深圳市雷赛控制技术有限公司
SHENZHEN LEADSHINE CONTROL TECHNOLOGY CO.,LTD

雷赛控制技术 DMC-E3032 V2.0

运动控制卡用户使用手册

2019.07.01

版 权 说 明

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

1.1 DMC-E3032 运动控制卡的特点

DMC-E3032运动控制卡是深圳市雷赛控制技术有限公司开发出具有自主知识产权的新型高性能EtherCAT总线运动控制卡。该控制卡在具备通用的EtherCAT总线功能的基础上，还拥有强大的运动功能。在总线接口方面，可通过EtherCAT总线协议组成网络，通信速率可达到100Mbps。其主要特点如下：

1）接线方便，通过网线连接EtherCAT总线伺服驱动器、EtherCAT总线IO模块；
2）总线负载能力强，最快可达250us，轴扩展方便，最多可扩展多达32轴；
3）支持对称或非对称梯形、S形速度曲线控制；
4）支持在线变速、变位功能；
5）支持位置控制、速度控制；
6）支持2 ~ 16轴直线插补；
7）支持2轴圆弧插补、空间圆弧插补、螺旋线插补；
8）支持高速位置锁存、比较、高速二维位置比较；
9）支持EtherCAT总线过程数据缓存功能；
10）支持EtherCAT总线群组化运动控制；

1.2 DMC-E3032 运动控制卡主要技术指标

表 1.1 DMC-E3032 主要技术指标

技术指标 \ 卡类型	DMC-E3032
支持在PC机中同时工作的卡数	8
单卡可扩展电机轴数	32
总线通讯速率	最大100Mbps
支持的总线周期	250us（16轴）、500us（32轴）、1ms（32轴）、2ms（32轴）
支持的插补坐标系个数	6
辅助编码器信号输入个数	2
辅助编码器计数器长度	32位有符号
辅助编码器输入信号频率	4 MHz （4倍频后为16MHz）
通用数字输入口数量	8（可扩展）

技术指标 \ 卡类型	DMC-E3032
通用数字输出口数量	8（可扩展）
通用数字输入口	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）
通用数字输入口最高响应频率	4 kHz
通用数字输出口	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）
高速位置锁存输入口数量（LTC）	4
高速位置比较输出口数量（CMP）	6
过程数据采集单个数据包	≤ 48 Bytes
过程数据采集总计数据包	≤ 48000 Bytes
群组化功能支持轴数	32
群组化功能单次数据点	100
群组化功能数据点缓存	1000
工作温度	0～50 ℃
贮存温度	-20～80 ℃
湿度	5～85 %，非结露
PCI总线插槽电源（输入）	+5VDC±5% ，最大1100 mA
外部电源（输入）	24VDC±5% ，10A
尺寸大小（mm）	182.00(长)×106.68(高)
驱动程序	支持Windows XP、Windows 7（32bit和64bit） Window10（32bit和64bit）
运动控制函数库	支持VC、VB6.0、C#、VB.NET、LabVIEW、Delphi等多种语言
调试软件	控制卡 Motion 软件

1.3 DMC-E3032 运动控制卡的典型应用

DMC-E3032运动控制卡可应用于各行各业自动化设备中。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）绕线机等。

其典型运动控制系统结构图如下：

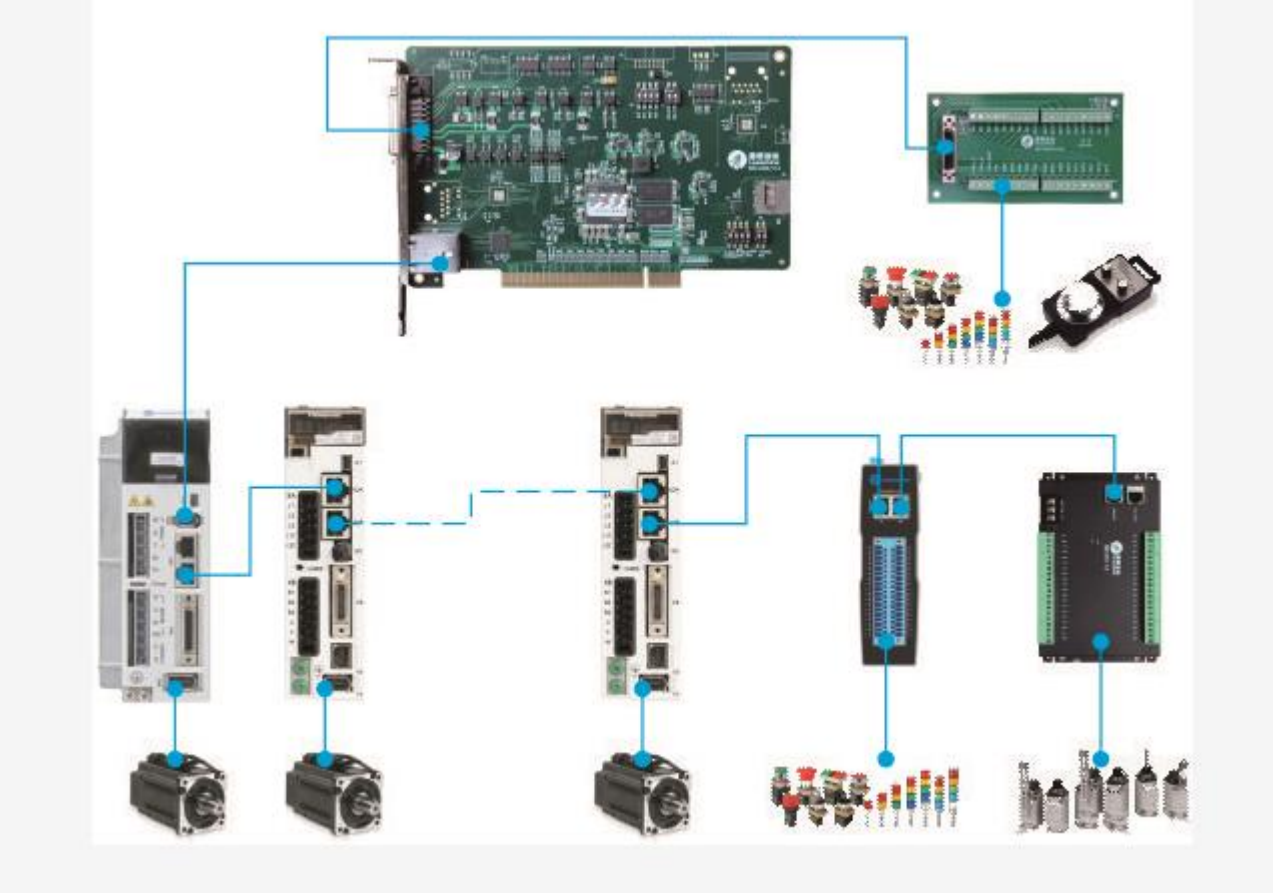


图1.1 DMC-E3032运动控制卡组成的运动控制系统结构图

1.4 订货信息

表 1.2 DMC-E3032 运动控制卡及主要配件订货代码

	订货代码	名称	数量	备注
配置 1	80.00.09.010460	运动控制卡 DMC-E3032-A08	1	无扩展接线板及连接电缆（无本地 IO、编码器输入、位置比较输出及手轮接口）
配置 2	80.00.09.010470	运动控制卡 DMC-E3032-A12	1	
配置 3	80.00.09.010480	运动控制卡 DMC-E3032-A16	1	
配置 4	80.00.09.010490	运动控制卡 DMC-E3032-A24	1	
配置 5	80.00.09.010450	运动控制卡 DMC-E3032-A32	1	
配置 6	80.00.09.010460	运动控制卡 DMC-E3032-A08	1	
	80.15.03.012060	接线板 ACC36-IO	1	

	10.25.03.010328	CABLE36-NR-20(36Pin 长 2m HPCN 带状插头)	1	带扩展接线板及连接电缆（有本地 IO、编码器输入、位置比较输出及手轮接口）
配置 7	80.00.09.010470	运动控制卡 DMC-E3032-A12	1	
	80.15.03.012060	接线板 ACC36-IO	1	
	10.25.03.010328	CABLE36-NR-20(36Pin 长 2m HPCN 带状插头)	1	
配置 8	80.00.09.010480	运动控制卡 DMC-E3032-A16	1	
	80.15.03.012060	接线板 ACC36-IO	1	
	10.25.03.010328	CABLE36-NR-20(36Pin 长 2m HPCN 带状插头)	1	
配置 9	80.00.09.010490	运动控制卡 DMC-E3032-A24	1	
	80.15.03.012060	接线板 ACC36-IO	1	
	10.25.03.010328	CABLE36-NR-20(36Pin 长 2m HPCN 带状插头)	1	
配置 10	80.00.09.010450	运动控制卡 DMC-E3032-A32	1	
	80.15.03.012060	接线板 ACC36-IO	1	
	10.25.03.010328	CABLE36-NR-20(36Pin 长 2m HPCN 带状插头)	1	

1.5 产品图片

运动控制卡 DMC-E3032卡的外观示意图及相关配件如图 1.2和图1.3 所示。

图1.2 DMC-E3032运动控制卡外观示意图

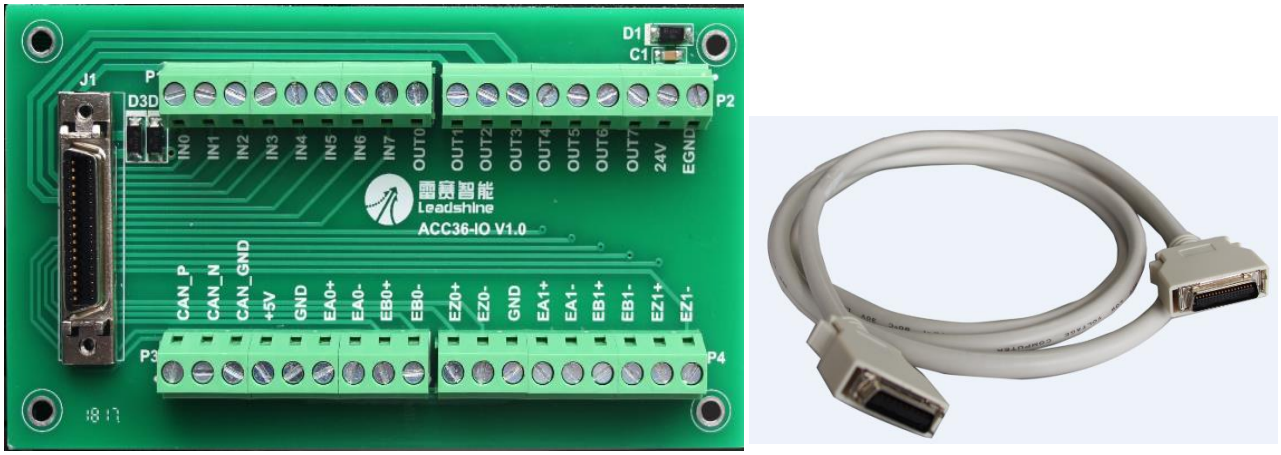


图1.3 DMC-E3032运动控制卡扩展接线板及电缆示意图

第 2 章　DMC-E3032 卡功能介绍

DMC-E3032运动控制卡是一款新型的EtherCAT总线运动控制卡。可以控制多个支持EtherCAT总线的伺服驱动器；适合于多轴点位运动、插补运动、手轮控制、编码器位置检测、IO 控制、位置比较、位置锁存等功能的应用。

DMC-E3032运动控制卡的运动控制函数库功能丰富、易学易用，用户开发应用软件十分方便。随卡免费提供的控制卡Motion调试软件，不但可以演示DMC-E3032卡的控制功能，而且可用于控制卡及运动控制系统的硬件测试。

2.1　运动控制功能

2.1.1　点位运动

点位运动是指：运动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。PC机执行点位运动指令，即调用点位运动函数，并自动将运动参数通过PCI总线接口传送至运动控制卡，使其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2.1所示。

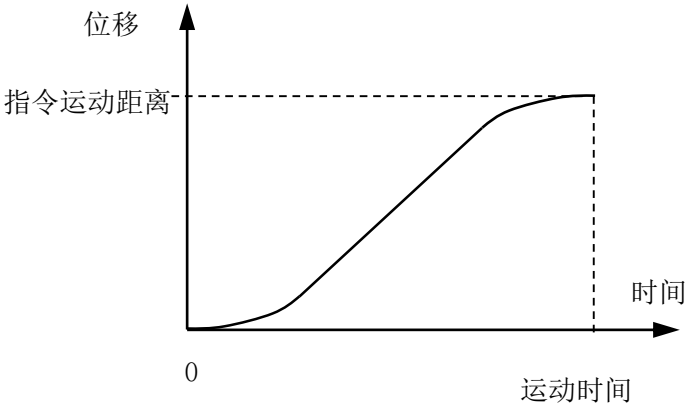


图2.1　定长运动位移曲线

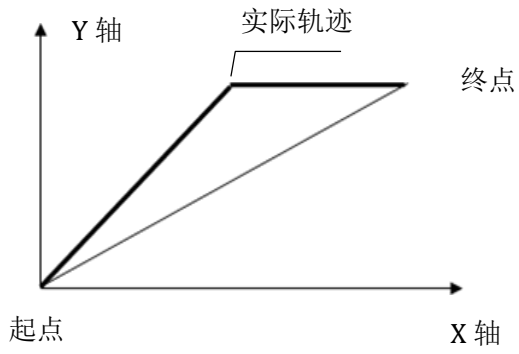


图2.2　两轴复合运动的轨迹

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2.2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。

2.1.1.1 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2.3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。

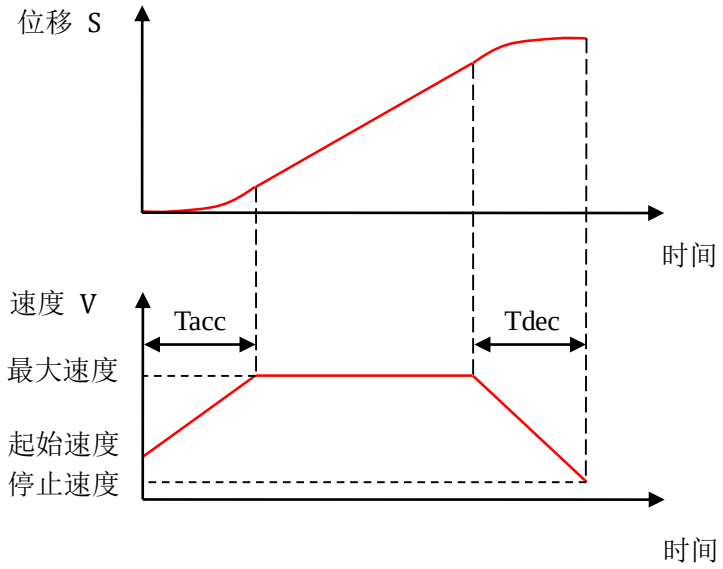


图2.3 梯形速度曲线及对应的位移曲线

Tacc：总加速时间， Tdec：总减速时间

2.1.1.2 S 形速度控制

为改善平台运动的平稳性，DMC-E3032运动控制卡还提供了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2.4所示。

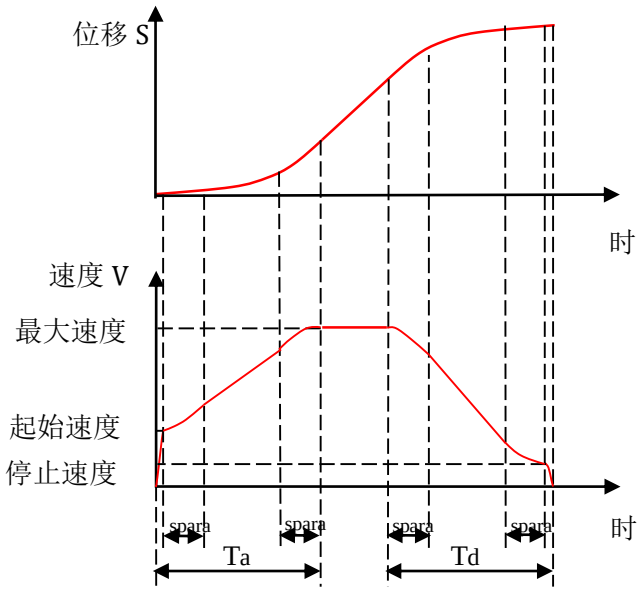


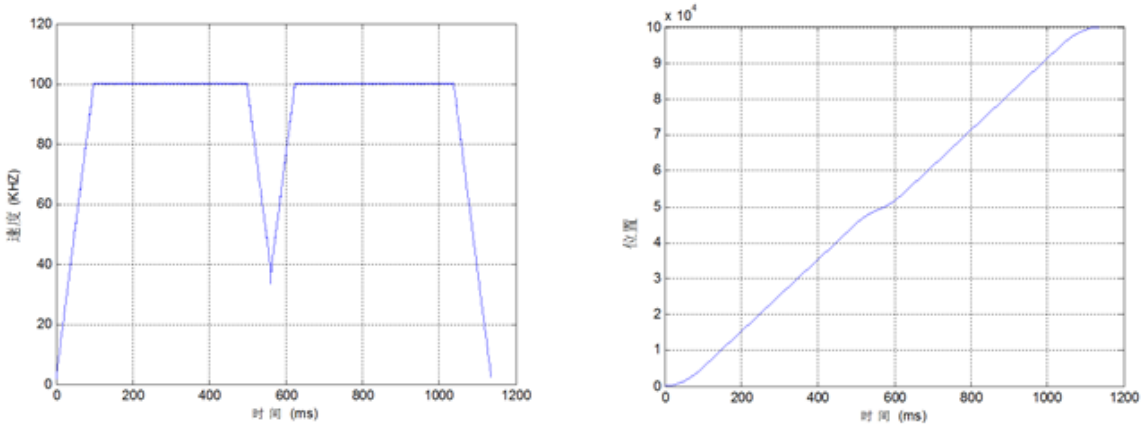
图2.4 S形速度曲线及对应的位移曲线

2.1.1.3 运动中改变终点位置

在进行点位运动的过程中，DMC-E3032 卡可以改变运动的终点位置，且位置可增可减。如图 2.5 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



(a) 速度曲线 (b) 位移曲线

图2.5 DMC-E3032卡改变终点位置的过程

2.1.1.4 运动中改变当前速度

在点位运动（或连续运动）过程中，DMC-E3032 卡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 型速度曲线相同，如图 2.6 所示。

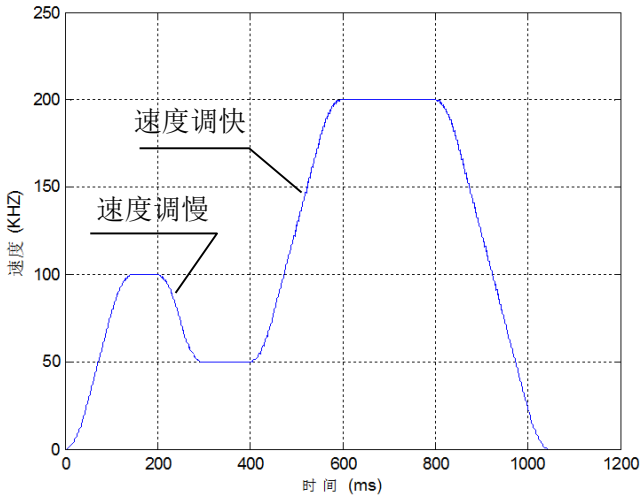


图2.6 DMC-E3032卡的调速过程

2.1.2 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，国外运动控制器将此指令称为 JOG 指令。

DMC-E3032 卡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 2.7 所示。

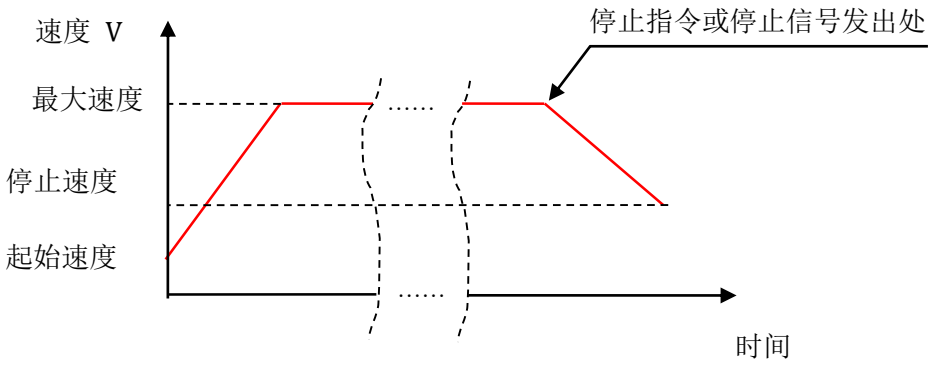


图2.7 连续运动速度曲线

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。

2.1.3 插补运动

为了实现轨迹控制，运动控制卡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 2.8 所示为各种曲线轨迹的示意图。

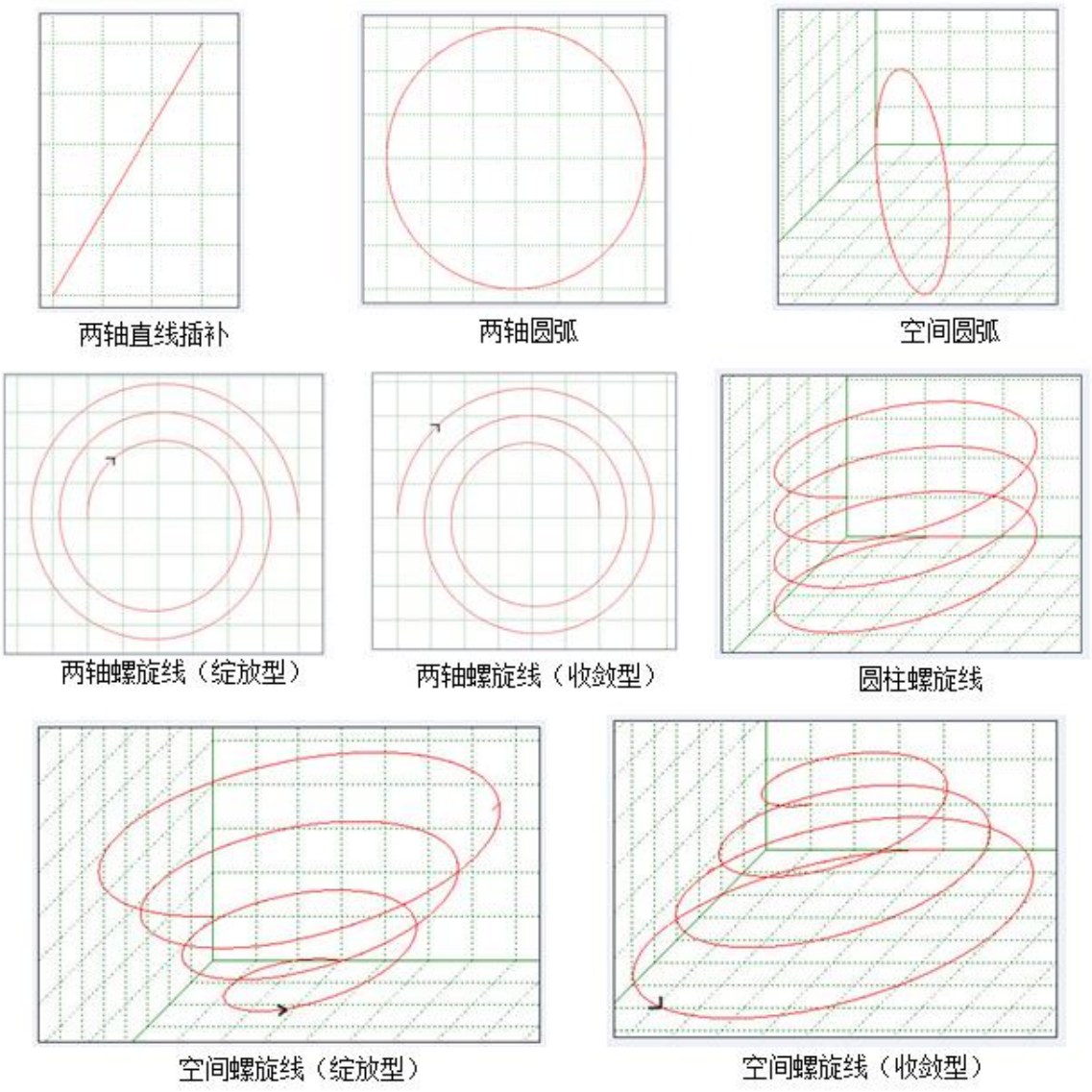


图 2.8　各种曲线轨迹示意图

DMC-E3032 可以进行任意 2~16 轴直线插补，同时，DMC-E3032 卡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，DMC-E3032 卡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。

2.1.4　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。运动系统动作时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。DMC-E3032 回原点参数可由控制卡设置，由驱动器完成。具体请参考驱动手册。

2.1.5 异常减速停止时间设置功能

DMC-E3032 卡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。

2.1.6 手摇脉冲发生器的控制

为了让用户能在手动模式下方便地调整机器的位置，DMC-E3032 卡提供了手摇脉冲发生器控制功能(手轮功能)。用户操控接在 DMC-E3032 卡上的手摇脉冲发生器(参见图 1.1、图 2.11)，手摇得慢，电机就转动得慢；手摇得快，电机就转动得快。

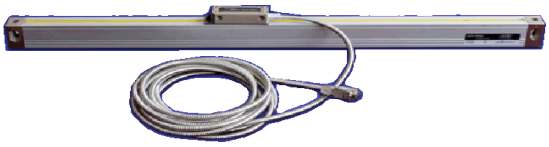
该功能多用于加工轨迹起点调整、刀具对位等工作中。

图2.11 手持式手摇脉冲发生器外形

2.2 编码器位置检测

DMC-E3032 卡有两个辅助编码器输入接口用于检测平台的位移或电机的转角。辅助编码器有 EA、EB、EZ 三个信号，脉冲计数信号由 EA 和 EB 端口输入；它可以接收两种类型的脉冲信号：正负脉冲输入或 A/B 相正交信号；EZ 信号是编码器零位信号。编码器外形如图 2.12 所示。

旋转编码器



光栅尺

图2.12 编码器外形

采用探针和编码器配合使用，DMC-E3032 卡通过位置触发功能，可完成对工件的位置检测工作，如图 2.13 所示。即：当探针接触到工件时，产生一个触发信号；DMC-E3032 卡接受该信号后，立即将编码器当前位置记录下来；通过记录工件的一系列数据，然后再通过软件处理，

即可获得该工件的外形尺寸。

图2.13 被测工件与探针

2.3 专用 IO 和通用 IO 控制

DMC-E3032 卡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。

2.3.1 位置比较输出

DMC-E3032 卡提供了位置触发输出信号的函数，包括单轴低速位置比较、单轴高速位置比较和一组二维高速位置比较。当电机运动到预先设置的位置时，自动触发特定的输出口。该功能在轨迹运动中用于控制点胶阀的开关、触发照相机快门等动作十分方便。

2.3.2 高速位置锁存

DMC-E3032 卡支持多种高速位置锁存功能，包括单次锁存、连续锁存以及锁存触发延时急停功能。连续锁存可实现对多个位置依次进行高速锁存，结合高速比较输出可以实现多个位置精确检测功能。高速触发急停可以实现在接收到触发信号时锁存当前位置并在设定的时间内停止运动这种特殊应用的精确定位功能。

2.3.3 通用 IO 信号

如图 1.1 所示，在电机运动的同时，DMC-E3032 卡还可接受按键、数字式传感器等信号，控制指示灯、继电器、电磁阀等器件。

2.3.4 EtherCAT IO 扩展功能

当设备需求的 IO 端口数量较多时，DMC-E3032 卡可配套 EtherCAT 总线 IO 模块对通用输入输出端口进行扩展。

2.4 多卡运行

DMC-E3032 卡的驱动程序支持最多 8 块 DMC-E3032 卡同时工作。因此，一台 PC 机可以同时控制多达 256 个电机同时动。

DMC-E3032 卡支持即插即用模式，用户可不必去关心如何设置卡的基地址和 IRQ 中断值。在使用多块运动控制卡时，首先要用运动控制卡上的拨码开关设置卡号；系统启动后，系统 BIOS 为相应的卡自动分配物理空间。

运动控制卡的编号是 0~7 号。在多卡系统中要控制某个电机运动，需要先确定卡号，再确定轴号。

第 3 章 硬件接口电路

3.1 硬件简介

雷赛 DMC-E3032 运动控制卡兼容 PCI V2.3 标准的 32Bit PCI 标准半长卡的尺寸规范，具体硬件系统框图如图 3.1 所示。其硬件布置及尺寸如图 3.2 所示：

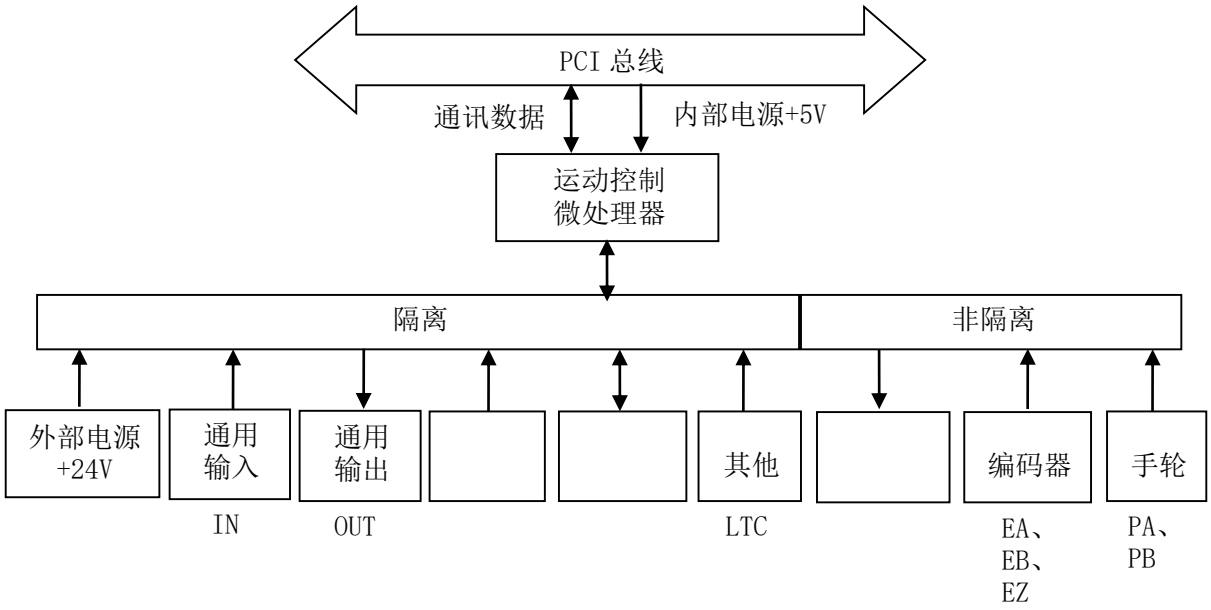


图3.1 运动控制卡硬件系统框图

DMC-E3032 运动控制卡硬件布置及尺寸如图 3.2 所示。

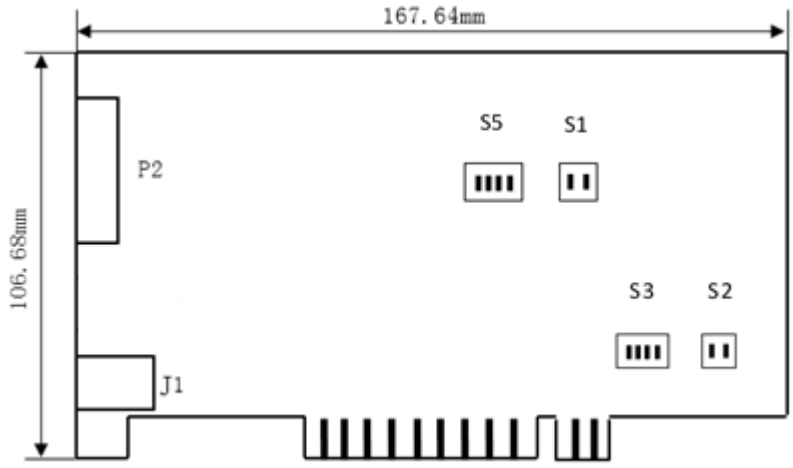


图 3.2 DMC-E3032 运动控制卡结构尺寸图

3.2 接口及引脚定义

DMC-E3032 运动控制卡的接口及脚位如表 3.1 所示。

表 3.1 接口及脚位

名称	功能介绍
P2	DB36 接口
J1	EtherCAT 接口
S5	控制卡 ID 号设置
S1	初始电平设置
S3、S2	控制卡启动方式设定

3.2.1 接口 P2 引脚定义

P2 为 DB36 接口，通过外接扩展接线板来使用，具体定义见附录 1 所示。

3.2.2 接口 J1 定义

J1EtherCAT 总线接口，可用于连接支持 EtherCAT 总线的伺服驱动器、EtherCAT 总线 IO 模块等。

3.3 控制卡与配件的连接

DMC-E3032 总线卡有一个选配的接线板，连接示意图如图 3.3 所示。

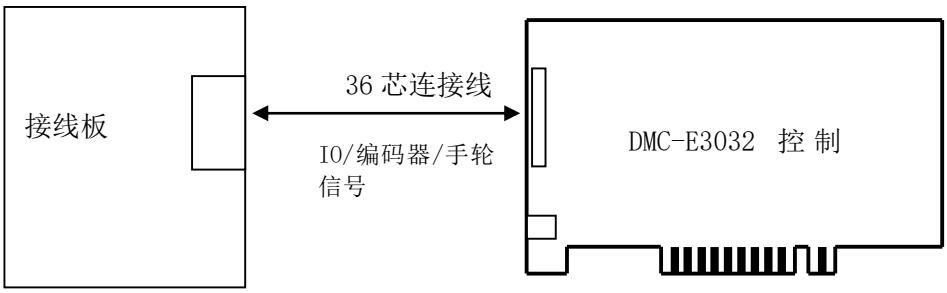


图3.3 DMC-E3032与配件连接示意图

3.4 编码器、手摇脉冲发生器接口电路

3.4.1 编码器信号输入接口

3.4.1.1 可接收的编码器信号类型

DMC-E3032 运动控制卡支持 2 种类型的辅助编码器信号输入：非 AB 相脉冲输入和 A/B 相正交信号。

1. 非 AB 相脉冲输入模式

该模式为脉冲+方向模式。在此模式下 EA 端口接收脉冲信号；EB 端口接收方向信号，高电平对应于计数器计数加，低电平对应于计数减。

2. AB 相正交信号输入模式

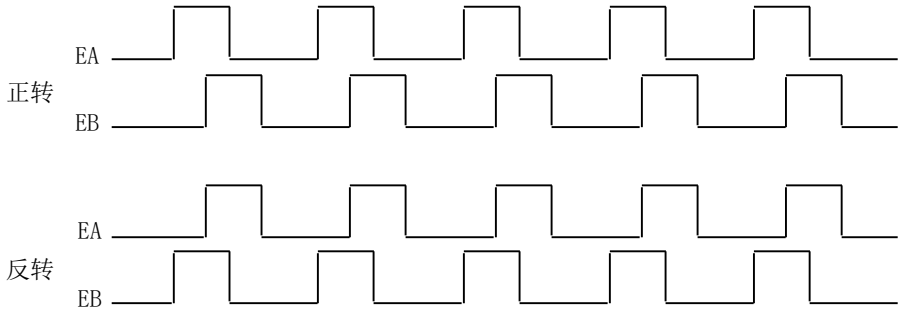


图3.4 A/B相正交信号

在这种模式下，EA 脉冲信号超前或滞后 EB 脉冲信号 90 度，而这种超前或滞后代表电机的运转方向。如图 3.4 所示，当 EA 信号超前 EB 信号 90° 时，被视为正转；当 EB 信号超前 EA 信号 90° 时，被视为反转。

为了提高编码器的分辨率，控制卡采用 4 倍频计数模式对 EA，EB 信号进行计数设置。

4 倍频计数：EA、EB 信号的上升沿和下降沿都参与触发计数器，故将一个脉冲周期就分为四份。所以，计数精度提高了 4 倍。

例如：如果使用的编码器为 2500 线，即电机转一周反馈的 EA、EB 脉冲数都为 2500 个。让电机转一周，若编码器输入模式为 4 倍频计数，编码器计数器的值为 10000。

3.4.1.2 编码器信号输入接口电路

如果使用差分输出的编码器，输入信号的正端接 EA+(或 EB+，EZ+)端，负端接 EA-(或 EB-，EZ-)端。如图 3.5 所示。

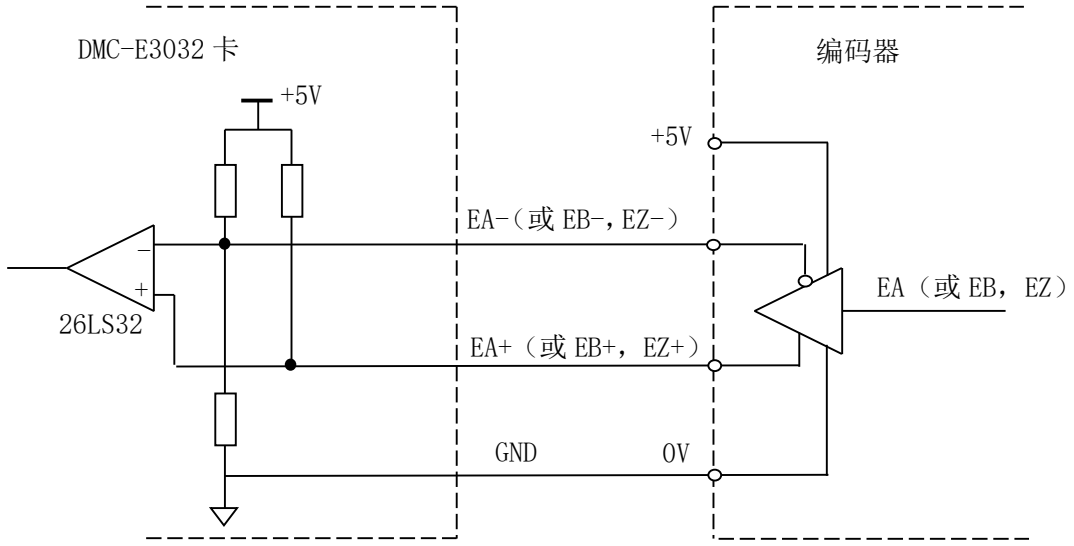


图 3.5　差分输出编码器接线原理图

如果使用集电极开路输出的编码器，则编码器输出信号接 EA+（或 EB+，EZ+）端，而 EA-（或 EB-，EZ-）端悬空。如图 3.6 所示。

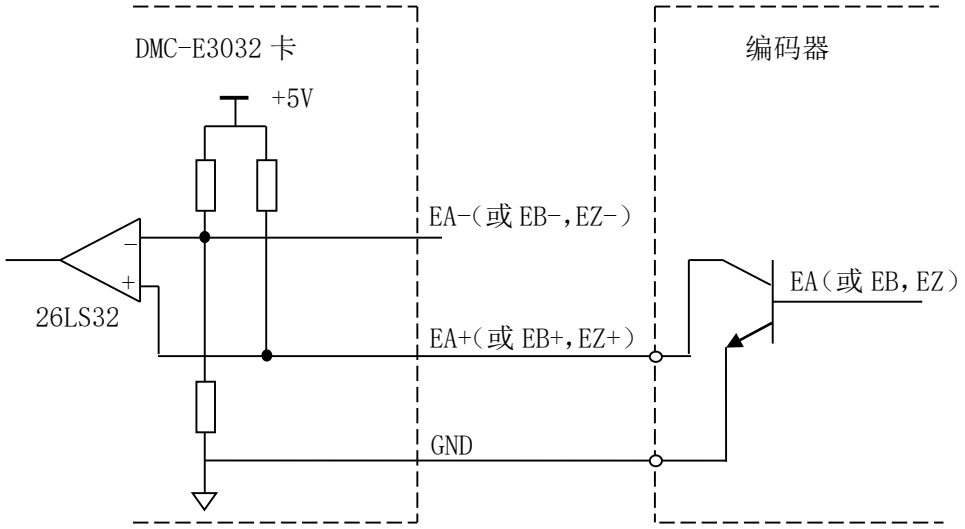


图 3.6　集电极开路输出的编码器接线原理图

注意：

1）编码器等脉冲输入信号的 EA+、EA-、EB+、EB-和 EZ+、EZ-的差分信号电压差必须高于 3.5V，小于 5V，且输出电流不应小于 6mA。

2）需要将输入设备的地线和控制卡的 GND 连接。

3.4.2　手摇脉冲发生器输入接口

DMC-E3032 卡为轴提供了手摇脉冲发生器（手轮）脉冲信号 PA、PB 的输入接口（接口与

辅助编码器输入复用），用户可以通过手摇脉冲发生器控制电机的运动，电机的运动距离和速度由手摇脉冲发生器输入的脉冲数和脉冲频率控制，具体使用请参考相关指令和函数。其接口原理如图 3.7 所示。

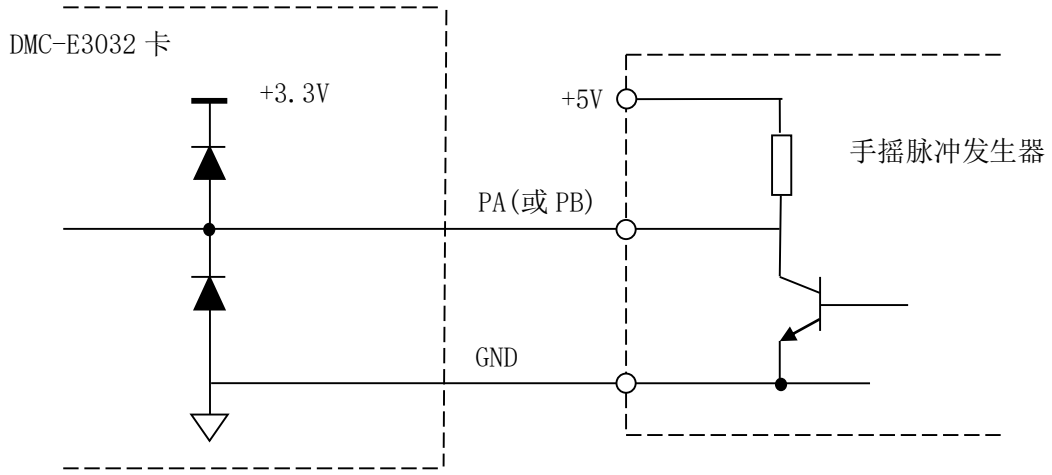


图 3.7　手轮脉冲输入接口原理图

3.5　专用 I/O 接口电路

3.5.1　高速位置锁存输入信号接口

DMC-E3032 卡有四路位置锁存输入信号（硬件对应 IN4-IN7），用于锁存辅助编码器位置。其接口电路原理如图 3.8 所示。

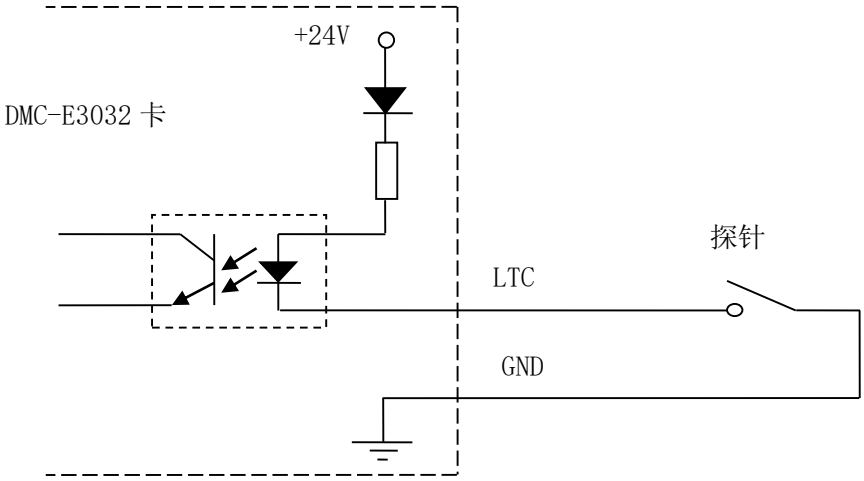


图 3.8　DMC-E3032 位置锁存输入信号接口原理图

3.5.5 高速位置比较输出信号接口

DMC-E3032 卡有六个高速位置比较器，每个高速位置比较器均配有 1 个硬件位置比较输出接口。通过软件使能后，可设置比较模式，当辅助编码器寄存器内数值满足触发条件时，硬件自动在 CMP 端口上输出一个开关信号。

DMC-E3032 的 CMP 接口原理图如图 3.9 所示。

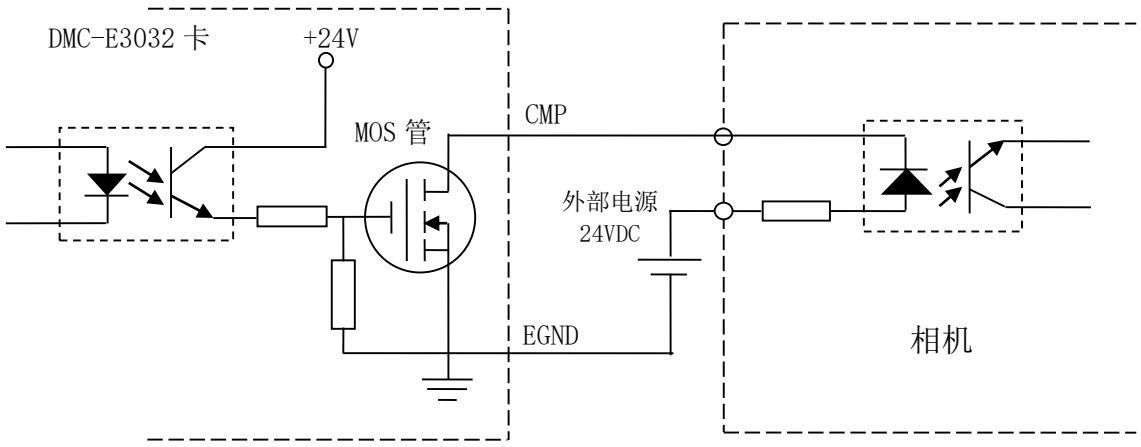


图 3.9 DMC-E3032 高速位置比较输出信号接口原理图

3.6 通用 I/O 接口电路

3.6.1 通用数字输入信号接口

DMC-E3032 卡有 8 路通用数字输入信号（其中 IN4-IN7 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 3.10 所示。

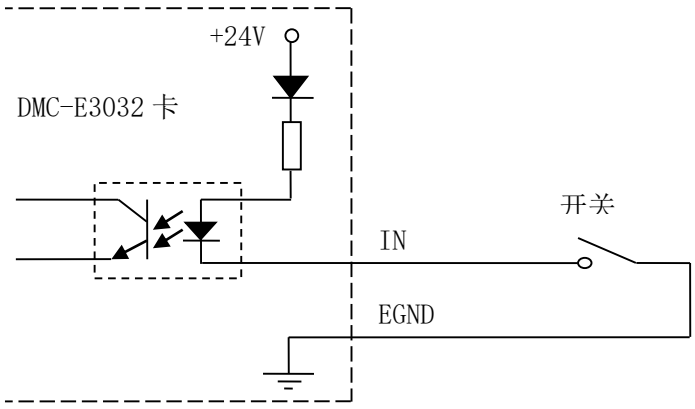


图 3.10 通用输入信号接口原理图

3.6.2 通用数字输出信号接口

DMC-E3032 有 8 路通用数字输出信号（其中 OUT2~OUT7 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 3.11 所示。

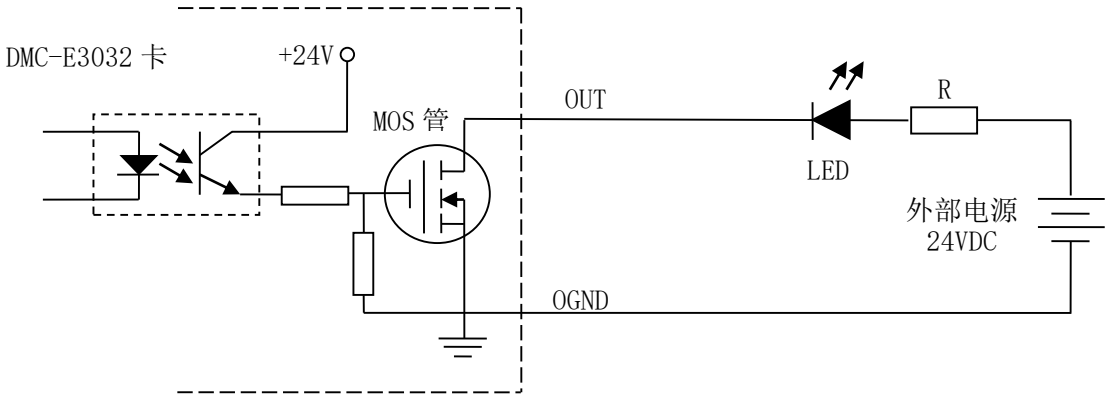


图 3.11 DMC-E3032 输出口接发光二极管

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，输出口为 1 时，灯不亮为原则。接线图如图 3.12 所示。

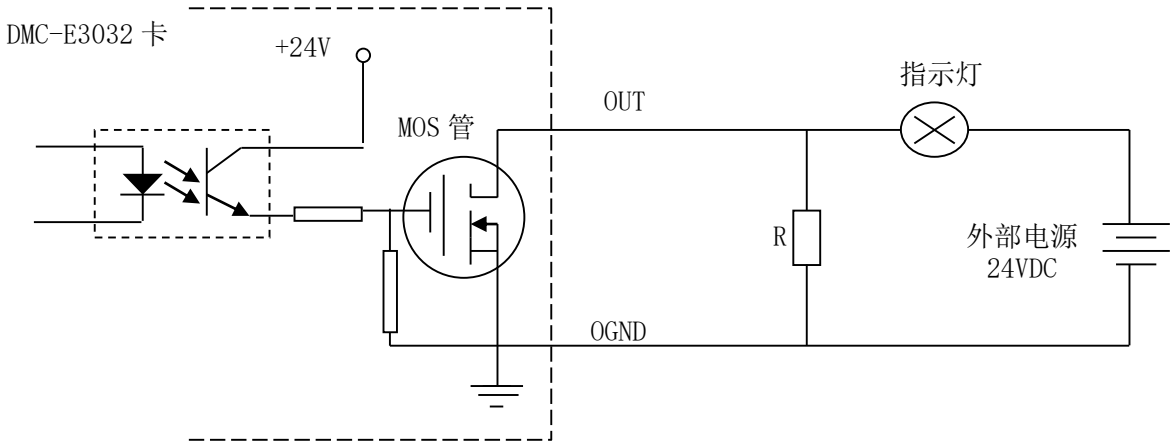


图 3.12 DMC-E3032 卡灯丝型指示灯接线图

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 3.27 所示。

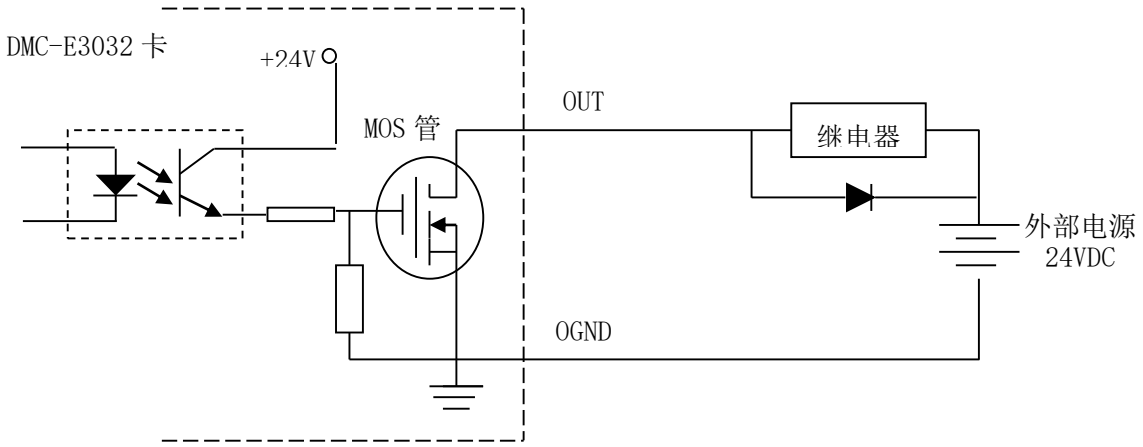


图 3.12　DMC-E3032 接小型继电器的接线图

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　硬件及驱动程序的安装

4.1　硬件安装步骤

4.1.1　硬件设置

如图 4.1 所示，DMC-E3032 运动控制卡上有 2 组拨码开关 S1、S2、S3、S5，用于设置 DMC-E3032 卡的工作参数。

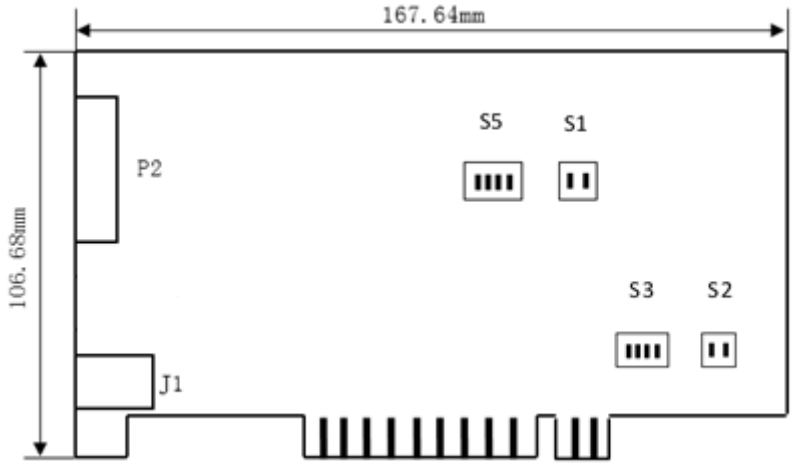


图 4.1　DMC-E3032 板卡拨码开关的位置

拨码开关 S1 用于设置总线卡接线板输出口初始电平，S5 用于设置控制卡卡号。具体位置及设置方法如表 4.1、4.2 所示。

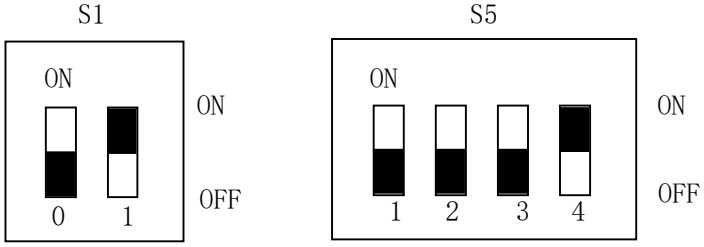


图 4.2　拨码开关示意图

表 4.1 拨码开关 S2 设置卡初始电平

拨码开关号	对应的输出口	拨码开关位置	初始输出电平
S1-1	OUT0~3	ON	高电平
		OFF	低电平
S1-2	OUT4~7	ON	高电平
		OFF	低电平

注意：1）拨码开关 S1 出厂默认值全部为 ON，即 OUT0～7 初始电平为高电平。

2）拨码开关 S1 的设置只是设置输出口的初始电平，不影响输出口高低电平与逻辑值的关系。

表 4.2 拨码开关 S5 设置运动控制卡号

S5-4	S5-3	S5-2	S5-1	控制卡号
	OFF	OFF	OFF	0
	OFF	OFF	ON	1
	OFF	ON	OFF	2
	OFF	ON	ON	3
	ON	OFF	OFF	4
	ON	OFF	ON	5
	ON	ON	OFF	6
	ON	ON	ON	7

注意：1）拨码开关 S5-1、S5-2、S5-3 出厂默认配置为 ON，即默认设置为 7 号卡。

2）当每张卡都设置为 0 号卡时，按靠近 CPU 的顺序自动排序。除此种情况外，如有两张卡以上设置为相同卡号，则初始化函数 dmc_board_init 会返回一个错误代码。

拨码开关 S2 和 S3 用于设置总线卡的启动方式，按照出厂设置使用就可以（QSPI FLASH）。

表 4.2 拨码开关 S2、S3 设置运动控制卡启动方式

	S3				S2	
BOOT MODE	4	3	2	1	2	1
QSPI FLASH	OFF	ON	ON	OFF	OFF	ON
SD	OFF	ON	ON	OFF	ON	OFF
JTAG	OFF	ON	OFF	ON	OFF	ON

4.1.2 硬件安装

DMC-E3032 运动控制卡硬件遵从 32bit PCI 卡结构标准，其安装方法与其它 PCI 卡，如：声卡、网卡等相似。具体步骤如下：

1）打开控制卡的包装，参考 4.1.1 节硬件设置的说明，按照实际需求，完成拨码开关的设置；

2）操作员要带好防静电手套，并触摸一下地线，完全释放身上的静电；

3）关闭 PC 机以及一切与 PC 相连的设备；

4）打开 PC 机的机箱；

5）选择一个靠近处理器的 32bit PCI 插槽，将控制卡垂直插入插槽中；

6）将控制卡用螺钉固定在 PC 机机箱上，确保紧固可靠；

7）将接线板用电缆线与控制卡对应的插座连接，并确保连接牢固可靠。

4.2 驱动程序安装步骤

DMC-E3032 运动控制卡的驱动程序遵从 32bit PCI 卡驱动标准,其安装方法与其它 PCI 卡，如：声卡、网卡等相似。雷赛为 DMC-E3032 运动控制卡提供驱动程序一键式安装包，用户只需按照步骤安装即可，下面简要讲述相关安装过程。

1、双击驱动文件 DMCDriver_Setup.exe，如果出现如图 4.2 所示对话框，点击“下一步”继续安装；如果出现如图 2 所示对话框，请选择“修复”，并点击“下一步”，跳转到第 5 步继续安装。

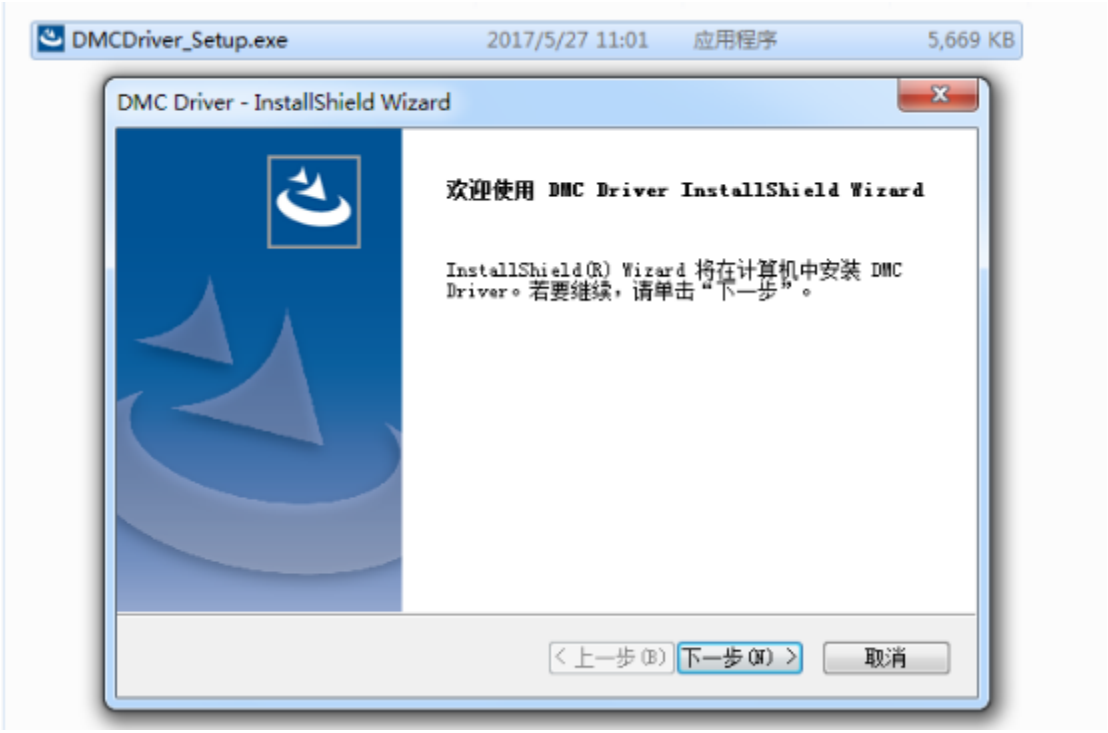


图 4.2　驱动程序开始安装

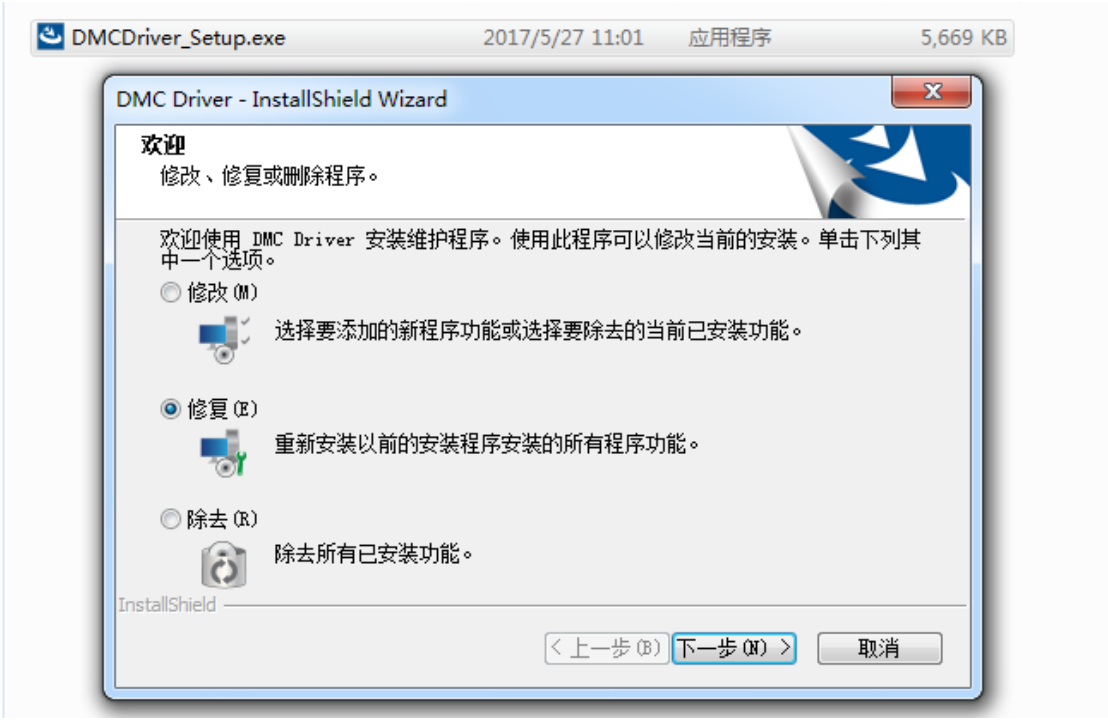


图 4.3　已安装驱动程序选项

2、如图 4.4 所示，输入用户名和公司名称，点击“下一步”继续安装。

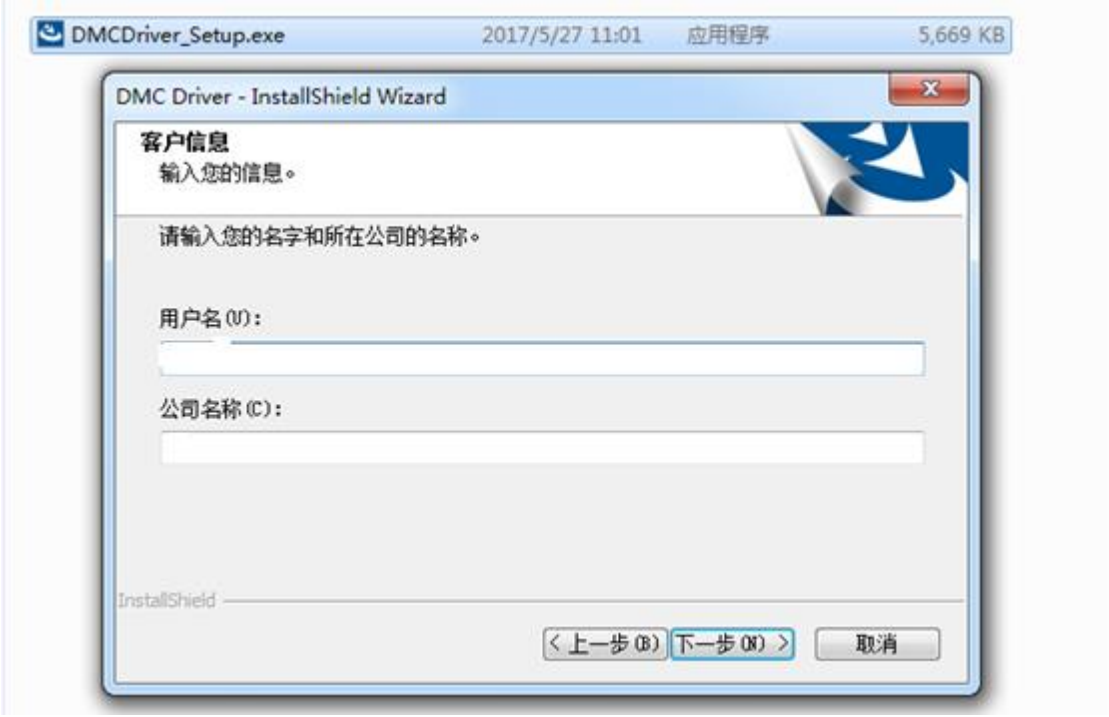


图 4.4　输入信息对话框

3、如图 4.5 所示，请选择“全部”，然后点击“下一步”继续安装。

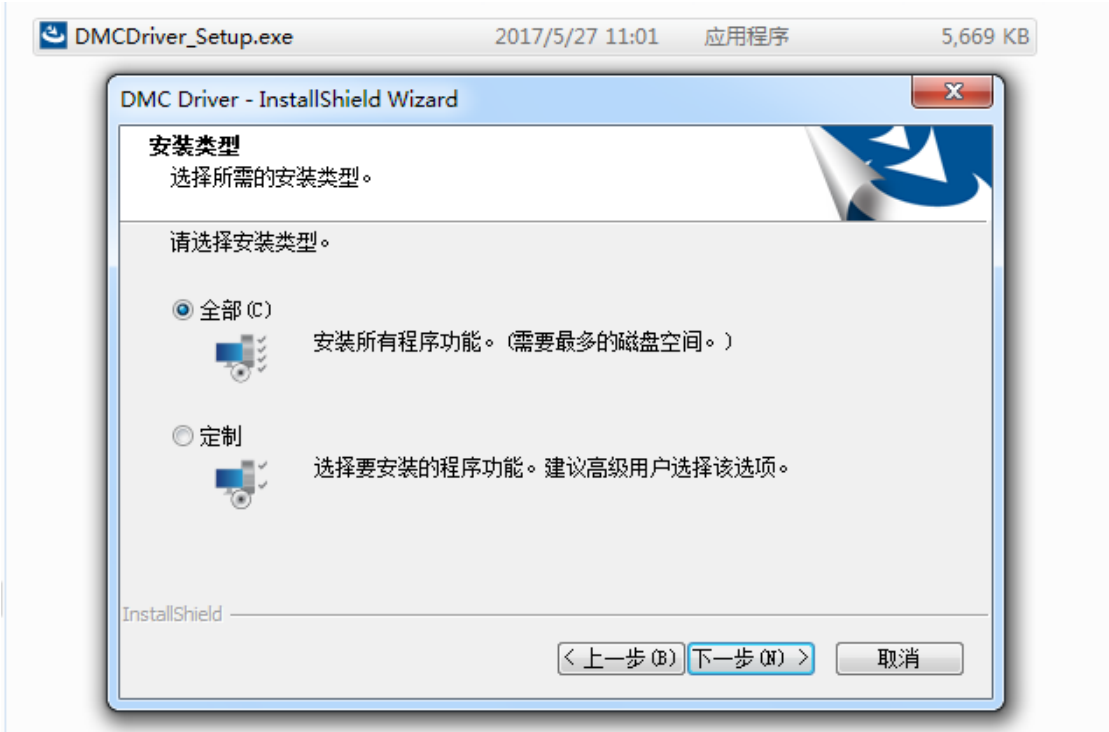


图 4.5　安装类型选择

4、如图 4.6 所示，点击“安装”按钮，开始安装，。

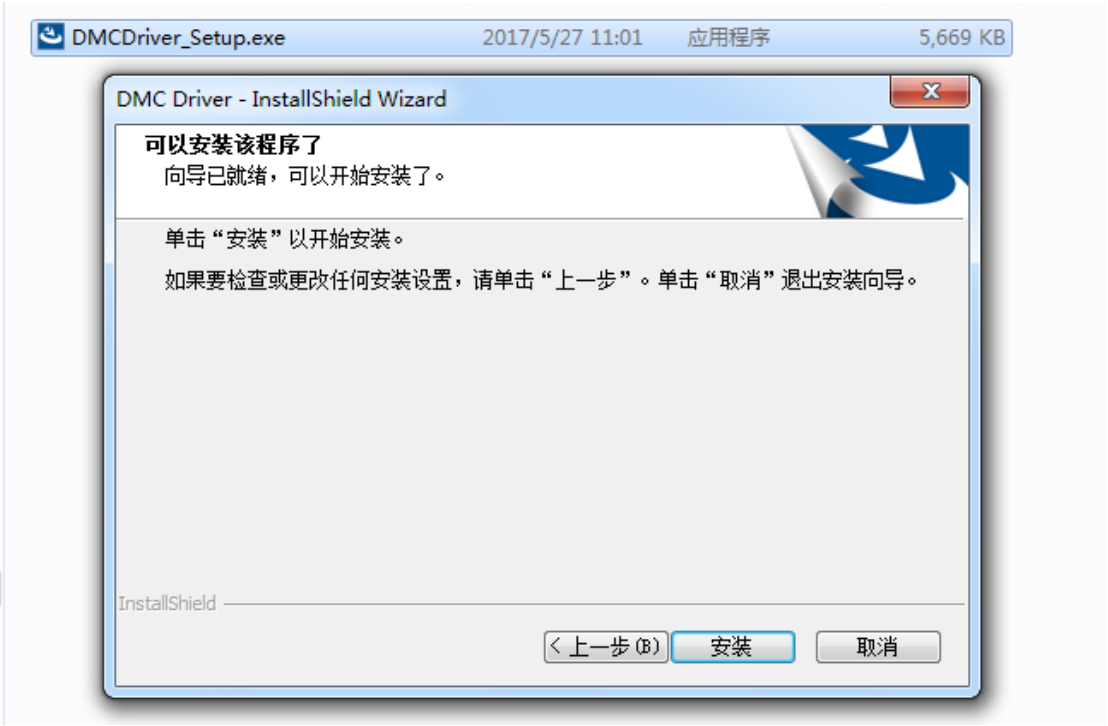


图 4.6　驱动程序开始安装

5、安装过程中会出现如图 4.7 所示的 Windows 安全提示对话框，请点击“始终安装此驱动程序软件”，并继续安装。

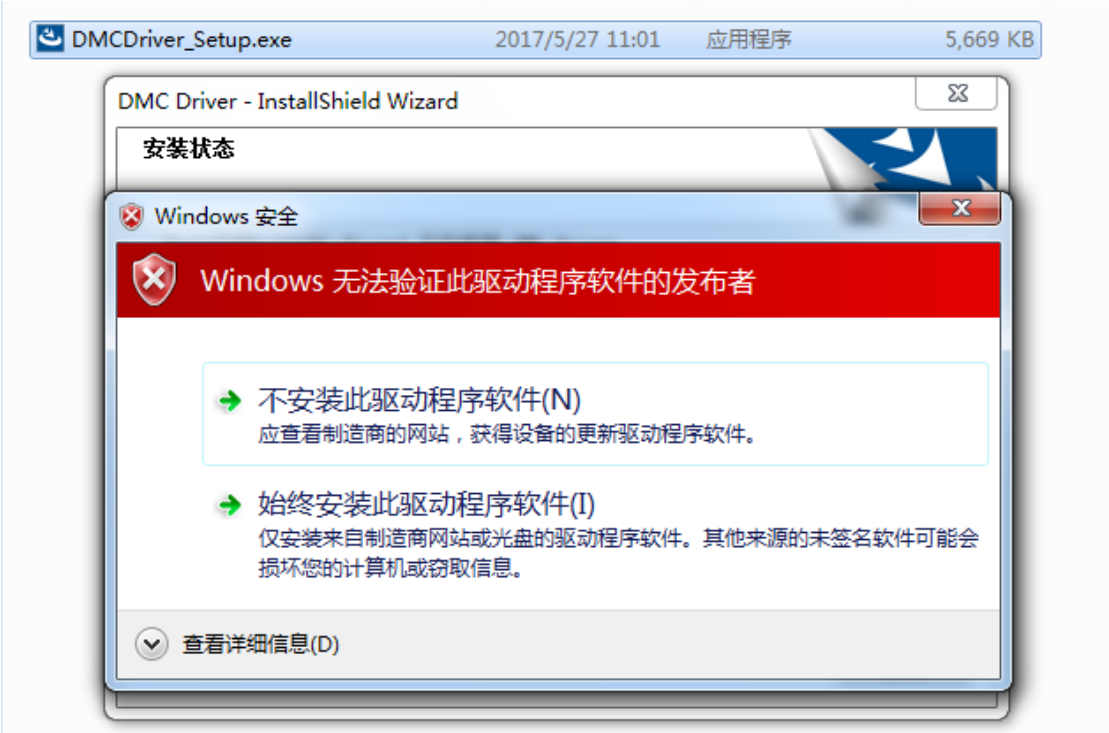


图 4.7　Windows 安全提示

6、安装完成后，显示界面如图 4.8 所示，点击“完成”。

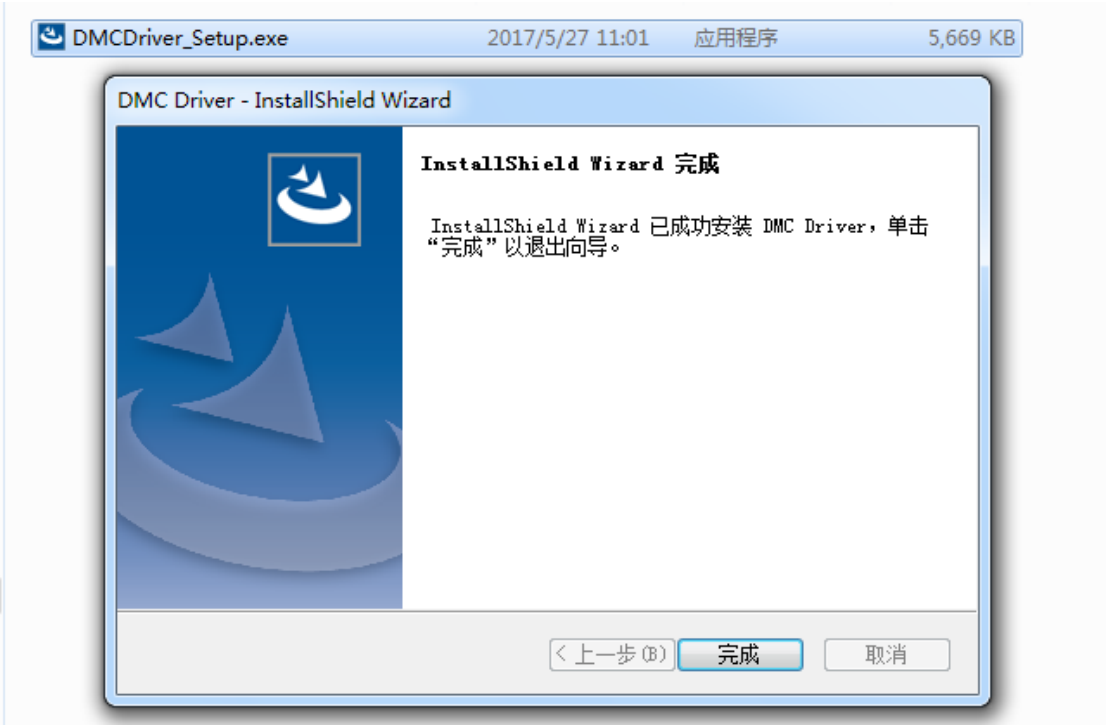


图 4.8 完成驱动程序安装

7、打开设备管理器，在“Jungo”选项下可以看到“DMC3K5K”和“LeisaiDrvr1230”注册信息。至此控制卡就可以正常使用了，如图 4.9 所示。

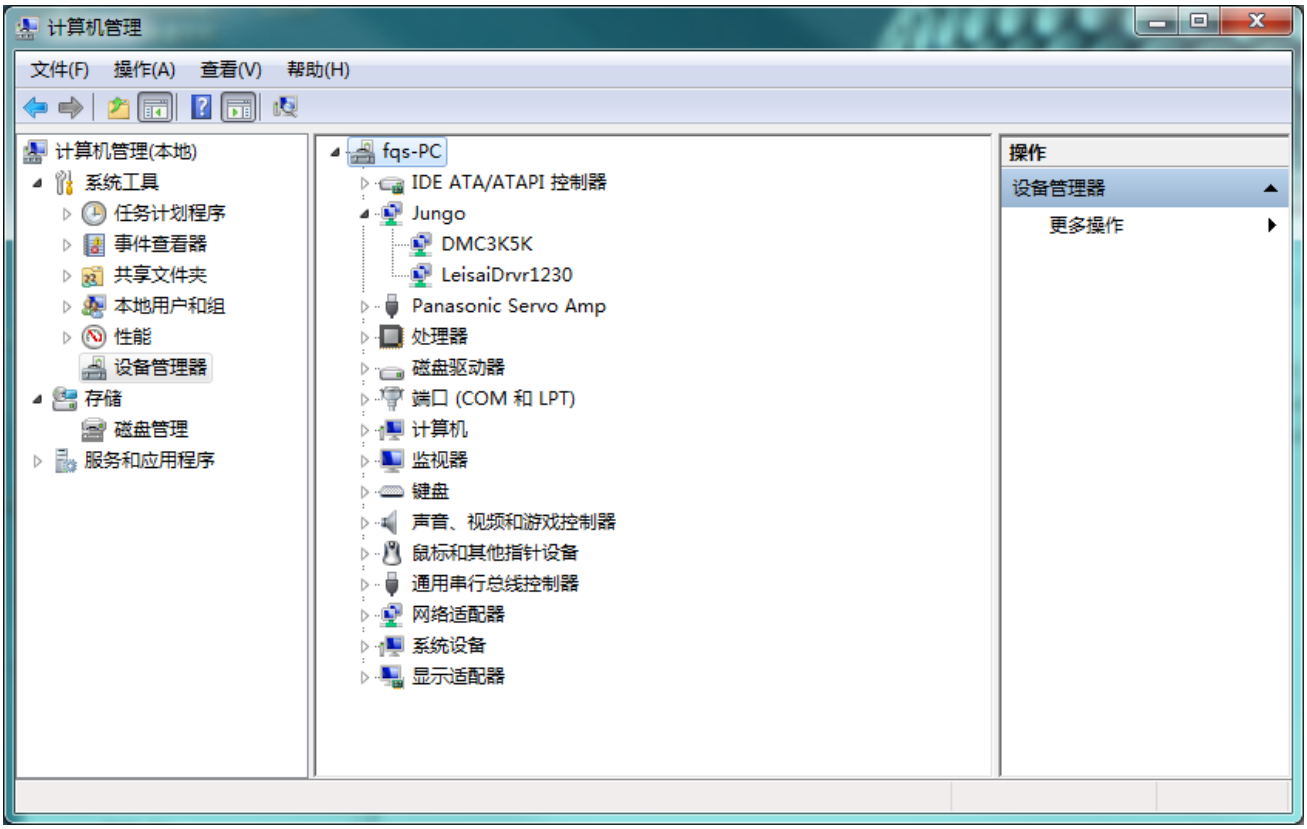


图 4.9 正确安装驱动程序后的设备信息

4.3 驱动程序卸载步骤

1、双击驱动文件 DMCDriver_Setup.exe，如果出现如图 4.10 所示对话框，请选择“除去”，并点击“下一步”。

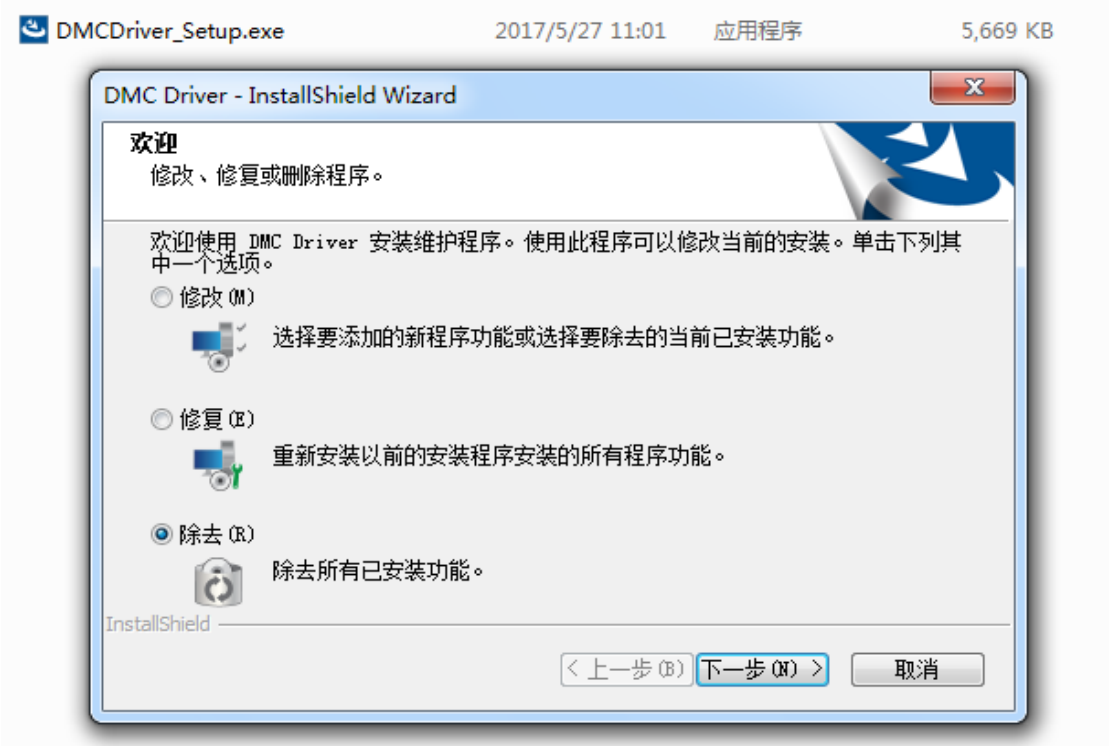


图 4.10 删除驱动选项

2、出现确认对话框，请选择“是”按钮。

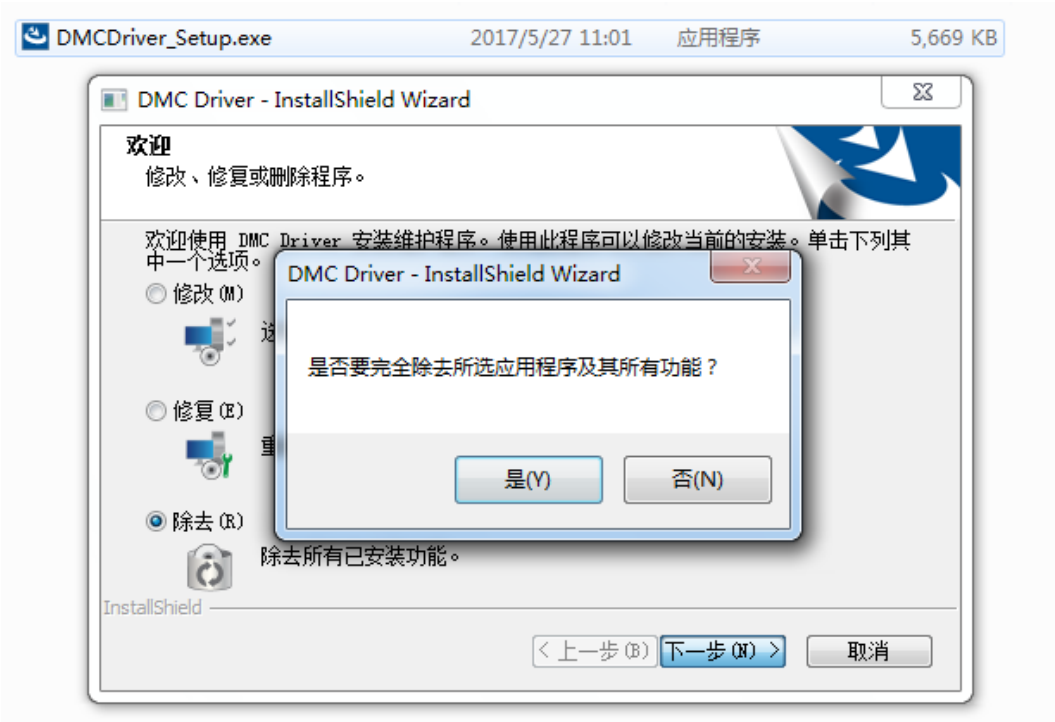


图 4.11　是否删除提示窗

3、卸载完成后，显示界面如图 4.12 所示，点击“完成”，完成卸载。

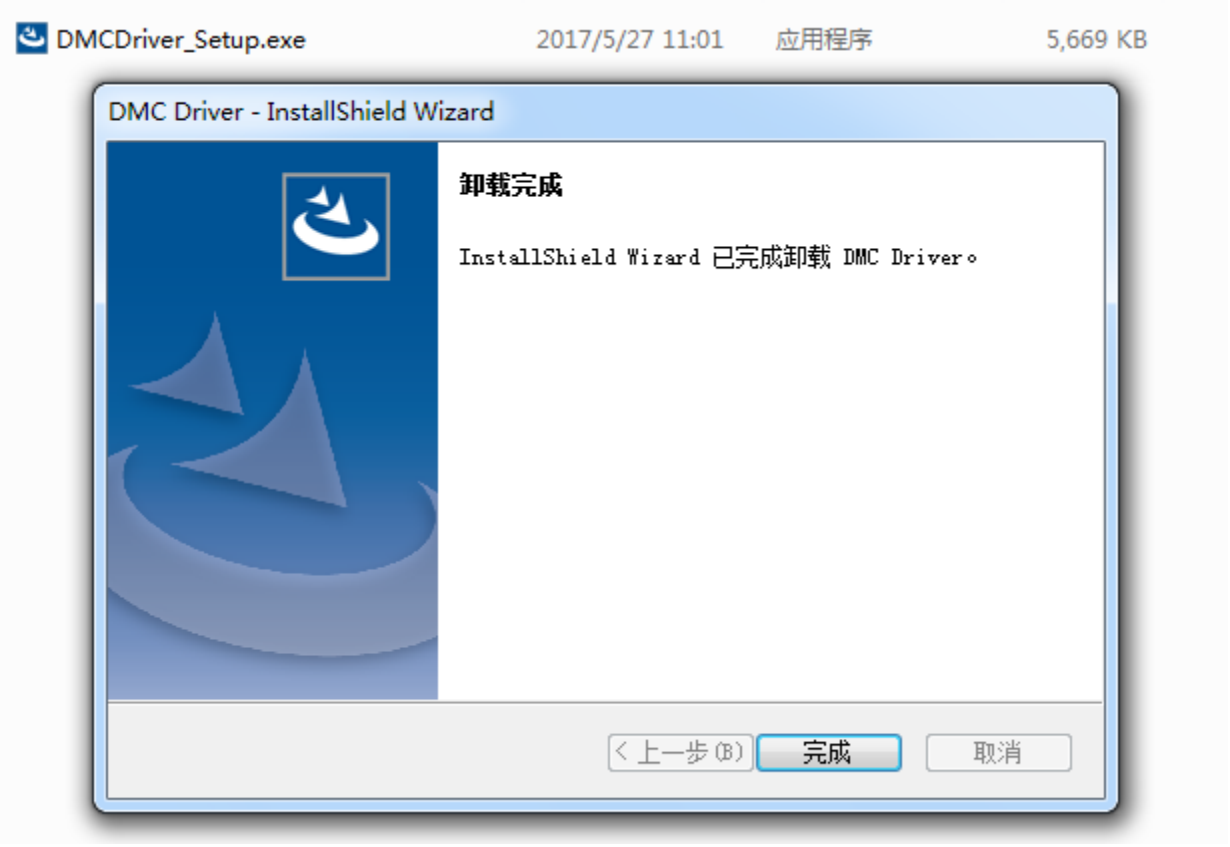


图 4.12　完成驱动程序的卸载

第 5 章　控制卡总线 Motion 的安装与使用方法

雷赛公司除了开发了驱动程序、函数库以外，还开发了一套控制卡总线 Motion。

用户在使用 VB、VC 或其它高级语言编写应用程序之前，可利用控制卡总线 Motion 软件快速熟悉 DMC-E3032 卡的硬件、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，需用总线 Motion 扫描从站并进行总线配置。

5.1　控制卡总线 Motion 的安装步骤

安装控制卡总线 Motion 测试软件的步骤如下：

1）使用控制卡总线 Motion 前，请确保已经通读本手册，并参照第 4 章硬件及驱动程序的安装，完成硬件设置、硬件安装和驱动程序安装。

2）启动 PC 机，进入 Windows 操作系统。

3）将 DMC-E3032 卡所配光盘放入光驱中，找到光盘中的“\MOTION\控制卡总线 Motion”目录，将其全部复制到硬盘中，弹出光盘。

4）在复制到硬盘中的控制卡总线 Motion 目录中找到 Leadshine.DMCMotion.exe，双击运行，即显示如图 5.1 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。

5）选择所需的卡，然后进行相应的操作。

6）至此，即可使用控制卡总线 Motion 软件进行测试。

注意：本软件是基于.NET 4.0 开发，请在使用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。

5.2　测试软件的功能与使用方法

当使用 DMC-E3032 控制卡时，控制卡总线 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。

控制卡 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。

各功能详细使用方法见控制卡 Motion 使用手册。

第 6 章　应用软件开发方法

DMC-E3032 运动控制卡的应用软件可以在 Visual Bisic 和 Visual C++等高级语言环境下开发。

如果您对 VB、VC 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 8 章“总线操作函数说明”及第 9 章“基本功能函数说明”。

6.1　基于 WINDOWS 平台的应用软件结构

使用雷赛运动控制卡的自动化设备运动控制系统构架如图 6.1 所示：

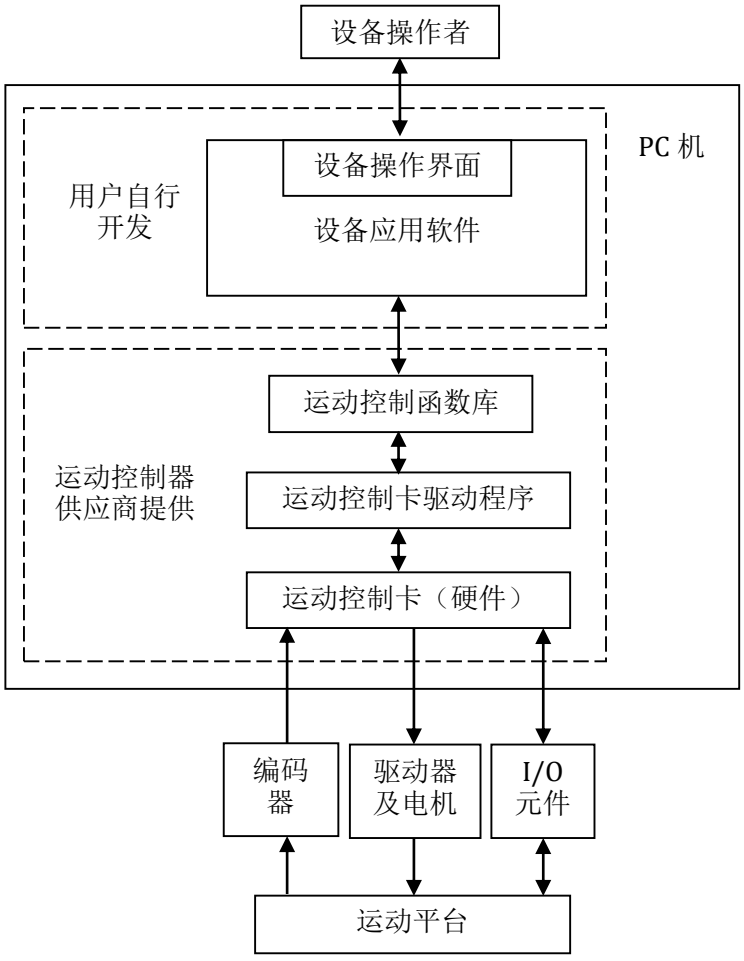


图 6.1　基于雷赛运动控制卡的自动化设备运动控制系统构架

从图 6.1 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）运动函数通过雷赛运动控制卡驱动程序向运动控制卡发出控制指令；

4)运动控制卡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 DMC-E3032 运动控制卡的硬件驱动程序和 DLL 运动函数库，包括控制卡初始化、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 6.2 所示。

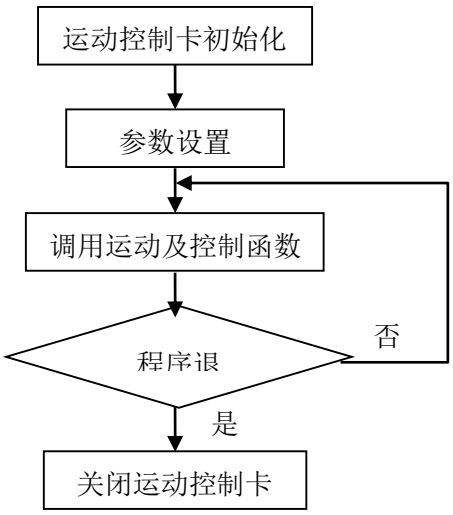


图 6.2　设备应用软件的典型流程

6.2　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 6.3 所示。

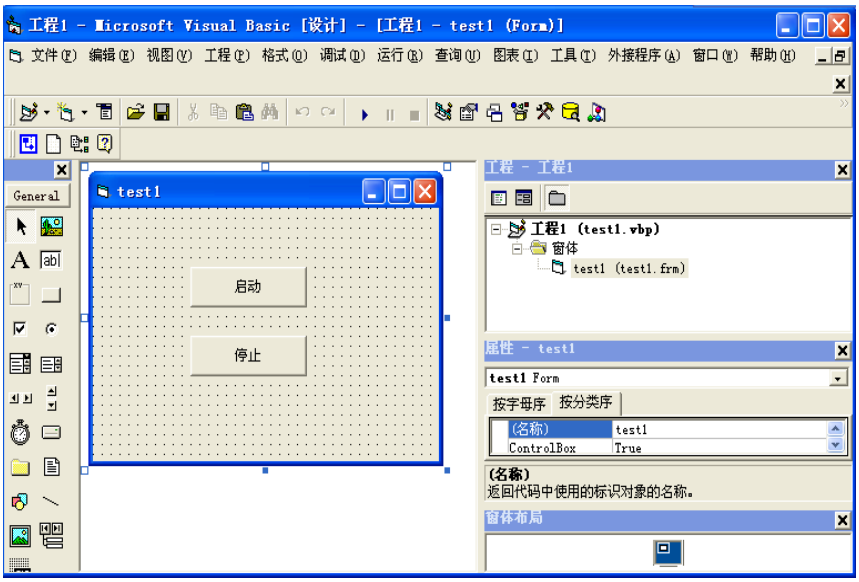


图 6.3 修改对话框（VB）

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 6.4 所示。

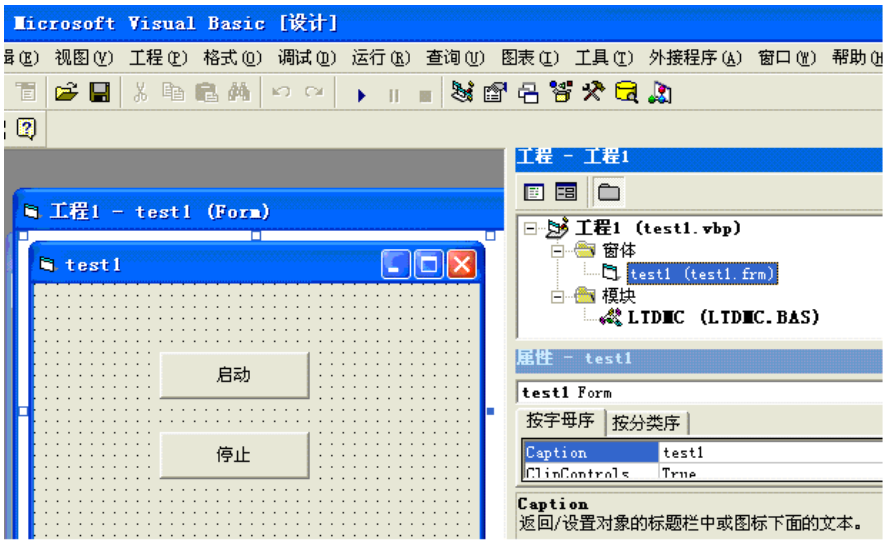


图 6.4 添加头文件

6）双击窗口控件，在 Form_Load 事件中添加代码 dmc_board_init。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 6.5 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 6.5 程序运行界面（VB）

6.3 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 6.6。

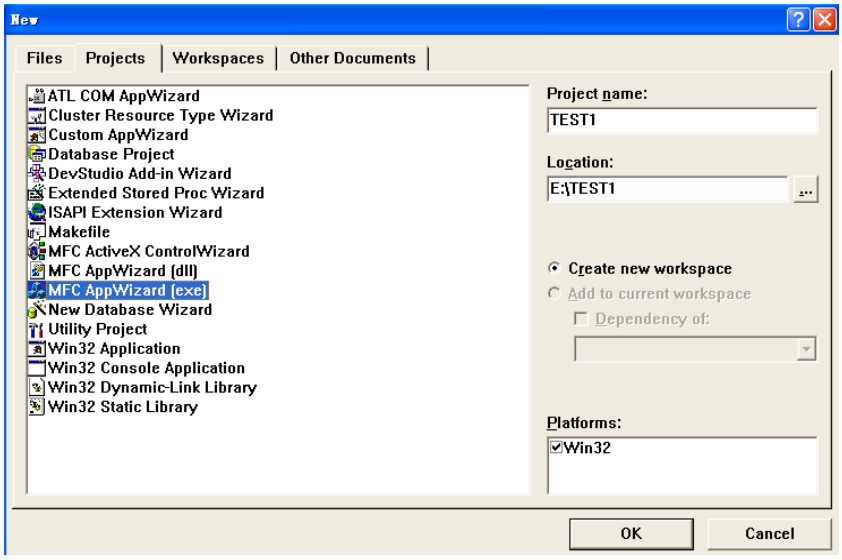


图 6.6 创建新工程

6）在应用程序类型中选择“基于对话”，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 6.7 所示。

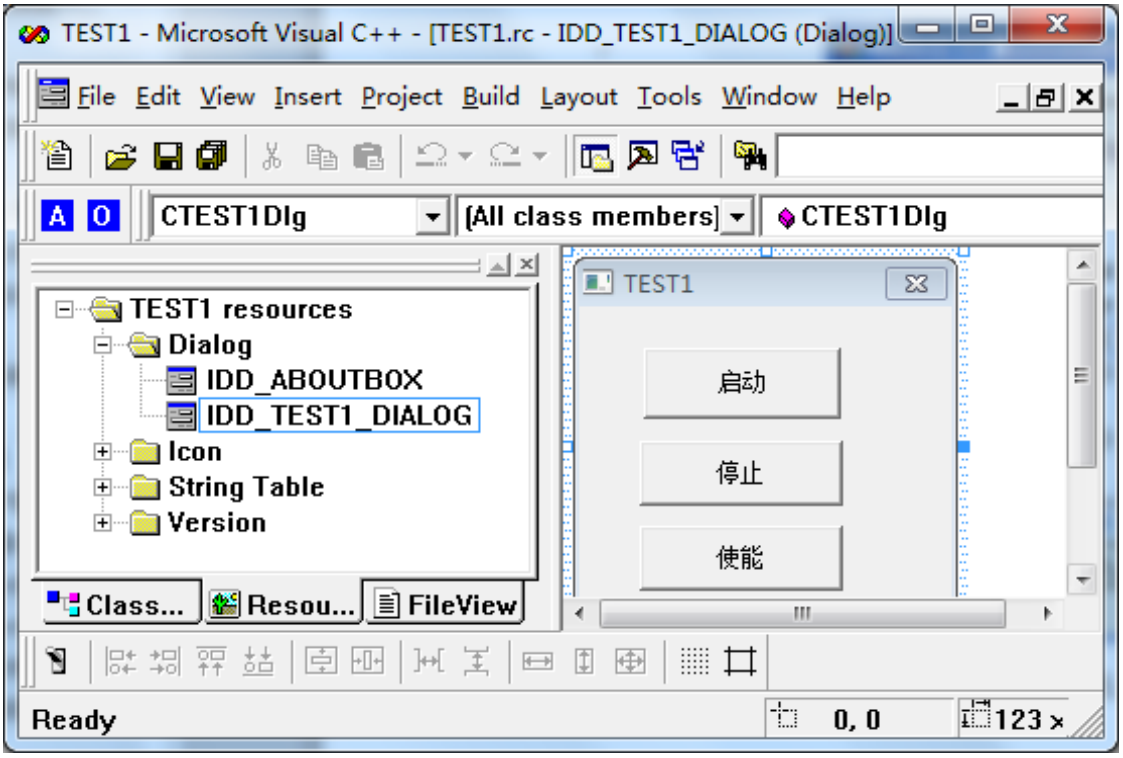


图 6.7 修改对话框

8)在相应的目录下找到 LTDMC.h、LTDMC.lib、LTDMC.dll 和 PVT.dll 文件，拷贝到 E:\test1 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 6.8 所示。

11）在 CTest1Dlg::OnInitDialog()函数中添加代码：dmc_board_init();如图 6.9。

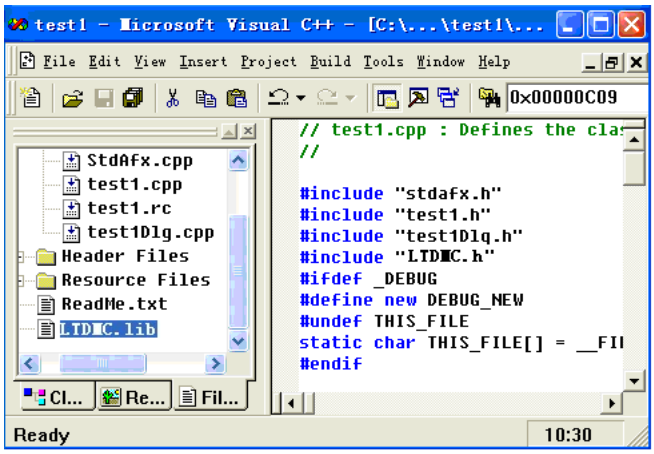


图 6.8 程序增加头文件

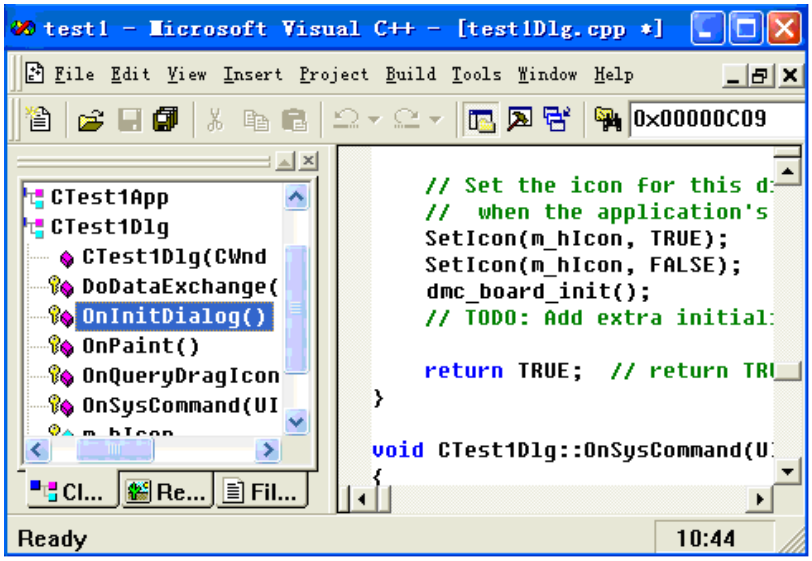


图 6.9 程序增加初始化函数

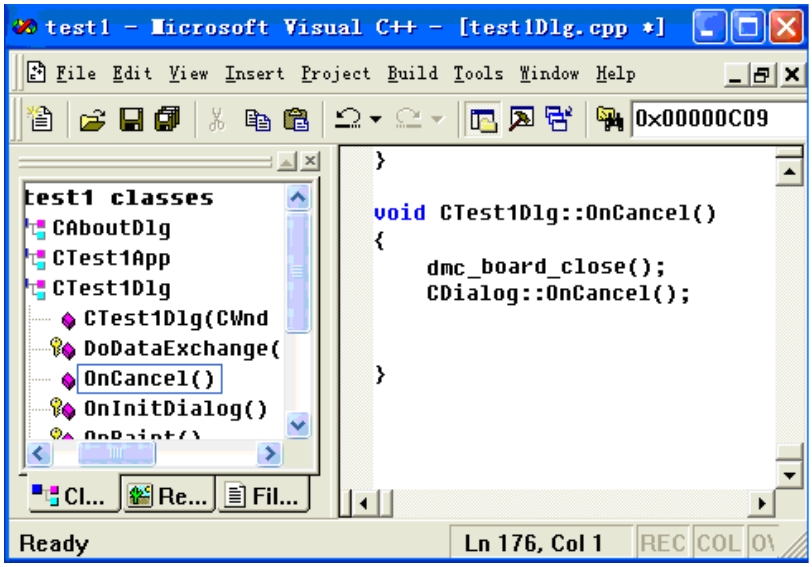


图 6.10 程序增加 OnCancel 函数

12）如图 6.10 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 6.11 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 6.11　程序运行界面

6.4　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 应用窗体程序”，并将名称修改为“test1”，如图 6.12 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 6.13 所示。

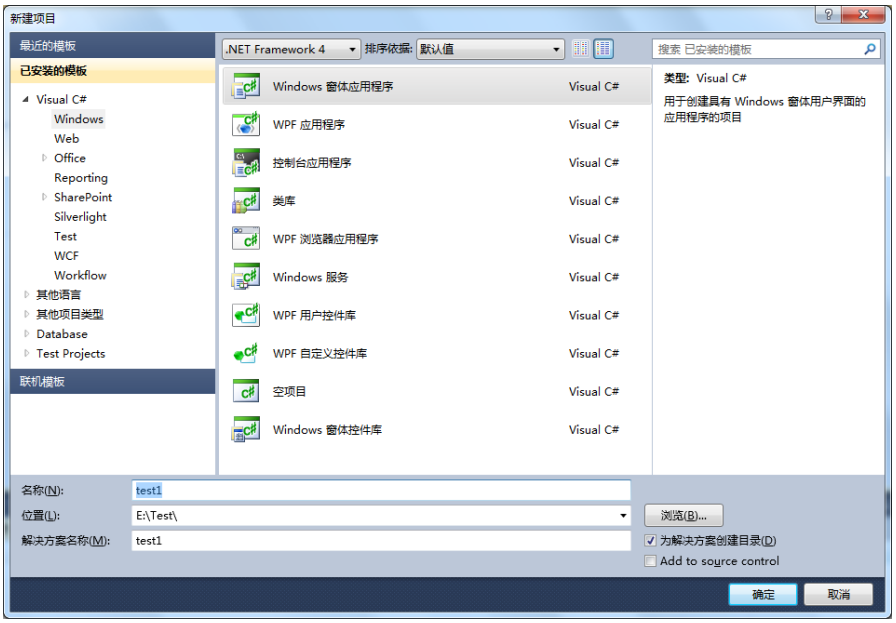


图 6.12　新建 Windous 窗体应用程序

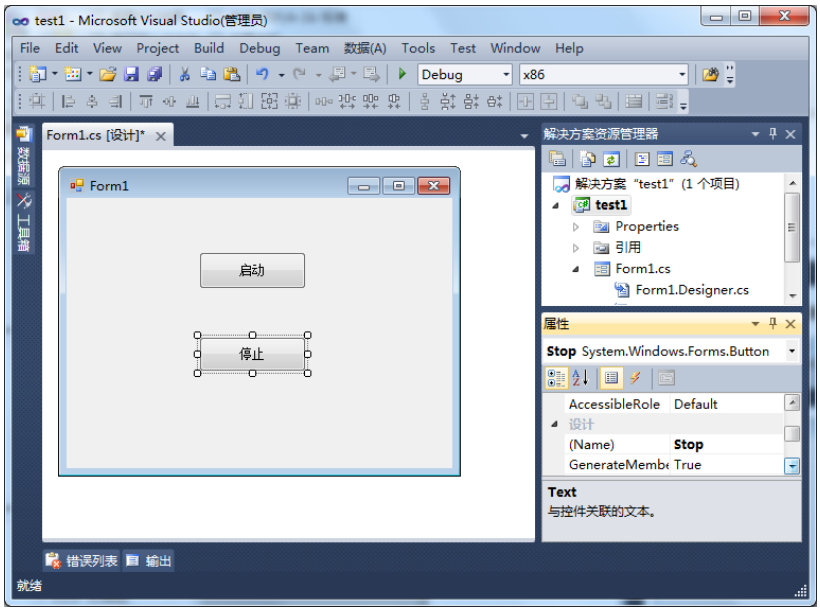


图 6.13　修改对话框

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、PVT.dll 和 LTDMC.cs 文件，将 LTDMC.dll、PVT.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 6.14 所示。

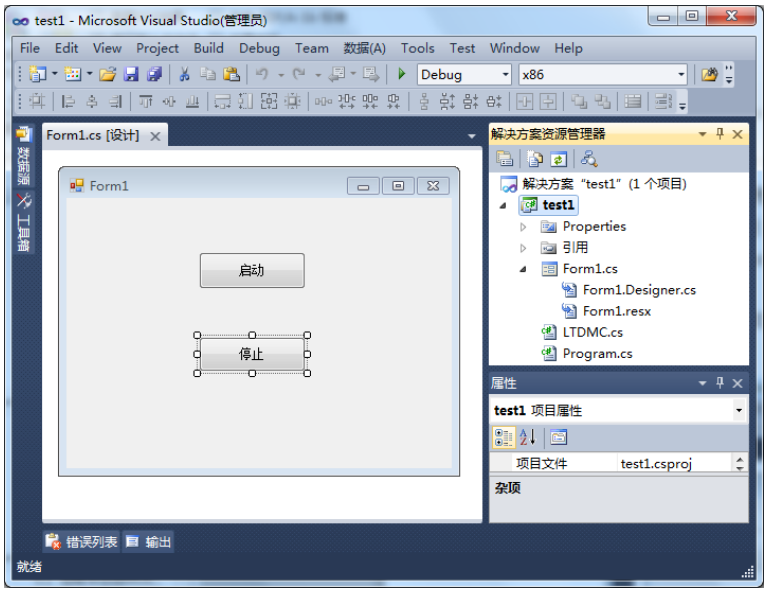


图 6.14　添加头文件

6）在代码文件开头处添加控制卡的命名空间：using csLTDMC;如图 6.15 所示。

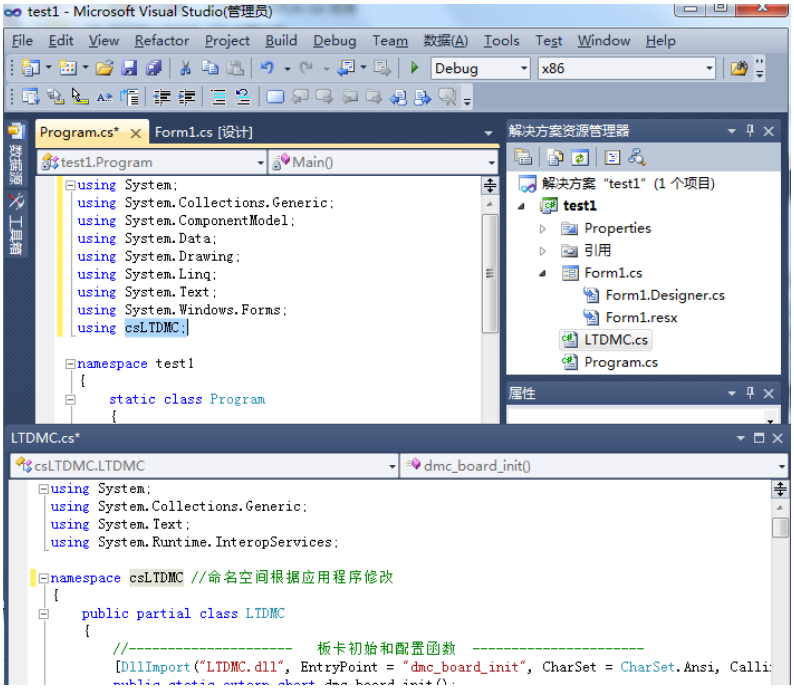


图 6.15　添加控制卡的命名空间

7）设定控制卡卡号为 3，如图 6.16 所示，双击窗口控件，在 Form1_Load 事件中添加代码 LTDMC.dmc_board_init()；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);
```

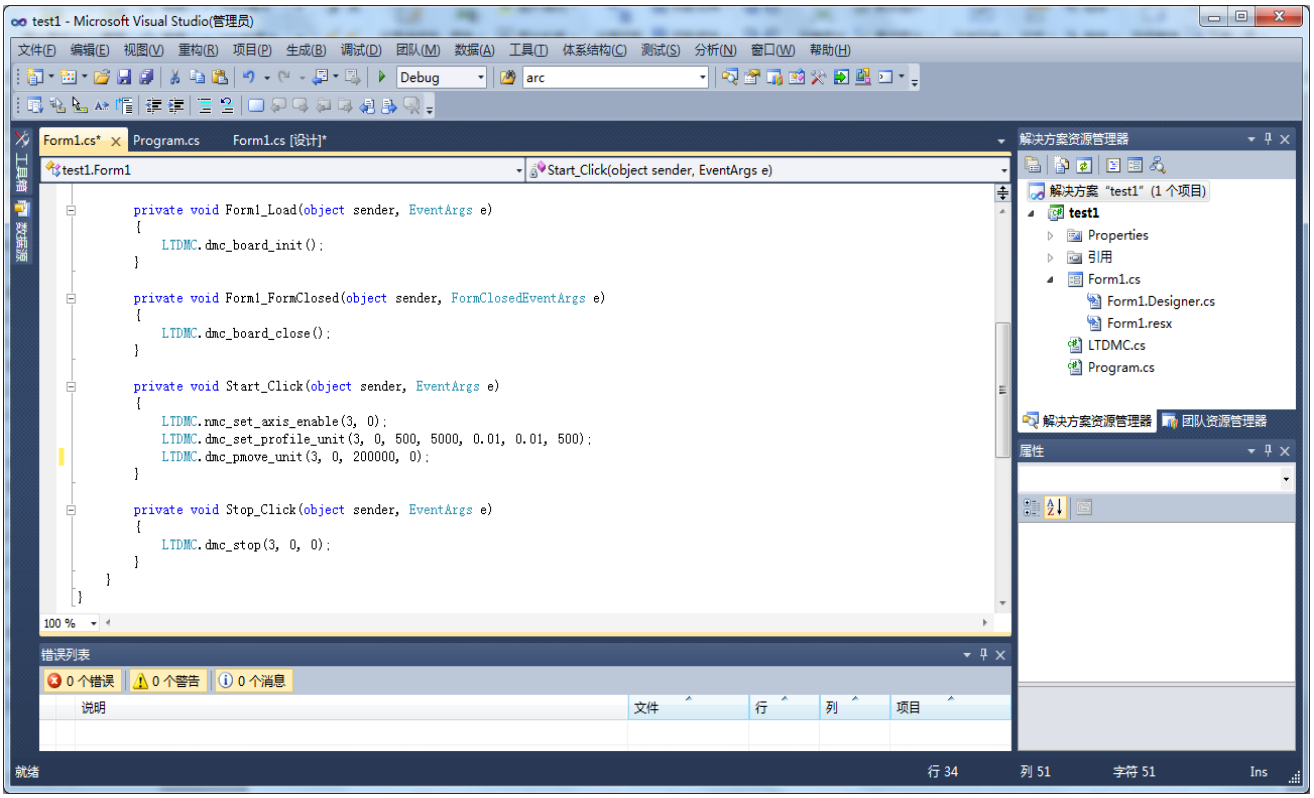


图 6.16　程序中调用运动控制卡库函数

7）程序编写完成。运行程序，显示界面如图 6.17 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。

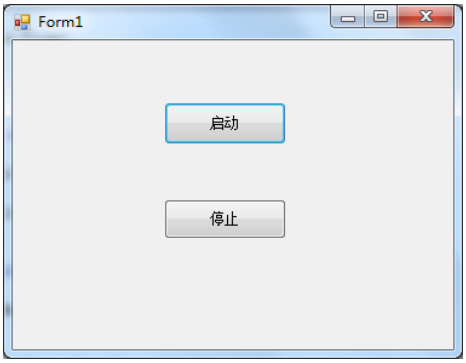


图 6.17　程序运行界面

第 7 章　DMC-E3032 的基本功能实现方法

本章介绍采用 C＃语言通过调用相关函数实现 DMC-E3032 卡的基本功能方法。

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

7.1　板卡初始化

板卡初始化设置相关函数如下表 7.1 所示。

表 7.1　板卡初始化及脉冲输出模式设置相关函数说明

名称	功能	参考
dmc_board_init	控制卡初始化函数	9.1 节
dmc_board_close	控制卡关闭函数	

在操作运动控制卡之前，必须调用函数 dmc_board_init 为运动控制卡分配资源。同样，当程序结束对运动控制卡的操作时，必须调用函数 dmc_board_close 释放运动控制卡所占用的 PC 系统资源，使得所占资源可被其它设备使用。

7.2　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。在使用运动控制卡进行运动控制之前，必须保证停止命令的有效性。

DMC-E3032 卡提供了急停设置函数 dmc_stop 来设置急停功能。

相关函数如下表 7.2 所示。

表 7.2 急停开关设置相关函数说明

名称	功能	参考
dmc_emg_stop	设置急停	9.8 节
dmc_stop	读取急停或减速停	

例程 7.1：紧急停止前 8 轴

```
……
ushort CardNo,Axis,IoType, MapIoType, MapIoIndex,Myenable,Mylogic;
```

```
CardNo=0;                    //卡号
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
}
……
```

7.3 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。DMC-E3032 卡提供了回原点运动的相关函数，参数传输到驱动器，主卡发送启动命令，由驱动器完成回零功能。

相关函数如下表 7.3 所示。

表 7.3 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	

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；

例程 7.2： 方式 1 回原点

```
……
ushort MyCardNo,Myaxis,Mymode;
MyCardNo=0;                  //卡号
Myaxis=0;                    //轴号
Mymode=1;                    //回零方式为 1
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();
```

```
}
……
```

7.4 点位运动的实现

DMC-E3032 卡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 7.1 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。

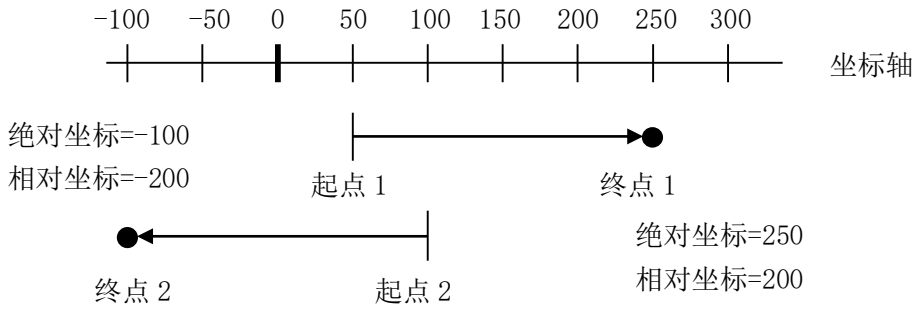


图 7.1 绝对坐标与相对坐标中轨迹终点的不同表达方式

DMC-E3032 卡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 7.4 所示。

表 7.4 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	9.3 节
dmc_set_s_profile	设置单轴速度曲线 S 段参数值	9.3 节
dmc_pmove_unit	定长运动	9.4 节
dmc_check_done	检测指定轴的运动状态	9.8 节

例程 7.3：执行点位运动

```
……
ushort MyCardNo, Myaxis, Myposi_mode;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;
int MyDist;
MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, Myaxis, 100);          //设置脉冲当量为 100pulse/unit
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 7.2。若在减速时改变目标位置，电机的速度变化曲线参见图 7.3。

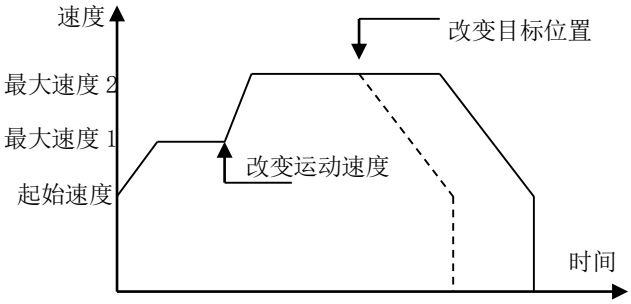


图 7.2　改变速度及改变目标位置

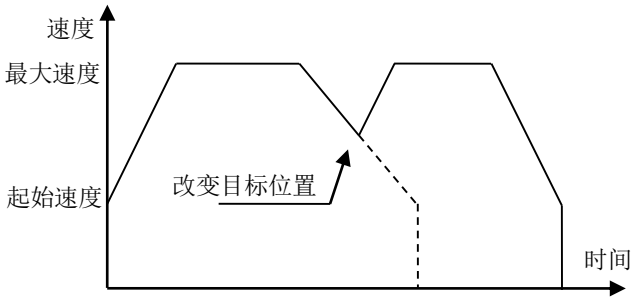


图 7.3　减速时改变目标位置

实现这 2 个功能的函数如表 7.5 所示。

表 7.5　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	9.4 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。此时控制卡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卡新计算的值所覆盖，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。

例程 7.4：点位运动中改变速度、改变终点位置

```
……
ushort MyCardNo, Myaxis, Myposi_mode, Mys_mode;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;
int MyDist;

MyCardNo = 0;                                          //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, Myaxis, 100);            //设置脉冲当量为 100pulse/unit
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 型速度曲线
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_position(MyCardNo, Myaxis, 100000, 0);//目标位置变为 100000unit
}
……
```

7.5 连续运动的实现

连续运动模式中，DMC-E3032 卡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 7.6 所示。

表 7.6 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	9.4 节
dmc_stop	指定轴停止运动	9.8 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 7.4 和图 7.5 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。

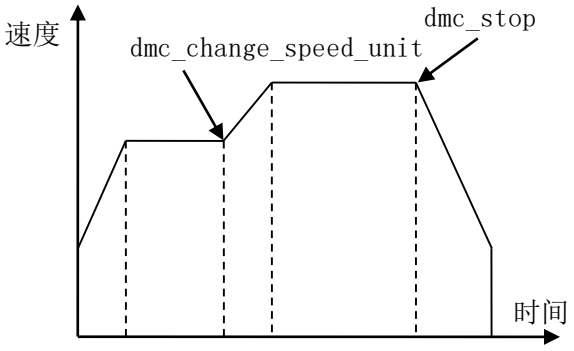


图 7.4 梯形运动中变速

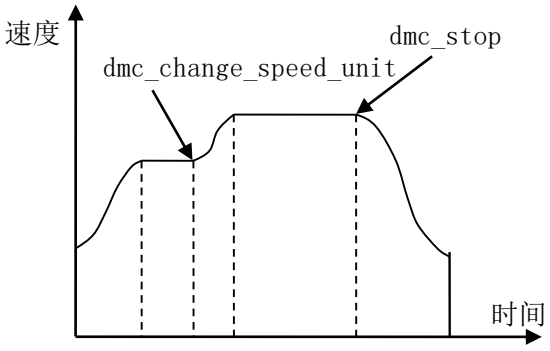


图 7.5 S 形运动中变速

例程 7.5：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;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, Mys_para;

MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, Myaxis, 100);      //设置脉冲当量为 100pulse/unit
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
}
……
```

7.6 异常减速停止时间设置功能的实现

DMC-E3032 卡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 7.14 所示。

表 7.14 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	9.18 节
dmc_get_dec_stop_time	读取减速停止时间设置	

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 7.10：设置异常减速停止时间为 0.5 秒

```
……
ushort MyCardNo, Myaxis;
double Stop_time;

MyCardNo=0;                                                  //卡号
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
……
```

7.7 手轮运动功能的实现

7.7.1 单轴手轮运动功能

DMC-E3032 卡支持单轴手轮运动功能。该功能允许用户设置手轮通道对应一个运动轴进行运动。相关函数如表 7.15 所示。

表 7.15 单轴手轮运动功能相关函数说明

名称	功能	参考
dmc_set_handwheel_inmode	设置单轴手轮运动控制输入方式	9.12 节
dmc_handwheel_move	启动手轮运动	

注意：当启动手轮运动后，只有发送 dmc_stop 或 dmc_emg_stop 命令才会退出手轮模式。

例 7.11：单轴手轮运动

```
……
ushort MyCardNo, Myaxis,Myinmode;
int Mymulti;
double vh;
MyCardNo=0;                                    //卡号
Myaxis=0;                                      //轴号
Myinmode = 0;                                  //手轮输入方式为 AB 相
Mymulti=10;                                    //手轮倍率为 10，方向正
vh=0;                                          //保留参数 0
LTDMC.dmc_set_handwheel_inmode(MyCardNo, Myaxis, Myinmode, Mymulti, vh);
                                               //设置手轮运动控制输入方式
LTDMC.dmc_handwheel_move(MyCardNo, Myaxis);//启动手轮运动
……
```

7.7.2 多轴手轮运动功能

DMC-E3032 卡支持多轴手轮运动功能。该功能允许用户设置手轮通道对应多个运动轴进行运动。相关函数如表 7.16 所示。

表 7.16 多轴手轮运动功能相关函数说明

名称	功能	参考
dmc_set_handwheel_inmode_extern	设置多轴手轮运动控制输入方式	9.13 节
dmc_handwheel_move	启动手轮运动	

注 意：当启动手轮运动后，只有发送 dmc_stop 或 dmc_emg_stop 命令才会退出手轮模式。

例 7.12：多轴手轮运动

```
……
ushort MyCardNo,Myinmode,MyAxisNum;
ushort []MyAxisList=new ushort [3];
MyAxisList[0]=0;
MyAxisList[1]=2;
MyAxisList[2]=5;                                    //设置参与手轮运动的轴数
int []Mymulti=new int [3];
Mymulti[0]=1;
Mymulti[1]=10;
Mymulti[2]=10;                                      //设置轴的倍率

MyCardNo = 0;                                       //卡号
Myinmode = 0;                                       //手轮输入方式为 AB 相
MyAxisNum=3;                                        //参与手轮运动的轴数
LTDMC.dmc_set_handwheel_inmode_extern(MyCardNo, Myinmode, MyAxisNum, MyAxisList, Mymulti);
                                                    //设置多轴手轮运动控制输入方式
LTDMC.dmc_handwheel_move(MyCardNo, MyAxisList[0]);
LTDMC.dmc_handwheel_move(MyCardNo, MyAxisList[1]);
LTDMC.dmc_handwheel_move(MyCardNo, MyAxisList[2]);//启动手轮运动
……
LTDMC.dmc_emg_stop(MyCardNo);                       //停止手轮运动
……
```

7.8 编码器检测的实现

DMC-E3032 卡的反馈位置计数器是一个 32 位有符号计数器，对通过控制卡辅助编码器接口 EA，EB 输入的脉冲（如编码器、光栅尺反馈脉冲等）进行计数，支持直接读取以 unit 为单位的编码器计数值。相关函数如表 7.17 所示。

表 7.17 编码器检测相关函数说明

名称	功能	参考
dmc_set_encoder_unit	设置当前编码器计数值	9.13 节
dmc_get_encoder_unit	读取当前编码器计数值	
dmc_set_extra_encoder_mode	设置辅助编码器计数模式	
dmc_get_extra_encoder_mode	读取辅助编码器计数模式	
dmc_set_extra_encoder	设置辅助编码器计数值	
dmc_get_extra_encoder	读取辅助编码器计数值	

例程 7.10：编码器检测

```
……
ushort MyCardNo, Myaxis, Mymode;
double Myencoder_value, MyX_Position;

MyCardNo = 0;                                          //卡号
Myaxis = 0;                                            //轴号
Myencoder_value = 0;                                   //设置 0 号轴的计数初始值为 0
LTDMC.dmc_set_encoder_unit(MyCardNo, Myaxis, Myencoder_value);
                                                       //设置 0 号轴的计数初始值
MyX_Position = 0;                                      //C＃中使用未赋值的变量会报错
LTDMC.dmc_get_encoder_unit(MyCardNo, Myaxis, ref MyX_Position);
                                                     //读轴0的计数器数值至变量MyX_Position
……
```

7.9 检测轴到位状态功能的实现

DMC-E3032 系列卡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 7.18 所示。

表 7.18 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	9.18 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 7.14：检测轴到位状态

```
……
ushort MyCardNo, Myaxis;
double Myfactor;
int Myerror;

MyCardNo=0;                                    //卡号
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
LTDMC.dmc_pmove_unit(MyCardNo, Myaxis, 1000, 1);
                                                     //定长运动，位移为为 1000 pulse、绝对模式
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(pulse)

1000-995=5(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。

7.10 通用 I/O 控制的实现

7.10.1 I/O 普通控制功能

用户可以使用 DMC-E3032 卡上的数字 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 7.19 所示。

表 7.19 通用 IO 普通控制功能相关函数说明

名称	功能	参考
dmc_read_inbit	读取指定控制卡的某一位输入口的电平状态	9.11 节

名称	功能	参考
dmc_write_outbit	对指定控制卡的某一位输出口置位	
dmc_read_outbit	读取指定控制卡的某一位输出口的电平状态	
dmc_read_inport	读取指定控制卡的全部输入口的电平状态	
dmc_read_outport	读取指定控制卡的全部输出口的电平状态	
dmc_write_outport	设置指定控制卡的全部输出口的电平状态	

注意：在使用 dmc_write_outport 对运动控制卡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。

例程 7.11：读取第 0 号卡的通用输入口 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;                          //卡号
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(MyCardNo, MyInbitno);
                                     //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 7.16：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 MyOutportValue;

MyCardNo=0;                           //卡号
Myportno=0;                           //IO 口组号
MyOutportValue = 0xFFFFFBFA;
          //(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);
                                     //读取指定控制卡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, MyOutportValue);//对全部输出口进行电平赋值
……
```

7.10.2 I/O 延时翻转功能

DMC-E3032 卡支持 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 7.20 所示。

表 7.20 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	9.11 节

例 7.17：对通用输出 IO 端口 0 进行延时翻转动作

```
……
ushort MyCardNo,MyOutbitno;
double MyDelayTime;
MyCardNo=0;      //卡号
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
……
```

7.10.3 I/O 计数功能

DMC-E3032 运动控制卡支持输入 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 7.21 所示。

表 7.21 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	9.11 节
dmc_set_io_count_value	设置 IO 计数值	
dmc_get_io_count_value	读取 IO 计数值	

例 7.18：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;//卡号
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
……
```

7.11 位置比较功能的实现

DMC-E3032 卡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；

3、添加/更新比较位置点；

4、开始运动并查看比较状态。

7.11.1 一维低速位置比较功能

DMC-E3032 卡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 7.22 所示。

表 7.22 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	9.15 节
dmc_compare_clear_points	清除一维位置比较点	
dmc_compare_add_point_cycle	添加一维位置比较点	
dmc_compare_get_current_point	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 7.19：一维低速位置比较

```
……
ushort MyCardNo,Myaxis,Myenable,Mycmp_source,Mydir,Myaction ;
int Mypos;
uint Myactpara;

MyCardNo=0;                          //卡号
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 10000pulse
Mydir = 1;                           //设置比较模式为大于等于
Myaction = 3;                        //设置触发功能为 IO 电平取反
Myactpara = 0;                       //设置输出 IO 端口 0 触发功能
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_position(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(MyCardNo, Myaxis, Mypos, Mydir, Myaction, Myactpara);
              //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 电平取反
LTDMC.dmc_compare_add_point(MyCardNo, Myaxis, 30000, 1, 1, 3);
              //添加比较点，位置 30000pulse，模式大于等于，触发时动作为输出端口 3 置高电平
LTDMC.dmc_set_profile _unit (MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);
              //设置梯形速度曲线
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 电平将翻转；当运动到 30000 时，通用输出口 3 输出高电平。

7.11.2 一维高速位置比较功能

DMC-E3032 卡共有六个高速位置比较器，每个高速位置比较器均配有 1 个硬件位置比较输出接口。DMC-E3032 卡的 CMP0-CMP5 端口（对应 OUT2-OUT7）电路使用的都是高速光耦（1MHz）。

每个 CMP 输出端口都可与辅助编码器关联，支持多种比较模式，用户只需在函数里设置便可以使用，非常灵活方便。

其中“等于”、“小于”、“大于”比较模式为单次比较模式。“队列”模式提供 127 个比较点，采用先添加先比较，比较完可追加比较点，也可一次性添加多个比较点。“线性”模式适

用于每个比较点位置都是按照线性增量依次增加的情况，用户只需要设置起始比较点位置以及线性增量值即可，非常方便。位置间时间间隔最小可达几微秒。

相关函数如表 7.23 所示。

表 7.23　一维高速位置比较相关函数说明

名称	功能	参考
dmc_hcmp_set_mode	设置高速比较模式	9.16 节
dmc_hcmp_set_config	配置高速比较器	
dmc_hcmp_set_liner	设置高速比较线性模式参数	
dmc_hcmp_clear_points	清除高速位置比较点	
dmc_hcmp_add_point	添加/更新高速比较位置	
dmc_hcmp_get_current_state	读取高速比较参数	
dmc_write_cmp_pin	控制指定 CMP 端口的输出	
dmc_read_cmp_pin	读取指定 CMP 端口的电平	

注意：（1）每个比较器的位置比较都是独立进行的。

（2）在队列及线性比较模式中，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被触发的。

例 7.21：单次比较模式（大于）

```
……
ushort MyCardNo,Myaxis,Mycmp_mode,Myhcmp,Mycmp_source,Mycmp_logic;
int MyTime;

MyCardNo=0;                          //卡号
Myaxis=0;                            //
Mycmp_mode = 3;                      //比较模式为模式 3
Myhcmp=0;                            //高速比较器，对应硬件 CMP 端口
Mycmp_source=1;                      //比较源为辅助编码器位置
Mycmp_logic=0;                       //低电平有效
MyTime=0;                            //脉冲宽度,只有比较模式为 4 或 5 时起作用
LTDMC.dmc_hcmp_set_mode(MyCardNo, Myhcmp, Mycmp_mode);//设置高速比较模式
LTDMC.dmc_hcmp_set_config(MyCardNo, Myhcmp, 0, Mycmp_source, Mycmp_logic, MyTime);
                                                    //配置高速比较器
LTDMC.dmc_hcmp_clear_points(MyCardNo, Myhcmp);     //清除比较点
LTDMC.dmc_hcmp_add_point(MyCardNo, Myhcmp, 5000);     //添加高速比较位置为 50000 pulse
……
```

运行结果：

当辅助编码器位置超过 50000 pulse 时，CMP0 端口输出低电平并保持。

例 7.22：队列比较模式

```
……
ushort MyCardNo, Myaxis, Mycmp_mode, Myhcmp, Mycmp_source, Mycmp_logic;
int MyTime,MyCmpPos;

MyCardNo=0;                                    //卡号
Myaxis=0;                                      //
Mycmp_mode = 4;                                //比较模式为模式 4
Myhcmp=0;                                      //高速比较器，对应硬件 CMP 端口
Mycmp_source=1;                                //比较源为辅助编码器位置
Mycmp_logic=0;                                 //低电平有效
MyTime = 500000;                               //脉冲宽度 500000us
LTDMC.dmc_hcmp_set_config(MyCardNo, Myhcmp, Myaxis, Mycmp_source, Mycmp_logic, MyTime);
                                               //配置高速比较器
LTDMC.dmc_hcmp_clear_points(MyCardNo, Myhcmp);//清除比较点
LTDMC.dmc_hcmp_set_mode(MyCardNo, Myhcmp, Mycmp_mode);//设置高速比较模式为队列模式
MyCmpPos = 10000;
LTDMC.dmc_hcmp_add_point(MyCardNo, Myhcmp, MyCmpPos);//添加高速比较位置为 10000 pulse
MyCmpPos = 30000;
LTDMC.dmc_hcmp_add_point(MyCardNo, Myhcmp, MyCmpPos);//添加高速比较位置为 30000 pulse
MyCmpPos = 70000;
LTDMC.dmc_hcmp_add_point(MyCardNo, Myhcmp, MyCmpPos);//添加高速比较位置为 70000 pulse
……
```

运行结果：

当辅助编码器经过位置 10000 pulse、30000 pulse、70000 pulse 时，CMP0 端口输出低电平，低电平持续时间为 500ms。

例 7.23：线性比较模式

```
……
ushort MyCardNo, Myaxis, Mycmp_mode, Myhcmp, Mycmp_source, Mycmp_logic;
int MyTime,MyCmpPos,MyCmpInc,MyCmpCount;

MyCardNo=0;                                    //卡号
Myaxis=0;                                      //
Mycmp_mode = 5;                                //比较模式为模式 5
Myhcmp=1;                                      //高速比较器，对应硬件 CMP 端口
Mycmp_source=1;                                //比较源为辅助编码器位置
Mycmp_logic=0;                                 //低电平有效
MyTime = 500000;                               //脉冲宽度 500000us
MyCmpInc=20000;                                //线性增量 20000 pulse
MyCmpCount=5;                                  //比较次数 5
LTDMC.dmc_hcmp_set_config(MyCardNo, Myhcmp, Myaxis, Mycmp_source, Mycmp_logic, MyTime);
                                               //配置高速比较器
```

```
LTDMC.dmc_hcmp_clear_points(MyCardNo, Myhcmp); //清除比较点
LTDMC.dmc_hcmp_set_mode(MyCardNo, Myhcmp, Mycmp_mode);//设置高速比较模式为线性比较模式
LTDMC.dmc_hcmp_set_liner(MyCardNo, Myhcmp, MyCmpInc, MyCmpCount);
                                                    //设置高速比较线性模式参数
MyCmpPos = 10000;
LTDMC.dmc_hcmp_add_point(MyCardNo, Myhcmp, MyCmpPos);//添加高速比较位置为 10000 pulse

……
```

运行结果：

当辅助编码器经过位置 10000、30000、50000、70000、90000 pulse 时，CMP1 端口输出低电平，低电平持续时间为 500ms。

7.12 高速位置锁存功能的实现

7.12.1 高速单次位置锁存功能

DMC-E3032 卡提供了高速单次位置锁存功能，位置锁存无触发延时时间，当捕获到位置锁存信号后立即锁存当前辅助编码器位置。DMC-E3032 支持 4 个锁存器，对应硬件端口号为 IN4-IN7，相关函数如表 7.24 所示。

表 7.24 高速单次锁存相关函数说明

名称	功能	参考
dmc_ltc_set_mode	配置锁存器	9.16 节
dmc_ltc_get_mode	读取锁存器配置	
dmc_ltc_set_source	配置锁存源	
dmc_ltc_get_source	读取锁存源配置	
dmc_ltc_get_value_unit	读取锁存值	
dmc_ltc_get_number	读取锁存器已锁存个数	
dmc_ltc_reset	复位指定卡的锁存器	

注意：在单次锁存中，多次触发高速锁存口只锁存第一次触发位置，只有调用函数清除锁存状态方可再次锁存。

例 7.24：高速单次位置锁存

```
……

ushort MyCardNo, Myaxis, Myltc_mode;
ushort Mylatch_source, Latch;
```

```
double Myfilter;
int My_latch_Value;
int number;

number = 0;                                    //锁存个数
My_latch_Value = 0;                            //锁存值
MyCardNo = 0;                                  //卡号
Myaxis = 0;                                    //固定 0
Myltc_logic = 0;                               //设置 LTC 触发方式为下降沿触发
Myltc_mode = 0;                                //设置锁存模式为单次锁存
Myfilter = 0;                                  //滤波时间
Mylatch_source = 1;                            //锁存源为辅助编码器位置

LTDMC.dmc_ltc_set_mode(MyCardNo, Latch, Myltc_mode, Myltc_logic, Myfilter);
                //配置锁存器 0，单次锁存模式，LTC 下降沿触发
LTDMC.dmc_ltc_set_source (MyCardNo, Latch ,Myaxis, Mylatch_source);
                //设置锁存源为辅助编码器位置
LTDMC.dmc_ltc_reset(MyCardNo, Latch);          //复位锁存器
LTDMC.dmc_set_profile_unit(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);
                                               //设置梯形速度曲线
LTDMC.dmc_pmove_unit (MyCardNo, Myaxis, 100000, 1); //定长运动，位移为100000，绝对模式
LTDMC.dmc_ltc_get_number(MyCardNo, Latch, Myaxis,ref number); // 读取锁存个数
while (number == 0) ;// 判断 0 号轴 LTC 锁存状态，若个数不为零，已锁存数据
{
    Application.DoEvents();
}
LTDMC. dmc_ltc_get_value_unit (MyCardNo, Latch, Myaxis,ref  My_latch_Value);
                        //读取锁存值，并赋值给变量 My_latch_Value
……
```

7.12.2 高速连续位置锁存功能

DMC-E3032 系列卡提供了高速连续位置锁存功能，位置锁存无触发延时时间，当捕获到位置锁存信号后立即锁存当前位置；但相邻两个锁存点之间的间隔时间应大于 10ms。

相关函数如表 7.25 所示。

表 7.25 高速连续锁存相关函数说明

名称	功能	参考
dmc_ltc_set_mode	配置锁存器	9.16 节
dmc_ltc_get_mode	读取锁存器配置	
dmc_ltc_set_source	配置锁存源	
dmc_ltc_get_source	读取锁存源配置	
dmc_ltc_get_value_unit	读取锁存值	
dmc_ltc_get_number	读取锁存器已锁存个数	
dmc_ltc_reset	复位指定卡的锁存器	

注意：在连续锁存，多次触发高速锁存口锁存所有的触发位置，超过 1000 次剔除最先的触发位置保存最新的触发位置。在每次锁存后，不需要调用函数清除锁存状态。但是在启动高速连续位置锁存之前，必须调用函数清除锁存状态。

例 7.25：高速连续位置锁存

```
……
ushort MyCardNo, Myaxis, Myltc_logic, Myltc_mode;
ushort Mylatch_source,Latch;
double Myfilter;
int number;
double[] My_latch_Value = new double[1000];

number = 0;                                    //锁存个数
MyCardNo = 0;                                  //卡号
Latch = 0;                                     //锁存器 0
Myaxis = 0;                                    //轴号
Myltc_logic = 0;                               //设置 LTC 触发方式为下降沿触发
Myltc_mode = 1;                                //设置锁存模式为连续锁存
Myfilter = 0;                                  //滤波时间

LTDMC.dmc_ltc_set_mode(MyCardNo, Latch, Myltc_mode, Myltc_logic, Myfilter);
                    //配置锁存器 0，连续锁存模式，LTC 下降沿触发
LTDMC. dmc_ltc_set_source (MyCardNo, Latch ,Myaxis, Mylatch_source);
                    //设置锁存源为辅助编码器位置
LTDMC.dmc_ltc_reset(MyCardNo, Latch);          //复位锁存器
```

```
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);//设置梯形速度曲线
LTDMC.dmc_pmove_unit(MyCardNo, Myaxis, 100000, 1); //定长运动，位移为 100000，绝对模式
while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0)
{
    Application.DoEvents();
}
My_latch_flag = LTDMC.dmc_get_latch_flag_extern(MyCardNo, Myaxis);
                    //读取锁存器已锁存个数，并赋值给变量 My_latch_flag
LTDMC.dmc_ltc_get_number(MyCardNo, Latch, Myaxis,ref number); // 读取锁存个数
for (int i = 0; i < number; i++)
{
    LTDMC.dmc_ltc_get_value_unit (MyCardNo, Latch, Myaxis,ref  My_latch_Value[i]);
          //读取锁存值，并赋值给变量 My_latch_Value
}
```

7.13 脉冲当量的设置

DMC-E3032 卡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 7.26 所示。

表 7.26 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	9.2 节

注意：设置脉冲当量功能需在轴使能前完成。

例 7.27：某设备中运动控制卡发送 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;                                          //卡号
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 7.28：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;                                          //卡号
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
```

7.14 插补运动的实现

插补运动是为了实现轨迹控制，运动控制卡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。

7.14.1 直线插补运动

DMC-E3032 卡可以进行任意 2~16 轴直线插补，插补计算由控制卡的硬件执行，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 7.16 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ΔX：ΔY，二轴合成的矢量速度为：

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

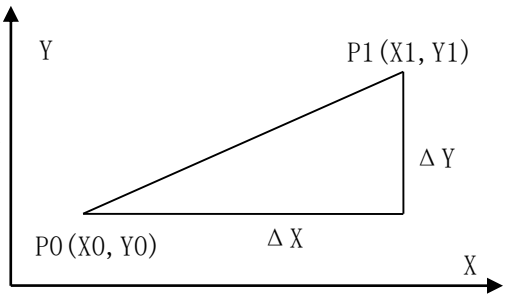


图 7.16 两轴直线插补

2. 三轴直线插补

如图 7.17 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ΔX：ΔY：ΔZ，三轴合成的矢量速度为：

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

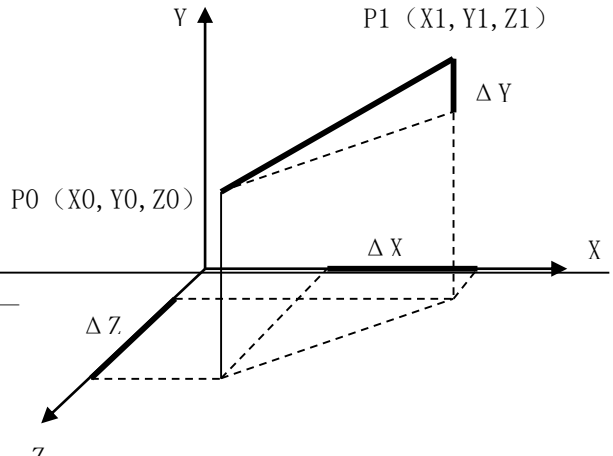


图 7.17　三轴直线插补

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。

DMC-E3032 总线卡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，DMC-E3032 总线卡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。插补计算由控制卡的硬件执行，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 7.27 所示。

表 7.27　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	9.5 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	9.6 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）。

例程 7.12： XY 轴直线插补

```
……
ushort MyCardNo, MyCrd;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;

MyCardNo = 0;                                          //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);                  //设置 0 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);                  //设置 1 号轴脉冲当量为 10pulse/unit
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();
}
……
```

例程 7.13：六轴直线插补

```
……
ushort MyCardNo, MyCrd;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;

MyCardNo = 0;                                          //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);                  //设置 0 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);                  //设置 1 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);                  //设置 2 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 3, 10);                  //设置 3 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 4, 10);                  //设置 4 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 5, 10);                  //设置 5 号轴脉冲当量为 10pulse/unit
```

```
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();
}
……
```

7.14.2 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，两轴同心圆插补运动。

7.14.2.1 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。

例程 7.14： XY 轴圆心圆弧插补

```
……
ushort MyCardNo, MyCrd;
double MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel;

MyCardNo = 0;                                   //卡号
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
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();
}
……
```

7.14.2.2 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 7.18 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。

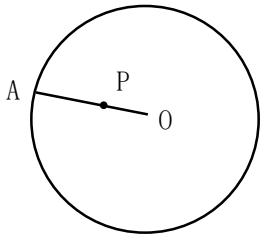


图 7.18 螺旋线插补基本原理

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 7.32： 两轴绽放螺旋线插补

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode;
int MyCircle;

MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);            //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);            //设置 Y 轴脉冲当量为 10pulse/unit
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 7.19 所示。

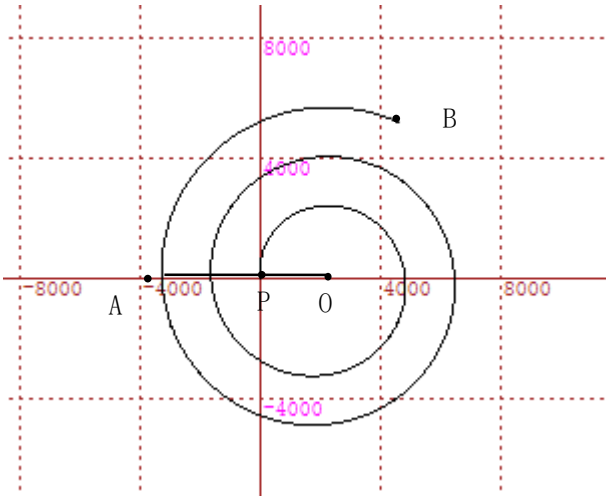


图 7.19　两轴绽放螺旋线插补

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 7.33：两轴收敛螺旋线插补

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode;
int MyCircle;

MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);           //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);           //设置 Y 轴脉冲当量为 10pulse/unit
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 7.20 所示。

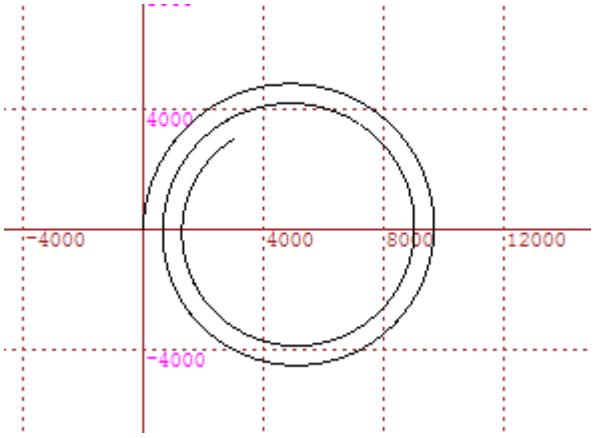


图 7.20　两轴收敛螺旋插补

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。

7.14.2.3　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 9.6 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卡的硬件执行，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。

例程 7.15：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;
int MyCircle;

MyCardNo = 0;                                   //卡号
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
```

```
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);          //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);          //设置 Y 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);          //设置 Z 轴脉冲当量为 10pulse/unit
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 7.21 所示。

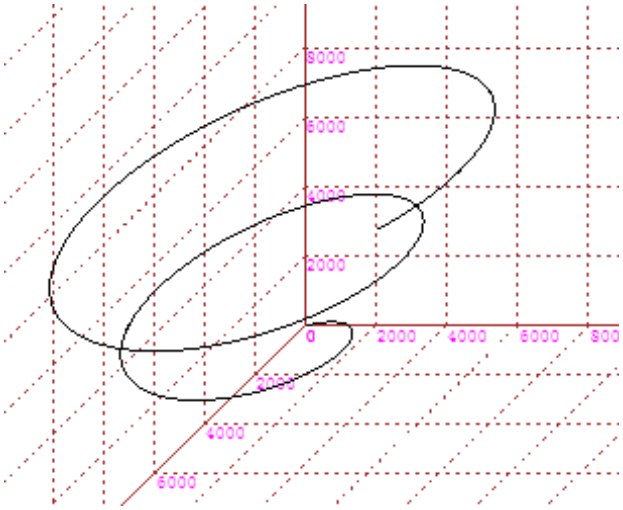


图 7.21　绽放螺旋插补

基于 XY 平面的运动轨迹如图 7.22 所示。

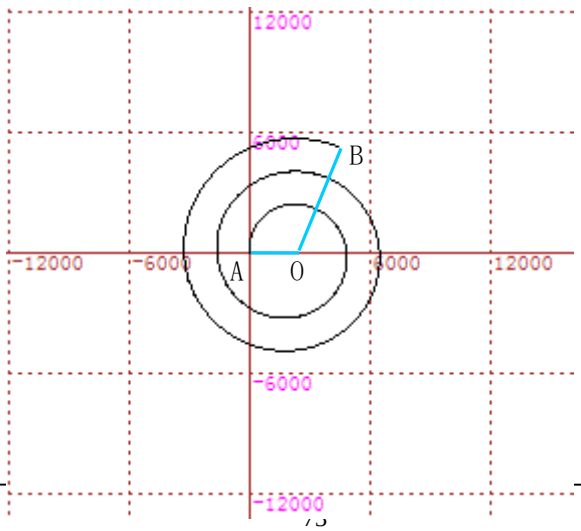


图 7.22　绽放螺旋插补基于 XY 平面的运动轨迹

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 8.2 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 9.6 节相关函数说明)。

例程 7.16：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;
int MyCircle;
MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);            //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);            //设置 Y 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);            //设置 Z 轴脉冲当量为 10pulse/unit
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 7.23 所示。

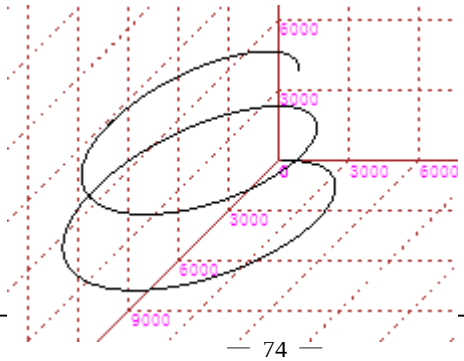


图 7.23　空间收敛螺旋线插补

基于 XY 平面的运动轨迹如图 7.24 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。

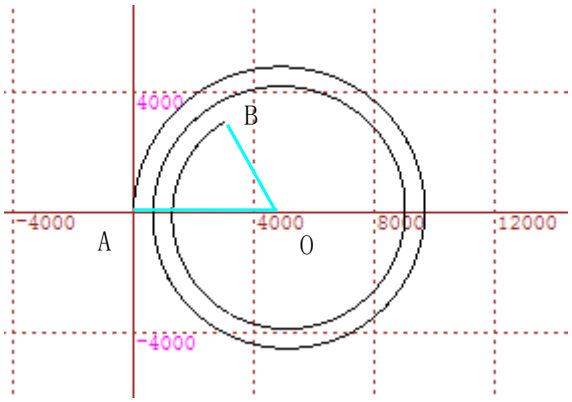


图 7.24　空间收敛螺旋线插补基于 XY 平面的运动轨迹

例程 7.17：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;
int MyCircle;

MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);            //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);            //设置 Y 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);            //设置 Z 轴脉冲当量为 10pulse/unit
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 7.25 所示。

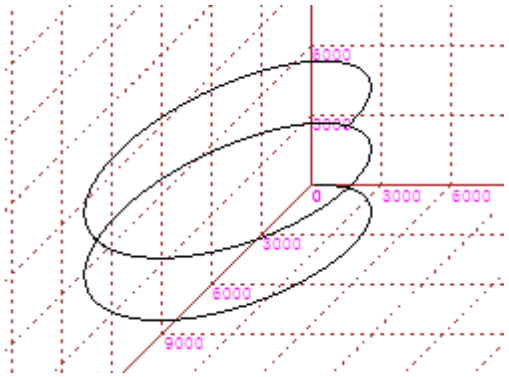


图 7.25　圆柱螺旋插补

基于 XY 平面的运动轨迹如图 7.26 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。

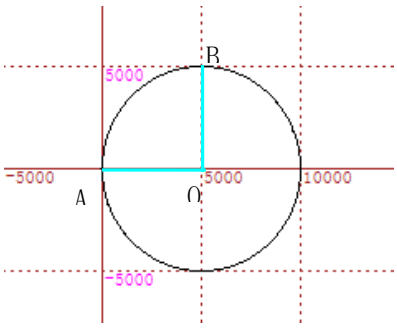


图 7.26　圆柱螺旋线插补基于 XY 平面的运动轨迹

7.14.2.4　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卡的硬件执行，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。

例程 7.18：六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;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);           //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);           //设置 Y 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);           //设置 Y 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 3, 10);           //设置 U 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 4, 10);            //设置 V 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 5, 10);            //设置 W 号轴脉冲当量为 10pulse/unit
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();
}
……
```

7.14.3 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。

7.14.3.1 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。

例程 7.19：两轴圆弧插补

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyArc_Dir, Myposi_mode;
double MyRadius;
int MyCircle;

MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);            //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);            //设置 Y 号轴脉冲当量为 10pulse/unit
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 7.28 所示。

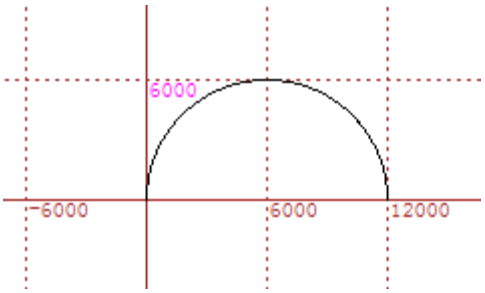


图 7.28　两轴圆弧插补

7.14.3.1　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 7.16.2.5 节插补模式下的辅助轴跟随）。

例程 7.20：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;
double MyRadius;
int MyCircle;
MyCardNo = 0;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);           //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);           //设置 Y 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);           //设置 Y 号轴脉冲当量为 10pulse/uni
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 7.29 所示。

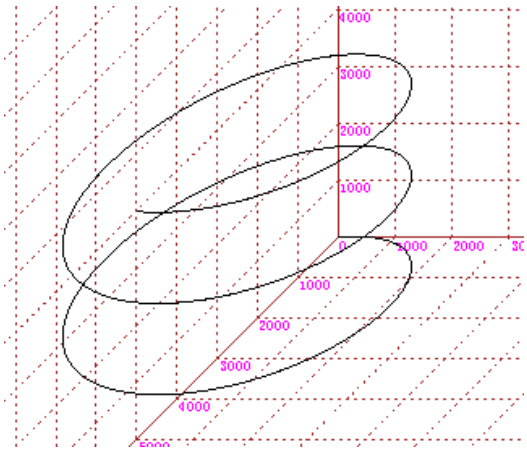


图 7.29　圆柱螺旋线插补

7.14.4 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 7.16.2.5 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 9.6 节插补运动）。

例程 7.21： 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;                                    //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);           //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);           //设置 Y 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);           //设置 Z 号轴脉冲当量为 10pulse/uni
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 7.30 所示。

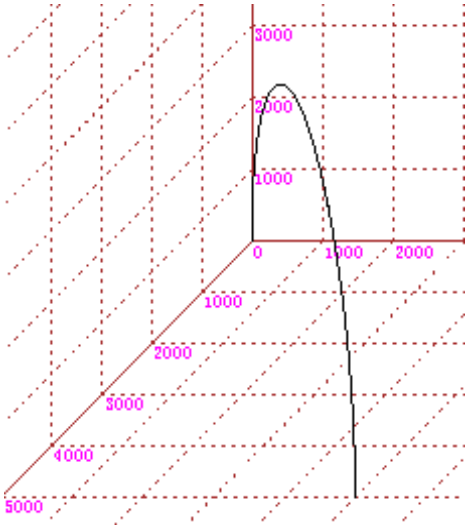


图 7.30　空间圆弧插补

例 7.42：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;                                          //卡号
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
LTDMC.dmc_set_equiv(MyCardNo, 0, 10);                  //设置 X 轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 10);                  //设置 Y 号轴脉冲当量为 10pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 2, 10);                  //设置 Y 号轴脉冲当量为 10pulse/uni
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 7.31 所示。

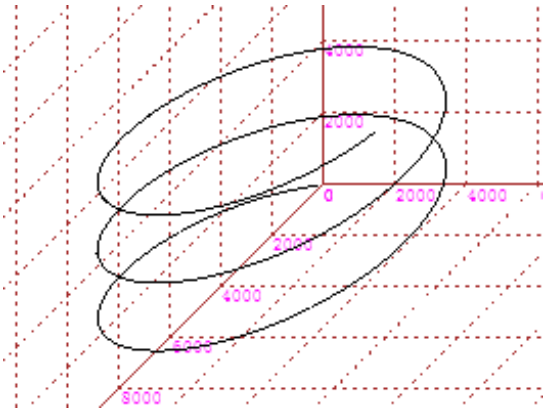


图 7.31 空间圆弧插补

7.15 过程数据缓存功能实现

过程数据（PDO）采集功能实现每周期采集当前的 TXPDO 和 RXPDO 数据。

该功能根据配置参数采集固定长度的数据，当达到设置的采集长度后不再采集新数据，但是可使用清除指令，或者重新启动采集指令实现再次采集。

采集后的数据包结构为：根据启动参数配置的采集对象的顺序保存每包数据，例如设置的参数顺序为 0x6040,0x6041,0x607A，其通过读取采集数据指令返回的数据包的排列即为 0x6040,0x6041,0x607A，并依次排列到最后一包数据。

对于特定触发条件的对象，可配置触发条件触发采集。其触发模式包括：等于，大于及小于三种模式；

相关函数如表 7.28 所示；

表 7.28 过程数据缓存相关函数说明

名称	功能	参考
nmc_start_pdo_trace	启动采集	8.5 节
nmc_get_pdo_trace	读取采集参数	
nmc_set_pdo_trace_trig_para	设置触发采集参数	
nmc_get_pdo_trace_trig_para	读取触发采集参数	
nmc_clear_pdo_trace_data	清除采集数据	
nmc_stop_pdo_trace	停止采集	
nmc_read_pdo_trace_data	读取采集数据结果	
nmc_get_pdo_trace_num	读取已采集的数据包个数	
nmc_get_pdo_trace_state	读取采集状态	

过程数据（PDO）采集功能使用步骤：

第一步：停止 PDO 采集，清空 PDO 采集数据；
第二步：设置采集的触发条件，满足触发条件后实现采集功能；触发对象，可为任何已配置的 PDO 通讯对象，触发模式可配置为：0 表示等于 trig_value 时触发；1 表示大于 trig_value 时触发；2 表示小于 trig_value 时触发；
第三步：设置采集的参数，添加需要采集的 PDO 数据 0x6060、0x6061、0x6040 、0x6041、0x607A 等；输入的所有索引对象的数据长度构成的数据包长度不超过 48bytes，采集包个数与单个数据包字节长度的乘积不能超过 48000bytes；设置需要采集从站的地址以及采集包个数，启动采集；
第四步：读取数据；可全部读取，也可部分读取；

功能具体使用可参照 pdo 缓存测试例程；

7.16 群组化运动控制功能实现

群组化运动控制支持用户算法数据下发至控制卡执行，从而实现用户特定轨迹需求。该方式用户需要填入每个总线通讯周期运动轴所需要的位置命令，可以实现多轴同时启动和停止，但是用户必须要有能力自行运算所有轴在做加减速时的位置命令。

群组化运动控制功能实现步骤：

第一步：停止群组化运动控制，设定群组化运动轴数及对应轴号；

第二步：添加对应运动轴位置命令数据至控制卡，单次添加数据点最多 100 点，最多可添加 1000 点；

第三步：启动群组化运动，可查看群组缓存剩余空间，动态添加数据；

第四步：监控群组化运动状态，可随时停止群组运动；

功能具体使用可参照群组化运动控制测试例程；

第 8 章　总线操作函数说明

8.1　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 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卡

参　数：CardNo　　控制卡卡号

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

8.2　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　　控制卡卡号

PortNum　　EtherCAT 端口号，固定为 2

CycleTime　　EtherCAT 总线循环周期，单位：us，支持 250/500/1000/2000

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

　　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　　控制卡卡号

　　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　　控制卡卡号

　　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　　控制卡卡号

　　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　　控制卡卡号

　　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　　控制卡卡号

　　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　　控制卡卡号

　　TotalIn　　EtherCAT 总线 IO 输入数，包括板卡自带以及 IO 模块上的输入

　　TotalOut　　EtherCAT 总线 IO 输出数，包括板卡自带以及 IO 模块上的输出

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　　控制卡卡号

　　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：设置控制卡工作模式

参　数：CardNo　　控制卡卡号

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：读取控制卡工作模式

参　数：CardNo　　控制卡卡号

controller_mode　　控制卡的工作模式，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　　控制卡卡号

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　　控制卡卡号

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 控制卡卡号

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 控制卡卡号

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 控制卡卡号

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　　控制卡卡号

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

8.3　总线轴控制函数

short nmc_set_axis_enable(WORD CardNo,WORD axis)

功　能：设置 EtherCAT 总线驱动器使能

参　数：CardNo　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号

axis　　EtherCAT 总线轴轴号

返回值：错误代码

short nmc_get_axis_io_in(WORD CardNo,WORD axis)

功　能：读取EtherCAT总线驱动器数字量输入状态

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

返回值：EtherCAT 总线驱动器数字量输入状态

short nmc_get_axis_io_out(WORD CardNo,WORD axis)

功　能：读取EtherCAT总线驱动器数字量输出状态

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

返回值：EtherCAT 总线驱动器数字量输出状态

short nmc_set_axis_io_out(WORD CardNo,WORD axis,DWORD iostate)

功　能：设置EtherCAT总线驱动器数字量输出状态

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

iostate　　EtherCAT 总线驱动器数字量输出状态

返回值：错误码

short nmc_get_axis_setting_contrlmode(WORD CardNo,WORD axis,long* contrlmode);

功　能：获取EtherCAT总线轴配置控制模式

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

contrlmode　　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　　控制卡卡号

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_contrlword(WORD CardNo,WORD axis,long* Contrlword);

功　能：获取EtherCAT总线轴控制字

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

statusword　　EtherCAT 总线轴控制字

返回值：错误码

short nmc_read_inbit_extern(WORD CardNo, WORD Channel，WORD NoteID, WORD IoBit,WORD* IoValue)

功　能：读取指定 EtherCAT 扩展模块的某个输入端口的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

IoBit　　输入端口号

IoValue　　指定输入端口的电平，0：低电平，1：高电平

返回值：错误代码

short nmc_read_outbit_extern(WORD CardNo, WORD Channel，WORD NoteID, WORD IoBit,WORD* IoValue)

功　能：读取指定 EtherCAT 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

IoBit　　输出端口号

IoValue　　指定输出端口的电平，0：低电平，1：高电平

返回值：错误代码

short nmc_write_outbit_extern(WORD CardNo, WORD Channel，WORD NoteID, WORD IoBit,WORD IoValue)

功　能：设置指定 EtherCAT 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

IoBit　　输出端口号

IoValue　　指定输出端口的电平，0：低电平，1：高电平

返回值：错误代码

short nmc_read_inport_extern(WORD CardNo, WORD Channel , WORD NoteID, WORD PortNo,WORD* IoValue)

功　能：读取指定 EtherCAT 扩展模块的输入口组的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

PortNo　　输入组号（32 位一组，从 0 开始，如 0 表示 0-31 位，1 表示 32-63 位）

IoValue　　输入组各输入端口的值，bit0~bit31 的值分别代表第 0~31 号输入口的电平，1：低电平，0：高电平

返回值：错误代码

short nmc_read_outport_extern(WORD CardNo, WORD Channel , WORD NoteID, WORD PortNo,WORD* IoValue)

功　能：读取指定 EtherCAT 扩展模块的输出口组的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

PortNo　　输出组号（32 位一组，从 0 开始，如 0 表示 0-31 位，1 表示 32-63 位）

IoValue　　输出组各输出端口的值，bit0~bit31 的值分别代表第 0~31 号输出口的电平，1：低电平，0：高电平

返回值：错误代码

short nmc_write_outport_extern(WORD CardNo, WORD Channel , WORD NoteID, WORD PortNo,WORD IoValue)

功　能：设置指定 EtherCAT 扩展模块的输出口组的电平

参　数：CardNo　　控制卡卡号

Channel　　总线通道号，固定为 2；

NoteID　　EtherCAT 地址，从 1001 开始，按从站数往后累加

PortNo　　输出组号（32 位一组，从 0 开始，如 0 表示 0-31 位，1 表示 32-63 位）

IoValue　　输出组各输出端口的值，bit0~bit31 的值分别代表第 0~31 号输出口的电平，1：低电平，0：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，不支持板卡自带 IO 以及驱动器本体 IO；

8.4　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　　控制卡卡号

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_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　　控制卡卡号

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　　控制卡卡号

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：回读 EtherCAT 总线轴回零状态

参　数：CardNo　　控制卡卡号

axis　　EtherCAT 总线轴轴号

state　　回零状态，1：回零完成，0：回零未完成

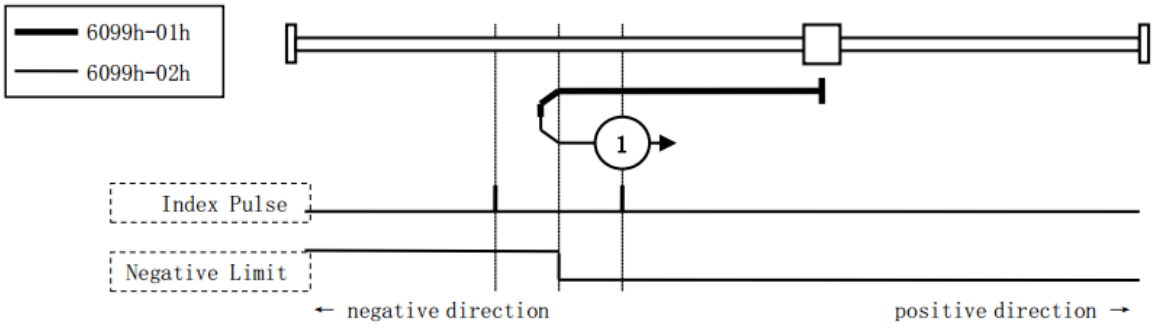
返回值：错误代码

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，其他家品牌 EtherCAT 总线驱动器回零模式请参照其手册描述。

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。

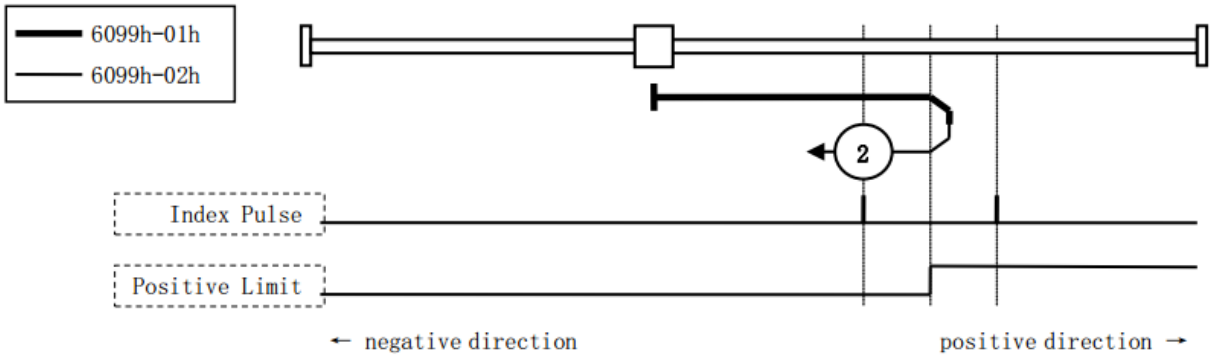


Homing on negative limit switch and index pulse

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。

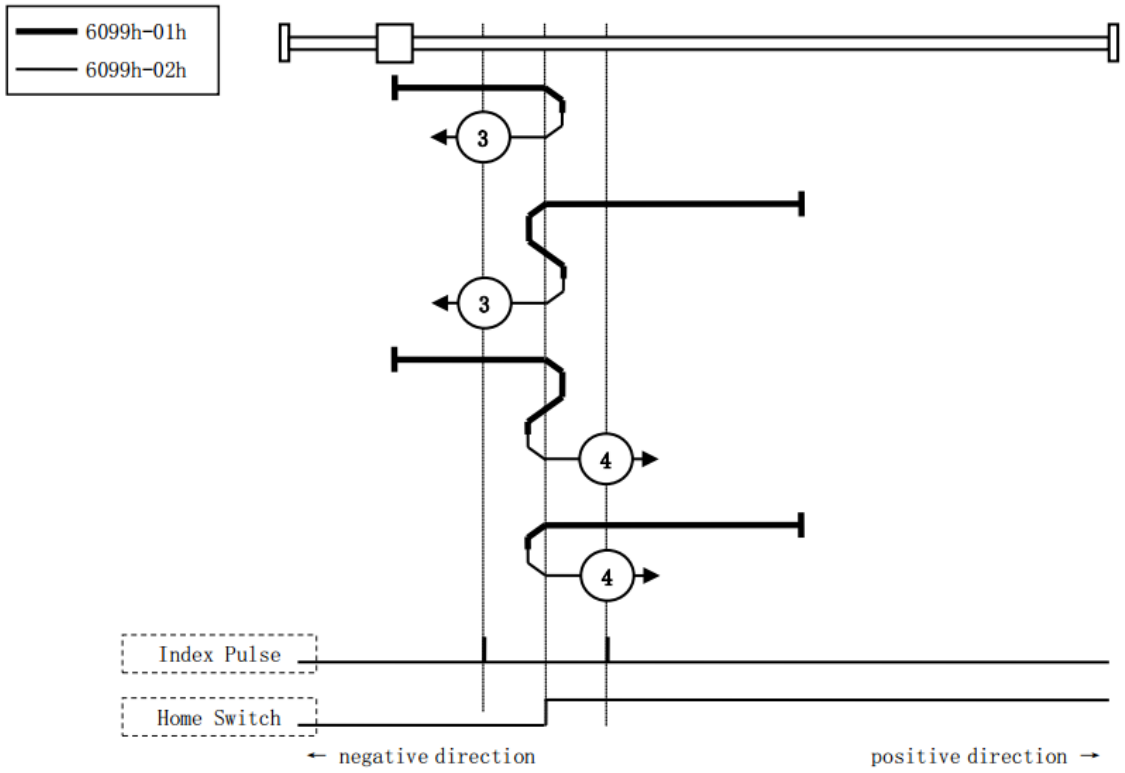


Homing on positive limit switch and index pulse

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。

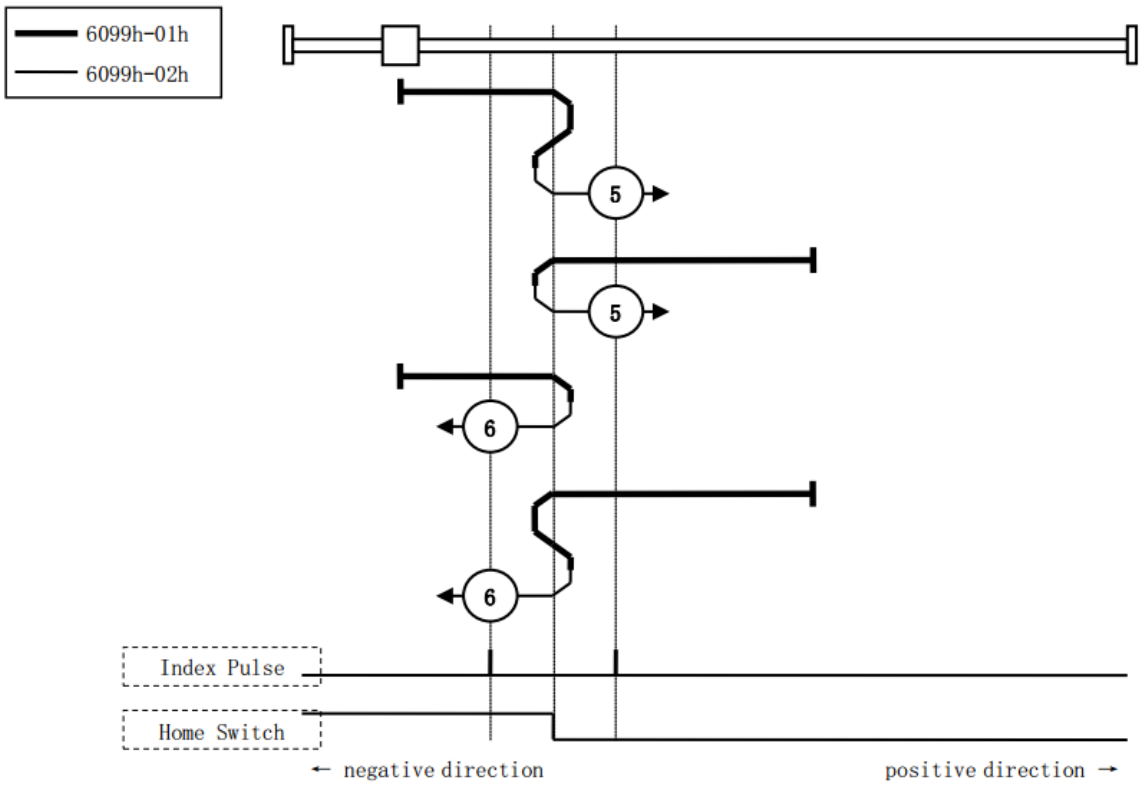


Homing on positive home switch and index pulse

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。

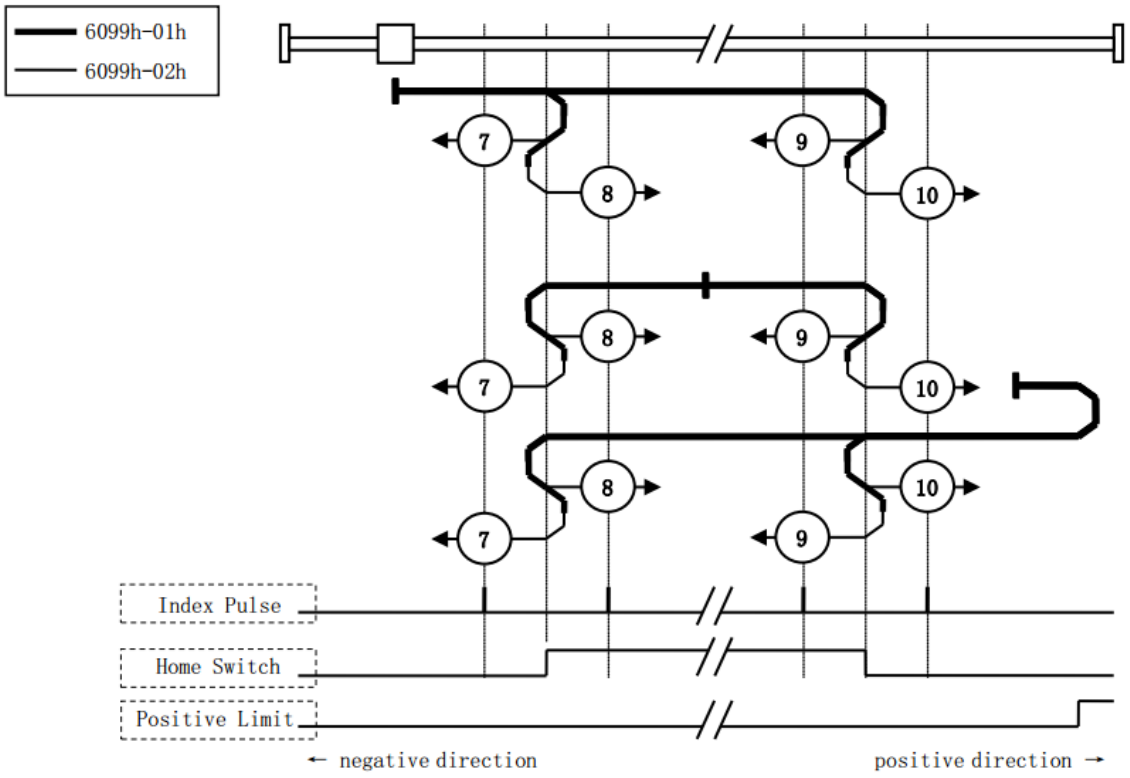


Homing on negative home switch and index pulse

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)

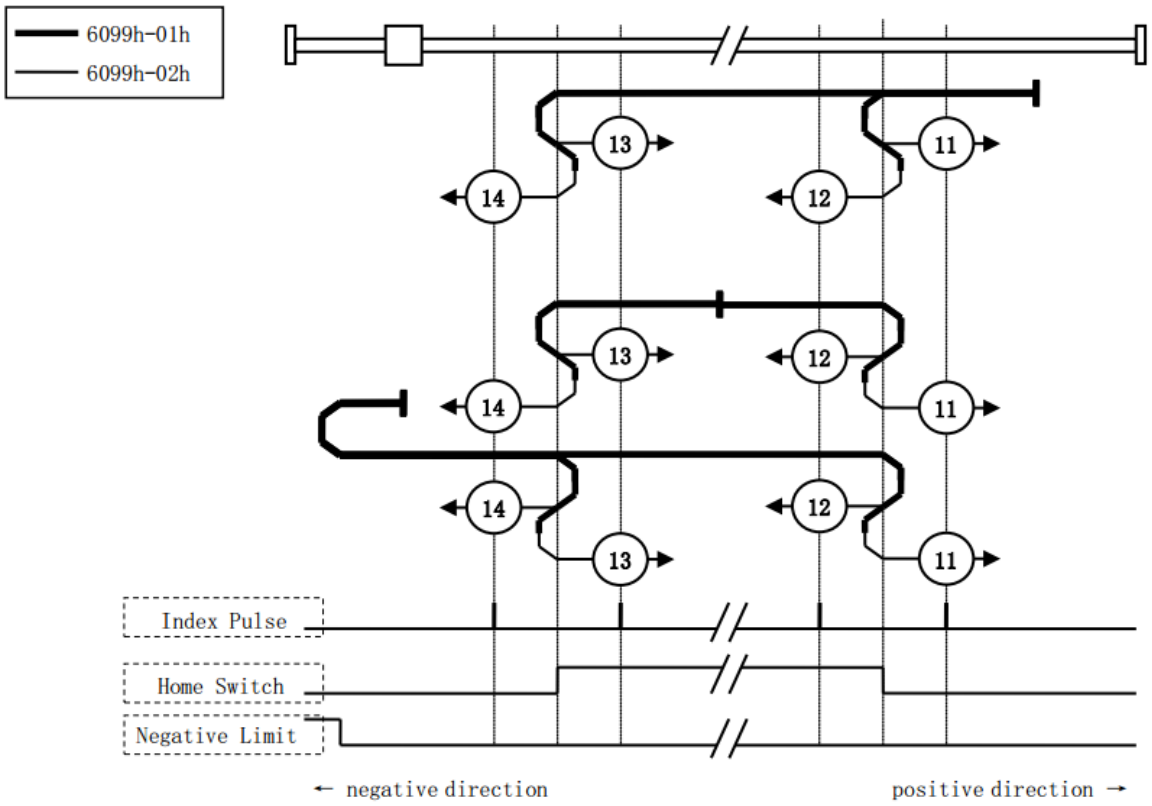


Homing on home switch and index pulse - positive initial motion

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)



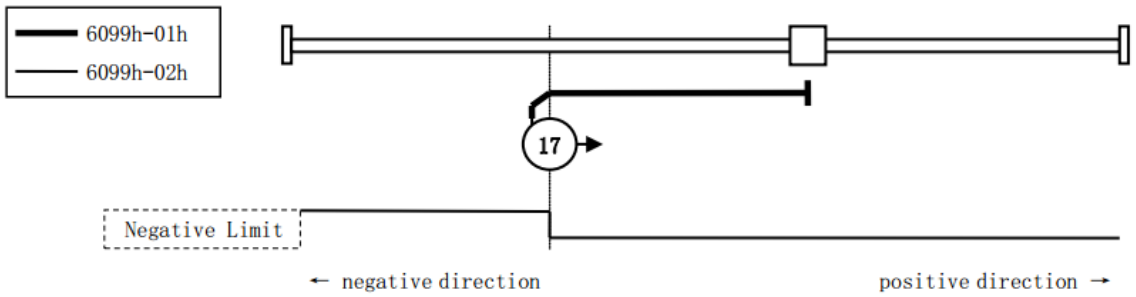
Homing on home switch and index pulse - negative initial motion

回零模式 17

・此方法是，类似于回零模式 1。

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

(请参照下图)



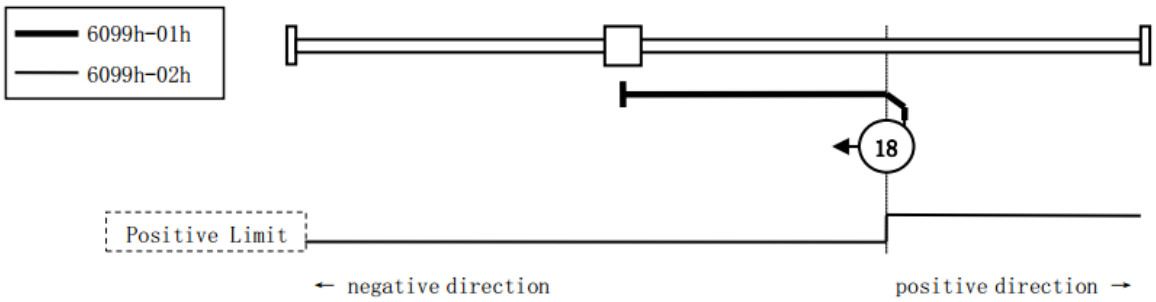
Homing on negative limit switch

回零模式 18

・此方法是，类似于 回零模式 2。

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

(请参照下图)



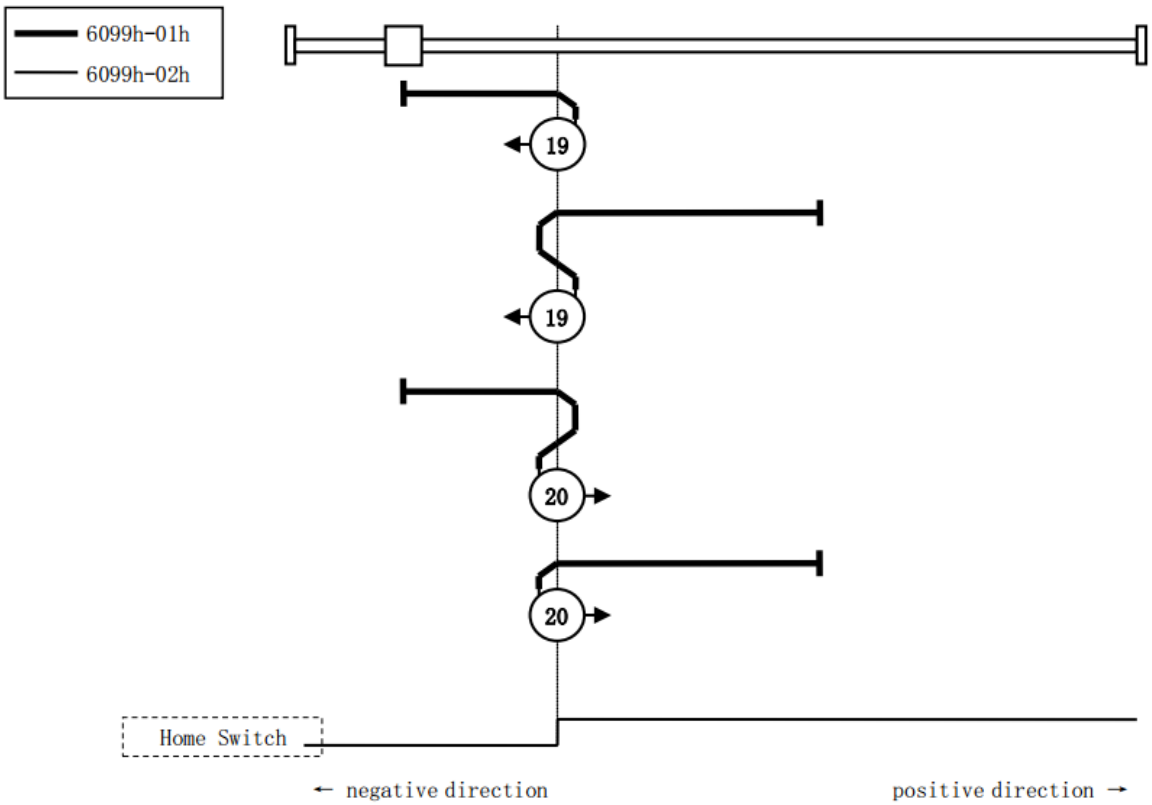
Homing on positive limit switch

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)



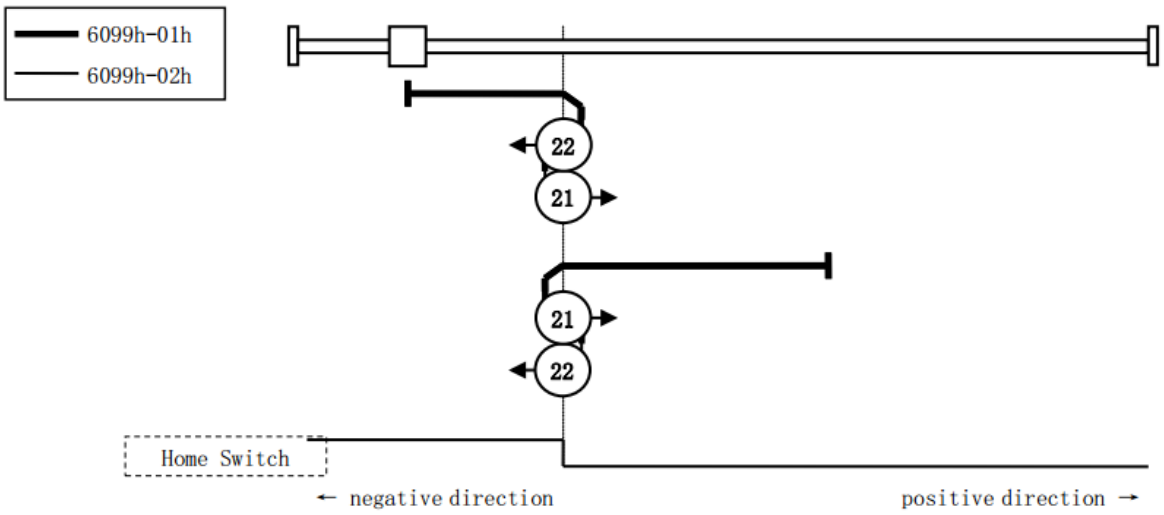
Homing on positive home switch

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)



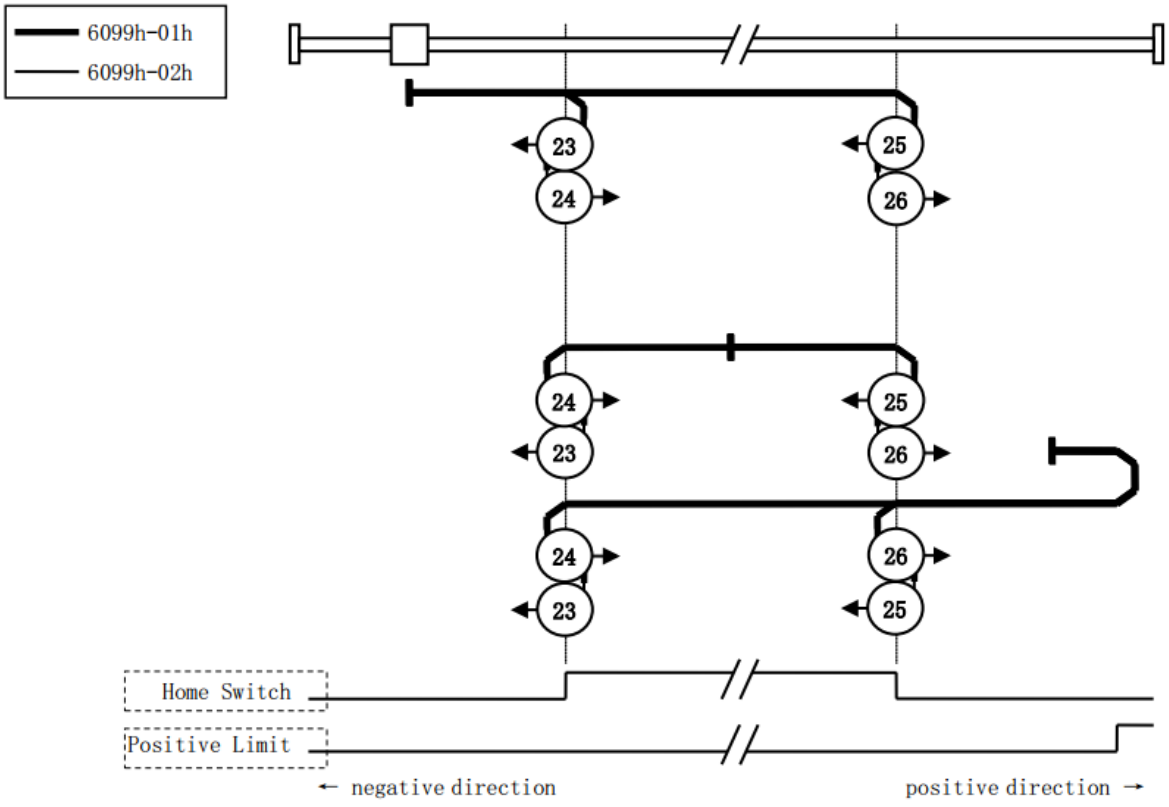
Homing on negative home switch and index pulse

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)



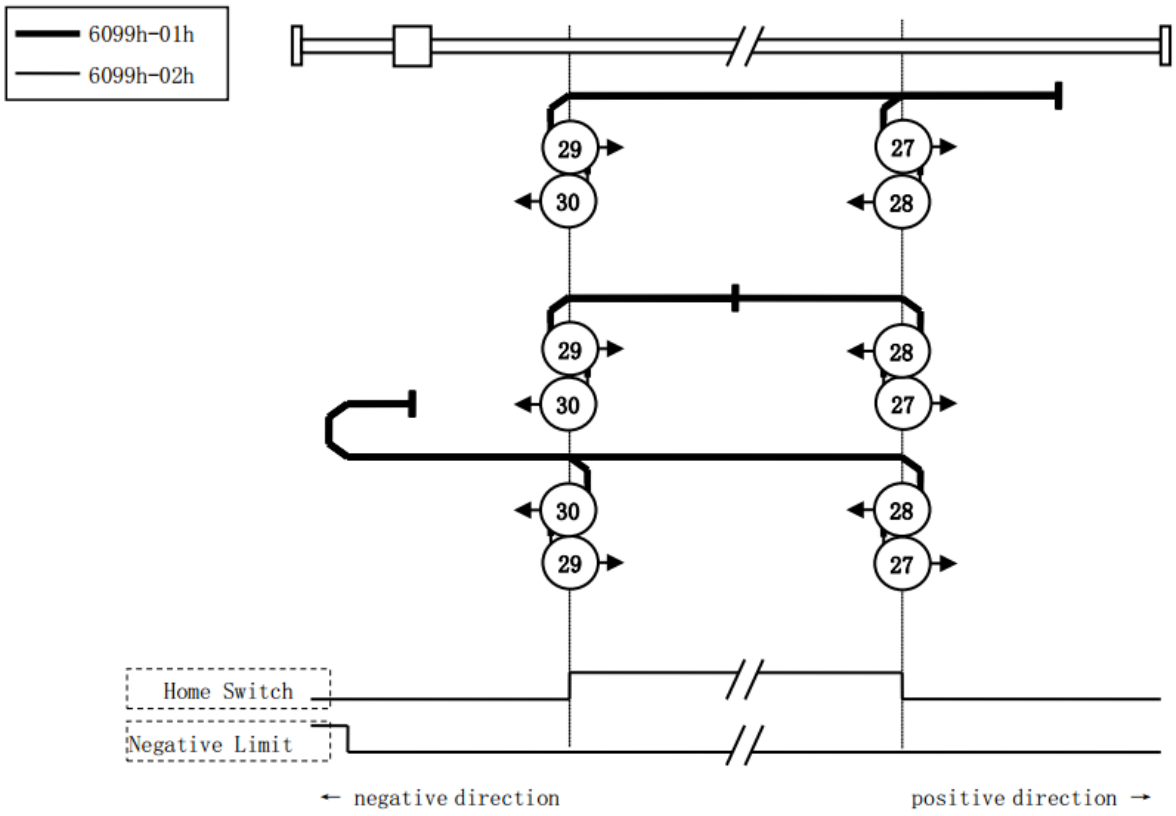
Homing on home switch and index pulse - positive initial motion

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)

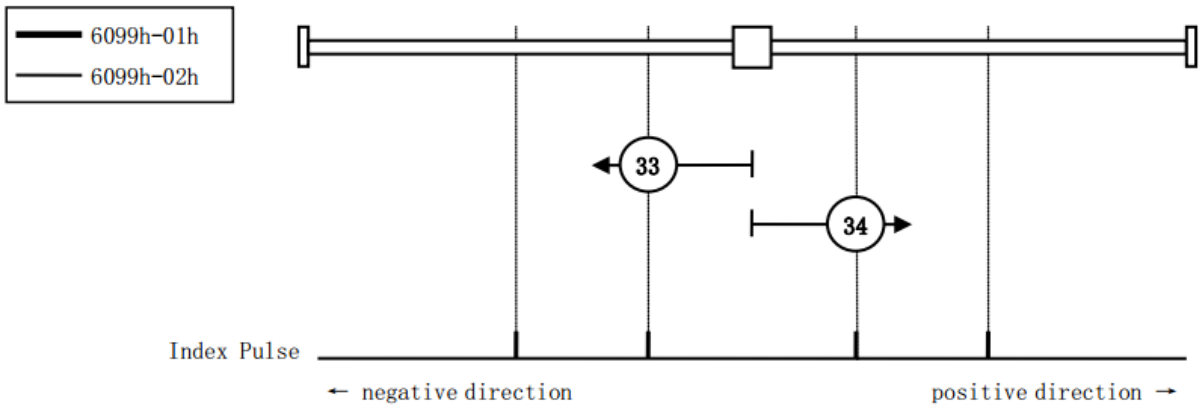


Homing on home switch and index pulse - negative initial motion

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。



Homing on index pulse

8.5 总线高级功能函数

short nmc_set_fieldbus_error_switch(WORD CardNo, WORD channel,WORD data);

功　能：当总线报掉线报错以后，设置网络拓扑轴是否可以操作

参　数：CardNo　　控制卡卡号

channel　EtherCAT 通道号，固定 2；

data　　1 表示使能，0 表示不使能；

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；默认为不使能；

返回值：错误代码

short nmc_get_fieldbus_error_switch(WORD CardNo, WORD channel,WORD* data);

功　能：当总线报掉线报错以后，读取网络拓扑轴是否可以操作配置

参　数：CardNo　　控制卡卡号

channel　EtherCAT 通道号，固定 2；

data　　1 表示使能，0 表示不使能；

返回值：错误代码

第 9 章　基本功能函数说明

9.1　板卡设置函数

short dmc_board_init(void)
功　能：控制卡初始化函数，分配系统资源
参　数：无
返回值：0：　　没有找到控制卡，或者控制卡异常
　　1~8：　控制卡数
　　负值：　表明有 2 张或 2 张以上控制卡的硬件设置卡号相同；返回值取绝对值后减 1 即为该卡号

short dmc_cool_reset(WORD CardNo)
功　能：控制卡冷复位函数
参　数：CardNo　　控制卡卡号
返回值：错误代码
注　意：执行复位操作后，必须关闭控制卡，等待 15 秒方可执行初始化控制卡，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卡。

short dmc_soft_reset(WORD CardNo)
功　能：控制卡热复位函数
参　数：CardNo　　控制卡卡号
返回值：错误代码
注　意：执行复位操作后，必须关闭控制卡，等待 15 秒方可执行初始化控制卡，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卡。

short dmc_original_reset(WORD CardNo)
功　能：控制卡初始复位函数
参　数：CardNo　　控制卡卡号
返回值：错误代码
注　意：控制卡初始复位函数会删除控制卡总线配置文件。执行复位操作后，必须关闭控制卡，等待 15 秒方可执行初始化控制卡，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卡。

short dmc_board_close(void)
功　能：控制卡关闭函数，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：获取控制卡硬件 ID 号
参　数：CardNun　　返回初始化成功的卡数
　　CardTypeList　　返回控制卡固件类型数组
　　CardIdList　　返回控制卡硬件 ID 号数组，卡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：获取控制卡发布版本号
参　数：CardNo　　控制卡卡号
　　ReleaseVersion　　返回控制卡发布版本号
返回值：错误代码

short dmc_get_card_version(WORD CardNo, DWORD *CardVersion)
功　能：获取控制卡硬件版本号
参　数：CardNo　　控制卡卡号
　　CardVersion　　返回控制卡硬件版本号
返回值：错误代码

short dmc_get_card_soft_version(WORD CardNo, DWORD *FirmID, DWORD *SubFirmID)
功　能：获取控制卡固件版本号
参　数：CardNo　　控制卡卡号
　　FirmID　　返回控制卡固件类型
　　SubFirmID　　返回控制卡固件版本号
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：获取控制卡动态库文件版本号

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：获取控制卡本地 IO 输入输出口数

参　数：CardNo　　控制卡卡号

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　　控制卡卡号

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）可以在控制卡 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　　控制卡卡号

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）可以在控制卡 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　　控制卡卡号
　　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

9.2　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　　卡号
　　axis　　指定轴号
　　equiv　　脉冲当量，单位：pulse/unit
返回值：错误代码
注　意：1）轴脉冲当量建议在总线轴使能前设置；
　　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　　卡号
　　axis　　指定轴号
　　equiv　　返回脉冲当量
返回值：错误代码

short dmc_set_softlimit(WORD CardNo, WORD axis, WORD enable, WORD source_sel, WORD SL_action, long N_limit, long P_limit)
功　能：设置软限位
参　数：CardNo　　控制卡卡号
　　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_action　　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：pulse

P_limit　　正限位位置，单位：pulse
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(WORD CardNo, WORD axis, WORD* enable, WORD* source_sel, WORD* SL_action, long* N_limit, long* P_limit)
功能：读取软限位设置
参　数：CardNo　　控制卡卡号
axis　　指定轴号
enable　　返回使能状态
source_sel　　返回计数器选择
SL_action　　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

9.3　单轴运动速度曲线设置函数

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　　卡号
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　　卡号

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　　控制卡卡号
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　　控制卡卡号
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

9.4　单轴运动函数

short dmc_pmove_unit(WORD CardNo, WORD axis, double Dist, WORD posi_mode)
功　能：定长运动
参　数：CardNo　　卡号
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　　控制卡卡号
　　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　　卡号
　　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　　卡号
　　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　　卡号
　　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。此时控制卡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

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卡新计算的值所覆盖，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

9.5 插补速度设置函数

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 卡号

Crd 坐标系号，取值范围：0~5

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）DMC-E3032 卡支持 6 个坐标系（参数 Crd）。六个坐标系的速度可独立设置，执行插补时六个坐标系可独立进行插补运动（即可同时进行六组插补运动）

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 卡号

Crd 坐标系号，取值范围：0~5

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 卡号
Crd 坐标系号，取值范围：0~5
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 卡号
Crd 坐标系号，取值范围：0~5
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

9.6 插补运动函数

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 卡号
Crd 坐标系号，取值范围：0~5
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　　卡号

Crd　　坐标系号，取值范围：0~5

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　　卡号

Crd　　坐标系号，取值范围：0~5

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，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　　卡号
Crd　　坐标系号，取值范围：0~5
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

9.7　位置计数器控制函数

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　　卡号

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　　卡号

axis　　指定轴号

pos　　返回当前位置值，单位：unit

返回值：错误代码

9.8　运动状态检测及控制相关函数

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　　卡号

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_check_done(WORD CardNo, WORD axis)
功 能：检测指定轴的运动状态
参 数：CardNo 控制卡卡号
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 控制卡卡号
Crd 指定控制卡上的坐标系号（取值范围：0~5）
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

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 控制卡卡号
axis 指定轴号，
返回值：见表 9.2

表 9.2 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 控制卡卡号
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　　控制卡卡号

　　Crd　　指定控制卡上的坐标系号（取值范围：0~5）

　　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　　控制卡卡号

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_position_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　　控制卡卡号

　　axis　　指定轴号

　　pos　　目标位置，单位：unit

返回值：错误码

9.9　状态检测

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　　卡号

　　axis　　指定轴号

　　run_mode　　返回运动模式：

　　0：空闲

　　1：Pmove

　　2：Vmove

3：Hmove
4：Handwheel
5：Ptt / Pts
6：Pvt / Pvts
7：保留
8：保留
9：保留
10：Continue

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_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　　卡号
Crd　　坐标系号，取值范围：0~5

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　　卡号
axis　　指定轴号
StopReason　停止原因：
0：正常停止
1：保留
2：保留
3：LTC 外部触发立即停止，IMD_STOP_AT_LTC

4：EMG 立即停止，IMD_STOP_AT_EMG
5：正硬限位立即停止，IMD_STOP_AT_ELP
6：负硬限位立即停止，IMD_STOP_AT_ELN
7：正硬限位减速停止，DEC_STOP_AT_ELP
8：负硬限位减速停止，DEC_STOP_AT_ELN
9：正软限位立即停止，IMD_STOP_AT_SOFT_ELP
10：负软限位立即停止，IMD_STOP_AT_SOFT_ELN
11：正软限位减速停止，DEC_STOP_AT_SOFT_ELP
12：负软限位减速停止，DEC_STOP_AT_SOFT_ELN
13：命令立即停止，IMD_STOP_AT_CMD
14：命令减速停止，DEC_STOP_AT_CMD
15：其它原因立即停止，IMD_STOP_AT_OTHER
16：其它原因减速停止，DEC_STOP_AT_OTHER
17：未知原因立即停止，IMD_STOP_AT_UNKOWN
18：未知原因减速停止，DEC_STOP_AT_UNKOWN
19：保留，DEC_STOP_AT_DEC

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　　卡号
　　axis　　指定轴号
返回值：错误代码

9.10　输入输出 IO 函数

short dmc_read_inbit(WORD CardNo, WORD bitno)
功　能：读取指定控制卡的某个输入端口的电平
参　数：CardNo　　控制卡卡号
　　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
返回值：指定的输入端口电平：0：低电平，1：高电平

short dmc_write_outbit(WORD CardNo, WORD bitno, WORD on_off)
功　能：设置指定控制卡的某个输出端口的电平

参　数：CardNo　　控制卡卡号
　　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
　　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：读取指定控制卡的某个输出端口的电平
参　数：CardNo　　控制卡卡号
　　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
返回值：指定输出端口的电平，0：低电平，1：高电平

DWORD dmc_read_inport(WORD CardNo, WORD portno)
功　能：读取指定控制卡的全部输入端口的电平
参　数：CardNo　　控制卡卡号
　　portno　　保留参数，固定值为 0；所有输入口按顺序排列，每 32bit 为一个 port 口号，例如 portno=0 表示 in0-in31，portno=1 表示 in32-in63，以此类推；
返回值：　bit0~bit31 的定义见表 9.3

表 9.3　DMC-E3032 卡 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
8-31	8-31	扩展输入口

DWORD dmc_read_outport(WORD CardNo, WORD portno)
功　能：读取指定控制卡的全部输出口的电平
参　数：CardNo　　控制卡卡号
　　portno　　保留参数，固定值为 0；所有输出口按顺序排列，每 32bit 为一个 port

口号，例如 portno=0 表示 out0-out31，portno=1 表示 out32-out63，以此类推；

返回值： bit0~bit31 的定义见表 9.4

short dmc_write_outport(WORD CardNo, WORD portno, DWORD port_value)

功　能：设置指定控制卡的全部输出口的电平

参　数：CardNo　　控制卡卡号

portno　　保留参数，固定值为 0；所有输出口按顺序排列，每 32bit 为一个 port 口号，例如 portno=0 表示 out0-out31，portno=1 表示 out32-out63，以此类推；

port_value　bit0~bit31 的定义见表 9.4

返回值：错误代码

表 9.4　DMC-E3032 卡 dmc_read_outport、dmc_write_outport 函数返回值各位的定义表

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
8-31	8-31	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　　控制卡卡号

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　　控制卡卡号

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　　控制卡卡号
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　　控制卡卡号
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　　控制卡卡号
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

9.11　轴 IO 映射函数

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　　控制卡卡号
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

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　　控制卡卡号
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

9.12　手轮功能函数

short dmc_set_handwheel_inmode (WORD CardNo, WORD axis, WORD inmode, long multi, double vh)

功　能：设置单轴手轮运动控制输入方式

参　数：CardNo　　控制卡卡号
axis　　指定轴号
inmode　　手轮输入方式，0：脉冲+方向信号；1：A、B 相位正交信号
multi　　手轮倍率，正数表示默认方向，负数表示与默认方向反向
vh　　保留参数，固定值为 0

返回值：错误代码

short dmc_get_handwheel_inmode (WORD CardNo, WORD axis, WORD* inmode, long * multi, double* vh)

功　能：读取单轴手轮运动控制输入方式

参　数：CardNo　　控制卡卡号

axis　　指定轴号

inmode　　返回手轮输入方式

multi　　返回手轮倍率

vh　　保留参数

返回值：错误代码

short dmc_handwheel_move(WORD CardNo, WORD axis)

功　能：启动手轮运动

参　数：CardNo　　控制卡卡号

axis　　指定轴号

返回值：错误代码

注　意：当启动手轮运动后，只有发送 dmc_stop 或 dmc_emg_stop 命令后才会退出手轮模式

short dmc_set_handwheel_inmode_extern(WORD CardNo, WORD inmode, WORD AxisNum, WORD* AxisList, long* multi)

功能：设置多轴手轮运动控制输入方式

参　数：CardNo　　控制卡卡号

inmode　　手轮输入方式：0：脉冲+方向信号；1：A、B 相位正交信号

AxisNum　　参与手轮运动的轴数

AxisList　　参与手轮运动的轴号数组

multi　　手轮倍率数组:正数表示默认方向，负数表示与默认方向反向

返回值：错误代码

注　意：通过该函数设置可以使一个手轮通道控制多个轴同时运动，运动倍率都以第一个轴的倍率。

short dmc_get_handwheel_inmode_extern(WORD CardNo, WORD *inmode, WORD *AxisNum, WORD* AxisList, long* multi)

功能：读取多轴手轮运动控制输入方式;

参　数：CardNo　　控制卡卡号

inmode　　返回手轮输入方式

AxisNum 返回参与手轮运动的轴数
AxisList 返回参与手轮运动的轴号数组
multi 返回手轮倍率数组
返回值：错误代码

9.13 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 卡号
axis 指定轴号
pos 编码器计数值，单位：unit
返回值：错误代码

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 卡号
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

short dmc_set_extra_encoder_mode(WORD CardNo, WORD channel, WORD inmode, WORD multi);
功 能：设置辅助编码器输入方式
参 数：CardNo 控制卡卡号
channel 辅助编码器通道，0，通道 0，1，通道 1
inmode 辅助编码器输入方式，0：脉冲+方向信号；1：A、B 相位正交信号
multi 辅助编码器计数模式，固定 1：4 倍频计数
返回值：错误代码

short dmc_get_extra_encoder_mode(WORD CardNo, WORD channel, WORD* inmode, WORD* multi);
功 能：读取辅助编码器输入方式
参 数：CardNo 控制卡卡号

channel 辅助编码器通道
inmode 返回辅助编码器输入方式
multi 返回辅助编码器计数模式
返回值：错误代码

short dmc_set_extra_counter_reverse(WORD CardNo,WORD channel,WORD reverse);
功 能：设置辅助编码器 AB 相计数反相
参 数：CardNo 控制卡卡号
axis 辅助编码器通道，通道 0/1
reverse 0-A 超前 B 计数，1-B 超前 A 计数
返回值：错误代码

short dmc_get_extra_counter_reverse(WORD CardNo,WORD channel,WORD *reverse);
功 能：读取辅助编码器 AB 相计数反相
参 数：CardNo 控制卡卡号
axis 辅助编码器通道，通道 0/1
reverse 0-A 超前 B 计数，1-B 超前 A 计数
返回值：错误代码

short dmc_set_extra_encoder(WORD CardNo, WORD channel, int pos);
功 能：设置辅助编码器计数值
参 数：CardNo 卡号
channel 辅助编码器通道
pos 辅助编码器计数值，单位：pulse
返回值：错误代码

short dmc_get_extra_encoder(WORD CardNo, WORD channel, int* pos);
功 能：读取辅助编码器计数值
参 数：CardNo 卡号
channel 辅助编码器通道
pos 辅助编码器计数值，单位：pulse
返回值：错误代码

9.14 高速位置锁存函数

short dmc_ltc_set_mode(WORD CardNo, WORD latch, WORD ltc_mode, WORD ltc_logic, double filter)
功　能：配置锁存器
参　数：CardNo　　控制卡卡号
　　latch　　锁存器号，0-3
　　ltc_mode　　锁存模式，0：单次锁存，1：连续锁存
　　ltc_logic　　锁存信号的触发方式，0：下降沿锁存，1：上升沿锁存，2：双边沿锁存
　　filter　　滤波时间，单位：us
返回值：错误代码

short dmc_ltc_get_mode(WORD CardNo, WORD latch, WORD *ltc_mode, WORD *ltc_logic, double *filter)
功　能：读取锁存器配置
参　数：CardNo　　控制卡卡号
　　latch　　锁存器号,0-3
　　ltc_mode　　读取锁存模式，0：单次锁存，1：连续锁存
　　ltc_logic　　读取锁存信号的触发方式，0：下降沿锁存，1：上升沿锁存，2：双边沿锁存
　　filter　　读取滤波时间，单位：us
返回值：错误代码

short dmc_ltc_set_source(WORD CardNo, WORD latch, WORD axis, WORD latch_source)
功　能：配置锁存源
参　数：CardNo　　控制卡卡号
　　latch　　锁存器号,0-3
　　axis　　锁存器辅助编码器通道号，0、1
　　latch_source　　锁存源，0：指令位置，1：辅助编码器计数
返回值：错误代码

short dmc_ltc_get_source(WORD CardNo, WORD latch, WORD axis, WORD *latch_source)
功　能：读取锁存源配置
参　数：CardNo　　控制卡卡号

latch　　锁存器号,0-3
axis　　锁存器辅助编码器通道号
latch_source　　读取锁存源，0：指令位置，1：辅助编码器计数

返回值：错误代码

long dmc_ltc_get_value_unit(WORD CardNo, WORD latch,WORD axis,double *value)

功　能：读取锁存值

参　数：CardNo　　控制卡卡号
latch　　锁存器号,0-3
axis　　锁存器辅助编码器通道号
value　　锁存值

返回值：锁存值

注　意：1）该函数只可以读辅助编码器锁存值；
2）当选择锁存方式为连续锁存时，该函数第一次执行的时候，读取的是锁存器的第一个锁存值，第二次执行的时候，读取的是锁存器的第二个锁存值，以此类推；
3）单次锁存时，调用 dmc_get_latch_value 不会自动清除已锁存个数，须调用 dmc_reset_latch_flag;

short dmc_ltc_get_number(WORD CardNo, WORD latch,WORD axis,int *number)

功　能：读取锁存器锁存个数

参　数：CardNo　　控制卡卡号
latch　　锁存器号,0-3
axis　　锁存器辅助编码器通道号
number　　读取锁存器个数

返回值：已锁存个数，0 表示无锁存值

注　意：连续锁存时，函数 dmc_get_latch_value 每执行一次，该函数的返回值减 1

short dmc_ltc_reset(WORD CardNo, WORD latch)

功　能：复位锁存器

参　数：CardNo　　控制卡卡号
latch　　锁存器号,0-3

返回值：错误代码

注　意：当使用锁存功能前，必须先调用此函数复位锁存器的标志位

9.15 软件锁存函数

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 控制卡卡号
latch 锁存器号
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_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 控制卡卡号
latch 锁存器号
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 控制卡卡号
latch 锁存器号
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　　控制卡卡号
　　latch　　锁存器号
　　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　　控制卡卡号
　　latch　　锁存器号
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　　控制卡卡号
　　latch　　锁存器号
　　axis　　锁存对应轴号
　　number　　锁存个数
返回值：错误代码

short dmc_softltc_get_value_unit(WORD CardNo,WORD latch,WORD axis,double *value)
功　能：读取锁存值
参　数：CardNo　　控制卡卡号
　　latch　　锁存器号
　　axis　　锁存对应轴号
　　value　　读取锁存值
返回值：错误代码

9.16　位置比较函数

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　　控制卡卡号
　　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　　控制卡卡号
　　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　控制卡卡号
　　axis　　指定轴号
返回值：错误代码

short dmc_compare_add_point_cycle(WORD CardNo,WORD axis,long pos,WORD dir, DWORD bitno,DWORD cycle,WORD level)
功　能：添加一维位置比较点
参　数：CardNo　控制卡卡号
　　axis　　指定轴号
　　pos　　比较位置
　　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(WORD CardNo, WORD axis, long* pos)

功　能：读取当前一维比较点位置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号
　　pos　　返回当前比较点位置
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　　控制卡卡号
　　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　　控制卡卡号
　　axis　　指定轴号
　　PointNum　　返回可以加入的比较点数
返回值：错误代码

9.17　高速位置比较函数

short dmc_hcmp_set_mode(WORD CardNo, WORD hcmp, WORD cmp_mode)
功　能：设置高速比较模式
参　数：CardNo　　控制卡卡号
　　hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
　　cmp_mode　比较模式：
　　0：禁止（默认值）
　　1：等于
　　2：小于
　　3：大于
　　4：队列，提供 127 个点比较空间，采用先添加先比较，比较完可追加比较点，也可一次性添加多个比较点

5：线性，提供起始比较点，位置增量，比较次数

返回值：错误代码

注　意：1）当选择模式 1 时，只有当前位置等于比较位置时，CMP 端口才输出有效电平

2）当选择模式 2 时，只要当前位置小于比较位置时，CMP 端口就一直保持有效电平

3）当选择模式 3 时，只要当前位置大于比较位置时，CMP 端口就一直保持有效电平

4）当选择模式 4 或 5 时，CMP 端口输出有效电平的时间通过 dmc_hcmp_set_config 函数的 time 参数（脉冲宽度）设置

short dmc_hcmp_get_mode(WORD CardNo, WORD hcmp, WORD* cmp_mode)

功　能：读取高速比较模式设置

参　数：CardNo　　控制卡卡号

hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）

cmp_mode　返回比较模式设置

返回值：错误代码

short dmc_hcmp_set_config(WORD CardNo, WORD hcmp, WORD axis, WORD cmp_source, WORD cmp_logic, long time)

功　能：配置高速比较器

参　数：CardNo　　控制卡卡号

hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）

axis　　辅助编码器通道号

cmp_source　比较位置源，固定值 1：辅助编码器计数器

cmp_logic　有效电平：0：低电平，1：高电平

time　　脉冲宽度，单位：us，取值范围：1us~20s

返回值：错误代码

注　意：1）该函数的 time 参数（脉冲宽度）只对队列和线性比较模式起作用

short dmc_hcmp_get_config(WORD CardNo, WORD hcmp, WORD* axis, WORD* cmp_source, WORD* cmp_logic, long* time)

功　能：读取高速比较器配置

参　数：CardNo　　控制卡卡号

hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）

axis　　返回辅助编码器通道号

cmp_source　返回比较位置源设置

cmp_logic　　返回有效电平设置
time　　返回脉冲宽度设置
返回值：错误代码

short dmc_hcmp_clear_points(WORD CardNo, WORD hcmp)
功　能：清除已添加的所有高速位置比较点
参　数：CardNo　　控制卡卡号
hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
返回值：错误代码

short dmc_hcmp_add_point(WORD CardNo, WORD hcmp, long cmp_pos)
功　能：添加/更新高速比较位置
参　数：CardNo　　控制卡卡号
hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
cmp_pos　　队列模式下：添加比较位置，单位：pulse
线性模式下：更新起始比较位置，单位：pulse
其他模式下：更新比较位置，单位：pulse
返回值：错误代码

short dmc_hcmp_set_liner(WORD CardNo, WORD hcmp, long Increment, long Count)
功　能：设置高速比较线性模式参数
参　数：CardNo　　控制卡卡号
hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
Increment　　位置增量值，单位：pulse（正值表示位置递增，负值表示位置递减）
Count　　比较次数，取值范围：1~65535
返回值：错误代码

short dmc_hcmp_get_liner(WORD CardNo, WORD hcmp, long* Increment, long* Count)
功　能：读取高速比较线性模式参数设置
参　数：CardNo　　控制卡卡号
hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
Increment　　返回位置增量值设置
Count　　返回比较次数设置
返回值：错误代码

short dmc_hcmp_get_current_state(WORD CardNo, WORD hcmp, long *remained_points, long *current_point, long *runned_points)
功　能：读取高速比较参数
参　数：CardNo　　控制卡卡号
　　hcmp　　比较器号，取值范围：0~5（对应硬件 OUT2~OUT7 端口）
　　remained_points　　返回可添加比较点数
　　current_point　　返回当前比较点位置，单位：pulse
　　runned_points　　返回已比较点数
返回值：错误代码

9.18　二维高速位置比较函数

short dmc_hcmp_2d_set_enable(WORD CardNo,WORD hcmp, WORD cmp_enable)
功　能：设置高速比较使能
参　数：CardNo　　卡号
　　hcmp　　高速比较器号
　　cmp_enable　　二维高速比较器使能，0：禁止，1：使能
返回值：错误代码

short dmc_hcmp_2d_get_enable(WORD CardNo,WORD hcmp, WORD *cmp_enable)
功　能：读取高速比较使能
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器号
　　cmp_enable　　返回二维高速比较器使能状态
返回值：错误代码

short dmc_hcmp_2d_set_config_unit(WORD CardNo,WORD hcmp,WORD cmp_mode,WORD x_axis, WORD x_cmp_source, double x_cmp_error, WORD y_axis, WORD y_cmp_source, double y_cmp_error,WORD cmp_logic,int time);
功　能：配置高速比较器
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器号
　　cmp_mode　　比较模式：

0：进入误差带后触发
1：进入误差带单轴等于后再触发

x_axis	x 轴关联轴号
x_cmp_source	x 轴比较位置源：0：指令位置，1：反馈位置
x_cmp_error	x 轴误差带设置，单位：unit
y_axis	y 轴关联轴号
y_cmp_source	y 轴比较位置源：0：指令位置，1：反馈位置
y_cmp_error	y 轴误差带设置，单位：unit
cmp_logic	有效电平：0：低电平，1：高电平
time	脉冲宽度，单位：us，取值范围：1us-20s

返回值：错误代码

注　意：1）如果 2 次高速比较输入相距很近，则自动重合为一个位置单位。

short dmc_hcmp_2d_get_config_unit(WORD CardNo,WORD hcmp,WORD * cmp_mode,WORD * x_axis, WORD * x_cmp_source, double * x_cmp_error, WORD * y_axis, WORD * y_cmp_source, double * y_cmp_error,WORD * cmp_logic,int * time);

功　能: 读取高速比较器

参　数：CardNo　　控制卡卡号
hcmp　　高速比较器号
cmp_mode　　比较模式：
0：进入误差带后触发
1：进入误差带单轴等于后再触发

x_axis	返回 x 轴关联轴号
x_cmp_source	返回 x 轴比较位置源：0：指令位置，1：反馈位置
x_cmp_error	x 轴误差带设置，单位：unit
y_axis	返回 y 轴关联轴号
y_cmp_source	返回 y 轴比较位置源：0：指令位置，1：反馈位置
y_cmp_error	x 轴误差带设置，单位：unit
cmp_logic	返回有效电平：0：低电平，1：高电平
time	返回脉冲宽度，单位：us，取值范围：1us~20s

返回值：错误代码

short dmc_hcmp_2d_set_pwmoutput(WORD CardNo,WORD hcmp,WORD pwm_enable,double duty,double freq,WORD pwm_number);

功　能：配置高速比较器 pwm 参数
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器号
　　pwm_enable　　pwm 模式使能
　　duty　　占空比
　　freq　　频率
　　pwm_number　　输出的 pwm 脉冲数
返回值：错误代码

short dmc_hcmp_2d_get_pwmoutput(WORD CardNo,WORD hcmp,WORD * pwm_enable,double * duty,double * freq,WORD * pwm_number);
功　能：配置高速比较器 pwm 参数
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器号
　　pwm_enable　　读取 pwm 模式使能状态
　　duty　　读取占空比
　　freq　　读取频率
　　pwm_number　　读取输出的 pwm 脉冲数
返回值：错误代码

short dmc_hcmp_2d_clear_points(WORD CardNo,WORD 2dhcmp)
功　能：清除所有缓冲区高速位置缓冲比较值，并退出当前比较状态
参　数：CardNo　　控制卡卡号
　　2dhcmp　　保留，参数值为 0
返回值：错误代码

short dmc_hcmp_2d_add_point_unit(WORD ConnectNo,WORD hcmp, double x_cmp_pos, double y_cmp_pos,WORD cmp_outbit)
功　能：添加/更新高速比较位置
参　数：CardNo　　控制卡卡号
　　2dhcmp　　高速比较器号
　　x_cmp_pos　　队列模式下：添加 x 比较位置，单位：unit
　　y_cmp_pos　　队列模式下：添加 x 比较位置，单位：unit
　　cmp_outbit　　输出口号

返回值：错误代码

short dmc_hcmp_2d_get_current_state_unit(WORD CardNo,WORD hcmp,int *remained_points,double *x_current_point,double *y_current_point,int *runned_points,WORD *current_state,WORD *current_outbit);

功　能：读取高速比较参数

参　数：CardNo　　控制卡卡号

hcmp　　高速比较器号

remained_points　　返回可添加比较点数

x_current_point　　返回当前 x 比较点位置

y_current_point　　返回当前 y 比较点位置

runned_points　　返回已比较点数

current_state　　比较器状态 1 正在输出 0 输出完成

current_outbit　　返回当前输出口

返回值：错误代码

short dmc_hcmp_2d_force_output(WORD CardNo,WORD 2dhcmp,WORD enable);

功　能：该函数用于强制二维比较输出，输出按照配置好的脉冲模式或者 pwm 模式

参　数：CardNo　　控制卡卡号

2dhcmp　　保留，参数值为 0

enable　　强制二维比较输出，设置为 1 使能

返回值：错误代码

9.19　异常信号接口函数

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　　控制卡卡号

axis　　指定轴号

stop_time　　减速时间，单位：s

返回值：错误代码

注　意：

1. 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；

2. T 型速度规划，电机实际停止时间等于设置的异常减速停止时间；S 型速度规划，电机实际停止时间等于设置的异常减速停止时间与 S 段时间之和；

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　　控制卡卡号
　　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

9.20　检测轴到位状态函数

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　　控制卡卡号
　　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　　控制卡卡号
　　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　　控制卡卡号
　　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　　控制卡卡号
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

9.21　密码管理函数

short dmc_write_sn(WORD CardNo, const char* new_sn)
功　能：修改密码
参　数：CardNo　　控制卡卡号
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　　控制卡卡号
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 3 次后，无法进行校验；若需再次校验，需将电脑关机，断电重启

9.22 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

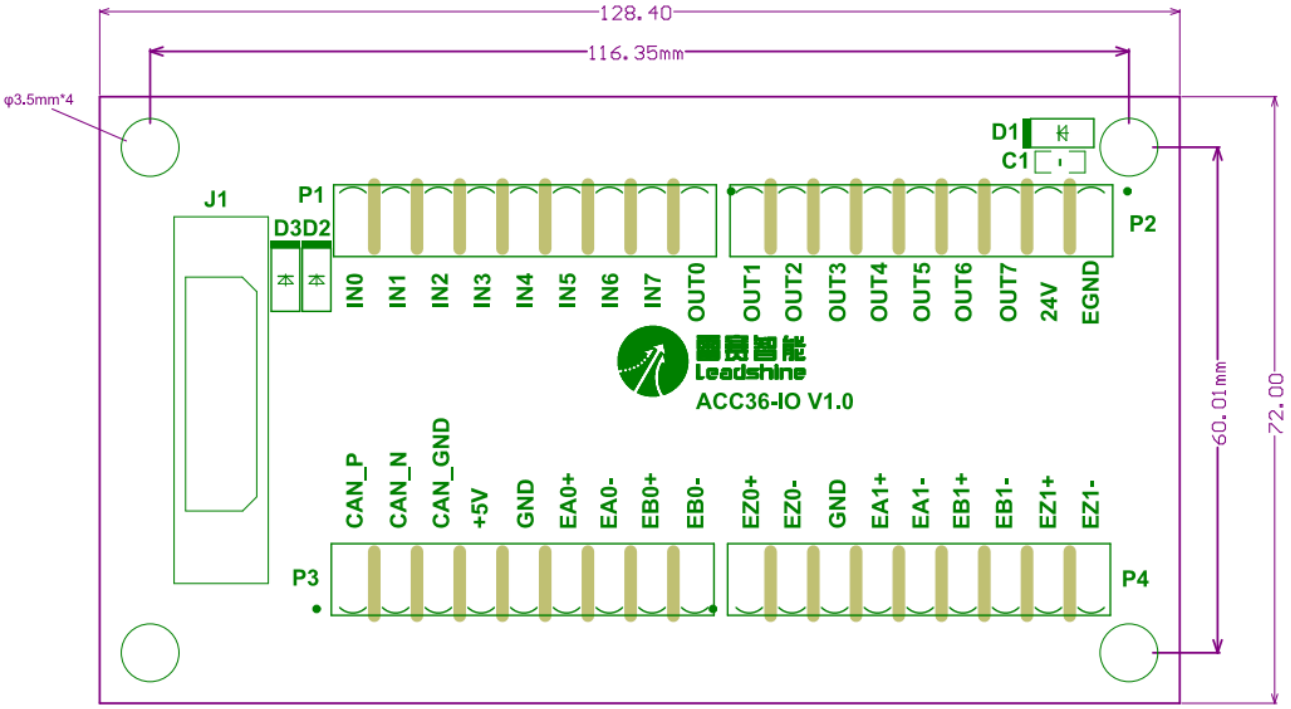
附录 1　ACC36-IO 接线板接口说明及相关尺寸

接线板具体引脚定义如表 2.2 所示。

表 2.2　接线板接口定义

名称	说　明	名称	说　明
1	24V	19	EGND
2	EGND	20	IN0
3	OUT0	21	IN1
4	OUT1	22	IN2
5	OUT2（高速）	23	IN3
6	OUT3（高速）	24	IN4（高速）
7	OUT4（高速）	25	IN5（高速）
8	OUT5（高速）	26	IN6（高速）
9	OUT6（高速）	27	IN7（高速）
10	OUT7（高速）	28	EA1+
11	EA0+	29	EA1-
12	EA0-	30	EB1+
13	EB0+	31	EB1-
14	EB0-	32	EZ1+
15	EZ0+	33	EZ1-
16	EZ0-	34	5V
17	DGND	35	CAN_L
18	CAN_GND	36	CAN_H

ACC36-IO 接线板实物图及尺寸图如下（单位：mm）：


128.40
116.35mm
φ3.5mm*4
D1
C1
J1
P1
D3D2
P2
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
24V
EGND
Leadshine
ACC36-IO V1.0
CAN_P
CAN_N
CAN_GND
+5V
GND
EA0+
EA0-
EB0+
EB0-
EZ0+
EZ0-
GND
EA1+
EA1-
EB1+
EB1-
EZ1+
EZ1-
P3
P4
60.01mm
72.00

附录 2 常见错误码说明

DMC-E3032 运动控制卡的主要总线错误码和函数返回值错误码见表 2.3 和 2.4。

表 2.3 总线错误码

错误代码	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	以太网链路电缆断开
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
0x00FF	总线配置文件错误，有未识别的设备，需要删除配置文件，重新扫描
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）；尤其是当控制卡中存有配置文件，而没有连接任何从站时常出现。

表 2.4 函数返回值错误码

错误代码	错误号	可能错误原因
ERR_NOERR	0	成功
ERR_UNKNOWN	1	总线初始化不成功
ERR_PARAERR	2	参数错误
ERR_TIMEOUT	3	通讯超时
ERR_CONTROLLERBUSY	4	控制器忙、控制器相应轴处于运动中
ERR_CONNECT_TOOMANY	5	链接太频繁
ERR_CANNOT_CONNECTETH	8	网络链接失败，请查看连线及通讯参数
ERR_HANDLEERR	9	句柄错误，网络未链接或链接已断开
ERR_SENDERR	10	发送失败，网络未链接或链接已断开
ERR_FIRMWAREERR	12	固件文件错误
ERR_FIRMWAR_MISMATCH	14	固件不匹配
ERR_CARD_NOT_SUPPORT	17	不支持的功能
ERR_FIRMWARE_INVALID_PARA	20	固件参数错误
ERR_FIRMWARE_STATE_ERR	22	固件当前状态不允许操作
ERR_FIRMWARE_CARD_NOT_SUPPORT	24	固件不支持的功能
ERR_PASSWORD_ERR	25	密码错误

ERR_PASSWORD_TIMES_OUT	26	密码错误输入次数受限
ERR_AXIS_SEL_ERR	30	手轮脉冲的轴档位选择超出范围（软件控制模式）
ERR_HAND_AXIS_NUM_ERR	31	手轮脉冲的轴映射数量超出范围
ERR_AXIS_RATIO_ERR	32	手轮脉冲的倍率档位选择超出范围（软件控制模式）
ERR_HANDWH_START	33	已进入手轮脉冲模式，不能切换软硬件控制模式
ERR_AXIS_BUSY_STATE	34	轴已在运动，不能切换到手轮模式
ERR_LIST_NUM_ERR	50	LIST 号超出范围
ERR_LIST_NOT_OEPN	51	LIST 没有初始化
ERR_PARA_NOT_VALID	52	参数不在有效范围
ERR_LIST_HAS_OPEN	53	LIST 已经打开
ERR_MAIN_LIST_NOT_OPEN	54	LIST 没有初始化
ERR_AXIS_NUM_ERR	55	轴数不在有效范围
ERR_AXIS_MAP_ARRAY_ERR	56	轴映射表为空
ERR_MAP_AXIS_ERR	57	映射轴错误
ERR_MAP_AXIS_BUSY	58	映射轴忙
ERR_PARA_SET_FORBIT	59	运动中不允许更改参数
ERR_FIFO_FULL	60	缓冲区已满
ERR_RADIUS_ERR	61	半径为 0 或小于两点的距离的一半
ERR_MAINLIST_HAS_START	62	LIST 已经启动
ERR_ACC_TIME_ZERO	63	加减速时间为 0
ERR_MAINLIST_NOT_START	64	主要 LIST 没有启动
ERR_POINT_SAME_ON_RADIUS	67	圆弧插补在半径模式下起点和终点不能重合
MCVP_SMOOTH_TIME_SET_ERROR	70	S 曲线加减速模式，平滑时间为零出错
MCVP_START_VEL_SET_ERROR	71	起跳速度小于零
MCVP_STEADY_VEL_SET_ERROR	72	最大速度小于等于零
MCVP_END_VEL_SET_ERROR	73	终点速度小于零
MCVP_TOTAL_LENGTH_SET_ERROR	74	规划长度小于等于零
MCVP_EVEN_TIME_ERROR	75	最小匀速时间小于零
MCVP_PLAN_MODE_SET_ERROR	76	规划模式非 T 型非 S 型
MCVP_ACC_SET_ERROR	77	加速时间等于零
MCVP_DEC_SET_ERROR	78	减速时间等于零
MCVP_PLAN_ERROR	79	规划长度小于等于零（速度规划函数返回，74 为长度计算时返回）
MCVP_SMOOTH_TIME_SET_ERROR	80	s 时间设置错误(小于等于 0)
MCVP_START_VEL_SET_ERROR	81	起始速度绝对值设置错误(小于 0)

MCVP_STEADY_VEL_SET_ERROR	82	最大速度绝对值设置错误(小于等于 0)
MCVP_END_VEL_SET_ERROR	83	停止速度绝对值设置错误(小于 0)
MCVP_TOTAL_LENGTH_SET_ERROR	84	运动距离为 0，无法运动
VP_EVEN_TIME_ERROR	85	最小匀速时间小于零
VP_PLAN_MODE_SET_ERROR	86	规划模式非 T 型非 S 型
VP_ACC_SET_ERROR	87	加速时间等于零
VP_DEC_SET_ERROR	88	减速时间等于零
VP_PLAN_ERROR	95	规划长度小于等于零（速度规划函数返回，84 为长度计算时返回）
ERR_AXIS_INDEX	101	所选轴超出最大值
ERR_SET_WHILE_MOVING	102	轴正在运动，不能设置参数
ERR_ENTER_WHILE_MOVING	103	轴正在运动，不能进入该模式
ERR_PEL_STATE	104	轴处于正限位，不能正向运动
ERR_NEL_STATE	105	轴处于负限位，不能负向运动
ERR_SOFT_PEL_STATE	106	轴处于软正限位，不能正向运动
ERR_SOFT_NEL_STATE	107	轴处于软负限位，不能负向运动
ERR_FORCE_IN_OTHER_MODE	108	轴处于非点位模式，不能强制变位
ERR_MAX_VEL_ZERO	109	设置最大速度错误，不能为 0
ERR_EQU_ZERO	110	设置轴当量错误，不能为 0
ERR_BACKLASH_NEG	111	设置反向间隙错误，不能为负值
ERR_MAX_PULSE	112	设置位置错误，已超出允许范围
ERR_ALAM_STATE	115	轴处于报警状态，不能正向运动
ERR_EMG_STATE	116	轴处于急停状态，不能负向运动
ERR_AXIS_BUS_CONFIGURING	117	设置轴处于未使能状态
ERR_AXIS_NOT_ETC_BUS	118	设置轴非总线轴，不能进行该总线操作
ERR_AXIS_NOT_SERVO_ON	119	轴的使能信号关闭，不能启动
ERR_AXIS_ENABLE	120	轴使能状态，不允许设置
ERR_CMP_EXCEED_MAX_AXISES	121	所选比较轴超出范围
ERR_CMP_EXCEED_MAX_INDEX	122	比较点数已满，不能继续添加
ERR_CMP_EXCEED_MAX_IO	123	进行比较的 IO 超出范围
ERR_CMP_EXCEED_MAX_CHAN	124	选择的高速比较 IO 超出范围
ERR_PWM_IO_FULL	126	PWM 输出器已满
ERR_PWM_IO_COUNT	127	PWM 输出次数不能为 0
ERR_PWM_IO_BUSY	128	PWM 输出口已被占用
ERR_MAP_AXISIO_MAX_AXISES	130	映射的轴超出范围
PVT_ERROR_AXISES_OVER_RANGE	140	所选轴超出范围
PVT_ERROR_INDEX_OVER_RA	141	控制点已满，不能继续添加

NGE		
PVT_ERROR_INDEX_EXCEED	142	控制点已满，不能继续添加
PVT_ERROR_TIME_EROOR	143	插入段时间为 0 或者负数
HOME_ERROR_AXISES_OVER_RANGE	200	所选轴超出最大值
HOME_ERROR_SAMPLE_PERIOD	201	回零周期设置错误
HOME_ERROR_MAX_VEL	202	设置的最大速度为 0
HOME_ERROR_MAX_ACC	203	设置的加速度小于等于 0
HOME_ERROR_MAX_DEC	204	设置的减加速度小于等于 0
HOME_ERROR_MOVE_DISTANCE	205	回零距离不正确
HOME_ERROR_MOVE_MODE	206	错误的回零模式
HOME_ERROR_BOTH_LIMIT	207	同时处于正、负限位,无法启动回零运动
HOME_ERROR_WRONG_DIR	208	错误的回零方向
ERR_TRACE_HAS_STARTED	210	TRACE 功能已经启动，请停止后再次启动
ERR_2ND_MOVE_DIR	220	软着陆功能中的着陆反向运动
ERR_BUS_TIMEOUT	252	总线连接超时
ERR_TIMEOUT	258	通讯超时，检查接线及通讯参数配置
ERROR_OFFSET_TRACE_NO_MEMORY	501	没有内存空间
	502	总线错误，不允许操作
ERR_OVER_MAX_SLAVES	10000+1	超过轴个数限制
ERR_NO_ADDRESS	10000+3	地址不存在
ERR_NO_EXIST_AXIS	10000+4	轴不存在
ERR_AXIS_EXIST	10000+5	设置的轴号不存在
ERR_NO_EXIST_IO	10000+6	IO 点不存在
ERR_NO_SLAVE_TYPE	10000+8	设置的从站类型不存在
ERR_INVALID_PARA_SLAVE_ECT	10000+9	无效 EtherCAT 参数
ERR_INVALID_PARA_SDO	10000+11	无效 SDO 参数
ERR_INVALID_PARA_FIELDBUS_PORT	10000+12	无效主站
ERR_NO_MAPPING	10000+13	配置文件不存在
ERR_OVER_MAX_SLAVES_ECT	10000+15	超过 EtherCAT 从站个数限制
ERR_OVER_MASTERNUM	10000+16	设置的主站类型错误
ERR_FILEMANNAGE_ECT_NOEXIST	10000+17	映射文件不存在
ERR_FILESIZE_ECT_EMPTY	10000+18	配置文件为空
ERR_FILEFORMAT_ECT	10000+19	配置文件格式错误
ERR_MAPPINGINFO_ECT	10000+20	轴及 IO 映射错误
ERROR_ENI_INVALID	10000+21	下载的配置文件错误
ERROR_AXISRUNMODE	10000+22	轴运行模式设置错误

ERROR_FIELDBUS_TYPE	10000+23	总线类型错误
ERROR_PP_AXIS_MOVEING	10000+ 24	PP 模式下轴在运动中
ERR_PARAR	10000+ 25	参数错误

附录 3 运动控制函数索引

函数分类	函数名	描 述	索引
板卡设置函数	dmc_board_init	控制卡初始化函数	9.1 节
	dmc_board_close	控制卡关闭函数	
	dmc_get_CardInfList	获取控制卡硬件 ID 号	
	dmc_get_card_version	获取控制卡硬件版本号	
	dmc_get_card_soft_version	获取控制卡固件版本号	
	dmc_get_card_lib_version	获取控制卡动态库文件版本号	
	dmc_get_total_axes	获取当前卡的轴数	
	dmc_download_configfile	下载参数文件	
	dmc_download_firmware	下载固件文件	
脉冲当量设置函数	dmc_set_equiv	设置指定轴的脉冲当量	9.2 节
	dmc_get_equiv	读取指定轴的脉冲当量	
回原点运动函数	nmc_set_home_profile	设置 EtherCAT 总线轴回零参数	8.3 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	
限位开关设置函数	dmc_set_softlimit	设置软限位	9.2 节
	dmc_get_softlimit	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	9.7 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	读取当前速度值	9.8 节
	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	9.3 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	9.4 节
	dmc_vmove	指定轴连续运动	
	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）	
通用输入输出 IO 函数	dmc_read_inbit	读取指定控制卡的某一位输入口的电平状态	9.11 节

函数分类	函数名	描 述	索引
	dmc_write_outbit	设置指定控制卡的某一位输出口的电平状态	
	dmc_read_outbit	读取指定控制卡的某一位输出口的电平状态	
	dmc_read_inport	读取指定控制卡的全部输入口的电平状态	
	dmc_read_outport	读取指定控制卡的全部输出口的电平状态	
	dmc_write_outport	设置指定控制卡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	
手轮功能函数	dmc_set_handwheel_inmode	设置单轴手轮运动控制输入方式	9.12 节
	dmc_get_handwheel_inmode	读取单轴手轮运动控制输入方式	
	dmc_handwheel_move	启动手轮运动	
	dmc_set_handwheel_inmode_extern	设置多轴手轮运动控制输入方式	
	dmc_get_handwheel_inmode_extern	读取多轴手轮运动控制输入方式	
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	9.13 节
	dmc_get_encoder_unit	读取指定轴编码器反馈位置脉冲计数值	
	dmc_set_extra_encoder_mode	设置辅助编码器计数方式	
	dmc_get_extra_encoder_mode	读取辅助编码器计数方式	
	dmc_set_extra_encoder	设置辅助编码器计数值	
	dmc_get_extra_encoder	读取辅助编码器计数值	
高速位置锁存函数	dmc_ltc_set_mode	设置指定轴的 LTC 信号	9.14 节
	dmc_ltc_get_mode	读取指定轴的 LTC 信号设置	
	dmc_ltc_set_source	设置锁存源	
	dmc_ltc_get_source	读取锁存源	
	dmc_ltc_get_value_unit	读取锁存值	
	dmc_ltc_get_number	读取已锁存个数	
	dmc_ltc_reset	复位指定卡的锁存器的标志位	
位置比较函数	dmc_compare_set_config	设置一维位置比较器	9.15 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	添加一维位置比较点	
	dmc_compare_get_current_point	读取当前一维比较点位置	

函数分类	函数名	描 述	索引
	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	
高速位置比较函数	dmc_hcmp_set_mode	设置高速比较模式	9.16 节
	dmc_hcmp_get_mode	读取高速比较模式设置	
	dmc_hcmp_set_config	配置高速比较器	
	dmc_hcmp_get_config	读取高速比较器配置	
	dmc_hcmp_add_point	添加/更新高速比较位置	
	dmc_hcmp_set_liner	设置高速比较线性模式参数	
	dmc_hcmp_get_liner	读取高速比较线性模式参数设置	
	dmc_hcmp_clear_points	清除高速位置比较点	
	dmc_hcmp_get_current_state	读取高速比较参数	
异常信号接口函数	dmc_set_emg_mode	设置 EMG 急停信号	9.17 节
	dmc_get_emg_mode	读取 EMG 急停信号设置	
	dmc_set_io_dstp_mode	设置减速停止信号	
	dmc_get_io_dstp_mode	读取减速停止信号设置	
	dmc_set_dec_stop_time	设置减速停止时间	
	dmc_get_dec_stop_time	读取减速停止时间设置	
检测轴到位状态函数	dmc_set_factor_error	设置位置误差带	9.18 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_write_sn	修改密码	9.19 节
	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	9.20 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	9.10 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	9.5 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	9.6 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）	
EtherCAT	nmc_set_node_od	设置从站对象字典参数值	第 8 章

函数分类	函数名	描 述	索引
总线操作函数	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	
	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	

附录 4 常见问题解决方法

序号	问题描述	可能原因及解决方法

1	控制卡无法扫描到设备	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. 控制卡是否正常工作，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	控制卡连接多台 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. 控制卡是否正常工作，可冷复位系统完成后再扫描
4	板卡插上后,PC 机系统还不能识别控制卡	检查板卡驱动是否正确安装，在 WINDOWS 的设备管理器（可参看 WINDOWS 帮助文件）中查看驱动程序安装是否正常。如果发现有相关的黄色感叹号标志，说明安装不正确，需要按照软件部分安装指引，重新安装； 计算机主板兼容性差，请咨询主板供应商； PCI 插槽是否完好； PCI 金手指是否有异物，可用酒精清洗。
5	PC 机不能和控制卡通讯	PCI 金手指是否有异物，可用酒精清洗； 参考软件手册检查应用软件是否编写正确。
6	控制卡已经正常工作，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。
7	电机可以转动，但工作不正常	检查控制卡和驱动器是否正确接地，抗干扰措施是否做好。
8	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。
9	不能读入辅助编码器信号	请检查编码器信号类型是否是脉冲 TTL 方波； 参看所选编码器说明书，检查接线是否正确； 编码器供电是否正常； 检查函数调用是否正确。
10	对辅助编码器的读数不准确	检查全部编码器及触发源的接线； 做好信号线的接地屏蔽。
11	不能锁存辅助编码器读数	检查触发源的接线； 检查函数的调用是否正确。
12	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。
13	数字输入信号不能读取	接线是否正常；检查函数调用。
14	数字输出信号不正常	接线是否正常；检查函数调用。
15	电脑休眠后找不到卡	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：深圳市南山区塘朗学苑大道 1001 号南山智园 A3 栋 9 楼
邮　编：518055
电　话：0755-26415968
传　真：0755-26417609
Email：info@szleadtech.com.cn
网　址：http://www.szleadtech.com.cn