ZMotion Basic 编程手册

Version 3.2.5

前言

正运动技术是一家专注于运动控制研究和通用运动控制产品研发的国家级高新技术企业，公司汇聚了来自华为、中兴等公司的优秀人才，在坚持自主创新的同时，积极联合各大高校致力于运动控制基础技术研究，是国内工控领域发展最快的企业之一，也是国内少有、完整掌握运动控制核心、技术和实时工控平台软件技术的企业 。

正运动技术的所有产品严格遵循华为的 IPD-CMM 开发流程，具备电信级的稳定性和可靠性，具有良好的软硬件兼容性和扩展性。正运动技术提供强大易用的 ZDevelop 开发环境，支持 ZBasic、ZPLC 梯形图、ZHMI 组态二次开发，并可混合编程，可实时仿真和在线跟踪 Debug，也支持各种上位机、各种操作系统调用不同编程语言的函数库来开发。

本手册编写的目的是为了更好的服务全国所有的客户，为客户提供更为全面的参考资料，我司致力于对产品的不断优化改进，让客户快速了解正运动的产品，产品手册也会持续更新。

本手册包含正运动 ZDevelop 软件的使用，指令详解，程序运行逻辑说明，运动缓冲原理，扩展模块，轴的应用，控制器介绍，控制器与其他元件的接线参考示范，可支持多种通信方式。除此之外还提供典型行业的应用例程供编程参考。

相关编程手册
ZMotion PC 函数库编程手册
ZMotion PLC 编程手册
ZMotion HMI 编程手册
ZMotion 机械手指令说明手册
ZDevelop 使用手册

以上资料和硬件手册均可从正运动官方网站下载，网址：www.zmotion.com.cn

版权说明

调试机器要注意安全！请务必在机器中设计有效的硬件或者机械安全保护装置，并在软件中加入出错处理程序，否则所造成的损失，正运动公司没有义务或责任对此负责。

目录

第一章 运动控制产品简介

1.1 运动控制产品概述

运动控制是一项采用计算机、电子元件、电动机、机械元件等对物体的位置和速度进行精密控制的技术，主要涉及步进电机、伺服电机的控制，运动控制器就是控制电动机运行方式的专用控制器，控制结构模式一般是：控制装置+驱动器+（步进或伺服）电机。

正运动技术的运动控制产品包括脉冲型独立式运动控制器、脉冲型网络运动控制卡、总线型独立式运动控制器、总线型 PCI 运动控制卡等，能满足各行各业的运动控制需求。

运动控制产品支持直线、圆弧、空间圆弧、椭圆、螺旋等插补运动功能，单个插补通道最多支持 16 轴，最多 16 个通道并行插补。支持速度前瞻、电子凸轮、电子齿轮、螺距补偿、固步跟踪、运动叠加、虚拟轴、脉冲闭环、硬件位置锁存、位置比较输出、连续插补、运动暂停等功能；部分运动控制产品内置了 SCARA、DELTA、六关节运动控制算法，可轻松满足 30 多种机械手独立或叠加应用。

总线型运动控制产品支持 EtherCAT、RTEX 等多种工业以太网运动控制总线，在性能和稳定性方面均处于领先地位，并可支持 EtherCAT 与 RTEX 总线混合使用，可实现多种总线轴和脉冲轴的混合插补运动，可支持总线轴硬件位置锁存和位置比较输出。

正运动技术坚持把产品质量置于首位，坚持以满足客户需求为中心、以创造价值为根本、以提升体验为追求，锲而不舍地为智能制造提供更有价值的运动控制产品、方案与服务。

1.2 正运动运动控制产品优点

正运动运动控制产品用统一的方式提供了包括并联机械手和六关节机械手在内的 30 多种机械手类型支持，可以单个控制器控制多个机械手，同时支持多个机械手叠加。机械手详细介绍参见“正运动机械手指令说明”文档。

国内首家推出同时支持 EtherCAT 总线和 RTEX 总线的双总线 PCI 控制卡与双总线运动控制器，EtherCAT 总线周期最快达 100 微秒，支持总线轴和脉冲轴混合插补，同时还支持总线轴硬件位置锁存与位置比较输出。

正运动产品具备齐全的运动控制功能和扩展性，正运动推出的 ZMC 系列（Zmotion Motion Controller 的简称）运动控制器功能强大、应用广泛，ZMC 系列控制器最多可支持 128 轴的运动控制，采用优化的网络通讯协议可以实现实时的运动控制。单个电脑可支持 256 个 ZMC 控制器同时连接。

正运动还为控制产品提供了强大的 ZDevelop 软件开发环境，操作简单易学。

运动控制器支持以太网、USB、CAN、RS485 串口、RS232 串口等通讯接口，通过 CAN 总线或 EtherCAT 可以连接各个扩展模块，从而扩展输入输出点或脉冲运动轴数（CAN 总线两端需要并接 120 欧姆的电阻）。

运动控制器大多支持 U 盘保存或读取数据（部分产品不具备）。

控制器具有如下优点：

1．硬件组成简单，把运动控制器连入 PC 就可组成系统；

2．除了 ZDevelop 软件，还支持各种操作系统与编程语言进行上位机软件开发（例如 C、C++、C#、VC、VB、LINUX、DELPHI、PYTHON，WINCE，MAC 等）；

3．运动控制软件的代码通用性和可移植性较好；

4．不需要太多培训工作，就可以进行开发，简单易学，支持多人同时开发。

1.3 主要功能描述

控制器的主要功能简介：

1．编程功能

项目			描述
任务			以指定条件执行 I/O 刷新和用户程序的功能，支持多任务同时进行，互不干扰，最大任务数在 ZDevelop 软件“控制器状态”查看
调试			支持断点调试和单步调试，以及任务运行状态查看
中断			支持三种类型的中断（外部中断、定时器中断、掉电中断）
设定监控窗口			监控变量常量，输入输出
编程语言种类			BASIC，PLC，HMI
在线命令			在线命令栏输入指令参数立即执行
通讯模块			允许通过触摸屏、计算机及其它控制器等进行访问
通讯方式	串口		232 串口，485 串口，USB 接口
	网口		通讯速度快，接线方便，支持 MODBUS_RTU 协议和自定义通讯
	CAN 总线		多用于 ZIO 扩展模块
数据类型	自定义数组	功能	集中相同数据类型的元素
		范围指定	明确指示只能取事先确定范围内的值
	自定义变量		整型，浮点型

2．单轴运动控制功能

项目		描述
控制模式		位置控制、速度控制、转矩控制
轴类型		电机轴、虚拟轴、编码器轴、虚拟编码器轴
可管理的位置		指令位置、反馈位置
位置控制	绝对位置定位	指定绝对坐标的目标位置，进行定位
	相对位置定位	指定自指令当前位置起的移动距离，进行定位
	周期位置控制	位置控制模式下，每个控制周期输出指令位置
速度控制	速度控制	位置控制模式下进行速度控制
	周期速度控制	速度控制模式下，每个控制周期输出速度指令
转矩控制	转矩控制	对电机进行转矩控制
同步控制	凸轮动作	使用指定的凸轮表开始凸轮动作
	齿轮动作	设定主轴和从轴间的齿轮比，连接轴号，执行齿轮动作
手动操作	手动运动	在编程软件中使用“手动运动”，使轴执行动作
控制辅助	轴错误复位	解除轴异常
	原点回零	驱动电机，使用正负限位开关信号、原点开关信号确定机械原点
	原点回零模式	通过选择回零模式的参数，控制回零方式
	强制停止	取消当前运动和缓冲运动，使轴减速停止
	立即停止	切断脉冲发送，使轴立即停止
	速度设定	变更轴的目标速度
	当前位置变更	将轴的指令当前位置和反馈当前位置变更为任意值
	锁存功能	根据外部信号触发的发生记录轴的位置
	位置监视	判断轴的指令位置或反馈当前位置是否在指定范围内
	轴间偏差监视	监控指定轴的指令位置或反馈位置的差异量是否超过了容许值
	位置矫正	使指令当前位置和反馈当前位置的偏差归零
	转矩限制	通过伺服驱动器的转矩限制功能的有效/无效切换和转矩限制值的设定，限制输出转矩
	指令位置补偿	对动作中的轴进行位置补偿
	起始速度	设定开始轴动作时的初速度
	强制速度	自定义速度的 SP 运动的强制速度

3．其他运动控制功能

项目		描述
多轴	平面直线插补	指定绝对/相对位置坐标，进行两轴的直线插补
	平面圆弧插补	指定绝对/相对位置坐标，进行两轴的圆弧插补
	平面椭圆插补	指定绝对/相对位置坐标，进行两轴的椭圆插补
	空间直线插补	指定绝对/相对位置坐标，三个轴或多轴在三维空间进行插补
	空间圆弧插补	指定绝对/相对位置坐标，三个轴或多轴在三维空间进行插补
	螺旋插补	指定绝对/相对运动轨迹，两个轴进行圆弧插补，第三轴螺旋
	样条插补	TABLE 表存入绝对/相对位置样条控制点，可分为二轴平面样条插补、三轴空间样条插补、三轴空间加螺旋插补
	多轴周期位置控制	位置控制模式下，每个控制周期输出位置指令
	轴分组	指定哪几个轴为一组，可取消分组
	连续插补	开启插补动作连续，可关闭
	轴组强制停止	使插补动作中的所有轴减速停止
	轴组立即停止	使插补动作中的所有轴立即停止
	动态变速	变更插补动作中的合成目标速度
	位置获取	获取插补运动当前位置和编码器反馈位置
凸轮	凸轮表生成	通过输入参数指定的凸轮属性和凸轮节点生成凸轮表
	保存凸轮表	将通过输入参数指定的凸轮表保存到 TABLE
参数	指令设定	使用指令改写部分轴或轴组参数
	轴参数的变更	通过编程软件查看，在线命令实时变更轴参数
	自动加减速控制	可根据指令设定轴或轴组动作时的加减速曲线。
	变更加减速度	加减速动作中也可变更加减速度
	到位检查	判断是否运动到设定位置，可查看剩余运动距离
	停止方法选择	多个停止指令，且不同的参数代表不同的方式
	软限位	限制轴的动作范围
	位置偏差	轴的当前位置与编码器反馈位置的偏差
	支持外部输入信号	将立即停止开关信号、正负方向限位开关信号、原点开关信号由外部输入信号控制，信号可翻转

1.4 应用场景

正运动技术的运动控制产品经过众多合作伙伴多年的开发应用，产品广泛应用于 3C 电子半导体、点胶设备、激光加工、印刷包装、特种机床、机器人、舞台娱乐、医疗器械等自动化领域。

电子产品加工行业有贴片机、点胶机、印刷电路板钻孔机、绕线机、焊接机、上下料机械手、紧螺钉机等设备。

纺织服装行业有经编机、染色机、印花机、工业缝纫机、绣花机、切布机、精梳机、捻线机、制鞋机等设备。

印刷包装行业有自动吹瓶机、制袋机、模切机、烫金机、开箱机、装箱机，贴标机、自动颗粒包装机、袋装包装机、报纸印刷机、凹版印刷机等。

哪里有自动化设备，哪里就有运动控制，正运动控制器以其优异的性能、完善的功能为各行各业均能提供优秀的解决方案。

1.5 控制器接口

这里用 ZMC420SCAN 总线型运动控制器为例展开说明，更多控制器型号和规格参见“选型指南”章节或《正运动控制器硬件手册》。

ZMC420SCAN 总线型运动控制器支持 EtherCAT/RTEX 总线连接，最多支持 20 个轴的运动控制，支持不同类型轴(4 脉冲轴+EtherCAT 总线轴/RTEX 总线轴/编码器轴/振镜轴/虚拟轴)的混合插补，支持直线插补、任意圆弧插补、空间圆弧、螺旋插补、电子凸轮、电子齿轮、同步跟随、虚拟轴设置等，采用优化的网络通讯协议可以实现实时的运动控制。

ZMC 运动控制器支持以太网、USB、CAN、485、232 等通讯接口，通过 CAN 总线或 EtherCAT 总线可以连接相应扩展模块，从而扩展输入输出点数或脉冲轴数（CAN 总线两端需要并接 120 欧姆的电阻），扩展方法参见“模块扩展”。

EtherCAT 总线采用标准以太网 RJ45 接口。

产品特点：

脉冲轴输出模式：方向/脉冲或双脉冲。

振镜轴输出模式：XY2-100 协议。

支持编码器位置测量，可以配置为手轮输入模式。

每轴最大输出脉冲频率 10MHz。

通过 EtherCAT 总线，最多可扩展到 4096 个隔离输入口或输出口。

轴正负限位信号和原点信号可以随意配置为任何输入口。

输出口最大输出电流可达 300mA，另带 2 个 500mA 大电流输出口，可直接驱动部分电磁阀。

支持电子凸轮、电子齿轮、位置锁存、同步跟随、虚拟轴等功能。

支持硬件比较输出(HW_PSWITCH2)，硬件定时器，运动中精准输出。

支持脉冲闭环，螺距补偿等功能。

支持 ZBasic 多文件多任务编程。

多种程序加密手段，保护客户的知识产权。

具有掉电检测、掉电存储功能。

功能参数：

项目	描述
基本轴数	20
最多扩展轴数	20
基本轴类型	EtherCAT 总线轴/RTEX 总线轴/4 个差分脉冲轴（输出口可配置 6 个单端脉冲轴）/4 个振镜轴
内部 IO 数	24 进 12 出（带过流保护），另外每个差分脉冲轴有 1 进 1 出
最多扩展 IO 数	4096 进 4096 出
PWM 数	12（输出频率 1M）
内部 AD/DA 数	2 路 AD，2 路 DA（0-10V）
最多扩展 AD/DA	256 路 AD，128 路 DA
脉冲位数	64
编码器位数	64

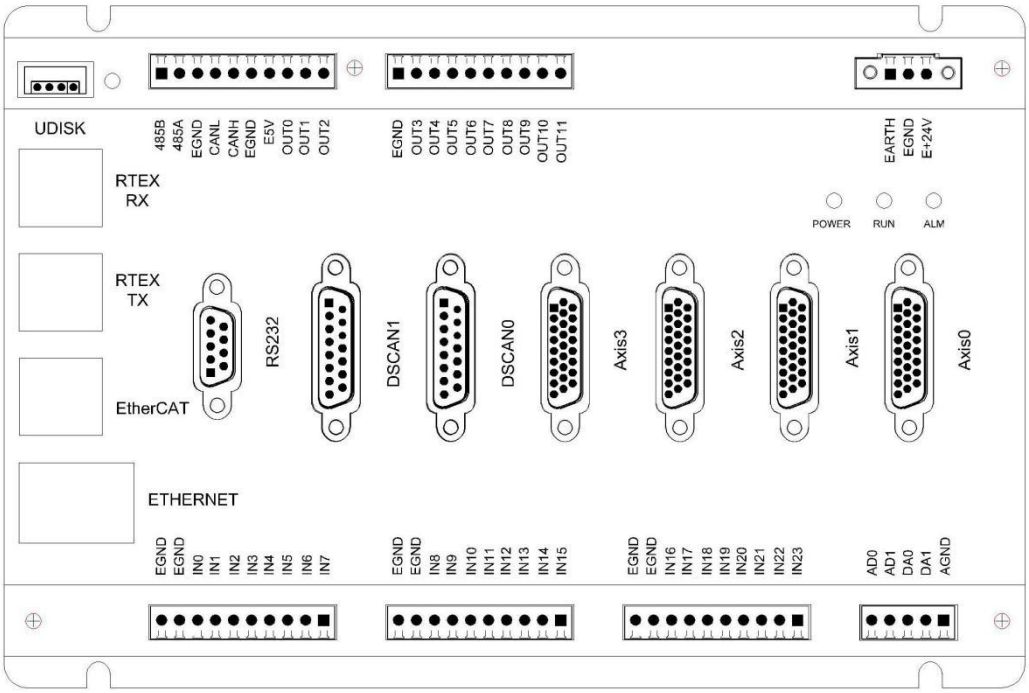
速度加速度位数	64
脉冲最高速率	10MHz
每轴运动缓冲数	4096
数组空间	640000
程序空间	8000Kbyte
Flash 空间	256MByte
电源输入	24V 直流输入（功耗 10W 内，不用风扇散热），IO 口负载没计算在内
通讯接口	RS232，RS485，以太网，U 盘，CAN，RTEX，EtherCAT
外形尺寸	216*144mm

轴 6/7/8/9/可以令 ATYPE=3 配置为编码器，从而映射到轴 0-3 的编码器，轴 4/5/6/7 可以令 ATYPE=21 配置为振镜轴。

单端脉冲轴需要通过 AXIS_ADDRESS 来强制配置本地轴的方式来使用。任意轴号都可以映射到本地轴，需要通过 AXIS_ADDRESS 来强制配置。

通用输出带 12 个 PWM 功能或单端脉冲轴的功能，通过 ATYPE 和 PWM 来设置切换。ZMC4 系列支持正运动 XPLC 的功能，可以通过网络来做组态显示。

ZMC420SCAN 控制器外观如下图所示：



输入/输出接口信号

输入/输出信号可分为两类，数字信号（对应 IN/OUT）和模拟信号（对应 AD/DA）。

1．数字信号

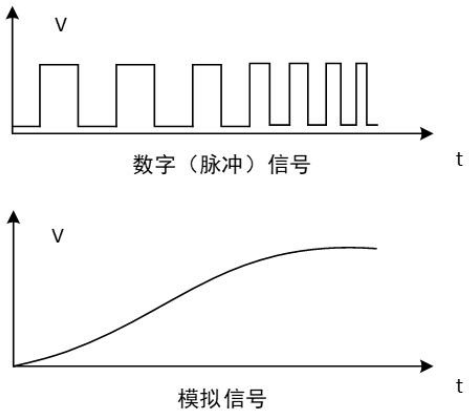
在数值上和时间上均不连续的信号称为数字信号，数字信号自变量是离散的。

在计算机中，数字信号的大小常用有限位的二进制数表示，逻辑“1”代表高电平，逻辑“0”代表低电平。传输过程中如果混入了干扰信号，只要干扰信号不超过阈值，就可以还原原来的信号，即使超过阈值也可以用一定的编码技术，将出错的信号检测出来并加以修正，与模拟信号相比，数字信号在传输过程中有更好的抗干扰能力，传输距离更远且失真幅度小。

脉冲信号属于数字信号，只有高电平和低电平，两个比较重要的参数为脉冲频率和脉冲数，有时候也

会使用到脉冲信号的占空比、上升沿和下降沿的信息。

脉冲信号之间的时间间隔称为周期，而将在单位时间内所产生的脉冲个数称为频率。



2．模拟信号

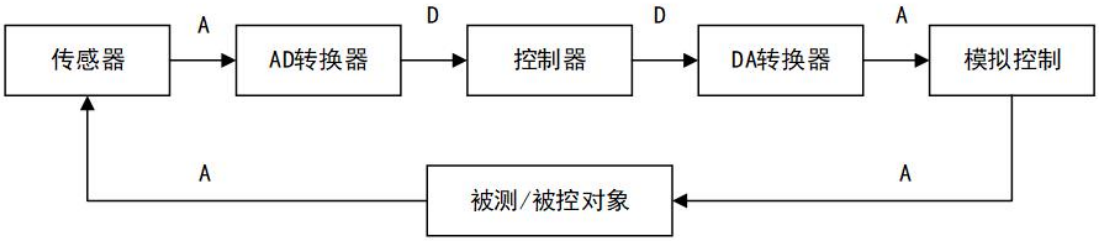
在数值上和时间上连续变化的信号称为模拟信号，其信号的幅度，或频率，或相位随时间作连续变化，或在一段连续的时间间隔内，其代表信息的特征量可以在任意瞬间呈现为任意数值的信号。从自然界感知的大部分物理量都是模拟性质的，如速度、压力、温度、声音、重量以及位置等都是最常见的物理量。常见的模拟信号波形有正弦波、调幅波、阻尼震荡波、指数衰减波等。

模拟信号的处理可以通过模拟电路实现，由于不存在量化误差，它可以对自然界物理量的真实值进行尽可能逼近的描述。但是模拟信号经过多次复制或长距离传输的情况下，随机噪声的影响会非常显著，噪声效应会使信号产生有损，有损后的模拟信号几乎不可能被再次还原。

如用模拟信号控制伺服电机，模拟信号的电压值和电机输出的转速成正比；若用脉冲信号控制步进电机，脉冲信号的频率和步进电机的输出转速成正比，步进电机输出的旋转角度大小由脉冲数决定。

3．模数转换和数模转换

AD 是模拟量到数字量的转换，依靠的是模数转换器(Analog to Digital Converter)，简称 ADC。DA 是数字量到模拟量的转换，依靠的是数模转换器(Digital to Analog Converter)，简称 DAC。



控制器内部带两路 AD 和两路 DA，电压范围均为 0-10V，需要用到更多 AD/DA 可使用扩展模块扩展。

1）模数转换 AD

数字信号是在模拟信号的基础上进行采样、保持、量化和编码得到的。

采样：把输入的模拟信号按适当的时间间隔（采样脉冲）得到各个时刻的样本值；

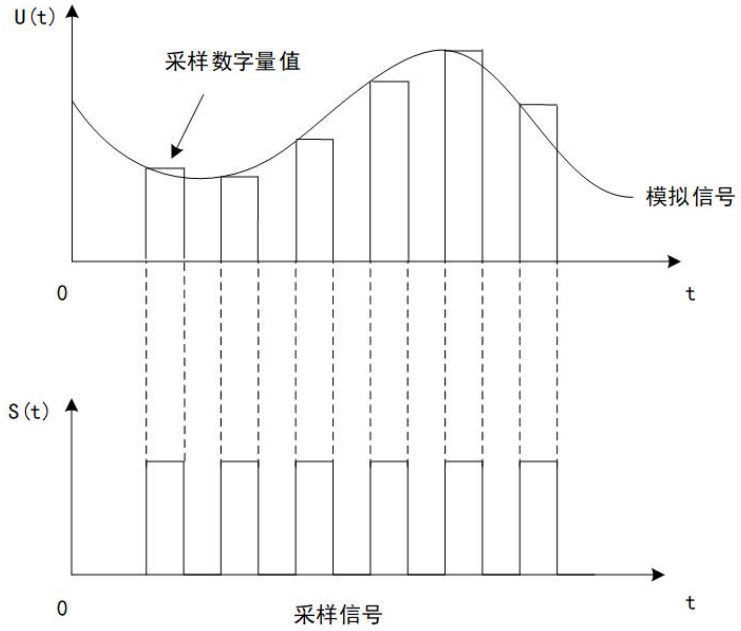
保持：每次采样后，将采样值保持一段时间；

量化：把经采样的各个时刻的值用二进制来表示，将模拟信号量程分成许多离散量级，并确定输入信号所属的量级；

编码：把量化生成的二进制数排列形成脉冲序列，编码是对每一量级分配唯一的数字码，并确定与输入信号相对应的代码。最普通的码制是二进制，它有 2 的 n 次方个量级（n 为位数）。

数字信号是模拟信号的近似，既然近似就不可能一模一样，采样点越多，得到的数字信号就越接近模拟信号。数字信号便于直接数字处理和运算，可以将任何信号转换成数字量再交给计算机进行处理，处理完成后再按一定精度将数字量转换成可接受的模拟量。

模数转换原理图如下：



2）数模转换 DA

数模转换器 DAC 主要由数字寄存器、模拟电子开关、位权网络、求和运算放大器和基准电压源（或恒流源）组成。用存于数字寄存器的数字量的各位数码，分别控制对应位的模拟电子开关，数字量是用代码按数位组合起来表示的，每位代码都有一定的位权，为了将数字量转换成模拟量，必须将每一位的代码按其位权的大小转换成相应的模拟量，然后将这些模拟量相加，即可得到与数字量成正比的总模拟量，从而实现了数字—模拟转换。

最大输入数字量对应最大电压输出值，使数码为 1 的位在位权网络上产生与其位权成正比的电压值，根据数字量大小由运算放大器对各数码加权值求和，并转换成电压值，即得到数字量对应的模拟量。

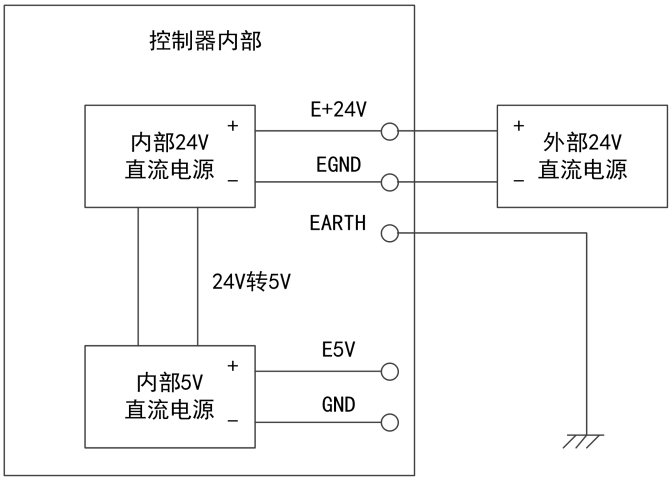
例如：二进制数字量 $D_3D_2D_1D_0$，比例系数 K=模拟量最大值/数字量最大值对应十进制数，此时模拟量 $=K（D_3\times2^3+D_2\times2^2+D_1\times2^1+D_0\times2^0）$

1.5.1 电源接口

ZMC420SCAN 系列控制器采用单电源供电，ZMC0-2 系列和 ZIO 扩展卡采用双电源供电（单电源供电只需接入外部 24V 直流主电源；双电源供电不仅需要主电源，还需要给 IO 供电，双电源供电外部主电源为一路，IO 电源再采用另一路外部电源供电，不得采用同一个电源，以防信号干扰）。

控制器的 5V 轴和编码器接口采用内部电源，IO 采用外部电源，ZIO 扩展模块扩展出来的轴和编码器采用外部电源。

电源接口如下图，内部 5V 直流电源由 24V 电源转换得到，部分控制器型号没有 5V 输出接口。



针脚号	名称	说明
1	E+24V	外部电源 24V 输入
2	EGND	外部电源地
3	FG/EARTH	安规地/屏蔽层

1.5.2 通讯接口

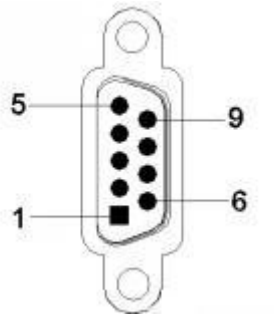
CAN 总线上连接多个控制器时，需要在最两边控制器的 CANL 与 CANH 端并接 120 欧姆的电阻。

针脚号	名称	说明
1	485B	485-
2	485A	485+
3	EGND	外部电源地
4	CANL	CAN 差分数据-
5	CANH	CAN 差分数据+

ZMC4 系列的通讯接口采用外部 24V 电源，与其他控制器或触摸屏连接时要留意。

1.5.3 RS232 接口

控制器使用 RS232 串口与电脑连接时需要使用双母头的 2.3 交叉线。

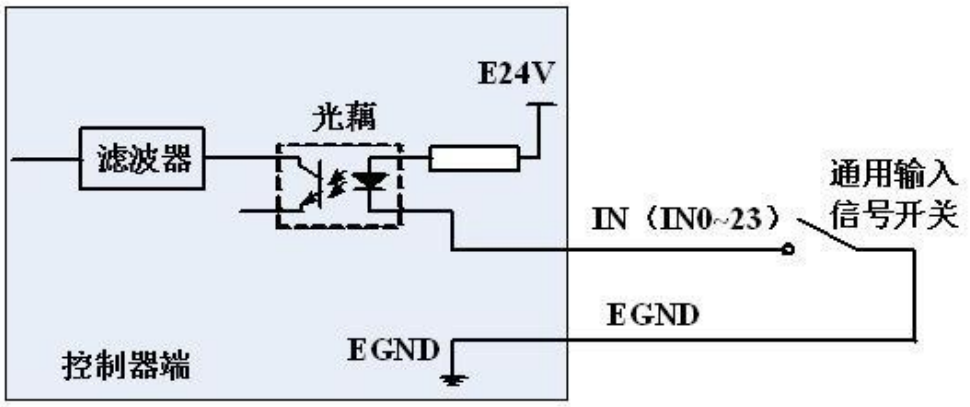


其 9PIN 引脚定义如下：

针脚号	名称	说明
2	RXD	接收数据引脚
3	TXD	发送数据引脚
5	EGND	外部电源地
9	E5V	外部电源 5V 输出，可用于对文本屏供电

1.5.4 通用输入口

每轴信号里面另有 1 个通用输入口，见脉冲轴接口描述。



输入/输出口的特殊功能参见对应型号控制器手册确定，不同型号有所差异。

输入 0 与输入 1 同时具有锁存输入 A 与锁存输入 B 的功能，输入 2 与输入 3 同时具有锁存输入 C 与锁存输入 D 的功能。

单端编码器轴 6-9 需要通过 AXIS_ADDRESS 强制配置本地轴才能使用。

针脚号	名称	说明	缺省或建议功能
1	EGND	外部电源地	
2	EGND	外部电源地	
3	IN0	输入 0	锁存 A,EA6
4	IN1	输入 1	锁存 B,EB6
5	IN2	输入 2	锁存 C,EZ6
6	IN3	输入 3	锁存 D,EA7
7	IN4	输入 4	EB7
8	IN5	输入 5	EZ7
9	IN6	输入 6	EA8
10	IN7	输入 7	EB8
11	EGND	外部电源地	
12	EGND	外部电源地	
13	IN8	输入 8	EZ8
14	IN9	输入 9	EA9

15	IN10	输入 10	EB9
16	IN11	输入 11	EZ9
17	IN12	输入 12	
18	IN13	输入 13	
19	IN14	输入 14	
20	IN15	输入 15	
21	EGND	外部电源地	
22	EGND	外部电源地	
23	IN16	输入 16	
24	IN17	输入 17	
25	IN18	输入 18	
26	IN19	输入 19	
27	IN20	输入 20	
28	IN21	输入 21	
29	IN22	输入 22	
30	IN23	输入 23	

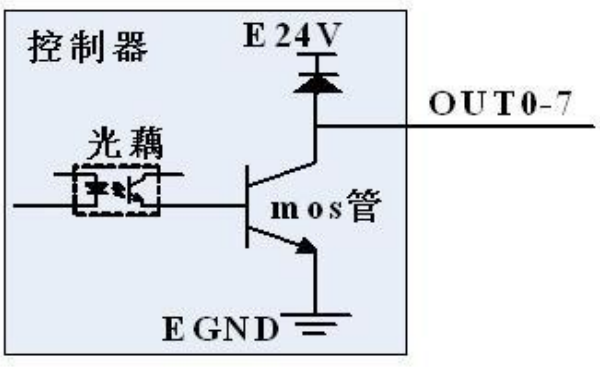
光耦和滤波器的作用：

运动控制器上大多为数字电路，所用集成电路芯片多由 5V 直流电源供电。自动化设备中大多数电气元件，如：传感器，继电器，电磁阀，指示灯，触摸屏等，多由 24V 直流电源供电，为了不让 24V 直流电源回路上的干扰信号影响运动控制器的正常工作，在运动控制器的输入输出信号接口中，均采用光电隔离电路将控制器的电源回路隔离，避免干扰信号通过信号线、电源线进入运动控制器。

为了进一步提高运动控制器的可靠性，在输入口加设低通滤波器，禁止高频干扰信号进入运动控制器。

1.5.5 通用输出口

每轴信号里面另有 1 个不带电流放大的通用输出口，见轴接口描述。



OUT0-11 具有 PWM 的功能，当 PWM 关闭时为通用输出，单端脉冲轴 6-11 需要通过 AXIS_ADDRESS 强制配置本地轴才能使用。

ZMC420SCAN 的 PWM 输出受正常输出功能的控制，只有输出 ON 的时候 PWM 才能实际输出，这样可以用来控制激光能量。

ZMC420SCAN 的输出 0-7 的具有精准输出的功能，普通的输出操作需要等待一个控制器周期才可执行，

而精准输出操作，可在电机发出一个脉冲内响应，大大提高了输出效率。每个输出口的精准输出功能相互独立，一个控制器周期内触发一次精准输出，再次触发需要等到下一个控制器周期。

指令 MOVEOP_DELAY 与 AXIS_ZSET 可以用来设置每个 MOVE_OP 指令是否使用精准输出以及精准输出的延时，从而实现激光的 PSO（位置同步输出）控制。

针脚号	名称	说明
1	EGND	外部电源地
2	E5V	外部 5V 电源输出
3	OUT0	输出 0，PWM0，PUL6
4	OUT1	输出 1，PWM1，DIR6
5	OUT2	输出 2，PWM2，PUL7
6	EGND	外部电源地
7	OUT3	输出 3，PWM3，DIR7
8	OUT4	输出 4，PWM4，PUL8
9	OUT5	输出 5，PWM5，DIR8
10	OUT6	输出 6，PWM6，PUL9
11	OUT7	输出 7，PWM7，DIR9
12	OUT8	输出 8，PWM8，PUL10
13	OUT9	输出 9，PWM9，DIR10
14	OUT10	输出 10，PWM10，PUL11
15	OUT11	输出 11，PWM11，DIR11

1.5.6 AD/DA 信号

ZMC420SCAN 内部 DA 采用了内部电源。

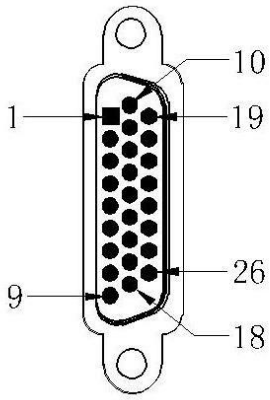
针脚号	名称	说明
1	AD0	0-10V 模拟输入口 0
2	AD1	0-10V 模拟输入口 1
3	DA0	0-10V 模拟输出口 0
4	DA1	0-10V 模拟输出口 1
5	AGND	内部电源模拟地

1.5.7 U 盘接口信号

针脚号	名称	说明
1	V	U 盘+5V 电源输出
2	D-	差分数据 D-
3	D+	差分数据 D+
4	GND	内部电源地

1.5.8 脉冲轴接口信号

提供了 0V 和+5V 输出，可以为编码器提供 5V 电源。轴使用前，要通过 ATYPE 参数来配置轴的使用方式。ATYPE 需要与轴的类型适配。

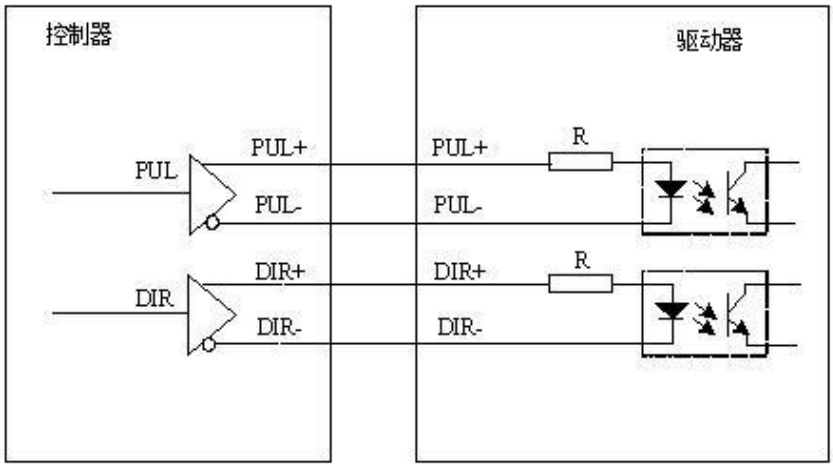


针脚号	信号	说明
1	EGND	外部电源地
2	IN24-29/ALM	通用输入，建议做驱动报警
3	OUT12-17/ENABLE	通用输出，建议驱动使能
4	EA-	编码器输入
5	EB-	编码器输入
6	EZ-	编码器输入
7	+5V	内部电源+5v 输出
8	备用	备用
9	DIR+	伺服或步进方向输出
10	GND	内部电源地
11	PUL-	伺服或步进脉冲输出
12	备用	备用
13	GND	内部电源地
14	OVCC	外部 E+24V 输出（建议仅供伺服 IO）
15	备用	备用
16	备用	备用
17	EA+	编码器输入
18	EB+	编码器输入
19	EZ+	编码器输入
20	GND	内部电源地
21	GND	内部电源地
22	DIR-	伺服或步进方向输出
23	PUL+	伺服或步进脉冲输出
24	GND	内部电源地
25	备用	备用

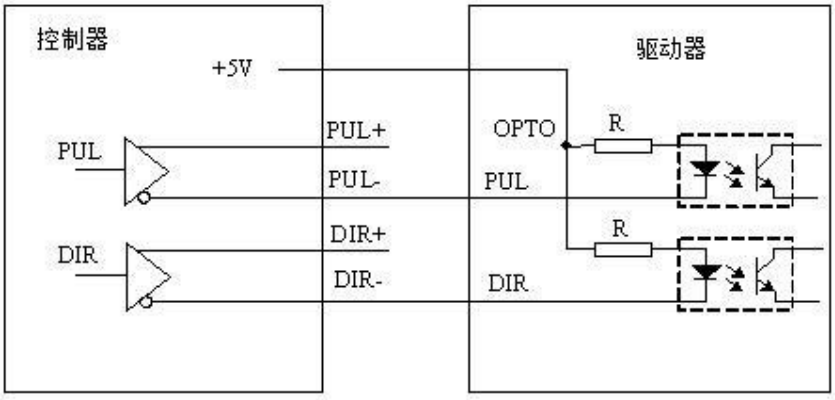
26	备用	备用

ZMC420SCAN 控制器脉冲轴接口和驱动器之间有两种连接方式：低速差分连接，高速差分连接。两种接线方式参见附录“控制器与驱动器接线参考”

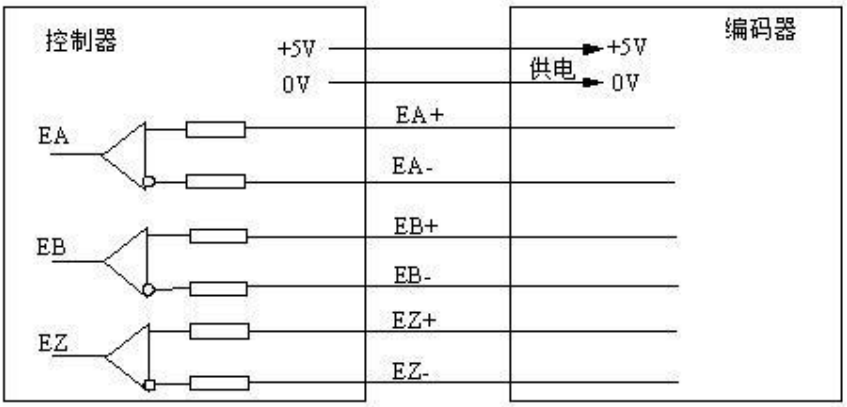
1．差分连接方式：



2．单端连接方式（共阳极）：



3．编码器连接方式：



ZMC 控制器每个脉冲轴都提供了独立的编码器输入端口，编码器输入信号包括 EA+/-、EB+/-、EZ+/-，每个轴都有三对差分的 A 相、B 相和 Z 索引信号，EA 和 EB 用来进行位置计数，EZ 可用作原点信号。

差分连接和单端连接的区别

单端接线有两种方法，分别为共阴极和共阳极接法，共阴极接法为判断信号线与地的电压差，共阳极接法为判断信号与电源电压的电压差。控制器的脉冲口和编码器仅支持共阳极接法，下文仅针对共阳极接法展开说明。

单端接线和差分接线都是在两根铜导线上进行传输，单端接线是判断信号与+5V 电源的电压差，一根接信号负端，另一根接+5V 电源。单端接线的输入信号均以共同的+5V 电压为基准。

差分接线是判断两个信号线的电压差。对于差分输入，当信号受干扰时，差分的两线会同时受影响，但电压差变化不大，而单端接线的一线变化时，由于电源电压不变，所以电压差变化相对较大，因此差分接线方式的抗干扰性能比单端接线方式好。

1.6 控制器的使用

1.6.1 安全提示

1．注意事项

控制器集成度高，设计尺寸小巧轻便，易于安装，用户可以有效地利用空间。可以将控制器安装在面板或标准导轨上，并且可以选择水平或垂直安装方式。

作为安装布置系统中各种设备的基本规则，将控制器等低压逻辑型设备与热辐射、高压和电噪声隔离开，远离粉尘、腐蚀性气体、水、油、化学品等场所。

在面板上配置控制器的布局时，由于控制器长时间运行会产生发热现象，应考虑将控制器布置在较凉爽区域，少暴露在高温环境中会延长电子设备的使用寿命，温度过高的环境可能会导致控制器无法正常使用。

安装时还要考虑面板中设备的布线，避免将低压信号线和通信电缆铺设在具有交流动力线和高能量快速开关直流线的槽中。

四周留出足够的空隙以便控制器冷却和接线，控制器可通过自然对流冷却、为保证适当冷却，在设备上方和下方必须留出一定的空隙。

2．警告

控制器是弱电设备，需要将控制器安装在外壳、控制柜或电控室内等不易触碰的地方，避免非操作人员接触。

机械运行时为了您的人身安全请勿靠近。

请勿对本产品进行分解、修理或改装。

在外部采用控制电路构成紧急停止电路、联锁电路、限制电路等于安全保护相关的电路。

安装或拆卸控制器时应先将控制系统的电源全部断开，避免发生触电或意外设备操作导致不必要的损失。

不遵守这些以上要求可能会导致人员重伤和财产损失，正运动技术公司不承担相应风险与责任。

3．接线要求

使用螺丝将控制器固定，防止产品跌落或遭受异常冲击导致使用故障。

为保证通讯稳定，控制器通讯电缆应选用带屏蔽层的高性能电缆。

采用 24V 直流电源给控制器供电，另 IO 口需要单独供电，和控制器电源分开。

控制器网络的各个部件为了安全起见，确保控制器和相关设备的所有公共端和接地连接在同一个点接地，该点应该直接连接到系统的大地接地。确定接地点时，应考虑安全接地要求和保护性中断装置的正常运行。

完成所有接线工作后再给控制电路通电，请勿在带电的情况下操作。

所有线路连接应尽可能短，能减少干扰，保证通信质量。

1.6.2 使用参考步骤

1.6.2.1 准备工作

软件：安装 ZDevelop 编程软件或控制器支持上位机的其他编程软件（例如 C、C++、C#、VC、VB、LINUX、DELPHI、PYTHON，WINCE，MAC 等）。

设备：控制器、计算机、24V 直流电源、驱动器、步进电机或伺服电机、接线端子、IO 设备、扩展模块等根据需求选择。

连接线：控制器与计算机通讯的连接线，控制器与驱动器轴接口的连接线，IO 接口、电源接口等的连接线。

1.6.2.2 程序设计

1．系统架构设计

根据功能需求选择所需元件和连接线，熟悉与该功能相关的控制指令的使用方法，设计系统软件的整体构成，包括变量设计、任务设计、程序功能设计等。

2．软件设定与编程

使用 ZDevelop 软件按步骤 1 的设计编写程序，软件快速使用方法参见本文“新建工程”小节，或打开 ZDevelop 软件菜单栏“帮助”-“ZDevelop 帮助”查看软件具备的各种功能用法介绍，编写任务和程序模块，进行程序模拟调试。

编程需要设置的参数：BASE 选择轴号，UNITS 脉冲当量，SPEED 轴速度，ACCEL 轴加速度，DECEL 轴减速度，ATYPE 轴类型，各类插补运动指令。

若使用 EtherCAT 总线或 RTEX 总线连接驱动器，编程时需要进行总线初始化操作（参见“总线初始化”例程）。若需要扩展模块，例如扩展轴或 IO 点数，编程时需要对扩展的轴资源进行轴映射（参见“轴映射”）；扩展的 IO 资源需要进行 IO 映射，ZCAN 扩展使用扩展板上的拨码开关设置扩展 IO 的编号（参见“CAN 总线通讯”章节），EtherCAT 总线扩展使用 NODE_IO 指令设置扩展的 IO 编号，通过 IO 编号即可访问扩展资源。

1.6.2.3 安装与接线

安装各种单元，将各单元与控制器采用合适的连接线连接。

计算机与控制器接线：可以采用串口通信或网口通讯，串口通信连接控制器的 RS232 串口，网口通讯连接控制器的 EtherNET 网口。

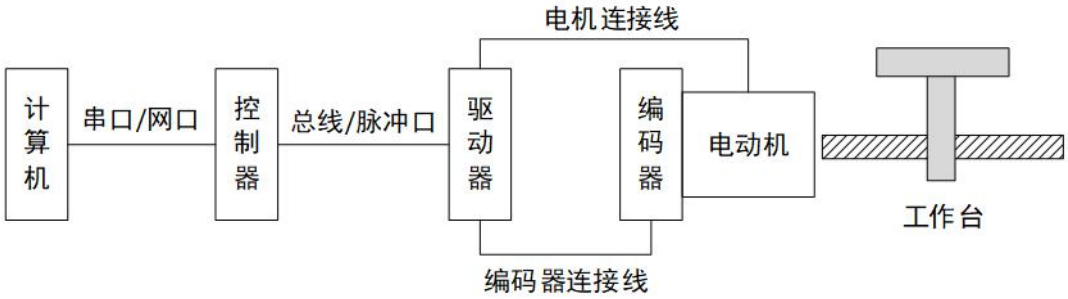
驱动器与控制器接线：驱动器可连接控制器的脉冲轴接口、EtherCAT 总线接口、RTEX 总线接口。驱动器接脉冲口时参见附录VI “控制器与驱动器接线参考”，使用总线连接只需用电缆直接插入对应的 EtherCAT 或 RTEX 接口。

电源接线：将+24V 直流电源正极接控制器电源模块的 24V 接口，负极接 GND 接口，驱动器接 220V 交流电，IO 设备接在控制器对应的 IO 接口上，部分型号控制器 IO 需要采用 24V 直流 IO 电源单独供电。

扩展模块接线：扩展 IO 或脉冲轴，可使用 CAN 总线扩展或 EtherCAT 总线扩展。详细方法参见“模块

扩展”章节

接线参考：

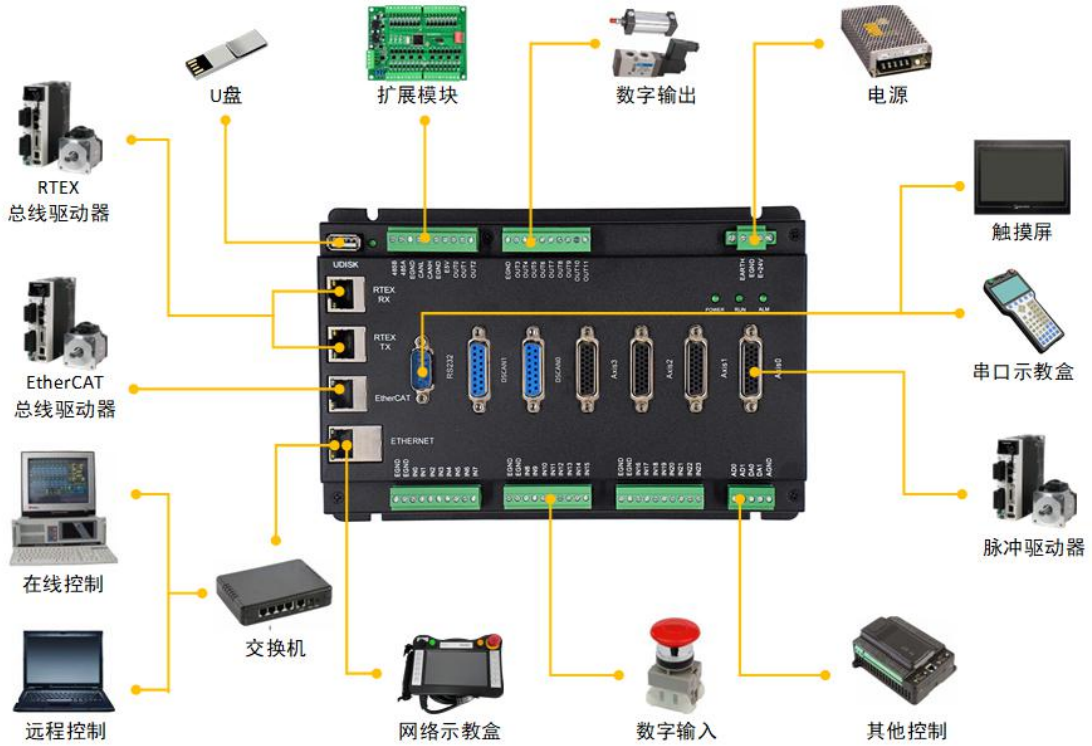


1.6.2.4 试运行

请确认接线无误后上电，将调试好的程序下载到控制器，开始试运行。使用示波器窗口或其他参数监控窗口确认动作是否符合要求。

1.6.3 控制系统接线参考

如下图所示控制器与其他元件参考接线网络示范。



各个部分元件经由不同通信接口接入控制器，使用 CAN 总线或 EtherCAT 总线接扩展模块，驱动器可使用脉冲轴接口、EtherCAT 总线接口和 RTEX 总线接口，部分型号输出口支持配置为单端脉冲轴。串口与网口用于与上位机进行通信或网络触摸屏通信。

多轴运动控制系统以运动控制器和伺服系统为核心元件，运动控制器负责运动控制命令解码、各个轴的位置控制、彼此之间的相对运动、加减速轮廓控制等，其关键作用在于降低整个运动控制系统的路径误

差；伺服驱动器负责伺服电机的位置控制，主要在于降低伺服轴的跟随误差。

1.6.3.1 伺服系统简介

伺服系统又称随动系统，是用来精确地跟随或复现某个给定过程的反馈控制系统，伺服系统使物体的位置、状态等输入被控量能够跟随给定值的变化而变化的自动控制系统。

最基本的伺服系统包括执行元件（伺服电机等）、检测反馈元件（传感装置，如光栅尺、编码器等）和驱动器，除此之外，还需要计算机和控制器，用以给驱动器发送指令。

对伺服系统基本要求：

抗干扰能力强：在各种扰动作用时，系统输出动态变化小，恢复时间快，震荡次数少。

稳定性好：伺服系统在给定输入和外界干扰下，能在短暂的过渡过程后，达到新的平衡状态或恢复到原来的平衡状态。

动态响应快：动态响应是伺服系统的重要动态性能指标，要求系统对给定的跟随速度足够快、超调小。

精度高：伺服系统的精度是指输出量跟随给定值的精确程度，对于精密加工的场合显得尤其重要。

1．伺服电机

伺服电机是指在伺服系统中控制机械元件运转的电机，伺服电机内部带有编码器，能实时反馈运动数据给伺服驱动器。

伺服电机可使控制速度，位置精度非常准确，可以将电压信号转化为转矩和转速以驱动控制对象。伺服电机转子转速受输入信号控制，并能快速反应，在自动控制系统中，用作执行元件，可把所收到的电信号转换成电机轴上的角位移或角速度输出。

2．伺服驱动器

伺服驱动器是用来控制伺服电机的一种控制器，其作用类似于变频器作用于普通交流马达，属于伺服系统的一部分，它的主要作用是按照控制命令的要求，对功率进行放大、变换与调控等处理之后传递给伺服电机。

3．传感装置

传感装置最常用的是编码器，一般伺服电机内自带编码器，用以反馈采集的实际运动数据给驱动器，从而实现运动控制闭环。

4．伺服驱动器对伺服电机的控制

伺服驱动器一般通过位置、速度和力矩三种方式对伺服电机进行控制，实现高精度的传动系统定位。

1）位置控制

位置控制模式一般是通过外部输入的脉冲的频率来确定转动速度的大小，通过脉冲的个数来确定转动的角度，也有些伺服可以直接对速度和位移进行赋值，位置模式可以对速度和位置都有很严格的控制。

2）速度模式

速度模式通过模拟量的输入或脉冲频率的输入进行转动速度的控制，在有上位控制装置的外环 PID 控制时速度模式也可以进行定位，但必须把电机的位置信号或负载的位置信号给反馈上位机做运算用。

3）转矩控制

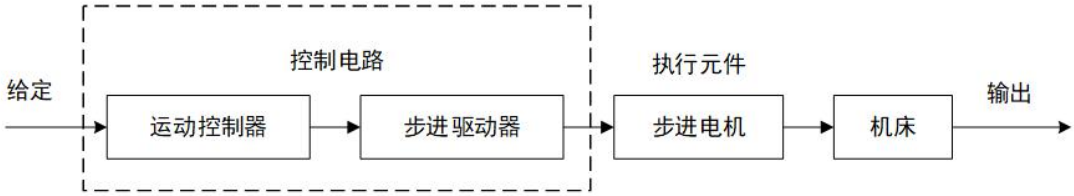
转矩控制方式是通过外部模拟量的输入或直接的地址的赋值来设定电机轴对外的输出转矩的大小，可以通过即时的改变模拟量的设定来改变设定的力矩大小。

1.6.3.2 运动控制系统分类

运动控制系统按照所用电机划分，可分为步进式伺服系统、直流电机伺服系统、交流电机伺服系统；按照控制方式划分，可分为开环伺服系统、闭环伺服系统和半闭环伺服系统。

1．开环控制系统

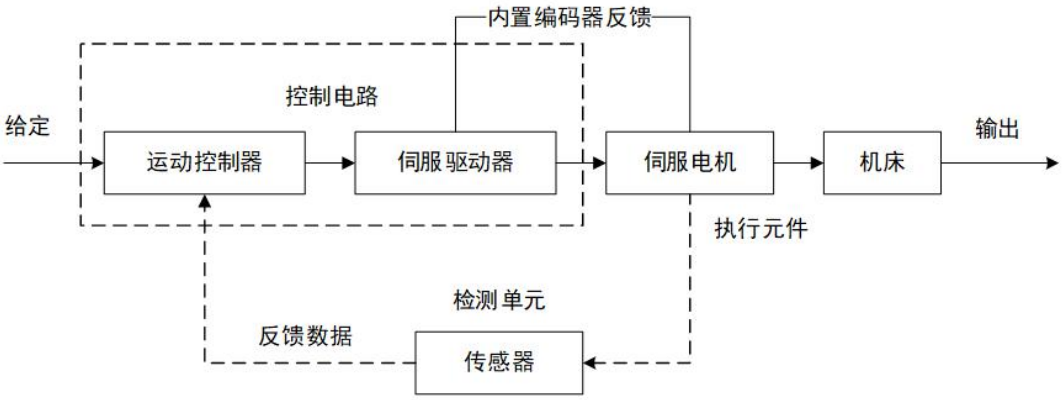
开环控制系统是数控机床中最简单的控制系统，由控制电路、执行元件和机床三大部分组成，没有反馈装置，执行元件一般为步进电机，通常称以步进电机作为执行元件的开环系统为步进式伺服系统，控制电路的主要任务是将指令脉冲转化为驱动执行元件所需要的信号。



其控制原理较简单，开环进给伺服系统的精度较低，速度也受到步进电动机性能的限制。但由于其结构简单，易于调整，在精度要求不太高的场合中得到较广泛的应用。

2．半闭环控制系统

采用旋转型角度测量元件（如编码器）和伺服电动机按照反馈控制原理构成的位置伺服系统，称作半闭环控制系统，半闭环控制系统的检测装置有两种安装方式：一种是把角位移检测装置安装在丝杠末端；另一种是把角位移检测装置安装在电动机轴端。



通常把安装在丝杠上的检测元件组成的伺服系统称为半闭环系统，把安装在工作台上的检测元件组成的伺服系统称为闭环系统。半闭环控制系统的精度比闭环要差一些，但驱动功率大，快速响应好，因此适用于各种数控机床。对半闭环控制系统的机械误差，可以在数控装置中通过间隙补偿和螺距误差补偿来减小系统误差。

第二章 编程基础

本手册以 BASIC 编程语言为例进行详细说明，用 PC 编程的客户需要获取更多资料请参考正运动“Zmotion PC 函数库编程手册”。

2.1 ZDevelop 编程软件使用

2.1.1 软件简介

ZDevelop 编程软件支持 ZBasic 语言，ZBasic 是 ZMotion 运动控制器所使用的 Basic 编程语言，提供所有标准程序语法、变量、数组、条件判断、循环及数学运算。此扩展的 Basic 指令和函数能提供广泛的运动控制功能，例如单轴运动、多轴的同步和插补运动，同时还有对数字和模拟 I/O 的控制。

ZBasic 支持以下功能：

1．自定义 SUB 过程，可以把一些通用的功能编写为自定义 SUB 过程，方便程序编写和修改。

2．G 代码形式的 SUB 过程，支持 G00、G01、G02、G03、G04、G90、G92 等常用指令。

3．支持全局的变量（GLOBAL）、数组和 SUB 过程；文件模块变量、数组和 SUB 过程；以及局部变量（LOCAL）。

4．中断程序（掉电中断、外部中断、定时器中断），例如掉电中断，通过掉电中断时保存数据，可以使得掉电的状态得到恢复。

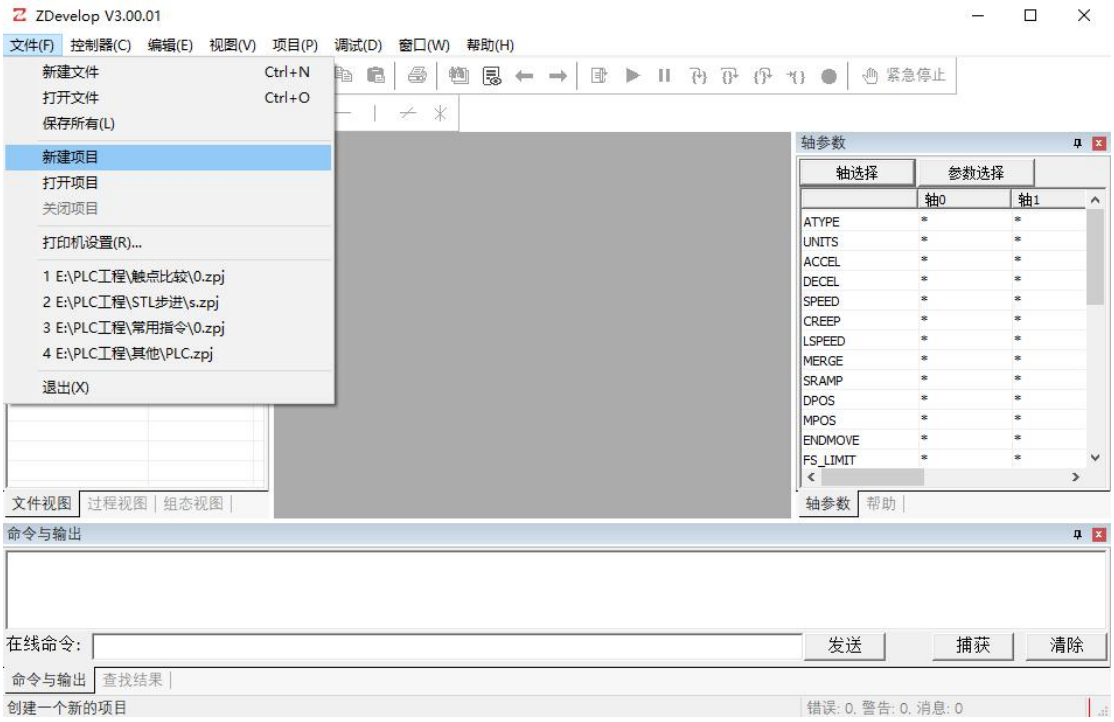
ZBasic 具有实时多任务的特性，多个 ZBasic 程序可以同时构建并多任务实时运行，使得复杂的应用变得简单易行。

通过 PC 在线发送 BASIC 命令也可以实现同样的效果，控制器内置的 BASIC 程序和 PC 在线 BASIC 命令可以同时多任务运行。

2.1.2 新建工程

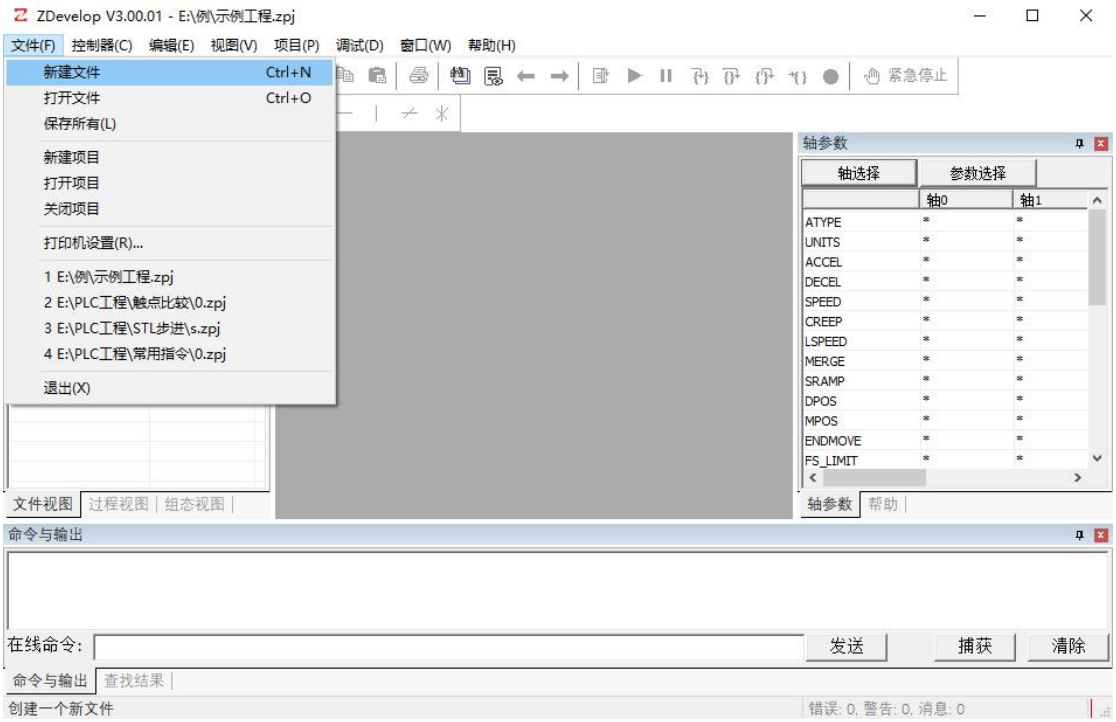
在电脑里新建一个文件夹用来保存即将要建立的工程。打开 ZDevelop 编程软件，当前说明例程的 ZDevelop 软件版本为 V3.00.01，更新软件版本请前往正运动官方网站下载，网址：www.zmotion.com.cn

1．新建项目：菜单栏“文件”-“新建项目”。



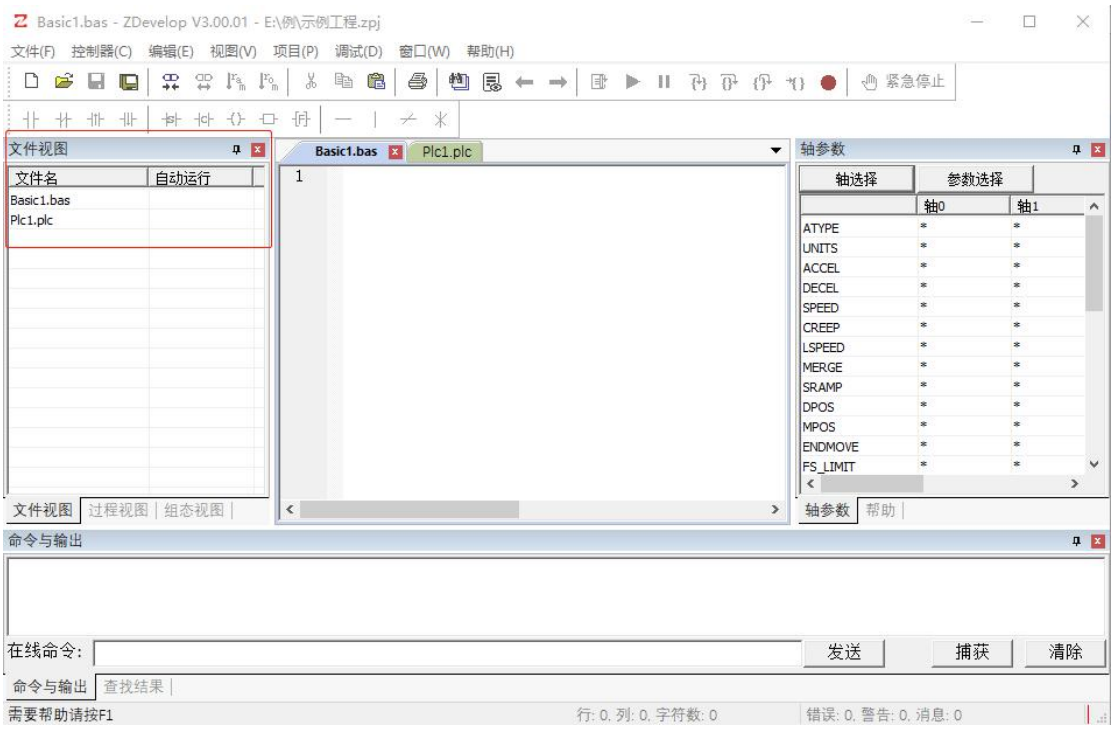
点击“新建项目”后弹出“另存为”界面，选择一个文件夹打开，输入文件名后保存项目，后缀为“.zpj”。

2．新建文件：菜单栏“文件”－“新建文件”。

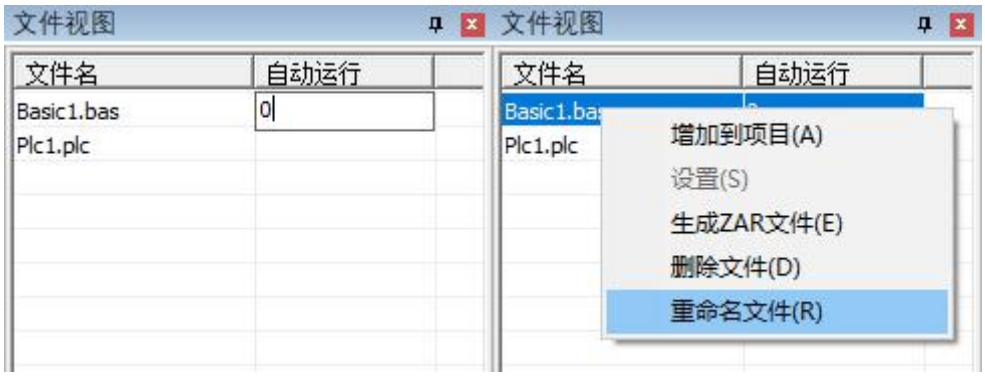


点击“新建文件”后，出现下图弹窗，选择新建的文件类型后确认。Basic/Plc/Hmi 分别针对 3 种不同类型的文件，表示 ZDevelop 支持的三种编程方式，基础连接使用步骤相同，支持 Basic/Plc/Hmi 混合编程。

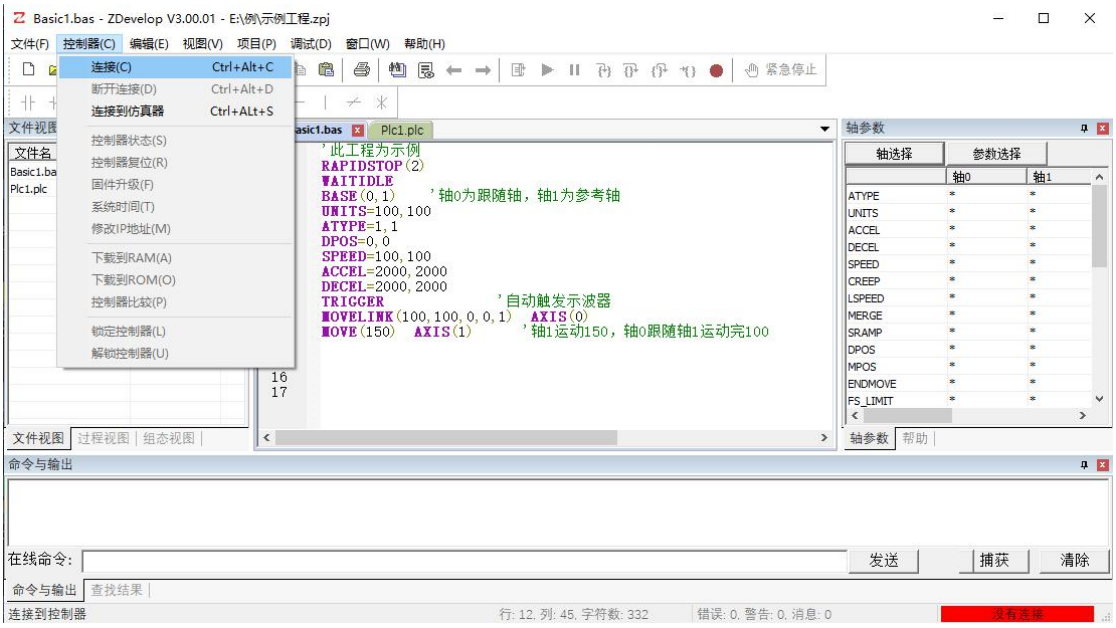
3. 保存文件：确认后新建的文件会自动加入到项目“文件视图”中，如下图。在程序编辑窗口写好程序后，保存文件  ，新建的文件会自动保存到项目 zpj 所在的文件下。



4. 设置文件自动运行：双击文件右边自动运行的位置，输入任务号“0”。
文件名称可自定义，在文件处点击鼠标“右键”-“重命名文件”修改。



5. 运行程序：在程序输入窗口编辑好程序，点击“控制器”-“连接”或“连接到仿真器”。“连接”表示连接到控制器，连接方法参见下节。



连接是否成功输出窗口会打印出信息提示。

连接到 ZMC005WEA：

```
Connected to Controller:ZMC005WEA Version:4.70-20170622.
```

连接到仿真器：

```
Connected to Controller:ZMC4xx-Simu Version:4.60-20180102.
```

点击菜单栏按钮 “下载到 RAM”或按钮 “下载到 ROM”，下载成功命令和输出窗口会有提示，同时程序下载到控制器并自动运行。RAM 下载掉电后程序不保存，ROM 下载掉电后程序保存。下载到 ROM 的程序下次连接上控制器之后程序会自动运行。

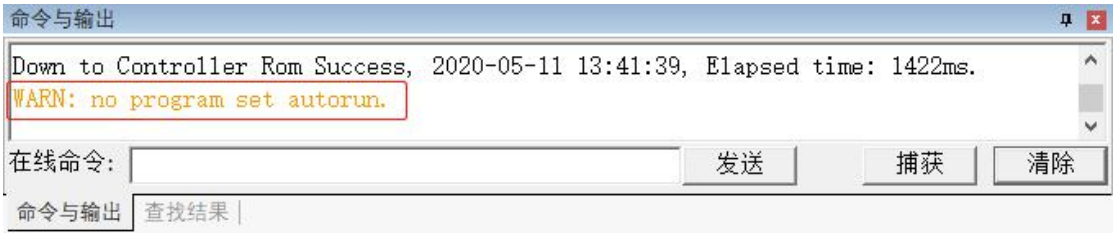
⚠ZMC00x 系列控制器不支持下载到 RAM。

⚠不建立项目的时候，只有 Bas 文件无法下载到控制器。

⚠自动运行的数字 0 表示任务编号，以任务 0 运行程序，任务编号不具备优先级。

⚠若整个工程项目内的文件都不设置任务编号，下载到控制器时，系统提示如下信息。

WARN: no program set autorun.

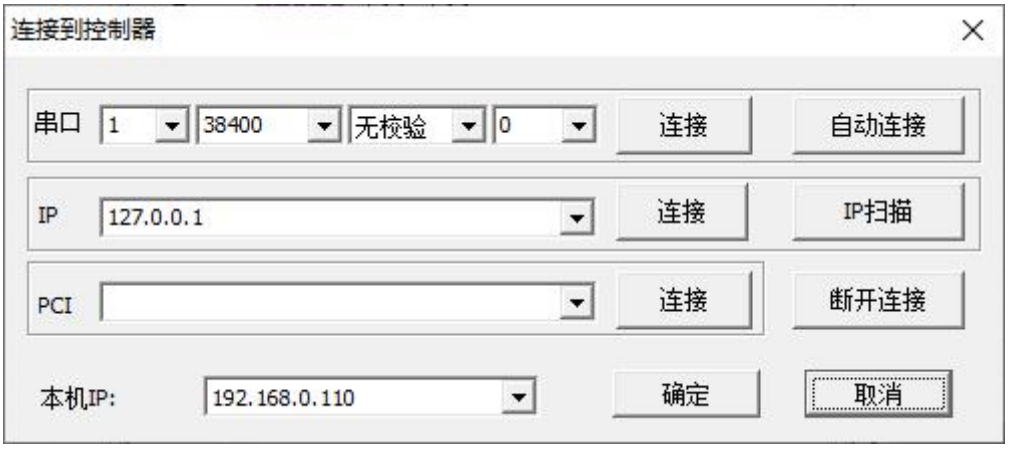


⚠打开工程项目时，选择打开项目 zpj 文件，若只打开其中的 Bas 文件，程序无法下载到控制器。

连接到运动控制器

通过“控制器”-“连接”菜单，可以连接到控制器。

ZDevelop 支持串口和以太网连接到控制器。



串口连接：选择需要连接的串口编号、设置波特率、校验位、停止位之后，点击连接，连接是否成功会在软件输出窗口自动打印出相应信息。

网口连接：IP 地址列表下拉选择时，会自动查找当前局域网可用的控制器 IP 地址，选择需要通讯的 IP 地址后连接即可。

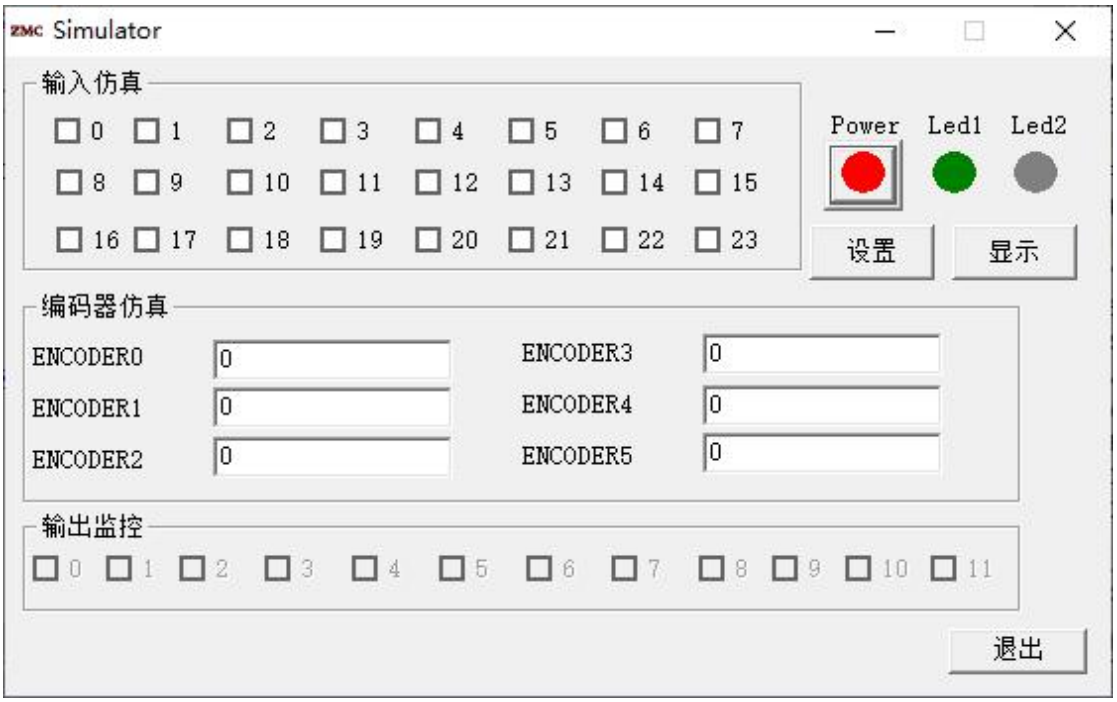
详细连接方法参见通讯方式章节。

连接到仿真器

ZDevelop 支持离线仿真，在无控制器情况下可以使用。

通过 “控制器”-“连接到仿真器”菜单可以自动启动仿真器，或当仿真器启动后可以通过 IP 地址“127.0.0.1”来连接。

当程序包含 Hmi 工程时，点击“显示”来实现 Hmi 界面仿真。



Hmi 示例工程运行效果如下，点击按钮即可直接设置轴参数。

2.1.4 在线命令与输出

在线命令与输出窗口可以查询与输出控制器的各种参数、打印程序运行结果、打印程序错误信息，软件开发人员在程序中给出的打印输出函数（由?、PRINT、WARN、ERROR、TRACE 等命令输出）。

注意问号使用英文符号，中文符号输入无效。ERRSWITCH 为 TRACE、WARN、ERROR 指令的控制开关，不同的参数值对应不同的输出效果：

0：TRACE、WARN、ERROR 指令全部不输出

1：只输出 ERROR 指令

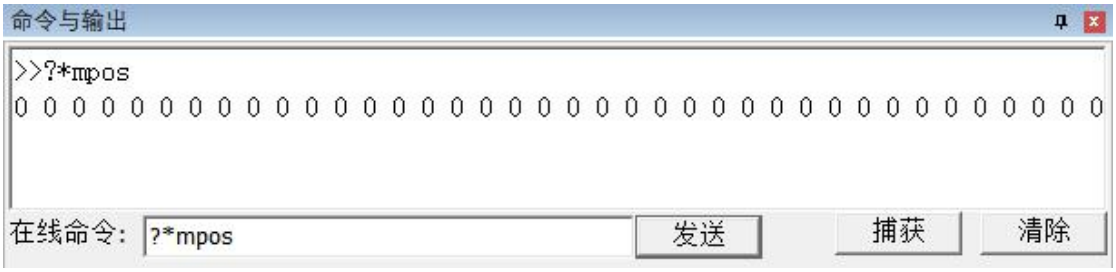
2：输出 WARN、ERROR 指令

3：TRACE、WARN、ERROR 指令全部输出

在线命令与输出窗口如下所示，“>>”表示 ZDevelop 在线命令输入的指令，在线命令输入“print 1+2”窗口会打印计算结果。

连接了控制器或仿真器就可以使用此功能，不受程序运行状态的限制。

使用在线命令可以查看各种轴的状态。如下图，在线命令输入“?*mpos”窗口会打印出多个轴的测量反馈位置 mpos。



查看系统的状态，“?*max”表示打印所有规格参数。

常用的打印查看命令有：

?*SET：打印所有参数值

?*TASK：打印任务信息

任务正常时只打印任务状态

任务出错时还会打印出错误任务号，具体错误行

?*MAX：打印所有规格参数

?*FILE：打印程序文件信息

?*SETCOM：打印当前串口的配置信息

?*BASE：打印当前任务的 BASE 列表（140123 以后版本支持）

?*数组名：打印数组的所有元素，数组长度不能太长

?*参数名：打印一个所有轴的单个参数

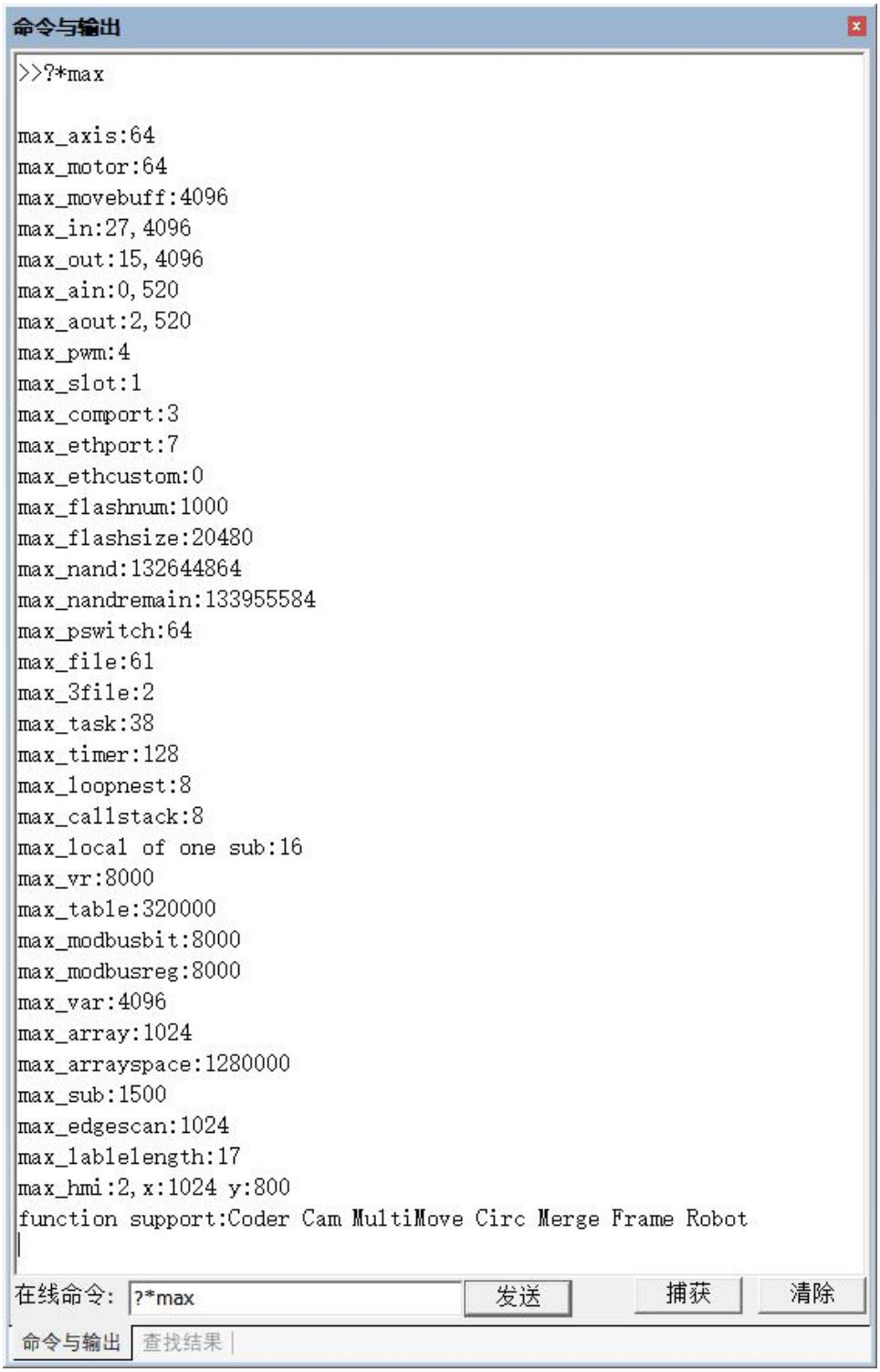
?*ETHERCAT：打印 EtherCAT 总线连接设置状态

?*RTEX：打印 Rtex 总线连接设置状态

?*FRAME：打印机械手参数，需要 161022 及以上固件支持

?*SLOT：打印出控制器槽位口信息（RTEX 口，EtherCAT 口）

?*PORT：打印所有 PORT 通讯口



修改变量的值。通过“在线命令”可以实现 VR 变量、TABLE 变量、MODBUS 变量、全局变量、系统设置、轴参数、轴状态变量的设置与修改。下图以修改 VR 变量值为例。

2.1.5 示波器的使用

示波器属于程序调试与运行中极其重要的一个部分，ZDevelop2.61 以上版本支持示波器功能，在“视图”-示波器中打开。示波器必须先启动后触发才能成功采样，打开示波器设置好之后点击启动，可手动触发，也可在程序里加入“TRIGGER”指令自动触发示波器采样。

编号：选择需要采集信息的轴号、数字 IO 编号、模拟量 IO 编号、TABLE 编号、VR 编号、MODBUS 数据编号等

数据源：选择抓取的数据类型，下拉菜单选择

偏移：波形上下偏移

刻度：纵轴刻度

水平刻度：横轴刻度

XY 模式：勾选是切换成 XY 平面显示，不勾选时时间为横轴，纵轴为各数据轴显示。

设置：设置示波器相关参数，以下以 ZDevelop3.00.00 版本为例说明。

若要设置示波器参数，如轴编号、数据源以及启动设置，要先停止示波器再设置，点击“设置”按钮，弹出如下所示“示波器”设置窗口。

通道数：要采样的数据通道

深度：总共采样的数据次数，深度越大采样范围越大。

间隔：采样时间间隔，单位为系统周期，与控制器固件版本有关，一般默认 1ms，指令 SERVO_PERIOD

查看。一般来说，间隔越小，采样数据越准确，相同时间内数据量越大。

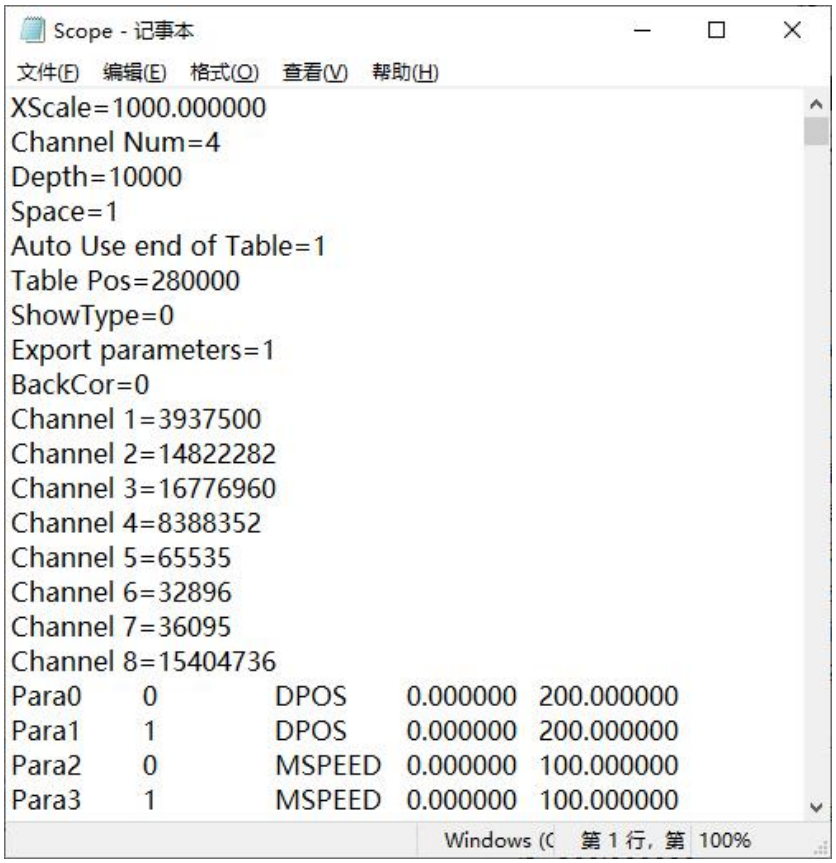
TABLE 位置：设置抓取数据存放的位置，一般默认自动使用 TABLE 数据末尾空间，也可以自己配置，但是设置时注意不要与程序使用的 TABLE 数据区域重合。

背景颜色/通道颜色：设置背景与每个通道波形对应的颜色。

显示类型：点和线段两种曲线类型可选。线段更容易发现异常点的数据显示。

导出参数显示数据更加完整，通道配置类型、采样时间等也一并记录下来导出。

导出采样数据方法：现在设置里勾选“导出参数”，启动示波器采样，采样完成后点击“导出”，选择文件夹保存示波器数据。



示波器采样使用方法：

建立项目文件，写入程序，连接控制器，程序里需要包含 TRIGGER 指令，程序完成后打开菜单栏“视图”-“示波器”。

在示波器窗口点击“设置”，选择采样通道数，采样深度，采样间隔，采样数据 TABLE 存储位置（一般来说采用默认位置即可，采样数据量很大的时候可以将 TABLE 存储位置扩大）和采样类型，设置完成关闭设置窗口。

再选择采样数据编号和数据源点击“启动”按钮。

将程序下载到控制器运行，此时示波器开始采样，显示出不同数据源的波形。可调整显示刻度和波形偏移，便于观察不同波形。

若波形精度或显示不完整，可点击“停止”按钮后再打开“设置”，调整好采样间隔和采样深度后重新执行采样过程。

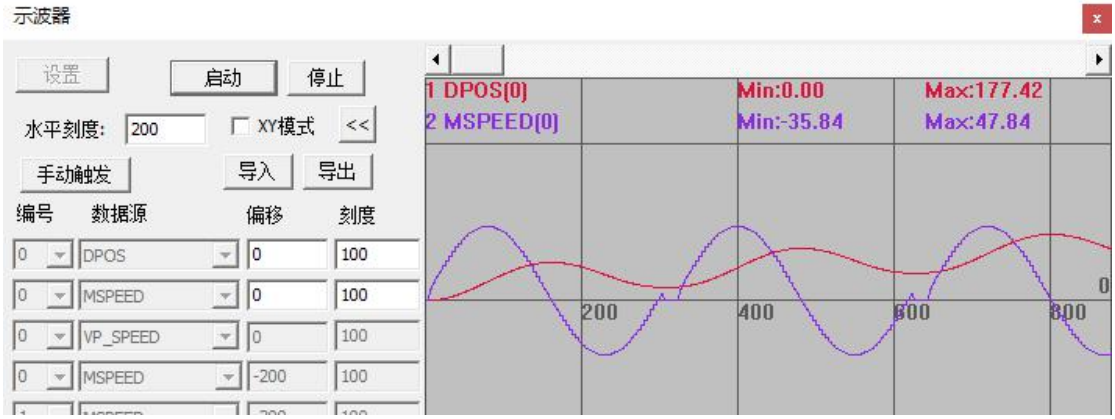
调整深度/间隔示例一：

```
RAPIDSTOP(2)
WAIT IDLE(0)
ERRSWITCH = 3
BASE(0) '选择第 0 轴
ATYPE=1    '脉冲方式步进或伺服
DPOS = 0
UNITS = 100'脉冲当量
SPEED = 200
ACCEL = 2000
DECEL = 2000
TRIGGER
'计算 TABLE 的数据
DIM deg, rad, x, stepdeg
stepdeg = 5     '可以通过这个来修改段数，段数越多速度越平稳
FOR deg=0 TO 360 STEP stepdeg
    rad = deg * 2 * PI/360      '转换为弧度
    X = deg * 25 + 10000 * (1-COS(rad)) '计算每小段位移
    TABLE(deg/stepdeg,X)              '存储 TABLE
    TRACE deg/stepdeg,X
NEXT deg
WHILE 1'循环运动
    IF IN(0) = ON THEN '输入 0 有效启动运动
        CAM(0, 360/stepdeg, 0.1, 300)     '虚拟跟踪总长度 300
        WAIT UNTIL IDLE '等待运动停止
        DELAY(100) '延时
    ENDIF
WEND
END
```

点击示波器工具中的“设置”，设置通道数为 2，设置深度、间隔等参数后确定。采集 DPOS(0)和 MSPEED(0)数据，水平刻度均为 200，竖直刻度均为 100，无偏移。

深度：10000，间隔：5

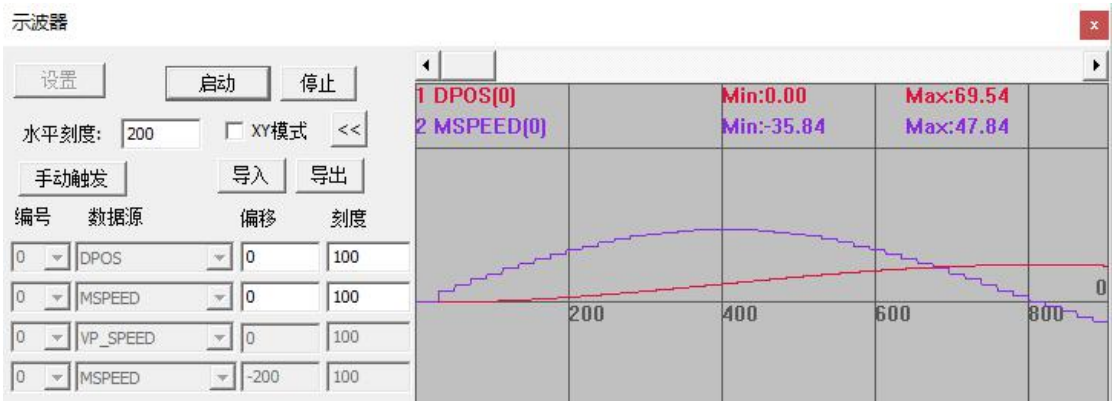
如果系统的 SERVO_PERIOD=1000,也就是 1ms 轨迹规划周期，意味着 5ms 采集一个数据点，一共采集 10000 次数据，采集时间长度为 50s。



深度：10000，间隔：1

如果系统的 SERVO_PERIOD=1000,也就是 1ms 轨迹规划周期，意味着 1ms 采集一个数据点，一共采集 10000 次数据，采集时间长度为 10s。

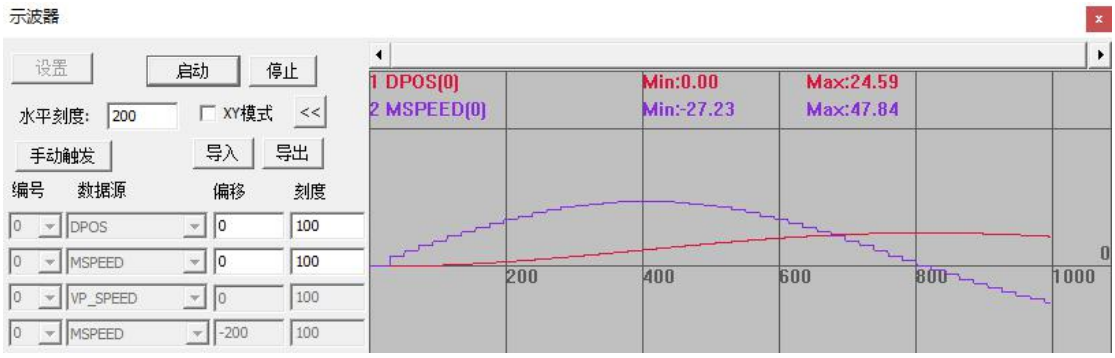
深度不变，间隔调小之后采样波形更为精确。



深度：1000，间隔：1

如果系统的 SERVO_PERIOD=1000,也就是 1ms 轨迹规划周期，意味着 1ms 采集一个数据点，一共采集 1000 次数据，采集时间长度为 1s。

将深度调小之后采样时间较短，只能采集到部分波形。



连续轨迹加工的示波器示例二：

```
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0,1)
DPOS=0,0
ACCEL=500,500                    '设置加速度
DECEL=500,500                    '设置减速度
SPEED=100,100                    '运行速度
SRAMP=100
CORNER_MODE=32                   '启动倒角
ZSMOOTH = 10                     '倒角参考半径
TRIGGER                          '自动触发示波器
MOVE(100,0)
MOVE(0,100)                      '上面两条直线间自动倒角
```

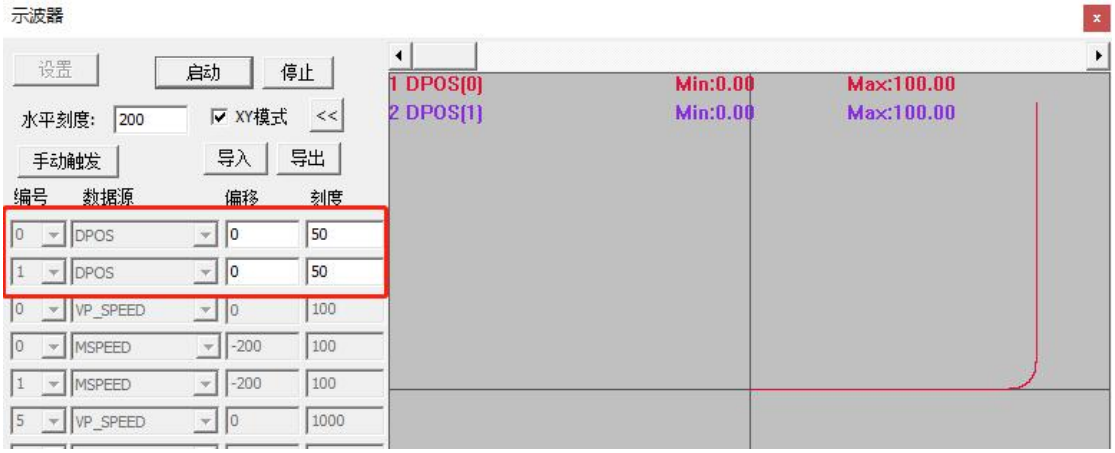
点击示波器工具中的“设置”，设置通道数：2，深度：10000，如下图所示。

设置示波器的水平刻度：1000，代表如果选择时间为横轴模式，一格为 1000 个采样周期。数据源分别为 DPOS(0)、DPOS(1)。偏移分别为 0、0。刻度为 100、100。

勾选 XY 模式，代表 DPOS(0)、DPOS(1)的显示数据分别为 X、Y 轴内容。

点击启动后下载程序运行，因为程序中有 TRIGGER 指令会自动开始绘图，如下所示。可以很清晰的看到 DPOS(0)、DPOS(1)走出的轨迹如下图所示。

调整 DPOS(0)、DPOS(1)中的刻度为 50、50，可以看到整个示波器图形被放大，可以看到 CORNER_MODE 设置后倒圆角的显示内容。

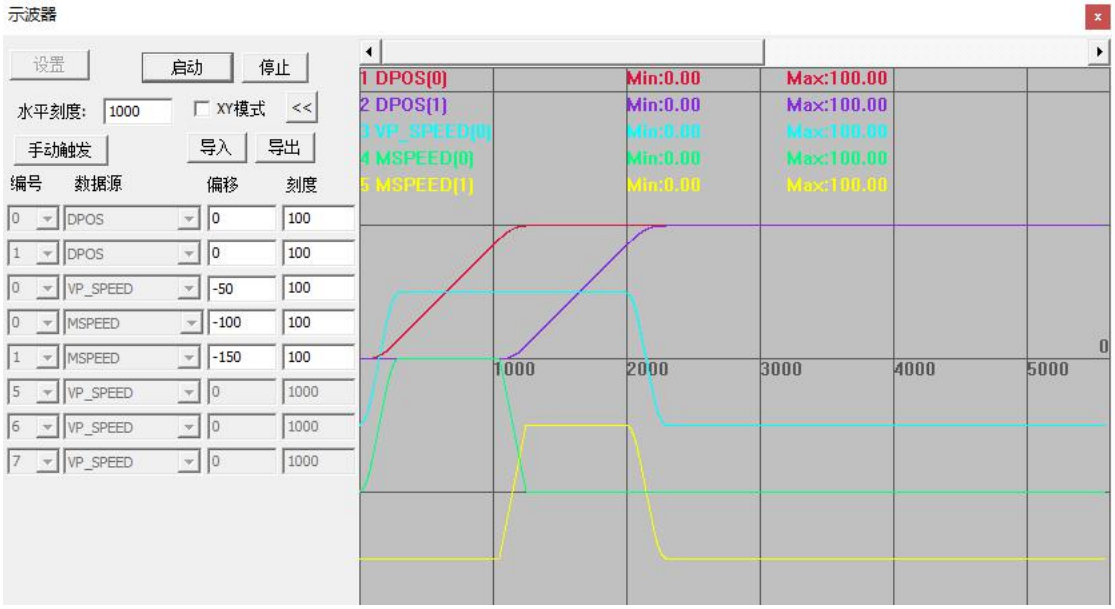


也可以设置不同参数，看以时间轴为横轴的位置、合成速度、各轴分速度等相关的曲线。

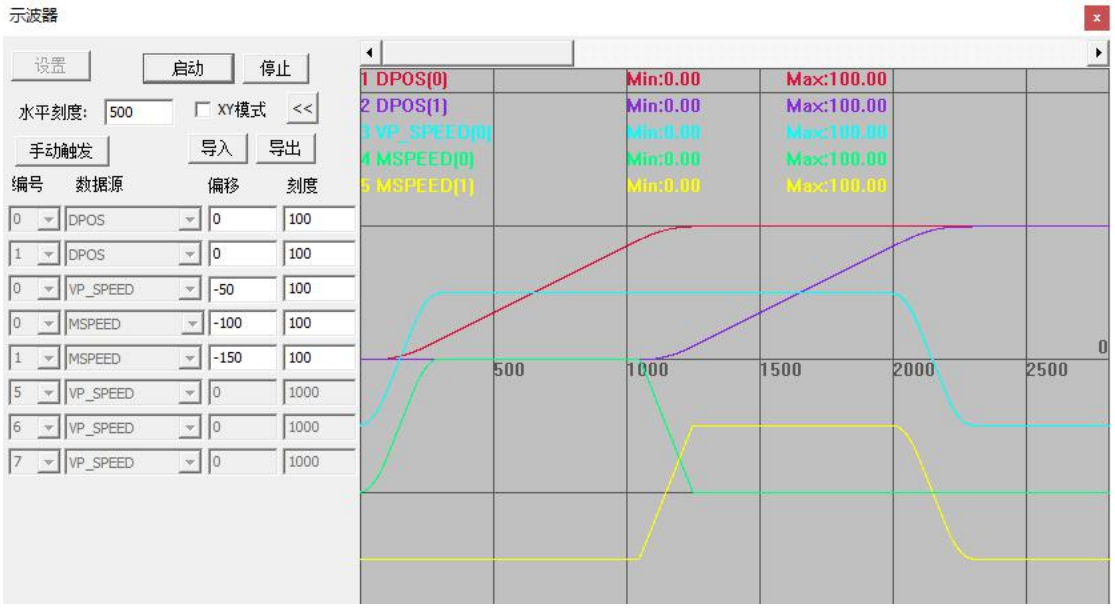
通道数：5，深度：10000，间隔：1

设置 5 个通道的采样数据分别为：

点击启动后下载程序运行，因为程序中有 TRIGGER 指令会自动开始绘图，如下所示。可以很清晰的看到 DPOS(0)、DPOS(1)、VP_SPEED(0)、MSPEED(0)、MSPEED(1)走出的时间轴数据如下图中所示（时间刻度为每格 1000 次采样）。



也可以设置水平刻度为 500，可以看到采样数据的长度更长。

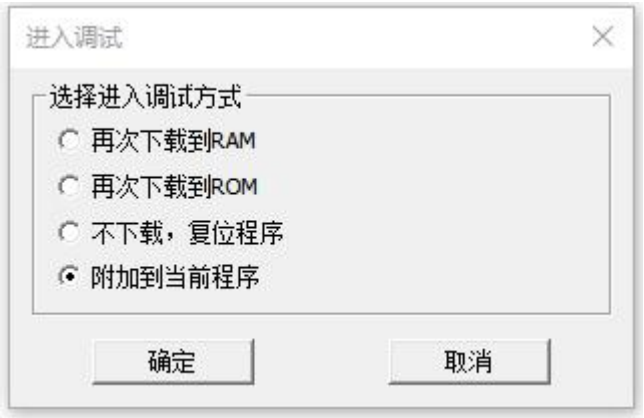


TRIGGER 指令可以非常简洁与灵活的辅助波形查看与问题查找，示波器功能对查找问题非常有用，可以针对性的用好，进一步提高调试效率。

2.1.6 程序调试

⚠调试机器要注意安全！请务必在机器中设计有效的安全保护装置，并在软件中加入出错处理程序，否则所造成的损失，正运动技术公司没有义务或责任对此负责。

ZDevelop 连接控制器后，从菜单栏选择“调试”-“启动/停止调试”，弹出如下窗口，选择“再次下载到 RAM”表示程序再次下载到 RAM 运行，“再次下载到 ROM”表示程序再次下载到 ROM 运行，“不下载，复位程序”表示不下载程序，仅重新运行之前下载的程序，“附加到当前程序”表示此时程序不下载，仅在窗口显示目前的运行状态。



如下图，此时可以查看各任务运行情况、监视内容、子函数堆栈调用过程、子函数局部变量值。

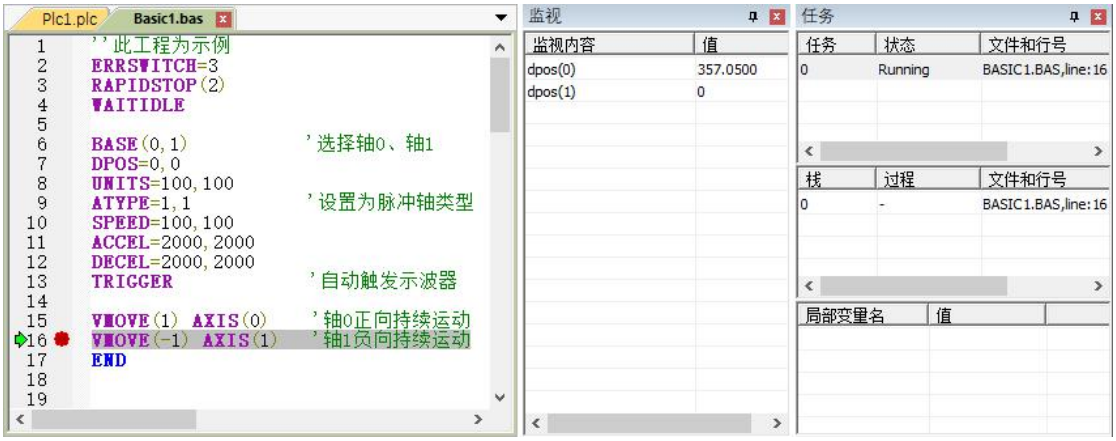
请保证 PC 程序文件和控制器的程序文件一致，否则可能导致光标位置错误。调试只能在控制器 UNLOCK 状态时进行程序调试。

断点调试可以查看程序运行的具体过程，主要用于判断程序逻辑错误。配合监视内容和轴参数变化情况可以查看程序每执行一步对寄存器、变量、数组等的影响。断点快捷键 F9 添加，或“调试”-“增删断点”，断点可以添加多个。

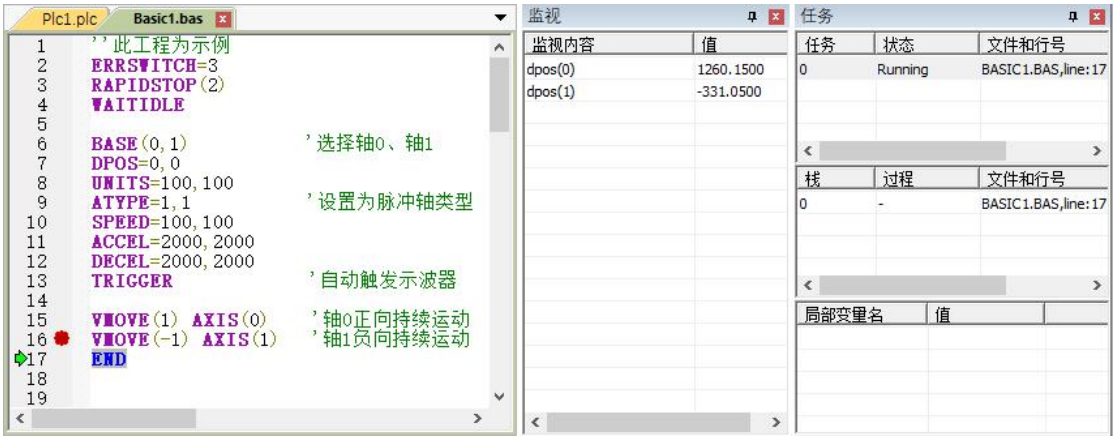
添加断点后，程序运行会停止在断点处，此时断点处对应的指令还没有执行。如下图，绿色箭头表示当前停止在哪一行，“栈”表示断点停止在哪一个任务，“文件和行号”也可以查看停止位置。其他任务此时也同时停止在各自执行到的位置。

前面已经扫描的程序功能不受影响，如下图，如下图，第 15 行 VMOVE(1)　AXIS(0)已经执行，此时

轴 0 正向运动，后面没有扫描到的程序不执行，第 16 行 VMOVE(-1)　AXIS(1)还未执行，此时轴 1 不动。



程序停止在断点处后，就可以进行逐步调试，快捷键 F11，按一次程序向下执行一步。如下图，此时程序执行到第 17 行，轴 1 开始负向运动，第 17 行语句未执行。



单步调试完一段程序确认无误后，可以点击  三角符号恢复程序运行，也可以使用快捷键 F5 运行程序。

如果断点是设置在循环中，那么下次循环运行到断点处时还是会停止程序。

程序调试完成后，需要清除所有断点才能下载到控制器运行，断点不清除就下载程序到控制器，命令与输出区域会打印如下警告信息：Warn file:"BASIC1.BAS" line:11 task:0, Paused.

2.2 编程基础知识

2.2.1 程序

程序是由序列组成的，告诉计算机如何完成一个具体的任务。程序是软件开发人员根据用户需求开发的、用程序设计语言描述的适合计算机执行的指令（语句）序列。

ZBasic 语法不区分大小写，程序中指令的所有标点符号应为英文格式。

一个程序应该包括以下两方面的内容：

1．对数据的描述。在程序中要指定数据的类型和数据的组织形式，即数据结构。（参见：DIM、GLOBAL、CONST 等变量定义语句描述。）

2．对操作的描述。即操作步骤，也就是算法。结合到运动控制就是运动与动作的工艺。

2.2.1.1 常见程序结构

为编写算法，我们一般要用到三种程序结构描述方式：顺序、选择、循环。

1．顺序

在没有条件和循环的情况下，程序总是从上往下运动。当设置自动运行时，文件缺省都是从文件开始顺序往下执行的。

功能块 1

功能块 2

如上，功能块 1 先执行，然后是功能块 2

BASIC 编程下，程序从上往下扫描一次。

PLC 编程下，程序从上往下周期扫描。

2．选择

根据执行条件的不同，选择不同的语句执行。主要的选择语句有：IF THEN，ON GOTO ，ON GOSUB 等。

例程 1：

```
DIM   aa
aa=1
IF   aa= 0   THEN
     语句 1
ELSEIF   aa=1   THEN
     语句 2
ELSE
     语句 3
ENDIF
END
```

例程 2：

```
DIM   a
a=100
ON   a>10   GOTO   label1
a=1000
END            '主程序结束

label1:
PRINT   a
END            'goto 跳转无法 return 返回。
```

3．循环

程序重复执行，则称为循环。主要的循环语句有：FOR NEXT，WHILE WEND ，REPEAT UNTIL 等。

例程 1：

```
DIM   a
```

```
a=0
FOR i = 1 TO 10 STEP 1
    a = a+1
    PRINT   a
NEXT
```

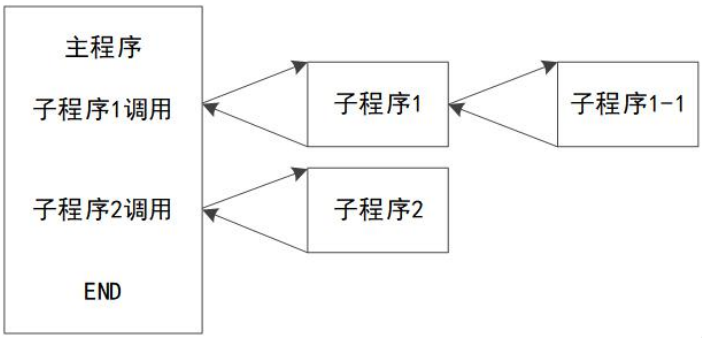
例程 2：

```
DIM   a
a=0
WHILE   IN(1)=OFF   '直到输入 1 有效，退出循环
    a=a+1
    PRINT a
    DELAY(1000)
WEND
```

2.2.1.2 子程序

编程过程中会经常应用到子程序（SUB 过程调用），运用子程序可将编程模块化，各模块之间的关系尽可能简单，在功能上相对独立，相当于简化了主程序，使编程效率更高更易读，能有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子程序和子任务，便于开发和维护。

主程序和子程序的执行逻辑如下图：



主程序运行到子程序调用行就会调用该子程序，当前主程序内容会压栈，子程序执行完后返回到子程序调用主程序出栈，主程序继续往下执行。

主程序调用子程序最多 8 层。

子程序分为全局 SUB，文件模块 SUB，全局 SUB 可以应用在所有文件中，文件模块 SUB 只能用于当前文件，子程序还能用于传递参数以及返回参数。详细用法参见 SUB 指令描述。

2.2.2 数据相关

2. 2. 2. 1 数据定义

1. 变量

变量是用户可以自定义的变量，分为全局变量（GLOBAL）、文件模块变量（DIM）、局部变量（LOCAL）三种，其中全局变量可以在任意文件中使用；文件模块变量只能在本程序文件内部使用；局部变量主要用在 SUB 中，其他文件无法使用。

变量可以不经过定义直接赋值，此时的变量默认为文件模块变量。

变量用于暂时保存与外部之间传输的输入输出数据、任务内部处理时的数据。换言之，它是用于保存带名称和数据类型等属性的数据，无需指定变量与存储器地址之间的分配。

变量名称是用于识别变量类型的名称。一个任务内，不能对相同变量名称的变量进行多次声明，但不同的任务中，可对相同变量名称的局部变量进行声明，并将其视为不同的变量进行处理。

DIM 定义变量,可以一次定义多个,逗号间隔,变量的定义和赋值必须分行进行。全局变量定义 GLOBAL DIM，文件模块变量定义 DIM，局部变量定义 LOCAL DIM。例如：

```
GLOBAL DIM    a,b,c
DIM    e,f
LOCAL DIM    m,n
```

数组、SUB：有全局数组（GLOBAL）和文件模块数组（DIM）两种。数组、SUB 必须先定义再使用。

1）文件模块 SUB 定义格式：

```
SUB    函数名()
       代码块
END    SUB
```

2）全局 SUB 定义格式：

```
GLOBAL    SUB    函数名()
       代码块
END    SUB
```

建议把变量数组定义语句放自动运行文件的最前面，保证被执行到，把 SUB 过程放在主函数 END 后面，在主函数内调用到时才进入 SUB。

数组指定是指集中相同属性的数据后对其进行统一，并对数据类型进行的指定。构成数组指定的各数据称为“元素”。

2．常量

变量的值因代入该变量的数据而异。与之相对的固定不变的值为常数。

例如：CONST   VALUE = 100000   'VALUE 的值一经定义不能再修改，只可读取 。

CONST 定义常量，一次只能定义一个，且定义与赋值必须在一行。常量可定义为全局常量 GLOBAL CONST，全局常量可以在任意文件中使用，不存在 LOCAL CONST 的写法。常数与变量不同，不是保存在存储器中的信息，常见的常量有布尔型，字符串型，时间型，日期型，整型等。

2.2.2.2 数据类型

在计算机内部，数据都是以二进制的形式存储和运算的，二进制数据中的位（bit）是计算机存储数据的最小单位。

一个二进制位只能表示 0 或 1 两种状态，要表示更多的信息，就要把多个位组合成一个整体，一般以 8 位二进制组成一个基本单位字节（Byte）。

字节是计算机数据处理的最基本单位，并主要以字节为单位解释信息。一般情况下，一个 ASCII 码占用一个字节，一个汉字国际码占用两个字节。计算机型号不同，其字长是不同的，常用的字长有 8、16、32 和 64 位。

单位转换：1Byte=8bit，1KB=1024B，1MB=1024KB，1GB=1024MB。

常见进制有二进制，八进制，十进制，十六进制。各种运动参数默认为十进制数据。

名称	说明
位（Bit）	位为二进制数值之最基本单位，其状态非 1 即 0
位数（Nibble）	由连续的 4 个位所组成（如 bit3～bit0）一个位数可用以表示十进制数字 0～15 或十六进制 0～F
字节（Byte）	由连续之两个位数所组成（8 位，bit7～bit0）。可表示十进制数字 0～255 或十六进制的 00～FF
字符（Word）	由连续之两个字节所组成（16 位，bit15～bit0）可表示十进制数字 0～65535 或十六进制的 4 个位数值 0000～FFFF
双字符（Double Word）	由连续之两个字符所组成（32 位，bit31～bit0），可表示十进制数字 0～2^{32}-1 或十六进制的 8 个位数值 00000000~FFFFFFFF

数据类型是指对变量表示的值的形式和范围进行特定的规定。声明该变量时，数据类型的大小根据存储器内的数据范围大小而定，存储器内的数据范围越大，可表示的值的范围就越大。

基本数据类型	描述
布尔型	值为 0 或 1 的数据类型
整数型	值为整数的数据类型
实数型	值为实数的数据类型
日期型	以日期格式表示 DD：MM：YYYY
时间型	以时间格式表示 hh：mm：ss
字符串型	值为字符串的数据类型

指令的输入/输出变量的数据类型由指令确定。

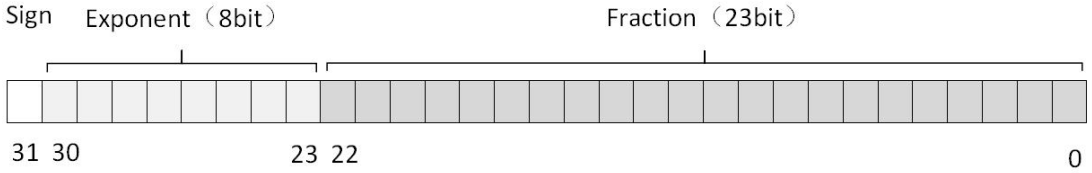
自定义变量的数据类型属于动态类型。

将整数赋值给变量时，变量就是整型；将浮点数赋值给变量，变量就是浮点型。

自定义数组的数据类型分为单精度浮点数和双精度浮点数，参照下文浮点数相关说明。

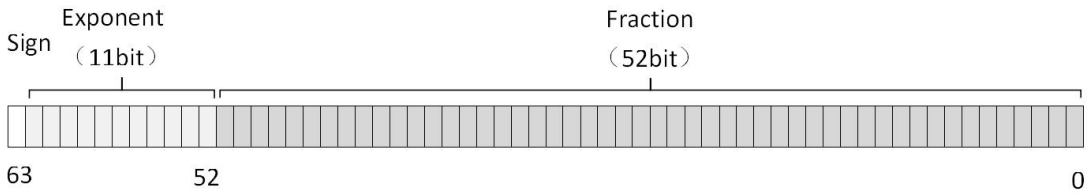
单精度浮点数 32 位，如下图。

数据格式属于单精度浮点数的有：VR、MODBUS_IEEE、TABLE 及自定义数组与变量（ZMC3 系列及之前的控制器）。



双精度浮点数 64 位，如下图。

数据格式属于双精度浮点数的有：TABLE 及自定义数组与变量（ZMC4 系列及之后的控制器）。



常用寄存器数据类型表：

寄存器类型	数据类型	取值范围

MODBUS_BIT	布尔型	0 或 1
MODBUS_REG	16 位整型	-32768 到 32767
MODBUS_LONG	32 位整型	-2147483648 到 2147483647
VR_INT		
MODBUS_IEEE	32 位浮点型	-3.4028235E+38 到-1.401298E-45
VR		
TABLE，自定义数组，变量（ZMC3 系列及之前）		
TABLE，自定义数组，变量（ZMC4 系列及之后）	64 位浮点型	1.7E-308 到 1.7E+308
VRSTRING	字符	一个字符占一个 VR
MODBUS_STRING	字符	一个字符占 8 位

所有数据所需的存储器容量与各数据的总数据大小（容量值）不一致，原因在于，分配至存储器的数据的开头位置自动配置至各数据类型的“校准值（边界值）”倍数位置，各数据类型之间会产生空白。即使数据类型的种类相同，整体占用的数据大小仍会因数据类型的顺序而异。

2.2.2.3 数据操作

不同类型数据之间的操作，会产生下列问题：

1．数据丢失：浮点型向整型转换时会丢失小数部分。

例程：

```
VR(0)=10.314
MODBUS_REG(0)=0
MODBUS_REG(0)=VR(0)
?MODBUS_REG(0)          '结果为 10
```

2．强制转换：整型存储到浮点型寄存器后会变成浮点型，再使用整型操作数据可能会不正确。

3．常见使用问题：获取日期时，不要使用单精度浮点型存储，因为日期格式是 8 位的，而单精度浮点数有效位只有 6 位，建议直接使用 32 位整型 MODBUS_LONG 来存储。

部分参数必须用字符串类型常量或变量，各种字符串可以通过“+”合并，对字符串的单个字节的操作需要利用数组来进行。

字符串相关指令列表：

字符串指令	描述
DIM	定义的数组可以直接作为字符串使用，每个元素表示一个字节
“”	双引号直接定义字符串常量
CHR	把 ASCII 转成一个字符串常量，此字符串只有一个字节
MODBUS_STRING	标准 MODBUS 协议定义字符串，每个 16 位寄存器存储 2 个字节
VRSTRING	VR 列表作为字符串使用，一个 VR 存储一个字节
+	操作符，两个字符串合并
VAL	数字符串转换为数值
TOSTR	数值转换为字符串
STRCOMP	字符串比较函数

DMCPY	数组拷贝函数，可以用于拷贝字符串
HEX	返回为十六进制格式，只能用于打印
DATE$	按 dd：mm：yyyy 格式返回 DATE 设置日期
DAY$	返回本日的星期英文名
TIME$	按 24 小时的格式 hh：mm：ss 返回当前时间

2.2.2.4 参数

参数分轴参数、任务参数、系统参数等，参数可读可写（除少数只读参数外）。

在运动前要设置好轴参数，以及相关安全设置（正负硬件限位、正负软件限位、急停减速度等）。

参数有自动保存和非自动保存两类。

1.自动保存的参数修改后自动存储，重新上电不会恢复为缺省值，相关指令有：IP_ADDRESS、APP_PASS 和 LOCK 等密码指令、CANIO_ADDRESS 等。

2.非自动保存的参数上电后恢复为缺省值，要重新修改，比如 SETCOM 设置串口参数，每次上电后都要再设置一次，所以 SETCOM 指令要放在程序开头。

2.2.2.5 掉电存储

控制器具有掉电非易失保护寄存器VR和多个扇区存储FLASH块，ZDevelop在线命令?FLASH_SECTES查看扇区数量，?FLASH_SECTSIZE 查看扇区大小，可以用于保存掉电的数据。

ONPOWEROFF 掉电中断可以用编写的程序记录掉电时的位置到 VR，系统重新上电时，使用程序将 VR 的数据恢复到当前位置。

使用 SETCOM 指令可以把 VR 与 MODBUS_REG 寄存器匹配起来，指令参数 variable 需设置为 0，详细设置参见 SETCOM 指令。

VR 是掉电非易失的，可以无限次的读写，但是长时间不开机可能导致数据丢失，机器设备关键参数建议存储到 FLASH，，上电是由 FLASH 读出写至各相关变量。

FLASH 是有写入寿命限制，不可以无限次的擦写，经常改写的数据建议写入 VR。

2.2.3 ZDevelop 的三种编程方式

ZDevelop 软件可以支持三种编程方式，分别为 ZBASIC、ZPLC 和 ZHMI 组态编程。使用 ZDevelop 软件编写的程序可以下载到正运动控制器里。

ZBASIC、ZPLC 和 ZHMI 之间可以多任务运行，ZBASIC 可以多任务号运行，ZPLC 与 ZHMI 均只能一个任务号运行。

例如：如下图，同一项目内的两个不同的 BASIC 文件可以设置不同任务号分别运行，同一项目内的 PLC/HMI 文件都只能有一个任务号。

文件名	自动运行
Basic1.bas	0
Basic2.bas	1

文件名	自动运行
Plc1.plc	0
Plc2.plc	

ZPLC 编程和 ZBASIC 编程均具有简单易懂、逻辑结构清晰的特点，能满足多种编程要求，目前应用十

分广泛。HMI 组态编程适用于正运动 ZHD 系列示教盒，其公司的示教盒请使用该公司提供的组态编程软件。

PLC 与 BASIC 相关寄存器对应关系：

PLC		BASIC	
输入继电器 X	X0~X7	输入口 IN	IN(0)~IN(7)
	X10~X17		IN(8)~IN(15)
	X20~X27		IN(16)~IN(23)
	…		…
	X1770~X1777		IN(1016)~IN(1023)
输出继电器 Y	Y0~Y7	输出口 OP	OP(0)~OP(7)
	Y10~Y17		OP(8)~OP(15)
	Y20~Y27		OP(16)~OP(23)
	…		…
	Y1770~Y1777		OP(1016)~OP(1023)
辅助继电器 M	M0	MODBUS_BIT	MODBUS_BIT(0)
	M1		MODBUS_BIT(1)
	…		…
	M1023		MODBUS_BIT(1023)
特殊继电器 D	D0	MODBUS_REG	MODBUS_REG(0)
	D1		MODBUS_REG(1)
	…		…
	D1023		MODBUS_REG(1023)
浮点寄存器 DT	DT0	TABLE	TABLE(0)
	DT1		TABLE(1)
	…		…
	DT1023		TABLE(1023)
EXE @BASIC 指令		BASIC 指令	

输入继电器 X 对应 IN，PLC 编程下，X 为 8 进制（X0~X7，X10~X17，…），而控制器的输入口 IN 为 10 进制，编写程序时要特别注意，做进制转换，比如 IN20 口对应 X24，IN8 对应 X10。

输出继电器 Y 对应 OP，PLC 编程下，Y 为 8 进制（Y0~Y7，Y10~Y17，…），而控制器的输出口 OUT 为 10 进制，编写程序时要特别注意，做进制转换，比如 OUT20 口对应 Y24，OUT8 对应 Y10。

辅助继电器 M 对应 MODBUS_BIT。

特殊继电器 D 对应 MODBUS_REG。

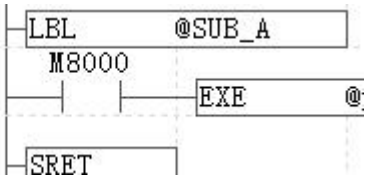
浮点寄存器 DT 对应 TABLE，可以用于与 ZBASIC 之间传输数据。

PLC 里使用“EXE @BASIC 指令表达式”可调用 BASIC 指令。

Basic 里可使用语句“RUN “xxx.plc”,任务编号”来启动 PLC 任务。

使用“CALL SUB_FUNC”或“RUNTASK SUB_FUNC”来调用 PLC 子程序 LBL。更多详细内容参见“ZMotion PLC 编程手册”。

```
'Basic开启任务调用PLC文件
run "Plc1.plc",6
'Basic开启任务调用PLC子函数
runtask 2,SUB_A
END
```



2.3 寄存器

控制器寄存器主要有 TABLE、FLASH、VR、MODBUS 寄存器。将 ZDevelop 软件与控制器连接后，可通过 ZDevelop 软件“控制器”-“控制器状态”查看该控制器各寄存器的空间大小，也可以通过在线命令和输出窗口输入“?*max”来查看各寄存器的数量，不同的控制器存储容量大小不同。

2.3.1 TABLE

TABLE 是控制器自带的一个超大数组，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），掉电不保存。编写程序时，TABLE 数组不需要再定义，可直接使用，索引下标从 0 开始。

ZBasic 的某些指令可以直接读取 TABLE 内的值作为参数，比如 CAM，CAMBOX，CONNFRAME，CONNREFRAME，MOVE_TURNABS，B_SPLINE，CAN，CRC16，DTSMOOTH，PITCHSET，HW_PSWITCH 等指令，示波器采样的参数也存储在 TABLE 里。因此在开发应用中要注意多个 TABLE 区域的分配与使用，不要与示波器采样的数据存储区域重合。TSIZE 指令读取 TABLE 空间大小。

TABLE 使用时先将参数存储在 TABLE 的某个位置，再使用指令调用 TABLE 数据。TABLE(0) = 10 表示 table(0)赋值 10。TABLE(10,100,200,300)表示 table(10)赋值 100，table(11)赋值 200，table(12)赋值 300。

TABLE 作为参数传递时用法大致相同，以 CAM 凸轮指令为例：

CAM(start point, end point, table multiplier, distance)

start point：起始点 TABLE 编号，存储第一个点的位置

end point：结束点 TABLE 编号

table multiplier：位置乘以这个比例，一般设为脉冲当量值

distance：参考运动的距离

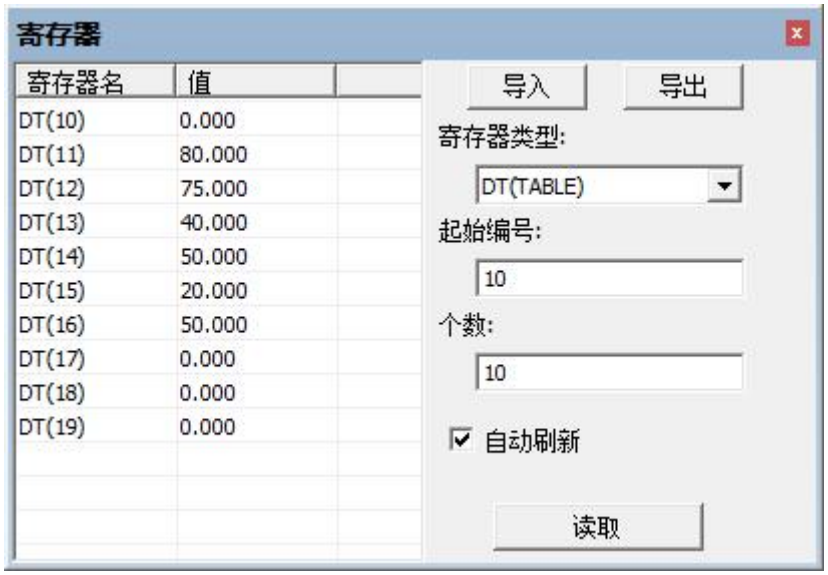
使用方法示例：

```
TABLE(10,0,80,75,40,50,20,50,0)   'table 从 10 开始存数据，table(10)赋值 0，table(11)赋'值 80，依次往下
CAM(10,17,100,500)                '运动轨迹为 table10 到 17
```

查看 TABLE 内数据的方式有 2 种：

第一种：在在线命令行输入?*TABLE(10,8)查询 TABLE(10)开始，依次 8 个数据。

第二种：在寄存器中查看 DT(TABLE)数据，起始编号从 10 开始，个数 8 个。



2.3.2 FLASH

严格来讲，FLASH 不是寄存器，但它与寄存器密切相关，所以放于此章叙述。

FLASH 具有掉电存储功能，读写次数限制为十万次，长期不上电也不会丢失数据。一般用于存放较大的，不需要频繁读写的数据，比如加工的工艺文件。读与写时要注意保证要操作的变量，数组等名称和次序高度一致，如果不一致会导致数据错乱。

FLASH 使用时是按块编号，块数 FLASH_SECTES 指令查看，不同的控制器 FLASH 块数与块数据大小都不同，每块数据大小 FLASH_SECTSIZE 指令查看。

可以在在线命令行查看，如下图。

CAN 通讯设置的参数，IP 地址、APP_PASS、LOCK 密码等系统参数存储到 FLASH。

FLASH 使用方法：

1）数据存储到 FLASH 块：把 VAR，ARRAY1，ARRAY2 数据依次写入 FLASH 块 1。

FLASH_WRITE 1，VAR，ARRAY1，ARRAY2

2）FLASH 块数据读取：把 FLASH 块 1 的数据依次读入 VAR，ARRAY1，ARRAY2。

FLASH_READ 1，VAR，ARRAY1，ARRAY2

2.3.3 VR

VR 寄存器具有掉电存储功能，可无限次读写，但数据容量较小，一般只有 1024 或者更少，用于保存需要不断修改的数据，例如轴参数、坐标等。数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），可

使用 VR_INT 强制保存为整型，VRSTRING 强制保存为字符串。

VR 的掉电保存原理是控制器内部有缺电存储器，但数据容量较小，所以数据量较大的或需要长久保存的数据最好写到 FLASH 块或导出到 U 盘。

VR 使用方法：

1）数据存储到 VR：VR(index) = value

2）读取 VR 数据：VAR1 = VR(index)

VR 寄存器还可用于 RTEX 控制器传递读写数据，DRIVE_WRITE 参数写入，DRIVE_READ 参数读取，具体使用方法参见第十六章总线相关的 RTEX 总线指令。

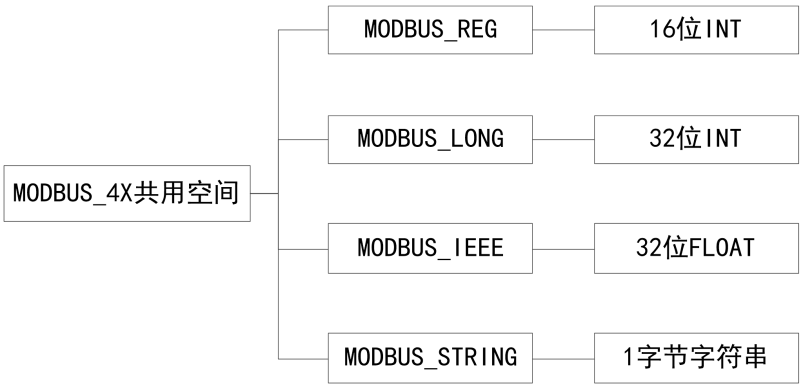
使用 CLEAR 指令清除 VR 内的全部数据，CLEAR_BIT 指令将 VR 某个位置 0，READ_BIT 指令读取 VR 寄存器的某个位数据，SET_BIT 指令将 VR 某个位置 1。

2.3.4 MODBUS

MODBUS 寄存器符合 MODBUS 标准通讯协议，分为位寄存器和字寄存器两类。MODBUS 寄存器的数据掉电不保存。

位寄存器：MODBUS_BIT，触摸屏一般称为 MODBUS_0X，布尔型

字寄存器：MODBUS_REG、MODBUS_LONG、MODBUS_IEEE、MODBUS_STRING，触摸屏一般叫 MODBUS_4X，类型如下图。



控制器中 MODBUS 字寄存器占用同一个变量空间，其中一个 LONG 占用两个 REG 地址，一个 IEEE 也占用两个 REG 地址，使用时要注意错开字寄存器编号地址。

MODBUS_LONG(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

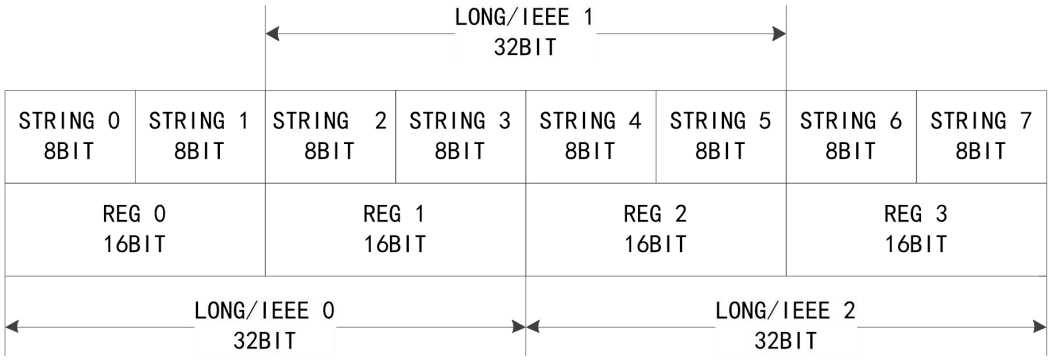
MODBUS_LONG(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

MODBUS_IEEE(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

MODBUS_IEEE(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

所以要注意 MODBUS_REG,MODBUS_LONG,MODBUS_IEEE 地址在用户应用程序中不能重叠。

4X 空间示意图：



例程：

```
MODBUS_REG(0)=0   '初始化置 0
MODBUS_REG(1)=0   '初始化置 0
'modbus_long 赋值 70000
'modbus_reg 范围-32768~32767
MODBUS_LONG(0)=70000
?MODBUS_REG(0),MODBUS_REG(1)
'打印出 reg(0)为 4464，reg(1)为 1
'long(0)=reg(1)*2^16+reg(0)
```

在串口设置（SETCOM 参数）过程中，寄存器选择为 VR 时，此时一个 VR 映射到一个 MODBUS_REG，其中 VR 是 32 位浮点型，MODBUS_REG 是 16 位有符号整数型，从 VR 传递数据给 MODBUS_REG 会丢失小数部分，当 VR 数据超过正负 15 位时，MODBUS_REG 数据会改变；MODBUS_REG 传递数据给 VR 不会有问题，见如下例程，更多信息参见 SETCOM 指令。

例程：

```
VR(0)=0                    '初始化 VR(0)和 MODBUS_REG(0)为 0
MODBUS_REG(0)=0
SETCOM(38400, 8,1,0,0,4,0)   '设置 VR 映射到 MODBUS_REG
VR(0)=100.345              '设置 VR(0)=100.345
?MODBUS_REG(0)               '打印结果为 100，VR 已经映射到 REG，但是
                             'REG 是整型，所以小数部分丢失
MODBUS_REG(0)=200          'REG(0)设为 200
?VR(0)                     '打印结果为 200，REG 变化也会改变 VR
```

当使用 MODBUS 协议与其他设备通讯时，就需要将数据放在 MODBUS 寄存器内进行传递，比如与触摸屏通讯。不进行 MODBUS 通讯时，亦可将 MODBUS 寄存器作为控制器本地数组使用。

控制器直接从 MODBUS_BIT 地址 10000 开始与输入 IN 口对应，20000 与输出 OUT 口对应（注意读取的 IO 是原始的状态，INVERT_IN 反转输入指令不起作用），30000 与 PLC 编程的 S 寄存器对应。

MODBUS_IEEE 地址 10000 开始对应轴 DPOS 区间，11000 开始对应轴 MPOS 区间，12000 开始对应

轴 VP_SPEED 区间；MODBUS_REG 的 13000 开始对应模拟量 DA 输出区间，14000 开始对应模拟量 AD 输入区间。

MODBUS_BIT 地址	意义
0~7999	用户自定义使用
8000~8099	PLC 编程的特殊 M 寄存器
8100~8199	轴 0-99 的 IDLE 标志
8200~8299	轴 0-99 的 BUFFER 剩余标志
10000~14095	对应输入 IN 口
20000~24095	对应输出 OUT 口
30000~34095	对应 PLC 编程的 S 寄存器

MODBUS_REG 地址 MODBUS_IEEE 地址	意义
0~7999	用户自定义使用，可混用 MODBUS_REG、MODBUS_IEEE、MODBUS_LONG
8000~8099	PLC 编程的特殊 D 寄存器
10000~10198	对应各轴 DPOS，读写用 MODBUS_IEEE
11000~11198	对应各轴 MPOS，读写用 MODBUS_IEEE
12000~12198	对应各轴 VPSPEED，读用 MODBUS_IEEE
13000~13127	模拟量输出 AOUT，读写用 MODBUS_REG
14000~14255	模拟量输出 AIN，读用 MODBUS_REG

2.4 多任务

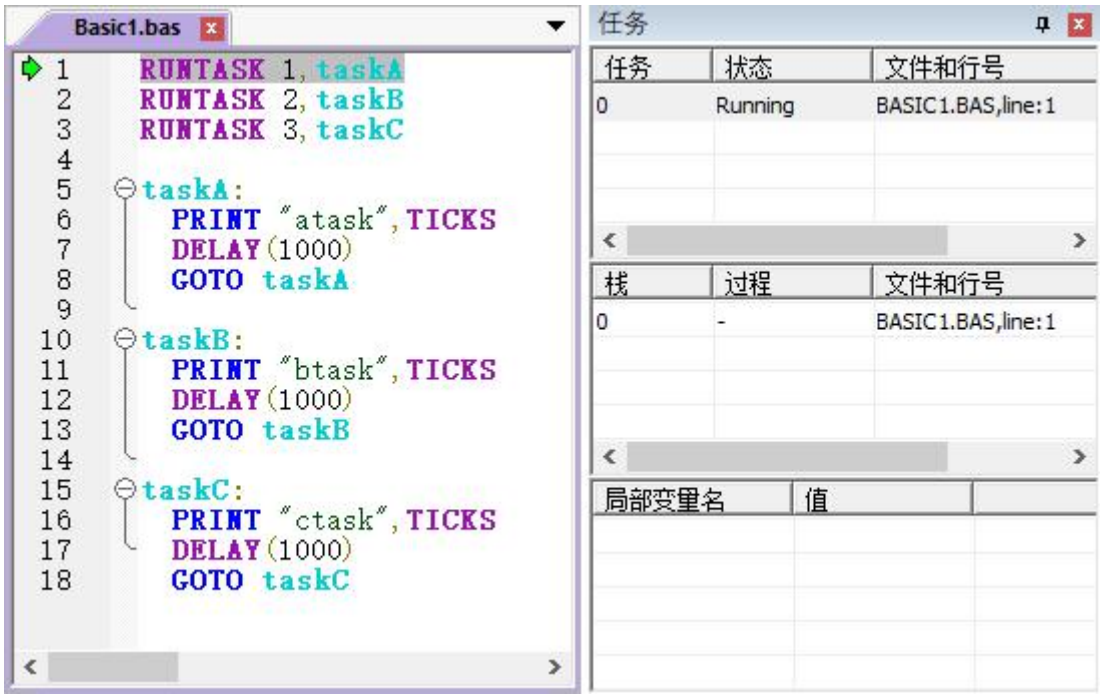
任务是 I/O 刷新和执行用户程序等一系列指令处理的功能。如果多个程序能够互不干扰的同时运行，则称为多任务，一个任务是指一个正在运行的程序。（参见 RUN、RUNTASK 等任务指令描述）

正运动运动器控制均支持多任务使用，每个任务都有自己的编号，此编号没有优先级意义，只是标识当前程序属于哪一个任务，具体任务数量，可在 ZDevelop 软件菜单栏“控制器状态”里查看，如下图，表示该控制器最多支持 26 个任务，任务编号为 0-25。

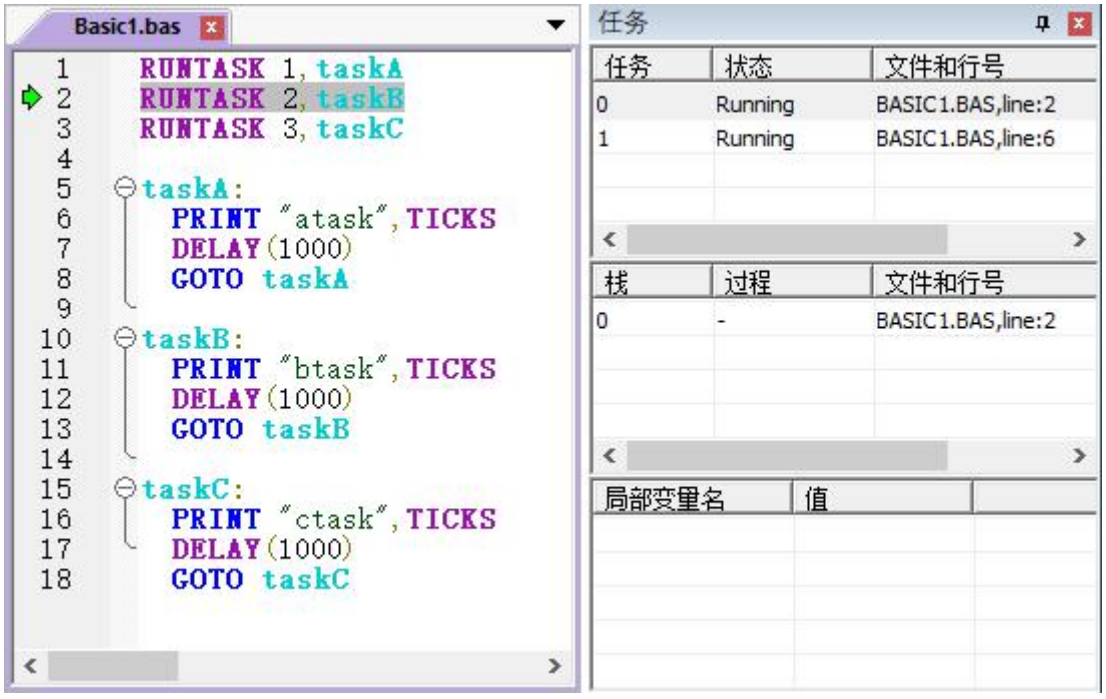
任务本身一旦启动后就与其他任务没有任何关联（在没有共有全局变量，互锁等功能时）。Basic 程序自上而下扫描，扫描到任务便开始执行，本任务时间片到了，任务调度器切换到下一个任务执行，可多个任务同时进行，互不干扰。

如下例程，开始单步调试，程序扫描到 RUNTASK1 后开启任务 taskA，开启后继续扫描下一行 RUNTASK2 开启任务 taskB，RUNTASK3 开启任务 taskC，taskA、taskB、taskC 作为独立的任务分别执行

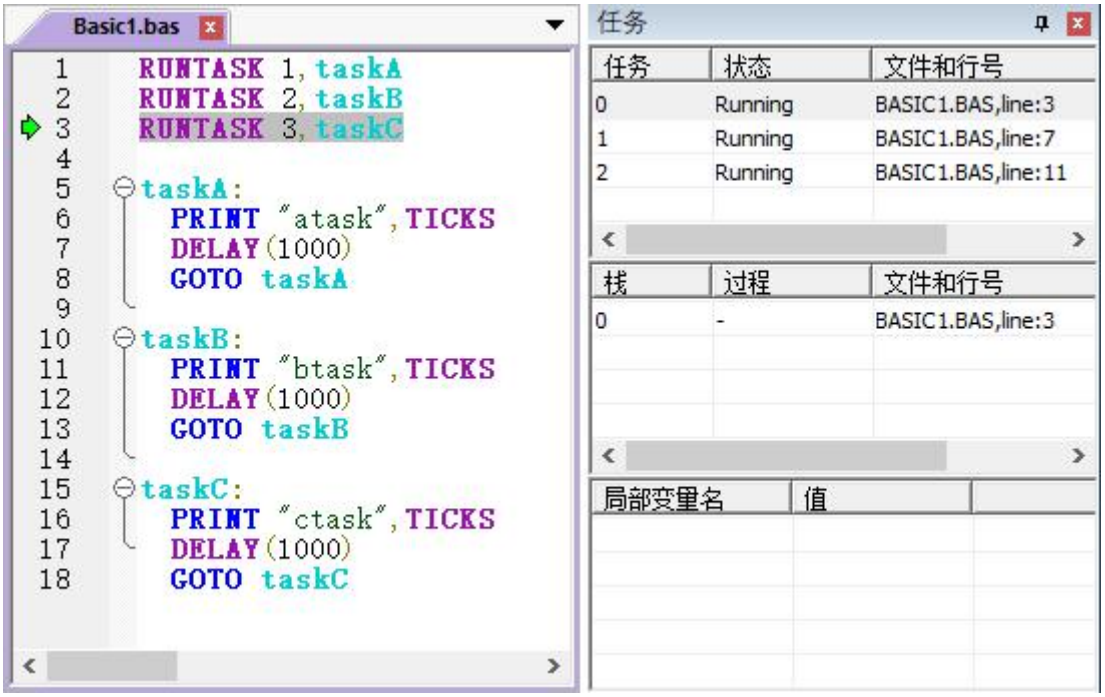
下去。在程序调试窗口可以看到程序执行情况。



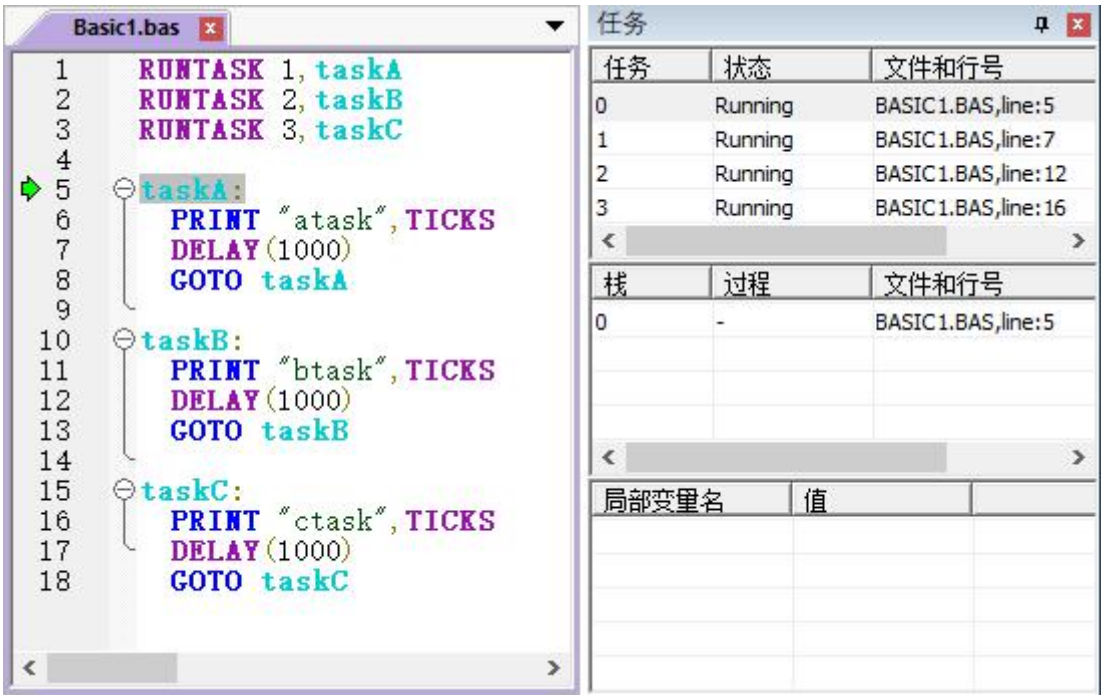
上电只有自动运行的任务 0



任务 0 启动任务 1 运行

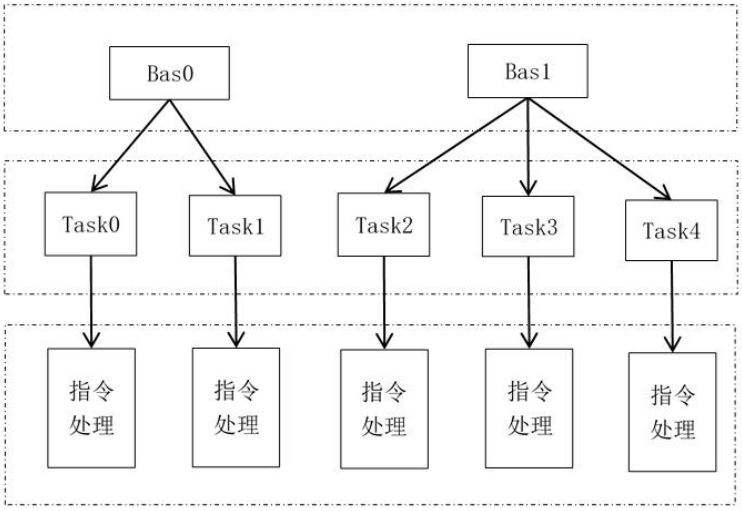


任务 0 启动任务 1、任务 2 运行



任务 0 启动任务 1、任务 2、任务 3 运行

一个项目下可以包含多个 bas 文件、多个 PLC 文件和多个 HMI 文件，每个 bas 文件可以有多个 task，给每个 bas 文件设置不同的任务号之后也可以同时执行，任务之间没有优先级，各个任务独立运行互不干扰，各个 task 之间独立并行，如下图，bas0 文件下包含两个任务，bas1 文件下包含三个任务。



同一项目下任务文件可相互调用，只需将要调用任务的 bas 文件设为主任务，即给该 bas 文件设置自动运行任务号，被调用的 bas 文件不设置自动运行即可。

控制器支持多任务多文件运行，可以一个文件里面启动多个任务，也可以一个任务执行跨到多个文件里面。

多任务操作指令有：

END：当前任务正常结束

STOP：停止指定文件运行的任务

STOPTASK：停止指定任务

HALT：停止所有任务

RUN：启动新任务运行一个文件

RUNTASK：启动新任务运行一个 SUB 或者运行一个带标签的程序

?*MAX：可以查看控制器支持的任务总数与文件总数。

多任务冲突影响：

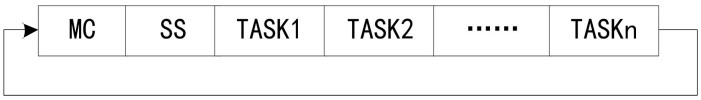
当任务分别在不同的文件上运行时，相互之间不会有影响。如果两个任务运行在同一个文件上，要留意是否共用变量，特别注意是否同时修改同一个变量，例如循环变量。LOCAL 局部变量不受多任务的影响，因此对循环变量请尽量定义为 LOCAL 变量。SUB 传入的参数会自动定义为 LOCAL 变量。

跨文件调用：

跨文件调用时，SUB 必须定义为全局的（GLOBAL）。被调用的 SUB 内部可以访问 SUB 所在文件的变量（已经定义的），或是调用 SUB 所在的其他 SUB，也可以调用访问其它文件的全局内容。

可以将单个文件编译为库文件以供其它程序调用，菜单栏“文件”-“编译”将文件保存为 zlb 格式。

运动控制器每个运动控制周期(Servo Period)包含 MC，SS，以及用户多任务程序的运行，如下图所示：



MC：Motion Control 与 EtherCAT 通讯，Motion Control 包含：单轴运动控制，多轴插补运动，机械手正反解算法，EtherCAT 通讯包含 PDO 通讯与 SDO 通讯。

SS：System Service，包含 RS232 串口通讯，RS485 串口通讯，CAN 通讯，EtherNET 通讯（MODBUS RTU 主从通讯以及 ZDevelop 相关软件服务）

TASK0、TASK1、…、TASKn：对应于各个任务的运行。

若两个任务的动作是互斥的，不可以同时运行，程序中可以加入互锁环节，设置条件可以让其他任务

不运行。

例程：任务互锁

```
GLOBAL DIM   iflock1
iflock1=0      '必须提前初始化
STOPTASK 1                '停止所有任务
STOPTASK 2
RUNTASK 1,TASK1                    '启动任务 1
RUNTASK 2,TASK2                     '启动任务 2
END

'sub 子过程调用
SUB mutex_apply()
    WAIT UNTIL iflock1=0
    iflock1=1
END SUB

SUB mutex_release()
    iflock1=0
END SUB

'任务运行
SUB TASK1()
    WHILE 1
        ?"任务 1 开始运行"
        mutex_apply()
        ' 这里进行可能互斥的操作
        ? "task 1 applyed"
        mutex_release()
        DELAY 1000
    WEND
END SUB

SUB TASK2()
    WHILE 1
        ?"任务 2 开始运行"
        mutex_apply()
        ' 这里进行可能互斥的操作
        ? "task 2 applyed"
        mutex_release()
        DELAY 1000
    WEND
END SUB
```

2.5 三种中断类型

ZBasic 中断分为三种，分别为掉电中断、外部中断、定时器中断。使用中断前必须开启中断总开关，为了避免程序没有初始化好进入中断，控制器上电时中断开关缺省是关闭的。

1．掉电中断：必须是全局的 SUB 函数。控制器只有 1 个掉电中断。掉电中断执行的时间特别有限，只能写少数几条语句，将数据存储在 VR 里。相关函数：INT_ENABLE，ONPOWEROFF。

2．外部中断：可设置上升沿触发或下降沿触发，必须是全局的 SUB 函数，目前只有中断 IN 口 0-31 才可以使用。必须是支持 PLC 功能的固件才可使用，详情请咨询正运动技术人员。相关函数：INT_ONn，INT_OFFn。

3．定时器中断：达到设定时间后执行的功能，必须是全局的 SUB 函数，定时器个数根据控制器型号查看。相关函数：ONTIMERn。

相关指令用法参见第十五章中断相关。

掉电中断例程：

```
INT_ENABLE = 1
dpos(0)=vr(0)       '上电读取保存的数值，恢复坐标
dpos(1)=vr(1)
dpos(2)=vr(2)

'客户应用程序区域

END

GLOBAL SUB ONPOWEROFF ()
    vr(0) = dpos(0)      '保存坐标
    vr(1) = dpos(1)
    vr(2) = dpos(2)
END SUB
```

2.6 G 代码

ZMC 系列运动控制器作为一个多轴运动控制器，支持标准的计算机数控（Computerized Numerical Control，检称 CNC）功能，实现简易的数控机床控制，同时也可应用于其它一些通过 G 代码进行定位及路径规划的场合。

G 代码（G-code）是最为广泛使用的计算机数控编程语言，主要在计算机辅助制造中用于控制自动机床。

ZBasic 支持 G 代码形式的 SUB 过程，支持标准格式的 G 代码。可根据实际加工需求来自定义 G 代码功能，形成 GSUB 形式来解析 CNC 文件。支持 UG、MasterCam、ArtCAM 等多种 CAD/CAM 软件生成的 NC 加工代码，可应用于雕铣机、精雕机、钻攻中心和加工中心等机床加工场合。

G 代码使用方法参见简易例程章节“自定义 G 代码”。

第三章 通讯方式

3.1 串口通讯

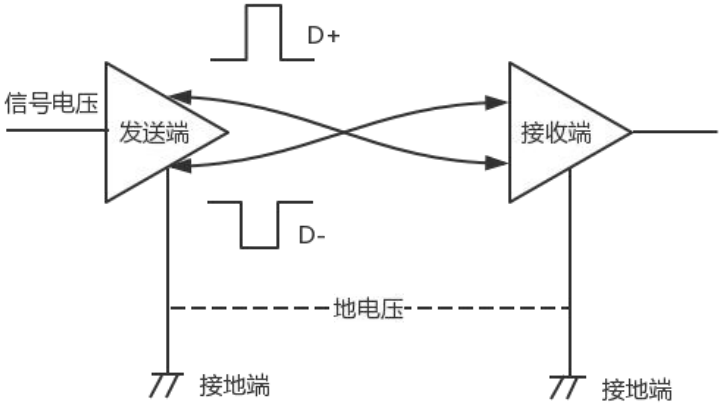
串口是计算机上一种通用的设备通信协议。串口通信的概念较为简单，串口按位（bit）发送和接收字节。尽管比按字节（byte）收发的并行通信慢，但是串口可以在使用一根线发送数据的同时，用另一根线接收数据。结构简单并且能够实现远距离通信。

3.1.1 RS485 串口

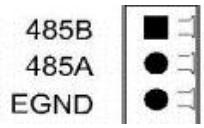
正运动产品的 RS485 通讯口支持 MODBUS 通讯协议的 RTU 模式，可以做 MODBUS 主站或从站，HMI、PLC 或者其它 MODBUS 主站设备可以对运动控制器内部装置进行数据读写操作。

RS485 接口采用差分传输方式，也称作平衡传输，能有效提高抗干扰能力，接线口如下，一根线定义为 A，另一根线定义为 B，发送端：AB 线之间的电平在+2~+6V 为逻辑 1，AB 线之间的电平在-2~-6V 为逻辑 0；接收端：AB 线之间的电平大于 200mV 为逻辑 1，AB 线之间的电平小于-200mV 为逻辑 0

RS485 工作原理：



RS485 无具体的物理形状，可根据实际情况设置合适的接口，如下图所示，控制器的 RS485 接口采用了简易接线方式。接线使用带屏蔽层的双绞线，通讯网络接地，为避免强电对其干扰，同时应避免和强电电路走在一起。



控制器的 RS485 总线接口最大支持 128 节点，即具有多站通讯能力，多个设备用 RS485 连接在一起应采用串联连接，不能使用星型连接和分叉连接。控制器做从站的时候要使用 ADDRESS 指令设置地址。

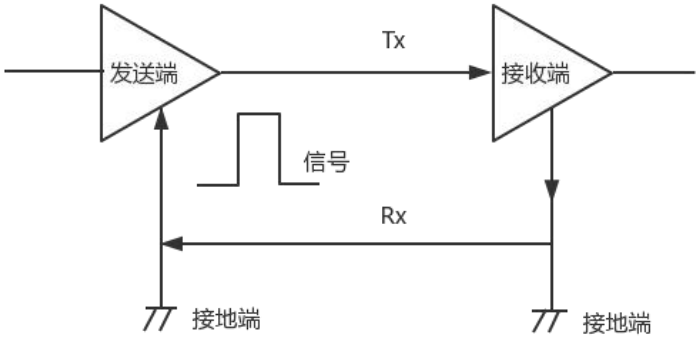
数据传输的速率由 SETCOM 指令设置波特率，最大可选 115200，最大传输距离标准值为 1200m，传输速率与传输距离成反比，传输距离越长，传输速率越低，跟通讯电缆的品质，现场使用环境也有很大关系。RS485 接口是采用平衡驱动器和差分接收器的组合，抗共模干扰能力增强，即抗噪声干扰性好。

3.1.2 RS232 串口

RS232 是最常见的串口，一般个人计算机上会有两组 RS232 接口，分别称为 COM1 和 COM2。

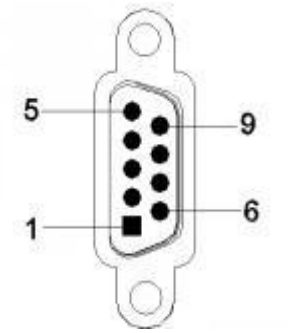
运动控制器的 RS232 通讯口支持 MODBUS 通讯协议 RTU 模式，可以做 MODBUS 主站或从站，HMI、PLC 或者其它 MODBUS 主站设备可以对运动控制器内部装置进行数据读写操作。

RS232 工作原理：



在 TXD 和 RXD 上：采用负逻辑，逻辑 0 表示电压范围为+5~+15V；逻辑 1 表示电压范围为-5~-15V。

RS232 常用的是 DB-25 和 DB-9 两种接口，控制器采用 DB-9，针脚信号说明如下：



针脚号	名称	功能	信号传输方向
1	DCD	载波检测	DTE←DCE
2	RXD	接收数据（串行输入）	DTE←DCE
3	TXD	发送数据（串行输出）	DTE→DCE
4	DTR	DTE 就绪（数据终端准备就绪）	DTE→DCE
5	GND	信号地	-
6	DSR	DCE 就绪（数据建立就绪）	DTE←DCE
7	RTS	请求发送	DTE→DCE
8	CTS	允许发送	DTE←DCE
9	E5V	外部电源 5V 输出	-

RTS：请求发送，用来表示 DTE 请求 DCE 发送数据，当终端要发送数据时，使该信号有效，状态为 ON，向 MODEM 请求发送，用来控制 MODEM 是否要进入发送状态。

CTS：允许发送，用来表示 DTE 准备好接收 DCE 发来的数据，是对请求发送信号 RTS 的响应，当 MODEM 已经准备好要接收终端传来的数据，并向前发送是，使该信号有效，通知终端开始沿发送数据线 TXD 发送数据。

DCD：接收线信号检出，用来表示 DCE 已接通通信链路，告知 DTE 准备接收数据，当本地的 MODEM 收到由通信链路另一端的 MODEM 送来的载波信号时，使 RLSD 信号有效，通知终端准备接收，并且由 MODEM 将接收下来的载波信号解调成数字两数据后，沿接收数据线 RXD 送到终端。

与电脑连接需要采用双母头的 2.3 交叉线。RS232 传输二进制数据采用的是起止式异步串行通讯协议，

异步串行通讯可靠性高，因为接收方是根据起始位判断一个字符传输得到开始时间，即使收发双方的时钟频率有偏差，也不会因偏差累计而导致错位，但是由于每个字符都要有起始位和停止位等信息，导致传输速率较低。

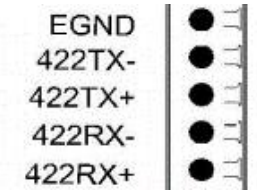
RS232 通讯距离越长，波特率越小，最小为 9600，传输距离也有限，在通讯环境较好的情况下传输距离可达 15m 左右，接口采用信号发送线与信号返回线构成的共地传输形式，容易产生共模干扰，所以抗噪声干扰性较弱。

3.1.3 RS422 串口

仅 ZMC3 系列的部分控制器型号带 RS422 串口。

RS422 是传统 Apple 计算机的串口连接标准，它定义了接口电路的特性，数据传输特性与 485 相同。RS-422 采用四线制，分别标示为 RX+/RX-（接收信号），RT+/RT- （发送信号），一根信号地线，共 5 根线，四线接口由于采用单独的发送和接收通道，因此不必控制数据方向。

如右图所示，控制器的 RS422 接口采用了简易接线方式，但相比 RS485 和 RS232，布线成本高，接线容易搞错。控制器的 RS422 接口仅支持接入一个设备。



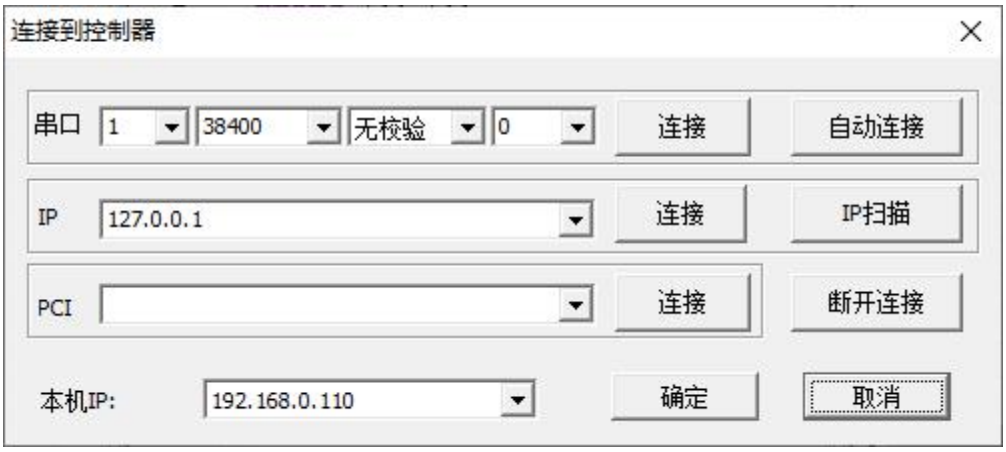
RS485 是 RS422 的改进，对 RS422 的功能进行了扩展，故 RS422 是 RS485 的子集，因而所有 RS422 设备可以受 RS485 控制。

RS232、RS422、RS485 总线对比：

串口类型	RS232	RS422	RS485
工作方式	单端	差分	差分
最多支持节点数	1 收 1 发	1 收 1 发	1 收 128 发
最大传输距离	15 米	1200 米	1200 米
最大传输速率	115200bit/s	115200bit/s	115200bit/s
通信方式	全双工双向通信	全双工双向通信	半双工双向通信

3.1.4 串口接线方法

将控制器与需要连接的设备进行物理连接，选择合适的串口对应连接起来，物理连接完成后打开 ZDevelop 软件，点击菜单栏的“连接”，弹出下面窗口：



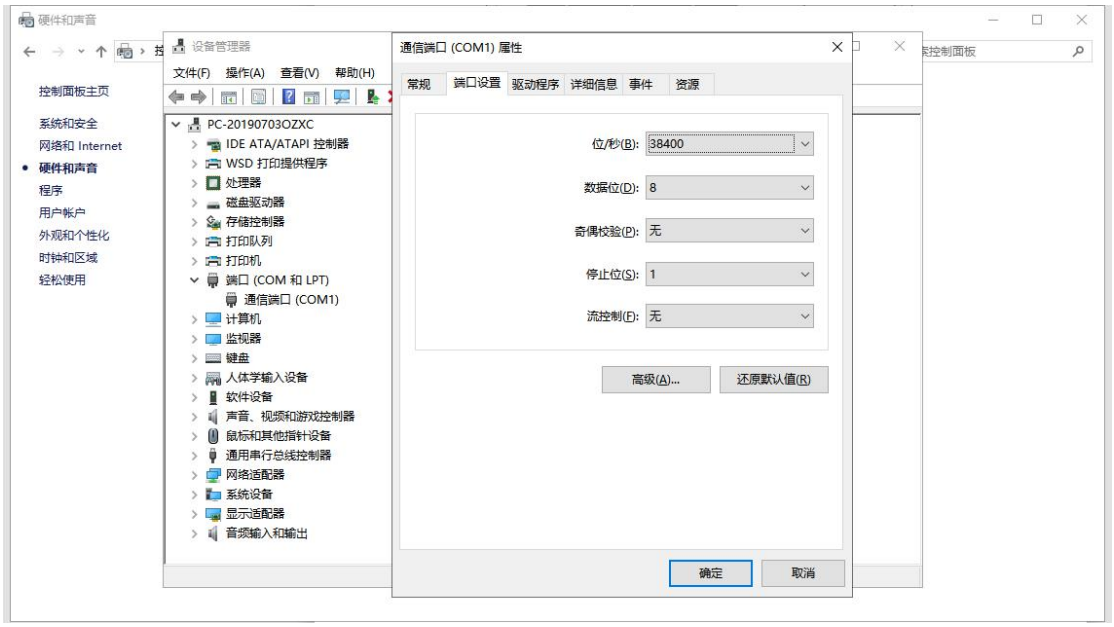
第一行为串口连接方式，选择需要连接的串口编号、设置波特率、校验位、停止位之后，点击连接，连接是否成功会在软件输出窗口自动打印出相应信息。

若连接失败，按下面方法依次排查：

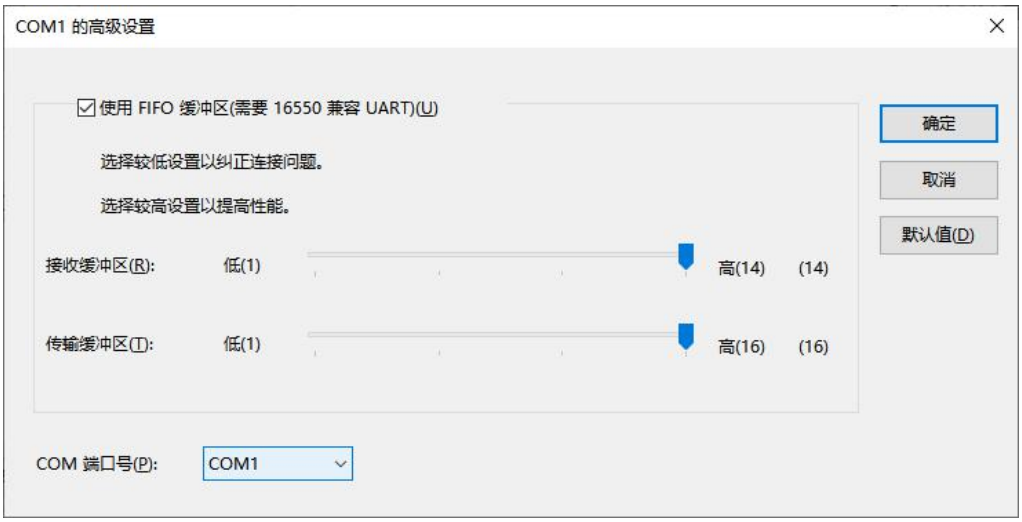
1．查看串口连接线是否为交叉线。

2．“连接到控制器”里的 COM 口编号、参数是否选择正确。

3．打开电脑“设备管理器”-“端口”-“通信端口（COM）”-“端口设置”，查看 COM 口设置是否正确，控制器串口默认参数 波特率 38400，数据位 8，停止位 1，校验位无。



在“端口设置”-“高级”选项中可更改 com 端口号，通过下拉列表选择。



4. 当通过串口连接到控制器时，对应的控制器串口必须配置为 MODBUS 从协议模式（缺省模式），断电重启即可恢复。

5. COM 口是否已被其他程序占用，如串口调试助手等。

6. PC 端是否有足够的串口硬件。

7. 更换串口线/电脑测试。

3.1.5 USB 接口

正运动大部分运动控制器都带有一个标准 USB 接口。

USB 是一种高速串行传输总线，是一个外部总线标准，用于规范电脑与外部设备的连接和通讯。是应用在 PC 领域的接口技术。

USB 具有传输速度快、使用方便、支持热插拔、连接灵活、独立供电等优点，可以连接键盘、鼠标、大容量存储设备等多种外设，该接口也被广泛用于智能手机中。计算机等智能设备与外界数据的交互主要以网络和 USB 接口为主。

控制器的 USB 接口仅支持接 U 盘，不支持 USB 线与其他设备通讯。

控制器可上传或下载数据到 U 盘，还可以使用相关指令可以将 U 盘数据保存到 TABLE 或 VR 寄存器，使用 FILE 指令可对 U 盘进行多种操作。不管在何种编程环境下，均可使用 U 盘在各个控制器之间拷贝和下载程序，还可以用于升级固件时存储固件文件。

U 盘操作参见“U 盘相关指令”章节。

3.2 网口通讯

运动控制器按有无网口可划分为两类，不带网口、带网口，其中带网口的控制器网络接口有 EtherNET 和 EtherCAT 两种，有的控制器只带其中一种接口，有的控制器两种接口都带，具体带网口情况参见控制器硬件手册说明，只有带 EtherCAT 口的控制器才支持 EtherCAT 总线协议。

EtherNET 网口采用 TCP/IP 协议，通讯引脚定义：

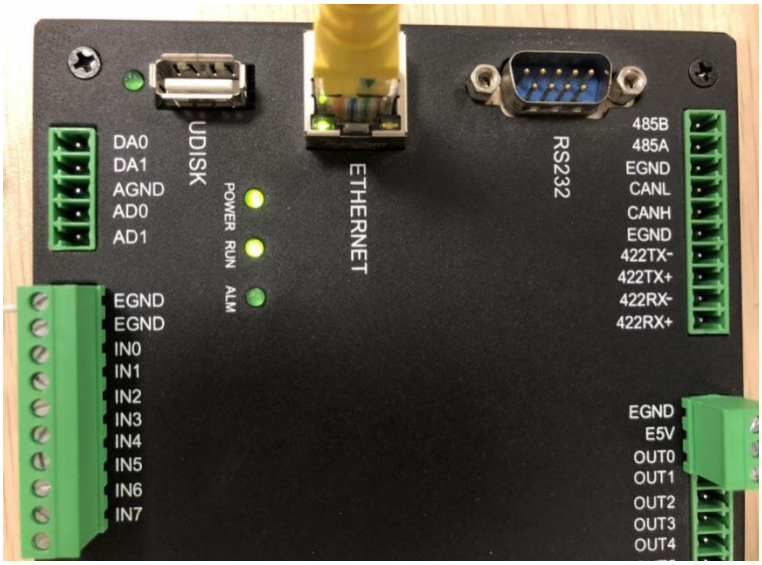
引脚	信号	描述
1	Tx+	传输数据正极
2	Tx-	传输数据负极

3	Rx+	接收数据正极
4	保留	保留
5	保留	保留
6	Rx-	接收数据负极
7	保留	保留
8	保留	保留

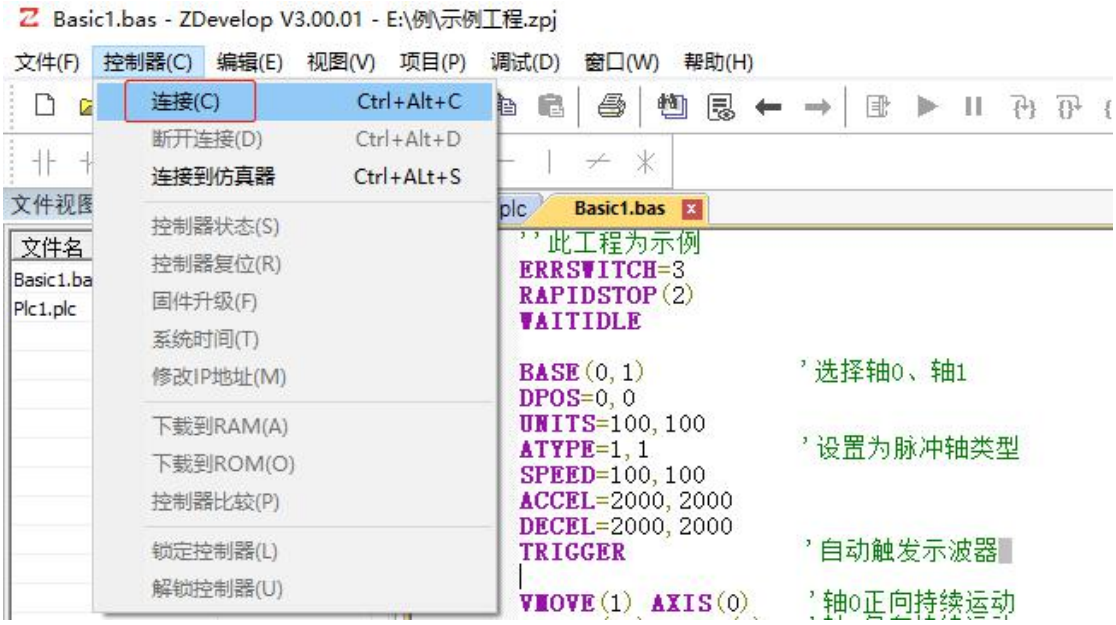
网口连接步骤：

1．控制器上电：将控制器接入 24v 电源，此时控制器上 POWER 和 RUN 对应的指示灯亮，表示控制器上电成功。

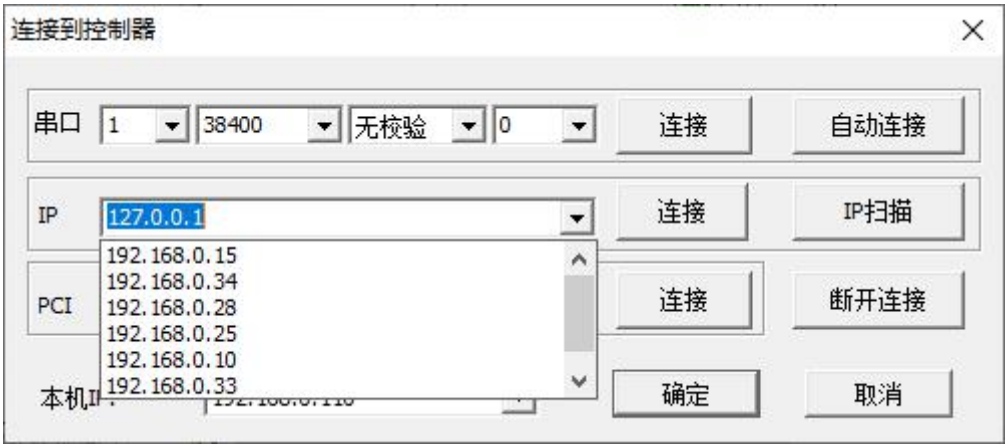
2．控制器通过网线连接电脑：将网线插入 EtherNET 接口后，网口的指示灯亮表示网口接线成功。如下图所示，两处的灯都亮方可进行下一步。



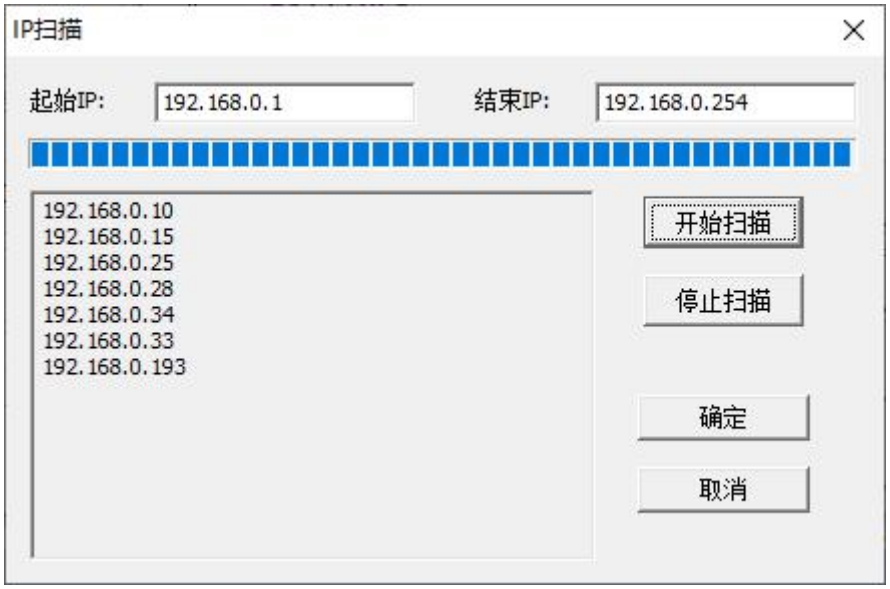
3．软件连接：线路连接无误后，点击 ZDevelop 软件菜单栏“控制器”-“连接”，如下图所示，或直接点击  按钮，弹出“连接到控制器”窗口，默认 IP 地址为 192.168.0.XXX，软件自动获取可用控制器 IP 地址，在弹出的窗口里手动输入或使用自动扫描的结果，选择正确的控制器 IP 地址连接即可。



手动输入 IP 地址：输入框里直接输入，或在下拉列表里选择软件自动扫描出来的可用 IP 地址。



自动扫描 IP 地址：如果软件无法自动扫描可用 IP 地址（即输入框下拉列表不显示数据），此时可采用手动扫描方式，点击“IP 扫描”-“开始扫描”，此时上图输入框下拉列表里就有了扫描出的 IP 数据供选择。



控制器连接失败可能故障原因及解决办法：

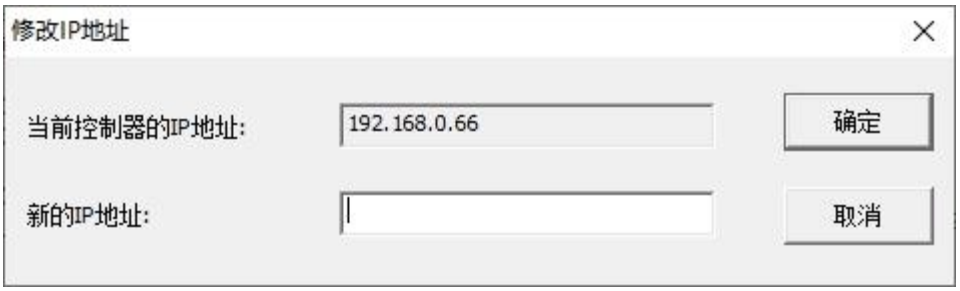
序号	故障描述	解决办法
1	控制器接入电源后 POWER 和 RUN 指示灯不亮	检查电源有无问题 若电源正常检查控制器是否烧坏
2	控制器上电后接入网线，网口指示灯不亮	查看网线两端是否插好 网线本身是否损坏 网线插线槽是否损坏 注意网线是接入 EtherNET 口，而不是 EtherCAT 口
3	连接控制器失败	检查控制器 IP 地址与 PC 是否处于同一 IP 段，需要处于同一网段才可连接使用，详细修改方法参见下页 控制器通道全部被占用，建议把不用的暂时关闭后尝试连接 电脑如果卡顿严重，建议尝试多次软件连接 重启控制器再次重复以上连接步骤
4	连接成功，在使用途中偶尔会掉线	网线建议采用如下图所示，金属接头带屏蔽层的网线。在强干扰的情况下，使用水晶头不带屏蔽层的网线会导致通讯不稳定，偶尔发生掉线的情况。 

电脑 IP 地址修改方法：

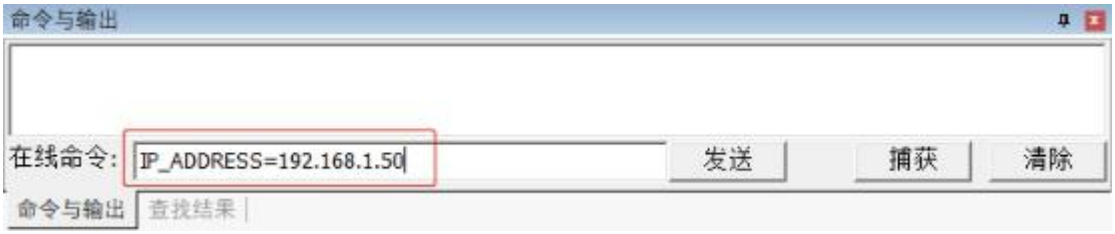
1．查看电脑本地 IP 协议版本 4 地址是否为 192.168.0.xxx，前三段与控制器一致，最后一段不能一样，控制器出厂默认 IP 192.168.0.11。如果 IP 地址的第三段不一样，则需要把对应的子网掩码改为 0。设置好之后再进行软件连接。

如果控制器 IP 被修改，不处于 192.168.0.XXX 这个网段，此时只能先通过串口连接控制器，然后在“控制器”-“控制器状态”来获取控制器 IP 地址，修改本机 IP 或控制器 IP，或控制器 IP 使二者处于同一网段。

修改控制器 IP 地址方法有多种，可点击菜单栏“控制器”-“修改 IP 地址”，弹出如下窗口，此时会显示当前控制器 IP，在窗口可直接输入新的 IP 地址。



或在 ZDevelop 菜单栏“控制器”-“控制器状态”查看或在线命令获取控制器 IP 地址，控制器 IP 用指令 IP_ADDRESS 修改。



修改 IP 后，控制器与 ZDevelop 的连接会断开，此时再次选择新设置的 IP 地址连接即可。

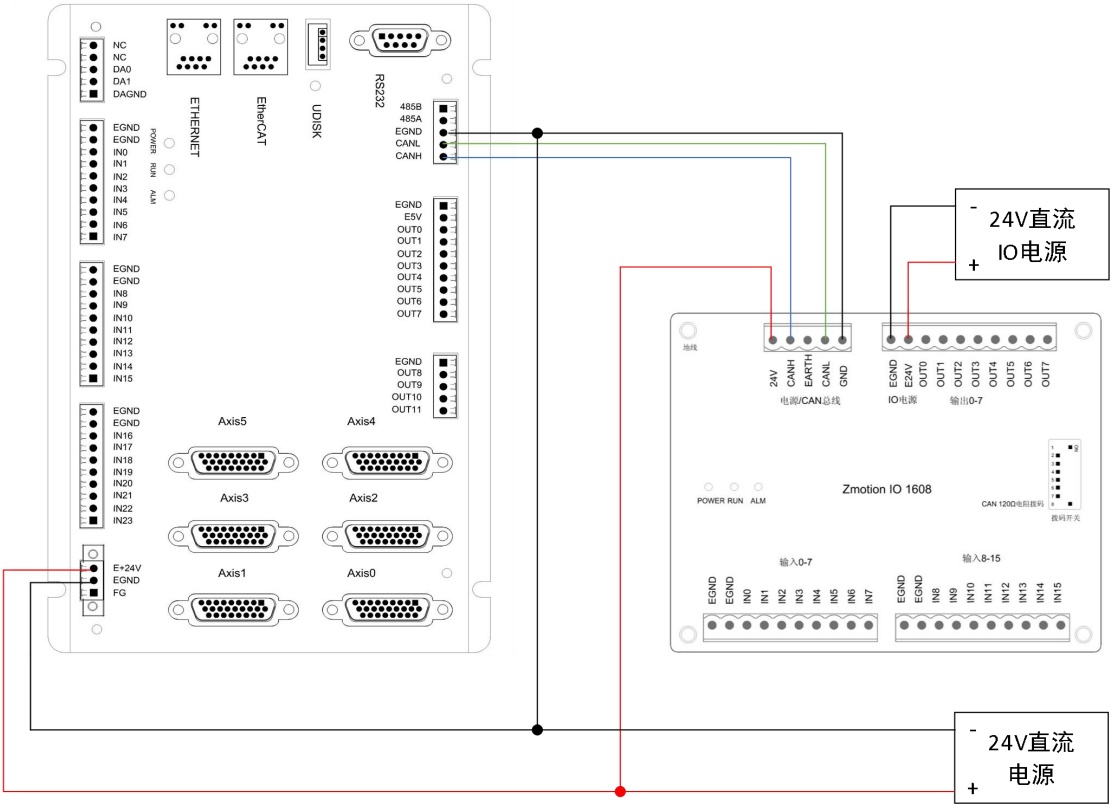
3.3 CAN 总线通讯

运动控制器支持 CAN 总线通讯，可以采用 CAN 总线连接扩展模块和其他控制器，CAN 接口引脚如下：

针脚号	名称	说明
1	GND	内部电源地
2	CANL	CAN 差分数据-
3	EARTH/SHIELD	安规地/屏蔽层
4	CANH	CAN 差分数据+
5	+24V	内部电源 24V 输入

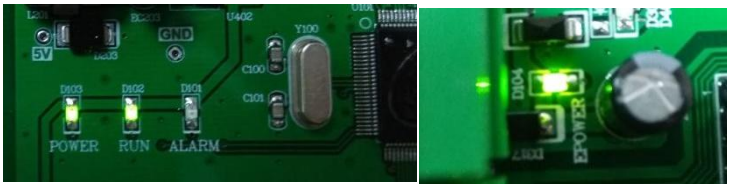
CAN 扩展接线方法：

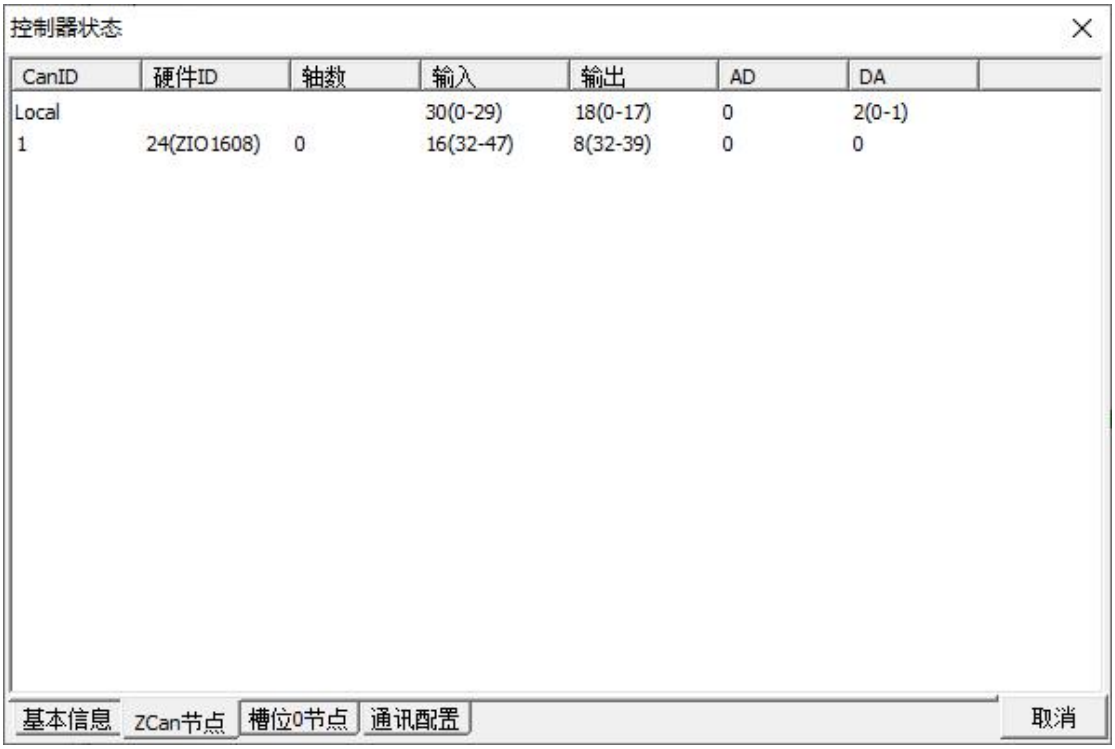
1．接线如下图所示，将扩展模块的 CANL 和 CANH 对应连接控制器的 CANL 和 CANH，CAN 总线通讯双方的 GND 端连接在一起，均接入控制器电源负极，将控制器和扩展模块+24V 端接入控制器电源正极，带拨码开关的直接将第 8 位拨为 ON 即表示接入 120 欧姆电阻，不带拨码开关的，在扩展模块的 CANL 和 CANH 之间接一个 120 欧姆的电阻，防止扩展模块被烧坏。



2．在扩展模块的 IO 输出端接入另一个 24V 电源 2，给输出口独立供电。

3．电源接通后，接线电阻拨码都设置正确，扩展板块上的电源指示灯（POWER）会和运行灯（RUN）亮，IO 电源灯（EPOWER）亮，报警灯（ALM）不亮。同时 ZDevelop 软件的“控制器”-“控制器状态”-“ZCan 节点”显示扩展板信息。





4．ALMRM 指示灯亮请检查接线，电阻以及拨码设置是否正确，以及控制器的 CANIO_ADDRESS 指令是否设置为主端（32）。

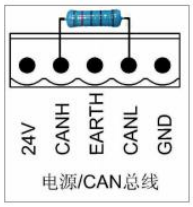
注意事项：

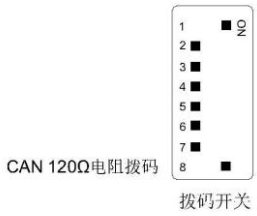
1．CAN 总线通讯双方必须保证对应 GND 连上或是控制器主电源和扩展模块主电源使用同一个电源。控制器和扩展模块用不同电源供电时，CAN 总线通讯双方必须保证对应 GND 连上，否则可能烧坏 CAN 芯片。

2．请把内部电源 24V 和外部数字量 IO 电源 24V 分开供电，特别是现场电磁干扰严重的情况下，必须采用两个 24V 电源，或是一个能提供两路隔离 24V 输出的电源；当通过串口连接触摸屏时，触摸屏的电源使用内部电源 24V 来提供。

3．现场电磁干扰严重的情况下以及同时接了 3 个及以上扩展版时，最好使用双绞屏蔽线来连接 CANL，CANH，防止通讯掉线。（这里用网线做下演示，实际应该用专门的 2 芯双绞线，如下右图）

4．CAN 总线上连接多个扩展模块时，需要在最两边扩展模块的 CANL 与 CANH 端并接一个 120 欧姆的电阻如下左图所示。V1.3 以上硬件版本带 8 位拨码开关的扩展模块，如下右图所示，扩展模块上集成了 120 欧姆电阻在 CANL 和 CANH 之间。由拨码 8 控制，只需要把拨码 8 拨为 ON，电阻即可接通，不需要另外在端子外部接 120 欧姆电阻。





拨码开关 1-8 名称和说明：

拨码	名称	说明
1	ID0	CAN 地址拨码
2	ID1	CAN 地址拨码
3	ID2	CAN 地址拨码
4	ID3	CAN 地址拨码
5	ID4	CAN 速度拨码
6	ID5	CAN 速度拨码
7	SPEC	特殊功能预留
8	120 欧姆	CAN 120 欧姆电阻拨码

V1.3以上硬件版本IO板上一共有8个拨码，1-4设定CAN地址，5、6设定CAN速度，7预留，8为120欧电阻拨码，拨ON时电阻接通。

拨码每位OFF时对应值0，ON时对应值1，组合值=拨码4×8+拨码3×4+拨码2×2+拨码 1，控制器根据CAN拨码地址来设定对应IO板的IO口范围。（可以通过查看ZDevelop软件的控制器状态窗口来查看对应的IO起始编号）

拨码 1-4 选择 CAN 地址，数字量 IO 编号分配表：

组合值	起始 IO 编号	结束 IO 编号
0	16	31
1	32	47
2	48	63
3	64	79
4	80	95
5	96	111
6	112	127
7	128	143
8	144	159
9	160	175
10	176	191
11	192	207
12	208	223
13	224	239

14	240	255
15	256	271

扩展版拨码开关根据当前已包含 IO 点数（IN 和 OP 最多的那个），使用 1-4 号拨码设置 ID，如控制器本身包含 28 个 IN，16 个 OP，那么第一个扩展版设置的起始地址应超过最大值 28，按上表规则应将拨码设置为组合值 1（二进制组合值 0001，从右往左对应拨码 1-4，此时拨码 1 置 ON，其他置 OFF），此时扩展版上的 IO 编号=扩展版编号值+起始 IO 编号值，其中，29-31 空缺出来的 IO 编号舍去不用。后续的扩展版则依次按 IO 点数继续确认拨码设置。

当控制器或扩展模块的IO编号范围重复时，只有一个有效。建议重新设置拨码使编号不重复。

拨码5-6选择CAN速度：

组合值	说明
0	速度 500KBPS
1	速度 250KBPS
2	速度 125KBPS
3	速度 1MBPS

3.4 触摸屏通讯

此处只讲解控制器与威纶触摸屏连接方法，串口接线定义请参考 3.1 串口通讯章节，以及对应的控制器硬件手册和威纶触摸屏手册。

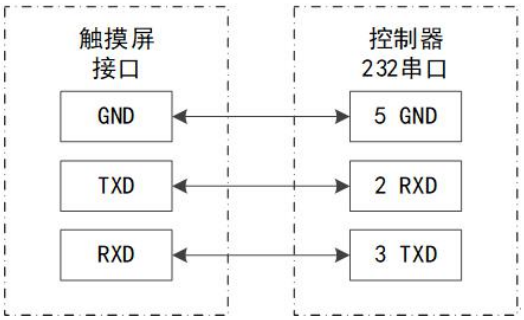
威纶触摸屏与控制器的 MODBUS 地址为一一对应关系。0X——MODBUS_BIT，4X——MODBUS_REG/LONG/IEEE(根据数据类型）。

部分厂家的触摸屏与控制器的 MODBUS 地址关系不是一一对应的，使用时需要注意。

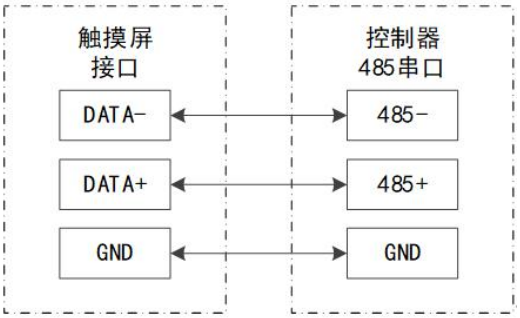
控制器程序指 ZDevelop 编写的程序，下载到控制器，PC 程序无法下载到控制器，需要配合计算机调用正运动动态库使用。触摸屏程序指配套触摸屏编程软件编写的程序，下载到触摸屏，触摸屏与控制器使用串口连接时，两端接口定义要一致。使用网口连接时，直接用一根网线连接即可。

不同型号触摸屏的串口针脚号排布不同，接线时按照如下规则一一连接针脚号即可。

RS232 串口连接：



RS485 串口连接：

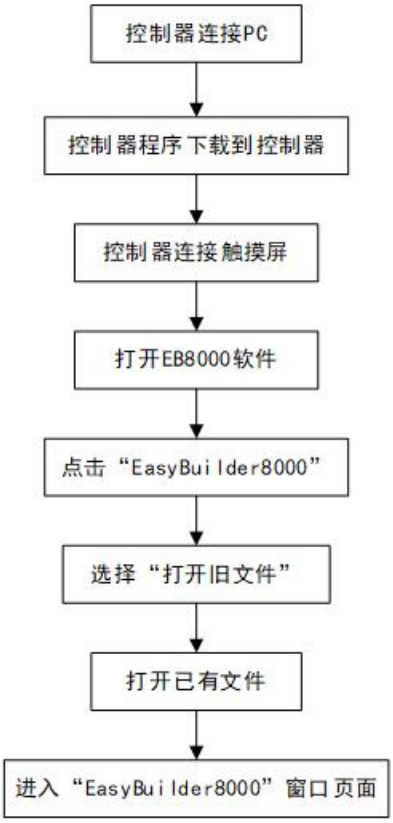


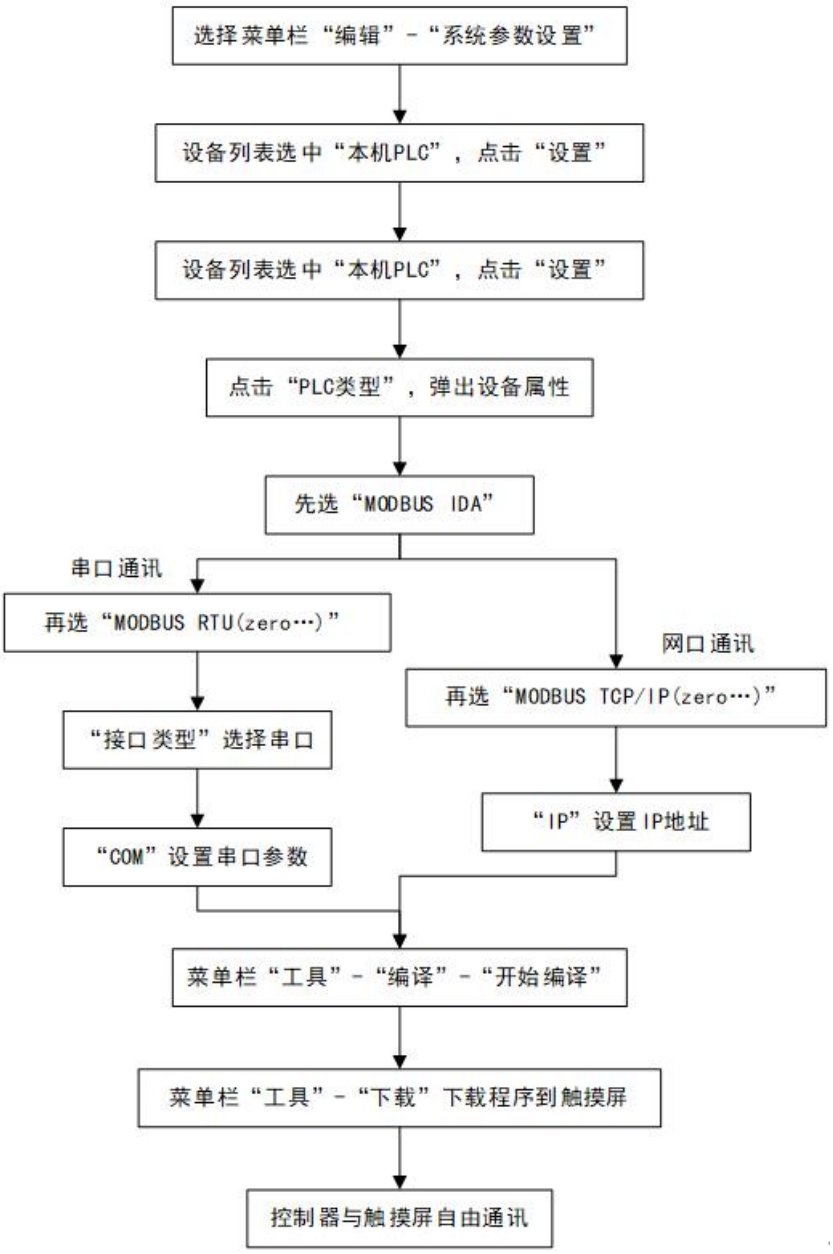
网线接口定义：

针脚号	名称	描述
1	TX+	发送数据+
2	TX-	发送数据-
3	RX+	接收数据+
4	n/c	未使用
5	n/c	未使用
6	RX-	接收数据-
7	n/c	未使用
8	n/c	未使用

3.4.1 控制器与触摸屏连接

控制器与触摸屏连接流程图如下，控制器和触摸屏分开编程，连接完成后控制器和触摸屏可脱离 PC 机进行通讯。


控制器连接PC
控制器程序下载到控制器
控制器连接触摸屏
打开EB8000软件
点击“EasyBuilder8000”
选择“打开旧文件”
打开已有文件
进入“EasyBuilder8000”窗口页面



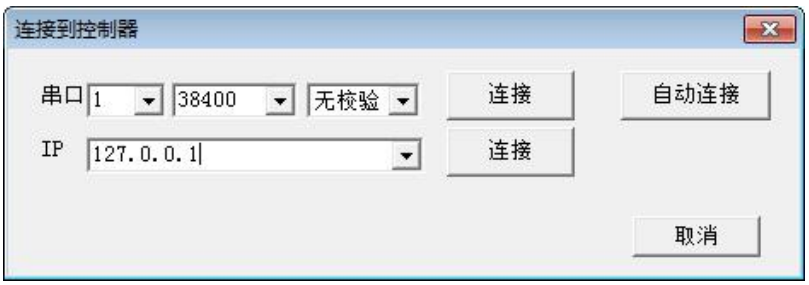
详细连接流程如下：

1．程序下载到控制器

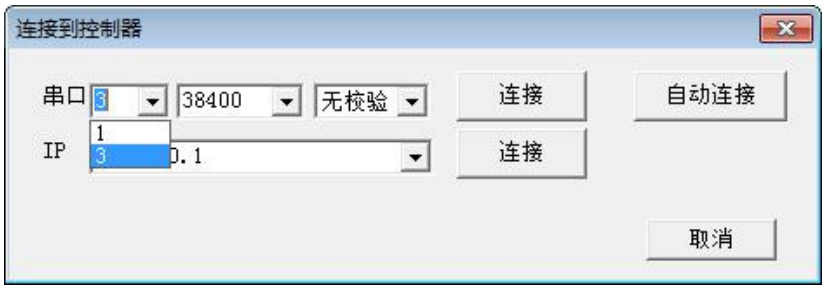
将控制器接好电源，确认控制器“POWER”灯亮起，用 RS232 串口线或网线将控制器与 PC 连接。

打开 ZDevelop 软件，点击“文件”-“打开项目”，选择示例文件。

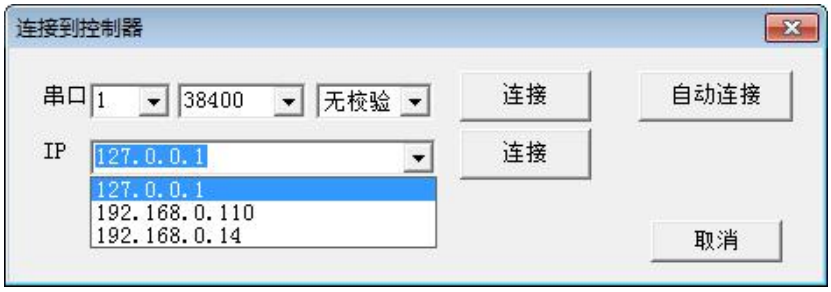
点击“控制器”-“连接”，有串口和 IP 两种设置，如果使用的是串口连接，需要设置 PC 端的端口参数；如果使用的是网口连接，需要设置 PC 端的 IP 地址。



控制器串口默认波特率为 38400，数据位 8bit，无校验，停止位 1bit。根据这个在设备管理器里设置好 PC 端口参数。



控制器默认的出厂 IP 为 192.168.0.11，需将 PC 的 IP 地址设为与控制器处于同一网段，PC 端 IP 地址修改。



PC 端串口参数或者 IP 地址设置好后，点击对应方式“连接”，命令行出现“Connected to Controller”表示连接成功。

再点击 “下载到 ROM”，出现“Down to Controller Rom Success”表示程序已经下载到控制器。

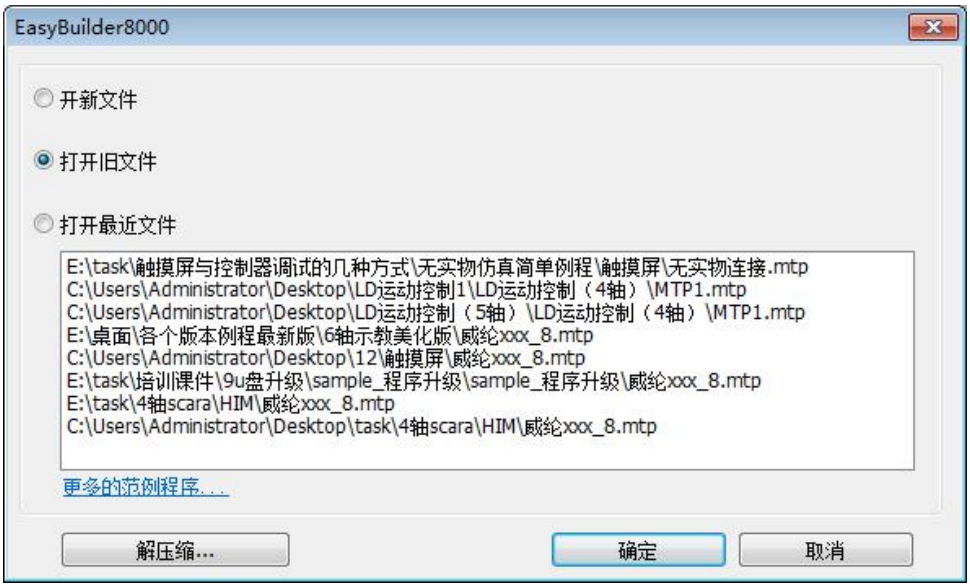
程序下载成功后，如果希望能实时查看运行参数，就不需要断开 ZDevelop 软件连接，但是会占用控制器的一个接口；如果不需要查看运行时参数，可点击 “断开连接”，并拔掉连接线。

详细连接方法参见“编程软件使用”章节以及“串口接线”、“网口接线”的详细连接方法。

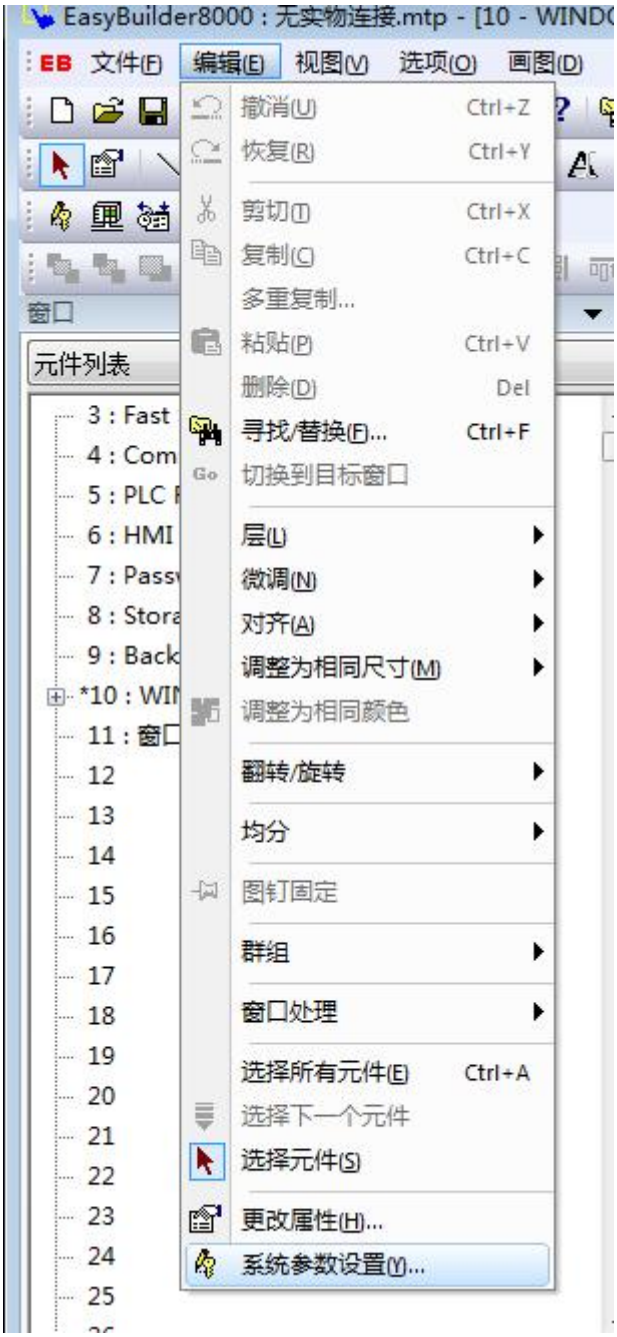
2．控制器与触摸屏连接

将威纶触摸屏通过网线或者串口线（根据触摸屏型号）与控制器连接。接通触摸屏电源，确认触摸屏亮起。通过 USB 线（推荐）或者网线连接触摸屏与 PC 端，用以下载触摸屏程序，打开 EB8000 软件，出现如下窗口。

点击“EasyBuilder8000”。

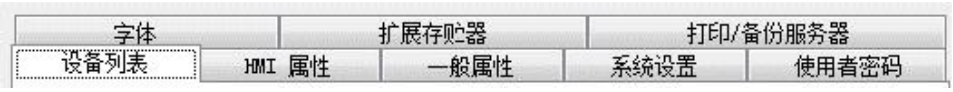


选择“打开旧文件”后确定，选择已有的文件打开。



打开目标文件后选择菜单栏“编辑”-“系统参数设置”。

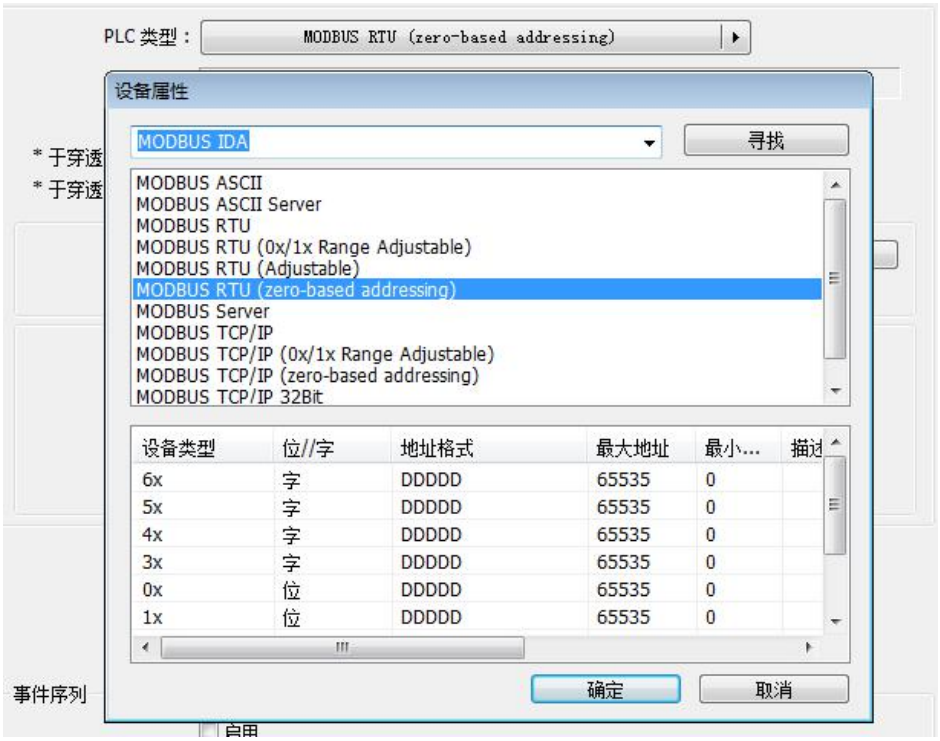
选择设备列表。



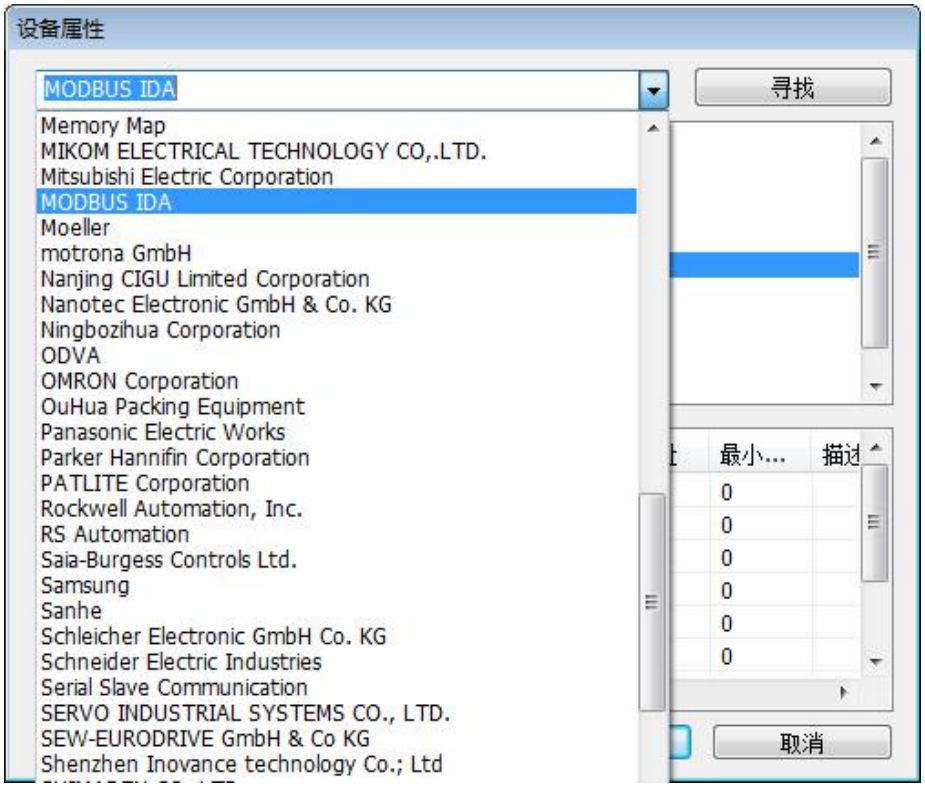
设备列表编号栏，将显示“本机触摸屏”和“本机 PLC”。如果没有“本机 PLC”请点击“新增”；如果有“本机 PLC”，请点击“设置”。本例点击设置。

弹出设备属性界面。

选择 PLC 类型，弹出对话框。

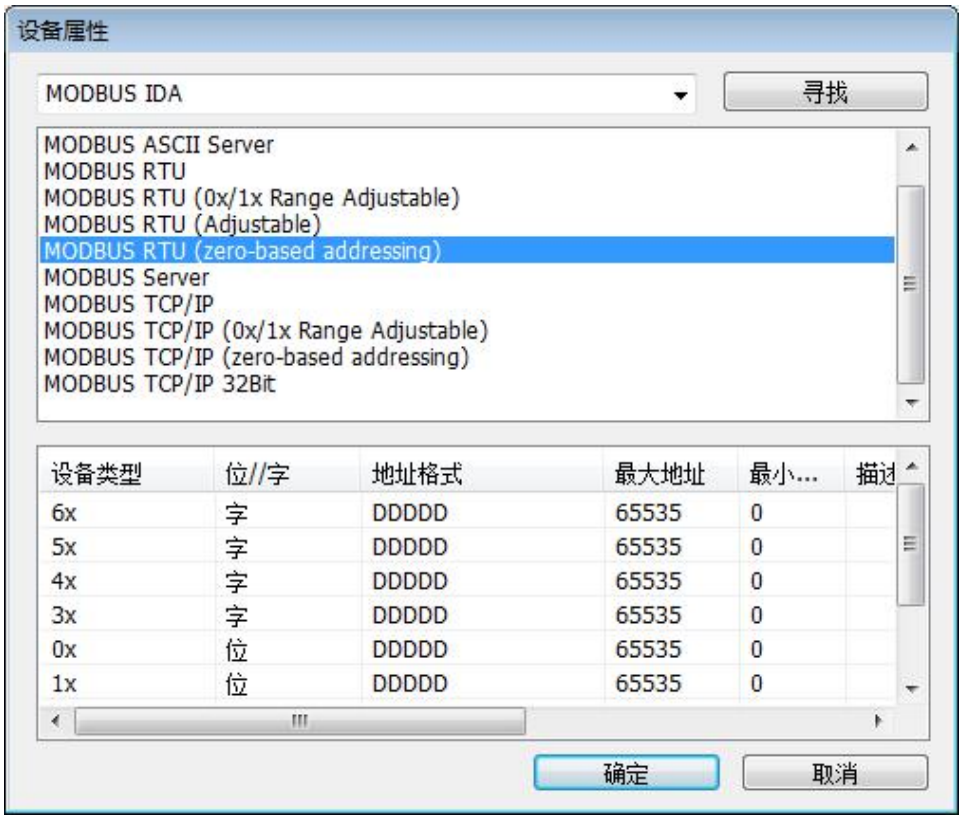


先选 MODBUS IDA 通讯协议。

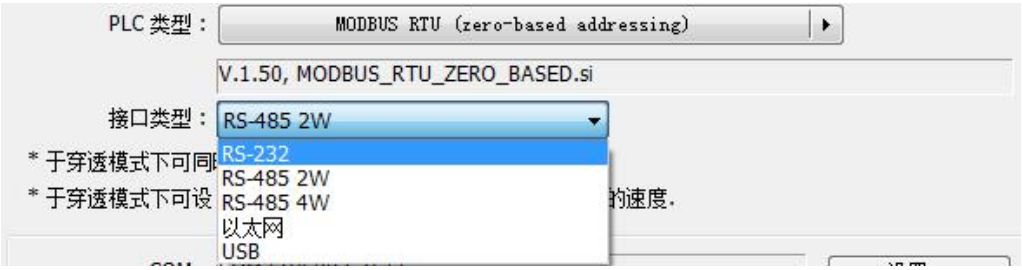


1）串口连接设置

如果使用串口线连接触摸屏与控制器，则选择下图模式。



此时接口类型自动根据控制器的串口类型选择，本例为 RS-232。

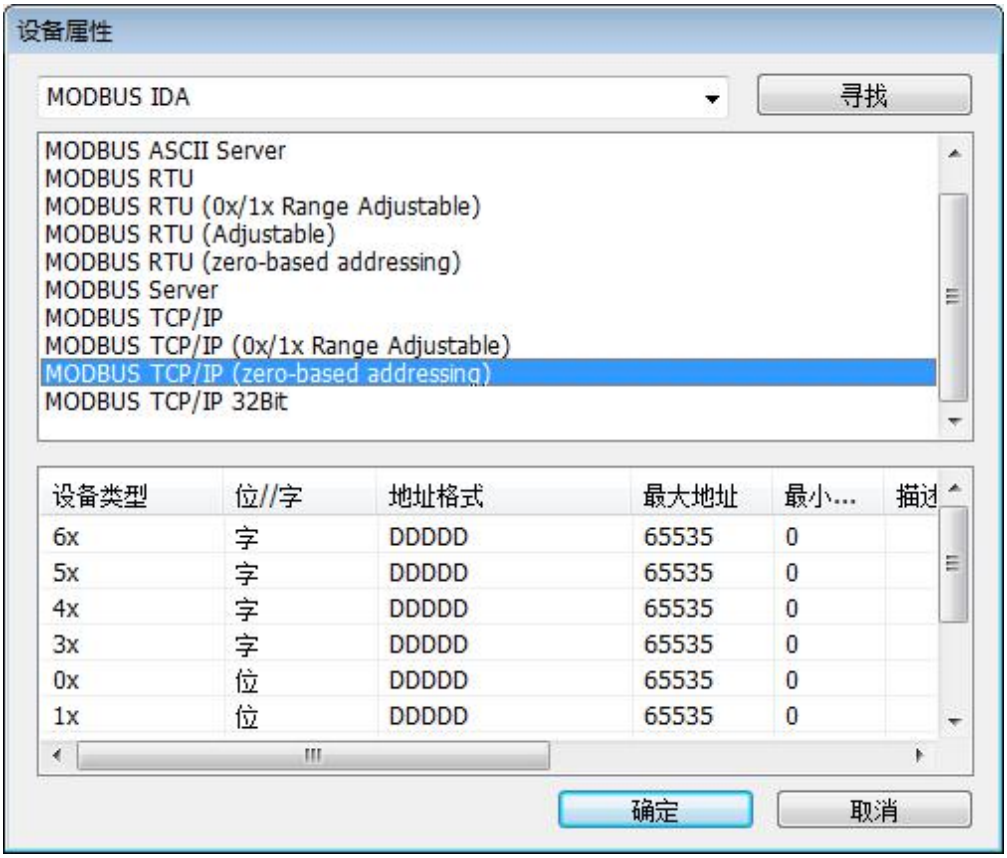


然后进行 com 口设置，点击“设置”。数据改为 COM2(9600,N,8,1)。

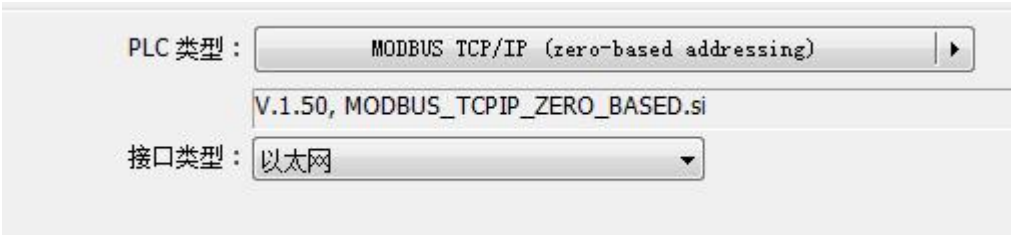
弹出通讯端口设置窗口，此时通讯端口的参数必须与连接到控制器的端口参数一致，控制器 RS232 串口参数：波特率 38400，数据位 8bit，停止位 1bit，无校验。

2）网口连接设置

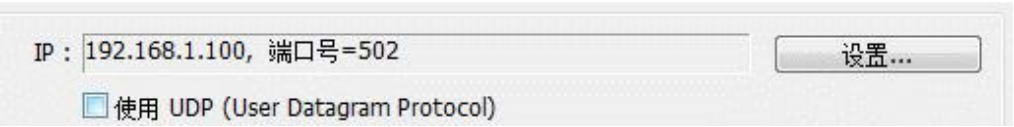
如果以网线连接触摸屏与控制器，选择下图模式。



此时接口类型自动会装换为以太网。

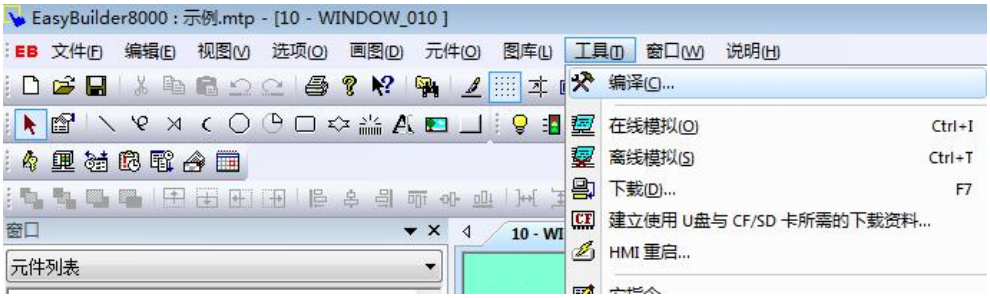


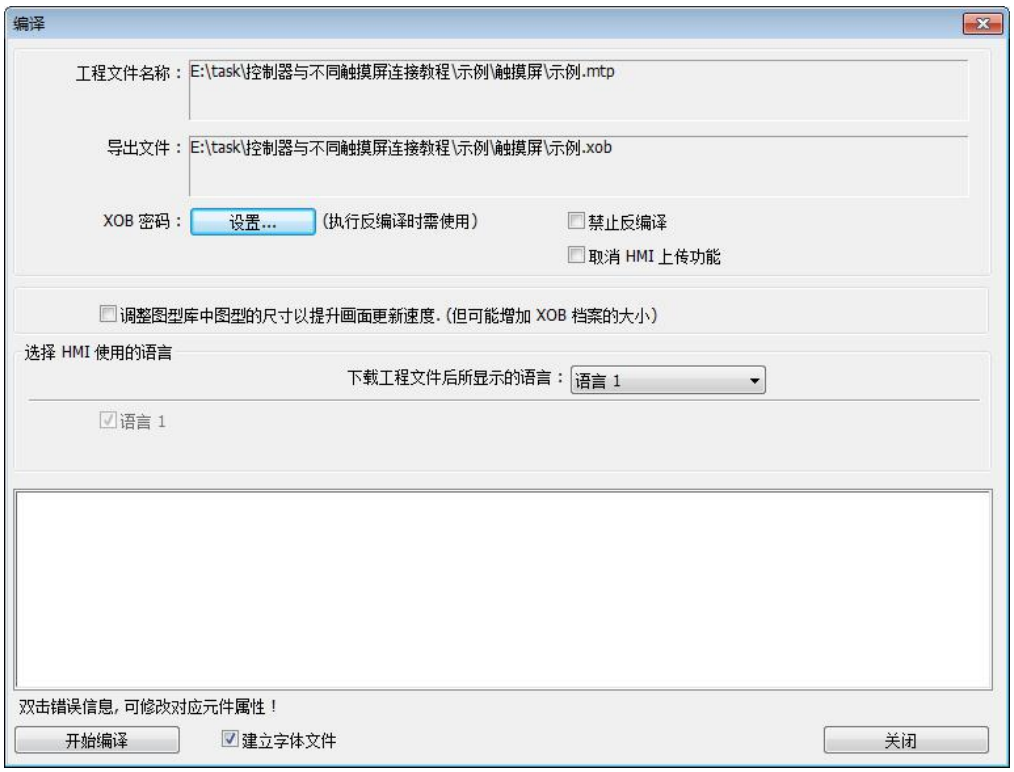
然后进行 ip 设置，点击“设置”。



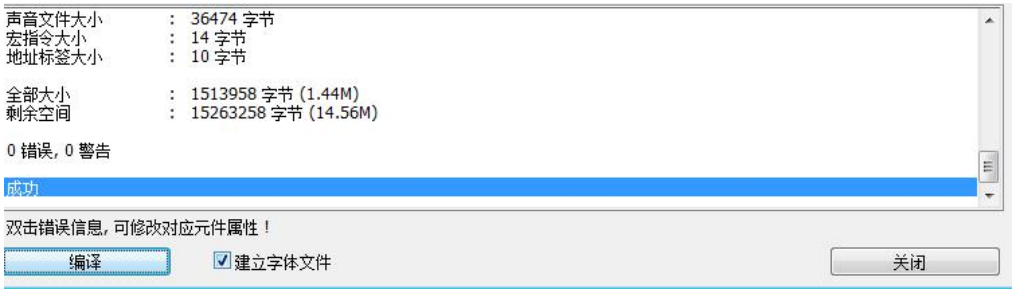
弹出 IP 地址设置窗口，此时将 IP 地址必须设为与控制器 IP 完全一致，控制器 IP 出厂默认为 192.168.0.11。

连接参数设置完成后，保存，点击“工具”-“编译”。

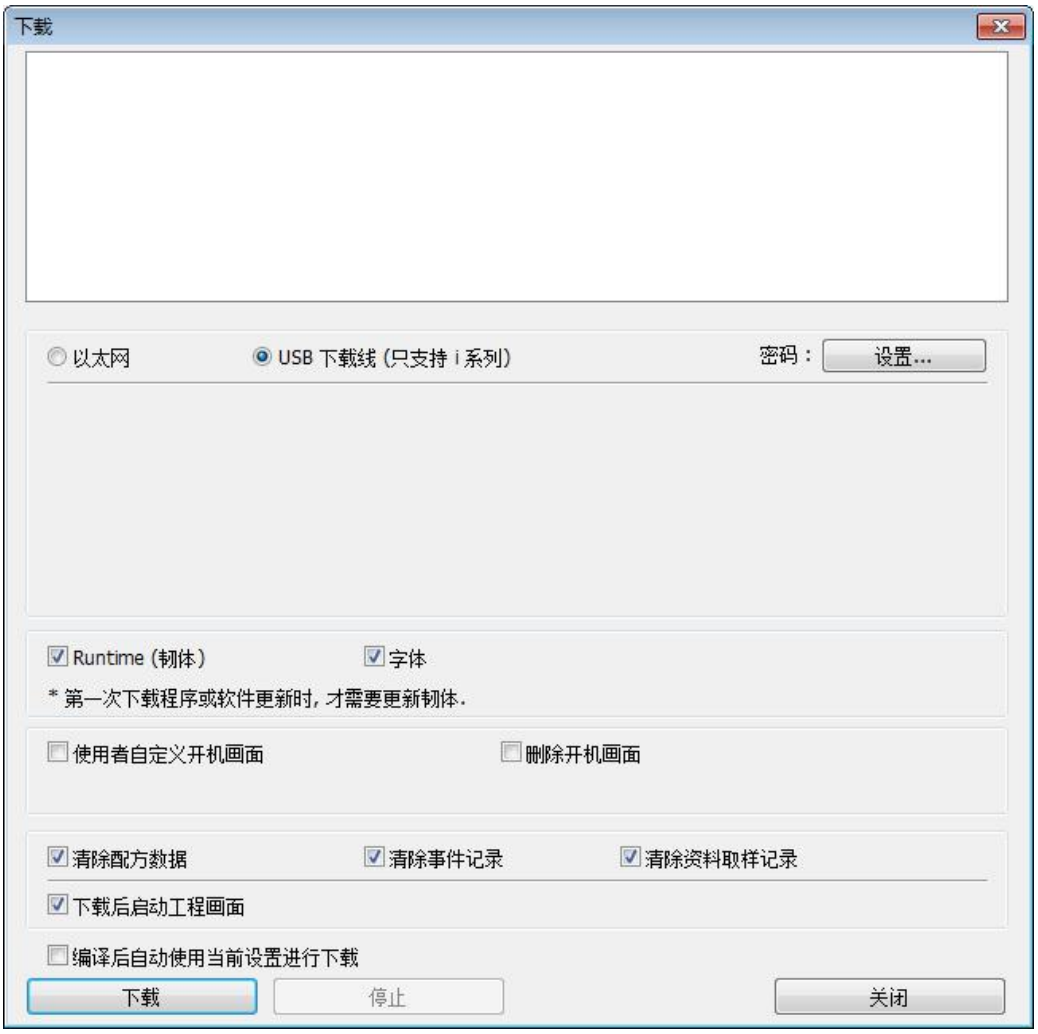




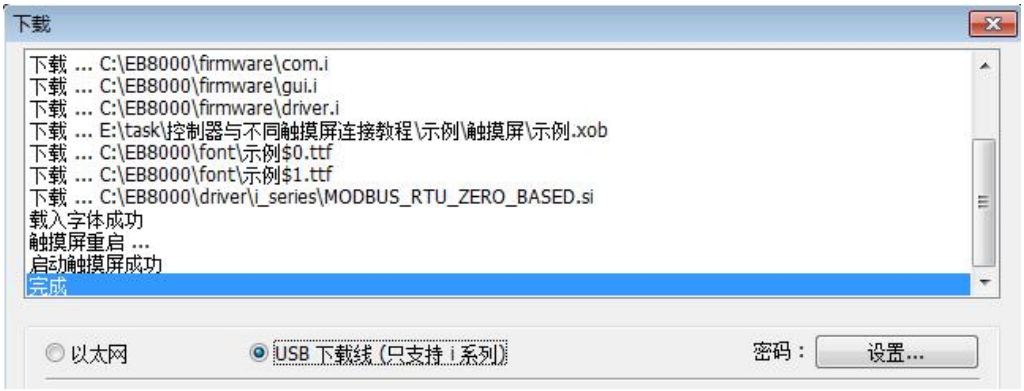
点击“开始编译”，编译完成后点击“关闭”退出编译界面。



再点击“工具”-“下载”，弹出下载界面。



威纶触摸屏程序有两种下载方式，一种是以太网，需要设置 IP 地址；一种是 USB 下载，只支持 i 系列。本例使用 USB 线连接触摸屏与 PC 端，所以采用 USB 下载。点击“下载”。



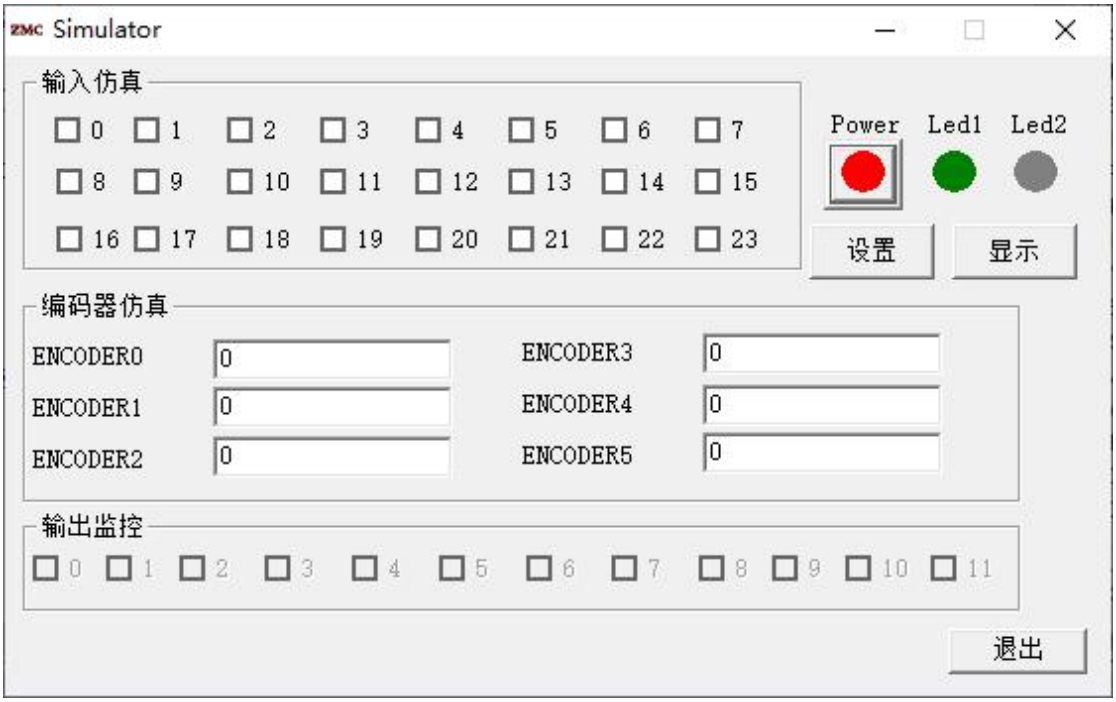
下载过程中触摸屏会重启，下载完成后，程序已写入触摸屏，可以断开触摸屏与电脑的连接。此时触摸屏与控制器就可以相互通信。

3.4.2 ZHmi 仿真效果

以正运动“光盘资料”-“控制器例程”-“入门程序”-“HMI 组态程序”-“单轴运动”例程为例，此

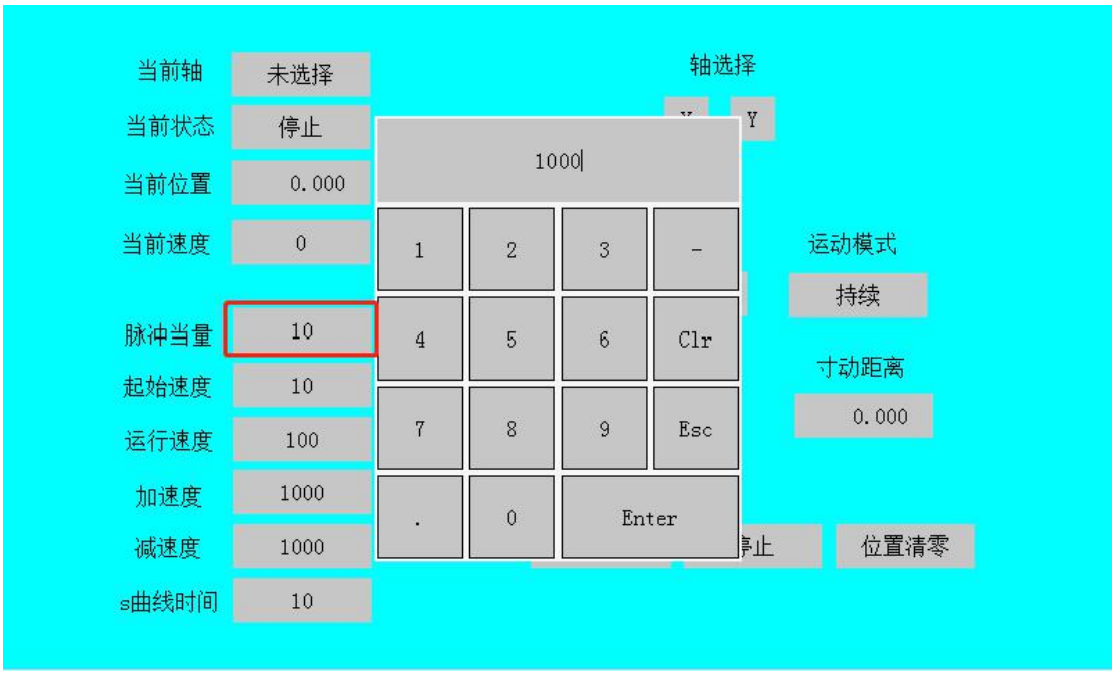
时不需要连接触摸屏和控制器即可在 ZDevelop 软件上仿真出程序效果。

打开 ZDevelop 软件，编辑好 Hmi 相关程序，连接到仿真器后下载程序到 ROM。



点击仿真器上的“显示”按钮，弹出“xplc screen”仿真界面。

点击界面上的灰色按钮，即可模拟触摸屏设置，下图以设置脉冲当量为例，自定义数值输入后确认，轴实际运动时将会按照此参数运动。



3.4.3 触摸屏脱机仿真

威纶触摸屏可以通过软件在 PC 上与 ZDevelop 进行脱机模拟，操作过程如下。

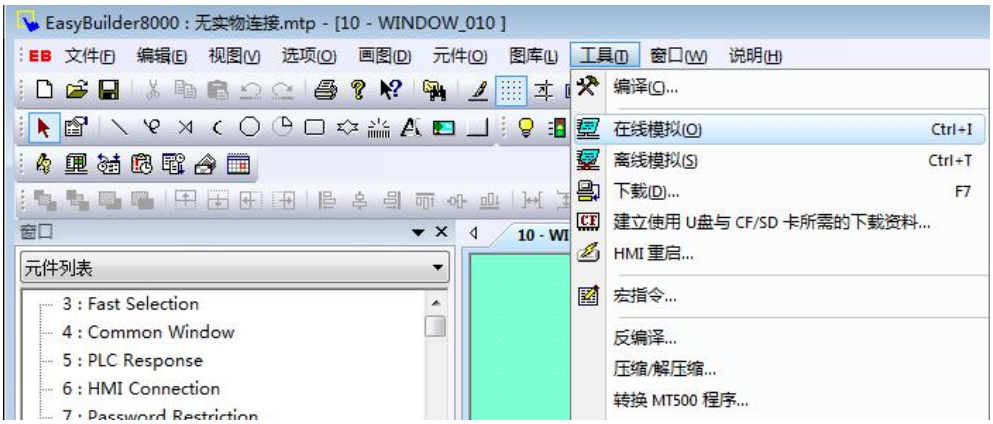
1．ZDevelop 操作

打开 ZDevelop 软件，连接仿真器，将程序下载到 ROM。

2．EB8000 操作

因为 ZSimulator 只能网口连接，所以 EB8000 脱机仿真时也只能通过网口与仿真器连接。按照“控制器与触摸屏连接”章节的网口连接方法进行操作，唯一不同的是 IP 地址为仿真器的 IP，在弹出 IP 地址设置窗口，将 IP 地址设为与 ZSimulator IP 地址同一网段，Zsimulator 的 IP 为 127.0.0.1，所以这里 IP 设为 127.0.0.xx。

设置完成后保存，点击“工具”-“在线模拟”，等待弹出程序界面，即可根据程序进行在线模拟操作，模拟结果参照例程效果。



3．注意事项

1）ZDevelop 打开的程序文件在连接到仿真器后，一定要下载。

2）EB8000 只能通过网口模式与 ZDevelop 进行在线模拟。

3）EB8000 的 IP 一定要与仿真器的 IP 处于同一网段。

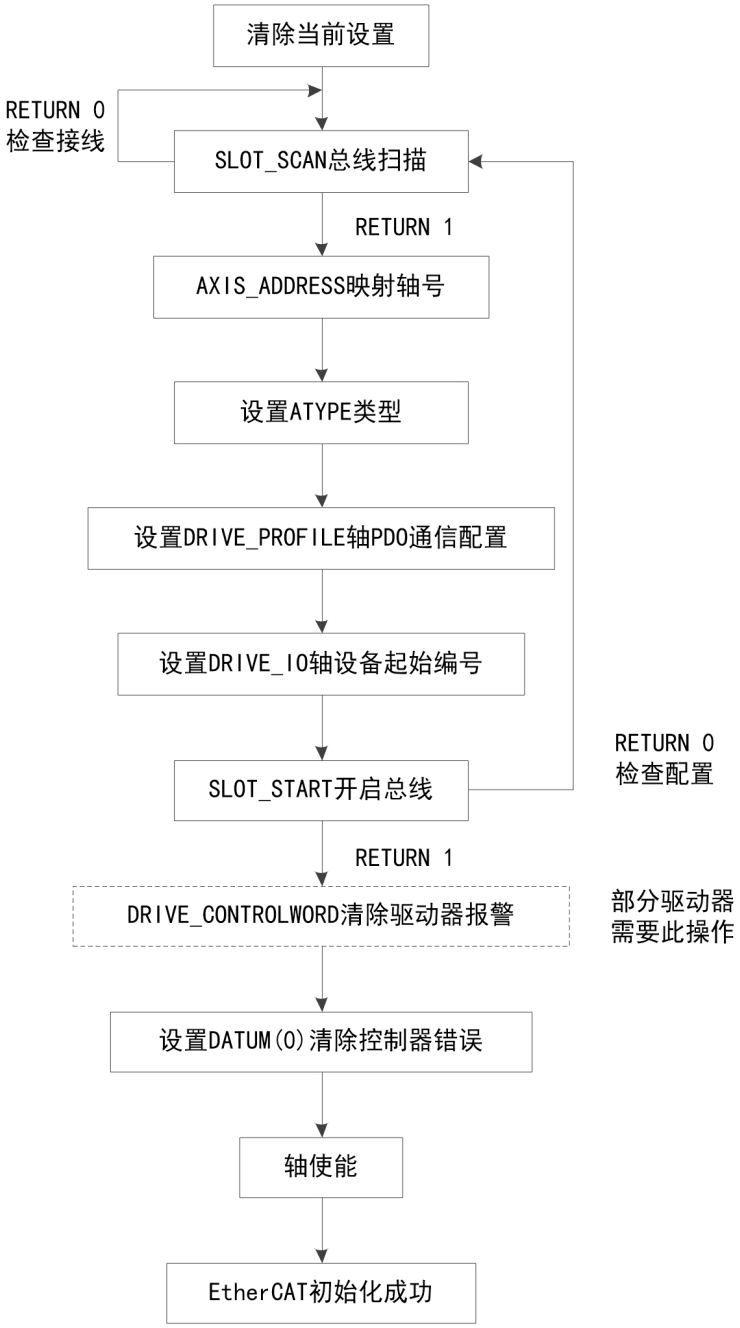
4）如果连接不上，请检查 EB8000 的系统参数设置是否正确，ZDevelop 的程序是否下载。

3.5 EtherCAT 总线初始化

控制器作为 EtherCAT 主站控制 EtherCAT 从站时，上电后使用软件程序对配置的从站进行初始化。初始化过程由控制器完成，不需要用户操作。

EtherCAT 初始化程序参见例程

初始化程序一般过程：



1. 使用 SLOT_SCAN 扫描设备，判断 RETURN 是否正确，未连接设备时不会报错。
2. 通过 NODE_INFO/ NODE_AXIS_COUNT 等对设备类型、信息等进行判断。

3．依次设置 AIXS_ADDRESS，ATYPE，DRIVE_PROFILE，DRIVE_IO 等。

4．SLOT_START 启动设备。

5．设置每个轴的 AXIS_ENABLE=1，打开轴的使能，设置 WDOG=1，所有轴使能允许。（部分驱动器需要使用 DRIVE_CONTROLWORD 指令清除驱动器报警）

6．建立连接后主站和从站即可进行周期性数据交换。

涉及基本概念：

1．NODE 节点编号：根据 EtherCAT 设备接线顺序排列，编号从 0 开始到设备个数减 1。

2．驱动器编号：根据 EtherCAT 设备接线顺序排列，编号从 0 开始到 EtherCAT 驱动器个数减 1，只在 AXIS_ADDRESS 配置时有作用，即只计算驱动器设备，其他扩展 IO 等设备不计入驱动器编号。

3．轴号：控制器对连接在控制器上的驱动电机设备设定的编号，编号从 0 开始到连接设备总个数减 1，通过 AXIS_ADDRESS 指令可以将轴号映射到任何一个连接的驱动器设备（脉冲型控制器不支持本地轴号映射）。

设置时注意事项：

1．设备号按照连接先后顺从 0 序依次递增，需要支持 EtherCAT 总线。

2．总线控制器上脉冲轴的轴号是固定的。总线也可以调用脉冲轴轴号，ATYPE=65 时总线调用，注意切换时 DPOS 会发生改变，所以最好不要使轴号冲突。

3．用 DRIVE_CONTROLWORD(轴号)清除驱动器报警后，需要延时 200ms，再用 DATUM(0)清除控制器报警，不延时可能需要下载两次程序才能清除报警。

4．如果使用了 DRIVE_CONTROLWORD(轴号)对驱动器操作，那么此驱动器之后的所有驱动器会继承此次设置。所以最好对每个驱动器都设置一次。

5．运行中连接或断开设备，需要重新扫描，状态才能更新，比如 NODE_COUNT(0)，返回当前连接设备个数，重新扫描后返回的个数才会改变。

6．AXIS_ADDRESS(i)=1，选择的是第一个驱动器，而不是第一个设备。比如连接了两个扩展板后再连接两个驱动器，设备号依次为扩展板 A：0，扩展板 B：1，驱动器 A：2，驱动器 B：3。此时 AXIS_ADDRESS(i)=1 选择的是驱动器 A，AXIS_ADDRESS(i)=2 选择的是驱动器 B。

按照顺序连续选择，不可先选 2 再选 1，不可选完 1 跳过 2 选择 3。

必须是控制器支持 EtherCAT 才可以使用相关的指令。与 RTEX 总线使用一套程序指令，但是功能有所区别，详细请参见第十六章总线指令说明。

EtherCAT 相关指令：

指令	含义
SLOT_SCAN	扫描设备
SLOT_START	总线启动
SLOT_STOP	总线停止
ATYPE	轴类型 ，例如 65 为 EtherCAT 周期位置控制
AXIS_ADDRESS	轴地址配置
WDOG	轴统一使能控制
SERVO_PERIOD	刷新率
AXIS_ENABLE	轴使能
SDO_WRITE	SDO 操作
SDO_READ	SDO 操作
SDO_WRITE_AXIS	驱动器 SDO 操作
SDO_READ_AXIS	驱动器 SDO 操作
?*ETHERCAT	总线信息输出

NODE_COUNT	设备个数
NODE_STATUS	设备状态
NODE_AXIS_COUNT	设备带驱动个数
NODE_IO	设备 IO 编号
NODE_AIO	设备 AIO 编号
NODE_INFO	设备信息读取
NODE_PROFILE	设备 PROFILE 设置，选择 PDO 报文内容
DRIVE_FE	驱动误差
DRIVE_FE_LIMIT	驱动误差设置
DRIVE_STATUS	驱动状态字
DRIVE_TORQUE	驱动器力矩反馈
DRIVE_MODE	驱动器运动模式
DRIVE_PROFILE	驱动器 PROFILE 设置，选择 PDO 报文内容
DRIVE_CONTROLWORD	驱动器控制字
DRIVE_CW_MODE	驱动器控制字操作模式
DRIVE_IO	远程 IO 编号设置
AXISSTATUS	轴状态

EtherCAT 配置与实际连接相符机制：

主站对从站进行初始化后，无论软件中配置从站数量与 EtherCAT 通讯口实际连接是否相符，配置成功的从站都可以通过主站控制。如软件中配置了两个 EtherCAT 从站，实际连接一个，则连接的这一个可以通过运动指令控制，主站和从站建立连接后，即使软件中配置的另一个从站再接入网络，主站也不会和后接入网络的从站建立连接。

EtherCAT 从站掉线与恢复机制：

EtherCAT 从站与主站建立连接后，如果因为通讯线缆拔出等外部原因导致部分 EtherCAT 从站通讯掉线，主站不会与掉线的从站重新建立连接，掉线的从站不能通过运动指令控制，没有掉线的 EtherCAT 从站不受影响。如果因为驱动器错误等内部原因导致的无法与总线通讯，查看该错误能否清除，清除后重新使能即可使用，不能清除需要断电重新执行总线初始化过程。

掉线的从站如果需要重新和主站建立连接，需要拔掉控制器与第一个伺服驱动器之间的 EtherCAT 线缆再插上，或者给控制器重新上电。如果进行上述操作，会影响正常运行的从站，正常运行的从站会和主站重新建立连接，如果有轴处于运行状态，会导致轴立即停止。

3.6 RTEX 总线初始化

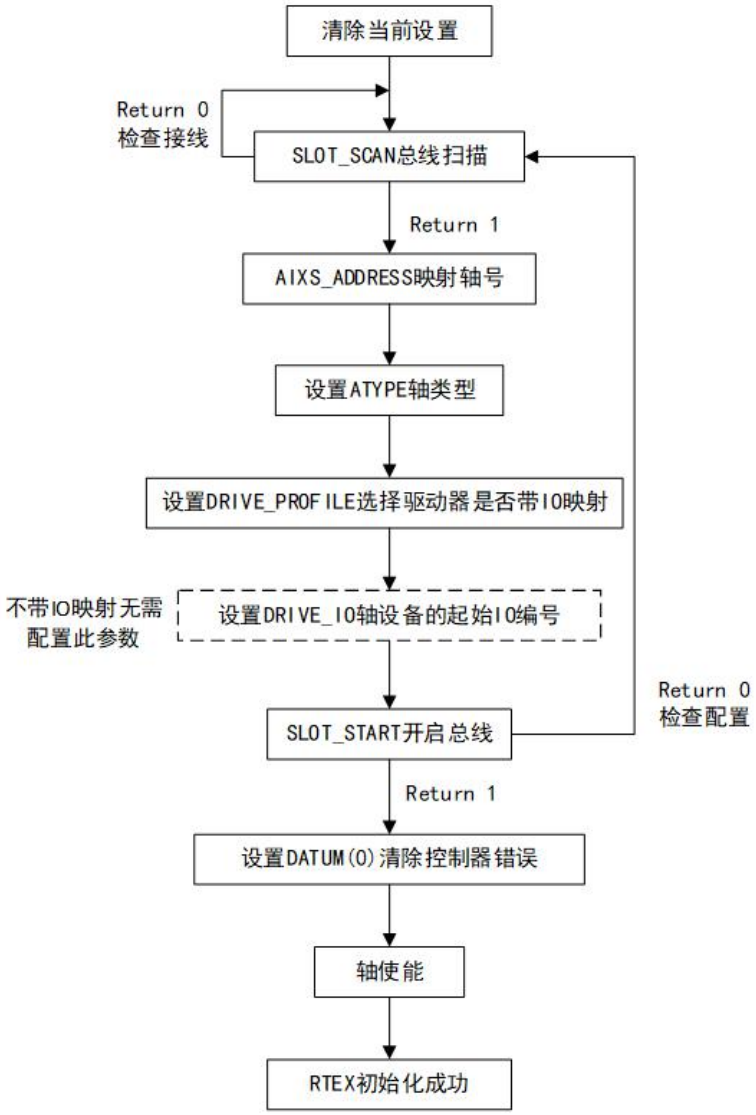
RTEX 是松下为实现运动控制高速实时性要求独自开发的高端总线技术，通过简化数据通讯包，实现高速通信，速度可达 100Mbps。仅支持连接松下驱动器。

控住器 RTEX 总线通讯有两个接口，分别为 RX 和 TX，接线时控制器 RX——驱动器 TX，控制器 TX——驱动器 RX。

RTEX 驱动器的控制字会自动设置，需要手动设置时请先设置 DRIVE_CW_MODE。

初始化过程由控制器完成，不需要用户操作，上电后程序开始执行。

初始化程序一般过程：



1．使用 SLOT_SCAN 扫描设备，判断 RETURN 是否正确，未连接设备时会报错。

2．通过 NODE_INFO/ NODE_AXIS_COUNT 等对设备类型、信息等进行判断。

3．依次设置 AIXS_ADDRESS，ATYPE，DRIVE_PROFILE，DRIVE_IO 等。

4．SLOT_START 启动设备。

5．建立连接后主站和从站即可进行周期性数据交换。

涉及基本概念：

1．NODE 节点编号：根据 RTEX 设备接线顺序排列，编号从 0 开始到设备个数减 1。

2．驱动器编号：根据 RTEX 设备接线顺序排列，编号从 0 开始到 RTEX 驱动器个数减 1，只在 AXIS_ADDRESS 配置时有作用，即只计算驱动器设备，其他扩展 IO 等设备不计入编号。

3．轴号：控制器对连接在控制上的驱动设备设定的编号，编号从 0 开始到连接设备总个数减 1，通过 AXIS_ADDRESS 指令可以将轴号映射到任何一个连接的驱动器设备（脉冲型控制器不支持本地轴号映射）。

必须是控制器支持 RTEX 才可以使用相关的指令。与 EtherCAT 总线使用一套程序指令，但是功能有所区别，详细请参见第十六章总线指令说明。

RTEX 相关指令：

指令	含义
SLOT_SCAN	扫描设备

SLOT_START	总线启动
SLOT_STOP	总线停止
ATYPE	轴类型 ，例如 50 为 RTEX 周期位置控制
AXIS_ADDRESS	轴地址配置
WDOG	轴统一使能控制
SERVO_PERIOD	刷新率
AXIS_ENABLE	轴使能
?*Rtex	总线信息输出
NODE_COUNT	设备个数
NODE_STATUS	设备状态
NODE_AXIS_COUNT	设备带驱动个数
NODE_IO	设备 IO 编号
NODE_AIO	设备 AIO 编号
NODE_INFO	设备信息读取
DRIVE_STATUS	驱动状态字
DRIVE_TORQUE	驱动器力矩反馈
DRIVE_PROFILE	驱动器 PROFILE 设置，选择 PDO 报文内容
DRIVE_CONTROLWORD	驱动器控制字
DRIVE_CW_MODE	驱动器控制字操作模式
DRIVE_IO	远程 IO 编号设置
DRIVE_CLEAR	驱动器清除报警，没有错误时使用控制器会警告
DRIVE_READ	驱动器读参数
DRIVE_WRITE	驱动器写参数
AXISSTATUS	轴状态

RTEX 从站掉线与恢复机制与 EtherCAT 相同，参见上节 EtherCAT 的说明。

第四章 运动控制系统

4.1 运动缓冲

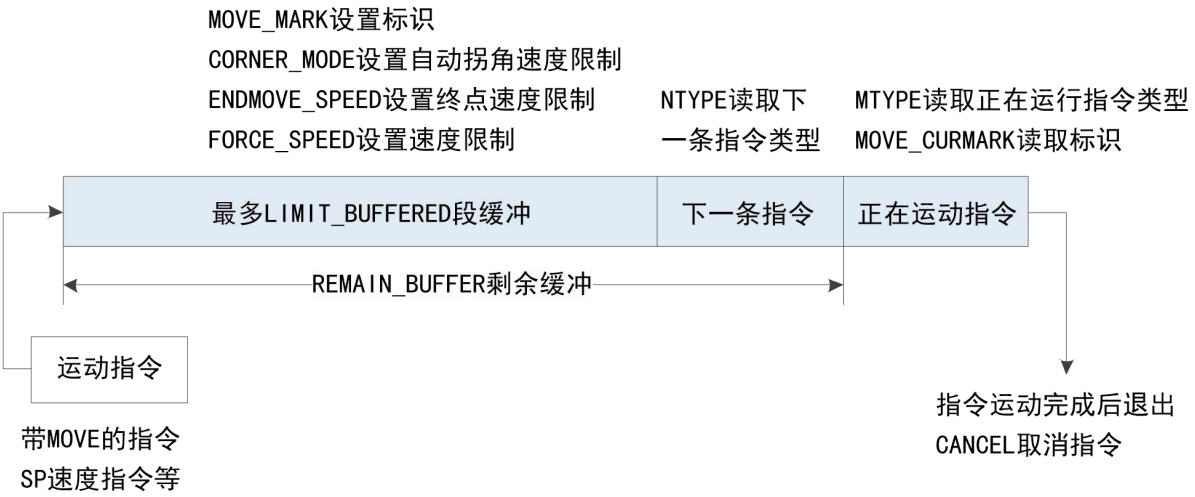
运动指令主要指直线、圆弧等各种插补运动指令以及带 MOVE 前缀的一些指令，用来实现点位运动、直线插补、圆弧插补、螺旋插补、凸轮运动等，在运动前要设置好轴参数。

运动指令可分为单轴运动指令和多轴运动指令，单轴运动指令一次只能操作一个轴的运动，轴号缺省状态下对主轴进行操作；多轴运动指令一次可以操作多个轴运动。

ZMotion 运动控制器具有多级的缓冲，当前有运动指令正在执行时，后面调用的运动指令会填入缓冲，避免程序的阻塞。当多级运动缓冲区全部被塞满时,如果程序继续运行更多的运动指令，程序也会被阻塞。直到指令依次完成并退出，缓冲区有了空位，指令才会继续进入缓冲。

SP 指令也属于运动指令，使用带 SP 指令时，FORCE_SPEED 、ENDMOVE_SPEED 和 STRATMOVE_SPEED 会随 SP 运动指令写入运动缓存区。

可以用 MOVE_OP 把 IO 操作指令填入缓冲,这样在运动指令之间自动加入 IO 输出。用 MOVE_DELAY 把延时指令填入缓冲，这样在运动指令之间自动延时。

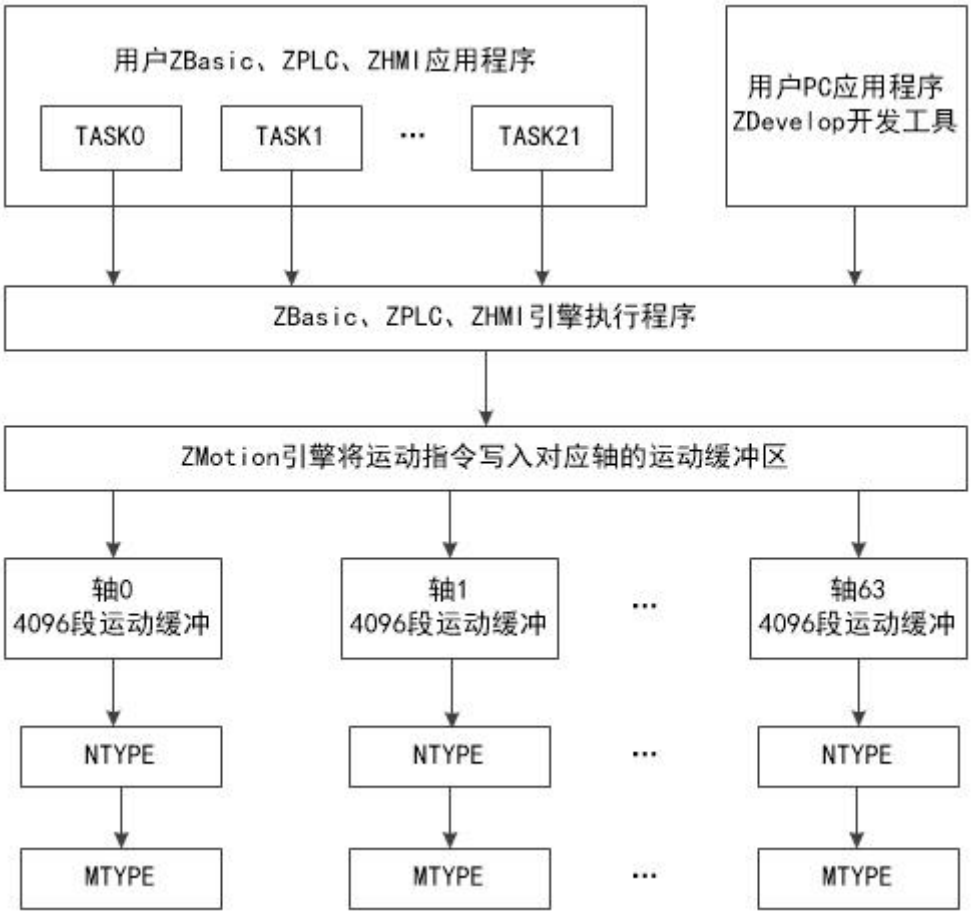


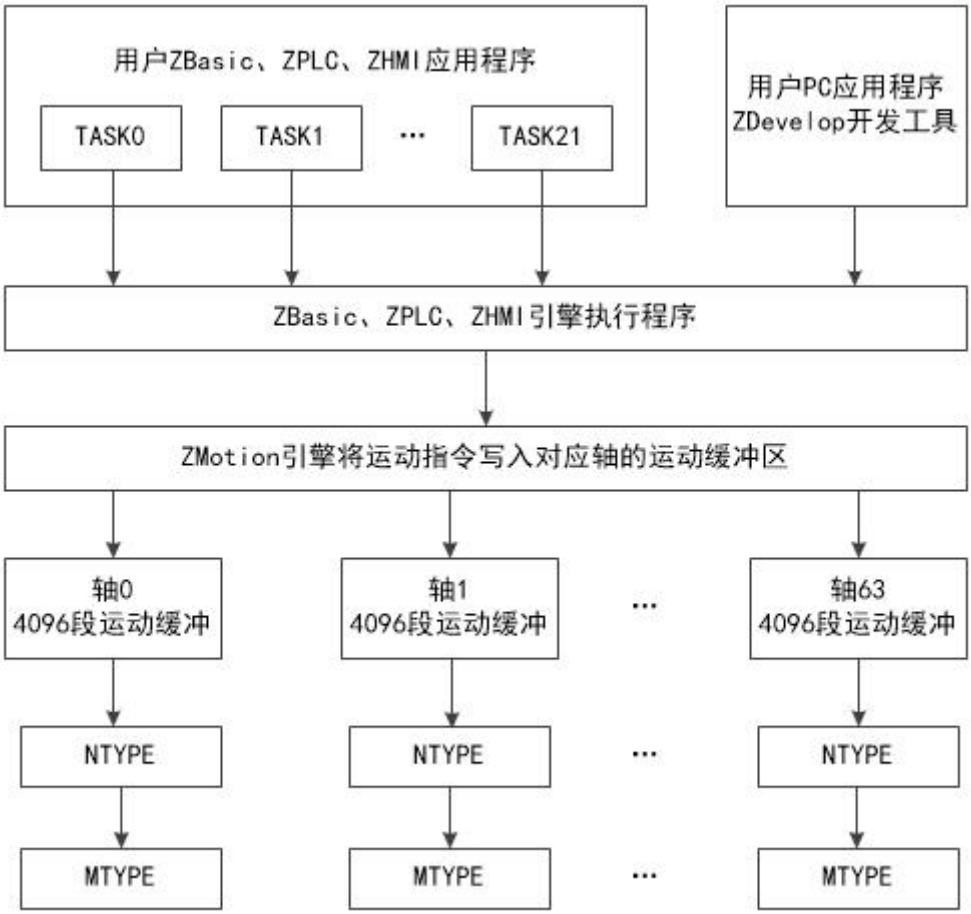
MTYPE：当前正在运行的指令类型。

NTYPE：当前正在进行的运动指令后面的第一条指令类型。

每个轴的运动缓冲都是独立的，互不干扰，如下图。ZMC4 系列运动控制器每个轴可支持多达 4096 段运动缓冲（不同型号的控制器缓冲个数有区别，具体情况参见控制器硬件手册说明）。

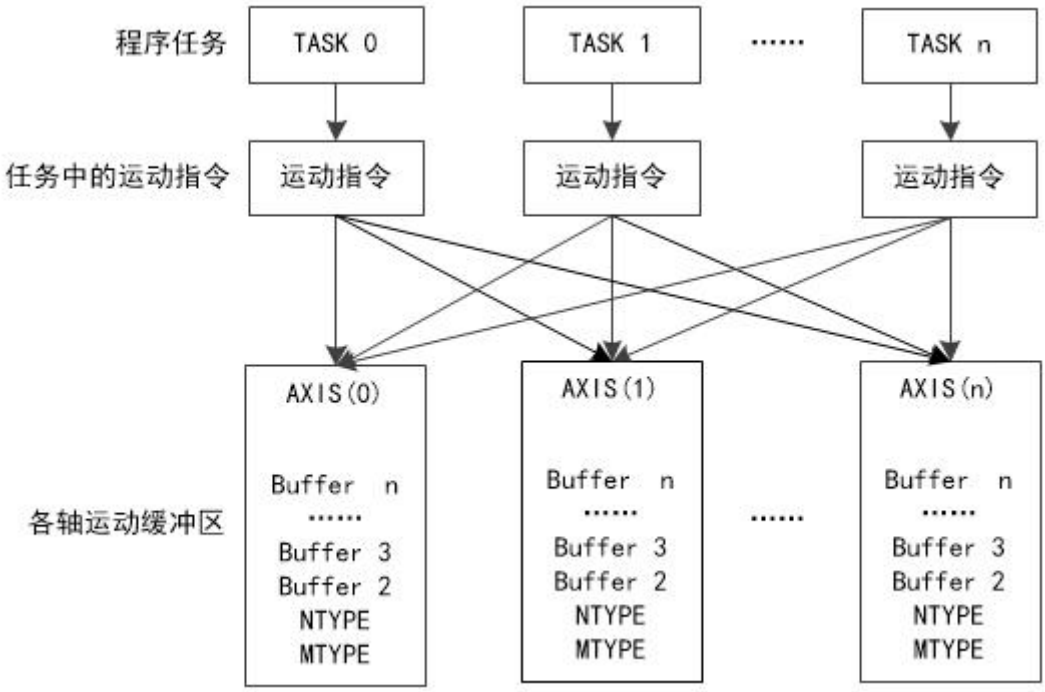
正运动控制器运动缓冲执行机制如下图：


用户ZBasic、ZPLC、ZHMI应用程序
TASK0
TASK1
…
TASK21
用户PC应用程序
ZDevelop开发工具
ZBasic、ZPLC、ZHMI引擎执行程序
ZMotion引擎将运动指令写入对应轴的运动缓冲区
轴0
4096段运动缓冲
轴1
4096段运动缓冲
…
轴63
4096段运动缓冲
NTYPE
NTYPE
…
NTYPE
MTYPE
MTYPE
…
MTYPE



任务与轴的关系：

根据指令要求，任务会自动分配运动指令到需要运动的轴的运动缓冲区。单轴运动时，所有任务的运动指令都进入目标轴的运动缓冲区。多轴插补运动时，每个任务都可以有独立的主轴，默认 BASE 指令的第一个轴为主轴，任务里的运动指令会分配到插补运动主轴的运动缓冲区，插补运动采用主轴的运动参数。



MOVEMODIFY 和 MOVEMODIFY2 属于特例，这两个指令不会进入运动缓冲区。

4.2 插补运动

4.2.1 插补概念

插补是机床数控系统依照一定方法确定刀具运动轨迹的过程，插补是一个实时进行的数据密化的过程，不论是何种插补算法，运算原理基本相同，其作用都是根据给定的信息进行数字计算，不断计算出参与运动的各坐标轴的进给指令，然后分别驱动各自相应的执行部件产生协调运动，以使被控机械部件按理想的路线与速度移动。

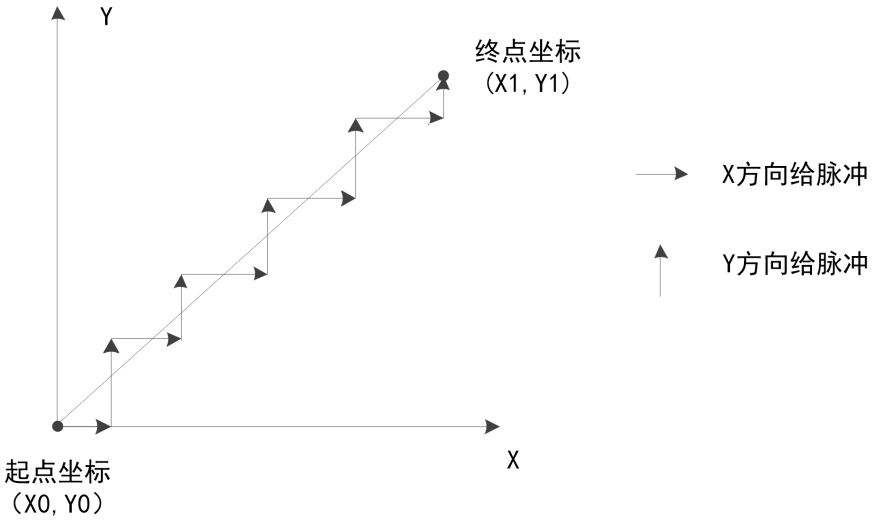
插补最常见的两种方式是直线插补和圆弧插补。插补运动至少需要两个轴参与，进行插补运动时，首先需要建立坐标系，将规划轴映射到相应的坐标系中，运动控制器根据坐标映射关系，控制各轴运动，实现要求的运动轨迹。

1．插补原理

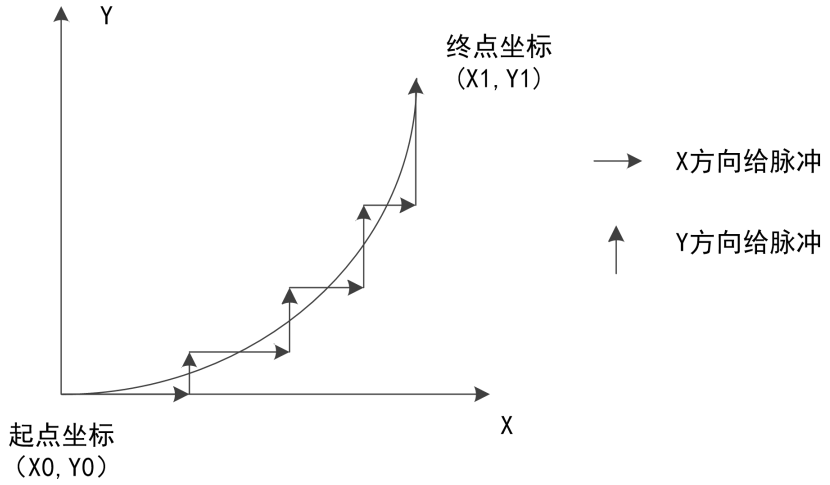
直线插补方式中，两点间的插补沿着直线的点群来逼近。首先假设在实际轮廓起始点处沿 X 方向走一小段（给一个脉冲当量轴走一段固定距离），发现终点在实际轮廓的下方，则下一条线段沿 Y 方向走一小段，此时如果线段终点还在实际轮廓下方，则继续沿 Y 方向走一小段，直到在实际轮廓上方以后，再向 X 方向走一小段，依次循环类推，直到到达轮廓终点为止。这样实际轮廓是由一段段的折线拼接而成，虽然是折线，但每一段插补线段在精度允许范围内非常小，那么此段折线还是可以近似看做一条直线段，这就是直线插补。

正运动控制器采用硬件插补，插补精度在一个脉冲内，所以轨迹放大依然平滑。

假设轴需要在在 XY 平面上从点(X0,Y0)运动到点(X1,Y1)，其直线插补的加工过程如下图所示。



圆弧插补与直线插补类似，给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制轴沿这些点运动，加工出圆弧曲线。圆弧插补可以是平面圆弧（至少两个轴），还可以是空间圆弧（至少三个轴）。假设轴需要在 XY 平面第一象限走一段逆圆弧，圆心为起点，其圆弧插补的加工过程如下图所示。



控制器的空间圆弧插补功能是根据当前点和圆弧指令参数设置的终点和中间点（或圆心），由三个点确定圆弧，并实现空间圆弧插补运动，坐标为三维坐标，至少需要三个轴分别沿 X 轴、Y 轴和 Z 轴运动。

2．运动控制器的插补模式

运动控制器的插补运动模式具有以下功能：

1) 可以实现直线插补、圆弧插补、空间圆弧插补、椭圆插补、螺旋插补等；

2) 可以在多个坐标系多通道进行多轴插补运动；

3) 每轴均有运动缓存区，可以实现运动的暂停、恢复等功能，停止插补运动的一个轴，其他轴跟着全部停止；

4) 具有缓存区延时和缓存区数字量同步输出的功能；

5) 具有预处理功能，控制器自行分析计算目标轨迹，能够实现小线段高速平滑的连续轨迹运动。

二轴直线插补

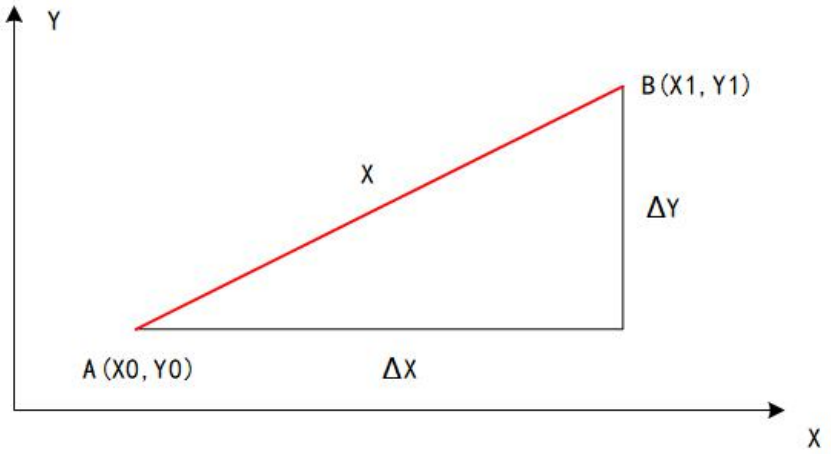
轴 0 和轴 1 两轴参与直线插补运动，如下图，2 轴直线插补运动从 A 点运动到 B 点，XY 轴同时启动，并同时到达终点，设置轴 0 的运动距离为ΔX，轴 1 的运动距离为ΔY，主轴是 BASE 的第一个轴（此时主轴为轴 0），插补主轴运动速度为 S（主轴的设置速度），各个轴的实际速度为主轴的分速度，不等于 S，此

时：

主轴运动距离：$X=[(\Delta X)^2+(\Delta Y)^2]^{1/2}$

轴 0 实际速度：$S0=S\times\Delta X/X$

轴 1 实际速度：$S1=S\times\Delta Y/X$



三轴直线插补

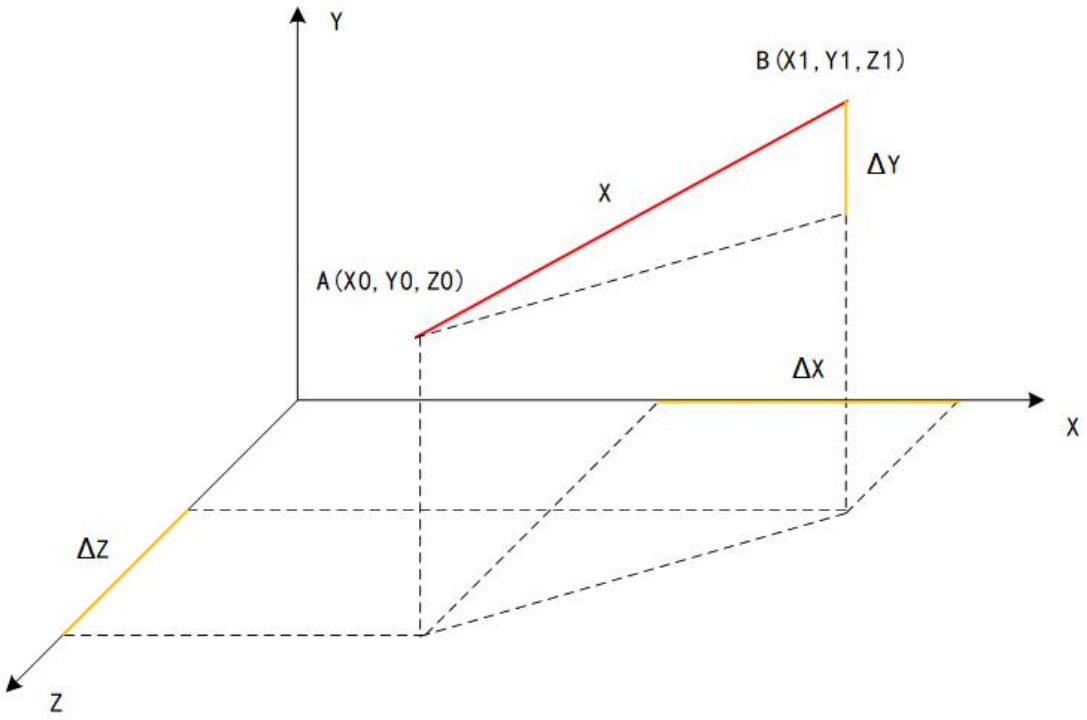
轴 0、轴 1 和轴 2 三轴参与直线插补运动，如下图，3 轴直线插补运动从 A 点运动到 B 点，XYZ 轴同时启动，并同时到达终点，设置轴 0 的运动距离为ΔX，轴 1 的运动距离为ΔY，轴 2 的运动距离为ΔZ，插补主轴轴 0 的运动速度为 S，各个轴的实际速度为主轴的分速度，不等于 S，此时：

主轴运动距离为 $X=[(\Delta X)^2+(\Delta Y)^2+(\Delta Z)^2]^{1/2}$

轴 0 实际速度：$S0=S\times\Delta X/X$

轴 1 实际速度：$S1=S\times\Delta Y/X$

轴 2 实际速度：$S1=S\times\Delta Z/X$



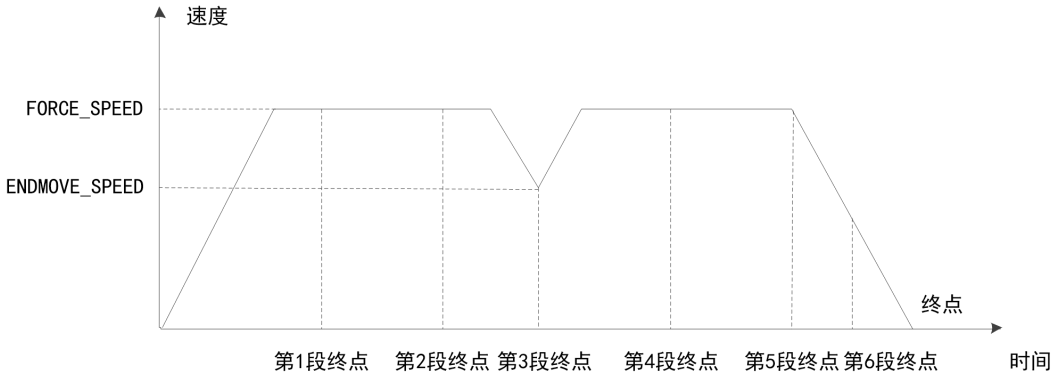
多轴直线插补

多轴直线插补可以理解为轴的多个自由度，是在三维空间里的直线插补。以四轴插补为例，一般是三个轴在 XYZ 平面走直线，另一个轴为旋转轴，按照一定的比例关系做跟随运动。

4.2.2 连续插补

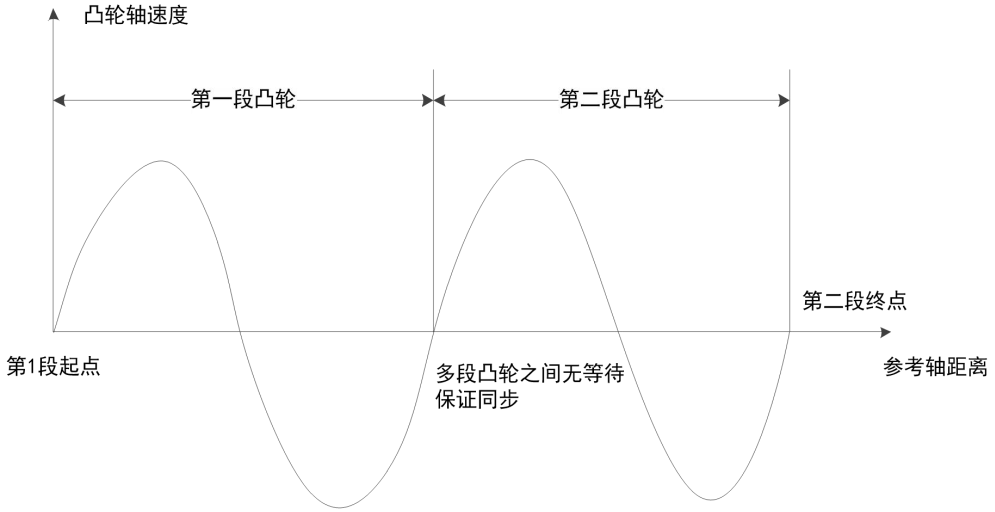
不开启 MERGE 连续插补，上一条插补运动完成后，执行下一条插补时，会先减速停止，再重新加速执行插补运动，实际应用时这种情况会导致加工效率低下，所以需要使连续的插补运动之间不减速，这就是连续插补功能。

若要使插补动作连续，设置 MERGE=ON 以后，相同主轴的插补运动会自动被连续起来，连续两段运动之间不减速，而且 SP 指令可以手动设置运动速度和结束速度，参见：MERGE，SP，CORNER_MODE，ENDMOVE_SPEED，FORCE_SPEED 等指令说明。

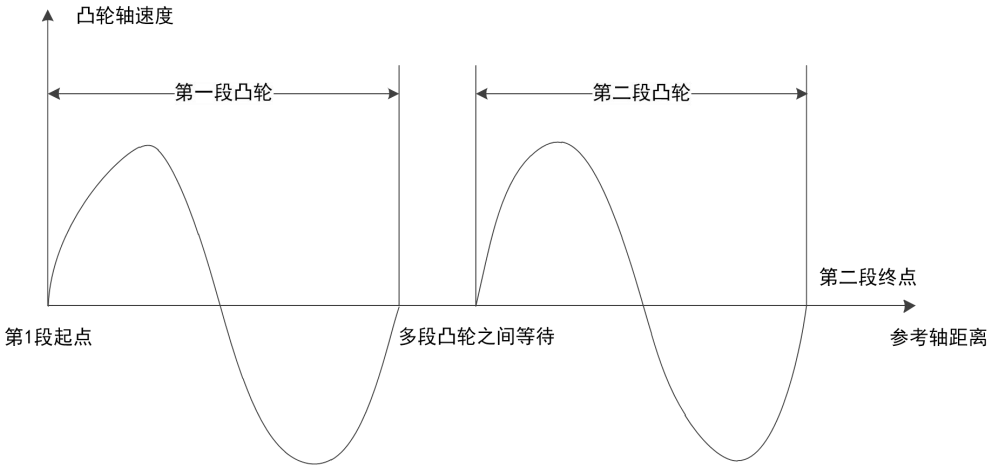


4.2.3 连续凸轮

多段凸轮指令被存入运动缓冲区后，下一段凸轮会在前一段凸轮完成后立刻开始，保证同步。



若凸轮指令没有被存入运动缓冲区，多段凸轮间会有等待时间，运动不连续。参见：CAM，CAMBOX。



4.3 前瞻预处理

在实际工业加工中，只有正确的预测起始速度和末端速度，才能保证运动过程中速度平滑过渡，否则当轨迹出现急拐角时，如果没有足够的时间和距离降低运动速度，会使拐角处的向心加速度超过伺服控制能力，产生很大的轮廓误差，会对机床和轴造成一定损伤。因此，在运动控制过程中，必须要超前处理若干程序段，识别出急拐角，及时发出减速命令，控制轴的速度，这种功能被称为前瞻预处理。

轨迹前瞻控制或速度前瞻控制时数控系统的一项关键技术，其工作原理是控制系统在控制加工时扫描运动缓冲区的指令，分析这些指令的轨迹变化后动态调节拐角减速度，实现多段插补运动高速平稳衔接，减少机床的冲击，从而提高加工精度。

前瞻运动相关指令：

指令	说明	用法
CONNER_MODE	拐角模式设置	CORNER_MODE=模式值
MERGE	连续插补	MERGE= ON
DECEL_ANGLE	开始减速的角度	使用时角度单位换算成弧度
STOP_ANGLE	停止减速的角度	使用时角度单位换算成弧度
ZSMOOTH	倒角半径	ZSMOOTH=倒角半径值
FULL_SP_RADIUS	限速半径	小圆限速最大圆弧半径
FORCE_SPEED	强制速度	等比减速以该参数为参考值

CORNER_MODE 参数说明：

位	值	描述
0	1	预留
1	2	自动拐角减速。 按 ACCEL，DECEL 加减速度。 此参数是在 MOVE 函数调用前生效。 减速角度根据 DECEL_ANGLE 和 STOP_ANGLE 指令设定。 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。
2	4	预留
3	8	自动小圆限速，半径小于设置值时限速，大于限制值时不限速。 此参数在 MOVE 函数调用前修改生效。 限制速度与 FORCE_SPEED 有关。

		限制速度= FORCE_SPEED * 实际半径/FULL_SP_RADIUS 限速半径 FULL_SP_RADIUS 设置。
4	16	预留
5	32	自动倒角设置。 此参数在 MOVE 函数调用前修改生效。 此 MOVE 运动自动和前面的 MOVE 运动做倒角处理，倒角半径参考 ZSMOOTH。

用户调用如 MOVE 等直线插补指令和圆弧插补指令将运动数据存入运动缓存区。前瞻运动参数也一并存入运动缓冲区，前瞻运动参数可以多次调用，不同模式也可以混合使用，例如 CONNER_MODE=2+8，表示同时使用自动拐角减速和小圆限速，缓冲区数据依次执行，根据轨迹段的要求设置合适的拐角模式。

使用方法说明：分析进行插补的轨迹，若轨迹为直线插补考虑使用自动拐角或自动倒角，分析轨迹情况选择；若轨迹为圆弧可考虑使用自动小圆限速（小圆限速仅限圆弧使用）和自动倒角；若轨迹为多种曲线混合，几种模式可以同时使用。

拐角减速能控制减速角度范围的运动速度，运动曲线不发生改变，实现轨迹精准控制；倒角减速适用于轨迹拐角较大的场合，倒角处运动轨迹做自动平滑处理，由于倒角处运动轨迹发生了变化，对轨迹精度要求很高的场合不宜选用自动倒角模式。

相同轨迹使用自动拐角减速和自动倒角的区别参见如下两个例子。例程中倒角整体速度快，运行时间相对较短。

1．自动拐角减速：mode=2

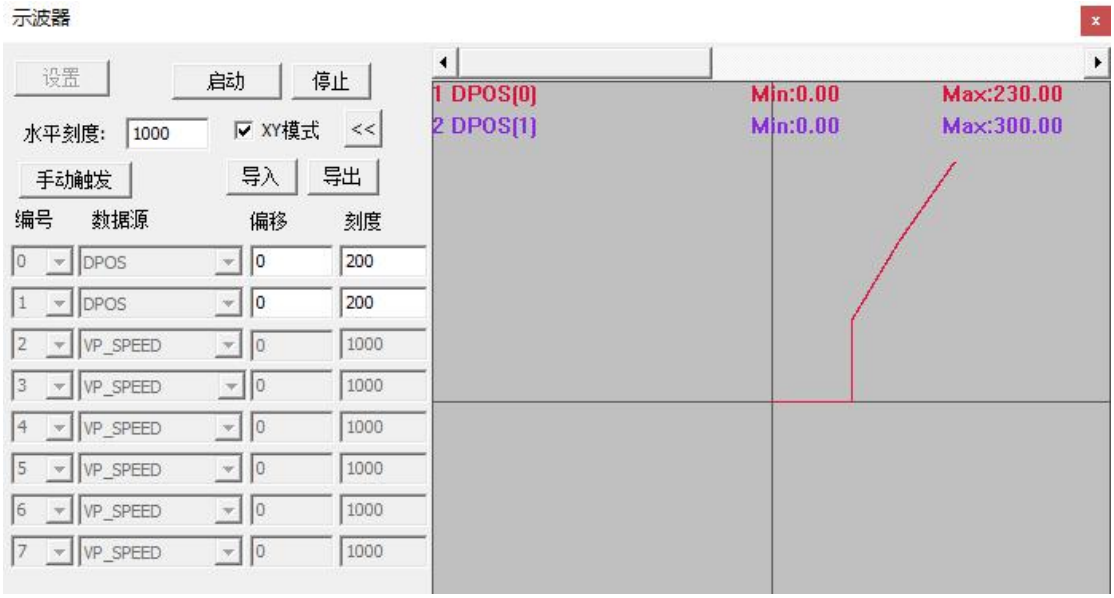
例程：配合减速角度使用

```
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0,1)
DPOS=0,0
ATYPE=1,1
UNITS=100,100
ACCEL=500,500          '设置加速度
DECEL=500,500          '设置减速度
SPEED=100,100          '设置速度
MERGE=ON               '开启连续插补
CORNER_MODE=2          '启动拐角减速
DECEL_ANGLE=15*(PI/180)    '设置开始减速角度
STOP_ANGLE=45*(PI/180)     '设置结束减速角度
FORCE_SPEED=100        '等比减速时起作用
TRIGGER                '自动触发示波器
MOVE(100,0)
MOVE(0,100)        '运动角度大于 45°，完全减速
MOVE(60,100)       '运动角度 30.96°处于 15°~45°，等比减速
MOVE(70,100)       '运动角度 4.03°小于 15°，不减速
```

运动轨迹如下：

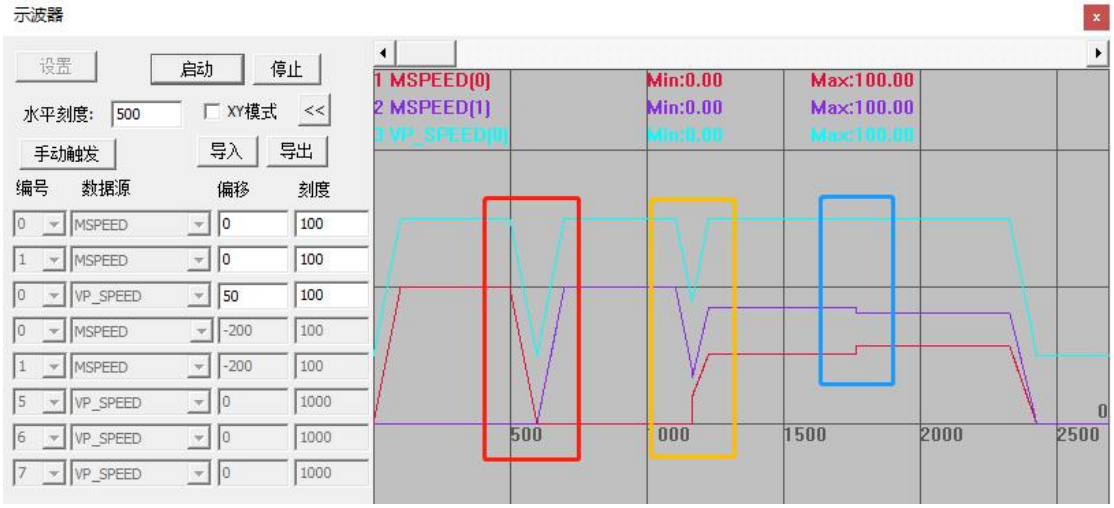
示波器设置旋转参数：

DPOS(0)与 DPOS(1)组成 XY 模式的曲线如图所示：



速度曲线：

VP_SPEED 表示轴组的合成速度，运动轨迹未变，CORNER_MODE=2 后通过自动速度前瞻调节速度来达到保护机床的目的。第二段相对第一段运动角度大于 45°，参与插补的两个轴均减速到 0 后再加速运动，如红色框图所示；第三段相对第二段运动角度在减速区域内，拐角时等比减速，如黄色框图所示；第四段相对第三段运动角度小于 15°，无需减速，此时两轴的合成速度为 100，如蓝色框图所示。

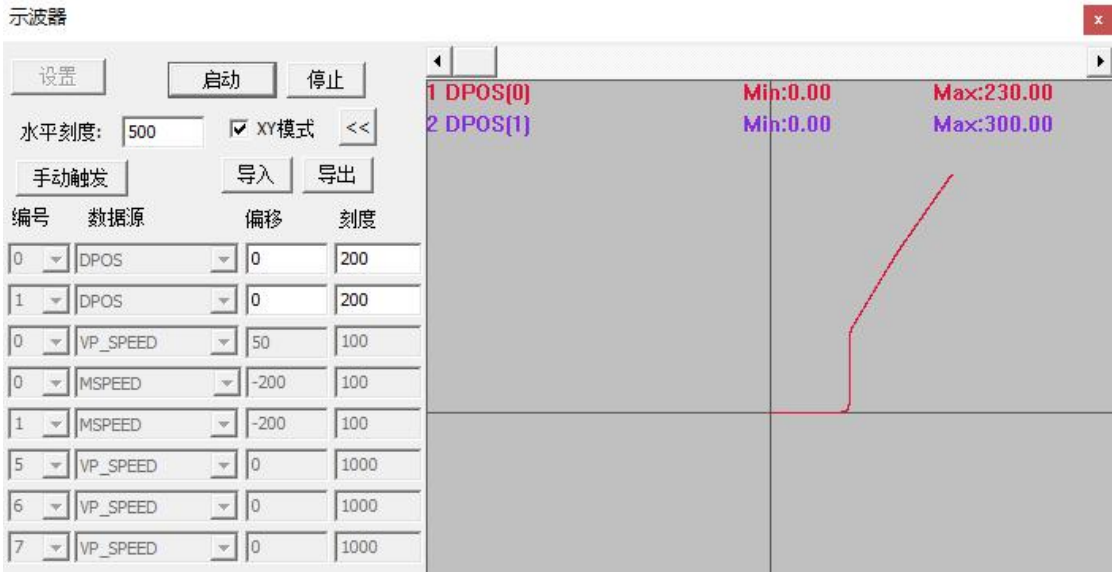


MSPEED(0)为轴 0 的分速度
MSPEED(1)为轴 1 的分速度
VP_SPEED(0)为轴组合成矢量速度

2．自动倒角减速：mode=32
例程：配合 ZSMOOTH 倒角半径使用

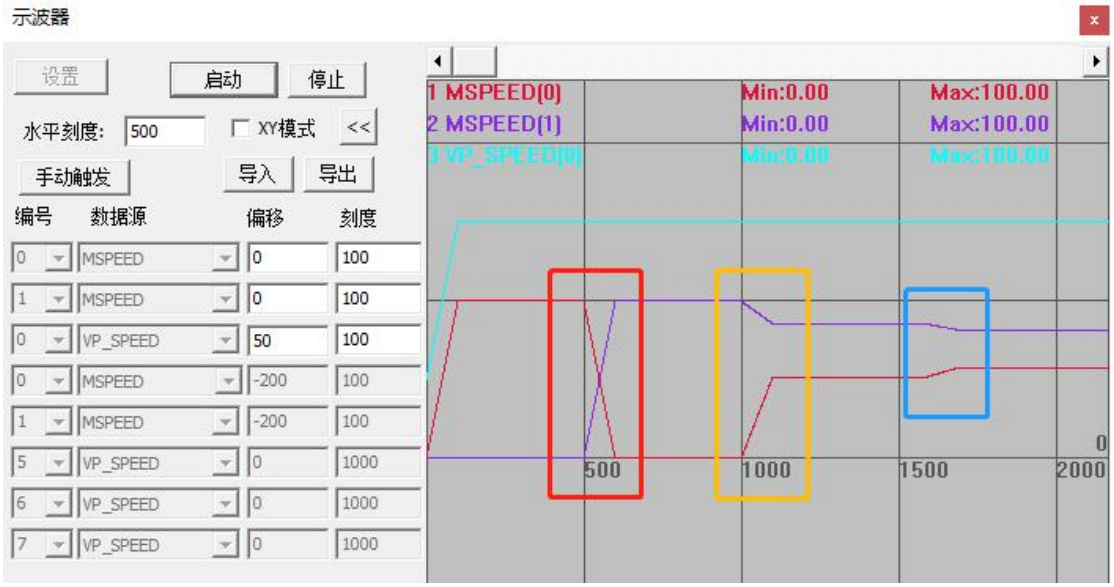
```
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0,1)
DPOS=0,0
UNITS=100,100
SPEED=100,100              '设置速度
ACCEL=500,500              '设置加速度
DECEL=500,500              '设置减速度
MERGE=ON                   '开启连续插补
CORNER_MODE=32             '启动倒角减速
ZSMOOTH=10                 '设置倒角半径
TRIGGER                    '自动触发示波器
MOVE(100,0)
MOVE(0,100)                '开启倒角，限速
MOVE(60,100)               '限速
MOVE(70,100)               '限速
```

XY 模式下 CORNER_MODE=32 运动轨迹如下：



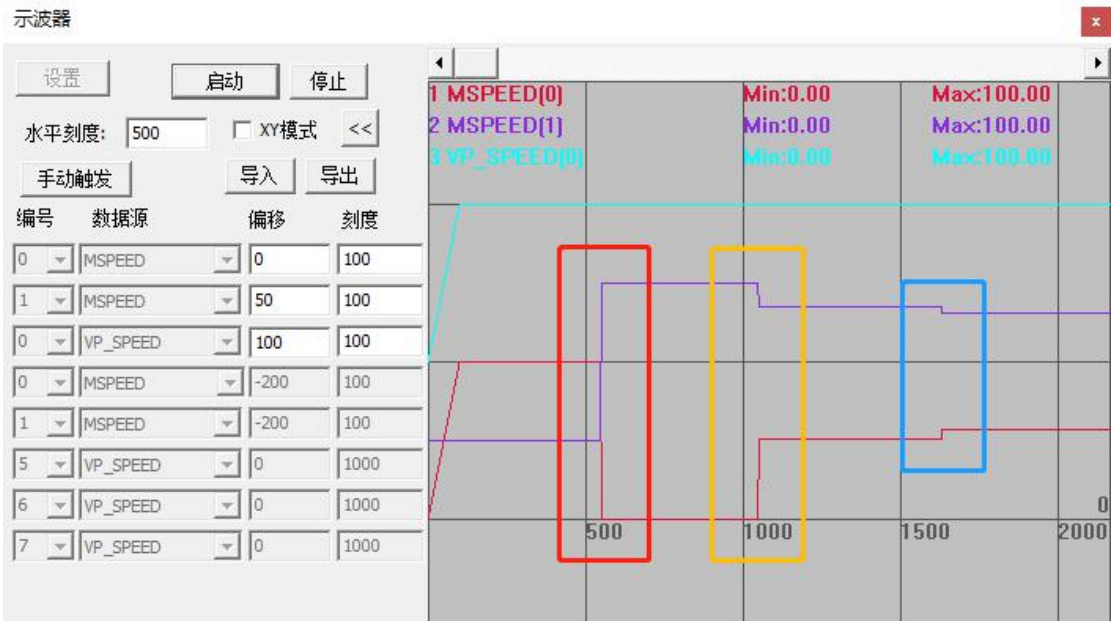
速度曲线如下：

主轴速度 VP_SPEED 在运动拐角处并未减速，仅对拐角处运动轨迹做了平滑处理，运动轨迹发生了微小变化，但是插补运动效率自动倒角要高于自动拐角。



不设置 CORNER_MODE(CORNER_MODE=0)时，因为开启了连续插补，合成主轴不减速，如下图可以看出拐弯过程中，其他轴为了自适应主轴的速度，加减速非常快，会对机床产生较大冲击。

如下图红色框图，黄色框图，蓝色框图三个部分的速度冲击。



XY 模式下 CORNER_MODE=0 运动轨迹如下：

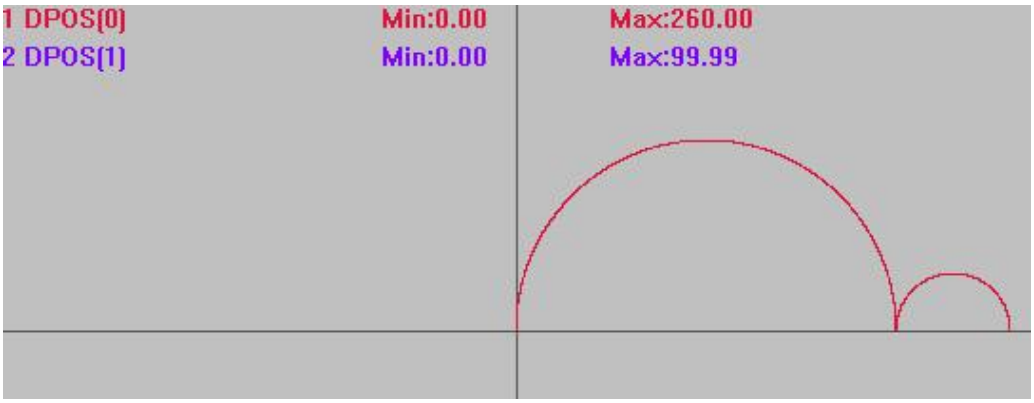
3．小圆限速 mode=8

例程：设置限速半径 FULL_SP_RADIUS 和限速速度 FORCE_SPEED

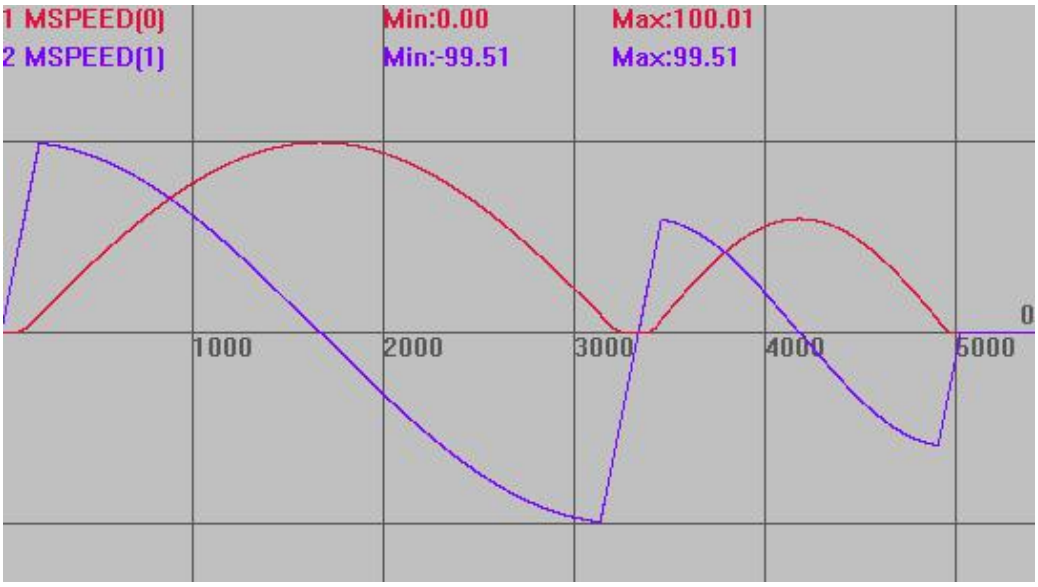
```
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0,1)
DPOS=0,0
UNITS=100,100
SPEED=100,100              '运行速度
ACCEL=500,500              '设置加速度
DECEL=500,500              '设置减速度
CORNER_MODE=8              '启动小圆限速
FORCE_SPEED=120            '小圆限速
FULL_SP_RADIUS=60          '限速半径 60
TRIGGER                    '自动触发示波器
MOVECIRC(200,0,100,0,1)    '半径大于限制值，不限制速度，按 SPEED 运行
```

MOVECIRC(60,0,30,0,1)　　'半径小于限制值，此时速度 60,120*30/60=60

运动轨迹：



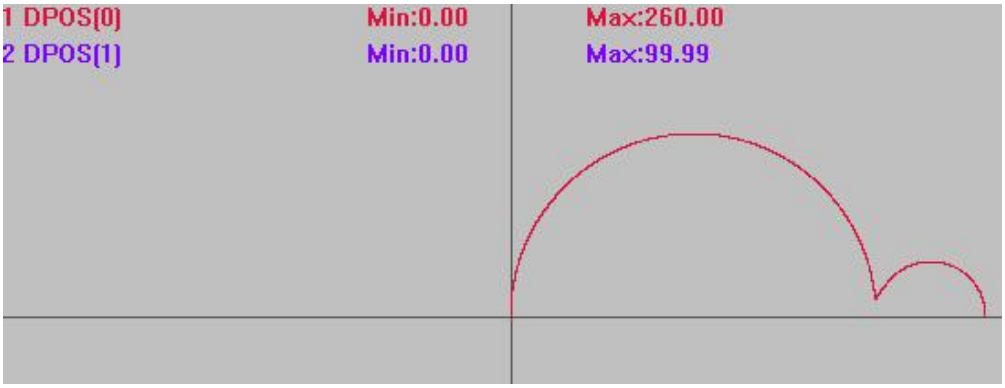
速度曲线：限速



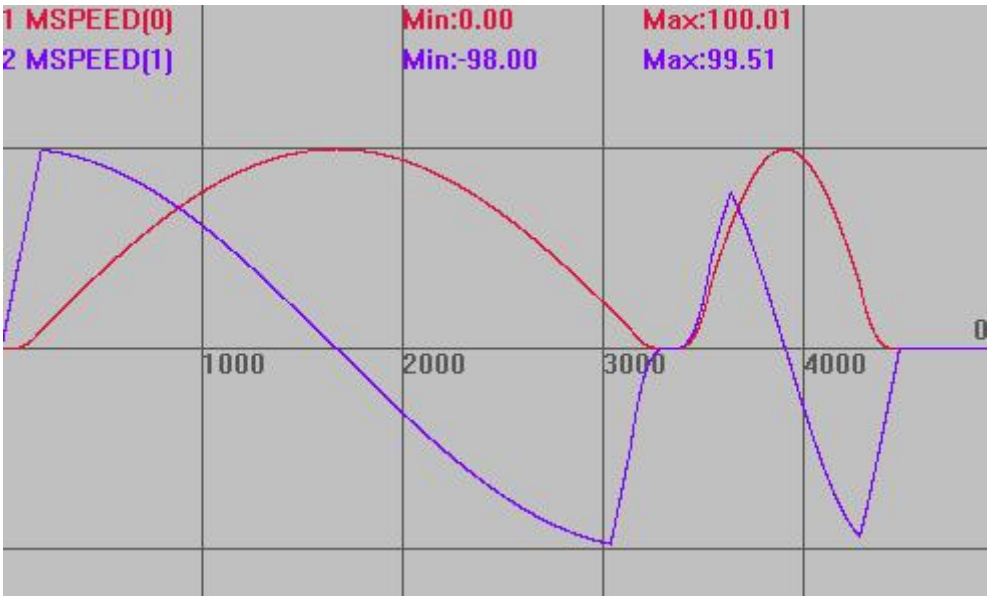
倒角不仅可以用于直线，还可以用于圆弧、螺旋等运动。
若本例程使用倒角得结果如下：

```
CORNER_MODE=32　　'启动倒角减速
ZSMOOTH=20　　'设置倒角半径
```

运动轨迹：



速度曲线：不限速



4.4 常见运动模式

常见的三种运动模式如下：

1．点位运动：仅对终点位置有要求，与运动的中间过程即运动轨迹无关。要求定位速度较快，在运动的加速段和减速段，采用不同的加减速控制策略。

2．连续轨迹运动：该控制又称为轮廓控制，系统在高速运动的情况下，既要保证系统加工的轮廓精度，还要保证轴的运动速度不受影响，对小线段加工时，有多段程序预处理功能。

3．同步运动：是指多个轴之间的运动协调控制，可以是多个轴在运动全程中进行同步，也可以是在运动过程中的局部有速度同步，主要应用在有电子齿轮箱和电子凸轮功能的系统控制中，产业上有印染、印刷、造纸、轧钢、同步剪切等行业。

4.4.1 凸轮

相关指令：

指令	说明	用法
CAM	凸轮表运动	CAM（凸轮表起始位置，结束位置，比例，运动距离）
CAMBOX	跟随凸轮表运动	参考指令章节
TABLE	保存凸轮表数据	TABLE（数据起始地址，数据区域）
MOVELINK	自动凸轮	定义参考轴和跟随轴，以及同步运动模式
MOVESLINK	自动凸轮 2	定义参考轴和跟随轴，以及同步运动模式

凸轮的作用是将旋转运动转换为线性运动，包括直线运动、摆动、匀速运动和非匀速运动。凸轮可分为两类：机械凸轮和电子凸轮。

1．机械凸轮

1）组成部分

机械凸轮机构是由凸轮，从动件和机架三个基本构件组成的高副机构，由凸轮的回转运动或往复运动推动从动件作规定往复移动或摆动的机构。

凸轮是一个具有曲线轮廓或凹槽的构件，通过在金属盘片上加工出一定形状的轮廓曲线，一般为主动件，作等速回转运动或往复直线运动。与凸轮轮廓接触，并传递动力和实现预定的运动规律的构件，一般做往复直线运动或摆动，称为从动件。机架作为支撑结构。如下图所示凸轮基本结构。

2）工作原理

凸轮具有曲线轮廓或凹槽，有盘形凸轮、圆柱凸轮和移动凸轮等，其中圆柱凸轮的凹槽曲线是空间曲线，因而属于空间凸轮。从动件与凸轮作点接触或线接触，有滚子从动件、平底从动件和尖端从动件等。尖端从动件能与任意复杂的凸轮轮廓保持接触，可实现任意运动，但尖端容易磨损，适用于传力较小的低速机构中。

为了使从动件与凸轮始终保持接触，可采用弹簧或施加重力。具有凹槽的凸轮可使从动件传递确定的运动，为确动凸轮的一种。一般情况下凸轮是主动的，但也有从动或固定的凸轮。多数凸轮是单自由度的，但也有双自由度的劈锥凸轮。

3）机构分类

机械凸轮机构的类型很多，通常按照凸轮和从动件的形状、凸轮形状分类。

从动件的分类：尖底从动件、滚子从动件和平底从动件；

凸轮形状可分为：盘形凸轮、移动凸轮、圆柱凸轮。

4）优缺点

优点：机械凸轮机构结构紧凑，最适用于要求从动件作间歇运动的场合。它与液压和气动的类似机构比较，运动可靠，因此在自动机床、内燃机、印刷机和纺织机中得到广泛应用。

缺点：由于凸轮机构属于高副机构，故凸轮与从动件之间为点或线接触，不便润滑、易于磨损，有噪声，盘片的加工制造要求较高，维修复杂。因此凸轮机构多用于传动力不大的控制机构和调节机构。

2．电子凸轮

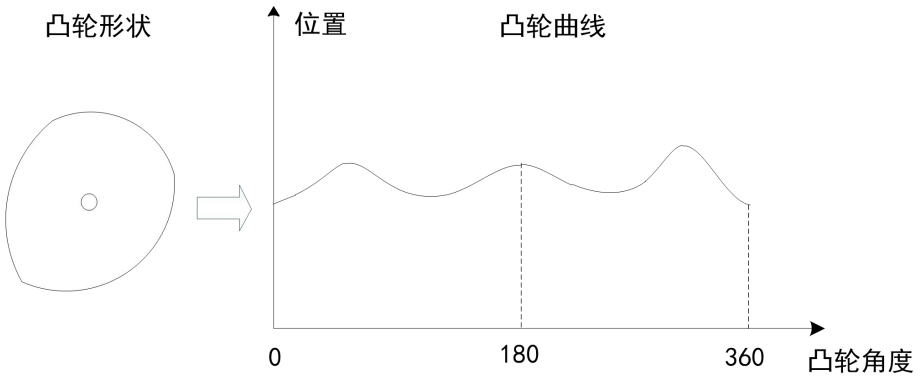
1）组成部分

电子凸轮是利用构造的凸轮曲线来模拟机械凸轮，以达到与机械凸轮系统相同的凸轮轴与主轴之间相对运动的软件系统，通过控制器控制伺服电机来模拟机械凸轮的功能，不需要另外安装机械结构。

2）工作原理

电子凸轮属于多轴同步运动，这种运动是基于主轴外加一个或多个从轴系统，是在机械凸轮的基础上发展而来，电子凸轮多用于周期性的曲线运动场合。

如下图，机械凸轮按照凸轮的轮廓可以得出一段转动角度与加工位置运动轨迹，此轨迹为弧线，将该段弧线分解成无数个直线或圆弧轨迹，组合起来得到一串趋近于该弧线的运动轨迹，电子凸轮直接将此段轨迹运动参数装入运动指令，即可控制轴走出目标轨迹。



3）优点

电子凸轮用软件来控制信号，改变程序的相关运动参数就能改变运动曲线，应用灵活性高，工作可靠，操作简单，不需要额外安装机械构件，因而不存在磨损的情况。

4）举例说明

凸轮指令有 CAM（凸轮表运动），CAMBOX（跟随凸轮表运动），MOVELINK（自动凸轮），MOVESLINK（自动凸轮 2）以 CAM 指令为例，CAM 指令包含四个参数，分别为 start point：TABLE 存储的起始点的位置，end point：TABLE 存储的结束点的位置，table multiplier：运动位置比例，distance：运动参考距离。

自动凸轮应用参见 MOVELINK 指令例程。

CAM 凸轮表运动的例程：

```
RAPIDSTOP(2)
WAIT IDLE(0)
ERRSWITCH = 3
BASE(0) '选择第 0 轴
ATYPE=1    '脉冲方式步进或伺服
DPOS = 0
UNITS = 100'脉冲当量
SPEED = 200
ACCEL = 2000
DECEL = 2000
TRIGGER
'计算 TABLE 的数据
DIM deg, rad, x, stepdeg
stepdeg = 5     '可以通过这个来修改段数，段数越多速度越平稳
FOR deg=0 TO 360 STEP stepdeg
    rad = deg * 2 * PI/360      '转换为弧度
    X = deg * 25 + 10000 * (1-COS(rad)) '计算每小段位移
    TABLE(deg/stepdeg,X)              '存储 TABLE
    TRACE deg/stepdeg,X
NEXT deg
WHILE 1'循环运动
    IF IN(0) = ON THEN '输入 0 有效启动运动
        CAM(0, 360/stepdeg, 0.1, 300)     '虚拟跟踪总长度 300
        WAIT UNTIL IDLE '等待运动停止
```

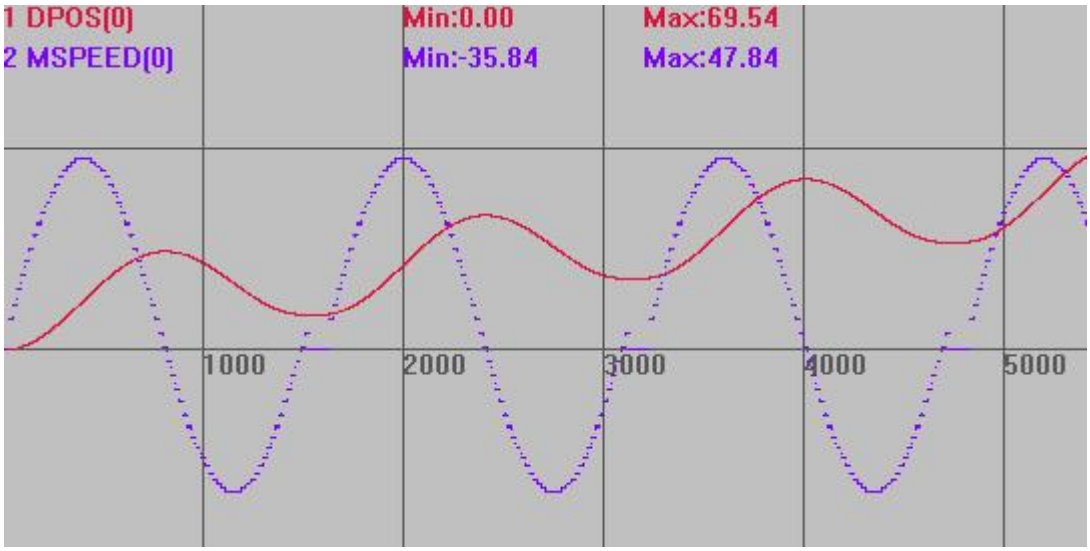
```
            DELAY(100) '延时
        ENDIF
    WEND
    END
```

编程思路：计算多段位移，将数据存储至 TABLE，将 TABLE 参数装入 CAM 指令，轴按照 TABLE 存储的位置指令开始往下走一个周期。

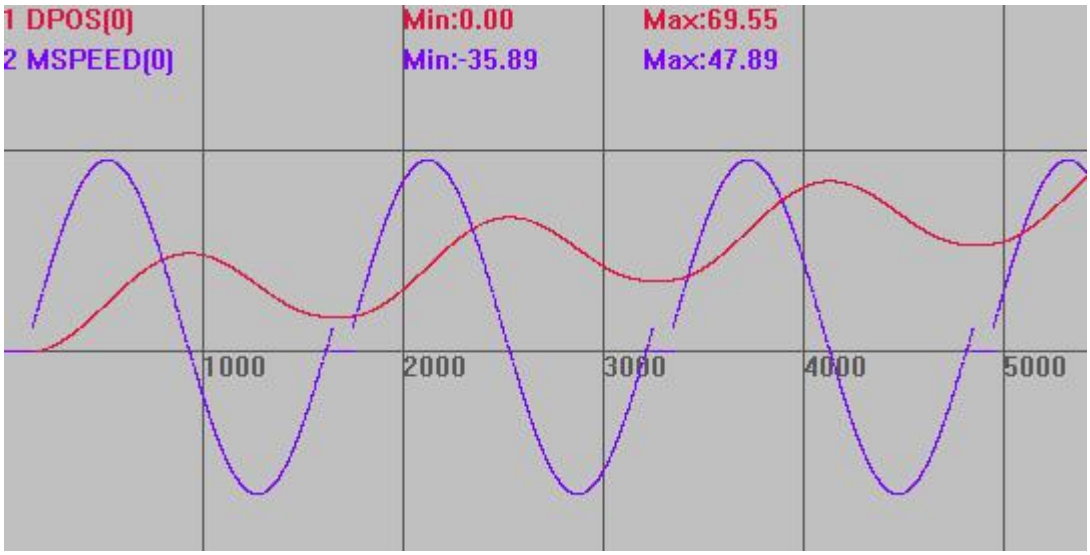
此处只有轴 0 运动，轴 1 不运动。

凸轮起始位置从 0 开始，根据 stepdeg 的值来设置运动段数，段数越多轨迹越平滑，设置 stepdeg = 5，表示凸轮表存储了 73 段运动轨迹（TABLE 位置 0-72），CAM 指令将存储的轨迹参数运行完一次就停止，需要持续运行可在程序中加入循环。

水平刻度 1000，垂直刻度 50，得到运动轨迹如下：



其他条件不变，仅设置 stepdeg =1，凸轮表存储的轨迹数增多，得到如下曲线，对比上下两个曲线可知，在实际应用过程中为确保轨迹精确，凸轮表存储轨迹时尽量细化。



4.4.2 电子齿轮

相关指令：

指令	说明	用法
CONNECT	连接轴运动	CONNECT（比率，连接轴轴号）
CONNPATH	连接轴运动	CONNPATH（比率，连接轴轴号）
CANCEL	取消连接	可以不带参数直接使用

与凸轮不同，电子齿轮的连接是线性的，凸轮可以做不规则运动。电子齿轮用电气控制技术来代替机械传动机构，可以用控制器装载配套指令来实现。

电子齿轮模式能够将两轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接。把被跟随的轴叫主轴，把跟随的轴叫从轴，通过将跟随轴按照某个比率连接到主轴上，从而达到主轴运动时，连接的跟随轴也跟随运动，连接的是脉冲个数，要考虑不同轴 UNITS 的比例。电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置或编码器位置。

传动比率：主轴速度与从轴速度的比例。电子齿轮能够灵活的设置传动比，比率可以通过重复调用指令动态变化，节省机械系统的安装时间。当主轴速度变化时，从轴会根据设定好的传动比自动改变速度。

以 CONNECT 为例，使用该指令时需要设置两个参数，连接的比例值和连接轴的轴号。

假设连接轴 0 的 UNITS 为 100，被连接轴 1 的 UNITS 为 10。使用 CONNECT 连接，连接比例 ratio 为 1，指令 CONNECT(1,1) AXIS(0)表示含义：当轴 0 运动 s0=100 时，轴 1 运动 s1=s0*UNITS(0)*ratio/UNITS(1)。

取消连接使用 CANCEL 指令。

电子齿轮的作用：

1．脉冲补偿，减少上位机负担（因为目前用的发送脉冲的元件，都有发送脉冲频率的限制）。

2．匹配电机发出的脉冲数与机械最小移动量，可将指令输入 1 个脉冲对应的工件（或电机）移动量设定为任意值；可实现电机的无极变速，在电机启动和停止时，可以防止失步和过冲现象，这样就能充分发挥电机的潜能。

3．传递同步运动信息，实现坐标的联动、运动形式之间的变换（旋转-旋转，旋转-直线，直线-直线）、简化控制等。

一般来说，电机与驱动机构是直连的，机械结构固定后，传动比也就固定了；利用电子齿轮可以增加传动系统的柔性，减少传动元件数量和传动链长度，还可以实现小数传动比，这样就提高了传动精度。

在没有电子齿轮之前，通常在外部设定速度指令不变的情况下，通过改变电机轴输出侧的机械齿轮的比值来改变电机的转速，而通过电子齿轮比的参数值设定，可以在机械设定的速度指令不变的情况灵活改变电机的连接速率比值以匹配用户的实际需求，由于是通过电子器件来实现类似机械齿轮的速比变化，所以称为电子齿轮比。

电子齿轮只能改变速比，并不具备机械齿轮的速比变化同时的转矩变化，所以需要将电机的输出转矩进行低速放大时，还是需要使用机械齿轮。

4.4.3 JOG 运动

相关指令：

指令	说明	用法
JOGSPEED	JOG 时的速度	JOGSPEED=速度值
FWD_JOG	正向 JOG 输入对应的输入口编号	FWD_JOG=输入编号

REV_JOG	负向 JOG 输入对应的输入口编号	REV_JOG=输入编号
FHSPEED	在 FHOLD_IN 被按下时的保持速度	FHSPEED=速度值
FHOLD_IN	保持输入对应的输入点编号	FHOLD_IN=输入编号
FAST_JOG	快速点动输入编号	FAST_JOG=输入编号

在 JOG 运动模式下，各轴可以独立设置目标速度、加速度、减速度、速度平滑等运动参数，能够独立运动或停止。

JOG 运动由开关信号控制，可以正向运动和负向运动，通过 FWD_JOG 指令映射正向点动开关、REV_JOG 指令映射负向点动开关，控制器收到信号输入时，对应轴按照 JOGSPEED 速度慢速运动，信号输入中断时轴减速停止，需要持续的 JOG 运动时，保持开关的输入状态即可。

控制器还支持快速点动，FAST_JOG 指令设置快速点动开关，按下快速点动开关轴以 SPEED 运动，没有按下开关时轴以 JOGSPEED 运动。

FHOLD_IN 映射保持输入设置，当有输入信号时，运动轴的速度按照 FHSPEED 的速度参数继续执行当前运动，当 FHOLD_IN 取消输入，轴继续运动，但是运动速度变回 SPEED 速度。

当 REV_JOG 和 FWD_JOG 同时有信号输入时，轴按照 FWD_JOG 正向运行。

4.4.4 手轮

相关指令：

指令	说明	用法
CONNECT	连接手轮	CONNECT（比率，连接轴轴号）
CANCEL	取消手轮连接	可以不带参数直接使用

手轮也称为手动脉冲发生器、手脉、手摇脉冲发生器等。用于数控机床、印刷机械等的零位补正和信号分割。当手轮旋转时，编码器产生与手轮运动相对应的信号。通过数控系统选定坐标并对坐标进行定位。

手轮功能是指通过特定的编码器作为手轮脉冲变化输入源，检测编码器脉冲输入变化，在根据手轮设定的参数，驱动手轮跟随轴运动，该功能主要用于插补运动中辅助运动，指定的编码器可以为端子板上的编码器，也可以是 EtherCAT 总线上的编码器模块。

手轮跟随运动属于位置紧跟型，即手轮脉冲变化 n 个，跟随轴跟随运行 n*ratio 个脉冲，速度，加速度按主轴的参数进行规划。

进入手轮运动的条件：跟随手轮的轴处于静止状态，无运动；编码器处于未绑定轴，可用于轴的位置反馈；跟随轴处于使能状态；跟随轴未被设置成手轮模式。

退出手轮模式使用 CANCEL 指令。手轮连接比率可随时切换。

使用流程：设置手轮轴和跟随轴类型，设置各项基本运动参数，使用 CONNECT 指令对手轮和跟随轴按照一定比率进行连接，此时手轮就可以带动跟随轴运动，运动完成 CANCEL 取消手轮连接。

例程：跟随手轮运动

```
RAPIDSTOP(2)
WAIT IDLE(0)
ERRSWITCH = 3
CONST axishand = 0
BASE(axishand)    '选择第 0 轴接手轮
ATYPE=6   '脉冲+方向的手轮，正交输入手轮使用 3
BASE(1)    '轴 1 被手轮控制
ATYPE=1    '配置为脉冲轴
```

```
DPOS = 0,0
UNITS = 100,100    '脉冲当量，每 MM100 脉冲
SPEED = 200,200
ACCEL = 2000,2000
DECEL = 2000,2000
SRAMP = 20
CLUTCH_RATE = 0    '采用速度加速度来进行限制
DIM poslast
poslast = DPOS
WHILE 1
    IF IN(0) = ON AND IN(1) = OFF THEN
        CONNECT(1, axishand)'链接到轴 0，倍率  1
    ELSEIF IN(1) = ON AND IN(0) = OFF THEN
        CONNECT(10, axishand)     '链接到轴 0，倍率  10
    ELSEIF IN(0)= ON AND IN(1) = ON THEN
        CONNECT(50, axishand)          '链接到轴 0，倍率 50，对步进，倍率太高会出现丢步或长时间
才能结束
    ELSEIF MTYPE = 21 THEN
        CANCEL          '取消 CONNECT
    ENDIF
    IF poslast <> DPOS THEN
        poslast = DPOS
        TRACE DPOS
    ENDIF
WEND
END
```

4.5 原点回零

相关指令：

指令	说明	用法
DATUM	原点回零模式选择	DATUM（回零模式值）
DATUM_IN	原点开关输入	DATUM_IN =输入编号
FWD_IN	正向限位开关	FWD_IN =输入编号
REV_IN	负向限位开关	REV_IN =输入编号
SPEED	快速找原点速度	直接设置速度值
CREEP	找原点爬行速度	直接设置速度值
INVERT_IN	信号反转	INVERT_IN =（输入编号，ON/OFF）

在高精度自动化设备上都有自己的参考坐标系，工件的运动可以定义为在坐标系上的运动，坐标系的原点即为运动的起始位置，各种加工数据都是以原点为参考点计算的，所以启动控制器执行运动指令之前，设备都要进行回零操作，回到设定的参考坐标系原点，若不进行回零操作，会导致后续运动轨迹错误。

正运动控制器提供了多种回零方式，通过 DATUM 指令设置，不同模式值选择不同的回零方式。此指

令为单轴回零指令，多轴回零时，需要对每个轴都使用 DATUM 指令。

Z 信号回零必须配置为带 Z 信号 ATYPE。原点开关的输入点由参数 DATUM_IN 决定，正负限位开关通过 FWD_IN 和 REV_IN 设置。

运动控制器为 0 触发有效，输入为 OFF 状态时，表示到达原点/限位，常开类型信号需要采用 INVERT_IN 反转电平。

值	描述
0	清除所有轴的错误状态。
1	轴以 CREEP 速度正向运行直到 Z 信号出现。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
2	轴以 CREEP 速度反向运行直到 Z 信号出现。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
3	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。 找原点阶段碰到正限位开关会直接停止。 爬行阶段碰到负限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS
4	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关。 找原点阶段碰到负限位开关会直接停止。 爬行阶段碰到正限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
5	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
6	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关，然后再继续以爬行速度正转直到碰到 Z 信号。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
8	轴以 SPEED 速度正向运行，直到碰到原点开关。 碰到限位开关会直接停止。
9	轴以 SPEED 速度反向运行，直到碰到原点开关。 碰到限位开关会直接停止。
21	使用 Ethercat 驱动器回零功能，此时 mode2 有效。 设置驱动器回零方式（6098h），缺省 0 表示使用驱动器当前的回零方式。 会使用轴的 SPEED, CREEP, ACCEL, DECEL，乘以 UNITS 后自动设置驱动器的 6099h，609Ah 动作时序： 6098h 回零方式→6099h 速度→609Ah 加速度→6060h 切换当前模式

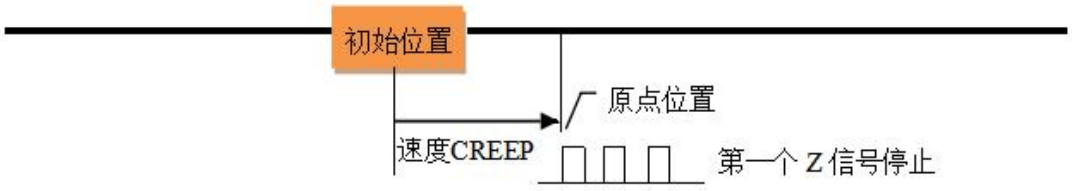
对于原点在正负限位中间的情况，在各个模式上加 10，表示在回零过程中碰到了限位不取消运动，而是继续反向去找原点，其他条件均与原模式相同。由于原点在正负限位开关之间，因此在回零途中至多遇到一个限位开关。以下回零方式 5~8 均为加 10 模式。

总线控制器使用以上控制器找原点模式完成后，需要手动清零 MPOS。回零模式加 100（模式 100+n

和 110+n 分别对应 n 和 10+n），表示接入编码器后可以自动清零 MPOS(仅限 4 系列，ATYPE=4)

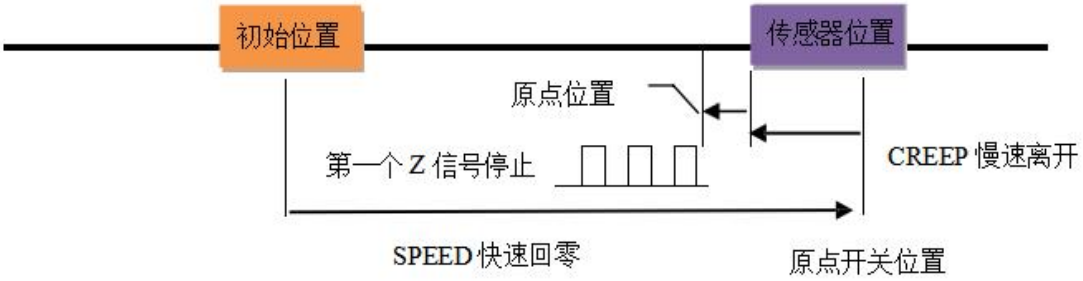
常见回零方式详解：

方式 1：Z 信号模式，轴以 CREEP 速度运行直到 Z 信号出现。DPOS 值自动重置为 0 同时纠正 MPOS。只有在 ATYPE 设置为 4 或 7，并且将对应轴编码器 Z 相接入时有效，回零途中碰到正负限位开关直接停止。mode=1 时正向回零，mode=2 时负向回零。



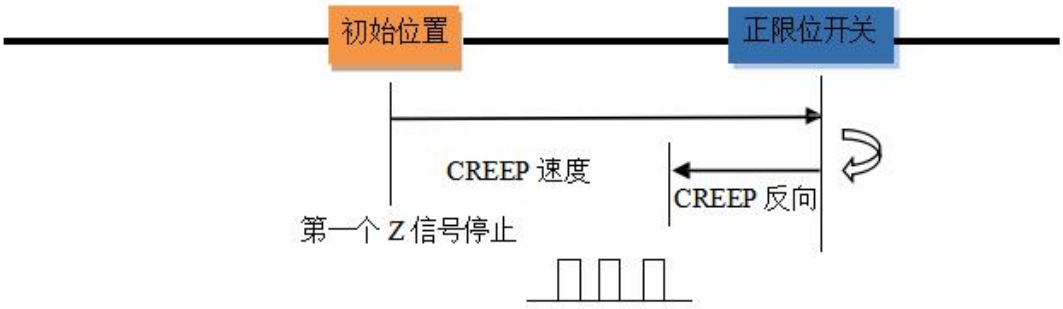
方式 2：原点+反找模式，轴以 SPEED 速度向原点运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。DPOS 值自动重置为 0 同时纠正 MPOS，回零途中碰到正负限位开关直接停止。mode=3 时正向回零，mode=4 时负向回零。

方式 3：mode=5，原点+反找+Z 信号模式，轴以 SPEED 速度向原点运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。DPOS 值自动重置为 0 同时纠正 MPOS。只有在 ATYPE 设置为 4 或 7，并且将对应轴编码器 Z 相接入时有效，回零途中碰到正负限位开关直接停止。mode=5 正向回零，mode=6 负向回零。

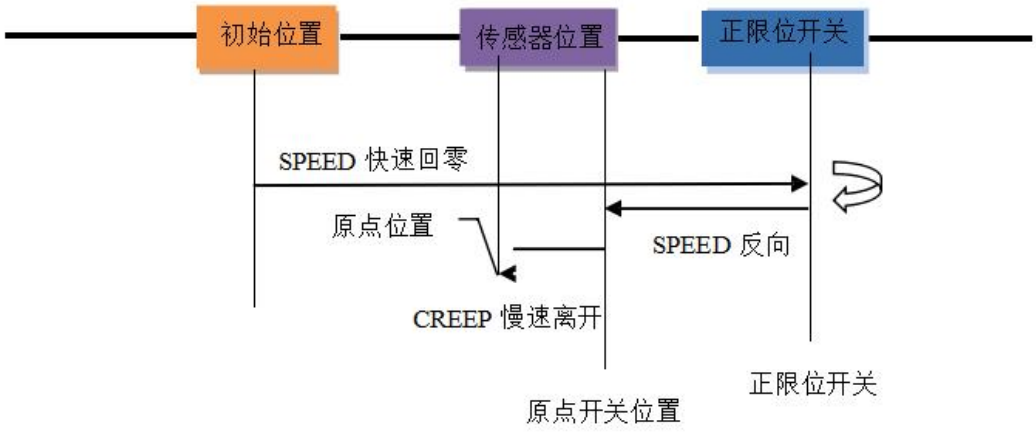


方式 4：mode=8，原点一次回零模式，轴以 SPEED 速度向运行，直到碰到原点开关。DPOS 值自动重置为 0 同时纠正 MPOS，回零途中碰到正负限位开关直接停止。mode=8 正向回零，mode=9 负向回零。

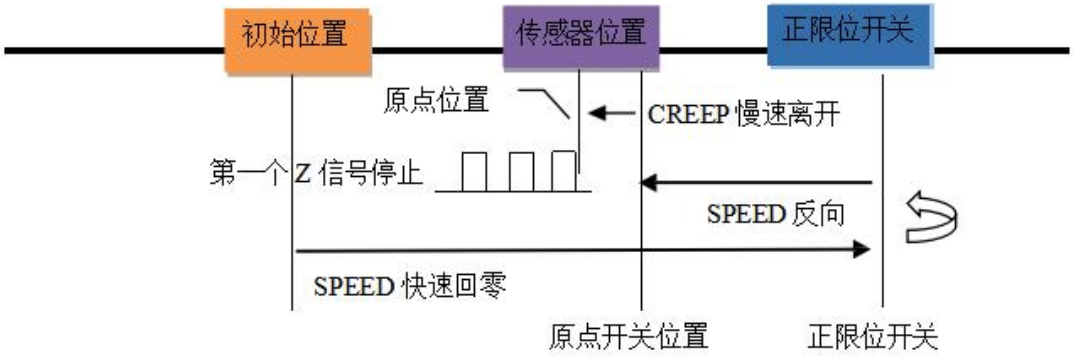
方式 5：mode=11，Z 信号模式，轴以 CREEP 速度运行，遇到限位开关不停止，继续以 CREEP 速度方向运行直到 Z 信号出现。



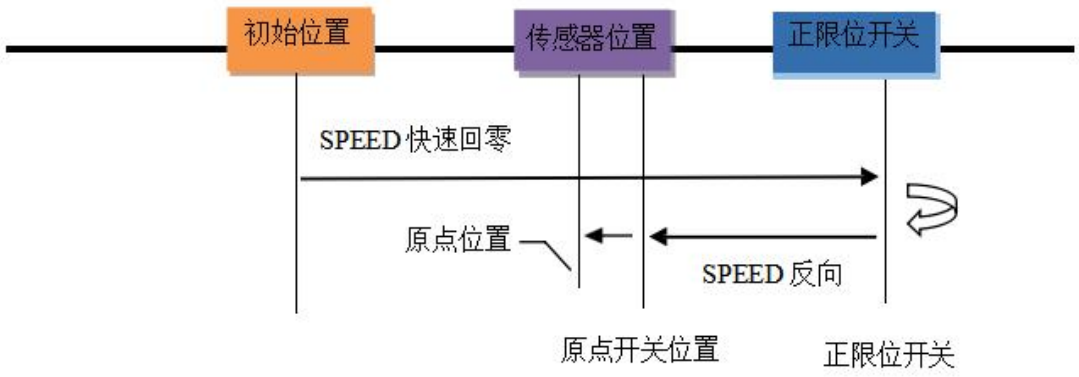
方式 6：mode=13 正向运行，原点+反找模式+限位反向，轴以 SPEED 速度向原点运行，碰到正向限位开关后不停止，再以 SPEED 速度反向运行直到碰到原点开关，然后轴以 CREEP 速度慢速运动直到离开原点开关。



方式 7：mode=15，原点+反找+Z 信号模式，轴以 SPEED 速度向原点运行，中途遇到限位开关不停止，继续以 SPEED 速度反向运动，直到碰到原点开关，然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。



方式 8：mode=18，原点一次回零模式，轴以 SPEED 速度向运行，中途遇到限位开关不停止，继续以 SPEED 速度反向运动，直到碰到原点开关后停止。



4.6 限位相关指令：

指令	说明	用法
FS_LIMIT	正向软限位设置	FS_LIMIT=正限位位置
RS_LIMIT	负向软限位设置	RS_LIMIT=正限位位置
FWD_IN	映射正限位输入	FWD_IN=输入编号
REV_IN	映射负限位输入	REV_IN=输入编号

运动控制器能够通过安装限位开关或者设置软限位来限制各轴的运动范围。硬限位开关和软限位开关用于工艺对象轴的允许运动范围和工作范围。

硬限位开关是限制轴的最大“允许行进范围”的限位开关。硬限位开关是物理开关元件，硬限位开关由指令映射到相应输入开关信号上，根据开关信号是常开还是常闭确定是否要对信号进行翻转，设置完成后，碰到硬限位开关，对应轴立即停止运动，停止减速度为 FASTDEC。

与硬限位开关不同，软限位开关只通过软件程序设定来实现，而无需借助外部的开关元件。软限位开关将限制轴的“工作范围”，由指令直接设置限位位置，轴走到设置位置后立即停止运动，它们应位于机床限制行进范围的相关硬限位开关的内侧。由于软限位开关的位置较为灵活，因此可根据当前的运行轨迹和具体要求调整轴的工作范围。

工作台碰到限位开关或者规划位置超越软限位时，运动控制器紧急停止工作台的运动。限位触发以后，轴无法继续运动，此时需要调整轴的位置，使其远离限位位置才能重新开始运动。

正/负软限位 FS_LIMIT 和 RS_LIMIT 的值需要位于-REP_DIST 和+REP_DIST 之间，软限位参数才起作用，否则正/负向软限位设置无效。取消软限位时，建议不要去修改 REP_DIST 的值，将 FS_LIMIT 和 RS_LIMIT 设置一个较大值即可。

例程：正、负软限位的应用

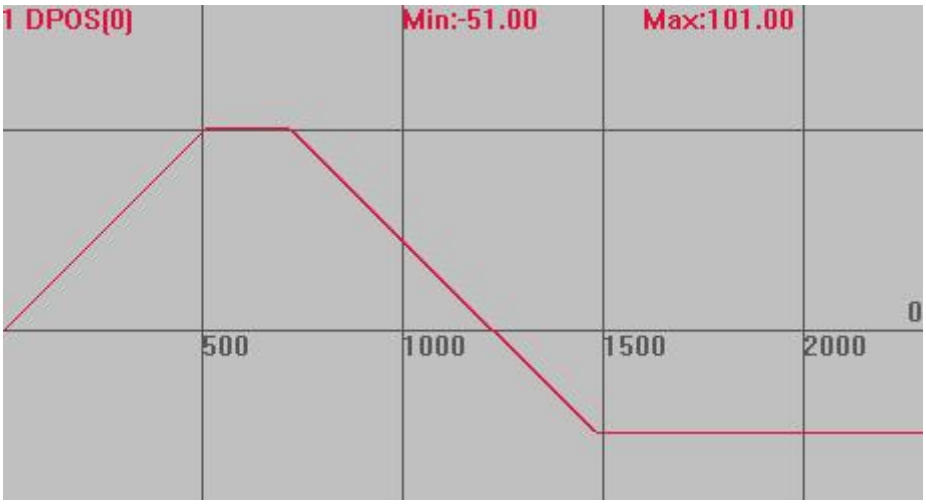
ERRSWITCH = 3

```
RAPIDSTOP(2)
WAIT IDLE
BASE(0)            '选择 X 轴运动
DPOS = 0
ATYPE=1            '脉冲方式步进或伺服
UNITS = 100        '脉冲当量，每 mm100 脉冲
SPEED = 200
ACCEL = 20000
DECEL = 20000
TRIGGER
'设置软限位
REP_DIST = 100000000    '缺省不用修改这个值
RS_LIMIT = -50          '负向软限位，必须大于-REP_DIST 才会生效
FS_LIMIT = 100          '正向软限位，必须小于 REP_DIST 才会生效
VMOVE(1)           '正向持续运动
WAIT  UNTIL  AXISSTATUS  AND (512)  '判断正向限位是否发生
WAIT IDLE
PRINT "SOFTLIMT FS", *DPOS
DELAY(200)
VMOVE(-1)          '负向持续运动
WAIT  UNTIL  AXISSTATUS  AND (1024)    '判断负向限位是否发生
WAIT IDLE
PRINT "SOFTLIMT RS", *DPOS
RS_LIMIT = -200000000       '关闭软限位
FS_LIMIT = 200000000
END
```

打印结果：

```
Axis:0 AXISSTATUS:200h,FSOFT
SOFTLIMT FS      101
Axis:0 AXISSTATUS:400h,RSOFT
SOFTLIMT RS      -51
```

运动轨迹如下，正向软限位设置的 100，使得轴运动到 100 位置后被迫停止，负向软限位设置的-50，轴负向运动到此位置后无法继续运动。



4.7 位置锁存

相关指令：

指令	说明	用法
REGIST	设置锁存方式	REGIST（方式值）
REG_INPUTS	锁存输入口映射	REG_INPUTS=$输入口编号
MARK	判断锁存是否触发	WAIT UNTIL MARK
MARKB	判断第二个锁存是否触发	WAIT UNTIL MARKB
MARKC	判断第三个锁存是否触发	WAIT UNTIL MARKC
MARKD	判断第四个锁存是否触发	WAIT UNTIL MARKD
REG_POS	保存锁存的测量反馈位置	打印 REG_POS
REG_POSB	返回锁存 2 的测量反馈位置	打印 REG_POSB
REG_POSC	返回锁存 3 的测量反馈位置	打印 REG_POSC
REG_POSD	返回锁存 4 的测量反馈位置	打印 REG_POSD
OPEN_WIN	锁存触发的开始坐标范围点	OPEN_WIN=POS
CLOSE_WIN	锁存触发的结束坐标范围点	CLOSE_WIN=POS

锁存就是把信号暂存以维持某种电平状态，输出端的状态不会随输入端的状态变化而变化，仅在有锁存信号时输入的状态才被保存到输出，直到下一个锁存信号到来时才改变。使用锁存功能可以将数据暂存，方便读取。

控制器的锁存功能主要用来锁存编码器的位置 MPOS（4 系列及以上控制器最新固件支持虚拟轴、脉冲轴锁存），当锁存信号被触发时，当前位置信息立即被捕获到位置锁存器中，并将前一次锁存的位置坐标清除。读取锁存位置信息时，读取的是最后一次锁存信号触发时锁存的位置信息。不同型号控制器锁存通道口个数以及位置有所差别，参见相应型号控制器的硬件手册。

需要注意的是位置锁存的操作接口是按轴号进行访问的，不同类型的轴锁存参数不同，使用前首先确定轴的类型，支持锁存的轴类型分为以下几种：本地脉冲轴、EtherCAT 轴、RTEX 轴，还需要注意控制器锁存口个数，避免锁存数据溢出导致错误。

总线轴类型采用 R0，R1，Z 脉冲这三种锁存；脉冲轴类型采用 R2，R3 锁存。

EtherCAT 总线控制器除了支持控制器锁存还支持驱动器锁存，此时使用驱动器 IO 点实现锁存，具体

模式查看指令语法。RTEX 只支持控制器锁存。

支持 EtherCAT 驱动器锁存与控制器锁存同时使用时，需要有 4 锁存通道功能。4 个通道指的是 MARK、MARKB、MARKC、MARKD，通过 REG_INPUTS 指定锁存输入口对应的锁存通道。当锁存产生时，轴状态 MARK 会被设置为 ON，同时锁存到的位置会被存储在参数 REG_POS 内。

每个轴的输入信号 R0、R1、Z 信号可以使用锁存功能，R0、R1 输入一般对应到输入口 0 和 1。当使用两个信号锁存时，第二个信号锁存使用 MARKB 和 REG_POSB，MARK 和 REG_POS 需配对使用，即编号一致。

上升沿/下降沿是以控制器内部状态而言，不同类型的输入口可能不一致，需要确认实际锁存的边沿。

锁存功能使用方法：

1）确定当前硬件条件是否满足锁存需求，确定需要锁存位置的轴；

2）设置锁存输入映射口；

3）设置锁存模式，锁存触发；

4）锁存完成打印锁存位置信息；

5）可读取锁存位置起始坐标和结束坐标，锁存位置可被其他指令调用。

锁存对应参数：REGIST（mode）

值	描述
1	当 Z 脉冲上升沿时的绝对位置送到 REG_POS
2	当 Z 脉冲下降沿时的绝对位置送到 REG_POS
3	当输入信号 R0 上升沿的绝对位置送到 REG_POS
4	当输入信号 R0 下降沿的绝对位置送到 REG_POS
6	输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB
7	输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REG_POSB
8	输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB
9	输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REG_POSB
10	输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB
11	输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB
12	输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB
13	输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB
14	输入信号 R1 上升沿时的绝对位置送到 REG_POSB(14 以后 150804 以后版本支持，每个锁存通道独立，支持 4 通道锁存)
15	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
16	Z 信号上升沿时的绝对位置送到 REG_POSB
17	Z 信号下降沿时的绝对位置送到 REG_POSB
18	输入信号 R2 上升沿时的绝对位置送到 REG_POSC
19	输入信号 R2 下降沿时的绝对位置送到 REG_POSC
20	输入信号 R3 上升沿时的绝对位置送到 REG_POSD
21	输入信号 R3 下降沿时的绝对位置送到 REG_POSD

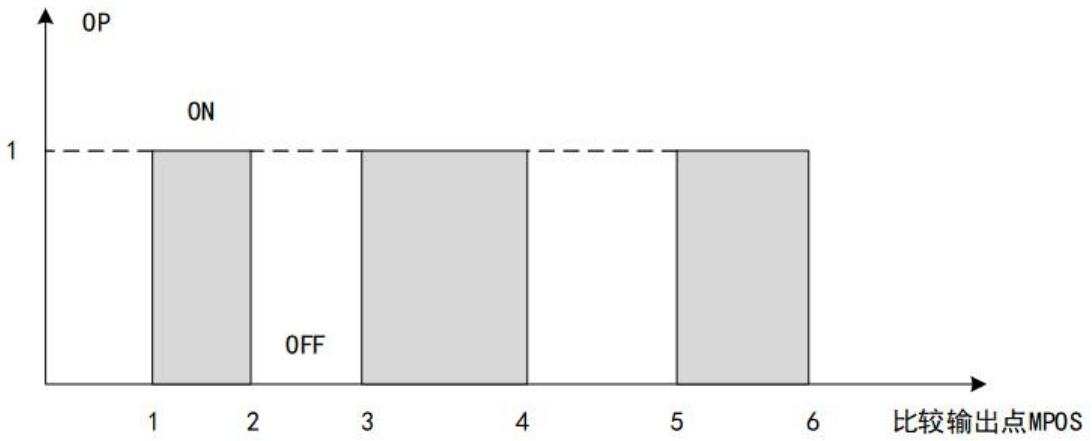
100+mode: 只能使用单一通道的 mode， 通过把模式加 100 来支持连续锁存，锁存结果存储到 TABLE 里面。分别对两个通道进行连续锁存，可以实现上下边沿的连续锁存。（4 系列控制器 20170523 以上固件支持）

值	描述
1	当 Z 脉冲上升沿时的绝对位置送到 REG_POS
2	当 Z 脉冲下降沿时的绝对位置送到 REG_POS
3	当输入信号 R0 上升沿的绝对位置送到 REG_POS
4	当输入信号 R0 下降沿的绝对位置送到 REG_POS
14	输入信号 R1 上升沿时的绝对位置送到 REG_POSB
15	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
16	Z 信号上升沿时的绝对位置送到 REG_POSB
17	Z 信号下降沿时的绝对位置送到 REG_POSB
23	当输入信号 R0 上升沿的绝对位置送到 REG_POSB
24	当输入信号 R0 下降沿的绝对位置送到 REG_POSB
33	当输入信号 R0 上升沿的绝对位置送到 REG_POS，下一次切换下降沿，轮流切换
34	当输入信号 R0 下降沿的绝对位置送到 REG_POS，下一次切换上升沿，轮流切换
35	当输入信号 R1 上升沿的绝对位置送到 REG_POSB，下一次切换下降沿，轮流切换下一次切换下降沿，轮流切换
36	当输入信号 R1 下降沿的绝对位置送到 REG_POSB，下一次切换上升沿，轮流切换

4.8 硬件比较输出

运动控制器内有位置比较单元，硬件比较输出是通过比较轴是否到达设定位置，来操作输出口动作，一般使用时将编码器位置与设定位置比较，当编码器的位置到达一个设定比较位置时，触发相应输出口电平翻转一次。

如下图所示，到达设置的位置 1，电平翻转，到达位置 2 电平再次翻转，到达位置 3 电平再翻转，直达比较完所有的点后，电平维持最后一次翻转后的状态。



目前只有 4 系列的脉冲轴与编码器轴、ZMC304X-HW、ECI2410-HW、ZMC306E、ZMC308B 支持此功能，需要最新固件，具体使用方法查看第十一章 HW_PSWITCH 和 HW_PSWITCH2 指令、第六章的 MOVE_OP 指令。

输出位置比较数据有两种方式：TABLE 表指定位置输出和周期输出模式。

若比较位置为大量连续的等间距输出，可使用周期模式比较数据输出，此时需设置起始比较输出位置、间隔距离以及重复周期。

若比较位置为非等间距位置值，可使用 TABLE 表指定位置进行输出，将需要比较输出的位置数据存储到 TABLE 表里，轴走到设定位置后控制输出口开启和关闭，此时需保证 TABLE 位置数据在所有比较点完成前不要修改，且 TABLE 表中的数据为单调增加的正向距离值或单调减少的负向距离值，否则会出现错误。

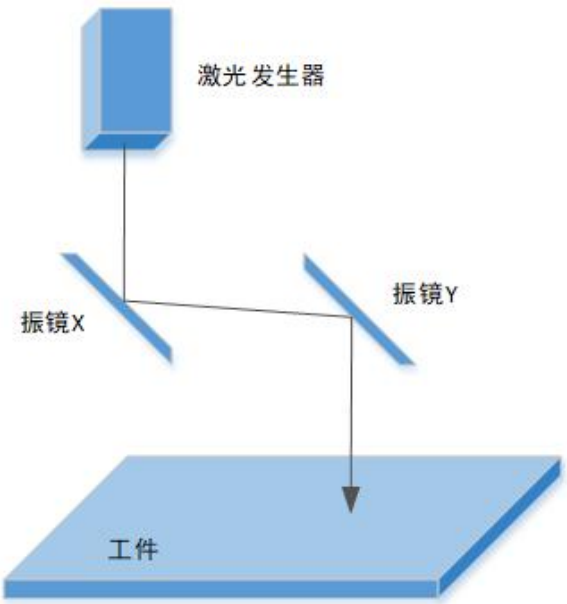
比较主轴带编码器输入时，自动使用编码器位置来触发，可以使用 MOVEOP_DELAY 参数来调整输出准确时刻。不同的总线驱动器效果可能有差异，也可以通过 MOVEOP_DELAY 参数来调整。

4.9 振镜控制系统

4.9.1 振镜说明

1．振镜工作原理

激光振镜是一种专门用于激光加工领域的特殊的运动器件，它靠两个振镜反射激光，形成 XY 平面的运动。激光振镜不同于一般的电机，激光振镜具有非常小的惯量，且在运动的过程中负载非常小，只有两个小的反射镜片 X 和 Y，分别用不同的电机控制偏转，系统的响应非常快。

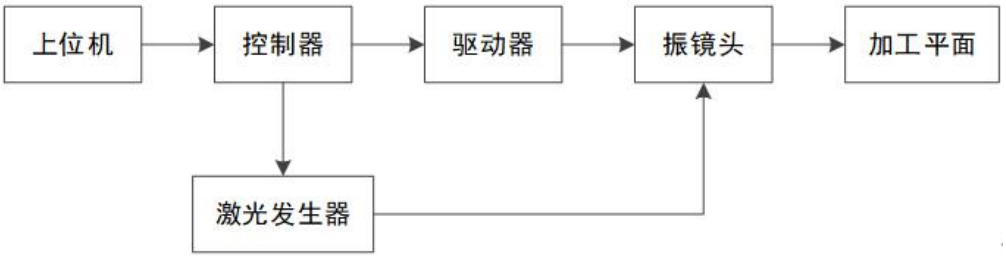


激光振镜运动两种基本的运动：一种为跳转运动，一 种为打标运动。

跳转运动的过程中，轴移动到要加工的位置，激光呈关闭状态，不影响轨迹的加工，因此可以以很大的速度运动。打标运动过程中，激光呈开启状态，进行轨迹的加工，因此用户需要根据实际加工要求设置合适的运动的速度。

振镜是一种优良的矢量扫描器件。它是一种特殊的摆动电机（激光振镜），基本原理是通电线圈在磁场中产生力矩，但与旋转电机不同，其转子上通过机械纽簧或电子的方法加有复位力矩，大小与转子偏离平衡位置的角度成正比，当线圈通以一定的电流而转子发生偏转到一定的角度时，电磁力矩与回复力矩大小相等，故不能象普通电机一样旋转，只能偏转，偏转角与电流成正比。

2．振镜控制系统基本结构



振镜系统的由以上几个部分组成一个基础系统，其中振镜头主要元件为 X/Y 两个反射镜片、分别控制 X/Y 镜片旋转的两个电机，根据实际需求还可加入人机操作系统、编码器等。

3．对控制器的基本要求

因为激光打标机是靠 X/Y 振镜偏转的配合，将激光反射到工作台面上，进行精确的雕刻。而振镜的控制是由控制器开环控制的，所以要求必须为线性，即输入信号同偏转角度之间为线性关系。因振镜是快速精密机械，所以要求从一个工作状态到另一个工作状态要求加速度越大越好，这样，打标空等时间就无限小。

振镜运动采用缓存区运动方式，即用户需要向轴运动缓存区传递运动及工艺数据，然后启动缓存区运动，运动控制器则会依次连续执行用户所传递的运动数据，直到所有的运动数据全部运动完成。

在激光振镜运动控制系统中不但有运动的控制，还有激光的控制。如何有效地处理振镜运动和激光开关的配合是一个很重要的问题，只有有效的协调了激光和运动的关系，才能运动出精确的轨迹。

运动控制：打标运动时，激光会按照设定的打标速度沿着给定的打标轨迹运动，在执行打标相关指令时，激光振镜运动控制器会自动开启激光。如果下一条仍是打标指令，激光一直呈开启状态，直到最后一条打标指令结束，或缓存区指令执行完毕，中途在缓冲区若遇到跳转指令，则激光自动关闭，直到遇到打标指令，激光才重新开启。开始运动前为保证打标轨迹正确需调整振镜坐标，同时清空缓冲区。

激光控制：主要包括控制激光的开关控制与发出激光的时长，控制激光的开断使用 OP 指令，激光能量的控制可根据激光器的不同，对应通过模拟量，数字量输出口，以及输出口 PWM 的占空比对应控制能量的大小。

4．主要应用

主要用于激光打标，包括激光切割、舞台灯光控制、激光打孔等。是一种非接触式、无污染、无磨损的新标记工艺，采用自动化控制，可靠性大大提高。激光打标是利用高能量密度的激光束，随着激光束在材料表面有规律的移动，同时控制激光束的开断，使目标材料表面发生物理或化学变化，激光束就可以在材料表面加工出一个指定的图案。

相比传统标记工艺，激光打标有如下优点：

标记速度快，字迹清晰。

非接触式加工，污染小，无磨损。

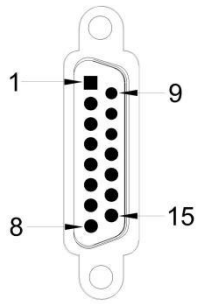
工作方便，防伪能力强。

高速自动运行，生产成本低，运行可靠。

5．ZMC420SCAN 控制器振镜接口信号

ZMC420SCAN 是一款支持激光振镜控制的控制器，自带的每个通用输出口都支持 PWM 功能。

本地轴号 4/5 可以 ATYPE=21 配置为第 1 个振镜，本地轴号 6/7 可以 ATYPE=21 配置为第 2 个振镜，通过 AXIS_ADDRESS 配置可以更改轴号。



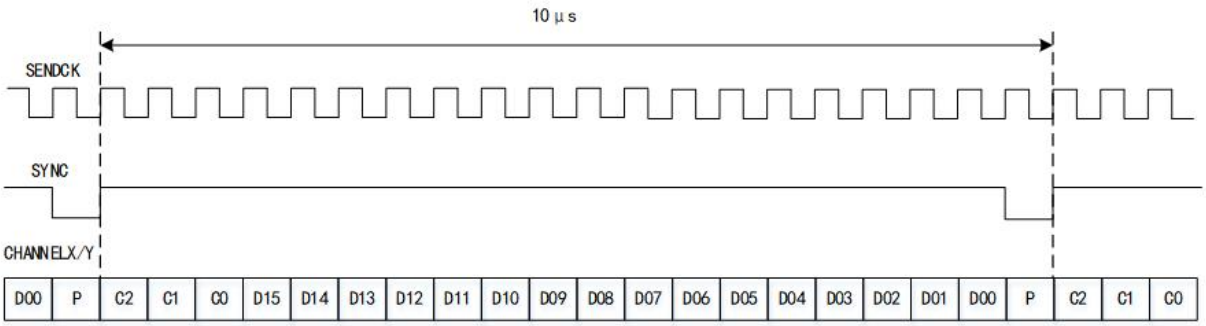
针脚号	信号	说明
1	CLOCK-	时钟信号-
2	SYNC-	同步信号-
3	X channel-	振镜 X 通道信号-
4	Y channel-	振镜 Y 通道信号-
5	NC	保留
6	STATUS-	振镜状态信号-
7	NC	保留
8	GND	数字地
9	CLOCK+	时钟信号+
10	SYNC+	同步信号+
11	X channel+	振镜 X 通道信号+
12	Y channel+	振镜 Y 通道信号+
13	NC	保留
14	STATUS+	振镜状态信号+
15	GND	数字地

6．XY2-100 振镜协议

ZMC420SCAN 支持 XY2-100 振镜协议。

在振镜控制系统中，XY2-100 协议作为数字化激光扫描振镜的接口定义及通信协议被广泛地使用。通讯协议是指双方实体完成通信或服务所必需遵循的规律和约定，XY2-100 协议包括四路信号：SENDCK（时钟信号）、SYNC（同步信号）、CHANNEL X（X 通道数据）、CHANNEL Y（Y 通道数据），这四路信号是一种同步串行传输过程。

数据时序结构如下图所示：



SENDCK 信号是一个频率为 2MHz 的时钟信号，当它从低电平到高电平跳变时，数据位被写入；当它从高电平到低电平跳变时，数据位被反射系统采样。

SYNC 信号用于提供数据转换的同步信息，当它从低电平到高电平时第一位数据被发送；从高电平到低电平时最后一位校验位被发送。

CHANNEL X/Y 是数据信号，它由 20 位组成，其中 C2、C1、C0 是振镜运动方向值，参考值为 001，D15 ~D0 是数据位，它是 16 位二进制数，用来控制振镜转过的角度大小，最后一位 P 为奇偶校验位，当发送的数据中有偶数个“1”时，对应校验位为“0”，当发送的数据中有奇数个“1”时，对应校验位为“1”。

7．振镜矫正

振镜一般是需要通过对振镜的矫正来实现，控制振镜移动准确的位置距离，振镜矫正实际上是进行一个理论振镜移动距离与实际振镜移动距离建立一种对应关系，然后在移动的过程中将对应的移动距离与建立的关系相结合，从而达到振镜准确移动位置的目的，达成振镜矫正的效果。

振镜矫正指令如下：

ZSCAN_CORRECT(ixy,imode,imaxline,imaxrow,x1,y1,x2,y2,tableindex)

ixy：值为 0 或 1，两个振镜选择；0-第一个振镜，1-第二个振镜

imode：0-关闭矫正功能；1-使用分区矫正

imaxline：行数，Y 方向的点数为行数

imaxrow：列数，X 方向的点数为列数

x1,y1,x2,y2：理论的左下角与右上角的位置

tableindex：测量的实际坐标开始存储的 table 索引，先 X 再 Y，先第一行（按列数存储），再下一行

振镜矫正点数最多支持 64*64 的矫正点数量，以建立左下角以及右上角的理论坐标，并且通过该理论坐标与写入对应 TABLE 数组中的测量出的实际图形坐标进行对应的处理，对当前连接振镜接口的振镜轴进行矫正，振镜矫正参数是断电不保存的，因此在使用过程中需注意，重新上电后需要重新进行矫正振镜。

振镜是绝对值系统, 上电之后控制器就一直处于和电机通信的状态, 修改振镜轴 DPOS 会引起振镜的偏移，因此在振镜使用过程中不要随便修改振镜轴的 DPOS 值，如果需要运动，可以通过 MOVEABS 移动到对应的位置。

4.9.2 振镜应用流程

1．在振镜轴的使用中需先将对应连接的振镜轴 4、5（6、7）设置轴类型为 21。

2．对对应振镜轴设置轴参数，设置的轴脉冲会影响运动时候的振镜运动距离，因此可将脉冲当量固定为大小，之后通过振镜矫正指令，矫正当前位置的振镜轴运动正确的距离移动。

3．如果在振镜运动过程中需要对激光的开关光操作，应该选定高速输出口进行控制开关光，并打开对应的精准输出设置，从而达到输出口在达到位置后短时间内进行出光，达成对激光的准确控制。

4．如果需要振镜轴回零，可以通过 MOVEABS 指令将振镜轴运动到 0 的位置，并且在振镜运动的过程中不能随便进行修改振镜轴的 DPOS 值，否则会造成振镜轴的偏移，对应的振镜电机也会抖动。

5．振镜轴可以通过轴映射 AXIS_ADDRESS 指令进行轴号调换，通过其他轴号进行操作振镜轴，达到调换轴的目的，除此之外当前振镜轴的方向修改无法通过指令进行修改振镜轴的方向，需要在振镜矫正的部分对需要反向的振镜坐标进行取反操作，之后再重新进行矫正，达到对振镜轴方向修改的目的。

6．可操作振镜轴与普通电机轴，建立连续插补，建立振镜轴与普通轴之间的联动，实现混合插补。

激光器控制注意事项：

激光器的能量控制有一下控制方式：1.模拟量控制能量。2.数字信号组合控制能量。3.通过 PWM 占空比控制能量的输出。

1.模拟量控制能量，模拟量的精度是 10 位的电压大小为 0-10V，数值大小 0-4096 对应控制能量的大小。

2.数字信号组合控制能量，由输出口信号进行组合，能量对应每种组合选择能量。
例：联品激光 mopa 类激光器能量组合如下：

	设置 1	设置 2	设置 3	设置 4	设置 5
针 1	0	0	0	0	1
针 2	0	0	0	0	1
针 3	0	0	0	0	1
针 4	0	0	0	0	1
针 5	0	0	0	1	1
针 6	0	0	1	1	1
针 7	0	1	1	1	1
针 8	1	1	1	1	1
电流	50%	75%	87.5%	93.75%	100%
激光器功率	52%	77%	89%	93%	100%

振镜例程

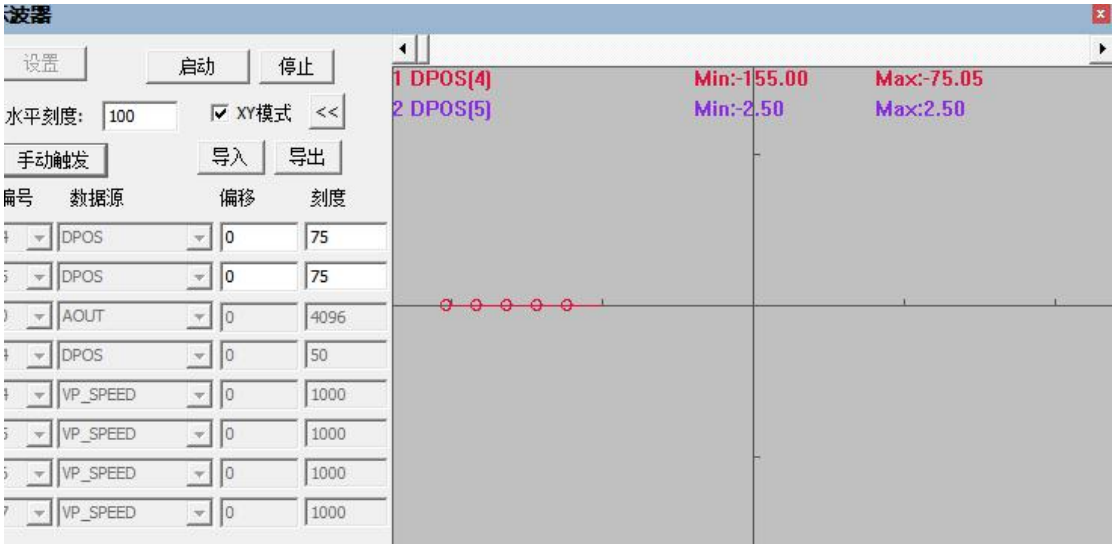
例一：振镜两轴插补运动
说明：振镜两轴插补实现打标 1 行 5 个 5mm 的小线段圆：

```
'设置振镜轴轴号，并配置轴类型
BASE(4,5)
ATYPE=21,21
'设置基本参数
UNITS=300,300
SPEED=5000,5000
ACCEL=SPEED*20,SPEED*20
DECEL=SPEED*20,SPEED*20
MOVEABS(0,0)
FORCE_SPEED=5000
'打开连续插补
MERGE=ON
AXIS_ZSET(4)=3     '开启 MOVE_OP 的精准输出功能
'设置频率
PWM_FREQ(2)=2000
WHILE 1
    IF MODBUS_BIT(0)=ON    THEN
        MODBUS_BIT(0)=OFF
        OP(0,OFF)
```

```
BASE(4,5)
'能量开关
OP(11,ON)
'MO 开关
OP(1,ON)

'打标 5 个小线段圆，轨迹移动数据
FOR j = 0 TO 4
    MOVE(-15, 0)
    MOVE_OP(0,ON)
    MOVE(-0.038, 0.434)
    MOVE(-0.113, 0.421)
    MOVE(-0.184, 0.395)
    MOVE(-0.250, 0.357)
    MOVE(-0.308, 0.308)
    MOVE(-0.357, 0.250)
    MOVE(-0.395, 0.184)
    MOVE(-0.421, 0.113)
    MOVE(-0.434, 0.038)
    MOVE(-0.434, -0.038)
    MOVE(-0.421, -0.113)
    MOVE(-0.395, -0.184)
    MOVE(-0.357, -0.250)
    MOVE(-0.308, -0.308)
    MOVE(-0.250, -0.357)
    MOVE(-0.184, -0.395)
    MOVE(-0.113, -0.421)
    MOVE(-0.038, -0.434)
    MOVE(0.038, -0.434)
    MOVE(0.113, -0.421)
    MOVE(0.184, -0.395)
    MOVE(0.250, -0.357)
    MOVE(0.308, -0.308)
    MOVE(0.357, -0.250)
    MOVE(0.395, -0.184)
    MOVE(0.421, -0.113)
    MOVE(0.434, -0.038)
    MOVE(0.434, 0.038)
    MOVE(0.421, 0.113)
    MOVE(0.395, 0.184)
    MOVE(0.357, 0.250)
    MOVE(0.308, 0.308)
    MOVE(0.250, 0.357)
    MOVE(0.184, 0.395)
```

```
            MOVE(0.113, 0.421)
            MOVE(0.038, 0.434)
            WAIT IDLE
            MOVE_OP(0,OFF)
        NEXT
    ENDIF
WEND
```

运动效果图：



例二：振镜轴与普通轴混合插补运动

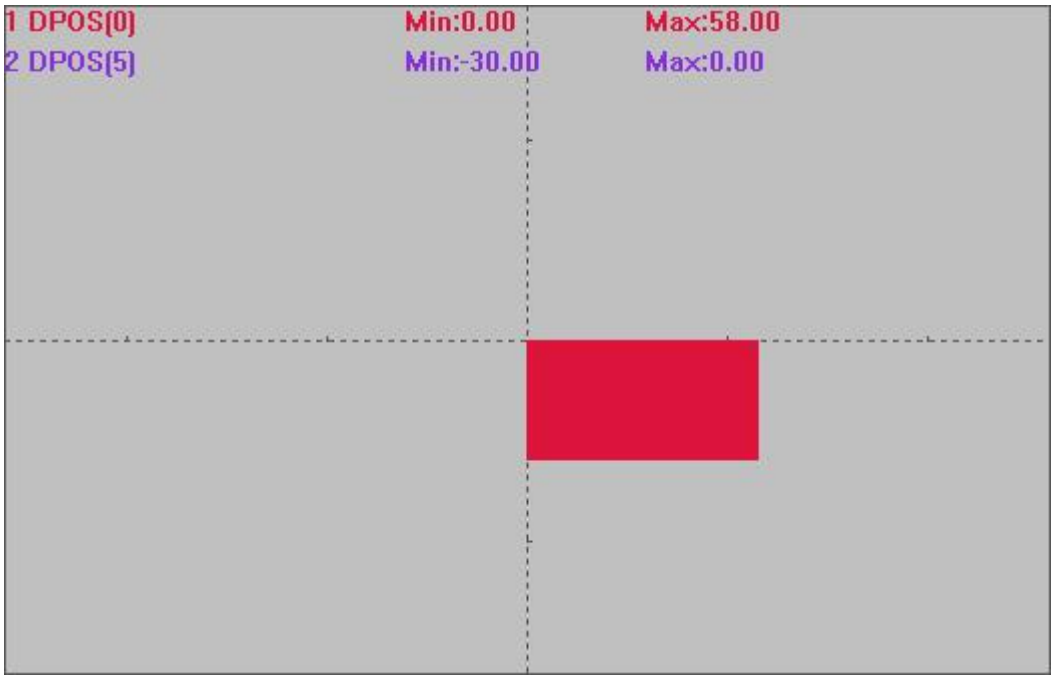
说明：振镜轴与旋转轴建立插补两轴配合运动，进行打标清洗图形。

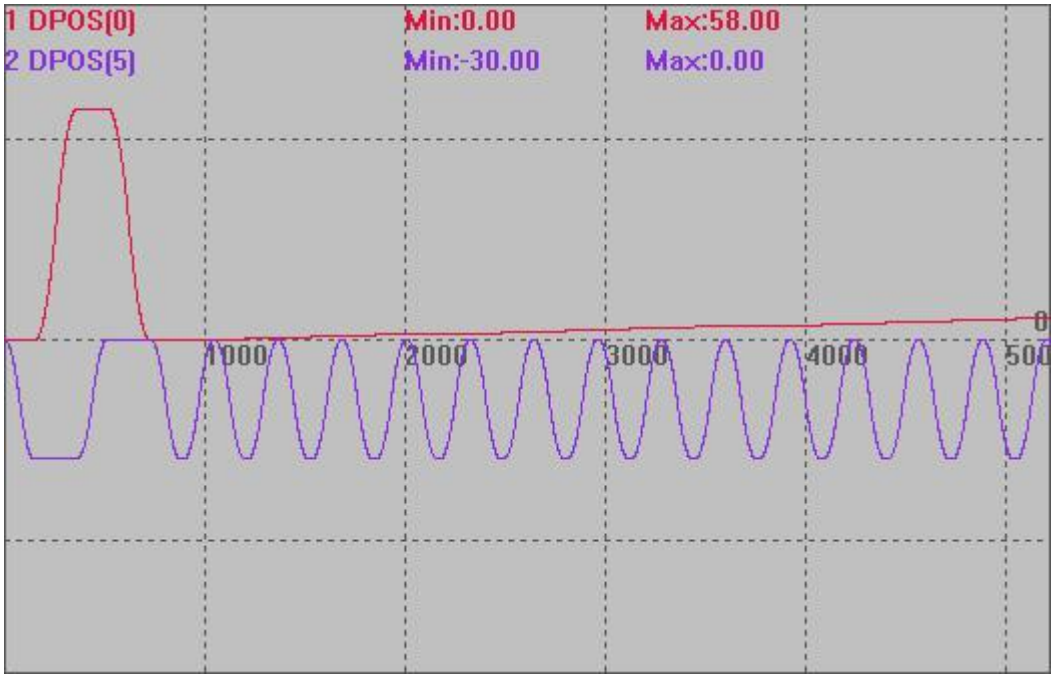
清洗长度 58，清洗宽度 30，要清洗的工件放于旋转轴轴 0 上，轴 5 控制激光运动，Y 轴方向往复运动清洗。

```
BASE(0,5)
ATYPE=7,21
UNITS = 10000/360,10000/18
SPEED=1000,5000
ACCEL=SPEED*5, SPEED*5
DECEL=SPEED*5, SPEED*5
MOVEABS(0,0)
MERGE=ON     '打开连续插补
AXIS_ZSET(0)=3     '开启主轴 MOVE_OP 的精准输出功能
OP(12,ON)        '使能脉冲轴轴 0
PWM_FREQ(2)=2000       '设置 DB25 外控激光频率口的大小
PWM_DUTY(11)= 0.8     '设置能量
PWM_FREQ(11) = 2000

WHILE 1
    LOCAL i          '循环条件
    LOCAL sum        '累计旋转角度
```

```
    IF MODBUS_BIT(0)=ON    THEN
        sum = 0
        MODBUS_BIT(0)=off
        OP(0,OFF)
        OP(11,ON)          '能量开关
        OP(1,ON)           'mo 开关
        WA 100

        MOVE_OP(0,ON)
        TRIGGER

        MOVE(0,-30)
        WAITIDLE
        MOVE(58,0)
        WAITIDLE
        MOVE(0,30)
        WAITIDLE
        MOVE(-58,0)
        WAITIDLE

        '旋转轴旋转一定角度并对旋转轴上面的打标图形进行填充清洗
        FOR i = 0 TO 57.6 STEP 0.4              '填充清洗
            sum = sum + 0.4
            MOVE(0,-30)
            MOVE(0.4,0)
            MOVE(0,30)

        NEXT
        ?"圆筒旋转角度", sum
        MOVE_OP(0,OFF)
        MOVE(-58,0)
    ENDIF
WEND
END
```

运动效果图：





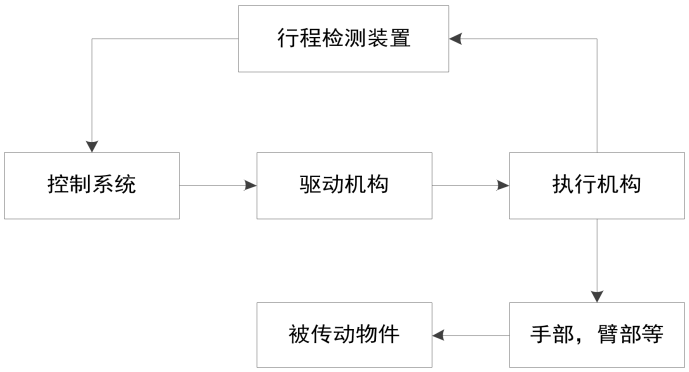
4.10 机械手

4.10.1 机械手组成

机械手是一种模仿人手部的运动，按照给定的程序、轨迹和要求实现自动抓取、搬运的自动化机械。

机械手基本组成部分

机械手主要由执行机构、驱动机构和控制系统三部分组成，各个组成部分关系如下图所示：



1．执行机构

1）手部：直接与工件接触的部分，一般是回转型或平动型，手部的取物方式可分为抓取式、真空吸盘和电磁吸盘等，抓取式可分为内抓式和外抓式，手指一般为两指，也有多指。

2）腕部：连接手部和臂部的部件，并可用来调节被抓取物体的方位，以扩大机械手的动作范围，并使机械手变得更灵巧，适应性更强。腕部有独立的自由度，可回转运动、上下摆动、左右摆动。一般腕部没有回转运动的再增加一个上下摆动即可满足工作需求，有些动作较为简单的机械手，为了简化结构，可以不设腕部，直接用臂部运动驱动手部搬运工件。

3）臂部：是机械手的重要持握部件，作用是支撑腕部和手部（包括工作或夹具），并带动他们做空间运动。臂部可以将手部送到空间运动范围内的任意一点，如果改变手部姿态，则用腕部的自由度加以实现，因此，一般来说，臂部需具有三个自由度，即手臂的伸缩、左右旋转和升降运动才能满足基本要求。

手臂的各种运动通常用驱动机构（例如液压缸和气缸）和各种传动机构来实现，从臂部的受力情况分析，它在工作中既受腕部、手部和工件的负载的影响，而且自身运动较多，受力复杂，因此，它的结构、工作范围、灵活性、定位精度以及抓取物的大小和重量会直接影响机械手的工作性能。

4）行走机构：主要由电动机、齿轮、带轮等部分组成，电机工作带动齿轮、带轮传动，带轮带动轮子运动实现行走动作，目前应用相对较少。

2．驱动机构

驱动机构是工业机械手的重要组成部分，根据动力源的不同，工业机械手的驱动器大致可分为液压、气动、电动、和机械驱动等四类，采用液压机构驱动的机械手具有结构简单、尺寸紧凑、重量轻、控制方便等优点。

3．控制系统

机械手控制有两种方式，点动控制和连续控制，大多数机械手采用可编程控制器进行连续控制。

点动控制为控制机械手从一个点运动到另一个点，对中间的行走轨迹没有要求；连续控制对轨迹有规定，机械手需按照指令的指示走出对应运动轨迹。

4.10.2 机械手相关定义

4.10.2.1 关节轴与虚拟轴

关节轴是指实际机械结构中的旋转关节，在程序中一般显示旋转角度。由于电机与旋转关节会存在减速比，所以设置 units 时要按照实际关节旋转一圈来设，同时 table 中填写结构参数时也要按照旋转关节中心

计算，而不是按照电机轴中心计算。

虚拟轴不是实际存在的，抽象为世界坐标系的 6 个自由度，依次为 X、Y、Z、RX、RY、RZ。可以理解为空间直角坐标系的三个直线轴和三个旋转轴，用来确定机械手末端工作点的加工轨迹与坐标。

4.10.2.2 坐标系

1．关节坐标系

每个轴相对原点位置的绝对角度，包含机械手所有关节，各关节之间相互独立，坐标单位为角度。操作其中一个关节时不影响其他关节。

2．直角坐标系

1）世界坐标系：世界坐标系是被固定在空间上的标准直角坐标系，以机械手的底盘为坐标原点，其位置根据机械手类型确定。虚拟轴操作时就是根据世界坐标系运动，此时各关节会自动解算需要旋转的角度。

2）用户坐标系：用户对每个作业空间进行定义的直角坐标系，它用于位置寄存器的示教与执行，位置补偿命令的执行等，在没有定义的时候，将由世界坐标系来代替该坐标系。

机械手算法的主要目的是将关节坐标系与直角坐标系建立联系。

坐标系转换是指在描述同一个空间时，由原来的坐标系转换为另一个坐标系。机械手使用中，常用于确定工件坐标系。

工件坐标系是固定于工件上的笛卡尔坐标系，工件在坐标系相对于世界坐标系存在转换。每个机械手可以拥有诺干工件坐标系，用来表示不同的工件，或者表示同一工件在不同的位置。

虚拟轴满足 XYZ 三轴的机械手类型支持此功能。

4.10.2.3 正解运动与逆解运动

1．逆解运动

CONNFRAME 对应的运动是逆解运动，此指令作用在关节轴上。此时只能操作虚拟轴，虚拟轴可以在笛卡尔坐标系中做直线、圆弧、空间圆弧等运动，关节轴在 CONNFRAME 的作用下会自动运动到逆解后的位置。

逆解运动模式是指控制器的两种运动模式。机械手保证结束点的位置准确的前提下，会对运动过程的轨迹准确与速度平滑间做个取舍。

逆解运动模式通过设置 CLUTCH_RATE 连接速度实现，控制器默认 CLUTCH_RATE 值为 1000000。

关节轴的 CLUTCH_RATE	运动模式描述
0	平滑模式：此模式下关节轴使用自己的速度加速度做速度规划，高速时轨迹会有变形。适用于对运动轨迹精度要求不高的场合。
非 0	强制模式：此模式下关节轴完全按照虚拟轴的速度加速度进行规划。此模式可准确回到设定位置，但高速运动时会抖动。

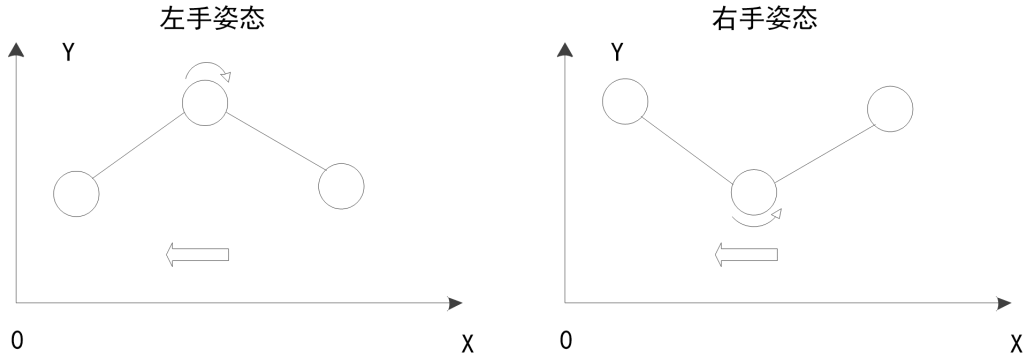
2．正解运动

CONNREFRAME 对应的运动是正解运动，此指令作用在虚拟轴上。此时只能操作关节轴，关节轴也可以做各种运动，但实际运动的轨迹不是直线圆弧，这种模式一般用于手动调整关节位置或上电点位回零。

关节插补运动是机械手在正解模式下，控制末端点走直线、圆弧等插补运动。

姿态

机械手姿态在数学上来说，是同一组虚拟轴数值有多组关节轴的解。即机械手在笛卡尔坐标系中运动到某一坐标点，可以有多种运动轨迹，这些运动轨迹就对应着不同姿态，如下图 SCARA 的两种姿态，在 X 方向运动，关节轴可以有两种运动方式。



奇异点

逆解模式下，机械手运动到某一特定位置时，会失去某个自由度，该位置就叫做奇异点，实际使用过程中应避免运动到奇异点。例如当 SCARA 机械手完全伸直时，此时无法在 X 方向平动，需要操作往 X 负方向运动时，结构无法判断使用哪种姿态运动，此时机械手臂无法运动，出现此状况时，先使用正解模式调整关节轴位置，之后再切换到逆解模式使用。

4.10.3 机械手支持的功能

1．机械手控制

控制机械手末端工作点在世界坐标系下运动。支持多种机械手类型，一个控制器可同时控制多个机械手。机械手有几个电机称为几关节机械手，控制实际机械关节运动的电机轴称为机械手的关节轴，所有的关节轴构成关节坐标系，关节轴在此坐标系中按角度旋转。

2．坐标系旋转

机械手工作点运动的坐标系，参照世界坐标系进行旋转与偏移。可构建用户坐标系。控制机械手末端工作点在世界坐标系下运动，世界坐标系的坐标轴假想为虚拟轴，按距离单位移动。

3．机械参数校正

根据机械手关节示教的坐标和特点自动校正当前的机械手参数。

4．机械手计算

末端工作点坐标与关节轴的坐标之间的计算。

5．机械手运动仿真

ZRobotView 仿真软件，显示机械手的动作。

6．控制器仿真

支持离线仿真，在无控制器情况下可以使用。

机械手建立是通过 CONNREFRAME（正解）指令和 CONNFRAME（逆解）指令设置，CONNREFRAME 时虚拟轴 MTYPE（运动类型）值为 34，CONNFRAME 时关节轴 MTYPE 值为 33，可以通过 MTYPE 来查看特定轴是否位于对应的模式。

关节轴和虚拟轴通过 CONNREFRAME 或 CONNFRAME 指令来指明，只要轴数足够，控制器支持多机械手。

程序可以通过运动指令（所有的运动指令都可以使用）控制关节轴或虚拟轴运动，但同一时刻只能操作虚拟轴或者关节轴，二者无法同时操作。

操作关节轴运动时，虚拟轴需要位于 CONNREFRAME 模式，从而自动指向当前的空间坐标；操作虚拟轴运动时，关节轴需要位于 CONNFRAME 模式，从而自动指向当前的关节轴坐标。

通过 CANCEL 或 RAPIDSTOP 指令可以取消机械手模式。

4.10.4 机械手应用举例

一般来说，调机台的时候选择正解模式；生产加工时选择逆解模式，通过查看轴参数 ATYPE 就可确定当前机械手处于何种模式。机械手运动过程中会出现拐角，需要设置拐角减速，防止机台高速拐角时抖动。

机械手逆解指令语法：

CONNFRAME（frame，tablenum, viraxis0, viraxis1,[…]）

frame：结构类型，1- scara，（如需针对特殊的机械手类型定制，请联系厂家）

tablenum：存储转换参数的 TABLE 位置。依次存放：数据存储在 TABLE 的起始位置，第一个关节轴长度，第二个关节轴长度，第一个关节轴一圈脉冲数，第二个关节轴一圈脉冲数

viraxis0：虚拟坐标系第一个轴

viraxis1：虚拟坐标系第二个轴

[…] ：虚拟坐标系第 N 个轴，此运动轴可以是实际轴

编程参考步骤：

1．参数定义：定义关节长度和各个轴之间的距离，设置各个轴的脉冲当量。

2．关节轴设置：选择关节轴轴号，设置轴类型、脉冲当量（关节轴脉冲当量需要转换成角度）、速度参数、设置逆解运动模式（CLUTCH_RATE）、拐角减速等。

3．虚拟轴设置：选择虚拟轴轴号，设置轴类型（ATYPE=0）、脉冲当量。

4．将机械手参数存储在 TABLE 里。

5．建立机械手连接。

六自由度机械手例程：

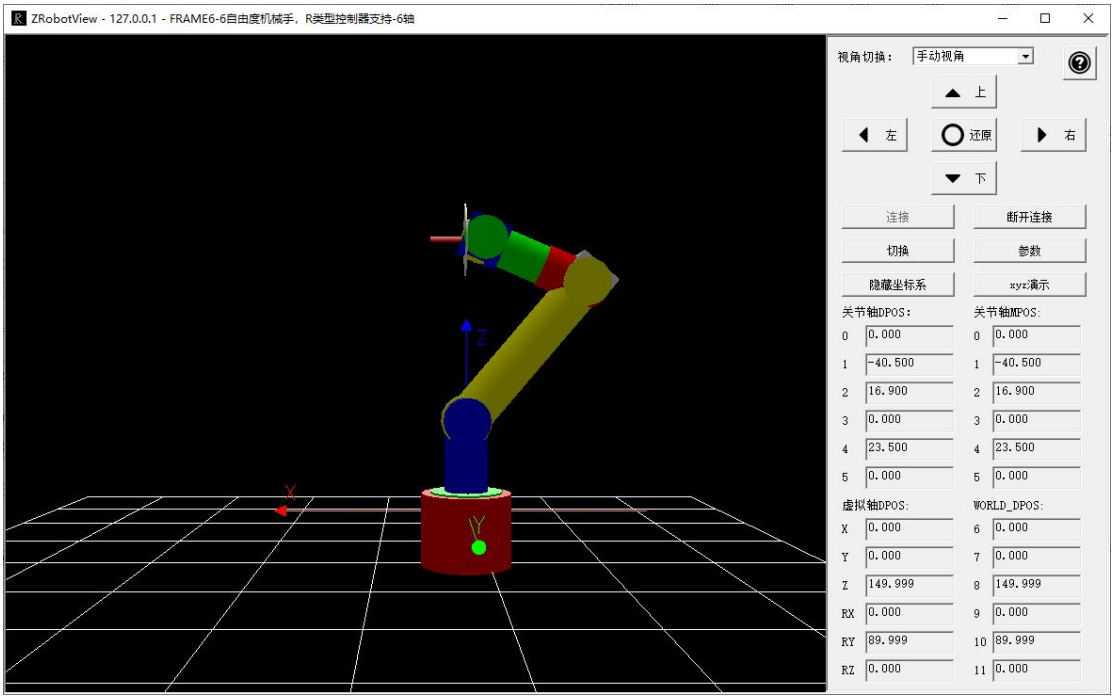
```
''''''''''''''''''''参数定义''''''''''''''''''''
DIM LargeZ              '基座的垂直高度
DIM L1                  '1 轴到 2 轴的 X 偏移；转盘中心到大摆臂中心的偏移。
DIM L2                  '大摆臂长度
DIM L3                  '3 轴中心到 4 轴中心距离
DIM L4                  '4 轴到 5 轴的距离。
DIM D5                  '5 转一圈，6 转动的圈数，0 表示不关联。
DIM PulesVROneCircle    '虚拟姿态轴一圈脉冲数
DIM SmallZ              '末端到 5 轴的垂直距离
DIM SmallX, SmallY              '末端到转盘中心的 XY 偏移
DIM InitRx, InitRy, InitRz    '初始的姿态 ，（0，0，0 ）指向 z 正向
''''''''''''''''''''参数赋值''''''''''''''''''''
LargeZ=50
L1=0
L2=100
L3=0
L4=60
D5=0
SmallZ=10
SmallX=0
SmallY=0
InitRx=0
InitRy=0
InitRz=0
```

```
PulesVROneCircle=360*1000
DIM u_m1          '电机 1 一圈脉冲数
DIM u_m2          '电机 2 一圈脉冲数
DIM u_m3          '电机 3 一圈脉冲数
DIM u_m4          '电机 4 一圈脉冲数
DIM u_m5          '电机 5 一圈脉冲数
DIM u_m6          '电机 6 一圈脉冲数
u_m1=3600
u_m2=3600
u_m3=3600
u_m4=3600
u_m5=3600
u_m6=3600
DIM i_1           '关节 1 传动比
DIM i_2           '关节 2 传动比
DIM i_3           '关节 3 传动比
DIM i_4                  '关节 4 传动比
DIM i_5           '关节 5 传动比
DIM i_6           '关节 6 传动比
i_1=1
i_2=1
i_3=1
i_4=1
i_5=1
i_6=1
DIM u_j1     '关节 1 实际一圈脉冲数
DIM u_j2     '关节 2 实际一圈脉冲数
DIM u_j3     '关节 3 实际一圈脉冲数
DIM u_j4     '关节 4 实际一圈脉冲数
DIM u_j5     '关节 5 实际一圈脉冲数
DIM u_j6     '关节 6 实际一圈脉冲数
u_j1=u_m1*i_1
u_j2=u_m2*i_2
u_j3=u_m3*i_3
u_j4=u_m4*i_4
u_j5=u_m5*i_5
u_j6=u_m6*i_6
""""""""""""""""关节轴设置""""""""""""""""
BASE(0,1,2,3,4,5)                    '选择关节轴号 0、1、2、3、4、5
ATYPE=1,1,1,1,1,1                    '轴类型设为脉冲轴
UNITS = u_j1/360,u_j2/360,u_j3/360,u_j4/360,u_j5/360 ,u_j6/360 '把 units 设成每°脉冲数
DPOS=0,0,0,0,0,0                     '设置关节轴的位置，此处要根据实际情况来修改
SPEED=100,100,100,100,100,100        '速度参数设置
ACCEL=1000,1000,1000,1000,1000,1000
```

```
DECEL=1000,1000,1000,1000,1000,1000
CLUTCH_RATE=0,0,0,0,0,0      '使用关节轴的速度和加速度限制
MERGE=ON                     '开启连续插补
CORNER_MODE = 2              '启动拐角减速
DECEL_ANGLE = 15 * (PI/180)  '开始减速的角度 15 度
STOP_ANGLE = 45 * (PI/180)   '降到最低速度的角度 45 度
'''''''''''''''''''虚拟轴设置'''''''''''''''''''
BASE(6,7,8,9,10,11)
ATYPE=0,0,0,0,0,0                    '设置为虚拟轴
TABLE(0,LargeZ,L1,L2,L3,L4,D5,u_j1,u_j2,u_j3,u_j4,u_j5,u_j6,PulesVROneCircle,SmallX,SmallY,SmallZ,
InitRx,InitRy,InitRz)        '根据手册说明填写参数
UNITS=1000,1000,1000,1000,1000,1000   '运动精度，要提前设置，中途不能变化
'''''''''''''''''''建立机械手连接'''''''''''''''''''
WHILE 1
    IF   SCAN_EVENT(IN(0))>0 THEN   '输入 0 上升沿触发
        BASE(0,1,2,3,4,5)                '选择关节轴号
        CONNFRAME(6,0,6,7,8,9,10,11)          '启动逆解连接。
        WAIT LOADED              '等待运动加载，此时会自动调整虚拟轴的位置。
        ?"逆解模式"
    ELSEIF   SCAN_EVENT(IN(0))<0 THEN '输入 0 下降沿触发
        BASE(6,7,8,9,10,11)              '选择虚拟轴号
        CONNREFRAME(6,0,0,1,2,3,4,5)          '启动正解连接。
        WAIT LOADED              '等待运动加载。
        ?"正解模式"
    ENDIF
WEND
```

可启用 ZRobotView 软件仿真，使用方法：将此段程序下载到控制器之后，打开 ZRobotView 软件，点击右方“连接”按钮，选择与控制器的连接方式后确认连接（网口通讯选择与控制器同一 IP 网段，串口通讯选择与控制器相同串口号、波特率等），此时将会自动建立模拟机械手，仿真运动，还可以使用 ZDevelop 软件的“手动运动”功能，在该界面通过手动改变轴的坐标模拟机械手运动。

六自由度机械手 ZRobotView 软件仿真图：



第五章 轴相关

在运动控制系统中，将运动控制的对象称为“轴”，运动控制系统中的一个电机控制的运动平台称为一个运动轴，每个运动轴只有一个自由度，可以做直线运动或旋转运动，轴的分类如下：

轴种类	描述
电机轴	使用控制器 EtherCAT 总线或 RTEX 总线接口和 EtherCAT 或 RTEX 伺服驱动器连接，分配为实际伺服驱动器加以使用，将 1 台伺服电机作为 1 个轴使用。 使用控制器脉冲轴接口时，可将高速差分脉冲输出单元或高速单端脉冲输出单元分配给该轴。
虚拟轴	运动控制器内的虚拟轴，不使用实际伺服驱动器，或作为同步控制的虚拟主轴和作为机械手算法中的笛卡尔坐标轴使用。
编码器轴	使用控制器本地编码器轴接口，分配为实际编码器输入加以使用。

编码器简介

编码器（encoder）是将信号或数据进行编制，转换为可用以通讯、传输和存储信号形式的设备，编码器可以把角位移或直线位移转换成电信号，

编码器为传感器的一种，主要用来检测机械运动的速度、位置、角度、距离等参数，伺服电机均配备编码器，伺服驱动器将编码器测得的数据仿真输出给控制器，形成反馈以便更精确控制伺服电机旋转的位置、速度等。

编码器按工作原理可分为绝对值型和增量型，按检测原理可细分为光学式、磁式、感应式和电容式。

增量式编码器是将位移转换成周期性的电信号，再把这个电信号转变成计数脉冲，用脉冲的个数表示位移的大小。绝对式编码器的每一个位置对应一个确定的数字码，因此它的显示值只与测量的起始和终止位置有关，而与测量的中间过程无关。

增量式编码器轴旋转时，有相应的脉冲输出，其旋转方向的判别和脉冲数量的增减借助后部的判向电路和计数器来实现。其计数起点任意设定，可实现多圈无限累加和测量。还可以把每转发出一个脉冲的 Z 信号，作为参考机械零位。编码器轴转一圈会输出固定的脉冲，脉冲数由编码器光栅的线数决定。需要提高分辨率时，可利用 90 度相位差的 A、B 两路信号对原脉冲数进行倍频，或者更换分辨率更高的编码器。

绝对式编码器有与位置相对应的代码输出，通常为二进制码或 BCD 码。从代码数大小的变化可以判别正/反方向和位移所处的位置，绝对零位代码还可以用于停电位置记忆。绝对式编码器单圈的测量范围常规为 0-360 度。

二者区别：增量型编码器的位置是从零位标记开始计算脉冲数量的，开始计数前需要先寻找零位标记，而绝对型编码器的位置是由输出代码的读数直接确定的，在一圈里，每个位置的输出代码的读数是唯一的。因此，当电源断开时，绝对型编码器并不与实际的位置分离，如果电源再次接通，那么位置读数仍是当前的。

虚拟轴不存在物理性编码器，因此，虚拟轴与实际电机轴存在如下不同：

1．作为常时伺服 ON 状态进行处理；

2．当前反馈位置＝指令当前位置；

3．反馈当前速度＝指令当前速度；

编码器轴与电机轴/虚拟轴存在如下不同：

1．没有指令位置 DPOS，仅有反馈位置 MPOS；

2．无法使用动作类的运动控制指令；

3．单轴位置控制时指定为编码器轴/虚拟编码器轴会导致无法运动。

各种类型的轴可使用位置的种类如下所示。

轴种类	位置的种类	
	指令位置 DPOS	反馈位置 MPOS
电机轴	可以使用	可以使用
虚拟轴	可以使用	MPOS=DPOS
编码器轴	DPOS=MPOS	可以使用

5.1 电机轴与编码器轴

1．轴的基础知识

电机轴：主动运行，根据控制器发出的脉冲来运动，一般分为伺服电机和步进电机。发送的脉冲数根据 DPOS*UNITS 确定。

编码器轴：被动运行，跟随电机转动，产生脉冲，反馈给驱动器（伺服）/控制器（步进），接收的脉冲数（UNITS）得到 MPOS 值，DPOS 值默认跟随 MPOS。控制器接收的脉冲数根据 ENCODER 指令确定。

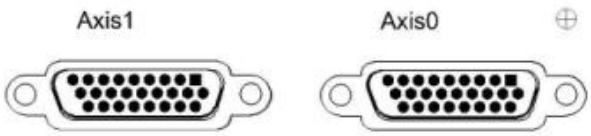
轴类型：一般电机轴 ATYPE=1，编码器轴 ATYPE=3/6，具体查看 ATYPE 指令。

轴接线：电机和编码器接线查看附录“控制器与驱动器接线参考”。

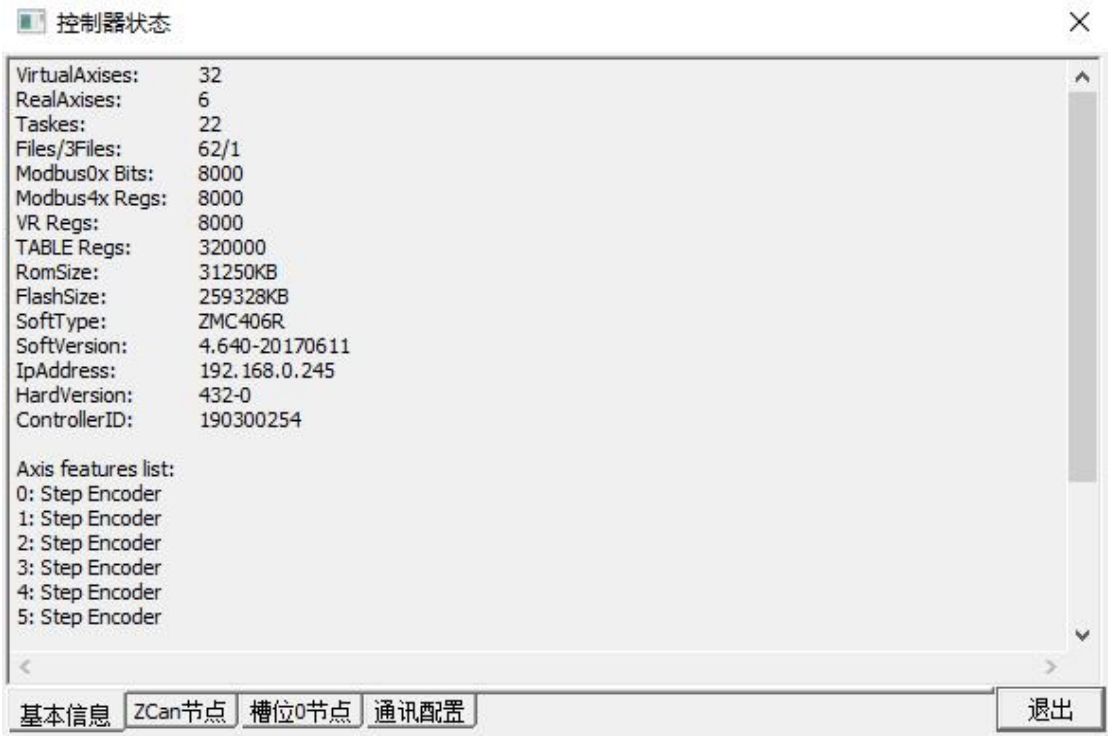
2．轴号说明

1）脉冲控制器轴号

电机轴号是与其连线的 DB 轴端子接口的编号，印在外壳上，形如 Axis0，Axis1…（没有 DB 接口的请查看相应控制器硬件手册确定）



以 ZMC432 控制器状态为例，DB 轴端子接口的编码器轴号在 ZDevelop 软件中可查看“控制器状态”确定（或者硬件手册），如下图，电机轴号是轴 0，对应的编码器轴号是轴 6，电机轴 1 对应编码器轴 7，依次往下；假设电机脉冲和编码器都连接在 Axis0 接口，那么电机轴号 0，编码器轴号映射到轴 6；假设电机连接在 Axis1 接口，编码器连接在 Axis2 接口，那么电机轴号 1，编码器轴号 8。



2）总线控制器轴号

总线控制器包括总线轴和脉冲轴，总线轴轴号通过 AXIS_ADDRESS 指令来映射总线连接的驱动器设备的轴号；脉冲轴轴号和脉冲控制器轴号一致。

3．修改电机运动方向的方法

脉冲轴：1）INVERT_STEP 指令选择脉冲模式

2）STEP_RATIO 指令将分母设为负值

3）驱动器修改转动方向

总线轴：1）STEP_RATIO 指令将分母设为负值

2）驱动器修改转动方向

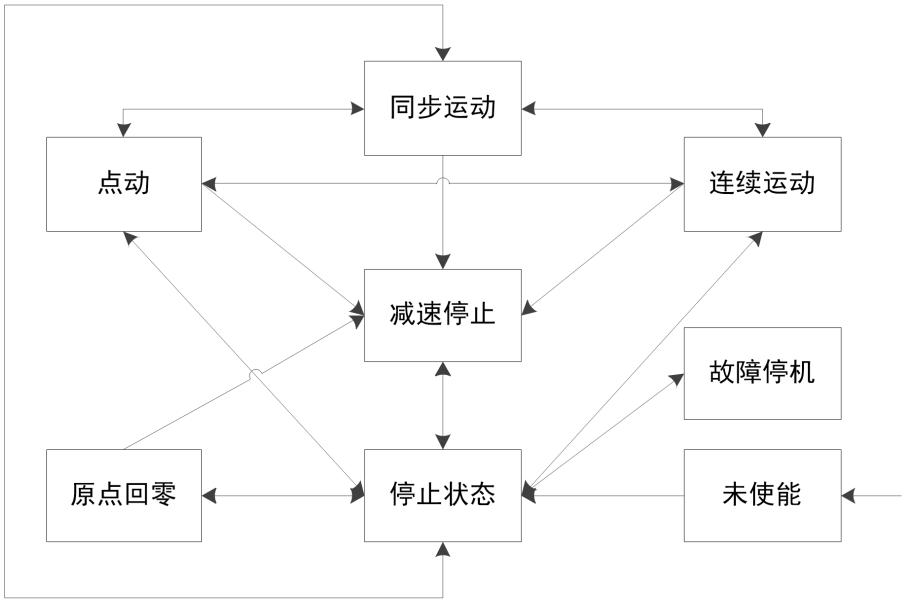
5.2 轴状态

可通过 AXISSTATUS 指令查看，根据每一位状态获取轴当前的状态。值对应的位状态如下表所示。

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD 速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h

11	CANCEL 执行中	2048	800h
12	脉冲频率超过 MAX_SPEED 限制需要修改降速或修改 MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

轴的基本状态转换图：



状态名称	定义
未使能	轴处于伺服 OFF 的状态，轴无法运动。 脉冲轴使用 OP 使能，总线轴使用 AXIS_ENABLE 与 WDOG 使能。
故障停机	轴处于伺服 OFF，停止中，轴发生异常停止。 可通过 AXIS_STOPREASON 查看故障停机原因。
停止状态	轴处于伺服 ON，停止的状态。
减速停止	通过强制停止指令或急停按钮使轴减速停止的状态，轴处于伺服 ON 状态。包括运动完成停止、指令取消运动和出现故障停止这几种情况。停止过程中无法执行其他轴运动指令。
点动	正在执行指定了目标位置的定位的状态，包括等待定位完成的状态和定位动作正在执行的状态。VMOVE、FORWORD、REVERSE 等指令控制轴的点动。
同步运动	通过同步控制指令同步控制轴的状态，包括同步控制指令切换后的同步等待状态。CONNECT、CAM、CAMBOX、MOVELINK、MOVESLINK 等指令控制轴同步运动。
连续运动	正在执行未指定目标位置的连续动作的状态，速度控制或转矩控制时变为此状态。MERGE=ON 开启。
原点回零	通过找原点指令（DATUM）搜索原点，确定回零方式并回到原点。

典型运动状态流程：

1．上电使能初始化操作，操作完成后等待或应用程序发出指令后开始运动。

2．调整机床初始位置，各轴回零。

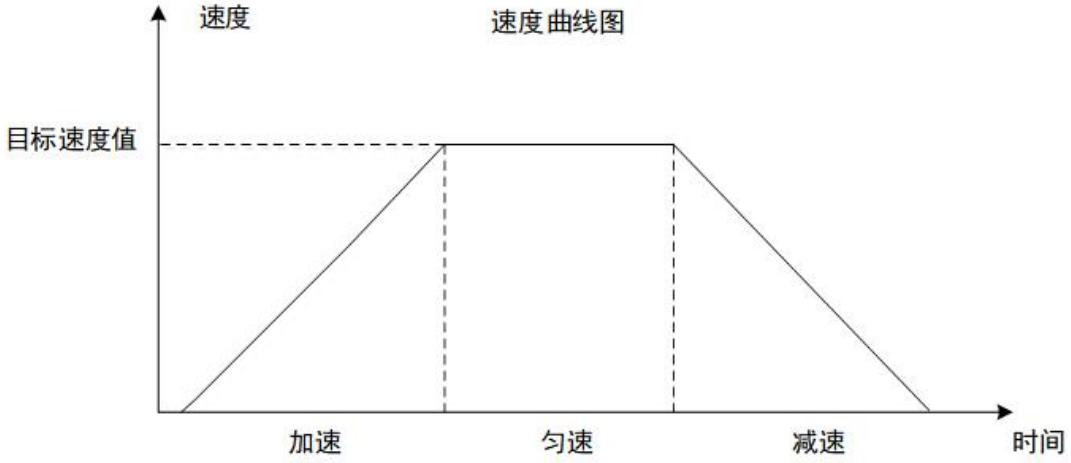
3．回零完成，准备接收运动指令开始执行，有三种典型的运动方式：点动、同步运动、连续运动。不同运动方式可由对应运动指令直接切换，可以不进行停止操作。

4．执行完运动或收到停止命令后，轴减速停止，完全停下后处于停止状态。

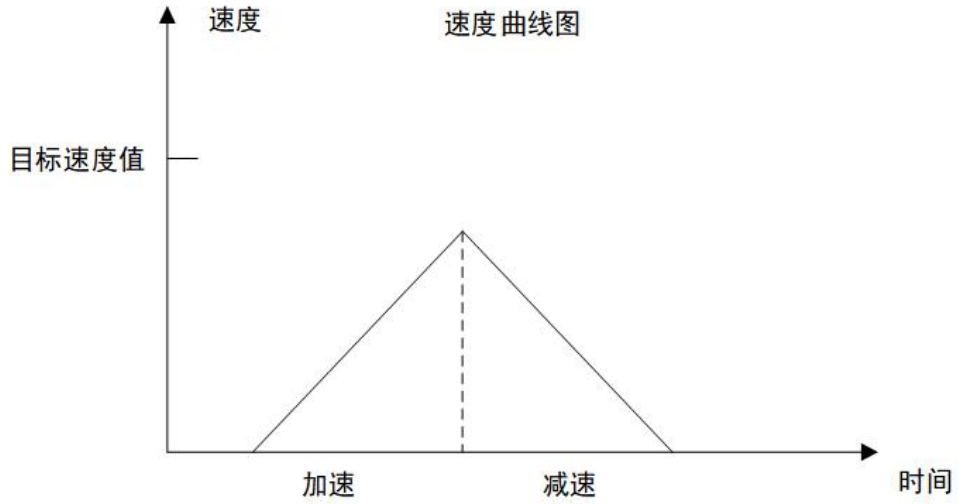
轴回零过程中遇到限位或故障情形可能会直接停止。

5.3 轴速度

速度曲线一般分为三个阶段：加速阶段、匀速阶段、减速阶段，如下图所示。



当位移较短时，可能不存在匀速阶段，仅有加减速阶段，如下图所示。



5.3.1 两种曲线类型

1．梯形图曲线

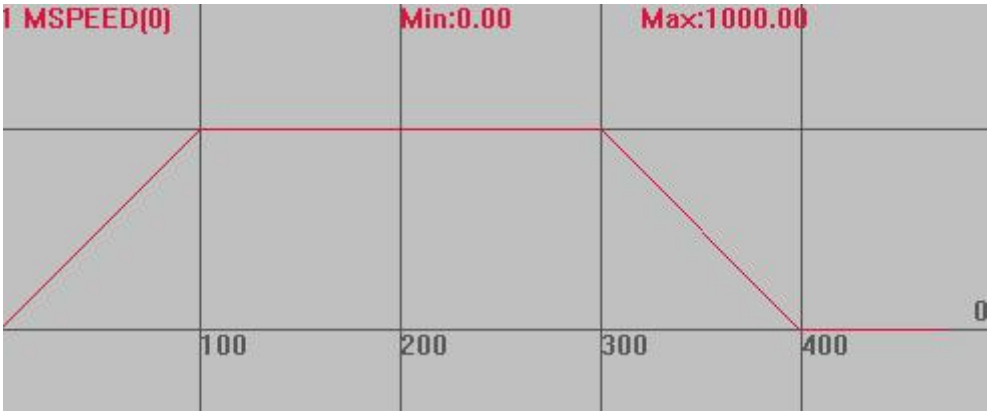
不设置 SRAMP（令 SRAMP 等于 0），速度曲线为梯形曲线，在此速度规划方式下，速度曲线按梯形曲线变化。保持速度、加减速度等参数值不变。

速度到达设定数值后匀速运动，若只设置加速度，减速度为 0 时，减速度会自动等于加速度的值。一般在运动前就设置好相应的加减速，运动过程中不要修改，运动中调整会导致运动轨迹变化。

例程如下：

```
RAPIDSTOP(2)
WAIT    IDLE
BASE(0)
MPOS=0
DPOS =0
UNITS = 100
SPEED = 1000
ACCEL = 10000
DECEL = 10000
SRAMP=0
TRIGGER
MOVE(300)
```

此时得到速度曲线如下：此时加减速过程较快，速度变化对机床冲击较大。



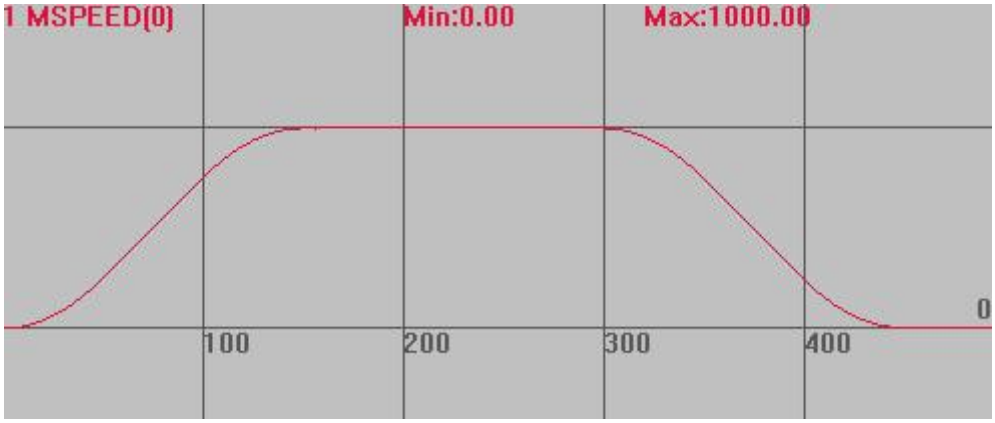
2．S 型曲线

通过设置 SRAMP 的值来设定合适的加减速变化率，使得速度曲线平滑，在机械启停或加减速时减少抖动。SRAMP 值的范围在 0-250 毫秒之间，设置后加减速过程会变长相应的时间，时间越长速度曲线越平滑。设置时间若超过 250 毫秒，按照 250 毫秒进行平滑。

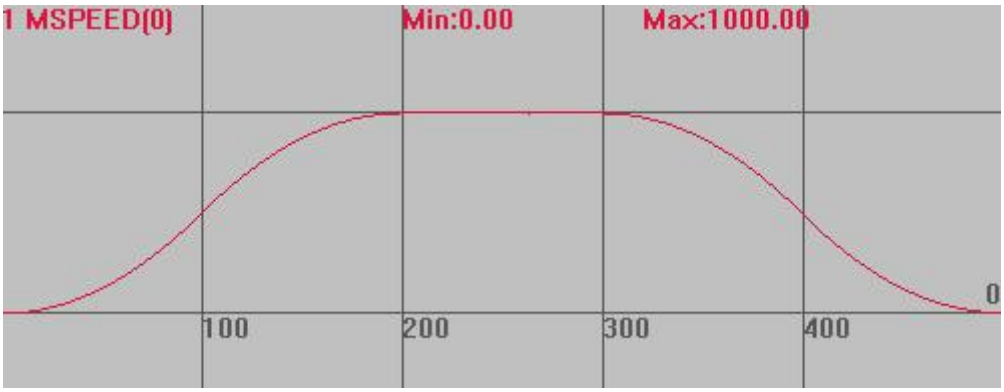
例程：

```
RAPIDSTOP(2)
WAIT    IDLE
BASE(0)
DPOS = 0
MPOS = 0
UNITS = 100
SPEED = 1000
ACCEL = 10000
DECEL = 10000
SRAMP=50
TRIGGER
MOVE(300)
```

当 SRAMP=50 时，得到 S 型曲线如下:



当 SRAMP=100 时，得到 s 型曲线如下：



5.3.2 起始速度、结束速度、SP 速度

起始速度 STARTMOVE_SPEED：自定义速度的 SP 运动的开始速度。

结束速度 ENDMOVE_SPEED：自定义速度的 SP 运动的结束速度。

强制速度 FORCE_SPEED：自定义速度的 SP 运动的强制速度。

以上三个参数只有使用了带 SP 的运动指令才生效，参数都被带入运动缓冲。

不使用时请将 STARMOVE_SPEED 和 ENDMOVE_SPEED 设置较大值，否则下一段运动指令还会沿用此参数。

用下面两段类似程序来具体化描述。

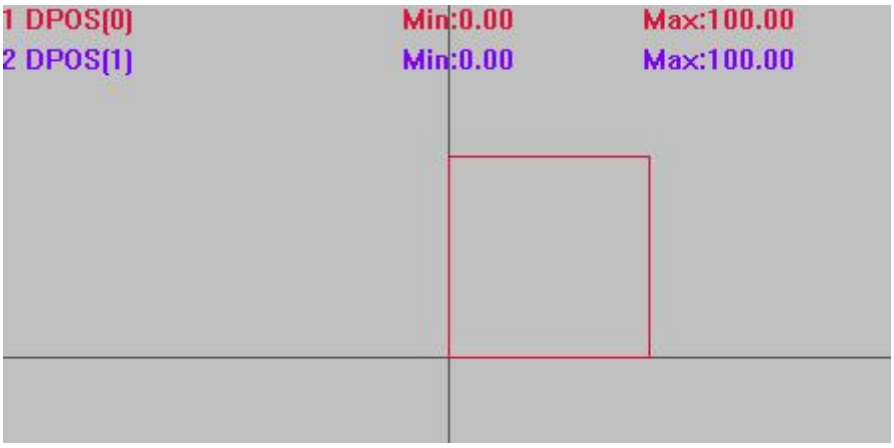
1．不设置启动速度、结束速度和 SP 速度

程序如下，XY 轴插补运动，走一个方框：

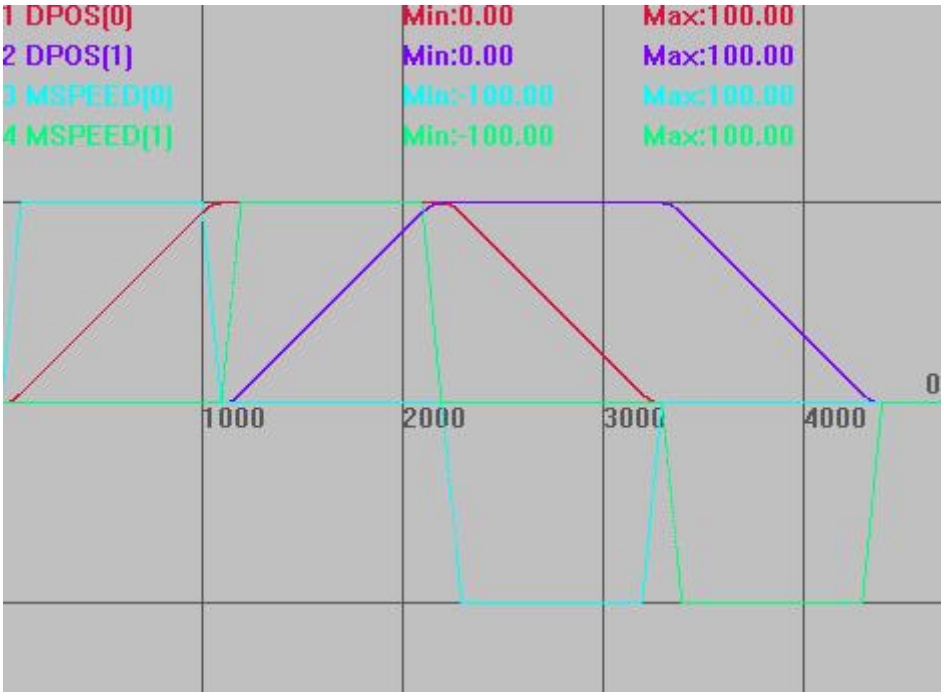
```
RAPIDSTOP(2)
WAIT IDLE
BASE(0,1)     '选择 XY
DPOS = 0,0
MPOS = 0,0
ATYPE=1,1'脉冲方式步进或伺服
UNITS = 100,100   '脉冲当量
SPEED = 100,100
ACCEL =1000,1000
DECEL = 1000,1000
```

```
MERGE = ON '启动连续插补
TRIGGER
CORNER_MODE = 2'启动拐角减速
DECEL_ANGLE = 15 * (PI/180)'开始减速的角度 15 度
STOP_ANGLE = 45 * (PI/180)'降到最低速度的角度 45 度
MOVE(100,0)
MOVE(0,100)
MOVE(-100,0)
MOVE(0,-100)
WAIT IDLE   '等待运动停止
DELAY(100)  '延时
END
```

XY 模式位移如下：



轴运动轨迹与速度曲线：



2. 设置启动速度、停止速度和 SP 速度

例程如下，走一个方框，每一段的速度和停止速度都不一样：

```
RAPIDSTOP(2)
WAIT IDLE
BASE(0,1)     '选择 XY
TRIGGER
DPOS = 0,0
MPOS = 0,0
ATYPE=1,1'脉冲方式步进或伺服
UNITS = 100,100'脉冲当量
SPEED = 100,100
ACCEL = 200,200
DECEL = 200,200
MERGE = ON '启动连续插补
'第一段
FORCE_SPEED = 50'第一段速度 50
ENDMOVE_SPEED = 10'第一段结束的速度 10，
MOVESP(100,0)
'第二段
FORCE_SPEED = 60'第二段速度 60
ENDMOVE_SPEED = 15
STARTMOVE_SPEED = 15 '第二段的起始速度 15
MOVESP(0,100)
'第三段
FORCE_SPEED = 80'第三段速度 80
ENDMOVE_SPEED = 20
STARTMOVE_SPEED = 20 '第三段的起始速度 20
MOVESP(-100,0)
'第四段
FORCE_SPEED = 120'第四段速度 120
ENDMOVE_SPEED = 30
STARTMOVE_SPEED = 30 '第四段的起始速度 30
MOVESP(0,-100)
WAIT IDLE   '等待运动停止
DELAY(100) '延时
END
```

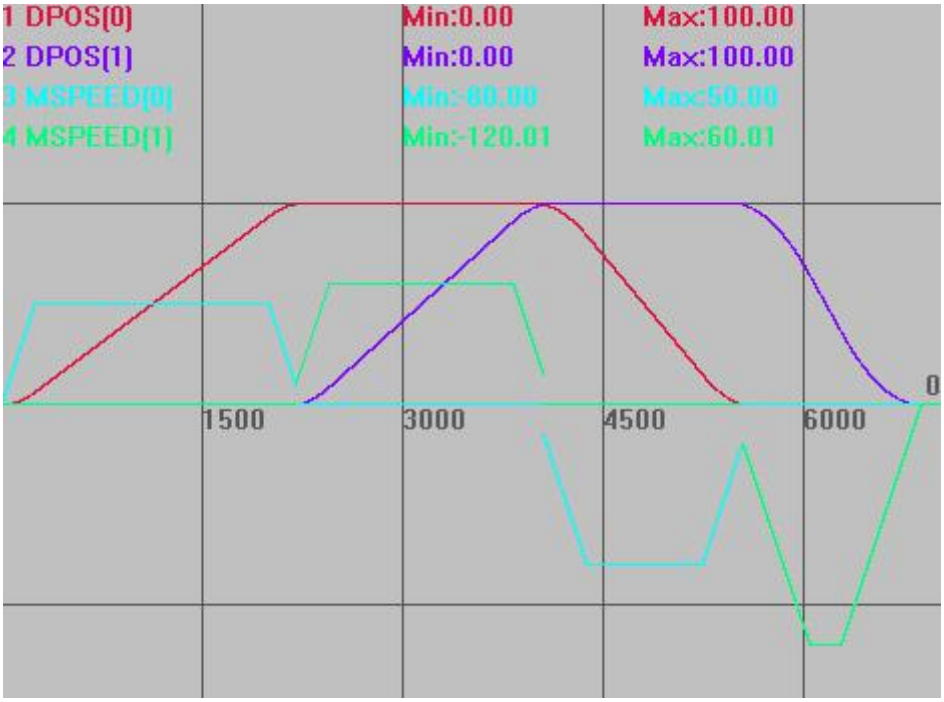
设置 FORCE_SPEED 后，MOVESP 运动指令的速度参照最近的 FORCE_SPEED 速度，MOVE 指令的速度不受 FORCE_SPEED 影响，仍按 SPEED 速度运动。

轴刚启动时，速度都是从 0 开始，此时设置起始速度 STARTMOVE_SPEED 无效。

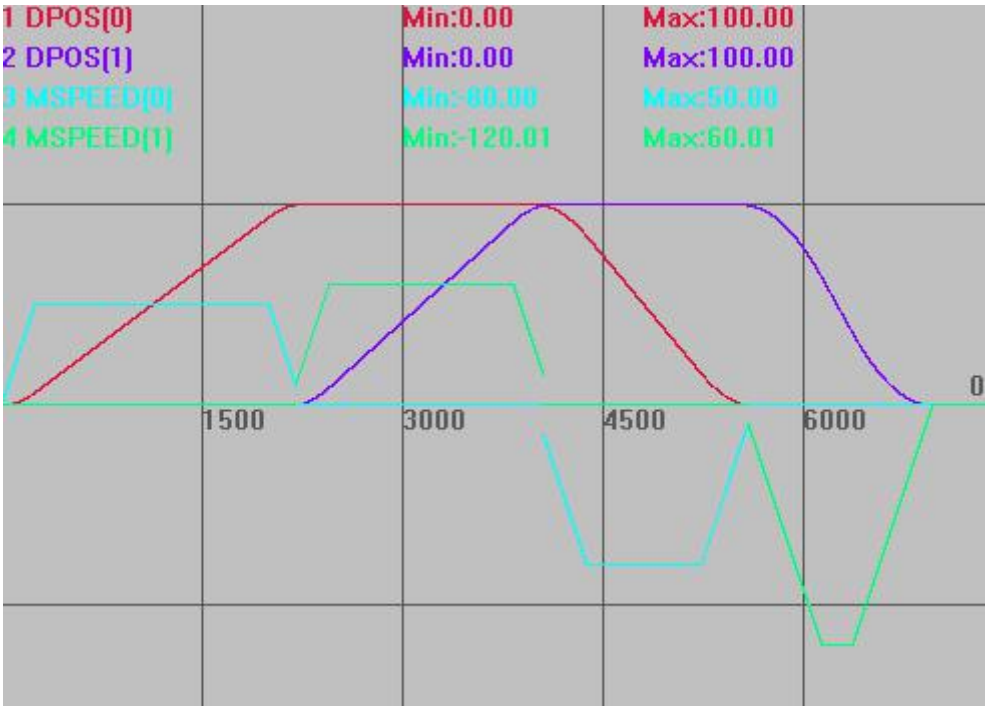
连续插补时，起始速度会受到上一段结束速度的影响。第二段起始速度 15 因为高于第一段的结束速度 10，因此实际的起始速度为 10；第三段起始速度为 20 因为高于第二段的结束速度 15，因此实际的起始速度为 15； 第四段起始速度为 30，因为高于第二段的结束速度 20，因此实际的起始速度为 20。

运动第四段之后，轴便停止了，此时终止速度 ENDMOVE_SPEED 设置无效。

速度、位置曲线如下，水平刻度为 1500，竖直刻度均为 100。



其他条件不变，仅将第四段的起始速度由 30 改为 10，此时减到第三段结束速度 20 的时候还要继续按同样的减速度减到 10，所以得出如下曲线：

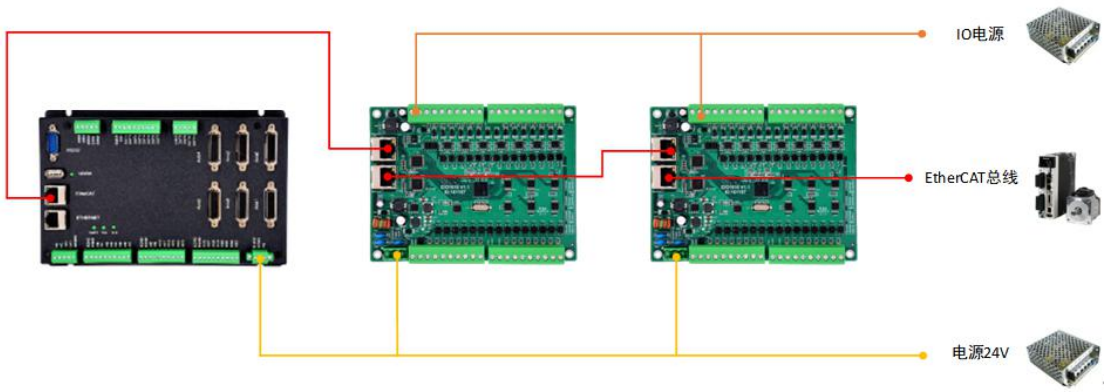


5.4 模块扩展

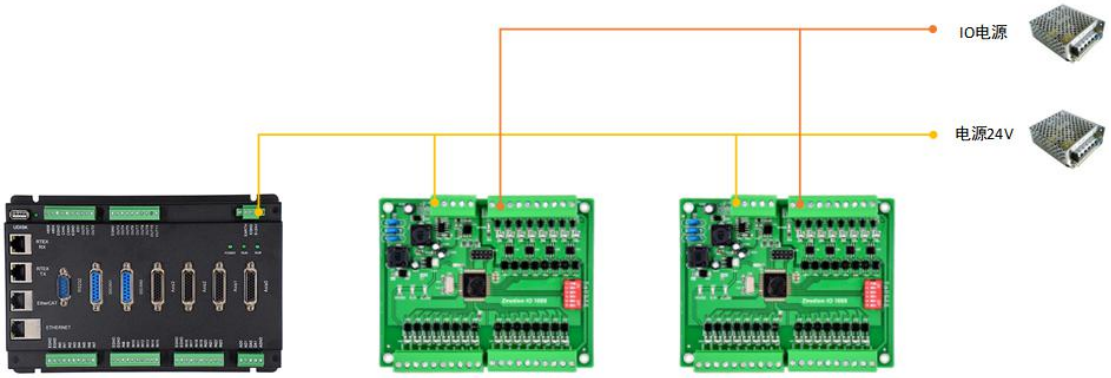
根据实际应用需求，扩展模块可以用来扩展脉冲轴、IO 点数以及模拟量输入输出。扩展模块分三种：EtherCAT 扩展模块、ZCAN 扩展模块、ZMIO300 扩展模块，不同的模块作用有所差异，各模块性能参数参见选型指南“扩展模块”章节。

扩展方法参见下图：

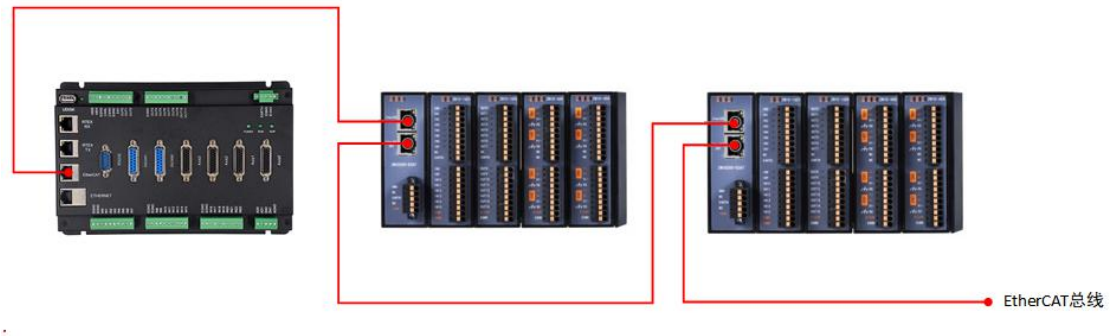
1．EtherCAT 扩展模块应用框图



2．ZCAN 扩展模块应用框图



3．ZMIO300 扩展模块应用框图



5.4.1 IO 扩展

当控制器板载 IO 等资源不够的时候，需要增加扩展模块，控制器支持同时连接多个扩展模块，一般采用 CAN 总线（ZCAN 协议）或 EtherCAT 总线进行扩展。

1．CAN 总线扩展方式：

CAN 总线扩展一般连接 ZCAN 扩展模块，ZCAN 总线扩展模块的 ZIO 扩展板通过拨码来区分，控制器上程序只需通过 IN、OP、AIN、AOUT 等指令就可以访问到扩展模块上的资源。详情可参考“CAN 总线通讯”。

判断模块是双电源供电还是单电源供电，区在在于 IO 接口是否要单独供电，当 IO 接口需要单独供电时，模块供电采用双电源，ZIO 数字量扩展板采用双电源供电。IO 接口无需供电时，模块供电采用一个电源即可，ZAIO 模拟量扩展模块采用单电源供电。

控制器根据型号的不同，供电方式有所差异，ZMC0~2 系列控制器采用双电源供电，3 系列及以上采用单电源供电，详情查看控制器硬件手册确认。

请把内部电源 24V 和外部数字量 IO 电源 24V 分开供电，特别是现场电磁干扰严重的情况下，必须采用两个 24V 电源，或是一个能提供两路隔离 24V 输出的电源。当通过串口连接触摸屏时，触摸屏的电源使用内部电源 24V 来提供。

ECI/ZMC 部分控制器采用单电源供电时扩展板两路电源共一路即可。

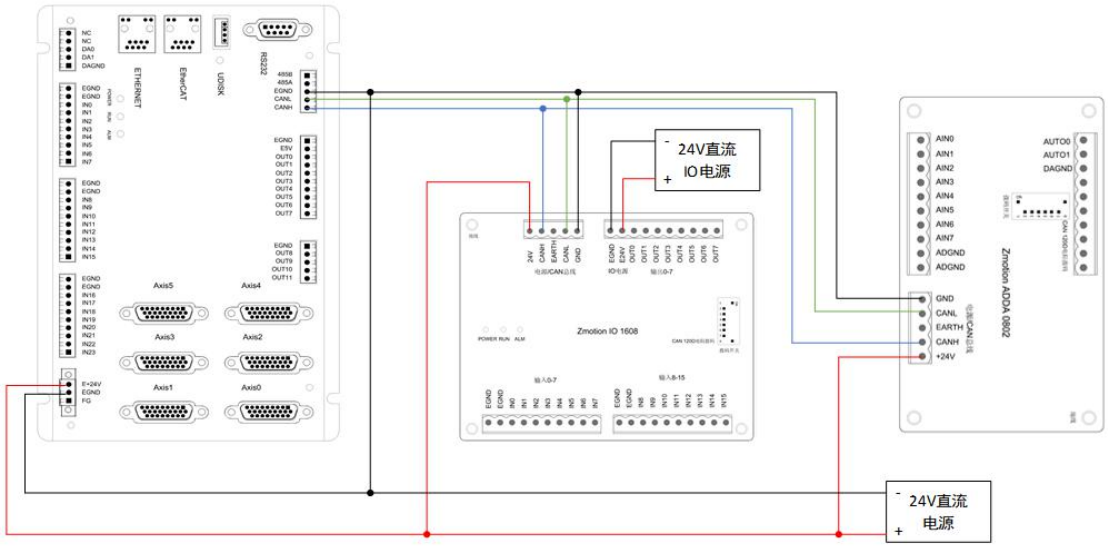
⚠CAN 总线通讯双方必须保证对应 GND 连上或是控制器主电源和扩展模块主电源用同一个电源。控制器和扩展模块用不同电源供电时：控制器主控电源地要连接扩展模块电源的 GND，否则可能烧坏 CAN。

⚠CAN 总线上链接多个扩展模块时，需要在最两边扩展模块的 CANL 与 CANH 端并接一个 120 欧姆的电阻。V1.3 以上硬件版本带 8 位拨码开关的扩展模块，板上面集成了 120 欧姆电阻在 CANL 和 CANH 之间，由拨码 8 控制，拨 ON 时电阻接通，只需要把最末端扩展模块的拨码 8 拨 ON，不需要另外在端子外部接 120 欧电阻。

电源、CAN 总线接线参考：

1）单电源供电的控制器和双电源供电的 ZIO 数字量扩展模块连接使用接线参考：

图中控制器为 ZMC432，ZMC432 的 IO 不需要供电，故采用单电源即可，ZIO1608 数字量扩展模块 IO 需要供电，采用双电源，ZAIO0802 模拟量扩展板采用也单电源供电。此处 CANH 和 CANL 两端没有外接电阻，将扩展板拨码开关的 8 位拨为 ON。



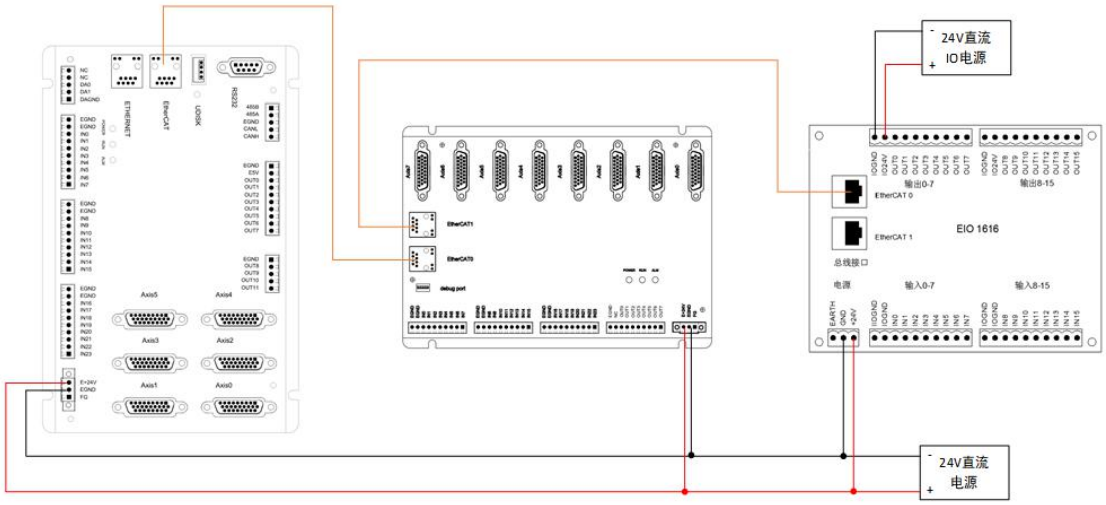
2）双电源供电的控制器和双电源供电的 ZIO 数字量扩展模块连接接线参考：

在上图单电源控制器接线的基础上，让 24V 直流 IO 电源另给控制器 IO 供电。

2．EtherCAT 总线扩展方式

EtherCAT 扩展接线较为简单，采用 EIO 扩展板，只需将各个模块的 EtherCAT 口相互连接即可，模块电源和 IO 电源的接法与 CAN 总线扩展模块相同。

连线参考如下所示。控制器为 ZMC432，采用单电源供电，EIO24088 扩展板采用单电源供电，EIO1616 扩展板采用双电源供电。



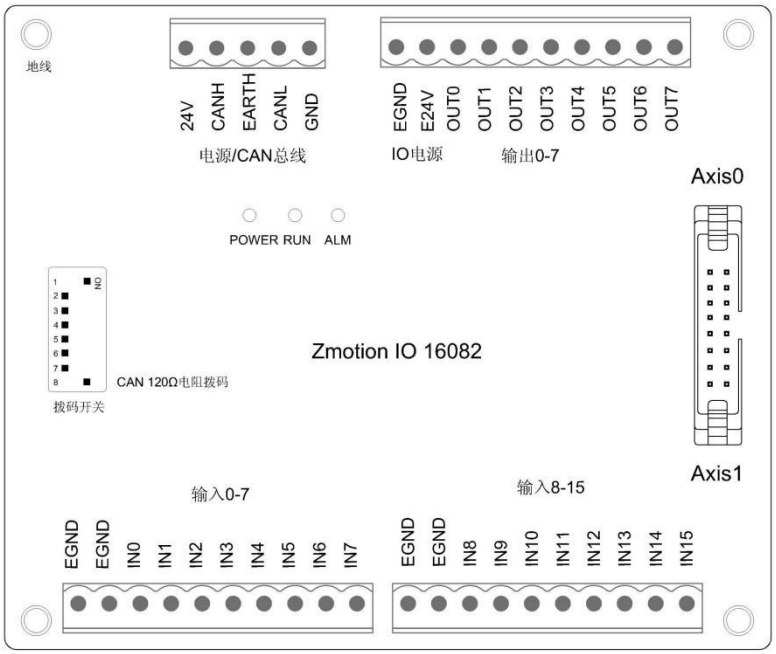
5.4.2 轴扩展

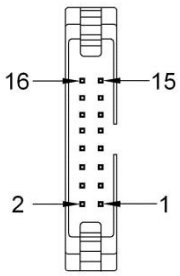
轴扩展方法与 IO 扩展接线方法相同，同样可以采用 ZCAN 总线或 EtherCAT 总线扩展，选择轴扩展模块的时候需要选择带轴接口的模块。

1．CAN 总线扩展方式

例如 ZIO16082 扩展板带两个轴接口（Axis0 和 Axis1），可以扩展两个轴，另带 16 路输入 8 路输出。

与 IO 扩展不同的是，IO 扩展通过扩展板上的拨码开关设置扩展 IO 编号后，控制器即可访问扩展 IO 口，轴扩展需要将扩展轴映射到控制器的轴资源上才可使用，轴映射方法见下节。





每个端子含两个轴信号接口，可以配置为步进轴或编码器轴。

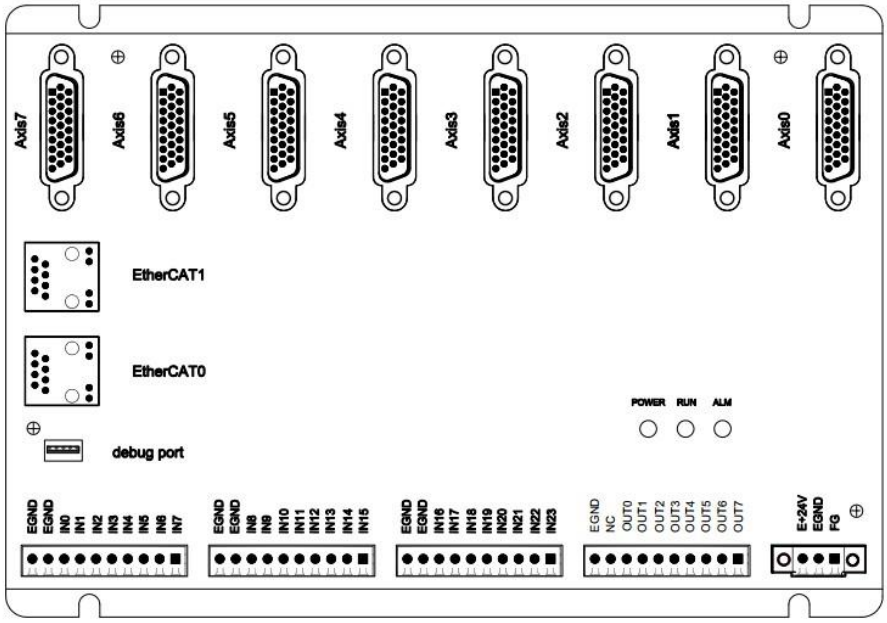
针脚号	脉冲输出方式名称	编码器方式名称
1	PUL1+(脉冲差分+)	EA1+(编码器差分+)
2	PUL1-(脉冲差分-)	EA1-(编码器差分-)

3	DIR1+(方向差分+)	EB1+(编码器差分+)
4	DIR1-(方向差分-)	EB1-(编码器差分-)
5	内部0V	内部0V
6		EZ1+(编码器差分+)
7		EZ1-(编码器差分-)
8	内部+5V电源	内部+5V电源
9	PUL0+(脉冲差分+)	EA0+(编码器差分+)
10	PUL0-(脉冲差分-)	EA0-(编码器差分-)
11	DIR0+(方向差分+)	EB0+(编码器差分+)
12	DIR0-(方向差分-)	EB0-(编码器差分-)
13	内部0V	内部0V
14		EZ0+(编码器差分+)
15		EZ0-(编码器差分-)
16	内部+5V电源	内部+5V电源

2．EtherCAT 总线扩展方式

例如 EIO24088 扩展板带八个轴接口，可以扩展八个脉冲轴，另带 16 路输入，8 路输出。

接线时 EtherCAT 输入输出口请勿混淆，EtherCAT 0 接口对应 EtherCAT IN 输入，EtherCAT 1 接口对应 EtherCAT OUT 输出。EtherCAT 0 接口用来接到主站（控制器）的 EtherCAT 网口或前级从站（伺服/EIO24088）的 EtherCAT 1 接口，EtherCAT 1 接口接到后级从站（伺服/EIO24088）的 EtherCAT 0 接口。



轴接口采用 DB26 母头接口，针脚号描述参见第一章“控制器脉冲轴接口信号”，与驱动器之间的两种接线方式参见附录“控制器与驱动器接线参考”。

5.5 轴映射

扩展轴时，基本步骤如下：

1．扩展模块与控制器模块接线

2．驱动器与扩展轴的轴接口接线，查看“控制器状态”的 ZCAN 节点信息和槽位节点信息里是否显示了扩展资源信息。

3．映射轴号

4．重新设置轴类型（ATYPE）

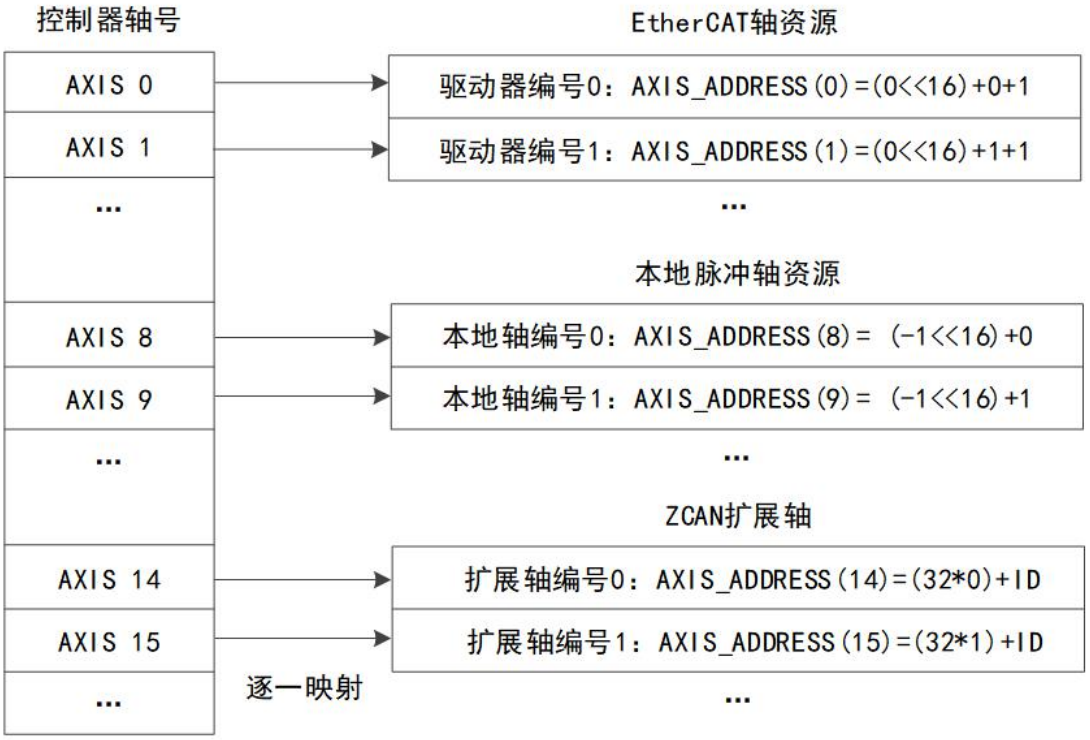
映射轴号方法：

参见 AXIS_ADDRESS 轴映射指令。

支持本地脉冲轴与 EtherCAT 轴混合插补。映射轴号之后即可调用扩展轴资源。

一个多轴驱动器上可以连接多个电机，一个电机表示一个轴，每个电机都需要轴号映射，此时相当于该驱动器可以控制多个轴，下图以一个驱动器连接一个电机的情况进行说明。

轴映射关系举例，如下所示：以 ZMC432 控制器为例，轴 0-7 作为总线轴，轴 8-13 作为脉冲轴，14 之后作为扩展轴号。



轴号可以随意映射，也可以按图上的映射方法依次配置，按顺序依次映射不容易出错，不同类型的轴通道号排序分别独立，轴号均从 0 开始计数（总线产品支持本地脉冲轴映射，较老的控制器版本，如脉冲型控制器不支持本地轴号映射）。

EtherCAT 的轴号和本地脉冲轴的轴号为各自独立的编码顺序，例如，在某次配置中需要用到两个本地脉冲轴和两个 EtherCAT 轴，配置时轴映射关系如下：

AXIS 0——本地脉冲轴 0；

AXIS 1——本地脉冲轴 1；

AXIS 2——EtherCAT 轴 0；

AXIS 3——EtherCAT 轴 1；

轴映射一般从编号 0 开始依次往下。

默认配置文件是按照所接硬件资源的通道总数进行配置。如果硬件资源大于软件资源，默认映射是将所有的软件资源按顺序映射至相应的硬件资源，多余的未映射硬件资源不可控。

例程：扩展轴轴号映射到本地使用的方法

```
ERRSWITCH = 3
BASE(0) '选择第 0 轴
CANCEL(2)
WAIT IDLE
DPOS = 0
MPOS = 0
CONST canioid = 0       'ZCAN 扩展模块的拨码 ID 0-15
CONST canioaxis = 0     'ZCAN 扩展模块的本地轴号，一般为 0 或 1
AXIS_ADDRESS(0)=canioid+(32*canioaxis)   '设置映射到 ZCAN 扩展模块 canioid 的轴 canioaxis 上
ATYPE = 8       'ZCAN 步进轴，映射完成重新设置轴类型，设置完成即可使用
UNITS = 100     '脉冲当量
SPEED = 200
ACCEL = 2000
DECEL = 2000
WHILE 1       '循环运动
    IF IN(0) = ON THEN     '输入 0 有效左运动
        VMOVE(-1)
    ELSEIF IN(1) = ON THEN     '输入 1 有效右运动
        VMOVE(1)
    ELSEIF (NOT IDLE) THEN
        CANCEL
    ENDIF
    DELAY(20)
WEND
END
```

5.6 轴的运动控制模式

轴的运行方式是运动控制非常重要的一部分。轴的实际运行方式取决于轴类型。可以通过指令 ATYPE 设置，通过轴参数状态查看轴类型。在用户程序初始化时应第一时间完成轴的配置与设置。

所有未分配的轴都设置为虚拟轴，参数 ATYPE 的值为 0。

运动控制器将运动命令应用于轴组，使用 BASE 命令定义。

如果运动命令涉及多个轴，则进行插补运动，轴组按照 BASE 命令定义的顺序确定哪个是主轴。

在每个运动控制周期开始时，执行的运动序列期间将包含以下内容：

1. 将 BASIC 运动指令传输到运动缓冲区；
2. 读取输入信息；
3. 加载运动数据；
4. 计算每段运动的速度曲线；
5. 计算轴位置；
6. 执行位置伺服，编码器反馈数据给驱动器；
7. 更新输出。

控制器支持的轴类型如下：

ATYPE 类型	描述
0	虚拟轴
1	脉冲方向方式的步进或伺服
2	模拟信号控制方式的伺服
3	正交编码器
4	脉冲方向输出+正交编码器输入
5	脉冲方向输出+脉冲方向编码器输入
6	脉冲方向方式的编码器
7	脉冲方向方式步进或伺服+EZ 信号输入
8	ZCAN 扩展脉冲方向方式步进或伺服
9	ZCAN 扩展正交编码器
10	ZCAN 扩展脉冲方向方式的编码器
21	振镜轴类型，需要控制器支持 缺省系统周期 250us，振镜刷新周期 50us，与固件有关 可以使用普通轴的所有运动控制指令，支持振镜轴与其它轴类型混合插补
50	RTEX 周期位置模式，需 RTEX 控制器
51	RTEX 周期速度模式，需 RTEX 控制器
52	RTEX 周期力矩模式，需 RTEX 控制器 请先关闭驱动器 2 自由度控制模式，并设置设置速度限制
65	EtherCAT 周期位置模式，需支持 EtherCAT
66	EtherCAT 周期速度模式，需支持 EtherCAT DRIVE_PROFILE 要设置为 20 或以上
67	EtherCAT 周期力矩模式，需支持 EtherCAT DRIVE_PROFILE 要设置为 30 或以上
70	EtherCAT 自定义操作，只读取编码器，需支持 EtherCAT

ATYPE=0 虚拟轴

多轴同步运动时作为机器的主轴，其他所有轴都遵循此虚拟主轴。

作为其他轴的叠加轴，给实际运动的轴叠加一个虚拟轴，这些虚拟轴可以用 BASIC 命令 ADDAX（轴叠加）进行设置，然后将各个虚拟轴的运动叠加到实轴。

ATYPE=1 脉冲轴

轴的运动由控制器发脉冲控制，脉冲的方向决定电机的转向，根据发送脉冲的频率控制轴运动的快慢。

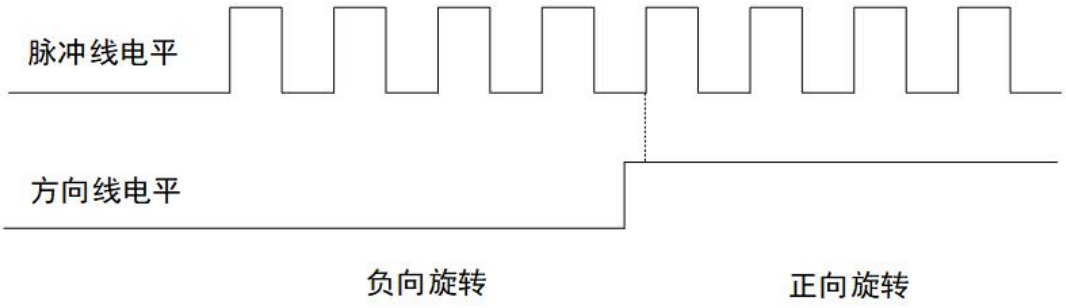
控制器脉冲输出的三种模式：

控制信号是单脉冲方式还是双脉冲方式，主要由控制器与驱动器共同配置决定，如果控制器发送的控制脉冲是单脉冲控制方式，驱动器采用单脉冲；如果控制器发送的控制脉冲是双脉冲控制方式，驱动器采用双脉冲方式。

1．脉冲方向模式：

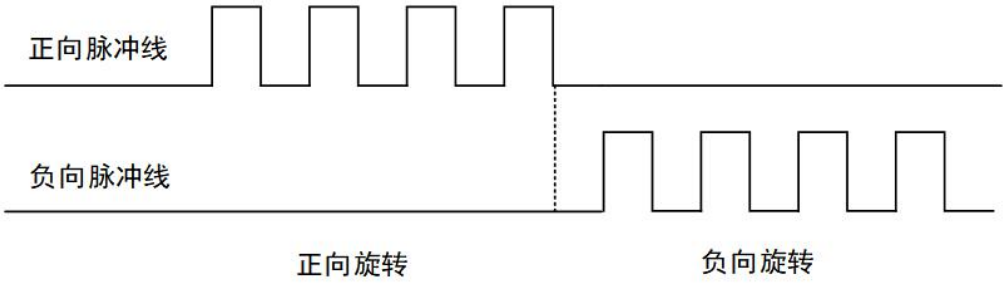
PUL+、PUL-输出指令脉冲串，脉冲数对应电机运行距离，而脉冲频率对应电机运行速度。

DIR+、DIR-输出方向信号，该信号的不同电平对应电机不同的转动方向。此种模式在驱动器中最多。



2．CW/CCW：双脉冲工作方式

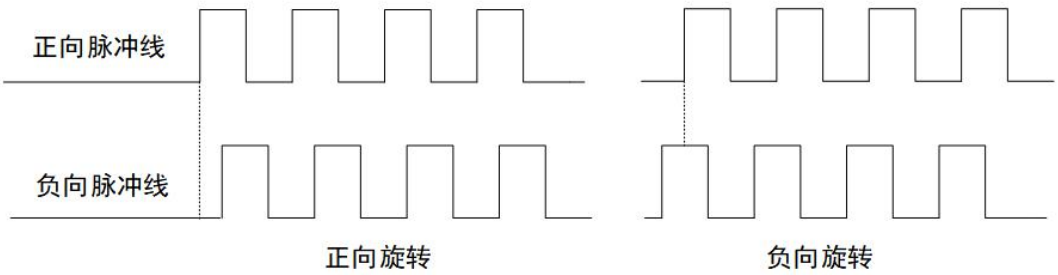
两根线都输出脉冲信号，CW 为正方向脉冲信号，CCW 为反方向脉冲信号，通常都是差分方式输出，两信号相位相差角度，根据相位超前或滞后来决定。



3．AB 相：双脉冲工作方式

指两个相互独立的相同脉冲信号（都是方波），正方向脉冲信号在负方向脉冲信号之前产生，二者相位相差 90 度，此时为正向旋转；负方向脉冲信号在正方向脉冲信号之前产生，二者相位相差 90 度，此时为负向旋转。

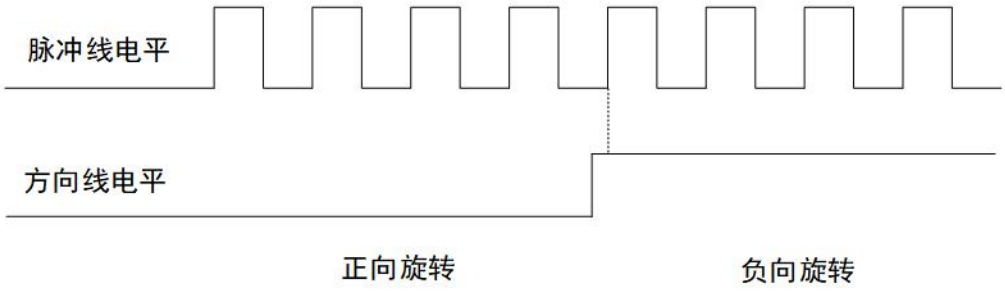
通过两个脉冲之间的相位差来达到计数或编码的作用。



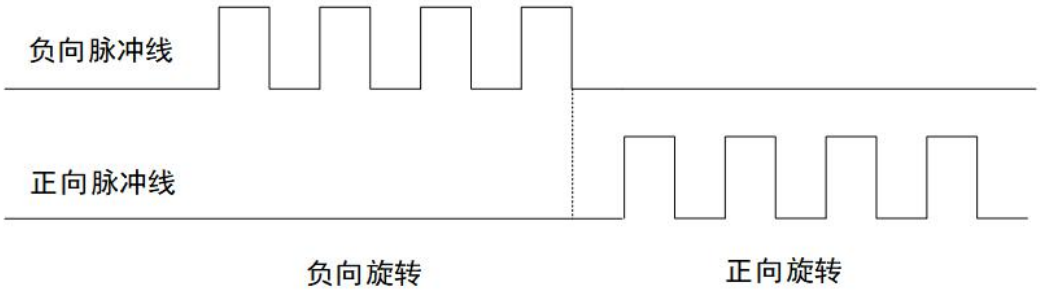
极性对调

若切换脉冲线的正负，即原本正方向脉冲信号变为负方向脉冲信号，负方向脉冲信号变为正方向脉冲信号，此时的运动方向将与上面的情况相反。

1．脉冲方向模式：



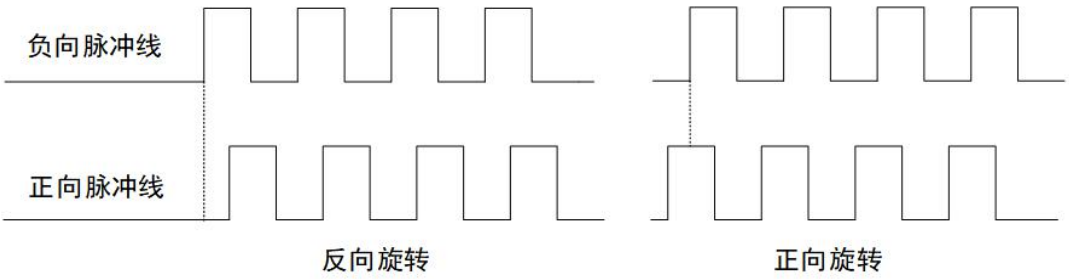
2．CW/CCW：双脉冲工作方式



3．BA 相：双脉冲工作方式

这种模式下判断旋转方向的标准是观察哪个方向先发出脉冲信号，旋转方向即为负方向

负方向脉冲信号在正方向脉冲信号之前产生，二者相位相差 90 度，此时为负向旋转，正方向脉冲信号在负方向脉冲信号之前产生，二者相位相差 90 度，此时为正向旋转。



以上模式中，脉冲方向模式和双脉冲模式两种极性，共对应 8 种不同的运动状态，AB 相/BA 相模式为部分控制器定制模式，使用 INVERT_STEP（步进脉冲输出模式）指令来设置控制器的脉冲发送。

只有 EtherCAT 总线轴才可以使用以下三种模式。

ATYPE=65 EtherCAT 周期位置模式

选择此模式使用运动指令控制电机运动。

ATYPE=66 EtherCAT 周期速度模式

此时使用 DAC 指令控制电机以设置值的速度运行，速度单位有两个，脉冲数/S 和 R/MIN，由驱动器确定，使用时先给较小的数值，观察电机速度情况，再加大。

ATYPE=67 EtherCAT 周期力矩模式

此时使用 DAC 指令控制电机以设置值的力矩运行，力矩控制模式下 DAC 值范围 0-1000，对应 0-100%，比如 DAC=10，此时电机力矩为 1%。

轴控制模式切换直接修改 ATYPE 数值即可，但是注意速度和力矩模式切换时，先将 DAC=0，防止出现事故。

第六章 运动指令

当前运动指令正在执行时，后续调用的运动指令会自动缓冲起来，ZMotion 运动控制器每个轴最多可以支持多达 4096 级运动缓冲（不同型号控制器缓冲个数有区别），当缓冲全部占用时，后续调用的运动指令会阻塞当前任务，直到缓冲有空位，任务才会继续运行。

每个运动指令都带有一个 MOVE_MARK 参数，通过 MOVE_CURMARK 可以知道当前运行到哪一条运动缓冲了。

MOVE 等单轴运动指令采用本轴的 SPEED 等各自单轴的轴参数。

MOVE 等多轴插补运动指令采用 BASE 主轴的 SPEED 等轴参数作为矢量合成速度，但其有相应的 SP 指令，SP 指令可以为每个运动指定多种不同的速度参数，例如： FORCE_SPEED、STARTMOVE_SPEED、ENDMOVE_SPEED，参见相应的*SP 指令。

轴参数 MERGE 用来设置单轴定位或轴组的多轴插补指令中间是否减速到零，MERGE=OFF 时减速到 0，MERGE=ON 时不减速，此时 BASE 主轴的轴参数 CORNER_MODE 会设置多轴插补之间是否自动减速到必要的速度。

ZMotion 运动控制器支持单轴或者轴组的运动暂停与恢复，参见 MOVE_PAUSE，MOVE_RESUME。

ZMotion 运动控制器支持运动叠加，参见 ADDAX。

6.1 单轴运动指令

ADDAX -- 运动叠加

类型	单轴运动指令
描述	**运动叠加，把一个轴的运动叠加到另一个轴。** **ADDAX 指令叠加的是脉冲个数，而不是设置的 units 单位。** 转换关系：叠加轴运动距离*叠加轴 UNITS/被叠加轴 UNITS=被叠加轴运动距离。 假设轴 A 的 UNITS 是 100，轴 B 的 UNITS 是 50,叠加轴运动 100 把轴 A 的运动叠加到轴 B，此时轴 A 显示运动了 100，轴 B 运动了 100*100/50=200。 把轴 B 的运动叠加到轴 A，此时轴 B 显示运动了 100，轴 A 运动了 100*50/100=50。 轴之间不能相互同时叠加，A 叠加到 B 后，B 不能再叠加到 A。 支持串联叠加，A 运动叠加到 B，B 在叠加到 C。 支持并联叠加，A 运动同时叠加到 B、C。 叠加时速度从被叠加轴开始变化，加减速按照叠加轴加减速及两轴 units 比例确定。 ADDAX 在轴 MTYPE 为 FRAME 或 REFRAME 的时候不起作用。
语法	ADDAX(axis)
适用控制器	通用

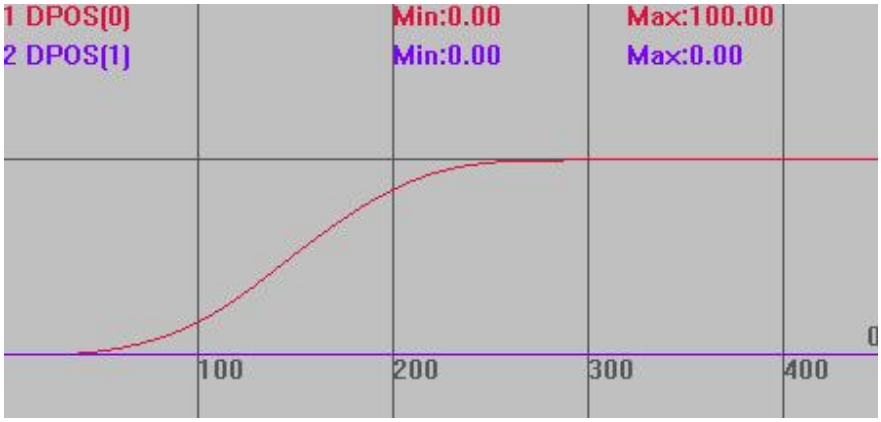
例子

```
BASE(0,1)
ATYPE=1,1
UNITS=100,200              '轴 0 UNITS 设 100，轴 1 UNITS 设 200
SPEED=1000,1000            '速度设 1000
ACCEL=10000,10000          '加速度 10000
DECEL=10000,10000          '减速度 10000
ADDAX(0)   AXIS(1)           '轴 0 的运动叠加到轴 1，按脉冲个数叠加
DPOS=0,0                   '设置位置为 0,0
TRIGGER                    '自动触发示波器
MOVE(100)                  '轴 0 运动 100，此时轴 1 运动 100*100/200=50
                           '要考虑到两轴 UNITS 的转换
WAIT IDLE                  '等待运行完
ADDAX(-1)   AXIS(1)          '取消叠加
```

不使用叠加指令的运动轨迹（无特殊说明图中示波器曲线均未设置偏移）

DPOS(0)垂直刻度 100

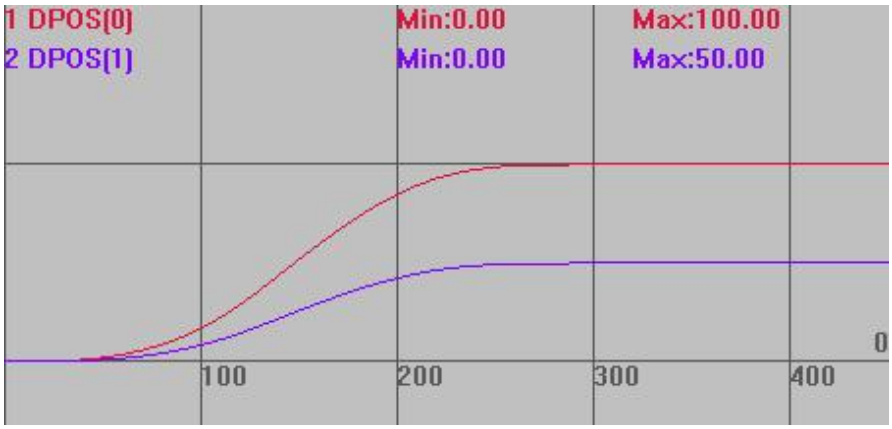
DPOS(1)垂直刻度 100



叠加指令使用后的运动轨迹

DPOS(0)垂直刻度 100

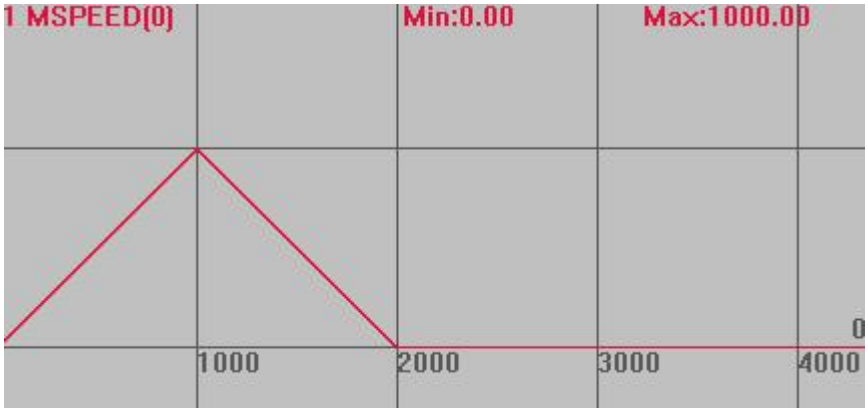
DPOS(1)垂直刻度 100



CANCEL -- 单轴停止/轴组停止

<table>
<tr><td>类型</td><td>单轴运动指令</td></tr>
<tr><td>描述</td><td>BASE 轴减速停止，如果轴参与插补，也停止插补运动。

如果指定轴在 BASE 轴列表中，无论 CANCEL 主轴或者 BASE 轴列表中的轴，都停止轴组的插补运动
MODE0~2 减速度按 FASTDEC 和 DECEL 中最大的值。
CANCEL 后要调用绝对位置运动，必须先 WAIT IDLE 等待停止完成。</td></tr>
<tr><td>语法</td><td>CANCEL(mode)
mode：模式选择
<table>
<tr><td>0（缺省）</td><td>取消当前运动</td></tr>
<tr><td>1</td><td>取消缓冲的运动</td></tr>
<tr><td>2</td><td>取消当前运动和缓冲运动</td></tr>
<tr><td>3</td><td>立即中断脉冲发送</td></tr>
</table></td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>例一
BASE(0)
DPOS=0
SRAMP=0
ATYPE=1
UNITS=100
SPEED=1000
ACCEL=1000
DECEL=1000 '减速度设为 1000
FASTDEC=10000 '快减减速度设为 10000
TRIGGER '自动触发示波器
MOVE(1000) '当前运动
MOVE(-1000) '缓冲运动
CANCEL(1) '此时轴只执行完 MOVE(1000)

运动轨迹
MSPEED(0)垂直刻度 1000
 
</td></tr>
</table>

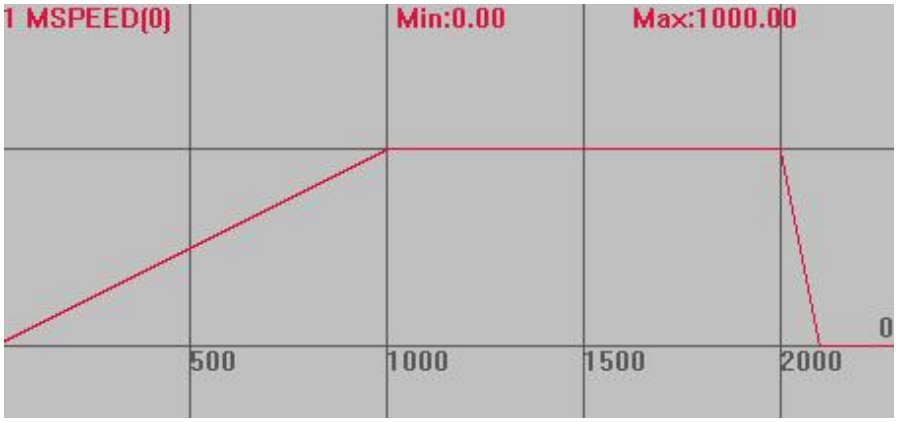
因为取消的是缓冲，当前运动还是按减速度正常停止。

例二

```
BASE(0)
ATYPE=1
DPOS=0
SPEED=100
ACCEL=1000
DECEL=1000                    '减速度设为 1000
FASTDEC=10000                 '快减减速度设为 10000
TRIGGER                       '自动触发示波器
MOVE(10000)                   '当前运动 10000
DELAY(2000)                   '延时 2 秒
CANCEL(3)      '此时直接切断脉冲发送，轴立即停止，减速度为 10000
```

运动轨迹

MSPEED(0)垂直刻度 1000

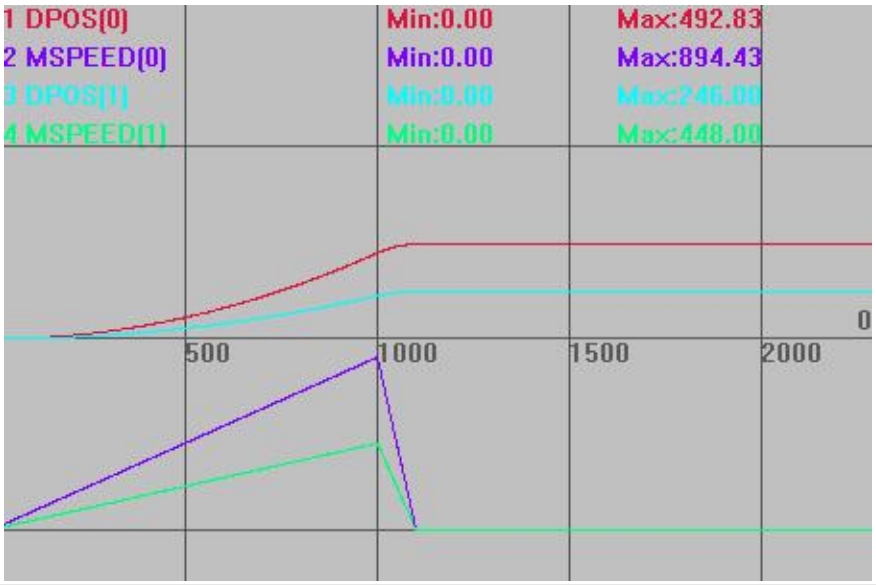


例三

```
BASE(0,1)
DPOS=1,1
ATYPE=1,1
SPEED=1000,1000
ACCEL=1000
DECEL=1000                    '减速度设为 1000
FASTDEC=10000                 '快减减速度设为 10000
SRAMP=0,0
TRIGGER
MOVE(1000,500)                '插补运动
DELAY(1000)                   '延时 1 秒
CANCEL(2)  AXIS(1)    '停止轴 1，轴 1 参与了插补，插补运动也停止减速度为 10000
```

运动轨迹和速度曲线

DPOS(0)垂直刻度 1000，无偏移

	MSPEED(0)垂直刻度 1000，偏移-1000 DPOS(1)垂直刻度 1000，无偏移 MSPEED(1)垂直刻度 1000，偏移-1000 
相关指令	RAPIDSTOP ，DECEL，FASTDEC

DATUM -- 回零

类型	单轴运动指令
描述	**单轴找原点运动。** 原点开关通过 DATUM_IN 设置，正负限位开关分别通过 FWD_IN 和 REV_IN 设置。 ZMC 控制器为 0 触发有效，输入为 OFF 状态时，表示到达原点/限位，常开类型信号需要采用 INVERT_IN 反转电平。 ECI 控制器为 1 触发有效，输入为 ON 状态时，表示到达原点/限位，常闭类型信号需要采用 INVERT_IN 反转电平。 Z 信号回零必须配置为带 Z 信号 ATYPE（ATYPE=4 或者 7）。 多轴回零时，需要每个轴都使用 datum 指令。 总线控制器使用控制器找原点模式完成后，需要手动清零 MPOS。
语法	DATUM (mode)，DATUM(21,mode2) mode：找原点模式，加 10 表示碰到限位后反找，不会碰到限位停止,例如 13=模式 3+限位反找 10，用于原点在正中间的情况。 ATYPE=4，回零模式加 100（模式 100+n 和 110+n 分别对应 n 和 10+n），表示接入编码器后可以自动清零 MPOS(仅限 4 系列)

值	描述
0	清除所有轴的错误状态。
1	轴以 CREEP 速度正向运行直到 Z 信号出现。 碰到限位开关会直接停止。

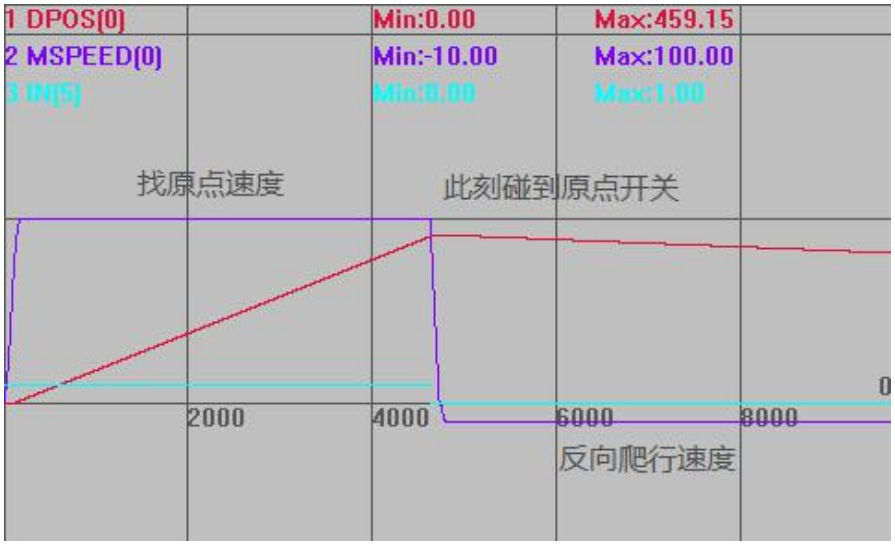
	DPOS 值重置为 0 同时纠正 MPOS。
2	轴以 CREEP 速度反向运行直到 Z 信号出现。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
3	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。 找原点阶段碰到正限位开关会直接停止。 爬行阶段碰到负限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS
4	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关。 找原点阶段碰到负限位开关会直接停止。 爬行阶段碰到正限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
5	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
6	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关，然后再继续以爬行速度正转直到碰到 Z 信号。 碰到限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。
8	轴以 SPEED 速度正向运行，直到碰到原点开关。 碰到限位开关会直接停止。
9	轴以 SPEED 速度反向运行，直到碰到原点开关。 碰到限位开关会直接停止。
21	使用 Ethercat 驱动器回零功能，此时 mode2 有效。 设置驱动器回零方式（6098h），缺省 0 表示使用驱动器当前的回零方式。 会使用轴的 SPEED, CREEP, ACCEL, DECEL，乘以 UNITS 后自动设置驱动器的 6099h，609Ah 动作时序： 6098 回零方式→6099 速度→609A 加速度→6060 切换当前模式

mode2：mode=21 时有效，缺省 0，非 0 时设置到驱动器回零方式，根据驱动器手册数据字典 6098h 设置值。

适用控制器	通用
例子	例一 直接找原点 BASE(0) DPOS=0 ATYPE=1

SPEED = 100 '找原点速度
CREEP = 10 '反向爬行速度
DATUM_IN=5 '输入 IN5 作为原点开关
INVERT_IN(5,ON) '反转 IN5 电平信号，常开信号进行反转(ZMC 控制器)
TRIGGER '自动触发示波器
DATUM(3) '轴 0 先以 100units/s 正向回零，找到原点后以 10units/s 直到离开原点，同时 DPOS 清 0

运动轨迹与速度曲线
DPOS(0)垂直刻度 500
MSPEED(0)垂直刻度 100
IN(5)垂直刻度 10

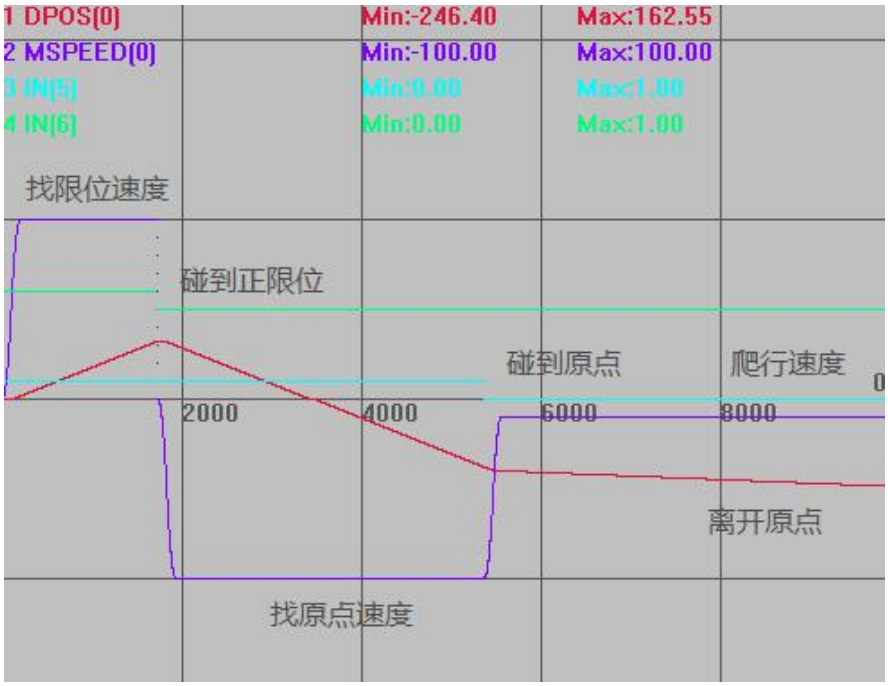


碰到原点开关之后反向爬行直到离开原点开关，此时 DPOS 清 0，回零运动完成。为了直观体现，所以爬行过程比较长，实际应用中，爬行过程很短。

例二 限位反找原点
BASE(0)
DPOS=0
ATYPE=1
SPEED = 100 '找原点速度
CREEP = 10 '反向爬行速度
DATUM_IN=5 '输入 IN5 作为原点开关
FWD_IN=6 '输入 IN6 作为正限位开关
INVERT_IN(5,ON) '反转电平信号，一般常闭的信号才进行反转
INVERT_IN(6,ON)
TRIGGER '自动触发示波器
DATUM(13) '轴 0 先以 100units/s 正向运动，碰到正限位反向运行，仍以 100units/s 找原点，找到原点后以 10units/s 直到离开原点，同时 DPOS 清 0

运动轨迹与速度曲线
DPOS(0)垂直刻度 500
MSPEED(0)垂直刻度 100

IN(5)垂直刻度 10
IN(6)垂直刻度 10



例三 EtheCAT 总线回零(松下 A6N 伺服)
先根据 EtherCAT 初始化例程正常使能电机

```
SPEED=100                    '找零速度*UNITS 后自动写入 6099
CREEP=10                     '爬行速度*UNITS 后自动写入 6099

ACCEL=1000                   '加减速*UNITS 后自动写入 609A
DECEL=1000
DATUM(21,0)                  '按驱动器当前回零模式开始回零，此时按按驱动器信号判断，而不是按控制器信号判断
WHILE 1
    TABLE(0)=DRIVE_STATUS    '读取状态字判断回零状态
    IF READ_BIT2(10,TABLE(0)) THEN   '根据下图确定
      IF READ_BIT2(12,TABLE(0)) THEN
        ?"回零完成"
      ENDIF
    ENDIF
  WEND
END
```

驱动器厂家的回零过程可能不同，具体根据驱动器手册确定。
bit13，bit12，bit10 的值组合含义如下表：

bit13	bit12	bit10	描述
0	0	0	原点回零动作中
0	0	1	原点回零动作中断或未开始

0	1	0	原点回零动作完成，但未到达目标位置
0	1	1	原点回零动作正常完成
1	0	0	检出原点回零异常还在动作中
1	0	1	检出原点回零异常，停止状态

例四 Rtex 总线回零(松下 A6N 伺服)
先根据 Rtex 初始化例程正常使能电机

```
SPEED=100                    '找零速度
ACCEL=1000                   '加减速
DECEL=1000
DATUM(21,$11)                '按驱动器当前回零模式开始回零，此时按按驱动器信号判断，而不是按控制器信号判断
```

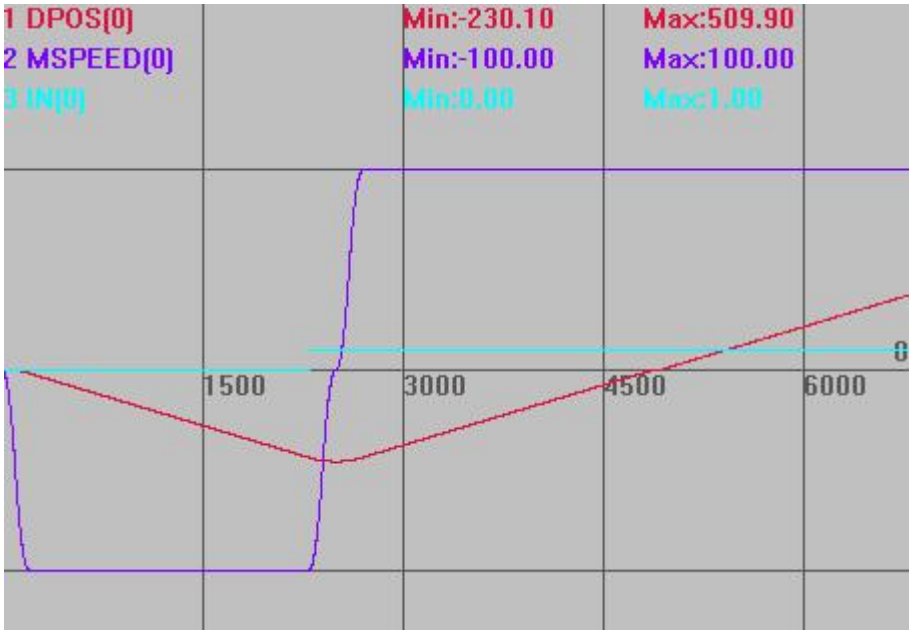
根据驱动器手册确定回零模式

初始化模式	11h	Z 相
	12h	HOME ↑ *2
	13h	HOME ↓ *3
	14h	POT ↑ *2
	15h	POT ↓ *3
	16h	NOT ↑ *2
	17h	NOT ↓ *3
	18h	EXT1 ↑ *2
	19h	EXT1 ↓ *3
	1Ah	EXT2 ↑ *2
	1Bh	EXT2 ↓ *3
	1Ch	EXT3 ↑ *2
	1Dh	EXT3 ↓ *3

相关指令	DATUM_IN, INVERT_IN

VMOVE -- 持续运动

类型	单轴运动指令
描述	**连续往一个方向运动。** 当前面的 VMOVE 运动没有停止时，此 VMOVE 指令会自动替换前面的 VMOVE 指令并修改方向，因此无需 CANCEL 前面的 VMOVE 指令。
语法	VMOVE(dir1) dir1：-1 负向运动，1 正向运动
适用控制器	通用
例子	BASE(0) DPOS=0 ATYPE=1 SPEED=100

	ACCEL=1000 DECEL=1000 '减速度设为 1000 SRAMP=100 **VMOVE**(-1) '持续负向运动 WAIT UNTIL IN(0)=ON '等待输入 1 有效 **VMOVE**(1) '持续正向运动 DPOS(0)垂直刻度 500，无偏移 MSPEED(0)垂直刻度 100，无偏移 IN(0)垂直刻度 10，无偏移 
相关指令	FORWARD，REVERSE

FORWARD -- 正向运动

类型	单轴运动指令
描述	**BASE 选择轴正向运动** 必须 CANCEL 后才能切换 REVERSE。
语法	FORWARD [axis(轴号)]
适用控制器	通用
例子	例一 BASE(0) **FORWARD** '轴 0 持续正向运动 WAIT UNTIL IN(1)=ON '等待输入 1 有效 CANCEL(2) 例二 **FORWARD** AXIS(1) '轴 1 正向运动

	WAIT UNTIL IN(1)=ON '等待输入 1 有效 CANCEL(2) AXIS(1)
相关指令	REVERSE，VMOVE

REVERSE -- 负向运动

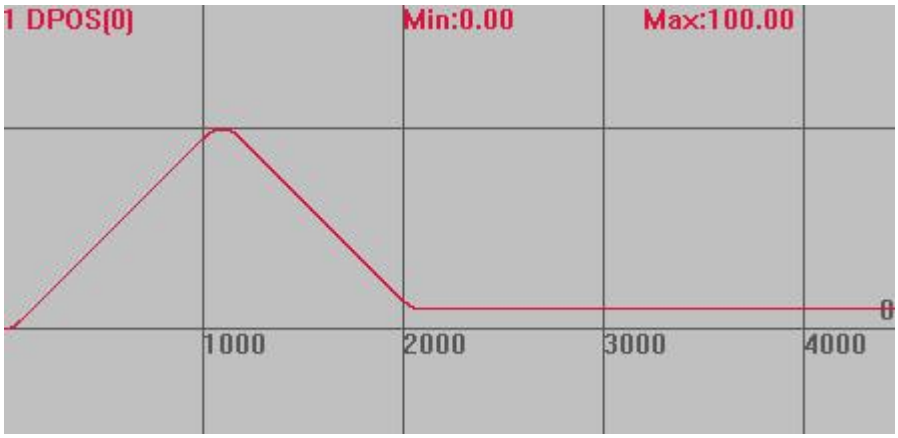
类型	单轴运动指令
描述	**BASE 选择轴负向运动。** 必须 CANCEL 后才能切换 FORWARD。
语法	REVERSE [axis(轴号)]
适用控制器	通用
例子	例一 BASE(0) **REVERSE** '轴 0 持续负向运动 WAIT UNTIL IN(1)=ON '等待输入 1 有效 CANCEL(2) 例二 **REVERSE** AXIS(1) '轴 1 负向运动 WAIT UNTIL IN(1)=ON '等待输入 1 有效 CANCEL(2) AXIS(1)
相关指令	FORWARD，VMOVE

MOVEMODIFY -- 修改运动位置

类型	单轴运动指令
描述	**修改上一个运动的目标位置。** 前面没有运动时与 MOVEABS 效果一样，但不会进入运动缓冲。见例一 需要 WAIT 指令才能正确使用。见例二 连续插补时使用 MOVEMODIFY 会破坏速度连续性。 MOVEMODIFY 同时对多轴运动时不一定为直线插补。
语法	MOVEMODIFY (distance) distance：单个轴的运动距离 目前只支持单轴修改。
适用控制器	通用
例子	例一 BASE(0) UNITS=100 '脉冲当量设置 DPOS=0

```
SPEED=100                 '速度设置
ACCEL=1000                '加速度设置
DECEL=1000
TRIGGER                   '自动触发示波器
MOVEABS(100)
MOVEABS(10)               '此时轴会先运动到 100，在往回运动到 10
```

运动轨迹
DPOS(0)垂直刻度 100



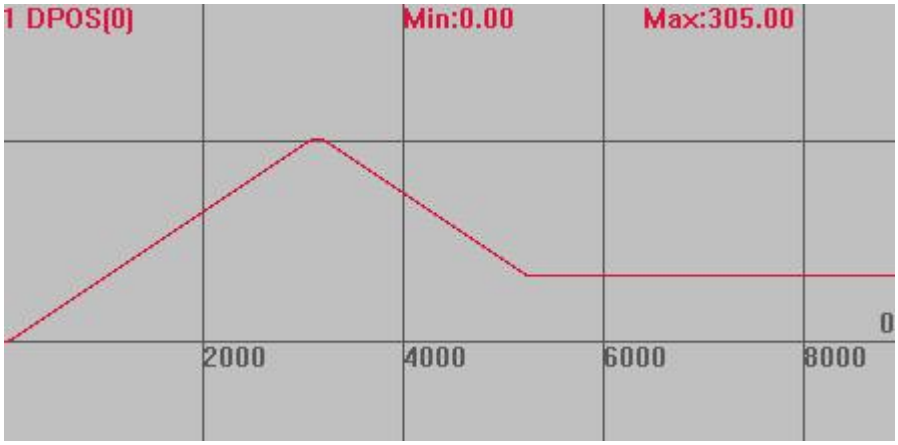
将 MOVEABS 指令变为如下指令：

```
MOVEMODIFY(100)
MOVEMODIFY(10) '此时轴会直接运动到 10 位置，MOVEMODIFY 不进入运动缓冲
```

运动轨迹
DPOS(0)垂直刻度 100

例二

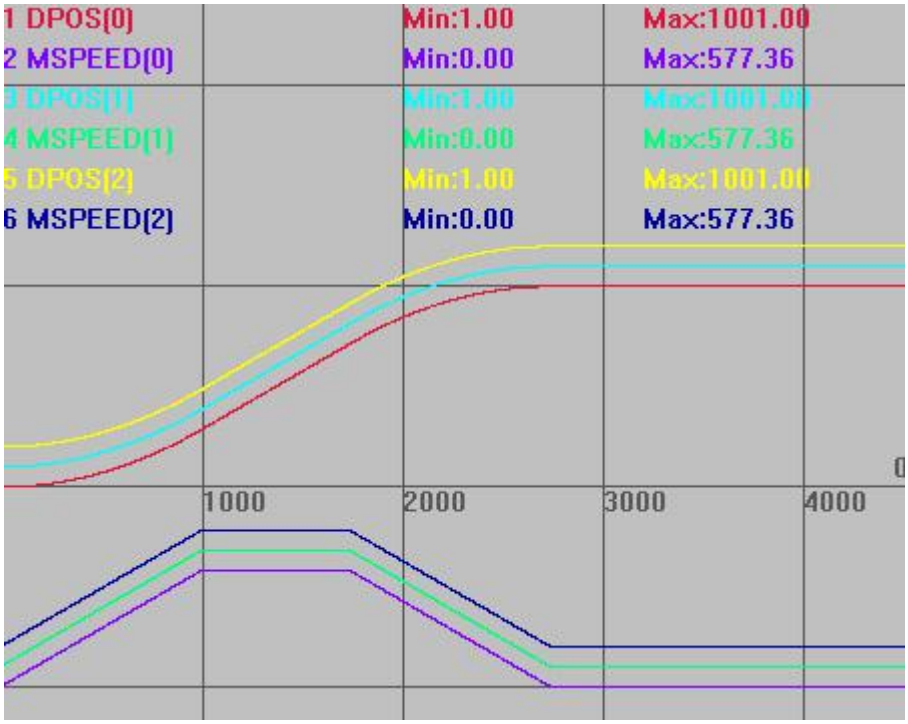
```
BASE (0)
UNITS=100                 '脉冲当量设置
DEFPOS(0)
SPEED=100                 '速度设置
ACCEL=1000                '加速度设置
DECEL=1000
TRIGGER                   '自动触发示波器
```

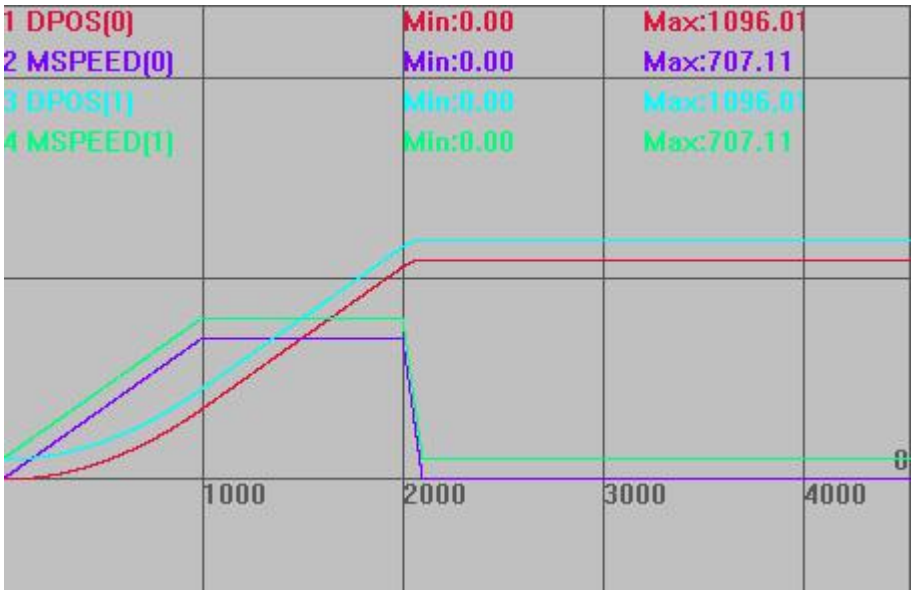
	MOVEABS(500) WAIT UNTIL DPOS >=300 '等待运动到 300 时，才修改终点位置 **MOVEMODIFY** (100) '修改目标位置为 100，此时减速停止后再反向运动 使用 WAIT，运动轨迹 DPOS(0)垂直刻度 300  不使用 WAIT，直接运动到 100，运动轨迹 DPOS(0)垂直刻度 300 
相关指令	MOVEMODIFY2

6.2 多轴运动指令

RAPIDSTOP -- 全部轴停止

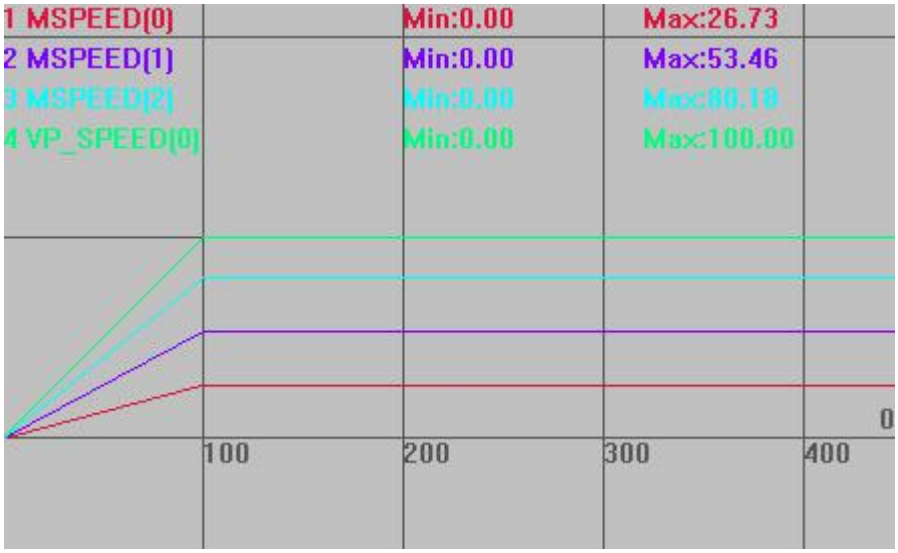
类型	多轴运动指令
描述	**所有轴立即停止，如果轴参与插补，也停止插补运动。** Mode0~2 减速度按 FASTDEC 和 DECEL 中最大的值。 RAPIDSTOP 后要调用绝对位置运动，必须先 WAIT IDLE 等待停止完成。
语法	RAPIDSTOP (mode) mode：模式选择 0（缺省） 取消当前运动

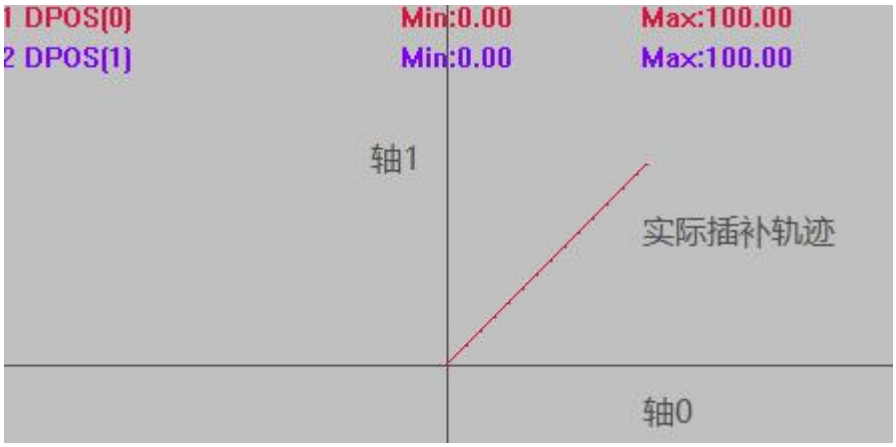
		1	取消缓冲的运动
		2	取消当前运动和缓冲运动
		3	立即中断脉冲发送
适用控制器	通用		
例子	例一 BASE(0,1,2) DPOS=1,1,1 ATYPE=1,1,1 UNITS=100,100,100 SPEED=1000 '插补合成运动速度 100 ACCEL=1000 DECEL=1000 '减速度设为 1000 FASTDEC=10000 '快减减速度设为 10000 TRIGGER MOVE(1000,1000,1000) '当前运动 MOVE(-1000,-1000,-1000) '缓冲运动 **RAPIDSTOP**(1) '此时轴只执行完当前运动 运动轨迹和速度曲线 DPOS(0)垂直刻度 1000，无偏移 MSPEED(0)垂直刻度 1000，偏移-1000 DPOS(1)垂直刻度 1000，偏移 100 MSPEED(1)垂直刻度 1000，偏移-900 DPOS(2)垂直刻度 1000，偏移 200 MSPEED(2)垂直刻度 1000，偏移-800  		

	例二 BASE(0,1) DPOS=0,0 ATYPE=1,1 SPEED=1000 ACCEL=1000 DECEL=1000 '减速度设为 1000 FASTDEC=10000 '快减减速度设为 10000 TRIGGER MOVE(10000,10000) '插补运动 10000 DELAY(2000) '延时 2 秒 **RAPIDSTOP** (3) '此时直接切断脉冲发送，轴立即停止，减速度 10000 运动轨迹和速度曲线 DPOS(0)垂直刻度 1000，无偏移 MSPEED(0)垂直刻度 1000，无偏移 DPOS(1)垂直刻度 1000，偏移 100 MSPEED(1)垂直刻度 1000，偏移 100 
相关指令	CANCEL，DECEL，FASTDEC

MOVE -- 直线运动

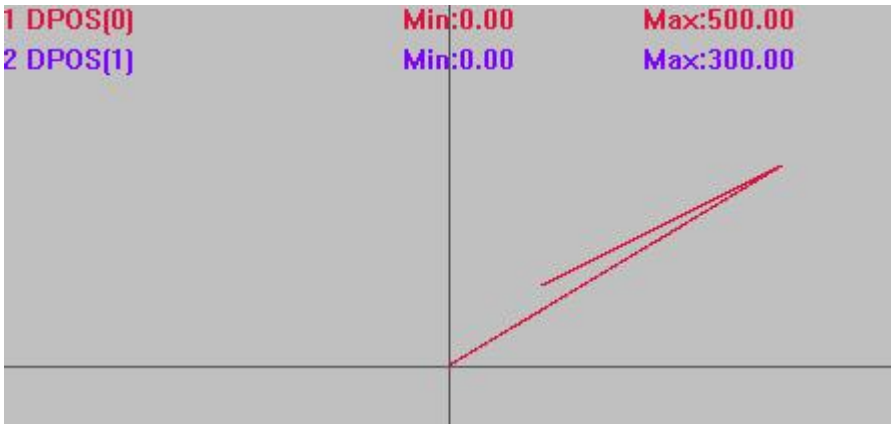
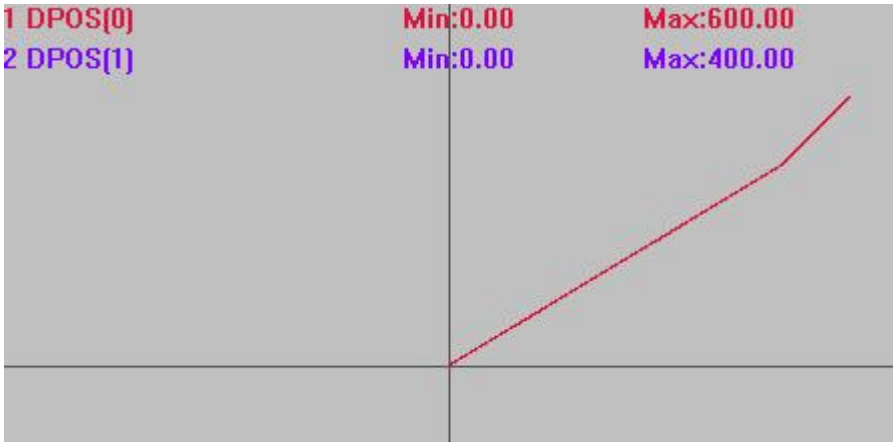
类型	多轴运动指令
描述	**直线插补运动，相对运动一段距离。** 插补运动时只有主轴速度参数有效，主轴是 BASE 的第一个轴，运动参照这个轴的参数。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。

	插补运动距离 X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ 运动时间 T=X/主轴 SPEED
语法	MOVE(distance1 [,distance2 [,distance3 [,distance4...]]]) distance1：第一个轴运动距离 distance2：下一个轴运动距离
适用控制器	通用
例子	例一 BASE(0,1,2) '主轴为轴 0 ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 '脉冲当量设置 SPEED=100,10,1000 '只有主轴速度 100 起作用，作为合成运动的速度 ACCEL=1000,1000,1000 DECEL=1000,1000,1000 DPOS = 0,0,0 TRIGGER '自动触发示波器 **MOVE**(500,1000,1500) '轴 0，1，2 直线插补相对距离 WAIT IDLE '等待运动停止 PRINT *DPOS '打印结果，500,1000,1500 插补运动各轴实际速度为主轴速度的分速度 MSPEED(0)垂直刻度 100 MSPEED(1)垂直刻度 100 MSPEED(2)垂直刻度 100 VP_SPEED(0)垂直刻度 100  例二 BASE(0,1) ATYPE=1,1

	UNITS=100,100 '脉冲当量设置 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 DPOS=0,0 MPOS=0,0 TRIGGER '自动触发示波器 **MOVE**(100,100) 插补轨迹 DPOS(0)水平刻度 100 DPOS(1)垂直刻度 100 
相关指令	MOVEABS,*SP

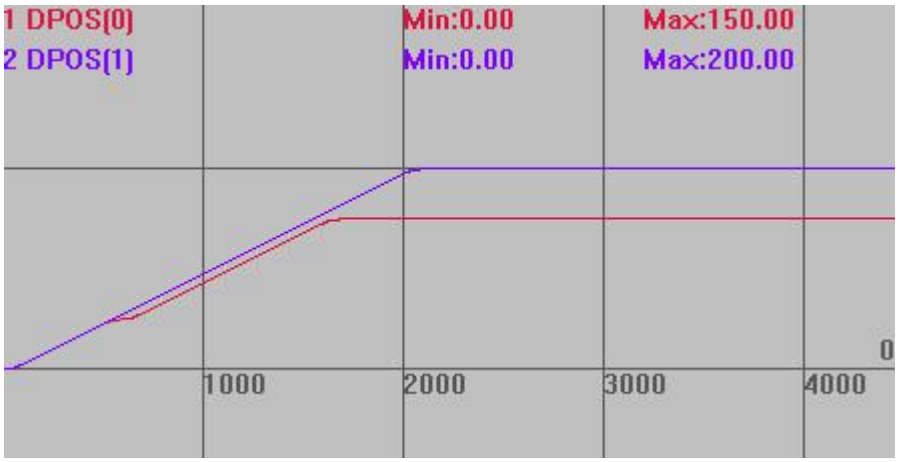
MOVEABS -- 直线运动-绝对

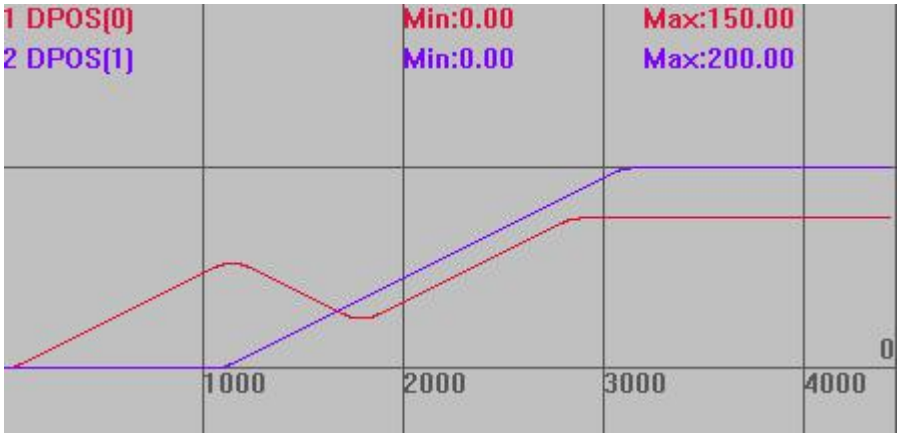
类型	多轴运动指令
描述	**直线插补运动，绝对运动到指定坐标**。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MOVEABS(position1[, position2[, position3[, position4...]]]) position1：第一个轴运动坐标 position2：下一个轴运动坐标
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1 UNITS=100,100 DPOS=0,0 MPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000

	TRIGGER '自动触发示波器 **MOVEABS**(500,300) '轴 0 运动到 500，轴 1 运动到 300，插补运动 **MOVEABS**(100,100) '轴 0 往回运动到 100，轴 1 往回运动到 100 插补轨迹 DPOS(0)水平刻度 300 DPOS(1)垂直刻度 300  如果使用 MOVE 相对运动，其他条件不变，将 MOVEABS 指令改为 MOVE 指令。 插补轨迹 DPOS(0)水平刻度 300 DPOS(1)垂直刻度 300 
相关指令	MOVE，*SP

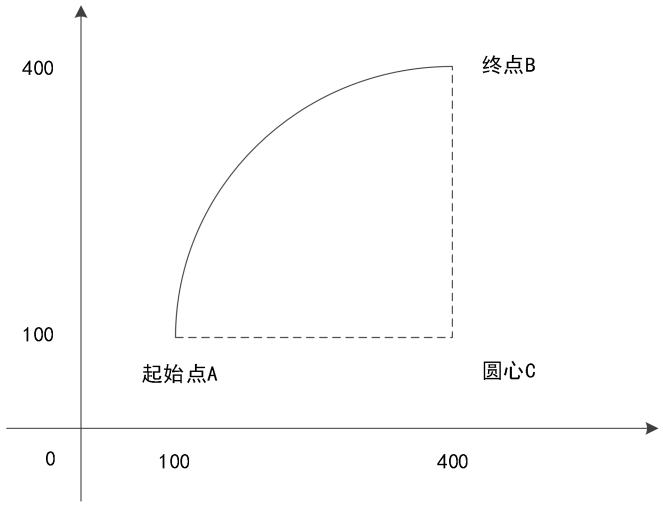
MOVEMODIFY2 -- 运动新位置

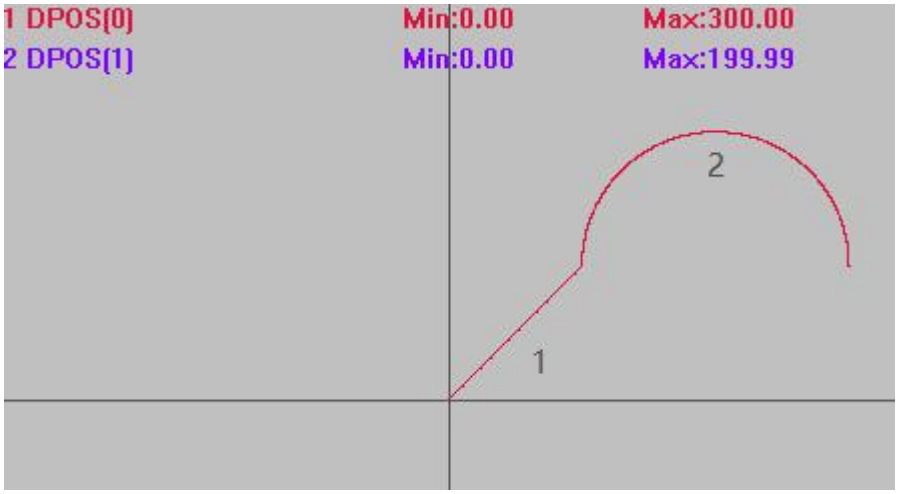
类型	运动指令
描述	**强制停止原来的运动，接着按原来的速度与加速度定位到新目标位置。** 前面没有运动时与 MOVEABS 效果一样，但不会进入运动缓冲。见例一。 需要 WAIT 指令才能正确使用。见例二。 连续插补时使用 MOVEMODIFY2 会破坏速度连续性。

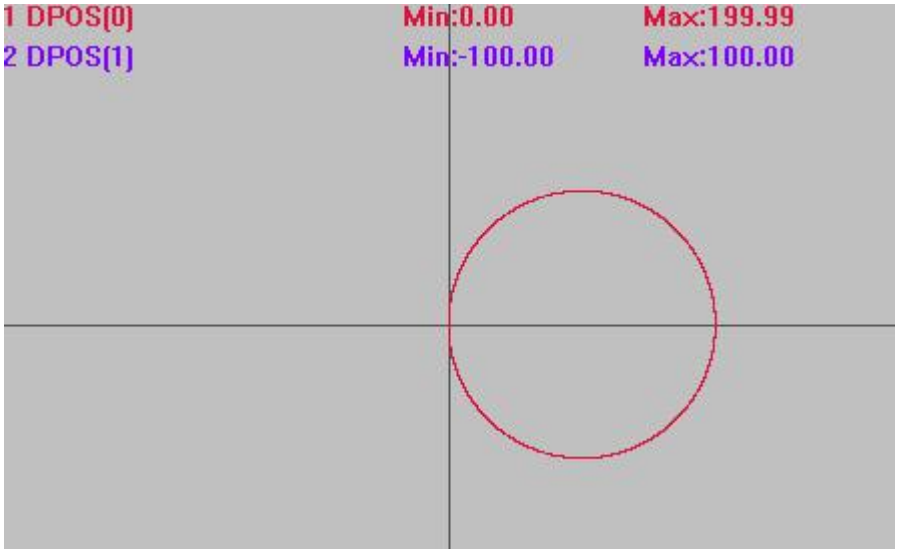
	MOVEMODIFY2 同时对多轴运动时不一定为直线插补。
语法	MOVEMODIFY2 (abspos1, abspos2,[...]) abspos1：BASE 第一个轴的新目标位置 abspos2：BASE 第二个轴的新目标位置 3X 系列 20161209 以上固件支持. 4 系列 20170509 以上固件支持。
适用控制器	特殊固件
例子	例一 BASE(0,1) ATYPE=1,1 '设为脉冲轴类型 DPOS=0,0 SPEED = 100,100 ACCEL=1000 '加速度设置 DECEL=1000 TRIGGER MOVE(200) **MOVEMODIFY2(50,200)** '取消 MOVE（200），强制定位新的位置 50,200 MOVE(100) 运动轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200  例二 BASE(0,1) ATYPE=1,1 '设为脉冲轴类型 DPOS=0,0 SPEED = 100,100 ACCEL=1000 '加速度设置 DECEL=1000 TRIGGER MOVE(200)

	WAIT UNTIL DPOS(0)>=100 '等待轴 0 运行过 100 **MOVEMODIFY2(50,200)** MOVE(100) 运动轨迹 竖直刻度同上 
相关指令	MOVEMODIFY

MOVECIRC -- 圆心画弧

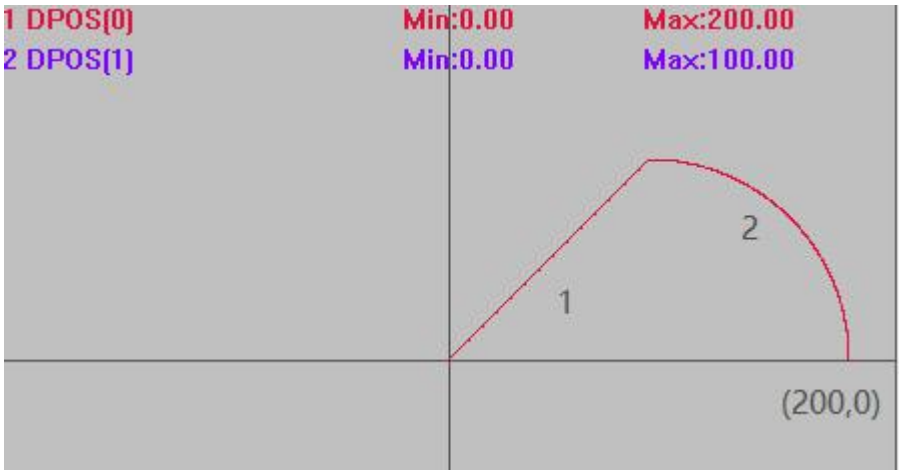
类型	多轴运动指令
描述	**两轴圆弧插补，圆心画弧，相对运动。** BASE 第一轴和第二轴进行圆弧插补，相对移动方式，当终点距离为 0 时为整圆。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 使用时需获得圆心和圆弧终点相对于起始点坐标。 坐标位置请确保正确，否则实际运动轨迹会错误。  假设起始点 A 坐标为（100,100），圆心 C 坐标为（400,100），终点 B 坐标为（400,400）。 圆心 C 相对于起始点 A 的坐标为（300,0），终点 B 相对于起始点 A 的坐标为（300,300）。

语法	MOVECIRC(end1, end2, centre1, centre2, direction) end1：终点第一个轴运动坐标，相对于起始点 end2：终点第二个轴运动坐标，相对于起始点 centre1：圆心第一个轴运动坐标，相对于起始点 centre2：圆心第二个轴运动坐标，相对于起始点 direction：0-逆时针，1-顺时针
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1　　'设为脉冲轴类型 UNITS=100,100 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER　　'自动触发示波器 MOVE(100,100)　　'先运动 100,100 位置 **MOVECIRC**(200,0,100,0,1)　'半径 100 顺时针画半圆，终点坐标（300,100） 插补轨迹 DPOS(0)垂直刻度 150 DPOS(1)垂直刻度 150  其他条件不变，将运动指令修改。 **MOVECIRC**(0,0,100,0,0)　'半径 100，圆心（100，0）逆时针画圆 插补轨迹 竖直刻度同上

	
相关指令	MOVECIRCABS，MOVECIRC2，*SP

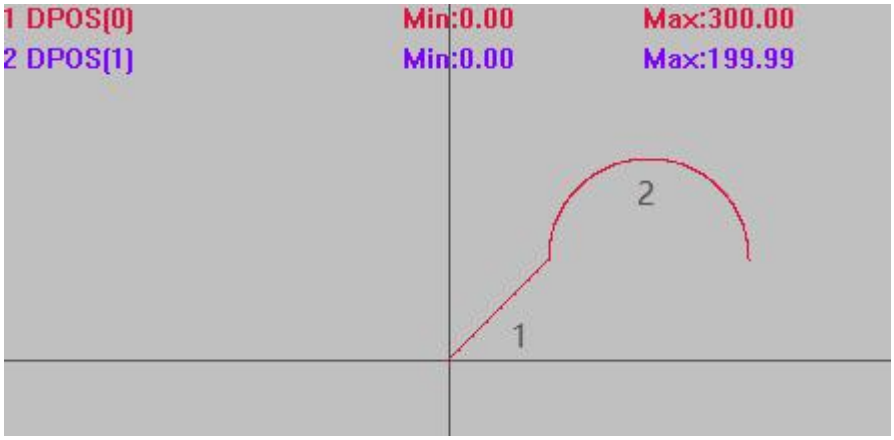
MOVECIRCABS -- 圆心画弧-绝对

类型	多轴运动指令
描述	**两轴圆弧插补，圆心画弧，绝对运动。** BASE 第一轴和第二轴进行圆弧插补，绝对移动方式。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MOVECIRCABS(end1, end2, centre1, centre2, direction) end1：终点第一个轴运动坐标，绝对位置 end2：终点第二个轴运动坐标，绝对位置 centre1：圆心第一个轴运动坐标，绝对位置 centre2：圆心第二个轴运动坐标，绝对位置 direction：0-逆时针，1-顺时针 坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1 '设为脉冲轴类型 UNITS=100,100 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER '自动触发示波器 MOVE(100,100) '先运动 100,100 位置 **MOVECIRCABS**(200,0,100,0,1) '半径 100 顺时针画 1/4 圆，终点坐标（200,0）

	插补轨迹 DPOS(0)垂直刻度 100 DPOS(1)垂直刻度 100 
相关指令	MOVECIRC，MOVECIRC2ABS，*SP

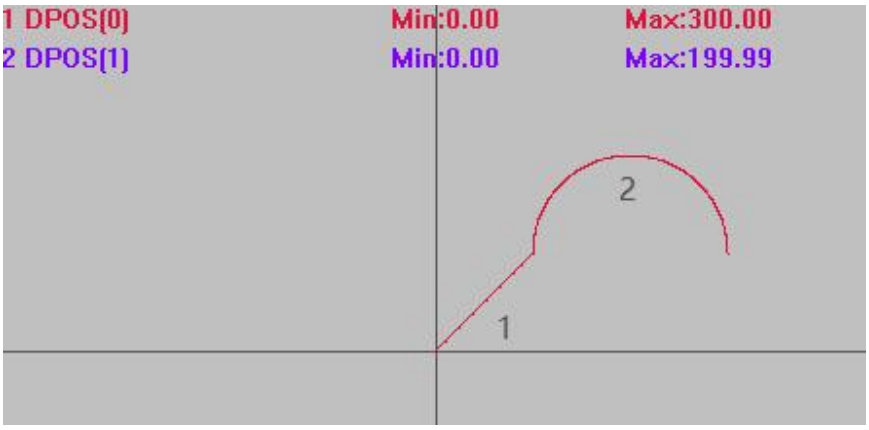
MOVECIRC2 -- 三点画弧

类型	多轴运动指令
描述	**两轴圆弧插补，三点画弧，相对运动。** BASE 第一轴和第二轴进行圆弧插补，相对移动方式。相对移动方式，相对起始点的距离。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。
语法	MOVECIRC2(mid1, mid2, end1, end2) mid1：中间点第一个轴坐标，相对起始点距离 mid2：中间点第二个轴坐标，相对起始点距离 end1：结束点第一个轴坐标，相对起始点距离 end2：结束点第二个轴坐标，相对起始点距离 坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1　　'设为脉冲轴类型 UNITS=100,100 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000

	DECEL=1000,1000 TRIGGER '自动触发示波器 MOVE(100,100) '运动 100，100 位置 **MOVECIRC2**(100,100,200,0) '三点画半圆，相对坐标 插补轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200 
相关指令	MOVECIRC2ABS，MOVECIRC，*SP

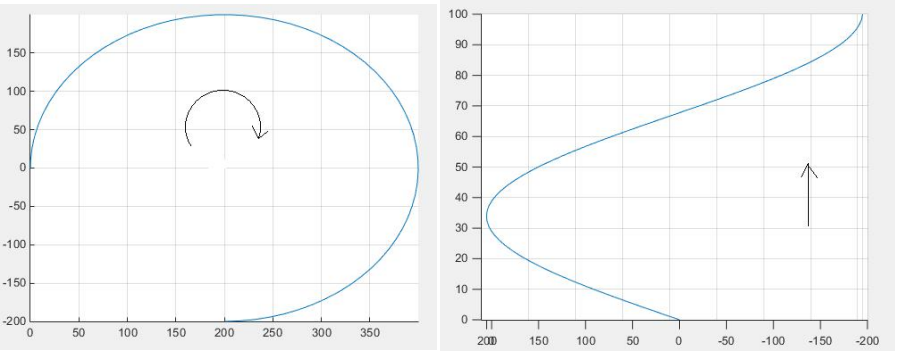
MOVECIRC2ABS -- 三点画弧-绝对

类型	多轴运动指令
描述	**圆弧插补，三点画弧，绝对运动。** BASE 第一轴和第二轴进行圆弧插补，绝对移动方式。自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。
语法	MOVECIRC2ABS(mid1, mid2, end1, end2) mid1：中间点第一个轴坐标，绝对位置 mid2：中间点第二个轴坐标，绝对位置 end1：结束点第一个轴坐标，绝对位置 end2：结束点第二个轴坐标，绝对位置 坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1 '设为脉冲轴类型 UNITS=100,100 DPOS=0,0

	SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER '自动触发示波器 MOVE(100,100) '先运动 100,100 位置 **MOVECIRC2ABS**(200,200,300,100) '三点画半圆，绝对坐标 插补轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200 
相关指令	MOVECIRC2，MOVECIRCABS，*SP

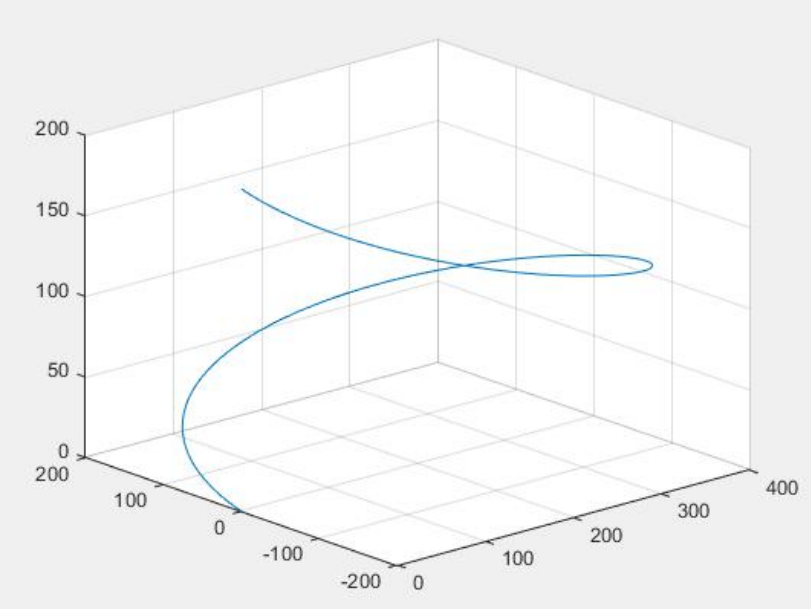
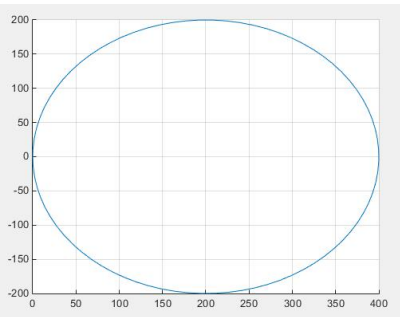
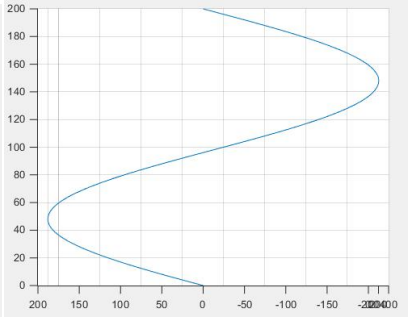
MHELICAL -- 圆心螺旋

类型	多轴运动指令
描述	**螺旋插补，圆心画弧，相对运动。** BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，相对于起始点。自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 可完整螺旋一圈，从 Z 方向看为一整圆。
语法	MHELICAL(end1,end2,centre1,centre2,direction,distance3,[mode]) end1：结束点第一个轴运动坐标，相对于起始点 end2：结束点第二个轴运动坐标，相对于起始点 centre1：圆心第一个轴运动坐标，相对于起始点 centre2 ：圆心第一个轴运动坐标，相对于起始点 direction：0-逆时针，1-顺时针 distance3：第三个轴运动距离 mode：第三轴的速度计算 值 | 描述 0(缺省) | 第三轴参与插补速度计算 1 | 第三轴不参与插补速度计算 坐标位置请确保正确，否则实际运动轨迹会错误。

适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1　　'设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100　　'主轴速度 ACCEL=1000 ,1000,1000　　'主轴加速度 DECEL=1000 ,1000,1000 **MHELICAL**(200,-200,200,0,1,100)　'原点作为起点，中心(200,0)，终点'(200,-200)，顺时针，Z 轴参与速度计算，运动 100 插补轨迹 
相关指令	MHELICAL2，MHELICALABS，*SP

MHELICALABS -- 圆心螺旋-绝对

类型	多轴运动指令
描述	**螺旋插补，圆心画弧，绝对运动。** BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 可完整螺旋一圈，从 Z 方向看为一个整圆。
语法	MHELICALABS(end1,end2,centre1,centre2,direction,distance3,[mode]) end1：第一个轴运动坐标 end2：第二个轴运动坐标 centre1：第一个轴运动圆心 centre2　：第二个轴运动圆心 direction：0-逆时针，1-顺时针 distance3：第三个轴运动坐标 mode：第三轴的速度计算： 值 | 描述 0(缺省) | 第三轴参与插补速度计算 1 | 第三轴不参与插补速度计算

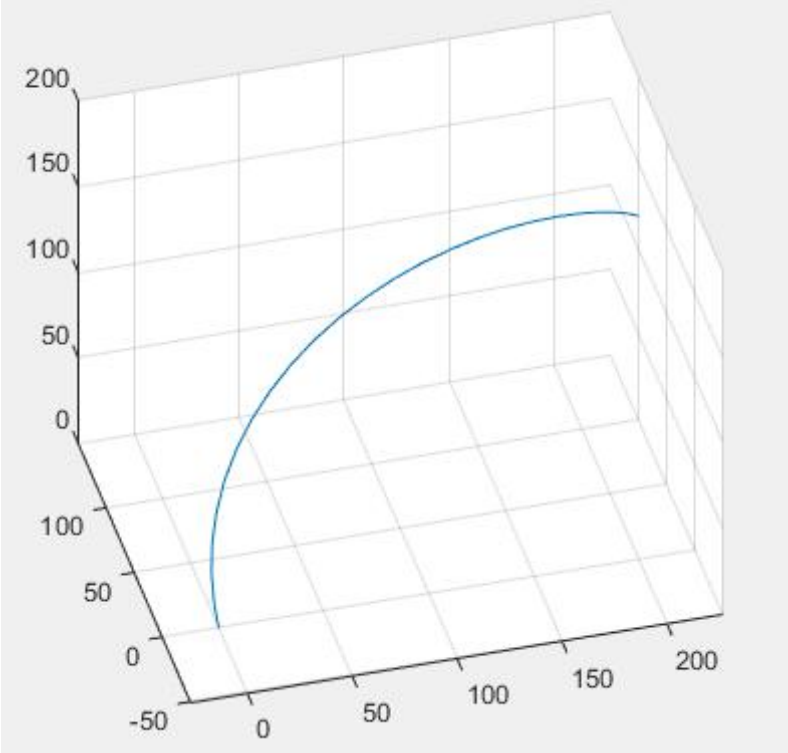
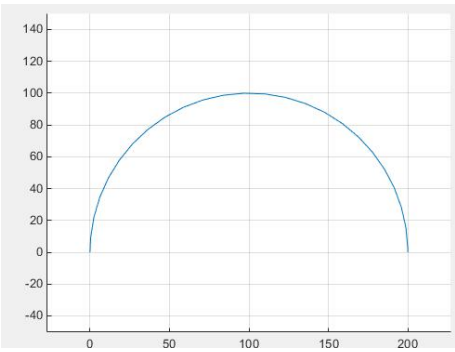
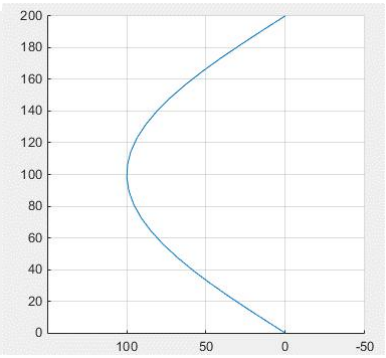
	坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1　　'设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100　　'主轴速度 ACCEL=1000 ,1000,1000　　'主轴加速度 DECEL=1000 ,1000,1000 **MHELICALABS**(0,0,200,0,1,200)　　'起点原点，中心(200,0)，终点（0,0），'顺时针，Z轴参与速度计算，运动 200 插补轨迹   
相关指令	MHELICAL，MHELICAL2ABS，*SP

MHELICAL2 -- 三点螺旋

类型	多轴运动指令
描述	**螺旋插补，三点画弧，相对运动。**

<table>
<tr><td></td><td>BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，相对移动方式。
自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。</td></tr>
<tr><td>语法</td><td>MHELICAL2(mid1, mid2, end1, end2, distance3,[mode])
mid1：中间点第一个轴坐标，相对起始点距离
mid2：中间点第二个轴坐标，相对起始点距离
end1：结束点第一个轴坐标，相对起始点距离
end2：结束点第二个轴坐标，相对起始点距离
distance3：第三个轴运动距离，相对起始点距离
mode：第三轴的速度计算
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0(缺省)</td><td>第三轴参与插补速度计算</td></tr>
<tr><td>1</td><td>第三轴不参与插补速度计算</td></tr>
</table>
坐标位置请确保正确，否则实际运动轨迹会错误。</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>BASE(0,1,2)
ATYPE=1,1,1 '设为脉冲轴类型
UNITS=100,100,100
DPOS=0,0,0
SPEED=100,100,100 '主轴速度
ACCEL=1000 ,1000,1000 '主轴加速度
DECEL=1000 ,1000,1000
MHELICAL2(100,100,200,0,200) '起点原点，中间点(100,100)，终点（200,0），Z 轴参与速度计算，运动 200

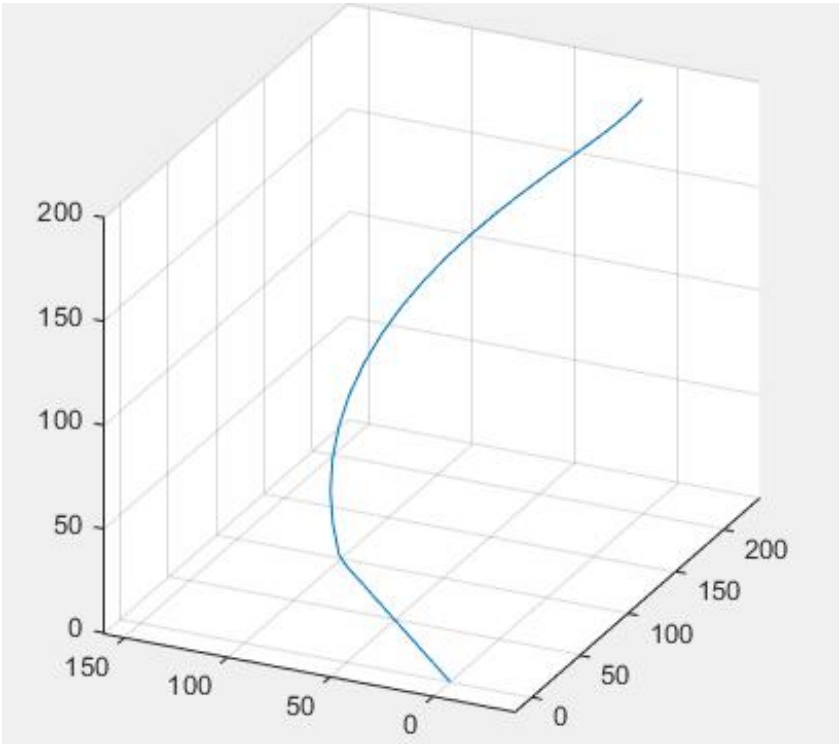
插补轨迹</td></tr>
</table>

	  
相关指令	MHELICAL2ABS，MHELICAL，*SP

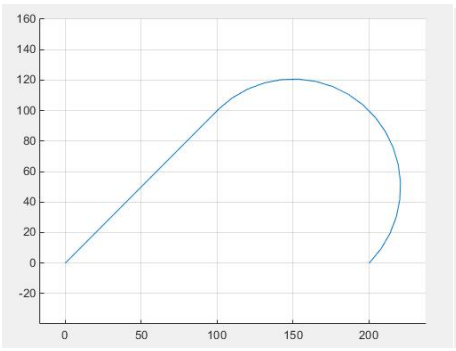
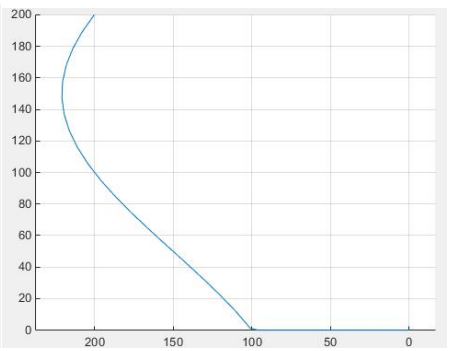
MHELICAL2ABS -- 三点螺旋-绝对

类型	多轴运动指令
描述	**螺旋插补，三点画弧，绝对运动。** BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。
语法	MHELICAL2ABS(mid1, mid2, end1, end2, distance3,[mode]) mid1：中间点第一个轴坐标 mid2：中间点第二个轴坐标 end1：结束点第一个轴坐标 end2：结束点第二个轴坐标

<table>
<tr><td></td><td>
distance3：第三个轴结束点坐标，勘误：20150306 以前的版本此参数有问题，建议使用 MHELICAL2 相对指令
mode：第三轴的速度计算
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0(缺省)</td><td>第三轴参与插补速度计算</td></tr>
<tr><td>1</td><td>第三轴不参与插补速度计算</td></tr>
</table>
坐标位置请确保正确，否则实际运动轨迹会错误。
</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>
BASE(0,1,2)
ATYPE=1,1,1　　'设为脉冲轴类型
UNITS=100,100,100
DPOS=0,0,0
SPEED=100,100,100　　'主轴速度
ACCEL=1000 ,1000,1000　　'主轴加速度
DECEL=1000 ,1000,1000
MOVE(100,100)　　'运动到（100,100）
MHELICAL2ABS(200,100,200,0,200)　　'起点(100,100)，中间点'(200,100)，结束点（200,0），Z 轴参与速度计算，运动 200

插补轨迹


</td></tr>
</table>

	 
相关指令	MHELICAL2，MHELICALABS，*SP

MECLIPSE -- 椭圆

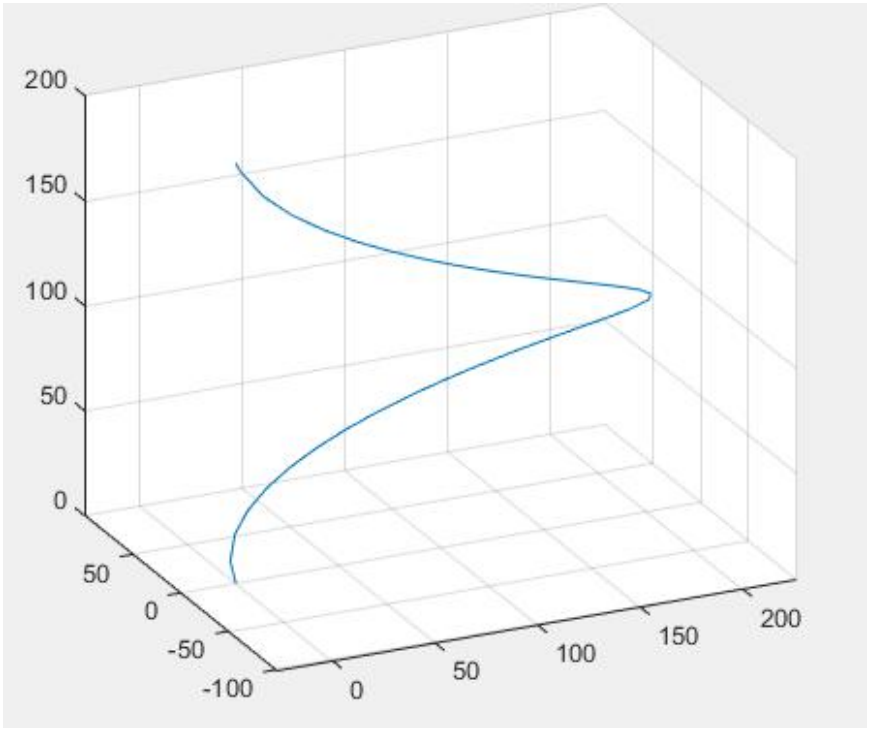
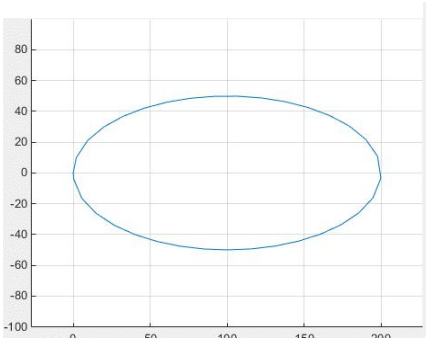
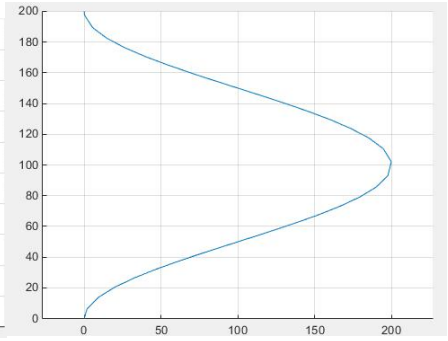
类型	多轴运动指令
描述	**椭圆插补，中心画弧，相对运动，可选螺旋。** BASE 第一轴和第二轴进行椭圆插补，相对移动方式，可选第三个轴同步螺旋。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 可画整椭圆。 只能画长轴（短轴）与 X 平行或垂直的椭圆。
语法	MECLIPSE (end1, end2, centre1, centre2, direction, adis, bdis[, end3]) end1：终点第一个轴运动坐标，相对于起始点 end2：终点第二个轴运动坐标，相对于起始点 centre1：中心第一个轴运动坐标，相对于起始点 centre2：中心第二个轴运动坐标，相对于起始点 direction：0-逆时针，1-顺时针 值 | 描述 0 | 逆时针 1 | 顺时针 adis：第一轴的椭圆半径，半长轴或者半短轴都可 bdis：第二轴的椭圆半径，半长轴或者半短轴都可，ab 相等时自动为圆弧或螺旋 end3：第三个轴的运动距离，需要螺旋时填入 坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	例一 不进行螺旋 RAPIDSTOP(2) WAIT IDLE(0) BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 SPEED=100,100,100 '主轴速度

ACCEL=1000 ,1000,1000 '主轴加速度
DECEL=1000 ,1000,1000
DPOS=0,0,0
TRIGGER '自动触发示波器
MECLIPSE(0,0,100,0,1,100,50) '中心(100,0)，终点(0,0)，半短轴 50，半长轴 100，顺时针画整椭圆，不进行螺旋

插补轨迹
DPOS(0)垂直刻度 100
DPOS(1)垂直刻度 100

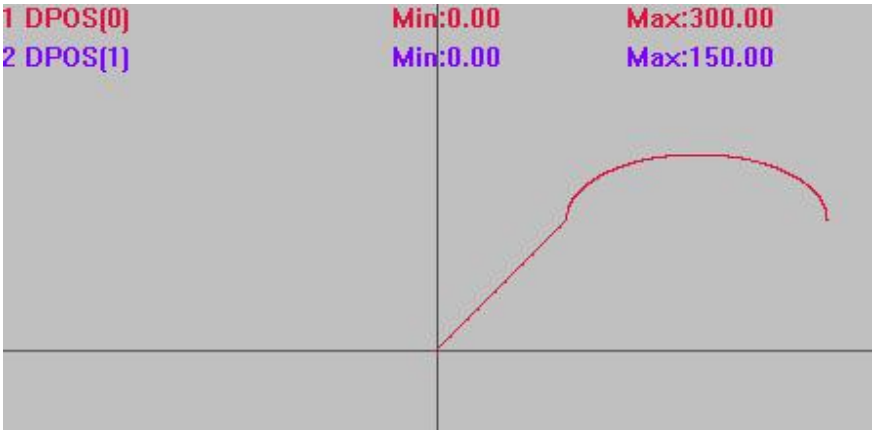
例二 进行螺旋
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0,1,2)
ATYPE=1,1,1 '设为脉冲轴类型
UNITS=100,100,100
SPEED=100,100,100 '主轴速度
ACCEL=1000 ,1000,1000 '主轴加速度
DECEL=1000 ,1000,1000
DPOS=0,0,0
TRIGGER '自动触发示波器
MECLIPSE(0,0,100,0,1,100,50,200) '中心(100,0)，终点(0,0)，半短轴 50，半长轴 100，顺时针画整椭圆圆弧，进行螺旋，第三轴运动距离 200

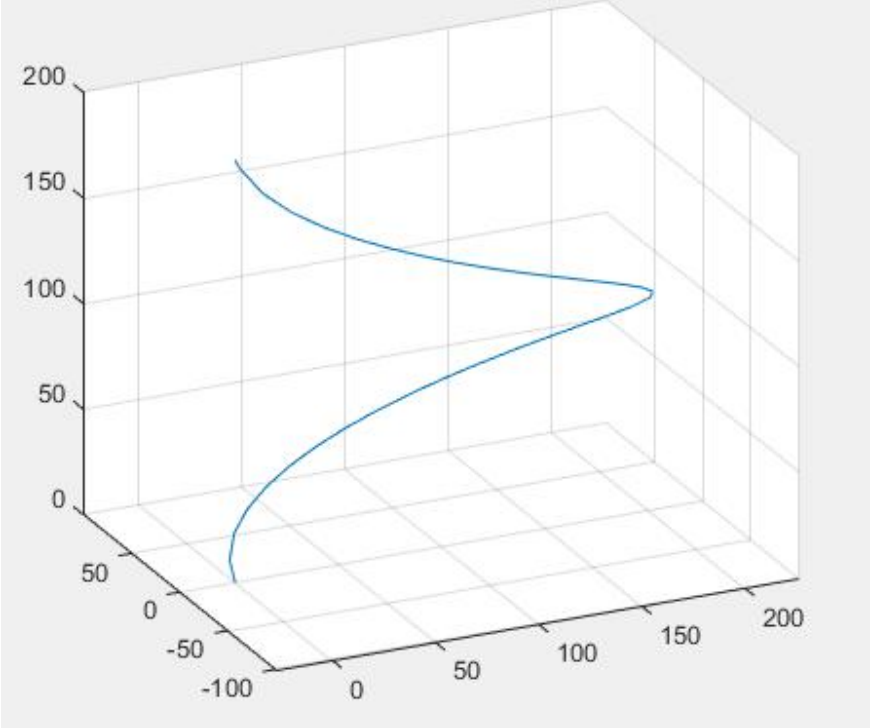
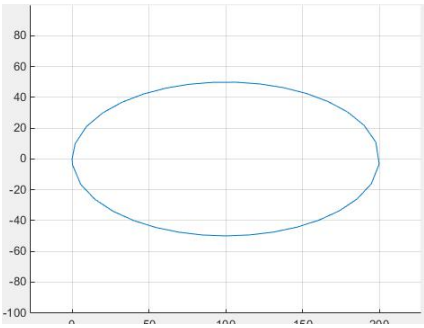
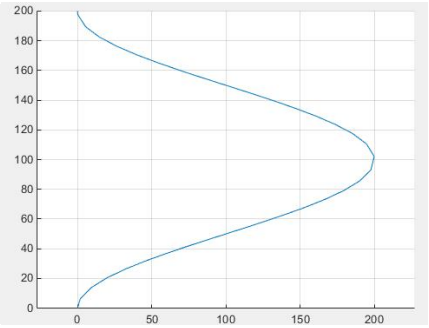
插补轨迹

	  
相关指令	MECLIPSEABS，*SP

MECLIPSEABS -- 椭圆-绝对

类型	多轴运动指令
描述	**椭圆插补，中心画弧，绝对运动，可选螺旋。** BASE 第一轴和第二轴进行椭圆插补，绝对移动方式，可选第三个轴同步螺旋。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 可画整椭圆。
语法	MECLIPSEABS(end1, end2, centre1, centre2, direction, adis, bdis[, end3]) end1：终点第一个轴运动坐标 end2：终点第二个轴运动坐标 centre1：中心第一个轴运动坐标 centre2：中心第二个轴运动坐标 direction：0-逆时针，1-顺时针

	值 | 描述 0 | 逆时针 1 | 顺时针 adis：第一轴的椭圆半径，半长轴或者半短轴都可 bdis：第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋 end3：第三个轴的运动坐标，需要螺旋时填入 坐标位置请确保正确，否则实际运动轨迹会错误。
适用控制器	通用
例子	例一 不进行螺旋 BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100 '主轴速度 ACCEL=1000 ,1000,1000 '主轴加速度 DECEL=1000 ,1000,1000 TRIGGER '自动触发示波器 MOVE(100,100) **MECLIPSEABS**(300,100,200,100,1,100,50) '中心(200,100)，终点(300,100)，半短轴 50，半长轴 100，画半椭圆圆弧，不进行螺旋 插补轨迹 DPOS(0)垂直刻度 150 DPOS(1)垂直刻度 150  例二 进行螺旋 BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100 '主轴速度 ACCEL=1000 ,1000,1000 '主轴加速度

	DECEL=1000 ,1000,1000 **MECLIPSEABS**(0,0,100,0,1,100,50,200) '中心(100,0)，终点(0,0)，半短轴 50，半长轴 100，画整椭圆，同时螺旋 插补轨迹   
相关指令	MECLIPSE，*SP

MSPHERICAL -- 空间圆弧

类型	多轴运动指令
描述	**空间圆弧插补运动，相对移动方式，可选螺旋。** 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MSPHERICAL(end1,end2,end3,centre1,centre2,centre3,mode[,distance4] [,distance5]) end1：第 1 个轴运动距离参数 1 end2：第 2 个轴运动距离参数 1 end3：第 3 个轴运动距离参数 1

centre1：第 1 个轴运动距离参数 2
centre2：第 2 个轴运动距离参数 2
centre3 ：第 3 个轴运动距离参数 2
mode：指定前面参数的意义

值	描述
0	当前点，中间点，终点三点定圆弧 距离参数 1 为终点距离，距离参数 2 为中间点距离
1	当前点，圆心，终点定圆弧 走最短的圆弧 距离参数 1 为终点距离，距离参数 2 为圆心的距离
2	当前点，中间点，终点三点定圆 距离参数 1 为终点距离，距离参数 2 为中间点距离
3	当前点，圆心，终点定圆 先走最短的圆弧，再继续走完整圆 距离参数 1 为终点距离，距离参数 2 为圆心的距离

distane4：第四轴螺旋的功能，指定第 4 轴的相对距离，此轴不参与速度计算
distane5：第五轴螺旋的功能，指定第 5 轴的相对距离，此轴不参与速度计算

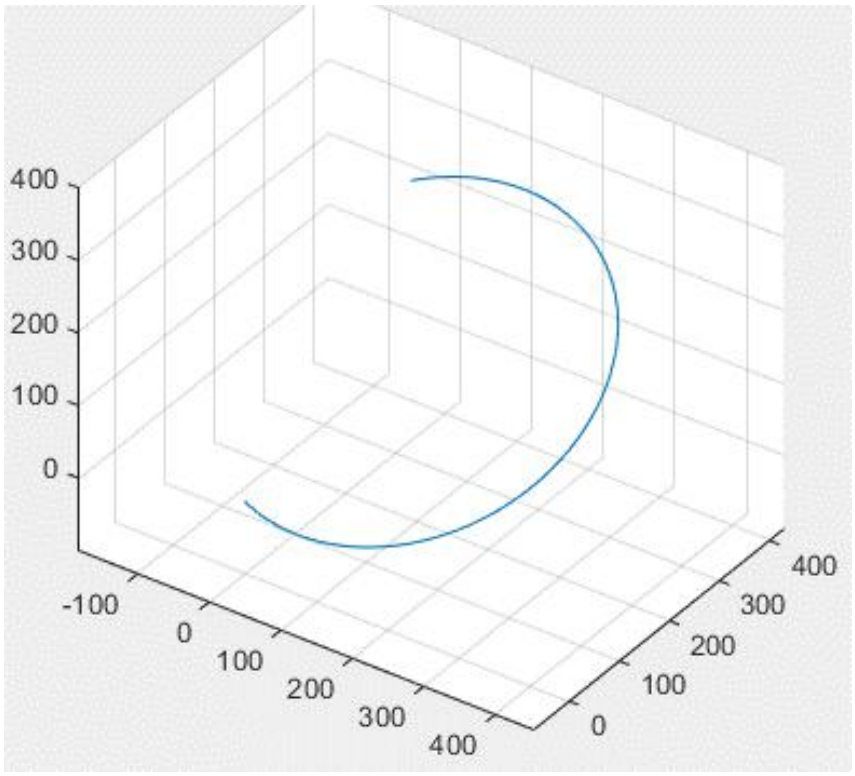
坐标位置请确保正确，否则实际运动轨迹会错误。

适用控制器

通用

例子

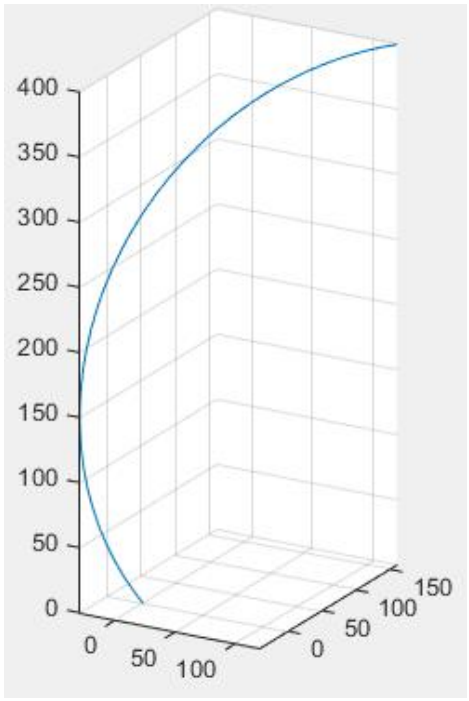
```
BASE(0,1,2)
ATYPE=1,1,1                '设为脉冲轴类型
UNITS=100,100,100
DPOS=0,0,0
SPEED=100,100,100          '主轴速度
ACCEL=1000 ,1000,1000      '主轴加速度
DECEL=1000 ,1000,1000

使用圆心为(120,160,150)，半径 250 的圆演示 4 种 mode 的运动轨迹
mode 0：
MSPHERICAL(120,160,400,240,320,300,0) '终点(120,160,400)，中间点(240,320,300)，模
                                      式 0，三点画弧

插补轨迹
```



mode 1：

MSPHERICAL(120,160,400,120,160,150,1) '相对位置，终点(120,160,400)，圆心(120,160,150)，模式 1，走最短的圆弧

插补轨迹

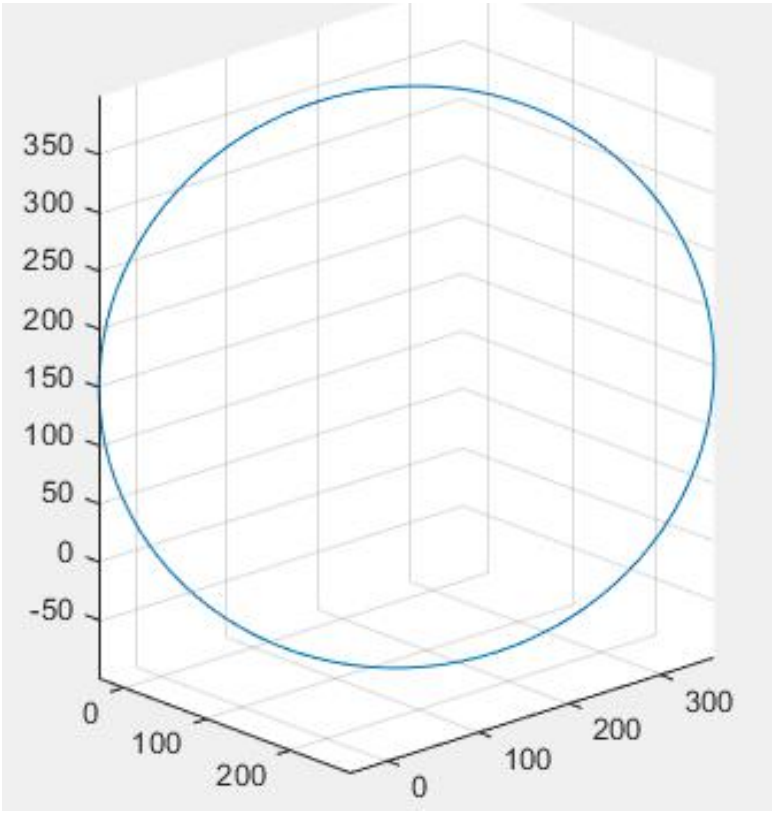


mode 2：

MSPHERICAL(120,160,400,240,320,300,2)'终点(120,160,400)，中间点(240,320,300)，模

式 2，三点画圆

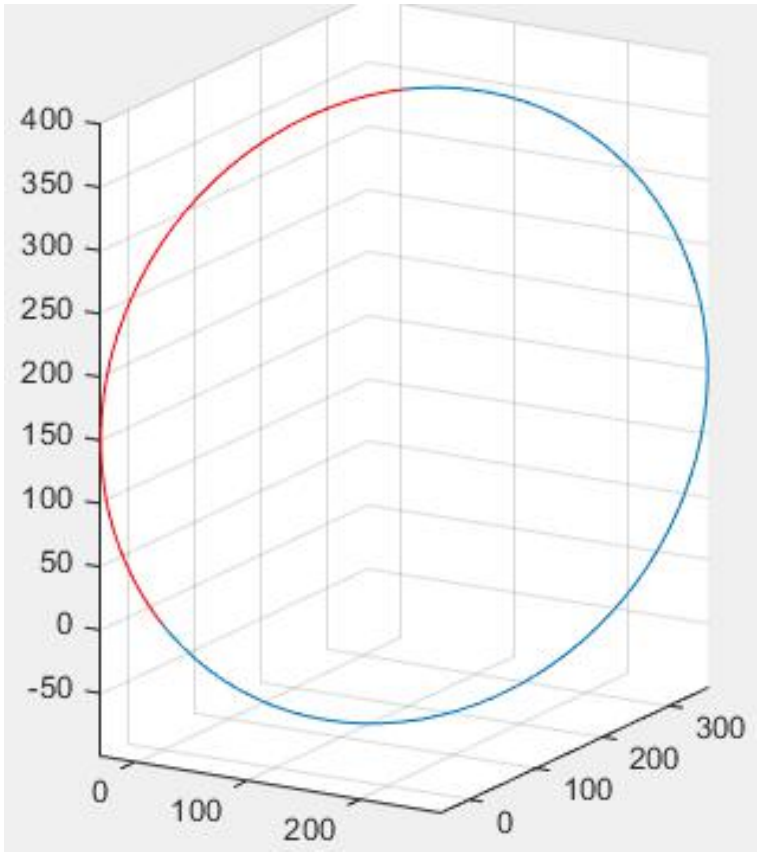
插补轨迹



mode 3：

MSPHERICAL(120,160,400,120,160,150,3) '终点(120,160,400)，圆心(120,160,150)，模式 3，先走最短的圆弧(红色部分)，再走完整圆

插补轨迹

MOVESPIRAL -- 渐开线圆弧

类型	多轴运动指令
描述	**渐开线圆弧插补运动，相对移动方式，可选螺旋。** 当前点和圆心距离确定起始半径，当起始半径 0 时角度无法确定，直接从 0 角度开始。见例一。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MOVESPIRAL(centre1,centre2,circles,pitch[,distance3][,distance4]) centre1：圆心的第 1 轴相对距离 centre2：圆心的第 2 轴相对距离 circles：要旋转的圈数，可以为小数圈，负数表示顺时针，每圈终点位置为起点和圆心连线上的一点，见例二 pitch：每圈的扩散距离，可以为负 distance3：第 3 轴螺旋的功能，指定第 3 轴的相对距离，此轴不参与速度计算 distance4：第 4 轴螺旋的功能，指定第 4 轴的相对距离，此轴不参与速度计算
适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1　　'设为脉冲轴类型 UNITS=100,100,100

```
DPOS=0,0,0
SPEED=100,100,100          '主轴速度
ACCEL=1000 ,1000,1000      '主轴加速度
DECEL=1000 ,1000,1000
TRIGGER                    '自动触发示波器
```

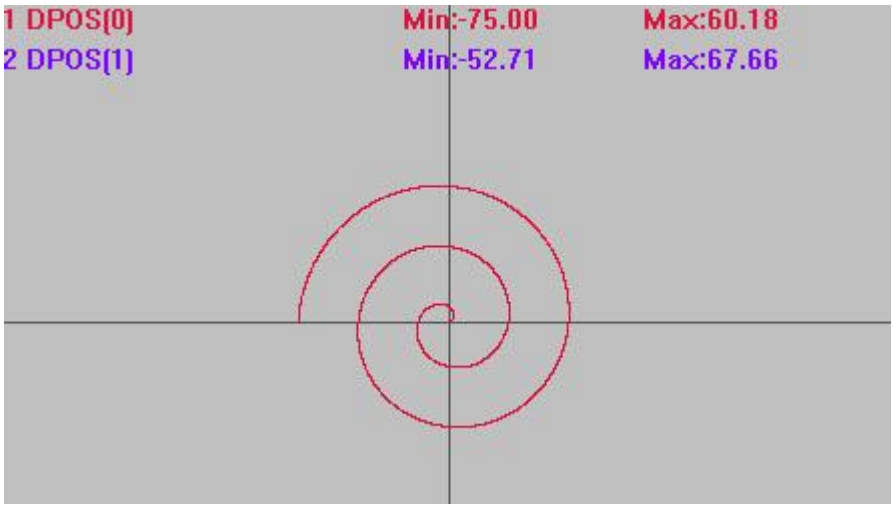
例一 从起点扩散

```
MOVESPIRAL(0,0,2.5,30)   '此时以起始位置为中心，逆时针旋转 2.5 圈，每圈扩散 30
```

插补轨迹

DPOS(0)垂直刻度 100

DPOS(1)垂直刻度 100



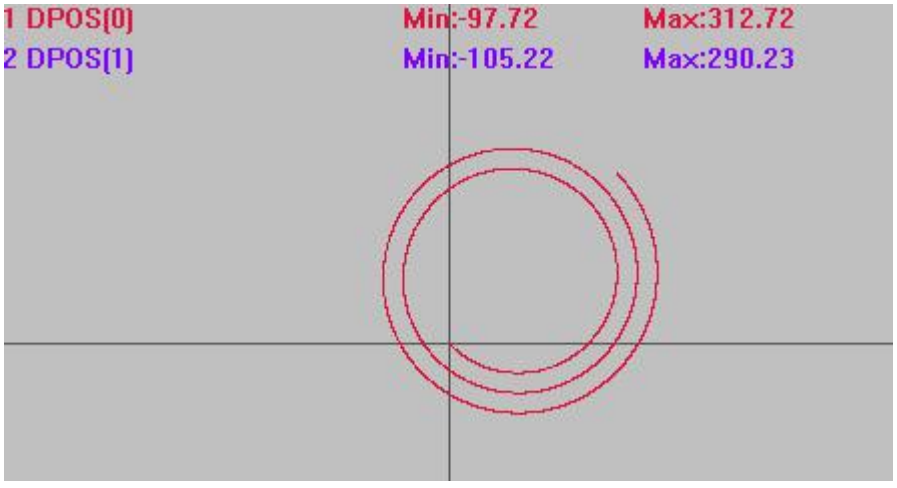
例二 不螺旋

```
MOVESPIRAL (100,100,2.5,30)   '起始半径 100，以(100,100)为圆心，逆时针旋转 2.5 圈，每圈向外扩散 30
```

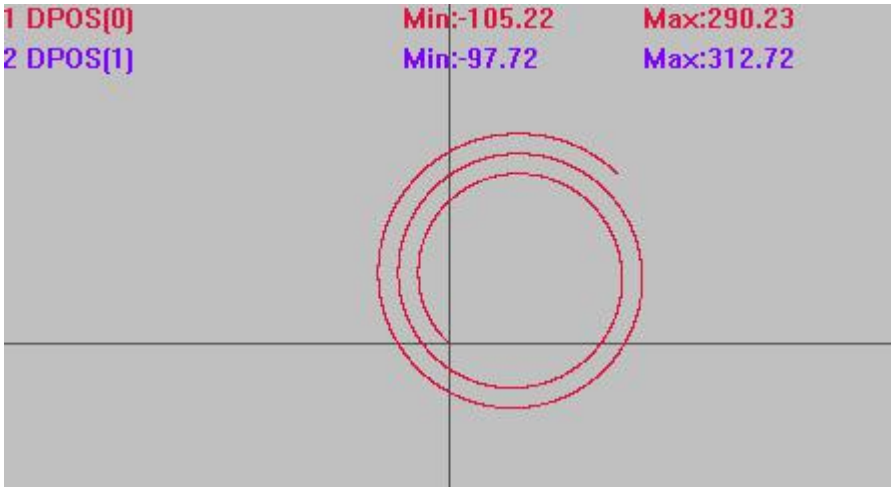
插补轨迹（若轨迹圈数显示不全，将采样间隔适当调大）

DPOS(0)垂直刻度 300

DPOS(1)垂直刻度 300



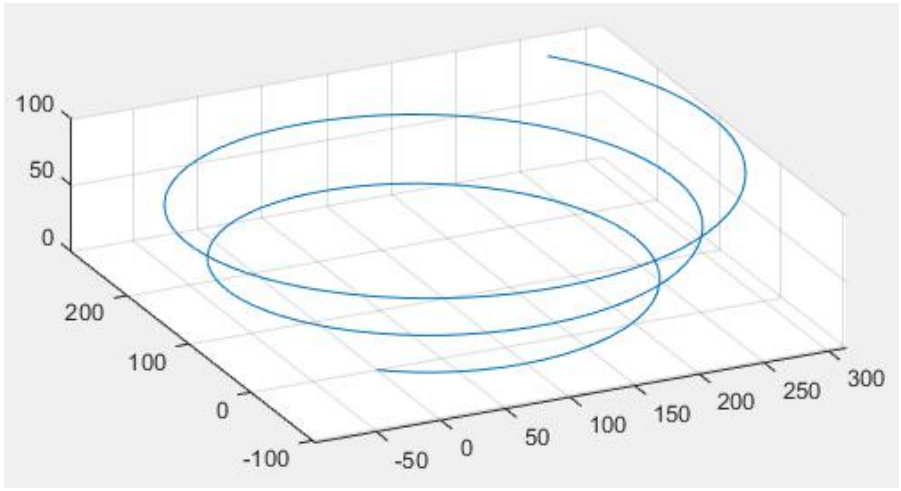
MOVESPIRAL (100,100,-2.5,30)　'旋转圈数为负数时(-2.5)，顺时针旋转



例三 螺旋

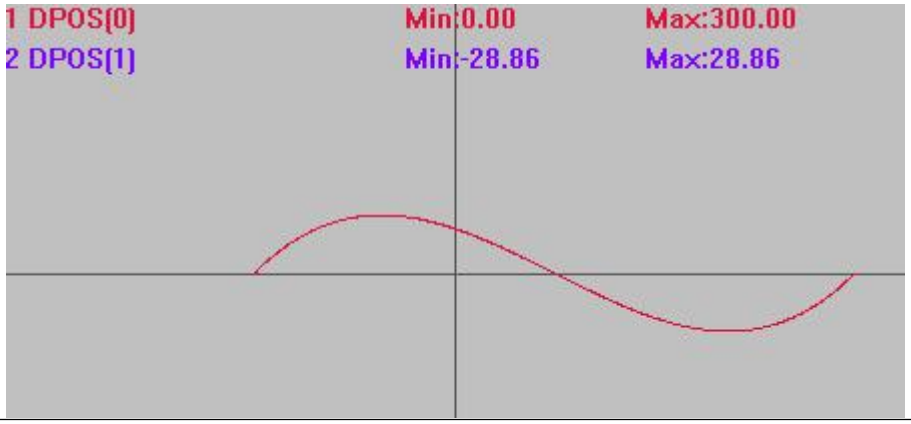
MOVESPIRAL(100,100,2.5,30,100)　'起始半径 100，以(100,100)为圆心，逆时针旋转 2.5 圈，每圈向外扩散 30，同时 Z 轴向上运动到 100

插补轨迹



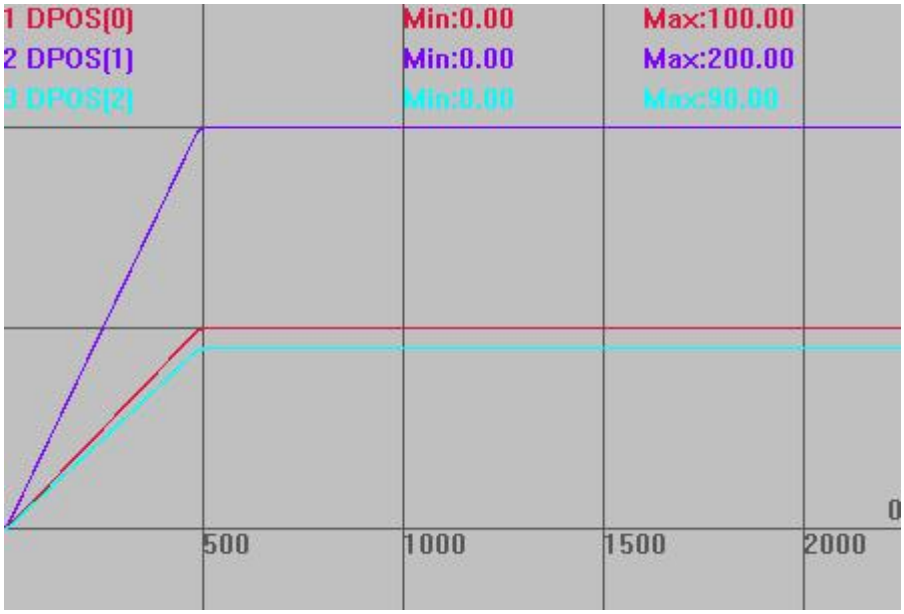
MOVESPLINE / MOVESPLINEABS -- 样条插补

类型	特殊运动指令
描述	**样条插补运动，相对/绝对运动。** 样条运动的控制点位置提前写入 TABLE 数组。 此运动没有 SP 指令，自定义速度的连续插补运动通过 CORNER_MODE 的 BIT8 来设置。
语法	MOVESPLINE (axes,mode ,dtendcontrol4, dtcontrol2, dtcontrol3) axes：插补的轴个数 mode：模式，0-使用 3 阶贝塞尔样条

	dtendcontrol4：第 4 个控制点存储 table 索引，对贝塞尔曲线就是终点 dtcontrol2：第 2 个控制点存储 table 索引 dtcontrol3：第 3 个控制点存储 table 索引 对贝塞尔曲线，第 1 个控制点就是当前位置。
适用控制器	4 系列 170507 以上固件支持，306x 系列 161208 以上固件支持
例子	BASE(0,1) DPOS=0,0 ATYPE=1,1　　'设为脉冲轴类型 SPEED=100,100　　'主轴速度 ACCEL=1000,1000　　'主轴加速度 DECEL=1000,1000 TRIGGER CORNER_MODE=2 + 256　'设置 SP 运动，使用 FORCE_SPEED FORCE_SPEED=100 TABLE(0, 100, 100)　'TABLE(0)(1)存储第二个控制点，相对起点距离 TABLE(10, 200, -100)　'TABLE(10)(11)存储第三个控制点，相对起点距离 TABLE(20, 300, 0)　'TABLE(20)(21)存储第四个控制点，终点距离 **MOVESPLINE**(2, 0, 20, 0, 10)　'2 轴相对样条插补 插补轨迹 DPOS(0)垂直刻度 100，偏移-100 DPOS(1)垂直刻度 100，无偏移 
相关指令	CORNER_MODE

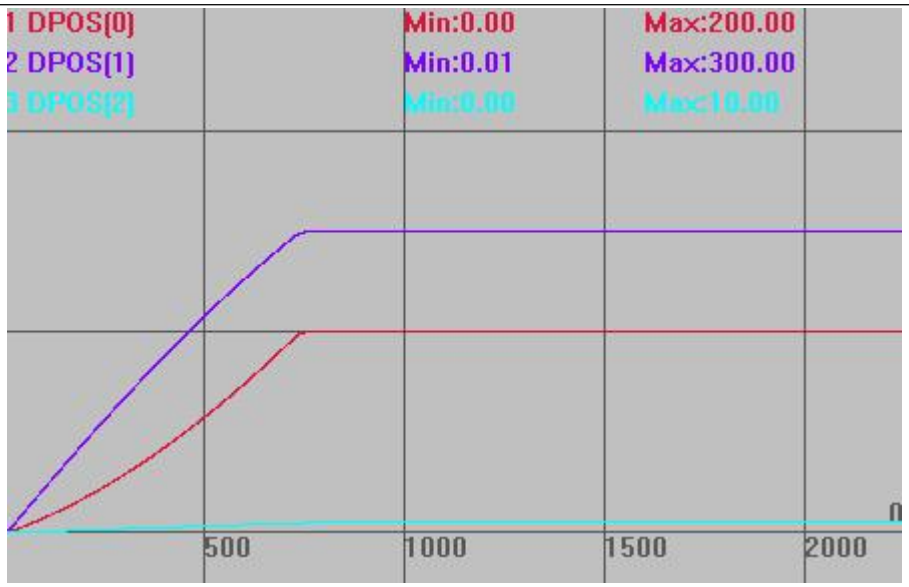
MOVE_TURNABS -- 旋转台插补

类型	多轴运动指令
描述	**旋转的插补运动，保证在旋转台上是直线运动。** 旋转功能是指工作平台在与 XY 平行的平面上旋转，旋转的正向与 XY 的正向要一致（右手法则）。 旋转的参数存储在 TABLE 表里面，依次存储 R 轴号，R 轴一圈脉冲数，X 轴号，Y 轴

	号，X 圆心，Y 圆心。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 建议直接使用机械手的旋转台算法，详情查看《正运动机械手指令说明》的 frame11/17。
语法	MOVE_TURNABS(tablenum,position1[,position2[,position3[,position4...]]]) tablenum：存储旋转参数的 table 编号 position1：第一个轴运动坐标 position2：下一个轴运动坐标
适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100 '主轴速度 ACCEL=1000 ,1000,1000 '主轴加速度 DECEL=1000 ,1000,1000 '主轴减速度 TABLE(0, 3, 3600, 0,1, 0,0) '设置旋转台的参数 TRIGGER **MOVE_TURNABS**(0,100,200,90) '直线插补至目标位置 WAIT IDLE '等待运动停止 插补轨迹 DPOS(0)垂直刻度 100 DPOS(1)垂直刻度 100 DPOS(2)垂直刻度 100 
相关指令	MCIRC_TURNABS

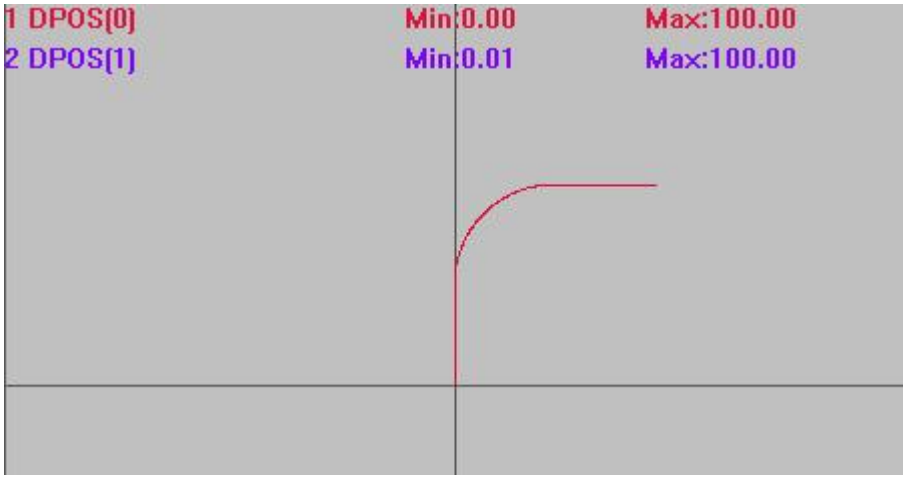
MCIRC_TURNABS -- 旋转台插补-绝对

类型	多轴运动指令
描述	**旋转的插补运动，保证在旋转台上是圆弧运动。** 旋转功能是指工作平台在与 XY 平行的平面上旋转，旋转的正向与 XY 的正向要一致（右手法则）。 旋转的参数存储在 TABLE 表里面，依次存储 R 轴号，R 轴一圈脉冲数，X 轴号，Y 轴号， X 圆心，Y 圆心。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MCIRC_TURNABS(tablenum, refpos1, refpos2, mode,end1, end2 [, dis3, dis4, dis5]) tablenum：存储旋转参数的 table 编号 refpos1：第一个轴参考点，绝对位置 refpos2：第二个轴参考点，绝对位置 mode：1-参考点是当前点前面，2-参考点是结束点后面，3-参考点在中间，采用三点定圆的方式 end1：第一个轴结束点，绝对位置 end2：第二个轴结束点，绝对位置 dis3：螺旋轴的结束位置
适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100 '主轴速度 ACCEL=1000 ,1000,1000 '主轴加速度 DECEL=1000 ,1000,1000 '主轴减速度 TABLE(0, 3, 3600, 0,1, 0,0) '设置旋转台的参数 TRIGGER TURN_POSMAKE(0,100,200,5,10) **MCIRC_TURNABS**(0,TABLE(10),TABLE(11),3,200,300,10) '圆弧的同时 3 轴旋转 WAIT IDLE 插补轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200 DPOS(2)垂直刻度 200

	
相关指令	MOVE_TURNABS，TURN_POSMAKE

MOVESMOOTH -- 倒圆角

类型	多轴运动指令
描述	**空间直线倒圆运动。** 根据下一个直线运动的绝对坐标在拐角自动插入圆弧，加入圆弧后会使得运动的终点与直线的终点不一致，拐角过大时不会插入圆弧，当距离不够时会自动减小半径。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。 此指令为早期开发指令，有轴数限制，建议使用 CORNER_MODE 指令，功能更多。
语法	MOVESMOOTH (end1, end2, end3, next1, next2, next3, radius) end1：第 1 个轴运动绝对坐标 end2：第 2 个轴运动绝对坐标 end3：第 3 个轴运动绝对坐标 next1：第 1 个轴下一个直线运动绝对坐标 next2：第 2 个轴下一个直线运动绝对坐标 next3：第 3 个轴下一个直线运动绝对坐标 radius：插入圆弧的半径，当过大的时候自动缩小
适用控制器	通用
例子	BASE(0,1,2) ATYPE=1,1,1 '设为脉冲轴类型 UNITS=100,100,100 DPOS=0,0,0 SPEED=100,100,100 '主轴速度 ACCEL=1000 ,1000,1000 '主轴加速度 DECEL=1000 ,1000,1000 '主轴减速度 TRIGGER '自动触发示波器 **MOVESMOOTH** (0,100,0,100,100,0,50) '加入圆弧后，实际运动到了'(50,100,0)

	MOVEABS(100,100,0) 插补轨迹 DPOS(0)垂直刻度 100 DPOS(1)垂直刻度 100 

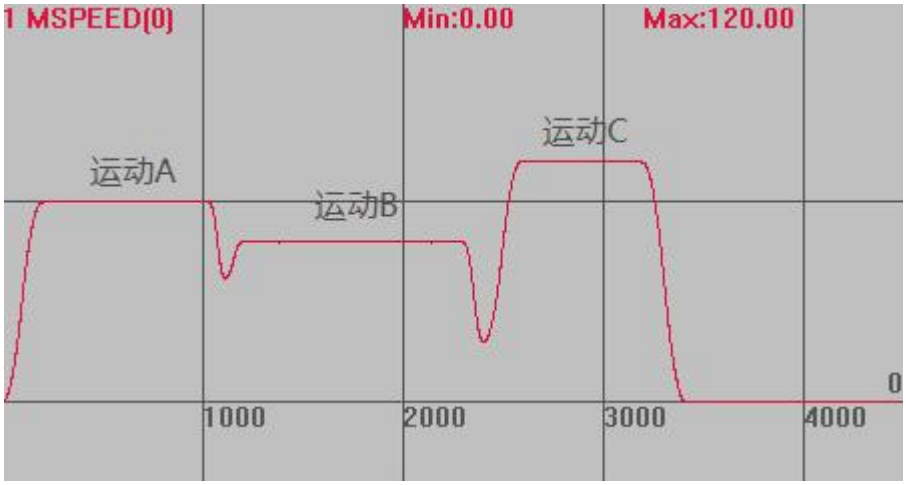
*SP -- 运动单独速度

类型	多轴运动指令
描述	**用于设置每段运动的运行速度、起始速度和终止速度。** 多轴运动指令有对应 SP 运动指令，此时可以使用轴参数 FORCE_SPEED 、STRATMOVE_SPEED 和 ENDMOVE_SPEED 来设置每个运动的速度、起始速度和终止速度，当不需要设置每个运动速度时，不需要使用 SP 指令。
语法	SP 指令有： MOVESP， MOVEABSSP， MOVECIRCSP, MOVECIRCABSSP，MHELICALSP ， MHELICALABSSP ， MECLIPSESP ， MECLIPSEABSSP ，MSPHERICALSP。 FORCE_SPEED 、ENDMOVE_SPEED 和 STRATMOVE_SPEED 会随 SP 运动指令写入运动缓存区。
适用控制器	通用
例子	例一 BASE(0) ATYPE=1 UNITS=100 DPOS=0 ACCEL=1000 DECEL=1000 SRAMP=100 MERGE=ON '打开连续插补

```
SPEED=100                    '运行速度 100
FORCE_SPEED=80               '限制速度 80
STARTMOVE_SPEED=60           '起始速度 60
ENDMOVE_SPEED=30             '终止速度 30
TRIGGER                      '自动触发示波器
MOVE(100)                    '运动 A，不使用 SP 限速
MOVESP(100)                  '运动 B，使用 SP 限速
FORCE_SPEED=120              '限制速度 120
ENDMOVE_SPEED=30             '终止速度 30
MOVESP(100)                  '运动 C，使用 SP 限速
```

速度轨迹

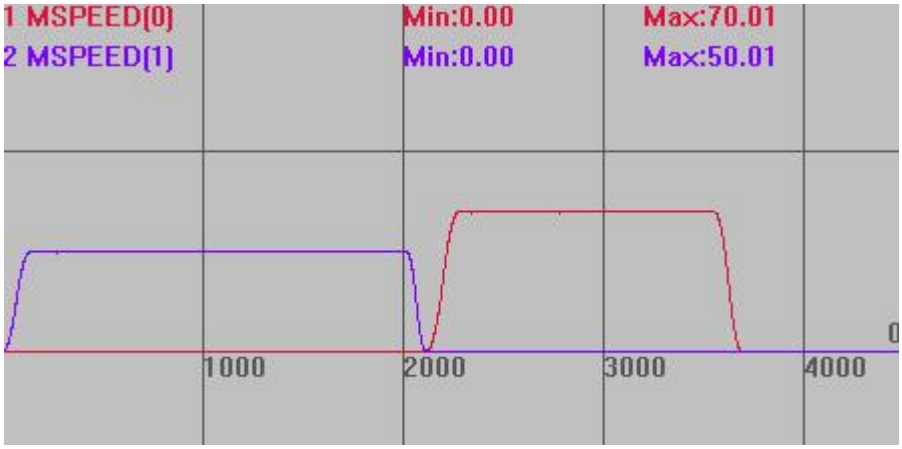
MSPEED(0)垂直刻度 100



不使用 SP 指令时，运行速度为 SPEED，使用 SP 指令后，运行速度为 FORCE_SPEED。当 STARTMOVE_SPEED 和 ENDMOVE_SPEED 同时设置时，STARTMOVE_SPEED 生效。

例二

```
BASE(0,1)
DPOS=0,0
UNITS=100,100
ACCEL = 1000 ,1000
DECEL = 1000 ,1000
MERGE =ON
CORNER_MODE=2+8          '启动自动拐角减速与小圆限速
DECEL_ANGLE = 15 * (PI/180)        '设置开始减速角度
STOP_ANGLE = 45 * (PI/180)         '设置结束减速角度
FULL_SP_RADIUS = 5               '设置小圆限速最大半径
SPEED = 1000 ,1000
STARTMOVE_SPEED = 1000           '设置一个较大的起始速度
ENDMOVE_SPEED = 1000         '设置一个较大的结束速度
FORCE_SPEED= 50       '每段速度受 speed 限制，将 speed 设置为一个较大值即可
```

	TRIGGER '自动触发示波器 **MOVESP**(0,100) '该段速度为 50units/s FORCE_SPEED = 70 '每段速度受 speed 限制，将 speed 设置为一个较大值即可 **MOVESP**(100,0) '该段速度为 70units/s 速度轨迹 MSPEED(0)垂直刻度 100 MSPEED(1)垂直刻度 100 
相关指令	FORCE_SPEED，ENDMOVE_SPEED，STRATMOVE_SPEED

6.3 特殊运动指令

MOVE_PAUSE -- 运动暂停

类型	特殊运动指令
描述	**BASE 轴运动暂停。** 只有在单轴或多轴插补运动时有效，多轴联动时一起暂停。 可以通过 AXISSTATUS 来查询是否有暂停。 当轴已经暂停或不在运动中时，调用这个指令会有警告输出，但不影响程序运行。某些运动不支持暂停，如 VMOVE、同步运动指令等。
语法	MOVE_PAUSE (mode) mode：暂停方式

0（缺省）	暂停当前运动。
1	在当前运动完成后正准备执行下一条运动指令时暂停
2	在当前运动完成后正准备执行下一条运动指令时，并且两条指令的 MARK 标识不一样时暂停 这个模式可以用于一个动作由多个指令来实现时，可以在一整个动作完成后暂停
3	强制暂停，IDLE 模式下也可以进入暂停状态 20170513 固件版本增加此功能

适用控制器	通用

例子	BASE(0) DPOS=0 SPEED=100 例一　mode 0 MOVE(1000)　　'当前运动 MOVEABS(-100)　　'缓冲运动 **MOVE_PAUSE**(0)　　'模式 0，暂停当前运动 ?DPOS(0)　　'打印结果，0 '此时当前运动只运行了极短时间，扫描到 MOVE_PAUSE 时直接暂停 例二　mode 1 MOVE(1000)　　'当前运动 MOVEABS(-100)　　'缓冲运动 **MOVE_PAUSE**(1)　　'模式 1，先完成当前运动再暂停 WAIT IDLE ?DPOS(0)　　'打印结果，1000 '此时运行完当前运动，DPOS 为 1000 例三　mode 2 MOVE_MARK=1　　'标号手动设为 1 MOVE(200)　　'当前运动 MOVE_MARK=1　　'标号设为与上一个运动一样 MOVEABS(-100)　　'缓冲运动 MOVEABS(100)　　'不设置运动标号，自动加一 **MOVE_PAUSE**(2)　　'模式 2，先完成当前运动，然后直到下一条运动指令的标号与当前标号不一致时再暂停 DELAY(3000)　　'等待暂停后 ?DPOS(0)　　'打印结果，-100，（速度过慢会导致打印时当前运动还在进行，导致结果大于-100） '此时运行完当前运动，下一条运动的标号被手动设置成相同的，继续执行，直到第 3 条运动标号不一致，暂停
相关指令	MOVE_MARK，MOVE_RESUME，AXISSTATUS

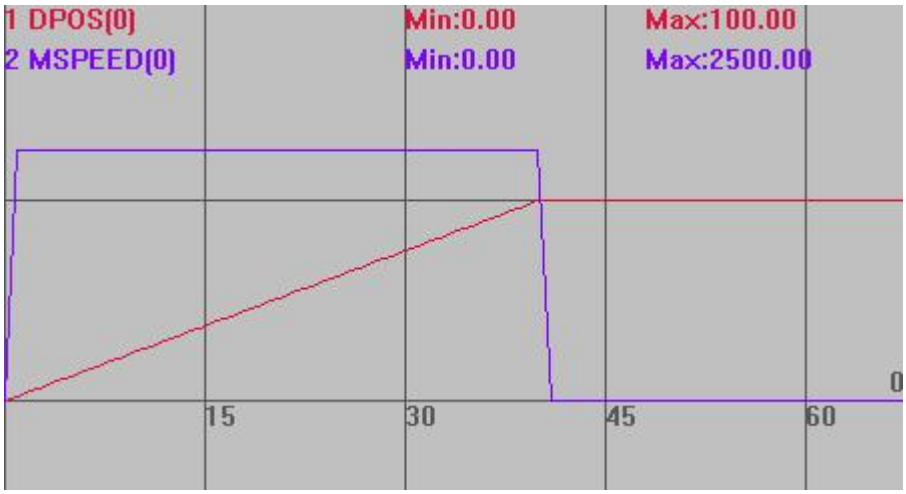
MOVE_RESUME -- 运动恢复

类型	特殊运动指令
描述	**当 BASE 轴暂停暂停时，从暂停处继续运动。** 可以通过 AXISSTATUS 来查询是否有暂停。
语法	MOVE_RESUME
适用控制器	通用
例子	BASE(0)

	UNITS=100 DPOS=0 SPEED=100 ACCEL=1000 DECEL=1000 MOVE(100) '当前运动 MOVE(100) '缓冲运动 MOVE_PAUSE(1) '当前运动完成后暂停 WA 2000 '等待当前运动完成 ?DPOS(0) '打印结果，100 DELAY(1000) **MOVE_RESUME** '继续运行 WAIT IDLE ?DPOS(0) '打印结果，200
相关指令	MOVE_PAUSE，AXISSTATUS

MOVE_PT -- 单位时间距离

类型	特殊运动指令
描述	**在一段时间内驱动电机运动设置的距离。** 一般是 PC 每个周期计算好对应的坐标，然后传给控制器。 运动时的速度=(DIS/TICKS)*1000 units/s。 不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送。
语法	MOVE_PT (ticks, dis1,dis2…) ticks：时间的长度。ticks 会自己不断减 1，1ticks 大约 1ms dis1：运动的距离 SERVO_PERIOD 为 1000us 的控制器 1 TICKS 为 1ms(不同 SERVO_PERIOD 的 TICKS 时间不一样)
适用控制器	通用
例子	BASE(0) UNITS=100 DPOS = 0 SPEED=100 ACCEL=1000 DECEL=1000 TRIGGER '自动触发示波器 FOR i=0 TO 9 **MOVE_PT** (4, 10) '在 4tick 内，运动 10units，速度=2500 units/s NEXT WAIT IDLE

	PRINT *DPOS '打印结果，100 插补速度 DPOS(0)垂直刻度 100 MSPEED(0)垂直刻度 2000 
相关指令	MOVE_PTABS

MOVE_PTABS -- 单位时间距离绝对

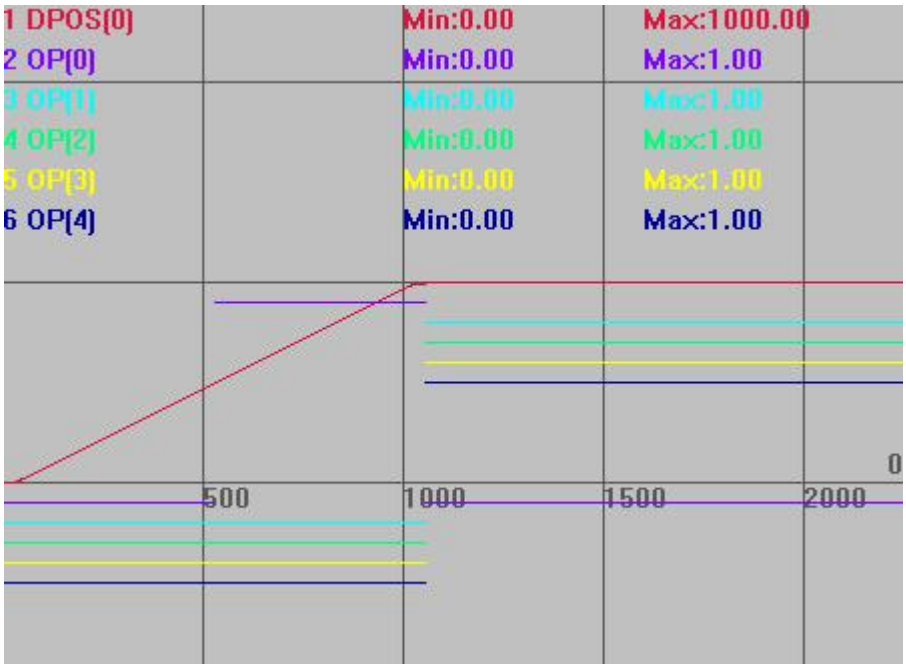
类型	特殊运动指令
描述	**在一段时间内驱动电机运动到某个位置。** 一般是 PC 每个周期计算好对应的坐标，然后传给控制器。 运动时的速度=(DIS/TICKS)*1000 units/s。 不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送。
语法	MOVE_PTABS(ticks,dis1,dis2,…) ticks：时间的长度。ticks 会自己不断减 1，1ticks 大约 1ms dis1：运动的绝对位置
适用控制器	通用
例子	BASE(0,1) DPOS=0,0 **MOVE_PTABS** (3, 20,20) '3tick 时间内运动到到 20,20 WAIT IDLE PRINT *DPOS '打印结果，20,20
相关指令	MOVE_PT

MOVE_OP -- 缓冲输出

类型	特殊运动指令

描述	**BASE 轴运动缓冲加入一个输出口操作。** 这个指令 LOAD 执行时不做任何运动，只操作输出口。语法同 OP 指令。 普通模式，误差在一个扫描周期，所有控制器都可使用。 精准位置输出模式，误差 1 微秒以内，4 系列产品，20170421 以上版本支持。 1. 只有支持硬件比较输出的 OP 口才支持精准输出功能。 2. 每个生效的精准输出 MOVE_OP 需要间隔 1 个周期时间才能继续生效，此间隔内新的 MOVE_OP 自动使用普通方式，超过此间隔，新 MOVE_OP 可继续生效。连续的 MOVE_OP，因为没有间隔时间，只有第一个生效。（某些控制器可以多个精准输出同时触发，具体查看控制器硬件手册说明，例如 ZMC420SCAN 前 8 个输出口支持精准输出，而且每个输出口可以同时使用精准输出） 3. 在控制器支持 OP 口独立的情况下，不同 OP 口就算多个轴 MOVE_OP 精准输出也不冲突，在 OP 口精准输出功能不独立的情况，同时使用精准功能可能冲突。 4. MOVE_OP 精准功能基于 BASE 主轴，多个轴插补时，与 BASE 主轴 ATYPE 类型不一样的从轴，无法保证精准位置输出。
语法	语法一：MOVE_OP ([ionum],value) ionum：输出编号，从 0 开始 value：输出状态，多个输出口操作时按位来指明多个口状态 语法二：MOVE_OP (ionum1, ionum2,value[,mask]) ionum1：要操作的第一个输出通道 ionum2：要操作的最后一个输出通道 value：输出状态，多个输出口操作时按位来指明多个口状态 mask：按位来设置值，指定哪些 IO 需要操作，不填时从第一个通道到最后一个通道都操作
适用控制器	通用
例子	普通模式使用 BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=1000 DECEL=1000 TRIGGER '自动触发示波器 MOVE(500) **MOVE_OP** (0,ON) '等待上条运动完成后，OUT0 口输出信号 MOVE(500) **MOVE_OP** (0,OFF) '等待上条运动完成后，OUT0 口关闭信号 **MOVE_OP**(1,4,15) 'OUT1-4 口输出信号，15 对应二进制 1111 轨迹曲线，为方便查看，进行了竖直偏移 DPOS(0)垂直刻度 1000 OP(0)垂直刻度 1,偏移-0.1 OP(1)垂直刻度 1,偏移-0.2

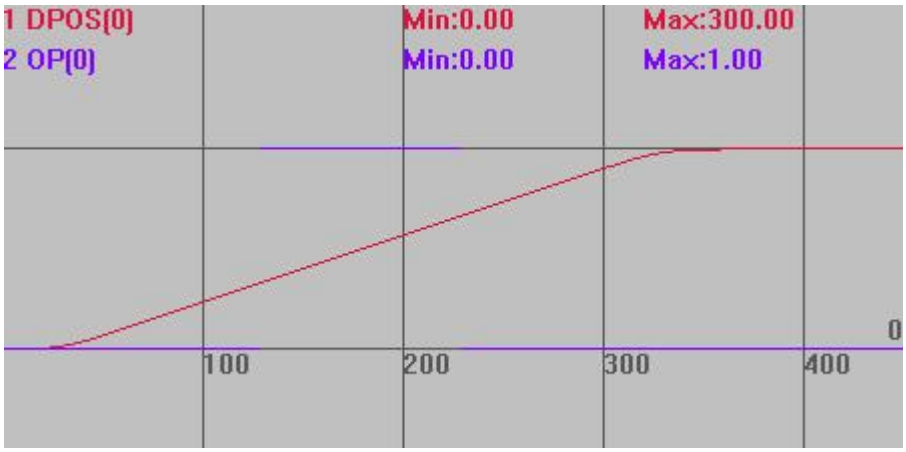
OP(2)垂直刻度 1,偏移-0.3
OP(3)垂直刻度 1,偏移-0.4
OP(4)垂直刻度 1,偏移-0.5



精准位置输出模式
例一

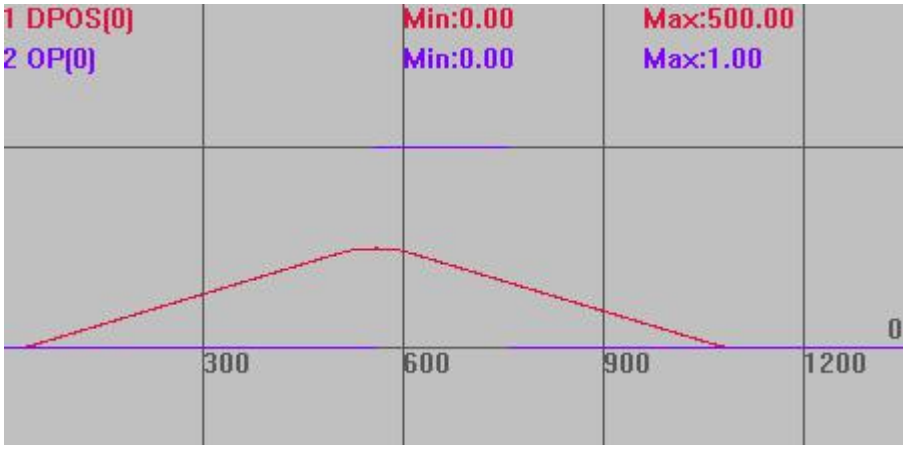
```
?"ZSET:",SYSTEM_ZSET
BASE(0)
UNITS=100
DPOS=0
SPEED=200
ACCEL=1000
DECEL=1000
TRIGGER                    '自动触发示波器
ATYPE=1
MERGE=1
MOVE(100)
MOVE_OP(0,1)   '精准生效
MOVE(100)      '超过 2MS 时间，下个 MOVE_OP 可继续精准生效
MOVE_OP(0,0)   '精准生效
MOVE(100)
```

轨迹曲线
DPOS(0)垂直刻度 300
OP(0)垂直刻度 1

	 例二 20170505 以上固件版本支持 AXIS_ZSET = 3+16 'BIT4 支持编码器精确位置 BASE(0) ATYPE= 4 '轴使用脉冲加编码器类型，必须实际有接编码器线 DPOS=0 MPOS=0 UNITS=1000 SPEED= 1000 ACCEL = 1000 MERGE=1 TRIGGER OPNUM = 0 MOVEOP_DELAY = 1.5 '实际输出时间延迟 1.5ms OP(opnum,0) '初始化 OP HW_TIMER(2, 10000, 5000, 1, 0, opnum) '输出变为 on 后, 5000us 后切换为 off MOVE(200) **MOVE_OP**(opnum,1) '输出 on 后靠 HW_TIMER 来延时关闭 MOVE(100) **MOVE_OP**(opnum,1) MOVE(100)
相关指令	OP，MOVE_OP2 精确位置输出模式 SYSTEM_ZSET，AXIS_ZSET

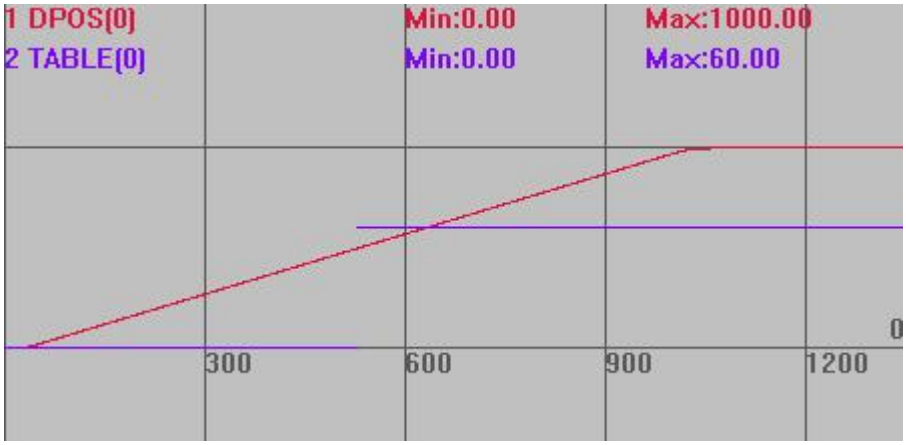
MOVE_OP2 -- 缓冲输出 2

类型	特殊运动指令
描述	**BASE 轴运动缓冲加入一个输出口操作，指定时间后输出状态翻转。** 这个指令 LOAD 执行时不做任何运动，只操作输出口。

	单个轴同一时间只支持一个脉冲输出，第二个 MOVE_OP2 指令会自动关闭前面指令的脉冲。
语法	MOVE_OP2(ionum,state,offtimems) ionum：输出编号，从 0 开始 state：输出状态 offtimems：多少 ms 时间后翻转，以产生脉冲输出的效果
适用控制器	通用
例子	BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=1000 DECEL=1000 OP(0,OFF)　　'关闭 OUT0 口 TRIGGER　　'自动触发示波器 MOVE(500) **MOVE_OP2** (0,ON,1000)　'等待上条运动完成，输出口 0 输出一个 1s 的脉冲，脉冲时间不会阻碍下一条运动执行 MOVE(-500) 轨迹曲线 DPOS(0)垂直刻度 1000 OP(0)垂直刻度 1 
相关指令	MOVE_OP，OP

MOVE_TABLE -- 缓冲 Table

类型	特殊运动指令
描述	**BASE 轴运动缓冲加入一个 TABLE。** 这个指令 LOAD 执行时不做任何运动，只修改 TABLE。此指令的 MTYPE 与 MOVE_OP 一致。

语法	MOVE_TABLE(tablenum, value) tablenum：table 编号 value：要修改的值
适用控制器	通用
例子	BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=1000 DECEL=1000 TABLE(0)=0 '先令 TABLE(0)=0 ?TABLE(0) '打印确认 TABLE(0)的值 '打印结果，0 TRIGGER '自动触发示波器 MOVE(500) **MOVE_TABLE**(0, 60) '等待运动完成后，TABLE(0)赋值 60 MOVE(500) WAIT IDLE ?TABLE(0) '打印修改后 TABLE(0)的值，打印结果，60 轨迹曲线 DPOS(0)垂直刻度 1000 TABLE(0)垂直刻度 100 
相关指令	TABLE

MOVE_PARA -- 缓冲参数

类型	特殊运动指令
描述	**BASE 轴运动缓冲修改参数。** 这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的 MTYPE 与 MOVE_OP 一致。

语法	MOVE_PARA(paraname, index, value) paraname：参数名，必须是?*set 里面的非只读参数 index：参数编号 value：参数值
适用控制器	20170503 以上固件
例子	BASE(0)　　'选择轴 0 ATYPE=1 SPEED=100 PRINT SPEED　　'打印结果 100 MOVE_PARA(speed ,0,200)　　'修改轴 0 的 SPEED 参数值为 200 DELAY(1000) PRINT SPEED　　'打印结果 200

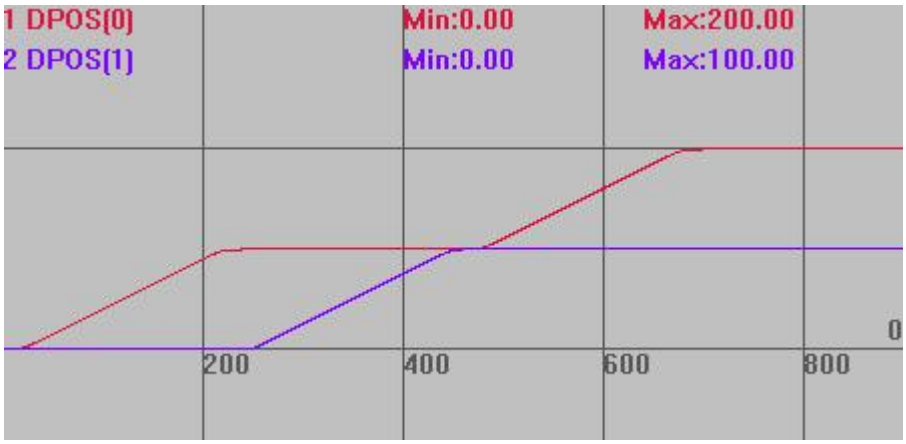
MOVE_PWM -- 缓冲 PWM

类型	特殊运动指令
描述	**BASE 轴运动缓冲操作 PWM。** 这个指令 LOAD 执行时不做任何运动，只操作 PWM。此指令的 MTYPE 与 MOVE_OP 一致。 PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，PWM 频率一定要在 PWM 开关之前调整。
语法	MOVE_PWM(pwmindex,duty[,freq]) pwmindex：pwm 编号 duty：占空比，指有效电平占整个周期的比例；范围 0-1，设置 0 时关闭 pwm；一个周期中先输出有效电平，再输出无效电平 freq：频率，缺省为 1KHz，硬件最大为 1MHz，软件最大为 2KHz
适用控制器	20170503 以上固件
例程	RAPIDSTOP(2) WAIT IDLE TRIGGER TICKS=0 BASE(0) SPEED = 1000 MOVE(10) **MOVE_PWM**(0, 0.111, 2000)　　'轴 0 运行到 10 时，操作 PWM0 MOVE_DELAY(111) **MOVE_PWM**(0, 0.333) MOVE_DELAY(111) **MOVE_PWM**(0, 0.555, 3000) MOVE(100) WHILE NOT IDLE **MOVE_PWM**(0, 0, 1000)　　'关闭 PWM

	? -TICKS, PWM_FREQ(0), PWM_DUTY(0) WA 10 WEND
相关指令	PWM_DUTY，PWM_FREQ

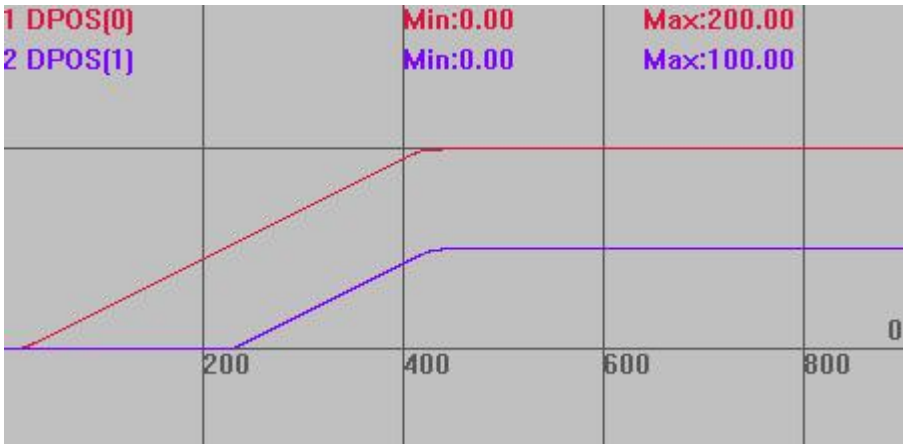
MOVE_SYNMOVE -- 缓冲触发其他轴

类型	特殊运动指令
描述	**BASE 轴运动缓冲触发其他轴运动，当前轴等待。** 此指令的 MTYPE 与 MOVE_DELAY 一致。
语法	MOVE_SYNMOVE(axisnum,dis[,ifsp]) axisnum：需要同步运动的轴号 dis：相对运动距离 ifsp：是否使用 sp 运动，缺省不使用
适用控制器	20170503 以上固件
例程	RAPIDSTOP(2) WAIT IDLE TRIGGER TICKS=0 BASE(0,1) DPOS=0,0 UNITS=100,100 SPEED = 100,100 ACCEL=1000,1000 DECEL=1000,1000 MOVE(100) **MOVE_SYNMOVE**(1,100,0)　'轴 0 运行到 100 时，轴 1 开始运动 100 MOVE(100)　　'待轴同步完成，轴 0 再继续 运动轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200

	
相关指令	MOVE_ASYNMOVE

MOVE_ASYNMOVE -- 缓冲触发其他轴 2

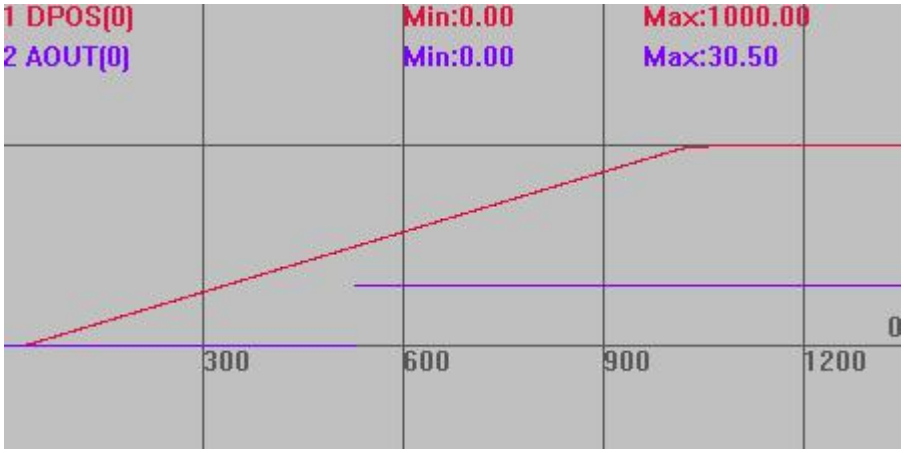
类型	特殊运动指令
描述	**BASE 轴运动缓冲触发其他轴，当前轴不等待。** 此指令的 MTYPE 与 MOVE_OP 一致。
语法	MOVE_ASYNMOVE(axisnum,dis[,ifsp]) axisnum：需要同步运动的轴号 dis：相对运动距离 ifsp：是否使用 sp 运动，缺省不使用
适用控制器	20170503 以上固件
例程	RAPIDSTOP(2) WAIT IDLE TRIGGER TICKS=0 BASE(0,1) DPOS=0,0 UNITS=100,100 SPEED = 100,100 ACCEL=1000,1000 DECEL=1000,1000 MOVE(100) **MOVE_ASYNMOVE**(1,100,0)　'轴 0 运行到 100 时，轴 1 开始运动 100 MOVE(100)　　'轴 0 持续运动 运动轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200

	
相关指令	MOVE_SYNMOVE

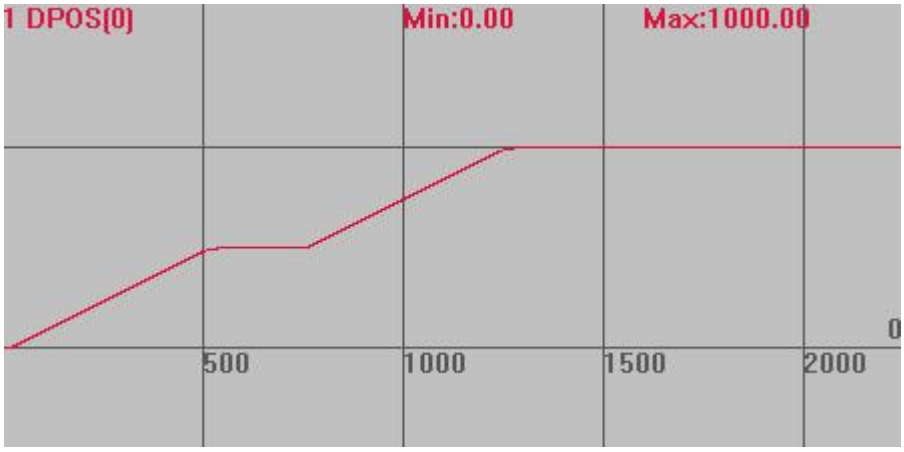
MOVE_TASK -- 缓冲开启任务

类型	特殊运动指令
描述	**BASE 轴运动缓冲加入启动 TASK。** 这个指令 LOAD 执行时不做任何运动，只启动任务，此指令的 MTYPE 与 MOVE_OP 一致。 当任务已经启动时，会报错，但不影响程序执行。
语法	MOVE_TASK(tasknum, label) tasknum：任务号 label：函数名或标号
适用控制器	通用
例子	BASE(0) DPOS=0 UNITS=100 ACCEL=1000 DECEL=1000 SPEED=100 ACCEL=1000 DECEL=1000 MOVE(100) **MOVE_TASK**(1, task_move))　'第一个运动完成后，将 task_move 作为任务 1 启动 MOVE(100) END TASK_MOVE: PRINT "TASK_MOVE" END
相关指令	RUNTASK，RUN

MOVE_AOUT -- 缓冲模拟量输出

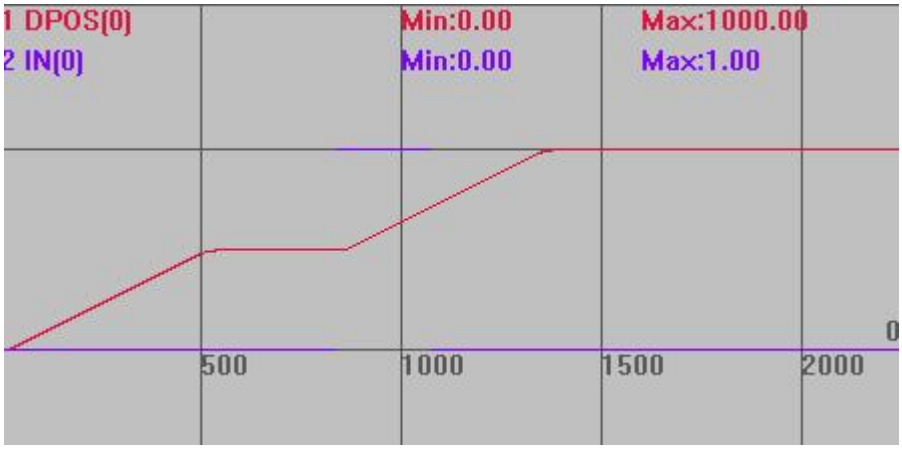
类型	特殊运动指令
描述	**BASE 轴运动缓冲加入一个 AOUT 指令。** 这个指令 LOAD 执行时不做任何运动，只修改 AOUT 值。此指令的 MTYPE 与 MOVE_OP 一致。
语法	MOVE_AOUT(danum, value) danum：DA 编号 value：要修改的值
适用控制器	通用，实际只有包含 DA 通道的控制器生效
例子	BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=1000 DECEL=1000 AOUT(0)=0 'DA0 通道赋值 0 ?AOUT(0) '打印确认 TRIGGER '自动触发示波器 MOVE(500) **MOVE_AOUT**(0, 30.5) '第一个运动完成后,将 DA0 通道赋值 30.5 MOVE(500) WAIT IDLE ?AOUT(0) '打印 DA0，30.5 运动轨迹 DPOS(0)垂直刻度 1000 AOUT(0)垂直刻度 100 
相关指令	AOUT

MOVE_DELAY -- 缓冲延时

类型	特殊运动指令
描述	**BASE 轴运动缓冲加入一个延时。** 这个指令 LOAD 执行时不做任何运动，只延时指定时间。 前面的运动指令结束时速度会自动将为 0。
语法	move_delay(timems) 别名：move_wa(timems) timems：延时，毫秒数
适用控制器	通用
例子	BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=2000 DECEL=2000 TRIGGER '自动触发示波器 MOVE(500) **MOVE_DELAY**(1000) '两个 MOVE 中间等待 1 秒 MOVE(500) 运动轨迹 DPOS(0)垂直刻度 1000 
相关指令	DELAY

MOVE_WAIT -- 缓冲等待

类型	特殊运动指令
描述	**BASE 轴运动缓冲加入一个条件判断。** 这个指令 LOAD 执行时不做任何运动，只等待指定的条件满足，前面的运动指令结束时速度会自动降为 0。

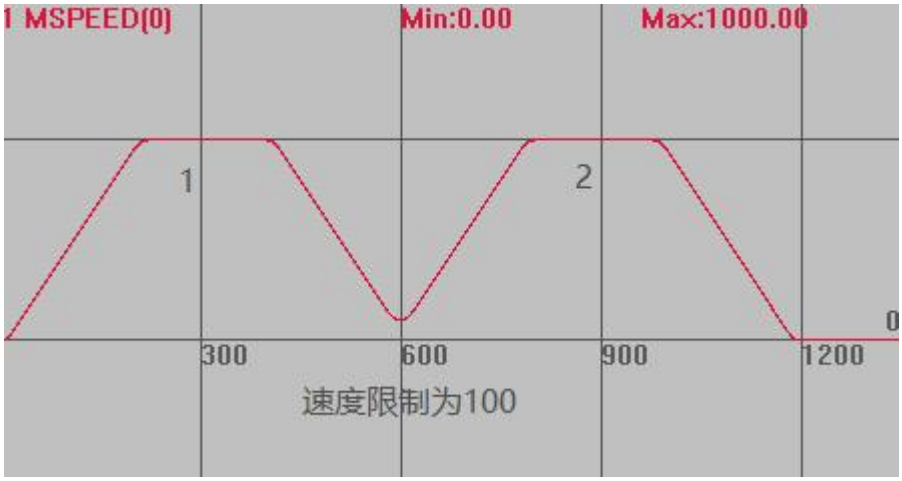
语法	MOVE_WAIT(paraname, paranum, eq, value) paraname：选择参数名 (参数可以为：DPOS，MPOS，IN，AIN，VPSPEED，MSPEED，MODBUS_REG，MODBUS_IEEE，MODBUS_BIT，VECTOR_BUFFERED，REMAIN) paranum：参数编号或轴号 eq：1 ≥ -1 ≤ 对 IN 等 BIT 类型参数无效 0 不建议使用 value：比较值
适用控制器	固件 150802 以上版本， 或 XPLC160405 以上版本支持。
例子	BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=2000 DECEL=2000 TRIGGER '自动触发示波器 MOVE(500) **MOVE_WAIT**(IN, 0, 0, 1) '等待 IN(0)有信号，才执行下一条运动缓冲 MOVE(500) 运动轨迹 DPOS(0)垂直刻度 1000 IN(0)垂直刻度 1 
相关指令	WAIT UNTIL

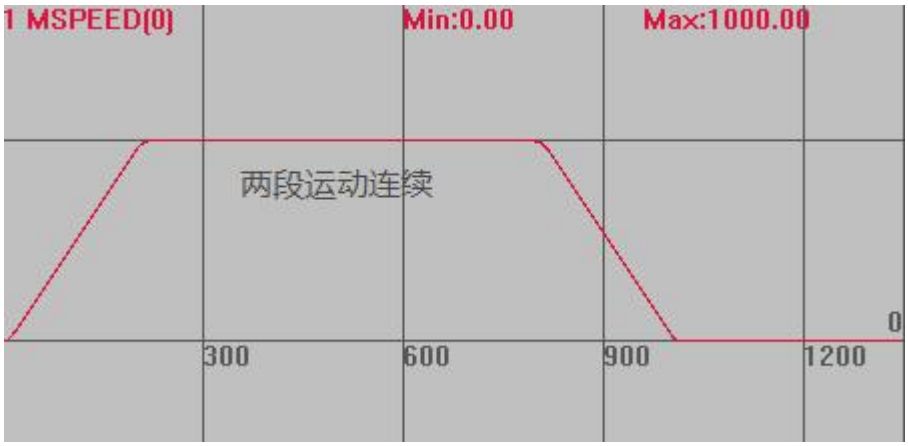
MOVE_CANCEL--缓冲停止

类型	特殊运动指令
描述	**把 CANCEL 指令写入运动缓冲。**
语法	MOVE_CANCEL(iaxis, imode)

	iaxis：要操作的轴号 imode：选择 CANCEL 模式，同 CANCEL 指令
适用控制器	通用
例子	MOVE_CANCEL(1,0) AXIS(0)　'轴 0 缓冲里面写入停止轴 1 的指令
相关指令	CANCEL

MOVELIMIT -- 速度限制

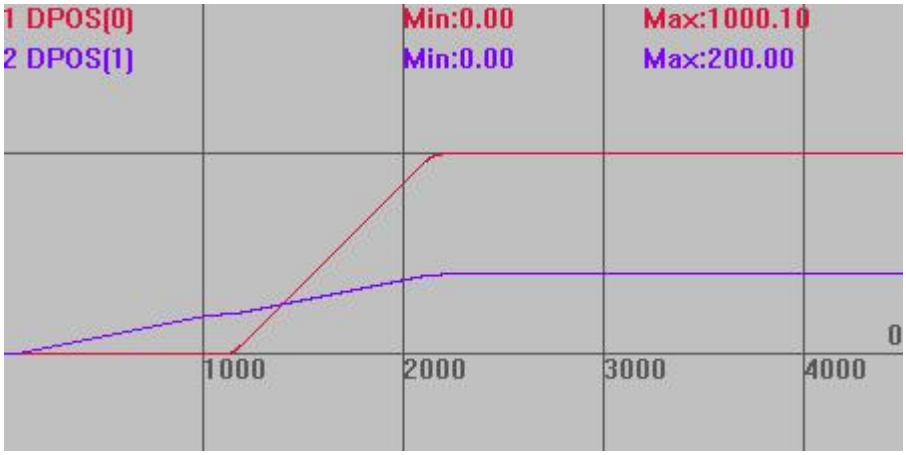
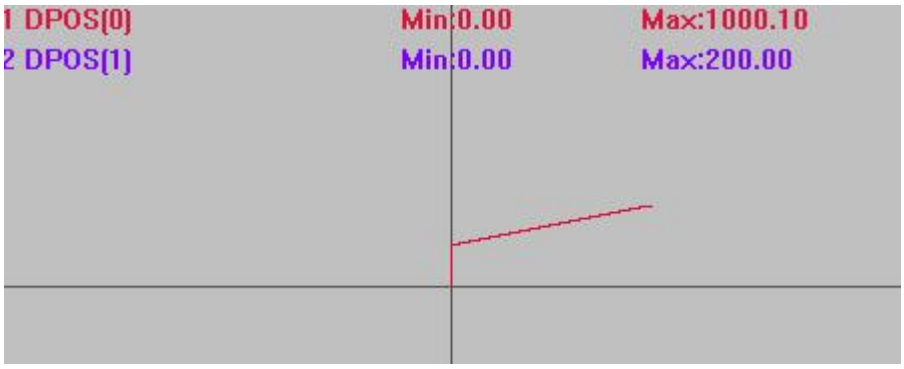
类型	特殊运动指令
描述	**在当前的运动末尾位置增加速度限制，用于强制拐角减速。**
语法	MOVELIMIT(limitspeed) limitspeed：限制到的速度
适用控制器	通用
例子	BASE(0) UNITS=100 DPOS=0 SPEED=1000　　'轴速度 ACCEL=1000　　'轴加速度 DECEL=1000 SRAMP=100 MERGE=ON　　'开启连续插补，多段运动间速度连续 TRIGGER　　'自动触发示波器 MOVE(2000) **MOVELIMIT**(100)　　'在两个运动的中间降速到 100 MOVE(2000) 用 MOVELIMIT 限制速度时，插补速度 MSPEED(0)垂直刻度 1000  不限制速度时

	
相关指令	CORNER_MODE

6.4 同步运动指令

CONNECT -- 同步运动

类型	同步运动指令
描述	**将当前轴的目标位置与 driving_axis 轴的测量位置通过电子齿轮连接。** 连接的是脉冲个数，要考虑不同轴 UNITS 的比例。取消连接时用 CANCEL。 假设连接轴 0 的 UNIST 为 100，被连接轴 1 的 UNITS 为 10。 使用 CONNECT 连接，比例 ratio 为 1，当轴 0 运动 s1=100 时，轴 1 运动 =s1*UNITS(0)*ratio/UNITS(1)，此时运动 1000。 比率可以通过重复调用指令动态变化。 常用于手轮使用。
语法	CONNECT（ratio，driving_axis） ratio：比率，可正可负，注意是脉冲个数的比例 driving_axis：连接轴的轴号，手轮时为编码器轴
适用控制器	通用
例子	RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0,1) ATYPE=1,1 UNITS=10,100 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER '自动触发示波器 MOVE(100) AXIS(1) '轴 1 运动 100，此时轴 0 不动

	WAIT IDLE(1) **CONNECT**(1,1)　AXIS(0)　　'轴 0 连接到轴 1，比例为 1 MOVE(100)　AXIS(1)　　'轴 1 运动 100，轴 0 运动 1000 运动轨迹 DPOS(0)垂直刻度 1000 DPOS(1)垂直刻度 500  合成轨迹（XY 模式） 
相关指令	CLUTCH_RATE，CONNPATH

CONNPATH -- 同步运动 2

类型	同步运动指令
描述	**将当前轴的目标位置与 driving_axis 轴的插补矢量长度通过电子齿轮连接。** 连接的是脉冲个数，要考虑不同轴 UNITS 的比例。取消连接时用 CANCEL。 比率可以通过重复调用指令动态变化。
语法	CONNPATH（ratio，driving_axis） Ratio：比率，可正可负，注意是脉冲个数的比例 driving_axis：连接轴的轴号，手轮时为编码器轴
适用控制器	通用
例子	RAPIDSTOP(2) WAIT　IDLE(0)

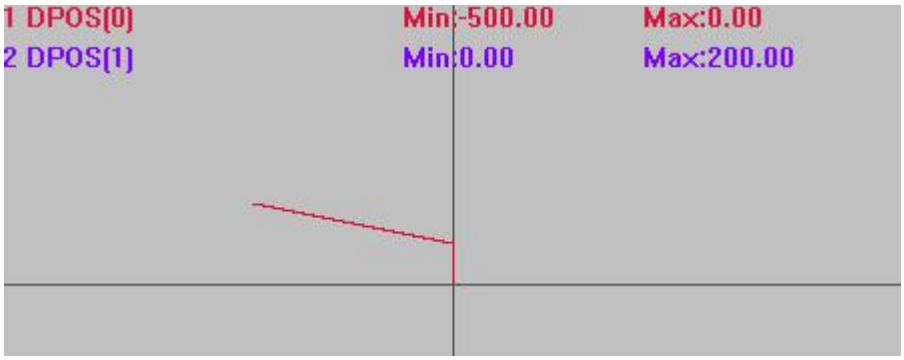
```
WAIT    IDLE(1)
BASE(0,1)
DPOS=0,0
ATYPE=1,1
UNITS=10,100
SPEED=100,100
ACCEL=1000,1000
DECEL=1000,1000
TRIGGER                    '自动触发示波器
MOVE(100)    AXIS(1)            '轴 1 运动 100，此时轴 0 不动
WAIT IDLE(1)
CONNPATH(-0.5,1)    AXIS(0)        '轴 0 连接到轴 1，比例为-0.5
MOVE(100)    AXIS(1)            '轴 1 运动 100，轴 0 运动-500
```

运动轨迹

DPOS(0)垂直刻度 500

DPOS(1)垂直刻度 500

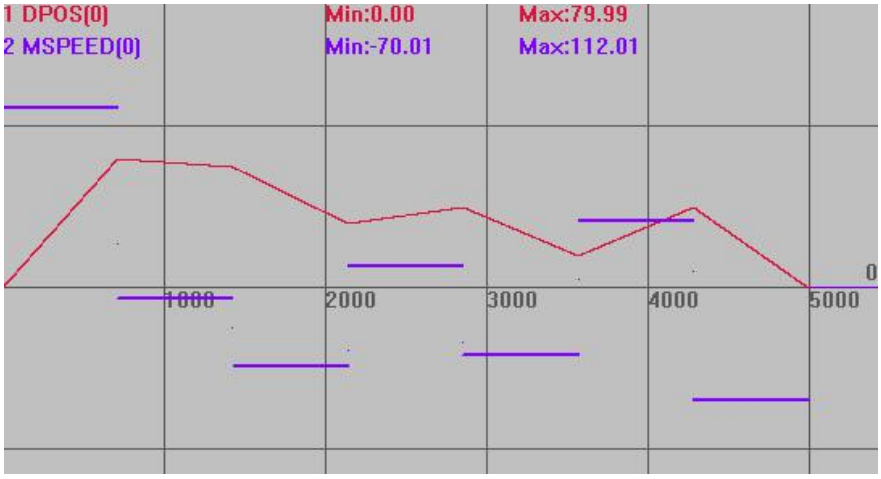
合成轨迹(XY 模式)



相关指令	CLUTCH_RATE，CONNECT

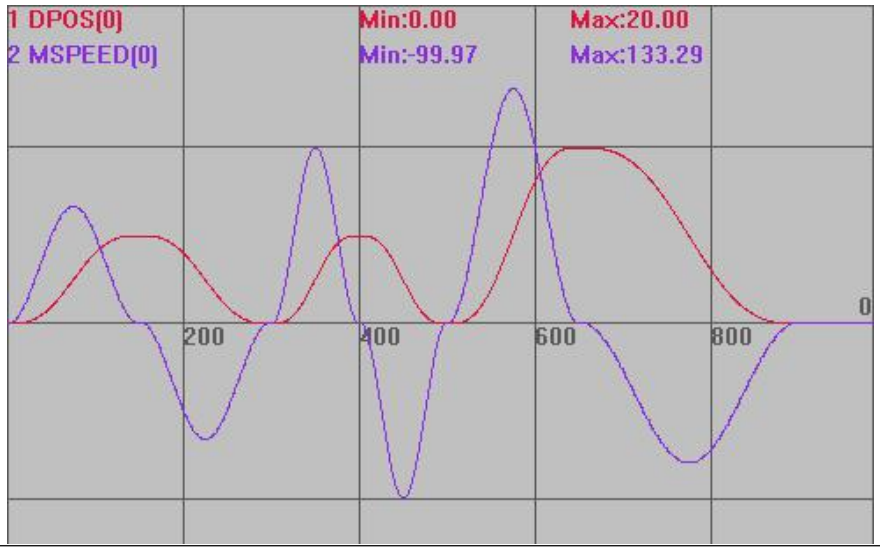
CAM -- 凸轮表运动

类型	同步运动指令
描述	**CAM 指令根据存储在 TABLE 中的数据来决定轴的运动，这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的绝对位置。** 注：两个或多个 CAM 指令可以同时使用同一段 TABLE 数据区进行操作。 运动的总时间由设置速度和第四个参数决定，运动的实际速度根据 TABLE 轨迹与时间自动匹配。 TBALE 数据需要手动设置，第一个数据为引导点，建议设为 0。 TABLE 数据*table multiplier 这个比例=发出的脉冲数。 请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。
语法	CAM(start point, end point, table multiplier, distance) start point：起始点 TABLE 编号，存储第一个点的位置 end point：结束点 TABLE 编号 table multiplier：位置乘以这个比例，一般设为脉冲当量值 distance：参考运动的距离 总时间=distance/轴 speed
适用控制器	通用
例子	例一 RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0) UNITS=100 DPOS=0 ACCEL=1000 DECEL=1000 SPEED=100 TABLE(10,0,80,75,40,50,20,50,0) 'table 从 10 开始存数据 TRIGGER '自动触发示波器 **CAM**(10,17,100,500) '运动轨迹为 table10 到 17，运动总时间为 500/100=5s 轨迹与速度 DPOS(0)垂直刻度 100 MSPEED(0)垂直刻度 100



例二 CAM 在高速高精度运动上的应用

```
DIM    num_p,scale,m,t     '变量定义
num_p=100
scale=500
FOR p=0 TO num_p
   TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale) 'table 存储凸轮表运动参数
NEXT
RAPIDSTOP(2)
WAIT    IDLE(0)
WAIT    IDLE(1)
BASE(0)          '选择轴 0
DEFPOS(0)
SERVO=ON
UNITS=500
SPEED=1000
ACCEL=1000000
DECEL=1000000
TRIGGER

m=10             '代表距离的倍数
t=0.3            '运行时间
SPEED=1000
CAM(0,100,m,SPEED*t)
WAIT IDLE

m=10
t=0.3
SPEED=1000
CAM(0,100,-m,SPEED*t)
WAIT IDLE
```

	m=10 t=0.2 SPEED=500 CAM(0,100,m,SPEED*t) WAIT IDLE m=10 t=0.2 SPEED=500 CAM(0,100,-m,SPEED*t) WAIT IDLE m=20 t=0.3 SPEED=1000 CAM(0,100,m,SPEED*t) WAIT IDLE m=20 t=0.5 SPEED=500 CAM(0,100,-m,SPEED*t) WAIT IDLE 插补轨迹 DPOS(0)垂直刻度 20 MSPEED(0)垂直刻度 100 
相关指令	CAMBOX

CAMBOX -- 跟随凸轮表运动

<table>
<tr><td>类型</td><td>同步运动指令</td></tr>
<tr><td>描述</td><td>CAMBOX 指令根据存储在 TABLE 中的数据来决定轴的运动，这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的距离。跟随轴的运动根据参考轴的运动来进行。
注：两个或多个 CAMBOX 指令可以同时使用同一段 Table 数据区进行操作。

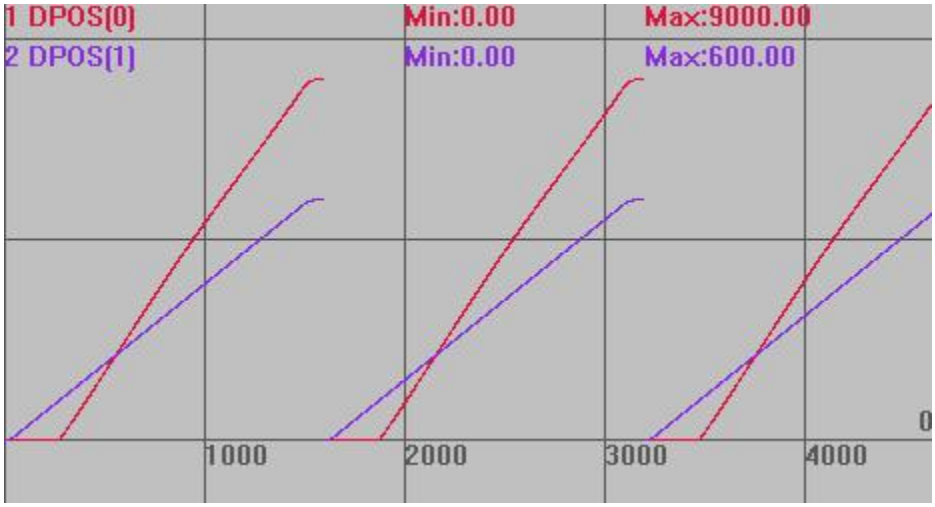
当前轴运动的总时间由参考轴的运动距离和轴速度确定，速度自动匹配。
Table 数据需要手动设置，第一个数据为引导点，建议设为 0。
Table 数据*table multiplier 这个比例=发出的脉冲数。
请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。
20170428 以上固件支持。</td></tr>
<tr><td>语法</td><td>CAMBOX(start_point, end_point, table_multiplier, link_distance , link_axis[, link_options][, link_pos][, link_offpos])
start point：起始点 TABLE 编号，存储第一个点的位置
end point：结束点 TABLE 编号
table multiplier：位置乘以这个比例，一般设为脉冲当量值
link_distance：参考轴运动的距离
link_axis：参考轴轴号
link_options：与参考轴的连接方式，不同的二进制位代表不同的意义
<table>
<tr><td>位</td><td>意义</td></tr>
<tr><td>bit0</td><td>当参考轴 MARK 信号事件触发时，当前轴与参考轴开始进行连接运动</td></tr>
<tr><td>bit1</td><td>当参考轴运动到设定的绝对位置时，当前轴与参考轴开始连接运动</td></tr>
<tr><td>bit2</td><td>自动重复连续双向运行。(通过设置 REP_OPTION=1，可以取消重复)</td></tr>
<tr><td>bit4</td><td>从中间某个位置启动，配合掉电中断实现恢复凸轮</td></tr>
<tr><td>bit5</td><td>只有参考轴的正向运动才连接</td></tr>
<tr><td>bit8</td><td>当参考轴 MARKB 信号事件触发时，当前轴与参考轴开始进行连接运动，锁存轴号为参考轴的轴号，需要最新固件支持</td></tr>
</table>
link_pos：当 link_options 参数设置为 2 时，该参数表示连接开始启动的绝对位置
link offpos：当 link_options 参数 bit4 置为 1 时，该参数表示主轴已经运行完的相对位置</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>ERRSWITCH = 3
RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
BASE(0,1) '选择轴号
ATYPE=1,1 '脉冲方式步进或伺服
DPOS = 0,0
UNITS = 100,100 '脉冲当量
SPEED = 200,200</td></tr>
</table>

```
ACCEL = 2000,2000
DECEL = 2000,2000
'计算 TABLE 的数据
DIM deg, rad, x, stepdeg
stepdeg = 2                 '可以通过这个来修改段数，段数越多速度越平稳
FOR deg=0 TO 360 STEP stepdeg
    rad = deg * 2 * PI/360     '转换为弧度
    x = deg * 25 + 10000 * (1-COS(rad))/100
    TABLE(deg/stepdeg,x)            '存储 TABLE
    TRACE deg/stepdeg,x
NEXT deg
TRIGGER                     '自动触发示波器
WHILE 1             '循环运动
    IF IN(0) = ON THEN    '输入 0 有效启动运动
        DPOS = 0,0
        CAMBOX(0,360/stepdeg, 100, 500, 1,2,100)    AXIS(0) '参考轴轴 1 运动到 100 位置时，跟随轴轴 0 启动
        MOVE(600)    AXIS(1)
        WAIT UNTIL IDLE AND IDLE(1)    '等待运动停止
        DELAY(100) '延时
    ENDIF
WEND
END             '停止当前任务
```

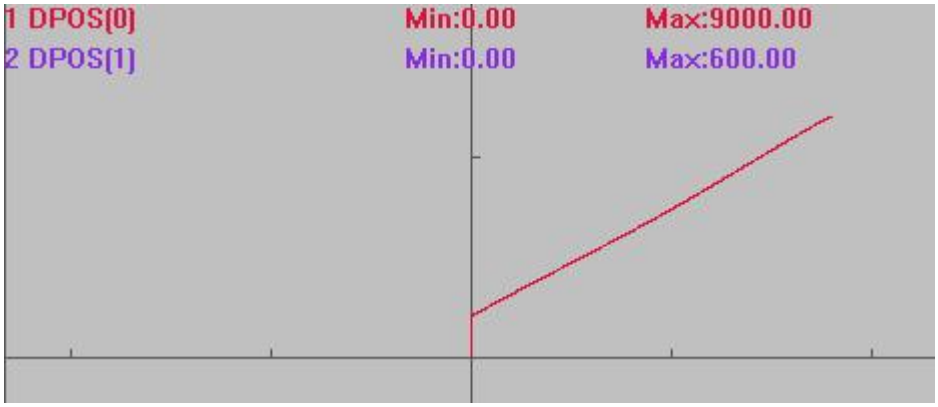
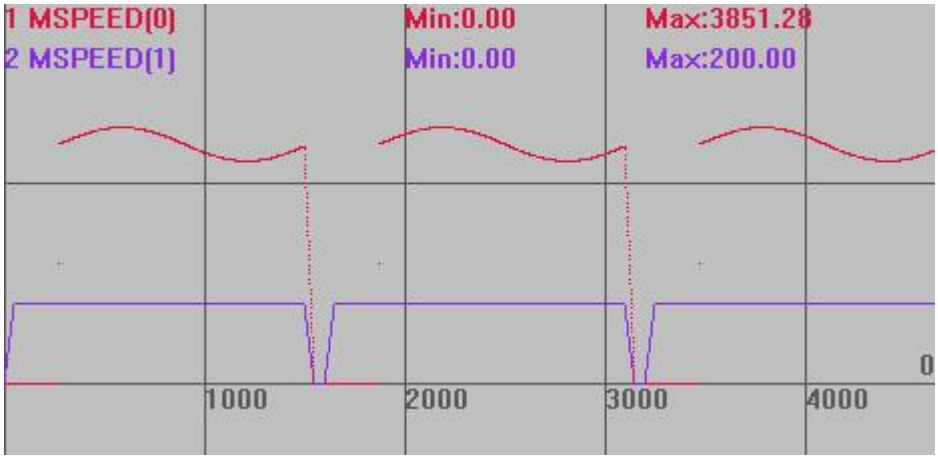
插补轨迹

DPOS(0)垂直刻度 5000

DPOS(1)垂直刻度 500



XY 模式下合成轨迹

	 速度曲线 MSPEED(0)垂直刻度 3000 MSPEED(1)垂直刻度 500 
相关指令	CAM

MOVELINK -- 自动凸轮

类型	同步运动指令
描述	**此指令用于自定义的凸轮运动，该运动带有可设置的加减速阶段。** 被连接轴为参考轴，连接轴为跟随轴。 连接轴的距离分成 3 个阶段应用于参考轴的运动，分别是加速部分、匀速部分和减速部分。 在加速和减速阶段为了与速度匹配，link distance（基本轴运动距离）必须是 distance（跟随轴运动距离）的两倍。 请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。
语法	MOVELINK (distance, link dist, link acc, link dec, link axis[,link options] [,link pos][,link offpos])

distance：从连接开始到结束，跟随轴移动的距离，此参数可正可负，为正数正方向跟随，为负数负方向跟随，采用 units 单位

link dist：参考轴在连接的整个过程中移动的绝对距离，采用 units 单位

link acc：在跟随轴加速阶段，参考轴移动的绝对距离，采用 units 单位

link dec：在跟随轴减速阶段，参考轴移动的绝对距离，采用 units 单位

注：如果参数 3 和参数 4 的和大于第 2 个参数，他们会被自动按比例减小，使其和值与第 2 个参数值相等

link axis：参考轴的轴号

link options：连接模式选项，不同的二进制位代表不同的意义

模式	位	描述
1	位 0	连接精确开始于参考轴上 MARK 事件被触发的时刻
2	位 1	连接开始于参考轴到达一个绝对位置时(见 link pos 参数描述)
4	位 2	当此位被设置时，MOVELINK 会自动重复执行并且可以反向(这个模式可以通过设置轴参数 REP_OPTION 的第 1 位为 1 来清除)
8	位 3	当设置时,采用 S 曲线加减速。20170502 以上固件支持
16	位 4	从中间某个位置启动，配合掉电中断实现恢复跟随
32	位 5	只有参考轴为正向运动才连接
256	位 8	连接精确开始于参考轴上 MARKB 事件被触发的时刻，需要最新固件支持

link pos：当 link options 参数设置为 2 时，该参数表示基本轴在该绝对位置值时，连接开始

link offpos：当 link_options 参数 bit4 置为 1 时，该参数表示主轴已经运行完的相对位置。20170428 以上固件支持

适用控制器

通用

例子

例一

```
RAPIDSTOP(2)
WAIT  IDLE(0)
WAIT  IDLE(1)
BASE(0,1)      '轴 0 为跟随轴，轴 1 为参考轴
UNITS=100,100
ATYPE=1,1
DPOS=0,0
SPEED=100,100
ACCEL=2000,2000
DECEL=2000,2000
TRIGGER                    '自动触发示波器
MOVELINK(100,100,0,0,1)  AXIS(0)      '不设置加减速阶段时，效果与 CONNECT 相同，区别在不需要考虑 UNITS 的不同，且不会有累积误差。此时运动比例 1:1
MOVE(150)  AXIS(1)        '轴 1 运动 150，轴 0 跟随轴 1 运动完 100
```

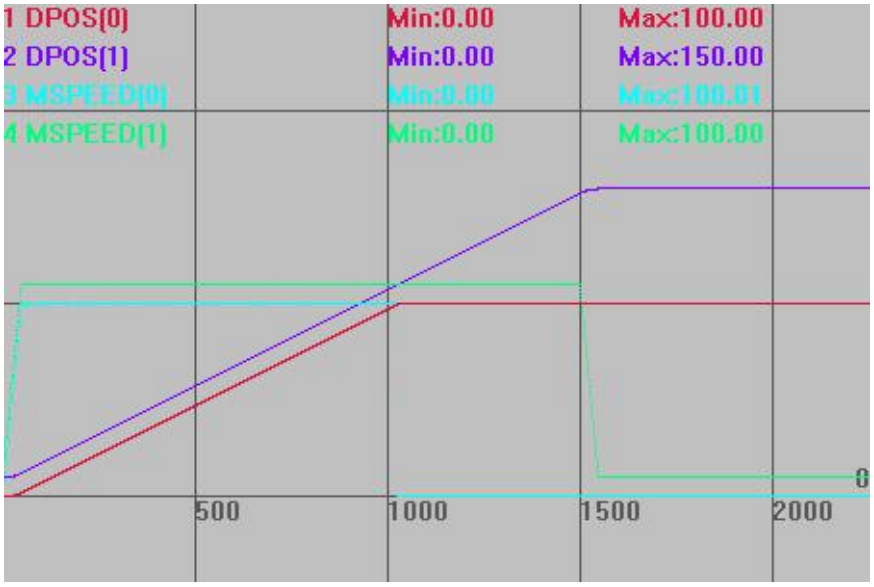
插补轨迹与速度曲线

DPOS(0)垂直刻度 100，无偏移

DPOS(1)垂直刻度 100，偏移 10

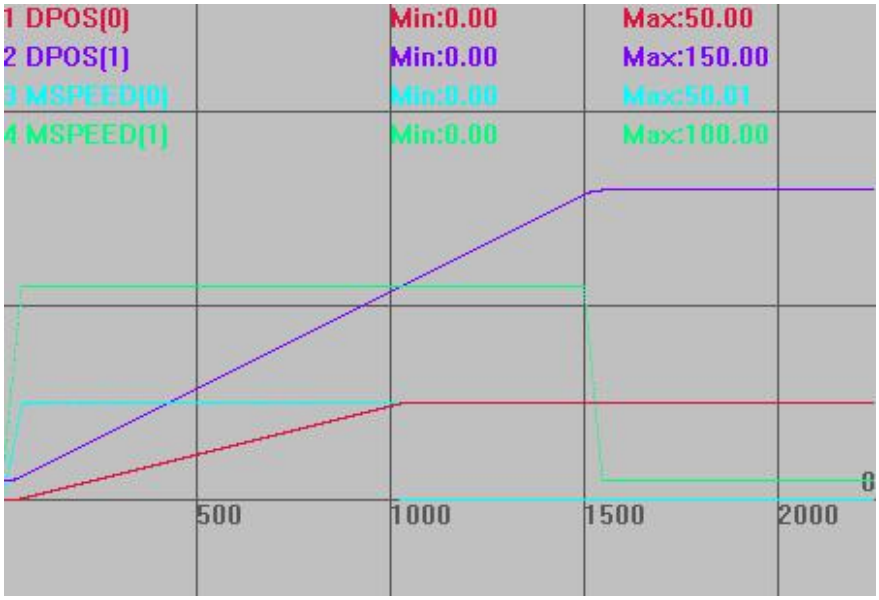
MSPEED(0)垂直刻度 100，无偏移

MSPEED(1)垂直刻度 100，偏移 10



MOVELINK(50,100,0,0,1)

竖直刻度同上



例二 飞剪应用

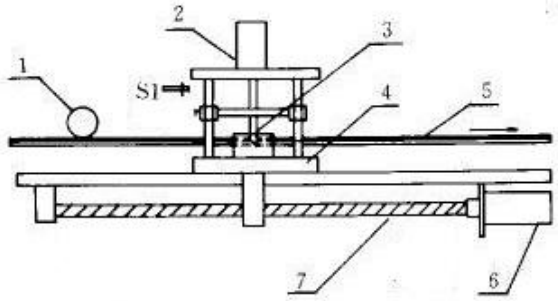


图1 飞剪系统原理图
1—测长轮；2—液压缸；3—刀具；4—工作台；
5—型材；6—伺服电机；7—丝杠

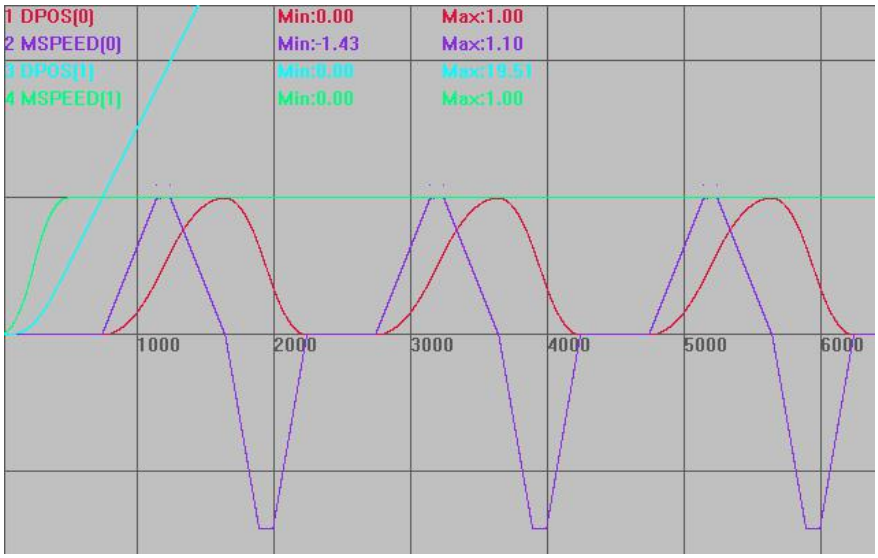
型材持续运动，工作台先静止；直到型材持续运动了某段距离，工作台开始加速；待工作台速度与型材一致，然后开关 S1 工作刀具下剪，剪切完后刀具回升；工作台开始减速，然后退回起始点。重复过程，剪切得到设定长度的型材。

假设要切的型材长度为 4m，工作台运行距离 1m，轴 1 为基本轴（型材传送），轴 0 为跟随轴（追剪工作台），OUT0 口控制刀具，飞剪部分程序如下：

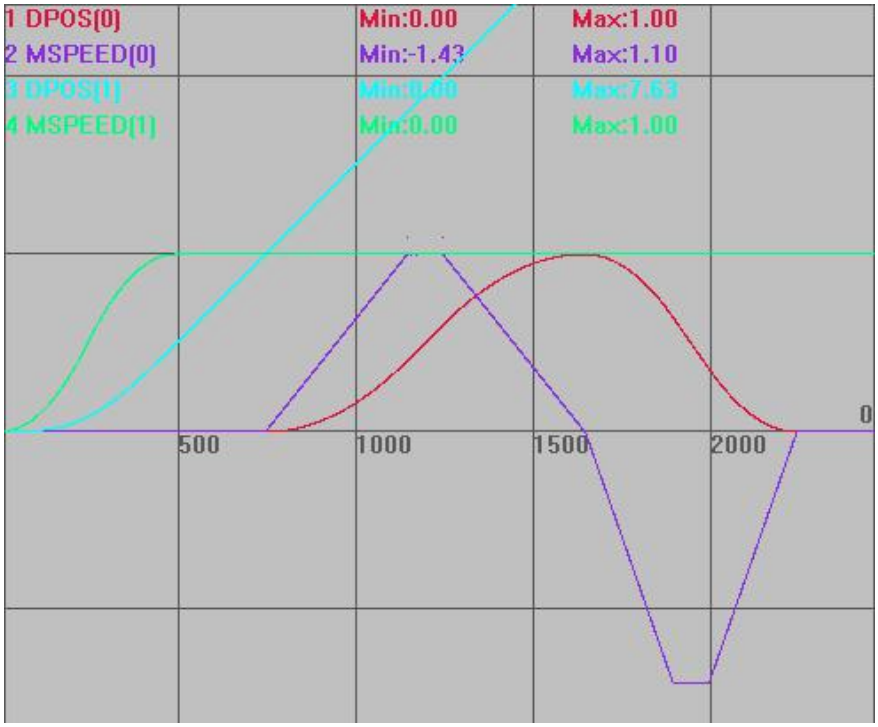
```
RAPIDSTOP(2)
WAIT    IDLE(0)
WAIT    IDLE(1)
BASE(0,1)
UNITS=10000,10000
ATYPE=1,1
DPOS=0,0
SPEED=1,1       '型材运行速度 1m/s,60m/min
ACCEL=2,2
DECEL=2,2

VMOVE(1) AXIS(1)      '型材持续运动
TRIGGER               '自动触发示波器

WHILE 1
        BASE(0)
        MOVELINK(0,1,0,0,1) AXIS(0)     '型材运动 1m 前，工作台静止
        MOVELINK(0.4,0.8,0.8,0,1) AXIS(0)   '工作台加速阶段
        MOVELINK(0.2,0.2,0,0,1) AXIS(0)      '速度同步跟随 0.2m
        MOVE_OP2(0,on,1000)        '刀具下剪，1s 后回升(时间要计算好)
        MOVELINK(0.4,0.8,0,0.8,1) AXIS(0)   '工作台减速阶段
        MOVELINK(-1,1.2,0.5,0.5,1) AXIS(0)   '工作台回到起始点
WEND
```

运动轨迹和速度曲线：
DPOS(0)垂直刻度 1，无偏移

MSPEED(0)垂直刻度 1，无偏移
DPOS(1)垂直刻度 1，无偏移
MSPEED(1)垂直刻度 1，无偏移



一个周期内的运行曲线：

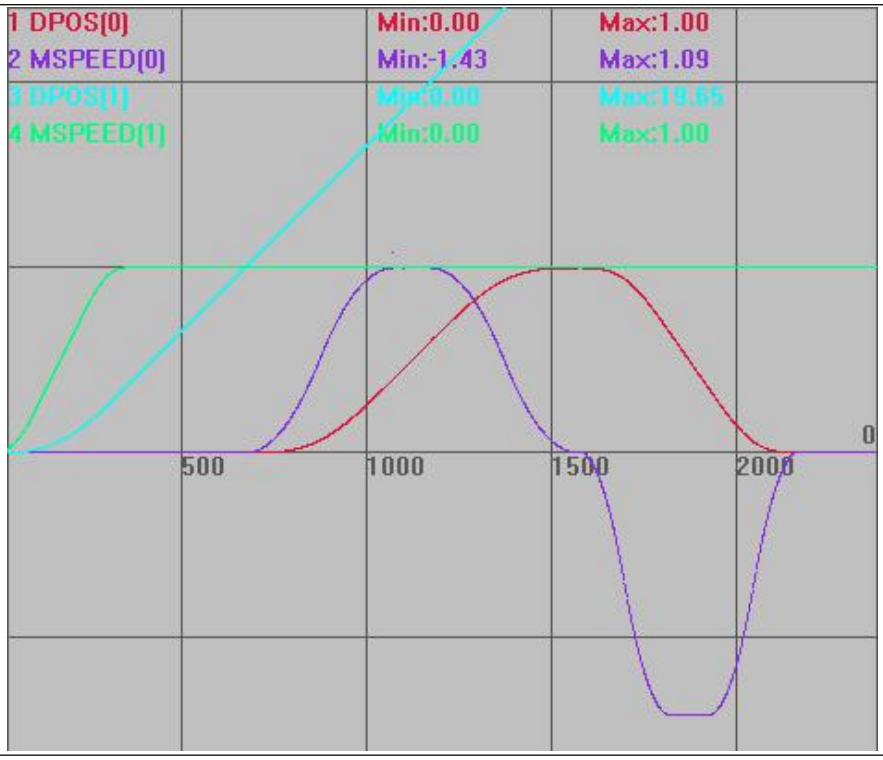


工作台(跟随轴)的运动距离：0.4(加速阶段)+0.2(跟随同步)+0.4(减速阶段)=1m 单位，然后-1m 返回运动。
型材(参考轴)的运动距离：1+0.8+0.2+0.8+1.2=4m 单位，全程匀速。

例三 设置 link options bit3=1 时，从轴追剪轴采用 S 曲线加减速

```
RAPIDSTOP(2)
WAIT    IDLE(0)
WAIT    IDLE(1)
```

```
DATUM(0)
BASE(0,1)
UNITS=10000,10000
ATYPE=1,1
DPOS=0,0
SPEED=1,1          "型材运行速度 1m/s,60m/min
ACCEL=2,2
DECEL=2,2
SRAMP=200,200

VMOVE(1) AXIS(1)        '型材持续运动
TRIGGER                 '自动触发示波器

WHILE 1
    BASE(0)
    MOVELINK(0,1,0,0,1,8) AXIS(0)    '型材运动 1m 前，工作台静止
    MOVELINK(0.4,0.8,0.8,0,1,8) AXIS(0)   '工作台加速阶段
    MOVELINK(0.2,0.2,0,0,1,8) AXIS(0)      '速度同步跟随 0.2m
    MOVE_OP2(0,on,1000)    '刀具下剪，1s 后回升(时间要计算好)
    MOVELINK(0.4,0.8,0,0.8,1,8) AXIS(0)   '工作台减速阶段
    MOVELINK(-1,1.2,0.5,0.5,1,8) AXIS(0)   '工作台回到起始点
WEND

运动轨迹和速度曲线：
DPOS(0)垂直刻度 1，无偏移
MSPEED(0)垂直刻度 1，无偏移
DPOS(1)垂直刻度 1，无偏移
MSPEED(1)垂直刻度 1，无偏移
```

	
相关指令	MOVESLINK ，MOVELINK_MODIFY

MOVESLINK -- 自动凸轮 2

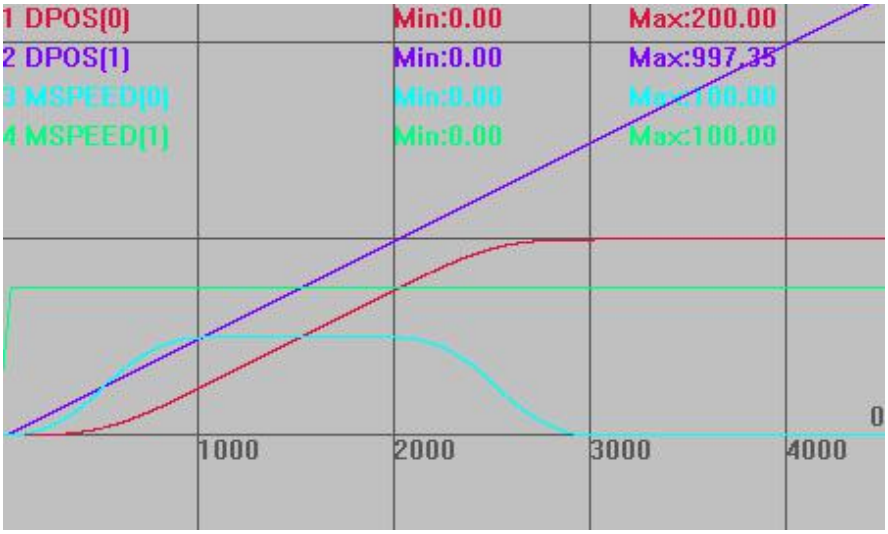
类型	同步运动指令
描述	**此指令用于自定义的凸轮运动，该运动自动规划中间曲线，不用计算凸轮表。** 被连接轴为参考轴，连接轴为跟随轴。 在加速和减速阶段为了与速度匹配，下一条 MOVESLINK 的 start sp 必须与当前 MOVESLINK 的 end sp 相同。 请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。
语法	MOVESLINK (distance,link dist,start sp,end sp,link axis[,link options][,link pos][, link offpos]) [, link options] [, link pos] 可选参数不填时，逗号不能省掉，控制器根据参数的位置来判断是什么参数。 distance：从连接开始到结束，跟随轴移动的距离，此参数可正可负，为正数正方向跟随，为负数负方向跟随，采用 units 单位 link dist：参考轴在连接的整个过程中移动的绝对距离，采用 units 单位 start sp：启动时跟随轴和参考轴的速度比例，units/units 单位，负数表示跟随轴负向运动 end sp：结束时跟随轴和参考轴的速度比例，units/units 单位，负数表示跟随轴负向运动 注：当 start sp = end sp = distance/link dist 时，匀速运动 link axis：参考轴的轴号 link options：连接模式选项，不同的二进制位代表不同的意义

模式	位	描述

<table>
<tr><td></td><td>
<table>
<tr><td>1</td><td>位 0</td><td>连接精确开始于参考轴上 MARK 事件被触发的时刻</td></tr>
<tr><td>2</td><td>位 1</td><td>连接开始于参考轴到达一个绝对位置时(见 link pos 参数描述)</td></tr>
<tr><td>4</td><td>位 2</td><td>当此位被设置时，MOVELINK 会自动重复执行并且可以反向(这个模式可以通过设置轴参数 REP_OPTION 的第 1 位为 1 来清除)</td></tr>
<tr><td>16</td><td>位 4</td><td>使用 link offpos 从中间来启动，配合掉电中断实现恢复，20170428 以上固件支持</td></tr>
<tr><td>32</td><td>位 5</td><td>只有参考轴为正向运动才连接</td></tr>
<tr><td>256</td><td>位 8</td><td>连接精确开始于参考轴上 MARKB 事件被触发的时刻，需要最新固件支持</td></tr>
</table>
link pos：当 link options 参数设置为 2 时，该参数表示参考轴在该绝对位置值时，连接开始
link offpos：当 link_options 参数 bit4 置为 1 时，该参数表示主轴已经运行完的相对位置。20170428 以上固件支持。
</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>功能与 MOVELINK 相同，仅是参数设置区别
例一
RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
DATUM(0)

BASE(0,1)
UNITS=100,100
ATYPE=1,1
DPOS=0,0
SPEED=100,100
ACCEL=2000,2000
DECEL=2000,2000
TRIGGER '自动触发示波器
MOVESLINK(50,100,0,1,1) AXIS(0) '轴 0 跟踪轴 1 运动，从速度 0 到速度一致
MOVESLINK(100,100,1,1,1) AXIS(0) '轴 0 跟踪轴 1 运动，匀速 100units
MOVESLINK(50,100,1,0,1) AXIS(0) '轴 0 跟踪轴 1 运动，减速到 0
VMOVE(1) AXIS(1)

运动轨迹和速度曲线
DPOS(0)垂直刻度 200，无偏移
DPOS(1)垂直刻度 200，无偏移
MSPEED(0)垂直刻度 200，无偏移
MSPEED(1)垂直刻度 200，偏移 50</td></tr>
</table>



例二 追剪

```
RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
DATUM(0)

BASE(0,1)
UNITS=10000,10000
ATYPE=1,1
DPOS=0,0
SPEED=1,1
ACCEL=2,2
DECEL=2,2
SRAMP=200,200
TRIGGER                    '自动触发示波器
VMOVE(1)   AXIS(1)
WHILE 1
MOVESLINK(0,1,0,0,1)   AXIS(0)    '型材运动 1 单位前，工作台静止
MOVESLINK(0.4,0.8,0,1,1)   AXIS(0)   '工作台加速阶段
MOVESLINK(0.2,0.2,1,1,1)   AXIS(0)     '速度跟随阶段
MOVESLINK(0.4,0.8,1,0,1)   AXIS(0)   '工作台减速阶段
MOVESLINK(-1,1.2,0,0,1)   AXIS(0)   '工作台回到起始点
WEND
```

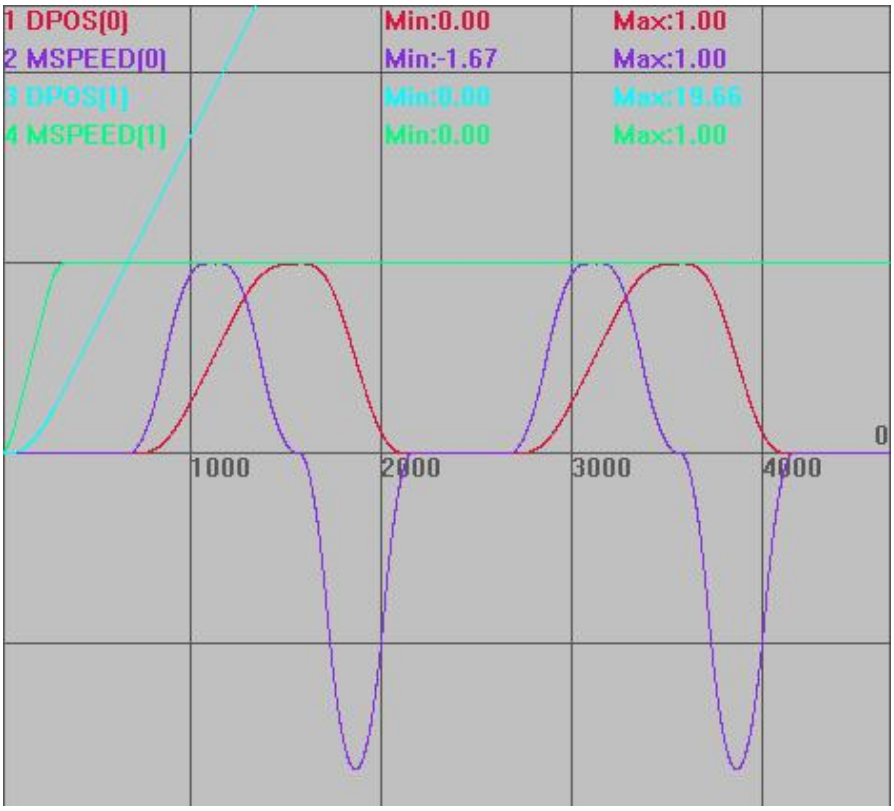
运动轨迹和速度曲线

DPOS(0)垂直刻度 1，无偏移

DPOS(1)垂直刻度 1，无偏移

MSPEED(0)垂直刻度 1，无偏移

MSPEED(1)垂直刻度 1，无偏移

	
相关指令	MOVELINK

MOVELINK_MODIFY -- 同步距离修改

类型	轴参数
描述	**相对修改 MOVELINK 指令的同步区长度。** 带入运动缓冲，只在同步段后设置生效。
语法	VAR1 = MOVELINK_MODIFY，MOVELINK_MODIFY = expression
适用控制器	20160926 以后固件版本支持。
例子	例一 RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0,1) UNITS=100,100 ATYPE=1,1 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER '自动触发示波器 **未修改同步距离**

```
MOVELINK(10,20,20,0,1)          '工作台加速阶段
MOVELINK(100,100,0,0,1)  '同步阶段 100
MOVELINK(10,20,0,20,1)          '减速阶段
VMOVE(1)    AXIS(1)
```

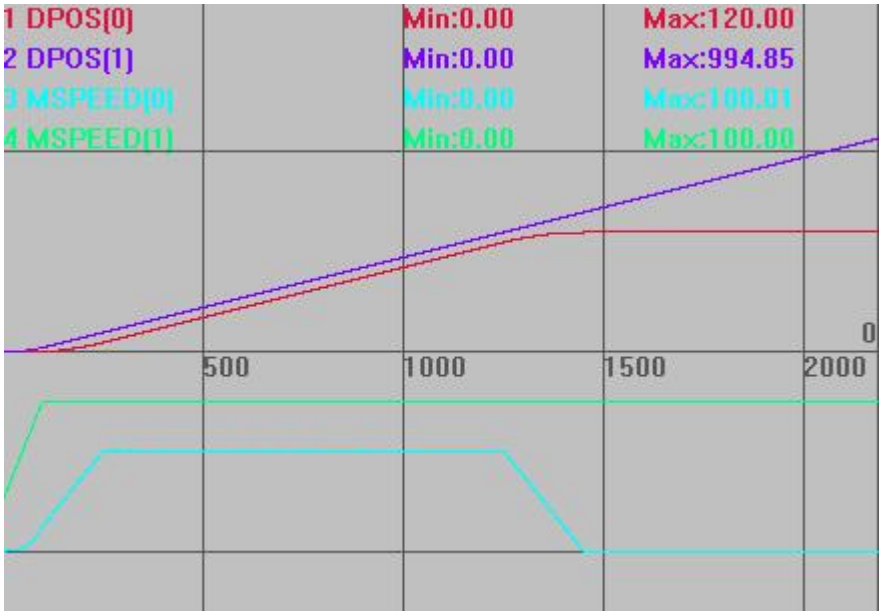
运动轨迹和速度曲线

DPOS(0)垂直刻度 200，无偏移

DPOS(1)垂直刻度 200，无偏移

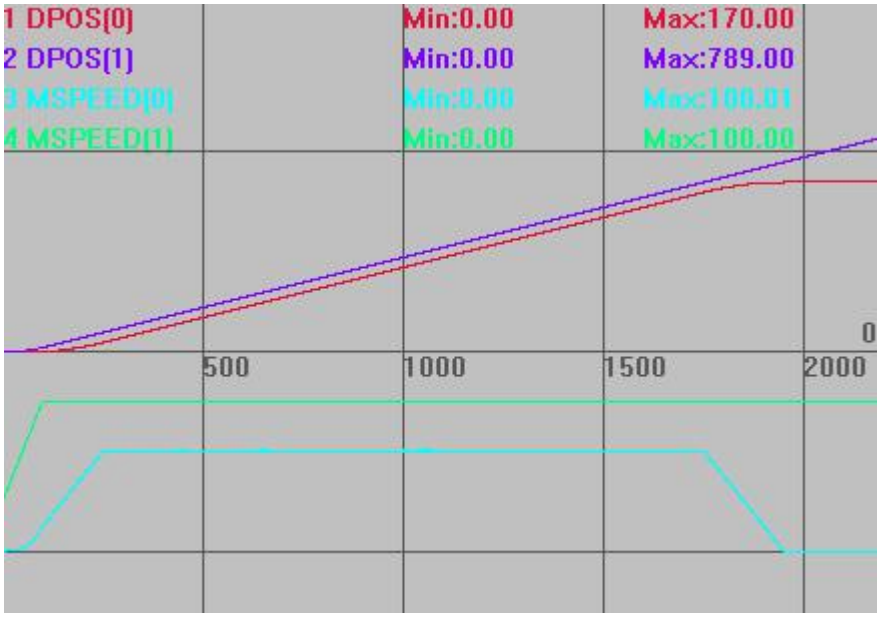
MSPEED(0)垂直刻度 200，偏移-200

MSPEED(1)垂直刻度 200，偏移-150



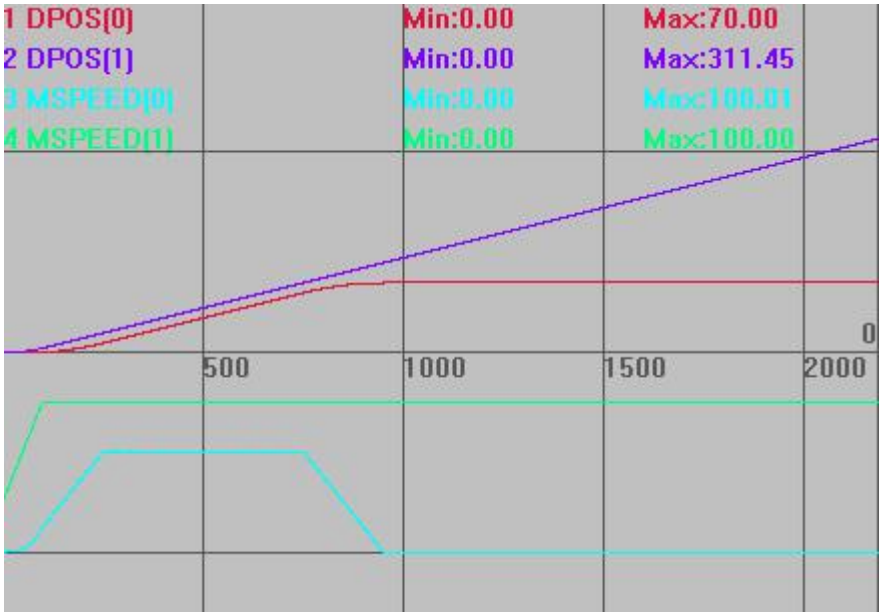
其他条件同上，增加同步距离

```
MOVELINK(10,20,20,0,1)          '工作台加速阶段
MOVELINK(100,100,0,0,1)  '同步阶段 100
MOVELINK_MODIFY = 50            '修改同步段为 100+50
MOVELINK(10,20,0,20,1)          '减速阶段
```



其他条件同上，减少同步距离

```
MOVELINK(10,20,20,0,1)        '工作台加速阶段
MOVELINK(100,100,0,0,1)       '同步阶段 100
MOVELINK_MODIFY = -50         '修改同步段为 100-50
MOVELINK(10,20,0,20,1)        '减速阶段
```



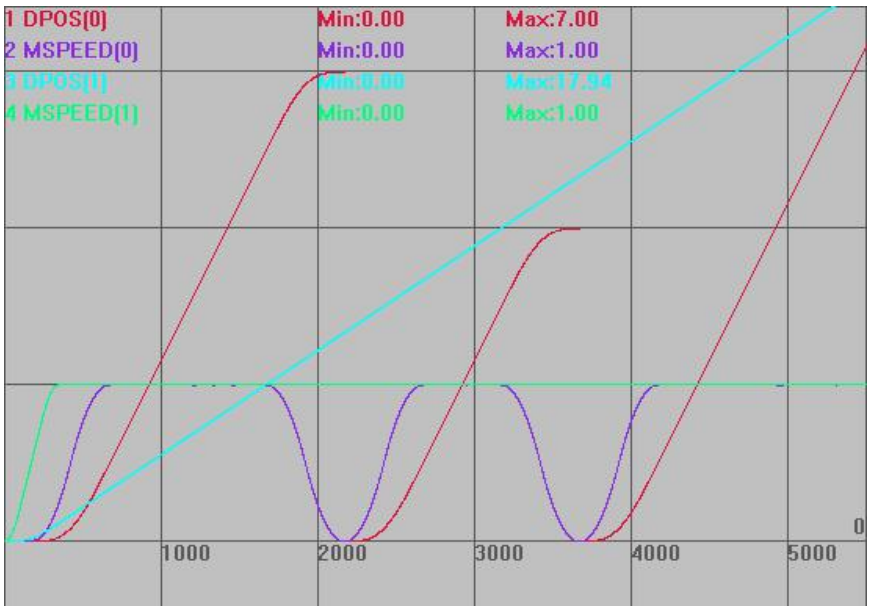
注意，只能在同步段后使用此指令，在加减速段使用会报错,无法修改。

```
MOVELINK(10,20,20,0,1)   '工作台加速阶段
MOVELINK_MODIFY = 50
MOVELINK(100,100,0,0,1)  '同步阶段 100
```

```
Axis:0 MOVELINK_MODIFY:50.000 failed.
```

例二 从轴追剪轴采用 S 曲线加减速

```
RAPIDSTOP(2)
WAIT    IDLE(0)
WAIT    IDLE(1)
DATUM(0)

BASE(0,1)
UNITS=10000,10000
ATYPE=0,0
DPOS=0,0
SPEED=1,1     '型材运行速度 1m/s,60m/min
ACCEL=2,2
DECEL=2,2
SRAMP=200,200

STOPTASK 1
RUNTASK 1,Task_FlyShear
DELAY(200)

VMOVE(1)    AXIS(1)        '型材持续运动
TRIGGER                    '自动触发示波器
END

Task_FlyShear:
WHILE 1
      BASE(0)
      'MOVELINK_MODIFY=0              '先清空
      MOVELINK(3,4,1,1,1,8)    AXIS(0)
      WAIT IDLE(0)

      BASE(0)
      DPOS=0
      'MOVELINK_MODIFY=0              '先清空
      MOVELINK(3,4,1,1,1,8)    AXIS(0)
      WAIT    UNTIL MPOS(0)>1          '等待跟随轴距离>2
      MOVELINK_MODIFY=-1             '把跟随轴距离减少 1
      WAIT UNTIL MOVELINK_MODIFY=0 '等待同步偏移完成
      WAIT    IDLE(0)

      BASE(0)
      DPOS=0
      'MOVELINK_MODIFY=0             '先清空
      MOVELINK(3,4,1,1,1,8)    AXIS(0)
      WAIT UNTIL MPOS(0)>1         '等待跟随轴距离>2
```

	MOVELINK_MODIFY=1 '把跟随轴距离增加 1 WAIT UNTIL MOVELINK_MODIFY=0 '等待同步偏移完成 WEND 运动轨迹和速度曲线 DPOS(0)垂直刻度 1，无偏移 MSPEED(0)垂直刻度 1，无偏移 DPOS(1)垂直刻度 3，无偏移 MSPEED(1)垂直刻度 1，无偏移 
相关指令	MOVELINK

MOVESYNC -- 同步运动

类型	运动设置指令
描述	**同步运动，皮带上物体跟随，此运动非插补运动，不保证运动轨迹为直线。** 要求皮带轴与 BASE 跟随轴的长度单位是一样的。 当 BASE 轴完成跟随动作后，此指令结束，结束时如果皮带物体相对感应位置已经运动了一段距离， BASE 轴并不位于 pos 的绝对位置，并且以跟随速度运行中。 MOVESYNC 支持连续使用，会自动保证速度连续，中间可以插入 MOVE_OP 指令，为了避免运动结束时高速跟随中直接停止，最后一条 MOVESYNC 请使用-1 模式。 此指令属于凸轮指令，不支持运动暂停。
语法	MOVESYNC(mode,synctime,syncposition,syncaxis,pos1[,pos2, pos3…]) mode：模式

模式	描述
-1	同步结束模式，运动到指定的绝对位置，此模式运动如果后面紧接着

	其它 MOVESYNC 指令，会被覆盖,此模式下 syncaxis 无效
-2	强制结束模式，调用时强制停止原来的 MOVESYNC，运动到指定结束位置，此模式运动如果后面紧接着其它 MOVESYNC 指令，会被覆盖,此模式下 syncaxis 无效
0	BASE 第 1 个轴（x）跟随皮带轴物体
10	BASE 第 2 个轴（y）跟随皮带轴物体
20	BASE 第 3 个轴跟随皮带轴物体

mode=0+angle，angle：皮带旋转角度，角度 = 皮带与 BASE 第 1/2 轴的正向旋转夹角。例如 Mode = PI/4，皮带在 45 度的方向；Mode=PI/2，皮带在 y 方向；Mode=PI，皮带在 x 负向；Mode=(PI*1.75)，皮带在-45 度的方向

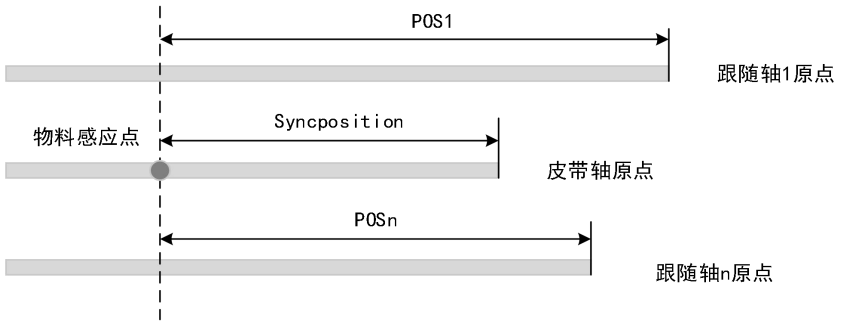
synctime：同步时间，ms 单位，本运动在指定时间内完成，完成时 BASE 轴跟上皮带且保持速度一致。0 表示根据运动轴的速度加速度来估计同步时间，可能不准确

syncposition：皮带轴物体被感应到时皮带轴的位置，此指令支持皮带轴坐标循环，但是在指令被调用时确保此参数位置和当前皮带轴位置之间没有发生坐标修改或循环操作，因此此指令调用时不要在坐标循环点附近

syncaxis：皮带轴轴号，-1 表示没有皮带轴，直接运动到 pos1 的位置

pos1：皮带轴物体被感应到时的 BASE 第 1 个轴绝对位置

posn：皮带轴物体被感应到时的 BASE 第 n 个轴绝对位置



适用控制器

4 系列 170601 以上固件支持。

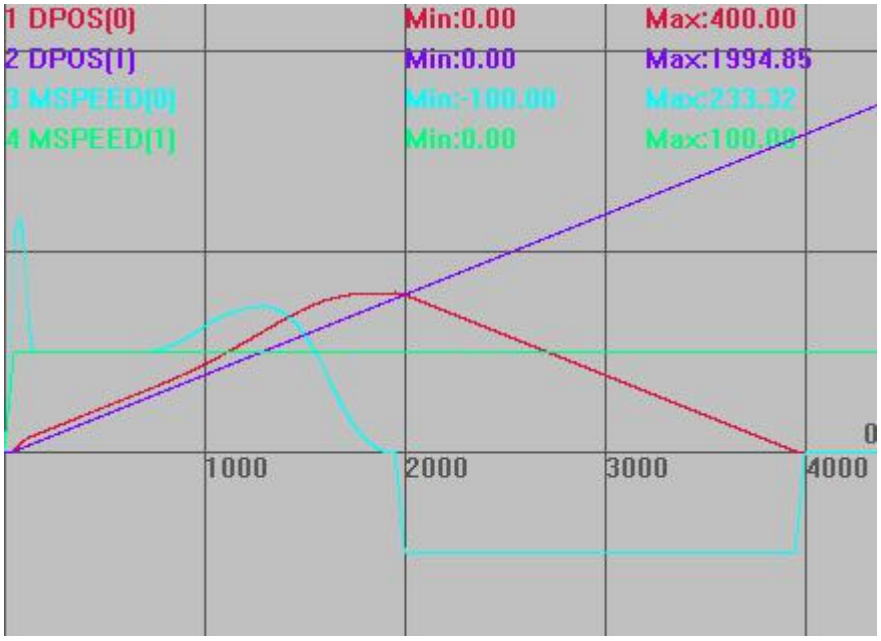
例子

例一 皮带取料

```
RAPIDSTOP(2)
WAIT  IDLE(0)
WAIT  IDLE(1)
BASE(0,1)
DPOS=0,0
UNITS=100,100
ATYPE=1,1
SPEED=100,100
ACCEL=1000,1000
DECEL=1000,1000
TRIGGER
MOVESYNC(0, 0, 100, 1, 120)  '同步到皮带物体上
MOVE_OP(1,1) '下降，如果 Z 轴下降,可以用 MOVESYNC 运动指令代替
MOVE_OP(0,1) '开吸嘴
```

MOVESYNC(0, 1000, 100, 1, 120) '继续同步 1s
MOVE_OP(1,0) '上升
MOVESYNC(-1, 0, 0, -1, 400) '走到放料位置 400 处
MOVE_OP(1,1) '下降
MOVE_OP(0,0) '关吸嘴
MOVE_DELAY(2) '延时 2ms, MOVESYNC 连续运动中不能插入此类语句
MOVE_OP(1,0) '上升
MOVEABS(0) '回到原点
VMOVE(1) AXIS(1) '皮带轴运动

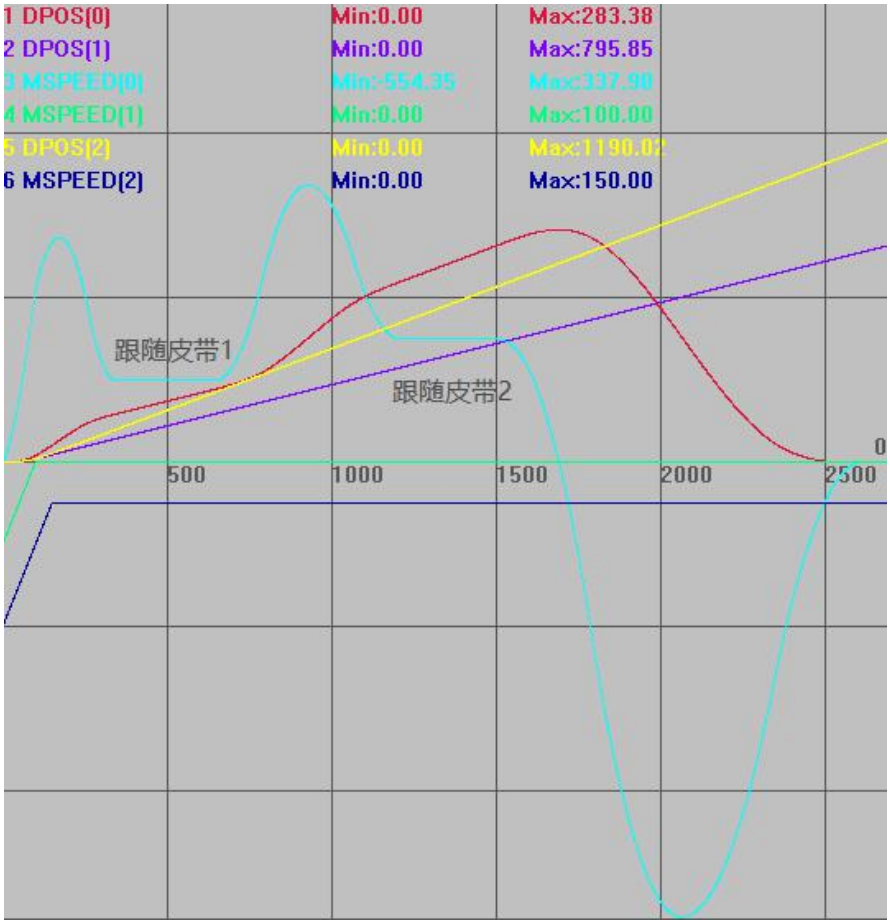
运动轨迹和速度曲线
DPOS(0)垂直刻度 500，无偏移
DPOS(1)垂直刻度 500，无偏移
MSPEED(0)垂直刻度 200，无偏移
MSPEED(1)垂直刻度 200，无偏移



例二 皮带取料，放另外的皮带上
RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
BASE(0,1,2)
DPOS=0,0,0
UNITS=100,100,100
ATYPE=1,1,1
SPEED=1000,100,150 '设置不同速度
ACCEL=1000,1000,1000
DECEL=1000,1000,1000
TRIGGER
MOVESYNC(0, 0, 50, 1, 80) '同步到皮带物体上

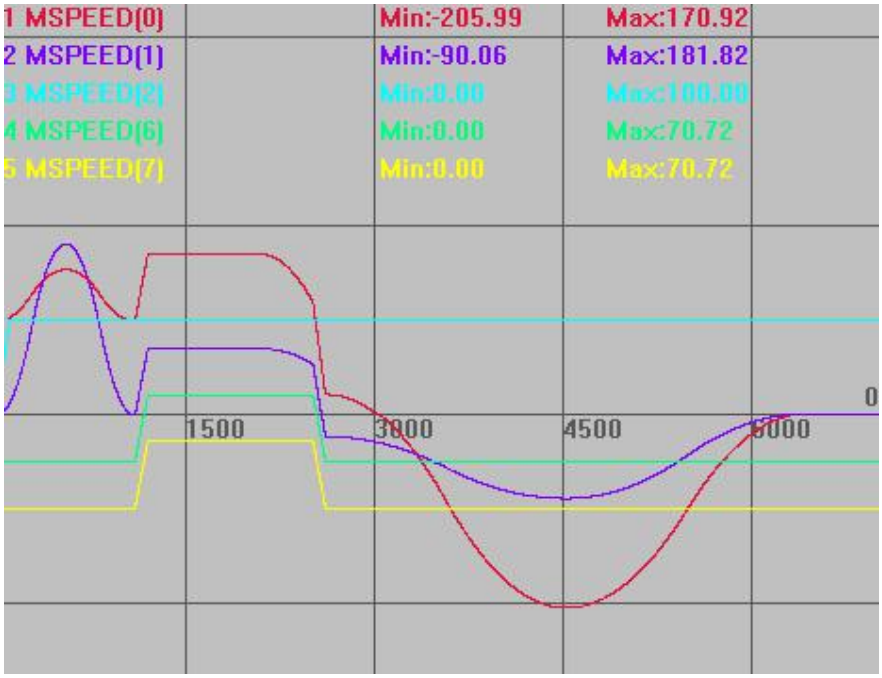
MOVE_OP(0,1) '开吸嘴
MOVE_OP(1,1) '下降,如果 Z 轴下降,可以用 movesync 运动指令代替
MOVESYNC(0, 300, 50, 1, 80) '继续同步 2ms
MOVE_OP(1,0) '上升
MOVESYNC(0, 0, 100, 2, 150) '走到第二个皮带上对应位置
MOVE_OP(1,1) '下降
MOVE_OP(0,0) '关吸嘴
MOVESYNC(0,300, 100, 2, 150) '同步 2ms
MOVE_OP(1,0) '上升
MOVESYNC(-1, 0, 0, -1, 0) '走到停止位置
VMOVE(1) AXIS(1) '皮带轴 1 运动
VMOVE(1) AXIS(2) '皮带轴 2 运动

运动轨迹和速度曲线
DPOS(0)垂直刻度 200，无偏移
DPOS(1)垂直刻度 200，无偏移
MSPEED(0)垂直刻度 200，无偏移
MSPEED(1)垂直刻度 200，偏移-200
DPOS(2)垂直刻度 200，无偏移
MSPEED(2)垂直刻度 200，偏移-200

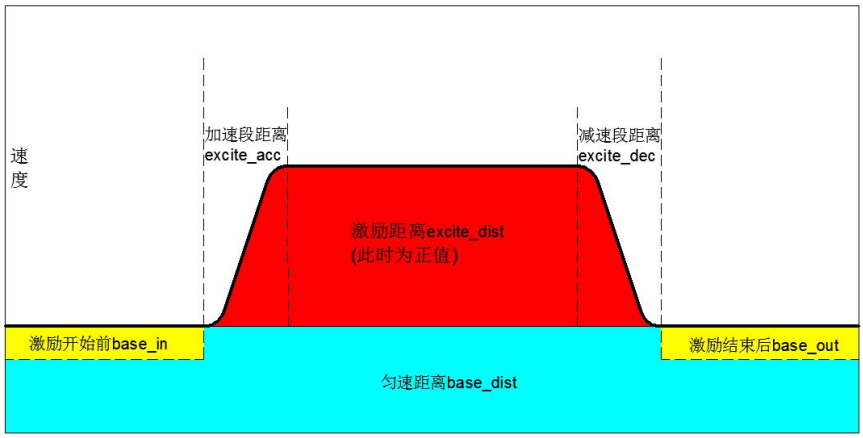


例三 皮带物体上雕刻

```
RAPIDSTOP(2)
WAIT  IDLE(0)
WAIT  IDLE(1)
BASE(0,1,2,6,7)
UNITS=100,100,100,100,100
DPOS=0,0,0,0,0
SPEED=100,100,100,100,100
ACCEL=1000,1000,1000,1000,1000
DECEL=1000,1000,1000,1000,1000
TRIGGER
ADDAX(6)  AXIS(0)  '在虚拟轴上雕刻,叠加到实际轴上
ADDAX(7)  AXIS(1)
BASE(0, 1)
MOVESYNC(0, 0, 50, 2, 80 ,100)  '同步到皮带物体上
MOVE_TASK(1, task1)  '触发叠加轴雕刻
MOVESYNC(0, 1000, 50, 2, 80 ,100)  '较长雕刻运动时间
MOVESYNC(-1, 0, 0, -1, 0 ,0)  '走到停止位置
VMOVE(1)  AXIS(2)  '皮带轴运动
END

TASK1:
    DELAY (2)      '叠加雕刻过程中绝对运动指令位置会不准,延时避开指令调用
    BASE(6, 7)
    MOVE(100,100)  '雕刻，双面划线
    WAIT IDLE  '等待雕刻结束
    BASE(0, 1)
    MOVESYNC(-2, 0, 0, -1, 0 ,0)  '雕刻结束强制走到停止位置

运动轨迹和速度曲线
MSPEED(0)垂直刻度 200，无偏移
MSPEED(1)垂直刻度 200，无偏移
MSPEED(2)垂直刻度 200，无偏移
MSPEED（6）竖直刻度 200，偏移-50
MSPEED（7）竖直刻度 200，偏移-100
```



FLEXLINK -- 激励运动

类型	同步运动指令
描述	**此指令用于轴的激励运动，分为匀速运动和激励运动两部分。**  请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。
语法	FLEXLINK(base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, link_axis, link_options, [start_pos], [link_offpos]) base_dist：跟随轴匀速运动距离 excite_dist：跟随轴激励运动距离，+值表示增加，-值表示减少，跟随轴运动总距离=base_dist+excite_dist link_dist：整个指令过程，跟随轴运动完成，主轴移动的距离 base_in：激励开始前，跟随轴运动距离占 base_dist 的百分比

base_out：激励完成后，跟随轴剩余运动距离占 base_dist 的百分比（两者相加不要超过 100%，否则 excite_dist 无效）

excite_acc：激励过程中，跟随轴加速阶段运动距离占 excite_dist 的百分比，excite_dist 为负值时，为减速阶段

excite_dec：激励过程中，跟随轴减速阶段运动距离占 excite_dist 的百分比，excite_dist 为负值时，为加速阶段

(以上 4 个参数在 excite_dist 不为 0 时生效)

link_axis：主轴轴号

link_options：与参考轴的连接方式，不同的二进制位代表不同的意义

位	描述
bit0	当参考轴锁存信号事件 MARK 触发
bit1	当参考轴运动到设定的绝对位置时，当前轴与参考轴开始连接运动
bit2	自动重复连续双向运行(通过设置 REP_OPTION=1，可以取消重复)
bit4	凸轮中间启动
bit5	只有参考轴的正向运动才连接
bit8	markb 启动

start_pos：绝对位置触发

link_offpos：凸轮中间启动

适用控制器

4 系列 20170518 以上固件版本支持。

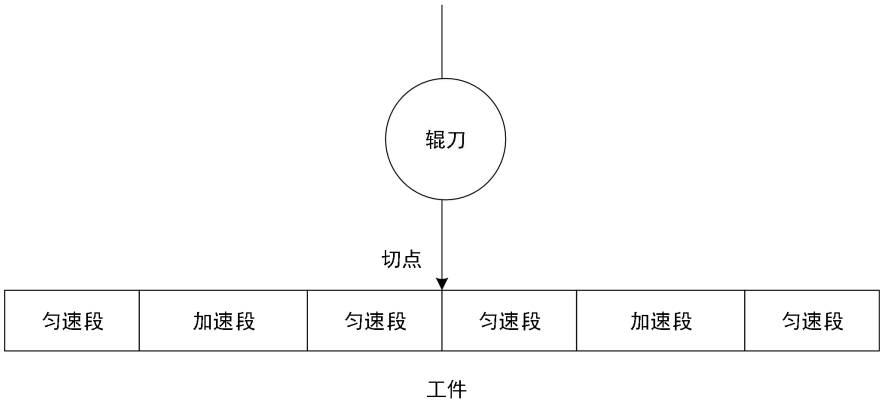
3 系列 20161212 以上固件版本支持。

例子

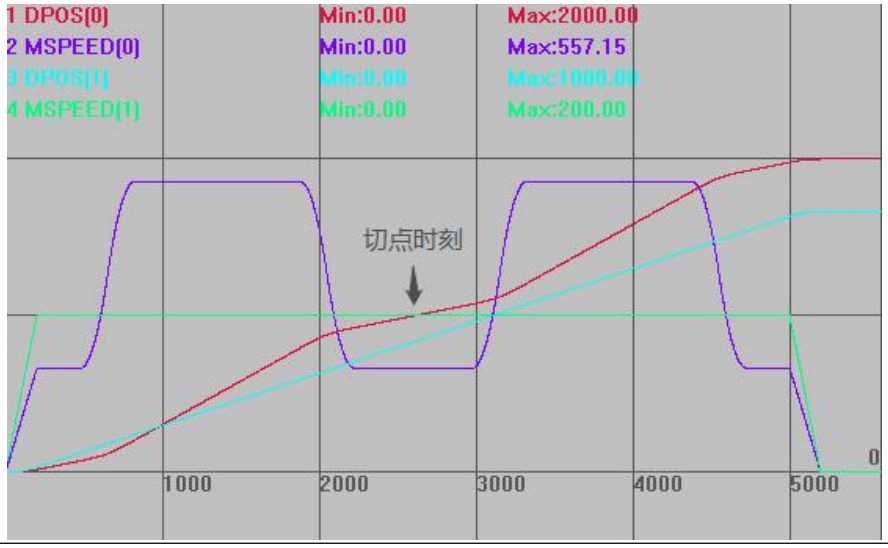
轮切应用

假设辊刀切一次周长 500，工件长度 1000，辊刀恒速，移动距离小于工件长度，此时工件需要有加速过程。设置工件前后为匀速段，中间为加速段。

实际应用中一般是辊刀变速，这里为了演示方便，使用工件变速。




```
RAPIDSTOP(2)
WAIT  IDLE(0)
WAIT  IDLE(1)
BASE(0,1)
DPOS=0,0
UNITS=100,100
SPEED = 200,200
ACCEL=1000,1000
DECEL=1000,1000
```

	TRIGGER **FLEXLINK**(500,500,500,15,15,20,20,1) AXIS(0) '辊刀运行 500 距离，工件运行 1000 距离，在切点时刻保证速度一致，且对应一个周期位置 **FLEXLINK**(500,500,500,15,15,20,20,1) AXIS(0) MOVE(1000) AXIS(1) 运动轨迹和速度曲线 DPOS(0)垂直刻度 1000，无偏移 MSPEED(0)垂直刻度 300，无偏移 DPOS(1)垂直刻度 600，无偏移 MSPEED(1)垂直刻度 200，无偏移 

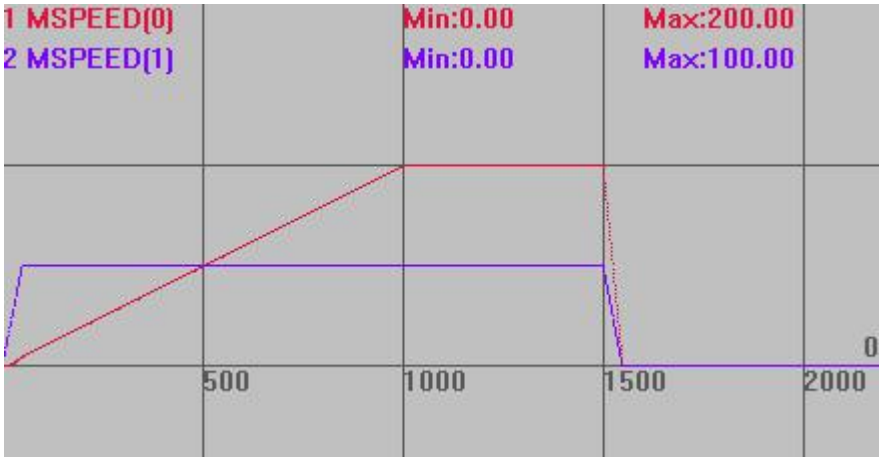
6.5 运动设置指令

CLUTCH_RATE -- 连接速度

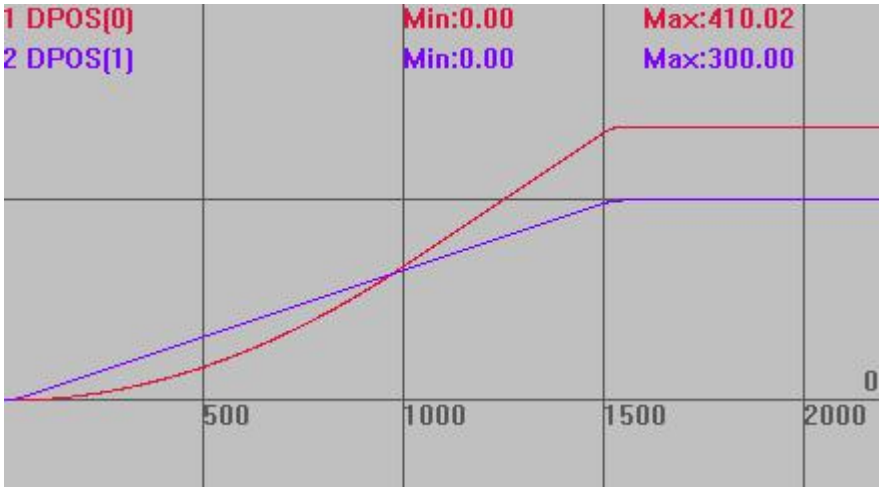
类型	轴参数
描述	**connect 连接的速度，默认值 1000000。** 用于定义连结率从 0 到设置倍率的改变时间，ratio/秒，见例一。 设置值如果不能远大于 CONNECT 连接比例的话，实际连接比例会减小，见例一位移曲线图。 当设置为 0 时，根据跟随轴的速度/加速度参数来跟踪连接，比较适合于手轮运动（当速度不够高时可能导致要持续运动一段时间才能结束）。
语法	CLUTCH_RATE= value
适用控制器	通用
例子	例一 BASE(0,1) ATYPE=1,1 DPOS=0,0

UNITS=100,100
SPEED=100,100
ACCEL=1000,1000
DECEL=1000,1000
CLUTCH_RATE=1 '设置连接速度为 1ratio/s
TRIGGER '自动触发示波器
CONNECT(2,1) AXIS(0) '连接倍率为 2，需要 2 秒建立连接
MOVE(300) AXIS(1) '运动轴 1，轴 0 跟随

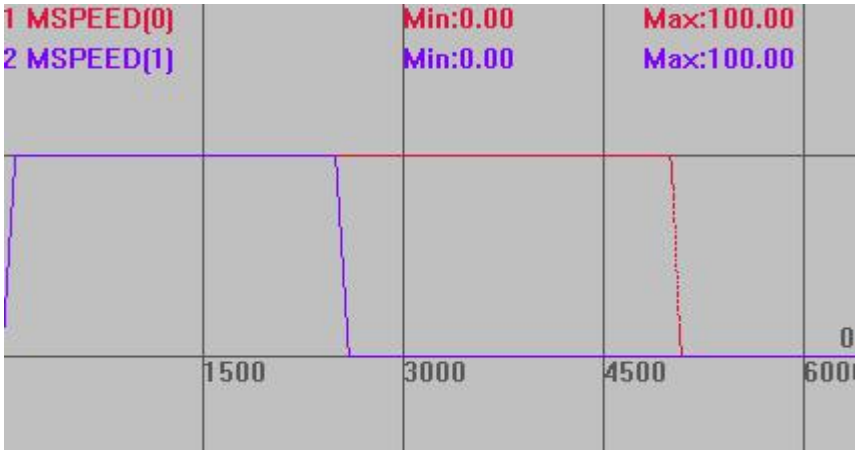
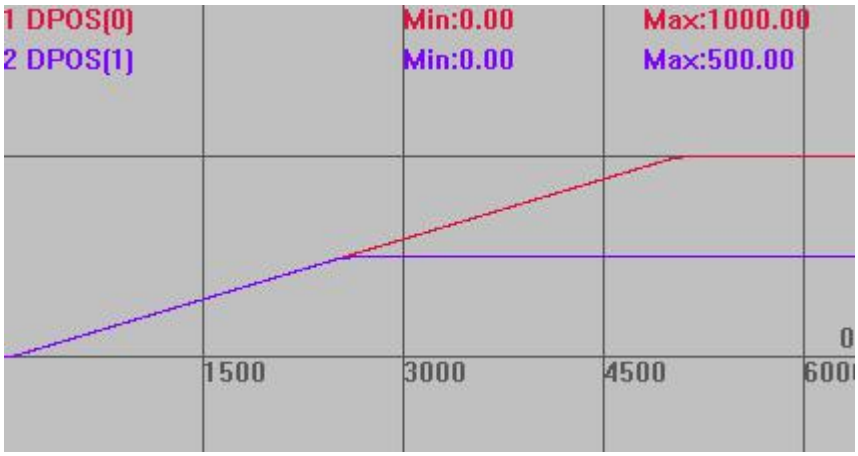
速度曲线，连接时间根据连接比例和 clutch_rate 确定
MSPEED(0)垂直刻度 200
MSPEED(1)垂直刻度 200



位移曲线，CLUTCH_RATE 值较小，实际运动比例并没有达到 2:1
DPOS(0)垂直刻度 300
DPOS(1)垂直刻度 300



例二
BASE(0,1)
DPOS=0,0
ATYPE=1,1
UNITS=100,100

	SPEED=100,100 ACCEL=500,500 DECEL=500,500 **CLUTCH_RATE** = 0 '根据跟随轴的速度/加速度来跟踪连接，不一定同步 TRIGGER '自动触发示波器 CONNECT(2,1) AXIS(0) '此时连接时间根据跟随轴速度加速度确定，本例中 0.2s MOVE(500)AXIS(1) 速度曲线 MSPEED(0)垂直刻度 100 MSPEED(1)垂直刻度 100  位移曲线 DPOS(0)垂直刻度 1000 DPOS(1)垂直刻度 1000 
相关指令	CONNECT

ENCODER_RATIO -- 编码器齿轮比

类型	运动设置指令

描述	**编码器输入齿轮比，缺省(1,1)，设置负值可切换方向**
语法	ENCODER_RATIO(mpos_count, input_count,[, mode]) mpos_count：分子，不要超过 65535 input_count：分母，不要超过 65535 mode：模式 模式 | 描述 1 | AB 1X 模式 2 | AB 2X 模式 4 | AB 4X 模式 必须先设置 ATYPE，再调用 mode 设置。 ZMC406 20170502 以后固件支持。
适用控制器	通用
例子	**ENCODER_RATIO**(4,1)　'编码器 4 倍输入
相关指令	PP_STEP，ENCODER

STEP_RATIO -- 电机齿轮比

类型	运动设置指令
描述	**设置步进输出齿轮比，缺省(1,1)。范围 1-65535。** 设置为负值可修改电机方向，但一般不建议。脉冲电机可用 INVERT_STEP 指令；总线电机最好在驱动器上修改。 建议不要随便修改此参数，可以通过修改脉冲当量来达到相同的效果。
语法	STEP_RATIO(output_count, input_count) output_count：分子，不要超过 65535 input_count：分母，不要超过 65535
适用控制器	通用
例子	**STEP_RATIO** (16,1) '16 倍脉冲个数输出，把脉冲当量乘以 16 也可以达到同样的效果

BACKLASH -- 反向间隙补偿

类型	运动设置指令
描述	**设置轴的反向间隙，扩展轴无效。**
语法	BACKLASH(enable [,dist[,speed,acceleration]]) enable：使能与否 dist：距离，units 单位 speed：反向间隙速度 acceleration：反向间隙加速度
适用控制器	通用

例子	**BACKLASH**(0)　　'关闭反向间隙功能 **BACKLASH**(1,0.1)　　'设置反向间隙 0.1mm

PITCHSET -- 螺距补偿

类型	运动设置指令
描述	**设置轴的螺距补偿，扩展轴无效。** 每点的补偿脉冲个数存储在 TABLE 表里面。
语法	PITCHSET(enable [,startpos,disone,maxpoint,tablenum]) enable：使能与否 startpos：开始补偿的 mpos 位置，units 单位，startpos 对应的点不存储 disone：每个点之间的距离，units 单位 maxpoint：补偿的总点数 tablenum：螺距补偿表存储的 table 位置，从 startpos 下一个点开始存储，脉冲数单位
适用控制器	通用
例子	BASE(0) ATYPE=1 UNITS=100 SPEED=100 ACCEL=100 TABLE(0,10,20,30,40,-10,-20,-30,-40)　'TBALE 存贮螺距补偿值，补偿值是脉冲个数，不是补偿距离值 DPOS=0 **PITCHSET(1,300,100,8,0)**　'MPOS=300 时，开始补偿 8 个点，间隔 100 TRIGGER MOVE(1500)

开始补偿点位置（MPOS）	补偿值（脉冲个数）
300	-
400	10 个脉冲
500	20 个脉冲
600	30 个脉冲
700	40 个脉冲
800	-10 个脉冲（反向补偿脉冲）
900	-20 个脉冲（反向补偿脉冲）
1000	-30 个脉冲（反向补偿脉冲）
1100	-40 个脉冲（反向补偿脉冲）

6.6 机械手指令

CONNFRAME -- 机械手逆解

<table>
<tr><td>类型</td><td>同步运动指令</td></tr>
<tr><td>描述</td><td>将当前关节坐标系的目标位置与虚拟坐标系的位置关联。
CLUTCH_RATE=0 时关节坐标系的运动速度和加速度受 SPEED 等参数的限制。

⚠ 当关节轴告警等出错时，此运动会被 CANCEL。

⚠ 不要在虚拟轴高速运动中 CANCEL 此运动，会引起轴高速运动中停止。

⚠ 此命令 LOAD 时会自动修改虚拟轴的坐标，使得与关节轴一致，因此调用后需要 WAIT LOADED 后才开始运动。
⚠ 连接过程中不要修改虚拟轴的坐标，或是调用 DATUM 等可能改坐标的运动，这样会导致关节轴快速运动到新的虚拟位置。
⚠ CONNFRAME 生效后，关节轴的 MTYPE 为 33，此时无法直接运动关节轴，必须通过运行虚拟轴来间接运动关节轴，当要直接移动关节轴时，先调用 CANCEL 取消 CONNFRAME，再运动关节轴。
⚠ 当虚拟轴和实际轴都是旋转轴时，例如末端旋转轴，虚拟轴和实际轴的脉冲当量注意要一致。</td></tr>
<tr><td>语法</td><td>CONNFRAME(frame，tablenum, viraxis0, viraxis1,[…])
frame：坐标系类型，1- scara（如需针对特殊的机械手类型定制，请联系厂家）
tablenum：存储转换参数的 TABLE 位置；frame=1 时，依次存放：第一个关节轴长度，第二个关节轴长度，第一个关节轴一圈脉冲数，第二个关节轴一圈脉冲数
viraxis0：虚拟坐标系第一个轴
viraxis1：虚拟坐标系第二个轴
[…]：虚拟坐标系第 N 个轴，此运动轴可以是实际轴，具体由机械手类型确定

FRAME 机械结构一览表
详细说明请查看《正运动机械手指令说明》文档。
其他特殊的机械结构如需支持请联系厂家。
<table>
<tr><td>frame</td><td>结构类型</td></tr>
<tr><td>1</td><td>标准 SCARA 机械手</td></tr>
<tr><td>101</td><td>SCARA+摆动，虚拟轴定义 4 个</td></tr>
<tr><td>105</td><td>SCARA+摆动，虚拟轴定义 5 个</td></tr>
<tr><td>106</td><td>特殊 SCARA</td></tr>
<tr><td>107</td><td>特殊 SCARA</td></tr>
<tr><td>108</td><td>特殊 5 轴 SCARA</td></tr>
</table>
</td></tr>
</table>

		11	旋转台
		17	双旋转台
		18	偏移旋转台
		19	偏移双旋转台
		3	码垛机械手
		103	码垛变形，喷涂机械手
		5	旋转伸缩机械手
		15	XY 滑台
		102	2 轴 delta
		2	3 轴 delta，R 类型控制器支持
		12	4 轴 delta，R 类型控制器支持
		13	3 轴垂直蜘蛛手，R 类型控制器支持
		25	5 关节机械手
		6	6 自由度机械手，R 类型控制器支持
		26	特殊 6 自由度
		36	特殊 6 自由度
		100	XYZ+2 轴手腕，虚拟轴定义 3 个
		104	XYZ+2 轴手腕，虚拟轴定义 5 个
适用控制器	通用		
例子	参见 FRAME_ROTATE 指令例程		
相关指令	CONNREFRAME		

CONNREFRAME -- 机械手正解

类型	同步运动指令
描述	**将虚拟轴的坐标与关节轴的坐标关联，关节轴运动后，虚拟轴自动走到相应的位置。** 此指令为 CONNFRAME 的反运动指令。 ⚠当虚拟轴 CONNREFRAME 运动 LOAD 时，关节轴的 CONNFRAME 运动会自动被 CANCEL。 ⚠当关节轴的 CONNFRAME 运动 LOAD 时，虚拟轴 CONNREFRAME 运动会自动被 CANCEL。
语法	CONNREFRAME(frame，tablenum, axis0, axis1,[…]) frame：坐标系类型，1- scara（如需针对特殊的机械手类型定制，请联系厂家） tablenum：存储转换参数的 TABLE 位置；frame=1 时，依次存放：第一个关节轴长度，第二个关节轴长度，第一个关节轴一圈脉冲数，第二个关节轴一圈脉冲数 axis0：关节坐标系第一个轴 axis1：关节坐标系第二个轴 […]：关节坐标系第 N 个轴 BASE 轴与参数里面的轴的位置相反。

适用控制器	通用
例子	L1=10 '第一个关节轴长度 L2=10 '第二个关节轴长度 BASE(0,1) '假设关节轴号是 0/1 UNITS=10,10 '这里以度为单位 DPOS=0,0 '设置关节轴的位置，此处要根据实际情况来修改 BASE(6,7) ATYPE=0,0 '设置为虚拟轴 TABLE(0,L1,L2,3600,3600) '参数存储从 TABLE(0)开始存储，电机一圈 360 个脉冲 UNITS=1000,1000 '此脉冲当量要提前设置，中途不能变化 CONNREFRAME(1,0,0,1) '轴 6/7 的坐标根据轴 0/1 来计算运动关节轴 BASE(0,1) MOVEABS(90,0) '虚拟轴的坐标变为 0,20
相关指令	CONNFRAME

FRAME -- 机械手类型

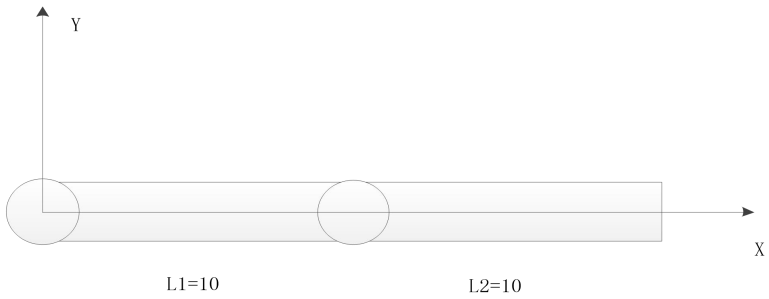
类型	机械手指令
描述	**选择机械手类型**，请查看正运动机械手手册。
相关指令	CONNFRAME

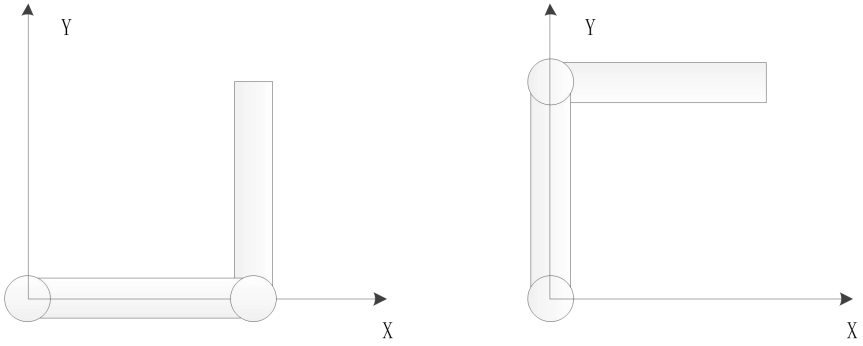
FRAME_STATUS -- 机械手轴状态

类型	机械手指令
描述	**指明当前的机械手姿态。** 不是机械手状态时返回-1，FRAME_TRANS2 指令会用到这个姿态。 只对 SCARA、类 SCARA 和 6 自由度结构有多种姿态。 SCARA 左手姿态值为 0，右手姿态值为 1。
语法	VAR1=FRAME_STATUS（AXIS）
适用控制器	通用
例子	在线命令输入?FRAME_STATUS，打印出当前姿态。 >>?**FRAME_STATUS**
相关指令	BASE

FRAME_TRANS2 -- 正逆解坐标转换

类型	机械手计算指令
描述	**坐标转换函数。**

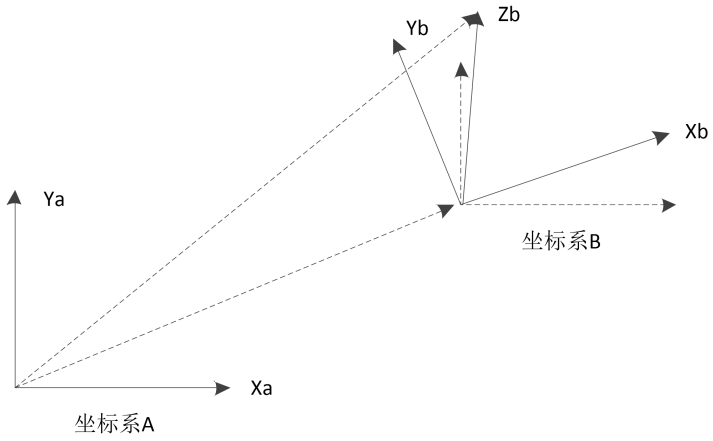
<table>
<tr><td></td><td>使用时必须 base 正确的轴号。逆解时，base 虚拟轴；正解时，base 关节轴。
按照正确的顺序填入相应个数的数据。填入的个数与数据，与?*frame 一致。</td></tr>
<tr><td>语法</td><td>FRAME_TRANS2(tablein，tableout, dir)
tablein：table 数组中的索引号，从这个索引开始连续存储数据，正解时输入关节坐标，逆解时输入虚拟轴坐标列表，最后附加姿态
tableout：table 这个索引开始存储数据，正解时输出虚拟坐标，最后附加姿态，逆解时输出关节坐标列表
dir：模式选择
<table>
<tr><td>模式</td><td>类型</td><td>描述</td></tr>
<tr><td>0</td><td>逆解</td><td>虚拟轴到关节轴，无姿态，自动使用当前的姿态</td></tr>
<tr><td>1</td><td>正解</td><td>关节轴到虚拟轴，无姿态输出</td></tr>
<tr><td>2</td><td>逆解</td><td>输入虚拟轴坐标，最后加上姿态</td></tr>
<tr><td>3</td><td>正解</td><td>输出虚拟轴坐标，输出最后一个位置填姿态</td></tr>
</table></td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>以 scara 结构为例，第一关节轴 L1=10，第二关节轴 L2=10。table(100)作为输入坐标的存放位置，table(200)作为输出坐标的存放位置。
建立连接后，关节轴原点坐标（0,0），此时虚拟轴坐标为（20,0），如下图。
 

虚拟轴坐标 (10,10) 时，有两种姿态
关节轴 (0,90) 和关节轴 (90,-90)
 

坐标转换表
<table>
<tr><td>状态</td><td>BASE 轴</td><td>输入 X、Y、姿态</td><td>使用指令模式</td><td>输出关节坐标</td></tr>
<tr><td rowspan="6">逆解</td><td rowspan="6">虚拟轴</td><td>table(100,20,0,0)</td><td rowspan="2">frame_trans2(100,200,0)</td><td>table(200,0,0)</td></tr>
<tr><td>table(100,10,10,1)</td><td>table(200,0,90)</td></tr>
<tr><td>table(100,10,10,1)</td><td>frame_trans2(100,200,2)</td><td>table(200,90,-90)</td></tr>
<tr><td>输入关节坐标</td><td>使用指令模式</td><td>输出 X、Y、姿态</td></tr>
<tr><td>table(100,0,0)</td><td>frame_trans2(100,200,1)</td><td>table(200,20,0,0)</td></tr>
</table></td></tr>
</table>

		table(100,0,90)		table(200,10,10,0)
		table(100,90,-90)		table(200,10,10,0)
		table(100,90,-90)	frame_trans2(100,200,3)	table(200,10,10,1)

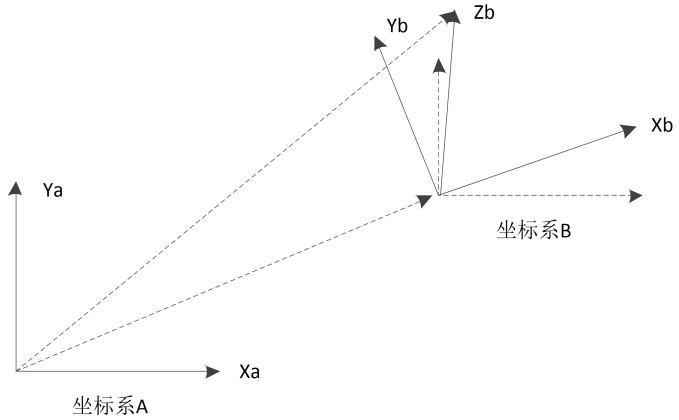
FRAME_ROTATE -- 工件坐标系变换

类型	机械手计算指令
描述	**用于对工件坐标系进行平移和旋转。** 目前只对 FRAME6 同时旋转姿态，其他具有 XYZ 虚拟轴的可以支持 XYZ 三个轴旋转，姿态轴不能旋转。 旋转后，虚拟轴 WORLD_DPOS 表示世界坐标不会改变，虚拟轴 DPOS 表示工件坐标会改变。 使用时需控制器当前存在机械链接。 BASE 轴可以是虚拟轴与关节轴任意一个；BASE 轴无机械手链接时会报 1025 的错误。多种机械手叠加的情况，根据 BASE 轴来识别是哪个机械手模式；如果 BASE（axis_1,axis_2）中的 axis_1 是模式 1 的机械手轴，axis_2 是模式 2 的机械手轴，那么结果是模式 1 的机械手进行坐标计算，即以 BASE 轴的顺序来计算。
语法	FRAME_ROTATE(X,Y,Z,RX,RY,RZ) X：坐标系 B 沿 X^的平移距离 Y：坐标系 B 沿 Y^的平移距离 Z：坐标系 B 沿 Z^的平移距离 RX：坐标系 B 沿 X^的旋转的角度 RY：坐标系 B 沿 Y^的旋转的角度 RZ：坐标系 B 沿 Z^的旋转的角度  坐标系旋转： 旋转的方法是：x_y_z 固定角坐标系。

	首先将做坐标系{B}和一个已知参考坐标系{A}重合。先将{A}绕 Xa 旋转 RX 角，在绕 Ya 旋转 RY 角，最后绕 Za 旋转 RZ 角。
适用控制器	通用
例程	例一 以 FRAME=2，DETLA 为例，对其绕 X 轴旋转 90 度。 BASE(0,1,2) RAPIDSTOP ATYPE = 1,1,1 UNITS=3600/360,3600/360,3600/360 DPOS=0,0,0 BASE(6,7,8) ATYPE = 0,0,0 TABLE(0,40,10,32,85,3600,3600,3600, 0, 0, 0) UNITS = 100,100,100 BASE(0,1,2) CONNFRAME(2,0,6,7,8) WAIT LOADED BASE(6,7,8) **FRAME_ROTATE**(0,0,0,PI/2,0,0) ?"DPOS(7)-WORLD_DPOS(7)=",DPOS(7)-WORLD_DPOS(7) ?"DPOS(8)-WORLD_DPOS(8)=",DPOS(8)-WORLD_DPOS(8) 输出行输出结果： DPOS(7)-WORLD_DPOS(7)=-58.1400 DPOS(8)-WORLD_DPOS(8)=58.1400 例二 以 FRAME=1，SCARA 为例；对其相对于 Z 轴旋转 90 度。 BASE(0,1,2,3) RAPIDSTOP ATYPE = 1,1,1,1 UNITS=3600/360,3600/360,3600/360,1000 DPOS=0,0,0,0 BASE(6,7,8,9) ATYPE = 0,0,0,0 TABLE(0,100,100,3600,3600,3600) UNITS = 100,100,3600/360,1000 BASE(0,1,2,3) CONNFRAME(1,0,6,7,8,9) WAIT LOADED BASE(6,7,8,9) **FRAME_ROTATE**(0,0,0,0,0,PI/2) ?"DPOS(7)-WORLD_DPOS(7)=",DPOS(7)-WORLD_DPOS(7) ?"DPOS(8)-WORLD_DPOS(8)=",DPOS(8)-WORLD_DPOS(8) 输出行输出结果：

	DPOS(7)-WORLD_DPOS(7)=-200 DPOS(8)-WORLD_DPOS(8)=0
相关指令	FRAME_ROTATE2

FRAME_ROTATE2 -- 坐标系变换计算

类型	机械手计算指令
描述	**手动计算坐标系旋转后的坐标值。** 使用时需控制器当前存在机械链接。 base 轴，可以是虚拟轴与关节轴中的任意一个；base 轴无机械手链接，会报 1025 的错误。多种机械手叠加的情况，根据 base 轴来识别是哪个机械手模式。如果 base（axis_1,axis_2）中的 axis_1 是模式 1 的机械手轴，axis_2 是模式 2 的机械手轴，那么结果是模式 1 的机械手进行坐标计算，即以 base 轴的顺序来计算。
语法	FRAME_ROTATE2(tablein, tableout, dir[, x,y,z[, rx,ry,rz]]) ret = FRAME_ROTATE2(tablein, tableout, dir[, x,y,z[, rx,ry,rz]]) tablein：转换前，填写的坐标存储 table 位置 tableout：转换后，输出的坐标存储 table 位置 dir：方向选择 值 | 描述 0 | DPOS 到 WORLD_DPOS 1 | WORLD_DPOS 到 DPOS X：坐标系 B 沿 X^的平移距离 Y：坐标系 B 沿 Y^的平移距离 Z：坐标系 B 沿 Z^的平移距离 RX：坐标系 B 沿 X^的旋转的角度 RY：坐标系 B 沿 Y^的旋转的角度 RZ：坐标系 B 沿 Z^的旋转的角度 [, x,y,z[, rx,ry,rz]]：不填时使用当前的缺省 rotate 参数 Ret：返回成功与否；-1-成功，0-未成功 
适用控制器	通用

例程

例一 以 FRAME=2，DETLA 为例；对其绕 X 轴旋转 90 度。

```
BASE(0,1,2)
RAPIDSTOP
ATYPE = 1,1,1
UNITS=3600/360,3600/360,3600/360
DPOS=0,0,0
BASE(6,7,8)
ATYPE = 0,0,0
TABLE(0,40,10,32,85,3600,3600,3600, 0, 0, 0 )
UNITS = 100,100,100
BASE(0,1,2)
CONNFRAME(2,0,6,7,8)
WAIT LOADED
FOR i=0    TO 2
TABLE(100+i)=DPOS(i+6)
NEXT
BASE(6,7,8)
FRAME_ROTATE(0,0,0,PI/2,0,0)
BASE(6,7,8)
FRAME_ROTATE(0,0,0,PI/2,0,0)
WAIT LOADED
ret=FRAME_ROTATE2(100,200,1,0,0,0,PI/2,0,0)
IF ret=-1 THEN
      ?"计算值"
      ?"DPOS(6)=",TABLE(200)
      ?"DPOS(7)=",TABLE(201)
      ?"DPOS(8)=",TABLE(202)
      ?"比较值"
      ?"DPOS(6)比较",TABLE(200)-DPOS(6)
      ?"DPOS(7)比较",TABLE(201)-DPOS(7)
      ?"DPOS(8)比较",TABLE(202)-DPOS(8)
ENDIF
```

输出行输出结果：

```
计算值
DPOS(6)=0
DPOS(7)=-58.1400
DPOS(8)=0.0000
比较值
DPOS(6)比较 0
DPOS(7)比较 0
DPOS(8)比较 0.0000
```

例二 以 FRAME=1，SCARA 为例，对其相对于 Z 轴旋转 90 度。

	BASE(0,1,2,3)
	RAPIDSTOP
	ATYPE = 1,1,1,1
	UNITS=3600/360,3600/360,3600/360,1000
	DPOS=0,0,0,0
	BASE(6,7,8,9)
	ATYPE = 0,0,0,0
	TABLE(0,100,100,3600,3600,3600)
	UNITS = 100,100,3600/360,1000
	BASE(0,1,2,3)
	CONNFRAME(1,0,6,7,8,9)
	WAIT LOADED
	FOR i=0 TO 3
	TABLE(100+i)=DPOS(i+6)
	NEXT
	BASE(6,7,8,9)
	FRAME_ROTATE(0,0,0,0,0,PI/2)
	WAIT LOADED
	RET=**FRAME_ROTATE2**(100,200,1,0,0,0,0,0,PI/2)
	IF RET=-1 THEN
	?"计算值"
	?"DPOS(6)=",TABLE(200)
	?"DPOS(7)=",TABLE(201)
	?"DPOS(8)=",TABLE(202)
	?"DPOS(9)=",TABLE(203)
	?"比较值"
	?"DPOS(6)比较",TABLE(200)-DPOS(6)
	?"DPOS(7)比较",TABLE(201)-DPOS(7)
	?"DPOS(8)比较",TABLE(202)-DPOS(8)
	?"DPOS(9)比较",TABLE(203)-DPOS(9)
	ENDIF
	
	输出行输出结果：
	计算值
	DPOS(6)=-0.0000
	DPOS(7)=-200
	DPOS(8)=0
	DPOS(9)=0
	比较值
	DPOS(6)比较 -0.0000
	DPOS(7)比较 0
	DPOS(8)比较 0

相关指令	FRAME_ROTATE

WORLD_DPOS -- 世界坐标系

类型	轴状态
描述	**虚拟轴参照世界坐标系的坐标值，没有旋转时与 DPOS 相同。**
语法	var1=WORLD_DPOS(axis)
适用控制器	通用
例程	在线命令打印 >>?*WORLD_DPOS

MOVER_L/MOVER_LABS -- 关节轴直线插补

类型	运动指令
描述	**关节轴直线插补。** 机械手关节插补运动，机械手末端直线运动到指定坐标。 此指令正解模式下使用，直接操作关节轴时会有改变姿态的可能，所以要确保起点结束点姿态要保持一致，否则会报错。
语法	MOVER_L(distance1 [,distance2 [,distance3 [,distance4...]]]) distance1：第一个轴运动距离 distance2：下一个轴运动距离
适用控制器	4 系列 20170511 以上固件支持
例程	BASE(0,1) DPOS=0,0 BASE(6,7) ATYPE = 0,0 '设置为虚拟轴 UNITS=1000,1000 TABLE(0,L1,L2, 100*360, 100*360, 360) CONNREFRAME(1,0,0,1) '第 6/7 轴作为虚拟的 XY 轴，启动连接 WAIT LOADED '关节运动 BASE(0,1) SPEED=400 SRAMP=100 ACCEL=1000 DECEL=1000 MERGE = 1 CORNER_MODE=32 '启动倒角 ZSMOOTH=2 MOVEABS(45,90) '关节运动，此时运动关节角度 **MOVER_LABS**(90, 0) '末端直线运动 WAIT IDLE '等待运动停止

	PRINT *DPOS
相关指令	MOVER_C，MOVER_C3

MOVER_C/MOVER_CABS -- 关节轴平面圆弧

类型	运动指令
描述	**关节轴直接走圆弧插补运动。** 此指令正解模式下使用。 BASE 虚拟的 XYZ 轴，否则无法确定 XYZ，此时的参数也是虚拟轴的距离。
语法	MOVER_C/MOVER_CABS（end1,end2,centre1,centre2,mode,[dis1,…,disn]） end1：第 1 个轴运动距离参数 1 end2：第 2 个轴运动距离参数 1 centre1：第 1 个轴运动距离参数 2 centre2：第 2 个轴运动距离参数 2 mode：模式

值	描述
0	当前点，中间点，终点三点定圆弧。 距离参数 1 为终点距离，距离参数 2 为中间点距离。
1	当前点，圆心，终点定圆弧。 走最短的圆弧。 距离参数 1 为终点距离，距离参数 2 为圆心的距离。
2	当前点，中间点，终点三点定圆。 距离参数 1 为终点距离，距离参数 2 为中间点距离。
3	当前点，圆心，终点定圆。 先走最短的圆弧，再继续走完整圆。 距离参数 1 为终点距离，距离参数 2 为圆心的距离。

	dis1- disn：螺旋轴的距离
适用控制器	4 系列 20170511 以上固件支持
例程	L1 = 500 L2 = 500 TABLE(0,L1,L2, 100*360, 100*360, 360) '参数存储在 TABLE0 开始的位置，电机一圈 360 个脉冲 BASE(6,7) CONNREFRAME(1,0,0,1)　'第 6/7 轴作为虚拟的 XY 轴，启动连接 WAIT LOADED　　'等待运动加载 BASE(6,7)　'REFRAME 直接 MOVER 运动虚拟轴，会自动转换到关节轴上 MOVER_LABS(500) **MOVER_C**(500,0, 250,250, 0)
相关指令	MOVER_L，MOVER_C3

MOVER_C3/MOVER_C3ABS -- 关节轴空间圆弧

<table>
<tr><td>类型</td><td>运动指令</td></tr>
<tr><td>描述</td><td>关节轴直接走空间圆弧插补运动。
此指令正解模式下使用。
BASE 虚拟的 XYZ 轴，否则无法确定 XYZ，此时的参数也是虚拟轴的距离。</td></tr>
<tr><td>语法</td><td>MOVER_C3 (endx, endy, endz, midx, midy, midz, mode[, dis1][,dis2][,dis3])
end1：第 1 个轴运动距离参数 1
end2：第 2 个轴运动距离参数 1
end3：第 3 个轴运动距离参数 1
centre1：第 1 个轴运动距离参数 2
centre2：第 2 个轴运动距离参数 2
centre3：第 3 个轴运动距离参数 2
mode：模式
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>当前点，圆心，终点定圆弧。
走最短的圆弧。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>当前点，圆心，终点定圆。
先走最短的圆弧，再继续走完整圆。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table>
dis1- disn：螺旋轴的距离</td></tr>
<tr><td>适用控制器</td><td>4 系列 20170511 以上固件支持</td></tr>
<tr><td>例程</td><td>L1 = 500
L2 = 500
TABLE(0,L1,L2, 100*360, 100*360, 360)　　'参数存储在 TABLE0 开始的
'位置，电机一圈 360 个脉冲
BASE(6,7)
CONNREFRAME(1,0,0,1)　'第 6/7 轴作为虚拟的 XY 轴，启动连接
WAIT LOADED　　'等待运动加载

BASE(6,7,8)　　'REFRAME 直接 MOVER 运动虚拟轴，会自动转换到关节
'轴上
MOVER_LABS(400)
MOVER_C3ABS(200,0,0,600,400,0, 0)</td></tr>
<tr><td>相关指令</td><td>MOVER_L，MOVER_C</td></tr>
</table>

FRAME_CAL -- 参数矫正

类型	机械手计算指令
描述	**根据机械手关节示教的坐标和特点自动校正当前的机械手参数。** Tablein 中存储的关节轴坐标获取，用当前的零点位置与机械手参数建立机械手链接前提下，控制机械手末端点运动到校正点，取校正点的关节轴坐标。 校正当前零点位置与理论零点的偏差，计算出在理论零点的关节轴坐标。 校正机械手参数的值（校正部分参数），计算出理论的机械参数的值。 FRAME_CAL 只做计算，指令返回值为-1 计算成功，返回 0 计算失败。 FRAME_CAL 的 BASE 轴必须是在 FRAME 状态的轴。
语法	FRAME_CAL(tablein,space,groups,tableaux, zeroout, [tableout2]) tablein：关节坐标存储的 table 起始编号，每个点的关节坐标按顺序存储，多个点之间间隔 space space：每两个点之间间隔的 table 元素 groups：点数 tableaux：辅助参数的 table 编号，部分 frame 需要 zeroout：计算出来理论零点时关节轴的绝对坐标的 table 编号 tableout2：计算出来的机械手参数存储的位置，不填时直接存储原参数的位置
适用控制器	通用
例程	查看机械手指令说明手册章节

第七章 程序结构与流程指令

7.1 程序符号

' -- 注释

类型	特殊字符
描述	后面全部是注释，直到下一行开始。
适用控制器	通用

_ -- 换行

类型	特殊字符
描述	后面换行继续。 在条件判断语句、内存存储、打印输出时尽量不要使用。
适用控制器	通用

: -- 标号

类型	语法结构
描述	用户过程标号，标号可以作为无参数的 SUB 过程来使用。
语法	Label 标号名称，不能与现有的关键词冲突。
适用控制器	通用
例子	GOTO label1 END '主程序结束 label1: '加：定义标号 END

7.2 数据定义指令

CONST -- 常量定义

类型	语法指令
描述	定义符号表示的常数，从而避免直接用数值表示。

语法	CONST　CVARNAME = value CVARNAME：常量名 value：常量值
适用控制器	通用
例子	例一 **CONST** MAX_VALUE = 100000　　'定义常量 TABLE(0)=MAX_VALUE　　'table(0)赋值 10000 例二 GLOBAL **CONST** MAX_AXIS=6　　'定义总轴数
相关指令	DIM

DIM -- 变量定义

类型	语法指令
描述	**定义文件模块变量，数组。** 当变量不定义就赋值时，会自动定义为文件模块变量。 文件模块变量只能在本程序文件内部使用。 数组可以作为字符串使用，一个元素表示一个字节。
语法	DIM　varname, arrayname(space) varname：变量名称 arrayname：数组名称 space：数组长度
适用控制器	通用
例子	**DIM** ARRAY1(100)　　'定义数组 ARRAY1 **DIM** VAR1　　'定义变量 VAR1 VAR2 = 100　　'赋值语句会自动定义文件模块变量 ARRAY1 = "asdf" ARRAY1(0,100,200,300)　'对数组进行连续赋值 'ARRAY1(0)=100，ARRAY1(1)=200，ARRAY1(2)=300
相关指令	CONST，LOCAL，GLOBAL

LOCAL -- 局部定义

类型	语法指令
描述	**定义局部变量，局部变量。** 局部变量主要用于 SUB 中。 同一个 SUB 的局部变量有个数限制，SUB 过程的输入参数自动转化为局部变量。 不同任务的同一个 SUB 过程调用生成不同的局部变量，同一个任务的 SUB 过程递归调

	用也生成不同的局部变量。
语法	SUB subA() LOCAL localname 'localname 局部变量名 …… ENDSUB
适用控制器	通用
例子	SUB aaa() **LOCAL** v1 '定义局部变量 V1 v1=100 END SUB
相关指令	DIM，GLOBAL

GLOBAL -- 全局定义

类型	语法指令
描述	**定义全局变量，数组；定义全局 SUB 过程。** 全局定义的量可以在整个项目的任意程序文件中使用。
语法	语法 1：GLOBAL VAR1 语法 2：GLOBAL SUB SUB1() 语法 3：GLOBAL CONST CVARNAME = value 参数： VAR1：变量名称 SUB1：过程名称 CVARNAME：常量名称 value：常量值
适用控制器	通用
例子	**GLOBAL** SUB g_sub2() '定义全局过程 g_sub2，可以在任意文件中使用 **GLOBAL** CONST g_convar = 100 '定义全局使用的常量 **GLOBAL** g_var2 '定义全局变量 g_var2
相关指令	DIM，LOCAL

7.3 数组操作指令

DMINS -- 数组链表插入

类型	语法指令
描述	**数组的链表操作，插入后当前元素以及后面的所有元素往后移动一个位置。** 谨慎对超长数组进行操作，特别是数组 TABLE。
语法	DMINS arrayname(pos) arrayname：数组名称

	pos：数组索引
适用控制器	通用
例子	DIM aa(6)　　'定义数组 aa FOR i=0 TO 4　　'赋值 0,1,2,3,4 aa(i)=i NEXT ?*aa　　'打印数组所有元素 **DMINS** aa(0)　　'插入元素 0，原有的所有元素都想后移动一个位置 aa(0) = 10　　'为插入元素赋值 ?*aa　　'打印插入后的数组所有元素
相关指令	DMDEL，DMCPY

DMADD -- 数组批量增加

类型	语法指令
描述	**对数组元素值进行批量增加。** 不要一次修改超过 500 元素。
语法	DMADD　arrayname (pos, size , data) arrayname：数组名 pos：起始的索引 size：要修改的个数，注意加上 pos 不要超过数组大小 data：要增加的值
适用控制器	通用
例子	DIM aaa(20)　　'定义一个空间为 20 的数组 ?*aaa　　'打印，全为 0 **DMADD** aaa(10,5,2)　'从元素 10 开始，修改 5 个元素值+2 ?*aaa　　'打印，其中 10、11、12、13、14 为 2，其余为 0 **DMADD** aaa(10,5,2)　'从元素 10 开始，修改 5 个元素值+2 ?*aaa　　'打印，其中 10、11、12、13、14 为 4，其余为 0
相关指令	DMINS，DMCPY

DMDEL -- 数组链表删除

类型	语法指令
描述	**数组的链表操作；删除数组元素，删除后当前元素后面的所有元素向前移动。** 谨慎对超长数组进行操作，特别是数组 TABLE。
语法	DMDEL　arrayname(pos)

	arrayname：数组名称 pos：数组索引
适用控制器	通用
例子	DIM aa(6) '定义数组 aa FOR i=0 TO 4 '赋值 0,1,2,3,4 aa(i)=i NEXT ?*aa '打印数组所有元素 **DMDEL** aa(0) '删除数组 a 的第 1 个元素 ?*aa '打印删除后的数组所有元素
相关指令	DMINS，DMCPY

DMCPY -- 数组拷贝

类型	语法指令
描述	**数组拷贝，从数组 Src 拷贝到数组 Des。** 谨慎对超长数组进行操作，特别是数组 TABLE。
语法	DMCPY arraydes(startpos),arraysrc(startpos)[,size] arrayname：数组名称 startpos：数组起始索引 size：拷贝的个数，超过最大值会自动缩减
适用控制器	通用
例子	GLOBAL aa(6),bb(6) '定义数组 aa，bb FOR i=0 TO 4 '赋值 aa 0,1,2,3,4 aa(i)=i NEXT ?*aa '打印数组所有元素 ?*bb **DMCPY** aa(0), bb(0),6 '把数组 bb 的值赋值给数组 aa ?*aa '打印数组复制后所有元素 ?*bb
相关指令	DMINS，DMDEL

7.4 自定义子函数指令

SUB -- 自定义子函数 SUB

类型	语法指令
描述	**用户自定义 SUB 过程，可以前面增加 GLOBAL 描述以定义全局使用的 SUB 过程。**
语法	SUB label([para1] [,para2]…)

	… END SUB 参数： label：过程名称，不能与现有的关键词冲突 para1：过程调用时传入的参数，自动作为 LOCAL 局部变量 para2：过程调用时传入的参数，自动作为 LOCAL 局部变量
适用控制器	通用
例子	SUB sub1()　'定义过程 SUB1，只能在当前文件中使用 ?1 ... END SUB GLOBAL SUB g_sub2()　'定义全局过程 g_sub2，可以在任意文件中使用 ?2 ... END SUB GLOBAL SUB g_sub3(para1,para2)　'定义全局过程 g_sub3，传递两个参数 ?Para1,para2 ... RETURN para1+para2　'函数返回参数相加 END SUB
相关指令	SUB_PARA，SUB_IFPARA

SUB_PARA -- SUB 传递参数

类型	语法指令
描述	**选择 SUB 的传入参数。**
语法	SUB_PARA(address) address：第几个传递参数，从 0 开始
适用控制器	通用
例子	SUB AAA(NUM1,NUM2,NUM3) ?**SUB_PARA(0)**　'调用 AAA 时，打印传递的第一个 num1 值 ?**SUB_PARA(1)**　'打印传递的第二个 num2 值 ?**SUB_PARA(2)**　'打印传递的第三个 num3 值 END SUB
相关指令	SUB，SUB_IFPARA

SUB_IFPARA -- SUB 传参判断

类型	语法指令
描述	**判断 SUB 是否传入参数。**
语法	SUB_IFPARA(address) -1 传入，0 未传入 address：第几个传递参数，从 0 开始
适用控制器	通用

例子	AAA(0,100) '传入 num1,num2
	AAA(,100) '只传入 num2
	END
	SUB AAA(NUM1,NUM2)
	IF **SUB_IFPARA(0)** THEN '调用 AAA 时，判断 num1 是否传入
	?1 '传入打印 1
	ELSE
	?0
	ENDIF
	END SUB

相关指令	SUB，SUB_PARA

GOSUB/CALL -- SUB 调用

类型	程序结构
描述	**SUB 过程调用，只能调用本文件的 SUB 过程或全局 SUB 过程。** 直接调用 SUB 过程时，可以省掉 GOSUB 语句。 SUB 过程没有参数传递时，可以省掉括号()。 GOSUB 后当前的内容会压栈，不能在调用的 SUB 程序中访问当前的局部变量，RETURN 返回时出栈。
语法	GOSUB/CALL label label：SUB 过程名
适用控制器	通用
例子	'主程序 main: **GOSUB** sub1() sub2(1,2) '传入 1 给 para1，2 给 para2 CALL sub3 END '定义的 SUB SUB sub1()

	a=100 PRINT "sub1" RETURN SUB sub2(para1,para2) a=200 PRINT "sub2",para1,para2 RETURN GLOBAL SUB sub3() '可以在另外一个程序文件中 a=300 PRINT "sub3" RETURN

GSUB -- 自定义子函数-G 代码格式

类型	语法指令
描述	**用户自定义 GSUB 过程。** 可以前面增加 GLOBAL 描述以定义全局使用的 GSUB 过程，GSUB 过程调用的时候采用 G 代码语法，不要加括号。
语法	GSUB label([char1] [,char2]…) … END SUB 参数： Label：过程名称，不能与现有的关键词冲突 char1：过程调用时传入的字母参数，自动作为 LOCAL 局部变量 char2：过程调用时传入的字母参数，自动作为 LOCAL 局部变量 字母参数只能为单字符
适用控制器	通用
例子	G01 X100 Y100 Z100 U100 '调用 G01 END '主程序结束 GLOBAL GSUB G01(X, Y, Z, U) '定义 GSUB 过程 G01 … END SUB
相关指令	GSUB_PARA，GSUB_IFPARA

GSUB_PARA -- GSUB 传递参数

类型	语法指令
描述	**选择 GSUB 的传入参数。**
语法	GSUB_PARA(char) char：GSUB 定义时传入的字母参数
适用控制器	通用
例子	GSUB AAA(X,Y,Z) ?**GSUB_PARA(X)** '调用 AAA 时，打印传递的 X 值 ?**GSUB_PARA(Y)** '打印传递的 Y 值 ?**GSUB_PARA(Z)** '打印传递的 Z 值 END SUB
相关指令	GSUB，GSUB_IFPARA

GSUB_IFPARA -- 判断 GSUB 是否传入参数

类型	语法指令
描述	**判断 SUB 是否传入参数。**
语法	GSUB_IFPARA(char) char：GSUB 定义时传入的字母参数 值：-1-传入，0-未传入
适用控制器	通用

例子	AAA X0 Y100 '传入 X,Y
	AAA X0 '只传入 X
	END
	
	GSUB AAA(X,Y)
	IF **GSUB_IFPARA(Y)** THEN '调用 AAA 时，判断 Y 是否传入
	?1 '传入打印 1
	ELSE
	?0
	ENDIF
	END SUB

相关指令	，GSUB_PARA

END SUB -- 自定义函数结束

类型	程序结构
描述	**用户自定义 SUB 过程结束，参见 SUB。**

适用控制器	通用

RETURN -- 函数返回值

类型	程序结构
描述	**用户 SUB 过程返回或返回值。** 返回值缺省 0，在外面可以通过 RETURN 来读取上个 SUB 调用的返回值。 不同任务的返回值不同。
语法	RETURN
适用控制器	通用
例子	CALL sub1 ?**RETURN** '结果为 111 END '主程序结束 SUB sub1() **RETURN** 111 '返回 111 END SUB

7.5 跳转指令

GOTO -- 强制跳转

类型	程序结构
描述	**强制跳转，与 GOSUB 的区别是 GOTO 不会压栈。**
语法	GOTO label
适用控制器	通用
例子	a=100 **GOTO** label1 '强制跳转至 label1 a=1000 END '主程序结束 label1: PRINT a '结果是 a=100 END 'label1 结束

ON GOSUB -- 条件跳转

类型	程序结构

描述	**当 expression 条件为真时，调用过程 label。**
语法	ON expression GOSUB label expression：判断条件 label：跳转 sub、label 名称
适用控制器	通用
例子	a=100 **ON** a>10 **GOSUB** label1 'a>10 时调用 label 过程 a=1000 PRINT a END '主程序结束 label1: PRINT a RETURN 'SUB 过程要返回

ON GOTO -- 条件跳转 2

类型	程序结构
描述	**条件跳转，当 expression 条件为真时跳转，不会压栈。**
语法	ON expression GOTO label expression：判断条件 label：跳转 sub、label 名称
适用控制器	通用
例子	a=100 **on** a>10 **goto** label1 a=1000 END '主程序结束 label1: PRINT a END 'goto 跳转无法 return 返回

7.6 条件判断指令

IF -- 条件判断结构

类型	程序结构
描述	**条件判断，同标准 BASIC 结构。**
语法	IF <condition1> THEN commands

	ELSEIF <condition2> THEN
	commands
	ELSE
	commands
	ENDIF
	参数：
	condition1：条件
	condition2：条件

适用控制器	通用
例子	例一 DIM a '定义变量 a=12 '赋值 **IF** a>11 THEN '条件判断 TRACE "the val a is bigger then 11" ELSEIF a<11 THEN TRACE "the val a is less then 11" ENDIF 例二 IF IN(0) THEN OUT(0,ON) '当单行时可以不用 endif
相关指令	THEN，ENDIF

THEN -- 条件判断结构

类型	程序结构
描述	**参见：IF**
适用控制器	通用

ENDIF -- 条件判断结构

类型	程序结构
描述	**参见：IF**
适用控制器	通用

ELSEIF -- 条件判断结构

类型	程序结构
描述	**参见：IF**

适用控制器	通用

7.7 循环指令

FOR -- for 循环结构

类型	程序结构
描述	**循环语句，采用标准 BASIC 语法。**
语法	FOR variable=start TO end [STEP increment] commands NEXT variable 参数： variable：变量名称 start：起始循环值 end：结束循环值 increment：循环步进增量，可选 多任务时，请不要使用（非 local 时）相同的 variable 循环变量，否则会相互干扰。
适用控制器	通用
例子	LOCAL a **FOR** a=1 TO 100 '1 循环至 100 PRINT a '打印 a NEXT
相关指令	TO，STEP，NEXT

TO -- for 循环结构

类型	程序结构
描述	**参见：FOR**
适用控制器	通用

STEP -- for 循环结构

类型	程序结构
描述	**参见：FOR**
适用控制器	通用

NEXT -- for 循环结构

类型	程序结构
描述	**参见：FOR**
适用控制器	通用

WHILE -- while 循环结构

类型	程序结构
描述	**条件满足时执行循环。**
语法	WHILE condition … WEND
适用控制器	通用
例子	a=0 **WHILE** IN(4)=OFF '直到输入 4 有效，退出循环 a=a+1 PRINT a DELAY(1000) WEND

WEND -- while 循环结构

类型	程序结构
描述	**参见：WHILE**
适用控制器	通用

EXIT -- 退出循环

类型	程序结构
描述	**循环退出语句。**
语法	EXIT FOR , EXIT WHILE
适用控制器	通用
例子	LOCAL a FOR a=1 TO 100 '1 循环至 100 PRINT a '打印 a IF a> 20 THEN **EXIT** FOR '必须采用这种方式，否则 IF 和 ENDIF 不匹配

	NEXT

REPEAT -- 条件循环

类型	程序结构
描述	**循环语句。** 循环执行 commands，condition 为真时退出循环。
语法	REPEAT commands UNTIL condition
适用控制器	通用
例子	a=0 **REPEAT** '循环执行下面语句 PRINT a a=a+1 DELAY(1000) UNTIL IN(4)=ON '直到输入 4 有效

UNTIL -- 条件结构

类型	程序结构
描述	**参见：REPEAT 、WAIT**
适用控制器	通用

7.8 等待执行指令

DELAY -- 延时

类型	语法指令
描述	**延时 delay time，单位毫秒。** 别名：wa
语法	DELAY(delay time) delay time：毫秒数
适用控制器	通用
例子	**DELAY**(100) '延时 100ms

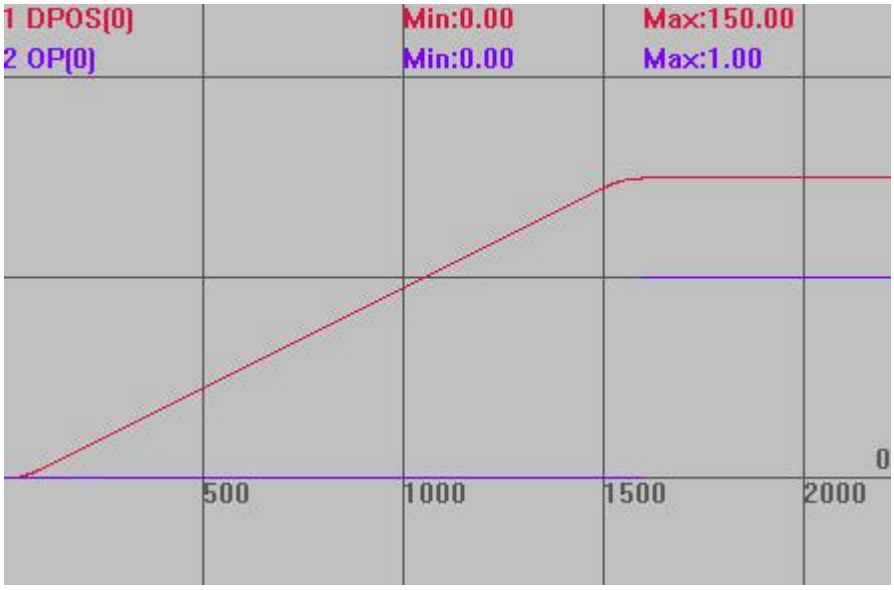
WAIT UNTIL -- 等待条件满足

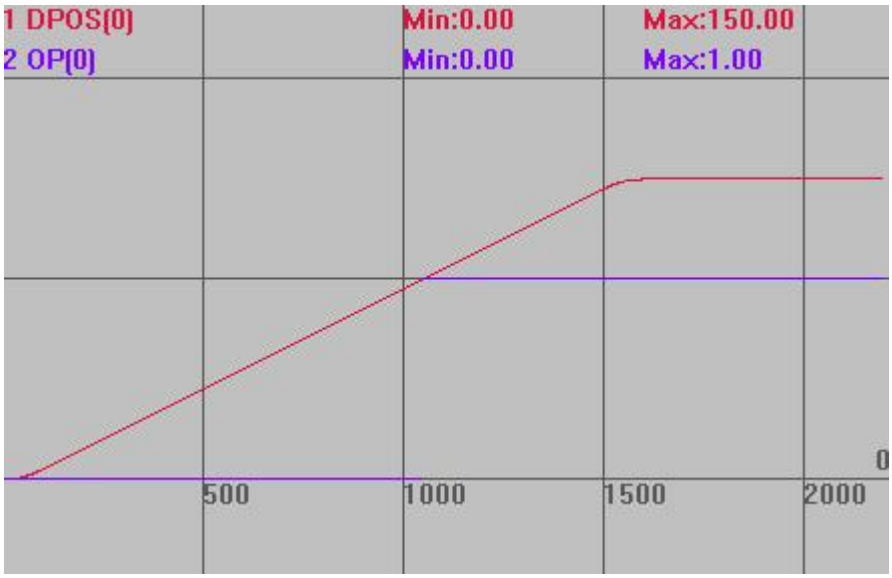
类型	程序结构
描述	**等待直到条件满足。**
语法	WAIT UNTIL condition1 [and codition2 or codition3 ...] 可以通过逻辑运算操作多个条件。
适用控制器	通用
例子	例一 **WAIT UNTIL** DPOS(0) > 0　'等待轴 0 位置超过 0 例二 与 TICKS 一起使用 TICKS=2000　　'ticks 设为 2000 **WAIT UNTIL** TICKS<0　'等待 2s ?"执行下一步" 例三 与逻辑条件一起使用 **WAIT UNTIL** IDLE(0)=-1 AND IDLE(1)=-1 AND IDLE(2)=-1 '等待轴 0、1、2 都停止

WAIT IDLE -- 等待轴停止

类型	语法指令
描述	**等待 BASE 轴运动完成，BASE 轴运动未完成时，不执行后面的程序。** 等效于 WAIT UNTIL IDLE。IDLE 作为轴参数，支持轴参数的语法。
语法	WAIT IDLE
适用控制器	通用
例子	例一 BASE(0,1) MOVE(100,100) **WAIT IDLE**　　'等待当前插补运动结束 例二 BASE(0,1) MOVE(100,100) BASE(2,3) MOVE(200,200) **WAIT UNTIL　IDLE**(0) AND **IDLE**(1) AND **IDLE** (2) AND **IDLE**(3) '等待轴 0，1，2，3 停止 ?"运动完成"

WAIT LOADED -- 等待轴缓冲空

类型	语法指令
描述	**等待 BASE 轴运动缓冲空，此命令会阻塞不执行后面的程序。** 缓冲区最后一个运动可以正确执行，同时程序向下扫描。 等效于 WAIT UNTIL LOADED，LOADED 作为轴参数，支持轴参数的语法。
语法	WAIT LOADED
适用控制器	通用
例子	与 WAIT IDLE 的区别 BASE(0) ATYPE=1 UNITS=100 DPOS=0 SPEED=100 ACCEL=1000 MERGE=1 TRIGGER MOVE(100)　'当前运动 MOVE(50)　'缓冲运动，此时缓冲区只有这一条运动 '当本条运动执行时，缓冲区就已经清空 WAIT IDLE　'使用 wait idle 时，要等到所有运动完成才能往下执行 OP(0,ON)　'打开 op0 运动轨迹 DPOS(0)垂直刻度 100 OP(0)垂直刻度 1  **WAIT LOADED**　'使用 wait loaded 时，运动缓冲空即可往下执行 OP(0,ON)


1 DPOS(0)
2 OP(0)
Min:0.00
Min:0.00
Max:150.00
Max:1.00
0
500
1000
1500
2000

第八章 任务相关指令

ZBASIC 支持实时多任务运行，一个文件上也可以同时运行多个任务。通过 RUN 可以从文件第一行开始运行任务，通过 RUNTASK 可以随意指定 SUB 过程开始运行。

8.1 任务启停指令

RUN -- 启动文件任务

类型	任务指令
描述	**新建一个任务来执行控制器上的一个文件。** 重复启动同一任务会报错。 多次使用 RUN 指令不带任务号参数时，会让同一个文件对应多个任务，建议使用 RUNTASK 指令开启任务。 多任务操作指令有： END：当前任务正常结束 STOP：停止指定文件 STOPTASK：停止指定任务 HALT：停止所有任务 RUN：启动文件执行 RUNTASK：启动任务在一个 SUB 上执行
语法	RUN "filename"[, tasknum] filename：程序文件名，BAS 文件可不加扩展名 tasknum：任务号，缺省查找第一个有效的
适用控制器	通用
例子	**RUN** "aaa", 1 '启动任务 1 运行 aaa.bas 文件
相关指令	RUNTASK

RUNTASK -- 启动 SUB 任务

类型	任务指令
描述	**把一个 SUB 过程或是标号作为一个新的任务执行。** 重复启动同一任务会报错。
语法	RUNTASK tasknum, label tasknum：任务号 label：自定义 SUB 过程（不能带参数）或标号
适用控制器	通用
例子	**RUNTASK** 1, taska '启动任务 1 来跟踪打印位置 MOVE(1000,100)

	MOVE(1000,100) END taska: '循环打印位置 WHILE 1 PRINT *mpos DELAY(1000) WEND END
相关指令	RUN

END -- 结束

类型	任务指令
描述	**当前任务结束。** 当一个文件中有主程序和 SUB 过程时，一定要在主程序结束后写 END，如果不写 END，程序执行完主程序后，就会依次再执行 SUB 子程序，直到子程序的 END SUB 才停止。
适用控制器	通用
相关指令	RUN，RUNTASK

STOP -- 停止文件任务

类型	任务指令
描述	**程序强制停止，操作文件。** 再启动任务前都要停止任务。 使用 STOP 指令不带任务号时，一次只会停掉一个任务，而不是此文件对应的所有任务，当一个文件内有多个任务时，建议用 STOPTASK 指令。
语法	STOP program name, [tasknum] program name：程序文件名，bas 文件可不用带扩展名 tasknum：任务号，当程序文件有启动多个任务时，缺省任务号最小的任务
适用控制器	通用
例子	RUN aaa, 1 '执行 aaa.bas **STOP** aaa, 1 '停止任务 1
相关指令	STOPTASK，HALT

STOPTASK -- 停止 SUB 任务

类型	任务指令

描述	**任务强制停止，操作 SUB 和标记。** 再启动任务前都要停止任务。
语法	STOPTASK [tasknum] tasknum：任务号，缺省当前任务
适用控制器	通用
例子	**STOPTASK** 2 '停止任务 2
相关指令	STOP，HALT

HALT -- 停止全部任务

类型	任务指令
描述	**停止全部任务。** 此指令仅限 PC 软件调用，BASIC 程序中若使用此指令，会导致整个程序停止，控制器无法运行。
语法	HALT
适用控制器	通用
例子	**HALT** '停止所有任务
相关指令	STOP，STOPTASK

PAUSE -- 暂停全部任务

类型	任务指令
描述	**暂停全部任务。** 使用断点生效后也会进入暂停状态。 此指令仅限 PC 软件调用，BASIC 程序中若使用此指令，会导致整个程序停止，控制器无法运行。 暂停任务再恢复后，任务继续往下执行。
语法	PAUSE
适用控制器	通用
例子	**PAUSE** '暂停所有任务
相关指令	PAUSETASK

PAUSETASK -- 暂停指定任务

类型	任务指令
描述	**暂停某一个任务。** 暂停任务再恢复后，任务继续往下执行。

语法	PAUSETASK tasknum tasknum：任务号，缺省当前任务
适用控制器	通用
例子	**PAUSETASK** 1 '暂停任务 1
相关指令	RESUMETASK

RESUMETASK -- 恢复指定任务

类型	任务指令
描述	**恢复某一个任务。** 暂停任务再恢复后，任务继续往下执行。
语法	RESUMETASK tasknum tasknum：任务号，缺省当前任务
适用控制器	通用
例子	PAUSETASK 1 '暂停任务 1 **RESUMETASK** 1 '继续运行任务 1
相关指令	PAUSETASK

8.2 三次文件任务指令

FILE3_RUN -- 执行 FILE3 任务

类型	任务指令
描述	**3 次程序文件的启动命令。** 3 次程序文件是一种超大文件，可以动态加载，使用 basic 语法。不支持条件判断、跳转等操作。可以通过指令下载，也可以通过工具软件 zfile3view 进行浏览、上传和下载操作。
语法	FILE3_RUN "filename", tasknum filename：3 次程序的文件名，必须事先下载到控制器里面 Tasknum：任务号，缺省查找第一个有效的
适用控制器	必须带大容量存储的控制器和 2015 以上的固件版本支持。
例子	**FILE3_RUN** "aaa.z3p", 1 '在任务 1 运行 3 次文件 aaa.z3p
相关指令	FILE3_ONRUN

FILE3_ONRUN --FILE3 回调函数

类型	回调函数
描述	**3 次文件启动时会自动调用。**

语法	GLOBAL　FILE3_ONRUN: 标号 GLOBAL SUB　FILE3_ONRUN()　自定义 SUB 过程（不能带参数） 回调函数属于 3 次文件任务。
适用控制器	通用

例子	FILE3_RUN "aaa.z3p",　1　　'在任务 1 运行 3 次文件 aaa.z3p
	END
	
	GLOBAL SUB　**FILE3_ONRUN**()　'3 次任务启动时自动调用
	　IF　1= PROCNUMBER　THEN
	　　BASE(0,1,2)　　'选择 3 次文件的运行轴列表
	　　SPEED=1000
	　　ACCEL=10000
	　ELSE
	　　BASE(4,5,6)
	　ENDIF
	END SUB

相关指令	FILE3_RUN

FILE3_GOTO -- FILE3 强制跳转

类型	任务函数
描述	**对三次任务有效，强制跳转到指定的行号开始运行。**
语法	FILE3_GOTO(linenum) 　　linenum：要跳转的行号，从 1 开始编号
适用控制器	通用
相关指令	FILE3_LINE，FILE3_RUN

FILE3_LINE -- FILE3 行号

类型	任务函数
描述	**返回当前 3 次文件运行的行号，无论是否三次文件因为 SUB 调用进入 BASIC 文件，总是返回 3 次文件的行号。**
语法	VALUE=FILE3_LINE([taskid]) 　　taskid：三次文件的任务号，不填返回函数调用当前任务的
适用控制器	通用
相关指令	FILE3_RUN，FILE3_GOTO

8.3 任务参数指令

BASE_MOVE -- 指定主轴

类型	任务参数
描述	**用于强制指定插补运动函数的 BASE 主轴，此参数不修改实际的运动。** 缺省值为-1，此时不生效。 20160326 以上固件版本支持。 每个任务都有独立的 BASE_MOVE 参数。 MOVE、MOVEABS、MOVECIRC、MOVE_OP、MOVE_TASK 等插补类型函数有效，凸轮点位等单轴函数无效。
语法	VAR1 = BASE_MOVE，BASE_MOVE = value
适用控制器	特殊固件支持
例子	**BASE_MOVE**=2　　'强制本任务的插补运动使用轴 2 为主轴，速度参数也使用轴 2 的 MERGE(2)=ON　　'轴 2 启动连续 SPEED(2)=100 ACCEL(2)=1000 BASE(0,1) MOVE(100,100)　'轴 0/1 插补，轴 2 也强制作为主轴参与，但运动距离为 0 MOVE_OP(1,1) BASE(1) MOVE(100)　　'1 轴运动 100，也使用轴 2 为主轴 **BASE_MOVE**=-1　　'取消本任务的强制主轴

PROC_STATUS -- 任务状态

类型	任务状态
描述	**当前任务的状态。** 0　任务停止 1　任务正在运行 3　任务暂停中
语法	VAR1 = PROC_STATUS(tasknum) tasknum：任务号
适用控制器	通用
例子	PRINT **PROC_STATUS**(0)　　'打印任务 0 状态 在线命令输入 >>PRINT **PROC_STATUS**(0) 输出：1
相关指令	PROC

PROC -- 任务编号

类型	任务修正辅助指令
描述	**当访问任务参数，任务状态时可以指定其他的任务。**
语法	PROC(tasknum) tasknum：任务号 可以省略，参考 AXIS
适用控制器	通用
例子	例一 完整写法 PRINT **PROC**_STATUS **PROC**(1) '打印任务 1 运行状态 例二 简写 PRINT **PROC**_STATUS(1) '打印任务 1 运行状态
相关指令	PROCNUMBER

PROCNUMBER -- 当前任务编号

类型	任务特殊状态，系统状态
描述	**当前任务的任务号。** 当程序不知道当前任务号的时候，通过此指令来访问。 这个状态不能通过 PROC 来修正。
语法	VAR1 = PROCNUMBER
适用控制器	通用
例子	PRINT **PROCNUMBER** '打印当前任务号
相关指令	PROC

PROC_LINE -- 任务行号

类型	任务状态
描述	**任务的当前行号，只能获取其他任务的。**
语法	VAR1 = PROC_LINE(tasknum) tasknum：任务号
适用控制器	通用
例子	PRINT **PROC_LINE** (0) '打印任务 0 运行到那一行 在线命令输入 >>PRINT **PROC_LINE** 输出：100

相关指令	PROC

ERROR_LINE -- 任务错误行号

类型	任务状态
描述	**当前任务的错误行号。** 一般只用在出错后，在线命令查看。
语法	VAR1 = ERROR_LINE(tasknum) tasknum：任务号
适用控制器	通用
例子	在线命令输入 >>?**ERROR_LINE**(1) '打印任务 1 错误行
相关指令	PROC，ERROR_SET

RUN_ERROR -- 任务错误码

类型	任务状态
描述	**任务中最先出现的错误编号。**
语法	VAR1 = RUN_ERROR(tasknum) tasknum：任务号
适用控制器	通用
例子	?* **RUN_ERROR**(0)　　'打印任务 0 最先出现的错误号 2043
相关指令	ERROR_LINE

TICKS -- 任务计数周期

类型	任务参数
描述	**当前任务的计数周期数，每个周期减一，单位是毫秒。** 每个任务都有独立的 TICKS 参数，ZMC00x 系列，ZMC1xx 系列的周期为 1 毫秒。 系统刷新周期修改不影响本 TICKS 的计时。
语法	VAR1 = TICKS，TICKS = value
适用控制器	通用
例子	**TICKS** = 1000 WAIT UNTIL　**TICKS** < 0　　'等待 TICKS< 0 后程序往下执行 MOVE(100)

相关指令	TIME_TICKUS

TIME_TICKUS -- 任务计数周期

类型	任务参数
描述	**当前任务的计数周期数，每个周期加一，单位是微秒。** 每个任务都有独立的 TIME_TICKUS 参数，32 位整数。 系统刷新周期修改不影响本 TIME_TICKUS 的计时。
语法	VAR1 = TIME_TICKUS，TIME_TICKUS = value
适用控制器	通用
例子	**TIME_TICKUS**=0 DELAY(1)　　'延时 1 毫秒 ?TIME_TICKUS　　'打印结果：1000，单位微秒
相关指令	TICKS

第九章 运算符及数学函数指令

ZBASIC 支持标准 BASIC 的所有运算符，也采用标准 BASIC 的优先级。

优先级顺序：算术运算符 > 比较运算符 > 计算逻辑运算符。相同优先级的运算符采用从左到右的顺序计算。

算术运算符		比较运算符		逻辑运算符	
描述	符号	描述	符号	描述	符号
求幂	^	等于	=	逻辑非	Not
负号	-	不等于	<>	逻辑与	And
乘	*	小于	<	逻辑或	Or 或 |
除	/	大于	>	逻辑异或	Xor
整除	\	小于等于	<=	逻辑等价	Eqv
求余	Mod 或 %	大于等于	>=		
加	+				
减	-				
左移位	<<				
右移位	>>				

9.1 算术运算指令

+ -- 加法运算

类型	运算符
描述	**将两个表达式相加。**
语法	expression1 + expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 1+2 输出：3

- -- 减法运算

类型	运算符
描述	**将表达式 1 减去表达式 2。**
语法	expression1 - expression2 expression1：任意有效的表达式 expression2：任意有效的表达式

适用控制器	通用
例子	在线命令输入 >>PRINT 2-(2-1) 输出：1

* -- 乘法运算

类型	运算符
描述	**将表达式 1 与表达式 2 相乘。**
语法	expression1 * expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 10*(1+2) 输出：30

/ -- 除法运算

类型	运算符
描述	**将表达式 1 除以表达式 2。**
语法	expression1 / expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 10/3 输出：3.3333

\ -- 整除运算

类型	运算符
描述	**整数除法。**
语法	expression1 \ expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入

	>>PRINT 10 \ (1+2) 输出：3

<< -- 左移位

类型	运算符
描述	**左移位。**
语法	expression1 << expression2 expression1：任意有效的表达式 expression2：任意有效的表达式 运算优先级低于其他四则运算符，共同使用时需要加括号()，见例三。
适用控制器	通用
例子	例一 直接操作数值 在线命令输入 >>PRINT 8<<1 '二进制左移一位 输出：16 在线命令输入 >>PRINT 8<<2 '二进制左移两位 输出：32 例二 操作变量、寄存器 DIM bb bb=8 MODBUS_REG(0)=8 PRINT bb<<1,bb<<2 PRINT MODBUS_REG(0)<<1,MODBUS_REG(0)<<2 例三 优先级比较 >>PRINT 8<<1+1 '此时二进制左移 2 位 输出：32 >>PRINT (8<<1)+1 '此时二进制左移 1 位 输出：17

>> -- 右移位

类型	运算符
描述	**右移位。**

语法	expression1 >> expression2 expression1：任意有效的表达式 expression2：任意有效的表达式 运算优先级低于其他四则运算符，共同使用时需要加括号()，见例三。
适用控制器	通用
例子	例一 直接操作数值 在线命令输入 >>PRINT 8>>1 '二进制右移一位 输出：4 在线命令输入 >>PRINT 8>>2 '二进制右移两位 输出：2 例二 操作变量、寄存器 DIM bb bb=8 MODBUS_REG(0)=8 PRINT bb>>1,bb>>2 PRINT MODBUS_REG(0)>>1,MODBUS_REG(0)>>2 例三 优先级比较 >>PRINT 8>>1+1 '此时二进制右移 2 位 输出：2 >>PRINT (8>>1)+1 '此时二进制右移 1 位 输出：5

MOD -- 求余数

类型	运算符
描述	**求余数。**
语法	expression1 MOD expression2 expression1：任意有效的表达式，取整数部分。 expression2：任意有效的表达式，取整数部分。
适用控制器	通用
例子	在线命令输入 >>PRINT 10 **MOD** (1+2) 输出：1

ABS -- 绝对值

类型	数学函数
描述	**求绝对值。**
语法	ABS(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT **ABS**(-11) '结果是 11

9.2 比较运算指令

= -- 比较/赋值运算

类型	运算符
描述	**比较运算符**：如果表达式 1 等于表达式 2，返回 TRUE，否则返回 FALSE **赋值运算符**：将表达式 2 赋值给前面的变量或参数等。
语法	expression1 = expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	例一 ON IN(0)=ON GOTO label1 '如果输入通道 0 为 ON，程序跳转到标识“label1:”首行开始执行 label1: PRINT 12 '打印 12 例二 DIM aaa aaa = 100 'VAR1 赋值为 100 PRINT aaa

<> -- 不等于

类型	运算符
描述	**如果表达式 1 不等于表达式 2，返回 TRUE，否则返回 FALSE。**
语法	expression1 <> expression2 expression1：任意有效的表达式 expression2：任意有效的表达式

适用控制器	通用
例子	ON MODBUS_BIT(0)<>0 GOTO label1 '如果 MODBUS 位寄存器 0 非零值，程序跳转到标识"label1:"开始执行 label1: PRINT 11 '打印 11

> -- 大于

类型	运算符
描述	**如果表达式 1 大于表达式 2，返回 TRUE，否则返回 FALSE。**
语法	expression1 > expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	WAIT UNTIL MPOS>100 该条语句将使程序循环等待，直到测量反馈位置大于 100 为止。 例一 DIM q '定义变量 q= 2>1 '2 大于 1，所以返回 true PRINT q '打印返回值 例二 DIM a '定义变量 a=0 '变量赋值 REPEAT '循环执行 a=a+1 '加一 ?a '打印 DELAY(200) '延时 UNTIL a>10 '条件判断，直到 a 里面的值大于 10 时，停止循环执行

>= -- 大于等于

类型	运算符
描述	**如果表达式 1 大于或等于表达式 2，返回 TRUE，否则返回 FALSE。**
语法	expression1 >= expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用

例子	DIM a　'定义变量 a= 1>=3　'1 小于 3，所以返回 FALSE PRINT a　'打印返回值

< -- 小于

类型	运算符
描述	**如果表达式 1 小于表达式 2，返回 TRUE，否则返回 FALSE。**
语法	expression1 < expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	VAR1=1<0 因为 1 大于 0，所以 VAR1 的值将等于 FALSE(0)。

<= -- 小于等于

类型	运算符
描述	**如果表达式 1 小于或等于表达式 2，返回 TRUE，否则返回 FALSE。**
语法	expression1 <= expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	VAR1=1<=1 VAR1 的值将等于 TRUE (-1)。

9.3 逻辑运算指令

AND -- 按位与

类型	运算符
描述	**按位与操作符，只操作整数部分。** AND　AND　结果 0　0　0 0　1　0 1　0　0 1　1　1
语法	expression1 AND expression2

	expression1：任意有效的表达式 expression2：任意有效的表达式 expression1 和 expression2 的二进制形式的每一个位上的二进制数字进行按位与（AND）运算之后的结果
适用控制器	通用
例子	在线命令输入 >>PRINT 1 **AND** 2 输出：0 具体操作过程 1 对应二进制 01 2 对应二进制 10 与计算后 对应二进制 00 输出十进制 0

OR -- 按位或

类型	运算符
描述	**按位或操作符，只操作整数部分。** OR 结果 0 0 0 0 1 1 1 0 1 1 1 1
语法	expression1 OR expression2 expression1：任意有效的表达式 expression2：任意有效的表达式 expression1 和 expression2 的二进制形式的每一个位上的二进制数字进行按位或（OR）运算之后的结果
适用控制器	通用
例子	在线命令输入 >>PRINT 1 **OR** 2 输出：3 具体操作过程 1 对应二进制 01 2 对应二进制 10 或计算后 对应二进制 11 输出十进制 3

NOT -- 按位非

类型	运算符
描述	**按位非操作符，只操作整数部分，小心对 ON 等整数 NOT。** NOT 结果 0 -1 1 -2
语法	NOT expression1 expression1：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT **NOT** 1 输出：-2 具体操作过程 1 对应二进制 ... 0000 0001 非计算后 对应二进制 ... 1111 1110 输出十进制 -2

XOR -- 按位异或

类型	运算符
描述	**逻辑异或操作符，按位异或，只操作整数部分。** XOR XOR 结果 0 0 0 0 1 1 1 0 1 1 1 0
语法	expression1 XOR expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 1 **XOR** 1 输出：0 具体操作过程 1 对应二进制 01 异或计算后 对应二进制 00 输出十进制 0

EQV -- 按位同或

类型	运算符
描述	**按位同或操作符，只操作整数部分。** EQV 0 0 → 1 EQV 0 1 → 0 EQV 1 0 → 0 EQV 1 1 → 1
语法	expression1 EQV expression2 expression1：任意有效的表达式 expression2：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 2 **EQV** 1 输出：-4 具体操作过程 2 对应二进制 ... 0000 0010 1 对应二进制 ... 0000 0001 同或计算后 对应二进制 ... 1111 1100 输出十进制 -4

9.4 三角函数指令

SIN -- 三角函数正弦

类型	数学函数
描述	**正弦三角函数，输入参数为弧度单位。**
语法	SIN(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT **SIN**(PI/6)　　'结果是 0.5000

ASIN -- 三角函数反正弦

类型	数学函数
描述	**反正弦三角函数，返回值为弧度单位。**
语法	ASIN(expression)

	expression：任意有效的表达式
适用控制器	通用
例子	PRINT **ASIN**(0.5)　　'结果是 0.52360

COS -- 三角函数余弦

类型	数学函数
描述	**余弦三角函数，输入参数为弧度单位。**
语法	COS(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT **COS**(PI/3)　　'结果是 0.5000

ACOS -- 三角函数反余弦

类型	数学函数
描述	**反余弦三角函数，返回值为弧度单位。**
语法	ACOS(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT **ACOS**(0.5)　　'结果是 1.04720=PI/3

TAN -- 三角函数正切

类型	数学函数
描述	**求正切三角函数，输入参数为弧度单位。**
语法	TAN(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT　**TAN**(PI/3)　　'结果是 1.732

ATAN -- 三角函数反正切

类型	数学函数
描述	**求反正切三角函数，返回值为弧度单位。**

语法	ATAN(expression) expression：任意有效的表达式
适用控制器	通用
例子	PRINT **ATAN**(1)　　'结果是 0.7854　= (45/180)*PI

ATAN2 -- 三角函数反正切 2

类型	数学函数
描述	**反正切三角函数，返回值为弧度单位。**
语法	ATAN2(y, x) y：y 坐标 x：x 坐标
适用控制器	通用
例子	PRINT **ATAN2**(1,0)　　'结果是 1.5708

9.5 指数运算指令

EXP -- 指数

类型	数学函数
描述	**指数函数。**
语法	EXP([base,] expvalue) base：底数，缺省为 e expvalue：指数
适用控制器	通用
例子	例一 PRINT　**EXP**(2,4)　　'结果是 16 (2*2*2*2) 例二 PRINT　**EXP**(1)　　'结果是 2.7183

SQR -- 平方根

类型	数学函数
描述	**平方根函数。**
语法	SQR(expression) expression：任意有效的表达式
适用控制器	通用

例子	a= **SQR**(4) PRINT a '结果是 2

LN -- 自然对数

类型	数学函数
描述	**自然对数函数。**
语法	LN(expression) expression：任意有效的表达式
适用控制器	通用
例子	a= **LN**(1) PRINT a '结果是 0

LOG -- 对数底为 10

类型	数学函数
描述	**对数，底数为 10。**
语法	LOG(expression) expression：任意有效的表达式
适用控制器	通用
例子	a= **LOG**(100) PRINT a '结果是 2

9.6 数据操作指令

SET_BIT -- 按位设置

类型	数学指令或函数
描述	**位操作，只对整数，修改对应位为 1。** 分为命令语法和函数语法。
语法	命令语法：SET_BIT(bit#, vr#)直接操作 VR 寄存器 bit#：位编号：0-31 vr#：要操作的 VR 变量编号，整数部分 命令语法使用时没有返回值，直接操作，修改操作对象的值。 函数语法：ret = SET_BIT(bit#,int) ret：操作结果 bit#：位编号：0-31 int：要操作的表达式，取整数部分

	函数语法使用时返回操作后的结果，操作对象的值不变。
适用控制器	通用
例子	例一 命令语法 VR(23)=0.333 **SET_BIT**(0,23) 'VR(23)的第 0 位将置为 1，并清除小数部分 ?VR(23) '结果为 1 例二 函数语法 DIM a,b a=0.333 b=0 b=**SET_BIT** (0,a) '设置 a 的第 0 位的，结果赋值到 b，并清除小数 PRINT a,b '打印结果 0.333，1 a 没有改变，b 为 1
相关指令	CLEAR_BIT，READ_BIT，READ_BIT2

CLEAR_BIT -- 按位置 0

类型	数学指令或函数
描述	**位操作，只对整数，修改对应位为 0。** 分为命令语法和函数语法。
语法	命令语法：CLEAR_BIT(bit#,vr#)直接操作 VR 寄存器 bit#：位编号：0-31 vr#：要操作的 VR 变量编号，整数部分 命令语法使用时没有返回值，直接操作，修改操作对象的值。 函数语法：ret = CLEAR_BIT(bit#,int) ret：操作结果 bit#：位编号：0-31 int：要操作的表达式，取整数部分 函数语法使用时返回操作后的结果，操作对象的值不变。
适用控制器	通用
例子	例一 命令语法 VR(23)=3.333 **CLEAR_BIT**(0,23) 'VR(23)的第 0 位将被清除(设置为 0) ?VR(23) '打印结果 2，小数被去除 例二 函数语法 DIM a,b a=3.333 b=0 b= **CLEAR_BIT**(0,a) '返回清除 a 的第 0 位和小数后的结果给 b PRINT a,b '结果为 3.333,2 a 不变,b 为 2

相关指令	SET_BIT，READ_BIT，READ_BIT2

READ_BIT -- 按位读取

类型	数学函数
描述	**位操作，只对整数，读取对应位状态。** 只能操作 VR 寄存器，非 VR 参考 READ_BIT2
语法	ret = READ_BIT(bit#, vr#) ret：读取结果：1 或 0 bit#：位编号：0-31 vr#：要操作的 VR 变量编号
适用控制器	通用
例子	VR(23)=3.333 PRINT **READ_BIT**(0,23)　　'读取 VR(23)的第 0 位，结果为 1
相关指令	SET_BIT，CLEAR_BIT，READ_BIT2

READ_BIT2 -- 按位读取 2

类型	数学函数
描述	**位操作，只对整数，读取对应位状态。**
语法	ret = READ_BIT2(bit#, int) ret：读取结果：1 或 0 bit#：位编号：0-31 int：要操作的表达式，取整数部分
适用控制器	通用，20130813 以后的固件版本提供了支持
例子	DIM a,b b=1.64 a=**READ_BIT2**(0,b)　　'读取 b 的第 0 位,赋值给 a PRINT a　　'输出 a 值，结果为 1
相关指令	SET_BIT，，READ_BIT

FRAC -- 返回小数

类型	数学函数
描述	**返回小数部分，总是大于 0 的部分。**
语法	FRAC(expression) expression：要操作的数
适用控制器	通用

例子	a=**FRAC**(1.235) PRINT a '结果是 0.235

INT -- 返回整数

类型	数学函数
描述	**返回整数部分。**
语法	INT(expression) expression：任意有效的表达式
适用控制器	通用
例子	a=INT(1.235) PRINT a '结果是 1 ?**INT**(-1.1) '打印结果，-2，因为小数部分总处理为正数。

SGN -- 返回符号

类型	数学函数
描述	**返回符号。** 1 大于 0 0 等于 0 -1 小于 0
语法	SGN(expression) expression：任意有效的表达式
适用控制器	通用
例子	a=**SGN**(-1.235) PRINT a '结果是-1

IEEE_IN -- 组合浮点数

类型	数学函数
描述	**把四个字节组合成一个单精度浮点数。**
语法	IEEE_IN(byte0,byte1,byte2,byte3) byte0－byte3：四个字节
适用控制器	通用
例子	VAR = **IEEE_IN**(VR(10),VR(11),VR(12),VR(13)) '将 VR(10)~VR(13)这四个数据合成一个单精度浮点数

IEEE_OUT -- 提取单字节

类型	数学函数
描述	**从一个单精度浮点数里面取一个字节。**
语法	byte_n = IEEE_OUT(var, n) var：单精度浮点数 N：0-3，取第几个字节
适用控制器	通用
例子	例一 VAR = **IEEE_OUT**(VR(1),2)　　'提取 VR(1)的第二个字节 例二 GLOBAL VAR0,VAR1,VAR2,VAR3 VAR0=0 VAR1=0 VAR2=0 VAR3=0 VR(1)=123.456 VAR0 = IEEE_OUT(VR(1),0) VAR1 = IEEE_OUT(VR(1),1) VAR2 = IEEE_OUT(VR(1),2) VAR3 = IEEE_OUT(VR(1),3) VR(2)=0 VR(2)=IEEE_IN(VAR0,VAR1,VAR2,VAR3) 运行结果： 

$ -- 16 进制

类型	特殊字符
描述	**表示紧接着的数据是 16 进制格式。**
语法	$hexnum
适用控制器	通用
例子	在线命令输入 >>PRINT　$F

	输出：15

9.7 字符串操作指令

CHR -- ASCII 码打印

类型	字符串函数
描述	**返回 ASCII 打印，只用于 PRINT。**
语法	CHR(expression) expression：任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT **CHR**(66) 输出：B

HEX -- 16 进制打印

类型	字符串函数
描述	**返回十六进制格式，只用于 PRINT。**
语法	HEX(expression) expression：任意有效的表达式，只取整数部分
适用控制器	通用
例子	在线命令输入 >>PRINT **HEX**(15) 输出：f '十六进制

STRLEN -- 返回字符串长度

类型	字符串函数
描述	**返回字符串长度。**
语法	len=STRLEN(str) str：字符串
适用控制器	通用
例子	DIM str_a(20) str_a="len123" ?STRLEN(str_a) 打印结果：6

TOSTR -- 格式化输出

类型	字符串函数
描述	**格式化输出函数，变量转换成字符串。**
语法	TOSTR(VAR1, [N],[DOT]) VAR1：任意有效的表达式 N：输出数据总位数，包括小数点和符号位。当 N 设置负数时，表示右对齐 DOT：输出的小数个数，当 N 太小没有小数位置时，不输出小数 输出的是字符串类型。只有第一个参数默认打印到小数点后四位。
适用控制器	通用
例子	例一 在线命令输入 >> PRINT **TOSTR**(2-100,6,2) 输出：-98.00 例二 DIM aa(20) aa="asd13"+**TOSTR**(354) ?aa　　'打印结果 asd13354.0000

STRCOMP -- 字符串比较

类型	字符串函数
描述	**字符串比较函数，根据两个字符串的情况返回>0、=0、<0。** 比较长度在 500 字节内，否则返回值出错误。
语法	STRCOMP(str1, str2) str1：字符串 1 str2：字符串 2
适用控制器	通用
例子	DIM　AAA(10) AAA = "abc" 在线命令输入 >>PRINT　**STRCOMP**(AAA, "abc") 输出：0

STRFIND -- 字符串搜索

类型	字符串函数

描述	**字符串搜索函数。**
语法	STRFIND(str1, str2 [, firstindex]) str1：待搜索的字符串 str2：搜索模板字符串 firstindex：从哪个位置开始搜索，缺省 0 返回：>= 0，返还查找到的 index；< 0，没有找到
适用控制器	通用
例子	DIM AAA(10),BBB(3) AAA="AD23GF41" BBB="23G" ?**STRFIND**(AAA,BBB)　'打印搜索到的索引位置，2

VAL -- 字符转数值

类型	字符串函数
描述	**字符串转换为数值。** 只能转换数字字符，遇到字母、符号字符停止。
语法	VAL(str1) str1：字符串
适用控制器	通用
例子	例一 VAR1 = **VAL**("123") ?VAR1　　'打印结果，123 例二 VAR2 = **VAL**("123QWE23") ?VAR2　　'打印结果，123

9.8 常数指令

PI -- 圆周率

类型	常数
值	3.14159
适用控制器	通用

TRUE -- 真值

类型	常数
值	-1
适用控制器	通用

FALSE -- 假值

类型	常数
值	0
适用控制器	通用

ON -- 开启

类型	常数
值	1
适用控制器	通用

OFF -- 关闭

类型	常数
值	0
适用控制器	通用

9.9 高级运算指令

CRC16 -- CRC 检验计算

类型	数学函数
描述	**CRC16 CCITT 计算。**
语法	CRC16(arrayname, index, size[, initial] [, poly]) arrayname：数据存储所在的数组，一个字节占一个位置 index：数据存储所在的数组起始索引 size：计算字节数 initial：CRC 计算初始值，缺省$FFFF

	poly：多项式，暂时只支持 modbus 的$A001 和 CCITT 的$1021，缺省$A001
适用控制器	通用
例子	TABLE(0, $FE, $48 , $06 , $00 , $6D , $00 , $00 , $00)　'8 个数据存储在 TABLE CRCVALUE = **CRC16**(TABLE, 0, 8)　'计算 CRC，结果$1A0D TABLE(8)= CRCVALUE\256　'计算的 CRC 加在数据的后面，大端模式 TABLE(9)= CRCVALUE AND $FF

DTSMOOTH -- table 平滑

类型	数学函数
描述	**对 TABLE 存储的点坐标进行平滑调整。**
语法	DTSMOOTH (axis, dtfirst, space, points,　imode,　referradius) axis：轴数 dtfirst：第一个点的 TABLE 索引 space：两个点之间的索引间隔(就是一个点存储占用的空间) points：总点个数 imode：0- 绝对方式，对曲率半径小于参考值的进行调整 referradius：参考曲率半径，可以根据 半径=(速度平方)/拐弯加速度来计算参考
适用控制器	3X 系列 20161206 以上固件支持 4 系列 20170508 以上固件支持
例子	TABLE(0, 0,0) TABLE(5, 99,0) TABLE(10, 100,0) TABLE(15, 100, 1) TABLE(20, 101, 1) TABLE(25, 200, 1) **DTSMOOTH**(2, 0, 5,　6, 0, 5) ?*TABLE(0, 2) ?*TABLE(5, 2) ?*TABLE(10, 2) ?*TABLE(15, 2) ?*TABLE(20, 2) ?*TABLE(25, 2)

B_SPLINE -- B 样条平滑

类型	数学函数
描述	**将 TABLE 中的数据进行 B 样条平滑。**
语法	B_SPLINE(type, data_start, points, data_out, ratio) type：类型，目前只支持 1-B 样条

	data_start：图形数据在 TABLE 中的起始位置 points：图形数据的个数 data_out：平滑后的图形数据在 TABLE 中起始位置 ratio：B_SPLINE 函数的平滑比率，平滑后的个数为 points * ratio
适用控制器	通用
例子	**B_SPLINE**(1,0,10,100,10)　　'平滑一个 10 点的图形数据，源图形点在 table 的位置为从 0 到 9，平滑为 100 点的数据，并且平滑后的数据从 table 地址 100 开始存放

TURN_POSMAKE -- 旋转坐标计算

类型	数学函数
描述	**旋转功能的计算函数，计算旋转台上点(X,Y)旋转 R 度后点的绝对坐标，旋转的正向与 XY 的正向要一致（右手法则）。**
语法	TURN_POSMAKE(tablenum, posx, posy, disR, tableout) tablenum：旋转功能参数存储 table 编号 posx：X 方向的坐标 posy：Y 方向的坐标 disR：旋转轴的相对偏移 tableout：存储计算后的坐标
适用控制器	通用
相关指令	MCIRC_TURNABS

ZCUSTOM -- 运动参数计算

类型	数学函数
描述	**计算各种运动指令中的参数，详细功能请看下方语法功能描述。**
语法	功能 2：计算空间圆弧或直线上一定距离后的点的位置。 Table 表参数按三点画圆模式填写其他两个点参数。 填写相对坐标，返回也是相对的位置。 语法：ZCUSTOM(2,tableend,tablemid,tableout,mode,vectdis) tableend：存储圆弧终点的 table 索引 tablemid：存储圆弧中间点的 table 索引，与当前点一起构成圆弧的 3 个点 tableout：输出计算数据的 table 索引 mode：模式

值	描述
1	相对起点的空间圆弧弧长
2	相对终点的空间圆弧弧长
3	相对起点的直线距离，此时 tablemid 不起作用
4	相对终点的直线距离，此时 tablemid 不起作用
5	相对计算空间圆弧的圆心，此时 vectdis 无效。此时 tableout 依次输出

	xyz 圆心，弧度范围，弧长

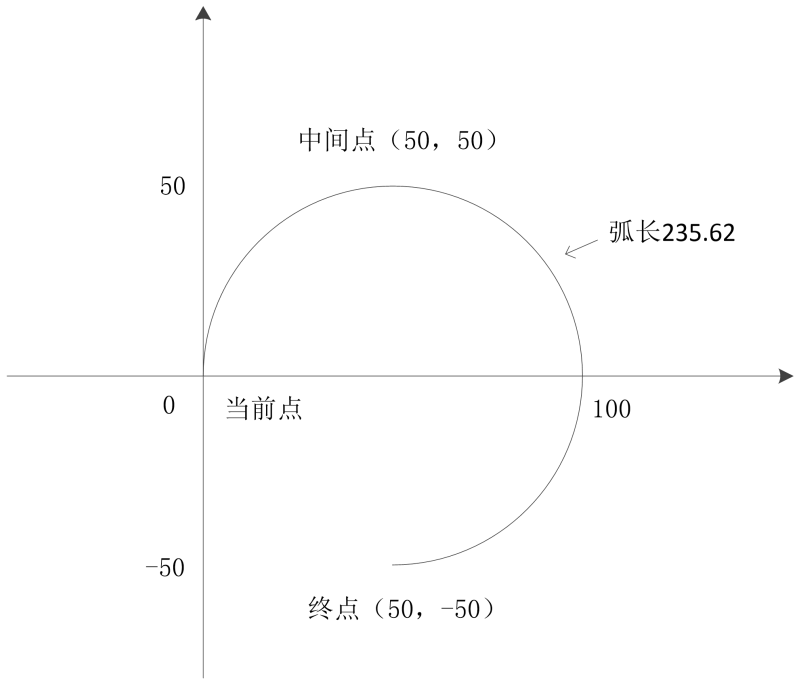
vectdis：要计算的点相对 mode 的距离，负数表示往前。圆弧模式时，正数表示顺时针，负数表示逆时针

功能 6：计算空间圆弧的起点和终点的切线角度方向，弧度单位。
填写相对坐标，返回也是相对的位置。
语法：ZCUSTOM(6,tableend,tablemid,tableout)
tableend：存储圆弧终点的 table 索引
tablemid：存储圆弧中间点的 table 索引，与当前点一起构成圆弧的 3 个点
tableout：输出计算数据的 table 索引，依次输出:起点 xy 方向，起点 Z 方向，终点 xy 方向，终点 Z 方向，角度弧度单位

功能 7：输入速度比例，计算 MOVESLINK 的从轴位置，主轴位置。
语法：ZCUSTOM(7, distance, link dist, start sp, end sp, 速度比例, tableout)
distance：从连接开始到结束，跟随轴移动的距离，采用 units 单位
link dist：参考轴在连接的整个过程中移动的绝对距离，采用 units 单位
star sp：启动时跟随轴和参考轴的速度比例，units/units 单位，负数表示跟随轴负向运动
end sp：结束时跟随轴和参考轴的速度比例，units/units 单位，负数表示跟随轴负向运动
速度比例：需要计算的点的速度比例，与参数里面的起始结束点比例意义相同
tableout：正向找的从轴距离，正向找的主轴距离，从反向找的从轴距离，反向找主轴距离，占用 4 个 TABLE（一段曲线，速度比例可能会有多个解）

功能 8：MOVESLINK 输入从轴位置，反算主轴的位置。
语法：ZCUSTOM(8,distance,link dist,start sp,end sp,distancemoved, tableout)
distance：从连接开始到结束，跟随轴移动的距离
link dist：从连接开始到结束，主轴移动的绝对距离
start sp：起始脉冲速度比例
end sp：结束脉冲速度比例
distancemoved：从轴已经运动距离
tableout：输出 table 索引，输出对应主轴的位置，如多个解返回第一个

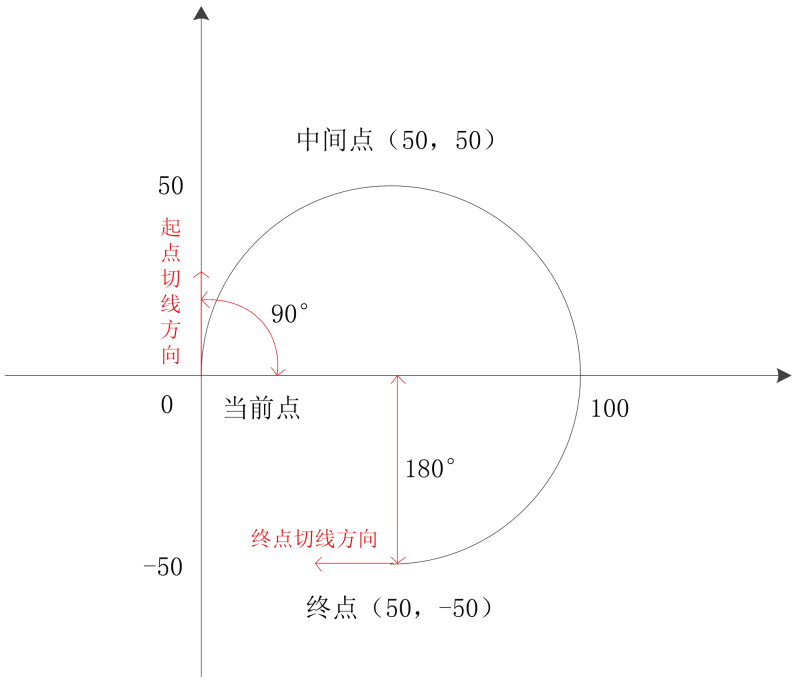
功能 9：FLEXLINK 输入从轴位置，反算主轴的位置。
语法：ZCUSTOM (9, base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, distancemoved, tableout)
base_dist：跟随轴匀速运动距离
excite_dist：跟随轴激励运动距离，这个与 base_dist 相反的时候无法反算
link_dist：整个指令过程，跟随轴运动完成，主轴移动的距离
base_in：激励开始前，跟随轴运动距离占 base_dist 的百分比
base_out：激励完成后，跟随轴剩余运动距离占 base_dist 的百分比，两者相加不要超过 100%
excite_acc：激励过程中，跟随轴加速阶段运动距离占 excite_dist 的百分比，excite_dist 为负值时，为减速阶段

	excite_dec：激励过程中，跟随轴减速阶段运动距离占 excite_dist 的百分比，excite_dist 为负值时，为加速阶段 distancemoved：从轴已经运动脉冲数 tableout：输出 Table 索引，输出对应主轴的位置，如多个解返回第一个 功能 10：从工件坐标上的三点在原来坐标系上的坐标来计算 FRAME_ROTATE 旋转转换的参数，每个点要存储 xyz 的三个方向的坐标。 语法：ZCUSTOM (10, dtzero, dtx, dty, dtout) dtzero：工件零点在原来坐标系上的位置 dtx：工件坐标系 X 轴上的点在原来坐标系上的位置 dty：工件坐标系 Y 轴上的点在原来坐标系上的位置 dtout：输出 TABLE 索引，分别存储：X，Y，Z，RX，RY，RZ
适用控制器	通用
例子	例一 功能 2 使用  TABLE(0,50,-50,0)　　'设置终点坐标，相对位置 TABLE(3,50,50,0)　　'设置中间点坐标，相对位置 '模式 1，相对起点 **ZCUSTOM(2,0,3,10,1,78.54)**　'相对起点位置顺时针弧长为 78.54 的圆上点 ?TABLE(10),TABLE(11),TABLE(12)　'输出相对坐标为 50,50,0 **ZCUSTOM(2,0,3,10,1,-78.54)**　'相对起点位置逆时针弧长为 78.54 的圆上点 ?TABLE(10),TABLE(11),TABLE(12)　'输出相对坐标为 50,-50,0 '模式 2，相对终点 **ZCUSTOM(2,0,3,10,2,78.54)**　'相对终点位置顺时针弧长为 78.54 的圆上点 ?TABLE(10),TABLE(11),TABLE(12)　'输出坐标为 50,-50,0 **ZCUSTOM(2,0,3,10,2,-78.54)**　'相对终点位置逆时针弧长为 78.54 的圆上点

?TABLE(10),TABLE(11),TABLE(12)　'输出坐标为 100,0,0

'模式 5，计算此时三点画圆的圆心、弧度、弧长
ZCUSTOM(2,0,3,10,5,0)　'此时距离参数不起作用
?TABLE(10),TABLE(11),TABLE(12)　'输出坐标为 50,0,0
?TABLE(13),TABLE(14)　　'输出弧度 4.712，弧长 235.62

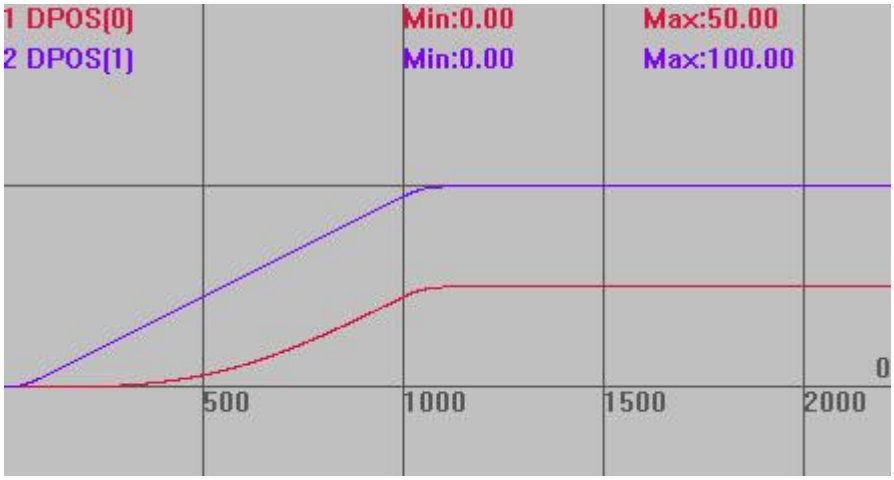
例二 功能 6，计算圆弧起点和终点的切线弧度



TABLE(0,50,-50,0)　　'设置终点坐标，相对位置
TABLE(3,50,50,0)　　'设置中间点坐标，相对位置
ZCUSTOM(6,0,3,10)　　'计算当前点和终点切线的弧度
?TABLE(10),TABLE(11)　'输出当前点 1.571,0　(1.571=90*PI/180)
?TABLE(12),TABLE(13)　'输出终点-3.142,0　(-3.142=-180*PI/180)

例三 功能 8，反算 MOVESLINK 主轴位置

```
BASE(0,1)
DPOS=0,0
Units=100,100
SPEED=100,100
ACCEL=1000,1000
TRIGGER
MOVESLINK(50,100,0,1,1)　'建立 MOVESLINK 连接
MOVEABS(100)　AXIS(1)
ZCUSTOM(8,50,100,0,1,25,10) '根据建立连接的参数，计算从轴运动 25 时主轴的位置
?TABLE(10)　　'输出主轴位置 73.801
```

	运动轨迹 DPOS(0)垂直刻度 100 DPOS(1)垂直刻度 100 

ZMATH64 -- 64 位计算

类型	64 位计算指令
描述	**对存储 D 寄存器(MODBUS 寄存器)的 64 位数进行计算。** 一个 64 位有符号整数占用 4 个寄存器(小端模式)。 只操作 MODBUS 寄存器，不处理 VR 映射等。 4 系列控制器 20170629 固件以上版本支持。
语法	ZMATH64(opmode, dindex1, dindex2) opmode：操作编号 dindex1、dindex2：MODBUS 寄存器编号

操作编号	执行操作	说明
1	64 位整数加法	D64(dindex1) += D64(dindex2)
2	64 位整数减法	D64(dindex1) -= D64(dindex2)
3	64 位整数乘法	D64(dindex1) *= D64(dindex2)
4	64 位整数除法	D64(dindex1) /= D64(dindex2)
5	64 位整数余	D64(dindex1) %= D64(dindex2)
11	64 位整数读取	D64(dindex1) = D32(dindex2)
12	64 位整数转换	D32(dindex1) = D64(dindex2)
13	64 位整数读取	D64(dindex1) = D32IEEE(dindex2)
14	64 位整数转换	D32IEEE(dindex1) = D64(dindex2)
15	赋值	D64(dindex1) = double64(dindex2)
16	赋值	double64(dindex1) = D64(dindex2)
17	赋值	double64 (dindex1) = D32IEEE (dindex2)
18	赋值	D32IEEE (dindex1) = double64 (dindex2)

	<table><tr><td></td><td></td><td></td></tr><tr><td>21</td><td>double 加法</td><td>double64 (dindex1) += double64 (dindex2)</td></tr><tr><td>22</td><td>double 减法</td><td>double64 (dindex1) -= double64 (dindex2)</td></tr><tr><td>23</td><td>double 乘法</td><td>double64 (dindex1) *= double64 (dindex2)</td></tr><tr><td>24</td><td>double 除法</td><td>double64 (dindex1) /= double64 (dindex2)</td></tr><tr><td>25</td><td>double 余，求小数</td><td>double64 (dindex1) %= double64 (dindex2)</td></tr></table> D32IEEE 表示浮点数存储，同 MODBUS_IEEE。 D64 表示 64 位有符号整数存储，可以通过两个 MODBUS_LONG 来读取高 32 和低 32 位。
适用控制器	通用
例子	MODBUS_LONG(0)=100 MODBUS_LONG(8)=20 **ZMATH64**(1,8,0) '64 位整数加法计算，两个数相加后存储在 MODBUS_LONG(8)起始地址 ?MODBUS_LONG(0) '打印结果，100 ?MODBUS_LONG(8) '打印结果，120
相关指令	MODBUS_IEEE，MODBUS_LONG，MODBUS_REG

MODBUS_DOUBLE -- 读取 MODBUS

类型	64 位指令
描述	**从 MODBUS 读取 double 数据，可以赋值给其它变量数组。** 3 系列和 3 系列以下等数组为 float 的不支持这个指令。
语法	MODBUS_DOUBLE(index) Index：modbus 寄存器编号
适用控制器	通用
例子	MODBUS_LONG(0)=100 MODBUS_LONG(8)=200 ZMATH64(16, 0, 8) '64 位赋值 ?**MODBUS_DOUBLE**(0) '打印结果，200 ?MODBUS_LONG(0) '打印结果，0 ?MODBUS_LONG(8) '打印结果，200
相关指令	ZMATH64

第十章 轴参数与轴状态指令

轴参数修改语法：SPEED = value,针对 BASE 轴。也可以通过 SPEED AXIS(axisnum) = value 来强制取特定轴，其中 axis 可以省掉，变为 SPEED(axisnum)=value。

可以通过 SPEED=value1,vlaue2…来同时设置多个 BASE 轴的参数。

轴参数可写可读，读取时 axis 也可以省掉：VAR1 = SPEED(axisnum)。

轴状态一般是只读的，值会随时内部变动，MPOS,DPOS 等可以直接修改的除外。

⚠当进行插补运动时，主轴的参数作为插补的参数，BASE 多个轴时，第一个轴作为主轴。

10.1 轴选择

BASE -- 轴选择/轴组选择

类型	轴参数
描述	**选择要设置参数、参与运动的轴。** 缺省值依次为：0，1，2… 程序中在下一条 BASE 指令被执行前，都以上一条 BASE 指令选择轴。 每个任务拥有各自独立的轴列表，会记住任务中 BASE 选择的轴或者轴组，用于不同机台的控制。 当插补运动的时候，第一个轴的运动参数作为插补参数。见例一。 如果在 BASE 指令中没有列出所有的轴，BASE 指令自动将剩余轴顺序排列在后面。见例二。
语法	BASE(axis<,second axis><,third axis>...) axis：第一个轴 second axis：下一个轴 … 参数最多为控制器支持的轴数，参照对应控制器硬件手册。
适用控制器	通用
例子	例一 **BASE**(0,1,2,3) '轴列表选择为：0,1,2,3 SPEED=100,10,20,30 '此时轴 0,1,2,3 设置了对应速度，但是插补运动时，只有主轴轴 0 的速度 100 起作用 MOVE(100,100,100,100) '轴 0,1,2,3 联合插补运动，合成运动的速度为 100，各轴速度为分速度 例二 **BASE**(1) '轴列表选择为：1

	MOVE(100,100,100) '轴 1,2,3 做插补运动 例三 **BASE**(0,2,5) '轴列表选择为：0,2,5 MOVE(100,100,100) '轴 0,2,5 做插补运动

AXIS -- 临时轴选择

类型	辅助指令
描述	**临时修改一个运动指令或轴参数到一个指定轴上去执行。** 对轴参数，AXIS 可以省掉。
语法	AXIS(expression) expression：临时修正的新轴号，执行完后，轴选择仍以 BASE 指令为准
适用控制器	通用
例子	例一 BASE(0) MOVE(1000) **AXIS**(1) '此时强制轴 1 运动 1000units MOVE(100) '轴 0 运动 100units 例二 BASE(1) UNITS **AXIS**(0)=100 '强制指定轴 0 UNITS 设为 100 '可简写为 UNITS(0)=100 UNITS(2)=200 '强制指定轴 2 UNITS 设为 200 UNITS=10 '轴 1 UNITS 设为 10
相关指令	BASE

10.2 基本参数指令

UNITS -- 脉冲当量

类型	轴参数
描述	**脉冲当量，指定每单位发送的脉冲数，支持 5 位小数精度。** 控制器以 UNITS 作为基本单位，修改后坐标显示会随 UNITS 改变比例变化。 比如：UNITS=10 对应 DPOS=3000，MPOS=3000 修改 UNITS=100 后 对应 DPOS=300，MPOS=300
语法	可读：VAR1 = UNITS 或 VR1=UNITS(轴号) 可写：UNITS= expression 或 UNITS(轴号)= expression

适用控制器	通用
例子	**如何设置** 假设电机 U=3600 脉冲转一圈，丝杠一圈螺距 P=2mm 电机转 1° 对应的 UNITS： **UNITS**=U/360=3600/360=10，此时 MOVE(1)，电机转 1° 工作台走 1mm 对应的 UNTIS： **UNITS**=U/P=3600/2=1800，此时 MOVE(1)，工作台走 1mm 机台存在减速比时，要把减速比算上，假设减速比 i=2:1 **UNITS**=U*i/P=3600*2/2=3600 **编程使用** BASE(0,1,2)　　'选择轴 0，轴 1，轴 2 **UNITS**=10,100,1000,30　　'轴 0 设为 10，轴 1 设为 100，轴 2 设为 1000，轴 3 设为 30 与 BASE 联合使用时，数据多出 BASE 选择的轴数，依次往后设置轴。 数据少于 BASE 选择的轴数，就只设置对应的轴。 **UNTIS**(2)=100　　'直接设置轴 2，与 BASE 选择轴无关

ATYPE -- 轴类型

类型	轴参数
描述	**轴功能类型设置，只能设置为轴具备的特性**（轴特性可查询硬件手册或使用 ZDevelop 软件连接上控制器以后查看控制器状态）。 最好是在程序初始化的时候就设置好 ATYPE。 ZCAN 扩展轴要先设置 AXIS_ADDRESS，并在设置后延迟 2 个 tick 再调用运动指令，受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。 对部分产品型号带有独立的编码器，可以使用相应虚拟轴来做编码器轴使用，例如 ZMC206 的电机轴为 0-5 轴，编码器可以通过轴 6-11 来控制，详细可通过 ZDevelop 软件连接上控制器以后查看控制器状态。
语法	VAR1 = ATYPE，ATYPE = expression

ATYPE 类型	描述
0	虚拟轴
1	脉冲方向方式的步进或伺服
2	模拟信号控制方式的伺服
3	正交编码器
4	脉冲方向输出+正交编码器输入
5	脉冲方向输出+脉冲方向编码器输入
6	脉冲方向方式的编码器
7	脉冲方向方式步进或伺服+EZ 信号输入
8	ZCAN 扩展脉冲方向方式步进或伺服
9	ZCAN 扩展正交编码器

<table>
<tr><td></td><td>
<table>
<tr><td>10</td><td>ZCAN 扩展脉冲方向方式的编码器</td></tr>
<tr><td>21</td><td>振镜轴类型，需要控制器支持
缺省系统周期 250u，振镜刷新周期 50u，与固件有关
可以使用普通轴的所有运动控制指令，支持振镜轴与其它轴类型混合插补</td></tr>
<tr><td>50</td><td>RTEX 周期位置模式，需 Rtex 控制器</td></tr>
<tr><td>51</td><td>RTEX 周期速度模式，需 Rtex 控制器</td></tr>
<tr><td>52</td><td>RTEX 周期力矩模式，需 Rtex 控制器
请先关闭驱动器 2 自由度控制模式，并设置设置速度限制</td></tr>
<tr><td>65</td><td>ECAT 周期位置模式，需支持 EtherCAT</td></tr>
<tr><td>66</td><td>ECAT 周期速度模式，需支持 EtherCAT
Profile 要设置为 20 或以上</td></tr>
<tr><td>67</td><td>ECAT 周期力矩模式，需支持 EtherCAT
PROFILE 要设置为 30 或以上</td></tr>
<tr><td>70</td><td>ECAT 自定义操作，只读取编码器，需支持 EtherCAT</td></tr>
</table>

电机模式 INVERT_STEP 指令设置，默认脉冲方向</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>例一 脉冲类型
BASE(0,1)
ATYPE = 1,1 '轴 0,1 设为脉冲控制类型
UNITS=100,100 '脉冲当量设为 100
SPEED=100,100 '速度 100 units/s
ACCEL=1000 ,1000 '加速度 1000 units/s/s
DECEL=1000 '减速度 1000 units/s/s
MOVE(100,100) '直线插补

例二 EtherCAT 总线控制
SLOT_SCAN(0) '总线扫描
BASE(0)
AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
ATYPE(0)=65 '轴类型 65，位置控制
SLOT_START(0) '开启总线
AXIS_ENABLE=1 '轴使能
WDOG=1 '所有轴使能
UNITS=100 '脉冲当量设为 100
SPEED=100 '速度 100 units/s
ACCEL=1000 '加速度 1000 units/s/s
DECEL=1000 '减速度 1000 units/s/s
MOVE(5000)

例三 Rtex 力矩模式
SLOT_SCAN(0) '总线扫描
BASE(0)</td></tr>
</table>

	AXIS_ADDRESS(0)=1　　'第一个驱动器映射到轴 0 **ATYPE(0)=52**　　'轴类型 52，Rtex 力矩控制 DRIVE_WRITE(6*256+47,0)　'关闭 2 自由度控制 DRIVE_WRITE(3*256+17,0)　'选择参数 3.21 作为速度限制 DRIVE_WRITE(3*256+21,2000)　'最大速度限制为 2000r/min SLOT_START(0)　　'开启总线 AXIS_ENABLE=1　　'轴使能 WDOG=1　　'所有轴使能 DAC=100　　'此时 DAC 发送值控制，具体查看 DAC 指令 例四 振镜轴 BASE(4,5) UNTIS=1,1 **ATYPE=21,21**　　'设置为振镜轴
相关指令	AXIS_ADDRESS，INVERT_STEP

AXIS_ADDRESS -- 轴地址设置

类型	轴参数
描述	**扩展轴时的轴地址配置**。 **1.ZCAN 扩展轴时**，扩展板上带 8 位拨码开关（V1.3 以上硬件版本）。 受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。 必须先设置 AXIS_ADDRESS，再设置 ZCAN 扩展轴 ATYPE 类型，修改后必须重新设置 ATYPE。见例一。 位 1-4：CAN 地址拨码，组合值 0-15 位 5-6：CAN 速度拨码，不同组合值速度不同 位 7：特殊功能预留 位 8：120 欧姆电阻拨码，拨 ON 时电阻接通 规则：AXIS_ADDRESS(轴号)=(32*0)+ID　　'扩展板的本地轴接口 0 AXIS_ADDRESS(轴号)=(32*1)+ID　　'扩展板的本地轴接口 1 **2.总线驱动器映射轴号**，将连接的驱动器按编号一一映射。 驱动器编号根据接线顺序排列，编号从 0 开始到 EtherCAT 驱动器个数减 1。 驱动器编号与设备号不同，设备号包括 ECAT 接口所有连接的设备，驱动器编号只算连接的驱动器。 必须先设置 AXIS_ADDRESS，再设置 ECAT 轴 ATYPE 类型，修改后必须重新设置 ATYPE。见例二。 位 0-15：驱动器编号加 1，0-自动指定 位 16-31：SLOT 编号（多槽位时） 规则：AXIS_ADDRESS(轴号)=(槽位号<<16)+驱动器编号+1

<table>
<tr><td></td><td>
3.本地脉冲轴号重映射，4 系列控制器支持本地脉冲或编码器轴号重新映射，固件 160608 以上版本支持。
重新映射时注意先把原本地脉冲轴设置为虚拟轴。修改后必须重新设置 ATYPE。见例四。
<table>
<tr><td>位 0-15</td><td>映射的本地脉冲轴号</td></tr>
<tr><td>位 16-31</td><td>高 16 位全设为 1（等同于十进制下高 16 位= -1）</td></tr>
</table>
规则：BASE(重映射的轴号)
ATYPE=0，设置轴类型为 0，低版本不设置会报错
BASE(要修改的本地脉冲轴号)
ATYPE=0，设置轴类型为 0
AXIS_ADDRESS(重映射的轴号)= (-1<<16) +要修改的本地脉冲轴号
BASE(重映射的轴号)
ATYPE=1/7
</td></tr>
<tr><td>语法</td><td>VAR1 = AXIS_ADDRESS，AXIS_ADDRESS = expression</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>
例一 ZCAN 扩展轴
AXIS_ADDRESS (6)=2+(32*1) '轴 6 映射到 ZCAN 扩展模块 ID2 的轴 1
ATYPE(6)=8 'ZCAN 扩展轴类型 脉冲方向方式步进或伺服
UNITS(6)=100 '脉冲当量 100
SPEED(6)=100 '速度 100uits/s
ACCEL(6)=1000 '加速度 1000units/s/s
MOVE(100) AXIS(6) '扩展轴运动 100units

例二 EtherCAT 手动映射轴号
AXIS_ADDRESS (0)=0+1 '第一个 Ecat 驱动器，编号 0，绑定为轴 0
AXIS_ADDRESS (2)=1+1 '第二个 Ecat 驱动器，编号 1，绑定为轴 2
AXIS_ADDRESS (1)=2+1 '第三个 Ecat 驱动器，编号 2，绑定为轴 1
ATYPE(0)=65 '设置为 Ecat 类型
ATYPE(1)=65
ATYPE(2)=65

例三 EtherCAT 自动映射轴号
AXIS_ADDRESS (0)=0 '自动指定 slot0 的驱动器，按照接线顺序，从轴 0 开始依次映射轴号(不建议使用，建议例二)
ATYPE(0)=65 '轴 0 配置为 ECAT 模式

例四 EtherCAT 控制器修改脉冲轴号
'修改前，操作轴 0，对应控制器上的轴 0 接口
BASE(16) '重映射的轴号
ATYPE(16)=0
BASE(0) '要修改的本地脉冲轴号
ATYPE(0)=0 '先将本地脉冲轴 0 设为虚拟轴
AXIS_ADDRESS (16)= (-1<<16)+0 '绑定本地脉冲轴 0，高 16 位=-1
ATYPE(16)=1 '轴 16 配置为脉冲轴，使用本地脉冲轴 0
'此时，操作轴 0，对应 ECAT 驱动器，操作轴 16，对应控制器轴 0 接口
</td></tr>
</table>

	例五 振镜轴号重映射 ATYPE(4)=0 ATYPE(5)=0 BASE(X) '希望映射的轴号 **AXIS_ADDRESS** = (-1<<16)+4 '重映射第一个振镜轴 ATYPE = 21 BASE(Y) '希望映射的轴号 **AXIS_ADDRESS** = (-1<<16)+5 '重映射第二个振镜轴 ATYPE = 21
相关指令	ATYPE

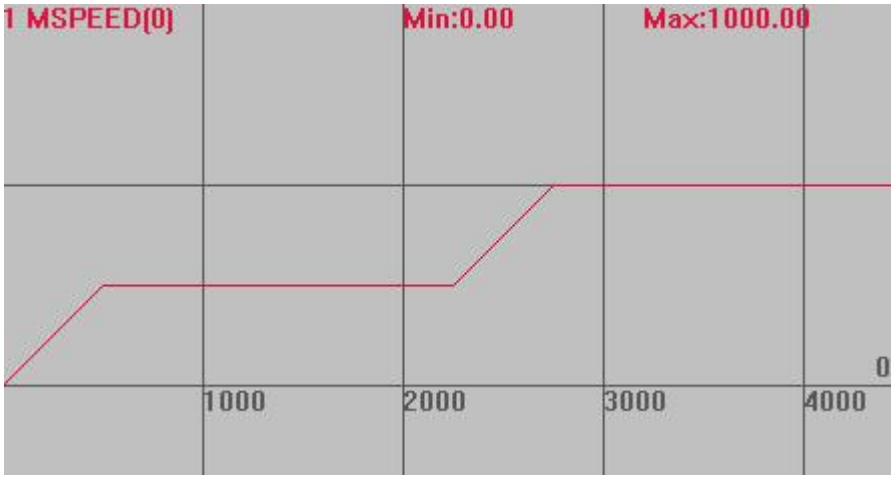
AXIS_ENABLE -- 单轴使能

类型	轴参数
描述	**各轴使能设置**。 EtherCAT 总线轴需设置，还必须设置 WDOG=1 总使能。
语法	AXIS_ENABLE = 1/0 1 使能，0 关闭
适用控制器	通用
例子	**AXIS_ENABLE**(0) = 1 '打开轴 0 使能
相关指令	WDOG

10.3 速度参数指令

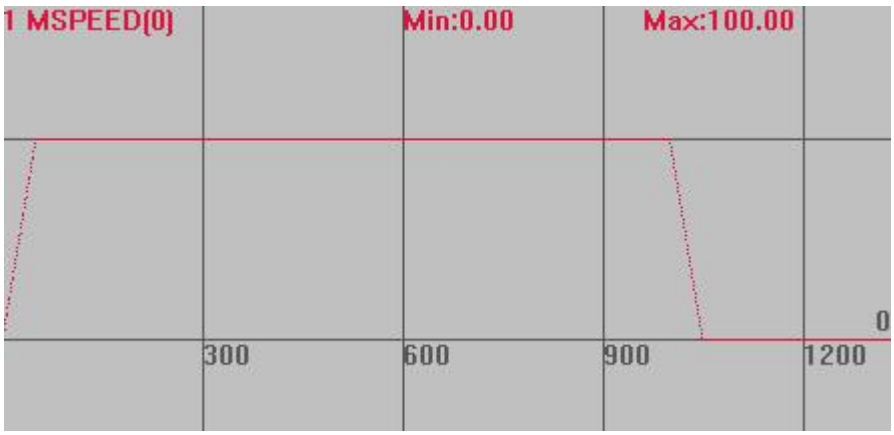
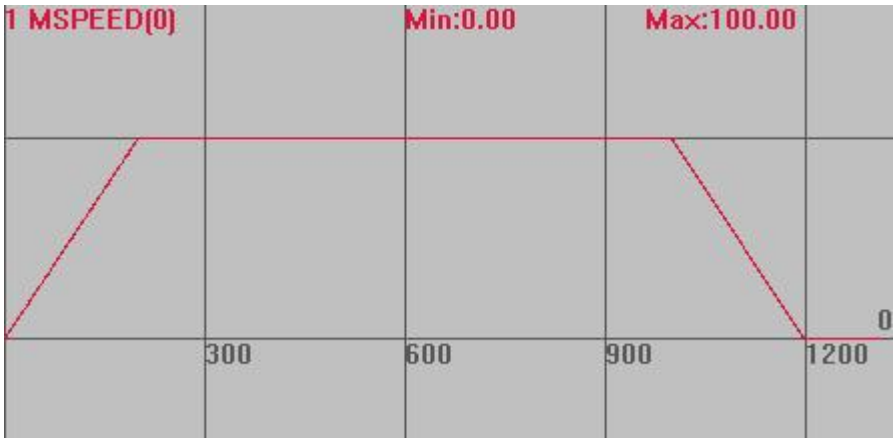
SPEED -- 运动速度

类型	轴参数
描述	**轴速度，单位为 units/s。** 当多轴运动时，作为插补运动的速度。 SPEED 修改后，立刻生效，可以实现动态变速，但是改变瞬间会抖。希望平滑变速请使用 SPEED_RATIO 指令。 当 SP 指令的 FORCE_SPEED 大于 SPEED 时，SPEED 速度也会生效(140716 以后版本 SPEED 不起作用)。
语法	VAR1 = SPEED，SPEED = expression
适用控制器	通用
例子	BASE(0) UNITS=100 '脉冲当量 **SPEED** =500 '设置 0 轴速度 500units/s ACCEL=1000 '加速度 1000units/s/s

	DPOS=0 '坐标清 0 TRIGGER '自动触发示波器 VMOVE(1) '持续运动 WAIT UNTIL DPOS(0)>1000 '等待轴 0 运行到 1000 **SPEED**=1000 '速度改为 1000 速度曲线 MSPEED(0)垂直刻度 1000 
相关指令	FORCE_SPEED，SPEED_RATIO

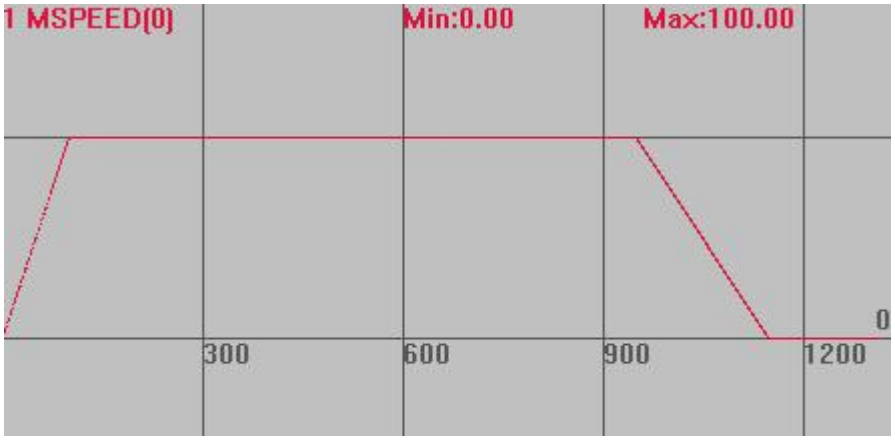
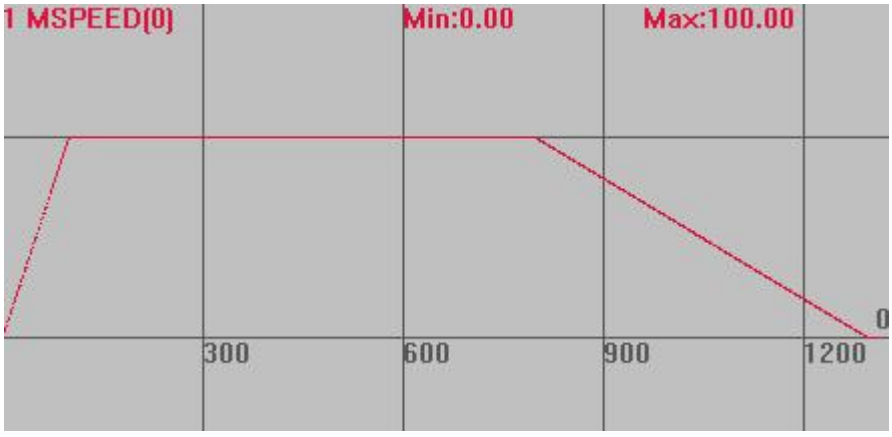
ACCEL -- 加速度

类型	轴参数
描述	**轴加速度，单位为 units/s/s。** 当多轴运动时，插补运动的加速度为主轴加速度。 建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。
语法	可读：VAR1 = ACCEL(轴号) 可写 ACCEL(轴号) = expression
适用控制器	通用
例子	例一 设置方式 BASE(1,2,3,4) 'BASE 选择轴 **ACCEL**=100, 100, 100, 100 '设置轴 1，2，3，4 的加速度 ACCEL(2)=200 '设置轴 2 的加速度 例二 BASE(0) UNITS=100 '脉冲当量 DPOS=0 '坐标清 0 **ACCEL**=2000 DECEL=2000 SPEED=100

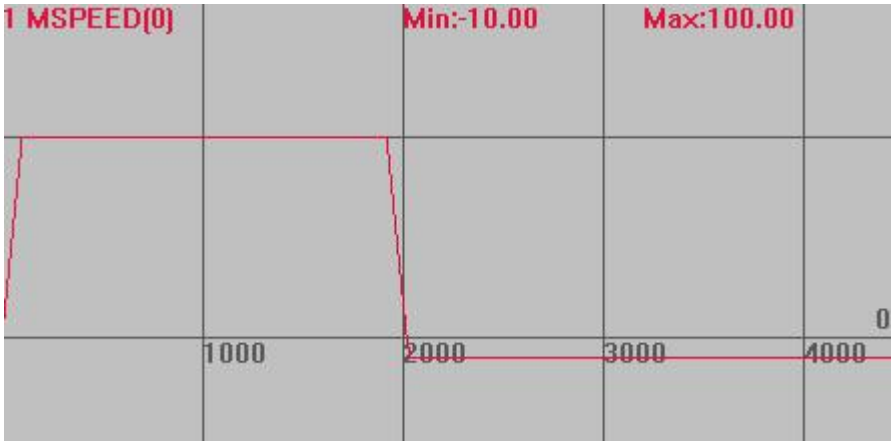
	MOVE(100) 加速过程 MSPEED(0)垂直刻度 100  **ACCEL**=500 DECEL=500 
相关指令	DECEL，SPEED

DECEL -- 减速度

类型	轴参数
描述	**轴减速度，单位为 units/s/s。** 当多轴运动时，作为插补运动的减速度。 当设置为 0 时，自动等于加速度 ACCEL 值，进行对称的加减速。 建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。
语法	VAR1 = DECEL，DECEL = expression
适用控制器	通用
例子	例一 BASE(1,2,3,4)

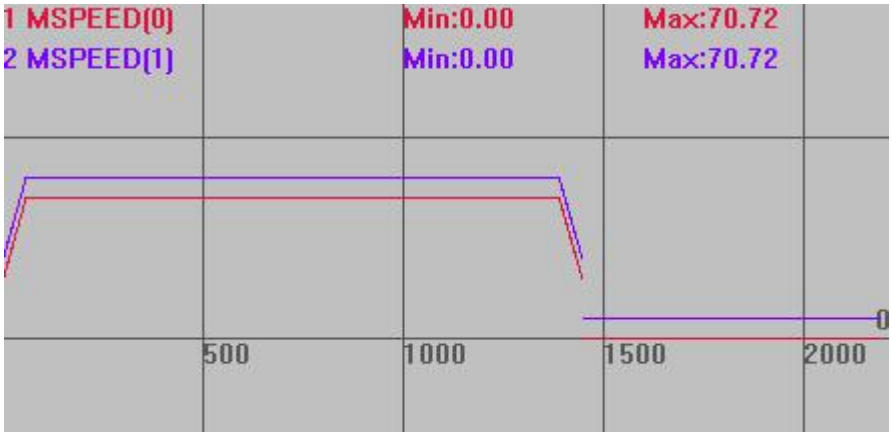
	DECEL =100, 100, 100, 100 '设置轴 1，2，3，4 的减速度 **DECEL** (0)=200 '设置轴 0 的减速度 PRINT DECEL (0) 例二 BASE(0) '选择轴 0 UNITS=100 '脉冲当量 DPOS=0 '坐标清 0 SPEED=100 '设置速度为 100 units/s ACCEL=1000 **DECEL**=500 '减速度为 500units/s/s TRIGGER '自动触发示波器 MOVE(200) '移动 200units 速度曲线 MSPEED(0)垂直刻度 100  DECEL=200 时 
相关指令	ACCEL，FASTDEC

CREEP -- 爬行速度

类型	轴参数
描述	**轴回零时爬行速度，用于原点搜寻，单位为 units/s。**
语法	VAR1 = CREEP，CREEP = expression
适用控制器	通用
例子	BASE(0) DPOS=0 UNITS=100 ACCEL=1000 DECEL=1000 SPEED = 100　　'运行速度设置为 100units/s **CREEP** = 10　　'爬行速度设置为 10units/s DATUM_IN=0　　'设置 IN0 为轴零原点 INVERT_IN(0,ON)　　'反转电平 TRIGGER　　'自动触发示波器 DATUM(3)　'先按速度 100 反找回零，遇原点后按速度 10 反向离开原点 速度曲线 MSPEED(0)垂直刻度 100 
相关指令	DATUM

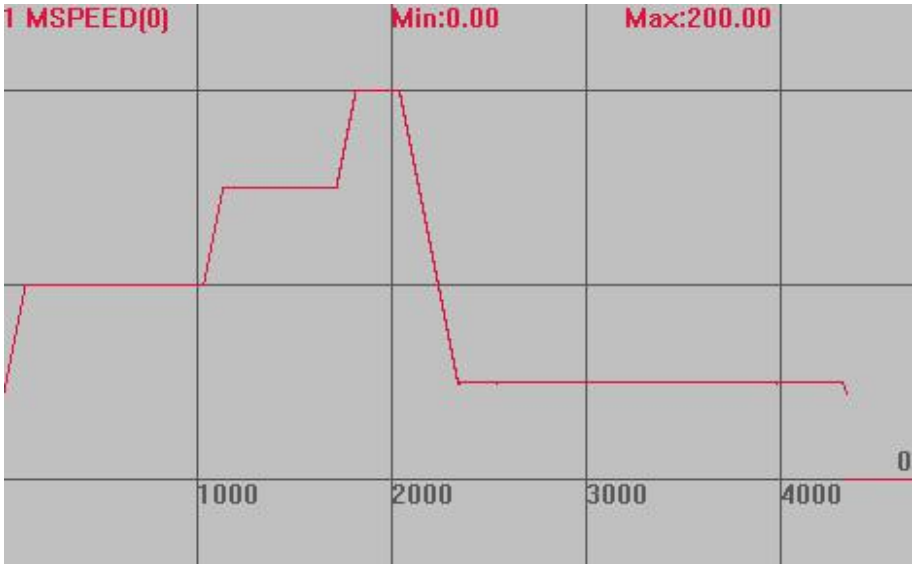
LSPEED -- 起始速度

类型	轴参数
描述	**轴起始速度，同时用于停止速度，缺省 0，单位为 units/s。** 当多轴运动时，作为插补运动的起始速度。 当需要追求效率时，可以考虑设置起始速度。
语法	VAR1 = LSPEED，LSPEED = expression
适用控制器	通用

例子	例一 BASE(0,1) '选择轴 0 为主轴 DPOS=0,0 UNITS=100,100 '脉冲当量 100 SPEED=100,100 '主轴速度 100units/s ACCEL=1000,1000 DECEL=1000,1000 **LSPEED**=40 '起始速度 40units/s TRIGGER '自动触发示波器 MOVE(100,100) '各轴运动距离 速度曲线 MSPEED(0)垂直刻度 100，无偏移 MSPEED(1)垂直刻度 100，偏移 10 
相关指令	SPEED

FORCE_SPEED -- SP 速度

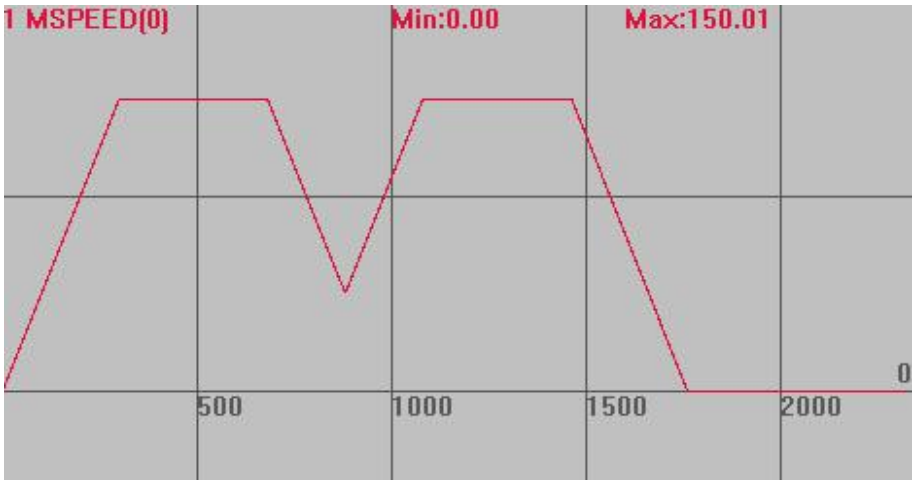
类型	轴参数
描述	**自定义速度的 SP 运动的强制速度，单位为 units/s** 这个参数被带入运动缓冲。 当 FORCE_SPEED 大于 SPEED 时，SPEED 同时也会生效限制住最高速度(140716 以后版本 SPEED 不起作用)。 如果要求进入本段的时候 FORCE_SPEED 已经降为对应的速度，请设置 STARTMOVE_SPEED。
语法	VAR1 = FORCE_SPEED，FORCE_SPEED = expression
适用控制器	通用
例子	BASE(0) DPOS=0 UNITS=100 '脉冲当量 100

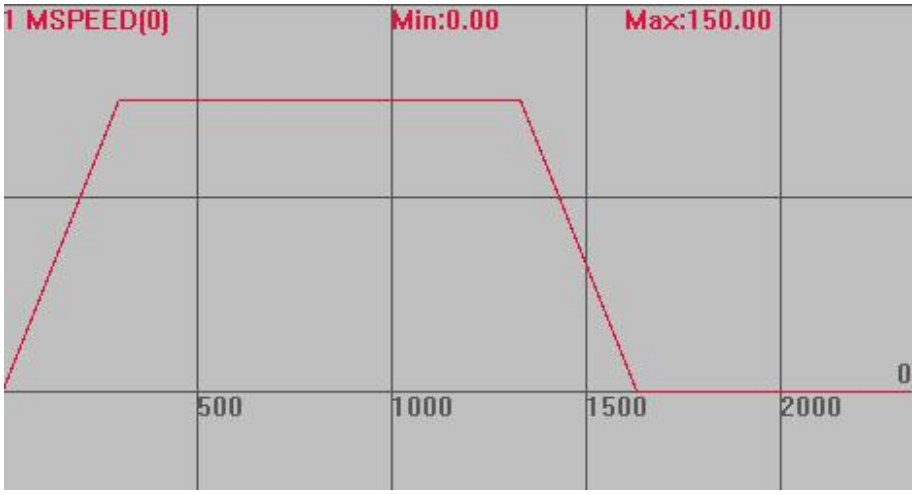
	ACCEL=500 DECEL=500 SPEED = 100 '速度 100units/s **FORCE_SPEED**=150 '自定义速度 150units/s TRIGGER '自动触发示波器 MOVE(100) '不带 SP 使用 SPEED 速度 MOVESP(100) '速度为 150 **FORCE_SPEED**=200 MOVESP(100) '速度为 200 **FORCE_SPEED**=50 MOVESP(100) '速度为 50 速度曲线 MSPEED(0)垂直刻度 100 
相关指令	*SP，

STARTMOVE_SPEED -- SP 运动开始速度

类型	轴参数
描述	**自定义速度的 SP 运动的开始速度，这个参数被带入运动缓冲。** 只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。
语法	VAR1 =STARTMOVE_SPEED，STARTMOVE_SPEED = expression
适用控制器	通用
例子	参考 ENDMOVE_SPEED 例程
相关指令	FORCE_SPEED，*SP，ENDMOVE_SPEED

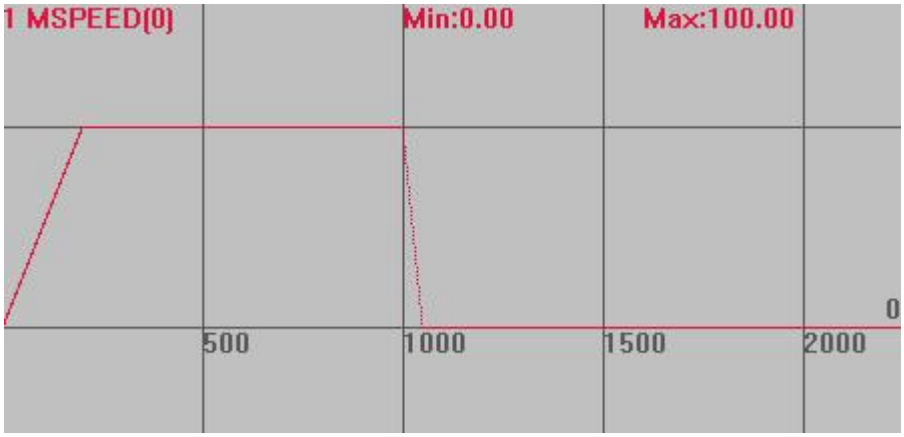
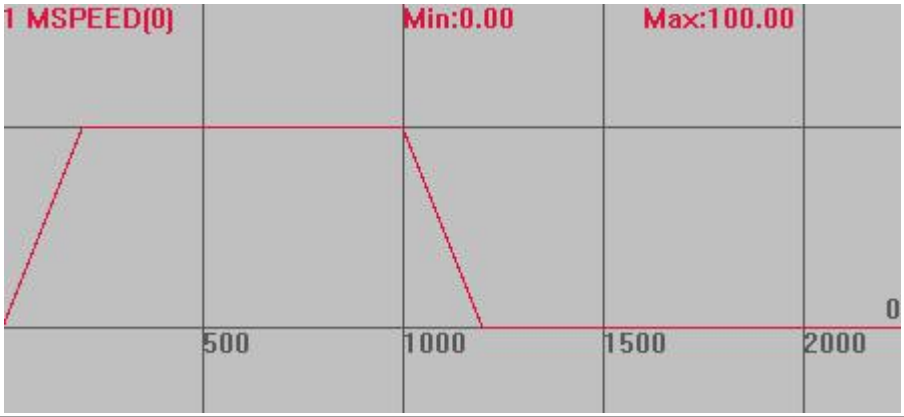
ENDMOVE_SPEED -- SP 运动结束速度

类型	轴参数
描述	**自定义速度的 SP 运动的结束速度，这个参数被带入运动缓冲。** 只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。
语法	VAR1 = ENDMOVE_SPEED，ENDMOVE_SPEED = expression
适用控制器	通用
例子	BASE(0) DPOS=0 UNITS=100 MERGE=1 '开启连续插补 SPEED=100 ACCEL=500 DECEL=500 FORCE_SPEED=150 '限制速度为 150units/s **ENDMOVE_SPEED**=50 '强制结束速度为 50units/s TRIGGER '自动触发示波器 MOVESP(100) MOVESP(100) 速度曲线 MSPEED(0)垂直刻度 100  不使用 ENDMOVE_SPEED 限制速度时

	
相关指令	FORCE_SPEED，*SP，STARTMOVE_SPEED

FASTDEC -- 快减减速度

类型	轴参数
描述	**快减减速度，单位为 units/s/s。** 在 CANCEL，RAPIDSTOP，达到限位或异常停止时自动采用。 当设置为 0 值或小于 DECEL 值时自动为 DECEL。
语法	VAR1 = FASTDEC，FASTDEC= expression
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 UNITS=100 SPEED=100 '速度设为 100 ACCEL=500 DECEL=500 '减速度 500 **FASTDEC**=2000 '快减减速度 2000 TRIGGER '自动触发示波器 VMOVE(1) '持续正转 DELAY (1000) '等待 1s CANCEL(2) '停止 减速曲线 MSPEED(0)垂直刻度 100

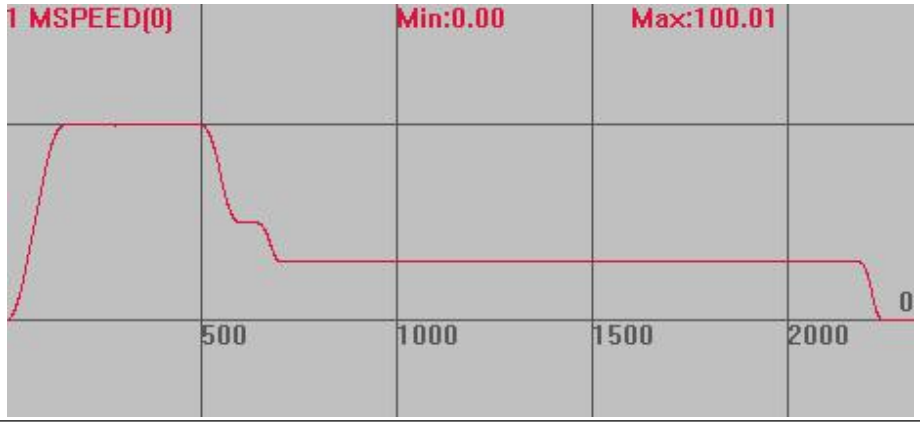
	 FASTDEC=10 时，采用 DECEL 进行减速 
相关指令	DECEL

MSPEED -- 实际反馈速度

类型	轴状态
描述	**轴的测量反馈位置速度，单位是 units/s。**
语法	VAR1 = MSPEED
适用控制器	通用
相关指令	UNITS，VP_SPEED

SPEED_RATIO -- 速度比例

类型	轴参数
描述	**轴速度比例，0-1。** 轴实际运动速度为 SPEED*SPEED_RATIO。 用于运动中根据加减速的平滑速度改变。
语法	SPEED_RATIO(轴号) = value

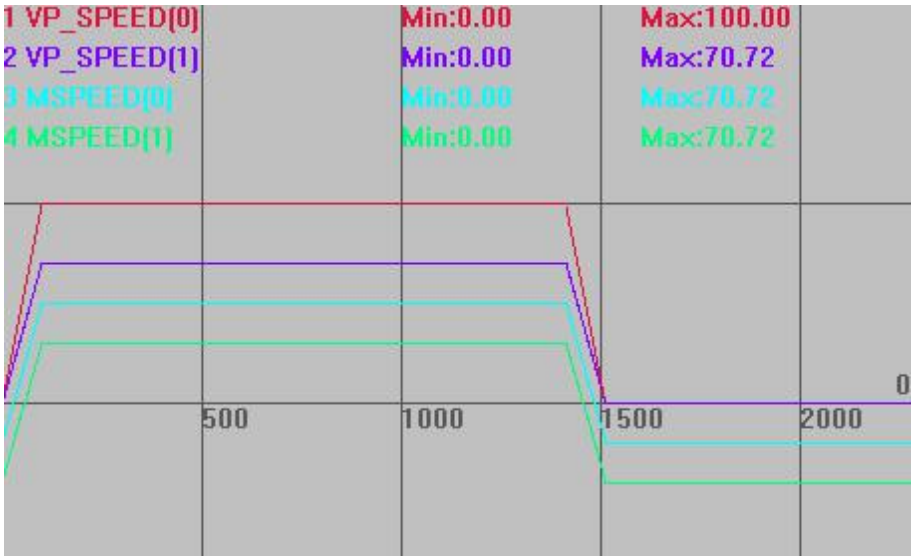
	value：比例，取值范围 0-1 不指定轴号时，默认使用 BASE 指令指定的轴号，插补运动可作用于所有轴，否则只作用于 BASE 的第一个轴。 在线命令不带轴号时，默认对轴 0 起作用。
适用控制器	最新固件支持
例子	RAPIDSTOP(2) WAIT IDLE **SPEED_RATIO** = 1 TRIGGER BASE(0)　　'选择轴 0 DPOS=0 UNITS=100 SPEED = 100 ACCEL = 1000 DECEL = 1000 MERGE = ON SRAMP = 50 MOVE(100) DELAY(500)　　'等待 0.5s **SPEED_RATIO** = 0.5　　'速度降为 50 WAIT UNTIL VP_SPEED < 80 DELAY(100)　　'等待 0.1s **SPEED_RATIO** = 0.3　　'速度降为 30 END 速度曲线 VP_SPEED(0)垂直刻度 100 
相关指令	FORCE_SPEED，SPEED

SRAMP -- 加减速曲线

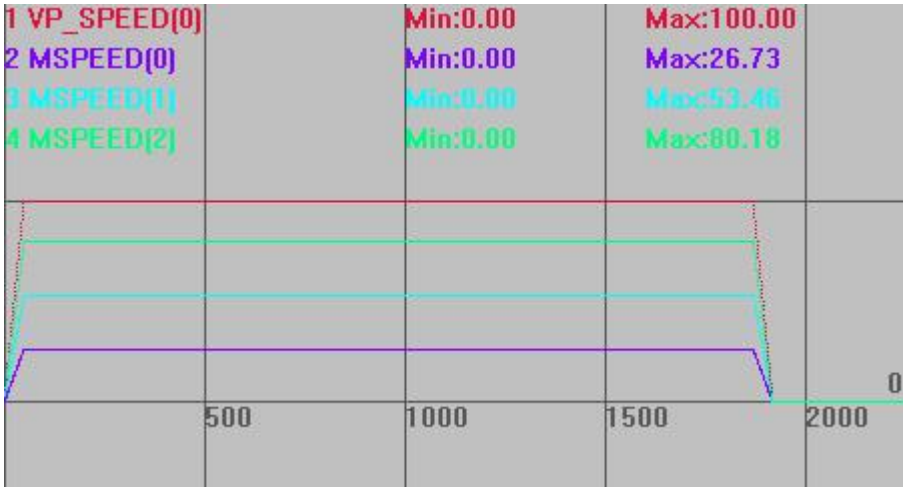
类型	轴参数

描述	**加减速过程 S 曲线设置。**
语法	VAR1 = SRAMP，SRAMP = smoothms smoothms：0-250 毫秒，设置后加减速过程会延长相应的时间
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 UNITS=100 '脉冲当量 100 SPEED=100 '速度 1000units/s ACCEL=1000 '加速度 1000units/s/s DECEL=1000 '加速度 1000units/s/s **SRAMP**=100 'S 曲线时间 100ms TRIGGER '自动触发示波器 MOVE(100) '运动 100units 速度曲线 MSPEED(0)垂直刻度 100  SRAMP=0 时 
相关指令	ACCEL，DECEL

VP_SPEED -- 当前运动速度

类型	轴状态
描述	**返回轴当前运动的速度，单位为 units/s。** 当多轴运动时，主轴返回的是插补运动的速度，不是主轴的分速度。 非主轴返回的是相应的分速度，与 MSPEED 效果一致。 VP_SPEED 在默认情况下是为显示多轴合成速度设计的，是没有负值的，除非把 SYSTEM_ZSET 的 bit0 的值设置为 0，就可以用来显示单轴的命令速度，可正可负。
语法	VAR1 = VP_SPEED
适用控制器	通用
例子	BASE(0,1) DPOS=0 ,0 '坐标清 0 UNITS=100,100 '脉冲当量 SPEED =100,100 '主轴速度 100units/s ACCEL=1000,1000 '加速度 1000units/s/s DECEL=1000,1000 '减速度 1000units/s/s TRIGGER '自动触发示波器 MOVE(100,100) '两轴各运动 100units 速度曲线 主轴 VP_SPEED 是插补合成运动的速度。 非主轴 VP_SPEED 是轴当前分速度，与 MSPEED 一致。 VP_SPEED(0)垂直刻度 100，无偏移 VP_SPEED(1)垂直刻度 100，无偏移 MSPEED(0)垂直刻度 100，偏移-20 MSPEED(1)垂直刻度 100，偏移-40 
相关指令	MSPEED，SPEED

INTERP_FACTOR -- 插补速度计算

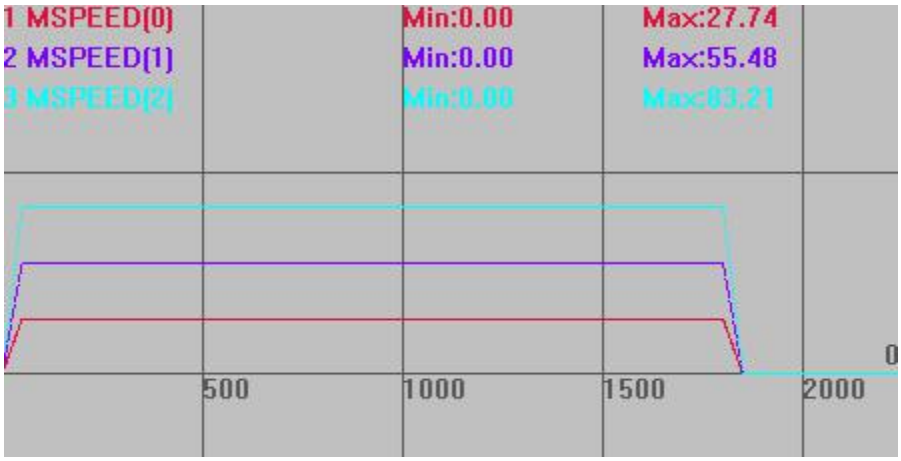
类型	轴参数
描述	**插补时轴是否参与速度计算，缺省参与计算（1）。** 此参数只对直线的任意轴和螺旋的第三个轴起作用。 运动后请及时还原，否则会导致后续运动不正确。 部分轴不参与速度计算时，先按参与计算的轴的插补计算出参与轴的分速度和运动总时间，再把不参与计算的轴的运动距离依次除以总时间，得到不参与计算的各轴的相应速度。见例二。 不能把插补的实际运动的所有轴都设置 0，会导致实际速度无穷大。
语法	INTERP_FACTOR = 0/1 0-不参与计算，1-参与计算
适用控制器	通用
例子	例一 全部参与速度计算 BASE(0,1,2) '选择轴 0 为主轴 DPOS=0,0,0 ATYPE=1,1,1 UNITS=100,100,100 '脉冲当量 100 SPEED=100,100,100 '主轴速度 100units/s ACCEL=1000,1000,1000 DECEL=1000,1000,1000 **INTERP_FACTOR=1,1,1** '3 轴都参与速度计算 TRIGGER '自动触发示波器 MOVE(100,200,300) '各轴分运动距离 此时根据合成运动速度 100 可以计算出各轴分速度，如下图。 VP_SPEED(0)垂直刻度 100 MSPEED(0)垂直刻度 100 MSPEED(1)垂直刻度 100 MSPEED(2)垂直刻度 100 

例二 部分轴不参与速度计算

INTERP_FACTOR=0,1,1 ，轴 0 不参与速度计算。

此时先计算出仅有 2、3 轴插补时的分速度、运动总时间，然后再以轴 0 的运动距离除以总时间，得到轴 0 的速度，如下图。

垂直刻度同上

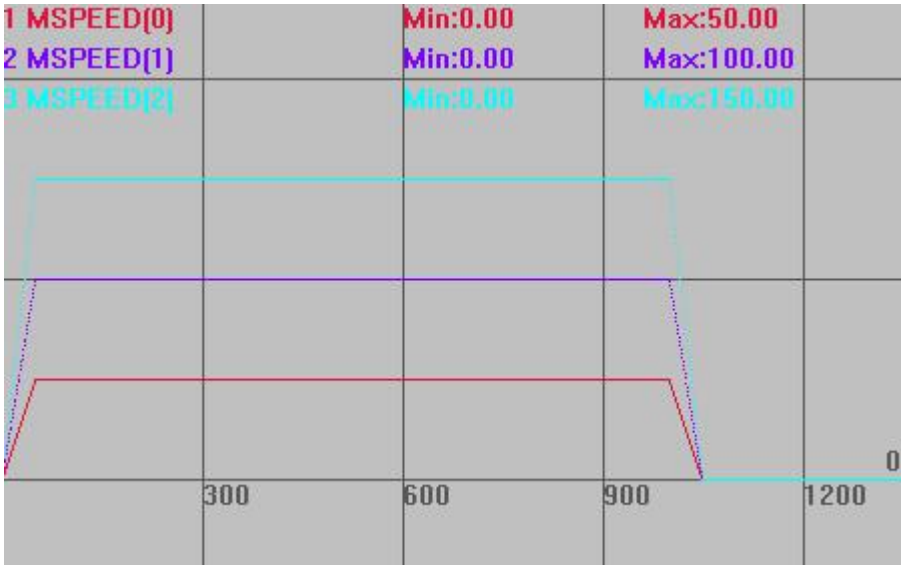


例三 只有一个轴参与计算

INVERT_FACTOR=0,1,0 只有轴 1 进行运动计算。

此时轴 1 的速度就是主轴速度 100，然后计算出运动总时间。其他轴依次用运动距离除以总时间，得到各轴速度，如下图。

垂直刻度同上



相关指令	BASE_MOVE

10.4 轴状态查看指令

MTYPE -- 当前运动类型

<table>
<tr><td>类型</td><td>轴状态</td></tr>
<tr><td>描述</td><td>当前正在进行的运动指令类型。
当插补联动时，对从轴总是返回主轴的运动指令类型。</td></tr>
<tr><td>语法</td><td>VAR1 = MTYPE
<table>
<tr><th>MTYPE</th><th>运动指令类型</th></tr>
<tr><td>0</td><td>IDLE (没有运动)</td></tr>
<tr><td>1</td><td>MOVE</td></tr>
<tr><td>2</td><td>MOVEABS</td></tr>
<tr><td>3</td><td>MHELICAL</td></tr>
<tr><td>4</td><td>MOVECIRC</td></tr>
<tr><td>5</td><td>MOVEMODIFY</td></tr>
<tr><td>6</td><td>MOVESP</td></tr>
<tr><td>7</td><td>MOVEABSSP</td></tr>
<tr><td>8</td><td>MOVECIRCSP</td></tr>
<tr><td>9</td><td>MHELICALSP</td></tr>
<tr><td>10</td><td>FORWARD，VMOVE(1)</td></tr>
<tr><td>11</td><td>REVERSE，VMOVE(-1)</td></tr>
<tr><td>12</td><td>DATUMING</td></tr>
<tr><td>13</td><td>CAM</td></tr>
<tr><td>14</td><td>FWD_JOG</td></tr>
<tr><td>15</td><td>REV_JOG</td></tr>
<tr><td>20</td><td>CAMBOX</td></tr>
<tr><td>21</td><td>CONNECT</td></tr>
<tr><td>22</td><td>MOVELINK</td></tr>
<tr><td>23</td><td>CONNPATH</td></tr>
<tr><td>25</td><td>MOVESLINK</td></tr>
<tr><td>26</td><td>MSPIRAL</td></tr>
<tr><td>27</td><td>MECLIPSE，MECLIPSEABS，MECLIPSESP，MECLIPSEABSSP</td></tr>
<tr><td>28</td><td>MOVE_OP/MOVE_OP2，MOVE_TABLE，MOVE_TASK
MOVE_PARA，MOVE_PWM，MOVE_ASYNMOVE，MOVE_AOUT</td></tr>
<tr><td>29</td><td>MOVE_DELAY，MOVE_WAIT，MOVE_SYNMOVE</td></tr>
<tr><td>31</td><td>MSPHERICAL，MSPHERICALSP</td></tr>
<tr><td>32</td><td>MOVE_PT</td></tr>
<tr><td>33</td><td>CONNFRAME</td></tr>
<tr><td>34</td><td>CONNREFRAME</td></tr>
</table></td></tr>
</table>

适用控制器	通用

例子	WHILE 1 '循环判断
	IF **MTYPE**=0 THEN
	?"没有运动"
	ELSEIF **MTYPE**=1 THEN
	?"直线插补"
	ELSEIF **MTYPE**=4 THEN
	?"圆弧插补"
	...
	ENDIF
	WEND

相关指令	NTYPE，REMAIN_BUFFER

NTYPE -- 下条运动类型

类型	轴状态
描述	**当前正在进行的运动指令后面的第一条指令类型。** 当插补联动时，对从轴总是返回主轴的运动指令类型。
语法	VAR1 = NTYPE
适用控制器	通用

例子	WHILE 1 '循环判断
	IF **NTYPE**=0 THEN
	?"下一条结束运动"
	ELSEIF **NTYPE**=1 THEN
	?"下一条直线插补"
	ELSEIF **NTYPE**=4 THEN
	?"下一条圆弧插补"
	...
	ENDIF
	WEND

相关指令	MTYPE

AXISSTATUS -- 轴状态

类型	轴状态
描述	**查看轴的各种状态。** 按十进制显示数值，按二进制对应位判断状态。
语法	VAR1 = AXISSTATUS

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h

	3	远程驱动器报错	8	8h
	4	正向硬限位	16	10h
	5	反向硬限位	32	20h
	6	找原点中	64	40h
	7	HOLD 速度保持信号输入	128	80h
	8	随动误差超限出错	256	100h
	9	超过正向软限位	512	200h
	10	超过负向软限位	1024	400h
	11	CANCEL 执行中	2048	800h
	12	脉冲频率超过 MAX_SPEED 限制.需要修改降速或修改 MAX_SPEED	4096	1000h
	14	机械手指令坐标错误	16384	4000h
	18	电源异常	262144	40000h
	21	运动中触发特殊运动指令失败	2097152	200000h
	22	告警信号输入	4194304	400000h
	23	轴进入了暂停状态	8388608	800000h
适用控制器	通用			
例子	例一 直接读取位（推荐编程时使用） 碰到了正限位时 VAR = READ_BIT2(4,**AXISSTATUS(0)**) '读取轴 0 是否正限位 PRINT VAR '打印结果，1，发生了正向硬限位 例二 返回数值判断（推荐直接查看时使用） **?AXISSTATUS(1)** '查看轴 1 的状态，结果，48 '48=32+16，同时处于正负限位中，一般是没有反转正负限位开关电平 例三 总线通讯错误 在正确使能电机后： 将通讯接线断开，AXISSTATUS 会显示 4，与远程轴通讯出错。 将编码器线断开，对应轴 AXISSTATUS 会显示 8，远程驱动器报错。			
相关指令	AXIS_STOPPREASON			

IDLE -- 运动状态

类型	轴状态
描述	**轴运动状态判断，只能判断是运动中还是停止。** 运动中，返回 0，运动结束，返回-1。 当轴关联为机械手，CONNFRAME 逆解时，关节轴一直返回 0；CONNREFRAME 正解时，虚拟轴一直返回 0。
语法	VAR1 = IDLE

适用控制器	通用
例子	例一 IF **IDLE**(0) THEN '如果轴 0 停止 BASE(1) MOVE(100) ENDIF 例二 BASE(0,1) MOVE(100,100) BASE(2,3) MOVE(200,200) **WAIT UNTIL IDLE(0) AND IDLE(1) AND IDLE (2) AND IDLE(3)** '等待轴 0，1，2，3 停止
相关指令	LOADED，WAIT IDLE

ADDAX_AXIS -- 叠加轴号

类型	轴状态
描述	**ADDAX 指令所叠加轴的轴号，-1 表示没有叠加。**
语法	VAR1 = ADDAX_AXIS
适用控制器	通用
例子	ADDAX(0) AXIS(1) '轴 0 的运动叠加到轴 1 ?**ADDAX_AXIS**(1) '打印出轴 1 叠加的轴，结果，0 ADDAX(-1) AXIS(1) '取消叠加
相关指令	ADDAX

AXIS_STOPREASON -- 轴停止原因

类型	轴状态
描述	**轴历史停止原因锁存。**
语法	写 0 清除，按位锁存，意义与 AXISSTATUS 相同。 20150731 以上版本支持。
适用控制器	通用

例子	IF **AXIS_STOPREASON** AND (512+1024) THEN
	PRINT "axis el stoped"
	ENDIF

相关指令	AXISSTATUS

LINK_AXIS -- 连接轴号

类型	轴状态
描述	**返回当前连接运动的参考轴号，无连接时返回-1。**
语法	VAR1 = LINKAX
适用控制器	通用
例子	CONNECT(2,1)AXIS(0)　　'轴 0 连接到轴 1 上 ?**LINK_AXIS**(0)　　'打印轴 0 的参考轴号，结果，1
相关指令	CAMBOX，MOVELINK，CONNECT

10.5 运动前瞻指令

CORNER_MODE -- 拐角设置

类型	轴参数
描述	**拐角减速等模式设置。**
语法	CORNER_MODE = mode mode：不同的位代表不同的意义，位可以同时使用。

位	值	描述
0	1	预留
1	2	**自动拐角减速** 按 ACCEL,DECEL 加减速度 此参数是在 MOVE 函数调用前生效 减速角度定义查看 DECEL_ANGLE 和 STOP_ANGLE 指令 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED
2	4	预留
3	8	**自动小圆限速，半径小于设置值时限速，大于限制值时不限速** 此参数在 MOVE 函数调用前修改生效 限制速度按 FORCE_SPEED 限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS 限速半径 FULL_SP_RADIUS 设置
4	16	预留
5	32	**自动倒角设置** 此参数在 MOVE 函数调用前修改生效 此 MOVE 运动自动和前面的 MOVE 运动做倒角处理，倒角半径参考 ZSMOOTH 此倒角针对插补的所有轴，20150701 以后固件支持

适用控制器	通用
例子	为了方便说明每个位的功能，例程都是单独位设置，实际应用时可以多位同时使用。

比如 CONNER_MODE=2+8，打开自动拐角减速和小圆限速

例一 拐角减速

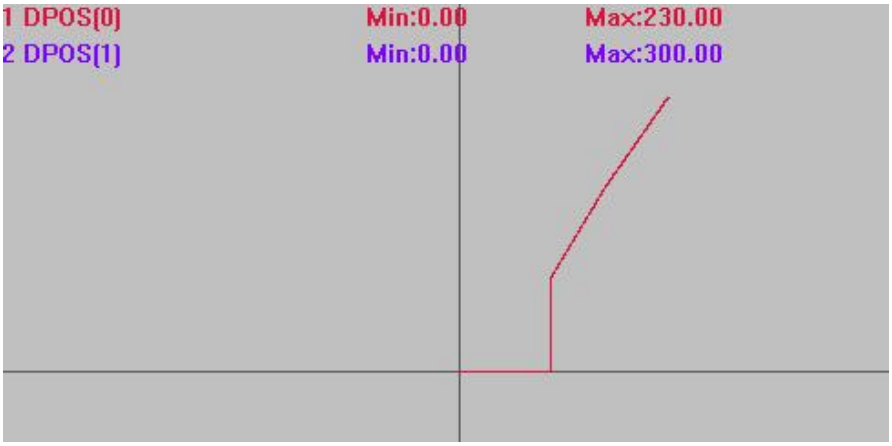
开启拐角减速后，若设置 SP 运动指令的 STARTMOVE_SPEED(SP 指令起始速度)或 ENDMOVE_SPEED(SP 指令结束速度)参数，拐角减速失效，拐角处的速度按以上两个 SP 速度指令设置的大小运动。

```
BASE(0,1)
DPOS=0,0
UNITS=100,100
ACCEL=500, 500                    '设置加速度
DECEL=500,500                     '设置减速度
SPEED=100,100                     '设置速度
MERGE=ON                          '开启连续插补
CORNER_MODE=2                     '启动拐角减速
DECEL_ANGLE = 15 * (PI/180)       '设置开始减速角度
STOP_ANGLE = 45 * (PI/180)        '设置结束减速角度
FORCE_SPEED=100       '等比减速时起作用
TRIGGER               '自动触发示波器
MOVE(100,0)
MOVE(0,100)           '运动角度大于 45°，完全减速
MOVE(60,100)          '运动角度 30.96°处于 15°~45°，等比减速
MOVE(70,100)          '运动角度 4.03°小于 15°，不减速
```

轨迹曲线

DPOS(0)垂直刻度 200

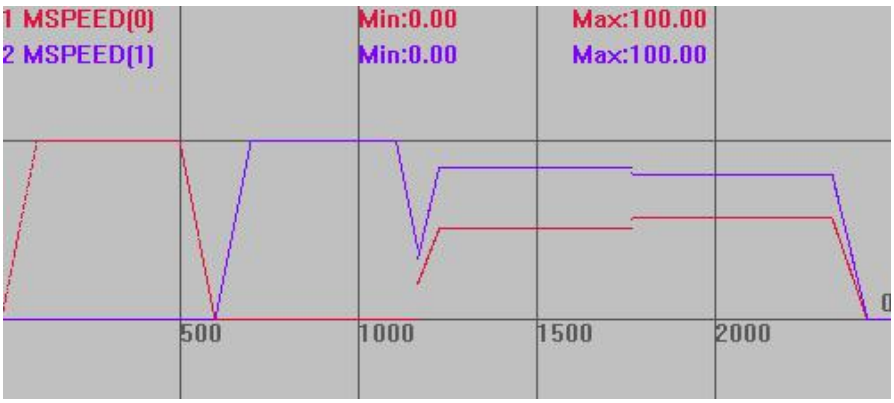
DPOS(1)垂直刻度 200



速度曲线

MSPEED(0)垂直刻度 100

MSPEED(1)垂直刻度 100



使用仿真器查看时曲线可能会有误差，最好连控制器查看。

例二 小圆限速

```
BASE(0,1)
DPOS=0,0
UNITS=100,100
ACCEL=500,500                    '设置加速度
DECEL=500,500                    '设置减速度
SPEED=100,500                    '运行速度
CORNER_MODE=8                    '启动小圆限速
FORCE_SPEED=120                  '小圆限速
FULL_SP_RADIUS=60                '限速半径 60
TRIGGER                          '自动触发示波器
MOVECIRC(200,0,100,0,1)          '半径大于限制值时，不限制速度，按 SPEED 运行此时速度为 100
```

轨迹曲线

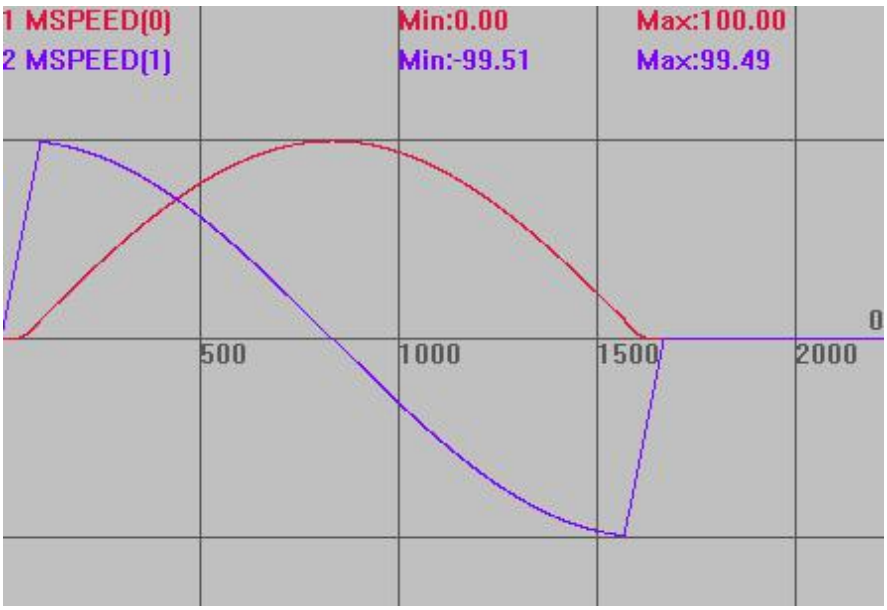
DPOS(0)垂直刻度 100

DPOS(1)垂直刻度 100

速度曲线

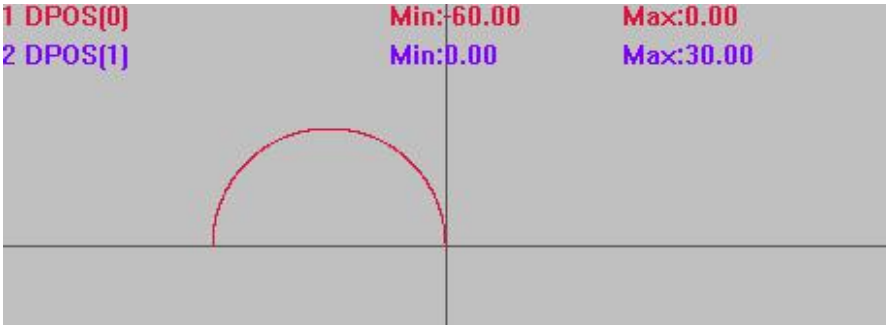
MSPEED(0)垂直刻度 100

MSPEED(1)垂直刻度 100

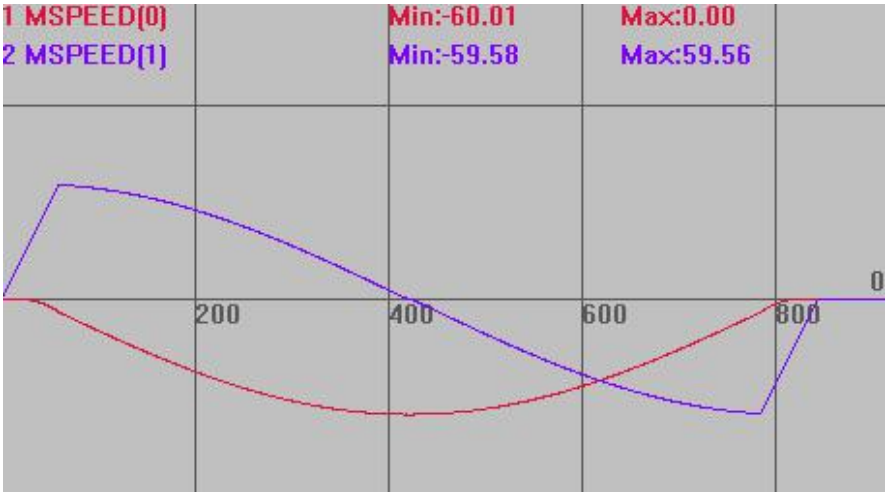


'半径小于限制值时，限速=FORCE_SPEED * 实际半径/FULL_SP_RADIUS
MOVECIRC(-60,0,-30,0,0) '此时速度 60=120*30/60

轨迹曲线
DPOS(0)垂直刻度 100
DPOS(1)垂直刻度 100



速度曲线
MSPEED(0)垂直刻度 100
MSPEED(1)垂直刻度 100



例三 倒角

倒角是可以用于直线、圆弧、螺旋等运动的，例程为了直观显示，只做了直线拐角

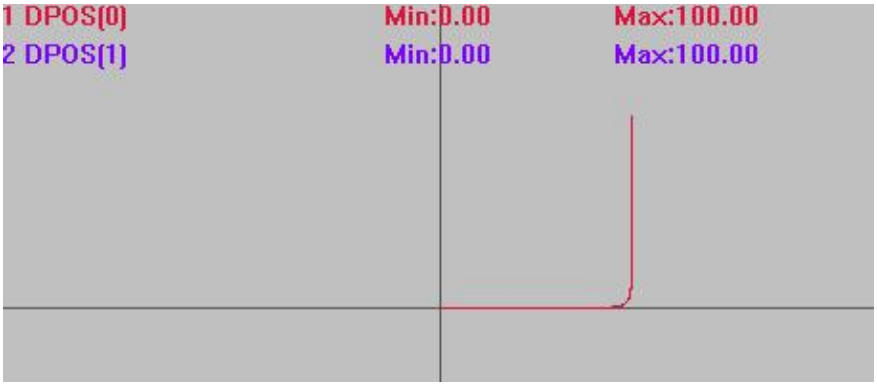
```
BASE(0,1)
DPOS=0,0
ACCEL=500,500                 '设置加速度
DECEL=500,500                 '设置减速度
SPEED=100,100                 '运行速度
CORNER_MODE=32        '启动倒角
ZSMOOTH = 10          '倒角参考半径
TRIGGER               '自动触发示波器
MOVE(100,0)
MOVE(0,100)           '上面两条直线间自动倒角
```

插补轨迹

使用倒角

DPOS(0)垂直刻度 100

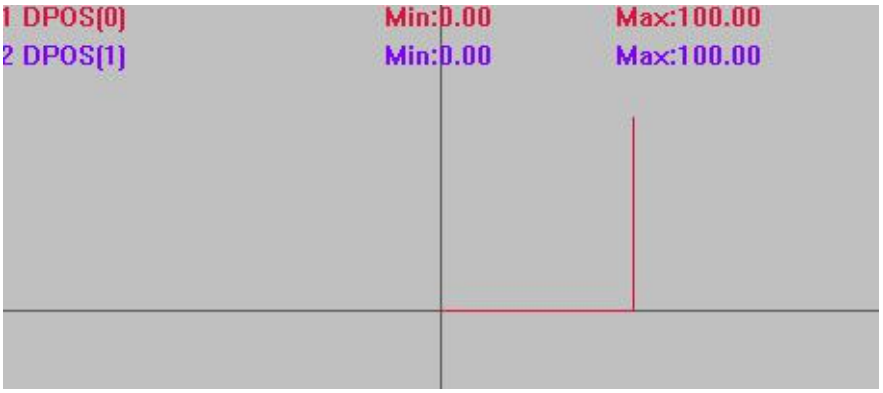
DPOS(1)垂直刻度 100



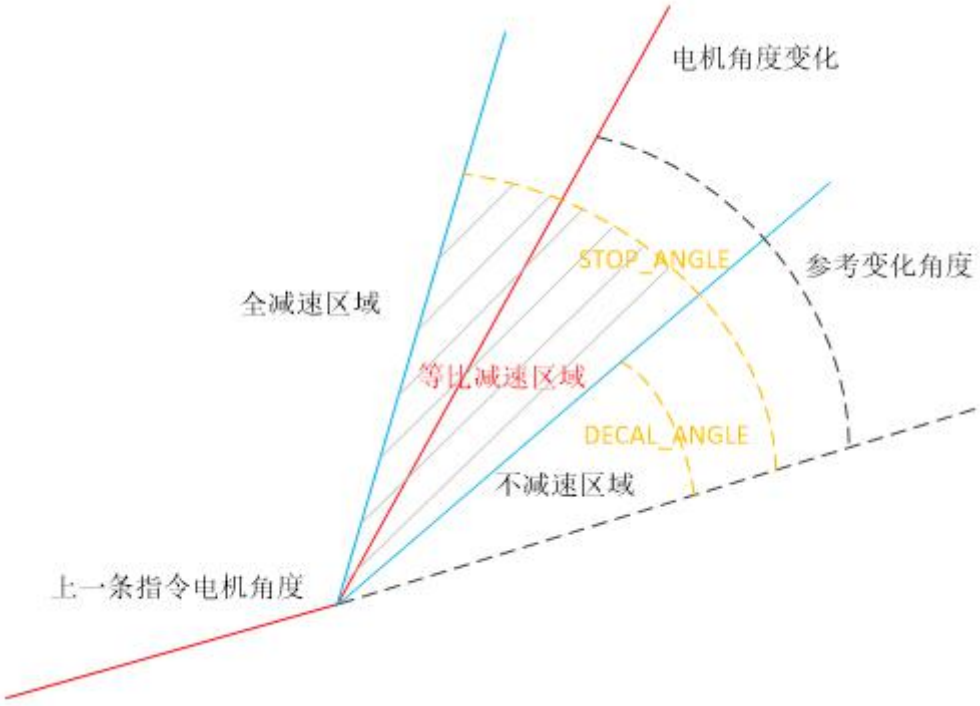
不使用倒角

DPOS(0)垂直刻度 100

DPOS(1)垂直刻度 100

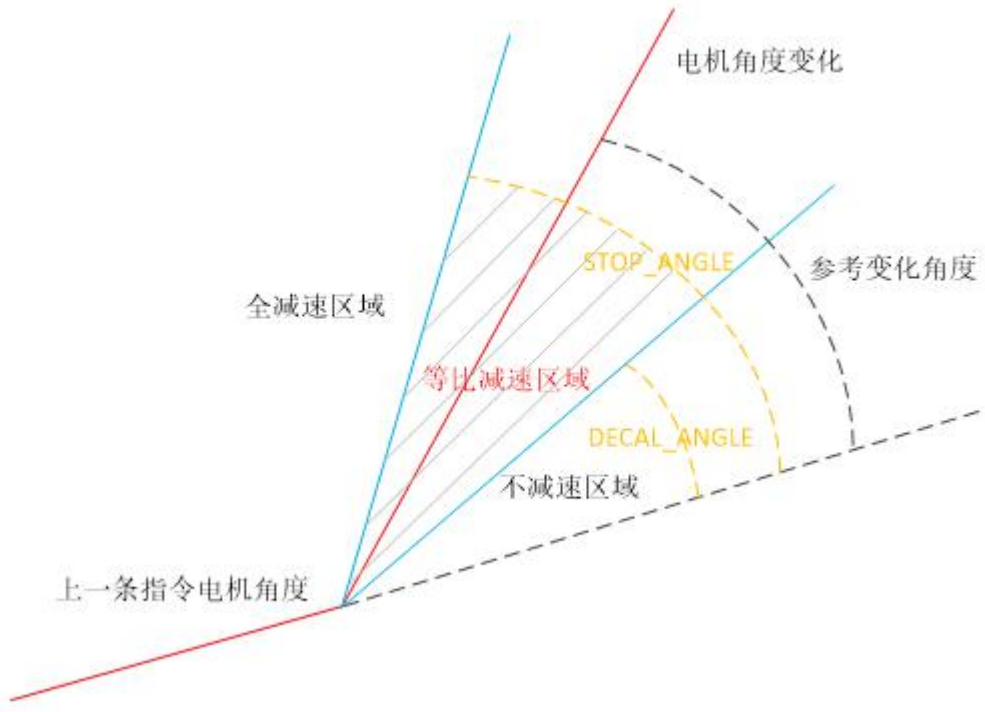
	
相关指令	MERGE，STOP_ANGLE，DECEL_ANGLE，FULL_SP_RADIUS，ZSMOOTH

DECEL_ANGLE -- 拐角减速开始

类型	轴参数
描述	**开始减速的角度，单位是弧度。** 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。 角度转换弧度公式：角度值*(PI/180) 减速角度是指电机的参考角度相对上一条运动的变化值。如下图。 此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。  下一插补运动轨迹处于下方时取绝对值。 直线与圆弧相连时按照圆弧的切线方向计算角度。 与 STOP_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。

语法	VAR1 = DECEL_ANGLE，DECEL_ANGLE = expression
适用控制器	通用
例子	参考 CORNER_MODE 例程一 CORNER_MODE=2 **DECEL_ANGLE** = 25 * (PI/180)　　'设置开始减速拐角，弧度 STOP_ANGLE = 45 * (PI/180)　　'设置结束减速拐角，弧度 FORCE_SPEED=SPEED　　'一定要设置 FORCE_SPEED
相关指令	STOP_ANGLE，

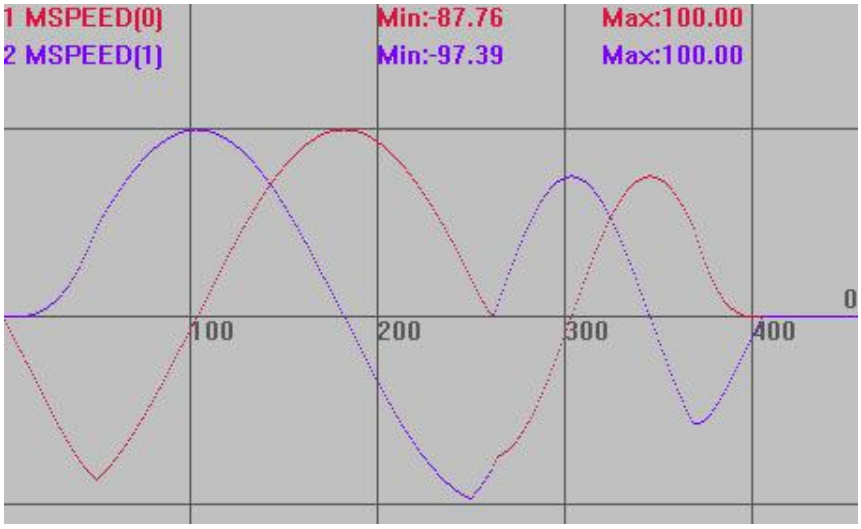
STOP_ANGLE -- 拐角减速结束

类型	轴参数
描述	**停止减速到的角度，单位是弧度。** 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。 角度转换弧度公式：角度值*(PI/180) 减速角度是指电机的参考角度相对上一条运动的变化值。如下图。 此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。  下一插补运动轨迹处于下方时取绝对值。 直线与圆弧相连时按照圆弧的切线方向计算角度。 与 DECEL_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。
语法	VAR1 = STOP_ANGLE，STOP_ANGLE= expression
适用控制器	通用
例子	参考 CORNER_MODE 例程一

	CORNER_MODE=2 DECEL_ANGLE = 25 * (PI/180) **STOP_ANGLE** = 90 * (PI/180) FORCE_SPEED=SPEED '一定要设置 FORCE_SPEED
相关指令	DECEL_ANGLE，CORNER_MODE

FULL_SP_RADIUS -- 限速半径

类型	轴参数
描述	**小圆限速的最大圆弧半径，单位是 units。** 当半径大于 FULL_SP_RADIUS 时，速度是用户程序指定的速度值，当半径小于 FULL_SP_RADIUS 时，控制器会按比例减小速度。 VP_SPEED = FORCE_SPEED*radius/FULL_SP_RADIUS。 设置自动倒角时为倒角的参考半径。
语法	VAR1 = FULL_SP_RADIUS，FULL_SP_RADIUS = expression
适用控制器	通用
例子	BASE(0,1) '选择轴 0 为主轴 DPOS=0,0 '坐标清 0 UNITS=100,100 ATYPE=1,1 '轴类型设置 SPEED=100,100 '主轴速度 100units/s ACCEL=1000,1000 '加速度 1000units/s/s DECEL=1000,1000 '加速度 1000units/s/s FORCE_SPEED=150 '自定义速度 150units/s CORNER_MODE=8 '开启小圆限速 **FULL_SP_RADIUS**=8 '限速半径 8 TRIGGER '自动触发示波器 MOVECIRC(10,10,0,10,1) '圆弧半径 10，不限速，按 speed=100 运行 MOVECIRC(4,4,0,4,1) '圆弧半径 4，限速，速度按 4/8*150=75 运行 速度曲线 MSPEED(0)垂直刻度 100 MSPEED(1)垂直刻度 100

	
相关指令	CORNER_MODE，FORCE_SPEED，SPLIMIT_RADIUS

SPLIMIT_RADIUS -- 限速值

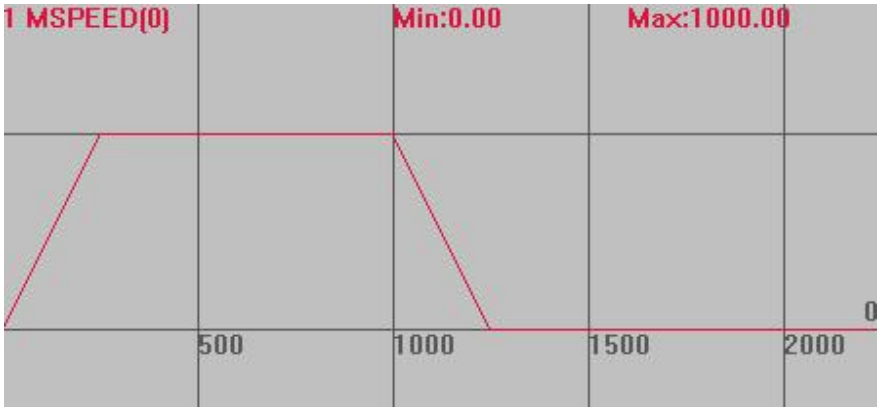
类型	轴参数
描述	**小圆限速的最小速度，单位是 units。** 当小于 LSPEED 时，按 LSPEED 值。
语法	VAR1 = SPLIMIT_RADIUS，SPLIMIT_RADIUS = expression
适用控制器	4 系列控制器 170518 以上固件版本
相关指令	CORNER_MODE，FULL_SP_RADIUS

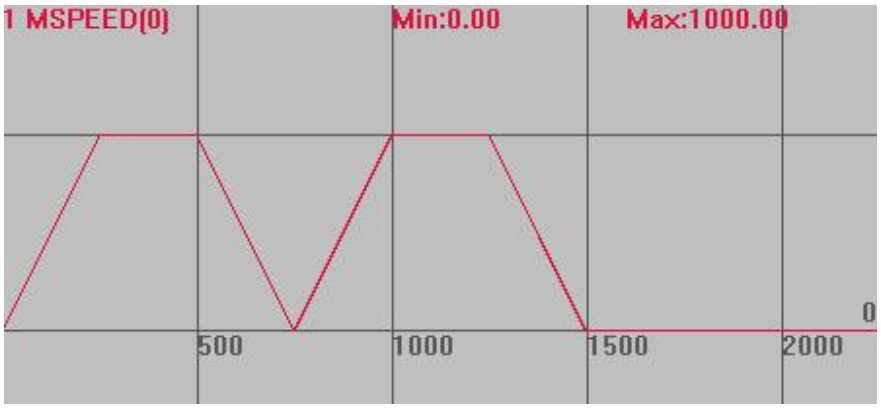
ZSMOOTH -- 倒角半径

类型	轴参数
描述	**参考倒角半径，CONER_MODE 使用。** 根据拐角角度自动计算实际拐角半径大小。超过路径时限制为 50%。 90° 时拐角半径为设置值。
语法	VAR1 = ZSMOOTH，ZSMOOTH = smoothdistance
适用控制器	通用
例子	参考 CONER_MODE 例三
相关指令	CORNER_MODE

MERGE -- 连续插补

类型	轴参数

描述	**前后缓冲的运动连接到一起而不减速，用于连续插补。** 当 MERGE 设置为 ON 时，多段插补间仍减速，可能原因如下： 1. 可能 MERGE 并没有设置成功，可以打印查看。 2. 控制器是点位运动型号，不支持连续插补，可联系厂家。 3. 设置了 CORNER_MODE 拐角减速，打印确认。 4. 使用了带 SP 的运动指令，并设置了 ENDMOVE_SPEED，STARTMOVE_SPEED，此时速度由这两条指令确定。 5. 多条插补间切换了主轴，主轴速度参数改变了。 6. 多条插补间加入了 MOVE_DELAY 延时指令，即使延时写 0，也会导致减速。
语法	MERGE = ON/OFF
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 UNITS=100 SPEED=1000 '速度 1000units ACCEL=1000 DECEL=1000 **MERGE**=ON '打开连续插补 TRIGGER '自动触发示波器 MOVE(1000) '运动 100units MOVE(1000) 速度曲线 MSPEED(0)垂直刻度 100  MERGE=OFF 时 MSPEED(0)垂直刻度 100

	
相关指令	*SP，CORNER_MODE

10.6 运动缓冲指令

LOADED -- 缓冲空

类型	轴状态
描述	**除了当前运动外，没有缓冲的指令了，此时返回 TURE。**
语法	VAR1 = LOAED
适用控制器	通用
相关指令	IDLE，WAIT LOADED

MOVES_BUFFERED -- 当前缓冲数

类型	轴状态
描述	**返回当前被缓冲起来的运动指令个数。** 不要用总缓冲空间数减去 MOVES_BUFFERED 这个参数来判断是否有剩余的缓冲空间，特殊的运动指令可能会占用多个缓冲空间，采用 REMAIN_BUFFER 状态函数来判断更准确。
语法	VAR1 = MOVES_BUFFERED
适用控制器	通用
例子	PRINT **MOVES_BUFFERED** '打印结果，0
相关指令	LOADED，LIMIT_BUFFERED，REMAIN_BUFFER

REMAIN_BUFFER -- 剩余缓冲数

类型	特殊轴状态
描述	**返回可以剩余使用的缓冲个数。**

	此状态比较特殊，本身可以带参数，因此修正轴号时 AXIS 不能省掉。 当返回 0 时表示当前轴的缓冲空间满，此时如果继续调用当前轴的运动指令会阻塞任务直到缓冲有空位。
语法	VAR1 = REMAIN_BUFFER ([mtype]) AXIS(axisnum) MTYPE 缺省为最复杂的运动，如空间圆弧。
适用控制器	通用
例子	DIM movetime '定义变量 movetime = 0 WHILE movetime < 100 '条件循环 IF **REMAIN_BUFFER**(1) > 0 THEN '如果有剩余缓冲，调用直线运动指令 MOVE(10) movetime = movetime +1 ENDIF WEND '调用了 100 次 move(10)
相关指令	LOADED，LIMIT_BUFFERED

MOVE_MARK -- 运动标号

类型	轴参数
描述	**下一条要调用的运动指令的 MARK 标号，这个标号会和运动指令一起写入运动缓冲。** 每调用一条运动指令，MOVE_MARK 会自动加一。 如果要强制指定 MOVE_MARK，需要每次运动前都设定一次。 通过 MOVE_PAUSE 指令可以在 MARK 不同的边界处暂停。
语法	VAR1 = MOVE_MARK，MOVE_MARK = expression
适用控制器	通用
例子	**MOVE_MARK** =1 '设置为标号 1 MOVE(100) **MOVE_MARK** =1 '设置为标号 1 MOVE(100) **MOVE_MARK** =2 '设置为标号 2 MOVE(200) MOVE_PAUSE (2) 'MOVE(200)前暂停
相关指令	MOVE_CURMARK

MOVE_CURMARK -- 当前运动标号

类型	轴状态
描述	**返回当前轴正在运动指令的 MOVE_MARK 标号。**
语法	VAR1 = MOVE_CURMARK
适用控制器	通用

例子	MOVE_MARK =1 MOVE(100) MOVE_MARK =2 MOVE(200) MOVE_MARK =3 MOVE(300) WAIT UNTIL **MOVE_CURMARK** = 2　　'等待 MOVE(200)开始执行的时候，开输出口 1 OP(1,ON)
相关指令	MOVE_MARK

LIMIT_BUFFERED -- 运动缓冲限制

类型	系统参数
描述	**限定运动缓冲个数，不能超过控制器的最大值。**
语法	LIMIT_BUFFERED = value，VAR1 = LIMIT_BUFFERED
适用控制器	通用
例子	在线命令打印： >>?LIMIT_BUFFERED 4096 修改运动缓冲个数限制值： >>LIMIT_BUFFERED=2000 >>?LIMIT_BUFFERED 2000
相关指令	REMAIN_BUFFER，MOVES_BUFFERED

10.7 位置相关指令

DPOS -- 轴指令位置

类型	轴状态
描述	**轴的虚拟坐标位置，或称需求位置。** 写 DPOS 会自动转换为 OFFPOS 偏移，不会移动电机。 以 UNITS 作为单位。
语法	VAR1 = DPOS，DPOS=expression
适用控制器	通用
例子	**DPOS**(0) = 0　　'轴 0 坐标偏移至 0 ?***DPOS**　　'命令行查看当前 DPOS，打印结果，0 0 0 0 0 0 0 0 0 0 0 0
相关指令	MPOS，ENDMOVE，OFFPOS，DEFPOS

MPOS -- 编码器反馈位置

类型	轴状态
描述	**轴的测量反馈位置，单位是 units。** 写 MPOS 会自动转换为 OFFPOS 偏移。
语法	VAR1 = MPOS，MPOS=expression
适用控制器	通用
例子	**MPOS**(0) = 50 'MPOS 偏移至 50 ?***MPOS* '打印结果，50 0 0 0 0 0 0 0 0 0 0 0
相关指令	DPOS，ENDMOVE

DEFPOS -- 位置偏移

类型	设置轴坐标指令
描述	**设置当前轴位置为一个新的绝对位置值，不会对已运行/进入缓冲区的运动产生影响。**
语法	DEFPOS(pos1 [,pos2[, pos3[, pos4.....]]]) pos1：绝对位置，采用 unit 定义单位 pos2：下一个轴绝对位置，采用 unit 定义单位
适用控制器	通用
例子	例一 BASE(0,1) '选择轴 0，轴 1 ATYPE=1,1 UNITS=100,100 '脉冲当量设为 100 DPOS=0,0 '把 DPOS 清 0 MOVE(100,100) '轴 0 和轴 1 运动 100 WAIT IDLE ?DPOS(0),DPOS(1) '此时 DPOS 都为 100 **DEFPOS**(0,10) '设置当前位置 ?DPOS(0),DPOS(1) '此时 DPOS 为 0，10 例二 与 OFFPOS 相对改变不同，DEFPOS 是改变为绝对位置 BASE(0,1) '选择轴 0，轴 1 DPOS=100,100 '设置当前位置为 100,100 ?DPOS(0),DPOS(1) '打印确认，当前位置为 100,100 **DEFPOS**(10,20) '设置当前位置为 10,20 ?DPOS(0),DPOS(1) '当前位置为 10,20 **DEFPOS**(10,20) '多次调用 DEFPOS **DEFPOS**(10,20) ?DPOS(0),DPOS(1) '此时当前位置仍为 10,20

	OFFPOS=10,20 '多次调用 OFFPOS OFFPOS=10,20 ?DPOS(0),DPOS(1) '此时当前位置变为 30,60（10+10+10,20+20+20）
相关指令	DPOS，

OFFPOS -- 偏移位置

类型	轴参数
描述	**相对偏移修改所有的坐标，不会对已运行/进入缓冲区的运动产生影响。** 当修改完成后，OFFPOS 还原为 0。
语法	VAR1 = OFFPOS，OFFPOS = expression
适用控制器	通用
例子	例一 相对偏移位置 BASE(0) MOVEABS(1000) WAIT IDLE **OFFPOS** = -1000 '坐标偏移 1000 PRINT DPOS(0) '打印结果 0 例二 不改变在运行运动 BASE(0) MOVEABS(1000) '运动到绝对位置 1000 **OFFPOS** =500 '位置偏移 500 WAIT IDLE PRINT DPOS(0) '打印当前位置 1500，此时电机仍运动 1000 例三 与 DEFPOS 改变为绝对位置不同，OFFPOS 是相对改变 BASE(0,1) '选择轴 0，轴 1 DPOS=100,100 '设置当前位置为 100,100 ?DPOS(0),DPOS(1) '打印确认 DEFPOS(10,20) '设置当前位置为 10,20 ?DPOS(0),DPOS(1) '当前位置为 10,20 DEFPOS(10,20) '多次调用 DEFPOS DEFPOS(10,20) ?DPOS(0),DPOS(1) '此时当前位置仍为 10,20 **OFFPOS**=10,20 '多次调用 OFFPOS **OFFPOS**=10,20 ?DPOS(0),DPOS(1) '此时当前位置变为 30,60（10+10+10,20+20+20）
相关指令	DPOS，DEFPOS

ENDMOVE -- 当前运动目标位置

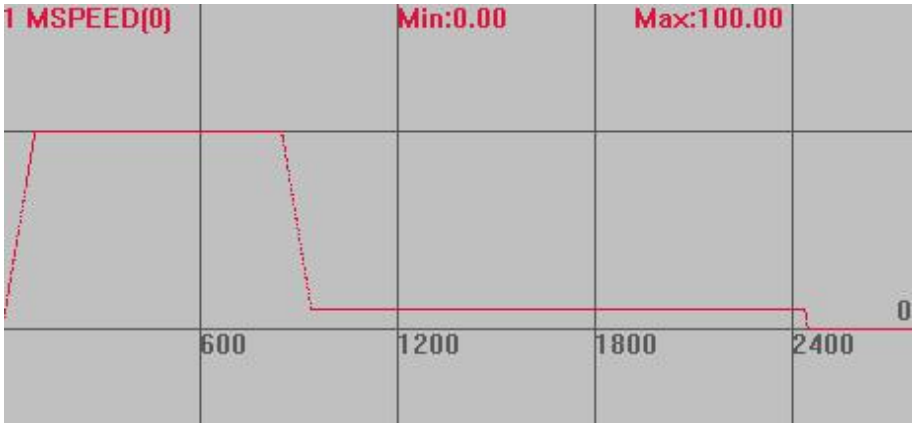
类型	轴状态
描述	**轴当前运动的最终目标绝对位置。** 对 VMOVE，DATUM 等非固定距离的运动，ENDMOVE 并不准确，或时刻在变化。
语法	VAR1 = ENDMOVE
适用控制器	通用
例子	BASE(0) SPEED = 10 '速度 10units/s DPOS = 0 '坐标清 0 MOVE(100) '运动 100units WAIT IDLE PRINT **ENDMOVE**(0) '运行到该条语句时当前运动 move(100)完成 '打印结果，100 MOVE(200) WAIT IDLE PRINT **ENDMOVE**(0) '运行到该条语句时当前运动 move(200)完成，打印结果 300
相关指令	DPOS，MPOS，ENDMOVE_BUFFER

VECTOR_MOVED -- 当前运动距离

类型	轴状态
描述	**返回轴当前运动的距离，units 单位。** 对多轴插补是矢量距离，使用之前最好手动清零。
语法	VAR1 = VECTOR_MOVED VECTOR_MOVED=0
适用控制器	通用
例子	**VECTOR_MOVED**=0 '手动清 0 MOVE(100) WAIT IDLE ? **VECTOR_MOVED** '打印出轴 0 运动距离，结果，100
相关指令	ENDMOVE

REMAIN -- 当前运动剩余距离

类型	轴状态
描述	**返回轴当前运动还未完成的距离，units 单位。**
语法	VAR1 = REMAIN
适用控制器	通用
例子	BASE(0) '选择轴 0

	DPOS=0 UNITS=100 SPEED=100 '速度 100units/s ACCEL=1000 '加速度 1000units/s DECEL=1000 TRIGGER '自动触发示波器 MOVE(100) '运动 100units WAIT UNTIL **REMAIN**<20 '等待剩余距离小于 20 SPEED=10 '修改速度 速度曲线 MSPEED(0)垂直刻度 100 
相关指令	VECTOR_BUFFERED

VECTOR_BUFFERED --缓冲剩余距离

类型	轴状态
描述	**返回轴当前运动和缓冲运动还未完成的距离，units 单位。** 对多轴插补是矢量距离。
语法	VAR1 = VECTOR_BUFFERED
适用控制器	通用
例子	BASE(0) '选择轴 0 UNITS=100 '脉冲当量 100 SPEED=100 '速度 100units/s ACCEL=1000 '加速度 1000units/s/s MOVE(100) '当前运动 100units MOVE(300) '缓冲运动 300units MOVE(-1000) '缓冲运动-1000units ?**VECTOR_BUFFERED** '返回剩余运动距离，结果，1400
相关指令	REMAIN

ENDMOVE_BUFFER -- 缓冲最终位置

类型	轴状态
描述	**轴当前和缓冲的所有运动的最终目标位置。** 可用来进行绝对和相对位置的转换，见例二。 对 VMOVE，DATUM 等非固定距离的运动，ENDMOVE_BUFFER 并不准确，或时刻在变化。 使用了循环坐标指令 REP_OPTION 后，ENDMOVE_BUFFER 根据 REP_OPTION 模式按 REP_DIST 设置值依次递减，即最小精度是 REP_DIST（模式 1）或 2 倍 REP_DIST（模式 0），见例三。
语法	VAR1 = ENDMOVE_BUFFER
适用控制器	通用
例子	例一 BASE(0) SPEED = 10 DPOS = 0 MOVE(100) MOVE(200) PRINT **ENDMOVE_BUFFER**(0) '直接打印出最终运动完后的绝对坐标 '打印结果，300 例二 相对绝对转换 使用 ENDMOVE_BUFFER 与相对运动指令可实现绝对运动的功能，常用于 MSPHERICAL 等只有相对模式的指令。 BASE(0) UNITS=100 SPEED=100 ACCEL=1000 DPOS=0 WHILE 1 MOVE(100- **ENDMOVE_BUFFER**(0)) '运动到 100 位置，不会继续运动 WEND 例三 循环坐标时的返回值 BASE(0) UNITS=100 SPEED=100 ACCEL=1000 DPOS=0 TRIGGER MOVE(1000) REP_DIST=100 '设置坐标循环范围

	REP_OPTION=1 '0~100 循环 WHILE 1 ?**ENDMOVE_BUFFER**(0) '此时打印出 1000,900,800...,100,0，打印的最小精度是 100 WEND
相关指令	DPOS，MPOS，ENDMOVE

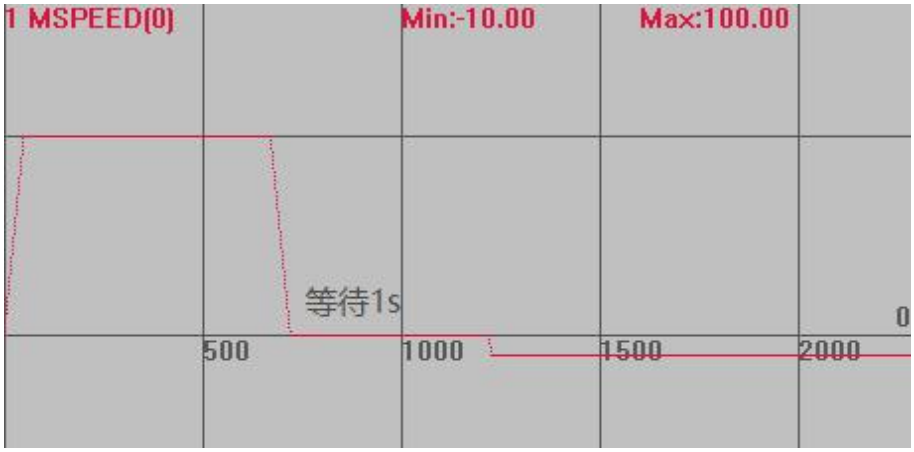
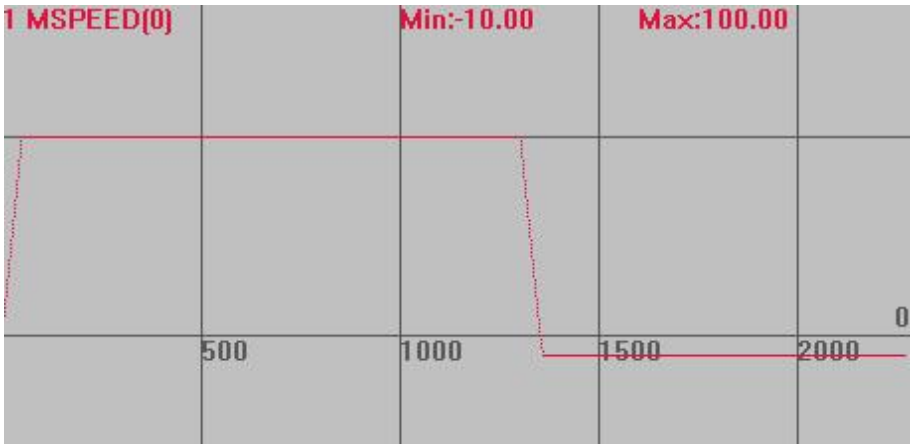
10.8 原点回零指令

DATUM_IN -- 映射原点输入

类型	轴参数
描述	**通用输入口设置为原点开关信号，-1 无效。** 设置了原点开关后，ZMC 控制器输入 OFF 时认为有信号输入，要相反效果可以用 INVERT_IN 反转电平。(ECI 系列控制器除外)。
语法	VAR1 = DATUM_IN，DATUM_IN = expression
适用控制器	通用
例子	BASE(0,1,2,3) **DATUM_IN** =6 ,7,8,9 '将轴 0，1，2，3 原点输入对应到输入口 6，7，8，9 INVERT_IN(6,ON) '把原点信号反转 INVERT_IN(7,ON) INVERT_IN(8,ON) INVERT_IN(9,ON)
相关指令	DATUM，FWD_IN，REV_IN，INVERT_IN

HOMEWAIT -- 回零反找延时

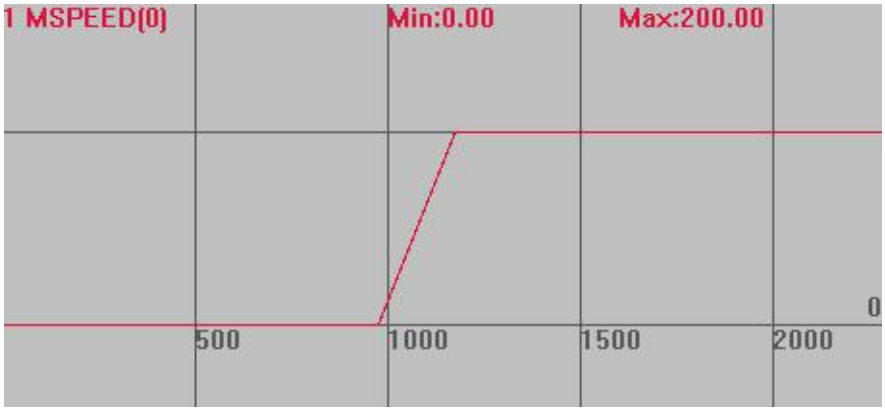
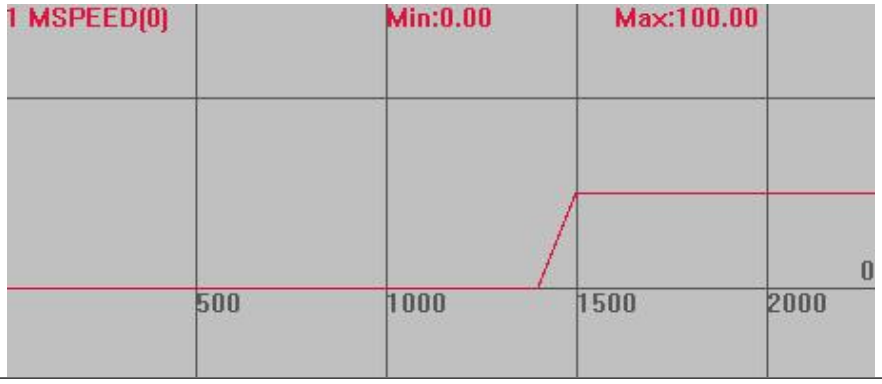
类型	轴参数
描述	**此参数设置等待的时间，单位是毫秒。** 对脉冲方式的伺服驱动器，回原点运动时，当反找时要等待一定时间。 控制器默认值为 2ms。
语法	VAR1 = HOMEWAIT，HOMEWAIT = expression
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 '坐标清 0 UNITS=100 ATYPE=1 SPEED=100 '找原点速度 100units/s ACCEL=1000,1000 '加速度 1000units/s/s DECEL=1000,1000 '加速度 1000units/s/s CREEP=10 '爬行速度 10units/s

	DATUM_IN=0 '原点信号设为输入 IN0 INVERT_IN(0,ON) '反转信号 **HOMEWAIT**=1000 '设置反找等待 1s TRIGGER '自动触发示波器 DATUM(3) '正向找原点 速度曲线 碰到 IN0 口时，会先等待 1s，然后在反向爬行 MSPEED(0)垂直刻度 100  HOMEWAIT=2 时，几乎不停止 MSPEED(0)垂直刻度 100 
相关指令	DATUM

10.9 JOG 运动指令

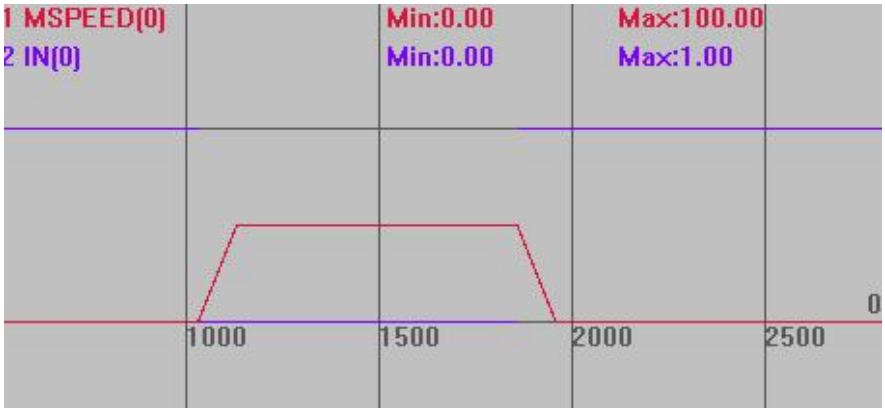
FAST_JOG -- 映射点动输入

类型	轴参数
描述	**快速点动的输入的编号，-1 为无效。**

	如果设置快速点动输入，速度由 SPEED 参数给出。如果没有输入设置，速度由 JOGSPEED 参数给出。见例程。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。
语法	VAR1 = FAST_JOG，FAST_JOG = expression
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 '坐标清 0 UNITS=100 ATYPE=1 SPEED=100 '设置速度为 100 units/s ACCEL=500 '加速度为 500units/s/s JOGSPEED=200 '点动速度设为 200units/s **FAST_JOG**(0)=0 '轴 0 的快速输入设为 IN0 口 FWD_JOG(0)=1 '正向点动开关设为 IN1 口 INVERT_IN(0,ON) '反转电平 INVERT_IN(1,ON) TRIGGER '自动触发示波器 速度曲线 IN0 无输入时，按下 IN1 并保持，轴速度为 JOGSPEED=200 MSPEED(0)垂直刻度 200  IN0 有输入时，按下 IN1，轴速度为 SPEED=100 MSPEED(0)垂直刻度 200 

相关指令	REV_JOG，FWD_JOG，SPEED，JOGSPEED

FWD_JOG -- 映射正向 JOG 输入

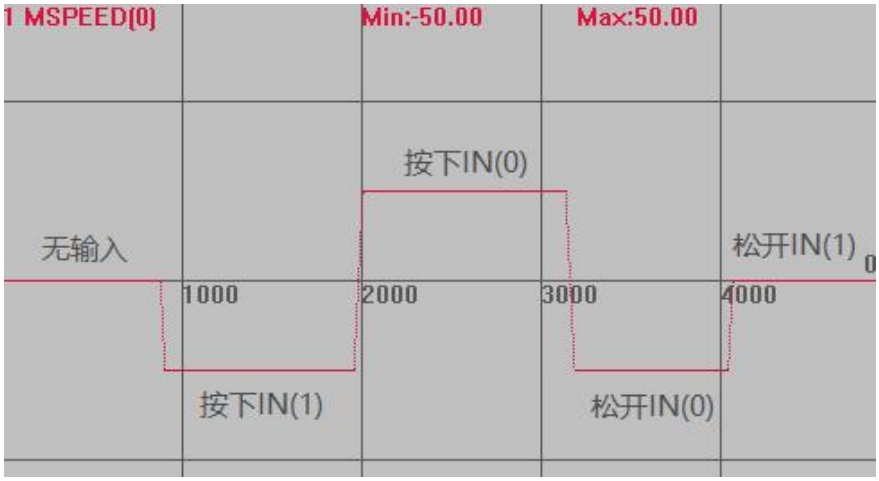
类型	轴参数
描述	**正向 JOG 输入对应的输入口编号，-1 无效。** 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。 当有信号输入时，对应轴按照 JOGSPEED 速度正向运动。
语法	VAR1 = FWD_JOG，FWD_JOG= expression
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 '坐标清 0 UNITS=100 ATYPE=1 SPEED=100 '速度 100 ACCEL=500 '加速度为 500units/s/s DECEL=500 JOGSPEED=50 'JOG 速度 50 **FWD_JOG**=0 '输入 IN0 作为正向 JOG 开关 INVERT_IN(0,ON) '反转信号 TRIGGER '自动触发示波器 输入 0 口有信号输入时，轴 0 正向运行，速度为 50。 输入 0 口取消信号输入时，轴 0 停止。 速度曲线 MSPEED(0)垂直刻度 100 
相关指令	REV_JOG，JOGSPEED，FAST_JOG

REV_JOG -- 映射负向 JOG 输入

类型	轴参数
描述	**负向 JOG 输入对应的输入口编号，-1 无效。** 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。 当有信号输入时，对应轴按照 JOGSPEED 速度负向运动。 当 REV_JOG 和 FWD_JOG 同时有信号输入时，轴按照 FWD_JOG 运行。
语法	VAR1 = REV_JOG，REV_JOG= expression
适用控制器	通用
例子	参考 FWD_JOG 例子
相关指令	FWD_JOG，JOGSPEED，FAST_JOG

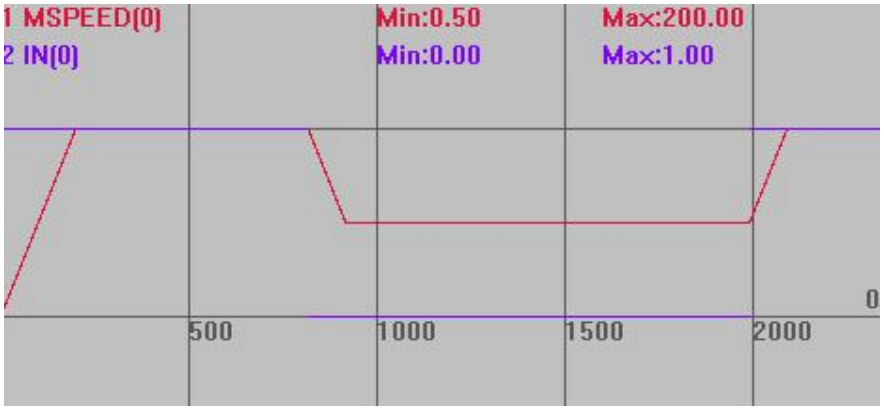
JOGSPEED -- JOG 速度

类型	轴参数
描述	**JOG 时的速度，单位为 units/s。** 当 REV_JOG/FWD_JOG 被设置，对应输入点按下时，并保持当前输入状态，电机将以 JOGSPEED 慢速运动，输入点松开运动停止。
语法	JOGSPEED= value，VAR1=JOGSPEED
适用控制器	通用
例子	例一 BASE(0) '选择轴 0 DPOS=0 '坐标清 0 UNITS=100 '脉冲当量 SPEED =100 '主轴速度 100units/s ACCEL=1000 '加速度 1000units/s/s DECEL=1000 '减速度 1000units/s/s TRIGGER '自动触发示波器 **JOGSPEED**=50 'JOG 速度 50 FWD_JOG=0 '输入 IN0 作为正向 JOG 开关 REV_JOG=1 '输入 IN1 作为负向 JOG 开关 INVERT_IN(0,ON) '反转信号 INVERT_IN(1,ON) 输入 0 口有信号输入时，轴 0 正向运行，速度为 50。 输入 1 口有信号输入时，轴 0 负向运行，速度为 50。 同时有信号输入时，轴 0 正向运行。 速度曲线

	MSPEED(0)垂直刻度 100 
相关指令	REV_JOG，FWD_JOG，FAST_JOG

FHOLD_IN -- 映射保持输入

类型	轴参数
描述	**保持输入对应的输入点编号，-1 无效。** 如果有输入信号，运动轴的速度由 FHSPEED 参数设置。当前运动没有被取消。当取消输入，过程中的运动速度返回程序速度。见例程。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。 此参数只适用速度控制方式(带 SP 后缀指令)。如果运动不是速度控制（CAMBOX，CONNECT，MOVELINK），不会起作用。
语法	VAR1 = FHOLD_IN，FHOLD_IN = expression
适用控制器	通用
例子	BASE(0) '选择轴 0 DPOS=0 '坐标清 0 UNITS=100 ATYPE=1 ACCEL=500 '加速度为 500units/s/s DECEL=500 FORCE_SPEED=200 '设置速度为 200 units/s FHSPEED=200 '保持速度设为 100units/s **FHOLD_IN**(0)=0 '轴 0 的保持输入设为 IN0 口 INVERT_IN(0,ON) '反转电平 TRIGGER '自动触发示波器 MOVESP(10000) '运动 当 IN0 无输入时，轴以 FORCE_SPEED=200 的速度运动

	当 IN0 有输入时，轴以 FHSPEED=100 的速度运动，取消输入后，速度变回 200 速度曲线 MSPEED(0)垂直刻度 200 
相关指令	FHSPEED

FHSPEED -- 保持速度

类型	轴参数
描述	**轴保持速度，在 FHOLD_IN 被按下保持时的速度，单位为 units/s。** 对应的输入处于保持状态时才能一直以此速度运动。
语法	VAR1 = FHSPEED，FHSPEED = expression
适用控制器	通用
例子	见 FHOLD_IN 例程
相关指令	FHOLD_IN，SPEED

10.10 编码器相关指令

ENCODER -- 编码器原始值

类型	轴状态
描述	**编码器硬件寄存器原始值。** 内部参数，只有配置为需要使用编码器的 ATYPE 时才可以读取。 驱动器有多圈绝对值编码器时，读取的就是多圈值。
语法	VAR1 = ENCODER
适用控制器	通用
例子	?***ENCODER** '打印各轴编码器值，打印结果，0 0 0 0 0 0 0 0 0 0 0 0
相关指令	MPOS，ENCODER_RATIO

ENCODER_STATUS -- 编码器状态

类型	轴状态
描述	**编码器的 EA EB EZ 的状态。**
语法	VAR1 = ENCODER_STATUS 位 / 值 / 描述 0 / 1 / EA 状态 1 / 2 / EB 状态 2 / 4 / EZ 状态
适用控制器	通用
例子	?***ENCODER_STATUS**　　'打印全部轴的编码器状态
相关指令	ATYPE，MPOS

ENCODER_FILTER -- 编码器滤波

类型	轴参数
描述	**用于使皮带编码器的速度均匀，缺省值为 1，设置范围 0.001~1。** 5 系控制器支持，4 系列固件 170706 以上固件版本支持。
语法	ENCODER_FILTER = VALUE
适用控制器	通用
相关指令	ENCODER_RATIO

PP_STEP -- 编码器内部比例

类型	轴参数
描述	**编码器输入内部会乘以这个比例。** 这个参数效果与 ENCODER_RATIO 叠加，缺省 1。
语法	PP_STEP = VALUE
适用控制器	通用
相关指令	ENCODER_RATIO

10.11 锁存相关指令

REGIST -- 锁存

类型	位置锁存指令

<table>
<tr><td>描述</td><td>
REGIST 指令用来锁存轴的测量反馈位置。

支持编码器轴锁存，4 系列及以上控制器最新固件支持虚拟轴、脉冲轴锁存。
EtherCAT 支持驱动器锁存，此时使用驱动器 IO 点实现锁存，具体模式查看指令语法。
Rtex 只支持控制器锁存。
4 系列及以上控制器支持 4 锁存通道。
锁存输入口 432：2 个，432N：4 个，412：8 个。
支持 EtherCAT 驱动器锁存与控制器锁存同时使用，需要有 4 锁存通道功能。
4 个通道指 MARK, MARKB, MARKC, MARKD，通过 REG_INPUTS 指定锁存输入口对应的锁存通道。

当锁存产生时，轴状态 MARK 会被设置为 ON，同时锁存到的位置会被存储在参数 REG_POS 内。
每个轴有输入信号 R0,R1, EZ 信号可以使用锁存功能。当使用两个信号锁存时，第二个信号锁存使用 MARKB 和 REG_POSB。
R0，R1 输入一般对应到输入口 0 和 1，详细请查看控制器的硬件手册中通用输入章节。
</td></tr>
<tr><td>语法</td><td>
语法一
REGIST(mode)
mode：锁存方式
上升下降沿是以控制器内部状态而言。如果设置成上升沿触发，则锁存会在外部输入口由导通状态进入截止状态的一瞬间触发；如果设置成下降沿触发，则锁存会在外部输入口由截止状态进入导通状态的一瞬间触发；具体要设置成上升沿还是下降沿触发，要根据外部输入口信号跳变的实际需求。
脉冲轴类型一般采用 R0，R1，Z 脉冲这三种锁存；总线轴类型采 R2，R3 锁存。
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>1</td><td>当 Z 脉冲上升沿时的绝对位置送到 REG_POS</td></tr>
<tr><td>2</td><td>当 Z 脉冲下降沿时的绝对位置送到 REG_POS</td></tr>
<tr><td>3</td><td>当输入信号 R0 上升沿的绝对位置送到 REG_POS</td></tr>
<tr><td>4</td><td>当输入信号 R0 下降沿的绝对位置送到 REG_POS</td></tr>
<tr><td>6</td><td>输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>7</td><td>输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>8</td><td>输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>9</td><td>输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>10</td><td>输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>11</td><td>输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>12</td><td>输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB</td></tr>
<tr><td>13</td><td>输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB</td></tr>
</table>
</td></tr>
</table>

14	输入信号 R1 上升沿时的绝对位置送到 REG_POSB(14 以后 150804 以后版本支持，每个锁存通道独立，支持 4 通道锁存)
15	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
16	Z 信号上升沿时的绝对位置送到 REG_POSB
17	Z 信号下降沿时的绝对位置送到 REG_POSB
18	输入信号 R2 上升沿时的绝对位置送到 REG_POSC
19	输入信号 R2 下降沿时的绝对位置送到 REG_POSC
20	输入信号 R3 上升沿时的绝对位置送到 REG_POSD
21	输入信号 R3 下降沿时的绝对位置送到 REG_POSD

语法二

REGIST(100+mode, tableindex, numes)

mode：锁存方式

tableindex：连续锁存的内容存储的 table 位置，第一个 table 元素存储锁存的个数，后面存储锁存的坐标，最多保存个数= numes-1，溢出时循环写入

numes：占用的 table 个数

通过把模式加 100 来支持连续锁存，锁存结果存储到 TABLE 里面。
分别对两个通道进行连续锁存，可以实现上下边沿的连续锁存。
ECI：20150829 以上固件支持。
4 系列控制器：20170523 以上固件支持。

100+mode：只能使用单一通道的 mode，加 100 表示使用连续锁存

值	描述
1	当 Z 脉冲上升沿时的绝对位置送到 REG_POS
2	当 Z 脉冲下降沿时的绝对位置送到 REG_POS
3	当输入信号 R0 上升沿的绝对位置送到 REG_POS
4	当输入信号 R0 下降沿的绝对位置送到 REG_POS
14	输入信号 R1 上升沿时的绝对位置送到 REG_POSB
15	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
16	Z 信号上升沿时的绝对位置送到 REG_POSB
17	Z 信号下降沿时的绝对位置送到 REG_POSB
23	当输入信号 R0 上升沿的绝对位置送到 REG_POSB
24	当输入信号 R0 下降沿的绝对位置送到 REG_POSB
33	当输入信号 R0 上升沿的绝对位置送到 REG_POS,下一次切换下降沿，轮流切换。
34	当输入信号 R0 下降沿的绝对位置送到 REG_POS，下一次切换上升沿，轮流切换。
35	当输入信号 R1 上升沿的绝对位置送到 REG_POSB,下一次切换下降沿，轮流切换。下一次切换下降沿，轮流切换。
36	当输入信号 R1 下降沿的绝对位置送到 REG_POSB，下一次切换上升沿，轮流切换。

适用控制器	有锁存 IN 口。

例子

例一　锁存脉冲轴 0 的输入信号 R0 上跳沿时的位置，并打印。

```
BASE(0)
REG_INPUTS=0    '将 R0-R3 都对应输入口 0
ATYPE=1         '脉冲轴
REGIST(3)       '选择 R0 锁存模式
WAIT UNTIL MARK     '等待锁存触发
PRINT REG_POS       '打印锁存位置
```

例二　锁存编码器轴 0 的输入信号 R1 上跳沿时的位置，并打印。

```
BASE(0)
REG_INPUTS=0    '将 R0-R3 都对应输入口 0
ATYPE=3         '编码器轴
REGIST(14)      '选择 R1 锁存模式
WAIT UNTIL MARKB    '等待锁存触发
PRINT REG_POSB      '打印锁存位置
```

例三　锁存 EtherCAT 总线轴 0 的输入信号 R2 上跳沿时的位置，并打印。

```
'先进行总线初始化使能后再执行锁存，总线轴使用 R2R3 锁存
BASE(0)
REG_INPUTS=0     '将 R0-R3 都对应输入口 0
ATYPE=65         'EtherCAT 总线轴
REGIST(18)       '选择 R2 锁存模式
WAIT UNTIL MARKC     '等待锁存触发
PRINT REG_POSC       '打印锁存位置
```

例四　PC 交互位置锁存，一般用于运动抓拍，通过锁存的位置得知抓拍时的实际位置。

```
GLOBAL g_start
GLOBAL g_posx, g_posy
WHILE 1
    WAIT UNTIL g_start=1    '等待 PC 发出启动
    REGIST(4) AXIS(0)    '输入 0 锁存，24V 变 0V 的时刻
    REGIST(4) AXIS(1)
    WAIT UNTIL MARK(0) AND MARK(1)
    g_start=0
    g_posx=REG_POS(0)
    g_posy=REG_POS(1)
    PRINT g_posx, g_posy
WEND
```

例五　100+mode 连续锁存

```
DIM num
num=1
BASE(6)
ATYPE=6
REGIST(100+4,0,100)      '自动循环，不需要再写入到 while 循环中，table(0)保存锁存
```

	次数，table(1-100)存储每次锁存的数据超过 99 次后，table(0)清 0，重新从 table(1)记录数据
	WHILE 1
	WAIT UNTIL MARK
	?reg_pos,TABLE(num),TABLE(0) '打印
	IF num=100 THEN
	num=1
	ELSE
	num=num+1
	ENDIF
	WA 1 '延时 1ms，防抖
	WEND

相关指令	MARK，MARKB，REG_POS，REG_POSB

REG_INPUTS -- 锁存输入映射

类型	轴状态
描述	**设置 R0-R3 输入锁存的输入口映射，每 4bit 对应一个锁存口。**
语法	VAR1 = REG_INPUTS bit / 锁存口 / 输入口范围(例如：ZMC306E) bit0-3 / R0 / 0-3 bit4-7 / R1 / 0-3 bit8-11 / R2 / 0-3 bit12-15 / R3 / 0-3 输入口范围：不同的控制器可以使用的锁存口个数不同。
适用控制器	3 系列部分 4 系列及以上支持锁存功能，最新固件
例子	BASE(6) ATYPE=3 REG_INPUTS=$3210 'R0-R3 分别对应输入口 0，1，2，3 REG_INPUTS=$1111 'R0-R3 都对应输入口 1
相关指令	REGIST

MARK -- 锁存触发

类型	轴状态
描述	**返回锁存事件是否产生。** 当 REGIST 指令执行时，MARK 变真，返回-1。当执行完成时，MARK 变成假，返回 0。
语法	VAR1 = MARK
适用控制器	通用

例子	BASE(0) '选择轴 0 MOVE(100) '运动 100units REGIST(3) '上升沿 R0 WAIT UNTIL **MARK** '等待触发
相关指令	REG_POS，REGIST

MARKB -- 锁存 2 触发

类型	轴状态
描述	**返回对应第二个锁存通道的锁存事件是否产生。** 当 REGIST 指令执行时，MARKB 变真，返回-1。当执行完成时，MARKB 变成假，返回 0。
语法	VAR1 = MARKB
适用控制器	通用
例子	BASE(0) '选择轴 0 MOVE(100) '运动 100units REGIST(14) '上升沿 R1 WAIT UNTIL **MARKB** '等待触发
相关指令	REG_POSB，REGIST

MARKC -- 锁存 3 触发

类型	轴状态
描述	**返回对应第三个锁存通道的锁存事件是否产生。** 当 REGIST 指令执行时，MARCK 变真，返回-1。当执行完成时，MARKC 变成假，返回 0。
语法	VAR1 = MARKC
适用控制器	通用
例子	BASE(0) '选择轴 0 MOVE(100) '运动 100units REGIST(18) '上升沿 R2 WAIT UNTIL **MARKC** '等待触发
相关指令	REG_POSC，REGIST

MARKD -- 锁存 4 触发

类型	轴状态
描述	**返回对应第四个锁存通道的锁存事件是否产生。**

	当 REGIST 指令执行时，MARKD 变真，返回-1。当执行完成时，MARKD 变成假，返回 0。
语法	VAR1 = MARKD
适用控制器	通用
例子	BASE(0) '选择轴 0 MOVE(100) '运动 100units REGIST(20) '上升沿 R3 WAIT UNTIL **MARKD** '等待触发
相关指令	REG_POSD，REGIST

OPEN_WIN -- 锁存开始坐标范围

类型	轴参数
描述	**锁存触发的开始坐标范围点。** 预留
语法	OPEN_WIN = pos
相关指令	REGIST，CLOSE_WIN

CLOSE_WIN -- 锁存结束坐标范围

类型	轴参数
描述	**锁存触发的结束坐标范围点。** 预留
语法	CLOSE_WIN = pos
相关指令	REGIST，OPEN_WIN

REG_POS -- 锁存位置

类型	轴状态
描述	**保存锁存的测量反馈位置(MPOS)，units 单位。**
语法	VAR1 = REG_POS
适用控制器	通用
例子	MOVE(100) '运动 100units REGIST(3) '上升沿 R0 WAIT UNTIL MARK '等待锁存发生 PRINT **REG_POS** '打印出锁存位置
相关指令	REGIST，MARK

REG_POSB -- 锁存 2 位置

类型	轴状态
描述	**返回锁存寄存器 2 的测量反馈位置(MPOS)，units 单位。**
语法	VAR1 = REG_POSB
适用控制器	通用
例子	MOVE(100) '运动 100units REGIST(16) '上升沿 EZ 信号触发 WAIT UNTIL MARKB '等待第二个锁存触发 PRINT **REG_POSB** '打印触发时位置
相关指令	REGIST，MARKB

REG_POSC -- 锁存 3 位置

类型	轴状态
描述	**返回锁存寄存器 3 的测量反馈位置(MPOS)，units 单位。**
语法	VAR1 = REG_POSC
适用控制器	通用
例子	MOVE(100) '运动 100units REGIST(18) '上升沿信号触发 WAIT UNTIL MARKC '等待第二个锁存触发 PRINT **REG_POSC** '打印触发时位置
相关指令	REGIST，MARKC

REG_POSD -- 锁存 4 位置

类型	轴状态
描述	**返回锁存寄存器 4 的测量反馈位置(MPOS)，units 单位。**
语法	VAR1 = REG_POSD
适用控制器	通用
例子	MOVE(100) '运动 100units REGIST(20) '上升沿信号触发 WAIT UNTIL MARKD '等待第二个锁存触发 PRINT **REG_POSD** '打印触发时位置
相关指令	REGIST，MARKD

10.12 限位参数指令

FS_LIMIT -- 正向软限位设置

类型	轴参数
描述	**正向软限位位置，单位是 units。** 当 FS_LIMIT 大于 REP_DIST 时，参数不起作用，正向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 FS_LIMIT 设置一个较大值即可。 FS_LIMIT 的值默认为 200000000。 软限位无法作为 DATUM 回零时的限位信号参考。
语法	VAR1 =FS_LIMIT，FS_LIMIT = expression
适用控制器	通用
例子	BASE(0) '选择轴 0 ATYPE=1 '轴类型设置 UNITS=100 '脉冲当量 100 DPOS=0 '坐标清 0 SPEED=100 '速度 100units/s ACCEL=1000 '加速度 1000units/s/s **FS_LIMIT**=200 '设置正向软限位 200units MOVE(300) '运动 300units 此时轴运动到 200 位置时会停止，并报错 Axis:0 AXISSTATUS:200h,FSOFT 继续操作轴时只能向负向运行。 取消设置时，只需要把值设大些。 **FS_LIMIT**=2000000 '取消正向软限位设置
相关指令	RS_LIMIT，FWD_IN， REV_IN

RS_LIMIT -- 负向软限位设置

类型	轴参数
描述	**负向软限位位置，单位是 units。** 当 RS_LIMIT 的绝对值大于 REP_DIST 时，参数不起作用，负向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 RS_LIMIT 设置一个较大值即可。 RS_LIMIT 的值默认为-200000000。 软限位无法作为 DATUM 回零时的限位信号参考。
语法	VAR1 =RS_LIMIT，RS_LIMIT = expression
适用控制器	通用
例子	**RS_LIMIT** = -50 '设置负向软限位 50units **RS_LIMIT**= - 2000000 '取消负向软限位设置

相关指令	FS_LIMIT，FWD_IN，REV_IN

FWD_IN -- 映射正限位输入

类型	轴参数
描述	**正向硬件限位开关对应的输入点编号，-1 无效。** 控制器限位信号生效后，会立即停止轴，停止减速度为 FASTDEC。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。
语法	VAR1 = FWD_IN，FWD_IN = expression
适用控制器	通用
例子	BASE(0,1,2,3)　　'选择轴 0,1,2,3 **FWD_IN**=6,7,8,9　　'分别设置正向限位开关 INVERT_IN(6,ON)　　'反转信号 INVERT_IN(7,ON) INVERT_IN(8,ON) INVERT_IN(9,ON) 当开关输入信号时，对应轴的轴状态会报警 Axis:0 AXISSTATUS:10h,FWD。 此时只可以负向运动。
相关指令	REV_IN,，FS_LIMIT，FASTDEC

REV_IN -- 映射负限位输入

类型	轴参数
描述	**负向硬件限位开关对应的输入点编号，-1 无效。** 控制器限位信号生效后，会立即停止轴，停止减速度为 FASTDEC。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。
语法	VAR1 = REV_IN，REV_IN = expression
适用控制器	通用
例子	参考 FWD_IN 例子
相关指令	FWD_IN，RS_LIMIT，FASTDEC

ALM_IN -- 映射报警输入

类型	轴参数
描述	**驱动器告警对应的输入口编号，-1 无效。** 控制器报警信号生效后，会立即停止轴，停止减速度为 FASTDEC。 设置了告警输入口后，ZMC 控制器缺省为 OFF 有效，常开信号用 INVERT_IN 反转电平； ECI 控制器缺省为 ON 有效，常闭信号用 INVERT_IN 反转电平。
语法	VAR1 = ALM_IN，ALM_IN = expression
适用控制器	通用
例子	BASE(0,1) **ALM_IN** = 10,11　　'将轴 0 告警信号定义到输入口 10，轴 1 定义到 11 INVERT_IN(10,ON)　'反转电平开启 INVERT_IN(11,ON)
相关指令	DATUM_IN，FWD_IN，REV_IN，INVERT_IN，FASTDEC

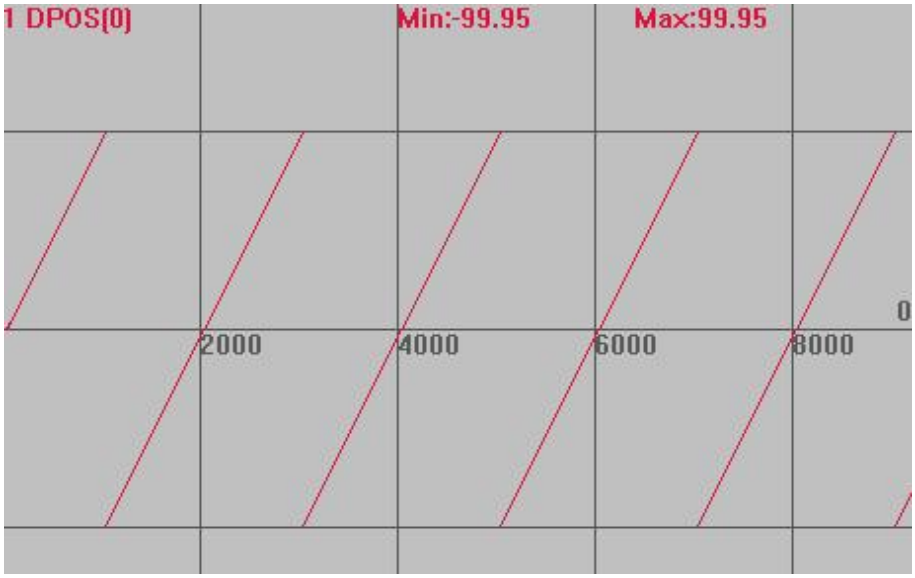
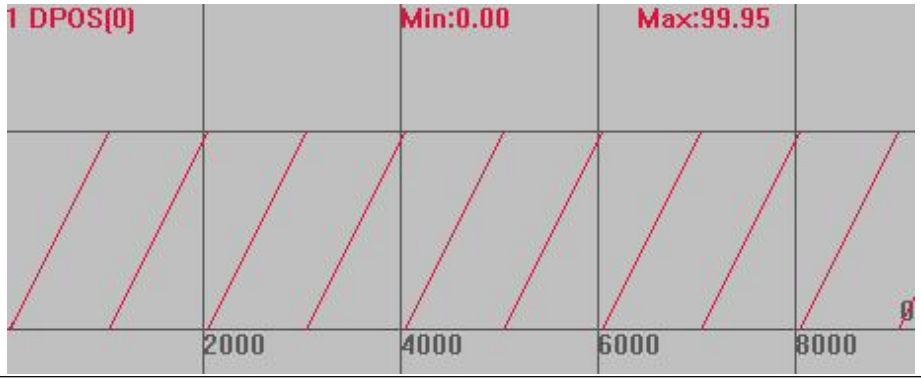
10.13 限幅参数指令

REP_OPTION -- 坐标循环模式

类型	轴参数
描述	**坐标重复设置。** 可用于限制凸轮主轴的坐标循环范围，来实现多条凸轮曲线连续。 使用绝对运动模式时，如果目标位置处于坐标循环范围内，可以正确运动；如果处于坐标循环范围外，运动不正确。 相对运动不受影响。
语法	VAR1 = REP_OPTION，REP_OPTION = opt opt：按位来表示不同的意义

位	值	描述
0	1	0-循环范围 - REP_DIST 到 + REP_DIST 1-循环范围为 0 到 + REP_DIST
1	2	写 1 禁止 CAMBOX 和 MOVELINK 的重复运动，禁止生效后还原为 0
2	4	预留
4	16	1-不使用 REP_DIST，0-使用 REP_DIST

适用控制器	通用
例子	BASE(0)　　'选择轴 0 ATYPE=1 UNITS=100　　'脉冲当量 100 DPOS=0　　'坐标清 0 SPEED=100　　'速度 100units/s

	ACCEL=1000 '加速度 1000units/s/s DECEL=1000 '加速度 1000units/s/s REP_DIST=100 '设置坐标循环范围 **REP_OPTION**=0 '设置循环模式 TRIGGER '自动触发示波器 VMOVE(1) '持续运动 坐标曲线 DPOS(0)垂直刻度 100  REP_OPTION=1 时 MSPEED(0)垂直刻度 100 
相关指令	CAMBOX，MOVELINK，REP_DIST

REP_DIST -- 坐标循环位置

类型	轴参数
描述	**根据 REP_OPTION 设置来自动循环轴 DPOS 和 MPOS 坐标。**
语法	VAR1 = REP_DIST，REP_DIST = expression
适用控制器	通用

例子	参考 REP_OPTION
相关指令	REP_OPTION

FE -- 当前随动误差

类型	轴状态
描述	**随动误差，该值等于 DPOS-MPOS。**
语法	VAR1 = DPOS
适用控制器	通用
相关指令	MPOS，DPOS

FE_LIMIT -- 最大随动误差设置

类型	轴参数
描述	**最大允许的随动误差值。** 预留
语法	VAR1 = FE_LIMIT，FE_LIMIT = expression
相关指令	FE，FE_RANGE

FE_RANGE -- 报警时随动误差

类型	轴参数
描述	**报警时的随动误差值。** 预留
语法	VAR1 = FE_RANGE，FE_RANGE = expression
相关指令	FE，FE_LIMIT

10.14 高级设置指令

INVERT_STEP -- 脉冲模式设置

类型	轴参数
描述	**伺服/步进脉冲输出模式设置。** 有脉冲方向、双脉冲、正交脉冲三种模式，控制器默认为脉冲方向控制（模式 0）。正交脉冲目前只有 4 系列及以上的控制器支持。

<table>
<tr><td></td><td>反馈位置（MPOS）信息牵涉到很多复杂功能，例如 MOVE_OP 精准模式，所以控制器暂不支持修改反馈位置的方向，需要修改时可以修改驱动相关参数，例如三菱为 PA14。</td></tr>
<tr><td>语法</td><td>INVERT_STEP = mode
mode：模式选择，缺省 0
低 8 位（位 0-位 7）表示的模式值如下：
0-3 脉冲方向模式，脉冲线+方向线
4-7 双脉冲方式（或称 CW/CCW），正向脉冲线+负向脉冲线
8-9 AB 输出(部分控制器定制)
各个模式对应的电平如下：
<table>
<tr><td rowspan="2">模式值</td><td rowspan="2">描述</td><td colspan="2">松下设置参考</td><td>三菱设置参考</td></tr>
<tr><td>Pr0.06</td><td>Pr0.07</td><td>PA13</td></tr>
<tr><td>0</td><td>脉冲/方向（脉冲正逻辑）（正向）</td><td>0</td><td>3</td><td>××01h</td></tr>
<tr><td>1</td><td>脉冲/方向（脉冲负逻辑）（正向）</td><td>/</td><td>/</td><td>××11h</td></tr>
<tr><td>2</td><td>脉冲/方向（脉冲正逻辑）（负向）</td><td>1</td><td>3</td><td>××01h</td></tr>
<tr><td>3</td><td>脉冲/方向（脉冲负逻辑）（负向）</td><td>/</td><td>/</td><td>××11h</td></tr>
<tr><td>4</td><td>双脉冲（方向负逻辑）（正向）</td><td>/</td><td>/</td><td>××10h</td></tr>
<tr><td>5</td><td>双脉冲（方向负逻辑）（负向）</td><td>/</td><td>/</td><td>××10h</td></tr>
<tr><td>6</td><td>双脉冲（方向正逻辑）（正向）</td><td>1</td><td>1</td><td>×× 00h（默认）</td></tr>
<tr><td>7</td><td>双脉冲（方向正逻辑）（负向）</td><td>0(默认)</td><td>1(默认)</td><td>×× 00h（默认）</td></tr>
</table>
高 8 位（位 8-位 15）表示方向变化保护时间，单位微秒：0-255

常用模式为 0、2、6、7。
如果模式设定不正确，步进马达可能会在换向时丢失一步的位置，当不确定步进马达的设置时，可以设置 100 微秒左右的保护时间。</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>设置为脉冲方向模式：
INVERT_STEP = 256*100+0 '设置 100 微秒的保护时间，模式为 0
设置为双脉冲模式：
INVERT_STEP = 256*100+6 '设置 100 微秒的保护时间，模式为 6

查询脉冲模式设置：
在线命令栏输入如下指令查询。
?INVERT_STEP(0) '打印轴 0 的脉冲模式设置值
?*INVERT_STEP '打印全部轴的脉冲模式设置值</td></tr>
<tr><td>相关指令</td><td>STEP_RATIO</td></tr>
</table>

MAX_SPEED -- 脉冲频率限制

类型	轴参数
描述	**脉冲输出的最高频率限制。**

	一旦发现超过此设置值会强制，并且设置 AXISSTATUS。 对编码器轴，设置值低于 500K 时会启用编码器滤波，设置值高于 1M 时会取消编码器滤波设置。默认值为 1000000（老固件的默认脉冲频率为 500000）。 使用直线电机速度较快时，一般容易脉冲频率超限，可把数值适当设置大点。
语法	MAX_SPEED = value
适用控制器	通用
例子	**MAX_SPEED** AXIS(n)=4000000　　'设置轴 n 脉冲速度限制 4000000
相关指令	AXISSTATUS

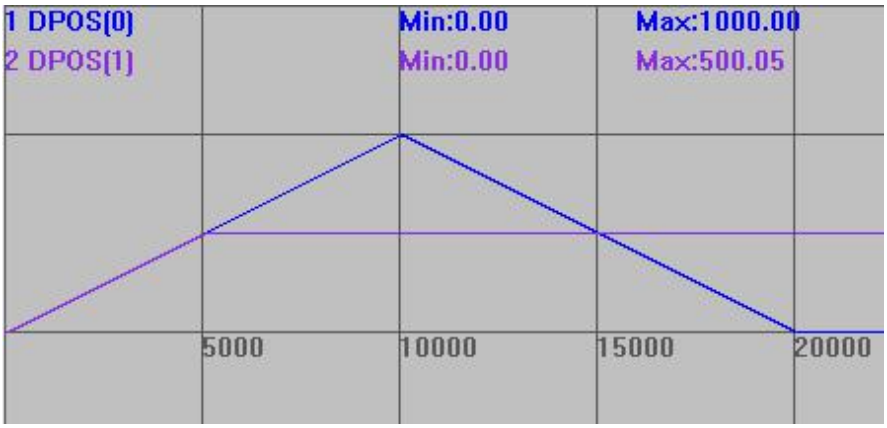
AXIS_ZSET -- 精准 op 设置

类型	轴参数
描述	**对轴启用 MOVEOP 精准输出功能，作用在轴组的主轴上。** SYSTEM_ZSET 修改的同时会修改当前 BASE 轴的 AXIS_ZSET，以兼容旧的程序，一般不建议使用 SYSTEM_ZSET 指令。 设置参数： bit0：1-VP_SPEED 缺省使用插补速度，0-VP_SPEED 使用单轴的速度 bit1：1-使用 MOVE_OP 精确输出功能， 0- MOVE_OP 原来的方式 bit4：1-对带编码器功能的轴，使用编码器位置的 MOVE_OP 精准方式 多个编码器轴插补时，使用 BASE 运动主轴的设置
语法	可读：VALUE=AXIS_ZSET 可写：AXIS_ZSET=VALUE
适用控制器	20170517 以上固件版本
例程	例一 开启精准输出 BASE(0) ATYPE=1 DPOS=0 SPEED=100 ACCEL=1000 DECEL=1000 AXIS_ZSET(0)=2　'开启 MOVE_OP 的精准输出功能 MOVE(100) MOVE_OP(0,1)　'精准生效，选择输出通道 0 例二 开启多个编码器精准输出口 一般 4 系列有 4 个通道可以用于精准输出，部分型号有 8 个，假设设备上有 3 个点胶工位都要精准输出。 BASE(0,1,2)　'选择轴 0 为主轴 AIXS_ZSET(0)=19　'开启主轴 MOVE_OP 的编码器精准输出功能 …… BASE(3,4,5)

	AIXS_ZSET(3)=19 …… BASE(6,7,8) AIXS_ZSET(6)=19 ……
相关指令	SYSTEM_ZSET，MOVE_OP

AXIS_MODE -- connect 运动保持

类型	轴参数
描述	**设置 BIT1=1，防止限位和软限位导致 CONNECT 运动退出。** 设置为 0 时，当碰到限位后，主从轴的 CONNECT 连接关系断开，消除限位报警后，此时操作主轴，从轴不再跟随；设置为 2，碰到限位后仍然保持连接关系，消除报警后，从轴仍然保持关联。 固件版本 20170616 及以上支持此功能。
语法	VAR1 = AXIS_MODE，AXIS_MODE = expression
例程	例一　不设置 AXIS_MODE 参数 RAPIDSTOP(2) WAIT IDLE BASE(0,1) ATYPE=1,1 UNITS=100,100 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 AXIS_MODE=0,2　　'设置参数 FS_LIMIT=1000,500 TRIGGER　　'自动触发示波器 CONNECT(1,0)　AXIS(1)　　'轴 1 连接到轴 0，比例为 1 MOVE(1000)　AXIS(0) MOVE(-1000)　AXIS(0) 运动轨迹如下： DPOS(0)垂直刻度 1000，无偏移 DPOS(1)垂直刻度 1000，无偏移



轴 1 碰到限位后停止，和轴 0 的 connect 连接断开，后续轴 0 的运动与轴 1 无关。

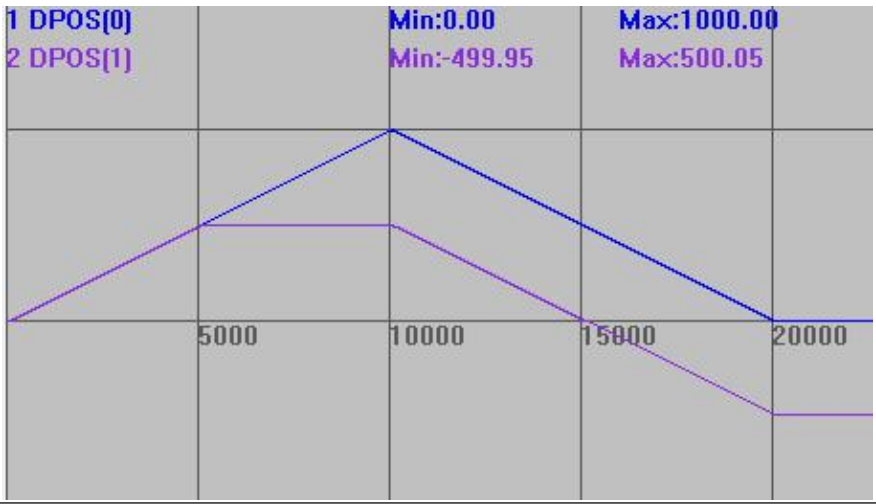
例二　设置 AXIS_MODE 参数

```
RAPIDSTOP(2)
WAIT IDLE
BASE(0,1)
ATYPE=1,1
UNITS=100,100
DPOS=0,0
SPEED=100,100
ACCEL=1000,1000
DECEL=1000,1000
AXIS_MODE=0,2
FS_LIMIT=1000,500
TRIGGER                          '自动触发示波器
CONNECT(1,0) AXIS(1)         '轴 1 连接到轴 0，比例为 1
MOVE(1000) AXIS(0)
MOVE(-1000) AXIS(0)
```

运动轨迹如下：

DPOS(0)垂直刻度 1000，无偏移

DPOS(1)垂直刻度 1000，无偏移



	轴 1 碰到限位后停止，但和轴 0 的 connect 并未断开，运动到限位位置内，跟随轴 0 运动。
适用控制器	通用
相关指令	CONNECT，FS_LIMIT，RS_LIMIT

MOVEOP_DELAY -- 缓冲输出延时

类型	轴参数
描述	**BASE 轴缓冲信号延时输出。** 设置在 BASE 主轴上，当 MOVE_OP 精准功能使用时，可以调整实际触发 OP 操作的时间，毫秒(ms)单位，支持小数，实际最长延时时间 100ms。 设置为负数，可以提前打开 OP，一般步进可以提前 2ms，伺服可以提前 20ms，点胶起胶可以使用。
语法	MOVEOP_DELAY=timems timems：毫秒数
适用控制器	20170505 以上固件版本
相关指令	MOVE_OP，HW_PSWITCH，HW_PSWITCH2

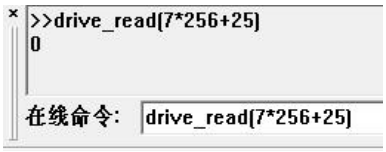
DAC -- 总线轴模拟量控制

类型	轴参数
描述	**伺服轴 DA 直接控制，速度或力矩模式下支持。** 单位为 DA 模块的刻度，12 为或 16 位。 速度控制时要看驱动器具体单位。 力矩控制时单位为千分之一，等于 1000 时表示 100%力矩。
语法	VAR1 = DAC，DAC = expression
使用控制器	带 EtherCAT 接口或 Rtex 接口，2017 以后固件支持
例程	例一 Rtex 速度控制 请先使用 Rtex 初始化例程，并将例程中的 ATYPE 设置为 51。 然后可在 ZDevelop 在线命令直接发送 dac 指令，如下图  此时电机会以 10r/min 的速度旋转，发送 dac= - 10 会反向旋转。 也可以在程序中发送 dac 指令。 速度单位根据驱动器手册确认，如下：

指令速度
[大小]：带符号 32bit
[单位]：通过 Pr7.25（RTEX 速度单位）设定

Pr7.25	单位
0	[r/min]
1	[指令单位/s]

[设定单位]：负向最大过速度等级~正向最大过速度等级
r/min 单位下设定时，在内部演算时换算到指令单位/s，换算后的值限制在以下范围：-80000001h~7FFFFFFFh

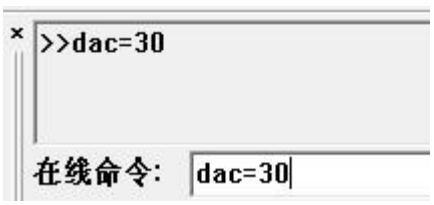


此时单位为【r/min】

例二 Rtex 力矩控制
请先将驱动器参数 pr6.47 的第一位置为 0，关闭 2 自由度控制模式；参数 pr3.17 设置速度限制，如下图。（参照松下 Rtex 手册）
Pr3.17(速度限制选择)的设定值是 1 时，可以通过 SL_SW 切换转矩控制时的速度限制值。

分类	3	3	3
No.	17	21	22
属性	B	B	B
参数名称	速度限制选择	速度限制值 1	速度限制值 2
设定范围	0~1	0~20000	0~20000
单位	/	r/min	r/min
功能	值 | SL_SW: 0 | 1 0 | Pr3.21 1 | Pr3.21 | Pr3.22 设定转矩控制时的速度限制值的选择方式	设定转矩控制时的速度限制；转矩控制中在不超过速度限制值设定的速度内进行控制，另外，内部值被 Pr5.13(过速度等级)、Pr6.15（第二过速度等级）、Pr9.10（最大过速度等级）的最小设定速度进行限制	Pr3.17（速度限制选择）=1 时，设定 SL_SW 为 1 时的速度限制，另外，内部值被 Pr5.13（过速度等级）、Pr6.15（第二过速度等级）、Pr9.10（最大过速度等级）的最小设定速度进行限制

再使用 Rtex 初始化例程，并将例程中的 ATYPE 设置为 52。
然后可在 ZDevelop 在线命令直接发送 dac 指令，此时电机会开始转动。



当前驱动器为 0.03 的力矩，如果 dac 发送过小时，电机无法克服摩擦力，不能转动。

	也可以在程序中发送 dac 指令。
	力矩控制时 dac 的单位为千分之一的力矩，dac=1000 是表示 100%
	[大小]：带符号 32bit
	[单位]：0.1%
	[设定范围]：负向电机最大转矩~正向电机最大转矩
	最大转矩限制[%]=100×Pr9.07/(Pr9.06×2½)
	
	例三 EtherCAT 速度控制
	FOR I=0 TO 1 '初次使用将所有轴设为普通脉冲类型
	ATYPE(I)=1
	NEXT
	SLOT_SCAN(0) '总线扫描开始
	
	IF NODE_COUNT(0,0)>0 THEN
	AXIS_ADDRESS(0)=1 '将第一个驱动器映射到轴 0
	ATYPE(0)=66 '66 为速度控制模式
	DRIVE_PROFILE(0)=20 '速度控制要设置为 20
	DELAY (200)
	SLOT_START(0) '总线扫描成功，启动总线
	
	DRIVE_CONTROLWORD(0)=128 '清除驱动器错误
	DELAY (2)
	DRIVE_CONTROLWORD(0)=6
	DELAY (2)
	DRIVE_CONTROLWORD(0)=15
	DELAY (2)
	DELAY(20)
	DATUM(0) '清除控制器错误
	BASE(0)
	AXIS_ENABLE=1 '映射轴使能打开
	WDOG=1 '轴使能
	DAC=10000 '电机以 10000 脉冲每秒的速度转动
	ENDIF
	
	例四 EtherCAT 力矩控制
	将例三中的 ATYPE 改为 67，DRIVE_PROFILE 改为 30 即可。
	此时 dac 发送范围 0~1000,1000 表示 100%力矩，反向选择发送负值即可。

相关指令	SERVO

ERRORMASK -- 错误时操作

类型	轴参数

描述	**和 AXISSTATUS 做与运算来决定哪些错误需要关闭 WDOG。**
语法	VAR1 = ERRORMASK，ERRORMASK = expression
适用控制器	通用
相关指令	AXISSTATUS，WDOG

ZSCAN_CORRECT -- 振镜矫正

类型	轴参数
描述	**矫正振镜轴参数。**
语法	ZSCAN_CORRECT(ixy,imode,imaxline,imaxrow,x1,y1,x2,y2,tableindex) ixy：值为 0 或 1，两个振镜选择；0-第一个振镜，1-第二个振镜 imode：0-关闭矫正功能；1-使用分区矫正 imaxline：行数，Y 方向的点数为行数 imaxrow：列数，X 方向的点数为列数 x1,y1,x2,y2：理论的左下角与右上角的位置 tableindex：测量的实际坐标开始存储的 table 索引，先 X 再 Y，先第一行（按列数存储），再下一行
适用控制器	带振镜轴控制器

10.15 预留指令

D_GAIN -- 微分增益

类型	轴参数
描述	**微分增益，非模拟量伺服不支持。** 预留
语法	VAR1 = D_GAIN
相关指令	P_GAIN，I_GAIN，OV_GAIN，VFF_GAIN

I_GAIN -- 积分增益

类型	轴参数
描述	**积分增益，非模拟量伺服不支持。** 预留
语法	I_GAIN = expression
相关指令	P_GAIN，D_GAIN，OV_GAIN，VFF_GAIN

OV_GAIN -- 速度增益

类型	轴参数
描述	**速度增益，非模拟量伺服不支持。** 预留
语法	VAR1 = OV_GAIN，OV_GAIN = expression
相关指令	P_GAIN，D_GAIN，I_GAIN，VFF_GAIN

P_GAIN -- 比例增益

类型	轴参数
描述	**比例增益，非模拟量伺服不支持。** 预留
语法	VAR1 = P_GAIN，P_GAIN = expression
相关指令	I_GAIN，D_GAIN，OV_GAIN，VFF_GAIN

VFF_GAIN -- 前馈增益

类型	轴参数
描述	**速度反馈的前馈增益，非模拟量伺服不支持。** 预留
语法	VAR1 = VFF_GAIN，VFF_GAIN = expression
相关指令	P_GAIN，I_GAIN，D_GAIN，OV_GAIN

SERVO -- 闭环开关

类型	轴参数
描述	**闭环开关设置。** 预留
语法	VAR1 = SERVO，SERVO = ON/OFF
相关指令	WDOG

TRANS_DPOS

类型	轴状态
描述	预留

第十一章 输入输出相关指令

11.1 输入相关指令

IN -- 输入口

类型	输入输出函数
描述	**读取输入，没有参数时返回 0-31 的状态。** 读取的是 INVERT_IN 翻转以后的状态。 ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16 +拨码组合值*16），EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看硬件手册。 注意 IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。
语法	IN([channel1],[channel2]) channel1：要读取的起始输入通道 channel2：要读取的结束输入通道，没有结束通道时，返回单个通道的输入状态
适用控制器	通用
例子	a=**IN**(1)　　'读取通道 1 的输入
相关指令	OP，INVERT_IN

AIN -- 模拟量输入

类型	输入输出函数
描述	**读取模拟输入，返回 AD 转换模块的刻度值。** 12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。 ZAIO 扩展板的 AD 通道号与拨码有关，起始值为（8 +拨码组合值*8），ZMIO 总线扩展 AD 使用 NODE_AIO 指令只能设置为 8 的倍数，详细查看硬件手册。 注意 AIO 映射编号要大于控制器自身最大的 AIO 编号，不能与控制器的编号重合。
语法	VAR=AIN(channel) channel：模拟输入通道 0-127
适用控制器	通用
例子	a=**AIN**(1)　　'读取通道 1 的 AD a=**AIN**(1) *10 /4095　　'通道 1 的电压值
相关指令	AOUT

ZSIMU_IN -- 仿真 IN 输入

类型	仿真器专用指令
描述	**模拟输入 IN 口的输入。**
语法	ZSIMU_IN[([ionum ,] value)] ionum：输入编号，从 0 开始，没有这个参数时输出 0-31 value：输入状态
适用控制器	通用
例子	**ZSIMU_IN**(0,1)　　'仿真输入 0 有效
相关指令	IN

ZSIMU_AIN -- 仿真模拟量输入

类型	仿真器专用指令
描述	**模拟模拟量输入口的输入。**
语法	ZSIMU_AIN(ionum, value)
适用控制器	通用
例子	**ZSIMU_AIN**(0,1024)　　'仿真通道 0
相关指令	AIN

ZSIMU_ENCODER -- 仿真编码器输入

类型	仿真器专用指令
描述	**模拟编码器输入口的输入。**
语法	ZSIMU_ENCODER(axis num, value) axis num：轴编号，从 0 开始 value：ENCODER 的仿真值
适用控制器	通用
例子	**ZSIMU_ENCODER**(0,1024)　　'ENCODER = 1024
相关指令	ENCODER

INVERT_IN -- 反转输入

类型	特殊指令
描述	**反转输入状态，可以读取判断是否有反转。**
语法	INVERT_IN(channel, state)，　VAR1= INVERT_IN(channel)

	channel：输入通道 state：ON/OFF
适用控制器	通用
例子	**INVERT_IN**(1,ON)　　'ZMC 系列控制器输入 OFF 时认为有信号输入(ECI 系列控制器与之相反) FWD_IN(0)=1　　'输入 IN1 作为轴 0 的正向限位信号
相关指令	IN

IN_SCAN -- 扫描输入变化

类型	输入输出函数
描述	**扫描输入变动，返回值 1(TRUE)-变动，0(FALSE)-没有变动。** 此函数必须固定不断的扫描，返回的是两次扫描之间的变动，可以通过 IN_EVENT 读取变动的具体情况，使用的是 INVERT_IN 翻转以后的状态。 固件版本 20140214 以后的才提供这个支持，扫描范围有宽度限制。 00x 系列控制器只能在单个任务中使用。
语法	VAR1=IN_SCAN([channel1][,channel2]) channel1：要读取的起始输入通道 channel2：要读取的结束输入通道，没有结束通道时，扫描单个输入
适用控制器	通用
例子	WHILE 1 IF **IN_SCAN**(0,23) THEN　　'扫描 IN0-23 口电平变化 IF IN_EVENT(0) > 0 THEN　　'IN0 上升沿触发 PRINT "IN0 UP", IN_BUFF(0) ELSEIF IN_EVENT(0) < 0 THEN　　'IN0 下降沿触发 PRINT "IN0 DOWN", IN_BUFF(0) ENDIF ENDIF WEND
相关指令	IN_EVENT，SCAN_EVENT，IN_BUFF

IN_EVENT -- 读取输入变化

类型	输入输出函数
描述	**读取输入的变化情况。** 1-上升沿，-1-下降沿，0-没有变化。 此函数必须与 IN_SCAN 配合使用。
语法	VAR1 = IN_EVENT(IONUM)
适用控制器	通用

例子	参见 IN_SCAN
相关指令	IN_SCAN，SCAN_EVENT

SCAN_EVENT -- 检测变化

类型	输入输出函数
描述	**检测表达式的内容变化。** OFF- ON 返回 1，ON-OFF 返回-1，不变返回 0。 150810 之后固件版本支持，之前版本用 IN_EVENT 和 IN_SCAN。
语法	ret = SCAN_EVENT (expression) expression：任意有效的表达式，结果会转成 BOOL 类型
适用控制器	通用
例子	例一 输入信号扫描 WHILE 1 IF SCAN_EVENT(IN(0)) > 0 THEN 'IN0 上升沿触发 PRINT "IN0 ON" ELSEIF SCAN_EVENT(IN(0))<0 THEN 'IN0 下降沿触发 PRINT "IN0 OFF" ENDIF WEND 例二 寄存器、变量扫描 WHILE 1 IF SCAN_EVENT(TABLE(0)) > 0 THEN 'TABLE0 上升沿触发 PRINT "TABLE0 ON" ELSEIF SCAN_EVENT(TABLE(0))<0 THEN 'TABLE0 下降沿触发 PRINT "TABLE0 OFF" ENDIF WEND 在线命令操作 table(0)，打印相关结果
相关指令	IN_SCAN，IN_EVENT

IN_BUFF -- 读取输入缓冲

类型	输入输出函数
描述	**读取 IN_SCAN 缓冲的当前输入，没有参数时返回 0-31 的状态。** 读取的是 INVERT_IN 翻转以后的状态。
语法	IN_BUFF([channel1],[channel2]) channel1：要读取的起始输入通道，必须是 IN_SCAN 的输入范围

	channel2：要读取的结束输入通道，没有结束通道时，返回单个通道的输入状态
适用控制器	通用
例子	参见 IN_SCAN
相关指令	IN_SCAN

INFILTER -- 输入口滤波

类型	系统参数
描述	**本地输入口的滤波参数。** 值越大，滤波时间越长， 2-9，缺省 2。
语法	VAR1 = INFILTER，INFILTER= expression
适用控制器	通用
例子	**INFILTER**= 5 '在干扰严重场合加大滤波时间

11.2 输出相关指令

OP -- 输出口

类型	输入输出指令和函数
描述	**输出或读取输出状态。** 当在表达式中使用时，自动为函数语法。 ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16 +拨码组合值*16），EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看硬件手册。 注意 IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。 最多可操作 32 个输出口。
语法	OP([ionum],value) 或 OP(ionum1, ionum2,value[,mask]) 或 OP([firstnum],[finalnum]) ionum：输出编号，从 0 开始 value：输出状态，多个输出口操作时按位来指明多个口状态 ionum1：要操作的第一个输出通道 ionum2：要操作的最后一个输出通道 mask：用位来指定哪些 IO 需要操作，不填时第一个通道到最后一个通道都操作 firstnum：输出编号，从 0 开始 finalnum：输出编号，从 0 开始，没有这个参数时，读取单个输出口状态
适用控制器	通用
例子	例一 单个操作 '翻转输出口 0 IF **OP** (0) = ON THEN

	OP (0, OFF)
	ELSE
	OP (0, ON)
	ENDIF
	
	例二 区域操作
	OP(0,7,$FF) 'bit0-bit7 全开
	DELAY(1000)
	OP(0,7,0)
	
	OP(8,15,$FF) 'bit8-bit15 全开
	DELAY(1000)
	OP(8,15,0)
	
	OP(0,15,$FFFF) 'bit0-bit15 全开
	DELAY(1000)
	OP(0,15,0)
	
	OP(0,31,$FFFFFFFF) 'bit0-bit31 全开
	DELAY(1000)
	OP(0,31,0)

相关指令	READ_OP，MOVE_OP

AOUT -- 模拟量输出

类型	输入输出指令或函数
描述	**模拟通道输出。** 12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。
语法	AOUT(channel) = value channel：模拟输出通道 0-63 扩展板 DA 通道号与拨码有关，起始值为（4 +拨码组合值*4），详细查看硬件手册。
适用控制器	通用
例子	**AOUT**(1) = 0 '关闭输出 DA 通道 1 **AOUT**(1) = 4095 'DA1 口输出 10V 电压
相关指令	AIN

READ_OP -- 读取输出口

类型	输入输出函数
描述	**读取输出状态。**

	同 OP，多个输出口操作时按位来指明多个口状态。
语法	READ_OP ([firstnum[,[finalnum]) firstnum：起始输出编号，从 0 开始 finalnum：结束输出编号，从 0 开始，没有这个参数时，读取单个输出口的状态
适用控制器	通用

例子	'翻转输出口 0
	IF **READ_OP** (0) = ON THEN
	OP (0, OFF)
	ELSE
	OP (0, ON)
	ENDIF

相关指令	OP

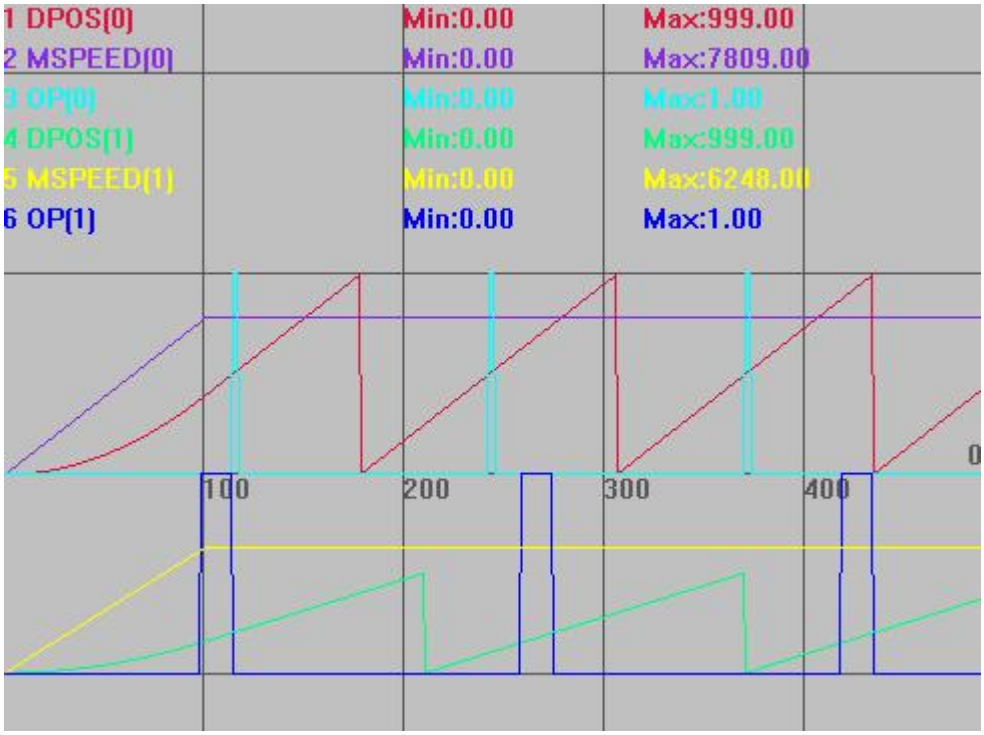
11.3 位置比较输出指令

PSWITCH -- 软件位置比较输出

类型	输入输出指令
描述	**根据位置比较来操作输出口。** 如果多个 PSWITCH 操作同一个输出口，需要编号顺序排在一起。 使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。
语法	PSWITCH(num,enable,[,axis,op num,op state,set pos,reset pos]) num：比较器的编号，ZMC1xx 系列有 16 个比较器，编号：0-15 enable：操作比较器的使能 - ON 启动 / OFF 取消 axis：指定要获取位置的轴号 op num：操作的 IO 编号 op state：输出的状态，1 表示在下面位置范围内输出为 ON，0 表示在下面位置范围内输出为 OFF set pos：设定产生输出的起始位置，采用 units 单位 reset pos：设定输出复位的位置，采用 units 单位 不同型号控制器支持的比较器个数不同，使用?*max 指令打印查看 max_pswitch 参数确认个数。
适用控制器	通用
例子	RAPIDSTOP(2) WAIT IDLE DELAY(1000) ERRSWITCH = 3 BASE(0,1) '选择轴号 ATYPE=1,1 '脉冲方式步进或伺服 DPOS = 0,0

```
UNITS = 1,1         '脉冲当量
SPEED = 10000,10000
ACCEL=SPEED(0)*10,SPEED(1)*10
DECEL=SPEED(0)*10,SPEED(1)*10
REP_OPTION=1,1        '设置坐标循环范围为 0 到+ REP_DIST
REP_DIST=1000,1000
TRIGGER
MOVE(10000,8000)
PSWITCH(0,ON,0,0,ON,500,520)
PSWITCH(1,ON,1,1,ON,300,400)
END
```

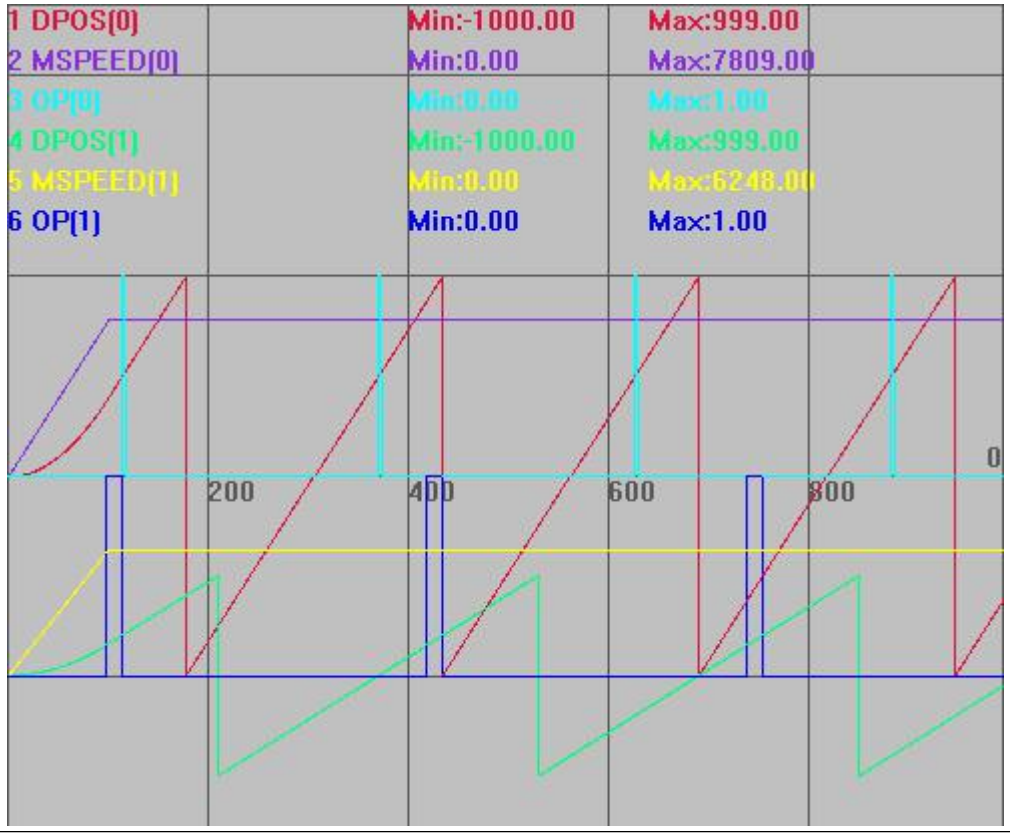
DPOS(0)垂直刻度 1000，无偏移
MSPEED(0)垂直刻度 10000，无偏移
OP(0)垂直刻度 1，无偏移
DPOS(0)垂直刻度 2000，偏移-2000
MSPEED (0)垂直刻度 10000，偏移-10000
OP(0)垂直刻度 5，偏移-1



以上例程，仅修改如下指令，得出波形如下图。

```
REP_OPTION=0,0        '设置坐标循环范围为- REP_DIST 到+ REP_DIST
```

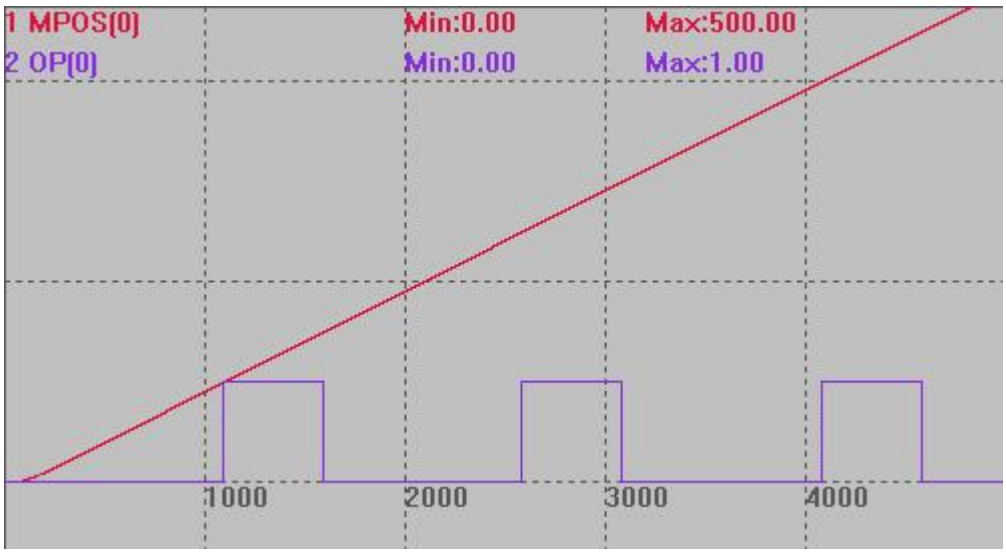
DPOS(0)垂直刻度 1000，无偏移
MSPEED(0)垂直刻度 10000，无偏移
OP(0)垂直刻度 1，无偏移
DPOS(0)垂直刻度 2000，偏移-2000

	MSPEED (0)垂直刻度 10000，偏移-10000 OP(0)垂直刻度 5，偏移-1 
相关指令	HW_PSWITCH

HW_PSWITCH -- 硬件位置比较输出

类型	轴指令
描述	**硬件比较输出，不同的轴对应不同的输出口。** 缺省轴 0-5 分别对应输出口 0 1 2 3 0 1。总共 4 个比较输出口。 可以连续调用两个 HW_PSWITCH 指令，通过函数方式可以获取能调用的指令个数。 每个比较点触发都会使得当前输出口电平翻转。 HW 缓冲数 1024 个，可以连续调用 1024 个 HW 指令。 HW 指令调用后,不受后面的坐标修改功能的影响，HW 指令 TABLE 存储的坐标必须在调用时是正确的，因此尽量手动修改坐标,要规避 HW 指令与坐标循环自动修改的随机冲突。 因为自动循环修改坐标不受程序控制，无法确定是在 HW 的前面还是后面，这样 TABLE 里面的坐标就无法确定。 此指令仅支持脉冲轴硬件位置比较输出，总线轴请使用 HW_PSWITCH2 指令。 使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。

语法	HW_PSWITCH(mode, direction, reserve, tablestart, tableend) Buff = HW_PSWITCH([axisnum]) mode：1-启动比较器，2- 停止并删除没完成的比较点 direction：0-坐标负向，1- 坐标正向，2-不判断方向 reserve：预留 tablestart：第一个比较点坐标所在 TABLE 编号 tableend：最后一个比较点坐标所在 TABLE 编号 HW_PSWITCH 没有比较完所有点的话，一定要设置 mode 值为 2，通过 HW_PSWITCH(2)指令停止并删除没有完成的比较点，否则后面此输出通道会工作不正常。
适用控制器	4 系列及以上产品，最新固件 ZMC420SCAN 不支持 HW_PSWTICH
例子	以下例程测试环境：ZMC432，固件：20170709（仿真器无法运行此指令）。 BASE(0)　　'选择轴 0 默认输出 OP(0) ATYPE=4　　'采用编码器位置作比较输出参考点,无编码器改成脉冲轴类型 UNITS=100 SPEED=100 ACCEL=500 MPOS=0 OP(0,OFF) TABLE(0, 100,150,250,300,400,450) 'OP0 在 MPOS100-150 间打开，150-250 间关闭，250-300 间打开，300-400 间关闭，400-450 间打开，450 后关闭 **HW_PSWITCH**(2)　'停止并删除没有完成的比较点 **HW_PSWITCH**(1, 1, 0, 0, 5)　'启动比较输出 TRIGGER MOVEABS(500) 比较输出图 MPOS(0)垂直刻度 200 OP(0)垂直刻度 2

	
相关指令	PSWITCH，HW_PSWITCH2

HW_TIMER -- 硬件定时

类型	特殊命令
描述	**硬件定时器，用于硬件比较输出后一段时间后还原电平。** HW_TIMER 只有 1 个，每次调用会强制停止之前的调用。 ZMC420SCAN 每个输出口的 HW_TIMER 功能独立。
语法	HW_TIMER(mode, cyclonetime, optime, reptimes, opstate, opnum) done = HW_TIMER_DONE mode：0 停止，2-启动 cyclonetime：周期时间，us 单位 optime：有效时间，us 单位 reptimes：重复次数 opstate：输出缺省状态，输出口变为非此状态后开始计时 opnum：输出口编号，必须能硬件比较输出的口
适用控制器	4 系列产品支持
例子	以下例程测试环境：ZMC432，固件：20170709（仿真器无法运行此指令）。 为了直观显示波形，周期设置的较大。 例一：不重复输出，周期为 1 RAPIDSTOP(2) WAIT IDLE(0) BASE(0) ATYPE=1 UNITS=100 SPEED=100 ACCEL=500

DPOS=0
TRIGGER
OP(0, OFF)
HW_TIMER(2, 1000000, 500000, 1, OFF, 0) '输出口 0 变为 on 后，硬件定时器触发开始计时，500ms 后切换为 off
OP(0, ON)
运行效果：OP(0)保持输出 500ms 后关闭。

例二：周期调整为 2，输出两次
RAPIDSTOP(2)
WAIT IDLE(0)
BASE(0)
ATYPE=1
UNITS=100
SPEED=100
ACCEL=500
DPOS=0
TRIGGER
OP(0, OFF)
HW_TIMER(2, 1000000, 500000, 2, OFF, 0) '输出口 0 变为 on 后，硬件定时器触发开始计时，500ms 后切换为 off
OP(0, ON)
运行效果：以 1000ms 为周期，输出两个周期，每个周期前 500ms 开启，后半个周期关闭。

相关指令	HW_PSWITCH

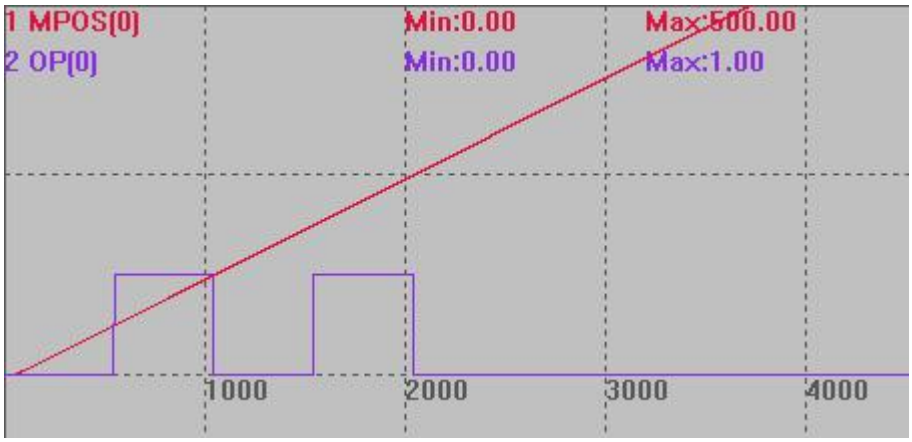
HW_PSWITCH2 -- 总线硬件位置比较输出

类型	轴指令
描述	**总线硬件位置比较输出，必须使用特定的输出口。** 4 系列产品有 4 个比较输出口，可以选择不同的比较输出口，一般为 OUT0/1/2/3 口。 比较主轴带编码器输入时，自动使用编码器位置来触发，可以使用 MOVEOP_DELAY 参数来调整输出准确时刻。 不同的总线驱动器效果可能有差异，也可以通过 MOVEOP_DELAY 参数来调整。 HW_PSWITCH2 与 MOVE_OP 精准使用同样的硬件资源，不建议在同一个通道同时使用，可以在不同的通道同时使用。 每个系统周期内只能比较一次，系统周期通过 SERVO_PERIOD 查询。 TABLE 位置数据在所有比较点完成前不要修改。 脉冲轴和总线轴均支持此指令。 使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。
语法	命令语法：HW_PSWITCH2(mode, [...]) 函数语法：Buff = HW_PSWITCH2([axisnum]) **Mode=1:单轴比较，见例一** HW_PSWITCH2(1,opnum,opstate,tablestart,tableend[,direction]) mode：1-启动比较器 opnum：对应的输出口 opstate：第一个比较点的输出状态 tablestart：第一个比较点绝对坐标所在 TABLE 编号 tableend：最后一个比较点绝对坐标所在 TABLE 编号 direction：第一个点判断方向，0 坐标负向，1 坐标正向，-1 不使用方向 **Mode=2:清除比较点** HW_PSWITCH2(2) mode：2-停止并删除没完成的比较点 **Mode=3:矢量比较方式，见例二** HW_PSWITCH2(3, opnum, opstate, tablestart, tableend) mode：3-启动比较器 opnum：对应的输出口 opstate：第一个比较点的输出状态 tablestart：第一个比较点绝对坐标所在 TABLE 编号 tableend：最后一个比较点绝对坐标所在 TABLE 编号

	使用矢量距离比较时，与 VECTOR_MOVED 进行比较，建议连续运动前设置 VECTOR_MOVED 初始值 **Mode=4:矢量比较方式，单个比较点** HW_PSWITCH2(4, opnum, opstate, vectstart) mode：4-启动比较器 opnum：对应的输出口 opstate：第一个比较点的输出状态 vectstart：比较点绝对坐标 **Mode=5:矢量比较方式，周期脉冲模式，见例三** HW_PSWITCH2(5,opnum, opstate, vectstart, repes, cycledis, ondis) mode：5-启动比较器 opnum：对应的输出口 opstate：第一个比较点的输出状态，认为是有效状态，反之认为无效状态 vectstart：比较点绝对坐标 repes：重复周期，个周期内比较两次，先输出有效状态，再输出无效状态 cycledis：周期距离，每隔这个距离输出 opstate, ondis 后还原为无效状态 ondis：输出有效状态的距离，(cycledis- ondis)为无效状态距离 **Mode=6:矢量比较方式，周期模式，与 HW_TIMER 一起使用，见例四** HW_PSWITCH2(6, opnum, opstate, vectstart, repes, cycledis) mode：6-启动比较器 opnum：对应的输出口 opstate：第一个比较点的输出状态 vectstart：比较点绝对坐标 repes：重复周期，一个周期只比较一次 cycledis：周期距离，每隔这个距离输出一次 HW_PSWITCH2 没有比较完所有点的话，一定要设置 mode 值为 2，通过 HW_PSWITCH2(2)指令停止并删除没有完成的比较点，否则后面此输出通道会工作不正常。
适用控制器	4 系列及以上产品，最新固件
例子	以下例程测试环境：ZMC432，固件：20170709（仿真器无法运行此指令）。 例一 单轴比较 总线轴使能过程省略，参考总线初始化例程。 BASE(0) DPOS=0 MPOS=0 OP(0,OFF) TABLE(0,50,100,150,200)　'比较点坐标设置 **HW_PSWITCH2**(2)　'停止并删除没有完成的比较点 **HW_PSWITCH2**(1, 0, 1, 0, 3,1)　'比较 4 个点，操作输出口 0

TRIGGER '触发示波器
MOVE(500)

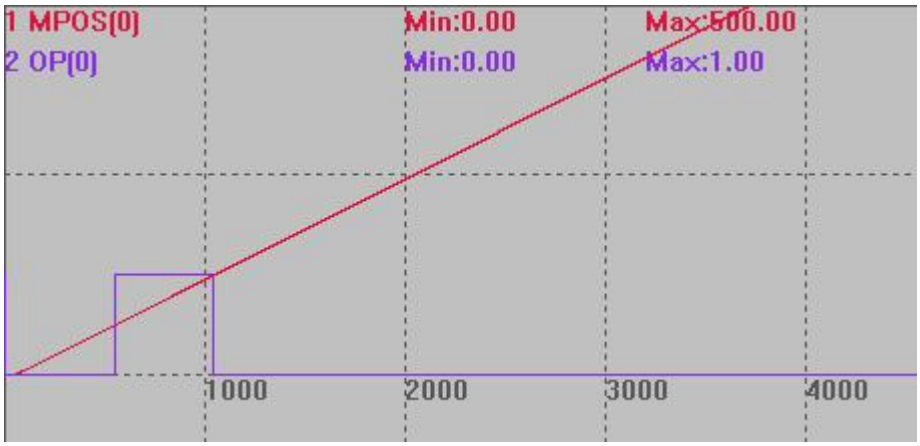
比较输出图
运动到 50 时，关闭 OUT0；位置 100 时，打开 OUT0；位置 150 时，关闭 OUT0；位置 200 时，打开 OUT0。
MPOS(0)垂直刻度 200
OP(0)垂直刻度 2



例二 矢量单轴比较模式
总线轴使能过程省略，参考总线初始化例程。

```
BASE(0)
DPOS=0
MPOS=0
OP(0,OFF)
TABLE(0,50,100,150,200)    '比较点坐标设置
VECTOR_MOVED=100   '设置矢量起始位置
HW_PSWITCH2(2)   '停止并删除没有完成的比较点
HW_PSWITCH2(3, 0, 1, 0, 3)    '比较 4 个点，操作输出口 0
TRIGGER      '触发示波器
MOVE(500)
```

比较输出图
MPOS(0)垂直刻度 200
OP(0)垂直刻度 2

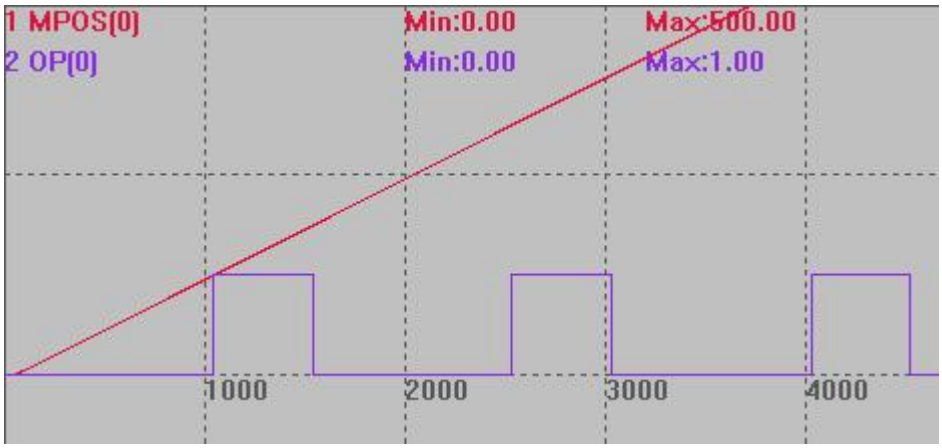


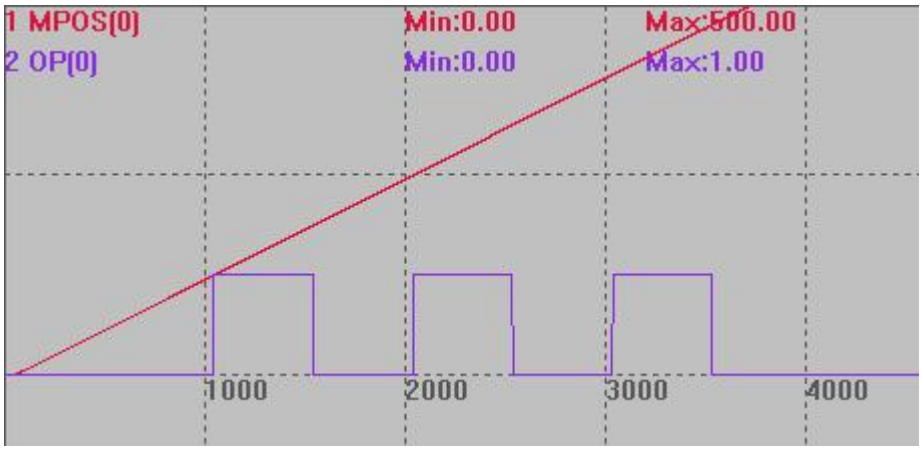
可以看出信号操作直接从例一的位置 100 处开始，矢量比较模式就是将 MPOS(当前位置)+VECTOR_MOVED(之前的位移)与设置位置进行比较。

例三 周期比较模式，距离复位
总线轴使能过程省略，参考总线初始化例程。

```
BASE(0)
DPOS=0
MPOS=0
OP(0,OFF)
TABLE(0,50,100,150,200)     '比较点坐标设置
VECTOR_MOVED=0     '设置矢量起始位置为 0，方便观察
HW_PSWITCH2(2)   '停止并删除没有完成的比较点
HW_PSWITCH2(5,0,1,100,3,150,50)   '位置 100 开始比较，比较 3 次
'周期距离 150，输出有效距离 50
TRIGGER       '触发示波器
MOVE(500)
```

比较输出图
位置 100 开始比较，比较 3 次，一个和周期中前 50 输出有效状态，后 100 输出无效状态。
MPOS(0)垂直刻度 200
OP(0)垂直刻度 2



	例四 周期比较模式，时间复位 总线轴使能过程省略，参考总线初始化例程。 BASE(0) DPOS=0 MPOS=0 OP(0,OFF) TABLE(0,50,100,150,200) '比较点坐标设置 VECTOR_MOVED=0 '设置矢量起始位置为 0，方便观察 **HW_PSWITCH2**(2) '停止并删除没有完成的比较点 **HW_PSWITCH2**(6,0,1,100,3,100) '位置 100 开始比较，比较 3 次周期距离 100，输出有效时间由 HW_TIMER 指令确定 HW_TIMER(2,1000000,500000,1,off,0) '输出变为 on 后 500ms 变为 off TRIGGER MOVE(500) 比较输出图 位置 100 时开始比较，比较 3 次，输出有效信号后 500ms 反转。 MPOS(0)垂直刻度 100 OP(0)垂直刻度 2 
相关指令	PSWITCH，HW_PSWITCH，MOVEOP_DELAY，REG_IMPUTS

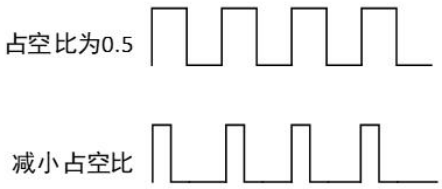
11.4 PMW 控制指令

PWM_FREQ -- PWM 频率

类型	PWM 控制函数
描述	**PWM 频率设置或读取。** PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，不要将频率设为 0，PWM 频率一定要在 PWM 开关之前调整。
语法	PWM_FREQ (index, freq) 或 PWM_FREQ (index)=freq index：编号，从 0 开始 freq：频率，硬件 PWM 1M，软件 PWM 2k

适用控制器	支持 PWM 功能的控制器才支持此函数。
例子	**PWM_FREQ** (0)=1000　'1K 频率 ?**PWM_FREQ** (0)
相关指令	PWM_DUTY，MOVE_PWM

PWM_DUTY -- pwm 占空比

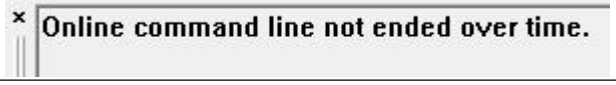
类型	PWM 控制函数
描述	**PWM 占空比设置或读取。** PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，PWM 频率一定要在 PWM 开关之前调整。 占空比指有效电平占整个周期的比例。 一个周期中先输出有效电平，再输出无效电平。 
语法	PWM_DUTY(index, duty)　PWM_DUTY(index)=duty index：编号，从 0 开始 duty：占空比，0-1，当设置 0 的时候，PWM 关闭
适用控制器	支持 PWM 功能的控制器才支持此函数。
例子	**PWM_DUTY**(0)=0.5 ?**PWM_DUTY**(0) 打印结果 0.5
相关指令	PWM_FREQ，MOVE_PWM

第十二章 通讯相关指令

12.1 串口通讯指令

SETCOM -- 串口配置

<table>
<tr><td>类型</td><td>系统指令</td></tr>
<tr><td>描述</td><td>串口的配置。
控制器重新上电后，SETCOM 参数会还原成默认值，所以请在程序开头写 SETCOM 设置。
ZMC00x 系列不支持 MODBUS 主端。</td></tr>
<tr><td>语法</td><td>SETCOM (baudrate,databits,stopbits,parity,port[,mode] [,variable] [,timeout])
baudrate：串口波特率 9600 19200 4800 115200 38400(缺省) 57600 128000 256000
databits：数据位数 8
stopbits：停止位，只能设置 0/1/2
parity：是否校验：
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0（缺省）</td><td>无校验</td></tr>
<tr><td>1</td><td>奇校验</td></tr>
<tr><td>2</td><td>偶校验</td></tr>
</table>
port：串口 PORT 编号 0-1，参见 PORT 描述，不同的控制器不一样
mode：协议
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>RAW 数据模式，无协议，此时可以使用 GET #，PRITNT #</td></tr>
<tr><td>4（缺省）</td><td>MODBUS 从端（16 位整数）</td></tr>
<tr><td>14</td><td>MODBUS 主端（16 位整数)</td></tr>
<tr><td>15</td><td>直接命令执行模式，此时可以直接从串口输入字符串命令(换行符结束)</td></tr>
</table>
variable：寄存器选择，0-VR，1-TABLE，2-系统 MODBUS 寄存器
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>VR，此时一个 VR 映射到一个 MODBUS_REG
VR 是 32 位浮点型，REG 是 16 位整数型，从 VR 传递数据给 REG 会丢失小数部分，当 VR 数据超过正负 15 位时，REG 数据会改变
REG 传递数据给 VR 不会有问题</td></tr>
<tr><td>1</td><td>TABLE，此时一个 TABLE 数据映射到一个 MODBUS_REG（不推荐）
TABLE 是 32 位浮点型，REG 是 16 位整数型，从 TABLE 传递数据给 REG 会丢失小数部分，当 TABLE 数据超过正负 15 位时，REG 数据会改变
REG 传递数据给 TABLE 不会有问题</td></tr>
</table>
</td></tr>
</table>

	<table><tr><td>2（缺省）</td><td>系统 MODBUS 寄存器，此时 VR 与 MODBUS 寄存器是两片独立区间</td></tr><tr><td>3</td><td>VR_INT 模式，此时一个 VR_INT 映射到两个 MODBUS_REG</td></tr></table> timeout：mode=14 时为消息超时时间，毫秒单位，缺省值 1000。 ⚠variable 参数是全局的设置，所有的端口共一个。 ⚠当设置为 VR 或 TABLE 时，没有用到的输出口与输入口会连在一起。 ⚠当设置为 VR 或 TABLE 时，通用输出口会映射到 MODBUS_BIT(0)的位置，输入口会映射到 MODBUS_BIT(1000)的位置，因此此时不要使用 MODBUS_BIT 作为触摸屏的按钮。
适用控制器	通用
例子	例一 mode0 RAW 模式 DIM char1 '定义变量 **SETCOM**(38400,8,1,0,0,0) '配置串口为 RAW 模式 WHILE 1 GET #0, char1 '发送给通道 0 的字符保存到 char1 PRINT char1 '按 ASCII 码打印通道 0 接受的字符 PUTCHAR #0, char1 '将接收的字符再发送回去 WEND 松下 A6 伺服编码器读取参照第十三章简易例程 例二 MODBUS 通讯设置 **SETCOM**(38400,8,1,0,0,4,2) '设置串口 0 为 modbus 从端，波特率 38400 **SETCOM**(38400,8,1,0,1,14,2,1000) 设置串口 1 为 modbus 主端，波特率 38400 具体例程参照 MODBUSM_DES 指令例程 例三 直接字符命令方式 **setcom**(38400, 8,1,0,0,15) '设置串口 0 为直接字符串命令方式 此时在串口调试助手或其他设备，直接发送相关指令可直接操作控制器。  发送前 UNITS=10000 发送后 UNITS=100 一定要换行发送，否则控制器报错 

	例四 寄存器模式 0 VR(0)=0 '初始化 VR(0)和 REG(0)为 0 MODBUS_REG(0)=0 **SETCOM**(38400, 8,1,0,0,4,0) '设置 VR 映射到 MODBUS_REG VR(0)=100.345 '设置 VR(0)=100.345 ?MODBUS_REG(0) '打印结果为 100，VR 已经映射到 REG，但是 REG 是整型，所以小数部分丢失 MODBUS_REG(0)=200 'REG(0)设为 200 ?VR(0) '打印结果为 200，REG 变化也会改变 VR 例五 寄存器模式 2 VR(0)=0 '初始化 VR(0)和 REG(0)为 0 MODBUS_REG(0)=0 **SETCOM**(38400,8,1,0,0,4,2) '设置 VR 与 MODBUS_REG 独立 VR(0)=100.345 '设置 VR(0)=100.345 ?MODBUS_REG(0) '打印结果为 0，VR 不影响 REG MODBUS_REG(0)=200 'REG(0)设为 200 ?VR(0) '打印结果为 100.345，REG 也不影响 VR
相关指令	ADDRESS，PROTOCOL，MODBUSM_DES，PORT

ADDRESS -- 控制器站号

类型	系统参数
描述	**控制器的所有串口的 MODBUS 协议站号 1- 127，缺省=1。**
语法	ADDRESS = value
适用控制器	通用
例子	PRINT ADDRESS '打印出协议站号 打印结果：1
相关指令	SETCOM，PORT，PROTOCOL

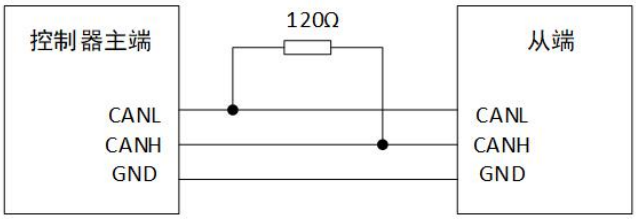
12.2 CAN 通讯指令

CAN -- CAN 通讯

类型	系统指令
描述	**直接通过 CAN 总线收发数据。** 可以实现多个控制器通过 CAN 来通讯，但是同一个 CAN 网络上只能有一个主端(CANIO_ADDRESS=32)。

<table>
<tr><td></td><td>
较新的固件版本才支持这个功能，如不支持请联系厂家。

接线如下图所示：



CANL - CANL
CANH - CANH
CANL 和 CANH 间连接 1 个 120 欧电阻用于阻抗匹配。连接带拨码开关的扩展模块时，将拨码第八位拨为 ON 即表示接入 120 欧电阻，无需额外接电阻。
</td></tr>
<tr><td>语法</td><td>
CAN(channel, function, tablenum)
channel：CAN 通道，0 表示第一个通道，-1 表示缺省通道
function：功能号
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>6</td><td>接收，没有数据时，identifier<0</td></tr>
<tr><td>7</td><td>发送</td></tr>
<tr><td>16
需要升级固件</td><td>带扩展支持接收，没有数据时，identifier<0</td></tr>
<tr><td>17
需要升级固件</td><td>发送扩展数据，普通数据使用 7 发送</td></tr>
</table>
tablenum：数据存储的 TABLE 位置

6、7 模式时依次存储：
identifier：Can 通讯对象(Cob-Id)，11 位，前 4 位是功能码，后 7 位是节点 ID，ZCAN 使用的高位数据预留，建议 0-511。接收时如果小于 0 则表明没有收到数据。
bytes：数据区域字节数，最多为 8。
data：数据区域，字节（0-FF）。

16、17 模式时依次存储：
identifier：Can 通讯对象(Cob-Id)，11 位，前 4 位是功能码，后 7 位是节点 ID，ZCAN 使用的高位数据预留，建议 0-511。接收时如果小于 0 则表明没有收到数据。
Identifier extend：扩展 id，增加高 11 位与低 18 位，共同组成 CAN 的 29 位 ID，不存在填-1。
bytes：数据区域字节数，最多为 8。
data：数据区域，字节（0-FF）。

29 位扩展帧 ID 从左边非 0 开始拆分为高 11 位和低 18 位，例子如下：
扩展帧 ID 为 18EF1300
对应二进制为 0001 1000 1110 1111 0001 0011 0000 0000
高 11 位为 11000111011，对应 10 进制 1595
</td></tr>
</table>

	低 18 位为 110001001100000000，对应 10 进制 201472 所以使用 CAN17 模式时，identifier=1595，Identifier extend=201472
适用控制器	通用
例子	例一 '发送端： TABLE(0,1,8,1,2,3,4,5,6,7,8) '发送 cobid=1，8 个字节，依次为 1-8 **CAN**(0,7,0) '发送 '接收端： CANIO_ADDRESS=1 '设置为非主控，此参数设置一次即可 **CAN**(0,6,0) '接受 ?TABLE(0) 例二 '发送端 TABLE(0,1,10,8,1,2,3,4,5,6,7,8) '发送 cobid=1，扩展 id10，8 个字节，依次'为 1-8 **CAN**(0,17,0) '发送 '接收端： CANIO_ADDRESS=1 '设置为非主控，此参数设置一次即可 **CAN**(0,16,0) '接收 ?TABLE(0)
相关指令	CANIO_ADDRESS，CANIO_STATUS，CANIO_ENABLE

CANIO_ADDRESS -- CAN 通讯设置

类型	系统参数
描述	**控制器上 CAN 的 CANID 和速度的设置。** 扩展板的 CAN 速度通过拨码开关设置。 自动存储到 FLASH，重启后生效。 低 8 位（位 0-7）表示 CANID，范围 0-32，缺省 32，32 表示为主控制器。 高 8 位（位 8-15）表示 CAN 速度。 高 8 位值 | CAN 速度 0 | 500 KBPS（缺省值） 1 | 250KBPS 2 | 125KBPS 3 | 1 MBPS 一个网络中，不要配置多个主控制器。 CAN 地址和速度设置必须重启后生效。
语法	CANIO_ADDRESS = value
适用控制器	通用

例子	CANIO_ADDRESS = 32 '设置主端，CAN 波特率 500KBPS CANIO_ADDRESS = 32+ 256 '设置主端，CAN 波特率为 250KBPS CANIO_ADDRESS = 32+ 512 '设置主端，CAN 波特率为 125KBPS CANIO_ADDRESS = 32+ 768 '设置主端，CAN 波特率为 1MBPS CANIO_ADDRESS = 1 '设置 CANID 为 1，此时为从端，500 KBPS，不能连接 IO 板 CANIO_ADDRESS = 1 +256 '设置 CANID 为 1，此时为从端，250 KBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 1 +512 '设置 CANID 为 1，此时为从端，125 KBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 1 +768 '设置 CANID 为 1，此时为从端，1 MBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 3 '设置 CANID 为 3，此时为从端，500 KBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 8 +256 '设置 CAN ID 为 8，此时为从端，250 KBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 16 +512 '设置 CAN ID 为 16，此时为从端，125 KBPS，作为 ZCAN 从站使用 CANIO_ADDRESS = 31 +768 '设置 CAN ID 为 31，此时为从端，1MBPS，作为 ZCAN 从站使用 具体使用参考第十七章的 CAN 通讯例子
相关指令	CANIO_ENABLE，CAN，CANIO_STATUS

CANIO_ENABLE -- CAN 使能

类型	系统参数
描述	**使能或禁止内部 CAN 主端功能。** 当 CANIO_ADDRESS 设置 32 时，缺省是使能的。
语法	CANIO_ENABLE = ON/OFF
适用控制器	通用
例子	CANIO_ADDRESS = 32 '设置主端，CAN 波特率 500KBPS CANIO_ENABLE = ON '打开 CAN 主端功能 CANIO_ENABLE = OFF '关闭 CAN 主端功能
相关指令	CANIO_ADDRESS

CANIO_STATUS -- CAN 扩展板状态

类型	系统状态

描述	**获取当前的 IO 板的状态，返回 ON/OFF。** 20140325 以上的固件版本支持
语法	CANIO_STATUS(cardnum) cardnum：扩展模块的拨码 ID
适用控制器	通用
例子	例一 ?***CANIO_STATUS** '输出所有 IO 板的状态 例二 IF **CANIO_STATUS**(1) =0 THEN '判断 IO 板的连接状态 PRINT "扩展模块 1 没有连接好" ENDIF
相关指令	CAN，CANIO_ENABLE，CANIO_INFO

CANIO_INFO -- CAN 扩展板信息

类型	系统状态
描述	**读取当前的 IO 板的信息，返回参数值。** 4 系列控制器 170715 以上固件版本支持。
语法	CANIO_INFO(canid, isel [, moduleid]) canid：扩展模块的拨码 ID isel：读取的参数

0	正运动
1	DEVICE，设备编号
2	
3	
4	
IO 的个数	描述
10	IN 个数，整体的
11	OP 个数
12	AIN 个数
13	AOUT 个数
增加	
16	模块数
17	子模块的类型号，必须后面带 moduleid id
预留	
20	子模块输入
21	子模块输出
22	子模块 AIN
23	子模块 AOUT

moduleid：ZMIO 子模块的信息，部分 isel 参数选择需要填这个参数

适用控制器	通用

例子	例一 ?**CANIO_INFO**(1,1) '打印 ID 是 1 的扩展模块的设备编号
相关指令	CAN，CANIO_ENABLE，CANIO_STATUS

12.3 自定义通讯指令

GET # -- 读取字符

类型	系统指令
描述	**从 RAW 方式通讯或自定义网口通讯的通道里面读取一个字节，存入一个变量。**
语法	语法 1：GET #PORT, VARIABLE 语法 2：GET #PORT, ARRAY[(startindex)] [,maxchares] 语法 3：charesget = GET #PORT, VARIABLE 语法 4：charesget = GET #PORT, ARRAY[(startindex)] [,maxchares] port：通道号 variable：存放的变量名 startindex：存放数组的起始地址 maxchares：存放的最多数量 语法 1、2 没有读取到会阻塞，这个函数一般在多任务里面进行调用。 语法 3、4 会返回读取到的字节数。 20150522 版本以前只支持语法 1。 UDP 接收必须使用语法 4，采用数组来接收，数组长度不要比一次的 UDP 包长度小。 UDP 每次都是读取一整个包，如果数组长度不够，多余的会丢弃掉。 .UDP_SERVER 模式时，每收到一个包，PORT_TARGET 自动变为包的发送方，因此可以同时接收多个从端的数据。
适用控制器	通用
例子	例一 DIM VAR1 SETCOM(38400,8,1,0,0,0) '开启 RAW 方式 **GET** #0, VAR1 '从通道 0 读取数据 PRINT VAR1 '打印读取数据 例二 DIM ARRAY1(101) SETCOM(38400,8,1,0,0,0) '开启 RAW 方式 CHARES = **GET** #0, ARRAY1, 100 '从通道 0 最多读取 100 个字符 If CHARES > 0 THEN ARRAY1(CHARES) = 0 '设置结束 0 PRINT ARRAY1 '字符串打印出来

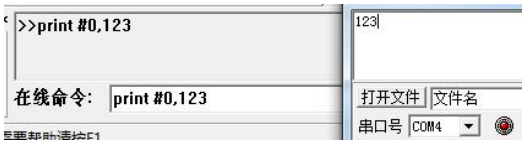
	ENDIF
相关指令	PORT，，PRINT #

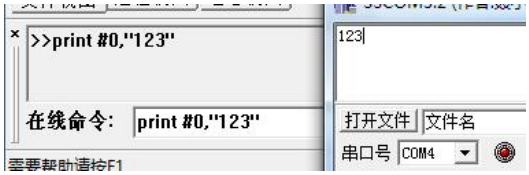
OPEN # -- 打开自定义网口通讯

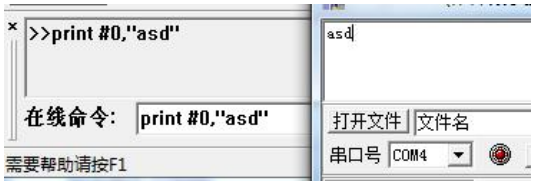
类型	系统指令
描述	**开启自定义的以太网通讯。** 新固件版本提供此功能。
语法	OPEN #PORT, "mode", portnum [, ipaddress] port：通讯通道，参见 PORT 描述，选择自定义网络通道 mode：讯主从，“TCP_CLIENT”-从，“TCP_SERVER”－主，“UDP_CLIENT”-UDP 从，“UDP_SERVER”－UDP 主 portnum：CP 或 UDP 端口号，主端为本地端口号，从端为对方端口号 ipaddress：对方 IP 地址，字符串，从端的时候要提供 UDP_SERVER 必须先接收对方的数据，才能发送回数据(除非用 PORT_TARGET 先强制指定对方)。 UDP_CLIENT 本地端口号随机，必须先发送给对方，对方才能知道端口号，此模式时不是指定对方的包会丢弃掉。 UDP 自定义通讯需要 4 系列控制器 20170628 以上固件版本；XPLC 系列控制器 20170702 以上固件版本。
适用控制器	通用
例子	例一 **OPEN** #11, "TCP_SERVER", 10 '主端设置 **OPEN** #10, "TCP_CLIENT", 10,"192.168.1.112" '从端设置 例二 OPEN #10, "UDP_SERVER", 1000 'UDP 主端设置 OPEN #11, "UDP_CLIENT", 60000, "192.168.0.120" 'UDP 从端设置 **详细例程参考**自定义网口通讯
相关指令	PORT，PORT_TARGET，SETCOM，PRINT #

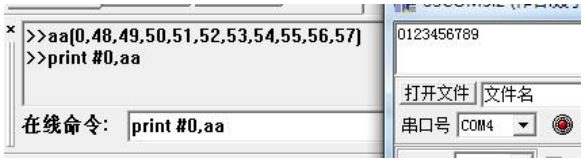
PRINT # -- 输出字符串

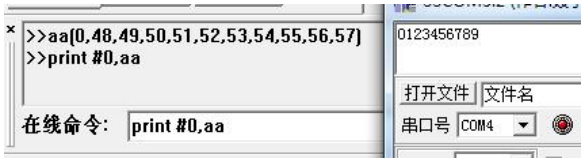
类型	系统指令
描述	**从 RAW 方式通讯或自定义网口通讯的通道口里面输出字符串，碰到 0 结束。** 每次调用都是一个 UDP 包发送。 PRINT #直接发送为字符串(""符号自动去除)，不需要 ASCII 转码处理，一次只能发一个数据，如下所示：

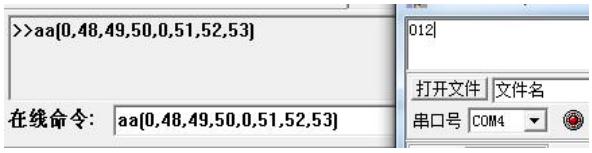
<table>
<tr><td></td><td>






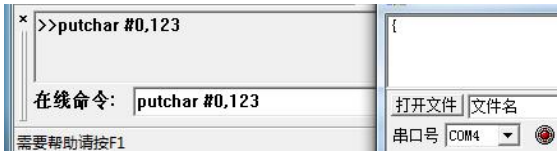
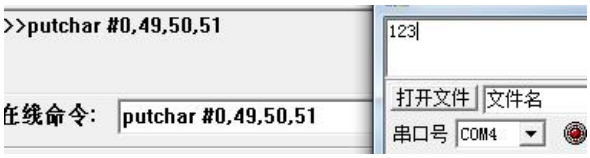


PRINT #发送数组为 ASCII 码，遇 0 停止，如下所示：




</td></tr>
<tr><td>语法</td><td>PRINT #PORT, "字符串"
port：通道号</td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>DIM VAR(10) '定义数组
VAR = "AAAA" '数组赋值
SETCOM(38400,8,1,0,0,0)'开启 RAW 方式
PRINT #0,VAR '从通道 0 输出数组数据</td></tr>
<tr><td>相关指令</td><td>PUTCHAR #，GET #，PORT，SETCOM</td></tr>
</table>

PUTCHAR # -- 输出字符

类型	系统指令
描述	**从 RAW 方式通讯或自定义网口通讯的通道口输出字符，数组可以代表多个字符。** 每次调用都是一个 UDP 包发送。 PUTCHAR #发送数据为 ASCII 码，多个数据逗号隔开，不能直接发送字符串，如下所示：    PUTCHAR #发送数组为 ASCII 码，遇 0 输出为 0，会将整个数组完全发送，所以要确保数组数据正确，如下所示：  
语法	PUTCHAR #PORT, 字符 port：通道号 PUTCHAR #PORT, ARRAY(INDEX, NUMES) port：通道号 index：开始输出的位置 numes：输出的字节个数，二进制方式 20170503 以后固件支持打印出部分数据
适用控制器	通用
例子	DIM VAR1, ARRAY1(10) '定义数组 VAR1 = $FE '数组 1 赋值 ARRAY1 = "ABCDEFGHIJ" '数组 2 赋值 SETCOM(38400,8,1,0,0,0) '开启 RAW 方式

	PUTCHAR #0, VAR1 '从通道 0 输出数组 1 数据 **PUTCHAR** #0, ARRAY1 '从通道 0 输出数组 2 数据
相关指令	PORT，SETCOM，GET #

PORT_TARGET -- IP 和端口号配置

类型	字符串指令
描述	**配置通讯对方的 IP 地址和端口号。** 4 系列控制器 20170628 以上固件版本，XPLC 系列控制器 20170702 以上固件版本支持。
语法	命令语法： PORT_TARGET(port))="ipaddress:portnum"或 PORT_TARGET(port))="ipaddress" port：通道号 ipaddress：对方 IP 地址 portnum：端口号 返回语法：VAR=PORT_TARGET(port)) 返回 IP 地址字符串。
适用控制器	通用
例子	PORT_TARGET="192.168.0.12:502" PORT_TARGET(port))="192.168.0.12" ?PORT_TARGET(port) DIM IPSTRING(100) IPSTRING = PORT_TARGET(port) ?IPSTRING
相关指令	OPEN #

12.4 打印输出指令

PRINT -- 打印信息

类型	打印输出函数
描述	**打印信息到 PC 端 ZDevelop 软件的输出窗口。** **别名：?** 通过*参数名可以打印参数值；*特殊字符串可以打印一些特殊的信息。
语法	PRINT expression, "string" 或 ? expression,"string" Expression：有效表达式 *SET：打印所有参数值 *TASK：打印任务信息 任务正常时只打印任务状态 任务出错时还会打印出错误任务号，具体错误行 *MAX：打印所有规格参数

	*FILE：打印程序文件信息 *SETCOM：打印当前串口的配置信息 *BASE：打印当前任务的 BASE 列表（140123 以后版本支持） *数组名：打印数组的所有元素，数组长度不能太长 *参数名：打印一个所有轴的单个参数 ?*ETHERCAT：打印 EtherCAT 总线连接设置状态 ?*RTEX：打印 Rtex 总线连接设置状态 ?*FRAME：打印机械手参数，需要 161022 及以上固件支持 ?*SLOT：打印出控制器槽位口信息（RTEX 口，EtherCAT 口） ?*PORT：打印所有 PORT 通讯口
适用控制器	通用
例子	例一 在线命令输入 >>**PRINT** 1+2 输出：3 例二 在线命令输入 >>**PRINT** *task　　'打印所有任务状态 Task:0 Running. file:"hmi.bas" line:280: Task:1 Stopped. Task:2 Stopped. Task:3 Stopped. Task:4 Stopped. Task:5 Stopped. Task:6 Stopped. 例三 在线命令输入 >> ?*mpos 输出：21872.400 0 0 0 0 0 0 0 0 0 0 0 0
相关指令	TRACE

ERRSWITCH -- 信息输出设置

类型	系统参数
描述	**调试输出开关，控制 TRACE WARN ERROR 调试输出语句是否真的输出信息。**
语法	ERRSWITCH=switch switch：调试输出的开关

值	描述
0	TRACE WARN ERROR 指令全部不输出
1	只输出 ERROR 指令

		2	输出 WARN ERROR 指令
		3	TRACE WARN ERROR 指令全部输出
		4	TRACE WARN ERROR 指令全部输出，以及运动指令监控
适用控制器	通用		
例子	例一 **ERRSWITCH** = 3 'TRACE 信息都会输出 例二 **ERRSWITCH** = 0 'TRACE DPOS(0) 'trace 打印出轴 0 的 dpos ?DPOS(0) 'print 打印，结果，0 例三 **ERRSWITCH** = 4 >>MOVE(111) MOVE(111)AXIS(0)TASK(23)		'打印全输出，此时 WARN（报警）、ERROR（错误）、 '不输出，控制器错误信息和报警信息都不输出 '此时无法用 '可以用 '输出所有信息，以及运动指令监控 '打印当前操作的轴及运动的任务
相关指令	TRACE，WARN，ERROR		

TRACE -- 打印信息 2

类型	打印输出函数
描述	**打印信息到 PC 端 ZDEVELOP 软件的输出窗口。** 受 ERRSWITCH 开关的控制。
语法	同 PRINT
适用控制器	通用
例子	ERRSWITCH = 3 'trace 打印全输出 DPOS(0) =0 **TRACE** "DPOS(0) =" DPOS(0) 打印结果 0
相关指令	ERRSWITCH ，PRINT，WARN，ERROR

WARN -- 警告信息

类型	打印输出函数
描述	**程序自动打印报警信息到 PC 端 ZDEVELOP 软件的输出窗口。** 也可手动打印信息。 受 ERRSWITCH 开关的控制。
语法	程序报警时自动打印。 手动打印同 PRINT。
适用控制器	通用

例子	参考 TRACE 例子
相关指令	ERRSWITCH ，PRINT，TRACE，ERROR

ERROR -- 错误信息

类型	打印输出函数
描述	**程序自动打印错误信息到 PC 端 ZDEVELOP 软件的输出窗口。** 也可手动打印信息。 受 ERRSWITCH 开关的控制。
语法	程序错误时自动打印。 手动打印同 PRINT。
适用控制器	通用
例子	参考 TRACE 例子
相关指令	ERRSWITCH ，PRINT，TRACE，WARN

12.5 通道参数指令

PORT -- 通道编号

类型	通道修正辅助指令
描述	**当访问通道参数时可以指定其他的通道。** 其他指令需要选择通道时查看。
语法	PORT (portnum) portnum：通道号 不同型号的控制器支持的通道数不同。在线命令?*port 查看，见例程。 ZMC00x 系列

通道号	协议
0	串口 A
1	串口 B

ZMC1-2xx 系列，支持同时两个以太网连接，端口号为 502，支持触摸屏的 MODBUS-TCP 协议连接。

通道号	协议
0	RS232 串口
1	RS485 串口
2	以太网接口，连接 1
3	以太网接口，连接 2
10	自定义网络通讯通道 1
11	自定义网络通讯通道 2

ZMC3xx 系列：带 3 个串口。

通道号	协议
0	RS232 串口
1	RS485 串口
2	RS422 串口
3	以太网接口，连接 1
4	以太网接口，连接 2
10	自定义网络通讯通道 1
11	自定义网络通讯通道 2

ZMC4xx 系列：带 2 个串口。

通道号	协议
0	RS232 串口
1	RS485 串口
2	以太网接口，连接 1
3	以太网接口，连接 2
10	自定义网络通讯通道 1
11	自定义网络通讯通道 2
20	网络控制器互联通道，不支持 PORT_STATUS 查询。

ECI 系列：只有一个串口。

通道号	协议
0	RS232 串口
1	以太网接口，连接 1
2	以太网接口，连接 2

适用控制器

通用

例程

查看控制器通道

```
>>?*PORT
Port:0-COM.
Port:1-COM.
Port:2-ETH.
Port:3-ETH.
Port:4-ETH.
Port:5-ETH.
Port:6-ETH.
Port:7-ETH.
Port:10-ECUSTOM.
Port:11-ECUSTOM.
Port:12-ECUSTOM.
Port:13-ECUSTOM.
Port:14-ECUSTOM.
Port:15-ECUSTOM.
Port:20-CONNECT.
```

其中 COM 为串口通道，ETH 为网口通讯通道，ECUSTOM 为自定义网口通讯通道，

	CONNNECT 为控制器互联通道。
相关指令	FILE_PORT，PORT_STATUS，PROTOCOL

PORT_STATUS -- 通道状态

类型	通道参数，只读
描述	**返回当前通道的状态，串口总是返回 1。** MODBUS_TCP 链接上以后，PORT_STATUS 会变为 1。 当对应链接已经作为从端使用时，控制器自动搜索没有使用的 PORT 作为 MODBUS_TCP 主端链接，此时程序无法确定实际使用的是哪个 PORT。 新固件版本提供此功能。
语法	VAR1 = PORT_STATUS(port) port：通道号 值 | 协议 0 | 没有连接 1 | 有连接使用
适用控制器	通用
例子	?**PORT_STATUS**(0)　　'返回 232 串口通道状态，结果，1
相关指令	PORT，SETCOM，ADDRESS

FILE_PORT -- 当前通道文件号

类型	通道参数
描述	**返回或设置当前通道的缺省文件号。** 设定后可以通过在线命令来访问对应文件的文件模块变量或数组。 ZDevelop 集成开发环境会自动使用此参数，不要在使用 ZDevelop 时修改此参数。
语法	VAR1 = FILE_PORT，FILE_PORT = filenum
适用控制器	通用
例子	>>**FILE_PORT** = 0　　'当前连接的通道的 FILE_PORT 参数
相关指令	PORT

PROTOCOL -- 通道通讯协议

类型	通道参数，只读
描述	**返回当前通道的通讯协议。**
语法	VAR1 = PROTOCOL(port)

	port：通道号 返回值 值 | 协议 0 | RAW 数据格式，无协议 3 | MODBUS 协议，控制器为从端（缺省） 14 | MODBUS 协议，控制器为主端 15 | 直接命令执行方式
适用控制器	通用
相关指令	PORT，SETCOM，ADDRESS

12.6 MODBUS 通讯指令

MODBUS_BIT -- 位寄存器

类型	MODBUS 位寄存器
描述	**修改或者读取 BIT 寄存器，布尔型，触摸屏一般叫 0x 寄存器。** 另：特殊的 modbus 0x 寄存器，通过这些寄存器可以直接从触摸屏读取和设置 IO，注意读取的 IO 是原始的状态，INVERT_IN 不起作用。 寄存器（zero based） | 意义 10000- | 输入 IN,每个 IN 占 1 个寄存器 20000- | 输出 OP,每个 OP 占用 1 个寄存器 30000- | PLC 的 S 寄存器，每个占用 1 个寄存器
语法	MODBUS_BIT(first,[last]) = value first：位寄存器编号，编号从 0 开始 last：位寄存器编号，编号从 0 开始
适用控制器	通用
例子	DIM VAR **MODBUS_BIT**(100) = 1 'bit100 赋值 1 VAR = **MODBUS_BIT**(100) '把 bit100 的值赋值到变量 VAR

MODBUS_IEEE -- 字寄存器-32 位浮点型

类型	MODBUS 字寄存器
描述	**修改或者读取字寄存器，32 位浮点型。触摸屏一般叫 4x 寄存器。** ZMOTION 运动控制器专门具备 MODBUS 字寄存器，会占用两个寄存器的地址。 字寄存器间的关系查看“MODBUS 寄存器”。

	控制器缺省使用系统 MODBUS 字寄存器，要修改时请设置 SETCOM 的第 7 个参数。 另：特殊的 MODBUS 4x 寄存器，通过这些寄存器可以直接从触摸屏读取一些状态。 寄存器（zero based） | 类型 | 意义 10000- | IEEE | DPOS 读取，每个轴占两个寄存器 11000- | IEEE | MPOS 读取，每个轴占两个寄存器 12000- | IEEE | VP_SPEED 读取，每个轴占两个寄存器
语法	MODBUS_IEEE (regnum) = value regnum：寄存器编号，编号从 0 开始
适用控制器	通用
例子	DIM VAR **MODBUS_IEEE**(100) = 100.10 'ieee100 赋值 100.10 VAR = **MODEBUS_IEEE**(100) 'ieee100 赋值给变量 VAR
相关指令	MODBUS_REG，MODBUS_LONG，MODBUS_STRING

MODBUS_LONG -- 字寄存器-32 位整型

类型	MODBUS 字寄存器
描述	**修改或者读取字寄存器，32 位整型。触摸屏一般叫 4x 寄存器。** ZMOTION 运动控制器专门具备 MODBUS 字寄存器，会占用两个寄存器的地址。 字寄存器间的关系查看“MODBUS 寄存器”。 控制器缺省使用系统 MODBUS 字寄存器，要修改时请设置 SETCOM 的第 7 个参数。
语法	MODBUS_LONG(regnum) = value regnum：寄存器编号，编号从 0 开始
适用控制器	通用
例子	DIM VAR **MODBUS_LONG**(100) = 100 'long100 赋值 100 VAR = **MODEBUS_LONG**(100) 'long100 赋值给 VAR
相关指令	MODBUS_REG，MODBUS_IEEE，MODBUS_STRING

MODBUS_REG -- 字寄存器-16 位整型

类型	MODBUS 字寄存器
描述	**修改或者读取字寄存器，16 位整型，触摸屏一般叫 4x 寄存器。** ZMOTION 运动控制器专门具备 MODBUS 字寄存器。 不同型号控制器的字寄存器个数有区别。 字寄存器间的关系查看“MODBUS 寄存器”。

	控制器缺省使用系统 MODBUS 字寄存器，要修改时请设置 SETCOM 的第 7 个参数。 另：特殊的 modbus 4x 寄存器，通过这些寄存器可以直接从触摸屏读取一些状态。 寄存器（zero based） 类型 意义 13000- 16 DA 输出 14000- 16 AD 读取
语法	MODBUS_REG(regnum) = value regnum：寄存器编号，编号从 0 开始
适用控制器	通用
例子	DIM VAR **MODBUS_REG**(100) = 100 'reg100 赋值 100 VAR = **MODBUS_REG**(100) 'reg100 赋给变量 VAR
相关指令	MODBUS_IEEE，MODBUS_LONG，MODBUS_STRING

MODBUS_STRING -- 字寄存器-字节

类型	MODBUS 字寄存器，字符串函数
描述	**读取 MODBUS 寄存器区域的字符串，按字节来读取。触摸屏一般叫 4x 寄存器。** 字寄存器间的关系查看“MODBUS 寄存器”。
语法	MODBUS_STRING(index, chares) index：起始的 MODBUS 寄存器编号，编号从 0 开始。不同型号控制器的字寄存器个数有区别 chares：读取的字符总数
适用控制器	通用
例子	DIM ARR(8) **MODBUS_STRING** (0, 8) = "abc" 'string0 开始保存字符串，共 8 字节 print **MODBUS_STRING**(0, 8) '打印保存的字符串，打印结果，abc ARR = **MODBUS_STRING**(0,8) '字符换赋值给数组
相关指令	MODBUS_REG，MODBUS_LONG，MODBUS_IEEE

MODBUSM_DES -- modbus 通讯连接

类型	通讯指令
描述	**设置或读取 MODBUS 主端的对方。** 通讯等待时只会阻塞当前这个任务，不影响其他任务。 使用 485 串口与多个设备通讯时，建立通讯连接可加入等待或延时，等上一个设备连接成功再连接下一个设备，防止出现通讯失败的情况。
语法	MODBUSM_DES (address[,port],[timer],[resendset]) ADDRESS1 = MODBUSM_DES([port]) address：对端的 modbus 协议站号 port：当前 modbus 主通讯的 port 号

	timer：消息超时时间设置，缺省 1000ms。 resendset：超时消息重发设置，0-不重发，1-SEND 指令重发，2-SEND 与 MODBUSM 指令都重发。 MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。 SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
适用控制器	通用

例子	例一 多主端-多从端通讯
	SETCOM(38400,8,1,0,0,14,2,1000) '设置串口 0 为 modbus 主端，通讯等待 1s
	SETCOM(38400,8,1,0,1,14,2,1000) '设置串口 1 为 modbus 主端，通讯等待 1s
	
	WHILE 1
	IF IN(0)=1 THEN 'IN0 高电平时使用串口 0
	IF IN(1)=1 THEN
	MODBUSM_DES(1,0) 'IN1 高电平时与从端站号 1 通讯
	MODBUSM_REGSET(0,10,0) '本地寄存器复制到对端
	MODBUSM_REGGET(20,10,20) '对端寄存器复制到本地
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	
	ELSE
	MODBUSM_DES(2,0) 'IN1 低电平时与从端站号 2 通讯
	MODBUSM_REGSET(30,10,30) '本地寄存器复制到对端
	MODBUSM_REGGET(40,10,40) '对端寄存器复制到本地
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	ENDIF
	?"通道 0 状态=", MODBUSM_STATE '打印通讯状态
	
	ELSE 'IN0 低电平时使用串口 1
	IF IN(1)=1 THEN
	MODBUSM_DES(1,1) 'IN1 高电平时与从端站号 1 通讯
	MODBUSM_REGSET(50,10,50) '本地寄存器复制到对端
	MODBUSM_REGGET(60,10,60) '对端寄存器复制到本地
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	ELSE
	MODBUSM_DES(2,1) 'IN1 低电平时与从端站号 2 通讯
	MODBUSM_REGSET(70,10,70) '本地寄存器复制到对端
	MODBUSM_REGGET(80,10,80) '对端寄存器复制到本地
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	ENDIF
	?"通道 1 状态=", MODBUSM_STATE '打印通讯状态
	ENDIF
	WEND

相关指令	SETCOM，PROTOCOL，PORT，MODBUSM_DES2

MODBUSM_DES2 -- 控制器间网口通讯

类型	通讯指令
描述	**控制器间网口通讯，也可作为 MODBUS_TCP 主端通讯。** ?*PORT 可以显示出当前可用的网络链接通道。 >>?*port Port:0-COM. Port:1-COM. Port:2-ETH. Port:3-ETH. Port:4-ETH. Port:5-ETH. Port:6-ETH. Port:7-ETH. Port:10-ECUSTOM. Port:11-ECUSTOM. Port:12-ECUSTOM. Port:13-ECUSTOM. Port:14-ECUSTOM. Port:15-ECUSTOM. Port:20-CONNECT. 作为 MODBUS_TCP 主端时： 选择一个 ETH 模式的 PORT 口作为 MODBUS_TCP 链接通道(不建议用第一个和最后一个)。 当选择的通道已经作为从端被占用时，控制器自动选择一个没有占用的 ETH 模式的 PORT 口作为 MODBUS_TCP 主端链接。 作为控制器互联使用时： 选择 CONNECT 模式的 PORT 口作为控制器互联通道。
语法	MODBUSM_DES2 (id,port,"desipaddress",[timer],[resendset]) id：对方控制器的 MODBUS 从端 ID，缺省 1 port：支持两种模式，?*PORT 确认通道号及模式 ETH 时，作为 MODBUS_TCP 主端通道 CONNECT 时，做为控制器互联通道 desipaddress：字符串， 对方控制器的 IP 地址 timer：消息超时时间设置，缺省 1000ms resendset：超时消息重发设置，0-不重发，1-SEND 指令超时重发，2-SEND 与 MODBUSM 指令都超时重发。 MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。 SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
适用控制器	4 系列产品, 20170117 及以上固件版本支持
例子	例一 MODBUS 主端通讯连接建立 MODBUS_TCP 主端通讯不用考虑丢包。

	MODBUSM_DES2(1,4,"192.168.0.12") '按站号和 IP 与从端通讯使用控制器通道 4，?*port 确认
	WHILE 1
	LASTTICK=TICKS
	FOR i =0 TO 9999
	MODBUS_REG(0) = i
	MODBUSM_REGSET(0,10,0) '设置对端寄存器
	MODBUS_REG(0) = 99
	MODBUSM_REGGET(0,10,0) '读取对端寄存器
	
	IF MODBUS_REG(0) <> I THEN
	?"REG(0)=" MODBUS_REG(0),"STATE=" MODBUSM_STATE
	'打印在第几次通讯时错误
	ENDIF
	NEXT
	?LASTTICK-TICKS '打印通讯时间
	WEND
	END
	
	例二 控制器间通讯连接建立
	网络环境不好时， 互连通讯有很小的丢包可能。
	MODBUSM_DES2($fe,20,"192.168.0.25" ,10) '控制器从端站号为 fe
	'控制器主端通道号为 20,?*port 确认设置 10ms 超时时间
	MODBUSM_REGSET(0,10,0) '本地寄存器复制到对端
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	IF MODBUSM_STATE<>0 THEN MODBUSM_REGSET(0,10,0) '出错重发一次
	MODBUSM_REGGET(20,10,20) '对端寄存器复制到本地
	WAIT UNTIL MODBUSM_STATE <> 1 '等待消息结束
	IF MODBUSM_STATE<>0 THEN MODBUSM_REGGET(20,10,20) '出错重发一次
	END

相关指令	ADDRESS，PORT

MODBUSM_STATE -- modbus 通讯状态

类型	通讯状态
描述	**MODBUS 主通讯状。** 值 描述 0 状态正常 1 等待应答中 2 等待超时 3 应答错误
语法	VAR1 = MODBUSM_STATE 当前 modbus 主端通讯状态

适用控制器	通用

例子	SETCOM(38400,8,2,0,1,14,2,1000) '485 口为 modbus 主端，消息超时时间'1s
	MODBUSM_DES(1,1) '与对端站号 1 通讯
	MODBUSM_REGGET(0,10,0) '获取对端寄存器值
	WAIT UNTIL **MODBUSM_STATE** <> 1 '等待消息结束
	'最多等待 1s，为消息超时时间，获取消息或超时后变为相应值
	?**MODBUSM_STATE** '打印通讯结果
	IF **MODBUSM_STATE**=0 THEN
	?"通讯正常"
	ELSEIF **MODBUSM_STATE**=2 THEN
	?"通讯超时"
	ELSEIF **MODBUSM_STATE**=3 THEN
	?"通讯错误"
	ENDIF

相关指令	PROTOCOL，PORT，SETCOM

MODBUSM_REGSET -- 写对端保持寄存器

类型	通讯指令
描述	**把本地 MODBUS 保存寄存器设置到对端。** **对应标准协议功能码 06 或 16，写保持寄存器。**
语法	MODBUSM_REGSET (startreg, num, local_reg) startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	参考 MODBUSM_REGGET 例一
相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

MODBUSM_REGGET -- 读对端保持寄存器

类型	通讯指令
描述	**把对端的 MODBUS 保存寄存器复制到本地。** **对应标准协议功能码 03，读保持寄存器。**
语法	MODBUSM_REGGET (startreg, num, local_reg) startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	台达绝对值编码器读取

	GLOBAL DIM flag_abs '编码器读取正确标志
	flag_abs = 0
	GLOBAL DIM total_pul '读到个总脉冲个数
	SETCOM(38400,8,2,0,1,14) '设置 485 口为 MODBUS 主端，波特率 38400
	MODBUSM_DES(1,1) '设置 485 端口，对方站号 1 P3-00
	MODBUS_LONG(300) = 2 '用 300，301 传输数据
	MODBUSM_REGSET(98,2,300) '设置 P0-49 = 2，更新参数
	MODBUSM_REGGET(98,2,300)
	TICKS = 1000
	
	WHILE (MODBUS_LONG(300) AND TICKS > 0) '等到 P0-49 变成 0 或 1 秒超时，即更新完成或者不成功
	MODBUSM_REGGET(98,2,300)
	WEND
	
	IF TICKS < 0 THEN
	PRINT "伺服更新不成功"
	flag_abs = 1
	RETURN
	ENDIF
	
	MODBUSM_REGGET(100,6,310)
	IF MODBUS_LONG(310) = 0 THEN '编码器正常
	flag_abs = 0
	total_pul = MODBUS_LONG(314)
	ELSE
	PRINT "编码器出错"
	flag_abs = 2
	ENDIF
	
	IF flag_abs = 0 THEN '正确
	dpos(0) = -total_pul/units(0) '测试出来的脉冲个数值是反的，取反还原坐标
	ENDIF
	END

相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

MODBUSM_3XGET -- 读对端输入寄存器

类型	通讯指令
描述	**把对端的 MODBUS 输入寄存器复制到本地。** **对应标准协议功能码 04，读输入寄存器。**
语法	MODBUSM_3XGET (startreg, num, local_reg)

	startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	**MODBUSM_3XGE**T(0,10,0)　'把对端输入寄存器 0-9 复制到通讯本地寄存器 0-9
相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

MODBUSM_BITSET -- 写对端线圈

类型	通讯指令
描述	**把本地 MODBUS 位寄存器设置到对端。** **对应标准协议功能码 05 或 15，写线圈。**
语法	MODBUSM_BITSET (startreg, num, local_reg) startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	**MODBUSM_BITSET** (0,10,0)　'把本地位寄存器 0-9 复制到通讯对端的寄存器 0-9
相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

MODBUSM_BITGET -- 读对端线圈

类型	通讯指令
描述	**把对端 MODBUS 位寄存器复制到本地。** **对应标准协议功能码 01，读线圈。**
语法	MODBUSM_BITGET (startreg, num, local_reg) startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	MODBUSM_BITGET (0,10,0)　'把通讯对端的位寄存器 0-9 复制到本地位寄存器 0-9
相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

MODBUSM_1XGET -- 读对端离散输入

类型	通讯指令
描述	**把对端 MODBUS 位寄存器复制到本地。** **对应标准协议功能码 02，读离散输入。**

语法	MODBUSM_1XGET (startreg, num, local_reg) startreg：对端的寄存器起始编号，从 0 开始 num：寄存器个数 local_reg：从本地系统 MODBUS 寄存器中取值，起始编号
适用控制器	通用
例子	**MODBUSM_1XGET** (0,10,0)　'把通讯对端的位寄存器 0-9 复制到本地位寄存器 0-9
相关指令	ADDRESS，PROTOCOL，PORT，SETCOM

12.7 控制器互联直接命令指令

SEND_RESULT -- 读取 send 结果

类型	通讯指令
描述	**读取 send 指令的结果。** 返回值：0-成功，其它为错误，包括从控制器返回的错误码。 MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	VAL=SEND_RESULT
适用控制器	4 系列以上控制器支持，支持版本号 20170618
相关指令	SEND_CMD

SEND_CMD -- send 命令

类型	通讯指令
描述	**主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。** 发送 BASIC 内容：cmdstring (参数列表) MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_CMD(cmdstring,可选参数列表) cmdstring：命令字符串 可选参数列表：个数可变，不带参数的时候，不添加括号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_CMD("MOVE",DIS1,DIS2,DIS3) SEND_CMD("MOVEABS",DIS1)

相关指令	SEND_RESULT

SEND_CMDAXIS -- send 命令

类型	通讯指令
描述	**主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。** 发送 BASIC 内容：cmdstring (参数列表)，AXIS(iaxis) MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_CMDAXIS (cmdstring,iaxis,可选参数列表) cmdstring：命令字符串 iaxis：轴号 可选参数列表：个数可变，不带参数的时候，不添加括号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_CMDAXIS("MOVE",IAXIS,DIS1)
相关指令	SEND_RESULT，SEND_CMD

SEND_ASSIGN -- send 命令

类型	通讯指令
描述	**主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。** 发送 BASIC 内容：cmdstring (参数列表)=value MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_ASSIGN (cmdstring,value,可选参数列表) cmdstring：命令字符串 value：赋值的内容 可选参数列表：个数可变，不带参数的时候，不添加括号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_ASSIGN("DPOS",0,0)　'生成 DPOS(0)=0 SEND_ASSIGN("DPOS(1)",0)　'生成 DPOS(1)=0
相关指令	SEND_RESULT

SEND_QUERY -- send 命令

类型	通讯指令
描述	**主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。** 发送 BASIC 内容：cmdstring(参数列表)。 不带参数的时候，不添加括号。 接收的内容根据 SEND_QUERYSET 的设置填到 TABLE 里面。 MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_QUERY (cmdstring,可选参数列表) cmdstring：命令字符串 可选参数列表：个数可变，不带参数的时候，不添加括号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_QUERYSET(0,1) SEND_QUERY("?dpos",0) 'table(0)保存控制器 DPOS(0)的内容 SEND_QUERY("?REMAIN_BUFFER(1)AXIS(0)",0) '发送字符串内容(“?REMAIN_BUFFER(1) AXIS(0)”) SEND_QUERYSET(0,2) SEND_QUERY("?dpos(0),dpos(1)") '从控制器会返回两个数据
相关指令	SEND_RESULT，SEND_QUERYSET

SEND_QUERYSET -- send 命令

类型	通讯指令
描述	**主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。** 发送 BASIC 内容：cmdstring (参数列表) AXIS(iaxis) MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_QUERYSET (dtindex, dtnumes) dtindex：接收内容存储的 TABLE 编号 dtnumes：接收最多存储的 TABLE 个数
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_QUERYSET (0,1) SEND_QUERYSET (0,1)
相关指令	SEND_RESULT，SEND_QUERY

12.8 控制器互联文件发送指令

SEND_ZAR -- U 盘操作

类型	通讯指令
描述	**通过主控制器的 U 盘升级从控制器的程序，结果通过 MODBUSM_STATE 查看。** MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_ZAR("ufilename") ufilename：U 盘文件名，支持字符串的数组和其它字符串类型
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_ZAR("1.ZAR")
相关指令	MODBUSM_STATE，SEND_PERCENT

SEND_FLASH --数据拷贝

类型	通讯指令
描述	**主控制器的 U 盘和从控制器的 FLASH 相互拷贝，结果通过 MODBUSM_STATE 查看。** MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_FLASH (dir,uid,flashid) dir：1-U 盘拷贝到控制器 FLASH，0-控制器 FLASH 拷贝到 U 盘 uid：U 盘的文件编号，规则同 U_WRITE flashid：控制器的 FLASH 编号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_FLASH (1,1,1)
相关指令	MODBUSM_STATE，SEND_PERCENT

SEND_FILE -- 拷贝 U 盘数据

类型	通讯指令
描述	**主控制器的 U 盘和从控制器的文件相互拷贝，只支持 BIN 文件和 Z3P 文件，不支持文件功能的控制器会返回失败。**

	MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_FILE (dir, "ufile","contrfile") dir：1-U 盘拷贝到控制器 FLASH，0-控制器 FLASH 拷贝到 U 盘 ufile：U 盘的文件名 contrfile：控制器的文件名
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_FILE(1,"1.bin","1.bin")
相关指令	MODBUSM_STATE，SEND_PERCENT

SEND_IFLASH -- 拷贝 flash 数据

类型	通讯指令
描述	**主控制器的 FLASH 和从控制器的 FLASH 相互拷贝，结果通过 MODBUSM_STATE 查看。** MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	SEND_IFLASH (dir,id,flashid) dir：1-主控制器 FLASH 拷贝到控制器 FLASH，0-控制器 FLASH 拷贝到主控制器 FLASH id：主控制器的 FLASH 编号 flashid：控制器的 FLASH 编号
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	SEND_FLASH (1,1,1)
相关指令	MODBUSM_STATE，SEND_PERCENT

SEND_PERCENT -- 查询指令进度

类型	通讯指令
描述	**SEND 等需要长时间完成的指令返回百分比，可以用来显示进度条，范围 0-100。** MODBUSM 指令超时重发要注意，从控制器可能收到 2 次消息，对寄存器的扫描也可能出现 2 次。SEND 指令超时重发对从控制器没有影响，重要标志性变量可以通过 SEND 指令来修改。
语法	percent = SEND_PERCENT ()

适用控制器	4 系列以上控制器支持，支持版本号 20170618
相关指令	SEND_ZAR，SEND_FLASH，SEND_FILE

ZAR_CONTROL -- 查看控制器类型

类型	U 盘函数
描述	**查看主控制器 U 盘的 ZAR 程序文件的对应控制器类型，此类型在 ZDevelop 中设置，避免主控制器和从控制器的文件混淆。**
语法	id = ZAR_CONTROL ("ufilename") ufilename：U 盘文件名，ZAR 文件
适用控制器	4 系列以上控制器支持，支持版本号 20170618

例子	id = ZAR_CONTROL("1.ZAR")
	IF(id/3000=3) Then 'ZHD 系列示教盒
	PRINT "zhd program"
	ENDIF

相关指令	SEND_ZAR，CONTROL

第十三章 系统相关指令

所有的日期时间参数或指令，LOCK 后不能修改。

13.1 控制器加密指令

APP_PASS -- 密码

类型	系统指令
描述	**控制器应用密码。** 下载 ZAR 包时可以选择校验这个密码，校验错误将不能下载。 LOCK 后不能修改 APP_PASS。 APP_PASS 采用不可逆算法加密，一旦忘记，将无法获知。
语法	APP_PASS(pass) pass：字母或数字，“_”等少数特殊符号，总共不要超过 16 个字符，不能设置为变量或表达式，否则变量名等会被认为是密码。
适用控制器	通用
例子	**APP_PASS**(Zmotion)
相关指令	LOCK

LOCK -- 锁定控制器

类型	系统指令
描述	**锁定控制器，不让对控制器操作。** 上位机的 zpj 程序仍可以修改，无法下载到控制器，但生成的 zar 文件仍可下载。 会阻止对控制器的枚举操作，比如打印所有数组等，但可以打印具体数组的值。 LOCK 密码采用不可逆算法加密，一旦忘记，将不可获知，pass 不能设置变量或表达式，否则变量名等会被认为是密码。
语法	LOCK (pass) pass：字母或数字，“_”等少数特殊符号，总共不要超过 16 个字符
适用控制器	通用
例子	**LOCK**(passwd)　　'锁定控制器，密码为 passwd
相关指令	UNLOCK

UNLOCK -- 解锁控制器

类型	系统指令

描述	**解锁控制器。** LOCK 密码采用不可逆算法加密，一旦忘记，将不可获知。
语法	UNLOCK (pass) pass：LOCK 时的密码
适用控制器	通用
相关指令	LOCK

13.2 系统时间指令

DATE -- 系统日期

类型	系统参数
描述	**设置系统日期，掉电保存，或是返回 2000 年 1 月 1 日以后的天数。** 仿真器无法修改日期。
语法	DATE= DD:MM:YYYY 或 DD:MM:YY
适用控制器	通用
例子	DATE=27:2:13 在线命令输入 >>PRINT　DATE 输出：4806 即 2013:2:27-2000:1:1 = 4806
相关指令	DATE$，RTC_DATE

DATE$ -- 系统日期

类型	字符串函数
描述	**字符串函数，按 DD:MM:YYYY 格式返回 DATE 设置日期。**
语法	DATE$
适用控制器	通用
例子	DATE=27:2:13 在线命令输入 >>PRINT　**DATE$** 输出：27:02:2013
相关指令	DATE，RTC_DATE

DAY -- 系统星期

类型	系统参数

描述	**设置系统时钟的星期，0-6，0 表示星期天，掉电保存。** 仿真器无法修改日期。 DAY 不随 DATE 和 RTC_DATE 改变而改变，两者独立。
语法	VAR=DAY，DAY=expression
适用控制器	通用
例子	例一 **DAY** = 3 例二 在线命令输入 >>PRINT　**DAY** 输出：3
相关指令	DAY$

DAY$ -- 系统星期

类型	字符串函数
描述	**字符串函数，返回 DAY 设置星期。** DAY$不随 DATE 和 RTC_DATE 改变而改变，两者独立。
语法	DAY$
适用控制器	通用
例子	在线命令输入 >>PRINT　**DAY$** 输出：Wednesday
相关指令	DAY

RTC_DATE --系统日期

类型	系统参数
描述	**设置或者获取系统日期，掉电保存，从 2000 年 1 月 1 日起。** 注意与 DATE 指令的表示格式是不一样的，返回值类型为整数。 仿真器无法修改日期。
语法	RTC_DATE = YYYYMMDD 或 YYMMDD
适用控制器	通用
例子	**RTC_DATE** = 20130227 在线命令输入 >>PRINT　**RTC_DATE** 输出：20130227
相关指令	DATE，DATE$

TIME -- 系统时间

类型	系统参数
描述	**设置系统时钟的时间，返回 0 点以后的秒数。** 仿真器无法修改日期。
语法	TIME = hh:mm:ss
适用控制器	通用
例子	例一 **TIME**= 11:14:40 例二 在线命令输入 >>PRINT **TIME** 输出：40541
相关指令	TIME$，RTC_TIME

TIME$ -- 系统时间

类型	字符串函数
描述	**字符串函数，按 24 小时的格式 hh:mm:ss 返回当前时间。**
语法	TIME$
适用控制器	通用
例子	在线命令输入 >>PRINT **TIME$** 输出：11:29:46
相关指令	TIME，RTC_TIME

RTC_TIME -- 系统时间

类型	系统参数
描述	**设置或者获取系统时间。** 注意与 TIME 指令的表示方式是不一样的。
语法	RTC_TIME = hhmmss
适用控制器	通用
例子	在线命令输入 >>**RTC_TIME** = 113706 >>PRINT **RTC_TIME** 输出：113706
相关指令	TIME，TIME$

ZPASSTODATE -- 时间计算

类型	系统参数
描述	**根据密码字符串计算到期天数。** 20170613 以上固件支持。 注意：此指令必须与软件 ZMotion DateToPass Tools 工具软件配合使用，软件请确认 clientstring 并联系厂家获取。 此指令防止枚举，必须间隔 5 秒以上再次调用。
语法	rtc_date = ZPASSTODATE ("clientstring", "passstring") clientstring：客户专用字符串，必须保密，与 ZMotion DateToPass Tools 工具软件配对使用，使用字符串语法输入 passstring：密码字符串,，10-13 位，使用字符串语法输入 rtc_date：返回保护时间，整数方式，格式 20180101，密码错误时，返回 0 或负数
适用控制器	通用

例子	vardate = ZPASSTODATE(clientstring,passstring)　'读取时间
	IF　RTC_DATE >=　vardate　THEN　'判断是否到期
	?"进入加密状态"
	ELSE
	?"解锁时间"
	ENDIF

相关指令	RTC_TIME

13.3 轴系统参数指令

WDOG -- 轴总使能

类型	系统参数
描述	**控制所有轴的使能。** 使用 EtherCAT 总线时，必须使 WDOG=1。
语法	WDOG=0/1
适用控制器	通用
例子	**WDOG**=1　'所有轴使能打开
相关指令	AXIS_ENABLE

DISABLE_GROUP -- 轴分组

类型	系统指令
描述	**一停多，把几个轴设置为一组，驱动器告警后会关闭组内的所有使能(ECAT 产品支持)，脉冲轴设置无意义。** 一般用于多工位分组。
语法	DISABLE_GROUP(AXIS1，AXIS2，…)
适用控制器	通用
例子	**DISABLE_GROUP**(-1)　　'取消所有组设置，此时报警关闭 WDOG **DISABLE_GROUP**(0,5,1)　'轴 0,5,1 设置一组 **DISABLE_GROUP**(4,2)　　'轴 4,2 设置一组 当轴 0/5/1 出现问题时，轴 0、轴 5 和轴 1 都会掉使能，而轴 4 和轴 2 不会，同样当轴 4/2 出现问题时，轴 4 和轴 2 会掉使能，轴 0、轴 5 和轴 1 不会
相关指令	WDOG，AXIS_ENABLE

ERROR_AXIS -- 报错轴号

类型	系统状态
描述	**出现错误的第一个轴号，-1 没有。**
语法	Var=ERROR_AXIS
适用控制器	通用
例子	?ERROR_AXIS　　'打印第一个出错轴号，打印结果，-1，当前没有轴错误
相关指令	MOTION_ERROR

MOTION_ERROR -- 报错轴列表

类型	系统状态
描述	**出现错误的轴列表。** 每个位表示一个轴号。位 0-n 表示轴 0-n。
语法	Var=MOTION_ERROR
适用控制器	通用
例子	PRINT　**MOTION_ERROR**　　'打印结果，0，未出错
相关指令	ERROR_AXIS

ERROR_SET -- 报错输出

类型	系统指令
描述	**当 BASIC 程序运行出错的时候，输出口自动打开，并且把错误信息写到对应的 MODBUS 寄存器，输出口状态还原时 BASIC 程序会自动重新运行。** 20130906 以上版本提供支持，寄存器长度至少 32 个字节。
语法	ERROR_SET(输出口,modbus 寄存器地址[,errsubname]) errsubname：150721 以上版本提供；设置一个 SUB 过程进行临时处理，语法错误停止时此函数自动被调用，此过程不要使用 WAIT 等可能阻塞的指令，而且必须很精简。此过程里面再出现语法错误不会再被处理。
适用控制器	通用
例子	**ERROR_SET**(1,200,error_deal) MOV(30)　　'拼写错误，此时运行错误，输出口 1 打开，并在寄存器记录错误信息 'Modbus_string(200,32) ="sample_move.bas,6,e2043" END SUB error_deal()　　'报错时调用函数 ?"进入错误处理 sub"　'打印信息 '以下编写需要的功能程序 END SUB 可以在线命令栏输入?MODBUS_STRING(200,32)查看错误信息： MODBUS_STRING(200,32) ="sample_move.bas,6,e2043" sample_move.bas：表示文件名 6：表示当前出现错误的行号 e2043：表示错误码

13.4 IP 参数指令

IP_ADDRESS -- IP 地址

类型	系统参数，自动存储到 FLASH
描述	**控制器 IP 地址。** 只有带以太网接口的控制器支持，读取时以 32 位整数返回，见例程。 修改立刻生效，使用网口连接时，修改后网口会断开，需重连。 单网卡连接控制器，网卡与控制器处于同一网段即可。 多网卡连接控制器，网卡之间要为不同网段，控制器连接哪个网卡就设置为对应网段。
语法	IP_ADDRESS = dot.dot.dot.dot
适用控制器	通用

例子	在线命令输入 >>**IP_ADDRESS**=192.168.0.26 >>PRINT **IP_ADDRESS** 输出：436250816 具体转换过程如下： 四段转换为二进制 192 -- 1100 0000 168 -- 1010 1000 0 -- 0000 0000 26 -- 0001 1010 二进制重新组合 26 0 168 192 0001 1010 0000 0000 1010 1000 1100 0000 转化为十进制 436250816
相关指令	IP_GATEWAY，IP_NETMASK

IP_GATEWAY -- IP 网关

类型	系统参数，自动存储到 FLASH
描述	**控制器 IP 网关。** 只有带以太网接口的控制器支持，读取时以 32 位整数返回。
语法	IP_GATEWAY = dot.dot.dot.dot
适用控制器	通用
例子	在线命令输入 >>**IP_GATEWAY** =192.168.0.1 >>PRINT **IP_GATEWAY** 输出：16820416 具体转换过程参考 IP_ADDRESS 例程
相关指令	IP_NETMASK，IP_ADDRESS

IP_NETMASK -- IP 掩码

类型	系统参数，自动存储到 FLASH
描述	**控制器 IP 网络掩码。** 只有带以太网接口的控制器支持，读取时以 32 位整数返回。
语法	**IP_NETMASK** = dot.dot.dot.dot

适用控制器	通用
例子	在线命令输入 >>**IP_NETMASK** =255.255.252.0 >>PRINT **IP_NETMASK** 输出：16580607 具体转换过程参考 IP_ADDRESS 例程
相关指令	IP_GATEWAY，IP_ADDRESS

13.5 控制器信息指令

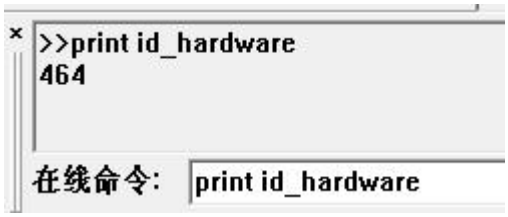
VERSION_DATE -- 系统固件版本

类型	系统状态
描述	**系统固件版本号。**
语法	VAR1 = VERSION_DATE
适用控制器	通用
例子	?***VERSION_DATE** '打印出固件版本 结果:20161122  “控制器状态”-“SoftVersion”查看

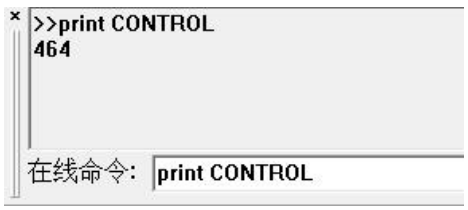
VERSION -- 系统软件版本

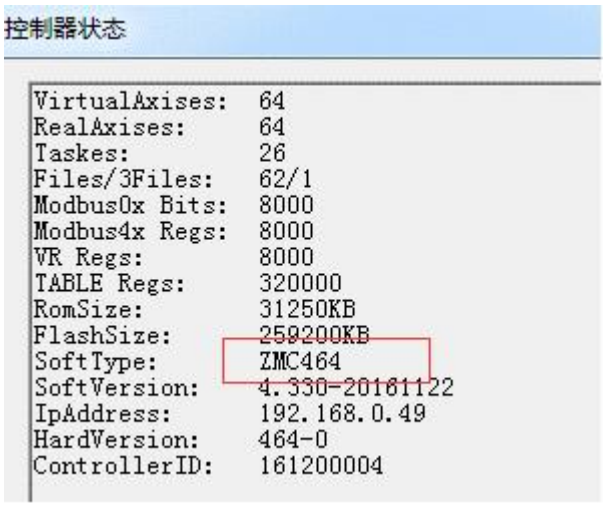
类型	系统状态
描述	**系统软件版本号。**
语法	VAR1 = VERSION
适用控制器	通用
例子	?***VERSION** '打印出软件版本 结果:4.3300  “控制器状态”-“SoftVersion”查看

ID_HARDWARE -- 控制器硬件型号

类型	系统只读参数
描述	**返回控制器硬件型号。**
语法	Val=ID_HARDWARE
适用控制器	通用
例子	在线命令输入 >>PRINT **ID_HARDWARE** '打印控制器型号 输出：464  “控制器状态”-“HardVersion”查看
相关指令	CONTROL

CONTROL -- 控制器软件型号

类型	系统只读参数
描述	**返回控制器软件型号。**
语法	Val=CONTROL
适用控制器	通用
例子	在线命令输入 >>PRINT **CONTROL** '打印控制器型号 输出：464  “控制器状态”-“SoftType”查看

	
相关指令	ID_HARDWARE

SYSTEM_ZSET -- 控制器设置

类型	系统参数
描述	**控制器设置。** 设置参数： bit0：1-VP_SPEED 缺省使用插补速度， 0-VP_SPEED 使用单轴的速度 bit1：1-使用 MOVE_OP 精确输出功能,，0-MOVE_OP 普通方式 bit4：1-对带编码器功能的轴，使用编码器位置的 MOVE_OP 精准方式 bit7：1-总线时钟优化，0-脉冲轴时钟优化 SYSTEM_ZSET 一旦开启，所有支持精准输出功能的输出口都变为精准模式，部分控制器型号在一个控制器周期内只能操作一个精准输出口，新版本固件不建议使用此指令，直接采用 AXIS_ZSET 指令对主轴开启精准输出。 20170505 以上固件版本支持这个位设置。 使用前要求 MPOS 必须跟随 DPOS，编码器精准功能受驱动器的响应影响，输出时刻速度越平稳，效果越好。 通过打印此指令查看 BIT1 的值来判断控制器是否支持精确位置输出，见例程。
语法	可读：value=SYSTEM_ZSET 可写：SYSTEM_ZSET=value
适用控制器	通用

例子	判断控制器是否支持精确位置输出
	IF READ_BIT2(1,SYSTEM_ZSET)=1 THEN '读取 bit1 的值
	?"支持精确位置输出"
	ELSE
	?"不支持精确位置输出"
	ENDIF

相关指令	MOVE_OP，AXIS_ZSET

LEDOUT -- 控制器指示灯

类型	系统指令
描述	**操作控制器指示灯。** 电源指示灯无法操作。
语法	LEDOUT (num,state) num：指示灯编号；1-RUN 指示灯，2-ALM 指示灯 state：状态；0-关闭，1-打开
适用控制器	通用
例子	**LEDOUT**(1,0)　'关闭运行灯 **LEDOUT**(2,0)　'关闭报警灯

SERIAL_NUMBER -- 控制器唯一 ID

类型	系统只读参数
描述	**返回控制器唯一 ID。** 是一个唯一的序列号，生成 ZAR 包的时候也可以和这个 ID 绑定，这样这个 ZAR 只能为这个控制器所用。
语法	Var=SERIAL_NUMBER
适用控制器	通用
例子	例一 PRINT　**SERIAL_NUMBER**　'打印控制器 ID 打印结果： 191201941 例二 控制器的 9 位 ID 保存到 VR，由于 4 系列以下控制器 VR 为单精度浮点型，只有 8 位有效数值，可采用两个 VR 空间保存。 ?**SERIAL_NUMBER** GLOBAL giA,giB giA = **SERIAL_NUMBER** MOD 100000　'取余数 VR(0)=giA giB = **SERIAL_NUMBER** \ 100000　'整除 VR(1)=giB PRINT giA,VR(0) PRINT giB,VR(1) 打印结果： 191201941 1941　1941 1912　1912

SERVO_PERIOD -- 总线通讯周期

类型	系统参数
描述	**总线伺服通讯周期。** 缺省 1000 微秒，修改功能预留，目前只读。 **Rtex 驱动使用时请按以下设置** 控制器周期 500us 时，将驱动器的 P7.20 设置为 3，P7.21 设置为 1。 控制器周期 1000us 时，将驱动器的 P7.20 设置为 6，P7.21 设置为 1。 查看 Rtex 驱动器设置请看例子。
语法	value=SERVO_PERIOD
适用控制器	通用
例子	在线命令打印控制器通讯周期和 **Rtex 驱动器设置** >>PRINT **SERVO_PERIOD** '打印伺服更新周期，结果，1000 >>DRIVE_READ(7*256+20)AXIS(0) '打印 Rtex 轴 0 驱动器周期设置 >>DRIVE_READ(7*256+21)AXIS(0) '打印 Rtex 轴 0 驱动器周期比值设置
相关指令	SERVO

SYS_ZFEATURE -- 系统规格

类型	系统参数
描述	**获取系统最大规格。** ECI150830 以上固件支持，4 系列 170530 以上固件支持。
语法	num = SYS_ZFEATURE (code) num：返回值 code：获取数据类型 0- 最大虚拟轴数 1- 支持电机个数 2- 本体输入个数 3- 本体输出个数 4- 本体模拟输入 AIN 5- 本体模拟输出 AOUT 6- PWM 个数 7- BASIC 任务数，不包含中断任务 8- 总线槽位数 9- 3 次文件个数 10- 串口链接数 11- 网络链接数 12- 网络自定义链接数 0-不支持 13- 网络互联主端数，0-不支持， （从端总是支持） 14- FLASH 块数 15- FLASH 块大小

	16- VR 个数 17- MODBUS_BIT 个数 18- MODBUS_REG 个数 19- 定时器个数 20- 数组空间 21- 虚拟最大输入个数，对应 PLC 的 X 寄存器个数 22- 虚拟最大输出个数，对应 PLC 的 Y 寄存器个数 23- 虚拟最大 AIN 个数 24- 虚拟最大 AOUT 个数 25- PLC 计数器 26- PLC S 寄存器 27- PLC V 寄存器 28- PLC Z 寄存器 29- PLC L 寄存器 30- HMI 个数，（包括网络 HMI 与本体 HMI） 31- 本体自带 HMI 个数
适用控制器	通用
例子	?SYS_ZFEATURE (0)　　'打印最大轴数 ?SYS_ZFEATURE (17)　　'打印 MODBUS_BIT 个数
相关指令	/

13.6 TABLE 数组指令

TABLE -- 系统缺省数组

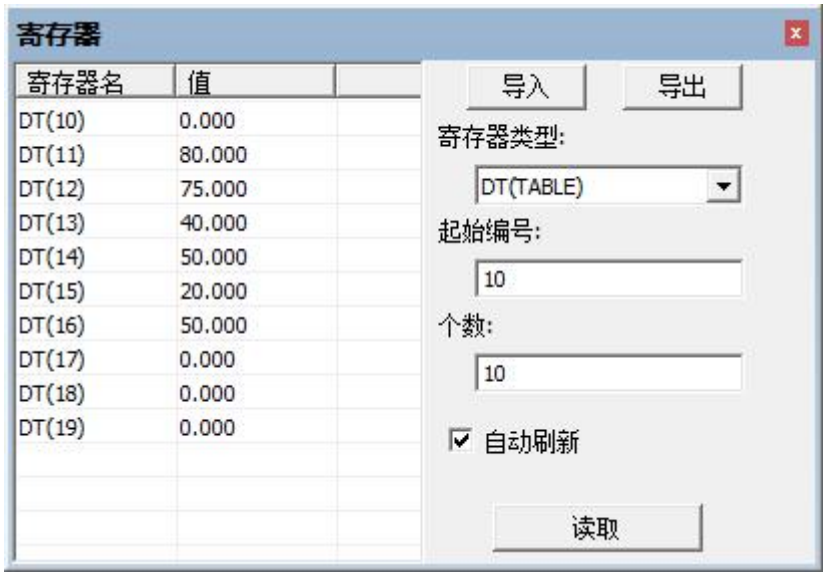
类型	系统数组
描述	**系统缺省建立的全局数组，所有程序都可以访问。** **数据录波缓冲区，凸轮数据表，螺距补偿表，机械手参数，等等都是用 table 来存储。**
语法	TABLE(index) = value , VAR1 = TABLE(index)，TABLE(index [, value1..])
适用控制器	通用
例子	**TABLE**(0) = 10　　'table(0)赋值 10 **TABLE**(10,100,200,300)　　'table(10)赋值 100，table(11)赋值 200，table(12)赋值 300
相关指令	TSIZE

TSIZE -- table 大小

类型	系统参数
描述	**TABLE 的所有元素个数，可以修改 TABLE 空间大小。** 请在程序开头修改，不能超过 TABLE 最大空间。
语法	Var=TSIZE　　TSIZE=Value

适用控制器	通用
例子	读取： PRINT TSIZE '打印出控制器 table 大小 设置： TSIZE=10000 '设置 table 的大小，不能超过控制器 table 最大 size
相关指令	TABLE

TABLESTRING -- 字符串格式打印 table

类型	字符串函数
描述	**按照字符串格式打印 table 里的数据。** 数据自动转换，打印出来的数据为 ASCII 码。
语法	TABLESTRING(index, length) index：打印数据起始地址 length：要打印的数据长度
适用控制器	通用
例子	例一 TABLESTRING(0,5)= "abc" '从 tablestring(0)开始保存字符串 PRINT TABLESTRING(0,5) '打印保存的字符串 '打印结果：abc 例二 TABLE(100,68,58,92) PRINT TABLESTRING(100,3) '字符串格式打印数据，转换为 ASCII 码 '打印结果：D:\ 在 ZDevelop 软件的“视图”-“寄存器”可以查看 TABLE 寄存器的每个位置的数据。 

13.7 示波器相关指令

TRIGGER -- 触发示波器

类型	系统指令
描述	**开始执行示波器的数据采样。** 150723 以后版本支持，与 zdevelop 的示波器功能合用。
语法	TRIGGER
适用控制器	通用
例子	**TRIGGER** '示波器开始抓取下面运动的每时刻速度，存储在示波器、功能设置的 TABLE 区间 MOVE(10000)
相关指令	SCOPE

SCOPE -- 数据采样

类型	系统指令
描述	**数据采样，保存到 TABLE，最多同时采样 8 类数据。** 使用 TRIGGER 开始自动采样，采样时间=采样周期*采样个数。
语法	SCOPE(enable[, period]) SCOPE(enable, period, table_start, table_stop, p0 [,p1 [,p2 [,p3 [,p4 [,p5 [,p6 [,p7]]]]]]]) enable：使能与否 period：系统周期，一般为 1ms，可用 SERVO_PERIOD 查看 table_start：采样数据存储在 TABLE 的起始位置 table_stop：TABLE 结束位置，减去起始位置为采样个数 p0~p7：采样数据类型，等分存储在 TABLE 范围
适用控制器	通用
例子	BASE(0) ATYPE=1 UNITS=100 DPOS=0 SPEED=100 ACCEL=1000 **SCOPE**(ON,10,0,1000,DPOS(0),MSPEED(0)) '每隔 10ms 抓取 dpos 和 mspeed 存储在 TABLE 0~1000，0~499 存 dpos，500~1000 存 mspeed 共采样 1000/2*10=5s TRIGGER '开始抓取 MOVE(10000)
相关指令	TRIGGER，SCOPE_POS

SCOPE_POS -- 采样点数

类型	系统指令
描述	**只读，返回当前 SCOPE 采样数据的点数**
语法	VAR = SCOPE_POS
适用控制器	通用
例子	BASE(0) ATYPE=1 UNITS=100 DPOS=0 SPEED=100 ACCEL=1000 SCOPE(ON,10,0,1000,DPOS(0),MSPEED(0))　　'采样设置 TRIGGER　　'开始抓取 MOVE(10000) WHILE 1 ?**SCOPE_POS**　　'返回当前已采样保存的点数 WEND
相关指令	SCOPE

13.8 VR 相关指令

CLEAR -- 清除 VR

类型	系统指令
描述	**清除所有 VR 内容。**
语法	CLEAR()
适用控制器	通用
例子	**CLEAR**()　　'清除 VR 全部数据
相关指令	VR

VR -- 掉电保存

类型	系统指令
描述	**掉电保存寄存器组，32 位浮点型。** 注意不同型号控制器的数量有区别。 保存浮点数，与 VR_INT 和 VRSTRING 共用一个空间。
语法	VR(index) = value , VAR1 = VR(index)
适用控制器	通用
例子	**VR**(0) = 10.58 aaa = **VR**(0)

	?aaa 打印结果 10.5800
相关指令	VR_INT，VRSTRING

VR_INT -- 掉电保存整型

类型	系统指令
描述	**掉电保存寄存器组，32 位整型。** 注意不同型号控制器的数量有区别。 只能保存整型，与 VR 和 VRSTRIG 共用一个空间。
语法	VR_INT(index) = value , VAR1 = VR_INT(index)
适用控制器	通用
例子	**VR_INT**(0) = 10.58 aaa = **VR_INT**(0) ?aaa 打印结果 10，仅保留整数部分
相关指令	VR，VRSTRING

VRSTRING -- 掉电保存字符串

类型	字符串函数
描述	**用 VR 来存储字符串。** 保存 ASCII 码，与 VR 和 VR_INT 共用一个空间，一个字符占用一个 VR。
语法	VRSTRING (index[, chares]) index：起始的 VR 编号，编号从 0 开始 chares：读取的字符总数
适用控制器	通用
例子	在线命令输入 >> **VRSTRING** (0, 8) = "abc"　　'保存字符串 >>PRINT　**VRSTRING** (0, 8) 输出：abc
相关指令	VR，VR_INT

第十四章 存储相关指令

ZMotion 运动控制器都拥有内部 FLASH 存储器，部分型号控制器具备外部存储器接口，可以接 U 盘或 SD 卡，参见具体控制器的硬件手册。

U 盘或 SD 卡要格式化为 FAT 格式。

14.1 U 盘相关指令

FILE -- U 盘文件操作

<table>
<tr><td>类型</td><td>文件指令</td></tr>
<tr><td>描述</td><td>加载、搜索控制器或 U 盘文件。
根据对应字符串选择功能。

VPLC5 系列是 Linux 系统，读取文件名 filename 是区分大小写的，名称必须都大写。</td></tr>
<tr><td>语法</td><td>value = FILE "function" ,…
<table>
<tr><td>"LOAD_ZAR"</td><td>FILE "LOAD_ZAR","filename"
加载 U 盘里面的 ZAR 升级文件。
filename：程序文件名

升级失败返回 0，同时会 WARN 输出失败原因。
升级成功后会自动启动 ZAR 文件，因此返回值 TRUE 没有意义。
⚠升级后原来的调试状态的断点会被清除掉。</td></tr>
<tr><td>"LOAD_TCF"</td><td>FILE "LOAD_TCF","filename",tableindex,maxsize
专有文件 TCF 文件读取。
filename：文件名
tableindex：存储起始 TABLE 索引
maxsize：存储 TABLE 总索引个数
<table>
<tr><td>TABLE0</td><td>标志位</td><td></td></tr>
<tr><td>TABLE1</td><td>总点数</td><td></td></tr>
<tr><td>TABLE2</td><td>胶头编号</td><td></td></tr>
<tr><td>TABLE100</td><td>点类型</td><td>第一个点</td></tr>
<tr><td>TABLE101</td><td>X 坐标</td><td></td></tr>
<tr><td>TABLE102</td><td>Y 坐标</td><td></td></tr>
<tr><td>TABLE103</td><td>Z 坐标</td><td></td></tr>
<tr><td>TABLE104</td><td>预留</td><td></td></tr>
<tr><td>TABLE105</td><td>点类型</td><td>第二个点</td></tr>
<tr><td>TABLE106</td><td>X 坐标</td><td></td></tr>
<tr><td>TABLE107</td><td>Y 坐标</td><td></td></tr>
</table></td></tr>
</table></td></tr>
</table>

	TABLE108	Z 坐标
	TABLE109	预留

"LOAD_BYTE"	FILE"LOAD_BYTE","filename",tableindex,maxsize, offset 字节方式加载文件。 filename：文件名 tableindex：存储起始 TABLE 索引 maxsize：存储 TABLE 总索引个数 offset：文件开始读取的字节偏移

TABLE0	总字节数
TABLE1	读取到的第一个字节
TABLE2	第二个字节
TABLEn	第 n 个字节

"FIND_FIRST"	FILE "FIND_FIRST", type, vr 搜索 U 盘文件。 type：1-文件/2-文件夹/ ".extend" 文件后缀名 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FIND_NEXT"	FILE "FIND_NEXT", vr 搜索下一个 U 盘文件。 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FIND_PREV"	FILE "FIND_PREV", vr 搜索上一个 U 盘文件。 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FLASH_FIRST"	FILE "FLASH_FIRST", type, vr 搜索 FLASH 文件，只支持 BIN 和 Z3P 文件。 type：1-文件/2-文件夹/ ".extend" 文件后缀名 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FLASH_NEXT"	FILE "FLASH_NEXT", vr 搜索下一个 FLASH 文件。 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FLASH_PREV"	FILE "FLASH_PREV", vr 搜索上一个 FLASH 文件。 vr：vrstring(vr)存储查找的结果，超过最大 vr 时，使用 modbus_string。
"FLASH_DEL"	FILE "FLASH_DEL", string 删除指定文件。 string：文件名全称，带扩展名。
"DELETE"	FILE "DELETE","filename" 删除 U 盘指定文件。 filename：文件名全称，带扩展名。

<table>
<tr><td></td><td>
<table>
<tr><td>"COPY_FROM"</td><td>FILE "COPY_FROM", "FLASH 文件名"[, "U 盘文件名"]
FLASH 文件拷贝到 U 盘，只支持 BIN 和 Z3P 文件。
FLASH 文件名规则：SD 块号.BIN
如 SD0.BIN 表示 FLASH 块号 0
SD1.BIN 表示 FLASH 块号 1</td></tr>
<tr><td>"COPY_TO"</td><td>FILE "COPY_TO", "U 盘文件名"[, "FLASH 文件名"]
U 盘文件拷贝到 FLASH，只支持 BIN 和 Z3P 文件。</td></tr>
<tr><td>"FLASH_COPY"</td><td>FILE "FLASH_COPY" "src","des"
支持文件夹的拷贝
可以带盘符 A、C
C 盘，表示 FLASH 目录
A 盘，表示 U 盘</td></tr>
<tr><td>"FLASH_DEL"</td><td>FILE "FLASH_DEL" "fileorddir"
支持文件夹的删除
可以带盘符 A、C
C 盘，表示 FLASH 目录
A 盘，表示 U 盘</td></tr>
<tr><td>"MAKE_DIR"</td><td>FILE "MAKE_DIR" "路径"</td></tr>
</table>
function：功能选择
</td></tr>
<tr><td>适用控制器</td><td>通用，使用 U 盘功能时要求控制器带 U 盘接口</td></tr>
<tr><td>例子</td><td>

例一 下载 zar 升级程序

```
DIM   result                              '定义变量
IF U_STATE=TRUE THEN                      'U 盘插入判断
  result = FILE "find_first",".zar",10    '扫描第一个 zar 格式文件，文件名保存到 VR
  IF result=TRUE THEN                     '扫描成功判断
    FILE"load_zar",VRSTRING(10,20)   '下载扫描到文件名与存储到 VR
                                           '里字符相同的 zar 文件
  ENDIF
ENDIF
END
```

例二 查找 zar 升级程序

```
FILE "find_next",10          '查找下一个 zar 文件存储结果到 vrstring(10)
FILE "find_prev",20          '查找上一个 zar 文件存储结果 vrstring(20)
```

例三 FLASH 与 U 盘数据互相拷贝

```
DIM   a,aa(8)
a=10
FOR   i=0 TO 7
      aa(i)=i
NEXT
WHILE 1
IF   SCAN_EVENT(IN(0))> 0 THEN
      FLASH_WRITE 1,a aa
```

</td></tr>
</table>

```
            FILE"copy_from","sd1.bin"     '将 flash 块 1 的数据复制到 U 盘的 sd1 文件
            PRINT "flash 块的数据复制到 U 盘"
        ELSEIF    SCAN_EVENT(IN(1))> 0 THEN
            FILE"copy_to","sd1.bin"   '读取 sd1 的数据写入 flash 块 1
            PRINT "U 盘数据写入 flash"
            FLASH_READ 1,a,aa
            PRINT *aa
        ENDIF
WEND
END
```

例四　读取/删除 U 盘文件

FILE"LOAD_BYTE","00.txt",200,10,0　　'读取 U 盘中 00.txt 文本文件的数据保存到 table(200)开始的 10 个地址中，偏移量为 0，从第一个字符开始读取

FILE　"DELETE" , "sd0.bin"　　'删除 U 盘上名称为 sd0.bin 的文件

00.txt 文件内容：ZMOTION

读取结果：第一个位置存储字符个数，后面的为依次存储字符数据。

寄存器名	值
DT(200)	7.000
DT(201)	90.000
DT(202)	77.000
DT(203)	79.000
DT(204)	84.000
DT(205)	73.000
DT(206)	79.000
DT(207)	78.000
DT(208)	0.000
DT(209)	0.000

U_STATE -- U 盘状态

类型	系统状态函数
描述	**检查 U 盘是否插入。** 有插入返回 TRUE，否则返回 FALSE。 不具备外部存储接口的控制器不支持此命令。 U 盘里面的文件不要放太多。
语法	Val=U_STATE
适用控制器	带 U 盘接口
例子	?**U_STATE**　　'打印 U 盘状态 IF　**U_STATE** = TRUE THEN　　'U 盘已插入 　　U_READ　1, VAR, ARRAY1, ARRAY2(1)　　'U 盘数据读取 ENDIF

相关指令	U_READ，U_WRITE

U_READ -- 从 U 盘读取

类型	存储指令
描述	**从外部存储器读取数据到变量或数组里面。** 不具备外部存储接口的控制器不支持此命令。 文件格式为 32 位 ieee 浮点数顺序存储，一个变量或一个数组元素占用一个浮点数，可以通过 PC 事先做好文件，然后用 U_READ 指令读取。
语法	U_READ sect_num, [,varname] [,arrayname] [,arrayname(a)][, arrayname(a,length)] sect_num：文件编号，对应到 SD【filenum】.BIN varname：变量名 arrayname：数组名，可以为 TABLE，VR a：操作的数组索引 length：操作的数组元素个数
适用控制器	带 U 盘接口
例子	IF U_STATE = TRUE THEN 'U 盘已插入 **U_READ** 1, VAR, TABLE(0), ARRAY2(1) '读取 U 盘文件 SD1 数据 ENDIF
相关指令	U_WRITE，U_READ2，U_STATE

U_READ2 -- 从 U 盘读取 2

类型	存储指令
描述	**从外部存储器读取数据到变量或数组里面。** 不具备外部存储接口的控制器不支持此命令。 文件格式为 32 位 ieee 浮点数顺序存储，一个变量或一个数组元素占用一个浮点数，可以通过 PC 事先做好文件，然后用 U_READ 指令读取。
语法	U_READ sect_num,star_num[,varname][,arrayname][,arrayname(a)][,arrayname(a,length)] sect_num：文件编号，对应到 SD【filenum】.BIN start_num：文件内读取的起始位置 varname：变量名 arrayname：数组名，可以为 TABLE，VR a：操作的数组索引 length：操作的数组元素个数
适用控制器	带 U 盘接口
例子	IF U_STATE = TRUE THEN 'U 盘已插入 **U_READ2** 1,10,VAR, TABLE(0), ARRAY2(1) '读取 U 盘文件 SD1 数据从 SD1 位置 10 开始

	ENDIF
相关指令	U_READ，U_WRITE，U_STATE

U_READDSB -- DSB 文件读取

类型	存储指令
描述	**DSB 文件读取。** 不具备外部存储接口的控制器不支持此命令。
语法	U_READDSB "filename", tableindex, maxsize [, flag] filename：文件名 tableindex：存储起始 TABLE 索引 maxsize：存储 TABLE 总索引个数 flag：预留
适用控制器	带 U 盘接口

编号	内容	
0	标志位：标明一行的变量数(暂无用)	
1	针迹数:	
2--29	文件名:28 个字符	
30	换色次数：	
31	+X:	
32	-X:	
33	+Y:	
34	-Y:	
35	AX: +	
36	AY: -	
37	MX: +	
38	MY: +	
39	PD:	
40	L_limt	左边界坐标
41	R_limt	右边界坐标
42	U_limt	上边界坐标
43	D_limt	下边界坐标
99	已读到总行数	
100	行类型	第一个点
101	参数 1：X 相对位移	
102	参数 2：Y 相对位移	
103	参数 3：相对起针点 X 坐标	
104	参数 4：相对起针点 Y 坐标	
105	行类型	第二个点
106	参数 1：X 相对位移	
107	参数 2：Y 相对位移	

相关指令	U_READ，U_WRITE，U_STATE

U_WRITE -- 输出到 U 盘

类型	存储指令
描述	**存储变量或者数组,数组的单个或部分元素到外部存储器里面。** 不具备外部存储接口的控制器不支持此命令。 文件格式为 32 位 IEEE 浮点数顺序存储，一个变量或一个数组元素占用一个浮点数，可以通过 PC 事先做好文件，然后用 U_READ 指令读取。
语法	U_WRITE sect_num, [,varname] [,arrayname] [,arrayname(a)] [,arrayname(a,length)] sect_num：文件编号，对应到 SD【filenum】.BIN varname：变量名 arrayname：数组名，可以为 TABLE，VR a：操作的数组索引 length：操作的数组元素个数
适用控制器	带 U 盘接口
例子	IF U_STATE = TRUE THEN 'U 盘已插入 **U_WRITE** 0, TABLE(0，10) 'TBALE0-10 的数据写入 U 盘文件 SD0 ENDIF
相关指令	U_READ，U_STATE

STICK_READ -- U 盘读取到 table

类型	存储指令
描述	**拷贝外部存储器的数据到 TABLE 里面,。** value=TRUE 表示操作成功，否则操作失败。 建议使用 U_READ。 不具备外部存储接口的控制器不支持此命令。
语法	value = STICK_READ (filenum, table_start [,format]) filenum：文件号，对应到 SD【filenum】 table_start：开始操作的起始 TABLE 号 format：写入文件的格式

值	描述
0（缺省）	float 格式存储，.BIN
1	文本格式存储，.CSV

适用控制器	带 U 盘接口
例子	**STICK_READ** (0,10,0)'拷贝外部存储器名称为 SD0 的 bin 文件数据到'TABLE，从 TABLE(10)开始保存
相关指令	U_READ，STICK_READVR

STICK_WRITE -- table 输出到 U 盘

类型	存储指令
描述	**拷贝 TABLE 的数据到外部存储器里面。** value=TRUE 表示操作成功，否则操作失败。 建议使用 U_WRITE。 不具备外部存储接口的控制器不支持此命令。
语法	value = STICK_WRITE(filenum, table_start [,length [,format]]) filenum：文件号，对应到 SD【filenum】 table_start：开始操作的起始 TABLE 号 length：要操作的 TABLE 元素个数，缺省 128 format：写入文件的格式 值 | 描述 0（缺省） | float 格式存储，.BIN 1 | 文本格式存储，.CSV
适用控制器	带 U 盘接口
例子	**STICK_WRITE**(0,0,128,0)　'拷贝 table 前 128 个元素以 float 格式到外部存储器，产生 SD0.BIN 文件
相关指令	U_WRITE，STICK_WRITEVR

STICK_READVR -- U 盘读取到 vr

类型	存储指令
描述	**拷贝外部存储器的数据到 VR 里面。** value=TRUE 表示操作成功，否则操作失败。 建议使用 U_READ。 不具备外部存储接口的控制器不支持此命令。
语法	value = STICK_READVR (filenum, vr_start [,format]) filenum：文件号，对应到 SD【filenum】 vr_start：开始操作的起始 VR 号 format：文件的格式 值 | 描述 0（缺省） | float 格式存储 1 | 文本格式存储
适用控制器	带 U 盘接口
例子	**STICK_READVR** (0,20,0)　'拷贝外部存储器名为 SD0 的 bin 文件数据到 VR，从 VR(20) 开始存储。
相关指令	U_READ，STICK_READ

STICK_WRITEVR -- vr 输出到 U 盘

类型	存储指令
描述	**拷贝 VR 的数据到外部存储器里面。** value=TRUE 表示操作成功，否则操作失败。 建议使用 U_WRITE。 不具备外部存储接口的控制器不支持此命令。
语法	value = STICK_READVR (filenum, vr_start [,format]) filenum：文件号，对应到 SD【filenum】 vr_start：开始操作的起始 VR 号 length：要操作的元素个数，缺省 128 format：文件的格式 值 | 描述 0（缺省） | float 格式存储，.BIN 1 | 文本格式存储，.CSV
适用控制器	带 U 盘接口
例子	STICK_WRITEVR (0,0,128,0)　'拷贝 VR 前 128 个元素以 float 格式存贮到外部存储器，生成 SD0.BIN 文件
相关指令	U_WRITE，STICK_WRITE

14.2 FLASH 相关指令

FLASH_WRITE -- flash 存储

类型	存储指令
描述	**存储变量或者数组，数组的单个或部分元素到内部 FLASH 里面，掉电保存。** 内部 FLASH 采用顺序存储的方式，读取的顺序必须与存储时的顺序一致。 内部 FLASH 有存储次数限制，不要随意循环操作。
语法	FLASH_WRITE sect_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a,length)] sect_num：flash 块编号,不同类型不一样 varname：变量名 arrayname：数组名，可以为 TABLE，VR，MODBUS a：操作的数组索引 length：操作的数组元素个数
适用控制器	通用
例子	例一 FLASH_WRITE　1, VAR, ARRAY1, ARRAY2(1) '把 VAR,ARRAY1,ARRAY2(1)的数据依次写入 flash 块 1

	例二 TABLE(1)=123.456 FLASH_WRITE 1, TABLE(1) TABLE(1)=200 FLASH_READ 1, TABLE(1) ?TABLE(1) '打印结果 123.45600 例三 FLASH 存储是 float 精度，对 32 位整数精度的数据，要使用 2 个 MODBUS_REG 来实现 MODBUS_LONG 的存储 MODBUS_LONG(1)=123456 '使用 MODBUS_REG(1)和 MODBUS_REG(2)存储 FLASH_WRITE 1, MODBUS_REG(1,2) '从 MODBUS_REG(1)开始取两个元素写入 FLASH 块，等价于 FLASH_WRITE 1, MODBUS_REG(1), MODBUS_REG(2) MODBUS_LONG(1)=100 FLASH_READ 1, MODBUS_REG(1,2) ?MODBUS_REG(1) '打印结果-7616 ?MODBUS_REG(2) '打印结果 1 ?MODBUS_LONG(1) '打印结果 123456
相关指令	FLASHVR，FLASH_READ

FLASH_READ -- flash 读取

类型	存储指令
描述	**从内部 FLASH 读取数据到变量，或数组里面。** 内部 FLASH 采用顺序存储的方式，读取的顺序必须与存储时的顺序一致。 读取未被写入过的 Flash 块时，会提示 Online command warn, File:C:\SDn.BIN read error，不影响使用。
语法	FLASH_READ sect_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a,length)] sect_num：flash 块编号,不同类型不一样 varname：变量名 arrayname：数组名，可以为 TABLE，VR a：操作的数组索引 length：操作的数组元素个数
适用控制器	通用
例子	FLASH_READ 1, VAR, ARRAY1, ARRAY2(1) '读取 FLASH 块 1 的数据，依次保存到变量 VAR，数组 ARRAY1，ARRAY2(1)
相关指令	FLASHVR，FLASH_WRITE

FLASHVR -- 拷贝 RAM 的数据

类型	存储指令

描述	**拷贝 RAM 的数据到 FLASH 里面。** ZMC00x（除了 005）、ECI 全系列控制器把 TABLE 数据存储到了 FLASH 的最后一个块，用 FLASH_WRITE, FLASH_READ 指令也可以操作到这个块，要避免冲突。 其他系列控制器把 TABLE 数据存储到另一个独立的存储区域，无法操作。
语法	FLASHVR (function) function：功能选择 值 | 描述 -1 | 整个 TABLE 都存储到 FLASH 并且上电时自动读取到 TABLE -2 | 取消 FLASH 上电时读取到 TABLE
适用控制器	通用
例子	**FLASHVR** (-1)　　'table 存到 flash 块，重新上电后再从 flash 读回 table
相关指令	FLASH_WRITE

FLASH_SECTSIZE -- flash 变量数

类型	系统状态函数
描述	**读取内部 FLASH 一个块可以存储的变量个数。** 不同控制器个数不同。
语法	value = FLASH_SECTSIZE
适用控制器	通用
例子	? FLASH_SECTSIZE 打印结果 20480　'ZMC1xx 系列 1024　'ZMC00x 系列 ...
相关指令	FLASH_SECTES

FLASH_SECTES -- flash 块数

类型	系统状态函数
描述	**读取控制器内部 FLASH 的总块数。** 不同控制器块数不同。 对 ZMC00x 系列控制器，FLASHVR 会使用用最后一个块，要避免冲突。
语法	value = FLASH_SECTES
适用控制器	通用
例子	?FLASH_SECTES　　'打印出 flash 块数 打印结果 128　'ZMC005
相关指令	FLASH_SECTSIZE

第十五章 中断相关指令

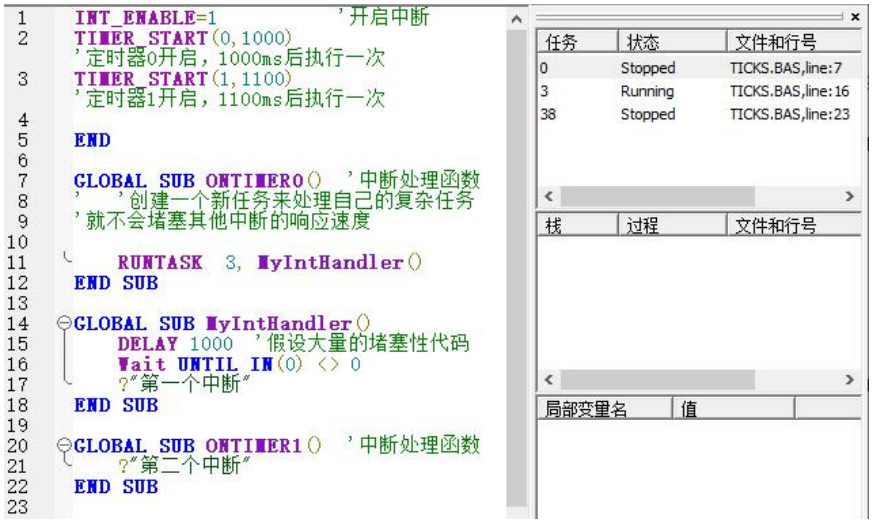
15.1 三种中断指令

INT_ENABLE -- 中断总开关

<table>
<tr><td>类型</td><td>系统参数</td></tr>
<tr><td>描述</td><td>中断总开关。
为了避免程序没有初始化好进入中断，中断开关缺省是关闭的。

控制器内部只有一个任务在处理所有的中断信号响应，有一个固定的中断任务号，如果中断处理函数过多，并且中断处理函数的代码太长，会造成所有的中断响应变慢，甚至是中断堵塞，影响其他中断执行。
解决办法：
(1)尽量减少中断的数量，很多应用都可以用循环扫描来处理。
(2)如果有一个中断处理函数特别长的话，就调用一个单独的任务来处理中断中的复杂任务，这样就不会堵塞其他的中断响应。</td></tr>
<tr><td>语法</td><td>INT_ENABLE = switch
<table><tr><td>值</td><td>描述</td></tr><tr><td>0（缺省）</td><td>关闭</td></tr><tr><td>1</td><td>打开</td></tr></table></td></tr>
<tr><td>适用控制器</td><td>通用</td></tr>
<tr><td>例子</td><td>错误示范，中断堵塞
如下图，定时器中断 0 开启，IN(0)为 0，导致中断函数堵塞在第 9 行，由程序无打印结果可得此时定时器中断 1 无法运行。
 

正确示范
若中断需要处理大量代码，在中断内创建一个任务来处理，如下图任务 3，执行下面程序，打印出“第二个中断”，定时器中断 0 堵塞，定时器中断 1 的运行不受影响。</td></tr>
</table>

<table>
<tr><td></td><td>


对应代码如下：
INT_ENABLE=1 '开启中断
TIMER_START(0,1000) '定时器 0 开启，1000ms 后执行一次
TIMER_START(1,1100) '定时器 1 开启，1100ms 后执行一次
END

GLOBAL SUB ONTIMER0() '中断处理函数
'创建一个新任务来处理自己的复杂任务，就不会堵塞其他中断的响应速度
RUNTASK 3, MyIntHandler()
END SUB

GLOBAL SUB MyIntHandler()
DELAY 1000 '假设大量的堵塞性代码
WAIT UNTIL IN(0) <> 0
?"第一个中断"
END SUB

GLOBAL SUB ONTIMER1() '中断处理函数
?"第二个中断"
END SUB
</td></tr>
<tr><td>相关指令</td><td>ONPOWEROFF，ONTIMERn</td></tr>
</table>

ONPOWEROFF -- 掉电中断 SUB

类型	中断
描述	**掉电中断程序入口，必须是全局的 SUB 过程。** 控制器只有 1 个掉电中断。 掉电中断执行的时间特别有限，只能写少数几条语句。
语法	GLOBAL ONPOWEROFF() … END SUB
适用控制器	通用
例子	INT_ENABLE = 1 dpos(0)=vr(0) '上电读取保存的数值，恢复坐标

	dpos(1)=vr(1) dpos(2)=vr(2) END GLOBAL SUB **ONPOWEROFF** () vr(0) = dpos(0) '保存坐标 vr(1) = dpos(1) vr(2) = dpos(2) END SUB
相关指令	INT_ENABLE

INT_ONn -- 外部输入中断 SUB

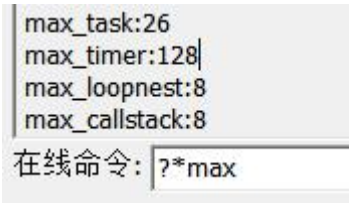
类型	中断
描述	**外部输入中断程序入口，上升沿触发，必须是全局的 SUB 过程。** 必须支持 PLC 功能的固件才可使用。 中断 IN 口 0~31。
语法	GLOBAL SUB INT_ONn() n 是 IN 编号 … END SUB
适用控制器	支持 PLC 功能的固件
例子	INT_ENABLE=1 '开启中断 END GLOBAL SUB **INT_ON**0 () '中断程序 PRINT "输入 IN0 上升沿触发" END SUB
相关指令	INT_OFFn，INT_ENABLE

INT_OFFn -- 外部输入中断 SUB

类型	中断
描述	**外部输入中断程序入口，下降沿触发，必须是全局的 SUB 过程。** 必须支持 PLC 功能的固件才可使用。 中断 IN 口 0~31。
语法	GLOBAL SUB INT_OFFn() n 是 IN 编号 … END SUB
适用控制器	支持 PLC 功能的固件
例子	INT_ENABLE=1 '开启中断 END GLOBAL SUB **INT_OFF**0 () '中断程序

	PRINT "输入 IN0 下降沿触发" END SUB
相关指令	INT_ONn，INT_ENABLE

ONTIMERn -- 定时器中断 SUB

类型	中断
描述	**定时器中断程序入口，必须是全局的 SUB 过程。** 不同的控制器型号定时器的个数不同，通过 Zdevelop 软件连接控制器，在线命令发送“?*max”查看。 
语法	GLOBAL SUB ONTIMERn()　　'n 是定时器编号 … END SUB
适用控制器	通用
例子	INT_ENABLE=1　　'开启中断 TIMER_START(0,100)　　'定时器 0 开启，100ms 后执行一次 END GLOBAL SUB **ONTIMER0**()　'中断程序 PRINT "**ontimer0** enter" 'TIMER_START(0,100)　'希望周期执行的话，要在 sub 里再打开 END SUB
相关指令	INT_ENABLE，TIMER_START

ONTIMERn -- 定时器中断 SUB

类型	系统参数
描述	**中断周期执行 BASIC 功能，每个 SERVO_PERIOD 执行一次。** 4 系列以上， 20170630 以上版本开放。
语法	命令语法：INT_CYCLE(function, taskid [, subname]) 参数描述： function：1-启动，2-停止 taskid：使用的 BASIC 任务号，BASIC 本身不能使用 subname：周期执行的 SUB 名称，程序必须足够精简 函数语法：var = INT_CYCLE(function, taskid) 参数描述：

	function： 3 -返回状态：1-使能，0-停止 4 -返回时间，上次的执行时间 us 5 -返回时间，执行最长时间 us 6 -返回时间最长限制 us 7 -返回错误情况的错误码，当 BASIC 中断函数执行错误的情况下，错误会设置 8 -返回错误行号 taskid：使用的 BASIC 任务号
适用控制器	通用
例子	DIM times INT_CYCLE(1,1,intisr) END GLOBAL SUB intisr() times=times+1 MOVE_PT(1,1) '每周期运行 END SUB
相关指令	INT_ENABLE

15.2 定时器指令

TIMER_IFEND -- 定时器状态

类型	系统函数
描述	**返回定时器是否结束。**
语法	value = TIMER_IFEND (timernum) 返回 0：定时器正在定时，即未执行对应中断程序。 返回 1：定时器定时完成，开始执行对应中断程序。 支持 PLC 的固件在启动定时器前打印 0，不支持 PLC 的固件打印 1。
适用控制器	通用
例子	INT_ENABLE=1 '开启中断 ?**TIMER_IFEND**(0) '定时器未启动，打印结果 0 '不支持 PLC 的固件则打印 1 TIMER_START(0,2000) '定时器 0 启动，定时 2s ?**TIMER_IFEND**(0) '打印结果，0，定时器正在定时 DELAY(2000) ?**TIMER_IFEND**(0) '打印结果，1，定时器定时完成，执行中断
相关指令	ONTIMERn，TIMER_START

TIMER_START -- 启动定时器

类型	系统指令
描述	**启动系统定时器，定时器只执行 1 次。**
语法	TIMER_START(timernum, time_ms) timernum：编号，0-定时器个数减 1 time_ms：定时器长度，单位毫秒 time 100 及以上为累积性计时器。
适用控制器	通用
例子	参考 TIMER_IFEND 例程
相关指令	ONTIMERn，TIMER_IFEND

TIMER_STOP -- 停止定时器

类型	系统指令
描述	**强制停止系统定时器。**
语法	TIMER_STOP (timernum) timernum：编号，0-定时器个数减 1
适用控制器	通用
例子	INT_ENABLE=1 '开启中断 TIMER_START(0,2000) '定时器 0 启动，周期 2s ?TIMER_IFEND(0) '打印结果，0，未运行 DELAY(2000) ?TIMER_IFEND(0) '打印定时器 0 状态 '打印结果，1，已运行 **TIMER_STOP**(0) '停止计时器 0 ?TIMER_IFEND(0) '打印结果，0，未运行
相关指令	ONTIMERn，TIMER_IFEND

第十六章 总线相关指令

16.1 编号释义

槽位号

槽位号是指控制器上接口的编号，总线接口缺省为 0。当控制器上有多个总线接口，?*SLOT 查看。

在指令说明中，用 SLOT 代表槽位号。

运动控制器支持单总线时，槽位号为 0；支持双总线时，EtherCAT 总线槽位号为 0，RTEX 总线槽位号为 1。

```
>>?*slot

Slot:0-ETHERCAT.
```

```
>>?*slot

Slot:0-ETHERCAT.
Slot:1-RTEX.
```

设备号

设备号是指一个槽位上连接的所有设备的编号，从 0 开始，按连接顺序依次增加，可以通过 NODE_COUNT(slot)查看。

在指令说明中，用 node 代表设备号。

驱动器编号

控制器会自动识别出槽位上的驱动器，编号从 0 开始，按连接顺序依次增加。

驱动器编号与设备号不同，假设控制器连接了 3 个设备，驱动器前面连接了两个其他 IO 设备，那么驱动器此时的设备号 node=2，而驱动器编号为 0。

16.2 基础指令

SLOT_SCAN -- 总线扫描

类型	总线指令
描述	**总线扫描** 通过 RETURN 返回成功与否， 返回-1 扫描成功，0 扫描失败。 Rtex 扫描失败会直接报错 6209。 遇到不支持的设备类型，RETURN 会返回 0。 Rtex 控制器未连接设备时会报错，EtherCAT 控制器不会。

	扫描后可以使用 NODE 指令读取相关设备信息，并使用 DRIVE 相关指令配置。
语法	SLOT_SCAN (slot) slot：控制器 EtherCAT 槽位号或 RTEX 槽位号，0-缺省
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	aa: '标记 aa
	SLOT_SCAN(0) '总线扫描
	? RETURN '打印返回值，-1 成功，0 失败
	IF RETURN THEN
	?NODE_COUNT(0) '扫描成功，返回当前连接的设备个数
	ELSE
	?"扫描失败"
	DELAY (1000) '等待 1s
	GOTO aa '扫描失败，跳转到 aa，重新扫描
	ENDIF

相关指令	SLOT_START，SLOT_STOP

SLOT_START -- 总线开启

类型	总线指令
描述	**总线启动。** 通过 RETURN 返回成功与否。返回-1 启动成功，0 启动失败。 需在总线扫描 SLOT_SCAN 成功后再执行。 执行前，先设置好 AXIS_ADDRESS，ATYPE，DRIVE_PROFILE
语法	SLOT_START (slot [,opstate]) slot：控制器 EtherCAT 槽位号或 RTEX 槽位号，0-缺省 opstate：启动到的 EtherCAT 状态，4-SAFEOP，8- OP(缺省) NODE_PDOBUFF 指令才使用
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	aa: '标记 aa
	SLOT_SCAN(0) '总线扫描
	IF RETURN THEN '扫描成功，进入轴设置
	AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
	ATYPE(0)=65 '轴类型 65，位置控制
	DRIVE_PROFILE(0)=0 'PDO 周期扫描设置
	SLOT_START(0) '开启总线
	ELSE
	?"扫描失败"
	DELAY (1000) '等待 1s
	GOTO aa '扫描失败，跳转到 aa，重新扫描
	ENDIF

相关指令	SLOT_STOP， SLOT_SCAN，NODE_PDOBUFF

SLOT_STOP -- 总线停止

类型	总线指令
描述	**总线停止。** 通过 RETURN 返回成功与否。返回-1 停止成功 ，0 停止失败。 停止总线，轴使能会掉。
语法	SLOT_STOP (slot) slot：控制器 EtherCAT 槽位号或 RTEX 槽位号，0-缺省
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	aa: '标记 aa
	SLOT_SCAN(0) '总线扫描
	IF RETURN THEN '扫描成功，进入轴设置
	AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
	ATYPE(0)=65 '轴类型 65，位置控制
	DRIVE_PROFILE(0)=0 'PDO 周期扫描设置
	SLOT_START(0) '开启总线
	WHILE 1
	IF SCAN_EVENT(IN(0))>0 THEN '输入 IN 口上升沿触发
	SLOT_STOP(0) '停止总线
	ENDIF
	WEND
	ELSE
	?"扫描失败"
	DELAY (1000) '等待 1s
	GOTO aa '扫描失败，跳转到 aa，重新扫描
	ENDIF

相关指令	SLOT_START，SLOT_SCAN

?*SLOT -- 打印总线接口

类型	EtherCAT 辅助指令
描述	**查看控制器总线接口编号及类型。**
语法	?*SLOT
适用控制器	带总线接口
例子	**?*SLOT** 打印结果 Slot:0-ETHERCAT '当前只有一个 EtherCAT 总线，编号为 0

?*ETHERCAT -- 打印 EtherCAT 总线状态

类型	EtherCAT 辅助指令
描述	调试时使用，可以显示每个 NODE 的重要状态。

	总线扫描后才能显示设备状态。
语法	?*ETHERCAT
适用控制器	带 EtherCAT 接口
例子	**?*ETHERCAT** 打印结果： Slot:0 contain 1 nodes. Lostcount:0-0. Node:0 status:1 manid:7595h productid:0h axises:1 Alstate:8 Node_profile:0. BindAxis:0 Drive_profile:0 Controlword:fh drive_status:1237h Drive_mode:8h target:ffffe067h encode:ffffe068h. Slot 0 contain 1 nodes：0 槽位口共连接了 1 个设备 Lostcount 0-0：丢包数 Node：设备连接 NODE 编号 Status：设备连接状态，参考 NODE_STATUS Manid：厂商 ID Productid：设备 ID Axises：设备总轴数 AL Status：设备 OP 状态 Node_profile：设备 Profile 设置 Bindaxis：映射到控制器轴号 Drive_profile：设备收发 PDO 设置 Controlword：控制字 Drive_status：设备当前状态，参考 DRIVE_STATUS Drive_mode：设备控制模式 Target：电机位置 Encode：编码器位置
相关指令	PRINT

?*RTEX -- 打印 Rtex 总线状态

类型	Rtex 辅助指令
描述	调试时使用，可以显示每个 NODE 的重要状态。 总线启动后才能显示设备状态。
语法	?*RTEX
适用控制器	带 RTEX 接口
例子	**?*RTEX** 打印结果： Slot:0 contain 1 nodes. Lostcount:0-0. Node:0 status:1 manid:616e6150h devicetype:31h axises:1 Alstate:1. BindAxis:0 Drive_profile:0 Controlword:80h drive_status:3c1h target:fffffe5ch encode:fffffe5ch. Slot 0 contain 1 nodes：0 槽位口共连接了 1 个设备 Lostcount 0-0：丢包数 Node：设备连接 NODE 编号 Status：设备连接状态，参考 NODE_STATUS Mainid：厂商 ID Productid：设备 ID Axises：设备总轴数 AL Status：设备 OP 状态

	Node_profile：设备 Profile 设置 Bindaxis：映射到控制器轴号 Drive_profile：设备收发 PDO 设置 Controlword：控制字 Drive_status：设备当前状态，参考 DRIVE_STATUS Drive_mode：设备控制模式 Target：电机位置 Encode：编码器位置
相关指令	PRINT

16.3 SDO 操作指令

SDO_WRITE -- 数据字典写入

类型	总线指令，仅 EtherCAT 可用
描述	**通过设备号和槽位号进行 SDO 写入。** 通过 RETURN 返回成功与否，-1 写入成功，0 写入失败。 需连接好设备，扫描总线后才能执行。 只有可写的数据字典才能写入。
语法	SDO_WRITE (slot, node, index, subindex ,type ,value) slot：槽位号 0-缺省 node：设备编号 0- index：数据字典编号，前面可加"$"表示 16 进制，如$6060 subindex：子编号 type：数据类型 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 value：数据值
适用控制器	带 EtherCAT 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	SDO_WRITE(0,0,$6060,0,2,8) '0 号设备设置控制模式为 8，位置控制
	ENDIF

相关指令	SDO_WRITE_AXIS，SDO_READ

SDO_WRITE_AXIS -- 数据字典写入

类型	总线指令，仅 EtherCAT 可用

<table>
<tr><td>描述</td><td>通过轴号 SDO 写入。
通过 RETURN 返回成功与否，-1 写入成功，0 写入失败。
需连接好设备，扫描总线后才能执行。
只有可写的数据字典才能写入。</td></tr>
<tr><td>语法</td><td>SDO_WRITE_AXIS(axis, index, subindex ,type ,value)
axis：轴号
index：数据字典编号，前面可加"$"表示 16 进制，如$6060
subindex：子编号
type：数据类型
<table><tr><td>1</td><td>boolean</td></tr><tr><td>2</td><td>integer 8</td></tr><tr><td>3</td><td>integer 16</td></tr><tr><td>4</td><td>integer 32</td></tr><tr><td>5</td><td>unsigned 8</td></tr><tr><td>6</td><td>unsigned 16</td></tr><tr><td>7</td><td>unsigned 32</td></tr></table>value：数据值</td></tr>
<tr><td>适用控制器</td><td>带 EtherCAT 接口</td></tr>
<tr><td>例子</td><td>正确的连接了一个 EtherCAT 轴设备再使用例程
SLOT_SCAN(0) '总线扫描
IF NODE_COUNT(0)>0 THEN
AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
ATYPE(0)=65 '轴类型 65，位置控制
DRIVE_PROFILE(0)=0 'PDO 周期扫描设置
SDO_WRITE_AXIS(0,$6060,0,2,8) '轴 0 设备设置控制模式为 8，位置控制
ENDIF</td></tr>
<tr><td>相关指令</td><td>SDO_WRITE，SDO_READ_AXIS</td></tr>
</table>

SDO_READ -- 数据字典读取

<table>
<tr><td>类型</td><td>总线指令，仅 EtherCAT 可用</td></tr>
<tr><td>描述</td><td>通过设备号和槽位号进行 SDO 读取。
通过 RETURN 返回成功与否，-1 写入成功，0 写入失败
需连接好设备，扫描总线后才能执行。
只有可读的数据字典才能读取。</td></tr>
<tr><td>语法</td><td>SDO_READ (slot, node, index, subindex ,type，tablenum)
slot：槽位号 0-缺省
node：设备编号 0-
index：数据字典编号，前面可加"$"表示 16 进制，如$6060
subindex：子编号
type：数据类型
<table><tr><td>1</td><td>boolean</td></tr><tr><td>2</td><td>integer 8</td></tr><tr><td>3</td><td>integer 16</td></tr><tr><td>4</td><td>integer 32</td></tr><tr><td>5</td><td>unsigned 8</td></tr></table></td></tr>
</table>

	6 unsigned 16 7 unsigned 32 tablenum：读取的数据存储的 TABLE 位置
适用控制器	带 EtherCAT 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	SDO_ READ (0,0,$6061,0,2,0) '读取 0 号设备的控制模式，数据存到 table(0)
	?table(0) '打印出数据
	ENDIF

相关指令	SDO_READ_AXIS，SDO_WRITE

SDO_READ_AXIS -- 数据字典读取

类型	总线指令，仅 EtherCAT 可用
描述	**通过轴号进行 SDO 读取**。 通过 RETURN 返回成功与否，-1 写入成功，0 写入失败 需连接好设备，扫描总线后才能执行。 只有可读的数据字典才能读取。
语法	SDO_READ_AXIS(axis, index, subindex ,type，tablenum) axis：轴号 index：数据字典编号，前面可加"$"表示 16 进制，如$6060 subindex：子编号 type：数据类型 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 tablenum：读取的数据存储的 TABLE 位置
适用控制器	带 EtherCAT 接口

例子	正确的连接了一个 EtherCAT 轴设备再使用例程
	
	SLOT_SCAN(0) '总线扫描
	IF NODE_COUNT(0)>0 THEN
	AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
	ATYPE(0)=65 '轴类型 65，位置控制
	DRIVE_PROFILE(0)=0 'PDO 周期扫描设置
	SDO_WRITE_AXIS(0,$6060,0,2,8)'轴 0 设备设置控制模式为 8，位置控制
	SDO_READ_AXIS (0,$6061,0,2,0) '读取 0 号设备的控制模式，数据存到 table(0)
	?table(0) '打印出数据
	ENDIF

相关指令	SDO_READ，SDO_WRITE_AXIS

16.4 设备相关指令

NODE_COUNT -- 设备个数

类型	总线指令
描述	**通过总线连接的设备总个数。** 总线扫描后可用。
语法	只读：var = NODE_COUNT (slot) slot：槽位号，0-缺省 可以直接打印出来，见例一 可以直接作为数据，见例二
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	例一 SLOT_SCAN(0) ? **NODE_COUNT**(0)　　'打印出 0 槽口连接的设备数 例二 SLOT_SCAN(0) IF **NODE_COUNT**(0) = 3 THEN　'指定连接了多少设备，才进行下一步轴映射、轴类型设置等程序代码块 ENDIF
相关指令	NODE_INFO

NODE_STATUS -- 设备状态

类型	总线指令
描述	**设备状态，总线扫描后可用** BIT 位 | 意义 0 | 指明 node 是否存在；1- 存在 1 | 通讯状态；1- 出错； 2 | 节点状态；1- 出错； 值为 1 时，bit0 为 1，bit1 和 bit2 为 0，设备通讯正常 值为 3 时，bit0 和 bit1 为 1，bit2 为 0，设备通讯出错
语法	只读：var = NODE_STATUS (slot, node) slot：槽位号，0-缺省 node：设备编号，编号从 0 开始
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	?**NODE_STATUS**(0,0)　　'读取设备 0 的状态，打印值 1，通讯正常
	ENDIF

相关指令	NODE_INFO

NODE_AXIS_COUNT -- 设备电机数

类型	总线指令
描述	**每个设备带电机个数读取。** 必须总线扫描后才能读取。
语法	只读：var = NODE_AXIS_COUNT (slot, node) slot：槽位号，0-缺省 node：设备编号，编号从 0 开始
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	? **NODE_AXIS_COUNT** (0,0) '打印出设备 0 带电机个数
	ENDIF

相关指令	NODE_INFO

NODE_IO -- 设备 IO

类型	EtherCAT 总线指令
描述	**设备的 IO 起始编号设置，单个设备的输入输出的起始编号一样。** 设置结果只会是 8 的倍数。 必须总线扫描后才能读取。 一般用于 EIO 扩展板的 IO 设置，其他设备含有 IO 口时也可使用。
语法	可读：var = NODE_IO (slot, node) 可写：NODE_IO(slot, node)=iobase slot：槽位号，0-缺省 node：设备编号，编号从 0 开始
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	NODE_IO(0,0)=32 '设置设备 0 的 IO 起始编号为 32
	?**NODE_IO**(0,0) '打印出设备 0 的 IO 起始编号
	ENDIF

相关指令	NODE_AIO

NODE_AIO -- 设备模拟量

类型	总线指令
描述	**设备的 AIO 起始编号设置，单个设备的输入输出的起始编号一样。** 必须总线扫描后才能读取。 一般用于 EIO 扩展板的 AIO 设置，其他设备含有 AIO 口时也可使用。
语法	可读：var = NODE_AIO (slot, node[,idir])

<table>
<tr><td></td><td>可写：NODE_AIO(slot, node[,idir])=Aiobase
slot：槽位号，0-缺省
node：设备编号，编号从 0 开始
idir：AD/DA 选择
<table><tr><td>0</td><td>缺省，同时设置 AIN、AOUT；读取时只读 AIN</td></tr><tr><td>3</td><td>AIN</td></tr><tr><td>4</td><td>AOUT</td></tr></table></td></tr>
<tr><td>适用控制器</td><td>带 EtherCAT 接口或 RTEX 接口</td></tr>
<tr><td>例子</td><td>SLOT_SCAN(0)
IF NODE_COUNT(0)>0 THEN
NODE_AIO(0,0,3)=3 '设置设备 0 的 AIN 起始编号为 3
?NODE_AIO(0,0,3) '打印出设备 0 的 AIN 起始编号
ENDIF</td></tr>
<tr><td>相关指令</td><td>NODE_IO</td></tr>
</table>

NODE_INFO -- 设备信息

<table>
<tr><td>类型</td><td>EtherCAT 总线指令</td></tr>
<tr><td>描述</td><td>总线设备的信息读取。
必须总线扫描后才能读取。</td></tr>
<tr><td>语法</td><td>只读：var = NODE_INFO (slot, node, sel)
slot：槽位号，0-缺省
node：设备编号，编号从 0 开始
sel：信息编号
<table><tr><td>值</td><td>描述</td></tr><tr><td>0</td><td>VENDER，厂商编号</td></tr><tr><td>1</td><td>DEVICE，设备编号</td></tr><tr><td>2</td><td>VERSION，版本</td></tr><tr><td>3</td><td>ALIAS，别名，一般用来识别驱动器</td></tr><tr><td>4</td><td>预留</td></tr></table>
<table><tr><td>IO 的个数</td><td>描述</td></tr><tr><td>10</td><td>IN 个数</td></tr><tr><td>11</td><td>OP 个数</td></tr><tr><td>12</td><td>AIN 个数</td></tr><tr><td>13</td><td>AOUT 个数</td></tr><tr><td>14</td><td>预留</td></tr></table></td></tr>
<tr><td>适用控制器</td><td>带 EtherCAT 接口或 RTEX 接口</td></tr>
<tr><td>例子</td><td>SLOT_SCAN(0)
IF NODE_COUNT(0)>0 THEN
?NODE_INFO(0,0,10) '读取设备 0 的输入 IN 个数
?NODE_INFO(0,0,0) '读取设备 0 的厂商编号
?NODE_INFO(0,0,11) '读取设备 0 的输出 OP 个数
ENDIF</td></tr>
<tr><td>相关指令</td><td>SLOT_SCAN</td></tr>
</table>

NODE_PROFILE -- PDO 预设置

类型	总线指令
描述	**总线设备的 profile 设置。** 预留，总线启动后不能再修改。
语法	可读：var= NODE_PROFILE(slot,node) 可写：NODE_PROFILE(slot, node) = iprofile[, reserve]
适用控制器	带 EtherCAT 接口或 RTEX 接口

NODE_PDOBUFF -- 特殊设备 PDO 设置

类型	总线指令
描述	**特殊 EtherCAT 设备的 PDO 支持。** 非轴和 IO 的设备，通过这个指令来读写 PDO，例如电源设备。 轴和 IO 类型的设备，已经可以通过轴参数和 IO 指令来访问 PDO，不能使用这个指令。 SLOT_START 会自动提前读取当前的 PDO 列表，以及可写 PDO 的当前值。 可以在 SLOT_START 调用前通过 SDO 修改 PDO 列表，或相关数据字典的当前值。 需要启动以后才能修改，可以先用 SOD_START 启动到 SAFEOP，然后设置初始化 PDO 状态.，再启动到 OP。
语法	命令语法：NODE_PDOBUFF (slot, node, index, subindex ,type) 函数语法：Buff = NODE_PDOBUFF (slot, node, index, subindex ,type) slot：槽位号 0-缺省 node：设备编号 0- index：数据字典编号，前面可加"$"表示 16 进制，如$6060 subindex：子编号 type：数据类型
适用控制器	带 EtherCAT 接口，4 系列产品, 20170508 以上版本支持
例子	>>**NODE_PDOBUFF**(0,0, $6040, 0, 3) = 15 >>?**NODE_PDOBUFF**(0,0, $6041, 0, 3)
相关指令	SDO_WRITE，SDO_READ

1	boolean
2	integer 8
3	integer 16
4	integer 32
5	unsigned 8
6	unsigned 16
7	unsigned 32

NODE_PRESET -- 设备预配置

类型	EtherCAT 总线指令
描述	**总线设备预先配置，配置以后没有挂上外设也提前启动总线。** 总线启动后不能再修改。 预设置后，可以通过 NODE_STATUS 判断外设是否挂上。 预设值的类型与实际不一样，将不能启动总线。 中途连入的外设没有预设值，也没有扫描到，也不能启动总线。 固件 20160601 以上版本支持。
语法	命令语法 1：NODE_PRESET (slot, node, manuid, productid) 命令语法 2：NODE_PRESET (slot, -1) 清除所有预设置 函数语法 1: VALUE = NODE_PRESET (slot, node) 返回是否有预设置 函数语法 1: VALUE = NODE_PRESET (slot) 返回预设值的最大个数 slot：槽位号，0-缺省 node：设备编号，0- manuid：厂商 ID 编号，参考 NODE_INFO productid：设备 ID 编号，参考 NODE_INFO
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	**NODE_PRESET**(0,-1)'清除原来的预设置 **NODE_PRESET**(0,0, $83, 5) '设置第一个 NODE 为 OMRON 驱动器 SLOT_SCAN(0) ?"SCAN RESULT:",RETURN, "MAX", NODE_COUNT(0) '此时会自动显示总数为 1 FOR i= 0 TO NODE_COUNT(0) -1 ? "node", i ? "status",NODE_STATUS(0,i) ? "manu:",NODE_INFO(0,i,0) ? "dev:",NODE_INFO(0,i,1) ? "motor:", NODE_AXIS_COUNT(0,i) NEXT
相关指令	NODE_STATUS

16.5 驱动器相关指令

DRIVE_MODE -- 驱动器模式

类型	轴参数
描述	**驱动器的控制模式，对应数据字典 0x6060。** 必须设置正确的 ATYPE（设置为 65/66/67）以后才能操作这个参数。
语法	可读：var=DRIVE_MODE (axis) 可写：DRIVE_MODE (axis)= value axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	…　　'轴使能过程，参考第十七章简易例程
	IF NODE_COUNT(0)>0 THEN
	　DRIVE_MODE(0)=8　　'轴 0 设备设置为位置控制模式
	　? **DRIVE_MODE**(0)　　'打印轴 0 的控制模式
	ENDIF

DRIVE_PROFILE -- 驱动器 PDO 设置

类型	EtherCAT 轴参数

描述	**每个轴的发送 pdo 接收 pdo 的配置选择。**
	必须设置正确的 ATYPE（设置为 65/66/67）以后才能操作这个参数。
	详细配置请咨询厂家。
	
	EtherCAT 总线
	-1 表示使用驱动器的内置缺省 PDO 列表，20160601 以上版本支持，缺省 PDO 不带 0X6060 时，无法使用 datum(21)回零指令。
	
	-1-驱动器默认设置，需要控制器版本 20160601 及以上
	
	0-缺省配置，csp 位置模式
	{0x60400010, 0x607a0020, 0x60600008},
	//控制字　　目标位置　　模式
	{0x60410010, 0x60640020},
	//状态字　　反馈位置
	
	1-csp 位置模式+力矩反馈
	{0x60400010, 0x607a0020, 0x60600008},
	//控制字　　目标位置　　模式
	{0x60410010, 0x60640020, 0x60770010},
	//状态字　　反馈位置　　当前力矩
	
	2-csp 位置模式+力矩反馈+锁存 1up
	{0x60400010, 0x607a0020, 0x60b80010, 0x60600008},
	//控制字　　目标位置　　probe 设置　　模式
	{0x60410010, 0x60640020, 0x60770010, 0x60b90010, 0x60ba0020},
	//状态字　　反馈位置　　当前力矩　　probe 状态　　probe 位置
	
	3-csp 位置模式+力矩限制+力矩反馈+锁存 1 上升沿
	{0x60400010, 0x607a0020, 0x60b80010, 0x60720010, 0x60600008},
	//控制字　　目标位置　　probe 设置　　力矩限制　　模式
	{0x60410010, 0x60640020, 0x60770010, 0x60b90010, 0x60ba0020},
	//状态字　　反馈位置　　当前力矩　　probe 状态　　probe 位置
	
	4-csp 位置模式+力矩反馈+驱动器 IO 输入
	{0x60400010, 0x607a0020, 0x60600008},
	//控制字　　目标位置　　模式
	{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
	//状态字　　反馈位置　　当前力矩　　驱动器 IO 输入

5-csp 位置模式+力矩反馈+驱动器 IO 输出+驱动器 IO 输入
{0x60400010, 0x607a0020, 0x60fe0120,0x60600008},
//控制字　　目标位置　　IO 输出　　模式
{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
//状态字　　反馈位置　　当前力矩　驱动器 IO 输入

6-特殊驱动器专用

7-特殊驱动器专用

8-特殊驱动器专用

9-固件版本 160504 支持
{0x60400010,0x607a0020,0x60fe0120,0x60b80010,0x60720010,0x60600008},
//控制字　　目标位置　IO 输出(32 个)　probe 设置　力矩限制　模式
{0x60410010,0x60640020,0x60770010,0x60fd0020,x60b90010,0x60ba0020},
//状态字　　反馈位置　　当前力矩　　驱动器 IO 输入(32 个)　probe 状态　probe 位置

10-固件版本 160504 以上支持,加 drive_fe 部分
{0x60400010,0x607a0020,0x60fe0120,0x60b80010,0x60720010,0x60600008},
//控制字　　目标位置　　IO 输出　　probe 设置　力矩限制　　模式
{0x60410010,0x60640020,0x60770010,0x60fd0020,0x60b90010,0x60ba0020,0x60f40020},
//状态字　　反馈位置　　当前力矩　驱动器 IO 输入　probe 状态　probe 位置　drive_fe

11-固件版本 160504 以上支持,probe 专用测试
{0x60400010, 0x607a0020, 0x60b80010, 0x60600008},
//控制字　　目标位置　　probe 设置　　模式
{0x60410010,0x60640020,0x60770010,0x60b90010,0x60ba0020,0x60bb0020,　0x60bc0020,
0x60bd0020},
//状态字　　反馈位置　　当前力矩　　probe 状态　probe 位置 1/位置 2/位置 3/位置 4

12-固件版本 160504 以上支持,特殊驱动器专用
{0x60400010, 0x607a0020, 0x60600008},
//控制字　　目标位置　　模式
{0x60410010, 0x60640020, 0x60fd0020},
//状态字　　反馈位置　　驱动器 IO 输入

13-固件版本 160504 以上支持,带速度前馈与加速度前馈
{0x60400010, 0x60B20010, 0x607a0020, 0x60B10020, 0x60600008},
//控制字　　加速度前馈　目标位置　　速度前馈　　模式
{0x60410010, 0x60640020, 0x60770010, 0x60fd0020, 0x606c0020},
//状态字　　反馈位置　　当前力矩　驱动器 IO 输入　实际速度

17-固件版本 160504 以上支持，csp/csv/cst 三种模式可以切换
{0x60400010,0x60710010,0x60ff0020,0x607a0020,0x60b80010,0x60720010, x60600008},
//控制字　　周期力矩　　周期速度　目标位置　probe 设置　力矩限制　　模式
{0x60410010,0x60770010,0x60640020,0x60fd0020,0x60b90010,0x60ba0020, x60bb0020},
//状态字　　当前力矩　　反馈位置　驱动器 IO 输入　probe 状态　probe 位置 1/位置 2/

18-固件版本 160504 以上支持,csp/csv/cst 三种模式可以切换+力矩反馈读取
{0x60400010,0x60710010,0x60ff0020,0x607a0020,0x60b80010,0x60720010,0x60600008},
//控制字　　周期力矩　　周期速度　目标位置　　probe 设置　力矩限制　　模式
{0x60410010,0x60770010,0x60640020,0x60fd0020,0x60b90010,0x60ba0020,0x60bb0020,

0x60bc0020, 0x60bd0020},
//状态字 当前力矩 反馈位置 驱动器 IO 输入 probe 状态 probe 位置 1/位置 2/ probe 位置 3/位置 4

20-固件版本 160504 以上支持,csp 位置+csv 速度
{0x60400010, 0x60ff0020, 0x607a0020, 0x60600008},
//控制字 目标速度 目标位置 模式
{0x60410010, 0x60640020},
//状态字 反馈位置

21-固件版本 160504 以上支持,csp 位置+csv 速度+力矩反馈
{0x60400010,0x60ff0020,0x607a0020,0x60600008},
//控制字 目标速度 目标位置 模式
{0x60410010,0x60640020,0x60770010},
//状态字 反馈位置 当前力矩

22-固件版本 160504 以上支持,csp 位置+csv 速度+力矩反馈+色标锁存 1 上升沿
{0x60400010, 0x60ff0020, 0x607a0020, 0x60b80010, 0x60600008},
//控制字 目标速度 目标位置 probe 设置 模式
{0x60410010, 0x60640020, 0x60770010, 0x60b90010, 0x60ba0020},
//状态字 反馈位置 当前力矩 probe 状态 probe 位置

23-固件版本 160504 以上支持,csp 位置+csv 速度+力矩反馈+锁存 1 上升沿+力矩限制
{0x60400010,0x60ff0020,0x607a0020,0x60b80010,0x60720010,0x60600008},
//控制字 目标速度 目标位置 probe 设置 力矩限制 模式
{0x60410010,0x60640020, 0x60770010,0x60b90010,0x60ba0020},
//状态字 反馈位置 当前力矩 probe 状态 probe 位置

24-固件版本 160504 以上支持,csp 位置+csv 速度+IO 输入+位置+力矩反馈
{0x60400010, 0x60ff0020, 0x607a0020, 0x60600008},
//控制字 目标速度 目标位置 模式
{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
//状态字 反馈位置 当前力矩 驱动器 IO 输入

25-固件版本 160504 以上支持,csp 位置+csv 速度+IO 输入+位置+力矩反馈
{0x60400010, 0x60ff0020, 0x607a0020, 0x60fe0120,0x60600008},
//控制字 目标速度 目标位置 驱动器 IO 输出 模式
{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
//状态字 反馈位置 当前力矩 驱动器 IO 输入

30-固件版本 160504 以上支持,csp 位置+cst 转矩
{0x60400010, 0x60710010, 0x607a0020, 0x60600008},
//控制字 目标转矩 目标位置 模式
{0x60410010, 0x60640020},
//状态字 反馈位置

31-固件版本 160504 以上支持,csp 位置+cst 转矩+力矩反馈
{0x60400010,0x60710010,0x607a0020,0x60600008},
//控制字 目标力矩 目标位置 模式
{0x60410010,0x60640020,0x60770010},
//状态字 反馈位置 当前力矩

32-固件版本 160504 以上支持,csp 位置+cst 转矩+力矩反馈+锁存 1 上升沿
{0x60400010, 0x60710010, 0x607a0020, 0x60b80010 , 0x60600008},

	//控制字 目标转矩 目标位置 probe 设置 模式
	{0x60410010, 0x60640020, 0x60770010, 0x60b90010, 0x60ba0020},
	//状态字 反馈位置 当前力矩 probe 状态 probe 位置
	
	33-固件版本 160504 以上支持,csp 位置+cst 转矩+力矩反馈+锁存 1 上升沿+ 力矩限制
	{0x60400010,0x60710010,0x607a0020,0x60b80010,0x60720010,0x60600008},
	//控制字 目标转矩 目标位置 probe 设置 力矩限制 模式
	{0x60410010, 0x60640020, 0x60770010, 0x60b90010, 0x60ba0020},
	//状态字 反馈位置 当前力矩 probe 状态 probe 位置
	
	34-固件版本 160504 以上支持,csp 位置+cst 转矩+力矩反馈+驱动器 IO 输入
	{0x60400010, 0x60710010, 0x607a0020, 0x60600008},
	//控制字 目标转矩 目标位置 模式
	{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
	//状态字 反馈位置 当前力矩 驱动器 IO 输入
	
	35-固件版本 160504 以上支持,csp 位置+cst 转矩+力矩反馈+IO 输入+IO 输出
	{0x60400010, 0x60710010, 0x607a0020, 0x60fe0120,0x60600008},
	//控制字 目标转矩 目标位置 驱动器 IO 输出 模式
	{0x60410010, 0x60640020, 0x60770010, 0x60fd0020},
	//状态字 反馈位置 当前力矩 驱动器 IO 输入
	
	Rtex 总线
	0- 不带 IO 映射
	1- 带驱动器 IO 映射
	带 IO 映射时，按 DRIVE_IO 指令设置起始地址

语法	可读：var= DRIVE_ PROFILE(axis) 可写：DRIVE_ PROFILE(axis)= value axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	AXIS_ADDRESS(1)=1
	ATYPE(1)=65
	DRIVE_ PROFILE(1)=-1 '轴 1 PDO 设置为-1，设备默认值
	? **DRIVE_ PROFILE**(1) '打印出轴 1 的 PDO 设置
	ENDIF

相关指令	ATYPE，NODE_PROFILE

DRIVE_CW_MODE -- 驱动器设置

类型	轴参数
描述	**驱动器设置参数** 必须设置正确的 ATYPE（设置为 65/66/67）以后才能操作这个参数。 部分版本为使用方便，可以直接操作 MODE。 RTEX 的控制字不要随便修改。 0-控制器自动调整 DRIVE_CONTROLWORD 参数，此时 DRIVE_CONTROLWORD 指令

	无效。 1- 可以手动调整，此时可以使用 DRIVE_CONTROLWORD 指令。
语法	可读：var= DRIVE_CW_MODE(axis) 可写：DRIVE_CW_MODE(axis)=value axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口
相关指令	ATYPE，DRIVE_CONTROLWORD

DRIVE_CONTROLWORD -- 驱动器控制字

类型	轴参数
描述	**驱动控制字，按位操作** 必须设置正确的 ATYPE（设置为 65/66/67）以后才能操作这个参数。 对于 EtherCAT 驱动器， 对应数据字典 0x6040 EtherCAT 控制器 ATYPE=65 时控制字会根据 WDOG/AXIS_ENABLE 自动切换以使能驱动器，主要位操作如下图。具体位意义请查看对应驱动器手册。

command	bits of the controlword					PDS transitions
	bit 7	bit 3	bit 2	bit 1	bit 0	
	fault reset	enable operation	quick stop	enable voltage	switch on	
shut down	0	-	1	1	1	2,6,8
switch on	0	0	1	1	1	3
switch on+enable opetation	0	1	1	1	1	3+4（*1）
enable opetation	0	1	1	1	1	4,16
disable voltage	0	-	-	0	-	7,9,10,12
quick stop	0	-	0(*2)	1	-	7,10,11
disable operation	0	1	1	1	1	5
fault reset	↑	-	-	-	-	15

对于 Rtex 驱动器
Rtex 控制器控制字缺省自动设置，需要手动时先将 DRIVE_CW_MODE 设为 1，不能随便修改，位意义如下图，详细说明请看松下 Rtex 手册的 4-2-3 章节。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Servo_On	0	0	Gain_SW	TL_SW	HM_Ctrl	0	0
bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
Hard_Stop	Smooth_Stop	Pause	0	SL_SW	0	EX-OUT2	EX-OUT1

语法	可读：var=DRIVE_CONTROLWORD(axis) 可写：DRIVE_CONTROLWORD(axis)=value axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	例一 EtherCAT
	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	AXIS_ADRESS(0)=1
	ATYPE(0)=65 'EtherCAT 总线位置控制
	DRIVE_PROFILE(0)=0 'PDO 设为 0
	DRIVE_CONTROLWORD(0)=128 '伺服错误清除
	DELAY (100)
	DRIVE_CONTROLWORD(0)=6 '伺服 shutdown
	DELAY (100)
	DRIVE_CONTROLWORD(0)=15 '伺服 switch on
	ENDIF
	
	例二 RTEX
	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	AXIS_ADRESS(0)=1
	ATYPE(0)=50 'RTEX 总线位置控制
	DRIVE_PROFILE(0)=1 '伺服带 IO
	DRIVE_CW_MODE=1 '手动设置控制字
	DRIVE_CONTROLWORD(0)=128 '伺服使能
	ENDIF

相关指令	ATYPE，DRIVE_CW_MODE

DRIVE_STATUS -- 驱动器状态

类型	轴状态
描述	**驱动器的当前状态，按位判断状态。** 必须设置正确的 ATYPE（EtherCAT 总线设置为 65/66/67，RTEX 总线设置为 50/51/52）以后才能读取。 对于 EtherCAT 驱动器，对应数据字典 6041 根据此 bit 可以确认 PDS 的状态，以下表示状态和对应的 bit

状态字	PDS state	
xxxx xxxx x0xx 0000 b	not ready to switch on	初始化 未完成状态
xxxx xxxx x1xx 0000 b	switch on disabled	初始化 完成状态
xxxx xxxx x01x 0001 b	ready to switch on	主电路电源 off 状态
xxxx xxxx x01x 0011 b	switch on	伺服使能 off/伺服准备
xxxx xxxx x01x 0111 b	operation enabled	伺服使能 on
xxxx xxxx x00x 0111 b	quick stop active	快速停止
xxxx xxxx x0xx 1111 b	fault reaction active	异常（报警）判断
xxxx xxxx x0xx 1000 b	fault	异常（报警）状态

其他位意义请查看对应驱动器手册说明。

对于 Rtex 驱动器
Rtex 驱动器的状态字位意义如下图，详细含义请查看松下 Rtex 手册第 4-2-3 章节。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Servo_ Active	Servo_ Ready	Alarm	Warning	Torque_Limited	Homing_Complete	In_Progress	In_Position
bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
SI-MON5 /E-STOP	SI-MON4/ EX-SON	SI-MON3/E XT3/STOP	SI-MON2/E XT2/RET	SI-MON1/ EXT1	HOME	POT/NOT	NOT/POT

语法	只读：toq = DRIVE_STATUS (axis) axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	ATYPE(0)=65 '总线位置控制
	? **DRIVE_STATUS** (0) '打印轴 0 设备的状态
	ENDIF

相关指令	ATYPE，DRIVE_PROFILE

DRIVE_IO -- 驱动器 IO

类型	轴参数
描述	**直接配置 DRIVE 的 IO 的起始编号，输入输出都是同样的编号。** DRIVE_PROFILE 支持读取驱动器 IO 时起作用。
语法	可读：var= DRIVE_IO (axis) 可写：DRIVE_IO (axis)=value

<table>
<tr><td></td><td>axis：轴号

EtherCAT 总线伺服的输入输出点功能及地址参考数据字典 60FD，60FE（某些厂家不是按照标准协议地址，请查看对应驱动器手册）

Rtex 总线伺服的输入输出点功能对应控制器的地址=DRIVE_IO+编号
<table>
<tr><td>DRIVE_IO+以下编号</td><td>伺服功能</td></tr>
<tr><td colspan="2">输入点</td></tr>
<tr><td>0</td><td>NOT/POT</td></tr>
<tr><td>1</td><td>POT/NOT</td></tr>
<tr><td>2</td><td>HOME</td></tr>
<tr><td>3</td><td>SI-MON1/EXT1</td></tr>
<tr><td>4</td><td>SI-MON2/EXT2</td></tr>
<tr><td>5</td><td>SI-MON3/EXT3</td></tr>
<tr><td>6</td><td>SI-MON4/EX-SON</td></tr>
<tr><td>7</td><td>SI-MON5/E-STOP</td></tr>
<tr><td colspan="2">输出点</td></tr>
<tr><td>0</td><td>EX-OUT1</td></tr>
<tr><td>1</td><td>EX-OUT2</td></tr>
</table></td></tr>
<tr><td>适用控制器</td><td>带 EtherCAT 接口或 RTEX 接口</td></tr>
<tr><td>例子</td><td>SLOT_SCAN(0)
...　　'轴使能过程，参考第十七章简易例程
IF NODE_COUNT(0)>0 THEN
　DRIVE_IO(1)=32　　'轴 1 设备的 IO 起始编号设为 32
　DIM var　　'定义变量 var
　var=DRIVE_IO(1)　　'var 赋值为轴 1 设备的 IO 起始编号
　?var　　'直接打印出轴 1 设备的 IO 起始编号
ENDIF</td></tr>
<tr><td>相关指令</td><td>NODE_IO，DRIVE_PROFILE</td></tr>
</table>

DRIVE_TORQUE -- 驱动器力矩

类型	EtherCAT 轴状态
描述	**驱动器的当前力矩。** 必须设置正确的 ATYPE（设置为 65/66/67）与 DRIVE_PROFILE 以后才能读取。
语法	只读：var = DRIVE_TORQUE(axis) axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	　ATYPE(0)=65　　'总线位置控制
	　DRIVE_PROFILE(0)=1　　'pdo 设为 1，含有返回力矩功能
	　? **DRIVE_TORQUE** (0)　　'打印轴 0 的力矩
	ENDIF

相关指令	ATYPE，DRIVE_PROFILE

DRIVE_FE -- 驱动器误差

类型	轴状态
描述	**驱动器的当前误差超限，对应数据字典 0x60F4。** 必须设置正确的 ATYPE（设置为 65/66/67）以后才能设置。
语法	只读：var = DRIVE_FE (axis) axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口

例子	SLOT_SCAN(0) '总线扫描
	IF NODE_COUNT(0)>0 THEN
	AXIS_ADDRESS(0)=1 '第一个驱动器映射到轴 0
	ATYPE(0)=65 '轴类型 65，位置控制
	DRIVE_PROFILE(0)=0 'PDO 周期报文设置，根据 DRIVE_PROFILE
	?DRIVE_FE (0) '读取轴 0 设备的跟踪误差值
	ENDIF

相关指令	DRIVE_FE_LIMIT

DRIVE_FE_LIMIT -- 驱动器误差限制

类型	EtherCAT 轴参数
描述	**驱动器的当前误差超限设置。** 必须设置正确的 ATYPE（设置为 65/66/67）以后才能设置。预留。
语法	可读：var=DRIVE_FE_LIMIT(axis) 可写：DRIVE_FE_LIMIT(axis)= value axis：轴号
适用控制器	带 EtherCAT 接口或 RTEX 接口
相关指令	DRIVE_FE

DRIVE_CLEAR -- 清除报警

类型	总线指令
描述	**操作当前 BASE 轴，清除驱动器报警。** 通过 RETURN 返回成功与否。 驱动器没有错误时，使用指令控制器会警告 6015，不影响程序运行。
语法	BASE(轴号) DRIVE_CLEAR(para) para： 值 | 描述 0 | 清除当前告警 1 | 清除历史告警

	2	清除外部输入告警
适用控制器	带 EtherCAT 接口或 RTEX 接口	

例子	SLOT_SCAN(0)
	IF NODE_COUNT(0)>0 THEN
	BASE(0) '选择轴 0
	DRIVE_CLEAR(0) '清除当前告警
	ENDIF	

相关指令	DRIVE_READ，DRIVE_WRITE	

DRIVE_READ -- 参数读取

类型

总线指令，仅 Rtex 控制器可用

描述

操作当前 BASE 轴，读取驱动器参数。

通过 RETURN 返回成功与否。

语法

DRIVE_READ(para [,vr_index])

para：

值	描述
1	参数分类*256+参数编号（Pr7.20=7*256+20）
2	参数=130,读取钳位的状态,BIT0,BIT1 表示两个通道的状态
3	参数=$10000+(ssid)读取 RTEX 驱动器系统信息，字符串存储在 VRSTRING
4	参数=$20000+(报警功能码)+($1000*索引)读取报警信息
5	参数=$30000+(监视功能码)+($1000*索引)读取监视器信息

vr_index：读取数据存储到 vr 里面，若没有，直接在输出栏打印出来

以下为常用参数

1.伺服参数

参数	功能	值
Pr0.00	设置电机转向	0：CW 1：CCW
Pr0.01	控制模式设置	一般设置为 0：半闭环控制
Pr0.08	电机一圈脉冲数	0-8388608（根据实际电机）
Pr0.09	齿轮比分子设定	0-1073741824
Pr0.10	齿轮比分母设定	1-1073741824
Pr4.01	正限位信号设置	常闭：00818181h（8487297） 常开：00010101h（65793）
Pr4.02	负限位信号设置	常闭：00828282h（8553090） 常开：00020202h（131586）
Pr4.03	Home 信号设置	常闭：00A2A2A2h（10658466） 常开：00222222h（2236962）

2.Rtex 通讯参数

Pr7.20	Rtex 通讯周期	-1：将 Pr7.91 的设定生效 3：0.5ms 6：1.0ms
Pr7.21	Rtex 指令更新周期比	1：1 倍 2：2 倍
Pr7.91	Rtex 通讯周期扩展	62500 ns

		125000 ns 250000 ns 500000 ns 1000000 ns 2000000 ns

3.驱动器系统信息

SSID	含义
$01	厂商名
$05	设备类型
$12	驱动器型号
$13	驱动器序列号
$14	驱动器软件版本
$15	驱动器类型
$22	电机型号
$23	电机序列号

4.报警信息

功能码	功能	索引
$000	读取当前报警/报警履历	0：本次的报警码 1：上次的报警码 2：前两次的报警码 ... 14：前 14 次的报警码
$001	清除当前报警	0：清除本次的报警
$011	清除所有报警	0：清除报警履历
$021	外部位移传感器的错误清除	0：通过串行通信类型的外部位移传感器清除箝位的错误。 进行外部位移传感器的错误清除后，请断开控制电源后重启。

5.监视器

功能码	功能	索引
$01	位置偏差，指令单位	0：滤波后的指令的偏差
$02	编码器分辨率，脉冲/转	0：电机编码器分辨率
$04	指令位置，指令单位	0：滤波后的内部指令位置
$05	实际速度，Pr7.25 单位	0：电机实际速度
$06	内部指令转矩，0.1%	0：到电机的指令转矩
$07	实际位置，指令单位	0：电机实际的位置
$09	箝位位置 1，指令单位	0：CH1 箝位的电机的实际位置
$0A	箝位位置 2，指令单位	0：CH2 箝位的电机的实际位置

适用控制器

带 RTEX 接口

例子

总线开启后才能使用

例一

```
IF NODE_COUNT(0)>0 THEN
  BASE(0)                          '选择轴 0
  DRIVE_READ(7*256+11,0)    '读取 Pr7.11 参数数值，保存到 vr(0)
  ?vr(0)                           '打印
ENDIF
```

	例二
	IF NODE_COUNT(0)>0 THEN
	  BASE(0) '选择轴 0
	  DRIVE_READ($10000+$01) '读取厂商名，直接打印出来
	ENDIF

相关指令	DRIVE_WRITE，DRIVE_CLEAR

DRIVE_WRITE -- 参数写入

类型	总线指令，仅 Rtex 控制器可用
描述	**操作当前 BASE 轴，驱动器参数修改。** 通过 RETURN 返回成功与否。
语法	DRIVE_WRITE(para,value) para： 值：1 描述：参数分类*256 + 参数编号（Pr7.20=7*256+20） 值：2 描述：特殊参数=128，当前参数写入 EEPROM（此时 value=1） 值：3 描述：特殊参数=$40000 + typecode + (设置值*256)，使用驱动器的回零箝位功能 value：参数值

1.伺服参数

伺服参数	功能	值
Pr0.00	设置电机转向	0：CW 1：CCW
Pr0.01	控制模式设置	一般设置为 0：半闭环控制
Pr0.08	电机一圈脉冲数	0-8388608（根据实际电机）
Pr0.09	齿轮比分子设定	0-1073741824
Pr0.10	齿轮比分母设定	1-1073741824
Pr4.01	正限位信号设置	常闭：00818181h（8487297） 常开：00010101h（65793）
Pr4.02	负限位信号设置	常闭：00828282h（8553090） 常开：00020202h（131586）
Pr4.03	Home 信号设置	常闭：00A2A2A2h（10658466） 常开：00222222h（2236962）
转矩相关		
Pr0.13	第一转矩限制	0~500%
Pr5.21	转矩限制选择 转矩控制时，固定为 Pr0.13（第 1 转矩限制）。	如下图所示

设定值	TL_SW=0		TL_SW=1	
	负方向	正方向	负方向	正方向
0，[1]	Pr0.13			
2	Pr5.22	Pr0.13	Pr5.22	Pr0.13
3	Pr0.13		Pr5.22	
4	Pr5.22	Pr0.13	Pr5.22	Pr5.25

Pr5.22	第二转矩限制	0~500%
Pr5.25	正方向转矩限制	0~500%
Pr5.26	负方向转矩限制	0~500%
速度相关		
Pr3.12	加速时间设置	0~10000ms （达到 1000.r/min）
Pr3.13	减速时间设置	0~10000ms （达到 1000.r/min）
Pr3.14	S 加减速设置	0~1000ms
Pr3.17	速度限制选择 转矩控制时的速度限制值的方式选择	设定值 | SL_SW 0 | 1 [0] | Pr3.21 1 | Pr3.21 | Pr3.22 设置为 1 时，根据 RTEX 通信指令 SL_SW 的数值进行选择
Pr3.21	速度限制值 1	0~20000r/min
Pr3.22	速度限制值 2	0~20000r/min

2.Rtex 通讯参数

Pr7.20	Rtex 通讯周期	-1：将 Pr7.91 的设定生效 3：0.5ms 6：1.0ms
Pr7.21	Rtex 指令更新周期比	1：1 倍 2：2 倍
Pr7.91	Rtex 通讯周期扩展	62500 ns 125000 ns 250000 ns 500000 ns 1000000 ns 2000000 ns

3.回零箝位模式

typecode	描述
$50	位置箝位状态监视器
$51	位置箝位 1 启动
$52	位置箝位 2 启动
$53	位置箝位 1,2 启动
$54	位置箝位 1 解除
$58	位置箝位 2 解除
$5c	位置箝位 1,2 解除

4.设置值

设置值=钳位 1 设置 +($10* 钳位 2 设置)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LATCH_SEL2				LATCH_SEL1			

箝位 1/2 设置	描述
$0	Z 相信号触发
$1	EXT1 的逻辑上升沿
$2	EXT2 的逻辑上升沿

	$3	EXT3 的逻辑上升沿
	$9	EXT1 的逻辑下降沿
	$10	EXT2 的逻辑下降沿
	$11	EXT3 的逻辑下降沿
适用控制器	支持 RTEX 的控制器	

例子	总线开启后才能使用
	例一
	IF NODE_COUNT(0)>0 THEN
	BASE(0) '选择轴 0
	DRIVE_WRITE(7*256+11,6) '写 Pr7.11 参数为 6
	DRIVE_READ(7*256+11,0) '读取 Pr7.11 参数数值，保存到 vr(0)
	?vr(0) '打印，打印值 6
	ENDIF
	
	例二
	IF NODE_COUNT(0)>0 THEN
	BASE(0) '选择轴 0
	DRIVE_WRITE(128,1) '将修改的参数写入 EEPROM
	ENDIF	

相关指令	DRIVE_READ，DRIVE_CLEAR	

第十七章 简易例程

常用操作

IO 操作

```
WHILE 1                                  '循环检测输入信号
    IF   IN(0) = ON   THEN           '输入口 0 有效
      OP(2, OFF)                    '关闭输出口 2
    ELSE
      OP(2, ON)                     '打开输出口 2
    ENDIF
WEND
```

SP 指令连续插补

```
ERRSWITCH = 3          '全部信息输出
BASE(0,1,2,3)        '选择  X Y Z U
RAPIDSTOP(2)
WAIT IDLE

DPOS = 0,0,0,0
ATYPE=1,1,1,1                        '脉冲方式步进或伺服
UNITS = 100,100,100,100              '脉冲当量，每 mm100 脉冲
SPEED = 200,200,200,200              '此速度会对 FORCE_SPEED 进行限制
ACCEL = 2000,2000,2000,2000          '加速度设置
DECEL = 2000,2000,2000,2000          '减速度设置
MERGE = ON                           '启动连续插补
CORNER_MODE = 0              '不启动自动拐角减速，手动设置 STARTMOVE_SPEED，ENDMOVE_SPEED
'DECEL_ANGLE = 15 * (PI/180)          '开始减速的角度  15 度
'STOP_ANGLE = 45 * (PI/180)           '降到最低速度的角度 45 度

WHILE 1      '循环运动
    IF IN(0) = ON THEN                '输入 0 有效启动运动
        TRACE "start movesp"
        '走一个方框，每一段的速度和停止速度都不一样
        FORCE_SPEED = 100                '第一段速度 100
        ENDMOVE_SPEED = 10               '第一段结束的速度 10
        MOVESP(100,0)

        FORCE_SPEED = 150                '第二段速度 150
        ENDMOVE_SPEED = 15
        STARTMOVE_SPEED = 15  '第二段的起始速度 15，因为高于第一段的结束速度 10，因此实际的
                              起始速度为 10
        MOVESP(0,100)
```

```
        FORCE_SPEED = 200              '第三段速度 200
        ENDMOVE_SPEED = 20
        STARTMOVE_SPEED = 20  '第三段的起始速度 20， 因为高于第二段的结束速度 15，因此实际的
                                起始速度为 15
        MOVESP(-100,0)

        FORCE_SPEED = 300              '第四段速度 300，受 SPEED 限制其实为 200
        ENDMOVE_SPEED = 30
        STARTMOVE_SPEED = 30     '第三段的起始速度 30，因为高于第二段的结束速度 20，因此实际
                                的起始速度为 20
        MOVESP(0,-100)
        WAIT IDLE                     '等待运动停止
        DELAY(100)                    '延时
    ENDIF
WEND
END
```

字符串与数据相互转化

```
DIM val1,val2,array1(15)          '定义变量和数组
val1=1234                         '变量 1 赋值 1234
FOR i=0 TO 14
    array1(i)=0                   '清空数组
NEXT
?val1,val2                        '打印两个变量确认值
?*array1                        '打印数组确认值

array1=TOSTR(val1)           '数据转成字符串，赋值给数组
?*array1                          '再次打印数组确认
array1=TOSTR(val1)+"1asf"   'tostr 也可与其他字符串合并
?*array1

val2=val(array1)              '字符串转化为数据赋值给变量 2
?val2                             '打印变量 2 确认
'只有数字字符可以来回转化，子母、符号字符不可以
```

手轮

手轮就是一个编码器，通常用来对点校准工件位置坐标。手轮运动就是类似与机械齿轮传动的运动，通过不同的比例线性运动。

```
ERRSWITCH = 3                     '全部信息输出
CONST AXISHAND = 0
BASE(AXISHAND)                    '选择第 0 轴接手轮
ATYPE=6                           '脉冲+方向的手轮，正交输入手轮使用 3
BASE(1)                           '轴 1 被手轮控制
ATYPE=1                           '步进
DPOS = 0
```

```
UNITS = 100                          '脉冲当量，每 mm100 脉冲
SPEED = 200
ACCEL = 3000
DECEL = 3000
SRAMP = 20
CLUTCH_RATE = 0                      '采用速度加速度来进行限制

DIM POSLAST                          '记录上一个位置变量
POSLAST = DPOS
WHILE 1                              '手动选择手轮连接倍率
    IF IN(0) = ON AND IN(1) = OFF THEN
        CONNECT(1, AXISHAND)     '链接到轴 0，倍率  1
    ELSEIF IN(1) = ON AND IN(0) = OFF THEN
        CONNECT(10, AXISHAND)        '链接到轴 0，倍率  10
    ELSEIF IN(0)= ON AND IN(1) = ON THEN
        CONNECT(50, AXISHAND)        '链接到轴 0，倍率  50,对步进，倍率太高会出现丢步或长时间才
                                     能结束
    ELSEIF MTYPE = 21 THEN           '取消 CONNECT
        CANCEL
    ENDIF

    IF POSLAST <> DPOS THEN
        POSLAST = DPOS
        TRACE DPOS
    ENDIF
WEND
END
```

飞剪应用

参见 MOVELINK 自动凸轮指令例二。

位置比较输出

参见第十一章位置比较输出小节，分为硬件位置比较输出和软件位置比较输出。

掉电保存

参见 ONPOWEROFF 掉电中断例程。

机械手应用

参见第四章机械手“六自由度机械手”的应用例程。

编码器读取

松下 A6 编码器读取例程

```
'**********************绝对值编码器部分*****************************
SETCOM(38400,8,1,0,1,0)          '设置 485 口 PORT1 为自定义协议

GLOBAL DIM tempchar              '接收的一个字节

GLOBAL DIM neqbuff(2)            '发送识别码   485 为 81H，05H
neqbuff(0) = $81
neqbuff(1) = $05

GLOBAL DIM eotbuff(2)            '接收识别码   485 为 80H,04H
eotbuff(0) = $80
eotbuff(1) = $04

GLOBAL DIM ackbuff               '接收应答   06H
ackbuff = $06

GLOBAL DIM cmdbuff(20)           '发送命令数组
GLOBAL DIM getbuff(20)           '接收的字符串

GLOBAL DIM getnum                '接收的字节数
getnum = 0

GLOBAL DIM highdata              '编码器多圈数据
GLOBAL DIM lowdata                   '单圈数据

runtask 4,get_char               '启动接收字符串任务

MODBUS_REG(0)=0
WHILE 1
    IF MODBUS_REG(0)=1 THEN          '判断是否接收到了数据
        MODBUS_REG(0)=0
        getmpos(1,45)            '读取站号 1  的多圈与单圈值。
    ENDIF
WEND
END

'读坐标
GLOBAL SUB getmpos(sifunum,rcr)          '读伺服编号为 1 的电机的绝对值位置
    cmdbuff(0) = $00
    cmdbuff(1) = sifunum
    cmdbuff(2) = $d2
    cmdbuff(3) = rcr

    neqbuff(0) = $80 + sifunum
    neqbuff(1) = $05

    eotbuff(0) = $80
    eotbuff(1) = $04
```

```
    getnum = 0
    putchar #1,neqbuff
    TICKS = 2000                    '延时
    WAIT UNTIL (getnum = 2) OR TICKS < 0
      IF getnum = 2 THEN                        '如果接到了 2 个字符
        IF(getbuff(0)=$80+sifunum) AND (getbuff(1)=$04) THEN'收到应答发送命令
          getnum = 0
          PUTCHAR #1,cmdbuff                                    '发送读取编码器命令
          TICKS = 2000
          WAIT UNTIL (getnum = 3) OR TICKS < 0
          IF (getbuff(0) = $06) AND (getbuff(1) = $80) AND (getbuff(2) = $05)    THEN
                                                                  '收到发送请求，给应答
            getnum = 0
            PUTCHAR #1,eotbuff              '发送应答，等待接收数据
            TICKS = 2000
            WAIT UNTIL (getnum = 15) OR TICKS < 0
            IF getnum = 15 THEN         '读到数据  11-10  为多圈数据  9-7 为单圈数据
              PUTCHAR #1,ackbuff
              highdata = getbuff(11) * $100 + getbuff(10)
              lowdata = getbuff(9) * $10000 + getbuff(8) * $100 +getbuff(7)
        PRINT getbuff(11),getbuff(10),getbuff(9),getbuff(8),getbuff(7),getnum
            ELSE
              PRINT getnum,getbuff(0) ,getbuff(1),"超时重新读取 1"
            ENDIF
          ELSE
            PRINT getbuff(0) ,getbuff(1),"超时重新读取 2"
          ENDIF
      ELSE
          PRINT getbuff(0) ,getbuff(1),"驱动器无应答"
      ENDIF
    ELSE
      PRINT "驱动器无应答"
    ENDIF
END SUB

'串口接收
GLOBAL SUB get_char()
    WHILE 1
        GET #1,tempchar
        getbuff(getnum) = tempchar
        getnum = getnum + 1
    WEND
END SUB
```

自定义 G 代码

```
ERRSWITCH = 3                   '全部信息输出
BASE(0,1,2,3)                   '选择 X Y Z U，G01 里面有指定，不能随意修改
RAPIDSTOP(2)
WAIT IDLE

DPOS = 0,0,0,0
ATYPE=1,1,1,1                   '脉冲方式步进或伺服
```

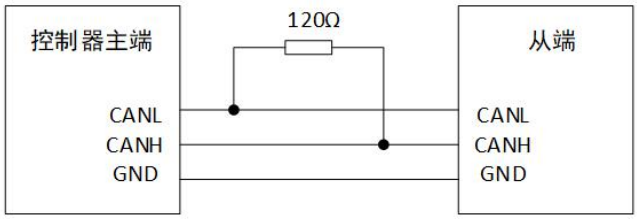
```
UNITS = 100,100,100,100        '脉冲当量，每 MM100 脉冲
SPEED = 200,200,200,200
ACCEL = 2000,2000,2000,2000
DECEL = 2000,2000,2000,2000
MERGE = ON                     '启动连续插补
CORNER_MODE = 2                '启动拐角减速
DECEL_ANGLE = 15 * (PI/180)
STOP_ANGLE = 45 * (PI/180)

G_INIT()                       'G 初始化
WHILE 1                        '循环运动
    IF IN(0) = ON THEN         '输入 0 有效启动运动
        '走一个方框
        G91 '相对位置
        G01 X100 Y0            '运动轨迹
        G01 X0 Y100
        G01 X-100 Y0
        G01 X0 Y-100
        WAIT IDLE              '等待运动停止
        DELAY(100)             '延时
    ENDIF
WEND
END                            '使得本文件自动运行时退出，避免重复执行后方的 SUB 文件

'相对位置模式
GLOBAL SUB G_INIT()
DIM coor_rel
coor_rel = 1                   '相对位置模式
END SUB

GLOBAL GSUB G01(X Y Z U)
    TRACE "G01 entered, distance:" sub_para(0),sub_para(1),sub_para(2),sub_para(3)'调试输出
    IF coor_rel THEN
        MOVE(sub_para(0),sub_para(1),sub_para(2),sub_para(3))          '相对位置
    ELSE
        LOCAL xdis, ydis, zdis, udis
        IF sub_ifpara(0) THEN                  '判断是否有参数传入 SUB
            xdis = sub_para(0)
        ELSE
            xdis = ENDMOVE_BUFFER(0)
        ENDIF
        IF sub_ifpara(1) THEN
            ydis = sub_para(1)
        ELSE
            ydis = ENDMOVE_BUFFER(1)
        ENDIF
        IF sub_ifpara(2) then
            zdis = sub_para(2)
        ELSE
            zdis = ENDMOVE_BUFFER(2)
        ENDIF
        IF sub_ifpara(3) then
            udis = sub_para(3)
        ELSE
            udis = ENDMOVE_BUFFER(3)
```

```
        ENDIF
        MOVEABS(xdis,ydis,zdis,udis)                '绝对位置
    ENDIF
END SUB

'绝对位置模式
GLOBAL GSUB G90()
    TRACE "G90 entered"
    coor_rel = 0
END SUB

'相对
GLOBAL GSUB G91()
    TRACE "G91 entered"
    coor_rel = 1
END SUB

'延时
GLOBAL GSUB G04(P)
    TRACE "G04 entered"
        IF sub_ifpara(0) THEN
            DELAY (sub_para(0))
        ELSE
        ENDIF
END SUB

GLOBAL GSUB G00(X Y Z U)
    TRACE "G00 entered, distance:" sub_para(0),sub_para(1),sub_para(2),sub_para(3)'调试输出
    IF coor_rel THEN
        MOVE(sub_para(0),sub_para(1),sub_para(2),sub_para(3))
    ELSE
        LOCAL xdis, ydis, zdis, udis
        IF sub_ifpara(0) THEN
            xdis = sub_para(0)
        ELSE
            xdis = ENDMOVE_BUFFER(0)
        ENDIF
        IF sub_ifpara(1) then
            ydis = sub_para(1)
        ELSE
            ydis = ENDMOVE_BUFFER(1)
        ENDIF
        IF sub_ifpara(2) THEN
            zdis = sub_para(2)
        ELSE
            zdis = ENDMOVE_BUFFER(2)
        ENDIF
        IF sub_ifpara(3) THEN
            udis = sub_para(3)
        ELSE
            udis = ENDMOVE_BUFFER(3)
        ENDIF
        MOVEABS(xdis,ydis,zdis,udis)
    ENDIF
END SUB
```

模块通讯

CAN 通讯

控制器间接线



CANL - CANL
CANH - CANH
在控制器主端 CANL 和 CANH 间连接一个 120 欧电阻。

主端程序

```
CANIO_ADDRESS = 32              '主端
STOPTASK 1
RUNTASK 1,task_canget           '启动接受任务
GLOBAL if_send
if_send = 1

WHILE 1
    IF if_send = 1 THEN
        TABLE(0,0,8,1,2,3,4,5,6,7,8)  '向 can cob id 为 0 的控制器发送 8 个字节依次为 1-8
        CAN(0,7,0)                    '0 - CAN 通道号、7 - 发送、0-数据 table 起始地址
        if_send = 0
    ENDIF
WEND
END

GLOBAL SUB task_canget()            '接受任务
    WHILE 1
    CAN(0,6,10)         '数据接受，存放在 table(10)之后，table(10)表示发送端 CANID，<0 时表示没有数据
                        'table(11)表示接受到到数据个数, table(12)...表示数据
        IF(table(10) >= 0 )   THEN      '有接受到数据
            ?"数据发送端 ID:",table(10)
            ?"数据字节数:",table(11)
            ?"数据："table(12),table(13),table(14),table(15),table(16),table(17),table(18)
        ENDIF                       '0 - CAN 通道号、7 - 发送、0-数据 table 起始地址
    WEND
END SUB
```

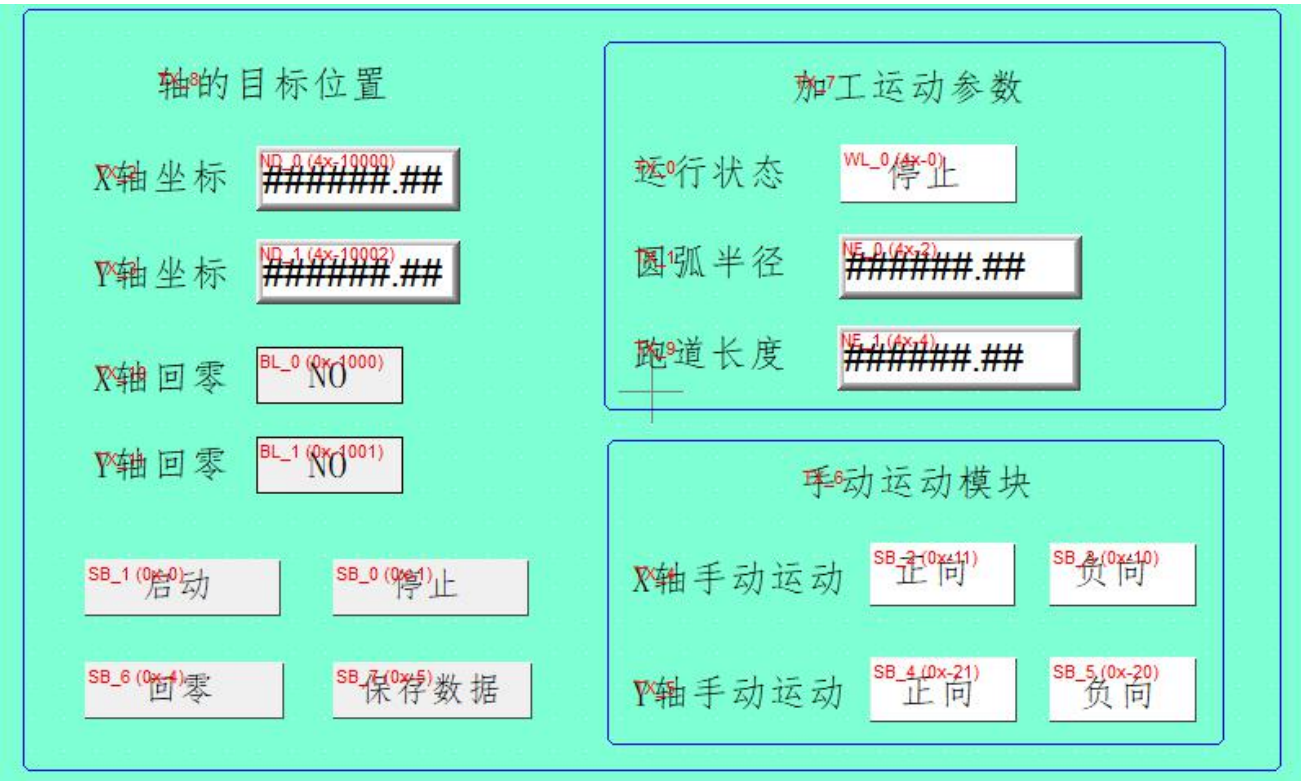
从端程序

```
CANIO_ADDRESS = 0               '从端
STOPTASK 1
RUNTASK 1,task_canget           '启动接受任务
GLOBAL if_send
if_send = 0
```

```
 WHILE 1
        WAIT UNTIL if_send = 1     '等待有数据发送过来
       TABLE(0,32,1,$ff)    '向 can cob id 为 32 的控制器器发送 1 个字节，依次为 FF
       CAN(0,7,0)                    '0 - CAN 通道号、7 -  发送、0-数据 table 起始地址
       if_send = 0
 WEND
 END

GLOBAL SUB task_canget()              '接受任务
    WHILE 1
       CAN(0,6,10)  '数据接受，存放在 table(10)之后，table(10)表示发送端 CANID，<0 时表示没有数据
                              'table(11)表示接受到到数据个数, table(12)...表示数据
       IF(table(10) >= 0 )   THEN       '有接受到数据
            ?"数据发送端 ID:",table(10)
            ?"数据字节数:",table(11)
            ?"数据："table(12),table(13),table(14),table(15),table(16),table(17),table(18)
            if_send = 1
       ENDIF
    WEND
END SUB
```

触摸屏通讯

该例程触摸屏地址从 0 开始，也有部分触摸屏地址是从 1 开始的，地址从 1 开始的触摸屏上的 1 对应程序中的 0。




```
'**********初始化模块*********
ERRSWITCH = 3                  '全部信息输出
RAPIDSTOP(2)
```

```
WAIT IDLE

BASE(0,1)                       '选定 X Y 轴
DPOS=0,0
ATYPE=1,1
UNITS = 100,100
SPEED = 100,100
ACCEL = 1000,1000
DECEL = 1000,1000
SRAMP = 100,100

DIM run_state                    '运行状态
run_state = 0                    '0 停止，1 运行，2 回零
MODBUS_REG(0) = run_state        '显示运行状态

DIM radius,length                '半径,长度
radius = 100                     '缺省半径大小
length = 300                     '缺省长度
FLASH_READ 0,radius,length
MODBUS_IEEE(2) = radius          '显示半径大小
MODBUS_IEEE(4) = length          '显示长度

DIM home_done                    '回零完成的标志位  0 未回零,1 已回零
home_done = 0                    '上电进入未回零状态

MODBUS_BIT(0) = 0                '启动 按钮复归
MODBUS_BIT(1) = 0                '停止 按钮复归
MODBUS_BIT(4) = 0                '回零 按钮复归
MODBUS_BIT(5) = 0                '保存数据 按钮复归

MODBUS_BIT(1000)=0                   'X 轴 回零标志为 0
MODBUS_BIT(1001)=0                   'Y 轴 回零标志为 0

STOPTASK 2
RUNTASK 2, guidetask          '启动手动运行任务

'**********按键扫描模块*********
WHILE 1                                 '扫描触摸屏端按钮输入
    IF MODBUS_BIT(0)= 1 THEN            '启动按钮按下
        MODBUS_BIT(0) = 0               '按钮复归

        IF run_state = 0 THEN                   '待机停止状态
            IF home_done = 0 THEN               '未回零时不启动运动
                TRACE "before move need home"
            ELSEIF home_done = 1 THEN           '已回零启动任务运行
                TRACE "move start"
                STOPTASK 1                      '软件安全，停止任务 0
                RUNTASK 1, movetask             '启动运行加工任务 1
            ENDIF
        ENDIF

    ELSEIF MODBUS_BIT(1) = 1 THEN       '停止按钮按下
        TRACE "move stop"
        MODBUS_BIT(1) = 0               '按钮复归
```

```
        RAPIDSTOP(2)
        STOPTASK 1
        RAPIDSTOP(2)

        run_state = 0                       '停止标志
        MODBUS_REG(0) = run_state           '显示状态
    ENDIF

    IF MODBUS_BIT(4) = 1 THEN               '回零按钮按下
       MODBUS_BIT(4) = 0                    '回零复归

        IF run_state= 0    THEN
            stoptask 1
            runtask 1,home_task             '启动回零任务
        ENDIF
    ENDIF

    '"保存数据处理
    IF MODBUS_BIT(5) = 1 THEN               '保存数据  按钮按下
        MODBUS_BIT(5) = 0                   '保存数据  按钮复归

        print "写入数据到 FLASH"
        radius = MODBUS_IEEE(2)
        length = MODBUS_IEEE(4)
        FLASH_WRITE 0,radius,length    '往扇区 0 写入数据
    ENDIF
WEND
END

'**********加工运动模块*********
movetask:                               '运行画圆弧+跑道的任务
    run_state =1                        '进入运行状态
    MODBUS_REG(0) = run_state

    radius = MODBUS_IEEE(2)             '读取半径
    length = MODBUS_IEEE(4)             '读取长度

    TRIGGER
    BASE(0,1)                           '选定  X Y 轴
    MOVEABS(0,0)

    MOVE(length,0)                      '从原点开始走跑道轨迹
    MOVECIRC(0,radius*2,0,radius,0)
    MOVE(-length,0)
    MOVECIRC(0,-radius*2,0,-radius,0)
    WAIT IDLE(0)

    run_state = 0                       '进入待机状态
    MODBUS_REG(0) = run_state
END

'**********回零任务*********
home_task:
    TRACE "enter home task"
```

```
    run_state = 2                   '回零标志
    MODBUS_REG(0) = run_state       '显示状态

    TRIGGER

    BASE(0,1)
    CANCEL(2) AXIS(0)               '先轴 0,轴 1 停止
    CANCEL(2) AXIS(1)
    WAIT IDLE(0)
    WAIT IDLE(1)

    'DATUM(3) AXIS(0)               '实际设备轴 0 的归零
    'DATUM(3) AXIS(1)               '实际设备轴 1 的归零
    MOVEABS(0) AXIS(0)              '虚拟设备轴 0 的归零
    MOVEABS(0) AXIS(1)              '虚拟设备轴 1 的归零

    WAIT IDLE(0)
    MODBUS_BIT(1000)=1              '设置轴 0 已归零的标志

    WAIT IDLE(1)
    MODBUS_BIT(1001)=1              '设置轴 1 已归零的标志

    home_done = 1
    TRACE "home task done"

    run_state = 0                   '回到待机状态
    MODBUS_REG(0) = run_state
END

'**********手动运动*********
guidetask:
WHILE 1
    IF run_state = 0 THEN                   '判断是否处于停止状态
        BASE(0)
        IF MODBUS_BIT(10) = 1 THEN          '左
            MODBUS_BIT(10) = 0
            VMOVE(-1)
        ELSEIF MODBUS_BIT(11) = 1 THEN      '右
            MODBUS_BIT(11) = 0
            VMOVE(1)
        ELSEIF MTYPE = 10 OR MTYPE = 11 THEN    '非 VMOVE 运动
            CANCEL(2)
        ENDIF

        BASE(1)
        IF MODBUS_BIT(20) = 1 THEN          '左
            MODBUS_BIT(20) = 0
            VMOVE(-1)
        ELSEIF MODBUS_BIT(21) = 1 THEN      '右
            MODBUS_BIT(21) = 0
            VMOVE(1)
        ELSEIF MTYPE = 10 OR MTYPE = 11 THEN    '非 VMOVE 运动
            CANCEL(2)
        ENDIF
```

```
    ENDIF
    DELAY(100)
WEND
END
```

自定义网口通讯

```
OPEN #11, "TCP_SERVER",10     '使用自定义网口通道 2，作为主端，端口号 10

GLOBAL DIM tempchar
GLOBAL CONST datamax=20
GLOBAL DIM datanum
            datanum=0
GLOBAL DIM DATA(datamax)

STOPTASK 1
RUNTASK 1,aaa
WHILE 1
    tempchar = 0                    '清除之前的字符
    GET #11,tempchar                '获取单个字符到 tempchar
    PRINT tempchar                  '打印出字符的 ASCII 码
    DATA(datanum) = tempchar        '保存到数组
    datanum = datanum + 1
    IF datanum = datamax THEN       '超过数组空间
        datanum = 0
        FOR i = 0 TO datamax-1      '清除数组空间内容
            DATA(i) = 0
       NEXT
    ENDIF
    IF   tempchar = 59 THEN         '号终止位

        PRINT #11,"ok1245"
    ENDIF
        DELAY(10)
WEND
END

SUB aaa()
    WHILE 1
    tempchar = 0                    '清除之前的字符
    GET #10,tempchar                '获取单个字符到 tempchar
    PRINT tempchar                  '打印出字符的 ASCII 码
    DATA(datanum) = tempchar        '保存到数组
    datanum = datanum + 1
    IF datanum = datamax THEN       '超过数组空间
        datanum = 0
        FOR i = 0 TO datamax-1      '清除数组空间内容
            DATA(i) = 0
       NEXT
    ENDIF
    IF   tempchar = 59 THEN         '号终止位
        PRINT #10,"aaaaaaaaaa"
    ENDIF
```

```
    WEND
END SUB
```

控制器之间通讯

最新固件支持此功能，将主端和从端程序分别下载到不同的控制器中，用网线连接控制器的网口（可通过交换机）。

```
''''''''''''''''''''主端控制器
DIM i,j,lasttick
MODBUS_REG(j)=0
MODBUSM_DES2($1 , 20, "192.168.0.11") '控制器 IP 不能相同
WHILE 1
    lasttick=TICKS
    FOR i =0 TO 9999
        MODBUS_REG(0) = i
        MODBUSM_REGSET(0,1,0)
        MODBUS_REG(0) = 99
        MODBUSM_REGGET(0,1,0)
        IF MODBUS_REG(0) <> i THEN PRINT "MODBUS_REG(0)=" MODBUS_REG(0),
"MODBUSM_STATE=" MODBUSM_STATE
    NEXT
    ?lasttick-TICKS
WEND
END
''''''''''''''''''''从端控制器
DIM j
ADDRESS=1
MODBUS_REG(j)=0
WHILE 1
    IF MODBUS_REG(0) <> 0 then
        SPEAKOUT(100)
    ENDIF
WEND
END
```

自定义串口通讯和字符串使用

```
SETCOM(38400,8,1,0,0,0)    '配置串口为 RAM 模式
DIM TEMPVAR                '定义变量
DIM VALUE
DIM CHLIST(10)             '定义数组
FOR i=0 TO 9
    GET #0, TEMPVAR        '从通道 0 读取数据
    CHLIST(i)=TEMPVAR      '读取的数据依次存储到数组
NEXT
TRACE CHLIST               '调试
VALUE = VAL(CHLIST)        '转成变量
PRINT #0, TOSTR(CHLIST)    '转成字符串输出
```

总线初始化

EtherCAT 初始化程序

使用要求：控制器带 EtherCAT 接口，伺服驱动器必须支持 EtherCAT 总线，ZDevelop 需要 2.5 以上版本。此初始化程序只进行了总线使能操作，轴的脉冲当量、轴速度、运动轨迹需要在上位机进行设置。

```
''''''''''''''''''''初始化准备
RAPIDSTOP(2)
WAIT IDLE
FOR i=0 to 10                          '取消原来的总线轴设置
    ATYPE(i)=0
NEXT
''''''''''''''''''''EtherCAT 总线初始化
SLOT_SCAN(0)                               '开始扫描
IF RETURN THEN
    ?"总线扫描成功","连接设备数："NODE_COUNT(0)
    ?
    ?"开始映射轴号"
    AXIS_ADDRESS(0)=0+1          '映射轴号
    ATYPE(0)=65                  'EtherCAT 类型
                                 '65 位置控制，66 速度控制，67 力矩控制
    DRIVE_PROFILE(0)= -1         '伺服 PDO 功能
                                 'ATYPE=66 时设置 DRIVE_PROFILE=20
                                 'ATYPE=67 时设置 DRIVE_PROFILE=30
    DISABLE_GROUP(0)                 '每轴单独分组
    ?"轴号映射完成"
    DELAY (100)

    SLOT_START(0)                 '总线开启
    IF RETURN THEN
        ?"总线开启成功"
        ?"开始清除驱动器错误(根据驱动器数据字典设置)"
        DRIVE_CONTROLWORD(0)=0           '配合伺服清除错误
        DELAY (10)
        DRIVE_CONTROLWORD(0)=128         'bit7=1 强制伺服清除错误
        DELAY (10)
        DRIVE_CONTROLWORD(0)=0           '配合伺服清除错误
        DELAY (10)
        DATUM(0)                         '清除控制器所有轴错误
        DELAY (100)
        ?"轴使能准备"
        AXIS_ENABLE(0)=1                  '轴 0 使能
        WDOG=1                            '使能总开关
        ?"轴使能完成"
    ELSE
        ?"总线开启失败"
```

```
    ENDIF
ELSE
    ?"总线扫描失败"
ENDIF
END
```

Rtex 初始化程序

使用要求：控制器带 RTEX 接口，采用松下 RTEX 伺服驱动器。

```
RAPIDSTOP(2)
WAIT IDLE
FOR i=0 TO 10                              '取消原来的总线轴设置
    ATYPE(i)=0
NEXT

SLOT_SCAN(0)                               '开始扫描
IF RETURN THEN
    ?"总线扫描成功","连接设备数："NODE_COUNT(0)
    ?"开始映射轴号"
    AXIS_ADDRESS(0)=0+1          '映射轴号
    ATYPE(0)=50                  'Rtex 类型，50 位置控制，51 速度控制，52 力矩控制
    DRIVE_PROFILE(0)=1           '伺服带 IO 映射
    DISABLE_GROUP(0)                   '每轴单独分组
    ?"轴号映射完成"
    DELAY (100)

    SLOT_START(0)                '总线开启
    IF RETURN THEN
        ?"总线开启成功"
        DRIVE_CLEAR(0)                     '清除 RTEX 总线远程轴错误
        DATUM(0)                           '清除控制器错误
        DELAY (100)
        ?"轴使能准备"
        AXIS_ENABLE(0)=1                   '轴 0 使能
        WDOG=1                                   '使能总开关
        ?"轴使能完成"
    ELSE
        ?"总线开启失败"
    ENDIF
ELSE
    ?"总线扫描失败"
ENDIF
END
```

第十八章 错误与调试

控制器在使用过程中，因为接线不对、程序逻辑问题、指令使用错误等各种原因，会出现电机没有按照设定轨迹运行、控制器报错等各种问题。

如何来排查原因和解决问题，第一原则是关闭其他软件，使用 ZDevelop 软件排查，需要熟练使用以下功能：手动运动调试、断点调试、示波器抓取、寄存器查看、在线命令发送、程序打印信息、IO 口快速检测。

18.1 常见问题列表

现象	排查方法
电机不动	手动运动调试
触摸屏寄存器数据值不对	寄存器查看
没有按照程序预期运行	断点调试+打印程序信息
输入输出口不起作用	IO 口快速检测
机台抖动大	示波器抓取

18.1.1 问题排查

程序报错先排查程序问题：

⚠当程序运动出错后，ZDevelop 软件会显示出错信息，如果出错信息没有看到，可以通过命令行输入?*task 再次查看出错信息，双击出错信息可以自动切换到程序出错位置。下表中未找到对应代码的，请查看“错误码列表”章节。

问题	可能原因
2043:Unknown function is met	不认识的标识符，控制器不支持的功能
stop of error:2049: Line not ended.	①部分命令必须占用一整行 ②GSUB 调用不需要括号()
stop of error:2033: Unknown label is met.	①未定义的变量或数组 ②未定义的 SUB 过程 ③有定义数组，但是定义的语句没有执行到，可能是对应文件没有设置自动运行
2048:Function can only be used in expression	函数必须返回值，不用在一行的开头地方
2064:Param few	函数参数过少
2063:Param too many	函数参数过多
2072:Need = sign	未写“=”号
2060:Syntax format error	指令语法错误
error:1010	重复暂停
error:1011	没有运动，无法暂停

程序运行错误解决后，仍然不能正常运行，若电机不动，再检查以下设置。

1．驱动器原因：

驱动器的出厂设置一般没有反转 IO 电平，会导致驱动器限位报警，要根据驱动器手册设置限位电平反转。比如松下伺服要将 Pr4.01、Pr4.02 的参数分别设置为 010101h（65793）、020202h（131586）。其他品牌驱动器请根据相关驱动器手册操作。

对应参数	出厂设置值（10 进制）	位置控制/全闭环控制	
		信号名称	逻辑
Pr4.00	00323232h（3289650）	SI-MON5	常开（ON）
Pr4.01	00818181h（8487297）	POT	常闭（NC）
Pr4.02	00828282h（8553090）	NOT	常闭（NC）

信号名称	记号	设定值	
		常开（ON）	常闭（NC）
无效	-	00h	设定不可
正方向驱动禁止输入	POT	01h	81h
负方向驱动禁止输入	NOT	02h	82h

2．程序可能原因：

1）UNITS 设置过小，电机运动极为缓慢。肉眼无法分辨。

2）电机处于异常状态（限位、报警…），无法运动，打印 AXISSTATUS 的值判断。

3）电机接线错误，发出脉冲无法正确传递。

4）轴使能 OP 口未打开（伺服电机才需要打开）。

5）程序处理使得电机无法运动，下载空程序确认。

6）驱动器报警。

以下原因为总线轴才会出现：

7）总线扫描、开启失败，打印返回值确认。

8）未打开 WDOG 总使能、AXIS_ENABLE 轴使能指令。

9）驱动器状态设置错误，具体查看驱动器手册。

3．问题排查步骤：

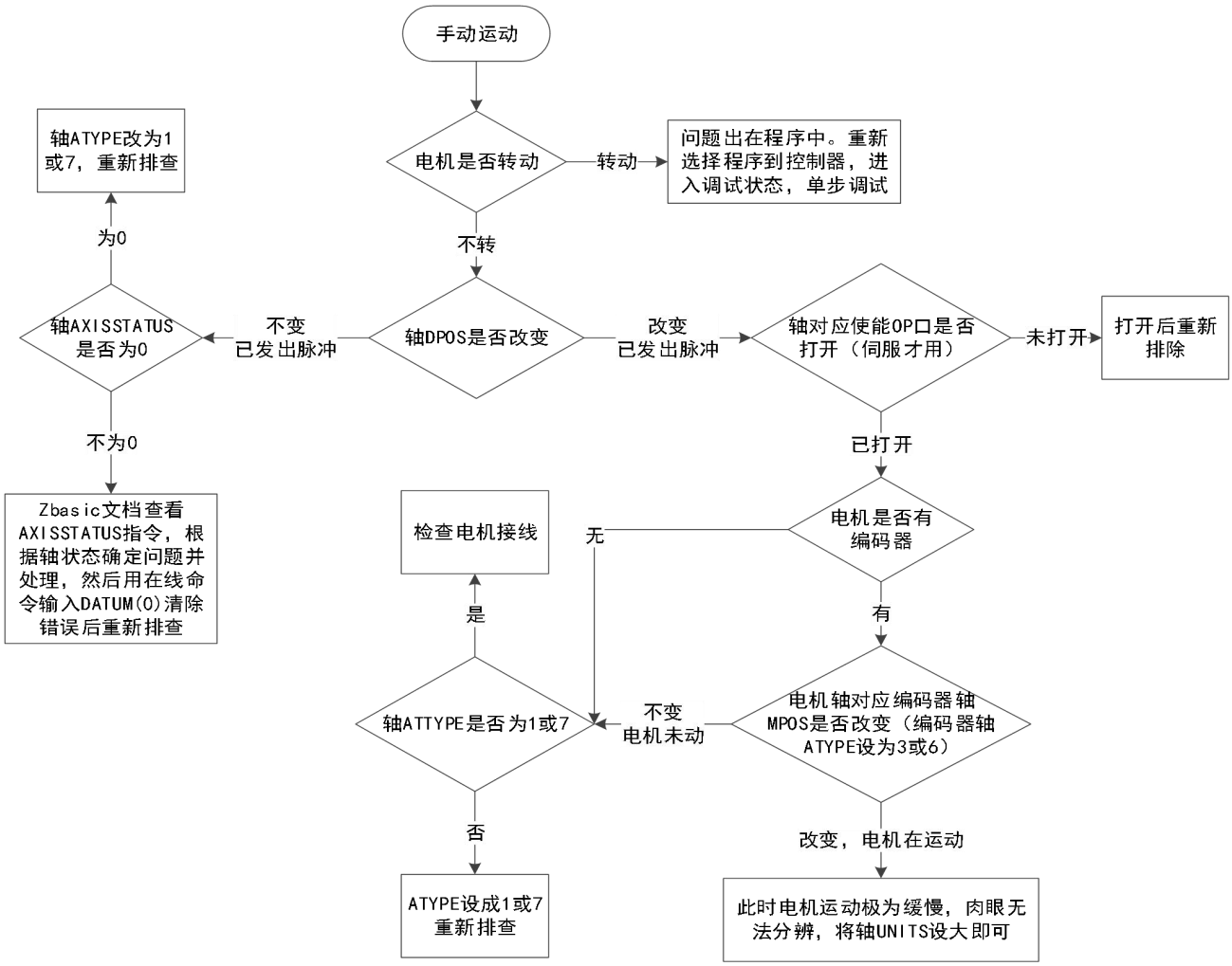
1）使用 ZDevelop 软件进行问题排查。

2）关闭除 ZDevelop 的其他连接控制器的软件、程序等，避免外部因素影响。

3）使用 ZDevelop 下载一个空程序到控制器，避免内部因素影响。

4）打开 ZDevelop 的“视图”-“手动运动”和“视图”-“轴参数”，便于操作和查看。

5）脉冲轴按照下方步骤进行排查。



电机只能单向运动，可能原因：

1. 电机处于限位状态，查看 AXISSTATUS 确认。
2. 电机控制模式不对，INVERT_STEP 设置为相应模式（双脉冲或脉冲+方向）。
3. 电机接线问题，确认接线。

18.2 解决办法

18.2.1 手动运动调试

手动运动可以排查电机接线问题。

关闭所有除 ZDevelop 的软件，同时使用 ZDevelop 连接控制器，下载空程序，并在“视图”-“轴参数”中选择轴号，手动设置好轴类型 ATYPE、脉冲当量 UNITS、、加速度 ACCEL、减速度 DECEL、速度 SPEED，然后打开“视图”-“手动运动”，手动操作电机。（下图为 ZDevelop V3.00.01 版本）

操作方法：按住“左”/“右”不放，电机持续运动，松开停止。“指令位置”显示当前发出的脉冲 DPOS（单位为 UNITS）。填写“距离”参数，点击“运动”，勾选“绝对”时，电机运动到距离参数位置；不勾选“绝对”时，电机按距离参数继续运动。

“左”“右”操作时可能会出现的问题及解决办法：

1．电机不动，但 DPOS 改变。

此时控制器脉冲已发出，检查驱动器有无报警、电机接线。

也可能是 UNITS 设置过小，电机在转动，但是转动不明显。

2．电机只往一个方向转动。

查看电机控制方式，目前控制器脉冲轴只能用双脉冲和脉冲+方向两轴控制模式，不可使用正交脉冲控制。

3．只操作某一边时电机才转动。

检查电机接线。

电机当前控制模式与控制器当前控制模式不同，控制器默认为脉冲+方向控制，使用 INVERT_STEP 指令可以修改。

4．电机不动，DPOS 也不改变。

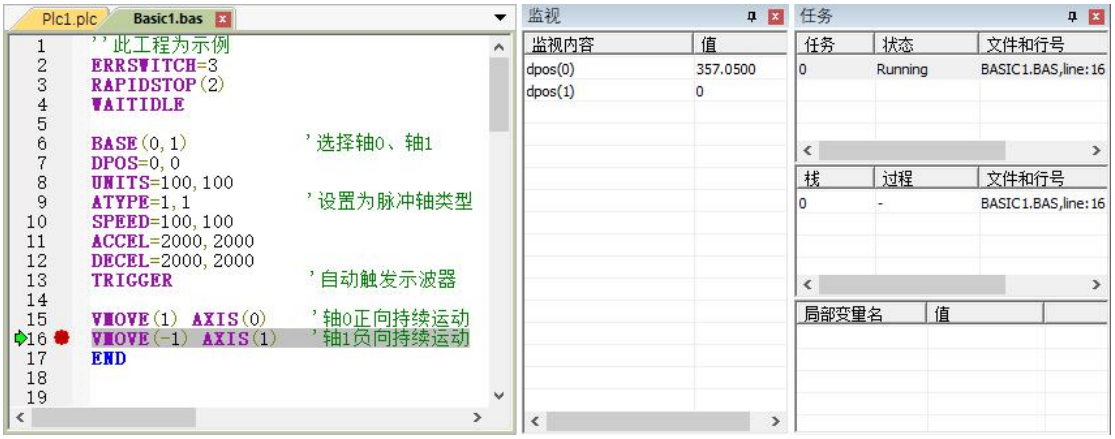
检查轴参数 AXISSTATUS 是否报警。

18.2.2 断点调试

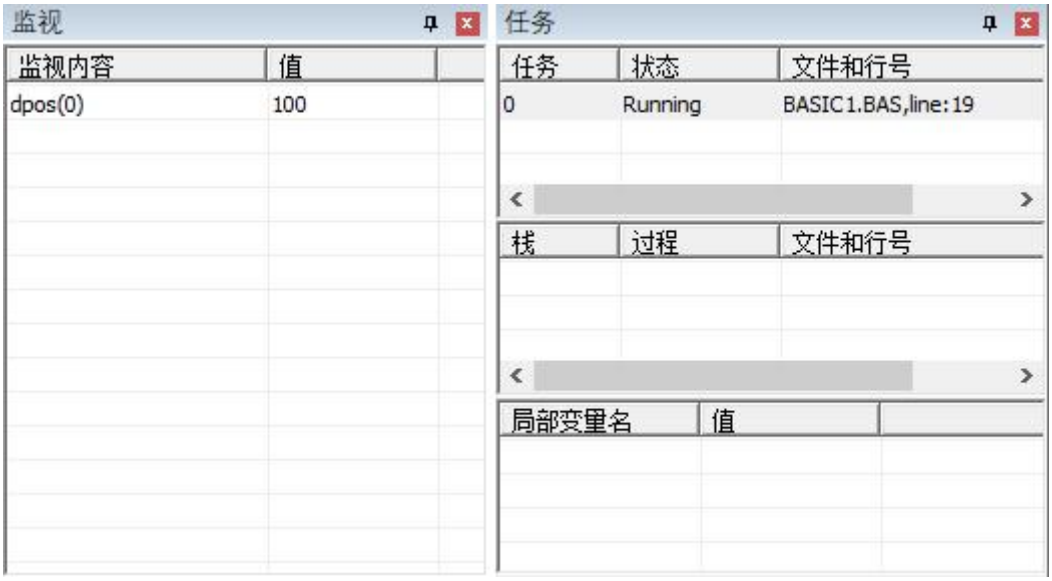
断点调试可以查看程序运行的具体过程，主要用于判断程序逻辑错误。配合监视内容可以查看程序每执行一步对寄存器、变量、数组等的影响。调试程序和控制器程序必须一致。

断点快捷键 F9 添加，可以添加多个，调试完毕后将所有断点移除后再次下载程序到控制器运行。

对于使用 ZDevelop 软件开发的程序，连接好控制器，点击“调试”-“启动/停止调试”进入调试状态。详情参见“程序调试”。



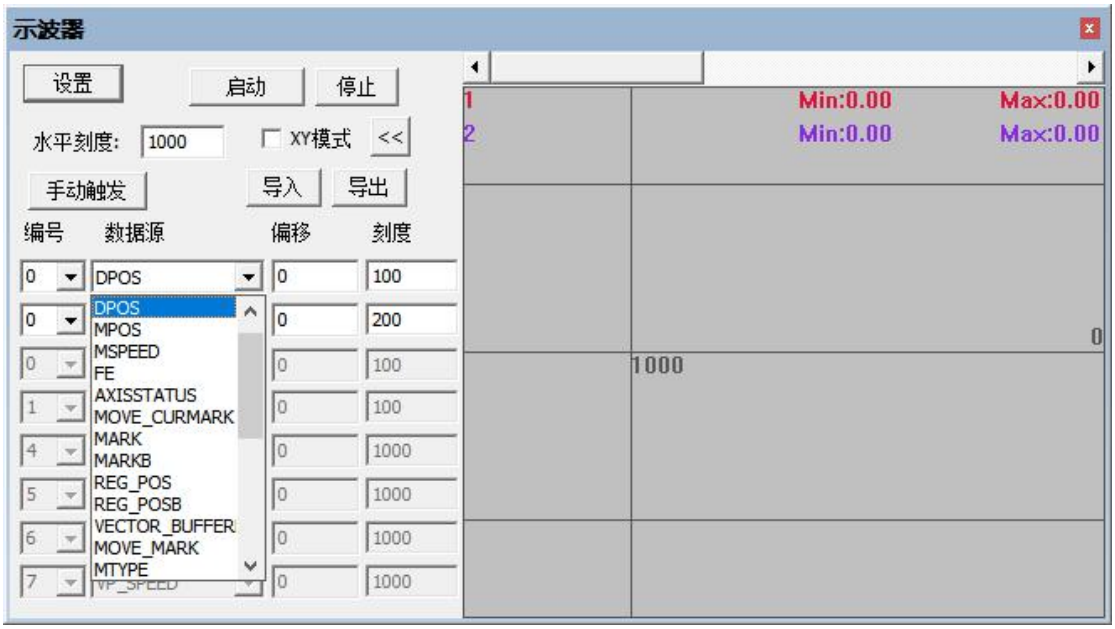
此时可以查看个任务运行情况、监视内容、子函数调用过程、子函数局部变量值。



18.2.3 示波器抓取

示波器可以抓取各类数据，数据具体种类查看“视图”-“示波器”-“数据源”。

示波器抓取一般用来判断电机实际运行速度和实际位置。

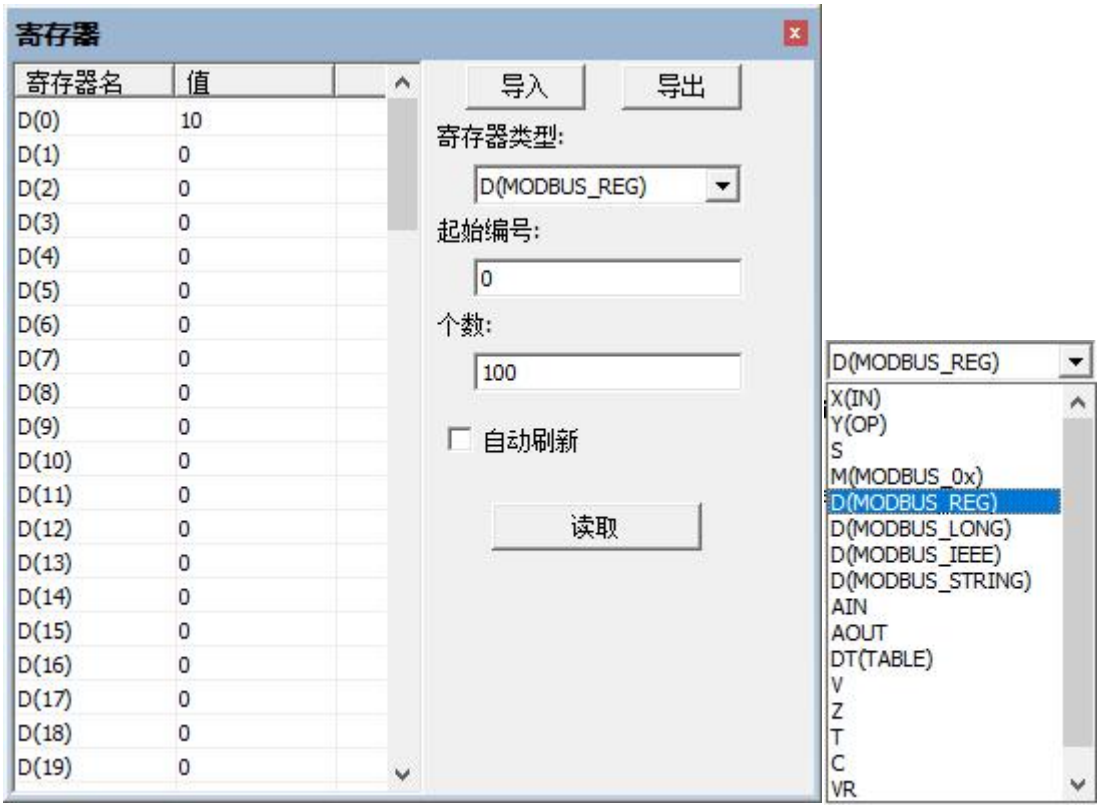


XY 模式可以查看二维的路径轨迹，需要将前两个通道的“数据源”设置为 DPOS 或 MPOS。具体使用方法查看“示波器的使用”章节。

机台实际运行抖动大时，可以用示波器抓取编码器 MSPEED，查看波形是否光滑；如果光滑说明脉冲发送平稳，此时查看速度曲线加减速过程是否过陡，匀速时速度值是否过大。

18.2.4 寄存器查看

使用 Zdevelop 软件可以方便的查看寄存器的数据，点击“视图”-“寄存器”查看。寄存器种类有 IN、OP、MODBUS_4X、MODBUS_0X、TABLE、VR、AIN、AOUT。



必须支持 PLC 功能的控制器才能使用。

比如出现触摸屏某个寄存器参数没有按照预期变化时，此时可直接查看触摸屏对应的寄存器类型及编号，确认控制器中此寄存器数值有无变化。

如果变化，说明控制器与触摸屏通讯出现问题，检查接线和通讯参数设置；如果无变化，说明程序执行有问题，断点调试检查程序执行是否正确。

18.2.5 在线发送命令

在线命令可以实时执行发送的指令，一般用于直接判断控制器指令功能是否正常。

比如程序已经执行，但实际电机没有按照程序预设功能运行，无法确定是否是程序中其他任务或流程影响，还是控制器功能错误。此时就可以下载空程序到控制器，然后直接在线命令发送功能指令，观察现象即可判断问题原因。

18.2.6 打印程序信息

在各段程序间打印信息，可以方便的查看程序段是否执行，执行了几次，执行时对应的参数是什么。

打印指令为 PRINT，也可以使用简写“?”（注意是英文符号）。

```
1
2   dim ccv(20),aa,bb
3   ccv="ad"
4   aa=100
5   bb=300
6
7   base(0,1,2)
8   atype=1,1,1
9   units=200,200,200
10  speed=100,100,100
11  accel=1000,1000,1000
12
13  stoptask 1
14  runtask 1,aaa
```

```
16  while 1
17      ?bb
18      move(bb)
19  wend
20  end
21
22
23  sub aaa(num)
24    while 1
25      ?aa
26      move(aa)
27    wend
28  end sub
29
```

18.2.7 IO 口快速检测

ZDevelop 连接控制器，点击“视图”-“输入口”、“输出口”，同时，将输出口和输入口一对一连接起来（将输入口和输出口的 EGND 接在一起），然后操作“输出口”，可以查看到“输入口”中对应状态变化。

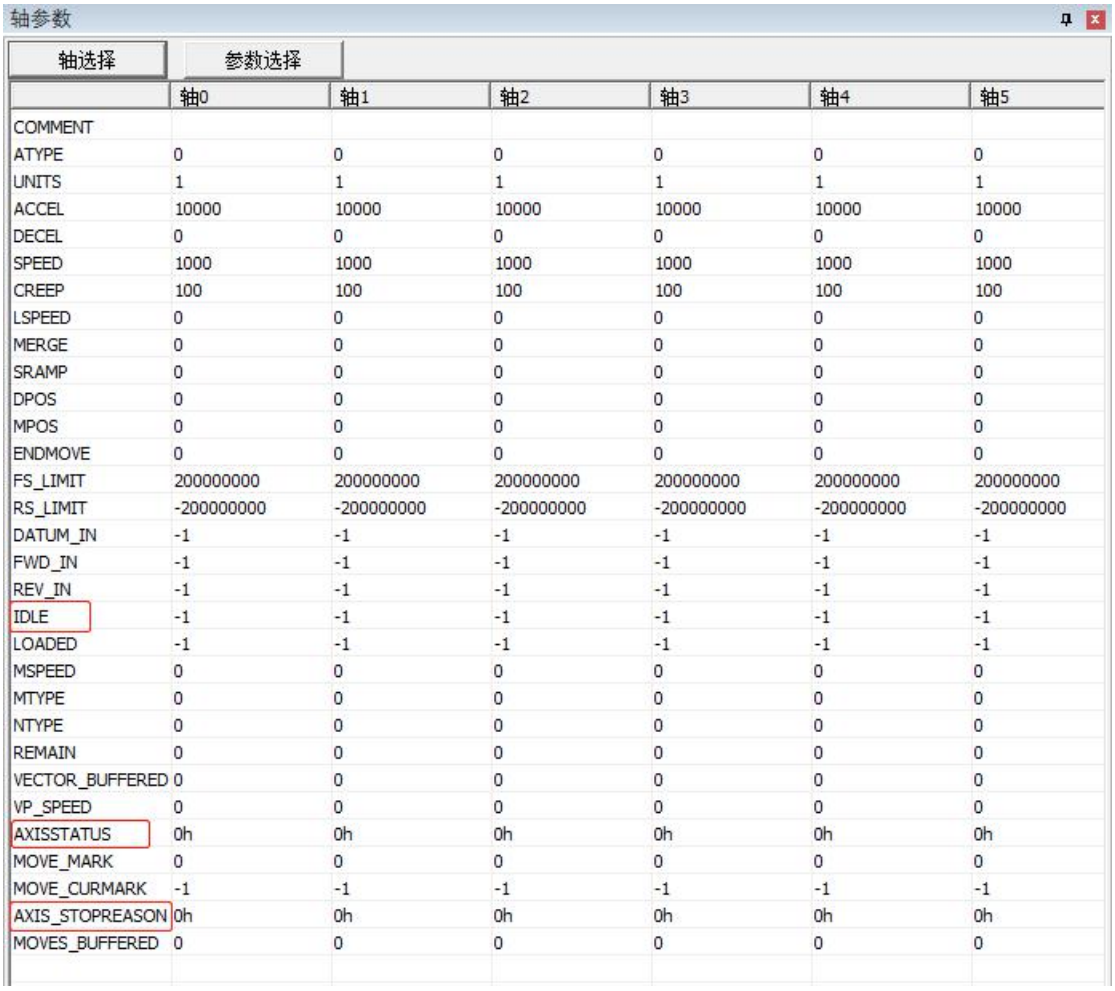
某些控制器 IO 口需要另接 24V 电源给 IO 单独口供电。

如果检测扩展板 IO，先确认 CANL 和 CANH 接线时是否接入 120 欧电阻；然后确认拨码开关设置是否正确（不能覆盖控制器原本 IO 编号）；再确认主电源与 IO 电源是否接好；最后按照上面所述进行检测。

18.2.8 轴参数状态判断

轴的基本参数在轴参数窗口里随时查看，例如 ATYPE、UNITS、SPEED 等重要参数的设置情况。在此窗口可直接修改轴参数，修改完成立即生效，只读的参数不支持修改。

轴运行状态判断主要依据 IDLE、AXISSTATUS 和 AXIS_STOPREASON 这三个指令，在轴参数窗口对这三个指令参数实时监控。



	轴0	轴1	轴2	轴3	轴4	轴5
COMMENT						
ATYPE	0	0	0	0	0	0
UNITS	1	1	1	1	1	1
ACCEL	10000	10000	10000	10000	10000	10000
DECEL	0	0	0	0	0	0
SPEED	1000	1000	1000	1000	1000	1000
CREEP	100	100	100	100	100	100
LSPEED	0	0	0	0	0	0
MERGE	0	0	0	0	0	0
SRAMP	0	0	0	0	0	0
DPOS	0	0	0	0	0	0
MPOS	0	0	0	0	0	0
ENDMOVE	0	0	0	0	0	0
FS_LIMIT	200000000	200000000	200000000	200000000	200000000	200000000
RS_LIMIT	-200000000	-200000000	-200000000	-200000000	-200000000	-200000000
DATUM_IN	-1	-1	-1	-1	-1	-1
FWD_IN	-1	-1	-1	-1	-1	-1
REV_IN	-1	-1	-1	-1	-1	-1
IDLE	-1	-1	-1	-1	-1	-1
LOADED	-1	-1	-1	-1	-1	-1
MSPEED	0	0	0	0	0	0
MTYPE	0	0	0	0	0	0
NTYPE	0	0	0	0	0	0
REMAIN	0	0	0	0	0	0
VECTOR_BUFFERED	0	0	0	0	0	0
VP_SPEED	0	0	0	0	0	0
AXISSTATUS	0h	0h	0h	0h	0h	0h
MOVE_MARK	0	0	0	0	0	0
MOVE_CURMARK	-1	-1	-1	-1	-1	-1
AXIS_STOPREASON	0h	0h	0h	0h	0h	0h
MOVES_BUFFERED	0	0	0	0	0	0

1、IDLE 指令用于判断加在轴上的运动指令是否完成，运动中返回 0，运动结束返回-1，程序中一般使用 WAIT IDLE(轴号)语句判断轴状态。

2、AXISSTATUS 查看轴的各种状态。按十进制显示数值，按二进制对应位判断状态，可同时发生多个错误。

轴参数窗口显示的是八进制，使用 PRINT 指令打印的值为十进制。

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD 速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL 执行中	2048	800h
12	脉冲频率超过 MAX_SPEED 限制，需要修改降速或修改	4096	1000h

	MAX_SPEED		
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

3、AXIS_STOPREASON 轴历史停止原因锁存，写 0 清除，按位锁存，锁存的是 AXISSTATUS 的信息。

18.3 常见问题解答

1．调用运动指令，但是轴没有动？

有限位或告警输入，或限位告警的电平配置错误。

特殊信号缺省是常闭的，国内部分产品是常开的。需要设置 INVERT_IN 把对应的口翻转过来。

轴的 ATYPE 配置错误。

2．提示不认识的 LABEL？

变量没有在使用前声明。

LABEL 有字符敲错。

3．使用“MPOS=”赋值修改坐标后坐标有误差？

DPOS 是虚拟位置，MPOS 是反馈位置，对虚拟轴，MPOS=DPOS，修改 MPOS 时确认运动已经完全停止，否则会有不确定的因素，为保险起见，可以使用 DPOS=XX 赋值来修改坐标，此时只用虚拟轴停止即可。

4．RUN 启动多任务出错？

RUNSTOP 后面接 BAS 文件名。

RUNTASK 后面接 SUB 过程名。

RUNTASK 之前，可以先查询一下这个任务号是否已存在运行。

5．TICKS 计算出来的值有问题？

TICKS 是减计数的。

6．连续插补如何实现？

设置参数 MERGE=ON，如果需要拐角减速和小圆限速，则要使用更多的参数。

```
MERGE=ON'启动连续插补
CORNER_MODE=2'启动拐角减速
DECEL_ANGLE=15*(PI/180)'开始减速的角度 15 度
STOP_ANGLE=45*(PI/180)'降到最低速度的角度 45 度
FULL_SP_RADIUS=100'小圆
```

使用 MOVE_MARK 和 MOVE_CURMARK 来判断连续插补当前运动到哪一段；规划每条线段的限速可采用 MOVESP 等指令，同时设置好 FORCE_SPEED,ENDMOVE_SPEED 等参数。

7．点位运动如何实现？

没有专门的点位运动指令，直接使用 MOVE 指令来实现，一次只操作一个轴。

```
MOVE AXIS(0)
MOVE AXIS(1)
```

8．VMOVE 是否可以操作多个轴？

不可以，一条指令只能操作一个轴，多轴必须分多条命令。

```
VMOVE(1) AXIS(0)
VMOVE(1) AXIS(1)
```

9．插补运动的参数如何设置？

BASE 的第一个轴为主轴，主轴的参数就是插补运动的参数。

```
BASE(1,2)
MOVE(1000,122)   '轴 1 的参数为插补参数
MOVE(1000,122)   AXIS(0)   轴 0 的参数为插补参数
```

10．中断函数为什么没有用？

中断开关缺省是关闭的，可以查看中断开关系统参数是否打开。

11．能否使用输入口跳变作为中断函数？

不可以，输入口的个数理论上可上万，无法实现这样多的中断函数。

12．如何从 PC 访问控制器定义的变量？

最好把要访问的变量，数组等定义为全局变量。

通过 EXECUTE 函数来修改或读取。

13．是否所有的指令都可以从 PC 直接发送？

WAIT 等部分指令限制从 PC 发送，因为会造成 PC 命令通道阻赛。

也不能通过 PC 命令来定义变量数组。

14．如何立刻停止运动？

不存在绝对的立刻停止，通过设置 FASTDEC 参数为很大值，可以使得限位或 RAPIDSTOP 时可以快速停止。

停止之后再次运动，最好在 RAPIDSTOP 之后加上 WAIT IDLE 等待所有的轴停止。

15．手轮如何实现？

采用 CONNECT 指令来实现，参见手轮例程。

16．FORWARD 和 REVERSE 能不能立刻修改运动方向？

不能，FORWARD 和 REVERSE 每次调用都会加入缓冲，因此最好 CANCEL 之后再调用新的运动指令。

另外一种方式是使用 VMOVE，VMOVE 会自动判断以前的指令，如果也是 VMOVE 则无需 CANCEL，直接修改运动方向，因此要实现类似 JOG 的功能，最好用 VMOVE 来实现。

17．如何通过 MODBUS 直接访问一些系统状态？

通过特殊的 MODBUS 寄存器可以访问 IO、位置等。位 10000 开始表示输入，20000 开始表示输出。

字寄存器每个轴占两个字，FLOAT 格式。

```
#DEFINE  MODBUS_REGNUM_DPOS_X   10000
#DEFINE  MODBUS_REGNUM_MPOS_X   11000
#DEFINE  MODBUS_REGNUM_VP_SPEED_X   12000
```

18．CANCEL 后运动还在执行？

CANCEL 缺省 0：取消当前运动，缓冲的运动不取消；带参数 2，CANCEL(2)后当前运动和缓冲运动都取消。

19．能否使得打印命令受控，这样不需要打印的时候可以不用删除代码？

使用 TRACE、WARN、ERROR 来打印，通过对 ERRSWITCH 指令赋值不同的值可以控制对应的指令是否输出。

20．是否支持不同脉冲当量的轴做圆弧插补？

支持。

21．设置了 FORCE_SPEED，但是实际运动的速度没有达到 FORCE_SPEED？

SPEED 参数也会对插补运动产生限制，而且是动态的，把 SPEED 修改即可。

22．需要掉电能保存数据，怎么办？

VR 的数据是掉电保存的，如果其他数据要保存，可以写入 FLASH，上电时再读取出来。

23．定义了变量，但是另外一个文件中访问不到？

变量分为全局，文件，局部三种类型，缺省的都是文件变量，只能在当前文件中访问，请修改为 GLOBAL

定义，则可以所有的地方访问。

24．定义了 SUB，但是另外一个文件中无法调用？

SUB 分为全局和文件两种类型，缺省的是文件类型，只能在当前文件中访问，请修改为 GLOBAL 定义，则可以全局访问。

25．程序运行中出错，如何查错？

不要重启，用 ZDevelop 工具连接上去（选择附加到当前程序的方式），即可查看程序运行内部状态和出错位置。

通过在线命令栏输入：“?*TASK”也可以参考出错原因和位置。

26．内部的 WARN 输出如何关闭？

设置系统参数 ERRSWITCH<2 即可关闭 WARN 输出。

27．PRINT 输出的排列不整齐，如何处理？

可以采用 TOSTR 格式化输出。

28．OFFPOS 修改生效的等待时间有多长？

非常短，基本上是立即生效的。

29．ERROR:2032:INVALIDCHARISMET 的原因是什么？

一般是出现了中文字符，检查引号等符号是否是中文的。

30．如果想自动支持多个型号的控制器，应该在程序中怎么处理？

通过 CONTROL 参数可以获取控制器基本型号。

通过 FLASH_SECTES FLASH_SECTSIZE 可以获取 FLASH 的规格。

31．网络连接不上？

检查 PC 的 IP 地址，需要在同一网段。

检查控制器 IP 地址，可以用串口连接上去查看。

检查 PC 防火墙设置。

32．PC 无法同时连接上 INTERNET 和控制器？

方式 1：修改控制器 IP 地址与 PC 和网关位于同一个网段。

方式 2：修改 PC 的子网掩码，使得网关和控制器 IP 地址位于同一个网段，比如改为：255.255.128.0

33．IDLE 以后 MPOS 还在变化？

IDLE 以后 DPOS 不会再变化，MPOS 并不一定稳定，可以延时一会，或是等待 MPOS 到达结束位置。

34．通过 DEFPOS 的方式来触发 CAMBOX 运动启动偶尔不成功？

不能 DEFPOS 修改后立刻改回去，修改后要延时一个周期，确保 CAMBOX 运动有检测到起始条件。如果是立即启动的不需要用位置触发方式。

35．如何把一个不自动运行的文件单独执行一次？

通过命令行输入“RUN FILENAME”,FILENAME 为需要执行的文件名，注意如果文件有修改，则需要全部重新下载。

36．限位的时候冲过去了？

提高 FASTDEC 的设置，或是加快限位信号的挡片感应范围。

37．使用运动 SP 指令时总是边界出现减速？

检查 STARTMOVE_SPEED 与 ENDMOVE_SPEED 这两个参数，这两个参数与 FORCE_SPEED 一样是会带入 SP 指令的缓冲的。

38．设置了 MERGE 但是没有连续？

检查 MERGE 是否设置在 MOVESP 指令的第一个 BASE 轴上。

39．传感器信号跳动，一个信号受其他信号的影响？

检查是否 IO 传感器的电源与控制器的 IO 电源不是同一个，此时需要将两者的底线连接起来。

40．回零时轴在原地附近时出现慢速一直反找的现象？

控制器针对伺服类电机可能冲过原点的情况有加特别的保护，此时可能是应该控制器检测到原点即停止，但是完全停止时又检测不到原点了，因此误以为轴已经冲过了原点感应范围。解决办法：LSPEED 在回零时设置为 0，让回零产生一定的减速距离。

41．WHILE NOT 1 跳不出循环？

ZBasic 的 NOT 是按位取反，NOT 1 的结果是-2，所以还是不为 FALSE，应该使用 WHILE FALSE 或者 WHILE NOT TRUE，TRUE 的值为-1，取反后刚好等于 0。

42．IO 扩展模块的 IO 口编号如何计算？

根据模块的 ID 来确定，ID 通过拨码开关的组合值来确定，类似 8421 方式的组合。

对 ZIO1608 所有拨码 OFF，ID0 – 输入 16 – 31 输出 16-23

第一个拨码 ON，其他 OFF，ID1 – 输入 32 – 47 输出 32-39

拨码 1、2 为 ON，其他 OFF，ID3 – 输入 64 – 79 输出 64-71

依次类推。

43．如何检查 IO 板是否上电？

IO 板需要两路 24V 电源，看 IO 板上相应的电源灯是否连接上。

44.如何检查 CAN 总线要求的 120 欧电阻是否接上？

查看 IO 板上的告警灯是否亮，120 欧姆电阻在最后一个 IO 板上 CANH 和 CANL 之间加上。

44．检查 IO 板拨码开关是否正确设置？

所有的 IO 板拨码不能重复。

45．在 ZDevelop 软件里面控制器状态查看是否有对应的 IO 板？

可以查看 ID 与拨码是否一致。

查看 IO 板的输入输出编号是否正确。

附录 I 错误码列表

错误码	意义	可能原因
210	文件过大	
212	状态错误	Resume 时为非暂停状态
213	文件下载上传出错，丢包	PC 函数调用返回此错误
214	下载文件的长度校验错误	
215	缓冲长度不够	发送字符串命令过长时返回此错误
217	控制器不支持或禁止的功能	
218	调用传递的参数错误	
219	下载冲突，同时启动了多个文件下载	
220	文件名错误，有特殊字符	
221	文件名错误，超过长度	
222	文件不存在	
223	密码保护限制	
224	密码保护限制 2	
260	硬件错误	
261	磁盘没有格式化	
262	RTC 错误	
263	NORFLASH 错误	
264	RAM 错误	
265	NANDFLASH 错误	
266	U 盘错误	
267	FPGA 错误	
268	以太网硬件错误	
271	备份电源错误	
272	子卡不存在	
273	文件丢失	
274	系统文件错误	
275	无主控，子卡上产生	
276	程序文件校验错误	
277	程序文件错误导致不启动	
278	ZAR 校验 APPPASS 出错	密码错误
279	ZAR 校验 ID 出错	
280	BAS 文件超过最大数量	
281	子卡 ID 冲突，或多主冲突	

282	不支持的功能	
283	参数文件丢失，可以强制修改需要 flash 保存的参数来自动生成	
284	zar 与控制器不匹配	该 zar 绑定了控制器 ID，只能下载到该控制器
285	图片文件错误	
286	字体文件错误	
288	控制器发生以上异常，导致下次开机报错	
1000	运动模块返回错误偏移	
1002	无运动缓冲	
1004	从轴运动中	
1005	不支持的运动功能	
1006	圆弧位置错误	坐标参数输入错误，无法画出圆弧
1007	椭圆 AB 参数错误	
1008	运动模块输入参数错误	
1009	运动中，无法操作	
1010	暂停等重复操作	
1011	IDLE 无法做暂停等操作	
1012	当前运动不支持暂停	
1013	找不到暂停点	
1014	ATYPE 不支持	
1015	ZCAN 的 ATYPE 冲突	
1016	轴不支持的功能	
1017	FRAME 校正数据错误	
1018	FRAME 校正数据过少	
1019	FRAME 校正数据满足条件的数据过少	
1020	FRAME 校正数据辅助参数过少	
1021	FRAME 校正数据间隔过小，小于关节轴数	
1022	FRAME 的输入坐标错误	
1023	FRAME 状态下坐标不能强制修改	
1024	FRAME 逆解异常	
1025	不是 FRAME 状态	
1026	FRAME HAND 错误	
1027	姿态在插补中不能切换	
1028	特殊关节轴与虚拟轴当量要求一样	
1030	CORNER_MODE 7 位设置了但不支持此运动	
1031	CORNER_MODE 7 位设置了但不是 FRAME 状态	
1032	AXIS_ADDRESS 错误	

1033	插补轴数过多	
1034	INTCYCLE 超时	
2000	ZBASIC 模块偏移，模块内参数错误	
2001-2020	ZBASIC 模块内部错误	
2021	手动停止	
2022	因其他任务错误导致本任务停止	
2023	试图修改只读状态参数	
2024	数组越界	
2025	变量数超过控制器规格	
2026	数组数超过控制器规格	
2027	数组空间超过控制器规格	
2028	SUB 数超过控制器规格	
2029	标识符命名错误	识别到指令书写错误或未加注释的中文
2030	标识符命名过长	
2031	没有右括号	
2032	不认识的字符	指令参数逗号为中文符号报此错误
2033	表达式中碰到不认识的名称	未定义的变量或数组
2034	SUB 不能在表达式中使用	
2043	不认识的命令标识符，当前行第一个标识名称	指令书写错误
2044	堆栈溢出	定义为 SUB 过程的变量再做其他用途
2045	数学表达式太复杂，不同控制器的规格不一样	
2046	没有找到结束引用标号	
2047	指令没有返回值，不能用于表达式计算	
2048	函数必须返回值，不用在一行的开头地方	
2049	特殊指令必须单独一行	部分命令必须占用一整行
2050	参数或数组需要索引	
2051	变量不能使用索引	
2052	数组重定义且长度不一致	相同数组定义多次
2053	数组定义长度参数错误，负数或过大	
2054	标识符已经定义为 SUB 过程，不能再做其他用途	SUB 过程标志符不可以再次定义
2055	标识符已经定义为参数，不能再做其他用途	
2056	标识符预留，不能使用	
2057	出现不能识别的字符	例如“&”字符系统无法识别
2058	SUB 调用重复出栈	
2060	语法格式错误	FOR 后面缺少 TO
2062	函数参数范围错误	包括任务号超过范围也返回这个错误自动运行任务号出错也是这个错误码
2063	函数参数过多	如 CONNECT(1,1,1)指令参数太多

2064	函数参数太少	如 CONNECT(1)指令参数缺少
2065	缺少操作数	
2066	操作符后面缺少操作数	
2067	操作符前面缺少操作数	
2068	系统不认识的操作符	
2069	缺少双目操作符	两个指令在同一行时，中间需要操作符连接
2070	CALL 必须调用 SUB	
2072	需要赋值符号	数据之间不加逗号用空格表示会出错
2073	空文件	
2074	SUB 定义的标识符名称冲突	
2075	要启动的任务已经运行中	
2076	多个参数要使用逗号隔开	
2077	括号不配对，无左括号	
2078	IF 判断的嵌套太多	
2079	循环语句嵌套太多	
2080	插补轴数太少	
2081	CONST 常量，不能修改	定义好的常量数据再次赋值报错
2082	命令不能从 PC 在线发送	
2083	SUB 定义的参数太多	
2084	SUB 带参数，不能用于 GOTO 语句	
2085	局部标识符定义太多	
2086	LOCAL 变量名与文件变量名或其它标识符名称冲突	
2087	LOCAL 不支持数组定义	
2088	GSUB 定义的参数字母重复	
2089	GSUB 定义的参数只能为单字母	
2090	不能修改只读参数	
2091	GSUB_IFPARA 函数使用场合错误	
2092	除数为零	
2093	超过缓冲	
2094	在线命令阻赛时间过长	
2095	参数重名	
2096	值没有初始化就使用了	
2097	轴号冲突	
2099	内部错误	
2100	SCANEDGE 个数过多	
2101	ZINDEX 类型不匹配	
2102	ZVOBJ 个数超过规格	
2103	ZVOBJ 定义不一致	
2104	ZINDEX 值错误	

2110	回调指令没有使能	
2120	结构定义冲突，不能多个地方同时定义	
2121	名称与系统指令冲突	
2122	结构定义不能递归	
2123	结构 item 与结构名称冲突	
2124	语法错误， 需要结构类型	
2125	结构 item 错误	
2126	需要结构元素	
2127	需要结构变量	
2128	结构个数超出规格	
2129	结构元素超出规格	
2130	结构类型没有定义	
2150	函数返回没有立即完成	
2151	函数不能立即返回, 当前表达式中不支持	
2152	动态堆栈溢出	
2901	系统错误，定义的标识符过多 包括变量，数组，过程，过程参数等等	
3201	超过缓冲	
3202	文件非正常结束	
3203	程序结构指令不配对	IF 指令之后缺少 THEN
3204	内部状态错误	
3205	不支持的功能	
3206	内部调用参数错误	
3231	资源不够	
3242	os 错误	
3243	U 盘没有插入	
3244	文件重复打开	
3245	文件过大	
3248	文件名错误	
3249	文件名过长	
3250	文件不存在	
3301	圆弧的三点在一条线上	
3302	两条直线平行，没有交点	

3401	MODBUS 主端参数错误，一般长度超过	
3402	消息响应超时	
3407	MODBUS 返回参数错误	
3408	MODBUS 返回不支持	
3421	MOBUS 从端返回不支持的功能码	
3422	MOBUS 从端返回地址空间错误	
3423	MODBUS 从端返回数据长度不对	
3424	MODBUS 从端返回长度过长	
3501	ZCAN 返回无子卡	
3502	ZCAN 返回子卡无对应轴	
4000-4500 PLC 模块的错误		
4002	参数错误	
4003	未知类型	
4004	未知函数	
4005	压栈太多 STL	
4006	压栈太多	
4007	程序太复杂， BLOCK 太多	
4008	没有压栈 BLOCK	
4009	没有压栈 STL	
4010	没有压栈	
4014	文件内容错误	
4015	RET 必须在 STL 的后面	
4016	超过范围	
4017	低于范围	
4018	L 没有定义	
4019	不支持 G 代码函数	
4020	不能 GOTO 跨 PLC 与 BASIC	
4021	PLC 主任务只有一个	
4022	语法错误	
4023	FOR NEXT 错误，不匹配	
4024	FOR NEXT 错误， 无 NEXT	
4026	FOR MC 混用	
4027	FOR STL 混用	
4030	必须 PLC 主任务中使用	
4031	必须中断中使用	
4032	参数个数少	
4033	参数个数多	
4034	要 8 的倍数	

4035	寄存器标识错误	
4036	寄存器类型错误	
4037	LV 个数超过	
4038	只读	
4500-5000 PLC 上位机端错误		
4503	内存不够	
4504	回流到母线上	
4505	回流	
4506	AND 类型不能直接接母线	
4510	悬空	
4511	最右端必须是输出类型	
5000-5500 HMI 模块的错误		
5000	LCD 号错误	
5001	Hmi 文件错误	窗口号相同
5002	LCD 号冲突	
5003	不支持对象	
5004	内存不够	
5005	控件层次错误	
5006	窗口号超过	
5007	无效窗口号	
5010	对象属性丢失	
5011	输入窗口有多个显示元件	
5012	ACTION 类型错误	
5013	事件过多	
5014	返回上个窗口失败	
5015	不能关闭基本窗口	
5016	字体中找不到对应字符	
5017	必须在 HMI 任务中使用	
5020	控件 ID 冲突	
5021	LCD 号错误	
5022	找不到可用 LCD	
5023	LCD 没有打开	
5024	LCD 无数据	
5025	程序复位	
5026	LCD 已经打开了	
5027	不是网络 LCD	
5028	不支持的压缩方式	
5029	颜色深度不支持	
5030	不支持的数据类型	

5031	设备号错误	
5032	LCD_SEL 不能使用	
5033	设置 REDRAW 不能再 DRAW 阶段	
5034	DRAW 函数只能在 DRAW 阶段	
5035	操作不能再 DRAW 阶段调用	
5036	内部 LCD 分辨率固定	
5037	LCD 分辨率超过	
5038	库文件名错误	
5039	字符过多	
5501-5599　PC 端 PLC 文件编译的错误		
5503	内存不够	
5504	回流到母线上	
5505	回流	
5506	AND 类型指令不能直接接母线	
5510	右边悬空，没有接输出指令	
5511	最右边不是输出类型指令	
5512	最右边不能连接在一起	
5513	输出类型指令必须在最右边	
5514	不支持的指令类型	
5517	寄存器没有值	
5518	DOT 值超过范围	
5519	索引寄存器超过范围	
5520	字符数过多	
5521	寄存器类型错误	
5522	寄存器值错误	
5523	寄存器个数过多	
5524	寄存器个数过少	
5525	STL 使用错误	
5526	RET 使用错误	
5527	重复 RET	
5528	END 或 LBL 的位置错误	
5529	函数不能直接接母线	
5530	出栈没有压栈	
5531	MPP 太多	
5532	寄存器类型使用错误	
5533	ANB 错误，块数不够	
5534	ORB 错误，块数不够	
5535	ANB 错误，输出操作后不能合并	
5536	ORB 错误，输出操作后不能合并	

5537	AND 直接接母线	
5538	OR 直接接母线	
5539	OR 不能在 OUT 指令的后面	
5540	STL 和 MC 不能共用	
5541	MC 不能直接接母线	
5542	_@寄存器要括号	
5543	注释错误	
5544	梯形图列数过多	
5545	输出类型不能直接接母线	
6000- ECAT 总线错误		
6000	ECAT 模块错误，SLOT 编号错误	
6001	内部错误， 功能不支持	
6005	参数错误	
6006	支持的设备类型数超过限制	
6009	操作 NODE 超过个数	
6012	资源不够	
6013	从设备反应超时	
6014	缓冲不够	
6015	应答包 WKC 错误	
6016	SDO 应答超长	
6017	SDO 应答错误	
6018	SDO 应答数据长度错误	
6019	WKC 超时	
6020	STATE 切换超时	
6021	SDO ABORT	
6023	NODE PROFILE 错误	
6024	轴 PROFILE 错误	
6025	轴数超过	
6029	PDO 包长度超过系统规格	
6031	设备个数超过	
6042	设备不支持	
6045	邮箱超时	
6047	数据类型错误	
6049	模块不支持的子模块	
6050	模块子模块数量超过	
6051	模块不认识的子模块	
6208	RTEX 驱动 ID 冲突	
6209	扫描超时	一般为网线接触问题
6210	RTEX 初始化失败	

6211	RTEX 扫描结果错误	
20000- PC 端错误		
20000	PC 端产生错误的偏移	
20002	参数错	
20003	超时	可能是 fifo 缓冲阻塞
20006	操作系统错误	一般为串口号错误
20007	串口打开失败	
20008	网络打开失败	
20009	句柄错误	
20010	发送错误	
20011	文件错误	
20012	文件长度错误	
20013	文件名过长	
20014	文件不存在	
20015	ZLB 库文件错误	
20016	文件没有编译	一般 PLC 文件需要编译
20020	固件文件不匹配	
20021	不支持的功能	
20023	xplcterm 没有正确运行，或是权限不够	
20024	PCI 卡链接时找不到卡或没有驱动	
20030	输入缓冲长度不够	
20100	应答缓冲长度不够	
30000	30000 以上是 ZAUX 辅助库产生的错误码	

附录Ⅱ MODBUS 协议

MODBUS 简介

1．基本概念说明

MODBUS 是 MODICON 公司最先倡导的一种软的通讯规约，经过大多数公司的实际应用，逐渐被认可，成为一种标准的通讯规约，只要按照这种规约进行数据通讯或传输，不同的系统之间就可以通讯。目前，在 RS232/RS485 通讯过程中，广泛采用这种协议。

MODBUS 协议包括 MODBUS_RTU 协议和 MODBUS_ASC 协议,两者区别在于数据链路不同，RTU 协议通讯传输的数据为二进制数据，ASC 协议通讯传输的数据需要转换成 ASCII 码数据，两者在帧结构上也有区别，RTU 协议以通信时间来判断数据是否传输完成，ASC 协议以起始帧符和结束帧符来判断，从通信效率来看，RTU 协议大概为 ASC 协议的 2 倍，处理简单，应用更广泛。

MODBUS 为单主多从通信方式，采用的是主问从答方式，每次通信都是由主站首先发起，从站被动应答，不允许独立的终端设备之间的数据交换，这样各终端设备不会在它们初始化时占据通讯线路，而仅限于响应到达本机的查询信号。如变频器之类的被控设备，一般内置的是从站协议，而 PLC 之类的控制设备，则需具有主站协议、从站协议。

每个 MODBUS 系统只能采用一种协议，在配置每个控制器时，在一个 MODBUS 网络上的所有设备都必须选择相同的传输模式和串口参数。

正运动控制器 MODBUS 主从通讯均采用 RTU 协议。

2．传输方式

传输方式是指一个数据帧内一系列独立的数据结构以及用于传输数据的有限规则，下面定义了与 MODBUS_RTU 协议相兼容的传输方式。

每个字节的位：1 个起始位，8 个数据位（最小的有效位先发送），无奇偶校验位，1 个停止位，错误检测(Errorchecking)：CRC（循环冗余校验）。

当数据帧到达终端设备时，它通过一个简单的“端口”进入被寻址到的设备，该设备去掉数据帧的数据头，读取数据，如果没有错误，就执行数据所请求的任务，然后，它将自己生成的数据加入到取得的“信封”中，把数据帧返回给发送者。返回的响应数据中包含了以下内容：终端从机地址(Address)、被执行了的命令(Function)、执行命令生成的被请求数据(Data)和一个校验码(Check)。发生任何错误都不会成功响应，错误返回一个错误指示帧。

RTU 模式下的通讯数据结构：

结构	描述
开始	保持无输入数据≥10ms
通讯站号	从站地址：8 位二进制数地址
功能码	功能码：8 位二进制数地址
数据（n-1）…0	数据内容：n×8 位二进制数，n≤202
CRC 校验低字节	CRC 校验由两个 8 位二进制数组成低字节
CRC 校验高字节	CRC 校验由两个 8 位二进制数组成高字节
结束	保持无输入数据≥10ms

3．地址域（Address）

地址域在帧的开始部分，由一个字节（8 位二进制码）组成，十进制为 0～255。这些位标明了用户指定的终端设备的地址，该设备将接收来自与之相连的主机数据。每个终端设备的地址必须是唯一的，仅仅

被寻址到的终端会响应包含该地址的查询。当终端发送回一个响应，响应中的从机地址数据便告诉了主机哪台终端正与之进行通信。

4．功能域（Function）

功能域代码告诉被寻址到的终端执行何种功能。下表为常用功能码举例，以及它们的名称和功能。

代码	名称	功能
01	读取线圈状态	取得一组逻辑线圈的当前状态（ON/OFF）
03	读数据寄存器	获得一个或多个寄存器的当前二进制值

5．数据域(Data)

数据域包含了终端执行特定功能所需要的数据或者终端响应查询时采集到的数据。这些数据的内容可能是数值、参考地址或者设置值。例如：功能域码告诉终端读取一个寄存器，数据域则需要指明从哪个寄存器开始及读取多少个数据，内嵌的地址和数据依照类型和从机之间的不同内容而有所不同。

6．错误校验域(Check)

该域允许主机和终端检查传输过程中的错误。有时，由于电噪声和其它干扰，一组数据在从一个设备传输到另一个设备时在线路上可能会发生一些改变，出错校验能够保证主机或者终端不去响应那些传输过程中发生了改变的数据，这就提高了系统的安全性和效率，错误校验使用了 16 位循环冗余的方法（CRC16）。

7．错误检测的方法

错误校验（CRC）域占用两个字节，包含了一个 16 位的二进制值。CRC 值由传输设备计算出来，然后附加到数据帧上，接收设备在接收数据时重新计算 CRC 值，然后与接收到的 CRC 域中的值进行比较，如果这两个值不相等，就发生了错误。

生成一个 CRC 的流程为：

1）预置一个 16 位寄存器为 0FFFFH（全 1），称之为 CRC 寄存器。

2）把数据帧中的第一个字节的 8 位与 CRC 寄存器中的低字节进行异或运算，结果存回 CRC 寄存器。

3）将 CRC 寄存器向右移一位，最高位填以 0，最低位移出并检测。

4）如果最低位为 0，重复第三步（下一次移位）；如果最低位为 1，将 CRC 寄存器与一个预设的固定值（0A001H）进行异或运算。

5）重复第三步和第四步直到 8 次移位。这样处理完了一个完整的八位数据。

6）重复第 2 步到第 5 步来处理下一个八位数据，直到所有的字节处理结束。

7）最终 CRC 寄存器的值就是 CRC 的值。

MODBUS 功能码及数据编址

1．MODBUS-RTU 帧格式

正运动控制器专门配备 2048 个 MODBUS 位寄存器，不同型号控制器位寄存器个数不完全相同。N 为寄存器个数。

各个功能码的帧格式基本相同，分为请求帧和响应帧，格式参考以下示范，N 为寄存器个数。

功能码 0x01（01）：读线圈

请求帧格式：从机地址+0x01+线圈起始地址+线圈数量+CRC 检验

序号	数据(字节)意义	字节数量	说明
1	从机地址	1 个字节	八位二进制数
2	0x01（功能码）	1 个字节	读线圈
3	线圈起始地址	2 个字节	高位在前，低位在后，见线圈编址

4	线圈数量	2 个字节	高位在前，低位在后
5	CRC 校验	2 个字节	高位在前，低位在后

响应帧格式：从机地址+0x01+字节数+线圈状态+CRC 检验

序号	数据(字节)意义	字节数量	说明
1	从机地址	1 个字节	八位二进制数
2	0x01（功能码）	1 个字节	读线圈
3	字节数	1 个字节	值：[(N+7)/8]
4	线圈状态	[(N+7)/8]个字节	每 8 个线圈合为一个字节，最后一个若不足 8 位，未定义部分填 0。前 8 个线圈在第一个字节，地址最小的线圈在最低位。依次类推
5	CRC 校验	2 个字节	高位在前，低位在后

2．错误响应帧

错误响应：从机地址+（功能码+0x80）+错误码+CRC 校验。

序号	数据(字节)意义	字节数量	说明
1	从机地址	1 个字节	八位二进制数
2	功能码+0x80	1 个字节	错误功能码
3	错误码	1 个字节	1~4
4	CRC 校验	2 个字节	高位在前，低位在后

3．Modbus 支持的全部功能码：

功能码	名称	作用
01	读取线圈状态	取得一组逻辑线圈的当前状态（ON/OFF)
02	读取输入状态	取得一组开关输入的当前状态（ON/OFF)
03	读取保持寄存器	在一个或多个保持寄存器中取得当前的二进制值
04	读取输入寄存器	在一个或多个输入寄存器中取得当前的二进制值
05	强置单线圈	强置一个逻辑线圈的通断状态
06	预置单寄存器	把具体二进值装入一个保持寄存器
07	读取异常状态	取得 8 个内部线圈的通断状态，这 8 个线圈的地址由控制器决定
08	回送诊断校验	把诊断校验报文送从机，以对通信处理进行评鉴
09	编程（只用于 484）	使主机模拟编程器作用，修改 PC 从机逻辑
10	控询（只用于 484）	可使主机与一台正在执行长程序任务从机通信，探询该从机是否已完成其操作任务，仅在含有功能码 9 的报文发送后，本功能码才发送
11	读取事件计数	可使主机发出单询问，并随即判定操作是否成功，尤其是该命令或其他应答产生通信错误时
12	读取通信事件记录	可使主机检索每台从机的 Modbus 事务处理通信事件记录。如果某项事务处理完成，记录会给出有关错误
13	编程(184/384484584)	可使主机模拟编程器功能修改 PC 从机逻辑
14	探询(184/384484584)	可使主机与正在执行任务的从机通信，定期控询该从机是否已完成其程序操作，仅在含有功能 13 的报文发送后，本功能码才得发送
15	强置多线圈	强置一串连续逻辑线圈的通断
16	预置多寄存器	把具体的二进制值装入一串连续的保持寄存器
17	报告从机标识	可使主机判断编址从机的类型及该从机运行指示灯的状态
18	(884 和 MICRO84)	可使主机模拟编程功能，修改 PC 状态逻辑
19	重置通信链路	发生非可修改错误后，是从机复位于已知状态，可重置顺序字节
20	读取通用参数(584L)	显示扩展存储器文件中的数据信息

21	写入通用参数(584L)	把通用参数写入扩展存储文件或修改
22～64	保留作扩展功能备用	-
65～72	保留作用户功能备用	留作用户功能的扩展编码
73～119	非法功能	-
120～127	保留	留作内部作用
128～255	保留	用于异常应答

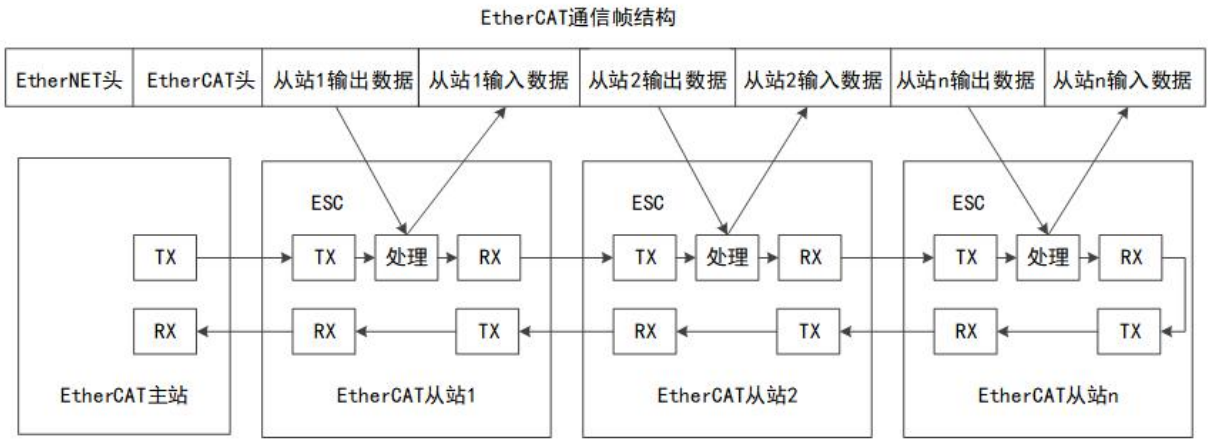
附录Ⅲ ETHERCAT 通讯

EtherCAT 总线优势

EtherCAT 总线是基于以太网的开发架构的实时工业现场总线通讯协议，目前是最快的工业以太网技术之一，提供了纳秒级精确同步，具有高性能、拓扑结构灵活，低成本、高精度、应用简单等优点。

常以多芯以太网电缆连接，EtherCAT 网络通讯速率为 100Mbps，相邻两个节点之间的距离的不超过 50 米。

EtherCAT 充分利用了以太网的全双工特性，使用主从模式介质访问控制。EtherCAT 网络和普通以太网有明显不同，同一个 EtherCAT 网络内，只有一个 EtherCAT 主站，另外 EtherCAT 从站有专门处理 EtherCAT 通讯数据的芯片 ESC(EtherCAT Slave Controller)。ESC 芯片可以在 EtherCAT 数据帧通过时，取出主站发送给该从站的数据，并将该从站需要传送给主站的数据插入到 EtherCAT 数据帧中，网络中的最后一个 EtherCAT 从站 ESC 自动闭环，将处理过的报文依次返回给主站，数据传输示意图如下图所示。



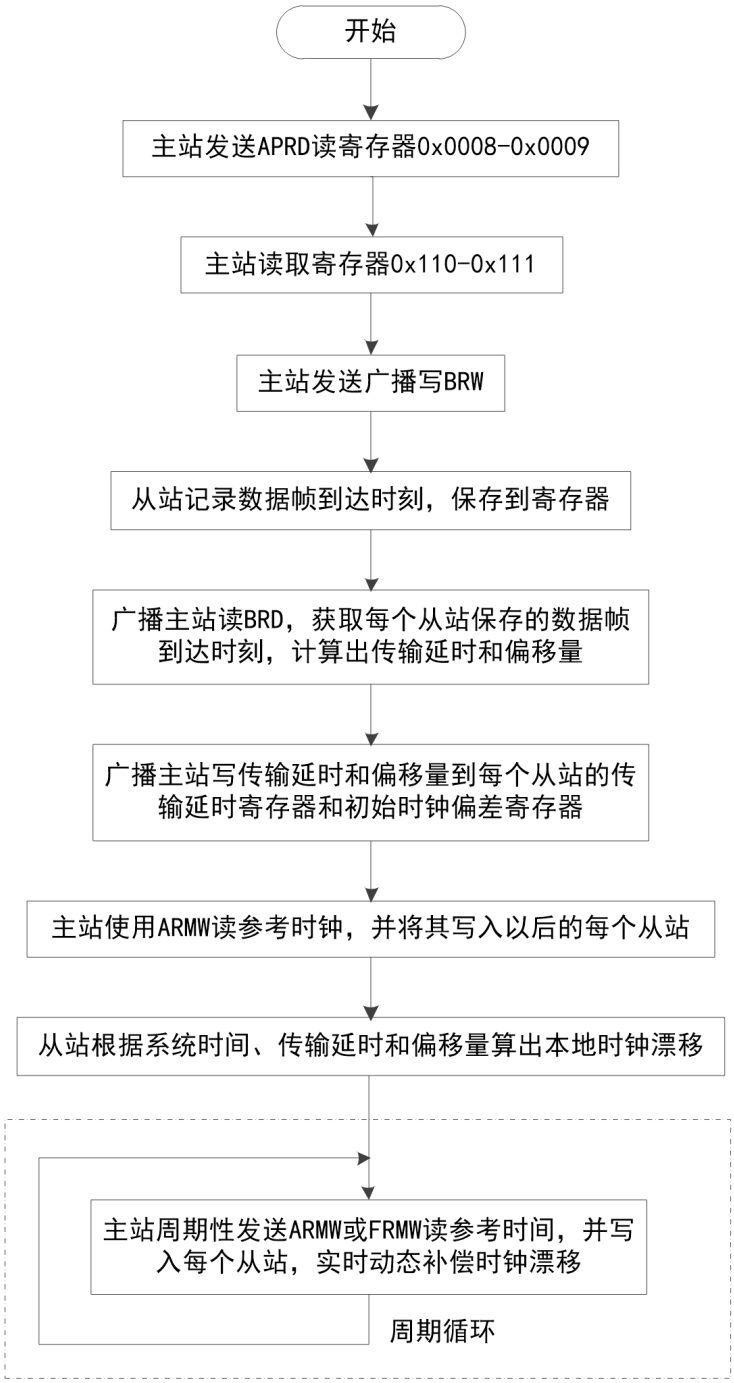
同一 EtherCAT 数据帧可会嵌入多种类型的子报文，达到较高的数据帧利用率，每个子报文有其独特的命令字和寻址方式。EtherCAT 实行动态数据处理机制，从从站直接读取数据帧中的数据快或者改变数据帧中的一部分数据，这样引入的总延时小于 100 纳秒。

EtherCAT 时钟同步概念

在运动控制系统的多轴运动网络中，往往需要让多个从站同时开始或停止运动，EtherCAT 具有很好的同步性能，利用“分布时钟（DC）”机制可实现各从站节点之间小于 1 微秒的时钟同步精度，这在要求分布系统同时工作的场合显示十分重要。

系统启动时，各从站的本地时钟和参考时钟之间有一定的差异，称为时钟偏移量，主站连接的第一个具有分布时钟功能的从站作为参考时钟，以参考时钟来同步其他设备和主站时钟。由于各从站使用的晶振等原因，

从站的计时周期会有微小偏差，称为时钟漂移，数据帧在各个从站之间传播时的延时称为传输延时，包括物理层和链路层的延时，这些因素都将进行补偿，使的补偿后的本地系统时间和网络系统时间达到时钟同步，从而使得整个 EtherCAT 网络系统同步。



术语解释

APRD：读数据，按照从站在网段内的的位置来选取从站的存储空间

BRW：广播写，写入所有联网从站的物理存储区域

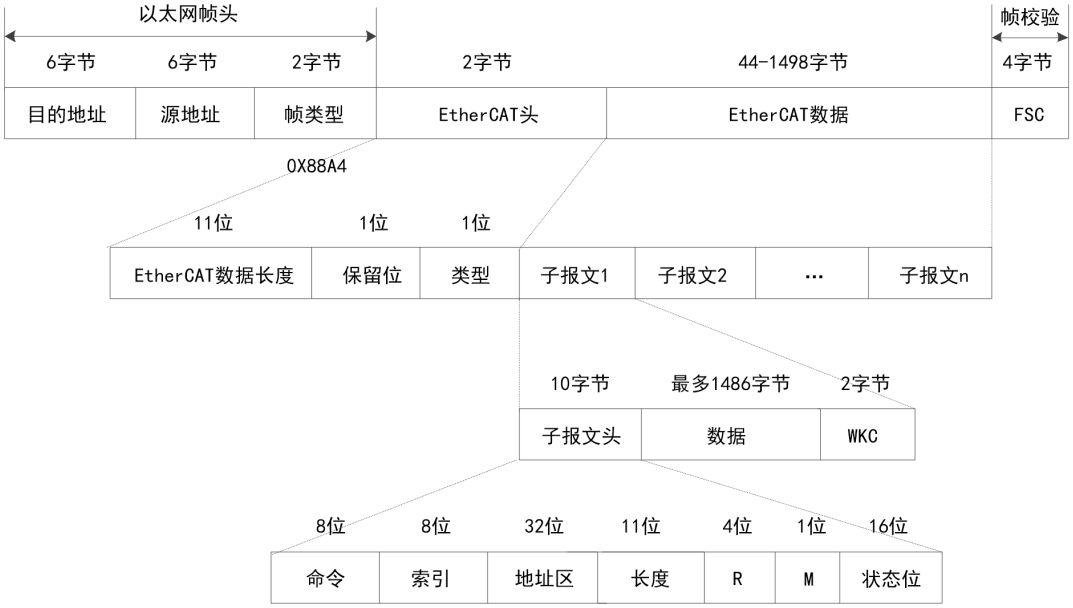
BRD：广播读，读取所有联网从站的物理存储区域

ARMW：一读多写，主站按从站的物理连接顺序读取下一个从站的寄存器值，并将读取的数据写入到其他从站的同一寄存器

FRWM：设置一读多写

控制器和从站的 EtherCAT 通讯

EtherCAT 总线是基于以太网的现场总线，所以 EtherCAT 数据帧仍使用 UDP/IP 以太网数据帧结构，EtherCAT 数据帧结构如下图所示。EtherCAT 数据段包括 2 字节的 EtherCAT 数据头和 44~1498 字节的 EtherCAT 数据，数据区由一个或多个 EtherCAT 子报文组成。EtherCAT 数据可以以某种协议定义和解析，只要主站和从站都符合定义的协议即可，目前使用比较多的协议有两种：COE（CANopen Over EtherCAT）和 SOE(Sercos Over EtherCAT），ZMC 系列运动控制器采用 COE 协议。

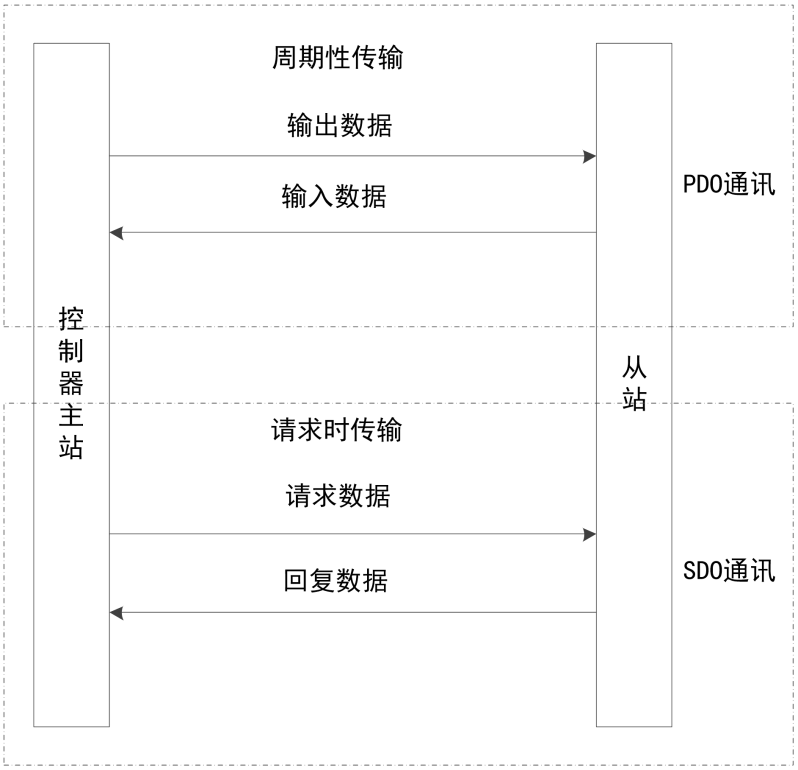


名称		含义
目的地址		目的 MAC 地址
源地址		源 MAC 地址
帧类型		0x88A4
EtherCAT 头	EtherCAT 数据长度	EtherCAT 数据区长度，即所有的 EtherCAT 子报文长度总和
	类型	1 表示主站与从站进行通信；其他保留
EtherCAT 数据		EtherCAT 数据域，包含一个或多个 EtherCAT 子报文
FCS		帧校验序列
子报文头	命令	EtherCAT 命令类型
	索引	帧索引，由主站指引，以防止帧重复或丢失
	地址区	从站地址（不同的寻址方式含义不同）
	长度	子报文数据区的长度
	R	循环帧标志，1 为循环帧，0 为非循环帧
	M	最后一个子报文标志，0 为该子报文为最后一个，1 为子报文非最后一个，后面还有其他子报文
	状态位	所有从站的事件请求寄存器的逻辑或
数据区		EtherCAT 数据
WKC		工作寄存器

控制器 EtherCAT 通讯口和 EtherCAT 从站之间通过 COE（CANopen over EtherCAT）协议进行数据交换。

控制器和从站之间数据传输方式有两种，一种是按指定时间周期性交换数据，称之为 PDO(Process Data Object)，另外一种为请求应答式交换数据，称之为 SDO(Service Data Object)。

ErherCAT 总线通信过程如下：



过程数据对象(PDO)

PDO 数据用于周期性数据读取和控制，读写速度快。主站和从站通过 PDO 进行数据交换时，一方发送数据后，另一方不需要应答。控制器通过运动指令控制 EtherCAT 从站时，控制器和从站之间通过 PDO 方式进行数据交换。

通过 DRIVE_PROFILE 指令可以选择 PDO 配置，目前提供约 20 几种配置选择，查看该指令说明确认，DRIVE_PROFILE 设置不能满足的就自定义 PDO，采用 SDO 相关指令操作数据字典配置需要的 PDO。

可用来在许多节点之间交换即时的数据。可透过一个 PDO，传送最多 8 字节（64 位）数据给设备，或由设备接收最多 8 字节（64 位）的数据。网络上的每个节点都会检测发送节点发出的数据信息，然后节点会决定接收到的信息是否需要处理。一个 PDO 可以由对象字典中几个不同索引的数据组成，规划方式是透过对象字典中对应 PDO mapping 及 PDO 参数索引。

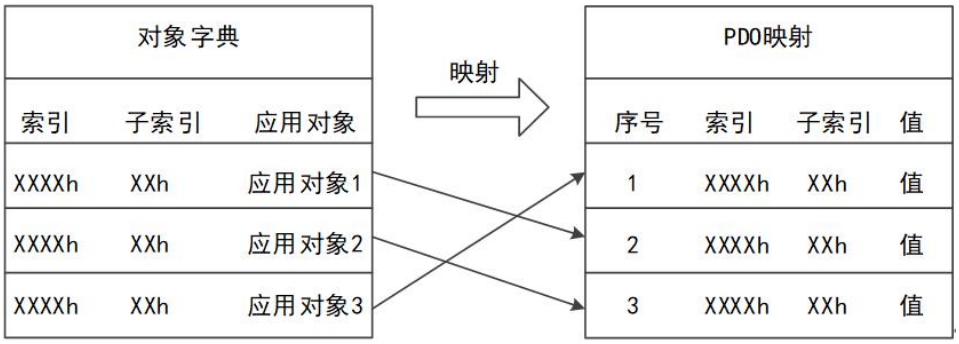
PDO 分为两种：传送用的 TxPDO 及接收用的 RxPDO。一个节点的 TxPDO 是将数据由此节点传输到其他节点，而 RxPDO 则是接收由其他节点传输的数据。一个节点分别有 4 个 TxPDO 及 4 个 RxPDO。PDO 报文数据域中每个字节都用作数据传输，因此报文利用率高。

每个 PDO 在对象字典中用 2 个对象描述：

1）PDO 通讯参数：包含哪个 COB-ID 将被 PDO 使用，传输类型，禁止时间和定时器周期。

2）PDO 映射参数：包含一个对象字典中对象的列表，这些对象映射到 PDO 里，包括它们的数据长度。生产者和消费者必须知道这个映射，以解释 PDO 内容。

PDO 中的所有传送数据必须由对象字典中映射进来：



此时 PDO 的传输顺序为：应用对象 3，应用对象 1，应用对象 2。

服务数据对象(SDO)

SDO 数据用于主站需要读或者写从站参数时才发送通讯数据。此种方式只能主站读或写从站的数据，主站发送数据后从站需要应答。

SDO 可用来存取远端节点的对象字典，读取或设定其中的数据。数据字典的读写分别通过指令 SDO_READ、SDO_READ_AXIS 和 SDO_WRITE、SDO_WRITE_AXIS 实现。

SDO 报文中包含索引和子索引信息，如此方便对象在对象字典中定位，而且对象字典中的复合数据结构易于通过 SDO 访问。SDO 的触发方式为命令响应型，即 SDO 客户发出读/写请求后，SDO 服务器须给予回应；客户端和服务器均可以主动终止 SDO 的传输；请求报文和响应报文通过不同的 COB-ID 进行区分。

SDO 可以传送任意长度的数据。如果传送的数据超过 4 个字节，则必须实行分段传送。最后一段数据包含一个结束标志。

常用数据字典功能

一般在 EtherCAT 中可传输多个协议，ZMC 系列控制器的 EtherCAT 总线通信应用层采用的是 CANopen DS402 协议（也称 CIA402 协议），属于 CANopen 协议的“伺服与运动控制”行规，该协议行规广泛应用在基于 CAN 总线和 EtherCAT 总线网络的运动控制。市面上的大多数设备都按照此标准协议开发，使得大多数设备通用性很强。

CANopen 是一种架构在控制局域网络上的高层通信协协议，为了适应不同类型设备的通讯控制，分别制定了不同的子协议，包括通信子协议及设备子协议。

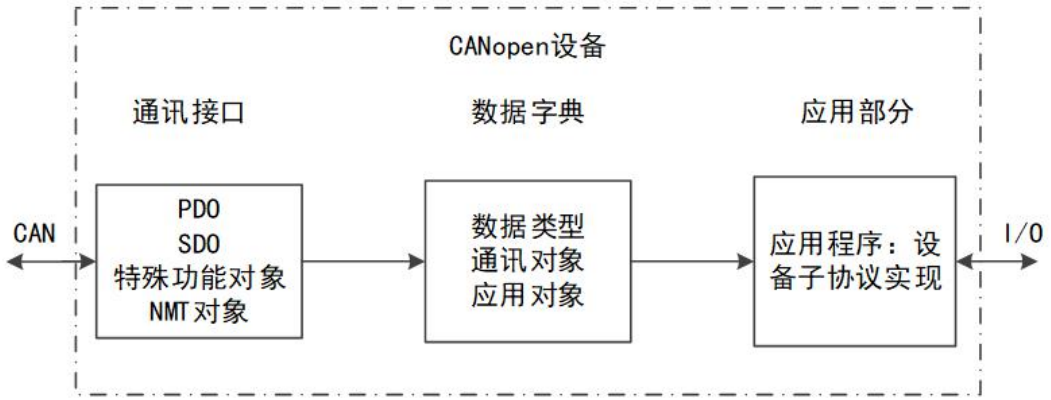
一个 CANopen 设备模块可分为三个部分：

通信部分：PDO 对象，SDO 对象，特殊功能对象，网络管理对象。

对象字典：数据类型，通讯对象，应用对象。

应用部分：应用程序，设备描述。

CANopen 使用基于对象的方法来定义标准设备，每个设备都表现为一组对象的集合，能够被网络所访问。CANopen 设备模型如下图所示，从下图可以看出对象字典是通讯程序和上层应用程序之间的接口。



CANopen 的核心概念是设备对象字典（Object Dictionary，OD），它是一个有序的对象组，每个对象采用一个 4 位 16 进制的索引值来寻址，为了允许访问数据结构中的单个元素，同时定义了一个 8 位的子索引。

对象：设备内特定构建的抽象表现形式，是数据、参数、手法的集合体。

对象字典：包含数据性对象，通讯对象，应用对象的记述内容的数据结构。

CANopen 网络中每个节点都有一个对象字典。对象字典包含了描述这个设备和它的网络行为的所有参数。DS402 对象字典的结构参照下表，节点的对象字典的有关范围在 0x1000-0x9FFF 之间。

索引	内容
0x0001-0x0FFF	协议类型描述，数据类型，行规类型描述，配置表信息
0x1000-0x1FFF	通信区域
0x2000-0x5FFF	设备商自定义对象的功能属性，用于设置功能码，静态参数
0x6000-0x9FFF	行规定义的数据对象，用于设备控制与监视
0xA000-0xFFFF	保留

此处仅对常用的数据对象进行说明。

索引 160h~17FFh 用以 RxPDO 映射的设定，索引 1A00h~1BFFh 用于 TxPDO 映射的设定。

可映射到 RxPDO 中的对象如下：

索引	子索引	对象名称	数据类型	大小	访问	PDO 映射	控制模式
1600h	—	RxPDO 的对象	—	—	—	—	—
	00h	对象数	U8	1 字节	RW	NO	无关
	01h	第 1 个映射对象	U32	4 字节	RW	NO	无关
	02h	第 2 个映射对象	U32	4 字节	RW	NO	无关
	03h	第 3 个映射对象	U32	4 字节	RW	NO	无关
	……						
	20h	第 32 个映射对象	U32	4 字节	RW	NO	无关

RW 读写数据未保存在掉电存储器中，请勿重复映射相同对象，若同一对象多次映射，仅最后一次映射数据有效。

若更改了出厂默认值，每次接通电源均需要重新设定为目标数值。

可映射到 TxPDO 中的对象如下：

索引	子索引	对象名称	数据类型	大小	访问	PDO 映射	控制模式
1A00h	—	TxPDO 的对象	—	—	—	—	—
	00h	对象数	U8	1 字节	RW	NO	无关
	01h	第 1 个映射对象	U32	4 字节	RW	NO	无关
	02h	第 2 个映射对象	U32	4 字节	RW	NO	无关

	03h	第 3 个映射对象	U32	4 字节	RW	NO	无关
	……						
	20h	第 32 个映射对象	U32	4 字节	RW	NO	无关

常用数据字典：

索引	子索引	位长度	对象名称
6040h	00h	10h	控制字
6060H	00h	08h	操作模式
6071H	00h	10h	目标扭矩
6072H	00h	10h	最大扭矩
607AH	00h	20h	目标位置
607FH	00h	20h	最大描述文件速度
6081H	00h	20h	描述文件速度
6083H	00h	20h	描述文件加速度
6084H	00h	20h	描述文件减速度
60B0H	00h	20h	位置偏置
60B1H	00h	20h	速度偏置
60B2H	00h	10h	扭矩偏置
60B8H	00h	10h	外部锁定功能
60E0h	00h	10h	正向扭矩限制值
60E1h	00h	10h	负向扭矩限制值
60FEh	01h	20h	数字输出
60FFh	00h	20h	目标速度

6040H 控制字

主控单元控制伺服的运行状态的控制字，用于控制伺服轴的使能、启动、停止、报警、复位等运行状态。在 EtherCAT 初始化过程里会用到此控制字。

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
控制字	0-65535	U16	RW	RxPDO	全部	NO

位信息详情：

位	含义
0	伺服准备，停止所有运动，置 1 有效
1	接通主电路，置 1 有效
2	快速停机，置 1 有效
3	使能打开
4-6	控制模式依存位
7	伺服错误信息清除，上升沿有效
8	暂停
9-15	预留

使用控制字时需要多个位配合使用达成某个功能，单个位赋值没有意义。

在原点回零模式中的作用：

位	名称	值	说明
4	原点回零开始	0	原点回零动作无法开始

		1	开始或继续执行原点回零动作
8	停止	0	位 4 有效
		1	按设置停止

6041H 状态字

用以读取伺服当前的运行状态。

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
状态字	0-65535	U16	RO	RxPDO	全部	NO

位信息详情：

位	含义
0	伺服无故障
1	等待打开伺服使能
2	伺服正在运行
3	伺服故障
4	接通主电路
5	快速停机
6	伺服准备好
7	报警
8	预留
9	远程控制
10	为 0 没到目标状态，为 1 到达（速度或位置）
11	为 0 未达到软件位置限制，为 1 到达，软件位置限制生效
12-13	控制模式依存位
14	预留
15	为 0 原点回零未完成，为 1 已回零

在原点回零模式中的作用：

位 13	位 12	位 10	说明
0	0	0	正在执行原点回零动作
0	0	1	原点回零动作中断中或尚未开始回零动作
0	1	0	原点回零动作已完成，但尚未到达目标位置
0	1	1	原点回零动作正常完成
1	0	0	发生原点回零异常，速度不为零
1	0	1	发生了原点回零异常，速度为零
1	1	0	保留
1	1	1	保留

6060H 控制模式设定

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
控制模式设定	-128~127	I8	RW	RxPDO	全部	YES

设置值详情：

设置值	控制模式
-128~-1	预留
0	模式未设置或未改变

1	轮廓位置控制模式
2	速度控制模式
3	轮廓速度控制模式
4	轮廓转矩控制模式
5	预留
6	原点复位位置控制模式
7	补偿位置控制模式
8	周期同步位置控制模式
9	周期同步速度控制模式
10	周期同步转矩控制模式
11~127	预留

伺服驱动器处于任何状态下，从轮廓位置模式或周期同步位置模式切换成其他模式后，未执行的指令将被忽略不执行，切换过程中，首先执行快速停机，完成停机后，切换目标模式。

伺服处于回零模式，且正在运行时，不可切换模式；回零过程完成或被中断时，可以切换模式。

伺服在运行状态中收到切换为周期同步控制模式时，请间隔至少 1ms 再发送指令，否则会导致指令丢失或执行错误。

6061H 控制模式查看

6060H 设置后，通过 6061H 查看伺服驱动器控制模式设置是否成功。

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
控制模式查看	-128~127	I8	RO	TxPDO	全部	NO

设置值详情：

设置值	控制模式
-128~-1	预留
0	模式未设置或未改变
1	轮廓位置控制模式
2	速度控制模式
3	轮廓速度控制模式
4	轮廓转矩控制模式
5	预留
6	原点复位位置控制模式
7	补偿位置控制模式
8	周期同步位置控制模式
9	周期同步速度控制模式
10	周期同步转矩控制模式
11~127	预留

通过变更 6060H 的值，可以切换驱动器的控制模式，切换后再次使用 6061H 判断现在的控制模式。控制模式切换时，请更新和 6060H 同步的控制模式相关的 RxPDO 的对象，在变更后的控制模式下，不支持的对象的值是不定的。

控制模式的切换到动作执行完成需要花费 2ms，此期间 6061H 和控制模式相关的 TxPDO 的对象值是不定的，控制器模式连续切换短于 2ms 间隔会发生命令异常保护。

控制模式的切换一定在电机停止中进行，无法立即切换，需要先停机后切换，无法保证电机动作中（包含原点回零、减速停止）控制模式切换情况的动作。

6098H 回零模式

设置原点复位的方式。

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
回零方式	-128~127	I8	RW	RxPDO	HM	YES

通过 EtherCAT 总线回零时先要设置轴的回零方式，根据需要的回零模式选择参数，具体设置方法参见原点回零章节。

原点正在回零时无法更换回零方式，需要等待电机停止后再更换。

6099H 回零速度

找原点时运行速度设置。

名称	子索引	数据范围	数据类型	读写	PDO	控制模式	EEPROM
回零方式	00h	2	U8	RO	/	HM	NO
	01h	0~4294967295	U32	RW	RxPDO	HM	YES
	02h	0~4294967295	U32	RW	RxPDO	HM	YES

609AH 回零加速度

找原点时运行加速度设置

名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
原点复位加速度	0~4294967295	U32	RW	RxPDO	HM	YES

附录Ⅳ RTEX 总线

RTEX 总线简介

松下利用在控制、驱动等方面的技术优势，自主开发出高速网络化的 RTEX 总线，以契合市场的高阶需求。RTEX 是一个开放的总线，支持与各大控制器厂商形成解决方案。

RTEX 是适应于小系统的实时总线，小型设备组成的流水线更具有快速应变的优势。

目前，RTEX 总线支持 32 个节点，每个完整的数据包都包括了 32 个节点的输出信息与反馈信息，它共分为 64 个数据块。此外，RTEX 总线提供控制字寄存器与状态字寄存器。其中每个数据块大小为 16bytes，只包含有必要的位置、速度信息以及一些命令字和状态字。基于 RTEX 总线的主站为核心，主机侧都会一次性对所有节点发出控制命令，同时获取到所有节点的反馈信号，并完成对所有节点输出的控制。

若要提供大于 32 节点的网段，RTEX 同样可以通过追加通道数来实现节点数的倍增，每个通道最后汇聚于计算机 CPU，因而实时性不会受到任何影响。

在伺服马达的控制中，RTEX 总线可以进行 PP（轮廓位置）、CP（周期位置）、CV（周期速度）及 CT（周期转矩）等方式的控制。

控制模式		描述
Nop	Nop	一般不使用，仅在网络确立后暂时送信时使用，除此之外的情况不要使用 如果接受了此指令，基于之前的指令进行控制
轮廓位置控制模式（PP）	Profile Position Mode	设定目标位置、目标速度、加减速度等参数 在伺服驱动器内部生成位置指令动作的位置控制模式
周期位置控制模式（CP）	Cyclic Position Mode	在上位装置生成位置指令，在指令更新周期下更新指令位置后动作的位置控制模式
周期速度控制模式（CV）	Cyclic Velocity Mode	在上位装置生成速度指令，在通信周期下更新指令速度后动作的速度控制模式
周期转矩控制模式（CT）	Cyclic Torque Mode	在上位装置生成转矩指令，在通信周期下更新指令转矩后动作的转矩控制模式

PP 控制通常指点位的控制，就是发出目标位置坐标来控制位置，它常用于点到点的控制，不需要关心行走轨迹。CP 控制则需要通过上位控制器不断进行加减速度，位置的计算，这不仅要保证最终的位置，同时还要确保到达最终位置的轨迹也达到预期值。所以，利用 RTEX 总线，可以保证伺服电机的同步运行，从而避免龙门控制等机械容易因为运行速度不一致而导致卡死的现象发生。

RTEX 总线优势

RTEX 总线就是实时连接着控制器和伺服驱动器的高速运动控制总线，RTEX 总线仅支持连接松下伺服驱动器使用，技术特点如下：

1）在 0.5ms 的时间内实现最大为 32 节点的通信；

2）基于 100Mbps 的全双工以太网通讯，支持环状拓扑结构；

3）具备高精度的插补同步时钟，保证全部伺服可与上位装置同时动作，同时实现 CP 控制；

4）允许整个网络的线缆长度最大 200 米，节点间线缆长度最大 60 米；

5）抗噪性达到 IEC61000-4-4 标准，能适应 2.5kV/s 的噪声；

6）全部控制系统采用数字网络信号连接，达到低成本、易连接、易扩展的性能；

7）连接线缆少。

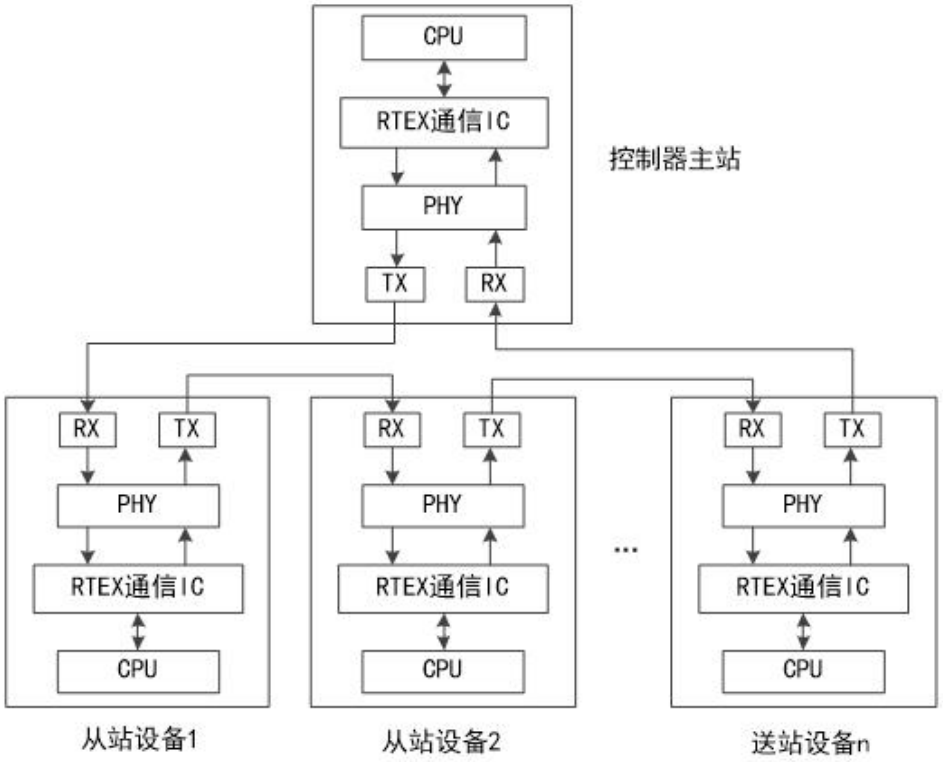
RTEX 总线给用户带来的利处是：开发简单、高性能、简单低成本、高可靠性。

RTEX 总线通讯

搭载 RTEX 通信 IC 的主站上位装置和从站进行环形连接，构成多轴伺服通讯系统构成如下如所示，其中 PHY 为物理层芯片。连接线应使用带屏蔽层的双绞线电缆。

通信和伺服的同步位确立状态下，指令收信、相应送信的时序是不定的，同步是否完成可以通过指令读取当前状态来判断。

RTEX 通信 IC 包括送信存储器、收信寄存器、控制寄存器和状态寄存器，送信存储器用于存储数据指令，收信寄存器用于存储响应数据，指令数据可分为 16 字节和 32 字节两种，32 字节数据使用两个连续的 16 字节数据块。



1. 指令数据块

指令数据块构成如下表（16 字节模式/32 字节模式共通），指令从控制器主站发送给伺服从站。

Byte	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	C/R(0)	Update_Counter		MAC-ID(0~31)				
1	TMG_CNT	Command_Code						
		cyclic 指令代码			非 cyclic 指令代码			
2	Servo_On	0	0	Gain_SW	TL_SW	Homing_Ctrl	0	0
3	Hard_Stop	Smooth_Stop	Pause	0	SL_SW	0	EX-OUT2	EX-OUT1
4~7	Command_Data1							
8~11	Command_Data2							
12~15	Command_Data3							

未使用的 bit 请设定为 0，多个字节数据自动配置为低字节优先。

指令数据块各部分含义如下：

名称	描述

指令帧头	C/R	区别指令/响应 指令设定为 0 设定为 0 以外发生 Err86.0（数据异常保护 1）报警
	Update_Counter	设定指令更新周期下加起来的值 在伺服侧将检出指令更新的时间作为目的 伺服将此数据反馈到响应中，也可以利用看门狗时钟确认伺服是否正常动作
	MAC-ID	设定伺服驱动器的节点地址 若设定的节点地址与实际的设定值不同，发生 Err86.0（数据异常保护 1）报警
指令代码	TMG_CNT	在轴间全同步模式下使用
	Command_Code	设定指令代码，分为指令位置等实时数据传送用到 cyclic 指令代码和作为参数设定等的事件触发型数据传送用的非 cyclic 指令代码两大类
	Command_Data1	设定用 cyclic 指令代码规定的数据
	Command_Data2	设定用非 cyclic 指令代码规定的数据
	Command_Data3	设定用非 cyclic 指令代码规定的数据

cyclic 指令代码：分配到指令 byte1 的 bit6~4 规定 byte4~7 的数据。

非 cyclic 指令代码：分配到指令 byte1 的 bit3~0 规定 byte8~15 的数据。

cyclic 指令代码数据		非 cyclic 指令代码数据	
Bit6~4	作用	Bit3~0	作用
0	NOP	0	通常指令
1	轮廓位置控制	1	重启指令
2	周期位置控制	2	系统 ID 指令
3	周期速度控制	4	原点复位指令
4	周期转矩控制	5	报警指令
5~7	预留	6	参数指令
		7	轮廓指令
		10	监视器指令
		3、8~9、11~15	预留

控制位：

Byte	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
2	Servo_On	0	0	Gain_SW	TL_SW	Homing_Ctrl	0	0
3	Hard_Stop	Smooth_Stop	Pause	0	SL_SW	0	EX-OUT2	EX-OUT1

释义：

名称	描述
Servo_On	设定伺服 ON/OFF 指令；0-ON，1-OFF 在接口连接器（X4）分配外部伺服 ON 输入（EX-SON）时，通过与外部伺服 ON 输入的 AND 条件执行伺服 ON 指令
Gain_SW	设定增益切换指令；0-选择第 1 增益，1-选择第 2 增益 实时自动调整无效，当第 2 增益有效，且根据 RTEX 通信增益切换有效时，此时可以切换增益
TL_SW	设定转矩限制切换指令 参数 Pr5.21（转矩限制选择）的值为 3 或 4 时，此信号有效
Homing_Ctrl	设定原点复位动作控制指令 此 bit 为 1 时，检出原点基准触发信号

	原点复位指令以外，此信号无效
Hard_Stop	轮廓位置控制模式时，设定为 1 时，立即停止内部指令生成处理，结束轮廓动作 停止后此 bit 即使返回 0 也不在重新开始停止前的动作，需要重新设置动作
Smooth_Stop	轮廓位置控制模式时，设定为 1 时，通过设定的减速度减速停止，结束轮廓动作 停止后此 bit 即使返回 0 也不在重新开始停止前的动作，需要重新设置动作
Pause	轮廓位置控制模式时，通过设定的减速度减速停止，暂时结束轮廓动作 停止后或者减速中如果此 bit 返回 0，重新开始停止前的动作
SL_SW	设定转矩控制时速度限制的切换指令 参数 Pr3.17（速度限制选择）为 1 时，此信号有效
EX-OUT2 EX-OUT1	进行外部输出信号的 RTEX 操作输出（EX-OUT1/EX-OUT2） 在接口连接器（4X）分配 RTEX 操作输出（EX-OUT1/EX-OUT2）时，此信号变为有效 此信号对伺服控制无影响

2．响应数据块

响应数据块构成如下表（16 字节模式/32 字节模式共通），响应通过从站被发送到主站。

Byte	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	C/R(1)	Update_Counter_ Echo		Actual_MAC-ID(0~31)				
1	CMD_ Error	Command_Code_Echo						
2	Servo_ active	Servo_ ready	alarm	warning	Torque_ limited	Homing_c omplete	In_Progres s/ac_off	in_posi tion
3	SI-MON5 /E-STOP	SI-MON4/ EX-SON	SI-MON3/ EXT3/STOP	SI-MON2/ EXT2/RET	SI-MON1/ EXT1	HOME	POT/NOT	NOT/P OT
4~7	Response_Data1							
8~11	Response_Data2							
12~15	Response_Data3							

未使用 bit 将返回 0，多个字节数据自动配置为低字节优先。

指令数据块各部分含义如下：

名称		描述
响应帧头	C/R	区别指令/响应 响应将 1 返回
	Update_Counter_Echo	将 Update_Counter 值的 echo back 值返回 使用到伺服驱动器收信处理的有无确认
	Actual_MAC-ID	返回伺服驱动器的节点地址 是伺服驱动器的实际设定值（电源接通时的旋转开关设定值），不是 echo back
指令代码	CMD_Error	指令错误发生时返回 1 在接收指令阶段（处理执行前），错误发生时变为 1
	Command_Code_Echo	将指令代码 echo back 值返回
	Response_Data1	返回 Pr7.29（监视器选择 1）指定的监视器数据 配置为低字节优先
	Response_Data2	将非 cyclic 指令代码规定的响应数据返回 将非 cyclic 指令代码为 0h（通常指令）返回时，Pr7.30（监视器选择 2）指定监视器数据，Pr7.30=0 时与驱动器互换，返回实际速度（Type_Code=05h）配置为低字节优先
	Response_Data3	将非 cyclic 指令代码规定的响应数据返回

		将非 cyclic 指令代码为 0h（通常指令）返回时，Pr7.31（监视器选择 3）指定监视器数据，Pr7.31=0 时与驱动器互换，返回转矩（Type_Code=06h） 配置为低字节优先

响应状态标志：

Byte	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
2	Servo_ Active	Servo_ Ready	Alarm	Warning	Torque_Limited	Homing_Complete	In_Progress/ac_off/ Pr7.112	In_Position

释义：

名称	作用
Servo_Active	伺服 ON 状态下（电机 ON-通电，OFF-不通电）将 1 返回 通过动态制动器等减速时，伺服是 OFF 状态 Pr7.24（扩展功能设定 3）bit4=1 时，Servo_Active 标志是到伺服 ON 指令可以接收的状态后，强制返回伺服 OFF 状态 在 PANATERM 指令的伺服 ON 状态时返回 0
Servo_Ready	伺服准备状态时返回 1 主电源确认、未发生报警、通信和伺服同步确认的条件全部成立时变为 1
Alarm	发生报警返回 1
Warning	发生警告返回 1 箝位警告状态通过 Pr6.27（警告箝位状态设定）进行设定
Torque_Limited	转矩限制下将 1 返回 根据参数等限制内部转矩指令状态下变为 1 关于转矩控制时的输出条件，根据 Pr7.03（转矩限制中输出设定）可以设定
Homing_Complete	在原点复位完成（箝位模式，带停止功能的箝位模式除外）时返回 1，在完成（原点确定）后保持 1 在功能扩展版 5 之前，如在增量模式下，收到原点复位指令（箝位模式，带停止功能的箝位模式除外）后，会暂时清 0 从功能扩展版 6 起，不论在增量模式下，还是在绝对式模式下（绝对式编码器作为绝对式使用时，或者全闭环控制模式下外部位移传感器作为绝对式使用时），都会在收到原点复位指令时暂时清 0 执行原点复位后为 0，之后若取消原点复位，仍保持为 0 在绝对式模式下若原点复位失败并重启电源，则值会从 1 开始 绝对式模式时（绝对式编码器作为绝对式使用时，或者全闭环控制模式下外部位移传感器作为绝对式使用时），控制电源接通后，为了进行原点确认，初始值为 1，相反增量式模式时为 0 重启指令执行时也被初始化成与控制电源接通后相同的位置信息，此 bit 也被初始化 在来自安装支持软件的「试运转功能」、「频率特性解析功能（FFT 功能）」、「绝对式编码器的多圈清除」、「Z 相搜索功能」、「配合增益功能」、「引脚定义设定」执行完成时，也与控制电源接通后 同样，位置信息会进行初始化，并且本 bit 也会进行初始化 增量式模式时，回退动作后变为 0
In_Progress/AC_OFF/Pr7.112	In_Progress 设定时，轮廓位置控制模式下，内部指令位置生成中将 1 返回，内部指令位置生成完成（退出完成）将 0 返回 AC_OFF 设定时，发生主电源 OFF 警告将 1 返回 在 Pr7.23 bit15=1（根据 Pr7.112 的设定）且 Pr7.112（RTEX 通信状态标志选择）中设定 RET_Status 时，回退动作中以及 Err85.2/Err87.3（回退动作异常）判定中将返回 1；回退动作中，请维持 Servo_On=1。如果不维持，则会发生 Err85.2/Err87.3（回退动作异常）

<table>
<tr><td></td><td>在 Pr7.23 bit15=1（根据 Pr7.112 的设定）且 Pr7.112（RTEX 通信状态标志选择）中设定 V_Full_Status 时，在虚拟全闭环控制模式状态中将返回 1
当 Pr7.23 bit15=1（按照 Pr7.112 的设定）并且在 Pr7.112（RTEX 通信状态旗标选择）中设定 CMP_OUT_Status 时，在位置比较输出功能有效时返回 1</td></tr>
<tr><td>In_Position</td><td>如下表表示，各控制模式下标志的功能有所改变
<table>
<tr><th>功能</th><th>控制</th><th>模式内容</th></tr>
<tr><td>定位完成</td><td>位置控制</td><td>定位完成状态下将 1 返回
与外部输出信号下的定位完成输出一样，输出条件通过参数 Pr4.31（定位完成范围）、Pr4.32（定位完成输出设定）、Pr4.33（INP 保持时间）进行设定
在虚拟全闭环控制模式状态中，虚拟全闭环控制模式时外部位移传感器位置变化量判定功能有效（Pr6.98 bit9=1）时，如果外部位移传感器位置变化量变为 Pr3.32（虚拟全闭环控制模式外部位移传感器位置变化量判定阈值）的设定值以上便返回 1</td></tr>
<tr><td rowspan="2">速度一致</td><td>速度控制</td><td>电机实际速度与指令速度一致时返回 1
与外部输出信号的速度一致输出（V-COIN）同样，输出条件通过参数 Pr4.35（速度一致宽度）进行设定</td></tr>
<tr><td>转矩控制</td><td>电机实际速度与速度限制值一致时返回 1
输出条件通过参数 Pr4.35（速度一致宽度）进行设定</td></tr>
</table></td></tr>
</table>

输入信号响应状态标志：

Byte	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
3	SI-MON5/E-STOP	SI-MON4/EX-SON	SI-MON3/EXT3/STOP	SI-MON2/EXT/RET	SI-MON1/EXT1	HOME	POT/NOT	NOT/POT

未进行端子功能分配时，此状态的 bit 的值为 0。

另外，同一端子在每个控制模式可以分配另外的功能，但是切换控制模式时，请注意有效状态和无效状态的改换。基本上每个端子在所有的控制模式下请分配相同的功能。

因为 SI-MON1 和 EXT1、SI-MON2 和 EXT2 和 RET、SI-MON3 和 EXT3 和 STOP、SI-MON4 和 EX-SON、SI-MON5 和 E-STOP 分别配置相同的 bit 所以不能重复设定。重复设定时发生 Err33.0（输入重复分配异常 1 保护）或者 Err33.1（输入重复分配异常 2 保护）。

此状态非物理状态（输入晶体管 ON/OFF 状态），将逻辑状态（功能激活侧为 1）返回。但是关于驱动禁止输入（POT/NOT）可以设定状态的逻辑。

EXT1、EXT2、EXT3 并非箝位完成状态，表示输入信号的状态。

关于驱动禁止输入（POT/NOT），功能无效时（Pr5.04=1）关于响应条件、状态位配置、状态逻辑根据参数 Pr7.23（RTEX 功能扩展设定 2）5.04=1）可以进行下述设定。MINAS-A4N 系列为 CCWL、CWL，但是 MINAS-A6N 系列变化为 POT、NOT，所以与 MINAS-A4N 系列互换，需要恰当的设定此参数与 Pr0.00（旋转方向设定）。

RTEX 参数读写使用 DRIVE_READ 指令和 DRIVE_WRITE 指令操作驱动器参数进行相关设定。

相关设定参数：

分类	No.	属性	参数名称	设定范围	单位	描述
0	00	C	旋转方向设定	0~1	—	设定指令方向与电机旋转方向的关系 0-CW 为正方向，1-CCW 为正方向

0	01	R	控制模式设定	0~6	—	设定伺服驱动器的控制模式 0：半闭环控制 位置/速度/转矩控制模式可切换 6：全闭环控制 只有位置控制（轮廓/周期）
0	08	C	电机每旋转一圈指令脉冲数	0~2^{23}	pulse	设定电机每旋转一圈指令脉冲数
0	09	C	电子齿轮分子	0~2^{30}	—	设定电子齿轮比的分子
0	10	C	电子齿轮分母	0~2^{30}	—	设定电子齿轮比的分母
0	13	B	第一转矩限制	0~500	%	设定电机输出转矩的第 1 限制值，参数值受适用电机最大转矩限制
3	12	B	加速时间设定	0~10000	ms	设定加速的时间
3	12	B	减速时间设定	0~10000	ms	设定减速的时间
3	14	B	S 加减速设定	0~1000	—	对加减速进行 S 曲线处理
3	17	B	速度限制选择	0~1		选择速度限制； 0-速度限制 1，1-速度限制 2
3	21	B	速度限制值 1	0~20000	r/min	设定速度限制值，内部值被 Pr5.13「过速度等级设定」、Pr6.15「第 2 过速度等级设定」、以过速度保护水平的内部值的最小设定速度进行限制
3	22	B	速度限制值 2	0~20000	r/min	设定 Pr3.17「速度限制选择」=1 设定时、SL_SW 为 1 时的速度限制值。 内部值被 Pr5.13「过速度等级设定」、Pr6.15「第 2 过速度等级设定」、以过速度保护水平的内部值的最小设定速度进行限制
5	21	B	转矩限制选择	1~4	—	设定正方向/负方向的转矩限制选择方式 在设定为 0 时在内部设定为 1
5	22	B	第二转矩限制	0~500	%	设定电机的输出转矩的第 2 限制值，电机的最大转矩限制
5	25	B	正方向转矩限制	0~500	%	Pr5.21（转矩限制选择）=4 设定时，TL_SW 为 1 时，设定正方向转矩限制，参数值被适用电机的最大转矩限制
5	26	B	负方向转矩限制	0~500	%	Pr5.21（转矩限制选择）=4 设定时，TL_SW 为 1 时，设定正方向转矩限制，参数值被适用电机的最大转矩限制
7	10	A	软件限制功能	0~3	—	设定轮廓位置控制(PP)时的软件限位功能的有效/无效 有效时的软件限位值，通过 Pr7.11(正侧软件限位值)与 Pr7.12(负侧软件限位值)设定 0-两侧软件限位有效 1-正侧软件限位无效、负有效 2-正侧软件限位有效、负无效 3-两侧软件限位无效 由于本设定值而无效的限位信号(PSL/NSL)，RTEX 通信状态为 0，原点复位未完成时也为 0
7	11	A	正向软件限位值	-1073741823~1073741823	指令单位	设定正方向以及负方向的软件限位 超过限制时，RTEX 通信的状态

7	12	A	负向软件限位值	-1073741823~1073741823	指令单位	
7	20	R	RTEX 通讯周期设定	-1~12	—	设定 RTEX 的通讯周期 -1：将 Pr7.91 的设定设为有效 3：0.5ms 6：1.0ms
7	21	R	RTEX 指令更新周期比设定	1~2	—	设定 RTEX 通信的通信周期和指令更新周期的比 设定值=指令更新周期/通信周期 1：1 倍 2：2 倍
7	22	R	RTEX 功能扩展设定 1	-32768~32767	—	bit0 设定 RTEX 通信数据的大小 0：16 字节模式 1：32 字节模式 bit1 设定使用了 TMG_CNT 多个轴的同步模式，未使用 TMG_CNT 时请设为 0 0：轴间半同步模式（部分非同步） 1：轴间完全同步模式 bit4 半闭环控制时外部位移传感器位置信息监视器功能设定 0：无效 1：有效 （全闭环控制时跟此 bit 的设定无关，可以监测外部位移传感器的位置信息）
7	91	R	RTEX 通信周期扩展设定	0~2000000	ns	设定 Pr7.20=1 时的 RTEX 通信的通信周期 只可设定 62500,、125000、250000、500000、1000000、2000000 这几个参数值，否者会发生 Err93.5“参数设定异常保护 4”

A：一直有效。

B：禁止电机动作中及指令发出中变更参数。

C：在控制电源复位、RTEX 通信的复位指令的软件复位模式或属性 C 参数有效化模式执行后有效。

R：控制电源重启后有效。

RTEX 的通信周期（Pr7.20、Pr7.91）和指令更新周期（Pr7.21）需要与上位装置的周期一致，同时，RTEX 的扩展功能（Pr7.22）的设定也需要与上位装置一致，设定若不同无法保证动作执行。

模式设定示例如下：

通信周期 0.5ms，指令更新周期 1ms，半闭环控制，16 字节模式，轴间半同步模式的情况下的设置。

Pr0.01=0（半闭环控制）

Pr7.20=3（通信周期 0.5ms）

pr7.21=2（指令更新周期 1ms=0.5ms*2 倍）

pr7.22=0（16 字节模式、轴间半同步模式）

（在 Pr7.20 不等于-1 时，可以不设定 Pr7.91）

驱动器不动作原因排查：

序号	项目	描述
0	无原因	无法检出不旋转原因，通常是可旋转的状态
1	伺服未处于准备状态	驱动器的主电源未输入

		报警发生 通信和伺服的同步未完成 重启指令下属性 C 参数有效化模式处理中等
2	伺服未使能	伺服 ON 指令未输入；指令的 Servo_On bit 为 0，EX_SON（外部伺服 ON 输入）进行了分配，信号为 OFF 等
3	驱动禁止输入有效	Pr5.05=0～1（驱动禁止时时序，立即停止除外）时 Pr5.04=0（驱动禁止输入有效），正方向驱动禁止输入（POT）为 ON 时，动作指令为正方向；负方向驱动禁止输入（NOT）为 ON，动作指令为负方向 Pr5.05=2（驱动禁止时时序：立即停止）时 Pr5.04=0（驱动禁止输入有效），与是否有动作指令输入无关，正方向驱动禁止输入（POT）或者负方向驱动禁止输入（NOT）为 ON 的状态下停止
4~5	转矩限制设定小	有效的转矩限制设定值，设定在额定的 5%以下
7	位置指令输入频率低	每个控制周期的位置指令为 1 指令单位以下
10	RTEX 通信的指令速度小	来自 RTEX 通信的指令速度设定为 30r/min 以下
11	厂家使用	—
12	RTEX 通信的指令转矩小	来自 RTEX 通信的指令转矩减小到额定转矩的 5%以下
13	速度限制小	Pr3.17=0 时，Pr3.21 速度限制值设定在 30r/min 以下 Pr3.17=1 时，指令的 SL_SW bit 指定的参数(Pr3.21 或者 Pr3.22)的速度限制值设定 30r/min 以下
14	其他原因	不是主要原因 1～13 的任一个，电机不运转（指令小、负载重、锁定、碰撞、驱动器、电机的故障等）

附录Ⅴ CAN 总线

CAN 总线简介

控制器局域网总线 CAN（Controller Area Network）是一种用于实时应用的分布式控制系统串行通讯协议总线，是由德国博世公司在 20 世纪 80 年代专门为汽车行业开发的一种串行通信总线。CAN 总线推荐使用带屏蔽的双绞线来传输信号，由于其高性能、高可靠性以及独特的设计而越来越受到人们的重视，而且能够检测出产生的任何错误，是世界上应用最广泛的现场总线之一。

CAN 协议用于各种不同元件之间的通信，以此取代昂贵而笨重的配电线束，具有很高的实时性能和应用范围，从位速率最高可达 1Mbps 的高速网络到低成本多线路的 50Kbps 网络都可以任意搭配，该协议的方便性使其在汽车、安全防护、其他自动化和工业领域得到了广泛的应用。

CAN 总线技术规范（Version 2.0）包括 A 和 B 两个部分。其中 2.0A 给出了 CAN 报文标准格式，地址范围由 11 个识别位定义，而 2.0B 给出了标准的和扩展的两种格式，其地址范围由 29 个识别位定义。正运动控制器 CAN 总线标准帧和扩展帧均支持。

CAN 总线的特点：

1）具有实时性强、传输距离较远、抗电磁干扰能力强、成本低等优点。

2）采用双线串行通信方式，检错能力强，可在高噪声干扰环境中工作。

3）具有优先权和仲裁功能，多个控制模块通过 CAN 总线连接到控制器，形成多主机局部网络。

4）可根据报文的 ID 决定接收或屏蔽该报文。

5）可靠的错误处理和检错机制。

6）发送的信息遭到破坏后，可自动重发。

7）节点在错误严重的情况下具有自动退出总线的功能。

8）报文不包含源地址或目标地址，仅用标志符来指示功能信息、优先级信息。

CAN 总线的工作原理

CAN 总线使用串行数据传输方式，可以 1Mb/s 的速率在 40m 的双绞线上运行，也可以使用光缆连接，而且在这种总线上支持多主控制器。当 CAN 总线上的一个节点（站）发送数据时，它以报文形式广播给网络中所有节点。对每个节点来说，无论数据是否是发给自己的，都对其进行接收。

每组报文开头的 11 位（标准格式）/29 位（扩展格式）字符为标识符，定义了报文的优先级，这种报文格式称为面向内容的编址方案。在同一系统中标识符是唯一的，不可能有两个站发送具有相同标识符的报文，当几个站同时竞争总线读取时，这种配置十分重要。

当一个站要向其它站发送数据时，该站的 CPU 将要发送的数据和自己的标识符传送给本站的 CAN 芯片，并处于准备状态；当它收到总线分配时，转为发送报文状态。CAN 芯片将数据根据协议组织成一定的报文格式发出，这时网络上的其它站处于接收状态。每个处于接收状态的站对接收到的报文进行检测，判断这些报文是否是发给自己的，以确定是否接收它。

由于 CAN 总线是一种面向内容的编址方案，因此很容易建立高水准的控制系统并灵活地进行配置。我们可以很容易地在 CAN 总线中加进一些新站而无需在硬件或软件上进行修改。当所提供的新站是纯数据接收设备时，数据传输协议不要求独立的部分有物理目的地址。它允许分布过程同步化，即总线上控制器需要测量数据时，可由网络上获得，无须每个控制器都有自己独立的传感器。

CAN 总线协议的基本概念

1．CAN 总线协议内容

从 OSI 网络模型的角度来看同，现场总线网络一般只实现了第 1 层（物理层）、第 2 层（数据链路层）、第 7 层（应用层）。因为现场总线通常只包括一个网段，因此不需要第 3 层（传输层）和第 4 层（网络层），也不需要第 5 层（会话层）第 6 层（描述层）的作用。

CAN 现场总线仅仅定义了第 1 层、第 2 层，实际设计中，这两层完全由硬件实现，设计人员无需再为此开发相关软件或固件。

同时，CAN 只定义物理层和数据链路层，没有规定应用层，本身并不完整，需要一个高层协议来定义 CAN 报文中的 11/29 位标识符、8 字节数据的使用。

2．CAN 总线协议基本规则

总线访问：采用载波监听多路访问，CAN 控制器能够在总线空闲时，就是节点侦听到网络上至少存在 3 个空闲位（隐性位）时开始发送，采用硬同步，所有的控制器同步位于帧起始的前沿。过了一定时间，并在一定条件后，重同步。

仲裁：各节点向总线发电平时，也对总线上电平进行读取，并与自身发送的电平进行比较，相同则发下一位，直至全部发完。不同则说明网络上有更高优先级的信息帧正在发送，即停止发送，退出竞争。

编码/解码：帧起始域，仲裁域，控制域，数据域和 CRC 序列均使用位填充技术进行编码，就是 5 个连续的同状态电平插入一位与它相补的电平，还原时每 5 个同状态电平后的相补电平被删除。

出错标注：当检测到位错误、填充错误、形式错误或应答错误时，检测出错条件的 CAN 控制器将发送一个出错标志。

超载标注：一些控制器会发送一个或多个超载帧以延迟下一个数据帧或远程帧的发送。

3．CAN 总线节点数量

CAN 网络上的节点不分主从，任一节点在任意时刻均可主动地向网络上其他节点发送信息，通信方式灵活，利用这一特点可以方便地构成多机备份系统，CAN 只需通过报文滤波即可实现点对点、一点对多点及全局广播等几种方式传送接收数据。 CAN 上的节点数主要决定于总线驱动电路，报文标识符可达 2032 种（CAN 2.0A），而扩展标准（CAN 2.0B）的报文标识符几乎不受限制。

CAN 总线报文传输

在 CAN2.0 协议版本中有两种不同的帧格式，不同之处为标识符域的长度不同，含有 11 位标识符的帧称之为标准帧，而含有 29 位标识符的帧称为扩展帧。如 CAN1.2 版本协议所描述，两个版本的标准数据帧格式和远程帧格式分别是等效的，而扩展格式是 CAN2.0B 协议新增加的特性。

无论是哪种帧格式，在报文传输时都有以下四种不同类型的帧，不同的帧具有不同的传输结构，下面将分别介绍四种传输帧的结构，只有严格按照该结构进行帧的传输，才能被节点正确接收和发送。

数据帧：从发送端携带数据到接收端。

远程帧：总线单元发出远程帧，请求发送具有同一识别符的数据帧。

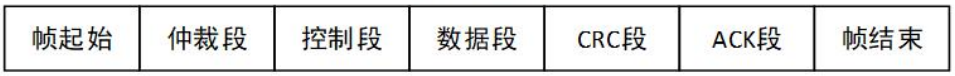
错误帧：任何单元检测到总线错误就发出错误帧。

过载帧：过载帧用以在先行和后续的数据帧（或远程帧）之间提供附加的延时。同时帧间空间用来间隔数据帧/远程帧与其他帧。

1．数据帧

数据帧由七种不同的位段组成：帧起始、仲裁段、控制段、数据段、CRC 段、应答段和帧结束。数据段的长度可以为 0～8 个字节。

数据帧结构



1）帧起始（SOF）：帧起始（SOF）标志着数据帧和远程帧的起始，仅由一个“显性”位组成。在 CAN 总线的同步规则中，当总线空闲时（处于隐性状态），才允许站点开始发送信号。所有的站点必须同步于首先开始发送报文的站点的帧起始前沿（该方式称为“硬同步”）。

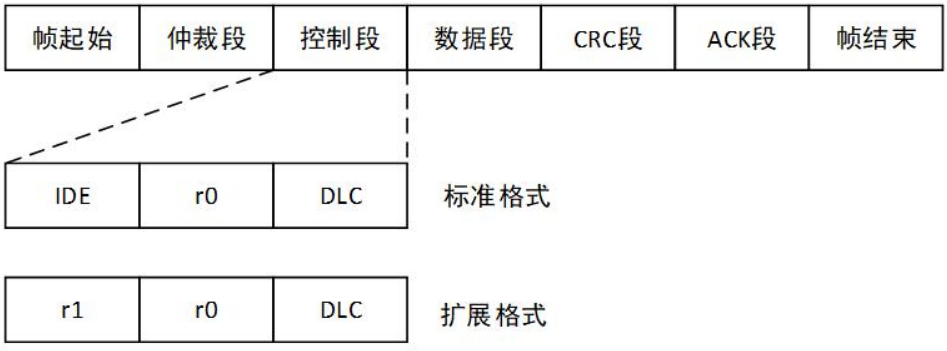
2）仲裁段：仲裁段由标识符和 RTR 位（远程帧发送标识位，占 1bit，显性）组成，标准帧格式与扩展帧格式的仲裁段格式不同。标准格式里，仲裁段由 11 位标识符和 RTR 位组成，标识符位为 ID0～ID10；扩展帧格式里，仲裁段包括 29 位标识符、SRR 位（替代远程请求，占 1bit，显性）、IDE 位（标志符扩展，占 1bit）、RTR 位。其标识符为 ID0～ID28。

为了区别标准帧格式和扩展帧格式，IDE 位为显性，表示数据帧为标准格式；IDE 位为隐性，表示数据帧为扩展帧格式。仲裁段传输顺序为从最高位到最低位，其中最高 7 位不能全为零。RTR 位在数据帧里必须为“显性”，而在远程帧里必须为“隐性”。它是区别数据帧和远程帧的标志。

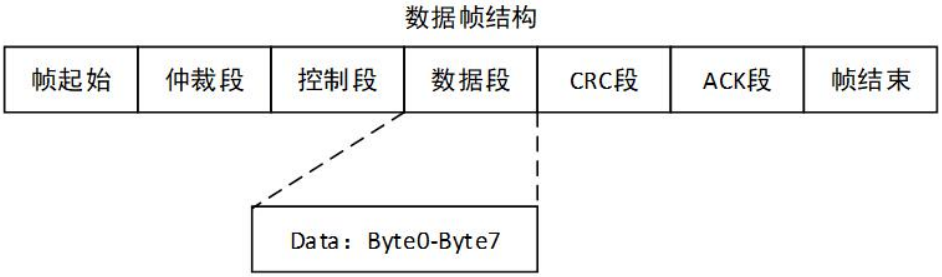
数据帧结构

3）控制段：控制段由 6 位组成，包括 2 个保留位（r0、r1 各占 1bit，总是用隐性电平填充）及 DLC（4 位数据长度码，占 4bit），允许的数据长度值为 0～8 字节。

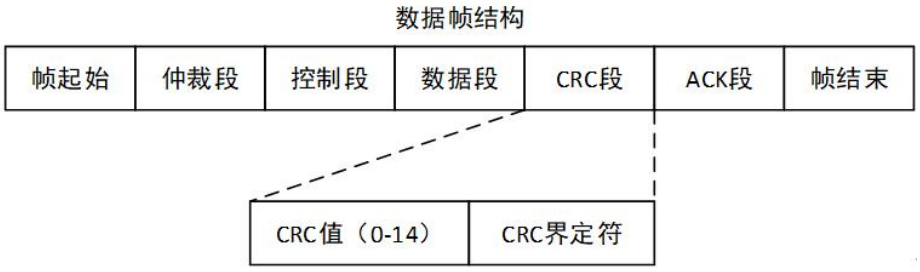
数据帧结构



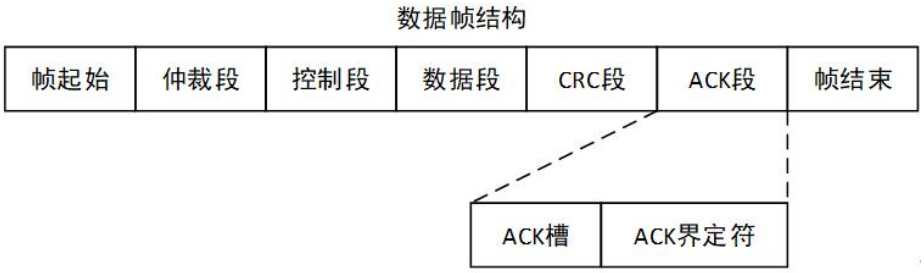
4）数据段：发送缓冲区中的数据按照长度代码指示长度发送。对于接收的数据，同样如此。一个数据帧传输的数据量可为 0～8 字节，每个字节包含 8 位，传输时首先发送的是 MSB（最高位）。



5）CRC 校验码段：它由 CRC 域（15 位）及 CRC 边界符（一个隐性位）组成。CRC 计算中，被除的多项式包括帧的起始段、仲裁段、控制段、数据段及 15 位为 0 的解除填充的位流给定。此多项式被下列多项式 $X^{15}+X^{14}+X^{10}+X^{8}+X^{7}+X^{4}+X^{3}+1$ 除（系数按模 2 计算），相除的余数即为发至总线的 CRC 序列。发送时，CRC 序列的最高有效位被首先发送/接收。之所以选用这种帧校验方式，是由于这种 CRC 校验码对于少于 127 位的帧是最佳的。



6）ACK 应答段：应答段由发送方发出的两个（应答间隙及应答界定）隐性位组成，所有接收到正确的 CRC 序列的节点将在发送节点的应答间隙上将发送的这一隐性位改写为显性位。因此，发送节点将一直监视总线信号已确认网络中至少一个节点正确地接收到所发信息。应答界定符是应答段中第二个隐性位，由此可见，应答间隙两边有两个隐性位：CRC 段和应答界定位。



7）帧结束：每一个数据帧或远程帧均由一串七个隐性位的帧结束段结尾。这样，接收节点可以正确检测到一个帧的传输结束。

2．远程帧

远程帧也有标准格式和扩展格式，而且都由 6 个不同的位段组成：帧起始、仲裁段、控制段、CRC 段、应答段、帧结尾。与数据帧相比，远程帧的 RTR 位为隐性，没有数据段，数据长度编码段可以是 0～8 个字节的任何值，这个值是远程帧请求发送的数据帧的数据段长度。当具有相同仲裁段的数据帧和远程帧同时发送时，由于数据帧的 RTR 位为显性，所以数据帧获得优先，发送远程帧的节点可以直接接收数据。

数据帧与远程帧的区别：

比较内容	数据帧	远程帧

ID	发送节点的 ID	被请求发送节点的 ID
SRR	显性电平（0）	隐性电平（1）
RTR	显性电平（0）	隐性电平（1）
DLC	发送数据长度	请求的数据长度
是否有数据段	是	否
CRC 校验范围	帧起始+仲裁段+控制段+数据段	帧起始+仲裁段+控制段

3．错误帧

错误帧由两个不同的段组成：第一个段是来自控制器的错误标志，第二个段为错误分界符。

1）错误标志：有两种形式的错误标志，可由其他 CAN 总线协议控制器的显性位改写。

激活（Active）错误标志。它由 6 个连续显性位组成。

认可（Passive）错误标志。它由 6 个连续隐性位组成。

2）错误界定：错误界定符由 8 个隐性位组成。传送了错误标志以后，每一站就发送一个隐性位，并一直监视总线直到检测出 1 个隐性位为止，然后就开始发送其余 7 个隐性位。

4．过载帧

过载帧由两个区域组成：过载标识域及过载界定符域。

下述三种状态将导致过载帧发送：

1）接收方在接收一帧之前需要过多的时间处理当前的数据（接收尚未准备好）；

2）在帧空隙域检测到显性位信号；

3）如果 CAN 节点在错误界定符或过载界定符的第 8 位采样到一个显性位节点会发送一个过载帧。

附录Ⅵ 控制器与驱动器接线参考

控制器 DB9 母头接口

1．ZMC316 脉冲轴接口与松下 A5 伺服驱动器低速差分接线

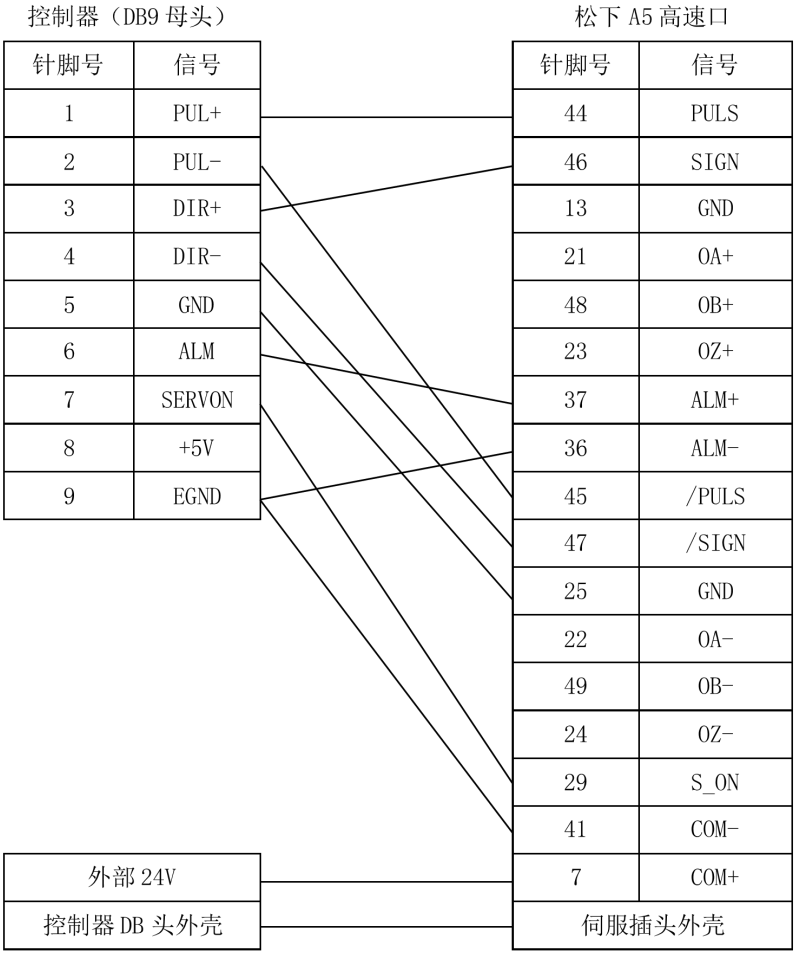
控制器（DB9 母头）

针脚号	信号
1	PUL+
2	PUL-
3	DIR+
4	DIR-
5	GND
6	ALM
7	SERVON
8	+5V
9	EGND
外部 24V	
控制器 DB 头外壳	

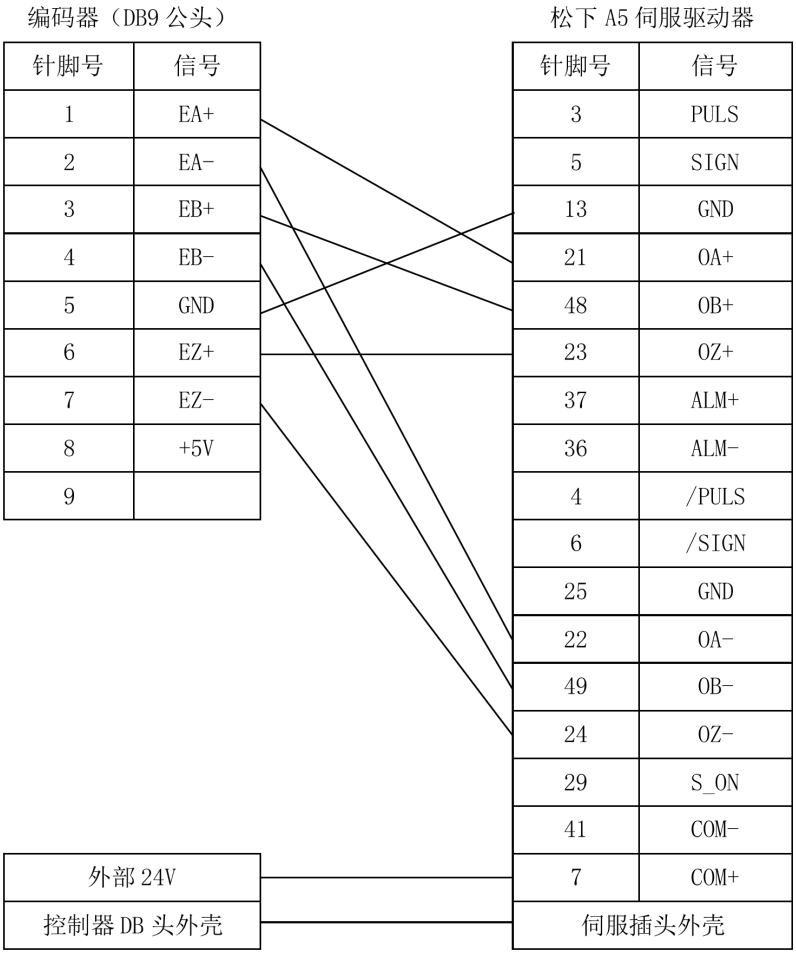
松下 A5 伺服驱动器

针脚号	信号
3	PULS
5	SIGN
13	GND
21	OA+
48	OB+
23	OZ+
37	ALM+
36	ALM-
4	/PULS
6	/SIGN
25	GND
22	OA-
49	OB-
24	OZ-
29	S_ON
41	COM-
7	COM+
伺服插头外壳	

2．ZMC316 脉冲轴接口与松下 A5 伺服驱动器高速差分接线

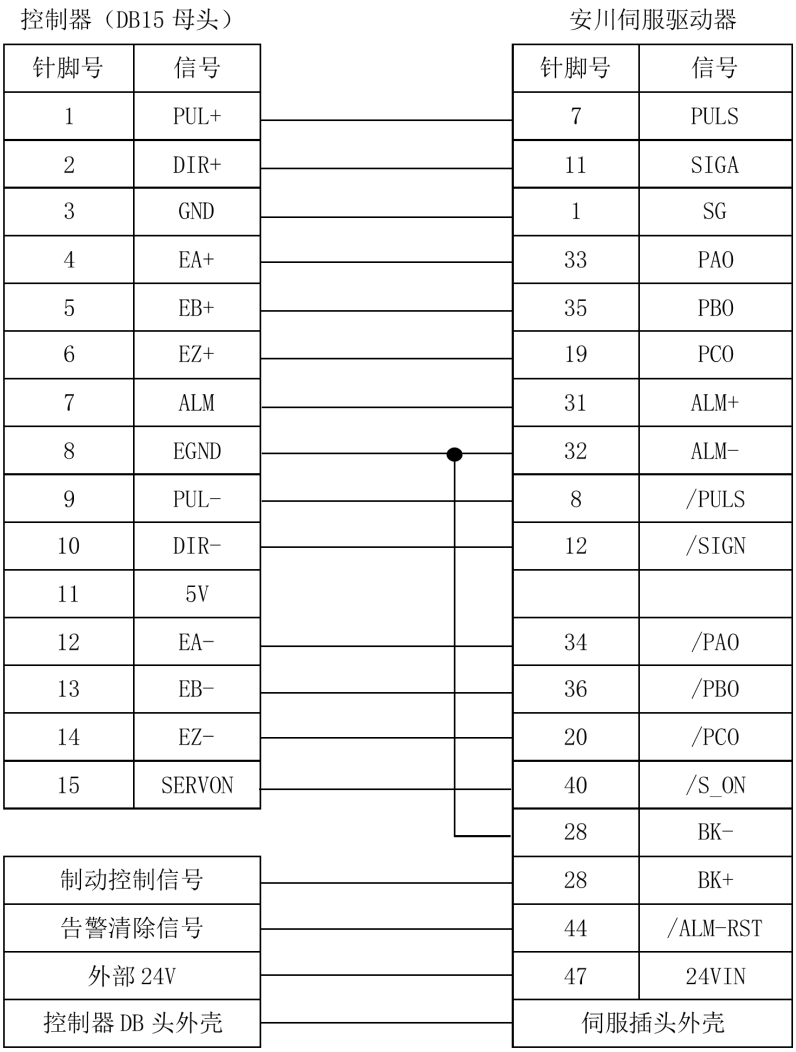


3．ZMC316 编码器接口与松下 A5 伺服驱动器接线



控制器 DB15 母头接口

1．ZMC204 与安川伺服驱动器接线



2．ZMC204 与松下 A5 伺服驱动器低速差分脉冲口接线

控制器（DB15 母头）

针脚号	信号
1	PUL+
2	DIR+
3	GND
4	EA+
5	EB+
6	EZ+
7	ALM
8	EGND
9	PUL-
10	DIR-
11	5V
12	EA-
13	EB-
14	EZ-
15	SERVON

外部 24V
控制器 DB 头外壳

松下 A5 伺服驱动器

针脚号	信号
3	PULS
5	SIGN
13	GND
21	OA+
48	OB+
23	OZ+
37	ALM+
36	ALM-
4	/PULS
6	/SIGN
25	GND
22	OA-
49	OB-
24	OZ-
29	S_ON
41	COM-
7	COM+
伺服插头外壳	

3．ZMC204 与松下 A5 伺服驱动器高速差分脉冲口接线

控制器（DB15 母头）

针脚号	信号
1	PUL+
2	DIR+
3	GND
4	EA+
5	EB+
6	EZ+
7	ALM
8	EGND
9	PUL-
10	DIR-
11	5V
12	EA-
13	EB-
14	EZ-
15	SERVON

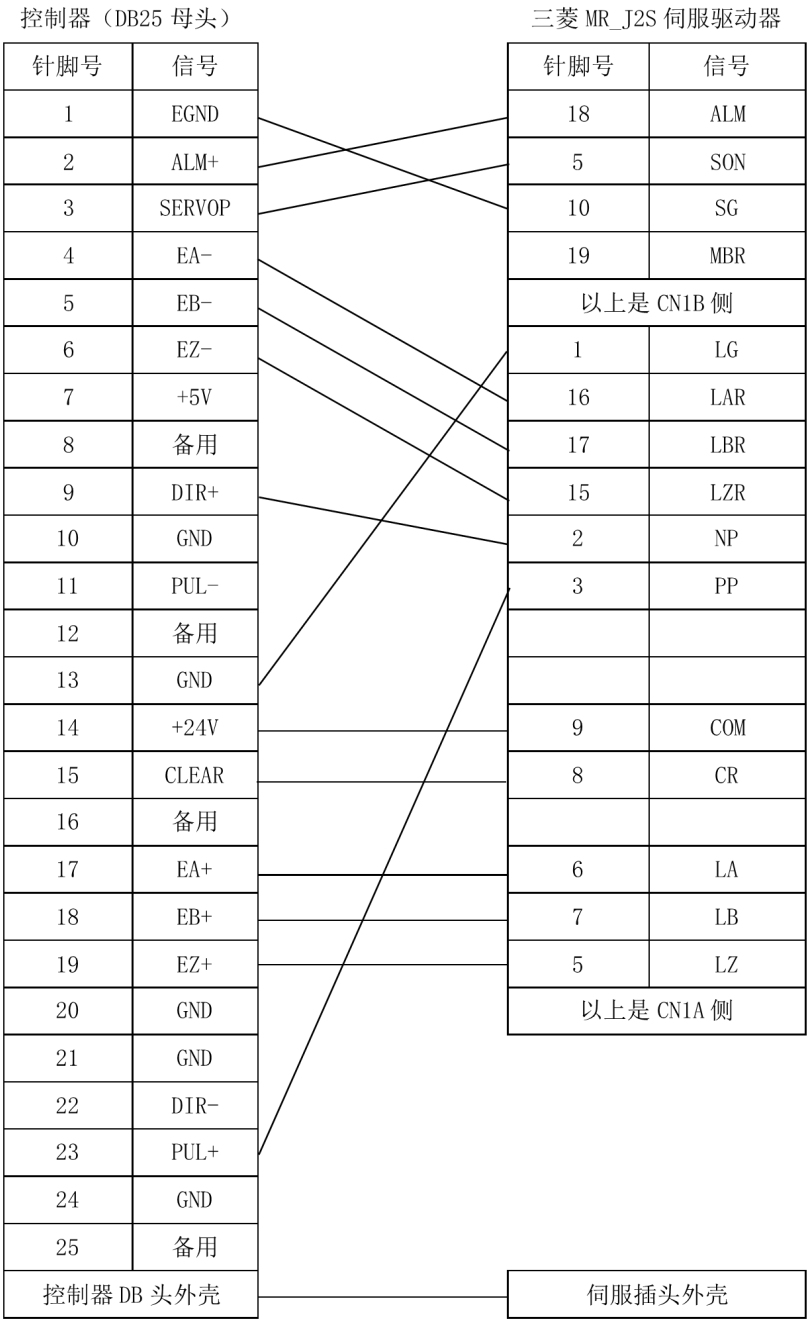
外部 24V
控制器 DB 头外壳

松下 A5 伺服高速口

针脚号	信号
44	PULS
46	SIGN
13	GND
21	OA+
48	OB+
23	OZ+
37	ALM+
36	ALM-
45	/PULS
47	/SIGN
25	GND
22	OA-
49	OB-
24	OZ-
29	S_ON
41	COM-
7	COM+
伺服插头外壳	

控制器 DB25 母头接口

1．ZMC308 与三菱 MR_J2S 伺服驱动器接线



针脚号	信号
1	EGND
2	ALM+
3	SERVOP
4	EA-
5	EB-
6	EZ-
7	+5V
8	备用
9	DIR+
10	GND
11	PUL-
12	备用
13	GND
14	+24V
15	CLEAR
16	备用
17	EA+
18	EB+
19	EZ+
20	GND
21	GND
22	DIR-
23	PUL+
24	GND
25	备用
控制器 DB 头外壳	

针脚号	信号
18	ALM
5	SON
10	SG
19	MBR
以上是 CN1B 侧	
1	LG
16	LAR
17	LBR
15	LZR
2	NP
3	PP
9	COM
8	CR
6	LA
7	LB
5	LZ
以上是 CN1A 侧	
伺服插头外壳	

2．ZMC308 与松下 A5 伺服驱动器低速差分接线

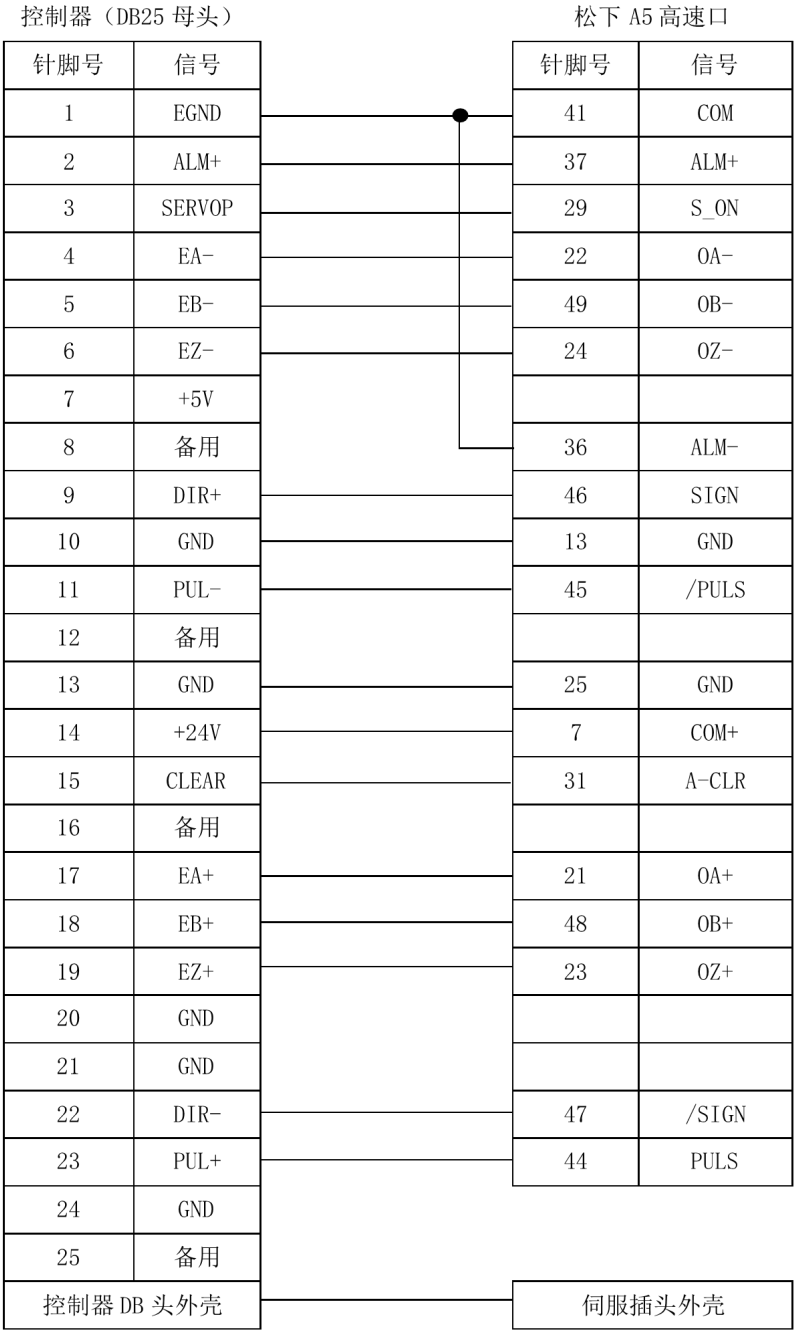
控制器（DB25 母头）

针脚号	信号
1	EGND
2	ALM+
3	SERVOP
4	EA-
5	EB-
6	EZ-
7	+5V
8	备用
9	DIR+
10	GND
11	PUL-
12	备用
13	GND
14	+24V
15	CLEAR
16	备用
17	EA+
18	EB+
19	EZ+
20	GND
21	GND
22	DIR-
23	PUL+
24	GND
25	备用
控制器 DB 头外壳	

松下 A5 伺服驱动器

针脚号	信号
41	COM
37	ALM+
29	S_ON
22	OA-
49	OB-
24	OZ-
36	ALM-
5	SIGN
13	GND
4	/PULS
25	GND
7	COM+
31	A-CLR
21	OA+
48	OB+
23	OZ+
6	/SIGN
3	PULS
伺服插头外壳	

3．ZMC308 与松下 A5 伺服驱动器高速差分接线

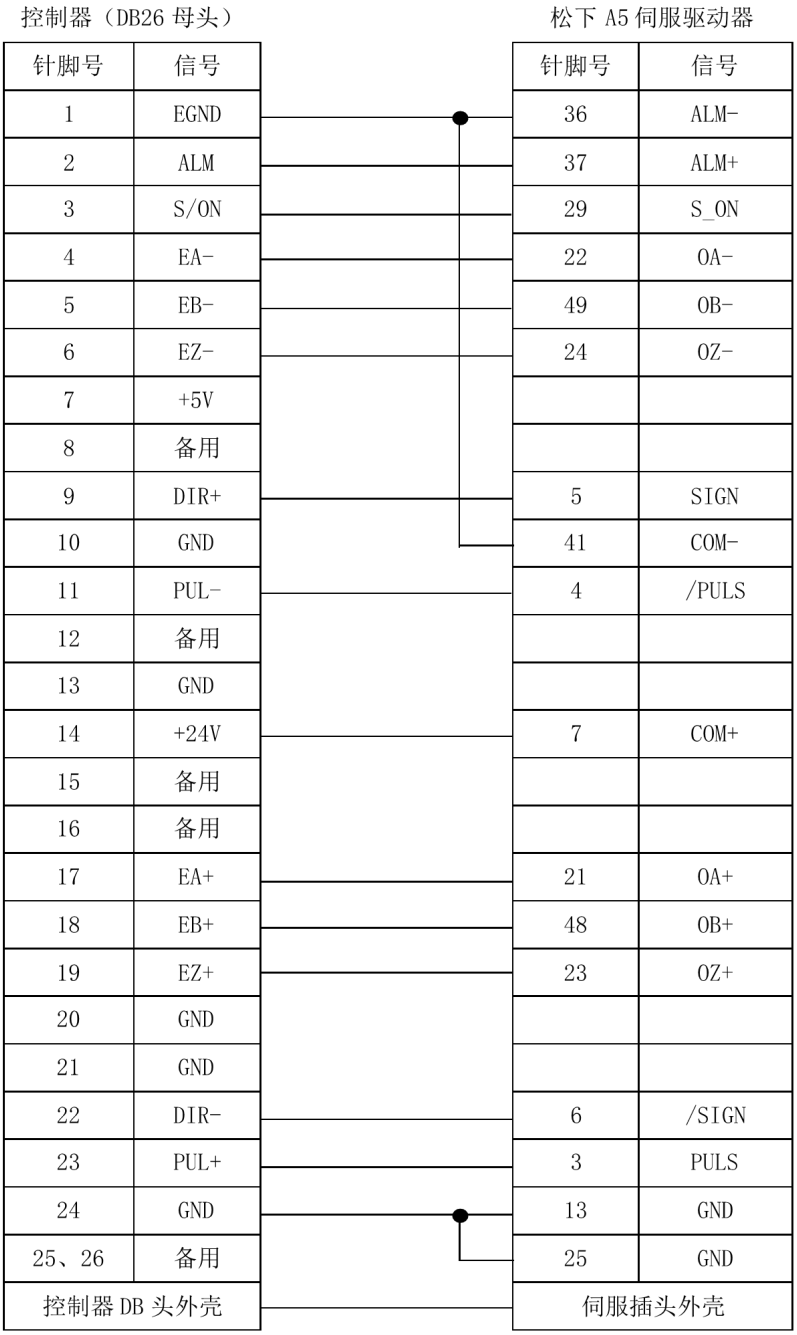


针脚号	信号
1	EGND
2	ALM+
3	SERVOP
4	EA-
5	EB-
6	EZ-
7	+5V
8	备用
9	DIR+
10	GND
11	PUL-
12	备用
13	GND
14	+24V
15	CLEAR
16	备用
17	EA+
18	EB+
19	EZ+
20	GND
21	GND
22	DIR-
23	PUL+
24	GND
25	备用
控制器 DB 头外壳	

针脚号	信号
41	COM
37	ALM+
29	S_ON
22	OA-
49	OB-
24	OZ-
36	ALM-
46	SIGN
13	GND
45	/PULS
25	GND
7	COM+
31	A-CLR
21	OA+
48	OB+
23	OZ+
47	/SIGN
44	PULS
伺服插头外壳	

控制器 DB26 母头接口

1．ZMC460N 与松下 A5 伺服驱动器低速差分接线



控制器（DB26 母头）		松下 A5 伺服驱动器	
针脚号	信号	针脚号	信号
1	EGND	36	ALM-
2	ALM	37	ALM+
3	S/ON	29	S_ON
4	EA-	22	OA-
5	EB-	49	OB-
6	EZ-	24	OZ-
7	+5V		
8	备用		
9	DIR+	5	SIGN
10	GND	41	COM-
11	PUL-	4	/PULS
12	备用		
13	GND		
14	+24V	7	COM+
15	备用		
16	备用		
17	EA+	21	OA+
18	EB+	48	OB+
19	EZ+	23	OZ+
20	GND		
21	GND		
22	DIR-	6	/SIGN
23	PUL+	3	PULS
24	GND	13	GND
25、26	备用	25	GND
控制器 DB 头外壳		伺服插头外壳	

2．ZMC420N 与松下 A5 伺服驱动器高速差分接线

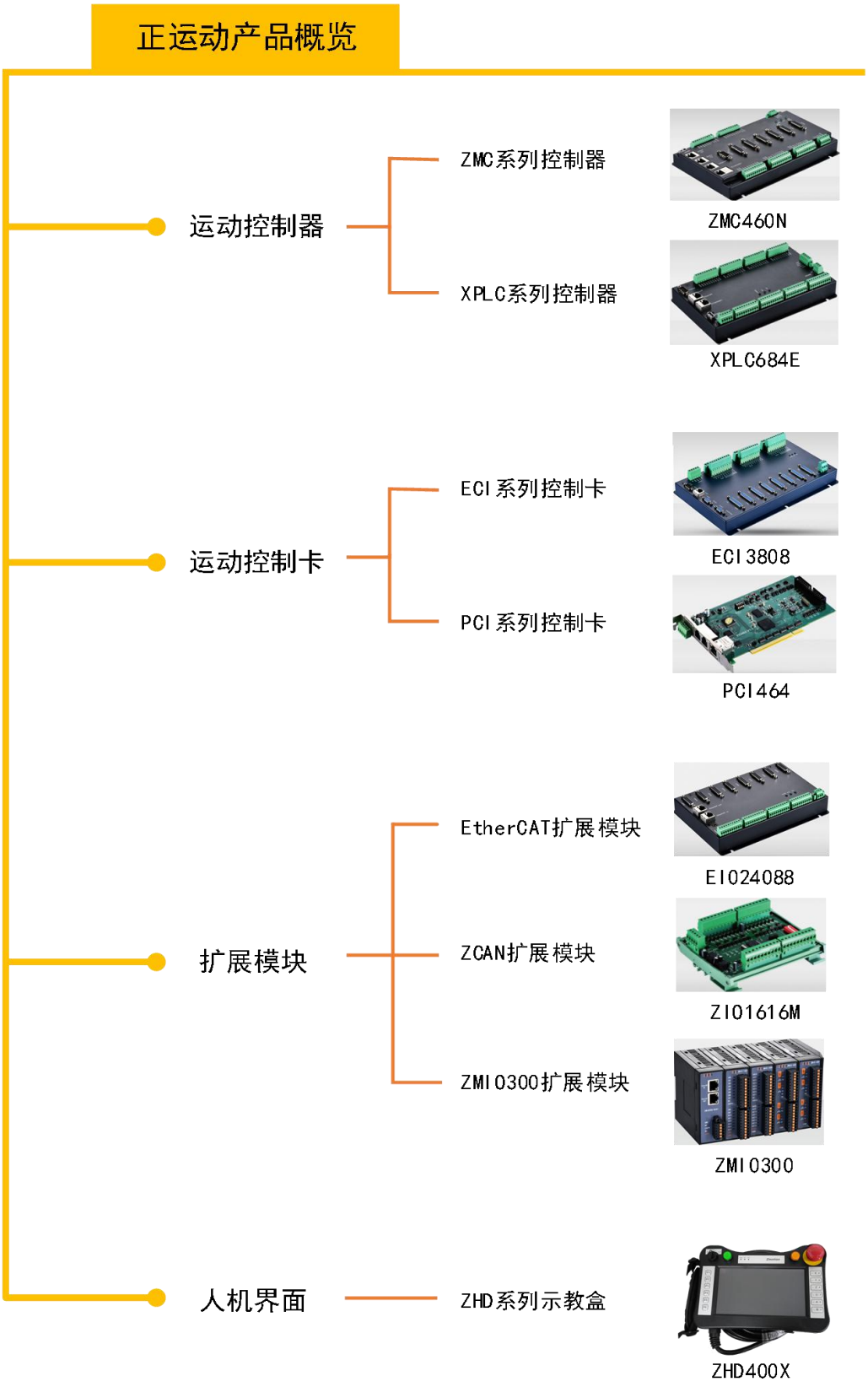
控制器（DB26 母头）

针脚号	信号
1	EGND
2	ALM
3	S/ON
4	EA-
5	EB-
6	EZ-
7	+5V
8	备用
9	DIR+
10	GND
11	PUL-
12	备用
13	GND
14	+24V
15	备用
16	备用
17	EA+
18	EB+
19	EZ+
20	GND
21	GND
22	DIR-
23	PUL+
24	GND
25、26	备用
控制器 DB 头外壳	

松下 A5 高速口

针脚号	信号
36	ALM-
37	ALM+
29	S_ON
22	OA-
49	OB-
24	OZ-
46	SIGN
41	COM-
45	/PULS
7	COM+
21	OA+
48	OB+
23	OZ+
47	/SIGN
44	PULS
13	GND
25	GND
伺服插头外壳	

附录Ⅶ 选型指南


正运动产品概览
运动控制器
ZMC系列控制器
ZMC460N
XPLC系列控制器
XPLC684E
运动控制卡
ECI系列控制卡
ECI3808
PCI系列控制卡
PCI464
扩展模块
EtherCAT扩展模块
EI024088
ZCAN扩展模块
ZIO1616M
ZMIO300扩展模块
ZMIO300
人机界面
ZHD系列示教盒
ZHD400X

ZMC 系列控制器

ZMC0 系列

ZMC0 系列经济型多轴运动控制器是脉冲型的独立式运动控制器。控制器本身最多支持 5 轴，可扩展至 6 轴的运动控制，用以实现直线插补、圆弧插补、螺旋插补等简单的轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、流水线等 6 轴以内的脉冲应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	ECAT 口	网口	U 盘
ZMC002	2	2	4	16/8	-	-	2	-	-	-	-
ZMC004	4	2	4	16/8	-	-	2	-	-	-	-
ZMC004H	4	2	6	16/8	-	-	2	-	-	-	-
ZMC004WEA	4	1	4	16/16	2	2	2	1	1	-	-
ZMC004BEA	4	1	4	16/16	2	2	2	1	1	-	-
ZMC005	5	1	6	24+5/8+5	-	2	2	-	-	-	1

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸 (mm)	功能描述
ZMC002	5M	16	124K	3	21	183*116	点位、直线、圆弧、凸轮、连续插补
ZMC004	5M	16	124K	3	21	183*116	
ZMC004H	5M	32	256K	6	1024	183*116	
ZMC004WEA	500K（单端）	32	380K	6	1024	177*122	
ZMC004BEA	100K（单端）	16	380K	6	32	177*122	
ZMC005	5M	64	300K	6	21	226*127	

ZMC1 系列

ZMC1 系列经济型多轴运动控制器是脉冲型的独立式运动控制器。控制器本身最多支持 6 轴，可扩展至 12 轴的运动控制，用以实现直线插补、任意圆弧插补（包括平面圆弧和空间圆弧）、螺旋插补、椭圆插补等较复杂的轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、非标设备、流水线等 4-12 轴脉冲应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	422	485	网 口	U 盘
ZMC104S	4	2	12	24/8	-	-	1	-	1	1	1
ZMC106S	6	2	12	24/8	-	-	1	-	1	1	1

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ZMC104S	8M	128	360K	6	1024	226*127	点位、直线、圆弧、凸轮、连续插补
ZMC106S	8M	128	360K	6	1024	226*127	

ZMC2 系列

ZMC2 系列经济型多轴运动控制器可分为脉冲型和总线型（部分型号支持 EtherCAT 总线）。可扩展至 12 轴的运动控制，用以实现直线插补、任意圆弧插补（包括平面圆弧和空间圆弧）、螺旋插补、椭圆插补等较复杂的轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、非标设备、印刷包装设备、纺织服装设备、医疗设备、流水线等多种应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	ECAT 口	网口	U 盘
ZMC204	4	4	12	24+4/8+4	-	-	1	1	-	1	1
ZMC206	6	6	12	24+6/8+6	-	-	1	1	-	1	1
ZMC204E	4	1	12	24+1/8+1	2	2	1	1	1	1	1
ZMC212	12	2	16	24/8	2	2	1	1	-	1	1

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ZMC204	10M	128	460K	6	1024	252*133	点位、直线、圆弧、凸轮、连续插补、机械手指令
ZMC206	10M	128	460K	6	1024	252*133	
ZMC204E	10M	128	460K	12	1024	205*122	
ZMC212	10M	128	460K	13	1024	280*127	

ZMC3 系列

ZMC3 系列高性能多轴运动控制器可分为脉冲型和总线型(部分型号支持 EtherCAT 总线、RTEX 总线)。脉冲型控制器本身最多支持 16 轴，最多可扩展至 24 轴。总线型控制器本身最多支持 6 轴，可扩展至 12 轴的运动控制。均用以实现直线插补、任意圆弧插补（包括平面圆弧和空间圆弧）、螺旋插补、椭圆插补等较复杂的轨迹控制需求。

可用于机器人（SCARA、Delta、6 关节）、电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、非标设备、印刷包装设备、纺织服装设备、舞台娱乐设备、医疗设备、流水线等多种应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	422	485	ECAT 口	RTEX 口
ZMC303	3	3+1	10	24+3/8+3	2	2	1	1	1	-	-
ZMC304	4	4+1	10	24+4/8+4	2	2	1	1	1	-	-
ZMC304X	4	4	12	24+4/12+4	2	2	1	1	1	-	-
ZMC306X	6	6	12	24+6/12+6	2	2	1	1	1	-	-
ZMC306E	6	2	12	24+2/16+2	2	2	1	-	1	1	-
ZMC306N	6	2	12	24+2/16+2	2	2	1	-	1	1	1
ZMC306	6	6	16	40+6/16+12	4	2	1	1	1	-	-
ZMC308	8	8	16	40+8/16+16	4	2	1	1	1	-	-
ZMC316	15+1	3	24	24+15/8+15	2	2	1	1	1	-	-

续表

产品型号	网口	U 盘	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ZMC303	1	1	10M	128	300K	6	1024	205*134	点位、直线、圆弧、凸轮、连续插补、机械手指令
ZMC304	1	1	10M	128	300K	6	1024	205*134	
ZMC304X	1	1	10M	128	2000K	12	1024	205*134	
ZMC306X	1	1	10M	128	2000K	12	1024	205*134	
ZMC306E	1	1	10M	256	6144K	10	1024	205*135	
ZMC306N	1	1	10M	256	6144K	10	1024	205*135	
ZMC306	1	1	10M	128	2000K	13	1024	292*188	
ZMC308	1	1	10M	128	2000K	13	1024	292*188	
ZMC316	1	1	10M	128	2000K	16	1024	202*180	

ZMC4 系列

ZMC4 系列高性能多轴运动控制器可分为脉冲型和总线型(部分型号支持 EtherCAT 总线、RTEX 总线)。脉冲型控制器本身最多支持 12 轴，最多可扩展至 32 轴。总线型控制器本身最多支持 64 轴 EtherCAT 总线或 60 轴 RTEX 总线。均用以实现直线插补、任意圆弧插补（包括平面圆弧和空间圆弧）、螺旋插补、椭圆插补等较复杂的轨迹控制需求。

可用于机器人（SCARA、Delta、6 关节）、电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、激光加工设备、非标设备、印刷包装设备、纺织服装设备、舞台娱乐设备、医疗设备、流水线等多种应用场合。

产品型号	电机轴数	编码器数	振镜轴	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	ECAT 口	RTEX 口
ZMC406	6	6	-	32	24+6/12+6	-	2	1	1	1	-
ZMC412	12	12	-	32	24+12/12+12	-	2	1	2	-	-
ZMC432	32	6	-	32	24+6/12+6	-	2	1	1	1	-
ZMC432N	32	6	-	38	24+6/12+6	-	2	1	1	-	1
ZMC464	64	3	-	64	24+3/12+3	-	2	1	1	1	-
ZMC464R	64	3	-	64	24+3/12+3	-	2	1	1	1	-
ZMC420scan	20	8	4	20	24+4/12+4	4	2	1	1	1	1
ZMC430N	30	6+6	-	60	24+6/12+6	4	2	1	1	1	1
ZMC460N	60	6+6	-	60	24+6/12+6	2	2	1	1	1	1

续表

产品型号	网口	U 盘	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ZMC406	1	1	10M	4096	32M	22	8000	205*136	点位、直线、圆弧、凸轮、连续插补、机械手指令
ZMC412	1	1	10M	4096	32M	22	8000	250*164	
ZMC432	1	1	10M	4096	32M	22	8000	205*136	
ZMC432N	1	1	10M	4096	32M	22	8000	216*146	
ZMC464	1	1	10M	4096	32M	22	8000	205*136	
ZMC464R	1	1	10M	4096	32M	22	8000	205*136	
ZMC420scan	1	1	10M	4096	32M	22	8000	216*144	
ZMC430N	1	1	10M	4096	32M	22	8000	216*143	
ZMC460N	1	1	10M	4096	32M	22	8000	216*143	

XPLC 系列控制器

XPLC 系列经济型多轴运动控制器是兼容 EtherCAT 总线和脉冲型可支持梯形图编程的独立式运动控制器，控制器本身最多支持 8 轴 EtherCAT 总线，可扩展至 16 轴的运动控制，用以实现复杂轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、非标设备、印刷包装设备、纺织服装设备、医疗设备、流水线等多种应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	ECAT 口	网口	U 盘
XPLC004E	4	-	8	16/16	-	2	1	1	1	1	1
XPLC006E	6	-	8	16/16	-	2	1	2	1	1	1
XPLC664E	6	-	12	32/32	2	2	1	1	1	1	1
XPLC864E	8	-	12	32/32	2	2	1	1	1	1	1

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
XPLC004E	-	32	6144K	6	1024	160*114.5	点位、直线、圆弧、凸轮
XPLC006E	-	32	6144K	6	1024	160*114.5	
XPLC664E	500K（单端）	128	2M	10	1024	219*135	
XPLC864E	500K（单端）	128	2M	10	1024	219*135	

ECI 系列控制卡

ECI0032/0064 系列

网络型 IO 控制卡，支持 IO 及 IO 扩展，最多可同时扩展 256 个输入和 256 个输出。模拟量最多可扩展至 128 路 AD 和 64 路 DA。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	422	网口	U 盘
ECI0032	0	0	0	16/16	-	-	1	-	-	1	-
ECI0032B	0	0	0	16/16	-	-	1	-	-	1	-
ECI0064	0	0	0	32/32	-	-	1	-	-	1	-
ECI0064B	0	0	0	32/32	-	-	1	-	-	1	-

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ECI0032	-	-	3K	1	-	150*114	16 进 16 出 (带过流保护)
ECI0032B	-	-	3K	1	-	150*114	16 进 16 出 (带过流保护) 可脱机运行
ECI0064	-	-	3K	1	-	192*129	32 进 32 出 (带过流保护)
ECI0064B	-	-	3K	1	-	192*129	32 进 32 出 (带过流保护) 可脱机运行

ECI1000 系列

ECI1000 系列经济型多轴运动控制卡是脉冲型+网络型的运动控制卡，控制卡本身最多支持 4 轴，可扩展至 6 轴，用以实现直线插补、圆弧插补、螺旋插补等简单的轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、流水线等 6 轴以内脉冲应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	422	网口	U 盘
ECI1300	3	1(24V)	6	36/12	-	-	1	-	-	1	-
ECI1308	3	1(24V)	6	36/12	-	-	1	-	-	1	-
ECI1400	4	1(24V)	6	36/12	-	-	1	-	-	1	-
ECI1408	4	1(24V)	6	36/12	-	-	1	-	-	1	-

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ECI1300	5M	128	2K	1	-	205*138	点位、凸轮
ECI1308	5M	128	2K	1	-	205*138	点位、直线、圆弧、凸轮、连续插补
ECI1400	5M	128	2K	1	-	205*138	点位、凸轮
ECI1408	5M	128	2K	1	-	205*138	点位、直线、圆弧、凸轮、连续插补

ECI2000 系列

ECI2000 系列经济型多轴运动控制卡是脉冲型、模块化的网络型的运动控制卡，控制卡本身最多支持 6 轴，可扩展至 12 轴，用以实现简单的轨迹控制需求。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、流水线等 12 轴以内脉冲应用场合。

产品型号	电机轴数	编码器数	专用手轮接口	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	422	网口	U 盘
ECI2400	4	4	-	12	24+4/8+4	-	-	1	-	-	1	-
ECI2408	4	4	-	12	24+4/8+4	-	-	1	-	-	1	-
ECI2600	6	6	-	12	24+6/8+6	-	-	1	-	-	1	-
ECI2608	6	6	-	12	24+6/8+6	-	-	1	-	-	1	-
ECI2410	4	4	1	12	24+8/16+4	2	2	1	-	-	1	-
ECI2418	4	4	1	12	24+8/16+4	2	2	1	-	-	1	-

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ECI2400	10M	128	4K	1	-	201*134	点位、凸轮
ECI2408	10M	128	4K	1	-	201*134	点位、直线、圆弧、凸轮、连续插补、机械手指令
ECI2600	10M	128	4K	1	-	201*134	点位、凸轮
ECI2608	10M	128	2K	1	-	201*134	点位、直线、圆弧、凸轮、连续插补、机械手指令
ECI2410	10M	128	4K	1	-	220*139	点位、凸轮
ECI2418	10M	128	2K	1	-	220*139	点位、直线、圆弧、凸轮、连续插补、机械手指令

ECI3000 系列

ECI3000 系列经济型多轴运动控制卡是脉冲型+网络型的运动控制卡，控制卡本身最多支持 8 轴，可扩展至 10 轴，用以实现直线插补、圆弧插补、螺旋插补等简单的轨迹控制需求。

支持 IO 及 IO 扩展，最多可同时扩展 256 个输入和 256 个输出。模拟量最多可扩展至 128 路 AD 和 64 路 DA。

可用于电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、流水线等 10 轴以内脉冲应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	232	485	422	网口	U 盘
ECI3600	6	6+1	12	40+6/16+12	4	2	1	-	-	1	-
ECI3608	6	6+1	12	40+6/16+12	4	2	1	-	-	1	-
ECI3800	8	8+1	12	40+8/16+16	4	2	1	-	-	1	-
ECI3808	8	8+1	12	40+8/16+16	4	2	1	-	-	1	-

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述
ECI3600	10M	128	4K	1	-	292*188	点位、凸轮
ECI3608	10M	128	4K	1	-	292*188	点位、直线、圆弧、凸轮、连续插补、机械手指令
ECI3800	10M	128	4K	1	-	292*188	点位、凸轮
ECI3808	10M	128	2K	1	-	292*188	点位、直线、圆弧、凸轮、连续插补、机械手指令

PCI 总线型运动控制卡

PCI 系列高性能多轴运动控制卡是总线型的运动控制卡，控制卡本身最多支持 64 轴的运动控制，用以实现直线插补、圆弧插补、螺旋插补、椭圆插补等复杂的轨迹控制需求。

可用于机器人（SCARA、Delta、6 关节）、电子半导体设备（如检测类设备、组装类设备、焊锡机）、点胶设备、激光加工设备、非标设备、印刷包装设备、纺织服装设备、舞台娱乐设备、医疗设备、流水线等多种应用场合。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	内部 AD	内部 DA	ECAT 口	RTEX 口	网口	U 盘
PCI104E	4	1	12	8/8	-	-	1	-	1	1
PCI106E	6	1	12	8/8	-	-	1	-	1	1
PCI464-16	16	1	64	8/8	-	-	1	1	1	1
PCI464-32	32	1	64	8/8	-	-	1	1	1	1
PCI464	64	1	64	8/8	1	-	1	1	1	1

续表

产品型号	脉冲频率	轴运动缓冲数	程序空间	任务数	掉电存储	尺寸(mm)	功能描述	可选配件
PCI104E	10M	128	460K	6	1024	205*134	点位、直线、圆弧、凸轮、连续插补、机械手指令	转接线(ZP72-02) 屏蔽电缆(DB37-150) 接线板(EXDB37M-37)
PCI106E	10M	128	460K	6	1024	205*134		
PCI464-16	10M	512	1920K	22	8000	205*134		
PCI464-32	10M	512	1920K	22	8000	205*134		
PCI464	10M	512	1920K	22	8000	205*135		

扩展模块

EtherCAT 扩展模块

可支持多个脉冲轴及 IO 的远程扩展，最多支持 8 路脉冲轴扩展。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	ECAT IN/OUT	尺寸(mm)	功能描述
EIO1616	-	-	-	16/16	1/1	143*107	可选配标准模组尺型号为 EIO1616M
EIO24088	8	8	8	24/8	1/1	210*147	8 轴脉冲轴扩展

ZCAN 扩展模块

可支持最多 2 个脉冲轴扩展及 IO 的远程扩展。

产品型号	电机轴数	编码器数	总轴数	内部输入/输出口	AD	DA	尺寸(mm)	功能描述
ZIO0808	-	-	-	8/8	-	-	98*72	可选配标准模组尺型号为 ZIO0808M
ZIO0016	-	-	-	0/16	-	-	98*72	可选配标准模组尺型号为 ZIO0016M
ZIO1608	-	-	-	16/8	-	-	126*99	可选配标准模组尺型号为 ZIO1608M
ZIO1616	-	-	-	16/16	-	-	142*107	可选配标准模组尺型号为 ZIO1616M
ZIO1632	-	-	-	16/32	-	-	192*107	可选配标准模组尺型号为 ZIO1632M
ZAIO0802	-	-	-	-	8	2	120*72	可选配标准模组尺型号为 ZIO0802M
ZIO16082	2	2	2	16/8	-	-	126*106	可选配标准模组尺型号为 ZIO16082M

ZMIO300 扩展模块

立式 EtherCAT 总线扩展模块、可支持 IO 及模拟量，单个耦合器最多支持 16 个扩展子模块。

产品型号	数字量输入	数字量输出	输入模拟量	输出模拟量	ECAT IN/OUT	尺寸(mm)	功能描述
ZMIO300-ECAT	-	-	-	-	1/1	108*32*95	ECAT 通讯模块
ZMIO300-16DI	16	-	-	-	-	108*32*95	输入模块(NPN PNP)
ZMIO300-16DO	-	16	-	-	-	108*32*95	输出模块(NPN)
ZMIO300-16DOP	-	16	-	-	-	108*32*95	输出模块(PNP)
ZMIO300-4AD	-	-	4	-	-	108*32*95	AD 模块(16bit)
ZMIO300-4DA	-	-	-	4	-	108*32*95	DA 模块(16bit)

人机界面

ZHD 系列是一款开放式的带触摸屏功能的可编程示教盒，通过 ZDevelop 软件和 ZBasic 语法来开发界面程序、可以在线跟踪调试，均带有急停按钮。

ZHD300/ZHD400 可适配各种控制器；ZHD300X/ZHD400X 只能匹配支持 ZHMI 功能的控制器，且控制器软件开发需要 ZDevelop2.70 以上版本。

产品型号	分辨率	尺寸(mm)	按键数	支持协议	功能描述
ZHD300	480*272	280*131	47	MODBUS 自定义协议	中英文字符，直线，圆弧
ZHD400	800*480	230*165	18		按键和触摸屏配合使用
ZHD300X	480*272	280*131	47	ZMC 组态协议	按键和触摸屏配合使用
ZHD400X	800*480	230*165	18		按键和触摸屏配合使用