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

DMC5X10 系列运动控制卡

用户使用手册

2023.09.27

版 权 说 明

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

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

目 录

第 1 章　产品概述

1.1　DMC5X10 系列运动控制卡的特点

DMC5X10系列运动控制卡是雷赛控制技术有限公司开发出具有自主知识产权的新型运动控制卡。其CPU更高级，运动控制算法更完善，控制性能更快速、更优秀。其主要特点如下：

- 支持对称或非对称梯形、S形速度曲线规划。
- 支持点位运动、连续运动。支持在线变速、变位功能。
- 回零模式丰富，支持13种回零模式。
- 支持4个插补坐标系，每个坐标系支持2~12轴直线插补、任意平面圆弧插补、空间圆弧插补。
- 支持2~3轴螺旋线插补、两轴同心圆插补、两轴矩形插补等运动。
- 支持小线段前瞻连续插补运动，连续插补缓冲区可装5000条指令，支持圆弧限速。
- 支持反向间隙补偿功能，可降低机械传动反向间隙的影响。
- 支持螺距补偿功能，可根据激光干涉仪采集的补偿位置表进行实时补偿，保证设备更高精度。
- 支持软着陆功能，实现平稳停止。
- 支持龙门运动功能。
- 支持PWM直接输出、PWM跟随功能。
- 支持一维、二维低速位置比较输出功能，多达256个比较缓冲区，可灵活配置比较输出模式，比较周期可达1毫秒以内。
- 支持一维、二维高速位置比较输出功能。位置间时间间隔最小可达几微秒。
- 支持软件位置锁存、支持高速位置锁存功能：单次锁存、连续锁存、高速触发急停锁存。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发脉冲这种特殊应用的精确定位功能。
- 支持DA输出功能，可在指定的通道上输出需要的电压值（DMC5410A）。
- 支持多达5000个位置点的PVT运动曲线规划高级功能，根据位置点的相关数据：时间、位置、速度，实现在准确的时间点以准确的速度到达确定的位置；可通过自定义数据实现复杂轨迹多轴连续插补运动功能。
- 支持手轮运动配置功能。可任意配置一个轴或多个轴按不同的倍率跟随一个手轮运动。任意配置跟随轴增强了手轮复用性能，多轴跟随手轮可实现多轴协同精确定位功能。

- 支持IO输出延时翻转功能、IO计数及滤波功能。
- 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，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。
- 支持轴IO映射配置功能、虚拟IO映射配置功能。支持将轴信号配置到任意一个硬件输入口，如：可将限位接口当原点信号或通用输入口。该功能可减少现场接线、换线的困难。
- 支持CAN IO/ADDA扩展模块

1.2 DMC5X10 系列卡主要技术指标

表 1.1 DMC5X10 系列卡主要技术指标

技术指标 \ 卡类型	DMC5C10 DMC5C10-PCIe	DMC5810 DMC5810-PCIe	DMC5610 DMC5610-PCIe	DMC5410A DMC5410A-PCIe
电机轴数	12	8	6	4
支持在PC机中同时工作的卡数	8			
控制电机的脉冲信号频率范围	1 Hz～4 MHz			
控制电机的脉冲信号频率精度	1 Hz			
脉冲信号输出最大电流	20 mA （吸入）			
脉冲信号长度	28位有符号			
直线插补精度	±0.8 pulse			
圆弧插补精度	±1.5 pulse			
支持的插补坐标系个数	4			
编码器信号输入个数	8	8	6	4
编码器计数器长度	28位有符号			
编码器输入信号频率	4 MHz （4倍频后为16MHz）			
手轮输入信号最大频率	500 kHz			
通用数字输入口数量	16（可扩展）			
通用数字输出口数量	16（可扩展）			14（可扩展）
通用数字输入口	光电隔离，RC滤波			
通用数字输入口输入电流	5～10 mA			
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，漏型）			
CAN扩展	最多支持连接8个CAN扩展模块			
高速位置锁存输入口数量（LTC）	2			1
高速位置比较输出口数量（CMP）	4			2
机械正负限位输入口数量（±EL）	24	16	12	8
机械原点信号输入口数量（ORG）	12	8	6	4
伺服到位信号输入口数量（INP）	8	8	6	4
伺服报警信号输入口数量（ALM）	8	8	6	4

卡类型 / 技术指标	DMC5C10 DMC5C10-PCIe	DMC5810 DMC5810-PCIe	DMC5610 DMC5610-PCIe	DMC5410A DMC5410A-PCIe
伺服准备好信号输入口数量（RDY）	8	8	6	4
伺服使能信号输出口数量（SEVON）	8	8	6	4
伺服误差清除信号输出口数量（ERC）	8	8	6	4
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 ，10A			
尺寸大小（mm）	182.00(长)×106.68(高)			
驱动程序	支持Windows XP、Windows 7、Windows 10、Linux操作系统			
运动控制函数库	支持C/C++、C#、VB6.0、VB.NET、LabVIEW等多种语言			
调试软件	免费提供Motion调试软件			

1.3 DMC5X10 系列运动控制卡的典型应用

DMC5X10系列运动控制卡已广泛应用于各行各业自动化设备中。主要设备有：

- 电子产品加工、装配设备，如：丝印机、贴片机、PCB钻孔机等
- 激光加工设备，如：激光打标机、激光切割机等
- 机器视觉及自动检测设备，如：影像测量仪、电路板自动检测设备等
- 生物、医学自动采样、处理设备
- 专用工业机器人

1.3.1 DMC5C10 卡的典型应用

图 1.1 为 DMC5C10 运动控制卡组成的 12 轴运动控制系统的典型结构图。

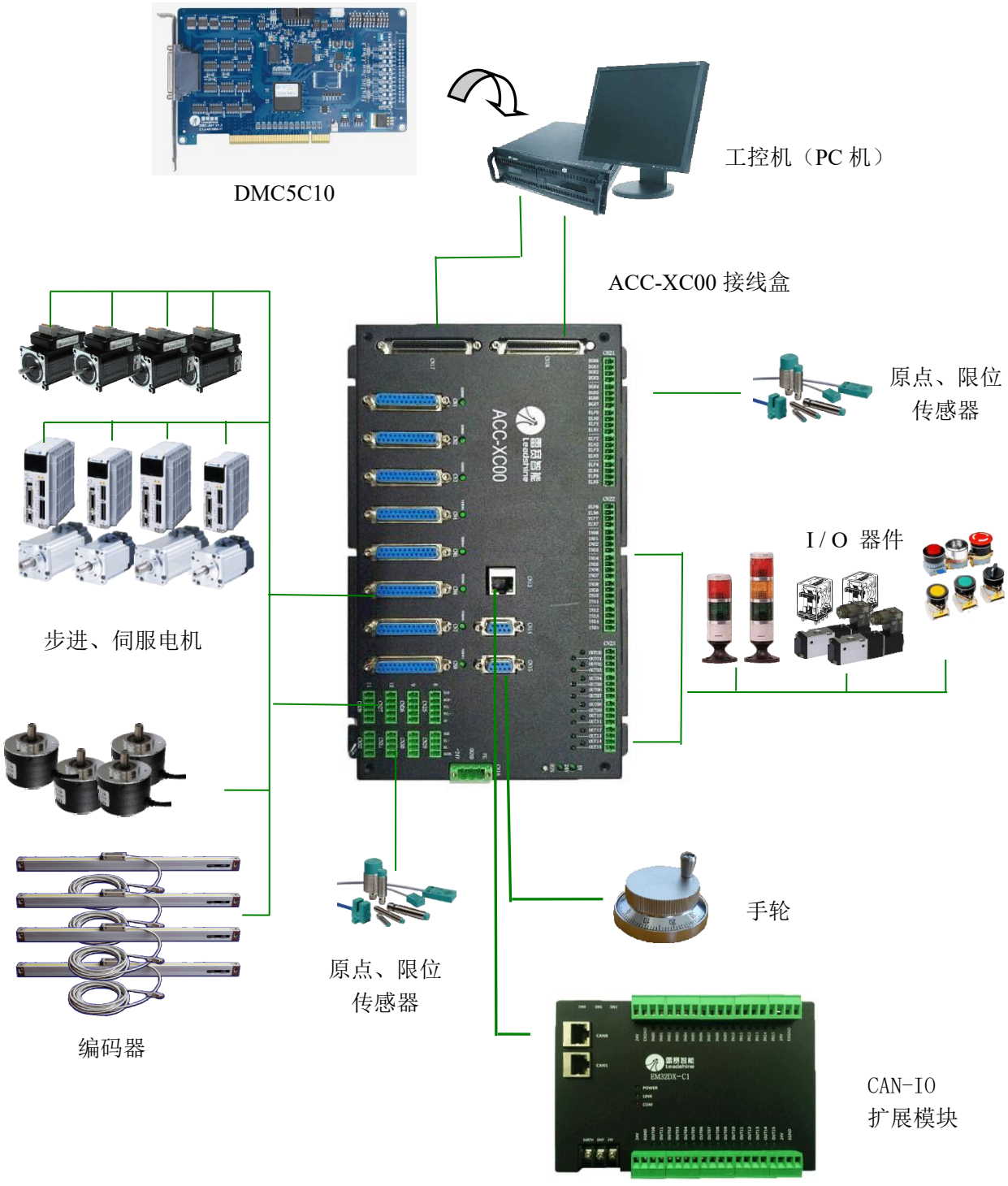


图 1.1　DMC5C10 运动控制卡组成的运动控制系统结构图

1.3.2　DMC5810 卡的典型应用

图 1.2 为 DMC5810 运动控制卡组成的 8 轴运动控制系统的典型结构图。

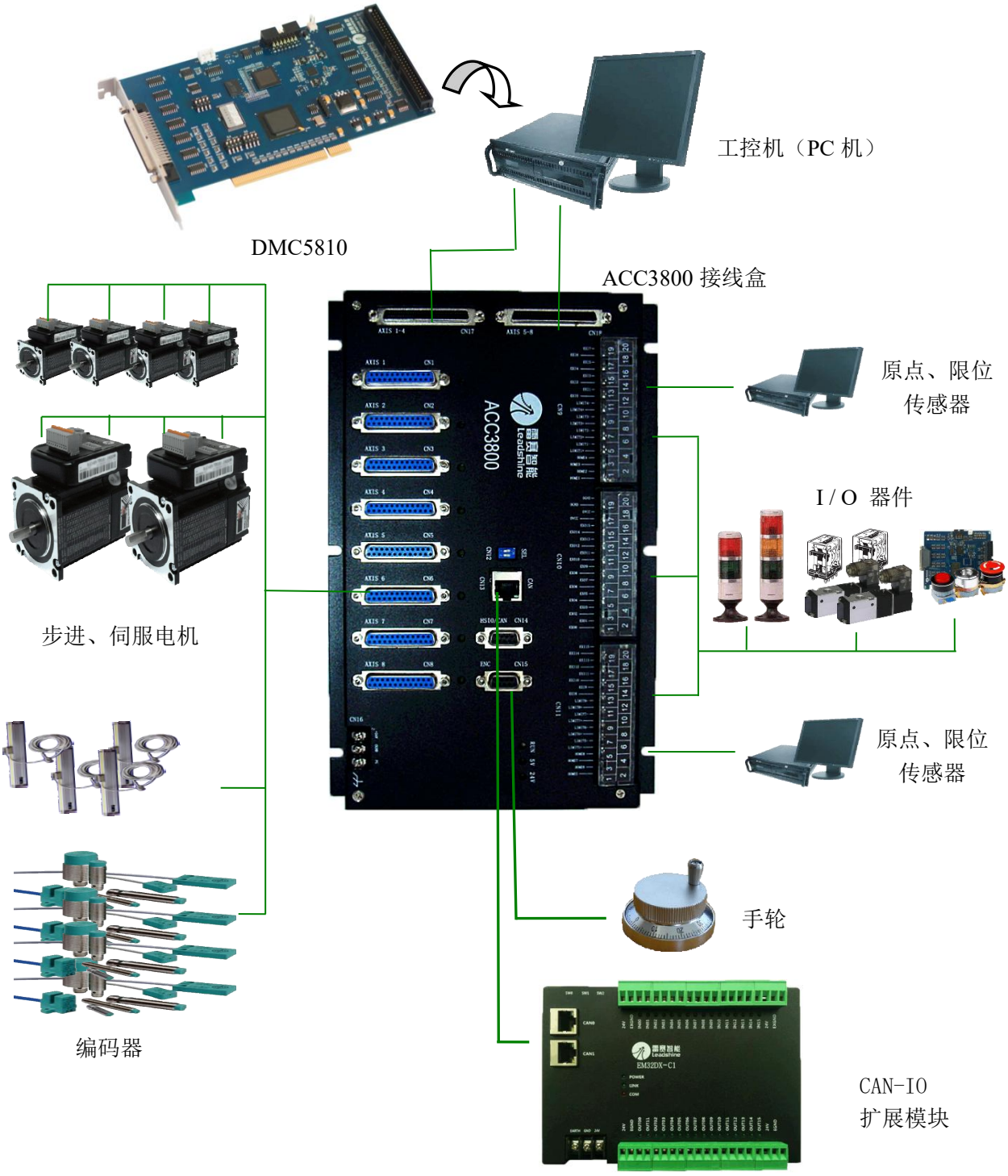


图1.2　DMC5810运动控制卡组成的运动控制系统结构图

1.3.3 DMC5610 卡的典型应用

图 1.3 为 DMC5610 运动控制卡组成的 6 轴运动控制系统的典型结构图。

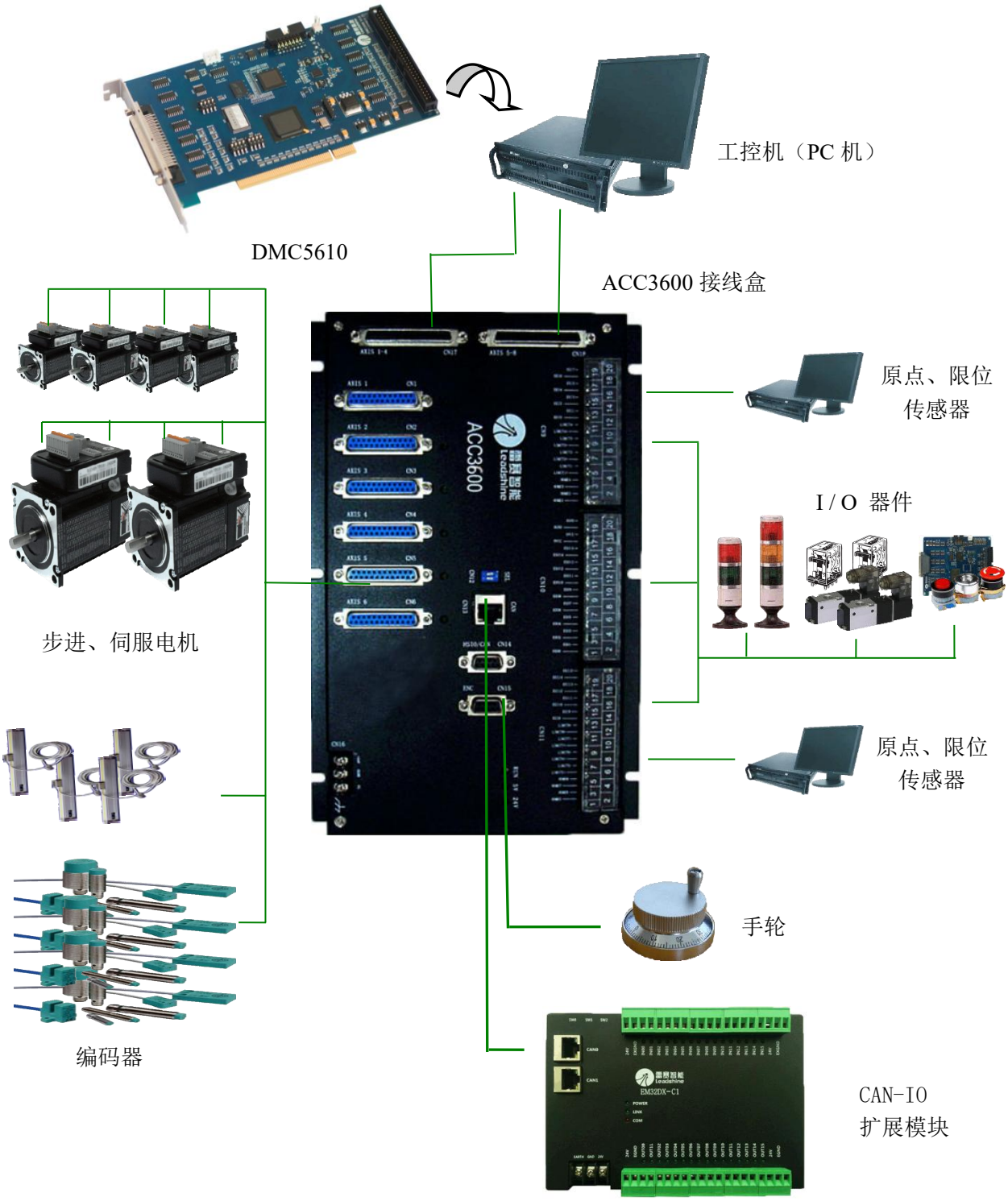


图1.3 DMC5610运动控制卡组成的运动控制系统结构图

1.3.4 DMC5410A 卡的典型应用

图 1.4 为 DMC5410A 运动控制卡组成的 4 轴运动控制系统的典型结构图。

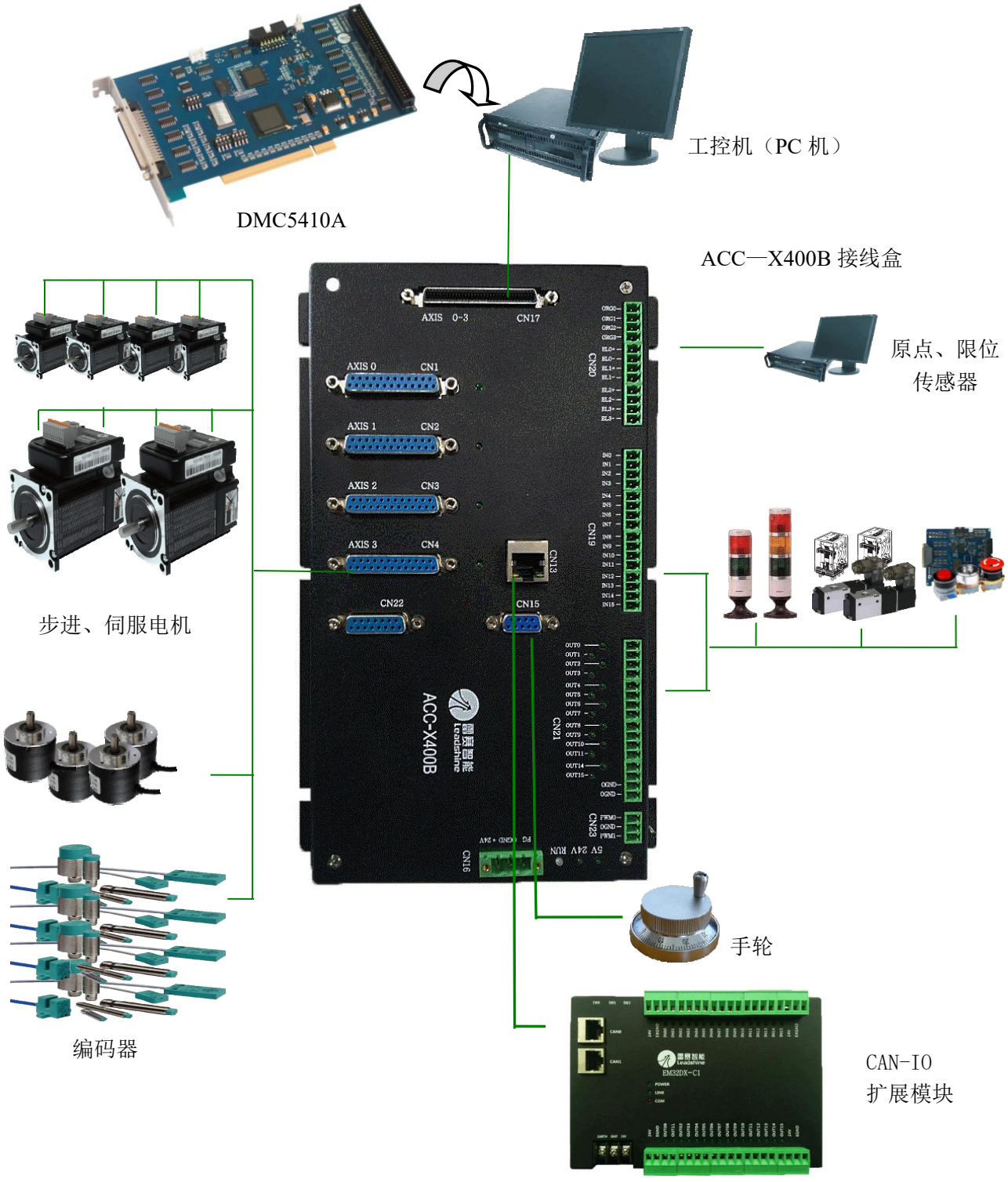


图1.4 DMC5410A运动控制卡组成的运动控制系统结构图

1.4 订货信息

表 1.2 DMC5C10 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5C10	80.00.15.011400	必选
ACC-XC00 接线盒	80.15.00.010400	必选
电缆线HPCN68P转MINI68P-20 1-35双绞线（2米）	10.25.03.010570	必选
CAN-IO扩展模块		选配

表 1.3 DMC5810 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5810	80.00.15.011350	必选
接线盒ACC3800	80.15.00.010900	必选
转接板ACC64TO68组合件	80.15.99.011250	必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

表 1.4 DMC5610 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5610	80.00.15.011300	必选
接线盒ACC3600	80.15.00.010850	必选
转接板ACC64TO68组合件	80.15.99.011250	必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

表 1.5 DMC5410A 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5410A	80.00.15.011250	必选
ACC-X400B 接线盒	80.15.00.010300	必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

表 1.6　DMC5C10-PCIe 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5C10-PCIe		必选
ACC2-XC00 接线盒		必选
电缆线HPCN68P转MINI68P-20 1-35双绞线（2米）	10.25.03.010570	必选
CAN-IO扩展模块		选配

表 1.7　DMC5810-PCIe 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5810-PCIe		必选
接线盒ACC2-3800		必选
转接板ACC64TO68组合件	80.15.99.011250	必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

表 1.8　DMC5610-PCIe 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5610-PCIe		必选
接线盒ACC2-3600		必选
转接板ACC64TO68组合件	80.15.99.011250	必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

表 1.9　DMC5410A-PCIe 运动控制卡及主要配件订货代码

名称	订货代码	说 明
运动控制卡DMC5410A-PCIe		必选
ACC2-X400B 接线盒		必选
68芯电缆线CABLE68-NR-20（2米）	10.25.03.010650	必选
CAN-IO扩展模块		选配

1.5 产品图片

运动控制卡 DMC5410A、DMC5810、DMC5610 及 DMC5C10 卡的外观示意图如图 1.5、1.6 所示。其中 DMC5810 卡与 DMC5610、DMC5410A 卡的外观相同。

图1.5 DMC5810/5610/5410A运动控制卡外观示意图

图1.6 DMC5C10运动控制卡外观示意图

运动控制卡 DMC5410A-PCIe、DMC5810-PCIe、DMC5610-PCIe 及 DMC5C10-PCIe 卡的外观示意图如图 1.7、1.8 所示。其中 DMC5810-PCIe 卡与 DMC5610-PCIe、DMC5410A-PCIe 卡的外观相同。

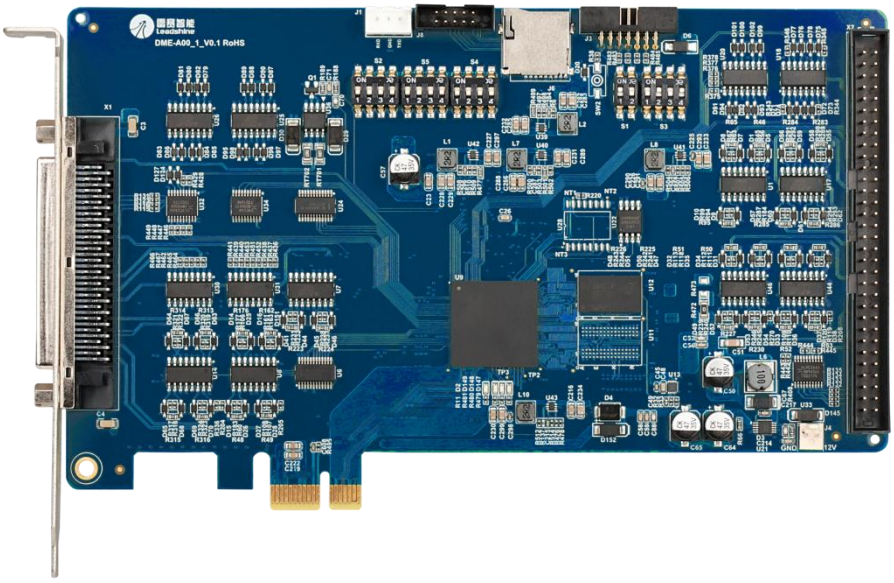


图1.7　DMC5810-PCIe/5610-PCIe/5410A-PCIe运动控制卡外观示意图

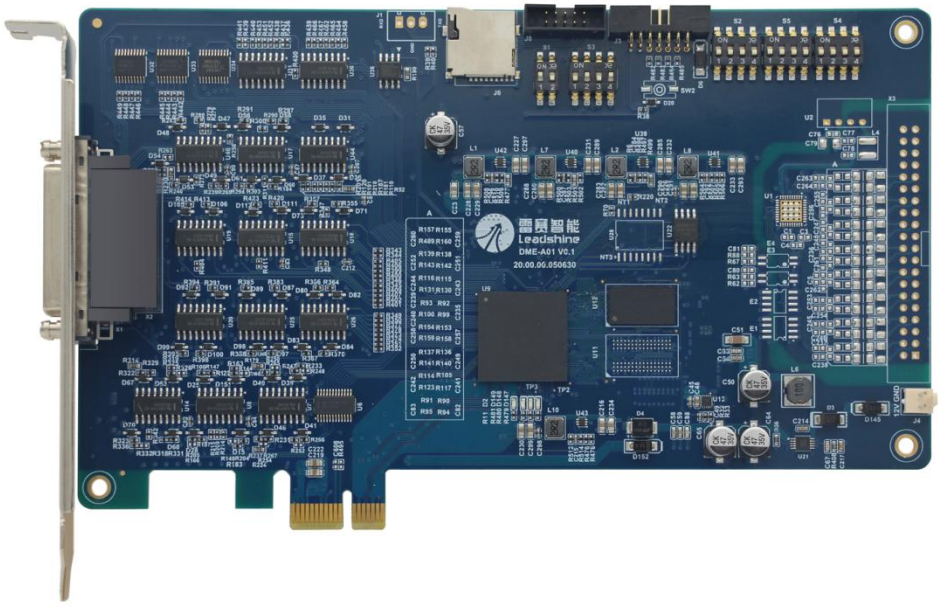


图1.8　DMC5C10-PCIe运动控制卡外观示意图

第 2 章　DMC5X10 系列卡功能介绍

雷赛 DMC5X10 系列运动控制卡是一款新型的 PCI/PCIe 总线运动控制卡。可以控制多个步进电机或数字式伺服电机；适合于多轴点位运动、插补运动、轨迹规划、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。

DMC5X10 系列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 Motion 调试软件，不但可以演示 DMC5X10 系列卡的控制功能，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。点位运动的运动距离由脉冲数决定，运动速度由脉冲频率决定。

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，使其按设定的速度输出脉冲；当输出脉冲数等于命令脉冲数时，运动控制卡停止脉冲输出。位移与时间的关系如图 2.1 所示。

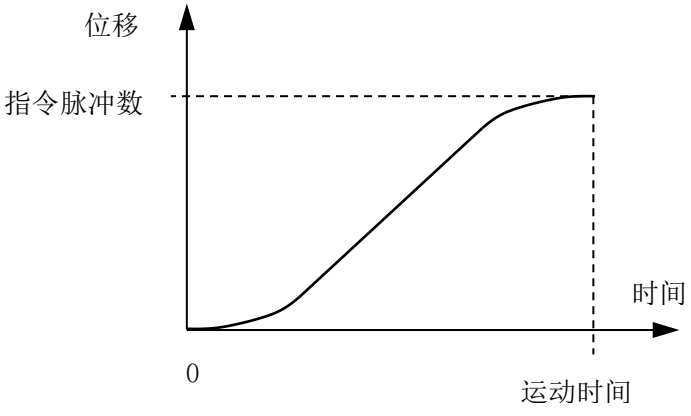


图2.1　定长运动位移曲线

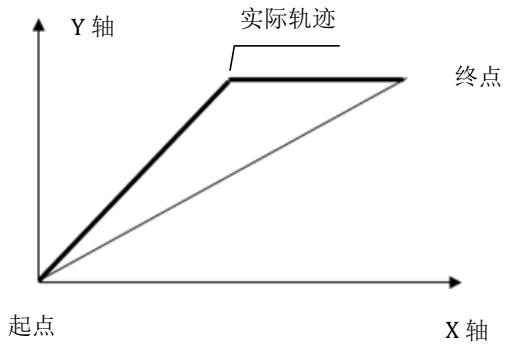


图2.2　两轴复合运动的轨迹

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。

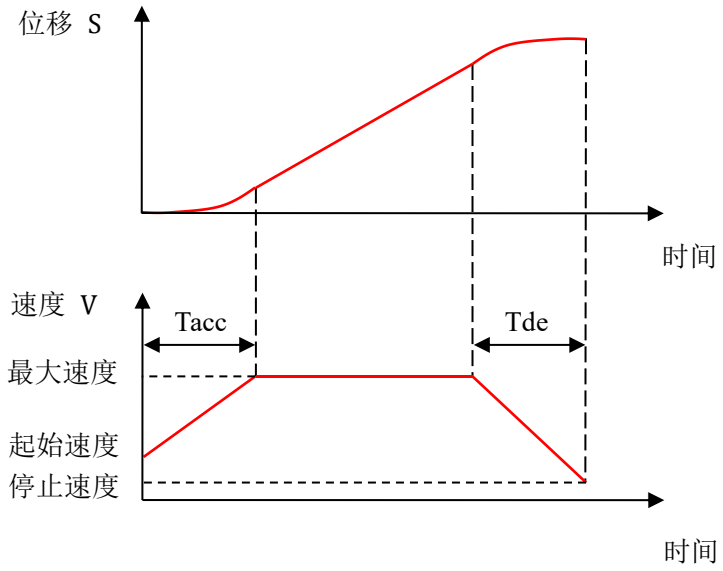


图2.3 梯形速度曲线及对应的位移曲线

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，DMC5X10 系列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，指令脉冲频率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，指令脉冲频率作 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，这时脉冲输出停止，如图 2.4 所示。

2.1.1.3 运动中改变终点位置

在进行点位运动的过程中，DMC5X10 系列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 个脉冲；但当运动了 550ms 时，将终点改为 100000 个脉冲，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。

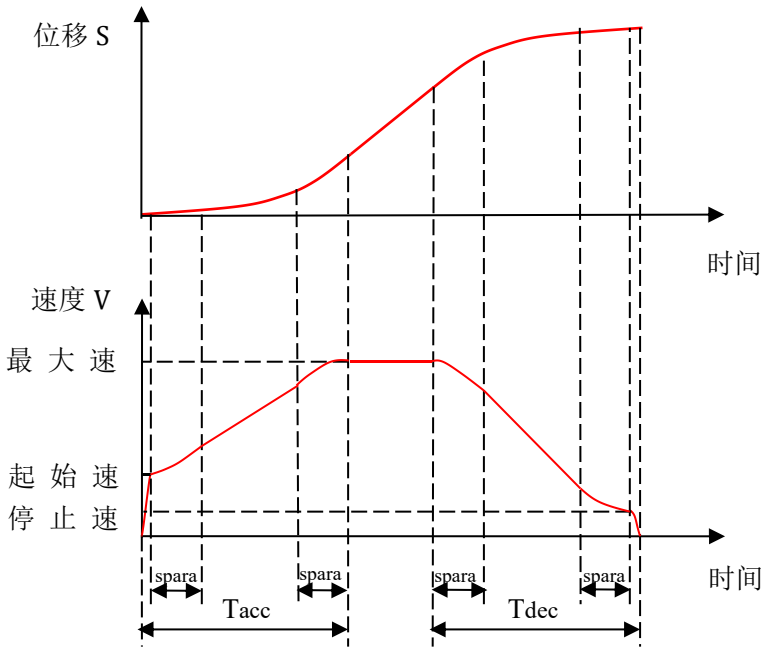
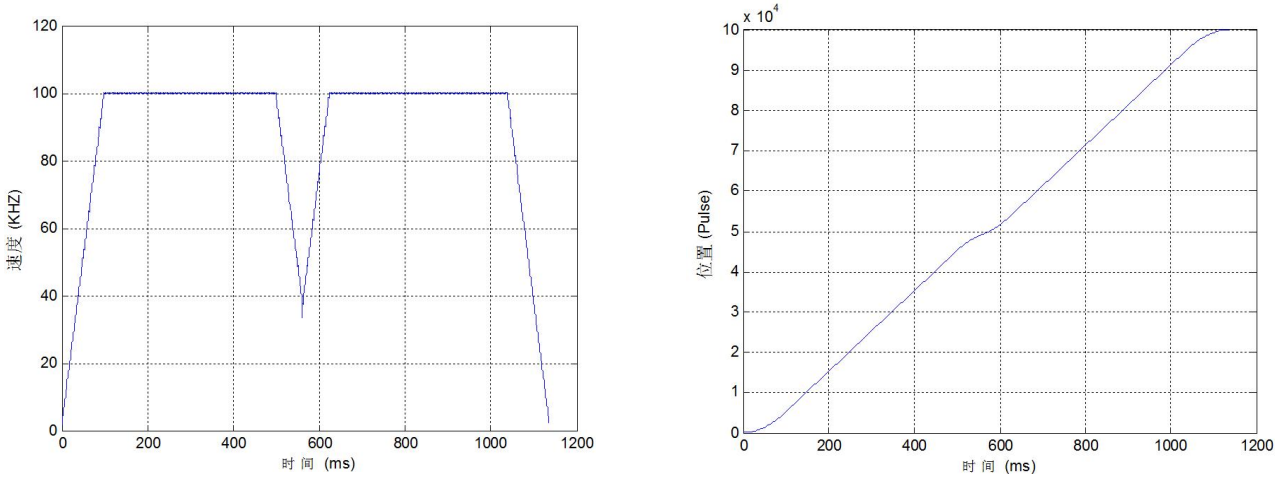


图2.4　S形速度曲线及对应的位移曲线

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　DMC5X10系列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，DMC5X10 系列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。

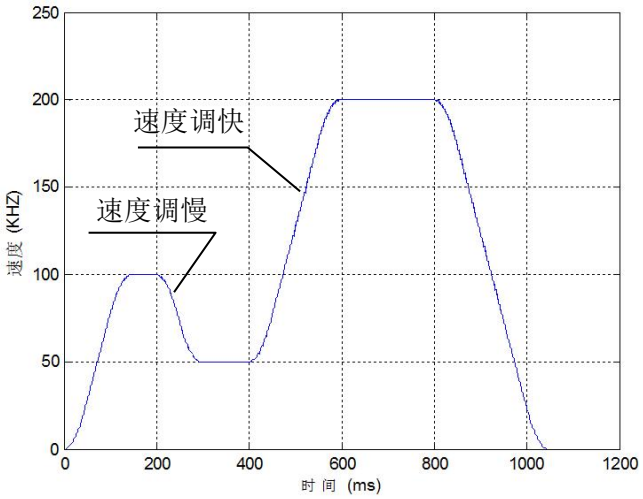


图2.6　DMC5X10系列卡的调速过程

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。

DMC5X10 系列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、急停信号才会按启动时的速度曲线减速停止。连续运动指令的速度与时间曲线如图 2.7 所示。

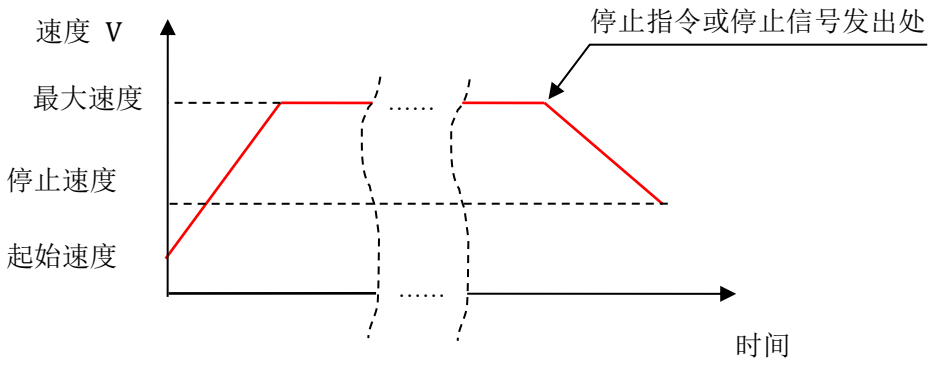


图2.7　连续运动速度曲线

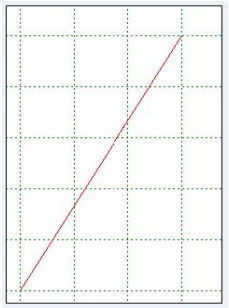
该功能的主要用途是：速度控制，如：传送带的速度、包装机连续送料速度等。

2.1.3　插补运动

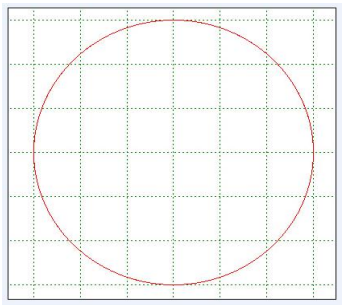
为了实现轨迹控制，运动控制卡按照一定的控制策略控制多轴联动，使运动平台用微小直

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、抛物线插补等。

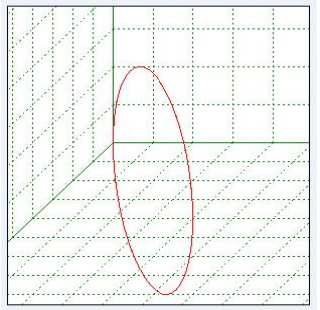
如图 2.8 所示为各种曲线轨迹的示意图。



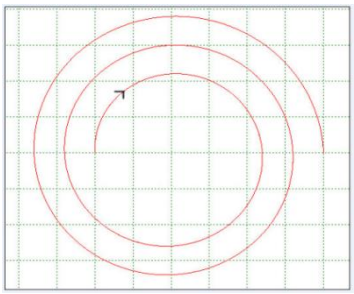
两轴直线插补



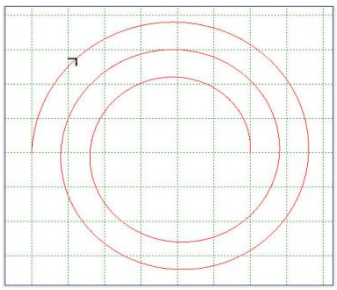
两轴圆弧



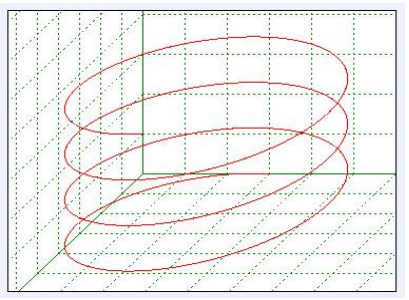
空间圆弧



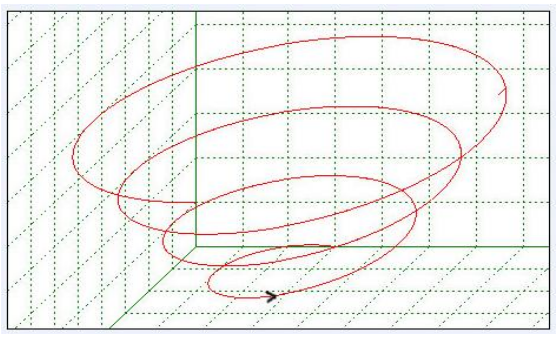
两轴螺旋线（绽放型）



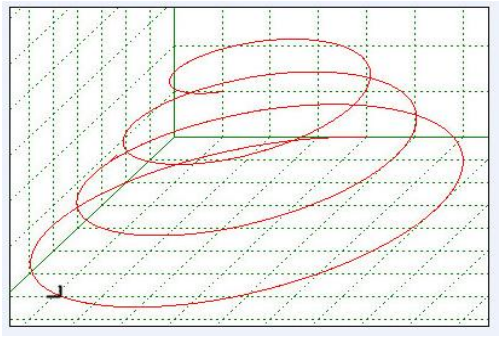
两轴螺旋线（收敛型）



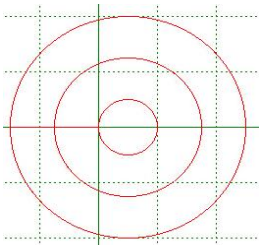
圆柱螺旋线



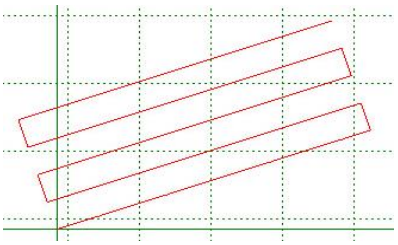
空间螺旋线（绽放型）



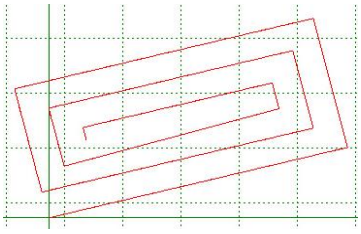
空间螺旋线（收敛型）



同心圆



逐行模式矩形插补



渐开线模式矩形插补

图 2.8　各种曲线轨迹示意图

DMC5C10 可以进行任意 2~12 轴直线插补，DMC5810 卡可以进行任意 2~8 轴直线插补，DMC5610 卡可以进行任意 2~6 轴直线插补,DMC5410A 可以进行任意 2~4 轴直线插补；同时，DMC5X10 系列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、两轴同心圆插补、两轴矩形插补等。此外，当插补轴数大于 3 时，DMC5X10 系列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 连续插补运动

DMC5X10 系列卡提供了连续插补运动功能，采用连续插补可实现速度的平滑过渡，减小机器的振动，生产出高质量的工件，同时能提高机器的加工速度。连续插补指令支持直线插补，圆弧插补，螺旋线插补，IO 控制等。各种曲线轨迹示意图见图 2.8。

2.1.5 小线段插补

小线段前瞻插补功能能够提前对加工轨迹进行分析和处理，发现高曲率点和尖锐拐角，然后对路径上的速度进行规划，找出减速点，满足加工精度的同时满足机器的加减速特性，在保证速度最大化的同时实现速度的平滑过渡。通过前瞻控制可以避免高速加工过程中由于速度急剧变化对机床产生的冲击，提高加工精度。小线段前瞻插补指令支持直线插补，圆弧插补，螺旋线插补等。

2.1.6 圆弧限速

在插补运动中支持圆弧限速功能，可以减小设备冲击，保证加工精度，提升圆弧加工品质。

2.1.7 PVT 运动功能

DMC5X10 系列卡共有四种 PVT 模式，分别为 PTT、PTS、PVT、PVTS 模式。其中 PTT、PTS 运动模式用于单轴速度规划；PVT、PVTS 运动模式则用于多轴轨迹规划。用户可以根据实际需求选择适合的 PVT 模式。

2.1.7.1 单轴速度规划功能

DMC5X10 系列卡中提供了两种 PVT 模式来实现单轴速度规划，分别为 PTT 运动模式和 PTS 运动模式。

1. PTT 运动模式

PTT 模式中第一个字母 P 表示位置（Position）、第二个字母 T 表示时间（Time），最后一个字母 T 表示梯形（Trapezoid）；PTT 模式表示在梯形速度曲线下规划点位运动。

用户通过输入一系列位置和时间参数，自定义单轴的运动规律。首先将速度曲线分割成若

干段，如图 2.9 所示。

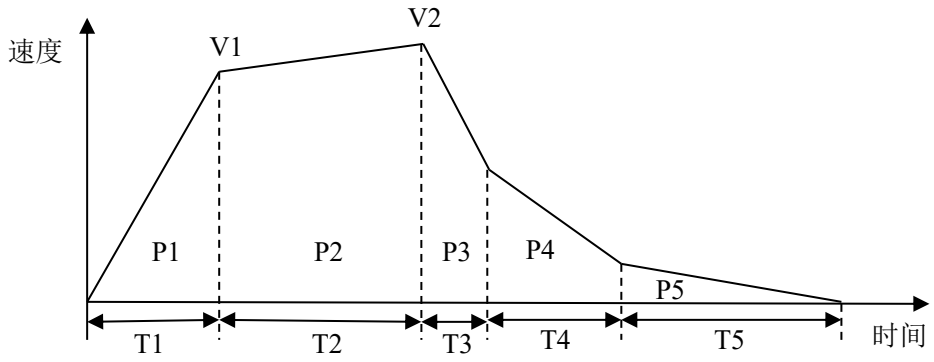


图 2.9　PTT 运动速度曲线

整个速度曲线被分割成 5 段。

第 1 段起点速度为 0；经过时间 T1 后，位移量为 P1，即速度曲线所围面积。因此第 1 段的终点速度为 V1=2×P1/T1；

第 2 段起点速度为 V1，经过时间 T2 后，位移量为 P2，即速度曲线所围面积。因此第 2 段的终点速度为 V2=2×P2/T2-V1；

第 3、4、5 段依此类推。

在定义一段完整的 PTT 运动时，第 1 段的起点位置和时间被设为 0；各段的终点位置和时间都是相对于该段起点的相对值。位置单位为脉冲，时间单位为秒。

运动控制卡执行 PTT 运动时，根据用户定义的各段位移和时间参数，计算各点的速度，形成一条连续的速度曲线。

2. PTS 运动模式

PTS 运动模式是 PTT 的扩展功能模式。PTS 的最后一个字母 S 表示 S 形速度曲线；PTS 模式是在 S 形速度曲线下规划点位运动；和 PTT 模式相比，其各段速度过渡更加平滑。

用户通过输入一系列位置、时间、百分比参数，自定义单轴的运动规律。其中位置、时间参数定义和 PTT 模式相同；“百分比”参数是指：相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比。

以图 2.10 为例说明之。数据点 P2 和 P3 之间加速度不变，因此数据点 P2 的百分比为 0。数据点 P3 和 P4 之间加速度变化时间为 2×ta，速度变化时间为 2×ta+te，因此数据点 P3 的百分比为[2×ta/(2×ta+te)]×100%。

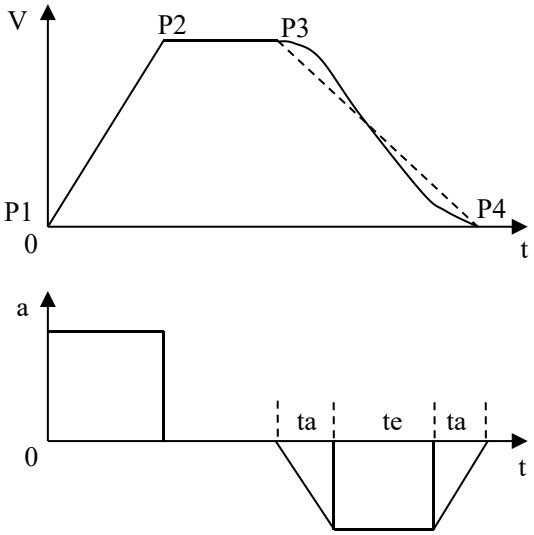


图 2.10 加速度变化时间百分比定义

调整百分比参数，可以改变 S 形速度曲线的形状。如图 2.10 所示，当数据点 P3 的百分比为 0 时，数据点 P3 和 P4 之间的速度曲线为直线（如图 2.10 中虚线所示）；当数据点 P3 的百分比不为 0 时，数据点 P3 和 P4 之间的速度曲线为 S 形曲线（如图 2.10 中实线所示）。“百分比”参数越大，S 段曲线越长。

2.1.7.2 多轴高级轨迹规划功能

当直线插补、圆弧插补不能满足轨迹规划的需求时，可以使用 DMC5X10 系列卡的两种高级 PVT 运动功能。

DMC5X10 系列卡提供的两种多轴轨迹规划的功能分别为 PVT、PVTS 运动模式；第二字母 V 表示速度（Velocity），最后一个字母 S 表示平滑。

PVT 模式用于对各点的位置、速度、时间都有要求的轨迹规划；PVTS 模式用于只对各点的位置、时间有要求，而对各点的速度无严格要求的轨迹规划。

1. PVT 运动模式

PVT 模式使用一系列数据点的位置、速度、时间参数自定义运动规律。

DMC5X10 系列卡采用 3 次样条插值算法对位置、速度和时间参数进行曲线拟合。即位置、速度曲线满足 3 次多项式函数关系。

位移方程为： $p = at^3 + bt^2 + ct + d$

速度方程为： $v = \frac{dp}{dt} = 3at^2 + 2bt + c$

如果给定轨迹上的一组“位置、速度、时间”参数，即可用 3 次样条函数逼近该轨迹。如图 7.21 所示为采用 PVT 模式拟合平面椭圆的轨迹图。

注意：每个点的位置与速度参数需要仔细设计，否则难以得到理想的运动轨迹。

2. PVTS 运动模式

PVTS 运动模式是 PVT 模式的简化模式。PVTS 运动模式只需要定义各数据点的位置、时间参数，以及起点速度和终点速度。运动控制卡根据各数据点的位置、时间参数计算运动轨迹的速度，确保各数据点速度连续和加速度连续。

由于对各点速度没有特殊要求，因此使用 PVTS 运动模式可以得到更平滑的运动轨迹。如图 7.23 所示为采用 PVTS 模式拟合空间圆弧的轨迹图。

2.1.8 回原点运动

如图 2.11 所示，在运动平台上，每个轴都有一个位置传感器用于设置一个位置参考点，即原点位置，以便进行位置控制。在正常运动之前，都需要用回零指令控制平台向原点方向运动，当控制卡检测到原点信号 ORG 后，平台自动停止，并将停止位置作为该轴的原点。

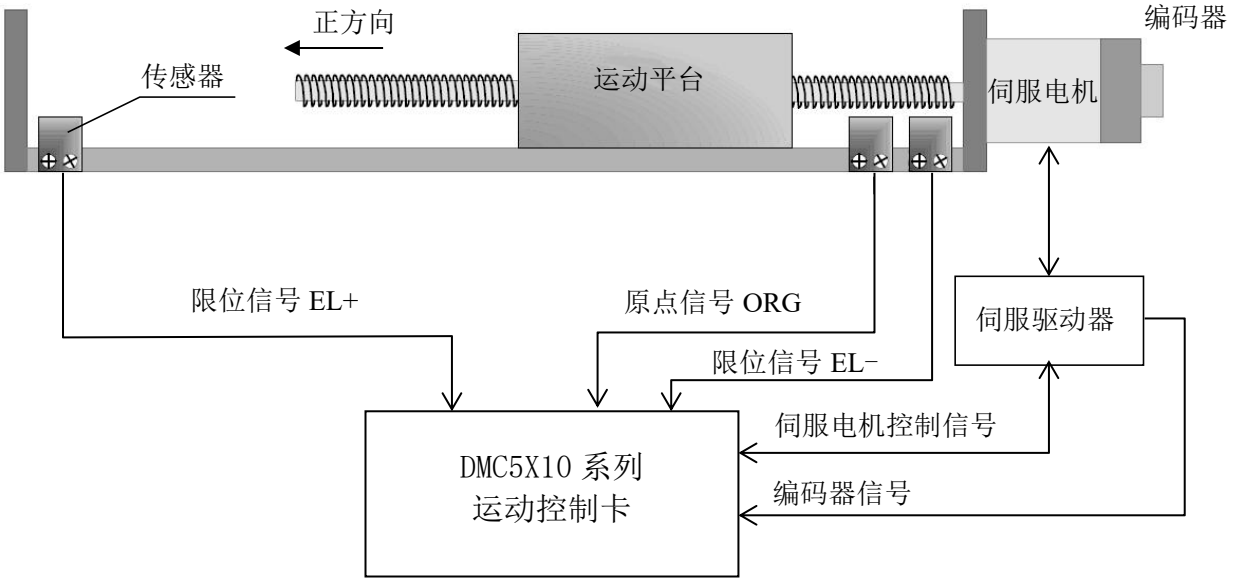


图2.11 运动平台传感器信号及电机控制信号与运动控制卡的关系

2.1.9 异常减速停止时间设置功能

DMC5X10 系列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。

2.1.10 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，DMC5X10 系列卡提供了手摇脉冲发生器（简称：手轮）控制功能。用户操控接在 DMC5X10 系列卡上的手摇脉冲发生器（参见图

1.1、图 1.2、图 2.12），手摇得慢，电机就转动得慢；手摇得快，电机就转动得快。

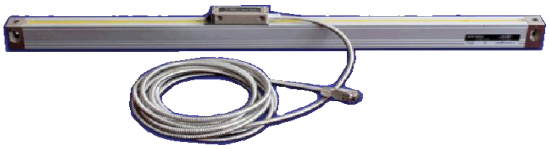
该功能多用于加工轨迹起点调整、刀具对位等工作中。

图2.12　手持式手摇脉冲发生器外形

2.2　编码器位置检测

DMC5X10 系列卡每轴都有一个编码器输入接口用于检测平台的位移或电机的转角。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 2.13 所示。

旋转编码器



光栅尺

图2.13　编码器外形

图2.14　被测工件与探针

采用探针和编码器配合使用，DMC5X10 系列卡通过位置触发功能，可完成对工件的位置检测工作，如图 2.14 所示。即：当探针接触到工件时，产生一个触发信号；DMC5X10 系列卡接受该信号后，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。

2.3　专用 IO 和通用 IO 控制

DMC5X10 系列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。专用 IO 信号用于原点、限位、伺服电机等控制，有专用控制指令相对应。

2.3.1 原点信号和限位信号

常用的专用 IO 信号有：运动平台上用于正向行程限位的传感器信号 EL+、反向行程限位的传感器信号 EL-；以及用于平台定位的原点传感器信号 ORG 等。参见图 2.11。

2.3.2 伺服电机控制信号

设备中有交流伺服电机时，使用 DMC5X10 系列卡上的伺服电机控制信号 SEVON、RDY、ALM、INP 和 ERC 就显得十分方便。

SEVON 是控制卡输出给伺服电机驱动器的控制信号，当 SEVON 信号为无效状态时，伺服驱动器不使能，电机处于自由状态；当 SEVON 信号有效时，伺服驱动器使能，电机锁紧，等待指令脉冲信号。

RDY 是伺服电机驱动器发给控制卡的状态信号，当 RDY 信号为有效时，表示伺服驱动器已经准备好，控制卡接收到该信号后就可以向伺服驱动器发出运动命令；如果 RDY 信号无效，表示伺服驱动器还未准备好，这时控制卡发出指令脉冲信号，伺服驱动器也不会运动。

ALM 信号是从伺服电机驱动器发给控制卡的状态信号，用来报告伺服驱动器或电机出错。控制卡接收到 ALM 信号时，将立即停止发出脉冲，该过程是一个硬件处理过程。

INP 信号是从伺服电机驱动器发给控制卡的状态信号，告知运动控制卡伺服电机已经停止。

伺服电机驱动器通常有一个位置偏差计数器，记录指令脉冲和位置反馈脉冲之间的偏差。伺服电机驱动器将控制电机运动使位置偏差趋于 0，但是电机实际位置总是滞后于指令脉冲。所以当运动控制卡的指令脉冲发送完毕时，伺服电机并没有立即停止，而是继续运动，如图 2.15 所示，直到位置偏差趋于 0；这时驱动器将发出一个 INP 信号。

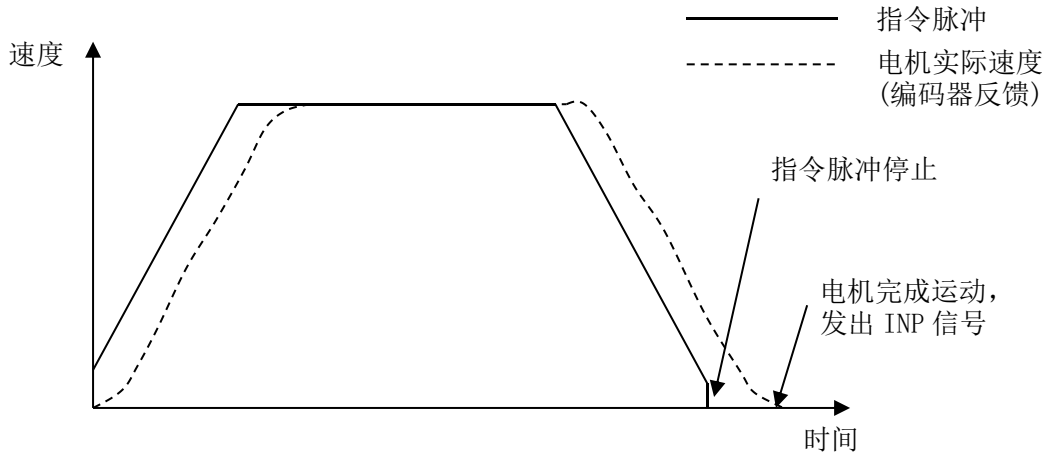


图 2.15　伺服定位完成时的 INP 信号

ERC 信号是控制卡输出给伺服电机驱动器的控制信号。

伺服驱动器依赖电机目标位置（即要求电机达到的位置）和当前位置（即电机已经到达的位置）之间的误差来驱动电机运动，若这个误差为零电机将停止运动。当伺服驱动器接收到运动控制卡发出的 ERC 信号后，会立即清除误差并停止电机运转。

2.3.3 位置比较输出

DMC5X10 系列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低速位置比较。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。

2.3.4 高速位置锁存

DMC5X10 系列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发脉冲这种特殊应用的精确定位功能。

2.3.5 通用 IO 信号

如图 1.1、图 1.2 所示，在电机运动的同时，DMC5X10 系列卡还可接受按键、数字式传感器等信号，控制指示灯、继电器、电磁阀等器件。

2.3.6 轴 IO 映射功能

DMC5X10 系列卡支持轴 IO 映射配置功能，支持将轴专用 IO 信号配置到任意一个硬件输入口，如：可将限位接口当原点信号。该功能可减少现场接线、换线的困难。

2.3.7 虚拟 IO 映射功能

DMC5X10 系列卡支持虚拟 IO 映射功能。通过该功能函数的设置可以实现专用通用 IO 输入接口的滤波功能，并且可以通过专用函数读取该端口滤波后的电平状态。

2.3.8 CAN 扩展模块

当设备需求的 IO 和 ADDA 端口数量较多时，DMC5X10 系列卡可配套 CAN 扩展模块对通用输入输出端口进行扩展。

2.4 PWM 输出功能

DMC5X10 系列卡提供了 PWM 输出功能。PWM（脉宽调制）基本原理就是对逆变电路开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需要的波形。

2.5 多卡运行

DMC5X10 系列卡的驱动程序支持最多 8 块 DMC5X10 系列卡同时工作。因此，一台 PC 机可以同时控制多达 96 个电机同时运动。

DMC5X10 系列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。DMC5C10 卡上 12 个轴的编号为 0~11；DMC5410A 卡上四个轴的编号为 0~3 号；DMC5810 卡上的八个轴的编号为 0~7 号；DMC5610 卡上的六个轴的编号为 0~5 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

DMC5X10 系列卡兼容 PCI V2.3 标准的 32Bit PCI 标准半长卡规范。硬件接口电路有：4 轴脉冲和方向控制信号、伺服电机控制信号、编码器信号、机械限位与原点信号、手轮脉冲信号以及通用输入输出信号等。具体硬件系统框图如图 3.1 所示。

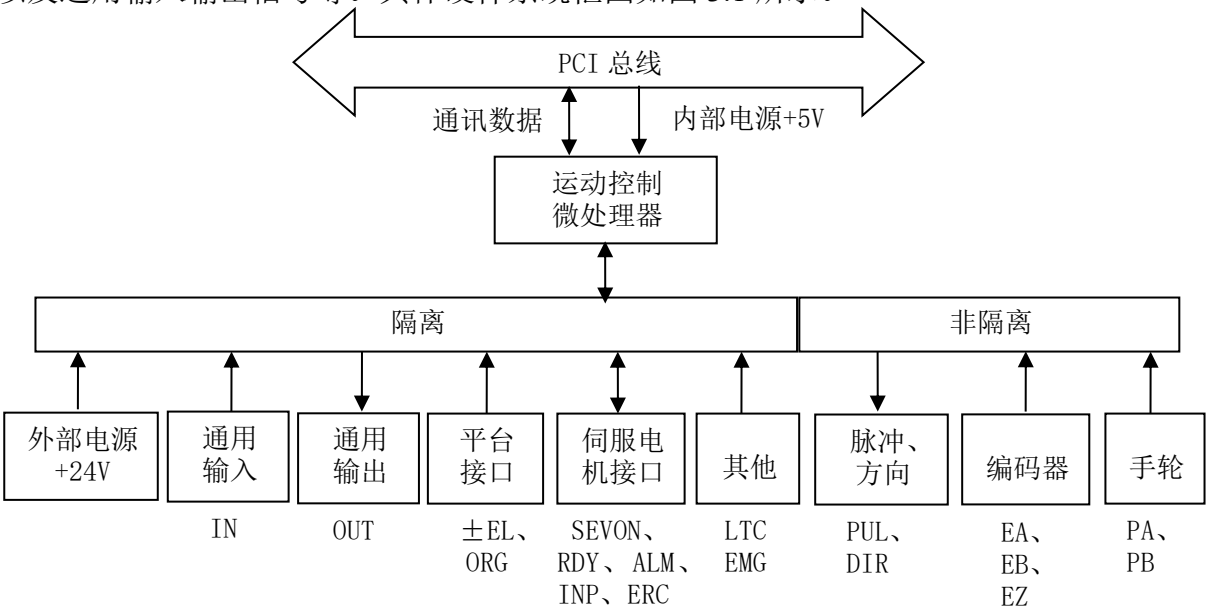


图3.1　运动控制卡硬件系统框图

DMC5X10 系列运动控制卡硬件布置及尺寸如图 3.2 所示。

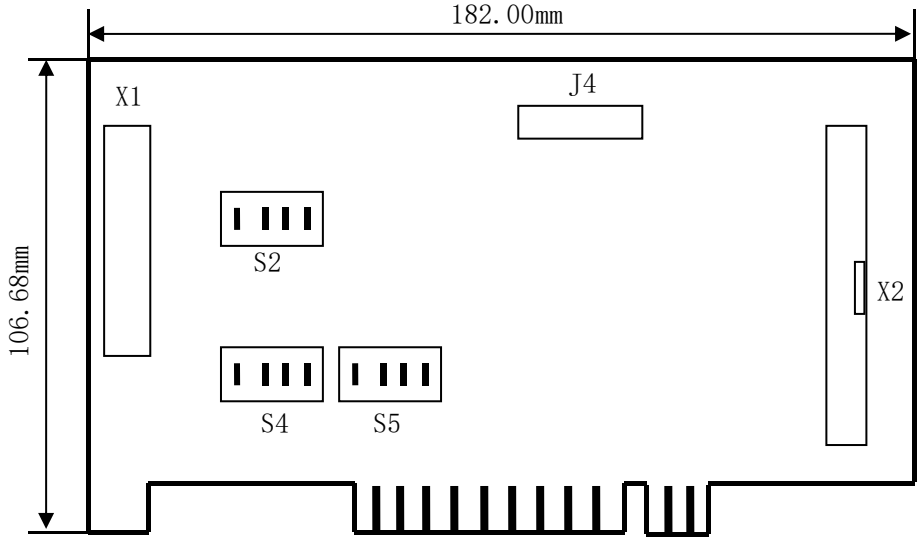


图3.2　DMC5X10系列卡硬件布置及尺寸图

3.2 控制卡与配件的连接

3.2.1 DMC5C10 控制卡与配件的连接

DMC5C10 控制卡有一个必配的接线盒 ACC-XC00，连接示意图如图 3.3 所示，DMC5C10-PCIe 控制卡有一个必配的接线盒 ACC2-XC00，连接方式与 PCI 一致。

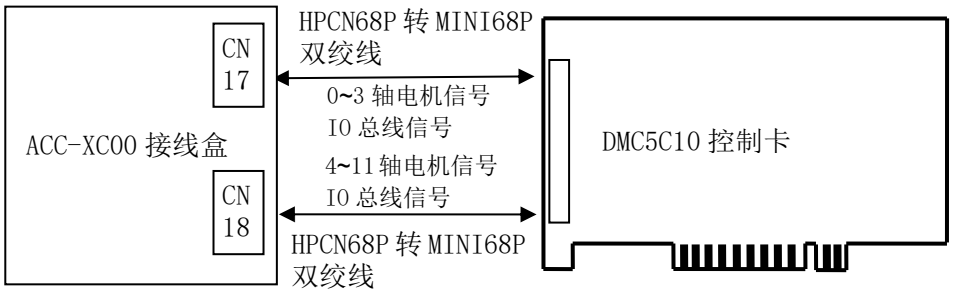


图3.3 DMC5C10与配件连接示意图

3.2.2 DMC5810 控制卡与配件的连接

DMC5810 控制卡有一个必配的接线盒 ACC3800，,连接示意图如图 3.4 所示，DMC5810-PCIe 控制卡有一个必配的接线盒 ACC2-3800，连接方式与 PCI 一致。

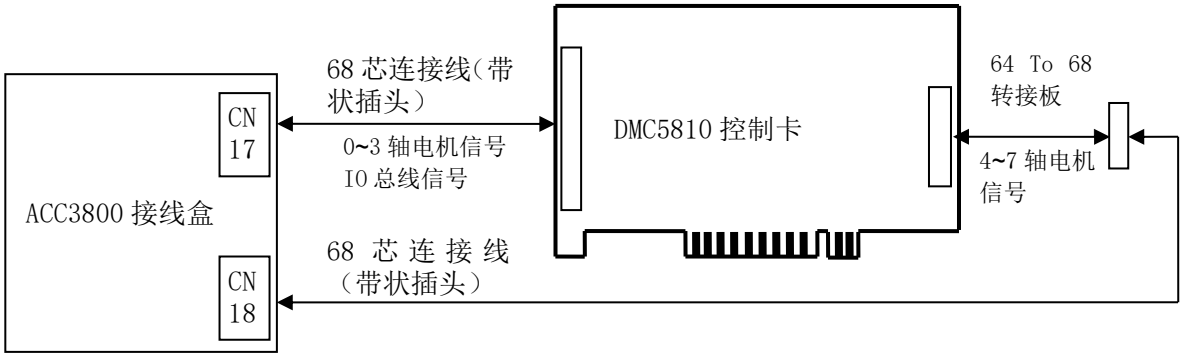


图3.4 DMC5810与配件连接示意图

3.2.3 DMC5610 卡与配件的连接

DMC5610 卡有一个必配的接线盒 ACC3600，连接示意图如图 3.5 所示，DMC5610-PCIe 控制卡有一个必配的接线盒 ACC2-3600，连接方式与 PCI 一致。

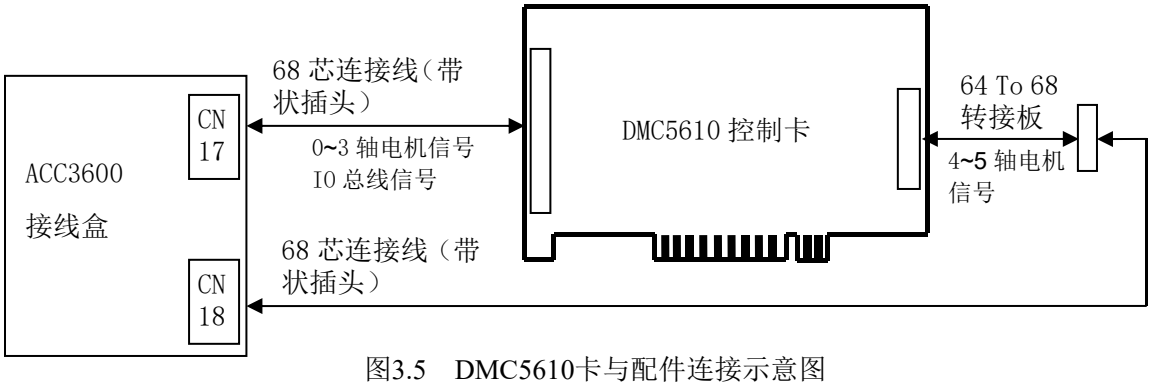


图3.5　DMC5610卡与配件连接示意图

3.2.4 DMC5410A 卡与配件的连接

DMC5410A 卡有一个必配的接线盒 ACC-X400B，连接示意图如图 3.6 所示，DMC5410-PCIe 控制卡有一个必配的接线盒 ACC2-X400B，连接方式与 PCI 一致。

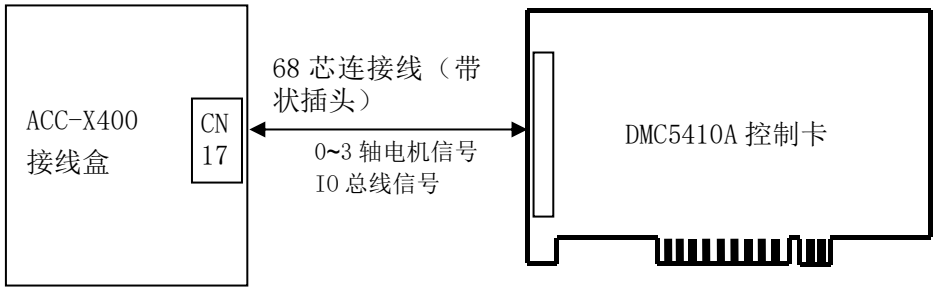


图3.6　DMC5410A卡与配件连接示意图

3.3　电机控制信号接口电路

DMC5X10 系列卡提供了两种脉冲量输出模式，一种是脉冲+方向信号模式，另一种是正/负脉冲模式，如图 3.7 和图 3.8 所示。默认情况下，控制卡输出脉冲+方向信号模式。用户可以通过系统配置在这两种模式之间切换。

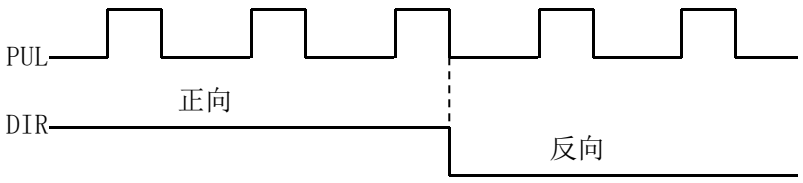


图3.7　单脉冲模式

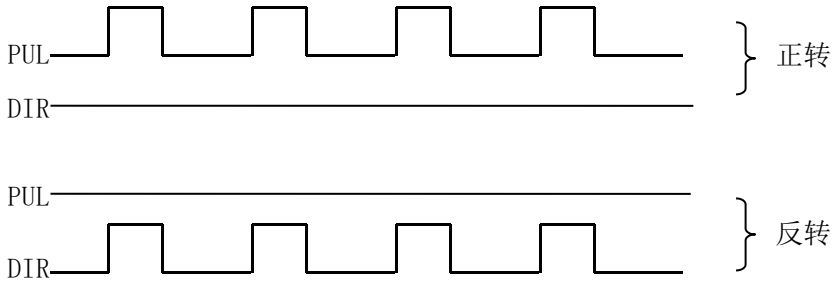


图3.8　双脉冲模式

3.3.1　DMC5810 卡的电机控制信号接口电路

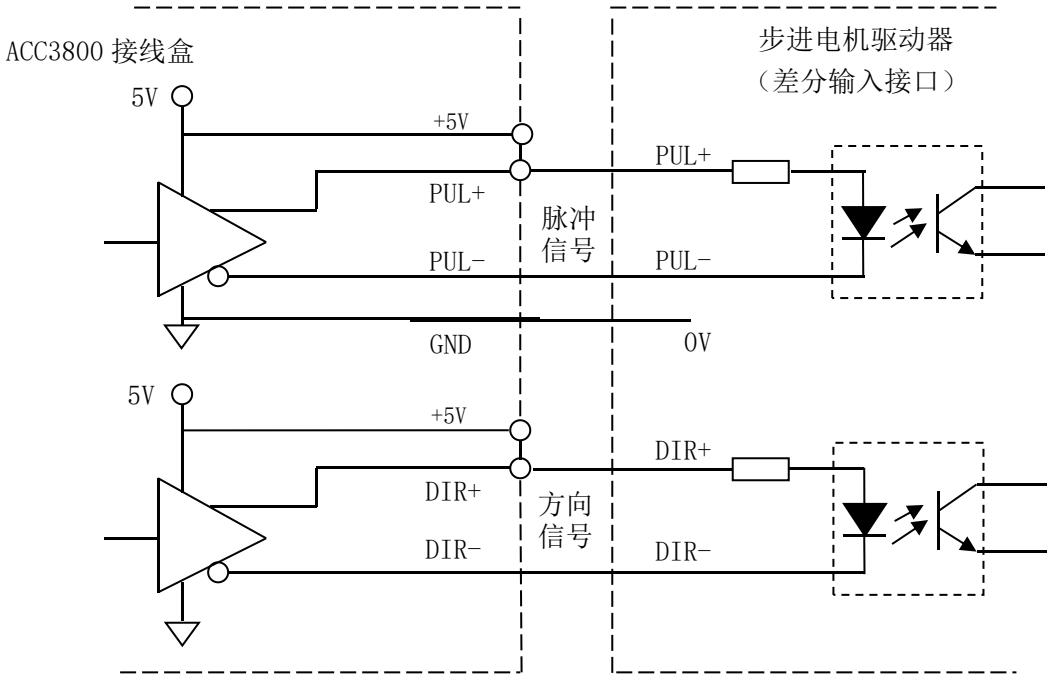


图3.9　ACC3800接线盒差分输出方式接线图

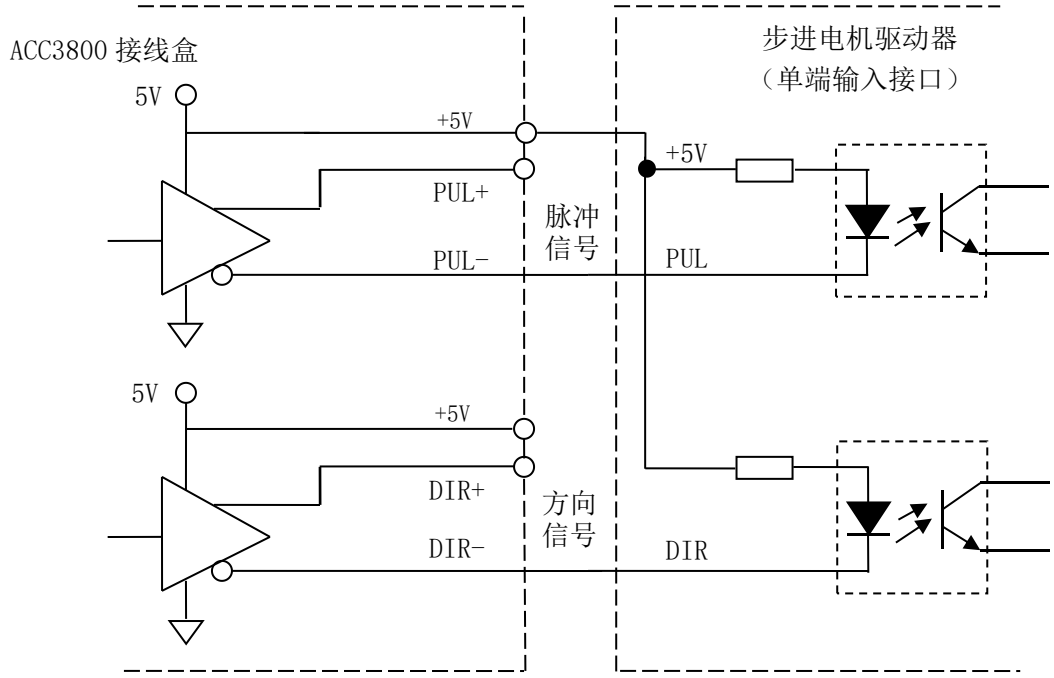


图3.10　ACC3800接线盒单端输出方式接线图

DMC5810 卡有 8 路电机控制信号接口，通过 ACC3800 接线盒与外部电机驱动器相连。

接线盒 ACC3800 上的 CN1~CN8 轴控制端口上有提供+5V 的电源信号。当电机驱动器为单端输入接口时，可使用+5V 和 PUL-端口输出脉冲信号，使用+5V 和 DIR-端口输出方向信号；当电机驱动器为差分输入接口时，使用 PUL+和 PUL-输出脉冲信号，使用 DIR+和 DIR-输出方向信号。如图 3.9、3.10 所示。

3.3.2　DMC5C10/DMC5610/5410A 卡的电机控制信号接口电路

DMC5C10 卡有 12 路电机控制信号接口，通过 ACC-XC00 接线盒与外部驱动器相连；DMC5610 卡有 6 路电机控制信号接口，通过 ACC3600 接线盒与外部电机驱动器相连；DMC5410A 卡有 4 路电机控制信号接口，通过 ACC-X400B 接线盒与外部电机驱动器相连。

接线盒 ACC-XC00 上的 CN1~CN8 以及 CN25~CN28 轴控制端口上有提供+5V 的电源信号，接线盒 ACC3600 上的 CN1~CN6 轴控制端口上有提供+5V 的电源信号，接线盒 ACC-X400B 上的 CN1~CN4 轴控制端口上有提供+5V 的电源信号。当电机驱动器为单端输入接口时，可使用+5V 和 PUL-端口输出脉冲信号，使用+5V 和 DIR-端口输出方向信号；当电机驱动器为差分输入接口时，使用 PUL+和 PUL-输出脉冲信号，使用 DIR+和 DIR-输出方向信号。

接线盒 ACC-XC00/ACC3600/ACC-X400B 的电机控制信号接口电路与接线盒 ACC3800 的相同，关于其接口电路示意图可参考图 3.9~3.10。

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

DMC5X10 系列运动控制卡支持 2 种类型的编码器信号输入：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

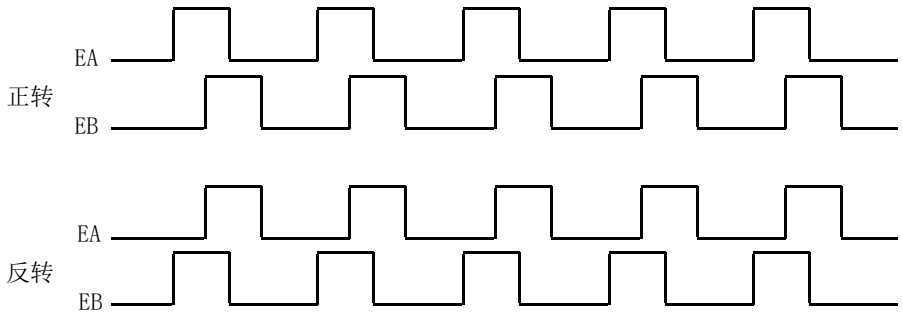


图3.11 A/B相正交信号

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 3.16 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，用户还可选用 4 倍、2 倍频计数模式对 EA，EB 信号进行计数设置。

1 倍频计数：若只用 EA 信号的上升沿触发计数器，一个脉冲周期就计数一次。

2 倍频计数：EA、EB 信号的上升沿都参与触发计数器，故将一个脉冲周期就分为两份。所以，计数精度提高了 2 倍。

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；若设置为 2 倍频计数，编码器计数器的值为 5000；若设置为 1 倍频计数，编码器计数器的值为 2500。

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 3.12 所示。

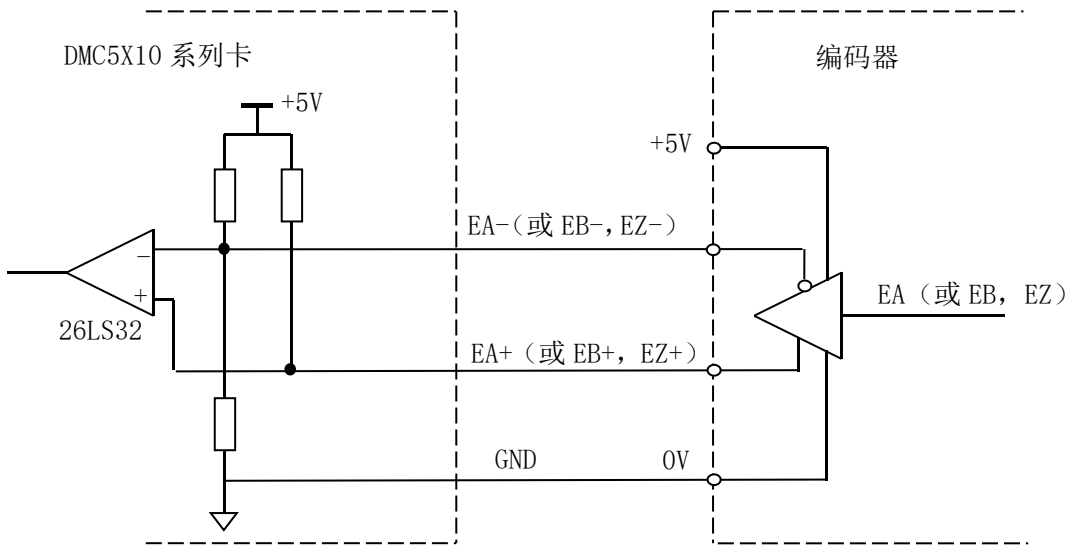


图 3.12 差分输出编码器接线原理图

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 3.13 所示。

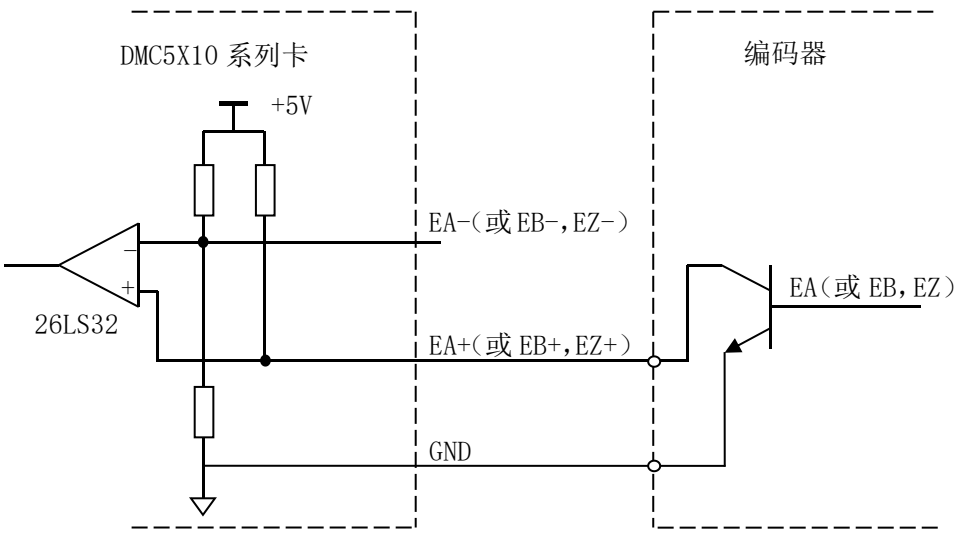


图 3.13 集电极开路输出的编码器接线原理图

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）DMC5X10 系列卡仅最后 2 轴的编码器口同时支持单端和差分接法，其它轴的编码器口只支持差分接法。如 DMC5C10 卡 1~6 轴只支持差分接法，7~8 轴同时支持单端和差分接法，9~12 轴无编码器口。

3.4.2 手摇脉冲发生器输入接口

DMC5X10 系列卡为每个轴提供了手摇脉冲发生器（简称：手轮）脉冲信号 PA、PB 的输入接口，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。其接口原理如图 3.14 所示。

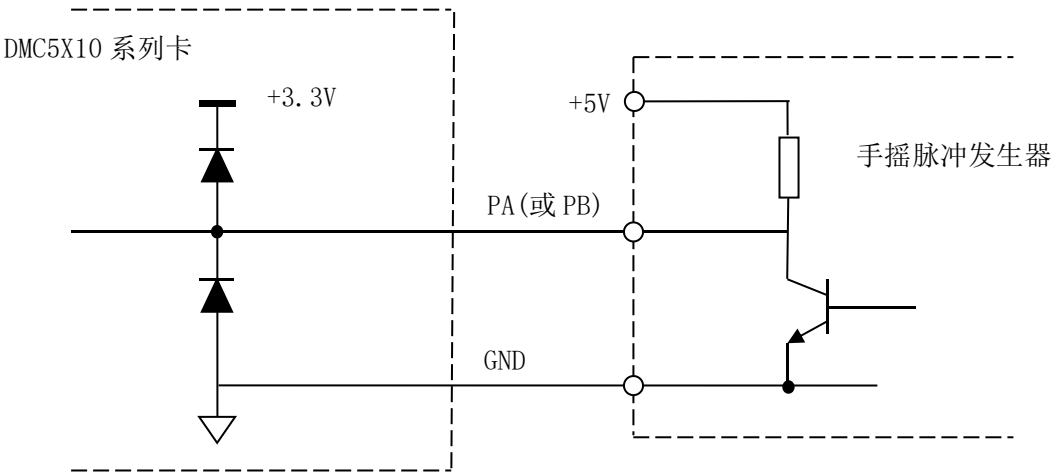


图 3.14 手轮脉冲输入接口原理图

3.5 专用 I/O 接口电路

3.5.1 原点信号输入接口

DMC5X10 系列卡为每个轴都提供了 1 个原点位置传感器信号的输入端口 ORG，其接口电路如图 3.15 所示。

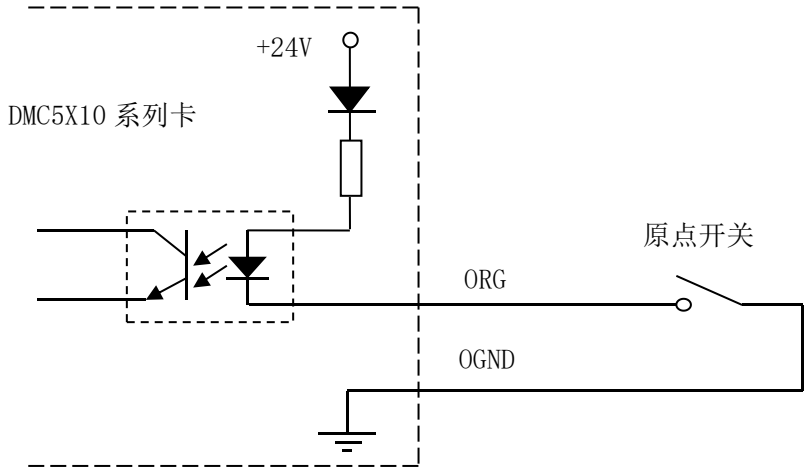


图 3.15　原点信号接口电路原理图

原点信号由 ORG 输入端口接入运动控制卡后，经过光耦后进入微处理器。光电隔离电路可以有效地将外部干扰信号隔离在控制卡之外，有效地提高了系统的可靠性。

3.5.2　限位信号输入接口

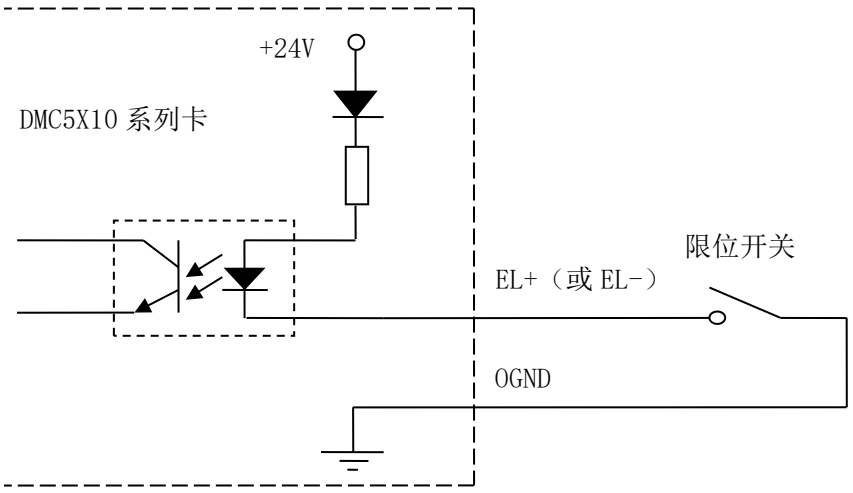


图 3.16　限位信号接口电路原理图

DMC5X10 系列卡为每个轴提供了 2 个机械限位信号 EL+ 和 EL-。EL+为正向限位信号，EL-为反向限位信号。当运动平台触发限位开关时，EL+或 EL-即有效，控制卡将禁止运动平台继续向前运动。其接口电路如图 3.16 所示。

注意：用户需根据使用的限位开关类型来设置限位开关的有效工作电平。当使用常开型限位开关时，应通过软件选择 EL+、EL-信号为低电平有效；当使用常闭型限位开关时，应选择 EL+、EL-信号为高电平有效。

3.5.3 伺服电机驱动器控制信号接口

DMC5X10 系列卡为每一轴(DMC5C10 后四轴除外）均提供了伺服电机驱动器专用信号接口，其中信号 RDY、ALM 和 INP 用于监控伺服电机状态，信号 SEVON 和 ERC 用于设置伺服电机状态。

3.5.3.1 驱动器使能信号

当伺服电机驱动器的 SEVON 信号为无效状态时，伺服电机驱动器不工作，电机处于自由状态；当 SEVON 信号有效时，伺服电机驱动器进入工作状态，电机锁紧。其接口电路原理图如图 3.17 所示。

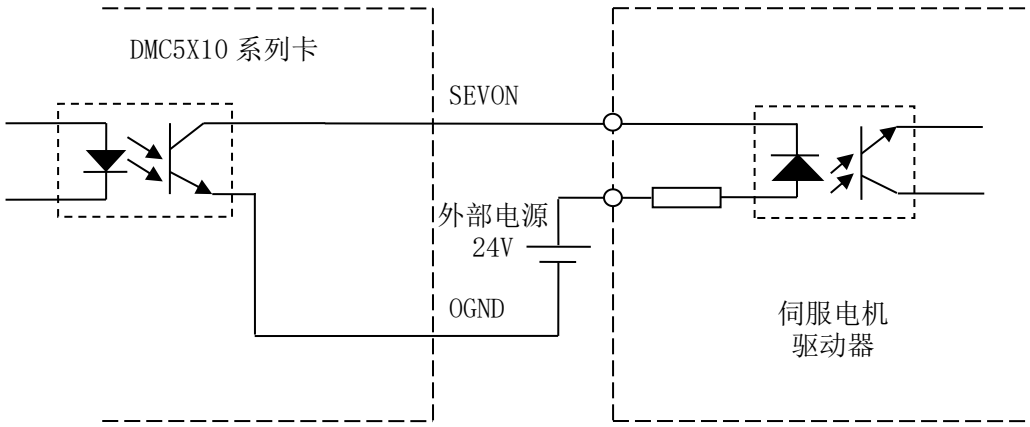


图 3.17 SEVON 信号接口原理图

3.5.3.2 驱动器准备好信号

当伺服电机驱动器处于准备好状态，RDY 信号就会自动将该信号置为有效。此时，运动控制卡可以向伺服电机驱动器发出运动命令。其接口电路原理图如图 3.18 所示。

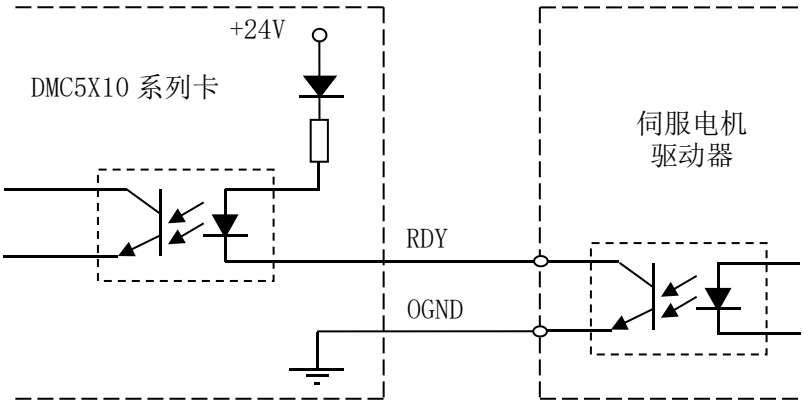


图 3.18 RDY 信号接口原理图

3.5.3.3 驱动器报警信号

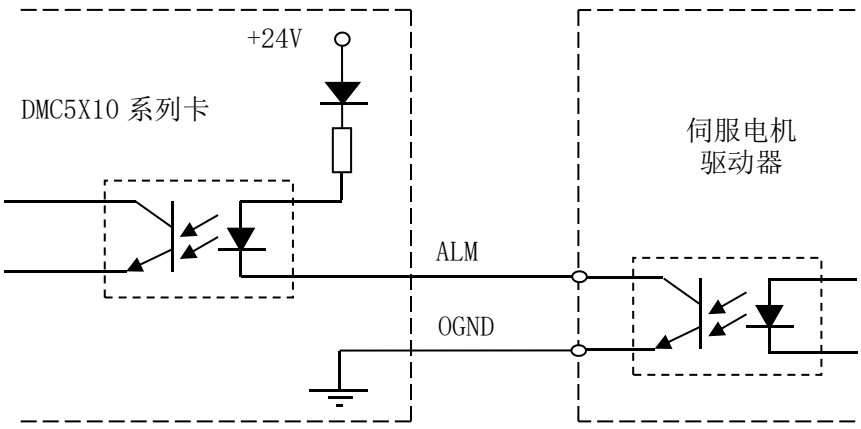


图 3.19　ALM 信号接口原理图

ALM 信号是伺服电机驱动器发出的报警信号。当运动控制卡接收到 ALM 信号后，将立即中止发送运动指令脉冲，或先减速再停止发送脉冲。其接口电路原理图如图 3.19 所示。

3.5.3.4　驱动器位置到达信号

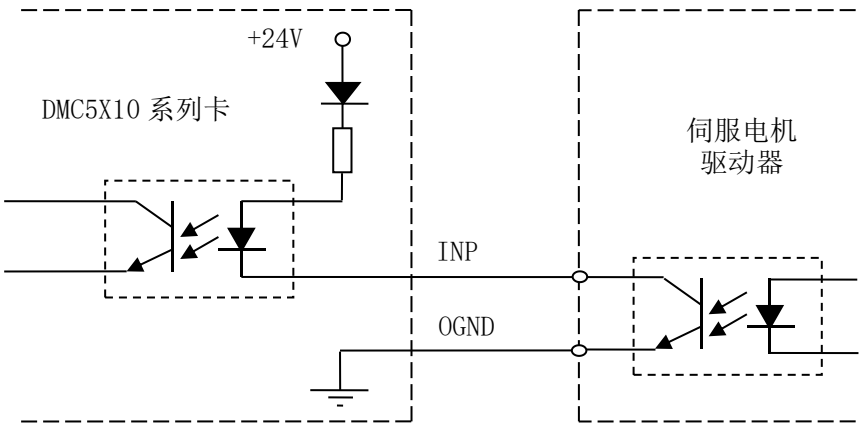


图 3.20　INP 信号接线原理图

雷赛 DMC5X10 系列卡均为每一轴(DMC5C10 后四轴除外）提供了用于监控伺服电机定位结果的 INP 信号接口。典型接口及接线原理如图 3.20。

注意：当使能了 INP 信号功能时，只有在 INP 信号为有效状态下，对应的轴才能进行运动，否则此时检测轴的状态是正在运行的（对轴运动作限制）。

3.5.3.5　驱动器误差清除信号

雷赛 DMC5X10 系列卡均为每一轴(DMC5C10 后四轴除外）提供了用于清零伺服驱动位置偏差计数器的 ERC 信号输出接口。典型接口及接线原理如图 3.21。

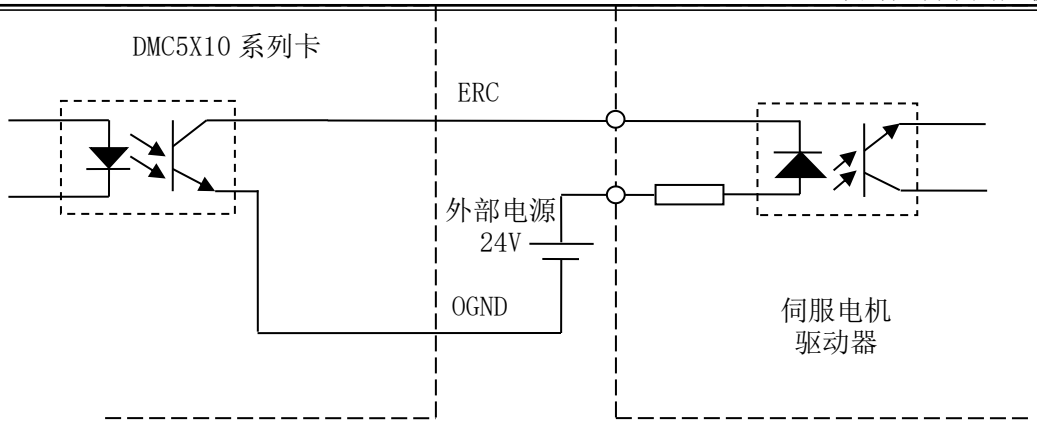


图 3.21　ERC 信号接线原理图

3.5.4　高速位置锁存输入信号接口

DMC5C10/DMC5810/5610/5410A 卡每四轴(DMC5C10 后四轴除外）共用一个位置锁存输入信号 LTC，信号 LTC0 锁存 0~3 轴的当前编码器位置或指令位置，信号 LTC1 锁存 4~7 轴的当前编码器位置或指令位置。其接口电路原理如图 3.22 所示。

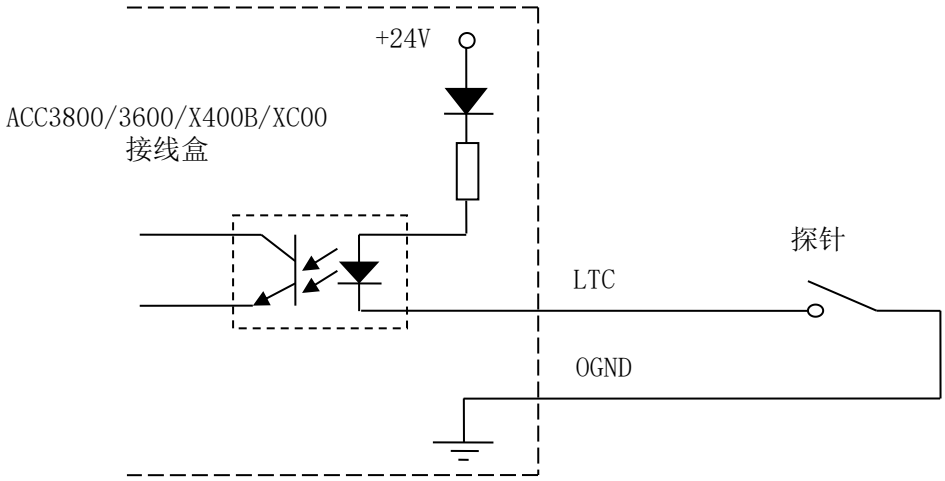


图 3.22　ACC3800/3600/X400B/XC00 位置锁存输入信号接口原理图

3.5.5　高速位置比较输出信号接口

DMC5X10 系列卡共有四个高速位置比较器，每个高速位置比较器均配有 1 个硬件位置比较输出接口，其中 DMC5410A/5600/5800 最后 2 路输出口为高速口，DMC5C10 最后 4 路输出口为高速口。通过软件使能后，可分别设置比较模式以及关联电机轴号，当该轴的指令寄存器内的数值或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。

ACC-XC00/ACC3800/3600/ACC-X400B 接线盒的 CMP 接口原理图如图 3.23 所示。

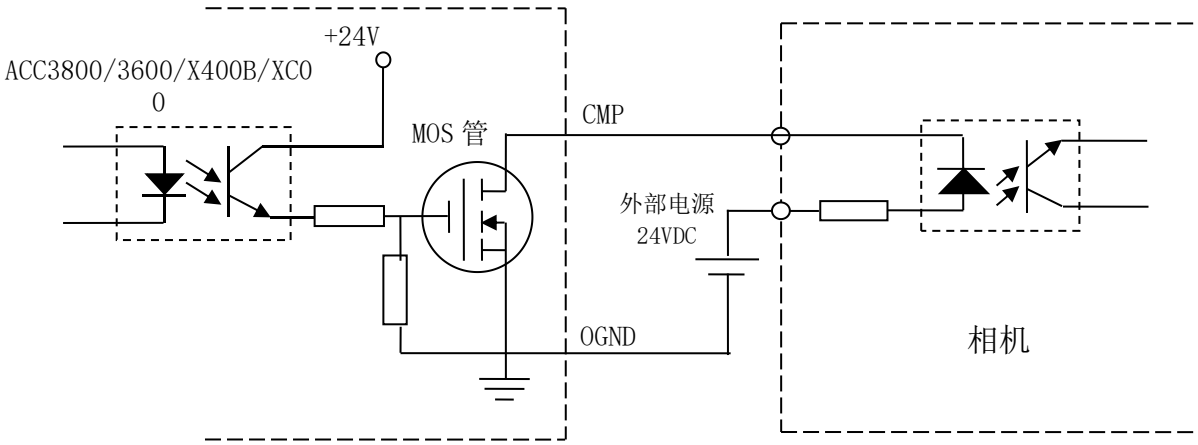


图 3.23　ACC3800/3600/X400B/XC00 高速位置比较输出信号接口原理图

3.6　通用 I/O 接口电路

DMC5X10 系列卡除了提供专用的数字 I/O 接口外还提供了大量的通用数字 I/O 接口。

3.6.1　通用数字输入信号接口

DMC5C10/DMC5810/DMC5610 卡有 16 路通用数字输入信号（其中 IN14、IN15 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 3.24 所示。

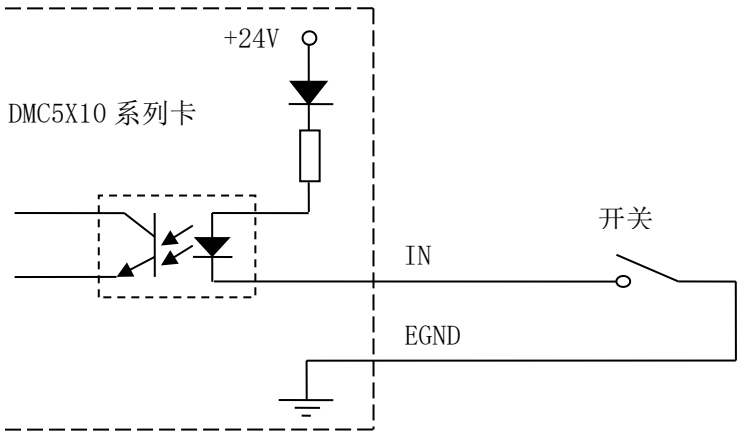


图 3.24　通用输入信号接口原理图

3.6.2　通用数字输出信号接口

DMC5C10 卡通过与 ACC-XC00 接线盒相连，有 16 路通用数字输出信号（其中 OUT12~15 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。

DMC5810 卡通过与 ACC3800 接线盒相连，有 16 路通用数字输出信号（其中 OUT12~15 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。

DMC5610 卡通过与 ACC3600 接线盒相连，有 16 路通用数字输出信号（其中 OUT12~15 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。

DMC5410A 卡通过与 ACC3800 接线盒相连，有 14 路通用数字输出信号（其中 OUT14~15 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 3 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 3.25 所示。

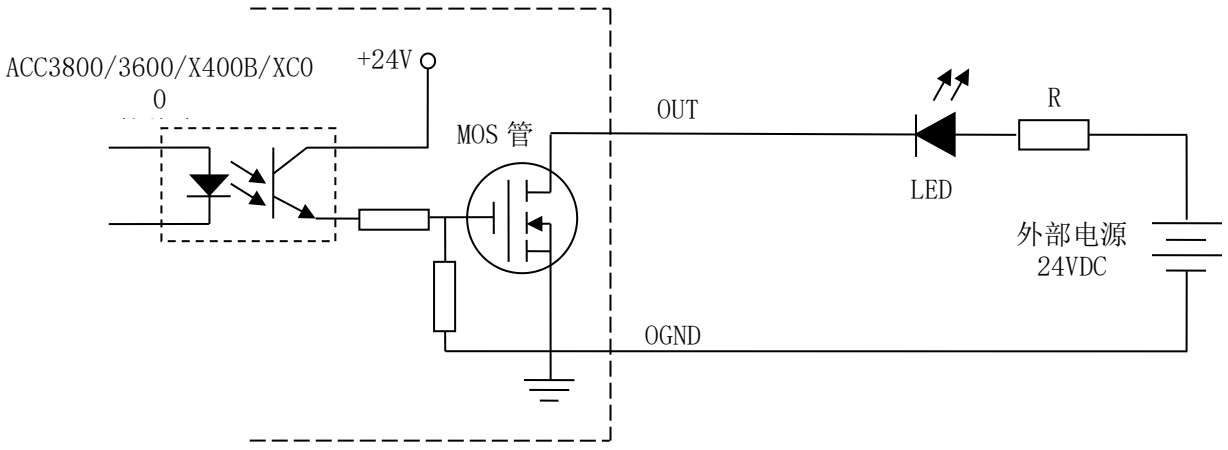


图 3.25　ACC3800/3600/X400B/XC00 输出口接发光二极管

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 3.26 所示。

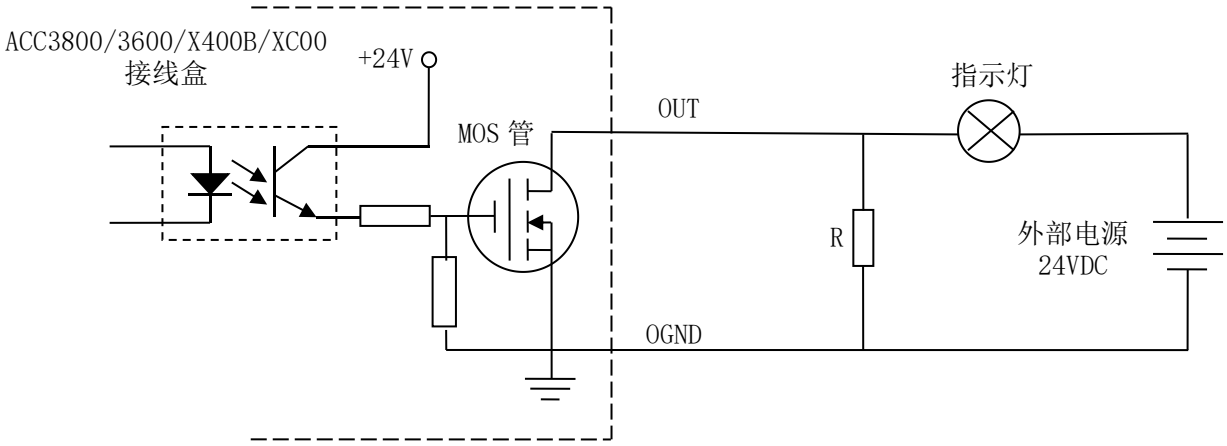


图 3.26　ACC3800/3600/X400B/XC00 灯丝型指示灯接线图

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。

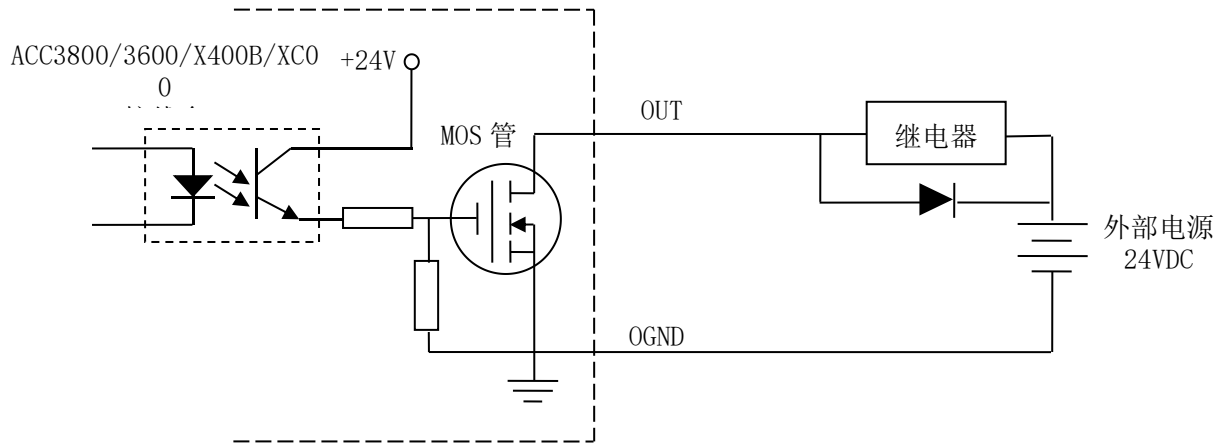


图 3.27　ACC3800/3600/X400B/XC00 接小型继电器的接线图

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。

3.7　CAN 扩展模块接口电路

CAN-IO 扩展模块是 DMC5X10 系列运动控制卡的配套产品，扩展 DMC5X10 系列卡的 IO 和 ADDA 端口。通过菊花链的连接方式可以将多个 CAN 扩展模块挂在同一块运动控制卡下面，最多支持连接 8 个 CAN 扩展模块。

CAN 扩展模块各接口具体定义详见各个 CAN 扩展模块手册。

3.7.1 CAN-IO 扩展模块的连接与设置

使用 CAN-IO 扩展模块时，必须先将各个扩展模块与 ACC-XC00/ACC3800/3600/ACC-X400B 等接线盒连接，然后设置好各个扩展模块的节点号及终端电阻。

3.7.1.1 CAN 扩展模块的连接

CAN 扩展模块与 ACC3800/3600/ACC-X400B 等接线盒是通过菊花链的形式连接，如图 3.28 所示。

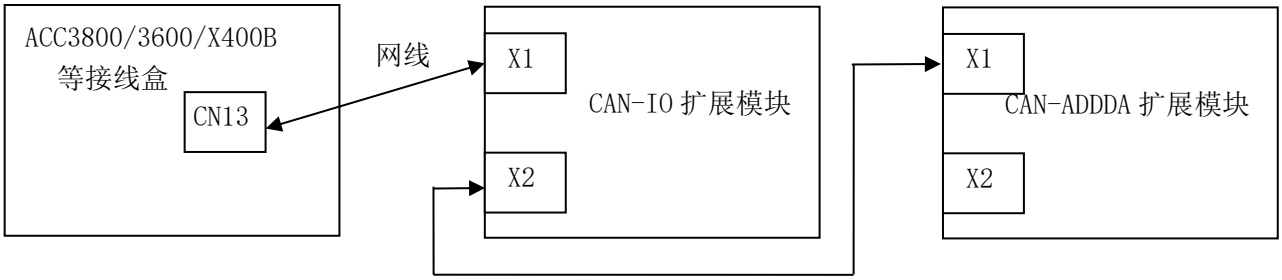


图 3.28 CAN 扩展模块与 ACC3800/3600/ACC-X400B 等接线盒的连接示意图

3.7.1.2 终端电阻及模块启动的设置

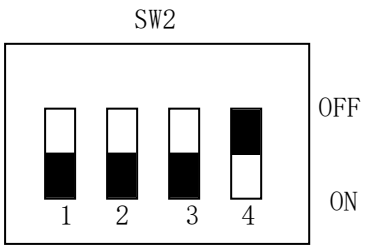


图 3.29 拔码开关示意图

表 3.1 S1 拔码开关的设置

拔码开关号	开关状态	定义
S1-1	ON	OUT~OUT7 初始电平为低
	OFF	OUT~OUT7 初始电平为高
S1-2	ON	OUT8~OUT15 初始电平为低
	OFF	OUT8~OUT15 初始电平为高
S1-3	ON	保留
	OFF	保留
S1-4	ON	终端电阻连接
	OFF	终端电阻断开

当使用多个 CAN-IO 扩展模块时，菊花链上的最后一个 CAN-IO 扩展模块的终端电阻建议设置为连接状态。如果只使用一个 CAN-IO 扩展模块，那么此扩展模块的终端电阻必须设置为连接状态。并且，需要运行的 CAN-IO 扩展模块必须设置为 RUN 状态。

这两个功能的设置都是通过拔码开关 SW2 实现的，具体方法如表 3.1 所示。

3.7.1.3 CAN-IO 扩展模块节点号的设置

CAN-IO 扩展模块的节点号是通过 SW1 设置的，波特率是通过 SW0 来设置的，且须在模块上电之前设置。SW0、SW1、SW2 重新设置后，模块需重新上电才有效。

当使用多个 CAN-IO 扩展模块时，菊花链上的第一个 CAN-IO 扩展模块的 CAN 节点号必须设置为 1，第二个扩展模块的 CAN 节点号必须设置为 2，第三个扩展模块的 CAN 节点号必须设置为 3，以此类推。

3.7.2 通用数字输入信号接口

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 3.30 所示。

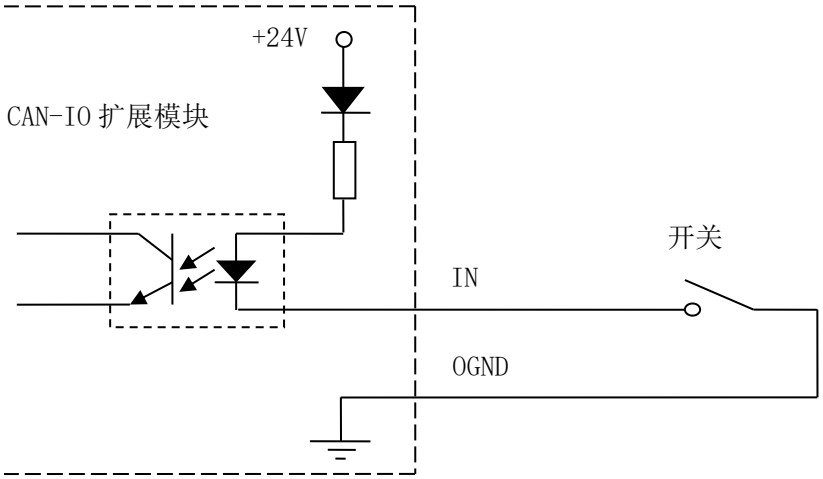


图 3.30 通用输入信号接口原理图

3.7.2 通用数字输出信号接口

所有输出接口均由 MOS 管驱动，单路输出电流可达 300mA，可用于对继电器、电磁阀、信号灯或其它设备的控制。其接口电路都加有光电隔离元件，可以有效隔离外部电路的干扰，提高了系统的可靠性。输出电路采用 OD 设计，上电默认 MOS 管关断。

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。

下面给出了通用数字输出信号接口控制 3 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 3.31 所示。

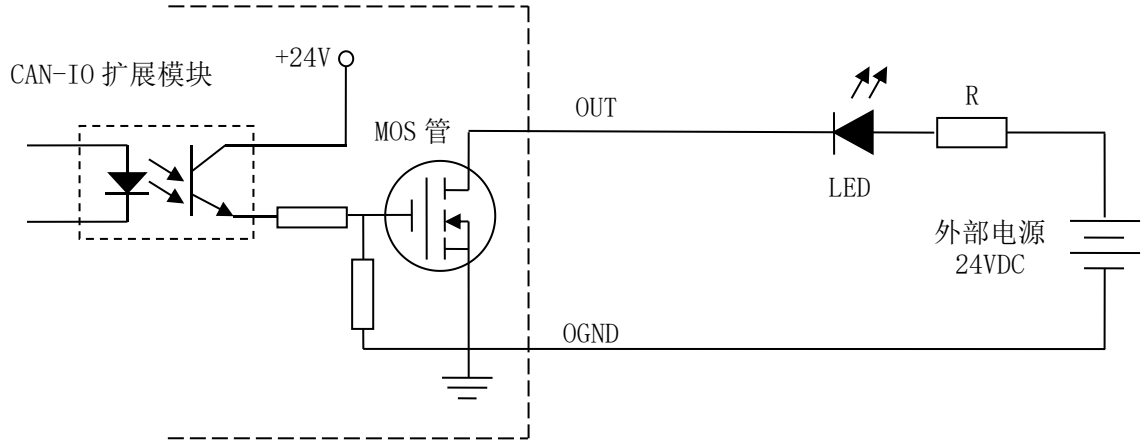


图 3.31　通用输出口接发光二极管

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 3.32 所示。

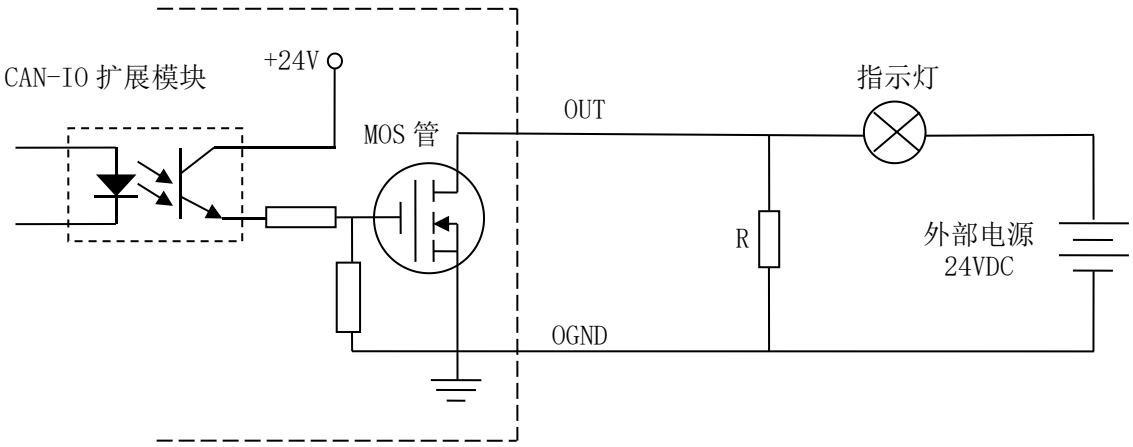


图 3.32　通用输出口接灯丝型指示灯

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 3.33 所示。

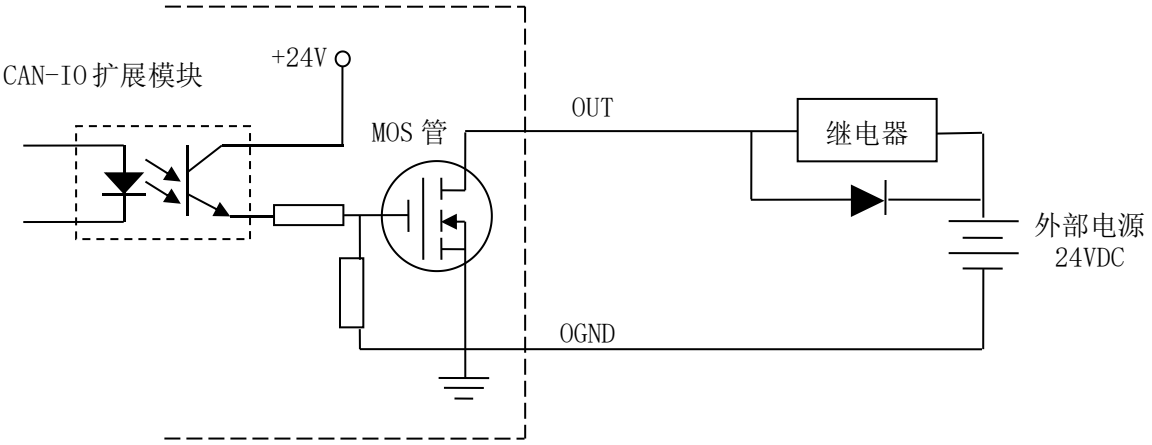


图 3.33　通用输出口接小型继电器

第 4 章　硬件及驱动程序的安装

4.1　硬件安装步骤

4.1.1　DMC5810/DMC5610/5410A 硬件设置

如图 4.1 所示，DMC5810 运动控制卡上有 3 组拨码开关 S2、S4 和 S5，用于设置 DMC5810 卡的工作方式和参数。

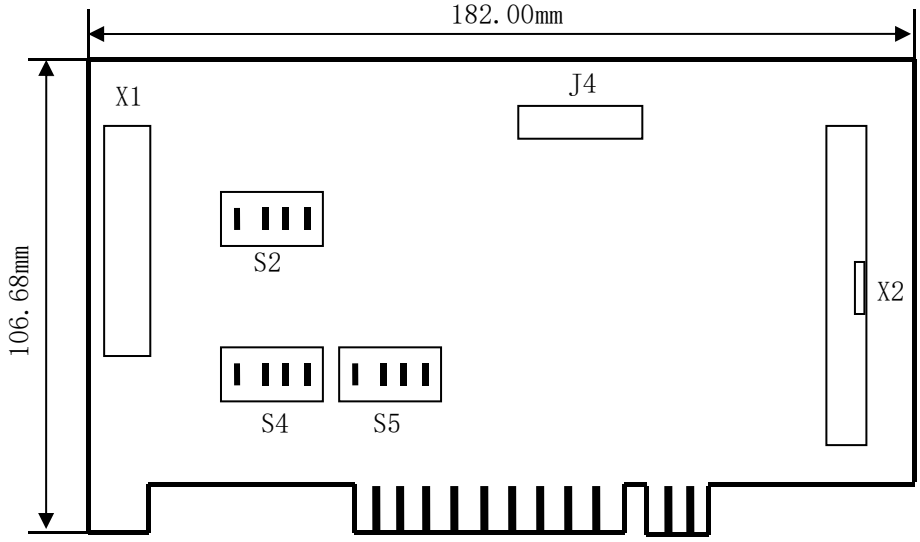


图 4.1　DMC5810 板卡跳线开关、拨码开关的位置

1.电机脉冲信号输出方式的选择

DMC5810 控制卡主板上没有跳线开关用于设置电机脉冲信号输出方式；但是在接线盒 ACC3800 上的 CN1~CN8 轴控制端口上提供+5V 电源。当使用+5V 和 PUL-端口时，则指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则指令脉冲信号输出方式为差分输出。参见图 3.9~3.10。

2.拨码开关设置

如图 4.6 所示，DMC5810 卡上有三个 4 位拨码开关 S2、S4 和 S5，其中 S2、S4 暂时保留，无定义。S5 用于卡号设置。S5 具体使用方法如表 4.2 所示。

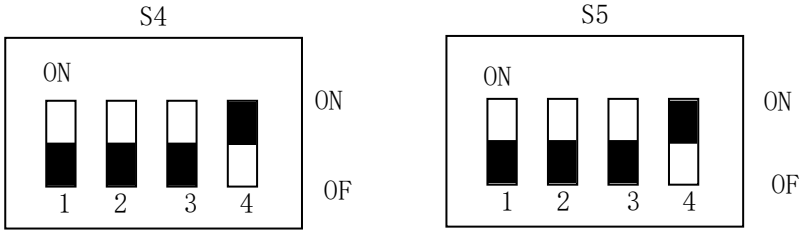


图 4.2　拨码开关示意图

表 4.1　拨码开关 S5 设置运动控制卡号

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 OFF，即默认设置为 0 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。

4.1.2　DMC5C10 硬件设置

DMC5C10 卡硬件设置与 DMC5810 卡基本相同，关于 DMC5C10 卡的硬件设置可参考章节 4.1.1　DMC5810 硬件设置。

4.1.3　DMC5810-PCIe/DMC5610-PCIe/5410A-PCIe 硬件设置

DMC5810-PCIe/DMC5610-PCIe/5410A-PCIe 硬件设置与 DMC5C10-PCIe 卡基本相同，关于 DMC5810-PCIe/DMC5610-PCIe/5410A-PCIe 卡的硬件设置可参考章节 4.1.4 DMC5C10-PCIe 硬件设置。

4.1.4　DMC5C10-PCIe 硬件设置

如图 4.3 所示，DMC5C10-PCIe 运动控制卡上有 3 组拨码开关 S2、S4 和 S5，用于设置 DMC5C10-PCIe 卡的工作方式和参数。

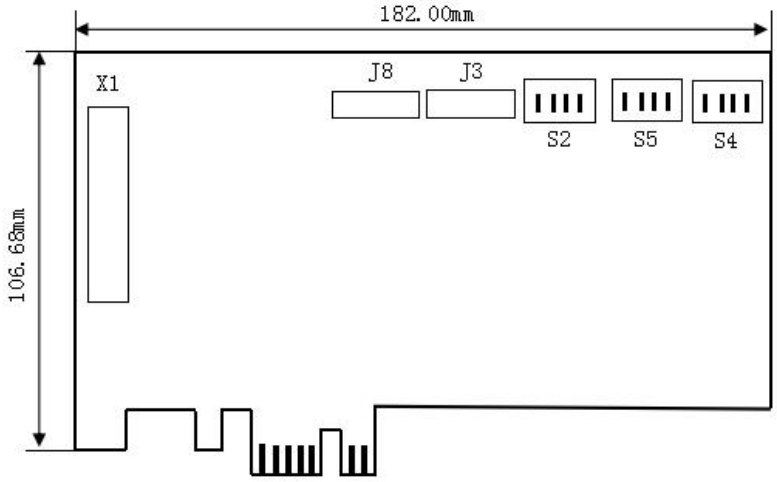


图 4.3　DMC5C10-PCIe 板卡跳线开关、拨码开关的位置

1.电机脉冲信号输出方式的选择

DMC5C10-PCIe 控制卡主板上没有跳线开关用于设置电机脉冲信号输出方式;但是在接线盒 ACC2-XC00 上的 CN1~CN8 轴控制端口上提供+5V 电源。当使用+5V 和 PUL-端口时，则指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则指令脉冲信号输出方式为差分输出。参见图 3.9~3.10。

2.拨码开关设置

如图 4.3 所示，DMC5C10-PCIe 卡上有三个 4 位拨码开关 S2、S4 和 S5，其中 S2、S4 暂时保留，无定义。S5 用于卡号设置。S5 具体使用方法如表 4.3 所示。

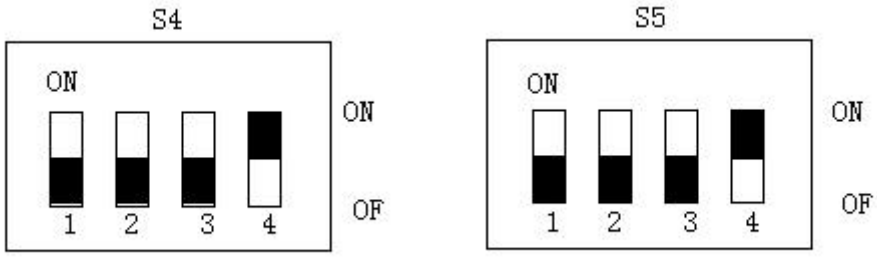


图 4.4　拨码开关示意图

表 4.3　拨码开关 S5 设置运动控制卡号

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

S5-3	S5-2	S5-1	控制卡号
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 OFF，即默认设置为 0 号卡。
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。

4.1.5 硬件安装

DMC5X10 系列卡硬件遵从 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 或 4.1.2 节硬件设置的说明，按照实际需求，完成跳线开关、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

1、双击驱动文件 DMCDriver_Setup.exe，如果出现如图 1 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。

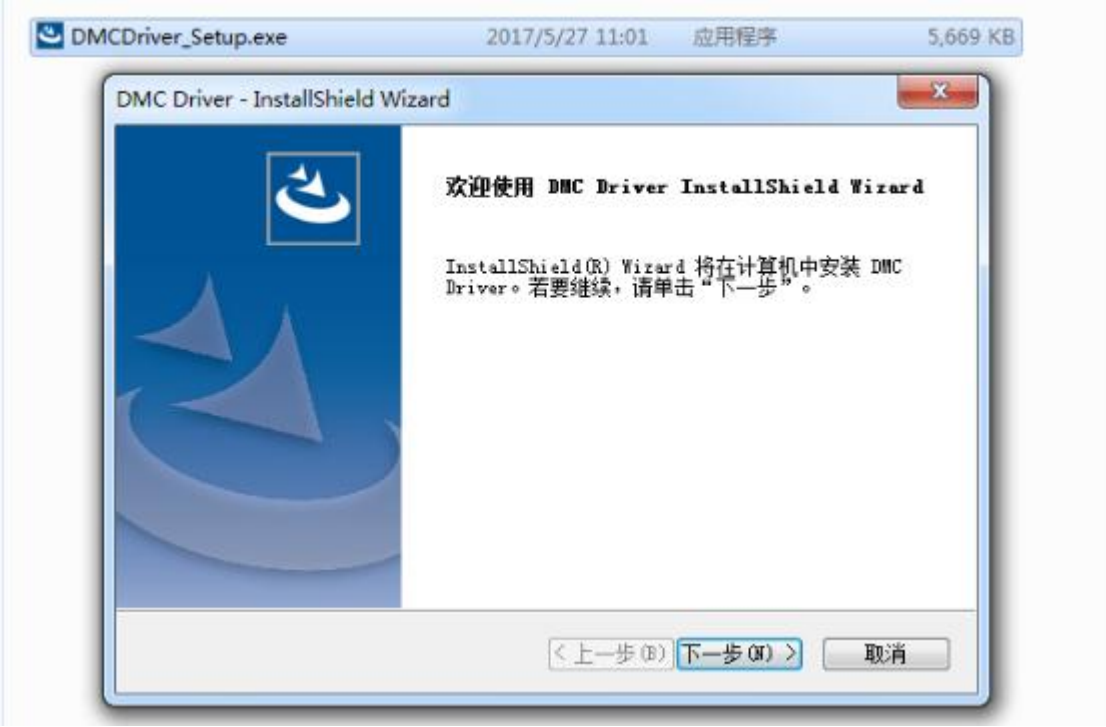


图 1 驱动程序开始安装

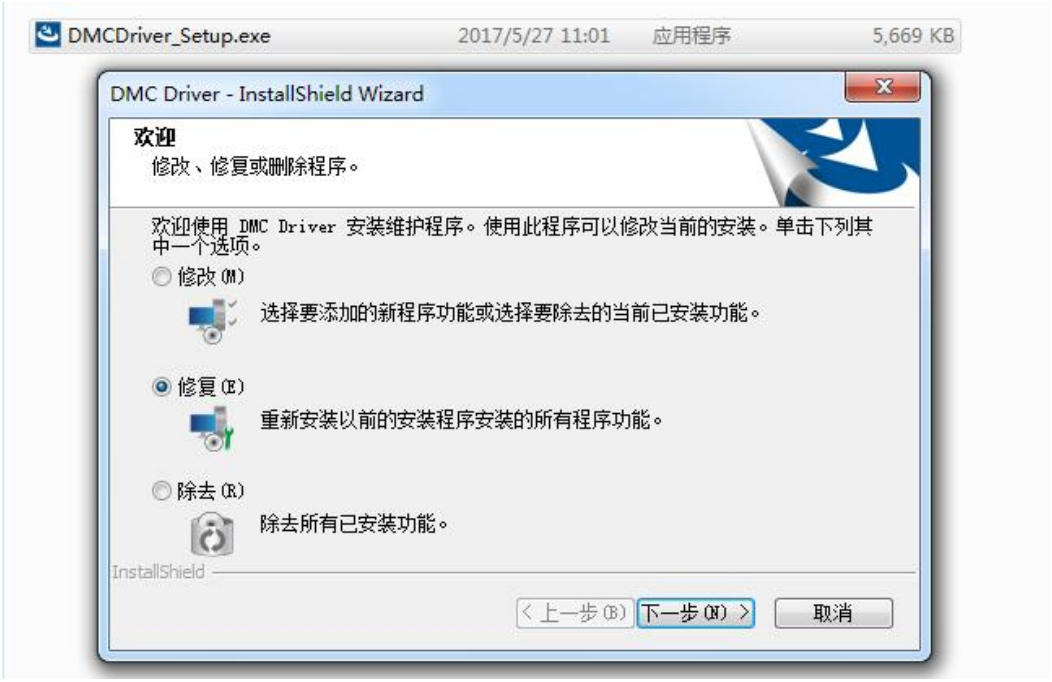


图 2 已安装驱动程序选项

2、如图 3 所示，输入用户名和公司名称，点击“下一步”继续安装。

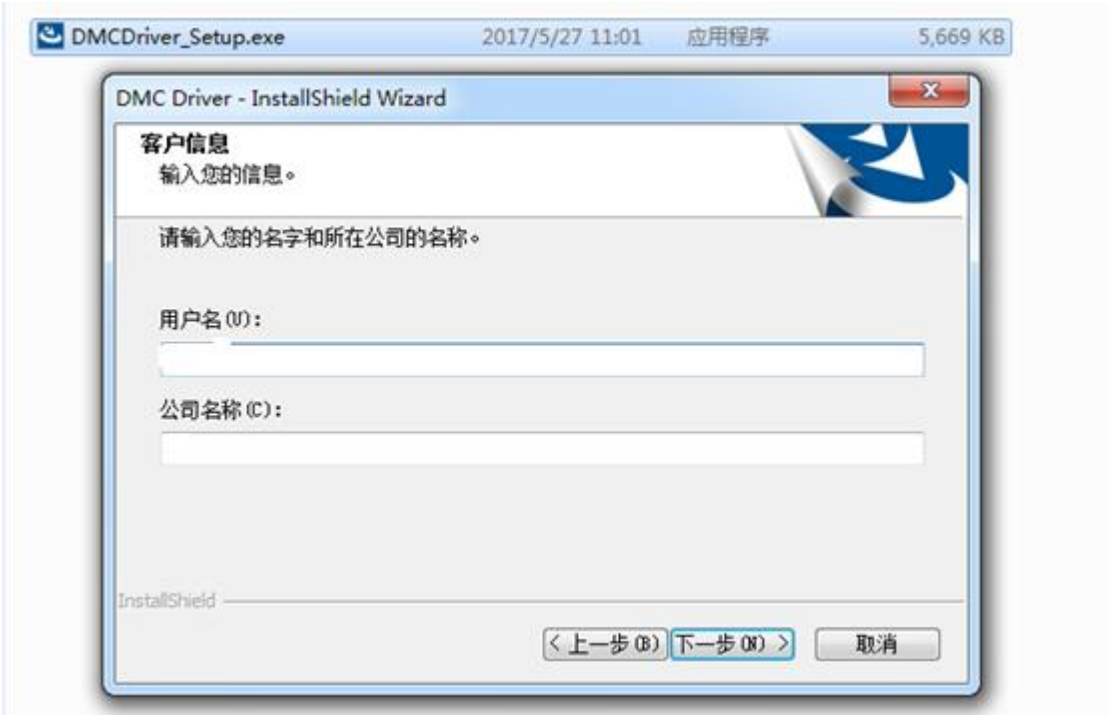


图 3 输入信息对话框

3、如图 4 所示，请选择“全部”，然后点击“下一步”继续安装。

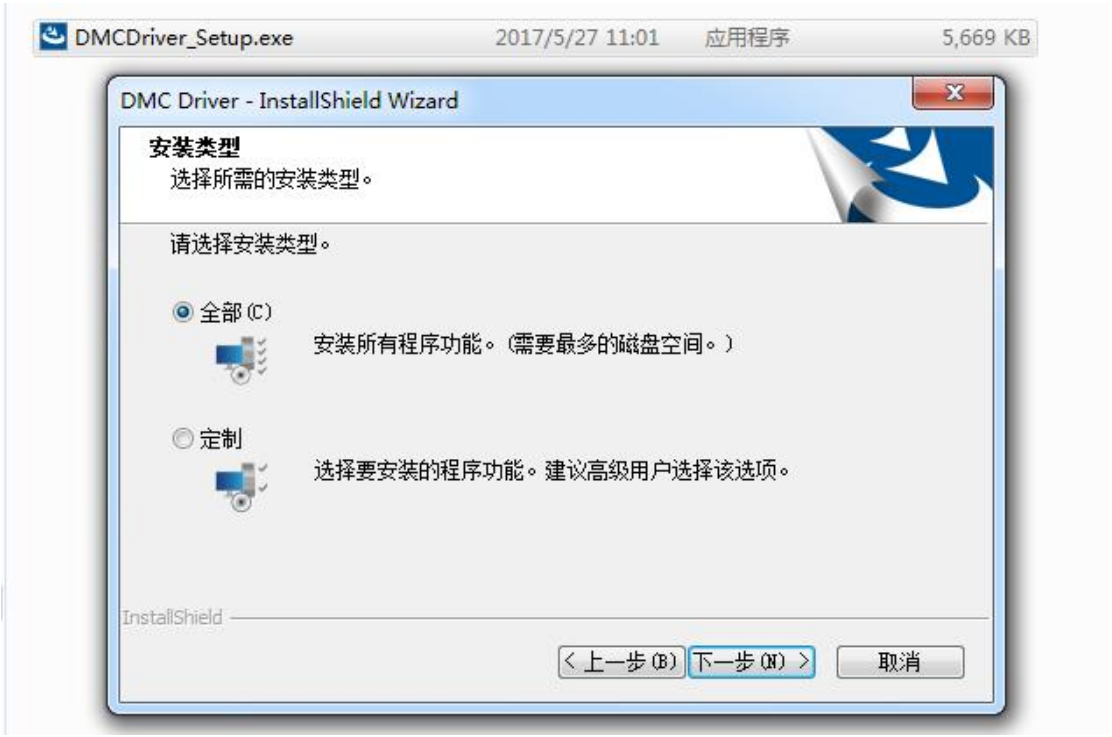


图 4 安装类型选择

4、如图 5 所示，点击“安装”按钮，开始安装，。

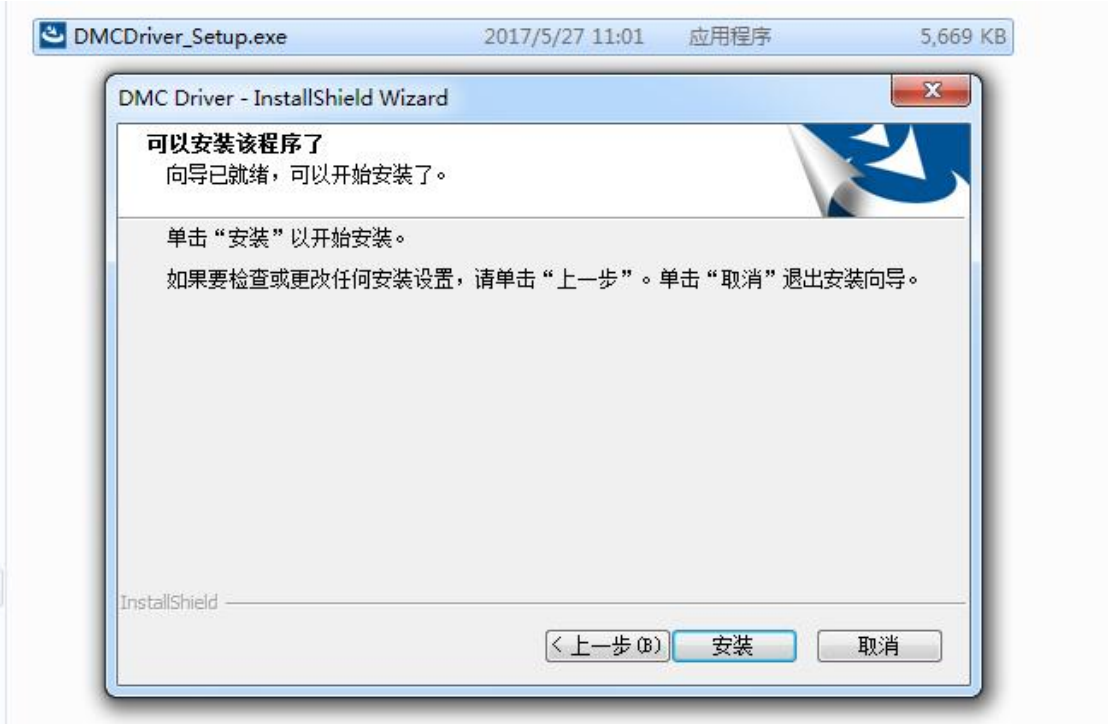


图 5 驱动程序开始安装

5、安装过程中会出现如图 6 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 6 Windows 安全提示

6、安装完成后，显示界面如图 7 所示，点击“完成”。

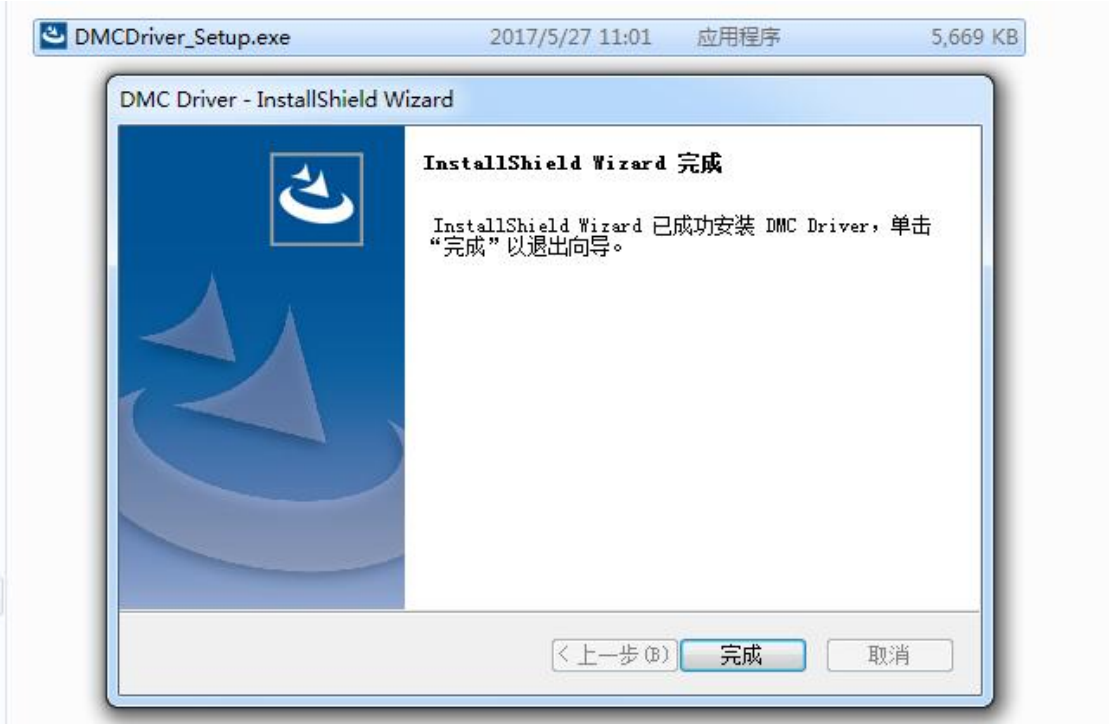


图 7 完成驱动程序的安装

7、打开设备管理器， 在“ Jungo” 选项下可以看到“ DMCx/NMCx” 和“ LeiSaiDriver” 注册信息。 至此， 控制卡就可以正常使用了， 如图 8 所示。

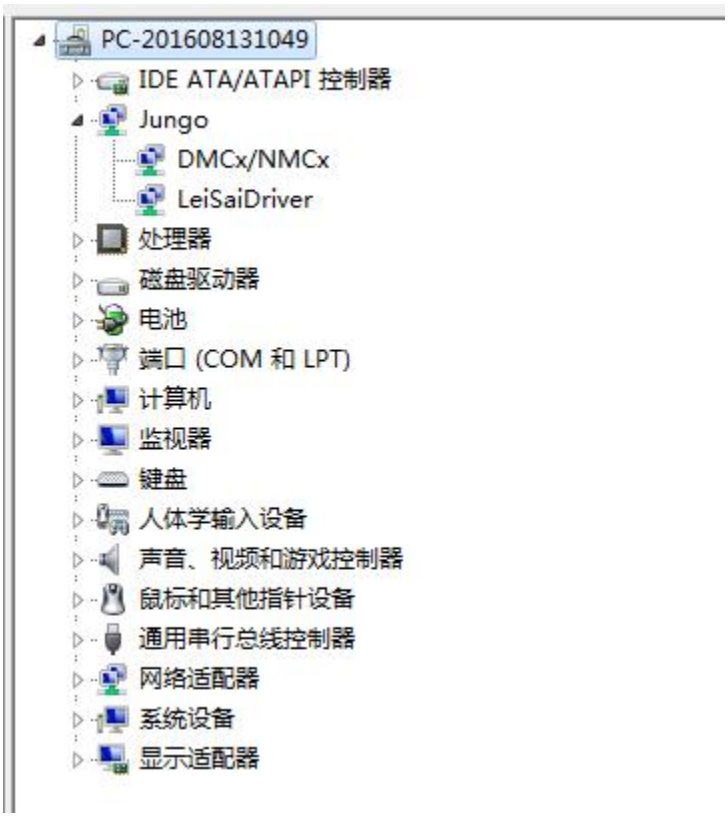


图 8 正确安装驱动程序后的设备信息

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 1 所示对话框，请选择“除去”，并点击“下一步”。

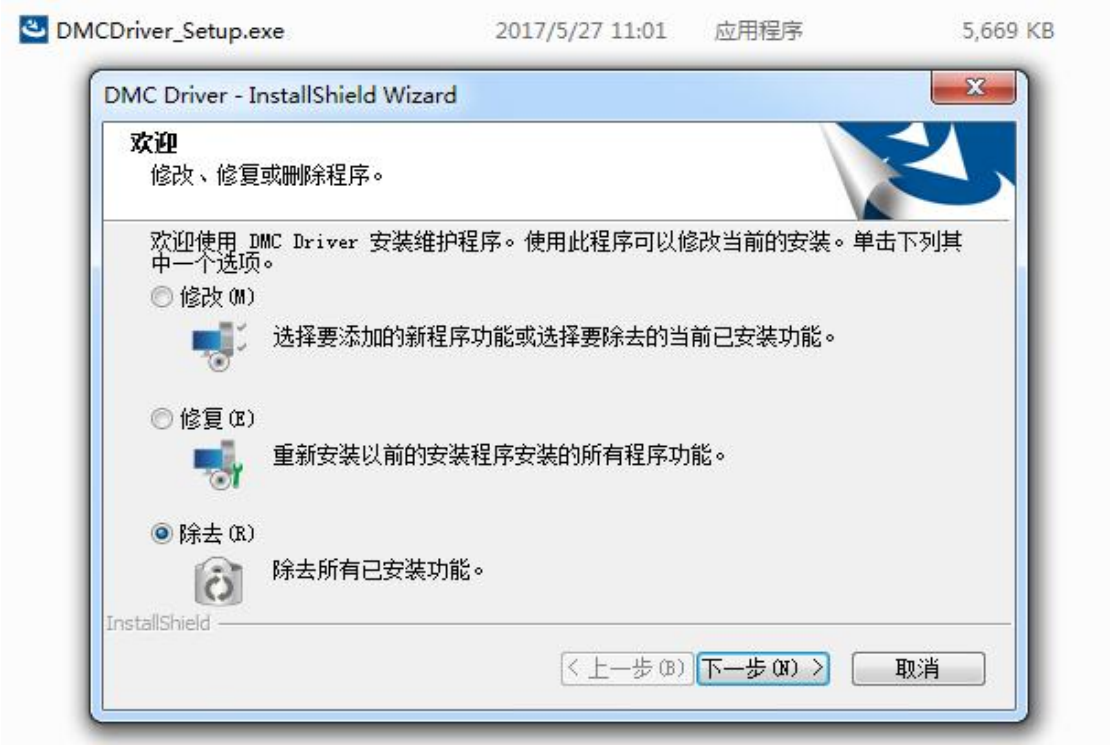


图 1 删除驱动选项

2、出现确认对话框，请选择“是”按钮。

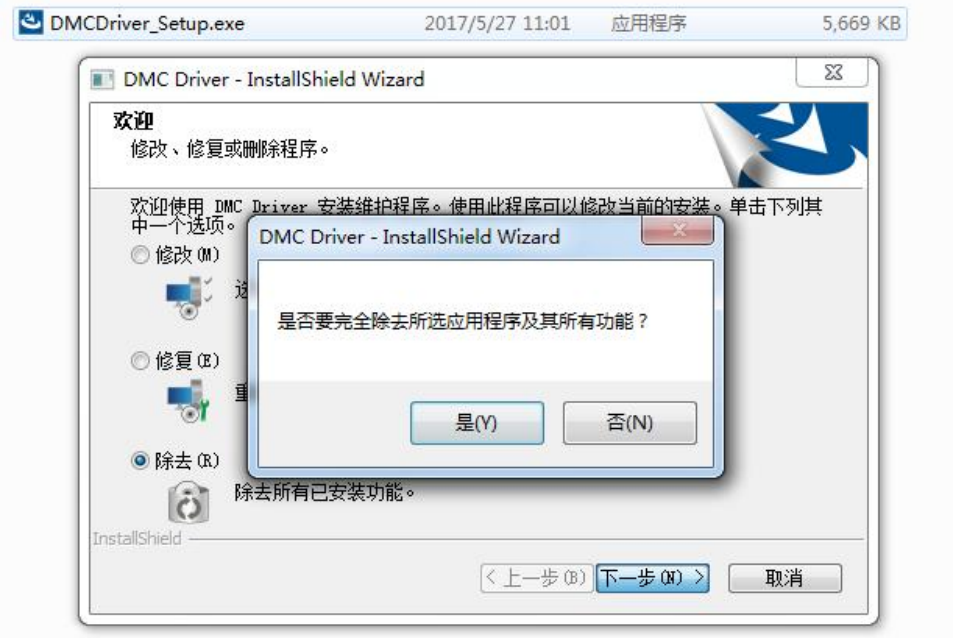


图 2 是否删除提示窗

3、卸载完成后，显示界面如图 3 所示，点击“完成”，完成卸载。

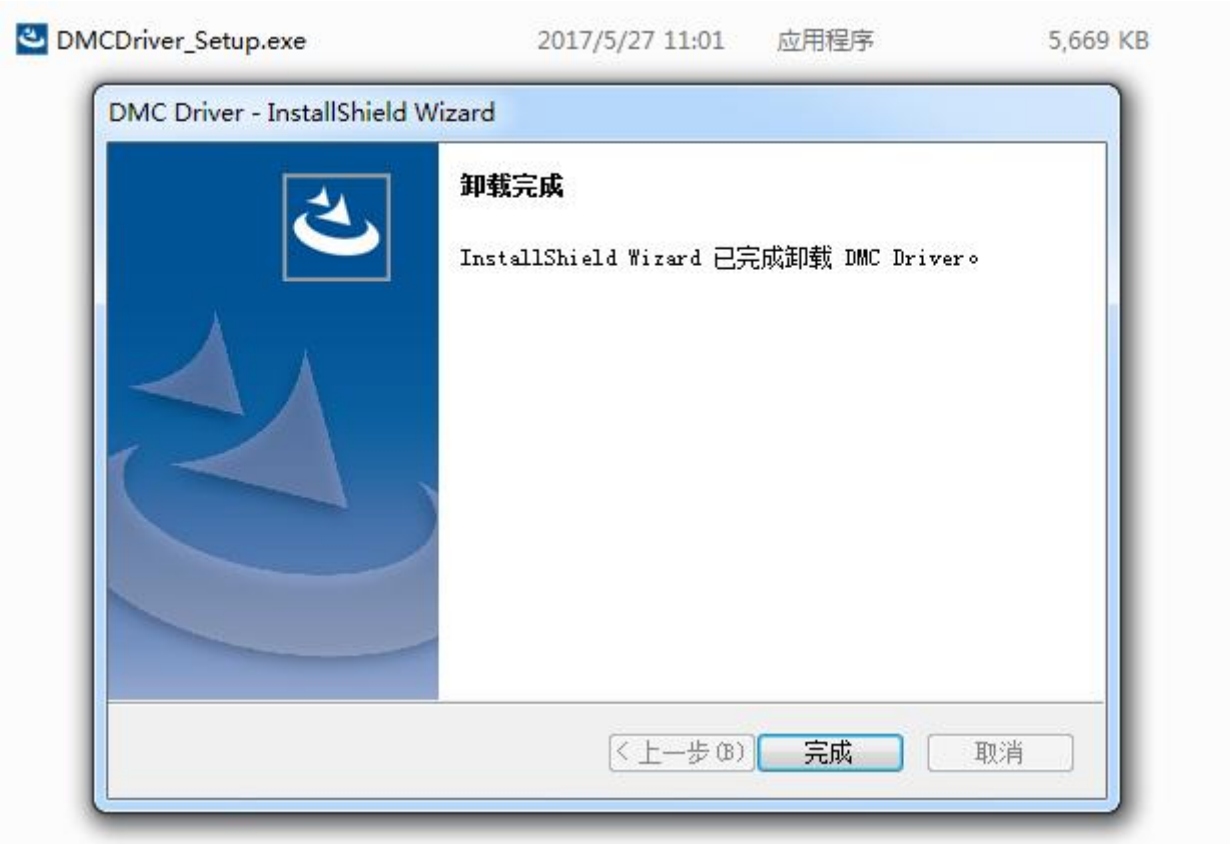


图 3 完成驱动程序的卸载

第 5 章　控制卡 motion 使用

5.1 概述

控制卡 Motion 软件是基于 Windows 平台开发的控制卡调试工具软件，支持雷赛智能公司的多种型号控制卡，其具有调试控制卡所有功能的能力，操作方便快捷。

控制卡 Motion 软件兼容了传统的脉冲控制卡和 EtherCAT 总线控制卡的功能，不但可以调试单轴运动、多轴运动、插补运动等传统运动功能和 IO 状态、轴状态等监视功能，还可以实现 EtherCAT 总线配置下载等相关功能。

5.2 功能描述

控制卡 Motion 软件是一款控制卡辅助调试软件，适用于初次使用雷赛控制卡入门用户或控制卡使用调试查错。主要功能如下（注意：以下功能与具体型号控制卡有关，不同类型控制卡可能功能有所不同）：

- 参数设置，支持各个轴所有参数的设置，包括脉冲模式、脉冲当量、加减速时间、回零模式、限位、伺服等参数设置。
- 运动状态，支持控制卡本地或 EtherCAT 模块的轴状态、IO 状态、轴专用信号监视。
- 单轴测试，支持单个轴定长运动、定速运动、回零运动测试。
- 多轴测试，支持直线插补、多类型圆弧/螺旋线单段插补运动测试。
- 手轮测试，支持手轮倍率、通道、模式等相关参数设置，启动、停止等相关操作。
- PVT 测试，支持 PVT 模式、PVT 数据等设置，以及 PVT 理论曲线显示等，支持多轴 PVT 同时操作。
- 单轴比较，支持比较点编辑、比较器状态监视等功能
- 二维比较，支持二维比较点编辑、比较器状态监视等功能
- 原点锁存，支持原点锁存模式、锁存源、触发方式等参数设置，配置、复位锁存器等操作。
- 高速锁存，支持锁存方式、锁存源、触发方式等参数设置，配置、复位锁存器等相关操作。
- 连续插补，支持直线插补、圆弧/螺旋线、延时、IO 输出等连续插补相关配置，以及插补速度、前瞻等参数设置，插补器状态、缓存数量、当前行号等相关状态监视。

- PWM 测试，直接设置输出 PWM，显示 PWM 曲线图。
- AD 测试，监视所有通道的 AD 电压，并显示曲线图。
- DA 测试，直接设置输出 DA 电压，显示 DA 曲线图。
- CAN 状态，支持 CAN 状态显示以及连接操作。
- EtherCAT 总线配置，扫描并配置 EtherCAT 总线参数，设置轴映射、IO 映射等相关操作。
- 板卡信息，显示控制卡型号、版本、固件类型、轴数量等相关信息。
- 固件升级，支持在线升级控制卡固件。
- 硬件复位，支持复位重启控制卡。
- 信息输出，支持输出函数调用过程，以及函数错误码。
- 错误码查询，支持查询控制卡函数调用返回的错误码对应的解释查询。

详细情况见附件《控制卡 Motion 使用手册》。

第 6 章　应用软件开发

DMC5X10 系列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“函数库详解”。

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：

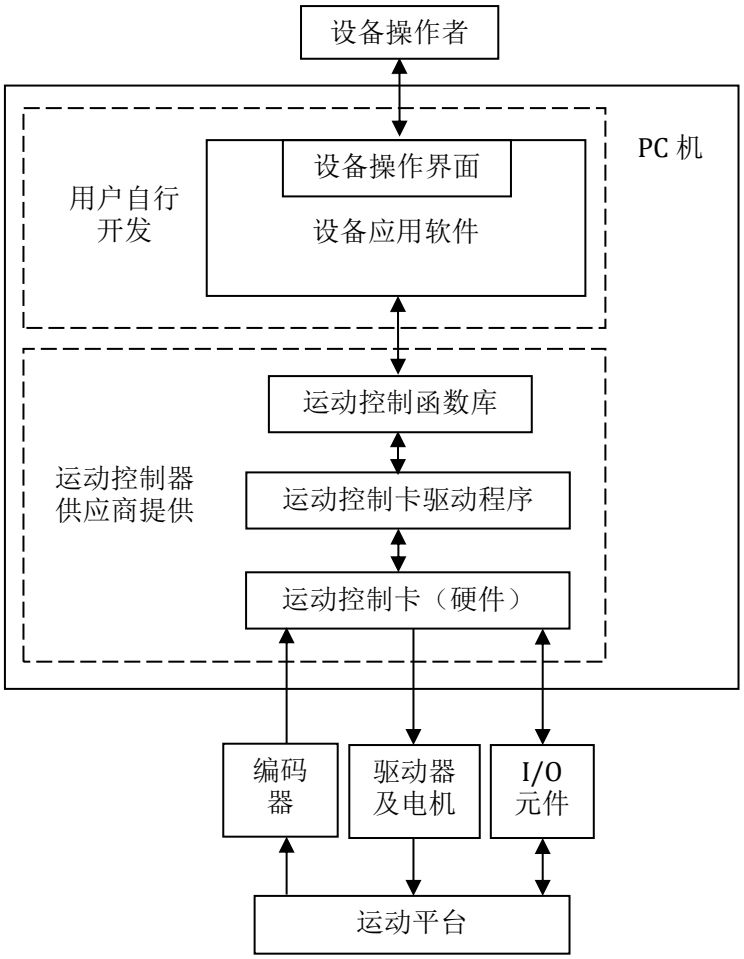


图 6.1　基于雷赛运动控制卡的自动化设备运动控制系统构架

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 DMC5X10 系列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。

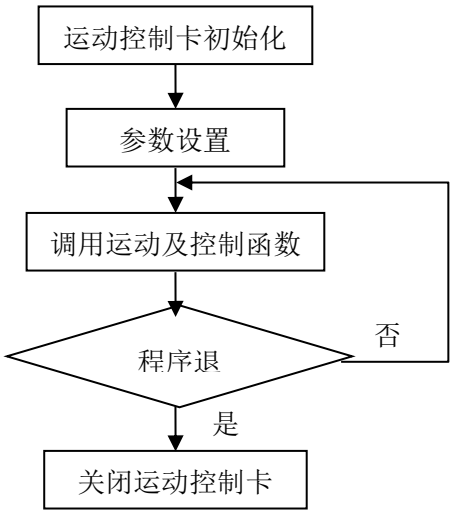


图 6.2　设备应用软件的典型流程

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）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。

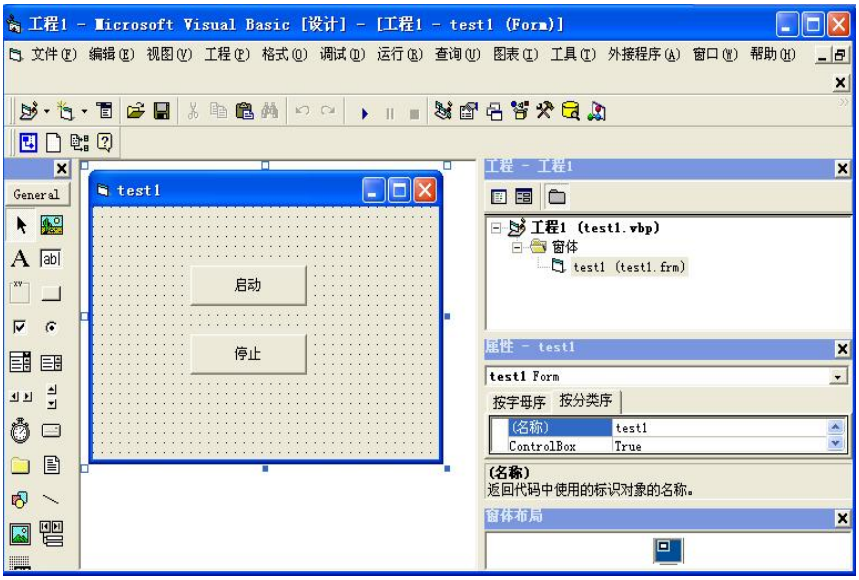


图 6.3　修改对话框（VB）

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。

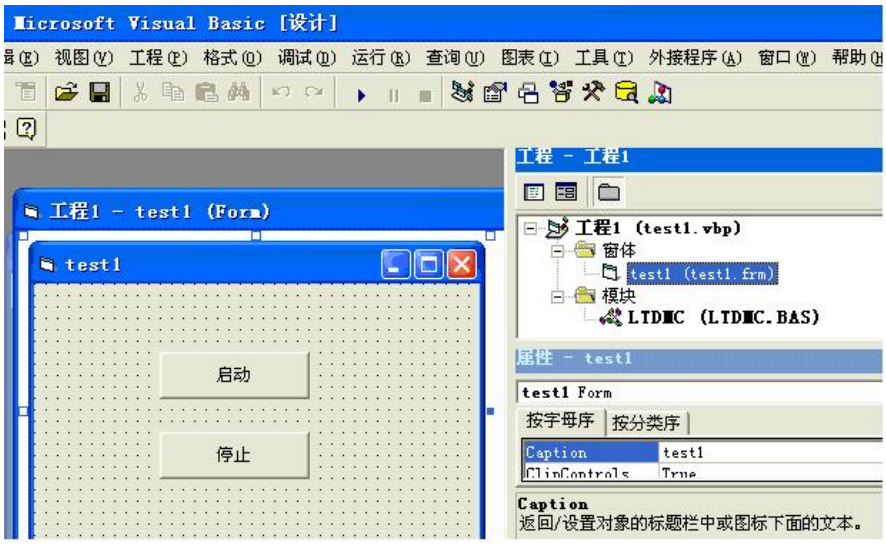


图 6.4　添加头文件

6）如图 6.5 所示，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 0,0,500,5000, 0.01,0.01,500

dmc_pmove 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

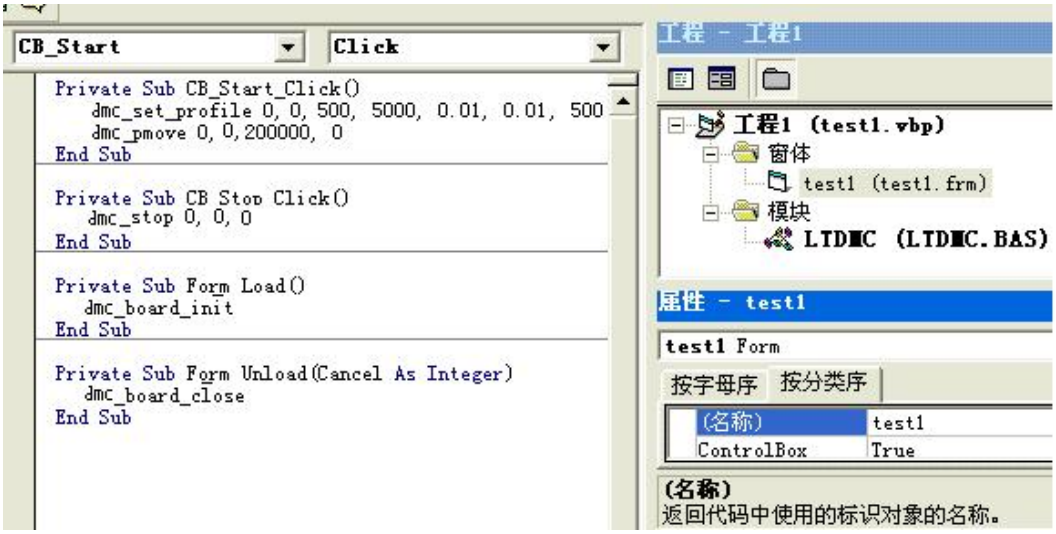


图 6.5　程序中调用运动控制卡库函数

7）程序编写完成。运行程序，显示界面如图 6.6 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 6.6　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 6.7。

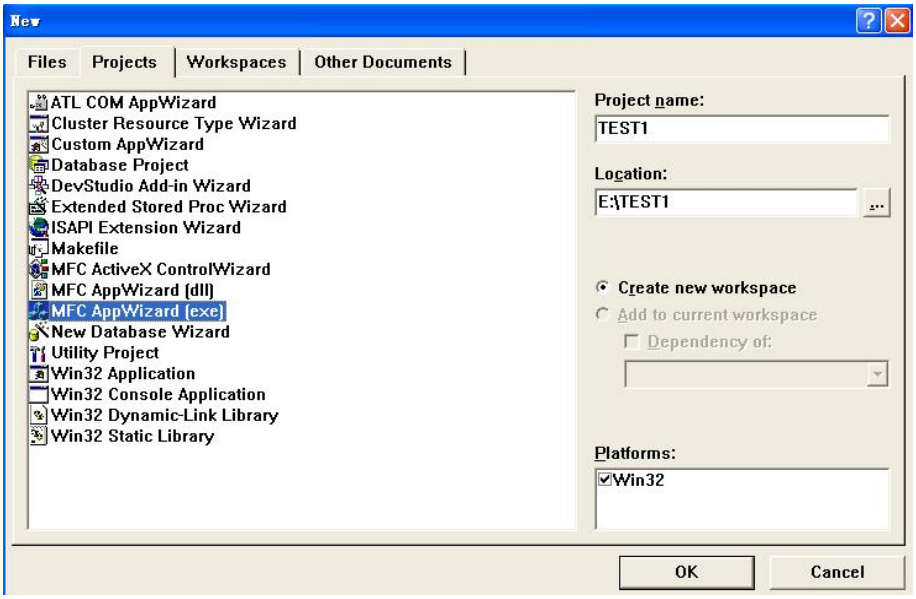


图 6.7　创建新工程

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”和“停止”；并分别命名为“IDC_BUTTON_Start”和“IDC_BUTTON_Stop”，如图 6.8 所示。

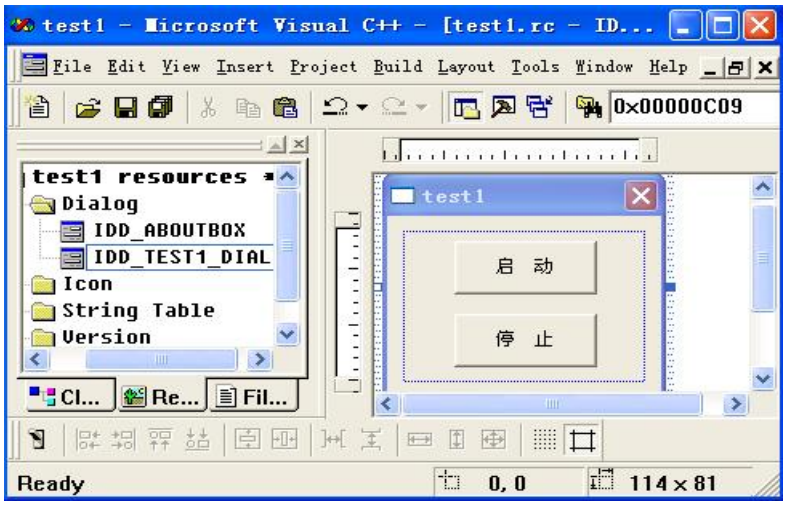


图 6.8　修改对话框

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:\tes1 目录。

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

10）打开 test1.cpp 文件，在程序开始部分添加相应语句：#include “LTDMC.h”，如图 6.9

所示。

11）在 CTest1Dlg::OnInitDialog()函数中添加代码：dmc_board_init();如图 6.10。

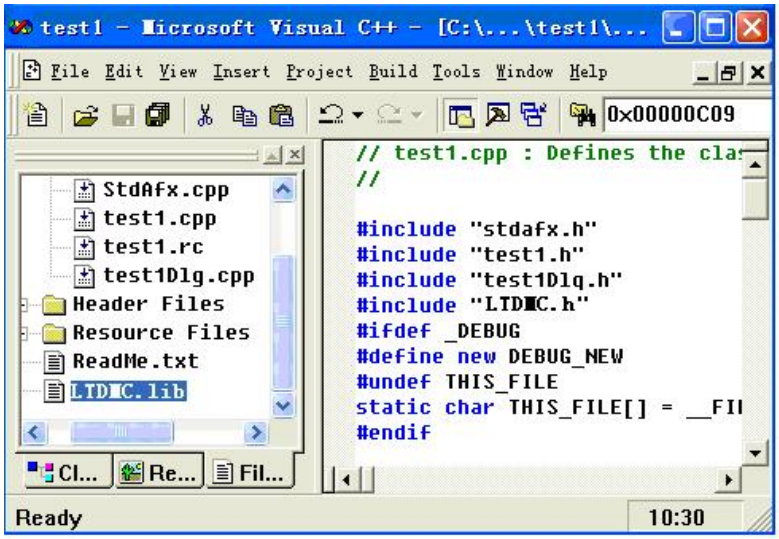


图 6.9　程序增加头文件

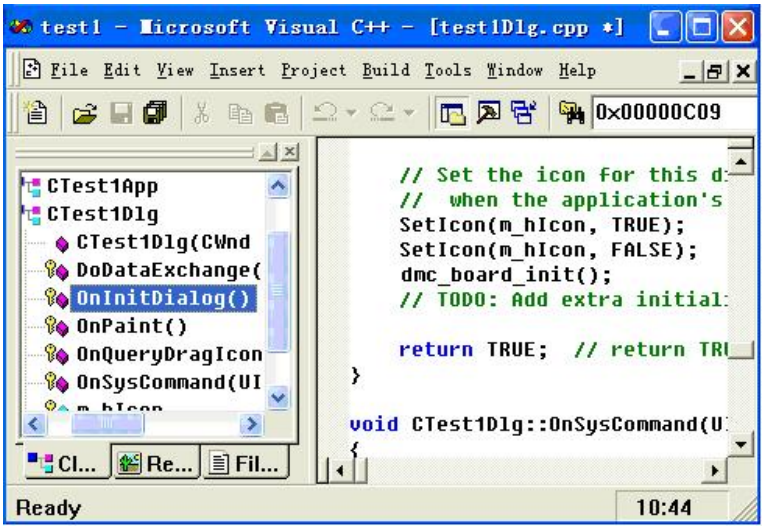


图 6.10　程序增加初始化函数

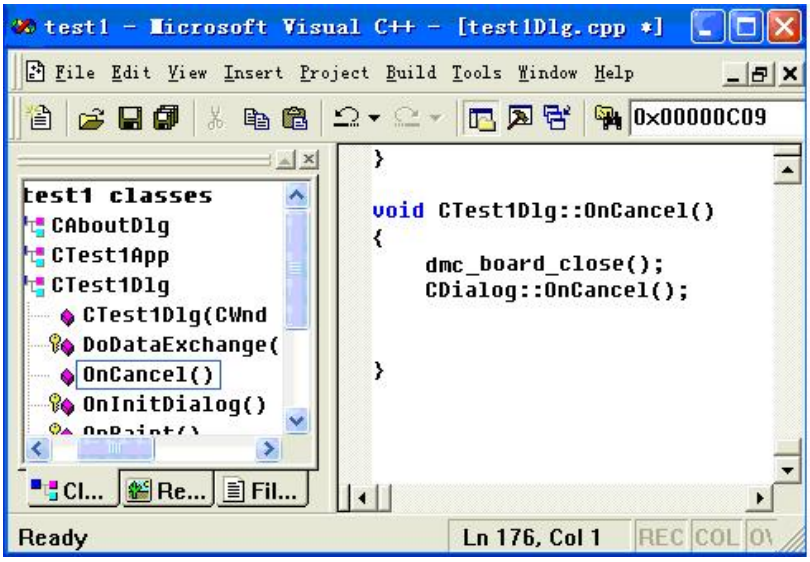


图 6.11　程序增加 OnCancel 函数

12）如图 6.11 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(0,0,500,5000, 0.01,0.01,500);
dmc_pmove(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);
```

如图 6.12 所示。

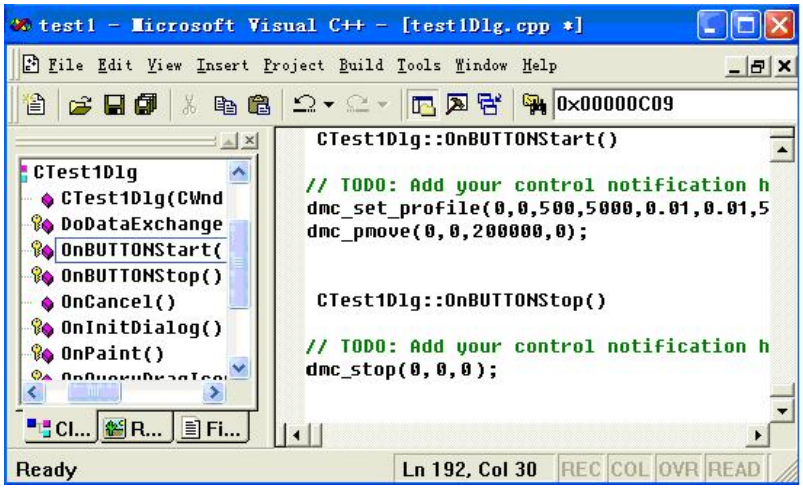


图 6.12　程序中调用运动控制卡库函数

14）编译程序后，运行程序，显示图 6.13 所示的界面。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 6.13　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 6.14 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 6.15 所示。

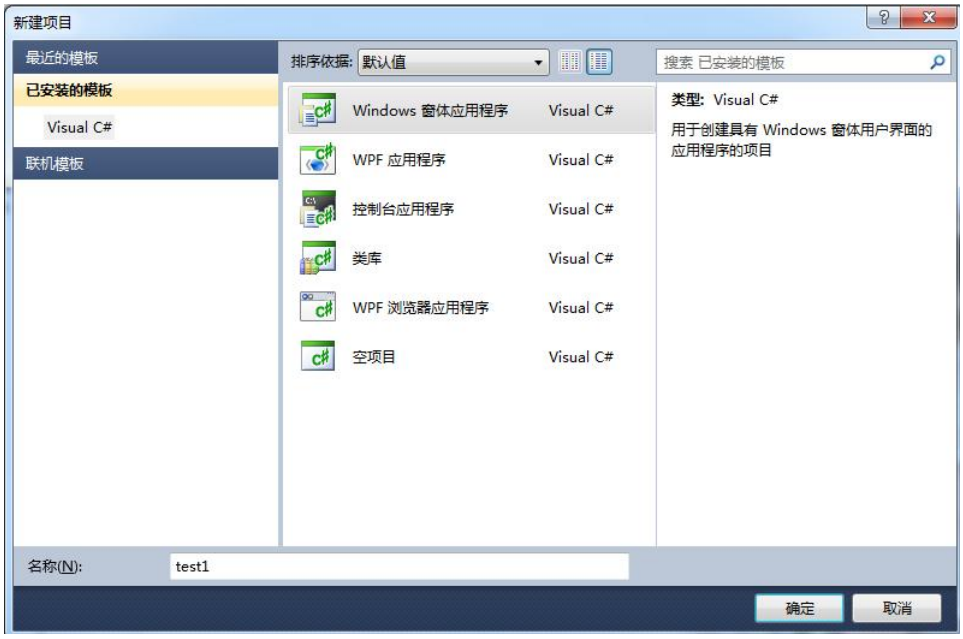


图 6.14　新建 Windous 窗体应用程序

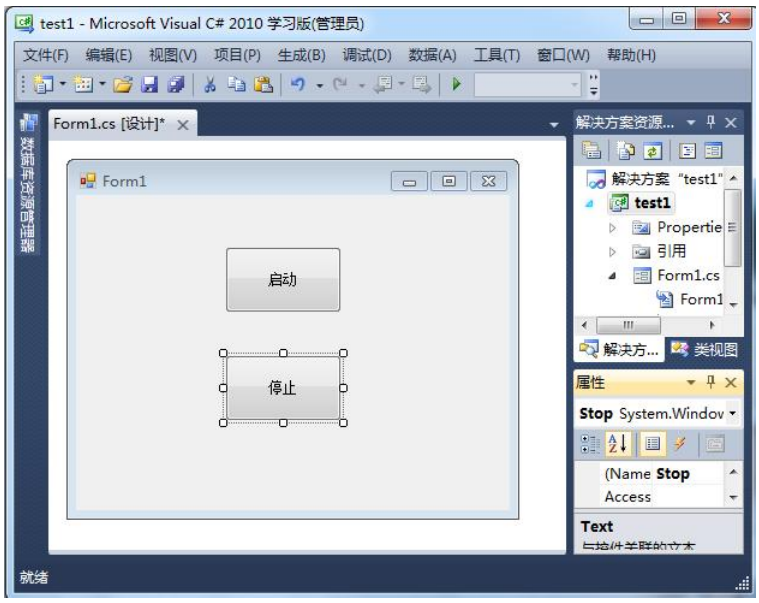


图 6.15　修改对话框

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 6.16 所示。

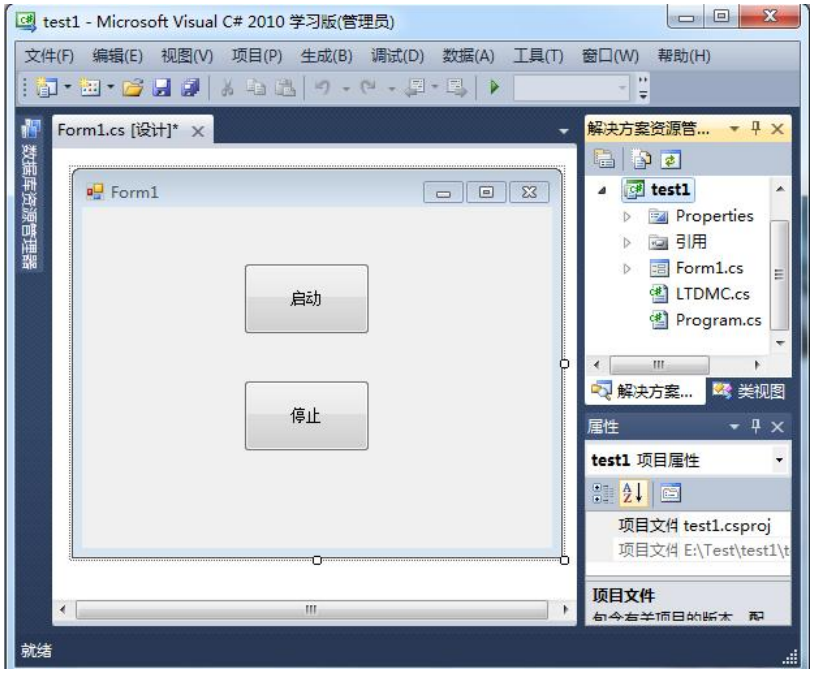


图 6.16　添加头文件

6）在代码文件开头处添加控制卡的命名空间：using csLTDMC;如图 6.17 所示。

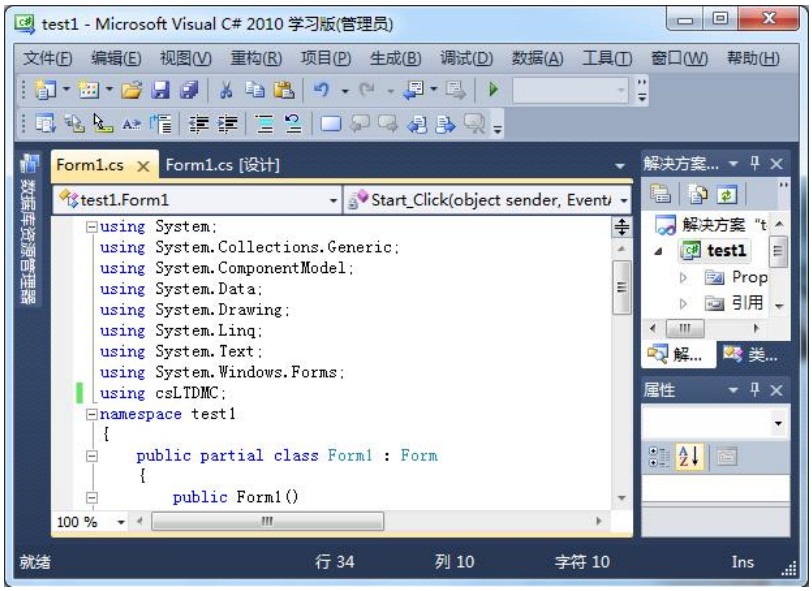


图 6.17　添加控制卡的命名空间

7）如图 6.18 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.dmc_set_profile (0,0,500,5000, 0.01,0.01,500);
LTDMC.dmc_pmove(0, 0, 200000, 0);
```

双击“停止”按钮，在 Stop_Click 事件中添加代码如下：

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

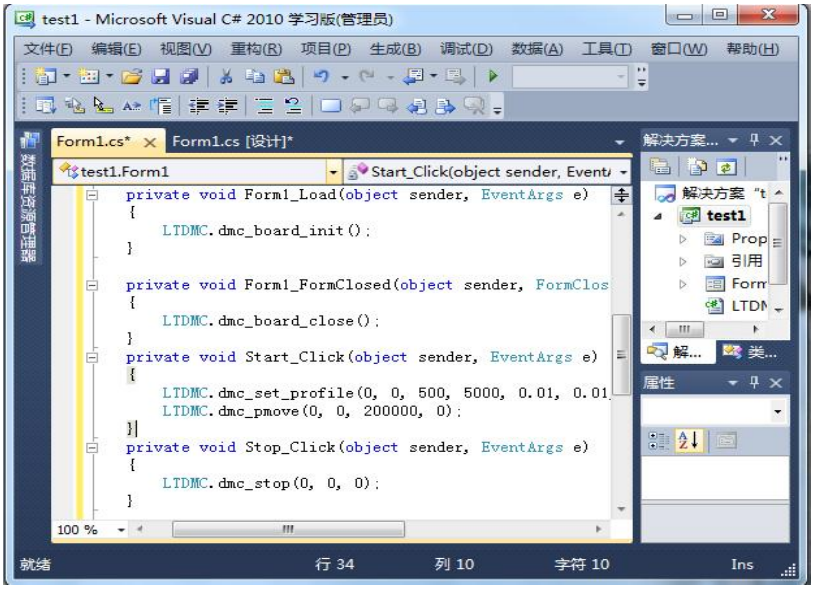


图 6.18　程序中调用运动控制卡库函数

7）程序编写完成。运行程序，显示界面如图 6.19 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。

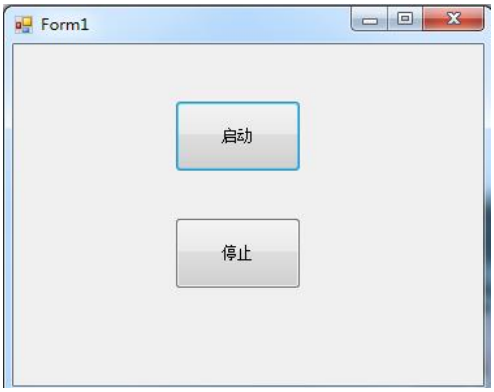


图 6.19　程序运行界面

第 7 章　以脉冲为单位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DMC5X10系列卡以脉冲为单位的基本功能方法。

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

7.1　板卡初始化及脉冲输出模式的设置

板卡初始化及脉冲输出模式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	
dmc_set_pulse_outmode	设置指定轴的脉冲输出模式	9.2 节

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。

DMC5X10 系列卡采用指令脉冲控制步进/伺服电机。由于市面上的众多电机驱动器厂家信号接口要求各有不同（常用的有六种类型），所以在使用运动控制卡控制具体的电机驱动器时，必须根据电机驱动器的接收信号类型，使用函数 dmc_set_pulse_outmode 对运动控制卡的脉冲输出模式进行正确的设定，电机才能正常工作。

关于脉冲输出模式类型详见章节 9.2 脉冲模式设置函数。

7.2　限位开关及急停开关的设置

在设备进行运动功能调试之前，必须确保安全机制的有效。

限位开关在运动平台出现超出行程的运动时，起到限制作用，使电机减速或紧急停止，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，必须保证限位开关的有效性。

急停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，必须保证急停开关的有效性。

7.2.1 限位开关的设置

DMC5X10 系列卡提供了限位开关设置函数 dmc_set_el_mode 来设置限位功能。用户必须根据设备的限位开关硬件接线，来设置限位开关工作的有效电平。

相关函数如下表 7.2 所示。

表 7.2 限位开关设置相关函数说明

名称	功能	参考
dmc_set_el_mode	设置限位开关信号	9.5 节
dmc_get_el_mode	读取限位开关信号设置	

假设设备正常运动时限位开关为高电平，运动平台碰到限位开关时为低电平，则此时应设置控制卡限位开关的有效电平为低电平。下面以 DMC5X10 系列运动控制卡为例对限位开关进行设置。

例程 7.1：设置限位开关为低电平

```
……
ushort MyCardNo, Myaxis, Myel_enable, Myel_logic, Myel_mode;

MyCardNo = 0;            //卡号
Myaxis = 0;              //轴号
Myel_enable = 1;         //正负限位使能
Myel_logic = 0;          //正负限位低电平有效
Myel_mode = 0;           //正负限位停止方式为立即停止
LTDMC.dmc_set_el_mode (MyCardNo,Myaxis,Myel_enable,Myel_logic,Myel_mode );
                         //设置 0 号轴限位信号
……
```

7.2.2 急停开关的设置

DMC5X10 系列卡提供了急停开关设置函数 dmc_set_emg_mode 来设置限位功能。用户必须根据设备的急停开关硬件接线，来设置急停开关工作的有效电平。

相关函数如下表 7.3 所示。

表 7.3 急停开关设置相关函数说明

名称	功能	参考
dmc_set_emg_mode	设置急停开关信号	9.20 节
dmc_get_emg_mode	读取急停开关信号设置	

DMC5X10 系列卡没有专用于 EMG 急停开关的硬件接口，用户需要根据自己的需求对轴

IO 进行映射配置，对应接口电路进行接线，然后调用急停开关设置函数进行设置。

假设使用控制卡的通用输入口 0 作为所有轴的急停信号。设备正常运动时急停开关为高电平，当急停开关为低电平时紧急停止运动，则此时应设置控制卡急停开关的有效电平为低电平。具体设置见例程 7.2，关于轴 IO 映射功能的实现详见 7.14 节 轴 IO 映射功能的实现。

例程 7.2：设置通用输入口 0 为所有轴的急停信号，低电平有效

```
……
ushort CardNo,Axis,IoType, MapIoType, MapIoIndex,Myenable,Mylogic;
UInt16     Filter;

CardNo=0;                    //卡号
IoType=3;                    //映射信号类型为急停信号
MapIoType=6;                 //映射对应的信号类型为通用输入
MapIoIndex=0;                //映射通用输入口 0
Filter=1;                    //滤波时间 1s
Myenable=1;                  //急停使能信号有效
Mylogic=0;                   //急停信号低电平有效
for(Axis=0;Axis<8;Axis++)//循环，依次对 0~7 号轴进行设置
{
LTDMC.dmc_set_AxisIoMap(CardNo, Axis, IoType, MapIoType, MapIoIndex, Filter);
                             //设置轴 IO 映射关系
LTDMC.dmc_set_emg_mode(CardNo, Axis, Myenable, Mylogic);//设置 EMG 信号使能
}
……
```

7.3 回原点运动的实现

7.3.1 回原点步骤

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。DMC5X10 系列卡共提供了 13 种回原点方式，DMC5C10 后四轴只提供了六种回原点方式。

回原点运动主要步骤如下：

1）使用 dmc_set_home_pin_logic 函数设置原点开关的有效电平；

2）使用 dmc_set_homemode 函数设置回原点方式；

3）设置回原点运动的速度曲线；

4）使用 dmc_home_move 函数执行回原点运动；

5）回到原点后，使用 dmc_get_home_result 函数读取回零结果，正常完成则指令脉冲计数器自动清零。

7.3.2 回原点方式

DMC5X10 系列卡提供了 13 种回原点运动的方式，**注意**：DMC5C10 前 8 轴提供 13 种回零方式，后四轴只支持 0、1、2、10、11、12 六种回零模式。

方式 0：一次回零

该方式以设定速度回原点；适合于行程短、安全性要求高的场合。动作过程为：电机从初始位置以恒定速度向原点方向运动，当到达原点开关位置，原点信号被触发，电机立即停止（过程 0）；将停止位置设为原点位置，如图 7.1 所示。

图 7.1 一次回零方式示意图

方式 1：一次回零加回找

该方式先进行方式 0 运动，完成后再反向回找原点开关的边缘位置，当原点信号第一次无效的时候，电机立即停止；将停止位置设为原点位置如图 7.2 所示。

图 7.2 一次回零加回找方式示意图

方式 2：两次回零

如图 7.3 所示，该方式为方式 0 和方式 1 的组合。先进行方式 1 的回零加反找，完成后再进行方式 0 的一次回零。可参见方式 0 和方式 1 的说明。

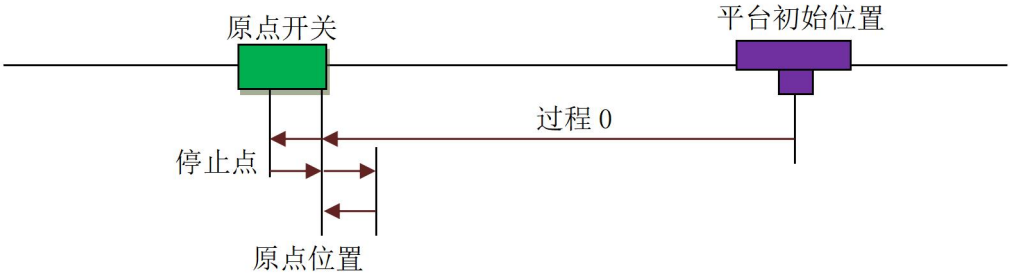


图 7.3 二次回零方式示意图

方式 3：一次回零后再找 EZ 信号

该方式在回原点运动过程中，当找到原点信号后，还要等待该轴的 EZ 信号出现，此时电机停止。回原点过程如图 7.4 所示。

图 7.4　一次回零后再找 1 个 EZ 信号回零方式示意图

方式 4：记 1 个 EZ 信号回零

该方式在回原点运动过程中，当检测到该轴的 EZ 信号出现一次后，此时电机停止。回原点过程如图 7.5 所示。

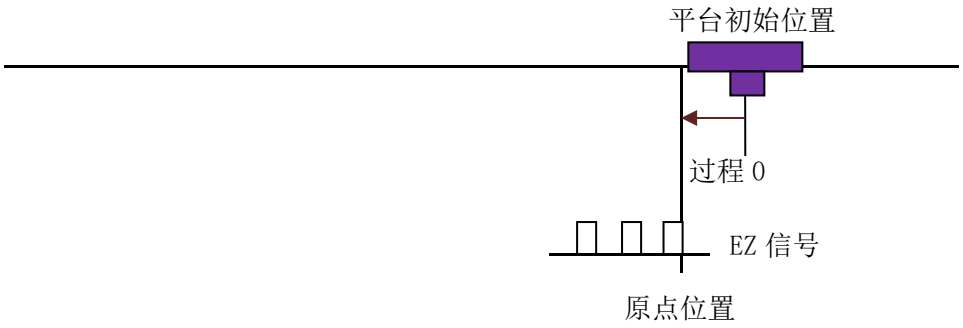


图 7.5　记 1 个 EZ 信号回零方式示意图

方式 5：一次回零再反找 EZ 信号

该方式在回原点运动过程中，当找到原点信号后，减速停止，然后以反找速度反向找到 EZ 生效此时电机停止。回原点过程如图 7.6 所示。

图 7.6　一次回零反找一个 EZ 进行回零

方式 6：原点锁存

如图 7.7 所示，电机先以设定速度回原点，当原点开关边沿触发时，将当前位置锁存下来，同时电机减速停止。电机减速停止完成后再反向回找锁存位置，运动到锁存位置，电机停止。

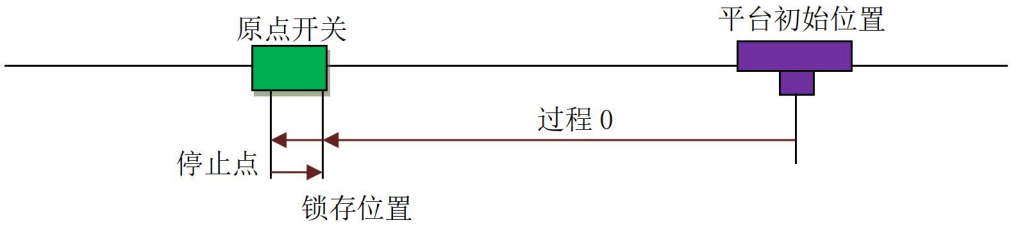


图 7.7　原点锁存回零方式示意图

方式 7：原点锁存加同向 EZ 锁存

该方式先以方式 6 执行一次原点锁存回零，完成后继续沿设定回零方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 7.8 所示。

图 7.8　原点锁存加同向 EZ 锁存回零方式示意图

方式 8：单独记一个 EZ 锁存

在回零过程中检测到 EZ 有效边沿出现，锁存当前位置，执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 7.9 所示。

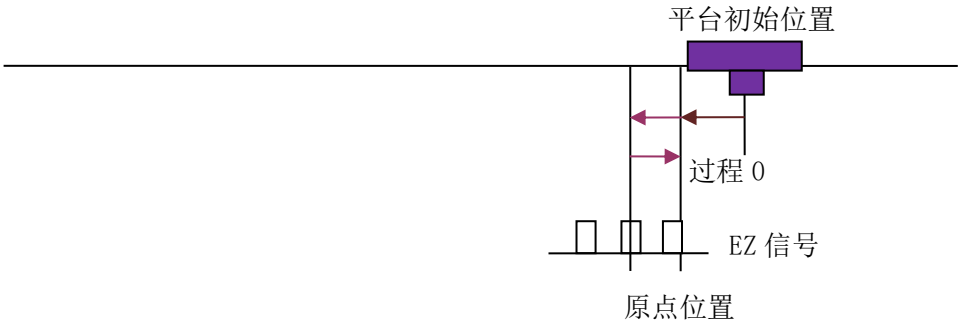


图 7.9　单独记一个 EZ 锁存回零方式示意图

方式 9：原点锁存加反向 EZ 锁存

该方式先以方式 6 执行一次原点锁存回零，完成后以与设定回零方向相反的方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 7.10 所示。

图 7.10　原点锁存加反向 EZ 锁存进行回零

方式 10：一次限位回零

该方式以设定速度回原点；适合于行程短、安全性要求高的场合。动作过程为：电机从初始位置以恒定速度向限位方向运动，当到达限位开关位置，限位信号被触发，电机立即停止（过程 0）；将停止位置设为原点位置，如图 7.11 所示。

图 7.11　一次限位方式示意图

方式 11：一次限位回零加回找

该方式先进行方式 10 运动，完成后再反向回找限位开关的边缘位置，当限位信号第一次无效的时候，电机立即停止；将停止位置设为原点位置如图 7.12 所示。

方式 12：两次限位回零

如图 7.13 所示，该方式为方式 10 和方式 11 的组合。先进行方式 11 的回零加反找，完成后再进行方式 10 的一次回零。可参见方式 10 和方式 11 的说明。

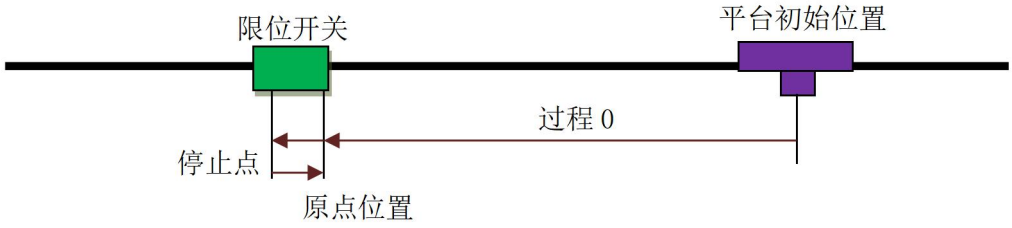


图 7.12　一次限位加回找方式示意图

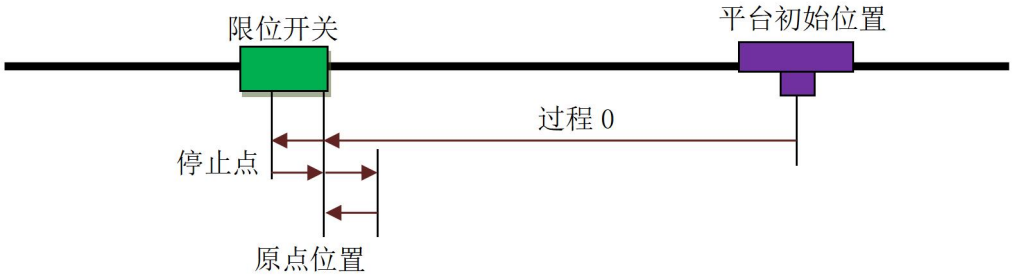


图 7.13　二次限位方式示意图

相关函数如下表 7.4 所示。

表 7.4　回原点相关函数说明

名称	功能	参考
dmc_set_home_pin_logic	设置原点信号的有效电平	9.3 节
dmc_set_homemode	选择回原点模式	
dmc_home_move	按指定的方向和速度方式开始回原点	
dmc_get_home_result	读取回零状态	

例程 7.3：方式 1 低速回原点

```
……
ushort MyCardNo,Myaxis,Myorg_logic,Myfilter,Myhome_dir,Mymode,MyEZ_count;
double   Myvel_mode;

MyCardNo=0;                  //卡号
Myaxis=0;                    //轴号
Myorg_logic=0;               //原点信号有效电平为低电平有效
Myfilter=0;                  //保留参数
Myhome_dir=1;                //回零方向为正向
Myvel_mode = 0;              //回零模式为低速
Mymode=0;                    //回零方式为一次回零
MyEZ_count=0;                //保留参数
LTDMC.dmc_set_home_pin_logic(MyCardNo, Myaxis, Myorg_logic, Myfilter);
                             //设置原点信号
LTDMC.dmc_set_homemode(MyCardNo, Myaxis, Myhome_dir, Myvel_mode, Mymode, MyEZ_count);
```

```
                                    //设置回原点模式
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 500, 1000, 0.1, 0.1, 500);
                                    //设置 0 号轴梯形速度曲线参数
LTDMC.dmc_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

DMC5X10 系列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 7.14 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。

在 DMC5X10 系列卡的函数库中距离或位置的单位为 pulse；速度单位为 pulse/s。

DMC5X10 系列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。

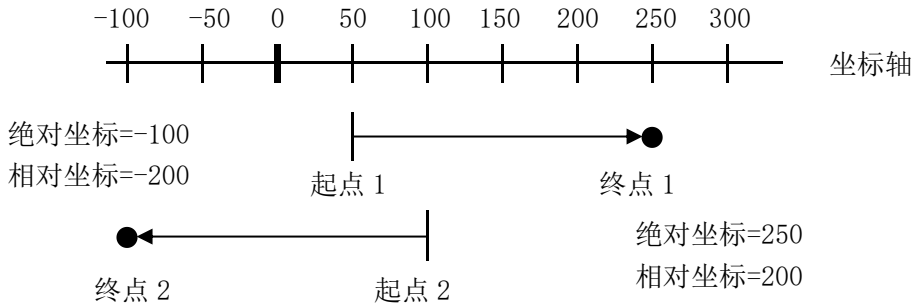


图 7.14　绝对坐标与相对坐标中轨迹终点的不同表达方式

7.4.1 梯形速度曲线下的点位运动

梯形速度曲线是位置控制中最基本的速度控制方式。参见图 2.3。

相关函数如表 7.5 所示。

表 7.5　梯形速度点位运动相关函数说明

名称	功能	参考
dmc_set_profile	设置单轴运动速度曲线	9.8 节
dmc_pmove	指定轴点位运动	9.9 节
dmc_check_done	检测指定轴的运动状态	9.7 节

例程 7.4：执行以梯形速度曲线作点位运动

```
……
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 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
MyDist=60000;                                       //位移为 6000pulse
Myposi_mode=0;                                       //保留参数 0
LTDMC.dmc_set_pulse_outmode(MyCardNo, Myaxis, 0);//设置 0 号轴脉冲输出方式
LTDMC.dmc_set_profile(MyCardNo, Myaxis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel);
                                                    //设置单轴运动速度曲线
LTDMC.dmc_pmove(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，如图 7.15 所示。若在减速时改变目标位置，电机的速度将如图 7.16 所示发生变化。

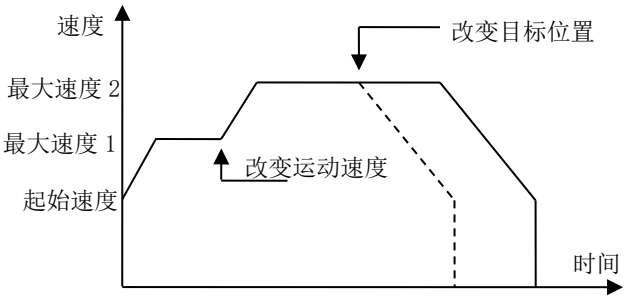


图 7. 14 改变速度及改变目标位置

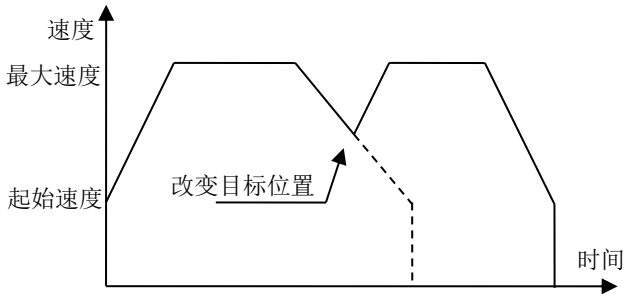


图 7. 15 减速时改变目标位置

实现这 2 个功能的函数如表 7.6 所示：

表 7.6　梯形速度下改变速度、终点的相关函数说明

名称	功能	参考
dmc_change_speed	在线改变指定轴的当前运动速度	9.9 节
dmc_reset_target_position	在线改变指定轴的当前目标位置	

例程 7.5：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
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
MyDist = 60000;                                        //位移为 6000pulse
LTDMC.dmc_set_pulse_outmode(MyCardNo, Myaxis, 0);      //设置 0 号轴脉冲输出方式
LTDMC.dmc_set_profile(MyCardNo, Myaxis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel);
                                                       //设置单轴运动速度曲线
LTDMC.dmc_pmove(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(MyCardNo, Myaxis, 10000, 0);//执行在线变速，速度变为 10000pulse/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 100000pulse
}
……
```

7.4.2　S 形速度曲线运动模式

梯形速度曲线较简单；而 S 速度曲线运动更平稳。参见图 2.4。

相关函数如表 7.7 所示。

表 7.7　S 形速度控制相关函数说明

名称	功能	参考

dmc_set_profile	设置单轴运动速度曲线	9.8 节
dmc_set_s_profile	设置单轴运动速度曲线 S 段参数值	
dmc_pmove	指定轴点位运动	9.9 节
dmc_check_done	检测指定轴的运动状态	9.7 节

例程 7.6：执行以 S 形速度曲线作点位运动

```
……
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
MyMin_Vel = 200;                                       //起始速度 200pulse/s
MyMax_Vel = 5000;                                      //最大速度 5000pulse/s
MyTacc = 0.05;                                         //加速时间 0.01s
MyTdec = 0.05;                                         //减速时间 0.01s
MyStop_Vel = 200;                                      //停止速度 200pulse/s
MyDist = 60000;                                        //位移为 6000pulse
Myposi_mode = 0;                                       //保留参数 0
Mys_para=0.01;                                         //s 段时间 0.01s
LTDMC.dmc_set_pulse_outmode(MyCardNo, Myaxis, 0);      //设置 0 号轴脉冲输出方式
LTDMC.dmc_set_profile(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_pmove(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();
}
……
```

如果因为距离太短或加速太慢原因导致电机速度在加速段不能升至设定的最大值 Max_Vel 时，理论上加速段将突然切换至减速段，从而引起该轴出现较大震动。为了避免出现这种问题，DMC5X10 系列卡内置有自动调整功能，使得加减速段的过渡保持平滑，如图 7.16 所示。

在 S 形速度曲线下的点位运动过程中，也可以调用 dmc_change_speed 和 dmc_reset_target_position 函数实时改变运行速度和目标位置。但多轴插补运行情况下不能实时改变运行速度和目标位置。

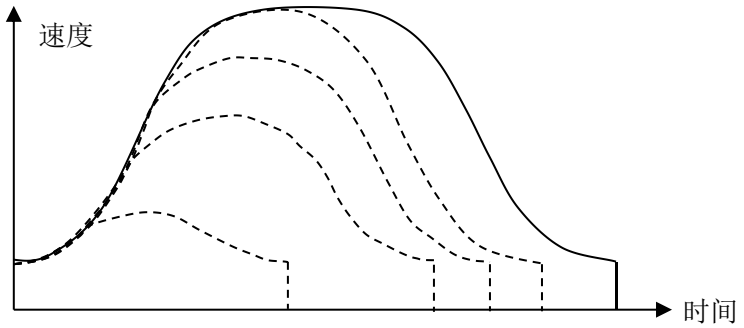


图 7.16　自动降速避免尖三角形

7.4.3　多轴联动

多轴同时做点位运动，称之为多轴联动。

DMC5X10 系列卡可以控制多个电机同时执行 dmc_pmove 这类单轴运动函数。所谓同时执行，是在程序中顺序调用 dmc_pmove 等函数，因为程序执行速度很快，在几微秒内电机都开始运动，感觉是同时开始运动。

多轴联动在各轴速度设置不当时，各轴停止时间不同、在起点与终点之间运动的轨迹也不是直线，参见图 2.2。

如果从起点到终点都需要按照规定的路径运动，就必须采用插补运动功能。

7.5　连续运动的实现

连续运动中，DMC5X10 系列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启动时的速度曲线减速停止。相关函数如表 7.8 所示。

表 7.8　连续运动相关函数说明

名称	功能	参考
dmc_vmove	指定轴连续运动	9.9 节
dmc_stop	指定轴停止运动	9.7 节

在执行连续运动过程中，可以调用 dmc_change_speed 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 7.17 和图 7.18 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。

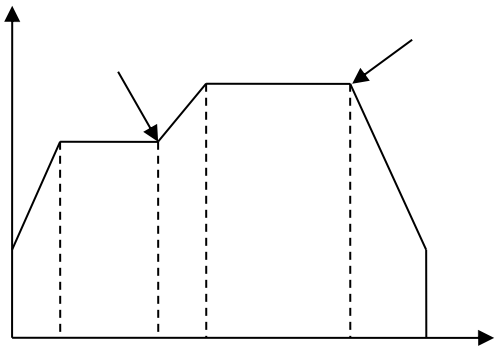


图 7.17　梯形运动中变速

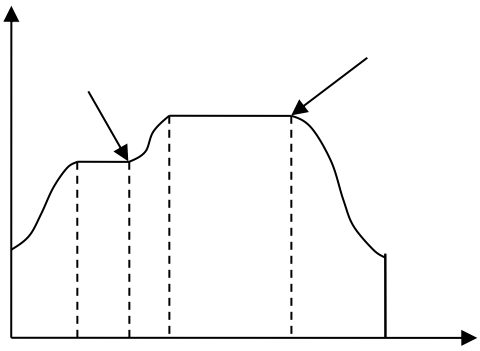


图 7.18　S 形运动中变速

例程 7.7：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 200pulse/s
MyMax_Vel = 5000;                                      //最大速度 5000pulse/s
MyTacc = 0.05;                                         //加速时间 0.01s
MyTdec = 0.05;                                         //减速时间 0.01s
MyStop_Vel = 200;                                      //停止速度 200pulse/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_pulse_outmode(MyCardNo, Myaxis, 0);      //设置 0 号轴脉冲输出方式
LTDMC.dmc_set_profile(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(MyCardNo, Myaxis, 10000, 0);
                                                       //在线变速，速度变为 10000pulse/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 PVT 运动功能的实现

DMC5X10 系列卡共提供四种 PVT 模式，分别为 PTT、PTS、PVT、PVTS 模式。其中 PTT、PTS 运动模式用于单轴速度规划功能，PVT、PVTS 运动则用于多轴轨迹规划功能，用户可以根据实际需求选择适合的 PVT 模式。

7.6.1 单轴任意速度规划功能的实现

DMC5X10 系列卡提供了两种 PVT 模式来实现单轴任意速度规划的功能，分别为 PTT 运动模式和 PTS 运动模式。PTT 运动模式是用于单轴梯形速度的规划，而 PTS 运动模式则用于单轴 S 形速度的规划。

7.6.1.1 PTT 运动模式

PTT 模式非常灵活，能够实现单轴任意速度规划。用户通过直接输入位置和时间参数描述运动规律。相关函数如表 7.9 所示。

表 7.9 PTT 模式运动相关函数说明

名称	功能	参考
dmc_PttTable	向指定数据表传送数据，采用 PTT 描述方式	9.12 节
dmc_PvtMove	启动 PVT 运动	

例程 7.8：PTT 模式运动

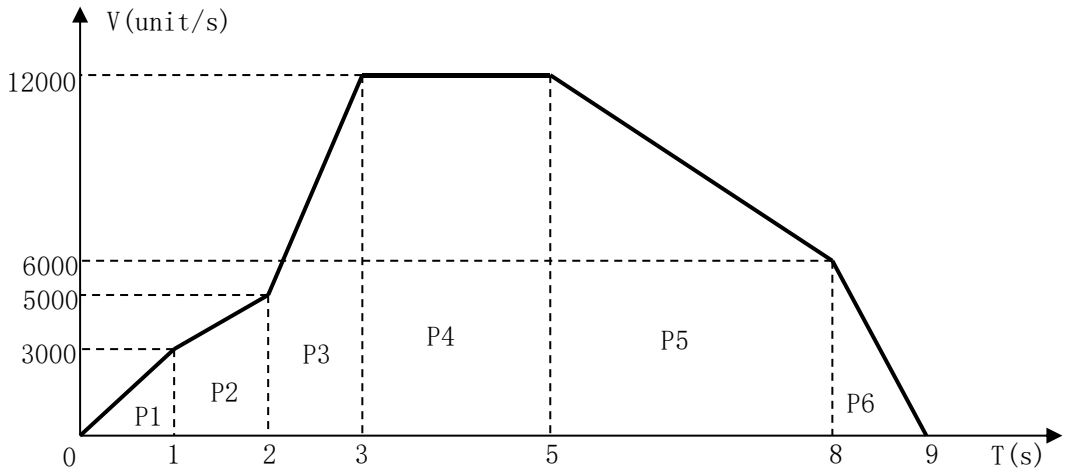


图 7.19 PTT 模式下的 V-T 曲线规划

规划如图 7.19 所示速度曲线，用 PTT 模式规划该速度曲线非常简单。

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

将各段位移量累加，得到 PTT 模式下各点的位置和时间数据，如表 7.10 所示。

表 7.10　PTT 模式数组数据

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

编写程序如下：

```
……
ushort MyCardNo,Myaxis, MyAxisNum, MyCount;
ushort []My_AxisList=new ushort [1];
MyCardNo = 0;                                    //卡号为 0
Myaxis=0;                                        //轴号 0
MyAxisNum=1;                                     //参与运动的轴数为 1
MyCount = 7;                                     //有 7 组数据
My_AxisList[0] = 0;                              //0 号轴参与 PTT 运动

double[] MyPTime = new double[7] {0,1,2,3,5,8,9 };//定义 PTT 时间数组
int[] MyPPos = new double[7] { 0, 1500, 5500, 14000, 38000, 65000, 68000 };
                                                 //定义 PTT 位置数组
LTDMC.dmc_PttTable(MyCardNo,Myaxis, MyCount, MyPTime, MyPPos);
                                                 //采用 PTT 模式传递数据
LTDMC.dmc_PvtMove(MyCardNo, MyAxisNum, My_AxisList);//执行 PTT 运动
……
```

PTT 运动结果(位置-时间曲线、加速度-时间曲线)如图 7.20、7.21 所示。

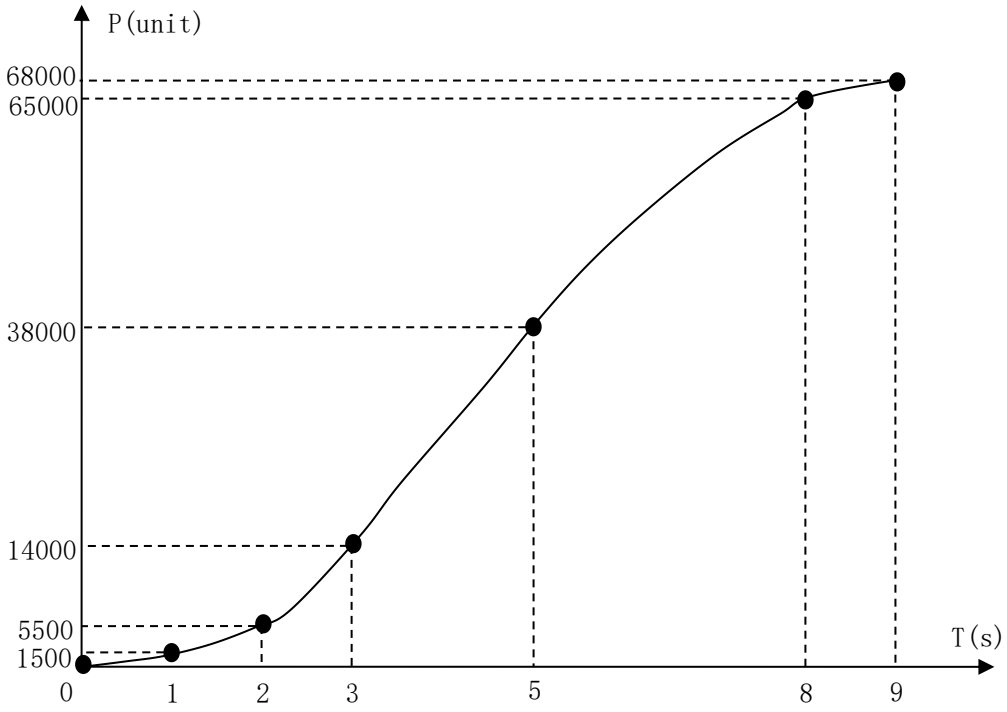


图 7.20　PTT 运动得到的位移曲线

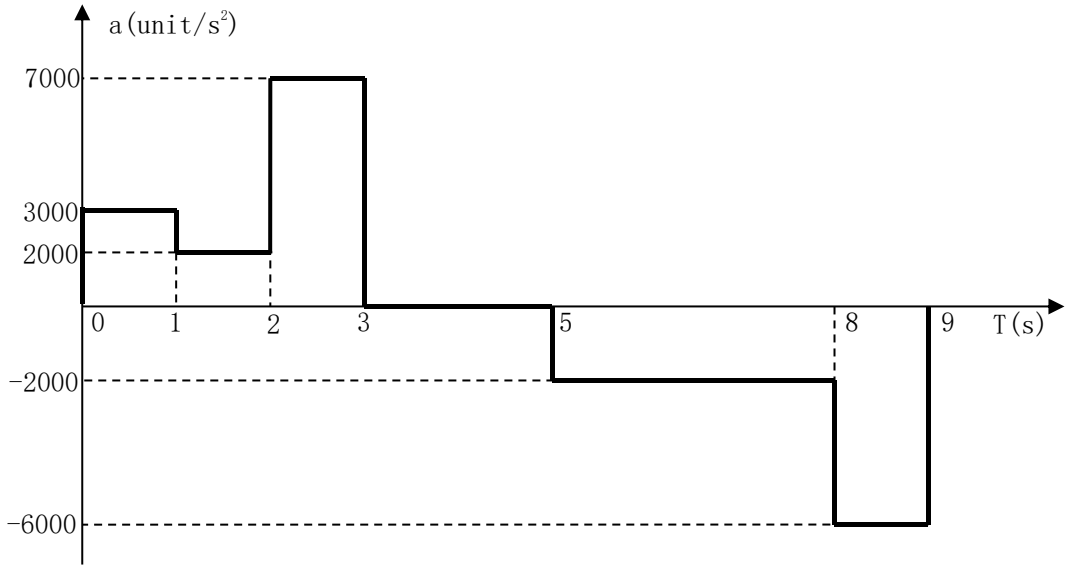


图 7.21　PTT 运动的加速度曲线

7.6.1.2　PTS 运动模式

PTS 运动模式是 PTT 的扩展功能模式，可以使各数据点的速度过渡更加平滑。用户通过输入位置、时间、百分比参数描述运动规律。数据点的百分比参数是指：相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比。相关函数如表 7.11 所示。

表 7.11　PTS 模式运动相关函数说明

名称	功能	参考
dmc_PtsTable	向指定数据表传送数据，采用 PTS 描述方式	9.12 节
dmc_PvtMove	启动 PVT 运动	

例程 7.9：PTS 模式运动

由图 7.19 可知，例程 7.8 的加速度曲线存在突变，因此，运动过程中有冲击现象。如果要获得更加平滑的速度曲线，可以使用 PTS 模式运动，其速度曲线比图 7.19 所示的速度曲线平滑。

设置各点的速度百分比参数如表 7.12 所示；其对应的加速度-时间曲线如图 7.22 所示，每段的加减速时间为该段时间值×速度百分比；每段的加速度最大值与 PTT 模式下的加速度值不一样，这是因为内部算法作了平滑处理。其对应的位置-时间曲线、速度-时间曲线如图 7.23、7.24 所示。

表 7.12　PTS 模式数组数据

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

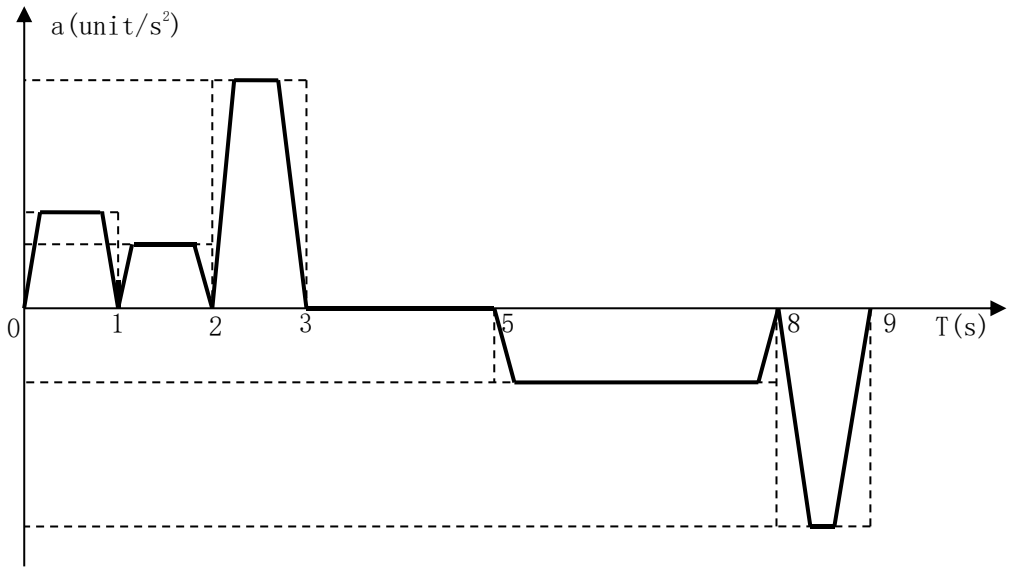


图 7.22　PTS 模式下的加速度曲线

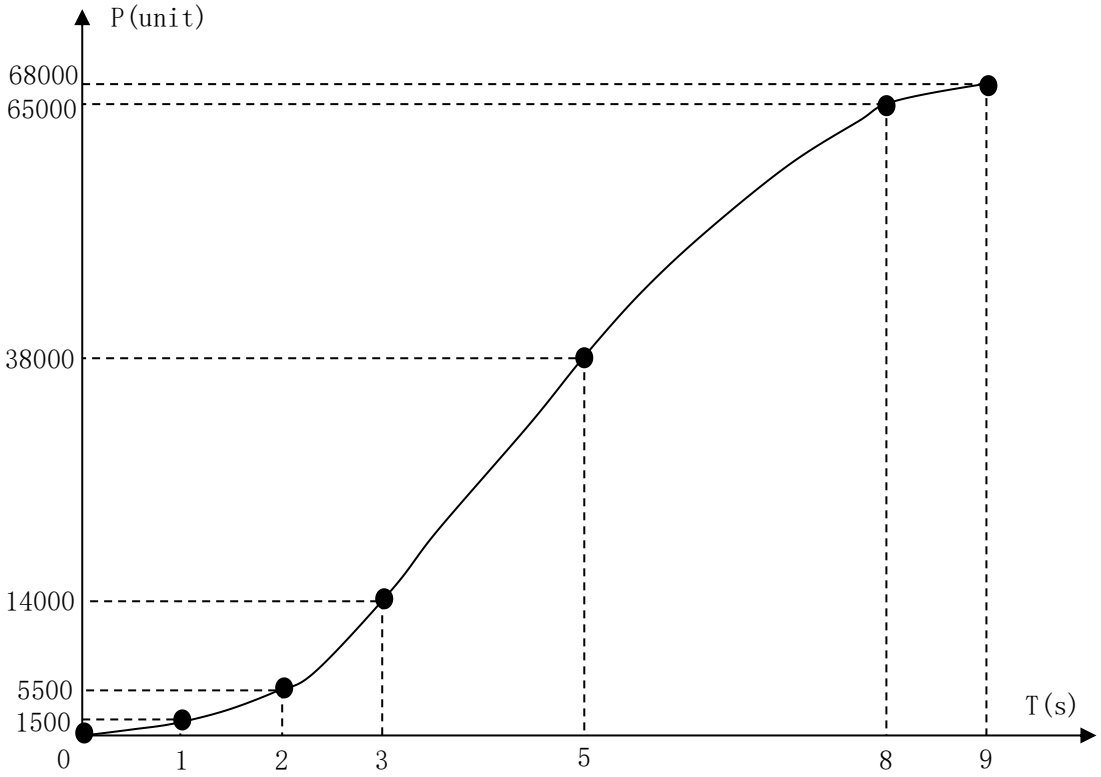


图 7.23　PTS 运动得到的位移曲线

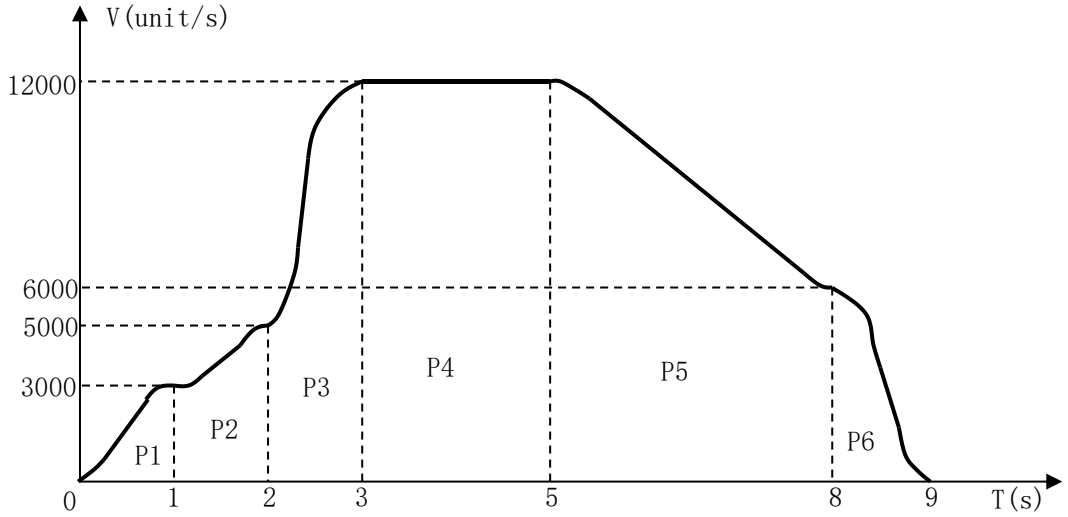


图 7.24　PTS 运动得到的速度曲线

编写程序如下：

```
……
ushort MyCardNo, Myaxis, MyAxisNum, MyCount;
ushort[] My_AxisList = new ushort[1];
MyCardNo = 0;                          //卡号为 0
Myaxis = 0;                            //轴号 0
MyAxisNum = 1;                         //参与运动的轴数为 1
```

```
MyCount = 7;                                    //有 7 组数据
My_AxisList[0] = 0;                             //0 号轴参与 PTS 运动

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

7.6.2 多轴高级轨迹规划功能的实现

DMC5X10 系列卡具有 PVT 高级运动曲线规划的功能，当用户需要规划一些特殊的运动轨迹而使用单轴运动及插补运动无法满足需求时，可以尝试使用 PVT 来规划自己的运动轨迹。

DMC5X10 系列卡共提供了两种 PVT 模式来实现多轴轨迹规划的功能，分别为 PVT、PVTS 运动模式。PVT 模式用于对各点的位置、时间、速度都有要求的轨迹规划，PVTS 模式用于只对各点的位置、时间有要求，而对各点的速度无太多要求的轨迹规划。

7.6.2.1 PVT 运动模式

PVT 模式使用一系列数据点的位置、速度、时间参数描述运动规律。

相关函数如表 7.13 所示。

表 7.13 PVT 模式运动相关函数说明

名称	功能	参考
dmc_PvtTable	向指定数据表传送数据，采用 PVT 描述方式	9.12 节
dmc_PvtMove	启动 PVT 运动	

注意：当设置的各点 P、V、T 数据不合理时，很难得到理想的轨迹曲线。

理想轨迹上取点越多，实际轨迹越接近理想轨迹。

例程 7.10：PVT 模式运动

如图 7.25 所示，要控制运动平台按椭圆轨迹运动，但 DMC5X10 系列卡中并没有提供椭圆插补函数，用户可以使用 PVT 函数自己设计该轨迹。

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

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

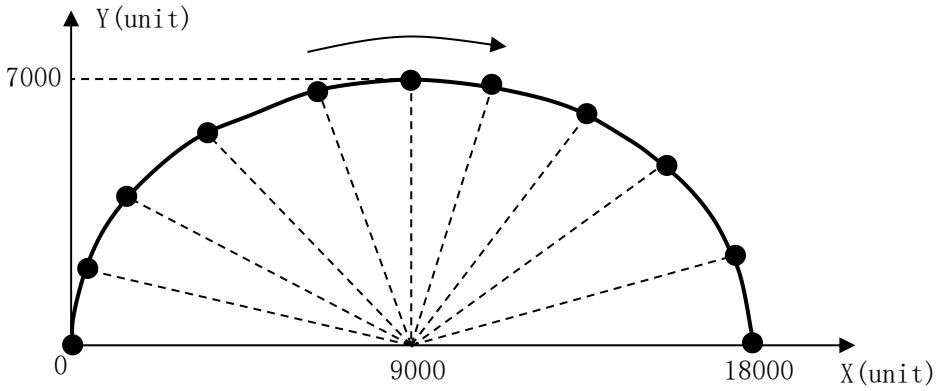


图 7.25　上半椭圆轨迹和分段

使用 PVT 模式运动上半椭圆轨迹的步骤为：

1、将该轨迹分成圆弧角相等的十段轨迹，如图 7.23 所示；计算各点坐标值，即得 P 值。程序如下：

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

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

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

对椭圆方程（7.1）式求导，可得 X、Y 轴方向的速度分量为：

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

上式中 $\dfrac{d\theta}{dt}$ 即为角速度ω。

各点的 X、Y 轴方向的速度分量可由以下程序计算得出。程序如下：

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

for (int i = 0; i < 11; i++)
{
    MyPTimeX[i] = i;               //存储 X 轴各点时间数据
    MyPTimeY[i] = i;               //存储 Y 轴各点时间数据
}
MyWVel = -Math.PI / 10;            //计算角速度
MyPVelX[0] = 0; MyPVelX[10] = 0;   //起始点与终止点 X 轴速度设为 0
MyPVelY[0] = 0; MyPVelY[10] = 0;   //起始点与终止点 Y 轴速度设为 0
for (int i = 1; i < 10; i++)
{
    MyPVelX[i] = -a * Math.Sin((10 - i) * Math.PI / 10) * MyWVel;//计算其他点 X 轴速度
    MyPVelY[i] = b * Math.Cos((10 - i) * Math.PI / 10) * MyWVel;//计算其他点 Y 轴速度
}
```

计算出各点 X、Y 轴的 P、V、T 数据如表 7.14 所示。

表 7.14　PVT 数据

	X 轴			Y 轴		
序号	P(unit)	V(unit/s)	T(s)	P(unit)	V(unit/s)	T(s)
0	0	0	0	0	0	0
1	440	873.731	1	2163	2091.479	1
2	1719	1661.927	2	4115	1779.117	2
3	3710	2287.443	3	5663	1292.603	3
4	6219	2689.048	4	6657	679.560	4
5	9000	2827.431	5	7000	-0.003	5
6	11781	2689.046	6	6657	-679.566	6
7	14290	2287.438	7	5663	-1292.608	7
8	16281	1661.921	8	4114	-1779.120	8
9	17560	873.724	9	2163	-2091.481	9
10	18000	0	10	0	0	10

3、使用 dmc_PvtTable 函数向数据表传递数组数据。

程序如下：

```
ushort MyCardNo, MyCountX, MyCountY;
ushort[] My_AxisList = new ushort[] { 0, 1 };//定义轴列表变量
```

```
MyCardNo = 0;                                          //卡号
MyCountX = 11;                                         //X 轴 11 组数据
MyCountY = 11;                                         //Y 轴 11 组数据
LTDMC.dmc_PvtTable(MyCardNo, My_AxisList[0], MyCountX, MyPTimeX, MyPPosX, MyPVelX);
                                                       // 以 PVT 描述方式向 X 轴传送 PVT 数据
LTDMC.dmc_PvtTable(MyCardNo, My_AxisList[1], MyCountY, MyPTimeY, MyPPosY, MyPVelY);
                                                       //以 PVT 描述方式向 Y 轴传送 PVT 数据
```

4、使用 dmc_pvt_move 函数执行 PVT 运动。

程序如下：

```
ushort My_AxisNum;
My_AxisNum = 2;                                        //参与 PVT 运动的轴数为 2
LTDMC.dmc_PvtMove(MyCardNo, My_AxisNum, new ushort[] { 0, 1 });//启动 PVT 运动
```

执行完上述程序后，得到 PVT 运动轨迹如图 7.26 所示。

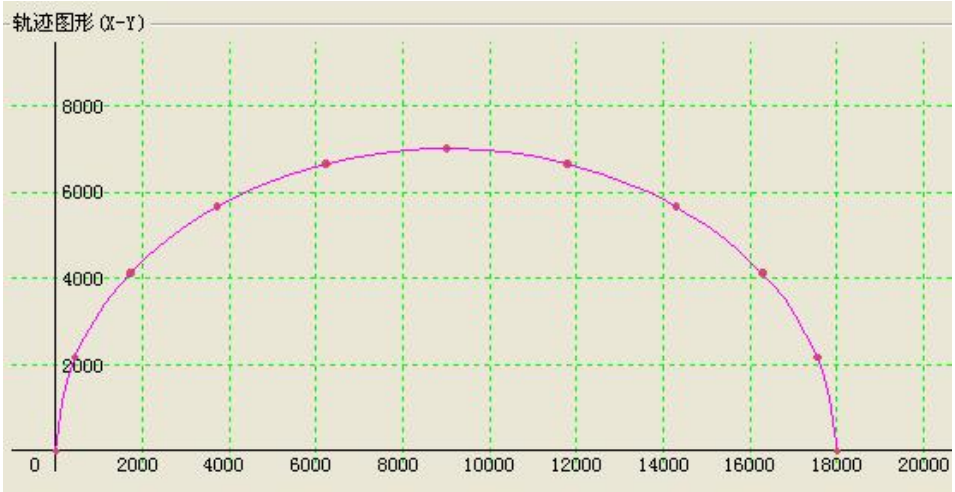


图 7.26　PVT 模式运动得到的上半椭圆轨迹

7.6.2.2　PVTS 运动模式

PVTS 运动模式只需要定义理想轨迹上数据点的位置和时间，以及起点速度和终点速度。运动控制器根据各数据点的位置、时间参数计算各点之间的轨迹位置和速度，确保各段轨迹的速度连续、加速度连续。相关函数如表 7.15 所示。

表 7.15　PVTS 模式运动相关函数说明

名称	功能	参考
dmc_PvtsTable	向指定数据表传送数据，采用 PVTS 描述方式	9.12 节
dmc_PvtMove	启动 PVT 运动	

例程 7.11：PVTS 模式运动

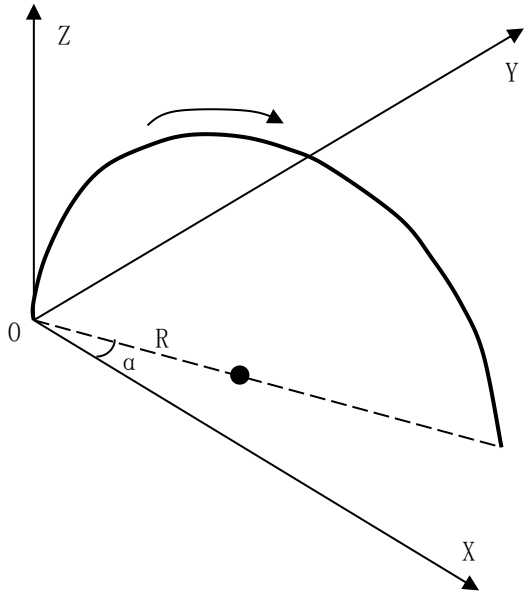


图 7.27　空间圆弧轨迹

如图 7.27 所示，设计空间圆弧轨迹。设其半径 R=15000unit，其映射在 XY 平面上的轨迹与 X 轴的夹角α=π/6，轨迹运动总时间为 10s。该空间圆弧的方程为：

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

若对轨迹上各点速度无精确要求，采用 PVTS 模式设计该轨迹的步骤如下：

1、将该轨迹分成角度相等的十份，计算各点的位置坐标，即 P 值。

2、根据各点的 P 值，计算出各点的 T 值。设每段轨迹运动的时间相等。

3、设定起点速度与终点速度都为 0。

4、使用 dmc_PvtsTable 函数向数据表传递数组数据。

5、使用 dmc_PvtMove 函数执行 PVT 运动。

编写程序如下：

```
……
double[] MyPTime = new double[11];
int[] MyPPosX = new int[11];
int[] MyPPosY = new int[11];
int[] MyPPosZ = new int[11];
ushort MyCardNo, MyCount, R, My_AxisNum;
ushort[] My_AxisList = new ushort[] { 0, 1, 2 };//轴号列表
 double MyPVelBeginX=0, MyPVelEndX=0, MyPVelBeginY=0, MyPVelEndY=0, MyPVelBeginZ=0,
 MyPVelEndZ=0;                                    //各轴起始速度、停止速度设置
```

```
    MyCardNo=0;                                          //卡号
    R = 15000;                                           //定义空间圆弧半径
    MyCount = 11;                                        //设置 X、Y、Z 轴的数据点数
    My_AxisNum = 3;                                      //0、1、2 号轴（即 X、Y、Z 轴）参加 PVT 运动
    for (int i = 0; i < 11; i++)
    {
        MyPPosX[i] =(int)( R * Math.Cos(Math.PI / 6) * Math.Cos((10 - i) * Math.PI / 10) + R * Math.Cos(Math.PI / 6));   //计算 X 轴各点位置坐标
        MyPPosY[i] = (int)(R * Math.Sin(Math.PI / 6) * Math.Cos((10 - i) * Math.PI / 10) + R * Math.Sin(Math.PI / 6));   //计算 Y 轴各点位置坐标
        MyPPosZ[i] = (int)(R * Math.Sin((10 - i) * Math.PI / 10));//计算 Z 轴各点位置坐标
    }
    for (int i = 0; i < 11; i++)
    {
        MyPTime[i] = i;                                  //各点时间坐标
    }
    LTDMC.dmc_PvtsTable(MyCardNo, My_AxisList[0], MyCount, MyPTime, MyPPosX, MyPVelBeginX, MyPVelEndX);   //以 PVTS 描述方式向 X 轴传送 PVT 数据
    LTDMC.dmc_PvtsTable(MyCardNo, My_AxisList[1], MyCount, MyPTime, MyPPosY, MyPVelBeginY, MyPVelEndY);   //以 PVTS 描述方式向 Y 轴传送 PVT 数据
    LTDMC.dmc_PvtsTable(MyCardNo, My_AxisList[2], MyCount, MyPTime, MyPPosZ, MyPVelBeginZ, MyPVelEndZ);   //以 PVTS 描述方式向 Z 轴传送 PVT 数据
    LTDMC.dmc_PvtMove(MyCardNo, My_AxisNum, My_AxisList);//启动 PVT 运动
    ……
```

以 PVTS 模式得到的空间圆弧轨迹如图 7.28 所示。

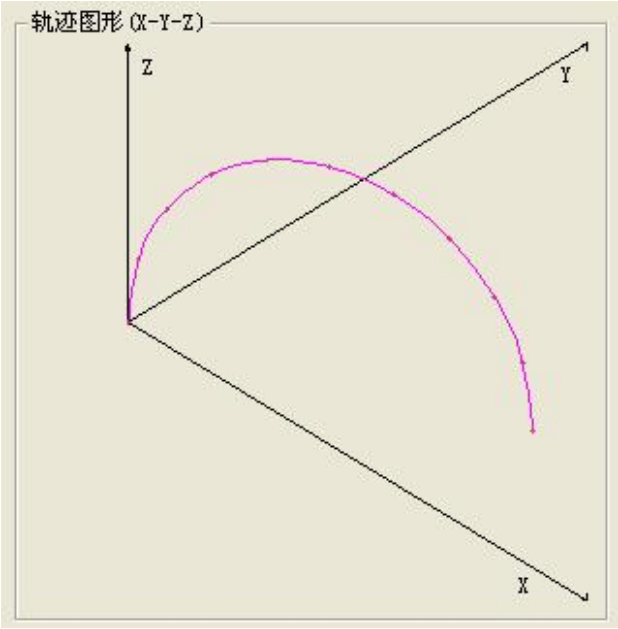


图 7.28　PVTS 模式运动得到的空间圆弧轨迹

7.7 异常减速停止时间设置功能的实现

DMC5X10 系列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 7.16 所示。

表 7.16 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	9.20 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。

例 7.12：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
……
```

7.8 手轮运动功能的实现

7.8.1 单轴手轮运动功能

DMC5X10 系列卡支持单轴手轮运动功能。手轮支持 AB 相四倍频方式和脉冲+方向方式。该功能允许用户设置一个手轮通道对应一个运动轴进行运动。相关函数如表 7.17 所示。

表 7.17 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	9.15 节
dmc_handwheel_move	启动手轮运动	
dmc_set_handwheel_channel	手轮通道选择设置	

注意：1）DMC5410A/5C10 只有低速手轮通道，DMC5610/DMC5810 有高、低速两种手轮通道，手轮外部硬件接口只有一个，通过该功能可以设置这个手轮通道对应的轴号（即每

次只能控制一个轴运动）。

2）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 7.13：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 相四倍频
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
……
```

7.8.2 多轴手轮运动功能

DMC5X10 系列卡支持多轴手轮运动功能。手轮支持 AB 相四倍频方式和脉冲+方向方式。该功能允许用户设置一个手轮通道对应多个运动轴进行运动。相关函数如表 7.18 所示。

表 7.18 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	9.15 节
dmc_handwheel_move	启动手轮运动	
dmc_set_handwheel_channel	手轮通道选择设置	

注 意：1）DMC5410A/5C10 只有一个低速手轮通道,DMC5810/5610 有高、低速两种手轮通道，手轮外部硬件接口只有一个，通过该功能可以设置这个手轮通道对应的轴号（即每次只能控制一个轴运动）。

2）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 7.14：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]=20;                                      //设置各轴的倍率

MyCardNo = 0;                                       //卡号
Myinmode = 0;                                       //手轮输入方式为 AB 相四倍频
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]);//启动手轮运动
……
LTDMC.dmc_emg_stop(MyCardNo);                       //停止手轮运动
……
```

7.9 编码器检测的实现

DMC5X10 系列卡的反馈位置计数器是一个 32 位正负计数器，对通过控制卡编码器接口 EA，EB 输入的脉冲（如编码器、光栅尺反馈脉冲等）进行计数。相关函数如表 7.19 所示。

表 7.19 编码器检测相关函数说明

名称	功能	参考
dmc_set_counter_inmode	设置编码器输入口的计数方式	9.16 节
dmc_get_encoder	读取编码器反馈的脉冲计数值	
dmc_set_encoder	设置编码器的脉冲计数值	

例程 7.15：编码器检测

```
……
ushort MyCardNo, Myaxis, Mymode;
int Myencoder_value, My_Position;;

MyCardNo=0;                                         //卡号
Myaxis=0;                                           //轴号
Mymode=3;                                           //设置编码器的计数方式为 4 倍频，AB 相
Myencoder_value=0;                                  //设置编码器计数初始值为 0
LTDMC.dmc_set_counter_inmode(MyCardNo, Myaxis, Mymode);   //设置编码器的计数方式
LTDMC.dmc_set_encoder(MyCardNo, Myaxis, Myencoder_value);//设置编码器计数初始值为 0
My_Position = LTDMC.dmc_get_encoder(MyCardNo, Myaxis);    //读计数器数值至变量 My_Position
……
```

7.10 检测轴到位状态功能的实现

DMC5X10 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 7.20 所示。

表 7.20 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	9.23 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 7.16：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_set_pulse_outmode(MyCardNo, Myaxis, 0);//设置轴脉冲输出方式
LTDMC.dmc_pmove(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

7.11 通用 I/O 控制的实现

7.11.1 I/O 普通控制功能

用户可以使用 DMC5X10 系列卡上的数字 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 7.21 所示。

表 7.21 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	9.14 节
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 7.17：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 7.18：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
……
```

7.11.2 I/O 延时翻转功能

DMC5X10 系列卡支持 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 7.22 所示。

表 7.22 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	9.14 节

注意：该功能只适用于通用输出端口 OUT0~15。

例 7.19：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
……
```

7.11.3 I/O 计数功能

DMC5C10/DMC5810/5610/5410A 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 7.23 所示。

表 7.23 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	9.14 节
dmc_set_io_count_value	设置 IO 计数值	
dmc_get_io_count_value	读取 IO 计数值	

例 7.20：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 = 0.001;//滤波时间 0.001s
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
……
```

7.12 位置比较功能的实现

DMC5X10 系列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。

7.12.1 一维低速位置比较功能

DMC5X10 系列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小于 1ms。相关函数如表 7.24 所示。

表 7.24 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	9.18 节
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	
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 7.21：一维低速位置比较

```
……
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, 3000, 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(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);
                //设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 50000, 0);
                //定长运动，位移为 50000，相对模式
……
```

运行结果：

当运动到 10000pulse 时，通用输出口 0 电平将翻转；当运动到 30000pulse 时，通用输出口 3 输出高电平。

7.12.2 二维低速位置比较功能

DMC5X10 系列卡提供了一组二维低速位置比较，最多都可以添加 256 个比较点。二维低速位置比较的触发延时时间小于 1ms。相关函数如表 7.25 所示。

表 7.25 二维低速位置比较相关函数说明

名称	功能	参考
dmc_compare_set_config_extern	设置二维位置比较器	9.18 节
dmc_compare_clear_points_extern	清除二维位置比较点	
dmc_compare_add_point_extern	添加二维位置比较点	
dmc_compare_get_current_point_extern	读取当前二维位置比较点位置	
dmc_compare_get_points_runned_extern	查询已经比较过的二维比较点个数	
dmc_compare_get_points_remained_extern	查询可以加入的二维比较点个数	

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

例 7.22：二维低速位置比较

```
……
ushort MyCardNo, Mycrd, Myenable, Mycmp_source, Mydir, Myaction;
uint Myactpara;

MyCardNo = 0;                    //卡号
Mycrd = 0;                       //坐标系号
Myenable = 1;                    //设置比较使能
Mycmp_source = 0;                //比较源：指令位置
```

```
Mydir=1;                                    //比较模式：大于等于
Myaction=3;                                 //触发时 io 取反
Myactpara = 3;                              //3 号输出口执行动作
LTDMC.dmc_compare_set_config_extern(MyCardNo, Myenable, Mycmp_source);//设置二维比较器
LTDMC.dmc_compare_clear_points_extern(MyCardNo);//清除二维位置比较点
LTDMC.dmc_compare_add_point_extern(MyCardNo, new ushort[] { 0, 1 }, new int[] { 5000, 5000 },
new ushort[] { 1, 1 }, Myaction, Myactpara);//添加二维位置比较点
LTDMC.dmc_set_vector_profile_unit(MyCardNo, Mycrd, 0, 3000, 0.01, 0.01, 0);
                                            //设置插补运动速度曲线
LTDMC. dmc_arc_move_center_unit (MyCardNo, Mycrd,2, new ushort[] { 0, 1 }, new double[] { 0, 0 }, new
double[] { 5000, 0 }, 0, 0, 0);             //圆弧插补运动
……
```

7.12.3 一维高速位置比较功能

DMC5X10 系列卡共有四个高速位置比较器，每个高速位置比较器均配有 1 个硬件位置比较输出接口。DMC5X10 系列卡的 CMP0、CMP1 端口电路使用的是普通光耦（速率 4KHz），CMP2、CMP3 端口电路使用的是高速光耦（1MHz）。

每个 CMP 输出端口都可以与任意轴关联，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 7.26 所示。

表 7.26 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	9.19 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 7.23：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;                                     //轴号
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=0;                               //比较源为指令位置
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, 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_add_point(MyCardNo, Myhcmp, 5000);       //添加高速比较位置为 50000 pulse
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);//设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 50000, 0);                //定长运动，位移为 50000，相对模式
……
```

运行结果：

当 2 号轴运动超过位置 50000 pulse 时，CMP0 端口输出低电平并保持。

例 7.24：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;                                      //轴号
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=0;                                //比较源为指令位置
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
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);
                                                    //设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 100000,1);        //定长运动，位移为 100000，绝对模式
……
```

运行结果：

当 0 号轴运动经过位置 10000、30000、70000 pulse 时，CMP0 端口输出低电平，低电平持续时间为 500ms。

例 7.25：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;                                           //轴号
Mycmp_mode = 5;                                     //比较模式为模式 5
Myhcmp=0;                                           //高速比较器，对应硬件 CMP 端口
Mycmp_source=0;                                     //比较源为指令位置
Mycmp_logic=0;                                      //低电平有效
MyTime = 500000;                                    //脉冲宽度 500000us
MyCmpInc=20000;                                     //线性增量 20000pulse
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
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);//设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 100000,1);//定长运动，位移为 100000，绝对模式
……
```

运行结果：

当 0 号轴运动经过位置 10000、30000、50000、70000、90000 pulse 时，CMP1 端口输出低电平，低电平持续时间为 500ms。

7.13 高速位置锁存功能的实现

7.13.1 高速单次位置锁存功能

DMC5X10 系列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位置。相关函数如表 7.27 所示。

表 7.27 高速单次锁存相关函数说明

名称	功能	参考
dmc_set_ltc_mode	设置指定轴的 LTC 信号	9.17 节
dmc_set_latch_mode	设置锁存方式	
dmc_get_latch_value	读取锁存值	
dmc_get_latch_flag	读取已锁存个数	
dmc_reset_latch_flag	复位指定卡的锁存器的标志位	

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 7.26：高速单次位置锁存

```
……
ushort MyCardNo, Myaxis, Myltc_logic, Myltc_mode, Myall_enable;
ushort Mylatch_source, Mytriger_chunnel;
double Myfilter;
int My_latch_Value;

MyCardNo = 0;                                    //卡号
Myaxis = 0;                                      //轴号
Myltc_logic = 0;                                 //设置 LTC 触发方式为下降沿触发
Myltc_mode = 0;                                  //保留参数
Myfilter = 0;                                    //保留参数
Myall_enable = 0;                                //锁存方式为单次锁存
Mylatch_source = 0;                              //锁存源为指令位置
Mytriger_chunnel = 0;                            //保留参数
LTDMC.dmc_set_ltc_mode(MyCardNo, Myaxis, Myltc_logic, Myltc_mode, Myfilter);
                                                 //设置 0 号轴的 LTC 信号
LTDMC.dmc_set_latch_mode(MyCardNo,Myaxis,Myall_enable,Mylatch_source,Mytriger_chunnel);
                                                 //设置锁存方式为单次锁存
LTDMC.dmc_reset_latch_flag(MyCardNo, Myaxis); //复位 0 号轴的锁存状态
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);//设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 100000, 1); //定长运动，位移为 100000，绝对模式
while (LTDMC.dmc_get_latch_flag(MyCardNo, Myaxis) == 0) ;//判断 0 号轴 LTC 锁存状态
```

```
{
    Application.DoEvents();
}
My_latch_Value = LTDMC.dmc_get_latch_value(MyCardNo, Myaxis);
                                    //读取锁存值，并赋值给变量 My_latch_Value
……
```

7.13.2 高速连续位置锁存功能

DMC5X10 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 1ms。

相关函数如表 7.28 所示。

表 7.28 高速连续锁存相关函数说明

名称	功能	参考
dmc_set_ltc_mode	设置指定轴的 LTC 信号	9.17 节
dmc_set_latch_mode	设置锁存方式	
dmc_reset_latch_flag	复位指定卡的锁存器的标志位	
dmc_get_latch_value	读取锁存值	
dmc_get_latch_flag	读取已锁存个数	

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 7.27：高速连续位置锁存

```
……
ushort MyCardNo, Myaxis,i;
long My_latch_Value;

MyCardNo = 0;                                    //卡号
Myaxis = 0;                                     //轴号
i = 0;
int[] My_latch_Value = new int[1000];
LTDMC.dmc_set_ltc_mode(MyCardNo, Myaxis, 0, 0, 0);
                                                 //设置 0 号卡 0 号轴的 LTC 信号，下降沿触发
LTDMC.dmc_set_latch_mode(MyCardNo, Myaxis, 2, 0, 0);
                                                 //设置 0 号轴的锁存源是指令位置，连续锁存
LTDMC.dmc_reset_latch_flag(MyCardNo, Myaxis); //复位 0 号轴的锁存状态
LTDMC.dmc_set_profile(MyCardNo, Myaxis, 1000, 10000, 0.01, 0.01, 1000);//设置梯形速度曲线
LTDMC.dmc_pmove(MyCardNo, Myaxis, 100000, 1); //定长运动，位移为 100000，绝对模式
```

```
while (LTDMC.dmc_check_done(MyCardNo, Myaxis) == 0)
{
    Application.DoEvents();
}
short number= LTDMC.dmc_get_latch_flag(MyCardNo, Myaxis);
if(number > 0)
{
    My_latch_Value = new long[number];
    for (int i = 0; i < number; i++)
    {
        //读取锁存值，并赋值给变量 My_latch_Value
        My_latch_Value[i] = LTDMC. dmc_get_latch_value(MyCardNo, Myaxis);
    }
}
……
```

7.14 轴 IO 映射功能的实现

DMC5X10 系列卡提供了轴 IO 映射功能，该功能允许用户对专用 IO 信号的硬件输入接口进行任意配置，相关函数如表 7.29 所示。

表 7.29 轴 IO 映射相关函数说明

名称	功能	参考
dmc_set_axis_io_map	设置轴 IO 映射关系	9.21 节
dmc_get_axis_io_map	读取轴 IO 映射关系设置	

例 7.28：设置第 0 号卡的第 2 轴原点接口作为第 0 轴的正限位信号

```
……
ushort   CardNo,Axis,IoType, MapIoType, MapIoIndex;
double Filter;

CardNo = 0 ;                          //卡号
Axis = 0;                             //指定轴号：第 0 轴
IoType = 0;                           //指定轴的 IO 信号类型为：正限位信号
MapIoType = 2 ;                       //轴 IO 映射类型：原点信号
MapIoIndex = 2;                       //轴 IO 映射索引号：第 2 轴
Filter = 0.01 ;                     //0.01 秒滤波
LTDMC.dmc_set_axis_io_map(CardNo, Axis, IoType, MapIoType, MapIoIndex, Filter);
```

```
                                        //设置轴 IO 映射
    ……
```

例 7.29：设置第 0 号卡的通用输入口 0 接口作为所有轴的急停信号，并且设置急停信号为低电平有效

```
    ……
    ushort CardNo, Axis, IoType, MapIoType, MapIoIndex;
    double   Filter;

    CardNo = 0;                                    //卡号
    IoType = 3;                                    //指定轴的 IO 信号类型为：急停信号
    MapIoType = 6;                                 //轴 IO 映射类型：通用输入端口
    MapIoIndex = 0;                                //轴 IO 映射索引号：通用输入 0
    Filter = 0.01;                                 //0.01 秒滤波
    for (Axis = 0; Axis <= 7; Axis++)
    {
        LTDMC.dmc_set_axis_io_map(CardNo, Axis, IoType, MapIoType, MapIoIndex, Filter);
                                                   //设置轴 IO 映射关系
    }
    for (Axis = 0; Axis <= 7; Axis++)
    {
        LTDMC.dmc_set_emg_mode(CardNo, Axis, 1, 0);//设置 EMG 信号使能，低电平有效
    }
    ……
```

7.15 原点锁存功能的实现

DMC5X10 系列卡提供了原点锁存功能，该功能可以实现在碰到原点信号时将当前位置锁存，使用该功能可以实现精确回零运动，用户可通过两种模式实现原点锁存回零，方式一：直接设置回零模式为原点锁存模式，调用回零函数实现；方式二：应用层配置锁存参数，调用 vmove 运动实现，方式二实现步骤如下：

1）使用 dmc_set_homelatch_mode 函数设置原点锁存模式；

2）使用 dmc_reset_homelatch_flag 函数清除原点锁存标志；

3）设置轴运动参数，并启动运动；

4）判断原点锁存标志是否有效，有效则停止运动；

5）重新设置轴运动速度为低速度，并使用定长运动到原点锁存位置；

6）回到原点锁存位置后，指令脉冲计数器清零。

相关函数如表 7.30 所示。

表 7.30　原点锁存相关函数说明

名称	功能	参考
dmc_set_homelatch_mode	设置原点锁存模式	9.4 节
dmc_reset_homelatch_flag	清除原点锁存标志	
dmc_get_homelatch_flag	读取原点锁存标志	
dmc_get_homelatch_value	读取原点锁存值	

例 7.30：使用原点锁存功能进行精确回零运动

```
……
ushort MyCardNo, MyAxis, Myenable, Mylogic, Mysource;
int    MyHomelatchValue;

MyCardNo = 0;                                    //卡号
MyAxis = 0;                                      //轴号
Myenable = 1;                                    //使能原点锁存
Mylogic = 0;                                     //锁存方式为下降沿锁存
Mysource = 0;                                    //锁存位置源为指令位置
LTDMC.dmc_set_homelatch_mode(MyCardNo, MyAxis, Myenable, Mylogic, Mysource);
                                                 //设置原点锁存
LTDMC.dmc_reset_homelatch_flag(MyCardNo, MyAxis);//清除原点锁存标志位
LTDMC.dmc_set_profile(MyCardNo, MyAxis, 1000, 10000, 0.1, 0.1, 1000);
                                                 //设置 0 号轴运动参数，最大速度为 10000pulse/s
LTDMC.dmc_vmove(MyCardNo, MyAxis, 0);        //0 号轴执行连续运动，负方向
while (LTDMC.dmc_get_homelatch_flag(MyCardNo, MyAxis) == 0)
{
    Application.DoEvents();                  //判断原点锁存标志位
}
LTDMC.dmc_stop(MyCardNo, MyAxis, 0);         //减速停止
while (LTDMC.dmc_check_done(MyCardNo, MyAxis) == 0)
{
    Application.DoEvents();                  //判断运动状态
}
    LTDMC.dmc_set_profile(MyCardNo, MyAxis, 500, 1000, 0.1, 0.1, 500);
                                                 //设置 0 号轴运动参数，最大速度为 1000pulse/s
MyHomelatchValue = LTDMC.dmc_get_homelatch_value(MyCardNo, MyAxis);
                                                 //获取原点锁存值，并赋值给变量 MyHomelatchValue
LTDMC.dmc_pmove(MyCardNo, MyAxis, MyHomelatchValue, 1);
                                                 //0 号轴定长运动到原点锁存位置，绝对模式
while (LTDMC.dmc_check_done(MyCardNo, MyAxis) == 0)
{
    Application.DoEvents();                  //判断运动状态
```

```
}
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
……
```

7.16 CAN 扩展模块

CAN 扩展模块是 DMC5X10 系列运动控制卡的配套产品，扩展 DMC5X10 系列卡的 IO 端口和 ADDA 端口。通过菊花链的连接方式可以将多个 CAN 扩展模块挂在同一块运动控制卡下面，最多支持连接 8 个 CAN 扩展模块。

CAN 通讯的波特率最大为 1Mbps，通讯状态的刷新频率为 4ms。

DMC5X10 系列卡提供了 CAN 扩展函数，用户通过这些函数的调用可以很方便地实现对扩展端口的操作。一般步骤如下：

1）使用 nmc_set_connect_state 函数进行 CAN 通讯连接；

2）使用 nmc_get_connect_state 函数读取 CAN 通讯状态，如果出现连接异常，则重新执行步骤 1；

3）连接成功后，则可以对 CAN 扩展模块的输入输出端口进行操作。

相关函数如表 7.31 所示。

表 7.31　CAN-IO 相关函数说明

名称	功能	参考
nmc_set_connect_state	设置 CAN 通讯状态	9.22 节
nmc_get_connect_state	读取 CAN 通讯状态	
nmc_write_outbit	设置指定 CAN-IO 扩展模块的某个输出口的电平	
nmc_read_outbit	读取指定 CAN-IO 扩展模块的某个输出口的电平	
nmc_read_inbit	读取指定 CAN-IO 扩展模块的某个输入口的电平	
nmc_write_outport	设置指定 CAN-IO 扩展模块的输出端口的电平	
nmc_read_outport	读取指定 CAN-IO 扩展模块的输出端口的电平	
nmc_read_inport	读取指定 CAN-IO 扩展模块的输入端口的电平	
nmc_set_da_mode	设置指定 CAN-ADDA 扩展模块的某个输出口的模式	
nmc_get_da_mode	读取指定 CAN-ADDA 扩展模块的某个输出口的模式	
nmc_set_ad_mode	设置指定 CAN-ADDA 扩展模块的某个输入口的模式	
nmc_get_ad_mode	读取指定 CAN-ADDA 扩展模块的某个输入口的模式	
nmc_set_da_output	设置指定 CAN-ADDA 扩展模块的某个输出口的电压/电流	
nmc_get_da_output	读取指定 CAN-ADDA 扩展模块的某个输出口的电压/电流	
nmc_get_ad_input	读取指定 CAN-ADDA 扩展模块的某个输入端口的电压/电流	

例 7.31：假设 DMC5810 卡扩展了 1 个 CAN-IO 扩展模块，读取该模块的通用输入口 1 的电平

值，并对该模块的通用输出口 3 置低电平

```
……
ushort CardNo, CANCount, CANStatus, GetCANCount, GetCANStatus，CANBaud ;
ushort CANNum, MyInBitno, MyOutBitno, MyOutLevel;
ushort MyInValue;

CardNo = 0;                          //卡号
CANCount = 1;                        //节点数为 1，因只有 1 个扩展 CAN-IO 模块
CANStatus = 1;                       //通讯状态为连接
CANBaud =0;
GetCANCount = 0;
GetCANStatus = 0;
LTDMC. nmc_set_connect_state (CardNo, CANCount, CANStatus,CANBaud);
                                     //连接 CAN-IO 扩展模块
LTDMC. nmc_get_connect_state (CardNo, ref  GetCANCount, ref  GetCANStatus);
                                     //获取当前 CAN 通讯状态
CANNum = 1;                          //节点号为 1，选择菊花链上的第 1 个扩展 CAN-IO 模块
MyInBitno = 1;                       //通用输入端口 1
LTDMC.nmc_read_inbit(CardNo, CANNum, MyInBitno，ref MyInValue);
                  //获取节点号为 1 的扩展模块上的输入端口 1 的电平，并将该值赋于变量 MyInValue
MyOutBitno = 3;   //通用输出端口 3
MyOutLevel = 0;   //低电平
LTDMC.nmc_write_outbit(CardNo, CANNum, MyOutBitno, MyOutLevel);
                  //控制节点号为 1 的扩展模块上的输出端口 3 输出低电平
……
```

例 7.32：假设 DMC5810 卡扩展了 2 个 CAN-IO 扩展模块，读取节点号 1 扩展模块的通用输入口 0 的电平值，并对节点号 2 扩展模块的通用输出口 7 置低电平

```
……
ushort CardNo, CANCount, CANStatus, GetCANCount, GetCANStatus,CANBaud;
ushort CANNum, MyInBitno,MyOutBitno, MyOutLevel;
ushort   MyInValue;

CardNo = 0;                          //卡号
CANCount = 2;                        //节点数为 2，因有 2 个扩展 CAN-IO 模块
GetCANCount = 0;
GetCANStatus = 0;
CANBaud =0;
CANStatus = 1;                       //通讯状态为连接
LTDMC. nmc_set_connect_state (CardNo, CANCount, CANStatus,CANBaud );
                                     //连接 CAN-IO 扩展模块
LTDMC. nmc_get_connect_state (CardNo, ref  GetCANCount, ref  GetCANStatus);
```

```
                                        //获取当前 CAN 通讯状态
CANNum = 1;                              //节点号为 1，选择菊花链上的第 1 个扩展 CAN-IO 模块
MyInBitno = 0;                          //通用输入端口 0
LTDMC.nmc_read_inbit(CardNo, CANNum, MyInBitno，ref MyInValue);
                    //获取节点号为 1 的扩展模块上的输入端口 0 的电平，并将该值赋于变量 MyInValue
CANNum = 2;      //节点号为 2，选择菊花链上的第 2 个扩展 CAN-IO 模块
MyOutBitno = 7;//通用输出端口 7
MyOutLevel = 0;//低电平
LTDMC.nmc_write_outbit(CardNo, CANNum, MyOutBitno, MyOutLevel);
                    //控制节点号为 2 的扩展模块上的输出端口 7 输出低电平
……
```

例 7.33：假设 DMC5810 卡扩展了 2 个 CAN-IO 扩展模块，读取 1 号模块的通用输入口 1 的电平值，并对 2 号模块的通用输出口 23 置低电平

```
……
short err=0;
ushort CardNo, CANCount, CANStatus, GetCANCount, GetCANStatus,CANBaud ;
ushort CANNum;
ushort MyInBitno, MyOutBitno, MyOutLevel, MyInValue;

CardNo = 0;                              //卡号
CANCount = 2;                            //节点数为 2，因有 2 个扩展 CAN-IO 模块
CANStatus = 0;                          //通讯状态为连接
CANBaud = 0;                             //波特率 1000Kbps
GetCANCount = 0;
GetCANStatus = 0;

err = LTDMC.nmc_set_connect_state(CardNo, CANCount, CANStatus, CANBaud);
                                        //连接 CAN-IO 扩展模块
err = LTDMC.nmc_get_connect_state(CardNo, ref    GetCANCount, ref    GetCANStatus);
                                        //获取当前 CAN 通讯状态
CANNum = 1;                              //节点号为 1，选择菊花链上的 1 号扩展 CAN-IO 模块
MyInBitno = 1;                          //通用输入端口 1
MyInValue = 0;
err = LTDMC.nmc_read_inbit(CardNo, CANNum, MyInBitno, ref MyInValue);
                    //获取节点号为 1 的扩展模块上的输入端口 1 的电平，并将该值赋变量 MyInValue
CANNum = 2;                              //节点号为 2，选择菊花链上的 2 号扩展 CAN-IO 模块
MyOutBitno = 23;   //通用输出端口 23
MyOutLevel = 0;   //低电平
err = LTDMC.nmc_write_outbit(CardNo, CANNum, MyOutBitno, MyOutLevel);
                    //控制节点号为 2 的扩展模块上的输出端口 23 输出低电平
……
```

例 7.34：假设 DMC5810 卡扩展了 2 个 CAN-ADDA 扩展模块，模块的输入输出均为电压模式，读取节点号 1 扩展模块的 AD 输入口 1 的电平值，并对节点号 2 扩展模块的 DA 输出口 1 置 2V 电平

```
……
short err = 0;
ushort CardNo, CANCount, CANStatus, GetCANCount, GetCANStatus, CANBaud;
ushort CANNum;
ushort MyADno, MyDAno;
Int32 MyDAValue, MyADValue;

CardNo = 0;                          //卡号
CANCount = 2;                        //节点数为 2，因有 2 个扩展 CAN 模块
GetCANCount = 0;
GetCANStatus = 0;
CANBaud =0;                          //波特率 1000Kbps
CANStatus = 1;                       //通讯状态为连接
LTDMC.nmc_set_connect_state(CardNo, CANCount, CANStatus,CANBaud );
                                     //连接 CAN 扩展模块
LTDMC.nmc_get_connect_state(CardNo, ref    GetCANCount, ref    GetCANStatus);
                                     //获取当前 CAN 通讯状态
CANNum = 1;                          //节点号为 1，选择菊花链 i 上的 1 号扩展 CAN 模块
MyADno = 1;                          //AD 输入端口 1
MyADValue = 0;
err = LTDMC.nmc_get_ad_output(CardNo, CANNum, MyADno, ref MyADValue);
          //获取节点号为 1 的扩展模块上的输入端口 1 的电平，并将该值赋于变量 MyInValue
CANNum = 2;                          //节点号为 2，选择菊花链上的 2 号扩展 CAN 模块
MyDAno = 1;   //DA 输出端口 1
MyDAValue = 2000;   //2V 输出
err = LTDMC.nmc_set_da_output(CardNo, CANNum, MyDAno, MyDAValue);
          //控制节点号为 2 的扩展模块上的输出端口 1 输出 2V
```

7.17 点位运动综合例程

如图 7.29 所示，此例程为单轴点位运动例程。控制电机运动的方式分为模拟按钮及外部 IO 输入控制两种方式；选择模拟按钮时，即通过软件界面按钮控制电机正反转；选择外部 IO 输入时，即通过外部硬件 IO 输入来控制电机正反转。在输入参数框里，可以设定电机轴号、S 形曲线参数、外部 IO 输入的端口号及每次运行的距离。在显示框里，可以监控此时电机的速度、位置及电机状态。此外，还添加了两个定时器控件，定时器不仅负责读取当前速度、位置、电机状态并显示在显示框中，而且还负责选择外部 IO 输入时，监测外部 IO 输入的状态，当读取的电平为有效电平时控制电机转动。

图 7.29 单轴点位运动界面

如果选用 0 号轴（即硬件接口为第 0 轴电机信号接口）控制步进驱动器，正转信号通过外部输入 IO 端口 1 输入，反转信号通过外部输入 IO 端口 2 输入，正转状态通过外部输出 IO 端口 1 输出，反转状态通过外部输出 IO 端口 2 输出。其硬件接线图如图 7.30 所示。

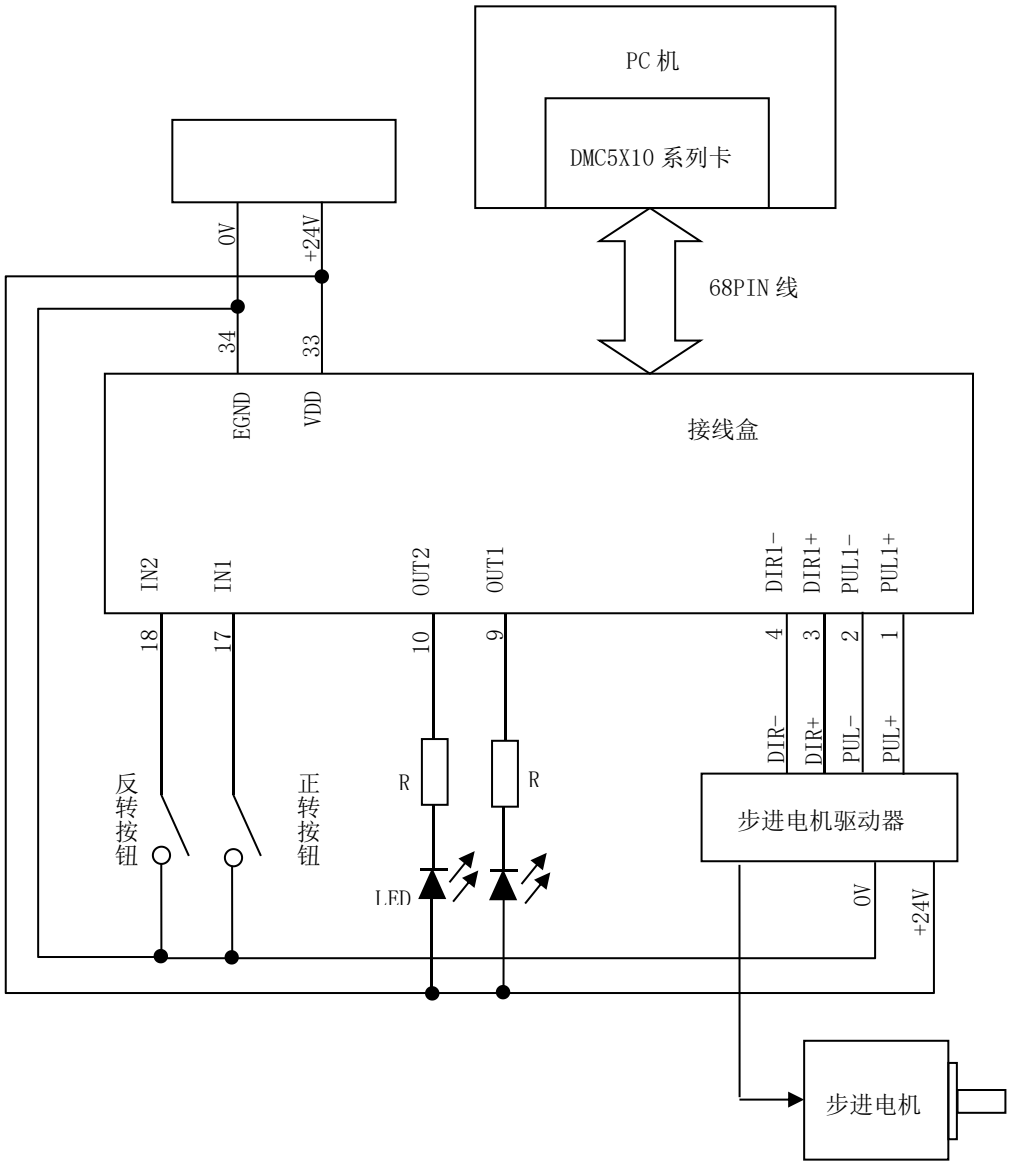


图 7.30　点位运动硬件接线图

编写 C＃代码如下：

```
//定义全局变量
    short MyCardNum;                                    //定义卡数
    double MyFirSpe, MyRunSpe, MyStopSpe, MyTacc, MyTdec, MyTsacc;
                    //定义起始速度、最大速度、停止速度、加速时间、减速时间，S 段时间
    Int32 MyRunDis;                                     //定义位移为、改变位置
    ushort    MyRunMode, bitno,abitno;                  //定义运动方向，位移模式,输入输出口号
    ushort    MyAxis, MyCardNO,    MyCardCount, ForTurnOut, RevTurnOut;
                                                        //判断为 T 形或 S 形曲线,轴号,卡号,卡数
    UInt32[] MyCardTypeArr = new UInt32[8];     //定义固件类型
    ushort[] MyCardID = new ushort[8];              //定义卡号

//窗口装载函数中对控制卡进行初始化
```

```
private void Form1_Load(object sender, EventArgs e)//窗口装载函数
    {
        GetCtrValue_Init();                                    //获取文本框值子程序
        MyCardNum = LTDMC.dmc_board_init();                    //控制卡初始化
        MyCardCount = Convert.ToUInt16(MyCardNum);//控制卡卡数
        LTDMC.dmc_get_CardInfList(ref MyCardCount, MyCardTypeArr, MyCardID);
                                                               //获取控制卡硬件 ID 号
        MyCardNO = MyCardID[0];
        if (MyCardNum < 1 || MyCardNum > 8)                    //如果当前卡数<1 或>8
        {
            if (MessageBox.Show("初始化运动控制卡失败！是否继续运行？", "提示",
        MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk) == DialogResult.No)
            {
                Application.Exit();                            //退出程序
            }
        }
        else
        {
            MessageBox.Show("初始化卡成功，当前选择的卡号为：" + Convert.ToString(MyCardNO,
        10).ToUpper());
        }
        timer1.Enabled = true;
    }

//窗口卸载函数中对控制卡进行释放
private void Form1_close(object sender, EventArgs e)//窗口卸载程序
    {
        LTDMC.dmc_board_close();                               //关闭运动控制卡
    }

//获取文本框值程序
private void GetCtrValue_Init()                                //获取文本框值
    {
        MyAxis = Convert.ToUInt16(Axis.Text);                  //设置轴号
        MyFirSpe = Convert.ToDouble(FirSpe.Text);
        MyRunSpe = Convert.ToDouble(RunSpe.Text);
        MyTacc = Convert.ToDouble(Tacc.Text);
        MyTdec = Convert.ToDouble(Tdec.Text);
        MyStopSpe = Convert.ToDouble(StopSpe.Text);
        MyTsacc = Convert.ToDouble(Tsacc.Text);
        MyRunDis = Convert.ToInt32(RunDis.Text);
        MyRunMode = Convert.ToUInt16(mode.Text);               //获取各参数值
        ForTurnOut = Convert.ToUInt16(MyForTurnOut.Text.Trim());//控制正转输出口
        RevTurnOut = Convert.ToUInt16(MyRevTurnOut.Text.Trim());//控制反转输出口
```

```
    bitno = Convert.ToUInt16(MyForTurnNum.Text.Trim());        //控制正转输入口
    abitno = Convert.ToUInt16(MyRevTurnNum.Text.Trim());       //控制反转输入口
}

//获取各文本框参数值程序
private void Axis_TextChanged_1(object sender, EventArgs e)
{
    if (Axis.Text != "")
    {
        try
        {
            MyAxis = Convert.ToUInt16(Axis.Text.Trim());//设置轴号
        }
        catch (Exception)
        { }
    }
}

private void FirSpe_TextChanged(object sender, EventArgs e)
{
    if (FirSpe.Text != "")
    {
        try
        {
            MyFirSpe = Convert.ToDouble(FirSpe.Text.Trim());//起始速度
        }
        catch (Exception)
        { }
    }
}

private void StopSpe_TextChanged(object sender, EventArgs e)
{
    if (StopSpe.Text != "")
    {
        try
        {
            MyStopSpe = Convert.ToDouble(StopSpe.Text.Trim());//停止速度
        }
        catch (Exception)
        { }
    }
}
```

```
private void RunSpe_TextChanged(object sender, EventArgs e)
    {
        if (RunSpe.Text != "")
        {
            try
            {
                MyRunSpe = Convert.ToDouble(RunSpe.Text.Trim());//运行速度
            }
                catch (Exception)
                { }
        }
    }

private void Tacc_TextChanged(object sender, EventArgs e)
    {
        if (Tacc.Text != "")
        {
            try
            {
                MyTacc = Convert.ToDouble(Tacc.Text.Trim());//加速时间
            }
                catch (Exception)
                  { }
        }
    }

private void Tdec_TextChanged(object sender, EventArgs e)
{
        if (Tdec.Text != "")
        {
            try
            {
                MyTdec = Convert.ToDouble(Tdec.Text.Trim());//减速时间
            }
                catch (Exception)
                { }
        }
    }

private void Tsacc_TextChanged(object sender, EventArgs e)
    {
        if (Tsacc.Text != "")
        {
            try
```

```
            {
                MyTsacc = Convert.ToDouble(Tsacc.Text.Trim());//S 段时间
            }
                catch (Exception)
                { }
        }
    }

private void RunDis_TextChanged(object sender, EventArgs e)
    {
        if (RunDis.Text != "")
        {
            try
            {
                MyRunDis = Convert.ToInt32(RunDis.Text.Trim());//运行距离
            }
                catch (Exception)
                { }
        }
    }

private void mode_TextChanged(object sender, EventArgs e)
    {
        if (mode.Text != "")
        {
            try
            {
                MyRunMode = Convert.ToUInt16(mode.Text.Trim());//位移模式
            }
                catch (Exception)
                { }
        }
    }

private void MyForTurnOut_TextChanged(object sender, EventArgs e)
    {
        if (MyForTurnOut.Text != "")
        {
            try
            {
                ForTurnOut = Convert.ToUInt16(MyForTurnOut.Text.Trim());//控制正转输出口
            }
                catch (Exception)
            { }
```

```
            }
        }

private void MyRevTurnOut_TextChanged_1(object sender, EventArgs e)
    {
        if (MyRevTurnOut.Text != "")
        {
            try
            {
                RevTurnOut = Convert.ToUInt16(MyRevTurnOut.Text.Trim());//控制反转输出口
            }
                catch (Exception)
                { }
        }
    }

private void MyForTurnNum_TextChanged(object sender, EventArgs e)
    {
        if (mode.Text != "")
        {
            try
            {
                bitno = Convert.ToUInt16(MyForTurnNum.Text.Trim());//控制正转输入口
            }
                catch (Exception)
                { }
        }
    }

private void MyRevTurnNum_TextChanged(object sender, EventArgs e)
    {
        if (MyRevTurnNum.Text != "")
        {
            try
            {
                abitno = Convert.ToUInt16(MyRevTurnNum.Text.Trim());//控制反转输入口
            }
                catch (Exception)
                { }
        }
    }
//电机正转按钮程序
private void MyForTurn_Click(object sender, EventArgs e)//电机正转
    {
```

```
        ForTurn();
    }
private void ForTurn()
    {
        if (LTDMC.dmc_check_done(MyCardNO, MyAxis) == 0)
        {
            MessageBox.Show("电机正在运动，请稍候再尝试！");
        }
        else
        {
            LTDMC.dmc_set_pulse_outmode(MyCardNO, MyAxis, 0);
            LTDMC.dmc_set_s_profile(MyCardNO, MyAxis, 0, MyTsacc);//S 形速度曲线参数
            LTDMC.dmc_set_profile(MyCardNO, MyAxis, MyFirSpe, MyRunSpe, MyTacc, MyTdec,
            MyStopSpe);                                          //单轴速度曲线参数
            LTDMC.dmc_set_s_profile(MyCardNO, MyAxis, 0, MyTsacc);//S 形速度曲线参数
            LTDMC.dmc_pmove(MyCardNO, MyAxis, MyRunDis, MyRunMode);
            LTDMC.dmc_write_outbit(MyCardNO, ForTurnOut, 0);//设置正转输出口低电平
      }

//电机反转按钮程序
private void RevTurn_Click(object sender, EventArgs e)//电机反转
    {
        backTurn();
    }
private void backTurn()
    {
        if (LTDMC.dmc_check_done(MyCardNO, MyAxis) == 0)
        {
           MessageBox.Show("电机正在运动，请稍候再尝试！");
        }
        else
        {
            LTDMC.dmc_set_pulse_outmode(MyCardNO, MyAxis, 0);
            LTDMC.dmc_set_s_profile(MyCardNO, MyAxis, 0, MyTsacc);//S 形速度曲线参数
            LTDMC.dmc_set_profile(MyCardNO, MyAxis, MyFirSpe, MyRunSpe, MyTacc, MyTdec,
            MyStopSpe);                                          //单轴速度曲线参数
            LTDMC.dmc_set_s_profile(MyCardNO, MyAxis, 0, MyTsacc);//S 形速度曲线参数
            LTDMC.dmc_pmove(MyCardNO, MyAxis, -MyRunDis, MyRunMode);
            LTDMC.dmc_write_outbit(MyCardNO, RevTurnOut, 0);       //设置反转输出口低电平
        }
      }
//定时器程序
private void timer1_Tick(object sender, EventArgs e)//电机正反转及显示灯控制
    {
```

```
        if (LTDMC.dmc_check_done(MyCardNO, MyAxis) == 0)//电机正反转及显示灯控制
        {
            if (LTDMC.dmc_read_outbit(MyCardNO, ForTurnOut ) == 0)//如果正转输出口低电平
            {
                pictureBox2.BackColor = Color.Green;                    //正转显示灯绿色
            }
            else                                                        //如果正转输出口高电平
            {
            pictureBox2.BackColor = Color.Red;                          //正转显示灯红色
            }
            if (LTDMC.dmc_read_outbit(MyCardNO, RevTurnOut) == 0) //如果反转输出口低电平
            {
                pictureBox1.BackColor = Color.Green;                    //反转显示灯绿色
            }
            else                                                        //如果反转输出口高电平
            {
                pictureBox1.BackColor = Color.Red;                      //反转显示灯红色
            }
            }
        else
        {
            LTDMC.dmc_write_outbit(MyCardNO, ForTurnOut, 1);       //正转输出口写高电平
            LTDMC.dmc_write_outbit(MyCardNO, RevTurnOut, 1);       //反转输出口写高电平
            pictureBox2.BackColor = Color.Red;                     //正转显示灯红色
            pictureBox1.BackColor = Color.Red;                     //反转显示灯红色
        }
      Myvel.Text = LTDMC.dmc_read_current_speed(MyCardNO, MyAxis).ToString ();//读取当前速度
      Mypos.Text=LTDMC.dmc_get_position(MyCardNO, MyAxis).ToString();              //读取当前位置
    }

//模拟按钮程序
private void bt_CheckedChanged(object sender, EventArgs e)
    {
        MyForTurn.Visible = true ;                                          //显示正转按钮
        RevTurn.Visible = true ;                                            //显示反转按钮
    }

//IO 按钮程序
private void IO_CheckedChanged(object sender, EventArgs e)
    {
        MyForTurn.Visible = false ;                 //隐藏按钮
        RevTurn.Visible = false ;
        LTDMC.dmc_read_inbit(MyCardNO, bitno);//输入口控制电机正转
        if (LTDMC.dmc_read_inbit(MyCardNO, bitno) == 0)
```

```
        {
            ForTurn();                                  //执行电机正转程序
        }
        else
        {
           pictureBox2.BackColor = Color.Red;      //pictureBox2 显示红色
        }
        LTDMC.dmc_read_inbit(MyCardNO, abitno);//输入口控制电机反转
        if (LTDMC.dmc_read_inbit(MyCardNO, abitno) == 0)
        {
            backTurn();                                  //执行电机反转程序
        }
        else
        {
            pictureBox1.BackColor = Color.Red;//pictureBox1 显示红色
        }
    }

//清零按钮程序
private void MyResetPos_Click(object sender, EventArgs e)
    {
        LTDMC.dmc_set_position(MyCardNO, MyAxis, 0);//当前位置设为零点
    }

//退出按钮程序
private void myExit_Click(object sender, EventArgs e)
    {
        Application.Exit();                              //退出程序
    }
```

第 8 章　基于脉冲当量的高级功能实现方法

本章介绍采用 C＃语言通过调用相关函数实现 DMC5X10 系列卡基于脉冲当量的高级功能方法。

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

8.1　板卡初始化及脉冲输出模式的设置

详见 7.1 节 板卡初始化及脉冲输出模式的设置。

8.2　脉冲当量的设置

DMC5X10 系列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 8.1 所示。

表 8.1　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	10.1 节

例 8.1：某设备中运动控制卡每发送 100pulse，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 100pulse/mm。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 100pulse/mm
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 8.2：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
……
```

8.3 回原点运动的实现

DMC5X10 系列卡共提供了 13 种回原点方式。关于回零方式介绍详见 7.3.2 节 回原点方式。

基于脉冲当量的回原点运动主要步骤如下：

1）使用 dmc_set_home_pin_logic 函数设置原点开关的有效电平；

2）使用 dmc_set_homemode 函数设置回原点方式；

3）设置脉冲当量值；

4）设置 dmc_set_profile_unit 回原点运动的速度曲线；

5）使用 dmc_home_move 函数执行回原点运动；

6）回到原点后，使用 dmc_get_home_result 读取回零结果，正常完成，指令脉冲计数器自动清零。

相关函数如表 8.2 所示。

表 8.2 基于脉冲当量的回零运动相关函数说明

名称	功能	参考
dmc_set_home_pin_logic	设置原点信号的有效电平	9.3 节
dmc_set_homemode	选择回原点模式	
dmc_home_move	按指定的方向和速度方式开始回原点	
dmc_get_home_result	读取回零状态	

例程 8.3：方式 1 低速回原点

```
……
MyCardNo=0;                 //卡号
Myaxis=0;                   //轴号
Myorg_logic=0;              //原点信号有效电平为低电平有效
Myfilter=0;                 //保留参数
Myhome_dir=1;               //回零方向为正向
```

```
Myvel_mode = 0;              //回零模式为低速
Mymode=0;                    //回零方式为一次回零
MyEZ_count=0;                //保留参数
LTDMC.dmc_set_home_pin_logic(MyCardNo, Myaxis, Myorg_logic, Myfilter);
                             //设置原点信号
LTDMC.dmc_set_homemode(MyCardNo, Myaxis, Myhome_dir, Myvel_mode, Mymode, MyEZ_count);
                             //设置回原点模式
LTDMC.dmc_set_profile_unit(MyCardNo, Myaxis, 500, 1000, 0.1, 0.1, 500);
                             //设置 0 号轴梯形速度曲线参数
LTDMC.dmc_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();
}
```

8.4 点位运动的实现

当执行基于脉冲当量的高级运动指令时，其距离或位置单位为 unit；速度单位为 unit/s。

DMC5X10 系列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 8.3 所示。

表 8.3　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	10.3 节
dmc_set_s_profile	设置单轴速度曲线 S 段参数值	9.8 节
dmc_pmove_unit	定长运动	10.3 节
dmc_check_done	检测指定轴的运动状态	9.7 节

说明：执行 dmc_pmove_unit 时，将完全按照用户设置的速度曲线进行点位运动。

例程 8.4：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_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 7.9。若在减速时改变目标位置，电机的速度变化曲线参见图 7.10。

实现这 2 个功能的函数如表 8.4 所示。

表 8.4　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	10.3 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 8.5：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_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
}
……
```

8.5 连续运动的实现

连续运动模式中，DMC5X10 系列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启动时的速度曲线减速停止。

相关函数如表 8.5 所示。

表 8.5 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	9.9 节
dmc_stop	指定轴停止运动	9.7 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。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参见图 7.12 和图 7.13。

例程 8.6：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_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
}
……
```

8.6 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。

8.6.1 直线插补运动

DMC5410A 卡可以进行任意 2~4 轴直线插补，DMC5610 卡可以进行任意 2~6 轴直线插补，DMC5810 卡可以进行任意 2~8 轴直线插补，DMC5C10 卡可以进行任意 2~12 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 8.1 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：

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

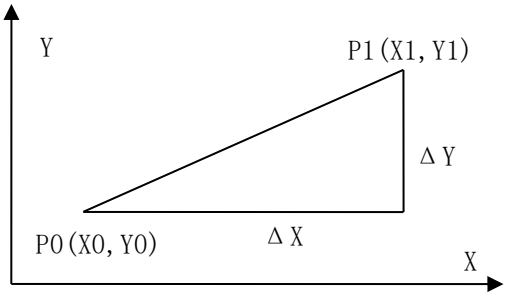


图 8.1 两轴直线插补

2. 三轴直线插补

如图 8.2 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：

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

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：

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

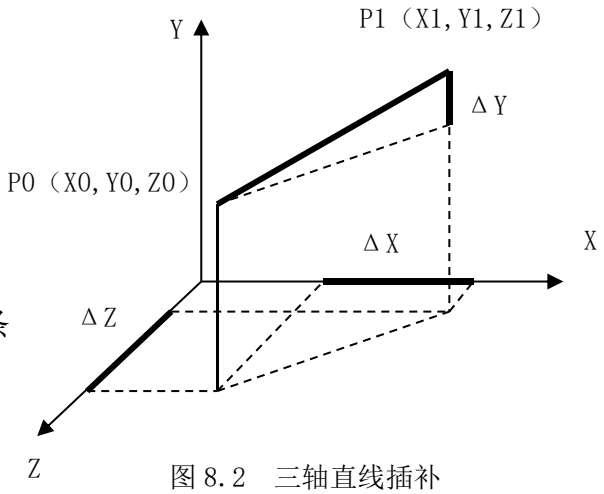


图 8.2 三轴直线插补

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。

DMC5X10 系列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、矩形插补等。此外，当插补轴数大于 3 时，DMC5X10 系列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 8.6 所示。

表 8.6 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	10.4 节

dmc_set_vector_s_profile	设置插补运动速度曲线的平滑时间	
dmc_line_unit	直线插补运动	10.5 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_rectangle_move_unit	矩形插补运动	

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 2~12 轴，连续插补模式下可支持 2~6 轴）。

例程 8.7： 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_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 8.8： 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_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();
}
……
```

8.6.2 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。

8.6.2.1 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 8.9：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();
}
……
```

8.6.2.2 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 8.3 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。

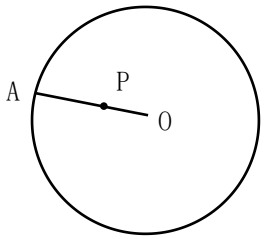


图 8.3 螺旋线插补基本原理

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 8.10：两轴绽放螺旋线插补

```
……
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_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 8.4 所示。

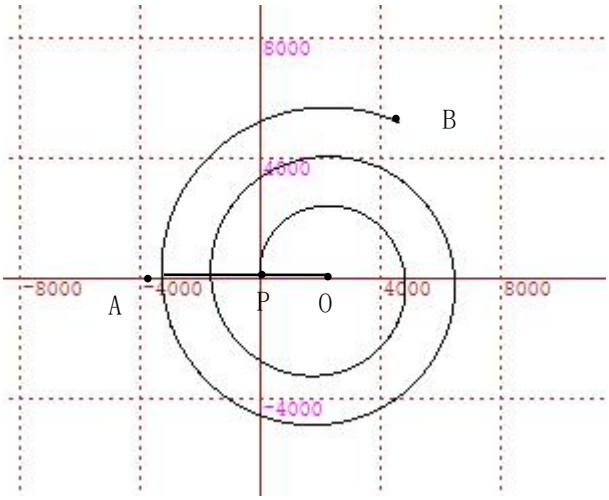


图 8.4　两轴绽放螺旋线插补

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 8.11：两轴收敛螺旋线插补

```
……
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_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 8.5 所示。

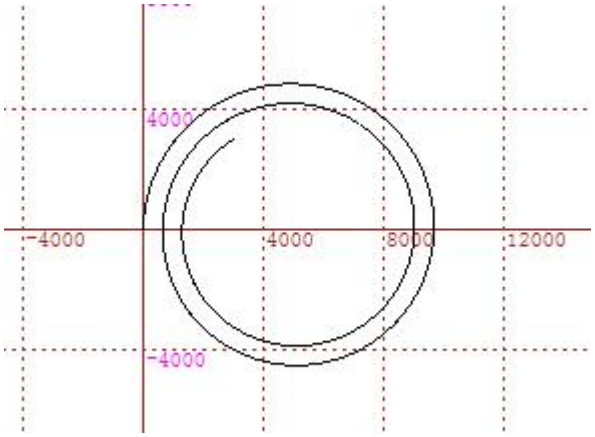


图 8.5　两轴收敛螺旋插补

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。

8.6.2.3　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 10.5 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 8.12：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_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 8.6 所示。

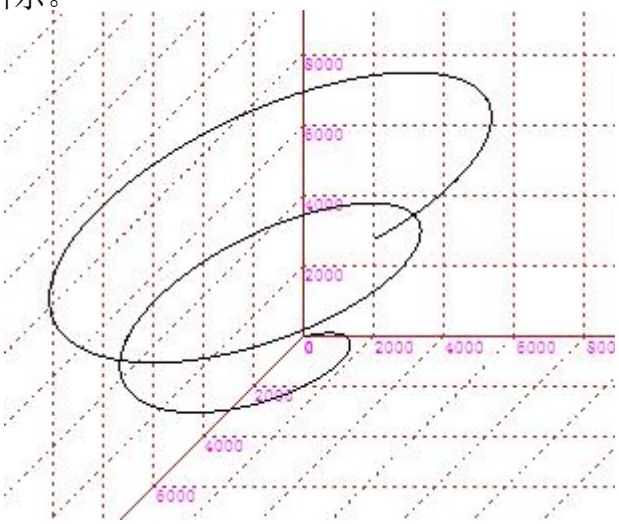


图 8.6　绽放螺旋插补

基于 XY 平面的运动轨迹如图 8.7 所示。

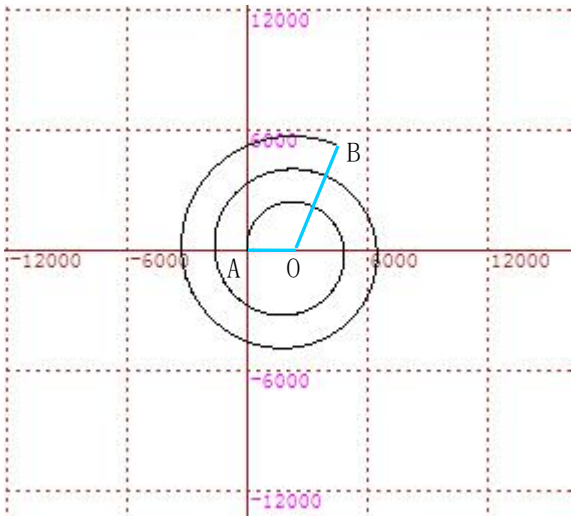


图 8.7　绽放螺旋插补基于 XY 平面的运动轨迹

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 10.5 节相关函数说明）。

例程 8.13：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_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 8.8 所示。

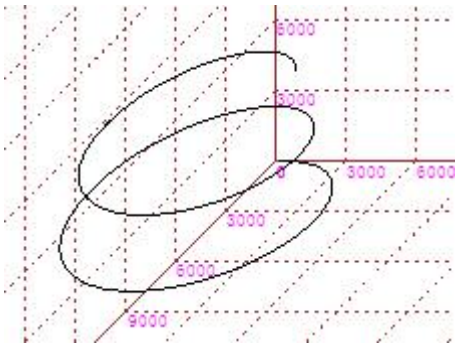


图 8.8　空间收敛螺旋线插补

基于 XY 平面的运动轨迹如图 8.9 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。

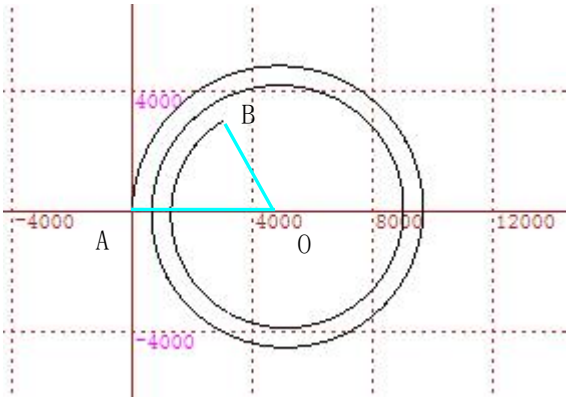


图 8.9　空间收敛螺旋线插补基于 XY 平面的运动轨迹

例程 8.14：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_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 8.10 所示。

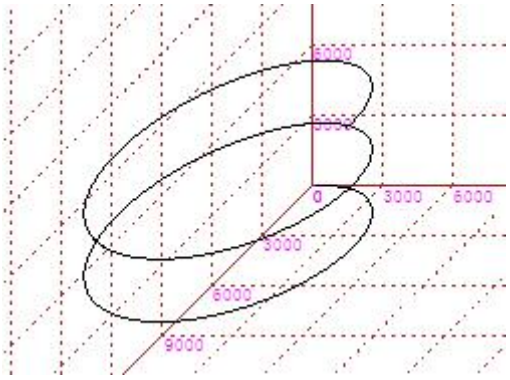


图 8.10　圆柱螺旋插补

基于 XY 平面的运动轨迹如图 8.11 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。

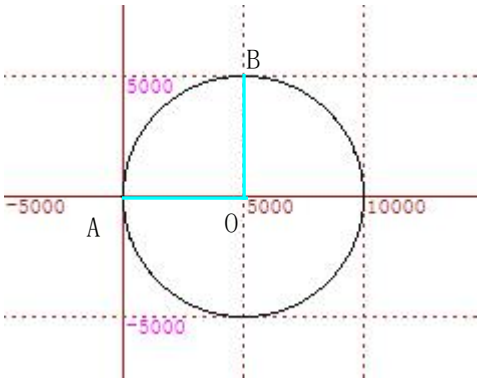


图 8.11　圆柱螺旋线插补基于 XY 平面的运动轨迹

8.6.2.4　基于圆心圆弧的同心圆插补

当圆弧圈数为负数时，函数 dmc_arc_move_center_unit 可执行两轴同心圆插补运动。

例程 8.15：XY 轴同心圆插补

```
……
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 = -3;                                   //设置同心圆圈数为 4
Myposi_mode = 0;                                 //设置圆弧插补模式为相对坐标模式
```

```
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 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);
                                                    //执行同心圆插补运动
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 8.12 所示。

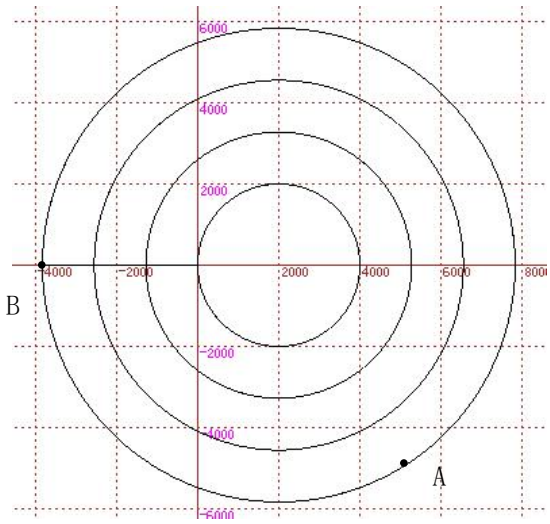


图 8.12　同心圆插补

在同心圆插补中，设置的终点坐标在最后一圈圆弧上，同时为了满足最后一圈轨迹为整圆，实际终点位置（即最终的停止位置）为最后一圈圆弧的起点位置。如图 8.12 所示，设置的终点坐标为点 A（5000，-5000），实际的终点位置为点 B，同时点 B 为最后一圈的起点，点 A 在最后一圈圆弧上。

8.6.2.5　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 8.16：六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_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();
}
……
```

8.6.3 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。

8.6.3.1 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 8.17：两轴圆弧插补

```
……
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_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 8.13 所示。

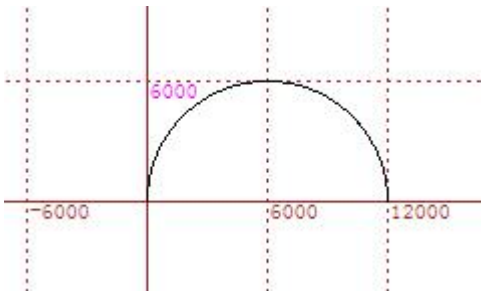


图 8.13　两轴圆弧插补

8.6.3.1　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 8.6.2.5 节插补模式下的辅助轴跟随）。

例程 8.18：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_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 8.14 所示。

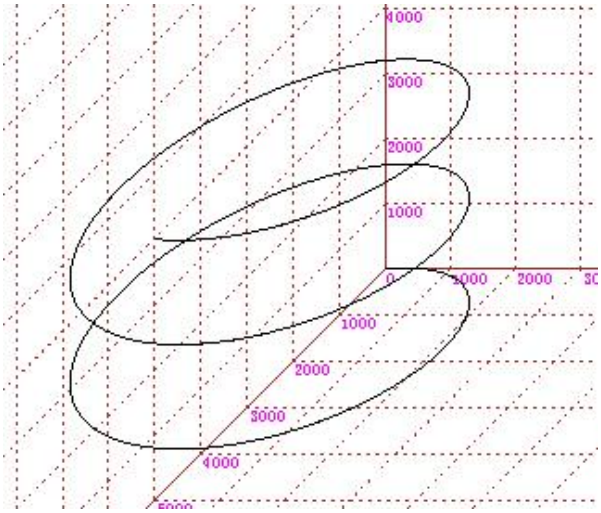


图 8.14　圆柱螺旋线插补

8.6.4　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 8.6.2.5 节插补模式下的辅助轴跟随）

8.6.4.1　基于三点圆弧的空间圆弧插补

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 10.5 节插补运动）。

例程 8.19：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_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 8.15 所示。

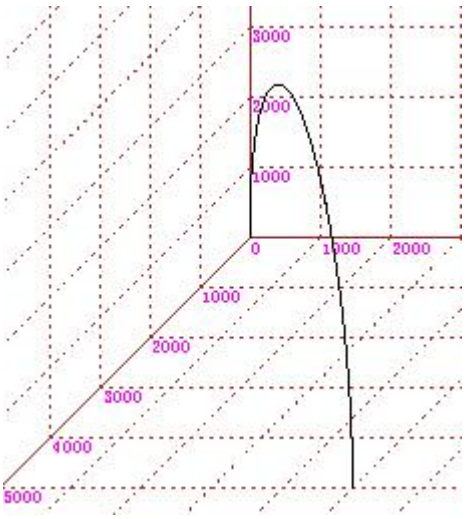


图 8.15　空间圆弧插补

8.6.4.2　基于三点圆弧的空间圆弧插补

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自然数时，函数可 dmc_arc_move_3points_unit 实现空间圆弧插补（参数设置规则详见 10.5 节插补运动）。

例 8.20：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_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 8.16 所示。

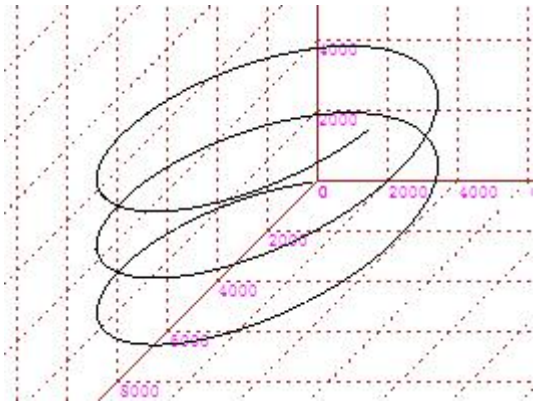


图 8.16　空间圆弧插补

8.6.5　矩形插补运动

dmc_rectangle_move_unit 函数可以执行两轴矩形插补运动，包括逐行模式与渐开线模式。矩形插补运动支持前瞻模式和非前瞻模式(dmc_conti_set_lookahead_mode)，前瞻模式矩形拐角会平滑处理变成圆弧，无法到达矩形端点；非前瞻模式则精确到达矩形端点。一般建议运行矩形插补运动前，调用前瞻函数关闭前瞻模式。

8.6.5.1　逐行模式的矩形插补运动

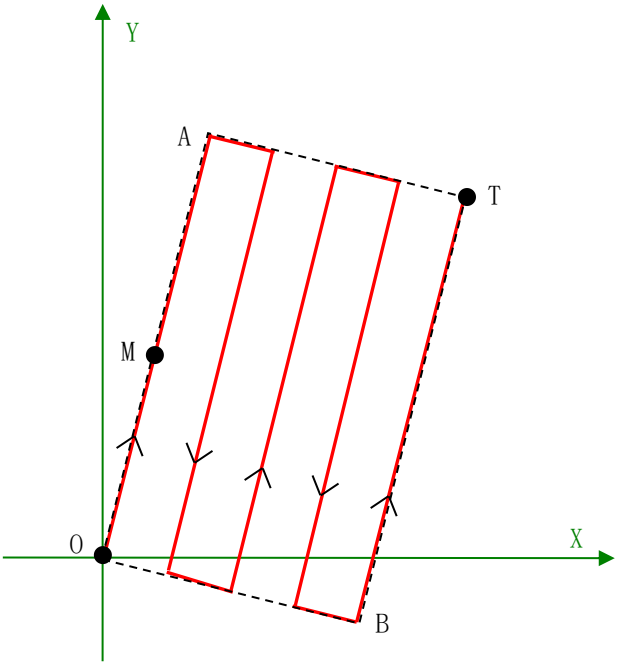


图 8.17　逐行模式的矩形插补运动示意图

如图 8.17 所示，要进行一次逐行模式的矩形插补运动，必须确定以下三个参数：

（1）对角位置，如图 8.15 中的点 T，该点确定了矩形的对角点。

（2）矩形方向标记位置，如图 8.15 中的点 M。矩形方向标记位置确定了矩形插补运动第一条边的运动方向，该方向为射线 OM 的方向。

此时，通过对角位置坐标及矩形边的方向，则可以确定该矩形区域的框架，如图 8.12 中的矩形 OATB。

（3）行数，如图 8.15 中矩形插补运动的行数为 5。通过行数参数，可以设置在指定矩形区域内进行矩形逐行插补运动的行数。

例 8.21：逐行模式的矩形插补运动

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyCount, Myrect_mode,Myposi_mode;

MyCardNo = 0;                                    //卡号
MyCrd = 0;                                       //坐标系号
MyaxisNum =2;                                    //插补运动轴数为 2
MyCount = 6;                                     //行数为 6
Myrect_mode = 0;                                 //设置矩形插补模式为逐行模式
Myposi_mode = 0;                                 //设置运动模式
LTDMC.dmc_conti_set_lookahead_mode(MyCardNo, MyCrd, 0, 100, 1, 1000);//关闭前瞻模式
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
```

```
                                //设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
    LTDMC.dmc_rectangle_move_unit(0, 0, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 3000, 2000 },
new double[] { 700, 0 }, MyCount, Myrect_mode, Myposi_mode);
                                //执行逐行矩形插补运动
    ……
```

运行结果如图 8.18 所示。

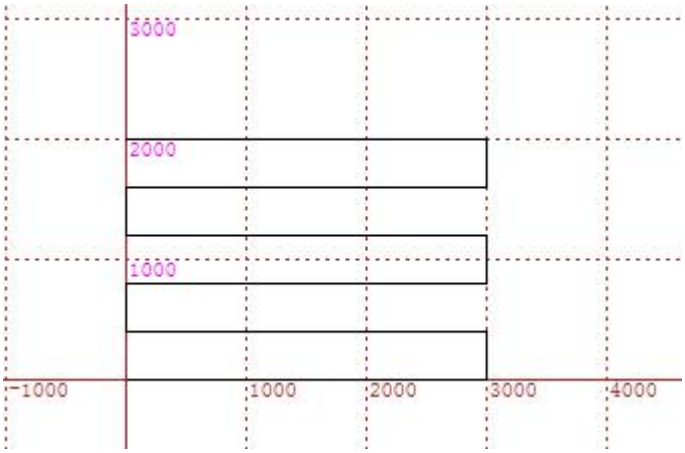


图 8.18　逐行模式的矩形插补运动

8.6.5.2　渐开线模式的矩形插补运动

如图 8.17 所示，要进行一次渐开线模式的矩形插补运动，必须确定以下三个参数：

（1）对角位置，如图 8.17 中的点 T，该点确定了矩形的对角点。

（2）矩形方向标记位置，如图 8.17 中的点 M。矩形方向标记位置确定了矩形插补运动第一条边的运动方向，该方向为射线 OM 的方向。

此时，通过对角位置坐标及矩形边的方向，则可以确定该矩形区域的框架，如图 8.17 中的矩形 OATB。

（3）圈数，如图 8.19 中矩形插补运动的圈数为 3。通过圈数参数，可以设置在指定矩形区域内进行矩形区域渐开线插补运动的圈数。

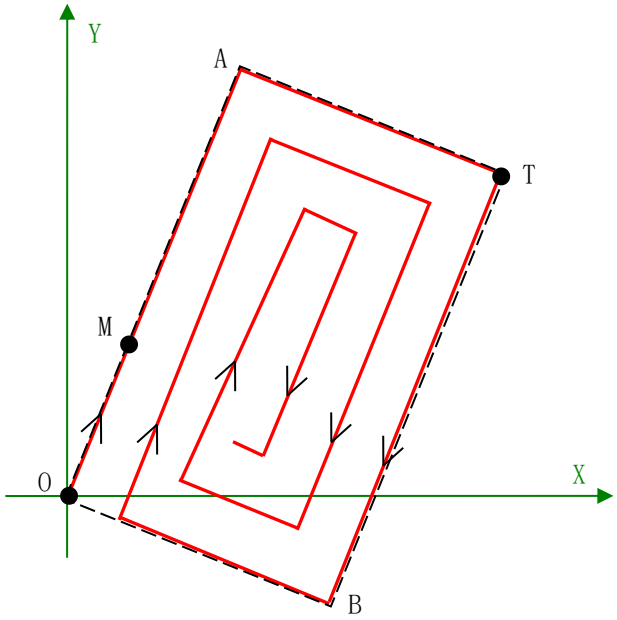


图 8.19　渐开线模式的矩形插补运动示意图

例 8.22：渐开线模式的矩形插补运动

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyCount, Myrect_mode, Myposi_mode;

MyCardNo = 0;                                    //卡号
MyCrd = 0;                                       //坐标系号
MyaxisNum = 2;                                   //插补运动轴数为 2
MyCount = 4;                                     //行数为 4
Myrect_mode = 1;                                 //设置矩形插补模式为渐开线模式
Myposi_mode = 0;                                 //设置运动模式
LTDMC.dmc_conti_set_lookahead_mode(MyCardNo, MyCrd, 0, 100, 1, 1000);//关闭前瞻模式
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
                    //设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_rectangle_move_unit(0, 0, MyaxisNum, new ushort[] { 0, 1 }, new double[] {2000, 1500 }, new
double[] { 200, 500 }, MyCount, Myrect_mode, Myposi_mode);
                    //执行渐开线模式的矩形插补运动
……
```

运行结果如图 8.20 所示。

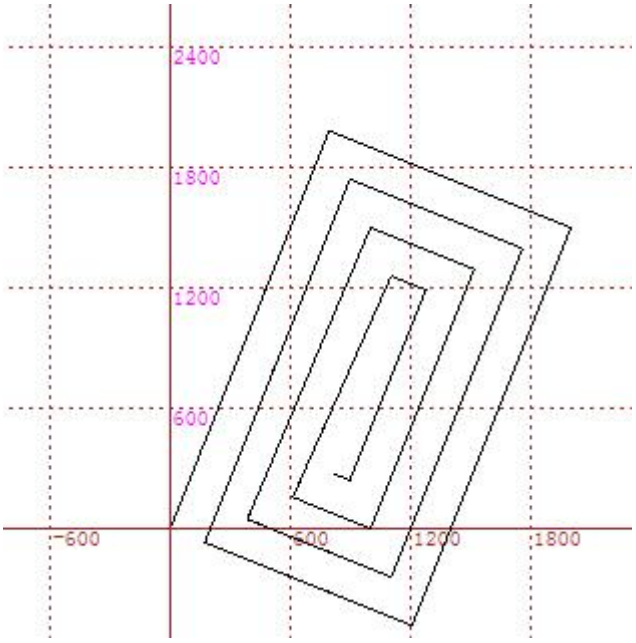


图 8.20　渐开线模式的矩形插补运动

8.7　连续插补运动的实现

8.7.1　连续插补运动

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

DMC5X10 系列卡提供了连续插补运动功能。连续插补指令支持直线插补，圆弧插补，螺旋线插补，矩形插补，IO 控制等。各种曲线轨迹示意图参见图 2.8。

此外，DMC5X10 系列卡支持四个坐标系，每个坐标系的连续缓冲区最多可缓存 5000 条指令。四个坐标系的速度可独立设置，执行连续插补时四个坐标系可独立进行连续插补运动，即可同时进行四组连续插补运动。

相关函数如表 8.7 所示。

表 8.7　连续插补运动相关函数说明

名称	功能	参考
dmc_set_vector_profile_unit	设置连续插补运动速度曲线	10.4 节
dmc_conti_open_list	打开连续插补缓冲区	10.6 节
dmc_conti_start_list	开始连续插补	
dmc_conti_close_list	关闭连续插补缓冲区	

名称	功能	参考
dmc_set_vector_profile_unit	设置连续插补运动速度曲线	10.4 节
dmc_conti_line_unit	连续插补中直线插补指令	
dmc_conti_arc_move_center_unit	连续插补中基于圆心圆弧扩展的螺旋线插补指令（可作两轴圆弧插补）	
dmc_conti_arc_move_radius_unit	连续插补中基于半径圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）	
dmc_conti_arc_move_3points_unit	连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴或三轴圆弧插补）	

实现基本连续插补运动的一般步骤如下：

1）使用函数 dmc_conti_open_list 打开连续插补缓冲区；

2）使用 dmc_set_vector_profile_unit 函数设置连续插补速度曲线；

3）编写连续插补运动指令；

4）使用函数 dmc_conti_start_list 启动连续插补运动；

5）使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

例程 8.23：连续插补运动

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int Mymark;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 3;                                         //插补轴数为 3
Myposi_mode = 0;                                       //相对坐标模式
Mymark = 0;                                            //自动编号
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
                                                       //打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
                    //设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new double[]
{ 2000, 3000, 1000 }, Myposi_mode, Mymark);        //直线插补，相对模式
MyaxisNum = 2;                                         //重新定义插补轴数为 2
LTDMC.dmc_conti_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 1500, 0 }, new double[] { 750, 0 }, 1, 0, 0, 0);
                                                       //XY 平面圆弧插补，逆时针，相对坐标模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);       //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);       //关闭连续插补缓冲区
……
```

8.7.2 连续插补运动的暂停延时功能

DMC5X10 系列卡对连续插补运动提供了暂停延时功能，相关函数如表 8.9 所示。

表 8.8 连续插补运动的暂停延时功能相关函数说明

名称	功能	参考
dmc_conti_delay	连续插补中暂停延时指令	10.6 节

注意：1）延时时间为运动停止时的等待时间。

2）当延时时间设置为 0 时，延时时间将无限长。

例程 8.24：连续插补运动的延时功能

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int Mymark;
double MyDelayTime;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum =2;                                          //插补轴数为 3
Myposi_mode = 0;                                       //相对坐标模式
Mymark = 0;                                            //自动编号
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
                                                       //打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
                          //设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s
LTDMC.dmc_set_s_profile(MyCardNo, MyCrd, 0, 0.02);//设置连续插补 S 段时间为 0.02s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000,1000 }, Myposi_mode, Mymark);                   //直线插补，相对模式
MyDelayTime=2;                                         //延时时间为 2 秒
LTDMC.dmc_conti_delay(MyCardNo, MyCrd, MyDelayTime, Mymark);//暂停延时 2 秒
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 1000, 3000 }, Myposi_mode, Mymark);                  //直线插补，相对模式

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

运行结果：

当完成第一段直线插补运动后，延时 2 秒，然后再继续执行后续运动。其速度曲线如图 8.21 所示。

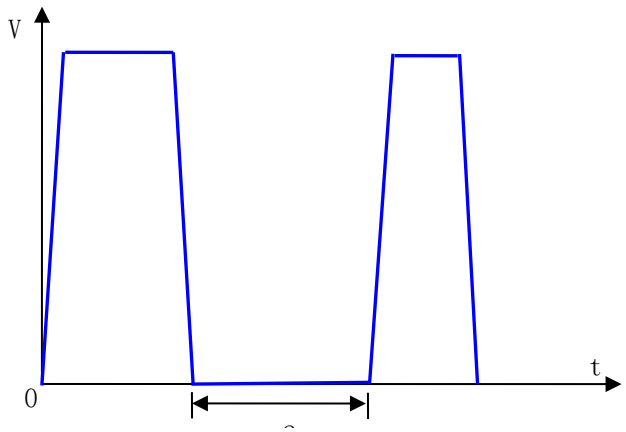


图 8.21　连续插补中暂停延时功能例程的速度曲线

8.7.3　连续插补运动的速度比例调整功能

DMC5X10 系列卡对连续插补运动提供了速度比例调整功能，相关函数如表 8.10 所示。

表 8.9　连续插补运动相关函数说明

名称	功能	参考
dmc_conti_set_override	设置连续插补速度比例	10.6 节

例程 8.25：连续插补运动中的速度比例调整

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int Mymark;
double MyVelPer ;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum =2;                                          //插补轴数为 3
Myposi_mode = 0;                                       //相对坐标模式
Mymark = 0;                                            //自动编号
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
                                                       //打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
                //设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s
LTDMC.dmc_set_s_profile(MyCardNo, MyCrd, 0, 0.02);//设置连续插补 S 段时间为 0.02s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 30000,0 }, Myposi_mode, Mymark);                     //直线插补，相对模式

MyVelPer = 1.5;                                        //设置速度比例为设置值的 1.5 倍
LTDMC.dmc_conti_set_override (MyCardNo, MyCrd, MyVelPer);
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
```

```
{ 30000, 0 }, Myposi_mode, Mymark);                    //直线插补，相对模式

MyVelPer = 2;                                          //设置速度比例为设置值的 2 倍
 LTDMC.dmc_conti_set_override (MyCardNo, MyCrd, MyVelPer);
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 30000, 0 }, Myposi_mode, Mymark);                    //直线插补，相对模式

MyVelPer = 1;                                          //设置速度比例为设置值的 1 倍
LTDMC.dmc_conti_set_override (MyCardNo, MyCrd, MyVelPer);
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);       //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);       //关闭连续插补缓冲区
……
```

运行结果：

执行第一段直线插补时，速度为 2000unit/s；执行第二段直线插补时，速度为 2000*1.5unit/s；执行第三段直线插补时，速度为 2000*2unit/s。其速度曲线如图 8.22 所示。

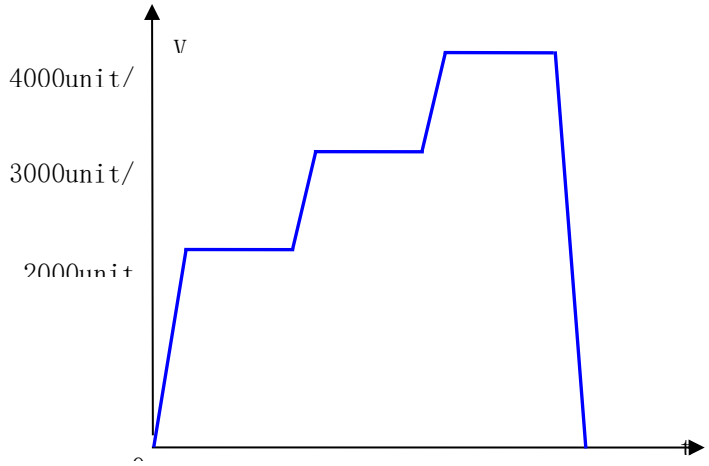


图 8.22　连续插补中速度比例调整功能例程的速度曲线

……

8.7.4　连续插补 IO 控制

DMC5X10 系列卡在连续插补运动过程总支持 IO 控制，通过 IO 控制函数的调用，用户可以轻易实现各种 IO 控制功能。

8.7.5.1　连续插补暂停及异常停止时的 IO 输出控制

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

相关函数如表 8.11 所示。

表 8.11　连续插补运动的暂停延时功能相关函数说明

名称	功能	参考
dmc_conti_set_pause_output	设置连续插补暂停及异常停止时 IO 输出状态	10.9 节

例程 8.27：连续插补暂停时，通用输出口 0 及输出口 2 输出高电平，恢复运动时不恢复暂停前的 IO 状态

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyAction, Myposi_mode;
int Mymask, Mystate, mark;
MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 2;                                         //插补运动轴数为 2
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
MyAction = 2;     //激活模式为 1：暂停时输出设定的 IO 状态，恢复运行时不恢复暂停前的 IO 状态
Myposi_mode = 0;//相对运动模式
mark = 0;           //自动编号
Mymask = Convert.ToInt32("101", 2); //选择通用输出口 0 和 2 输出，第 0 位及第 2 位值为 1
Mystate = Convert.ToInt32("101", 2);//输出口 0 和 2 输出高电平，第 0 位及第 2 位值为 1
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                                                       //打开连续插补缓冲区
LTDMC.dmc_conti_set_pause_output(MyCardNo, MyCrd, MyAction, Mymask, Mystate);
                                                       //设置连续插补暂停时 IO 输出状态
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000, 4000 }, Myposi_mode, mark); //执行连续插补运动
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 2000, 4000 }, Myposi_mode, mark); //执行连续插补运动
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);       //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);       //关闭连续插补缓冲区
……
```

运行结果：

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

8.7.5.2　连续插补中的等待 IO 输入功能

当进行连续插补运动时，用户可以在缓冲区插入等待 IO 输入指令。当运动控制卡执行到此指令时，其只有在接受到输入 IO 信号或超出超时时间后，才会执行后续运动，超时时间可以自由设置。相关函数如表 8.12 所示。

表 8.12　连续插补等待 IO 输入相关函数说明

名称	功能	参考
dmc_conti_wait_input	连续插补等待 IO 输入	10.9 节

注意：1）当超时时间设为 0 时，运动控制卡将一直等待 IO 输入信号，超时时间为无限长。

2）超时时间为运动停止时的等待时间。

例程 8.28：连续插补中，先运行一段直线插补，然后等待通用输入口 0 为低电平时（或等待时间超过 5 秒后），才执行后续运动

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel,MyTimeOut;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 2;                                         //插补运动轴数为 2
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 4000, 0.1, 0, 0);
        //设置插补速度曲线参数，插补运动最大矢量速度 4000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
        //打开连续插补缓冲区
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum,new ushort []{0,1},new double
[]{4000,5000},0,0);//直线插补运动
Mybitno=0;          //通用输入口 0
MyLevel=0;          //等待输入低电平
MyTimeOut=5;        //等待 5s
LTDMC.dmc_conti_wait_input(MyCardNo, MyCrd, Mybitno, MyLevel, MyTimeOut, 0);//等待 IO 输入
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000,3000 }, 0, 0);                                  //直线插补运动
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);     //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);     //关闭连续插补缓冲区
……
```

8.7.5.3　连续插补中的普通 IO 输出控制

当进行连续插补运动时，用户可以在缓冲区插入普通 IO 输出控制指令。

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

相关函数如表 8.13 所示。

表 8.13　连续插补中的普通 IO 输出控制相关函数说明

名称	功能	参考
dmc_conti_delay_outbit_to_start	连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）	10.9 节

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

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

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

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

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

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

例程 8.29：连续插补中的 IO 输出控制

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel, MyTimeOut,MyDelayMode;
double MyDelayVal,MyRevTime;
MyCardNo = 0;                                    //卡号
MyCrd = 0;                                       //坐标系号
MyaxisNum = 2;                                   //插补运动轴数为 2
LTDMC.dmc_set_equiv(MyCardNo, 0, 100);          //设置 X 轴脉冲当量为 100pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 100);          //设置 Y 轴脉冲当量为 100pulse/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_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
            //打开连续插补缓冲区
Mybitno = 2;//通用输出口 2
MyLevel = 0;//输出电平为低电平
MyDelayVal = 1;//滞后时间为 1s
MyDelayMode   = 0;//滞后模式为滞后时间
MyRevTime = 0.5 ;//电平延时翻转时间为 0.5s
LTDMC.dmc_conti_delay_outbit_to_start(MyCardNo, MyCrd, Mybitno,MyLevel, MyDelayVal,
MyDelayMode,MyRevTime);
                //相对于轨迹段起点,滞后 1 秒，输出口 2 输出低电平，低电平持续时间为 0.5s
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 4000, 4000 }, 0, 0);//第一段轨迹直线插补
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] {2000, 4000 }, 0, 0);//第二段轨迹直线插补

Mybitno = 0;//通用输出口 0
MyLevel = 0;//输出电平为低电平
MyRevTime = 0.5 ;//电平延时翻转时间为 0.5s
LTDMC.dmc_conti_write_outbit(MyCardNo, MyCrd, Mybitno,MyLevel,MyRevTime);//IO 立即输出

LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] {5000, 3000 }, 0, 0);//第三段轨迹直线插补
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);       //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);       //关闭连续插补缓冲区
……
```

运行结果：

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

例程 8.30：清除段内未执行完的 IO 动作功能

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Mybitno, MyLevel, MyTimeOut, MyDelayMode;
double MyDelayVal, MyRevTime;
MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 2;                                         //插补运动轴数为 2
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 3000, 0.1, 0, 0);
                    //设置插补速度曲线参数，插补运动最大矢量速度 3000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                    //打开连续插补缓冲区
Mybitno = 2;       //通用输出口 2
MyLevel = 0;       //输出电平为低电平
MyDelayVal = 1; //滞后时间为 1s
MyDelayMode = 0;//滞后模式为滞后时间
MyRevTime = 3;    //电平延时翻转时间为 3s
LTDMC.dmc_conti_delay_outbit_to_start(MyCardNo, MyCrd, Mybitno, MyLevel, MyDelayVal,
MyDelayMode, MyRevTime);
                    //相对于轨迹段起点,滞后 1 秒，输出口 2 输出低电平，低电平持续时间为 0.5s
LTDMC.dmc_conti_clear_io_action(MyCardNo, MyCrd, 0x4);
                    //清除段内未执行完的通用输出口 2 的动作
```

```
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 6000, 0 }, 0, 0);    //第一段轨迹直线插补
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 4000 }, 0, 0);//第二段轨迹直线插补
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);      //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);      //关闭连续插补缓冲区
……
```

运行结果：

当第一段直线插补开始 1s 后，通用输出口 2 输出低电平，低电平持续时间为 3s。但继续运行 3s 后，已进入了第二段直线插补运动。此时，由于使用了函数 dmc_conti_clear_io_action 清除段内未执行完的 IO 操作，所以通用输出口 2 的电平将持续保持低电平，不会翻转。如图 8.25 所示。

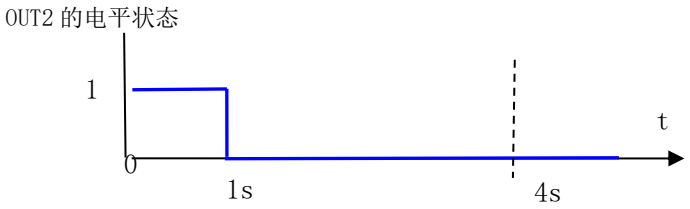


图 8.25　通用输出口 2 的电平时序图(清除段内未执行完的 IO 操作)

如果在上例中没有在第一段直线插补运动中插入指令 dmc_conti_clear_io_action 清除段内未执行完的 IO 操作，那么通用输出口 2 的电平将会在到达设置的延时翻转时间后翻转，尽管此时已进入了第二段直线插补运动。如图 8.26 所示。

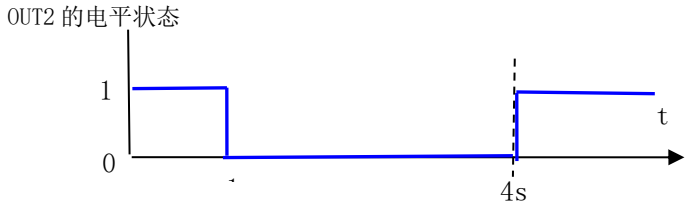


图 8.26　通用输出口 2 的电平时序图(未执行清除段内未执行完的 IO 操作)

8.7.5.4　连续插补中精确位置 CMP 输出控制

当进行连续插补运动时，用户可以在缓冲区插入精确位置 CMP 输出控制指令，当运动到达设定位置时，指定的 CMP 端口输出预先设置的电平。

在每段运动轨迹中，至多可添加 10 个 IO 操作，包括普通 IO 控制及精确位置 CMP 输出控制。精确位置 CMP 输出基本无触发延时，其分辨率为 1us。

相关函数如表 8.14 所示。

表 8.14　连续插补中的精确位置 CMP 输出控制相关函数说明

名称	功能	参考
dmc_conti_accurate_outbit_unit	连续插补中精确位置 CMP 输出控制	10.9 节

注意：1）此功能为一维高速位置比较（队列模式）的扩展功能。当启用精确位置 CMP 输出控制时，会占用高速比较器资源，所以一维高速位置比较功能与精确位置 CMP 输出控制功能不能在同一时间内使用，否则可能会出现错误动作。关于一维高速位置比较功能详见 7.13.3 一维高速位置比较功能。

2）执行精确位置 CMP 输出时，每个位置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。如果期望在连续插补中使用该功能，可以在打开连续插补缓冲区时就调用函数 dmc_hcmp_clear_points 清除相应比较器的比较点。

3）该指令的触发位置为相对于轨迹段起点的距离在坐标系内关联轴上的分量距离。

例程 8.31：连续插补中，当执行第一段直线插补时，0 号轴运行到相对起点的 2000unit 位置处，CMP0 端口输出低电平，低电平持续时间为 100us。

```
……
ushort MyCardNo, MyCrd, MyaxisNum, MyCMPno, MyLevel, MyMapAxis, MyPosSour;
double MyRelDist, MyRevTime;
MyCardNo = 0;                                   //卡号
MyCrd = 0;                                      //坐标系号
MyaxisNum = 2;                                  //插补运动轴数为 2

LTDMC.dmc_set_equiv(MyCardNo, 0, 100);          //设置 X 轴脉冲当量为 100pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 100);          //设置 Y 轴脉冲当量为 100pulse/unit
LTDMC.dmc_hcmp_clear_points(MyCardNo, 0);       //清除 CMP0 比较器所有比较点
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0,2000, 0.1, 0, 0);
            //设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
            //打开连续插补缓冲区
MyCMPno = 0 ;      //CMP0 端口
MyLevel = 1   ;    //输出电平为低电平
MyMapAxis = 0    ; //坐标系内关联轴号，X 轴，对应打开连续插补缓冲区中的 X 轴号，此例中为 0 轴
MyRelDist = 2000;//相对起点距离，2000unit
MyPosSour = 0 ;   //位置源为指令位置计数器
MyRevTime =100 ; //电平延时翻转时间为 100us
LTDMC.dmc_conti_accurate_outbit_unit(MyCardNo, MyCrd, MyCMPno, MyLevel, MyMapAxis,
MyRelDist, MyPosSour, MyRevTime);
                //设置精确位置 CMP 输出控制

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

```
[]{4000,2000},0,0);                                    //直线插补运动
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 3000, 3000 }, 0, 0);                                 //直线插补运动
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);           //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);           //关闭连续插补缓冲区

……
```

8.8 封闭型螺旋线插补运动的实现

DMC5X10 系列卡支持设置螺旋线插补运动模式，通过该设置可以实现封闭型螺旋线插补运动。相关函数如表 8.15 所示。

表 8.15 设置螺旋线插补运动模式相关函数说明

名称	功能	参考
dmc_conti_set_involute_mode	设置螺旋线插补运动模式	10.10 节
dmc_conti_get_involute_mode	读取螺旋线插补运动模式设置	

注意：1）该功能只对基于圆心圆弧扩展的螺旋线插补运动函数 dmc_arc_move_center_unit、dmc_conti_arc_move_center_unit 起作用。

2）当绽放型螺旋线插补运动设置为封闭时，先执行绽放型螺旋线插补运动，当其运动到终点后，仍然继续运行一圈（半径大小为目标位置与圆心位置的差值），将该螺旋线封闭。

3）当收敛型螺旋线插补运动设置为封闭时，先运行一个封闭的圆（半径大小为当前位置与圆心位置的差值），回到起始点后，再进行收敛型螺旋线插补运动。

例程 8.32：封闭型平面螺旋线插补运动（连续插补运动）

```
……
ushort MyCardNo, MyCrd, MyaxisNum,MyIfCirClosed;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 2;                                         //插补运动轴数为 2
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0,12000, 0.1, 0, 0);
            //设置插补速度曲线参数，插补运动最大矢量速度 12000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
          //打开连续插补缓冲区
MyIfCirClosed=1;
LTDMC.dmc_conti_set_involute_mode(MyCardNo, MyCrd, MyIfCirClosed);
          //设置连续插补螺旋线模式：封闭型
```

LTDMC.dmc_conti_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 5000, 0 }, new double[] { 500, 0 }, 0, 3, 0, 0);
//XY 平面绽放型螺旋线插补，顺时针，3 圈，相对坐标模式

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

……

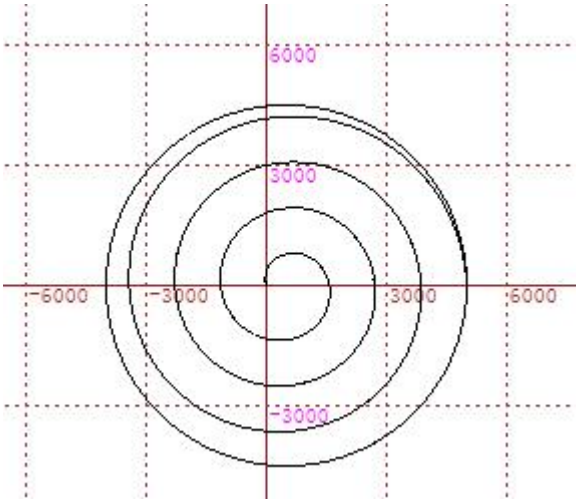


图 8.27 封闭型平面螺旋插补运动

例程 8.33：封闭型空间螺旋线插补运动（连续插补运动）

……
ushort MyCardNo, MyCrd, MyaxisNum;

MyCardNo = 0; //卡号
MyCrd = 0; //坐标系号
MyaxisNum = 3; //插补运动轴数为 3
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 12000, 0.1, 0, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 12000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1,2 });
//打开连续插补缓冲区
MyIfCirClosed=1;
LTDMC.dmc_conti_set_involute_mode(MyCardNo, MyCrd, MyIfCirClosed);
//设置连续插补螺旋线模式：封闭型

LTDMC.dmc_conti_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1,2 }, new double[] { 5000, 0,4000 }, new double[] { 3500,0 ,0 }, 0, 2, 0, 0);
//XY 平面绽放型螺旋线插补，顺时针，2 圈，相对坐标模式

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

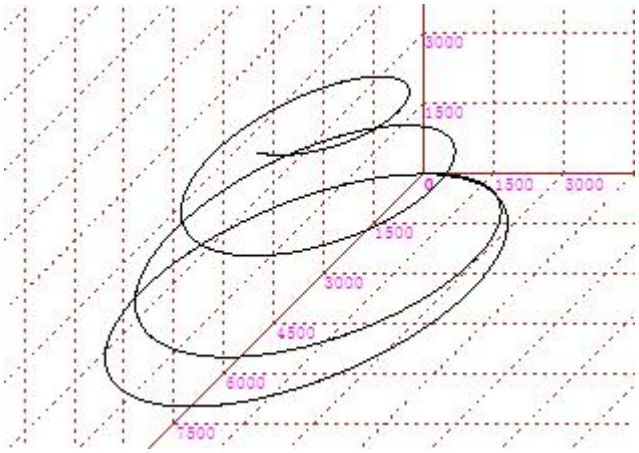


图 8.28　封闭型空间螺旋插补运动

8.9　连续插补小线段前瞻功能的实现

DMC5X10 系列卡提供了小线段前瞻功能，此功能支持直线插补、圆弧插补、螺旋线插补、IO 控制等。

相关函数如表 8.16 所示。

表 8.16　小线段前瞻相关函数说明

名称	功能	参考
dmc_conti_set_lookahead_mode	设置连续插补前瞻参数	10.8 节
dmc_conti_get_lookahead_mode	读取连续插补前瞻参数	

注意：当前瞻段数设置小于 25 段的时候会强制修改为 25 段。

一般建议通过以下顺序来调节参数（轮廓误差 PathError、前瞻加速度 LookaheadAcc、加速时间 Tacc 和减速时间 Tdec），从而达到提高加工效率的目的：

第一步：按照加工工艺和精度要求，设置合理的轮廓误差 PathError，设定轮廓误差后，再执行后续调节步骤。若调节过程改变轮廓误差值，建议重新返回第一步，按照参数调节步骤重新调节。

第二步：设定轮廓误差后，接着进行前瞻加速度 LookaheadAcc 设置，一般遵循前瞻加速度从小到大调节的原则（如从 1000 开始，每次按 2 倍或 10 倍递增），然后按照事先设定好的加减速时间（Tacc 和 Tdec），在设备上运行测试程序。此时注意观察不同前瞻加速度设置时，

设备运行的震动情况（特别是在尖锐角拐弯处）。当前瞻加速度增大到某值时设备震动较明显，表示设备能承受的加速度值已达到极限值，可适当把前瞻加速度回调一档，并最终确定前瞻加速度 LookaheadAcc。另外，若前瞻加速度设置很小时，运行测试样例时设备震动也比较厉害，建议查看设置的加减速时间（Tacc 和 Tdec）是不是过小（一般加减速时间设置小于 10ms 或以下），过小的加减速时间容易导致急加速和减速过程，也可能引发设备震动情况。此时，可先把加减速时间适当调大（如设置为 100~200ms 之间），然后再安装第二步开始调节前瞻加速度。

第三步：确认轮廓误差和前瞻加速度以后，可着手插补加速时间和减速时间的调节，从而实现进一步提高效率的目的。

例程 8.34：直线插补小线段前瞻

```
……
ushort MyCardNo, MyCrd, MyaxisNum;
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
MyaxisNum = 2;                                         //插补轴数为 2

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_set_vector_s_profile(MyCardNo, MyCrd, 0, 0.05);//设置插补速度曲线的平滑时间

LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                                                       //打开连续插补缓冲区
UInt16 enable = 1;
LTDMC.dmc_conti_set_lookahead_mode(MyCardNo, MyCrd, enable, 500, 5, 500);
            //设置连续插补前瞻参数,前瞻段数 200,允许误差 5unit，前瞻加速度 500
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, 2, new ushort[] { 0, 1 }, new double[] { 10000,
0 }, 0,0);                                             //执行两轴直线插补运动

MyMax_Vel = 1000;                                      //最大速度 1000unit/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_conti_line_unit(MyCardNo, MyCrd, 2, new ushort[] { 0, 1 }, new double[] { 5000,
10000 }, 0, 0);
```

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

8.10 反向间隙补偿功能的实现

DMC5X10 系列卡提供了反向间隙补偿功能，以降低机械传动反向间隙的影响。相关函数如表 8.17 所示。

表 8.17 反向间隙补偿功能相关函数说明

名称	功能	参考
dmc_set_backlash_unit	设置反向间隙值	10.11 节

例程 8.35：反向间隙补偿功能

```
……
ushort MyCardNo, Myaxis, MyCrd, MyaxisNum;
double MyBacklash, MyBacklashTime;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
Myaxis = 1;                                            //轴号

MyBacklash = 10;                                       //反向间隙设置值为 10unit
MyaxisNum = 2;                                         //插补轴数为 2
MyBacklashTime = 0.02;                                 //反向间隙时间为 0.02s
LTDMC.dmc_set_equiv(MyCardNo, 0, 100);                 //设置 X 轴脉冲当量为 100pulse/unit
LTDMC.dmc_set_equiv(MyCardNo, 1, 100);                 //设置 Y 号轴脉冲当量为 100pulse/unit
LTDMC.dmc_set_backlash_unit(MyCardNo, Myaxis, MyBacklash);
                                                       //Y 轴进行反向间隙补偿设置
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 500, 0.1, 0, 0);
                    //设置插补速度曲线参数，插补运动最大矢量速度 500unit/s，加减速时间 0.1s
LTDMC.dmc_set_s_profile(MyCardNo, MyCrd, 0, 0.02);
                    //设置连续插补 S 段时间为 0.02s
LTDMC.dmc_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 600, -600 }, new double[] { 600, 0 }, 0, 1, 0);
                  //XY 轴执行顺时针方向圆弧插补运动，圆弧圈数为 1，相对坐标模式
……
```

运行结果：

Y 轴设置反向间隙为 10unit,当 Y 轴开始反向运动时，为了补偿间隙误差，Y 轴先反向运

动 10unit 然后继续进行圆弧插补运动，如图 8.29 所示。

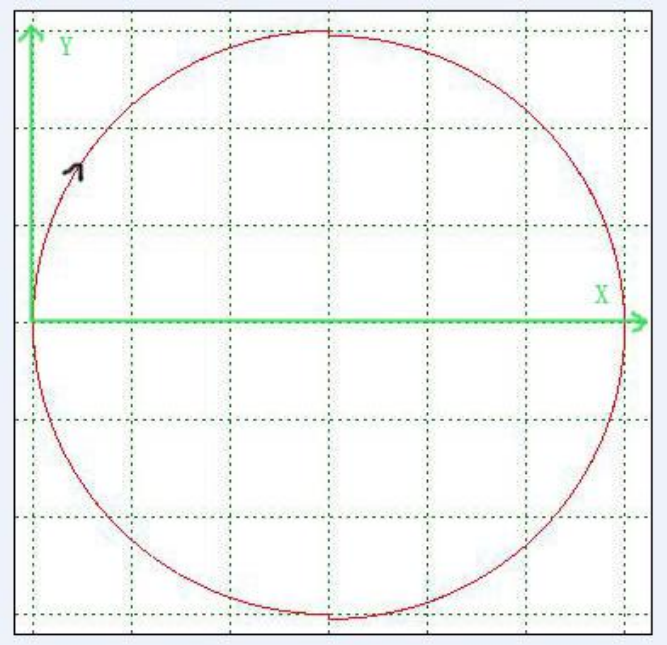


图 8.29　反向间隙补偿示意图

8.11　限位开关及急停开关的设置

详见 7.2 节 限位开关及急停开关的设置。

8.12　异常减速停止时间设置功能的实现

详见 7.7 节 异常减速停止时间设置功能的实现（DMC5C10/DMC5810/5610/5410A）。

8.13　手轮运动功能的实现

详见 7.8 节 手轮运动功能的实现。

8.14　编码器检测的实现

DMC5X10 系列卡支持直接读取以 unit 为单位的编码器计数值。相关函数如表 8.12 所示。

表 8.18　编码器检测相关函数说明

名称	功能	参考
dmc_set_counter_inmode	设置编码器输入口的计数方式	9.16 节
dmc_set_encoder_unit	设置当前编码器计数值	10.2 节
dmc_get_encoder_unit	读取当前编码器计数值	

例程 8.36：编码器检测

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

MyCardNo = 0;                                          //卡号
Myaxis = 0;                                            //轴号
Mymode = 3;                                            //设置编码器的计数方式为 4 倍频，AB 相
Myencoder_value = 0;                                   //设置 0 号轴的计数初始值为 0
LTDMC.dmc_set_counter_inmode(MyCardNo, Myaxis, Mymode);
                                                       //设置 0 号轴的编码器计数方式
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
……
```

8.15　通用 I/O 控制的实现

详见 7.12 节 通用 I/O 控制的实现。

8.16　轴 IO 映射功能的实现

详见 7.14 节 轴 IO 映射功能的实现。

8.17　PWM 输出功能的实现

DMC5X10 系列卡均提供了 PWM 输出功能。如图 8.30 所示，DMC5810 卡输出的 PWM 波形的周期为 t2（频率即为 1/t2），占空比为 t1/t2，幅值为 V1 =　5V。

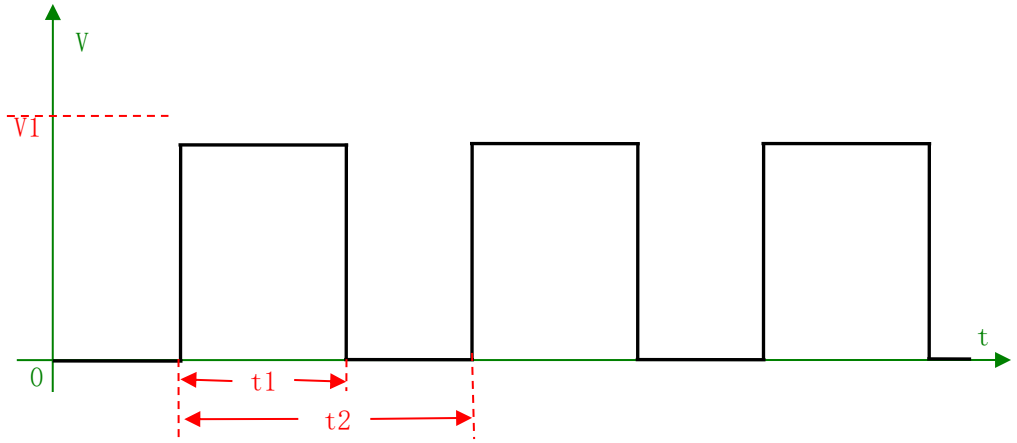


图 8.30　PWM 输出示意图

8.17.1　PWM 立即输出功能

DMC5X10 系列卡支持 PWM 立即输出功能，用户只需要简单设置 PWM 输出通道、频率及占空比参数即可实现 PWM 输出功能，非常方便。

相关函数如表 8.19 所示。

表 8.19　PWM 立即输出功能相关函数说明

名称	功能	参考
dmc_set_pwm_enable	设置 PWM 使能状态	10.12 节
dmc_set_pwm_output	设置 PWM 输出	

注意：1）DMC5C10 中，当使能 PWM 功能后，11 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，11 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1。

2）DMC5810 中，当使能 PWM 功能后，7 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，7 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1。

3）DMC5610 中，当使能 PWM 功能后，5 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，5 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1。

4）当使用通道 0 时，硬件接线可接 PUL+与 GND（PUL-与 GND 则输出相反信号）；当使用通道 1 时，硬件接线可接 DIR+与 GND（DIR-与 GND 则输出相反信号）。

5）DMC5410A 中 CN23 有两路 PWM 专用输出口，详见附录 5。

6）当使能 PWM 功能后，运动控制卡的相应轴端口将不能输出电机控制信号。

例程 8.37：PWM 输出功能

```
……
ushort MyCardNo, Myenable, MyPwmNo;
double MyfDuty, MyfFre;
```

```
MyCardNo = 0;                          //卡号
Myenable = 1;                          //PWM 输出使能状态：使能
MyPwmNo = 0;                           //PWM 输出通道为 0 通道，即 7 号轴的脉冲端口（PUL+与 PUL-）
MyfDuty = 0.5;                         //PWM 输出占空比为 50%
MyfFre = 10000;                        //PWM 输出频率为 10000Hz
LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);
    //使能 PWM 输出，DMC5810 卡 7 号轴的脉冲端口作为 PWM 通道 0，方向端口作为 PWM 通道 1
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
    //设置 PWM 输出，PWM 输出通道为通道 0（即 7 号轴的脉冲端口），占空比为 50%，频率为 10000Hz
……
……
Myenable = 0;                          //PWM 输出使能状态：禁止
LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);//禁止 PWM 输出，7 号轴恢复输出电机控制信号
……
```

运行结果：

DMC5810 的 7 号轴脉冲端口（PUL+与 PUL-）输出 PWM 波形，其占空比为 50%，频率为 10000Hz（即周期为 0.1ms）。

8.17.2 连续插补中 PWM 输出功能

DMC5X10 系列卡支持连续插补中插入 PWM 指令。相关函数如表 8.20 所示。

表 8.20 连续插补中 PWM 输出功能相关函数说明

名称	功能	参考
dmc_set_pwm_enable	设置 PWM 使能状态	10.12 节
dmc_set_pwm_onoff_duty	设置 PWM 开关状态对应的占空比	10.13 节
dmc_conti_set_pwm_output	连续插补中 PWM 输出设置	
dmc_conti_set_pwm_follow_speed	连续插补中 PWM 速度跟随	
dmc_conti_delay_pwm_to_start	连续插补中相对于轨迹段起点 PWM 滞后输出	
dmc_conti_ahead_pwm_to_stop	连续插补中相对于轨迹段终点 PWM 提前输出	
dmc_conti_write_pwm	连续插补中缓冲区立即 PWM 输出	

连续插补中控制 PWM 输出分为不跟随模式和跟随模式。

不跟随模式控制 PMM 输出一般步骤如下：

1）使用函数 dmc_set_pwm_enable 使能 PWM 输出；

2）使用函数 dmc_set_pwm_output 设置 PWM 占空比为 0，设置 PWM 频率（设置 PWM 占空比为 0，即此时 PWM 功能已使能，但暂不输出 PWM 波形）；

3）使用函数 dmc_conti_open_list 打开连续插补缓冲区；
4）添加连续插补中 PWM 输出指令 dmc_conti_set_pwm_output；
5）添加连续插补运动指令；
6）使用函数 dmc_conti_start_list 启动连续插补运动；
7）使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

跟随模式控制 PWM 输出一般步骤如下：
1）使用函数 dmc_set_pwm_enable 使能 PWM 输出；
2）使用函数 dmc_set_pwm_output 设置 PWM 占空比为 0，设置 PWM 频率（设置 PWM 占空比为 0，即此时 PWM 功能已使能，但暂不输出 PWM 波形）；
3）使用函数 dmc_conti_open_list 打开连续插补缓冲区；
4） 使用函数 dmc_set_pwm_onoff_duty 设置 PWM 打开（fOnDuty）及关闭状态（fOffDuty）的占空比，使用函数 dmc_conti_set_pwm_output 设置；
5)使用函数 dmc_conti_set_pwm_follow_speed 设置 PWM 跟随模式，函数中跟随模式 mode 有 5 种模式，不同模式，说明如下
a．跟随模式 mode=0（不跟随，保持状态），PWM 一直输出，输出的频率为步骤 2 中设置的频率，输出的占空比为步骤 4 中设置的占空比，具体采用打开（fOnDuty）还是关闭状态（fOffDuty）的占空比，由步骤 6 中添加的 PWM 指令中参数 on_off(0 采用关闭状态占空比，1 采用打开状态占空比)确定。
b. 跟随模式 mode=1（不跟随，输出低电平），根据步骤 6 中添加的 PWM 输出指令中参数 on_off 确定。
当 on_off = 1（打开模式），输出低电平；
当 on_off = 0 (关闭模式)，输出步骤 2 中设置的频率，步骤 4 中关闭状态的占空比。
c. 跟随模式 mode=2（不跟随，输出高电平），根据步骤 6 中添加的 PWM 输出指令中参数 on_off 确定。
当 on_off = 1（打开模式），输出高电平；
当 on_off = 0（关闭模式），输出步骤 2 中设置的频率，步骤 4 中关闭状态的占空比。
d. 跟随模式 mode=3（跟随，占空比自动调整），根据步骤 6 中添加的 PWM 输出指令中参数 on_off 确定。
当 on_off = 1（打开模式），输出频率为步骤 5 中函数 dmc_conti_set_pwm_follow_speed 设置的 outvalue 值，跟随的占空比 =（当前轨迹段插补速度/ MaxVel）*MaxValue。MaxVel 为 dmc_conti_set_pwm_follow_speed 函数设置的最大运行速度，MaxValue 为 dmc_conti_set_pwm_follow_speed 函数设置的

最大占空比。

当 on_off = 0(关闭模式)，输出的频率为步骤 5 中函数 dmc_conti_set_pwm_follow_speed 设置的 Outvalue，占空比为步骤 4 中关闭状态（fOffDuty）对应的占空比。

e. 跟随模式 mode=4（跟随，频率自动调整），根据步骤 6 中添加的 PWM 输出指令中参数 on_off 确定。

当 on_off = 1（打开模式），输出的占空比为步骤 5 中函数 dmc_conti_set_pwm_follow_speed 设置的 outvalue 值，跟随的频率=（当前轨迹段插补速度/ MaxVel）*MaxValue。MaxVel 为 dmc_conti_set_pwm_follow_speed 函数设置的最大运行速度，MaxValue 为 dmc_conti_set_pwm_follow_speed 函数设置的最大频率。

当 on_off = 0(关闭模式)，输出步骤 4 中函数 dmc_conti_set_pwm_output 设置的 PWM 频率，步骤 4 中关闭状态（fOffDuty）对应的占空比。

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

7）添加连续插补运动指令；

8）使用函数 dmc_conti_start_list 启动连续插补运动；

9）使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

注意：步骤 6 添加的 PWM 跟随输出指令，配置参数 on_off = 1，打开了模式，必须在关闭插补缓存区（步骤 9）之前，再次调用步骤 6 中的 PWM 跟随输出指令，配置参数 on_off = 0,将模式关闭，即打开关闭必须成对出现，否则再次运行会出现 PWM 不输出的情况。

例 8.38：连续插补中 PWM 输出功能，不跟随模式

```
……
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum;
double MyfDuty, MyfFre;
MyCardNo = 0;                    //卡号
Myenable = 1;                    //PWM 输出使能状态：使能
MyPwmNo = 0;                     //PWM 输出通道为 0 通道，即本地 OUT2
MyfDuty = 0;                     //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 0;                      //PWM 输出频率为 0Hz

LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);
  //使能 PWM 输出
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
  //设置 PWM 输出，占空比为 0，频率为 0Hz
  //设置插补速度曲线参数，插补运动最大矢量速度 500unit/s，加减速时间 0.1s
```

```
 MyaxisNum = 2;                                    //插补运动轴数为 2
MyCrd = 0;                                         //参与插补运动的坐标系 0
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 500, 0.1, 0.1, 0);
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
 LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 4000 }, 0, 0); //第一段轨迹，直线插补，相对模式
 LTDMC.dmc_conti_set_pwm_output(MyCardNo, MyCrd, MyPwmNo,0.8,200);//下一段运动开始，改变占
空比和频率
 LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 8000, 8000 }, 0, 0); //第二段轨迹，直线插补，相对模式
 LTDMC.dmc_conti_set_pwm_output(MyCardNo, MyCrd, MyPwmNo, 0.5, 100000);//下一段运动开始，改
变占空比和频率
 LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 10000, 10000 }, 0, 0); //第三段轨迹，直线插补，相对模式
 LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
……
```

运行结果：

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

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

例程 8.39：连续插补中 PWM 输出功能，跟随模式 0

```
……
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum;
ushort MyPwmMode, MyOnOff, MyDeMode,MyAhMode ;
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue;
double MyOutValue, MyDeVal, MyRevTime,MyAhVal ;

MyCardNo = 0;                          //卡号
Myenable = 1;                         //PWM 输出使能状态：使能
MyPwmNo = 0;                           //PWM 输出通道为 0 通道，即 7 号轴的脉冲端口（PUL+与 PUL-）
MyfDuty = 0;                          //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 20000;                       //PWM 输出频率为 20000Hz

LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);
  //使能 PWM 输出，DMC5810 卡 7 号轴的脉冲端口作为 PWM 通道 0，方向端口作为 PWM 通道 1
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
  //设置 PWM 输出，PWM 输出通道为通道 0（即 7 号轴的脉冲端口），占空比为 0，频率为 20000Hz

MyCrd = 0;                                         //参与插补运动的坐标系 0
MyaxisNum = 2;                                    //插补运动轴数为 2
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
```

```
                                                    //打开连续插补缓冲区
    MyfDuty = 0.8;                                  //设置 PWM 打开状态的占空比为 80%
    MyOffFre = 0.05;                                //设置 PWM 关闭状态的占空比为 5%
    LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
                                                    //设置 PWM 开关状态对应的占空比
    MyPwmMode = 0;                                  //设置 PWM 跟随模式为 0（即非速度跟随，保持状态）
    MyMaxVel = 0;
    MyMaxValue = 0;
    MyOutValue = 0;                                 //当 PWM 跟随模式为 0、1、2 时，此三个参数无意义
    LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
    MyMaxValue, MyOutValue);                        //设置 PWM 跟随模式为 0（即非速度跟随，保持状态）

    MyOnOff = 1   ;                                 //设置 PWM 输出状态：打开
    MyDeMode = 0    ;                               //设置 PWM 输出滞后模式：滞后时间
    MyDeVal = 2 ;                                   //设置 PWM 输出滞后时间：2s
    MyRevTime = 0   ;                               //保留参数，固定值为 0
    LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo,    MyCrd,    MyPwmNo,    MyOnOff,    MyDeVal,
MyDeMode,
    MyRevTime);                                     //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
    LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 500, 0.1, 0, 0);
                        //设置插补速度曲线参数，插补运动最大矢量速度 500unit/s，加减速时间 0.1s
    LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
    { 4000, 0 }, 0, 0);                             //第一段轨迹，直线插补，相对模式

    MyOnOff = 0 ;                                   //设置 PWM 输出状态：关闭
    MyAhMode = 0;                                   //设置 PWM 输出提前模式：提前时间
    MyAhVal = 1;                                    //设置 PWM 输出提前时间：1s
    MyRevTime = 0;                                  //保留参数
    LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo,    MyCrd,    MyPwmNo,    MyOnOff,    MyAhVal,
MyAhMode,
    MyRevTime);                                     //相对于轨迹段终点,提前 1 秒，控制 PWM 输出

    LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
    { 0, 4000 }, 0, 0);                             //第二段轨迹，直线插补，相对模式
    LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
    LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
    ……
```

运行结果：

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

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

例程 8.40：连续插补中 PWM 输出功能，跟随模式 1

```
……
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum;
ushort MyPwmMode, MyOnOff, MyDeMode,MyAhMode ;
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue;
double MyOutValue, MyDeVal, MyRevTime,MyAhVal ;

MyCardNo = 0;                          //卡号
Myenable = 1;                          //PWM 输出使能状态：使能
MyPwmNo = 0;                           //PWM 输出通道为 0 通道，即 7 号轴的脉冲端口（PUL+与
PUL-）
MyfDuty = 0;                           //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 20000;                        //PWM 输出频率为 20000Hz

LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);
  //使能 PWM 输出，DMC5810 卡 7 号轴的脉冲端口作为 PWM 通道 0，方向端口作为 PWM 通道 1
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
  //设置 PWM 输出，PWM 输出通道为通道 0（即 7 号轴的脉冲端口），占空比为 0，频率为 20000Hz

MyCrd = 0;                             //参与插补运动的坐标系 0
MyaxisNum = 2;              //插补运动轴数为 2
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                            //打开连续插补缓冲区
MyfDuty = 0.8;                         //设置 PWM 打开状态的占空比为 80%
MyOffFre = 0.05;                       //设置 PWM 关闭状态的占空比为 5%
LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
                                       //设置 PWM 开关状态对应的占空比
MyPwmMode = 1;//设置 PWM 跟随模式为 1（即不跟随，输出低电平）
MyMaxVel = 0;
MyMaxValue = 0;
MyOutValue = 0;//当 PWM 跟随模式为 0、1、2 时，此三个参数无意义
LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
MyMaxValue, MyOutValue);//设置 PWM 跟随模式为 1（不跟随，输出低电平）

MyOnOff = 1;                //设置 PWM 输出状态：打开
MyDeMode = 0;               //设置 PWM 输出滞后模式：滞后时间
MyDeVal = 2;                //设置 PWM 输出滞后时间：2s
MyRevTime = 0;              //保留参数，固定值为 0
LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo,  MyCrd,  MyPwmNo,  MyOnOff,  MyDeVal,
MyDeMode,
MyRevTime);                 //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0,1000, 0.1, 0, 0);
                  //设置插补速度曲线参数，插补运动最大矢量速度 1000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
```

```
{ 4000, 0 }, 0, 0);       //第一段轨迹，直线插补，相对模式

MyOnOff = 0;                //设置 PWM 输出状态：关闭
MyAhMode = 0;               //设置 PWM 输出提前模式：提前时间
MyAhVal = 1;                //设置 PWM 输出提前时间：1s
MyRevTime = 0;              //保留参数
LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyAhVal,
MyAhMode,
MyRevTime);                 //相对于轨迹段终点,提前 1 秒，控制 PWM 输出

LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 0, 4000 }, 0, 0);                         //第二段轨迹，直线插补，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);//开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);//关闭连续插补缓冲区
……
```

运行结果：

第一段直线插补运动后，2s 之前，按照 dmc_set_pwm_output 设置的占空比及频率输出 PWM，此例程由于占空比设置为 0，估输出低电平。

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

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

例程 8.41：连续插补中 PWM 输出功能，跟随模式 3

```
……
ushort MyCardNo, Myenable, MyPwmNo, MyCrd, MyaxisNum, MyPwmMode, MyOnOff, MyDeMode,
MyAhMode;
double MyfDuty, MyfFre, MyOffFre, MyMaxVel, MyMaxValue, MyOutValue, MyDeVal, MyRevTime,
MyAhVal;

MyCardNo = 0;                   //卡号
Myenable = 1;                   //PWM 输出使能状态：使能
MyPwmNo = 0;                    //PWM 输出通道为 0 通道，即 7 号轴的脉冲端口（PUL+与
PUL-）
MyfDuty = 0;                    //PWM 输出占空比为 0，（即此时暂不输出 PWM 波形）
MyfFre = 20000;                 //PWM 输出频率为 20000Hz

LTDMC.dmc_set_pwm_enable(MyCardNo, Myenable);
  //使能 PWM 输出，DMC5810 卡 7 号轴的脉冲端口作为 PWM 通道 0，方向端口作为 PWM 通道 1
LTDMC.dmc_set_pwm_output(MyCardNo, MyPwmNo, MyfDuty, MyfFre);
  //设置 PWM 输出，PWM 输出通道为通道 0（即 7 号轴的脉冲端口），占空比为 0，频率为 20000Hz
MyCrd = 0;                      //参与插补运动的坐标系 0
```

```
MyaxisNum = 2;            //插补运动轴数为 2
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
                          //打开连续插补缓冲区
MyfDuty = 0.8;                          //设置 PWM 打开状态的占空比为 80%
MyOffFre = 0.05;                        //设置 PWM 关闭状态的占空比为 5%
LTDMC.dmc_set_pwm_onoff_duty(MyCardNo, MyPwmNo, MyfDuty, MyOffFre);
                                        //设置 PWM 开关状态对应的占空比
MyPwmMode = 3;        //设置 PWM 跟随模式为 3（即速度跟随，占空比自动调整，频率固定）
MyMaxVel = 4000;      //速度最大值为 4000 unit/s
MyMaxValue = 1;       //PWM 占空比最大值为 100%
MyOutValue = 30000; //PWM 频率固定为 30kHz
LTDMC.dmc_conti_set_pwm_follow_speed(MyCardNo, MyCrd, MyPwmNo, MyPwmMode, MyMaxVel,
MyMaxValue, MyOutValue);//设置 PWM 跟随模式为 3（即速度跟随，占空比自动调整，频率固定）

MyOnOff = 1;              //设置 PWM 输出状态：打开
MyDeMode = 0;             //设置 PWM 输出滞后模式：滞后时间
MyDeVal = 2;              //设置 PWM 输出滞后时间：2s
MyRevTime = 0;            //保留参数，固定值为 0
LTDMC.dmc_conti_delay_pwm_to_start(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyDeVal,
MyDeMode,
MyRevTime);               //相对于轨迹段起点,滞后 2 秒，控制 PWM 输出
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 1000, 0.1, 0, 0);
                          //设置插补速度曲线参数，插补运动最大矢量速度 1000unit/s，加减速时间 0.1s

LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 4000, 0 }, 0, 0);  //第一段轨迹，直线插补，相对模式

MyOnOff = 0;              //设置 PWM 输出状态：关闭
MyAhMode = 0;             //设置 PWM 输出提前模式：提前时间
MyAhVal = 1;              //设置 PWM 输出提前时间：1s
MyRevTime = 0;            //保留参数
LTDMC.dmc_conti_ahead_pwm_to_stop(MyCardNo, MyCrd, MyPwmNo, MyOnOff, MyAhVal,
MyAhMode,
MyRevTime);               //相对于轨迹段终点,提前 1 秒，控制 PWM 输出
//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_set_vector_profile_unit(MyCardNo, MyCrd, 0, 3000, 0.1, 0, 0);
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[]
{ 0, 4000 }, 0, 0);//第二段轨迹，直线插补，相对模式

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

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

……

运行结果：

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

当第二段直线插补运行时，到达匀速段 PWM 输出波形占空比为 75%，频率为 30kHz。加减速段占空比时跟随速度变化。

当第二段直线插补结束前 1 秒时，PWM 通道 0 将关闭 PWM 速度跟随输出，此时 PWM 输出波形频率为 30kHz，占空比为 5%。

8.18 圆弧限速功能的实现

DMC5X10 系列卡提供了圆弧限速功能，在执行圆弧插补运动时，加入圆弧限速指令，可实现圆弧限速功能，相关函数如表 8.21 所示。

表 8.21 圆弧限速相关函数

名称	功能	参考
dmc_set_arc_limit	设置圆弧限速参数	10.14 节
dmc_get_arc_limit	读取圆弧限速参数	

例程 8.42:圆弧限速功能的实现

```
……
ushort MyCardNo, MyCrd, MyaxisNum, Myposi_mode;
int Mymark;

MyCardNo = 0;                                          //卡号
MyCrd = 0;                                             //坐标系号
MyaxisNum = 3;                                         //插补轴数为 3
Myposi_mode = 0;
Mymark = 0;                                            //自动编号
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 });
                                                       //打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 2000, 0.1, 0, 0);
        //设置插补速度曲线参数，插补运动最大矢量速度 2000unit/s，加减速时间 0.1s
LTDMC.dmc_set_equiv(MyCardNo, 2, 100);                 //设置 Y 号轴脉冲当量为 100pulse/uni
LTDMC.dmc_set_arc_limit(MyCardNo, MyCrd, 1, 0, 0);//圆弧限速

LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1, 2 }, new double[]
{ 2000, 3000, 1000 }, Myposi_mode, Mymark);          //直线插补，相对模式
```

```
MyaxisNum = 2;                                          //重新定义插补轴数为 2
LTDMC.dmc_conti_arc_move_center_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
double[] { 1500, 0 }, new double[] { 750, 0 }, 1, 0, 0, 0);
                                                        //XY 平面圆弧插补，逆时针，相对坐标模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd);          //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd);          //关闭连续插补缓冲区
```

8.19 轨迹运动综合例程

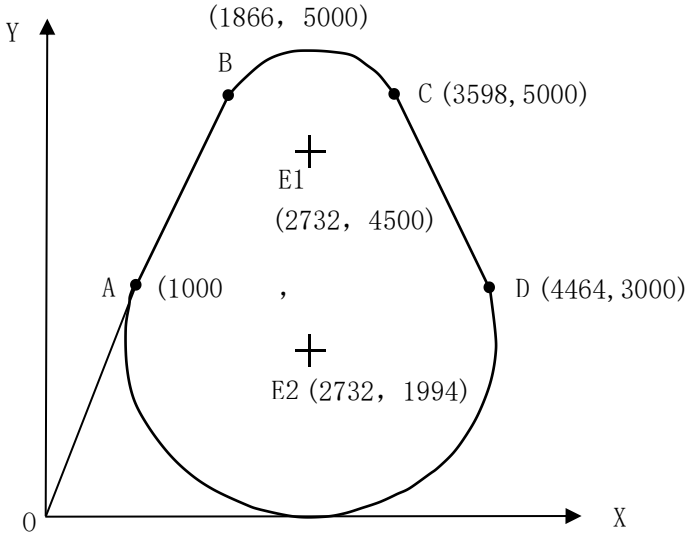
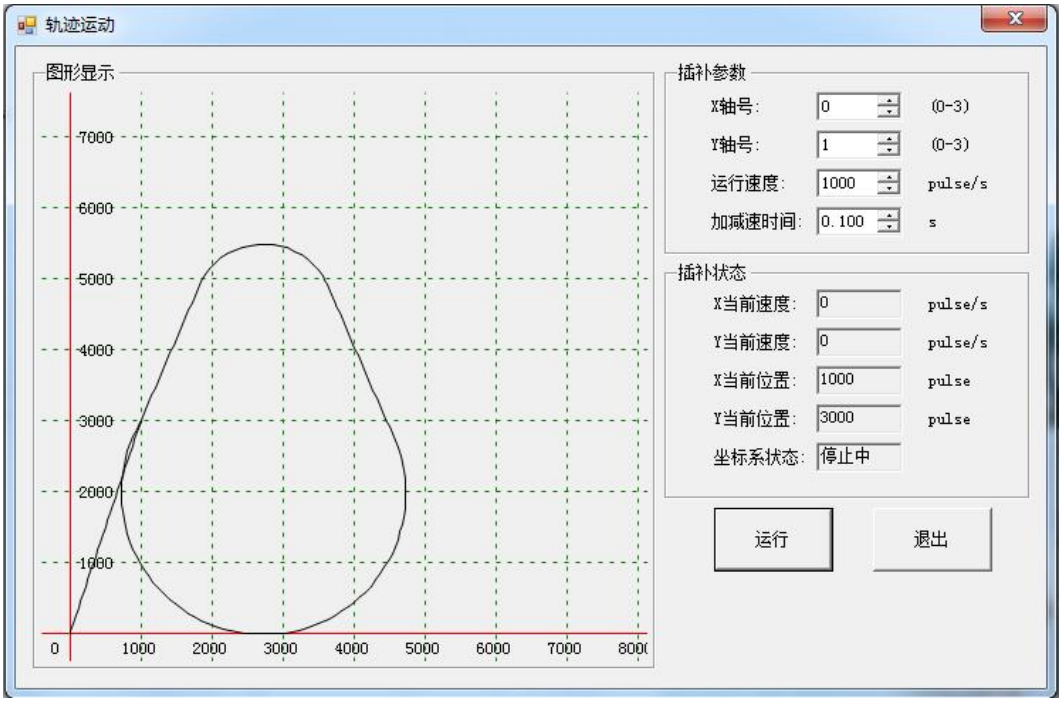


图 8.31 轨迹运动实例

运动轨迹 ABCD 如图 8.31 所示，其中 AB、CD 为直线运动；BC、DA 为圆弧运动。设当前位置位于系统原点 O 处，则先执行直线插补运动 OA，然后执行直线插补运动 AB，然后执行圆弧插补运动 BC，然后执行直线插补运动 CD，最后执行圆弧插补运动 DA。即运动步骤为 O->A->B->C->D->A。



8.31　轨迹运动实例程序运行界面

程序界面如图 8.29 所示，在插补参数输入框可以输入 X、Y 方向的电机轴号，插补运动参数。显示框中可以监控当前速度、位置、电机状态信息。图形显示框中则实时显示运动图形。此外，还添加一个定时器控件，其主要负责读取当前速度、位置、电机状态并显示在显示框中；图形显示框的曲线绘制。

在此实例中，采用绝对位置模式进行程序设计，C＃代码如下：

```
//定义全局变量
    private ushort _CardID = 0;//定义卡号

//窗口装载函数中对控制卡进行初始化
private void Form1_Load(object sender, EventArgs e)//窗口装载函数
  {
    short num = LTDMC.dmc_board_init();//获取卡数量
    if (num <= 0 || num > 8)
    {
        MessageBox.Show("初始卡失败!", "出错");
    }
    ushort _num = 0;                    //定义变量，卡数
    ushort[] cardids = new ushort[8];//定义变量，控制卡固件类型数组
    uint[] cardtypes = new uint[8];   //定义变量，控制卡硬件 ID 号数组，卡号按从小到大顺序排列
    short res = LTDMC.dmc_get_CardInfList(ref _num, cardtypes, cardids);
    if (res != 0)
    {
        MessageBox.Show("获取卡信息失败!");
    }
```

```
        _CardID = cardids[0];                //获取控制卡固件类型数组
        timer1.Start();                      //启动定时器 1
        DrawImage();                         //执行绘图程序
    }
        private List<Point> _Points = new List<Point>();//存直线连接的点

//绘图
private void DrawImage()                     //绘图程序
    {
        int width = pictureBox1.Width;       //pictureBox1 的宽度
        int height = pictureBox1.Height;     //pictureBox1 的高度
        Bitmap bm = new Bitmap(width, height);  //初始化 bm
        //封装绘图图画
        Graphics g = Graphics.FromImage(bm);    //定义变量 g
        Matrix matrix = new Matrix(1, 0, 0, -1, 0, height);//初始化 matrix
        g.Transform = matrix;
        g.SmoothingMode = SmoothingMode.AntiAlias;//平滑处理
        //坐标系绘制
        Pen pen = new Pen(Color.Red);        //定义画笔颜色为红色
        g.DrawLine(pen, new Point(0, 20), new Point(width, 20));//绘制 X 轴
        g.DrawLine(pen, new Point(20, 0), new Point(20, height));//绘制 Y 轴
        //定义画笔
        SolidBrush brush = new SolidBrush(Color.Black);    //定义画笔颜色为黑色
        g.ResetTransform();
        g.DrawString("0", this.Font, brush, 5, height - 15);//坐标原点
        g.Transform = matrix;
        //网格绘制
        pen = new Pen(Color.Green);          //定义绿色画笔
        pen.DashPattern = new float[] { 3f, 5f };//获取或设置自定义的短划线和空白区域的数组
        int n = (int)((width-20) / 50);      //定义变量 n
        for (int i = 0; i < n; i++)
        {
        int _x = (i + 1) * 50 + 20;          //定义横坐标_x
        g.DrawLine(pen, new Point(_x, 0), new Point(_x, height));//标注 X 轴上的点的坐标
        g.ResetTransform();
        string txt=((i+1)*1000).ToString();
        g.DrawString(txt, this.Font, brush, _x-15, height-15);
        g.Transform = matrix;
        }
        n = (int)((height-20) / 50);
        for (int i = 0; i < n; i++)
        {
        int _y = (i + 1) * 50 + 20;          //定义纵坐标_y
        g.DrawLine(pen, new Point(0, _y), new Point(width, _y));//标注 Y 轴上的点的坐标
```

```
        g.ResetTransform();
        string txt = ((i + 1) * 1000).ToString();
        g.DrawString(txt, this.Font, brush, 25, height - _y-5);
        g.Transform = matrix;
        }
        //轨迹绘制
        pen = new Pen(Color.Black);                                    //定义画笔颜色
        n=1000/50;
        for (int i = 0; i < _Points.Count;i++ )                        //绘制轨迹图像
        {
        if(i>0)
        {
        g.DrawLine(pen, new Point(_Points[i].X / n+20, _Points[i].Y / n+20), new Point(_Points[i - 1].X / n+20,
        _Points[i - 1].Y / n+20));
        }
        }
        g.Dispose();                                                   //释放绘图资源
        pictureBox1.Image = bm;                                        //显示图像
    }
//运动状态判断
private bool CheckMove()
    {
        ushort x = decimal.ToUInt16(numericUpDown1.Value);//X 轴号
        ushort y = decimal.ToUInt16(numericUpDown2.Value);//Y 轴号
        if (LTDMC.dmc_check_done(_CardID, x) == 0 || LTDMC.dmc_check_done(_CardID, y) == 0)
                                                                       //判断 X、Y 轴运动状态
        {
            return true;                                               //返回真值
        }
        else
        {
            return false;                                              //返回假值
        }
    }

//运行按钮程序
private void button1_Click(object sender, EventArgs e)
    {
        ushort x = decimal.ToUInt16(numericUpDown1.Value);//X 轴号
        ushort y = decimal.ToUInt16(numericUpDown2.Value);//Y 轴号

        _Points.Clear();                                //清零
        DrawImage();
        LTDMC.dmc_set_equiv(_CardID, x, 1);
```

```
LTDMC.dmc_set_equiv(_CardID, y, 1);     //设置各轴脉冲当量
LTDMC.dmc_set_position(_CardID, x, 0);//当前位置设为零点
LTDMC.dmc_set_position(_CardID, y, 0);//当前位置设为零点
if (LTDMC.dmc_check_done_multicoor(_CardID, 0) == 0)
{
    MessageBox.Show("电机运动中,请稍后尝试!", "提示");
    return;
}
LTDMC.dmc_set_vector_profile_unit(
_CardID,
0,
0,
decimal.ToDouble(numericUpDown3.Value),
decimal.ToDouble(numericUpDown4.Value),
decimal.ToDouble(numericUpDown4.Value),
0
);                                          //设置插补速度曲线
ThreadPool.QueueUserWorkItem(
delegate
{
LTDMC.dmc_line_unit(_CardID, 0, 2, new ushort[] { x, y }, new double[] { 1000,3000 }, 1);
                                            //OA 段直线插补
Thread.Sleep(10);                           //睡 10ms
  while (CheckMove())                       //判断运动状态
  {
      Thread.Sleep(10);                     //睡 10ms
  }
LTDMC.dmc_line_unit(_CardID, 0, 2, new ushort[] { x, y }, new double[] { 1866, 5000 }, 1);
                                            //AB 段直线插补
Thread.Sleep(10);
  while (CheckMove())
  {
      Thread.Sleep(10);
  }
LTDMC.dmc_arc_move_center_unit(_CardID, 0,2, new ushort[] { x, y }, new int[] { 3598, 5000 }, new int[]
{ 2732, 4500 },0, 0, 1);//BC 段圆弧插补
Thread.Sleep(10);
  while (CheckMove())
  {
      Thread.Sleep(10);
  }
LTDMC.dmc_line_unit(_CardID, 0, 2, new ushort[] { x, y }, new double[] { 4464, 3000 }, 1);
                                            //CD 段直线插补
Thread.Sleep(10);
```

```
        while (CheckMove())
        {
            Thread.Sleep(10);
        }
    LTDMC.dmc_arc_move_center_unit(_CardID, 0,2, new ushort[] { x, y }, new int[] { 1000, 3000 }, new int[]
    { 2732, 1994 },0, 0, 1);//DA 段圆弧插补
    Thread.Sleep(10);
        while (CheckMove())
        {
            Thread.Sleep(10);
        }
    }
    );
}

//退出按钮程序
private void button3_Click(object sender, EventArgs e)
{
        this.Close();//退出
}

//定时器程序
private void timer1_Tick(object sender, EventArgs e)
{
    ushort x = decimal.ToUInt16(numericUpDown1.Value);                    //x 轴号
    ushort y = decimal.ToUInt16(numericUpDown2.Value);                    //y 轴号
    int posx= LTDMC.dmc_get_position(_CardID, x);                          //获取 X 轴坐标位置
    int posy = LTDMC.dmc_get_position(_CardID, y);                         //获取 Y 轴坐标位置
    Point pt = new Point(posx, posy);                                      //定义二维平面中的点
    if (_Points.Count > 0)                                                 //二维点数大于 0
    {
        int _x = _Points[_Points.Count - 1].X - posx;
        int _y = _Points[_Points.Count - 1].Y - posy;
        if (_x != 0 || _y != 0)
        {
            _Points.Add(pt);
            DrawImage();                                                   //执行绘图程序
        }
    }
    else
    {
        _Points.Add(pt);                                                   //添加直线连接的点
    }
    extBox3.Text = posx.ToString();                                        //X 当前位置
```

```
        textBox4.Text = posy.ToString();                                    //Y 当前位置
        textBox1.Text = LTDMC.dmc_read_current_speed(_CardID, x).ToString();//X 当前速度
        textBox2.Text = LTDMC.dmc_read_current_speed(_CardID, y).ToString();//Y 当前速度
        if (CheckMove())
        {
            textBox5.Text = "运行中";
        }
        else
        {
            textBox5.Text = "停止中";                                       //坐标系状态显示
        }
    }

//窗口卸载函数中对控制卡进行释放
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        timer1.Stop();
        LTDMC.dmc_board_close();                //释放控制卡资源
    }
```

8.20 连续插补中 DA 跟随功能的实现

DMC5410A 卡支持连续插补中插入 DA 输出指令。相关函数如表 8.22 所示。

表 8.22 连续插补中 DA 输出功能相关函数说明

名称	功能	参考
dmc_conti_set_da_enable	设置连续插补中 DA 输出使能	10.16 节
dmc_conti_set_da_follow_speed	设置连续插补中 DA 速度跟随	
dmc_conti_get_da_follow_speed	读取连续插补中 DA 速度跟随	
dmc_conti_set_encoder_da_follow_enable	设置 DA 单轴编码器速度跟随	
dmc_conti_get_encoder_da_follow_enable	读取 DA 单轴编码器速度跟随	

连续插补中控制 DA 跟随输出一般步骤如下：

1）使用函数 dmc_conti_open_list 打开连续插补缓冲区；

2）使用函数 dmc_conti_set_da_enable 使能 DA 速度跟随输出；

3）使用函数 dmc_conti_set_da_follow_speed 设置 DA 跟随参数；

4）添加连续插补运动指令；

5）使用函数 dmc_conti_start_list 启动连续插补运动；

6）使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

例程 8.42：连续插补中 DA 跟随功能

```
……
    ushort MyCardNo, Myenable, MyDANo, MyCrd, MyaxisNum;
    double MyMaxVel, MyMaxValue, MyAccOffset, MyDecOffset, MyDaAcc, MyDaDec;

    MyCardNo = 0;                       //卡号
    Myenable = 1;                       //DA 输出使能状态：使能
    MyDANo = 0;                         //DA 输出通道台为 0 通道台
    MyCrd = 0;                          //参与插补运动的坐标系 0
    MyaxisNum = 2;                              //插补运动轴数为 2
    LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
    //打开连续插补缓冲区
    LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 5000, 0.1, 0.1, 0);
    //设置插补速度曲线参数，插补运动最大矢量速度 5000unit/s，加减速时间 0.1s
    LTDMC.dmc_conti_set_da_enable(MyCardNo, MyCrd, Myenable, MyDANo, 0);    //使能 DA 输出通道 0

    MyMaxVel = 5000;                            //设置 DA 跟随最大速度为 5000puls/s
    MyMaxValue = 8;                             //设置 DA 跟ú随最大输出值为 8V
    MyAccOffset = 1;                            //设置 DA 跟随加速段的偏差为 1V
    MyDecOffset = 2;                            //设置 DA 跟随减速段的偏差为 2V
    MyDaAcc = 0.1; ;                            //设置 DA 加速时间为 0.2s
    MyDaDec = 0.1; ;                            //设置 DA 减速时间为 0.2s
    LTDMC.dmc_conti_set_da_follow_speed(MyCardNo, MyCrd, MyDANo, MyMaxVel, MyMaxValue,
MyAccOffset, MyDecOffset, MyDaAcc, MyDaDec); //设置 DA 跟随参数
    LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 10000,
0 }, 0, 0);             //第一段轨迹，直线插补 1，相对模式
    LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 0,
5000 }, 0, 0);              //第二段轨迹，直线插补 2，相对模式
    LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
    LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
```

运行结果：

DA 输出跟随插补速度变化，如图 8.32

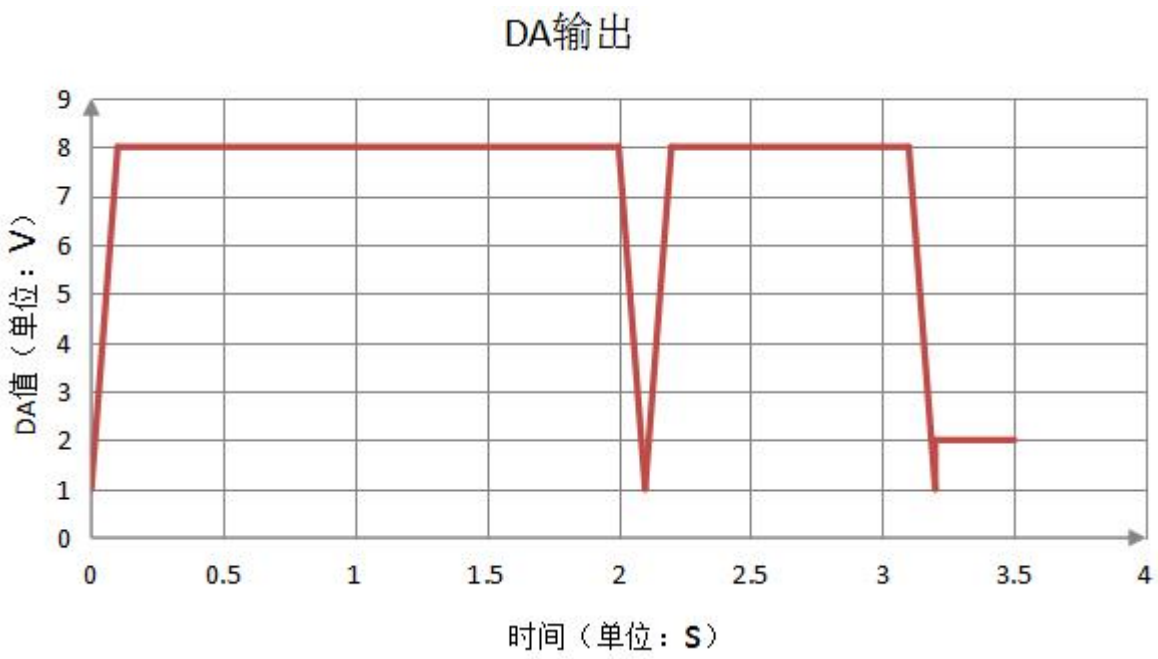


图 8.32　DA 跟随实例

8.21　二维高速位置比较功能的实现

DMC5X10 系列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。

DMC5X10 系列卡只有 OUT14/OUT15 可作为二维高速位置比较输出口。

每个比较输出端口可以与任意两轴关联，支持队列比较模式，最多可以添加 256 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。位置间时间间隔最小可达几微秒。

相关函数如表 8.23 所示。

表 8.23　二维高速位置比较相关函数说明

名称	功能	参考
dmc_hcmp_2d_set_enable	设置二维高速比较使能	10.17 节
dmc_hcmp_2d_get_enable	读取二维高速比较使能	
dmc_hcmp_2d_set_config_unit	配置二维高速比较器	
dmc_hcmp_2d_get_config_unit	读取二维高速比较器配置	
dmc_hcmp_2d_clear_points	清除二维高速比较位置	

dmc_hcmp_2d_add_point_unit	添加/更新二维高速比较位置	
dmc_hcmp_2d_get_current_state_unit	读取二维高速比较参数	
dmc_hcmp_2d_force_output	强制二维比较输出	

注意：1）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。

2）当使用高速二维比较功能时，必须确保配置的输出口有效电平与输出口的当前电平相反，如若相同，必须在使能高速比较功能之前，调用 dmc_write_outbit 将输出口的状态置反，然后再使能高速比较功能，配置高速输出口的有效电平。否则比较功能不能得到预期效果，该输出口将一直维持比较前的状态（如配置的高速输出口的有效电平 cmp_logic 为高电平，输出口当前状态也为高电平，则比较输出不能得到预期效果）。

例 8.43：二维高速比较

```
……
short iret = 0;
ushort MyCardNo = 0;                    //卡号
ushort Mycrd = 0;                       //坐标系号
ushort hcmp = 0;
ushort cmp_mode = 0;                    //比较输出模式：进入误差带后触发
ushort x_axis = 0;                      //X 轴轴号
ushort x_cmp_source = 0;                //X 轴比较源：指令位置
ushort y_axis = 1;                      //Y 轴轴号
ushort y_cmp_source = 0;                //Y 轴比较源：指令位置
double x_cmp_error = 50;                //X 误差带：50unit
double y_cmp_error = 50;                //Y 误差带：50unit
ushort cmp_logic = 0;                   //输出有效电平：低电平
int time = 200;                         //输出脉冲宽度：200us
ushort cmp_outbit = 14;                 //输出端口号
ushort enable = 1;
iret = LTDMC.dmc_hcmp_2d_set_enable(MyCardNo, hcmp, enable); //二维高速位置比较功能使能
iret = LTDMC.dmc_hcmp_2d_clear_points(MyCardNo, hcmp);//清除比较点
iret = LTDMC.dmc_hcmp_2d_set_config_unit(MyCardNo, hcmp, cmp_mode, x_axis, x_cmp_source,
x_cmp_error ,y_axis, y_cmp_source, y_cmp_error, cmp_logic, time);
                                        //配置二维高速位置比较比较器
iret = LTDMC.dmc_hcmp_2d_add_point_unit(MyCardNo, hcmp, 1000, 0, cmp_outbit);
                                        //添加第一个比较区域
```

```
iret = LTDMC.dmc_hcmp_2d_add_point_unit(MyCardNo, hcmp, 2000, 0, cmp_outbit);
                                                    //添加第二个比较区域
iret = LTDMC.dmc_hcmp_2d_add_point_unit(MyCardNo, hcmp, 3000, 0, cmp_outbit);
                                                    //添加第三个比较区域
iret = LTDMC.dmc_set_vector_profile_unit(MyCardNo, Mycrd, 0, 3000, 0.01, 0.01, 0);
                                                    //设置插补运动速度曲线
iret = LTDMC.dmc_line_unit(MyCardNo, Mycrd, 2, new ushort[] { 0, 1 }, new double[] { 5000, 0 }, 0);
                                        //直线插补运动
……
```

8.22 连续插补中位置跟随功能的实现

DMC5X10 系列卡提供连续插补位置跟随功能。该功能使插补系以外的轴能跟随插补运动的合位移运动，从而实现刀具在加工过程中处于合适的方向和位置。相关函数如表 8.24 所示。

表 8.24 连续插补中刀向跟随功能相关函数说明

名称	功能	参考
dmc_conti_gear_unit	连续插补中位置跟随	10.18 节

连续插补中控制位置跟随输出一般步骤如下：

1）使用函数 dmc_conti_open_list 打开连续插补缓冲区；

2）使用函数 dmc_conti_gear_unit 设置位置跟随参数；

3）添加连续插补运动指令；

4）使用函数 dmc_conti_start_list 启动连续插补运动；

5）使用函数 dmc_conti_close_list 关闭连续插补缓冲区。

例程 8.42：连续插补中位置跟随功能。

设定跟随旋转轴转一圈的脉冲数为 10000.

```
ushort MyCardNo, MyCrd, MyaxisNum;

MyCardNo = 0;                    //卡号
MyCrd = 0;                       //参与插补运动的坐标系 0
MyaxisNum = 2;                              //插补运动轴数为 2
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
            //打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 5000, 0.1, 0.1, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 5000unit/s，加减速时间 0.1s
LTDMC.dmc_set_profile_unit(_CardID, 2, 0, 5000, 0.1, 0.1, 0);
//********第一段**********//
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 10000, 10000 }, 0, 0);//第一段轨迹，直线插补 1，相对模式
//********第二段**********//
LTDMC.dmc_conti_pmove_unit(MyCardNo, MyCrd, 2, -1250, 0, 0, 0);//设置位置偏转，逆时针 45 度
LTDMC.dmc_conti_gear_unit(MyCardNo, MyCrd, 2, 5000, 0, 0);//设置位置跟随运动，顺时针 180 度
LTDMC.dmc_conti_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new
```

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

运行结果：

插补运动轨迹曲线，如图 8.33

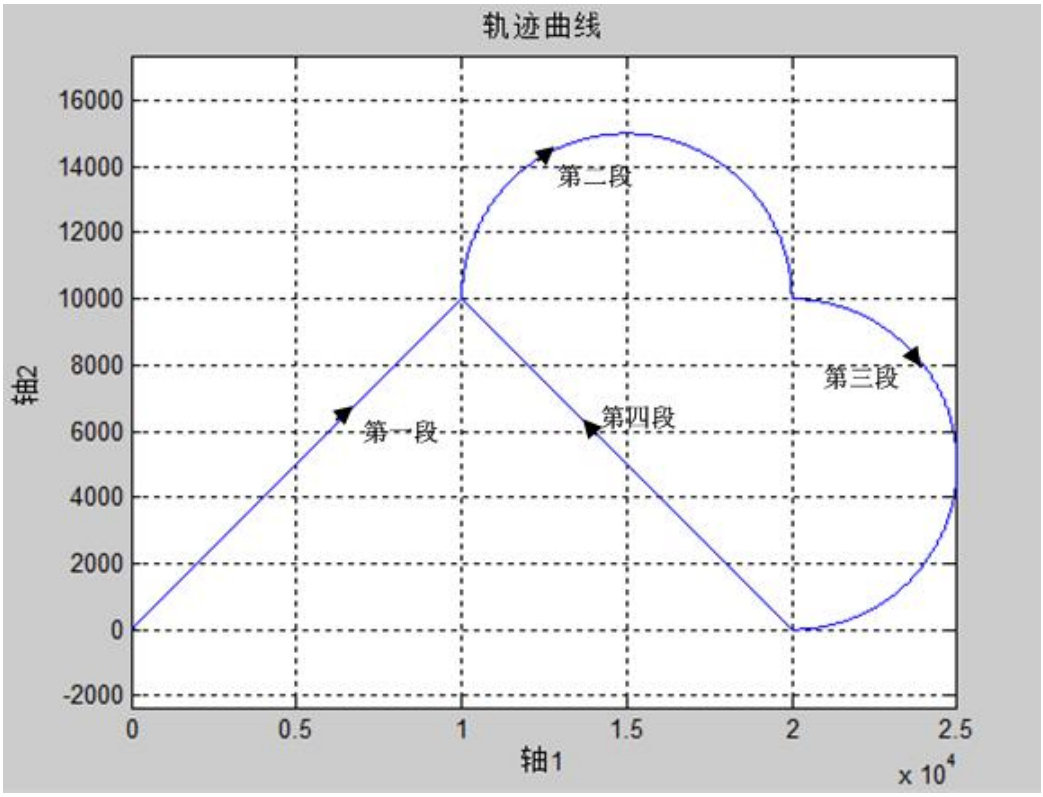


图 8.33　刀具轨迹曲线

8.23 螺距补偿功能的实现

DMC5X10 系列卡提供插补运动螺距补偿功能。该功能使插补轴根据设置进行位置补偿。相关函数如表 8.25 所示。

表 8.25 螺距补偿功能相关函数说明

名称	功能	参考
dmc_enable_leadscrew_comp	设置螺距补偿的使能与禁止	10.19 节
dmc_set_leadscrew_comp_config_unit	配置螺距补偿参数	
dmc_get_leadscrew_comp_config_unit	读取配置螺的距补偿参数	

注意：

1、 一般在回零完成之后使用该功能

2、 补偿值为相对于起点的误差值

3、 注意边界点是不补偿的

连续插补中使用螺距补偿功能一般步骤如下：

1）使用函数 dmc_enable_leadscrew_comp 使能螺距补偿；

2）使用函数 dmc_set_leadscrew_comp_config_unit 设置螺距补偿参数；

3）配置运动参数启动运动。

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

|_____|_____|_____|_____|_____|_____|_____|_____|

1 2 3 4 5 6 7 8 9

0 7 10 9 6 -1 -5 -3 -1 → 正方向运动补偿量 unit

0 -9 -5 -4 0 3 6 4 2 ← 反方向运动补偿量 unit

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

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

往负方向运动时，各点需要补偿的脉冲数为（0, -9，-5，-4，0, 3, 6, 4，2 ）；

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

例 8.43：螺距补偿功能

```
ushort MyCardNo, MyCrd, MyaxisNum;

ushort MyCardNo, MyCrd, MyaxisNum,axis,enable, npos;
double startpos, lenpos;
```

```
short err = 0;

MyCardNo = 0;                    //卡¨号
MyCrd = 0;                       //参与插补运动的坐标系
MyaxisNum = 2;                   //插补运动轴数为 2
axis = 0;                        //螺距补偿轴
enable = 0;                      //设置螺距补偿使能ü
npos = 8;                        //螺距补偿点数
startpos = 0;                    //补偿的起始位置
lenpos = 8000;                   //补偿的长度
double[] comPos = new double[] { 0, 7, 10, 9, 6, -1, -5, -3,-1 }; // 正方向运动时，各点位置需要补偿的脉冲数，一共 9 个点
double[] comNeg = new double[] { 0, -9, -5, -4, 0, 3, 6, 4,2 }; // 负方向运动时，各点位置需要补偿的脉冲数，一共 9 个点
err = LTDMC.dmc_set_leadscrew_comp_config_unit(MyCardNo, axis, npos, startpos, lenpos, comPos, comNeg);  //配置螺距补偿参数
err = LTDMC.dmc_enable_leadscrew_comp(MyCardNo, axis, enable); //使能轴的螺距补偿
LTDMC.dmc_conti_open_list(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 });
//打开连续插补缓冲区
LTDMC.dmc_set_vector_profile_unit(MyCardNo, MyCrd, 0, 5000, 0.1, 0.1, 0);
//设置插补速度曲线参数，插补运动最大矢量速度 5000unit/s，加减速时间 0.1s
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 1000, 1000 }, 0, 0);//第一段轨迹，直线插补，相对模式
LTDMC.dmc_conti_arc_move_radius_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 2000, 0 }, 1000, 1, 0, 0, 0);//第二段轨迹，圆弧插补，相对模式
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { 3000, 1000 }, 0, 0);//第三段轨迹，直线插补，相对模式
LTDMC.dmc_conti_line_unit(MyCardNo, MyCrd, MyaxisNum, new ushort[] { 0, 1 }, new double[] { -6000, 2000 }, 0, 0);//第四段轨迹，直线插补，相对模式
LTDMC.dmc_conti_start_list(MyCardNo, MyCrd); //开始连续插补
LTDMC.dmc_conti_close_list(MyCardNo, MyCrd); //关闭连续插补缓冲区
```

8.24 龙门功能的实现

DMC5X10 系列卡提供了龙门跟随功能，相关函数如表 8.26 所示。

表 8.26 龙门相关函数

名称	功能	参考
dmc_set_gear_follow_profile	设置轴跟随模式参数	10.20 节
dmc_get_gear_follow_profile	读取轴跟随模式参数	
dmc_set_grant_error_protect	设置龙门模式主从轴编码器跟随误差停止阀值	

名称	功能	参考
dmc_get_grant_error_protect	读取龙门模式编码器位置跟随误差停止阈值	

注意：龙门功能函数调用有先后关系要求，需先调用 dmc_set_gear_follow_profile 建立龙门关系，再调用 dmc_set_grant_error_protect 设置龙门误差带，不可对调先后顺序，否则会容易报错误差超限制。

例程 8.44：定长运动中龙门功能的实现

```
……
ushort MyCardNo, Myaxis, Axis, Myposi_mode, Master_axis;
double MyEquiv, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec, MyStop_Vel, MyDist, Ratio;

MyCardNo = 0;                                          //卡号
Myaxis = 0;                                            //轴号
Axis = 1;                                              //跟随轴轴号
Master_axis = 0;                                       //主轴轴号
LTDMC.dmc_set_pulse_outmode(MyCardNo, Myaxis, 0);//设置 0 号轴脉冲输出方式
MyEquiv = 100;                                         //脉冲当量值为 100pulse/unit
MyMin_Vel = 500;                                       //设置起始速度为 500unit/s
MyMax_Vel = 2000;                                      //设置最大速度为 2000unit/s
MyTacc = 0.01;                                         //设置加速时间为 0.02s
MyTdec = 0.01;                                         //设置减速时间为 0.01s
MyStop_Vel = 500;                                      //设置停止速度为 1000unit/s
LTDMC.dmc_set_profile_unit(MyCardNo, Master_axis, MyMin_Vel, MyMax_Vel, MyTacc, MyTdec,
MyStop_Vel);                                           //设置 0 号轴速度曲线参数

MyDist = 10000;                                        //设置运动距离为 10000unit
Myposi_mode = 0;                                       //设置运动模式为相对坐标模式
Ratio = 1;                                             //设置跟随倍率为 1
LTDMC.dmc_set_gear_follow_profile(MyCardNo, Axis, 1, Master_axis, Ratio);
                                                       //1 号轴跟随主轴运动
LTDMC.dmc_pmove_unit(MyCardNo, Master_axis, MyDist, Myposi_mode); //0 号轴定长运动
```

8.25 软着陆功能的实现

软着陆功能主要用来实现 pmove 运动可以一个较低的速度平稳地接近目标位置，降低运动部件对接触点的冲击。如上图所示，重点想实现的功能为 pmove 结束时能以一个很低速度靠近结束点。将 pmove 分为两段，第一段以 vs, vm，ve 进行规划，第二段以最大速度 ve 进行规划。同样，强制变位置 dmc_update_target_position_extern 也有该项功能。常用于晶圆取放等场合，相关函数如表 8.27 所示。

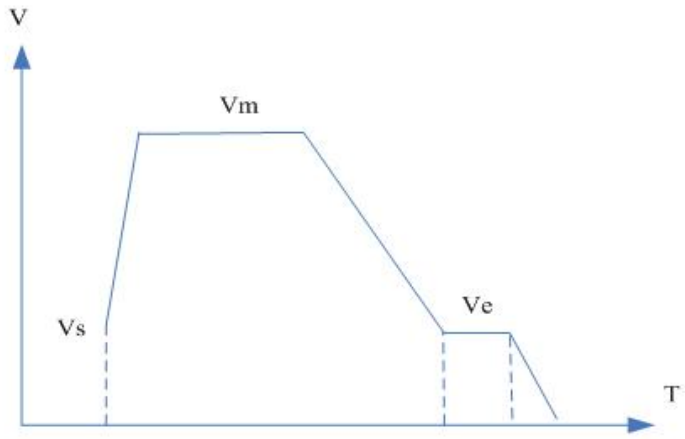


表 8.27 软着陆运动功能相关函数说明

名称	功能	参考
dmc_pmove_extern	实现 profile、pmove 整合，缩短指令时间	10.21 节
dmc_t_pmove_extern	实现 profile、pmove 整合，缩短指令时间，并且实现软着陆	
dmc_update_target_position_extern	强行改变指定轴的当前目标位置并且实现软着陆	

备注：软着陆不支持 S 型速度曲线

例 8.45：软着陆运动

```
……
double start,speed,stop, acc,dec,dis,softdist;
ushort cardID,axis ,posi_mode;

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

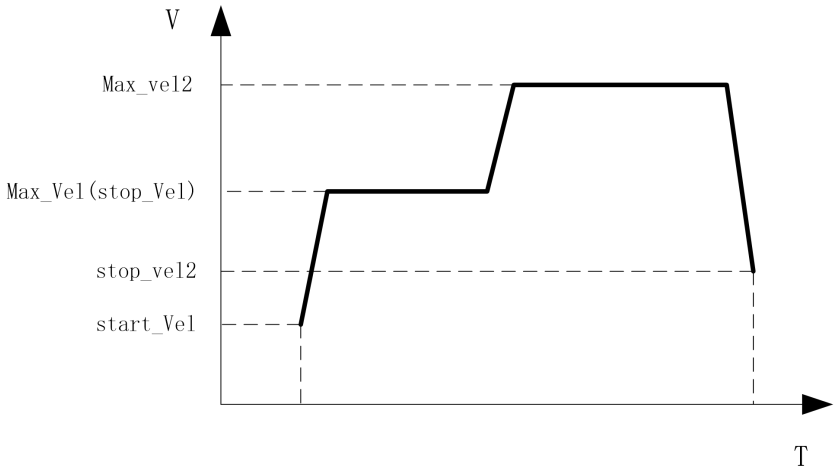
```
acc = 0.1;                                  //加速时间
dec = 0.1;                                  //减速时间
posi_mode = 1;                              //绝对运动模式
LTDMC.dmc_t_pmove_extern(cardID ,axis, dis, dis + softdist, start, speed, stop, acc, dec, posi_mode); //执行软着陆运动
……
```

例 8.46：强制变位软着陆运动

```
……
double start,speed,stop, acc,dec,dis,softdist;
ushort cardID,axis ,posi_mode;
int pos;

cardID = 0;
axis = 0;                                   //轴号
dis =1000;                                  //定长运动距离
softdist =40;                               //软着陆时低速运动距离
start = 10;                                 //启动速度
speed = 500;                                //运行速度
stop = 10;                                  //停止速度，软着陆低速运动时速度
acc = 0.1;                                  //加速时间
dec = 0.1;                                  //减速时间
posi_mode = 1;                              //绝对运动模式
s_para = 0;                                 //平滑时间
LTDMC.dmc_ pmove_extern(cardID ,axis, dis, start, speed, stop, acc, dec, s_para, posi_mode); //执行运动
while (true)
    {
        pos = LTDMC.dmc_get_position(cardID ,axis);
        if (pos > dis -50)
        {
            break;
         }
         Application.DoEvents();
     }
LTDMC.dmc_update_target_position_extern(cardID ,axis, dis + 500, dis + 500+ softdist, 0, 0)   //强制变位并且实现软着陆
……
```

8.26 软启动功能的实现



软启动功能主要用来实现启动时以一个较低速度平稳运动，然后快速 pmove 到目标位置，保证启动平稳的同时又不失效率。如下图所示，重点想实现的功能为 pmove 启动时能以一个很低速度平稳运行。将 pmove 分为两段，第一段以 start_Vel、Max_Vel、stop_Vel 进行规划，第二段以最大速度 Max_Vel2 进行规划。常用于晶圆抓取等场合。相关函数如表 8.28 所示

表 8.28 软启动运动功能相关函数说明

名称	功能	参考
dmc_t_pmove_extern_softstart_unit	实现 profile，pmove 整合，缩短指令时间，并且实现软启动	10.21 节

备注：软启动不支持 S 型速度曲线

例 8.47：软启动运动

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

cardID = 0;
axis = 0;                          //轴号
targetPos =1000;                   //目标位置
midPos = 60;                       //软启动时低速运动位置
start = 10;                        //启动速度
speed_low = 30;                    //第一段低速速度
stop_mid = 30;                     //停止速度，软着陆低速运动时速度
delayms = 0;                       //第一段完成后延时时间ms
speed_high = 500;                  //第二段高速速度
```

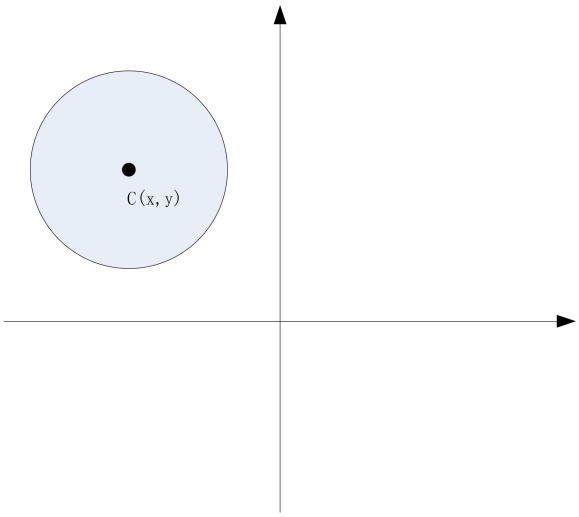
```
stop_end = 10;                          //第二段停止速度
acc = 0.1;                              //加速时间
dec = 0.1;                              //减速时间
posi_mode = 1;                          //绝对运动模式
LTDMC. dmc_t_pmove_extern_softstart_unit (cardID ,axis, midPos, targetPos, start, speed_low, stop_mid, delayms, speed_high, stop_end, acc, dec, posi_mode); //执行软启动运动
```

8.27 圆形区域限位功能的实现

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

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

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



备　注：1） 由于触发圆形限位运动停止，获取停止原因返回值为 1103；

2） 轴在运动中，不可设置圆形限位参数；

3） 开启功能后，可通过重复调用 dmc_set_arc_zone_limit_config 函数修改参数；

4） 使用先配置圆形限位参数，再使能功能。

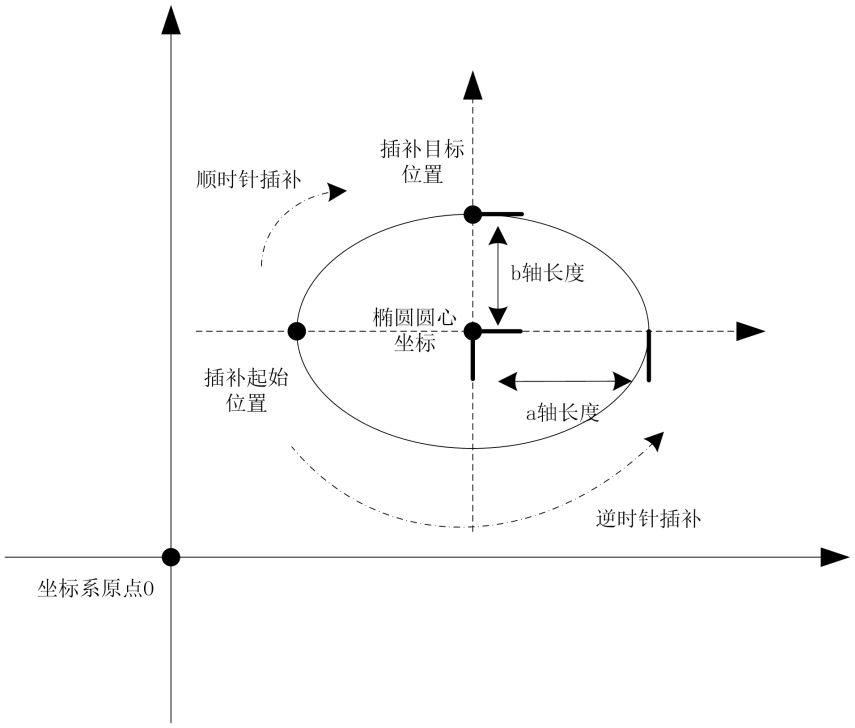
相关函数如表 8.29 所示

表 8.29　圆形区域限位功能相关函数说明

名称	功能	参考
dmc_arc_zone_limit_enable	开启/关闭圆形区域限位功能	10.23 节
dmc_get_arc_zone_limit_enable	获取圆弧限位功能的使能状态	
dmc_set_arc_zone_limit_config_unit	配置圆形区域限位参数	
dmc_get_arc_zone_limit_config_unit	回读配置的圆形区域限位参数	
dmc_get_arc_zone_limit_axis_status	查询相应轴的状态	

8.28 椭圆插补及切向跟随的实现

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



备　注：1）a 轴定义为平行于 x 轴的椭圆半轴长度，b 轴定义为平行于 y 轴的椭圆半轴长度；

2）插补起始位置为坐标系轴当前位置；

3）目前仅支持标准椭圆，即长轴和短轴与 XY 平行；

4）运动开始前，用户应当将旋转轴调整到与插补曲线的起点位置相切的角度；

5）插补运动中，不能取消跟随运动；

6）插补运动结束后，跟随关系**自动解除**，下次运动时，需要根据需要**重新进行配置**

切向跟随运动可以配合椭圆插补或圆弧插补运动，实现跟随轴的旋转运动角度与曲线当前插补位置的切向保持一致。

该功能的使用流程为：

1）调用 dmc_set_tangent_follow 指令配置相应坐标系跟随参数；

2）启动相应坐标系的插补运动。

相关函数如表 8.30 所示

表 8.30　椭圆插补及切向跟随功能相关函数说明

名称	功能	参考
dmc_ellipse_move	启动椭圆插补运动	10.23 节
dmc_set_tangent_follow	设置切向跟随参数	
dmc_get_tangent_follow_param	读取切向跟随参数	
dmc_disable_follow_move	取消指定坐标系的跟随运动	

第 9 章　以脉冲为单位的基本运动及其他功能函数说明

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_board_reset(void)

功　能：控制卡硬件复位函数

参　数：无

返回值：错误代码

注　意：1.执行复位操作后，必须等待 5 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 5 秒后执行初始化控制卡。

2.执行硬件复位，控制卡所有资源全部重新启动，如:关闭所有的输出（通用输出、专用输出口）并停止轴运动。

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

注　意：多应用程序访问单卡或多卡时，调用该函数后，会释放工控机中所有插入的控制卡系统资源，此时正在执行的运动立即停止。

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_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_release_version(WORD CardNo,char *ReleaseVersion)

功　能：读取发布版本号

参　数：CardNo　　控制卡卡号，0-7

　　ReleaseVersion　　产品发布版本

返回值：错误代码

short dmc_get_total_axes(WORD CardNo, DWORD *TotalAxis)

功　能：获取当前卡的轴数

参　数：CardNo　　控制卡卡号

　　TotalAxis　　返回当前卡的轴数

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 IO 点数

参　数：CardNo　　控制卡卡号，0-7

TotalIn　　IO 输入数

TotalOut　　IO 输出数

返回值：错误代码

注　意：读取数量包括扩展模块上的 IO 数

dmc_get_total_adcnum(WORD CardNo,WORD* TotalIn,WORD* TotalOut)

功　能：获取控制卡 AD/DA 数

参　数：CardNo　　控制卡卡号，0-7

TotalIn　　本地 AD 输入数

TotalOut　　本地 DA 输出数

返回值：错误代码

注　意：读取数量包括扩展模块上的 AD/DA 数

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 Motion 软件中“参数设置”界面下，将各轴参数设置好，然后点击“参数文件操作”-“读取”将参数文件保存。最后在编写程序时，使用函数 dmc_download_configfile 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 Motion 软件中“帮助”->“固件升级”菜单下直接升级固件。

short dmc_check_if_crc_support(WORD CardNo)

功　能：检验接线盒是否支持通讯校验

参　数：CardNo　　控制卡卡号

返回值：1 接线盒支持通讯校验，0 接线盒不支持通讯校验

9.2　脉冲模式设置函数

short dmc_set_pulse_outmode(WORD CardNo, WORD axis, WORD outmode)

功　能：设置指定轴的脉冲输出模式

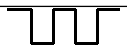
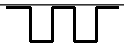
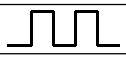
参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，　DMC5410A：0~3

outmode　　脉冲输出方式选择，其值如表 9.1 所示

返回值：错误代码

表 9.1　指令脉冲输出模式

输出脉冲类型 OUTMODE	正方向脉冲		负方向脉冲	
	PULSE 输出端	DIR 输出端	PULSE 输出端	DIR 输出端
0		高电平		低电平
1		高电平		低电平
2		低电平		高电平
3		低电平		高电平
4		高电平	高电平	
5		低电平	低电平	
6	AB 相输出，A 超前 B 90°		AB 相输出，B 超前 A 90°	

注意：在调用运动函数（如：dmc_vmove 等）输出脉冲之前，一定要根据驱动器接收脉冲的模式调用 dmc_set_pulse_outmode 设置控制卡脉冲输出模式

short dmc_get_pulse_outmode(WORD CardNo, WORD axis, WORD* outmode)

功　能：读取指定轴的脉冲输出模式设置

参　数：CardNo　　控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5， DMC5410A：0~3

outmode 返回脉冲输出方式

返回值：错误代码

9.3 回原点运动函数

short dmc_set_home_pin_logic(WORD CardNo, WORD axis, WORD org_logic, double filter)

功 能：设置 ORG 原点信号

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

org_logic ORG 信号有效电平，0：低有效，1：高有效

filter 保留参数，固定值为 0

返回值：错误代码

short dmc_get_home_pin_logic(WORD CardNo, WORD axis, WORD *org_logic, double *filter)

功 能：读取 ORG 原点信号设置

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

org_logic 返回设置的 ORG 信号有效电平

filter 保留参数

返回值：错误代码

short dmc_set_homemode(WORD CardNo, WORD axis, WORD home_dir, double vel_mode, WORD mode, WORD EZ_count)

功 能：设置回原点模式

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

home_dir 回零方向，0：负向，1：正向

vel_mode 回零速度模式：

0：低速回零，即以本指令前面的 dmc_set_profile_unit 函数设置的

起始速度运行

1：高速回零，即以本指令前面的 dmc_set_profile_unit 函数设置的最大速度运行

mode 回零模式：

0：一次回零

1：一次回零加回找

2：二次回零

3：原点加同向 EZ

4：单独记一个 EZ

5：原点加反向 EZ

6：原点锁存

7：原点锁存加同向 EZ

8：单独记一个 EZ 锁存

9：原点锁存加反向 EZ

10. 一次限位回零

11. 一次限位回零加反找

12. 二次限位回零

EZ_count 保留参数

返回值：错误代码

注　意：原点信号的反找速度是 dmc_set_profile_unit 的起始速度。

short dmc_get_homemode(WORD CardNo, WORD axis, WORD* home_dir, double* vel, WORD* mode, WORD* EZ_count)

功　能：读取回原点模式

参　数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

home_dir 返回回零方向

vel 返回回零速度模式

mode 返回回零模式

EZ_count 保留参数

返回值：错误代码

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

功　能：设置回零遇限位是否反找
参　数：CardNo　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　Enable　　使能是否遇限位反找　0：不使能反找 1：使能反找
返回值：错误代码
说　明：设置回零过程当中遇限位是否反找，卡初始化后配置一次即可，以后均不用配置（卡断电需重新配置）。DMC5X10 系列卡默认使能限位反找功能，如需关闭限位反找功能，则使用此函数关闭。**限位反找功能只针对非限位回零方式起作用。**

short dmc_get_home_el_return(WORD CardNo,WORD axis,WORD *enable)
功　能：读取回零遇限位是否反找
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　Enable　　读取遇限位反找使能状态　0：不使能反找 1：使能反找
返回值：错误代码

short dmc_set_home_position_unit(WORD CardNo,WORD axis,WORD mode,double position)
功　能：设置回零偏移量及清零模式
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　mode　　清零模式，0：不清零，1：回零完成后清零，再偏移；2：回零以及偏移完成后清零
　　position　　偏移量（正值正向偏移；负值负向偏移；0 无偏移）。正限位回零只能设置负向偏移，负限位回零只能设置正向偏移
返回值：错误代码

short dmc_get_home_position_unit (WORD CardNo, WORD axis, WORD * mode, double * position)
功　能：读取回零偏移量及清零模式
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：

0~5，DMC5410A：0~3

mode 读取清零模式

position 读取回零偏移量

返回值：错误代码

short dmc_set_el_ret_deviation(WORD CardNo, WORD axis, WORD enable， WORD deviation)

功 能：设置限位反找偏移距离

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

enable: 使能：1 使能 0 禁止

deviation 偏移距离,单位：pulse

返回值：错误代码

说 明：设置回零过程当中遇限位反找时，反向搜索原点信号的距离，到达设置的偏移距离，仍未找到原点信号，则回零失败，用于保护机台安全。**限位反找偏移距离只针对非限位回零方式起作用。**

short dmc_get_el_ret_deviation(WORD CardNo, WORD axis, WORD* enable, double* deviation)

功 能：读取限位反找偏移距离

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

enable: 回读的使能状态，1 使能 0 禁止

deviation 回读的偏移距离

返回值：错误代码

short dmc_home_move(WORD CardNo,WORD axis)

功 能：回原点运动

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

返回值：错误代码

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

功　能：读取回零执行状态
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　state　　回零执行状态，1：回零完成，0：回零未完成
返回值：错误代码

9.4　原点锁存和 EZ 锁存函数

short dmc_set_homelatch_mode(WORD CardNo, WORD axis, WORD enable, WORD logic, WORD source)
功　能：设置原点锁存模式
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7,DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　原点锁存使能，0：禁止，1：允许
　　logic　　触发方式，0：下降沿，1：上升沿
　　source　　位置源选择，0：指令位置计数器，1：编码器计数器
返回值：错误代码
注　意：DMC5C10 后四轴无原点锁存功能

short dmc_get_homelatch_mode(WORD CardNo, WORD axis, WORD* enable, WORD* logic, WORD* source)
功　能：读取原点锁存模式设置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　返回原点锁存使能状态
　　logic　　返回触发方式
　　source　　返回位置源选择
返回值：错误代码

short dmc_reset_homelatch_flag(WORD CardNo, WORD axis)
功　能：清除原点锁存标志

参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：错误代码

long dmc_get_homelatch_flag(WORD CardNo, WORD axis)
功　能：读取原点锁存标志
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：原点锁存标志，0：未锁存，1：锁存

long dmc_get_homelatch_value(WORD CardNo, WORD axis)
功　能：读取原点锁存值
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：锁存值，单位：pulse

WORD dmc_get_homelatch_value_unit(WORD CardNo, WORD axis,double* pos)
功　能：读取原点锁存值
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　pos　　锁存值，单位：unit
返回值：错误码

short dmc_set_ezlatch_mode(WORD CardNo, WORD axis, WORD enable, WORD logic, WORD source)
功　能：设置 EZ 锁存模式
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10:0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　EZ 锁存使能，0：禁止，1：允许

logic　　触发方式，0：下降沿，1：上升沿
source　　位置源选择，0：指令位置计数器，1：编码器计数器
返回值：错误代码
注　意：DMC5C10 后四轴无 EZ 锁存功能

short dmc_get_ezlatch_mode(WORD CardNo, WORD axis, WORD* enable, WORD* logic, WORD* source)
功　能：读取原点锁存模式设置
参　数：CardNo　　控制卡卡号
axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
enable　　返回 EZ 锁存使能状态
logic　　返回触发方式
source　　返回位置源选择
返回值：错误代码

short dmc_reset_ezlatch_flag(WORD CardNo, WORD axis)
功　能：清除 EZ 锁存标志
参　数：CardNo　　控制卡卡号
axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：错误代码

long dmc_get_ezlatch_flag(WORD CardNo, WORD axis)
功　能：读取 EZ 锁存标志
参　数：CardNo　　控制卡卡号
axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：EZ 锁存标志，0：未锁存，1：锁存

long dmc_get_ezlatch_value(WORD CardNo, WORD axis)
功　能：读取 EZ 锁存值
参　数：CardNo　　控制卡卡号
axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，

DMC5410A：0~3
返回值：锁存值，单位：pulse

WORD dmc_get_ezlatch_value_unit(WORD CardNo, WORD axis,double* pos)
功　能：读取 EZ 锁存位置值
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　pos　　EZ 锁存位置值，单位：unit
返回值：错误码

9.5　限位开关设置函数

short dmc_set_el_mode(WORD CardNo, WORD axis, WORD el_enable, WORD el_logic, WORD el_mode)
功　能：设置 EL 限位信号
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　el_enable　　EL 信号的使能状态：
　　0：正负限位禁止
　　1：正负限位允许
　　2：正限位禁止、负限位允许
　　3：正限位允许、负限位禁止
　　el_logic　　EL 信号的有效电平：
　　0：正负限位低电平有效
　　1：正负限位高电平有效
　　2：正限位低有效，负限位高有效
　　3：正限位高有效，负限位低有效
　　el_mode　　EL 制动方式：
　　0：正负限位立即停止
　　1：正负限位减速停止
　　2：正限位立即停止，负限位减速停止
　　3：正限位减速停止，负限位立即停止

返回值：错误代码

short dmc_get_el_mode(WORD CardNo,WORD axis, WORD *el_enable, WORD *el_logic, WORD *el_mode)

功　能：读取 EL 限位信号设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

el_enable　　返回设置的 EL 信号使能状态

el_logic　　返回设置的 EL 信号有效电平

el_mode　　返回 EL 制动方式

返回值：错误代码

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

功　能：设置软限位

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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：unit

P_limit　　正限位位置，单位：unit

返回值：错误代码

注　意：正、负限位位置可为正数也可为负数，但正限位位置应大于负限位位置

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

功能：读取软限位设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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 unit
P_limit　　返回正限位 unit
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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
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

9.6 位置计数器控制函数

short dmc_set_position(WORD CardNo, WORD axis, long current_position)
功　能：设置指定轴的指令脉冲位置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　current_position　　指令脉冲位置，单位：pulse
返回值：错误代码

long dmc_get_position(WORD CardNo, WORD axis)
功　能：读取指定轴的指令脉冲位置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：指令脉冲位置，单位：pulse

9.7 运动状态检测及控制相关函数

double　dmc_read_current_speed(WORD CardNo, WORD axis)
功 能：读取指定轴的当前速度值
参 数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：指定轴的速度，单位：pulse/s
注　意：执行插补运动时，该函数读取的速度为各轴分量速度；轴负向运动时，回读为负值

short dmc_LinkState(WORD CardNo, WORD* State)
功　能：检测主卡与接线盒的通讯连接状态
参　数：CardNo　　控制卡卡号
　　State　　连接状态，0：连接，1：断开
返回值：错误代码

short　dmc_check_done(WORD CardNo, WORD axis)

功　能：检测指定轴的运动状态

参　数：CardNo　　控制卡卡号

　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5

DMC5410A：0~3

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：0~3）

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5

DMC5410A：0~3

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
8	INP	1：表示伺服到位信号 INP 为 ON；　0：OFF
9	EZ	1：表示 EZ 信号为 ON；　0：OFF
10	RDY	1：表示伺服准备信号 RDY 为 ON； 0：OFF
11	DSTP	1：表示减速停止信号 DSTP 为 ON； 0：OFF

位号	信号名称	描述
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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

stop_mode　　制动方式，0：减速停止，1：紧急停止

返回值：错误代码

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：0~3）

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

long dmc_get_target_position(WORD CardNo, WORD axis)

功　能：读取正在运动轴的目标位置（绝对坐标）

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

返回值：目标位置，单位：pulse

9.8 单轴运动速度曲线设置函数

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

功　能：设置单轴运动速度曲线

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Min_Vel　　起始速度，单位：pulse/s (最大值为 4M)

Max_Vel　　最大速度，单位：pulse/s (最大值为 4M)

Tacc　　加速时间，单位：s (最小值为 0.001s)

Tdec　　减速时间，单位：s (最小值为 0.001s)

Stop_Vel　　停止速度，单位：pulse/s (最大值为 4M)

返回值：错误代码

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

功　能：读取单轴速度曲线参数

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Min_Vel　　返回起始速度，单位：pulse/s

Max_Vel　　返回最大速度，单位：pulse/s

Tacc　　返回加速时间，单位：s

Tdec　　返回减速时间，单位：s

Stop_Vel　　返回停止速度，单位：pulse/s

返回值：错误代码

short　dmc_set_acc_profile(WORD CardNo, WORD axis, double Min_Vel, double Max_Vel, double acc, double dec, double Stop_Vel)

功　能：设置单轴速度曲线参数，加速度模式

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Min_Vel　　起始速度，单位：pulse/s (最大值为 4M)
Max_Vel　　最大速度，单位：pulse/s (最大值为 4M)
acc　　加速度，单位：pulse /（s²）
dec　　减速度，单位：pulse /（s²）
Stop_Vel　　停止速度，单位：pulse/s (最大值为 4M)
返回值：错误代码

short　dmc_get_acc_profile(WORD CardNo, WORD axis, double *Min_Vel, double *Max_Vel, double *acc, double *dec, double * Stop_Vel)
功　能：读取单轴速度曲线参数，加速度模式
参　数：CardNo　　控制卡卡号
axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
Min_Vel　　返回起始速度，单位：pulse/s
Max_Vel　　返回最大速度，单位：pulse/s
acc　　返回加速度，单位：pulse / s²
dec　　返回减速度，单位：pulse / s²
Stop_Vel　　返回停止速度，单位：pulse/s
返回值：错误代码

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5
DMC5410A：0~3
s_mode　　保留参数，固定值为 0
s_para　　S 段时间，单位：s；范围：0~1 s
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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：

0~5

DMC5410A：0~3

s_mode　保留参数

s_para　返回设置的 S 段时间

返回值：错误代码

9.9　单轴运动函数

short dmc_pmove(WORD CardNo, WORD axis, long Dist, WORD posi_mode)

功　能：指定轴点位运动

参　数：CardNo　控制卡卡号

axis　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Dist　目标位置，单位：pulse

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：当运动模式为相对坐标模式时，目标位置大于 0 时正向运动，小于 0 时反向运动

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5

DMC5410A：0~3

dir　运动方向，0：负方向，1：正方向

返回值：错误代码

short　dmc_change_speed(WORD CardNo, WORD axis, double Curr_Vel, double Taccdec)

功　能：在线改变指定轴的当前运动速度

参　数：CardNo　控制卡卡号

axis　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Curr_Vel　改变后的运动速度，单位：pulse/s

Taccdec　保留参数，固定值为 0

返回值：错误代码

注　意：1）该函数适用于单轴运动中的变速

2）变速一旦成立，该轴的默认运行速度将会被改写为 Curr_Vel，也即当调用 dmc_get_profile 回读速度参数时会发生与 dmc_set_profile 所设置的不一致的现象

3）在连续运动中 Curr_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 Curr_Vel 只允许正值

short　dmc_reset_target_position(WORD CardNo, WORD axis, long dist, WORD posi_mode)

功　能：在线改变指定轴的当前目标位置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

dist　　目标位置，单位：pulse

posi_mode　　保留参数，固定值为 0

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 dist 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

short dmc_update_target_position(WORD CardNo, WORD axis, long dist, WORD posi_mode)

功　能：强行改变指定轴的当前目标位置（在线/非在线）

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

dist　　目标位置，单位：pulse

posi_mode　　保留参数，固定值为 0

返回值：错误代码

注　意：1）该函数适用于指定轴停止状态或点位运动中的变位

2）参数 dist 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

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

功　能：配置原点信号（ORG）触发在线变速变位置参数

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：

0~5，DMC5410A：0~3

tar_vel 改变后的运动速度，仅允许正值，单位:pulse/s

tar_rel_pos 相对原点信号触发处的偏移量(相对距离)，单位:pulse

trig_mode 触发方式 0：下降沿触发，1：上升沿触发

source 位置锁存源 ：0 指令位置（固定）

注 意：变速变位可同时执行，速度有变化则执行变速，位置有变速则执行变位；使能后，则对应的轴回零会失效；需关闭该功能才可正常回零；只适用于 pmove 运动；

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

功 能：读取配置的原点信号(ORG)触发在线变速变位置参数

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

tar_vel 改变后的运动速度，仅允许正值，单位:pulse/s

tar_rel_pos 相对原点信号触发处的偏移量(相对距离)，单位:pulse

trig_mode 触发方式 0：下降沿触发，1：上升沿触发

source 位置锁存源 ：0 指令位置（固定）

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

功 能：原点信号(ORG)触发在线变速变位置使能

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Enable 使能 0：功能关闭；1：多次触发，2：单次触发

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

功 能：读取原点信号(ORG)触发在线变速变位置的状态

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Enable 使能； 0：功能关闭；1：多次触发，2：单次触发

trig_num 原点信号触发的次数

trig_pos　　触发时对应的位置，单位：pulse

注　意：调用 pmove 的时候会自动将触发次数和触发位置清零

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

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

参　数：CardNo　　指定卡号

axis　　指定轴号

dist　　终点位置,单位：pulse

Min_Vel　　起始速度

Max_Vel　　最大速度

Tacc　　加速时间

Tdec　　减速时间

stop_Vel　　停止速度

s_para　　平滑时间，单位：s，范围[0~0.5]

posi_mode　　运动模式，0：相对模式，1：绝对模式

返回值：错误代码

short dmc_t_pmove_extern(WORD CardNo,WORD axis, double mid_pos, double target_pos, double Min_Vel, double Max_Vel, double stop_Vel, double acc, double dec, WORD posi_mode)

功　能：实现 profile，pmove 整合，缩短指令时间，并且实现软着陆

参　数：CardNo　　指定卡号

axis　　指定轴号

mid _pos　　第一段的终点位置,单位：pulse

aim_pos　　第二段的终点位置,单位：pulse

Min_Vel　　起始速度

Max_Vel　　最大速度

stop_Vel　　停止速度

acc　　加速时间

dec　　减速时间

posi_mode　　运动模式，0：相对模式，1：绝对模式

返回值：错误代码

注　意：软着陆不支持 S 型速度规划

short dmc_update_target_position_extern(WORD CardNo,WORD axis, double mid_pos, double aim_pos,double vel,WORD posi_mode)

功　能：强行改变指定轴的当前目标位置并且实现软着陆

参　数：CardNo　　指定卡号
axis　　指定轴号
mid _pos　　第一段的终点位置,单位：pulse
aim_pos　　第二段的终点位置,单位：pulse
vel:　　保留参数，固定值为 0
posi_mode　　保留参数，固定值为 0

返回值：错误代码

注　意：1）该函数适用于指定轴停止状态或点位运动中的变位，并且实现软着陆

2）参数 mid _pos，aim_pos 为绝对坐标位置值，无论当前的运动模式为绝对坐标还是相对坐标模式。mid _pos 以速度规划指令中的最大速度运行，aim_pos 以速度规划指令中的停止速度运行。

short dmc_t_pmove_extern_softstart_unit(WORD CardNo, WORD axis, double MidPos, double TargetPos, double start_Vel, double Max_Vel, double stop_Vel, DWORD delay_ms, double Max_vel2,double stop_vel2, double acc_time, double dec_time, WORD posi_mode)

功　能：实现 profile，pmove 整合，缩短指令时间，并且实现软启动

参　数：CardNo　　指定卡号
axis　　指定轴号
MidPos　　第一段 pmove 终点位置,单位：pulse
TargetPos　　第二段 pmove 终点位置,单位：pulse
start_Vel　　第一段 pmove 起始速度
Max_Vel　　第一段 pmove 最大速度
stop_Vel　　第一段 pmove 停止速度
delay_ms　　第一阶段完成后延迟时间（单位：毫秒）0-100s
Max_vel2　　第二段 pmove 最大速度
stop_vel2　　第二段 pmove 停止速度
acc_time　　加速时间(单位：s)，范围：0.001s~100s
dec_time　　减速时间(单位：s)，范围：0.001s~100s
posi_mode　　运动模式，0：相对模式，1：绝对模式

返回值：错误代码

注　意：软启动不支持 S 型速度规划

9.10 PVT 运动函数

short dmc_PttTable(WORD CardNo, WORD axis, DWORD count, double *pTime, long *pPos)

功　能：向指定数据表传送数据，采用 PTT 模式

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

count　　数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos　　数据点位置数组，单位：pulse；数组长度：count

返回值：错误代码

注 意：（1）DMC5C10 后四轴无 PVT 功能。

（2）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

（3）调用该指令向数据表中传递数据时，会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_PtsTable (WORD CardNo, WORD axis, DWORD count, double *pTime, long *pPos, double *pPercent)

功　能：向指定数据表传送数据，采用 PTS 模式

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

count　　数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos　　数据点位置数组，单位：pulse；数组长度：count

pPercent　　数据点百分比数组，百分比的取值范围：[0,100]；数组长度：count

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_PvtTable(WORD CardNo, WORD axis, DWORD count, double *pTime, long *pPos, double *pVel)

功　能：向指定数据表传送数据，采用 PVT 模式

参　数：CardNo　控制卡卡号

axis　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

count　数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime　数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos　数据点位置数组，单位：pulse；数组长度：count

pVel　数据点速度数组，单位：pulse/s；数组长度：count

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表

short dmc_PvtsTable (WORD CardNo, WORD axis, DWORD count, double *pTime, long *pPos, double velBegin, double velEnd)

功　能：向指定数据表传送数据，采用 PVTS 模式

参　数：CardNo　控制卡卡号

axis　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

count　数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime　数据点时间数组，单位：s（精度：ms）；数组长度：count

pPos　数据点位置数组，单位：pulse；数组长度：count

velBegin　设置的第一点的速度，单位：pulse/s

velEnd　设置的最后一点的速度，单位：pulse/s

返回值：错误代码

注 意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该指令向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应

当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short dmc_PvtMove(WORD CardNo, WORD AxisNum, WORD* AxisList)
功　能：启动 PVT 运动
参　数：CardNo　　控制卡卡号
　　AxisNum　　轴数
　　AxisList　　轴列表
返回值：错误代码

9.11　伺服驱动专用接口函数

short　dmc_write_sevon_pin(WORD CardNo, WORD axis, WORD on_off)
功　能：控制指定轴的伺服使能端口的输出
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
　　on_off　　设置伺服使能端口电平，0：低电平，1：高电平
返回值：错误代码

short　dmc_read_sevon_pin(WORD CardNo, WORD axis)
功　能：读取指定轴的伺服使能端口的电平
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
返回值：伺服使能端口电平，0：低电平，1：高电平

short　dmc_read_rdy_pin(WORD CardNo, WORD axis)
功　能：读取指定轴的 RDY 端口的电平
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
返回值：RDY 端口电平，0：低电平，1：高电平

short　dmc_set_inp_mode(WORD CardNo, WORD axis, WORD enable, WORD inp_logic)

功　能：设置指定轴的 INP 信号
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　INP 信号使能，0：禁止，1：允许
　　inp_logic　　INP 信号的有效电平，0：低有效，1：高有效
返回值：错误代码
注意：当使能 INP 信号功能后，只有在 INP 信号为有效状态时，对应的轴才能进行运动，否则此时检测轴的状态是正在运行（即对轴运动作限制）

short　dmc_get_inp_mode(WORD CardNo, WORD axis, WORD *enable, WORD *inp_logic)
功　能：读取指定轴的 INP 信号设置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　返回 INP 信号使能状态
　　inp_logic　　返回设置的 INP 信号有效电平
返回值：错误代码

short　dmc_set_alm_mode(WORD CardNo, WORD axis, WORD enable, WORD alm_logic, WORD alm_action)
功　能：设置指定轴的 ALM 信号
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　enable　　ALM 信号使能，0：禁止，1：允许
　　alm_logic　　ALM 信号的有效电平，0：低有效，1：高有效
　　alm_action　　ALM 信号的制动方式，0：立即停止（只支持该方式）
返回值：错误代码

short dmc_get_alm_mode(WORD CardNo, WORD axis, WORD *enable, WORD *alm_logic, WORD *alm_action)
功　能：读取指定轴的 ALM 信号设置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：

0~5，DMC5410A：0~3
enable 返回 ALM 信号使能状态
alm_logic 返回设置的 ALM 信号有效电平
alm_action 返回 ALM 信号的制动方式
返回值：错误代码

short dmc_write_erc_pin(WORD CardNo, WORD axis, WORD sel)
功 能：控制指定轴的 ERC 信号输出
参 数：CardNo 控制卡卡号
axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
sel 输出电平，0：低电平，1：高电平
返回值：错误代码

short dmc_read_erc_pin(WORD CardNo, WORD axis)
功 能：读取指定轴的 ERC 端口电平
参 数：CardNo 控制卡卡号
axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：ERC 端口电平，0：低电平，1：高电平

9.12 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：0~15
返回值：指定的输入端口电平：0：低电平，1：高电平

short dmc_read_inbit_ex(WORD CardNo, WORD bitno，WORD *status)
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：0~15
status 指定的输入端口电平：0：低电平，1：高电平

返回值：错误代码

说　明：实现功能同 dmc_read_inbit 函数

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：0~15

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　　输出端口号，取值范围：0~15

返回值：指定输出端口的电平，0：低电平，1：高电平

注　意：DMC5410A 中 OUT12、0UT13 输出口无效

short dmc_read_outbit_ex(WORD CardNo, WORD bitno，WORD *status)

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　　输出端口号，取值范围：0~15

status　　指定输出端口的电平：0：低电平，1：高电平

返回值：错误代码

说　明：实现功能同 dmc_read_outbit 函数

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　　IO 组号，取值范围：0、1

返回值：全部输入端口的电平

DMC5C10/DMC5810/5610/5410A：bit0~bit31 的定义见表 9.3

short dmc_read_inport_ex(WORD CardNo, WORD portno，DWORD *status)

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 0-7

portno　　IO 组号，取值范围：0、1

status　　全部输入端口的电平，bit0~bit31 的定义见表 9.3

返回值：错误代码

表 9.3　dmc_read_inport/ dmc_read_inport_ex 函数返回值各位的定义表

第 0 组（portno 参数）			第 1 组（portno 参数）		
函数返回值的 bit	输入口号	输入口名称	函数返回值的 bit	输入口号	输入口名称
0	0	IN0	0	32	ORG0
1	1	IN1	1	33	ORG1
2	2	IN2	2	34	ORG2
3	3	IN3	3	35	ORG3
4	4	IN4	4	36	ORG4
5	5	IN5	5	37	ORG5
6	6	IN6	6	38	ORG6
7	7	IN7	7	39	ORG7
8	8	IN8	8	40	ALM0
9	9	IN9	9	41	ALM1
10	10	IN10	10	42	ALM2
11	11	IN11	11	43	ALM3
12	12	IN12	12	44	ALM4
13	13	IN13	13	45	ALM5
14	14	IN14/LTC0	14	46	ALM6
15	15	IN15/LTC1	15	47	ALM7
16	16	EL0+	16	48	RDY0
17	17	EL1+	17	49	RDY1
18	18	EL2+	18	50	RDY2
19	19	EL3+	19	51	RDY3
20	20	EL4+	20	52	RDY4
21	21	EL5+	21	53	RDY5
22	22	EL6+	22	54	RDY6
23	23	EL7+	23	55	RDY7
24	24	EL0-	24	56	INP0
25	25	EL1-	25	57	INP1
26	26	EL2-	26	58	INP2
27	27	EL3-	27	59	INP3
28	28	EL4-	28	61	INP4
29	29	EL5-	29	61	INP5
30	30	EL6-	30	62	INP6
31	31	EL7-	31	63	INP7

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
返回值：全部输出口的电平，bit0~bit31 的定义见表 9.4

short dmc_read_outport_ex(WORD CardNo, WORD portno，DWORD* status)
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
　　status　　全部输出口的电平，bit0~bit31 的定义见表 9.4
返回值：错误代码
说　明：实现功能同 dmc_read_outport 函数

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
　　port_value　　bit0~bit31 的定义见表 9.4
返回值：错误代码

表 9.4　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	函数返回值的 bit	输出口号	输出口名称
0	0	OUT0	16	16	SEVON0
1	1	OUT1	17	17	SEVON1
2	2	OUT2	18	18	SEVON2
3	3	OUT3	19	19	SEVON3
4	4	OUT4	20	20	SEVON4
5	5	OUT5	21	21	SEVON5
6	6	OUT6	22	22	SEVON6
7	7	OUT7	23	23	SEVON7
8	8	OUT8	24	24	ERC0
9	9	OUT9	25	25	ERC1
10	10	OUT10	26	26	ERC2
11	11	OUT11	27	27	ERC3
12	12	OUT12/CMP0	28	28	ERC4
13	13	OUT13/CMP1	29	29	ERC5
14	14	OUT14/CMP2	30	30	ERC6
15	15	OUT15/CMP3	31	31	ERC7

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：0~15

reverse_time　　延时翻转时间，单位：s

返回值：错误代码

注　意：1）该函数只能对 OUT0~15 端口进行操作

2）当延时翻转时间参数设置为小于 0.001 时，此时延时翻转时间将为无限大

short dmc_set_output_status_repower（WORD CardNo, WORD enable）

功　能：控制卡接线盒断电重新上电输出口是否保持断电前状态

参　数：CardNo　　控制卡卡号

enable　　使能配置，1 保持，0 复位

返回值：错误代码

short dmc_set_io_count_mode(WORD CardNo, WORD bitno, WORD mode, double filter_time)

功　能：设置 IO 计数模式；DMC5C10/DMC5810/5610/5410A 卡专用

参　数：CardNo　　控制卡卡号

bitno　　输入端口号，取值范围：0~15

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

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：0~15

mode　　返回 IO 计数模式

filter_time　　返回滤波时间，单位：s

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：0~15
　　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：0~15
　　CountValue　　返回 IO 计数值
返回值：错误代码

9.13　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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　inmode　　手轮输入方式，0：A、B 相位正交信号(四倍频模式)；1：脉冲+方向信号
　　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
注　意：DMC5C10/DMC5410A 只有一个低速手轮通道，DMC5810/5610 有高、低速两种手轮通道，手轮外部硬件接口只有一个，通过该函数可以设置这个手轮通道对应的轴号（即每次只能控制一个轴运动），使用不同型号控制卡时请注意设置正确的手轮通道

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5

DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5

DMC5410A：0~3

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_channel(WORD CardNo,WORD index)

功 能：手轮通道选择设置；DMC5C10/DMC5810/5610/5410A 卡专用

参 数：CardNo 控制卡卡号

index 0：高速通道（默认值）

1：低速通道

返回值：错误代码

手轮通道的说明：使用高速通道与低速通道的效果是一样的，其区别是：高速通道是通过 ACC3800/3600 接线盒 CN18 口连接到控制卡；低速通道是通过 ACC-XC00/ACC3800/3600/ACC-X400 接线盒 CN17 口连接到控制卡

short dmc_get_handwheel_channel(WORD CardNo, WORD* index)

功 能：读取手轮通道选择设置

参 数：CardNo 控制卡卡号

index 返回设置的手轮通道

返回值：错误代码

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：A、B 相位正交信号(四倍频模式)；1：脉冲+方向信号
　　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

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

short dmc_set_handwheel_inmode_decimals(WORD CardNo,WORD axis,WORD inmode,double 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，取值范围：DMC5COO：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　Inmode　手轮输入方式，0：A、B 相位正交信号；1：脉冲+方向信号
　　multi　　手轮倍率，可以为小数值，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
注　意：DMC5C10/DMC5410A 只有一个低速手轮通道，DMC5810/5610 有高、低速两种手轮通道，手轮外部硬件接口只有一个，通过该函数可以设置这个手轮通道对应的轴号（即每次只能控制一个轴运动），使用不同型号控制卡时请注意设置正确的手轮通道

short dmc_get_handwheel_inmode_decimals(WORD CardNo,WORD axis,WORD *inmode,double

*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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Inmode　返回手轮输入方式

Multi　返回手轮倍率，可以为小数值

Vh　保留参数

返回值：错误代码

short　dmc_set_handwheel_inmode_extern_decimals(WORD　CardNo,WORD　inmode,WORD AxisNum,WORD* AxisList,double* 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：A、B 相位正交信号；1：脉冲+方向信号

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:值可以为小数，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

short　dmc_get_handwheel_inmode_extern_decimals(WORD　CardNo,WORD*　inmode,WORD* AxisNum,WORD* AxisList,double *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，可以为小数值

返回值：错误代码

9.14 编码器函数

short dmc_set_counter_inmode(WORD CardNo, WORD axis, WORD mode)
功 能：设置编码器的计数方式
参 数：CardNo 控制卡卡号
axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
mode 编码器的计数方式：
0：非 A/B 相(脉冲/方向)
1：1×A/B
2：2×A/B
3：4×A/B
返回值：错误代码
注意：DMC5C10 后四轴无编码器计数功能

short dmc_get_counter_inmode(WORD CardNo, WORD axis, WORD *mode)
功 能：读取编码器的计数方式
参 数：CardNo 控制卡卡号
axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
mode 返回编码器的计数方式
返回值：错误代码

short dmc_set_encoder_dir(WORD CardNo, WORD axis,WORD dir)
功 能：设置编码器方向
参 数：CardNo 控制卡卡号 0-7
axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
dir 编码器配置方向：0：A 在前为正 1：B 在前为正
返回值：错误代码
注 意：该配置需要控制卡 FPGA X028 及更高版本支持

short dmc_set_encoder(WORD CardNo, WORD axis, long encoder_value)
功 能：设置指定轴编码器脉冲计数值

参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　encoder_value　　编码器计数值，单位：pulse
返回值：错误代码
说　明：此函数 axis 参数为 8 时可以设置手轮编码器计数值，DMC5C10/5410A 无高速手轮通道，不支持设置手轮编码器计数值

long　dmc_get_encoder(WORD CardNo, WORD axis)
功　能：读取指定轴编码器脉冲计数值
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3
返回值：编码器计数值，单位：pulse
说　明：此函数 axis 参数为 8 时可以读手轮编码器计数值，DMC5410A 不支持读取手轮编码器计数值；其他型号卡均支持读取手轮编码器计数值

short dmc_set_ez_mode(WORD CardNo, WORD axis, WORD ez_logic, WORD ez_mode, double filter)
功　能：设置指定轴的 EZ 信号
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　ez_logic　　EZ 信号有效电平，0：低有效，1：高有效
　　ez_mode　　EZ 模式，默认为 0，1 表示 EZ 触发编码器清零功能（电平触发）
　　filter　　保留参数，固定值为 0
返回值：错误代码

short dmc_get_ez_mode(WORD CardNo, WORD axis, WORD *ez_logic, WORD *ez_mode, double *filter)
功　能：读取指定轴的 EZ 信号设置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

ez_logic　　返回设置的 EZ 信号有效电平
ez_mode　　保留参数
filter　　保留参数
返回值：错误代码

9.15　软件锁存和高速位置锁存函数

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：配置软件锁存器
参　数：CardNo　　控制卡卡号
latch　　锁存器号，范围 0~3
ltc_enable　　使能锁存器
ltc_mode　　锁存模式，0-单次锁存，1-连续锁存
ltc_inbit　　锁存触发信号
ltc_logic　　锁存触发边沿，0-下降沿，1-上升沿，2-双边沿
filter　　滤波时间，单位：us
返回值：错误代码

short dmc_softltc_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，范围 0~3
ltc_enable　　使能锁存器
ltc_mode　　锁存模式，0-单次锁存，1-连续锁存
ltc_inbit　　锁存触发信号
ltc_logic　　锁存触发边沿，0-下降沿，1-上升沿，2-双边沿
filter　　滤波时间，单位：us
返回值：错误代码

short dmc_softltc_set_source(WORD CardNo,WORD latch,WORD axis,WORD ltc_source)
功　能：配置锁存源
参　数：CardNo　　控制卡卡号

latch 锁存器号，范围 0~3
axis 锁存对应轴号
ltc_source 配置锁存源，0-指令位置，1-编码器反馈位置
返回值：错误代码

short dmc_softltc_get_source(WORD CardNo,WORD latch,WORD axis,WORD *ltc_source)
功 能：读取锁存源配置
参 数：CardNo 控制卡卡号
latch 锁存器号，范围 0~3
axis 锁存对应轴号
ltc_source 配置锁存源，0-指令位置，1-编码器反馈位置
返回值：错误代码

short dmc_softltc_reset(WORD CardNo,WORD latch)
功 能：复位锁存器
参 数：CardNo 控制卡卡号
latch 锁存器号，范围 0~3
返回值：错误代码

short dmc_softltc_get_number(WORD CardNo,WORD latch,WORD axis,int *number)
功 能：读取锁存个数
参 数：CardNo 控制卡卡号
latch 锁存器号，范围 0~3
axis 锁存器对应的轴号
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，范围 0~3
axis 锁存器对应的轴号
value 读取锁存值，单位：unit
返回值：错误代码

short dmc_set_ltc_mode(WORD CardNo, WORD axis, WORD ltc_logic, WORD ltc_mode, double filter)

功　能：设置指定轴的 LTC 信号

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

ltc_logic　　LTC 信号的触发方式，0：下降沿锁存，1：上升沿锁存

ltc_mode　　保留参数，固定值为 0

filter　　保留参数，固定值为 0

返回值：错误代码

注意：DMC5C10 后四轴无高速锁存功能

short dmc_get_ltc_mode(WORD CardNo, WORD axis, WORD *ltc_logic, WORD *ltc_mode, double *filter)

功　能：读取指定轴的 LTC 信号设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

ltc_logic　　返回设置的 LTC 信号的触发方式

ltc_mode　　保留参数

filter　　保留参数

返回值：错误代码

short dmc_set_latch_mode(WORD CardNo, WORD axis, WORD all_enable, WORD latch_source, WORD triger_chunnel)

功　能：设置锁存方式

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5COO：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

all_enable　　锁存方式：

0：单次锁存

1：（保留）

2：连续锁存

3：触发延时急停

latch_source　　锁存源，0：指令位置计数器，1：编码器计数器

triger_chunnel　保留参数，固定值为 0

返回值：错误代码

注　意：

1)DMC5410A 只有一个高速锁存通道 LTC0，DMC5810/5610 有两个高速锁存通道 LTC0，LTC1；LTC0 锁存 0~3 号轴，LTC1 锁存 4~7 号轴。

2）DMC5C10/DMC5810/5610 中，触发延时急停模式只对 0 号轴及 4 号轴起作用；DAM5410A 中，触发延时急停模式只对 0 号轴起作用；LTC0 对应为 0 号轴，LTC1 对应为 4 号轴。

short dmc_get_latch_mode(WORD CardNo, WORD axis, WORD* all_enable, WORD* latch_source, WORD* triger_chunnel)

功　能：读取锁存方式

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

all_enable　　返回锁存方式设置

latch_source　　返回锁存源设置

triger_chunnel　　保留参数

返回值：错误代码

short dmc_set_latch_stop_time(WORD CardNo, WORD axis, long time)

功　能：设置 LTC 端口触发延时急停时间

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5410A：0，　DMC5C10/DMC5810/5610：0、4

time　　触发延时停止时间，单位：us，取值范围：1us~50ms

返回值：错误代码

注　意：DMC5C10/DMC5810/5610 中，触发延时急停模式只对 0 号轴及 4 号轴起作用；DMC5410A 中，触发延时急停模式只对 0 号轴起作用；LTC0 对应为 0 号轴，LTC1 对应为 4 号轴。

short dmc_get_latch_stop_time(WORD CardNo, WORD axis, long* time)

功　能：读取 LTC 端口触发延时急停时间

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5410A：0，DMC5C10/DMC5810/5610：0、4

time　　返回触发延时停止时间设置，单位：us

返回值：错误代码

short dmc_SetLtcOutMode(WORD CardNo, WORD axis, WORD enable, WORD bitno)

功　能：LTC 反相输出设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10/DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

enable　　使能状态：0：禁止，1：使能

bitno　　通用输出 IO 口号，取值范围：0~15

返回值：错误代码

注　意：当某输出端口作为 LTC 反相输出后，该端口将不能通过通用 IO 函数设置输出值

short dmc_GetLtcOutMode(WORD CardNo, WORD axis, WORD *enable, WORD* bitno)

功　能：读取 LTC 反相输出设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10/DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Enable　　返回使能状态：0：禁止，1：使能

bitno　　返回设置的通用输出 IO 口号

返回值：错误代码

long dmc_get_latch_value(WORD CardNo, WORD axis)

功　能：读取锁存值

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

返回值：锁存值

注　意：1）该函数既可以读编码器锁存值，也可以控制卡计数器的锁存值。

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_get_latch_value_unit(WORD CardNo, WORD axis,double* latchPos)

功　能：读取锁存值

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

latchPos　锁存位置

返回值：错误码

short dmc_get_latch_flag(WORD CardNo, WORD axis)

功　能：读取锁存器已锁存个数

参　数：CardNo　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

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_reset_latch_flag(WORD CardNo, WORD axis)

功　能：复位指定卡的锁存器的标志位

参　数：CardNo　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：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。

9.16　低速位置比较函数

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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

注意：DMC5C10 后四轴无位置比较功能

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

　　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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

返回值：错误代码

short dmc_compare_add_point(WORD CardNo,WORD axis, long pos, WORD dir, WORD action, DWORD actpara)

功　能：添加一维位置比较点

参　数：CardNo　　控制卡卡号

　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5 DMC5410A：0~3

　　pos　　比较位置，单位：pulse

　　dir　　比较模式，0：小于等于，1：大于等于

　　action　　比较点触发功能编号，见表 9.5

　　actpara　　比较点触发功能参数，见表 9.5

返回值：错误代码

short dmc_compare_add_point_unit(WORD CardNo,WORD axis, double pos, WORD dir, WORD action, DWORD actpara)

功　能：添加一维位置比较点
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5
DMC5410A：0~3
　　pos　　比较位置，单位：unit
　　dir　　比较模式，0：小于等于，1：大于等于
　　action　　比较点触发功能编号，见表 9.5
　　actpara　　比较点触发功能参数，见表 9.5
返回值：错误代码

表 9.5　dmc_compare_add_point 函数 action, actpara 参数值

action	actpara	功能
1	IO 号	IO 置为高电平
2	IO 号	IO 置为低电平
3	IO 号	取反 IO
5	IO 号	输出 500us 脉冲
6	IO 号	输出 1ms 脉冲
7	IO 号	输出 10ms 脉冲
8	IO 号	输出 100ms 脉冲
13	轴号	停止指定轴
15	变速值（单位：pulse/s）	在线变速

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5
DMC5410A：0~3
　　pos　　返回当前比较点位置，单位：pulse
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_current_point_unit(WORD CardNo, WORD axis, double* pos)
功　能：读取当前一维比较点位置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，

DMC5410A：0~3

pos 返回当前比较点位置，单位：unit

返回值：错误代码

注意：当前位置被比较之后会被清除

short dmc_compare_get_points_runned(WORD CardNo, WORD axis, long* PointNum)

功 能：查询已经比较过的一维比较点个数

参 数：CardNo 控制卡卡号

axis 指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

PointNum 返回可以加入的比较点数

返回值：错误代码

short dmc_compare_set_config_extern (WORD CardNo, WORD enable, WORD cmp_source)

功 能：设置二维位置比较器

参 数：CardNo 控制卡卡号

enable 二维位置比较功能状态，0：禁止，1：使能

cmp_source 二维位置比较源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

short dmc_compare_get_config_extern(WORD CardNo, WORD* enable, WORD* cmp_source)

功 能：读取二维位置比较器设置

参 数：CardNo 控制卡卡号

enable 返回比较功能状态

cmp_source 返回比较源

返回值：错误代码

short dmc_compare_clear_points_extern(WORD CardNo)

功　能：清除已添加的所有二维位置比较点

参　数：CardNo　　控制卡卡号

返回值：错误代码

short dmc_compare_add_point_extern(WORD CardNo, WORD* axis, long* pos, WORD* dir, WORD action, DWORD actpara)

功　能：添加二维位置比较点

参　数：CardNo　　控制卡卡号

axis　　指定卡上的即将进行位置比较的轴列表(两个轴)

pos　　二维位置比较位置列表，单位：pulse

dir　　比较模式列表，0：小于等于，1：大于等于

action　　二维位置比较点触发功能编号，见表 9.6

actpara　　二维位置比较点触发功能参数，见表 9.6

返回值：错误代码

表 9.6　dmc_compare_add_point_ex 函数 action, actpara 参数值

action	actpara	功能
1	IO 号	IO 置为高电平
2	IO 号	IO 置为低电平
3	IO 号	取反 IO
5	IO 号	输出 500us 脉冲
6	IO 号	输出 1ms 脉冲
7	IO 号	输出 10ms 脉冲
8	IO 号	输出 100ms 脉冲
13	轴号	停止指定轴

short dmc_compare_add_point_extern_unit(WORD CardNo, WORD* axis, double* pos, WORD* dir, WORD action, DWORD actpara)

功　能：添加二维位置比较点

参　数：CardNo　　控制卡卡号

axis　　指定卡上的即将进行位置比较的轴列表(两个轴)

pos　　二维位置比较位置列表，单位：unit

dir　　比较模式列表，0：小于等于，1：大于等于

action　　二维位置比较点触发功能编号，见表 9.6

actpara　二维位置比较点触发功能参数，见表 9.6

返回值：错误代码

short dmc_compare_get_current_point_extern(WORD CardNo, long *pos)

功　能：读取当前二维位置比较点位置

参　数：CardNo　控制卡卡号

pos　返回当前二维位置比较点位置，单位：pulse

返回值：错误代码

short dmc_compare_get_current_point_extern_unit(WORD CardNo, double *pos)

功　能：读取当前二维位置比较点位置

参　数：CardNo　控制卡卡号

pos　返回当前二维位置比较点位置，单位：unit

返回值：错误代码

short dmc_compare_get_points_runned_extern(WORD CardNo, long *PointNum)

功　能：查询已经比较过的二维比较点个数

参　数：CardNo　控制卡卡号

PointNum　返回已经比较过的二维位置比较点数

返回值：错误代码

short dmc_compare_get_points_remained_extern(WORD CardNo, long *PointNum)

功　能：查询可以加入的二维比较点个数

参　数：CardNo　控制卡卡号

PointNum　返回可以加入的二维位置比较点数

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　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
cmp_source　比较位置源：0：指令位置计数器，1：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
2）DMC5C10 后四轴无高速位置比较功能

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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）

axis　　返回关联轴号设置

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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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_add_point_unit(WORD CardNo, WORD hcmp, double cmp_pos)

功　能：添加/更新一维高速比较位置

参　数：CardNo　　控制卡卡号

hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）

cmp_pos　　队列模式下：添加比较位置，单位：unit

线性模式下：更新起始比较位置，单位：unit

其他模式下：更新比较位置，单位：unit

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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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_set_liner_unit(WORD CardNo, WORD hcmp, double Increment, long Count)
功　能：设置一维高速比较线性模式参数
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
　　Increment　　位置增量值，单位：unit（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_unit(WORD CardNo, WORD hcmp, double* Increment, long* Count)
功　能：读取一维高速比较线性模式参数设置
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
　　Increment　　返回位置增量值设置，单位：unit
　　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　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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

short dmc_hcmp_get_current_state_unit(WORD CardNo, WORD hcmp, long *remained_points, double *current_point, long *runned_points)

功　能：读取一维高速比较参数

参　数：CardNo　　控制卡卡号

　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 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：unit

　　runned_points　　返回已比较点数

返回值：错误代码

short dmc_hcmp_fifo_set_mode(WORD CardNo,WORD hcmp, WORD fifo_mode)

功　能：启用缓存方式添加比较位置

参　数：CardNo　　控制卡卡号

　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）

　　fifo_mode　　是否启用缓存方式，0：不启用，1：启用

返回值：错误代码

备　注：高速比较模式设置为队列模式时，缓存方式添加比较位置才有效

short dmc_hcmp_fifo_get_mode(WORD CardNo,WORD hcmp, WORD* fifo_mode)

功　能：缓存方式添加比较位置模式回读

参　数：CardNo　　控制卡卡号

　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）

　　fifo_mode　　回读是否启用缓存方式，0：不启用，1：启用

返回值：错误代码

short dmc_hcmp_fifo_add_table(WORD CardNo,WORD hcmp, WORD num,double *cmp_pos)
功　能：按数组的方式批量添加比较位置
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
　　num　　数据点数，范围：1~25000
　　cmp_pos　　比较位置数组，单位：pulse
返回值：错误代码

short dmc_hcmp_fifo_clear_points(WORD CardNo,WORD hcmp)
功　能：清除比较位置,也会把 FPGA 的位置同步清除掉
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
返回值：错误代码

short dmc_hcmp_fifo_get_state(WORD CardNo,WORD hcmp,long *remained_points)
功　能：读取剩余缓存状态，上位机通过此函数判断是否继续添加比较位置
参　数：CardNo　　控制卡卡号
　　hcmp　　高速比较器，取值范围：0~3（对应硬件 CMP0~CMP3 端口）
　　remained_points　　剩余缓存
返回值：错误代码
备　注：控制卡最大 25000+127 个缓存点，如果添加 25000 个点，读剩余缓存为 127

9.18　异常信号接口函数

short dmc_set_emg_mode(WORD CardNo, WORD axis, WORD enable, WORD emg_logic)
功　能：设置 EMG 急停信号
参　数：CardNo　　控制卡卡号
　　axis　　DMC5C10 卡：指定轴号，取值范围：0~7
　　DMC5810 卡：指定轴号，取值范围：0~7
　　DMC5610 卡：指定轴号，取值范围：0~5
　　DMC5410A 卡：指定轴号，取值范围：0~3
　　enable　　允许/禁止信号功能，0：禁止，1：允许
　　emg_logic　　EMG 信号有效电平，0：低有效，1：高有效

返回值：错误代码

注意：DMC5C10 后四轴不支持异常停止功能

short dmc_get_emg_mode (WORD CardNo, WORD axis, WORD *enbale, WORD * logic)

功　能：读取 EMG 急停信号设置

参　数：CardNo　　控制卡卡号

axis　　DMC5C10 卡：指定轴号，取值范围：0~7

DMC5810 卡：指定轴号，取值范围：0~7

DMC5610 卡：指定轴号，取值范围：0~5

DMC5410A 卡：指定轴号，取值范围：0~3

enable　　返回 EMG 信号功能状态

logic　　返回设置的 EMG 信号有效电平

返回值：错误代码

short dmc_set_io_dstp_mode(WORD CardNo, WORD axis, WORD enable, WORD logic)

功　能：设置减速停止信号

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7,DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

enable　　允许/禁止硬件信号功能，0：禁止，1：允许

logic　　外部减速停止信号有效电平，0：低有效，1：高有效

返回值：错误代码

注　意：减速停止信号（DSTP）的减速时间由函数 dmc_set_dec_stop_time 设置

short dmc_get_io_dstp_mode(WORD CardNo, WORD axis, WORD *enable, WORD *logic)

功　能：读取减速停止信号设置

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

enable　　返回 DSTP 硬件信号功能状态

logic　　返回设置的外部减速停止信号有效电平

返回值：错误代码

short dmc_set_dec_stop_time(WORD CardNo, WORD axis, double stop_time)

功　能：设置异常减速停止时间
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　stop_time　　异常减速时间，单位：s
返回值：错误代码
注　意：点位运动当发生异常停止时，如：限位信号（软硬件）被触发、减速停止信号(DSTP)被触发等进行减速停止时，减速停止时间都为 dmc_set_dec_stop_time 函数里设置的减速时间。

short dmc_get_dec_stop_time(WORD CardNo, WORD axis, double *stop_time)
功　能：读取减速停止时间设置
参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
　　stop_time　　返回设置的减速时间，单位：s
返回值：错误代码

short dmc_set_vector_dec_stop_time(WORD CardNo, WORD Crd, double stop_time)
功　能：设置插补运动异常减速停止时间
参　数：CardNo　　控制卡卡号：0~7
　　Crd　　坐标系号，取值范围：0-3
　　stop_time　　异常减速时间，单位：s
返回值：错误代码
注　意：插补运动时，当异常信号产生，如：限位信号（软硬件）被触发、减速停止信号(DSTP)被触发，将进行减速停止，减速停止时间为此函数里设置的减速时间。若此函数设置的减速时间大于 dmc_set_vector_profile_unit 里面的减速时间，则取最短的减速时间。

short dmc_get_vector_dec_stop_time(WORD CardNo, WORD Crd, double* stop_time)
功　能：回读设置的插补运动异常减速停止时间
参　数：CardNo　　控制卡卡号：0~7
　　Crd　　坐标系号，取值范围：0-3
　　stop_time　　返回设置的减速时间，单位：s
返回值：错误代码

9.19 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　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5,DMC5410A：0~3

IoType　　指定轴的 IO 信号类型：

0：正限位信号，AxisIoInMsg_PEL

1：负限位信号，AxisIoInMsg_NEL

2：原点信号，AxisIoInMsg_ORG

3：急停信号，AxisIoInMsg_EMG

4：减速停止信号，AxisIoInMsg_DSTP

5：伺服报警信号，AxisIoInMsg_ALM

6：伺服准备信号，AxisIoInMsg_RDY（保留）

7：伺服到位信号，AxisIoInMsg_INP

MapIoType　　轴 IO 映射类型：

0：正限位输入端口，AxisIoInPort_PEL

1：负限位输入端口，AxisIoInPort_NEL

2：原点输入端口，AxisIoInPort_ORG

3：伺服报警输入端口，AxisIoInPort_ALM

4：伺服准备输入端口，AxisIoInPort_RDY

5：伺服到位输入端口，AxisIoInPort_INP

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 0~15 整数，表示该映射对应的具体通用输入端口号

2）当轴 IO 映射类型设置为 0~5 时，此参数可设置 0~7 整数，表示该映射所对应的具体轴号

3）设置该值为 255 表示取消轴 IO 映射关系

filter_time　　轴 IO 信号滤波时间，单位：s

返回值：错误代码

注　意：1）DMC5C10 后四轴支持轴 IO 映射功能

2）该函数可以实现对专用 IO 信号的硬件输入接口进行任意配置

3）dmc_set_axis_io_map 函数用法详见 7.14 节 轴 IO 映射功能的实现

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　　指定轴号，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

IoType　　轴 IO 信号类型

MapIoType　　返回轴 IO 映射类型

MapIoIndex　　返回轴 IO 映射索引号

filter_time　　返回轴 IO 信号滤波时间，单位：s

返回值：错误代码

short dmc_set_special_input_filter(WORD CardNo, double filter_time)

功　能：统一设置所有专用 IO 的滤波时间

参　数：CardNo　　控制卡卡号

filter_time　　轴 IO 信号滤波时间，单位：s

返回值：错误代码

9.20　虚拟 IO 映射函数

short dmc_set_io_map_virtual(WORD CardNo, WORD bitno, WORD MapIoType, WORD MapIoIndex, double filter_time)

功　能：设置虚拟 IO 映射关系

参　数：CardNo　　控制卡卡号

bitno　　虚拟 IO 口号，取值范围：0~15

MapIoType　　虚拟 IO 映射类型：

0：正限位输入端口，AxisIoInPort_PEL

1：负限位输入端口，AxisIoInPort_NEL

2：原点输入端口，AxisIoInPort_ORG

3：伺服报警输入端口，AxisIoInPort_ALM

4：伺服准备输入端口，AxisIoInPort_RDY

5：伺服到位输入端口，AxisIoInPort_INP
6：通用输入端口，AxisIoInPort_IO
7：急停信号输入端口，AxisIoInPort_EMG
8：通用输出端口

MapIoIndex 虚拟 IO 映射索引号：
1）当虚拟 IO 映射类型设置为 6 时，此参数可设置为 0~15 整数，表示该映射对应的具体通用输入端口号
2）当虚拟 IO 映射类型设置为 0~5 时，此参数可设置 0~7 整数，表示该映射所对应的具体轴号
3）设置该值为 255 表示取消虚拟 IO 映射关系

filter_time 虚拟 IO 信号滤波时间，单位：s

返回值：错误代码

注　意：1) 该函数可以实现专用通用 IO 输入接口的滤波功能
2）该函数可以实现专用 IO 作为虚拟通用 IO 使用
3）DMC5C10 后四轴支持虚拟 IO 映射功能

short dmc_get_io_map_virtual(WORD CardNo, WORD bitno, WORD* MapIoType, WORD* MapIoIn
dex, double* filter_time)

功　能：读取虚拟 IO 映射关系设置

参　数：CardNo 控制卡卡号
bitno 虚拟 IO 口号，取值范围：0~15
MapIoType 返回虚拟 IO 映射类型
MapIoIndex 返回虚拟 IO 映射索引号
filter_time 返回虚拟 IO 信号滤波时间，单位：s

返回值：错误代码

short dmc_read_inbit_virtual(WORD CardNo, WORD bitno)

功　能：读取滤波后的虚拟 IO 口电平状态

参　数：CardNo 控制卡卡号
bitno 虚拟 IO 口号，取值范围：0~15

返回值：指定的虚拟 IO 口电平：0：低电平，1：高电平

注　意：1）此函数需要配合虚拟 IO 映射功能使用。

2）dmc_read_inbit_virtual 与 dmc_read_inbit 函数的区别是：dmc_read_inbit 函数是不经过滤波直接读取硬件端口的电平状态，dmc_read_inbit_virtual 函数通过虚拟 IO 映射后读取相应端口滤波后的电平状态。

9.21 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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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。关于检测轴到位状态函数的用法详见章节 7.10 检测轴到位状态功能的实现

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

返回值：0：表示编码器位置在设定的目标位置的误差带之外

1：表示编码器位置在设定的目标位置的误差带之内

注　意：1）该函数只适用于单轴运动

2）检测函数请在 dmc_check_done 检测到轴停止后调用，函数调用后会等待轴到位后返回，如果调用函数 100ms 内未到位，函数超时返回认为不到位

short dmc_set_pulse_encoder_count_error(WORD CardNo,WORD axis,WORD error)

功能：设置脉冲计数值和编码器反馈值之间差值的报警阀值

输入参数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，　DMC5410A：0~3

error　　差值报警阀值

返回值：错误代码

short dmc_get_pulse_encoder_count_error(WORD CardNo,WORD axis,WORD *error)

功　能：回读脉冲计数值和编码器反馈值之间差值的报警阀值

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5 ，DMC5410A：0~3

error　　返回设定报警阀值，单位：unit

返回值：错误代码

short dmc_check_pulse_encoder_count_error(WORD CardNo,WORD axis,int* pulse_position, int* enc_position);

功　能：检查脉冲计数值和编码器反馈值之间差值是否超过报警阀值

参　数：CardNo　　控制卡卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5 ，DMC5410A：0~3
　　pulse_position　返回指令位置值，单位：unit
　　enc_position　　返回编码器位置值，单位：unit
返回值：　0：差值小于报警阀值 ，1：差值大于等于报警阀值

short　dmc_set_encoder_count_error_action_config(WORD　CardNo,WORD　enable,WORD stopmode)
功　能：检测指令位置与编码器偏差超过报警阀值时停止运动
参　数：CardNo　　控制卡卡号
　　Enable　　使能值，0：禁止，1：使能允许
　　Stopmode　　运动停止模式，0：减速停止，1：立即停止
返回值：错误代码

short　dmc_get_encoder_count_error_action_config(WORD　CardNo,WORD*　enable,WORD* stopmode)
功　能：回读检测指令位置与编码器偏差超过报警阀值时停止运动
参　数：CardNo　　控制卡卡号
　　Enable　　回读使能值，0：禁止，1：使能允许
　　Stopmode　回读运动停止模式，0：减速停止，1：立即停止
返回值：错误代码

9.22　CAN 扩展函数

short nmc_set_connect_state(WORD CardNo, WORD NodeNum, WORD state, WORD Baud)
功　能：设置 CAN 通讯状态
参　数：CardNo　　控制卡卡号
　　NodeNum　　CAN 节点数，取值范围：1~8
　　state　　设置通讯状态，0：断开，1：连接
　　Baud　　设置控制卡波特率
　　波特率参数：0,1,2,3,4,5
　　对应波特率：1000Kbps，800Kbps,500Kbps,250Kbps,125Kbps,100Kbps
返回值：错误代码

注　意：1）当关闭运动控制卡时，CAN 通讯不会被自动断开；当再次初始化运动控制卡时，CAN 通讯依然保持之前的状态。

2）当连接 CAN 通讯时，必须使用 nmc_get_connect_state 函数读取 CAN 的通讯状态，确认 CAN 通讯已正常连接。当连接出现异常时，可再次调用 nmc_set_connect_state 函数进行连接。

3）设置波特率时需保证控制卡波特率与模块波特率相对应。

short nmc_get_connect_state (WORD CardNo, WORD* NodeNum, WORD* state)

功　能：读取 CAN 通讯状态

参　数：CardNo　　控制卡卡号

NodeNum　　返回 CAN 节点数，取值范围：1~8

state　　返回 CAN-IO 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

short nmc_write_outbit(WORD CardNo, WORD NoteID, WORD IoBit,WORD IoValue)

功　能：设置指定 CAN-IO 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

IoBit　　输出端口号

IoValue　　输出电平，0：低电平，1：高电平

返回值：错误代码

short nmc_write_outbit_ex(WORD CardNo,WORD NoteID,WORD IoBit,WORD IoValue,WORD* state)

功　能：设置指定 CAN-IO 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

IoBit　　输出端口号

IoValue　　输出电平，0：低电平，1：高电平

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_write_outbit，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户操作输出后，需同时判断 CAN 的连接状态，确认输出是否执行正常。

short nmc_read_outbit(WORD CardNo, WORD NoteID, WORD IoBit,WORD* IoValue)

功　能：读取指定 CAN-IO 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

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_outbit_ex(WORD CardNo,WORD NoteID,WORD IoBit,WORD *IoValue,WORD* state)

功　能：读取指定 CAN-IO 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

IoBit　　输出端口号

IoValue　　指定输出端口的电平，0：低电平，1：高电平

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_read_outbit，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取输出后，需同时判断 CAN 的连接状态，确认输出读取是否正常。

short nmc_read_inbit(WORD CardNo, WORD NoteID, WORD IoBit,WORD* IoValue)

功　能：读取指定 CAN-IO 扩展模块的某个输入端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

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_inbit_ex(WORD CardNo,WORD NoteID,WORD IoBit,WORD *IoValue,WORD* state)

功　能：读取指定 CAN-IO 扩展模块的某个输入端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

IoBit 输入端口号
IoValue 指定输入端口的电平，0：低电平，1：高电平
state CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备 注：该函数实现功能同 nmc_read_inbit，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取输入后，需同时判断 CAN 的连接状态，确认输入读取是否正常。

short nmc_write_outport(WORD CardNo, WORD Node, WORD PortNo, DWORD outport_val)

功 能：设置指定 CAN-IO 扩展模块的输出端口的电平

参 数：CardNo 控制卡卡号
Node CAN 节点号，取值范围：1~8
PortNo 端口号，0 表示 0~31 号口，1 表示 32~63 号口
outport_val 输出值，bit0~bit32 的值分别代表第 0~32 号输出口的电平

返回值：错误代码

short nmc_write_outport_ex(WORD CardNo,WORD NoteID,WORD portno,DWORD outport_val,WORD* state)

功 能：设置指定 CAN-IO 扩展模块的输出端口的电平

参 数：CardNo 控制卡卡号
Node CAN 节点号，取值范围：1~8
PortNo 端口号，0 表示 0~31 号口，1 表示 32~63 号口
outport_val 输出值，bit0~bit32 的值分别代表第 0~32 号输出口的电平
state CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备 注：该函数实现功能同 nmc_write_outport，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户操作输出后，需同时判断 CAN 的连接状态，确认输出是否执行正常。

short nmc_read_outport(WORD CardNo, WORD Node, WORD PortNo，DWORD* outport_val)

功 能：读取指定 CAN-IO 扩展模块的输出端口的电平

参 数：CardNo 控制卡卡号
Node CAN 节点号，取值范围：1~8
PortNo 端口号
outport_val 输出端口的值，bit0~bit31 的值分别代表第 0~31 号输出口的电平

返回值：错误码

short nmc_read_outport_ex(WORD CardNo,WORD NoteID,WORD portno,DWORD *outport_val,WORD* state)

功　能：读取指定 CAN-IO 扩展模块的输出端口的电平

参　数：CardNo　　控制卡卡号

Node　　CAN 节点号，取值范围：1~8

PortNo　　端口号

outport_val　输出端口的值，bit0~bit31 的值分别代表第 0~31 号输出口的电平

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_read_outport，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取输出后，需同时判断 CAN 的连接状态，确认输出读取是否正常。

short nmc_read_inport(WORD CardNo, WORD Node, WORD PortNo，DWORD* inport_val)

功　能：读取指定 CAN-IO 扩展模块的输入端口的电平

参　数：CardNo　　控制卡卡号

Node　　CAN 节点号，取值范围：1~8

PortNo　　端口号

inport_val　　输入端口的值，bit0~bit31 的值分别代表第 0~31 号输入口的电平

返回值：错误码

short nmc_read_inport_ex(WORD CardNo,WORD NoteID,WORD portno,DWORD *inport_val,WORD* state)

功　能：读取指定 CAN-IO 扩展模块的输入端口的电平

参　数：CardNo　　控制卡卡号

Node　　CAN 节点号，取值范围：1~8

PortNo　　端口号

inport_val　　输入端口的值，bit0~bit31 的值分别代表第 0~31 号输入口的电平

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_read_inport，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取输入后，需同时判断 CAN 的连接状态，确认输入读取是否正常。

short nmc_set_da_output(WORD CardNo, WORD NoteID, WORD channel,double Value)

功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输出端口的值

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

channel　　模拟量输出通道号，范围 0~1

Value　　输出值，单位：mV/mA

返回值：错误代码

short nmc_set_da_output_ex(WORD CardNo,WORD NoteID,WORD channel,double Value,WORD* state)

功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输出端口的值

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

channel　　模拟量输出通道号，范围 0~1

Value　　输出值，单位：mV/mA

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_set_da_output，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户设置模拟量输出后，需同时判断 CAN 的连接状态，确认设置模拟量输出值是否正常。

short nmc_get_da_output(WORD CardNo, WORD NoteID, WORD channel,double* Value)

功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输出端口的值

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

channel　　模拟量输出通道号，范围 0~1

Value　　回读的输出值，单位：mV/mA

返回值：错误代码

short nmc_get_da_output_ex(WORD CardNo,WORD NoteID,WORD channel,double *Value,WORD* state)

功　能：读取指定 CAN-ADDA 扩展模块的某个输出端口的电平

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输出通道号，范围 0~1
Value　　回读的输出值，单位：mV/mA
state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_get_da_output，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户回读模拟量输出后，需同时判断 CAN 的连接状态，确认回读模拟量输出值是否正常。

short nmc_get_ad_input(WORD CardNo, WORD NoteID, WORD channel,double* Value)

功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输入端口的值

参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
Value　　输入端口的电平，单位：mV/mA

返回值：错误代码

short nmc_get_ad_input_ex(WORD CardNo,WORD NoteID,WORD channel,double *Value,WORD* state)

功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输入端口的值

参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
Value　　输入端口的电平，单位：mV/mA
state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_get_ad_input，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取模拟量输入后，需同时判断 CAN 的连接状态，确认读取模拟量输入值是否正常。

short nmc_set_ad_mode(WORD CardNo,WORD NoteID,WORD channel,WORD mode,DWORD buffer_nums)

功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输入端口的模式

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
mode　　输入模式，0：电压模式，1：电流模式
buffer_nums　　保留参数
返回值：错误代码

short nmc_set_ad_mode_ex(WORD CardNo,WORD NoteID,WORD channel,WORD mode,DWORD buffer_nums,WORD* state)
功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输入端口的模式
参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
mode　　输入模式，0：电压模式，1：电流模式
buffer_nums　　保留参数
state　　CAN 通讯状态，0：断开，1：连接，2：异常
返回值：错误代码
备　注：该函数实现功能同 nmc_set_ad_mode，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取模拟量输入端口模式后，需同时判断 CAN 的连接状态，确认读取模拟量输入端口模式是否正常。

short nmc_get_ad_mode(WORD CardNo,WORD NoteID,WORD channel,WORD* mode,DWORD buffer_nums)
功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输入端口的模式
参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
mode　　回读的输入模式，0：电压模式，1：电流模式
buffer_nums　　保留参数
返回值：错误代码

nmc_get_ad_mode_ex(WORD CardNo,WORD NoteID,WORD channel,WORD* mode,DWORD buffer_nums,WORD* state)
功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输入端口的模式
参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输入通道号，范围 0~3
mode　　回读的输入模式，0：电压模式，1：电流模式
buffer_nums　　保留参数
state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_get_ad_mode，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户回读模拟量输入端口模式后，需同时判断 CAN 的连接状态，确认回读模拟量输入端口模式是否正常。

short nmc_set_da_mode(WORD CardNo,WORD NoteID,WORD channel,WORD mode,DWORD buffer_nums)

功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输出端口的模式

参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输出通道号，范围 0~1
mode　　输出模式，0：电压模式，1：电流模式
buffer_nums　　保留参数

返回值：错误代码

short nmc_set_da_mode_ex(WORD CardNo,WORD NoteID,WORD channel,WORD mode,DWORD buffer_nums,WORD* state)

功　能：设置指定 CAN-ADDA 扩展模块的某个模拟量输出端口的模式

参　数：CardNo　　控制卡卡号
NoteID　　CAN 节点号，取值范围：1~8
channel　　模拟量输出通道号，范围 0~1
mode　　输出模式，0：电压模式，1：电流模式
buffer_nums　　保留参数
state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_set_da_mode，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户读取模拟量输出端口模式后，需同时判断 CAN 的连接状态，确认读取模拟量输出端口模式是否正常。

short nmc_get_da_mode(WORD CardNo,WORD NoteID,WORD channel,WORD* mode,DWORD buffer_nums)

功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输出端口的模式

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

channel　　模拟量输出通道号，范围 0~1

mode　　输出模式，0：电压模式，1：电流模式

buffer_nums　　保留参数

返回值：错误代码

short nmc_get_da_mode_ex(WORD CardNo,WORD NoteID,WORD channel,WORD* mode,DWORD buffer_nums,WORD* state)

功　能：读取指定 CAN-ADDA 扩展模块的某个模拟量输出端口的模式

参　数：CardNo　　控制卡卡号

NoteID　　CAN 节点号，取值范围：1~8

channel　　模拟量输出通道号，范围 0~1

mode　　输出模式，0：电压模式，1：电流模式

buffer_nums　　保留参数

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

备　注：该函数实现功能同 nmc_get_da_mode，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户回读模拟量输出端口模式后，需同时判断 CAN 的连接状态，确认回读模拟量输出端口模式是否正常。

short nmc_write_to_flash (WORD CardNo,WORD PortNum,WORD NodeNum)

功　能：保存模式设置到模块 flash

参　数：CardNo　　控制卡卡号

PortNum　　保留参数，设为 0

NodeNum　　节点号，1-8

返回值：错误代码

注　意：1）保存后模块会断开连接，需要重新连接才能进行正常控制。

2）设置的模式会断电保存，上电后电压或电流模式为最后一次断电前设置的模式。

short nmc_write_to_flash_ex(WORD CardNo,WORD PortNum,WORD NodeNum,WORD* state);

功　能：保存模式设置到模块 flash

参　数：CardNo　　控制卡卡号

PortNum　　保留参数，设为 0

NodeNum　　节点号，1-8

state　　CAN 通讯状态，0：断开，1：连接，2：异常

返回值：错误代码

注　意：1）保存后模块会断开连接，需要重新连接才能进行正常控制。

2）设置的模式会断电保存，上电后电压或电流模式为最后一次断电前设置的模式。

备　注：该函数实现功能同 nmc_write_to_flash，同时集成 nmc_get_connect_state 函数功能，可读取 CAN 连接状态，用户保存模式设置到模块后，需同时判断 CAN 的连接状态，确认执行正常。

9.23　密码管理函数

short dmc_enter_password_ex(WORD CardNo, const char* spass);

功　能：密码登录

参　数：CardNo：　卡号

spass：　旧密码，密码长度不大于 255 个字符

返回值：错误代码

注　意：

1）调用写入密码（dmc_write_sn）函数前，需要调用此函数进行验证登录，验证成功后才可调用写密码函数修改密码。

2）密码登录连续失败 5 次后，无法进行登录；若需再次登录，需将电脑关机，断电重启。

3）用户可以在系统软件开启时加入密码登录函数核对密码，以此对系统软件进行加密。

short dmc_write_sn(WORD CardNo, const char* new_sn)

功　能：写入密码

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：校验密码

参　数：CardNo　　控制卡卡号

check_sn　　旧密码，密码长度不大于 255 个字符

返回值：校验状态，0：失败，1：成功，2 超出校验次数

注　意：密码校验连续失败 5 次后，无法进行校验；若需再次校验，需将电脑关机，断电重启

9.24　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

第 10 章　基于脉冲当量的高级运动函数说明

10.1　脉冲当量设置

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

equiv　　脉冲当量，单位：pulse/unit

返回值：错误代码

注　意：1）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、连续插补运动）

2）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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

equiv　　返回脉冲当量

返回值：错误代码

10.2　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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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：Gear
8：Cam
9：Line
10：Continue
11：Stop_Mode（停止模式）
12：Limit_Mode（限位模式）
14：E_Gantry_Mode（龙门模式）
17：E_Gantry_Mode_Out（退出龙门模式）
18：U_Move（门型模式）
19：FOLLOW_MOVE（插补速度跟随模式）
20：SINE_ASCILLATE_MODE（正弦振荡曲线模式）
21：FOLLOW_LINER_VEL_MODE（插补速度跟随模式）
22：PMOVE_CRD_PAUSE（插补暂停定长运动模式）

返回值：错误代码

说　明：1）该函数适用于所有运动（包括以脉冲为单位的运动及基于脉冲当量的运动，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
3）当执行以脉冲为单位的两轴圆弧插补运动时，使用该函数读取运动模式为 6

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：0~3

返回值：运动状态，0：运动中，1：暂停中， 3：未启动，4：空闲，5：异常停止

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。如果缓冲区内运动指令都执行完成后，用户没有调用关闭缓冲区的话，回读的插补系状态仍是运动中。
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)“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，此时被加到坐

标系中的轴不能再被调用。

5)“空闲”指坐标系未打开的状态，调用 stoplist 后进入此状态。

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

pos　　返回当前位置值，单位：unit

返回值：错误代码

备　注：轴号设置为 255 表示读取所有轴的当前指令位置

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~7，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

pos　　返回当前编码器计数值，单位：unit

返回值：错误代码

备　注：轴号设置为 255 表示读取所有轴的当前反馈位置

short dmc_get_target_position_unit(WORD CardNo, WORD axis,double* pos)

功　能：读取正在运动轴的目标位置（绝对坐标）

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

pos　　返回目标位置，单位：unit。

返回值：错误代码

short dmc_read_current_speed_unit(WORD CardNo, WORD axis, double *current_speed)

功　能：读取指定轴的当前速度

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

current_speed　返回速度值，单位：unit/s

返回值：错误代码

注　意：1）当执行基于脉冲当量的插补及连续插补运动时，使用该函数读取的为各轴的分速度值

2）**轴号设置为 255 表示读取所有轴的当前速度**

short dmc_read_vector_speed_unit(WORD CardNo,WORD Crd,double *current_speed)

功　能：读取插补运动的合速度

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

current_speed　返回插补合速度值，单位：unit/s

返回值：错误代码

short dmc_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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

StopReason 停止原因：

0：正常停止
1：ALM 立即停止
2：ALM 减速停止
3：LTC 外部触发立即停止
4：EMG 立即停止
5：正硬限位立即停止
6：负硬限位立即停止
7：正硬限位减速停止
8：负硬限位减速停止
9：正软限位立即停止
10：负软限位立即停止
11：正软限位减速停止
12：负软限位减速停止
13：命令立即停止
14：命令减速停止
15：其它轴引起的立即停止
16：其它轴引起的减速停止
17：模拟量超限，触发急停
18：保留
19：DSTP 信号引起的减速停止
20：前瞻预处理和速度规划来不及，底层自动调用 stoplist
21：原点不在两个限位之间停止
22：回零方向和当前有效限位端相反（正限位有效时，回零方向为负方向；负限位有效时，回零方向为正方向）
23：正负限位同时有效
24：没有找到 EZ 信号
25：回零位置溢出停止
26：动加速度过大，异常停止
27：双原点停止
28：区域限位立即停止
29：区域限位减速停止
30：龙门超差保护减速停止
31：龙门超差保护立即停止

32：插补规划错误停止
33：单轴跟踪误差保护立即停止
34：单轴跟踪误差保护减速停止
35：插补缓冲区错误停止
36：龙门主从轴超差保护立即停止
37：龙门主从轴超差保护减速停止
38：轴输出脉冲频率超过 4M
39：IO 触发立即停止
40：IO 触发减速停止
49：看门狗超时
50：两轴碰撞检测立即停止
51：两轴碰撞检测减速停止
52：龙门从轴 alm 引起主轴立即停止
53：从轴开始启动时主轴绝对位置错误（不在主轴运动方向上）
54：同步时主轴绝对位置错误（不在主轴运动方向上）
55：从轴开始运动位置大于同步时主轴绝对位置错误
56：同步时从轴绝对位置（不在从轴运动方向上）
57：不支持主轴运动模式（仅支持跟随主轴做定长和定速运动模式）
58：从轴追赶目标速度小于同步时从轴的速度,建议调大追赶目标速度
59：从轴同步位置过小，不足以从轴从当前速度直接加速或减速至同步速度,建议调大从轴同步位置或调大加减速度
60：初始化时主轴运动方向错误(运动速度为零)
61：从轴开始运动时主轴未达到匀速状态,建议调大从轴启动时主轴的位置
62：GearInPos 实际加速度大于设置的最大加速度
63：GearInPos 实际减速度大于设置的最大减速度
64：从轴追赶实际可达速度大于追赶设置的最大目标速度,且以最大追赶速度运行无法实现规定时间内达到从轴的位移，建议调大追赶目标速度或减小从轴同步位置
65：从轴同步速度在只有加速或减速过程也不可达速度,建议调大从轴从轴同步位置或调小齿轮比或调大从轴开始运动的主轴位置
66：二分法求取合适的可达速度，迭代次数超过限制次数让无法找到合适的速度值
67：主轴 pmove 目标位置小于主轴同步位置

68：主轴 pmove 目标位置小于从轴开始运动时的主轴位置
69：从轴追赶过程加速周期或减速周期为 0，有可能主轴同步位置和从轴开始运动时的主轴位置相隔太近
70：主轴目标速度为零
71：从轴实际可达速度大于追赶速度时，重算以追赶速度为目标速度的匀速周期数小于等于零
72：GearInPos 主轴位移与规划结果不一致
73：GearInPos 主轴 vmove 减速阶段不支持
74：从轴起始反向运动方向模式错误(1~4)
75：从轴起始反向运动加速度为零
76：连续轨迹等待外部 io 输入信号有效超时，停止插补系运动
201：正负限位之间全程没找到原点信号
202：回零方向不匹配
203：正负限位同时有效
204：正负限位之间全程没有 EZ 信号
205：位置溢出
206：双原点错误
207：外部信号触发回零停止
208：驱动器回零被中断停止

返回值：错误代码

short dmc_clear_stop_reason(WORD CardNo, WORD axis)
功　能：清除指定轴的停止原因
参　数：CardNo　　卡号
　　axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
返回值：错误代码

short dmc_calculate_arclength_3point(double *start_pos, double *mid_pos, double *target_pos, double circle, double *ArcLength)
功　能：计算三点圆弧的弧长（只能用于计算空间三轴圆弧弧长）
参　数：start_pos　　起始位置数组，此数组为三维数组，单位：unit
　　mid_pos　　经过点位置数组，此数组为三维数组，单位：unit
　　target_pos　　目标位置数组，此数组为三维数组，单位：unit

circle　　圈数

ArcLength　　返回弧长值

返回值：错误代码

说　明：该指令只能计算圆弧的弧长，不能计算渐近渐开线的弧长

10.3　点位运动

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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

注　意：由于运动控制卡的最大脉冲输出频率为 4MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 4MHz

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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　意：该函数不适用于连续插补

short dmc_set_profile_unit_acc(WORD CardNo, WORD axis, double Min_Vel, double Max_Vel, double Acc, double Dec, double Stop_Vel)

功　能：设置单轴运动速度曲线，加速度模式

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Min_Vel　　起始速度，单位：unit/s

Max_Vel　　最大速度，单位：unit/s

Acc　　加速度，单位：unit /（s²）

Dec　　减速度，单位：unit /（s²）

Stop_Vel　　停止速度，单位：unit/s

返回值：错误代码

注　意：由于运动控制卡的最大脉冲输出频率为 4MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 4MHz。

short dmc_get_profile_unit_acc(WORD CardNo, WORD axis, double* Min_Vel, double* Max_Vel, double* Acc, double* Dec, double* Stop_Vel)

功　能：读取单轴运动速度曲线，加速度模式

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

Min_Vel　　返回起始速度设置

Max_Vel　　返回最大速度设置

Acc　　返回加速度设置

Dec　　返回减速度设置

Stop_Vel　　返回停止速度设置

返回值：错误代码

注　意：该函数不适用于连续插补

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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

注意：目标位置和脉冲当量的乘积范围为-134217728~+134217728

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

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 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

10.4　插补速度设置

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

功　能：设置插补运动速度曲线

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

Min_Vel　　最小速度，单位：unit/s

Max_Vel　　最大速度，单位：unit/s

Tacc　　加速时间，单位：s

Tdec　　减速时间，单位：s

Stop_Vel　　停止速度，单位：unit/s

返回值：错误代码

注　意：1）DMC5X10 系列卡支持四个坐标系（参数 Crd）。四个坐标系的速度可独立设置，执行插补时四个坐标系可独立进行插补运动（即可同时进行四组插补运动）

2）由于运动控制卡的最大脉冲输出频率为 4MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 4MHz

3）该指令既可以设置单段插补运动速度，也可以设置连续插补运动速度

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：0~3

Min_Vel 返回最小速度设置
Max_Vel 返回最大速度设置
Tacc 返回加速时间设置
Tdec 返回减速时间设置
Stop_Vel 返回停止速度
返回值：错误代码

short dmc_set_vector_s_profile(WORD CardNo, WORD Crd, WORD s_mode, double s_para)
功　能：设置插补运动速度曲线的平滑时间
参　数：CardNo 卡号
Crd 坐标系号，取值范围：0~3
s_mode 保留参数，固定值为 0
s_para 平滑时间，单位：s，范围：0~1
返回值：错误代码

short dmc_get_vector_s_profile(WORD CardNo, WORD Crd, WORD s_mode, double *s_para)
功　能：读取设置的插补运动速度曲线平滑时间
参　数：CardNo 卡号
Crd 坐标系号，取值范围：0~3
s_mode 保留参数，固定值为 0
s_para 返回平滑时间设置
返回值：错误代码

10.5　插补运动

short dmc_line_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double* Target_Pos, WORD posi_mode)
功　能：直线插补运动
参　数：CardNo 卡号
Crd 坐标系号，取值范围：0~3
AxisNum 轴数，取值范围：5410A：2~4，DMC5C10/5810/5610：2~6
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：0~3

AxisNum　　轴数，取值范围：

螺旋线插补模式时：5410A：2~4，DMC5C10/5810/5610：2~6

同心圆插补模式时：**保留参数**，固定值为 2

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：

负数：表示此时执行的为同心圆插补

该值的绝对值加 1 表示同心圆的圈数。如，-1 即表示 2 圈同心圆插补，-2 表示 3 圈同心圆插补…

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或同心圆插补

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）

5) DMC5C10 后四轴不支持圆弧插补运动

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：0~3

AxisNum　　轴数，取值范围：5410A：2~4，DMC5C10/5810/5610：2~6

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Arc_Radius　　圆弧半径值（负值为优弧，正值为劣弧），单位：unit

Arc_Dir　　圆弧方向，0：顺时针，1：逆时针

Circle　　圈数，取值范围：大于等于 0

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

注　意：1）当轴数为 2 时，轴列表前两轴进行平面圆弧插补

2）当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

3）当轴数大于 3 时，轴列表前三轴进行圆柱螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等

short dmc_arc_move_3points_unit(WORD CardNo,WORD Crd,WORD AxisNum,WORD* AxisList, double *Target_Pos, double *Mid_Pos, long Circle, WORD posi_mode)

功　能：基于三点圆弧扩展的圆柱螺旋线插补运动（可作两轴及三轴圆弧插补）

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

AxisNum　　轴数，取值范围：5410A：2~4，DMC5C10/5810/5610：2~6

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Mid_Pos　　中间位置数组，单位：unit

Circle　　圈数：

负数：表示此时执行的为空间圆弧插补

该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧…

自然数：表示此时执行的为圆柱螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

注　意：1）当轴数为 2 时，轴列表前两轴进行平面圆弧插补

2）当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

3）当轴数大于 3、运动轨迹为螺旋插补时，列表前三轴进行螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等

short dmc_rectangle_move_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD *AxisList, double *Target_Pos, double *Mark_Pos, long Count, WORD rect_mode, WORD posi_mode)

功　能：矩形插补运动

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

AxisNum　　轴数，**保留参数**，固定值为 2

AxisList　　轴号列表

Target_Pos　　对角位置数组，单位：unit

Mark_Pos　　矩形方向标记位置数组，单位：unit

Count　　行数/圈数

rect_mode　　矩形插补模式，0：逐行，1：渐开线

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

注　意：该指令只支持 2 轴矩形插补运动。矩形插补运动支持前瞻模式和非前瞻模式(dmc_conti_set_lookahead_mode)，前瞻模式矩形拐角会平滑处理变成圆弧，无法到达矩形端点；非前瞻模式则精确到达矩形端点。

short dmc_axis_follow_line_enable(WORD CardNo,WORD Crd,WORD enable_flag)

功　能：直线插补参与插补轴数设置，默认情况下为所有轴都参与直线插补
参　数：CardNo　　卡号
　　Crd　　坐标系号，取值范围：0~3
　　enable_flag　使能轴数值，
　　0：最多只有 3 个轴参与直线插补，其它轴跟随运动
　　1：坐标系所有轴都参与直线插补
返回值：错误代码
注　意：该指令只针对连续插补中直线插补运动

10.6　连续插补运动

short dmc_conti_open_list(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList)
功　能：打开连续插补缓冲区
参　数：CardNo　　卡号
　　Crd　　坐标系号，取值范围：0~3
　　AxisNum　　轴数，取值范围：DMC5410A：2~4，DMC5C10/5810/5610：2~6
　　AxisList　　轴号列表：
　　AxisList[0]：X 轴
　　AxisList[1]：Y 轴
　　AxisList[2]：Z 轴
　　AxisList[3]：U 轴
　　AxisList[4]：V 轴
　　AxisList[5]：W 轴
返回值：错误代码
说　明：1）连续缓冲区最多可缓存 5000 条指令
　　2）当打开连续插补缓冲区后，则进入连续插补模式；此时，除非当执行完缓冲区中的指令或是调用停止连续插补指令 dmc_conti_stop_list 后，参与连续插补的运动轴才能退出连续插补模式

short dmc_conti_start_list(WORD CardNo, WORD Crd)
功　能：开始连续插补
参　数：CardNo　　卡号
　　Crd　　坐标系号，取值范围：0~3
返回值：错误代码

short dmc_conti_close_list(WORD CardNo, WORD Crd)

功　能：关闭连续插补缓冲区

参　数：CardNo　　卡号

　　Crd　　坐标系号，取值范围：0~3

返回值：错误代码

short dmc_conti_pause_list(WORD CardNo,WORD Crd)

功　能：暂停连续插补

参　数：CardNo　　卡号

　　Crd　　坐标系号，取值范围：0~3

返回值：错误代码

说　明：当暂停连续插补后，连续插补运动将减速停止，当再次调用 dmc_conti_start_list 指令时运动控制卡将继续运行之前未完成的连续插补轨迹。

short dmc_conti_stop_list(WORD CardNo, WORD Crd, WORD stop_mode)

功　能：停止插补运动

参　数：CardNo　　卡号

　　Crd　　坐标系号，取值范围：0~3

　　stop_mode　　停止模式，0：减速停止，1：立即停止

返回值：错误代码

说　明：1）该函数适用于所有插补运动

2）当正在执行插补运动时，通过此指令可以中止插补运动，并使参与插补的运动轴退出插补模式

short dmc_conti_set_override(WORD CardNo,WORD Crd,double Percent)

功　能：设置连续插补段速度比例

参　数：CardNo　　卡号

　　Crd　　坐标系号，取值范围：0~3

　　Percent　　速度比例，取值范围：0~1.0

返回值：错误代码

说　明：1）当计算机执行到此指令时，该指令将存入缓冲区，从该指令的下一条运动函数开始运动时起作用。

2）当调整速度比例后，后续连续插补运动中将一直保持调整后的速度比例，直到再

次调用速度比例设置函数。

short dmc_conti_change_speed_ratio (WORD CardNo, WORD Crd, double Percent)

功　能：动态调整连续插补速度比例

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

Percent　　速度比例，取值范围：0~1.0

返回值：错误代码

说　明：1) 当计算机执行到此指令时，运动控制卡将立即调整连续插补速度比例，此指令一般在调试中使用；

2) 该函数必须在打开缓冲区（dmc_conti_open_list）后才能调用；

short dmc_conti_delay(WORD CardNo, WORD Crd, double delay_time, long mark)

功　能：连续插补中暂停延时指令

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

delay_time　　延时时间，单位：秒

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）延时时间为运动停止时的等待时间

2）当延时时间设置为 0 时，延时时间将无限长

short dmc_conti_line_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double* Target_Pos, WORD posi_mode, long mark)

功　能：连续插补中直线插补指令

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

AxisNum　　轴数，DMC5410A：2~4，DMC5C10/5810/5610：2~6

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

short dmc_conti_arc_move_center_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD*

AxisList, double *Target_Pos, double *Cen_Pos, WORD Arc_Dir, long Circle, WORD posi_mode, long mark)

功　能：连续插补中基于圆心圆弧扩展的螺旋线插补指令（可作两轴圆弧插补）

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

AxisNum　　轴数，取值范围：DMC5410A：2~4，DMC5C10/5810/5610：2~6

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：

负数：表示此时执行的为同心圆插补

该值的绝对值加 1 表示同心圆的圈数。如，-1 即表示 2 圈同心圆插补，-2 即表示 3 圈同心圆插补…

非负数：表示此时执行的为螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 1 圈螺旋线插补，1 即表示 2 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 dmc_conti_open_list 的说明

2）当轴数为 2 时，轴列表前两轴进行平面螺旋或同心圆插补

3）当轴数为 3、运动轨迹为螺旋插补时，轴列表前两轴平面为基面，进行平面螺旋插补；同时，轴列表第三轴运动指定高度，该轴终点位置与该轴起点位置的差值为螺旋线段相对于基面的高度

4）当轴数大于 3、运动轨迹为螺旋插补时，主动轴进行螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 dmc_conti_open_list 的说明

5）当运动轨迹为螺旋插补时：

轴列表前两轴组成的基面上，当起始点到圆心的距离小于终点到圆心的距离，为绽放螺旋线

轴列表前两轴组成的基面上，当起始点到圆心的距离大于终点到圆心的距离，为收敛螺旋线

轴列表前两轴组成的基面上，当起始点到圆心的距离等于终点到圆心的距离，为圆弧插补（插补轴数为 3 时则为圆柱螺旋线）

short dmc_conti_arc_move_radius_unit(WORD CardNo, WORD crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double Arc_Radius, WORD Arc_Dir, long Circle, WORD posi_mode, long mark)

功　能：连续插补中基于半径圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

AxisNum　　轴数，取值范围：DMC5410A：2~4，DMC5C10/5810/5610：2~6

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Arc_Radius　　圆弧半径值（负值为优弧，正值为劣弧），单位：unit

Arc_Dir　　圆弧方向，0：顺时针，1：逆时针

Circle　　圈数，取值范围：大于等于 0

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 dmc_conti_open_list 的说明

2）当轴数为 2 时，轴列表前两轴进行平面圆弧插补

3）当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度

4）当轴数大于 3 时，主动轴进行圆柱螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 dmc_conti_open_list 的说明

short dmc_conti_arc_move_3points_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD* AxisList, double *Target_Pos, double *Mid_Pos, long Circle, WORD posi_mode, long mark)

功　能：连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴及三轴圆弧插补）

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3
AxisNum　　轴数，取值范围：DMC5410A：2~4，DMC5C10/5810/5610：2~6
AxisList　　轴号列表
Target_Pos　目标位置数组，单位：unit
Mid_Pos　　中间位置数组，单位：unit
Circle　　圈数
负数：表示此时执行的为空间圆弧插补
该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧…
自然数：表示此时执行的为圆柱螺旋线插补
该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…
posi_mode　运动模式，0：相对坐标模式，1：绝对坐标模式
mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 dmc_conti_open_list 的说明
2）当轴数为 2 时，轴列表前两轴进行平面圆弧插补
3）当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度
4）当轴数大于 3 时，主动轴进行圆柱螺旋插补或空间圆弧插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 dmc_conti_open_list 的说明

short dmc_conti_rectangle_move_unit(WORD CardNo, WORD Crd, WORD AxisNum, WORD *AxisList, double *Target_Pos, double *Mark_Pos, long Count, WORD rect_mode, WORD posi_mode, long mark)

功　能：连续插补中矩形插补指令

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
AxisNum　　轴数，**保留参数**，固定值为 2
AxisList　　轴号列表
Target_Pos　　对角位置数组，单位：unit

Mark_Pos　　矩形方向标记位置数组，单位：unit
Count　　行数/圈数
rect_mode　　矩形插补模式，0：逐行，1：渐开线
posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式
mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：该指令只支持 2 轴矩形插补运动

short dmc_conti_pmove_unit(WORD CardNo, WORD Crd, WORD axis, double dist, WORD posi_mode, WORD mode, long imark)

功　能：连续插补中控制指定轴运动

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
axis　　指定轴号
dist　　目标位置，单位：unit
posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式
mode　　模式：
0：暂停启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动；当本段定长运动结束后，再执行下一段插补运动）
1：直接启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动，并且同时执行下一段插补运动）
mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）该指令可以实现在连续插补运动中，控制指定轴做定长运动
2）该轴不能为参与连续插补的运动轴，即不能为插补系轴列表中的轴
3）在使用该指令控制轴运动前，必须先使用函数 dmc_set_profile_unit 设置该轴的运行速度

short dmc_conti_pmove_unit_pausemode(WORD CardNo, WORD axis,double TargetPos, double Min_Vel,double Max_Vel,double stop_Vel,double acc,double dec,double smooth_time,WORD posi_mode)

功　能：连续插补暂停后，执行单轴运动

参　数：CardNo　　卡号
Axis　　轴号

TargetPos 目标位置
Min_Vel 单轴运动起始速度
Max_Vel 单轴运动运行速度
stop_Vel 单轴运动停止速度
acc 单轴运动加速度
dec 单轴运动减速度
smooth_time 单轴运动平滑时间
posi_mode 运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

short dmc_conti_return_pausemode(WORD CardNo, WORD Crd, WORD axis)
功　能：连续插补暂停，执行单轴运动后，回到暂停位置
参　数：CardNo 卡号
Crd 插补系号，0-3
Axis 轴号
返回值：错误代码

10.7　连续插补缓冲区检测

long dmc_conti_remain_space(WORD CardNo, WORD Crd)
功　能：查询连续插补缓冲区剩余插补空间
参　数：CardNo 卡号
Crd 坐标系号，取值范围：0~3
返回值：连续插补缓冲区剩余大小

long dmc_conti_read_current_mark (WORD CardNo, WORD Crd)
功　能：读取连续插补缓冲区当前插补段号
参　数：CardNo 卡号
Crd 坐标系号，取值范围：0~3
返回值：当前连续插补段号

10.8　连续插补小线段前瞻功能

short dmc_conti_set_lookahead_mode(WORD CardNo, WORD Crd, WORD enable, long

LookaheadSegments, double PathError, double LookaheadAcc)

功　能：设置连续插补前瞻参数

参　数：CardNo 卡号

Crd	坐标系号，取值范围：0~3
enable	前瞻使能状态，0：禁止，1：使能
LookaheadSegments	前瞻段数，**取值范围：25~5000，小于 25 时，默认为 25**
PathError	允许误差范围，取值范围：非负数,单位：unit
LookaheadAcc	前瞻加速度，取值范围：非负数，单位：unit/s^2

返回值：错误代码

注　意：

1. 小线段前瞻支持圆弧过渡和非圆弧过渡两种方式，设置轨迹误差范围为零时没有圆弧过渡，不为零时默认有圆弧过渡，且普通连续插补没有圆弧过渡功能。前瞻加速度 LookaheadAcc 控制拐角过渡时的速度大小。一般地，前瞻加速度越大，拐角速度越大，加工效率越高。前瞻加速度一般指机械系统允许的加速度，前瞻加速度过大，容易引起机械震动。
2. 当前瞻段数设置小于 25 段的时候会强制修改为 25 段。

short dmc_conti_get_lookahead_mode(WORD CardNo, WORD Crd, WORD* enable, long* LookaheadSegments, double* PathError, double* LookaheadAcc)

功　能：读取连续插补前瞻参数

参　数：

CardNo	卡号
Crd	坐标系号，取值范围：0~3
enable	前瞻使能状态，0：禁止，1：使能
LookaheadSegments	前瞻段数，取值范围：非负数
PathError	允许误差范围，取值范围：非负数,单位：unit
LookaheadAcc	前瞻加速度，取值范围：非负数,单位：unit/s^2

返回值：错误代码

10.9　连续插补 IO 控制

short dmc_conti_set_pause_output(WORD CardNo, WORD crd, WORD action, long mask, long state)

功　能：设置连续插补暂停及异常停止时 IO 输出状态

参　数：

CardNo	卡号
Crd	坐标系号，取值范围：0~3

action　　激活模式：
0：保持原状
1：暂停连续插补时输出设定的 IO 状态，恢复运行时不恢复暂停前的 IO 状态
2：暂停连续插补时输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态
3：当暂停、停止连续插补，或遇到其他异常停止（如碰到 EMG 信号）时，输出设定的 IO 状态

mask　　选择输出端口标志：bit0~bit31 代表 Out0~Out31，位值为 1 时输出，位值为 0 不输出

state　　输出电平状态：bit0~bit31 代表 Out0~Out31，位值为 1 时输出高电平，位值为 0 时输出低电平

返回值：错误代码

激活模式 3 的说明：1）暂停连续插补时，运动控制卡输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态
2）停止连续插补、或遇到其他异常停止时，运动控制卡输出设定的 IO 状态，但是再次启动连续插补时不会恢复之前的 IO 状态

short dmc_conti_get_pause_output(WORD CardNo, WORD crd, WORD* action, long* mask, long* state)

功　能：读取连续插补暂停及异常停止时 IO 输出状态设置

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
action　　返回激活状态设置
mask　　返回输出选择标志设置
state　　返回输出电平状态设置

返回值：错误代码

short dmc_conti_wait_input(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double TimeOut, long mark)

功　能：连续插补等待 IO 输入.当运动控制卡执行到此指令时，只有在接受到输入 IO 信号或超出超时时间后，才会执行后续运动

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3

bitno　　输入口号，取值范围：0~31
on_off　　电平状态，0：低电平，1：高电平
TimeOut　　超时时间，单位：s
mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

注　意：1）当超时时间设为 0 时，运动控制卡将一直等待 IO 输入信号，超时时间为无限长
2）超时时间为运动停止时的等待时间

short dmc_conti_delay_outbit_to_start(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double delay_value, WORD delay_mode, double ReverseTime)

功　能：连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
bitno　　输出口号，取值范围：0~31
on_off　　电平状态，0：低电平，1：高电平
delay_value　滞后值，单位：s（滞后时间模式）或 unit（滞后距离模式）
delay_mode　滞后模式，0：滞后时间，1：滞后距离
ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

注　意：1）设置的 IO 操作，将在该指令的下一条轨迹中起作用
2）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变
3）当滞后模式选择为滞后距离时，位置源为指令位置计数器

short dmc_conti_delay_outbit_to_stop(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double delay_time, double ReverseTime)

功　能：连续插补中相对于轨迹段终点 IO 滞后输出

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
bitno　　输出口号，取值范围：0~31
on_off　　电平状态，0：低电平，1：高电平
delay_time　滞后时间，单位：s
ReverseTime　保留参数，固定值为 0

返回值：错误代码

注　意：1）设置的 IO 操作，将在该指令的下一条轨迹结束后起作用

2）如果使用了 dmc_conti_clear_io_action 函数清除段内未执行完的 IO 动作时，那么该指令将不会被执行

short dmc_conti_ahead_outbit_to_stop(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double ahead_value, WORD ahead_mode, double ReverseTime)

功　能：连续插补中相对于轨迹段终点 IO 提前输出（段内执行）

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

bitno　　输出口号，取值范围：0~31

on_off　　电平状态，0：低电平，1：高电平

ahead_value　提前值，单位：s（提前时间模式）或 unit（提前距离模式）

ahead_mode　提前模式，0：提前时间，1：提前距离

ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

注　意：1）设置的 IO 操作，将在该指令的下一条轨迹中起作用

2）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变

3）当滞后模式选择为滞后距离时，位置源为指令位置计数器

short dmc_conti_accurate_outbit_unit(WORD CardNo, WORD Crd, WORD cmp_no, WORD on_off, WORD map_axis, double rel-dist, WORD pos_source, double ReverseTime)

功　能：连续插补中精确位置 CMP 输出控制

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

cmp_no　　CMP 输出端口号，取值范围：0~3

on_off　　电平状态，0：低电平，1：高电平

map_axis　　坐标系内关联轴号：

0：X 轴

1：Y 轴

2：Z 轴

3：U 轴

4：V 轴

5：W 轴

rel-dist　　相对于轨迹段起点的距离在关联轴上的分量距离

pos_source　位置源，0：指令位置计数器，1：编码器计数器

ReverseTime　电平输出后的延时翻转时间，单位：us，范围：1us~20s

返回值：错误代码

注　意：1）设置的 IO 操作，将在该指令的下一条轨迹中起作用

2）ReverseTime 参数不能设置为 0，当该参数设置为 0 时，相应 CMP 口将不会被操作

3）如果 ReverseTime 参数设置过大，那么当该段轨迹执行完毕时，CMP 端口电平仍会自动翻转，即使此时未达到设置的电平延时翻转时间

4）此功能为一维高速位置比较（队列模式）的扩展功能。当启用精确位置 CMP 输出控制时，会占用高速比较器资源，所以一维高速位置比较功能与精确位置 CMP 输出控制功能不能在同一时间内使用，否则可能会出现错误动作。

5）执行精确位置 CMP 输出时，每个位置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。如果期望在连续插补中使用该功能，可以在打开连续插补缓冲区时就调用函数 dmc_hcmp_clear_points 清除相应比较器的比较点

6）关于 XYZUVW 轴对应定义详见打开函数 dmc_conti_open_list 的说明

short dmc_conti_write_outbit(WORD CardNo, WORD Crd, WORD bitno, WORD on_off, double ReverseTime)

功　能：连续插补中缓冲区立即 IO 输出

参　数：CardNo　卡号

Crd　坐标系号，取值范围：0~3

bitno　输出口号，取值范围：0~31

on_off　电平状态，0：低电平，1：高电平

ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

说　明：1）当计算机执行到此指令时，将该指令存入缓冲区，当缓冲区中的上一段运动指令执行完毕时，该指令将执行

2）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变

short dmc_conti_clear_io_action(WORD CardNo, WORD Crd, DWORD IoMask)

功　能：清除段内未执行完的 IO 动作

参　数：CardNo　卡号

Crd　坐标系号，取值范围：0~3

IoMask　清除标志：bit0~bit31 分别表示 Out0~Out31 输出口；位值：1：清除对

应输出口段内未执行完的动作，比如翻转时间到达后 IO 翻转动作；0：不操作

返回值：错误代码

说　　明： 该函数对 dmc_conti_delay_outbit_to_start 、 dmc_conti_ahead_outbit_to_stop 、 dmc_conti_delay_outbit_to_stop 指令起作用

10.10　设置螺旋线插补运动模式

short dmc_conti_set_involute_mode(WORD CardNo, WORD Crd, WORD mode)

功　能：设置螺旋线插补运动模式

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

mode　　螺旋线运动模式，0：不封闭，1：封闭

返回值：错误代码

注　意： 1）该函数只对基于圆心圆弧扩展的螺旋线插补运动函数 dmc_arc_move_center_unit、dmc_conti_arc_move_center_unit 起作用

2）当绽放型螺旋线插补运动设置为封闭时，先执行绽放型螺旋线插补运动，当其运动到终点后，仍然继续运行一圈（半径大小为目标位置与圆心位置的差值），将该螺旋线封闭

3）当收敛型螺旋线插补运动设置为封闭时，先运行一个封闭的圆（半径大小为当前位置与圆心位置的差值），回到起始点后，再进行收敛型螺旋线插补运动

short dmc_conti_get_involute_mode(WORD CardNo, WORD Crd, WORD *mode)

功　能：读取螺旋线插补运动模式设置

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

mode　　返回设置的螺旋线运动模式

返回值：错误代码

10.11　反向间隙设置

short dmc_set_backlash_unit(WORD CardNo, WORD axis, double backlash)

功　能：设置指定轴的反向间隙值

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

backlash　　反向间隙值，单位：unit

返回值：错误代码

short dmc_get_backlash_unit(WORD CardNo, WORD axis, double *backlash)

功　能：读取指定轴的反向间隙值设置

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

backlash　　返回反向间隙设置值

返回值：错误代码

10.12　PWM 功能

short dmc_set_pwm_enable(WORD CardNo, WORD enable)

功　能：设置 PWM 使能状态

参　数：CardNo　　卡号

enable　　PWM 使能状态，0：禁止，1：使能

返回值：错误代码

注　意：

1）DMC5C10 中，当使能 PWM 功能后，11 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，11 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1；

2）DMC5810 中，当使能 PWM 功能后，7 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，7 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1；

3）DMC5610 中，当使能 PWM 功能后，5 号轴的脉冲端口（PUL+与 PUL-）作为 PWM 输出通道 0，5 号轴的方向端口（DIR+与 DIR-）作为 PWM 输出通道 1；

4）当使用通道 0 时，硬件接线可接 PUL+与 GND（PUL-与 GND 则输出相反信号）；当使用通道 1 时，硬件接线可接 DIR+与 GND（DIR-与 GND 则输出相反信号）；

5)　DMC5410A 中 CN23 有两路 PWM 专用输出口，详见附录 5；

6）当使能 PWM 功能后，运动控制卡的相应轴端口将不能输出电机控制信号；

short dmc_get_pwm_enable(WORD CardNo, WORD *enable)

功　能：读取 PWM 使能状态设置

参　数：CardNo　　卡号
　　enable　　返回 PWM 使能状态
返回值：错误代码

dmc_set_pwm_enable_extern(WORD CardNo,WORD channel, WORD enable)
功　能：按通道号设置 PWM 使能状态
参　数：CardNo　　卡号
　　channel　　PWM 通道
　　enable　　PWM 使能状态，0：禁止，1：使能
返回值：错误代码

dmc_get_pwm_enable_extern(WORD CardNo,WORD channel, WORD* enable);
功　能：按通道号读取 PWM 使能状态
参　数：CardNo　　卡号
　　channel　　PWM 通道
　　enable　　PWM 使能状态，0：禁止，1：使能
返回值：错误代码

short dmc_set_pwm_output(WORD CardNo, WORD pwm_no, double fDuty, double fFre)
功　能：设置 PWM 立即输出
参　数：CardNo　　卡号
　　pwm_no　　PWM 通道，取值范围：0~1
　　fDuty　　占空比，取值范围：0~1
　　fFre　　频率，取值范围：0~500KHz
返回值：错误代码

short dmc_get_pwm_output(WORD CardNo ,WORD pwm_no, double* fDuty, double* fFre)
功　能：读取 PWM 立即输出设置
参　数：CardNo　　卡号
　　pwm_no　　PWM 通道，取值范围：0~1
　　fDuty　　返回占空比设置值
　　fFre　　返回频率设置值
返回值：错误代码

10.13 连续插补 PWM 输出

short dmc_set_pwm_onoff_duty(WORD CardNo, WORD PwmNo,double fOnDuty, double fOffDuty)

功　能：设置 PWM 开关状态对应的占空比

参　数：CardNo　　卡号

PwmNo　　PWM 通道，取值范围：0~1

fOnDuty　　PWM 打开状态的占空比，取值范围：0~1

fOffDuty　　PWM 关闭状态的占空比，取值范围：0~1

返回值：错误代码

short dmc_get_pwm_onoff_duty(WORD CardNo, WORD PwmNo, double* fOnDuty, double* fOffDuty)

功　能：读取 PWM 开关状态对应占空比的设置

参　数：CardNo　　卡号

PwmNo　　PWM 通道，取值范围：0~1

fOnDuty　　返回 PWM 打开状态的占空比设置值

fOffDuty　　返回 PWM 关闭状态的占空比设置值

返回值：错误代码

short dmc_conti_set_pwm_output(WORD CardNo, WORD Crd, WORD pwm_no, double fDuty, double fFre)

功　能：连续插补中 PWM 输出设置

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

pwm_no　　PWM 通道，取值范围：0~1

fDuty　　占空比，取值范围：0~1

fFre　　频率，取值范围：0~2MHz

返回值：错误代码

注　意：

1） 该函数作用为设置 PWM 参数，调用该函数时不会有 PWM 信号输出；

2） 该函数需配合 dmc_conti_write_pwm 、 dmc_conti_delay_pwm_to_start 、 dmc_conti_ahead_pwm_to_stop、等函数使用时才能输出 PWM 信号

short dmc_conti_set_pwm_follow_speed(WORD CardNo, WORD Crd, WORD pwm_no, WORD

mode, double MaxVel, double MaxValue, double OutValue)

功　能：连续插补中 PWM 速度跟随

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

pwm_no　　PWM 通道，取值范围：0~1

mode　　跟随模式：

0：不跟随，保持状态

1：不跟随，输出低电平

2：不跟随，输出高电平

3：跟随，占空比自动调整

4：跟随，频率自动调整

MaxVel　　最大运行速度，单位：unit/s

MaxValue　　最大输出值：

跟随模式为 3 时：占空比，取值范围：0~1

跟随模式为 4 时：频率，取值范围：0~2MHz

OutValue　　固定输出值：

跟随模式为 3 时：频率，取值范围：0~2MHz

跟随模式为 4 时：占空比，取值范围：0~1

返回值：错误代码

注　意：1）当设置跟随模式为 3，即占空比自动调整时，运行速度（0~MaxVel）与占空比（0~MaxValue）成线性关系，参数 OutValue 确定 PWM 输出频率。

2）当设置跟随模式为 4，即频率自动调整时，运行速度（0~MaxVel）与输出频率（0~MaxValue）成线性关系，参数 OutValue 确定 PWM 输出占空比。

short dmc_conti_get_pwm_follow_speed(WORD CardNo, WORD Crd, WORD pwm_no, WORD* mode, double* MaxVel, double* MaxValue, double* OutValue)

功　能：读取 PWM 速度跟随参数设置

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

pwm_no　　PWM 通道，取值范围：0~1

mode　　返回跟随模式设置值

MaxVel　　返回最大运行速度设置值

MaxValue　　返回最大输出占空比或频率设置值

OutValue　　返回固定输出频率或占空比设置值

返回值：错误代码

short dmc_conti_delay_pwm_to_start(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double delay_value, WORD delay_mode, double ReverseTime)

功　能：连续插补中相对于轨迹段起点 PWM 滞后输出

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

pwm_no　　PWM 通道，取值范围：0~1

on_off　　输出状态，0：关闭，1：打开

delay_value　滞后值，单位：s（滞后时间模式）或 unit（滞后距离模式）

delay_mode　　滞后模式，0：滞后时间，1：滞后距离

ReverseTime　保留参数，固定值为 0

返回值：错误代码

注　意：1）调用此指令前，必须先调用函数 dmc_set_pwm_onoff_duty 设置 PWM 关闭及打开状态的占空比。PWM 波形的频率为之前调用 PWM 时设置的频率

2）当函数 dmc_conti_set_pwm_follow_speed 设置的跟随模式为 0 时：

参数 on_off 设置为 0（关闭状态）时，输出 PWM 波形的占空比为函数 dmc_set_pwm_onoff_duty 设置的关闭状态占空比值；

参数 on_off 设置为 1（打开状态）时，输出 PWM 波形的占空比为函数 dmc_set_pwm_onoff_duty 设置的打开状态占空比值

3）当函数 dmc_conti_set_pwm_follow_speed 设置的跟随模式不为 0 时：

参数 on_off 设置为 0（关闭状态）时，输出 PWM 波形的占空比为函数 dmc_set_pwm_onoff_duty 设置的关闭状态占空比值；

参数 on_off 设置为 1（打开状态）时，输出 PWM 波形为函数 dmc_conti_set_pwm_follow_speed 设置的跟随模式样式

short dmc_conti_ahead_pwm_to_stop(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double ahead_value, WORD ahead_mode, double ReverseTime)

功　能：连续插补中相对于轨迹段终点 PWM 提前输出

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

pwm_no　　PWM 通道，取值范围：0~1

on_off　　输出状态，0：关闭，1：打开

ahead_value　提前值，单位：s（滞后时间模式）或 unit（滞后距离模式）

delay_mode　　提前模式，0：滞后时间，1：滞后距离
ReverseTime　　保留参数，固定值为 0

返回值：错误代码

注　意：同函数“dmc_conti_delay_pwm_to_start”

short dmc_conti_write_pwm(WORD CardNo, WORD Crd, WORD pwmno, WORD on_off, double ReverseTime)

功　能：连续插补中缓冲区立即 PWM 输出

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
pwm_no　　PWM 通道，取值范围：0~1
on_off　　输出状态，0：关闭，1：打开
ReverseTime　　保留参数，固定值为 0

返回值：错误代码

注　意：同函数“dmc_conti_delay_pwm_to_start”

10.14　圆弧限速

short dmc_set_arc_limit(WORD CardNo, WORD Crd, WORD Enable, double MaxCenAcc, double MaxArcError)

功　能：设置指定坐标系的圆弧限速参数

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
Enable　　使能状态，0：禁止，1：使能
MaxCenAcc　　保留参数
MaxArcError　　保留参数

返回值：错误代码

short dmc_get_arc_limit(WORD CardNo, WORD Crd, WORD* Enable, double* MaxCenAcc, double* MaxArcError)

功　能：读取指定坐标系的圆弧限速参数

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
Enable　　使能状态，0：禁止，1：使能

MaxCenAcc 保留参数
MaxArcError 保留参数
返回值：错误代码

10.15 AD/DA 功能

short dmc_set_da_enable(WORD CardNo,WORD enable)
功 能：设置 DA 输出使能
参 数：CardNo 控制卡卡号
enable DA 使能状态，0：禁止，1：使能
返回值：错误代码

short dmc_get_da_enable(WORD CardNo,WORD* enable)
功 能：读取 DA 输出使能设置
参 数：CardNo 控制卡卡号
WORD enable DA 使能状态，0：禁止，1：使能
返回值：错误代码

short dmc_set_da_output(WORD CardNo, WORD channel,double Vout)
功 能：设置 DA 输出
参 数：CardNo 控制卡卡号
Channel DA 输出口号，取值范围 0、1
Vout DA 输出电压，输出电压范围-10V~10V
返回值：错误代码

short dmc_get_da_output(WORD CardNo, WORD channel,double* Vout)
功 能：读取 DA 输出设置
参 数：CardNo 控制卡卡号
Channel DA 输出口号，取值范围 0、1
Vout DA 输出电压，输出电压范围-10V~10V
返回值：错误代码

short dmc_get_ad_input(WORD CardNo,WORD channel,double* Vout)
功 能：读取 AD 输入

参　数：CardNo　　控制卡卡号
　　Channel　　AD 输入口号，取值范围 0~7
　　Vout　　AD 输入电压，输入电压范围-10V~10V
返回值：错误代码

short dmc_get_ad_input_all(WORD CardNo, double* Vout)
功　能：读取所有 AD 输入
参　数：CardNo　　控制卡卡号
　　Vout　　8 个通道的 AD 输入电压数组，输入电压范围-10V~10V
返回值：错误代码

10.16　连续插补 DA 跟随

short dmc_conti_set_da_follow_speed(WORD CardNo, WORD Crd, WORD da_no, double MaxVel, double MaxValue, double acc_offset, double dec_offset, double acc_dist, double dec_dist)
功　能：连续插补中 DA 速度跟随
参　数：CardNo　　卡号
　　Crd　　坐标系号，取值范围：0~3
　　da_no　　DA 通道，取值范围：0~1
　　MaxVel　　DA 跟随最大速度，单位：unit/s
　　MaxValue　　DA 跟随最大输出值，单位 V
　　acc_offset　　加速段偏差，单位 V
　　dec_offset　　减速段偏差，单位 V
　　acc_dist　　保留参数
　　dec_dist　　保留参数
返回值：错误代码

short dmc_conti_get_da_follow_speed(WORD CardNo, WORD Crd, WORD da_no, double *MaxVel, double *MaxValue, double *acc_offset, double *dec_offset, double *acc_dist, double *dec_dist)
功　能：读取连续插补中 DA 跟随速度
参　数：CardNo　　卡号
　　Crd　　坐标系号，取值范围：0~3
　　da_no　　DA 通道，取值范围：0~1

MaxVel　　DA 跟随最大速度
MaxValue　　DA 跟随最大输出值
acc_offset　　加速段偏差
dec_offset　　减速段偏差
acc_dist　　保留参数
dec_dist　　保留参数

返回值：错误代码

short dmc_conti_set_da_enable(WORD CardNo, WORD Crd, WORD enable,WORD channel,long mark);

功　能：连续缓冲区内的 da 跟随使能

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
enable　　DA 跟随，0：禁止，1：使能
channel　　通道号
mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

short dmc_conti_set_encoder_da_follow_enable(WORD CardNo, WORD Crd,WORD axis,WORD enable)

功　能：DA 单轴编码器速度跟随使能

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
axis　　轴号
enable　　编码器跟随，0：禁止 1：使能

返回值：错误代码

注　意：1）启用了 DA 跟随编码器速度后，DA 跟随插补合速度功能无效。

short dmc_conti_get_encoder_da_follow_enable(WORD CardNo, WORD Crd,WORD* axis,WORD* enable);

功　能：读取 DA 单轴编码器速度跟随使能

参　数：CardNo　　卡号
Crd　　坐标系号，取值范围：0~3
axis　　轴号

enable 编码器跟随，0：禁止 1：使能

返回值：错误代码

10.17 二维高速位置比较

short dmc_hcmp_2d_set_enable(WORD CardNo,WORD 2dhcmp, WORD cmp_enable)

功 能：设置高速比较使能

参 数：CardNo 卡号

2dhcmp 保留，参数值为 0

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 2dhcmp, WORD *cmp_enable)

功 能：读取高速比较使能

参 数：CardNo 控制卡卡号

2dhcmp 保留，参数值为 0

cmp_enable 返回二维高速比较器使能

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：配置二维高速比较器

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。

2）当使用高速二维比较功能时，必须确保配置的输出口有效电平与输出口的当前电平相反，如若相同，必须在使能高速比较功能之前，调用 dmc_write_outbit 将输出口的状态置反，然后再使能高速比较功能，配置高速输出口的有效电平。否则比较功能不能得到预期效果，该输出口将一直维持比较前的状态（如配置的高速输出口的有效电平 cmp_logic 为高电平，输出口当前状态也为高电平，则比较输出不能得到预期效果）。

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: 读取二维高速比较器

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：配置二维比较 PWM 输出模式

参　数：CardNo　　控制卡卡号

hcmp 参数值固定为 0
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：读取配置的二维比较 PWM 输出模式

参 数：CardNo 控制卡卡号
hcmp 参数值固定为 0
pwm_enable 返回 pwm 使能
duty 返回占空比
freq 返回频率
pwm_number 返回输出的 pwm 脉冲数

返回值：错误代码

short dmc_hcmp_2d_clear_points(WORD CardNo,WORD 2dhcmp)

功 能：清除所有缓冲区二维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 CardNo,WORD 2dhcmp, double x_cmp_pos, double y_cmp_pos,WORD cmp_outbit)

功 能：添加/更新二维高速比较位置

参 数：CardNo 控制卡卡号
2dhcmp 保留，参数值为 0
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，范围（14~15）

返回值：错误代码

short dmc_hcmp_2d_get_current_state_unit(WORD CardNo,WORD 2dhcmp, int *remained_points, double* x_current_point, double* y_current_point, int *runned_points, WORD *current_state, WORD *current_outbit)

功　能：读取二维高速比较参数

参　数：CardNo　　控制卡卡号

2dhcmp　　保留，参数值为 0

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，范围（14~15）

返回值：错误代码

short dmc_hcmp_2d_force_output(WORD CardNo,WORD hcmp,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

hcmp　　保留，参数值为 0

enable　　使能标识，1：使能　0：失能

返回值：错误代码

short dmc_hcmp_2d_set_config(WORD CardNo,WORD hcmp,WORD cmp_mode,WORD x_axis, WORD x_cmp_source, WORD y_axis, WORD y_cmp_source, long error,WORD cmp_logic,long time,WORD pwm_enable,double duty,long freq,WORD port_sel,WORD pwm_number);

功　能：配置二维高速比较器

参　数：CardNo　　控制卡卡号

hcmp　　参数值固定为 0

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

轴号范围：DMC5410A：0~3，DMC5610：0~6，DMC5810/DMC5C10：0~8

x_cmp_source　x 轴比较位置源：0：指令位置，1：反馈位置

y_axis　　y 轴关联轴号
　　轴号范围：DMC5410A：0~3，DMC5610：0~6，DMC5810/DMC5C10：0~8
y_cmp_source　y 轴比较位置源：0：指令位置，1：反馈位置
Error　　x/y 轴误差带设置，单位：pulse
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
pwm_enable　　pwm 模式使能
duty　　占空比
freq　　频率
port_sel　　输出口号，二维高速比较输出时，输出口端口号范围：14~15；当 pwm 使能后，此输出口输出 pwm 波形，输出端口号范围：14~15。
pwm_number　　输出的 pwm 脉冲数

返回值：错误代码

注　意：1）如果 2 次高速比较输入相距很近，则自动重合为一个脉 冲。
2)　PWM 模式使能后，通过 port_sel 设置的输出口号输出 PWM 波形。
3）老版本指令，该函数功能同 dmc_hcmp_2d_set_config_unit，建议使用带 unit 指令
4）当使用高速二维比较功能时，必须确保配置的输出口有效电平与输出口的当前电平相反，如若相同，必须在使能高速比较功能之前，调用 dmc_write_outbit 将输出口的状态置反，然后再使能高速比较功能，配置高速输出口的有效电平。否则比较功能不能得到预期效果，该输出口将一直维持比较前的状态（如配置的高速输出口的有效电平 cmp_logic 为高电平，输出口当前状态也为高电平，则比较输出不能得到预期效果）。

short dmc_hcmp_2d_get_config(WORD CardNo,WORD hcmp,WORD *cmp_mode,WORD *x_axis, WORD *x_cmp_source, WORD *y_axis, WORD *y_cmp_source, long *error,WORD *cmp_logic,long *time,WORD *pwm_enable,double *duty,long *freq,WORD *port_sel,WORD *pwm_number);

功　能: 读取二维高速比较器

参　数：CardNo　　控制卡卡号
hcmp　　参数值固定为 0
cmp_mode　　返回比较模式：
　　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
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
error 返回 x/y 轴误差带设置，单位：pulse
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
pwm_enable pwm 模式使能
duty 占空比
freq 频率
port_sel 输出口选择，端口号范围：14~15
pwm_number 输出的 pwm 脉冲数

返回值：错误代码

short dmc_hcmp_2d_add_point(WORD CardNo,WORD 2dhcmp, long x_cmp_pos, long y_cmp_pos)

功 能：添加/更新二维高速比较位置（老版本指令，建议使用新指令）

参 数：CardNo 控制卡卡号
2dhcmp 保留，参数值为 0
x_cmp_pos 队列模式下：添加 x 比较位置，单位：pulse
y_cmp_pos 队列模式下：添加 x 比较位置，单位：pulse

返回值：错误代码

备 注：老版本指令，该函数功能同 dmc_hcmp_2d_add_point_unit，建议使用带 unit 指令

short dmc_hcmp_2d_get_current_state(WORD CardNo,WORD 2dhcmp, long *remained_points, long* x_current_point, long* y_current_point, long *runned_points, WORD *current_state)

功 能：读取二维高速比较参数

参 数：CardNo 控制卡卡号
2dhcmp 保留，参数值为 0
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

返回值：错误代码

备　注：老版本指令，该函数功能同 dmc_hcmp_2d_get_current_state_unit，建议使用带 unit 指令。

//批量配置

short dmc_hcmp_2d_fifo_set_mode(WORD CardNo,WORD hcmp, WORD fifo_mode);

功　能：启用缓存方式添加比较位置

参　数：CardNo　　控制卡卡号 0-7

　　hcmp　　高速二维位置比较器，0-1，默认为 0

　　fifo_mode　　是否启用缓存方式，0：不启用，1：启用

返回值：错误代码

short dmc_hcmp_2d_fifo_get_mode(WORD CardNo,WORD hcmp, WORD* fifo_mode);

功　能：缓存方式添加比较位置模式回读

参　数：CardNo　　控制卡卡号 0-7

　　hcmp　　高速二维位置比较器，0-1，默认为 0

　　fifo_mode　回读是否启用缓存方式，0：不启用，1：启用

返回值：错误代码

short dmc_hcmp_2d_fifo_get_state(WORD CardNo,WORD hcmp,long *remained_points);

功　能：读取剩余缓存状态，上位机通过此函数判断是否继续添加比较位置

参　数：CardNo　　控制卡卡号 0-7

　　hcmp　　高速二维位置比较器，0-1，默认为 0

　　remained_points　剩余缓存

返回值：错误代码

备注：实际控制卡最大 25000+256 个缓存点，如果添加 25000 个点，读剩余缓存为 256；

short dmc_hcmp_2d_fifo_clear_points(WORD CardNo,WORD hcmp);

功　能：清除比较位置,也会把 FPGA 的位置同步清除掉

参　数：CardNo　　控制卡卡号 0-7

　　hcmp　高速二维位置比较器，0-1，默认为 0

返回值：错误代码

//添加大数据，会堵塞一段时间，直到数据添加完成

short dmc_hcmp_2d_fifo_add_table(WORD CardNo,WORD hcmp, WORD num,double *x_cmp_pos,double *y_cmp_pos);

功　能：按数组的方式批量添加比较位置

参　数：CardNo　　控制卡卡号 0-7

hcmp　高速二维位置比较器，0-1，默认为 0

num　数据点数，单次最多 100 点

x_cmp_pos X 轴比较位置数组

y_cmp_pos Y 轴比较位置数组

返回值：错误代码

10.18　连续插补位置跟随

short dmc_conti_gear_unit(WORD CardNo,WORD Crd,WORD axis,double dist,WORD follow_mode,long imark)

功　能：设置轴位置跟随

参　数：CardNo　　卡号

Crd　　坐标系号，取值范围：0~3

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

dist　　跟随距离，单位：unit

follow_mode　保留，参数值为 0

mark　　标号，任意指定，0 表示自动编号

返回值：错误码

注　意：1）进行跟随运动的轴不能是插补坐标系中的轴；发送该指令时轴如果在运动中，指令将不能正常执行

2）跟随距离是相对当前位置的相对距离，即下一段插补段运动过程中，跟随轴需要运动的距离。

10.19　螺距补偿功能

short dmc_enable_leadscrew_comp(WORD CardNo, WORD axis,WORD enable);

功　能：设置螺距补偿的使能与禁止

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：

0~5，DMC5410A：0~3

enable 螺距补偿使能状态，0：禁止，1：使能

返回值：错误码

short dmc_set_leadscrew_comp_config_unit(WORD CardNo,WORD axis,WORD n,double startpos,double lenpos,double *pCompPos,double *pCompNeg);

功 能：配置螺距补偿参数

参 数：CardNo 卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

N 点数

Startpos 补偿起始位置，单位 unit

Lenpos 补偿段的总长度，单位 unit

pCompPos 对应为正方向运动时，各点位置需要补偿的位置值，单位 unit

pCompNeg 对应为负方向运动时，各点位置需要补偿的脉冲数，单位 unit

返回值：错误码

注 意：1）最大补偿点数 256。

2）螺距补偿的长度不可设置为负值，补偿位置范围为从补偿起始位置的绝对位置往后的补偿长度。

3）补偿范围内的 N 段，正向经过补偿段时按 pCompPos 设置的参数进行补偿，负向经过补偿段时按 pCompNeg 设置的参数进行补偿。

short dmc_get_leadscrew_comp_config_unit(WORD CardNo,WORD axis,WORD* n,double* startpos,double* lenpos,double *pCompPos,double *pCompNeg);

功 能：读取螺距补偿参数

参 数：CardNo 卡号

axis 指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

N 返回点数

Startpos 返回补偿起始位置，单位 unit

Lenpos 返回补偿段的总长度，单位 unit

pCompPos 返回正方向运动时，各点位置需要补偿的脉冲数，单位 unit

pCompNeg 返回负方向运动时，各点位置需要补偿的脉冲数，单位 unit

返回值：错误码

short dmc_get_position_ex_unit(WORD CardNo, WORD axis, double *pos)

功　能：读取指定轴的螺距补偿后的指令位置

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

pos　　返回当前位置值，单位：unit

返回值：错误代码

short dmc_set_leadscrew_comp_config(WORD CardNo,WORD axis,WORD n,long startpos,long lenpos,long *pCompPos,long *pCompNeg);

功　能：配置螺距补偿参数

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

N　　点数

Startpos　　补偿起始位置

Lenpos　　补偿段的总长度

pCompPos　　对应为正方向运动时，各点位置需要补偿的脉冲数

pCompNeg　　对应为负方向运动时，各点位置需要补偿的脉冲数

返回值：错误码

注　意：1）最大补偿点数 256。

2）螺距补偿的长度不可设置为负值，补偿位置范围为从补偿起始位置的绝对位置往后的补偿长度。

3）补偿范围内的 N 段，正向经过补偿段时按 pCompPos 设置的参数进行补偿，负向经过补偿段时按 pCompNeg 设置的参数进行补偿。

4）该函数功能同 dmc_set_leadscrew_comp_config_unit，建议使用带 unit 指令

short dmc_get_leadscrew_comp_config(WORD CardNo,WORD axis,WORD* n,long* startpos,long* lenpos,long *pCompPos,long *pCompNeg);

功　能：读取螺距补偿参数

参　数：CardNo　　卡号

axis　　指定轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3

N　　返回点数

Startpos　　返回补偿起始位置
Lenpos　　返回补偿段的总长度
pCompPos　　返回正方向运动时，各点位置需要补偿的脉冲数
pCompNeg　　返回负方向运动时，各点位置需要补偿的脉冲数

返回值：错误码

10.20　龙门功能

short　dmc_set_gear_follow_profile(WORD　CardNo,WORD　axis,WORD,　enable,WORD master_axis,double ratio);

功　能：设置龙门跟随模式参数

参　数：CardNo　　卡号
Axis　　跟随轴轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
enable　　使能状态，0：禁止，1：使能
master_axis　　跟随主轴轴号，取值范围：DMC5C10：0~11，DMC5810：0~7，DMC5610：0~5，DMC5410A：0~3
ratio　　保留参数，设为 1

返回值：错误代码

注　意：1）龙门功能允许一对一关系，即一个主轴对应一个从轴，允许多对一，即多个从轴一个主轴，不允许一个从轴多个主轴
2）龙门模式的跟随轴没有硬限位和软限位，主轴遇到限位停止时，跟随轴同样停止
3）如果某个轴被设置为龙门模式的跟随轴后，check done 返回永远是运动状态，龙门关系必须手动解除
4）龙门模式主轴或从站报警 ALM，龙门运动将停止
5）龙门主轴允许与第三轴作插补运动
6）龙门功能函数调用有先后关系要求，需先调用 dmc_set_gear_follow_profile 建立龙门关系，再调用 dmc_set_grant_error_protect 设置龙门误差带，不可对调先后顺序，否则会容易报错误差超限制。

short　dmc_get_gear_follow_profile(WORD　CardNo,WORD,　axis,WORD*　enable,WORD* master_axis,double* ratio);

功　能：读取龙门跟随模式参数

参　数：CardNo　　卡号

Axis 跟随轴轴号，取值范围：0~11
enable 使能状态，0：禁止，1：使能
master_axis 跟随主轴轴号，取值范围：0~11
ratio 保留
返回值：错误代码

short dmc_set_grant_error_protect(WORD CardNo, WORD axis,WORD enable,DWORD dstp_error, DWORD emg_error)
功 能：设置龙门模式主从轴编码器跟随误差停止阀值
参 数：CardNo 卡号
axis 主轴轴号
enable 使能状态，0：禁止，1：使能
dstp_error 减速停止的误差阀值，单位：pulse
emg_error 立即停止的误差阀值，单位：pulse
返回值：错误码

short dmc_set_grant_error_protect_unit(WORD CardNo, WORD axis,WORD enable,double dstp_error, double emg_error)
功 能：设置龙门模式主从轴编码器跟随误差停止阀值
参 数：CardNo 卡号
axis 主轴轴号
enable 使能状态，0：禁止，1：使能
dstp_error 减速停止的误差阀值，单位：unit
emg_error 立即停止的误差阀值，单位：unit
返回值：错误码

short dmc_get_grant_error_protect(WORD CardNo, WORD axis,WORD* enable,DWORD* dstp_error, DWORD* emg_error)
功 能：读取龙门模式编码器位置跟随误差停止模式设置
参 数: CardNo 卡号
axis 主轴轴号
Enable 返回停止使能标志
dstp_error 返回减速停止误差阀值，单位：pulse
emg_error 返回立即停止误差阀值，单位：pulse

返回值：错误码

short dmc_get_grant_error_protect_unit(WORD CardNo, WORD axis,WORD* enable,double* dstp_error, double* emg_error)

功　能：读取龙门模式编码器位置跟随误差停止模式设置

参　数: CardNo　　卡号

axis　　主轴轴号

Enable　　返回停止使能标志

dstp_error　　返回减速停止误差阀值，单位：unit

emg_error　　返回立即停止误差阀值，单位：unit

返回值：错误码

10.21　IO 触发减速停止

short dmc_set_io_exactstop (WORD CardNo, WORD axis, WORD ioNum， WORD * ioList, WORD enalbe, WORD valid_logic, WORD action, WORD move_dir)

功　能：配置 IO 触发减速停止参数

参　数：CardNo　　指定卡号

axis　　指定轴号

ioNum　　输入数量，范围 1~16

ioList　　输入端口号数组

enable　　IO 逻辑减速停止使能, 1：打开此功能，0：关闭此功能

valid_logic　与/或之后的有效触发逻辑，1：高电平触发，0：低电平触发

action　　0：逻辑与，1：逻辑或

move_dir　　0：正方向停止，1：负方向停止

返回值：错误代码

说　明：配置减速停止相关参数。其中 move_dir 表示检测方向，如果设置为 0，只有速度方向为正向运动时候，减速停止才会起作用；单轴运动减速时间由单轴异常减速停止时间函数（dmc_set_dec_stop_time）设置，插补运动减速时间由插补异常减速停止时间函数（dmc_set_vector_dec_stop_time）设置。

short dmc_set_dec_stop_dist(WORD CardNo, WORD axis,int dist)

功　能：设置 IO 触发减速停止距离

参　数：CardNo　　指定卡号

axis　　指定轴号
dist　　停止距离，单位：pulse
返回值：错误代码

10.22　圆形区域限位功能

short dmc_set_arc_zone_limit_config_unit(WORD CardNo, WORD* AxisList, WORD AxisNum, double *Center, double Radius, WORD Source, WORD StopMode)
功　能：配置圆形区域限位参数
参　数：CardNo　　指定卡号
AxisList　　需要限位的轴号列表
AxisNum　　需要限位的轴个数，固定为 2
Center　　限位区域圆心，unit 单位
Radius　　限位区域半径，unit 单位
Source　　位置判断源，0 指令位置，1 反馈位置
StopMode　　限位触发后停止模式， 0 减速停止；1 急停
返回值：错误代码

short dmc_get_arc_zone_limit_config_unit(WORD CardNo, WORD* AxisList, WORD* AxisNum, double *Center, double * Radius, WORD* Source, WORD* StopMode)
功　能：获取配置的圆形区域限位参数
参　数：CardNo　　指定卡号
AxisList　　回读的需要限位的轴号列表
AxisNum　　回读的需要限位的轴个数，固定为 2
Center　　回读的限位区域圆心，unit 单位
Radius　　回读的限位区域半径，unit 单位
Source　　回读的位置源，0 指令位置，1 反馈位置
StopMode　　回读的限位触发后停止模式， 0 减速停止；1 急停
返回值：错误代码

short dmc_set_arc_zone_limit_enable(WORD CardNo, WORD enable)
功　能：圆弧限位功能使能
参　数：CardNo　　指定卡号
enable　　使能状态，0 未使能；1 使能

返回值：错误代码

short dmc_get_arc_zone_limit_enable(WORD CardNo, WORD* enable)
功　能：获取圆弧限位功能的使能状态
参　数：CardNo　　指定卡号
　　enable　　回读的使能状态，0 未使能；1 使能
返回值：错误代码

short dmc_get_arc_zone_limit_axis_status(WORD CardNo, WORD AxisNo)
功　能：查询相应轴的状态
参　数：CardNo　　指定卡号
　　AxisNo　　轴号
返回值：轴的限位状态；-1 该轴未设置区域限位；0 该轴在区域内；1 该轴触发区域限位

short dmc_set_arc_zone_limit_config(WORD CardNo, WORD* AxisList, WORD AxisNum, double *Center, double Radius, WORD Source, WORD StopMode)
功　能：配置圆形区域限位参数
参　数：CardNo　　指定卡号
　　AxisList　　需要限位的轴号列表
　　AxisNum　　需要限位的轴个数，固定为 2
　　Center　　限位区域圆心，pulse 单位
　　Radius　　限位区域半径，pulse 单位
　　Source　　位置判断源，0 指令位置，1 反馈位置
　　StopMode　　限位触发后停止模式， 0 减速停止；1 急停
返回值：错误代码
备　注：该函数功能同 dmc_set_arc_zone_limit_config_unit,建议使用带 unit 指令。

short dmc_get_arc_zone_limit_config(WORD CardNo, WORD* AxisList, WORD* AxisNum, double *Center, double * Radius, WORD* Source, WORD* StopMode)
功　能：获取配置的圆形区域限位参数
参　数：CardNo　　指定卡号
　　AxisList　　回读的需要限位的轴号列表
　　AxisNum　　回读的需要限位的轴个数，固定为 2
　　Center　　回读的限位区域圆心，pulse 单位

Radius　　回读的限位区域半径，pulse 单位
Source　　回读的位置源，0 指令位置，1 反馈位置
StopMode　　回读的限位触发后停止模式， 0 减速停止；1 急停
返回值：错误代码

10.23　看门狗功能

short dmc_set_watchdog_action_event(WORD CardNo, WORD event_mask)
功　能：设置看门狗触发响应事件
参　数：CardNo　　卡号
event_mask　　看门狗触发事件：bit0 停止发脉冲；bit1 停止轴使能；bit2 输出口复位，包括扩展口
返回值：错误代码

short dmc_get_watchdog_action_event(WORD CardNo, WORD* event_mask)
功　能：读取看门狗触发响应事件
参　数：CardNo　　卡号
event_mask　　回读配置的看门狗触发事件
返回值：错误代码

short dmc_set_watchdog_enable (WORD CardNo, double timer_period, WORD enable)
功　能：使能看门狗保护机制
参　数：CardNo　　卡号
timer_period　　看门狗触发时间，单位：ms。设置为 0 看门狗功能将无效
enable　　使能标志，0 关闭看门狗；1 开启看门狗
返回值：错误代码

short dmc_get_watchdog_enable (WORD CardNo, double * timer_period, WORD* enable)
功　能：回读看门狗状态
参　数：CardNo　　卡号
timer_period　　回读的看门狗触发时间
enable　　回读的使能标志
返回值：错误代码

short dmc_reset_watchdog_timer (WORD CardNo)

功　能：复位看门狗定时器

参　数：CardNo　　卡号

返回值：错误代码

备　注：上位机需要在设定的时间内不断喂狗（复位看门狗定时器），以保证控制卡和上位机通讯正常，否则会触发看门狗事件；

10.24　椭圆插补及切向跟随

short dmc_ellipse_move (WORD CardNo, WORD Crd, WORD AxisNum, WORD *AxisList,double* TargetPos,double* CenPos,double ALen,double BLen, WORD Dir, WORD PosMode)

功　能：启动椭圆插补运动

参　数：CardNo　　卡号

Crd　　坐标系号 0-3

AxisNum　　坐标系轴数量，保留参数，固定为 2

AxisList　　坐标系轴列表

TargetPos　　目标位置列表

CenPos　　椭圆圆心位置

Alen　　长半轴长度

Blen　　短半轴长度

Dir　　插补方向，0 顺时针插补，1 逆时针插补

PosMode　　插补模式，0 相对运动，1 绝对运动

返回值：错误代码

short dmc_set_tangent_follow (WORD CardNo, WORD Crd, WORD Axis, WORD FollowCurve, WORD RotDir,double DegreeEquiv)

功　能：设置切向跟随参数

参　数：CardNo　　卡号

Crd　　坐标系号 0-3

Axis　　跟随轴号

FollowCurve　　跟随曲线类型，0 跟随圆弧； 1 跟随椭圆

RotDir　　跟随轴旋转方向， 0 跟随轴负向旋转， 1 跟随轴正向旋转

DegreeEquiv　　角度当量，单位 pulse/degree，跟随轴每旋转一度，控制卡下发的

脉冲数

返回值：错误代码

short dmc_get_tangent_follow_param (WORD CardNo, WORD Crd, WORD * Axis, WORD * FollowCurve, WORD * RotDir, double* DegreeEquiv)

功　能：读取切向跟随参数

参　数：CardNo　　卡号

Crd　　坐标系号 0-3

Axis　　回读的跟随轴号

FollowCurve　　回读的跟随曲线类型

RotDir　　回读的跟随轴旋转方向

DegreeEquiv　　回读的角度当量

返回值：错误代码

short dmc_disable_follow_move (WORD CardNo, WORD Crd)

功　能：取消指定坐标系的跟随运动

参　数：CardNo　　卡号

Crd　　坐标系号 0-3

返回值：错误代码

附　录

附录 1　运动函数错误码说明

错误码	含义
0	无错误
-1	内部的一些空指针返回值
2	动态库层参数错误
3	PCI 通讯超时
4	动态库层检测到轴处于运动中
6	连续插补错误
8	无法连接错误
9	动态库层卡号错误
10	动态库层发送数据失败，请检查 PCI 槽位是否松动
12	固件文件错误
14	动态库层下载固件与控制卡/器型号不匹配
17	不支持的功能，预留，不能更改
19	链接对应型号类型不匹配
20	固件参数错误
22	固件当前状态不允许操作
24	固件不支持功能，保留原版本的错误码
25	密码错误
26	密码错误输入次数受限
1000	设置的轴号超出最大轴数或伺服轴数允许范围
1001	该轴已经被其它运动占用，不支持对该轴进行操作
1002	不支持该操作，轴或坐标系处于运动中，或轴 CHECK_DOWN 未完成
1003	不支持该操作，轴或坐标系处于运动或暂停中
1004	轴映射列表为空
1005	设置脉冲计数值和编码器反馈值之间差值的报警阀值为 0
1006	设置编码器和指令位置跟踪误差使能信号错误（使能信号不是 0 或 1）
1007	设置编码器和指令位置跟踪误差停止模式错误（停止模式不是 0 或 1）
1008	设置的脉冲当量小于等于 0
1009	QUEUE 队列初始化参数错误，初始化空间大小小于 0
1010	不支持该功能
1011	SPI CRC 校验码错误
1022	中断通道超限
1023	中断配置参数有误
1024	中断逻辑电平错误
1025	处于主从轴同步运动中
1026	数字为无穷大报错，请检查是否存在除零操作(分母为零)
1027	数字为无效数据，请检查是否存在零除以零(分子和分母都为零)
1028	轴运动状态异常，需清除轴异常状态再执行该操作
1029	轴运动状态异常，需清除轴异常状态再执行该操作
1030	急停保护状态，不允许操作轴运动和 IO 输出

1100	设置轴的目标位置大于轴软件正限位位置
1101	设置轴的目标位置小于轴软件负限位位置
1102	软件负限位位置大于或等于正限位位置，不允许使能限位功能
1103	圆弧区域限位
1104	区域限位半径为 0
1105	设置的限位维数大于 2 维
1106	设置的二维限位轴号相同
1107	设置的限位停止模式大于 1（取值只能 0 和 1）
1108	设置两轴碰撞检测时，两轴轴号相同
1110	设置安全区域检测，轴数错误
1111	设置安全区域检测，轴号相同
1112	设置安全区域检测，区域数量错误
1113	设置安全区域检测，轴位置区间左界和右界干涉
1114	设置安全区域检测，轴处于该区域内，禁止运动
1200	该轴已被配置为 PWM 输出轴，不允许其它操作
1201	该轴已被配置为 PWM 输出轴，1D 和 2D 比较输出使能失败，PWM 与比较输出互斥
1202	PWM 通道超限（共支持 2 通道）
1203	PWM 跟随模式超限（0~4:5 种模式）
1204	设置 PWM 占空比大于 1
1205	设置 PWM 频率超出范围
1206	PWM 通道被 PWM 跟随占用
1300	回零目标速度小于等于 0 或大于 4M 错误
1301	回零加速度小于等于 0 错误
1302	（1）回零方向与回零偏移量冲突，正向限位回零，偏移量不能为正，负向同理；（2）回零方向与当前限位状态冲突
1303	设置回零限位反找偏移值小于 0
1304	设置回零模式超范围（回零模式小于 1 或者大于 13）
1305	设置回零偏移清零模式超限（0-回零后不做任何动作，1-回零之后，清零，然后偏移，2-回零之后，偏移，清零）
1306	设置回零模式失败，由于该回零模式下变速变位置使能信号有效
1307	原点锁存回零位移为 0
1308	驱动器 ALM 信号有效，回零启动失败
1309	轴 EMG 信号有效，回零启动失败
1310	轴 DSTP 信号有效，回零起动失败
1311	正负限位都有效
1312	轴未使能
1313	轴号超出范围
1314	回零周期设置错误，内部
1315	正在回原点
1316	轴 PDO 未配置控制模式 6060，无法启动回零
1400	TRACE 功能已经开启，请停止后再开启
1401	设置采样周期等于 0
1402	设置对象数超过限制
1403	采集功能已经启动

1404	设置的采集类型不对
1405	设置的采集长度不对
1406	读取时候的存储数组为空指针
1407	采集对象为 0
1408	采集数据已经空了
1409	采集功能内存不足
1410	采集功能已经使能
1411	采集功能未启动
1412	采集的 PDO 不存在
1450	采集的 PDO 不存在
1451	采集通道超限
1452	采集大小超限
1453	采集节点数超限
1454	从站没有 PDO 数据
1455	没有内存
1456	索引配置错误
1457	配置的索引数与索引不匹配
1458	索引中有不识别的对象
1459	从站没有配置任何的 PDO,从站不存在，2.初始化的从站信息失败
1460	配置的触发条件不存在
1461	输入的读取数据的首地址小于采集到的数据包数；2.没有成功触发采集
1462	ERR_PDO_TRACE_INPUT_NULL_POINTER= TRACR_BASE_ADDR + 62, //参数输入了空指针
1500	添加 PT 点,百分比小于 0 或大于 100
1501	添加 PT 点，添加点数超出最大允许值（1001）
1502	添加 PT 点，数据点个数超出最大允许值（1001）
1503	添加 PVT 点，添加点数超出最大允许值（5001）
1504	添加 PVT 点，数据点个数超出最大允许值（5001）
1505	启动 PVT 时，检测轴运动模式发现不是 PVT 模式，故报错
1506	PVT 或 PT 点数或轴号超出范围
1507	添加 PVT TABLE 时 BLOCKNUM 等于 0
1508	PVT START 失败
1509	驱动器 ALM 信号有效，PVT/PT 启动失败
1510	轴 EMG 信号有效，PVT/PT 启动失败
1511	PVT 运动轴数大于控制卡最大轴数，启动 PVT 失败
1512	没有数据
1513	时间为 0
1514	速度、加速度或者减速度小于等于 0
1515	速度小于 0
1516	与上一个位置一样
1517	方向改变了但是上一个位置的速度不为 0
1518	下载编号不匹配
1519	第一组数据必须为 0
1600	螺距补偿点数超出最大值（256）或小于最小值（2）
1601	螺距补偿距离小于等于 0

1602	螺距补偿轴数超出允许范围
1603	轴脉冲补偿时间小于等于 0
1604	扩展螺距补偿表失败，内存不足
1605	扩展螺距补偿表数据点数量错误
1606	扩展螺距补偿表未使能
1607	螺距补偿数据包 ID 错误
1608	数值错误
1700	如果不是主从轴关系，则不予以解除龙门关系
1701	主轴已经作为从轴使用,配置龙门主从轴关系失败
1702	从轴已经作为主轴使用,配置龙门主从轴关系失败
1703	从轴已经作为从轴使用,配置龙门主从轴关系失败
1704	龙门超差保护设置减速停止误差大于等于急停误差错误
1705	获取指定从轴的主轴号失败，两轴不存在主从关系
1706	该轴为龙门主轴，不允许操作，请解除龙门关系后再操作
1707	主轴和从轴编码器计数模式不一致
1708	轴未使能
1709	没有进入龙门模式
1710	主轴超出范围
1711	从轴超出范围
1712	退出 GANTRY 模式, FOLLOW 模式，必须在主轴停止状态
1713	主轴在运动中不允许退出龙门
1714	从轴 ALM 信号有效
1715	从轴 EMG 信号有效
1716	从轴 DSTP 信号有效
1800	IO 映射轴参数或 IO 类型错误
1801	IO 输入输出口超出最大允许值（16）或映射轴号超出范围
1802	设置 IO 口状态值失败，已配置为一维高速比较输出,不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1803	设置 IO 口状态值失败，已配置为二维高速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1804	设置 IO 口状态值失败，已配置为一维低速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥
1805	设置 IO 口状态值失败，已配置为锁存输出，不允许对 IO 口进行操作，锁存输出和普通 IO 设置操作互斥
1806	获取 IO 值失败，IO 端口号超出有效范围（大于 801+16*8）
1807	设置全部输出口电平形参 PORTNO 错误（固定值为 0）
1808	设置 IO 延时输出翻转 IO 端口超出范围（大于 32 且小于 80）
1809	设置或读取轴 IO 映射轴号超出范围（大于最大轴数且不等于 15）
1810	配置或读取通用输入为轴的限位信号，输入端口超出范围（0~15）
1811	设置 IO 计数值，输入端口号超限（0~15）
1812	IO 通道实现 MS 级别的 PWM 输出器已用完
1813	IO 通道实现 MS 级别的 PWM 输出个数为 0
1814	IO 通道实现 MS 级别的 PWM 输出个数为 0
1815	IO 延时输出通道数超出范围
1816	设置 IO 口状态值失败，已配置为二维低速比较输出，不允许对 IO 口进行操作，比较输出和普通 IO 设置操作互斥

1817	设置 IO 口状态值失败，已配置为 PWM 输出，不允许对 IO 口进行操作
1818	IO 计数通道超限
1819	IO 计数模式错误
1820	设置 IO 口状态值失败，已配置锁存输出输出，不允许对 IO 口进行操作
1900	DA 通道超出范围（只有 0 和 1 两个通道）
1901	设置 DA 跟随最大速度等于 0
1902	模拟量输入通道超限（支持 8 路模拟量输入）
1903	连续插补 DA 跟随占用不允许操作
1904	DA 超范围
1950	接收数据包长度超过 PCI 最大接收数据包长度
1951	上传文件指令 ID 不匹配
1952	上传文件类型不匹配
1953	上传文件 FLASH 地址为空
1954	无效参数 ID，不支持该功能
1955	无效指令 ID，不支持该功能
1956	下载固件文件太长
1957	轴运动中，下载固件失败
1958	下载固件文件 ID 改变
1959	下载文件块号错误
1960	文件大小不匹配
1961	CRC 不匹配
1962	应答数据包长度超出 PCI 最大应答数据包长度
1963	上传固件文件 ID 改变
1964	上传文件块号错误
1965	下载固件指令 ID 错误
1966	在线升级过程中增加 ARM 层校验卡的型号，如果型号不对，则不予升级
1967	密码 ENTER 错误
1968	密码 ENTER 次数过多
1969	密码写入拒绝
1970	需要先进行密码登入
1971	数据保存位置超限
1972	下载文件过程中不允许其他命令通讯
1973	下载文件保存失败
1974	下载文件类型不匹配
1975	下载文件名不匹配
1976	下载变量文件号超过范围
1977	下载 BASIC 文件时文件还在运行中
1978	上传固件文件太长
1979	不支持上传文件
1980	下载文件名不匹配
1981	上传文件名不存在
1982	文件名错误
1983	文件太长
1984	预读取的字符长度不够
1985	下载文件不匹配，有可能是文件号和文件内容不匹配

2000	添加比较点功能参数和功能编号不匹配(或某值超出范围)
2001	比较缓冲区已满（256）
2002	比较点功能参数超出范围（0~15）
2003	比较组超出范围（最多 12 组：0~11）
2004	高速一维比较通道超出范围（4 通道：0~3）
2005	高速一维比较未使能，操作失败
2006	批量高速一维比较缓冲区满
2007	一维高速比较使能失败，已被配置为二维比较输出，1D 和 2D 互斥
2008	已经被配置为比较输出，PWM 使能失败，CMP 与 PWM 互斥
2009	比较通道超限或者轴数超限
2010	设置或读取一维比较参数配置，轴号超过最大值且不等于 255
2011	批量添加缓存比较点数大于 100
2012	高速比较缓冲区模式参数错误
2013	高速比较缓冲区模式方向参数错误
2014	高速比较时间参数设置错误
2015	高速一维比较器未使能
2016	一维比较器未使能
2017	一维比较输出口超出范围
2018	高速一维比较缓冲区已满
2100	比较组超出范围（最多 1：取值 0）
2101	二维高速比较使能失败，已被配置为一维比较输出，1D 和 2D 互斥
2102	高速二维比较配置 PWM 输出通道超限（大于等于 1 或小于 0）
2103	高速二维比较使能 PWM 输出信号有误（ENABLE>1）
2104	高速二维比较输出口设置超限（大于 15 或小于 12）
2105	添加高速二维比较点失败，FIFO 已满
2106	高速二维比较通道超出范围（1 通道：0）
2107	高速二维比较通道没有启用
2108	高速二维比较参数错误
2109	二维比较器未使能
2110	二维比较输出口超出范围
2111	高速二维比较缓冲区已满
2150	等间距比较超出通道数
2200	LTC 锁存通道超限（只有 2 通道）
2201	LTC 设置通用输出口超限（0~15）
2202	软锁存锁存器个数超限（0~15）
2203	锁存轴为 8 时，锁存源只支持编码器反馈位置锁存
2204	读取软锁存位置值无效
2205	软锁存轴号超限（0~15）
2206	软锁存缓冲存满
2207	锁存的滤波参数超出范围 0~64MS
2208	锁存功能未生效，可能是配置文件未配置生效
2209	原点锁存参数无效
2210	EZ 锁存参数无效
2300	CAN 状态设置模式超限（0-断开；1-连接；2-复位后自动连接）

2301	CAN 状态设置 CAN 节点超限（等于 0 或大于 8）
2302	接线盒未处于 LINK 状态
2400	设置的反向间隙补偿值小于 0
2401	读取反向间隙补偿值，保存反向间隙补偿值的地址无效
2500	主轴和从轴轴号相同冲突
2501	主轴已被占用，非空闲
2502	从轴已被占用，非空闲
2503	从轴未处于电子齿轮模式
2504	主轴已经配置为电子齿轮从轴，不允许重复配置为其它从轴的主轴
2505	从轴已经配置为电子齿轮主轴，不允许重复配置为其它从轴
2506	不支持跟随主轴的运动模式，目前支持主轴为 PMOVE 和 VMOVE 两种运动模式（内部错误码，请查看相应轴停止原因了解详情）
2507	设置同步状态超出范围（0-未同步，1-同步）
2508	设置跟随位置源错误（0-跟随主轴指令位置，1-跟随主轴编码器位置）
2509	设置电子齿轮分子为 0（不能为零，可以是正数或负数）
2510	设置电子齿轮分母小于或等于 0（正数）
2511	设置从轴跟随方向限制模式超限（0-不限制，正反方向都跟随(默认)　1-正方向跟随，负方向不跟随　2-负方向跟随，正方向不跟随）
2512	设置从轴跟随模式超限（0-速度同步（默认）　1-位置和速度同步）
2513	从轴已经配置为电子齿轮模式，不允许再次配置
2514	设置从轴追赶目标速度小于等于 0 或大于 4M 错误
2515	设置电子齿轮缓存模式错误（0-打断　1-等待）
2516	从轴已经配置为电子齿轮从轴，不允许重复配置为其它主轴的从轴
2517	GEARINPOS 模式，初始化时主轴运动方向错误（内部错误码，请查看相应轴停止原因了解详情）
2518	从轴开始启动时主轴绝对位置错误（不在主轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2519	同步时主轴绝对位置错误（不在主轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2520	从轴开始运动位置大于同步时主轴绝对位置（内部错误码，请查看相应轴停止原因了解详情）
2521	同步时从轴绝对位置（不在从轴运动方向上）（内部错误码，请查看相应轴停止原因了解详情）
2522	从轴追赶目标速度小于同步时从轴的速度，建议调大追赶目标速度（内部错误码，请查看相应轴停止原因了解详情）
2523	从轴同步位置过小，不足以从轴从当前速度直接加速或减速至同步速度,建议调大从轴同步位置或调大加减速度（内部错误码，请查看相应轴停止原因了解详情）
2524	从轴开始运动时主轴未达到匀速状态,建议调大从轴启动时主轴的位置（内部错误码，请查看相应轴停止原因了解详情）
2525	GEARINPOS 模式，设置从轴的最大加速度或减速度小于等于 0
2526	GEARINPOS 实际加速度大于设置的最大加速度（内部错误码，请查看相应轴停止原因了解详情）
2527	GEARINPOS 实际减速度大于设置的最大减速度（内部错误码，请查看相应轴停止原因了解详情）
2528	主轴空闲，不处于运动中，不允许在线配置电子齿轮比参数
2529	电子齿轮已经处于脱离耦合状态，操作失败

2530	从轴追赶实际可达速度大于追赶设置的最大目标速度,且以最大追赶速度运行无法实现规定时间内达到从轴的位移，建议调大追赶目标速度或减小从轴同步位置（内部错误码，请查看相应轴停止原因了解详情）
2531	从轴同步速度在只有加速或减速过程也不可达速度,建议调大从轴从轴同步位置或调小齿轮比或调大从轴开始运动的主轴位置（内部错误码，请查看相应轴停止原因了解详情）
2532	二分法求取合适的可达速度，迭代次数超过限制次数也无法找到合适的速度值（内部错误码，请查看相应轴停止原因了解详情）
2533	主轴 PMOVE 目标位置小于主轴同步位置（内部错误码，请查看相应轴停止原因了解详情）
2534	主轴 PMOVE 目标位置小于从轴开始运动时的主轴位置（内部错误码，请查看相应轴停止原因了解详情）
2535	从轴追赶过程加速周期或减速周期为 0，有可能主轴同步位置和从轴开始运动时的主轴位置相隔太近（内部错误码，请查看相应轴停止原因了解详情）
2536	主轴目标速度为零（内部错误码，请查看相应轴停止原因了解详情）
2537	从轴实际可达速度大于追赶速度时，重算以追赶速度为目标速度的匀速周期数小于等于零（内部错误码，请查看相应轴停止原因了解详情）
2538	反转禁止仅对模数轴有效，若禁止反转，而从轴不是模数轴则报错
2539	GEARINPOS 位置同步模式不支持在线变倍率
2540	GEARINPOS 主轴位移与规划结果不一致
2541	GEARINPOS 变倍率失败，输入主轴和从轴不属于主从轴关系
2542	上一次变倍率尚未处理完成，禁止再次触发变倍率
2543	从轴起始反向运动方向模式错误(1~4)
2544	从轴起始反向运动加速度为零
1545	回读主轴位置源设置错误：主轴轴号与设置主轴号不匹配
1546	电子齿轮从轴未处于使能状态，建立主从轴齿轮关系失败
1547	设置从轴的加速度小于等于 0
1548	设置从轴的减速度小于等于 0
1549	设置从轴的 JERK 小于 0
1550	设置的电子齿轮主轴与当前执行齿轮运动的主轴不匹配，执行电子齿轮失败
1551	EXECUTE 执行位超范围报错
1552	GEARINPOS 设置追赶阶段 X 轴或 Y 轴的位移量等于零
1553	主轴正向运动，从轴启动追赶时主轴位移小于等于 0 错误
1554	主轴正向运动，从轴同步位置小于等于从轴当前位置
1555	主轴正向运动，主轴同步位置小于等于主轴当前位置
1556	主轴正向运动，主轴同步位置小于等于从轴启动时主轴的位置
1557	主轴负向运动，从轴启动追赶时主轴位移大于等于 0 错误
1558	主轴负向运动，从轴同步位置大于等于从轴当前位置
1559	主轴负向运动，主轴同步位置大于等于主轴当前位置
1560	主轴负向运动，主轴同步位置大于等于从轴启动时主轴的位置
2600	凸轮轴未使能
2601	轴已经处于凸轮模式
2602	没进到凸轮模式
2603	主轴超出范围
2604	从轴超出范围
2605	主从轴同轴
2606	没有配置主轴

2607	添加的凸轮数据个数超出范围
2608	插入的凸轮数据有错误
2609	插入的凸轮数据表已满
2610	添加 CAM TABLE 时 BLOCKNUM 等于 0
2611	凸轮表编号超出允许范围
2612	凸轮表挺杆编号超出允许范围
2613	挺杆点列表已空，删除凸轮表挺杆点失败
2614	挺杆点列表已满，添加凸轮表挺杆点失败
2615	挺杆点列表找不到指定挺杆点信息(挺杆列表为空或者找不到匹配的挺杆编号)，读取挺杆点信息失败
2616	凸轮表凸轮点编号超出允许范围
2617	凸轮点列表找不到指定凸轮点信息(凸轮点列表为空或者找不到匹配的凸轮点编号)，读取指定凸轮点信息失败
2618	主轴位置小于 0，写凸轮信息失败
2619	从轴位置小于 0，写凸轮信息失败
2620	从轴速度小于 0，写凸轮信息失败
2621	从轴加速度小于 0，写凸轮信息失败
2622	挺杆正向经过模式设置超范围(0~3)
2623	挺杆负向经过模式设置超范围(0~3)
2624	挺杆的主轴位置小于 0，添加或写挺杆信息失败
2625	凸轮点数太少或太多(小于 3 或者大于 2048)
2626	指令模块类型错误，设置模块状态失败
2627	主轴缩放比例小于等于 0，CAM_IN 模块报错
2628	主轴缩放比例等于 0，CAM_IN 模块报错
2629	启动位置模式超范围，CAM_IN 模块报错
2630	啮合动作方式超范围，CAM_IN 模块报错
2631	从轴速度小于等于 0，CAM_IN 模块报错
2632	从轴加速度小于等于 0，CAM_IN 模块报错
2633	从轴减速度小于等于 0，CAM_IN 模块报错
2634	主轴位置来源超范围，CAM_IN 模块报错
2635	交接模式超范围，CAM_IN 模块报错
2636	挺杆点主轴位置大于主轴周期位置最大值，写挺杆点信息失败
2637	挺杆点主轴位置大于主轴周期位置最大值，增加挺杆点信息失败
2638	从轴加加速度小于 0，CAM_IN 模块报错
2639	启动啮合时，主轴位置超出主轴凸轮周期的最大值范围
2640	启动啮合时，主轴位置超出主轴的模允许范围
2670	旋切编号超限，范围：[1,8]
2671	旋切轴半径小于等于 0
2672	旋切刀头数量小于等于 0
2673	旋切进给轴半径小于等于 0
2674	旋切长度小于等于 0
2675	旋切同步开始位置小于等于 0
2676	旋切同步结束位置小于等于 0
2677	旋切刀具同步位移大于材料切割长度
2678	旋切组已被占用，请使用其它旋切组编号
2679	找不到指定旋转轴的旋切组编号，旋切组未激活或旋切轴有误
2680	旋切退出曲线结束时 X 轴位移大于切割长度

2701	输入参数报错为空指针，内部使用
2702	内存分配不成功，内部使用
2703	输入的跟随序号超过最大数限制
2704	轴未处于跟随模式
2705	条件不允许退出跟随模式
2706	条件不允许进入跟随模式
2707	参数错误
2708	跟随参数必须在插补系停止的时候设置
2709	跟随不支持
2710	跟随参数必须在插补系启动之前的时候设置
2721	输入参数报错为空指针，内部使用
2722	输入的跟随类型不对
2723	输入参数有错误
2724	状态机切换不正确，可能参数未设置，或者未使能
2741	输入参数报错为空指针，内部使用
2742	如果不是主从轴关系，则不予以解除
2743	检查从轴是否已经作为主轴使用
2744	误差检测值中 DSTP_ERROR 大于 EMG_ERROR 错误
2745	轴跟随功能没启用
2761	输入参数报错为空指针，内部使用
2762	PWM 通道未使能
2763	占空比小于 0 或者大于 1
2764	跟随模式超出范围
2765	跟随通道超出范围
2781	输入参数报错为空指针，内部使用
2782	内存分配不成功
2783	输入的跟随序号超过最大数限制
2801	与 RTCP 功能互斥
2802	没有配置参数
2803	维数错误
2811	跟随功能没有开启
2821	没有配置参数
2822	设置间隔为 0
2823	未使能
2824	等间距比较超出通道数
2825	等间距比较输出口超出范围
3000	目标速度小于等于零错误
3001	点位运动位移为零或规划时运动距离为零
3002	轴状态未处于完成或空闲状态，启动点位运动无效
3003	PMOVE 规划或设置的起跳速度小于 0

3004	PMOVE 规划或设置目标速度等于 0
3005	PMOVE 规划或设置结束速度小于 0
3006	正向硬件限位触发，不允许正向启动 PMOVE
3007	负向硬件限位触发，不允许负向启动 PMOVE
3008	轴 ALM 信号有效，启动 PMOVE 失败
3009	正向软件限位触发，不允许正向启动 PMOVE
3010	负向软件限位触发，不允许负向启动 PMOVE
3011	轴处于空闲或不处于 PMOVE 模式，强行变位操作失败
3012	在线变速失败，新的目标速度小于 0
3013	设置点位起跳速度大于 4M 频率
3014	设置点位目标速度大于 4M 频率
3015	设置点位结束速度大于 4M 频率
3016	设置软启动结束速度大于 4M 频率
3017	设置软启动目标速度大于 4M 频率
3018	该轴已配置为 PWM 输出轴，PMOVE ENTER 失败，PMOVE 与 PWM 轴互斥
3019	PMOVE 运动已经结束，不能进行在线变位操作
3020	当前处于回零运动模式，不允许该操作
3021	软着陆第一段 PMOVE 结束速度为零，导致第二段 PMOVE 无法规划
3022	软着陆第二段目标位置小于第一段目标位置，导致第二段 PMOVE 反向运动，容易引起危险
3023	轴 EMG 信号有效，启动 PMOVE 失败
3024	当前运动模式不是 VMOVE 和 PMOVE，不支持在线变速操作
3025	当前运动模式不是 PMOVE，不支持同时变速和变位速操作
3026	内部规划缓冲区出错
3027	软着陆第二段方向错误
3028	软着陆中加速度或者减速度为 0
3029	当前轴运动模式为空闲时，调用 VMOVE 和 PMOVE 在线变速报错
3030	当前处于软启动或软着陆运动，PMOVE 在线变速报错
3031	当前处于软启动或软着陆运动，PMOVE 在线变位报错
3032	PMOVE 平滑时间大于 1 秒报错
3033	设置的加速时间等于 0
3034	设置的减速时间等于 0
3035	设置的加速度小于等于 0
3036	设置的减速度小于等于 0
3037	设置软启动第二段目标速度小于第一段目标速度
3038	软着陆设置的停止速度必须小于最大速度
3039	软着陆设置的停止速度必须大于 0
3040	软着陆第一段运动距离为零
3041	软启动第二段目标位置小于或等于第一段目标位置，导致第二段 PMOVE 反向运动，容易引起危险
3042	软启动第一段运动距离为零
3043	设置的加速时间大于 100S
3044	设置的减速时间大于 100S
3045	设置的平滑时间大于加速或减速时间
3046	多轴同步点位运动轴数超范围(2~12)
3047	多轴同步点位运动轴号相同冲突错误
3048	多轴同步点位运动,轴规划总时间为零错误

3049	多轴同步点位运动,设置轴运动参数错误：未找到相应轴号
3050	加/减加速度为零
3100	正向硬件限位触发，不允许正向启动 VMOVE
3101	负向硬件限位触发，不允许负向启动 VMOVE
3102	轴 ALM 信号有效，启动 VMOVE 失败
3103	正向软件限位触发，不允许正向启动 VMOVE
3104	负向软件限位触发，不允许负向启动 VMOVE
3105	轴 EMG 信号有效，启动 VMOVE 失败
3150	正弦波频率限制
3151	正弦波单周期速度限制
3152	正弦波振幅错误
3153	正向硬件限位触发，设置 SINE 曲线参数失败
3154	负向硬件限位触发，设置 SINE 曲线参数失败
3155	轴 ALM 信号有效，设置 SINE 曲线参数失败
3156	正向软件限位触发，设置 SINE 曲线参数失败
3157	负向软件限位触发，设置 SINE 曲线参数失败
3158	轴 EMG 信号有效，设置 SINE 曲线参数失败
3159	轴 DSTP 信号有效，设置 SINE 曲线参数失败
3200	轴数超过最大轴数范围，且该轴运动模式不是手轮模式，不允许操作
3201	该轴运动模式不是手轮模式，设置手轮编码器位置失败
3202	手轮倍率为 0
3203	轴处于手轮运动中，禁止编码器清零
3204	驱动器 ALM 信号有效，手轮启动失败
3205	轴 EMG 信号有效，手轮启动失败
3206	轴未使能
3207	已经处于手轮模式
3208	总线配置状态
3209	轴号超出范围
3210	轴选超出范围
3211	倍选超出范围
3212	设置手轮编码器值为非零值
3300	门型正在运动
3301	轴未使能
3302	轴正在运动
3303	轴号超出范围
3304	插补系超出范围
3305	轴 ALM 信号有效
3306	轴 EMG 信号有效
3307	轴 DSTP 信号有效
3308	正向硬件限位触发
3309	负向硬件限位触发
3310	正向软件限位触发
3311	负向软件限位触发
3312	第二段运动与第一段方向不一致或者第二段距离为 0，启动 M_MOVE 失败

3313	没有开始速度规划
3314	指令缓存正在运动中，不允许相关的操作
3315	超过指令缓存最大限制轴数
3316	指令缓存当前已打开
3317	超过指令缓存组数(2 组)
3318	指令缓存没有打开
3319	压入的点位运动指令的轴数为 0
3320	压入的指令类型不存在
3321	触发条件不存在
3322	触发的 IO 条件超过最大个数（5）限制
3323	触发的轴条件超过最大个数（3）限制
3324	IO 口操作超过最大个数限制
3325	指令轴没有在 OPEN 时使能
3326	指令轴重复输入
3327	指令 BUFFER 已满
3328	指令 BUFFER 异常停止
3329	设置的反馈位置误差参数错误
3330	该模式下不允许删除指令
3331	没有对应得删除的 ID 存在
3332	指令缓存中的压入数据异常
3333	指令缓存输入指令为空指针
3500	单段插补模块已经初始化
3501	单段插补模块内存开辟失败
3502	单段插补模块内坐标系忙，不能清除模块
3503	单段插补模块指针为空，不能清除模块
3504	单段插补模块坐标系状态不为空闲，不能进行参数配置
3505	单段插补模块坐标系号超过最大值
3506	单段插补模块坐标系维度配置错误
3507	单段插补模块速度规划参数配置错误
3508	单段插补模块路径长度为 0
3509	单段插补模块路径类型不支持
3510	单段插补模块圆弧三点在一条直线上
3511	单段插补模块圆弧点坐标相同
3512	单段插补模块圆弧终点+半径模式，半径为 0 或小于两点的距离的一半
3513	单段插补模块圆弧半径为 0
3514	单段插补模块圆弧圈数小于 0
3515	单段插补模块圆弧 3 维空间参数模式错误
3516	单段插补模块圆弧终点半径方式，圆弧参数错误，可能在同一直线上
3517	单段插补模块圆弧计算空间向量函数输入参数指针为 0
3518	单段插补模块圆弧计算空间向量错误
3519	单段插补模块椭圆维度错误
3520	单段插补模块椭圆 A/B 轴长度错误
3521	单段插补模块椭圆起点或终点不在曲线上
3522	单段插补模块椭圆数据坐标相同
3523	单段插补模块坐标系不处于空闲状态，不能启动插补
3524	单段插补模块坐标系不处于完成状态，不能复位坐标系
3525	单段插补模块圆弧圈数超出范围

3526	单段插补交接模式设置超范围
3527	单段插补过渡模式设置超范围
3528	单段插补缓冲区已满
3529	单段插补缓冲区已空
3530	单段插补模块 FIFO 内存开辟失败
3531	单段插补模块 FIFO 已经初始化
3532	指针指向地址为空错误
3600	点位运动库轴数错误
3601	点位运动库缓冲区长度错误
3602	点位运动库插补周期错误
3603	点位运动库内存获取失败
3604	点位运动库库空间释放失败
3605	点位运动库没有初始化
3606	点位运动库运动类型不支持
3607	点位运动库运动缓存模式错误
3608	点位运动库缓冲区满
3609	点位运动库轴对象为空
3610	点位运动库轴处于 DONE 状态，不能添加指令
3611	点位运动库当前指令优先级较低，无法添加
3612	点位运动库处于边界情况，无法添加指令
3613	点位运动库执行 STOP 指令，禁止添加指令
3614	点位运动库轴不处于 DONE 状态，无法 RESET
3615	点位运动库叠加运动 OFF，不能执行动作
3616	PAUSE 状态时不能压入 HALT 指令
3617	点位运动库缓冲区指令缓存模式为目标速度衔接
3702	内存不足
3703	源数据点数量错误
3704	参数向量已经构建
3705	节点向量已经构建
3706	样条构建模式不支持
3707	轨迹模式不支持
3708	已达到最小控制点数量，样条构建失败
3709	超过最大控制点数量限制，样条构建失败
3710	已达到最大迭代次数，样条构建失败
3711	没有开启前瞻模式
3712	PCI 数据传输数量错误
3713	PCI 数据控制点数量错误
3714	PCI 数据节点向量数据错误
3715	PCI 数据样条长度为 0
3716	PCI 数据样条维度错误
3717	缓冲区内存不足

3718	缓冲区链表节点中的样条指针不为空
3719	样条轨迹禁止变倍率
3720	样条轨迹禁止暂停
3721	样条轨迹起点和上一个点不衔接
3722	贝塞尔过渡参数检查，两条曲线没有公共连接点
3723	贝塞尔过渡参数检查，三点共线
3724	贝塞尔过渡参数检查，半径圆弧模式输入信息不正确
3725	贝塞尔过渡参数检查，圆心圆弧模式输入信息不正确
3726	贝塞尔过渡参数检查，不支持的输入圆类型：（可使用类型：三点圆弧，圆心半径）
3727	贝塞尔过渡参数检查,半径圆心模式，没有输入圆弧运动方向
3728	贝塞尔过渡参数检查,半径圆心模式，没有输入圆弧运动平面
3729	贝塞尔过渡插补阶段，反算参数 T 出现错误
3730	贝塞尔计算直线交点出现错误
4000	坐标系插补有限状态不在有限状态机所列状态范围
4001	插补坐标系处于空闲状态时，START_LIST 操作无效
4002	插补坐标系处于空闲状态时，PAUSE_LIST 操作无效
4003	插补坐标系处于就绪状态时，OPEN_LIST 操作无效
4004	插补坐标系处于就绪状态时，PAUSE_LIST 操作无效
4005	插补坐标系处于运动状态时，OPEN_LIST 操作无效
4006	暂停或减速停，停止过程未完成，不允许调用重启运动
4007	插补坐标系处于暂停状态时，OPEN_LIST 操作无效
4008	插补坐标系处于空闲状态，不允许操作（添加指令）
4009	插补坐标系处于运动状态时，不允许操作插补模式
4010	插补坐标系处于运动状态时，不允许操作设置参数
4011	插补坐标系处于运动状态时，不允许重复启动
4012	插补坐标系处于关闭状态时，不允许操作（添加指令）
4013	插补坐标系处于暂停中状态时，不允许暂停
4014	插补坐标系处于单段插补模式，OPEN_LIST 操作无效
4015	插补坐标系处于连续插补中，不能进行单段插补
4016	插补坐标系不处于允许的状态中
4017	插补坐标系处于异常状态状态，操作无效，需先清除插补系异常状态后再操作
5000	设置插补系超出最大插补系允许范围
5001	设置插补目标速度大于 4M 频率
5002	设置螺旋线封闭模式失败（模式取值只能 0 和 1）
5003	插补维数和插补轴数不匹配
5004	坐标系尚未建立，轨迹插补计算失败
5005	插补维数超出范围（大于最大轴数或小于 2）
5006	缓冲区未打开
5007	STARTLIST 失败，由于上一次设置的速度倍率为 0
5008	初始化模块，动态分配前瞻和插补 FIFO 失败
5009	在线变倍率设置的倍率超出范围值（0~2）
5010	脉冲当量小于等于 0

5011	设置的目标速度小于等于 0
5012	设置连续插补段速度比例超限（0~2）
5013	缓存区处于异常状态，不允许操作
5014	缓存区空间已满，不允许操作
5015	插补路径矢量位移为零
5016	圆弧插补三点共线，无法求取圆弧参数
5017	终点+半径圆弧插补模式，起点和终点重合错误
5018	终点+半径圆弧插补模式，半径为 0 或小于两点的距离的一半
5019	增强圆弧插补，半径小于等于 0
5020	增强圆弧插补，旋转角度为零或者大于 360 度
5021	终点半径的圆弧插补方式，圆弧参数错误，可能在同一直线上
5022	添加轨迹轴列表轴号存在相同轴号错误
5023	添加轨迹轴号不在 OPENLIST 轴列表内
5024	增强圆弧插补不支持前瞻模式
5025	半径+终点圆弧插补，圈数为负值暂不支持
5026	圆心+终点圆弧插补，圈数为负值，暂不支持同心圆插补
5027	仅支持 T 形速度规划模式，设置速度插补模式有误，不支持变速
5028	由于变倍率引起的暂停或减速停，停止过程未完成，不允许调用变倍率
5029	上一次变倍率未处理完，不允许再次调用变倍率
5030	当前段处于减速阶段，不允许调用变倍率操作
5031	轨迹暂停后，存在点位运动轴未恢复至初始位置情况报错
5032	坐标系不处于暂停状态，不允许调用暂停返回运动
5033	上一次变倍率当前段变速未处理完，不允许再次调用变倍率
5034	在线变倍率未处理完，不允许调用轨迹暂停命令
5035	运动模式错误，轨迹暂停后恢复初始位置运动失败
5036	设置切向跟随时，被跟随的轨迹模式错误（模式仅支持圆弧和椭圆）
5037	设置切向跟随时，被跟随轴角度当量小于 0
5038	设置切向跟随时，跟随坐标系处于运动中，不允许设置切向跟随
5039	设置切向跟随时，跟随轴处于运动中
5040	解除切向跟随时报错，该轴处于运动中，不允许解除跟随功能
5041	解除切向跟随时，该轴不处于切向跟随模式，解除报错
5042	读取切向跟随参数失败，该轴不处于切向跟随模式或已被解除切向跟随
5043	该版本固件连续轨迹功能模块禁止，不支持连续轨迹功能
5044	运动中或暂停时禁止设置前瞻参数
5045	刀向跟随量设置值为 0
5046	插补坐标系不处于运动状态时，不允许操作在线变倍率
5047	平面圆弧插补前两轴或者空间圆弧插补前三个轴不在主轴范围内
5048	圆弧插补参数错误
5049	当前段尚未规划，不响应在线变倍率请求
5050	插补系运动重启未完成，不响应当次在线变倍率功能
5051	连续轨迹控制节点 IO 编号超出范围：0-本地 IO 1~8-扩展 IO
5052	设置单轴最大插补速度小于等于零错误
5053	设置单轴最大插补速度大于 4M
5054	圆心+角度圆弧插补参数无效错误
5055	直线插补在线变位置时，配置的轴参数不对
5100	速度规划函数出错
5101	圆弧限速速度小于等于 0

5102	设置规划参数错误，起始或结束速度小于 0，或者目标速度小于等于 0，或者加速度小于等于 0
5103	设置的单轴最大速度小于等于 0
5104	设置单段插补位移长度小于或等于 0
5105	设置的加速时间等于 0
5106	设置的减速时间等于 0
5107	置的前瞻加速度等于 0
5108	设置的最大加速度等于 0
5109	设置的平滑停止时间等于 0
5110	设置速度规划模式超限（0-T 形,1-S 形）
5111	设置的平滑时间大于 1 秒
5112	规划起始速度小于 0
5113	规划目标速度小于等于 0
5114	规划结束速度小于 0
5115	当前段剩余周转期不足以支持当前段变速
5116	当前段处于结束或者空闲阶段不支持当前段变速
5117	当前段速度规划模式错误，不支持当前段变速
5118	当前段新目标速度达不到，不支持当前段变速
5119	当前段剩余距离不足，不支持当前段变速
5120	设置的加速度等于 0
5121	设置的减速度等于 0
5122	设置的加速时间超出范围
5123	设置的减速时间超出范围
5124	规划起始速度超出范围
5125	规划目标速度超出范围
5126	规划结束速度超出范围
5127	坐标系暂停过程，插补轴点位运动操作类型超出范围（0~3）
5128	设置的 S 段时间大于加速或减速时间
5129	点胶位置超出当前段总长度，获取规划时间失败
5130	前瞻模式，用户自定义点胶位置超出当前段总长度，超出的点胶分配至下一段插入的圆弧轨迹
5200	设置插补维数超出范围（大于最大轴数）
5201	径+终点圆弧插补模式，半径为零
5202	不支持该圆弧插补模式，模式超限
5203	半径+终点圆弧插补模式，弦长大于直径错误（终点和起点的距离大于直径）
5204	3 维空间圆弧插补圆弧参数模式错误（模式为终点+半径模式）
5205	计算空间圆弧的转换矩阵，输入参数地址无效
5206	计算空间圆弧的转换矩阵，单位向量计算结果有误
5207	椭圆插补，长半轴或端半轴长度小于 0
5208	起点不在椭圆上
5209	终点不在椭圆上
5210	圆弧圆心终点模式，圆心坐标与起点相同
5211	连续轨迹直线段和圆弧自动分段段距小于等于 0
5212	连续轨迹直线段和圆弧自动分段未使能
5213	连续轨迹直线段和圆弧自动分段缓存已满
5214	连续轨迹类型错误，目前仅支持直线段和圆弧自动分段
5215	连续轨迹直线段和圆弧自动分段段数大于缓存剩余空间

5216	自动分段点胶或者用户自定义批量点胶位置，IO 号配置超出允许范围(0~15)
5217	自动分段点胶或者用户自定义批量点胶位置，IO 控制模式超出允许范围(0~4,6)
5218	不支持的插补模式，模式超限
5300	LIST 已经打开
5301	缓冲区未打开，关闭缓冲区操作无效
5302	圆弧插补获取插补平面时，插补轴数小于 2 错误
5303	圆弧插补获取插补平面时，插补轴数与插补轴列表维数不一致
5304	圆弧插补获取插补平面时，插补轴不在插补轴列表内
5305	设置环形缓冲区状态不在有限状态范围以内
5306	设置引起 RINGBUFFER 停止源错误（0-减速停引起，1-暂停引起）
5307	设置环形缓冲区异常状态错误（0-正常，1-减速停，2-急停）
5308	插补系号或者 IO 索引号超出范围
5309	插补轴正向硬件限位触发，OPENLIST 失败
5310	插补轴负向硬件限位触发，OPENLIST 失败
5311	插补轴轴 ALM 信号有效，OPENLIST 失败
5312	插补轴正向软件限位触发，OPENLIST 失败
5313	插补轴负向软件限位触发，OPENLIST 失败
5314	插补轴 EMG 信号有效，OPENLIST 失败
5315	工件坐标系超出范围
5316	旋转加速度或最大速度小于等于 0
5317	读取插补系工件坐标系号和使能报错，未使能或工件坐标系号为空
5318	插补轴正向硬限位有效，无法启动单段圆弧插补
5319	插补轴负向硬限位有效，无法启动单段圆弧插补
5320	插补轴正向软限位有效，无法启动单段圆弧插补
5321	插补轴负向软限位有效，无法启动单段圆弧插补
5322	插补轴正向硬限位有效，无法启动单段直线插补
5323	插补轴负向硬限位有效，无法启动单段直线插补
5324	插补轴正向软限位有效，无法启动单段直线插补
5325	插补轴负向软限位有效，无法启动单段直线插补
5350	工件坐标系超出范围
5351	旋转加速度或最大速度小于等于 0
5352	读取插补系工件坐标系号和使能报错，未使能或工件坐标系号为空
5353	开启五轴功能，OPEN LIST 维度需要为 5
5354	开启五轴功能，不能设置轮廓误差
5355	开启五轴功能，不能设置轮廓误差
5356	错误的五轴机器类型
5357	传递了空参数指针
5358	插补坐标系忙，不能改变功能
5359	插补器下标错误
5360	工件歪斜设置值错误，角度大于等于 360
5361	五轴插补系超出限制范围
5400	轴组编号超出有效值允许的范围
5401	连接组编号超出有效值允许的范围
5402	支持最大连接组数已满，不能再绑定主从关系

5403	主从轴轴号相同，不能再绑定主从关系
5404	多组主从轴组之间轴号冲突，不能再绑定主从关系
5405	未找到与指定主轴绑定的从轴
5406	轴组未使能
5407	轴组已使能或处于运动中
6000	模块初始化错误
6001	不支持
6002	参数错误
6003	缓存空
6004	缓存满
6005	错误索引
6006	缓存段无效
6010	插补系号错误
6011	插补系没有打开
6012	插补系已经打开
6013	插补系已经关闭
6014	插补系不能暂停
6015	插补系正在运行
6016	插补系正在暂停
6017	插补器处于暂停状态
6018	LIST 暂停后移动轴未归位,不能重启
6019	LIST 已经启动
6020	插补 IO 缓冲区已满
6021	插补段已被中断使用
6022	该模式下不能设置该功能
6023	运动中不运行更改参数
6024	参数不在有效范围
6025	主要的 LIST 没有初始化
6026	另外一个 LIST 已经启动
6027	主要 LIST 没有启动
6028	强制变位段号不匹配
6030	插补维数超限
6031	轴数不在有效范围
6032	插补轴号一样的
6033	轴不在 OPEN 时的列表中
6034	轴映射表为空
6035	映射轴错误
6036	映射轴忙
6037	轴的使能信号关闭，不能启动
6038	轴软件限位信号有效
6039	轴硬件限位信号有效
6040	轴减速停止信号有效
6041	轴急停信号有效
6042	轴报警信号有效
6043	轴已经被插补系占用

6044	轴可能被移位且未回位
6045	轴没有用 PMOVE 移动过，不支持回位操作
6050	半径为 0 或小于两点的距离的一半
6051	半径方式起点和终点重合
6052	圆弧轴不在 XYZ 里面
6053	圆弧三点在一条直线上
6054	圆弧圈数小于 0
6055	圆弧圈数超出范围
6056	样条曲线的起点位置错误，不与实际启动时刻位置一致
6060	S 曲线加减速模式，平滑时间为零出错
6061	起跳速度小于零
6062	最大速度小于等于零
6063	终点速度小于零
6064	规划长度小于等于零
6065	最小匀速时间小于零
6066	规划模式非 T 型非 S 型
6067	加速度小于等于零
6068	减速度小于等于零
6069	规划长度小于等于零
6070	参数变量空指针
6071	参数变量 2 空指针
7002	奇异点，无法到达
7003	输入建模参数报错
7004	输入运动范围无法到达
7010	通用参数错误
7011	机械臂未配置
7012	机械臂用户坐标系未配置
7013	机械臂工具坐标系未配置
7014	用户坐标系建立失败
7015	工具坐标系建立失败
7016	机械被使能关闭
7017	机械臂类型错误
7018	机械臂序号错误
7019	机械臂用户坐标系错误
7020	机械臂工具坐标系错误
7021	臂型错误
7022	关节数量错误
7023	关节映射错误
7024	结构参数错误
7025	超出工作空间
7026	关节限位触发
7027	关节限速触发

7028	关节限加速度触发
7029	关节限速触发
7030	关节限加速度触发
7031	机械臂处于奇异点
7032	雅克比行列式值为 0

附录 2　ACC-X400B 接线盒接口说明

ACC-X400B 接线盒是 DMC5410A 运动控制卡的配套产品，扩展 DMC5410A 控制卡的所有用户端子。

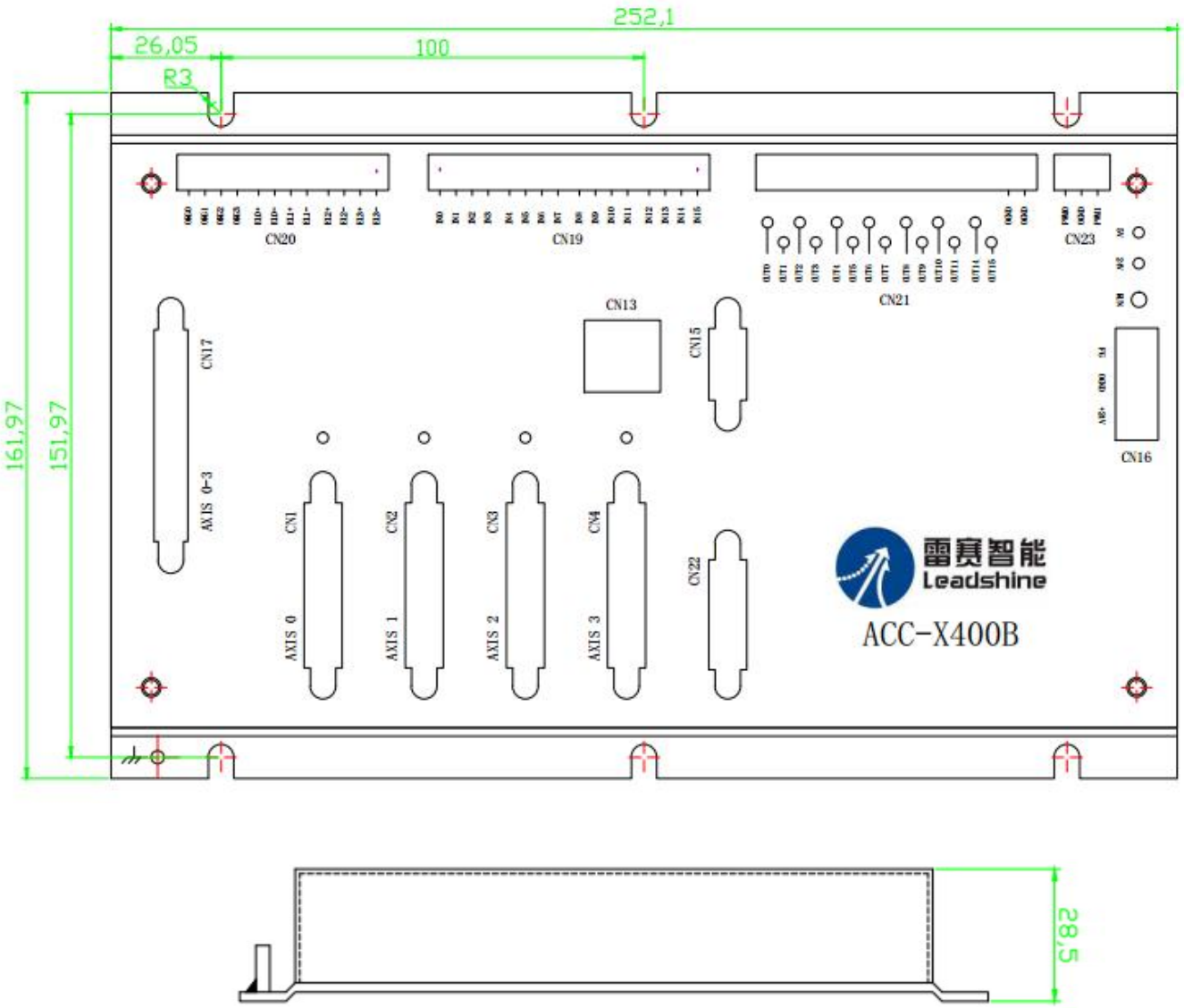


图 2.1　ACC-X400B 接线盒尺寸图

ACC-X400B 接线盒外形结构图如图 2.2 所示：

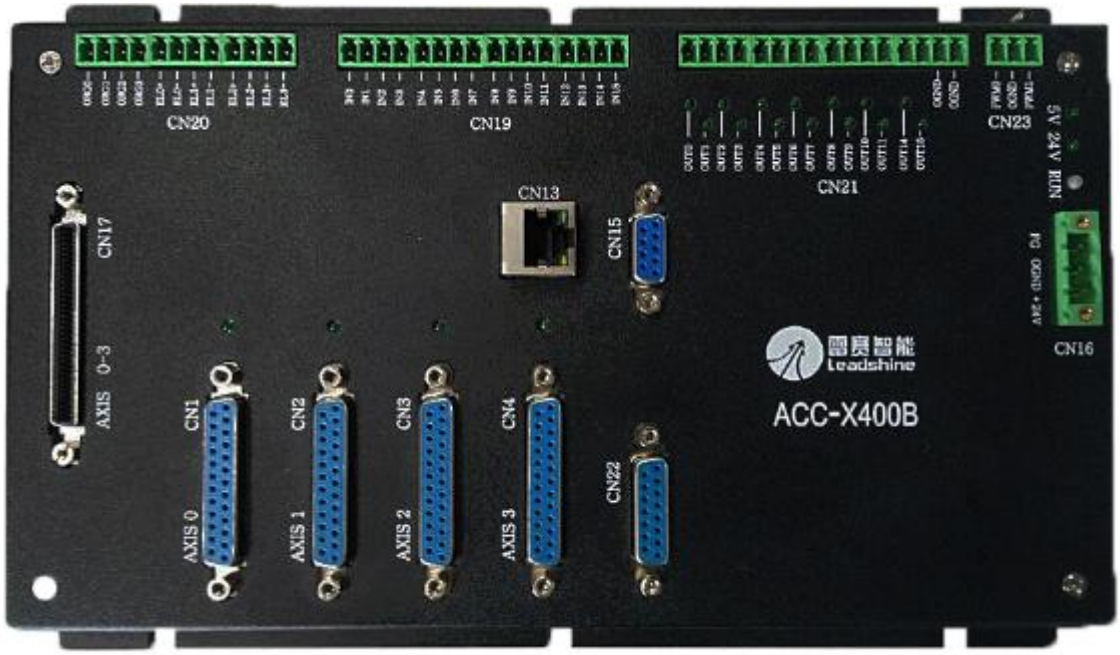


图 2.2　ACC-X400B 接线盒外形结构图

1、 ACC-X400B 接线盒接口及脚位

表 2.1　ACC-X400B 接线盒接口功能简述

名称	功能介绍
CN1~CN4	第 1~4 轴轴控制信号
CN13	CAN 总线接口（RJ45）
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端子（1~4 轴）
CN19	数字量通用输入端子
CN20	数字量专用输入端子
CN21	数字量输出端子
CN23	PWM 输出端子

2、 CN1~CN4 轴控制端子信号定义

1）ACC-X400B 接线盒 CN1 至 CN4 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 2.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN4）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC-X400B 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 2.2　接口 CN1~CN8 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACCX400B 接线盒编码器口 1~2 轴仅支持差分接法，3~4 轴同时支持单端和差分接法

3、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

4、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 2.3 所示。

表 2.3　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出

6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

5、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 OGND 的端子接外部电源地,标有 FG 的端子为机壳地接口。

6、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

7、 CN19 数字量输入端子定义

CN19 为数字量输入接口，其引脚号和信号对应关系见表 2.4 所示：

表 2.4　接口 CN9 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	IN0	I	通用输入（低速）	9	IN8	I	通用输入（低速）
2	IN1	I	通用输入（低速）	10	IN9	I	通用输入（低速）
3	IN2	I	通用输入（低速）	11	IN10	I	通用输入（低速）
4	IN3	I	通用输入（低速）	12	IN11	I	通用输入（低速）
5	IN4	I	通用输入（低速）	13	IN12	I	通用输入（低速）
6	IN5	I	通用输入（低速）	14	IN13	I	通用输入（低速）
7	IN6	I	通用输入（低速）	15	IN14	I	通用输入/LTC0（高速）
8	IN7	I	通用输入（低速）	16	IN15	I	通用输入/LTC1（高速）

8、 CN20 数字量输入接口定义

CN20 为数字量输入接口，其引脚号和信号对应关系见表 2.5 所示。

表 2.5　接口 CN20 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	ORG0	I	0 轴原点输入	7	EL1+	I	1 轴正限位
2	ORG1	I	1 轴原点输入	8	EL1-	I	1 轴负限位
3	ORG2	I	2 轴原点输入	9	EL2+	I	2 轴正限位
4	ORG3	I	3 轴原点输入	10	EL2-	I	2 轴负限位
5	EL0+	I	0 轴正限位	11	EL3+	I	3 轴正限位
6	EL0-	I	0 轴负限位	12	EL3-	I	3 轴负限位

9、 CN21 数字量输出接口定义

CN21 为数字量输出接口，其引脚号和信号对应关系见表 2.6 所示。

表 2.6　接口 CN21 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OUT0	O	通用输出（低速）	9	OUT8	O	通用输出（低速）
2	OUT1	O	通用输出（低速）	10	OUT9	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	11	OUT10	O	通用输出（低速）
4	OUT3	O	通用输出（低速）	12	OUT11	O	通用输出（低速）
5	OUT4	O	通用输出（低速）	13	OUT14	O	通用输出/CMP2（高速）
6	OUT5	O	通用输出（低速）	14	OUT15	O	通用输出/CMP3（高速）
7	OUT6	O	通用输出（低速）	15	OGND	O	外部电源地
8	OUT7	O	通用输出（低速）	16	OGND	O	外部电源地

10、CN22 模拟量输入、输出接口定义（选配）

CN22 为模拟量输入、输出接口，其引脚号和信号对应关系见表 2.7 所示。

表 2.7　接口 CN22 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	AIN0	I	模拟量输入	9	AOUT0	O	模拟量输出
2	AIN1	I	模拟量输入	10	AOUT1	O	模拟量输出
3	AIN2	I	模拟量输入	11	GND	O	内部电源地
4	AIN3	I	模拟量输入	12	GND	O	内部电源地
5	AIN4	I	模拟量输入	13	GND	O	内部电源地
6	AIN5	I	模拟量输入	14	GND	O	内部电源地
7	AIN6	I	模拟量输入	15	GND	O	内部电源地
8	AIN7	I	模拟量输入	16			

说明： 1）支持 8 路模拟量输入和 2 路模拟量输出；

2）AIN0~AIN7 输入电压为-10V~＋10V，12bit 精度；

3）ADC 输入阻抗不小于 100K 欧姆；

4）待测信号的地与接线盒模拟输入端子的 11~15 脚（GND）连接，被采样信号接 AIN0~AIN7；

5）DAC 输出电压范围-10V~＋10V，12bit 精度，默认输出电压 0V。

11、CN23 输出接口定义

CN23 为 PWM 输出接口，其引脚号和信号对应关系如表 F3.8 所示

F3.8　接口 CN23 引脚号和信号关系表

序	名称	I/O	说　明
1	PWM0	O	PWM 输出 0
2	OGND	O	24V 电源地
3	PWM1	O	PWM 输出 1

12、 指示灯定义

ACCX400B 接线盒表面有 3 个指示灯，分别为：

24VLED：外部电源指示灯；

5VLED：内部电源指示灯；

RUNLED：连接状态指示灯。绿色时表示接线盒与控制卡处于通讯状态；红色闪烁时表示接线盒与控制卡未连接成功。

附录 3　ACC3600 接线盒接口说明

ACC3600 接线盒是 DMC5610 运动控制卡的配套产品，扩展 DMC5610 卡的所有用户端口。ACC3600 接线盒外观如图 3.1 所示，接口示意图如图 3.2 所示，尺寸图如图 3.3 所示。

图 3.1　ACC3600 接线盒外观照片

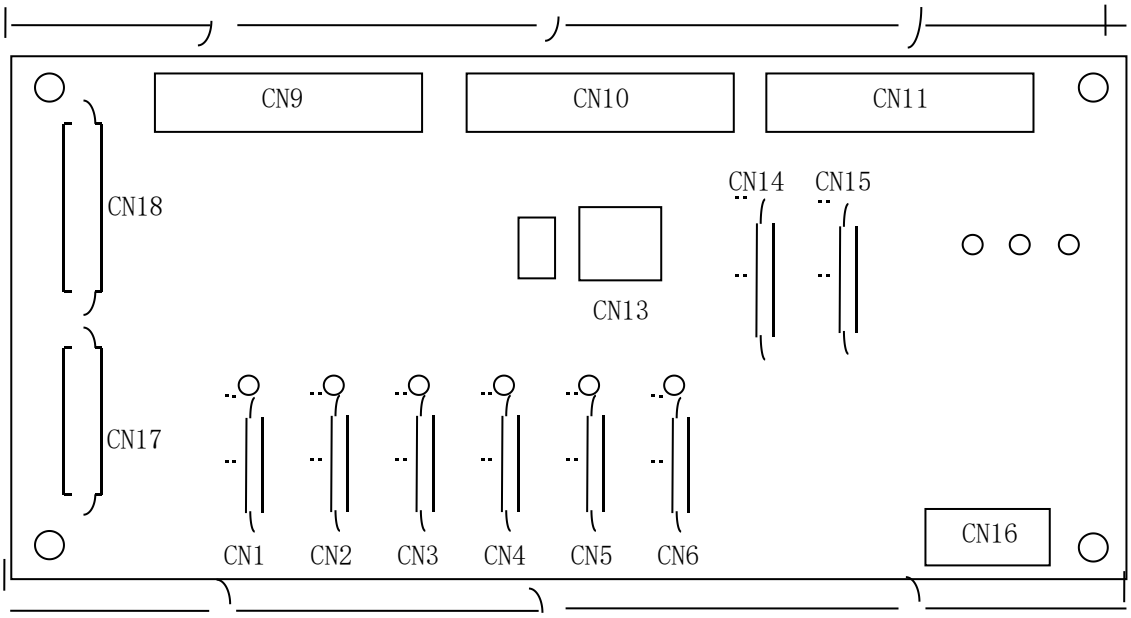


图 3.2　ACC3600 接线盒接口布置示意图

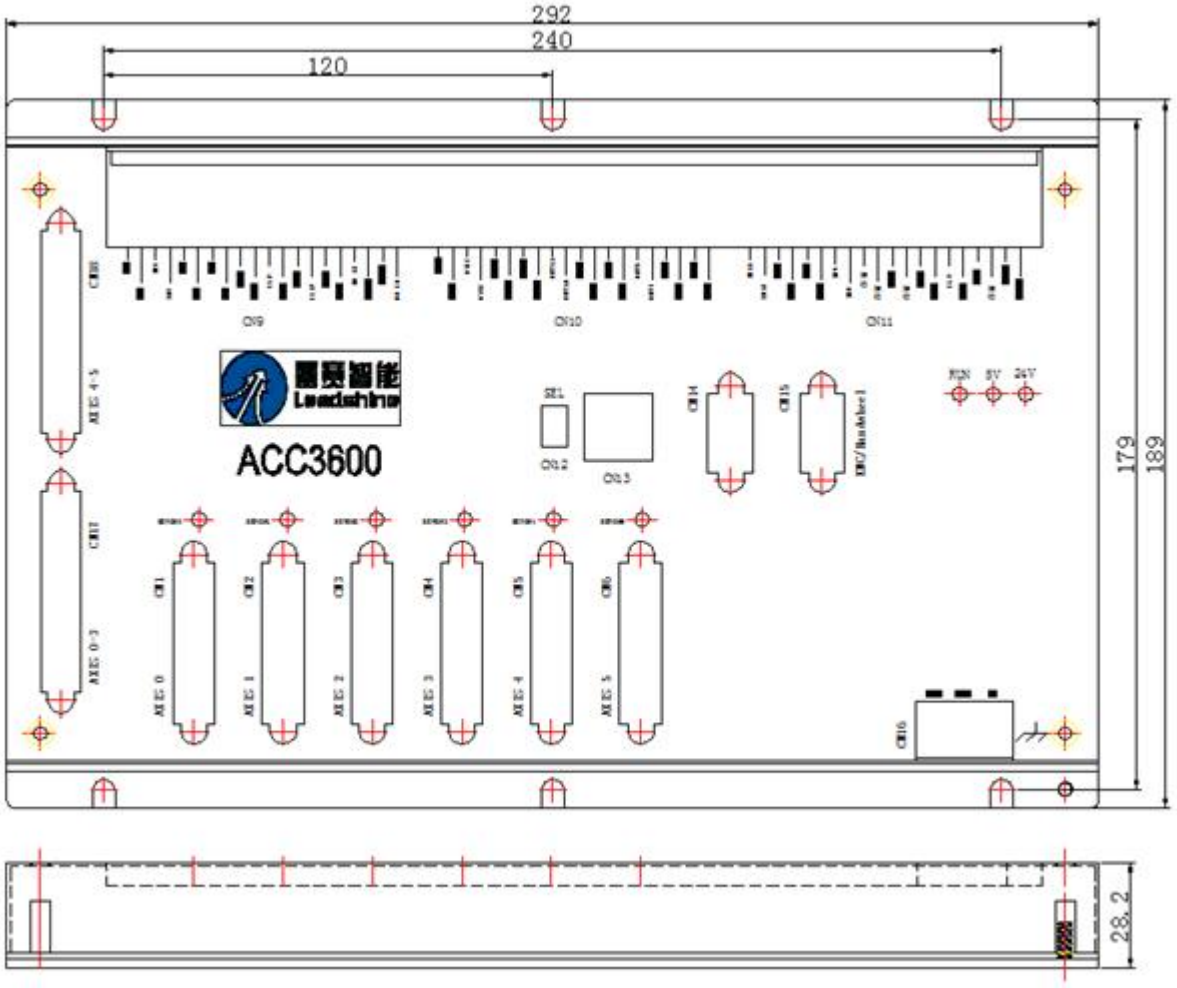


图 3.3　　ACC3600 接线盒尺寸图

表 3.1 ACC3600 接线盒接口功能简述

名称	功能介绍
CN1~CN6	第 0~5 轴轴控制信号
CN9	数字量输入端口
CN10	数字量输出端口
CN11	数字量输入端口
CN12	通用 IO 口初始电平选择开关
CN13	IO 扩展接口
CN14	保留
CN15	辅助编码器接口
CN16	24V 直流电源输入端口
CN17	运动控制卡连接端口（0~3 轴）
CN18	运动控制卡连接端口（4~5 轴）

1、 CN1~CN6 轴控制端口信号定义及使用说明

1）ACC3600 接线盒 CN1 至 CN6 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 3.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN6）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC3600 的 PCB 板的铜线宽度与厚度，6 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 3.2 接口 CN1~CN6 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC3600 接线盒编码器口 1~4 轴仅支持差分接法，5~6 轴同时支持单端和差分接法

2、 CN9 数字量输入端口定义

CN9 为数字量输入接口，其引脚号和信号对应关系见表 3.3 所示。

表 3.3 接口 CN9 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	HOME0	I	0 轴原点输入	11	EL3+	I	3 轴正限位
2	HOME1	I	1 轴原点输入	12	EL3-	I	3 轴负限位
3	HOME2	I	2 轴原点输入	13	IN0	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
4	HOME3	I	3 轴原点输入	14	IN1	I	通用输入（低速）
5	EL0+	I	0 轴正限位	15	IN2	I	通用输入（低速）
6	EL0-	I	0 轴负限位	16	IN3	I	通用输入（低速）
7	EL1+	I	1 轴正限位	17	IN4	I	通用输入（低速）
8	EL1-	I	1 轴负限位	18	IN5	I	通用输入（低速）
9	EL2+	I	2 轴正限位	19	IN6	I	通用输入（低速）
10	EL2-	I	2 轴负限位	20	IN7	I	通用输入（低速）

3、 CN10 数字量输出接口定义

CN10 为数字量输出接口，其引脚号和信号对应关系见表 3.4 所示。

表 3.4　接口 CN10 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OUT0	O	通用输出（低速）	11	OUT10	O	通用输出（低速）
2	OUT1	O	通用输出（低速）	12	OUT11	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	13	OUT12	O	通用输出/CMP0（高速）
4	OUT3	O	通用输出（低速）	14	OUT13	O	通用输出/CMP1（高速）
5	OUT4	O	通用输出（低速）	15	OUT14	O	通用输出/CMP2（高速）
6	OUT5	O	通用输出（低速）	16	OUT15	O	通用输出/CMP3（高速）
7	OUT6	O	通用输出（低速）	17	OVCC	O	+24V 输出
8	OUT7	O	通用输出（低速）	18	OVCC	O	+24V 输出
9	OUT8	O	通用输出（低速）	19	OGND	O	外部电源地
10	OUT9	O	通用输出（低速）	20	OGND	O	外部电源地

注意：当 CMP0~3 端口不设置为高速位置比较器输出端口时，可以设置为 OUT12~15 通用输出端口。

4、 CN11 数字量输入接口定义

CN11 为数字量输入接口，其引脚号和信号对应关系见表 3.5 所示。

表 3.5　接口 CN11 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	HOME4	I	4 轴原点输入	11	EL7+	I	保留
2	HOME5	I	5 轴原点输入	12	EL7-	I	保留
3	HOME6	I	保留	13	IN8	I	通用输入（低速）
4	HOME7	I	保留	14	IN9	I	通用输入（低速）
5	EL4+	I	4 轴正限位	15	IN10	I	通用输入（低速）
6	EL4-	I	4 轴负限位	16	IN11	I	通用输入（低速）
7	EL5+	I	5 轴正限位	17	IN12	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
8	EL5-	I	5 轴负限位	18	IN13	I	通用输入（低速）
9	EL6+	I	保留	19	IN14	I	通用输入/LTC0（高速）
10	EL6-	I	保留	20	IN15	I	通用输入/LTC1（高速）

5、 CN12 拔码开关定义（保留，固定为 ON）

6、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

7、 CN14 接口定义（保留）

8、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 3.6 所示。

表 3.6　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

9、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 OGND 的端子接外部电源地。

10、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

11、 CN18 接口定义

CN18 接口为 4~5 轴电机的控制信号及高速手轮通道与控制卡的接口。

12、 指示灯定义

ACC3600 模块表面有 3 个指示灯，分别为：

24VLED：外部电源指示灯；

5VLED：内部电源指示灯；

RUNLED：连接状态指示灯。绿色时表示接线盒与控制卡处于通讯状态；红色闪烁时表示接线盒与控制卡未连接成功。

附录 4　ACC3800 接线盒接口说明

ACC3800 接线盒是 DMC5810 卡的配套产品，扩展 DMC5810 卡的所有用户端口。ACC3800 接线盒外观如图 4.1 所示，接口示意图如图 4.2 所示，尺寸图如图 4.3 所示。

图 4.1　ACC3800 接线盒外观照片

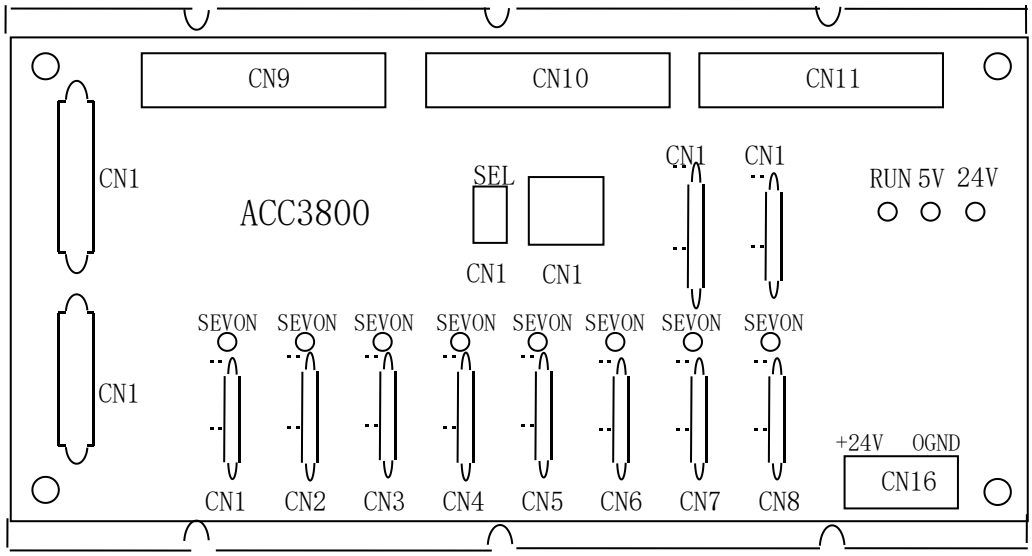


图 4.2　ACC3800 接线盒接口布置示意图

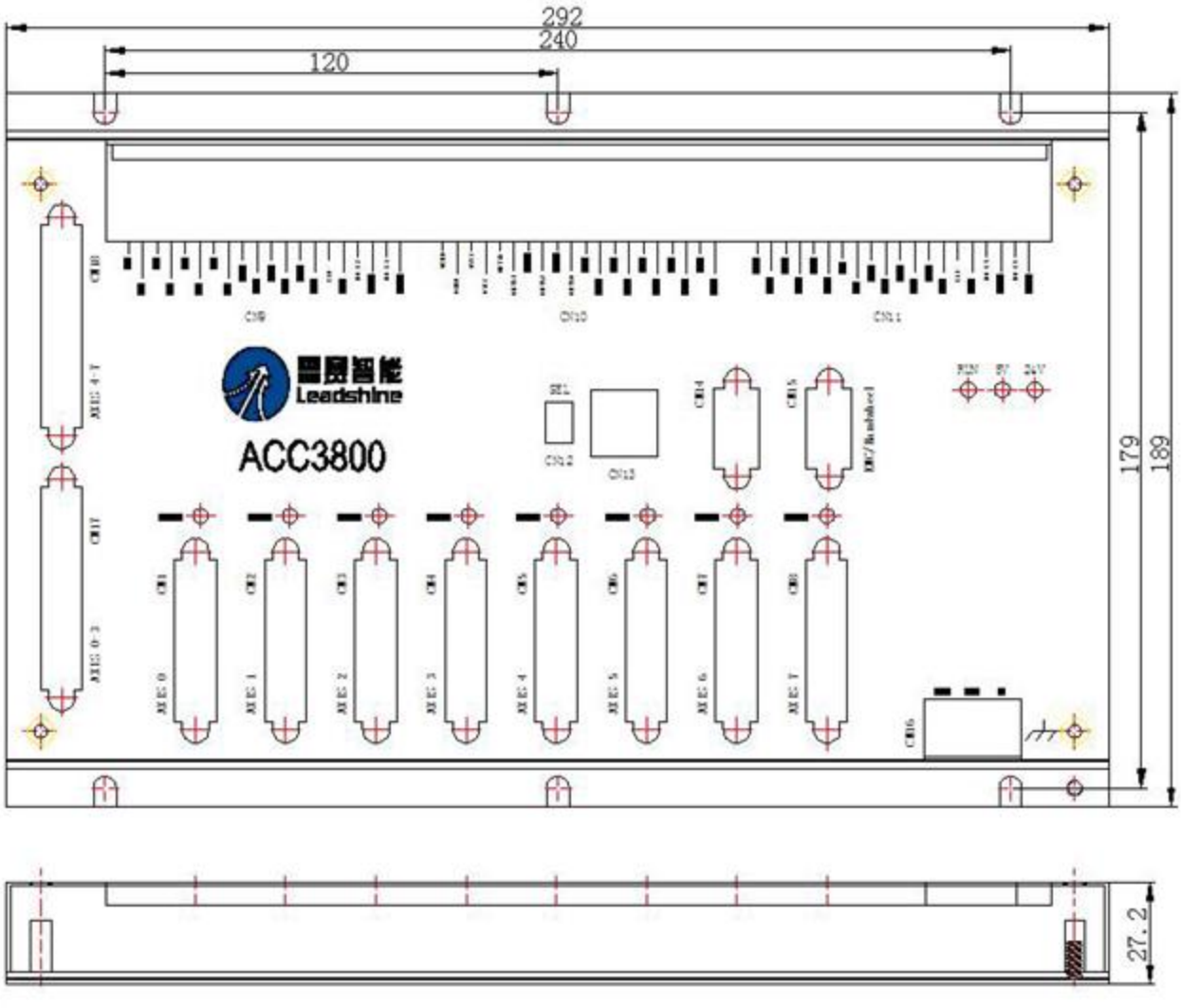


图 4.3　　ACC3800 接线盒尺寸图

表 4.1　ACC3800 接线盒接口功能简述

名称	功能介绍
CN1~CN8	第 0~7 轴轴控制信号
CN9	数字量输入端口
CN10	数字量输出端口
CN11	数字量输入端口
CN12	通用 IO 口初始电平选择开关
CN13	IO 扩展接口
CN14	保留
CN15	辅助编码器接口
CN16	24V 直流电源输入端口
CN17	运动控制卡连接端口（0~3 轴）
CN18	运动控制卡连接端口（4~7 轴）

1、 CN1~CN8 轴控制端口信号定义

1）ACC3800 接线盒 CN1 至 CN8 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 4.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN8）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC3800 的 PCB 板的铜线宽度与厚度，,8 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 4.2 接口 CN1~CN8 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC3800 接线盒编码器口 1~6 轴仅支持差分接法，7~8 轴同时支持单端和差分接法

2、 CN9 数字量输入端口定义

CN9 为数字量输入接口，其引脚号和信号对应关系见表 4.3 所示。

表 4.3 接口 CN9 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	HOME0	I	0 轴原点输入	11	EL3+	I	3 轴正限位
2	HOME1	I	1 轴原点输入	12	EL3-	I	3 轴负限位
3	HOME2	I	2 轴原点输入	13	IN0	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
4	HOME3	I	3 轴原点输入	14	IN1	I	通用输入（低速）
5	EL0+	I	0 轴正限位	15	IN2	I	通用输入（低速）
6	EL0-	I	0 轴负限位	16	IN3	I	通用输入（低速）
7	EL1+	I	1 轴正限位	17	IN4	I	通用输入（低速）
8	EL1-	I	1 轴负限位	18	IN5	I	通用输入（低速）
9	EL2+	I	2 轴正限位	19	IN6	I	通用输入（低速）
10	EL2-	I	2 轴负限位	20	IN7	I	通用输入（低速）

3、 CN10 数字量输出接口定义

CN10 为数字量输出接口，其引脚号和信号对应关系见表 4.4 所示。

表 4.4　接口 CN10 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OUT0	O	通用输出（低速）	11	OUT10	O	通用输出（低速）
2	OUT1	O	通用输出（低速）	12	OUT11	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	13	OUT12	O	通用输出/CMP0（普通）
4	OUT3	O	通用输出（低速）	14	OUT13	O	通用输出/CMP1（普通）
5	OUT4	O	通用输出（低速）	15	OUT14	O	通用输出/CMP2（高速）
6	OUT5	O	通用输出（低速）	16	OUT15	O	通用输出/CMP3（高速）
7	OUT6	O	通用输出（低速）	17	OVCC	O	+24V 输出
8	OUT7	O	通用输出（低速）	18	OVCC	O	+24V 输出
9	OUT8	O	通用输出（低速）	19	OGND	O	外部电源地
10	OUT9	O	通用输出（低速）	20	OGND	O	外部电源地

注意：当 CMP0~3 端口不设置为高速位置比较器输出端口时，可以设置为 OUT12~15 通用输出端口。

4、 CN11 数字量输入接口定义

CN11 为数字量输入接口，其引脚号和信号对应关系见表 4.5 所示。

表 4.5　接口 CN11 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	HOME4	I	4 轴原点输入	11	EL7+	I	7 轴正限位
2	HOME5	I	5 轴原点输入	12	EL7-	I	7 轴负限位
3	HOME6	I	6 轴原点输入	13	IN8	I	通用输入（低速）
4	HOME7	I	7 轴原点输入	14	IN9	I	通用输入（低速）
5	EL4+	I	4 轴正限位	15	IN10	I	通用输入（低速）
6	EL4-	I	4 轴负限位	16	IN11	I	通用输入（低速）
7	EL5+	I	5 轴正限位	17	IN12	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
8	EL5-	I	5 轴负限位	18	IN13	I	通用输入（低速）
9	EL6+	I	6 轴正限位	19	IN14	I	通用输入/LTC0（高速）
10	EL6-	I	6 轴负限位	20	IN15	I	通用输入/LTC1（高速）

5、 CN12 拔码开关定义（保留，固定为 ON）

6、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

7、 CN14 接口定义（保留）

8、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 4.6 所示。

表 4.6　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

9、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 OGND 的端子接外部电源地。

10、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

11、 CN18 接口定义

CN18 接口为 4~7 轴电机的控制信号及高速手轮通道与控制卡的接口。

12、 指示灯定义

ACC3800 模块表面有 3 个指示灯，分别为：

24VLED：外部电源指示灯；

5VLED：内部电源指示灯；

RUNLED：连接状态指示灯。绿色时表示接线盒与控制卡处于通讯状态；红色闪烁时表示接线盒与控制卡未连接成功。

附录 5　ACC-XC00 接线盒接口说明

ACC-XC00 接线盒是 DMC5C10 运动控制卡的配套产品，扩展 DMC5C10 控制卡的所有用户端子。

ACC-XC00 接线盒尺寸图如图 5.1 所示。

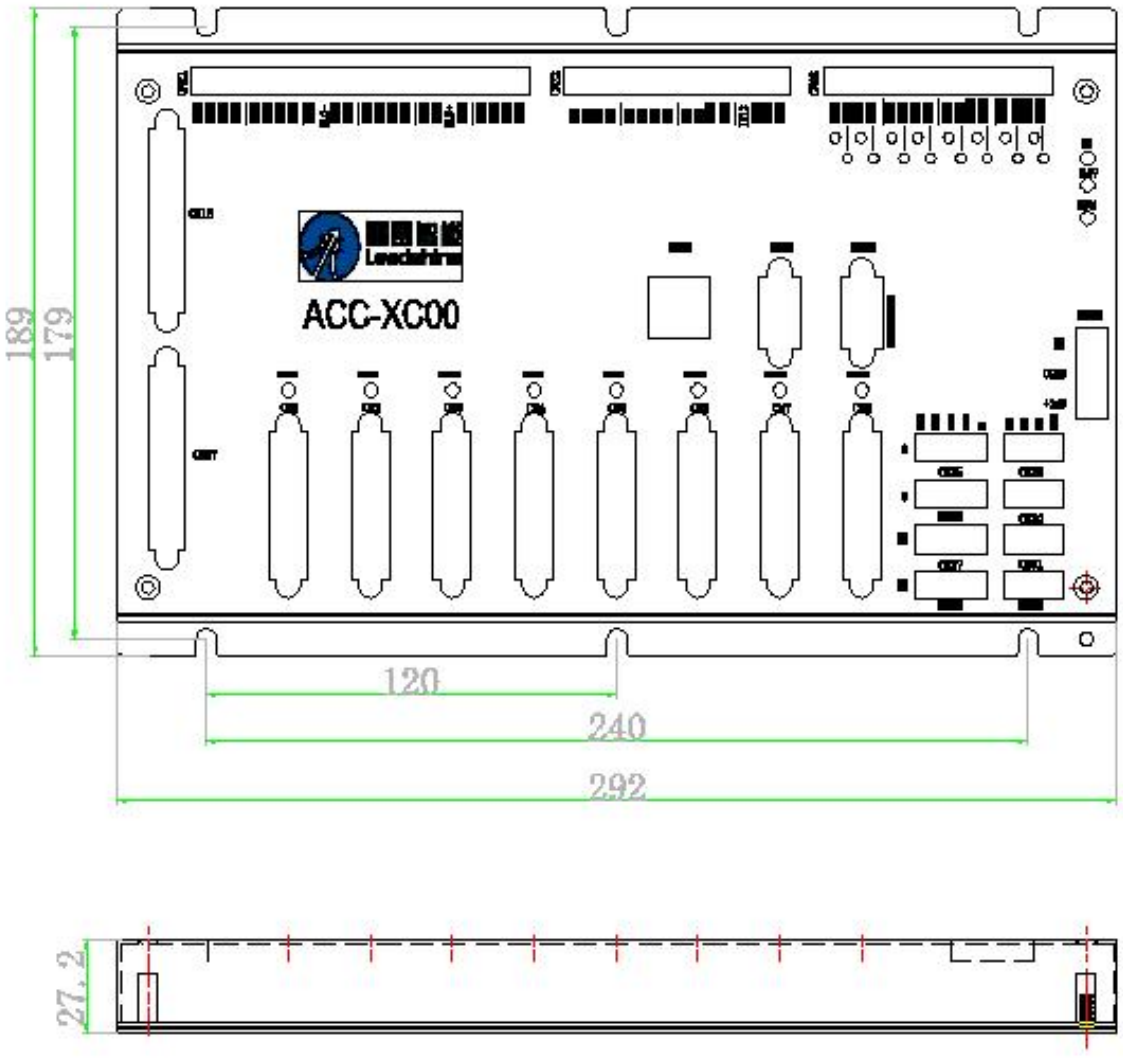


图 5.1　ACC-XC00 尺寸图

ACC-XC00 接线盒外形结构图如图 5.2 所示：

图 5.2　ACC-XC00 接线盒外形结构图

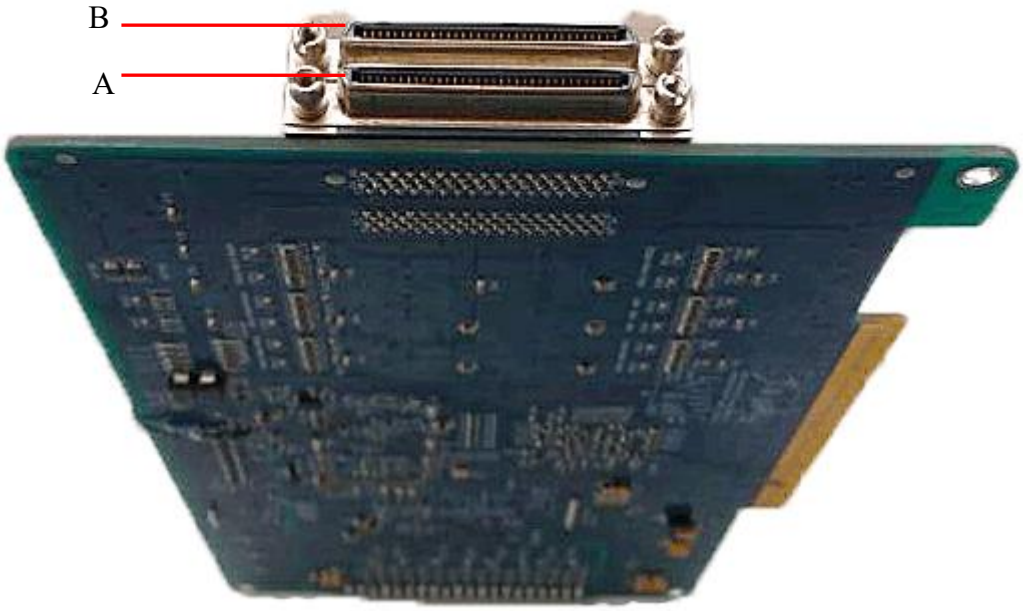


图 5.3　DMC5C10 侧视图

注　意：1）如图 5.3 所示，A为 0~3 轴的电机控制信号以及IO等信号端口，通过电缆线与ACC-XC00 的CN17 端口相连；B为 4~11 轴电机的控制信号以及IO等信号端口，通过电缆线与ACC-XC00 的CN18 端口相连。

2）DMC5C10 与 ACC-XC00 接线盒的连接示意图请参考第 3.2 节

1、 ACC-XC00 接线盒接口及脚位

表 5.1 ACC-XC00 接线盒接口功能简述

名称	功能介绍
CN1~CN8	第 1~8 轴轴控制信号
CN13	CAN 总线接口（RJ45）
CN14	保留
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端子（1~4 轴）
CN18	运动控制卡连接端子（5~12 轴）
CN21~CN23	第 1~8 轴原点、限位以及通用 IO 信号接口
CN25~28	第 9~12 轴轴控制信号
CN29~32	第 9~12 轴控制信号原点、限位信号接口

2、 CN1~CN8 轴控制端子信号定义

1）ACC-XC00 接线盒 CN1 至 CN8 以及 CN25 至 CN28 为电机控制信号端口，CN1 至 CN8 采用 DB25 母头，其引脚号和信号对应关系见表 F5.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN8）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC-XC00 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 5.2 接口 CN1~CN8 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出

10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC-XC00 接线盒 1~6 轴编码器口仅支持差分接法，7~8 轴同时支持单端和差分接法

3、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

4、 CN14 接口定义（保留）

5、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 5.3 所示。

表 5.3　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

6、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 OGND 的端子接外部电源地，标有 FG 的端子为机壳地接口。

表 5.4　接口 CN16 引脚号和信号关系表

序号	名称	功能
1	+24V	外部 24V 电源输入

2	OGND	外部 24V 电源地
3	FG	机壳地

7、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

8、 CN18 接口定义

CN18 接口为 4~11 轴电机的控制信号及高速手轮通道与控制卡的接口。

9、 CN21 专用输入接口定义（0~7 轴）

CN21 为数字量专用输入接口，其引脚号和信号对应关系见表 5.5 所示：

表 5.5　接口 CN21 引脚号和信号关系

序	名称	I/O	说明	序	名称	I/O	说明
1	ORG0	I	0 轴原点输入	13	EL2+	I	2 轴正限位输入
2	ORG1	I	1 轴原点输入	14	EL2-	I	2 轴负限位输入
3	ORG2	I	2 轴原点输入	15	EL3+	I	3 轴正限位输入
4	ORG3	I	3 轴原点输入	16	EL3-	I	3 轴负限位输入
5	ORG4	I	4 轴原点输入	17	EL4+	I	4 轴正限位输入
6	ORG5	I	5 轴原点输入	18	EL4-	I	4 轴负限位输入
7	ORG6	I	6 轴原点输入	19	EL5+	I	5 轴正限位输入
8	ORG7	I	7 轴原点输入	20	EL5-	I	5 轴负限位输入
9	EL0+	I	0 轴正限位输入	21	EL6+	I	6 轴正限位输入
10	EL0-	I	0 轴负限位输入	22	EL6-	I	6 轴负限位输入
11	EL1+	I	1 轴正限位输入	23	EL7+	I	7 轴正限位输入
12	EL1-	I	1 轴负限位输入	24	EL7-	I	7 轴负限位输入

10、CN22 通用输入接口定义

CN22 为数字量通用输入接口，其引脚号和信号对应关系见表 5.6 所示：

表 5.6　接口 CN21 引脚号和信号关系

序	名称	I/O	说　明	序	名称	I/O	说　明
1	IN0	I	通用输入（低速）	9	IN8	I	通用输入（低速）

2	IN1	I	通用输入（低速）	10	IN9	I	通用输入（低速）
3	IN2	I	通用输入（低速）	11	IN10	I	通用输入（低速）
4	IN3	I	通用输入（低速）	12	IN11	I	通用输入（低速）
5	IN4	I	通用输入（低速）	13	IN12	I	通用输入（低速）
6	IN5	I	通用输入（低速）	14	IN13	I	通用输入（低速）
7	IN6	I	通用输入（低速）	15	IN14	I	通用输入/LTC0（高速）
8	IN7	I	通用输入（低速）	16	IN15	I	通用输入/LTC1（高速）

11、CN23 数字量输出接口定义

CN23 为数字量输出接口，其引脚号和信号对应关系见表 5.7 所示。

表 5.7　接口 CN21 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OUT0	O	通用输出（低速）	9	OUT8	O	通用输出（低速）
2	OUT1	O	通用输出（低速）	10	OUT9	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	11	OUT10	O	通用输出（低速）
4	OUT3	O	通用输出（低速）	12	OUT11	O	通用输出（低速）
5	OUT4	O	通用输出（低速）	13	OUT12	O	通用输出/CMP0（高速）
6	OUT5	O	通用输出（低速）	14	OUT13	O	通用输出/CMP1（高速）
7	OUT6	O	通用输出（低速）	15	OUT14	O	通用输出/CMP2（高速）
8	OUT7	O	通用输出（低速）	16	OUT15	O	通用输出/CMP3（高速）

12、CN25 ~ CN28 电机接口端子定义

CN25~CN28 为 8~11 轴电机控制信号端口，其引脚号和信号对应关系见表 5.8 所示。

表 5.8　接口 CN25~CN28 引脚号和信号关系表

序号	名称	说明
1	DIR+	方向输出
2	DIR-	方向输出
3	PUL+	脉冲输出
4	PUL-	脉冲输出
5	5V	内部 5V 输出

说明：CN25 对应第 8 轴；CN26 对应第 9 轴；CN27 对应第 10 轴；CN28 对应第 11 轴

13、CN29 ~ CN32 专用输入接口定义（8~11 轴)

CN29~CN32 为 8~11 轴专用输入信号接口，其引脚号和信号对应关系见表 5.9 所示。

表 5.9　接口 CN25~CN28 引脚号和信号关系表

序号	名称	说明
1	ORG	原点输入
2	EL+	正限位输入
3	EL-	负限位输入
4	OGND	24V 电源地

说明：CN29 对应第 8 轴；CN30 对应第 9 轴；CN31 对应第 10 轴；CN32 对应第 11 轴

14、 指示灯定义

ACC-XC00 模块表面有 3 个指示灯，分别为：

24VLED：外部电源指示灯；

5VLED：内部电源指示灯；

RUNLED：连接状态指示灯。绿色时表示接线盒与控制卡处于通讯状态；红色闪烁时表示接线盒与控制卡未连接成功。

附录 6　ACC2-X400B 接线盒接口说明

ACC2-X400B 接线盒是 DMC5410A-PCIe 运动控制卡的配套产品，扩展 DMC5410A-PCIe 控制卡的所有用户端子。

ACC2-X400B 接线盒尺寸图如图 6.1 所示。

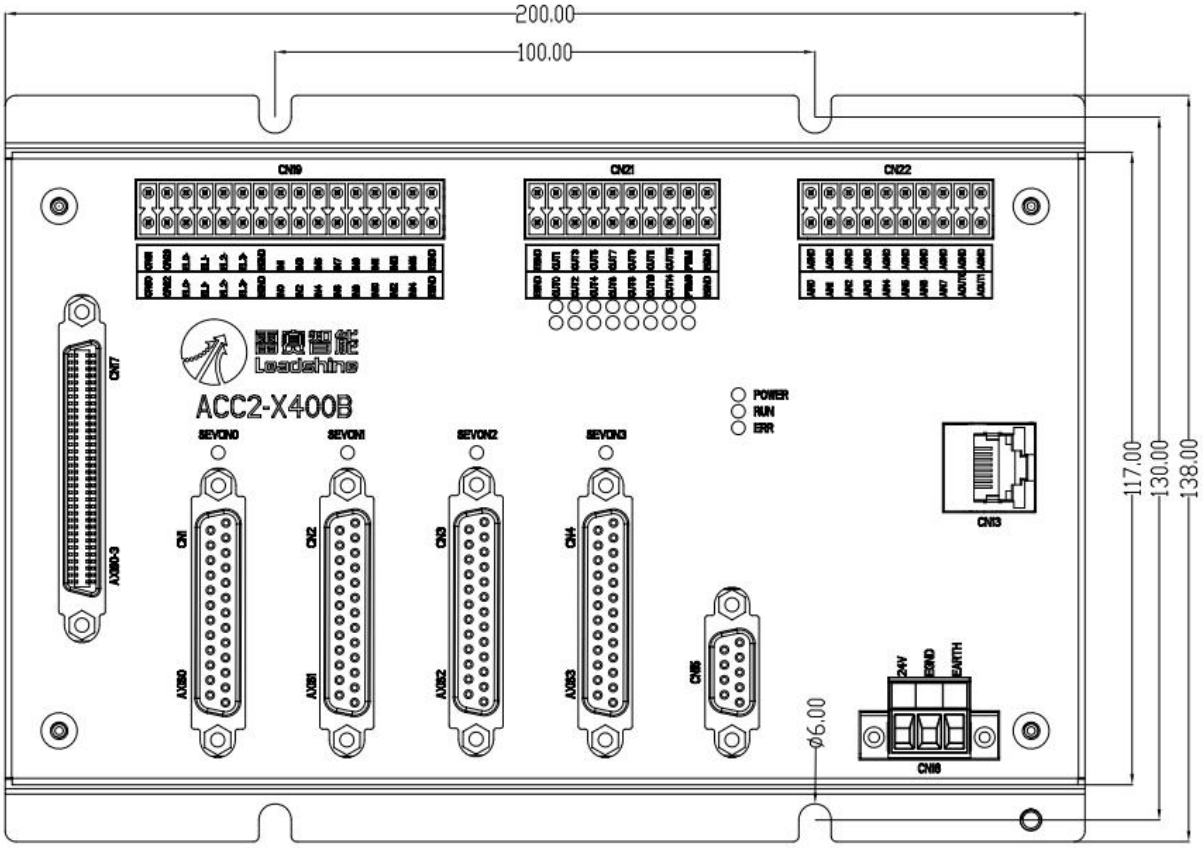


图 6.1　ACC2-X400B 接线盒尺寸图

ACC2-X400B 接线盒外形结构图如图 6.2 所示：

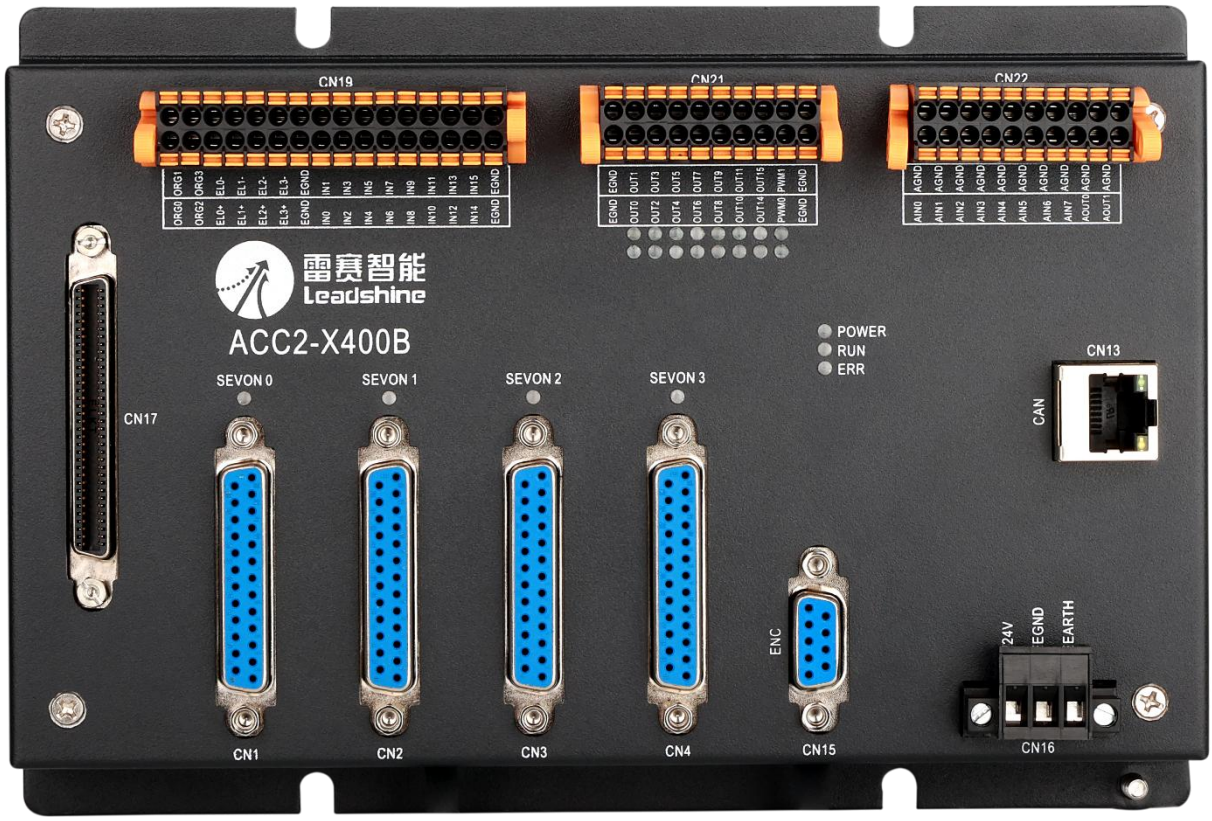


图 6.2　ACC2-X400B 接线盒外形结构图

1、 ACC2-X400B 接线盒接口及脚位

表 6.1　ACC2-X400B 接线盒接口功能简述

名称	功能介绍
CN1~CN4	第 1~4 轴轴控制信号
CN13	CAN 总线接口（RJ45）
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端子（1~4 轴）
CN19	第 1-4 轴原点及限位信号输入接口和数字量通用输入接口、
CN21	数字量输出端子和 PWM 输出接口
CN22	模拟量输入输出接口

2、 CN1~CN4 轴控制端子信号定义

1）ACC2-X400B 接线盒 CN1 至 CN4 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 6.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN4）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC2-X400B 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 6.2　接口 CN1~CN4 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC2-X400B 接线盒编码器口 1~2 轴仅支持差分接法，3~4 轴同时支持单端和差分接法

3、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

4、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 6.3 所示。

表 6.3　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

5、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 EGND 的端子接外部电源地,标有 EARTH 的端子为机壳地接口。

6、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

7、 CN19 原点及限位信号输入接口和数字量输入接口定义

CN19 为第 1-4 轴原点及限位信号输入接口和数字量通用输入接口，其引脚号和信号对应关系见表 6.4 所示：

表 6.4　接口 CN9 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	ORG0	I	0 轴原点输入	17	ORG1	I	1 轴原点输入
2	ORG2	I	2 轴原点输入	18	ORG3	I	3 轴原点输入
3	EL0+	I	0 轴正限位输入	19	EL0-	I	0 轴负限位输入
4	EL1+	I	1 轴正限位输入	20	EL1-	I	1 轴负限位输入
5	EL2+	I	2 轴正限位输入	21	EL2-	I	2 轴负限位输入
6	EL3+	I	3 轴正限位输入	22	EL3-	I	3 轴负限位输入
7	EGND	O	24V 电源地	23	EGND	O	24V 电源地
8	IN0	I	通用输入（低速）	24	IN1	I	通用输入（低速）
9	IN2	I	通用输入（低速）	25	IN3	I	通用输入（低速）
10	IN4	I	通用输入（低速）	26	IN5	I	通用输入（低速）
11	IN6	I	通用输入（低速）	27	IN7	I	通用输入（低速）
12	IN8	I	通用输入（低速）	28	IN9	I	通用输入（低速）
13	IN10	I	通用输入（低速）	29	IN11	I	通用输入（低速）
14	IN12	I	通用输入（低速）	30	IN13	I	通用输入（低速）

15	IN14	I	通用输入/LTC0（高速）	31	IN15	I	通用输入/LTC1（高速）
16	EGND	O	24V 电源地	32	EGND	O	24V 电源地

8、 CN21 数字量输出接口和 PWM 输出定义

CN21 为数字量输出端子和 PWM 输出接口，其引脚号和信号对应关系见表 6.6 所示。

表 6.6 接口 CN21 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	EGND	O	24V 电源地	11	EGND	O	24V 电源地
2	OUT0	O	通用输出（低速）	12	OUT1	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	13	OUT3	O	通用输出（低速）
4	OUT4	O	通用输出（低速）	14	OUT5	O	通用输出（低速）
5	OUT6	O	通用输出（低速）	15	OUT7	O	通用输出（低速）
6	OUT8	O	通用输出（低速）	16	OUT9	O	通用输出（低速）
7	OUT10	O	通用输出（低速）	17	OUT11	O	通用输出（低速）
8	OUT14	O	通用输出/CMP2（高速）	18	OUT15	O	通用输出/CMP3（高速）
9	PWM0	O	PWM 输出 0	19	PWM1	O	PWM 输出 1
10	EGND	O	24V 电源地	20	EGND	O	24V 电源地

9、 CN22 模拟量输入、输出接口定义

CN22 为模拟量输入、输出接口，其引脚号和信号对应关系见表 6.7 所示。

表 6.7 接口 CN22 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	AIN0	I	模拟量输入	11	GND	O	内部电源地
2	AIN1	I	模拟量输入	12	GND	O	内部电源地
3	AIN2	I	模拟量输入	13	GND	O	内部电源地
4	AIN3	I	模拟量输入	14	GND	O	内部电源地
5	AIN4	I	模拟量输入	15	GND	O	内部电源地
6	AIN5	I	模拟量输入	16	GND	O	内部电源地
7	AIN6	I	模拟量输入	17	GND	O	内部电源地
8	AIN7	I	模拟量输入	18	GND	O	内部电源地
9	AOUT0	O	模拟量输出	19	GND	O	内部电源地
10	AOUT1	O	模拟量输出	20	GND	O	内部电源地

说明： 1）支持 8 路模拟量输入和 2 路模拟量输出；

2）AIN0~AIN7 输入电压为-10V~＋10V，16bit 精度；

3）ADC 输入阻抗不小于 100K 欧姆；

4）待测信号的地与接线盒模拟输入端子的 11~18 脚（GND）连接，被采样信号接 AIN0~AIN7；

5）DAC 输出电压范围-10V~ + 10V，16it 精度，默认输出电压 0V。

10、 指示灯定义

ACC2-X400B 接线盒表面有 3 个指示灯，分别为：

POWER：24V 外部电源指示灯；

RUN：连接状态指示灯，绿色时表示接线盒与控制卡处于通讯状态；

ERR：错误指示灯，红色闪烁时表示接线盒与控制卡未连接成功；

附录 7　ACC2-3600 接线盒接口说明

ACC2-3600 接线盒是 DMC5610-PCIe 运动控制卡的配套产品，扩展 DMC5610-PCIe 控制卡的所有用户端子。

ACC2-3600 接线盒尺寸图如图 7.1 所示。

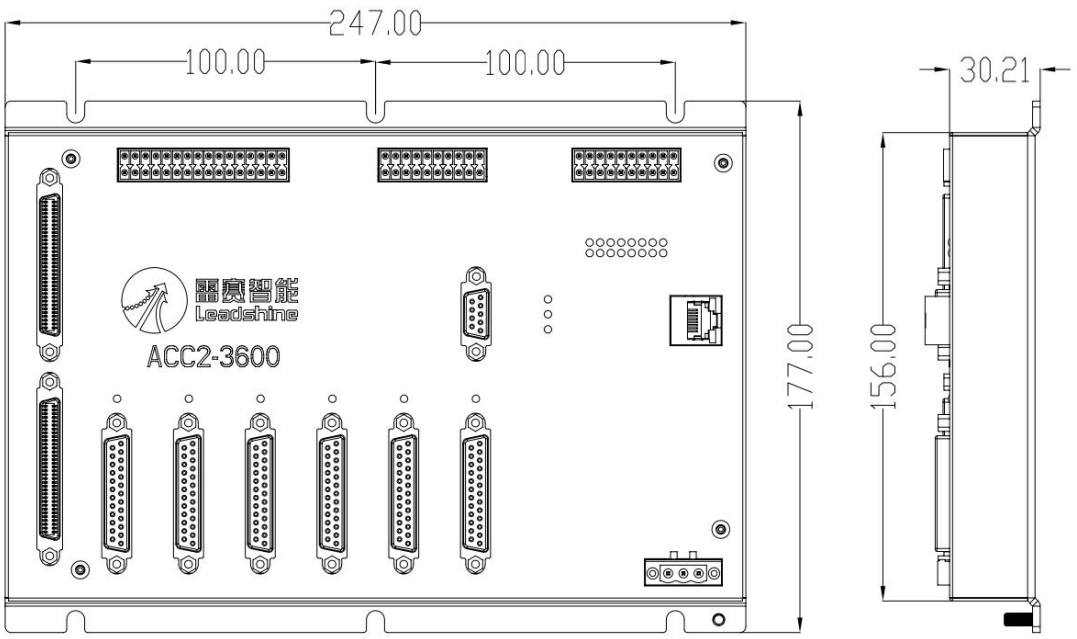


图 6.1　ACC2-3600 接线盒尺寸图

ACC2-3600 接线盒外形结构图如图 6.2 所示

图 6.2　ACC2-3600 接线盒外形结构图

1、 ACC2-3600 接线盒接口及脚位

表 7.1　ACC2-3600 接线盒接口功能简述

名称	功能介绍
CN1~CN6	第 1~6 轴轴控制信号
CN9	第 1-6 轴原点及限位信号输入接口
CN10	数字量输入端子
CN11	数字量输出端子
CN13	CAN 总线接口（RJ45）
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端口（0~3 轴）
CN18	运动控制卡连接端口（4~5 轴）

2、 CN1~CN6 轴控制端子信号定义

1）ACC2-3600 接线盒 CN1 至 CN6 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 7.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN6）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC2-3600 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 7.2　接口 CN1~CN6 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地

12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC2-3600 接线盒编码器口 1~4 轴仅支持差分接法，5~6 轴同时支持单端和差分接法

3、 CN9 原点及限位信号输入接口定义

CN9 为第 1-6 轴原点及限位信号输入接口，其引脚号和信号对应关系见表 7.3 所示：

表 7.3　接口 CN9 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	17	EGND	O	24V 电源地
2	ORG0	I	0 轴原点输入	18	ORG1	I	1 轴原点输入
3	ORG2	I	2 轴原点输入	19	ORG3	I	3 轴原点输入
4	ORG4	I	4 轴原点输入	20	ORG5	I	5 轴原点输入
5	NC		保留	21	NC		保留
6	EGND	O	24V 电源地	22	EGND	O	24V 电源地
7	EL0+	I	0 轴正限位输入	23	EL0-	I	0 轴负限位输入
8	EL1+	I	1 轴正限位输入	24	EL1-	I	1 轴负限位输入
9	EL2+	I	2 轴正限位输入	25	EL2-	I	2 轴负限位输入
10	EL3+	I	3 轴正限位输入	26	EL3-	I	3 轴负限位输入
11	EGND	O	24V 电源地	27	EGND	O	24V 电源地
12	EL4+	I	4 轴正限位输入	28	EL4-	I	4 轴负限位输入
13	EL5+	I	5 轴正限位输入	29	EL5-	I	5 轴负限位输入
14	NC		保留	30	NC		保留
15	NC		保留	31	NC		保留
16	EGND	O	24V 电源地	32	EGND	O	24V 电源地

4、 CN10 数字量输入接口定义

CN10 为数字量输入接口，其引脚号和信号对应关系见表 7.4 所示。

表 7.4　接口 CN10 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	11	EGND	O	24V 电源地
2	IN0	I	通用输入（低速）	12	IN1	I	通用输入（低速）
3	IN2	I	通用输入（低速）	13	IN3	I	通用输入（低速）
4	IN4	I	通用输入（低速）	14	IN5	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
5	IN6	I	通用输入（低速）	15	IN7	I	通用输入（低速）
6	IN8	I	通用输入（低速）	16	IN9	I	通用输入（低速）
7	IN10	I	通用输入（低速）	17	IN11	I	通用输入（低速）
8	IN12	I	通用输入（低速）	18	IN13	I	通用输入（低速）
9	IN14	I	通用输入/LTC0（高速）	19	IN15	I	通用输入/LTC1（高速）
10	EGND	O	24V 电源地	20	EGND	O	24V 电源地

5、 CN11 数字量输出接口定义

CN11 为数字量输出接口，其引脚号和信号对应关系见表 7.5 所示。

表 7.5　接口 CN11 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	11	EGND	O	24V 电源地
2	OUT0	O	通用输出（低速）	12	OUT1	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	13	OUT3	O	通用输出（低速）
4	OUT4	O	通用输出（低速）	14	OUT5	O	通用输出（低速）
5	OUT6	O	通用输出（低速）	15	OUT7	O	通用输出（低速）
6	OUT8	O	通用输出（低速）	16	OUT9	O	通用输出（低速）
7	OUT10	O	通用输出（低速）	17	OUT11	O	通用输出（低速）
8	OUT12	O	通用输出/CMP0（高速）	18	OUT13	O	通用输出/CMP1（高速）
9	OUT14	O	通用输出/CMP2（高速）	19	OUT15	O	通用输出/CMP3（高速）
10	EGND	O	24V 电源地	20	EGND	O	24V 电源地

注意：当 CMP0~3 端口不设置为高速位置比较器输出端口时，可以设置为 OUT12~15 通用输出端口。

6、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

7、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 7.6 所示。

表 7.6　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC

5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

8、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 EGND 的端子接外部电源地,标有 EARTH 的端子为机壳地接口。

9、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

10、CN18 接口定义

CN18 接口为 4~5 轴电机的控制信号及高速手轮通道与控制卡的接口。

11、指示灯定义

ACC2-3600 接线盒表面有 3 个指示灯，分别为：

POWER：24V 外部电源指示灯；

RUN：连接状态指示灯，绿色时表示接线盒与控制卡处于通讯状态；

ERR：错误指示灯，红色闪烁时表示接线盒与控制卡未连接成功；

附录 8　ACC2-3800 接线盒接口说明

ACC2-3800 接线盒是 DMC5810-PCIe 运动控制卡的配套产品，扩展 DMC5810-PCIe 控制卡的所有用户端子。

ACC2-3800 接线盒尺寸图如图 8.1 所示。

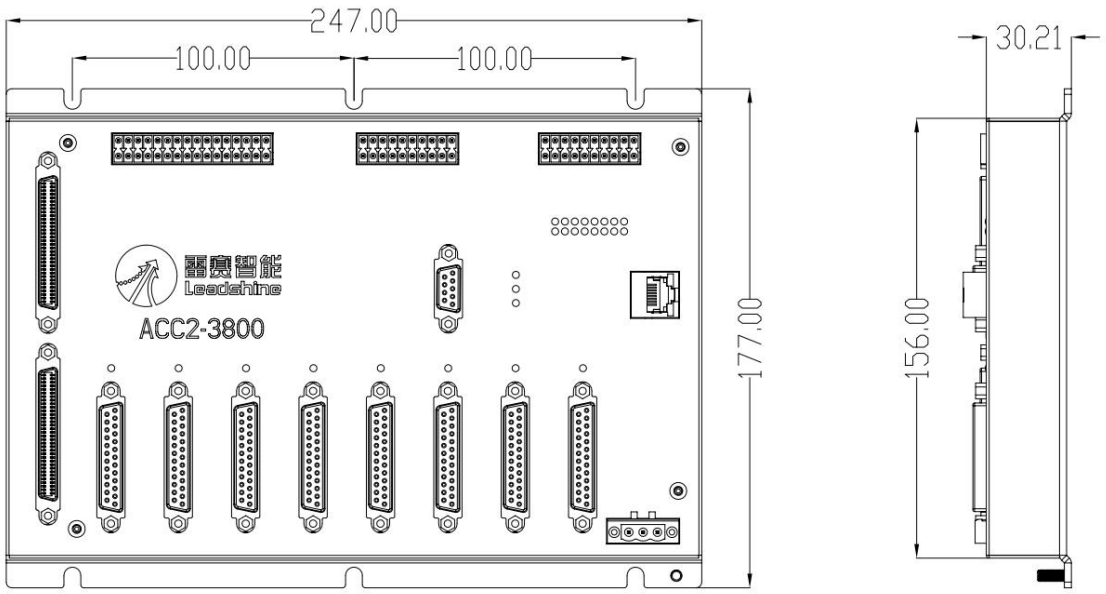


图 8.1　ACC2-3800 接线盒尺寸图

ACC2-3800 接线盒外形结构图如图 8.2 所示：

图 8.2　ACC2-3800 接线盒外形结构图

1、 ACC2-3800 接线盒接口及脚位

表 8.1　ACC2-3800 接线盒接口功能简述

名称	功能介绍
CN1~CN8	第 1~8 轴轴控制信号
CN9	第 1-8 轴原点及限位信号输入接口
CN10	数字量输入端子
CN11	数字量输出端子
CN13	CAN 总线接口（RJ45）
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端口（0~3 轴）
CN18	运动控制卡连接端口（4~7 轴）

2、 CN1~CN8 轴控制端子信号定义

1）ACC2-3800 接线盒 CN1 至 CN8 为电机控制信号端口，采用 DB25 母头，其引脚号和信号对应关系见表 7.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN6）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC2-3600 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 8.2　接口 CN1~CN8 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地

12	RDY	I	伺服准备完成	25	保留		保留
13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC2-3800 接线盒编码器口 1~6 轴仅支持差分接法，7~8 轴同时支持单端和差分接法

3、 CN9 原点及限位信号输入接口定义

CN9 为第 1-8 轴原点及限位信号输入接口，其引脚号和信号对应关系见表 8.3 所示：

表 8.3　接口 CN9 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	17	EGND	O	24V 电源地
2	ORG0	I	0 轴原点输入	18	ORG1	I	1 轴原点输入
3	ORG2	I	2 轴原点输入	19	ORG3	I	3 轴原点输入
4	ORG4	I	4 轴原点输入	20	ORG5	I	5 轴原点输入
5	ORG6	I	6 轴原点输入	21	ORG7	I	7 轴原点输入
6	EGND	O	24V 电源地	22	EGND	O	24V 电源地
7	EL0+	I	0 轴正限位输入	23	EL0-	I	0 轴负限位输入
8	EL1+	I	1 轴正限位输入	24	EL1-	I	1 轴负限位输入
9	EL2+	I	2 轴正限位输入	25	EL2-	I	2 轴负限位输入
10	EL3+	I	3 轴正限位输入	26	EL3-	I	3 轴负限位输入
11	EGND	O	24V 电源地	27	EGND	O	24V 电源地
12	EL4+	I	4 轴正限位输入	28	EL4-	I	4 轴负限位输入
13	EL5+	I	5 轴正限位输入	29	EL5-	I	5 轴负限位输入
14	EL6+	I	6 轴正限位输入	30	EL6-	I	6 轴负限位输入
15	EL7+	I	7 轴正限位输入	31	EL7-	I	7 轴负限位输入
16	EGND	O	24V 电源地	32	EGND	O	24V 电源地

4、 CN10 数字量输入接口定义

CN10 为数字量输入接口，其引脚号和信号对应关系见表 8.4 所示。

表 8.4　接口 CN10 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	11	EGND	O	24V 电源地
2	IN0	I	通用输入（低速）	12	IN1	I	通用输入（低速）
3	IN2	I	通用输入（低速）	13	IN3	I	通用输入（低速）
4	IN4	I	通用输入（低速）	14	IN5	I	通用输入（低速）

序	名称	I/O	说　明	序	名称	I/O	说　明
5	IN6	I	通用输入（低速）	15	IN7	I	通用输入（低速）
6	IN8	I	通用输入（低速）	16	IN9	I	通用输入（低速）
7	IN10	I	通用输入（低速）	17	IN11	I	通用输入（低速）
8	IN12	I	通用输入（低速）	18	IN13	I	通用输入（低速）
9	IN14	I	通用输入/LTC0（高速）	19	IN15	I	通用输入/LTC1（高速）
10	EGND	O	24V 电源地	20	EGND	O	24V 电源地

5、 CN11 数字量输出接口定义

CN11 为数字量输出接口，其引脚号和信号对应关系见表 8.5 所示。

表 8.5　接口 CN11 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	EGND	O	24V 电源地	11	EGND	O	24V 电源地
2	OUT0	O	通用输出（低速）	12	OUT1	O	通用输出（低速）
3	OUT2	O	通用输出（低速）	13	OUT3	O	通用输出（低速）
4	OUT4	O	通用输出（低速）	14	OUT5	O	通用输出（低速）
5	OUT6	O	通用输出（低速）	15	OUT7	O	通用输出（低速）
6	OUT8	O	通用输出（低速）	16	OUT9	O	通用输出（低速）
7	OUT10	O	通用输出（低速）	17	OUT11	O	通用输出（低速）
8	OUT12	O	通用输出/CMP0（高速）	18	OUT13	O	通用输出/CMP1（高速）
9	OUT14	O	通用输出/CMP2（高速）	19	OUT15	O	通用输出/CMP3（高速）
10	EGND	O	24V 电源地	20	EGND	O	24V 电源地

注意：当 CMP0~3 端口不设置为高速位置比较器输出端口时，可以设置为 OUT12~15 通用输出端口。

6、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

7、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 7.6 所示。

表 7.6　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC

5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

8、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 EGND 的端子接外部电源地,标有 EARTH 的端子为机壳地接口。

9、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

10、CN18 接口定义

CN18 接口为 4~5 轴电机的控制信号及高速手轮通道与控制卡的接口。

11、指示灯定义

ACC2-3800 接线盒表面有 3 个指示灯，分别为：

POWER：24V 外部电源指示灯；

RUN：连接状态指示灯，绿色时表示接线盒与控制卡处于通讯状态；

ERR：错误指示灯，红色闪烁时表示接线盒与控制卡未连接成功；

附录 9　ACC2-XC00 接线盒接口说明

ACC2-XC00 接线盒是 DMC5C10-PCIe 运动控制卡的配套产品，扩展 DMC5C10-PCIe 控制卡的所有用户端子。

ACC2-XC00 接线盒尺寸图如图 9.1 所示。

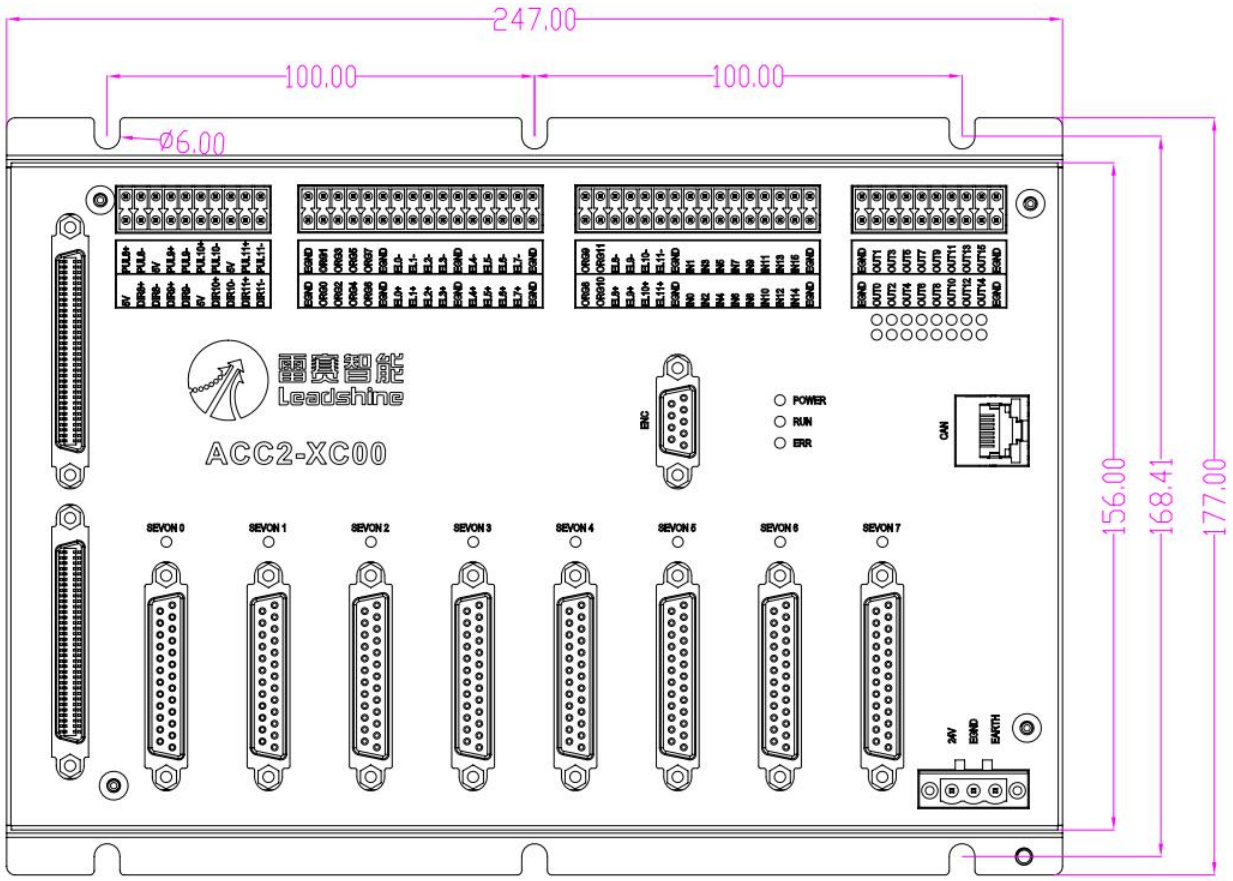


图 9.1　ACC2-XC00 尺寸图

ACC2-XC00 接线盒外形结构图如图 9.2 所示：

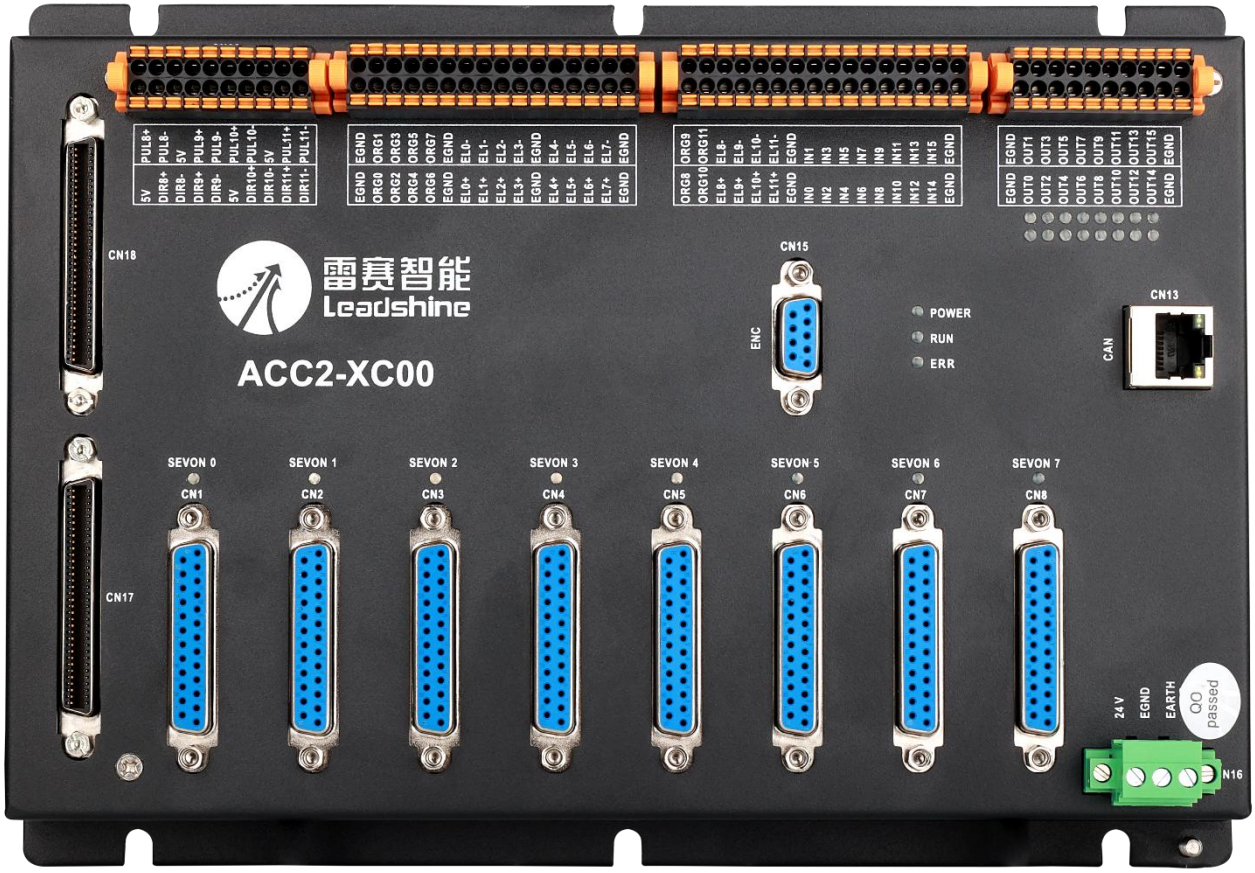


图 9.2　ACC2-XC00 接线盒外形结构图

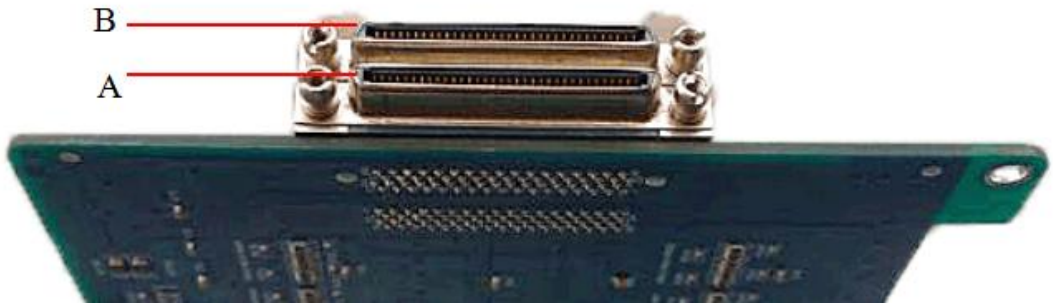


图 9.3　DMC5C10-PCIe侧视图

注　意：1）如图 9.3 所示，A为 0~3 轴的电机控制信号以及IO等信号端口，通过电缆线与ACC2-XC00 的CN17 端口相连；B为 4~11 轴电机的控制信号以及IO等信号端口，通过电缆线与ACC2-XC00 的CN18 端口相连。

2）DMC5C10-PCIe 与 ACC2-XC00 接线盒的连接示意图请参考第 3.2 节

1、 ACC2-XC00 接线盒接口及脚位

表 9.1 ACC2-XC00 接线盒接口功能简述

名称	功能介绍
CN1~CN8	第 1~8 轴轴控制信号
CN13	CAN 总线接口（RJ45）
CN15	辅助编码器接口
CN16	24V 直流电源输入端子
CN17	运动控制卡连接端子（1~4 轴）
CN18	运动控制卡连接端子（5~12 轴）
CN21~CN23	第 1~12 轴原点、限位以及通用 IO 信号接口
CN25	第 9~12 轴轴控制信号

2、 CN1~CN8 轴控制端子信号定义

1）ACC2-XC00 接线盒 CN1 至 CN8 以及 CN25 为电机控制信号端口，CN1 至 CN8 采用 DB25 母头，其引脚号和信号对应关系见表 9.2 所示。其中 24V 电源端口主要为控制卡的电机控制信号（伺服专用信号）供电。

2）电机控制信号端（CN1 至 CN8）的 24V 为控制信号电源，不能用于驱动器等动力负载供电。

3）根据 ACC2-XC00 接线盒的 PCB 板的铜线宽度与厚度，4 个电机控制信号端口的 24V 总输出电流需要控制在 1.5A 以内。

表 9.2 接口 CN1~CN8 引脚号和信号关系表

序	名称	I/O	说 明	序	名称	I/O	说 明
1	OGND	O	24V 电源地	14	24V	O	+24V 输出
2	ALM	I	驱动报警	15	ERC	O	驱动报警复位
3	SEVON	O	驱动允许	16	INP	I	到位信号
4	EA-	I	编码器输入	17	EA+	I	编码器输入
5	EB-	I	编码器输入	18	EB+	I	编码器输入
6	EZ-	I	编码器输入	19	EZ+	I	编码器输入
7	+5V	O	内部 5V	20	GND	O	内部 5V 地
8	保留	-	保留	21	GND	O	内部 5V 地
9	DIR+	O	方向输出	22	DIR-	O	方向输出
10	GND	O	内部 5V 地	23	PUL+	O	脉冲输出
11	PUL-	O	脉冲输出	24	GND	O	内部 5V 地
12	RDY	I	伺服准备完成	25	保留		保留

13	GND	O	内部 5V 地				

注意：（1）当使用+5V 和 PUL-端口时，则选择电机指令脉冲信号输出方式为单端输出；当使用 PUL+和 PUL-端口时，则选择电机指令脉冲信号输出方式为差分输出。

（2）ACC2-XC00 接线盒 1~6 轴编码器口仅支持差分接法，7~8 轴同时支持单端和差分接法

3、 CN13 IO 扩展接口定义

CN13 为 IO 扩展接口，采用 RJ45 接口，可连接 CAN-IO 扩展模块。

4、 CN15 辅助编码器/手轮接口定义

CN15 为辅助编码器/手轮接口，采用 DB9 母头。其引脚号和信号对应关系见表 9.3 所示。

表 9.3　接口 CN15 引脚号和信号关系表

序	名称	I/O	说　明
1	EA+	I	编码器/手轮输入
2	EB+	I	编码器/手轮输入
3	EZ+	I	编码器输入
4		-	NC
5	+5V	O	内部 5V 输出
6	EA-	I	编码器/手轮输入
7	EB-	I	编码器/手轮输入
8	EZ-	I	编码器输入
9	GND	O	内部 5V 地

5、 CN16 电源定义

接口 CN16 是接线盒的电源输入接口，板上标有 24V 的端子接+24，标有 OGND 的端子接外部电源地，标有 FG 的端子为机壳地接口。

表 9.4　接口 CN16 引脚号和信号关系表

序号	名称	功能
1	+24V	外部 24V 电源输入
2	EGND	外部 24V 电源地
3	EARTH	机壳地

6、 CN17 接口定义

CN17 接口为 0~3 轴电机的控制信号、IO 信号及低速手轮通道等与控制卡的接口。

7、 CN18 接口定义

CN18 接口为 4~11 轴电机的控制信号及高速手轮通道与控制卡的接口。

8、 CN21 专用输入接口定义

CN21 为数字量专用输入接口，其引脚号和信号对应关系见表 9.5 所示：

表 9.5　接口 CN21 引脚号和信号关系

序	名称	I/O	说明	序	名称	I/O	说明
1	EGND	O	24V 电源地	17	EGND	O	24V 电源地
2	ORG0	I	0 轴原点输入	18	ORG1	I	1 轴原点输入
3	ORG2	I	2 轴原点输入	19	ORG3	I	3 轴原点输入
4	ORG4	I	4 轴原点输入	20	ORG5	I	5 轴原点输入
5	ORG6	I	6 轴原点输入	21	ORG7	I	7 轴原点输入
6	EGND	O	24V 电源地	22	EGND	O	24V 电源地
7	EL0+	I	0 轴正限位输入	23	EL0-	I	0 轴负限位输入
8	EL1+	I	1 轴正限位输入	24	EL1-	I	1 轴负限位输入
9	EL2+	I	2 轴正限位输入	25	EL2-	I	2 轴负限位输入
10	EL3+	I	3 轴正限位输入	26	EL3-	I	3 轴负限位输入
11	EGND	O	24V 电源地	27	EGND	O	24V 电源地
12	EL4+	I	4 轴正限位输入	28	EL4-	I	4 轴负限位输入
13	EL5+	I	5 轴正限位输入	29	EL5-	I	5 轴负限位输入
14	EL6+	I	6 轴正限位输入	30	EL6-	I	6 轴负限位输入
15	EL7+	I	7 轴正限位输入	31	EL7-	I	7 轴负限位输入
16	EGND	O	24V 电源地	32	EGND	O	24V 电源地

9、 CN22 通用输入接口定义

CN22 为数字量通用输入和步进轴专用信号接口，其引脚号和信号对应关系见表 9.6 所示：

表 9.6　接口 CN21 引脚号和信号关系

序	名称	I/O	说　明	序	名称	I/O	说　明

1	ORG8	I	8 轴原点输入	17	ORG9	I	9 轴原点输入
2	ORG10	I	10 轴原点输入	18	ORG11	I	11 轴原点输入
3	EL8+	I	8 轴正限位输入	19	EL8-	I	8 轴负限位输入
4	EL9+	I	9 轴正限位输入	20	EL9-	I	9 轴负限位输入
5	EL10+	I	10 轴正限位输入	21	EL10-	I	10 轴负限位输入
6	EL11+	I	11 轴正限位输入	22	EL11-	I	11 轴负限位输入
7	EGND	O	24V 电源地	23	EGND	O	24V 电源地
8	IN0	I	通用输入（低速）	24	IN1	I	通用输入（低速）
9	IN2	I	通用输入（低速）	25	IN3	I	通用输入（低速）
10	IN4	I	通用输入（低速）	26	IN5	I	通用输入（低速）
11	IN6	I	通用输入（低速）	27	IN7	I	通用输入（低速）
12	IN8	I	通用输入（低速）	28	IN9	I	通用输入（低速）
13	IN10	I	通用输入（低速）	29	IN11	I	通用输入（低速）
14	IN12	I	通用输入（低速）	30	IN13	I	通用输入（低速）
15	IN14	I	通用输入/LTC0（高速）	31	IN15	I	通用输入/LTC0（高速）
16	EGND	O	24V 电源地	32	EGND	O	24V 电源地

10、CN23 数字量输出接口定义

CN23 为数字量输出接口，其引脚号和信号对应关系见表 9.7 所示。

表 9.7　接口 CN23 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	OUT0	O	通用输出（低速）	9	OUT1	O	通用输出（低速）
2	OUT2	O	通用输出（低速）	10	OUT3	O	通用输出（低速）
3	OUT4	O	通用输出（低速）	11	OUT5	O	通用输出（低速）
4	OUT6	O	通用输出（低速）	12	OUT7	O	通用输出（低速）
5	OUT8	O	通用输出（低速）	13	OUT9	O	通用输出（低速）
6	OUT10	O	通用输出（低速）	14	OUT11	O	通用输出（低速）
7	OUT12	O	通用输出/CMP0（高速）	15	OUT13	O	通用输出/CMP1（高速）
8	OUT14	O	通用输出/CMP2（高速）	16	OUT15	O	通用输出/CMP3（高速）

11、CN25 电机接口端子定义

CN25 为 8~11 轴电机控制信号端口，其引脚号和信号对应关系见表 9.8 所示。

表 9.8　接口 CN25 引脚号和信号关系表

序	名称	I/O	说　明	序	名称	I/O	说　明
1	5V	O	内部 5V 输出	11	PUL8+	O	轴 8 脉冲输出
2	DIR8+	O	轴 8 方向输出	12	PUL8-	O	轴 8 脉冲输出
3	DIR8-	O	轴 8 方向输出	13	5V	O	内部 5V 输出

4	DIR9+	O	轴 9 方向输出	14	PUL9+	O	轴 9 脉冲输出
5	DIR9-	O	轴 9 方向输出	15	PUL9-	O	轴 9 脉冲输出
6	5V	O	内部 5V 输出	16	PUL10+	O	轴 10 脉冲输出
7	DIR10+	O	轴 10 方向输出	17	PUL10-	O	轴 10 脉冲输出
8	DIR10-	O	轴 10 方向输出	18	5V	O	内部 5V 输出
9	DIR11+	O	轴 11 方向输出	19	PUL11+	O	轴 11 脉冲输出
10	DIR11-	O	轴 11 方向输出	20	PUL11-	O	轴 11 脉冲输出

12、指示灯定义

ACC2-XC00 接线盒表面有 3 个指示灯，分别为：

POWER：24V 外部电源指示灯；

RUN：连接状态指示灯，绿色时表示接线盒与控制卡处于通讯状态；

ERR：错误指示灯，红色闪烁时表示接线盒与控制卡未连接成功；

附录 10　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_reset	控制卡硬件复位函数	
	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_release_version	获取控制卡发布版本	
	dmc_get_total_axes	获取当前卡的轴数	
	dmc_get_total_ionum	获取控制卡 IO 点数	
	dmc_get_total_adcnum	获取控制卡模拟量数	
	dmc_download_configfile	下载参数文件	
	dmc_download_firmware	下载固件文件	
脉冲模式设置函数	dmc_set_pulse_outmode	设置指定轴的脉冲输出模式	9.2 节
	dmc_get_pulse_outmode	读取指定轴的脉冲输出模式设置	
回原点运动函数	dmc_set_home_pin_logic	设置 ORG 原点信号	9.3 节
	dmc_get_home_pin_logic	读取 ORG 原点信号设置	
	dmc_set_homemode	设置回原点模式	
	dmc_get_homemode	读取回原点模式	
	dmc_set_home_el_return	设置非限位回零模式遇限位反找模式	
	dmc_get_home_el_return	读取非限位回零模式遇限位反找模式	
	dmc_set_home_position_unit	设置回零偏移量及清零模式	
	dmc_get_home_position_unit	读取回零偏移量及清零模式	
	dmc_set_el_ret_deviation	设置限位返找偏移距离	
	dmc_get_el_ret_deviation	读取限位返找偏移距离	
	dmc_home_move	回原点运动	
原点锁存及 EZ 锁存函数	dmc_set_homelatch_mode	设置原点锁存模式	9.4 节
	dmc_get_homelatch_mode	读取原点锁存模式设置	
	dmc_reset_homelatch_flag	清除原点锁存标志	
	dmc_get_homelatch_flag	读取原点锁存标志	
	dmc_get_homelatch_value	读取原点锁存值	
	dmc_get_homelatch_value_unit	读取原点锁存值	
	dmc_set_ezlatch_mode	设置 EZ 锁存模式	
	dmc_get_ezlatch_mode	读取 EZ 锁存模式设置	
	dmc_reset_ezlatch_flag	清除 EZ 锁存标志	
	dmc_get_ezlatch_flag	读取 EZ 锁存标志	
	dmc_get_ezlatch_value	读取 EZ 锁存值	
	dmc_get_ezlatch_value_unit	读取 EZ 锁存值	

函数分类	函数名	描 述	索引
限位开关设置函数	dmc_set_el_mode	设置 EL 限位信号	9.5 节
	dmc_get_el_mode	读取 EL 限位信号设置	
	dmc_set_softlimit_unit	设置软限位	
	dmc_get_softlimit_unit	读取软限位设置	
	dmc_set_softlimit	设置软限位	
	dmc_get_softlimit	读取软限位设置	
位置计数器控制函数	dmc_set_position	设置指令脉冲位置	9.6 节
	dmc_get_position	读取指令脉冲位置	
运动状态检测及控制函数	dmc_read_current_speed	读取当前速度值	9.7 节
	dmc_get_target_position	读取轴的目标位置	
	dmc_LinkState	检测主卡与接线盒的通讯连接状态	
	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	设置单轴运动速度曲线	9.8 节
	dmc_get_profile	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	指定轴点位运动	9.9 节
	dmc_vmove	指定轴连续运动	
	dmc_change_speed	在线变速	
	dmc_reset_target_position	在线变位	
	dmc_update_target_position	强行变位（在线/非在线）	
	dmc_pmove_change_pos_speed_config	配置原点信号（ORG）触发在线变速变位置参数	
	dmc_get_pmove_change_pos_speed_config	读取配置的原点信号(ORG)触发在线变速变位置参数	
	dmc_pmove_change_pos_speed_enable	原点信号(ORG)触发在线变速变位置使能	
	dmc_get_pmove_change_pos_speed_state	原点信号(ORG)触发在线变速变位置使能	
PVT 运动函数	dmc_PttTable	向指定数据表传送数据，采用 PTT 模式	9.10 节
	dmc_PtsTable	向指定数据表传送数据，采用 PTS 模式	
	dmc_PvtTable	向指定数据表传送数据，采用 PVT 模式	
	dmc_PvtsTable	向指定数据表传送数据，采用 PVTS 模式	
	dmc_PvtMove	启动 PVT 运动	
伺服驱动专用接口函数	dmc_write_sevon_pin	控制指定轴的伺服使能端口的输出	9.11 节
	dmc_read_sevon_pin	读取指定轴的伺服使能端口的电平状态	

函数分类	函数名	描 述	索引
	dmc_read_rdy_pin	读取指定轴的 RDY 端口的电平状态	
	dmc_set_inp_mode	设置指定轴的 INP 信号	
	dmc_get_inp_mode	读取指定轴的 INP 信号设置	
	dmc_set_alm_mode	设置指定轴的 ALM 信号	
	dmc_get_alm_mode	读取指定轴的 ALM 信号设置	
	dmc_write_erc_pin	控制指定轴的 ERC 信号输出	
	dmc_read_erc_pin	读取指定轴的 ERC 端口电平状态	
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	9.12 节
	dmc_read_inbit_ex		
	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_outbit_ex		
	dmc_read_inport	读取指定控制卡的全部输入口的电平状态	
	dmc_read_inport_ex		
	dmc_read_outport	读取指定控制卡的全部输出口的电平状态	
	dmc_read_outport_ex		
	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_output_status_repower	控制卡接线盒断电重新上电输出口是否保持断电前状态	
	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	9.13 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_channel	手轮通道选择设置	
	dmc_get_handwheel_channel	读取手轮通道选择设置	
	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_counter_inmode	设置编码器的计数方式	9.14 节
	dmc_get_counter_inmode	读取编码器的计数方式	
	dmc_set_encoder	设置指定轴编码器反馈位置脉冲计数值	
	dmc_get_encoder	读取指定轴编码器反馈位置脉冲计数值	
	dmc_set_ez_mode	设置指定轴的 EZ 信号	
	dmc_get_ez_mode	读取指定轴的 EZ 信号设置	
位置锁存函数	dmc_softltc_set_mode	配置锁存器	9.15 节
	dmc_softltc_get_mode	读取锁存器配置	
	dmc_softltc_set_source	设置锁存源	

函数分类	函数名	描 述	索引
	dmc_softltc_get_source	读取锁存源	
	dmc_softltc_get_value_unit	读取锁存值	
	dmc_softltc_get_number	读取已锁存个数	
	dmc_softltc_reset	复位指定卡的锁存器的标志位	
	dmc_set_ltc_mode	设置指定轴的 LTC 信号	
	dmc_get_ltc_mode	读取指定轴的 LTC 信号设置	
	dmc_set_latch_mode	设置锁存方式	
	dmc_get_latch_mode	读取锁存方式	
	dmc_set_latch_stop_time	设置锁存触发延时急停时间	
	dmc_get_latch_stop_time	读取锁存触发延时急停时间设置	
	dmc_SetLtcOutMode	LTC 反相输出设置	
	dmc_GetLtcOutMode	读取 LTC 反相输出设置	
	dmc_get_latch_value	读取锁存值	
	dmc_get_latch_value_unit	读取锁存值	
	dmc_get_latch_flag	读取已锁存个数	
	dmc_reset_latch_flag	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	9.16 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_add_point_unit	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_current_point_unit	读取当前一维比较点位置	
	dmc_compare_get_points_runned	查询已经比较过的一维比较点个数	
	dmc_compare_get_points_remained	查询可以加入的一维比较点个数	
	dmc_compare_set_config_extern	设置二维位置比较器	
	dmc_compare_get_config_extern	读取二维位置比较器设置	
	dmc_compare_clear_points_extern	清除二维位置比较点	
	dmc_compare_add_point_extern	添加二维位置比较点	
	dmc_compare_add_point_extern_unit	添加二维位置比较点	
	dmc_compare_get_current_point_extern	读取当前二维位置比较点位置	
	dmc_compare_get_current_point_extern_unit	读取当前二维位置比较点位置	
	dmc_compare_get_points_runned_extern	查询已经比较过的二维比较点个数	
	dmc_compare_get_points_remained_extern	查询可以加入的二维比较点个数	
高速位置比较函数	dmc_hcmp_set_mode	设置高速比较模式	9.17 节
	dmc_hcmp_get_mode	读取高速比较模式设置	

函数分类	函数名	描 述	索引
	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_add_point_unit	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_set_liner_unit	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_get_liner_unit	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	
	dmc_hcmp_get_current_state_unit	读取高速比较参数	
	dmc_hcmp_fifo_set_mode	启用缓存方式添加比较位置	
	dmc_hcmp_fifo_get_mode	缓存方式添加比较位置模式回读	
	dmc_hcmp_fifo_add_table	按数组的方式批量添加比较位置	
	dmc_hcmp_fifo_clear_points	清除比较位置	
	dmc_hcmp_fifo_get_state	读取剩余缓存状态	
	dmc_write_cmp_pin	控制指定 CMP 管脚的输出	
	dmc_read_cmp_pin	读取指定 CMP 管脚的电平状态	
异常信号接口函数	dmc_set_emg_mode	设置 EMG 急停信号	9.18 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	
	dmc_set_vector_dec_stop_time	设置插补运动异常减速停止时间	
	dmc_get_vector_dec_stop_time	回读设置的插补运动异常减速停止时间	
轴 IO 映射函数	dmc_set_axis_io_map	设置轴 IO 映射关系	9.19 节
	dmc_get_axis_io_map	读取轴 IO 映射关系设置，	
	dmc_set_special_input_filter	设置所有专用 IO 滤波时间	
虚拟 IO 映射函数	dmc_set_io_map_virtual	设置虚拟 IO 映射关系	9.20 节
	dmc_get_io_map_virtual	读取虚拟 IO 映射关系设置	
	dmc_read_inbit_virtual	读取滤波后的虚拟 IO 口电平状态	
检测轴到位状态函数	dmc_set_factor_error	设置位置误差带	9.21 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	
	dmc_set_pulse_encoder_count_error	设置脉冲计数值和编码器反馈值之间差值的报警阀值	
	dmc_get_pulse_encoder_count_error	回读脉冲计数值和编码器反馈值之间差	

函数分类	函数名	描 述	索引
		值的报警阀值	
	dmc_check_pulse_encoder_count_error	检查脉冲计数值和编码器反馈值之间差值是否超过报警阀值	
	dmc_set_encoder_count_error_action_config	检测指令位置与编码器偏差超过报警阀值时停止运动	
	dmc_get_encoder_count_error_action_config	回读检测指令位置与编码器偏差超过报警阀值时停止运动	
CAN-IO 扩展函数	nmc_set_connect_state	设置 CAN-IO 通讯状态	9.22 节
	nmc_get_connect_state	读取 CAN-IO 通讯状态	
	nmc_write_outbit	设置指定 CAN-IO 扩展模块的某个输出端口的电平	
	nmc_write_outbit_ex		
	nmc_read_outbit	读取指定 CAN-IO 扩展模块的某个输出端口的电平	
	nmc_read_outbit_ex		
	nmc_read_inbit	读取指定 CAN-IO 扩展模块的某个输入端口的电平	
	nmc_read_inbit_ex		
	nmc_write_outport	设置指定 CAN-IO 扩展模块的全部输出口的电平	
	nmc_write_outport_ex		
	nmc_read_outport	读取指定 CAN-IO 扩展模块的全部输出口的电平	
	nmc_read_outport_ex		
	nmc_read_inport	读取指定 CAN-IO 扩展模块的全部输入端口的电平	
	nmc_read_inport_ex		
	nmc_set_da_mode	设置指定 CAN-ADDA 扩展模块的某个模拟量输出口的模式	
	nmc_set_da_mode_ex		
	nmc_get_da_mode	读取指定 CAN-ADDA 扩展模块的某个模拟量输出口的模式	
	nmc_get_da_mode_ex		
	nmc_set_ad_mode	设置指定 CAN-ADDA 扩展模块的某个模拟量输入口的模式	
	nmc_set_ad_mode_ex		
	nmc_get_ad_mode	读取指定 CAN-ADDA 扩展模块的某个模拟量输入口的模式	
	nmc_get_ad_mode_ex		
	nmc_set_da_output	设置指定 CAN-ADDA 扩展模块的某个模拟量输出口的电压/电流	
	nmc_set_da_output_ex		
	nmc_get_da_output	读取指定 CAN-ADDA 扩展模块的某个模拟量输出口的电压/电流	
	nmc_get_da_output_ex		
	nmc_get_ad_input	读取指定 CAN-ADDA 扩展模块的某个模拟量输入端口的电压/电流	
	nmc_get_ad_input_ex		
	nmc_write_to_flash	保存模式设置到模块 flash	
	nmc_write_to_flash_ex		
密码管理函数	dmc_write_sn	修改密码	9.23 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	9.24 节
	dmc_get_debug_mode	读取函数调用打印输出设置	
脉冲当量设	dmc_set_equiv	设置脉冲当量值	10.1 节

函数分类	函数名	描 述	索引
置	dmc_get_equiv	返回脉冲当量值设置	
状态检测	dmc_get_axis_run_mode	读取轴运动模式	10.2 节
	dmc_conti_get_run_state	读取连续插补运动状态	
	dmc_set_position_unit	设置当前指令位置计数器值	
	dmc_get_position_unit	读取当前指令位置计数器值	
	dmc_set_encoder_unit	设置当前编码器计数值	
	dmc_get_encoder_unit	读取当前编码器计数值	
	dmc_read_current_speed_unit	读取轴当前速度	
	dmc_get_target_position_unit	读取轴的目标位置	
	dmc_get_stop_reason	读取轴停止原因	
	dmc_clear_stop_reason	清除轴停止原因	
	dmc_calculate_arclength_3point	计算三点圆弧的弧长	
点位运动	dmc_set_profile_unit	设置单轴运动速度曲线	10.3 节
	dmc_get_profile_unit	读取单轴运动速度曲线	
	dmc_pmove_unit	定长运动	
	dmc_reset_target_position_unit	在线变位	
	dmc_update_target_position_unit	强行变位（在线/非在线）	
	dmc_change_speed_unit	在线变速	
插补 速度设置	dmc_set_vector_profile_unit	设置插补运动速度曲线	10.4 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	dmc_line_unit	直线插补运动	10.5 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_rectangle_move_unit	矩形插补运动	
	dmc_axis_follow_line_enable	直线插补参与插补轴数设置	
连续 插补运动	dmc_conti_open_list	打开连续插补缓冲区	10.6 节
	dmc_conti_start_list	开始连续插补	
	dmc_conti_close_list	关闭连续插补缓冲区	
	dmc_conti_pause_list	暂停连续插补	
	dmc_conti_stop_list	停止连续插补	
	dmc_conti_set_override	设置连续插补段速度比例	
	dmc_conti_change_speed_ratio	动态调整连续插补速度比例	
	dmc_conti_delay	连续插补中暂停延时指令	
	dmc_conti_line_unit	连续插补中直线插补指令	
	dmc_conti_arc_move_center_unit	连续插补中基于圆心圆弧扩展的螺旋线	

函数分类	函数名	描 述	索引
		插补指令（可作两轴圆弧插补）	
	dmc_conti_arc_move_radius_unit	连续插补中基于半径圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）	
	dmc_conti_arc_move_3points_unit	连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴或三轴圆弧插补）	
	dmc_conti_rectangle_move_unit	连续插补中矩形插补指令	
	dmc_conti_pmove_unit	连续插补中控制指定外轴运动指令	
	dmc_conti_pmove_unit_pausemode	连续插补暂停后，执行单轴运动	
	dmc_conti_return_pausemode	连续插补暂停，执行单轴运动后，回到暂停位置	
连续插补缓冲区检测	dmc_conti_remain_space	查询连续插补缓冲区剩余插补空间	10.7 节
	dmc_conti_read_current_mark	读取连续插补缓冲区当前插补段号	
连续插补小线段前瞻功能	dmc_conti_set_lookahead_mode	设置连续插补前瞻参数	10.8 节
	dmc_conti_get_lookahead_mode	读取连续插补前瞻参数	
连续插补 IO 控制	dmc_conti_set_pause_output	设置连续插补暂停及异常停止时 IO 输出状态	10.9 节
	dmc_conti_get_pause_output	读取连续插补暂停及异常停止时 IO 输出状态设置	
	dmc_conti_wait_input	连续插补等待 IO 输入	
	dmc_conti_delay_outbit_to_start	连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）	
	dmc_conti_delay_outbit_to_stop	连续插补中相对于轨迹段终点 IO 滞后输出	
	dmc_conti_ahead_outbit_to_stop	连续插补中相对于轨迹段终点 IO 提前输出（段内执行）	
	dmc_conti_accurate_outbit_unit	连续插补中精确位置 CMP 输出控制	
	dmc_conti_write_outbit	连续插补中缓冲区立即 IO 输出	
	dmc_conti_clear_io_action	清除段内未执行完的 IO 动作	
设置螺旋线插补运动模式	dmc_conti_set_involute_mode	设置螺旋线插补运动模式	
	dmc_conti_get_involute_mode	读取螺旋线插补运动模式设置	10.10 节
反向间隙设置	dmc_set_backlash_unit	设置反向间隙值	10.11 节
	dmc_get_backlash_unit	读取反向间隙值设置	
PWM 功能	dmc_set_pwm_enable	设置 PWM 使能状态	10.12 节
	dmc_get_pwm_enable	读取 PWM 使能状态设置	
	dmc_set_pwm_enable_extern	设置 PWM 使能状态	
	dmc_get_pwm_enable_extern	读取 PWM 使能状态设置	
	dmc_set_pwm_output	设置 PWM 输出	

函数分类	函数名	描 述	索引
	dmc_get_pwm_output	读取 PWM 输出设置	
连续插补 PWM 输出	dmc_set_pwm_onoff_duty	设置 PWM 开关状态对应的占空比	10.13 节
	dmc_get_pwm_onoff_duty	读取 PWM 开关状态对应占空比的设置	
	dmc_conti_set_pwm_output	连续插补中 PWM 输出设置	
	dmc_conti_set_pwm_follow_speed	连续插补中 PWM 速度跟随	
	dmc_conti_get_pwm_follow_speed	读取 PWM 速度跟随参数设置	
	dmc_conti_delay_pwm_to_start	连续插补中相对于轨迹段起点 PWM 滞后输出	
	dmc_conti_ahead_pwm_to_stop	连续插补中相对于轨迹段终点 PWM 提前输出	
	dmc_conti_write_pwm	连续插补中缓冲区立即 PWM 输出	
圆弧限速	dmc_set_arc_limit	设置圆弧限速参数	10.14 节
	dmc_get_arc_limit	读取圆弧限速参数	
DA/AD 功能	dmc_set_da_enable	设置 DA 输出使能	10.15 节
	dmc_get_da_enable	读取 DA 输出使能	
	dmc_set_da_output	设置 DA 输出	
	dmc_get_da_output	读取 DA 输出	
	dmc_get_ad_input	读取 AD 输入	
	dmc_get_ad_input_all	读取所有 AD 输入	
连续插补 DA 跟随	dmc_conti_set_da_enable	设置连续插补中 DA 输出使能	10.16 节
	dmc_conti_set_da_follow_speed	设置连续插补中 DA 速度跟随	
	dmc_conti_get_da_follow_speed	读取连续插补中 DA 速度跟随	
	dmc_conti_set_encoder_da_follow_enable	设置 DA 单轴编码器速度跟随	
	dmc_conti_get_encoder_da_follow_enable	读取 DA 单轴编码器速度跟随	
二维高速位置比较	dmc_hcmp_2d_set_enable	设置二维高速比较使能	10.17 节
	dmc_hcmp_2d_get_enable	读取二维高速比较使能	
	dmc_hcmp_2d_set_config_unit	配置二维高速比较器	
	dmc_hcmp_2d_get_config_unit	读取配置的二维高速比较器	
	dmc_hcmp_2d_clear_points	清除二维高速比较位置	
	dmc_hcmp_2d_set_pwmoutput	配置二维比较输出的 PWM 模式	
	dmc_hcmp_2d_get_pwmoutput	读取配置的二维比较 PWM 模式	
	dmc_hcmp_2d_add_point_unit	添加/更新二维高速比较位置	
	dmc_hcmp_2d_get_current_state_unit	读取二维高速比较参数	
	dmc_hcmp_2d_force_output	强制二维比较输出	
	dmc_hcmp_2d_set_config	配置二维高速比较器	
	dmc_hcmp_2d_get_config	读取二维高速比较器配置	
	dmc_hcmp_2d_add_point	添加/更新二维高速比较位置	
	dmc_hcmp_2d_get_current_state	读取二维高速比较参数	

函数分类	函数名	描 述	索引
连续插补刀向跟随	dmc_conti_gear_unit	连续插补中刀向跟随	10.18 节
螺距补偿功能	dmc_enable_leadscrew_comp	设置螺距补偿的使能与禁止	10.19 节
	dmc_set_leadscrew_comp_config	配置螺距补偿参数	
	dmc_get_leadscrew_comp_config	读取配置的螺距补偿参数	
	dmc_set_leadscrew_comp_config_unit	配置螺距补偿参数	
	dmc_get_leadscrew_comp_config_unit	读取配置的螺距补偿参数	
	dmc_get_position_ex_unit	读取螺距补偿后的位置	
龙门功能	dmc_set_gear_follow_profile	设置龙门跟随模式参数	10.20 节
	dmc_get_gear_follow_profile	读取龙门跟随模式参数	
	dmc_set_grant_error_protect	设置龙门模式主从轴编码器跟随误差停止阀值	
	dmc_get_grant_error_protect	读取龙门模式编码器位置跟随误差停止阈值	
	dmc_set_grant_error_protect_unit	设置龙门模式主从轴编码器跟随误差停止阀值	
	dmc_get_grant_error_protect_unit	读取龙门模式编码器位置跟随误差停止阈值	
软着陆/软启动功能	dmc_pmove_extern	实现 profile，pmove 整合，缩短指令时间，适用于高速场合	9.9 节
	dmc_t_pmove_extern	实现 profile，pmove 整合，缩短指令时间，并且实现软着陆	
	dmc_update_target_position_extern	强行改变指定轴的当前目标位置并且实现软着陆	
	dmc_t_pmove_extern_softstart_unit	实现 profile，pmove 整合，缩短指令时间，并且实现软启动	
IO 减速停	dmc_set_io_exactstop	配置 IO 触发减速停止参数	10.22 节
	dmc_set_dec_stop_dist	设置 IO 减速停止距离	
圆形区域限位	dmc_set_arc_zone_limit_config_unit	配置圆形区域限位参数	10.23 节
	dmc_get_arc_zone_limit_config_unit	获取配置的圆形区域限位参数	
	dmc_set_arc_zone_limit_enable	圆弧限位功能使能	
	dmc_get_arc_zone_limit_enable	获取圆弧限位功能的使能状态	
	dmc_get_arc_zone_limit_axis_status	查询相应轴的状态	
	dmc_set_arc_zone_limit_config	配置圆形区域限位参数	
	dmc_get_arc_zone_limit_config	获取配置的圆形区域限位参数	
看门狗功能	dmc_set_watchdog_action_event	设置看门狗触发响应事件	10.24 节
	dmc_get_watchdog_action_event	读取看门狗触发相应事件	
	dmc_set_watchdog_enable	使能看门狗保护机制	
	dmc_get_watchdog_enable	回读看门狗状态	
	dmc_reset_watchdog_timer	复位看门狗定时器	

函数分类	函数名	描 述	索引
椭圆插补及切向跟随	dmc_ellipse_move	启动椭圆插补运动	10.24 节
	dmc_set_tangent_follow	设置切向跟随参数	
	dmc_get_tangent_follow_param	读取切向跟随参数	
	dmc_disable_follow_move	取消指定坐标系的跟随运动	

附录 11　控制卡和主流驱动器电气接线图

1、控制卡与松下 Panasonic MSDA 系列驱动器接线

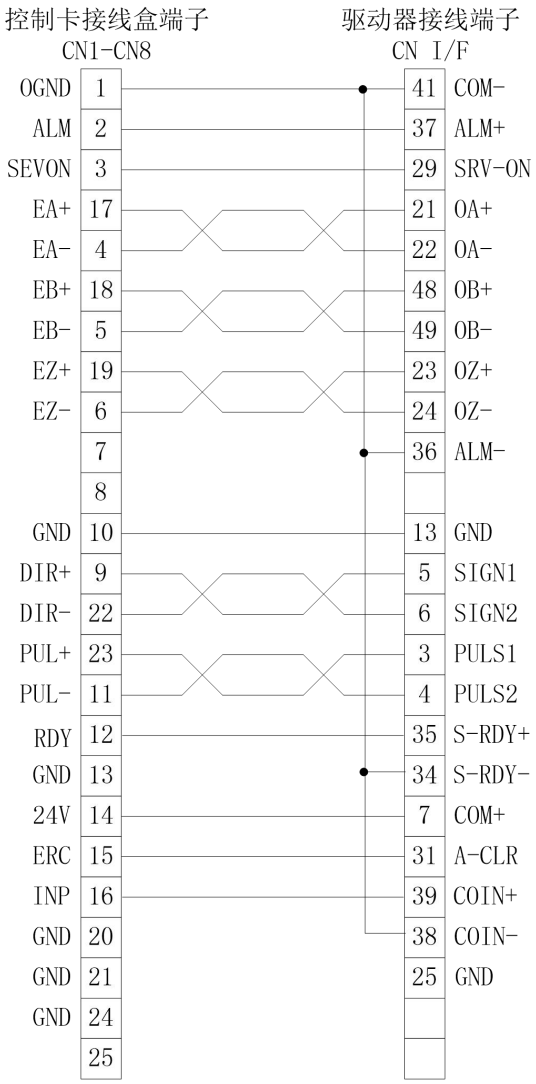


图 11.1　轴端口与松下 Panasonic MSDA 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

2、控制卡与安川 YASKAWA SGDM 系列驱动器接线

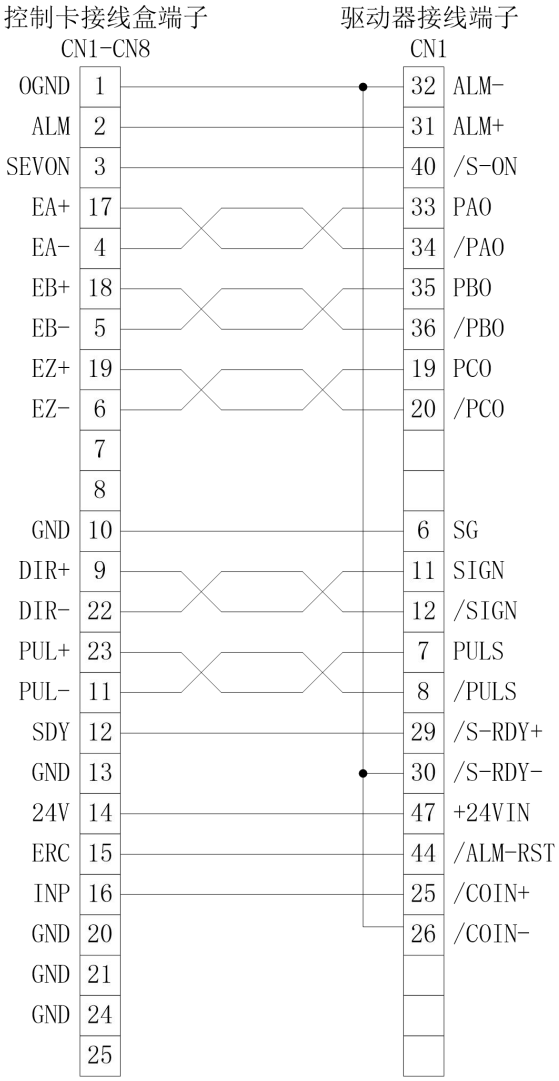


图 11.2　轴端口与安川 YASKAWA SGDM 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 SG 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 ALM-、/S-RDY-、/COIN-连接。

3、控制卡与安川 YASKAWA SGDE 系列驱动器接线

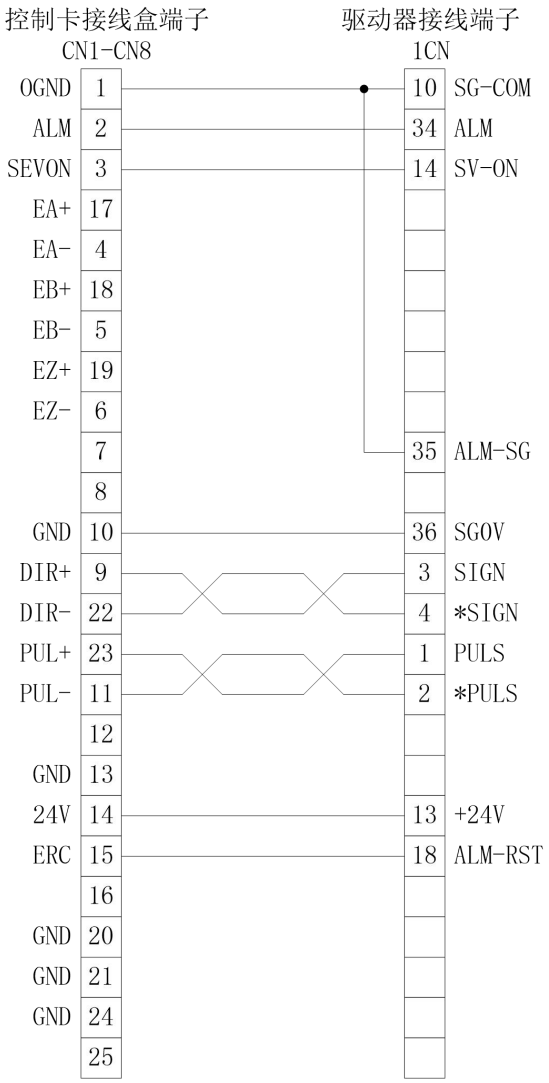


图 11.3　轴端口与安川 YASKAWA SGDE 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 SG 0V 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 SG-COM 连接。

4、控制卡与安川 YASKAWA SERVOPACK 系列驱动器接线

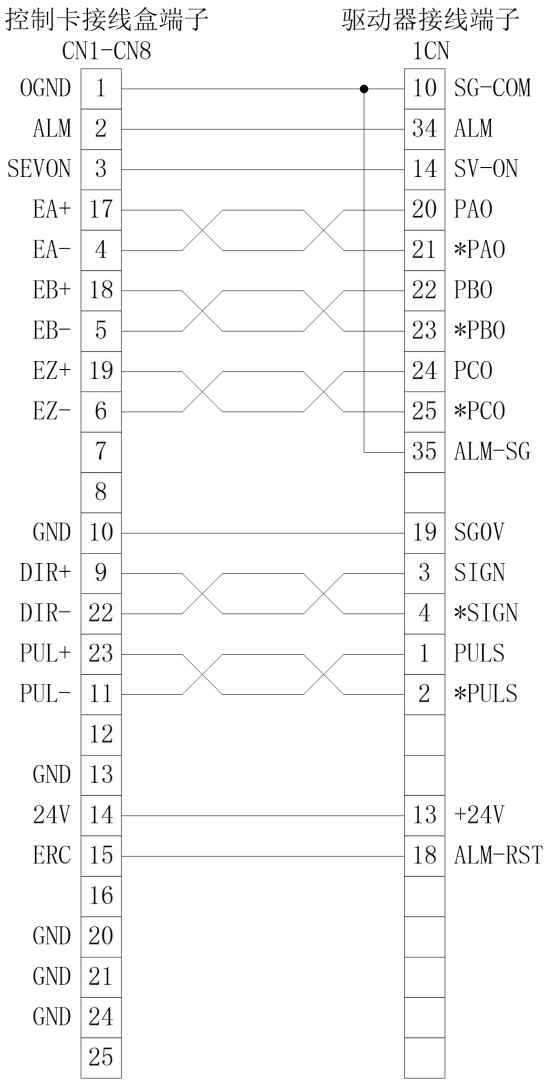


图 11.4　轴端口与安川 YASKAWA SERVOPACK　系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 SG 0V 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 SG-COM 连接。

5、控制卡与三菱 MELSERVO-J2-Super 系列驱动器接线

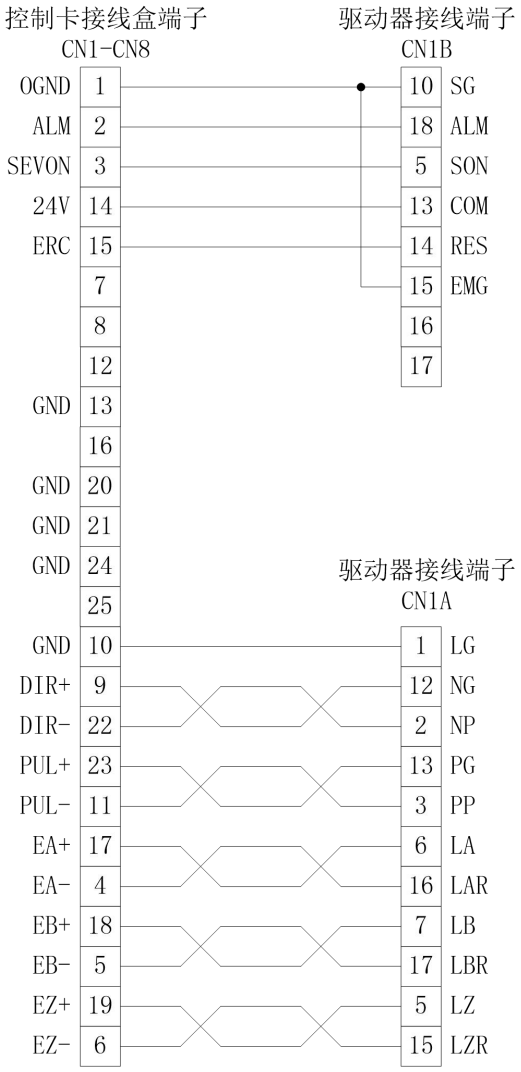
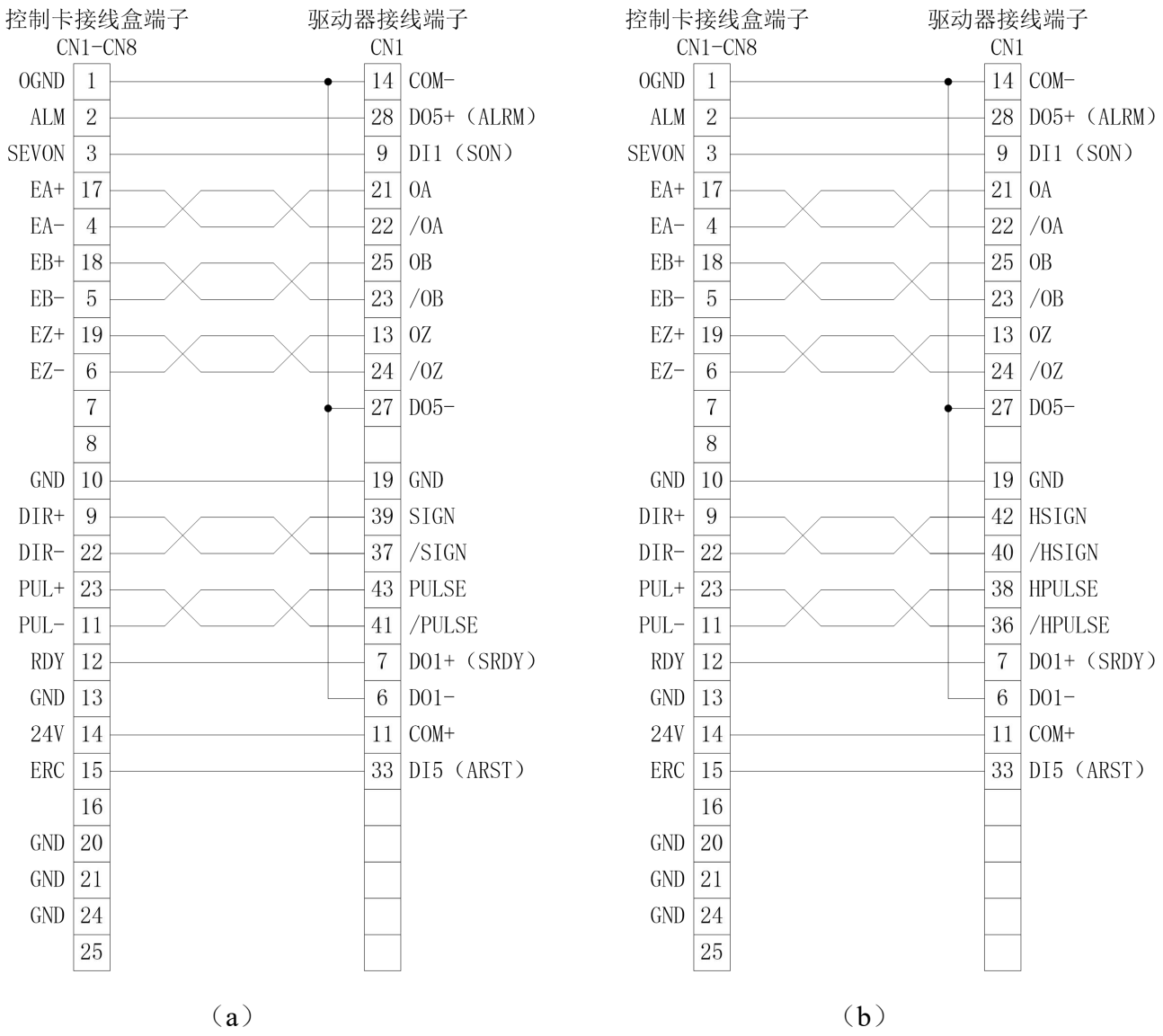


图 11.5　轴端口与三菱 MELSERVO-J2-Super 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 LG 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 SG 连接。

6、控制卡与台达 ASDA-B2 系列驱动器接线



（a） （b）

图 11.6　轴端口与台达 ASDA-B2 系列驱动器接线

注意：

1） 当脉冲频率在 500KHZ 以下时，PULSE+，PULSE-，DIR+，DIR-可接在驱动器端 43，41，39，37 引脚上，如上图（a）所示；

2） 当脉冲频率在 500KHZ~4MHZ 时，PULSE+，PULSE-，DIR+，DIR-可接在驱动器端 38，36，42，40 引脚上，如上图（b）所示；

3） 控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

7、控制卡与台达 ASDA-AB 系列驱动器接线

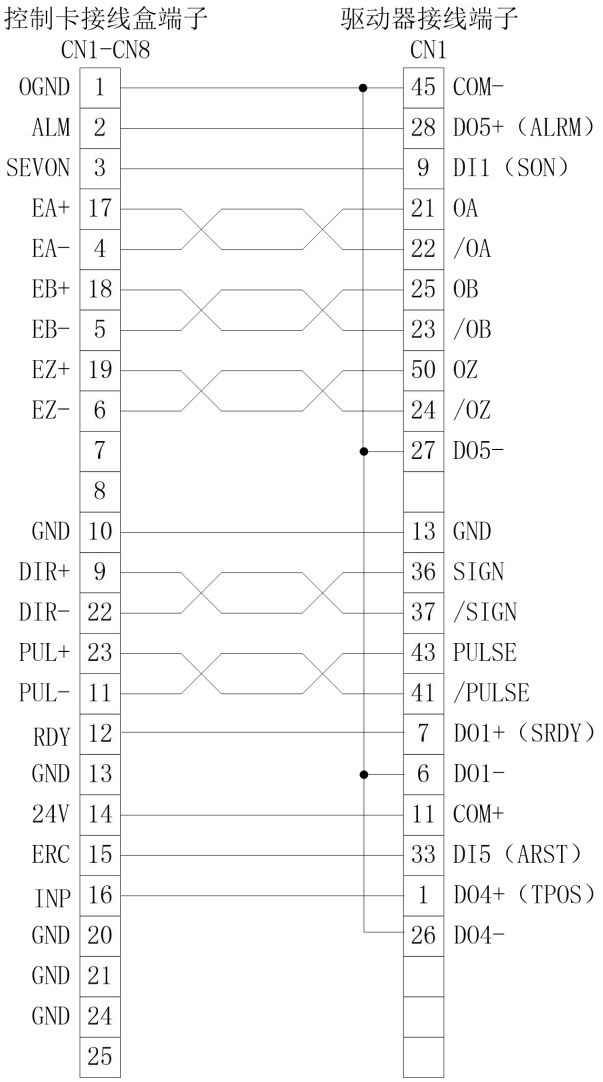


图 11.7　轴端口与台达 ASDA-AB 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

8、控制卡与汇川 IS500 系列驱动器接线

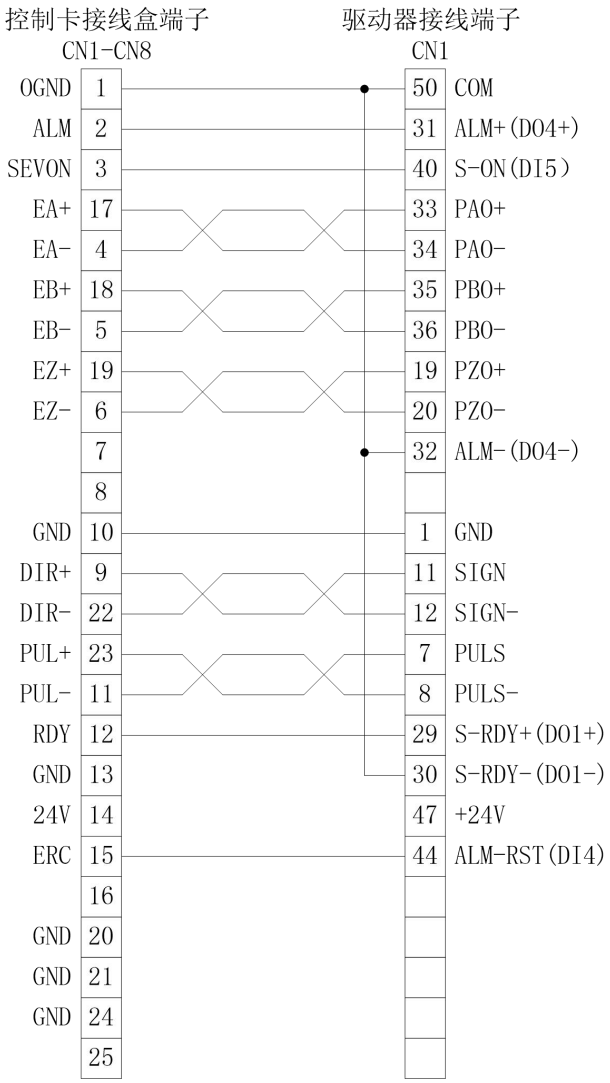
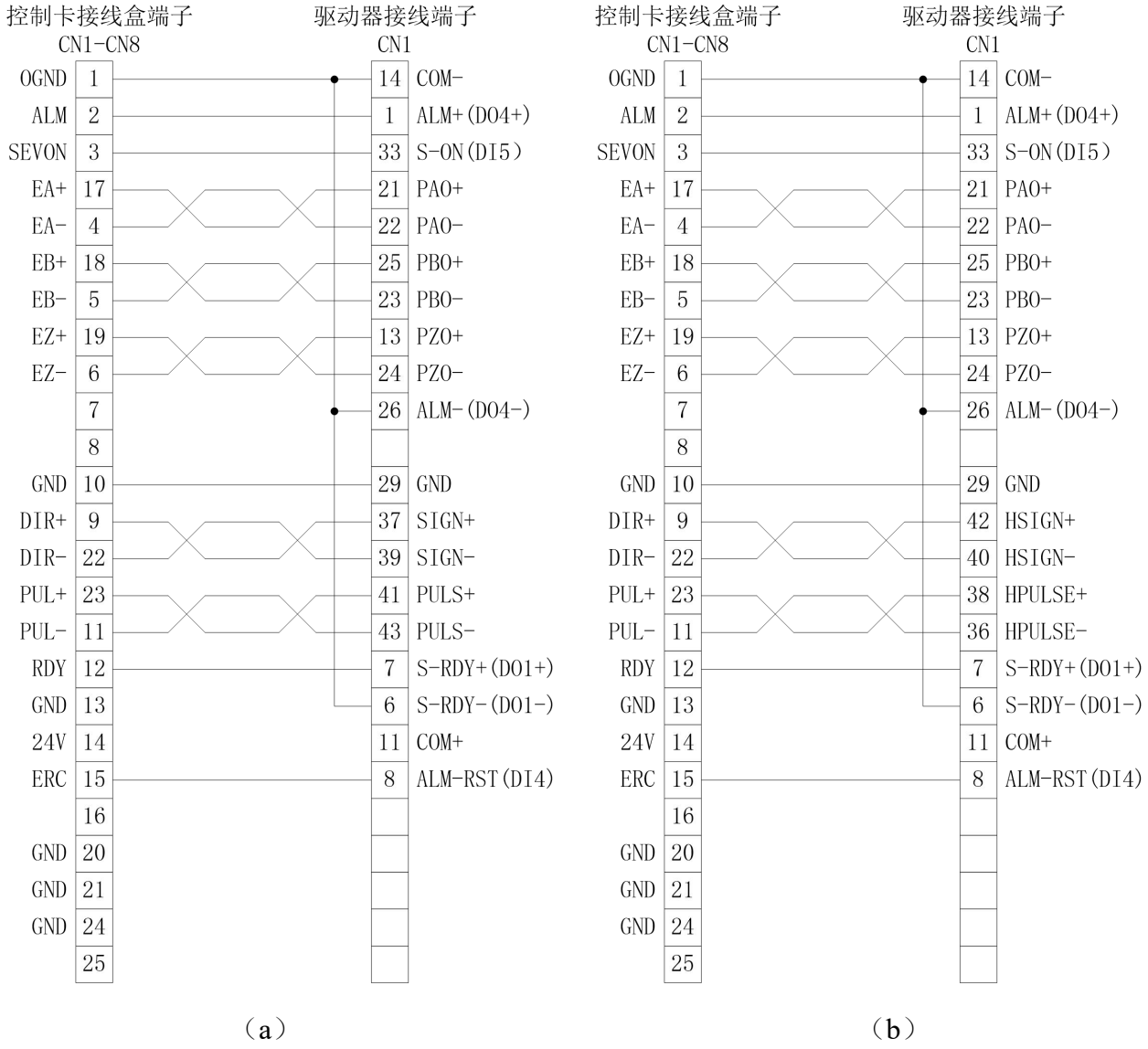


图 11.8　轴端口与汇川 IS500 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM 连接。

9、控制卡与汇川 IS650P 系列驱动器接线



（a）　　（b）

图 11.9　轴端口与汇川 IS650P 系列驱动器接线

注意：

1）当脉冲频率在 500KHZ 以下时，PULSE+，PULSE-，DIR+，DIR-可接在驱动器端 41，43，37，39 引脚上，如上图（a）所示；

2）当脉冲频率在 500KHZ~4MHZ 时，PULSE+，PULSE-，DIR+，DIR-可接在驱动器端 38，36，42，40 引脚上，如上图（b）所示；

3）控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

10、控制卡与雷赛 H2-506 驱动器接线

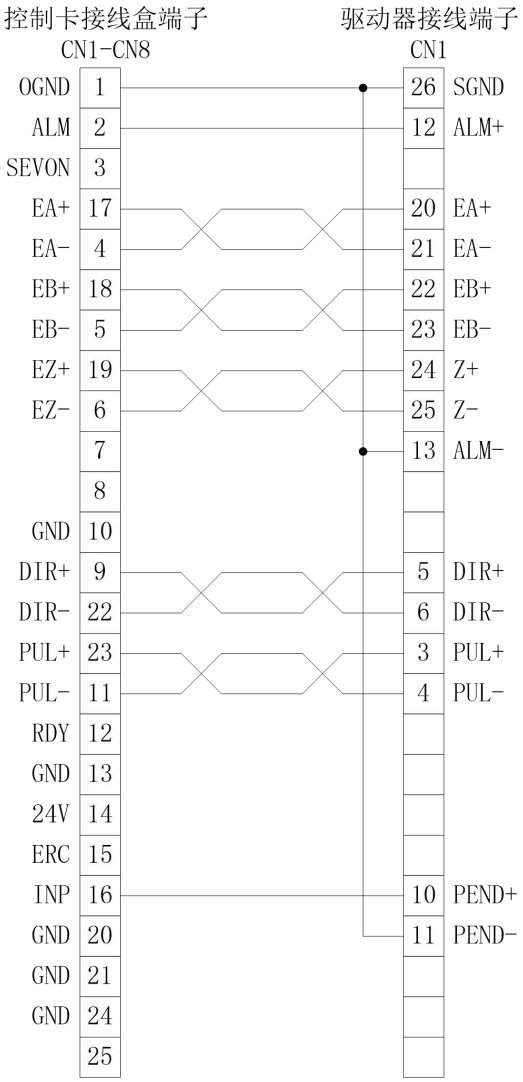


图 11.10　轴端口与雷赛 H2-506 驱动器接线

11、控制卡与雷赛 H2-758/1108/2206 驱动器接线

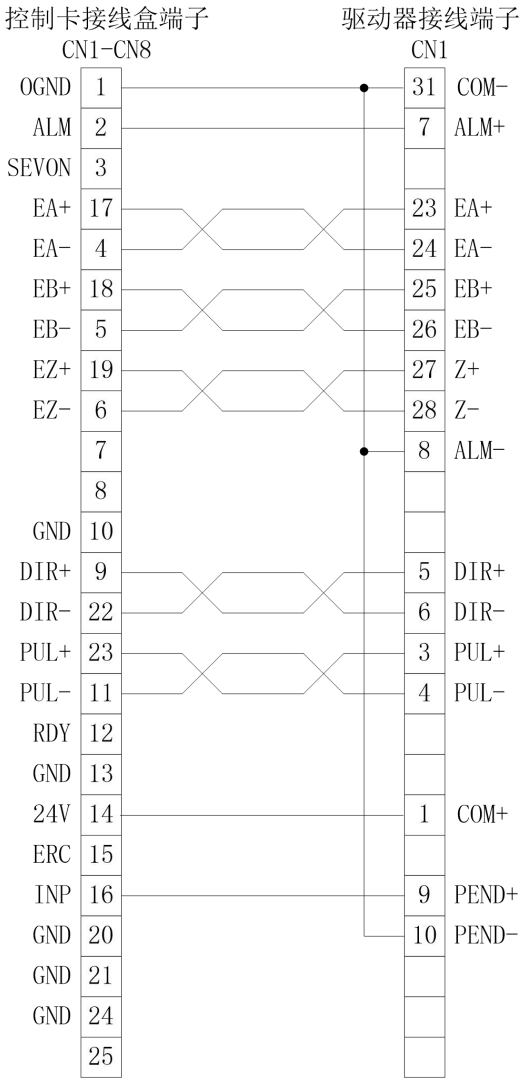


图 11.11　轴端口与雷赛 H2-758/1108/2206 驱动器接线

注意：控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

12、控制卡与雷赛 LD5 系列驱动器接线

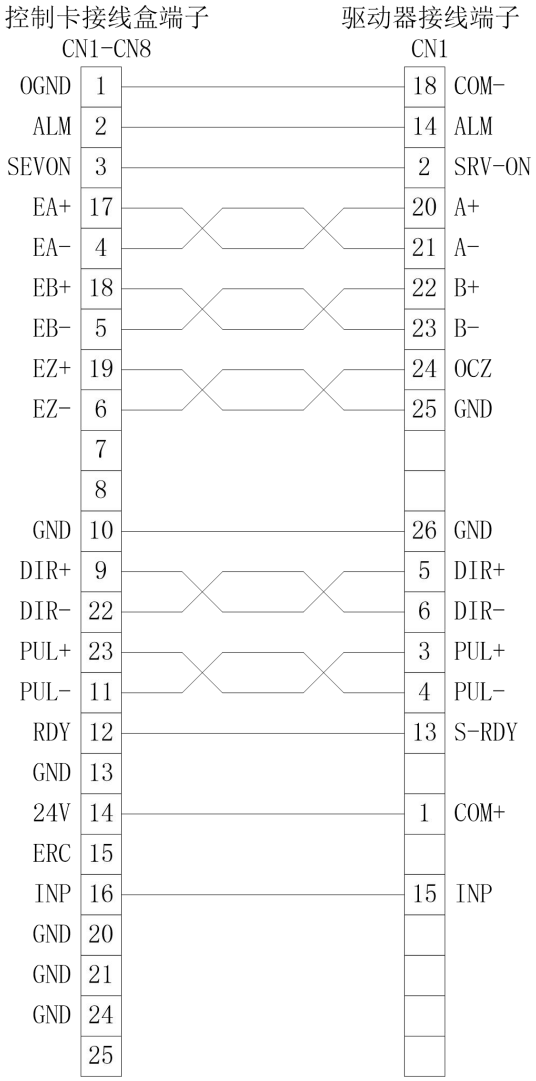


图 11.12　轴端口与雷赛 LD5 系列驱动器接线

注意：控制卡脉冲和编码器信号数字地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

13、控制卡与雷赛 L5Z 系列驱动器接线

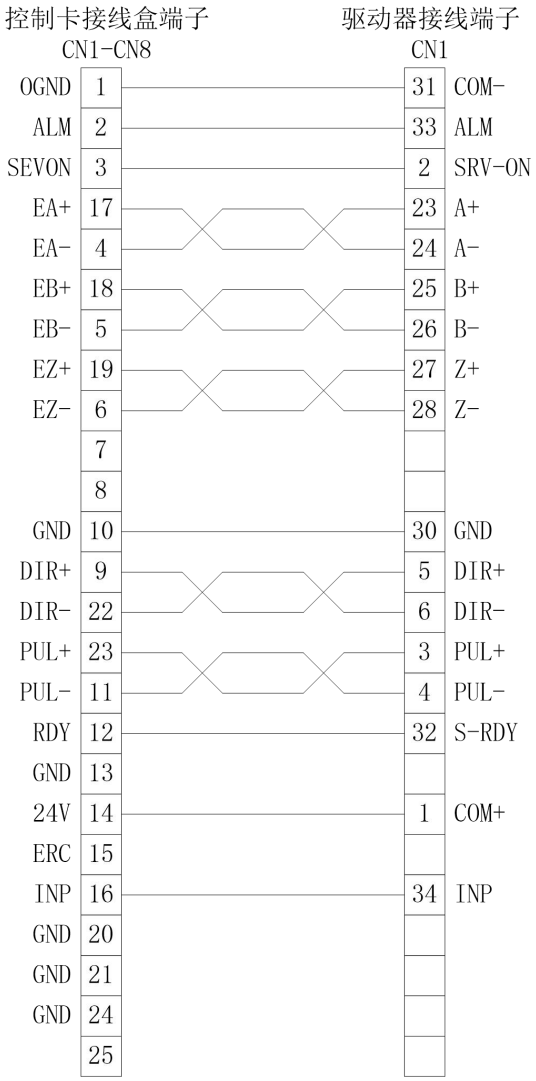


图 11.13　轴端口与雷赛 L5Z 系列驱动器接线

注意：控制卡脉冲和编码器信号地 GND 需与驱动器的地 GND 连接，控制卡的 24V 地 OGND 需与驱动器的 24V 地 COM-连接。

14、接线注意事项

1、布线建议

（1）不要将 110V/220V 交流电源电缆与直流 24V、IO 信号线缆、通信线缆等捆扎在一起，在空间允许范围内保持较远距离；

（2）不同类的电缆（如通讯信号线、电源线、数字 IO 信号线）布线时要分开，一般不能交叉重叠，当不可避免交叉时，应以直角交叉；

（3）高速 I/O、模拟量 I/O、现场总线、通信信号的电缆使用屏蔽线缆。

2、接线注意事项

（1）在安装及配线过程中，一定要确保外部电源处于关闭状态，防止触电及模块损坏；

（2）在连接端子处应扭绞导线，并以较短的长度接入端子，防止螺丝松动等情况下造成短路；

（3）电源接通后，24V 指示灯亮表明电源处于工作状态，如不亮，请考虑电源输入异常及模块故障可能。

3、接地处理

（1）ACC 系列接线盒，电源端子 24V 引脚与外部直流电源的正极连接。电源端子 OGND 引脚为 24V 电源输入地，接外部直流电源的负极。FG 为机壳地，应就近与机箱外壳相连。

（2）接地线应使用粗线，保证接地阻抗较小。

（3）ACC 系列接线盒上标记为 OGND 的引脚均为 24V 地，与电源端子 OGND 相通。标记为 GND 的引脚为 5V 信号地。

4、控制端口接线注意事项

（1）EtherCAT 和 Rtex 总线端口

主从站连接的网线应使用带屏蔽的五类双绞网线，RI45 接头带金属屏蔽。主从站的总线端口有输入和输出之分，不可接反。

（2）CAN 扩展接口

CAN 扩展总线上的所有从站波特率应保持一致，每个从站的节点号应不同。总线上的最后一个从站应接通终端电阻，中间的从站终端电阻不接通。

（3）编码器输入和脉冲输出端口

编码器输入和脉冲输出提供了差分接口，所以推荐用户以差分方式接线，差分信号使用屏蔽双绞线连接。差分信号两端数字地务必连通，即驱动器脉冲输入信号地与编码器输出信号地需与控制卡接线盒轴端口上的 GND 连接。

（4）手轮输入

控制卡手轮接口提供 5V 电源输出可作为手轮的电源输入，手轮的信号地需与手轮接口的 GND 地信号相连。

（5）数字输入输出

数字输入设备一端连接输入口一端连接到接线盒的 OGND（24V 地）端口。输出电路最大电流不能超过输出口最大工作电流，输出器件的地线需连接到接线盒的 OGND（24V 地）。输出口接感性负载时，需并联续流二极管。不允许将 24V 电源直接接到数字输出端口。

输入输出扩展电缆布线时，避免与动力线（高电压，大电流）等传输强干扰信号的电缆捆在一起，应该分开走线并且避免平行走线。

高速 IO 接口扩展电缆的总延长距离应该在 3.0m 以内。

（6）模拟量信号接口

模拟量信号连接时使用带屏蔽的线缆，固定线缆时不要将线缆与交流线缆、高压线缆等捆扎在一起，以减小噪声、电涌及感应的影响。

（7）PWM 信号接口

DMC3600、DMC3800、DMC3C00 卡的 PWM 信号与最后一个轴的脉冲输出信号引脚复用，PWM 输出 5V 信号，接线参照脉冲输出信号接线。

DMC3400A 的 PWM 输出功能需配合 ACC-X400B 接线盒使用，PWM 输出 24V 信号，PWM 接收设备地线与控制卡 OGND 连接。

附录 12　常见问题解决方法

出现问题	解决建议
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。
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。
板卡和驱动器电机连接后，发出脉冲时，电机不转动	板卡上的设置脉冲发送方式和驱动器的输入脉冲方式是否匹配，跳线 J1－J8 是否正确； 可以用 Motion 演示软件进行测试，观察脉冲计数等是否正常； 是否已经接上供给脉冲和方向的外部电源。
控制卡已经正常工作，正常发出脉冲，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 演示软件进行测试。 确保驱动器工作正常，没有出现报警。
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； 脉冲和方向信号输出端光电隔离电路中使用的限流电阻过大，工作电流偏小。
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。
能够控制电机，但工作时，回原点定位不准	检查屏蔽线是否接地； 原点信号开关是否工作正常； 所有编码信号和原点信号是否受到干扰。
限位信号不起作用	限位传感器工作不正常； 限位传感器信号受干扰； 应用程序紊乱。
不能读入编码器信号	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。
对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。
不能锁存编码器读数	检查触发源的接线； 检查函数的调用是否正确。
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。
数字输入信号不能读取	接线是否正常；检查函数调用。
数字输出信号不正常	接线是否正常；检查函数调用。
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：深圳市南山区沙河西路 3157 号南山智谷产业园 B 栋 15-20 楼
邮　编：518055
电　话：0755-26415968
传　真：0755-26417609
Email：info@szleadtech.com.cn
网　址：http://www.szleadtech.com.cn