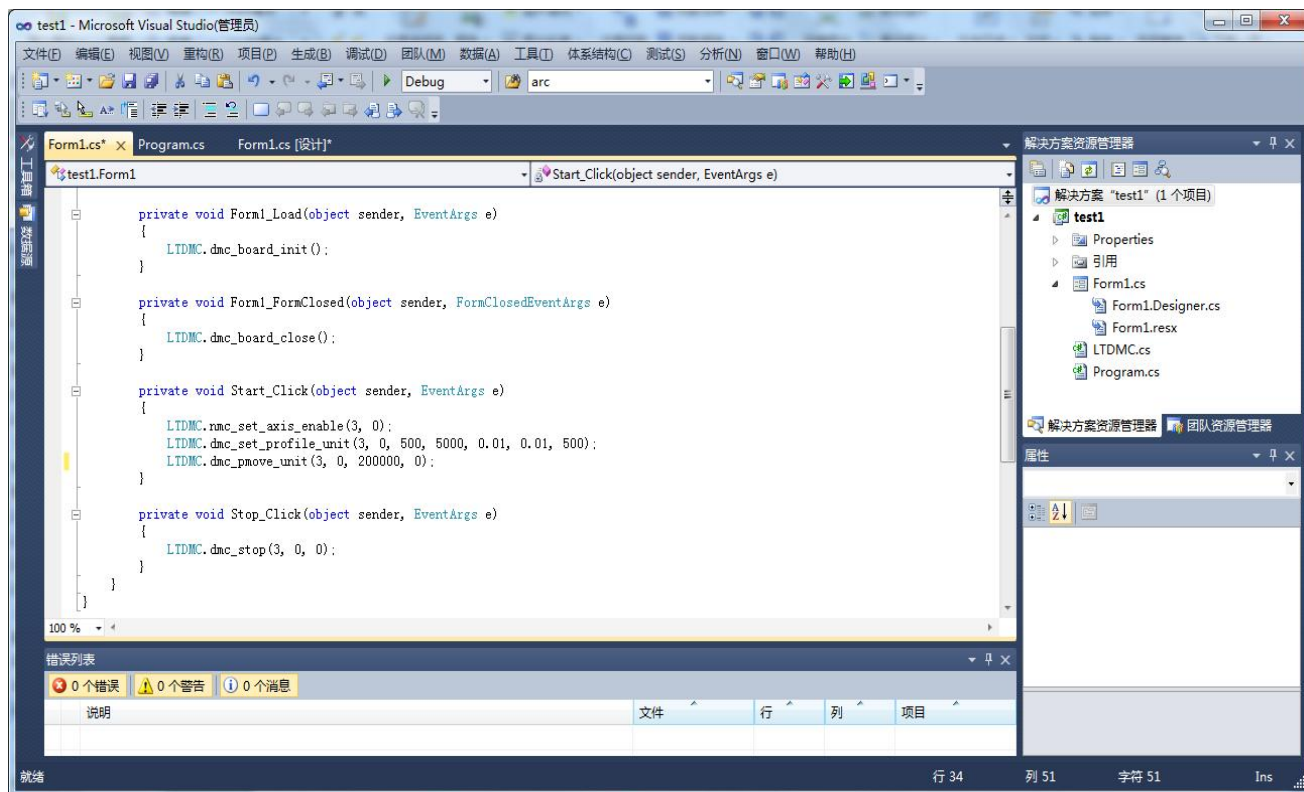


图 5.16 程序中调用控制器库函数

8) 程序编写完成。运行程序，显示界面如图 5.17 所示。按下“启动”按钮，第 0 轴就会走长度为 200000 的距离；运动中可以按下“停止”按钮，便会减速停止脉冲输出。

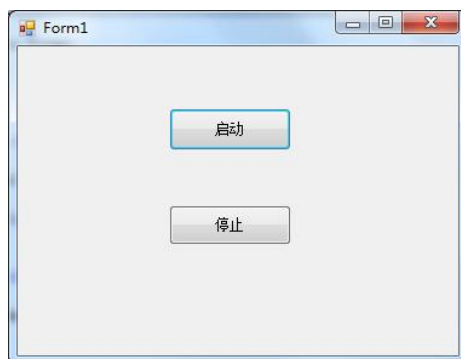


图 5.17 程序运行界面

第 6 章 雷赛控制 EMC 系列总线控制器基本功能实现方法

本章介绍采用 C# 语言通过调用相关函数实现 EMC-E3000/5000 系列运动控制器的基本功能方法。

注意：在编程之前，一定要用控制器 Motion 软件检测硬件系统，确保硬件接线正确。

6.1 控制器初始化

控制器初始化设置相关函数如下表 6.1 所示。

表 6.1 控制器初始化相关函数说明

名称	功能	参考
dmc_board_init_eth	控制器初始化函数	8.1 节
dmc_board_close	控制器关闭函数	

在操作控制器之前，必须调用函数 `dmc_board_init_eth` 为控制器分配资源。同样，当程序结束对控制器的操作时，必须调用函数 `dmc_board_close` 释放控制器所占用的 PC 系统资源，使得所占资源可被其它设备使用。

6.2 脉冲当量的设置

EMC-E3000/5000 系列运动控制器提供了脉冲当量设置功能，该功能可以自定义位置（位移）单位，并且提供了相应的各种高级运动函数。相关函数如表 6.2 所示。

表 6.2 脉冲当量相关函数说明

名称	功能	参考
dmc_set_equiv	设置脉冲当量值	8.2 节
dmc_get_equiv	读取设置的脉冲当量值	

注意：设置脉冲当量功能允许在轴使能状态下设置，但需 1 个总线周期时间才能生效。

例：某设备中控制器发送 100 的运动位置，设备平台前进 1mm。用户可以通过脉冲当量设置功能将运动的位移单位设置为 mm，速度单位则为 mm/s。脉冲当量为 100。相关代码如下：

.....

```
ushort MyCardNo, MyAxis;
```

```
double MyEquiv;  
MyCardNo=0; //控制器链接号  
MyAxis=0; //轴号  
MyEquiv=100; //设置脉冲当量为 100  
LTDMC.dmc_set_equiv(MyCardNo, MyAxis, MyEquiv); //设置脉冲当量  
.....
```

注意：设置当量为 100 后，若下次调用函数时没有设置脉冲当量，脉冲当量仍为 100。

例：如果用户仍希望以脉冲（pulse）为单位，那么用户可以将脉冲当量值设置为 1，即此时运动的位移单位为 pulse，速度单位则为 pulse/s。相关代码如下：

```
.....  
ushort MyCardNo, MyAxis;  
double MyEquiv;  
MyCardNo=0; //控制器链接号  
MyAxis=0; //轴号  
MyEquiv=1; //设置脉冲当量为 1  
LTDMC.dmc_set_equiv(MyCardNo, MyAxis, MyEquiv); //设置脉冲当量  
.....
```

6.3 停止命令的设置

在设备进行运动功能调试之前，必须确保安全机制的有效。

停止开关在运动过程中出现意外的运动时，能起到紧急停止运动的功能，提高设备运行时的安全性能。在使用控制器进行运动控制之前，必须保证停止命令的有效性。

EMC-E3000/5000 系列运动控制器提供了急停设置函数 `dmc_EMG_stop` 或 `dmc_stop` 来设置急停功能。

相关函数如下表 6.3 所示。

表 6.3 急停开关设置相关函数说明

名称	功能	参考
<code>dmc_EMG_stop</code>	紧急停止所有轴	8.16 节
<code>dmc_stop</code>	单轴急停或减速停	

例：紧急停止前 8 轴

```
.....  
ushort CardNo, Axis;  
CardNo=0; //控制器链接号  
for(Axis=0; Axis<8; Axis++) //循环，依次对 0~7 号轴进行设置  
{  
    LTDMC.dmc_stop(CardNo, Axis, 1); //紧急停止轴  
}
```

}

6.4 总线复位及电机使能

总线复位及电机使能相关函数如下表 6.4 所示。

表 6.4 电机使能失能相关函数说明

名称	功能	参考
nmc_set_axis_enable	设置 EtherCAT 总线驱动器使能	7.3 节
nmc_set_axis_disable	设置 EtherCAT 总线驱动器失能	
nmc_get_errcode	读取 EtherCAT 总线状态	7.2 节
nmc_clear_errcode	清除 EtherCAT 总线错误码	
dmc_soft_reset	总线热复位	8.1 节

EMC-E3000/5000 系列总线运动控制器构成的总线系统中，第一次操作控制器或更改过连接的 EtherCAT 从站设备时，需通过雷赛 Motion 扫描下载总线配置参数后，才可正常操作控制器。控制器初始化后需判断总线是否正常（nmc_get_errcode），正常时才可操作各从站。当 EtherCAT 总线报错时，可以通过执行总线热复位函数（dmc_soft_reset）来恢复总线操作。

所有电机在运动之前都需要进行使能操作，不管是总线伺服还是总线步进。EMC-E3000/5000 系列运动控制器控制轴使能，需要发送设置轴使能指令，并且在总线轴状态机（nmc_get_axis_state_machine）变成”操作使能”状态后才完成。

例：轴 0 使能操作

```
.....
ushort MyCardNo,MyAxis,errcode,statemachine;
MyCardNo=0;
MyAxis=0;
errcode=0;           //总线错误代码
statemachine=0;      //总线状态机
LTDMC.nmc_get_errcode(MyCardNo, 2,ref errcode);           //获取总线状态
if(errcode==0)       //总线正常才允许使能操作
{
    LTDMC.nmc_set_axis_enable(MyCardNo, MyAxis);//设置指定轴使能,此处 MyAxis 即当前指定轴
    LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis,ref statemachine);//获取轴状态机
    DateTime dt_start=DateTime.Now;//获取当前时间
    while(statemachine!=4) //监控轴状态机的值，该值等于 4 表示轴状态机处于使能状态
    {
        if((DateTime.Now - dt_start).TotalMilliseconds >= 1000)//1s 时间防止死循环
        {

```

```
        break;
    }
    LTDMC.nmc_set_axis_enable(MyCardNo, MyAxis); //设置轴使能
    LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis, ref statemachine); //获取状态机
}
}
else //总线不正常状态下不响应使能操作
{
    LTDMC.nmc_clear_errcode(MyCardNo, 2); //尝试清除总线错误
    return;
}
.....
```

6.5 回原点运动的实现

在进行精确的运动控制之前，需要设定运动坐标系的原点。运动平台上都设有原点传感器（也称为原点开关）。寻找原点开关的位置并将该位置设为平台的坐标原点的过程即为回原点运动。EMC-E3000/5000 系列运动控制器提供了回原点运动的相关函数，参数传输到驱动器，由驱动器完成回零功能。各驱动器回零后对回零偏移量的处理方式不同，请查看各驱动器手册说明。

相关函数如下表 6.5 所示。

表 6.5 回原点相关函数说明

名称	功能	参考
nmc_set_home_profile	设置 EtherCAT 总线轴回零参数	8.3 节
nmc_get_home_profile	读取 EtherCAT 总线轴回零参数	
nmc_home_move	启动 EtherCAT 总线轴回零	
dmc_get_home_result	读取回零状态	

回原点运动主要步骤如下：

- 1) 使用 nmc_set_home_profile 函数设置回原点模式及相关回原点运动参数；
- 2) 使用 nmc_home_move 函数执行回原点运动；
- 3) 使用 dmc_get_home_result 函数读取回原点执行的结果。

例：方式 33 回原点，直接找 EZ 方式

```
.....
ushort MyCardNo,Myaxis,Mymode,state,statemachine;
MyCardNo=0;           //控制器链接号
Myaxis=0;             //轴号
state=0;
statemachine=0;
Mymode=33;           //回零方式为 33
LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis,ref statemachine);//获取轴状态机
if(statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.nmc_set_home_profile(MyCardNo, Myaxis, Mymode, 500, 1000, 0.1, 0.1, 0);
    //设置回原点模式 ,设置 0 号轴梯形速度曲线参数
    LTDMC.nmc_home_move(MyCardNo, Myaxis);//执行回原点运动
    while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0)// 判断轴运动状态，等待回零运动完成
    {
        ApplicatIOn.DoEvents();
    }
    LTDMC.dmc_get_home_result(MyCardNo, Myaxis,ref state);//判断回零是否正常完成。
    if(state == 1) //回零正常完成
    {
        MessageBox.Show("回零完成");
    }
}
```

6.6 点位运动的实现

EMC-E3000/5000 系列运动控制器在描述运动轨迹时可以用绝对坐标也可以用相对坐标，如图 6.1 所示。两种模式各有优点，如：在绝对坐标模式中用一系列坐标点定义一条曲线，如果要修改中间某点坐标时，不会影响后续点的坐标；在相对坐标模式中，用一系列坐标点定义一条曲线，用循环命令可以重复这条曲线轨迹多次。

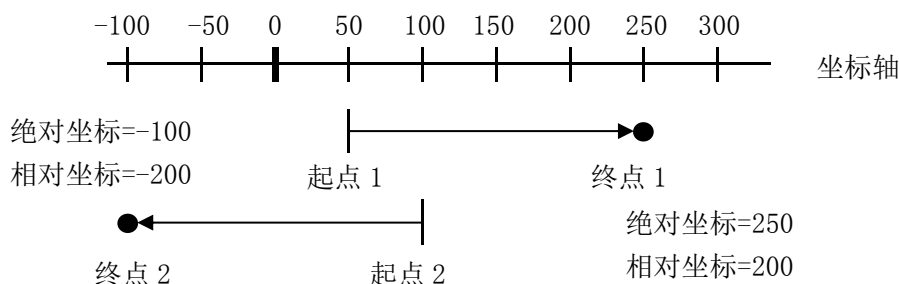


图 6.1 绝对坐标与相对坐标中轨迹终点的不同表达方式

EMC-E3000/5000 系列运动控制器在执行点位运动控制指令时，可使电机按照梯形速度曲线或 S 形速度曲线进行点位运动。梯形速度曲线参见图 2.3，S 形速度曲线参见图 2.4。

相关函数如表 6.6 所示。

表 6.6 基于脉冲当量的点位运动相关函数说明

名称	功能	参考
dmc_set_profile_unit	设置单轴运动速度曲线	8.3 节
dmc_set_s_profile	设置单轴速度曲线 S 段参数值	8.3 节
dmc_pmove_unit	定长运动	8.4 节
dmc_check_done	检测指定轴的运动状态	8.16 节

例：执行点位运动

```

.....
ushort MyCardNo, Myaxis, Myposi_mode, statemachine;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;
int MyDist;
MyCardNo = 0;                                //控制器链接号
Myaxis = 0;                                  //轴号
MyMin_Vel = 200;                             //起始速度 200unit/s
MyMax_Vel = 5000;                            //最大速度 5000unit/s
MyTacc = 0.01;                               //加速时间 0.01s
MyTdec = 0.01;                               //减速时间 0.01s
MyStop_Vel = 200;                           //停止速度 200unit/s
MyDist = 60000;                              //位移为 6000unit
Myposi_mode = 0;                             //保留参数 0
statemachine = 0;
LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis, ref statemachine); //获取轴状态机
if(statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.dmc_set_profile_unit(MyCardNo, Myaxis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
    MyStop_Vel);                                //设置单轴运动速度曲线
    LTDMC.dmc_pmove_unit(MyCardNo, Myaxis, MyDist, Myposi_mode); //执行点位运动
    while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0)        //判断轴运动状态，等待运动完
成
    {
        ApplicatIon.DoEvents();
    }
}
.....

```

在点位运行过程中，最大速度 Max_Vel 和目标位置 Dist 均可以实时改变，参见图 6.2。若在减速时改变目标位置，电机的速度变化曲线参见图 6.3。

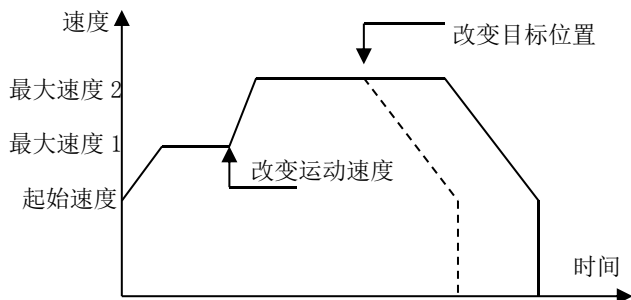


图 6.2 改变速度及改变目标位置

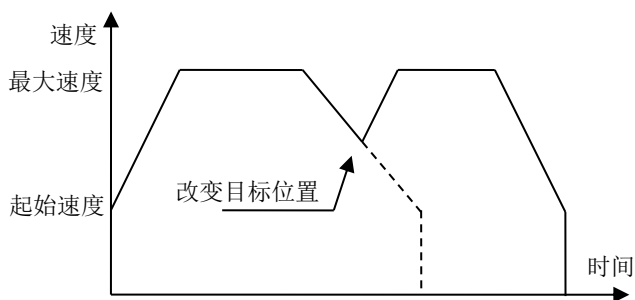


图 6.3 减速时改变目标位置

实现这 2 个功能的函数如表 6.7 所示。

表 6.7 点位运动中改变速度、目标位置的相关函数说明

名称	功能	参考
dmc_change_speed_unit	在线变速	8.4 节
dmc_reset_target_positIon_unit	在线变位	

注意：

- 1) 在线变位适用于点位运动；在线变速适用于点位及连续运动。
- 2) 在线变位后的目标位置为绝对坐标位置值，无论当前的运动模式为绝对坐标还是相对坐标模式。
- 3) 在线变速可以设置变速时间，设置的变速时间是从当前速度变速到新速度的时间。此时控制器会重新计算起始速度加速到最高速度所需的时间以及最高速度减速到停止速度所需的时间，即加减速时间会被重新计算。变速一旦成立，该轴的默认运行速度将会被改写为 New_Vel，加减速时间也会被控制器新计算的值所覆盖，也即当调用 dmc_get_profile_unit 回读速度参数时会发生与 dmc_set_profile_unit 所设置的值不一致的现象。

例：点位运动中改变速度、改变终点位置

```
.....
ushort MyCardNo, Myaxis, Myposi_mode, Mys_mode, statemachine;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;
int MyDist;
MyCardNo = 0; //控制器链接号
Myaxis = 0; //轴号
Myposi_mode=0; //相对运动模式
Mys_mode = 0; //参数保留
Mys_para = 0.02; //S 段时间为 0.02s
```

```

MyMin_Vel = 200;           //起始速度 200unit/s
MyMax_Vel = 5000;          //最大速度 5000unit/s
MyTacc = 0.05;             //加速时间 0.05s
MyTdec = 0.05;             //减速时间 0.05s
MyStop_Vel = 200;          //停止速度 200unit/s
MyDist = 60000;            //位移为 60000unit
statemachine = 0;
LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis, ref statemachine); //获取轴状态机
if (statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.dmc_set_profile_unit(MyCardNo, Myaxis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
MyStop_Vel); //设置单轴运动速度曲线
    LTDMC.dmc_set_s_profile(MyCardNo, Myaxis, Mys_mode, Mys_para); //设置 S 形速度曲线
    LTDMC.dmc_pmove_unit(MyCardNo, Myaxis, MyDist, Myposi_mode); //执行点位运动
    if (“改变速度条件”) //如果在线变速条件满足
    {
        LTDMC.dmc_change_speed_unit(MyCardNo, Myaxis, 10000, 0); //执行在线，速度变为 unit/s
    }
    if (“改变终点位置条件”) //如果在线变位条件满足
    {
        LTDMC.dmc_reset_target_positIOOn(MyCardNo, Myaxis, 100000, 0);
        //目标位置变为 100000unit
    }
}
.....

```

6.7 连续运动的实现

连续运动模式中，EMC-E3000/5000 系列运动控制器可以控制电机以梯形或 S 形速度曲线在指定的加速时间内从起始速度加速至最大速度，然后以该速度一直运行，直至调用停止指令或者该轴遇到限位信号才会按设定方式减速停止。

相关函数如表 6.8 所示。

表 6.8 连续运动相关函数说明

名称	功能	参考
dmc_vmove	指定轴连续运动	8.4 节
dmc_stop	指定轴停止运动	8.16 节

在执行连续运动过程中，可以调用 dmc_change_speed_unit 实时改变速度。注意：在以 S

形速度曲线连续运动时，改变最大速度最好在加速过程已经完成的恒速段进行。图 6.4 和图 6.5 为梯形和 S 形速度曲线下连续运动中变速和减速停止过程的速度曲线。

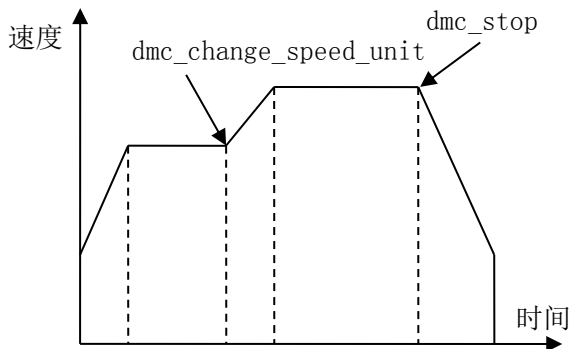


图 6.4 梯形运动中变速

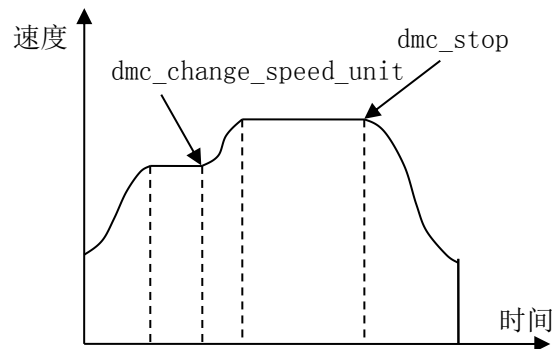


图 6.5 S 形运动中变速

例：以 S 形速度曲线加速的连续运动及变速、停止控制

```

.....
ushort MyCardNo, Myaxis, Mydir, Mystop_mode, statemachine;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;
MyCardNo = 0; //控制器链接号
Myaxis = 0; //轴号
MyMin_Vel = 200; //起始速度 200unit/s
MyMax_Vel = 5000; //最大速度 5000unit/s
MyTacc = 0.05; //加速时间 0.05s
MyTdec = 0.05; //减速时间 0.05s
MyStop_Vel = 200; //停止速度 200unit/s
Mys_para=0.01; //s 段时间
Mydir=1; //运动方向为正向
Mystop_mode = 0; //停止方式为减速停止
statemachine = 0;
LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis,ref statemachine); //获取轴状态机
if(statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.dmc_set_profile_unit(MyCardNo, Myaxis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
MyStop_Vel); //设置单轴运动速度曲线
    LTDMC.dmc_set_s_profile(MyCardNo, Myaxis, 0, Mys_para); //速度曲线为 s 形
    LTDMC.dmc_vmove(MyCardNo, Myaxis, Mydir); //执行连续运动
    if (“在线变速条件”) //如果在线变速条件满足
    {
        LTDMC.dmc_change_speed_unit(MyCardNo, Myaxis, 10000, 0);
//在线变速，速度变为 10000unit/s
    }
    if (“减速停止条件”) //减速停止条件满足
    {

```

```

        LTDMC.dmc_stop(MyCardNo, Myaxis, Mystop_mode);    //执行减速停止
    }
}

```

6.8 插补运动的实现

插补运动是为了实现轨迹控制，控制器按照一定的控制策略控制多轴联动，使运动平台用微小直线段精确地逼近轨迹的理论曲线，保证运动平台从起点到终点上的所有轨迹点都控制在允许误差范围内。

6.8.1 直线插补运动

EMC-E3000/5000 系列运动控制器可以进行任意 2~16 轴直线插补，插补计算由控制器的硬件执行，用户只需将插补运动的速度、加速度、终点位置等参数写入相关函数即可。

1. 两轴直线插补

如图 6.6 所示，2 轴直线插补从 P0 点运动至 P1 点，X、Y 轴同时启动，并同时到达终点；X、Y 轴的运动速度之比为 $\Delta X : \Delta Y$ ，二轴合成的矢量速度为：

$$\frac{\Delta P}{\Delta t} = \sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2 + \left(\frac{\Delta Y}{\Delta t}\right)^2}$$

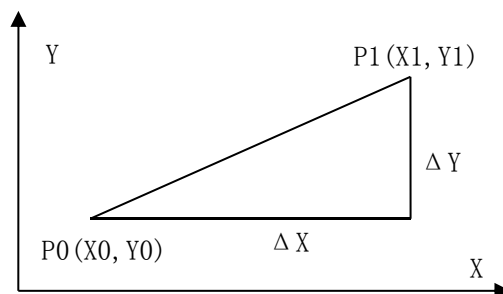


图 6.6 两轴直线插补

2. 三轴直线插补

如图 6.7 所示，在 X、Y、Z 轴内直线插补，从 P0 点运动至 P1 点。插补过程中 3 轴的速度比为 $\Delta X : \Delta Y : \Delta Z$ ，三轴合成的矢量速度为：

$$\frac{\Delta P}{\Delta t} = \sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2 + \left(\frac{\Delta Y}{\Delta t}\right)^2 + \left(\frac{\Delta Z}{\Delta t}\right)^2}$$

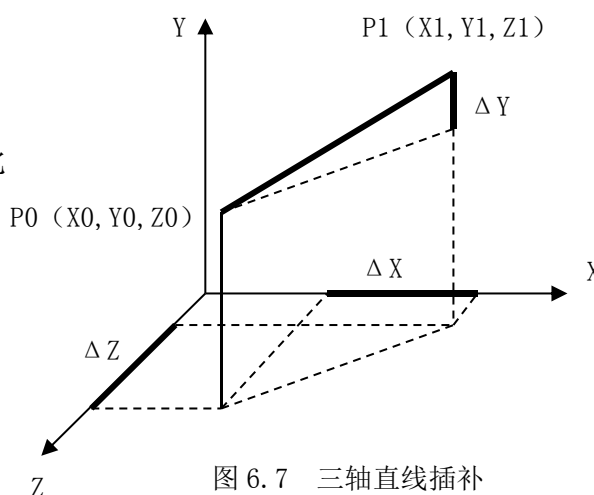


图 6.7 三轴直线插补

3. 四轴直线插补

4 轴插补可以理解为在 4 维空间里的直线插补。一般情况是 3 个轴进行直线插补，另一个旋转轴也按照一定的比例关系和这条空间直线一起运动。其合成矢量速度为：

$$\frac{\Delta P}{\Delta t} = \sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2 + \left(\frac{\Delta Y}{\Delta t}\right)^2 + \left(\frac{\Delta Z}{\Delta t}\right)^2 + \left(\frac{\Delta U}{\Delta t}\right)^2}$$

调用直线插补函数时，调用者需提供矢量速度，包括其最大矢量速度 Max_Vel 和加减速时间参数。

EMC-E3000/5000 系列总线运动控制器不仅可以进行两轴圆弧插补，并且可以进行两轴及三轴螺旋线插补、空间圆弧插补等。此外，当插补轴数大于 3 时，EMC-E3000/5000 系列总线运动控制器还支持前三轴做螺旋插补或圆弧插补的同时，后续轴跟随前三轴做线性运动。插补计算由控制器的硬件执行，用户只需将插补运动的速度曲线参数及运动参数写入相关函数即可。

各种曲线轨迹示意图如图 2.8 所示，相关函数如表 6.9 所示。

表 6.9 插补运动相关函数说明

名称	功能	参考
dmc_set_vector_profile_unit	设置插补运动速度曲线	8.5 节
dmc_get_vector_profile_unit	读取插补运动速度曲线	
dmc_set_vector_s_profile	设置插补运动速度曲线的平滑时间	
dmc_get_vector_s_profile	读取插补运动速度曲线的平滑时间	
dmc_line_unit	直线插补运动	8.6 节
dmc_arc_move_center_unit	基于圆心圆弧扩展的螺旋线插补运动（可作两轴圆弧插补）	
dmc_arc_move_radius_unit	基于半径圆弧扩展的圆柱螺旋线插补运动（可作两轴圆弧插补）	
dmc_arc_move_3points_unit	基于三点圆弧扩展的圆柱螺旋线插补运动（可作两轴及三轴圆弧插补）	

dmc_line_unit 函数可以执行多轴直线插补运动（其中单段插补模式下可支持 2~16 轴）。

例：XY 轴直线插补

```

.....
ushort MyCardNo, MyCrd, statemachine;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;
MyCardNo = 0;                                //控制器链接号
MyCrd = 0;                                    //坐标系号
MyMin_Vel = 200;                              //起始速度 200unit/s
MyMax_Vel = 5000;                             //最大速度 5000unit/s
MyTacc = 0.01;                                //加速时间 0.01s
MyTdec = 0.01;                                //减速时间 0.01s
MyStop_Vel = 200;                             //停止速度 200unit/s
statemachine = 0;

```

```
for(ushort Axis =0; Axis <2; Axis ++)  
{  
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机  
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态  
    {  
        MessageBox.Show("轴状态机错误");  
        return;  
    }  
}  
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,  
MyStop_Vel); //设置插补运动速度曲线  
LTDMC.dmc_line_unit(MyCardNo, MyCrd, 2, new ushort[] { 0, 1 }, new double [] { 20000, 20000 },  
0); //执行两轴直线插补运动  
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)  
    //判断坐标系运动状态，等待运动完成  
{  
    Application.DoEvents();  
}  
.....
```

例：六轴直线插补

```
.....  
ushort MyCardNo, MyCrd,statemachine;  
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;  
MyCardNo = 0; //控制器链接号  
MyCrd = 0; //坐标系号  
MyMin_Vel = 200; //起始速度 200unit/s  
MyMax_Vel = 5000; //最大速度 5000unit/s  
MyTacc = 0.01; //加速时间 0.01s  
MyTdec = 0.01; //减速时间 0.01s  
MyStop_Vel = 200; //停止速度 200unit/s  
statemachine = 0;  
for(ushort Axis =0; Axis <6; Axis ++)  
{  
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机  
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态  
    {  
        MessageBox.Show("轴状态机错误");  
        return;  
    }  
}  
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
```

```

MyStop_Vel);                                     //设置插补运动速度曲线
LTDMC.dmc_line_unit(MyCardNo, MyCrd, 6, new ushort[] { 0, 1, 2, 3, 4, 5 }, new double[] { 20000,
20000, 20000, 20000, 20000, 20000 }, 0);         //执行六轴直线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
                                                    //判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....

```

6.8.2 基于圆心圆弧的插补运动

dmc_arc_move_center_unit 函数可以执行两轴圆弧插补，两轴螺旋线插补（绽放与收敛型），空间螺旋线插补（绽放与收敛型），空间圆柱螺旋线插补运动。

6.8.2.1 基于圆心圆弧的两轴圆弧插补

当插补轴数为 2，且设置的圆心位置、目标位置等参数满足圆弧参数时（起始点到圆心的距离等于终点到圆心的距离）函数 dmc_arc_move_center_unit 可执行两轴圆弧插补运动。

例：XY 轴圆心圆弧插补

```

.....
ushort MyCardNo, MyCrd, statemachine;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;
MyCardNo = 0;                                     //控制器链接号
MyCrd = 0;                                         //坐标系号
MyMin_Vel = 200;                                  //起始速度 200pulse/s
MyMax_Vel = 5000;                                 //最大速度 5000pulse/s
MyTacc = 0.01;                                    //加速时间 0.01s
MyTdec = 0.01;                                    //减速时间 0.01s
MyStop_Vel = 200;                                 //停止速度 200pulse/s
statemachine = 0;
for(ushort Axis = 0; Axis < 2; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if (statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,

```

```
MyStop_Vel); //设置插补运动速度曲线
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, 2, new ushort[] { 0, 1 }, new double [] { 5000,
5000 }, new double [] { 5000, 0 }, 0, 0, 0); //执行 X、Y 轴圆弧插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....
```

6.8.2.2 基于圆心圆弧的两轴螺旋线插补

螺旋线插补基本原理：如图 6.8 所示，点 A 为圆 O 上一点，点 P 为线段 OA 上一点，当点 P 匀速由 O 向 A 运动，同时点 A 匀速沿圆 O 运动（线段 OA 等角速度沿 O 点转动），点 P 的轨迹即为绽放螺旋线；当点 P 由 A 向 O 匀速运动，同时点 A 匀速沿圆 O 运动，点 P 的轨迹即为收敛螺旋线。

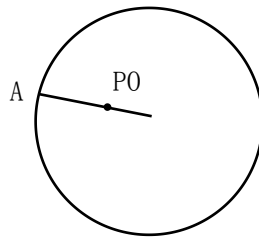


图 6.8 螺旋线插补基本原理

当插补轴数为 2，并设置满足螺旋线轨迹的圆心位置、目标位置等参数时，函数 `dmc_arc_move_center_unit` 可执行两轴螺旋线插补运动。当起始点到圆心的距离小于终点到圆心的距离，为绽放螺旋线；当起始点到圆心的距离大于终点到圆心的距离，为收敛螺旋线。

例：两轴绽放螺旋线插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
```

```

LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
if (statemachine != 4) //监控轴状态机的值, 该值等于 4 表示轴状态机处于准备好状态
{
    MessageBox.Show("轴状态机错误");
    return;
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
    //设置插补速度曲线参数, 插补运动最大矢量速度 4000unit/s, 加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 5000, 5000 }, new double[] { 2000, 0 }, MyArc_Dir, MyCircle, Myposi_mode);
    //执行螺旋线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
    //判断坐标系运动状态, 等待运动完成
{
    Application.DoEvents();
}
.....

```

运行结果如图 6.9 所示

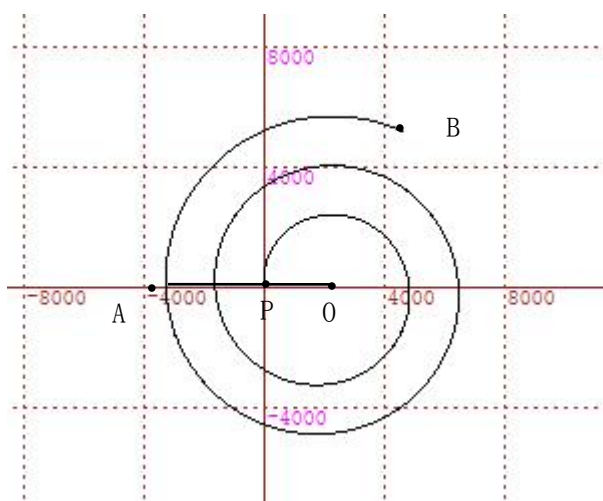


图 6.9 两轴绽放螺旋线插补

此运动中, 圆心位置为 $O(2000, 0)$, 目标位置为 $B(5000, 5000)$, 运动起点为坐标原点。

点 A、B 为以 O 为圆心, OB 为半径的圆上两点, 点 P 为 OA 上一点, 且其运动轨迹起点为坐标原点。当点 P 沿 OA 向点 A 运动时, 点 A 同时沿圆 O 顺时针运动, 运动两圈后再次到达 B 点时停止运动, 此时点 P 刚好到达 A 点 (即最后终点位置 B)。

例: 两轴收敛螺旋线插补

```

.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;

```

```

int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 6000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 6000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 3000, 3000 }, new double[] { 5000, 0 }, MyArc_Dir, MyCircle, Myposi_mode);
//执行螺旋线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....

```

运行结果如图 6.10 所示。

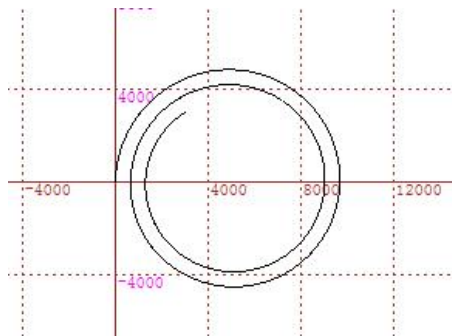


图 6.10 两轴收敛螺旋插补

此插补运动中，圆心位置为(5000，0)，目标位置为（3000，3000），运动起点为坐标原点。

6.8.2.3 基于圆心圆弧的空间螺旋线插补

函数 `dmc_arc_move_center_unit` 执行空间螺旋线插补运动是在两轴螺旋线插补运动的基础上扩展而来。在空间螺旋线插补中，前两轴（XY 轴）作平面螺旋线插补运动，第三轴（Z 轴）沿 Z 轴方向运动指定高度。空间螺旋线插补运动规则详见第 8.6 节函数说明。

当轴数大于 3，运动轨迹为螺旋插补时，列表前三轴进行螺旋插补的同时，后续轴做线性跟随运动。跟随轴与作插补运动的轴同时运动、同时停止，作线性跟随运动的速度计算由控制器的硬件执行，用户只需将插补运动的速度曲线参数及运动参数写入相关函数即可。

例：XYZ 轴基于圆心圆弧扩展的绽放螺旋线插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}

LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new
double[] { 5000, 5000, 6000 }, new double[] { 2000, 0, 0 }, MyArc_Dir, MyCircle, Myposi_mode);
//执行螺旋线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....
```

运行结果如图 6.11 所示。

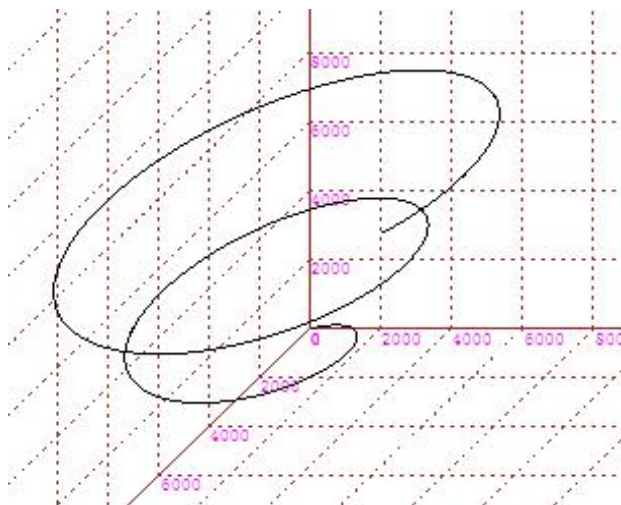


图 6.11 绽放螺旋插补

基于 XY 平面的运动轨迹如图 6.12 所示。

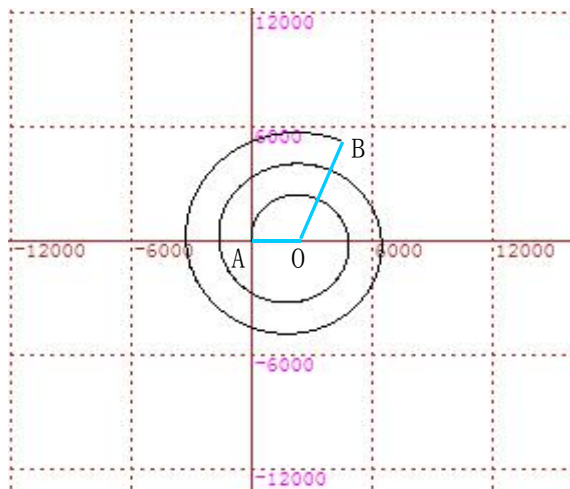


图 6.12 绽放螺旋插补基于 XY 平面的运动轨迹

此螺旋线插补运动中，轴列表前两轴所在的平面为 XY 平面。如上图 6.12 所示，在 XY 平面中，O 为插补运动的圆心，A 为运动的起点，B 为运动的终点，起点到圆心的距离 $S1=OA$ ，终点到圆心的距离 $S2=OB$ 。明显地， $S1 < S2$ ，因此，此插补运动轨迹为绽放螺旋线(具体执行不同类型的螺旋线插补运动的参数设置方法，除参考本例程外，请参考第 8.6 节相关函数说明)。

例：XYZ 轴基于圆心圆弧扩展的收敛螺旋线插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
int MyCircle;
MyCardNo = 0;                                     //控制器链接号
MyCrd = 0;                                         //坐标系号
```

```

MyaxisNum = 3; //插补运动轴数为 3
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis =0; Axis < MyaxisNum; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0,6000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 6000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new
double[] { 3000, 3000, 6000 }, new double[] { 5000, 0, 0 }, MyArc_Dir, MyCircle, Myposi_mode);
//执行螺旋线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....

```

运行结果如图 6.13 所示。

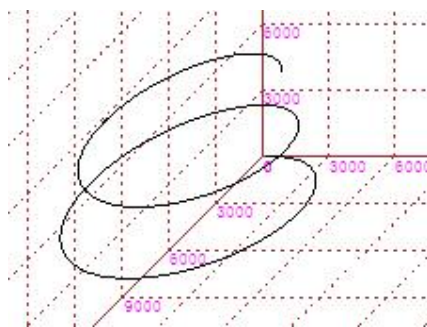


图 6.13 空间收敛螺旋线插补

基于 XY 平面的运动轨迹如图 6.14 所示。此螺旋线插补运动中，轴列表前两轴所在的平面为 XY 平面。在 XY 平面中，O 为插补运动的圆心，A 为运动的起点，B 为运动的终点，起点到圆心的距离 $S1=OA$ ，终点到圆心的距离 $S2=OB$ 。明显地， $S1>S2$ ，因此，此插补运动轨迹为收敛螺旋线。

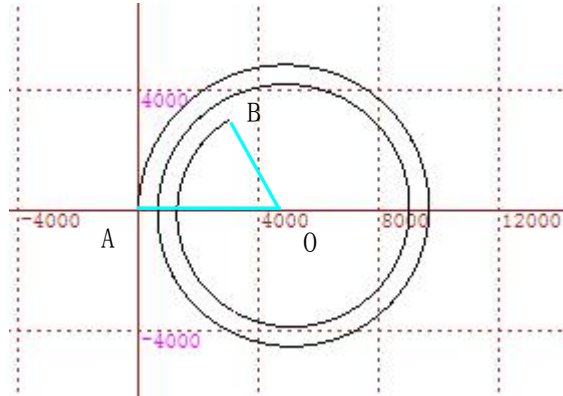


图 6.14 空间收敛螺旋线插补基于 XY 平面的运动轨迹

例：XYZ 轴基于圆心圆弧扩展的圆柱螺旋插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 6000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 6000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new
double[] { 5000, 5000, 6000 }, new double[] { 5000, 0, 0 }, MyArc_Dir, MyCircle, Myposi_mode);
//执行螺旋线插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
```

```
}
.....
```

运行结果如图 6.15 所示。

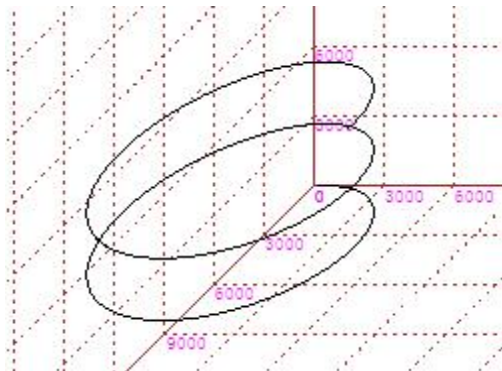


图 6.15 圆柱螺旋插补

基于 XY 平面的运动轨迹如图 6.16 所示。此螺旋线插补运动中，轴列表前两轴所在的平面为 XY 平面。在 XY 平面中，O 为插补运动的圆心，A 为运动的起点，B 为运动的终点，起点到圆心的距离 $S1=OA$ ，终点到圆心的距离 $S2=OB$ 。明显地， $S1=S2$ ，因此，此插补运动轨迹为圆柱螺旋线。

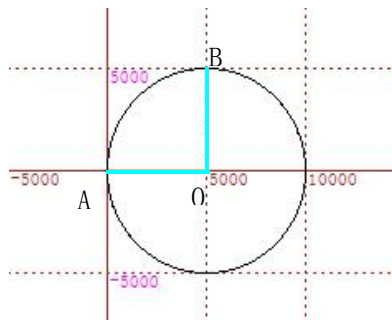


图 6.16 圆柱螺旋线插补基于 XY 平面的运动轨迹

6.8.2.4 插补模式下的辅助轴跟随

辅助轴跟随运动：当轴数大于 3，运动轨迹为螺旋线插补时，列表前三轴进行螺旋线插补的同时，后续轴做线性跟随运动，跟随轴支持的轴数最多为 3。跟随轴与作插补运动的轴同时运动、同时停止，作线性跟随运动的速度计算由控制器的硬件执行，用户只需将插补运动的速度曲线参数及运动参数写入相关函数即可。支持辅助轴跟随的函数有：基于圆心圆弧的插补运动 `dmc_arc_move_center_uni`、基于半径圆弧的插补运动 `dmc_arc_move_radius_unit`、基于三点圆弧的插补运动 `dmc_arc_move_3points_unit`。

例：六轴基于圆心圆弧扩展的空间螺旋插补（前三轴作螺旋线插补运动，后三轴线性

跟随)

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 6; //运动轴数为 6
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis =0; Axis < MyaxisNum; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2,3,4,5 },
new double[] { 5000, 5000, 6000,3000,4000,5000 }, new double[] { 2000, 0, 0,0,0,0 }, MyArc_Dir,
MyCircle, Myposi_mode);//执行插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    ApplicatIOn.DoEvents();
}
.....
```

6.8.3 基于半径圆弧的插补运动

dmc_arc_move_radius_unit 函数可以执行两轴圆弧插补，空间圆柱螺旋线插补运动。

6.8.3.1 基于半径圆弧的两轴圆弧插补

当插补轴数为 2，且设置的半径、目标位置等参数满足圆弧参数时，函数 `dmc_arc_move_radius_unit` 可执行两轴圆弧插补运动。

例：两轴圆弧插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
double MyRadius;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
MyRadius = 6000; //圆弧半径为 6000
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 0; //设置圆弧圈数为 0
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 6000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 6000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] {0, 1}, new double[] {12000, 0}, MyRadius, MyArc_Dir, MyCircle, Myposi_mode); //执行两轴圆弧插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完
{
    Application.DoEvents();
}
.....
```

运行结果如图 6.17 所示。

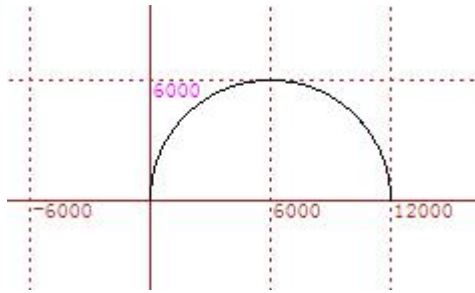


图 6.17 两轴圆弧插补

6.8.3.2 基于半径圆弧的圆柱螺旋线插补

dmc_arc_move_radius_unit 函数可执行圆柱螺旋插补，轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度。轴数大于 3 时，轴列表前三轴进行圆柱螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等（可参考第 6.8.2.4 节插补模式下的辅助轴跟随）。

例：XYZ 轴基于半径圆弧扩展的圆柱螺旋线插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode, statemachine;
double MyRadius;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
MyRadius = 2500; //设置半径大小
MyArc_Dir = 0; //设置圆弧方向为顺时针
MyCircle = 2; //设置螺旋线圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 3000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 3000unit/s，加减速时间 0.1s
```

```
LTDMC.dmc_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2}, new
double[] { 5000, 0, 4000}, MyRadius, MyArc_Dir, MyCircle, Myposi_mode);
```

//执行基于半径圆弧扩展的圆柱螺旋线插补

```
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
```

//判断坐标系运动状态，等待运动完成

```
{
    Application.DoEvents();
}
.....
```

运行结果如图 6.18 所示。

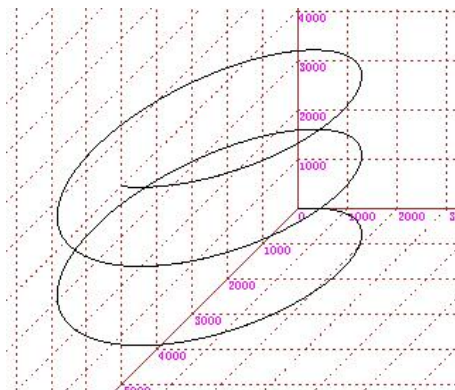


图 6.18 圆柱螺旋线插补

6.8.4 基于三点圆弧的插补运动

dmc_arc_move_3points_unit 函数可以执行两轴圆弧插补，空间圆弧插补，空间圆柱螺旋线插补运动。执行基于三点圆弧的插补运动时，需正确设置中间位置、目标位置等参数，当圆弧圈数设置不同数值时，可实现不同类型的插补运动。

当轴数大于 3 时，轴列表前三轴进行插补运动的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等（可参考第 6.8.2.4 节插补模式下的辅助轴跟随）

当正确设置运动轴数、中间位置、目标位置等参数后，设置圆弧插补圈数为负数时，函数 dmc_arc_move_3points_unit 可实现空间圆弧插补（参数设置规则详见 8.6 节插补运动）。

例：XYZ 轴基于三点圆弧扩展的空间圆弧插补

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int MyCircle;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
MyCircle = -1; //设置空间圆弧，圆弧圈数为 0
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
```

```

statemachine = 0;
for(ushort Axis =0; Axis < MyaxisNum; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 3000, 0.1, 0, 0);
    //设置插补速度曲线参数，插补运动最大矢量速度 3000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_3points_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new
double[] {5000,5000, 0 }, new double[] {2500,2500,2500 }, MyCircle, Myposi_mode);
    //执行顺时针方向三点圆弧插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
    //判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....

```

运行结果如图 6.19 所示。

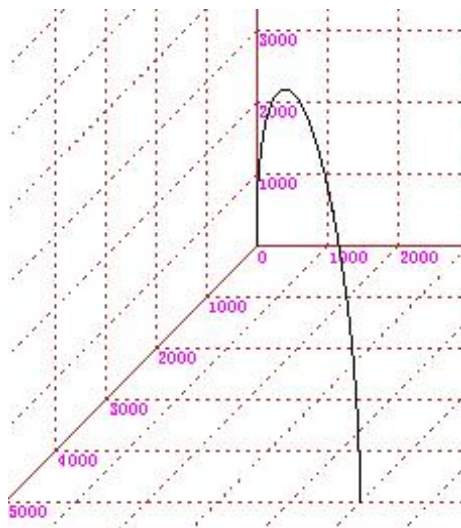


图 6.19 空间圆弧插补

例：XYZ 轴基于三点圆弧扩展的圆柱螺旋插补

```

.....
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int MyCircle;
MyCardNo = 0;
    //控制器链接号

```

```

MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
MyCircle = 2; //设置圆柱螺旋插补，圆弧圈数为 2
Myposi_mode = 0; //设置圆弧插补模式为相对坐标模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 6000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 3000unit/s，加减速时间 0.1s
LTDMC.dmc_arc_move_3points_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new
double[] { 5000, 5000, 5000 }, new double[] { 2500, -2500, 2500 }, MyCircle, Myposi_mode);
//执行顺时针方向三点圆弧插补运动
while (LTDMC.dmc_check_done_multicoor(MyCardNo, MyCrd) == 0)
//判断坐标系运动状态，等待运动完成
{
    Application.DoEvents();
}
.....

```

运行结果如图 6.20 所示。

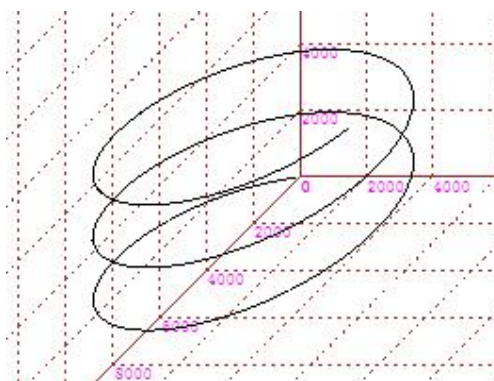


图 6.20 空间圆弧插补

6.8.5 矩形插补运动

dmc_rectangle_move_unit 函数可以执行两轴矩形插补运动，包括逐行模式与渐开线模式。

矩形插补运动支持前瞻模式和非前瞻模式(dmc_conti_set_lookahead_mode)，前瞻模式矩形拐角会平滑处理变成圆弧，无法到达矩形端点；非前瞻模式则精确到达矩形端点。一般建议运行矩形插补运动前，调用前瞻函数关闭前瞻模式。

6.8.5.1 逐行模式的矩形插补运动

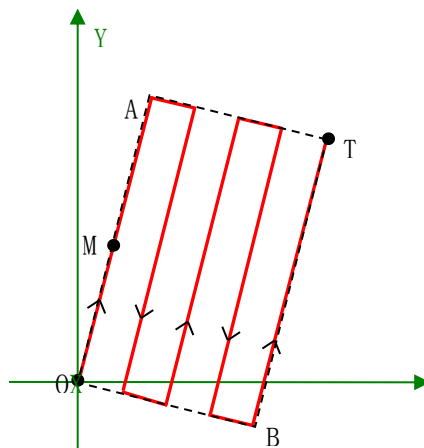


图 6.21 逐行模式的矩形插补运动示意图

如图 6.21 所示，要进行一次逐行模式的矩形插补运动，必须确定以下三个参数：

- (1) 对角位置，如图 6.21 中的点 T，该点确定了矩形的对角点。
- (2) 矩形方向标记位置，如图 6.21 中的点 M。矩形方向标记位置确定了矩形插补运动第一条边的运动方向，该方向为射线 OM 的方向。此时，通过对角位置坐标及矩形边的方向，则可以确定该矩形区域的框架，如图 6.21 中的矩形 OATB。
- (3) 行数，如图 6.21 中矩形插补运动的行数为 5。通过行数参数，可以设置在指定矩形区域内进行矩形逐行插补运动的行数。

例：逐行模式的矩形插补运动

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyCount, Myrect_mode, Myposi_mode, statemachine;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
MyCount = 6; //行数为 6
Myrect_mode = 0; //设置矩形插补模式为逐行模式
Myposi_mode = 0; //设置运动模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
```

```

{
    MessageBox.Show("轴状态机错误");
    return;
}
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
    //设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_rectangle_move_unit(0, 0, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 3000, 2000 },
new double[] { 700, 0 }, MyCount, Myrect_mode, Myposi_mode);
    //执行逐行矩形插补运动
.....

```

运行结果如图 6.22 所示。

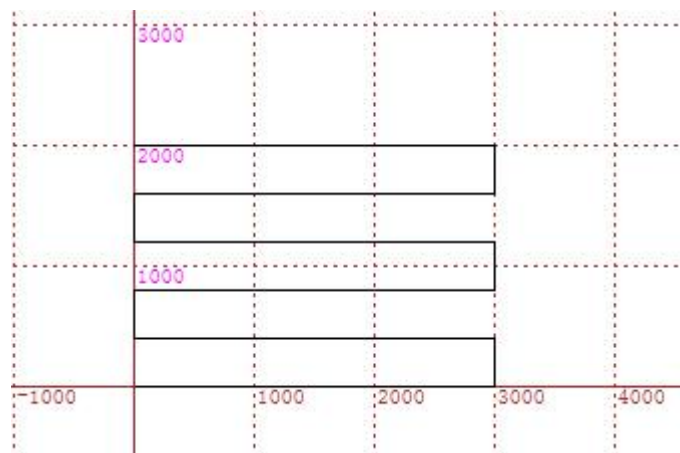


图 6.22 逐行模式的矩形插补运动

6.8.5.2 渐开线模式的矩形插补运动

如图 6.23 所示，要进行一次渐开线模式的矩形插补运动，必须确定以下三个参数：

- (1) 对角位置，如图 6.23 中的点 T，该点确定了矩形的对角点。
- (2) 矩形方向标记位置，如图 6.23 中的点 M。矩形方向标记位置确定了矩形插补运动第一条边的运动方向，该方向为射线 OM 的方向。此时，通过对角位置坐标及 矩形边的方向，则可以确定该矩形区域的框架，如图 6.23 中的矩形 OATB。
- (3) 圈数，如图 6.23 中矩形插补运动的圈数为 3。通过圈数参数，可以设置在指定矩形区域内进行矩形区域渐开线插补运动的圈数；

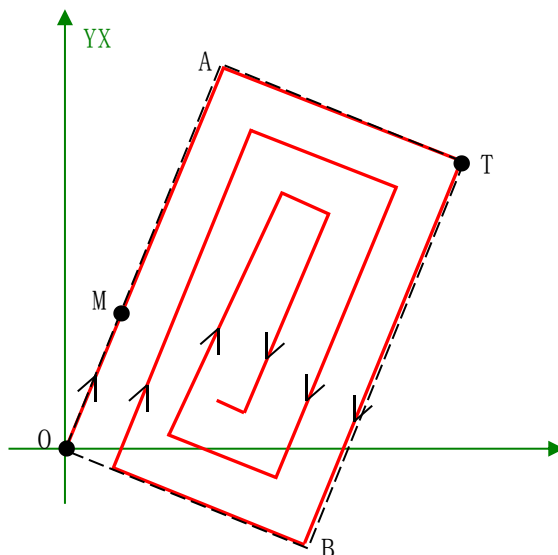


图 6.23 渐开线模式的矩形插补运动示意图

例：渐开线模式的矩形插补运动

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyCount, Myrect_mode, Myposi_mode, statemachine;
MyCardNo = 0;                                     //控制器链接号
MyCrd = 0;                                         //坐标系号
MyaxisNum = 2;                                    //插补运动轴数为 2
MyCount = 4;                                      //行数为 4
Myrect_mode = 1;                                  //设置矩形插补模式为渐开线模式
Myposi_mode = 0;                                  //设置运动模式
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_rectangle_move_unit(0, 0, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 2000, 1500 }, new
double[] { 200, 500 }, MyCount, Myrect_mode, Myposi_mode);
//执行渐开线模式的矩形插补运动
.....
```

运行结果如图 6.24 所示。

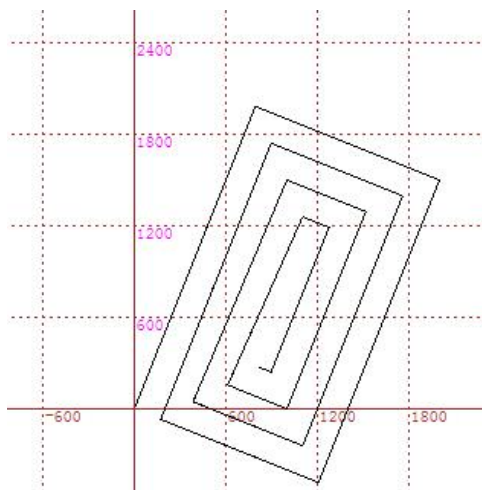


图 6.24 渐开线模式的矩形插补运动

6.9 连续插补运动的实现

EMC-E5000 系列支持连续插补运动，EMC-E3000 不支持连续插补运动。

在运动控制中，采用连续插补可实现速度的平滑过渡，减小机器的振动，可能提高机器的加工精度和加工速度。

EMC-E5000 控制器提供了连续插补运动功能。连续插补指令支持直线插补、圆弧插补、螺旋线插补、矩形插补、IO 控制等。各种曲线轨迹示意图参见图 2.8。

此外，EMC-E5000 控制器支持 4 个坐标系，每个坐标系的连续缓冲区最多可缓存 5000 条指令。4 个坐标系的速度可独立设置，执行连续插补时 4 个坐标系可独立进行连续插补运动，即可同时进行 4 组连续插补运动。

实现基本连续插补运动的一般步骤如下：

- 1) 如需要小线段前瞻功能，使用函数 `dmc_conti_set_lookahead_mode` 设置连续插补前瞻模式及参数；
- 2) 使用 `dmc_set_vector_profile_unit`、`dmc_set_vector_s_profile` 函数设置连续插补速度曲线；
- 3) 使用函数 `dmc_conti_open_list` 打开连续插补缓冲区；
- 4) 添加连续插补运动指令；
- 5) 使用函数 `dmc_conti_start_list` 启动连续插补运动；步骤 4 和 5 可以对调顺序，即可以添加运动指令完成后执行连续插补运动或启动运动后添加运动指令；若设置前瞻模式，需在步骤 4 预压一些运动指令，启动运动后再在步骤 5 后添加运动指令。
- 6) 使用函数 `dmc_conti_close_list` 关闭连续插补缓冲区。

需要注意的是 `dmc_conti_set_lookahead_mode` 设置连续插补前瞻模式及参数指令必须在 `dmc_conti_open_list` 打开连续插补缓冲区指令前调用。下面列出了连续插补中需要调用的函数列表：

a. 连续插补初始化及状态检测函数如下表所示。

表 6.10 连续插补运动相关函数说明

名称	功能	参考
dmc_conti_set_lookahead_mode	设置连续插补前瞻模式及参数	8.7 节
dmc_conti_get_lookahead_mode	回读连续插补前瞻模式及参数	
dmc_conti_open_list	打开连续插补缓冲区	
dmc_conti_start_list	开始连续插补	
dmc_conti_close_list	关闭连续插补缓冲区	
dmc_conti_pause_list	暂停连续插补	
dmc_conti_stop_list	停止连续插补	
dmc_conti_remain_space	查询插补缓冲区剩余插补空间	
dmc_conti_read_current_mark	读连续插补缓冲区当前插补段号	8.16 节
dmc_conti_get_run_state	读取连续插补运动状态	

b. 连续插补运动相关函数如下表所示。

名称	功能	参考
dmc_conti_line_unit	连续插补中直线插补指令	8.7 节
dmc_conti_arc_move_center_unit	连续插补中基于圆心圆弧扩展的螺旋线插	
dmc_conti_arc_move_radius_unit	连续插补中基于半径圆弧扩展的圆柱螺旋	
dmc_conti_arc_move_3points_unit	连续插补中基于三点圆弧扩展的圆柱螺旋	
dmc_conti_line_unit	连续插补中直线插补指令	

c. 连续插补 IO 操作相关函数如下表所示。

名称	功能	参考
dmc_conti_set_pause_output	设置连续插补暂停及异常停止时 IO 输出	8.10 节
dmc_conti_get_pause_output	读取连续插补暂停及异常停止时 IO 输出	
dmc_conti_wait_input	连续插补等待 IO 输入	
dmc_conti_delay_outbit_to_start	连续插补中相对于轨迹段起点 IO 滞后输	
dmc_conti_delay_outbit_to_stop	连续插补中相对于轨迹段终点 IO 滞后输	
dmc_conti_ahead_outbit_to_stop	连续插补中相对于轨迹段终点 IO 提前输	
dmc_conti_write_outbit	连续插补中缓冲区立即 IO 输出	
dmc_conti_clear_IO_actIOn	清除段内未执行完的 IO 动作	

d. 连续插补其它函数如下表所示。

名称	功能	参考
dmc_conti_delay	连续插补中暂停延时指令	8.7 节
dmc_set_arc_limit	设置圆弧限速功能	8.12 节
dmc_get_arc_limit	回读圆弧限速功能参数	

6.9.1 小线段前瞻及圆弧限速功能

EMC-E5000 控制器对连续插补运动提供了前瞻和非前瞻模式。连续插补指令支持直线插补、圆弧插补、螺旋线插补、IO 控制等，前瞻和非前瞻主要区别在于前瞻可很好应用于小线段轨迹，其轨迹连接处更平滑。

EMC-E5000 控制器对于圆弧提供了圆弧限速功能，主要为限制运行速度，使加速度值不超出设定范围。

相关函数如表 6.11 所示。

表 6.11 连续插补运动相关函数说明

名称	功能	参考
dmc_conti_set_lookahead_mode	设置连续插补前瞻模式及参数	8.9/8.12 节
dmc_conti_get_lookahead_mode	回读连续插补前瞻模式及参数	
dmc_set_arc_limit	设置圆弧限速参数	
dmc_get_arc_limit	读取圆弧限速参数	

例：连续插补运动，直线+圆弧（前瞻运动）

```

.....
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode, statemachine, mode, ArcLimit;
int Mymark, LookaheadSegment;
double PathError, LookaheadAcc;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补轴数为 3
Myposi_mode = 0; //相对坐标模式
Mymark = 0; //自动编号
mode = 1; //定义连续插补使能参数，0 连续，1 前瞻运动
LookaheadSegment = 200; //定义插补段数:200 段
PathError = 1; //定义轨迹误差: 1unit
LookaheadAcc = 10000; //定义拐弯加速度:10000unit/s2
ArcLimit = 1; //使能圆弧限速，0: 不使用，1 使能
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}

```

```

}
//第一步、设置圆弧限速功能使能
LTDMC.dmc_set_arc_limit(MyCardNo,MyCrd,ArcLimit,0,0);
//第二步、设置前瞻参数
LTDMC.dmc_conti_set_lookahead_mode(MyCardNo,MyCrd,mode,LookaheadSegment,PathError,Lookahead
Acc);
//第三步、打开连续插补缓冲区
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
//第四步、设置插补运动速度参数,最大矢量速度 2000unit/s, 加减速时间 0.1s
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
//第五步、添加直线插补段, 直线插补, 相对模式
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new double[]
{ 2000, 3000, 1000 }, Myposi_mode, Mymark);
//第六步、添加圆弧插补段, XY 平面圆弧插补, 逆时针, 相对坐标模式
MyaxisNum = 2; //重新定义插补轴数为 2
LTDMC.dmc_conti_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 1500, 0 }, new double[] { 750, 0 }, 1, 0, 0, 0);
//第七步、开始连续插补
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);
//第八步、关闭连续插补缓冲区
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);
.....

```

6.9.2 连续插补运动的暂停延时功能

EMC-E5000 控制器对连续插补运动提供了暂停延时功能，相关函数如表 6.12 所示。

表 6.12 连续插补运动的暂停延时功能相关函数说明

名称	功能	参考
dmc_conti_delay	连续插补中暂停延时指令	8.7 节

注意：1) 延时时间为运动停止时的等待时间。

2) 当延时时间设置为 0 时，延时时间将无限长。

例：连续插补运动的延时功能

```

.....
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode, statemachine;
int Mymark;
double MyDelayTime;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补轴数为 3

```

```

Myposi_mode = 0; //相对坐标模式
Mymark = 0; //自动编号
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
//打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s
LTDMC.dmc_set_s_profile(MyCardNo, MyCrd, 0, 0.02); //设置连续插补 S 段时间为 0.02s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000, 1000 }, Myposi_mode, Mymark); //直线插补，相对模式
MyDelayTime = 2; //延时时间为 2 秒
LTDMC.dmc_conti_delay(MyCardNo, MyCrd, MyDelayTime, Mymark); //暂停延时 2 秒
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 1000, 3000 }, Myposi_mode, Mymark); //直线插补，相对模式

LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....

```

运行结果：

当完成第一段直线插补运动后，延时 2 秒，然后再继续执行后续运动。其速度曲线如图 6.25 所示。

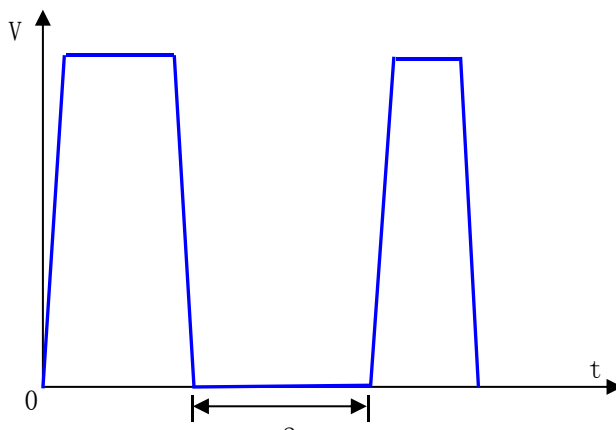


图 6.25 连续插补中暂停延时功能例程的速度曲线

6.9.3 连续插补 IO 控制

EMC-E5000 控制器在连续插补运动过程中支持缓存 IO 控制，通过 IO 控制函数的调用，用户可以轻易实现各种 IO 控制功能。

6.9.3.1 连续插补暂停及异常停止时的 IO 输出控制

当暂停、停止连续插补，或遇到其他异常停止（如碰到 EMG 信号）时，控制器可以按照用户预先设置的 IO 输出状态进行 IO 控制。

相关函数如表 6.15 所示。

表 6.15 连续插补运动的暂停延时功能相关函数说明

名称	功能	参考
dmc_conti_set_pause_output	设置连续插补暂停及异常停止时 IO 输出状态	8.10 节

例：连续插补暂停时，通用输出口 0 及输出口 2 输出高电平，恢复运动时不恢复暂停前的 IO 状态

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, MyActIO, Myposi_mode, statemachine;
int Mymask, Mystate, mark;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
MyActIO = 2; //激活模式为 1：暂停时输出设定的 IO 状态，恢复运行时不恢复暂停前的 IO 状态
Myposi_mode = 0; //相对运动模式
mark = 0; //自动编号
Mymask = Convert.ToInt32("101", 2); //选择通用输出口 0 和 2 输出，第 0 位及第 2 位值为 1
Mystate = Convert.ToInt32("101", 2); //输出口 0 和 2 输出高电平，第 0 位及第 2 位值为 1
```

```

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区
LTDMC.dmc_conti_set_pause_output(MyCardNo, MyCrd, MyActIOOn, Mymask, Mystate);
//设置连续插补暂停时 IO 输出状态
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000, 4000 }, Myposi_mode, mark); //执行连续插补运动
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 2000, 4000 }, Myposi_mode, mark); //执行连续插补运动
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....

```

运行结果：

在执行连续插补运动的过程中，当调用函数 `dmc_conti_pause_list` 暂停连续插补运动时，通用输出口 0 及输出口 2 输出高电平；再次调用函数 `dmc_conti_start_list` 恢复连续插补运动时，通用输出口 0 及输出口 2 仍然保持为高电平状态。

6.9.3.2 连续插补中的等待 IO 输入功能

当进行连续插补运动时，用户可以在缓冲区插入等待 IO 输入指令。当控制器执行到此指令时，其只有在接收到输入 IO 信号或超出超时时间后，才会执行后续运动，超时时间可以自由设置。相关函数如表 6.16 所示。

表 6.16 连续插补等待 IO 输入相关函数说明

名称	功能	参考
<code>dmc_conti_wait_input</code>	连续插补等待 IO 输入	8.10 节

注意： 1) 当超时时间设为 0 时，控制器将一直等待 IO 输入信号，超时时间为无限长。
2) 超时时间为运动停止时的等待时间。

例：连续插补中，先运行一段直线插补，然后等待通用输入口 0 为低电平时（或等待时间超过 5 秒后），才执行后续运动

```

.....
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel, MyTimeOut, statemachine;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态

```

```

    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}

LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
    //设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
    //打开连续插补缓冲区
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort [] {0,1}, new double
[] {4000,5000},0,0); //直线插补运动
Mybitno=0;           //通用输入口 0
MyLevel=0;           //等待输入低电平
MyTimeOut=5;         //等待 5s
LTDMC.dmc_conti_wait_input(MyCardNo, MyCrd, Mybitno, MyLevel, MyTimeOut, 0); //等待 IO 输入
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000,3000 }, 0, 0); //直线插补运动
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....

```

6.9.3.3 连续插补中的普通 IO 输出控制

当进行连续插补运动时，用户可以在缓冲区插入普通 IO 输出控制指令。

在每段运动轨迹中，至多可添加 10 个 IO 操作，包括普通 IO 控制及精确位置 CMP 输出控制。普通 IO 输出的分辨率为 1ms（误差在 0.2ms 以内）。

相关函数如表 6.17 所示。

表 6.17 连续插补中的普通 IO 输出控制相关函数说明

名称	功能	参考
dmc_conti_delay_outbit_to_start	连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）	8.10 节
dmc_conti_delay_outbit_to_stop	连续插补中相对于轨迹段终点 IO 滞后输出	
dmc_conti_ahead_outbit_to_stop	连续插补中相对于轨迹段终点 IO 提前输出（段内执行）	
dmc_conti_write_outbit	连续插补中缓冲区立即 IO 输出	
dmc_conti_clear_IO_actIOn	清除段内未执行完的 IO 动作	

其中，dmc_conti_delay_outbit_to_start 函数可以设置轨迹段内 IO 的输出位置或时间，该位置或时间是相对于该轨迹段起点的滞后值。

dmc_conti_delay_outbit_to_stop 函数可以设置轨迹段执行完后 IO 的输出时间，该时间是相对于该轨迹段终点的滞后值。

dmc_conti_ahead_outbit_to_stop 函数可以设置轨迹段内 IO 的输出位置或时间，该位置或时间是相对于该轨迹段终点的提前值。

dmc_conti_write_outbit 函数可以实现连续插补运动中的 IO 立即输出。

dmc_conti_clear_IO_actIOOn 函数可以实现当本段轨迹运行完成时，清除仍未执行完的 IO 操作，使其不会在后续轨迹段中被继续执行。该函数对 dmc_conti_delay_outbit_to_start、dmc_conti_ahead_outbit_to_stop、dmc_conti_delay_outbit_to_stop 指令起作用。

例：连续插补中的 IO 输出控制

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel, MyTimeOut, MyDelayMode, statemachine;
double MyDelayVal, MyRevTime;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区
Mybitno = 2; //通用输出口 2
MyLevel = 0; //输出电平为低电平
MyDelayVal = 1; //滞后时间为 1s
MyDelayMode = 0; //滞后模式为滞后时间
MyRevTime = 0.5; //电平延时翻转时间为 0.5s
LTDMC.dmc_conti_delay_outbit_to_start(MyCardNo, MyCrd, Mybitno, MyLevel, MyDelayVal,
MyDelayMode, MyRevTime);
//相对于轨迹段起点，滞后 1 秒，输出口 2 输出低电平，低电平持续时间为 0.5s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 4000,
4000 }, 0, 0); //第一段轨迹直线插补
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 2000, 4000 }, 0, 0); //第二段轨迹直线插补
```

```
Mybitno = 0; //通用输出口 0
MyLevel = 0; //输出电平为低电平
MyRevTime = 0.5; //电平延时翻转时间为 0.5s
LTDMC.dmc_conti_write_outbit(MyCardNo, MyCrd, Mybitno, MyLevel, MyRevTime); //IO 立即输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 5000, 3000 }, 0, 0); //第三段轨迹直线插补
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....
```

运行结果：

当第一段直线插补开始 1s 后，通用输出口 2 输出低电平，低电平持续时间为 0.5s。当第二段直线插补结束时，通用输出口 0 立即输出低电平，低电平持续时间为 0.5s。

例：清除段内未执行完的 IO 动作功能

```
.....
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel, MyTimeOut, MyDelayMode;
double MyDelayVal, MyRevTime;
MyCardNo = 0; //控制器链接号
MyCrd = 0; //坐标系号
MyaxisNum = 2; //插补运动轴数为 2
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 3000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 3000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区
Mybitno = 2; //通用输出口 2
MyLevel = 0; //输出电平为低电平
MyDelayVal = 1; //滞后时间为 1s
MyDelayMode = 0; //滞后模式为滞后时间
MyRevTime = 3; //电平延时翻转时间为 3s
LTDMC.dmc_conti_delay_outbit_to_start(MyCardNo, MyCrd, Mybitno, MyLevel, MyDelayVal,
```

```
MyDelayMode, MyRevTime);
```

```
//相对于轨迹段起点,滞后 1 秒, 输出口 2 输出低电平, 低电平持续时间为 3s
```

```
LTDMC.dmc_conti_clear_IO_actIOOn(MyCardNo, MyCrd, 0x4);
```

```
//清除段内未执行完的通用输出口 2 的动作
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]  
{ 6000, 0 }, 0, 0); //第一段轨迹直线插补
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]  
{ 4000, 4000 }, 0, 0);//第二段轨迹直线插补
```

```
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
```

```
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
```

```
.....
```

运行结果:

当第一段直线插补开始 1s 后, 通用输出口 2 输出低电平, 低电平持续时间为 3s。但继续运行 3s 后, 已进入了第二段直线插补运动。此时, 由于使用了函数 `dmc_conti_clear_IO_actIOOn` 清除段内未执行完的 IO 操作, 所以通用输出口 2 的电平将持续保持低电平, 不会翻转。如图 6.29 所示。

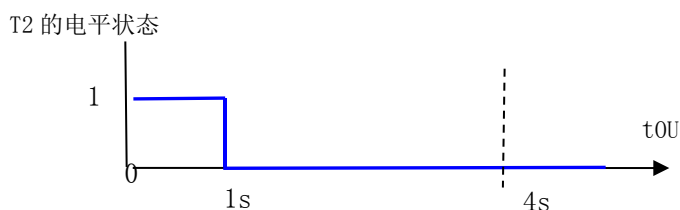


图 6.29 通用输出口 2 的电平时序图(清除段内未执行完的 IO 操作)

如果在上例中没有在第一段直线插补运动中插入指令 `dmc_conti_clear_IO_actIOOn` 清除段内未执行完的 IO 操作, 那么通用输出口 2 的电平将会在到达设置的延时翻转时间后翻转, 尽管此时已进入了第二段直线插补运动。如图 6.30 所示。

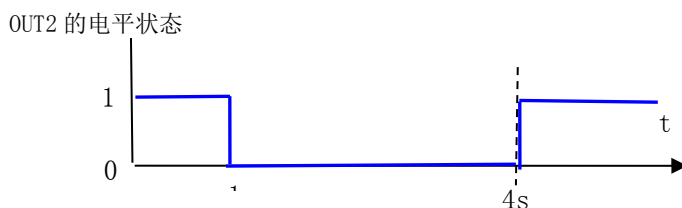


图 6.30 通用输出口 2 的电平时序图(未执行清除段内未执行完的 IO 操作)

6.8.4 连续插补中位置跟随功能

EMC-E5000 控制器提供连续插补位置跟随功能。该功能使插补系以外的轴能跟随插补运动的合位移运动, 从而实现刀具在加工过程中处于合适的方向和位置。相关函数如表 6.21 所示。

表 6.21 连续插补中位置跟随功能相关函数说明

名称	功能	参考
dmc_conti_gear_unit	连续插补中位置跟随	8.13 节

连续插补中控制位置跟随输出一般步骤如下：

- 1) 使用函数 dmc_conti_open_list 打开连续插补缓冲区；
- 2) 使用函数 dmc_conti_gear_unit 设置位置跟随参数；
- 3) 添加连续插补运动指令；
- 4) 使用函数 dmc_conti_start_list 启动连续插补运动；
- 5) 使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

例：连续插补中位置跟随功能。

设定跟随旋转轴转一圈的脉冲数为 10000.

```
ushort MyCardNo, MyCrd, MyaxisNum, statemachine;
```

```
MyCardNo = 0; //控制器链接号
```

```
MyCrd = 0; //参与插补运动的坐标系 0
```

```
MyaxisNum = 2; //插补运动轴数为 2
```

```
statemachine = 0;
```

```
for(ushort Axis=0; Axis < MyaxisNum; Axis++)
```

```
{
```

```
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
```

```
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
```

```
    {
```

```
        MessageBox.Show("轴状态机错误");
```

```
        return;
```

```
    }
```

```
}
```

```
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
```

```
    //打开连续插补缓冲区
```

```
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 5000, 0.1, 0.1, 0);
```

```
//设置插补速度曲线参数，插补运动最大矢量速度 5000unit/s，加减速时间 0.1s
```

```
LTDMC.dmc_set_profile_unit(_CardID, 2, 0, 5000, 0.1, 0.1, 0);
```

```
//*****第一段*****//
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 10000, 10000 }, 0, 0); //第一段轨迹，直线插补 1，相对模式
```

```
//*****第二段*****//
```

```
LTDMC.dmc_conti_pmove_unit(MyCardNo, MyCrd, 2, -1250, 0, 0, 0); //设置刀向偏转，逆时针 45 度
```

```
LTDMC.dmc_conti_gear_unit(MyCardNo, MyCrd, 2, 5000, 0, 0); //设置位置跟随运动，顺时针 180 度
```

```
LTDMC.dmc_conti_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
```

```
double[] { 10000, 0 }, 5000, 0, 0, 0, 0);//第二段轨迹，圆弧插补，相对模式
//*****第三段*****//
LTDMC.dmc_conti_pmove_unit(MyCardNo, MyCrd, 2, -2500, 0, 0, 0);//设置刀向偏转，逆时针 90 度
LTDMC.dmc_conti_gear_unit(MyCardNo, MyCrd, 2, 5000, 0, 0);//设置位置跟随运动，顺时针 180 度
LTDMC.dmc_conti_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 0, -10000 }, 5000, 0, 0, 0, 0);//第三段轨迹，圆弧插补，相对模式
//*****第四段*****//
LTDMC.dmc_conti_pmove_unit(MyCardNo, MyCrd, 2, 1250, 0, 0, 0);//设置刀向偏转，顺时针 45 度
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { -10000,
10000 }, 0, 0);//第四段轨迹，直线插补 2，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
```

运行结果：

插补运动轨迹曲线，如图 6.34

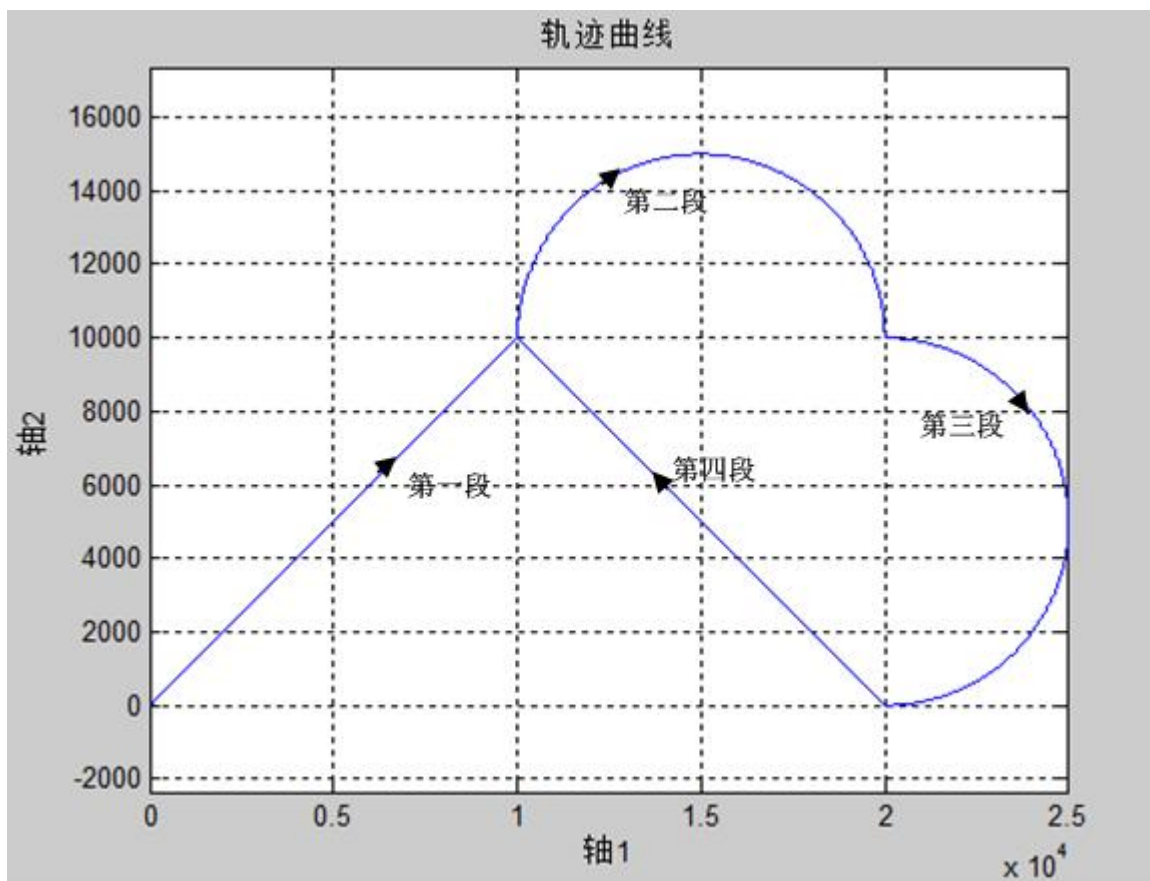


图 6.34 刀具轨迹曲线

6.10 PWM 输出功能的实现

EMC-E3000/5000 系列运动控制器提供了 6 路 PWM 输出, PWM0~PWM5 对应控制器本地 16 路输出的 OUT0~OUT5。如图 6.35 所示, EMC-E5064 控制器输出的 PWM 波形的周期为 t_2 (频率即为 $1/t_2$), 占空比为 t_1/t_2 , 幅值为 $V_1 = 24V$ 。

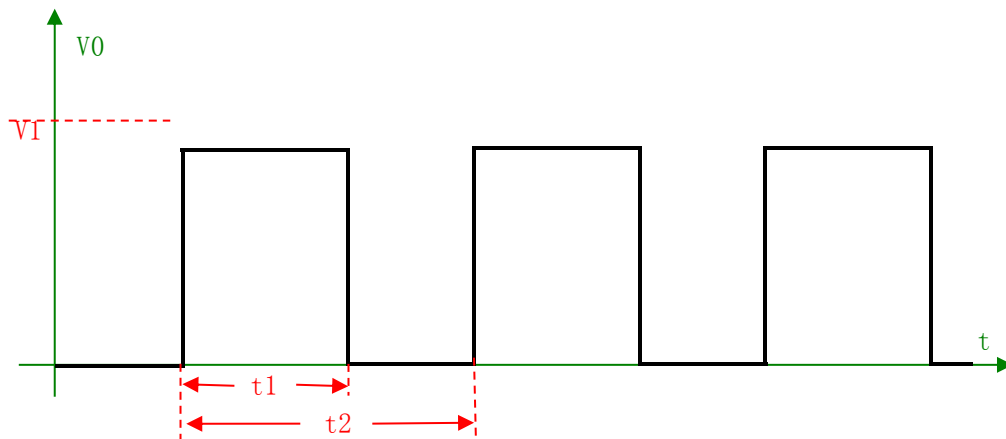


图 6.35 PWM 输出示意图

6.10.1 PWM 立即输出功能

EMC-E3000/5000 系列运动控制器支持 PWM 立即输出功能, 用户只需要简单设置 PWM 输出通道、频率及占空比参数即可实现 PWM 输出功能, 非常方便。

相关函数如表 6.22 所示。

表 6.22 PWM 立即输出功能相关函数说明

名称	功能	参考
dmc_set_pwm_enable_extern	设置 PWM 使能状态	8.12 节
dmc_set_pwm_output	设置 PWM 输出	

例：PWM 输出功能

```

.....
ushort MyCardNo, Myenable, MyPwmNo;
double MyfDuty, MyfFre;
MyCardNo = 0;           //控制器链接号
Myenable = 1;           //PWM 输出使能状态: 使能
MyPwmNo = 0;            //PWM 输出通道为 0 通道, 即本地 OUT2
MyfDuty = 0.5;          //PWM 输出占空比为 50%
MyfFre = 10000;         //PWM 输出频率为 10000Hz
LTDMC.dmc_set_pwm_enable_extern(MyCardNo, MyPwmNo, Myenable);
//使能 PWM 输出
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);

```

```
//设置 PWM 输出，占空比为 50%，频率为 10000Hz
```

```
.....
```

```
.....
```

```
Myenable = 0;
```

```
//PWM 输出使能状态：禁止
```

```
LTDMC.dmc_set_pwm_enable(MyCardNo, MyPwmNo, Myenable);//禁止 PWM 输出
```

```
.....
```

运行结果：

EMC-E5000 控制器的本地 OUT2 口输出 PWM 波形，其占空比为 50%，频率为 10000Hz（即周期为 0.1ms）。

6.10.2 连续插补中 PWM 输出功能

EMC-E5000 控制器支持连续插补中插入 PWM 指令。EMC-E3000 暂不支持连续插补中插入 PWM 指令，相关函数如表 6.23 所示。

表 6.23 连续插补中 PWM 输出功能相关函数说明

名称	功能	参考
dmc_set_pwm_enable_extern	设置 PWM 使能状态	8.14 节
dmc_set_pwm_onoff_duty	设置 PWM 开关状态对应的占空比	
dmc_conti_set_pwm_output	连续插补中 PWM 输出设置	
dmc_conti_set_pwm_follow_speed	连续插补中 PWM 速度跟随	
dmc_conti_delay_pwm_to_start	连续插补中相对于轨迹段起点 PWM 滞后	
dmc_conti_ahead_pwm_to_stop	连续插补中相对于轨迹段终点 PWM 提前	
dmc_conti_write_pwm	连续插补中缓冲区立即 PWM 输出	

连续插补中控制 PWM 输出分为不跟随模式和跟随模式。

不跟随模式控制 PWM 输出一般步骤如下：

- 1) 使用函数 dmc_set_pwm_enable 使能 PWM 输出；
- 2) 使用函数 dmc_set_pwm_output 设置 PWM 占空比为 0，设置 PWM 频率（设置 PWM 占空比为 0，即此时 PWM 功能已使能，但暂不输出 PWM 波形）；
- 3) 使用函数 dmc_conti_open_list 打开连续插补缓冲区；
- 4) 添加连续插补中 PWM 输出指令 dmc_conti_set_pwm_output；
- 5) 添加连续插补运动指令；
- 6) 使用函数 dmc_conti_start_list 启动连续插补运动；
- 7) 使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

跟随模式控制 PWM 输出一般步骤如下：

- 1) 使用函数 dmc_set_pwm_enable_extern 使能 PWM 输出；
- 2) 使用函数 dmc_conti_set_pwm_follow_speed 设置 PWM 跟随模式，函数中跟随模式 mod 有 5 种模式，不同模式，说明如下

a. 跟随模式 $\text{mode}=0$ （不跟随，保持状态），PWM 一直输出，输出的频率为步骤 2 中设置的频率，输出的占空比为步骤 4 中设置的占空比，具体采用打开（fOnDuty）还是关闭状态（fOffDuty）的占空比，由步骤 5 中添加的 PWM 指令中参数 on_off(0 采用关闭状态占空比，1 采用打开状态占空比)确定。

b. 跟随模式 $\text{mode}=1$ （不跟随，输出低电平），根据步骤 5 中添加的 PWM 输出指令中参数 on_off 确定。

当 on_off = 1（打开模式），输出低电平；

当 on_off = 0（关闭模式），输出步骤 2 中设置的频率，步骤 4 中关闭状态的占空比。

c. 跟随模式 $\text{mode}=2$ （不跟随，输出高电平），根据步骤 5 中添加的 PWM 输出指令中参数 on_off 确定。

当 on_off = 1（打开模式），输出高电平；

当 on_off = 0（关闭模式），输出步骤 2 中设置的频率，步骤 4 中关闭状态的占空比。

d. 跟随模式 $\text{mode}=3$ （跟随，占空比自动调整），根据步骤 5 中添加的 PWM 输出指令中参数 on_off 确定。

当 on_off = 1（打开模式），输出频率为步骤 2 中函数 dmc_conti_set_pwm_follow_speed 设置的 outvalue 值，跟随的占空比 =（当前轨迹段插补速度/MaxVel）*MaxValue。MaxVel 为 dmc_conti_set_pwm_follow_speed 函数设置的最大运行速度，MaxValue 为 dmc_conti_set_pwm_follow_speed 函数设置的最大占空比。

当 on_off = 0（关闭模式），输出的频率为步骤 2 中函数 dmc_conti_set_pwm_follow_speed 设置的 Outvalue，占空比为步骤 4 中关闭状态（fOffDuty）对应的占空比。

e. 跟随模式 $\text{mode}=4$ （跟随，频率自动调整），根据步骤 5 中添加的 PWM 输出指令中参数 on_off 确定。

当 on_off = 1（打开模式），输出的占空比为步骤 2 中函数 dmc_conti_set_pwm_follow_speed 设置的 outvalue 值，跟随的频率 =（当前轨迹段插补速度/MaxVel）*MaxValue。MaxVel 为 dmc_conti_set_pwm_follow_speed 函数设置的最大运行速度，MaxValue 为 dmc_conti_set_pwm_follow_speed 函数设置的最大频率。

当 on_off = 0（关闭模式），输出步骤 2 中函数 dmc_conti_set_pwm_follow_speed 设置的最大频率 MaxValue，步骤 4 中关闭状态（fOffDuty）对应的占空比。

3) 使用函数 dmc_conti_open_list 打开连续插补缓冲区；

4) 使用函数 dmc_set_pwm_onoff_duty 设置 PWM 打开（fOnDuty）及关闭状态（fOffDuty）的占空比；

5) 添加连续插补中 PWM 输出指令，如起点 PWM 滞后输出 dmc_conti_delay_pwm_to_start，终点 PWM 提前输出 dmc_conti_ahead_pwm_to_stop，缓存区立即 PWM 输出 dmc_conti_write_pwm 等；

- 6) 添加连续插补运动指令;
- 7) 使用函数 `dmc_conti_start_list` 启动连续插补运动;
- 8) 使用函数 `dmc_conti_close_list` 关闭连续插补缓冲区。

注意：步骤 5 添加的 PWM 跟随输出指令，配置参数 `on_off = 1`，打开了模式，必须在关闭插补缓存区（步骤 8）之前，再次调用步骤 5 中的 PWM 跟随输出指令，配置参数 `on_off = 0`，将模式关闭，即打开关闭必须成对出现，否则再次运行会出现 PWM 不输出的情况。

例：连续插补中 PWM 输出功能，不跟随模式

```
.....
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum, statemachine;
double MyfDuty, MyfFre;
MyCardNo = 0; //控制器链接号
Myenable = 1; //PWM 输出使能状态：使能
MyPwmNo = 0; //PWM 输出通道为 0 通道，即本地 OUT2
MyfDuty = 0; //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 0; //PWM 输出频率为 0Hz
LTDMC.dmc_set_pwm_enable_extern(MyCardNo, MyPwmNo, Myenable);
//使能 PWM 输出
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
//设置 PWM 输出，占空比为 0，频率为 0Hz
//设置插补速度曲线参数，插补运动最大矢量速度 500unit/s，加减速时间 0.1s
MyaxisNum = 2; //插补运动轴数为 2
statemachine = 0;
for(ushort Axis = 0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
MyCrd = 0; //参与插补运动的坐标系 0
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 500, 0.1, 0.1, 0);
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 4000 }, 0, 0); //第一段轨迹，直线插补，相对模式
LTDMC.dmc_conti_set_pwm_output(MyCardNo, MyCrd, MyPwmNo, 0.8, 200); //下一段运动开始，改变占空
比和频率
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
```

```
{ 8000, 8000 }, 0, 0); //第二段轨迹，直线插补，相对模式
```

LTDMC.dmc_conti_set_pwm_output(MyCardNo, MyCrd, MyPwmNo, 0.5, 100000); //下一段运动开始, 改变占空比和频率

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
```

```
{ 10000, 10000 }, 0, 0); //第三段轨迹，直线插补，相对模式
```

```
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
```

```
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
```

```
.....
```

运行结果：

当第二段直线插补，PWM 通道 0 输出 PWM 波形，频率为 200Hz，占空比为 80%（该 PWM 输出将一直保持，直到下次被改变）。

当第三段直线插补，PWM 通道 0 输出 PWM 波形，改变频率为 100kHz，占空比为 50%。

例：连续插补中 PWM 输出功能，跟随模式 0

```
.....
```

```
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum, statemachine;
```

```
ushort MyPwmMode, MyOnOff, MyDeMode, MyAhMode ;
```

```
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue;
```

```
double MyOutValue, MyDeVal, MyRevTime, MyAhVal ;
```

```
MyCardNo = 0; //控制器链接号
```

```
Myenable = 1; //PWM 输出使能状态：使能
```

```
MyPwmNo = 0; //PWM 输出通道为 0 通道，即本地 OUT2
```

```
MyfDuty = 0; //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
```

```
MyfFre = 20000; //PWM 输出频率为 20000Hz
```

```
LTDMC.dmc_set_pwm_enable(MyCardNo, MyPwmNo, Myenable);
```

```
//使能 PWM 输出
```

```
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
```

```
//设置 PWM 输出，占空比为 0，频率为 20000Hz
```

```
MyfDuty = 0.8; //设置 PWM 打开状态的占空比为 80%
```

```
MyOffFre = 0.05; //设置 PWM 关闭状态的占空比为 5%
```

```
LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
```

```
//设置 PWM 开关状态对应的占空比
```

```
MyCrd = 0; //参与插补运动的坐标系 0
```

```
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 500, 0.1, 0.1, 0);
```

```
//设置插补速度曲线参数，插补运动最大矢量速度 500unit/s，加减速时间 0.1s
```

```
MyaxisNum = 2; //插补运动轴数为 2
```

```
statemachine = 0;
```

```
for(ushort Axis = 0; Axis < MyaxisNum; Axis ++)
```

```
{
```

```
LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
```

```
if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
```

```
{
    MessageBox.Show("轴状态机错误");
    return;
}
}
MyPwmMode = 0; //设置 PWM 跟随模式为 0（即非速度跟随，保持状态）
MyMaxVel = 0;
MyMaxValue = 0;
MyOutValue = 0; //当 PWM 跟随模式为 0、1、2 时，此三个参数无意义
LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
MyMaxValue, MyOutValue); //设置 PWM 跟随模式为 0（即非速度跟随，保持状态）

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区

MyOnOff = 1 ; //设置 PWM 输出状态：打开
MyDeMode = 0 ; //设置 PWM 输出滞后模式：滞后时间
MyDeVal = 2 ; //设置 PWM 输出滞后时间：2s
MyRevTime = 0 ; //保留参数，固定值为 0
LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyDeVal, MyDeMode,
MyRevTime); //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 0 }, 0, 0); //第一段轨迹，直线插补，相对模式
MyOnOff = 0 ; //设置 PWM 输出状态：关闭
MyAhMode = 0; //设置 PWM 输出提前模式：提前时间
MyAhVal = 1; //设置 PWM 输出提前时间：1s
MyRevTime = 0; //保留参数
LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyAhVal,
MyAhMode,
MyRevTime); //相对于轨迹段终点,提前 1 秒，控制 PWM 输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 0, 4000 }, 0, 0); //第二段轨迹，直线插补，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....
```

运行结果：

当第一段直线插补开始 2s 后，PWM 通道 0 输出 PWM 波形，频率为 20kHz，占空比为 80%（该 PWM 输出将一直保持，直到下次被改变）。

当第二段直线插补结束前 1 秒时，PWM 通道 0 输出 PWM 波形，频率为 20kHz，占空比为 5%。

例：连续插补中 PWM 输出功能，跟随模式 1

.....

```
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum, statemachine;
```

```
ushort MyPwmMode, MyOnOff, MyDeMode, MyAhMode ;
```

```
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue;
```

```
double MyOutValue, MyDeVal, MyRevTime, MyAhVal ;
```

```
MyCardNo = 0; //控制器链接号
```

```
Myenable = 1; //PWM 输出使能状态：使能
```

```
MyPwmNo = 0; //PWM 输出通道为 0 通道，即 7 号轴的脉冲端口（PUL+与
```

PUL-）

```
MyfDuty = 0; //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
```

```
MyfFre = 20000; //PWM 输出频率为 20000Hz
```

```
LTDMC.dmc_set_pwm_enable_extern(MyCardNo, MyPwmNo, Myenable);
```

//使能 PWM 输出

```
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
```

//设置 PWM 输出，占空比为 0，频率为 20000Hz

```
MyfDuty = 0.8; //设置 PWM 打开状态的占空比为 80%
```

```
MyOffFre = 0.05; //设置 PWM 关闭状态的占空比为 5%
```

```
LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
```

//设置 PWM 开关状态对应的占空比

```
MyCrd = 0; //参与插补运动的坐标系 0
```

```
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 1000, 0.1, 0.1, 0);
```

//设置插补速度曲线参数，插补运动最大矢量速度 1000unit/s，加减速时间 0.1s

```
MyaxisNum = 2; //插补运动轴数为 2
```

```
statemachine = 0;
```

```
for(ushort Axis = 0; Axis < MyaxisNum; Axis ++)
```

```
{
```

```
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis, ref statemachine); //获取轴状态机
```

```
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
```

```
    {
```

```
        MessageBox.Show("轴状态机错误");
```

```
        return;
```

```
    }
```

```
}
```

```
MyPwmMode = 1; //设置 PWM 跟随模式为 1（即不跟随，输出低电平）
```

```
MyMaxVel = 0;
```

```
MyMaxValue = 0;
```

```
MyOutValue = 0; //当 PWM 跟随模式为 0、1、2 时，此三个参数无意义
```

```
LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
```

```
MyMaxValue, MyOutValue); //设置 PWM 跟随模式为 1（不跟随，输出低电平）
```

```
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
```

//打开连续插补缓冲区

```
MyOnOff = 1;           //设置 PWM 输出状态：打开
MyDeMode = 0;          //设置 PWM 输出滞后模式：滞后时间
MyDeVal = 2;           //设置 PWM 输出滞后时间：2s
MyRevTime = 0;         //保留参数，固定值为 0
LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyDeVal, MyDeMode,
MyRevTime);           //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 0 }, 0, 0);    //第一段轨迹，直线插补，相对模式
MyOnOff = 0;           //设置 PWM 输出状态：关闭
MyAhMode = 0;          //设置 PWM 输出提前模式：提前时间
MyAhVal = 1;           //设置 PWM 输出提前时间：1s
MyRevTime = 0;         //保留参数
LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyAhVal,
MyAhMode, MyRevTime); //相对于轨迹段终点,提前 1 秒，控制 PWM 输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 0, 4000 }, 0, 0);    //第二段轨迹，直线插补，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
.....
```

运行结果：

当第一段直线插补开始 2s 后，PWM 通道 0 输出低电平（该 PWM 输出将一直保持，直到下次被改变）。

当第二段直线插补结束前 1 秒时，PWM 通道 0 输出 PWM 波形，频率为 20kHz，占空比为 5%。

例：连续插补中 PWM 输出功能，跟随模式 3

```
.....
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum, MyPwmMode, MyOnOff, MyDeMode,
MyAhMode, statemachine;
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue, MyOutValue, MyDeVal, MyRevTime,
MyAhVal;
MyCardNo = 0;           //控制器链接号
Myenable = 1;           //PWM 输出使能状态：使能
MyPwmNo = 0;            //PWM 输出通道为 0 通道
MyfDuty = 0;            //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 20000;         //PWM 输出频率为 20000Hz
LTDMC.dmc_set_pwm_enable(MyCardNo, MyPwmNo, Myenable);
//使能 PWM 输出
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
//设置 PWM 输出，占空比为 0，频率为 20000Hz
MyfDuty = 0.8;          //设置 PWM 打开状态的占空比为 80%
```

```
MyOffFre = 0.05;           //设置 PWM 关闭状态的占空比为 5%
LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
                           //设置 PWM 开关状态对应的占空比
MyCrd = 0;                 //参与插补运动的坐标系 0
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 1000, 0.1, 0.1, 0);
                           //设置插补速度曲线参数，插补运动最大矢量速度 1000unit/s，加减速时间 0.1s
MyaxisNum = 2;             //插补运动轴数为 2
statemachine = 0;
for(ushort Axis =0; Axis < MyaxisNum; Axis++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
MyPwmMode = 3;             //设置 PWM 跟随模式为 3（即速度跟随，占空比自动调整，频率固定）
MyMaxVel = 4000;           //速度最大值为 4000 unit/s
MyMaxValue = 1;            //PWM 占空比最大值为 100%
MyOutValue = 30000;        //PWM 频率固定为 30kHz
LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
MyMaxValue, MyOutValue); //设置 PWM 跟随模式为 3（即速度跟随，占空比自动调整，频率固定）
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                           //打开连续插补缓冲区
MyOnOff = 1;               //设置 PWM 输出状态：打开
MyDeMode = 0;              //设置 PWM 输出滞后模式：滞后时间
MyDeVal = 2;               //设置 PWM 输出滞后时间：2s
MyRevTime = 0;             //保留参数，固定值为 0
LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyDeVal, MyDeMode,
MyRevTime);                //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 0 }, 0, 0); //第一段轨迹，直线插补，相对模式

MyOnOff = 0;               //设置 PWM 输出状态：关闭
MyAhMode = 0;              //设置 PWM 输出提前模式：提前时间
MyAhVal = 1;               //设置 PWM 输出提前时间：1s
MyRevTime = 0;             //保留参数
LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyAhVal,
MyAhMode, MyRevTime);      //相对于轨迹段终点,提前 1 秒，控制 PWM 输出
LTDMC.dmc_conti_set_speed_unit(MyCardNo, MyCrd, 3000);
//设置连续插补最大矢量速度 3000unit/s
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 0, 4000 }, 0, 0);//第二段轨迹，直线插补，相对模式
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{4000, 0 }, 0, 0); //第三段轨迹，直线插补，相对模式
```

```
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);//开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);//关闭连续插补缓冲区
```

.....

运行结果：

当第一段直线插补开始 2s 后，PWM 通道 0 启动 PWM 速度跟随输出（该 PWM 速度跟随输出将一直保持，直到被关闭），此时 PWM 输出波形占空比为 25%，频率为 30kHz。

当第二段直线插补运行时，PWM 输出波形占空比为 75%，频率为 30kHz。

当第三段直线插补结束前 1 秒时，PWM 通道 0 将关闭 PWM 速度跟随输出，此时 PWM 输出波形频率为 30kHz，占空比为 5%。

6.11 异常减速停止时间设置功能的实现

EMC-E3000/5000 系列运动控制器支持异常减速停止时间设置功能，用户根据现场实际需求情况设定减速停止时间可达到理想的减速效果，相关函数如表 6.24 所示。

表 6.24 异常减速停止时间设置相关函数说明

名称	功能	参考
dmc_set_dec_stop_time	设置减速停止时间	8.27 节
dmc_get_dec_stop_time	读取减速停止时间设置	
dmc_set_vector_dec_stop_time	设置插补系减速停止时间	
dmc_get_vector_dec_stop_time	读取插补系减速停止时间设置	

注意：当发生异常停止时，如：限位信号被触发、减速停止命令被触发等进行减速停止时，减速停止时间都为 dmc_set_dec_stop_time 函数里设置的减速时间。

例：设置异常减速停止时间为 0.5 秒

```
.....
ushort MyCardNo, Myaxis;
double Stop_time;
MyCardNo=0; //控制器链接号
Myaxis=0; //轴号
Stop_time=0.5; //减速时间 0.5s
LTDMC.dmc_set_dec_stop_time(MyCardNo, Myaxis, Stop_time); //设置轴异常减速停止时间
.....
```

6.12 检测轴到位状态功能的实现

EMC-E3000/5000 系列运动控制器提供了检测轴到位状态函数，使用这些函数可以设置单轴运动中允许的误差范围，并检测单轴运动是否处于允许的误差范围内。相关函数如表 6.28 所示。

表 6.28 检测轴到位状态功能相关函数说明

名称	功能	参考
dmc_set_factor_error	设置位置误差带	8.28 节
dmc_check_success_pulse	检测指令到位	
dmc_check_success_encoder	检测编码器到位	

注意：

- 1) 该功能检测只适用于单轴运动。
- 2) 检测函数请在 dmc_check_done 检测到轴停止后调用，函数调用后会等待轴到位后返回，如果调用函数 100ms 内未到位，函数超时返回认为不到位。

例：检测轴到位状态

```
.....
ushort MyCardNo, Myaxis, statemachine;
double Myfactor;
int Myerror;

MyCardNo=0; //控制器链接号
Myaxis=0; //轴号
Myfactor=5; //编码器系数为 5
Myerror=10; //位置误差带为 10pulse
LTDMC.dmc_set_factor_error(MyCardNo, Myaxis, Myfactor, Myerror);
//设置位置误差带

statemachine = 0;
LTDMC.nmc_get_axis_state_machine(CardNo, Axis, ref statemachine); //获取轴状态机
if (statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.dmc_pmove_unit (MyCardNo, Myaxis, 1000, 1);
    //定长运动，位移为 1000 pulse、绝对模式,脉冲当量未设置默认为 1
    while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0)
        //判断轴运动状态，等待运动完成
    {
        ApplicatIOn.DoEvents();
    }
}
```

```

if (LTDMC.dmc_check_success_encoder(MyCardNo, Myaxis) == 0)//检测编码器到位状态
{
    MessageBox.Show("编码器不到位!");
}
else
{
    MessageBox.Show("编码器到位!");
}
.....

```

运行结果说明：

运动的目标位置脉冲数为 1000 pulse，假设当前编码器反馈脉冲数为 199 pulse

由于误差带设为 10 pulse，编码器系数设为 5

处理如下：

$$199 * 5 = 995(\text{pulse})$$

$$1000 - 995 = 5(\text{pulse})$$

在误差带范围[-10,10]之内，此时则认为编码器到位，否则认为编码器不到位。

6.13 通用 I/O 控制的实现

6.13.1 I/O 普通控制功能

用户可以使用 EMC-E3000/5000 系列运动控制器上的数字 I/O 口用于检测开关信号、传感器信号等输入信号，或者控制继电器、电磁阀等输出设备的信号，通用输出口初始电平默认为高电平。相关函数如表 6.29 所示。

表 6.29 通用 IO 普通控制功能相关函数说明

名称	功能	参考
dmc_read_inbit_ex/dmc_read_inbit_ex_ex	读取指定控制器的某一位输入口的电平状态	8.17 节
dmc_write_outbit	对指定控制器的某一位输出口置位	
dmc_read_outbit/dmc_read_outbit_ex	读取指定控制器的某一位输出口的电平状态	
dmc_read_inport/dmc_read_inport_ex	读取指定控制器的全部输入口的电平状态	
dmc_read_outport/dmc_read_outport-ex	读取指定控制器的全部输出口的电平状态	
dmc_write_outport	设置指定控制器的全部输出口的电平状态	

注意：在使用 dmc_write_outport 对控制器的全部输出口进行置位，使用 dmc_read_inport、dmc_read_outport 进行 IO 电平读取显示时，应该使用十六进制数进行赋值（尽量避免使用十进制数，特别是在不支持无符号变量的开发环境下）。在对 IO 电平进行控制与读取时，使用十六进制数赋值远比使用十进制数赋值更加直观、方便。

例：读取 0 号链接的通用输入口 1 的电平值，并对通用输出口 3 置高电平

```
.....
ushort MyCardNo,MyInbitno,Mybitno,MyOutValue = 1 ;
short MyInValue;
MyCardNo=0;                //控制器链接号
MyInbitno=1;               //定义通用输入口 1
Mybitno=3;                 //定义输出口 3
MyOutValue = 1;            //定义输出电平为高电平
MyInValue = LTDMC.dmc_read_inbit_ex( );
                           //读取通用输入口 1 的电平值，并赋值给变量 MyInValue
LTDMC.dmc_write_outbit(MyCardNo, Mybitno, MyOutValue); //对通用输出口 3 置高电平
.....
```

例：读取全部输入 IO 口的电平值并进行显示，对全部输出 IO 口的电平进行初始化

```
.....
ushort MyCardNo,Myportno;
ulong a;
int n;
uint MyOutputValue;
MyCardNo=0;                //控制器链接号
Myportno=0;                //IO 口组号
MyOutputValue = 0xFFFFBFA;
                           //(0x 表示 16 进制数) 定义输出口电平值，输出口 0、2、10 为低电平，其余端口为高电平
a= LTDMC.dmc_read_inport(MyCardNo, Myportno);
                           //读取指定控制器的全部输入端口的电平
n = (int)a;                //将 ulong 型的 a 强制转换为 int 型的 n
string MyInportValue = Convert.ToString(n, 2);
                           //将信号返回值转换成二进制数，并赋值给 MyInportValue
LTDMC.dmc_write_outport(MyCardNo, 0, MyOutputValue); //对全部输出口进行电平赋值
.....
```

6.13.2 I/O 延时翻转功能

EMC-E3000/5000 系列运动控制器支持 I/O 延时翻转功能。该函数执行后，首先输出一个与当前电平相反的信号，延时设置的时间后，再自动翻转一次电平。相关函数如表 6.30 所示。

表 6.30 IO 延时翻转相关函数说明

名称	功能	参考
dmc_reverse_outbit	IO 输出延时翻转	8.17 节

例：对通用输出 IO 端口 0 进行延时翻转动作

```
.....
```

```

ushort MyCardNo, MyOutbitno;
double MyDelayTime;
MyCardNo=0;      //控制器链接号
MyOutbitno=0;    //通用输出口 0
MyDelayTime=0.5; //延时时间 0.5s
LTDMC.dmc_reverse_outbit(MyCardNo, MyOutbitno, MyDelayTime); //启动 IO 延时翻转
.....

```

6.13.3 I/O 计数功能

EMC-E3000/5000 系列总线运动控制器支持输入 IO 计数功能。该功能允许用户设置输入 IO 作为计数器使用。相关函数如表 6.31 所示。

表 6.31 IO 计数功能相关函数说明

名称	功能	参考
dmc_set_IO_count_mode	设置 IO 计数模式	8.17 节
dmc_set_IO_count_value	设置 IO 计数值	
dmc_get_IO_count_value	读取 IO 计数值	

例：设置通用输入 IO 端口 0 作为计数器

```

.....
ushort MyCardNo, MyInbitno, Mymode;
double Myfilter, Value;
uint MyCountValue;
MyCardNo = 0; //控制器链接号
MyInbitno = 0; //通用输入口 0
Mymode = 1; //上升沿计数
Myfilter = 0; //滤波时间保留
LTDMC.dmc_set_IO_count_mode(MyCardNo, MyInbitno, Mymode, Myfilter); //设置计数模式
MyCountValue=0; //计数值为 0
LTDMC.dmc_set_IO_count_value(MyCardNo, MyInbitno, MyCountValue); //计数器清零
Value = 0; //给变量 Value 赋初始值
LTDMC.dmc_get_IO_count_mode(MyCardNo, MyInbitno, ref Mymode, ref Value);
//读取计数器值，并赋值给变量 Value
.....

```

6.14 位置比较功能的实现

EMC-E3000/5000 系列运动控制器提供了位置比较功能，位置比较的一般步骤是：

- 1、配置比较器；
- 2、清除比较器数据；
- 3、添加/更新比较位置点；
- 4、开始运动并查看比较状态。

6.14.1 一维低速位置比较功能

EMC-E3000/5000 系列运动控制器对每个轴都提供了一组一维低速位置比较，每轴最多都可以添加 256 个比较点。一维低速位置比较的触发延时时间在 1-2 总线周期以内。相关函数如表 6.32 所示。

表 6.32 一维低速位置比较相关函数说明

名称	功能	参考
dmc_compare_set_config	设置一维位置比较器	8.15 节
dmc_compare_clear_points	清除一维位置比较点	
dmc_compare_add_point_cycle_unit	添加一维位置比较点	
dmc_compare_get_current_point_unit	读取当前一维比较点位置	
dmc_compare_get_points_runned	查询已经比较过的一维比较点个数	
dmc_compare_get_points_remained	查询可以加入的一维比较点个数	

注意：（1）每轴的位置比较都是独立进行的。

（2）执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行的，即如果有一个比较点没有被触发比较动作，那么后面的比较点是不会起作用的。

例：一维低速位置比较

```
.....
ushort MyCardNo,Myaxis,Myenable,Mycmp_source,Mydir,MyactIO ,Cycle,Level,statemachine;
double Mypos;
uint Myactpara,Bitno;

MyCardNo=0;           //控制器链接号
Myaxis=0;             //轴号
Myenable=1;           //设置比较使能
Mycmp_source = 0;      //设置比较源为指令位置
Mypos = 10000;         //设置比较位置为 10000unit
Mydir = 1;             //设置比较模式为大于等于
Cycle = 1000;          //1000 倍总线通信周期
Level = 0;             //输出 IO 端口 0 输出低电平，维持 1000 倍总线周期
Bitno = 0;             //输出 IO 端口 0
LTDMC.dmc_compare_set_config(MyCardNo, Myaxis, Myenable, Mycmp_source);
                        //设置 0 轴比较器
```

```

LTDMC.dmc_compare_clear_points(MyCardNo, Myaxis);//清除比较点
LTDMC.dmc_set_positlOn(MyCardNo, Myaxis, 0);      //设置 0 号轴的指令脉冲计数器绝对位置为 0
LTDMC.dmc_compare_add_point_cycle_unit (MyCardNo, Myaxis, Mypos, Mydir, Bitno, Cycle,Level);
            //添加比较点，位置 10000pulse，模式大于等于，触发时动作为输出端口 0 输出低电
运行，维持 1000 倍总线周期时间。
LTDMC.dmc_set_profile _unit (MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);
            //设置梯形速度曲线
statemachine = 0;
LTDMC.nmc_get_axis_state_machine(MyCardNo, MyAxis,ref statemachine);//获取轴状态机
if(statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
{
    LTDMC.dmc_pmove _unit (MyCardNo, Myaxis, 50000, 0);
            //定长运动，位移为 50000，相对模式
}
.....

```

运行结果：当运动到 10000 时，通用输出口 0 电平输出低电平，保持时间 1000 倍总线通信周期。

6.14.2 二维低速位置比较功能

EMC-E3000/5000 系列运动控制器提供了一组二维低速位置比较，最多都可以添加 256 个比较点。二维低速位置比较的触发延时时间小于 1ms。相关函数如表 6.33 所示。

表 6.33 二维低速位置比较相关函数说明

名称	功能	参考
dmc_compare_set_config_extern	设置二维位置比较器	8.21 节
dmc_compare_clear_points_extern	清除二维位置比较点	
dmc_compare_add_point_extern_cycle_2d	添加二维位置比较点	
dmc_compare_get_current_point_extern_unit	读取当前二维位置比较点位置	
dmc_compare_get_points_runned_extern	查询已经比较过的二维比较点个数	
dmc_compare_get_points_remained_extern	查询可以加入的二维比较点个数	

注意：执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行的，即如果有一个比较点没有被触发比较动作，那么后面的比较点是不会起作用的。

例：二维低速位置比较

```

.....
ushort MyCardNo, Mycrd, Myenable, Mycmp_source, Mydir, MyactlOn,statemachine;
uint Myactpara;
MyCardNo = 0;                                //控制器链接号

```

```
Mycrd = 0; //坐标系号
Myenable = 1; //设置比较使能
Mycmp_source = 0; //比较源：指令位置
Mydir=1; //比较模式：大于等于
Cycle = 1000; //1000 倍总线通信周期
Level = 0; //输出 IO 端口 0 输出低电平，维持 1000 倍总线周期
Bitno = 0; //输出 IO 端口 0
statemachine = 0;
for(ushort Axis =0; Axis < 2; Axis ++)
{
    LTDMC.nmc_get_axis_state_machine(MyCardNo, Axis,ref statemachine);//获取轴状态机
    if(statemachine != 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
    {
        MessageBox.Show("轴状态机错误");
        return;
    }
}
LTDMC.dmc_compare_set_config_extern(MyCardNo, Myenable, Mycmp_source);//设置二维比较器
LTDMC.dmc_compare_clear_points_extern(MyCardNo);//清除二维位置比较点
LTDMC.dmc_compare_add_point_extern_cycle_2d(MyCardNo, new ushort[] { 0, 1 }, new int[] { 5000, 5000 }, Mydir,Bitno,Cycle,Level);//添加二维位置比较点
LTDMC.dmc_set_vector_profile_unit(MyCardNo, Mycrd, 0, 3000, 0.01, 0.01, 0);
//设置插补运动速度曲线
LTDMC.dmc_arc_move_center_unit(MyCardNo, Mycrd,2, new ushort[] { 0, 1 }, new double[] { 0, 0 }, new double[] { 5000, 0 }, 0, 0, 0);
//圆弧插补运动
.....
```

6.15 位置锁存功能的实现

6.15.1 软件位置锁存功能

EMC-E3000/5000 系列运动控制器提供了软件位置锁存功能。软件位置锁存可以在软件层实现精确定位功能。当外部信号变化时，精确锁存到这一瞬间运动机构所在的位置，用户可以在之后查询这个位置的数值。每个锁存器都可以配置任意一个输入信号用于触发锁存动作，支持 4 组锁存器，可以用指令选择输入信号导通或截止的瞬间位置进行锁存，也可以选择锁存编码器反馈的位置或输出脉冲的位置。

EMC-E3000/5000 系列运动控制器支持多种软件位置锁存模式，可实现定位功能，其包括单次锁存、连续锁存。单次锁存在锁存信号有效后可锁存一次，再次锁存需复位锁存标志。

连续锁存可实现对多个位置依次进行锁存。在每次锁存后，不需要再复位锁存标志。锁存次数超过 1000 次，剔除最先的触发位置保存最新的触发位置。连续锁存间隔时间需大于 10ms。

软件位置锁存相关函数如表 6.37：

表 6.37 软件位置锁存相关函数说明

名称	功能	参考
dmc_softlrc_set_mode	配置锁存器	8.23 节
dmc_softlrc_get_mode	读取锁存器配置	
dmc_softlrc_set_source	配置锁存源	
dmc_softlrc_get_source	读取锁存源配置	
dmc_softlrc_get_value_unit	读取锁存值	
dmc_softlrc_get_number	读取锁存器已锁存个数	
dmc_softlrc_reset	复位指定控制器的锁存器	

例：软件位置连续锁存

.....

```

ushort MyCardNo, Myaxis, Mylrc_logic, Mylrc_mode, statemachine;
ushort Mylatch_source, Latch, Mylrc_inbit, Mylrc_enable;
double Myfilter;
int number;
double[] My_latch_Value;
number = 0;                                //锁存个数
MyCardNo = 0;                              //控制器链接号
Latch = 0;                                //锁存器 0
Myaxis = 0;                                //轴号
Mylrc_enable = 1                           //锁存器使能
Mylrc_inbit = 0;                           //配置锁存输入口为 0
Mylrc_logic = 0;                           //设置锁存输入口触发方式为下降沿触发
Mylrc_mode = 1;                            //设置锁存模式为连续锁存
Myfilter = 0;                              //保留参数
Mylrc_source = 0;                          //锁存源为指令位置
//第一步、配置锁存器 0，连续锁存模式，通用输入口 0 下降沿触发锁存指令位置
LTDMC.dmc_softlrc_set_mode(MyCardNo, Latch, Mylrc_enable, Mylrc_mode, Mylrc_inbit, Mylrc_logic,
Myfilter);
//第二步、设置 0 号锁存器的锁存源为 0 号轴的指令位置

```

```
LTDMC.dmc_softltc_set_source(MyCardNo, Latch, Myaxis, Myltc_source);  
//第三步、复位 0 号锁存器  
LTDMC.dmc_softltc_reset(MyCardNo, Latch);  
//设置梯形速度曲线  
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);  
statemachine = 0;  
LTDMC.nmc_get_axis_state_machine(MyCardNo, Myaxis, ref statemachine); //获取轴状态机  
if (statemachine == 4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态  
{  
    LTDMC.dmc_pmove(MyCardNo, Myaxis, 100000, 1); //定长运动，位移为 100000，绝对模式  
}  
while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0) //等待运动停止  
{  
    Application.DoEvents();  
}  
//第四步、读取锁存器已锁存个数  
LTDMC.dmc_softltc_get_number(MyCardNo, Latch, Myaxis, ref number);  
//第五步、读取锁存值  
if (number > 0)  
{  
    My_latch_Value = new double[number];  
    for (int i = 0; i < number; i++)  
    {  
        LTDMC.dmc_softltc_get_value_unit(MyCardNo, Latch, Myaxis, ref My_latch_Value[i]);  
    }  
}
```

6.16 螺距补偿功能的实现

EMC-E3000/5000 系列运动控制器提供螺距补偿功能。该功能使各轴根据设置的补偿值进行位置补偿。

相关函数如表 6.38 所示。

表 6.38 连续插补中刀向跟随功能相关函数说明

名称	功能	参考
dmc_enable_leadscrew_comp	设置螺距补偿的使能与禁止	8.25 节
dmc_set_leadscrew_comp_config_unit	配置螺距补偿参数	
dmc_get_leadscrew_comp_config_unit	读取配置的螺距补偿参数	

注意：

1. 一般在回零完成之后使用该功能，起点补偿值为 0
2. 补偿值为相对于起点的误差值

使用螺距补偿功能一般步骤如下：

- 1) 使用函数 dmc_enable_leadscrew_comp 使能螺距补偿；
- 2) 使用函数 dmc_set_leadscrew_comp_config_unit 设置螺距补偿数据；
- 3) 配置运动参数启动运动。

螺距补偿数据表一般由激光干涉仪测量而来，假设有如下图所示 9 个位置（8 个段，每段必须保持等长距离进行测量），若 2~9 点经测量补偿数据如下图所示。

1	2	3	4	5	6	7	8	9	
0	7	10	9	6	-1	-5	-3	0	→ 正方向运动补偿量 unit
0	-9	-5	-4	0	3	6	4	0	← 反方向运动补偿量 unit

假设轴1需测量段的总长度为8000unit，则每隔1000unit长度测量一次，起始点为0，

往正方向运动时，各点需要补偿的脉冲数为（0，7，10，9，6，-1，-5，-3，0）；

往负方向运动时，各点需要补偿的脉冲数为（0，-9，-5，-4，0，3，6，4，0）；

轴 1 往正方向运动时，将按照正方向运动补偿量进行补偿；往负方向运动时，将按照反方向运动补偿量进行补偿。

例：螺距补偿功能

```

ushort MyCardNo, MyCrd, MyaxisNum,axis,enable, npos;
double startpos, lenpos;
short err = 0;
MyCardNo = 0;           //控制器链接号
MyCrd = 0;              //参与插补运动的坐标系
MyaxisNum = 2;          //插补运动轴数为 2
axis = 0;               //螺距补偿轴
enable = 0;             //设置螺距补偿使能
npos = 9;               //螺距补偿点数
startpos = 0;           //补偿的起始位置
lenpos = 8000;          //补偿的长度
double [] comPos = new double[] { 0,7,10,9,6,-1,-5,-3,0}; // 正方向运动时，各点位置需要补偿的位置
double[] comNeg = new double[] { 0,-9,-5,-4,0,3,6,4,0 }; // 负方向运动时，各点位置需要补偿的位置
    
```

```

LTDMC.dmc_set_leadscrew_comp_config_unit(MyCardNo, axis, npos, startpos, lenpos, comPos, comNeg);
//配置螺距补偿参数
LTDMC.dmc_enable_leadscrew_comp(MyCardNo, axis, enable); //使能轴的螺距补偿
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 5000, 0.1, 0.1, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 5000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 1000,
1000 }, 0, 0); //第一段轨迹，直线插补，相对模式
LTDMC.dmc_conti_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 2000, 0 }, 1000, 1, 0, 0, 0); //第二段轨迹，圆弧插补，相对模式
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 3000,
1000 }, 0, 0); //第三段轨迹，直线插补，相对模式
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { -6000,
2000 }, 0, 0); //第四段轨迹，直线插补，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区

```

6.17 龙门功能的实现

EMC-E5000 系列提供了龙门跟随功能，相关函数如表 6.39 所示。

表 6.39 龙门相关函数

名称	功能	参考
dmc_set_gear_follow_profile	设置轴跟随模式参数	8.26 节
dmc_get_gear_follow_profile	读取轴跟随模式参数	
dmc_set_grant_error_protect_unit	设置龙门模式主从轴编码器跟随误差停止阈值	
dmc_get_grant_error_protect_unit	读取龙门模式编码器位置跟随误差停止阈值	

例：定长运动中龙门功能的实现

```

.....
ushort MyCardNo, Myaxis, Axis, Myposi_mode, Master_axis;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, MyDist, dstp_error, EMG_error;
MyCardNo = 0; //控制器链接号
Myaxis = 0; //轴号
Axis = 1; //跟随轴轴号
Master_axis = 0; //主轴轴号
MyMin_Vel = 500; //设置起始速度为 500unit/s
MyMax_Vel = 2000; //设置最大速度为 2000unit/s
MyTacc = 0.01; //设置加速时间为 0.02s
MyTdec = 0.01; //设置减速时间为 0.01s

```

```

MyStop_Vel = 500;                                     //设置停止速度为 1000unit/s
LTDMC.dmc_set_profile_unit(MyCardNo, Master_axis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
MyStop_Vel);                                           //设置 0 号轴速度曲线参数
MyDist = 10000;                                       //设置运动距离为 10000unit
Myposi_mode = 0;                                       //设置运动模式为相对坐标模式
LTDMC.dmc_set_gear_follow_profile(MyCardNo, Axis, 1, Master_axis, 1);
                                                         //1 号轴跟随 0 号主轴运动

dstp_error=100;                                       //减速停止误差阈值 100unit
EMG_error=100;                                         //立即停止误差阈值 100unit
LTDMC.dmc_set_grant_error_protect_unit(MyCardNo, Axis, 1, dstp_error, EMG_error)
                                                         //设置龙门模式主从轴编码器跟随误差停止阈值

LTDMC.dmc_pmove_unit(MyCardNo, Master_axis, MyDist, Myposi_mode); //0 号主轴启动运动

```

6.18 PVT 运动功能的实现

EMC-E3000/5000 系列运动控制器共提供四种 PVT 模式，分别为 PTT、PTS、PVT、PVTS 模式。其中 PTT、PTS 运动模式用于单轴速度规划功能，PVT、PVTS 运动则用于多轴轨迹规划功能，用户可以根据实际需求选择适合的 PVT 模式。

6.18.1 单轴任意速度规划功能的实现

EMC-E3000/5000 系列运动控制器提供了两种 PVT 模式来实现单轴任意速度规划的功能，分别为 PTT 运动模式和 PTS 运动模式。PTT 运动模式是用于单轴梯形速度的规划，而 PTS 运动模式则用于单轴 S 形速度的规划。

6.18.1.1 PTT 运动模式

PTT 模式非常灵活，能够实现单轴任意速度规划。用户通过直接输入位置和时间参数描述运动规律。相关函数如表 6.40 所示。

表 6.40 PTT 模式运动相关函数说明

名称	功能	参考
dmc_ptt_table_unit	向指定数据表传送数据，采用 PTT 描述方式	8.30 节
dmc_pvt_move	启动 PVT 运动	

例：PTT 模式运动

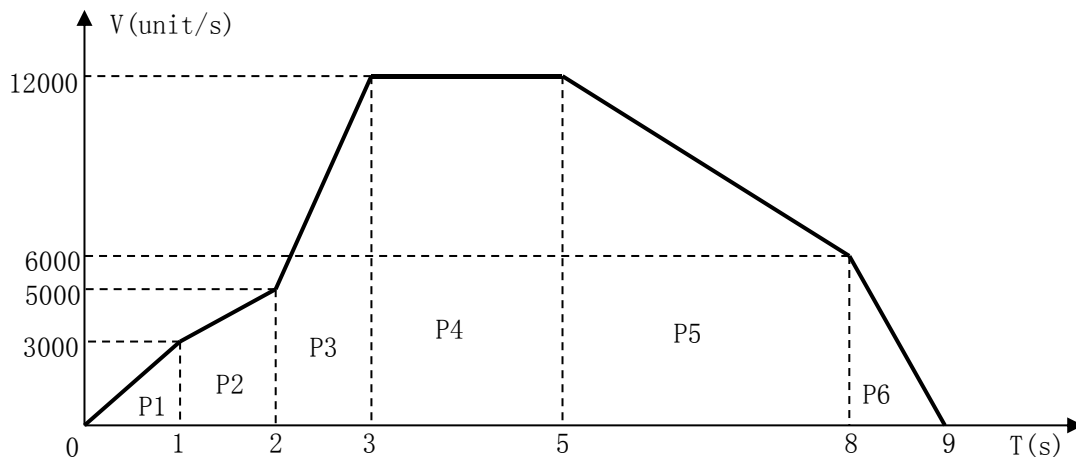


图 6.37 PTT 模式下的 V-T 曲线规划

规划如图 6.37 所示速度曲线，用 PTT 模式规划该速度曲线非常简单。

首先计算各段的位移量，即速度曲线和时间轴所围面积：P1=1500(unit)，P2=4000(unit)，P3=8500(unit)，P4=24000(unit)，P5=27000(unit)，P6=3000(unit)。

将各段位移量累加，得到 PTT 模式下各点的位置和时间数据，如表 6.41 所示。

表 6.41 PTT 模式数组数据

序号	位置 P(unit)	时间 T(s)
0	0	0
1	1500	1
2	5500	2
3	14000	3
4	38000	5
5	65000	8
6	68000	9

编写程序如下：

```

ushort MyCardNo, Myaxis, MyAxisNum, MyCount;
ushort []My_AxisList=new ushort [1];
MyCardNo = 0; //控制器链接号为 0
Myaxis=0; //轴号 0
MyAxisNum=1; //参与运动的轴数为 1
MyCount = 7; //有 7 组数据
My_AxisList[0] = 0; //0 号轴参与 PTT 运动
double[] MyPTime = new double[7] {0,1,2,3,5,8,9 };//定义 PTT 时间数组
double[] MyPPos = new double[7] { 0, 1500, 5500, 14000, 38000, 65000, 68000 };
//定义 PTT 位置数组
LTDMC.dmc_ptt_table_unit(MyCardNo,Myaxis,MyCount,MyPTime,MyPPos);
//采用 PTT 模式传递数据
LTDMC.dmc_pvt_move(MyCardNo, MyAxisNum, My_AxisList);//执行 PTT 运动
    
```

PTT 运动结果(位置-时间曲线、加速度-时间曲线)如图 6.38、6.39 所示。

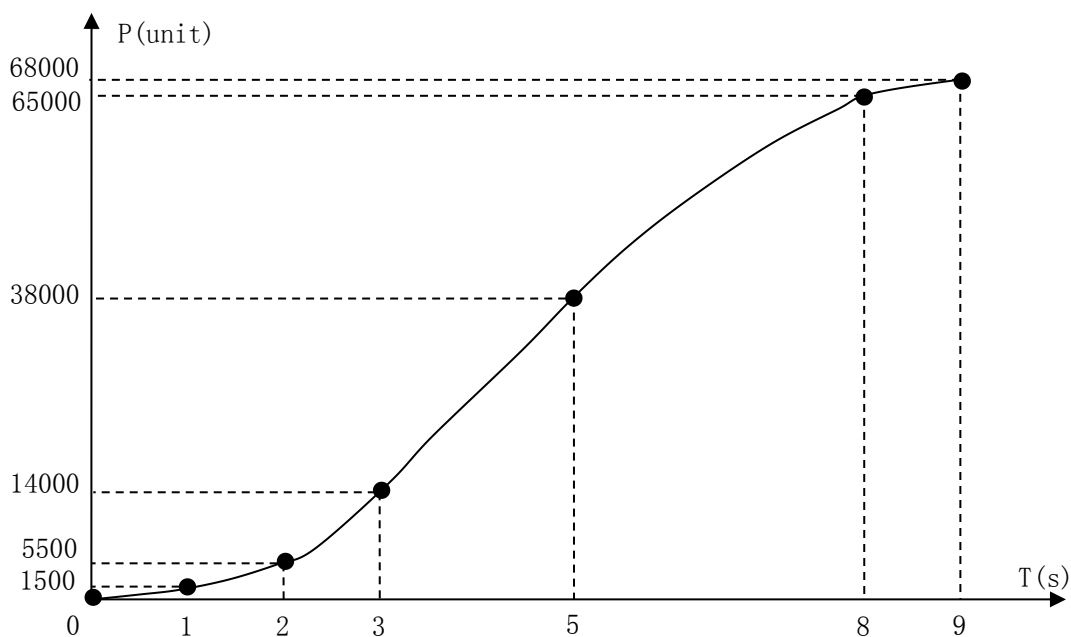


图 6.38 PTT 运动得到的位移曲线

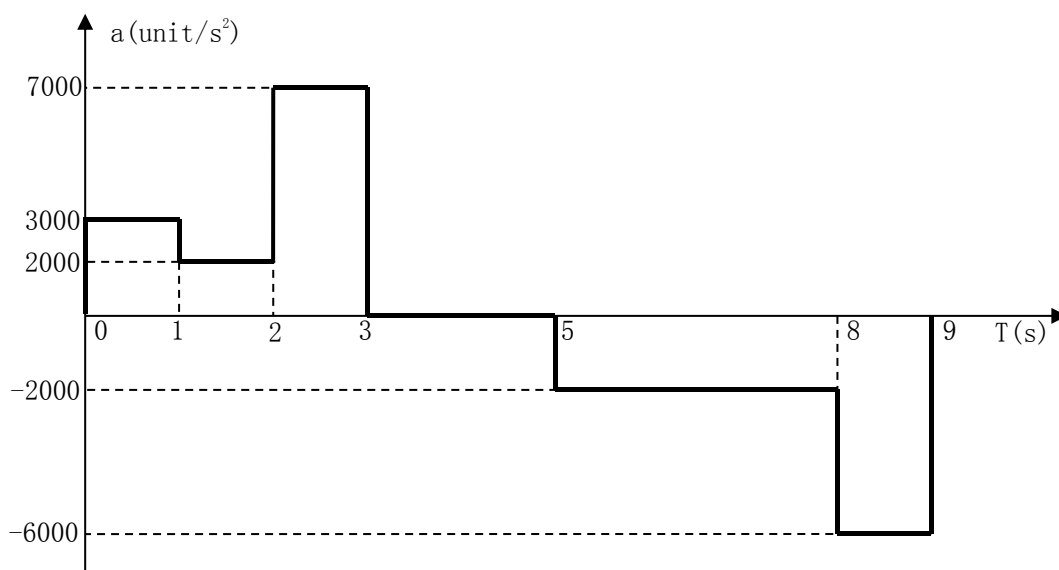


图 6.39 PTT 运动的加速度曲线

6.18.1.2 PTS 运动模式

PTS 运动模式是 PTT 的扩展功能模式，可以使各数据点的速度过渡更加平滑。用户通过输入位置、时间、百分比参数描述运动规律。数据点的百分比参数是指：相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比。相关函数如表 6.42 所示。

表 6.42 PTS 模式运动相关函数说明

名称	功能	参考
dmc_pts_table	向指定数据表传送数据，采用 PTS 描述方式	8.30 节
dmc_pvt_move	启动 PVT 运动	

例：PTS 模式运动

由图 6.39 可知，其加速度曲线存在突变，因此，运动过程中有冲击现象。如果要获得更加平滑的速度曲线，可以使用 PTS 模式运动，其速度曲线比图 6.37 所示的速度曲线平滑。

设置各点的速度百分比参数如表 6.43 所示；其对应的加速度-时间曲线如图 6.40 所示，每段的加减速时间为该段时间值×速度百分比；每段的加速度最大值与 PTT 模式下的加速度值不一样，这是因为内部算法作了平滑处理。其对应的位置-时间曲线、速度-时间曲线如图 6.41、6.42 所示。

表 6.43 PTS 模式数组数据

序号	P(unit)	T(s)	Percent(%)
0	0	0	0
1	1500	1	20
2	5500	2	40
3	14000	3	60
4	38000	5	0
5	65000	8	20
6	68000	9	80

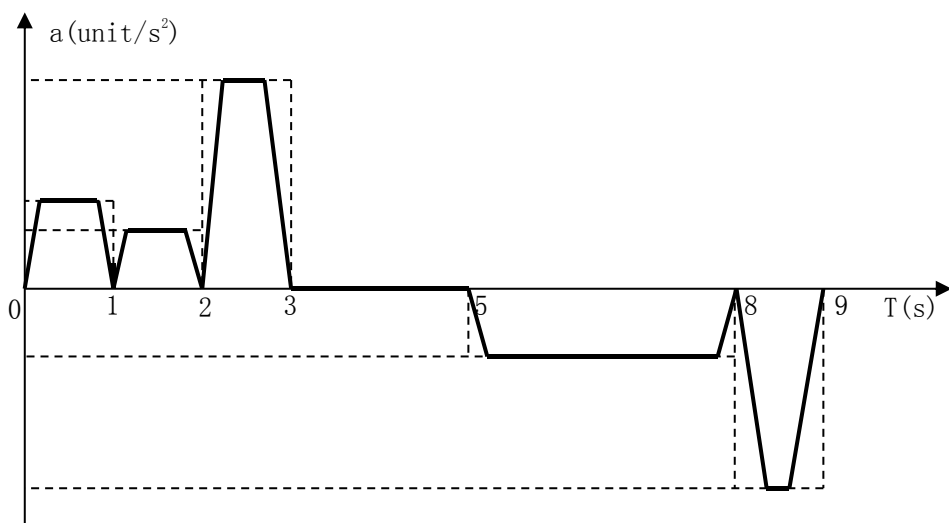


图 6.40 PTS 模式下的加速度曲线

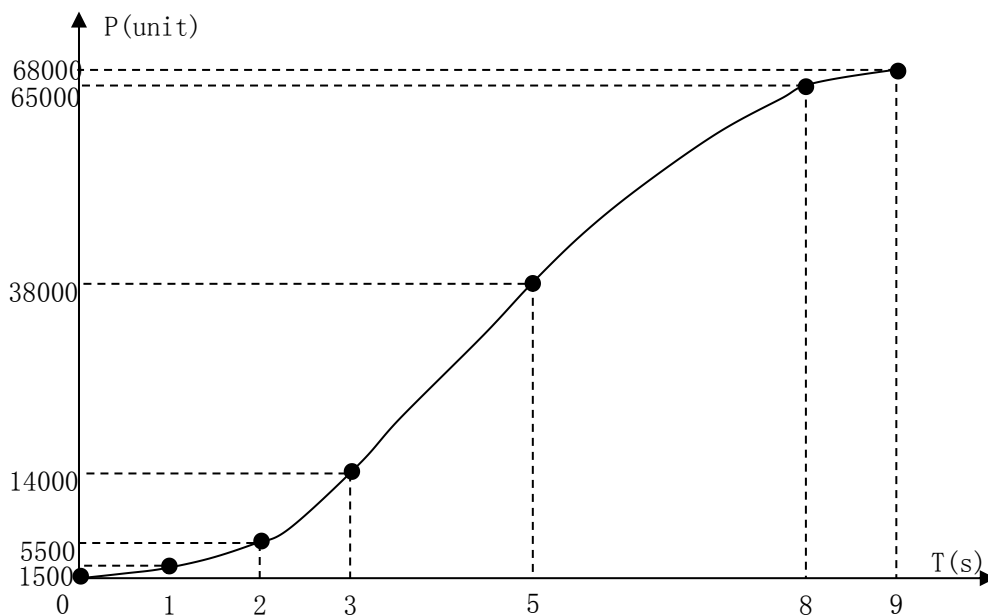


图 6.41 PTS 运动得到的位移曲线

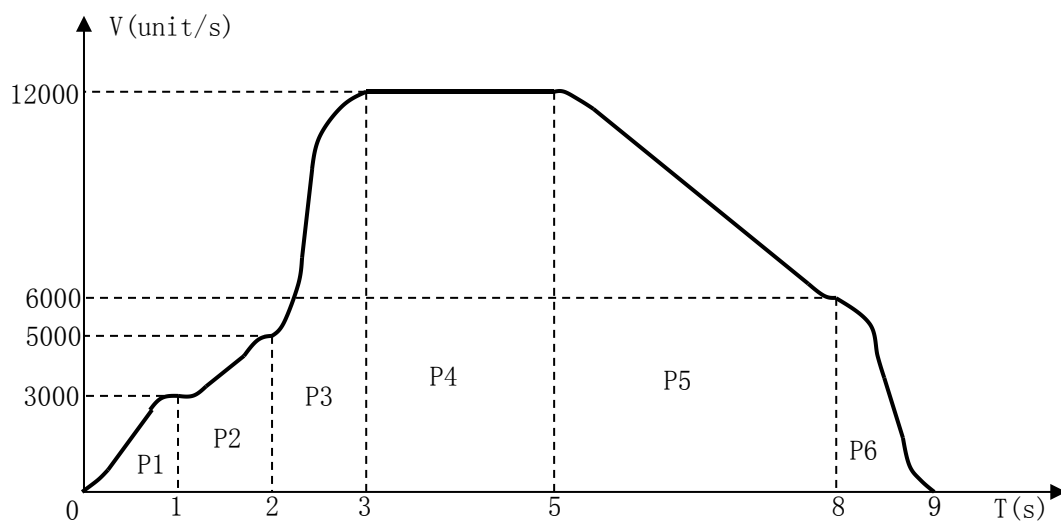


图 6.42 PTS 运动得到的速度曲线

编写程序如下：

```
.....
ushort MyCardNo, Myaxis, MyAxisNum, MyCount;
ushort[] My_AxisList = new ushort[1];
MyCardNo = 0;           //控制器链接号为 0
Myaxis = 0;             //轴号 0
MyAxisNum = 1;          //参与运动的轴数为 1
MyCount = 7;            //有 7 组数据
My_AxisList[0] = 0;     //0 号轴参与 PTS 运动
```

```
double[] MyPTime = new double[7] { 0, 1, 2, 3, 5, 8, 9 };    //定义 PTS 时间数组
double[] MyPPos = new double[7] { 0, 1500, 5500, 14000, 38000, 65000, 68000 };
                                                    //定义 PTS 位置数组
double[] MyPPer = new double[7] { 0, 20, 0, 60, 0, 20, 80 };//定义 PTS 百分比数组
LTDMC.dmc_pts_table_unit(MyCardNo, Myaxis, MyCount, MyPTime, MyPPos, MyPPer);
                                                    //采用 PTS 模式传递数据
LTDMC.dmc_pvt_move(MyCardNo, MyAxisNum, My_AxisList);      //执行 PTS 运动
.....
```

6.18.2 多轴高级轨迹规划功能的实现

EMC-E3000/5000 系列运动控制器具有 PVT 高级运动曲线规划的功能,当用户需要规划一些特殊的运动轨迹而使用单轴运动及插补运动无法满足需求时,可以尝试使用 PVT 来规划自己的运动轨迹。

EMC-E3000/5000 系列运动控制器共提供了两种 PVT 模式来实现多轴轨迹规划的功能,分别为 PVT、PVTS 运动模式。PVT 模式用于对各点的位置、时间、速度都有要求的轨迹规划,PVTS 模式用于只对各点的位置、时间有要求,而对各点的速度无太多要求的轨迹规划。

6.18.2.1 PVT 运动模式

PVT 模式使用一系列数据点的位置、速度、时间参数描述运动规律。

相关函数如表 6.44 所示。

表 6.44 PVT 模式运动相关函数说明

名称	功能	参考
dmc_pvt_table_unit	向指定数据表传送数据,采用 PVT 描述方式	8.30 节
dmc_pvt_move	启动 PVT 运动	

注意: 当设置的各点 P、V、T 数据不合理时,很难得到理想的轨迹曲线。

理想轨迹上取点越多,实际轨迹越接近理想轨迹。

例: PVT 模式运动

如图 6.43 所示,要控制运动平台按椭圆轨迹运动,但 EMC-E5000 控制器中并没有提供椭圆插补函数,用户可以使用 PVT 函数自己设计该轨迹。

设该椭圆的长半轴长 9000unit,短半轴长 7000unit;椭圆轨迹的角速度 ω 恒定,轨迹运动的总时间为 10s。该椭圆的方程为:

$$\begin{cases} x = 9000 \cos(\theta) + 9000 \\ y = 7000 \sin(\theta) \end{cases} \quad (7.1)$$

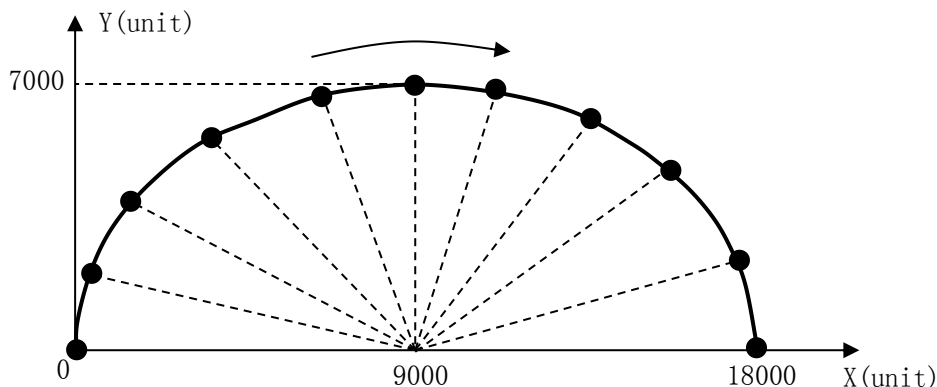


图 6.43 上半椭圆轨迹和分段

使用 PVT 模式运动上半椭圆轨迹的步骤为：

- 1、将该轨迹分成圆弧角相等的十段轨迹，如图 6.43 所示；计算各点坐标值，即得 P 值。程序如下：

```
double[] MyPPosX = new double[11]; //定义数组, 用于存储 PVT 的位置数据 (X 轴)
double[] MyPPosY = new double[11]; //定义数组, 用于存储 PVT 的位置数据 (Y 轴)

ushort a = 9000, b = 7000;           //定义椭圆长半轴、短半轴长
for (int i = 0; i < 11; i++)         //计算各点的 X、Y 坐标
{
    MyPPosX[i] = (a * Math.Cos((10 - i) * Math.PI / 10) + a);
                                     //存储 X 轴各点位置数据
    MyPPosY[i] = (b * Math.Sin((10 - i) * Math.PI / 10));
                                     //存储 Y 轴各点位置数据
}
```

- 2、根据各点坐标（即 P 值），计算出各点对应的速度的（V 值）和时间（T 值）。

对椭圆方程（6.1）式求导，可得 X、Y 轴方向的速度分量为：

$$\begin{cases} \dot{x} = -9000 \sin(\theta) \frac{d\theta}{dt} \\ \dot{y} = 7000 \cos(\theta) \frac{d\theta}{dt} \end{cases}$$

上式中 $\frac{d\theta}{dt}$ 即为角速度 ω 。

各点的 X、Y 轴方向的速度分量可由以下程序计算得出。程序如下：

```
double[] MyPVelX = new double[11]; //定义数组, 用于存储 PVT 中的速度数据 (X 轴)
double[] MyPTimeX = new double[11]; //定义数组, 用于存储 PVT 中的时间数据 (X 轴)
```

```
double[] MyPVelY = new double[11]; //定义数组，用于存储 PVT 中的速度数据（Y 轴）
double[] MyPTimeY = new double[11]; //定义数组，用于存储 PVT 中的时间数据（Y 轴）
double MyWVel; //定义角速度
for (int i = 0; i < 11; i++)
{
    MyPTimeX[i] = i; //存储 X 轴各点时间数据
    MyPTimeY[i] = i; //存储 Y 轴各点时间数据
}
MyWVel = -Math.PI / 10; //计算角速度
MyPVelX[0] = 0; MyPVelX[10] = 0; //起始点与终止点 X 轴速度设为 0
MyPVelY[0] = 0; MyPVelY[10] = 0; //起始点与终止点 Y 轴速度设为 0
for (int i = 1; i < 10; i++)
{
    MyPVelX[i] = -a * Math.Sin((10 - i) * Math.PI / 10) * MyWVel; //计算其他点 X 轴速度
    MyPVelY[i] = b * Math.Cos((10 - i) * Math.PI / 10) * MyWVel; //计算其他点 Y 轴速度
}
```

计算出各点 X、Y 轴的 P、V、T 数据如表 6.44 所示。

表 6.44 PVT 数据

	X 轴			Y 轴		
序号	P(unit)	V(unit/s)	T(s)	P(unit)	V(unit/s)	T(s)
0	0	0	0	0	0	0
1	440	873.731	1	2163	2091.479	1
2	1719	1661.927	2	4115	1778.117	2
3	3710	2287.443	3	5663	1292.603	3
4	6219	2688.048	4	6657	678.560	4
5	9000	2827.431	5	7000	-0.003	5
6	11781	2688.046	6	6657	-678.566	6
7	14290	2287.438	7	5663	-1292.608	7
8	16281	1661.921	8	4114	-1778.120	8
9	17560	873.724	9	2163	-2091.481	9
10	18000	0	10	0	0	10

3、使用 dmc_pvt_table_unit 函数向数据表传递数组数据。

程序如下：

```
ushort MyCardNo, MyCountX, MyCountY;
ushort[] My_AxisList = new ushort[] { 0, 1 }; //定义轴列表变量
MyCardNo = 0; //控制器链接号
MyCountX = 11; //X 轴 11 组数据
MyCountY = 11; //Y 轴 11 组数据
LTDMC.dmc_pvt_table_unit(MyCardNo, My_AxisList[0], MyCountX, MyPTimeX, MyPPosX, MyPVelX);
```

// 以 PVT 描述方式向 X 轴传送 PVT 数据

LTDMC.dmc_pvt_table_unit(MyCardNo, My_AxisList[1], MyCountY, MyPTimeY, MyPPosY, MyPVelY);

//以 PVT 描述方式向 Y 轴传送 PVT 数据

4、使用 dmc_pvt_move 函数执行 PVT 运动。

程序如下：

ushort My_AxisNum;

My_AxisNum = 2;

//参与 PVT 运动的轴数为 2

LTDMC.dmc_pvt_move(MyCardNo, My_AxisNum, new ushort[] { 0, 1 });//启动 PVT 运动

执行完上述程序后，得到 PVT 运动轨迹如图 6.44 所示。

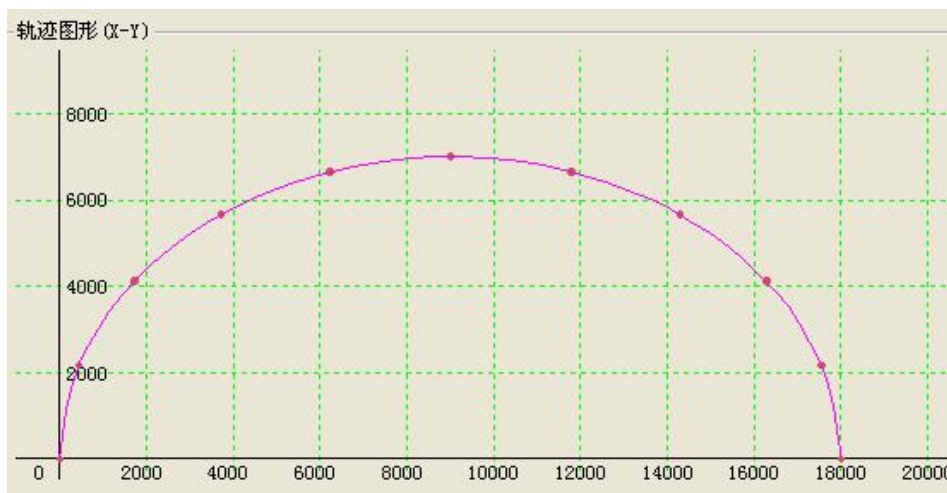


图 6.44 PVT 模式运动得到的上半椭圆轨迹

6.18.2.2 PVTS 运动模式

PVTS 运动模式只需要定义理想轨迹上数据点的位置和时间，以及起点速度和终点速度。控制器根据各数据点的位置、时间参数计算各点之间的轨迹位置和速度，确保各段轨迹的速度连续、加速度连续。相关函数如表 6.45 所示。

表 6.45 PVTS 模式运动相关函数说明

名称	功能	参考
dmc_pvts_table_unit	向指定数据表传送数据,采用 PVTS 描述方式	8.30 节
dmc_pvt_move	启动 PVT 运动	

例：PVTS 模式运动

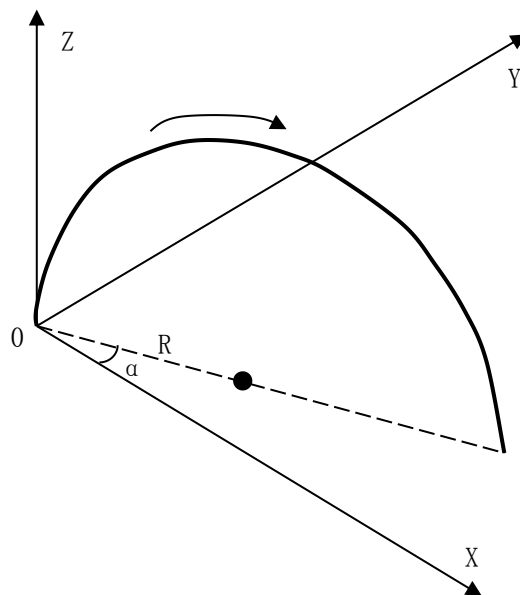


图 6.45 空间圆弧轨迹

如图 6.45 所示，设计空间圆弧轨迹。设其半径 $R=15000\text{unit}$ ，其映射在 XY 平面上的轨迹与 X 轴的夹角 $\alpha=\pi/6$ ，轨迹运动总时间为 10s。该空间圆弧的方程为：

$$\begin{cases} x = 15000 \cos(\frac{\pi}{6}) \cos(\theta) + 15000 \cos(\frac{\pi}{6}) \\ y = 15000 \sin(\frac{\pi}{6}) \cos(\theta) + 15000 \sin(\frac{\pi}{6}) \\ z = 15000 \sin(\theta) \end{cases} \quad (7.2)$$

若对轨迹上各点速度无精确要求，采用 PVTs 模式设计该轨迹的步骤如下：

- 1、将该轨迹分成角度相等的十份，计算各点的位置坐标，即 P 值。
- 2、根据各点的 P 值，计算出各点的 T 值。设每段轨迹运动的时间相等。
- 3、设定起点速度与终点速度都为 0。
- 4、使用 `dmc_pvts_table_unit` 函数向数据表传递数组数据。
- 5、使用 `dmc_pvt_move` 函数执行 PVT 运动。

编写程序如下：

```
.....
double[] MyPTime = new double[11];
double[] MyPPosX = new double[11];
double[] MyPPosY = new double[11];
double[] MyPPosZ = new double[11];
ushort MyCardNo, MyCount, R, My_AxisNum;
ushort[] My_AxisList = new ushort[] { 0, 1, 2 }; //轴号列表
double MyPVelBeginX=0, MyPVelEndX=0, MyPVelBeginY=0, MyPVelEndY=0, MyPVelBeginZ=0,
MyPVelEndZ=0; //各轴起始速度、停止速度设置
MyCardNo=0; //控制器链接号
R = 15000; //定义空间圆弧半径
```

```

MyCount = 11;                                //设置 X、Y、Z 轴的数据点数
My_AxisNum = 3;                               //0、1、2 号轴（即 X、Y、Z 轴）参加 PVT 运
动
for (int i = 0; i < 11; i++)
{
    MyPPosX[i] = ( R * Math.Cos(Math.PI / 6) * Math.Cos((10 - i) * Math.PI / 10) + R * Math.Cos(Math.PI
/ 6));    //计算 X 轴各点位置坐标
    MyPPosY[i] = (R * Math.Sin(Math.PI / 6) * Math.Cos((10 - i) * Math.PI / 10) + R * Math.Sin(Math.PI /
6));    //计算 Y 轴各点位置坐标
    MyPPosZ[i] = (R * Math.Sin((10 - i) * Math.PI / 10)); //计算 Z 轴各点位置坐标
}
for (int i = 0; i < 11; i++)
{
    MyPTime[i] = i;                            //各点时间坐标
}
LTDMC.dmc_pvts_table_unit(MyCardNo, My_AxisList[0], MyCount, MyPTime, MyPPosX, MyPVelBeginX,
MyPVelEndX);    //以 PVTS 描述方式向 X 轴传送 PVT 数据
LTDMC.dmc_pvts_table_unit(MyCardNo, My_AxisList[1], MyCount, MyPTime, MyPPosY, MyPVelBeginY,
MyPVelEndY);    //以 PVTS 描述方式向 Y 轴传送 PVT 数据
LTDMC.dmc_pvts_table_unit(MyCardNo, My_AxisList[2], MyCount, MyPTime, MyPPosZ, MyPVelBeginZ,
MyPVelEndZ);    //以 PVTS 描述方式向 Z 轴传送 PVT 数据
LTDMC.dmc_pvt_move(MyCardNo, My_AxisNum, My_AxisList); //启动 PVT 运动
.....

```

以 PVTS 模式得到的空间圆弧轨迹如图 6.46 所示。

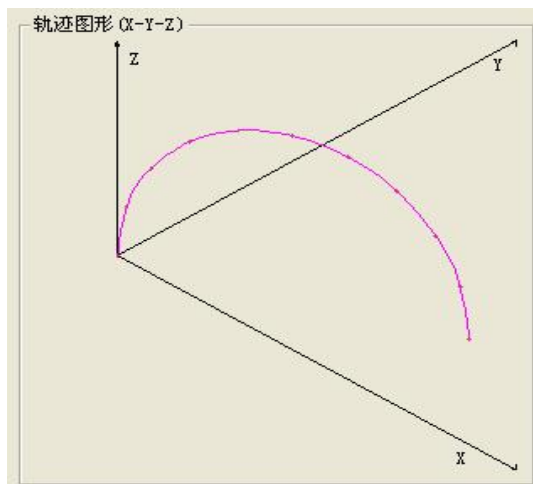


图 6.46 PVTS 模式运动得到的空间圆弧轨迹

6.19 看门狗功能的实现

```
ushort MyCardNo, event_mask, enable;
```

```
double imer_period
MyCardNo = 0; //控制器链接号
event_mask = 2 //停止轴使能
enable = 1; //使能
imer_period = 10 //看门狗触发时间；单位：ms
//第一步、设置看门狗事件： 看门狗事件位为停止轴使能
LTDMC.dmc_set_watchdog_action_event(MyCardNo, event_mask)
设置看门狗事件位：停止轴使能
//第二步：使能看门狗机制
LTDMC.dmc_set_watchdog_enable MyCardNo, timer_period, enable);
While ( true )
{
Thead.Sleep(1000);
LTDMC.dmc_reset_watchdog_timer(MyCardNo) //设复位看门狗计时器
}
```

6.20 软着陆功能的实现

软着陆功能主要用来实现 pmove 运动可以一个较低的速度平稳地接近目标位置，降低运动部件对接触点的冲击。如上图所示，重点想实现的功能为 pmove 结束时能以一个很低速度靠近结束点。将 pmove 分为两段，第一段以 v_s , v_m , v_e 进行规划，第二段以最大速度 v_e 进行规划。同样，强制变位置 dmc_update_target_positlOn_extern 也有该项功能。常用于晶圆取放等场合，相关函数如表 6.46 所示。

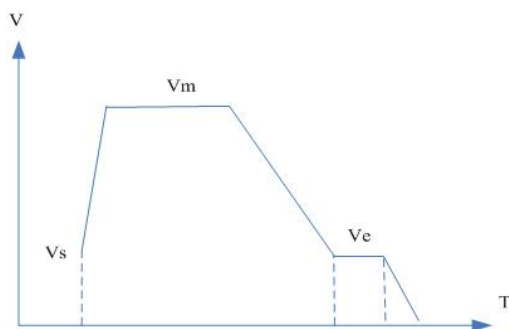


图 6.47 软着陆曲线示意图

表 6.46 软着陆运动功能相关函数说明

名称	功能	参考
dmc_pmove_extern_unit	实现 profile、pmove 整合，缩短指令时间	8.4 节
dmc_t_pmove_extern_unit	实现 profile、pmove 整合，缩短指令时间，并且实现软着陆	
dmc_update_target_positIon_extern_unit	强行改变指定轴的当前目标位置并且实现软着陆	

备注：软着陆不支持 S 形速度曲线

例：软着陆运动

```

.....
double start,speed,stop, acc,dec,dis,softdist;
ushort cardID,axis ,posi_mode;
cardID = 0;
axis = 0;                //轴号
dis =1000;               //定长运动距离
softdist =40;            //软着陆时低速运动距离
start = 10;              //启动速度
speed = 500;             //运行速度
stop = 10;               //停止速度，软着陆低速运动时速度
acc = 0.1;               //加速时间
dec = 0.1;               //减速时间
posi_mode = 1;           //绝对运动模式
LTDMC.dmc_t_pmove_extern_unit (cardID ,axis, dis, dis + softdist, start, speed, stop, acc, dec, posi_mode);
//执行软着陆运动
.....

```

例：强制变位软着陆运动

```

.....
double start,speed,stop, acc,dec,dis,softdist;
ushort cardID,axis ,posi_mode;
int pos;
cardID = 0;
axis = 0;                //轴号
dis =1000;               //定长运动距离
softdist =40;            //软着陆时低速运动距离
start = 10;              //启动速度
speed = 500;             //运行速度
stop = 10;               //停止速度，软着陆低速运动时速度

```

```

acc = 0.1;           //加速时间
dec = 0.1;           //减速时间
posi_mode = 1;       //绝对运动模式
s_para = 0;          //平滑时间
LTDMC.dmc_pmove_extern_unit(cardID,axis,dis,start,speed,stop,acc,dec,s_para,posi_mode); //执行运动
while (true)
{
    LTDMC.dmc_get_positOn(cardID,axis,ref pos);
    if (pos > dis -50)
    {
        break;
    }
    ApplicatIon.DoEvents();
}
LTDMC.dmc_update_target_positOn_extern_unit(cardID,axis,dis+500,dis+500+softdist,0,0) //强制变位并
且实现软着陆
.....

```

6.21 软启动功能的实现

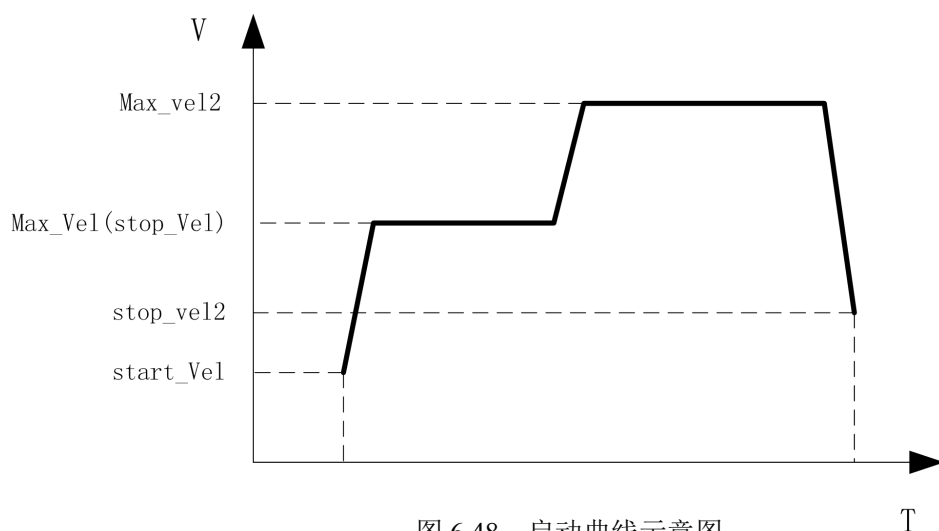


图 6.48 启动曲线示意图

软启动功能主要用来实现启动时以一个较低速度平稳运动，然后快速 pmove 到目标位置，保证启动平稳的同时又不失效率。如下图所示，重点想实现的功能为 pmove 启动时能以一个很低速度平稳运行。将 pmove 分为两段，第一段以 start_Vel、Max_Vel、stop_Vel 进行规划，第二段以最大速度 Max_Vel2 进行规划。常用于晶圆抓取等场合。相关函数如表 6.47 所示

表 6.47 软启动运动功能相关函数说明

名称	功能	参考
dmc_t_pmove_extern_softstart_unit	实现 profile, pmove 整合, 缩短指令时间, 并且实现软着陆	8.4 节

备注：软启动不支持 S 形速度曲线

例 8.47：软启动运动

```

.....
double start,speed_low,stop_mid,delayms,speed_high,stop_end,acc,dec,targetPos,midPos;
ushort cardID,axis,posi_mode;

cardID = 0;
axis = 0; //轴号
targetPos = 1000; //目标位置
midPos = 60; //软启动时低速运动位置
start = 10; //启动速度
speed_low = 30; //第一段低速速度
stop_mid = 30; //停止速度，软着陆低速运动时速度
delayms = 0; //第一段完成后延时时间ms
speed_high = 500; //第二段高速速度
stop_end = 10; //第二段停止速度
acc = 0.1; //加速时间
dec = 0.1; //减速时间
posi_mode = 1; //绝对运动模式
LTDMC.dmc_t_pmove_extern_softstart_unit(cardID,axis,midPos,targetPos,start,speed_low,stop_mid,
delayms,speed_high,stop_end,acc,dec,posi_mode); //执行软启动运动

```

6.22 圆形区域限位功能的实现

圆形区域限位，即运动仅在用户设定的圆形范围内进行，超出这个范围，运动会依据设定的减速模式停止，且禁止任何运动操作。仅支持平面圆形限位，不支持空间圆形限位。

首先，用户需要绑定两个轴进而确定该功能所处的坐标系。根据用户设置的圆心坐标及半径，可以确定该区域在坐标系中的位置。运动过程中，对两轴的每一个位置 x_i 、 y_i 都判定与圆心的距离，若超过半径 R ，则触发减速停或急停。

$$\sqrt{(x_i - x)^2 + (y_i - y)^2} < R$$

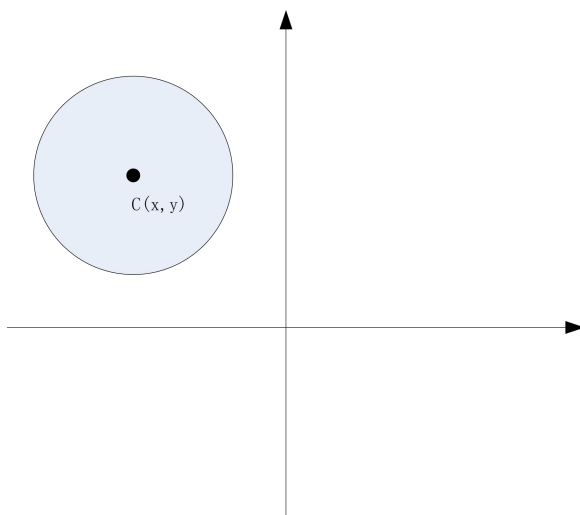


图 6.49 圆弧限位示意图

备 注:

- 1) 由于触发圆形限位运动停止，获取停止原因返回值为 1103;
- 2) 轴在运动中，不可设置圆形限位参数;
- 3) 开启功能后，可通过重复调用 `dmc_set_arc_zone_limit_config` 函数修改参数;
- 4) 除 Vmove 外，其它运动形式若开始运动时在限位区域外，则禁止运动;
- 5) 使用先配置圆形限位参数，再使能功能。

相关函数如表 6.48 所示

表 6.48 圆形区域限位功能相关函数说明

名称	功能	参考
<code>dmc_set_arc_zone_limit_enable</code>	开启/关闭圆形区域限位功能	8.3 节
<code>dmc_get_arc_zone_limit_enable</code>	获取圆弧限位功能的使能状态	
<code>dmc_set_arc_zone_limit_config_unit</code>	配置圆形区域限位参数	
<code>dmc_get_arc_zone_limit_config_unit</code>	回读配置的圆形区域限位参数	
<code>dmc_get_arc_zone_limit_axis_status</code>	查询相应轴的状态	

6.23 椭圆插补及切向跟随的实现

椭圆插补指令实现平面椭圆插补。如下图所示，用户需设置椭圆圆心坐标、a 轴长度、b 轴长度，以确定椭圆在坐标系中数学表达；设置插补目标位置，插补方向，以确定插补动作。

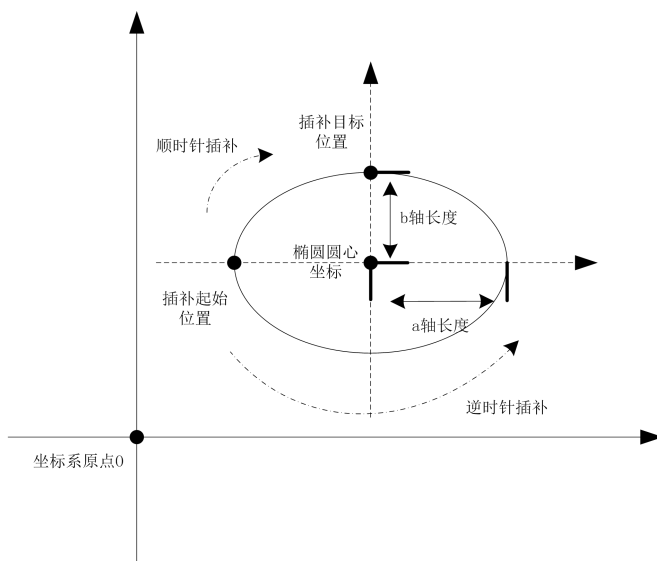


图 6.50 椭圆插补示意图

备 注:

- 1) a 轴定义为平行于 x 轴的椭圆半轴长度, b 轴定义为平行于 y 轴的椭圆半轴长度;
- 2) 插补起始位置为坐标系轴当前位置;
- 3) 目前仅支持标准椭圆, 即长轴和短轴与 XY 平行;
- 4) 运动开始前, 用户应当将旋转轴调整到与插补曲线的起点位置相切的角度;
- 5) 插补运动中, 不能取消跟随运动;
- 6) 插补运动结束后, 跟随关系自动解除, 下次运动时, 需要根据需要重新进行配置切向跟随运动可以配合椭圆插补或圆弧插补运动, 实现跟随轴的旋转运动角度与曲线当前插补位置的切向保持一致。

该功能的使用流程为:

- 1) 调用 `dmc_set_tangent_follow` 指令配置相应坐标系跟随参数;
- 2) 启动相应坐标系的插补运动。

相关函数如表 6.49 所示

表 6.49 椭圆插补及切向跟随功能相关函数说明

名称	功能	参考
<code>dmc_ellipse_move</code>	启动椭圆插补运动	8.6 节
<code>dmc_set_tangent_follow</code>	设置切向跟随参数	
<code>dmc_get_tangent_follow_param</code>	读取切向跟随参数	
<code>dmc_disable_follow_move</code>	取消指定坐标系的跟随运动	

第 7 章 总线操作函数说明

7.1 总线配置函数

本部分函数用于函数实现配置总线通讯（不再需要通过 Motion 配置），调用需按照特定格式，下面只做指令介绍，具体使用可参照例程-配置下载

`short nmc_set_cycletime(WORD CardNo,WORD PortNum,DWORD CycleTime)`

功 能：设置EtherCAT总线循环周期

参 数：CardNo 控制器链接号

 PortNum EtherCAT 端口号，固定为 2

 CycleTime EtherCAT 总线循环周期，单位：us，支持 500/1000/2000

返回值：错误代码

`short dmc_download_memfile(WORD CardNo, char* pBuffer, uint32 buffsize, char* pfilename,WORD filetype)`

功 能：下载总线配置文件到控制器

参 数：CardNo 控制器链接号

 pBuffer 配置文件字符串缓存，UTF8 编码

 buffsize 配置文件字符串大小

 pfilename 文件名（保留参数）

 filetype 文件类型

返回值：错误代码

7.2 总线通讯函数

short nmc_get_cycletime(WORD CardNo,WORD PortNum,DWORD* CycleTime)

功 能：读取EtherCAT总线循环周期

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 CycleTime EtherCAT 总线循环周期，单位：us，支持 500/1000/2000

返回值：错误代码

short nmc_set_node_od(WORD CardNo,WORD PortNum,WORD NodeNum, WORD
Index,WORD SubIndex,WORD ValLength,DWORD Value)

功 能：设置从站对象字典参数值

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 NodeNum 从站 EtherCAT 地址，第 i 个 EtherCAT 从站地址为 1000+i
 Index 对象字典索引
 SubIndex 对象字典子索引
 ValLength 对象字典索引长度(单位：bit)
 Value 对象字典索引参数值

返回值：错误代码

short nmc_get_node_od(WORD CardNo,WORD PortNum,WORD NodeNum, WORD
Index,WORD SubIndex,WORD ValLength,DWORD* Value)

功 能：读取从站对象字典参数值

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 NodeNum 从站 EtherCAT 地址，第 i 个 EtherCAT 从站为 1000+i
 Index 对象字典索引
 SubIndex 对象字典子索引

ValLength 对象字典索引长度(单位: bit)

Value 对象字典索引参数值

返回值: 错误代码

short nmc_get_errcode(WORD CardNo,WORD PortNum,WORD* errcode)

功 能: 读取 EtherCAT 总线状态

参 数: CardNo 控制器链接号

PortNum EtherCAT 总线端口号, 固定为 2

errcode EtherCAT 总线状态, 0 表示正常

返回值: 错误代码

short nmc_clear_errcode(WORD CardNo,WORD PortNum)

功 能: 清除 EtherCAT 总线错误码

参 数: CardNo 控制器链接号

PortNum EtherCAT 总线端口号, 固定为 2

返回值: 错误代码

short nmc_stop_etc(WORD CardNo,WORD* ETCState)

功 能: 停止EtherCAT总线运行

参 数: CardNo 控制器链接号

ETCState 0: 停止 EtherCAT 总线成功, 1: 停止 EtherCAT 总线失败

返回值: 错误代码

short nmc_get_consume_time_fieldbus(WORD CardNo,WORD PortNum,DWORD* Average_time,
DWORD* Max_time,uint64* Cycles)

功 能: 读取EtherCAT总线平均周期时间、最大周期时间、执行周期数

参 数: CardNo 控制器链接号

PortNum EtherCAT 端口号, 固定为 2

Average_time	EtherCAT 总线周期时间，单位：us
Max_time	EtherCAT 总线最大周期时间，单位：us
Cycles	EtherCAT 总线执行周期数

返回值：错误代码

short nmc_clear_consume_time_fieldbus(WORD CardNo,WORD PortNum)

功 能：清除EtherCAT总线平均周期时间、最大周期时间、执行周期数等记录

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2

返回值：错误代码

short nmc_get_total_slaves(WORD CardNo,WORD PortNum,WORD* TotalSlaves)

功 能：获取 EtherCAT 从站总数

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 TotalSlaves EtherCAT 从站总数

返回值：错误代码

short nmc_get_total_adcnum(WORD CardNo,WORD* TotalIn,WORD* TotalOut)

功 能：读取 EtherCAT 总线 AD/DA 输入输出口数

参 数：CardNo 控制器链接号
 TotalIn EtherCAT 总线 AD 输入数
 TotalOut EtherCAT 总线 DA 输出数

返回值：错误代码

short nmc_get_total_IOnum(WORD CardNo,WORD *TotalIn,WORD *TotalOut)

功 能：读取 EtherCAT 总线 IO 输入输出口数

参 数：CardNo 控制器链接号

TotalIn EtherCAT 总线 IO 输入数

TotalOut EtherCAT 总线 IO 输出数

返回值：错误代码

short nmc_get_total_axes(WORD CardNo,DWORD* TotalAxis)

功 能：读取 EtherCAT 总线轴和虚拟轴轴数

参 数：CardNo 控制器链接号

TotalAxis EtherCAT 总线轴和虚拟轴轴数

返回值：错误代码

short nmc_set_controller_workmode(WORD CardNo,WORD controller_mode)

功 能：设置控制器工作模式

参 数：CardNo 控制器链接号

controller_mode 控制器工作模式，0 表示仿真模式，1 表示 EtherCAT

总线模式

返回值：错误代码

short nmc_get_controller_workmode(WORD CardNo,WORD* controller_mode)

功 能：读取控制器工作模式

参 数：CardNo 控制器链接号

controller_mode 控制器的工作模式，0 表示仿真模式，1 表示 EtherCAT 总线模式

返回值：错误代码

short nmc_write_rxpdo_extra(WORD CardNo,WORD PortNum,Word address,Word DataLen,int Value)

功 能：设置从站扩展有符号 RxPDO 值

参 数：CardNo 控制器链接号

PortNum EtherCAT 端口号，固定为 2

address	扩展 PDO 的首地址
DataLen	数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）
Value	数据值

返回值：错误代码

short nmc_read_rxpdo_extra(WORD CardNo,WORD PortNum,Word address,Word DataLen, int * Value)

功 能：读取从站扩展有符号 RxPDO 值

参 数：CardNo 控制器链接号

PortNum EtherCAT 端口号，固定为 2

address 扩展 PDO 的首地址

DataLen 数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）

Value 数据值

返回值：错误代码

short nmc_read_txpdo_extra(WORD CardNo,WORD PortNum,Word address,Word DataLen, int * Value)

功 能：读取从站扩展有符号 TxPDO 值

参 数：CardNo 控制器链接号

PortNum EtherCAT 端口号，固定为 2

address 扩展 PDO 的首地址

DataLen 数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）

Value 数据值

返回值：错误代码

short nmc_write_rxpdo_extra_uint(WORD CardNo,WORD PortNum,Word address,Word DataLen,DWORD Value)

功 能：设置从站扩展无符号 RxPDO 值

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
address	扩展 PDO 的首地址
DataLen	数据长度, 按 16bit 计算, 最大值为 2 (表示 32bit 数据)
Value	数据值

返回值: 错误代码

```
short nmc_read_rxpdo_extra_uint(WORD CardNo,WORD PortNum,Word address,Word DataLen,  
DWORD * Value)
```

功 能: 读取从站扩展无符号 RxPDO 值

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
address	扩展 PDO 的首地址
DataLen	数据长度, 按 16bit 计算, 最大值为 2 (表示 32bit 数据)
Value	数据值

返回值: 错误代码

```
short nmc_read_txpdo_extra_uint(WORD CardNo,WORD PortNum,Word address,Word DataLen,  
DWORD* Value)
```

功 能: 读取从站扩展无符号 TxPDO 值

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
address	扩展 PDO 的首地址
DataLen	数据长度, 按 16bit 计算, 最大值为 2 (表示 32bit 数据)
Value	数据值

返回值: 错误代码

7.3 总线 IO 及轴控制函数

short nmc_set_fieldbus_error_switch(WORD CardNo,WORD PortNum,WORD Data)

功 能：设置总线掉线后是否能操作从站

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 Data 使能开关：1 使能，0 不使能；默认为 0 不使能

返回值：错误代码

备注：不使能：当总线报掉线报错以后，网络所有轴不允许操作，但是 IO 仍然可以操作。避免客户用 IO 操作做电源，驱动器上电等类似操作；使能：总线报掉线报错以后，未掉线之前的驱动器仍然能操作。

short nmc_get_fieldbus_error_switch(WORD CardNo,WORD PortNum,WORD* Data)

功 能：读取设置的总线掉线后是否能操作从站

参 数：CardNo 控制器链接号
 PortNum EtherCAT 端口号，固定为 2
 Data 读取设置的使能状态

返回值：错误代码

short nmc_get_current_fieldbus_state_info(WORD CardNo,WORD Channel,WORD *Axis,WORD *ErrorType,WORD *SlaveAddr,DWORD *ErrorFieldbusCode);

功 能：获取当前的时刻的总线错误码状态信息

参 数：CardNo 控制器链接号
 Channel: 通道号，EtherCAT 为 2 或者 3 号通道
 Axis: 如果当前总线是由驱动器从站引起，该值为轴号，否则为 0xFFFF
 ErrorType: 错误码类型，1 表示主站引起的总线错误，2 表示驱动器引起的从站错误，3 表示无错误，或者错误已纠正；
 SlaveAddr: 如果错误码类型为 2，该值为从站地址，从十进制的 1001 开始。否则为 0xFFFF

ErrorFieldbusCode: 表示实际的错误码值。

返回值: 错误代码

```
short nmc_get_detail_fieldbus_state_info(WORD CardNo,WORD Channel,DWORD  
ReadErrorNum,DWORD *TotalNum, DWORD *ActualNum, WORD *Axis,WORD  
*ErrorType,WORD *SlaveAddr,DWORD *ErrorFieldbusCode)
```

功 能: 获取历史总线错误码状态信息

参 数: CardNo 控制器链接号

Channel: 通道号, EtherCAT 为 2 或者 3 号通道

ReadErrorNum: 设置需要读取的错误码个数; 最大值为 1000

TotalNum: 总线正常运行到执行该指令时的网络错误码的总组数; 每组错误码包括: 轴号, 错误码类型, 从站地址及错误码值;

ActualNum 实际读回来的错误码组数;

Axis: 如果当前总线是由驱动器从站引起, 该值为轴号, 否则为 0xFFFF

ErrorType: 错误码类型, 1 表示主站引起的总线错误, 2 表示驱动器引起的从站错误, 3 表示无错误, 或者错误已纠正;

SlaveAddr: 如果错误码类型为 2, 该值为从站地址, 从十进制的 1001 开始。否则为 0xFFFF

ErrorFieldbusCode: 表示实际的错误码值。

返回值: 错误代码

```
short nmc_torque_move(WORD CardNo,WORD axis,int Torque,WORD PosLimitValid,double  
PosLimitValue,WORD PosMode)
```

功 能: 启动转矩运动

参 数: CardNo 控制器链接号

axis EtherCAT 总线轴轴号

Torque 设置启动转矩值

PosLimitValid 位置限制开关, 0: 限位不起作用, 1: 停止在限位位置, 2: 超出限

位停止

PosLimitValue 0x80: 强制切换成扭力模式

PosMode 位置模式, 0: 相对模式, 1: 绝对模式

返回值: 错误代码

注意: 启用的时候当超过限制位置时会退出转矩控制并且保持限制位置不动; 报错 180: 当启用位置限制的时候会根据转矩正负方向判断 PosLimitValue 与当前轴位置CurrentPos 的关系, Torque>0 时必须确保 PosLimitValue>CurrentPos,Torque<0 时必须确PosLimitValue<CurrentPos

short nmc_change_torque(WORD CardNo,WORD axis,int Torque);

功 能: 在线调整当前轴的转矩值

参 数: CardNo 控制器链接号

axis EtherCAT 总线轴轴号

Torque 在线改变实时转矩值

返回值: 错误代码

注意: 当要使用位置限制功能的时候务必确保新设置的 Torque 转矩值与启动时候的转矩值方向一致, 即要保证运动方向的正确性, 否则将无法到达限制位置。

short nmc_get_torque(WORD CardNo,WORD axis,int* Torque);

功 能: 获取当前轴的转矩值

参 数: CardNo 控制器链接号

axis EtherCAT 总线轴轴号

Torque 当前轴的实时转矩值

返回值: 错误代码

short nmc_set_slave_output_retain(WORD CardNo,WORD Enable)

功 能: 总线复位输出保持开关设置

参 数: CardNo 控制器链接号

Enable 使能配置, 1: 使能, 0: 不使能

返回值：错误代码

注 意：该功能使用需要从站模块支持才可实现复位保持。

```
short nmc_get_slave_output_retain(WORD CardNo,WORD * Enable);
```

功 能：读取总线复位输出保持开关设置

参 数：CardNo 控制器链接号
 Enable 使能配置，1：使能，0：不使能

返回值：错误代码

```
short nmc_set_axis_enable(WORD CardNo,WORD axis)
```

功 能：设置 EtherCAT 总线驱动器使能

参 数：CardNo 控制器链接号
 axis EtherCAT 总线轴的轴号，255 表示使能所有 EtherCAT 轴

返回值：错误代码

```
short nmc_set_axis_disable(WORD CardNo,WORD axis)
```

功 能：设置 EtherCAT 总线驱动器失能

参 数：CardNo 控制器链接号
 axis EtherCAT 总线轴的轴号，255 表示失能所有 EtherCAT 轴

返回值：错误代码

```
short nmc_get_axis_node_address(WORD CardNo,WORD axis , WORD* SlaveAddr,WORD*  
Sub_AlaveAddr)
```

功 能：读取 EtherCAT 轴节点地址信息

参 数：CardNo 控制器链接号
 axis EtherCAT 总线轴号
 SlaveAddr EtherCAT 总线轴地址
 Sub_AlaveAddr EtherCAT 总线轴子地址

返回值：错误代码

short nmc_get_axis_type(WORD CardNo,WORD axis, WORD* Axis_Type)

功 能：读取轴类型

参 数：CardNo 控制器链接号

axis 轴号

Axis_Type 轴的类型，0：虚拟轴，1：EtherCAT 轴，2：CANopen 轴，3：脉冲轴，4：未知类型轴

返回值：错误代码

short nmc_get_axis_state_machine(WORD CardNo,WORD axis, WORD* Axis_StateMachine);

功 能：读取EtherCAT总线轴状态机

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

Axis_StateMachine EtherCAT 总线轴状态机

0：未启动状态(NOT_READY_SWITCH_ON)

1：启动禁止状态(SWITCH_ON_DISABLE)

2：准备启动状态(READY_TO_SWITCH_ON)

3：启动状态(SWITCH_ON)

4：操作使能状态(OP_ENABLE)

5：停止状态(QUICK_STOP)

6：错误触发状态(FAULT_ACTIVE)

7：错误状态(FAULT)

返回值：错误代码

short nmc_get_axis_errcode(WORD CardNo,WORD axis,WORD *Errcode)

功 能：读取总线轴错误码

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

Errcode EtherCAT 总线轴错误码

返回值：错误代码

```
short nmc_clear_axis_errcode(ushort CardNo, ushort axis);
```

功 能：清除总线轴错误码

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

返回值：错误代码

```
DWORD nmc_get_axis_IO_in(WORD CardNo,WORD axis)
```

功 能：读取EtherCAT总线驱动器的数字量输入状态，32bit数据

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

返回值：EtherCAT 总线驱动器数字量输入状态，32bit 数据，bit 位为 1 表示有输入信号。

```
DWORD nmc_get_axis_IO_out(WORD CardNo,WORD axis)
```

功 能：读取EtherCAT总线驱动器的数字量输出状态，32bit数据

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

返回值：EtherCAT 总线驱动器数字量输出状态，32bit 数据，bit 位为 1 代表有输出信号

```
short nmc_set_axis_IO_out(WORD CardNo,WORD axis,DWORD IOstate)
```

功 能：设置EtherCAT总线驱动器的数字量输出状态，32bit数据

参 数：CardNo 控制器链接号

axis EtherCAT 总线轴轴号

IOstate EtherCAT 总线驱动器数字量输出状态，32bit 数据，bit 位为 1 代表

置输出信号

返回值：错误码

```
short nmc_get_axis_setting_ctrlmode(WORD CardNo,WORD axis,long* ctrlmode);
```

功 能：获取EtherCAT总线轴配置控制模式

参 数：CardNo 控制器链接号

 axis EtherCAT 总线轴轴号

 ctrlmode EtherCAT 总线轴配置控制模式（6 回零模式、8 CSP 模式）

返回值：错误码

```
short nmc_get_axis_statusword(WORD CardNo,WORD axis,long* statusword);
```

功 能：获取EtherCAT总线轴状态字

参 数：CardNo 控制器链接号

 axis EtherCAT 总线轴轴号

 statusword EtherCAT 总线轴状态字

返回值：错误码

```
short nmc_get_axis_ctrlword(WORD CardNo,WORD axis,long* Ctrlword);
```

功 能：获取EtherCAT总线轴控制字

参 数：CardNo 控制器链接号

 axis EtherCAT 总线轴轴号

 statusword EtherCAT 总线轴控制字

返回值：错误码

```
short nmc_write_outbit_extern(WORD CardNo,WORD PortNum,WORD NoteID,WORD  
IOBit,WORD IOValue)
```

功 能：设置 EtherCAT 扩展模块的某个输出端口的电平

参 数：CardNo 控制器链接号

 PortNum EtherCAT 端口号，固定为 2

NodeID	扩展模块的节点号,从 1001 开始, 按从站数往后累加
IOBit	输出端口号
IOValue	输出电平, 0: 低电平, 1: 高电平

返回值: 错误代码

short nmc_read_outbit_extern(WORD CardNo,WORD PortNum,WORD NodeID,WORD IOBit,WORD *IOValue)

功 能: 读取 EtherCAT 扩展模块的某个输出端口的电平

参 数: CardNo 控制器链接号

PortNum	EtherCAT 端口号, 固定为 2
NodeID	扩展模块的节点号, 从 1001 开始, 按从站数往后累加
IOBit	输出端口号
IOValue	返回输出电平, 0: 低电平, 1: 高电平

返回值: 错误代码

short nmc_read_inbit_extern(WORD CardNo,WORD PortNum,WORD NodeID,WORD IOBit,DWORD *IOValue)

功 能: 读取 EtherCAT 扩展模块的某个输入端口的电平

参 数: CardNo 控制器链接号

PortNum	EtherCAT 端口号, 固定为 2
NodeID	扩展模块的节点号, 从 1001 开始, 按从站数往后累加
IOBit	输入端口号
IOValue	输入端口电平, 0: 低电平, 1: 高电平

返回值: 错误代码

short nmc_write_outport_extern(WORD CardNo,WORD PortNum,WORD NodeID,WORD PortNo,WORD IOValue)

功 能: 设置 EtherCAT 扩展模块指定 IO 组号的全部输出口的电平

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
NodeID	扩展模块的节点号, 从 1001 开始, 按从站数往后累加
PortNo	IO 组号 (32 位一组, 从 0 开始, 如 0 表示 0-31 位, 1 表示 32-63 位)
IOValue	32 位数值, 输出 bit 位端口电平, 0: 低电平, 1: 高电平

返回值: 错误代码

注 意: IO 组号最小值为 0, 每 32 点 IO 一组; IO 数满 32, 组号加 1

```
short nmc_read_outport_extern(WORD CardNo,WORD PortNum,WORD NodeID,WORD  
PortNo,DWORD *IOValue)
```

功 能: 读取 EtherCAT 扩展模块指定 IO 组号的全部输出口的电平

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
NodeID	扩展模块的节点号, 从 1001 开始, 按从站数往后累加
PortNo	IO 组号 (32 位一组, 从 0 开始, 如 0 表示 0-31 位, 1 表示 32-63 位)
IOValue	32 位数值, 输出 bit 位端口电平, 0: 低电平, 1: 高电平

返回值: 错误代码

注 意: IO 组号最小值为 0, 每 32 点 IO 一组; IO 数满 32, 组号加 1

```
short nmc_read_inport_extern(WORD CardNo,WORD PortNum,WORD NodeID,WORD  
PortNo,DWORD *IOValue)
```

功 能: 读取 EtherCAT 扩展模块指定 IO 组号的全部输入口的电平

参 数: CardNo	控制器链接号
PortNum	EtherCAT 端口号, 固定为 2
NodeID	扩展模块的节点号, 从 1001 开始, 按从站数往后累加
PortNo	IO 组号 (32 位一组, 从 0 开始, 如 0 表示 0-31 位, 1 表示 32-63 位)
IOValue	32 位数值, 输入 bit 位端口电平, 0: 低电平, 1: 高电平

返回值：错误代码

注 意：按 EtherCAT 节点号读取和设置 IO 只适用于操作 EtherCAT IO 扩展模块，不支持控制器自带 IO 以及驱动器本体 IO；

`short nmc_set_offset_pos(WORD CardNo,WORD axis, double offset_pos)`

功 能：设置偏移量的位置值

参 数：	CardNo	控制器链接号
	axis	轴号
	offset_pos	偏移值，偏移值设为零，使用驱动器实际位置

返回值：错误代码

注意：只需初始化控制器后，设置一次。

`short nmc_get_offset_pos(WORD CardNo,WORD axis, double* offset_pos)`

功 能：获取设置的偏移量位置值

参 数：	CardNo	控制器链接号
	axis	轴号
	offset_pos	获取设置的偏移值

返回值：错误代码

7.4 回零运动函数

`short nmc_set_home_profile(WORD CardNo,WORD axis,WORD home_mode,double Low_Vel,double High_Vel,double Tacc,double Tdec,double Offsetpos)`

功 能：设置 EtherCAT 总线轴回零参数

参 数：	CardNo	控制器链接号
	axis	EtherCAT 总线轴轴号
	home_mode	EtherCAT 总线轴回零模式
	Low_Vel	EtherCAT 总线轴回零低速

High_Vel	EtherCAT 总线轴回零高速
Tacc	EtherCAT 总线轴回零加速时间
Tdec	EtherCAT 总线轴回零减速时间
Offsetpos	EtherCAT 总线轴回零偏移

返回值：错误代码

注意：各驱动器回零后对回零偏移量的处理方式不同，请查看各驱动器手册说明。

```
short nmc_get_home_profile(WORD CardNo,WORD axis,WORD* home_mode,double*  
Low_Vel,double* High_Vel,double* Tacc,double* Tdec,double* Offsetpos)
```

功 能：读取 EtherCAT 总线轴回零参数

参 数：CardNo	控制器链接号
axis	EtherCAT 总线轴轴号
home_mode	EtherCAT 总线轴回零模式
Low_Vel	EtherCAT 总线轴回零低速
High_Vel	EtherCAT 总线轴回零高速
Tacc	EtherCAT 总线轴回零加速时间
Tdec	EtherCAT 总线轴回零减速时间
Offsetpos	EtherCAT 总线轴回零偏移

返回值：错误代码

```
short nmc_home_move(WORD CardNo,WORD axis)
```

功 能：启动 EtherCAT 总线轴回零

参 数：CardNo	控制器链接号
axis	EtherCAT 总线轴轴号

返回值：错误代码

```
short dmc_get_home_result(WORD CardNo,WORD axis,WORD* state)
```

功 能：读取回零执行状态

参 数：CardNo	控制器链接号
------------	--------

axis 指定轴号，取值范围：0~31

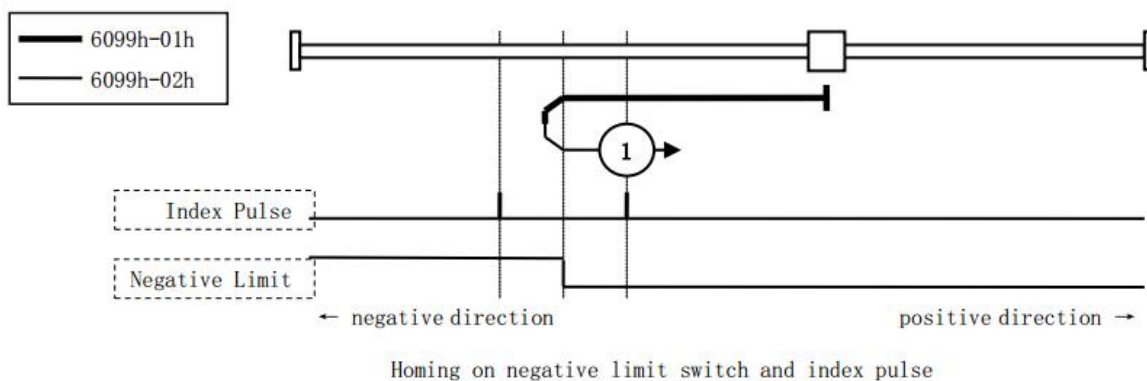
state 回零执行状态，1：回零完成，0：回零未完成

返回值：错误代码

下面以松下 A6B 系列驱动器为例，简单介绍 EtherCAT 总线各回零模式具体实现方式，其他品牌 EtherCAT 总线驱动器回零模式请参照驱动器手册描述。

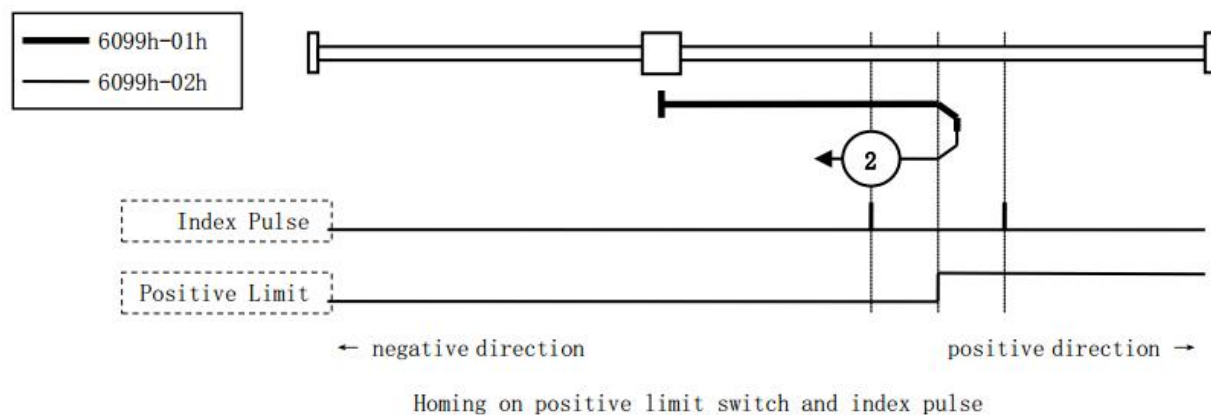
回零模式 1：

- 此方法是，如果未激活负限位开关，初始化动作方向是负方向。(图示为低电平状态下非激活状态)
- 原点检出位置是负限位信号为非激活后的在正方向侧位置的最初的 Index pulse 检出位置。



回零模式 2：

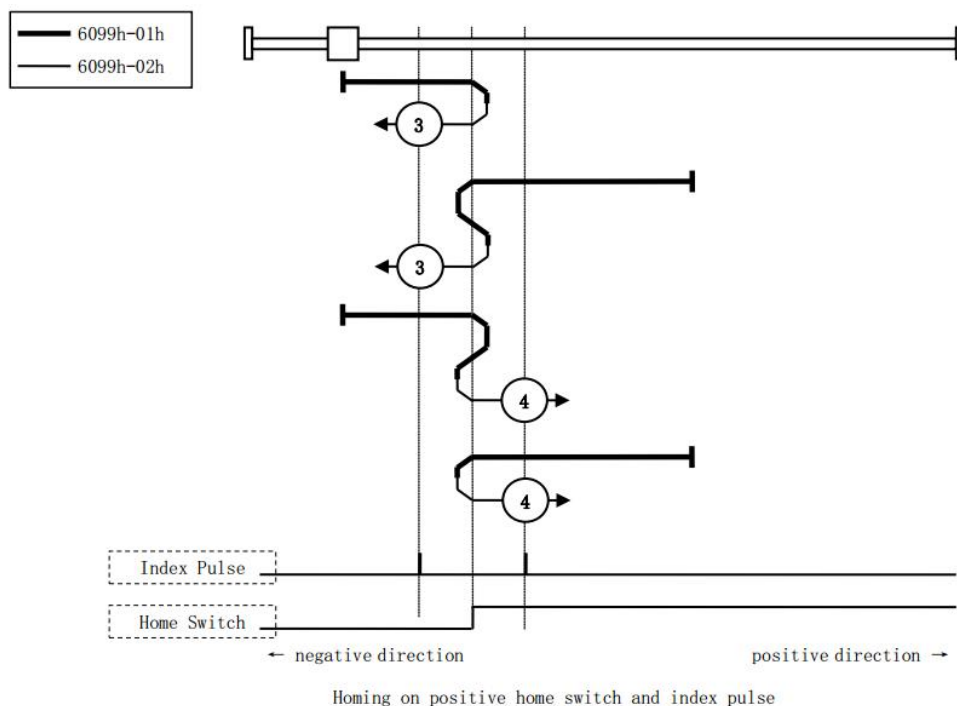
- 此方法是，如果未激活正限位开关，初始化动作方向是正方向。(图示为低电平状态下非激活状态)
- 原点检出位置是正限位信号为非激活后的在负方向侧位置的最初的 Index pulse 检出位置。



回零模式 3、4：

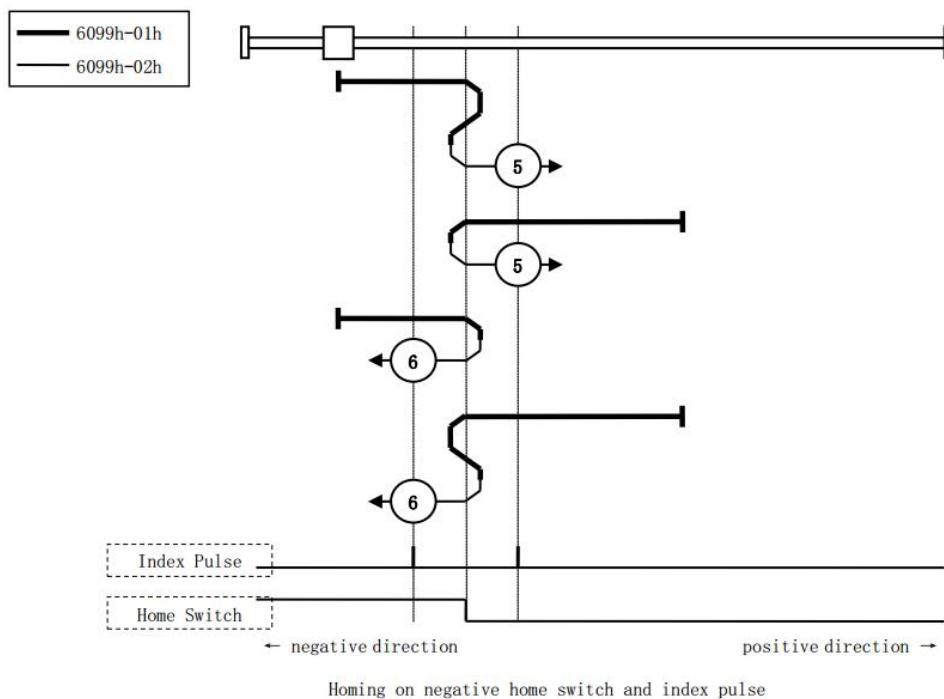
- 此方法是，基于起动时的 Home switch 的状态初始化动作方向变化。

- 原点检出位置是 Home switch 的状态变化后的负方向侧，或者负方向侧最初的 Index pulse 检出位置。



回零模式 5、6:

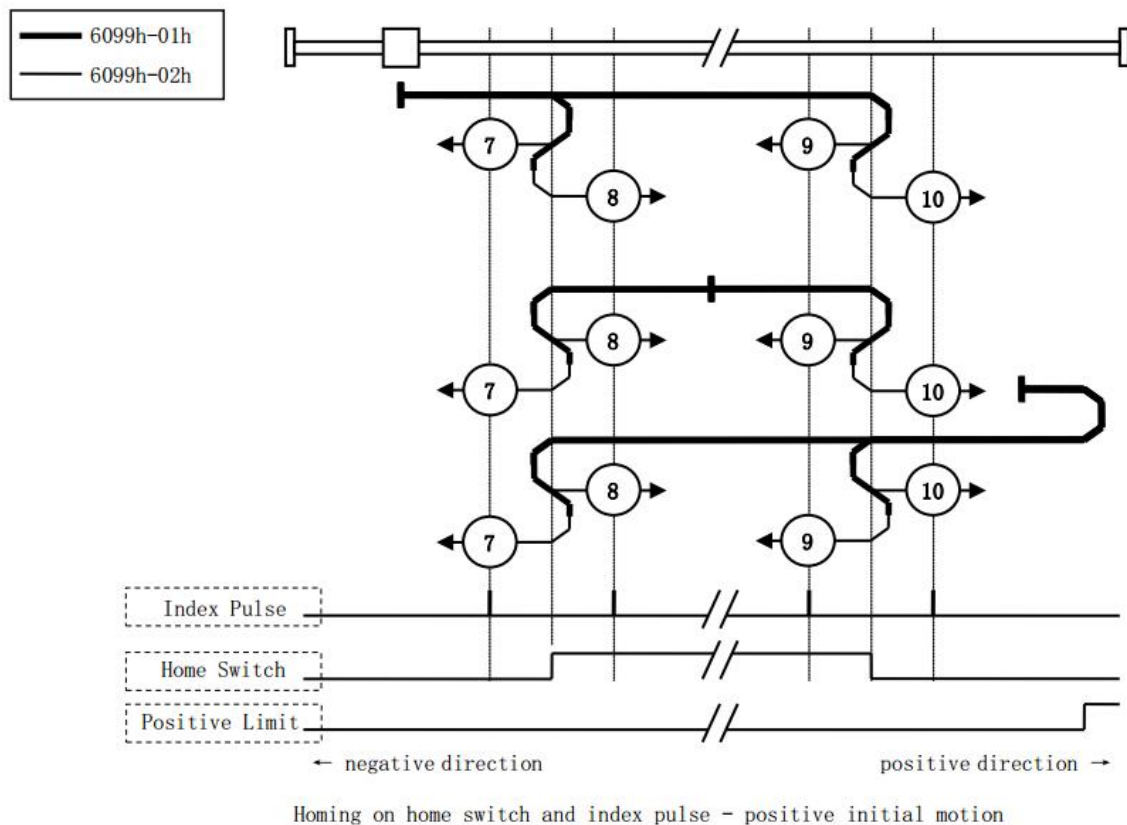
- 此方法是，基于起动时的 Home switch 的状态初始化动作方向变化。
- 原点检出位置是 Home switch 的状态变化后的负方向侧，或者正方向侧最初的 Index pulse 检出位置。



回零模式 7、8、9、10:

- 此方法是，使用 Home switch 和 Index pulse。
- 方法 7,8 的初始动作方向是 Home switch 如果在动作开始时已经激活，则为负方向。
- 方法 9,10 的初始化动作方向是 Home switch 如果在动作开始时已经激活，则为正方向。
- 原点检出位置是， Home switch 的上升沿或者下降沿附近的 Index pulse。

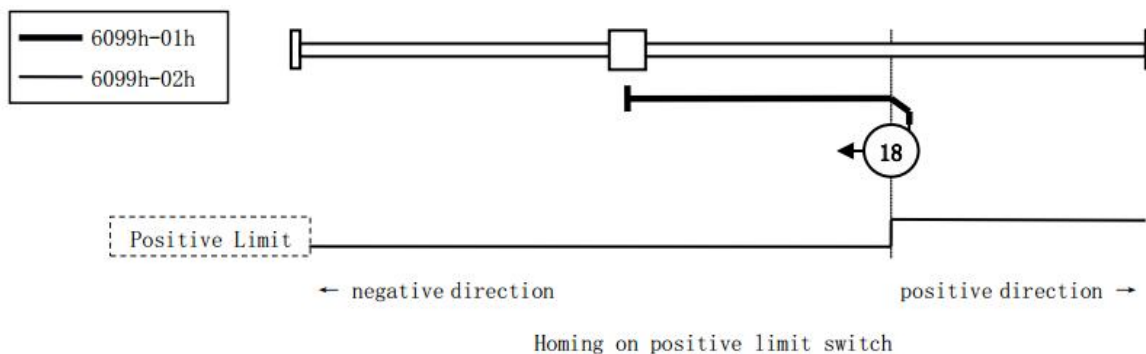
(请参照下图)



回零模式 11, 12, 13, 14

- 此方法是，使用 Home switch 和 Index pulse。
- 方法 11, 12 的初始化动作方向是 Home switch 如果在动作开始时已经激活，则为正方向。
- 方法 13, 14 的初始化动作方向是 Home switch 如果在动作开始时已经激活，则为负方向。
- 原点检出位置是， Home switch 的上升沿或者下降沿附近的 Index pulse。

(请参照下图)

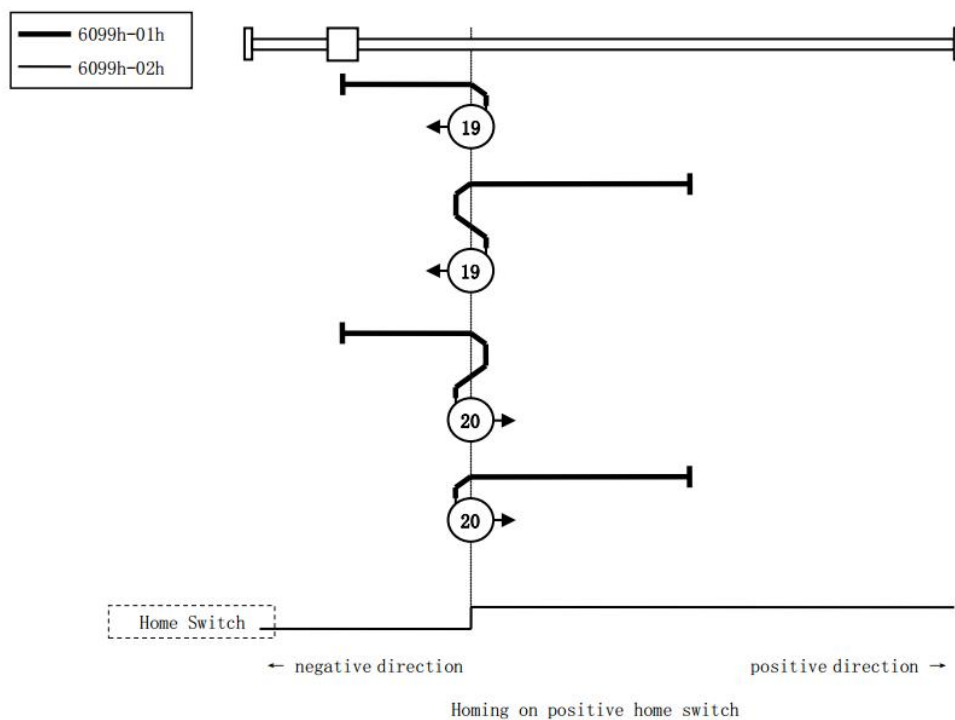


回零模式 19, 20

- 此方法是，类似于回零模式 3, 4。

不同的是，原点检出位置不是 Index pulse，而是 Home switch 变化的位置。

(请参照下图)

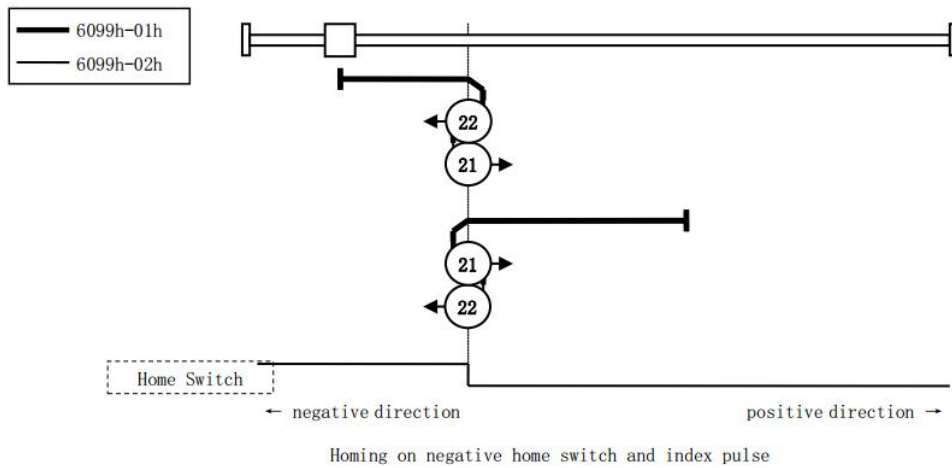


回零模式 21, 22

- 此方法是，类似于回零模式 5, 6。

不同的是，原点检出位置不是 Index pulse，而是 Home switch 变化的位置。

(请参照下图)

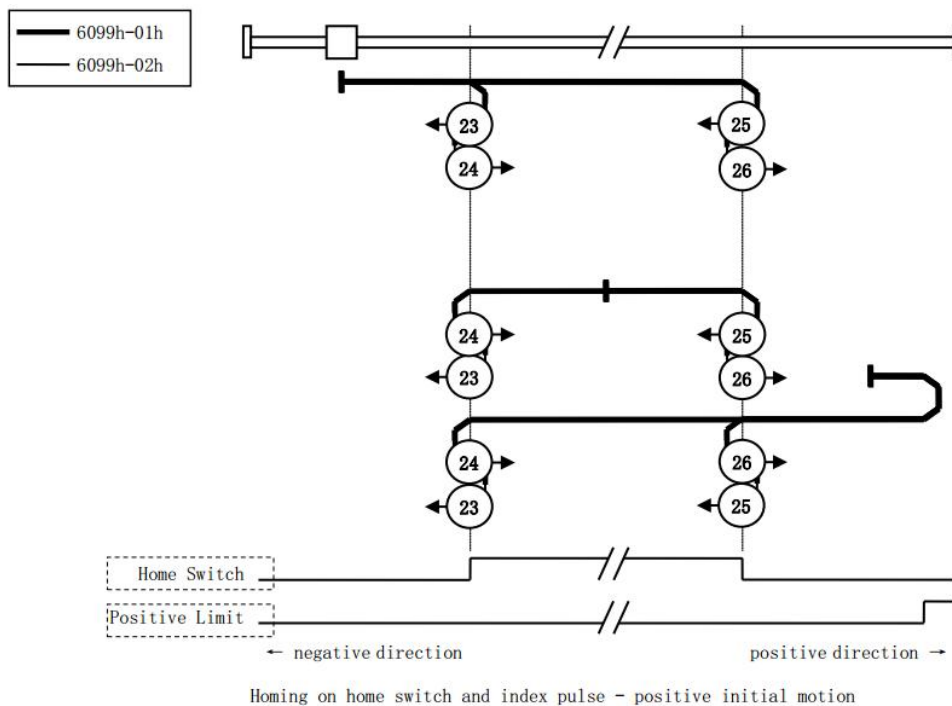


回零模式 23, 24, 25, 26

- 此方法是，类似于回零模式 7, 8, 9, 10。

不同的是，原点检出位置不是 Index pulse，而是 Home switch 变化的位置。

(请参照下图)

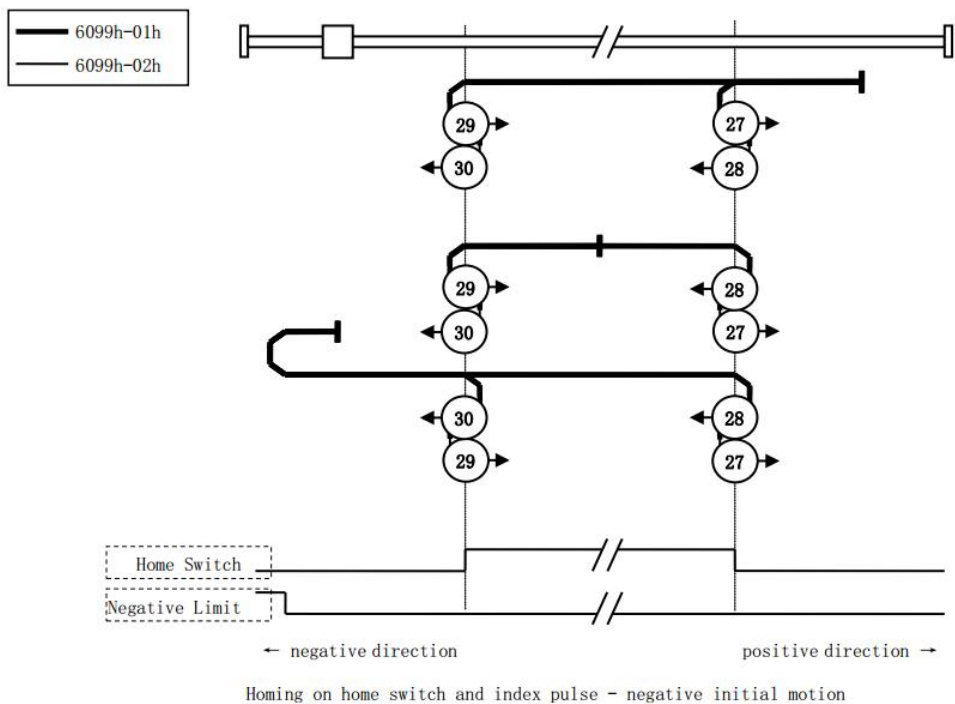


回零模式 27, 28, 29, 30

- 此方法是，类似于回零模式 11, 12, 13, 14。

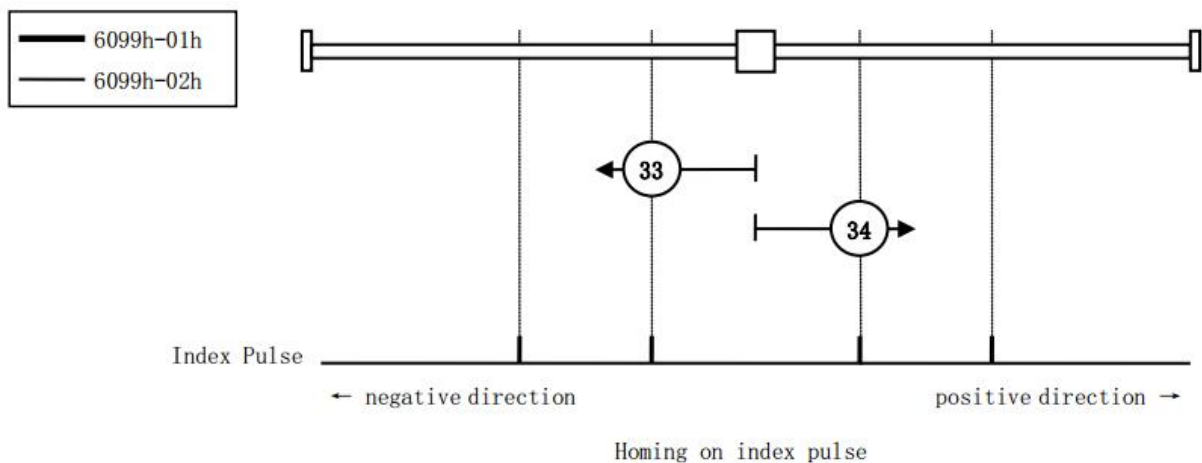
不同的是，原点检出位置不是 Index pulse，而是 Home switch 变化的位置。

(请参照下图)



回零模式 33, 34

- 此方法，仅使用 Index pulse。
- 在图中所示方向动作后检出 Index pulse 作为原点检出位置。



第 8 章 基本功能函数说明

8.1 控制器设置

`short dmc_board_init_eth(WORD CardNo,char* ipaddr)`

功 能：控制器初始化函数，分配系统资源

参 数：CardNo 控制器链接号
 Ipaddr 链接字符串，对应控制器的 IP 地址

返回值：错误代码

`short dmc_cool_reset(WORD CardNo)`

功 能：控制器冷复位函数

参 数：CardNo 控制器链接号

返回值：错误代码

注 意：执行复位操作后，必须关闭控制器，等待 15 秒方可执行初始化控制器，否则会出错。
出错后必须重新执行复位操作，再等待 15 秒后执行初始化控制器。

`short dmc_soft_reset(WORD CardNo)`

功 能：总线控制器热复位函数

参 数：CardNo 控制器链接号

返回值：错误代码

注 意：执行总线热复位操作后，循环调用 `nmc_get_errcode` 函数判断总线状态，如果总线状态返回正常，则可退出循环，总线热复位完成。

`short dmc_original_reset(WORD CardNo)`

功 能：控制器初始值复位函数

参 数：CardNo 控制器链接号

返回值：错误代码

注 意：主要用于删除系统中的所有变量及文件信息。包括：eni 文件，eeprom_config 文件，全局变量文件等。调用该函数只是删除了文件信息，此时需要，再调用一次冷复位或者热复位重启总线才可以完全清除系统的临时缓存信息。执行初始复位操作后，等待 2 秒后再调用冷复位或热复位控制器函数，否则会出错。

short dmc_board_close(void)

功 能：控制器关闭函数，释放系统资源

参 数：无

返回值：错误代码

short dmc_get_CardInfList (WORD* CardNun, DWORD* CardTypeList, WORD* CardIdList)

功 能：获取控制器硬件 ID 号

参 数：CardNun 返回初始化成功的控制器链接号

CardTypeList 返回控制器固件类型数组

CardIdList 返回控制器硬件 ID 号数组，控制器链接号按从小到大顺序排列

返回值：错误代码

注 意：参数 CardTypeList 类型为十六进制

short dmc_get_release_verSIOn(WORD CardNo,char *ReleaseVersIOn)

功 能：获取控制器发布版本号

参 数：CardNo 控制器链接号

ReleaseVersIOn 返回控制器发布版本号

返回值：错误代码

short dmc_get_card_verSIOn(WORD CardNo, DWORD *CardVersIOn)

功 能：获取控制器硬件版本号

参 数：CardNo 控制器链接号

CardVersIOn 返回控制器硬件版本号

返回值：错误代码

short dmc_get_card_soft_verSIOn(WORD CardNo, DWORD *FirmID, DWORD *SubFirmID)

功 能：获取控制器固件版本号

参 数：CardNo 控制器链接号

FirmID 返回控制器固件类型

SubFirmID 返回控制器固件版本号

返回值：错误代码

注 意：参数 FirmID 类型为十六进制

short dmc_get_card_lib_verSIOn(DWORD *LibVer)

功 能：获取控制器动态库文件版本号

参 数：LibVer 返回库版本号

返回值：错误代码

short dmc_get_total_IOnum(WORD CardNo,WORD *TotalIn,WORD *TotalOut)

功 能：获取控制器本地 IO 输入输出口数

参 数：CardNo 控制器链接号

 TotalIn 控制器本地 IO 输入数

 TotalOut 控制器本地 IO 输出数

返回值：错误代码

short dmc_download_configfile(WORD CardNo, const char *FileName)

功 能：下载参数文件

参 数：CardNo 控制器链接号

 FileName 文件路径：

 参数文件名+后缀：相对路径

 完整描述参数文件的路径+文件名后缀：绝对路径

返回值：错误代码

注 意：1) 当使用相对路径时，参数文件与程序必须在同一目录下

2) 可以在控制器 Motion 软件中“参数设置”界面下，将各轴参数设置好，然后点击“下载”将参数文件下载。

short dmc_download_firmware(WORD CardNo, const char *FileName)

功 能：下载固件文件

参 数：CardNo 控制器链接号

 FileName 文件路径：

 参数文件名+后缀：相对路径

 完整描述参数文件的路径+文件名后缀：绝对路径

返回值：错误代码

注 意：1) 当使用相对路径时，固件文件与程序必须在同一目录下

2) 可以在控制器 Motion 软件中“高级功能”->“固件升级”菜单下直接升级固件。

short dmc_get_progress(WORD CardNo,float* process)

功 能：获取下载文件进度

参 数: CardNo 控制器链接号
 Process 下载文件进度（下载过程值范围 0~1，下载完成瞬间置 0）
返回值: 错误代码

8.2 脉冲当量与限位设置

short dmc_set_equiv(WORD CardNo,WORD axis, double equiv)

功 能: 设置指定轴的脉冲当量

参 数: CardNo 控制器链接号
 axis 指定轴号
 equiv 脉冲当量，单位: pulse/unit

返回值: 错误代码

注 意: 1) 轴脉冲当量允许在总线轴使能状态下设置，但需一个总线周期时间生效；
 2) 该函数适用于基于脉冲当量的高级运动函数（包括点位、插补等运动）；
 3) 当使用基于脉冲当量的高级运动函数进行运动前，必须先使用该函数设置各运动轴的脉冲当量值，该值不能设置为 0；

short dmc_get_equiv(WORD CardNo,WORD axis, double *equiv)

功 能: 读取指定轴的脉冲当量设置

参 数: CardNo 控制器链接号
 axis 指定轴号
 equiv 返回脉冲当量

返回值: 错误代码

short dmc_set_softlimit_unit(WORD CardNo, WORD axis, WORD enable, WORD source_sel, WORD SL_actIOOn, double N_limit, double P_limit)

功 能: 设置单轴软限位

参 数: CardNo 控制器链接号
 axis 指定轴号
 enable 使能状态，0: 禁止，1: 允许
 source_sel 计数器选择，0: 指令位置计数器，1: 编码器计数器
 SL_actIOOn 限位停止方式，0: 立即停止 1: 减速停止
 N_limit 负限位位置，单位: unit
 P_limit 正限位位置，单位: unit

返回值：错误代码

注 意：正、负限位位置可为正数也可为负数，但正限位位置应大于负限位位置

```
short dmc_get_softlimit_unit(WORD CardNo, WORD axis, WORD* enable, WORD* source_sel,  
WORD* SL_actIOOn, double* N_limit, double* P_limit)
```

功能：读取单轴软限位设置

参 数：CardNo	控制器链接号
axis	指定轴号
enable	返回使能状态
source_sel	返回计数器选择
SL_actIOOn	返回限位停止方式
N_limit	返回负限位脉冲数
P_limit	返回正限位脉冲数

返回值：错误代码

```
short dmc_set_arc_zone_limit_enable(WORD CardNo, WORD enable);
```

功 能：配置圆弧限位功能使能状态

参 数：CardNo	控制器链接号
enable:	使能状态，0 未使能；1 使能

返回值：错误代码

注意：圆型限位目前只支持一组限位器，多次配置圆形限位则以最后一次配置为准，覆盖之前的配置； 圆形限位使用步骤：先配置圆形限位参数，再使能功能。

```
short dmc_get_arc_zone_limit_enable(WORD CardNo, WORD* enable);
```

功 能：读取圆弧限位功能使能状态

参 数：CardNo	控制器链接号
enable:	使能状态，0 未使能；1 使能

返回值：错误代码

```
short dmc_set_arc_zone_limit_config_unit(WORD CardNo, WORD* AxisList, WORD AxisNum,  
double *Center, double Radius, WORD Source, WORD StopMode);
```

功 能：设置圆弧区域限位配置信息

参 数：CardNo	控制器链接号
AxisList:	需要限位的轴列表，0-实际轴数-1

AxisNum: 需要限位的轴个数, 固定为 2
Center: 区域限位圆心, unit 单位
Radius: 区域限位半径, unit 单位
Source: 计数器选择, 0 指令位置, 1 反馈位置
StopMode: 限位触发后停止模式, 0 减速停止; 1 急停;

返回值: 错误代码

```
short dmc_get_arc_zone_limit_config_unit(WORD CardNo, WORD* AxisList, WORD* AxisNum,  
double *Center, double * Radius, WORD* Source, WORD* StopMode);
```

功 能: 读取圆弧区域限位配置信息

参 数: CardNo 控制器链接号
AxisList: 需要限位的轴列表, 0-实际轴数-1
AxisNum: 需要限位的轴个数, 固定为 2
Center: 区域限位圆心, unit 单位
Radius: 区域限位半径, unit 单位
Source: 计数器选择, 0 指令位置, 1 反馈位置
StopMode: 限位触发后停止模式, 0 减速停止; 1 急停;

返回值: 错误代码

```
short dmc_get_arc_zone_limit_axis_status(WORD CardNo, WORD AxisNo);
```

功 能: 查询相应轴的状态

参 数: CardNo 控制器链接号
AxisNo: 轴号, 0-实际轴数-1

返回值: -1 该轴没有被设置区域限位; 0 该轴在区域内; 1 该轴触发区域限位

8.3 单轴运动速度曲线设置

```
short dmc_set_profile_unit(WORD CardNo, WORD axis, double Min_Vel, double Max_Vel, double  
Tacc, double Tdec, double Stop_Vel)
```

功 能: 设置单轴运动速度曲线

参 数: CardNo 控制器链接号
axis 指定轴号
Min_Vel 起始速度, 单位: unit/s
Max_Vel 最大速度, 单位: unit/s

Tacc	加速时间，单位：s
Tdec	减速时间，单位：s
Stop_Vel	停止速度，单位：unit/s

返回值：错误代码

short dmc_get_profile_unit(WORD CardNo, WORD axis, double* Min_Vel, double* Max_Vel, double* Tacc, double* Tdec, double* Stop_Vel)

功 能：读取单轴运动速度曲线

参 数：CardNo	控制器链接号
axis	指定轴号
Min_Vel	返回起始速度设置
Max_Vel	返回最大速度设置
Tacc	返回加速时间设置
Tdec	返回减速时间设置
Stop_Vel	返回停止速度设置

返回值：错误代码

注：该函数不适用于连续插补

short dmc_set_s_profile(WORD CardNo, WORD axis, WORD s_mode, double s_para)

功 能：设置单轴速度曲线 S 段参数值

参 数：CardNo	控制器链接号
axis	指定轴号
s_mode	保留参数，固定值为 0
s_para	S 段时间，单位：s；范围：0~1

返回值：错误代码

short dmc_get_s_profile(WORD CardNo, WORD axis, WORD s_mode, double *s_para)

功 能：读取单轴速度曲线 S 段参数值

参 数：CardNo	控制器链接号
axis	指定轴号
s_mode	保留参数
s_para	返回设置的 S 段时间

返回值：错误代码

short dmc_set_profile_extern (WORD CardNo, WORD axis, double Min_Vel, double Max_Vel,

double Acc, double Dec, double Ajerk, double Djerk, double stop_vel)

功 能：设置单轴运动速度曲线，加加速度接口

参 数：CardNo 控制器链接号
 axis 指定轴号
 Min_Vel 起始速度，单位：unit/s
 Max_Vel 最大速度，单位：unit/s
 Acc 加速时间，单位：s²
 Dec 减速时间，单位：s²
 Ajerk 加加速度，单位：unit / (s³)
 Djerk 减减速度，单位：unit / (s³)
 Stop_Vel 停止速度，单位：unit/s

返回值：错误代码

8.4 单轴运动

short dmc_pmove_unit(WORD CardNo, WORD axis, double Dist, WORD posi_mode)

功 能：定长运动

参 数：CardNo 控制器链接号
 axis 指定轴号
 Dist 目标位置，单位：unit
 posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

short dmc_vmove(WORD CardNo, WORD axis, WORD dir)

功 能：指定轴连续运动

参 数：CardNo 控制器链接号
 axis 指定轴号
 dir 运动方向，0：负方向，1：正方向

返回值：错误代码

short dmc_reset_target_positIon_unit(WORD CardNo, WORD axis, double New_Pos)

功 能：在线改变指定轴的当前目标位置

参 数：CardNo 控制器链接号
 axis 指定轴号

New_Pos 新目标位置，单位：unit

返回值：错误代码

注 意：1) 该函数只适用于点位运动中的变位
2) 参数 New_Pos 为绝对位置值，无论当前的运动模式为绝对坐标还是相对坐标模式

short dmc_update_target_positIon_unit(WORD CardNo, WORD axis, double New_Pos)

功 能：强行改变指定轴的当前目标位置（在线/非在线）

参 数：CardNo 控制器链接号

axis 指定轴号

New_Pos 新目标位置，单位：unit

返回值：错误代码

注 意：1) 该函数适用于指定轴停止状态或点位运动中的变位
2) 参数 New_Pos 为绝对位置值，无论当前的运动模式为绝对坐标还是相对坐标模式

short dmc_change_speed_unit(WORD CardNo, WORD axis, double New_Vel, double Taccdec)

功 能：在线改变指定轴的当前运动速度

参 数：CardNo 控制器链接号

axis 指定轴号

New_Vel 新的运行速度，单位：unit/s

Taccdec 变速时间，单位：s

返回值：错误代码

注 意：1) 该函数适用于单轴运动中的变速
2) 设置的变速时间是从当前速度变速到新速度的时间。此时控制器会重新计算起始速度加速到最高速度所需的时间以及最高速度减速到停止速度所需的时间，即加速时间会被重新计算
3) 变速一旦成立，该轴的默认运行速度将会被改写为 New_Vel，加减速时间也会被控制器新计算的值所覆盖，也即当调用 dmc_get_profile_unit 回读速度参数时会发生与 dmc_set_profile_unit 所设置的值不一致的现象
4) 在连续运动中 New_Vel 负值表示往负向变速，正值表示往正向变速。在点位运动中 New_Vel 只允许正值

short dmc_pmove_change_pos_speed_enable(WORD CardNo,WORD axis, WORD enable)

功 能：IO 触发在线轴变速变位置使能

参 数：CardNo 控制器链接号

axis	轴号
enable	使能 0：功能关闭；1：多次触发，2：单次触发

返回值：错误代码

注 意：单次触发指变速变位完成后才响应下一个的 IO 信号，触发变速变位动作，变速变位期间不响应；多次触发指变速变位运动过程中也能响应下一个 IO 信号，触发变速变位运动。

short dmc_get_pmove_change_pos_speed_enable(WORD CardNo,WORD axis, WORD* enable)

功 能：读取 IO 触发在线变速变位置使能

参 数：CardNo 控制器链接号

axis	轴号
enable:	使能 0：功能关闭；1：多次触发，2：单次触发

返回值：错误代码

short dmc_pmove_change_pos_speed_config(WORD CardNo,WORD axis,double tar_vel, double tar_rel_pos, WORD trig_mode, WORD source)

功 能：高速 IO 触发在线变速变位置参数配置

参 数：CardNo 控制器链接号

axis	轴号
tar_vel	改变后的运动速度，仅允许正值，单位：pulse/s，0-8M
tar_rel_pos	相对 IO 触发处的位移量，单位：pulse，相对 IO 锁存位置，相对模式
trig_mode	触发方式 0：下降沿触发，1：上升沿触发
source	位置锁存源 0 指令位置，固定为指令位置

返回值：错误代码

注意：变速变位可同时执行，速度有变化则执行变速，位置有变速则执行变位；只适用于 pmove 运动；

short dmc_get_pmove_change_pos_speed_config(WORD CardNo,WORD axis,double* tar_vel, double* tar_rel_pos, WORD* trig_mode, WORD* source)

功 能：读取高速 IO 触发在线变速变位置参数配置

参 数：CardNo 控制器链接号

axis	轴号
tar_vel	改变后的运动速度，仅允许正值，单位：pulse/s，0-8M
tar_rel_pos	相对 IO 触发处的位移量，单位：pulse，相对 IO 锁存位置，相对模式
trig_mode	触发方式 0：下降沿触发，1：上升沿触发

source 位置锁存源 0 指令位置，1 编码器位置，固定为指令位置

返回值：错误代码

short dmc_pmove_change_pos_speed_inbit(WORD CardNo,WORD axis, WORD inbit, WORD enable)

功 能：高速 IO 触发在线变速变位置 IO 配置

参 数：CardNo 控制器链接号

axis 轴号 0-63

inbit 输入端口号，0-实际输入数-1

enable IO 配置使能，配置使能，0 未使能（未使能，脉冲控制器输入端口默认为对应轴的 ORG 信号），1 使能

返回值：错误代码

short dmc_get_pmove_change_pos_speed_inbit(WORD CardNo,WORD axis, WORD *inbit, WORD *enable)

功 能：读取高速 IO 触发在线变速变位置 IO 配置

参 数：CardNo 控制器链接号

axis 轴号 0-63

inbit 输入端口号，0-实际输入数-1

enable IO 配置使能，配置使能，0 未使能（未使能，脉冲控制器输入端口默认为对应轴的 ORG 信号），1 使能

返回值：错误代码

short dmc_get_pmove_change_pos_speed_state(WORD CardNo,WORD axis,WORD *trig_num, double *trig_pos)

功 能：读取高速 IO 触发在线变速变位置状态

参 数：CardNo 控制器链接号

axis 轴号 0-63

trig_num 触发次数

trig_pos 触发位置

返回值：错误代码

注意：调用 pmove 的时候会自动将触发次数和触发位置清 0；

short dmc_set_IO_exactstop (UInt16 CardNo, UInt16 axis, UInt16 IONum , UInt16[] IOList,

UInt16 enalbe, UInt16 valid_logic, UInt16 actIOn, UInt16 move_dir)

功 能：配置减速停止相关参数

参 数：CardNo 控制器链接号
axis 轴号
IONum: 输入数量,1-16
IOList: 输入端口号数组，最大 16 个输入
enable: IO 逻辑减速停止使能, ‘1’:打开此功能,‘0’: 关闭此功能
valid_logic: 与/或之后的有效触发逻辑, 0: 下降沿触发, 1: 上升沿触发
actIOn: ‘0’: 逻辑与,‘1’: 逻辑或
move_dir: ‘0’正方向停止,‘1’: 负方向停止

返回值：错误代码

说 明：配置减速停止相关参数。其中 move_dir 表示检测方向，如果设置为 0，只有速度方向为正向运动时候，减速停止才会起作用；

dmc_set_dec_stop_dist_unit(UInt16 CardID, UInt16 axis,double dis);

功 能：设置 IO 触发减速停止距离

参 数：CardNo 控制器链接号
axis 轴号
Dis 减速停止距离

返回值：错误代码

dmc_get_dec_stop_dist_unit (UInt16 CardID, UInt16 axis, double dis);

功 能：读取 IO 触发减速停止距离

参 数：CardNo 控制器链接号
axis 轴号
Dis 减速停止距离

返回值：错误代码

short dmc_t_pmove_extern_unit (UInt16 CardNo, UInt16 axis, double MidPos, double TargetPos, double Min_Vel, double Max_Vel, double stop_Vel, double acc, double dec, UInt16 posi_mode)

功 能：单轴软着陆功能

参 数：CardNo: 控制器链接号
Axis: 指定轴号, 0-实际轴数-1
MidPos: 第一段 pmove 终点位置

TargetPos:	第二段的终点位置
Min_Vel:	起始速度 (Vs)
Max_Vel:	最大速度 (Vm)
stop_Vel:	停止速度 (Ve)
acc:	加速时间 0.001-2~31s
dec:	减速时间 0.001-2~31s
posi_mode:	相对模式 0 / 绝对模式 1

返回值: 错误代码

short dmc_pmove_extern_unit (WORD CardNo,WORD axis, double dist, double Min_Vel, double Max_Vel, double Tacc, double Tdec, double stop_Vel, double s_para, WORD posi_mode)

功 能: 实现 profile, pmove 整合, 缩短指令时间, 适用于高速场合

参 数: CardNo 指定控制器链接号
axis 指定轴号, 0-实际轴数-1
dist 终点位置,单位: unit
Min_Vel 起始速度
Max_Vel 最大速度
Tacc 加速时间 0.001-2~31s
Tdec 减速时间 0.001-2~31s
stop_Vel 停止速度
s_para 平滑时间, 单位: s, 范围[0~0.5]
posi_mode 运动模式, 0: 相对模式, 1: 绝对模式

返回值: 错误代码

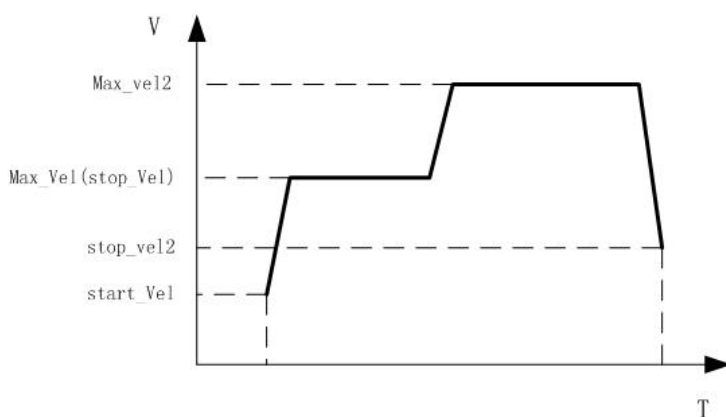
short dmc_t_pmove_extern_softstart_unit (UInt16 CardNo, UInt16 axis, double MidPos, double TargetPos, double start_Vel, double Max_Vel, double stop_Vel, UInt32 delay_ms, double Max_vel2,double stop_vel2, double acc_time, double dec_time, UInt16 posi_mode);

功 能: 单轴软启动

参 数: CardNo: 控制器链接号
axis: 指定轴号, 0-实际轴数-1
MidPos: 第一段 pmove 终点位置
TargetPos: 第二段 pmove 终点位置
start_Vel: 第一段 pmove 起始速度
Max_Vel: 第一段 pmove 最大速度

stop_Vel: 第一段 pmove 停止速度
 delay_ms: 第一阶段完成后延迟时间（单位：毫秒）0-100s
 Max_vel2: 第二段 pmove 最大速度
 stop_vel2: 第二段 pmove 停止速度
 acc_time: 加速时间(单位：秒)0.001s-100s
 dec_time: 减速时间(单位：秒) 0.001s-100s
 posi_mode: 相对模式：0/绝对模式：1

返回值：错误代码



软启动速度曲线示意

short dmc_update_target_positIOOn_extern_unit(WORD cardId,WORD axis, double mid_pos, double aim_pos,double vel,WORD posi_mode)

功 能：强行改变指定轴的当前目标位置并且实现软着陆

参 数：CardNo: 控制器链接号
 axis 指定轴号，0-实际轴数-1
 mid_pos 第一段的终点位置，单位：unit
 aim_pos 第二段的终点位置，单位：unit
 vel: 保留参数，固定值为 0
 posi_mode 保留参数，固定值为 0

返回值：错误代码

注 意：

- 1) 该函数适用于指定轴停止状态或点位运动中的变位，并且实现软着陆。
- 2) 参数 mid_pos, aim_pos 为绝对坐标位置值，无论当前的运动模式为绝对坐标还是相对坐标模式。mid_pos 以速度规划指令中的最大速度运行，aim_pos 以速度规划指令中的停止速度运行。

8.5 插补速度设置

short dmc_set_vector_profile_unit(WORD CardNo,WORD Crd,double Min_Vel,double Max_Vel,
double Tacc, double Tdec, double Stop_Vel)

功 能：设置插补运动速度曲线

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 Min_Vel 最小速度，单位：unit/s
 Max_Vel 最大速度，单位：unit/s
 Tacc 加速时间，单位：s
 Tdec 减速时间，单位：s
 Stop_Vel 停止
 速度，单位：unit/s

返回值：错误代码

注 意：1) EMC-E5000 控制器支持 4 个坐标系（参数 Crd）。四个坐标系的速度可独立设置，
 执行插补时四个坐标系可独立进行插补运动（即可同时进行四组插补运动）

short dmc_get_vector_profile_unit(WORD CardNo,WORD Crd,double* Min_Vel, double*
Max_Vel, double* Tacc, double* Tdec, double* Stop_Vel)

功 能：读取插补运动速度曲线

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 Min_Vel 返回最小速度设置
 Max_Vel 返回最大速度设置
 Tacc 返回加速时间设置
 Tdec 返回减速时间设置
 Stop_Vel 返回停止速度

返回值：错误代码

short dmc_set_vector_s_profile(WORD CardNo, WORD Crd, WORD s_mode, double s_para)

功 能：设置插补运动速度曲线的平滑时间

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 s_mode 保留参数，固定值为 0

s_para 平滑时间，单位：s，范围：0~1

返回值：错误代码

short dmc_get_vector_s_profile(WORD CardNo, WORD Crd, WORD s_mode, double *s_para)

功 能：读取设置的插补运动速度曲线平滑时间

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
s_mode 保留参数，固定值为 0
s_para 返回平滑时间设置

返回值：错误代码

8.6 插补运动

short dmc_line_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double* Target_Pos, WORD posi_mode)

功 能：直线插补运动

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
AxisNum 轴数，取值范围：2~16
AxisList 轴号列表
Target_Pos 目标位置列表，单位：unit
posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

short dmc_arc_move_center_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double *Cen_Pos, WORD Arc_Dir, long Circle, WORD posi_mode)

功 能：基于圆心圆弧扩展的螺旋线插补运动（可作两轴圆弧插补）

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
AxisNum 轴数，取值范围： 2~16
AxisList 轴号列表
Target_Pos 目标位置数组，单位：unit
Cen_Pos 圆心位置数组，单位：unit
Arc_Dir 圆弧方向，0：顺时针，1：逆时针

Circle

圈数:

自然数: 表示此时执行的为螺旋线插补

该值表示螺旋线的圈数。如, 0 即表示 0 圈螺旋线插补, 1 表示 1 圈螺旋线插补...

posi_mode

运动模式, 0: 相对坐标模式, 1: 绝对坐标模式

返回值: 错误代码

注 意: 1) 当轴数为 2 时, 轴列表前两轴进行平面螺旋

2) 当轴数为 3、运动轨迹为螺旋插补时, 轴列表前两轴平面为基面, 进行平面螺旋插补; 同时, 轴列表第三轴运动指定高度, 该轴终点位置与该轴起点位置的差值为螺旋线段相对于基面的高度

3) 当轴数大于 3、运动轨迹为螺旋插补时, 列表前三轴进行螺旋插补的同时, 后续轴做线性跟随运动, 运动时间与前三轴的运动时间相等

4) 当运动轨迹为螺旋插补时:

轴列表前两轴组成的基面上, 当起始点到圆心的距离小于终点到圆心的距离, 为绽放螺旋线

轴列表前两轴组成的基面上, 当起始点到圆心的距离大于终点到圆心的距离, 为收敛螺旋线

轴列表前两轴组成的基面上, 当起始点到圆心的距离等于终点到圆心的距离, 为圆弧插补 (插补轴数为 3 时则为圆柱螺旋线)

short dmc_arc_move_radius_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double Arc_Radius, WORD Arc_Dir, long Circle, WORD posi_mode)

功 能: 基于半径圆弧扩展的圆柱螺旋线插补运动 (可作两轴圆弧插补)

参 数: CardNo 控制器链接号

Crd 坐标系号, 取值范围: 0~3

AxisNum 轴数

AxisList 轴号列表

Target_Pos 目标位置数组, 单位: unit

Arc_Radius 圆弧半径值 (负值为优弧, 正值为劣弧), 单位: unit

Arc_Dir 圆弧方向, 0: 顺时针, 1: 逆时针

Circle 圈数, 取值范围: 大于等于 0

该值表示螺旋线的圈数。如, 0 即表示 0 圈螺旋线插补, 1 即表示 1 圈螺旋线插补...

posi_mode 运动模式, 0: 相对坐标模式, 1: 绝对坐标模式

返回值：错误代码

注 意：1) 当轴数为 2 时，轴列表前两轴进行平面圆弧插补

2) 当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

3) 当轴数大于 3 时，轴列表前三轴进行圆柱螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等

`short dmc_arc_move_3points_unit(WORD CardNo,WORD Crd,WORD AxisNum,WORD* AxisList,
double *Target_Pos, double *Mid_Pos, long Circle, WORD posi_mode)`

功 能：基于三点圆弧扩展的圆柱螺旋线插补运动（可作两轴及三轴圆弧插补）

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

AxisNum 轴数，取值范围：2~16

AxisList 轴号列表

Target_Pos 目标位置数组，单位：unit

Mid_Pos 中间位置数组，单位：unit

Circle 圈数：

负数：表示此时执行的为空间圆弧插补

该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧...

自然数：表示此时执行的为圆柱螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补...

posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

注 意：1) 当轴数为 2 时，轴列表前两轴进行平面圆弧插补

2) 当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

3) 当轴数大于 3、运动轨迹为螺旋插补时，列表前三轴进行螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等

`short dmc_rectangle_move_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD *AxisList,`

double *Target_Pos, double *Mark_Pos, long Count, WORD rect_mode, WORD posi_mode)

功 能：矩形插补运动

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 AxisNum 轴数，**保留参数**，固定值为 2
 AxisList 轴号列表
 Target_Pos 对角位置数组，单位：unit
 Mark_Pos 矩形方向标记位置数组，单位：unit
 Count 行数/圈数
 rect_mode 矩形插补模式，0：逐行，1：渐开线
 posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

注 意：该指令只支持 2 轴矩形插补运动。矩形插补运动支持前瞻模式和非前瞻模式(dmc_conti_set_lookahead_mode)，前瞻模式矩形拐角会平滑处理变成圆弧，无法到达矩形端点；非前瞻模式则精确到达矩形端点。

short dmc_axis_follow_line_enable(WORD CardNo,WORD Crd,WORD enable_flag)

功 能：直线插补参与插补轴数设置，默认情况下为所有轴都参与直线插补

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 enable_flag 使能轴数值，
 0：最多只有 3 个轴参与直线插补，其它轴跟随运动
 1：坐标系所有轴都参与直线插补

返回值：错误代码

注 意：该指令只针对连续插补-直线插补运动。

short dmc_ellipse_move (WORD CardNo, WORD Crd, WORD AxisNum, WORD
AxisList,double TargetPos,double* CenPos,double ALen,double BLen, WORD Dir, WORD
PosMode)

功 能：启动椭圆插补运动

参 数：CardNo 控制器链接号
 Crd 坐标系号 0-3
 AxisNum 坐标系轴数量，保留参数，固定为 2

AxisList	坐标系轴列表
TargetPos	目标位置列表
CenPos	椭圆圆心位置
Alen	长半轴长度
Blen	短半轴长度
Dir	插补方向, 0 顺时针插补, 1 逆时针插补
PosMode	插补模式, 0 相对运动, 1 绝对运动

返回值: 错误代码

short dmc_set_tangent_follow (WORD CardNo, WORD Crd, WORD Axis, WORD FollowCurve, WORD RotDir, double DegreeEquiv)

功 能: 设置切向跟随参数

参 数: CardNo	控制器链接号
Crd	坐标系号 0-3
Axis	跟随轴号
FollowCurve	跟随曲线类型, 0 跟随圆弧; 1 跟随椭圆
RotDir	跟随轴旋转方向, 0 跟随轴负向旋转, 1 跟随轴正向旋转
DegreeEquiv	角度当量, 单位 pulse/degree, 跟随轴每旋转一度, 控制器下发的脉冲数

返回值: 错误代码

short dmc_get_tangent_follow_param (WORD CardNo, WORD Crd, WORD * Axis, WORD * FollowCurve, WORD * RotDir, double* DegreeEquiv)

功 能: 读取切向跟随参数

参 数: CardNo	控制器链接号
Crd	坐标系号 0-3
Axis	回读的跟随轴号
FollowCurve	回读的跟随曲线类型
RotDir	回读的跟随轴旋转方向
DegreeEquiv	回读的角度当量

返回值: 错误代码

short dmc_disable_follow_move (WORD CardNo, WORD Crd)

功 能: 取消指定坐标系的跟随运动

参 数: CardNo 控制器链接号
 Crd 坐标系号 0-3
返回值: 错误代码

8.7 连续插补运动

该部分指令 EMC-E3000 不支持

short dmc_conti_open_list(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList)

功 能: 打开连续插补缓冲区

参 数: CardNo 控制器链接号
 Crd 坐标系号, 取值范围: 0~3
 AxisNum 轴数
 AxisList 轴号列表

返回值: 错误代码

说 明: 1) 连续缓冲区最多可缓存 5000 条指令

2) 当打开连续插补缓冲区后, 则进入连续插补模式; 此时, 除非当执行完缓冲区中的指令或是调用停止连续插补指令 dmc_conti_stop_list 后, 参与连续插补的运动轴才能退出连续插补模式

short dmc_conti_start_list(WORD CardNo, WORD Crd)

功 能: 开始连续插补

参 数: CardNo 控制器链接号
 Crd 坐标系号, 取值范围: 0~3

返回值: 错误代码

short dmc_conti_close_list(WORD CardNo, WORD Crd)

功 能: 关闭连续插补缓冲区

参 数: CardNo 控制器链接号
 Crd 坐标系号, 取值范围: 0~3

返回值: 错误代码

short dmc_conti_pause_list(WORD CardNo, WORD Crd)

功 能: 暂停连续插补

参 数: CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

返回值：错误代码

说明：当暂停连续插补后，连续插补运动将减速停止，当再次调用 dmc_conti_start_list 指令时控制器将继续运行之前未完成的连续插补轨迹

short dmc_conti_stop_list(WORD CardNo, WORD Crd, WORD stop_mode)

功 能：停止插补运动

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

stop_mode 停止模式，0：减速停止，1：立即停止

返回值：错误代码

说明：1) 该函数适用于所有插补运动

2) 当正在执行插补运动时，通过此指令可以中止插补运动，并使参与插补的运动轴退出插补模式

short dmc_conti_delay(WORD CardNo, WORD Crd, double delay_time, long mark)

功 能：连续插补中暂停延时指令

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

delay_time 延时时间，单位：秒

mark 标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 延时时间为运动停止时的等待时间

2) 当延时时间设置为 0 时，延时时间将无限长

short dmc_conti_change_speed_ratio (WORD CardNo, WORD Crd, double Percent)

功 能：动态调整连续插补速度比例

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

Percent 速度比例，取值范围：0~2.0

返回值：错误代码

说明：1) 当计算机执行到此指令时，控制器将立即调整连续插补速度比例，此指令一般在调试中使用；

2) 该函数必须在打开缓冲区（dmc_conti_open_list）后才能调用；

short dmc_conti_line_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double* Target_Pos, WORD posi_mode, long mark)

功 能：连续插补中直线插补指令

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
AxisNum 轴数
AxisList 轴号列表
Target_Pos 目标位置数组，单位：unit
posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式
mark 标号，任意指定，0 表示自动编号

返回值：错误代码

short dmc_conti_arc_move_center_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double *Cen_Pos, WORD Arc_Dir, long Circle, WORD posi_mode, long mark)

功 能：连续插补中基于圆心圆弧扩展的螺旋线插补指令（可作两轴圆弧插补）

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
AxisNum 轴数
AxisList 轴号列表
Target_Pos 目标位置数组，单位：unit
Cen_Pos 圆心位置数组，单位：unit
Arc_Dir 圆弧方向，0：顺时针，1：逆时针
Circle 圈数：
 非负数：表示此时执行的为螺旋线插补
 该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，
 1 即表示 1 圈螺旋线插补...
posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式
mark 标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 dmc_conti_open_list 的说明

2) 当轴数为 2 时，轴列表前两轴进行平面螺旋插补

3) 当轴数为 3、运动轨迹为螺旋插补时，轴列表前两轴平面为基面，进行平面螺旋

插补；同时，轴列表第三轴运动指定高度，该轴终点位置与该轴起点位置的差值为螺旋线段相对于基面的高度

- 4) 当轴数大于 3、运动轨迹为螺旋插补时，主动轴进行螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 `dmc_conti_open_list` 的说明

- 5) 当运动轨迹为螺旋插补时：

轴列表前两轴组成的基面上，当起始点到圆心的距离小于终点到圆心的距离，为绽放螺旋线

轴列表前两轴组成的基面上，当起始点到圆心的距离大于终点到圆心的距离，为收敛螺旋线

轴列表前两轴组成的基面上，当起始点到圆心的距离等于终点到圆心的距离，为圆弧插补（插补轴数为 3 时则为圆柱螺旋线）

`short dmc_conti_arc_move_radius_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double Arc_Radius, WORD Arc_Dir, long Circle, WORD posi_mode, long mark)`

功 能：连续插补中基于半径圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

AxisNum 轴数

AxisList 轴号列表

Target_Pos 目标位置数组，单位：unit

Arc_Radius 圆弧半径值（负值为优弧，正值为劣弧），单位：unit

Arc_Dir 圆弧方向，0：顺时针，1：逆时针

Circle 圈数，取值范围：大于等于 0

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补...

posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

mark 标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 `dmc_conti_open_list` 的说明

2) 当轴数为 2 时，轴列表前两轴进行平面圆弧插补

3) 当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第

三轴运动指定高度：该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

- 4) 当轴数大于 3 时，主动轴进行圆柱螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 `dmc_conti_open_list` 的说明

`short dmc_conti_arc_move_3points_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double *Mid_Pos, long Circle, WORD posi_mode, long mark)`

功 能：连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴及三轴圆弧插补）

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

AxisNum 轴数

AxisList 轴号列表

Target_Pos 目标位置数组，单位：unit

Mid_Pos 中间位置数组，单位：unit

Circle 圈数

负数：表示此时执行的为空间圆弧插补

该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧...

自然数：表示此时执行的为圆柱螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补...

posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

mark 标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 `dmc_conti_open_list` 的说明

2) 当轴数为 2 时，轴列表前两轴进行平面圆弧插补

3) 当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

4) 当轴数大于 3 时，主动轴进行圆柱螺旋插补或空间圆弧插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 `dmc_conti_open_list` 的说明

short dmc_conti_rectangle_move_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD *AxisList, double *Target_Pos, double *Mark_Pos, long Count, WORD rect_mode, WORD posi_mode, long mark)

功 能：连续插补中矩形插补指令

参 数：CardNo	控制器链接号
Crd	坐标系号，取值范围：0~3
AxisNum	轴数， 保留参数 ，固定值为 2
AxisList	轴号列表
Target_Pos	对角位置数组，单位：unit
Mark_Pos	矩形方向标记位置数组，单位：unit
Count	行数/圈数
rect_mode	矩形插补模式，0：逐行，1：渐开线
posi_mode	运动模式，0：相对坐标模式，1：绝对坐标模式
mark	标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：该指令只支持 2 轴矩形插补运动

short dmc_conti_pmove_unit(WORD CardNo, WORD Crd, WORD axis, double dist, WORD posi_mode, WORD mode, long imark)

功 能：连续插补中控制指定轴运动

参 数：CardNo	控制器链接号
Crd	坐标系号，取值范围：0~3
axis	指定轴号
dist	目标位置，单位：unit
posi_mode	运动模式，0：相对坐标模式，1：绝对坐标模式
mode	模式： 0：暂停启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动；当本段定长运动结束后，再执行下一段插补运动） 1：直接启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动，并且同时执行下一段插补运动）
mark	标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 该指令可以实现在连续插补运动中，控制指定轴做定长运动
2) 该轴不能为参与连续插补的运动轴，即不能为插补系轴列表中的轴

- 3) 在使用该指令控制轴运动前, 必须先使用函数 `dmc_set_profile_unit` 设置该轴的运行速度。

`short dmc_conti_pmove_unit_pausemode(WORD CardNo, WORD axis, double TargetPos, double Min_Vel, double Max_Vel, double stop_Vel, double acc, double dec, double smooth_time, WORD posi_mode)`

功 能: 连续插补暂停后, 执行单轴运动

参 数: CardNo 控制器链接号

Axis 轴号

TargetPos 目标位置

Min_Vel 单轴运动起始速度

Max_Vel 单轴运动运行速度

stop_Vel 单轴运动停止速度

acc 单轴运动加速度

dec 单轴运动减速度

smooth_time 单轴运动平滑时间

posi_mode 运动模式, 0: 相对坐标模式, 1: 绝对坐标模式

返回值: 错误代码

`short dmc_conti_return_pausemode(WORD CardNo, WORD Crd, WORD axis)`

功 能: 连续插补暂停, 执行单轴运动后, 回到暂停位置

参 数: CardNo 控制器链接号

Crd 插补系号, 0-7

Axis 轴号

返回值: 错误代码

8.8 连续插补缓冲区检测

`long dmc_conti_remain_space(WORD CardNo, WORD Crd)`

功 能: 查询连续插补缓冲区剩余插补空间

参 数: CardNo 控制器链接号

Crd 坐标系号, 取值范围: 0~3

返回值: 连续插补缓冲区剩余大小

long dmc_conti_read_current_mark (WORD CardNo, WORD Crd)

功 能：读取连续插补缓冲区当前插补段号

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3

返回值：当前连续插补段号

8.9 连续插补小线段前瞻功能

short dmc_conti_set_lookahead_mode(WORD CardNo, WORD Crd, WORD enable, long LookaheadSegments, double PathError, double LookaheadAcc)

功 能：设置连续插补前瞻参数

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 enable 前瞻使能状态，0：禁止，1：使能
 LookaheadSegments 前瞻段数，参数保留
 PathError 允许误差范围，取值范围：非负数,单位：unit
 LookaheadAcc 前瞻加速度，参数保留

返回值：错误代码

注意：小线段前瞻支持圆弧过渡和非圆弧过渡两种方式，设置轨迹误差范围为零时没有圆弧过渡，不为零时默认有圆弧过渡，且普通连续插补没有圆弧过渡功能

short dmc_conti_get_lookahead_mode(WORD CardNo, WORD Crd, WORD* enable, long* LookaheadSegments, double* PathError, double* LookaheadAcc)

功 能：读取连续插补前瞻参数

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 enable 前瞻使能状态，0：禁止，1：使能
 LookaheadSegments 前瞻段数
 PathError 允许误差范围，取值范围：非负数,单位：unit
 LookaheadAcc 前瞻加速度

返回值：错误代码

8.10 连续插补 IO 控制

short dmc_conti_set_pause_output(WORD CardNo, WORD crd, WORD actIOOn, long mask, long state)

功 能：设置连续插补暂停及异常停止时 IO 输出状态

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
actIOOn 激活模式：
 0：保持原状
 1：暂停连续插补时输出设定的 IO 状态，恢复运行时不恢复暂停前的 IO 状态
 2：暂停连续插补时输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态
 3：当暂停、停止连续插补，或遇到其他异常停止（如碰到 EMG 信号）时，输出设定的 IO 状态
mask 选择输出端口标志：bit0~bit31 代表 Out0~Out31，位值为 1 时输出，位值为 0 不输出
state 输出电平状态：bit0~bit31 代表 Out0~Out31，位值为 1 时输出高电平，位值为 0 时输出低电平

返回值：错误代码

激活模式 3 的说明：1) 暂停连续插补时，控制器输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态
2) 停止连续插补、或遇到其他异常停止时，控制器输出设定的 IO 状态，但是再次启动连续插补时不会恢复之前的 IO 状态

short dmc_conti_get_pause_output(WORD CardNo, WORD crd, WORD* actIOOn, long* mask, long* state)

功 能：读取连续插补暂停及异常停止时 IO 输出状态设置

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
actIOOn 返回激活状态设置
mask 返回输出选择标志设置
state 返回输出电平状态设置

返回值：错误代码

short dmc_conti_wait_input(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double TimeOut, long mark)

功 能：连续插补等待 IO 输入.当控制器执行到此指令时，只有在接受到输入 IO 信号或超出超时时间后，才会执行后续运动

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
bitno 输入口号，取值范围：0~31
on_off 电平状态，0：低电平，1：高电平
TimeOut 超时时间，单位：s
mark 标号，任意指定，0 表示自动编号

返回值：错误代码

注 意：1) 当超时时间设为 0 时，控制器将一直等待 IO 输入信号，超时时间为无限长
2) 超时时间为运动停止时的等待时间

short dmc_conti_delay_outbit_to_start(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double delay_value, WORD delay_mode, double ReverseTime)

功 能：连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
bitno 输出口号，取值范围：0~31
on_off 电平状态，0：低电平，1：高电平
delay_value 滞后值，单位：s（滞后时间模式）或 unit（滞后距离模式）
delay_mode 滞后模式，0：滞后时间，1：滞后距离
ReverseTime 电平输出后的延时翻转时间，单位：s

返回值：错误代码

注 意：1) 设置的 IO 操作，将在该指令的下一条轨迹中起作用
2) 当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变
3) 当滞后模式选择为滞后距离时，位置源为指令位置计数器

short dmc_conti_delay_outbit_to_stop(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double delay_time, double ReverseTime)

功 能：连续插补中相对于轨迹段终点 IO 滞后输出

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3

bitno 输出口号，取值范围：0~31
on_off 电平状态，0：低电平，1：高电平
delay_time 滞后时间，单位：s
ReverseTime 电平输出后的延时翻转时间，单位：s

返回值：错误代码

注 意：1) 设置的 IO 操作，将在该指令的下一条轨迹结束后起作用
2) 如果使用了 dmc_conti_clear_IO_actIOon 函数清除段内未执行完的 IO 动作时，那么该指令将不会被执行

short dmc_conti_ahead_outbit_to_stop(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double ahead_value, WORD ahead_mode, double ReverseTime)

功 能：连续插补中相对于轨迹段终点 IO 提前输出（段内执行）

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
bitno 输出口号，取值范围：0~31
on_off 电平状态，0：低电平，1：高电平
ahead_value 提前值，单位：s（提前时间模式）或 unit（提前距离模式）
ahead_mode 提前模式，0：提前时间，1：提前距离
ReverseTime 电平输出后的延时翻转时间，单位：s

返回值：错误代码

注 意：1) 设置的 IO 操作，将在该指令的下一条轨迹中起作用
2) 当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变
3) 当滞后模式选择为滞后距离时，位置源为指令位置计数器

short dmc_conti_write_outbit(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double ReverseTime)

功 能：连续插补中缓冲区立即 IO 输出

参 数：CardNo 控制器链接号
Crd 坐标系号，取值范围：0~3
bitno 输出口号，取值范围：0~31
on_off 电平状态，0：低电平，1：高电平
ReverseTime 电平输出后的延时翻转时间，单位：s

返回值：错误代码

说 明：1) 当计算机执行到此指令时，将该指令存入缓冲区，当缓冲区中的上一段运动指令

执行完毕时，该指令将执行

- 2) 当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变

short dmc_conti_clear_IO_actIOn(WORD CardNo, WORD Crd, DWORD IOMask)

功 能：清除段内未执行完的 IO 动作

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

IOMask 清除标志：bit0~bit31 分别表示 Out0~Out31 输出口；位值：1：清除对应输出口段内未执行完的动作，比如翻转时间到达后 IO 翻转动作；0：不操作

返回值：错误代码

说 明：该函数对 dmc_conti_delay_outbit_to_start、dmc_conti_ahead_outbit_to_stop、dmc_conti_delay_outbit_to_stop 指令起作用

8.11 圆弧限速

short dmc_set_arc_limit(WORD CardNo, WORD Crd, WORD Enable, double MaxCenAcc, double MaxArcError)

功 能：设置指定坐标系的圆弧限速参数

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

Enable 使能状态，0：禁止，1：使能

MaxCenAcc 保留参数

MaxArcError 保留参数

返回值：错误代码

short dmc_get_arc_limit(WORD CardNo, WORD Crd, WORD* Enable, double* MaxCenAcc, double* MaxArcError)

功 能：读取指定坐标系的圆弧限速参数

参 数：CardNo 控制器链接号

Crd 坐标系号，取值范围：0~3

Enable 使能状态，0：禁止，1：使能

MaxCenAcc 保留参数

MaxArcError 保留参数

返回值：错误代码

8.12 连续插补位置跟随

short dmc_conti_gear_unit(WORD CardNo,WORD Crd,WORD axis,double dist,WORD follow_mode,long imark)

功 能：设置轴位置跟随

参 数：CardNo 控制器链接号

 Crd 坐标系号，取值范围：0~3

 axis 指定轴号

 dist 跟随距离，单位：unit

 follow_mode 0 表示相对位置跟随模式，1 表示轴跟随插补速度的模式

 mark 标号，任意指定，0 表示自动编号

返回值：错误码

注 意：

- 1) 进行跟随运动的轴不能是插补坐标系中的轴；发送该指令时轴如果在运动中，指令将不能正常执行。
- 2) 跟随距离是相对当前位置的相对距离，即下一段插补段运动过程中，跟随轴需要运动的距离。

8.13 PWM 功能

short dmc_set_pwm_enable_extern(WORD CardNo, WORD pwm_no, WORD enable)

功 能：设置 PWM 使能状态

参 数：CardNo 控制器链接号

 pwm_no PWM 通道号，0-5

 enable PWM 使能状态，0：禁止，1：使能

返回值：错误代码

short dmc_get_pwm_enable_extern(WORD CardNo, WORD pwm_no, WORD enable)

功 能：读取 PWM 使能状态设置

参 数：CardNo 控制器链接号

 pwm_no PWM 通道号，0-5

 enable PWM 使能状态，0：禁止，1：使能

返回值：错误代码

`short dmc_set_pwm_output(WORD CardNo, WORD pwm_no, double fDuty, double fFre)`

功 能：设置 PWM 立即输出

参 数：CardNo 控制器链接号
 pwm_no PWM 通道，取值范围：0~5
 fDuty 占空比，取值范围：0~1
 fFre 频率，取值范围：0~500KHz

返回值：错误代码

`short dmc_get_pwm_output(WORD CardNo, WORD pwm_no, double* fDuty, double* fFre)`

功 能：读取 PWM 立即输出设置

参 数：CardNo 控制器链接号
 pwm_no PWM 通道，取值范围：0~5
 fDuty 返回占空比设置值
 fFre 返回频率设置值

返回值：错误代码

`short dmc_set_pwm_onoff_duty( WORD CardNo, WORD PwmNo, double fOnDuty, double fOffDuty)`

功 能：设置 PWM 开关状态对应的占空比

参 数：CardNo 控制器链接号
 PwmNo PWM 通道，取值范围：0~5
 fOnDuty PWM 打开状态的占空比，取值范围：0~1
 fOffDuty PWM 关闭状态的占空比，取值范围：0~1

返回值：错误代码

`short dmc_get_pwm_onoff_duty( WORD CardNo, WORD PwmNo, double* fOnDuty, double* fOffDuty)`

功 能：读取 PWM 开关状态对应占空比的设置

参 数：CardNo 控制器链接号
 PwmNo PWM 通道，取值范围：0~5
 fOnDuty 返回 PWM 打开状态的占空比设置值
 fOffDuty 返回 PWM 关闭状态的占空比设置值

返回值：错误代码

short dmc_conti_set_pwm_output(WORD CardNo, WORD Crd, WORD pwm_no, double fDuty, double fFre)

功 能：连续插补中 PWM 输出设置

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 pwm_no PWM 通道，取值范围：0~5
 fDuty 占空比，取值范围：0~1
 fFre 频率，取值范围：0~2MHz

返回值：错误代码

注 意：该函数不能与 dmc_conti_write_pwm、dmc_conti_delay_pwm_to_start、dmc_conti_ahead_pwm_to_stop 等函数同时使用

short dmc_conti_set_pwm_follow_speed(WORD CardNo, WORD Crd, WORD pwm_no, WORD mode, double MaxVel, double MaxValue, double OutValue)

功 能：连续插补中 PWM 速度跟随

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 pwm_no PWM 通道，取值范围：0~5
 mode 跟随模式：
 0：不跟随，保持状态
 1：不跟随，输出低电平
 2：不跟随，输出高电平
 3：跟随，占空比自动调整
 4：跟随，频率自动调整
 MaxVel 最大运行速度，单位：unit/s
 MaxValue 最大输出值：
 跟随模式为 3 时：占空比，取值范围：0~1
 跟随模式为 4 时：频率，取值范围：0~2MHz
 OutValue 固定输出值：
 跟随模式为 3 时：频率，取值范围：0~2MHz
 跟随模式为 4 时：占空比，取值范围：0~1

返回值：错误代码

- 注 意：**1) 当设置跟随模式为 3，即占空比自动调整时，运行速度（0~MaxVel）与占空比（0~MaxValue）成线性关系，参数 OutValue 确定 PWM 输出频率。
- 2) 当设置跟随模式为 4，即频率自动调整时，运行速度（0~MaxVel）与输出频率（0~MaxValue）成线性关系，参数 OutValue 确定 PWM 输出占空比。
- 3) 模式 0、1、2 本为占空比可调，固定的频率通过函数 `conti_set_pwm_follow_speed` 中的 `outvalue` 设置，如果该值设置为 0，采用上次设置的频率值。

`short dmc_conti_get_pwm_follow_speed(WORD CardNo, WORD Crd, WORD pwm_no, WORD* mode, double* MaxVel, double* MaxValue, double* OutValue)`

功 能：读取 PWM 速度跟随参数设置

参 数：CardNo 控制器链接号

 Crd 坐标系号，取值范围：0~3

 pwm_no PWM 通道，取值范围：0~5

 mode 返回跟随模式设置值

 MaxVel 返回最大运行速度设置值

 MaxValue 返回最大输出占空比或频率设置值

 OutValue 返回固定输出频率或占空比设置值

返回值：错误代码

`short dmc_conti_delay_pwm_to_start(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double delay_value, WORD delay_mode, double ReverseTime)`

功 能：连续插补中相对于轨迹段起点 PWM 滞后输出

参 数：CardNo 控制器链接号

 Crd 坐标系号，取值范围：0~3

 pwm_no PWM 通道，取值范围：0~5

 on_off 输出状态，0：关闭，1：打开

 delay_value 滞后值，单位：s（滞后时间模式）或 unit（滞后距离模式）

 delay_mode 滞后模式，0：滞后时间，1：滞后距离

 ReverseTime 保留参数，固定值为 0

返回值：错误代码

- 注 意：**1) 调用此指令前，必须先调用函数 `dmc_set_pwm_onoff_duty` 设置 PWM 关闭及打开状态的占空比。PWM 波形的频率为之前调用 PWM 时设置的频率
- 2) 当函数 `dmc_conti_set_pwm_follow_speed` 设置的跟随模式为 0 时：
- 参数 `on_off` 设置为 0（关闭状态）时，输出 PWM 波形的占空比为函数

dmc_set_pwm_onoff_duty 设置的关闭状态占空比值;

参数 on_off 设置为 1 (打开状态) 时, 输出 PWM 波形的占空比为函数 dmc_set_pwm_onoff_duty 设置的打开状态占空比值

3) 当函数 dmc_conti_set_pwm_follow_speed 设置的跟随模式不为 0 时:

参数 on_off 设置为 0 (关闭状态) 时, 输出 PWM 波形的占空比为函数 dmc_set_pwm_onoff_duty 设置的关闭状态占空比值;

参数 on_off 设置为 1 (打开状态) 时, 输出 PWM 波形为函数 dmc_conti_set_pwm_follow_speed 设置的跟随模式样式

short dmc_conti_ahead_pwm_to_stop(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double ahead_value, WORD ahead_mode, double ReverseTime)

功 能: 连续插补中相对于轨迹段终点 PWM 提前输出

参 数: CardNo 控制器链接号
Crd 坐标系号, 取值范围: 0~3
pwm_no PWM 通道, 取值范围: 0~5
on_off 输出状态, 0: 关闭, 1: 打开
ahead_value 提前值, 单位: s (滞后时间模式) 或 unit (滞后距离模式)
delay_mode 提前模式, 0: 滞后时间, 1: 滞后距离
ReverseTime 保留参数, 固定值为 0

返回值: 错误代码

注 意: 同函数“dmc_conti_delay_pwm_to_start”

short dmc_conti_write_pwm(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double ReverseTime)

功 能: 连续插补中缓冲区立即 PWM 输出

参 数: CardNo 控制器链接号
Crd 坐标系号, 取值范围: 0~3
pwm_no PWM 通道, 取值范围: 0~5
on_off 输出状态, 0: 关闭, 1: 打开
ReverseTime 保留参数, 固定值为 0

返回值: 错误代码

注 意: 同函数“dmc_conti_delay_pwm_to_start”

8.14 连续插补 DA 跟随

short dmc_conti_set_da_follow_speed(WORD CardNo, WORD Crd, WORD da_no, double MaxVel, double MaxValue, double acc_offset, double dec_offset, double acc_dist, double dec_dist)

功 能：连续插补中 DA 速度跟随

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 da_no DA 通道
 MaxVel DA 跟随最大速度，单位：unit/s
 MaxValue DA 跟随最大输出值，单位 V
 acc_offset 加速段偏差，单位 V
 dec_offset 减速段偏差，单位 V
 acc_dist 保留参数
 dec_dist 保留参数

返回值：错误代码

注 意：DA 跟随配置需在 open_list 前执行。

short dmc_conti_get_da_follow_speed(WORD CardNo, WORD Crd, WORD da_no, double *MaxVel, double *MaxValue, double *acc_offset, double *dec_offset, double *acc_dist, double *dec_dist)

功 能：读取连续插补中 DA 跟随速度

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 da_no DA 通道
 MaxVel DA 跟随最大速度
 MaxValue DA 跟随最大输出值
 acc_offset 加速段偏差
 dec_offset 减速段偏差
 acc_dist 保留参数
 dec_dist 保留参数

返回值：错误代码

short dmc_conti_set_da_enable(WORD CardNo, WORD Crd, WORD enable,WORD channel,long mark);

功 能：连续缓冲区内的 DA 跟随使能

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 channel 通道号
 enable DA 跟随，0：禁止，1：使能
 mark 标号，任意指定，0 表示自动编号

返回值：错误代码

8.15 位置计数器控制

short dmc_set_positIon_unit(WORD CardNo, WORD axis, double pos)

功 能：设置指定轴的当前指令位置计数器值

参 数：CardNo 控制器链接号
 axis 指定轴号
 pos 位置值，单位：unit

返回值：错误代码

short dmc_get_positIon_unit(WORD CardNo, WORD axis, double *pos)

功 能：读取指定轴的当前指令位置计数器值

参 数：CardNo 控制器链接号
 axis 指定轴号
 pos 返回当前位置值，单位：unit

返回值：错误代码

8.16 运动状态检测及控制

short dmc_get_axis_run_mode(WORD CardNo, WORD axis, WORD* run_mode)

功 能：读取指定轴的运动模式

参 数：CardNo 控制器链接号
 axis 指定轴号
 run_mode 返回运动模式：

0,空闲模式

1,定长模式

2,定速模式

- 3,回零模式
- 5,PT 模式
- 6,PVT 模式
- 7,电子齿轮模式
- 8,电子凸轮模式
- 9,单段直线模式
- 10,连续插补模式
- 11,停止模式
- 12,限位模式
- 13,驱动器 PP 模式
- 14,龙门模式
- 17,退出龙门模式
- 18,门型运动
- 19,插补速度跟随模式
- 20,正弦振荡曲线模式
- 21,插补速度跟随模式
- 22,转矩控制模式
- 23,PDO 缓存运动模式
- 24,驱动器 PV 模式
- 25,插补暂停定长运动模式

返回值：错误代码

说 明：1) 该函数适用于所有运动，使用该函数可读取当前的运动模式

2) 当执行基于脉冲当量的插补及连续插补运动时，使用该函数读取运动模式为 10

`short dmc_read_current_speed_unit(WORD CardNo, WORD axis, double *current_speed)`

功 能：读取指定轴的当前单轴速度

参 数：CardNo 控制器链接号
 axis 指定轴号
 current_speed 返回速度值，单位：unit/s

返回值：错误代码

注 意：当执行基于脉冲当量的插补运动时，使用该函数读取的为各轴的单轴速度

`short dmc_read_vector_speed_unit(WORD CardNo,WORD Crd,double *current_speed)`

功 能：读取插补运动的合速度

参 数: CardNo 控制器链接号
 Crd 坐标系号, 取值范围: 0~3
 current_speed 返回插补合速度值, 单位: unit/s

返回值: 错误代码

short dmc_check_done(WORD CardNo, WORD axis)

功 能: 检测指定轴的运动状态

参 数: CardNo 控制器链接号
 axis 指定轴号

返回值: 0: 指定轴正在运行, 1: 指定轴已停止

注 意: 此函数适用于单轴、PVT 运动

short dmc_check_done_multicoor(WORD CardNo, WORD Crd)

功 能: 检测坐标系的运动状态

参 数: CardNo 控制器链接号
 Crd 指定控制器上的坐标系号 (取值范围: 0~7)

返回值: 坐标系状态, 0: 正在使用中, 1: 正常停止

注 意: 此函数适用于插补运动

short dmc_conti_get_run_state(WORD CardNo, WORD crd)

功 能: 读取指定坐标系的插补运动状态

参 数: CardNo 控制器链接号
 Crd 坐标系号, 取值范围: 0~3

返回值: 运动状态, 0: 运动中, 1: 暂停中, 2: 正常停止, 3: 未启动, 4: 空闲

说明: 1)该函数适用于所有插补

2)“运动中”指的是插补运动正在进行

3)“暂停中”指的运动未完成时暂停指令生效后的状态, 调用 startlist 可重新启动

4)“正常停止”指的是缓冲区内运动都执行完成后运动停止

5)“未启动”指打开缓冲区后轴已被坐标系占用但还没开始运动的状态, 此时被加坐标系中的轴不能再被调用

6)“空闲”指坐标系未打开的状态, 调用 stoplist 后进入此状态

short dmc_get_stop_reason(WORD CardNo, WORD axis, long* StopReason)

功 能: 读取指定轴的停止原因

参 数: CardNo	控制器链接号
axis	指定轴号
StopReason	停止原因:
	0: 正常停止
	1: ALM 立即停止
	2: ALM 减速停止
	3: LTC 外部触发立即停止
	4: EMG 立即停止
	5: 正硬限位立即停止
	6: 负硬限位立即停止
	7: 正硬限位减速停止
	8: 负硬限位减速停止
	9: 正软限位立即停止
	10: 负软限位立即停止
	11: 正软限位减速停止
	12: 负软限位减速停止
	13: 命令立即停止
	14: 命令减速停止
	15: 其它轴引起的立即停止
	16: 其它轴引起的减速停止
	17: 模拟量超限, 触发急停
	18: 保留
	19: DSTP 信号引起的减速停止
	20: 前瞻预处理和速度规划来不及, 底层自动调用 stoplist
	21: 原点不在两个限位之间停止
	22: 回零方向和当前有效限位端相反 (正限位有效时, 回零方向为负方向; 负限位有效时, 回零方向为正方向)
	23: 正负限位同时有效
	24: 没有找到 EZ 信号
	25: 回零位置溢出停止
	26: 动加速度过大, 异常停止
	27: 双原点停止
	28: 区域限位立即停止
	29: 区域限位减速停止

- 30: 龙门超差保护减速停止
- 31: 龙门超差保护立即停止
- 32: 插补规划错误停止
- 33: 单轴跟踪误差保护立即停止
- 34: 单轴跟踪误差保护减速停止
- 35: 插补缓冲区错误停止
- 36: 龙门主从轴超差保护立即停止
- 37: 龙门主从轴超差保护减速停止
- 38: 轴输出脉冲频率超过 4M
- 39: IO 触发立即停止
- 40: IO 触发减速停止
- 49: 看门狗超时
- 50: 两轴碰撞检测立即停止
- 51: 两轴碰撞检测减速停止
- 52: 龙门从轴 alm 引起主轴立即停止
- 53: 从轴开始启动时主轴绝对位置错误（不在主轴运动方向上）
- 54: 同步时主轴绝对位置错误（不在主轴运动方向上）
- 55: 从轴开始运动位置大于同步时主轴绝对位置错误
- 56: 同步时从轴绝对位置（不在从轴运动方向上）
- 57: 不支持主轴运动模式（仅支持跟随主轴做定长和定速运动模式）
- 58: 从轴追赶目标速度小于同步时从轴的速度,建议调大追赶目标速度
- 59: 从轴同步位置过小，不足以从轴从当前速度直接加速或减速至同步速度,建议调大从轴同步位置或调大加减速速度
- 60: 初始化时主轴运动方向错误(运动速度为零)
- 61: 从轴开始运动时主轴未达到匀速状态,建议调大从轴启动时主轴的位置
- 62: GearInPos 实际加速度大于设置的最大加速度
- 63: GearInPos 实际减速度大于设置的最大减速度
- 64: 从轴追赶实际可达速度大于追赶设置的最大目标速度,且以最大追赶速度运行无法实现规定时间内达到从轴的位移,建议调大追赶目标速度或减小从轴同步位置
- 65: 从轴同步速度在只有加速或减速过程也不可达速度,建议调大从轴从轴同步位置或调小齿轮比或调大从轴开始运动的主轴位置
- 66: 二分法求取合适的可达速度，迭代次数超过限制次数让无法找到

合适的速度值

- 67: 主轴 pmove 目标位置小于主轴同步位置
- 68: 主轴 pmove 目标位置小于从轴开始运动时的主轴位置
- 69: 从轴追赶过程加速周期或减速周期为 0, 有可能主轴同步位置和从轴开始运动时的主轴位置相隔太近
- 70: 主轴目标速度为零
- 71: 从轴实际可达速度大于追赶速度时, 重算以追赶速度为目标速度的匀速周期数小于等于零
- 72: GearInPos 主轴位移与规划结果不一致
- 73: GearInPos 主轴 vmove 减速阶段不支持
- 74: 从轴起始反向运动方向模式错误(1~4)
- 75: 从轴起始反向运动加速度为零
- 76: 连续轨迹等待外部 IO 输入信号有效超时, 停止插补系运动
- 201: 正负限位之间全程没找到原点信号
- 202: 回零方向不匹配
- 203: 正负限位同时有效
- 204: 正负限位之间全程没有 EZ 信号
- 205: 位置溢出
- 206: 双原点错误
- 207: 外部信号触发回零停止
- 208: 驱动器回零被中断停止

返回值: 错误代码

short dmc_clear_stop_reason(WORD CardNo, WORD axis)

功 能: 清除指定轴的停止原因

参 数: CardNo 控制器链接号
 axis 指定轴号

返回值: 错误代码

DWORD dmc_axis_IO_status(WORD CardNo, WORD axis)

功 能: 读取指定轴有关运动信号的状态

参 数: CardNo 控制器链接号
 axis 指定轴号,

返回值: 见表 8.2

表 8.2 轴的运动信号状态

位号	信号名称	描述
0	ALM	1: 表示伺服报警信号 ALM 为 ON; 0: OFF
1	EL+	1: 表示正硬限位信号 +EL 为 ON; 0: OFF
2	EL-	1: 表示负硬限位信号-EL 为 ON; 0: OFF
3	EMG	1: 表示急停信号 EMG 为 ON; 0: OFF
4	ORG	1: 表示原点信号 ORG 为 ON; 0: OFF
6	SL+	1: 表示正软限位信号+SL 为 ON; 0: OFF
7	SL-	1: 表示负软件限位信号-SL 为 ON; 0: OFF
其他位	保留	

short dmc_stop(WORD CardNo, WORD axis, WORD stop_mode)

功 能：指定轴停止运动

参 数：CardNo 控制器链接号

axis 指定轴号

stop_mode 制动方式，0：减速停止，1：紧急停止

返回值：错误代码

注 意：此函数适用于单轴、PVT 运动

short dmc_stop_multicoor(WORD CardNo, WORD Crd, WORD stop_mode)

功 能：停止坐标系内所有轴的运动

参 数：CardNo 控制器链接号

Crd 指定控制器上的坐标系号（取值范围：0~7）

stop_mode 制动方式，0：减速停止，1：立即停止

返回值：错误代码

注 意：此函数适用于插补运动

short dmc_EMG_stop(WORD CardNo)

功 能：紧急停止所有轴

参 数：CardNo 控制器链接号

返回值：错误代码

注 意：此函数适用于所有运动模式

short dmc_get_target_positIOOn_unit(WORD CardNo, WORD axis, double *pos)

功 能：读取正在运动的目标位置（绝对坐标）

参 数：CardNo 控制器链接号

axis 指定轴号
pos 目标位置，单位：unit

返回值：错误码

注 意：此函数适用于 pmove 运动模式

8.17 输入输出 IO

short dmc_read_inbit_ex(WORD CardNo, WORD bitno)

功 能：读取指定控制器的某个输入端口的电平

参 数：CardNo 控制器链接号

 bitno 输入端口号，取值范围： 0~15，如果扩展 IO 模块，依次往后累加

返回值：指定的输入端口电平：0：低电平，1：高电平

short dmc_read_inbit_ex_ex(WORD CardNo, WORD bitno, WORD *status)

功 能：读取指定控制器的某个输入端口的电平

参 数：CardNo 控制器链接号

 bitno 输入端口号 0-实际输入口数-1

 status 指定的输入端口电平：0：低电平，1：高电平

返回值：错误代码

short dmc_write_outbit(WORD CardNo, WORD bitno, WORD on_off)

功 能：设置指定控制器的某个输出端口的电平

参 数：CardNo 控制器链接号

 bitno 输出端口号，取值范围： 0~15，如果扩展 IO 模块，依次往后累加

 on_off 输出电平，0：低电平，1：高电平

返回值：错误代码

short dmc_read_outbit(WORD CardNo, WORD bitno)

功 能：读取指定控制器的某个输出端口的电平

参 数：CardNo 控制器链接号

 bitno 输入端口号，取值范围： 0~15，如果扩展 IO 模块，依次往后累加

返回值：指定输出端口的电平，0：低电平，1：高电平

short dmc_read_outbit_ex(WORD CardNo, WORD bitno, WORD *status)

功 能：读取指定控制器的某个输出端口的电平

参 数：CardNo 控制器链接号

bitno 输入端口号

status 指定输出端口的电平，0：低电平，1：高电平

返回值：错误代码

DWORD dmc_read_inport(WORD CardNo, WORD portno)

功 能：读取指定控制器的全部输入端口的电平

参 数：CardNo 控制器链接号

portno IO 组号，EMC-E5000 固定取值为 0

返回值： bit0~bit31 的定义见表 8.3

表 8.3 EMC-E5000 控制器 dmc_read_inport 函数返回值各位的定义表

第 0 组（portno 参数）		
函数返回值的 bit	输入口号	输入口名称
0	0	IN0
1	1	IN1
2	2	IN2
3	3	IN3
4	4	IN4
5	5	IN5
6	6	IN6
7	7	IN7
8	8	IN8
9	9	IN9
10	10	IN10
11	11	IN11
12	12	IN12
13	13	IN13
14	14	IN14
15	15	IN15
15-31	15-31	扩展输入口

short dmc_read_inport_ex(WORD CardNo, WORD portno, DWORD *status)

功 能：读取指定控制器的全部输入端口的电平

参 数：CardNo 控制器链接号

portno IO 组号

status 全部输入端口的电平

返回值：错误代码

DWORD dmc_read_outport(WORD CardNo, WORD portno)

功 能：读取指定控制器的全部输出口的电平

参 数：CardNo 控制器链接号

portno IO 组号

返回值： bit0~bit31 的定义见表 8.4

short dmc_read_outport_ex(WORD CardNo, WORD portno, DWORD *status)

功 能：读取指定控制器的全部输出口的电平

参 数：CardNo 控制器链接号

portno IO 组号

status 对应输出 port 口号的电平

返回值：错误代码

short dmc_write_outport(WORD CardNo, WORD portno, DWORD port_value)

功 能：设置指定控制器的全部输出口的电平

参 数：CardNo 控制器链接号

portno IO 组号

port_value bit0~bit31 的定义见表 8.4

返回值：错误代码

表 8.4 EMC-E5000 控制器 dmc_read_outport、dmc_write_outport 函数返回值各位的定义表

第 0 组（portno 参数）		
函数返回值的 bit	输出口号	输出口名称
0	0	OUT0
1	1	OUT1
2	2	OUT2
3	3	OUT3
4	4	OUT4
5	5	OUT5
6	6	OUT6
7	7	OUT7
8	8	OUT8
9	9	OUT9
10	10	OUT710

第 0 组 (portno 参数)		
函数返回值的 bit	输出口号	输出口名称
11	11	OUT11
12	12	OUT12
13	13	OUT13
14	14	OUT14
15	15	OUT15
15-31	15-31	扩展输出口

short dmc_reverse_outbit(WORD CardNo, WORD bitno, double reverse_time)

功 能: IO 输出延时翻转

参 数: CardNo 控制器链接号
 bitno 输出端口号, 取值范围: 0~7, 如果扩展 IO 模块, 依次往后累加
 reverse_time 延时翻转时间, 单位: s

返回值: 错误代码

注 意: 延时翻转时间参数设置为 0 时, 此时延时翻转时间将为无限大

short dmc_set_io_count_mode(WORD CardNo, WORD bitno, WORD mode, double filter_time)

功 能: 设置 IO 计数模式;

参 数: CardNo 控制器链接号
 bitno 输入端口号, 取值范围: 0~7, 如果扩展 IO 模块, 依次往后累加
 mode IO 计数模式, 0: 禁用, 1: 上升沿计数, 2: 下降沿计数
 filter_time 滤波时间, 单位: s, 保留参数

返回值: 错误代码

short dmc_get_io_count_mode(WORD CardNo, WORD bitno, WORD *mode, double* filter_time)

功 能: 读取 IO 计数模式设置;

参 数: CardNo 控制器链接号
 bitno 输入端口号, 取值范围: 0~7, 如果扩展 IO 模块, 依次往后累加
 mode 返回 IO 计数模式
 filter_time 返回滤波时间, 单位: s, 保留参数

返回值: 错误代码

short dmc_set_io_count_value(WORD CardNo, WORD bitno, DWORD CountValue)

功 能: 设置 IO 计数值;

参 数: CardNo 控制器链接号

bitno 输入端口号，取值范围：0~7，如果扩展 IO 模块，依次往后累加
CountValue IO 计数值

返回值：错误代码

short dmc_get_io_count_value(WORD CardNo,WORD bitno,DWORD* CountValue)

功 能：读取 IO 计数值；

参 数：CardNo 控制器链接号

bitno 输入端口号，取值范围：0~7，如果扩展 IO 模块，依次往后累加

CountValue 返回 IO 计数值

返回值：错误代码

8.18 轴 IO 映射

short dmc_set_axis_io_map(WORD CardNo,WORD Axis,WORD IOType,WORD MapIOType,
WORD MapIOIndex,double filter_time)

功 能：设置轴 IO 映射关系

参 数：CardNo 控制器链接号

Axis 指定总线轴号

IOType 指定总线轴的 IO 信号类型：

3：急停信号，AxisIOInMsg_EMG

4：减速停止信号，AxisIOInMsg_DSTP

MapIOType 轴 IO 映射类型：

6：通用输入端口，AxisIOInPort_IO

MapIOIndex 轴 IO 映射索引号：

1) 当轴 IO 映射类型设置为 6 时，此参数可设置为 0~最大输入口数（包括 EtherCAT 扩展模块），表示该映射对应的具体通用输入端口号，设置为 65535 表示取消该映射关系

filter_time 轴 IO 信号滤波时间，单位：s

返回值：错误代码

short dmc_get_axis_io_map(WORD CardNo,WORD Axis,WORD IOType,WORD* MapIOType,
WORD* MapIOIndex,double* filter_time)

功 能：读取轴 IO 映射关系设置

参 数：CardNo 控制器链接号

Axis	指定总线轴号
IOType	轴 IO 信号类型
MapIOType	返回轴 IO 映射类型
MapIOIndex	返回轴 IO 映射索引号
filter_time	返回轴 IO 信号滤波时间，单位：s

返回值：错误代码

short dmc_set_io_map_virtual(WORD CardNo, WORD bitno, WORD MapIOType, WORD MapIOIndex, double filter_time)

功 能：设置虚拟 IO 映射关系

参 数：CardNo	控制器链接号
bitno	虚拟 IO 口号，取值范围：0~15
MapIOType	虚拟 IO 映射类型： 6：通用输入端口，AxisIOInPort_IO
MapIOIndex	虚拟 IO 映射索引号：1:当虚拟 IO 映射类型设置为 6 时，此参数可 设置为 0~15 整数，表示该映射对应的具体通用输入端口号，设置成 65535 表示取消该映射关系；
filter_time	虚拟 IO 信号滤波时间，单位：s，0.001 – 2 ³¹ s

返回值：错误代码

short dmc_get_io_map_virtual(WORD CardNo, WORD bitno, WORD* MapIOType, WORD* MapIOIndex, double* filter_time)

功 能：读取虚拟 IO 映射关系设置

参 数：CardNo	控制器链接号
bitno	虚拟 IO 口号，取值范围：0~15
MapIOType	返回虚拟 IO 映射类型
MapIOIndex	返回虚拟 IO 映射索引号
filter_time	返回虚拟 IO 信号滤波时间，单位：s

返回值：错误代码

short dmc_set_emg_mode(WORD CardNo, WORD axis, WORD enable, WORD EMG_logic)

功 能：设置 EMG 急停信号

参 数：CardNo	控制器链接号
axis	轴号 0-实际轴数-1

enable 允许/禁止信号功能，0：禁止，1：允许
EMG_logic EMG 信号有效电平，0：低有效，1：高有效

返回值：错误代码

short dmc_get_emg_mode (WORD CardNo, WORD axis, WORD *enable, WORD *logic)

功 能：读取 EMG 急停信号设置

参 数：CardNo 控制器链接号
 axis 轴号 0-实际轴数-1
 enable 返回 EMG 信号功能状态
 logic 返回设置的 EMG 信号有效电平

返回值：错误代码

short dmc_set_io_dstp_mode(WORD CardNo, WORD axis, WORD enable, WORD logic)

功 能：设置减速停止信号

参 数：CardNo 控制器链接号
 axis 指定轴号 0-实际轴数-1
 enable 允许/禁止硬件信号功能，0：禁止，1：允许
 logic 外部减速停止信号有效电平，0：低有效，1：高有效

返回值：错误代码

short dmc_get_io_dstp_mode(WORD CardNo, WORD axis, WORD *enable, WORD *logic)

功 能：读取减速停止信号设置

参 数：CardNo 控制器链接号
 axis 指定轴号 0-实际轴数-1
 enable 返回 DSTP 硬件信号功能状态
 logic 返回设置的外部减速停止信号有效电平

返回值：错误代码

8.19 编码器函数

short dmc_set_encoder_unit(WORD CardNo, WORD axis, double pos)

功 能：设置指定轴的当前编码器计数值

参 数：CardNo 控制器链接号
 axis 指定轴号

pos 编码器计数值，单位：unit

返回值：错误代码

注意：总线设置指令位置 `dmc_set_position_unit` 为 0，则指令位置和编码器位置同时清 0，总线设置编码器位置 `dmc_set_encoder_unit` 为 0，则指令位置和编码器位置同时清 0；不允许单独设置指令位置为 0 或者单独设置编码器位置为 0。

`short dmc_get_encoder_unit(WORD CardNo, WORD axis, double *pos)`

功 能：读取指定轴的当前编码器计数值

参 数：CardNo 控制器链接号

axis 指定轴号

pos 返回当前编码器计数值，单位：unit

返回值：错误代码

8.20 低速位置比较

`short dmc_compare_set_config(WORD CardNo, WORD axis, WORD enable, WORD cmp_source)`

功 能：设置一维位置比较器

参 数：CardNo 控制器链接号

axis 指定轴号

enable 比较功能状态，0：禁止，1：使能

cmp_source 比较源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

`short dmc_compare_get_config(WORD CardNo, WORD axis, WORD* enable, WORD* cmp_source)`

功 能：读取一维位置比较器设置

参 数：CardNo 控制器链接号

axis 指定轴号

enable 返回比较功能状态

cmp_source 返回比较源

返回值：错误代码

`short dmc_compare_clear_points(WORD CardNo, WORD axis)`

功 能：清除已添加的所有一维位置比较点

参 数: CardNo 控制器链接号
 axis 指定轴号

返回值: 错误代码

short dmc_compare_add_point_cycle_unit(WORD CardNo,WORD axis,double pos,WORD dir,
DWORD bitno,DWORD cycle,WORD level)

功 能: 添加一维位置比较点

参 数: CardNo 控制器链接号
 axis 指定轴号
 pos 比较位置 unit
 dir 比较模式, 0: 小于等于, 1: 大于等于
 bitno 比较输出口
 cycle 周期个数
 level 比较输出点电平 (level 只有当 cycle 为 0 的时候起作用, cycle 为 0 的时候触
 发时输出 level 电平)

返回值: 错误代码

short dmc_compare_get_current_point_unit(WORD CardNo, WORD axis, double* pos)

功 能: 读取当前一维比较点位置

参 数: CardNo 控制器链接号
 axis 指定轴号
 pos 返回当前比较点位置 unit

返回值: 错误代码

注 意: 当前位置被比较之后会被清除

short dmc_compare_get_points_runned(WORD CardNo, WORD axis, long* PointNum)

功 能: 查询已经比较过的一维比较点个数

参 数: CardNo 控制器链接号
 axis 指定轴号
 PointNum 返回已经比较过的点数

返回值: 错误代码

short dmc_compare_get_points_remained(WORD CardNo, WORD axis, long* PointNum)

功 能: 查询可以加入的一维比较点个数

参 数: CardNo 控制器链接号
 axis 指定轴号
 PointNum 返回可以加入的比较点数

返回值: 错误代码

short dmc_compare_set_config_extern (WORD CardNo, WORD enable, WORD cmp_source)

功 能: 设置二维位置比较器

参 数: CardNo 控制器链接号
 enable 二维位置比较功能状态, 0: 禁止, 1: 使能
 cmp_source 二维位置比较源, 0: 指令位置计数器, 1: 编码器计数器

返回值: 错误代码

short dmc_compare_get_config_extern(WORD CardNo, WORD* enable, WORD* cmp_source)

功 能: 读取二维位置比较器设置

参 数: CardNo 控制器链接号
 enable 返回比较功能状态
 cmp_source 返回比较源

返回值: 错误代码

short dmc_compare_clear_points_extern(WORD CardNo)

功 能: 清除已添加的所有二维位置比较点

参 数: CardNo 控制器链接号

返回值: 错误代码

short dmc_compare_add_point_extern_cycle_2d(WORD CardNo, WORD* axis, double* pos, WORD* dir, WORD bitno, DWORD cycle, WORD level)

功 能: 添加二维位置比较点

参 数: CardNo 控制器链接号
 axis 指定控制器上的即将进行位置比较的轴列表(两个轴)
 pos 二维位置比较位置列表, 单位: unit
 dir 比较模式列表, 0: 小于等于, 1: 大于等于
 bitno 比较输出口
 cycle 总线周期的倍数, 设置为 0, 代表输出口保持

level 比较点输出电平,0 低电平,1 高电平。输出维持时间为总线周期*cycle
返回值: 错误代码

short dmc_compare_get_current_point_extern(WORD CardNo, double *pos)

功 能: 读取当前二维位置比较点位置

参 数: CardNo 控制器链接号

pos 返回当前二维位置比较点位置, 单位: unit

返回值: 错误代码

short dmc_compare_get_points_runned_extern(WORD CardNo, long *PointNum)

功 能: 查询已经比较过的二维比较点个数

参 数: CardNo 控制器链接号

PointNum 返回已经比较过的二维位置比较点数

short dmc_compare_get_points_remained_extern(WORD CardNo, long *PointNum)

功 能: 查询可以加入的二维比较点个数

参 数: CardNo 控制器链接号

PointNum 返回可以加入的二维位置比较点数

返回值: 错误代码

8.21 软件位置锁存

short dmc_softltc_set_mode(WORD CardNo,WORD latch,WORD ltc_enable,WORD ltc_mode,WORD ltc_inbit,WORD ltc_logic,double filter)

功 能: 配置锁存器

参 数: CardNo 控制器链接号

latch 锁存器号, 范围 0~3

ltc_enable 使能锁存器

ltc_mode 锁存模式, 0-单次锁存, 1-连续锁存

ltc_inbit 锁存触发输入信号

ltc_logic 锁存触发边沿, 0-下降沿, 1-上升沿, 2-双边沿

filter 滤波时间, 单位: us

返回值: 错误代码

short dmc_softltc_get_mode(WORD CardNo,WORD latch,WORD *ltc_enable,WORD * ltc_mode,

WORD *ltc_inbit, WORD *ltc_logic, double *filter)

功 能：读取锁存器

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3
 ltc_enable 使能锁存器
 ltc_mode 锁存模式，0-单次锁存，1-连续锁存
 ltc_inbit 锁存触发信号
 ltc_logic 锁存触发边沿，0-下降沿，1-上升沿，2-双边沿
 filter 滤波时间，单位：us

返回值：错误代码

short dmc_softltc_set_source(WORD CardNo, WORD latch, WORD axis, WORD ltc_source)

功 能：配置锁存源

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3
 axis 锁存对应轴号
 ltc_source 配置锁存源，0-指令位置，1-编码器反馈位置

返回值：错误代码

short dmc_softltc_get_source(WORD CardNo, WORD latch, WORD axis, WORD *ltc_source)

功 能：读取锁存源配置

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3
 axis 锁存对应轴号
 ltc_source 配置锁存源，0-指令位置，1-编码器反馈位置

返回值：错误代码

short dmc_softltc_reset(WORD CardNo, WORD latch)

功 能：复位锁存器

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3

返回值：错误代码

short dmc_softltc_get_number(WORD CardNo, WORD latch, WORD axis, int *number)

功 能：读取锁存个数

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3
 axis 锁存对应轴号
 number 锁存个数

返回值：错误代码

```
short dmc_softlrc_get_value_unit(WORD CardNo,WORD latch,WORD axis,double *value)
```

功 能：读取锁存值

参 数：CardNo 控制器链接号
 latch 锁存器号，范围 0~3
 axis 锁存对应轴号
 value 读取锁存值

返回值：错误代码

8.22 螺距补偿

```
short dmc_enable_leadscrew_comp(WORD CardNo, WORD axis,WORD enable);
```

功 能：设置螺距补偿的使能与禁止

参 数：CardNo 控制器链接号
 axis 指定轴号
 enable 螺距补偿使能状态，0：禁止，1：使能

返回值：错误码

```
short dmc_set_leadscrew_comp_config_unit(WORD CardNo,WORD axis,WORD n,double  
startpos,double lenpos,double *pCompPos,double *pCompNeg);
```

功 能：配置螺距补偿参数

参 数：CardNo 控制器链接号
 axis 指定轴号
 N 点数
 Startpos 补偿起始位置，单位 unit
 Lenpos 补偿段的总长度，单位 unit
 pCompPos 对应为正方向运动时，各点位置需要补偿的位置值，单位 unit
 pCompNeg 对应为负方向运动时，各点位置需要补偿的脉冲数，单位 unit

返回值：错误码

- 注 意：**
- 1) 螺距补偿最大补偿点数 256，边界点不补偿。
 - 2) 螺距补偿的长度不可设置为负值，补偿位置范围为从补偿起始位置的绝对位置往后的补偿长度。
 - 3) 补偿范围内的 N 段，正向经过补偿段时按 pCompPos 设置的参数进行补偿，负向经过补偿段时按 pCompNeg 设置的参数进行补偿。

```
short dmc_get_leadscrew_comp_config_unit(WORD CardNo,WORD axis,WORD* n,double* startpos,double* lenpos,double *pCompPos,double *pCompNeg);
```

功 能：读取螺距补偿参数

参 数：CardNo	控制器链接号
axis	指定轴号
N	返回点数
Startpos	返回补偿起始位置，单位 unit
Lenpos	返回补偿段的总长度，单位 unit
pCompPos	返回正方向运动时，各点位置需要补偿的脉冲数，单位 unit
pCompNeg	返回负方向运动时，各点位置需要补偿的脉冲数，单位 unit

返回值：错误码

```
short dmc_get_positlOn_ex_unit(WORD CardNo, WORD axis, double *pos)
```

功 能：读取指定轴的螺距补偿后的指令位置

参 数：CardNo	控制器链接号
axis	指定轴号
pos	返回当前位置值，单位：unit

返回值：错误码

8.23 龙门功能

```
short dmc_set_gear_follow_profile(WORD CardNo,WORD axis,WORD, enable,WORD master_axis,double ratio);
```

功 能：设置龙门跟随模式参数

参 数：CardNo	控制器链接号
Axis	跟随轴轴号
enable	使能状态，0：禁止，1：使能

master_axis 主轴轴号
ratIO 保留参数，设为 1

返回值：错误代码

注 意：1) 龙门功能允许一对一关系，即一个主轴对应一个从轴，允许多对一，即多个从轴一个主轴，不允许一个从轴多个主轴；
2) 龙门模式的跟随轴没有硬限位和软限位，主轴遇到限位停止时，跟随轴同样停止；
3) 如果某个轴被设置为龙门模式的跟随轴后，check done 返回一直是运动状态，直至解除该从轴与主轴龙门关系；
4) 龙门函数调用有先后顺序区分，先调用 dmc_set_gear_follow_profile 建立龙门关系，再 dmc_set_grant_error_protect 设置龙门保护误差，不可对调设置顺序，否则会报错。

```
short dmc_get_gear_follow_profile(WORD CardNo,WORD, axis,WORD* enable,WORD*  
master_axis,double* ratIO);
```

功 能：读取龙门跟随模式参数

参 数：CardNo 控制器链接号

Axis 跟随轴轴号
enable 使能状态，0：禁止，1：使能
master_axis 跟随主轴轴号
ratIO 保留

返回值：错误代码

```
short dmc_set_grant_error_protect_unit(WORD CardNo, WORD axis,WORD enable,double  
dstp_error, double EMG_error)
```

功 能：设置龙门模式主从轴编码器跟随误差停止阈值

参 数：CardNo 控制器链接号

axis 主轴轴号
enable 使能状态，0：禁止，1：使能
dstp_error 减速停止的误差阈值，单位：unit
EMG_error 立即停止的误差阈值，单位：unit

返回值：错误码

```
short dmc_get_grant_error_protect_unit(WORD CardNo, WORD axis,WORD* enable,double*  
dstp_error, double* EMG_error)
```

功 能：读取龙门模式编码器位置跟随误差停止模式设置

参 数：CardNo 控制器链接号
 axis 主轴轴号
 Enable 返回停止使能标志
 dstp_error 返回减速停止误差阈值，单位：unit
 EMG_error 返回立即停止误差阈值，单位：unit

返回值：错误码

8.24 减速停止时间设置

short dmc_set_dec_stop_time(WORD CardNo, WORD axis, double stop_time)

功 能：设置减速停止时间

参 数：CardNo 控制器链接号
 axis 指定轴号
 stop_time 减速时间，单位：s

返回值：错误代码

注 意：当发生异常停止时，如：限位信号（软硬件）被触发、减速停止信号(DSTP)被触发等进行减速停止时，减速停止时间都为 dmc_set_dec_stop_time 函数里设置的减速时间

short dmc_get_dec_stop_time(WORD CardNo, WORD axis, double *stop_time)

功 能：读取减速停止时间设置；

参 数：CardNo 控制器链接号
 axis 指定轴号
 stop_time 返回设置的减速时间，单位：s

返回值：错误代码

short dmc_set_vector_dec_stop_time (WORD CardNo, WORD Crd, double stop_time)

功 能：设置插补系减速停止时间

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 stop_time 减速时间，单位：s

返回值：错误代码

注 意：调用 dmc_stop_multicoor 进行插补系减速停止或软限位触发减速停止时，减速停止时间都为 dmc_set_dec_stop_time 函数里设置的减速时间

short ddmc_get_vector_dec_stop_time (WORD CardNo, WORD Crd, double *stop_time)

功 能：读取插补系减速停止时间设置

参 数：CardNo 控制器链接号
 Crd 坐标系号，取值范围：0~3
 stop_time 返回设置的减速时间，单位：s

返回值：错误代码

8.25 检测轴到位状态

short dmc_set_factor_error(WORD CardNo, WORD axis, double factor, long error)

功 能：设置位置误差带

参 数：CardNo 控制器链接号
 axis 指定轴号
 factor 编码器系数
 error 位置误差带，单位：pulse

返回值：错误代码

编码器系数的说明：当使用 dmc_check_success_encoder 函数检测编码器是否到位时，其用于判断的编码器位置为：**编码器计数值乘以编码器系数的值**。

short dmc_get_factor_error(WORD CardNo,WORD axis,double* factor,long* error)

功 能：读取位置误差带设置

参 数：CardNo 控制器链接号
 axis 指定轴号
 factor 返回编码器系数设置
 error 返回位置误差带设置

返回值：错误代码

short dmc_check_success_pulse(WORD CardNo, WORD axis)

功 能：检测指令到位

参 数：CardNo 控制器链接号
 axis 指定轴号

返回值：0：表示指令位置在设定的目标位置的误差带之外

 1：表示指令位置在设定的目标位置的误差带之内

注 意：1) 该函数只适用于单轴运动

- 2) 检测函数请在 `dmc_check_done` 检测到轴停止后调用，函数调用后会等待轴到位后返回，如果调用函数 100ms 内未到位，函数超时返回认为不到位

`short dmc_check_success_encoder(WORD CardNo, WORD axis)`

功 能：检测编码器到位

参 数：CardNo 控制器链接号
axis 指定轴号

返回值：0：表示编码器位置在设定的目标位置的误差带之外

1：表示编码器位置在设定的目标位置的误差带之内

注 意：1) 该函数只适用于单轴运动

- 2) 检测函数请在 `dmc_check_done` 检测到轴停止后调用，函数调用后会等待轴到位后返回，如果调用函数 100ms 内未到位，函数超时返回认为不到位。

`short dmc_set_pulse_encoder_count_error(WORD CardNo,WORD axis,WORD error)`

功 能：设置脉冲计数值和编码器反馈值之间差值的报警阈值

参 数：CardNo 控制器链接号
axis 指定轴号
error 差值报警阈值

返回值：错误代码

`short dmc_get_pulse_encoder_count_error(WORD CardNo,WORD axis,WORD *error)`

功 能：回读脉冲计数值和编码器反馈值之间差值的报警阈值

参 数：CardNo 控制器链接号
axis 指定轴号
error 返回设定报警阈值，单位：unit

返回值：错误代码

`short dmc_check_pulse_encoder_count_error(WORD CardNo,WORD axis,int* pulse_positIon, int* enc_positIon);`

功 能：检查脉冲计数值和编码器反馈值之间差值是否超过报警阈值

参 数：CardNo 控制器链接号
axis 指定轴号
pulse_positIon 返回指令位置值，单位：unit
enc_positIon 返回编码器位置值，单位：unit

返回值： 0：差值小于报警阈值，1：差值大于等于报警阈值

short dmc_set_encoder_count_error_actIOOn_config(WORD CardNo, WORD enable, WORD stopmode)

功 能：检测指令位置与编码器偏差超过报警阈值时停止运动

参 数：CardNo 控制器链接号
Enable 使能值，0：禁止，1：使能允许
Stopmode 运动停止模式，0：减速停止，1：立即停止

返回值：错误代码

short dmc_get_encoder_count_error_actIOOn_config(WORD CardNo, WORD* enable, WORD* stopmode)

功 能：回读检测指令位置与编码器偏差超过报警阈值时停止运动

参 数：CardNo 控制器链接号
Enable 回读使能值，0：禁止，1：使能允许
Stopmode 回读运动停止模式，0：减速停止，1：立即停止

返回值：错误代码

8.26 反向间隙设置

short dmc_set_backlash_unit(WORD CardNo, WORD axis, double backlash)

功 能：设置指定轴的反向间隙值

参 数：CardNo 控制器链接号
axis 轴号
backlash 反向间隙值，单位：unit

返回值：错误代码

short dmc_get_backlash_unit(WORD CardNo, WORD axis, double *backlash)

功 能：读取指定轴的反向间隙值设置

参 数：CardNo 控制器链接号
axis 轴号
backlash 返回反向间隙设置值

返回值：错误代码

8.27 PVT 运动

short dmc_ptt_table_unit(WORD CardNo, WORD axis, DWORD count, double *pTime, double *pPos)

功 能：向指定数据表传送数据，采用 PTT 模式

参 数：CardNo 控制器链接号

axis 指定轴号

count 数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime 数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos 数据点位置数组，单位：unit；数组长度：count

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_pts_table_unit(WORD CardNo, WORD axis, DWORD count, double *pTime, double *pPos, double *pPercent)

功 能：向指定数据表传送数据，采用 PTS 模式

参 数：CardNo 控制器链接号

axis 指定轴号

count 数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime 数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos 数据点位置数组，单位：unit；数组长度：count

pPercent 数据点百分比数组，百分比的取值范围：[0,100]；数组长度：count

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_pvt_table_unit(WORD CardNo, WORD axis, DWORD count, double *pTime, double

***pPos, double *pVel)**

功 能：向指定数据表传送数据，采用 PVT 模式

参 数：CardNo 控制器链接号

axis 指定轴号

count 数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime 数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos 数据点位置数组，单位：unit；数组长度：count

pVel 数据点速度数组，单位：unit/s；数组长度：count

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_pvts_table_unit (WORD CardNo, WORD axis, DWORD count, double *pTime, double *pPos, double velBegin, double velEnd)

功 能：向指定数据表传送数据，采用 PVTS 模式

参 数：CardNo 控制器链接号

axis 指定轴号

count 数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime 数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos 数据点位置数组，单位：unit；数组长度：count

velBegin 设置的第一点的速度，单位：unit/s

velEnd 设置的最后一点的速度，单位：unit/s

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_pvt_move(WORD CardNo, WORD AxisNum, WORD* AxisList)

功 能：启动 PVT 运动

参 数: CardNo 控制器链接号
 AxisNum 轴数
 AxisList 轴列表

返回值: 错误代码

8.28 看门狗功能

short dmc_set_watchdog_actIon_event(WORD CardNo, WORD event_mask)

功 能: 设置看门狗触发响应事件

参 数: CardNo: 控制器链接号; 0-7
 event_mask 看门狗触发事件: bit0 停止更新指令位置; bit1 停止轴使能; bit2

输出口复位, 包括扩展口

返回值: 错误代码

short dmc_get_watchdog_actIon_event(WORD CardNo, WORD* event_mask)

功 能: 读取看门狗触发响应事件

参 数: CardNo: 控制器链接号; 0-7
 event_mask 看门狗触发事件: bit0 停止更新指令位置; bit1 停止轴使能; bit2

输出口复位, 包括扩展口

返回值: 错误代码

short dmc_set_watchdog_enable (WORD CardNo, double timer_perIOD, WORD enable)

功 能: 使能看门狗保护机制

参 数: CardNo: 控制器链接号; 0-7
 timer_perIOD 看门狗触发时间, 单位: ms。设置为 0 看门狗功能将无效
 enable 使能标志, 0 关闭看门狗; 1 开启看门狗, 其他值报错不处理

返回值: 错误代码

short dmc_get_watchdog_enable (WORD CardNo, double * timer_perIOD, WORD* enable)

功 能: 回读看门狗状态

参 数: CardNo: 控制器链接号; 0-7
 timer_perIOD 看门狗触发时间
 enable 使能标志

返回值: 错误代码

short dmc_reset_watchdog_timer (WORD CardNo)

功 能：复位看门狗定时器

参 数：CardNo： 控制器链接号；

上位机需要在设定的时间内不断喂狗（复位看门狗定时器），以保证控制器和上位机通讯正常，否则会触发看门狗事件；

8.29 密码管理

short dmc_enter_password_ex(WORD CardNo, const char* spass)

功能：密码登录

参数：CardNo： 控制器链接号

spass： 旧密码

返回值：错误代码

注意：修改旧密码前需要调用此密码进行验证，验证成功后才可调用写密码函数写入新的密码

short dmc_write_sn(WORD CardNo, const char* new_sn)

功 能：修改密码

参 数：CardNo 控制器链接号

new_sn 新密码，密码长度不大于 255 个字符

返回值：错误代码

short dmc_check_sn(WORD CardNo, const char* check_sn)

功 能：密码校验

参 数：CardNo 控制器链接号

check_sn 旧密码

返回值：校验状态，0：失败，1：成功

注 意：1) 用户可以在系统软件开启时加入密码校验动作，以此对系统软件进行加密

2) 密码校验失败 5 次后，无法进行校验；若需再次校验，需将电脑关机，断电重启

8.30 打印输出函数

short dmc_set_debug_mode(WORD mode, const char *FileName)

功 能：函数调用打印输出设置

参 数：mode 打印输出模式，0：只打印报错函数，1：全部打印，2：全部不打印

FileName 文件保存路径:

参数文件名+后缀: 相对路径

完整描述参数文件的路径+文件名后缀: 绝对路径

返回值: 错误代码

说明: 使能打印输出后, 可监控运动函数库的调用情况。在用户调用函数时, 将输出相关信息, 并保存在指定文件路径中; **函数设置模式 2 全部不打印需配合最新动态库 (20181030 及以后动态库) 使用。**

short dmc_get_debug_mode(WORD mode, char *FileName)

功 能: 读取函数调用打印输出设置

参 数: mode 返回打印输出使能状态

FileName 返回文件保存路径

返回值: 错误代码

附 录

附录 1 常见电路接线方法

1.1 通用输出接口

1.1.1 通用输出 IO 电路

EMC 系列运动控制器具有 16 路通用输出 OUT0-OUT15，支持 24V 负载。输出采用低损耗 MOS 开漏极输出，输出电流可达 500mA，可以直接驱动小型继电器和电磁阀等外设，内部的光耦、滤波器，可以隔离干扰信号，提高系统可靠性。

1) 与电磁阀连接电路

当负载为感性器件时，在开关时感性元件的两端会产生感应电压尖峰，电感越大，开关频率越快则电压尖峰越大，有可能击穿开关 MOS 管，所以需要在感性负载上并联一个续流二极管，减小关断时电感电流的变化率降低电压尖峰，保护 MOS 管。其连接示意图如下：

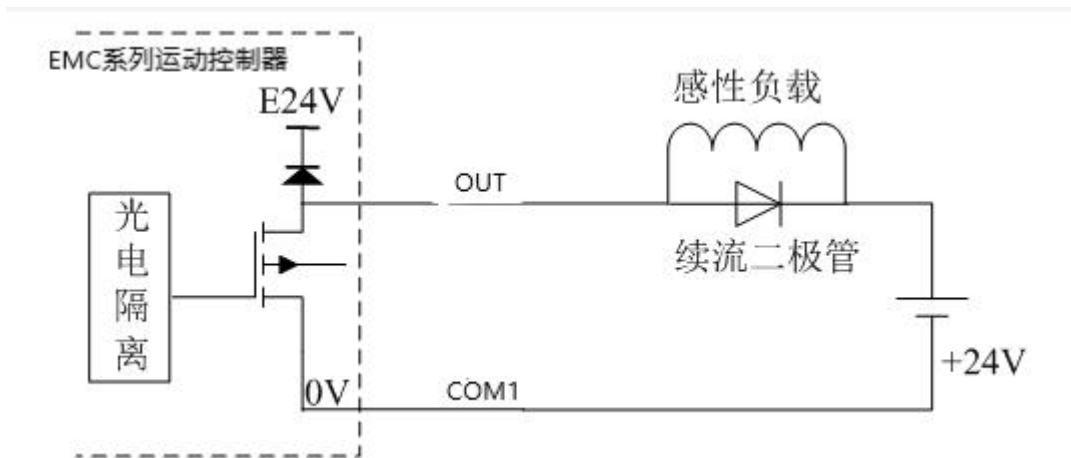


图 F1-1 控制器与电磁阀连接图

说明：+24V，0V 为外部电源和外部电源地。

注意：所有输出口使用时不能将电源端直接短接到输出口，这样会损坏内部电路结构引起控制器输出口失效。

2) 与中间继电器连接电路

控制器连接 OMRON 中间继电器 LY1J 24VDC 的电路图如图 F1-2 所示。

继电器为感性负载，其线圈外必须并联一个续流二极管，以保护控制器输出端口驱动元件。在关闭继电器的瞬间，不被线圈产生的感应电动势击穿。（OMRON 中间继电器 LY1J 线圈电压

24VDC，最大开关电压 250VAC 或 125VDC，最大开关电流 15A。)

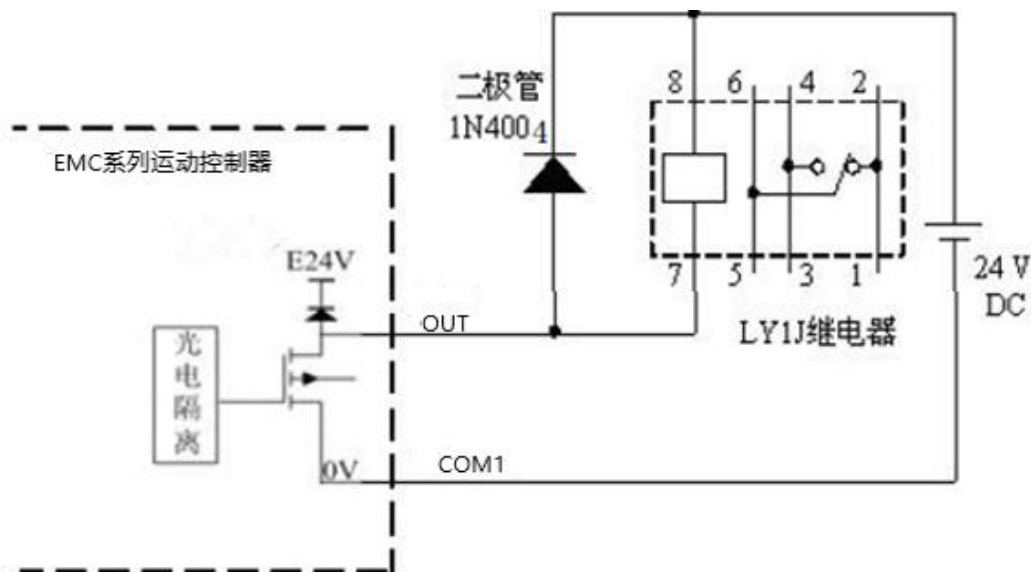


图 F1-2 控制器与中间继电器的接线图

3) 小型继电器:

继电器为感性负载，当继电器突然关断时，其电感会产生一个很大的反向电压，有可能击穿输出 MOS 管，模块内输出口有续流二极管，以保护输出口 MOS 管。继电器接线图如图 F1-3 所示:

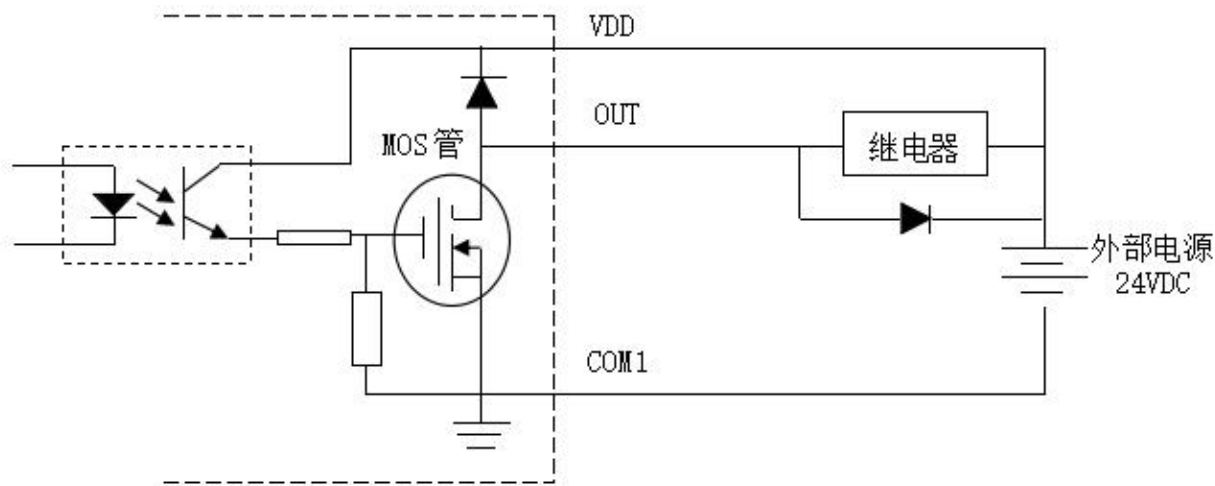


图 F1-3 通用输出接线图

⚠ 注意: 在使用通用数字输出端口时，切勿把外部电源直接接至通用数字输出端口上，否则会造成 MOS 管损坏。

1.1.2 高速输出 IO 电路

EMC-E3000/5000 系列运动控制器有 12 路高速输出 OUT0-OUT11，支持 24V 负载，其中 OUT0-OUT5，6 个高速 IO 口可以复用为 PWM 输出，PWM 使能时可复用为 PWM 输出，通用输出采用低速输出，输出电流可达 500 mA，可以直接驱动小型继电器和电磁阀等外设，内部的光耦、滤波器，可以隔离干扰信号，提高系统可靠性。接口连接方式同通用输出 IO。

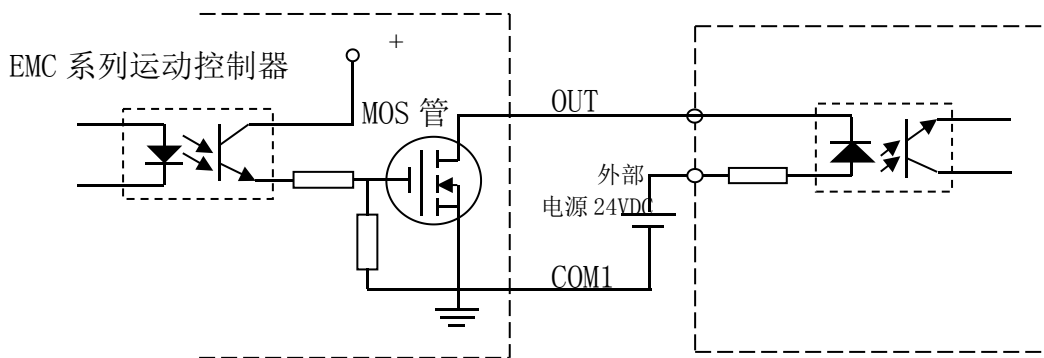


图 F1-4 高速输出接线图示

⚠注意：EMC 系列运动控制器 6 个高速输出口都可以复用为 PWM 功能。

EMC-E3000/5000 系列运动控制器具有 16 路通用输入 IN0-IN15，其中有 12 路高速输入 IN0-IN11。输入全部为光电隔离，支持 24V 电平。通用输入的接线如图 F1-5 所示。

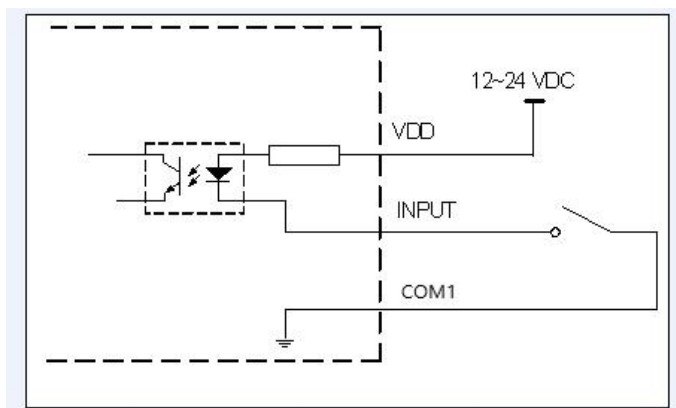


图 F1-5 通用输入接线图示

附录 2 常见错误码说明

EMC-E3000/5000 系列运动控制器的主要总线错误码和函数返回值错误码见表 F2.0 和 2.1。

表 F2.0 总线错误码

错误码（16 进制）	可能错误原因
0x0000	没有错误
0x0009	同步时钟丢失，第一个从站丢失
0x000E	总线初始化不成功，总线未连接
0x0010	总线初始化连接超时
0x0013	SDO 操作失败
0x0014	无效的 SDO 指令
0x001E	从站丢失
0x001F	读取配置文件错误，xml 文件有问题
0x0020	总线无法切换到 op 状态，总线连接有问题,可能的原因是总线没有下载配置文件
0x0022	从站的状态切换寄存器与其配置的 xml 不匹配，导致读写从站寄存器失败
0x0024	从站错误，可能的原因是从站无法切换状态机，查找从站错误码信息,即读取 0x603F 地址
0x0026	总线网络连接有问题，可能网线有干扰，导致丢数据包
0x0027	同步时钟丢失，网络时钟不稳定，1.该现象可能在总线连接过程中出现，可能运动中程序负载太大导致，可增加总线周期调试；2.网络干扰大导致
0x002A	从站无法切换到 OP 状态
0x002D	网络连接线已连接
0x0031	从站不支持 SDO 操作
0x0041	读写 SDO 超时
0x0047	该地址不支持 SDO 操作
0x0048	该地址不可读，只能写操作
0x0049	该地址不可写，只能读操作
0x004A	对象在对象字典中不存在
0x004B	不能映射为 PDO 对象
0x017B	网线断开,可能网线插错，入口与出口不匹配
0x01C2	总线的时钟发生较大的跳动，可能总线周期太小导致
0x0206	SDO 写操作失败
0x0207	SDO 读操作失败
0x020C	从站有报警出现，需求查找具体的从站错误码地址 0x603F
0x020D	从站地址无法收到数据，可能掉线等
0x0224	网络掉线，或者新加入了从站，该从站没有进入 OP 状态
0x0225	网络接线重新连上
0x0226	网络掉线，或者没有连接网线

0x0227	从站掉线
0x0229	从站状态错误，从站 AL-Status 寄存器错误，可能网络中网线被拔出后又插上
0x022B	无法连接从站地址
	其它的为错误码位乱码，是总线没有建立（即主站没有连接从站）；尤其是当控制器中存有配置文件，而没有连接任何从站时常出现。

表 F2.1 函数返回值错误码

错误码	含义
0	无错误
-1	内部的一些空指针返回值
2	动态库层参数错误
3	PCI 通讯超时
4	动态库层检测到轴处于运动中
6	连续插补错误
8	无法连接错误
9	动态库层控制器链接号错误
10	动态库层发送数据失败，请检查 PCI 槽位是否松动
12	固件文件错误
14	动态库层下载固件与控制器/器型号不匹配
17	不支持的功能，预留，不能更改
19	链接对应型号类型不匹配
20	固件参数错误
22	固件当前状态不允许操作
24	固件不支持功能，保留原版本的错误码
25	密码错误
26	密码错误输入次数受限
1000	设置的轴号超出最大轴数或伺服轴数允许范围
1001	该轴已经被其它运动占用，不支持对该轴进行操作
1002	不支持该操作，轴或坐标系处于运动中，或轴 CHECK_DOWN 未完成
1003	不支持该操作，轴或坐标系处于运动或暂停中
1004	轴映射列表为空
1005	设置脉冲计数值和编码器反馈值之间差值的报警阈值为 0
1006	设置编码器和指令位置跟踪误差使能信号错误（使能信号不是 0 或 1）
1007	设置编码器和指令位置跟踪误差停止模式错误（停止模式不是 0 或 1）
1008	设置的脉冲当量小于等于 0
1009	QUEUE 队列初始化参数错误，初始化空间大小小于 0
1010	不支持该功能
1011	SPI CRC 校验码错误
1012	控制器或控制器轴未使能，先使能轴再操作
1013	轴数超出最大轴数允许范围
1014	插补系数超出范围
1015	参数超出允许范围

1000	轴设置位置值超出范围，正负 0X7FFFFFFF
1017	轴 ALM 信号有效
1018	轴 EMG 信号有效
1019	轴 DSTP 信号有效
1020	设置超时
1021	设置超时 2
1022	中断通道超限
1023	中断配置参数有误
1024	中断逻辑电平错误
1025	处于主从轴同步运动中
1026	数字为无穷大报错，请检查是否存在除零操作(分母为零)
1027	数字为无效数据，请检查是否存在零除以零(分子和分母都为零)
1028	轴运动状态异常，需清除轴异常状态再执行该操作
1029	轴运动状态异常，需清除轴异常状态再执行该操作
1030	急停保护状态，不允许操作轴运动和 IO 输出
1100	设置轴的目标位置大于轴软件正限位位置
1101	设置轴的目标位置小于轴软件负限位位置
1102	软件负限位位置大于或等于正限位位置，不允许使能限位功能
1103	圆弧区域限位
1104	区域限位半径为 0
1105	设置的限位维数大于 2 维
1106	设置的二维限位轴号相同
1107	设置的限位停止模式大于 1（取值只能 0 和 1）
1108	设置两轴碰撞检测时，两轴轴号相同
1110	设置安全区域检测，轴数错误
1111	设置安全区域检测，轴号相同
1112	设置安全区域检测，区域数量错误
1113	设置安全区域检测，轴位置区间左界和右界干涉
1114	设置安全区域检测，轴处于该区域内，禁止运动
1200	该轴已被配置为 PWM 输出轴，不允许其它操作
1201	该轴已被配置为 PWM 输出轴，1D 和 2D 比较输出使能失败，PWM 与比较输出互斥
1202	PWM 通道超限（共支持 6 通道）
1203	PWM 跟随模式超限（0~4:5 种模式）
1204	设置 PWM 占空比大于 1
1205	设置 PWM 频率超出范围
1206	PWM 通道被 PWM 跟随占用
1300	回零目标速度小于等于 0 或大于 4M 错误
1301	回零加速度小于等于 0 错误
1302	（1）回零方向与回零偏移量冲突，正向限位回零，偏移量不能为正，负向同理；（2）回零方向与当前限位状态冲突
1303	设置回零限位反找偏移值小于 0
1304	设置回零模式超范围（回零模式小于 1 或者大于 13）

1305	设置回零偏移清零模式超限（0-回零后不做任何动作，1-回零之后，清零，然后偏移，2-回零之后，偏移，清零）
1306	设置回零模式失败，由于该回零模式下变速变位置使能信号有效
1307	原点锁存回零位移为 0
1308	驱动器 ALM 信号有效，回零启动失败
1309	轴 EMG 信号有效，回零启动失败
1310	轴 DSTP 信号有效，回零启动失败
1311	正负限位都有效
1312	轴未使能
1313	轴号超出范围
1314	回零周期设置错误，内部
1315	正在回原点
1316	轴 PDO 未配置控制模式 6060，无法启动回零
1400	TRACE 功能已经开启，请停止后再开启
1401	设置采样周期等于 0
1402	设置对象数超过限制
1403	采集功能已经启动
1404	设置的采集类型不对
1405	设置的采集长度不对
1406	读取时候的存储数组为空指针
1407	采集对象为 0
1408	采集数据已经空了
1409	采集功能内存不足
1410	采集功能已经使能
1411	采集功能未启动
1412	采集的 PDO 不存在
1450	采集的 PDO 不存在
1451	采集通道超限
1452	采集大小超限
1453	采集节点数超限
1454	从站没有 PDO 数据
1455	没有内存
1456	索引配置错误
1457	配置的索引数与索引不匹配
1458	索引中有不识别的对象
1459	从站没有配置任何的 PDO,从站不存在，2.初始化的从站信息失败
1460	配置的触发条件不存在
1461	输入的读取数据的首地址小于采集到的数据包数；2.没有成功触发采集
1462	ERR_PDO_TRACE_INPUT_NULL_POINTER= TRACR_BASE_ADDR + 62, //参数输入了空指针
1500	添加 PT 点,百分比小于 0 或大于 100
1501	添加 PT 点，添加点数超出最大允许值（1001）
1502	添加 PT 点，数据点个数超出最大允许值（1001）

1503	添加 PVT 点, 添加点数超出最大允许值 (5001)
1504	添加 PVT 点, 数据点个数超出最大允许值 (5001)
1505	启动 PVT 时, 检测轴运动模式发现不是 PVT 模式, 故报错
1506	PVT 或 PT 点数或轴号超出范围
1507	添加 PVT TABLE 时 BLOCKNUM 等于 0
1508	PVT START 失败
1509	驱动器 ALM 信号有效, PVT/PT 启动失败
1510	轴 EMG 信号有效, PVT/PT 启动失败
1511	PVT 运动轴数大于控制器最大轴数, 启动 PVT 失败
1512	没有数据
1513	时间为 0
1514	速度、加速度或者减速度小于等于 0
1515	速度小于 0
1516	与上一个位置一样
1517	方向改变了但是上一个位置的速度不为 0
1518	下载编号不匹配
1519	第一组数据必须为 0
1600	螺距补偿点数超出最大值 (256) 或小于最小值 (2)
1601	螺距补偿距离小于等于 0
1602	螺距补偿轴数超出允许范围
1603	轴脉冲补偿时间小于等于 0
1604	扩展螺距补偿表失败, 内存不足
1605	扩展螺距补偿表数据点数量错误
1606	扩展螺距补偿表未使能
1607	螺距补偿数据包 ID 错误
1608	数值错误
1700	如果不是主从轴关系, 则不予以解除龙门关系
1701	主轴已经作为从轴使用, 配置龙门主从轴关系失败
1702	从轴已经作为主轴使用, 配置龙门主从轴关系失败
1703	从轴已经作为从轴使用, 配置龙门主从轴关系失败
1704	龙门超差保护设置减速停止误差大于等于急停误差错误
1705	获取指定从轴的主轴号失败, 两轴不存在主从关系
1706	该轴为龙门主轴, 不允许操作, 请解除龙门关系后再操作
1707	主轴和从轴编码器计数模式不一致
1708	轴未使能
1709	没有进入龙门模式
1710	主轴超出范围
1711	从轴超出范围
1712	退出 GANTRY 模式, FOLLOW 模式, 必须在主轴停止状态
1713	主轴在运动中不允许退出龙门
1714	从轴 ALM 信号有效
1715	从轴 EMG 信号有效
1716	从轴 DSTP 信号有效

1800	IO 映射轴参数或 IO 类型错误
1801	IO 输入输出超出最大允许值（16）或映射轴号超出范围
1802	设置 IO 口状态值失败，已配置为一维高速比较输出,不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1803	设置 IO 口状态值失败，已配置为二维高速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1804	设置 IO 口状态值失败，已配置为一维低速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1805	设置 IO 口状态值失败，已配置为锁存输出，不允许对 IO 口进行操作，锁存输出和普通 IO 设置操作互斥
1806	获取 IO 值失败，IO 端口号超出有效范围（大于 801+16*8）
1807	设置全部输出口电平形参 PORTNO 错误（固定值为 0）
1808	设置 IO 延时输出翻转 IO 端口超出范围（大于 32 且小于 80）
1809	设置或读取轴 IO 映射轴号超出范围（大于最大轴数且不等于 15）
1810	配置或读取通用输入为轴的限位信号，输入端口超出范围（0~15）
1811	设置 IO 计数值，输入端口号超限（0~15）
1812	IO 通道实现 MS 级别的 PWM 输出器已用完
1813	IO 通道实现 MS 级别的 PWM 输出个数为 0
1814	IO 通道实现 MS 级别的 PWM 输出个数为 0
1815	IO 延时输出通道数超出范围
1816	设置 IO 口状态值失败，已配置为二维低速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1817	设置 IO 口状态值失败，已配置为 PWM 输出，不允许对 IO 口进行操作
1818	IO 计数通道超限
1819	IO 计数模式错误
1820	设置 IO 口状态值失败，已配置锁存输出输出，不允许对 IO 口进行操作
1900	DA 通道超出范围（只有 0 和 1 两个通道）
1901	设置 DA 跟随最大速度等于 0
1902	模拟量输入通道超限（支持 8 路模拟量输入）
1903	连续插补 DA 跟随占用不允许操作
1904	DA 超范围
1950	接收数据包长度超过 PCI 最大接收数据包长度
1951	上传文件指令 ID 不匹配
1952	上传文件类型不匹配
1953	上传文件 FLASH 地址为空
1954	无效参数 ID，不支持该功能
1955	无效指令 ID，不支持该功能
1956	下载固件文件太长
1957	轴运动中，下载固件失败
1958	下载固件文件 ID 改变
1959	下载文件块号错误
1960	文件大小不匹配
1961	CRC 不匹配

1962	应答数据包长度超出 PCI 最大应答数据包长度
1963	上传固件文件 ID 改变
1964	上传文件块号错误
1965	下载固件指令 ID 错误
1966	在线升级过程中增加 ARM 层校验控制器的型号，如果型号不对，则不予升级
1967	密码 ENTER 错误
1968	密码 ENTER 次数过多
1969	密码写入拒绝
1970	需要先进行密码登入
1971	数据保存位置超限
1972	下载文件过程中不允许其他命令通讯
1973	下载文件保存失败
1974	下载文件类型不匹配
1975	下载文件名不匹配
1976	下载变量文件号超过范围
1977	下载 BASIC 文件时文件还在运行中
1978	上传固件文件太长
1979	不支持上传文件
1980	下载文件名不匹配
1981	上传文件名不存在
1982	文件名错误
1983	文件太长
1984	预读取的字符长度不够
1985	下载文件不匹配，有可能是文件号和文件内容不匹配
2000	添加比较点功能参数和功能编号不匹配(或某值超出范围)
2001	比较缓冲区已满（256）
2002	比较点功能参数超出范围（0~15）
2003	比较组超出范围（最多 12 组：0~11）
2004	高速一维比较通道超出范围（4 通道：0~3）
2005	高速一维比较未使能，操作失败
2006	批量高速一维比较缓冲区满
2007	一维高速比较使能失败，已被配置为二维比较输出，1D 和 2D 互斥
2008	已经被配置为比较输出，PWM 使能失败，CMP 与 PWM 互斥
2009	比较通道超限或者轴数超限
2010	设置或读取一维比较参数配置，轴号超过最大值且不等于 255
2011	批量添加缓存比较点数大于 100
2012	高速比较缓冲区模式参数错误
2013	高速比较缓冲区模式方向参数错误
2014	高速比较时间参数设置错误
2015	高速一维比较器未使能
2016	一维比较器未使能
2017	一维比较输出口超出范围

2018	高速一维比较缓冲区已满
2100	比较组超出范围（最多 1；取值 0）
2101	二维高速比较使能失败，已被配置为一维比较输出，1D 和 2D 互斥
2102	高速二维比较配置 PWM 输出通道超限（大于等于 1 或小于 0）
2103	高速二维比较使能 PWM 输出信号有误（ENABLE>1）
2104	高速二维比较输出口设置超限（大于 15 或小于 12）
2105	添加高速二维比较点失败，FIFO 已满
2106	高速二维比较通道超出范围（1 通道：0）
2107	高速二维比较通道没有启用
2108	二维比较器未使能
2109	二维比较输出口超出范围
2110	二维比较输出口超出范围
2111	高速二维比较缓冲区已满
2150	等间距比较超出通道数
2200	LTC 锁存通道超限（只有 2 通道）
2201	LTC 设置通用输出口超限（0~15）
2202	软锁存锁存器个数超限（0~15）
2203	锁存轴为 8 时，锁存源只支持编码器反馈位置锁存
2204	读取软锁存位置值无效
2205	软锁存轴号超限（0~15）
2206	软锁存缓冲存满
2207	锁存的滤波参数超出范围 0~64MS
2208	锁存功能未生效，可能是配置文件未配置生效
2209	原点锁存参数无效
2210	EZ 锁存参数无效
2300	CAN 状态设置模式超限（0-断开；1-连接；2-复位后自动连接）
2301	CAN 状态设置 CAN 节点超限（等于 0 或大于 8）
2302	接线盒未处于 LINK 状态
2400	设置的反向间隙补偿值小于 0
2401	读取反向间隙补偿值，保存反向间隙补偿值的地址无效
2500	主轴和从轴轴号相同冲突
2501	主轴已被占用，非空闲
2502	从轴已被占用，非空闲
2503	从轴未处于电子齿轮模式
2504	主轴已经配置为电子齿轮从轴，不允许重复配置为其它从轴的主轴
2505	从轴已经配置为电子齿轮主轴，不允许重复配置为其它从轴
2506	不支持跟随主轴的运动模式，目前支持主轴为 PMOVE 和 VMOVE 两种运动模式（内部错误码，请查看相应轴停止原因了解详情）
2507	设置同步状态超出范围（0-未同步，1-同步）
2508	设置跟随位置源错误（0-跟随主轴指令位置，1-跟随主轴编码器位置）
2509	设置电子齿轮分子为 0（不能为零，可以是正数或负数）
2510	设置电子齿轮分母小于或等于 0（正数）

2511	设置从轴跟随方向限制模式超限（0-不限制，正反方向都跟随(默认) 1-正方向跟随，负方向不跟随 2-负方向跟随，正方向不跟随）
2512	设置从轴跟随模式超限（0-速度同步（默认） 1-位置和速度同步）
2513	从轴已经配置为电子齿轮模式，不允许再次配置
2514	设置从轴追赶目标速度小于等于 0 或大于 4M 错误
2515	设置电子齿轮缓存模式错误（0-打断 1-等待）
2516	从轴已经配置为电子齿轮从轴，不允许重复配置为其它主轴的从轴
2517	GEARINPOS 模式，初始化时主轴运动方向错误（内部错误码，请查看相应轴停止原因了解详情）
2518	从轴开始启动时主轴绝对位置错误（不在主轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2519	同步时主轴绝对位置错误（不在主轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2520	从轴开始运动位置大于同步时主轴绝对位置（内部错误码，请查看相应轴停止原因了解详情）
2521	同步时从轴绝对位置（不在从轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2522	从轴追赶目标速度小于同步时从轴的速度，建议调大追赶目标速度（内部错误码，请查看相应轴停止原因了解详情）
2523	从轴同步位置过小，不足以从轴从当前速度直接加速或减速至同步速度，建议调大从轴同步位置或调大加减速速度（内部错误码，请查看相应轴停止原因了解详情）
2524	从轴开始运动时主轴未达到匀速状态,建议调大从轴启动时主轴的位置（内部错误码，请查看相应轴停止原因了解详情）
2525	GEARINPOS 模式，设置从轴的最大加速度或减速度小于等于 0
2526	GEARINPOS 实际加速度大于设置的最大加速度（内部错误码，请查看相应轴停止原因了解详情）
2527	GEARINPOS 实际减速度大于设置的最大减速度（内部错误码，请查看相应轴停止原因了解详情）
2528	主轴空闲，不处于运动中，不允许在线配置电子齿轮比参数
2529	电子齿轮已经处于脱离耦合状态，操作失败
2530	从轴追赶实际可达速度大于追赶设置的最大目标速度,且以最大追赶速度运行无法实现规定时间内达到从轴的位移，建议调大追赶目标速度或减小从轴同步位置（内部错误码，请查看相应轴停止原因了解详情）
2531	从轴同步速度在只有加速或减速过程也不可达速度,建议调大从轴从轴同步位置或调小齿轮比或调大从轴开始运动的主轴位置（内部错误码，请查看相应轴停止原因了解详情）
2532	二分法求取合适的可达速度，迭代次数超过限制次数也无法找到合适的速度值（内部错误码，请查看相应轴停止原因了解详情）
2533	主轴 PMOVE 目标位置小于主轴同步位置（内部错误码，请查看相应轴停止原因了解详情）

2534	主轴 PMOVE 目标位置小于从轴开始运动时的主轴位置（内部错误码，请查看相应轴停止原因了解详情）
2535	从轴追赶过程加速周期或减速周期为 0，有可能主轴同步位置和从轴开始运动时的主轴位置相隔太近（内部错误码，请查看相应轴停止原因了解详情）
2536	主轴目标速度为零（内部错误码，请查看相应轴停止原因了解详情）
2537	从轴实际可达速度大于追赶速度时，重算以追赶速度为目标速度的匀速周期数小于等于零（内部错误码，请查看相应轴停止原因了解详情）
2538	反转禁止仅对模数轴有效，若禁止反转，而从轴不是模数轴则报错
2539	GEARINPOS 位置同步模式不支持在线变倍率
2540	GEARINPOS 主轴位移与规划结果不一致
2541	GEARINPOS 变倍率失败，输入主轴和从轴不属于主从轴关系
2542	上一次变倍率尚未处理完成，禁止再次触发变倍率
2543	从轴起始反向运动方向模式错误(1~4)
2544	从轴起始反向运动加速度为零
2545	回读主轴位置源设置错误：主轴轴号与设置主轴号不匹配
2546	电子齿轮从轴未处于使能状态，建立主从轴齿轮关系失败
2547	设置从轴的加速度小于等于 0
2548	设置从轴的减速度小于等于 0
2549	设置从轴的 JERK 小于 0
2550	设置的电子齿轮主轴与当前执行齿轮运动的主轴不匹配，执行电子齿轮失败
2551	EXECUTE 执行位超范围报错
2552	GEARINPOS 设置追赶阶段 X 轴或 Y 轴的位移量等于零
2553	主轴正向运动，从轴启动追赶时主轴位移小于等于 0 错误
2554	主轴正向运动，从轴同步位置小于等于从轴当前位置
2555	主轴正向运动，主轴同步位置小于等于主轴当前位置
2556	主轴正向运动，主轴同步位置小于等于从轴启动时主轴的位置
2557	主轴负向运动，从轴启动追赶时主轴位移大于等于 0 错误
2558	主轴负向运动，从轴同步位置大于等于从轴当前位置
2559	主轴负向运动，主轴同步位置大于等于主轴当前位置
2560	主轴负向运动，主轴同步位置大于等于从轴启动时主轴的位置
2600	凸轮轴未使能
2601	轴已经处于凸轮模式
2602	没进到凸轮模式
2603	主轴超出范围
2604	从轴超出范围
2605	主从轴同轴
2606	没有配置主轴
2607	添加的凸轮数据个数超出范围
2608	插入的凸轮数据有错误
2609	插入的凸轮数据表已满
2610	添加 CAM TABLE 时 BLOCKNUM 等于 0

2611	凸轮表编号超出允许范围
2612	凸轮表挺杆编号超出允许范围
2613	挺杆点列表已空，删除凸轮表挺杆点失败
2614	挺杆点列表已满，添加凸轮表挺杆点失败
2615	挺杆点列表找不到指定挺杆点信息(挺杆列表为空或者找不到匹配的挺杆编号)，读取挺杆点信息失败
2616	凸轮表凸轮点编号超出允许范围
2617	凸轮点列表找不到指定凸轮点信息(凸轮点列表为空或者找不到匹配的凸轮点编号)，读取指定凸轮点信息失败
2618	主轴位置小于 0，写凸轮信息失败
2619	从轴位置小于 0，写凸轮信息失败
2620	从轴速度小于 0，写凸轮信息失败
2621	从轴加速度小于 0，写凸轮信息失败
2622	挺杆正向经过模式设置超范围(0~3)
2623	挺杆负向经过模式设置超范围(0~3)
2624	挺杆的主轴位置小于 0，添加或写挺杆信息失败
2625	凸轮点数太少或太多(小于 3 或者大于 2048)
2626	指令模块类型错误，设置模块状态失败
2627	主轴缩放比例小于等于 0，CAM_IN 模块报错
2628	主轴缩放比例等于 0，CAM_IN 模块报错
2629	启动位置模式超范围，CAM_IN 模块报错
2630	啮合动作方式超范围，CAM_IN 模块报错
2631	从轴速度小于等于 0，CAM_IN 模块报错
2632	从轴加速度小于等于 0，CAM_IN 模块报错
2633	从轴减速度小于等于 0，CAM_IN 模块报错
2634	主轴位置来源超范围，CAM_IN 模块报错
2635	交接模式超范围，CAM_IN 模块报错
2636	挺杆点主轴位置大于主轴周期位置最大值，写挺杆点信息失败
2637	挺杆点主轴位置大于主轴周期位置最大值，增加挺杆点信息失败
2638	从轴加加速度小于 0，CAM_IN 模块报错
2639	启动啮合时，主轴位置超出主轴凸轮周期的最大值范围
2640	启动啮合时，主轴位置超出主轴的模允许范围
2670	旋切编号超限，范围：[1,8]
2671	旋切轴半径小于等于 0
2672	旋切刀头数量小于等于 0
2673	旋切进给轴半径小于等于 0
2674	旋切长度小于等于 0
2675	旋切同步开始位置小于等于 0
2676	旋切同步结束位置小于等于 0
2677	旋切刀具同步位移大于材料切割长度
2678	旋切组已被占用，请使用其它旋切组编号
2679	找不到指定旋转轴的旋切组编号，旋切组未激活或旋切轴有误
2680	旋切退出曲线结束时 X 轴位移大于切割长度

2701	输入参数报错为空指针，内部使用
2702	内存分配不成功，内部使用
2703	输入的跟随序号超过最大数限制
2704	轴未处于跟随模式
2705	条件不允许退出跟随模式
2706	条件不允许进入跟随模式
2707	参数错误
2708	跟随参数必须在插补系停止的时候设置
2709	跟随不支持
2710	跟随参数必须在插补系启动之前的时候设置
2721	输入参数报错为空指针，内部使用
2722	输入的跟随类型不对
2723	输入参数有错误
2724	状态机切换不正确，可能参数未设置，或者未使能
2741	输入参数报错为空指针，内部使用
2742	如果不是主从轴关系，则不予以解除
2743	检查从轴是否已经作为主轴使用
2744	误差检测值中 DSTP_ERROR 大于 EMG_ERROR 错误
2745	轴跟随功能没启用
2761	输入参数报错为空指针，内部使用
2762	PWM 通道未使能
2763	占空比小于 0 或者大于 1
2764	跟随模式超出范围
2765	跟随通道超出范围
2781	输入参数报错为空指针，内部使用
2782	内存分配不成功
2783	输入的跟随序号超过最大数限制
2801	与 RTCP 功能互斥
2802	没有配置参数
2803	维数错误
2811	跟随功能没有开启
2821	没有配置参数
2822	设置间隔为 0
2823	未使能
2824	等间距比较超出通道数
2825	等间距比较输出口超出范围
3000	目标速度小于等于零错误
3001	点位运动位移为零或规划时运动距离为零
3002	轴状态未处于完成或空闲状态，启动点位运动无效
3003	PMOVE 规划或设置的起跳速度小于 0
3004	PMOVE 规划或设置目标速度等于 0
3005	PMOVE 规划或设置结束速度小于 0
3006	正向硬件限位触发，不允许正向启动 PMOVE

3007	负向硬件限位触发，不允许负向启动 PMOVE
3008	轴 ALM 信号有效，启动 PMOVE 失败
3009	正向软件限位触发，不允许正向启动 PMOVE
3010	负向软件限位触发，不允许负向启动 PMOVE
3011	轴处于空闲或不处于 PMOVE 模式，强行变位操作失败
3012	在线变速失败，新的目标速度小于 0
3013	设置点位起跳速度大于 4M 频率
3014	设置点位目标速度大于 4M 频率
3015	设置点位结束速度大于 4M 频率
3016	设置软启动结束速度大于 4M 频率
3017	设置软启动目标速度大于 4M 频率
3018	该轴已配置为 PWM 输出轴，PMOVE ENTER 失败，PMOVE 与 PWM 轴互斥
3019	PMOVE 运动已经结束，不能进行在线变位操作
3020	当前处于回零运动模式，不允许该操作
3021	软着陆第一段 PMOVE 结束速度为零，导致第二段 PMOVE 无法规划
3022	软着陆第二段目标位置小于第一段目标位置，导致第二段 PMOVE 反向运动，容易引起危险
3023	轴 EMG 信号有效，启动 PMOVE 失败
3024	当前运动模式不是 VMOVE 和 PMOVE，不支持在线变速操作
3025	当前运动模式不是 PMOVE，不支持同时变速和变位速操作
3026	内部规划缓冲区出错
3027	软着陆第二段方向错误
3028	软着陆中加速度或者减速度为 0
3029	当前轴运动模式为空闲时，调用 VMOVE 和 PMOVE 在线变速报错
3030	当前处于软启动或软着陆运动，PMOVE 在线变速报错
3031	当前处于软启动或软着陆运动，PMOVE 在线变位报错
3032	PMOVE 平滑时间大于 1 秒报错
3033	设置的加速时间等于 0
3034	设置的减速时间等于 0
3035	设置的加速度小于等于 0
3036	设置的减速度小于等于 0
3037	设置软启动第二段目标速度小于第一段目标速度
3038	软着陆设置的停止速度必须小于最大速度
3039	软着陆设置的停止速度必须大于 0
3040	软着陆第一段运动距离为零
3041	软启动第二段目标位置小于或等于第一段目标位置，导致第二段 PMOVE 反向运动，容易引起危险
3042	软启动第一段运动距离为零
3043	设置的加速时间大于 100S
3044	设置的减速时间大于 100S
3045	设置的平滑时间大于加速或减速时间
3046	多轴同步点位运动轴数超范围(2~12)

3047	多轴同步点位运动轴号相同冲突错误
3048	多轴同步点位运动,轴规划总时间为零错误
3049	多轴同步点位运动,设置轴运动参数错误: 未找到相应轴号
3050	加/减加速度为零
3100	正向硬件限位触发, 不允许正向启动 VMOVE
3101	负向硬件限位触发, 不允许负向启动 VMOVE
3102	轴 ALM 信号有效, 启动 VMOVE 失败
3103	正向软件限位触发, 不允许正向启动 VMOVE
3104	负向软件限位触发, 不允许负向启动 VMOVE
3105	轴 EMG 信号有效, 启动 VMOVE 失败
3150	正弦波频率限制
3151	正弦波单周期速度限制
3152	正弦波振幅错误
3153	正向硬件限位触发, 设置 SINE 曲线参数失败
3154	负向硬件限位触发, 设置 SINE 曲线参数失败
3155	轴 ALM 信号有效, 设置 SINE 曲线参数失败
3156	正向软件限位触发, 设置 SINE 曲线参数失败
3157	负向软件限位触发, 设置 SINE 曲线参数失败
3158	轴 EMG 信号有效, 设置 SINE 曲线参数失败
3159	轴 DSTP 信号有效, 设置 SINE 曲线参数失败
3206	轴未使能
3208	总线配置状态
3209	轴号超出范围
3210	轴选超出范围
3211	倍选超出范围
3300	门型正在运动
3301	轴未使能
3302	轴正在运动
3303	轴号超出范围
3304	插补系超出范围
3305	轴 ALM 信号有效
3306	轴 EMG 信号有效
3307	轴 DSTP 信号有效
3308	正向硬件限位触发
3309	负向硬件限位触发
3310	正向软件限位触发
3311	负向软件限位触发
3312	第二段运动与第一段方向不一致或第二段距离为 0, 启动 M_MOVE 失败
3313	没有开始速度规划
3314	指令缓存正在运动中, 不允许相关的操作
3315	超过指令缓存最大限制轴数
3316	指令缓存当前已打开
3317	超过指令缓存组数(2 组)

3318	指令缓存没有打开
3319	压入的点位运动指令的轴数为 0
3320	压入的指令类型不存在
3321	触发条件不存在
3322	触发的 IO 条件超过最大个数（5）限制
3323	触发的轴条件超过最大个数（3）限制
3324	IO 口操作超过最大个数限制
3325	指令轴没有在 OPEN 时使能
3326	指令轴重复输入
3327	指令 BUFFER 已满
3328	指令 BUFFER 异常停止
3329	设置的反馈位置误差参数错误
3330	该模式下不允许删除指令
3331	没有对应得删除的 ID 存在
3332	指令缓存中的压入数据异常
3333	指令缓存输入指令为空指针
3500	单段插补模块已经初始化
3501	单段插补模块内存开辟失败
3502	单段插补模块内坐标系忙，不能清除模块
3503	单段插补模块指针为空，不能清除模块
3504	单段插补模块坐标系状态不为空闲，不能进行参数配置
3505	单段插补模块坐标系号超过最大值
3506	单段插补模块坐标系维度配置错误
3507	单段插补模块速度规划参数配置错误
3508	单段插补模块路径长度为 0
3509	单段插补模块路径类型不支持
3510	单段插补模块圆弧三点在一条直线上
3511	单段插补模块圆弧点坐标相同
3512	单段插补模块圆弧终点+半径模式，半径为 0 或小于两点的距离的一半
3513	单段插补模块圆弧半径为 0
3514	单段插补模块圆弧圈数小于 0
3515	单段插补模块圆弧 3 维空间参数模式错误
3516	单段插补模块圆弧终点半径方式，圆弧参数错误，可能在同一直线上
3517	单段插补模块圆弧计算空间向量函数输入参数指针为 0
3518	单段插补模块圆弧计算空间向量错误
3519	单段插补模块椭圆维度错误
3520	单段插补模块椭圆 A/B 轴长度错误
3521	单段插补模块椭圆起点或终点不在曲线上
3522	单段插补模块椭圆数据坐标相同
3523	单段插补模块坐标系不处于空闲状态，不能启动插补
3524	单段插补模块坐标系不处于完成状态，不能复位坐标系
3525	单段插补模块圆弧圈数超出范围
3526	单段插补交接模式设置超范围

3527	单段插补过渡模式设置超范围
3528	单段插补缓冲区已满
3529	单段插补缓冲区已空
3530	单段插补模块 FIFO 内存开辟失败
3531	单段插补模块 FIFO 已经初始化
3532	指针指向地址为空错误
3600	点位运动库轴数错误
3601	点位运动库缓冲区长度错误
3602	点位运动库插补周期错误
3603	点位运动库内存获取失败
3604	点位运动库库空间释放失败
3605	点位运动库没有初始化
3606	点位运动库运动类型不支持
3607	点位运动库运动缓存模式错误
3608	点位运动库缓冲区满
3609	点位运动库轴对象为空
3610	点位运动库轴处于 DONE 状态，不能添加指令
3611	点位运动库当前指令优先级较低，无法添加
3612	点位运动库处于边界情况，无法添加指令
3613	点位运动库执行 STOP 指令，禁止添加指令
3614	点位运动库轴不处于 DONE 状态，无法 RESET
3615	点位运动库叠加运动 OFF，不能执行动作
3616	PAUSE 状态时不能压入 HALT 指令
3617	点位运动库缓冲区指令缓存模式为目标速度衔接
3702	内存不足
3703	源数据点数量错误
3704	参数向量已经构建
3705	节点向量已经构建
3706	样条构建模式不支持
3707	轨迹模式不支持
3708	已达到最小控制点数量，样条构建失败
3709	超过最大控制点数量限制，样条构建失败
3710	已达到最大迭代次数，样条构建失败
3711	没有开启前瞻模式
3712	PCI 数据传输数量错误
3713	PCI 数据控制点数量错误
3714	PCI 数据节点向量数据错误
3715	PCI 数据样条长度为 0
3716	PCI 数据样条维度错误
3717	缓冲区内内存不足
3718	缓冲区链表节点中的样条指针不为空
3719	样条轨迹禁止变倍率
3720	样条轨迹禁止暂停

3721	样条轨迹起点和上一个点不衔接
3722	贝塞尔过渡参数检查, 两条曲线没有公共连接点
3723	贝塞尔过渡参数检查, 三点共线
3724	贝塞尔过渡参数检查, 半径圆弧模式输入信息不正确
3725	贝塞尔过渡参数检查, 圆心圆弧模式输入信息不正确
3726	贝塞尔过渡参数检查, 不支持的输入圆类型: (可使用类型: 三点圆弧, 圆心半径)
3727	贝塞尔过渡参数检查, 半径圆心模式, 没有输入圆弧运动方向
3728	贝塞尔过渡参数检查, 半径圆心模式, 没有输入圆弧运动平面
3729	贝塞尔过渡插补阶段, 反算参数 T 出现错误
3730	贝塞尔计算直线交点出现错误
4000	坐标系插补有限状态不在有限状态机所列状态范围
4001	插补坐标系处于空闲状态时, START_LIST 操作无效
4002	插补坐标系处于空闲状态时, PAUSE_LIST 操作无效
4003	插补坐标系处于就绪状态时, OPEN_LIST 操作无效
4004	插补坐标系处于就绪状态时, PAUSE_LIST 操作无效
4005	插补坐标系处于运动状态时, OPEN_LIST 操作无效
4006	暂停或减速停, 停止过程未完成, 不允许调用重启运动
4007	插补坐标系处于暂停状态时, OPEN_LIST 操作无效
4008	插补坐标系处于空闲状态, 不允许操作 (添加指令)
4009	插补坐标系处于运动状态时, 不允许操作插补模式
4010	插补坐标系处于运动状态时, 不允许操作设置参数
4011	插补坐标系处于运动状态时, 不允许重复启动
4012	插补坐标系处于关闭状态时, 不允许操作 (添加指令)
4013	插补坐标系处于暂停中状态时, 不允许暂停
4014	插补坐标系处于单段插补模式, OPEN_LIST 操作无效
4015	插补坐标系处于连续插补中, 不能进行单段插补
4016	插补坐标系不处于允许的状态中
4017	插补坐标系处于异常状态状态, 操作无效, 需先清除插补系异常状态后再操作
5000	设置插补系超出最大插补系允许范围
5001	设置插补目标速度大于 4M 频率
5002	设置螺旋线封闭模式失败 (模式取值只能 0 和 1)
5003	插补维数和插补轴数不匹配
5004	坐标系尚未建立, 轨迹插补计算失败
5005	插补维数超出范围 (大于最大轴数或小于 2)
5006	缓冲区未打开
5007	STARTLIST 失败, 由于上一次设置的速度倍率为 0
5008	初始化模块, 动态分配前瞻和插补 FIFO 失败
5009	在线变倍率设置的倍率超出范围值 (0~2)
5010	脉冲当量小于等于 0
5011	设置的目标速度小于等于 0
5012	设置连续插补段速度比例超限 (0~2)

5013	缓存区处于异常状态，不允许操作
5014	缓存区空间已满，不允许操作
5015	插补路径矢量位移为零
5016	圆弧插补三点共线，无法求取圆弧参数
5017	终点+半径圆弧插补模式，起点和终点重合错误
5018	终点+半径圆弧插补模式，半径为 0 或小于两点的距离的一半
5019	增强圆弧插补，半径小于等于 0
5020	增强圆弧插补，旋转角度为零或者大于 360 度
5021	终点半径的圆弧插补方式，圆弧参数错误，可能在同一直线上
5022	添加轨迹轴列表轴号存在相同轴号错误
5023	添加轨迹轴号不在 OPENLIST 轴列表内
5024	增强圆弧插补不支持前瞻模式
5025	半径+终点圆弧插补，圈数为负值暂不支持
5026	圆心+终点圆弧插补，圈数为负值，暂不支持同心圆插补
5027	仅支持 T 形速度规划模式，设置速度插补模式有误，不支持变速
5028	由于变倍率引起的暂停或减速停，停止过程未完成，不允许调用变倍率
5029	上一次变倍率未处理完，不允许再次调用变倍率
5030	当前段处于减速阶段，不允许调用变倍率操作
5031	轨迹暂停后，存在点位运动轴未恢复至初始位置情况报错
5032	坐标系不处于暂停状态，不允许调用暂停返回运动
5033	上一次变倍率当前段变速未处理完，不允许再次调用变倍率
5034	在线变倍率未处理完，不允许调用轨迹暂停命令
5035	运动模式错误，轨迹暂停后恢复初始位置运动失败
5036	设置切向跟随时，被跟随的轨迹模式错误（模式仅支持圆弧和椭圆）
5037	设置切向跟随时，被跟随轴角度当量小于 0
5038	设置切向跟随时，跟随坐标系处于运动中，不允许设置切向跟随
5039	设置切向跟随时，跟随轴处于运动中
5040	解除切向跟随时报错，该轴处于运动中，不允许解除跟随功能
5041	解除切向跟随时，该轴不处于切向跟随模式，解除报错
5042	读取切向跟随参数失败，该轴不处于切向跟随模式或已被解除切向跟随
5043	该版本固件连续轨迹功能模块禁止，不支持连续轨迹功能
5044	运动中或暂停时禁止设置前瞻参数
5045	刀向跟随量设置值为 0
5046	插补坐标系不处于运动状态时，不允许操作在线变倍率
5047	平面圆弧插补前两轴或者空间圆弧插补前三个轴不在主轴范围内
5048	圆弧插补参数错误
5049	当前段尚未规划，不响应在线变倍率请求
5050	插补系运动重启未完成，不响应当次在线变倍率功能
5051	连续轨迹控制节点 IO 编号超出范围：0-本地 IO 1~8-扩展 IO
5052	设置单轴最大插补速度小于等于零错误
5053	设置单轴最大插补速度大于 4M
5054	圆心+角度圆弧插补参数无效错误
5055	直线插补在线变位置时，配置的轴参数不对

5100	速度规划函数出错
5101	圆弧限速速度小于等于 0
5102	设置规划参数错误，起始或结束速度小于 0，或者目标速度小于等于 0，或者加速度小于等于 0
5103	设置的单轴最大速度小于等于 0
5104	设置单段插补位移长度小于或等于 0
5105	设置的加速时间等于 0
5106	设置的减速时间等于 0
5107	置的前瞻加速度等于 0
5108	设置的最大加速度等于 0
5109	设置的平滑停止时间等于 0
5110	设置速度规划模式超限（0-T 形,1-S 形）
5111	设置的平滑时间大于 1 秒
5112	规划起始速度小于 0
5113	规划目标速度小于等于 0
5114	规划结束速度小于 0
5115	当前段剩余周转期不足以支持当前段变速
5116	当前段处于结束或者空闲阶段不支持当前段变速
5117	当前段速度规划模式错误，不支持当前段变速
5118	当前段新目标速度达不到，不支持当前段变速
5119	当前段剩余距离不足，不支持当前段变速
5120	设置的加速度等于 0
5121	设置的减速度等于 0
5122	设置的加速时间超出范围
5123	设置的减速时间超出范围
5124	规划起始速度超出范围
5125	规划目标速度超出范围
5126	规划结束速度超出范围
5127	坐标系暂停过程，插补轴点位运动操作类型超出范围（0~3）
5128	设置的 S 段时间大于加速或减速时间
5129	点胶位置超出当前段总长度，获取规划时间失败
5130	前瞻模式，用户自定义点胶位置超出当前段总长度，超出的点胶分配至下一段插入的圆弧轨迹
5200	设置插补维数超出范围（大于最大轴数）
5201	径+终点圆弧插补模式，半径为零
5202	不支持该圆弧插补模式，模式超限
5203	半径+终点圆弧插补模式，弦长大于直径错误（终点和起点的距离大于直径）
5204	3 维空间圆弧插补圆弧参数模式错误（模式为终点+半径模式）
5205	计算空间圆弧的转换矩阵，输入参数地址无效
5206	计算空间圆弧的转换矩阵，单位向量计算结果有误
5207	椭圆插补，长半轴或端半轴长度小于 0
5208	起点不在椭圆上

5209	终点不在椭圆上
5210	圆弧圆心终点模式，圆心坐标与起点相同
5211	连续轨迹直线段和圆弧自动分段段距小于等于 0
5212	连续轨迹直线段和圆弧自动分段未使能
5213	连续轨迹直线段和圆弧自动分段缓存已满
5214	连续轨迹类型错误，目前仅支持直线段和圆弧自动分段
5215	连续轨迹直线段和圆弧自动分段段数大于缓存剩余空间
5216	自动分段点胶或者用户自定义批量点胶位置，IO 号配置超出允许范围 (0~15)
5217	自动分段点胶或者用户自定义批量点胶位置，IO 控制模式超出允许范围 (0~4,6)
5218	不支持的插补模式，模式超限
5300	LIST 已经打开
5301	缓冲区未打开，关闭缓冲区操作无效
5302	圆弧插补获取插补平面时，插补轴数小于 2 错误
5303	圆弧插补获取插补平面时，插补轴数与插补轴列表维数不一致
5304	圆弧插补获取插补平面时，插补轴不在插补轴列表内
5305	设置环形缓冲区状态不在有限状态范围以内
5306	设置引起 RINGBUFFER 停止源错误 (0-减速停引起，1-暂停引起)
5307	设置环形缓冲区异常状态错误 (0-正常，1-减速停，2-急停)
5308	插补系号或者 IO 索引号超出范围
5309	插补轴正向硬件限位触发，OPENLIST 失败
5310	插补轴负向硬件限位触发，OPENLIST 失败
5311	插补轴轴 ALM 信号有效，OPENLIST 失败
5312	插补轴正向软件限位触发，OPENLIST 失败
5313	插补轴负向软件限位触发，OPENLIST 失败
5314	插补轴 EMG 信号有效，OPENLIST 失败
5315	工件坐标系超出范围
5316	旋转加速度或最大速度小于等于 0
5317	读取插补系工件坐标系号和使能报错，未使能或工件坐标系号为空
5318	插补轴正向硬限位有效，无法启动单段圆弧插补
5319	插补轴负向硬限位有效，无法启动单段圆弧插补
5320	插补轴正向软限位有效，无法启动单段圆弧插补
5321	插补轴负向软限位有效，无法启动单段圆弧插补
5322	插补轴正向硬限位有效，无法启动单段直线插补
5323	插补轴负向硬限位有效，无法启动单段直线插补
5324	插补轴正向软限位有效，无法启动单段直线插补
5325	插补轴负向软限位有效，无法启动单段直线插补
5350	工件坐标系超出范围
5351	旋转加速度或最大速度小于等于 0
5352	读取插补系工件坐标系号和使能报错，未使能或工件坐标系号为空
5353	开启五轴功能，OPEN LIST 维度需要为 5
5354	开启五轴功能，不能设置轮廓误差

5355	开启五轴功能，不能设置轮廓误差
5356	错误的五轴机器类型
5357	传递了空参数指针
5358	插补坐标系忙，不能改变功能
5359	插补器下标错误
5360	工件歪斜设置值错误，角度大于等于 360
5361	五轴插补系超出限制范围
5400	轴组编号超出有效值允许的范围
5401	连接组编号超出有效值允许的范围
5402	支持最大连接组数已满，不能再绑定主从关系
5403	主从轴轴号相同，不能再绑定主从关系
5404	多组主从轴组之间轴号冲突，不能再绑定主从关系
5405	未找到与指定主轴绑定的从轴
5406	轴组未使能
5407	轴组已使能或处于运动中
6000	模块初始化错误
6001	不支持
6002	参数错误
6003	缓存空
6004	缓存满
6005	错误索引
6006	缓存段无效
6010	插补系号错误
6011	插补系没有打开
6012	插补系已经打开
6013	插补系已经关闭
6014	插补系不能暂停
6015	插补系正在运行
6016	插补系正在暂停
6020	插补 IO 缓冲区已满
6021	插补段已被中断使用
6022	该模式下不能设置该功能
6023	运动中不运行更改参数
6024	参数不在有效范围
6025	主要的 LIST 没有初始化
6026	另外一个 LIST 已经启动
6027	主要 LIST 没有启动
6028	强制变位段号不匹配
6030	插补维数超限
6031	轴数不在有效范围
6032	插补轴号一样的
6033	轴不在 OPEN 时的列表中
6034	轴映射表为空

6035	映射轴错误
6036	映射轴忙
6037	轴的使能信号关闭，不能启动
6038	轴软件限位信号有效
6039	轴硬件限位信号有效
6040	轴减速停止信号有效
6041	轴急停信号有效
6042	轴报警信号有效
6043	轴已经被插补系占用
6044	轴可能被移位且未回位
6045	轴没有用 PMOVE 移动过，不支持回位操作
6050	半径为 0 或小于两点的距离的一半
6051	半径方式起点和终点重合
6052	圆弧轴不在 XYZ 里面
6053	圆弧三点在一条直线上
6054	圆弧圈数小于 0
6055	圆弧圈数超出范围
6056	样条曲线的起点位置错误，不与实际启动时刻位置一致
6060	S 曲线加减速模式，平滑时间为零出错
6061	起跳速度小于零
6062	最大速度小于等于零
6063	终点速度小于零
6064	规划长度小于等于零
6065	最小匀速时间小于零
6066	规划模式非 T 型非 S 形
6067	加速度小于等于零
6068	减速度小于等于零
6069	规划长度小于等于零
6070	参数变量空指针
6071	参数变量 2 空指针
7002	奇异点，无法到达
7003	输入建模参数报错
7004	输入运动范围无法到达
7010	通用参数错误
7011	机械臂未配置
7012	机械臂用户坐标系未配置
7013	机械臂工具坐标系未配置
7014	用户坐标系建立失败
7015	工具坐标系建立失败
7016	机械被使能关闭
7017	机械臂类型错误
7018	机械臂序号错误
7019	机械臂用户坐标系错误

7020	机械臂工具坐标系错误
7021	臂型错误
7022	关节数量错误
7023	关节映射错误
7024	结构参数错误
7025	超出工作空间
7026	关节限位触发
7027	关节限速触发
7028	关节限加速度触发
7029	关节限速触发
7030	关节限加速度触发
7031	机械臂处于奇异点
7032	雅克比行列式值为 0
10000	总线报错
10001	总线连接的从站数量超过最大允许的从站数
10002	总线轴映射关系错误
10003	从站节点地址不存在
10004	总线轴号不存在
10005	输入的总线轴号错误
10006	总线 IO 口号不存在
10007	总线函数中输入的指针为空
10008	总线配置文件中的从站类型错误
10009	ETHERCAT 从站的函数输入参数无效
10010	CANOPEN 从站的函数输入参数无效
10011	总线函数操作 SDO 时输入参数无效
10012	总线输入的端口号无效
10013	总线未做映射配置
10014	CANOPEN 总线超过最大允许从站数
10015	从站超过最大允许从站数
10016	主站序号输入错误
10017	总线输入文件大小超过最大限制
10018	总线输入文件为空文件
10019	总线输入文件格式错误
10020	总线映射文件信息错误
10021	总线配置 ENI 文件错误
10022	总线轴运行模式错误
10023	总线类型输入错误
10024	总线运行 PP 模式错误
10025	总线 PDO 输入参数错误,长度类型超过 2 或者等于 0
10026	总线周期参数错误
10027	总线 PDO 缓存满
10028	运动中不允许扫描
10029	运动中不允许设置周期

10030	运动中不允许下载 ENI 文件
10031	运动中不允许下载配置文件
10032	总线错误不能运动
10033	PP 运动非 ETHERCAT 轴
10034	轴未使能
10035	轴已经使能
10036	轴不能使能
10037	转矩控制下转矩值与运动方向不匹配
10038	超出总线扩展模拟量通道数
10039	总线 IO 口号不存在
11000	SDO 报错，需要加上后面的 SDO 具体错误码
12000	回零高速对象 0X6099 01 的 SDO 报错码偏移值，加上 SDO 错误码
13000	回零低速 0X6099 02 的 SDO 报错码偏移值，加上 SDO 错误码
14000	回零加速度 0X609A 00 的 SDO 报错码偏移值，加上 SDO 错误码
15000	回零偏移 0X607C 00 的 SDO 报错码偏移值，加上 SDO 错误码
16000	回零模式 0X6098 00 的 SDO 报错码偏移值，加上 SDO 错误码

附录 3 常用 PDO 对象表

表 F3.0 PDO 对象

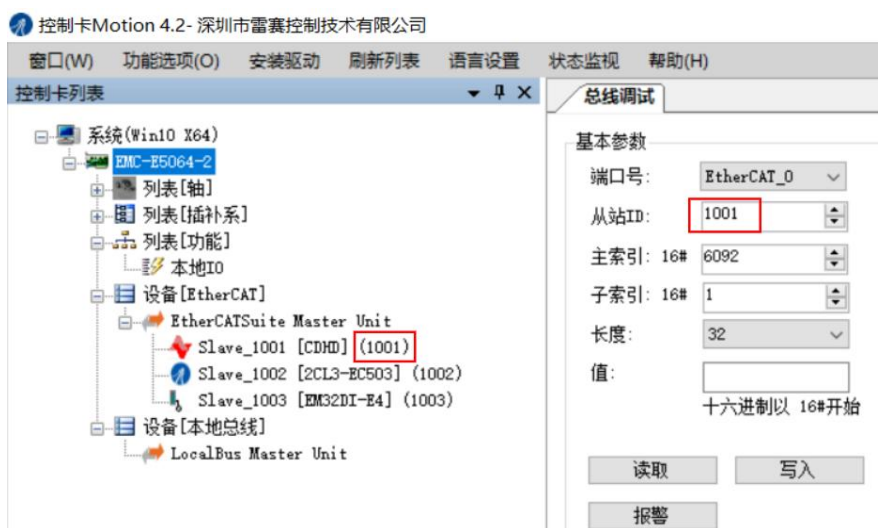
主索引 (HEX)	子索引 (HEX)	对象名称	数据类型	PDO 映射
603F	00	错误码(Error Code)	UINT16	T_PDO
6040	00	控制字(Controlword)	UINT16	R_PDO
6041	00	状态字(Statuword)	UINT16	T_PDO
6060	00	操作模式(Modes of operatIOn)	INT8	T_PDO
6061	00	操作模式显示(modes of operatIOn display)	INT8	R_PDO
6064	00	实际位置(positiOn actual value)	INT32	T_PDO
606C	00	速度实际值 (Velocity actual value)	INT32	T_PDO
607A	00	目标位置(Target positiOn)	INT32	R_PDO
60FD	00	数字输入(Digital inputs)	UINT32	T_PDO
60FE	01	物理输出开启(Physical outputs)	UINT32	R_PDO
60FE	02	物理输出使能(Bit mask)	UINT32	R_PDO
607C	00	原点偏移(Home offset)	INT32	NO
6098	00	回零方式(Homing method)	INT8	NO
6099	01	回零高速(Speed during search for switch)	UINT32	NO
6099	02	回零低速(Speed during search for zero)	UINT32	NO
609A	00	回零加速度(Homing acceleratIOn)	UINT32	NO
6092	01	每转脉冲数(Feed)	UINT32	NO
6071	00	目标转矩(Target touque)	INT16	R_PDO
6072	00	最大转矩(Max touque)	UINT16	R_PDO
6077	00	转矩实际值(Touque actual value)	INT16	T_PDO
6080	00	最大电机速度(Max motor speed)	UINT32	R_PDO
6087	00	转矩变化率(Touque slope)	UINT32	R_PDO
60B2	00	转矩偏移值(Touque offset)	INT16	R_PDO

以上对象字典可通过 SDO 方式读取和设置值。SDO 方式读取和设置可直接通过雷赛 Motion 软件的总线调试界面操作。总线调试界面进入方式，打开 Motion 后，右击“控制器列表区”的控制器 EMC-E3000/5000 ->单击“总线调试”菜单，进入总线调试窗口。



附图 3.1 总线调试界面

进入总线调试界面后，可以通过 SDO 方式读写各从站的对象字典。SDO 方式不允许频繁读写对象字典。下面以读取 EMC-E5640 连接的第一个从站（CDHD 驱动器）的每转脉冲数为例，预操作的对象字典主索引 6092，子索引 1，长度 UINT32，从站 ID 为 1001（第一个从站 ID 为 1001，依次往后排列），点击“读取”按钮，就可以读取到每转脉冲数，此驱动器设置为 10000。写入过程数据同理。



附图 3.2 对象字典操作

如果需频繁读写某个对象字典，则只能通过扩展 PDO 的方式操作，不能通过 SDO 方式操作。比如，预读取驱动器的到位（INP）信号，通过查看 CAN402 协议手册，我们知道对象字典 6041 状态字(Statuword)的 bit10 位为 INP 位。需通过 Motion 软件将 6041 状态字(Statuword)添加到扩展 PDO，操作过程，进入 Motion 后，在“控制器列表区”的“设备[EtherCAT]”->双击主站“EtherCATSuite Master Unit”菜单，进入主站配置界面，点击“主站”列表->点击“扩展 TxPDO”列表，进入扩展 TxPDO 编辑界面。映射类型选择“EtherCAT”，映射从站选择预操作的

从站 ID，映射变量选择状态字 Statuword，确认后下载配置文件到控制器，完成操作。最后通过调用扩展 PDO 的读取写入 API 函数监控修改值。Motion 软件也提供读取设置的操作界面，进入总线调试界面的扩展 PDO 操作。扩展 PDO API 操作函数如表 F3.0。

表 F3.0 扩展 PDO API 函数

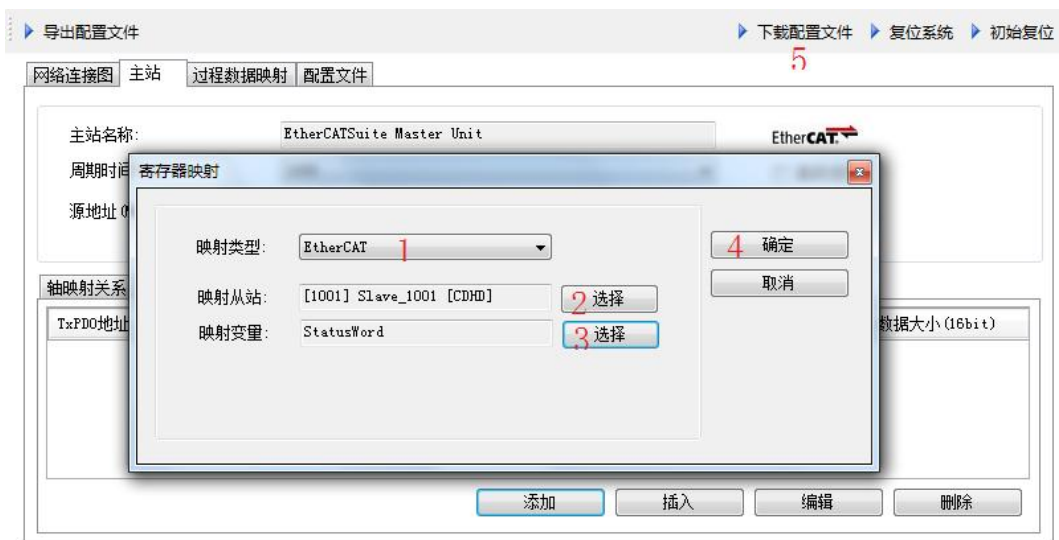
nmc_write_rxpdo_extra	设置从站扩展无符号 RxPDO 值
nmc_read_rxpdo_extra	读取从站扩展有符号 RxPDO 值
nmc_read_txpdo_extra	读取从站扩展有符号 TxPDO 值

Bit	Description	M / O
0	Ready to switch on	M
1	Switched on	M
2	Operation enabled	M
3	Fault	M
4	Voltage enabled	M
5	Quick stop	M
6	Switch on disabled	M
7	Warning	O
8	Manufacturer specific	O
9	Remote	M
10	Target reached	M
11	Internal limit active	M
12 - 13	Operation mode specific	O
14 - 15	Manufacturer specific	O

附图 3.3 状态字 6041 的 bit 位定义



附图 3.4 主站配置界面



附图 3.5 扩展 TxPDO 编辑界面

扩展PDO

地址:	0	RPDO读取
长度:	0	RPDO写入
值:		TPDO读取

十六进制以 16#开始

附图 3.6 扩展 PDO 读写界面

附录 4 运动控制函数索引

函数分类	函数名	描 述	索引
控制器设置函数	dmc_board_init_eth	控制器初始化函数	8.1 节
	dmc_board_reset	控制器硬件复位函数	
	dmc_board_close	控制器关闭函数	
	dmc_get_CardInfList	获取控制器硬件 ID 号	
	dmc_get_card_versIOn	获取控制器硬件版本号	
	dmc_get_card_soft_versIOn	获取控制器固件版本号	
	dmc_get_card_lib_versIOn	获取控制器动态库文件版本号	
	dmc_get_total_axes	获取当前控制器的轴数	
	dmc_download_configfile	下载参数文件	
	dmc_download_firmware	下载固件文件	
脉冲当量与限位设置函数	dmc_set_equiv	设置指定轴的脉冲当量	8.2 节
	dmc_get_equiv	读取指定轴的脉冲当量	
	dmc_set_softlimit_unit	设置软限位	
	dmc_get_softlimit_unit	读取设置的软限位	
回原点运动函数	nmc_set_home_profile	设置 EtherCAT 总线轴回零参数	7.3 节
	nmc_get_home_profile	设置 EtherCAT 总线轴回零参数	
	nmc_home_move	启动 EtherCAT 总线轴回零	
	dmc_get_home_result	读取回零结果	
限位开关设置函数	dmc_set_softlimit_unit	设置软限位	8.2 节
	dmc_get_softlimit_unit	读取软限位设置	
位置计数器控制函数	dmc_set_positIOn_unit	设置指令脉冲位置	8.15 节
	dmc_get_positIOn_unit	读取指令脉冲位置	
运动状态检测及控制函数	dmc_read_current_speed_unit	读取单轴的速度值	8.16 节
	dmc_read_vector_speed_unit	读取插补运动的矢量速度	
	dmc_check_done	检测指定轴的运动状态	
	dmc_check_done_multicoor	检测坐标系的运动状态	
	dmc_axis_IO_status	读取指定轴有关运动信号的状态	
	dmc_stop	指定轴停止运动	
	dmc_stop_multicoor	停止坐标系内所有轴的运动	
	dmc_EMG_stop	紧急停止所有轴	
单轴运动速度曲线设置函数	dmc_set_profile_unit	设置单轴运动速度曲线	8.3 节
	dmc_get_profile_unit	读取单轴运动速度曲线	
	dmc_set_s_profile	设置单轴速度曲线 S 段参数值	
	dmc_get_s_profile	读取单轴速度曲线 S 段参数值	
单轴运动函数	dmc_pmove_unit	指定轴点位运动	8.4 节
	dmc_vmove	指定轴连续运动	

函数分类	函数名	描 述	索引
	dmc_change_speed_unit	在线变速	
	dmc_reset_target_positIon_unit	在线变位	
	dmc_update_target_positIon_unit	强行变位（在线/非在线）	
	dmc_t_pmove_extern_unit	软着陆	
	dmc_pmove_extern_unit	指令整合 PMOVE	
	dmc_t_pmove_extern_softstart_unit	软启动	
	dmc_update_target_positIon_extern_unit	强制变位并实现软着陆	
通用输入输出 IO 函数	dmc_read_inbit_ex/ dmc_read_inbit_ex_ex	读取指定控制器的某一位输入口的电平状态	8.17 节
	dmc_write_outbit	设置指定控制器的某一位输出口的电平状态	
	dmc_read_outbit/ dmc_read_outbit_ex	读取指定控制器的某一位输出口的电平状态	
	dmc_read_inport/ dmc_read_inport_ex	读取指定控制器的全部输入口的电平状态	
	dmc_read_outport/ dmc_read_outport_ex	读取指定控制器的全部输出口的电平状态	
	dmc_write_outport	设置指定控制器的全部输出口的电平状态	
	dmc_reverse_outbit	IO 输出延时翻转	
	dmc_set_IO_count_mode	设置 IO 计数模式	
	dmc_get_IO_count_mode	读取 IO 计数模式设置	
	dmc_set_IO_count_value	设置 IO 计数值	
	dmc_get_IO_count_value	读取 IO 计数值	
编码器函数	dmc_set_encoder_unit	设置指定轴编码器反馈位置脉冲计数值	8.20 节
	dmc_get_encoder_unit	读取指定轴编码器反馈位置脉冲计数值	
软件位置锁存函数	dmc_softlrc_set_mode	配置锁存器	8.23 节
	dmc_softlrc_get_mode	读取锁存器配置	
	dmc_softlrc_set_source	设置锁存源	
	dmc_softlrc_get_source	读取锁存源	
	dmc_softlrc_get_value_unit	读取锁存值	
	dmc_softlrc_get_number	读取已锁存个数	
	dmc_softlrc_reset	复位指定控制器的锁存器的标志位	
低速位置比较函数	dmc_compare_set_config	设置一维位置比较器	8.21 节
	dmc_compare_get_config	读取一维位置比较器设置	
	dmc_compare_clear_points	清除一维位置比较点	
	dmc_compare_add_point_cycle_unit	添加一维位置比较点	

函数分类	函数名	描 述	索引
	dmc_compare_get_current_point_unit	读取当前一维比较点位置	
	dmc_compare_get_points_runned	查询已经比较过的一维比较点个数	
	dmc_compare_get_points_remained	查询可以加入的一维比较点个数	
螺距补偿	dmc_enable_leadscrew_comp	设置螺距补偿的使能与禁止	8.25 节
	dmc_set_leadscrew_comp_config_unit	配置螺距补偿参数	
	dmc_get_leadscrew_comp_config_unit	读取配置的螺距补偿参数	
	dmc_get_positlOn_ex_unit	读取指定轴的螺距补偿后的指令位置	
龙门功能	dmc_set_gear_follow_profile	设置龙门跟随模式参数	8.26 节
	dmc_get_gear_follow_profile	读取龙门跟随模式参数	
	dmc_set_grant_error_protect_unit	设置龙门模式主从轴编码器跟随误差停止阈值	
	dmc_get_grant_error_protect_unit	读取龙门模式编码器位置跟随误差停止阈值	
减速停止时间设置	dmc_set_dec_stop_time	设置减速停止时间	8.27 节
	dmc_get_dec_stop_time	读取减速停止时间设置	
	dmc_set_vector_dec_stop_time	设置插补系减速停止时间	
	dmc_get_vector_dec_stop_time	读取插补系减速停止时间设置	
检测轴到位状态函数	dmc_set_factor_error	设置位置误差带	8.28 节
	dmc_get_factor_error	读取位置误差带设置	
	dmc_check_success_pulse	检测指令到位	
	dmc_check_success_encoder	检测编码器到位	
密码管理函数	dmc_enter_password_ex	密码登录	8.31 节
	dmc_write_sn	修改密码	
	dmc_check_sn	密码校验	
打印输出函数	dmc_set_debug_mode	函数调用打印输出设置	8.32 节
	dmc_get_debug_mode	读取函数调用打印输出设置	
状态检测	dmc_get_axis_run_mode	读取轴运动模式	8.16 节
	dmc_get_stop_reason	读取轴停止原因	
	dmc_clear_stop_reason	清除轴停止原因	
插补速度设置函数	dmc_set_vector_profile_unit	设置插补运动速度曲线	8.5 节
	dmc_get_vector_profile_unit	读取插补运动速度曲线	
	dmc_set_vector_s_profile	设置插补运动速度曲线的平滑时间	
	dmc_get_vector_s_profile	读取插补运动速度曲线的平滑时间	
插补运动函数	dmc_line_unit	直线插补运动	8.6 节
	dmc_arc_move_center_unit	基于圆心圆弧扩展的螺旋线插补运动（可作两轴圆弧插补）	
	dmc_arc_move_radius_unit	基于半径圆弧扩展的圆柱螺旋线插补运动（可作两轴圆弧插补）	
	dmc_arc_move_3points_unit	基于三点圆弧扩展的圆柱螺旋线插补运动（可作两轴及三轴圆弧插补）	

函数分类	函数名	描 述	索引
连续插补运动	dmc_conti_open_list	打开连续插补缓冲区	8.7 节
	dmc_conti_start_list	开始连续插补	
	dmc_conti_close_list	关闭连续插补缓冲区	
	dmc_conti_pause_list	暂停连续插补	
	dmc_conti_stop_list	停止连续插补	
	dmc_conti_delay	连续插补中暂停延时指令	
	dmc_conti_line_unit	连续插补中直线插补指令	
	dmc_conti_arc_move_center_unit	连续插补中基于圆心圆弧扩展的螺旋线插补指令（可作两轴圆弧插补）	
	dmc_conti_arc_move_radius_unit	连续插补中基于半径圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）	
	dmc_conti_arc_move_3points_unit	连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴或三轴圆弧插补）	
	dmc_conti_rectangle_move_unit	连续插补中矩形插补指令	
	dmc_conti_pmove_unit	连续插补中控制指定外轴运动指令	
连续插补缓冲区检测	dmc_conti_remain_space	查询连续插补缓冲区剩余插补空间	8.8 节
	dmc_conti_read_current_mark	读取连续插补缓冲区当前插补段号	
连续插补小线段前瞻功能	dmc_conti_set_lookahead_mode	设置连续插补前瞻参数	8.9 节
	dmc_conti_get_lookahead_mode	读取连续插补前瞻参数	
连续插补 IO 控制	dmc_conti_set_pause_output	设置连续插补暂停及异常停止时 IO 输出状态	8.10 节
	dmc_conti_get_pause_output	读取连续插补暂停及异常停止时 IO 输出状态设置	
	dmc_conti_wait_input	连续插补等待 IO 输入	
	dmc_conti_delay_outbit_to_start	连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）	
	dmc_conti_delay_outbit_to_stop	连续插补中相对于轨迹段终点 IO 滞后输出	
	dmc_conti_ahead_outbit_to_stop	连续插补中相对于轨迹段终点 IO 提前输出（段内执行）	
	dmc_conti_accurate_outbit_unit	连续插补中精确位置 CMP 输出控制	
	dmc_conti_write_outbit	连续插补中缓冲区立即 IO 输出	
	dmc_conti_clear_IO_actIOon	清除段内未执行完的 IO 动作	
圆弧限速	dmc_set_arc_limit	设置圆弧限速参数	8.12 节
	dmc_get_arc_limit	读取圆弧限速参数	
连续插补位置跟随	dmc_conti_gear_unit	连续插补中刀向跟随	8.13 节
PWM 输出	dmc_set_pwm_enable/	设置 PWM 使能状态	8.14 节

函数分类	函数名	描 述	索引
	dmc_set_pwm_enable_extern		
	dmc_get_pwm_enable/ dmc_get_pwm_enable_extern	读取 PWM 使能状态设置	
	dmc_set_pwm_output	设置 PWM 输出	
	dmc_get_pwm_output	读取 PWM 输出设置	
	dmc_set_pwm_onoff_duty	设置 PWM 开关状态对应的占空比	
	dmc_get_pwm_onoff_duty	读取 PWM 开关状态对应占空比的设置	
	dmc_conti_set_pwm_output	连续插补中 PWM 输出设置	
	dmc_conti_set_pwm_follow_speed	连续插补中 PWM 速度跟随	
	dmc_conti_get_pwm_follow_speed	读取 PWM 速度跟随参数设置	
	dmc_conti_delay_pwm_to_start	连续插补中相对于轨迹段起点 PWM 滞后输出	
	dmc_conti_ahead_pwm_to_stop	连续插补中相对于轨迹段终点 PWM 提前输出	
	dmc_conti_write_pwm	连续插补中缓冲区立即 PWM 输出	
反向间隙设置	dmc_set_backlash_unit	设置反向间隙值	8.29 节
	dmc_get_backlash_unit	读取反向间隙值设置	
PVT 运动函数	dmc_ptt_table_unit	向指定数据表传送数据, 采用 PTT 模式	8.30 节
	dmc_pts_table_unit	向指定数据表传送数据, 采用 PTS 模式	
	dmc_pvt_table_unit	向指定数据表传送数据, 采用 PVT 模式	
	dmc_pvts_table_unit	向指定数据表传送数据, 采用 PVTS 模式	
	dmc_pvt_move	启动 PVT 运动	
EtherCAT 总线操作函数	nmc_set_fieldbus_error_switch	设置总线掉线后是否能操作从站	第 8 章
	nmc_get_fieldbus_error_switch	读取设置的总线掉线后是否能操作从站	
	nmc_set_node_od	设置从站对象字典参数值	
	nmc_get_node_od	读取从站对象字典参数值	
	nmc_set_axis_enable	使能 EtherCAT 总线驱动器	
	nmc_set_axis_disable	失能 EtherCAT 总线驱动器	
	nmc_get_total_axes	读取 EtherCAT 总线轴和虚拟轴轴数	
	nmc_get_total_adcnum	读取 EtherCAT 总线 AD/DA 输入输出口数	
	nmc_get_total_IOnum	读取 EtherCAT 总线 IO 输入输出口数	
	nmc_set_controller_workmode	设置控制器工作模式	
	nmc_get_controller_workmode	读取控制器工作模式	

函数分类	函数名	描 述	索引
	nmc_get_cycletime	读取 EtherCAT 总线循环周期	
	nmc_get_axis_type	读取轴类型	
	nmc_stop_etc	停止 EtherCAT 总线	
	nmc_get_consume_time_fieldbus	读取 EtherCAT 总线平均周期时间、最大周期时间、执行周期数	
	nmc_clear_consume_time_fieldbus	清除 EtherCAT 总线平均周期时间、最大周期时间、执行周期数的记录	
	nmc_get_axis_state_machine	读取 EtherCAT 总线轴状态机	
	nmc_get_axis_controlmode	读取 EtherCAT 总线轴控制模式	
	nmc_set_home_profile	设置 EtherCAT 总线轴回零参数	
	nmc_get_home_profile	读取 EtherCAT 总线轴回零参数	
	nmc_home_move	启动 EtherCAT 总线轴回零	
	nmc_get_errcode	读取 EtherCAT 总线状态	
	nmc_get_axis_node_address	读取 EtherCAT 轴节点地址信息	
	nmc_get_total_slaves	获取 EtherCAT 从站总数	
	nmc_write_outbit_extern	设置 EtherCAT 扩展模块的某个输出端口的电平	
	nmc_read_outbit_extern	读取 EtherCAT 扩展模块的某个输出端口的电平	
	nmc_read_intbit_extern	读取 EtherCAT 扩展模块的某个输入端口的电平	
	nmc_read_outport_extern	读取 EtherCAT 扩展模块指定 IO 组号的全部输出口的的电平	
	nmc_read_inport_extern	读取 EtherCAT 扩展模块指定 IO 组号的全部输入口的电平	

附录 5 常见问题解决方法

序号	问题描述	可能原因及解决方法
1	控制器无法扫描到设备	a. 检查各网口接线是否松动
		b. 检查驱动器是否上电、有无报错，若报错先断电重启，执行热复位再扫描
		c. 控制器是否正常工作，可复位系统完成后再扫描
2	驱动器无法使能	a. 该驱动器是否正常通讯，扫描设备看能否扫描到
		b. 该驱动器是否报错，若报错先断电重启再使能
3	控制器连接多台 ETHERCAT 驱动器时，扫描到设备的数量不对	a. 检查各网口接线是否松动
		b. 检查驱动器是否上电、有无报错，若报错先断电重启，执行热复位再扫描
		c. 控制器是否正常工作，可冷复位系统完成后再扫描
4	PC 机不能和控制器通讯	检查网线是否有松动，控制器 RUN 灯是否正常； 参考软件手册检查应用软件是否编写正确。
5	控制器已经正常工作，但电机不转动	检查驱动器和电机之间的连接是否正确。可以使用 Motion 软件进行测试。 确保驱动器工作正常，没有出现报警。
6	电机可以转动，但工作不正常	检查控制器和驱动器是否正确接地，抗干扰措施是否做好。
7	能够控制电机，但电机出现振荡或是过冲	可能是驱动器参数设置不当，检查驱动器参数设置； 应用软件中加减速时间和运动速度设置不合理。
8	锁存数据的重复精度差	检查函数调用； 程序中是否进行了去抖动处理； 触发信号的设定。
9	数字输入信号不能读取	接线是否正常；检查函数调用。
10	数字输出信号不正常	接线是否正常；检查函数调用。
11	电脑休眠后找不到控制器	刷新设备管理器或重启电脑



深圳市雷赛控制技术有限公司
SHENZHEN LEADSHINE CONTROL TECHNOLOGY CO.,LTD

深圳市雷赛控制技术有限公司

地 址：深圳市南山区沙河西路南山智谷产业园 B 栋 15 楼

邮 编：518055

电 话：0755-26415968

传 真：0755-26417609

Email: info@szleadtech.com.cn

网 址: <http://www.szleadtech.com.cn>